DIGITAL COMPUTER DESIGN

Logic, Circuitry, and Synthesis

EDWARD L. BRAUN

Aerospace Corporation
Los Angeles, California

1963

ACADEMIC PRESS • NEW YORK AND LONDON

ACADEMIC PRESS INC.
111 Fifth Avenue, New York, New York 10003

United Kingdom Edition published by
ACADEMIC PRESS INC. (LONDON) LTD.
Berkeley Square House, London W.1

LIBRARY OF CONGRESS CATALOG CARD NUMBER: 63-15725

First Printing, 1963
Second Printing, 1967

PRINTED IN THE UNITED STATES OF AMERICA

To my son Geffrey and my parents

Preface

This book provides an introductory treatment of the logical structure, electronic realization, and application of digital information processors. The extent of coverage of each major topic should also make the book useful as a review and reference text for persons experienced in the field. Each major chapter is a relatively self-contained unit in an important area: Boolean algebra for switching networks (Chapter 3), electronic building blocks for switching circuits (Chapter 4), memories for digital computers (Chapter 5), arithmetic operations in digital computers (Chapter 6), system design of GP (integral transfer) computers (Chapter 7), an extensive description of DDA (incremental transfer) computers (Chapter 8), and detection and correction of errors (Chapter 9), and input-output equipment (the Appendix).

With the exception of Chapters 4 and 5, the presentation is on a functional level, i.e., in terms of how elements with defined input-output characteristics may be organized to synthesize subsystems or systems with specified functional capabilities. Although functional descriptions and circuit problems cannot be separated completely, the discussion of detailed problems in the electronic realization of computers has been confined mainly to these two chapters. This was done for a number of reasons—first of all, in order not to obscure (by the intricacies and details of practical means of mechanization) the conceptual simplicity of fundamental principles treated in other chapters. This separation and the way material is organized in each chapter also facilitates looking-up particular topics. Also, while the entire field is evolving rapidly, developments in circuit techniques have advanced more rapidly than in logical design. When the writing of this book was begun, the clock-repetition rate of most digital computers was about 100 kc. Currently (1962) computer circuits are under development for operation in the microwave region of several hundred megacycles. (Developments in microwave, tunnel diode, and superconductive circuits are described in Chapters 4 and 5 and also referenced in the bibliographies of these chapters.)

The importance of digital information processing technology in the betterment of human welfare, in government, commerce, industry, science,

engineering, and military systems (as well as dangers inherent in misuse) makes it desirable that certain principles and the breadth of applications be widely appreciated. The text's emphasis on the functional approach makes much of the subject accessible to those with limited technical backgrounds. The presentation of material is designed to supplement instruction in university level classes and also to facilitate independent study. Because problems associated with various types of electronic circuits are largely confined to two chapters, the intelligent reader with limited knowledge in electrical and electronic circuits can (in accordance with his capability and inclinations) skim these chapters and still understand and benefit from the remainder of the text. (An additional reservation is that appreciation of all of Chapter 8 calls for a basic knowledge of ordinary differential equations.) The entire text can be understood by one having a background equivalent to a Bachelor's degree in electrical engineering with mathematics through differential equations.

Even after thorough study of branches of a subject, one may still have doubts on how to apply this knowledge in the synthesis of a particular design. Often this situation can be alleviated by the study of examples that illustrate in detail the application of the basic material. In the present case an effort was made to integrate material presented in earlier chapters by presenting (in Chapter 7) a detailed explanation of two simple digital computer logical designs.

The author is indebted to many organizations and individuals who have advanced the digital computer field and whose work forms the reservoir from which the material for this book was drawn. Many of these sources are listed in the bibliographies. Reports and publications of the Massachusetts Institute of Technology, John von Neumann and his colleagues at the Princeton Institute for Advanced Study, the University of Illinois, and the University of Manchester deserve special mention. If the principal source of any important material has not been properly referenced, the author invites this being called to his attention.

The author is grateful to the Northrop Corp. and Dr. Erik Ackerlind for the opportunity to enter the digital computer field. Thanks are due Lockheed Aircraft Corp., Hughes Aircraft Co., and Information Systems, Inc., for furnishing typing assistance, Mrs. Barbara Fine for typing the final manuscript, and the Aerospace Corp. for assistance on the subject index. Acknowledgment is due Mr. Geoffrey Post for his encouragement while the author was at Information Systems, Inc.

It is a pleasure to acknowledge valuable aid from the following individuals, who read and constructively criticized final page proofs: Chapter 3, William Shooman, System Development Corp.; Chapter 4 (transistor

circuits) and Chapter 5 (magnetic surface recording techniques), Marvin G. Ettinghoff, Librascope, Inc.; Chapter 5 (magnetic core memories), Milton Rosenberg, Electronic Memories, Inc.; Chapter 6, Dr. John M. Salzer, Space Technology Laboratories.

The list of acknowledgments would be incomplete without mention of my family's understanding and forbearance during the period of preparation of this book.

EDWARD LOUIS BRAUN

March 1963

1. Introduction

1.1. Uses of Number

The subject matter of this book is the stored program digital computer. We will consider its fundamental nature, ways of describing its logical organization, various means of mechanization, and principles and techniques useful in its synthesis and utilization. Since these machines accomplish their function by means of operations on numerically coded information, some preliminary discussion is in order on the subject of numbers. We will consider briefly the nature of numbers, certain symbols and notations used to represent them, and a description of mechanical and/or electronic means for representing numbers and operating on them.

Numerical symbols may be used for various purposes. Sometimes they are used merely as labels to distinguish one of a set of objects from the others. In other words, they can be used as names or symbols for objects. They are convenient to use as names of persons or things because they provide an inexhaustible supply of such names.

Ordinarily, one associates a definite order among numerals (or groups of numerals). Often, numerals are used for this characteristic alone, as in assigning them to houses on a street. The function of a street address is not to indicate how many houses there are on a street, but to indicate a particular house's position relative to other houses on the street, i.e., its order. The use of numerals to indicate the number of items in a set will be discussed in Section 1.2.

1.2. Counting

Before considering how numbers came to be associated with the process of counting, it is well to emphasize the distinction between ordinal and cardinal numbers since, in common usage, the word number alone may refer to either. When numbers are used solely for an order property that has been defined previously for them, they are called ordinal numbers (or ordinals)—for example, numbers indicating relative locations or points in time. Numbers used to designate the manyness of a set of things are called cardinal numbers (or cardinals). As mentioned in Section 1.1, numerals can be used merely as convenient tags or symbols to distinguish objects from one another. In this case, the numerals are used neither to

convey the property of ordinality nor cardinality. Within a computer, numbers may be used in any of these ways. An important property of all numerical symbols is that mathematical and logical operations can be performed on them to serve various useful purposes.

A fundamental numerical operation is that of determining whether the number of elements in one set is equal to, greater than or less than the number of elements in another set. An obvious procedure is to pair off an element in one set with an element of the other set, at the same time removing each element from its set, and to continue this process until one or the other set is depleted. For this process to be generally useful for enumerating elements in a set, it is necessary that one have available a standard set of sets. The smallest of these subsets will contain only one element, and the entire set of subsets may be built up from it simply by the addition of one element at a time. The idea of using a standard set of sets, formed from some easily transportable objects, resulted in a great convenience since it meant that one could determine the relative magnitude of two sets of objects and, also, the number of elements in each set without bringing the sets in proximity. Since each subset is included in the next larger subset of the set, the total number of elements to be provided did not have to exceed the largest set which might have to be enumerated. The most convenient set of elements at primitive man's disposal was the set comprised of his fingers (and toes), and it was only natural for him to use them (therein lies the origin of the quinary, bi-quinary and decimal number systems). As the need to enumerate larger sets developed, sets of small pebbles or beads (also easily transportable) came into use. At a later time, a symbol (or group of symbols) was assigned to each subset of the set. Then, the manyness of sets could be indicated conveniently in terms of these symbols. Finally, this led to the process of ordering the symbols in accordance with the manyness of the sets they represented, and to our present day convention in which we count by introducing the name of the symbol for the next larger set in a standard sequence each time the present set is augmented by "1." This same type of procedure allows us to count from any initially specified location in a sequence, to count backwards as well as forwards and, also, to count by multiple as well as single increments.

1.3. Numerical Symbols

Early man represented a single element by a mark like | or —, both because of their similarity to an extended finger and because they were easy to inscribe with a stick or other pointed instrument. Two elements

were represented by | | or $\overline{\underline{\ }}$. From the former is seen the origin of the Roman numeral II. The latter when written quickly, without removing the instrument from the writing surface, would appear as Z, and is the origin of 2. Similarly $\equiv$ becomes 3. The reasons for the choice of the other numerals are not so apparent, and need not be discussed here.

A major step forward in the representation of a collection of elements came with the use of special symbols to represent large collections of elements. For example, the Romans used V for five, X for ten, L for fifty, C for one hundred, D for five hundred, M for one thousand, etc. Even so, the representation of large numbers was cumbersome compared to present day notation. For example 3738 would be expressed as MMMDCCXXXVIII. Nevertheless, Roman numerals were retained in commercial accounting until the eighteenth century. One reason for their continued use was that they made addition and subtraction easier for those with little or no mathematical training. This was because they allowed these operations to be performed by a process more akin, on the surface, to counting than is addition (or subtraction) of modern numerals. Consider, for example, the addition of 854 to 3738

3738	MMMDCCXXXVIII
854	DCCCL IIII
4592	MMMMDLXXXXII

To obtain the sum in Roman numerals, one need only know that the sum of any number (from one through four) of the Roman numerals I, X, C, M, etc. was represented simply by writing each symbol in the sum a number of times equal to the frequency with which it appeared in the addends. The only other rules, which achieve a more compact notation, are that IIII + I = V, V + V = X, XXXX + X = L, L + L = C, CCCC + C = D, etc. Contrast this simple process of accumulating like symbols with modern decimal addition which requires memorization of the decimal addition table as well as knowing when sums or borrows are generated and their disposition.

1.4. Fundamentals of Computing Aids

The first significant mechanical aid to computing, the abacus, was invented in ancient times and is widely used in many parts of the world even today. It consists of an array of similar physical elements, each of which represents a count whose magnitude is determined by the row and column coordinates of the element's position. These physical elements are in the shape of beads, referred to by the Romans as calculi (the origin of the terms calculus, calculate, etc.). Each column is divided in

two by a crosspiece. There may be one or two beads above the crosspiece, and four or five below it. Each upper bead defines a count equal to that of five beads below the crosspiece in the same column. Both upper and lower beads within a column are free to move along the column. In the column on the extreme right each lower bead represents one. In the next column each lower bead represents ten. As one proceeds to the left, the value assigned to the beads in any column is ten times that assigned to the beads in the adjacent column on the right. A number is set into the abacus by pushing beads up to the crosspiece.

The abacus provided a speed advantage for addition or subtraction compared to the manipulation of written Roman numerals, due to the fact that beads could be moved about in less time than it took to write the operands and result in Roman numerals. Another reason for its use was that it provided a cheap means of temporary storage of information. Numbers could be readily inserted and erased simply by movement of the beads. Parchment and ink were not readily available, expensive, and therefore practical only for permanent records, documents, etc. Another cheap means of temporary storage that was used consisted of a board covered with a thin coat of wax. Marks could be scratched into the wax and erased by resmoothing the wax. However, this was a slow and tedious process. It was not until slates, blackboards, and paper came into common use just a few centuries ago that the abacus and similar devices known as counting boards were replaced in Europe.

In the abacus, an important concept appears whose significance was not appreciated until many centuries later. We refer to the idea of a positional notation, i.e., one where the value represented by a particular symbol is a function of where the symbol appears in a group. In the abacus, there is only one symbol, namely a bead, and these beads represent different magnitudes in accordance with their positions. The interpretation of a symbol according to its position is characteristic of modern numerical representation and one of its most important features. It makes it unnecessary to create new symbols for successively larger counts. With the positional notation, and its simple rule for going from one number to the next larger one, any new number, as large as we please, may be written from a basic small set of symbols. The number of symbols required to represent any magnitude depends on how many symbols are used in defining the basic set. In the decimal system, based on using the fingers for counting, ten symbols, the numerals 0, 1, 2, 3, . . . 9 comprise the basic set and the representation of a number such as

$$d_i\, d_{i-1}\, d_{i-2} \ldots d_0\, d_{-1}\, d_{-2} \ldots d_{-j}$$

is really the shorthand notation for

$$(d_i \times 10^i) + (d_{i-1} \times 10^{i-1}) + \ldots + (d_0 \times 10^0) + (d_{-1} \times 10^{-1}) + \ldots$$

where each d may represent any of the ten symbols: 0, 1, 2, 3, . . . 9. For example, the representation 2943.0 implies: $(2 \times 1000) + (9 \times 100) + (4 \times 10) + (3 \times 1) + (0 \times .1) = 2943.0$. A system of numerical representation in which only two symbols are used is referred to as the binary system. A representation of a number in the binary system, such as

$$b_m\, b_{m-1}\, b_{m-2} \ldots b_0\, b_{-1}\, b_{-2} \ldots b_{-n}$$

is the shorthand notation for

$$(b_n \times 2^n) + (b_{n-1} \times 2^{n-1}) + \ldots + (b_0 \times 2^0) + (b_1 \times 2^{-1}) + \ldots$$

where each b may represent either of the two symbols in the binary system: 0 or 1. Each of the symbols in a binary number is referred to as a bit (for binary digit). A comparison of the decimal and binary representations for numbers of magnitude from zero to ten is shown in Table 1.1. From it, a disadvantage of the binary system for written notation is apparent, namely the fact that the representation of a number requires, in general, more symbols than does the decimal system.

TABLE 1.1.

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

All mechanical, electromechanical, and electronic digital computing aids utilize the positional notation in the representation of numbers, for it allows a number of any magnitude to be formed from a defined, small set of symbols. This is of fundamental importance for the following reasons. First of all, it allows a digital unit to be fabricated from a relatively small number of standardized elements. Also, it is responsible for the high precision attainable, for the precision may be increased indefinitely simply by the use of more elements. For example, a common

mechanical method of representing a digit is by means of a notched wheel or disk which can be turned by a shaft through its center. At any time, the disk is defined to represent one of the symbols 0, 1, 2, . . . 9, depending on the disk's angular displacement from an arbitrary reference. To represent a number with, say, n digits, n similar disks are used. To perform an addition, it is necessary to displace each disk by an amount proportional to the value of the digit to be added to the order defined by each disk. It is also necessary to intercouple the disks in such a manner that when any disk passes from the 9 to the 0 state, a motion is imparted to the disk in the next more significant position such that it passes from state i to $i + 1$.

As a rule, mechanical and electromechanical computing aids used for normal computational work employ the decimal system. This is because it is a relatively simple matter to define and maintain ten distinct positions of a rotary element. In electronic computing aids, where a number is represented by such things as the amount of charge on a dielectric material, the state of magnetization of a magnetic element, or the voltage at some point in an electronic circuit, the use of the decimal system produces difficulties. These can all be attributed to the fact that it is difficult to control the placement of an electrical element precisely into one of ten stable states, and equally difficult to read the states of circuits to such precision. Because of such practical difficulties, all electronic digital computers are formed from binary elements, i.e., switching and storage devices which need assume only two distinguishable stable states. Hence, use of the binary rather than the decimal system is dictated. This is because two (or some power of two) is the most economical radix to use with binary elements simply because all possible configurations of a group of binary elements can then be utilized. The use of the binary system presents no great difficulty, and there is no intrinsic reason why one must use the decimal number system. One may choose any radix for the base of a number system. The Babylonians used the sexagesimal system (i.e., the base was 60, as opposed to 10 for the decimal), the Mayans used the duodecimal system. Outside of psychological reasons, stemming from its common use in all phases of human commerce, the decimal system is not the best to use for computing.

Even though numbers are represented within a machine in the binary number system, conversion between the two types of number systems can take place at the inputs and outputs of a machine so that, as far as the user is concerned, the machine operates in the decimal system. To facilitate this conversion, the binary-coded decimal system (see Chapter 6) may be used in the internal storage elements of a computer as well as in its input and output equipment.

1.5. Quantization

Media used for the recording of information are usually capable of responding to a continuous range of input signal intensity, from the so-called threshold level to the saturation level. For example, information can be represented on magnetic tape by the intensity of magnetization of specified areas on the tape, and this intensity is determined, over the threshold to saturation range, by the magnitude of current applied to a recording head. A measure of the amount of information stored in any one area is given by the total number of levels of magnetization that can be recorded and sensed. Ideally, it would be desirable to store a large amount of data with a minimum amount of storage media. However, in practice a compromise must be made in order to reduce the probable error in interpreting the recorded information when sensed at some later time. Use of quantization in the recording and sensing processes allows one to trade efficiency of storage for a greater probability of correct interpretation of the data.

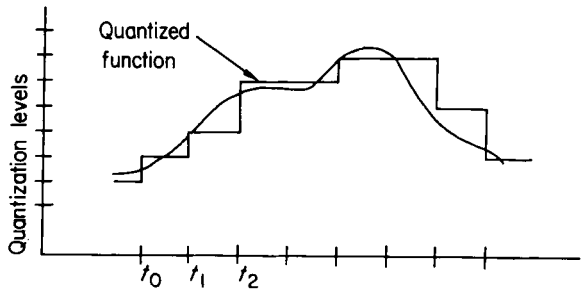


FIG. 1.1. Quantization of a function

The nature of the quantizing process is shown in Fig. 1.1. Note, first of all, that the range of values of the function is divided, on some basis, into a number of smaller subranges. The values selected to define the subranges are called levels of quantization. One way of quantizing a function is to replace its value by the value of the nearest level of quantization whenever it passes the halfway point between two levels, the value of the quantized function remaining constant between such occurrences. However, in practice, and as shown in Fig. 1.1, the quantization process is usually associated with a fixed period sampling process wherein the original function is inspected at times t_0, t_1, t_2 . . . and its value replaced by that of the closest level of quantization at these times. In a synchronous digital computer, the instants of time, t_i, would be specified by a timing

source referred to as a clock (see Chapter 3). Often one uses a quantizing process without realizing it as, for example, when reading a dial, gauge, or scale to the nearest unit.

The way in which quantization can be employed to reduce the probability of misinterpreting stored data will now be described. For the purpose of illustration, consider again a magnetic storage medium. For each state of magnetization of the medium, there is a finite probability that an accidental event will cause a transition to some other state of magnetization. Of course, the greater the separation between two states the less the probability that an accident will occur to cause the transition from one state to the other. Accordingly, the difference between levels of quantization can be defined in such a way that the probability of a signal on one level being mistaken for that on an adjacent level is less than a specified amount. An ideal, absolutely stable state of a storage medium exists only if a perfect switching action is involved, i.e., if an impulse of energy is required for a transition between two states, and only in this case could the probability of misinterpreting stored data be reduced to an absolute minimum. In practice this situation is adequately approximated by choosing levels sufficiently far apart that a large amplitude signal is required to switch the storage element from one level to the other. The passive storage elements used in all contemporary electronic digital computers are referred to as binary elements because of this type of arrangement. Bounds are specified about each level within which the sensing device reports the same value. This is done so that variations from the specified levels, due either to small irregularities in the medium or small transitions that may have occurred, are not sensed. Two advantages of the binary quantizing process are apparent. It allows for exactness and for reproducibility of results. Binary quantization reduces the problems associated with measuring physical parameters to a simple determination of the presence of signals near the threshold and saturation levels. Thus, the uncertainties entering into measurement are replaced by the relative certainty of detection of a large amplitude signal. Binary quantized data can be processed with relative immunity to the compounding of small errors that occurs in a nonquantized data system. In a binary system the end result of a series of operations will always be the same no matter how many times it is repeated. This is an especially important factor in the processing of commercial data where money and other items must be accounted for, not to within some tolerance of error, but to a precise figure.

Of course, for an increased efficiency of storage multilevel storage devices could be used if the additional levels could be recorded and

sensed without an objectionable increase in the probability of error of interpretation.

1.6. The Evolution of Computing Aids

It is only natural to expect that means for representing and processing information would be influenced by the technological level of each age. We have already observed that man first counted by means of his fingers, and later used line segments to represent elements in a collection. In early forms of the abacus, developed over 5000 years ago, a number was represented by the pattern in which a set of pebbles was arranged. Later the abacus evolved to its present form which differs principally in that beads are threaded on wires or thin rods that define the columns and the whole is enclosed in a frame. There were no significant new developments in computing aids until the seventeenth century. Then in 1642, the first desk calculator was invented by Blaise Pascal. It could perform addition and subtraction, and its operation was based on the use of toothed wheels. Leibnitz designed the so-called stepped wheel, and improved upon Pascal's machine by devising a means of multiplication by repeated addition. A machine with this feature was completed in 1694, but suffered from mechanical imperfections. Further improvement on Pascal's machine was made by Thomas de Colmar who, in 1820, produced the first successful machine for multiplication. In 1878, the Swedish engineer Odhner, invented the pin-wheel method of adding numbers from one to nine. His patents were subsequently incorporated in the Brunswiga hand calculating machines. The first successful key driven adding machine, the Comptometer, was developed in 1887 by D. E. Felt. After this time, a number of significant improvements were added by new designs as well as additions to old ones. Single operation multiplication was introduced by Leon Bollée in 1888. In 1889, a printing feature was added to the Comptometer. After 1910, electric drive motors were added to mechanical calculators. This allowed more complex circuits to be used, since keys or light parts of the machine's internal mechanism could be used to actuate control switches.

In the era of the Industrial Revolution, the idea of mechanical automata achieved a marked popularity, and many ingenious mechanical devices were developed which, upon being actuated would follow a prescribed set of motions. Two relatively important devices were developed at this time to control the motions of two quite different mechanisms. From them have evolved two input–output media widely used with present electronic digital computers, and which in their present form are basically

similar. One of these devices was a metallic disk or cylinder upon which bumps were placed at designated points to control the times at which different notes were struck in a music box. From it were developed player piano rolls and various types of punched paper tape, most notably those used to control teletype equipment. The other was the Jacquard card, used to control the weaving of patterns into cloth. It served as a model for the development of the Hollerith punched card. The holes in these cards are sensed by electric circuits connected to metal brushes that make contact through the holes. The punched card, on which a number is represented by a pattern of punched holes, was not conceptually an advance over the abacus. However, it afforded the first significant practical means of semiautomatic data processing. Its importance was derived from the many special types of electromechanical units that could be, and were devised for the rapid sorting, interpretation, and manipulation of data on cards. By 1945, punched card machines were in widespread use throughout the world for the tabulating, sorting, and analysis of data for accounting and statistical purposes.

Another important contribution, current with the development of punched card machines, was the development of relays for controlling complex telephone switching networks. These switching systems showed that it was possible to perform complicated logical operations with relays, and to obtain reliable operation by self-checking techniques. In 1938, Stibitz developed, at the Bell Telephone Laboratories, a relay computer capable of addition, subtraction, multiplication, or division of complex numbers and which could be remotely controlled. In subsequent relay computers a self-checking code was introduced to detect a malfunction in the transmission of numbers. Subsequently, other relay computers were developed at the Bell Telephone Laboratories. These machines and the Harvard Mark I Calculator, developed jointly by IBM and Harvard University, were the pioneer efforts in relay computers. The latter machine was the first large scale general purpose digital computer to be completed (1944). Punched cards were used as the input and main storage medium, and relays were used for the arithmetic unit. About the same time, the first electronic digital computer was built. This machine, termed the ENIAC (Electronic Numerical Integrator And Calculator), was developed by the Moore School of Engineering at the University of Pennsylvania. It contained about 18,000 vacuum tubes. The last decade has seen the rapid development of the stored program electronic digital computer. This type of machine has wide application because it can perform many types of information processing operations at high speed and without human intervention, once a suitable program of instructions has been entered into it.

Although the state of development of commercially available digital computers was quite limited up to the time of World War II, the concept of a stored program automatic digital computer had been worked out by Charles Babbage in about 1833 in his design for an "Analytical Engine." The plan of this machine called for 50 digit numbers and a storage capacity of 1000 numbers, and it was intended that Jacquard cards be used in two ways. So-called "operation cards" were to be used to convey instructions to the arithmetic unit, and "variable cards" to specify the locations in storage from which two operands were to be taken and the result of a computation stored. The plans called for punching the output data on cards, for the purpose of having them available for future computation. Also, there was to be a device for printing results directly and a means for producing stereotype molds to be used for printing additional copies. Babbage's writings show that he was aware that the same language could be used for numbers and instructions, and that the machine could be made to modify its own program in accordance with the results of computations.

1.7. The Representation of Numbers in an Electronic Digital Computer

We have seen that in a mechanical digital computer, a number is represented by discrete positions of a shaft and numerical information is transmitted between these elements by the coupling of discrete rotary motion. In an electronic digital computer, a number is commonly represented in binary form by the current state of a set of storage elements each of which is capable of, and restricted to, assuming two stable output voltage levels. Information is transmitted between these elements either serially in the form of voltage pulse trains on a single information channel, or in parallel by the signals currently present on each of a set of information channels. Each of the pulses in a train may represent a single increment (corresponding to counting), or a set of pulses beginning and ending at defined positions in time may represent a number in some binary coded form (corresponding to a positional notation). The former representation may be termed a unitary weighted pulse train, and the latter a binary coded pulse train. Both types of serial representation are shown in Fig. 1.2. In the binary coded pulse train, the first bit of the train to appear represents the least significant bit of the number. This is for reasons associated with the computation processes. Since the electrical waveform convention is that time flows from left to right, the order of digits in a number so

represented will be reversed from the order of digits in the conventional written notation.*

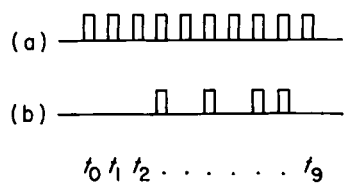


FIG. 1.2. Serial representation of a number by a pulse train
(a) Unitary weighted pulse train: the ten pulses represent ten unit increments
(b) Binary coded pulse train: The presence of a pulse at time t_i represents an increment of 2^i. The number shown is $2^3 + 2^5 + 2^7 + 2^8 = 424$

The parallel representation of a number within a computer is always in a binary coded form where different weights are assigned to the different channels in a group. Figure 1.3 illustrates how a four bit number would appear in a parallel representation. At each time, t_i, there may or may not

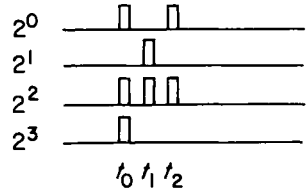


FIG. 1.3. Parallel representation of a number
$N = b_0 2^0 + b_1 2^1 + b_2 2^2 + b_3 2^3$ where $b_i = 1$ or 0
At t_0, N = 13; t_1, $N = 6$; t_2, $N = 5$

be a pulse on each of the lines. Each line has an assigned weight as shown, and the value of the number, N, appearing at any time is the weighted sum of the pulses on all the lines. An important distinction between serial and parallel weighted numerical representation is that in the

*Our numerical symbols were introduced from India by the Arabs who read from right to left. When introduced into English, the matter of reversing the order of a group of numerals to conform to our reading convention was overlooked. However, in reading it is not inconvenient to read the higher orders of a number first, for it is the complete configuration that conveys the value.

former case different weights are assigned to different time positions, i.e., the weight of a pulse is determined by the relative time of its appearance at a given point, while in the latter case different weights are assigned to different physical locations.

Since there may be either positive or negative pulses on a line, a choice may be made as to how to represent 0's and 1's. For example, a positive pulse may be chosen to represent 1, and a negative pulse to represent 0, or vice versa. When it is desirable to use pulses of only one polarity, the presence of a pulse may be used to represent 1 and the absence of a pulse 0, or vice versa. If a unitary weighted pulse train represents information that has been generated asynchronously, then a single line may be used for transmitting either positive or negative increments, but not both. To provide for both, two lines must be used, the presence of pulses on one line representing positive increments and the presence of pulses on the other representing negative increments. Either positive or negative pulses may be used to represent increments on either line, as shown in Fig. 1.4. The choice will depend on the characteristics of circuit elements. Schemes (a) and (b) are most commonly used.

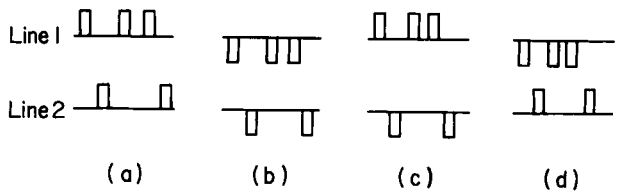


FIG. 1.4. Asynchronous unitary weighted pulse trains

The use of a numerical biasing technique permits the use of a single line to transmit unitary weighted information of both positive and negative sign if the information appears synchronously. Using this technique, the presence of a pulse on the line at any time defined by the timing source indicates a positive increment while the absence of a pulse indicates a negative increment. This technique depends on generating a train of alternate 1's and 0's in a system in the absence of positive or negative increments. This type of information transmission is sometimes referred to as binary information transfer, while the method described in the preceding paragraph is termed ternary transfer.

In a binary coded pulse train, a single line is always sufficient for the representation of positive or negative numbers. The sign of the number is indicated by the presence or absence of a pulse in a position reserved

for sign identification. The number may be represented either as an absolute value plus sign, or a complementary system may be used to represent negative numbers. (See Chapter 6.)

1.8. Arithmetic Processes in Digital Computers

A considerable amount of space in the text is devoted to a description of different ways of performing the commonly encountered operations of counting, addition, subtraction, multiplication, and division in a digital computer. Since these operations are apparently so simple when performed in our heads or with pencil and paper, some words of explanation may be in order. First of all, the characteristics of different physical elements used to perform these operations must be considered. A procedure suitable when using one type of element may not be suitable when using another. Also, certain logical formulations of these processes may be more economical equipment-wise than others. Finally, a vast number of different types of serial, parallel, and serial–parallel, i.e., semiparallel, operation is possible, some more suitable with particular physical elements than others. The average person, in performing computations, functions in a serial manner, i.e., he performs one operation at a time as, for example, in adding where he adds one column and produces one digit of the sum at a time. Since he can only write one digit at a time his serial arithmetic operations are adequate. However, within a digital computer, it is possible to incorporate control circuits that cause all digits of the addend and augend to be sensed simultaneously. The sum can then be recorded in far less time than required by serial arithmetic operation. The purpose of all parallel arithmetic operation is to decrease the time required for computation. This is paid for by an increased amount and complexity of equipment. When serial operation is not fast enough, and parallel operation is too expensive, a compromise may be made with serial–parallel procedures.

1.9. Redundancy

Let us return again to the subject of numerical representation used by the Romans. Not only were different symbols used for different orders, e.g., X for 10, C for 100, etc., but these symbols were written by convention in an ordered relation. For example, 2153 was written as MMCLIII, not as, say CLMMIII, LCIIIMM, etc. However, if one were to come upon one of the latter representations, he could still interpret it correctly. The Romans, though fond of order, did not appreciate the fact that placing symbols, by convention, in an ordered relation makes it unnecessary to have different symbols for different orders. Consequently,

the Roman notation had a redundancy which is not present in modern positional notation. As a rule, redundancy is a measure of how efficiently a particular set of symbols or type of notation conveys information. (In Chapter 9 the subject of redundancy is considered in relation to error detection and correction). The use of a positional notation allows a small set of symbols to be adequate for representing any magnitude, and a proper choice of radix (the number of different symbols) allows large magnitudes to be represented by a reasonable number of symbols.

It is not unusual for primitive notations to exhibit redundancies. The same can be said for initial designs of new equipment. In an "idealized" type of world where no unintentional disturbances, i.e., accidents, were possible, and everything always functioned as designed, redundancy would serve no purpose. However, in reality it often proves useful as a means of reducing or eliminating the detrimental effect of an accident. How this can be done will be discussed, for certain types of failures, in Chapter 9. Here we only wish to point out that once the logical requirements of a piece of equipment such as a digital computer are better understood, redundancy may be minimized for the sake of economy. However, even then one may find it desirable to incorporate certain intentional redundancies for the sake of improving the reliability of performance of a system composed of nonideal physical elements.

1.10. Computer Applications

Human progress is dependent on efficient means for the processing of information, as an aid to the creative processes of thought. Simply for the purpose of drawing an analogy, we will consider first certain functional similarities between information processing by humans and machines.

When an individual is confronted with a problem, he may call upon intuition, learning, and experience to solve that problem. All of these terms refer to the fact that he has available inherited and acquired information pertinent to the solution of specific problems. This information is stored (in ways as yet undetermined) in his memory. This memory is of sufficient capacity to store vast amounts of information pertinent to the solution of specific problems, and there are mechanisms for integrating various sections of this stored data in a manner appropriate to the solution of new and more complex problems. It will be shown in Chapter 2 that before a digital computer can produce the solution to a specific problem, it, too, must be furnished with information—in the form of a program which describes a specific sequence of operations to be performed. If a computer is to be able to solve different problems, it must be furnished with appropriate programs. These are stored in its memory,

referred to as the store, to which access may be gained by means of a control unit.

Though a stored program digital computer is conceptually a simple device, its high rate of operation and facility for arithmetic and logical operations, coupled with the ingenuity of its users, make it of great utility. Outside of their importance to many specific areas (delineated below) such machines are contributing materially to the acquisition of knowledge by processing vast amounts of data and performing computations to check new theories. An important indirect benefit they provide is the introduction of improved procedures and terminology to areas previously limited in their use of systematic mathematical and logical formulations.

Application of digital computer technology to more fields of human endeavor is increasing rapidly. In the business world, digital computers facilitate and accelerate the extensive routine data processing vital to daily commerce, e.g., processing of credit transaction data, customer billing, inventory control and various accounting operations. Information processors are essential to the military in many areas, e.g., in military intelligence data processing, early warning systems, command and control systems; for automatic navigation of ships, aircraft and space vehicles, automatic control of weapons systems, automatic checkout of complex electronic systems prior to use. Computers can be used in industrial automation for data refinement and assimilation, scanning of instrumentation for detection of alarm conditions, automatic data logging, evaluation of plant performance and control of machines, plants and processes. Computers can be used in factories to improve record keeping and scheduling of production. They can be applied to the regulation of traffic, e.g., 1) aircraft, train, steamship and freeway traffic, 2) messages to be routed through complex communications networks (such as satellite relay systems), 3) commodities like natural gas and oil, whose flow through hundreds of miles of pipeline distribution systems must be economically controlled. Application to management problems, whether in industrial, governmental or military areas, will be extensive because the amount of data upon which decisions must be based is increasing while the time available for decision making is decreasing. Some interesting applications on the horizon are: 1) teaching machines for efficient, automatic factual instruction, 2) machines to aid medical diagnosis, 3) large scale information storage and retrieval systems.

One of the most intriguing areas of investigation is the application of artificial intelligence systems* to problems which, though well defined, are too difficult for complete analysis. Obtaining a solution to some of these

*For a lucid survey of this subject see M. Minsky [1961] Steps toward artificial intelligence, *Proc. IRE*, 49, 8-30.

problems by an exhaustive search and test of all possibilities would require an unattainable amount of time even with the fastest machines. Therefore, various techniques are being investigated to produce computer programs that limit the search to manageable proportions. For example, there are: 1) pattern recognition programs which, by extracting significant features from a totality containing much that is irrelevant, classify problems into categories for which specific problem solving procedures may be prescribed, 2) learning programs which generalize on accumulated experience, 3) planning and administrative procedures for attacking the over-all problem and its interrelated parts, 4) inductive methods which, given a model of a universe, can generate useful predictions for it. All of these higher level information processors are based on externally observable features of schemes by which men attack new problems.

A mental process important to creativity is that by which relations between events are recognized, stored and new information being associated and integrated to form new ideas. This is usually on a gross level at first, but with continued refinement can lead to a useful model or set of laws which state these relationships in a quantitative manner. No one has yet brought forth a means by which a computer could create a useful new concept. Even if a machine could generate new theories, say by some statistical process of connecting various facts and testing them for consistency, it would still lack criteria for selecting meaningful ones unless well defined abstract and/or physical goals were implanted by a program (realization of a physical goal requiring interconnection between the computer and appropriate actuators).

Application of digital computers to higher level types of information processing has brought renewed speculation on whether machines can be made to think (reminiscent of earlier descriptions of digital computers as electronic brains). This speculation is meaningless since "thought" has never been adequately defined and serves merely as a label for a complex of mental processes whose mechanisms are not understood. The term "artificial intelligence" simply refers to higher level information processing performed by machines which have been furnished with heuristic and/or algorithmic devices for solving problems. This extension of man's intellect does not degrade his dignity, as some suggest, but is a further expression of his mental powers.

In case you are chagrined by the suggestion of being only an intelligent machine we submit an observation from Karl Jasper's *The Future of Mankind* (translated by E.B. Ashton, University of Chicago Press, 1961): "Intelligence alone loses sight of final ends, of life itself, of the totality of conditions of life in the pursuit of particular realizable goals. Something more must control as well as animate mere intelligence."

2. The Nature of Automatic Computation

2.1. Elements of Information Processing Systems and Types of Digital Computers

Before discussing the structure and techniques for utilization of digital computers, a few words may be in order concerning information processing systems in general. The term, information processing system, is used here to include all systems containing the following elements. (1) Sources of information from which data is obtained. (2) Transmission links which convey the source data to a central processor and from there to locations of end use. (3) The central processor which applies elementary and/or complex transformations to the original data to obtain a final set of data in a desired form. (4) Output terminals for the processed data, including cathode ray tubes or other visual displays, printers, recorders, and input signals to actuators in control systems.

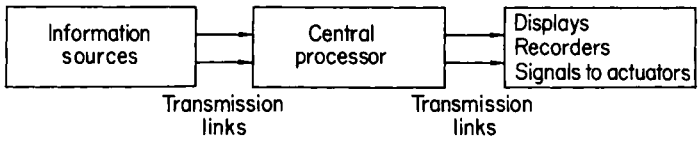


FIG. 2.1. Elements of an information processing system

The elements of an information processing system are shown in Fig. 2.1. Many devices and methods have been developed for the purpose of accomplishing each of the specialized functions:

(1) *Information collecting devices:* instruments for sensing pressure, temperature, electromagnetic radiation, fluid flow; composition analyzers; radars; human beings; etc.

(2) *Transmission links:* phone, teletype, coaxial lines, radio, vehicular transportation, human beings, etc.

(3) *Central processors:* desk calculators, sequencing devices (or programmers), coding devices, electronic digital computers, regulators, analog computers, human beings, etc. A central processor may perform one or more of the following types of operations on input data: arithmetic,

logical transformations, sorting and classification, conversion from one type of code or numerical representation to another, storing, sequencing.

(4) *Output terminals:* For scientific, engineering, or business studies, the outputs of a digital computer are usually graphs or printed data. For commercial applications such as billing of notices to customers, or payroll computations, the output is in printed form. In control applications, e.g., in industrial process control systems, or airborne navigation, flight management, and weapons control systems, a number of simple monitoring displays are provided in addition to the electrical signals generated for controlling the operation of various actuators within a regulator or servo system. In other control applications, e.g., tactical data systems for processing radar, logistics, and intelligence information, or air traffic control systems, input signals to various actuators must also be provided, but the major emphasis is on a large number of displays, including cathode ray tubes, display counters, and yes–no indicators, to inform responsible personnel of the various aspects of a situation as it develops.

When a "general purpose" digital computer, or GP machine, is referred to, a central processor is implied that is capable of any of the operations listed under item (3). A few words are in order concerning the term "general purpose" computer. Unfortunately, it often leads to confusion or awkward types of descriptions. This tag became affixed to the first large electronic digital computers. It arose because of the flexibility of these computers in solving many different types of problems. This flexibility derives from two principal sources. First, these machines are capable of executing a large number of different elementary operations, from which more complex operations can be obtained by combining the elementary ones. Second, the manner in which elementary operations are to be combined for the solution of a specific problem is specified by a sequence of coded instructions, termed a program, which is inserted in the computer's central memory known, too, as the main or central store, and which controls the execution of a problem. Different programs are inserted for the solution of different problems.

At a later date, similar computers were designed for specific applications. This allowed simplifications to be made, since only a single fixed program had to be provided for. However, the tag "general purpose" had already been assigned to this class of equipment, and therefore machines designed for a special function were termed fixed program GP machines.

Another class of machines was devised primarily for the purpose of solving differential equations. Its chief difference from the GP is that only single bits, i.e., increments of information rather than whole numbers, are transferred on its internal communication lines. This type of machine

is referred to as a digital differential analyzer, or DDA. It can be used to solve any of a number of different types of algebraic as well as differential equations, and, in this sense, is a general purpose computer. Most DDA's in use are of the fixed program type, designed for incorporation, separately or in conjunction with a GP machine (the two usually sharing a large capacity memory), into a control system. In this case, only a particular set of equations has to be solved, subject to different initial conditions and forcing functions.

To conclude, if a particular machine is designated as a special purpose or general purpose type, it is not always clear whether reference is being made to a GP machine or a DDA. One way out of this confusion is to refer to one type of machine as an absolute, arithmetic, or integral transfer computer and the other as an incremental or incremental analyzer, or incremental transfer type of computer. The additional classification of "general" or "special" purpose is made according to whether the machine has a variable or fixed program.

2.2. The Nature of Automatic Computation

Our purpose here is to demonstrate the simplicity of the concepts involved in the design of a general purpose arithmetic digital computer. First of all, it must be emphasized that a computer cannot perform any mathematical or logical operation beyond the capability of a suitably trained human being. Its great utility is derived from the high speed capability of the electronic circuits used to perform arithmetic and logical operations and other functions. There are descriptions of such circuits in Chapters 4, and 5. At this point, we will consider the fundamental nature of computational processes* performed by human beings, and how these processes may be simulated by an automatic computer.

Consider first how a human being with only pencil and paper performs a computation. Suppose, for example, that he wishes to determine how much money he has spent during the previous weeks. He has several bills, each of which has some amount of money specified on it. These bills can be considered as storage devices because they retain information, making it available when needed. To produce the sum total, he would most likely list the amounts of the individual bills on a piece of paper, and then proceed to find the sum. The following different types of elements entered into the operations described:

*Since these processes were evolved to facilitate computation by human beings, they may not necessarily be the best methods for computers. It may develop that specialized methods of computation will evolve, and eventually change our present day techniques of mathematical education.

(1) *An input element:* The pencil, controlled by the human being, which effected the transfer of the individual pieces of information from several places (the individual bills) to a single place (the tally sheet) where the required summation could be performed.

(2) *Storage elements.* Permanent: The bills which retain information and make it available when needed. Temporary: The tally sheet. After the sum is obtained the information on this sheet of paper may be erased, if required for no additional purpose.

(3) *Arithmetic element:* Certain parts of the brain which are capable of performing the operation of addition. The inputs to the brain's arithmetic section are derived via visual signals received from the pencil marks on the paper.

(4) *Output element:* The pencil, by means of which the human being records the answer on the sheet of paper.

(5) *Control element:* The control element controls the flow of information between the other elements. In this case it is the human being who determines from where information is to be accepted, what types of operation to perform, and where processed information (the answer) is to be stored.

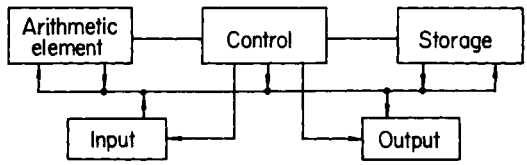


FIG. 2.2. Interrelation of elements in a digital computing system

The interrelation of the various elements is shown in Fig. 2.2. In our example, the data transmission lines shown in Fig. 2.2 are of two types. Information is transmitted from the paper to the eye via light rays, and from the eye to the brain via electrical signals propagated along a nerve bundle. It is transmitted from the brain to the paper via electrical signals which cause the fingers of the hand to move a writing instrument so that it forms the desired characters.

Certain variants of the operations described are possible. For example, the use of pencil and paper as input–output and temporary storage elements may be eliminated. Information from the bills may be sent to temporary storage positions in the brain, the computation performed in the brain's arithmetic elements, and the final answer stored in other storage elements of the brain. However, not many people have developed the

facility of performing a long sequence of arithmetic operations "in their head" and therefore mechanical aids are utilized. An important aid is the electromechanical desk calculator which permits long sequences of arithmetic operations to be carried out in less time and with less fatigue for the operator (consequently with less chance for error) than computation with only pencil and paper.

The types of operational capabilities required of a general purpose electronic digital computer will now be considered. An important point to remember is that the function of performing arithmetic and logical operations comprises only part of a computing system. Another important function is the transfer of information from one locality to another. In an average computation using a desk calculator an appreciable percentage of time is spent in information transfer operations, e.g., transferring information from original sources to data sheets, copying information from data sheets into the calculator, copying intermediate results from the calculator onto sheets of paper, preparing sheets of paper with the final tabulated answers.

It becomes apparent that in order to increase the speed of a complete computational process, it is necessary to increase the speed of the transfer operations as well as that of the arithmetic operations. This implies the elimination of the human operator once the computation is begun, for he is the bottleneck. Clearly one gains very little by decreasing the time required to perform an arithmetic operation from say, 1 sec to 1/100 sec, if at the end of each operation a human operator has to spend several seconds copying information out of and inserting new information into an arithmetic unit.

The utility of present electronic digital computers results not only from their basic high speed of operation, but also from the fact that they can perform without human intervention all the steps required in a computation involving thousands of operations. This is possible because of three distinct reasons. First of all, a method for solution of a given problem can be stated in terms of a relatively short program which lists all the elemental arithmetic, logical, and transfer operations that are to be performed in the course of solving a problem. This program is prevented from becoming too long by the use of iterative problem-solving techniques in which a sequence of operations is repeated until a desired result is obtained. This makes it unnecessary to write new steps for each repetition of the sequence. Instead, one merely writes the sequence and specifies that it be repeated, with new initial conditions each time, until a desired result is obtained, at which time an indication is provided by the machine. Second, the control unit of the computer causes the individual steps of a computation to be carried out as directed in the program. Third, the com-

puter can execute conditional transfer instructions. This permits the operations in a subsequent iterative section of a program to be initiated without human intervention upon the successful completion of a preceding iterative section of a program.

The general types of instructions that a computer should be capable of executing are:

(1) Combination transfer and arithmetic or logical instructions. These instructions cause information from specified memory locations to be brought to the arithmetic unit where operations such as the multiplication, division, addition, subtraction, and comparison of two numbers are performed. The result is left in the arithmetic unit at the end of an operation.

(2) Operations involving the arithmetic unit only with no reference to the memory; e.g., shift instructions.

(3) Transfer instructions which cause a transfer of information from one part of the computer, e.g., the memory or arithmetic unit, to one or more other parts of the computer.

(4) Control transfer instructions. There are two major types of control transfer instructions. The unconditional transfer instruction transfers control to an instruction out of sequence. The transfer may be to auxiliary programs outside the main program or may serve to skip instructions in a given sequence. This type of transfer does not involve the use of any data not contained in the instruction itself. One conditional transfer or test instruction operates as follows: If the number in the arithmetic unit (the accumulator) is $\geqslant 0$, control will proceed to the next instruction in sequence, but if the number is < 0, control will be shifted to the instruction located in the memory position specified by the conditional transfer instructions (for some variants see Section 7.2).

(5) Instructions involving the transmission of information from the input units and to the output units.

2.3. Computation by a Stored Program Digital Computer

The following discussion will show how automatic computation can be achieved even by a computer capable of executing only a very few simple instructions. Assume that the computer contains the following elements (shown schematically in Fig. 2.3):

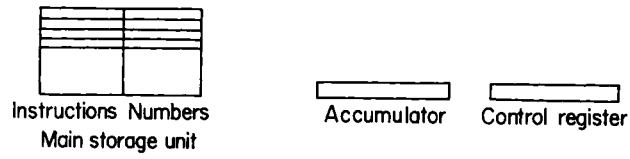


FIG. 2.3. Principal storage units in a digital computer

(1) A main storage unit which has adequate capacity to hold the coded representations of the instructions in a program, as well as numbers that must be stored, e.g., constants, initial values of problem parameters, and numbers generated during the computation which must be stored temporarily.

(2) An accumulator, which is a special storage register associated with the arithmetic unit. It holds an operand in a form accessible to the arithmetic unit, allowing certain operations to be performed on it. In operations involving two operands, it holds one while the second is located in the store and transmitted to the arithmetic unit. Also, it serves to store a result until it can be transmitted elsewhere.

(3) A register called the control register whose contents indicate from which location in the main storage to obtain the instruction to be executed next.

Assume also that the computer can execute only seven different instructions. The nature of these instructions as well as mnemonic codes for them are shown in Table 2.1.

TABLE 2.1. Instruction repertory of a simple, hypothetical computer

Code	Instruction
cA m	Add the contents of storage location m to the cleared accumulator.
A m	Add the contents of storage location m to the contents of the accumulator, leaving the sum in the accumulator.
S m	Subtract the contents of storage location m from the contents of the accumulator, leaving the difference in the accumulator.
C m	Copy the contents of the accumulator into storage location m.
U m	Transfer the address m to the control register.
T m	Test (i.e., inspect) the sign of the number in the accumulator. If the number is negative transfer the address m to the control register. If the number is zero or positive do nothing.
STOP	Go into an idle state.

The code used to represent an instruction consists basically of two parts: 1) an operation field in which is placed a code symbol for a specified operation, 2) an address field in which appears a number whose meaning depends on the operation. In the one-address type of machine described here (see Section 7.5.4 for a description of multi-address instructions) the number in the address field of certain instructions (for example, cA m, A m, S m and C m) indicates a storage location whose contents are to be transmitted elsewhere or altered; in transfer of control instructions (for example, U m, T m) the number in the address field indicates an address to which control will or may be transferred; in a binary

shift instruction it indicates the number of binary places to be shifted. The list of instructions in Section 7.2 indicates other uses to which the address field may be applied. Because failure to distinguish between an address and data stored at that address can be a major source of difficulty in writing and understanding programs, sometimes parentheses are placed about the number in the address field to emphasize that the contents of the designated location are being referred to. However, in this text parentheses will not be used in the instruction codes.

The STOP instruction defined in Table 2.1 implies that the computer is capable of being in either an active or idle state, and that external means are provided for placing the computer in one or the other state. This is the case and the actions of the computer in these states is outlined below.

State	Operations Performed
Idle:	Do nothing
Active:	When not otherwise occupied: (1) Add 1 to the number in the control register, leaving the sum there. (2) Read and execute the instruction in the storage location designated by the new number in the control register.

We will specify that when the computer is first set to an active state, the contents of the control register are set to zero. As a result of operation (1) in the active state, the number in the control register is changed to 1 and the instruction stored in the storage location designated by 1 is executed. After the execution of each instruction, the instruction in the next consecutively numbered storage location will be executed. Instructions U m and T m can cause an exception to this normal sequence of operations. If the instruction U m appears in storage location i, the next instruction executed after it will not be that in storage location $i + 1$, but rather that in storage location m. If the instruction T m appears in storage location j, the next instruction executed after it will be that in location $j + 1$ if the number currently in the accumulator is positive or zero; otherwise, it will be the instruction in location m. U m and T m are referred to as unconditional transfer and conditional transfer (or test) instructions, respectively.

The way has now been prepared to show how a large number of arithmetic and logical operations that may be required in the solution of a problem can be performed without the aid of outside intervention, provided the computer has certain elements and capabilities which have been described. As an example of automatic computation consider the program shown in Table 2.2. It is designed to find the highest factor of an integer, x. Each instruction is executed after its address, shown in column one, appears in the control register. The third column shows the contents of the accumulator after the execution of each instruction.

TABLE 2.2. Program for determining the highest factor of an integer, x.

Address	Instruction	Contents of Accumulator
001	cA 104	0
002	S 101	$-x$
003	A 102	$-x+jy_i$
004	T 003	$-x+jy_i$
005	C 103	$-x+jy_i$
006	cA 104	0
007	S 103	$x-jy_i$
008	T 010	$x-jy_i$
009	STOP	0
010	cA 102	y_i
011	S 105	y_i-1
012	C 102	y_i-1
013	U 001	y_i-1
	Numbers	
101	x	Constants and intermediate results are stored here.
102	y_i*	
103	—	
104	... 000	
105	... 001	

*The number stored here before the start of the program is the initial trial factor $y_0 = x - 1$.

An explanation of the program itself follows: The instructions of the program are placed in correct sequence in locations 001 through 013; 101 through 105 are reserved for storage of numbers. Location 101 holds the integer, x; 102 and 103 serve as temporary or working storage for numbers generated in the course of the program; 104 and 105 store the constants required by the program.

The instructions in locations 001 and 002 clear the accumulator and enter $-x$ into it. The instruction in location 003 produces $-x+y_i$. Each time instruction T 003 is obeyed, control is returned to location 003 until, after j cycles, the sign digit of the accumulator indicates that $-x+jy_i$ is either a positive number or zero.† When this occurs, the instruction T 003 advances control to location 005.

†In a system of numerical representation (see Section 6.1.4) in which the sign digit of zero is the same as that of a positive number, testing of the sign digit (which is the operation performed in a U m or T m instruction) will not distinguish between the two cases.

In order to detect whether $-x + jy_i$ is a positive number or zero, the quantity $-x + jy_i$ is converted by means of the instructions in locations 005, 006 and 007 into the quantity $x - jy_i$ (by storing $-x + jy_i$ in location 103 and then subtracting it from the cleared accumulator). If $-x + jy_i$ is a positive number the conversion results in a negative number, if it is zero the conversion has no effect. If y_i is a factor of x, $x - jy_i$ is zero. In this case, the instruction T 010 advances control to location 009 and completion of the program. If y_i is not a factor, the converted quantity $x - jy_i$ is not zero and T 010 advances control to location 010. The instructions in locations 010 through 012 produce a new trial factor $y_i - 1$ and store it in location 102. Instruction U 001 permits the whole sequence of instructions to be reiterated with $y_i - 1$ as the new trial factor.

All the integers from y_0 downwards will be tested until finally one is found that is a factor. When this occurs, the number in the accumulator will be zero when the test instruction T 010 is executed, so control will be advanced to the instruction in location 009. The STOP instruction puts the machine into an idle state, and the highest factor of x will be found in location 102.

The preceding example has shown how a computational problem may be solved completely without human intervention, provided certain specified conditions are met. Consideration of the example reveals the following important characteristics of computation with a stored program computer:

(1) The number of different types of instructions that the computer must be capable of executing need not be large.

(2) The number of instructions in the stored program is extremely small compared to the total number of instructions executed in the running of the program. This is possible because of the unconditional transfer instruction which may be used to provide recycling of a set of instructions.

(3) The conditional transfer, or test, instruction supplies the necessary means for breaking out of iterative loops at the correct point in the computation.

(4) The repertory of instructions the computer is capable of executing and the nature of its control unit (i.e., its rules of operation) are distinguishing features of a particular computer.

(5) The initial contents of the main storage unit, both instructions and numbers, are distinguishing features of a particular computation.

In the preceding description illustrating the operation of a stored program computer, it was assumed that both instructions and numbers were stored in a common storage unit. The reader may ask, then, how the computer can distinguish whether the contents of a particular storage

location represent a number or an instruction, since instructions are represented by numerical codes. The answer is that it cannot. If the control unit is directed to a specified storage location for the next instruction, the contents of that location will be interpreted as an instruction (even though it represents a number). Also, if the control unit is directed to a specified storage location for an operand, the contents of that location will be interpreted as a number (even though it represents an instruction). Such a situation does not, however, imply unavoidable confusion. The treatment of a number as an instruction may be avoided if the control unit is never directed to seek an instruction in a storage location not containing an instruction. The treatment of an instruction as a number may be avoided if the control unit is never directed to transfer to the arithmetic unit the contents of a storage location holding an instruction. However, while at first glance it may seem undesirable, the capability of operating on an instruction as a conventional number is actually an asset and contributes greatly to the utility of a stored program digital computer. This is because as a result of such operations one instruction may be converted to another. By this process, a computer can modify its own program and substitute new scquences of instructions for old ones, when required, during the course of a computation. This feature is valuable as a means of conserving storage.

If separate storage units were used for instructions and numbers, then the problem of possible misinterpretation of one as the other would not be present. Actually some of the earlier digital computers did have separate stores for instructions and numbers: for example, the Harvard Mark I computer. However, the advantages afforded by a common storage unit are so significant that such an arrangement is now common practice in digital computers. These advantages are first that it allows the use of a smaller total storage capacity since large variations in the relative amount of storage space allotted to instructions and numbers in different problems can be accommodated, provided the total storage requirement does not exceed the capacity of the computer. Secondly, the flexibility and efficiency of use are increased. As already indicated, when the program of instructions is stored in the same storage unit as numbers, they also may be transferred to the arithmetic unit. There they may be modified, under the control of other instructions, and used subsequently as systematically different instructions. Either or both the order and address codes of an instruction may be modified. In Section 7.5.5, there is a description of special devices that may be incorporated into the control unit to facilitate address modification. The example following illustrates an application in which the storage space required for a series of operations may be reduced if an address modification procedure is utilized.

A technique by which the addresses of certain designated instructions are modified as needed will be described with reference to a program for transferring the contents of one group of storage locations to another group. Assume that for some good reason it is desirable to transfer the contents of storage locations 401–425 to locations 451–475. This could be done without recourse to an address modification scheme by the program shown in Table 2.3.

TABLE 2.3. A program for relocating data in storage

Address	Instruction	Result
000	cA 401	Puts contents of 401 into accumulator
001	C 451	Duplicates contents of 401 (now in accumulator) in 451
002	cA 402	
003	C 452	
000		
.		
.		
.		
048	cA 425	
049	C 475	

Inspection of this simple program shows that there are only two distinct instructions; namely $cA(x_i)$ and $C(y_i)$, and each is repeated 25 times, with each address $x_i = 1 + x_{i-1}$, and each new address $y_i = 1 + y_{i-1}$.

An equivalent, but considerably shorter program can be obtained by storing only one pair of instructions, rather than 25, together with additional instructions that automatically change the addresses x_i, y_i, by 1 after each pair of instructions is executed. A program utilizing this address modification procedure is shown in Table 2.4.

Before continuing with an explanation of the program in Table 2.4, some prefatory remarks are in order. First of all, as a matter of convenience, different orders are designated by literal codes although within a machine they are represented by numerical codes. To understand the operation of this program, it is necessary to know that the order cA is, in this case, specified internally by the code 01. Also, as a rule, specific sections of the main store are reserved for the storage of constants and problem parameters. In this example, the storage locations for the constants are designated by the literal symbols *a*, *b*, rather than by numerical addresses. An important reason for the separation of instructions and numbers is that it prevents the accidental interpretation of a number as an instruction. This is because after starting at a specified point, the control

TABLE 2.4. A shorter program for relocating data in storage

Address	Instruction	
000	cA 401	
001	C 451	
002	cA *a*	
003	S 000	
004	T 012	(Exit to 012)
005	cA 000	
006	A *b*	
007	C 000	
008	cA 001	
009	A *b*	
010	C 001	
011	U 000	
012	Next instruction	
	Constants	
a	01 424	
b	00 001	

unit obtains its instructions from consecutively numbered storage locations. If a constant were stored within the main body of the program, say at location 008, then after the instruction in location 007 had been executed, the control unit would take the number from location 008 and interpret it as an instruction. The reason for the use of the literal symbols *a, b,* rather than specific storage location numbers is simply to indicate that the constants may be placed anywhere, so long as they are kept out of the main body of the program. Actually, the use of literal symbols to indicate addresses of the main body of the program as well as of constants and problem parameters is of considerable importance. First of all, it is useful to employ symbolic addresses in the initial preparation of relatively long programs because alterations may be made in one part of the program without the need for extensive changes of addresses throughout. For example, if actual addresses were used and a required instruction inadvertently omitted, then all addresses beyond that point, as well as all references to such addresses, would have to be altered. Of course, after the program has been adequately checked, and is ready to be inserted into the computer, actual addresses would be substituted for the symbolic addresses. Symbolic addresses are also of importance in the automatic

assembly of subroutines (defined in the closing paragraphs of this section) into working programs.

An explanation of the function of the different instructions in the program shown in Table 2.4 follows.

Address of Instruction	Effect of Instruction
000, 001	This part of the program has the same effect as the corresponding instructions in Table 2.3.
002, 003	Causes the instruction in storage location 000, i.e., the number 01 401 to be subtracted from the number 01 424 obtained from location *a*. (As a result a number $\geqslant 0$ will be in the accumulator at the time the instruction T 012 is executed, for the first 24 cycles of the iteration loop. On the twenty-fifth cycle, after all the required transfers have been performed, the number in the accumulator will be < 0 at the time T 012 is executed.
004	Causes control to be transferred to storage location 012 if the number in the accumulator is negative. This instruction provides the required exit from the program.
005, 006, 007	Adds 1 to the address of the instruction in location 000.
008, 009, 010	Adds 1 to the address of the instruction in location 001.
011	Returns control to the beginning of the program, enabling the sequence of operations to be repeated (with different addresses, in locations 000 and 001). After 25 such cycles the sequence will be automatically terminated by means of the instruction in location 004.

The program of Table 2.4 accomplishes the transfer of the 25 numbers with the storage of only 12 instructions and two constants, whereas the first program requires 50 words of storage. However, as often occurs, a reduction in storage requirements results in an increase in computation time. The program in Table 2.3 requires only 50 operations whereas that in Table 2.4 requires 293.

In practice, while frequently used operations such as addition and multiplication are built into a machine as instructions, more complex and less frequently used mathematical and logical functions are performed by means of special programs. (Each of these programs may be considered as a complex instruction.) When these functions are required in the course of solving a larger problem, they may be taken from a library of such programs and incorporated into the larger program. Programs for specific functions, which have already been designed and are available for incorporation into larger programs, are referred to as subprograms or

subroutines. The main program, the highest level of organization of a computer program, prescribes all operations not covered by a subroutine. Generally speaking, any sequence of instructions that a programmer finds convenient to treat as a sub-unit may be considered a subroutine.

Each time a subroutine, carefully planned to minimize storage requirements and execution time, is written, checked out, and made available for incorporation into a main program, the list of instructions the computer can execute is effectively augmented. The availability of a library of subroutines allows a programmer to utilize data processing operations not built into a computer's instruction repertory without having to write corresponding programs each time they are required. This greatly reduces the drudgery of program preparation—lessening both the time spent in program preparation and the probability of introducing errors. Also, since the use of subroutines allows the over-all program to be constructed from fewer blocks, the program becomes easier to comprehend and future modifications of it are simplified. Because the capacity of the main or high speed store is limited by cost and other practical engineering considerations, it is usually reserved for storage of the program to be executed while a complete library of subroutines is kept in an auxiliary store of lower speed and greater capacity.

For scientific and engineering computation, the most commonly used computational subroutines include those for extraction of square, cube, and higher order roots, and the solution of nth degree algebraic equations; generation of elementary functions—trigonometric, inverse trigonometric, hyperbolic, exponential and logarithmetic; interpolation; functional summation; matrix manipulation; integration of ordinary differential equations.

2.4. Program Preparation

The solution of a problem by means of a digital computer calls for the preparation and execution of a detailed plan of attack on the part of the person or persons responsible. The important items entering into such a plan are described below.

First of all, the problem must be analyzed and defined in detail. For scientific and engineering problems, this includes a statement of any simplifying assumptions or idealizations and results in an appropriate mathematical expression of the problem, usually in the form of one or more equations, together with any diagrams that may aid in clarifying the procedures to be used. Subsequently, the original mathematical expressions are replaced by appropriate explicit, finite, arithmetic and logical procedures adequate to provide the required degree of approximation to the exact solution. An important part of the analysis of the problem and its reduction to solution of numerical expressions, relates to the processes of

scaling and error analysis. The scaling of a problem depends on the range of magnitudes the problem variables can assume over the interval of interest. This range may either be estimated or, in some cases, compensated for by the computer itself. (See the discussion on scaling and on fixed- and floating-point operation in Chapter 6.) An error analysis is made to determine the accuracy that may be expected of the computer's answers. This accuracy will be influenced by two major sources of error, whose effect on the final answers must be estimated. They are truncation errors, which are introduced by the particular numerical approximation selected, and round-off errors introduced by the machine itself, as a result of the finite length of its registers.

For commercial problems of a bookkeeping or record keeping nature, such as preparation of financial statements, employee payroll deduction computations, or insurance premium billing operations, only precise quantities such as numbers and types of items, and dollars and cents are dealt with. As a result, the problems of error analysis associated with most scientific and engineering problems are not encountered. For these commercial applications, a detailed statement of the problem is usually made in English words accompanied by information flow diagrams. This description covers all pertinent procedures in the system and every eventuality that may be encountered.

After specific numerical procedures have been chosen, they must be translated into sequences of arithmetic and logical operations. This part of the process is referred to as programming. It consists of adapting the original problem definition to the capabilities of a computer. Preparation of the program calls for a thorough knowledge of the capabilities of the computer and its associated peripheral equipment. Since most problems to be solved by a digital computer require many sequences of arithmetic and logical operations, some type of mnemonic aid is called for. One that is commonly used is the so called problem flow diagram which as the name indicates shows the over-all flow of a problem. It provides organizational clarity, and indicates the general structure of a sequence of operations. It shows the location from which given quantities are obtained, where and how quantities to be generated are produced, where intermediate quantities are stored, and where output quantities are generated. It is useful, also, in showing how a complete program can be built up from simple processes for which programs have been worked out in the past and which are available from a subroutine library. The flow diagrams produced in the programming process differ from the diagrams used in the analysis in that they are intended for use with a particular computer and contain considerably more detail.

After an adequate flow diagram has been produced, the arithmetic and

logical operations indicated by it must be translated into a sequence of instructions that a particular computer is capable of executing. This process is referred to as coding and is the first part of the plan of attack discussed thus far that requires an exact, comprehensive knowledge of the particular machine to be used. Coding also includes such details as the specification of an arrangement for the storage of input information and intermediate results, and for the presentation of output information. The routine of coding can be minimized by the use of subroutines and various automatic coding techniques. Whenever subroutines are to be used, the main program must be designed in such a way that it allows for the inclusion of subroutines. The effect of automatic coding is that it allows the problem analyst's work to be brought to the machine in more or less general statements rather than in detailed step-by-step codes.

After a detailed sequence of instructions for the solution of a problem has been prepared, it must be inspected for mistakes that may have been inadvertently introduced. To assist in this inspection, a number of special mistake-hunting routines, usually referred to as debugging routines, have been developed.

After a program has been debugged, it is ready for running. In the event that the same program is run over and over, with variations in certain parameters, say, with different boundary conditions, the process is referred to as production running. After the program has been run, there remains the task of evaluating the results. This is an extensive subject in itself. and will not be treated here.

2.5. Program Flow Diagrams

It was stated in the preceding section that flow diagrams are useful in preparing a complete program for machine solution. For the sake of brevity, a flow diagram will be described for an operation that normally would be only a small part of an over-all program, namely a program for generating the square root of a given number N, where $0 \leqslant N < 1$. Since there are a number of numerical procedures by means of which the square root may be obtained, the first decision to be made is in regard to the choice of a particular method. Let us assume that because of its relative simplicity and rapid convergence, the iterative expression shown below is chosen

$$X_{i+1} = \frac{N}{2X_i} + \frac{X_i}{2}.$$

In this equation, N represents the number whose square root is to be obtained, and X_{i+1} the approximation to this root obtained after $(i + 1)$

iterations. The initial approximation X_0 may be obtained by providing a small table giving the value of the square root for a selected number of arguments. If such a table is not used, X_0 is assumed to be approximately equal to 1.0. The iteration is repeated until the difference $(X_{i+1} - X_i)$ becomes equal to or less than the precision required in the result.

In its most elementary form, a flow diagram may be composed simply of a series of boxes interconnected by directed line segments. Each box contains either a word or symbolic statement of an operation that is to be performed. It may include such items as a statement of an equation to be solved, a condition to be met, a check to be performed to determine whether an operation is legitimate or numerically accurate, the source of input data, the disposition of output data, etc. A useful preliminary procedure, at least for purposes of explanation, is to describe verbally the major operations that have to be performed. Such a description is shown below.

(1) Provide for the storage of problem parameters, constants, and intermediate quantities. For the square root program, these quantities include the constants N, 1, and ½, the intermediate quantities $N/2$, X_i, $NX_i/2$, and X_{i+1}.

(2) Provide for the execution of a sequence of arithmetic and logical operations adequate to obtain the desired result. For the square root program one such sequence is: (a) form the product ½ · N and store the result; (b) divide $N/2$ by x_i and store the result; (c) form the product ½ · X_i; (d) form X_{i+1} by combining the quantities generated in (b) and (c), and store; (e) test for convergence by subtracting X_i from X_{i+1}. (In Fig. 2.4 the notation $(i + 1) \rightarrow i$ means: use the value of X_{i+1} as the value of X_i in the next iteration).

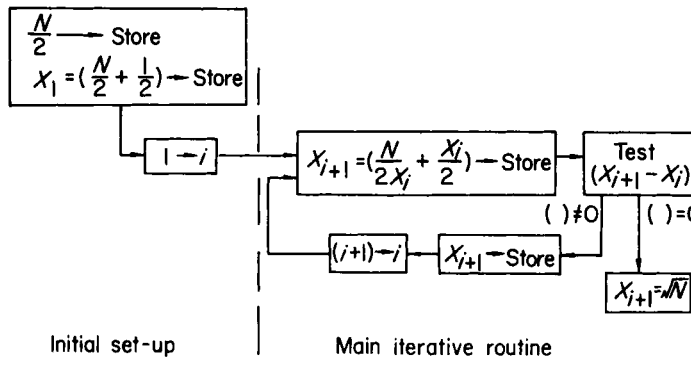


FIG. 2.4. Flow design for a square root program

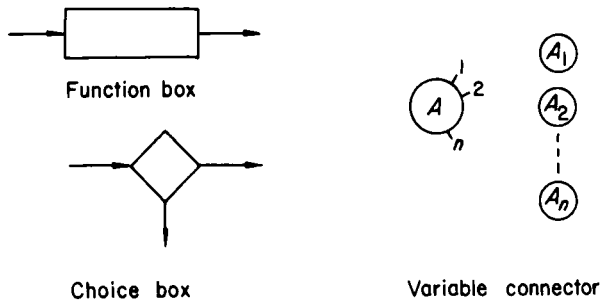


FIG. 2.5. A basic set of symbols for a flow diagram

A flow diagram indicating a square root program is shown in Fig. 2.4. The simple form of this diagram is adequate for the relatively simple problem illustrated. However, for more lengthy and complex problems, a better view of the program may be obtained by using a specialized set of symbols and notations. Figure 2.5 illustrates a basic set of symbols, adequate to describe any program. The function box will contain a verbal or mathematical statement of a particular function to be performed. The operations so indicated may be few or many in number and may or may not contain conditional operations. Regardless, there is only a single entry point and a single exit from the box. The choice box contains a question. Which of the two or more possible exit paths will be followed depends on the current result of computations which immediately precede and control the branching operation (as illustrated in Table 2.2 and 2.4). The variable connector symbols indicate to which of several addresses control can advance, as designated by the current address in a transfer of control instruction—this address being subject to modification during the running of the program.

In addition to the basic set of symbols described, a number of others may be provided to facilitate either the drawing or interpretation of a flow diagram. For example, remote points may be connected without lines by placing a circle containing the same symbol at the terminus of points to be connected. Special symbols may be used to indicate stopping points in the program, and the transfer of information into or out of the computer. Another highly useful device is a special form of box, termed an assertion box, in which annotations are placed explaining certain tricks or procedures used at various points in a program.

2.6. Automatic Sequencing Methods

Three distinct types of practical automatically sequenced digital computers have been developed, namely, machines in which the execution of instructions is controlled by 1) external devices (such as punched cards or tapes), 2) a plugboard connected to the internal circuits of the machine, or 3) an internally stored program. Since the subject of this book is the stored program computer, the other types will be mentioned only briefly.

In an externally programmed computer, instructions on punched cards or tapes, in a code meaningful to the computer and read under control of its internal circuits, direct the transfer of data into and out of sections of the arithmetic unit. The programming flexibility is poor compared to the stored program machine. Specifically, neither program iterations nor conditional transfer operations can be handled efficiently. For example, while one can resort to the cumbersome process of duplicating a set of instructions many times when the number of iterations required is known in advance, when it is not some other device must be employed. The simplest practical device is to prepare an endless loop of tape and have it read over and over until a number produced by the program being repeated indicates the process has been completed. However, if there are many loops this procedure, too, is uneconomical because of the many tape readers and the complexity of control that would be required. Finally, this type of computer lacks facilities for modification of instructions.

In a plugboard controlled machine, the sequence of operations is determined by the pattern of interconnecting jumper wires plugged into the board. More time is normally required to prepare a plugboard than a deck of punched cards. However, because there are many standard types of problems, especially in commercial applications, a removeable plugboard once wired for a particular program can be stored and available for future use. In practice, the plugboard wiring is manageable and the amount of equipment reasonable only when there are not more than a hundred or so steps in a given program sequence. A number of techniques for program iteration and conditional transfers of control are available in such machines.

3. Boolean Algebra

3.1. Introduction

The subject of Boolean algebra has important application in the design of systems composed of storage elements capable of assuming a discrete number of stable states and switching devices that trigger these elements from one stable state to another. An electronic digital computer is such a system. The utility of Boolean algebra in the design of digital equipment will be better appreciated after the discussion in this chapter of certain fundamentals, including a comparison of ordinary algebra and Boolean algebra, procedures for simplifying Boolean algebraic equations, and the basis for representing switching functions in binary computers by Boolean algebraic equations.

Algebra ordinarily refers to that branch of mathematics wherein quantitative relationships between entities are indicated by the use of numbers, letters (as symbols for the entities), and operational symbols (such as multiplication or addition signs). The rules of arithmetic are used in the solution of such algebraic equations.

The ways in which Boolean algebra differs from ordinary algebra are summarized below:

(1) There are no coefficients associated with the terms in a Boolean algebraic equation.

(2) Each letter designates which of two distinguishable events exists. As a matter of convenience, the value assigned to the letter upon the occurrence of one event is 0, and for the other it is 1.

(3) A Boolean algebraic function can only state whether one of two possible events exists, e.g., whether a circuit is open or closed, a signal present or not, a statement true or false, etc. The two possible values of the function are also usually designated by 0 and 1.

(4) There are a number of logical operators for producing Boolean algebraic functions. However, in general, they are not all used in the logical description of a digital computer for any Boolean function can be generated by the use of a proper subset of these operators. To date, the subset most commonly used includes the primitive operators referred to by the designation OR, AND, and NOT. If the two assigned values of a Boolean algebraic variable are interpreted as representing the numerical values of the binary number system, namely 0 and 1, then the three opera-

tors are analogous to addition, multiplication, and complementation, respectively, of binary elements. There is an exception* to the analogy in that the result of the OR (Boolean addition) operation on two or more 1's produces 1 as a result. These, as well as other Boolean operators will be described in the sections following.

(5) In Boolean algebra, there are no subtraction or division operators as in ordinary algebra, nor operators such as roots or transcendental operators. (Nevertheless, all the arithmetic operations of number algebra can be performed using only Boolean algebraic operators. Various means for accomplishing this are described in Chapter 6).

3.2. Logical Functions of Boolean Algebra

In this section three of the most common operations of Boolean algebra will be described. Later in the chapter other operations will be considered, also.

The "inclusive or" operation of Boolean algebra will be designated literally by OR and in equations by the symbol +. The Boolean equation representing the OR function, C, of two variables A, B is written as $C = (A + B)$. This expression means: C is true if A or B or both are true. An equivalent interpretation is: the truth of A or B or both implies, and is implied by, the truth of C. This relationship is defined in Table 3.1, which shows all possible combinations of values of A, B and defines the corresponding values for C. In Table 3.1 truth is indicated by 1 and falsity by 0. Reading across the table for all four possible cases, it is seen that C is true if, and only if, either A or B or both are true. Except for case (4), application of the OR operator produces the same result as the addition operator of ordinary algebra. This similarity accounts for the OR operator being sometimes referred to as the Boolean (or logical) addition operator, and also partly explains the choice of the symbol +.

TABLE 3.1. Truth table for the OR function C, of two variables: A, B.

Case	A	B	C
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	1

*An algebra can be formed from an arbitrary set of rules provided they are used systematically and do not invalidate each other.

The "and" operation of Boolean algebra will be designated literally by AND and in equations by placing the variables so to be operated upon adjacent to one another. For example, the AND function, C, of two variables A, B is written as $C = AB$. This expression means: C is true if, and only if, A and B are both true. An equivalent interpretation is: the truth of both A and B implies, and is implied by, the truth of C. This relationship is defined by Table 3.2. For all cases, application of the AND operator produces the same result as the multiplication operator of ordinary algebra. This similarity accounts for the AND operator being sometimes referred to as the Boolean (or logical) multiplication operator, and also partly explains the way chosen to represent it in equations.

TABLE 3.2. Truth table for the AND function, C, of two variables: A, B.

Case	A	B	C
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1

The complementation or negation operation of Boolean algebra will be designated literally by NOT and in equations by placing a bar over the variable or variables so to be operated upon. For example, the NOT function of a single variable, A, is written as $\bar{A}$. The symbol, $\bar{A}$, is read as "the complement of A" or "not A." Its meaning is defined by Table 3.3, which shows the relation between A and $\bar{A}$.

TABLE 3.3. Truth table for the NOT function, $\bar{A}$.

A	$\bar{A}$
0	1
1	0

3.3. Fundamentals of Boolean Algebra

The reader may verify, either through induction or the construction

of truth tables, that the commutative, associative, and distributive laws of number algebra apply to Boolean algebra.

For addition

$$A + B = B + A \qquad \text{Commutative}$$

$$A + (B + C) = (A + B) + C \qquad \text{Associative.}$$

One can demonstrate by means of a truth table, a proof of the associative law for three variables A, B, and C. The proof for other numbers of variables can be obtained by induction.

TABLE 3.4

	Column 1	2	3	4	5	6	7
	A	B	C	$(B+C)$	$A+(B+C)$	$(A+B)$	$(A+B)+C$
Case							
1	0	0	0	0	0	0	0
2	0	0	1	1	1	0	1
3	0	1	0	1	1	1	1
4	0	0	1	1	1	1	1
5	1	0	0	0	1	1	1
6	1	0	1	1	1	1	1
7	1	1	0	1	1	1	1
8	1	1	1	1	1	1	1

It is seen that column 5 is identical to column 7 for all possible combinations of values of A, B, and C.

For multiplication

$$AB = BA \qquad \text{Commutative}$$

$$A(BC) = (AB)C \qquad \text{Associative}$$

$$A(B + C) = AB + AC \qquad \text{Distributive.}$$

The reader may easily verify the above relations by means of a truth table.

Relationships useful in the manipulation of Boolean algebraic equations are listed in Eqs. (3-1)–(3-8). Some of these relationships will look incorrect if interpreted as ordinary numerical algebraic equations. The reader is warned, therefore, to be cognizant that these are Boolean algebraic equations, and to interpret their meaning accordingly.

Special cases of Boolean multiplication

$$AA = A \tag{3-1}$$

$$0A = 0 \tag{3-2}$$

$$1A = A \tag{3-3}$$

$$A\bar{A} = 0. \tag{3-4}$$

Special cases of Boolean addition

$$A + A = A \tag{3-5}$$

$$0 + A = A \tag{3-6}$$

$$1 + A = 1 \tag{3-7}$$

$$A + \bar{A} = 1. \tag{3-8}$$

An important theorem in Boolean algebra, referred to as the principle of dualization, states in effect

$$\text{if } A + B = 1 \tag{3-9}$$

$$\text{then } \bar{A}\bar{B} = 0. \tag{3-10}$$

It should be noted that Eq. (3-10) may be obtained from Eq. (3-9) merely by replacing each letter and/or truth value by its complement, and each addition operator by a multiplication operator.

In general

$$\text{if } A + B = C \tag{3-11}$$

$$\text{then } \bar{A}\bar{B} = \bar{C} \tag{3-12}$$

$$\text{and } A + B = \overline{(\bar{A}\bar{B})}. \tag{3-13}$$

A proof of Eq. (3-13) can be provided by means of a truth table

TABLE 3.5

Column	1	2	3	4	5	6	7
	A	B	$\bar{A}$	$\bar{B}$	$\bar{A}\bar{B}$	$\overline{(\bar{A}\bar{B})}$	$A+B$
Case							
1	0	0	1	1	1	0	0
2	0	1	1	0	0	1	1
3	1	0	0	1	0	1	1
4	1	1	0	0	0	1	1

It is seen that column 6 is identical to column 7 for all possible combinations of values of A and B.

A theorem due to De Morgan states that any Boolean function can be represented as a logical sum of logical products* (generally abbreviated, for convenience, to "a sum of products") each of which contains all input variables. A representation of this type is said to be in elemental form or disjunctive normal form. Equation (3-14) is an example.

$$y = ABC + \bar{A}B\bar{C} + A\bar{B}\bar{C} + \ldots \tag{3-14}$$

Such an expression may be obtained directly from a truth table by noting all combinations of input variables for which the output variable has a value of 1.

Equation (3-15) is obtained from Eq. (3-14) by the principle of dualization. It states that any Boolean equation can be written as a logical product of logical sums* (generally abbreviated, for convenience, to "a product of sums") each of which contains all input variables. A representation of this type is said to be in conjunctive normal form. Equation (3-15) is an example

$$\bar{y} = (\bar{A} + \bar{B} + \bar{C})(A + \bar{B} + C)\ldots \tag{3-15}$$

Some simple identities that can be obtained by means of the duality theorem are

$$\overline{(AB)} = \bar{A} + \bar{B} \tag{3-16}$$

$$\overline{(A + B)} = \bar{A}\,\bar{B} \tag{3-17}$$

$$\overline{(\bar{A})} = A. \tag{3-18}$$

Some tautologies† useful for simplifying elemental forms are listed below:

$$A + \bar{A} = 1 \tag{3-19}$$

$$\bar{A}A = 0 \tag{3-20}$$

$$A = A + A = A + A + A = \ldots \tag{3-21}$$

$$A = AA = AAA = \ldots \tag{3-22}$$

*Thus indicating that any Boolean equation can be written using the three operators, AND, OR, and NOT.

†A tautology is an equation that is true whatever be the truth values of the elementary propositions of which it is composed.

$$A(A + B) = A + AB = A \tag{3-23}$$

$$A + \bar{A}B = A + B \tag{3-24}$$

$$AB + \bar{A}C + BC = AB + \bar{A}C. \tag{3-25}$$

Equations (3-23), (3-24), and (3-25) merit special comment. Equation (3-23) may be factored as follows:

$$A + AB = A(1 + B) = A. \tag{3-26}$$

The validity of this expression is obvious upon recalling that $(1 + B) = 1$ and $1A = A$. Equation (3-23) is a special case of the identity:

$$A + f(A, B, C \ldots) = A + f(0, B, C, \ldots)$$

The validity of Eq. (3-24) may be demonstrated as follows. Considering the left side of Eq. (3-24)

$$A + \bar{A}B = A(1 + \bar{A} + B) + \bar{A}B \tag{3-27}$$

where the factor $(1 + \bar{A} + B) = 1$ (see Eq. (3-7)) is arbitrarily introduced to facilitate manipulation of the equation.

Expanding the expression on the right of Eq. (3-27)

$$A + \bar{A}B = A + A\bar{A} + AB + \bar{A}B. \tag{3-28}$$

Since $A = AA$, Eq. (3-28) may be written

$$A + \bar{A}B = AA + A\bar{A} + AB + \bar{A}B. \tag{3-29}$$

Factoring the right side of Eq. (3-29)

$$A + \bar{A}B = (A + \bar{A})(A + B). \tag{3-30}$$

Since $(A + \bar{A}) = 1$

$$A + \bar{A}B = 1(A + B) = A + B. \tag{3-31}$$

Equation (3-24) is a special case of the identity: $A + f(\bar{A}, B, C \ldots) = A + f(1, B, C \ldots)$, which can be generalized for functions involving three or more input variables to the form

$$g(A_i) + \overline{[g(A_i)]}h(B_j) = g(A_i) + h(B_j)$$

where A_i denotes a set of i variables and B_j a set of j variables. Any variable may appear in either set, and there is no limitation on the form of the functional relationships denoted by g and h.

The validity of Eq. (3-25) may be demonstrated as follows. Considering the right side of Eq. (3-25)

$$AB + \bar{A}C = (AB)1 + \bar{A}C. \tag{3-32}$$

Since $(1 + C) = 1$, Eq. (3-32) may be written

$$AB + \bar{A}C = AB(1 + C) + \bar{A}C. \tag{3-33}$$

Expanding the expression on the right side of (3-33)

$$AB + \bar{A}C = AB + ABC + \bar{A}C. \tag{3-34}$$

Factoring out C on the right side of Eq. (3-34)

$$AB + \bar{A}C = AB + C(AB + \bar{A}). \tag{3-35}$$

From Eq. (3-24), $(AB + \bar{A}) = \bar{A} + B$, so Eq. (3-35) may be written as

$$AB + \bar{A}C = AB + C(\bar{A} + B). \tag{3-36}$$

Finally, expanding the expression on the right side of Eq. (3-36)

$$AB + \bar{A}C = AB + C\bar{A} + CB \tag{3-37}$$

3.4. The Representation of Switching Functions by Boolean Equations

Now that certain fundamentals of Boolean algebra have been discussed, it is appropriate to state why this subject is of importance in the field of digital computer design. We recall from Chapter 1 that, because of practical difficulties in producing suitable multistable state electrical elements, all present electronic digital computers are composed principally of binary storage elements (with the exception of certain specialized components used to facilitate the data inputs to the computer and to display its outputs). A storage element must be capable of assuming different stable states, (e.g., voltage levels, states of magnetization, etc.) and of remaining for some specified time in the last state in which it was placed. Implicit in this statement is the assumption that each element is capable of being triggered or switched from one stable state to another. The signals used to trigger any particular circuit are determined according to certain criteria. For example, when an addition is being performed in an electromechanical computer, the condition that must be satisfied before a particular wheel is advanced by a notch, is that the less significant wheel (usually to its right) must pass from position 9 to position 0, i.e., produce a carry. As discussed in Chapter 1, the state of each element in an electronic digital computer is usually described by the voltage level at its output, and information is transmitted between elements by the routing of voltage signals. The output voltage states could be simply referred to as high or low, positive or negative, etc. However, for our purposes it is more

convenient and permissible to refer to a 1 state and a 0 state. Also, if some arbitrary symbol, say K_i, is assigned to the ith element, it is permissible to arbitrarily define one state of the element as the K_i state and the other as the $\bar{K}_i$ state. This arrangement is of great utility. First of all, it enables each binary element to be uniquely defined. Second, the current state of the elements of a machine can be described in terms of symbols rather than voltage levels. As a result, the condition or conditions for switching the state of any particular element can be specified and expressed in terms of the requisite coincident states of other elements. For example, if we wish element K_i to be triggered to the state $\bar{K}_i$ if and only if elements K_a and K_b are in the states K_a and K_b, respectively, then the required switching signal is $K_a K_b$. Therefore, if the switching signal input to the element, K_i is defined to be S_i, then $S_i = K_a K_b$. From this it becomes apparent that any switching signal requirement can be stated in terms of a Boolean algebraic function of the binary variables in the system. Such a Boolean algebraic expression will be referred to as a switching function. Since any function of binary variables has only two permissible values, it follows that any switching function has only two permissible values.

As stated earlier, any Boolean function can be formed by the use of the three Boolean operators, AND, OR, and NOT. Similarly, any switching function can be written utilizing only these three operators, and each operator can be considered as an elemental switching function. However, as stated earlier there are other Boolean operators, and at this point we will consider all the Boolean operators and switching functions for one and two variables.

The four possible switching functions of a single input variable, A, are described in Table 3.6. One may think of these switching functions as being represented physically by "black boxes," each having one input line to which either of two signals may be applied, and one output line on which either of two signals appear. The nature of the transformations produced by each box are shown in Table 3.6.

TABLE 3.6. Switching Functions, E_i, of a Single Input Variable, A.

Input, A	*Output, E_1*
0	0
1	0

Negation: $E_1 = 0$

Input, A	*Output, E_2*
0	0
1	1

Identity: $E_2 = A$

Input, A	*Output, E_3*
0	1
1	0

Complement: $E_3 = \bar{A}$

Input, A	*Output, E_4*
0	1
1	1

Tautology: $E_4 = 1$

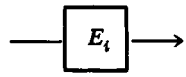


TABLE 3.7. Switching Functions, F_i, of Two Input Variables, (A, B).

A/B	0	1
0	0	0
1	0	0

Negation $F_1 = 0$

A/B	0	1
0	0	0
1	0	1

AND $F_9 = AB$

A/B	0	1
0	1	0
1	0	0

NOR $F_2 = \bar{A}\bar{B}$

A/B	0	1
0	1	0
1	0	1

Comparison $F_{10} = AB + \bar{A}\bar{B}$

A/B	0	1
0	0	1
1	0	0

Inhibiting gate $F_3 = \bar{A}B$

A/B	0	1
0	0	1
1	0	1

Single identity $F_{11} = B$

A/B	0	1
0	1	1
1	0	0

Single negation $F_4 = \bar{A}$

A/B	0	1
0	1	1
1	0	1

Conditional generator $F_{12} = \overline{(A\bar{B})}$

A/B	0	1
0	0	0
1	1	0

Inhibiting gate $F_5 = A\bar{B}$

A/B	0	1
0	0	0
1	1	1

Single identity $F_{13} = A$

A/B	0	1
0	1	0
1	1	0

Single negation $F_6 = \bar{B}$

A/B	0	1
0	1	0
1	1	1

Conditional generator $F_{14} = (\overline{A\bar{B}})$

A/B	0	1
0	0	1
1	1	0

Exclusive or $F_7 = A\bar{B} + \bar{A}B$

A/B	0	1
0	0	1
1	1	1

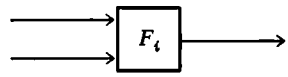
OR $F_{15} = A + B$

A/B	0	1
0	1	1
1	1	0

Sheffer stroke $F_8 = \overline{A\bar{B}}$

A/B	0	1
0	1	1
1	1	1

Tautology $F_{16} = 1$



Note that the functions E_1 and E_4 produce outputs independent of the input, and may be thought of as a 0 generator and a 1 generator, respectively. The function E_2 produces an output equal to the input. The only significant switching function of a single variable is E_3, which represents the complement operator.

The 16 possible switching functions of two input variables (A, B), are described in Table 3.7. It has already been stated that any Boolean algebraic equation can be expressed by the use of the AND, OR, and NOT

operators only. An analogous statement is that any switching function can be constructed utilizing these three switching functions. The functions F_9 and F_{15}, shown in Table 3.7, represent the AND and OR operators, while E_3, shown in Table 3.6, represents the NOT operator. Actually, AND, OR, and NOT do not represent a minimal set of independent operators. This can be proved by showing that the OR function can be generated by means of AND and NOT functions, and also that the AND function can be generated by means of OR and NOT functions. The proof of the first case is simply that $A + B = (\overline{\bar{A}\bar{B}})$. The proof of the second is that $AB = (\overline{\bar{A} + \bar{B}})$.

The functions F_4 and F_6 are independent of the values of B and A, respectively. Therefore, the lines on which each appears can be considered to be removed, in which case both F_4 and F_6 degenerate to a switching function of one variable, namely the NOT operator, E_3. F_1 and F_{12} are analogous to the 0 and 1 generators, E_1 and E_4, respectively; their outputs being completely independent of the inputs. F_{11} and F_{13} are independent of the values of A and B, respectively. Therefore, they are each equivalent to the single input switching function, E_2.

There are two functions that deserve special comment. They are the NOT–OR function, F_2, known as the NOR function, and the NOT–AND function, F_8, known as the $\overline{\text{AND}}$, NAND, or Sheffer stroke. The significant characteristic of each of these functions is that any Boolean equation or switching function can be constructed by the sole use of either of them. This places F_2 and F_8 in the category of universal switching functions. A simple proof of these statements is to show that all three primitive operations, namely AND, OR, and NOT can be obtained by the use of only the NOR or the $\overline{\text{AND}}$. Derivation of the primitive operations from NOR operators only is shown in Fig. 3.1. If only one of the input lines, say that for A,

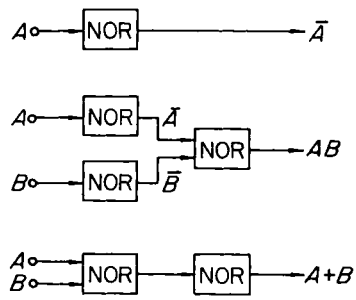


FIG. 3.1. Derivation of primitive logical operations by NOR operators only

is used, then $F_2 = A\bar{B}$ reduces to $F_2 = A$, and is equivalent to the "complement" operator, E_3. If two single-input F_2 operators are used to generate the complements of A and B and if A and B are each entered as inputs to a two-input F_2 operator, then the output is $F_2 = (\overline{A})\,(\overline{B}) = AB$. If A and B are entered as the inputs to an F_2 operator, the output will be $A\bar{B}$ which, if entered as the input to a single-input F_2 operator will yield $(A\bar{B}) = A + B$. Derivation of the primitive operations from the use of only $\overline{\text{AND}}$ operators is shown in Fig. 3.2. Note that the arrangements for

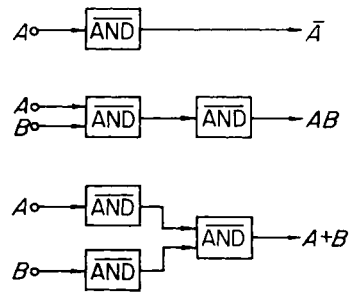


FIG. 3.2. Derivation of primitive logical operations by AND operators only

producing the AND and OR operations by means of $\overline{\text{AND}}$ functions is the same as that for producing the OR and AND operations, respectively, by means of NOR functions.

Only the functions F_3, F_5, F_7, F_{10}, F_{12}, and F_{14} remain to be considered. The functions $F_3 = \bar{A}B$ and $F_5 = A\bar{B}$ and their complements F_{14} and F_{12}, respectively, are of relatively little interest and may, along with F_{11} and F_{13} be considered relatively unimportant. The function F_7 is of special interest in that if A and B are binary variables, then F_7 represents the "exclusive or" function of these variables; also, if A and B represent the values of the individual bits of two numbers in binary form, F_7 produces the arithmetic sum (modulo 2). (For a description of binary addition by the use of logical operations, see Section 6.1.2.1.2).

3.5. Combinational Switching Networks

The term network is used to designate a group of switching functions integrated into a whole, and producing one or more required switching signals. Such networks are of fundamental importance in applications requiring the transmission of information signals. The most complex switching network in operation today is in the vast switching system used by the telephone companies to permit connection of any telephone to any

other telephone. Switching networks are widely used in a multitude of communication and control systems.

The importance of switching networks in digital computers derives from two reasons. First of all, a digital computer requires switching circuits to control the transfer of information from one section of the computer to another. Secondly, switching circuits may be used as arithmetic or logical operators that transform operands according to prescribed rules. A major part of a digital computer system as well as the bulk of other specialized switching systems is composed of so-called combinational switching networks. The term combinational is used to indicate that these networks are formed by combining the outputs of elementary switches, such as the ones described, and also to distinguish them from so-called sequential networks which consist of combinational networks into which storage elements have been incorporated. (Sequential networks are described later in this chapter.) A model of a combinational network is shown in Fig. 3.3. Each box represents some Boolean function of the

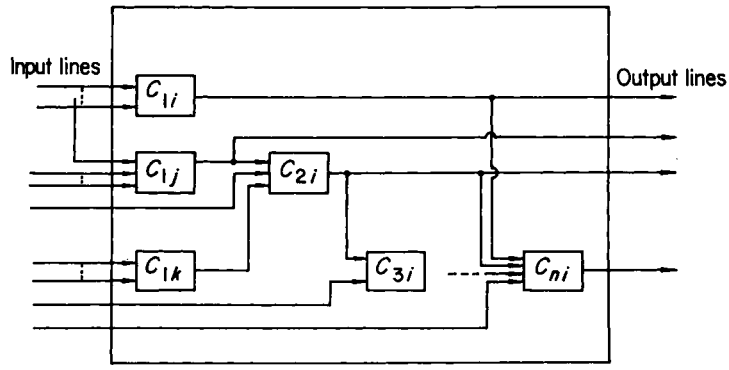


FIG. 3.3. Model of a combinational switching network

input variables. Each external input may be sent to one or more of the boxes, and the output of each box may have one or more destinations. A combinational network may consist of only a simple Boolean operator like an AND or an OR switch, or it may contain a large number of interrelated switches for generating several logical functions. A specific example of a simple combinational network is shown in Fig. 3.4.

When only the logical functions of a combinational switching network are considered, it is implicitly assumed that there is no delay from the time when input signals appear to when output signals are produced. In physically realizable networks this is not the case, for each switch introduces a small delay. However, the spacing between successive input

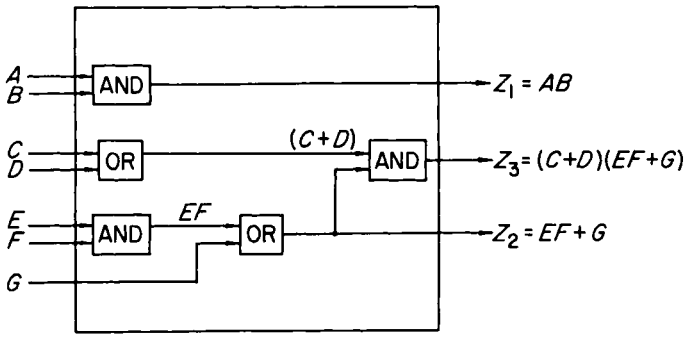


FIG. 3.4. A Combinational switching network

signals is such that the combinational function of the preceding set of input signals has been generated by the time a new set of input signals appears.

Even the most complicated switching networks may be formed in a straightforward manner. However, finding a network that must also meet other requirements, like minimizing the number of switching elements or maximizing the speed of operation, is another matter. Although a computer can be designed without recourse to Boolean algebra, its use affords certain conveniences. One of the most important of these is that a Boolean algebraic equation can be manipulated to yield equivalent functional forms. Then one can choose to mechanize that particular form which best satisfies certain specified physical requirements. The discussion following is intended to describe a number of techniques that are commonly used to simplify a Boolean equation or convert it to an equivalent form more desirable in the light of certain physical requirements.

3.5.1. REARRANGEMENT AND SIMPLIFICATION OF BOOLEAN EQUATIONS

When a term appears in the expression for a switching function which is superfluous, i.e., its removal does not alter values of the expression, that term is said to be redundant. A redundancy is often unintentional, appearing because its existence was not obvious in the original statement of the switching expression. One of the principal aims in rearranging Boolean expressions is to eliminate superfluous or redundant terms. However, they cannot always be removed by straightforward algebraic manipulation, e.g., see Eq. (3-25). Therefore, special devices must be employed for the detection and elimination of such terms. A number of methods useful for effecting rearrangement and simplification of Boolean equations will be described in this section.

*3.5.1.1. Trial and Error**

This method consists of finding simplifications by means of guesses based on experience and/or intuition, i.e., educated guesses, as to what device or procedure to employ. With experience, certain patterns will be recognized as containing superfluous terms, and therefore can be readily simplified. To aid this process, one may first try a number of regroupings of terms in the original expression so that simplifications may be made (by means of known tautologies) that were not apparent previous to the regrouping.

One of the devices that one can employ is to alter the form of an equation without altering its value by multiplying one or more terms by functions like $(X + \bar{X})$ or $(1 + X + Y + \ldots)$, or adding functions like $X\bar{X}$. Then simplifications may be produced by combining the extra terms generated with others in the original equation. Earlier in this chapter the right hand side of Eq. (3-25), $AB + \bar{A}C$, was manipulated to show its equivalence to $AB + \bar{A}C + BC$. The manipulation was begun by multiplying the first term of $AB + \bar{A}C$ by $1 + C$. Now we will start with the expression $AB + \bar{A}C + BC$ and show how it may be simplified by multiplying one of its terms by $A + \bar{A}$

$$\begin{aligned} f &= AB + \bar{A}C + BC \\ &= AB + \bar{A}C + BC(A + \bar{A}) \\ &= AB(1 + C) + \bar{A}C(1 + B) \\ &= AB + \bar{A}C. \end{aligned}$$

The introduction of a dummy factor like $A + \bar{A}$ in the example above may be considered as a special case of the general process of expanding terms in an expression. An example will make the point clear. If f is a function of four variables, A, B, C, and D, then the appearance of a three variable term like $A\bar{B}D$, for example, implies that the value of the term is independent of the value of C and therefore $A\bar{B}D = A\bar{B}CD + A\bar{B}\bar{C}D$. Similarly, if a two variable term like AB appears, it may be replaced by $AB\,(\bar{C}\bar{D} + \bar{C}D + C\bar{D} + CD)$. Note that if any term in the parenthesis is represented by X, the logical sum of the other three represents $\bar{X}$. As an example of what effect may be produced by expansion of a function, consider the expression $f = ABC + \bar{A}\bar{B}\bar{C} + A\bar{C}\bar{D} + \bar{A}CD$. If all

*Since it is difficult to handle any of the methods for a large number of variables, it may prove useful to attempt a quick trial and error simplification, first of all, for the purpose of assembling terms into groups which may be treated separately by any method applicable.

terms are expanded, one possible regrouping results in the expression $f = AB\bar{D} + \bar{A}\bar{B}D + BCD + \bar{B}\bar{C}D$. Note that, although logically equivalent, the two expressions have no common terms.

A simple procedure for detecting superfluous terms is as follows. First observe the values of the input variables for which the value of the term being tested is 1. Then, inspect other terms in the expression to see whether for the same values of the input variables, one or more of the other terms in the expression has the value 1. If so, either one, but not both, of the two terms is superfluous.

There are times when the form of an expression can be simplified by the deliberate introduction of redundant terms representing conditions which cannot exist physically or, if they could, would produce no detrimental effect on the function. As an example, consider the expression, $f = A\bar{B}C + A\bar{B}D + ACD + \bar{A}B\bar{D}$. If the signal AB cannot occur, which implies $AB \equiv 0$, then any product containing AB can be added to the expression for f without altering its value for any allowable values of the input variables. If the terms ABC, ABD, and $AB\bar{D}$ are added, there results

$$\begin{aligned} f &= (A\bar{B}C + ABC) + (A\bar{B}D + ABD) + ACD + (\bar{A}B\bar{D} + AB\bar{D}) \\ &= AC + AD + B\bar{D}. \end{aligned}$$

A shortcoming of the trial and error method is that, even though for an experienced person it may be the quickest, not all the simplest forms of an expression (when there are more than one) are likely to be obtained.

3.5.1.2. Converting a Boolean Sum of Products to a Product of Sums, or vice-versa

This technique leads to simplification if the equation as expressed contains more than half the possible functions of a given number of variables. Also, its use is dictated when one type of representation is preferable to the other, circuitwise.

A shorthand notation useful for converting a Boolean sum of products to a product of sums is as follows: Represent any product of n variables (where any variable may be either in its true or complemented form) by P_i, where i would be the binary number obtained by substituting a 1 or 0 for each variable, according to whether it is in its true or complemented state. For example, for a function of three variables, A, B, C

$$A\bar{B}C\text{: } i = 101, P_i = P_5 = A\bar{B}C$$

$$\bar{A}B\bar{C}\text{: } i = 010, P_i = P_2 = \bar{A}B\bar{C}.$$

A sum of variables is similarly represented by S_i. For example

$$(A + \bar{B} + C) = S_5$$

$$(\bar{A} + B + \bar{C}) = S_2.$$

All the P_i and S_i for three variables are shown in Table 3.8.

TABLE 3.8. Logical sums and products of three variables

i	P_i	S_i
000 = 0	$\bar{A}\bar{B}\bar{C} = P_0$	$\bar{A} + \bar{B} + \bar{C} = S_0$
001 = 1	$\bar{A}\bar{B}C = P_1$	$\bar{A} + \bar{B} + C = S_1$
010 = 2	$\bar{A}B\bar{C} = P_2$	$\bar{A} + B + \bar{C} = S_2$
011 = 3	$\bar{A}BC = P_3$	$\bar{A} + B + C = S_3$
100 = 4	$A\bar{B}\bar{C} = P_4$	$A + \bar{B} + \bar{C} = S_4$
101 = 5	$A\bar{B}C = P_5$	$A + \bar{B} + C = S_5$
110 = 6	$AB\bar{C} = P_6$	$A + B + \bar{C} = S_6$
111 = 7	$ABC = P_7$	$A + B + C = S_7$

Inspection of the table shows that in P_i, each variable has the complement of its value in S_{7-i}, i.e.

$$\bar{P}_i = S_{7-i}.$$

For n variables

$$\bar{P}_i = S_{(2^n-1)-i}. \tag{3-38}$$

There are 2^n products that can be formed from n binary variables. For each set of values the variables may assume, there is always one and only one product that has the value 1. Therefore

$$\sum_{i=0}^{(2^n-1)} P_i = 1. \tag{3-39}$$

For $n = 1$

$$A + \bar{A} = 1.$$

For $n = 2$

$$(A + \bar{A})(B + \bar{B}) = 1$$

$$\bar{A}\bar{B} + \bar{A}B + A\bar{B} + AB = 1$$

etc.

It is also true that

$$\prod_{i=0}^{(2^n-1)} S_i = 0. \tag{3-40}$$

The validity of this expression may be seen by taking the complement of

$$\sum_{i=0}^{(2^n-1)} P_i = 1.$$

For example, for $n = 2$

$$f = \bar{A}\bar{B} + \bar{A}B + A\bar{B} + AB = 1$$
$$\bar{f} = (A + B)(A + \bar{B})(\bar{A} + B)(\bar{A} + \bar{B}) = 0.$$

By the use of Eq. (3-39) and the fact that $f + \bar{f} = 1$, a complementary expression for a sum of products may be written as the sum of all products not appearing in the original expression, thus preserving the sum of products form. This latter expression can then be complemented using the relationship of Eq. (3-38) to yield a product of sums form of the original expression. This is illustrated in the following example

$$\begin{aligned} f &= ABC + \bar{A}BC + A\bar{B}C + \bar{A}\bar{B}C + A\bar{B}\bar{C} \\ &= P_7 + P_3 + P_5 + P_1 + P_4 \\ \bar{f} &= P_0 + P_2 + P_6 \\ f &= S_7 S_5 S_1 \\ &= (A + B + C)(A + \bar{B} + C)(\bar{A} + \bar{B} + C) \\ &= (A + B + C)(\bar{B} + C) = A\bar{B} + \bar{B}C + AC + BC + C \\ &= A\bar{B} + C. \end{aligned}$$

By the use of Eq. (3-40) and the fact that $f\bar{f} = 0$, a complementary expression for a product of sums may be written as the product of all sums not appearing in the original expression. This latter expression can then be complemented using the relationship of Eq. (3-38) to yield a sum of products form of the original expression. This is illustrated in the following example

$$f = (A + B + C)(A + \bar{B} + C)(\bar{A} + \bar{B} + \bar{C})(\bar{A} + B + \bar{C})$$
$$(\bar{A} + \bar{B} + C)$$
$$= S_7 S_5 S_4 S_2 S_1$$
$$f = S_0 S_3 S_6$$
$$f = P_7 + P_4 + P_1$$
$$= ABC + A\bar{B}\bar{C} + \bar{A}\bar{B}C.$$

The conversion of a sum of products to a product of sums may also be facilitated by use of the identity: $A + BC = (A + B)(A + C)$. This relationship, sometimes referred to as the second distributive law of Boolean algebra, may be derived from the expression of the first distributive law stated on page 41 by application of the duality principle. Its general form is $(X + Y_1 Y_2 \ldots Y_n) = (X + Y_1)(X + Y_2) \ldots (X + Y_n)$. (Note that the first distributive law of Boolean algebra applies to number algebra also, but the second does not).

Theoretically there is a one-to-one correspondence between the set of all possible functions of each type, so there is an equal probability of obtaining a simpler or more complex function after transformation. In actual practice, it is doubtful whether all functions are equally probable (i.e., that there is a random distribution). An example of a sum of products which yields a more complex form after conversion is the expression on the left in Eq. (3-41), and one that yields a simpler form is the expression on the left in Eq. (3-42).

$$AB + CD = (A + CD)(B + CD)$$
$$= (A + C)(A + D)(B + C)(B + D) \quad (3\text{-}41)$$

$$AC + AD + BC + BD = A(C + D) + B(C + D)$$
$$= (A + B)(C + D). \quad (3\text{-}42)$$

If one performs a double conversion, i.e., first converts a sum of products to a product of sums and then reconverts to a sum of products, the final expression obtained may have superfluous terms not removable by simple algebraic manipulation. As an example, consider the expression

$$f = \bar{A}B + \bar{B}\bar{C} + ABC.$$

This expression can be converted to a product of sums by repeated application of the identity $(X + Y_1 Y_2 \ldots Y_n = (X + Y_1)(X + Y_2) \ldots (X + Y_n)$:

$$
\begin{aligned}
f &= (\bar{A}B + \bar{B})(\bar{A}B + \bar{C}) + ABC \\
&= [A + (\bar{A} + \bar{B})(\bar{A}B + \bar{C})][B + (\bar{A} + \bar{B})(\bar{A}B + \bar{C})] \\
&\quad [C + (\bar{A} + \bar{B})(\bar{A}B + \bar{C})] \\
&= [A + (\bar{A} + \bar{B})][A + (\bar{A}B + \bar{C})] \\
&\quad [B + (\bar{A} + \bar{B})][B + (\bar{A}B + \bar{C})] \\
&\quad [C + (\bar{A} + \bar{B})][C + (\bar{A}B + \bar{C})] \\
&= [A + (\bar{C} + \bar{A})(\bar{C} + B)][B + (\bar{C} + \bar{A})(\bar{C} + B)][C + \bar{A} + \bar{B}] \\
&\quad [C + (\bar{C} + \bar{A})(\bar{C} + B)] \\
&= [A + (\bar{C} + \bar{A})][A + (\bar{C} + B)][B + (\bar{C} + \bar{A})][B + (\bar{C} + B)] \\
&\quad [C + \bar{A} + \bar{B}][C + (\bar{C} + \bar{A})][C + \bar{C} + B] \\
&= (B + \bar{C})(C + \bar{A} + \bar{B}).
\end{aligned}
$$

The final expression may be converted to a sum of products by multiplying the factors yielding

$$f = BC + \bar{B}\bar{C} + B\bar{A} + \bar{C}\bar{A}.$$

Either the third or fourth term (but not both) of this expression is superfluous. The third term may be eliminated by expanding $B\bar{A}$, i.e. replacing it by $B\bar{A}(C + \bar{C})$ and then factoring the resultant expression

$$f = BC + \bar{B}\bar{C} + BC\bar{A} + B\bar{C}\bar{A} + C\bar{A}$$

$$f = BC + \bar{B}\bar{C} + \bar{C}\bar{A}.$$

An identity that is sometimes useful for simplifying expressions obtained at intermediate steps in a conversion process is

$$F(X_1,X_2, \ldots X_m)\,[F(X_1,X_2, \ldots X_m) + G(X_1,X_2, \ldots X_n)] = F(X_1,X_2, \ldots X_m).$$

The validity of this expression becomes apparent if one multiplies the terms on the left and factors the result.

For different reasons, e.g., either to maintain uniformity of subassemblies, or because certain types of components function better in one type of circuit, it may be desirable to standardize, and use one type of network arrangement exclusively. If the type most suitable because of the characteristics of the components should require more components than the other type, this may be avoided, (according to the design of a particular machine) by redefining the presence and absence of a signal, and thereby interchanging the AND and OR functions. The majority of practical switching functions is less complex in the sum of products form, and visualization of the operations involved is usually easier in this form.

If there is no rigid requirement for a circuit to be of the pure sum

of products or product of sums form, a miscellaneous form of network may offer the most desirable solution. A miscellaneous form implies a multiple level switching network (described in Chapter 4).

The derivation of a simplified equivalent expression of a switching function by trial and error and algebraic manipulation can be quite difficult, even though its equivalence to the original expression can be readily shown once it has been derived. A principal advantage of the chart method described in the next section is that even though it is quite tedious it enables all equivalent forms to be found by a routine process.

3.5.1.3. Chart Methods

(a) THE HARVARD METHOD (See Staff of the Computation Laboratory [1951]):

The advantage of this method is that it gives all possible simplified expressions automatically. It consists of laying out a chart which contains for n variables, all products of length n or less (where a variable may or may not be complemented in each product). Table 3.9 is a chart for $n = 3$.

TABLE 3.9

ABC	*AB*	*AC*	*BC*
000	00	00	00
001	00	01	01
010	01	00	10
011	01	01	11
100	10	10	00
101	10	11	01
110	11	10	10
111	11	11	11

The form of the chart can be simplified by considering the functions as binary numbers and then replacing them with their decimal equivalents, as shown in Table 3.10.

The procedure is as follows:

(1) If the equation to be simplified is not in the form of a sum of products convert it to this form. Then, draw horizontal lines through those rows in which the products $ABC \ldots$ should be zero for this equation.

(2) Cross out wherever else they occur in a given column those numbers that were crossed out by the horizontal lines in step 1.

(3) In general, one may find that some row has only one combination

TABLE 3.10

ABC	*AB*	*AC*	*BC*
0	0	0	0
1	0	1	1
2	1	0	2
3	1	1	3
4	2	2	0
5	2	3	1
6	3	2	2
7	3	3	3

of a minimum number of variables that has not been crossed out. Such a combination is called essential and is circled together with all its appearances in a given column.

(4) Rows may still remain with neither horizontal lines drawn through them nor encircled elements. Each of these rows will contain two or more unmarked combinations of a minimum number of variables. Circle at least one arbitrary combination in each nondeleted row and encircle all occurrences per column of each arbitrary combination in a way that minimizes the number of combinations encircled.

(5) A minimal expression of the original equation is given by the Boolean sum of all encircled combinations. As an example, consider the expression

$$f = ABC + \bar{A}B + \bar{B}\bar{C}$$

ABC	*AB*	*AC*	*BC*
0	0	0	0
1	0	1	1
2	1	0	2
3	1	1	3
4	2	2	0
5	2	3	1
6	3	2	2
7	3	3	3

Steps 1 and 2 need no comment. In step 3, one finds that there are only

two rows (row 5 and 8) containing essential elements; these are 0 and 3 in column BC. In step 4, one finds the only nondeleted row with a non-encircled element to be row 3. One then has a choice of encircling either element 1 in column AB, or element 0 in column AC. In step 5, one of two minimal expressions is obtained for f, depending on the choice made in step 4. If the element in column AB were chosen, one obtains

$$f_1 = BC + \bar{A}B + \bar{B}\bar{C}.$$

The other solution is

$$f_2 = BC + \bar{A}\bar{C} + \bar{B}\bar{C}.$$

The equivalence of f_1 and f_2 to f and, hence, to each other may be shown as follows

$$\begin{aligned} f &= ABC + \bar{A}B + \bar{B}\bar{C} \\ &= B(AC + \bar{A}) + \bar{B}\bar{C} \\ &= BC + \bar{A}B + \bar{B}\bar{C} \\ f &= ABC + \bar{A}B + \bar{B}\bar{C} \\ &= ABC + \bar{A}BC + \bar{A}B\bar{C} + \bar{B}\bar{C} \\ &= BC(A + \bar{A}) + \bar{C}(\bar{A}B + \bar{B}) \\ &= BC + \bar{A}\bar{C} + \bar{B}\bar{C} \end{aligned}$$

As the number of variables increases, simplification by algebraic manipulation becomes more difficult. However, the chart methods can be utilized, and if the number of variables makes the process too cumbersome for manual execution, it can be programmed (within limits), for a digital computer.

(b) THE QUINE SIMPLIFICATION [1952, 1955]:

This method is similar to the Harvard method. It differs in that the chart is constructed for a specific case, and contains only the pertinent terms. Most of the advantages and disadvantages of the Harvard method apply to this method also, except that for a large number of variables it is not as cumbersome. The procedure is as follows:

(1) Expand all terms to include all variables, e.g., if there are three input variables and the term AB appears, replace it with ABC, $AB\bar{C}$. Eliminate duplicates of terms that may appear.

(2) Starting with the list of terms formed in step (1), add additional entries as follows. Whenever two entries in the original list differ by only one variable, e.g., $\bar{A}BC$, $\bar{A}B\bar{C}$, enter the similar part of the terms

on the list (in this case $\bar{A}B$), and enter checks opposite the two terms from which it was obtained. This process is continued until a group of unchecked terms remains which cannot be reduced by further combination. The unchecked terms, i.e., those never combined with any other terms, constitute a set of prime implicants. (A prime implicant is a logical product which is a term of every minimal form of a Boolean function).

(3) List the expanded terms of step (1), and the prime implicants of step (2) as column and row headings, respectively, of a table, and place marks in each row in those columns where the expanded term is implied by the prime implicant.

(4) If any column has only one mark, the corresponding prime implicant is essential and is to be included in the result. This column and all other columns included in the same prime implicants are eliminated.

(5) Whenever in the remaining table there are two columns such that each has marks only in rows where the other has marks, one of these columns may be eliminated.

(6) The remaining columns are examined to determine how they may be covered with the fewest prime implicants.

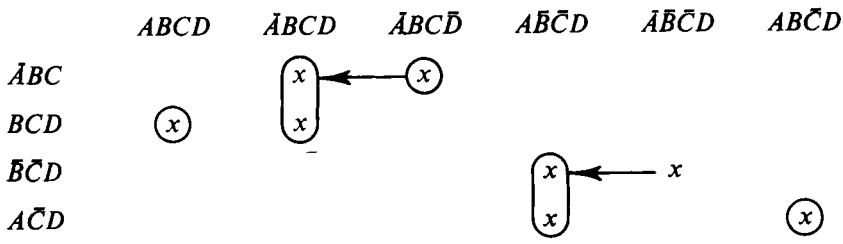
As an example, consider the expression

$$f = ABCD + \bar{A}BC + \bar{B}\bar{C}D + A\bar{C}D$$

Step 1: Step 2:

$ABCD$	$ABCD\surd$	$\bar{A}BC$
$\bar{A}BCD$	$\bar{A}BCD\surd$	BCD
$\bar{A}BC\bar{D}$	$\bar{A}BC\bar{D}\surd$	$\bar{B}\bar{C}D$
$A\bar{B}\bar{C}D$	$A\bar{B}\bar{C}D\surd$	$A\bar{C}D$
$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}\bar{C}D\surd$	
$AB\bar{C}D$	$AB\bar{C}D\surd$	
~~$A\bar{B}\bar{C}\bar{D}$~~		

Steps 3, 4, 5:



$$f = \bar{A}BC + BCD + \bar{B}\bar{C}D + A\bar{C}D$$

The result may be verified algebraically

$$f = ABCD + \bar{A}BC + \bar{B}\bar{C}D + A\bar{C}D$$
$$= BC(AD + \bar{A}) + \bar{B}\bar{C}D + A\bar{C}D$$
$$= BCD + \bar{A}BC + \bar{B}\bar{C}D + A\bar{C}D.$$

A systematic way of transforming a Boolean function to yield all minimal forms which are sums of products allows one to be chosen to replace a given switching function. This may be an improvement over the original expression even though the latter contains no superfluous terms. The procedure will now be summarized. First, the function is expanded to its elemental form, and by applications of the identity, $AB + A\bar{B} = A$, all basic terms (i.e., any correct term containing no superfluous variables) in the expression may be found. Then, a table is made indicating which of the basic terms are contained in each term in the elemental form. The terms that are basic are not known until each of them has been compared with all others, and further reductions are not possible. The use of the table enables all possible combinations of basic terms equivalent to the original expression to be found.

The simplest product of sums expression may also be obtained from this procedure. First, the original sum of products expression, f, is replaced by a complementary sum of products function, $\bar{f}$, utilizing Eq. (3-39) and the identity $f + \bar{f} = 1$. Then the complementary function is reduced by the process described to yield the simplest sum of products expression for $\bar{f}$. Complementing the expression for $\bar{f}$ yields the simplest product of sums expression of the original function, f.

A systematic procedure for analyzing switching functions may be useful even in the event that an algebraic simplification cannot be achieved, for an alternate expression may be found that would have been difficult to find otherwise. This alternate expression may be more desirable circuitwise, e.g., it might not put as great a load on a circuit already loaded heavily.

3.5.1.4. Map Methods

The Harvard method, previously described, consists of generating a list of possible simplifications followed by a choice between these possibilities. The method to be described here consists of presenting the function in a form in which possible simplifications are made more apparent, thereby reducing appreciably the routine work of the chart methods. An early form of the type of map to be considered here, proposed by Veitch,

[1952] is shown in Fig. 3.5. Each map provides a square for entry of any

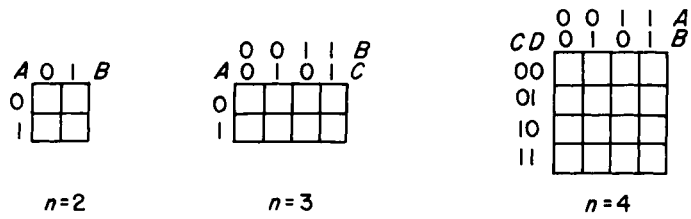


FIG. 3.5. Veitch maps

one of the 2^n logical products of n variables (which compares favorably with the 2^{2n} entries used in the Harvard method). The product designated by a particular square is obtained by noting the values of the variables in the column and row that intersects the square. A reorganization of the Veitch maps, proposed by Karnaugh, [1953] is shown in Fig. 3.6. The

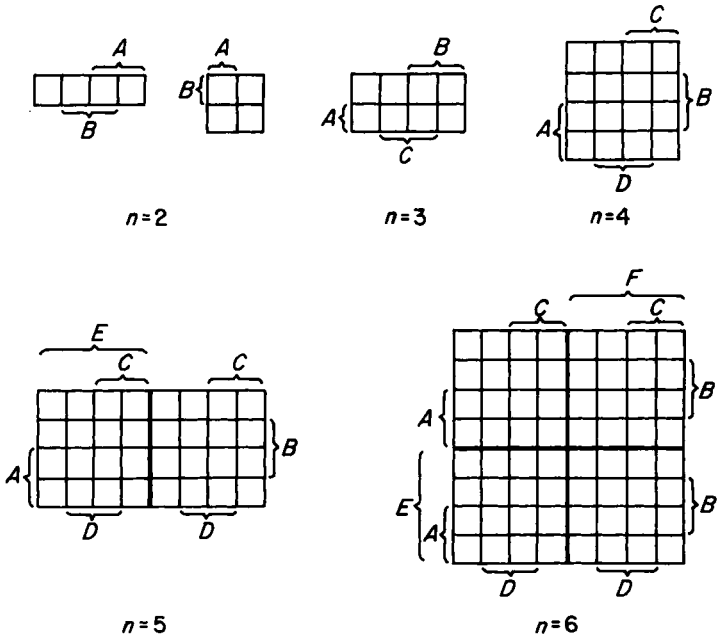


FIG. 3.6. Karnaugh maps

rows or columns within a bracket are those in which the designated variable has the value 1, while it is 0 elsewhere. Adjacent squares are defined as those that differ in the value of only one variable, so that squares at the opposite ends of a row or column are considered adjacent. Because the Karnaugh map is generally more convenient, it will be used in the examples that follow.

Use of the map method requires that the function to be simplified be represented first in its elemental form, i.e., as a logical sum of products. For each elemental term in the function, a mark such as a check or cross is placed in the square corresponding to that particular product of n variables. Then the map is inspected for the purpose of recognizing which of several possible groupings of terms represents the best factoring of terms in the function. It is desirable to choose these groups so that each encompasses as many positions as possible. Each checked square must be represented by at least one of the groups, though it may be included in two or more. The usefulness of the map derives from the fact that patterns of checks which will yield the simplest terms can, after sufficient practice, be easily and quickly determined by inspection. Some typical patterns that may be encountered in a four variable map are shown in Fig. 3.7.

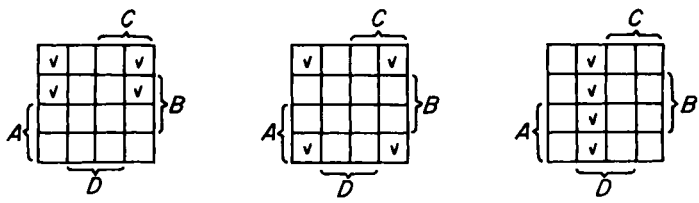


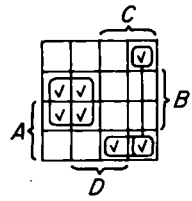
FIG. 3.7. Typical patterns in a map

$f = \bar{A}\bar{D}$ $\qquad$ $f = B\bar{D}$ $\qquad$ $f = \bar{C}D$

As a matter of definition, a group is the map of a logical product formed according to the following rule: the factors of the product are those variables whose values are fixed within the group, whether in the uncomplemented or complemented condition. Larger groups correspond to products of fewer variables, since fewer variables are fixed in them. To obtain a minimal expression, one chooses a set of groups which includes every checked square at least once. In general, it is desirable to make the selected groups as large (for less terms per product) and as few (for less products) as possible.

As a first e xample, consider the expression $f = \bar{A}B\bar{C} + AB\bar{C} + A\bar{B}C$

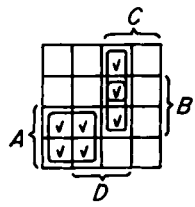
$+ \bar{A}B\bar{C}\bar{D}$. Each of the first three terms in the expression requires the entry of two checks (one for D and one for $\bar{D}$) while the last term requires only one. In Fig. 3.8, a particular grouping of these checks is shown which yields the simplified expression $f = B\bar{C} + A\bar{B}C + \bar{B}C\bar{D}$.

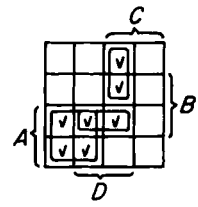


$$f = B\bar{C} + A\bar{B}C + \bar{B}C\bar{D}$$

FIG. 3.8. Derivation of an expression from appropriate groupings of checked squares

In Fig. 3.9 two alternate groupings of the terms in a given expression





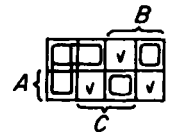
$$f_1 = A\bar{C} + \bar{A}CD + BCD$$
$$= A\bar{C} + CD(\bar{A} + B)$$

$$f_2 = A\bar{C} + \bar{A}CD + ABD$$
$$= A\bar{C} + D(\bar{A}C + AB)$$

FIG. 3.9. Alternate expressions based on different groupings of a function

are formed, resulting in different but equivalent expressions, f_1 and f_2.

This method is also useful in finding the simplest product of sums expression equivalent to a given sum of products. The map procedure is analogous to the procedure wherein one first replaces the original sum of products by the complementary sum of products, finds a simplest form of the latter, and complements it to yield a product of sums representation of the original function. As an example, consider the function $f = AB\bar{C} + \bar{A}BC + A\bar{B}C$. The complementary sum of products is: $\bar{f} = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$. By factoring, the latter expression may be reduced to $\bar{f} = \bar{B}\bar{C} + \bar{A}\bar{B} + \bar{A}\bar{C} + ABC$. Taking the complement of this expression yields: $f = (B + C)(A + B)(A + C)(\bar{A} + \bar{B} + \bar{C})$. When using the map method, (see Fig. 3.10), a simplified expression for the



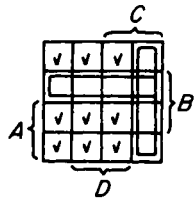
$$f = B\bar{C} + \bar{A}B + \bar{A}\bar{C} + ABC$$

$$f = (B + C)(A + B)(A + \bar{C})(\bar{A} + \bar{B} + \bar{C})$$

FIG. 3.10. Obtaining a simplified product-of-sums expression by groupings of unchecked squares

complementary function $\bar{f}$ is obtained by forming appropriate groupings of unchecked squares. Taking the complement of this expression then yields a simplified product of sums expression.

In some cases it may be more convenient to derive the simplest sum of products expression by means of an intermediate product of sums expression. As an example consider the function $f = AD + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}CD$. The map of this function is shown in Fig. 3.11. By forming



$$\bar{f} = \bar{A}B + C\bar{D}$$

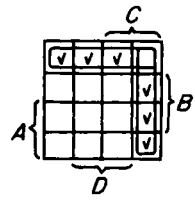
$$f = (A + \bar{B})(\bar{C} + D)$$

$$= AD + A\bar{C} + \bar{B}\bar{C} + \bar{B}D$$

FIG. 3.11. An expression based on the complement of groupings of unchecked squares

a particular set of groupings of the checked squares, the expression $f = AD + A\bar{C} + \bar{B}\bar{C} + \bar{B}D$ may be obtained. However, a simpler grouping may be formed by combining the unchecked squares into the two groups indicated in Fig. 3.11. This grouping yields $\bar{f} = \bar{A}B + C\bar{D}$. If this expression is then complemented, there results $f = (A + \bar{B})(\bar{C} + D)$, which is a factored form of $f = AD + A\bar{C} + \bar{B}\bar{C} + \bar{B}D$.

Figure 3.12 illustrates a "combination" solution; i.e., one obtained by considering both checked and unchecked squares. In order to reduce the number of groups, the square corresponding to $\bar{A}\bar{B}C\bar{D}$ is first assumed

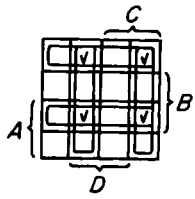


$$f = (\bar{A}\bar{B} + C\bar{D})(\bar{A}BC\bar{D}) = (\bar{A}\bar{B} + C\bar{D})(A + B + \bar{C} + D)$$

$$f = \bar{A}\bar{B}(\bar{C} + D) + C\bar{D}(A + B)$$

FIG. 3.12. An expression based on groups containing checked and unchecked squares

to be checked, and later this state is inhibited. After some practice, one can immediately, by inspection of the map, write a reduced expression such as the one shown in Fig. 3.13. Here the checked squares are con-



$$f = (\bar{A}\bar{B} + AB)(\bar{C}D + C\bar{D})$$

FIG. 3.13. An expression based on the intersection of four groups

sidered as a group defined by the intersection of four other groups, each containing both checked and unchecked squares.

An important advantage of the map and the Harvard chart method is the convenience it provides for taking advantage of nonallowable or indifferent combinations of the input variables. These combinations, referred to as redundant, may occur because the particular combination never is realized in practice or because it has no undesirable effect on the output. In searching for the simplest expression, one may or may not include redundant combinations in a group. As a rule, those redundancies are included which enlarge and combine the necessary groups as much as possible, but do not necessitate the selection of additional groups.

In conclusion it should be stated that often a solution as good or better than any map or chart solution may be obtained by other means. For example, the expression obtained from the map shown in Fig. 3.8 could have been obtained very easily by factoring

$$f = \bar{A}B\bar{C} + AB\bar{C} + \bar{A}\bar{B}C\bar{D} + A\bar{B}C$$
$$= B\bar{C}(\bar{A} + A) + \bar{B}C(\bar{A}\bar{D} + A)$$
$$= B\bar{C} + \bar{B}C(\bar{D} + A)$$

There are times, however, when the use of a map facilitates derivation of a simplified form of a Boolean function. As with chart methods, the process becomes more cumbersome as the number of variables increases. (For the special case of symmetric circuits the reader is referred to Caldwell [1954], Lee [1954], Slepian [1953], and Washburn [1949]).

In the preceding discussion of procedures for simplifying Boolean equations, two important items were neglected. First, there was no consideration of the problem of deriving a form of an equation corresponding to the combinational circuit most desirable from a physical standpoint. There are effects peculiar to different types of components and circuits which do not show up in consideration of the equations alone, but which place restrictions on logical formulations. The nature of the restrictions placed on logical formulations by the available circuitry, as well as the circuit implications of different logical formulations, will be considered in Chapter 4. The second item neglected was the subject of how digital computers may be used to simplify Boolean descriptions of new machines. This will be discussed in Chapter 7.

3.6. The Storage Function

In Section 3.4 there was reference to the fact that input signals to a combinational switching network cause the network to generate appropriate output signals after a short and unavoidable transit time. In other words, there is a free flow of signals between input and output without any significant delays or storage. Such networks operate on binary inputs but do not store either the inputs themselves or any transformation of them. They can only represent on their output lines some function or functions of the variables currently present on their input lines. By the incorporation of storage elements, inputs received by a combinational network at one time can be stored for combination with inputs received at other specified times.

In general, the concept of memory or storage implies the receipt of information at some time, t, and the retention of that information until some later time, $t + \Delta t$. There are basically two ways in which information may be retained — referred to as static and dynamic storage. There are many possible physical realizations of both types of storage, and

descriptions of a number of them are provided in Chapters 4 and 5. Because of considerations of reliability, discussed in Chapter 1, the internal storage elements in a digital computer are used as binary elements, although many of them could also be operated in a multistable mode. Since the parameters of physical elements used to represent information are continuous in nature, use of these parameters for discrete data storage requires that certain constraints be imposed on the behavior of the elements. For example, the angular position of a rotatable disk is a continuous quantity, but can be used to represent discrete data if only certain positions are defined, as in a notched counter wheel. A vacuum tube amplifier is an analog device, as evidenced by its transconductance curves, but can be constrained to behave like a discrete device by incorporating it into a bistable circuit called a flip-flop. Data may be stored in a continuous manner on the magnetic surface of a drum, disk, or tape. However, by restricting the use of the magnetic medium to the point where only the presence or absence of magnetization, or its direction, is considered, bistable storage elements are obtained.

The duration of a stable state varies with the particular element and the circuit in which it is used. For example, magnetic storage can be retained for an indefinitely long period. On the other hand, the electrical charge on a condenser will gradually leak off, and at first sight an electrical condenser might not seem suitable as a storage element. However, if it is used in an appropriate circuit, and its charge sampled at time intervals small compared to the interval it takes to lose an appreciable part of its charge, and circuitry provided to periodically regenerate the charges which otherwise would leak off, it becomes useable as a digital storage device. Electrostatic and ferroelectric storage systems described in Chapter 5 employ condenser-like elements.

An important differentiation that enters into the application of different storage devices is whether they are suited better for the storage of a single bit or small group of bits, a main storage unit with a capacity of anywhere from several hundred to tens of thousands of words, or for an auxiliary or file storage unit that may call for hundreds of thousands to millions of words.

Single bit storage elements may be formed from either static or dynamic units, and may be interconnected to form registers and accumulators. Immediate access to any of these elements is possible. The most commonly encountered type of single bit storage device is the flip-flop. Static flip-flops, whose logic and circuitry are described in Section 3.7 and Chapter 4, respectively, can be triggered to either of two states of static equilibrium. They are commonly formed from a pair of regeneratively

coupled amplifiers in a circuit designed to have two stable operating points. By incorporating an electromagnetic delay element (of the type used in electronic circuits for synchronization purposes) into a suitable feedback loop, one can form dynamic types of flip-flops, whose operation does not depend on circuits with stable operating points. Once introduced, a pulse is continually recirculated until the loop is effectively opened momentarily by an externally applied signal (see Section 3.7.5). The presence or absence of the recirculating pulse at an output terminal is arbitrarily tagged to represent either a 1 or 0 state. Static flip-flops are used where the combinational circuits are designed to operate on dc inputs, where 0 and 1 are represented by two voltage levels, and dynamic flip-flops are used in systems where 0 and 1 are represented by the absence or presence of voltage pulses at regularly spaced sampling times.

In a digital computer, (see Chapter 2), the totality of operations to be performed in the solution of a given problem is specified by data in the form of a program originally entered into the computer's main storage. Space in the main store is also used for the storage of initial conditions, and for intermediate and final results. In Chapter 5 there are descriptions of a number of types of main storage systems, including the two most widely used today; namely, magnetic drum and magnetic core storage systems. The control unit of the computer contains the circuits that perform the operations common to the execution of all instructions, namely selection of coded instructions and operands from the main store in some specified sequence, and the advancement of control to the next instruction in a sequence upon the execution of any given instruction. The control as well as the logical or arithmetic operations called for by a specific instruction must be derived from the data contained in a given instruction. Certain storage elements in the control and arithmetic units of a digital computer are reserved for the purpose of receiving this data from the main storage and holding it in a form suitable for input to the various control and arithmetic switching networks within the computer. The need for these storage elements in the control and arithmetic units arises because the data in the main store is usually in a form that cannot be used directly as an input to a voltage or current switching network. To elaborate, a 1 and 0 may be represented in the main store in any of the following ways: by the direction of magnetization of a cell on a magnetic surface, or of a magnetic core, the absence or presence of electric charge on the surface of a dielectric material, or the emergence from a delay line of a pulse train at some specified point during a specified time interval. Before data in any of these forms can be used to drive switching networks, certain preliminary operations are required. These

operations cause selection of a storage location and sensing of the information stored there. In static stores all words are equally accessible and means are provided to sense all bits of a word in parallel. In dynamic stores the access time varies with the location of a word relative to a transducer at the time of selection (see Chapter 5), and bits of a word are usually made available serially, thereby requiring a conversion of the data from serial to parallel form. In both cases means must be provided to transform the data from its form in the main store to representation in the form of the voltage or current states of a set of storage elements in the control and arithmetic units. The outputs of these storage elements can be used as inputs to the control and computation switching networks. Subsequently, the outputs of these networks may be used to alter the states of storage elements in the main store, the control unit or the arithmetic unit. In Section 3.8 there is a general description of how switching and storage elements are interconnected in a digital computer. Whenever it is desired that a network be sequence sensitive, i.e., its response be governed not only by the current input signals, but on preceding ones also, then such a network must contain storage elements. A system composed, not of switching circuits alone, but also containing storage elements, is usually referred to as a sequential switching network. A sequential switching network is, in general, reducible to a multioutput combinational network in many variables. The general characteristics of sequential switching networks will be described after the description in Section 3.7 of the functional characteristics of flip-flops.

3.7. Flip-Flops

The term flip-flop refers to a circuit having basically one, two, or three points, and one or two output points, which can be triggered to each of two stable states by appropriate signals at one or more input points. Once triggered to a particular state, a flip-flop will remain in that state until triggered by an appropriate new input signal. Flip-flops may be utilized for a number of purposes. These functions, described in succeeding chapters, include the following: to receive and retain information for controlling arithmetic and logical operations, to provide time delays for carries in synchronous adders, and to generate timing signals.

Besides differences in the number of input and output points, and the input–output logical relationships, designs will differ, also, in the nature of the points at which inputs are received, e.g., at the grid or cathode of a vacuum tube, and the characteristics of the waveform required to trigger the flip-flop from one stable state to the other, i.e., its amplitude, width, rise and fall times, and repetition rate. In practice, the

width of the input pulse must be only a small fraction of the period corresponding to the input pulse repetition frequency. The stability, reliability, and range of operating frequencies of a flip-flop are determined by circuit parameters. Representative flip-flop circuits are described in Chapter 4.

Before the advent of transistors, flip-flops were usually formed from a pair of vacuum tubes, either triodes or pentodes, regeneratively coupled in a circuit having the characteristic that when one tube was conducting, the other tube was cut off, and vice versa, thus providing the circuit with two stable states. An indication of the state of the circuit was usually obtained by examining the plate voltages. Output signals could be taken from any of a number of points, depending on the polarity, amplitude, and dc level of signal desired. Very few of the digital computers now being built or contemplated utilize vacuum tubes for flip-flops. Instead, transistors are now largely used for this purpose.

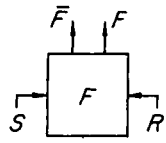
If the flip-flop action is obtained by the regenerative coupling of vacuum tube or transistor amplifiers, two output lines are available. After a switching action has occurred, i.e., in the steadystate, one of the lines will be at a relatively high voltage and the other at a relatively low voltage, depending on which tube or transistor is conducting current more heavily. The two output lines of a flip-flop, A, may be arbitrarily designated as A and $\hat{A}$. The flip-flop is said to be in state A, or $\hat{A}$, depending on the voltages of the output lines. Sometimes a flip-flop may have only one useable output line, as for example in the case where the flip-flop action is obtained by the negative resistance characteristic of a single transistor amplifier. In this case, the flip-flop is said to be in state A or $\hat{A}$, depending on whether the voltage on the output line is relatively high or low.

3.7.1. Static Flip-Flops

Throughout the remainder of the text, the block diagrams shown in Fig. 3.14 will be used to represent the different types of static flip-flops. They vary principally in the number and placement of their input lines. Though the letter F is used in Fig. 3.14, any capital letter may be used to designate a flip-flop (though the letter D is often reserved to indicate a delay element). Usually, the different flip-flops in an assemblage are designated by the same letter with a distinguishing numerical superscript or subscript attached to the letter. In some systems, different capital letters are used to designate different flip-flops or groups of flip-flops, and the particular letters used have some mnemonic significance. In a

TRUTH TABLE

R	S	F_t	F_{t+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	–
1	1	1	–

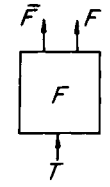


(a)

(a) The R–S flip-flop

TRUTH TABLE

T	F_t	F_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

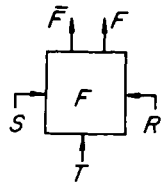


(b)

(b) The T flip-flop

TRUTH TABLE

R	S	T	F_t	F_{t+1}
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
1	0	0	0	0
1	0	0	1	0

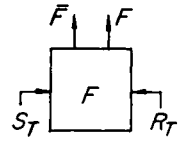


(c)

(c) The R–S–T flip-flop

TRUTH TABLE

R_T	S_T	F_t	F_{t+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



(d)

(d) The R_T–S_T flip-flop

FIG. 3.14.

block diagram, when there is no need to designate a specific flip-flop by letter, the symbol FF (for flip-flop) may be used.

Truth tables are shown in Fig. 3.14 to aid in the description of the input–output relationships of the types of flip-flops shown. In these tables, 1 indicates the presence and 0 the absence of signals at indicated points. When output line $F = 1$, the flip-flop is said to be in the F state, and when output line $\bar{F} = 1$, it is said to be in the $\bar{F}$ state.

The type of flip-flop whose block diagram and input–output truth table are shown in Fig. 3.14(a) is usually referred to as a set-reset type of flip-flop, or simply as an $R-S$ flip-flop, because a signal on the S line sets the flip-flop to the F state and a signal on the R line resets it to the $\bar{F}$ state. R and S refer to the two input lines and their current states, while F_t and F_{t+1} refer to the state of the flip-flop at times t and $(t + 1)$, respectively. $F_t = 0$ means the output line $\bar{F}$ has the value 1 at time t. $F_t = 1$ implies the output line F has the value 1. The condition on the output lines at time $(t + 1)$ will depend on the signals received on the input lines S and R at time t, as well as the state of the flip-flop at time t.

The logical nature of this type of flip-flop will be explained by considering the truth table which defines its operation. A signal on either input line, indicated by a 1 in the appropriate column, will cause the flip-flop to assume a corresponding state, i.e., if $RS = 1$ at time t, $F_{t+1} = 1$; if $\bar{R}S = 1$ at time t, $\bar{F}_{t+1} = 1$ (or $F_{t+1} = 0$). Note that once this type of flip-flop has been set to a particular state, additional signals on the corresponding input line produce no effect. The blanks in the last two rows of the table are used to indicate that the effect of simultaneous inputs is not considered in the design of this type of flip-flop. It is intended for use in systems where this event either cannot occur or is prevented by design from occurring during normal operation.

The type of flip-flop indicated in Fig. 3.14(b) is usually referred to as a complementing or trigger flip-flop or simply as a T flip-flop. It has only a single line for input signals, and successive signals on this line will cause it to trigger alternately from one state to another, which is evident from the truth table. Circuitwise, it is similar to the R–S flip-flop. The most significant difference is that it is symmetrically coupled to a single source of triggering. This makes the circuit especially useful as a counting element.

The flip-flops shown in Figs. 3.14(c) and 3.14(d) effectively combine the functions of both the flip-flops of Fig. 3.14(a) and Fig. 3.14(b). All theoretically possible input signal configurations are not shown in the table of Fig. 3.14(c), since the R–S–T flip-flop is restricted to operate under the condition that signals on more than one input line at a time are not allowed.

The R_T–S_T flip-flop, whose operation is defined by the truth table in Fig. 3.14(d), is similar to the R–S–T flip-flop in that it can be activated in three distinct ways. Although it has only two input lines, it can be set, reset, or triggered. It can be set to either of two specified states by a signal on either the S_T or R_T line or caused to flip from one state to another by application of a signal to both input lines. Inspection of the truth table shows that the logic of this flip-flop differs from that of the R–S flip-flop only in that simultaneous inputs are allowed and cause the flip-flop to change state.

3.7.2. The Characteristic Equation of a Flip-Flop and Derivation of the General Form of its Input Equations

The information in the truth table defining the response of a given type of flip-flop to signals on its input lines can be put in the form of a Boolean expression, which indicates the state of the flip-flop at time $t + 1$

in terms of the states of pertinent variables at time t. This expression is called a difference equation because of the time differential between receipt of an input signal and assumption of a new state. Since this expression also describes the logical nature of a particular type of flip-flop, it is referred to as the characteristic equation of the flip-flop.

The principal logical design problem in the design of a sequential switching network is to determine what the signals on the input lines of a given flip-flop should be in order for the flip-flop to assume a sequence of states in accordance with a difference equation for a given application. This problem may be attacked by first equating the characteristic equation of the flip-flop to the difference equation of a specific application. The latter equation, sometimes referred to as an application equation, is readily obtained from a table showing the state each flip-flop is to assume at each instant of time for each possible configuration of states, at the preceding instant of time, of all elements which may influence it. The application equation can always be written in the following general form

$$F_{t+1} = (xF + y\bar{F})_t \tag{3-43}$$

where x represents one function of the pertinent input variables and y another function of the same variables.

To illustrate how a specific application equation may be derived, let us consider three flip-flops, F^3, F^2, F^1, which are to be used as a counter whose contents are to be augmented by a single binary increment at each succeeding instant of time, and which is to be reset to zero after reaching its maximum value. The contents of this counter at successive instants of time are shown in Table 3.11. For each flip-flop, a specific application equation may be written showing the state it is to assume at time $t + 1$ in terms of the state at time t of itself and others in the group.

$$\begin{aligned}
F^3{}_{t+1} &= (\bar{F}^3 F^2 F^1 + F^3 \bar{F}^2 \bar{F}^1 + F^3 \bar{F}^2 F^1 + F^3 F^2 \bar{F}^1)_t \\
&= [\bar{F}^3 (F^2 F^1) + F^3 \overline{(F^2 F^1)}]_t \\
F^2{}_{t+1} &= (\bar{F}^3 F^2 F^1 + F^3 F^2 \bar{F}^1 + F^3 \bar{F}^2 F^1 + F^3 F^2 \bar{F}^1)_t \\
&= (\bar{F}^2 F^1 + F^2 \bar{F}^1)_t \\
F^1{}_{t+1} &= (\bar{F}^3 F^2 \bar{F}^1 + F^3 F^2 F^1 + F^3 \bar{F}^2 F^1 + F^3 F^2 \bar{F}^1)_t \\
&= (\bar{F}^1)_t.
\end{aligned} \tag{3-44}$$

Each equation states that the presence of any of the stated conditions at time t is to cause the indicated flip-flop to assume the state 1 at time $t + 1$ and any other configuration of the input variables is to cause the flip-flop to assume the state 0 at time $t + 1$. Note that the difference equa-

tions for the specific application neither state nor imply the logical properties of the flip-flops. It will be shown later that any of the types of flip-flops considered can satisfy such application equations.

TABLE 3.11

Time t				Time $t+1$		
F^3	F^2	F^1		F^3	F^2	F^1
0	0	0	⟶	0	0	1
0	0	1	⟶	0	1	0
0	1	0	⟶	0	1	1
0	1	1	⟶	1	0	0
1	0	0	⟶	1	0	1
1	0	1	⟶	1	1	0
1	1	0	⟶	1	1	1
1	1	1	⟶	0	0	0

The input equations for each type of flip-flop (which specify the signals required on the input lines to satisfy the application equation) will be derived now in terms of variables in the general form of the application equation.

Let us consider first, the R–S flip-flop. From its truth table, its characteristic equation is

$$\begin{aligned} F_{t+1} &= (\bar{R}S\bar{F} + R\bar{S}F + RSF)_t \\ &= (\bar{R}S\bar{F} + R\bar{S})_t. \end{aligned} \tag{3-45}$$

The restriction on the simultaneous occurrence of signals on both R and S is expressed algebraically by

$$(RS)_t = 0. \tag{3-46}$$

Since $(RS)_t = 0$, it may be added to Eq. (3-45) without altering its value. This results in a simplification of Eq. (3-45).

$$\begin{aligned} F_{t+1} &= (\bar{R}S\bar{F} + R\bar{S} + RS)_t \\ &= (\bar{R}F + S)_t. \end{aligned} \tag{3-47}$$

Equating the characteristic equation of the R–S flip-flop to the general form of a specific difference equation yields

$$F_{t+1} = (\bar{R}F + S)_t = (xF + y\bar{F})_t. \tag{3-48}$$

To obtain the input equations for the R–S flip-flop in terms of the input

functions x and y, one may proceed as follows. Construct a truth table showing the value of $xF + y\bar{F}$ for each possible configuration of states of x, y, and F. (See Table 3.12). From Eq. (3-48) it is clear that this also defines the value of $\bar{R}F + S$ for each configuration of x, y, F, and allows the values of R and S to be derived by logical inference. For example, consider row 2 or 4. Since $\bar{R}F + S = 0$ in these cases, both $\bar{R}F$ and S must each be equal to 0. Since $F = 1$, this implies that $\bar{R} = 0$, or equivalently that $R = 1$. In row 3 or 7, $\bar{R}F + S = 1$ while $F = 0$. This implies that $S = 1$, and since RS is always equal to zero for this type of flip-flop, $R = 0$. In row 1 or 5, $\bar{R}F + S = 0$, so $S = 0$, and $\bar{R}F = 0$. However, since $F = 0$, R may be either 0 or 1. Therefore, R is represented by the symbol k which may assume either value. In row 6 or 8, $\bar{R}F + S = 1$ while $F = 1$. This condition is satisfied if $\bar{R} = 1$, or equivalently if $R = 0$. When $R = 0$, S may be either 0 or 1 and is represented by k_5 and k_7, respectively. From columns 1, 2, 3, and 5, 6 of Table 3.12, the following equations for R and S, the so-called input equations may be obtained

TABLE 3.12

x	y	F_t	$F_{t+1} = (xF + y\bar{F})_t$ $= (\bar{R}F + S)_t$	R	S
0	0	0	0	k_0	0
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	1	0
1	0	0	0	k_4	0
1	0	1	1	0	k_5
1	1	0	1	0	1
1	1	1	1	0	k_7

TABLE 3.13

x	y	F_t	$F_{t+1} = (xF + y\bar{F})_t$ $= (\bar{T}F + T\bar{F})_t$	T
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	1
1	0	0	0	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	0

$$R = (k_0\,\bar{x}\bar{y}\bar{F} + \bar{x}\bar{y}F + \bar{x}yF + k_4\,x\bar{y}\bar{F}) \tag{3-49}$$

$$S = (\bar{x}y\bar{F} + k_5\,x\bar{y}F + xy\bar{F} + k_7\,xyF). \tag{3-50}$$

That this represents a general solution independent of the value of any k_i can be proved by substitution of the values of R and S given by Eqs. (3-49) and (3-50) into the term $(\bar{R}F + S)_t$ in Eq. (3-48). Therefore, to reduce Eqs. (3-49) and (3-50) to simple form, set $k_0 = k_4 = k_5 = k_7 = 0$

$$R = \bar{x}F \tag{3-51}$$

$$S = y\bar{F}. \tag{3-52}$$

The product RS will be zero independent of the values of x and y. This is assured since $F\bar{F} = 0$. However, in the special case where $\bar{x}y = 0$, the inclusion of F and $\bar{F}$ in the expressions for R and S is not necessary. The simplified expressions for R and S may be obtained from Eqs. (3-49) and (3-50) as follows. First set $k_0 = k_7 = 1$, and $k_4 = k_5 = 0$. Then

$$R = \bar{x}\bar{y}\bar{F} + \bar{x}\bar{y}F + \bar{x}yF \tag{3-53}$$

$$S = xy\bar{F} + xyF + \bar{x}y\bar{F}. \tag{3-54}$$

Since $\bar{x}y = 0$, it may be added to Eqs. (3-53) and (3-54) to yield

$$R = \bar{x}\bar{y} + \bar{x}y = \bar{x} \tag{3-55}$$

$$S = xy + \bar{x}y = y. \tag{3-56}$$

Often $\bar{x}$ and y are time functions in the form $\bar{x} = ut_a$ and $y = vt_b$. If t_a and t_b represent mutually exclusive instants of time or time intervals, the product $\bar{x}$y will be zero. In general one may consider the product $\bar{x}y$ as a sum of products. For $\bar{x}y$ to be equal to zero, each term of the sum must be zero. There is assurance that $RS = 0$, if each term of R contains the variable F while each term of S contains the variable $\bar{F}$, for then each term in the expression for RS will contain the factor $F\bar{F} = 0$. However, if the product of any term in the expression for either $\bar{x}$ or y with all the terms in the other is zero, then the F or $\bar{F}$ modifier (as the case may be) may be eliminated from that term. For example, assume

$$\bar{x} = k + l + m + \ldots$$

$$y = p + q + r + \ldots \tag{3-57}$$

where the letters on the right hand side of Eq. (3-57) represent functions of certain variables. If, say, $ky = 0$, then $R = (k + l + m + \ldots)F$ may be replaced by $R = k + (l + m + \ldots)F$.

The characteristic equations of the T, the R–S–T, and the R_T–S_T flip-flops will be derived next. From the truth table of the T flip-flop, its characteristic equation is

$$F_{t+1} = (\bar{T}F + T\bar{F})_t. \tag{3-58}$$

As in the case of the R–S flip-flop, one first constructs a truth table showing the value of $xF + y\bar{F}$ for each possible configuration of x, y, F. This time $xF + y\bar{F} = \bar{T}F + T\bar{F}$. By a process of logical inference the values of T may be obtained from the values of $\bar{T}F + T\bar{F}$. From columns 1, 2, 3, and 5 of Table 3.13, the following equation for T may be obtained

$$\begin{aligned} T &= \bar{x}\bar{y}F + \bar{x}y\bar{F} + x\bar{y}F + xy\bar{F} \\ &= \bar{x}F + y\bar{F}. \end{aligned} \tag{3-59}$$

If $\bar{x} = y$, Eq. (3-59) may be simplified to

$$T = \bar{x}. \tag{3-60}$$

From the truth table of the R–S–T flip-flop, its characteristic equation is

$$\begin{aligned} F_{t+1} &= (\bar{R}\bar{S}TF + \bar{R}S\bar{T}F + R\bar{S}\bar{T}F + RS\bar{T}F)_t \\ &= (\bar{R}\bar{S}TF + \bar{R}S\bar{T}F + R\bar{S}\bar{T})_t. \end{aligned} \tag{3-61}$$

Since, by definition, $RS = ST = RT = 0$, each may be added to Eq. (3-61). A simplified expression, readily obtainable from a Karnaugh map by selective inclusion of redundancies in a group (see Section 3.5.1.4), is

$$F_{t+1} = (\bar{R}\bar{T}F + T\bar{F} + S)_t. \tag{3-62}$$

To obtain the input equations, one constructs a truth table, as before, showing the value of $xF + y\bar{F}$ for each possible configuration of x, y, F. This time $xF + y\bar{F} = \bar{R}\bar{T}F + T\bar{F} + S)_t$. The values of R, S and T in

TABLE 3.14

x	y	F_t	$F_{t+1} = (xF + y\bar{F})_t$ $= (\bar{R}\bar{T}F + T\bar{F} + S)_t$	R	S	T
0	0	0	0	k_0	0	0
0	0	1	0	k_1	0	k_1*
0	1	0	1	0	k_2	k_2*
0	1	1	0	k_3	0	k_3*
1	0	0	0	k_4	0	0
1	0	1	1	0	k_5	0
1	1	0	1	0	k_6	k_6*
1	1	1	1	0	k_7	0

*k_i is the complement of k_i.

TABLE 3.15

x	y	F_t	$F_{t+1} = (xF + y\bar{F})_t$ $= (\bar{R}_T F + S_T \bar{F})_t$	R_T	S_T
0	0	0	0	k_0	0
0	0	1	0	1	k_1
0	1	0	1	k_2	1
0	1	1	0	1	k_3
1	0	0	0	k_4	0
1	0	1	1	0	k_5
1	1	0	1	k_6	1
1	1	1	1	0	k_7

Table 3.14 may be obtained by inference from the values of $\bar{R}T\bar{F} + T\bar{F} + S$. The input equations are

$$R = k_0\, \bar{x}\bar{y}\bar{F} + k_1\, \bar{x}\bar{y}F + k_3\, \bar{x}yF + k_4\, x\bar{y}\bar{F} \tag{3-63}$$

$$S = k_2\, \bar{x}y\bar{F} + k_3\, x\bar{y}F + k_6\, xy\bar{F} + k_7\, xyF \tag{3-64}$$

$$T = \bar{k}\, \bar{x}\bar{y}F + \bar{k}_2\, \bar{x}y\bar{F} + \bar{k}_3\, \bar{x}yF + \bar{k}_6\, xy\bar{F}. \tag{3-65}$$

If $k_1 = k_2 = k_3 = k_6 = 1$, and $k_0 = k_4 = k_5 = k_7 = 0$

$$R = \bar{x}F$$
$$S = y\bar{F} \qquad \text{Case (1)}$$
$$T = 0.$$

If all $k_i = 0$

$$R = 0$$
$$S = 0 \qquad \text{Case (2)}$$
$$T = \bar{x}F + y\bar{F}.$$

If $k_0 = k_1 = k_6 = k_7 = 1$, and $k_2 = k_3 = k_4 = k_5 = 0$

$$R = \bar{x}\bar{y}$$
$$S = xy \qquad \text{Case (3)}$$
$$T = \bar{x}y.$$

If $k_0 = k_1 = k_2 = k_6 = 1$, and $k_3 = k_4 = k_5 = k_7 = 0$

$$R = \bar{x}\bar{y}$$
$$S = y\bar{F} \qquad \text{Case (4)}$$
$$T = \bar{x}yF.$$

If $k_1 = k_3 = k_6 = k_7 = 1$, and $k_0 = k_2 = k_4 = k_5 = 0$

$$R = \bar{x}F$$
$$S = xy \qquad \text{Case (5)}$$
$$T = \bar{x}y\bar{F}.$$

In cases (1) and (2) the R–S–T flip-flop becomes equivalent to the R–S and T flip-flops, respectively. In some instances the solutions called for by (3), (4), or (5) may be simpler, i.e., require fewer combinational circuits and/or fewer inputs per combinational circuit than that called for by solutions (1) or (2). This is equivalent to saying that sometimes the use of an R–S–T type of flip-flop is more economical in the amount of circuitry required to form its input signals than either the R–S or T type of flip-flop.

From the truth table of the R_T–S_T flip-flop, its characteristic equation is

$$\begin{aligned} F_{t+1} &= (\bar{R}_T S_T F + \bar{R}_T S_T \bar{F} + R_T \bar{S}_T F + R_T S_T \bar{F})_t \\ &= (\bar{R}_T F + S_T \bar{F})_t. \end{aligned} \tag{3-66}$$

By a process now familiar, the values of R_T and S_T shown in Table 3.15 may be derived. The input equations are

$$\begin{aligned} R_T = {} & k_0 \bar{x}\bar{y}\bar{F} + \bar{x}\bar{y}F + k_2 \bar{x}y\bar{F} + \bar{x}yF \\ & + k_4 x\bar{y}\bar{F} + k_6 xyF. \end{aligned} \tag{3-67}$$

$$\begin{aligned} S_T = {} & k_1 \bar{x}\bar{y}F + \bar{x}y\bar{F} + k_3 \bar{x}yF + k_5 x\bar{y}F \\ & + xy\bar{F} + k_7 xyF. \end{aligned} \tag{3-68}$$

If $k_0 = k_2 = k_3 = k_7 = 1$, and $k_4 = k_6 = k_1 = k_5 = 0$

$$R_T = \bar{x} \tag{3-69}$$

$$S_T = y. \tag{3-70}$$

The general input equations (3-69) and (3-70) for the R_T–S_T flip-flop are equal to the input equations (3-55) and (3-56) of the R–S flip-flop when $\bar{x}y = 0$.

3.7.3. Derivation of Specific Input Equations from a Karnaugh Map of the Application Equation

The flip-flop input equations for a particular application can be obtained from a plot of the application equation (and any redundancies that may exist) on a Karnaugh map. For example, if the difference equation and redundancies are as follows

$$F_{t+1} = ABF + BC\bar{F} + B\bar{C}F + \bar{A}B\bar{F} \tag{3-71}$$

$$A\bar{B}\bar{C}F = AB\bar{C}\bar{F} = 0 \ldots \tag{3-72}$$

their plot on a Karnaugh map would be as shown in Fig. 3.15. (k marks a redundancy and may be assigned a checked or unchecked value).

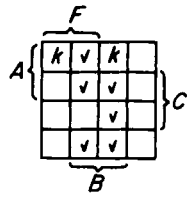


FIG. 3.15. Plot of the difference equation $F_{t+1} = ABF + BC\bar{F} + B\bar{C}F$ and the redundancies $AB\bar{C}F = AB\bar{C}F = 0$

Only those terms for which F is true appear in the left half of the map while those for which $\bar{F}$ is true appear in the right. Therefore, the values of x and y in the general application equation (3-43), are equal to the logical sum of the checked squares (and $\bar{x}$ and $\bar{y}$ are equal to the logical sums of the unchecked squares) in the left and right half planes respectively. Substituting the values of $\bar{x}$ and y obtained from inspection of the map into the general input equations (3-51) and (3-52) of the R–S flip-flop yields

$$\begin{aligned} R &= (\bar{B} + \bar{A}BC)F \\ &= (\bar{B} + \bar{A}C)F \end{aligned} \tag{3-73}$$

$$S = B\bar{F} \tag{3-74}$$

3.7.4. Derivation of Specific Input Equations from Consideration Only of Conditions Preceding a Change

The procedures described thus far for obtaining flip-flop input equations are inefficient because they consider all possible input signals to a flip-flop rather than only those which produce an effect. Since a condition which does not cause a flip-flop to change its state is irrelevant to its operation, there is no need to include it as a term in any input equation. Any such condition may be omitted by deriving the input equations in the following manner. Consider and include only those conditions whose existence at time t cause a change in the state of a flip-flop at time $t + 1$. In the case of an R–S flip-flop, it is only necessary to include in the S

and R equations those conditions which are to cause the flip-flop to change from state 0 to 1 and 1 to 0, respectively. For example, consider again the three stage counter comprised of flip-flops: F^3, F^2, F^1. We see that flip-flop F^1 changes state every time. Therefore, $R^1 = F^1$, $S^1 = \bar{F}^1$. Flip-flop F^2 changes from 0 to 1 when either $\bar{F}^3\bar{F}^2F^1$ or $F^3\bar{F}^2F^1$ is true. Therefore, $S^2 = \bar{F}^2F^1$. The change from 1 to 0 occurs when $\bar{F}^3F^2F^1$ or $F^3F^2F^1$ is true. Therefore, $R^2 = F^2F^1$. F^3 changes from 0 to 1 only when $F^3F^2F^1$ is true, so $S^3 = \bar{F}^3F^2F^1$, and from 1 to 0 only when $F^3F^2F^1$ is true, so $R^3 = F^3F^2F^1$. Actually, it is not even necessary to have two sets of columns, one for time t and one for time $t + 1$. The different rows can be written in a sequential order, i.e., if the contents of any given row are taken to represent the state of a group of elements at time t, that of the row above may be considered to represent the state at time $t - 1$ and that of the row below it the state at time $t + 1$. In short, succeeding rows correspond to succeeding instants of time. There is the additional convention that, when lower rows indicate later instants of time, one proceeds from the bottom row back to the top row. With this arrangement, the input equations for each flip-flop may be determined by scanning each column from top to bottom (in that order) and noting all the conditions of all the flip-flops just prior to a change of the flip-flop being considered. In the case of a trigger flip-flop, the logical sum of all these conditions represents the input equation. In the case of the R–S flip-flop the logical sum of all those conditions preceding a change from 0 to 1 represents S and the logical sum of those preceding a change from 1 to 0 represents R.

There are times when a set of flip-flops may go from any given state to either of two other states depending on the presence of external control signals. For example, the value of a counter may be either increased or decreased by a single increment each time a count command is received in accordance with whether the command says count-up or count-down. The way to derive the conditions for the count-up logic has already been described. The count-down logic is obtained in a similar manner, the difference being that each column is scanned from the bottom row to the top and the bottom row follows the top row. The logical product of the count-up logic and the count-up command is formed, and so is the logical product of the count-down logic and the count-down command. The logical sum of these two products represents the inputs to the flip-flops for the combined up–down counter action.

Another example will be provided here of how the input equations for a flip-flop may be determined directly from consideration of its func-

tion. Assume that we have a set of flip-flops: A^4, A^3, A^2, A^1 whose contents at time t_0 are given by K_4, K_3, K_2, K_1 where each K_i may be either 0 or 1. It is required that the contents of this register be made available by examination of A^1 at four successive instants of time. This requirement can be met if the contents of each flip-flop are shifted one place to the right each time a shift command, C_s, appears. The contents of the flip-flops at four successive instants of time are shown in Table 3.16.

TABLE 3.16

	A^4	A^3	A^2	A^1
t_0	K_4	K_3	K_2	K_1
t_1	—	K_4	K_3	K_2
t_2	—	—	K_4	K_3
t_3	—	—	—	K_4

If the contents of A^1 are to be inspected only during the interval t_0 through t_3, it is irrelevant whether A^4 is set to 1 or to 0 upon receipt of the first shift command. From consideration of Table 3.16, it is apparent that if an R–S type of flip-flop is used

$$S^4 = \ldots \qquad S^3 = A^4C_s$$
$$R^4 = \ldots \qquad R^3 = \bar{A}^4C_s$$
$$S^2 = A^3C_s \qquad S^1 = A^2C_s$$
$$R^2 = \bar{A}^3C_s \qquad R^1 = \bar{A}^2C_s.$$

In this case, even the construction of the simple table is not necessary, for the verbal statement of the requirements clearly indicates the nature of the flip-flop input equations.

3.7.5. DYNAMIC FLIP-FLOPS

The basic bistable active storage device utilized in ac coupled systems of circuit logic (described in Chapter 4) is the so-called dynamic flip-flop, sometimes referred to as the regenerative or ac flip-flop. Schematics of dynamic flip-flops are shown in Figs. 3.16(a), (b), and (c).

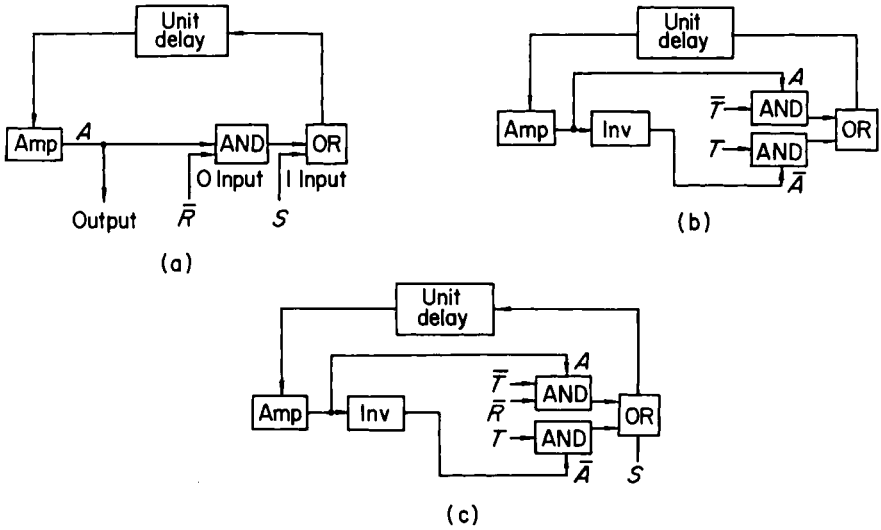


FIG. 3.16. Dynamic flip-flops: (a) set–reset type, (b) trigger type, (c) set–reset–complement type. [Amp denotes an amplifier. Inv denotes an inverter which amplifies and complements a signal. The unit delay is provided by an electromagnetic delay line].

The set reset type of dynamic flip-flop shown in Fig. 3.16(a) is set to the 1 state by applying a set pulse, S, to the 1 input line. As a result of the recirculation loop, a sequence of pulses (separated by a one pulse period delay) will be produced at the output. This condition of dynamic equilibrium may be used to represent a 1 state. The 0 state may be produced by closing the AND gate (see Section 4.2), thereby terminating the recirculation. This is done by making $\bar{R}$ false. In another scheme, the signal R is applied to an inhibitor used in place of the AND gate. A statement of conditions necessary to produce and maintain the 1 state is

$$A_{i+1} = S + \bar{R}A_i$$

where $i + 1$ and i are used to distinguish between the outputs of the amplifier at times $i + 1$ and i. Instead of using an amplifier to maintain pulse shape and amplitude, an AND gate controlled by clock pulses could be inserted in the loop ahead of the OR gate. In this case, a pulse applied to the 1 input line must be at a time such that, after passing through the delay unit, it arrives at the clock gate in coincidence with the next clock pulse.

The trigger type of dynamic flip-flop shown in Fig. 3.16(b) is designed to change state each time a trigger pulse T is applied. The input–output relation expressed in Boolean algebra is

$$A_{i+1} = \bar{A}_i T + A_i \bar{T}$$

which states that if the flip-flop is in the off or 0 condition designated by $\bar{A}_i$, a trigger pulse T will turn it on. It will stay on as long as another T pulse is not applied, i.e., as long as the condition $A_i\bar{T}$ exists.

The set–reset–complement type of dynamic flip-flop shown in Fig. 3.16(c) combines features of the set–reset and the trigger types. The input–output relationship is

$$A_{i+1} = S + \bar{A}_i T + A_i \bar{T} \bar{R}.$$

This states that an S pulse will set the flip-flop to the on condition. If it is off, a trigger pulse T will set it to the "on" condition. Once on, it will remain in this condition until a trigger or reset pulse is applied.

3.8. Sequential Switching Networks

As stated earlier, a sequential switching network is formed from switching elements and storage elements. As a prelude to considering what is required to completely specify at any given time the condition or state of the network, certain assumptions will be stated. Since the constrained logical behavior rather than the complete circuital behavior of such networks is of interest here, it will be assumed that the condition of the network will be observed only at discrete times, when the network is in the steady state, and that input signals are applied and output signals produced only at these discrete times. Another assumption, implicit in the nature of digital elements, is that a given element always produces precisely the same response to a given stimulus. For example, if all inputs to an AND gate have the value 1, the output of the gate will be 1. The appearance of a proper signal at the input to a trigger flip-flop will cause the flip-flop to change to its complementary state. It is apparent that a sequential switching network, which is an integrated collection of discrete valued stimulus response devices, can be considered as a single large behaviorial device, i.e., as an organism.

A complete behavioral description of the organism described is given by enumerating all the distinguishable states which it can assume, and the permissible transitions from any given state to other states. At any time of observation, the functional state of the organism is completely described by a statement of the current state of each storage element. If the organism has a total of m bistable storage elements, then it is capable, at most, of exhibiting 2^m distinguishable states. The state which the organism will assume after any given state, a, depends on its internal structure, i.e., the way in which its storage and switching elements are

interconnected, where lines that will carry input signals from external sources are connected, and the signals present on the input lines at the time the organism is in state a.

Of course, the function of an organism like a sequential switching network is not merely to assume a number of distinguishable stable states. It is designed to produce certain useable output signals. These output signals usually take the form of the output of certain combinational circuits or the states of specified storage elements in the network. The preceding discussion may be clarified by reference to Fig. 3.17. It indi-

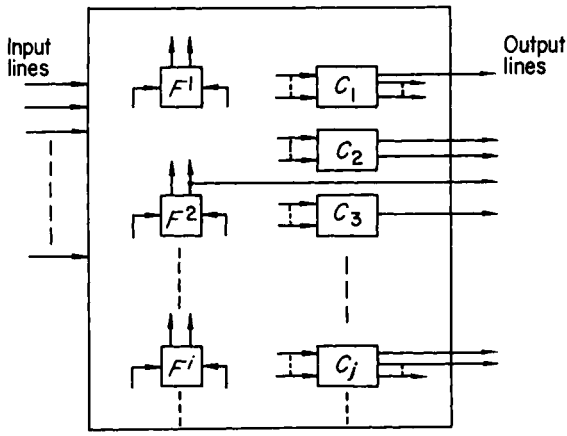


FIG. 3.17. Model of a sequential switching network. (F = storage element, C = combinational circuit)

cates the internal structure of a sequential switching network, and also the lines that convey input signals from external sources as well as those on which specified output signals appear. The network consists of m bistable storage elements $F^1, F^2, \ldots F^m$ and a number of combinational switching circuits $C^1, C^2, \ldots C^n$. Some of the input lines are used as inputs to the combinational circuits while others may be coupled directly to the input lines of the storage elements. However, there is no loss of generality if it is assumed that all the input lines are coupled only to combinational circuits. The other inputs to a combinational circuit come either from the outputs of specified storage elements or of combinational circuits. The inputs to the storage elements come either from the outputs of other storage elements or from the outputs of the combinational circuits. Again, there is no loss of generality if it is assumed inputs to the storage elements come only from the outputs of combinational circuits. The output lines may be coupled either to the outputs of specified combinational circuits or storage elements.

The logical state of a sequential network in the steady state may be described in terms of the states of its storage elements. For convenience, we will sometimes use the term "superstate" to emphasize "logical state of all internal storage elements of the network." When the context permits, sometimes the term "state" will be used interchangeably. The superstate that exists at time i is a function of the following

(1) The superstate at time $i - 1$
(2) The external inputs received by the network at time $i - 1$
(3) The structure of the network (which defines the sequence of permissible superstates).

For a network of m storage elements in which all 2^m possible superstates are allowed, during a sequence of n transitions one of $(2^m)^n$ possible superstate sequences will be generated. With like inputs, the superstate following a particular superstate will always be the same. This quality of exact repeatability is an important characteristic of digital machines. It means that in the solution of a given problem, a digital computer should always pass through the same sequence of superstates, and produce the same solution. If it does not, one knows that something is amiss. This is a property that can be utilized in error detecting techniques (see Chapter 9).

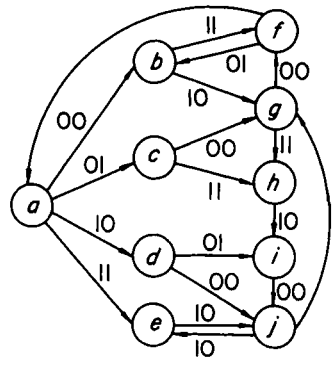


FIG. 3.18. Superstates and permissible superstate sequences of a sequential switching network

Figure 3.18 illustrates how a sequential network advances from one superstate to another. Each encircled letter designates a particular superstate which, we recall, refers to a particular state of all the storage elements $F^1, F^2, \ldots F^m$. In this example, it is assumed that there are only two input lines, A and B, and that at any specified instant, any of the four possible input signal combinations: 00, 01, 10, or 11 may be present.

The directed line segments and their associated numbers define the structure of the network, for they indicate to what superstate a particular superstate will advance when given input signals are received. In general, not all input signals will cause a change in a superstate. Only those input signals are shown which cause the network to advance from one superstate to another. In the example, when the network is in superstate *a*, any of the four possible input signal combinations will cause it to advance to another superstate. However, when the network is in superstate *e*, it can advance to another superstate, namely *j*, only if *A* and *B* have the values 1 and 0, respectively.

Some comment is in order here on the subject of the two basic modes in which a sequential network may be made to operate. One is termed synchronous operation and the other asynchronous. In synchronous operation, switching action can occur only at distinct times. These times are specified by uniformly spaced pulses received from a generator referred to as a clock. By combining all switching signals in AND combinations with clock signals, no switching action can occur except at times when clock signals are present. On the other hand, an asynchronous system is free-running. An asynchronous network may advance from its present superstate to another when signals are generated indicating that the transition from the preceding to the present superstate has been completed. These signals are generated by the network itself. The topic of synchronous and asynchronous operation will be discussed in Chapters 4 and 7.

The preceding description of the mode of operation of a sequential network leads, at least in principle, to a procedure for synthesis. We begin by assuming an arbitrary initial superstate for the network. Then we must specify, for each possible combination of superstate and input configurations, the output configuration to be produced and the succeeding superstate. This process is repeated until all allowable superstates have been specified. In practice, the network will be closed in the sense that it should be possible to go through some path from any superstate to any other. The structure of the network shown in Fig. 3.18 is such that this will be so.

3.8.1. Minimization of Storage Elements in a Sequential Network

Circuit analysis and synthesis procedures described by Huffman [1954] and Mealy [1955] allow one to minimize the number of storage elements required in a sequential switching network. However, they do so without concern for the number of switching elements required.

A little reflection indicates that the total number of storage elements required to perform some complex function or series of operations will depend on how many different storage configurations (superstates) are required during the course of these operations. In essence, the Huffman–Mealy method provides a formalized method for reducing the number of these configurations to a minimum. The method consists essentially of a procedure for eliminating unnecessary states. This is possible when the function of the network is so stated initially that a direct translation to a truth table produces one or more equivalent states.

One state is equivalent to another state if, for all possible configurations of input signals that may be presented at a time, the same outputs are produced and the two states advance to the same state. A procedure for detecting and eliminating equivalent states is as follows: 1) Construct a table in which all states are represented symbolically and which shows the state at time $n + 1$ and the outputs at time n as a function of the state at time n and the inputs at time n. 2) From this table of states form one or more sets of states in which all members of a set produce the same configuration of output signals in response to each of the possible input signal configurations, and assign a number to each set for identification. 3) Determine and list the assigned number of the set to which each state advances for all possible input configurations. 4) From each set in which there are now states that do not produce the same set of output configurations as other members of the set, form two or more new sets by grouping those states which do. 5) Repeat steps 2, 3 and 4 until no additional sets can be formed. At this point, all states in any one set are equivalent and may be replaced by a single state. The total number of states required is the number of sets formed. A simplified table of states can be produced now, and from it one can generate difference equations, flip-flop input equations and output equations.

If for any state one or more output signals are undefined, or there are configurations of input signals for which the next state is undefined, these undefined items may be assigned any value necessary to establish an equivalence.

Let us consider now the effect that the number of originally specified states has on the reduction process. For example, suppose there are 192 states. In this case, a reduction of 65 states (to 128) is required to save one flip-flop. If 130 states were called for originally, a reduction of only two states is required to save one flip-flop. Since the number of states that can be represented by n flip-flops is 2^n, and since $2^n - 2^{n-1}$ increases with n, the maximum number of states to be eliminated to save one flip-flop depends on the particular interval in which the originally

specified number of states falls. However, regardless of the magnitude of this number, an elimination of some states will generally reduce the amount of combinational circuitry.

The Huffman-Mealy method does not guarantee an optimum network, since

(1) The network with the fewest storage elements is not necessarily the most desirable, when other considerations, such as the number of switching elements, are taken into consideration.

(2) It provides no indication of how the different states should be represented by the flip-flops for the switching network to be minimal. (In other words it is no substitute for ingenuity)

(3) It cannot be guaranteed to give the smallest number of states unless there are no redundancies present in the network.

In regard to item (1), there are times when one may wish to use more than the minimum number of storage elements—for example, to realize savings in the number of switching elements, to simplify visualization of the function of the flip-flops (as an aid to maintenance), to facilitate a future expansion. The trade-offs possible between storage and switching elements are referred to in subsequent chapters. For example, in Section 4.2.7 there is a description of how additional flip-flops can be brought in to reduce the number and/or complexity of the combinational circuits. As indicated in Section 7.5, if 2^n rather than n flip-flops are used to generate 2^n distinct states (i.e. if a non-weighted code is used) so that particular flip-flops may be associated with particular classes of functions, the outputs of the flip-flops can be connected directly to gates without the use of intervening decoding and encoding switching networks (see Section 4.8).

If the magnitude of the simplification process becomes excessive, one can construct a computer program, incorporating the Huffman-Mealy algorithm or its equivalent, which can accept a functional description of a sequential network and from it proceed through the operations required to automatically produce tables of states, difference equations, flip-flop input equations and network output equations.

3.9. The Advantages of a Boolean Algebraic Description of a Digital Computer

At this point, let us review the nature of a digital computer for the purpose of showing why it may be described by Boolean algebraic equations, and the advantages that may be derived from such a description. We recall that a digital computer is composed principally of switching and storage elements. (Other physically essential elements of a nonlogical nature will be described in Chapter 4). A statement of the permissible

superstate sequences a computer can assume, in terms of the requisite conditions for transitions of superstates, is readily made by a Boolean algebraic description. This description implies how the elements are interconnected. For a network whose storage elements are all of the active type, this description is provided by the flip-flop input equations. In a practical computer, with a large capacity main store comprised of passive storage elements and, perhaps, delay line registers, the description must also include input equations for the record stations. (See, for example, those in Section 7.6.3). In a transition between two superstates, only a small number of bits is recorded into or read from the main store, and by means of a small set of recording and reading stations. This greatly simplifies the logical description of a machine, for the input signals from the main store to the active storage elements contain only the small set of signals from the reading stations and the input signals to the main store consist of the input signals to the small set of record stations.

The basic reasons why Boolean algebra is well suited for the description of a binary (or binary coded) digital computer are: It is easy to equate the two values of the binary number system to the two values of Boolean algebra. In a binary machine all signals whether representing arithmetic, logical, or control operations have to be specified only to the extent of existing or not. All the conditions to be met for signals to occur at prescribed times and places may be included in a Boolean expression and an equivalence established between the truth values of the expressions and the occurrence of the signals.

Although the use of Boolean algebra has certain limitations in the design of digital computers, it is a tool of major importance offering many significant advantages. Its use in preliminary design (with or without auxiliary block diagrams) permits a general description of switching operations without the use of circuit schematic diagrams, thus enabling the designer to focus his attention on logical organization. At the same time it enables him to move freely from the level of logical organization to that of electronic embodiment. Another important group of advantages follows from the fact that it provides formalized methods which, from an initial formulation, can generate all equivalent forms and alternate mechanizations. One of these may then be chosen on the basis of such criteria as a minimal number of circuit components, a minimal loading of circuits, the use of terms already developed elsewhere in the machine, fewer connections, etc. Often the economy is realized from the elimination of implicit redundancies in the original formulation of the switching function. At other times a simplification may result from the fact that a Boolean description facilitates the inclusion of certain nonallowable states as well as intentional redundancies for the purpose of improving reliability.

In Section 7.7 there is a description of the use of Boolean algebra as an aid to 1) automating many tedious aspects of computer design—for example, circuit loading computations and the generation of wiring tabulations (used in place of wiring diagrams), 2) constructing simulation programs for testing a new design, 3) utilizing time-sharing techniques (which permit not only savings in components but can aid the equalization of circuit loading since various functional requirements can be distributed in a number of ways among time-shared flip-flops).

A Boolean algebraic description of a preliminary design affords a convenient indication of where there are deficiencies in the design and how they may be corrected. Also, the fact that Boolean algebraic equations representing specific functions can be easily separated out and studied independently (except possibly in the case where there is considerable time-sharing) aids in the understanding of a machine's operation and simplifies maintenance.

In the synthesis of a digital computer, there must be generated a description of the machine in terms of the types of elements and circuits to be used for the entry, storage, and processing of the data, and for the display of solutions, and the way in which all these elements are to be organized, i.e., interconnected, into an integrated system for accomplishing certain objectives. This latter task is often referred to as logical design. The term, unfortunately, is sometimes interpreted to mean simply the writing of a description of a machine in terms of logical, i.e., Boolean algebraic, equations. However, this is only one phase of logical design. It begins with the conception of the general structure of a machine for satisfying specified requirements. This conception does not occur in the form of Boolean algebraic equations, but rather in terms of an arrangement which by experience and native skill a designer senses to be optimum for the purposes he has in mind. The derivation of this arrangement is based on the functional specifications, usually in terms of verbal statements, and is aided by preliminary organizational descriptions in the form of information flow block diagrams. After the general organizational structure has been outlined, a description of the machine may be written in terms of Boolean algebraic equations. If the original conception is not well thought out, simplification of the Boolean description will not relieve all its ills. Actually, the use of Boolean algebra is not even necessary to the design of a digital computer. Many of the early large digital computers were designed without its aid. However, Boolean algebra is an important tool which simplifies the description of a digital computer and provides the other advantages which have been referred to earlier. In addition, its use, in general, stimulates and facilitates the creation of

more sophisticated designs, i.e., those providing more efficient utilization of switching and storage elements.

3.10. Clock Pulse Generators and Timing Circuits

When establishing an equivalence between the two values of Boolean algebra and the physical states of various electrical or mechanical devices, a practical problem presents itself. Physical devices cannot instantaneously change their state because of mechanical inertia or electrical capacitance or inductance. For example, in passing between open and closed states, there is a time interval during which there is uncertainty as to which state a relay is in; elements like vacuum tubes, transistors, diodes, cannot be switched between states of high or low conduction without passing through intermediate states whose duration is determined by the electrical capacitance and inductance of these elements and the circuits in which they are incorporated. One way to bypass this difficulty is to specify that the states of circuit elements will be inspected only at discrete times when the elements can be considered to have reached steady states.

It is convenient, for purposes of synchronization and logical organization, as well as for the reason stated above, to provide a clock, consisting of uniformly spaced electrical signals which are continually generated by some device within the system. The clock signals define discrete time intervals and the gates are so designed that there is ample time to complete a switching action during the clock interval. One way of using the clock to control all switching operations in a computer is to combine the outputs of all combinational circuits in an AND gate with the clock signal. Then, a clock signal must be present in order for any of these circuits to produce an output.

Sometimes there is a requirement for a multiplicity of clock pulse trains, all of the same frequency, which are so phased that none of the pulses of one train are time coincident with those of another train. All of these pulse trains may be generated by a single source, referred to as a multiphase clock. A multiphase clock is used in a number of computers, e.g., the SEAC, and is also required for driving certain types of magnetic core logic circuits (see Chapter 4).

A number of distinct time signals can be generated by means of a binary counter (see Section 6.1.1) activated by signals from a clock generator. Signals from this clock pulse counter can be used to specify when prescribed switching operations are to occur. For example, if it is desirable that the output, K, of a combinational circuit be capable of influencing some other circuit only at time, T_8, then K is combined in an AND

gate with a signal that is true if, and only if the counter indicates time T_8, and the output of this gate is used as the input to the circuit in question. A clock pulse counter does not necessarily have to count in a conventional way (see Section 6.1.1.6).

A device for converting an input pulse train whose pulses may occur at random times into an output pulse train of the same average pulse rate, and with a fixed interval between pulses, is termed a synchronizer. To operate correctly, it must have timing pulses whose separation is less than that of any two input pulses. This insures the occurrence of a timing pulse between any two data pulses. Considered functionally, a synchronizer is a memory device in which a data pulse is read in at an arbitrary time, and always read out at a specified time, namely, upon the occurrence of the next timing pulse. Once a pulse has been read out, subsequent timing pulses have no effect on the circuit until after a new data pulse has been read in. As the name implies, a synchronizer is utilized to synchronize external inputs with a system's internal signals. It is a specialized case of a buffer register (see Section 4.9).

It has already been noted that in synchronous systems, information can be sensed only during the coincidence of a clock signal with information signals. Also, there is a fixed phase relationship between the clock signals and the information signals. For example, when a magnetic disk or drum is used as a storage medium, the clock signal is usually generated from a track on which uniformly spaced signals have been recorded. Since the information bits are recorded on the same surface, the proper phase relationship is maintained between the information bits and the clock bits, even if the angular velocity of the surface varies. In other systems the phase and frequency of the information bits may be determined by an independently generated clock signal. In still other systems it may be inconvenient or impossible to use an independently generated clock signal. In these systems the clock signal may be derived from the information bits themselves by means of a circuit referred to as a phased clock pulse generator. Such a circuit is described in the article by L. D. Seader [1957] listed in the bibliography of Chapter 5.

3.11. Subdivision of the Computer Synthesis Problem

The general method of sequential network synthesis referred to in Section 3.8 would, in practice, be unwieldy for machines requiring a great number of switching and storage elements. In the design of large, general purpose, stored program computers, it is convenient to subdivide the overall design problem into the problem of designing a number of smaller units which correspond to the major functional units of a general purpose com-

puter which were described in Chapter 2, namely the main storage, the arithmetic unit, the input equipment, the output equipment, and the control unit. There are, of course, many forms which each of these units can assume, both as a result of the particular circuitry used, and the logical arrangement of this circuitry. Chapter 4 is devoted principally to a description of the different sets of circuits that have been used as the logical building blocks of present-day digital computers, and Chapter 5 describes the major types of physical realization of large capacity storage units. Chapter 6 describes a large number of logical arrangements, and circuit mechanizations that could be employed to implement various arithmetic and control processes, e.g., synchronization, counting, addition, subtraction, multiplication, division, comparison, etc. In Chapter 7, the pertinent characteristics of storage and arithmetic units are reviewed, and a description of the control unit design problem is provided to aid in an appreciation of the problems of computer synthesis. The important concept of time-sharing is introduced to show how its use permits a reduction in the amount of equipment required for a computer (at the cost of a reduced speed of operation). Finally, the logical design of two computers, one employing a static and the other a dynamic type of main store is derived. These designs also illustrate that it is not always necessary to sharply subdivide a machine into separate compartments for the operations of storage, arithmetic, and control.

LITERATURE

Abhyankar, S. [1958] Minimal sum of products of sums, expressions of Boolean functions, *IRE Trans. El. Comp.* **EC-7,** 268-276.

Beatson, T. J. [1958] Minimization of components in electronic switching circuits, *Trans Amer. Inst. Elec. Engrs.* **77** (I), 283-291.

Birkhoff, G., and MacLane, S. [1948] *A Survey of Modern Algebra,* pp. 311-25, MacMillan, London.

Boole, G. [1854] *Investigation of the Laws of Thought,* reprint, Dover, New York, 1951.

Caldwell, S. H. [1954] The recognition and identification of symmetric switching functions, *Trans. Amer. Inst. Elec. Engrs.,* **73** (I) 142-147.

Epstein, G. [1958] Synthesis of electronic circuits for symmetric functions, *IRE Trans. El. Comp.* **EC-7,** 57-60.

Ghazala, M. J. [1957] Irredundant disjunctive and conjunctive forms of a Boolean function, *IBM Journal,* **1,** 171-176.

Ginsburg, S. [1959a] A synthesis technique for minimal state sequential machines, *IRE Trans. El. Comp.,* **EC-8,** 13-24.

Ginsburg, S. [1959b] A technique for the reduction of a given machine to a minimal state machine, *IRE Trans. El. Comp.,* **EC-8,** 346-355.

Hartree, D. R. [1949] *Calculating Instruments and Machines,* pp. 99-105, Univ. of Illinois Press, Urbana, Illinois.

Huffman, D. A. [1954] The synthesis of sequential switching circuits, *Jour. Franklin Inst.*, **257**, 161-190, 275-303.

Huffman, D. A. [1957] The design and use of hazard-free switching networks, *Jour. ACM*, **4**, 47-62.

Huntington, E. V. [1904] The algebra of logic, *Trans. Amer. Math. Soc.*, **5**, 288-309.

Karnaugh, M. [1953] The map method for synthesis of combinational logic circuits, *Trans. Amer. Inst. Elec. Engrs.*, **72** (I), 593-99.

Karnaugh, M. [1954] A map method for synthesis of logic circuits, *Trans. Amer. Inst. Elec. Engrs.*, **73** (I), 136.

Keister, W., Ritchie, A. E., Washburn, S. H. [1951] *The Design of Switching Circuits*, Van Nostrand, New York.

Kleene, S. C. [1952] *Introduction to Metamathematics*, Van Nostrand, Princeton, N.J.

Ledermann, W. [1957] *Introduction to the Theory of Finite Groups*, Interscience, New York.

Lee, C. Y. [1954] Switching functions on an n-dimensional cube, *Trans. Amer. Inst. Elec. Engrs.*, **73** (I), 289-91.

Lee, C. Y., and Chen, W. H. [1956] Several valued combinational switching circuits, *Trans. Amer. Inst. Elec. Engrs.*, **75** (I), 278-283.

Lewis, I. A. D. [1951] A symbolic method for the solution of some switching and relay circuit problems, *Proc. Inst. Elec. Engrs.*, **98** (I), 181-191.

McCluskey, Jr., E. J. [1956] Minimization of Boolean functions, *Bell System Tech. Jour.*, **35**, 1417-1444.

McCluskey, Jr., E. J. [1958] Iterative combinational switching networks—general design considerations, *IRE Trans. El. Comp.*, **EC-7**, 285-291.

McCulloch and Pitts [1943] A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophysics*, **5**, 115.

Mealy, G. H. [1955] A method for synthesizing sequential circuits, *Bell System Tech. Jour.*, **34**, 1045-1080.

Moore, E. F. [1952] A simplified universal Turing machine, *Proc. ACM Meeting, Toronto*, 50-55.

Moore, E. F. [1956] Gedanken experiments on sequential machines, *Automata Studies* (Annals of Mathematics Studies No. 34), (C. Shannon and J. McCarthy, eds.) pp. 129-153, Princeton Univ. Press.

Muller, D. E. [1954] Application of boolean algebra to switching circuit design and to error detection, *IRE Trans. El. Comp.*, **3**, 6-11.

Muller, D. E. [1956] Complexity in electronic switching circuits, *IRE Trans. El. Comp.*, **5**, 15-18.

Nelson, R. J. [1955a] Simplest normal truth functions, *Jour. Symbolic Logic.*, **20**, 105-108.

Nelson, R. J. [1955b] Weak simplest normal truth functions, *Jour. Symbolic Logic.*, **20**, 232-234.

Netherwood, D. B. [1959] Minimum sequential machines, *IRE Trans. El. Comp.*, **EC-8**, 339-345.

Paull, M. C. and Unger, S. H. [1959] Minimizing the number of states in incompletely specified sequential switching functions, *IRE Trans. El. Comp.*, **EC-8**, 356-367.

Quine, W. V. [1952] The problem of simplifying truth functions, *Amer. Math. Monthly*, **59**, 521-31.

Quine, W. V. [1955] A way to simplify truth functions, *Amer. Math. Monthly,* **62,** 627-31.

Reichenbach, H. [1947] *Elements of Symbolic Logic,* MacMillan, London.

Roth, J. P. and Jacobi, G. T. [1955] *A Topological Method for the Synthesis of Switching Circuits in n Variables,* General Electric Co. Report No. R55GL 345.

Roth, J. P. [1959] Algebraic topological methods in synthesis, *in Proceedings of an International Symposium on the Theory of Switching* (Pt. I) Harvard Univ. Press, Cambridge, Mass., pp. 57-73.

Rubinoff, M. [1959] Remarks on the design of sequential circuits, *in Proceedings of an International Symposium on the Theory of Switching* (Pt. II) Harvard Univ. Press, Cambridge, Mass., pp. 241-280.

Serrell, R. [1953] Elements of Boolean algebra for the study of information-handling systems, *Proc. IRE,* **41,** 1366-79. (Corrections [1954]. *Proc. IRE,* **42,** 475).

Shannon, C. E. [1938] A symbolic analysis of relay and switching circuits, *Trans. Amer. Inst. Elec. Engrs. Suppt.,* **57** (I), 713-723.

Shannon, C. E. [1949] The synthesis of two-terminal switching circuits, *Bell System Tech. Jour.,* **28,** 59-98.

Sheffer, H. M. [1913] A set of five independent postulates for Boolean algebras, with applications to logical constants, *Trans. Amer. Math. Soc.,* **14,** 481-488.

Slepian, D. [1953] On the number of symmetry types of Boolean functions of *n*-variables, *Canad. Jour. Math.,* **5,** 185-92.

Staehler, R. [1952] An application of boolean algebra to switching circuit design, *Bell System Tech. Jour.,* **31,** 280-305.

Staff of the Computation Laboratory, Harvard University [1951]. *Synthesis of Electronic Computing and Control Circuits,* Harvard Univ. Press, Cambridge, Massachusetts.

Turing, A. M. [1937] On computable numbers, with an application to the entscheidungsproblem, *Proc. London Math. Soc.,* Ser. 2, **42,** 1936-1937; Corrections, **43,** 544-546.

Turing, A. M. [1954] Solvable and unsolvable problems, *Science News,* **31,** 7-23, Penguin Books, Harmondsworth, England.

Urbano, R. H. and Mueller, R. K. [1956] A topological method for the determination of the minimal forms of a Boolean function, *IRE Trans. El. Comp.,* **EC-5,** 126-132.

Veitch, E. W. [1952] A chart method for simplifying truth functions, *Proceedings of the Association for Computing Machinery Meeting at Toronto, Ontario, 1952,* pp. 128-133, Assoc. Comput. Mach., New York.

Wang, Hao [1957] A variant to Turing's theory of computing machines, *Jour. ACM,* **4,** 63-92.

Washburn, S. H. [1949] Relay 'trees' and symmetric circuits, *Trans. Amer. Inst. Elec. Engrs.,* **68** (I), 582-86.

4. Circuit Descriptions of Switching and Storage Elements

In Chapter 3, the concept of a switching function and switching networks was introduced, and it was shown how sequential networks could be formed by a combination of storage elements and combinational switching networks. There are many physical elements available for the realization of both of these functions. Often, the same physical element can be utilized for switching, storage, and auxiliary functions such as power amplification by incorporating it into appropriate circuits. For example, vacuum tubes, transistors, and magnetic cores may all be used in both switching, storage, and amplification circuits. In this chapter descriptions will be provided of the most prominent of these circuits.

The functioning of switching networks and of complete digital computers can be appreciated even with a very limited knowledge of circuitry. However, a certain amount of circuit description will aid both in providing a better over-all orientation with respect to the subject and also a better appreciation of the problems associated with physical systems as contrasted to idealized systems of perfect elements. Whereas ideal elements can be interconnected without restriction, real elements cannot. Accordingly, different types of physical elements impose different restrictions on the logical arrangement of switching networks.

There is no attempt in this chapter to provide an exhaustive treatment of either switching or storage circuitry. This not only would consume an inordinate number of pages, but is unnecessary for the main purpose of this book, namely a presentation of the fundamentals essential to an understanding of the design and capabilities of digital computing machines. There is another cogent reason for limiting the discussion of computer circuitry. New components and techniques for their utilization are appearing on the scene at an amazing rate, and many, if not most, of the circuits common a few years ago have now been replaced. Most of the earlier machines used vacuum tubes, both as switching and storage elements. The first major change in a sequence of changes still continuing occurred when vacuum tube switching elements were replaced by semiconductor diodes. This was the situation when the writing of this book

was undertaken. Now, vacuum tubes have been replaced almost completely. Present machines utilize semiconductor diodes, transistors, or magnetic cores as switching elements, and transistors and magnetic cores as storage elements for small quantities of data. High speed and medium speed large capacity storage systems, described in Chapter 5, now almost universally use arrays of magnetic cores, and the surface of a magnetic drum or disc, respectively. Among more recent elements to appear, and which hold promise for faster switching and storage, are those that operate by exploiting the switching action between superconductivity and normal conductivity obtainable with certain materials at temperatures near absolute zero. Superconductive switching elements are described in Section 4.6 while superconductive storage elements are described in Section 5.4. Some other elements that have been investigated for computers operating beyond 100-Mc are described briefly in Section 4.7. Emphasis is placed on semiconductors (diodes and transistors) and on magnetic core devices because of their dominant position at this time. However, the principles and techniques of utilizing switching and storage elements that are presented should be helpful in exploiting new devices that may appear.

4.1. Systems of Circuit Logic

A compatible set of switching and storage circuits, adequate for the realization of any sequential switching network that may be called for in a digital computer is termed a system of circuit logic. In designing a digital computer, considerable effort is usually directed towards forming a system of circuit logic from a small set of standardized circuits. This yields a saving in engineering design effort and in the cost of manufacture, equipment maintenance and spare parts inventory. Several systems of circuit logic have been developed to date and some of the most outstanding ones will be discussed here. Among the items that must be considered when designing a set of circuits are: the selection of voltages or currents to represent the binary signals in various parts of the system, the design of circuits capable of a specified speed of response, the specification of permissible loading on the various circuits. In connection with the first item it should be remarked that whereas one set of voltages or currents may be used to represent the binary signals in one part of the system, others may be used in other parts of the system. The permissible loading on the various circuits is of considerable interest to the logical designer. Ideally, he would not like to have any restrictions placed on the interconnection of logical elements. When circuit considerations impose such restrictions he may have to reformulate his set of logical equations to a form realizable by the circuitry to be used. In general, for a given set of

building blocks, there is specified a set of restrictions on the permissible interconnections of the elements in the set. These restrictions are due principally to time delays introduced by various circuits, circuit interactions, and waveform degradations caused by passing through certain chains of elements.

The systems of circuit logic described in this chapter are classified under the headings of vacuum tube, transistor, and magnetic core systems. In each category, a number of different types of circuits and modes of operation are presented. However, certain arrangements can be used with more than one type of physical element. For example, the ac (dynamic pulse) system and the dc (asynchronous) system are described under vacuum tube systems because they were first developed using vacuum tube circuits and these vacuum tube versions are currently in operation. An ac system, originally based upon a combination of diode switching networks and a particular configuration of vacuum tube pulse amplification and regeneration circuits, can also be built with a similar set of transistor circuits. A dc system can also be constructed using transistors. At the University of Illinois Computer Laboratory, work is in progress on an asynchronous dc computer system which will use transistor circuitry and be approximately 100 times faster than the earlier ILLIAC computer.

There are a number of systems of circuit logic in which the combinational switching circuits are comprised of AND and OR circuits formed from semiconductor diodes, in a type of circuit referred to as a gate. These gates are widely used not only in computers but also in many specialized data processing units such as analog-to-digital converters and other peripheral equipment. Because of their wide application, diode gating circuits will be discussed separately in the sections following.

4.2. Gates

The term "gate" is often used for any of the elemental switching circuits of which combinational switching networks are composed. The term originated in electrical circuit terminology. The signals produced on an output line of a switching circuit were considered to be the signals on one of the input lines which had been permitted to pass through (i.e., gated) provided specified control signals were present on other input lines.

In so-called dc systems, the signals consist of two specified voltage levels. In ac systems, the signals consist of the presence or absence of a voltage pulse (of either polarity). There are also "mixed" systems where some of the signals are represented by pulses and others by voltage levels. Gates can be formed of either active elements, e.g., vacuum tube diodes,

triodes, multigrid tubes, or transistors, (all capable of amplifying signals) or from passive elements such as the diodes described in Section 4.2.1.

Among the most commonly encountered gates in digital computers are those that correspond to the Boolean AND and OR operators. In electrical circuit terminology the term "buffer" or "mixing circuit" is used for an OR circuit, and the term "coincidence gate" is used for an AND circuit. However, the term "gate" when used alone usually refers to a "coincidence gate".

An AND gate will produce a signal on its output line, if, and only if, there is a signal present on all inputs, of which there may be two or more. Some of the most widely used representations of AND gates are shown in Fig. 4.1. The representation on the left will usually be used in this text.

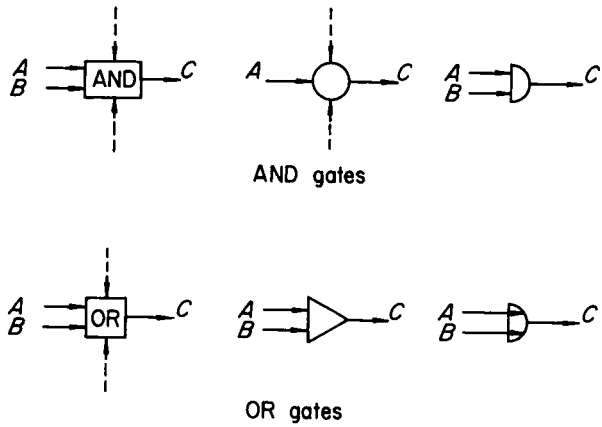


FIG. 4.1. Functional representations of AND and OR gates

The dashed lines indicate that there is no restriction relative to the sides to which input lines may be directed. There will also be no restriction relative to the side from which the output line may emanate. The equivalence of the electrical circuit viewpoint and the Boolean algebra viewpoint relative to a gate may be described with reference to this symbol. In electrical circuit terminology, it is said that a gate permits a train of pulses to pass from one of its inputs to its output provided specified signals (referred to as control signals) are present on the other input lines. In Boolean descriptions, 1 can be assigned to the presence and 0 to the absence of a signal. In Fig. 4.1 if B (assumed to be the control signal) has the value 1, the output $C = AB$ is equivalent to $C = A$. Therefore, the control signal B can be considered to let the input signal A pass through to the output.

An OR gate will produce a signal on its output line if there is a signal present at one or more of the inputs. Some of the most widely used representations of OR gates are shown in Fig. 4.1. The representation on the left will be used in this text. In electrical circuit terminology, the term "mixer" or "mixing circuit" is used to indicate that this type of circuit can be used to convert noncoincident trains of pulses on two or more lines to a single train of pulses on another line.

4.2.1. Diode Gating Circuits

Diodes are two-terminal devices exhibiting the property of rectification, i.e., the amount of current that passes between the two terminals depends not only on the amplitude of the voltage applied, but also on its polarity. The "ideal" diode represents an open or a short circuit depending on the polarity of the voltage applied. Among the more common diodes are vacuum tube diodes and those formed from semiconductors like selenium, germanium, and silicon. Because of their relatively large bulk, power consumption, and circuitry requirements, vacuum tube diodes have been completely replaced in digital computer circuits by semiconductor diodes. The latter are small, do not require the continuous dissipation of power as do the filaments of a vacuum tube, and have very simple circuit requirements. Selenium diodes are limited by their relatively slow switching action to frequencies less than 50 Kc. Germanium and silicon diode gates are operable at frequencies in the megacycle range, the attainable frequency of operation being a function not only of the diode's characteristics, but of the circuit in which it is incorporated.

The equivalent circuits of all the diodes discussed may be represented as shown in Fig. 4.2. In these schematics, (a) represents the equivalent circuit when a potential is applied in the forward direction, and (b) repre-

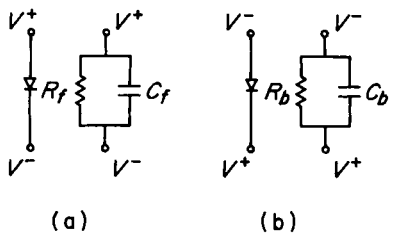


Fig. 4.2. Equivalent circuits of a semiconductor diode

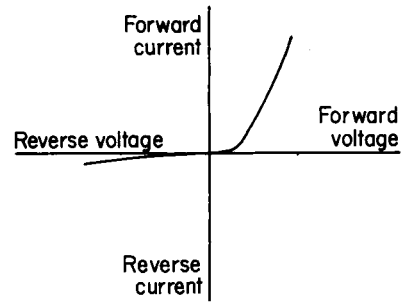


Fig. 4.3. Typical voltage–current characteristic of a semiconductor diode

sents the equivalent circuit when a potential is applied in the opposite direction. Figure 4.3 shows the flow of current in a semiconductor diode as a function of the magnitude and polarity of the applied voltage.

Both AND and OR gates may be physically realized by means of simple circuits utilizing diodes and resistors. The operation of these circuits depends upon the fact that when a voltage of one polarity is impressed across the terminals of a diode, it exhibits a very high resistance, R_b, (the so-called back resistance) and when a voltage of opposite polarity is applied, it exhibits a very low resistance, R_f (the so-called forward resistance). Figure 4.4 shows a schematic of semiconductor diode circuits that are used to realize AND or OR gates. Though only two inputs are

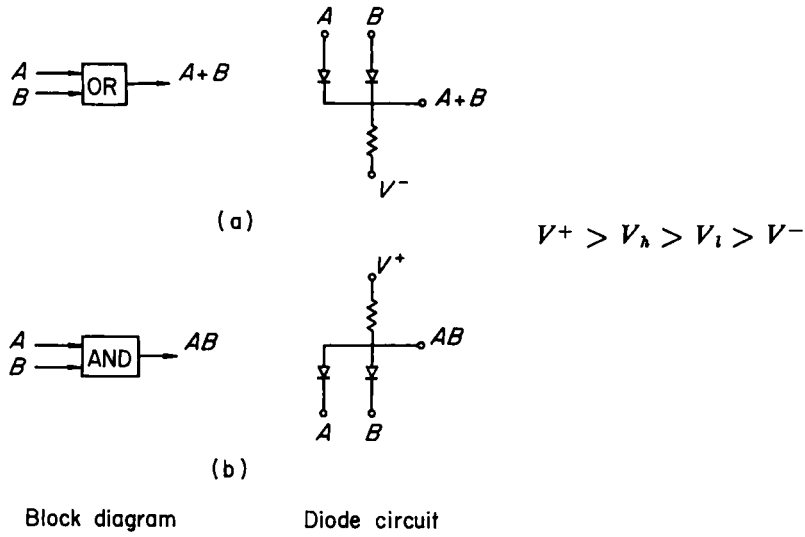


FIG. 4.4. Single level gates

shown for each gate, additional inputs can be accommodated simply by adding diodes in parallel. The circuits shown are intended to work with binary signals represented by two positive voltage levels. These voltages may be termed V_h and V_l, where the subscripts stand for high and low. In Fig. 4.4(a) if a voltage of magnitude V_l is applied to both inputs, each diode exhibits a high resistance which allows only a negligible current flow through the resistor. Therefore, the output voltage will be low. If V_h is applied to one input and V_l to the other, the diode connected to the input carrying V_h exhibits a low resistance while the other diode exhibits a high resistance. The net effect is that a substantial amount of current

can flow through the resistor to the terminal where V_h is applied. Since a diode conducting in the forward direction has a very low resistance, the voltage drop across it is negligible compared to the drop across the resistor, and therefore the output voltage will be approximately equal to V_h. If V_h is applied to both inputs, the voltage drop across the two forward conducting diodes in parallel is negligible so that the output voltage will be approximately equal to V_h. To summarize, the output voltage will be approximately equal to V_h whenever a voltage of magnitude V_h is applied to one or both of the inputs. If it is specified that V_h represents 1 and V_l represents 0, this circuit represents an OR gate. The operation of the AND circuit shown in Fig. 4.4(b) may be explained in a similar manner. In both of these circuits, if two negative, instead of positive, voltage levels are used, the AND gate for positive signals becomes an OR gate for negative signals, and the OR gate for positive signals becomes an AND gate for negative signals.

The problems of design in diode gating circuits are simple in principle. However, in actual practice they can become rather involved, especially in multilevel networks (described in the section following). Essentially the problem consists of specifying the two voltage levels corresponding to the values of a binary variable, selecting the two supply voltages, V^+ and V^-, and then determining resistor values such that the correct output voltage is produced for all possible combinations of input voltages. Also, a particular type of diode must be selected from the large number of different types available. The selection of operating voltages and diode types are not independent. For example, the reverse voltage (i.e., an applied voltage of polarity such that the diode exhibits a high impedance) that can be applied across a diode before it breaks down, the so-called breakdown voltage, varies with diode type. Also, although the characteristic curve shown in Fig. 4.3 is typical of many semiconductor diodes, there are some variations in the shape of the curve as well as the scale of the coordinates for different diode types. Another important factor in the design of high-speed circuits relates to the maximum rate at which a particular type of diode can be switched between states of high and low current conduction.

The preceding description of a diode gate circuit assumed not only the use of ideal diodes, but also the use of dc voltages for input signals and a no-load condition at the output of the circuit. The effect of nonideal diodes and loading will be considered in the description of multilevel gating circuits which follows.

4.2.2. MULTILEVEL GATING CIRCUITS

As described in Chapter 3, in the synthesis of a switching network,

it is a common occurrence for the output of a switching element to be used as the input to one or more others. When this occurs, the network is said to have more than one level. This discussion of multilevel gating circuits assumes that all networks are formed by interconnecting AND and OR gates. The level of a particular network is given by the total of the AND and OR gates in an AND–OR–AND– . . . or an OR–AND–OR– . . . chain. It may be determined either from the block diagram or circuit schematic by noting the largest number of AND and OR gates through which any input passes before reaching the output line. The number of levels may be determined, too, from the equation as follows. Represent an AND function by the notation $a(I, J, \ldots)$ and an OR function by $o(U, V, \ldots)$. If a Boolean equation is rewritten in this form, the number of levels is equal to the total number of parenthetical enclosures. For example

$$A(B + CD) = a\{A, o[B, a(C, D)]\}^*$$

indicating a three level gate.

In Fig. 4.5 are shown two examples of two-level diode gates. Only the AND–OR arrangement in Fig. 4.5(b) will be considered in detail, but similar remarks apply to the OR–AND arrangement. Assume that the supply voltages, signal voltage levels, and diode type have already been specified. Then, the only design problem remaining is that of determining resistor values. To minimize current requirements, they should be as large as practical. Also, they must allow the gating circuit to produce an output voltage of either V_h or V_l in accordance with the values of the signal voltages present at the input terminals.

For reasons which will become apparent, the resistor values in gating chains are determined starting at the load end. In Fig. 4.5(b), the value of R_1 may be determined by noting that, if all the inputs are at the level V_l, the current through R_1 must be sufficient to produce an output voltage, $V_{\text{out}} \leqslant V_l$. If there were no load, R_1 could have any value. When a load is present, it is in parallel with R_1. The resulting division of current flow limits the maximum allowable value of R_1. The value of R_2 may be determined as follows. Consider the point p in the circuit. The voltage at p must rise to V_h whenever A and B are both equal to V_h. If points A and B are left floating and if C and D are both equal to V_l, the voltage at p must be greater than or equal to V_h. If it were not, diodes D_5 and D_6 would be in the reverse direction when A and B were reconnected to V_h, and V_{out} would fail to rise to V_h. To assure that V_{out} can be pulled

---

* This operational notation for a function of other functions is sometimes referred to as the Polish notation because of its use in classic works by Polish logicians.

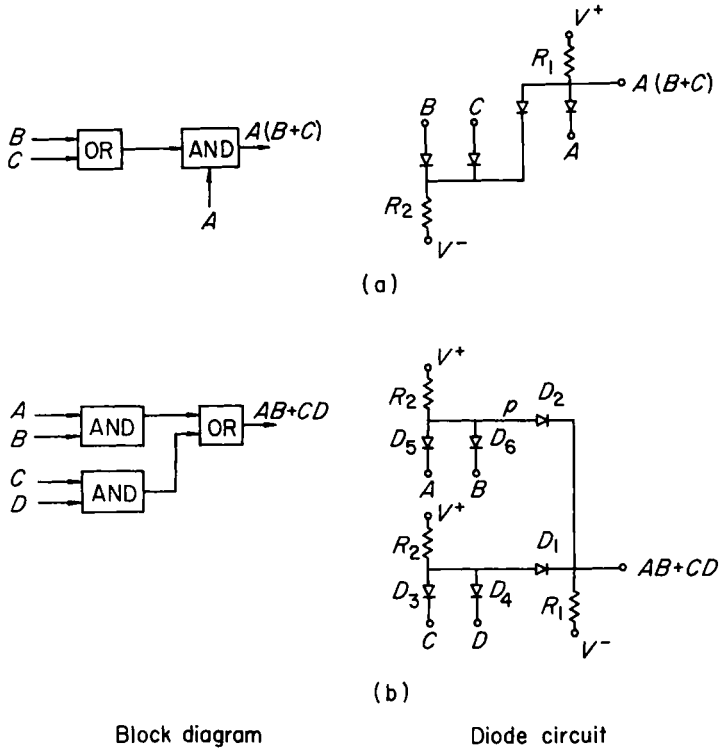


FIG. 4.5. Two-level gates

up to V_h, the following relationship must exist

$$R_2 \leqslant \frac{V^+ - V_h}{V_h - V^-} R_1.$$

The same relationship holds between R_1 and each R_2 in the event that there are more than two inputs to either the OR circuit or to any or all of the AND circuits.

The values of the resistors in higher level gates may be determined by following the type of analysis used for the two-level gate. The analysis proceeds by starting at the load end of the gate and determining the resistor values for higher levels in sequence. For a three-level OR–AND–OR network, the maximum value of the third level resistor(s) is given by

$$R_3 \leqslant \frac{nR_1R_2(V_l - V^-)}{nR_1(V^+ - V_l) - R_2(V_l - V^-)}$$

where n is equal to the number of inputs to the first level OR circuit.

Since the resistors in any given level draw current in a direction opposite to that in the next lower level, enough current must be drawn to

overcome the effects of the next lower level. The maximum permissible values of the resistors decrease rapidly as the level increases, requiring that larger currents be supplied by the input signals.

In the preceding discussion, it was tacitly assumed that the diodes had a zero forward resistance. However, as the curve in Fig. 4.3 indicates, this is not the case. As a result, an attenuation in voltage swing will occur between the input and output of a gate, and spurious signals may be introduced. Both of these effects occur because the current distribution through the diodes will not be the same for the conditions that are to produce a high and low output voltage, respectively, nor even for all the input configurations that should produce the same output voltage. As an example, consider first a multi-input OR gate. If all the inputs are at a high voltage, the current passing through the gating resistor will be equally distributed through all the diode paths (neglecting small variations in the diode forward resistances). The output voltage will be determined by the voltage division between the parallel diode forward resistances on the one hand and the gating resistor on the other. If some of the inputs are at a low voltage, the current from the gating resistor will only pass through those diodes whose inputs are high. In addition, there will be a reverse current from the diodes whose inputs are low through the diodes whose inputs are high. As a result, the voltage drop between the input and output will increase, whereas, according to the logical relationship desired, it should not. Now consider a multi-input AND gate. If all the inputs are at a low voltage, the output voltage will be determined by the voltage division between the parallel diode forward resistances and the gating resistor. If any input voltage becomes high, the corresponding diode will have a reverse voltage impressed upon it and therefore will exhibit a high back resistance. In addition, there will be a current flow from the diodes whose inputs are at a low voltage to the diode whose input is at a high voltage. As a result, the output voltage will rise slightly.

In both of the preceding examples, if the amplitude of the spurious signals generated is below a certain level, depending on the characteristics of the circuitry, they will not be detected by the system. In any event, such effects may be minimized by selecting diodes with very low forward and high back resistance. In multilevel circuits there is some compensation for the attenuation in voltage swing because for an AND circuit the shift occurs in a positive direction while for an OR circuit it occurs in a negative direction.

Another item not previously considered is that the diode back resistance, while very large, is not infinite. In a single level circuit this does not affect the permissible values for the gating resistor. However, it places

an extra load on the input signal. For example, in the OR gate of Fig. 4.4, if A is at V_h and B at V_l, current will flow from A to B through R_f of the first diode and R_b of the second. The sources of the input signals must be capable of supplying this current. In multilevel networks the effect of the diode back resistance is to reduce the maximum permissible values of the gating resistors.

The actual load to be placed on a given network must also be considered when determining the values of the gating resistors. The resistive part of the load may be treated, by Thevenin's theorem, as a resistor returned to a supply voltage V_s. If the first level of switching is an OR gate, the load can serve as the gating resistor provided V_l is positive with respect to V_s. If V_s is positive with respect to V_l, the maximum permissible value of the first-level gating resistor may be found by a procedure similar to that described earlier for determining a second-level resistor. In solving for a second-level resistor under actual load conditions, the first-level resistor must be replaced by the Thevenin equivalent of the first-level resistor and the load.

4.2.3. Voltage and Current Requirements in Gating Circuits

From the preceding discussion of multilevel diode gates, it is evident that the current required to drive the input lines increases rapidly with the number of levels. The current increase is a direct function of power supply and resistor tolerances, the number of diodes, and the signal voltage swing. The current increase is less for higher voltage supplies, but the power dissipation in network resistors becomes greater. As usual, a compromise must be made between increased power dissipation and increasing current in high level gates.

The current required to drive a multilevel circuit can be reduced by choosing V^+ and V^- such that $(V^+ - V_h)$ and $(V_l - V^-)$ are much larger than $(V_h - V_l)$. Though a large value of $(V^+ - V^-)$ facilitates switching action (since the absolute change in voltage will be greater in a given charge or discharge time), there is more power dissipation in the gating resistors and a risk that, in the event of an accidental open circuit, voltages in excess of the breakdown voltage may appear across diodes.

A major deficiency of diode gates is that their outputs cannot be heavily loaded. Since a diode gate does not constitute a constant current source, current amplifiers such as cathode followers or emitter followers must be incorporated in each two-level or three-level circuit. After propagation through several gates and current amplifiers, the input signals must have their voltage amplitude restored by means of voltage amplifiers. Because diode gates have a relatively low input impedance, it is necessary

that they be driven by a relatively low impedance (constant current) source.

Since the power consumption of switching circuits increases as the square of the voltage swing, the latter should be as small as possible consistent with reliable operation. However, at low signal levels the voltage drop across a diode becomes appreciable compared to the signal swing. For example, if a diode has a 0.25 volt forward voltage drop, and signal voltage swings of only 2 volts are used, there will be a 12.5% level shift of the signal through the diode. Also, at megacycle frequencies large currents are required to switch diode logic circuits, thus limiting the number of circuits that can be driven by one current amplifier. In Section 4.4.2.2, the use of transistor gates is described. These gates can be adequately switched by small signals, and a large number of them can be driven by a single transistor current amplifier.

In computing the load current to be supplied by a flip-flop in a large sequential network, account should be taken of the case where both outputs of a flip-flop drive a combinational circuit which is also driven by another flip-flop. In this case, the load to be supplied by the other flip-flop will be diminished because of the aid that is always received from the first. This type of situation is referred to as current sharing. When current sharing is taken into account, the total current drain on a flip-flop will be found to be less than if the sum of currents to each circuit were considered independently. For detailed descriptions of methods of computing currents and resistor values in diode networks, the reader is referred to Gluck *et al.* [1953], Hussey [1953], Scobey *et al.* [1956], and Yokelson and Ulrich [1955].

4.2.4. Switching Speed in Diode Gates

The switching speed obtainable in diode gates is adversely affected by a number of factors. The most important of these are circuit capacitances and diode recovery time. The effects of load capacitance, stray wiring capacitance, and diode interelectrode capacitance may be determined by conventional electrical network analysis. The net effect is that the value of the gating resistors must be reduced from the maximum value permissible when these capacitances are ignored. When a semiconductor diode has been conducting heavily in the forward direction, and the applied voltage is suddenly reversed, a time lag, referred to as diode recovery time, occurs before the diode assumes its normal value of back resistance. However, recovery time for some newer diodes is under 10 nanosec (where 1 nanosec $= 10^{-9}$ sec), so it need not be a serious limitation to high speed circuits. The finite back resistance of a diode also affects

switching speed. The net effect is that for a given switching speed, the value of the gating resistors must be reduced. The fact that the forward resistance of a diode is not actually zero has little effect compared to the tolerances in resistor values and uncertainties in circuit capacitances in a gating network.

4.2.5. PYRAMID GATES

Frequently it occurs that two or more switching functions required in the construction of a network have a number of common inputs. For example, suppose $f_1 = ABC$ and $f_2 = ABCDEF$. A saving in the total number of switching elements required may be realized by using an arrangement such as shown in Fig. 4.6(a), referred to as a pyramid. Though it is actually a two-level gate (AND–AND) each section can be designed independently. It is only necessary that the source of each input signal be capable of supplying the current drawn by the resistor in each section. A two-level OR–OR pyramid is shown in Fig. 4.6(b).

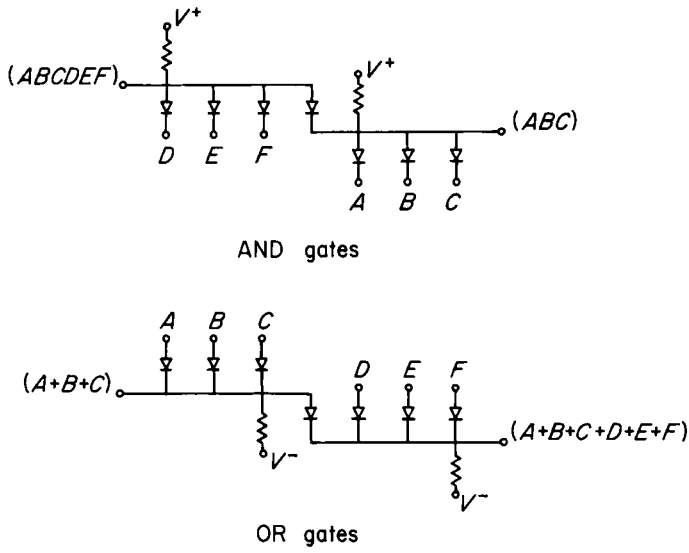


FIG. 4.6. Pyramid arrangements

Though, in general, a multi-input gate represents a single level network, an expression like $A + B + C + D$, for example, might be used in the two-level form $A + (B + C + D)$ if $(B + C + D)$ were available.

4.2.6. Algebraic Reduction of Higher Level Gates to Lower Level Gates

As we have seen, when all circuit parameters are considered and adequate safety tolerances included, the design of multilevel networks becomes complicated. Since the current at each level increases, and the values of voltages and resistors become more critical, more than three levels are seldom used. In fact, the number of levels is often limited to two.

Higher level gates may be reduced to lower order ones by multiplying out the factors in the higher level expressions. As an example, consider the expression, $f_4 = AB\,[(C + D + E)\,(\bar{C} + D + \bar{E}) + FG]$. As written, this represents a four-level network. Multiplying out the terms in parentheses yields an expression that represents a three-level network: $f_3 = AB\,[\bar{C}E + C\bar{E} + D + FG]$. Performing the indicated multiplication yields an expression that represents a two-level network: $f_2 = AB\bar{C}E + ABC\bar{E} + ABD + ABFG$. If the functions f_4, f_3, and f_2 were mechanized by diode networks, they would require 16, 14, and 19 diodes, respectively. As a rule, a lower-level network will require more diodes than a higher-level one. That this did not occur in going from the fourth- to the third-level network in the example is due to the simplification obtained as a result of the special nature of the terms in the parentheses. To see what happens in the worst case, replace the terms in the parentheses as follows: $f_4 = AB\,[(H + I + J)(K + L + M) + FG]$. In this case the corresponding three- and two-level networks will require 34 and 50 diodes, respectively.

4.2.7. Reduction of the Level or Number of Elements in a Combinational Circuit by the Use of Storage Elements

In Section 4.2.6 we saw that, as a rule, a lower-level gate produced from a higher-level one by algebraic manipulation requires more gating elements. In special cases it is possible to reduce both the level of combinational networks and the number of switching elements required by the introduction of auxiliary flip-flops. As an example consider the expression

$$f = (AB + CD + EF)(KL + MN + P).$$

In this form, 18 diodes would be required to generate f in a three-level combinational network. If all the terms were multiplied out to produce a second-order expression, 42 diodes would be required. We will consider

now the effect of introducing two auxiliary flip-flops, U, V. The term $(AB + CD + EF)$ will be used to set one flip-flop and the term $(KL + MN + P)$ will be used to set the other. Normally, there is a unit time delay from the instant at which an input signal occurs to the time at which its effect is observed at the output of a flip-flop. Therefore, time must be introduced into the expression for f as shown

$$f_t = (AB + CD + EF)_t\,(KL + MN + P)_t = (UV)_{t+1}.$$

Thus the original expression can be mechanized by the use of two set-reset flip-flops and two two-level combinational networks, one with nine diodes and the other with seven, plus the input circuits for carrying the reset signals to the flip-flops. In practice a flip-flop is triggered at times within each operating cycle defined by signals derived from a clock. The timing signal is combined with other terms in the flip-flop input equation. For example, the input equation for setting the flip-flop, U, might be $(AB + CD + EF)T_n$, where T_n could refer to a single clock signal, or a function of several timing signals.

Note that whereas a combinational circuit alone would indicate at time t whether the function f were true at time t, the sequential circuit does not provide this indication until time $t + 1$. For assurance that the value of $(AB + CD + EF)$ is compared with the value of $(KL + MN + P)$ that occurred at the same time, both flip-flops are reset after each comparison. Thus, the restrictions on the use of such auxiliary flip-flops are that the delay introduced be tolerable (normally, a system can be designed to accept such fixed delays), and that there is time to reset the flip-flops before introducing a new set of input signals.

4.2.8. Pulse-Pedestal Gate Circuit

A requirement that often arises in digital computer systems is the gating of a voltage pulse signal by another signal in the form of a dc voltage level. Diode gate circuits for positive and negative voltage signals are shown in Fig. 4.7. In both cases, the output pulse occurs during the period of coincidence of the input signals.

In Fig. 4.7(a) a dc supply in the load holds the output line to V_h.

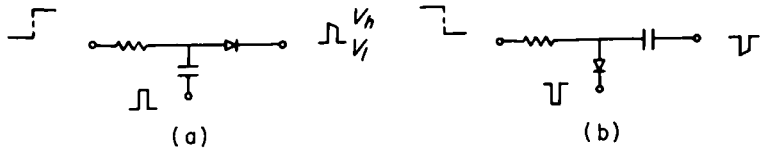


Fig. 4.7. Pedestal-pulse gate circuits

When the input to the resistor is V_l, the diode (which is back biased) blocks passage of an applied positive pulse of amplitude less than $(V_h - V_l)$. However, when the input to the resistor is V_h, the diode is biased in the forward direction, allowing passage of the pulse to the output. In Fig. 4.7 (b), if the input to the resistor is V_l, the diode is back biased and blocks passage of the applied pulse. When the input to the resistor is V_h, the diode (now biased in the forward direction) allows passage of the applied pulse.

The need for pedestal gating arises whenever a system calls for dc gating networks to be used with binary storage elements which require pulse inputs. The characteristics of different flip-flop circuits are described later in the chapter. When the triggering of a flip-flop from one stable state to another calls for input signals in the form of voltage pulses rather than levels, the pulse pedestal gate circuit can be used to convert the dc voltage level outputs of a dc gating network to the pulse type signals required as flip-flop inputs.

Pulse pedestal circuits are commonly used as a means of controlling the transition of a synchronous computer from one superstate to another. The outputs of all dc gates to be used for triggering flip-flops are combined in a pulse-pedestal circuit whose pulse input is derived from a clock pulse generator which supplies pulses at regular intervals. Since the pulse-pedestal circuit can have an output only if a clock pulse is present, the states of the flip-flops cannot be altered except at the time of occurrence of clock pulses.

4.2.9. Generation of Complementary Functions

If a switching function, f, is synthesized by some combination of elementary functions, then the complementary function, $\bar{f}$, can always be synthesized by some other function of the variables involved. In general, the complementary function can be realized physically only if some device is available to provide the complement of the switching variables. A flip-flop with two output lines provides both a signal and its complement and, therefore, permits the generation of $\bar{f}$ by means of AND and OR circuits alone. For example, if $f = (AB + AC)$, then $\bar{f} = (\bar{A} + \bar{B}\bar{C})$ can readily by generated by combinations of AND and OR circuits, provided $\bar{A}$, $\bar{B}$, and $\bar{C}$ are available. There are times, though, when it may be desirable to limit the number of elementary signals in complemented form used in constructing a switching function. These are occasioned by practical circuit considerations. Even though both the complemented and uncomplemented signals are available from a flip-flop, a power amplifier may be required for each of the outputs that is to be used as an input

to many other circuits. Also, additional wires are required, which may be a disadvantage if the signals have to be transmitted an appreciable distance.

It is often more convenient and simpler to form the complement of a complex signal, i.e., one developed from a large number of elementary signals, by applying it to the input of a suitably designed voltage amplifier, which has the characteristic that when a signal is applied to its input an amplified and inverted form of the signal appears at its output. Inverter circuits, which are physical realizations of the complement operator, are described in the sections on vacuum tube and transistor circuits which appear later in this chapter. Because inverters may introduce serious time lags and distortion of wave forms, especially if one or more of them are in cascade within a multilevel gate, it is often desirable to limit their use in switching networks. This may be done by transforming a given equation. For example, replacing $\overline{(A + B)}$ by $\bar{A}\bar{B}$ eliminates the need for an inverter (provided the switching variables are available in complemented as well as uncomplemented form).

Often, the problem of optimization of a switching network is equivalent to minimizing the number of elementary switching circuits, usually AND and OR gates, required for the realization of specified functions. When, in addition, it is desirable to limit the number of complemented signal sources or the number of inverters, derivation of an optimum circuit is not as clear cut.

4.3. Vacuum Tube Systems of Circuit Logic

The early electronic digital computers used vacuum tube circuitry extensively for gating and storage. However, after an evolutionary period of about ten years' duration, tubes were replaced more and more by other devices. At the present time practically all new machines under development utilize combinations of solid state devices for the functions of circuit logic (as well as for the main store).

Because of their historical importance, and the fact that a large number of machines using vacuum tubes in their logical circuitry are still in operation, a brief description of vacuum tube gating and storage circuits will be provided. One of the basic circuits is the familiar inverter circuit, shown in Fig. 4.8. To operate this circuit as a binary switch, the input signal is chosen to either cause the tube to be fully conducting or to be cut off. The function of the voltage divider is to scale down the output voltage of one circuit to the proper level for input to the grid of another circuit. The capacitor improves the circuit response time. An increase in voltage in the positive direction on the grid causes the tube to conduct more current, thereby increasing the voltage drop across the load resistor

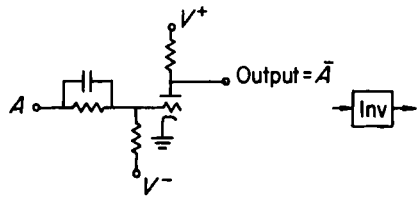


FIG. 4.8. A vacuum tube inverter

and reducing the output voltage. A decrease in voltage on the grid has the opposite effect. If a sufficiently large negative voltage is applied to the grid, the tube will be cut off, and the output voltage will be equal to the positive supply voltage. It is apparent that if the two values of a binary variable are represented by a pair of voltages, then application of one voltage to the input of the inverter can cause the complementary voltage to be produced at the output. Whether a given voltage represents a 1 or a 0 is at the discretion of the designer. He may, in fact, reverse the convention from place to place within a machine if by so doing he can effect simplifications in the over-all circuitry. Of course, account must be taken of what conventions are used in any section. In Fig. 4.9 a circuit

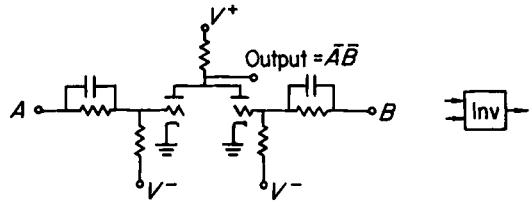


FIG. 4.9. A parallel inverter

comprised of two inverters sharing a common load resistor is shown. It serves as an inverting OR gate for positive signals and an inverting AND gate for negative signals. Both the AND and OR functions can be generated by combinations of inverter and parallel inverter circuits, as shown in Fig. 4.10.

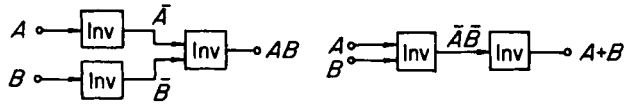


FIG. 4.10. AND and OR gates formed from inverters

A static flip-flop circuit can be formed from two inverters by regeneratively coupling the output of each to the input of the other. This circuit, based on the Eccles–Jordan multivibrator circuit, is useful both in machines using dc coupled gates as well as in machines using ac coupled gates. The basic nature of the vacuum tube static-flip-flop circuit is shown in Fig. 4.11. The circuit shown can actually be monostable or astable as well as bistable, depending on the impedances Z_1 and Z_2. The circuit will be bistable only if both Z_1 and Z_2 contain dc paths. Typically, Z_1 and Z_2 are identical parallel RC branches. The use of the circuit affects its design. For example, as a counter (see Chapter 6) the circuit requires symmetrical inputs. When used as a stage in a shift register, separate inputs would be required for data pulses and shift command signals. Usually, a flip-flop is not used to drive a diode gating circuit directly because spurious pulses coupled from one diode input line to other input lines may cause unwanted triggering of the flip-flop.

Theoretically, any combinational switching function can be derived from the use of the inverter and the parallel inverter circuits. This is because they provide the operations of Boolean complementation and addition which, as stated in Chapter 3, are adequate for generating any Boolean function. However, a number of other types of electronic circuits are available for various practical purposes. One of the most important of these circuits is the cathode follower, which is a physical realization of the "single identity" operator E_2 in Table 3.6. However, it is not used as a switching element, but for other purposes. Its high effective input impedance (compared to an ordinary amplifier) and a low effective output impedance (from 200–1000 ohms), make it useful as a current amplifier and an impedance matching device for coupling a high impedance circuit to a low impedance one. Figure 4.12 shows a schematic of a cathode follower utilizing a triode. Use of a pentode in such a circuit would provide a lower input capacitance and a higher gain. However, the

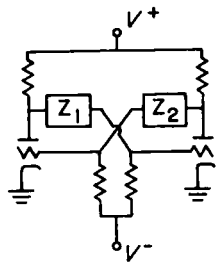


FIG. 4.11. Basic vacuum tube static flip-flop circuit

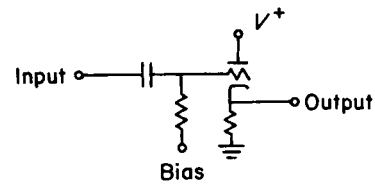


FIG. 4.12. Basic cathode follower circuit

pentode circuit is not as satisfactory for handling large input signals. In practice, cathode follower circuits are better suited for driving diode OR gates whereas inverters are better for driving diode AND gates. An OR gate for positive signals can be realized by a circuit comprised of two cathode followers sharing a common load resistor. When both tubes are fully conducting, the output voltage is high. Also, if either tube is cut off, there is only a negligible drop in output voltage. When both tubes are cut off, the output voltage drops to the value of the negative supply voltage.

4.3.1. The Diode Gate, Flip-flop System

In this system of circuit logic, diodes are used for AND and OR gates, inverters for complementation where desirable, and vacuum tube flip-flops for storage. The output signals of the flip-flops are coupled to the inputs of cathode followers, which are provided both to isolate the flip-flop from its load, and to provide the current source called for by the diode gates. Each steady state output signal of the switching network is used as one of the inputs to a pulse pedestal gate in the appropriate input circuit of a designated flip-flop. The other input to all of these pulse-pedestal gates comes from a clock pulse source. This arrangement places the entire system under control of the clock, for no flip-flop can be triggered except at times when clock pulses appear.

4.3.2. The Pentode Gate System

Multigrid tubes have also been used as gating elements in vacuum tube computers. Not all multigrid tubes can serve as practical gating elements because, in general, the different grids have different quantitative effects on the plate currents and therefore the signal voltages applied to them would have to be adjusted accordingly. One tube in which the control and suppressor grid each have approximately the same degree of control on plate current is the 6AS6. A specially designed gating tube, the GL-5915-A, has two independent control grids, and it can be utilized as a two input inverting AND gate for positive signals. The cut-off voltage is the same on both grids, and the tube is normally cut off by bias voltages applied to these grids. The application of two appropriately large positive signals to both grids simultaneously causes the tube to conduct, producing a negative output signal at the plate.

Logic circuitry using pentodes for gating was developed at MIT for the Whirlwind Computer. The pentode circuit, shown in Fig. 4.13 operates as an AND gate in which pulse signals applied to the grid are gated under

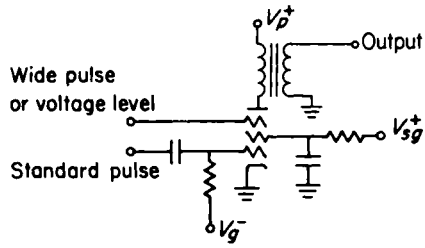


FIG. 4.13. Pentode pulse gate

the control of a dc gating signal or a wide pulse applied to the suppressor grid. A positive output pulse is produced at the point indicated when the gating signal and the input pulse are both positive and sufficiently large. An important characteristic of this circuit is that the output pulse can be made of suitable shape and amplitude to drive other pentode gates directly.

In a circuit logic system built around a pentode gate, the dc outputs of flip-flops would be used as the inputs to the suppressor grids of the pentode gates. The pulse outputs of these gates could be used either as inputs to diode OR gates or as inputs to flip-flops. In the latter case pulse transformers could be used at the inputs to the flip-flops in order to obtain negative pulses, which are more suitable for triggering flip-flops. The outputs of the diode OR gates can be used as the pulse inputs to other pentode gates or as inputs to the flip-flops.

Asynchronous and/or synchronous operation may be used with this system of circuit logic according to which produces a desired function with minimum circuitry. In asynchronous operation each network could be activated either by a start signal or an end of operation signal from another network.

4.3.3. THE AC SYSTEM

The so called ac system of circuit logic was developed at the National Bureau of Standards. It is used in their SEAC and DYSEAC computers, and in the MIDAC and MIDSAC computers built at the University of Michigan. In this type of system all signals are in the form of pulses, i.e., there are no signals in the form of dc voltage levels. The system includes diode gates for generating the AND and OR functions, pulse transformers for producing inversion, and electromagnetic delay lines for storage. The nonlogical, but essential, function of power amplification is provided by a vacuum tube.

This system of circuit logic is essentially formed from only one type of standardized unit, which is a combination pulse gate and regeneration circuit. This circuit, sometimes referred to as a pulse repeater, is shown schematically in Fig. 4.14. When appropriately combined with delay units

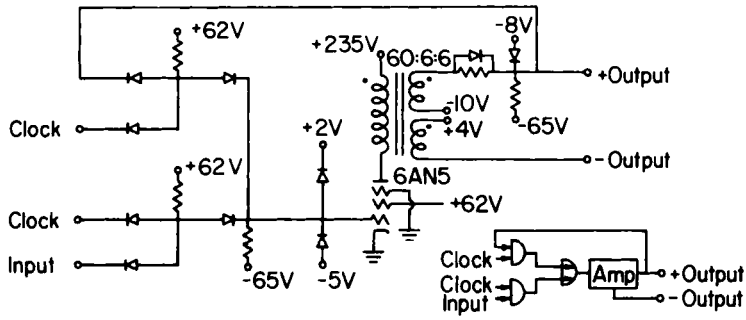


FIG. 4.14. Circuit and block diagram of a pulse gate and regeneration circuit for an ac system

it serves as a dynamic flip-flop. At the left end of the schematic are diode AND gates whose output is combined in an OR gate. (Though only two AND gates are shown, a large number of multi-input AND gates would usually be included in the circuit.) The clock signals applied to the AND gates serve to synchronize all input pulses. The principles involved in the design of diode switching circuits for pulse inputs are essentially the same as for dc level inputs, though considerations pertaining to switching speed are more important. Because of the various capacitances in a system, the signal that appears at an input may be degraded both in shape and amplitude. To limit this degradation to tolerable limits, one may regenerate the pulse after each small switching network. Pulse reshaping and synchronization to the timing of the clock waveform is achieved as follows: The signal input applied to the lower AND gate overlaps the leading edge of the clock waveform, thus assuring an output beginning with the leading edge of the clock. This output (clamped between +2 and −5 volts) is amplified and fed back to the input of the upper gate. After the signal input to the lower gate has decayed, this delayed and amplified signal sustains the output of the circuit until completion of the clock signal. In detail, the regeneration process is as follows. The input pulse to be regenerated is applied at the point shown at a time before the clock pulse is positive going. The signal passes through the OR gate to the grid of the tube, causing it to conduct. This produces a negative pulse at the plate of the tube and a positive pulse at the point shown on the secondary

of the transformer. This output pulse is fed back to the input of the tube via another AND gate, and the tube is kept in a state of conduction as long as the clock pulse is present. Each regeneration circuit derives large current amplification from a 10–1 stepdown turns ratio in the transformer. As a result, the output can drive up to 12 AND gates of other repeater circuits. (For a detailed description of this circuit the reader is referred to the articles by Elbourn and Witt [1953] and by Haueter, Alexander and Greenwald [1953].)

Inversion is accomplished by use of the negative output pulse of the transformer in the regeneration circuit. When such pulses are applied to any of the inputs of an AND gate, the effect is to inhibit the generation of a positive output signal. A two-input AND gate with one inhibiting input is a physical realization of the inhibiting switching function (see F_3 or F_5 in Table 3.7). It may be considered as a type of AND gate in which an output is not produced if certain control signals are present. Schematically, an inhibiting input to a gate is usually designated by a small circle placed on the input line where it touches the function box.

As indicated earlier, a dynamic flip-flop is used in this system. It is formed from the repeater circuit shown in Fig. 4.14 by returning the positive output of the transformer via a delay element to the input of a third AND gate connected as shown in Fig. 4.15. To synchronize the flip-flop's operation with the internal clock of the computer, a "start" pulse is applied to the 1 input, causing a stream of clock pulses to be recirculated via the delay line and the lower AND gate. To set the flip-flop to the 0 state, a pulse is applied to the inhibiting input of the AND gate, labelled in Fig. 4.15 as the 0 input. This halts the recirculation. The flip-flop just described is essentially that shown in Fig. 3.15(a).

The basic timing source in the National Bureau of Standards system is a 1 Mc sine wave. Since the input-output delay of the pulse gate and regeneration circuit is less than .25 μsecs, the basic timing waveform is distributed in four phases, 90° apart. When connecting circuits in cascade, successive ones are driven by successive phases of the clock. The net effect of this phasing scheme and the regenerative connection of the circuit is to insure that the signal inputs to a circuit clocked by phase n (which are restricted to the outputs of circuits clocked by phase $n - 1$) are present before the appearance of clock phase n and that clock phase n is present after the signal inputs have decayed as assumed in the description of Fig. 4.14 on page 121. Among practical difficulties associated with this system of circuit logic is the fact that one must keep track of the proper clock phase for each circuit and distribute the clock waveforms accordingly. Also, because multiple clock phases constitute an addi-

tional design parameter, the design effort to minimize the total number of circuits employed, or to minimize delays in propagation of pulses through cascaded circuits is increased. As far as economy is concerned, the components saved by a minimum requirement for separate amplification circuits are offset by the number of diodes used for other than circuit logic purposes.

4.3.4. The Asynchronous, dc Coupled System

In the logical description of combinational networks, it is usually assumed that an input variable has an instantaneous effect on the output of the network. In practice, unwanted delays may produce so-called hazards in the transient behavior which can result in malfunctions when the combinational circuits are incorporated into a sequential network.

In a synchronous computer, the elementary arithmetic and logical operations occur at fixed intervals defined and controlled by the clock. The over-all speed is determined by the expected response time of the slowest elements under estimated worst case conditions. The problem of hazards does not normally occur in such a system because the interval between successive clock pulses is specified to be long enough for transients to die out.

Generally speaking, in asynchronous systems each new operation is initiated by a completion signal produced by another group of circuits after the execution of the preceding operation. Therefore, individual switching operations do not require a predetermined duration corresponding to the maximum time required by any of them, but are determined solely by the electrical parameters of the circuit performing the operation. As a result, greater over-all speed is obtainable since each new operation can begin immediately upon completion of the preceding one.

There are varying degrees of asynchronous operation. For example, in an elementary form, a new completion signal is simply produced by routing the preceding completion signal through a delay whose magnitude corresponds to the maximum time required for a given set of circuits to operate. In another form, the operation of each set of circuits is examined by a checking circuit that provides completion signals only when the set of circuits completes its function. Finally, the asynchronous feature may be at the level of individual logic elements. In asynchronous circuits where there are no clock pulses, the signal propagation time through chains of elements is limited only by the response time of the elements and the over-all speed is determined by the average speed of the components.

The speed of synchronous circuits is usually indicated by specifying the clock frequency. A convenient measure of speed in an asynchronous

system is the operation time. This is the interval from when an input signal to a circuit reaches a critical value to when the output signal reaches a critical value.

As indicated earlier in this section, the new state to which a sequential network advance may depend upon the delays encountered in various paths within the network. To eliminate the hazards presented by the possibility of "races" to new states in asynchronous circuits, certain techniques have been worked out.* While considerable additional circuitry may be called for to eliminate hazards in the general case, the amount may be reduced appreciably or omitted if certain states or sequences would not occur naturally, or if external delays may be introduced economically.

Asynchronous systems are associated with dc coupled circuits as opposed to the ac coupled circuitry found in synchronous systems. In an ac coupled system, capacitors or transformers may be used to interconnect logical elements. In the dc coupled system not only are the elements interconnected by means of resistive networks but each element, e.g., a flip-flop, is also dc coupled internally. The signals in dc coupled circuits are normally in the form of one of two voltage or current levels, rather than a pulse or no pulse at specified times as in clock controlled systems, and as previously indicated, the steady state output signal of a logical block is coupled directly to the inputs of other blocks. The use of dc coupled circuitry in an asynchronous system makes proper operation independent of variations in shape of input waveforms and does not require strict control of propagation time to insure arrival of these waveforms in coincidence with a clock signal. Servicing is simplified because a dc coupled asynchronous machine can be put in a state of static equilibrium for as long as desired and its operation checked by an inspection of the steady state voltages at strategic points. Completion circuits may be included to detect failures in operation of other circuits, thereupon causing the machine to stop and indicate a malfunction. Another reason why asynchronous dc coupled circuitry can be faster, is that while in capacity coupled circuits a change in signal level must be followed by an inverse change to reach equilibrium of the capacitor, in a dc coupled circuit the signal needs to change in voltage in only one direction.

Because the operation of an asynchronous system can be made independent of the relative speeds of its elements, correct operation may be obtained without the need for matching speeds and without synchronizing signals. This is especially important when the individual circuits are so

* Huffman, D. A. [1957], Design of hazard-free switching circuits, *J. ACM*, **4**, 47-62. Unger, S. H. [1959], Hazards and delays in asynchronous sequential switching circuits, *IRE Trans. Circuit theory*, **6**, 12-25.

fast that the time required for the flow of information from one part of the computer to another is comparable to the operation times of the elements themselves. For example, signals are delayed by about 1 nanosec/foot, and transistor computer circuits with operation times less than 10 nanosec have been built. In very high speed systems, such as that under development at the University of Illinois, asynchronous circuitry offers an important advantage by not requiring precise knowledge of intra-system transit times to assure correct operation. On the other hand there are certain disadvantages. For example, an asynchronous system requires considerably more logical elements than a synchronous system because of the circuits required to generate completion signals and hold information while it is in transit, even after considering the saving resulting from the absence of circuitry for a clock and its gates. Also, an asynchronous system introduces engineering design problems because of the drift normally encountered with dc coupled circuits. However, the problem of drift in a switching circuit is considerably less than in a linear amplifier.

The elimination of hazards from asynchronous circuits makes them speed independent* in that correct operation does not depend on the relative speeds of their elements. Speed independent circuits allow a special type of completely asynchronous operation in which information can continue to flow only when all preceding elements in a chain have reacted to it. To meet the conditions of speed independence, individual logical elements must be specially designed and so-called last moving points provided to simplify the design procedures. A last moving point is a location in a circuit which by its new output gives proof that a new state of a circuit has been reached. However, because these circuits need not respond within a fixed interval, as circuits in synchronous systems must, reliability of operation is improved. Relative insensitivity to deterioration of components and variations in circuit parameters, which may also be used to relax requirements for uniformity in component specifications, often justifies the extra components used to eliminate hazards in asynchronous circuits.

With speed-independent networks within an asynchronous computer signal changes need not occur in a definite time sequence. Parallel actions can occur without giving rise to "race" conditions if logical elements are incorporated which have the logical property of producing an output only when all of several incoming signals have appeared. This output can be

* For formal definitions of speed independence see: Muller, D. E. and Bartky, W. S. A theory of asynchronous circuits, in *Annals of the Computation Lab.*, 29, pp. 204-243, Harvard Univ. Press, 1959; also, Univ. of Illinois Digital Computer Lab. Repts. Nos. 75 and 78, 1956 and 1957. Also, see Nelson, J. C. *Speed Independent Counting Circuits*, Univ. of Illinois Digital Computer Lab. Rept. No. 71, 1956.

used as a completion signal to indicate that all of several parallel operations have occurred, and permits more complex paralleling schemes than possible with synchronous circuits.

Three techniques have been used in the design of speed-independent* networks at the University of Illinois. The first consists of using certain rules for interconnecting previously designed circuits to form more complex ones. As an example, consider the interconnection of a counter and a shift register to form a circuit for generating any given number of shifts. These two units could be interconnected in a serial fashion, but the property of speed independence is best illustrated by having them operate in parallel, as shown in Fig. 4.16. The counter element A changes state i

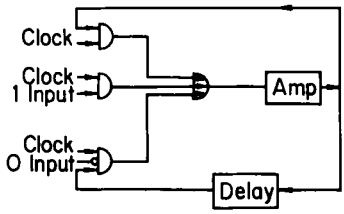


FIG. 4.15. Block diagram of a dynamic flip-flop

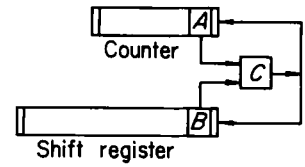


FIG. 4.16. Control of two parallel operations by an operation completion circuit, C

times before the counter stops, where i depends on the initial setting of the counter and is less than 2^n where $(2^n - 1)$ is the capacity of the counter. The shift register element B changes state whenever a shift occurs. The completion circuit C prevents either the counter or register from getting more than one step ahead of the other. If the shift register operates faster, the next shift will be delayed until the signal from A appears at the input to C. If the counter acts faster, initiation of the next count is delayed until the signal from B appears. The time taken by the complete system is the greater of the times taken by the two units plus the time for $2i$ operations of the completion circuit.

The second technique is used for designing the basic logical circuitry. The fundamental logical requirements of the machine may be described by a set of Boolean equations, as in the case of a synchronous computer. Then, the conversion from a synchronous to an asynchronous system may be made as follows. First, each flip-flop in the synchronous system is replaced by two, since a second one is required to store information dur-

* See: *On the Design of a Very High Speed Computer*, Univ. of Illinois Digital Computer Lab. Rept. No. 80, 1957.

ing the process of gating into the first. Secondly, a two-wire system is introduced to connect each flip-flop pair with direct connections from the second to the first. During the transmission of information, one line will always be 0 and the other 1. A 1 or a 0 on one of the lines determines a bit of information. To distinguish individual bits, the lines are cleared after each transmission by applying the same signal to both. The final step in converting to asynchronous operation consists of adding completion circuits where necessary.

The third technique makes use of a change chart which lists the signal changes which take place at the nodes of the network together with an ordering of these changes from which a speed independent circuit can be derived. The end result can be expressed by a set of Boolean functions. In the very high speed computer project at the University of Illinois, programs have been written for its ILLIAC computer for the purpose of simulating the behavior of circuits and testing them for speed independence. Without such programs the design of these circuits would not be practically feasible, since the checking process is usually too tedious to be performed by hand.

The asynchronous dc coupled system of circuit logic was proposed by the Princeton Institute for Advanced Study, and extensive refinements have been developed at the Digital Computer Laboratory of the University of Illinois. It was used with vacuum tube circuits in the I.A.S. MANIAC, University of Illinois ILLIAC and other computers of the I.A.S. family. Fig. 4.17 shows the principal gating circuits in the early ILLIAC computer. The inverter and twin cathode follower have already been described. The vacuum tube diode circuit acts as an OR gate for negative input signals and as an AND gate for positive ones. By placing the load resistor in the cathode circuit rather than in the plate circuit, one would obtain an OR gate for positive signals and an AND gate for negative signals.

4.4. Transistor Systems of Circuit Logic

4.4.1. Point-Contact Transistor Circuits

In designing systems of circuit logic based on the use of transistors, two major problems peculiar to transistor circuits must be taken into account. First, there are problems associated with the low input impedance of a transistor. Second, in high speed circuits, there is a problem due to delays in response, resulting from the storage of minority carriers when a transistor is operated in a region of saturation (see pages 129 and 132).

Systems of circuit logic for point-contact transistors are different from

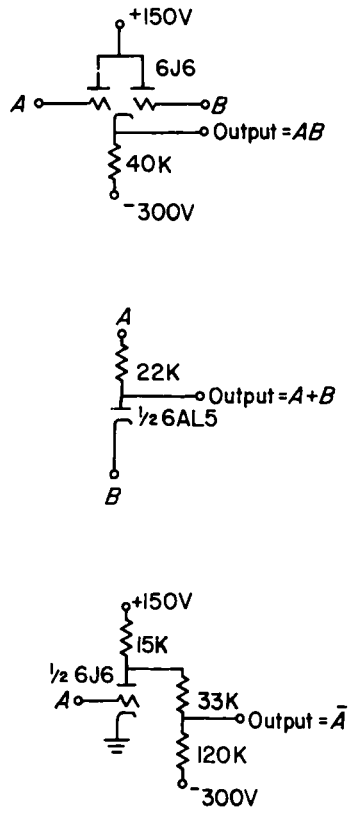


FIG. 4.17. ILLIAC gating circuits

those for junction transistors. For computer systems, point-contact transistors have now been replaced by junction transistors. The presentation of point-contact transistor circuitry is therefore limited, and included principally because of historical importance and inclusion of these circuits in older computers still in operation.

4.4.1.1. Point-Contact Transistor Flip-Flops

This section will be devoted to a brief summary of the characteristics and limitations of the most commonly used types of point-contact transistor flip-flop circuits. Three states of a transistor are of interest in the design of flip-flop circuits. They are: (1) The "active" state, in which the transistor behaves as an active, power amplifying element. (2) The "on" state, in which the current flow is such that the transistor appears as

a low resistance device. (3) The "off" state, in which the transistor appears as a high resistance device. The characteristic curve shows that a point-contact transistor connected as shown in Fig. 4-18 (a) has two stable

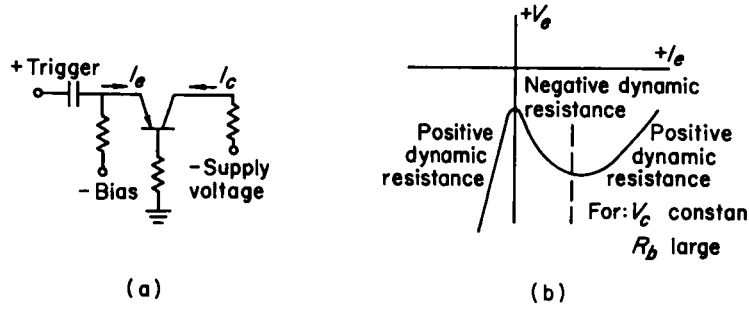


FIG. 4.18. (a) Single point-contact transistor flip-flop, and (b) Voltage-current characteristic of emitter circuit

states, one characterized by small negative values of emitter current I_e, and the other by large positive values of I_e. If the emitter bias voltage and resistance have appropriate values, the circuit can be triggered from one stable state to another. This type of operation is not possible with a single vacuum tube.

An obstacle to high speed operation of point-contact and junction transistors arises from the delays in response due to the phenomena of saturation and hole storage. If the collector voltage for a p-n-p transistor is not sufficiently negative to collect all holes supplied by the emitter, the holes tend to remain in the body of the transistor. Upon reduction of the emitter current to zero, the collector resistance will remain low until the stored holes are removed by the collector field. When the holes are generated by the emitter faster than the collector can remove them, the transistor is said to be saturated.

A capacitor placed between the emitter and collector terminals would increase the high frequency coupling between emitter and collector, thereby decreasing the transition time from one state to the other. However, this circuit has a disadvantage in that a high impedance collector is coupled back to a low impedance emitter (to reduce high frequency gain).

The circuit of Fig. 4.18(a) is shown triggered by a positive pulse which

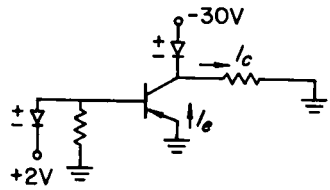


FIG. 4.19. Basic circuit of current type of single point-contact transistor flip-flop

produces a transition from low to high conduction. From Fig. 4.18(b) it is apparent that triggering of the circuit back to a state of low conduction requires application of a negative pulse. The need for pulses of opposite polarity to trigger the flip-flop at successive times can be circumvented by using a rectangular input waveform and differentiating it to yield a positive and negative pulse at the leading and trailing edge, respectively. The switching time of the circuit must be longer than the width of the rectangular input pulse, or else a single rectangular waveform will trigger the circuit through both states. The dependence of this circuit's operation on the shape of the input waveform and narrow triggering pulses make it undesirable from the standpoint of reliability.

The so-called "current" types of single transistor flip-flops are described by Williams, F. C. and Chaplin, G. B. B. [1953]. The basic circuit, shown in Fig. 4.19, depends for its operation on the fact that the transistor in its "active" state simulates a current amplifier and can, therefore, be designed to switch a current between an external diode and itself. This type of circuit is relatively insensitive to transistor parameter variation and can be designed for either a saturating or nonsaturating mode of operation. Its advantages are economy of components and power. However, there are certain disadvantages: (1) Since gates can be connected only at one point, two flip-flops are required if a variable and its complement must be used as signal sources. (2) The circuit is very sensitive to narrow noise pulses when it is in the "off" state, because there is no saturation to overcome. (3) The margins on pulse width and amplitude for complement triggering are not as good as for a two-transistor flip-flop.

The basic form of a two-transistor point-contact saturating flip-flop is shown in Fig. 4.20. A composite voltage–current curve, which can serve as a basis for the design of the dc circuit, may be obtained from the single characteristic curves of the two transistors by adding the emitter currents of each for successive emitter voltages. An operating point may be set anywhere on this characteristic curve by choosing a suitable load to be inserted in the position of R_e.

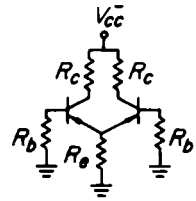


FIG. 4.20. Basic two-transistor (point-contact) saturating flip-flop.

A two-transistor flip-flop eliminates two objectionable characteristics of the one transistor flip-flop described—namely, dependence on triggering by narrow pulses produced by differentiating a rectangular input waveform and availability of only one output point for driving other circuits. The operation of the two-transistor flip-flop can be clarified by referring to the characteristic transistor gain function shown in Fig. 4.21. Assume that one transistor is in high conduction (operating on the right hand section of the gain curve). Therefore, its collector potential will be near ground, thus holding the other transistor in a state of low conduction by making its emitter potential negative with respect to its base. The circuit can be triggered to its other stable state by either positive or negative pulses applied to both emitters since the high gain region of the curve lies to the left of the operating point of one transistor and to the right of the operating point of the other. However, as also apparent from the curve, negative trigger pulses are preferable since the curve decreases much more rapidly in the negative direction.

Various types of coupling circuits may be used to increase the gain of the flip-flop feedback loop during the switching transients. For example, a capacitor may be placed between the base of each transistor and the collector of the other. Important characteristics of the capacitor coupled type of circuit are: (1) Stability with respect to noise triggering is improved by the hole storage in the "on" transistor, since any noise pulse too narrow to last beyond the turn off time of the saturated transistor will fail to trigger the flip-flop. (2) Triggering sensitivity as well as the width of pulses suitable for triggering is proportional to the size of the capacitors. (3) The maximum frequency of operation is inversely proportional to the size of the capacitors. Transformer coupling may be obtained by placing one winding of a transformer between points x and y and the other between the two bases (see Fig. 4.20). The advantage of the transformer coupled circuit is improved complement triggering sensitivity, because: (1) Each transistor acts like a blocking oscillator during

the switching time, thereby improving the switching transient. (2) The transformer couples the current directly from collector to base and the collector simulates a current generator in the switching.

The transistor flip-flops described in the preceding paragraphs are all driven to saturation. Once a point-contact (or junction) transistor has become saturated, it is difficult to turn off because of minority carrier storage effects. When triggering a p–n–p transistor at the emitter, a pulse width greater than the hole storage time is required. When triggering at the base, the pulse width does not have to be as wide as when triggering at the emitter, but considerable power must be applied to clear out stored holes. The time required to clear out stored minority carriers is referred to as minority carrier storage time. It may be computed from $t_s = K'_s \ln (1 + I_{BX}/I_{TO})$ in which I_{BX} is the excess base current due to saturation, I_{TO} is the base current during turn-off, and K'_s is a constant determined by the transistor's characteristics. If neither transistor in a flip-flop is driven to saturation, there will be no delay due to minority carrier storage. Thus by establishing a stable point in the active region the flip-flop's operable repetition rate can be increased. The nonsaturating flip-flop of Fig. 4.22 differs from the circuit of Fig. 4.20 in that two germanium-silicon diode pairs are added. They produce an essentially constant voltage difference, clamping the collector-base voltage and preventing the base from going negative with respect to the collector. The nonsaturating

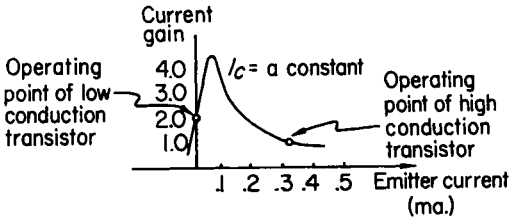


FIG. 4.21. Characteristic transistor gain function

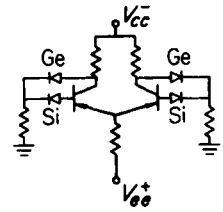


FIG. 4.22. A simple nonsaturating transistor (point-contact) flip-flop

circuit has certain limitations: (1) The output levels are not as consistent unless clamping diodes are added, and the output voltage swing is approximately half that from a saturating circuit. (2) It is more sensitive to narrow noise pulses. (3) Nonsaturating circuits are more sensitive to variability and drift in transistor parameters (a major problem with point-contact transistors) than saturating circuits. This is because the collector voltage varies with collector current instead of stabilizing at the comparatively stable collector voltage which exists at saturation. (However, stabilization schemes limit the speed of a circuit).

4.4.1.2. A Semiconductor Diode, Point-Contact Transistor System of Circuit Logic

In systems of circuit logic utilizing point-contact transistors, the role of the transistor is restricted to storage and amplification, (because of wide variation in the dc characteristics of the transistors) while semiconductor diodes are utilized for the function of gating. Fig. 4.23 shows such an arrangement, suitable for junction transistors also. The input to the

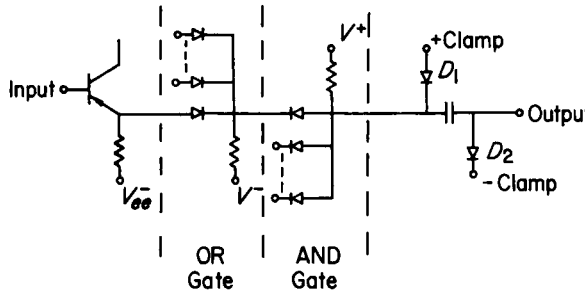


FIG. 4.23. A transistor driven diode gating network

transistor is one of the outputs of a transistor flip-flop. The function of the emitter follower circuit is to provide power gain so that several gates may be driven by the flip-flop. The output of the emitter follower is shown connected to an input of only one of several gates which it may drive. The output of this gate, an OR circuit, is shown driving an input of an AND gate. In systems of this type the gating networks are limited to two levels. If the output of the AND gate were required as an input to several other gates or flip-flops, it would be fed first to the input of a transistor amplifier. Diodes D_1 and D_2 are clamps that limit the pulse amplitude of the AND gate's output. Synchronizing clock signals could be applied as one of the inputs to each AND gate in the system. Then no gate in the system could produce an output except during the occurrence of a clock signal.

4.4.2. JUNCTION TRANSISTOR CIRCUITS

An important difference between point-contact and junction transistors lies in the parameter of current amplification, α, which is the ratio of the increment in collector current caused by an increment in emitter current. For point-contact transistors, α is greater than unity. For junction transistors, it is normally less than unity.

Another important difference is that junction transistors are available in two basic types, a so-called n–p–n as well as a p–n–p type. The point-contact transistors are generally of the p–n–p type. These terms were

chosen to indicate that in one type, the n–p–n, positive charge carriers or holes are in the majority in the base region, whereas in the p–n–p negative charge carriers or electrons are in the majority in the base region. With an n–p–n transistor, current flows in an opposite sense to that in the p–n–p transistor, and the supply voltages are of opposite polarity. With an n–p–n transistor, both emitter and collector resistances are high when these elements are positive with respect to the base. When the emitter is made negative with respect to the base, it emits electrons which are attracted toward the positive collector and constitute the collector current. The availability of junction transistors in both n–p–n and p–n–p types provides the designer with an added degree of freedom in forming systems of circuit logic. This will be brought out in the sections following.

Corresponding to the phenomenon of hole storage in a p–n–p transistor is that of electron storage in an n–p–n transistor. In an n–p–n transistor, if the collector voltage is not sufficiently positive to collect all electrons supplied by the emitter, the electrons become trapped in the base region. Where a distinction is not necessary, the term minority carrier is often used to denote either holes or electrons, whichever is in the minority.

The first high frequency transistor to become available was the grown junction germanium transistor. However, the alloy, mesa, micro-alloy and epitaxial transistors which appeared later are more suitable for switching circuits because of relatively low and consistent values of extrinsic base and collector resistance. The alloy transistor is capable of high peak power, while the mesa and micro-alloy transistors offer high speed operation, the former at high voltage ratings and the latter with good saturation characteristics. The epitaxial transistor offers high speed switching at higher power levels.

4.4.2.1. The Basic Junction Transistor Circuits; the Inverter and Emitter Follower

Corresponding to the vacuum tube inverter and cathode follower circuits are the transistor inverter and emitter follower circuits. They are useful not only for the functions of signal amplification (and inversion in the case of the inverter), but also serve as the basis of a number of systems of switching circuit logic.

A schematic of a basic p–n–p junction transistor inverter circuit is shown in Fig. 4.24. (For an n–p–n transistor, the polarity of the supply voltages would be of opposite sign). When the input is positive with respect to the emitter, the transistor does not conduct, and, therefore, the output voltage will be near that of the collector supply voltage. It is not equal to the supply voltage because of a small leakage current present in transistor circuits. When the input is negative with respect to the emitter, the

transistor will conduct and the output voltage will be near ground. One function of the input voltage divider is to convert the output voltage levels of an inverter to values appropriate for inputs to another circuit. Another important function is to limit the base input current in order to keep the transistor out of saturation. The capacitor reduces the time required for the output waveform of the circuit to follow sudden excursions in the amplitude of the input waveform. It improves the response to positive excursions (usually measured in terms of the waveform's rise time) by supplying a surge of input current when the transistor is put into the conducting state. It improves the fall time by providing a low impedance path for the removal of any stored minority charge carriers in the base-emitter region.

The output voltage waveform of the p–n–p transistor inverter has a fast rise time characteristic, but is not as good with respect to fall time. This is due to two causes. First, time is required to remove the minority carrier charge by collector current. Also, the current for returning the load to its negative potential must flow through R_c, which has a relatively high value, in order to limit collector current. For an n–p–n transistor, the opposite situation, with respect to the rise and fall times of the output voltage waveform, is true. If it is important that both rise and fall times be fast, then the p–n–p and the n–p–n circuit can be combined in a push–pull type of circuit.

Modifications of the basic inverter circuit have been developed aimed at reducing or preventing the buildup of minority charge carriers.

A schematic of a basic p–n–p junction transistor emitter follower circuit is shown in Fig. 4.25. A similar circuit is obtainable with an n–p–n

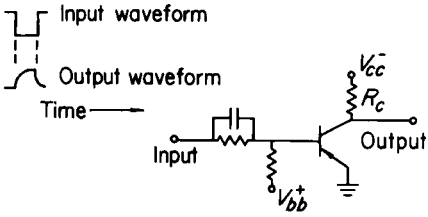


FIG. 4.24. A p–n–p transistor inverter

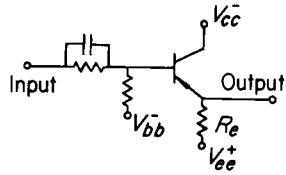


FIG. 4.25. A p–n–p transistor emitter follower

transistor. Both circuits provide a large current gain and a voltage gain slightly less than unity. They also provide a relatively high input impedance and a low output impedance. For the p–n–p circuit, the output signal will be somewhat more positive than the input signal. This small bias can be offset by means of the voltage divider in the input circuit.

If the supply voltages chosen are adequate to maintain a large voltage difference between base and collector, this will tend to alleviate saturation

effects. However, care must be exercised when this is done to prevent excessive dissipation in the transistor.

The rise time of the p–n–p circuit is adversely affected because of the requirement for current flow through R_e. The fall time of the n–p–n circuit suffers because of the same reason. As in the case of the inverter circuits, p–n–p and n–p–n transistors can be combined to provide a circuit with a good rise and fall time.

4.4.2.2. Junction Transistor Gating Circuits

A number of logical switching functions may be synthesized by combinations of the inverter and emitter follower circuits already described. For example, if two inverter circuits (see Fig. 4.24) are connected in parallel with a common collector resistor, there results a AND gate, i.e., given inputs A, B on the bases, the output at the common collector point is $\overline{AB}$. Also, if two emitter follower circuits (see Fig. 4.25) are connected with a common emitter resistor, the output at the common emitter point is AB. In these two cases, the use of n–p–n in place of p–n–p transistors would produce the OR, $(A + B)$, and NOR, $(\bar{A}\bar{B})$, functions, respectively. Figures 4.26(a) and (b) show how both p–n–p and n–p–n transistors may

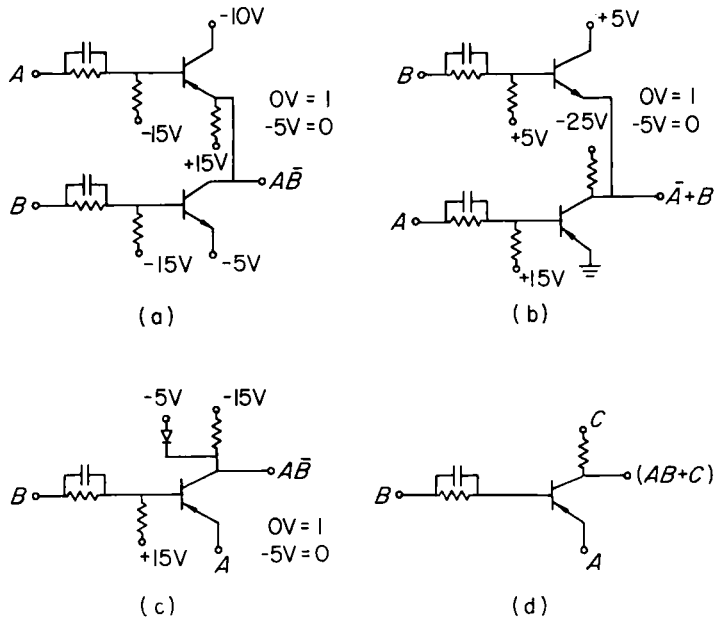


FIG. 4.26. Junction transistor gating circuits: (a) $A\bar{B}$, (b) $(\bar{A} + B)$, (c) $A\bar{B}$, (d) $(AB + C)$.

be combined to yield other logical functions of two variables. Actually, a single inverter circuit may be used to generate a function of two variables or even of three, as shown in Figs. 4.26(c) and (d). In these cases, signal sources are used to supply current that otherwise would be obtained from a power supply. Such circuits are critical in operation and considerable care must be exercised in their application.

4.4.2.3. A System of Circuit Logic Based on Transistor NOR *Circuits*

The transistor NOR circuit is a realization of the NOR switching function described in Chapter 3. It allows the synthesis of switching networks from various arrangements of a single logic building block. It reduces the problem of matching inputs and outputs which is present in systems composed of a number of different logic circuits, and also alleviates problems associated with the loading of logic circuits.

A representative transistor NOR circuit is shown in Fig. 4.27(a). The

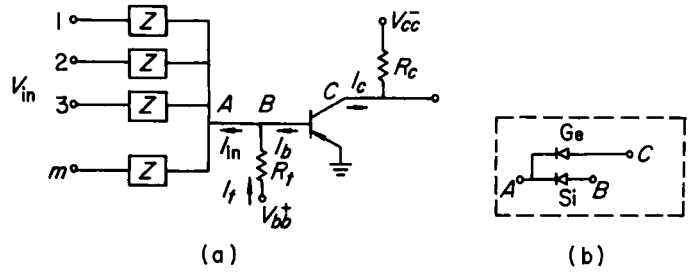


FIG. 4.27. (a) Basic transistor NOR circuit (p–n–p type), and (b) Modification for higher speed operation

boxes labelled Z may be resistors or diodes. Diodes are preferable for limiting input current and for preventing feedback of signals between inputs. The resistor R_t in conjunction with the supply voltage V_{bb} provides a bias to reduce the transistor leakage current I_c to a minimum when the transistor is cut off. When conducting, the transistor is saturated and offers a low impedance. The number of allowable inputs, m, is limited chiefly by the input loading, although a crosstalk factor would also have to be considered if the Z's were realized by relatively low resistances. The number of circuits, n, that the output is capable of driving is limited by the loading of the output. The response time of the circuit can be improved by placing a small capacitor, 30-100 $\mu\mu$f, in parallel with each input resistor, or, for an even higher frequency of operation, by incorporating a germanium and a silicon diode into the circuit, at the points shown

in Fig. 4.27(b). The subtraction of voltage drops between the diodes keeps the collector out of heavy saturation and permits operation in the area of 50 to 100 nanosec.

The circuit shown in Fig. 4.27(a) utilizes p–n–p transistors. By using supply voltages of opposite polarities, n–p–n transistors could be used instead. In that case the input and output signals would be positive instead of negative voltages.

When resistors are used for the impedances shown in Fig. 4.27(a) the circuit is often referred to as a TRL (for transistor-resistor logic) circuit. The basic switching and storage building blocks of the TRL system using n–p–n transistors are shown in Fig. 4.28. The flip-flop can be

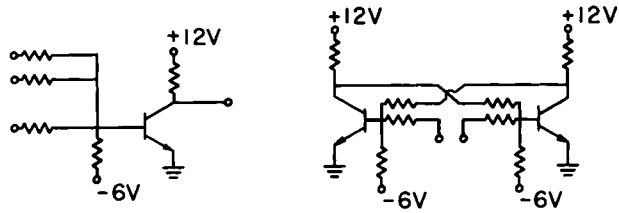


FIG. 4.28. **Basic transistor-resistor logic circuits (n–p–n type)**

considered as being formed either from two resistor coupled inverters or from two two-input TRL circuits. The TRL system of circuit logic (sometimes referred to as NOR logic) offers the advantage of reliability with simplicity and economy, e.g., it is less dependent on collector saturation and base-emitter voltage than DCTL circuits (described in Section 4.4.2.4).

The response time of the TRL circuit can be reduced by placing a capacitor in parallel with each gating resistor in order to produce a current spike at the leading and trailing edges of the signals. However, the use of the circuit is complicated by the fact that if there is a simultaneous transition of more than one input signal from a lower to an upper level, fictitious spikes appear in the output, even though one or more other inputs are at the lower level). These spikes can propagate through several stages, amplified at each, because the speed-up capacitors in each stage present a low impedance to these spikes. This difficulty may be overcome by restricting the logical design to prevent movement of more than one input at a time from a lower to an upper level, or by restricting the number of inputs to two. The former process reduces the flexibility of the system and the latter effectively cancels the economy of components of the TRL circuit. A more direct approach is to use higher frequency transistors rather than speed-up capacitors and extra transistors.

4.4.2.4. Systems Using Direct Coupled Transistor Circuits

Surface barrier and alloy junction transistors are characterized by low voltage drops, low power consumption, and rapid recovery time. Special gating and storage circuits have been devised to exploit these properties. The basic gating circuits developed by the Philco Corporation are simple, the transistors being used in a way similar to the way relays are used in switching circuits. Examples of these circuits are shown in Figs. 4.29(a) and (b). These circuits, characterized by a small number of passive components as well as direct coupling, are referred to as DCTL (for direct coupled transistor logic) circuits. The transistors must have a high ratio of base–emitter voltage to collector–emitter voltage at saturation (so that a saturated transistor can keep off a gate it is driving).

In Fig. 4.29(a) the transistors are connected in series, which is possible

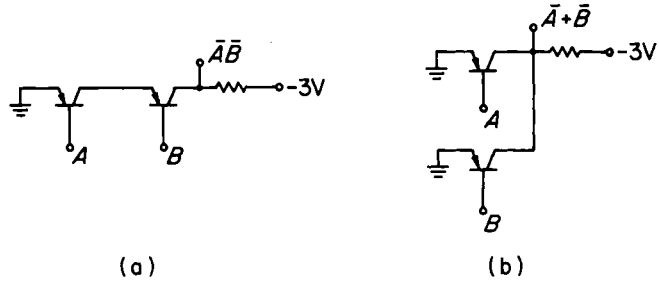


FIG. 4.29. Surface barrier transistor logic circuits: (a) NOR, (b) Sheffer stroke

because of the small emitter–collector voltage drop. The bases of the transistors are connected to control voltages originating from the output of other similar circuits or directly from flip-flop collectors. If any base is at ground potential, that particular transistor will be nonconducting. Therefore, the output will be at -3 volts. If all base voltages are sufficiently negative, the output will be close to ground potential. Therefore, this circuit produces the NOR switching function of the input variables. The circuit in Fig. 4.29(b) essentially represents a set of inverters in parallel. If any base voltage is sufficiently negative, the transistor will conduct, causing the output voltage to be near ground potential. Therefore, this circuit produces the Sheffer stroke switching function (F_8 in Table 3.7). These switching circuits have rise and fall times less than 0.1 μsec. One of their disadvantages is that the transfer characteristics are such that a noise pulse in excess of about 0.1 volts on the base may be amplified and appear in the output.

The basic DCTL flip-flop circuit uses only two transistors and two resistors. Figure 4.30(a) shows this basic flip-flop with associated input

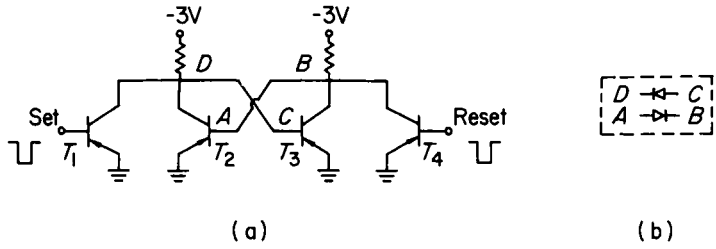


FIG. 4.30. Basic DCTL flip-flop with input circuits

circuits. The input circuits, consisting of transistors T_1 and T_4 serve to set or reset the flip-flop, and are cut off except during an input pulse. Depending on the state of the flip-flop, either transistor T_2 or T_3 may be conducting. The collector voltage of the non-conducting transistor is determined by the current drawn from the base of the conducting transistor (which saturates heavily). In the circuit shown, the two collector voltages are approximately -0.04 and -0.6 volts. The rise and fall times are both less than 0.1 μsec.

One limitation of the basic DCTL flip-flop circuit is that it contains no well determined transient memory or delay (although hole storage produces an uncontrolled delay) and so has an uncertain response to a pulse occurring at the same time as its set or reset input. Also, the low collector-voltage swing is insufficient to drive circuits requiring larger signals. A variation of the basic DCTL flip-flop circuit that alleviates some of its objectionable features is obtained by placing a silicon junction diode in the feedback path from the base of each transistor to the collector of the other with the orientation shown in Fig. 4.30(b). Each diode simulates a constant voltage source during forward conduction and, while recovering, in reverse conduction. The use of the diode results in a somewhat larger collector voltage swing, in less saturation, and in shorter resolution and fall times. The time constant improvements are due to the hole storage effects of the diodes which act to draw out the holes stored in the base of the conducting transistor. The diode also improves the inverter transfer characteristics.

A relatively simple way of preventing saturation in a circuit is to include breakdown diodes at appropriate points. The volt-ampere characteristic of an idealized breakdown diode is shown in Fig. 4.31(a). This characteristic is closely approximated by silicon junction diodes which are available with breakdown voltages from 4 volts and up. In Fig. 4.32 a modification of the basic direct coupled flip-flop is shown. It is kept out of saturation by the silicon junction diodes D_1 and D_2 which have a combined volt–ampere characteristic as shown in Fig. 4.31(b). Diodes

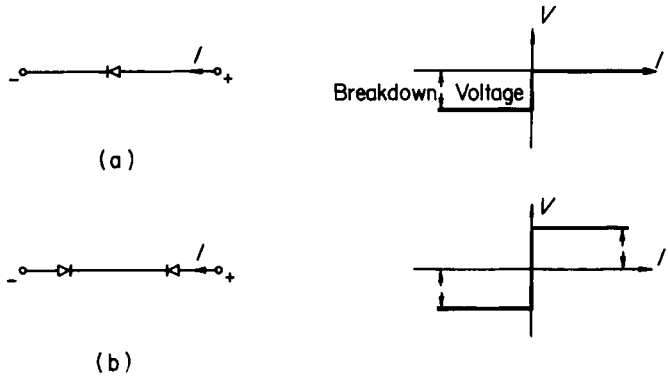


FIG. 4.31. Volt–ampere characteristics of idealized breakdown diodes

D_3 and D_4 are also breakdown diodes, but with a breakdown voltage greater than that of D_1 and D_2. D_3 and D_4 are always kept in the broken down state to maintain a constant voltage drop between the base of one transistor and the collector of the other. Under stable conditions one transistor conducts heavily and the other lightly. If transistor T_1 is conducting heavily, then D_1 is broken down and D_2 is conducting in the forward direction. The circuit of Fig. 4.32 using 2N711 transistors is operable up to about 5 Mc.

As already pointed out, a major factor limiting the switching speed of saturating transistor circuits is the delay caused by minority carrier storage. These stored carriers are most quickly removed by applying a reverse-bias voltage to the base–emitter diode. Therefore, a transistor may be switched off more rapidly by bringing its base to an off-bias voltage instead of to ground. The flip-flop circuit shown in Fig. 4.33 achieves a

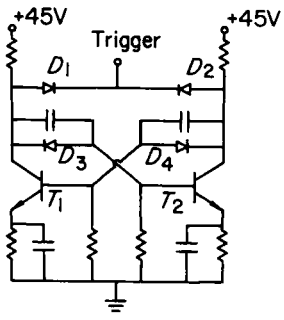


FIG. 4.32. A nonsaturating binary counter

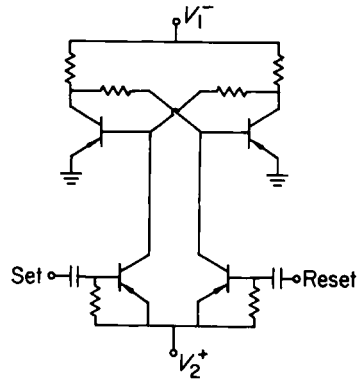


FIG. 4.33. Base gated flip-flop

higher switching rate than the basic DCTL flip-flop by the use of off-bias gating. When V_2 is properly chosen, not only does the gate switch off the conducting transistor, but also it supplies an amount of current to the load resistor of the nonconducting transistor equal to the amount the transistor will conduct after the switching action is completed. This pseudocollector current reduces the delay normally preceding the switch-on transient. As a result, the delay from the time of application of the trigger pulse until the end of the switching transient is only about 20 nanosec using SB-100 surface barrier transistors.

The single input flip-flop circuit shown in Fig. 4.34 is kept out of saturation by the use of diodes and dividing resistors which prevent the conducting transistor from saturating. When the collector voltage of the conducting transistor drops to about 0.5 volts, the diode conducts, preventing a further drop in collector–base voltage. This reduces the adverse effects of minority carrier storage, since there are fewer carriers to be removed from the base when the conducting transistor is triggered off. The steering diodes, D_1 and D_2, allow the trigger to be either a pulse or a square wave. Also, they provide isolation between the two sections of the flip-flop.

If in the DCTL flip-flop of Fig. 4.30, a parallel resistor, capacitor combination is placed in each base lead, there results a flip-flop with higher switching speed. This *RC* coupled flip-flop has a transition time about 20% less than the direct coupled circuit. The function of the resistor is to limit the base current in order to reduce hole storage delay time. The capacitor aids in switching off a transistor by driving its base to a positive voltage. Unfortunately, another effect of the resistance in the base circuit is to reduce the stability of the flip-flop, because it reduces the base current into the conducting transistor. There are a number of devices for improving the switching speed without sacrificing stability. One way of improving both speed and stability is to add an emitter follower in each of the cross-coupling arms. Also, the delay time due to hole storage can generally be reduced to one-half by the use of nonsaturating circuitry.

Figure 4.35 shows a modification of the basic *RC* coupled flip-flop, designed to provide nonsaturating operation. A resistor inserted from the base of each transistor to ground forms a voltage divider which limits the voltage swing of the base. A parallel *RC* network inserted between the common emitter point and ground provides dc feedback which causes the emitter to stay at a level about 0.3 volts positive with respect to the base of the conducting transistor. Since the base to emitter voltage is independent of the emitter resistance, the choice of resistance controls the emitter current. Fixing of the base voltage and emitter current determines the collector current. The transistor is kept out of saturation by

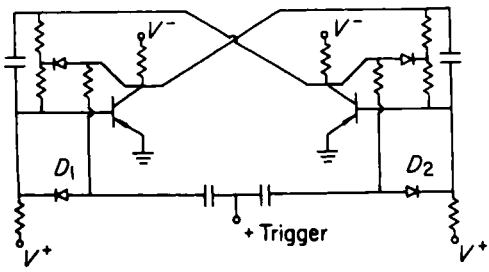


FIG. 4.34. A nonsaturating flip-flop

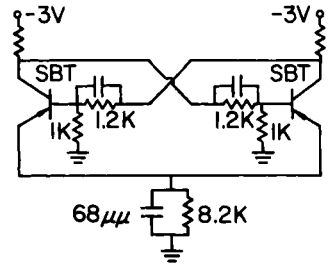


FIG. 4.35. A nonsaturating *RC* coupled flip-flop

choosing a collector resistance sufficiently small to hold the collector voltage of the conducting transistor sufficiently negative with respect to the base.

Figure 4.36 illustrates a flip-flop circuit in which an emitter follower is

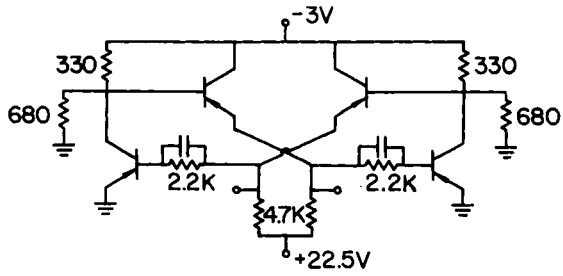


FIG. 4.36. A saturating emitter follower coupled flip-flop

used to provide an active coupling network. It allows fast switching action to be obtained because of two principal effects. First, it is not affected by hole storage. Second, its low source impedance allows high charging currents to be supplied to the stray and internal capacitances. The outputs of the emitter followers also provide convenient output terminals because of their buffering action. The transition time for the emitter coupled flip-flop is about 70% less than that of the direct coupled flip-flop. The emitter follower flip-flop circuit can be modified to yield non-saturating operation, by using the emitter biasing method employed to provide non-saturating operation in the *RC* coupled flip-flop (see Fig. 4.35).

4.4.2.5. A System of Dynamic Pulse Circuitry

The circuits shown in Fig. 4.37 have been used at IBM as the basis of a system of logic for a high speed parallel computer application. These

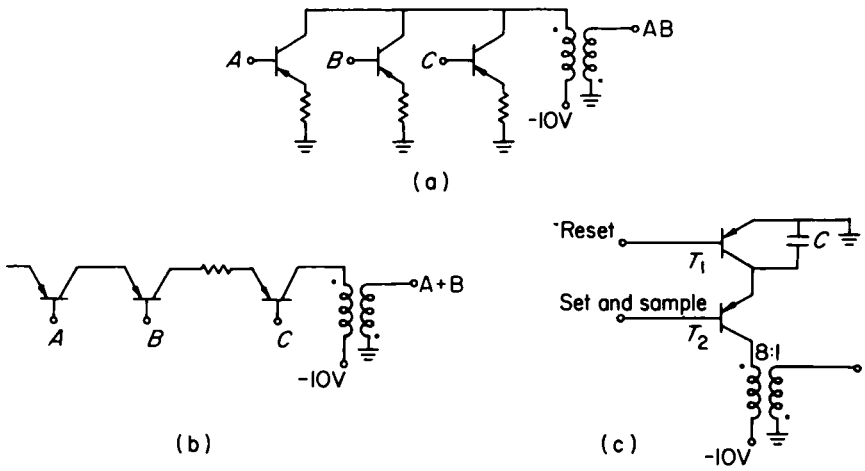


FIG. 4.37. A dynamic pulse system of circuit logic

circuits were designed to use high speed p–n–p drift transistors and to operate at a pulse repetition rate of 10 Mc. The input pulses are specified to be in the form of a one-half cycle sine wave with an amplitude of -1 volt and a maximum width of 40 nanosec. A noteworthy characteristic of the system is that only one power supply is required.

The AND and OR gates are similar in appearance to the direct coupled transistor gating circuits with dc input signals described in Section 4.4.2.4. These circuits differ principally in that the base input signals are pulses and the output of each circuit is fed to a pulse transformer which can drive several loads.

The pulse storage circuit was designed to operate with asynchronous inputs and to produce synchronous outputs. The storage function is achieved essentially by the storage of charge on the capacitor C. Pulse inputs to transistor T_2 charge the capacitor C, and the charging current produces an output pulse. The charge on C will gradually decrease, but for a period of 10 μsec will be large enough to inhibit further inputs to T_2. During this period, inputs to T_2 will not produce an output, but will reestablish the charge on C, allowing an additional 10 μsec of storage. An input to T_1 will remove the charge on C and allow the next input to T_2 to produce an output. The readout process destroys the stored information, and it must be rewritten if it is to be retained.

4.4.2.6. A Gated Pulse Amplifier System

The circuits shown in Fig. 4.38 have been used at Sylvania Electric Products, Inc. in a system designed to operate on both dc and pulse

type signals. One of the basic circuits of this system is the pulse pedestal gate, shown in Fig. 4.38 (a). It is used for the detection of coincidence

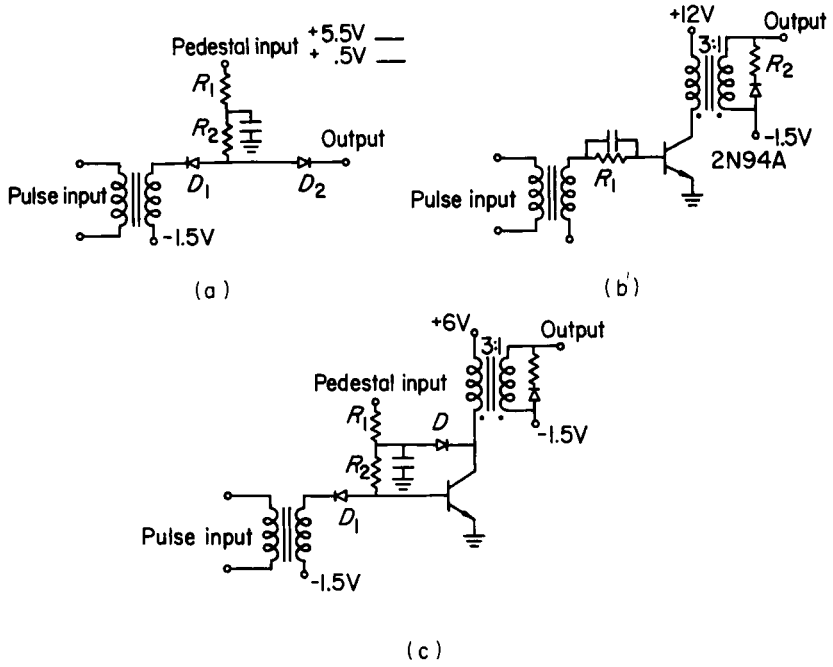


FIG. 4.38. A gated pulse amplifier system of circuit logic, (a) pulse-pedestal gate, (b) pulse amplifier, (c) combined pulse-pedestal gate and pulse amplifier.

between the dc output of a static flip-flop and a clock pulse. When the input voltage from the flip-flop is relatively high, current flows through R_1, R_2, D_1, and the transformer secondary winding, while D_2 is cut off. When a positive pulse is applied across the transformer secondary winding, D_1 is cut off and D_2 conducts, allowing the gated current to flow in the load circuit. When the input voltage from the flip-flop is relatively low, the current supply is effectively removed, thereby disabling the gate. The capacitor is provided to insure that a fast change in the pedestal level does not of itself produce an output pulse. The circuit is relatively insensitive to input pulse level variations.

Another major circuit of this system is the pulse amplifier shown in Fig. 4.38 (b). It is used wherever a large output current is required, as in the case where many flip-flops and gates must be driven by a single source. The function of the capacitor is to improve the collector current rise time by providing a large initial surge of base current. After the

capacitor is charged, resistor R_1 limits the drive current to minimize minority carrier storage effects. The discharge of the capacitor at the end of a pulse period facilitates rapid recovery of the transistor. In Fig. 4.38(c) the pulse amplifier is combined with the pulse pedestal gate to provide current gain for the gate output. This combination is used whenever the pulse output of a gate must pass through two or more other gates. The function of the diode D is to prevent saturation of the transistor.

For AND and OR operations in addition to those provided by the pulse-pedestal gates, conventional diode gates are used. However, as shown in Fig. 4.38(d), the output of each dc level gate is fed to an emitter follower output circuit. This effectively isolates the diode gates from their load. The inputs of these gates are obtained from the outputs of static flip-flops, and the dc outputs of the emitter followers may be used for the dc level inputs to the pulse-pedestal gates. The output circuit load itself serves as the load resistor of the emitter follower circuit.

The static flip-flop circuit used in this system is shown in Fig. 4.38(e). It was designed to operate at a pulse repetition frequency of

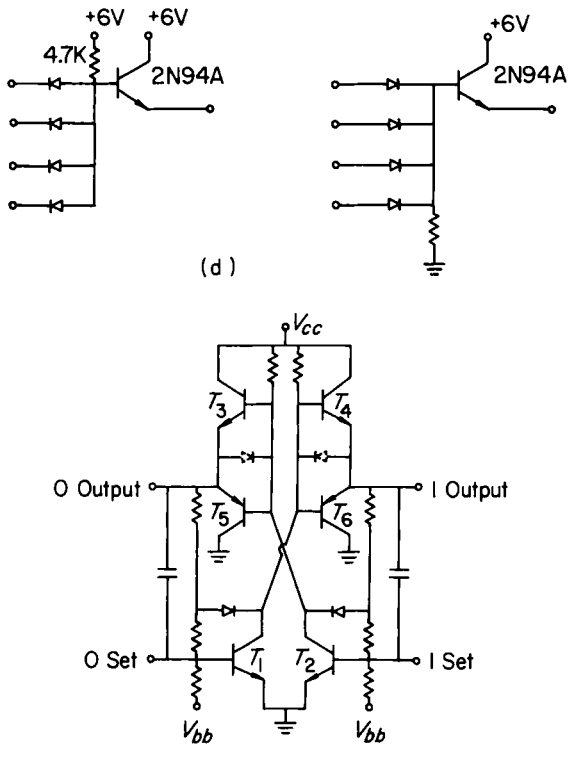


FIG. 4.38. (Continued from page 145): (d) dc gates, (e) flip-flop.

1 Mc. The circuit is basically the Eccles–Jordan configuration with push-pull emitter followers incorporated in the feedback paths from the collector of each transistor to the base of the other. The purpose of the emitter followers is to isolate the flip-flop transistors from loading effects of the internal regenerative coupling circuits and the external load. They permit faster switching action, and are capable of driving several gates. The emitter followers, T_5 and T_6, are required only when a dc AND gate is to be driven. They then supply a negative drive current and T_3 and T_4 discharge the load capacitance. For all other loads T_5 and T_6 can be replaced by diodes, as indicated in Fig. 4.38(e). T_3 and T_4 then supply the drive currents and the diodes discharge the load capacitance. The function of the other diodes is to hold the collector of the conducting transistor at a voltage ample to keep it out of saturation.

4.4.2.7. Systems Based on Current Switching Circuits

All the transistor switching circuits described thus far operate in what may be termed a voltage mode. In this section, systems of logic are described which are based on the use of circuits operating in a so-called current mode, wherein the current from an essentially constant current source is switched. Circuits of this type may be designed for either saturating or nonsaturating operation. The discussion following will be confined to nonsaturating current switching systems, these being capable of higher speed operations.

4.4.2.7.1. Nonsaturating Complementary Current Switching Systems

In the design of high speed circuits, consideration must be given to the delays that may be introduced by minority carrier storage as well as those due to the usual circuit parameters that limit frequency response. The greater the saturation delay due to minority carrier storage, the less time will there be available for transition to the switching threshold of the stages being driven. Therefore, by operating the transistor in a region out of saturation, the requirements on the rise and fall times of a circuit for a given over-all delay may be reduced.

While the circuits to be described in this section could use other transistors, they were designed for use with drift type transistors, and to introduce delays of only 20 nanosec per circuit. Circuits using these transistors are not only simple and relatively insensitive to noise, but also capable of a high frequency of operation. Also, the characteristics of drift transistors are such that their most favorable operation is found in a higher voltage, higher current region where the circuits are nonsaturating. To allow operation of the transistor within this optimal region

with low signal voltage swings, it is necessary that the circuits used be capable of controlling the operating region of the transistor so that it is kept within specified voltage and current ratings. These requirements are satisfied by using a transistor as a current generator driving other transistors. These circuits may be either ac or dc coupled.

Representative p–n–p and n–p–n current switches that can serve as the basis of a current switching system of logic are shown in Fig. 4.39.

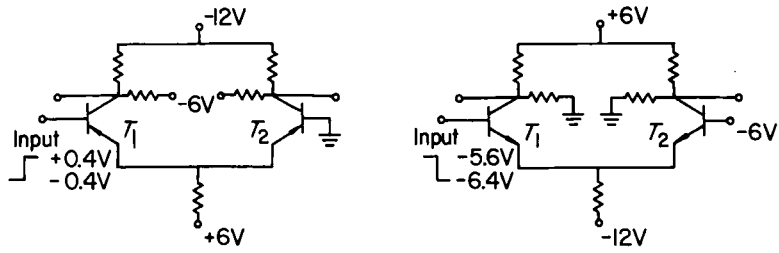


FIG. 4.39. Complementary current switches

These circuits are simply differential amplifiers in which the circuit parameters are chosen to allow the transistors to operate in a region of good frequency response and low collector capacitance. In the p–n–p circuit, one input is referenced to ground, and in the n–p–n circuit one input is referenced to −6 volts. The swing of the input signals in either circuit is just enough to switch current completely into either transistor. In the p–n–p circuit, when the input is at +0.4 volts, T_1 is biased off and T_2 is conducting, and when the input is at −0.4 volts, the reverse situation occurs. To make the output swing about −6 volts, a small current bias is added through the resistor returned to the −12 volt supply. For the p–n–p circuit, the output voltage swing is from −5.6 to −6.4 volts. Because of the voltage shift within each switch, the output of one switch cannot be coupled directly to the input of another of the same type. However, switches of opposite types can be directly coupled as observation of the input and output signals of each indicates.

Gating circuits may be formed from the basic current mode switch by connecting transistors in parallel with T_1 in either the p–n–p or n–p–n circuit. Figure 4.40 shows a three input gate formed by adding two transistors in parallel with T_1 of the p–n–p current mode switch. A similar gate constructed of n–p–n transistors would have outputs $(\bar{A}\bar{B}\bar{C})$ and $(A + B + C)$ corresponding to the outputs $(\bar{A} + \bar{B} + \bar{C})$ and ABC of the p–n–p gate shown in Fig. 4.40. Because these gates have two useable outputs, each of which is the complement of the other, there is no need

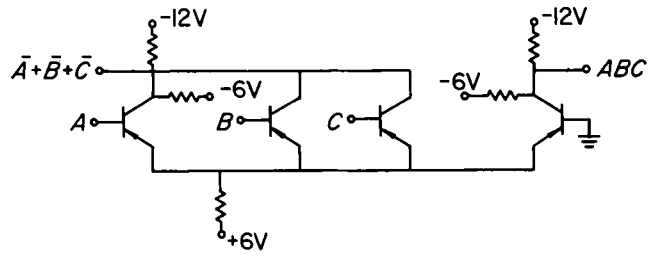


FIG. 4.40. A p–n–p complementary current switch gate

for a separate inverter building block in the system. As a result, there is also a reduced over-all delay in long logic chains frequently including inverters. Complex switching functions can be constructed by combining circuits similar to the one shown in Fig. 4.40. For example, assume a circuit similar to the one in Fig. 4.40 is used to generate $(\bar{D} + \bar{E})$ and DE. If these two output points are connected to the output points of $(\bar{A} + \bar{B} + \bar{C})$ and ABC, respectively, in Fig. 4.40, the complementary functions produced at those points by the resulting circuit would be $(\bar{A} + \bar{B} + \bar{C})(\bar{D} + \bar{E}) = \overline{(ABC + DE)}$ and $(ABC + DE)$. It is possible to generate any function using either p–n–p or n–p–n gates exclusively because each contains inversion in addition to its other logical properties. However, in general, use of both types of circuits will enable switching functions to be generated with fewer transistors than if only one type is used.

A flip-flop circuit can be formed by intercoupling a p–n–p and an n–p–n current mode switch. A flip-flop formed in this manner is shown in Fig. 4.41. Note that the direct rather than the inverted output of each

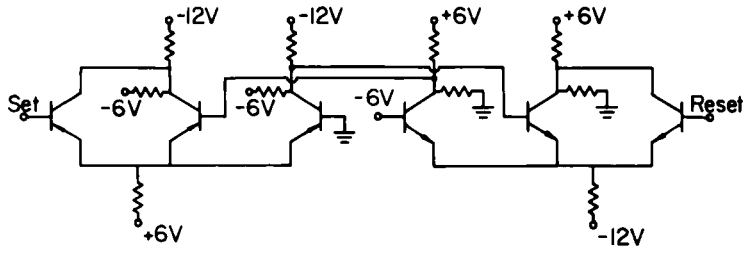


FIG. 4.41. A complementary current switch flip-flop

switch is coupled to the base of the other. The circuit is set to one state or the other by means of a pull over transistor added in parallel with each switch. An OR function can be incorporated into each of the flip-

flop's input circuits by adding transistors in parallel with the pull over transistors.

4.4.2.7.2. A SYSTEM BASED ON NONSATURATING COMPLEMENTARY CURRENT SWITCHING AND INHIBITING CIRCUITS. The basic nonsaturating complementary current switches used in this system are shown in Fig. 4.42. They are essentially the same as those described in the pre-

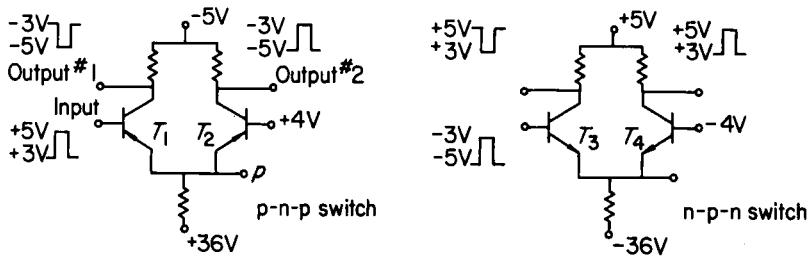


FIG. 4.42. Complementary current switches

ceding section. In the p–n–p switch, the constant current at point p will flow through whichever transistor has the more negative base voltage. Therefore, either T_1 or T_2 will conduct a constant current, according to whether the input to T_1 is greater or less than the bias voltage on T_2. Operation of the n–p–n switch is similar. Both circuits can be made nonsaturating because collector current is controlled. These circuits are not only suitable for high speed switching, but, because of the large voltage swings employed, relatively insensitive to noise.

If output number 2 is not required, a simplified form of the current switches may be formed. Figure 4.43 shows the simplified complementary

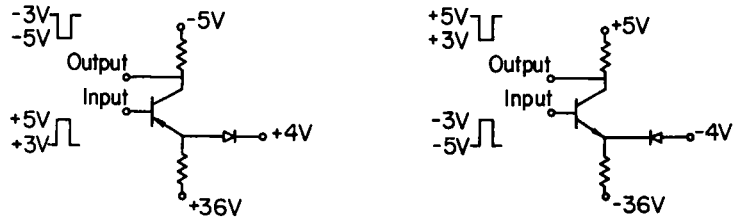


FIG. 4.43. Simplified complementary current switches

current switches in which transistors T_2 and T_4 of Fig. 4.42 have each been replaced by a semiconductor diode.

A modification of the basic p–n–p complementary current switch, referred to as an inhibitor, is shown in Fig. 4.44. When T_2 conducts,

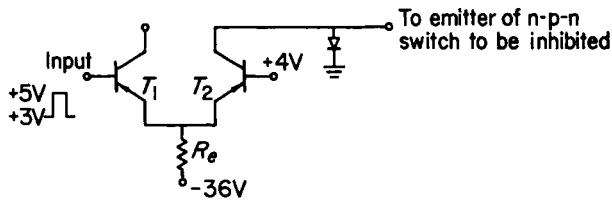


FIG. 4.44. p–n–p complementary current inhibitor

its entire collector current flows into the common emitter resistor of the n–p–n current switch to which the collector of T_2 is connected. This brings the emitters of the n–p–n switch to ground potential (provided that R_e is sufficiently less than the common emitter resistor of the n–p–n switch). Because the signal input to the n–p–n switch can never be more positive than ground, both transistors of the n–p–n switch are back biased, and both of its outputs will be +5 volts regardless of the input. Thus the n–p–n switch is inhibited. The diode to ground prevents saturation of T_2 under dc worst case conditions. The other collector of the inhibitor can either be connected to a −5 volt supply through a resistor, to provide a regular p–n–p switch output, or used to inhibit another n–p–n switch.

Figure 4.45 is a schematic of a simplified flip-flop circuit formed from

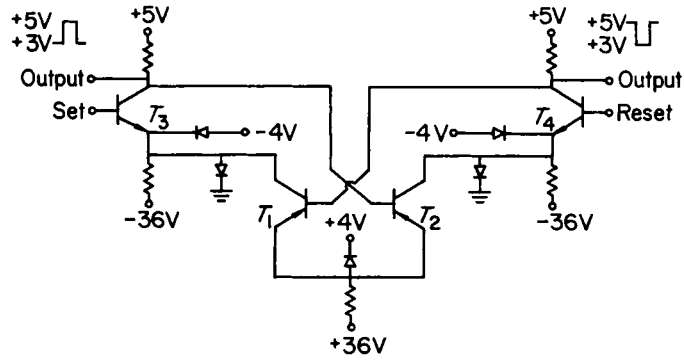


FIG. 4.45. Simplified complementary current flip-flop

an inhibitor and two simplified n–p–n current switches. The flip-flop is triggered by momentarily switching off the "on" transistor. Since it has outputs only of positive sign, it can drive only p–n–p gates and switches.

An important feature of this flip-flop is that both AND and OR gates can be incorporated within it, thus allowing appreciable savings in components in a large switching system. AND gates may be incorporated by adding transistors in parallel with T_3 and T_4, and OR gates incorporated by paralleling transistors with T_1 and T_2.

Figure 4.46 is a schematic of a flip-flop with both positive and nega-

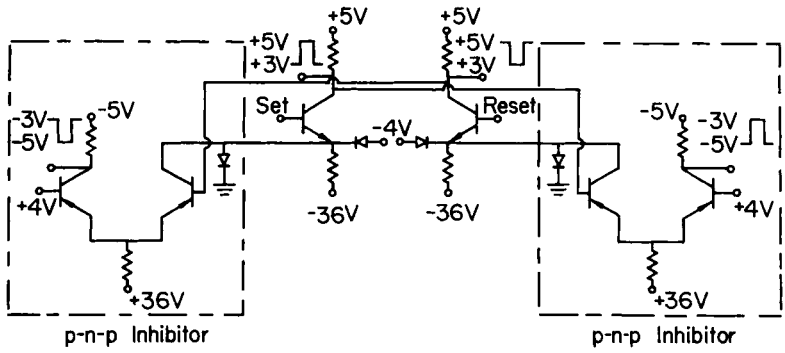


FIG. 4.46. A complementary current flip-flop with four outputs

tive outputs and capable of driving n–p–n as well as p–n–p circuits.

4.5. Magnetic Core Systems of Circuit Logic

4.5.1. INTRODUCTION

A magnetic core is a small toroid formed of a material exhibiting ferromagnetic properties. Representative dimensions for a core used in logic circuits are inner and outer diameters of 0.08 in. and 0.12 in. respectively, and a thickness of 0.03 in. The magnetic core is a versatile computer element capable not only of being switched from one stable state to the other within microseconds but also of remaining in a specified state without a continuous dissipation of energy. The use of the magnetic core as a gating element will be described first. In the succeeding section its use in information storage applications will be described.

Let us consider briefly the switching properties of a magnetic core. Figure 4.47 shows the idealized hysteresis loop of a ferromagnetic material. If a core is magnetized at the point of the loop designated as B_r and is subjected to a negative magnetic force, $-H_m$, it will traverse the downward path indicated by the arrow, and arrive at the point on the loop, 0_e. When the magnetic force is withdrawn, the core will return to

the point $-B_r$. While the application of $-H_m$ to a core in the B_r state produces a large change of flux $\Delta\Phi_1$, the application of $-H_m$ to a core at $-B_r$ produces a small change of flux $\Delta\Phi_2$. Similar remarks apply when $+H_m$ is applied to a core either at state $-B_r$ or $+B_r$, the point being that once a core has been switched to a stable state by a magnetic force, it will remain in that state until a comparable magnetic force of opposite sign is applied. (Successive application of forces of like sign essentially leave the core unaffected). This behavior is analagous to that of a set-reset type of flip-flop wherein a signal on an input line will have an effect only if the preceding input signal was on the other input line. When an applied magnetic force adequate to switch the core is withdrawn, the core will return to a state of positive or negative magnetic remanence ($+B_r$ or $-B_r$) according to whether the applied force was positive or negative, respectively. The two stable states $+B_r$ and $-B_r$ define the 1 and 0 state, respectively.

The hysteresis loop shown in Fig. 4.47 is, as stated, idealized since discontinuous breaks are not obtainable with existing ferromagnetic materials. The hysteresis loop of a representative ferromagnetic material which approaches the ideal is shown in Fig. 4.48. The more rectangular

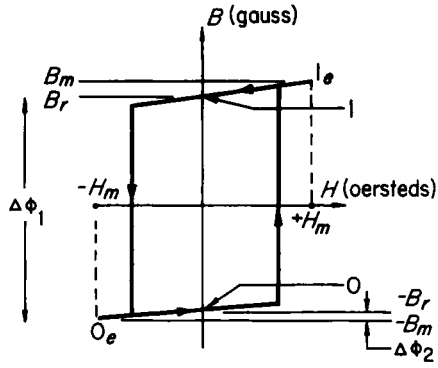


FIG. 4.47. Idealized hysteresis loop of a ferromagnetic material

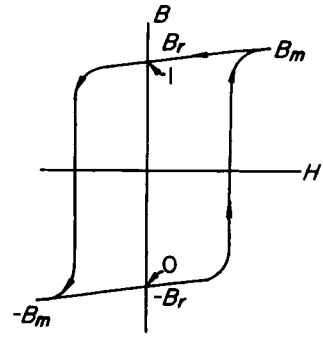


FIG. 4.48. Hysteresis loop of a representative ferromagnetic material

in shape the hysteresis loop, the more suitable is the material as a switching element.

The operation of a simple magnetic gate will now be described. Consider the circuit shown in Fig. 4.49. Three windings (two input windings, A, B, and one output winding C) are formed on a toroid of suitable magnetic properties. Assume that the core is initially in magnetic state $-B_m$, (see Fig. 4.48). If a positive current pulse of sufficient magnitude is

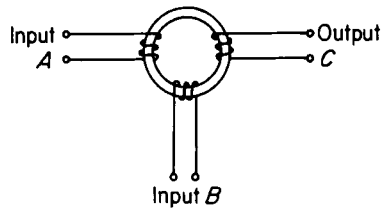


FIG. 4.49. A simple magnetic core gate

applied to winding A, the magnetic state of the core moves to $+B_m$. On removal of this pulse, the magnetic state recedes slightly to $+B_r$. If, subsequently, a negative current pulse is applied to B (i.e., one producing the opposite magnetizing effect as that produced by the pulse at A), the magnetic state of the core is switched to $-B_m$, (receding subsequently to $-B_r$).

It is clear that signals are required on both input windings alternately in order to obtain an appreciable signal on the output winding. Consecutive inputs on only one of the input windings (without an intervening input signal on the other input winding) produces only negligible output signals, since successive application of forces of like sign essentially leaves the core undisturbed. Specifically, two consecutive signals on winding A produce a change of flux $(B_m - B_r)$, while two consecutive signals on winding B produce a change of flux $-(B_m - B_r)$.

In the course of forming the desired output signal, certain other undesirable signals may be formed: A signal on an input winding may also induce a signal (of opposite sign to the desired output signal) in the output winding C. Also, a signal on one input winding may also cause a feedback signal into another input winding as well as produce an output signal on winding C. These unwanted signals may be effectively removed, if necessary, by means of appropriate circuitry.

A symbolic representation useful for describing magnetic core logic circuitry is shown in Figure 4.50. The core is represented by a circle. A

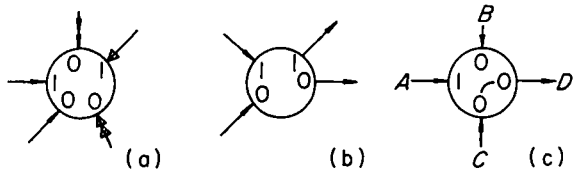


FIG. 4.50. A symbolic representation for magnetic core logical elements

line with an arrow pointing to the circle represents an input that sets the core to the binary state indicated just inside the circle. Open arrows imply pulses, and closed arrows dc signals. Double arrowheads of either type serve to indicate that the existence of this input will hold the core to that state despite the presence of other input signals. The symbol on the input line may indicate either a timing input, or designate the source of the input signal (and the time when it appears). Lines originating at the circle represent output circuits. A signal is present on the output line when the core is switched to the state shown at the origin of the line. In Figure 4.50(c), there is a significant output only when the core is switched from a 1 to a 0 state by a signal on line *C*. This is a conditional transfer circuit since an input on *B*, which also sets the core to the 0 state, does not produce an output.

A magnetic core may be used either as a controllable transformer or as a variable impedance, as shown in Fig. 4.51(a) and (b). In both

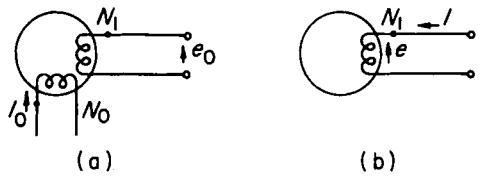


FIG. 4.51. Schematic of a magnetic core in terms of (a) a controllable transformer, (b) a variable impedance

schematics, conventional dot notation is used to indicate winding polarity, with the added definition that current into a dot terminal corresponds to a negative magnetizing force, and will set the core to the 0 state. In Fig. 4.51(b), if the core is in state 1 when current, *I*, is applied the core will switch. A relatively large counter-emf, *e*, will be generated in winding N_1 and the core will look like a relatively large impedance to the driving source. If the core is in state 0 when *I* is applied, the counter-emf will be small and the core will look like a small impedance to the driving source.

4.5.2. TRANSFER LOOPS (Loev *et al.* [1956])

Before continuing with a description of different types of gating circuits, the basic types of transfer loops that may be used for coupling these circuits to one another will be considered. A transfer loop is a circuit

that connects two or more cores for the purpose of information transfer. A transfer loop normally includes an output winding of a transmitting core, an input winding of a receiving core, and one or more diodes. The three types of transfer loops which will be discussed, namely the single diode loop, the split winding loop, and the inhibit loop, permit synthesis of all logical circuitry of a digital information processor.

A basic transfer loop is the so-called single diode loop. It permits permanent storage, and unconditional transfer of information to one or more receiving cores when an advance current is applied to the transmitting core. Such a loop, connecting core A to core B, is shown in Fig. 4.52. According to the symbolic representation defined, if an "input"

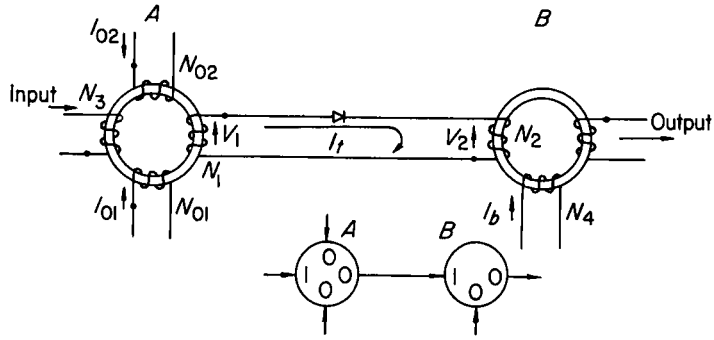


FIG. 4.52. Schematic and symbolic representation of a single diode transfer loop

current is applied to winding N_3, core A will be switched to state 1 (if it is in state 1 at the time the input current is applied, no effect is produced), and if an "advance" current is applied to either winding N_{01} or N_{02}, core A will be switched to state 0 (if it is in state 0 already, no effect is produced). The only condition for which the application of an "advance" current to core A can produce an effect on core B is if at the time of application, core A is in state 1 and core B in state 0. Then, as a result of the advance current being applied to core A, core B will be switched to state 1. This occurs as follows. The advance current, by definition, switches core A to state 0. This induces a signal voltage V_1 in winding N_1, causing a transfer current I_t to flow in the transfer loop. If the diode characteristics and the design of windings N_1 and N_2 are proper, the transfer current flowing through N_2 will be sufficient to switch core B to state 1. Additional receiving cores may be switched by connecting their input windings in series with the transfer loop. To show why the advancing of core B produces no effect on A it is only necessary to consider the case when B is in state 1 and A in state 0. The current I_b will switch B to

state 0 and induce a voltage in winding N_2. This voltage produces a current in the transfer loop tending to switch A to state 1. However, the current is limited to a value less than that required to switch A because of the relatively high impedance of N_1 relative to N_2, and the nonlinearity of the diode which causes it to present a higher impedance to smaller input signals. The reason why the application of current on the "input" line of core A has no effect on core B is that this current induces a voltage across N_1 opposite in polarity to V_1 (see Fig. 4.52), and for which the diode presents a high impedance to the flow of current.

The split winding loop, shown in Fig. 4.53, allows conditional trans-

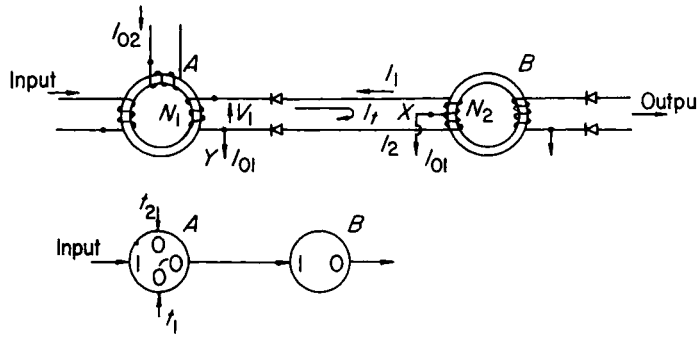


FIG. 4.53. Schematic and symbolic representation of a split winding transfer loop

fer between cores and permits logical operations upon isolated cores. It is only necessary to consider the operation of this circuit for two initial states of A, B. First, consider the case where A and B are both in state 0. When the advance current pulse I_{01} is applied, it divides into branch currents I_1 and I_2. Since A is in state 0, winding N_1 offers negligible impedance. Since I_1 flows into a dot terminal and I_2 into a nondot terminal on equal windings of A, the net magnetizing force on A is nearly zero, and does not change the state of A. If core A is in state 1 and B in state 0 when the advance current is applied, the 1 will be transferred from A to B. By design, N_1 is large relative to other impedances in the transfer loop and I_1 will be much smaller than I_2. The effect is that a transfer current, I_t, equal to one half the difference of I_1 and I_2 flows through a winding of N_2 turns on core B, and sets B to state 1. Branch current I_1 clears A to state 0. Information can be transferred from A to B only during the application of advance pulse I_{01}. At all other times one or the other of the diodes will inhibit the flow of transfer current. A can be switched back and forth between the 1 and 0 states during a sequence of operations without affecting B.

The split winding transfer loop is immune to the backward flow of signals; the switching of the load core B by other inputs cannot send noise current back to core A. Therefore, a single transmitting core in a split-winding transfer loop can switch as many as five or six receiving cores simultaneously. In general, this cannot be done with the single diode loop, because when the receiving cores are later sensed they have an additive effect for the backward flow of current.

The inhibit loop, shown in Fig. 4.54, is a special form of split-winding

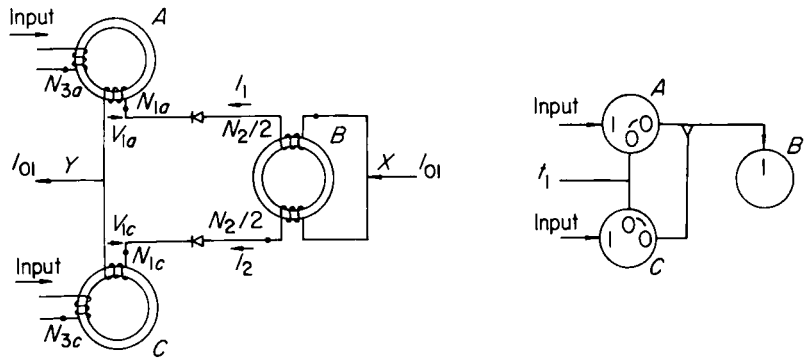


FIG. 4.54. Schematic and symbolic representation of an inhibit transfer loop

circuit interconnecting cores A and C as shown. This type of loop offers a reliable method for conditionally inhibiting the transfer of information from one core to another. If cores A and C are in the same state, no effect will be produced on core B when advance current I_{01} is applied, since $I_2 = I_1$. When A is in state 1 and C in state 0, $I_2 > I_1$ and the net magnetizing force applied to B is adequate to set it to 1. Conversely, when A is in state 0 and C in state 1, B is set to 0. In all cases, A and C will both be in state 0 after application of I_{01}.

The inhibit loop permits the synthesis of certain logical operations that might otherwise be difficult to realize. It also has the isolating advantages described for the split-winding transfer loop.

4.5.3. GATING CIRCUITS.

Magnetic core realizations of the most commonly used gating circuits will be described next. We will consider first the operation of negation. It may be obtained by a suitable choice of the inputs in Fig. 4.50(c). For example, let the input on A be a timing pulse, t_1, which unconditionally sets the core to state 1. If the signal to be negated, p, appears

on line B at time t_2, the core will be set to state 0 (without producing an output), and, therefore, the next shift pulse, t_3, appearing on line C does not produce an output on line D. Conversely, if a p pulse did not occur at time t_2 the application of t_3 will produce an output on line D. Thus, $D = \bar{p}t_3$.

Two methods of realizing the OR function with single diode transfer loops are shown in Figs. 4.55 and 4.56. The circuit in Fig. 4.55 can be used when both transmitting cores are energized by the same advance pulse, and that in Fig. 4.56 is required when A and C are energized by

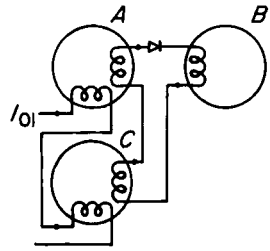


FIG. 4.55. Magnetic core OR gate with single advance pulse input

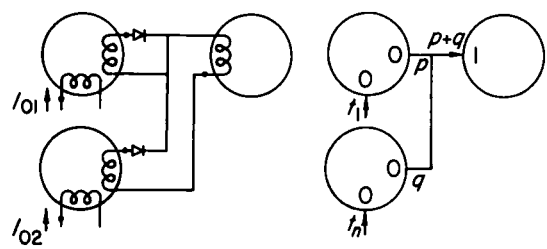


FIG. 4.56. Schematic and symbolic representation of a magnetic core OR gate with two advance pulse inputs

advance pulses occurring at different times. The number of inputs that can be mixed into one receiving core is limited mainly by the additive effects of 0 output voltages from transmitting cores. The OR function can also be obtained by using the split-winding transfer loop, e.g., by connecting the output windings N_1 of the two transmitting cores in series.

Two magnetic core AND circuits are shown in Fig. 4.57. The informa-

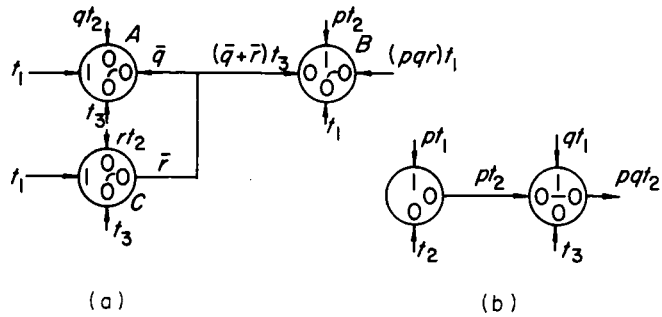


FIG. 4.57. (a) Magnetic core AND gate formed from "negation" circuits, and (b) magnetic core AND gate

tion pulses (which need not occur simultaneously) are designated by p, q, r, and the clock pulses by t_1, t_2, t_3. In Fig. 4.57(a) cores A and C each perform a negation, and comprise an OR circuit which sends a signal to B at time t_3 only if at time t_2 neither q nor r were present. Core B also performs a negation and an output is obtained at time t_1 only if p were present at time t_2, and if neither A nor C had produced a signal at the preceding time t_3. The AND circuit of Fig. 4.57(b) uses the output of one core A to provide the advance current pulse for a second core B. This advance current exists only if p had occurred and can produce an output from B only if q had occurred. Clock pulse t_3 is needed to clear B for the case where q occurs and p does not.

The operation of these core circuits is sequential even within a single switching function unit, in contrast to the operation of diode logical circuits. Therefore, in combining a number of units, it must be arranged that information pulses originating in various units at different times, arrive at a combining unit within a prescribed time interval. One method of satisfying this requirement is to use delay cores, another is to postpone the extraction of data from one or more of the originating units. The first scheme requires extra cores, the second a greater variety of shift (timing) pulses. However, the use of multiple timing sources does not necessarily result in a proportional increase in the number of power amplifiers. In any large system a number of power amplifiers will be necessary. If multiple timing sources are used, these amplifiers simply are distributed differently than in single or double source systems. Some of the advantages of magnetic core logic circuitry are low power and space consumption, high reliability of operation and a long expected life. Also, since cores are low impedance devices they are relatively free from pick-up and cross talk. The limitations of the magnetic core system of logic described are its serial nature, its inflexibility for certain applications, and its presently limited frequency of operation. The first of these limitations may be alleviated by use of multi-input magnetic core gates in conjunction with transistor flip-flops as described in the section following.

4.5.4. A Multi-input Core Gate, Transistor Flip-flop System*

A schematic of a multi-input magnetic core gate is shown in Fig. 4.58. The actual number of control windings on a core is variable. Whereas, in the diode gates described in Section 4.2, the steady state output of each gate was interrogated by a voltage clock pulse, in this arrangement a current clock pulse is used to interrogate the state of magnetization of

* Vorndran and Kaiser [1955].

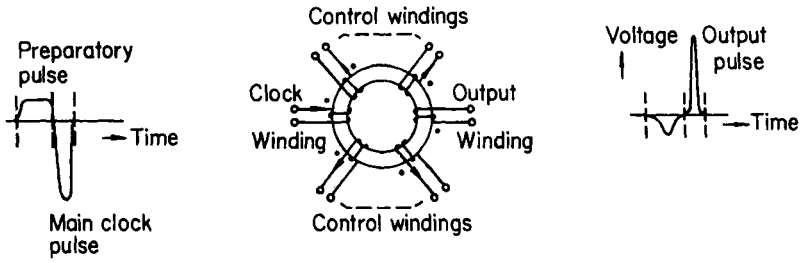


FIG. 4.58. A multi-input magnetic core gate showing the clock and output waveforms

the core. The form of the current pulse applied to each core is also shown in Fig. 4.58. The preparatory clock pulse applies a small positive magnetomotive force to the core and the main clock pulse a large negative magnetomotive force. If there is no current in any of the control windings, application of the composite clock pulse causes the magnetic state of the core to move along a hysteresis loop to saturation, first in the positive direction, then in the negative direction, and finally to be left at a point of negative remanence. Whenever a current is applied to one of the control windings, it is in the direction that produces a negative magnetomotive force. Even if only one control winding has current applied to it, the magnetic state of the core is sufficiently biased in the negative direction to prevent switching of the core to a state of positive saturation by the preparatory pulse. When current is present in more than one control winding, the core is biased even further in the negative direction. When current is not present in any control winding, the application of the composite clock pulse causes a voltage to be induced in the control windings, and the output winding in the form of a small negative signal due to the preparatory pulse, followed by a large positive signal due to the main clock pulse. When current is present in one or more control windings, the voltage induced in these windings is essentially zero. If the presence of current in a control winding is defined to represent 1 and the presence of a voltage pulse in the output winding also represents 1, then the arrangement shown in Fig. 4.58 acts as a multi-input NOR gate. An OR gate may be formed by connecting the output windings of two or more cores in series.

The currents in the cores may be controlled by vacuum tube or transistor flip-flops. A number of control windings, on different cores, can be connected in series across the output of a single flip-flop, though the voltage induced across these control windings by the flow of current in other windings must be considered in the design of the flip-flop. Either

a single output winding, or a number of them in series serve as inputs to a set-reset flip-flop. Accordingly, the output windings are referred to as trigger windings. When a 1 is produced on a trigger winding, the small negative pulse is blocked by a series diode and the large positive pulse used to trigger a flip-flop. When a 0 is produced on a trigger winding, the output signal may be considered negligible for a single core. However, when a 0 is produced on each of several trigger windings connected in series, the effect is compounded. The total number that can be so connected to a flip-flop input is limited by the voltage ratio of a 1 and a 0 on a trigger winding. Also, a clamping diode may be required on the flip-flop input if a 1 signal can appear on several series-connected trigger windings simultaneously. In both of the cases described, a larger number of input trigger windings can be accommodated by separating them into two or more series combinations.

The number of control windings per core is limited by the wire size, and, if a transistor flip-flop is used, the cutoff current of the output transistor. When all the control windings on a core are in the 0 state, the magnetomotive force due to the sum of these currents must be considerably less than that produced by a 1 on one control winding. Otherwise, the core will be negatively biased to the point where application of the clock pulse will not switch it.

Though each control winding is functionally equivalent to a diode in a diode gate, whether this represents a saving is questionable because of the expense of forming these windings compared to the very low price of diodes for frequencies under 250 Kc, the approximate limit of operation of the core circuits. While the only gating power required is clock pulse power, even at these frequencies there are problems in generating and transmitting the large currents required. The system also suffers from inflexibility in several respects. For example, because the loading on the gates is critical, the actual load must be individually computed for each network, and a change in one or more switching functions usually necessitates a recomputation of the loading. Also, if pyramiding is employed, in general more than one clock period is required to generate a function (since only OR combinations of gate outputs can be formed without the use of additional gates) and, conversely, if all switching functions are to be formed in one clock period, the savings of pyramiding cannot be realized. Though the system is flexible with respect to the choice or number of dc voltage levels, because the flip-flop inputs and outputs are ac coupled through the gate windings, the multi-input core gate did not come into any appreciable use for logical circuitry because of its limitations compared to other types of circuitry. For example, the transistor NOR circuits (section 4.4.2.3) are comparable in cost, do not have the logical deficiencies described, and permit a higher frequency of operation.

4.5.5. The Transfluxor

The transfluxor* is a magnetic core device composed of a material with a nearly rectangular hysteresis loop and having two or more apertures. It is similar to other core devices in that its response to an input signal is controlled by a setting pulse which it previously received and stored. However, the process of generating an output in the transfluxor does not affect the setting pulse, so a single setting pulse can be stored and then used to control the device indefinitely. The transfluxor can be used not only as an on–off switch, but also can be set to yield an output at any desired level between "off" and maximum "on".

The principles of operation and general properties of the transfluxor will be described briefly with reference to a two-aperture unit as shown in Fig. 4.59. The two circular apertures form three distinct legs in a mag-

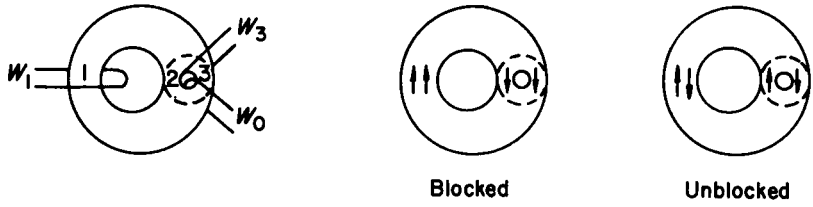


FIG. 4.59. Schematic of a two-aperture transfluxor and symbolic representation of blocked and unblocked states

netic circuit. The windings w_1, on leg 1 and w_3 and w_0 on leg 3 are shown, for the sake of simplicity, with single turns. Assume that a current pulse is applied to w_1 of direction and magnitude such that there results a clockwise flow of flux which saturates legs 2 and 3. Consider now the effect of applying an ac current to w_3, producing an alternating mmf along a path surrounding the smaller aperture. When this mmf has a clockwise or counterclockwise sense, it will tend to produce an increase in flux in leg 3 and a decrease in leg 2, and vice versa, respectively. In neither case are increases in flux possible since the legs are already saturated. Consequently, there can be no flux flow. When the transfluxor is in this state, no voltage is induced in the output winding w_0, and the transfluxor is said to be in a blocked state.

If there is applied to w_1 a current pulse of direction and magnitude such that a counterclockwise mmf is generated intense enough to produce a mmf in closer leg 2 larger than the coercive force H_c, but not large enough to allow the mmf in leg 3 to exceed the critical value, the saturation of leg 2 will reverse but leg 3 will not be affected. In this condition, the alternating mmf around the small aperture resulting from the ac cur-

* Rajchman and Lo [1956], [1955].

rent in w_3 will produce a corresponding flux flow around the small aperture. This flux flow, which may be thought of as a back-and-forth transfer of flux between legs 2 and 3, will reverse indefinitely with each reversal of phase in the ac current. The alternating flux flow induces a voltage in the output winding w_0 during this "unblocked" or "maximum set" state of the transfluxor.

To summarize, the transfluxor is blocked when the directions of remanent induction of the legs surrounding the smaller aperture are the same, and unblocked when they are opposite. In the blocked state the magnetic material around the small aperture provides essentially no coupling between the primary, w_3, and secondary, w_0, windings, while it provides a relatively large coupling between these two windings in the unblocked state.

A limit is imposed on the permissible amplitude of the ac by the fact that, when in the blocked state, a sufficiently large ac in the phase tending to produce counterclockwise flux flow could change the flux in leg 3 by transferring flux to leg 1. This limit is increased by the use of unequal hole diameters, making the flux path via legs 1 and 3 much longer than via legs 2 and 3.

The driving (clockwise) pulses, which cannot unblock a blocked transfluxor can be arbitrarily large. As a result, when the transfluxor is unblocked by proper setting, these pulses may not only provide the required minimal reversing magnetizing force around the small aperture, but also substantial power to deliver large output currents. The priming (counterclockwise) pulses must be of sufficient magnitude to provide the required magnetizing force around the small aperture, but insufficient to provide it around both apertures.

In the preceding description of the on–off operation of the transfluxor a "maximum set" state was referred to. However, the transfluxor can also be set to any level in a continuous range. Once set, it will deliver indefinitely an output proportional to the setting.

When priming and driving, the flux in leg 1 is not affected by the interchange of flux between legs 2 and 3, so there is no coupling between the input and output circuits. When setting, there is also no coupling between input and output circuits because no flux is changed in leg 3, but only in legs 1 and 2. The same applies when blocking occurs after driving, rather than priming, since the drive pulse has already saturated leg 3 in the direction of blocking and no further flux change is possible.

A hysteresis loop as rectangular as possible is desirable because: (1) it lessens the undersirable interchange of flux in the blocked condition due to imperfect saturation of legs 2 and 3; (2) it lessens the coupling

between legs 1 and 3, providing better isolation of control and output circuits; (3) it provides a sharper threshold for the setting current pulse, thereby making the transfluxor more suitable for switching applications.

The transfluxor exercises control by means of the amount of flux which can be transferred for an indefinitely long time between legs 2 and 3, and which can be set by a single pulse to any desired value in a continuous range. It operates as if the output magnetic circuit consisted of a conventional one-apertured core with the essential property that the effective cross-sectional area of that core can be adjusted by a single set pulse to any desired value from practically zero to a maximum value equal to the physical cross-sectional area of its smallest leg.

The use of more than two apertures creates many new modes of flux transfer and increases the kind and number of switching and storage functions. Consider the three-aperture transfluxor shown in Fig. 4.60.

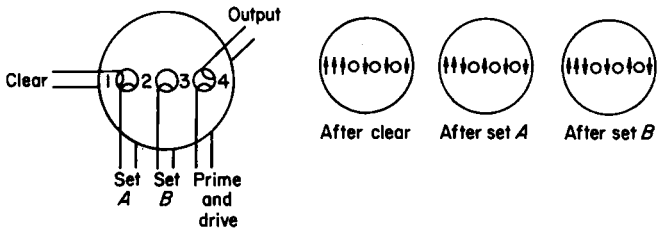


FIG. 4.60. Schematic of a three-aperture transfluxor and symbolic representations of different magnetic states

This device can be operated as a two-input sequential gate, i.e., an output is produced if the two inputs A, B are applied in the order A, B, and no output is produced if they are applied in the order B, A or if one input is absent. The operation is as follows: After a clear pulse, the legs, 2, 3, and 4 are saturated downward. The output flux path around the last aperture via legs 3 and 4 is blocked and neither the prime nor the drive pulse can produce any flux change. The flux path around the second aperture via legs 2 and 3 is also blocked so that the signal B cannot produce any flux change. However, the flux path around the first aperture via legs 1 and 2 is not blocked and the signal A can reverse the direction of flux in leg 2 by transfer of flux to leg 1. If A were present and leg 2 were reversed, the flux path via legs 2 and 3 is unblocked, with the result that the occurrence of B can now reverse the flux of legs 2 and 3. This returns leg 2 to its original downward direction, reverses leg 3 and unblocks the flux path via legs 3 and 4. The output flux path is now unblocked,

and a succession of priming and driving pulses will produce an output indefinitely.

A five-aperture transfluxor is shown in Fig. 4.61. This device may be

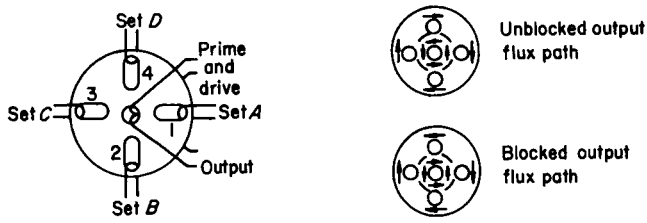


FIG. 4.61. Schematic of a five-aperture transfluxor and symbolic representations of blocked and unblocked output states

operated as a four-input AND gate. The occurrence, in any order, of all four input signals is required to open the gate. The operation of the unit is based on the fact that the output flux via legs 1, 2, 3, and 4 around the central aperture can be blocked by any one of the four legs, and is unblocked only when the sense of flux saturation around the central hole is the same in all legs. There are two unblocked states corresponding to two senses of flux rotation around the central aperture. One of these states may be eliminated by using one leg as a reference, yielding a three-input gate.

4.6. Superconductive Switching Elements

Circuit elements still under development which show great promise for computer applications are those based on the use of superconductive materials. In metals not classified as superconductors the electrical resistance drops as the temperature is lowered until a point is reached (in the low temperature region above absolute zero) where the resistance remains constant as the temperature is lowered further. In a superconductor the resistance in the low temperature area drops abruptly from a value that varies between 10^{-1} and 10^{-3} of room temperature resistivity to a value which, as far as has been determined, is equal to zero. The point at which this occurs is referred to as the critical temperature, T_c, or superconductive transition temperature.* Another important property of superconductors is that the application of a magnetic field greater than a critical value, H_c,

* There are 21 elements, as well as a number of alloys and compounds, that are superconductors with transition temperatures ranging between 0°K and 17°K.

destroys superconductivity. The value of H_c in the region $0°K \leqslant T \leqslant T_c$ is approximately equal to $H_0\ [1\text{-}(T/T_c)^2]$, where H_0 is the critical magnetic field at $T = 0°K$. The critical magnetic field curves in Fig. 4.62

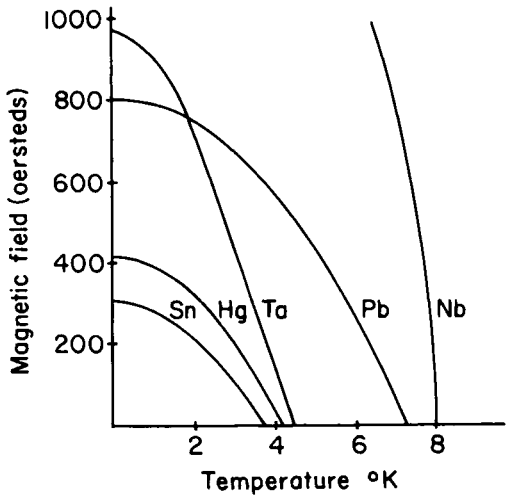


FIG. 4.62. Critical magnetic field, H_c, curves for several superconductors

show that increasing the magnetic field lowers the transition temperature. The area below each curve represents a region of superconductivity and the area above a region of normal resistivity. For example, if lead is held at a constant temperature of 5°K in a magnetic field of 200 oersteds, it will display superconductivity, while if the applied field is increased to 600 oersteds the superconductivity will be destroyed. The existence of a superconductive transition temperature and a critical value of magnetic field provides the nonlinearity required for switching. A superconducting switching element called a cryotron, invented by D. A. Buck, at M.I.T., is based upon exploitation of these phenomena. The temperature of the element is held constant and it is switched from a superconducting to a normal state by application of a magnetic field greater than H_c. For each superconductor, a choice of temperature about 0.2°K less than the zero field transition temperature allows operation with small magnetic fields—between 50 and 100 oersteds. For tantalum, this operating temperature is about 4.2°K. The fact that this corresponds to the boiling point of helium at a pressure of 1 atm., and therefore the temperature of most storage tanks for liquid helium, is one reason for the use of tantalum in the early experiments.

In an early experimental form, the cryotron consisted of a 1-in. strip of 0.010 in. tantalum wire inside a single layer control winding of 0.003 in. insulated niobium wire (shown schematically in Fig. 4.63(a)). Current

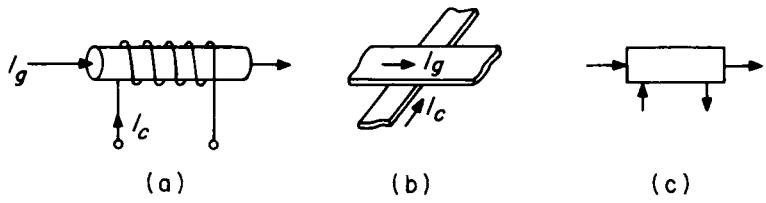


FIG. 4.63. (a) Wire-wound cryotron, (b) crossed-film cryotron, (c) block diagram of a cryotron

in the control winding creates a magnetic field which causes the central wire, designated as the gate wire, to change from a superconducting to a normal state. The control winding is a superconductor with a relatively high transition temperature. Niobium was used for this reason and because of its good ductility and tensile strength. Because of its relatively high value of critical field at the operating temperature, the control winding remains in a superconducting state at all times. Average power dissipation per cryotron is about 10^{-4} watts. A unique property of the cryotron compared to vacuum tubes or semiconductors is that control is independent of the sign of the control current and, therefore, when the gate circuit is in its superconducting state, there may be current flow in either direction.

Before continuing with a description of the cryotron, two other important properties of superconductors will be described: (1) The Meissner effect, namely, because in the superconducting state the magnetic induction is zero, there must be superficial currents which produce an internal field that cancels the applied field. (Actually the currents occupy a thin surface layer and the magnetic induction decreases to zero within a very small penetration depth which for most superconductors is of the order of 10^{-5} cm), (2) If the amount of current a superconductor carries exceeds a certain limit, the superconductivity will be destroyed. For conductors whose dimensions are large compared to the penetration depth, the value of this critical current, I_c, is that which produces a magnetic field, H_c, at the surface. The variation of I_c with temperature is similar to that of H_c.

The cryotron acts as a current amplifier since the resistance of the gate can be varied to control current. The condition for current gain is that the amount of current in the control winding that produces a critical

field is less than the amount of current in the gate circuit that destroys superconductivity. When $(4\pi n) > (2/r)$, n being the number of turns per unit length of the control winding, and r the radius of the gate wire, one element can be driven by another without intermediate current transformation.

In pulse circuits, the gate current of one cryotron becomes the control current of another. For this condition, the frequency at which the power gain becomes unity is given by R_g/L_c, where R_g is the normal resistance of the gate circuit, and L_c the inductance of the control winding. Though L_c and R_g are on different cryotrons, the L/R time constant (assuming all cryotrons in a network have the same values of L and R) serves as a convenient measure of potential operating speed. L/R is independent of the cryotron's length, (for L and R each increase linearly with length), and decreases as the fourth power of the diameter (for L decreases as the square of the diameter, and R increases as the inverse square).

In practice the frequency of operation of the cryotron turns out to be less than what would be expected from consideration of the L/R time constant alone. Current through a gate normally conducting raises its temperature (in some cases above T_c) and reduces the value of H_c. Because the control winding acts as a thermal insulator, the thermal time constant may be several hundred microseconds. Its relative importance can be reduced by increasing the number of turns on the control winding. This reduces the amount of control current required to produce a critical magnetic field at the expense of increasing L and the L/R time constant. Eddy current effects, which become relatively greater for smaller gate diameters, impose a further limit on the speed obtainable with a wire-wound cryotron. Magnetic flux, excluded from the gate wire in the superconducting state, penetrates into the wire when the superconductivity is destroyed by a magnetic field greater than H_c. The region of normal conduction starts as a thin outer shell, and as the flux moves inward, eddy currents are induced in this region. The heat generated by the eddy currents retards the growth of the normal phase. A similar phenomenon occurs in the normal to superconducting transition. Even with the smallest wire practical, delays resulting from eddy currents limit the transition speed to about a microsecond.

Various schemes have been proposed to increase the speed of the wire-wound cryotron: Since resistivity in the normal state varies over several powers of ten among various superconductors, an increase in speed may be obtained by using alloys of greater resistivity. Another scheme is based on the fact that the superconductive currents are within a thin surface layer. Thus, removal of the core of the gate wire would increase the resistance in the normal state (in proportion to the ratio of

original to new cross-sectional area) without impairing operation in the superconducting state. An equivalent arrangement would be to deposit a thin layer of superconducting material on an insulating wire. However, because of basic speed limitations, as well as difficult production problems encountered in miniaturization, further development of the wire-wound cryotron has been discontinued. For faster switching, one of the dimensions of the cryotron must be greatly reduced. An approach currently being investigated is the use of thin metallic films, deposited on a glass substrate, for both the control and gate conductors. (Thin film memory devices are described in Sections 5.3.6 and 5.3.7). The use of stencils in the deposition permits construction of several complete circuits at a time and, therefore, greater miniaturization and simpler fabrication. It is estimated that about 1000 thin film cryotrons could be contained in a cubic inch.

Films of lead and tin with thicknesses of 10^{-5} cm or less are easily evaporated. Tin is used for the gate and lead for the control conductor because H_c is much less for tin (Fig. 4.62 shows that at 3°K these values are 100 and 700 oersteds, respectively). In the crossed film cryotron (shown schematically in Fig. 4.63(b)) a gain > 1 is achieved by using a control strip much narrower than the gate strip. L/R time constants of about 100 mμsec appear to be obtainable with this type of arrangement.

A disadvantage of tin is that because its zero field value of T_c (3.7°K, see Fig. 4.62) is less than 4.2°K, the boiling point of helium at atmospheric pressure, the helium cryostat must be operated at a reduced pressure. The use of tantalum instead would eliminate this difficulty. However, suitable evaporated tantalum films are very difficult to produce because of tantalum's high melting point and its excellent properties as a getter. Films which have been produced thus far have exhibited anomalies which, it is believed, are due to impurities introduced during evaporation. For example, films formed at 10^{-5} mm Hg did not become superconducting even when cooled to 1.5°K, and films produced at about 10^{-6} mm Hg became superconductive only when cooled to 1.6°K. For bulk tantalum, the zero field value of T_c is about 4.4°K, as shown in Fig. 4.62. For this reason, considerable effort is being expended on developing the technology of producing improved thin films. At M.I.T., equipment is being developed to allow evaporation of films at a pressure of 10^{-10} mm Hg.

Future developments will depend on a better understanding of the phenomena of superconductivity, further data on the mechanism of superconducting circuits, and a study of the mechanism by which evaporative films are formed and the problems of producing films with desired physical properties. Until the various factors that influence the structure of the film are better understood, it is difficult to tell whether observed super-

conducting properties are representative of films in general or only of those produced by a specific process. Also, the use of new alloys and circuits that exploit the special characteristics of the thin film configuration can lead to improved switching times.

A flip-flop can be formed from two cryotrons by placing each gate circuit in series with the control winding on the other cryotron and connecting the two branches in parallel. If both branches are superconductive, the current divides equally. However, if one is superconducting and the other even partly in the normal state, all the current from the supply flows in the superconducting branch. Figure 4.64 shows a block diagram

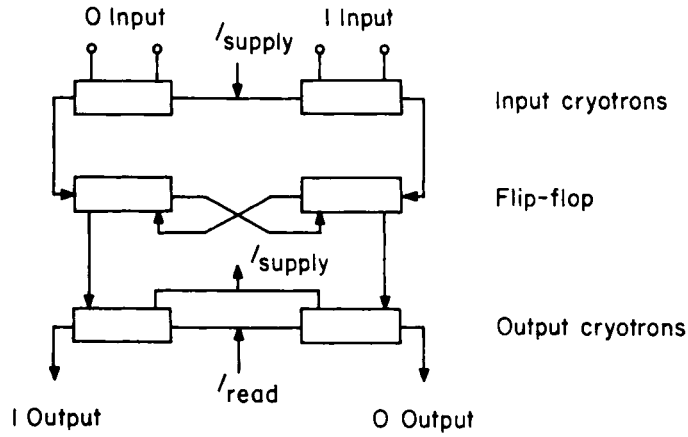


FIG. 4.64. Cryotron flip-flop with input and output cryotrons

of a cryotron flip-flop with input and output circuits. A pulse applied to the control winding of one input cryotron drives that branch resistive, eventually causing all the current to flow in the other branch. This represents one stable state. A pulse applied subsequently to the other input cryotron establishes the opposite stable state. If a number of input cryotrons are connected with their gate circuits in series, they function as an OR gate, and if connected with their gate circuits in parallel, as an AND gate. For each stable state of the flip-flop, one of the read-out cryotrons will be resistive and the other superconductive. A read-out pulse, applied at the junction of the read-out cryotron gate circuits, chooses a path accordingly.

4.7. Computing Elements for Gigacycle Operation

We will consider here briefly two types of circuits developed for operation at Gc frequencies (10^9 $\sim$), One, the parametric oscillator, is

based on a nonlinear circuit parameter, and is an example of a microwave circuit operating within a frequency band not extending to zero. The other is based on the negative resistance property of the tunnel diode and, like most of the circuits described earlier, uses base-band signals, occupying a band from zero (or near zero) to some upper limit.

In the phase locked subharmonic oscillator, energy is transferred by a nonlinear element from a pump frequency to a lower frequency whose relative phase represents the information. The nonlinear capacitance of a special semiconductor diode, referred to as a microwave or parametric diode, is used because of the diode's small size and suitability for operation at extremely high frequencies. The capacitive reactance is varied at the pump frequency, which is an even multiple (usually two) of the characteristic frequency of the tank circuit. The tank is controlled by injection of a steering signal which excites oscillation in either a 0° or 180° phase. These two phases represent the two values of a binary variable. Since the rise time of a signal takes at least 10 cycles of the carrier frequency an information rate of 1 Gc implies a fundamental frequency in excess of 10 Gc. The tolerances necessary in the pump and signal circuits are severe, and emphasis has shifted to tunnel diodes and thin film cryotrons.

The tunnel diode, so called because its operation is based on the quantum mechanical tunnel effect (which describes how electrical charges move through the device, see Esaki [1958]) is a two terminal device made from a heavily doped semiconductor, i.e., one with impurity densities 10^4 to 10^5 that of conventional p–n junction diodes. The characteristic of the tunnel diode which is exploited is the stable negative resistance that appears over part of the volt-ampere operating region (see Fig. 5.22(a)). The two regions on either side can be defined to represent the two binary states. Power gain can be realized when switching from a low voltage state to a high voltage state because a large change in output current can be obtained by means of a small input trigger current that drives the operating point over the maximum of the characteristic. One or two tunnel diodes can serve as a threshold majority gate and single bit store, provided there is suitable biasing and loading. Germanium tunnel diodes have been reported with switching times less than 1 nanosec. This speed is attributable to the fact that signals are transmitted within the diode by an electromagnetic field rather than the motion of charged carriers. There are other advantages too. They are smaller than transistors, operate over a temperature range greater than that of germanium and silicon transistors combined, and at nuclear radiation levels about two orders of magnitude greater than practical with transistors. Because of its inherent simplicity and low cost of fabrication it is potentially a low cost element.

To overcome the tunnel diode's bidirectional action in signal propagation, an auxiliary transistor or tunnel rectifier may be used. One hybrid circuit, a two-input NOR gate, uses a tunnel diode for threshold action and current gain, and a transistor for isolation between signal channels. Because gain is not sought from the transistor, it can be operated near its cutoff frequency. The circuits are connected by matched microstrip transmission lines. Each input signal is applied to a resistor, in the emitter circuit of the isolation transistor, which approximates a matched termination and keeps signal reflection to a minimum. The tunnel diode is coupled directly to the transistor collector, and capacitively to a 500 Mc sine wave current source used to retime and reshape all output signal waveforms. Because of phase inversion in each gate, two forms of the circuit are used—one with a p–n–p and one with an n–p–n transistor. In the absence of input signals, the diode in each circuit switches on opposite half cycles of the sine wave. Thus, two levels of logic can be performed during a full clock cycle.

At 1 Gc each operation takes 1 nanosec. It takes this much time for electromagnetic energy to propagate 1 foot in free space. Thus, in machines operating near this rate, propagation time becomes a limiting factor and the computing elements must be packed within a small space. To fully exploit the potentialities of microwave computer elements requires, among other things, the continued development and refinement of means of interconnecting these elements, e.g., the use of low impedance microstrip transmission lines and thin film conductors. With present low impedances represented by tunnel diodes at microwave frequencies there are parasitic inductances associated with the leads and as a result it may be necessary to solder the diodes directly between the conductors of a low impedance microstrip line. Parasitic oscillations at very low frequencies can be produced if the impedance of the bias supply is not maintained at a low enough level (dc instability).

4.8. Specialized Switching Networks

4.8.1. Translational Networks

Translational networks are used most often to translate a coded representation of data from one form to another in order to mechanize certain control functions. These networks are also referred to as switching matrices or function tables, because the elements of the network are often arranged schematically, or even physically, in an array resembling a matrix or table. A translational network is commonly constructed almost entirely from diodes and resistors.

As stated earlier, control signals are commonly obtained from the value of code bits stored in a group of flip-flops. Translational networks may be used to satisfy either of two basic types of requirements. In one, it is required that the presence of each of a set of permissible values of a code group causes one or more control lines to be energized. In the other, it is required that the presence of a signal on a given line causes two or more other lines to be energized. Translational networks satisfying each of these requirements are often referred to as many-to-one and one-to-many networks, respectively. In the general case, each of the permissible values of a coded group of bits must cause more than one output line to be energized. This calls for a many-to-many switching network, which, as will be shown later, can be constructed from a combination of a many-to-one and a one-to-many network.

The many-to-one network, also referred to as a decoding network, is devised so that for each combination of conditions on its input lines, only one of its output lines is energized. For a given output line to be energized, all input lines to which it is connected must be energized. In Fig. 4.65(a) is shown a schematic of a representative network, with two input variables and four output lines. Inspection of Fig. 4.65(a) shows

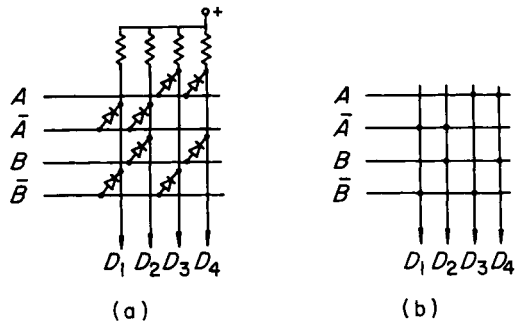


FIG. 4.65. Representations of a many-to-one rectangular switching matrix with two input variables

that each of the output lines D_1 through D_4 corresponds to the output of an AND gate. Specifically, the output signals have the following values

$$D_1 = \bar{B}\bar{A} \qquad D_3 = \bar{B}A$$

$$D_2 = B\bar{A} \qquad D_4 = BA$$

It is apparent then that a many-to-one switching network is nothing more than an assemblage of AND gates. The utility of considering the assemblage rather than the individual gates will be more apparent after the ensuing discussion.

For purposes of convenience, a network of the type shown in Fig. 4.65(a) may be represented symbolically as in Fig. 4.65(b). In Fig. 4.65(b), the resistors have been omitted, and the diode connections have been replaced by dots, with the understanding that all the dots on a vertical line represent inputs to an AND gate. Figures 4.66, 4.67, and 4.68 show three of the different forms that a matrix with three input variables can assume. Note that in Fig. 4.66 there are eight three-input AND gates, and in Figs. 4.67 and 4.68, there are 12 two-input AND gates. The number of diodes required in all three arrangements is the same. However, in Fig. 4.66, there is equal loading of the input signals. In Fig. 4.67, there is unequal loading of the input signals, for C and $\bar{C}$ appear as inputs to many more AND gates than either A and $\bar{A}$ or B and $\bar{B}$. In Fig. 4.68, there is more balanced loading of the input signals. In both Fig. 4.67 and

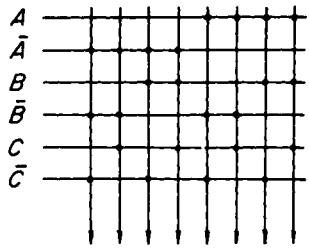


FIG. 4.66. A many-to-one rectangular switching matrix with three input variables

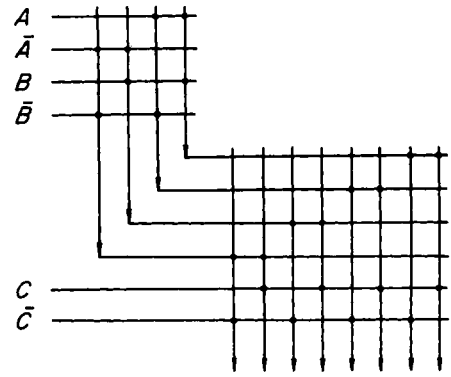


FIG. 4.67. A many-to-one pyramidal switching matrix

Fig. 4.68 use is made of pyramiding, i.e., the construction of complex

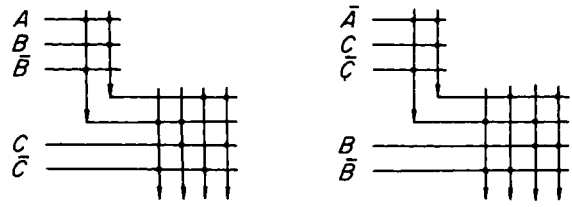


FIG. 4.68. A many-to-one pyramidal switching matrix

Boolean functions from simpler ones already available (see Section 4.2.5). For example, in Fig. 4.67, the functions $A\bar{B}\bar{C}$ and $A\bar{B}C$ are not formed birectly from the elementary inputs A, $\bar{B}$, C, and $\bar{C}$, but, instead, by combining each of the inputs C and $\bar{C}$ with the output of the AND gate generating $A\bar{B}$. The use of pyramiding does not provide a saving in diodes for three input variables, but for more than three input variables the savings in the number of diodes, compared to the rectangular matrix, increases. Table 4.1 shows the number of diodes required for a rectangular matrix, and also for a pyramidal matrix (assuming all minterms (logical products) of n variables are to be formed). In general, pyramidal arrangements may be formed as follows. First of all, the minterms of two input variables are formed. Each of these is combined with a third variable. Each of the minterms of three input variables is combined with a fourth variable, etc.

Neither the rectangular nor pyramidal matrix results in a translational network with the minimum number of diodes if n is greater than three. The minimum network can be obtained by the following procedure. First, the full set of variables is separated into two groups (equal if the number of variables is even and differing by one if odd). This partitioning process is continued until each group contains either two or three variables. The number of diodes is the same whether a rectangular or pyramidal matrix is used for each two and three variable group. The outputs of two groups are combined by an array of two-input AND gates, the process being repeated until all minterms have been formed. For large n, the number of ways of combining the groups becomes large and, in general, different combinations require different numbers of diodes. All possible combinations must be considered to determine the network with the minimum number of diodes. The form of matrix most economical of diodes for four input variables is shown in Fig. 4.69. The column on the far right

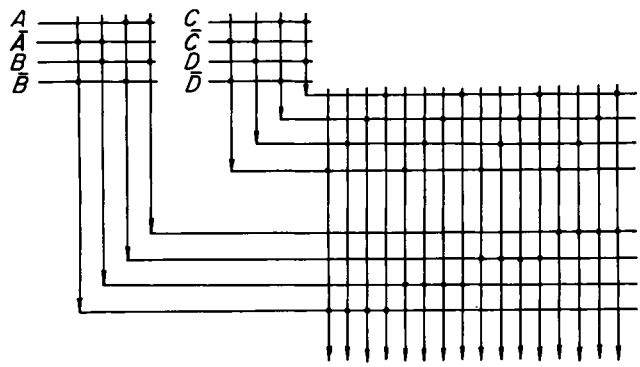


FIG. 4.69. The most economical form of a four-input many-to-one switching matrix

in Table 4.1 shows the minimum number of diodes required for a given number of input variables.

TABLE 4.1. Number of diodes, N, required for different forms of translational networks

Number of input variables	Rectangular matrix	Pyramidal matrix	Minimum network
n	$N = n(2^n)$	$N = 2^{n+2} - 8$	$2^{n+1} \leqslant N \leqslant 2^{n+2} - 8$
2	8	8	8
3	24	24	24
4	64	56	48
5	160	120	96
6	384	248	176

In the one-to-many network, also referred to as an encoding network, each of several energizable input lines may be used to energize all the output lines connected to it. These output signals can be used to cause a group of operations to be executed elsewhere in a system. The schematic of a one-to-many network with four input and five output lines is shown in Fig. 4.70. The control functions to be performed by this network may be expressed as follows

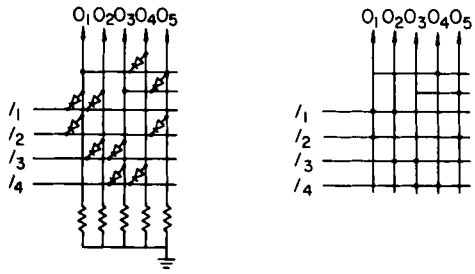


FIG. 4.70. Representations of a one-to-many switching matrix with pyramiding

A signal on line I_1 is to energize lines O_1, O_2, O_3

A signal on line I_2 is to energize lines O_1, O_3, O_5

A signal on line I_3 is to energize lines O_2, O_4, O_5

A signal on line I_4 is to energize lines O_3, O_4, O_5

These relations can also be expressed by considering all the input signal conditions that cause a particular output line to be energized. In this particular case

$$O_1 = I_1 + I_2$$
$$O_2 = I_1 + I_3$$
$$O_3 = I_1 + I_2 + I_4$$
$$O_4 = I_3 + I_4$$
$$O_5 = I_2 + I_3 + I_4$$

These relations show that a one-to-many network is simply an assemblage of OR gates. Any of the forms of the many-to-one matrix can also be used for the one-to-many matrix. It is only necessary to reverse the orientation of all diodes and return the resistors to a voltage of opposite polarity. Fig. 4.70 provides an example of pyramiding – O_3 being formed from an OR combination of O_1 and I_4, and O_5 from an OR combination of O_4 and I_2.

A many-to-many network can be readily constructed by using the output lines of a many-to-one network as inputs to a one-to-many network. One application of such a network in a digital computer would be to control the execution of the elementary commands called for by an instruction. A group of flip-flops can store the codes of various instructions, and a many-to-one network used to energize a unique output line for each code. By using the output lines of the many-to-one network as the inputs to a one-to-many network, a set of control lines can be energized for each instruction code. (See the discussion on microprogramming in Chapter 7.)

A many-to-many network can also be used as a mathematical function table. In this case, the network is so constructed that when the code of the argument is entered on the input lines, the pattern of signals produced on the output lines corresponds to a binary coded representation of the function. However, this is a very expensive and impractical type of mechanization for tables of appreciable size. Instead, tables of functions are usually stored in large capacity storage systems of the types described in Chapter 5.

4.8.2. Distribution and Collection Networks

If a computer has both dynamic and static storage units, there must be some means of converting from dynamic to static storage and vice versa. The former process is referred to as distribution (or staticizing) and the latter as collection. Distribution and collection are also referred to as serial-to-parallel and parallel-to-serial conversion since information is converted from a form in which there is access to only one bit at a time to a form in which there is access to several bits at a time, and

vice versa. A schematic of a distribution unit is shown in Fig. 4.71. D_1, D_2, and D_3 each represent a delay equal to that between successive bits. The signal, S, controls the gates G_1 through G_4 and is chosen to represent the time at which the contents of the delay line, composed of the individual delays, are to be sampled and transferred to the flip-flops, FF_1 through FF_4. Since information is advancing serially bit by bit from the input through the delay line, it is clear that the information that is transferred to static storage will be determined by the time of occurrence of the signal S. The length of the delay line in unit pulse times must, of course, equal the number of pulses to be staticized at a time.

A schematic of a collection unit is shown in Fig. 4.72. Information is

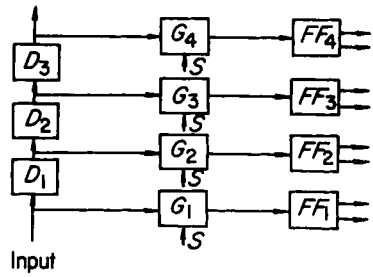


FIG. 4.71. Schematic of a distribution unit

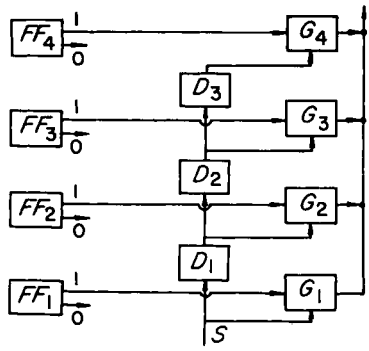


FIG. 4.72. Schematic of a collection unit

stored in the flip-flops. The signal S passes through the delay channel, arriving sequentially at the inputs to gates G_1 through G_4. Thus, at suc-

cessive times, a signal is entered onto the output line by each gate connected to a flip-flop in the 1 state.

If the period between successive bits is more than a few μsec, the delay line becomes too large, and distribution and collection are accomplished by means of reading information from a dynamic store into a shift register and stepping information from the register into the store, respectively. The shift register combines the properties of storage and unit delays. A description of various shift registers is provided next.

4.9. Shift Registers

Each stage of a shift register is built around a bistable element, together with control circuitry that, upon command, causes the contents of each stage R_i to be transferred to stage R_{i+1}. These elements may be vacuum tube or transistor flip-flops, magnetic cores, ferroelectric cells, or any other form of binary storage device. Shift registers are widely used in computers for storage, shifting, delay, serial-to-parallel and parallel-to-serial conversion, etc. If parallel input lines are provided, it can serve as a parallel-to-serial conversion device. Parallel readout lines enable it to be used as a serial-to-parallel converter.

The serial shift register serves principally to provide buffer storage, accepting information when available and delivering it when desired. It can also provide speed buffering by being pulsed at one rate when receiving information and at another when delivering it. Whenever a shift command is received, each stage is cleared and made to accept the previous contents of the preceding stage. If used simply as an external read-in device, shift pulses will be received every time a new bit is to be entered.

Information is shifted down the register by pulsing all stages simultaneously, thus causing each stored bit to advance simultaneously. The information in each stage must be stored somewhere while the following stage is being cleared. This intermediate storage may be by means of capacitors, electromagnetic delay lines, or an auxiliary flip-flop register. (Often the turnover time of the flip-flop itself provides an adequate delay.) The use of an auxiliary flip-flop register, with appropriate gating, permits bidirectional shifting. The operation of several types of shift registers will now be described.

In the shift register shown in Fig. 4.73 the content of each bistable storage element is shifted to the next higher order element when a shift pulse is applied on the line indicated. The output of each R_i is connected to the input of R_{i+1} by means of a gate which retains the state of R_i for a short time after its contents have been altered, and passes the state of R_i on to R_{i+1}.

Figure 4.74 shows an arrangement using inductive delays for inter-

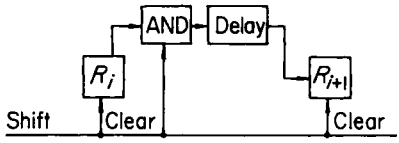


FIG. 4.73. A logical arrangement for a shift register

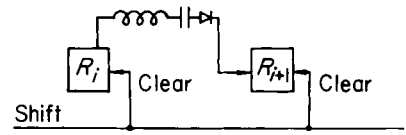


FIG. 4.74. A shift register with inductive intermediate storage

mediate storage. A clear pulse is applied to all stages, and causes any stage in the 1 state to emit a pulse. This pulse, after being delayed, resets the next higher order stage after a time greater than that necessary for it to recover from the previous clear operation.

In the arrangement of Fig. 4.75 the shift register proper is comprised

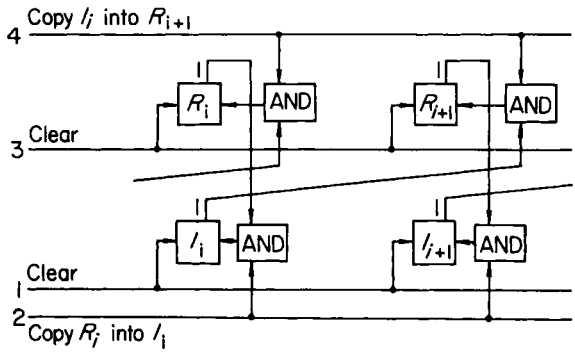


FIG. 4.75. A high speed dc coupled shift register with flip-flop intermediate storage

of one group of storage elements, R_i, while the storage elements, I_i, are used for intermediate storage. A single shift operation is effected by applying command pulses to input lines 1, 2, 3, and 4, in that order. A pulse on line 1 clears the intermediate storage register. The pulse on line 2 copies any 1's in the main shift register into the intermediate register. The pulse on line 3 clears the main shift register. The pulse on line 4 causes the 1's in the intermediate storage register to be copied into the shift register one bit to the right of their original positions.

An alternate logical arrangement is one wherein gates are provided for transferring 0's as well as 1's between stages. Such an arrangement is shown in Fig. 4.76. A pulse will pass through only one of the gates,

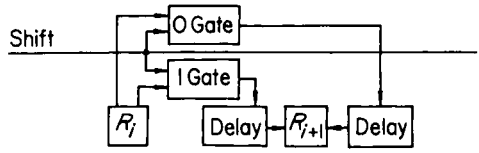


FIG. 4.76. Shift register with gates for transferring 0's and 1's

depending on which state the corresponding bistable element is in. The delays prevent a bistable element from being triggered before a satisfactory signal is transmitted to the next higher order. Though shown in the input lines to the bistable elements, the delays could have been placed in the output lines instead.

A shift register can also be formed from dynamic storage elements. A typical arrangement is shown in Figure 4.77. AND and OR gates are

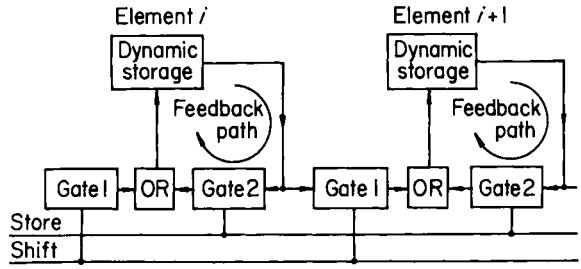


FIG. 4.77. Shift register comprised of dynamic storage elements

inserted into the feedback path of each dynamic storage element as shown. As long as a gate enabling signal is maintained on the store line, the normal feedback path is maintained via gate number 2. To produce a shift, the "store" signal is removed and a signal applied to the "shift" line. The first action interrupts the feedback path via gate number 2, but allows the circulating pulse from stage i to pass through gate number 1 of stage $i + 1$, thereby effecting the shift. After the shift is executed, the "store" signal is reapplied. For correct operation, the shift signal must be held operative for one pulse time, and must be properly phased relative to the clock pulse inputs to the dynamic storage elements.

In the shift registers described in the preceding paragraphs, the bistable elements could have been either vacuum tubes or transistors. When magnetic cores are used as the bistable element, other logical arrangements are possible. The various magnetic core shift registers,

some of which are described next, differ principally in the nature of their transfer loops and in the number of cores required per bit of storage.

The arrangement in Fig. 4.78 requires two cores per bit. One set of

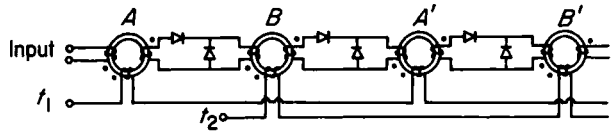


FIG. 4.78. Schematic of a two core per bit shift register with shunt diode transfer loops

cores is used for storage, and the other to provide delays. A complete shifting operation is performed by the successive application of shift current pulses, t_1 and t_2, to the windings shown. Application of the t_1 signal to a core containing a 1 will induce a large voltage in the transfer loop, as a result of which the magnetic state of the core to the right will be changed from 0 to 1. All cores to which the t_1 pulse has been applied are in the 0 state and are ready to accept input signals from the intermediate cores when the t_2 pulse is applied. For serial input operation, new information may be inserted into core A at any time between t_1 pulses. For parallel input operation, the new information is placed in alternate cores at a time when these cores are not otherwise pulsed, and is then shifted out serially by the alternate application of pulses t_1 and t_2. Parallel readout is obtainable by the addition of a separate readout winding. This type of circuit is at present operable at rates up to 250 Kc. A circuit similar to that of Fig. 4.78 may be obtained by the use of single diode transfer loops (described in Section 4.5.2).

Figure 4.79 shows a shift register with three cores for every two bits.

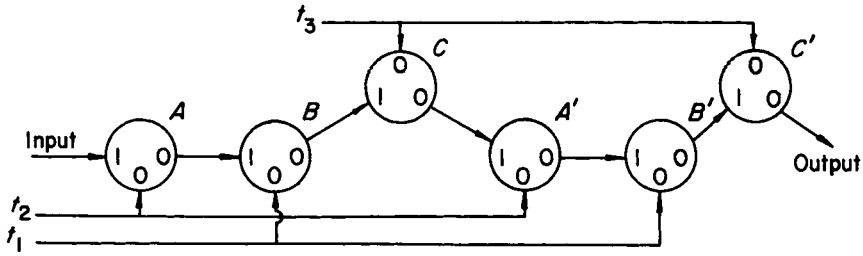


FIG. 4.79. A serial shift register using three cores per two bits and single diode transfer loops

If information is assumed initially to be in cores A and B (and A' and B', etc.), t_1 will transfer one bit from B to C, t_2 will transfer one bit from A to B, and t_3 will transfer one bit from C to A' at the same time that new information is inserted into A. The technique of using $n+1$ cores and $n+1$ advance pulses for every n bits may be extended indefinitely if the economics of the situation warrants it, i.e., if it is desirable to save cores and diodes at the expense of additional drivers.

A parallel shift register comprised of single diodes and split winding transfer loops is represented in Fig. 4.80. Cores A, B, C, . . . constitute

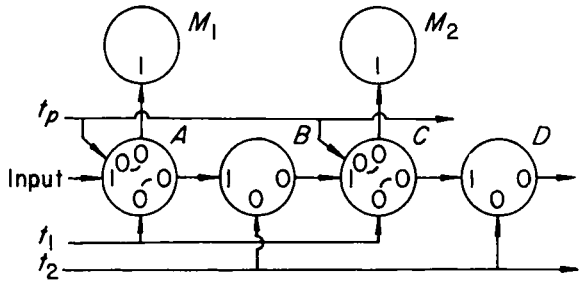


FIG. 4.80. A parallel shift register with single diode and split winding transfer loops

the shift register, and M_1, M_2 . . . are the output cores. Information is inserted serially under control of t_1 and t_2. During the serial shift the split-winding loops between the register cores and the output cores prevent the transfer of information to the output. When the last bit of a word of predetermined length is inserted into A, the preceding bits of that word are in C, E, . . . The application of advance pulse t_p will transfer the information in parallel into the output cores.

A reversible shift register, synthesized by means of split-winding transfer loops, is shown in Fig. 4.81. Information may be shifted from

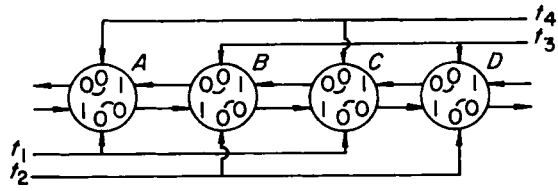


FIG. 4.81. A reversible shift register with split winding transfer loops

left to right by means of pulses t_1 and t_2, or from right to left by pulses

t_3 and t_4. Reversible operation is possible because of the isolating, reversible character of the split-winding transfer loop.

In the shift register shown in Fig. 4.82, temporary storage during the readout operation is obtained not by the use of additional cores but by condensers in *RC* delay networks. As shown, there is a shift winding on each core, and all of these windings are connected in series. The application of the shift pulse saturates all cores in the direction chosen to represent 0. This causes an output to be produced from each core which had been saturated in the direction representing 1. This output charges the condenser, which upon completion of the read-out operation discharges through the succeeding core in such a direction as to record a 1.

The shift register shown in Fig. 4.83 consists of a number of tran-

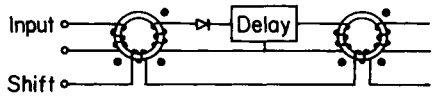


FIG. 4.82. A one core per bit serial shift register

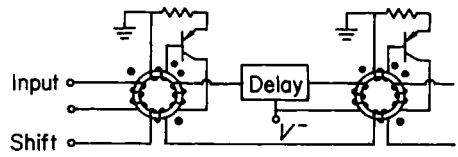


FIG. 4.83. A serial transistor–core shift register

sistor–core combinations in each of which the core provides storage and the transistor serves as a power amplifier. The transistor is normally cut off. When conducting it induces a positive magnetic field. If the core is originally in a saturated state at the bottom of the hysteresis loop, designating storage of a 1, the application of a small current pulse to the shift winding produces a change of flux which in turn induces a negative voltage at the base of the transistor. As a result, there is a flow of current through the collector and its associated winding. This current shifts the state of magnetization of the core still further in the same direction. This regenerative process continues until the core is shifted to the saturated state representing 0. At saturation, the permeability of the core is low so that the gain around the feedback loop consisting of the transistor and the two windings to which it is connected becomes less than unity, and the transistor current is cut off. If the core were originally in the 0 state, application of the shift pulse would have produced no effect since no change in flux would have been produced.

To set any core to the 1 state, current is applied to the input winding which causes a positive voltage at the base of the transistor which drives the transistor into cut-off. The output of each stage is used to reset the core in the following stage. After application of the shift pulse all the

stages which previously stored a 1 change to a 0 and during the regeneration process described each of these stages supplies energy which after a delay sets the succeeding stage to the 1 state.

The transistor–core shift register has three important advantages over core–diode shift registers. First of all, it does not require a shift pulse of predetermined amplitude and width, but only a small trigger pulse which causes each stage to generate its own shift pulse. Second, in core–diode shift registers the shift pulses are of large magnitude (since they supply the total energy) and pass through all the cores. For cores with nonsquare hysteresis loops, some output voltage is produced even by a stage in the 0 state. As a result, the ratio of output for 1 to that for 0, which may be considered a signal to noise ratio, is poor. In the transistor–core shift register, a stage in the 0 state never generates a shift pulse through its core. The third advantage relates to buffering between stages. In the core–diode shift register, an undesirable feedback action can occur because the series diode is so oriented that it passes current induced in the input winding of the second core upon readout, when that core is in the 1 state. This current will tend to set the first core to the 1 state. To alleviate this problem, a number of devices have been used in core–diode shift registers such as having a different number of turns in the input and output windings, a nonsymmetric coupling loop, or a diode shunted across the input winding. However, these devices all lower the efficiency and worsen the operating margins. In the transistor–core shift register perfect buffering is obtained because the collector is cut off during the read-in operation.

4.10. Auxiliary Circuits

In addition to their use in gating circuits and as the basic elements of flip-flops, inverters are also useful for a number of miscellaneous auxiliary functions calling for signal amplification. The vacuum tube cathode follower and its counterpart, the transistor emitter follower, also finds use not only as a switching element but for a number of applications calling for a current amplifier.

One or more members of a group of circuits referred to as trigger circuits may be used in a digital computer system for the purpose of generating discontinuous or impulsive output signals having either a specified periodicity or time relation to input signals. Trigger circuits are classified, according to the number of their absolutely stable states, as follows:

(1) Relaxation oscillator circuits, which continually oscillate between two quasi-stable states. Among the more common relaxation

oscillators (which may be either free running or synchronized) are the multivibrator and free-running blocking oscillator. The multivibrator is not widely used in digital computers. The blocking oscillator (useable from 1Kc to > 1Mc) is useful in clock amplifying circuits. From a clock source, such as a timing track on the surface of a magnetic drum or disk, it generates a synchronized output of impulsive signals with sufficient power to drive a large number of switching circuits.

(2) Single stable-state circuits which, upon being suitably triggered, pass to a quasi-stable state in which they remain for a time (determined by circuit parameters) before spontaneously returning to their stable state. These circuits are used to replace input signals that may be either intermittent or of varying or undesirable shape and amplitude by output pulses of standard shape and amplitude. They are also used to produce an output pulse suitably delayed with respect to the input. Among single stable-state circuits are the delay multivibrator (one-shot delay circuit), and the biased blocking oscillator. The one-shot delay circuit is suitable as a delay unit because the duration of its quasi-stable state is easily controlled over wide limits, and because there are points in the circuit where the potentials are steady except during a transition state. It is useful, too, as a source of rectangular pulses suitable for gating signals. The biased blocking oscillator is used to generate pulses of short rise and fall times and narrow width. It is stable in a quiescent state and brought to a quasi-stable state of short duration by a suitable trigger input. In this state a large oscillation is produced, usually lasting one period.

(3) Two stable-state circuits, which are switched from one stable state to the other by a suitable trigger input signal. Flip-flops are examples of such circuits, and they have been discussed under systems of circuit logic where they find their principal use.

The auxiliary circuits mentioned so far were current and voltage amplifiers used in conjunction with the switching network and special circuits for generating timing signals with adequate power. Another area in which special circuits are required is in the circuitry for gaining access to the various locations in the main store. Because of the large capacity of a main store, it is uneconomical to construct it from active storage elements. As a rule, it is formed instead from less expensive passive elements. A number of special circuits, described in Chapter 5, are required to gain access to a specified location in the main store and to record or sense information.

In addition to the auxiliary circuits that may be used in conjunction with the arithmetic, control, and main storage units, others will generally be required to couple the circuitry of the computer proper to various input and output devices. These circuits are used principally to transform signal levels between the computer and input–output devices. For example, high level current sources, driven by low level circuits in the computer, must be provided to drive output printers and other peripheral equipment.

LITERATURE

ELECTRONIC CIRCUITS (GENERAL)

Chance, B. *et al.* [1949] *Waveforms,* Radiation Laboratory Series, 19, McGraw-Hill, New York.

Hurley, R. B. [1958] *Junction Transistor Electronics,* John Wiley & Sons, New York.

Linvill, J. G. and Gibbons, J. F. [1961] *Transistors and Active Circuits,* McGraw-Hill, New York.

Pullen, K. A., Jr. [1961] *Handbook of Transistor Circuit Design,* Prentice-Hall, Englewood Cliffs, N. J.

Zimmerman, H. J. and Mason, S. J. [1959] *Electronic Circuit Theory,* John Wiley & Sons, New York.

DIODE GATING CIRCUITS

Brown, D. R. and Rochester, N. [1949] Rectifier networks for multiposition switching, *Proc. IRE* **37,** 139-147.

Gluck, S. E., Gray, H. J., Leondes, C. T., and Rubinoff, M. [1953] The design of logical OR-AND-OR pyramids for digital computers, *Proc. IRE,* **41,** 1388-92.

Hussey, L. W. [1953] Semiconductor diode gates *Bell System Tech. Jour.* **32,** 1137-54.

Scobey, J. E., White, W. A., and Salzberg, B. [1956] Fast switching with junction diodes (operating at reverse breakdown), *Proc. IRE,* **44,** 1880-1881.

Yokelson, B. J. and Ulrich, W. [1955] Engineering multistage diode logic circuits, *Trans. Amer. Inst. Elec. Engrs.* **74,** (I) 466-475.

VACUUM TUBE CIRCUITS

Bay, Z. and Grisamore, N. T. [1956] High-speed flip-flops for the millimicrosecond region, *IRE Trans. El. Comp.* **5,** 121-25.

Brown, R. M. [1955] Some notes on logical binary counters, *IRE Trans. El. Comp.* **4,** 67-69.

De Turk, J. E., Garner, H. L., Kaufman, J., Bethel, H. W., and Hock, R. E. [1954] Basic circuitry of the MIDAC and MIDSAC, *Univ. of Michigan Rept.* No. 1942-2-T.

Elbourn, R. D. and Witt, R. P. [1953], [1955] Dynamic circuit techniques used in SEAC and DYSEAC, *Proc. IRE,* **41,** 1380-1387; Computer development at the NBS, *NBS Circular 551,* Part 2, 27-38.

Haueter, R. C., Alexander, S. N., and Greenwald, S. [1953] SEAC, *Proc. IRE,* **41,** 1300-1313.

Johnson, R. F. and Ratz, A. G. [1956] A graphical method for flip-flop design, *Trans. Amer. Inst. Elec. Engrs.* **75,** (I), 52-56.

Meagher, R. E. and Nash, J. P. [1952] The ORDVAC, *Proceedings of the Joint AIEE-IRE Computer Conference* held Dec. 1951, 37-43 (Pub. Feb. 1952)

Paivinen, J. O. and Auerbach, I. L. [1953] Design of triode flip-flops for long term stability, *IRE Trans. El. Comp.*, **2,** 14-26.

Pressman, R. [1953] How to design bistable multivibrators, *Electronics,* **26,** 164-168.

Ritchie, D. K. [1953] The optimum dc design of flip-flops, *Proc. IRE,* **41,** 1614-1617.

Robertson, J. E. [1956] Odd binary asynchronous counters, *IRE Trans. El. Comp.*, **5,** 12-15.

Rosenheim, D. E. and Anderson, A. G. [1957] VHF pulse techniques and logical circuitry, *Proc. IRE,* **45,** 212-219.

Rubinoff, M. [1952] Notes on the design of Eccles-Jordan flip-flops, *Trans. Amer. Inst. Elec. Engrs.*, **71,** (I), 215-220.

Rubinoff, M. [1952] Further data on the design of Eccles-Jordan flip-flops, *Elec. Eng.* **71,** 905-10.

Sherertz, P. C. [1953] Electronic circuits of the NAREC computer, *Proc. IRE,* **41,** 1313-1320.

Zimbel, N. [1954] Packaged logical circuitry for a 4-mc computer, *IRE National Convention Record,* **2,** Part 4, 133-139.

TRANSISTOR CIRCUITS

Angell, J. B. and Keiper, F. P. [1953] Circuit applications of surface-barrier transistors, *Proc. IRE,* **41,** 1709-1712.

Baker, R. H. [1958] High-speed graded base transistor, digital-circuit techniques, *IRE-AIEE Conference on Transistor and Solid State Circuits.*

Baker, R. H. [1957] Boosting transistor switching speed, *Electronics,* **30,** 190-193.

Barnes, R. C. M., Howells, G. A. Cooke-Yarborough, E. H., and Stephen, J. H. [1956] Transistor arithmetic circuits for an inter-leaved-digit computer, *Proc. Inst. Elec. Engrs.*, **103** (B) Suppl. 3, 371-381.

Bashkow, T. R. [1955] D.C. graphical analysis of junction transistor flip-flops, *AIEE Transactions Paper No. 56-24.*

Beter, R. H., Bradley, W. E., Brown, R. B., and Rubinoff, M. [1955] Surface-barrier transistor switching circuits, *IRE National Convention Record,* **3,** Part 4, 139-145.

Beter, R. H., *et al.* [1955] Direct coupled transistor circuits, *Electronics,* **28,** 132-136.

Booth, G. W. and Bothwell, T. P. [1957] Basic logic circuits for computer applications, *Electronics,* **30,** 196-200.

Booth, G. W. [1956] Logic circuits for a transistor digital computer, *IRE Trans. El. Comp.* **5,** 132-138.

Bradley, W. E., *et al.* [1953] The surface-barrier transistor, *Proc. IRE,* **41,** 1702-1720.

Caldwell, S. H. [1959] Transistors in combinational switching circuits, *Proceedings of an International Symposium on the Theory of Switching* (Pt. II), pp. 138-143, Harvard Univ. Press, Cambridge, Mass.

Campbell, C. M. Jr. [1958] New configurations in non-saturating complementary current switching circuits, *IRE-AIEE Conference on Transistor and Solid-State Circuits.*

Carroll, W. N. and Cooper, R. A. [1958] Ten megacycle transistorized pulse circuits for computer application, *IRE-AIEE Conference on Transistor and Solid-State Circuits.*

Carlson, A. W. [1953] High speed transistor flip-flops, *Communications Lab., Air Force Cambridge Research Center Tech. Report 53-16.*

Chao, S. C. [1959] A generalized resistor-transistor logic circuit and some applications, *IRE Trans. El. Comp.*, **EC-8,** 8-12.

Chaplin, G. B. B. [1954] The transistor regenerative amplifier as a computer element, *Proc. Inst. Elec. Engrs.*, **101,** (III), 298-307.

Coblenz, A. and Owens, H. L. [1953] Equivalent transistor circuits and equations, *Electronics,* **26,** 156.

Coblenz, A. and Owens, H. L. [1953] Point-contact transistor operation, *Electronics,* **26,** 158.

Cooke-Yarborough, E. H. [1954] A versatile transistor circuit, *Proc. Inst. Elec. Engrs.*, **101** (III) 281-287.

Domenico, R. J. [1957] Simulation of transistor switching circuits on the IBM 704, *IRE Trans. El. Comp.*, **6,** 242-247.

Ebers, J. J. and Miller, S. L. [1955] Design of alloyed junction germanium transistors for high speed switching, *Bell System Tech. Jour.*, **34,** 761-781.

Ebers, J. J. and Moll, J. L. [1954] Large signal behavior of junction transistors, *Proc. IRE,* **42,** 1761-1772.

Easley, J. W. [1958] Transistor characteristics for direct-coupled transistor logic circuits, *IRE Trans. El. Comp.*, **7,** 7-16.

Felker, J. H. [1955] Performance of TRADIC transistor digital computer, *Proceedings of the Eastern Joint Computer Conference,* 1954, American Institute of Electrical Engineers, New York, 46-49.

Giacoletto, L. J. [1953] Terminology and equations for linear active four-terminal networks including transistors, *RCA Review,* **14,** 28.

Harris, J. R. [1958] Direct-coupled transistor logic circuitry, *IRE Trans. El. Comp.*, **7,** 2-6.

Henle, R. A. [1957] High speed transistor computer circuit design, *Proceedings of the Eastern Joint Computer Conference,* 1956, American Institute of Electrical Engineers, New York, 64-66.

Hunter, L. P. and Fleisher, H. [1952] Graphical analysis of some transistor switching circuits, *IBM Corp., Report No. 26.*

Kudlich, R. A. [1955] A set of transistor circuits for asynchronous, direct-coupled computers, *Proceedings of the Western Joint Computer Conference,* pp. 124-129, March, 1955.

Leichner, G. H. [1957] Designing computer circuits with a computer, *J. ACM,* **4,** 143-147.

Leichner, G. H. and Muerle, J. L. [1957] Computer circuits with 30 millimicrosecond operation times, *Rept. No. 77, Digital Computer Lab., University of Illinois.*

Linvill, J. G. [1955] Non-saturating pulse circuits using two junction transistors, *Proc. IRE,* **43,** 826-834.

Marcovitz, M. W. and Seif, E. [1958] Analytical design of resistor-coupled transistor logical circuits, *IRE Trans. El. Comp.*, **EC-7,** 109-119. (Corrections on p. 324 of same volume).

Moll, J. L. [1955] Junction transistor electronics, *Proc. IRE,* **43,** 1807-1819.

Moll, J. L., Tanenbaum, M., Goldey, J. M., and Holonyak, N. [1956] P-N-P-N transistor switches, *Proc. IRE,* **44,** 1174-1182.

Nelson, J. C. [1956] Speed independent counting circuits, *Rept. No. 71, Digital Computer Lab., University of Illinois.*

Pressman, A. I. [1959] *Design of Transistorized Circuits for Digital Computers,* Rider, New York.

Prom, G. J. and Crosby, R. L. [1956] Junction transistor switching circuits for high-speed digital computer applications, *IRE Trans. El. Comp.,* **5,** 192-196.

Rapp, A. K. and Wong, S. Y. [1956] Transistor flip-flops have high speed, *Electronics,* **29,** 180-181.

Rowe, W. D. and Roger, G. H. [1957] Transistor NOR circuit design, *Trans. Amer. Inst. Elec. Engrs.,* **76,** (I), 263-267.

Shea, R. F. [1957] *Transistor Circuit Engineering,* Wiley, New York.

Shockley, W., Sparks, M., and Teal, G. K. [1951] P-N junction transistors, *Phys. Rev.* **83,** 151.

Simkins, Q. W. [1958] Transistor resistor logic circuit analysis, *IRE-AIEE Conference on Transistor and Solid-State Circuits.*

Simkins, Q. W. and Vogelsong, J. H. [1956] Transistor amplifiers for use in a digital computer, *Proc. IRE,* **44,** 43-55.

Turner, R. J. [1954] Surface-barrier transistor measurements and applications, *Tele-Tech and Electronic Industries,* **13,** 78.

Wallace, R. L. and Pietenpol, W. J. [1951] Some circuit properties and applications of n-p-n transistors, *Bell System Tech. Jour.,* **30,** 530.

Walsh, J. L. [1958] IBM current mode transistor logical circuits, *Proceedings of Western Joint Computer Conference.*

Warnock, J. [1954] Junction transistor switching circuits, *IRE-AIEE Conference on Transistors.*

White, E. A. [1953] Graphical Analysis of Transistor Binaries, *Ballistic Research Labs., Report No. 706.*

Williams, F. C. and Chaplin, G. B. B. [1953] A method of designing transistor trigger circuits, *Proc. Inst. Elec. Engrs.,* **100** (II), 228-248.

Wolfendale, E., Morgan, L. P., and Stephenson, W. L. [1957] The junction transistor as a computing element, *Electronic Engineering,* **29,** 2-7, 83-87, 136-139.

Yourke, H. S. [1957] Millimicrosecond transistor current switching circuits, *IRE-AIEE Conference on Transistor and Solid State Circuits.*

MAGNETIC CORE LOGIC AND CONTROL CIRCUITS

Auerbach, I. L. and Disson, S. B. [1955] Magnetic elements in arithmetic and control circuits, *Electrical Engineering,* **74,** 766-770.

Beyer, R. T., Miller, G. H., Sack, H. S., and Trischka, J. W. [1947] Special magnetic amplifiers and their use in computing circuits, *Proc. IRE,* **35,** 1375-82.

Bonn, T. H. [1959] Analysis of magnetic amplifier circuits, *Proceedings of an International Symposium on the Theory of Switching* (Pt. II), 149-160, Harvard Univ. Press, Cambridge, Mass.

Bozorth, R. M. [1951] *Ferromagnetism,* Van Nostrand, New York.

Crane, H. D. [1959] A high speed logic system using magnetic elements and connecting wire only, *Proc. IRE,* **47,** 63-73.

Devenny, C. F. and Thompson, L. G., Ferromagnetic computer cores, *Tele-tech and Electronic Industries,* **14,** 58-59, 84-94.

E. E. Staff, MIT, *Magnetic Circuits and Transformers,* Wiley, New York.

Guterman, S. S. and Carey, W. M., Jr. [1955] A transistor-magnetic core circuit; A new device applied to digital computing techniques," *IRE National Convention Record,* **3,** Part 4, 84-94.

Guterman, S., Kodis, R. D., and Ruhman, S., Circuits to perform logical and control functions with magnetic cores, *IRE National Convention Record,* **2,** Part 4, 124-32.

Guterman, S. [1955] Logical and control functions performed with magnetic cores, *Proc. IRE,* **43,** 291-98.

Karnaugh, M. [1955] Pulse-switching circuits using magnetic cores, *Proc. IRE,* **43,** 570-584. (Includes a bibliography of 31 items.)

Kodis, R. D., Ruhman, S., and Wood, W. D. [1953] Magnetic shift register using one core per bit, *IRE National Convention Record,* **1,** Part 7, 38-42.

Loev, D., Miehle, W., Paivinen, J., and Wylen, J. [1956] Magnetic core circuits for digital data-processing systems, *Proc. IRE,* **44,** 154-62.

Miehle, W., Paivinen, J., Wylen, J. and Loev, D. [1955] Bimag circuits for digital data-processing systems, *IRE National Convention Record,* **3,** Part 4, 70-83.

Minnick, R. C. [1954] Magnetic switching circuits, *J. Appl. Phys.,* **25,** 479-485.

Newhouse, V. L. and Prywes, N. S. [1956] High speed shift registers using one core per bit, *IRE Trans. El. Comp.,* **5,** 114-120.

Prywes, N. S. [1958] Diodeless magnetic shift registers utilizing transfluxors, *IRE Trans. El. Comp.* **EC-7,** 316-324.

Rajchman, J. A. and Crane, H. D. [1957] Current steering in magnetic circuits, *IRE Trans. El. Comp.* **6,** 21-30.

Rajchman, J. A. and Lo, A. W. [1956] The transfluxor, *Proc. IRE,* **44,** 321-332.

Rajchman, J. A. and Lo, A. W. [1955] The transfluxor—a magnetic gate with stored variable setting, *RCA Review,* **16,** 303-311.

Ramey, R. A. [1953] The single-core magnetic amplifier as a computer element, *Trans. Amer. Inst. Elec. Engrs.,* **72** (I), 342-346.

Rosenfeld, J. L. [1958] Magnetic core pulse switching circuits for standard packages, *IRE Trans. El. Comp.,* **EC-7,** 223-228.

Sands, E. A. [1952] Behavior of rectangular hysteresis loop magnetic materials under current pulse conditions, *Proc. IRE,* **40,** 1246-1250.

Sands, E. A. [1953] An analysis of magnetic shift register operation, *Proc. IRE,* **41,** 993-999.

Scarrott, G. G., Harwood, W. J. and Johnson, K. C. [1956] The design and use of logical devices using saturable magnetic cores, *Proc. Inst. Elec. Engrs.,* **103** (B), Suppl. 2, 302-312.

Van Sant, O. J. [1954] Considerations for the selection of magnetic core materials for digital computer elements, *IRE National Convention Record,* **2,** Part 4, 109-113.

Vorndran, J. W. and Kaiser, H. R. [1955] Magnetic core-transistor logical system for digital computers, *Research Study No. 153,* Hughes Aircraft Co.

Wang, A. Various articles regarding "Static magnetic storage and delay line" contained in the *Harvard Computation Laboratory Progress Report Nos. 2, 3, 4, 5, 6, 8, 10,* Investigations for the Design of Digital Calculating Machinery, 1948, 1949, 1950.

Wang, A. [1951] Magnetic delay line storage, *Proc. IRE,* **39,** 401-07.

Wang, A. and Woo, W. D. [1950] Static magnetic storage and delay line, *J. Appl. Phys.,* **21,** 49-54.

Wylen, J. [1953] Pulse response characteristics of rectangular hysteresis loop ferromagnetic materials, *Trans. Amer. Inst. Elec. Engrs.,* **72** (I), 648-55.

SUPERCONDUCTIVE CIRCUITS

Buck, D. A. [1956] The Cryotron—A superconductive computer component, *Proc. IRE*, **44**, 482-493.

Proceedings of Madison Conference on Low Temperature Physics, [1958] Univ. of Wisconsin Press, Madison, Wisconsin.

Shoenberg, D. [1952] *Superconductivity*, 2nd Ed., Cambridge Univ. Press.

MICROWAVE CIRCUITS

Blattner, D. J. and Sterzer, F. [1959] Fast microwave logic circuits, *IRE National Convention Record*, **7**, Part 4, 252-258.

Lewis, W. D. [1959] Microwave logic, *Proceedings of an International Symposium on The Theory of Switching (Pt. II)*, 334-342, Harvard Univ. Press, Cambridge, Mass.

Microwave techniques for computing systems (a collection of articles on logic and circuitry for microwave computers) [1959] *IRE Trans. El. Comp.*, **EC-8**, 262-307.

von Neumann, J. [1957] Nonlinear capacitance or inductance switching, amplifying, and memory organs, U.S. Patent No. 2,815,488.

Sterzer, F. [1959] Microwave parametric subharmonic oscillator for digital computing, *Proc. IRE*, **47**, 1317-1324.

TUNNEL DIODE CIRCUITS

Akers, S. B. and Stabler, E. P. [1962] Logical design and implementation in a pumped tunnel diode-transistor logic system, *Gigacycle Computing Systems, AIEE Special Publication S-136*, 18-31.

Chow, W. F. [1960] Tunnel diode digital circuitry, *Digest of Technical Papers of IRE-AIEE International Solid State Circuits Conference*, Univ. of Pa., 32-33.

Esaki, L. [1958] New phenomenon in narrow germanium p-n junctions, *Physical Review*, **109**, 603-604.

Lesk, I. A., Holonyak, N., Jr., and Davidsohn, U. S. [1959] The tunnel diode—circuits and applications, *Electronics*, **32**, 60-64.

Lewin, M. H., Samusenko, A. G. and Lo, A. W. [1960] The tunnel diode as a logic element, *Digest of Technical Papers of IRE-AIEE International Solid State Circuits Conference*, Univ. of Pa., 10-11.

Neff, G. W., Butler, S. A. and Critchlow, D. L. [1960] Esaki (tunnel) diode logic circuits, *Digest of Technical Papers of IRE-AIEE International Solid State Circuits Conference*, Univ. of Pa., 16-17.

Peil, W. and Marolf, R. [1962] Computer circuitry for 500 Mc, *Digest of Technical Papers of IRE-AIEE International Solid State Circuits Conference*, Univ. of Pa.

Rhoades, W. T. [1962] Piecewise-linear switching analysis of a bistable tunnel diode logic circuit, *Gigacycle Computing Systems, AIEE Special Publication S-136*, 33-52.

Sims, R. C., Beck, E. R. and Kamm, V. C. [1961] A survey of tunnel diode digital techniques, *Proc. IRE*, **49**, 136-146.

5. Large Capacity Storage Systems

5.1. Introduction

This chapter provides an introduction to the subject of large capacity storage systems. We shall loosely define "large capacity" to mean anywhere from several thousand to several million bits. A storage system includes not only the storage medium but also the means for gaining access to specific locations in the store and for the recording and reading of information.

Large capacity storage systems for digital computers may be used for either internal or external storage functions. An internal store is used to hold the program of instructions to be executed and also provides space for the storage of intermediate and final results. An external store is used for the preparation of a program and auxiliary data in a form suitable for subsequent entry into a computer. External storage media are used also for the maintenance of large files of programs and other input data as well as for the storage of output data. Whether a specific type of storage system is better suited for internal or external storage or can be used for either will depend on a number of criteria. These will be considered in the sections following.

Criteria important in the evaluation of large capacity storage systems are: cost per bit, reliability, maintainability, physical size, power consumption, etc. These will be considered in the descriptions, later in this chapter, of specific storage systems. At this point we will comment, in a general way, on four important distinguishing characteristics of storage systems: namely, operating speed, volatility, erasability, and access time.

Operating speed refers to the rate at which information is transferred into or out of the storage system. Once access has been gained to a desired location in the store, the rate at which information is read out will depend on the nature of the store. The operating speed of an internal storage system need only be great enough to make the delay in recording into and reading out of storage compatible with the time required to execute arithmetic and logical operations. Since a higher operating speed increases the cost of a system, some compromise is usually reached between speed and cost. However, there are special situations where a high price may be paid for a small increase in speed.

Volatility, in reference to a storage medium is used to indicate whether power must be continually or periodically supplied to retain information previously stored. A volatile storage medium requires the application of power while a nonvolatile one does not. Examples of nonvolatile storage media are punched cards, punched tape, magnetic tape, the magnetic surface of a revolving drum or disk. Examples of volatile storage are the electrical charge on the surface of a cathode-ray storage tube, and the pulses recirculated in an acoustic delay line. If power were not applied to restore the charges on a cathode-ray tube, they would gradually leak off. The pulses recirculated through an acoustic delay line become attenuated and distorted in shape and therefore power must be applied to amplify and reshape these pulses. Thus, volatile storage systems, whether of the static or dynamic type, require that the stored information be periodically regenerated. The required frequency of the regeneration cycle for a particular system depends on the rate at which information degenerates in that system.

Erasability is another important characteristic of a storage medium. It is essential for an internal storage system but not necessarily for an external one. Storage in the form of punched paper cards or tape or photographic storage is not erasable while electrical phenomena such as magnetic dipoles, electrical charges, and voltage pulses are.

Access time, i.e., the time required to gain access to an item in storage, is one of the most important figures of merit of a large capacity storage system, because it limits the over-all speed of computation within a computer. The access time of a particular storage system will depend on the means employed to select items from storage. Primarily, the storage selection scheme depends on whether information is stored in a spatial or temporal pattern or in a combination of both. Theoretically, there is no basic difference since the location of stored information, like any other location, may be determined by specifying coordinates (in space and/or time) referred to a specific frame of reference. We will consider next the nature of both "time domain" and "space domain" storage.

Space domain or static storage refers to storage systems in which each storage element permanently occupies a specified physical location, and there is equal accessibility, at all times, to any of the elements in the store. Examples of static storage elements are relays, flip-flop circuits, cathode-ray tubes, magnetic cores. Nonvolatility, a low cost per bit plus a fast switching time make magnetic cores especially suitable for the main store of a high speed computer. Access may be gained to a particular one or group of storage elements by means of a switching network referred to as a selection network. This network selects a particular storage element, or group of elements, when it receives the coordinates assigned to the

particular element in the system. There is no problem of resolution or drift in the locating system as in time domain storage systems. The logical nature of such a selection network will now be described. Assume, first of all, that there is a transducer, T_i, associated with the address of each storage location in the memory. Whenever it is required to read out of the memory, it is only necessary to cause the output of a specified transducer to be read. Further, an arbitrary address, i.e., a set of coordinates, may be assigned to each item in the store. Then, to select a particular item from the store, it is only necessary to insert the address of the required item into a register. If the store has n addresses, then the register must have at least p binary places, where p is an integer greater than or equal to $\log_2 n$. The contents of this register can be used to control the selection of any address in the store by means of a many-to-one switching network. The Boolean expression for the output of this network is of the form

$$f = A_1T_1 + A_2T_2 + \ldots + A_pT_p \tag{5-1}$$

where each A_i represents a particular one of the 2^p states of the address register. It is apparent from Eq. (5-1) that when the address register holds the address, A_i, i.e., when A_i is true, the output of the network is simply T_i, the output of the specified transducer. It is also evident that the complexity of the switching network depends on the number of storage elements in the system.

To provide for the entry of information into any of the n locations of the store, n signals of the form

$$T_i = A_iI \tag{5-2}$$

must be formed, where I represents the information to be recorded. Since, in the basic general purpose type of computer, access is provided to only one store address at a time, only the transducer corresponding to the A_i which is currently true will receive the signal to be recorded.

If the switching network* is formed from diode gating circuits, appreciable savings may be realized by the use of pyramiding and minimum networks (described in Chapter 4).

In static stores containing elements whose states must be periodically regenerated (like the charged spots on cathode-ray tubes or the magnetic states of the cores in certain core storage systems) time is consumed not only in acquiring access to a particular item in storage, but also in

---

* In magnetic core storage systems, the selection problem may be simplified by the use of magnetic core selection switches. See Section 5.3.2.4 and references under Magnetic Core Storage at the end of this chapter.

regenerating information that otherwise would be lost by gradual deterioration and/or the process of interrogating the state of the elements.

Time domain or dynamic storage refers to storage systems in which there is access to a particular storage location only at specified times. The stored items pass a stationary transducer or set of transducers sequentially. To select a particular item, for reading or recording purposes, the time at which it will be accessible to one or more transducers must be specified. The access time to a particular item increases with the number of items (all other factors remaining constant) since more items must pass a transducer before a particular item is reached. An increase in the number of items does not, however, necessarily increase the complexity of the selection switching network. In a dynamic storage system the average time required to extract an item from the store is important. This average access time is simply one-half the maximum access time.

There are two principal types of dynamic storage systems. In one, information is stored by means of a carrier propagated and recirculated through a stationary delay medium, e.g., an acoustic wave propagated through a path of mercury (see Section 5.6). In the other, information is stored on a recording surface which is rotated to provide each stationary transducer with access to storage locations along a track; this type of store allowing two modes of operation—one in which storage locations (sectors) are assigned specific addresses, as in a static store, and another in which information is dynamically stored in a delay line formed by placement of a record and a read transducer along a single track. In the first type of delay line, the total delay between successive appearances of the same item of information is determined by the velocity of propagation of the signals through the medium and the length of the path from the input to the output transducer; in the latter type by the angular velocity of the rotating medium and the angular separation of the input and output transducers. Access is gained to a specific item, for either recording or reading, by energizing a gate to an input or output transducer, respectively, at the proper time.

Dynamic storage systems may further be classified as synchronous or asynchronous, depending on whether the relative speed of the stored data with respect to the transducer stations is constant or not. Magnetic drum and disk systems are considered synchronous, even though the angular rotation rates are not constant, because the stored data is referenced with respect to a clock recorded on one of the tracks (see Chapter 7). Within certain temperature limits ultrasonic delay lines provide a delay which is practically constant. Magnetostrictive delay lines provide an essentially constant delay over a considerably wider range of temperature. A magnetic tape storage system is considered asynchronous because of the relatively large fluctuations in speed of the moving tape.

Synchronous and asynchronous storage systems each call for a different type of selection scheme. Two fundamental ways of locating an item in a synchronous store are as follows: In one scheme, a marker pulse received from the storage system (or some other specified "start" signal) is used to open a gate so that clock pulses can be entered into a so-called pulse position counter. After a count has been reached equal to the number of bits in a word of storage, the counter is reset and simultaneously causes a pulse to be entered in a word counter. At any given time, the contents of the two counters indicate the word as well as the particular bit position in that word that is currently accessible. Therefore, to perform an operation on the contents of any particular storage location, it is only necessary to enter the address of that location in a register, and to provide logical circuitry that detects a coincidence between the contents of the register and the current contents of the word counter. In the second scheme, tags indicating the address of each word in the store are also stored, usually in a separate address line. Logical circuitry is provided to detect a coincidence between the address currently being read from the address line and that placed in the address register. This scheme of selection is particularly useful in magnetic drum or disk storage systems where the nonvolatility of recorded data allows the contents of an address line to be permanently recorded. Regardless of which scheme is used, it is usually desirable to search for and indicate the address of the succeeding word rather than the current one, in order to allow certain preparatory operations to be performed. In the first scheme this is accomplished by placing in the address register a number one less than the address actually sought. In the second scheme, the address tags are so placed that the tag read from the address line during any given word time is the address of the next word to be available from the main store.

The most convenient way of selecting an item in a magnetic tape store is by searching for a tag or address associated with each unit or block of data. The size of this block is based on the characteristics of the tape transport control mechanism (see Section 7.5.6 and A.1.3).

In a dynamic storage system both a static and a dynamic selection network may be desirable. As an example, consider a magnetic drum or disk memory with a single transducer associated with each channel. Here a static selection scheme is used to connect a particular read/record head to a common amplifier circuit, and a dynamic selection scheme to define the time interval during which the connection is made. Head selection networks are described further in Sections 5.2.6 and 7.6.3.

In the sections following, there are descriptions of various types of storage systems, each based on the exploitation of a particular physical

property or properties of specific materials. Most of the space is given to storage media based on magnetic phenomena because they now dominate the field. Practically all high speed memories are being built from ferrite cores (and to a much lesser extent, from multi-aperture devices). For medium speed, the magnetic drum (or disk) memory is dominant. The most versatile, and widely used combination input-output and auxiliary storage medium is magnetic tape. Promising new high speed memories magnetic in nature are superconductive devices (for about 1 to possibly 100 million bit capacities) and thin film devices (for capacities up to perhaps a million bits). For smaller capacities (about 10,000 to 100,000 bits) but very high speeds the tunnel-diode store is promising. The apertured ferrite plate and twistor are finding limited use, and their future is doubtful. The cathode ray tube memory is described because it was the first important high-speed memory and is still operational in many machines. The mercury delay line is included principally for historical reasons. Other types of delay lines and the diode-capacitor memory are included because of their usefulness in special though limited applications. The ferroelectric memory is included simply as a matter of general interest even though its development has not been completed and its potential advantages overshadowed by other memories developed subsequently.

5.2. Dynamic Magnetic Storage

In a dynamic type of magnetic storage system, information is recorded by means of a transducer which induces magnetic dipoles in a moving magnetic surface. Sensing of the recorded dipoles is facilitated by relative motion between the magnetic surface and a read transducer. (In a static magnetic storage system no mechanical motion is involved, recording and sensing each being accomplished by applying an electromagnetic force to separate, magnetically alterable elements.

Recording of binary data on a magnetic surface is based on the same magnetic phenomenon, namely magnetic hysteresis, used for the recording of data in magnetic cores. In Fig. 4.48 the hysteresis loop of a typical magnetic recording medium is shown. If a positive magnetizing force is applied of sufficient amplitude to bring the medium to the point B_m, then even after the magnetizing force has been completely removed, a residual flux density, $+B_r$, will remain. A similar set of events occurs after the application and removal of a magnetizing force of opposite polarity. If the applied magnetomotive forces are great enough to cause saturation of a cell on the magnetic surface, the residual fluxes B_r and $-B_r$ are practically independent of the previous condition of the magnetic surface. These residual fluxes may be used to indicate the recording of a 1 or 0.

5.2.1. The Recording Transducer

The transducer used for recording or sensing information on a magnetic surface is referred to as a magnetic head. Conventional head designs consist of a core of magnetic material around which are wound several turns of wire. In earlier head designs, the cores were usually formed from laminations of a metal like Permalloy or mu-metal. Heads, as shown in Fig. 5.1, are now usually formed from two ferrite pieces by joining them in such a way that a usable gap is formed, as shown in the figure. Wires can be conveniently wrapped around one or both of the pieces before they are joined. The head on the right in Fig. 5.1 simplifies the wire wrapping

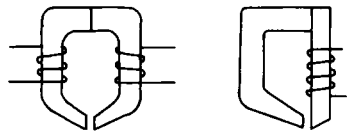


Fig. 5.1. Magnetic head designs

operation since it is done on the straight I section. It also allows closer spacing between heads, an important consideration in forming a short delay line. Ferrites are used because of their low eddy current losses and high permeabilities in the megacycle region. If a core is of high permeability, the flux set up by passing a current through the coil will largely be confined to the core material. The flux will fringe around the minute gap (a typical value being 0.0005 in.), and if the gap is placed sufficiently close to a magnetic surface, the fringing flux penetrates this surface in completing its path from one side of the gap to the other (see Fig. 5.2).

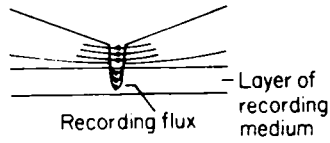


Fig. 5.2. Recording flux pattern for recording on a magnetizable surface

If the magnetic field is sufficiently strong and the gap is sufficiently close to the magnetic surface, a small but usable magnetic dipole will be induced in the surface in the region of the gap. The polarity of this dipole is determined by the direction of current flow in the head winding. The magnetic head is placed as close to the surface as mechanical considerations

will allow, because of the rapid attenuation of the flux density away from the gap. The nature of this attenuation is shown in Fig. 5.3.

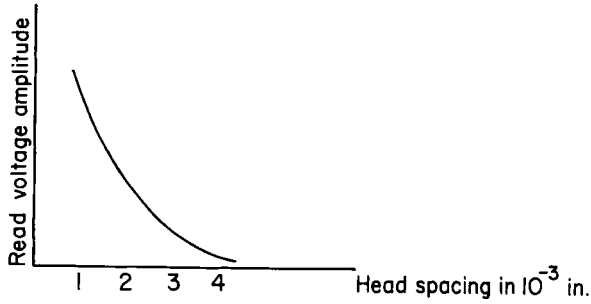


FIG. 5.3. Read voltage variation with head spacing

Often, a metallic nonmagnetic shim is placed in the gap in order to generate (from eddy currents) mmf's that oppose the main flux. As a result more flux is forced out of the gap. Within limits, the leakage flux may also be increased by a greater gap length with only slight degradation of maximum recording density obtainable if flux can be reversed rapidly relative to the speed of the medium.

Recording current may be reduced by an increased number of turns (although a limit is imposed by frequency response requirements, the resonant frequency being lowered by more turns), and by using a head whose magnetic reluctance $\mathcal{R}$ is less. Since $\mathcal{R} = l/\mu A$, where l is the length of the mean path of the flux, A the cross sectional area, and μ the permeability of the material through which the flux passes, a longer gap length (where μ is low) means a larger value of $\mathcal{R}$.

A convenient way to apply recording current in either of two directions is by means of a center tapped winding. Current applied to one half of the winding produces a flux opposite in sense to that produced by current applied to the other half. In a head used for reading only, the gap length should be small to reduce the magnetic reluctance (thereby increasing flux through the head) and to permit high density recording. A head with a single winding may be used for both recording and reading, the winding being switched to a record or read amplifier as needed. Usually, a combination record-read head has separate windings for these functions; also, its gap length is determined by the reading requirement. To prevent large signal pick-up from a record winding from entering a read amplifier, there should be adequate separation and/or shielding between record and read windings (on the same or different heads).

5.2.2. The Reading Process

It has been shown how binary data may be recorded by application of a recording current of sufficient amplitude to saturate a magnetic surface in either of two polarities. The polarity of the recorded dipole must subsequently be sensed by the reading process. Usually this is done by some type of phase detection system. In these systems the reading process is dependent on relative motion between the head and the residual flux patterns from the surface. (Static sensing schemes, referenced in the literature cited at the end of this chapter, are not, as a rule, suitable for digital computer applications). This causes a magnetomotive force to be induced in the head winding proportional to the rate of change of the flux, resulting in a small but useable voltage signal. Figure 5.4 shows

FIG. 5.4. Read voltage waveforms from induced positive (a) and negative poles (b)

the nature of these read voltage signals. The extent of the period in which the flux entering the read head is at a maximum value determines the width of the interval within the dashed lines. The recording waveform is usually such that this interval is small compared to the width of the read waveform. The sensing process is classified as nondestructive because the recorded information is not altered by it. Information recorded on a magnetic surface is considered nonvolatile because it is retained without periodic regeneration, and even when power is removed from the system (provided transient signals produced by the removal do not generate appreciable currents in the heads).

5.2.3. Efficiency of Storage

When a magnetic surface is used for data storage, it is usually desirable to be able to record as many bits per unit of area as possible. One may consider the question of the density of stored information obtainable, referred to as packing density, in terms of a linear and transverse recording density. The linear recording density is the number of pulses recorded per linear inch in any channel, and the transverse density is the number of channels recorded per inch in a direction perpendicular to the channels.

The linear recording density is a function of the width of the gap between the pole faces of the head's magnetic core. When the wavelength

λ of the recorded signal is equal to the gap length g there is a zero output. The gap length g is optimum when it is equal to $n\lambda/2$ (where n is an integer), the output signal being attenuated for gap lengths on either side of these points (see Section 5.2.4.). Transverse recording density is limited primarily by the compactness of the heads and the tolerance to which they can be positioned.

5.2.4. The Memory Transfer Function

The memory transfer function of a magnetic recording system is defined as the ratio of output voltage from the read head to input current to the record head. If the recording current were held constant, the memory transfer function would exhibit a 6 db/octave rise with frequency for a head whose output was proportional to the rate of change of flux, provided no other phenomena entered. Experimental results show that this is approximately the case at relatively low frequencies (up to about a few kilocycles). However, for higher frequencies the response falls away from the 6 db/octave rise and eventually decreases with frequency. This deviation, considered as a loss, is a result of the following contributing factors: spacing loss, thickness loss, gap loss, head loss, and self-demagnetization (see Wallace [1951]; also Hong [1958], Miyata and Hartel [1959], Hoagland and Bacon [1960] and Carne [1961]).

Spacing loss, resulting from separation of the head and recording medium has been shown experimentally to be approximately equal to 55 d/λ db, where d is the spacing between head and magnetic surface and λ the recorded wavelength (λ being a function of the recording frequency and speed of the medium). Factors which may cause the spacing to be increased erratically are: (1) foreign matter on or defects in the recording surface, (2) accumulation of static charge on the recording surface, (3) erratic movement of the recording surface away from the head. In magnetic drum recording (Section 5.2.6) the latter phenomenon, referred to as runout, results from any eccentricity in the path described by the moving surface. In magnetic disk recording (Section 5.2.7) it is much less pronounced, and may result from flexing of the disk.

The thickness loss (in db) has been estimated to be $20 \log_{10}[(2\pi\delta/\lambda) / (1\text{-exp } (-2\pi\delta/\lambda))]$*db, where δ is the thickness of the recording medium.

---

* This expression is based on an assumption of uniform magnetization throughout the thickness of the medium. If the thickness is large compared to the gap length, the recording field does not penetrate uniformly through the medium and the expression is not valid.

At low frequencies, for which $\lambda >> \delta$, the read voltage is proportional to δ and to the frequency. At higher frequencies, for which $\lambda << \delta$, the response is independent of the thickness since the limited penetration into the medium makes its thickness unimportant.

The nature of the gap loss can be visualized by noting that if the gap length is approximately equal to one wavelength, there is essentially no field produced to magnetize the medium. The gap loss has been estimated to be $20 \log_{10} [(\pi g/\lambda)/\sin (\pi g/\lambda)]$ db, where g is the effective gap (which is generally smaller than the actual gap in the head).

Eddy current and other losses within the heads are in accordance with the established behavior of magnetic circuits and will not be considered explicitly.

The phenomenon of self demagnetization appears in magnetic recording as a result of the fact that adjacent small magnetic dipoles exert a torque upon each other upon leaving the magnetic field of the record head, the torques tending to return the dipoles to a random state. The retentivity of the magnetic medium tends to hold them in the magnetized state, but experiments show that self-demagnetization increases rapidly once very short wavelengths are encountered. Figures of merit such as the ratio H/B are generally used to assess the effect of self demagnetization on a particular storage medium.

5.2.5. Magnetic Recording Coding Techniques

A number of techniques have been developed for translating a sequence of binary signals, in the form of a sequence of voltage or current levels or pulses, to a sequence of recording current signals suitable to actuate a recording head. The particular recording technique employed determines the procedures best suited for correctly interpreting the recorded flux patterns. This will be brought out in the ensuing parts of this section.

Each magnetic recording coding technique falls into one of two broad categories. In one, referred to as the "return to zero," or RZ system, the recording waveform is always returned to zero amplitude before generation of the waveform for the next bit position. In the non-return-to-zero or NRZ system the recording waveform is not returned and held at zero amplitude after each bit.

Techniques for erasing stored data also fall into two categories. In the ac method, the medium is returned to a nonmagnetized state by the application of high frequency ac signals to an erase head winding. In the dc method, the medium is saturated in one direction corresponding to that defined to represent 0 by the application of a large amplitude dc signal, or by a permanent magnet placed close to the recording surface.

5.2.5.1. RZ Recording

The return to zero or RZ method of recording can assume any one of a number of alternate forms. In the three-level form, a pulse of current is applied to the magnetic head in either one of two directions, according to whether a 1 or a 0 is to be recorded. For each bit recorded, the medium is saturated in one direction or the other. The recording current is always returned to zero between the recording of individual bits.

In an early form of two level RZ recording, a separate erase head is used to saturate the medium in a direction defined to represent 0, and a pulse is applied to the record head only when the medium is to be saturated in a direction representing 1. A head arrangement for this scheme of recording is shown in Fig. 5.5. After passing the read head, each cell

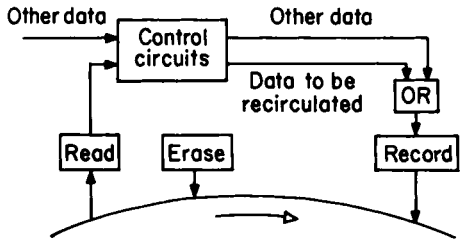


FIG. 5.5. A head arrangement for two-level *RZ* recording

is returned to the 0 state. This allows either a 1 or a 0 to be recorded in any cell, by the application or nonapplication of a current pulse, respectively, regardless of the state of the cell before it passed the erase head. Whenever it is desired to recirculate information already recorded, the output of the read head is coupled through appropriate switching circuitry, to the current amplifier that drives the record head.

The three level system has the feature that a distinct signal is produced within each cell (thereby allowing the absence of a read signal from a cell to be used as a definite indication of a malfunction). Because of this characteristic, it does not require a dc amplifier in the read circuit (which is required in a two level system for response to a series of 0's that may occur in the recorded information pattern).

The graph in Fig. 5.6(a) relates, for a particular sequence of bits, the recorded flux density, ϕ, that would pass through a reading head as a function of time. Figure 5.6(b) shows the output voltage of the head, which is proportional to the derivative of this flux. The patterns in Figs. 5.6(a) and (b) assume relatively small recording densities, i.e., a rela-

tively large spacing between recorded bits. Beyond a certain recording density, the results of interaction between adjacent recorded cells becomes noticeable as shown in Figs. 5.6(c) and (d). Note that the flux density returns to zero between adjacent cells only when there is a transition from a 1 to a 0 or vice versa, so the output voltage waveforms depend not only

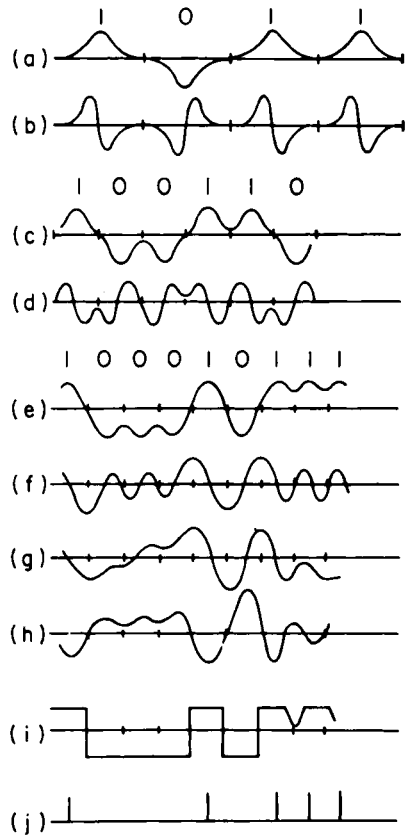


FIG. 5.6. Read waveforms in *RZ* recording for various recording densities, p_i

on the recording in individual cells, but on the particular sequence recorded. When the recording density is increased still further, the effects of interaction become more pronounced as shown in Figs. 5.6(e), (f), and (g). Though the various output voltage waveforms in Figs. 5.6(b), (d), (e), (f), and (g) appear markedly different, still they all possess one characteristic, by means of which a 1 can be distinguished from a

0. For a 1 the output signal is going negative in the second half of a cell, for a 0 it is going positive. Therefore, if the output voltage waveform is differentiated, a signal will be produced which will be negative in the center of each cell where a 1 is recorded, and positive where a 0 is recorded. Differentiation of the signal in Fig. 5.6(g) is shown in Fig. 5.6(h). After amplification, inversion, and clipping, the output signal would appear as shown in Fig. 5.6(i). If it is then applied to a coincidence gate with clock pulses timed to occur at the center of each cell, the final result would be as shown in Fig. 5.6(j).

A disadvantage of the differentiating technique is that it attenuates the read signal. As a result, additional amplification, which also amplifies pulses due to noise sources, must be introduced.

If the recording density is very high and a long sequence of either 1's or 0's is recorded, the small ripple signal resulting from the RZ nature of the recording, may not be adequate to produce a satisfactory output signal. Schemes for correctly interpreting the read waveform at high recording densities are generally complex and critical in operation.

5.2.5.2. NRZ Recording

The non return to zero, or NRZ method of recording is somewhat similar to the two-level form of RZ recording using dc erasing. However, there are two principal differences. First of all, recording current is applied for the recording of both 0's and 1's. Secondly, the recording current is applied for the full width of a cell so that there is no return to a reference zero state of saturation between cells. As a result, the medium is continuously magnetized to saturation in either of two directions, and reversals of direction occur only when there is a transition from a 1 to a 0 or vice versa.

The nature of the record current and the read voltage are indicated in Figs. 5.7(a) and (b). It is evident that in reading, a positive pulse indicates a transition from 0 to 1 and a negative pulse a transition from 1 to 0. A precise indication of the end of individual bit positions can be provided by pulses from a separate clock channel. The original waveform may be obtained from the read waveform by causing the positive pulses to trigger a read flip-flop to an "on" state, and the negative pulses to reset it, generating the waveform shown in Fig. 5.7(c).

Comparison of Fig. 5.7(b) with Fig. 5.6(b), and consideration of the nature of the record current waveform, shows that the NRZ system theoretically allows an information rate twice that obtainable with RZ recording. A limiting factor in both cases is the distance along the recording surface within which a transition can be made between states of

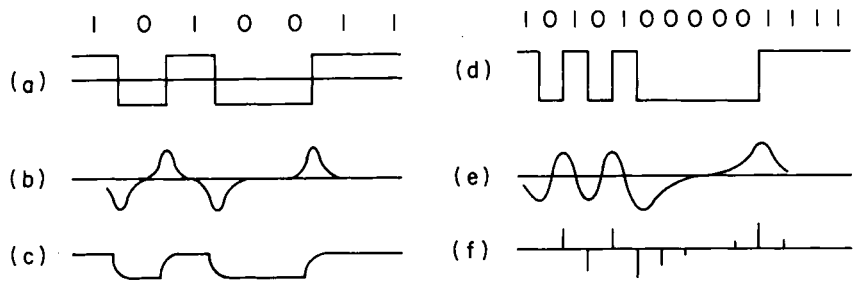


FIG. 5.7. Record and read waveforms in *NRZ* recording

positive and negative saturation. In RZ recording the frequency of flux reversals is independent of the information pattern, while in NRZ recording it is a maximum for a sequence of alternating 1's and 0's. However, even in this case NRZ recording calls for only half as many flux reversals and, therefore, only half the pulse repetition rate. The NRZ system has a greater duty cycle than the RZ system, since currents must flow through the recording heads in one direction or the other continuously. However, a compensating factor results from the fact that the actual pulse frequency at the recording head is reduced by one-half, compared to the RZ system. This allows a greater number of turns of wire to be used in the head, and thereby reduces the amplitude of the driving current that must be supplied to it to produce a specified flux density.

Referring to Fig. 5.7(b) one sees that there is a separation between the trailing edge of a positive pulse and the leading edge of the negative pulse. As the recording density is increased, a point will be reached at which the shoulder separating the two disappears. Beyond this point, the read voltage amplitude diminishes rapidly due to demagnetization effects in the recording medium as a result of interference between dipoles in adjacent cells. Of course, RZ recording is also limited (and at a lower information rate) by demagnetization effects.

Let us here delineate the principal advantages and limitations of both the RZ and NRZ recording techniques. First, RZ recording is subject to noise that appears because old information is not erased in the interval between adjacent pulses; while transformer coupling cannot be used in NRZ recording because current does always flow through the head winding. At low bit densities RZ recording allows use of a narrow band-pass read amplifier, while NRZ recording requires a wide band amplifier because of the low frequencies represented by uninterrupted streams of 0's or 1's and the high frequency presented by alternate 1's

and 0's. At higher bit densities, because of flux spreading and self-demagnetization a wide band amplifier is also required with RZ recording. Such an amplifier has a worse signal-to-noise ratio than a narrow band amplifier. If a wide band amplifier is not used, the read amplifier's output will vary appreciably with the frequencies presented by the information pattern, making it difficult to set a reliable threshold for signal discrimination. At high densities, an uninterrupted string of 1's or 0's also tends to cause flux saturation in a head, and, as a result, near zero values for induced voltage swings.

A so-called return-to-bias system is like RZ recording in that the direction of current is reversed between write 1 pulses (being returned not to zero amplitude, as in RZ recording, but to an opposite polarity that saturates the medium in a direction defined to represent 0) and like NRZ recording in that current is never returned and held at zero amplitude. Because uninterrupted streams of 0's are possible, it too requires a dc read amplifier. Many of the difficulties described here may be alleviated by phase modulation recording, described in Section 5.2.5.3.

5.2.5.3. Phase Modulation Recording

The distinguishing characteristic of the phase modulation system of recording is that two current signals, of equal duration and opposite polarity, are used for the recording of each bit. These signals may be either the RZ or the NRZ type, i.e., either current pulses or states may be used. Consideration of Figs. 5.8(a) and (b) shows that at higher

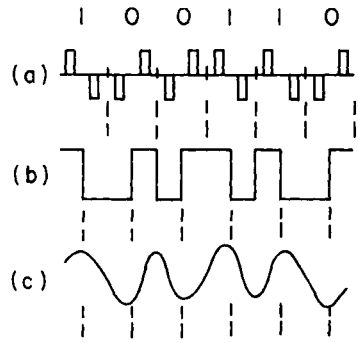


FIG. 5.8. Record and read waveforms in Ferranti phase modulation recording

recording densities there will be little difference between the RZ and NRZ current waveforms, although the duty cycle of the NRZ system is slightly higher. On the other hand, the NRZ system is slightly superior in writing over previously recorded information. (The upper dashed lines define the boundaries of cells and the lower ones define the centers.)

By the use of two polarities per bit one is assured that regardless of the information pattern there will be at least one flux reversal over the interval of two adjacent cells. This scheme also restricts the bandwidth requirements to the octave between the information rate and twice this frequency. Because of this narrow band pass, a higher signal-to-noise ratio is obtainable from the read amplifier. The output voltage waveform, in Fig. 5.8(c), shows either a positive or negative peak near the center of each cell, according to whether a 1 or 0 was recorded, this information being derived from the direction of the zero crossover. To recover the recorded information this waveform is sampled by pulses timed to occur at the center of each cell. An important feature of the phase modulation system is that the timing pulses may be derived from the significant zero crossings of the waveform itself. (Various schemes may be used to reject the nonsignificant zero-crossing that occurs between cells of like content).

Because this type of recording is phase (rather than amplitude) sensitive, the threshold level setting and signal interpretation problem is avoided. Even small amplitude signals will be detected as long as the noise pulse is significantly less than the signal. With amplitude sensitive systems an error can result from either a weak signal or large noise pulse alone. In phase modulation recording the signal-to-noise ratio must be very low for the zero crossing point to be shifted enough to result in misinterpretation of the recorded waveform.

As indicated in Fig. 5.8(c), there is an increase in peak amplitude in going between two cells not holding like data (i.e., from 0 to 1 or 1 to 0). By using a read system in which a flip-flop is triggered only in the event of a change signal, the demagnetization effects in the recording medium may be made negligible.

Variation of read voltage amplitude with recording density for phase modulation recording on a disk surface with three types of magnetic coating is shown in Fig. 5.9. (Plating thicknesses are a few tens of microinches, the oxide thickness in the 200 to 500 microinch range, the head characteristics as follows: a ¼ mil read-write gap, a Hi-Mu 80 laminated core structure with a 40 turn winding, inductance of about 25 microhenries). An extremely thin coating results in less self-demagnetization (which, for short wavelengths decreases sharply as thickness is reduced), greater resolution and less head trailing effect (i.e., demagnetization of the medium

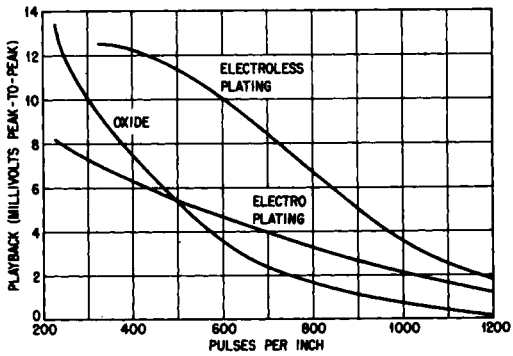


FIG. 5.9. Read voltage variation with recording density for three types of magnetic coatings (Courtesy of Remington-Rand UNIVAC; Jacoby, M. [1962])

still within the field of the head when record current polarity is reversed). A high ratio of coercivity to remanence also reduces self-demagnetization, and it is important that the hysteresis loop be rectangular to reduce the trailing effect and self-demagnetization (see Miyata and Hartel [1959]).

5.2.6. MAGNETIC DRUM STORAGE

At present, magnetic drums and disks (described in the section following) provide the most economical storage of large amounts of data (see Table 5.1) for medium speed storage applications. Drums are used both as the main store of medium speed computers and the auxiliary store of high speed computers. A block diagram of a dynamic magnetic storage

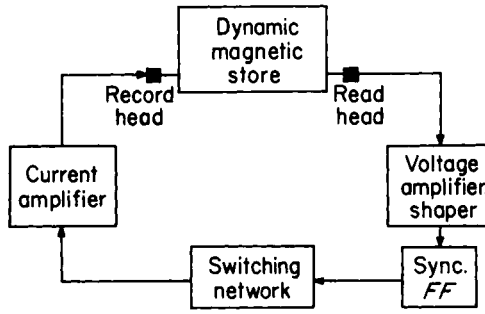


FIG. 5.10. Record-read system for one channel of a dynamic magnetic store

system, essentially the same circuitwise for a drum or a disk system is shown in Fig. 5.10. In either case, there is a metallic surface coated with a magnetizable medium, a motor for driving the surface, a set of read

and record heads, and circuitry as indicated. Usually the medium consists either of a magnetic oxide dispersion that has been sprayed onto the surface and burnished after hardening; or a magnetic material or compound that has been plated onto the surface (see Fig 5.9). The plated surface is superior to the oxide surface magnetically and also mechanically, for it produces a harder, less abrasive surface. As indicated in the discussion on recording techniques, the signals read off the surface are not of suitable amplitude or shape to be used by the sequential switching networks of a computer. Therefore, circuitry is provided to amplify the read signals and convert them to proper shape.

One of the most difficult problems in the design of a magnetic drum storage unit is to maintain a small clearance between the magnetic surface and the read and record heads. No more than a small clearance, 0.0002–0.0001 in., can be allowed if a large recording flux density, needed for saturation of the medium, is to be produced without "excessive" record currents and read amplifier gain. Saturation of the medium with less record current reduces circuitry, power consumption, the trailing effect of the head and improves resolution. There must be limited variation in this clearance during operation. Variations in this spacing stem from two principal sources, namely mechanical and temperature effects. Any eccentricity in the drum surface will cause cyclic variations in the clearance between a fixed head and the surface of the drum. The variation in the path described by the surface of the drum is referred to as "run out." It may be reduced to less than 1 mil by use of a cylinder that has been dynamically balanced and turned on its own bearings. (A slight variation in drum diameter from one end of the cylinder to the other can be compensated for by the initial setting of the heads.) Other sources of spacing variation are vibrations and the mechanical deformation that takes place when the drum is rotated at high angular velocities. If the drum and the structure supporting the heads do not have the same temperature coefficient of expansion, the changes in dimensions occurring from temperature deviations from the norm will alter the set clearance between the drum and its heads. To circumvent this problem, air supported head mounts have been developed whose compliance maintains a minute head to surface spacing. In the hydrodynamic type, the lift is produced by the film of air circulated by the rotating surface while in the less widely used hydrostatic type an external air supply ejects compressed air under the head.

Information on the drum surface is recorded along several distinct tracks. Each track is defined by the imaginary line traced by a head as the magnetic surface passes beneath it. The number of tracks per inch of axial length may vary, but currently may be anywhere from 20 to 80.

The width of the track is determined by the width of the head core. Though a wider track results in less storage capacity, it results in a larger read signal and reduces errors that might arise from flaws along a narrow track. There are many ways in which information may be arranged within the tracks. There may be parallel access to all tracks, or to several parts of a particular track, or there may be access to only one of several tracks at a given time. For the former cases, there are separate record and/or read amplifiers for each track or for each head, whereas in the latter case, only one record and one read amplifier plus appropriate selection circuitry might be used for the entire memory.

Access is gained to the various items of stored information by the same motion that is utilized for sensing. The drum angular velocity in different designs varies over a wide range (see Table 5.1). The maximum velocity is limited by the drum's moment of inertia, and hence, its physical dimensions. In static address dynamic storage systems (see page 197), if there is only one head per track, the maximum access time is the period of one revolution. The use of more heads (and associated circuitry) per track can reduce this figure. It may also be reduced by programming techniques like minimum access coding and address interlacing (see Section 7.5.4). If a track is used as a delay line, access time may be improved by less separation between record and read heads (implying additional head pairs to secure the same amount of storage).

In early static address systems for stored program arithmetic computers, a read and/or a record amplifier was supplied for each head. This is an uneconomical procedure since the mode of operation of these machines is such that only one position of the memory is consulted at any given time. An alternative is to use only one record and one read amplifier and cause either of them to be connected to the head desired. The connection is made by a selection matrix which causes a path to be established between the desired head and the input lines of the read amplifier or output lines of the record amplifier. If the speed requirements of a machine are sufficiently low, then the switching delay introduced by a relay network can be tolerated. For medium and high speed machines, electronic circuits are used which permit power switching at high speed. These selection circuits may be comprised of vacuum tubes or transistors or combinations of these elements with magnetic core switches. The switching network is controlled by the track (or head) part of an instruction's address which is held (with the sector number) in an address register. (See Section 7.5). For a description of a head selection switch formed from transistors and diodes and capable of selecting one out of a 100 heads, see Seader [1958].

5.2.7. Magnetic Disk Storage

There are various applications where the use of one or more magnetic disks, usually in the form of stainless steel or magnetically plated aluminum, may be preferable to a magnetic drum. One of the principal advantages of a magnetic disk store is that it provides a large amount of magnetic surface in a relatively small volume. If the disks are rigid, the problems of run-out associated with a magnetic drum type of store are alleviated. Even with relatively flexible disks, proper spacing between heads and disks can be maintained with less severe mechanical tolerances in manufacturing and less extensive maintenance procedures. Maintenance of a constant separation between the heads and the recording surface is usually achieved by use of an air bearing (see Section 5.2.6).

Single magnetic disk stores with capacities of from 100 to 500 kilobits have been produced which are competitive with drum stores such as those listed in Table 5.1. However, magnetic disks are gaining more extensive application in multi-disk units of several hundred megabits capacity, referred to as mass storage units, intended primarily for information retrieval systems. One of the earliest of these, designed at the National Bureau of Standards, (Rabinow [1953]) was to have 588 disks of 20 inch diameter and a bank of 128 heads. The capacity of this unit using a recording density of 100 bits/inch would be a quarter billion bits. The IBM-RAMAC disk file (Noyes [1957]) was the first operational multi-disk unit to be widely used with digital computers. It contains 50 disks so mounted as to rotate about a vertical axis. There are 100 tracks per side, each track having a capacity of 500, 8-bit alphanumeric characters. The disks, 24 in. in diameter and 0.1 in. thick, are of aluminum coated with iron oxide. The density of recording on the inside track is about 100 bits per inch and on the outer track about 55 bits per inch. The access mechanism, of which there may be one or more, positions a pair of heads to any track on any disk. These heads are mounted in a pair of arms which can be moved vertically to the level of any disk and then radially to straddle it. The average access time is 0.5 sec, the maximum 0.75 sec. Head spacing is maintained by an air bearing produced from tiny air jets in an annular manifold surrounding the magnetic elements. A 0.001-in. spacing is maintained despite any axial run-out in the disk. Reduced precision in radially positioning the heads is obtained by using an erase head that erases a wider track than the following write gap records. Thus, no magnetically disturbed track edges contribute noise to a newly recorded track which might not coincide precisely with the track previously written. The accuracy required in positioning a head along a track is reduced by use of a self-clocking system rather than a timing track. (See Seader [1957]

TABLE 5.1. Characteristics of Rotating Magnetic Stores

Drum Stores	Diameter (inches)	Tracks			Latency time (ms)	Capacity (kilobits)
Bryant 512-A	5	240			5	625
Ferranti 1009	10	144			10	432
Librascope LGP-30	6.5	64			17	131
Multi-Disk Mass Storage Units	**Diameter (inches)**	**Tracks/ surface**	**Zones/ surface**	**Max. head positioning time (ms)**	**Latencyh time (ms)**	**Capacity (megabits)**
Bryant 4000 (24 disks)	39	768	6	100	67	720
IBM 1301-2 (50 disks)	24	250	1	177	33	392
Telex II (64 disks)	31	256	2	150	50	617

The maximum time required to locate an item in storage is the sum of the maximum head positioning time and the latency time (the latter being defined as the period of one revolution). In a rotating memory with fixed heads the access time is some fraction of the latency time, depending on the number of heads per track and techniques for addressing information (see Section 7.5.4.). If constant frequency recording is used, the outer tracks will have a lower recording density than the inner one, so for improved utilization of storage capability, a large disk is usually divided into two or more zones, each operating at a different frequency and with the maximum pulse density on the innermost track of each zone. In earlier units, as cited above, the positioning arm moved axially as well as radially, presenting a severe mechanical design problem and resulting in a relatively long access time. In most current units a separate positioning arm for each disk holds as many heads as there are zones on both sides of a disk, and positions heads along tracks within a specified zone.

Present mass storage units, such as listed in Table 5.1, have capacities near one billion bits, with pulse densities averaging about 400 ppi (pulses per inch), and average access time per positioner under 100 ms. These capabilities can be increased greatly: advanced recording systems will permit a several fold increase in recording density, improved positioning systems can extend the number of tracks per inch, from the present average figure of about 50, by another order of magnitude, and positioning time can be reduced by use of several independent positioners.

5.3. Static Magnetic Storage

5.3.1. Magnetic Core Storage

Magnetic core storage devices are based on the use of materials exhibiting hysteresis loops which are practically rectangular. These materials include nickel iron alloys, molybdenum permalloys, and ferrites. Three basic ways these cores may be fabricated are: (1) from thin ribbons of a metallic material wound into a toroidal core, (2) from powdered metals sintered in toroidal form, and (3) from ferrites molded in toroidal form. Both metallic ribbons and ferrites are available that exhibit nearly rectangular hysteresis loops (similar to the loop shown in Fig. 4.48). The metallic cores offer the advantage (in relatively small capacity storage applications) of a lower coercive force. However, the ferrite cores have a faster switching action, are lower in cost and lend themselves to mass production. A typical ferrite core has an outside diameter of 0.050 in., an inside diameter of 0.030 in., and is 0.015 in. along its axis. For any given material, switching current requirements decrease linearly with the

diameter (since the flux path is along a circumference). Because a ferrite core is brittle, the ratio i.d./o.d. for small cores is held to about .60 to .75, adversely affecting hysteresis loop squareness.

For memory applications, the core material should satisfy the following requirements: 1) time required for switching (i.e., flux reversal) should be small to allow higher data rates; 2) effective incremental permeability at the positive and negative residual flux points should be small so that a partially disturbed core (see sections following) produces no effective output; 3) reduction of residual flux caused by repeated interrogating current should be small to lessen the probability of eventually losing recorded information. Materials whose loops exhibit a greater degree of rectangularity are better in all these respects. If a memory core is to be selected by coincident currents (see Section 5.3.2) it is important (for positive, reliable switching action) that the hysteresis loop have a square knee.

The switching time τ (in seconds) is defined by Eq. (5.3)

$$\tau = S_w/(H - H_c) \qquad (5\text{–}3)$$

where H is the applied magnetomotive force (in oersteds), H_c the coercive force of the material (the value of H where the hysteresis loop crosses the H axis) and S_w the switching constant. In coincident current operation, H cannot exceed $2H_c$. In practice H is chosen $\approx 1.5\ H_c$ for optimum discrimination between full and half excitation. For a wide variety of ferrites and metals, S_w does not vary significantly, but the coercive force, H_c, can be varied considerably by changes in composition and heat treatment. For faster operation, materials with a larger H_c are used, and higher drive currents are required.

Magnetic cores exhibit the following characteristics pertinent to their use in a large capacity storage system. First of all, they provide nonvolatility of stored data. Once a core has been set to a particular state, it will remain in that state until a disturbing force of proper sign and amplitude is applied. For example, if the power supply is cut off deliberately or accidentally, information in the cores will not be altered except if the transients are large enough to disturb the cores' remanent states. Also regeneration circuits do not have to be provided to prevent the loss of information resulting from a process of gradual degradation (as, e.g., in an electrostatic storage system, where the charge gradually leaks off unless periodically restored). However, the read-out process is essentially destructive in nature and as a result, either regeneration circuitry or a spe-

cial nondestructive read-out scheme must be provided (see Section 5.3.2.5). The switching time is fast, of the order of a microsecond.

5.3.2. Coincident Current Core Arrays

The coincident current type of memory depends on a two-level amplitude discrimination scheme for core selection. Cores with a rectangular hysteresis loop are arranged in two dimensional arrays, as shown in Fig. 5.11. To record or regenerate a bit of information in a single core,

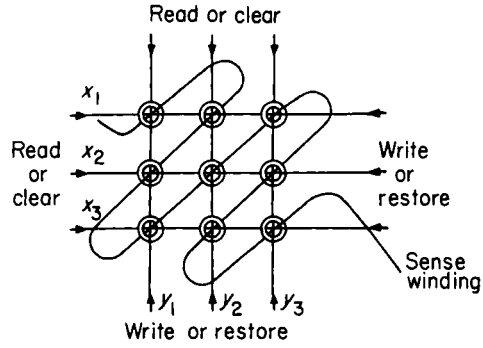


Fig. 5.11. Coincident current magnetic core storage array, showing directions of current flow for read or clear, and write or restore operations

a pulse of current of magnitude $I_m/2$ is applied simultaneously to the row and column wire threading that core. I_m is chosen of such magnitude that the magnetizing force H_m it produces is adequate to switch a core, whereas $I_m/2$ is not. Therefore, the core at the intersection of the energized X and Y lines will be switched, whereas all other cores along the same X and Y lines are only partially disturbed.

The read-out process is similar to the recording scheme. Interrogation pulses both of the same polarity are applied to the selected X and Y lines. If the application of these pulses causes the core to be switched, a voltage pulse will be induced in the read-out winding. For example, assume that the interrogation pulse is chosen of such polarity as to produce a negative magnetizing force. Then if the core is in state $-B_r$, (Fig. 4.48) the interrogation pulses produce no effect, whereas if it were in state $+B_r$, a switching action would be effected. Thus the state of the core prior to interrogation can be inferred from the effect of interrogation. After interrogation, the state of the cores before interrogation is restored by means of circuits actuated by the read-out signal.

The whole array is threaded by a single read-out wire, referred to as a sense winding. A voltage pulse will appear on this sense winding only when the interrogation of a selected core causes that core to switch from one state of saturation to the other. For minimum coupling between the drive and sense windings, they should be placed at right angles to one another. This would result in no flux leakage and a relatively small capacitance between the wires. Though the diagonal pattern provides some coupling because of the 45° relationship between the wires, it is still satisfactory while easier to fabricate. Note that in any row or column, the direction of the magnetically induced voltages due to a partial selection is opposite in alternate cores so that opposing induced voltages cancel each other, allowing the signal from the selected core to dominate.

Each of the straight wires used for driving and sensing effectively pass a single turn through each core. The large currents needed to produce an adequate magnetomotive force can be obtained from saturable transformer or diode decoding matrices. Multiple turns would reduce current requirements, but would make the array more costly and difficult to fabricate and decrease the operating speed.

In the writing and interrogating scheme described, current pulses must be generated in two directions, one for recording and one for sensing. If this is not convenient, two sets of wires may be used. Also, if the hysteresis loop of the core is far from ideal, it is desirable for improved reliability of operation to have a ratio greater than two between the current in the selected core and the largest current in any nonselected core. For example, a selection ratio of 3 to 1 can be obtained in either of the following ways: A current of $I_m/2$ is sent through the X and Y line of the core to be selected, as before, but in addition all other selection windings are driven with currents of $-I_m/6$. Thus the net current in all cores except the selected one is $I_m/3$ or $-I_m/3$. Another scheme providing a 3 to 1 selection ratio makes use of an extra winding passing through all cores, and requires simpler driving current circuitry. A current $2I_m/3$ is applied to the X and Y line of the core to be selected, and an opposing current $-I_m/3$ is passed through the additional winding. Thus I_m is applied to the selected core and either $I_m/3$ or $-I_m/3$ to all others. In both of these schemes the directions of all applied currents are reversed during read-out. Also the opposing currents are applied just prior to the selecting currents and maintained until the selecting currents have been removed.

There are certain disadvantages to the coincident current selection technique: Restriction of the magnitude of the applied mmf produces a limitation in switching speed and allows only small tolerances in the amplitude of the drive current. Also, the less than ideal rectangularity of

the hysteresis loop limits the permissible size of an array (see Section 5.3.2.2).

5.3.2.1. Selection of a Word at a Time

In a high speed computer a better balance can be achieved between time spent in arithmetic operations and in data transfer to and from the memory if there is parallel access to all bits of a word. One way to accomplish this is to provide as many two-dimensional arrays as there are bits in a word. Then each word is addressable simply by its X, Y coordinates, and a particular Z plane is always associated with a particular significant place in a word. A particular configuration which has been found to be practical is arranged as follows: Each two-dimensional array has a set of X and Y windings and a current of $I_m/2$ can be applied to any X and Y winding at a time. Corresponding X and Y windings in each plane are connected in series. For each XY plane, there is a separate Z drive winding (designated as an inhibit winding) which passes through all cores in the plane, and, also, a separate sense winding. The operation is as follows: since corresponding X and Y windings in different planes are connected in series, application of drive current to a particular XY address causes a 1 to be recorded in the corresponding positions of all planes. Therefore, to allow a recording of 1's in some XY planes and 0's in others, an opposing current, $-I_m/2$ is applied to the Z drive winding in all XY planes in which a 0 is to be recorded. In reading, the directions of the applied XY currents are reversed and the Z winding is not used. The individual bits of the word are read from the sense windings.

Though the inhibit and sense windings are not used simultaneously, it is not practical to replace them with a single winding for the following reasons: First of all, in order to cancel induced signals that are unwanted, the sense winding passes through alternate cores in opposite directions (see Fig. 5.11). The inhibit winding must pass through all cores in the same direction relative to the X and Y drive windings. Also, there would be a problem in isolating the large voltage applied to the inhibit winding from the corresponding sense amplifier. If not isolated, the amplifier would be driven to saturation and could not recover in time to respond to the small amplitude of the read-out signals.

Some important parameters of a core storage system are the simplicity of the wiring configuration, the selection ratio, and the number of drive lines. A more complex core selection system (see Section 5.3.2.4) can reduce the number of drive lines, e.g., a quadruple-coincident selection system for a 4096 bit storage plane can be built requiring only 64 drive lines compared to 256 unidirectional lines for a two-dimensional selection

system. However, the use of more dimensions in the selection system within the array means more wires, a smaller selection ratio, and passage of each driving current through more cores.

5.3.2.2. *Disturb Signals*

We will review here various unwanted signals that are generated during the operation of a coincident current memory because of less than ideal rectangularity of the cores' hysteresis loops and other causes. First of all, we note that only moderate rectangularity is sufficient to prevent demagnetization due to the cumulative effect of successive half amplitude signals. A more serious problem is that voltages induced in the sense winding by all the half excited cores on the selected lines are cumulative and may swamp the desired read-out signal. (The effects of disturb voltages induced directly from the drive windings may be essentially eliminated by threading the sense winding in such a way that there is an equal number of positive and negative flux linkages.

As an example, consider a selection ratio of one-half. In this case, whenever a core is selected by application of I_m, a half selection current of $I_m/2$ is applied to all other cores in the row and column intersecting the core. If a half select read signal is applied to a core containing a 1, the state of the core is changed from 1 to 1_r. If a half select write signal is applied to a core containing a 0, the state of the core is changed from 0 to 0_w. The resultant states are referred to as disturbed 1 and disturbed 0 states, respectively. When half select signals are applied alternately in the write and read directions, minor hysteresis loops are traversed as shown in Fig. 5.12 (a). Since the difference between the first and successive

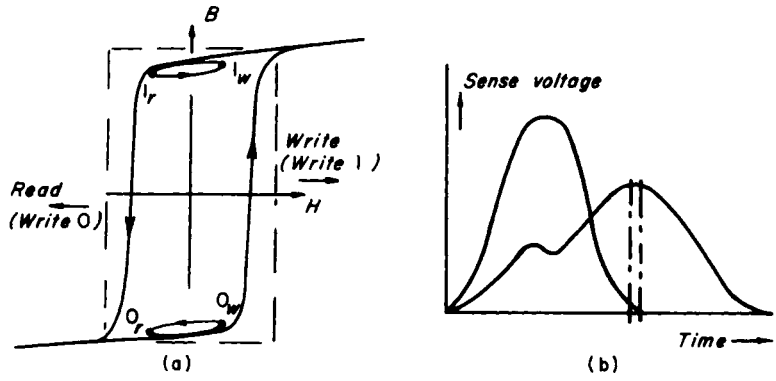


FIG. 5.12 (a) Minor hysteresis loops from half-select signals, (b) strobing of sense voltage.

minor loop traversals is usually small, it is assumed here that each core arrives at a stable minor hysteresis loop after several half select signals. The output flux obtained in reading a core in a disturbed 1 state is less than that from a core in an undisturbed 1 state, and when a core in a disturbed 0 state is read, an unwanted signal is produced by the process of restoring the undisturbed 0 state.

In reading, a partially selected core (operating in a symmetrical coincident current loop) will not produce the same small disturb signal when in the 1 remanent state as in the 0 remanent state. Earlier it was stated that to reduce the effects of disturb voltages, the sense winding should link all partially selected cores in a way that equalizes the number of positive and negative core linkages. However, there is a pattern of information along the selected X and Y lines for which the difference signal (commonly referred to as delta noise) is a maximum. The pattern (and its complement) yielding the maximum delta noise is called the worst storage pattern.

A technique used to alleviate the delta noise problem is to apply after each read or write operation a demagnetizing half excitation referred to as a post-write-disturb pulse. This tends to equalize the reversible flux changes for the two remanent states during subsequent half excitations, by carrying all cores to either the 1_r or 0_r state. The diagonal sense winding pattern effectively cancels voltages induced from the unselected cores except for the voltage differences caused by the relatively small difference in slope between the left hand portions of the 1 and 0 minor hysteresis loops.

The most frequently used method to discriminate against disturb voltages is to time strobe the output voltage. This technique is effective because the reversible flux changes on half excited cores occur faster than the irreversible flux changes on the selected core. Fig. 5.12 (b) shows sampling of the sense line voltage after the waveform on the left (representing a worst combination of half-select and delta noise voltages) has decayed.

The rectangularity and uniformity of present day ferrite cores is such that an array much larger than 64 × 64 cannot be operated reliably with a single sense winding. For larger arrays, the problem of disturb voltages can be solved by splitting the sense winding (see Best [1957]).

5.3.2.3. Core Storage Cycle

Let us review briefly the nature of writing and reading in a core storage array in which each bit of a word is read simultaneously from corresponding positions of all XY planes. To write, a drive current $-I_m/2$

is applied to a selected row and column in all planes, simultaneously with the application of an opposing current $I_m/2$ through the inhibit winding in all XY planes where a 0 is to be recorded in the selected position. Thus, writing is essentially a "write 1" process. To read, a drive current $I_m/2$ is applied to the selected row and column in a direction opposite to that for writing. Thus reading is essentially a "write 0" operation and previously recorded 1's are detected by inspecting the sense winding at the time when the cores switched from 1 to 0 are developing their maximum output voltage. This mode of operation makes it desirable to establish an operational sequence in which each write operation is preceded by a read operation regardless of whether a word is to be written into or read from the memory. This is because when writing, the preceding read (write 0) operation clears the word, and when reading a subsequent write operation (write 1) is needed to restore those cores switched from 1 to 0 by the read operation. Other routine operations necessary to the consultation of the memory in a particular system may be included in an over-all operational sequence referred to as a core storage cycle. For example, after the write operation, a post-write-disturb pulse may be applied to all cores (through the inhibit windings). During this period, the addressing circuits can be set to the address of the storage location to be consulted in the cycle following.

The cycle time of a random access memory depends on several factors: 1) time for address decoding, 2) transmission time along write and read lines, 3) switching time of a memory element (e.g., Eq. (5-3)), 4) time before reading to allow decay of a large signal pick-up by a sense winding during writing (even after measures to reduce pick-up and its effect on the sense amplifier), 5) delays in sense amplification circuits.

5.3.2.4. Memory Drive Systems

In an $n \times n$ coincident current array, the problem of switching into n^2 cores is reduced to that of switching into $2n$ channels. Two many-to-one diode matrix switches (described in Chapter 4) could be used for this purpose, the output of one switch driving the X lines and the output of the other driving the Y lines. Since two driver circuits are required per line to obtain current flow in two directions, $4n$ drivers are required. In some of the early core memories, there were two tubes and two one-turn windings at each line to provide the two polarities of drive. Each tube supplied a current of about 0.5 amps. In present large arrays, the number of current drivers is often reduced by the use of sets of external switching cores to drive the storage array. Also, joint use of coincident currents, diode matrices, and magnetic core matrices all contribute to an efficient selection system for a large array.)

An important fact entering into the design of a magnetic core switch is that it is only necessary for a single switch core to be driven to one state, while all others can be in the opposite state. For example, before energizing currents are applied to the core switch, all cores are in the same remanent state and after they are applied only one of them is switched. This is in contrast to the coincident-current memory matrix where the cores can be in any pattern of positive and negative remanent states.

The first type of magnetic switch we will consider is the biased coincident current switch shown in Fig. 5.13. All cores are biased by a

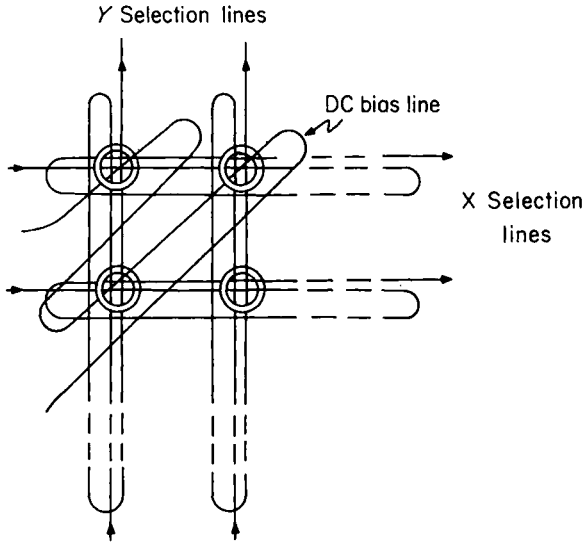


FIG. 5.13. Biased coincident-current magnetic core switch

direct current which is equal in magnitude and opposite in sign to the excitation of a row or column line. Simultaneous excitation of both a row and column multiturn line switches the core. At the termination of the row and column drives, the dc bias restores the selected core to its initial state. Each core of the array has an output winding (not shown in Fig. 5.13) coupled to a row of the memory array. A similar switch has output windings coupled to the columns of the memory arrays. The switch provides part of the address decoding, both polarities of output, and a better impedance match to the output of the drivers. The latter capability results from the fact that several turns can easily be provided on the switch cores because they are larger than the memory cores and only a relatively small number of them are required. For example, only

an 8×8 switch array is required to drive the X (or Y) lines of a 64×64 memory array. A switch similar to the one in Fig. 5.13 is the anticoincident current switch. It has no bias line, so a selection is made by first sending reset current through all X lines except the one containing the core to be selected. Then, current in the set direction is applied to the Y line containing that core. To reset, the Y line current is terminated and reset current applied to the X line containing the selected core.

We will consider next a multicoincidence magnetic switch. Figure 5.14(a) shows a switch of this type with three inputs and eight outputs. In this figure and also Fig. 5.14(b) the mirror system of notation for magnetic core circuits is employed to indicate the location and polarity of the windings: The cores are represented by the dark horizontal lines, and the various drive lines by the vertical lines. A short diagonal line at the intersection of a core and wire indicates that a particular drive winding is present on that core. The convention is to consider the current in a wire as a beam of light. The core will be driven to a set or reset state in accordance with whether the diagonal line, considered as a mirror, reflects the beam to the left or right. The operation is as follows: The output signals of the flip-flops holding the X (or Y) address control a set of drivers. After all cores of the switch have been reset by application of a current through the restoring winding, current is applied to one winding of each input pair according to whether the corresponding flip-flop is in the 1 or 0 state. For every combination of states of the flip-flops, one core will have all windings driving it toward the set state

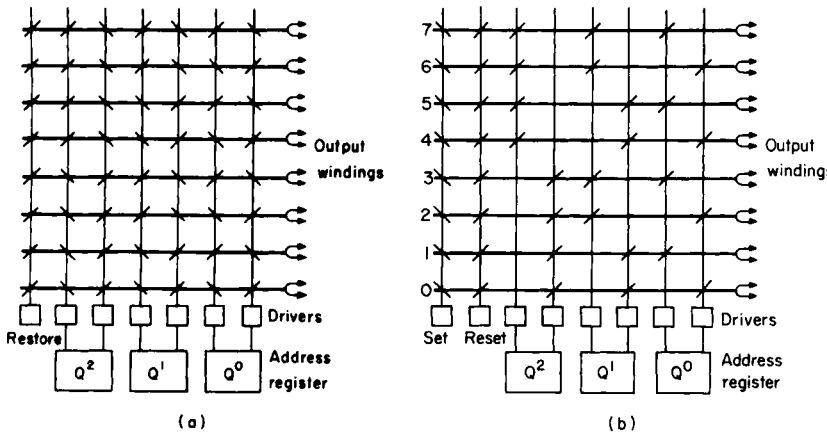


FIG. 5.14. (a) Multicoincidence magnetic core switch, (b) biased multicoincidence magnetic core switch

while others will have some windings driving them toward the set state and some toward the reset state. The number of turns of the reset winding is $k - 1$ times that on the set windings where k is the number of binary positions Q^i) so only the selected core is set. Since all of the driver currents are additive on the selected switch-core, in a larger matrix the drivers can supply less current to each winding. Also, a larger number of series windings will be driven by each driver.

In Fig. 5.14(b) another form of magnetic switch referred to as a biased multicoincidence switch is shown. In this case, the X (or Y) address flip-flops control a set of bias drivers. The magnitude of the current generated by any bias driver is adequate to switch any core it affects to the state N. For each address in the flip-flops, one and only one switch core will receive no bias current. For example, if the flip-flops hold 011 (binary three) a bias current is applied to each switch core except number three. After a particular core has been selected (by not having a negative, biasing current applied to it), the next step is as follows: A pulse of current is applied to all switch cores by the set driver in such a direction as to drive each of them to positive saturation if they were initially unbiased. However, since only the selected core has no initial bias, it is the only one actually driven to a state of positive saturation. Switching of the selected core induces a voltage in its output winding which may be used as an X or Y drive signal for the storage array. The selected switch core is returned to its initial state by application of current from the reset driver.

When the number of windings per switch-core, or the number of windings connected in series becomes greater than desirable, these figures may be reduced by doing some preselection with the aid of diode switching networks.

5.3.2.5 Nondestructive Readout

Before describing certain schemes for sensing the state of a magnetic core without altering it (in order to avoid the necessity of restoring it) we will review some basic characteristics of ferromagnetic materials. Such materials usually have small regions (about .1 mm in length) called domains in which all electron spins are aligned. (There are also cases where domains do not form, the magnetization changing direction in a continuous manner). Regions separating domains with different alignments are known as domain walls. When an external magnetic field is applied, domains similarly aligned grow, extending their walls and reducing adjacent domains. Domain rotation, i.e. rotation of the magnetic moments of all domains, may also occur (though this happens more frequently in fields of higher intensity).

Nondestructive readout based on the phenomenon of elastic motion of domain walls is described by Newhouse [1957]. A magnetizing force much greater than the coercive force can be applied without causing switching if it is applied for only a brief interval (about .1 μsec). During this period walls can be moved only short distances, within their elastic limit, and the movement is reversed when the force is removed. The readout voltage varies by a ratio of about three to one, depending on the core's remanent state, and the peak output is about 15% of that produced by switching. The reversible flux change may also be aided by two other processes: (1) temporary coherent rotation over a small angle within domains, (2) temporary domain reversals around imperfections in the material.

In quadrature field methods, readout is by means of a magnetic field orthogonal to the remanent flux (which is in a circular path around the core). Since sensing is based on rotation of magnetic moments rather than domain wall motion, the state of the core is inferred from the polarity rather than the amplitude of the readout voltage. The readout rate can be very high because the magnetic moments can rotate within the readout pulse's rise time. Wiring an array of elements as in Fig. 5.15 (a) is difficult; it is also difficult to produce cores with holes as shown in Fig. 5.15 (b). Though the FLUXLOK scheme, shown in Fig. 5.15 (c), requires more ampere turns for driving than the scheme of Fig. 5.15 (b), it uses a standard core, and the readout solenoid is relatively easy to wire. In the FLUXLOK system, the opposing circular mmf's set up in the core cancel one another, leaving its remanent state essentially unaffected. The magnetic moments are rotated slightly; but when the disturbing field is removed, realign to their original orientation. The output waveform has a positive and a negative pulse at the leading and trailing edge (whose interval is defined by the interrogating pulse), respectively, or vice versa, depending on the initial orientation of the circular flux.

In the RF sensing method (Widrow [1954]) a burst of RF current of one frequency is applied to a selected coordinate of an array and another frequency to a selected coordinate (using the same windings employed in writing). The output voltage produced by each core at the intersection has a difference frequency component. Its phase for the two possible residual flux states of the core differs by 180°. The cores are not switched because the frequencies (in the region of 6 Mc) are so high that there is not time in a half cycle for a permanent change, and current in the opposite direction during the next half cycle restores any temporary change. Because of its complexity and critical operation, this technique has found little application.

The so-called 0-flux method (Olsson [1960]) is based on possible

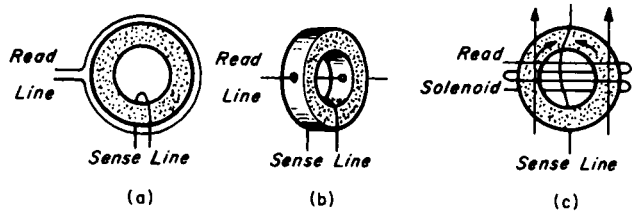


FIG. 5.15. Nondestructive sensing arrangements for magnetic core storage

use of the demagnetized (0-flux) state of a core to represent one of the two binary states. It allows a readout frequency comparable to that of conventional (destructive) readout schemes with lower readout currents (less than 100 ma for an .080 inch core of low coercivity).

Since reliability of operation is a prime consideration, evaluation of a sensing method must take into account such items as the squareness and uniformity of cores that are acceptable, the amount and complexity of circuits required and the output signal-to-noise ratio. Readout methods producing relatively low signal-to-noise ratios are better suited for a word organized memory (Section 5.3.2.6), where no half-activated cores are in series with an activated core on a sense line to contribute disturb signals.

5.3.2.6. The Word Organized Memory

We will now describe a magnetic core storage arrangement (see Fig. 5.16) somewhat different from those considered in the preceding sections.

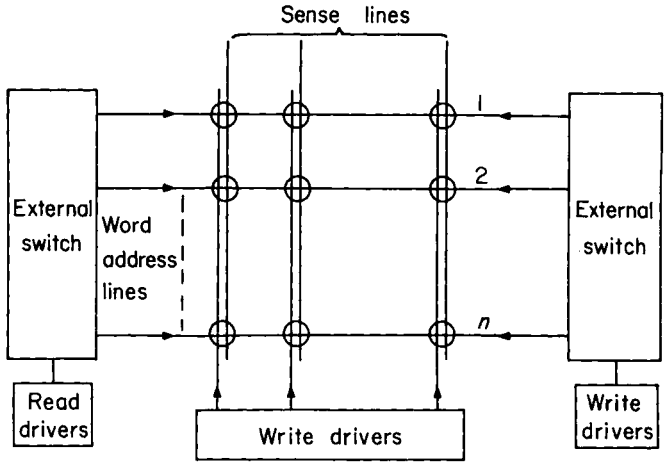


FIG. 5.16. Linear selection: word organized magnetic core storage array

The operation of this array, whose selection scheme is called "word-selection" or "linear-selection," is as follows: First of all, switching is performed external to the array itself. Secondly, each row corresponds to a word. To read a word, the selected row is driven with a pulse of sufficient amplitude to switch all cores in the row to positive saturation. This causes voltages to be induced on all column windings where the cores of the interrogated row were at a state of negative remanence. To write or rewrite a word, a current half the amplitude required for switching is applied to the selected row in the direction opposite to that for reading, i.e., one producing a negative mmf. Simultaneously, write currents of the same amplitude are applied to those columns in which the cores of the selected word are to be driven to a state of negative saturation. This results in a switching action where those columns intersect the selected row.

Because read-out is by external word addressing and involves no current coincidence, the read-out lines are free of disturb signals from cores of other words. Extraneous signals on the read-out lines occur only as a result of the minor flux changes of those cores on the selected row which are driven further into saturation and these are easily distinguishable from a major flux change. In contrast to coincident-current operation, the less than ideal squareness of the hysteresis loop does not limit the permissible size of the array. The read-out signal is clean and simpler to detect, there is no limitation on the magnitude of drive current that can be applied, as with internal coincident current selection, and because of the large tolerances, the driving circuits can be simpler, too. By increasing this current, the read-out switching speed can be increased to the limit imposed by the characteristics of the external switch and the heating of the memory cores should they be switched at the high repetition rates possible with the short switching time. Since the write operation does depend on current coincidence, the writing currents and hence the speed of writing are limited accordingly.

To summarize briefly, in coincident-current memories half-select currents are critical in amplitude and waveshape, the core material must have a high squareness ratio (since this affects the sum of half-select voltages), the cores must be selected for uniformity (to improve noise rejection by cancellation effects) and the sense amplifier design is critical because of high level noise from partially disturbed cores. In word organized memories these requirements are greatly relaxed; in particular, the greater tolerance on drive current allows reliable operation over a wider temperature range despite a temperature coefficient of coercivity of .5 to .7%/C° for ferrite core materials. On the other hand, above the region of 10,000 bits or so, a word-organized memory requires considerably more associated semiconductor elements for addressing, writing and sensing

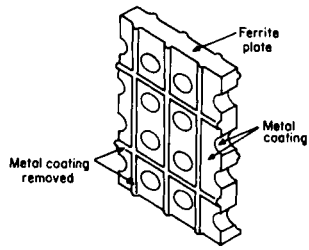


FIG. 5.17. Section of apertured ferrite plate

(about five times as much for a 100,000 bit memory and about ten times as much for a million bit memory).

5.3.3. THE APERTURED FERRITE PLATE

Production of magnetic core assemblies similar to those described in Section 5.3.2 involve complex assembly operations if the total number of bits is very large. This is because of the difficulties involved in threading wires through the tiny ferrite cores. The ferrite plate storage system, originated at RCA, is composed of thin (0.020 in. in prototype models) plates molded, with a regular array of holes, from a rectangular hysteresis loop ferrite (see Fig. 5.17). It combines the relatively high density of storage sites of a continuous medium with the high access rates of individual cells: These cells are defined by the apertures, for, by applying a magnetic field that produces either a clockwise or counterclockwise flow of magnetic flux in the small area of material surrounding each aperture, a bit can be stored in terms of the direction of remanent magnetization about each aperture. With a center-to-center spacing of the holes greater than twice the hole diameter, the interaction between holes is negligible. In a standard 16×16 array in which the plate is less than an inch square the holes are 0.025 in. in diameter with 0.050 in. between centers. About 300 ma are required for switching; the switching time is around 1.5 μsec.

Apertured ferrite plates can be operated either as a coincident-current or word organized memory. The "word select" type is used almost exclusively because it is less dependent on the uniformity of cells on a plate, in addition to its other advantages. (For a description of both types, see Rajchman [1957]. In the "word select," memory, a stack of plates can be driven by an external switch which energizes the selected word location without half-exciting other locations. The address selecting switch, too, can be made of a stack of plates, and can be set in register with the storage stack. An address selecting conductor, simply a straight

wire, is passed through each set of apertures in register. The switch stack is threaded, also, by X and Y selection windings.

To summarize, the apertured ferrite plate is more economical to fabricate and test than a corresponding number of cores, and the assembly and wiring of plates into a large memory is considerably simpler than the fabrication of a magnetic core assembly. An important problem still to be solved in the development of this type of memory is the production of plates with more uniform characteristics.

5.3.4. The Transfluxor Array

The blocked and unblocked remanent conditions of a transfluxor (see Chapter 4) may be used for binary storage in a random access store with coincident current selection. (See Fig. 5.18.) Coincident-current write pulses, applied simultaneously to the windings linking leg 1 (Fig. 4.59) in a selected row and column, set the selected transfluxor to either a blocked or unblocked state. For read-out, a pair of pulses, one in the prime and one in the drive direction, is applied to each read line. If the transfluxor is unblocked, fluxes in legs 2 and 3 reverse back-and-forth and return to their initial state. If it is blocked, they remain in their initial state. These flux reversals induce voltages on the sense winding.

In a coincident-current core memory the half-select currents must be precisely controlled since they must not switch a core. A coincident-current transfluxor array can be operated by biasing the small hole to saturation, applying a drive current to one line to overcome the bias (without switching) and to the other to cause switching. Since these drive currents can be larger than in a coincident-current core memory, switching can be faster. Because the drive current can vary over a wider range, operation is less temperature sensitive. Present coincident-current transfluxor memories can operate from $-10°C$ to $+65°C$ with cycle times of 6 to 10 μsecs and with a maximum of about 4096 bits per sense winding. A word-select transfluxor memory can operate from $-55°C$ to $+100°C$ with cycle times of about 1 μsec.

The nondestructive nature of the readout process in a transfluxor memory simplifies the readout circuits because there is no need to activate write circuits for selective restoring. A program can be read from memory with less chance of accidental erasure and, in certain applications requiring high speed reading only of a semi-permanent electrically alterable memory, the write circuits can be disconnected and removed, once the program has been loaded. Another potential advantage of a transfluxor memory is that write and read operations could be done simultaneously at two addresses.

In some respects the transfluxor memory does not compare as favorably with a core memory, e.g. its windings are more complex and difficult to

assemble, it is larger, transfluxors are more difficult to produce and test, and the drive system (write and read circuitry) is more complex.

5.3.5. The Twistor

The action of the twistor storage element is based on the fact that a torsion applied to a magnetic wire shifts the preferred direction of magnetization into a helical path inclined 45° from the axis. Figure 5.19

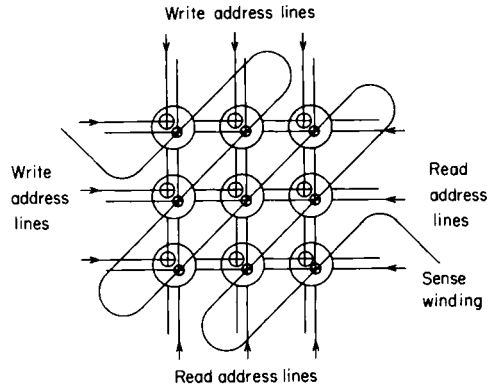


Fig. 5.18. A coincident-current transfluxor storage array

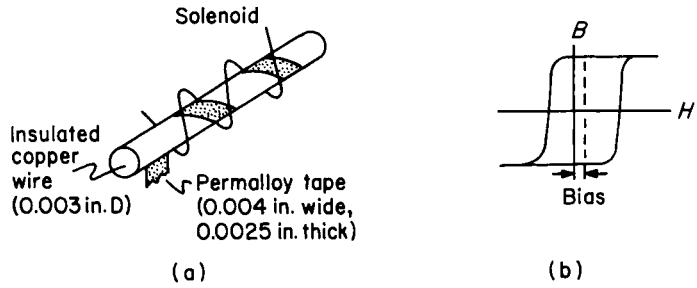


Fig. 5.19. Twistor storage element: (a) physical form and (b) hysteresis loop showing bias due to open magnetic structure

shows one form of the twistor storage element. It consists of an insulated nonmagnetic wire on which there is wound a magnetic wire or ribbon in the form of a helix, at an angle of 45° to the axis of the nonmagnetic wire. There is also a solenoid about the central wire. Coincident application of a current to the solenoid and central wire can produce a magnetic field whose lines of flux follow the path through the helical wire and from its ends join through the space around the element. Because of the

anisotropy produced by its longitudinal tension, the magnetic wire exhibits a hysteresis loop that is markedly rectangular. Also, because of the open magnetic structure there is a bias in the loop (see Fig. 5.19), proportional to the magnitude of the ambient magnetic field.

The form of twistor first reported was simply a twisted magnetic wire with a solenoid about it. Lines of strain produced by the twist yielded an anisotropy producing an easy direction of magnetization. The twistor action was obtained by application of current through both the magnetic wire and the solenoid. The advantage of the form of twistor shown in Fig. 5.19 is that the tension in the magnetic wire is permanently set as it is machine wrapped, allowing elements with uniform properties to be obtained, while the amount of twist in the simpler twistor is not as easily controlled.

A memory array of twistor wires is shown in Fig. 5.20. It is designed

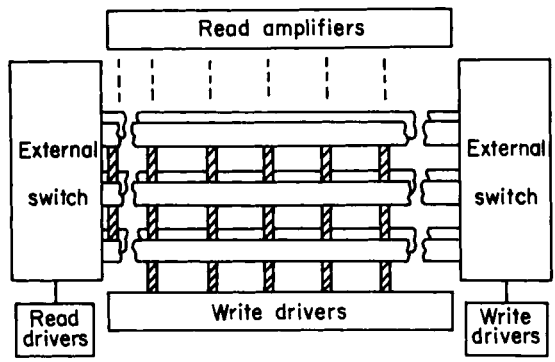


FIG. 5.20. Word organized twistor storage array

for linear selection. Insulated copper wires wound with magnetic wire or permalloy tape are embedded in parallel lines, spaced ten to the inch, between two sheets of plastic. Similar plastic strips with embedded ribbons of copper wire are placed about the twistor sheets to provide the word selection solenoids. Writing is by current coincidence, a pulse being applied to a selected word solenoid and to the twistor wires corresponding to the columns of the word in which 1's are to be stored. The speed of switching may be improved by use of a bias solenoid placed about the entire array. Direct current through this solenoid biases the hysteresis loop even farther to the left, thus allowing a greater half select current to be applied without causing switching. For read-out, the appropriate word solenoid is pulsed so as to change stored 1's to 0's, thereby generating a read-out signal in the central wire. This signal is of good amplitude because of the

number of turns of the magnetic wire. In the twistors previously magnetized in the 0 state, small reversible flux changes are produced whose radial component is opposite in sign to that of the irreversible change. The solenoid current is of the order of amperes, and the central wire current tenths of an ampere. The read-out signal is about 15 mv and the storage cycle time about 6 μsec.

A major disadvantage of the twistor memory is that, because of the open magnetic structures, it can be affected by ambient magnetic fields. However, the bias solenoid, placed about the entire array to permit faster switching, also yields some control over the ambient magnetic field. The principle advantages claimed for the twistor memory are a lower cost per bit and operation over a wide range of temperatures.

5.3.6. Magnetic Thin Film Storage Elements

One of the newest magnetic elements suitable for large capacity, high-speed storage systems is the magnetic film element. An array of memory elements is formed by distinct islands of a magnetic material vacuum deposited on a flat glass substrate. The deposition is made on a heated glass substrate in the presence of a dc magnetic field in the plane of the substrate so that each element assumes a preferred magnetic axis in the direction of the applied field. This establishes two stable states of magnetization (both parallel to the preferred direction). The film exhibits a square hysteresis loop in the preferred direction and an almost linear loop in the transverse direction. The squareness of the hysteresis loop can be worsened by the demagnetizing field due to free poles at the edges of the film. This effect is negligible, however, provided the ratio of film diameter to thickness is very large, e.g., the diameter must not be much less than 4 mm for a thickness of 2000 A°.

In the thin film element, utilization is made of a different phenomenon than usually* associated with the switching of ferrite cores. A reversal of magnetization is produced not by sequential rotation of the atomic magnetic moments, in the form of moving ferromagnetic domain walls, but by simultaneous rotation of all atomic magnetic moments (coherent rotation). This permits faster switching with moderate driving current. Also, because of the favorable surface to volume ratio of thin films, high

---

* See Shevel, W. L., Jr. [1959] Observations of rotational switching in ferrites, *IBM Jour.*, **3**, 93-95, which reports three mechanisms of flux reversal in ferrite cores: domain wall motion, incoherent and coherent rotation. The switching threshold for incoherent rotation is two to five times greater than for domain wall motion, and for coherent rotation about ten times greater.

repetition rates, which produce excess heating in core memories, are obtainable. A practical difficulty in this respect results from the fact that because of the air return flux path, the elements cannot be too closely spaced nor their diameter too small. As a result, the physical size of the array makes the propagation time of the driving signal greater than the switching time of the element.

A desirable feature of the thin film coincident current array is that it allows the conductors for the row, column, and inhibit drives to be made by printed circuit techniques. Conventional coincident current operation is obtainable by having these conductors run parallel to each other in the proximity of each element so that the effects of the currents are algebraically additive. The printed windings can be placed on both sides of the glass to minimize current requirements and the inductance of the drive lines. Because the read output signal is relatively low, the sense winding is placed close for maximum coupling, and perpendicular to the drive conductors to minimize noise pick-up. Instead of thin glass or glass epoxy sheets to carry the printed circuit wiring, it may be possible to vacuum deposit the necessary insulation between the lines as well as the lines themselves.

Because the flux return paths are through air, there must be adequate shielding from external fields. Delta flux effects, which occur as the result of a small rotation of partially selected elements, can be cancelled by positioning the sense windings at the proper angle to the preferred direction of magnetization. The major noise problems result from unbalanced mutual coupling between drive lines and sense wires and capacitive coupling between a selected drive line and sense line. The former problem results from random errors in registration of the etched circuit wiring. However, present printed circuit wiring techniques are adequate to reduce this source of error to the point where strobing of the output signal reduces both noise contributions to the point where over-all signal-to-noise ratio is adequate.

As noted earlier, the squareness of the hysteresis loop depends, within limits, on a large ratio of diameter to thickness of the film. At the same time, the read-out signal is proportional to both thickness and diameter. Thus, one can do some trading of storage density for magnitude of signal output. Of course, a great reduction in spot size increases the problems of wiring. In fact, storage has been demonstrated at densities of 10,000 per square inch with 0.005 in. spots, but the utilization of such high densities depends on the development of practical wiring techniques in addition to solution of other problems.

An experimental memory composed of thin film elements is in operation as part of the TX-2 computer at the Lincoln Laboratory of MIT.

Each element is a circular deposit of Permalloy (82% nickel, 18% iron) film, 1.6 mm in diameter and 750 Å thick. There is a 2.5 mm separation between centers of these elements which are on a flat glass substrate 0.1 mm thick. While successful operation has been demonstrated with a read-write cycle time of only 0.4 μsec, the experimental unit is operated with an 0.8 μsec cycle which is consistent with the speed of the arithmetic unit. The writing current required is about 150 ma. and the output signal is about 1 mv.

The magnetic film array offers a switching time about ten times faster than ferrite cores (although a good part of this gain is offset by delays in the additional amplifier circuits required for its low level output signals), simple fabrication of large arrays, greater economy resulting from lower power dissipation and simpler fabrication, and operation over a wider range of temperatures than ferrite cores. An evaluation of these advantages in the light of problems such as production of uniform elements, reliability, and cost awaits further development.

5.3.7. Superconductive Element Storage

In Section 4.6, there is a description of a superconductive circuit element, the cryotron, which may be either wire wound or vacuum deposited. However, as stated in Section 4.6, thin film devices are required for greater speed and ease of fabrication. A coincident current circuit can be formed by having two or more control conductors on an element so oriented that the net magnetic field that acts on the gate is due to the sum of the currents on the drive lines. A storage array can then be formed by placing cryotron flip-flops at the intersection of row and column drive lines. Principally because of fabrication difficulties, the use of this type of memory has been restricted thus far.

Subsequent to the introduction of the cryotron, the use of a persistent current in a superconducting ring was suggested as a memory device. Such currents can be maintained for years, and the two possible directions of current offer the two stable states of a binary storage element. A persistent supercurrent element reported by IBM is based on the principle of trapping flux in a superconducting film. In an experimental form, a thin superconducting film formed by vacuum deposition, serves as a flux barrier between drive windings on one side and a sense winding on the other when the applied drive current is below a critical value. Below this value, it is presumed that the flux lines are forced along the film surface, inducing circulating currents. Above the critical value, the film is forced into a normal conductive state, permitting flux lines to penetrate and link a part of the film. The induced currents that opposed the field now decay,

and after removal of the drive current the flux is apparently trapped in imperfections in the film.

A form of persistent current device which has been brought to an advanced state of development by IBM is shown in Fig. 5.21(a). In this form, two circulating rings of current are formed by means of the hole and crossbar, as shown in Fig. 5.21(b). A principal advantage of this

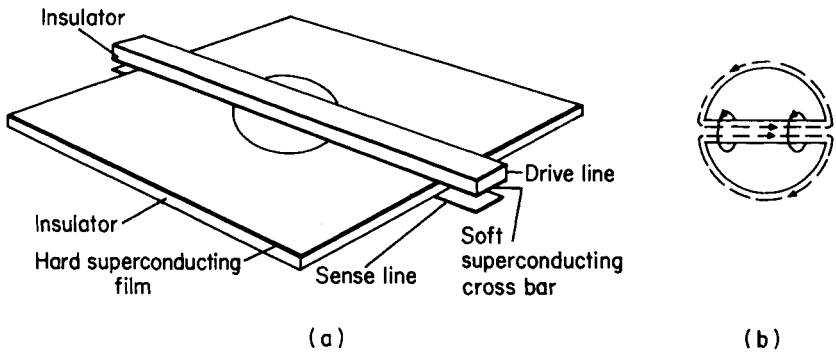


FIG. 5.21. Persistent-supercurrent thin film storage cell: (a) physical form, (b) path of circulating currents

construction is that the magnetic field of the persistent current is greatly localized. Also, it provides the features of the experimental cell while enabling better control over fabrication and the important parameters. The hole (about 3mm diameter) which need not be round, serves the function of the imperfections in the experimental cell. The over-all configuration is designed to trap flux in a doughnut pattern. A "hard" superconducting film (i.e., one having a high critical field) of lead supports and is in electrical contact with a thin crossbar of "soft" (i.e., low critical field) superconductive film. The crossbar (800 Å thick by 0.125 mm wide) is relatively "soft" because it is thinner than the sheet. The drive and sense lines, which are also strips of deposited lead, are placed parallel to the crossbar, the drive line above and the sense line below the sheet. Lines of force produced by current in the drive line are tightly coupled to the crossbar. The separation of the drive and sense lines by the superconducting film eliminates the delta noise problem of core memories (produced by the difference between half-select voltages), provided the crossbar is not driven to normal conduction, and the drive line, crossbar, and sense line are accurately aligned. The rise time, which is determined by the rate of transition from the superconducting to normal state and the fall time, determined by the L/R ratio of the cell, have an

upper limit of about 10 nanosec. The drive current is less than 150 ma. Selection can be by coincident current or other techniques. An array of memory elements including X, Y, and Z drive lines and sense line can be constructed as a unit from multiple layers of evaporated materials.

The advantages offered by this type of cell are: a sharp switching threshold, a high switching speed, high signal-to-noise ratios since the switching time of the cells is much less than the rise time of the drivers, low drive requirements, isolation of drive and sense circuits, and non-destructive read-out. However, a number of problems remain to be solved: (1) Additional data is necessary on the physical properties of thin films and the mechanism by which they are formed. Multilayer fabrication techniques must be developed to provide adequate control of the important parameters. (2) In the area of improving storage density, it is likely that the hole diameter can be reduced to 1mm (allowing about 100 cells/in.^{2}) and that about 20 layers of cells can be provided per inch—thus allowing from 10^6 to 10^7 bits/ft.^{3}. However, the problems of assembling so great a number of elements in a small volume are considerable. (3) There are the problems of operation in a liquid helium environment. Helium is evaporated as a result of the energy dissipated in switching the cells and also as a result of heat conduction along the wires from the cryostat to other parts of the computer. Actually a great number of switching operations and a large number of leads can be tolerated with a moderate consumption of helium; the switching of 10^{10} cells/sec. and the conduction of heat along 10,000 3-ft. leads each dissipates 1 watt for a total evaporation of 4 litres of helium per hour.

A continuous plane type of superconductive film memory (Burns *et al* [1961] is relatively easy to fabricate because no holes are required in the superconducting plane and its shielding action allows X, Y selection schemes without delta noise. Because memory cells, switching elements and connections can all be produced simultaneously by batch fabrication, very high capacity (10 to 100 million bits) memories with cycle times in the region of 5 μsec may be economically producible.

5.4. Tunnel-Diode Storage

The tunnel diode (see Section 4.7 also) is a unique device in that it exhibits a negative resistance characteristic in the gigacycle area. Two unconditionally stable states, A and B, established on a load line by a voltage source V_B and a series resistor R are shown in Fig. 5.22 (a). Switching between these stable states can be effected by a single pulse or coincident pulses of proper polarity and amplitude. For example, switching from state A to B can be effected by a total increment of current ΔI.

If the equilibrium state of a tunnel diode in the circuit of Fig. 5.22 (a)

is sensed by applying a large voltage or current pulse, the readout process will be destructive since the initial state is inferred from whether a switching action occurs. In Fig. 5.22 (b), a nonlinear backward diode (whose characteristic curve is the dashed line in Fig. 5.22 (a)) is used to allow nondestructive readout. Fig. 5.22 (c) illustrates a word organized array of such circuits. To read, a negative pulse is applied to the word line only, while the digit lines are held near a zero reference potential. If a tunnel diode is at *A*, the back bias across diode *D* is overcome (although current through the tunnel diode is not adequate to switch it) and a negative pulse appears at point *d*. If a tunnel diode is initially at *B*, there is no output at *d*. (Use of backward diodes to couple tunnel diodes to sense amplifiers also reduces attenuation of the readout signal.) Writing can be done by coincident word and digit pulses.

While magnetic memories may be extended to cycle times of 1 μsec to 100 nanosec, cycle times of 100 to 10 nanosec are obtainable from a tunnel diode memory. There are certain limitations, however. It is inherently volatile, requiring dc holding power. The drivers must supply large power for short periods, although the drive power per bit (about 2 mw) is small and the drive circuits simple compared to requirements of other storage devices operating at these high rates. Present estimates are that considerations of power consumption, drive circuitry, reliability and cost will limit the useful capacities of tunnel diode memories to about 4000 to 100,000 bits.

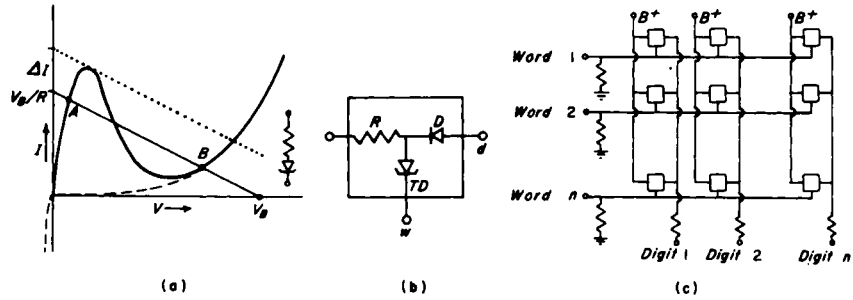


FIG. 5.22. (a) Tunnel-diode characteristic curve, (b) three-element memory cell, (c) linear array of three-element cells.

5.5. Cathode-Ray Tube Storage

There are a number of storage systems in which binary data is stored in the form of the presence or absence of a specified amount of electrical charge on the face of a conventional type of cathode-ray tube. A 5-inch cathode-ray tube, the size normally used, accommodates a 32 × 32 array of spots. These tubes provided an interim solution to the high speed internal

storage problem, providing an access time of 12 to 25 μsec. However, they are no longer being incorporated into new systems, having been completely displaced by magnetic core storage systems.

In all cathode-ray tube storage systems, two principles were commonly used to retain the stored data for any desired time interval, namely: (1) direct use of the secondary emission characteristics of the dielectric storage surface, (2) regeneration of each bit after read-out. The Williams storage system, also known as the interfering periphery or surface redistribution system, is the form of cathode-ray tube storage most widely used and is described next.

The Williams system utilizes a commercial type cathode-ray tube, together with a metallic collector screen or plate placed over the outside face of the tube, as indicated schematically in Fig. 5.23. If the inner screen

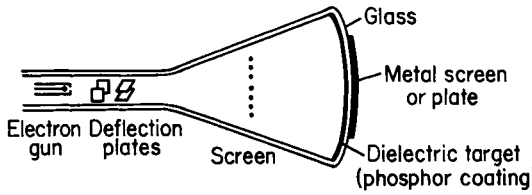


FIG. 5.23. Simplified schematic of cathode-ray storage tube

is bombarded by an electron beam, secondary electrons will be emitted. The variation of the secondary emission coefficient, γ, (for a P_1 type of phosphor screen) as a function of the potential of the surface is shown in Fig. 5.24. There is a drop in γ at point b because of the increased

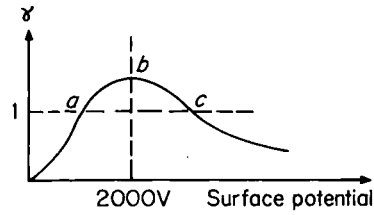


FIG. 5.24. Variation of secondary emission coefficient, γ, of a P_1 type of phosphor surface

ability of the surface at higher potentials to recapture electrons liberated from the surface by the incident beam. If the accelerating potential is sufficient to produce a ratio greater than 1, a potential well will be formed at some point on the screen, as indicated in Fig. 5.25. The slight negative

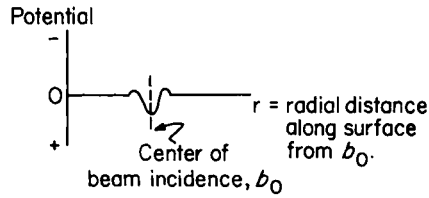


FIG. 5.25. Potential well formed by $\gamma > 1$

potential mounds are caused by some of the secondary electrons landing in the vicinity, and are useful in serving as barriers which reduce interaction between wells. The size of the well is approximately proportional to the diameter of the beam. If the beam is removed after producing a well, the potential distribution will remain for some time. If a metal screen is placed near the target, a small signal will be detected at the time this well is formed. This is essentially capacitive pick-up, the target and outer screen acting as plates of a capacitor and the screen reproducing the potential variations of the target.

For binary storage, two different patterns must be stored, one of which is used as a reading pattern. If the reading pattern is different from the stored pattern, the stored pattern will be lost and changed to the reading pattern. This change produces a total change of charge on the tube face, which is detected by the capacitive action of the external screen. Three of the several storage pattern systems based on this mode of operation are described next.

(1) The interrupted double spot system operates as follows: If, after a well is dug, the electron beam is cut off and displaced slightly, the distribution of Fig. 5.26(a) results. Since the second

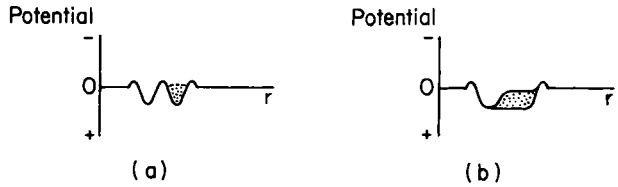


FIG. 5.26. Potential wells formed for storage of a 1 in (a) interrupted double spot, and (b) dot-dash storage schemes

well is formed in the vicinity of the first, secondary electrons emitted from the second well will tend to fill up the first well. This forms the basis for a binary storage system. The first well is formed for each entry. If a 1 is to be stored, a second well

is formed; if a 0 is to be stored, a second well is not formed. Read-out is accomplished as follows: The first well position is interrogated by the beam. If a second well had been formed, (i.e., a 1 stored), it would have caused filling of the first, and the interrogating beam would remove this fill, thereby producing an output signal. If a 0 had been stored, the interrogating beam would produce no filling and, therefore, no output signal.

(2) The dot-dash system is similar except for the fact that the beam is not interrupted during movement. When the beam is slightly displaced, the secondary electrons emitted from the new position partially cancel the positive charge of the first spot, as shown in Fig. 5.26(b). When the interrogating beam is directed to the first position, a large negative charging signal, or a smaller one, will be detected at the external plate, depending on whether a 0 or 1, respectively, is stored.

(3) The focus–defocus system differs from the interrupted double spot system in that the beam is left stationary and defocused after the original well is formed if a 1 is to be stored. The enlargement of the beam caused by defocusing causes electrons to be scattered over a wide area, filling the original deep well.

Because of the destructive nature of the read process, means must be provided to restore the contents of a storage cell after it is read, and this restoring operation is made a part of the storage cycle. Also, because of the volatility of the stored data (due to charge leakage), circuits are provided that scan and regenerate the contents of the raster, row by row. This regeneration is accomplished during specified cycles set aside for this purpose.

Specially constructed tubes are usually used for two important reasons. First, the beam diameter intercepting all storage positions must be very small or else there will be too much interaction between adjacent cells. Secondly, in tubes not specifically designed for computer applications, blemishes on the target surface may result in the presence of areas with relatively poor secondary emission characteristics.

An important parameter of cathode-ray tube storage systems is the "read-around" ratio. It is the number of times that a given cell can be interrogated in succession without destroying the contents of an adjacent cell. It determines the frequency at which the contents of all cells must be regenerated. A high "read-around" ratio is desirable to reduce the ratio of time spent in regenerating to the time spent in computing. However, improvement in the "read-around" ratio is obtained at the cost of reducing the storage capacity of a given tube.

A number of special forms of cathode-ray tubes were also developed

for use in digital computer storage systems: the barrier grid storage tube, the holding gun tube, and the selectron. None of these ever came into widespread use, however, for two principal reasons: a relatively high cost of manufacture and the introduction of magnetic core storage systems, which supplanted all types of cathode-ray tube storage systems. Therefore, they are not described here, but are referred to in the bibliography of this chapter.

5.6. Dynamic Delay Line Storage

In Section 5.2, Dynamic Magnetic Storage, it was shown how a recirculating delay line could be formed by directing information from the output of a read head to the input of a record head placed along the same track. In such a delay line each bit is recorded in a particular area of the medium and the medium itself is moved. In the delay lines to be described in this section, the pulses to be stored are propagated from a transmitter to a receiver via a stationary medium. Since it takes time to propagate the energy, the pulses in transit between the source and receiver can be considered to be stored in the medium. Delay line stores have been built in which the propagation of energy may be by means of electrical, acoustic, electromagnetic, piezoelectric, or magnetrostrictive phenomena. By repetitively reintroducing the signals into the delay medium, in synchronism with a time reference pulse, the temporal location of a particular item of information can always be specified, assuming the delay is held constant. Because any delay medium has limited bandwidth characteristics, causing distortion of the propagated pulses, circuitry must be provided to reshape the pulses and preserve the proper time relationships. The pulses are amplified, gated, and reintroduced into the delay media by means of these circuits.

The operation of a delay line as a dynamic information storing device is as follows: A temporal serial binary information pattern (pulse–no pulse) is fed into one end of a path consisting principally of the delay element. As a result, the temporal serial also becomes a spatial serial pattern (if one could inspect the entire contents of the line at any given time, and if the line had a capacity of n bits, one would see the last n bits that had been inserted into the line). Recirculation of information patterns is provided by closing the loop, from the end of the delay line back to the beginning, by means of the transducers, amplifiers, and gating circuits.

An important criterion of delay lines for use as a large capacity store is the amount of delay provided by a unit length of path. The rate of propagation in the delay medium should be sufficiently slow to allow

a large number of pulses of the input information to be stored in a physically practical length of line. The memory capacity of such lines is proportional to the length of line and the repetition rate of the applied signals. Some disadvantages of the electric, acoustic and magnetostrictive delay line stores are an access time that increases with the length of a line, and duplication of circuits required when information is stored in several lines to obtain a certain combination of capacity and access time.

5.6.1. Electrical Delay Lines

An electrical transmission line formed from either lumped or distributed elements of inductance and capacitance may be used as a delay line. The number of bits that can be stored in the line depends not only on the magnitude of the delay and the repetition rate of the applied pulses, but also on the rise time characteristic of the line. This is because an amount of delay at least equal to the sum of the rise and fall times of a pulse is required for each pulse to be stored, and the rise time characteristic of any type of line is proportional to its length. Therefore, increasing (decreasing) the length of the line proportionately increases (decreases) the amount of delay so that the number of pulses that can be stored remains essentially constant. The maximum number of pulses that can be stored in available lumped and distributed constant lines is about 30 and 15, respectively. The length of line chosen would be determined by the waveform of the pulses to be stored. Because the rate of propagation along an electrical transmission line is high, a considerable length of line is required to obtain milliseconds of delay. Therefore, electrical lines are not practical except for short delays, e.g., less than 50 μsec. These are suitable as one-word registers in dynamic serial-type computers operating at megacycle repetition rates, but are inefficient for a memory of large capacity.

Electrical delay lines greatly attenuate the input signal but introduce only negligible losses at the input or output. Conversely, acoustic delay lines, which are described next, produce little attenuation in the medium, but a large amount of attenuation is introduced in the coupling between the line and the input and output transducers.

5.6.2. Acoustic Delay Lines

The relatively slow velocity of propagation of an acoustic wave compared to an electric wave permits a greater delay to be obtained from a physically short line. Radiofrequency signals are transformed by an appropriate transducer into an acoustic signal. The acoustic wave is then propagated through an appropriate medium until it reaches a receiving

transducer which converts the acoustic signal back to an electrical one. In an acoustic delay line storage system the delay medium of Fig. 5.27

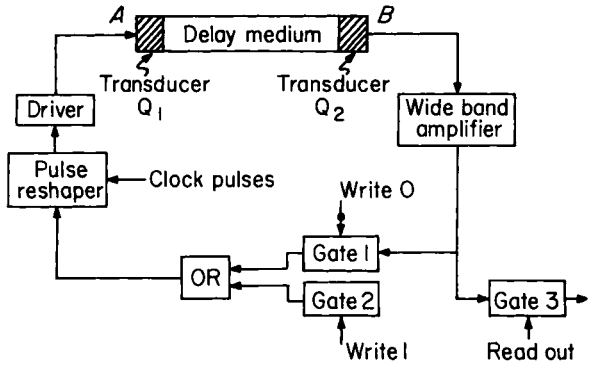


FIG. 5.27. Block diagram of an acoustic delay line store

would be an acoustic transmission line terminated at each end by a piezoelectric crystal. An electrical signal at A causes vibration of the crystal Q_1, and this disturbance is propagated down the acoustic line setting the crystal Q_2 into vibration, thereby generating an electrical signal at point B. Pulses received at Q_2 have to be reshaped because attenuation and dispersion, caused by transmission through the line, broadens and otherwise distorts the original pulse applied at Q_1. Regeneration is accomplished by using the amplified distorted signal to gate a clock pulse which is then recirculated instead of the distorted signal. New data is introduced into the line as follows: If a 1 is to be recorded, it is sent directly to the input of gate 2; if there is already a 1 recorded in this position (i.e., a signal is arriving from the amplifier via gate 1), the operation is merely redundant. If a 0 is to be recorded, gate 1 is inhibited. Information can be read out at any time by application of a read-out gating signal to gate 3.

Important parameters of an acoustic medium are: velocity of propagation, acoustic impedance, temperature coefficient of velocity, attenuation/unit length, signal frequency for which attenuation is a minimum, bandwidth characteristics, and phase distortion. Quartz, even though not the most efficient electric-acoustic transducer, is used because its acoustic impedance comes closest to matching the available media, and it has a small value of capacitance, thereby reducing the driving power required. For a quartz transducer, the most desirable acoustic media are (in order):

(1) zero temperature coefficient glass, (2) impurity-free fused quartz, (3) homogeneous magnesium alloys, (4) impurity-free mercury.

5.6.2.1. The Mercury Delay Line

One of the most widely used acoustic delay lines in the very early years of digital computer development consisted of a mercury transmission medium and quartz crystal transducers. For example, the BINAC had a tank of 18 separate delay lines, each storing 32 words of 36 bits, and the first UNIVAC had 100 delay lines, each storing 10 words of 91 bits. Mercury has a relatively low velocity of propagation, a good acoustic impedance match to quartz and freedom from unwanted modes of sound transmission found in solid media.

Mercury delay lines may be of two general types. In the single path type, the acoustic signal traverses the medium once, from transmitting crystal to receiving crystal. In the multiple reflection type the signal is reflected back and forth through the medium before reaching the receiving crystal, thereby effectively lengthening the delay path. A disadvantage of the latter system is that it requires critically machined parts.

The factors that limit the length of delay line used are: (1) The over-all attenuation and wave front dispersion that can be tolerated before the signal to noise (including reflections) ratio falls below a usable level (usually 10). (2) The access time required in the computer. The majority of delay lines have a delay of about 350 μsec. The limitation on the pulse repetition rate that may be used is not the acoustic line, but the electronic circuitry. Most computers using mercury delay line stores operate in the range of 1–4 mc.

There are many problems associated with a mercury delay line: (1) Unless the mercury is free from all contamination, a mismatch results at the crystal surface causing serious reflections. Long time stability is also a problem since mercury is an almost universal solvent. Best results are obtained from triple distilled mercury, and containers of borosilicate glass or stainless steel. (2) The quartz transducers must make extremely good contact with the mercury. The voltage output of the line is proportional to the deformation of the crystals, and air bubbles or contamination on the crystal surfaces would damp this vibration, resulting in a very large over-all loss of energy. The normal attenuation at each transducer is about 25 db. This accounts for most of the loss in the line, the attenuation through the mercury being only a few decibels per foot. (3) The mercury tank requires temperature control, since the velocity of propagation of an acoustic wave is directly proportional to the density of the medium which, in turn, is a function of temperature. These tempera-

ture regulating systems are fairly complex, consume a good deal of electronic circuitry and space, and add appreciably to the expense of a system. There are also problems associated with reflections in the lines, and with the external adjustments required to produce the desired total delay. A mercury delay line is thermally unstable and susceptible to mechanical shock, leakage, and contamination.

Mercury provides the following advantages compared to solid transmission media: (1) It is a stable liquid which can be matched to transducers such as quartz crystals. Solids generally present problems in coupling to suitable transducers. (2) It supports only longitudinal-compression and surface waves, the latter being suppressed when the liquid is enclosed in a tank. Solids generally support shear waves. (3) Careful distillation provides uniform characteristics. There are no localized strains to cause spurious patterns.

5.6.2.2. The Fused Quartz Delay Line

A delay line can be constructed using a rod of fused quartz as the delay medium, and quartz crystals as the transmitting and receiving transducers. Though both shear and compressional waves can be transmitted through the medium, the shear mode is preferable because its rate of propagation is less, thereby yielding a longer time delay for a given length of material. However, even in the shear mode, the velocity of propagation through fused quartz is almost three times as great as in mercury, and rods long enough for large delays are impractical. However, a long delay can be obtained in a practical form by use of a shape which causes the transmitted wave to be internally reflected several times before it reaches the receiving transducer. The multiple reflection line consists of a flat plate whose sides form a polygon. A typical plate might be formed from a fused quartz blank about 8 in. in diameter and 1/2 in. thick. The lengths and angles of the sides of the polygon are chosen to provide the desired multiple reflection path from the input transducer, which is bonded to one side, to the output transducer bonded to another side. A delay of about 1000 μsec can be obtained from an 8-in. diameter blank. The production of the multiple reflection line is a precision process that includes machining, grinding, and polishing of the facets. The blank must be of high purity to eliminate spurious reflections from air bubbles and foreign particles. The input and output transducers may be either ac or Y cut quartz crystals.

The fused quartz line is superior to the mercury line in that it has a higher signal to noise ratio, a temperature coefficient of velocity only about one third that of mercury, a better impedance match to quartz crystal

transducers, and better mechanical stability. On the other hand, it is difficult to produce high purity blanks, and the reflecting surfaces in multiple reflection lines require high precision machining.

Fused quartz has the lowest attenuation figure of solid materials suitable at high frequencies. In the shear mode of transmission the attenuation is about .08 db/ft per megacycle and in the longitudinal mode it is .05 db/ft per megacycle. This compares to about 1.8 db/ft at 15 megacycles for mercury in a .3 cm inner diameter tube (based on the expression for attenuation in a tube of mercury; .054 f/d*, where f is the frequency in megacycles and d the inner diameter of the tube in inches). The shear mode of transmission is the one principally used in quartz delay lines because it is not subject to mode conversion upon reflection (which can cause spurious pulses) and beam spreading is less, yielding an improved signal-to-noise ratio. The velocity at 20°C for fused quartz is about 150 inches/msec in the shear mode (233 in the longitudinal mode) compared to 57 inches/msec for mercury.

5.6.3 Magnetostrictive Delay Lines

The magnetostrictive delay line is based upon the magnetostrictive effect exhibited by certain materials. When a magnetizing force is applied to such a material, it exhibits a change in length. The reverse effect also occurs, i.e., a stress applied to such a material produces a change in its magnetic state. (Specifically, when tension is applied to a wire, the ferromagnetic domains align themselves in a direction away from the wire axis.) This change may be detected by the change in magnetization that occurs when a magnetic field is applied. The delay is obtained from the time required to propagate the stress wave disturbance along a length of wire, rod, or similar configuration of a material possessing magnetostrictive properties and having a high remanence. The amount of delay depends on the length and physical properties of the material.

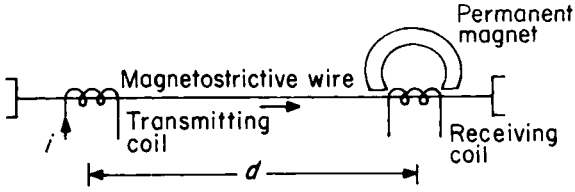


Fig. 5.28. Schematic of a magnetostrictive delay line

* Blackburn, J. F. (ed.) [1949] M.I.T. Radiation Lab. Series, Vol. 17, McGraw-Hill, New York. (Also, see Huntington *et al.* [1948].)

A schematic of a magnetostrictive delay line is shown in Fig. 5.28. When a pulse of current is applied to the transmitting coil, a contraction of the wire immediately inside the coil takes place. This causes a stress wave to be propagated along the line in both directions. To reduce reflections at both ends, damping pads of a suitable material, such as synthetic rubber, are clamped about the wire at both ends. Also, the parts of the line between each coil and the nearest end are annealed to provide added attenuation. At the vicinity of the receiving coil, R, there is a remanent magnetic dipole as the result of a polarizing field induced by a permanent magnet. The arrival of the stress wave at the region of R changes the permeability of the wire, momentarily disturbing the induced dipoles and the field cutting R. Thus, after a given delay, a voltage pulse is induced in the receiving coil as the result of a current pulse being applied to the transmitting coil.

The input current step produces a pulse in the output coil at the times of its leading and trailing edges, as shown in Fig. 5.29(a). The

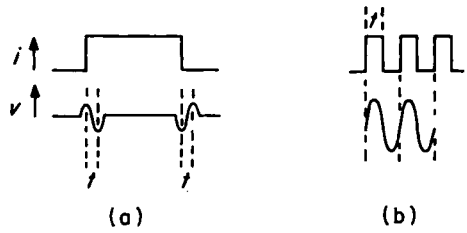


FIG. 5.29. Resolution of output voltage pulses as a function of width of the input waveform in a magnetostrictive delay line

time interval, t, between the positive and negative peaks of this pulse is determined by the geometry of the coil. To obtain a symmetrical voltage output pulse and a maximum packing of bits, the width of the input current pulse is made equal to t. The length of the transducer coils must be accurately determined to match the input pulse. If low levels of current and voltage are to be employed, the efficiency of the transducers should be high. It can be improved by the use of ferrite core materials to direct all flux into the wire and to sharply define the edges of the field.

The stress wave may be transmitted either in a longitudinal or torsional mode. On lines of appreciable length (say, greater than one foot) the wire is coiled to provide convenient packaging. For coiled lines of appreciable length, transmission in the torsional mode offers the advantages of a substantially reduced velocity of propagation and reduced dispersion. A mode conversion can be made near the entry and end-points of the

line, and the line between need not be of a magnetostrictive material. By the use of other materials, such as copper or phosphor bronze, in which the velocity of propagation is less than in nickel, the length of line for a given delay may be reduced. When nonferromagnetic materials are used either for a better velocity of propagation, temperature coefficient of velocity, etc., magnetic end pieces must be provided. This can be done either by brazing short pieces of nickel to the ends of the line, or by nickel plating a short length of the wire at each end. Taps may be provided as required on lines using the longitudinal mode with negligible attenuation. At present, lines using the torsional mode can be tapped only in the short longitudinal mode section at each end.

Figure 5.30 shows a block diagram of the circuitry used to regenerate

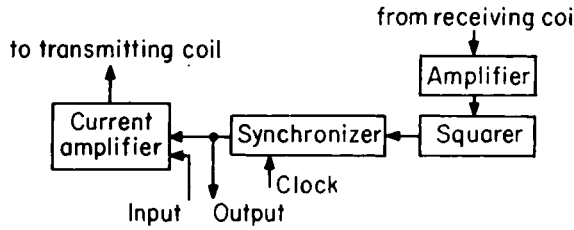


FIG. 5.30. Regenerative loop for a magnetostrictive delay line

and retime the output pulses each time they circulate through the delay line. The squaring circuit produces an output of fixed amplitude independent, within limits, of the input amplitude, and also biases off spurious signals due to unavoidable reflections from the terminations of the delay line. The synchronizer causes the recirculating pulses to be kept in synchronism with a master clock. The delay of the line is adjusted so that data pulses arrive back midway between two adjacent clock pulses. This provides a safety margin of half a bit period in either direction to allow for variation of delay with temperature, and also for variation of clock frequency. The current amplifier converts the synchronized voltage pulses into the current pulses required at the input coil.

Read-out coils may be placed at appropriate positions along the line. For computer storage, a number of delay lines can be assembled to provide either serial or parallel operation. For serial operation, all the bits of a word would be stored on a single delay line. For parallel operation, each bit of a word would be stored at corresponding time positions on different lines.

Some typical parameters of a magnetostrictive delay line are minimum

bit rates of 100 kc and maximum bit rates of 500 kc to 1 mc. The L40 delay line package of Ferranti Electric, Inc. provides a maximum delay of about 5000 μsec which, at a bit rate of 500 kc will store about 2500 bits.

5.7. Diode-Capacitor Storage

A 10,000 bit diode-capacitor memory built at the National Bureau of Standards for its SEAC computer stores binary information as the charge on a capacitor. The state of each element is defined by the sign of its charge. This system has a relatively high random access rate compared to acoustic delay line and cathode-ray tube stores. The basic circuit for a storage element of the array is shown in the dashed enclosure of Fig. 5.31(a). The points O_1, O_2, . . . O_n are used for both reading and recording. The two diodes associated with each storage capacitor act as a squeezer, connecting the capacitor to the read or record circuit when one of these operations is called for. During holding, i.e., between read and record operations, the two diodes are each back-biased so that only minute currents can flow into or out of the capacitor.

The contents of a particular word, i, are read as follows: Points, a_i, b_i are forced to ground potential. As a result, one diode in each of the n pairs within a word conducts, producing voltages across the associated resistors. If the initial charge on a capacitor produced a drop of -2 volts across it, when the squeeze is applied a pulse of -2 volts amplitude appears at the corresponding output point, decaying with a time constant RC. If the capacitor had been charged to $+2$ volts, a pulse of $+2$ volts would appear at the output. The positive and negative pulses are interpreted as the values 1 and 0, respectively. Regeneration is required after a read-out operation, because the capacitor is partially discharged.

To record in a particular word, each point O_1, O_2 . . . O_n is forced to the desired voltage, either $+2$ or -2 volts, while the diodes are being squeezed, i.e., while the diode bus lines are at zero potential. When the diodes are returned to their normal voltages, +4 and −4 volts, each presents a high impedance. Therefore, the charge on the capacitor cannot readily leak off, and will be unaffected by later changes at the corresponding output point, provided the absolute magnitude of the voltage at the output point does not exceed 2 volts.

A gating amplifier is needed at each output point to sense the polarity there during read-out, and to force O_i to the desired polarity during a recording operation. As shown in Fig. 5.31(a), a single gating amplifier serves the same bit on each of many words. A particular word is selected by squeezing the proper pair of buses to zero voltage, while holding all the other pairs at their normal value of $+4$ and -4 volts. Each bit of a

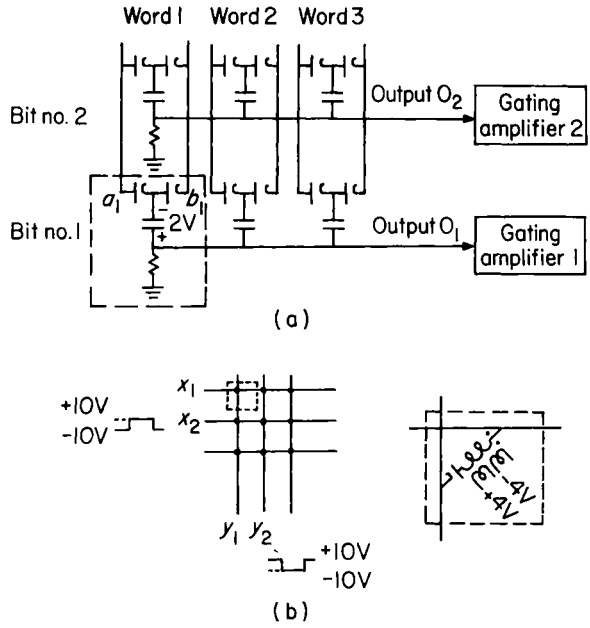


FIG. 5.31. Diode capacitor storage array, (a), and transformer AND gate selection matrix, (b)

word is accessible simultaneously at the gating amplifiers. A write operation in a particular word does not disturb the others since all diodes in all other words remain backward biased.

A selection matrix suitable for use with the diode-capacitor store is shown in Fig. 5.31(b). Its advantage here over a diode matrix is that it does not draw a large amount of standby power. However, it does require more input drivers than does a diode matrix. Normally the X and Y buses are held at -10 and $+10$ volts, respectively. This puts a backward bias on each diode so effectively no current flows through any transformer primary. If either one X bus is raised to $+10$ volts, or one Y bus is lowered to -10 volts, there would still be no current flow. However, if each of these operations is initiated simultaneously, one transformer at the intersection of the two buses will conduct. As a result, the transformer secondaries will apply a squeeze to the buses in the selected word.

The finite forward conductance of the diodes reduces the amplitude of the output pulse, and increases the time to charge the capacitor adequately during recording. However, the effect of finite diode back resistance is critical. During the holding operation, relatively long times will elapse, and even minute currents through the diodes could result in no

charge being left on a capacitor, or even a reversal in the sign of the charge. Thus the diode back resistance determines the maximum permissible time interval for a holding operation. To maintain the stored information, regeneration must be provided at periods less than this interval. This regeneration cycle is controlled by the memory control circuits which cause the contents of each cell to be read and re-recorded.

Though the cycle time of the NBS memory using junction diodes is 10 μsec, Kaufman [1959] describes a memory used to assess the difficulty of reducing cycle time for a 1000 bit plane to 10 nanosec, by using extremely fast diodes.

5.8. Ferroelectric Storage

Ferroelectrics are crystalline materials which have a permanent electric dipole moment, and in which the plot of polarization produced by varying the intensity of an applied electric field, exhibits a hysteresis loop similar to that of a ferromagnetic material. This hysteresis occurs because the dipole moment is reversible. Corresponding to the saturation and remanence of magnetic induction or flux density in a ferromagnetic material is the saturation and remanence of electric charge in a ferroelectric material. The use of the term ferroelectric does not imply that such materials contain any iron, but only that they are analogous, in the way described, to ferromagnetic materials. Among the ferroelectric materials are barium titanate, triglycine sulphate, potassium dihydrogen phosphate, and Rochelle salt. Single crystals of a ferroelectric show a particularly square hysteresis loop, as indicated in Fig. 5.32(a) which shows the shape of the hysteresis loop for barium titanate using a 50 $\sim$ voltage. Barium titanate has been investigated most extensively because of its relatively short switching time (about 1 μsec), high saturation polarization and high Curie temperature of 120°C which allows operation over a wide practical temperature range.

The construction of a ferroelectric memory cell, shown schematically in Fig. 5.32(b) is similar to that of a capacitor. However, as indicated in Fig. 5.32(a), the ferroelectric material exhibits an almost square hysteresis characteristic instead of the linear relationship between voltage and charge that exists over the operating range of a capacitor. If a positive field intensity greater than E_1 is applied, and then reduced to zero, a charge $+P_r$ remains. Similarly, if a negative field intensity greater than $-E_1$ is applied and then reduced to zero, a charge $-P_r$ remains. The applied electric field intensity is equal to the applied voltage, v, divided by the crystal thickness. The dynamic capacitance, C_f, of the ferroelectric storage element is equal to the ratio of change in polarization per unit volume to the change in the applied field. The capacitance can be shown

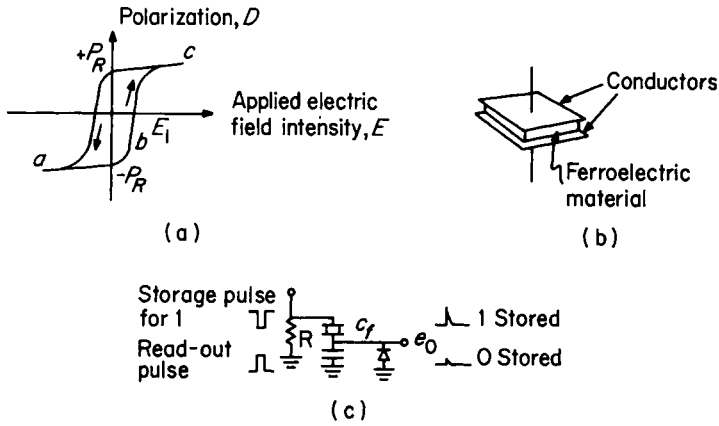


FIG. 5.32. Ferroelectric storage, (a) polarization hysteresis curve, (b) storage cell, (c) read-out circuit

to be equal to the area of the ferroelectric crystal used times dP/dV.

The storage element may be considered as a device with two stable states, corresponding to $+P_r$ and $-P_r$. If it is in state $+P_r$ and a positive pulse of voltage is applied, the peak output voltage will be small since the change in polarization from $+P_r$ to point c is small. A typical read-out circuit is shown in Fig. 5.32(c). Its output capacitance is chosen so that it has a relatively high capacitance compared to C_f, so a low output voltage can also be explained by the fact that the capacitance, C_f, of the storage element is low between $+P_r$ and point c. If the storage element is in state $-P_r$ and a positive pulse is applied, the output voltage will be much higher since the change in polarization from state $-P_r$ to c is large; also the capacitance of the storage element between $-P_r$ and point c is much larger than the capacitance of the output capacitor. A 1 is stored by the application of a negative pulse at the input terminal. Read-out is obtained by means of a positive voltage pulse. If the cell contains a 1, i.e., is at state, $-P_r$, a large charging current passes through R producing a large output signal as shown. If the cell contains a 0, i.e., is at state $+P_r$, application of the read-out pulse produces a small charging current through R and a negligible output signal. The rectifier prevents storage pulses from appearing at the output.

Because several independent sets of electrodes can be placed on the same crystal and satisfactory operation obtained without appreciable cross-talk between adjacent cells, a ferroelectric storage array can be constructed as shown in Fig. 5.33. On the two surfaces of a ferroelectric crystal, e.g.,

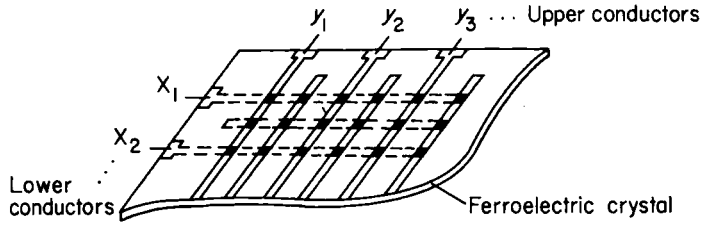


FIG. 5.33. Section of a ferroelectric storage array

barium titanate, conducting bars are vacuum deposited, the conductors on opposite sides running perpendicular to one another as shown. The conductor widths and spacings may range from 0.2 mm to 0.1 mm, and the crystal thickness is in the same range. A particular cell is selected by energizing the appropriate X and Y conductors. Switching can be accomplished with pulses of 10 volts amplitude and a few microseconds duration. Signal-to-noise ratios of about 10 to 1 can be obtained. A selection matrix for use with the memory can also be constructed from ferroelectric elements. A serious problem is that the writing of information into one part of the store causes partial voltages to be impressed upon unselected elements in the store. Even if the polarization of the unselected sections is only partially reversed by these voltages, the effect of many write operations is cumulative and may completely reverse unselected cells. One way of overcoming the lack of a definite coercive field is to use a diode at each cell to provide a bias which can only be overcome by simultaneous application of X and Y select voltages. However, this is an uneconomic procedure. Another system suggested is to deposit on one side of the crystal, before deposition of the electrodes, a semiconducting layer. This provides a nonlinear element at each cell in an economic way.

At high field strengths (about 5 kv/cm) required for short switching times, if an applied pulse does not cause switching, another pulse can be applied subsequently (as much as 10 min. later) to complete the switching. Also, there is no threshold field strength below which no switching occurs, so whatever field strength is applied, reversal will occur provided there is sufficient time for it. The variation of coercive field with speed of switching in single crystal barium titanate is evidenced by the fact that for a 50 cycle/sec hysteresis loop, the coercive field is about twice that for a 1 cycle/sec loop. The coercive field, as a function of frequency, increases up to the point where heating effects cause a decrease. This indicates the maximum frequency at which the crystal can be operated. For a practical cell volume (0.1 mm^2 by 0.1 mm thick) operation is attainable to about 100 kc before heating effects become noticeable.

The principal advantages of the ferroelectric memory are: (1) it is voltage operated requiring current only of the order of a few milliamps, and small power consumption; (2) it provides a good density of storage, a practical 20 × 20 array being obtainable from a crystal surface 1 in.^{2} and 0.004 in. thick with 0.004 in. conductor widths (100,000 bits/in.^{3} not counting volume for wiring, connections, etc.)

The major difficulties in the utilization of ferroelectric storage elements are the following: (1) With present techniques, it is difficult to grow large single crystals of good quality. This limits the amount of storage on a single crystal surface to about a 32 × 32 array. (2) Operation above a critical frequency causes a decrease in the coercive field as a result of heating effects, and may result in permanent damage to the ferroelectric properties. This limits the access time to about 10 μsec. The useful computing speed is limited further by the need for regeneration to prevent switching by the cumulative effect of successive partial disturb pulses.

Possibilities remain to be explored towards developing a material with a well defined coercive field. For example, impurities may be introduced into the crystal lattice in such a way that a permanently polarized surface layer is produced. In this case, the switching energy would be determined not by the energy to produce new domain walls, but to move existing ones. There is also the possibility that switching could be produced with lower energy by coherent rotation of the polarization if sufficiently small single domain particles were used. Another possible development is the use of thin evaporated films of ferroelectric material. Barium titanate has already been produced in this form.

LITERATURE

GENERAL REFERENCES

Bell, P. R., Forbes, G. D., and MacNichols, E. F., Jr. [1949] "Storage Tubes," in *Waveforms,* Radiation Laboratory Series No. 19, McGraw-Hill, New York.

Barkan, H. E. [1961] Advances in magnetism and magnetic materials, *Electro-Technology,* **67,** No. 2, 80-88

Begun, S. J. [1955] *Magnetic Recording,* Rinehart, New York.

Bozorth, R. M. [1956] *Ferromagnetism,* Van Nostrand, New York.

Clapp, L. C. [1961] High speed optical computers and quantum transition memory devices, *Proc. Western Joint Computer Conference,* 475-489.

Hershberg, P. I. [1962] Ferromagnetic domains, *Electro-Technology,* **69,** No. 1, 72-82.

Jaynes, E. T. [1953] *Ferroelectricity,* Princeton Univ. Press, Princeton, N. J.

Kittel, C. [1949] Physical theory of ferromagnetic domains, *Rvw. of Modern Physics,* **21,** 541-583.

Kittel, C. and Galt, J. K. [1956] Ferromagnetic domain theory, in *Solid State Physics,* **3,** Academic Press, New York.

Knoll, M. and Kazan, B. [1952] *Storage Tubes and Their Basic Principles,* Wiley, New York.

von Laue, M. [1952] *Theory of Superconductivity,* Academic Press, New York.

Rajchman, J. A. [1961] Computer memories—a survey of the state of the art, *Proc. IRE,* **49,** 104-127 (includes bibliography of 96 entries).

Shoenberg, D. [1952] *Superconductivity* (2nd ed.), Cambridge Univ. Press, Cambridge, England.

Staff of Cresap, McCormick and Paget [1962] Random access devices for medium to large computers, *Control Engrg.,* **9,** No. 4, 131-137.

Swanson, J. A. [1960] Physical versus logical coupling in memory systems, *IBM J. Research and Develop.* **4,** 305-310.

Several authors [1962] A collection of articles on superconductive phenomena, *IBM J. Research and Develop.* **6,** 3-125.

MAGNETIC HEAD DESIGN

Brower, D. F. [1955] A one turn magnetic reading and recording head for computer use, *IRE National Convention Record,* Pt. 4, 95-100.

Clark, D. L. and Merrill, L. L. [1947] Field measurements on magnetic recording heads, *Proc. IRE,* **35,** 1575.

Fan, G. J. [1961] A study of the playback process of a magnetic ring head, *IBM J. Research and Develop.,* **5,** 321-325.

Hoagland, A. S. [1956a] Magnetic recording head design, *Proc. Western Joint Computer Conference,* 26-31.

Hoagland, A. S. [1956b] Magnetic data recording theory: head design, *Trans. Amer. Inst. Elec. Engrs.,* Pt. I,, **75,** 506-512.

Hoagland, A. S. [1958] High resolution magnetic recording structures, *IBM J. Research and Develop.,* **2,** 91-104.

MAGNETIC RECORDING TECHNIQUES

Eadie, D. [1953] EDVAC drum memory phase system of magnetic recording, *Elec. Eng.,* **72,** 590-595.

Fuller, H. W., Husman, P. A., and Kelner, R. C. [1954] Techniques for increasing storage density of magnetic drum systems, *Proc. Eastern Joint Computer Conference,* 16-21.

Hoagland, A. S. [1955] A logical reading system for nonreturn-to-zero magnetic recording, *IRE Trans. El. Comp.,* **EC-4,** 93-95.

Hoagland, A. S. and Bacon, G. C. [1960] High density digital magnetic recording techniques, *IRE Trans. El. Comp.,* **9,** 2-11.

Jacoby, M. [1962] A critical study of mass storage devices and techniques with emphasis on design and criteria, *Proc. IRE-PGMIL National Winter Convention on Military Electronics,* 165-179.

Lubkin, S. [1954] An improved reading system for magnetically recorded digital data, *IRE Trans. El. Comp.,* **EC-3,** 22-25.

Miyata, J. J. and Hartel, R. R. [1959] The recording and reproduction of signals on magnetic medium using saturation type recording, *IRE Trans. El. Comp.,* **EC-8,** 159-169.

Potter, J. T. and Michel, P. C. [1952] High-density digital recording system, *IRE Trans. El. Comp.,* **EC-1,** 60-72.

Stephen, J. H. and Cooke-Yarborough, E. H. [1956] An interleaved-digit magnetic-

drum store for a transistor digital computer, *Proc. Inst. Elec. Engrs. (London)*, **103,** Pt. B, Supplement 3, Convention on Digital-Computer Techniques, 382-389.

Wallace, R. L., Jr. [1951] Reproduction of magnetically recorded signals, *Bell Syst. Tech. J.*, **30,** 1145-1173.

Williams, F. C. and Kilburn, T. [1952] Universal high-speed digital-computers: a magnetic store, *Proc. Inst. Elec. Engrs. (London)*, **99,** Pt. B, 94-106.

MAGNETIC DRUM STORAGE

Alrich, J. C. [1955] Engineering description of the ElectroData digital computer, *IRE Trans. El. Comp.*, **EC-4,** 1-10.

Bivans, E. W. [1955] Synchronizing magnetic drum storage speed, *Electronics*, **28,** 140-141.

Booth, A. D. [1960] High speed track selection for a magnetic drum store, *Electronic Engrg.*, **32,** 209-211

Carne, E. B. [1961] Magnetic memory drum design, *AIEE Trans.*, Pt. 1, **79,** 749-756.

Clayden, D. O., Page, L. J., and Osborne, C. F. [1956] The magnetic storage drum on the Ace Pilot model, *Proc. Inst. Elec. Engrs. (London)*, **103,** Pt. B, Supplement 3, Convention on Digital-Computer Techniques, 509-514.

Cohen, A. A. [1950] Magnetic drum storage for digital information processing systems, *MTAC*, **4,** 31-39.

Hollander, G. H. [1961] Drum organization for strobe addressing, *IRE Trans. El. Comp.*, **10,** 722-729.

Hong, K. [1958] Magnetic drum components for high storage density, *AIEE Trans.*, Pt. 1, **77,** 667-672.

Hughes, E. S., Jr. [1954] The IBM magnetic drum calculator Type 650; engineering and design considerations, *Proc. Western AIEE-IRE-ACM Computer Conference*, 140-154.

May, M., Miller, G. P., Howard, R. A. and Shifrin, G. A. [1959] A high speed, small size magnetic drum memory unit for subminiature digital computers, *Proc. 1959 Eastern Joint Computer Conference*, 191-199.

McGuigan, J. H. [1953] Combined reading and writing on a magnetic drum, *Proc. IRE*, **41,** 1438-1444.

Merry, I. W. and Maudsley, B. G. [1956] The magnetic-drum store of the computer Pegasus, *Proc. Inst. Elec. Engrs. (London)*, **103,** Pt. B, Supplement 2, Convention on Digital-Computer Techniques, 197-202.

Schaffer, R. R. [1960] A magnetic drum with a one megabit storage, *Electronic Industries*, **19,** No. 3, 114-117.

Seader, L. D. [1958] Magnetic-recording-head selection switch, *IBM J. Research and Develop.*, **2,** 36-42.

Williams, F. C. and West, J. C. [1951] Position synchronization of a rotating drum, *Proc. Inst. Elec. Engrs. (London)*, **98,** Pt. 2, 29-34.

MAGNETIC DISK STORAGE

Hoagland, A. S. [1961] A high track-density servo-access system for magnetic recording disk storage, *IBM J. Research and Develop.*, **5,** 287-296.

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

Noyes, T. [1957] The random-access memory accounting machine: II. The magnetic-disk, random-access memory, *IBM J. Research and Develop.*, **1,** 72-75.

Pearson, R. T. [1961] The development of the flexible-disk magnetic recorder, *Proc. IRE*, **49**, 164-174.

Rabinow, J. [1953] The notched-disc memory, *Elec. Eng.*, **71**, 745-749.

Seader, L. D. [1957] A self clocking system for information transfer, *IBM J. Research and Develop.*, **1**, 181-184.

STATIC SENSING

Chapin, D. M. [1949] A sensitive magnetometer for very small areas, *Rev. Sci. Instr.*, **20**, 945-946.

Hageman, D. H. A. [1951] A head for static reading of magnetic recording, M. S. Thesis, Dept. of Electrical Engineering, MIT.

Kilburn, T., Hoffman, G. R., and Wolstenholme, P. [1956] Reading of magnetic records by reluctance variation, *Proc. Inst. Elec. Engrs. (London)*, **103**, Pt. B, Supplement 2, Convention on Digital-Computer Techniques, 333-336.

Rubens, S. M. [1951] Static reading of magnetically stored digital information, *Report PX-29501, Contract NObsr-42001*, Engineering Research Associates, St. Paul, Minn., 25 pp.

MAGNETIC CORE MEMORIES AND SWITCHES

Aiken, Wang, and Woo [1948] Static magnetic recording, *Harvard Computation Laboratory Progress Report No. 1*, Investigations for the Design of Digital Calculating Machinery.

Alexander, M. A., Rosenberg, M., and Stuart-Williams, R. [1956] Ferrite-core memory is fast and reliable, *Electronics*, **29**, 158-161.

Auerbach, I. L. [1952] A static magnetic memory system for the Eniac, *Proc. ACM Convention*, 213-222, Pittsburgh.

Bartik, W. J. and Bonn, T. H. [1956] A small coincident-current magnetic memory, *IRE Trans. El. Comp.* **EC-5**, 73-78.

Best, R. L. [1957] Memory units in the Lincoln TX-2, *Proc. Western Joint Computer Conference*, 160-167.

Blachman, N. M. [1956] On the wiring of two-dimensional multiple-coincidence magnetic memorics, *IRE Trans. El. Comp.*, **EC-5**, 19-21.

Brown, D. R. and Albers-Schoenberg, E. [1953] Ferrites speed digital computers, *Electronics*, **26**, 146-149.

Carter, I. P. V. [1960] A new core switch for magnetic matrix stores and other purposes, *IRE Trans. El. Comp.*, **9**, 176-191.

Chien, R. T. [1960] A class of optimal noiseless load-sharing matrix switches, *IBM J. Research and Develop.*, **4**, 414-417.

Christopherson, W. A. [1961] Matrix switch and drive system for a low-cost magnetic-core memory, *IRE Trans. El. Comp.*, **10**, 238-246.

Constantine, G., Jr., [1960] New developments in load-sharing matrix switches, *IBM J. Research and Develop.*, **4**, 418-422.

Di Nolfo, R. S. [1954] Multi-coordinate selection systems for magnetic core storage, M.S. Thesis, MIT.

Forrester, J. W. [1951] Digital information storage in three dimensions using magnetic cores, *J. Appl. Phys.*, **22**, 44-48.

Foss, E. D. and Partridge, R. S. [1957] A 32,000-word magnetic core memory, *IBM J. Research and Develop.*, **1**, 102-109.

Haynes, M. K. [1952] Multidimensional magnetic memory selection systems, *IRE Trans. El. Comp.*, **EC-1**, 25-32.

Hunter, L. P. and Bauer, E. W. [1956] High-speed coincident-flux magnetic storage principles, *J. Appl. Phys.*, **27,** 1257-1261.

Karnaugh, M. [1959] Magnetic selectors, *Annals of the Computation Lab.*, **30,** 186-191, Harvard Univ. Press., Cambridge, Mass.

Lehman, M. [1961] Serial matrix storage systems, *IRE Trans. El. Comp.*, **10,** 247-252.

Marcus, M. P. [1959] Doubling the efficiency of the load-sharing matrix switch, *IBM J. Research and Develop.*, **3,** 194-196.

Menyuk, N. and Goodenough, J. G. [1955] Magnetic materials for digital computer components, I—a theory of flux reversal in polycrystalline ferromagnetics, *J. Appl. Phys.*, **26,** 8-18.

Minnick, R. C. [1959] Simultaneous access matrix storage systems, *in Proceedings of an International Symposium on the Theory of Switching* (Pt. II), pp. 144-148, Harvard Univ. Press, Cambridge, Mass.

Minnick, R. C. and Ashenhurst, R. L. [1955] Multiple-coincidence magnetic storage systems, *J. Appl. Phys.*, **26,** 575-579.

Papian, W. N. [1953] The MIT magnetic-core memory, *Proc. Eastern Joint Computer Conference,* 37-42.

Papian, W. N. [1955] New ferrite-core memory uses pulse transformers, *Electronics,* **28,** 194-197.

Raffel, J. I. [1954] Switch for register selection in a magnetic-core memory, M.S. Thesis, Dept. of Electrical Engineering, MIT.

Raffel, J. and Bradspies, S. [1955] Experiments on a three-core cell for high-speed memories, *IRE National Convention Record,* Pt. 4, 64-69.

Rajchman, J. A. [1953] A myriabit magnetic-core matrix memory, *Proc. IRE,* **41,** 1407-1420.

Robinson, A. A., Newhouse, V. L., Friedman, M. J., Bindon, D. G. and Carter, I.P.V. [1956] A digital store using a magnetic core matrix, *Proc. Inst. Elec. Engrs. (London),* **103,** Pt. B, Supplement 2, Convention on Digital-Computer Techniques, 295-301.

Schlaeppi, H. P. and Carter, I. P. V. [1960] Submicrosecond core memories using multiple coincidence, *IRE Trans. El. Comp.*, **9,** 192-198.

Straley, R., Heuer, A., Kane, B., and Tkach, G., [1960] Miniature memory plane for extreme environmental conditions, *J. Appl. Physics,* **31,** 126s-128s.

Stuart-Williams, R. [1961] Magnetic cores, characteristics and applications, *Auto. Control,* **15,** No. 7, 37-43.

Vogl, N. G., Jr. [1961] A new load-sharing matrix switch, *Digest of Technical Papers, 1961 Int'l. Solid State Circuits Conference,* 104-105.

Weisz, R. S. and Rosenberg, M. [1961] Wide temperature range coincident current core memories, *Proc. Western Joint Computer Conference,* 207-214.

Younker, E. L. [1957] A transistor-driven magnetic-core memory, *IRE Trans. El. Comp.*, **EC-6,** 14-20.

NONDESTRUCTIVE READ-OUT OF CORES

Buck, D. A. and Frank, W. I. [1954] Nondestructive sensing of magnetic cores, *Trans. Amer. Inst. Elec. Engrs.*, Pt. I, **72,** 822-830.

Kiseda, J. R., Petersen, H. E., Seelbach, W. C. and Teig, M. [1961] A magnetic associative memory, *IBM J. Research and Develop.*, **5,** 106-121.

Olsson, J. K. A. [1960] A method of storing binary information in ferrite memory cores with nondestructive readout, *Solid-State Physics in Electronics and Telecom.*, 404-410, Academic Press, New York.

Papoulis, A. [1954] The nondestructive read-out of magnetic cores, *Proc. IRE,* **42,** 1283-1288.

Shevel, W. L., Jr. and Gutwin, O. A. [1960] Partial switching, nondestructive-readout storage systems, *Digest of Technical Papers, 1960 Int'l. Solid State Circuits Conf.,* 62-63.

Tillman, R. M. [1960] Fluxlok—A nondestructive, random-access, electrically alterable, high-speed memory technique using standard ferrite cores, *IRE Trans. El. Comp.,* **9,** 323-328.

Thorensen, R. and Arsenault, W. R. [1955] A new nondestructive read for magnetic cores, *Proc. Western Joint Computer Conference,* 111-116.

Widrow, B. [1954] A radio-frequency nondestructive read-out for magnetic-core memories, *IRE Trans. El. Comp.,* **EC-3,** 12-15.

APERTURED FERRITE PLATE

Kaufman, M. M. and Newhouse, V. L. [1958] Operating range of a memory using two ferrite plate apertures per bit, *J. Appl. Phys.,* **29,** 487-488.

Rajchman, J. A. [1957] Ferrite apertured plate for random access memory, *Proc. IRE,* **45,** 325-334.

Rumble, W. G. and Warren, C. S. [1958] Coincident current applications of ferrite apertured plates, *IRE Wescon Convention Record,* Pt. 4, 62-65.

MULTI-APERTURE DEVICES

Abbott, H. W. and Suran, J. J. [1957] Temperature characteristics of the transfluxor, *IRE Trans. Electron Devices,* **4,** 113-119.

Hammel, D. G., Morgan, W. L. and Sidnam, R. D. [1959] A multiload transfluxor memory, *Proc. Western Joint Computer Conference,* 14-21.

Rajchman, J. A. and Lo, A. W. [1955] The transfluxor—a magnetic gate with variable setting, *RCA Review,* **16,** 303-311.

Rajchman, J. A. and Lo, A. W. [1956] The transfluxor, *Proc. IRE,* **44,** 321-332.

Vinal, A. W. [1961] The development of a multi-aperture reluctance switch, *Proc. Western Joint Computer Conference,* 443-474.

Wanlass, C. L. and Wanlass, S. D. [1959] BIAX high speed magnetic computer element, *1959 WESCON Convention Record,* Pt. 4, 40-54.

TWISTORS

Bobeck, A. H. [1957] A new storage element suitable for large sized memory arrays —the twistor, *Bell Syst. Tech. J.,* **36,** 1319-1340.

De Buske, J. J., Janik, J. Jr., and Simons, B. H. [1959] A card changeable nondestructive readout twistor store, *Proc. Western Joint Computer Conference,* 41-46.

Looney, D. H. [1959] A twistor matrix memory for semipermanent information, *Proc. Western Joint Computer Conference,* 36-41.

Preston, K., Jr. and Simkins, Q. W. [1959] Twistor buffer-store, *IRE-AIEE* Solid-*State Circuits Conference,* Philadelphia.

MAGNETIC FILM STORAGE

Bittmann, E. E. [1959a] Thin film memories, *IRE Trans. El. Comp.,* **EC-8,** 92-97.

Bittmann, E. E. [1959b] Using thin films in high speed memories, *Electronics,* **32,** 55-57.

Blois, M. S., Jr. [1955] Preparation of thin films and their properties, *J. Appl. Phys.*, **26,** 975-980.

Dietrich, W. and Proebster, W. E. [1959] Millimicrosecond magnetization reversal in thin magnetic films, *IBM J. Research and Develop.*, **3,** 375-376.

Dietrich, W., Proebster, W. E., and Wolf, P. [1960] Nanosecond switching in thin magnetic films, *IBM J. Research and Develop.*, **4,** 189-196.

Meier, D. A. [1959] A millimicrosecond magnetic switching and storage element, *J. Appl. Phys.*, **30,** Supplement, 45S-46S.

Petschauer, R. J. and Turnquist, R. D. [1961] A nondestructive readout film memory, *Proc. Western Joint Computer Conference,* 411-25.

Pohm, A. V. and Rubens, S. V. [1956] A compact coincident-current memory, *Proc. Eastern Joint Computer Conference,* 120-123.

Pohm, A. V. [1960] Magnetic film memories, a survey, *IRE Trans. El. Comp.*, **9,** 308-314.

Raffel, J. [1959] Operating characteristics of a thin-film memory, *J. Appl. Phys.*, **30,** Supplement, 60s-61s.

Raffel, J. I., Crowther, T. S., Anderson, H. A. and Herndon, T. O. [1961] Magnetic film memory design, *Proc. IRE,* **49,** 155-164.

Smith, D. O. [1958] Static and dynamic behavior of thin Permalloy films, *J. Appl. Phys.*, **29,** 264-273.

SUPERCONDUCTING STORAGE

Buck, D. A. [1956] The cryotron—a superconductive computer component, *Proc. IRE,* **44,** 482-493.

Burns, L. L., Jr., Alphonse, G. W. and Leck, G. W. [1961] Coincident-current superconductive memory, *IRE Trans. El. Comp.*, **10,** 438-446

Crittenden, E. C., Cooper, J. N. and Schmidlin, F. W. [1960] The "persistor"—a superconducting memory element, *Proc. IRE,* **48,** 1233-1246.

Crowe, J. W. [1957] Trapped-flux superconducting memory, *IBM J. Research Develop.*, **1,** 294-303.

Davies, P. M. [1962] A superconductive associative memory, *Proc. AFIPS Spring Joint Computer Conference,* 79-88.

Garwin, R. L. [1957] An analysis of the operation of a persistent-supercurrent memory cell, *IBM J. Research and Develop.*, **1,** 304-308.

Newhouse, V. L. [1961] Superconductive circuits for computing machines, *Electro-Technology,* **67,** No. 4, 78-89.

Newhouse, V. L. and Fruin, R. E. [1962] A cryogenic data addressed memory, *Proc. AFIPS Spring Joint Computer Conference,* 89-99.

Newhouse, V. L., Bremer, J. W., Edwards, H. H. [1960] An improved film cryotron and its application to digital computers, *Proc. IRE,* **48,** 1395-1404.

Rhoderick, E. H. [1959] Superconducting computer elements, *Brit. J. Appl. Phys.*, **10,** 193-198.

Rosin, R. F. [1962] An organization of an associative cryogenic computer, *Proc. AFIPS Spring Joint Computer Conference,* 203-212.

Seeber, R. R. [1960] Associative self-sorting memory, *Proc. 1960 Eastern Joint Computer Conference,* 179-187.

Seeber, R. R. and Lindquist, A. B. [1962] Associative memory with ordered retrieval, *IBM J. Research and Develop.*, **6,** 126-136.

Smallman, C. R., Slade, A. E., Cohen, M. L. [1960] Thin-film cryotrons, *Proc. IRE,* **48,** 1562-1582.

TUNNEL-DIODE STORAGE

Beck, E. R., Savitt, D. A. and Whiteside, A. E. [1961] Tunnel diode storage using current sensing, *Proc. Western Joint Computer Conference,* 427-442.

Berry, D. L. and Fisch, E. A. [1961] High-speed tunnel-diode memory, Digest of Technical Papers, *1961 Int'l. Solid State Circuits Conf.,* 112-113.

Chaplin, G. B. B. and Thompson, P. M. [1961] A fast word organized tunnel-diode memory using voltage mode selection. *Digest of Technical Papers, 1961 Int'l. Solid State Circuits Conference,* 40-41.

ELECTROSTATIC STORAGE

WILLIAMS TUBES

Eckert, J. P., Jr., Lukoff, H., and Smoliar, G. [1950] A dynamically regenerated electrostatic memory system, *Proc. IRE,* **38,** 498-510.

Edwards, D. B. G. [1956] The design and operation of a parallel-type cathode-ray tube storage system, *Proc. Inst. Elec. Engrs. (London),* **103,** Pt. B, Supplement 2, Convention on Digital-Computer Techniques, 319-326.

Graham, M. [1956] An improved method for Williams storage, *IRE Trans. El. Comp.,* **EC-5,** 140.

Haeff, A. V. [1947] A memory tube, *Electronics,* **20,** 80-83.

Kilburn, T. [1953] Universal high-speed digital computers: a decimal storage system, *Proc. Inst. Elec. Engrs. (London),* Pt. 2, **100,** 513-522.

Logue, J. C., Brenneman, A. E., and Koelsch, A. C. [1953] Engineering experience in the design and operation of a large-scale electrostatic memory, *IRE National Convention Record,* Pt. 7, 21-29.

Williams, F. C. and Kilburn, T. [1949], [1950] A storage system for use with binary-digital computing machines, *Proc. Inst. Elec. Engrs. (London),* **96,** Pt. 2, 183-202, Pt. 3, 77-100, **97,** Pt. 4, 453-454.

Williams, F. C., Kilburn, T., Litting, C. N. W., Edwards, D. B. G., and Hoffman, G. R. [1953] Recent advances in cathode-ray storage, *Proc. Inst. Elec. Engrs. (London)* Pt. 2, **100,** 523-543.

Wong, S. Y. [1955] High-density Williams storage, *IRE Trans. El. Comp.,* **4,** 156-158.

BARRIER-GRID TUBES

DeLano, R. B., Jr. [1954] A large-capacity storage tube for digital computer applications, *IRE National Convention Record,* Pt. 3, 125-130.

Graham, M., Miller, G. L., Pate, H. R. and Spinrad, R. [1959] The design of a large electrostatic memory, *IRE Trans. El. Comp.,* **8,** 479-485.

Hines, M. E., Chruney, M., and McCarthy, J. A. [1955] Digital memory in barrier-grid storage tubes *Bell Syst. Tech. J.,* **34,** 1241-1264.

HOLDING-GUN TUBES

Dodd, S. H., Klemperer, H., and Youtz, P. [1950] Electrostatic storage tube, *Elec. Eng.,* **69,** 990-995.

THE SELECTRON

Rajchman, J. A. [1951] The selective electrostatic storage tube, *RCA Rev.*, **12,** 53-97.

DIODE-CAPACITOR STORAGE

Conway, A. C. [1959] A fast random-access diode-capacitor store using transistors, *Proc. IEE,* Pt. B., **106,** Suppl. 16, 657-662.

Kaufman, M. M. [1959] Millimicrosecond diode-capacitor memory, *Proc. Natl. Electronics Conf.*, **15,** 215-225, Natl. Electronics Conf. Inc., Chicago.

Holt, A. W. [1952], [1953] An experimental rapid access memory using diodes and capacitors, *Proc. of ACM Toronto Meeting,* 133-141; *NBS Electronic Computer Lab. Report,* 133-141.

Slutz, R. J., Holt, A. W., Witt, R. P., and Friedman, D. C. [1955] Diode-capacitor memory, *NBS Circular 551,* Computer Development at the NBS, 102-107.

LUMPED AND DISTRIBUTED CONSTANT DELAY LINES

Anderson, J. R. [1953] Electrical delay lines for digital computer applications, *IRE Trans. El. Comp.*, **EC-2,** 5-13

Brillouin, L. N. [1948] Electromagnetic delay lines. *Proc. Symp. on Large-Scale Digital Calculating Machinery, 1947,* 110-124, Harvard Univ. Press.

Scarrott, G. G., Harwood, W. J., and Johnson, K. C. [1956] Electromagnetic delay networks for digital storage, *Proc. Inst. Elec. Engrs. (London),* **103,** Pt. B, Supplement 3, Convention on Digital-Computer Techniques, 476-482.

ACOUSTIC DELAY LINE STORAGE

Arenberg, D. L. [1948] Ultrasonic solid delay lines, *J. Acoust. Soc. Am.*, **20,** 1-26.

Arenberg, D. L. [1954] Ultrasonic delay lines, *IRE National Convention Record,* Pt. 6, 63-72.

Auerbach, I. L., Eckert, J. P., Jr., Shaw, R. F., and Sheppard, C. B. [1949] Mercury delay line memory using a pulse rate of several megacycles, *Proc. IRE,* **37,** 855-861.

Beveridge, H. N. and Keith, W. W. [1952] Piezoelectric transducers for ultrasonic delay lines, *Proc. IRE,* **40,** 828-835.

Emslie, A. G., Huntington, H. B., Shapiro, H., and Benfield, A. E. [1948] Ultrasonic delay lines, *J. Franklin Inst.*, Pt. II, **245,** 101-115.

Fagen, M. D. [1951] Performance of ultrasonic vitreous silica delay lines, *Proc. Natl. Electronics Conference,* **7,** 380-389.

Huntington, H. B., Emslie, A. G., and Hughes, V. W. [1948] Ultrasonic delay lines, *J. Franklin Inst.*, Pt. I, **245,** 1-24.

Mapleton, R. A. [1952] Elastic wave propagation in solid media, *J. Appl. Phys.*, **23,** 1346-1354.

May, J. E. [1954] Characteristics of ultrasonic delay lines using quartz and barium titanate ceramic transducers, *J. Acoust. Soc. Am.*, **26,** 347-355.

Mebs, R. W., Darr, J. H., Grimsley, J. D. [1953] Metal ultrasonic delay lines, *NBS Tech. News Bull.*, J. Research NBS, **51,** 209.

Newman, E. A., Clayden, D. O. and Wright, M. A. [1953] Mercury delay line storage system of the ACE pilot model electronic computer, *Proc. Inst. Elec. Engrs. (London),* Pt. 2, **100,** 445-452.

Pennell, E. S. [1952] Vitreous silica for ultrasonic delay line applications, *Proc. Natl. Electronics Conference,* **8,** 799-810.

Ryan, R. D. [1955] A mercury delay line storage unit, *J. Brit. IRE,* **15,** 419-427.

Spaeth, D. A., Rogers, T. F., and Johnson, S. J. [1954] Wide-band large dynamic range fused-quartz delay lines for increased capacity high-speed computer memories, *IRE National Convention Record,* Pt. 6, 73-76.

MAGNETOSTRICTIVE DELAY LINES

Beck, R. M. [1960] A high-speed serial general-purpose computer using magnetostrictive delay line storage, *Proc. 1960 Eastern Joint Computer Conference,* 283-297.

Bradbury, E. H. [1951] Magnetostrictive delay line, *Elec. Commun.,* **28,** 46-53.

Chaplin, G. B. B., Hayes, R. E., and Owens, A. R. [1955] A transistor digital fast multiplier with magnetostrictive storage, *Proc. Inst. Elec. Engrs. (London),* Pt. B, **102,** 412-425.

Fairclough, J. W. [1956] A sonic delay-line storage unit for a digital computer, *Proc. Inst. Elec. Engrs. (London),* **103,** Pt. B, Supplement 3, Convention on Digital-Computer Techniques, 491-496.

Robbins, R. C. and Millership, R. [1954] Applications of magnetostrictive delay lines, *Proc. Symp. on Automatic Digital Computation, National Physical Laboratory, 1953,* 199-212, London.

Rothbart, A. and Brown, A. J. [1962] What designers should know about magnetostrictive delay lines, *Electronics,* **35,** No. 15, 55-59.

Rothbart, A. and Brown, A. J. [1962] How to Specify Magnetostrictive delay lines, *Electronics,* **35,** No. 20, 54-57.

Scarrott, G. G. and Naylor, R. [1956] Wire-type acoustic delay lines for digital storage, *Proc. Inst. Elec. Engrs. (London),* **103,** Pt. B, Supplement 3, Convention on Digital-Computer Techniques, 497-508.

Williams, R. C. [1959] Theory of magnetostrictive delay lines for pulse and continuous wave transmission, *IRE Trans. Ultrasonics Engrg.,* **7,** 16-38.

FERROELECTRIC STORAGE

Anderson, J. R. [1952] Ferroelectric storage elements for digital computers and switching systems, *Elec. Eng.,* **71,** 916-922; also, Bell Telephone Laboratories Monograph 2014.

Anderson, J. R. [1956] A new type of ferroelectric shift register, *IRE Trans. El. Comp.,* **5,** 184-191.

Buck, D. A. [1952] Ferroelectrics for digital information storage and switching, M. S. Thesis, E. E. Dept., MIT.

Campbell, D. S. [1957] Barium titanate and its use as a memory store, *J. Brit. IRE,* **17,** 385-395.

Prutton, M. [1959] Ferroelectrics and computer storage, *J. Brit. IRE,* **19,** 93-102.

Pulvari, C. F. and McDuffie, G. E. Jr., [1958] Scanners for ferroelectric memory capacitors, *IRE Trans. El. Comp.,* **EC-7,** 34-40.

Pulvari, C. F. [1955] Memory matrix using ferroelectric condensers as bistable elements, *J. ACM,* **2,** 169-185.

PHOTOGRAPHIC STORAGE

King, G. W., Brown, G. W., and Ridenour, L. N. [1953] Photographic techniques for information storage, *Proc. IRE,* **41,** 1421-1428.

Lovell, C. A. [1958] High-speed, high-capacity photographic memory, *Proc. 1958 Eastern Joint Computer Conf.,* 34-38.

6. Arithmetic Operations

In Section 6.1 a number of the schemes which have been devised to perform certain elementary operations in a digital computer are described, namely, counting, generation of pulse patterns, addition, subtraction, multiplication, and division.

Section 6.2 describes certain basic procedures of numerical analysis, which can be used either directly as the basis for mechanization or indirectly as the basis for constructing programs by means of which operations for the extraction of roots and the generation of trigonometric functions may be performed.

Techniques for scaling a problem, i.e., taking measures to assure that the values of all variables generated in the course of computing will be within the bounds of the machine's capacity are described in Section 6.3.

Section 6.4 describes certain schemes that have been devised for converting numbers from binary to decimal notation and vice versa. These schemes are used principally to allow information in decimal form to be entered into or brought out of a binary digital computer.

6.1. Algorithms and Logical Designs for Mechanization of Basic Arithmetic Operations

6.1.1. Counting

6.1.1.1. *Counting with Set–Reset Flip-flops*

Each of n flip-flops in a collection can be assigned one of the weights $2^{n-1}, \ldots 2^2, 2^1, 2^0$ and so interconnected that signals that appear serially from a source S cycle the flip-flops through 2^n states starting with zero and ending with 2^{n-1} (in binary representation), and after state 2^{n-1} reset the counter to zero. The flip-flop input signals can be modified to allow resetting to zero at other times, e.g., for clearing prior to counting, or after reaching the value 9 if straight binary-coded decimal representation (see Table 6.8, Section 6.1.3.1) is to be used.

As an example, a counter comprised of three set–reset type flip-flops will be described. Table 6.1 shows the successive states of each flip-flop.

TABLE 6.1. Successive states of a three stage binary counter

A_3	A_2	A_1	Time
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

The nature of the signals required at the inputs to each flip-flop may be determined by inspection of the table, for an input signal to a particular flip-flop must be supplied only when the state of that flip-flop must change to represent the next count. The input requirements of each flip-flop will be considered in turn. Flip-flop A_1 (representing the least significant bit of the count) must change from one state to the other each time a signal from S appears, i.e., if the flip-flop is in state A_1, the next signal from S must set the flip-flop to state $\bar{A}_1$ and vice versa.
Therefore

$$a_1 = \bar{A}_1 S \qquad \bar{a}_1 = A_1 S$$

The second flip-flop A_2 changes from state 0 to 1, i.e., from $\bar{A}_2$ to A_2 upon receipt of a signal from S only if the state $\bar{A}_2 A_1$ existed prior to receipt of the signal. It changes from 1 to 0, i.e., from A_2 to $\bar{A}_2$ only if the preceding state was $A_2 A_1$. Therefore

$$a_2 = \bar{A}_2 A_1 S \qquad \bar{a}_2 = A_2 A_1 S$$

Similarly

$$a_3 = \bar{A}_3 A_2 A_1 S \qquad \bar{a}_3 = A_3 A_2 A_1 S.$$

The six required input signals to the three flip-flops may be formed by a diode gating network as shown in Fig. 6.1(a). Note that the technique of pyramiding, described in Chapter 4, is employed in the forming of this network. If, instead of set–reset flip-flops, single-input flip-flops are used, only three input signals are required, namely

$$c_1 = S \qquad c_2 = C_1 S \qquad c_3 = C_2 C_1 S.$$

This results in a simpler gating network, as shown in Fig. 6.1(b).

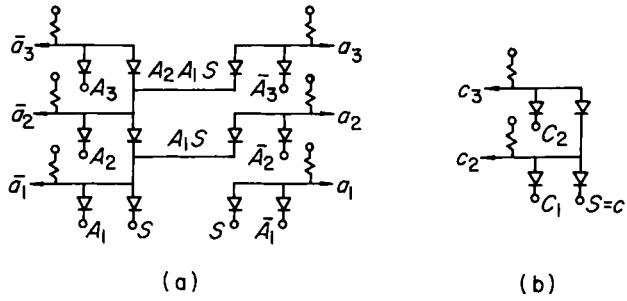


FIG. 6.1. Diode gating network for generating input signals to a three stage binary counter composed of (a) *R-S* flip-flops, (b) T flip-flops

6.1.1.2. *Bistable Counter Circuits*

The circuit shown in Fig. 6.2(a) is similar to the complement type

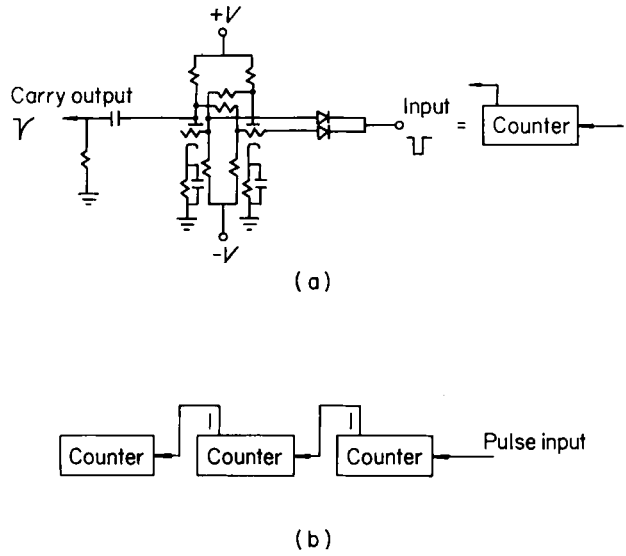


FIG. 6.2. (a) A bistable counter circuit, (b) Cascading of circuits to form a multistage counter

of static flip-flop. Successive pulses (in this case, negative) on the input line cause first one amplifier and then the other to be nonconducting. The counter is said to be in the 1 or 0 state depending on which amplifier is conducting. Upon application of the input pulse both amplifiers

are cut off. After the applied pulse has died away, the circuit becomes quiescent with the previously nonconducting amplifier now conducting and vice versa. The carry output of the circuit is simply the output of the amplifier which produces a negative pulse when the counter changes from the 1 to 0 state. A positive pulse appears on the carry output lead when the state of the circuit changes from 0 to 1, but this has no effect when used as the input to another bistable counter.

6.1.1.3. *Multistage Counters*

6.1.1.3.1. CONNECTION OF BISTABLE COUNTER CIRCUITS IN CASCADE. A multistage counter can be formed by connecting bistable counter circuits in cascade, as shown in Fig. 6.2(b). Such circuits produce a count by summing and storing a unitary weighted pulse stream input. Pulses can be applied to the input at intervals not less than the operating time of one stage. The inputs can be synchronous or not. The time required to assume a new steady state depends on the length of carry propagation. For an n stage counter the maximum time is nd, where d is the interval from when an input waveform to a stage reaches a critical value to when the output waveform of that stage reaches a critical value.

6.1.1.3.2. MULTISTAGE COUNTERS WITH ANTICIPATORY CARRY. Inspection of the counting process in the binary number system shows that upon the addition of an increment 2^{-n}, any columnar bit changes (from 1 to 0 or 0 to 1) only if all less significant bits contain 1. Therefore, upon receipt of an input pulse t, the condition for any stage B_i of a counter to change is provided by the signal

$$b_i = B_{i-1}B_{i-2} \ldots B_0 t.$$

If one were to mechanize the circuit directly from such equations, difficulties would occur, for a counter having a large number of stages, because of the many terms involved in the AND gate inputs to the more significant stages. The circuit shown in Fig. 6.3(a) alleviates this condition at the cost of a slight decrease in speed. It operates as follows: The current count is stored in a group of bistable counter circuits. Upon receipt of an input pulse the first stage may generate a carry. This carry is propagated through a series of gates, each of which is controlled by the state of a bistable counter circuit. The carry propagation is stopped by the first counter circuit in the 0 state. It takes less time to propagate a carry through a gate than to trigger a counter circuit. Therefore, each counter circuit can be triggered by the counter circuit in the next less

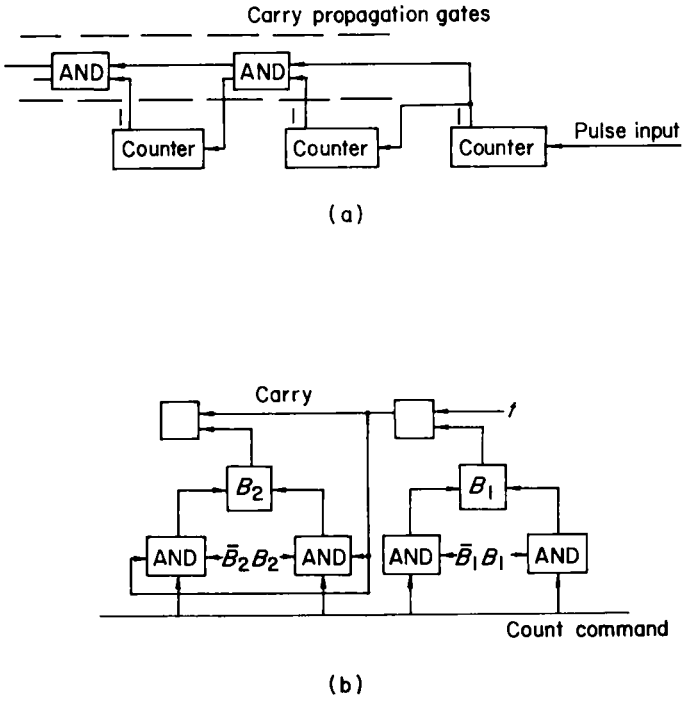


FIG. 6.3. Multistage counters with a two-input/stage carry propagation line, (a) using bistable counter circuits, (b) using *R–S* flip-flops

significant position only after it has served the function of passing a carry to the succeeding stage. The maximum time to reach the steady state is dependent on the speed of the carry propagation circuit.

The counter shown in Fig. 6.3(b) utilizes set–reset flip-flops. Its operation is based on the fact that any stage B_i should change from 0 to 1 or 1 to 0 in accordance with the following equations

$$b_i = \bar{B}_iB_{i-1} \ldots B_0t$$

$$\bar{b}_i = B_iB_{i-1} \ldots B_0t.$$

Upon presentation of an input pulse t, a carry is propagated from the least significant end of the counter until it reaches a stage in the 0 state. The duration of the count command signal must be greater than the propagation time of the maximum carry chain. Each stage that receives a carry signal is caused to change state after it has served the function of passing a carry to the succeeding stage. The maximum time to reach the steady state is dependent on the speed of the carry propagation circuit.

6.1.1.4. *Dynamic Binary Counters*

Let us consider how a counter may be constructed from a number of trigger type dynamic flip-flops of the type shown in Fig. 3.16(b). The input–output equation for the least significant stage A_1 is simple since the least significant bit alternates between 0 and 1 with each input pulse. Any other stage should change state when there is a transition in the preceding stage from 1 to 0. This transition is recognized by sensing the present output of a stage and its output one pulse period earlier. Accordingly, the input–output relations for a two-stage counter would be

$$(A_1)_{i+1} = (\bar{A}_1)_i T + (A_1)_i \bar{T}$$

$$(A_2)_{i+1} = (\bar{A}_2)_i [(A_1)_i (\bar{A}_1)_{i+1}] + (A_2)_i \overline{[(A_1)_i (A_1)_{i+1}]}$$

where A_1, A_2 represent the first and second stages, respectively. The term $(\bar{A}_2)_i (A_1)_i (\bar{A}_1)_{i+1}$ states that if the second stage is in the 0 state and stage one is changing from 1 to 0, stage two should change from 0 to 1. The term $(A_2)_i \overline{[(A_1)_i (\bar{A}_1)_{i+1}]}$ states that if A_2 is in the 1 state, it should not change to 0 until stage one changes from 1 to 0. Note that the expression for $(A_2)_{i+1}$ is of the same form as $(A_1)_{i+1}$.

With the arrangement shown in Fig. 6.4 a new accumulated count

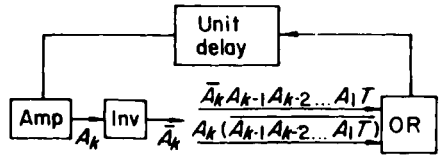


FIG. 6.4. Typical stage of a dynamic binary counter

is provided after only one pulse period. Each stage employs a trigger type of dynamic flip-flop as before. However, there is a difference in the input–output expression for each stage. In this case, each stage changes state whenever all the preceding stages are on and a count pulse T is applied. A significant difference in the two counters is that in the latter the number of variables in the inputs to the OR gate increases as the number of stages in the counter increases. This is not so in the case of the former counter.

6.1.1.5. *Use of a Multibit Delay Line as a Counter*

An n-bit number stored in a delay line can be changed by a single positive or negative increment at a time by means of a simple logical scheme. The scheme is based on the special nature of the change which

can occur when 2^{-n} is added to or subtracted from a number. When 2^{-n} is added to a binary number, starting from the least significant bit each 1 in a sequence of 1's will be changed to 0, and the first 0 that is encountered will be changed to a 1, at which point the process stops, i.e., the more significant bits will be left unchanged. For example

.011001111	Original number
.000000001	$+ 2^{-n}$
.011010000	Sum

This simple operation can be mechanized as follows: Assume that the bits of the original number appear serially, being represented by successive states of a flip-flop A. The only equipment required to add 2^{-n} tc this number is another flip-flop B and a simple combinational circuit. The flip-flop B will always be set to the 1 state prior to the addition process, and will be set to the 0 state by the first 0 that appears in the original number, i.e., by the signal $\bar{A}$. The correct sum may be formed by an exclusive OR (i.e., a sum modulo 2) gate having the input variables A, B. The arrangement is shown in Fig. 6.5. The actual process that

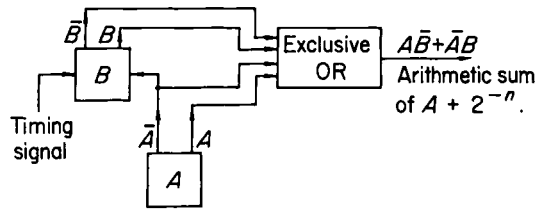


FIG. 6.5. Logical arrangement for serial addition of a single increment

takes place is indicated by the successive states of A, B

.011001111	Successive states of A
.000011111	Successive states of B
.011010000	$A\bar{B} + \bar{A}B$

The algorithm for subtraction of a single increment states that each sequence of 0's in the given number is changed to a sequence of 1's, and that the first 1 encountered is changed to a 0 at which point the process stops. By a simple modification, the above circuitry may also be used for subtraction of a single increment. The combinational circuit remains the same, but it is necessary to alter the 0 set input signal to the flip-flop B. Before either operation takes place, a signal P or $\bar{P}$, indicating an addi-

tion or subtraction, respectively, is provided, and the 0 set input signal to the flip-flop B becomes $\bar{A}P + A\bar{P}$. Therefore, during subtraction, the flip-flop B will be set to 1 initially as before, but now it will be set to the 0 state by the first 1 that appears in the original number A. The normal and the mechanized pseudo-operation for subtraction are illustrated below

.011001100	Original number	.011001100	Successive states of A
.000000001	-2^{-n}	.000000111	Successive states of B
.011001011	Difference	.011001011	$A\bar{B} + \bar{A}B$

Three-state arrangement:

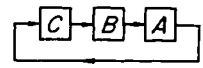
Flip-Flop A:
$a = B$
$\bar{a} = \bar{B}$

Flip-Flop B:
$b = C$
$\bar{b} = \bar{C}$

Flip-Flop C:
$c = A$
$\bar{c} = \bar{A}$

Sequence of States:

C	B	A
0	0	1
1	0	1
1	1	1
1	1	0
0	1	1
1	0	0
0	1	0



Information flow

(a)

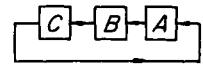
Flip-Flop A:
$a = C\bar{A}$
$\bar{a} = CA$

Flip-Flop B:
$b = A$
$\bar{b} = \bar{A}$

Flip-Flop C:
$c = B$
$\bar{c} = \bar{B}$

Sequence of States:

C	B	A
1	1	0
1	0	1
0	1	0
1	0	0
0	0	1
0	1	1
1	1	1



Information flow

(b)

FIG. 6.6 Three-stage pattern generators

6.1.1.6. Pattern Generators

If a group of flip-flops is to be used solely for the function of generating a number of distinguishable states (which are used to control other operations), it is not necessary that the states change in the same sequence as would a counting device. By eliminating the restriction, a saving can be effected in the number of gating elements required to generate the flip-flop input signals. Strictly speaking, it is not necessary that a counting device change states in a sequence of successively greater binary number representations, since it is possible to translate from any code to another. However, it is simpler to perform conventional arithmetic operations on the outputs of such counters.

Figure 6.6 shows the input equations for a three stage pattern generator, composed of R_T–S_T flip-flops (Section 3.7.1, Fig. 3.14(d)). If R–S flip-flops are used, the input equations to flip-flop A in Fig. 6.6(a) would be

$$a = B\bar{A} \qquad \bar{a} = BA.$$

This circuit arrangement has the characteristic that the configuration 000 is not used. If the circuit should ever get into this state, it could not normally leave it. This possibility may be avoided by altering the input equation a to

$$a = B\bar{A} + \bar{C}\bar{B}.$$

The term $\bar{C}\bar{B}$ has no effect other than to move the circuit from state 000 to 001.

A slightly different arrangement is shown in Fig. 6.6(b). It also has the characteristic that the configuration 000 is not used and represents a stalled state. To provide for getting out of this state, the input equation a is changed as follows

$$a = C\bar{A} + \bar{C}\bar{B}.$$

In addition to generating control signals of the form $f(A, B, C)$, circuits of this type can also be used to generate cycles of sequential pulse patterns. For example, in Fig. 6.6(b) the A output of flip-flop A generates the pattern 0100111.

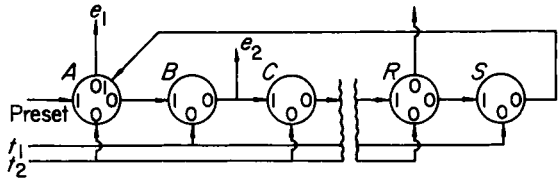


FIG. 6.7. A pattern generator formed from a magnetic shift register with end-around shift

A pattern generator can also be based on a shift register. For example, a magnetic shift register such as that shown schematically in Figure 6.7 could be used. (The basic operation of such registers is described in Section 4.9). By using only an end-around feedback, as in Figure 6.7, a pattern once inserted will be propagated so long as shift pulses are applied to t_1 and t_2. The length of this pattern is n, where n is the number of stages. By tapping off at various points, e.g., at e_1 and e_2, one can obtain the same sequential pattern with various displacements in time. Longer sequences (up to 2^{n-1}) can be obtained by feeding back signals from intermediate points and combining these with the end-around feedback by means of an OR gate or other logic.

6.1.2. BINARY ADDITION

6.1.2.1. Serial Binary Adders

The addition of two binary numbers, represented by $A_1 \ldots A_n$, and $B_1 \ldots B_n$, is accomplished by the generation of sum S_i and carry C_i bits respectively, as shown in Table 6.2, where C_{i-1} indicates the carry bit produced in the preceding bit column. For example, in the decimal system, the addition of 6 and 7 produces a sum of 3 (modulo 10) and a carry of 1, so the answer is 13.

Several methods of addition are known. They fall into two classes, namely those in which the number of 1's in corresponding bit positions of the addends and the generated carry pulse trains are counted, and those in which logical operations are built up to obey the addition table.

6.1.2.1.1. ADDITION BY COUNTING. Addition may be performed either by a digital or analog summation of increments. In the digital method, pulses representing corresponding bits of A_i, B_i, and C_{i-1} are arranged to occur at slightly different times and are counted by a two

TABLE 6.2.

A_i	B_i	C_{i-1}	Number of 1's in A_i, B_i, C_{i-1}	C_i (carry)	S_i (sum)
0	0	0	0	0	0
1	0	0			
0	1	0	1	0	1
0	0	1			
0	1	1			
1	0	1	2	1	0
1	1	0			
1	1	1	3	1	1

stage binary counter. Then the sum and carry bits are obtained from inspection of the states of the first and second stages, respectively. Afterwards, the counter is reset to zero pending the arrival of the next bits of the input numbers and the generated carry. The timing of the signals involved in this method is indicated in Fig. 6.8. The C_{i-1} pulse is

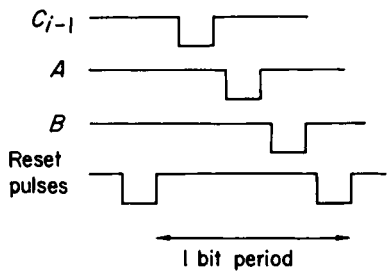
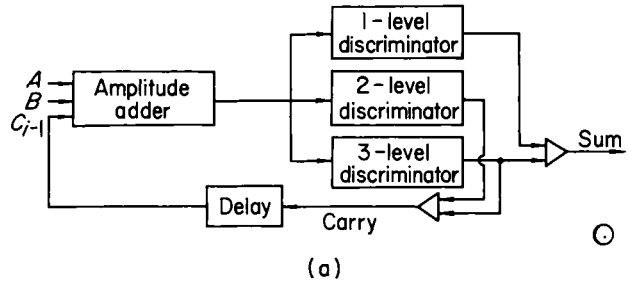


FIG. 6.8. Relative timing of signals to a sum and carry producing counter

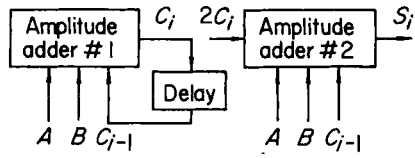
counted first so that the output pulse, which may occur at the same time as the reset pulse, can follow the input pulses as closely as possible. Even so, sufficient time to count two pulses must elapse between the first of the inputs and the output signals. This delay in operation is unavoidable with an adder of this type and may usually be considered a serious disadvantage.

In the analog technique, the input pulse trains are arranged to occur simultaneously. The amplitudes of these signals are added, and the number of 1's deduced from the level of the combined signal. Thus the output is available almost immediately (except for delays inherent in the natural

time constants of any electrical circuit). A block diagram of a type of level discriminating adder is shown in Fig. 6.9(a). The sum of the ampli-



(a)



(b)

FIG. 6.9. Level discriminating adders

tudes of the pulse trains A_i, B_i, C_{i-1} is applied to three discriminators of levels 1, 2, and 3, respectively. Each produces an output pulse if the corresponding number of inputs is 1. An improved and simpler form of level discriminating adder, shown in Fig. 6.9(b), is based on the fact that the sum bit may be obtained by subtracting twice the value of the carry bit from the number of 1's in A_i, B_i, and C_{i-1}. The output of amplitude adder No. 1 is of the opposite sign to the inputs, and its level is limited so that the same output is produced from two or three 1's; there is no output for a single 1 input. The output, C_i, after being delayed and restored to the original polarity, is designated C_{i-1}. The amplitude adder No. 2 gives an output pulse when $A_i + B_i + C_{i-1} - 2C_i = 1$, since C_i is of opposite sign to A_i, B_i, and C_{i-1}, and these pulses form the sum bits.

Neither level discriminating adder produces a correct answer until the input pulses have all reached their standard amplitudes. Normally, spurious results may be avoided by suppressing the output of the adder until the input pulses have reached their final amplitude. To make the levels sufficiently distinct, the permissible variation in the nominal amplitudes of the input and C_i pulses are usually held to within $\pm$ 5%. Though

this analog adding circuit requires a higher precision of operation than digital circuits, the precision is well within the capabilities of the state of the art.

6.1.2.1.2. ADDITION BY USE OF LOGICAL OPERATIONS. If a circuit is to be used only to add any single power of two to an arbitrary number, then a simplification is possible over the circuit required to add two arbitrary numbers. Consider Example 6.1

Example 6.1

n	7	6	5	4	3	2	1	0	
A =	0	1	0	1	1	0	1	1	
2^3 =	0	0	0	0	1	0	0	0	
S_i =	0	1	1	0	0	0	1	1	S_i = Sum (modulo 2) of $(A, 2^n, C_{i+1})$
C_i =	0	0	1	1	0	0	0	0	C_i = carry bit from the $(i-1)$th order.

Since 2^n and C_i can never be 1 at the same time, the adder circuit required will never have to deal with three input signals simultaneously, but only two. A schematic of such a circuit is shown in Fig. 6.10. Since 2^n and the delayed carry can never be 1 simultaneously, their logical sum may be thought of as a conventional single binary input signal, and is represented in Fig. 6.10 by B. A circuit that accepts two binary inputs,

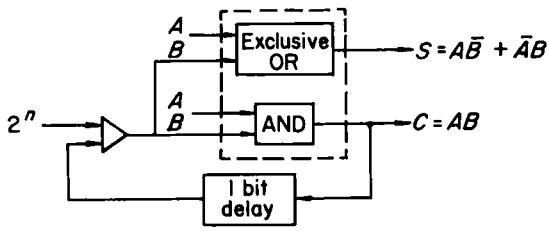


FIG. 6.10. A half-adder

A, B and produces the output signals S and C as shown in Fig. 6.10 is termed a half-adder. The term half-adder derives from the fact that a full adder can be constructed from two half-adders. This can easily be demonstrated. First, consider the equations for the sum and carry of an adder with arbitrary inputs, A_i, B_i. These equations can be obtained from Table 6.2 by noting the set of values of A_i, B_i, and C_{i-1} for which S_i and C_i, respectively are 1

$$S_i = A_i\bar{B}_i\bar{C}_{i-1} + \bar{A}_iB_i\bar{C}_{i-1} + \bar{A}_i\bar{B}_iC_{i-1} + A_iB_iC_{i-1} \tag{6-1}$$

$$C_i = A_iB_i\bar{C}_{i-1} + A_iB_iC_{i-1} + \bar{A}_iB_iC_{i-1} + A_i\bar{B}_iC_{i-1}. \tag{6-2}$$

When the outputs S and C of one half-adder are used as inputs to another half-adder as shown in Fig. 6.11, equivalent expressions for S_i and C_i result

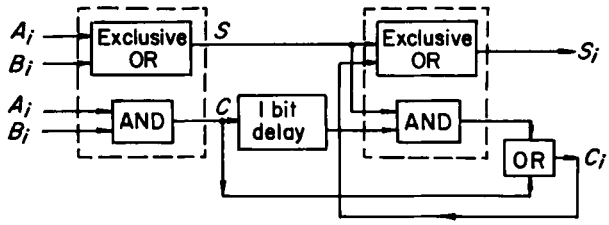


FIG. 6.11. An adder comprised of two half-adders

$$S_i = (A_i\bar{B}_i + \bar{A}_iB_i)\,\bar{C}_{i-1} + (\bar{A}_i\bar{B}_i + A_iB_i)\,C_{i-1}.$$

$$C_i = A_iB_i + (A_i\bar{B}_i + \bar{A}_iB_i)\,C_{i-1}$$

In the adder of Fig. 6.11 it is assumed that corresponding bits of the two addends arrive simultaneously. Both A_i and B_i must pass through two half-adders, whereas C_{i-1} passes only through one. The one bit delay is required to store C, till the bits of A and B in the next more significant place arrive, at which time it is added to them. If the two addends do not arrive simultaneously, the modification shown in Fig. 6.12 may be used. For example, its use would be indicated if the adder

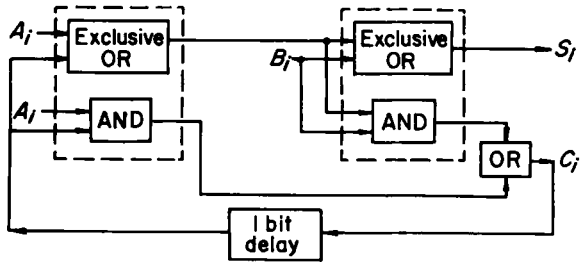


FIG. 6.12. An adder suitable for use with a recirculating type of main store

were to be included in the regenerative loop of a circulating type of storage unit. In this case, the input A_i from the main store will usually arrive slightly earlier than the external input B_i which passes through gating circuits before reaching the adder.

The sum and carry of two serial binary inputs may be generated

without recourse to half-adders by straightforward mechanization of the expressions in Eq. (6-1) and (6-2). Each equation can be mechanized by means of four three-input AND gates whose outputs are combined in a four-input OR gate. There is an additional requirement which, though not shown explicitly, is implied by these equations. A delay must be provided at the output of the combinational circuit that mechanizes Eq. (6-2) so that the carry produced at time $t-1$ can be combined with the bits of the addends appearing at time i. Actually Eq. (6-2) would not be mechanized directly because it is a redundant form. This is shown by the following algebraic manipulation which leads to the simplified expression for C_i given by Eq. (6-3)

$$\begin{aligned} C_i &= \bar{A}_i B_i C_{i-1} + A_i \bar{B}_i C_{i-1} + A_i B_i \bar{C}_{i-1} + A_i B_i C_{i-1} \qquad (6\text{-}2)\\ &= B_i C_{i-1}(\bar{A}_i + A_i) + A_i \bar{B}_i C_{i-1} + A_i B_i \bar{C}_{i-1}\\ &= B_i C_{i-1} + A_i \bar{B}_i C_{i-1} + A_i B_i \bar{C}_{i-1}\\ &= C_{i-1}(B_i + A_i \bar{B}_i) + A_i B_i \bar{C}_{i-1}\\ &= C_{i-1}(A_i + B_i) + A_i B_i \bar{C}_{i-1}\\ &= A_i(C_{i-1} + \bar{C}_{i-1} B_i) + C_{i-1} B_i\\ &= A_i(B_i + C_{i-1}) + C_{i-1} B_i\\ &= A_i B_i + A_i C_{i-1} + C_{i-1} B_i. \qquad (6\text{-}3) \end{aligned}$$

Equation (6-3), $C_i = A_i B_i + A_i C_{i-1} + C_{i-1} B_i$, could have been obtained directly from consideration of the binary addition process, by noting that a carry is produced whenever any two of the inputs are equal to 1, regardless of the value of the third input.

Since the output of a switching circuit can, in general, be used as the input to a number of points, it is economical, wherever possible, to construct required functions by incorporating and modifying simpler functions already formed. Therefore, it would be desirable to form the network for mechanization of Eq. (6-1) by means of an addition to the network for mechanizing Eq. (6-3). One way this could be realized would be to utilize the following expression

$$S_i = \bar{C}_i(A_i + B_i + C_{i-1}) + A_i B_i C_{i-1}. \qquad (6\text{-}4)$$

Comparison of column 9 with column 4 in Table 6.3 shows that Eqs. (6-1) and (6-4) are equivalent.

TABLE 6.3. Truth Table for Generation of $S_i = \bar{C}_i(A_i + B_i + C_{i-1}) + A_iB_iC_{i-1}$

1 A_i	2 B_i	3 C_{i-1}	4 C_i	5 $\bar{C}_i$	6 $(A_i+B_i+C_{i-1})$	7 $\bar{C}_i(A_i+B_i+C_{i-1})$	8 $A_iB_iC_{i-1}$	9 S_i
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	1	0	1
0	1	0	0	1	1	1	0	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	1	0	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	0	1	1

Figure 6.13 shows a schematic of an adder which produces the sum and

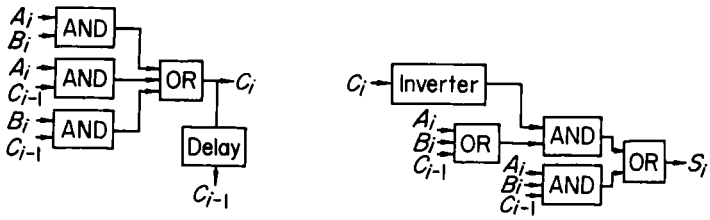


FIG. 6.13. An adder based on mechanization of the equations:

$$C_i = A_iB_i + A_iC_{i-1} + C_{i-1}B_i ,$$
$$S_i = \bar{C}_i(A_i + B_i + C_{i-1}) + A_iB_iC_{i-1}$$

carry in accordance with Eqs. (6-3) and (6-4). A comparison of the mechanizations of S_i by means of Eqs. (6-1) and (6-4) shows the following. Mechanization of Eq. (6-1) calls for four three-input AND gates and one four-input OR gate. Mechanization of Eq. (6.4) calls for one three-input and one two-input AND gate, one three-input and one two-input OR gate, and an inverter. While the latter arrangement requires fewer gating elements (10 compared to 16) it calls for a three level OR–AND–OR circuit, compared to a two level circuit for the former.

In all the adders based on logical operations which have been described thus far, a simple delay element was employed to cause the carry from one order to be combined with the addend bits of the next more significant order. In other adders, advantage may be taken of the fact that a flip-

flop's inherent delay in switching and its storage capability can be used to delay and store a carry. Figure 6.14 shows three different adders

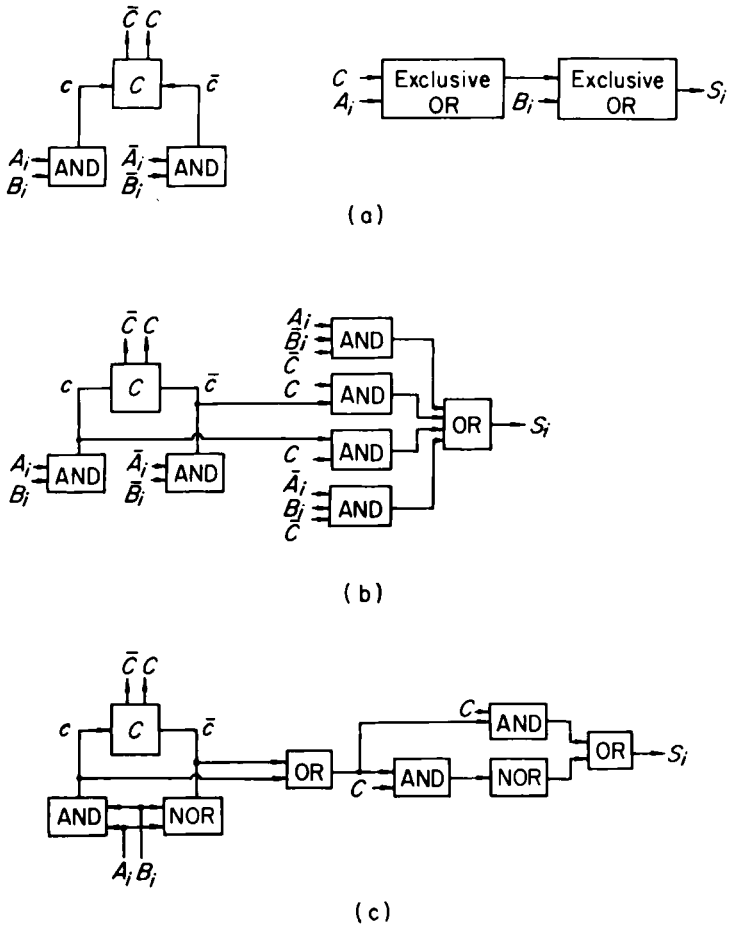


FIG. 6.14. Three adders utilizing a carry flip-flop

in which the required delay of the carry is obtained by use of a flip-flop. At each bit time, the output of the carry flip-flop will indicate whether a carry was produced by the next less significant order. For a set-reset type of flip-flop the input signals to the carry flip-flop are

$$c = A_iB_i \qquad c = \bar{A}_i\bar{B}_i.$$

The states C and $\bar{C}$ indicate that a carry was or was not produced during

the preceding bit time. The input equations to the flip-flop are derived by observing the following. Since the carry at the beginning of an addition is 0, the flip-flop is set initially to state $\bar{C}$. When A_iB_i has the value 1, a carry must be produced, so the carry flip-flop is set to state C. If a carry is produced in one position, it will be produced in the next more significant one unless the case $\bar{A}_i\bar{B}_i$ occurs. In other words, the cases $A_i\bar{B}_i$ and $\bar{A}_iB_i$ produce a carry signal if and only if there has been a carry from the preceding position. Therefore, if $\bar{A}_iB_i$ or $A_i\bar{B}_i$ occur, the state of the carry flip-flop is correct and need not be altered. When $\bar{A}_i\bar{B}_i$ occurs, a carry cannot be produced so the carry flip-flop must be reset to state $\bar{C}$.

The three arrangements shown in Fig. 6.14 differ in the nature of the logical circuits used in conjunction with the carry flip-flop C. Fig. 6.14(a) uses only AND and EXCLUSIVE OR circuits; Fig. 6.14(b) uses only AND and OR circuits; Fig. 6.14(c) uses AND, NOR, and OR circuits, and would be useful in the case where the complement of A and B were not available as an input to the adder. Any of a number of similar arrangements can be specified. The one chosen will depend on the types of logical building blocks one chooses to use, restrictions imposed on the maximum level of gating circuits, and the form in which the bits of the addends are available. The building blocks chosen will, in turn, depend on a number of factors: the frequency of operation, the relative reliability and cost of combinations of specified building blocks for a given application, the size and power requirements of different building blocks, etc.

Each of the adders described in this section can be differentiated from preceding switching networks in that for every combination of input variables there are two distinct output signals, namely, the sum and the carry. Such a circuit is considered a multiple output switching network. The classification is arbitrary, since both the sum and carry may be generated separately, i.e., by two single output switching networks. However, the important point is that by considering the adder circuit as a unit, i.e., a multiple output switching network, certain duplications of circuitry may be avoided (e.g., see Eq. (6-4) and the discussion preceding it) since often the same terms or factors may be required as part of both output functions.

It should be emphasized at this point that, while block diagrams sometimes serve as a convenience, they are not necessary to describe logical configurations. If suitable symbols have been provided for all switching and storage elements, all sequential circuits may be described logically by means of Boolean algebraic equations that describe the input-output relations in these circuits. The indicated uses for block diagrams and logical equations will be considered in more detail in Chapter 7.

6.1.2.2. *Use of a Delay Line for Augend–Sum Storage*

In the scheme for serial addition shown in Fig. 6.15, it is assumed

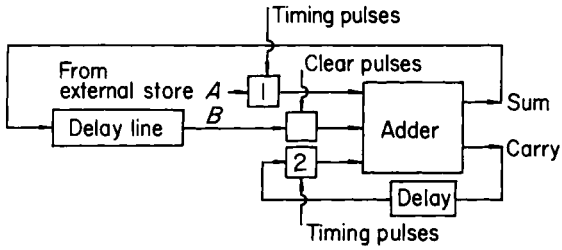


FIG. 6.15. A serial adder utilizing a delay line for augend–sum storage

that the number $B_1 \ldots B_n$ is stored in a delay line storage device of length nD, where D indicates a unit delay, and that the bits $A_1 \ldots A_n$ come from an external store. The bits on each input line are represented by pulses. The gates 1 and 2 will not pass incident bits $A_1 \ldots A_n$ nor carry bits, except at times prescribed by the arrival of suitable timing pulses. This enables any particular number to be selected from the external store and added to that already contained in the delay line. The clear pulses are applied when it is desired to erase the contents of the delay line. If they are applied during the time interval when there is an input on A, then the contents of the delay line B are replaced by the input on line A, since the arithmetic sum $A + 0$ is formed by the adder and entered into delay line B. This is effectively the same as if the contents of the delay line were first cleared and then the input on A added, except that it saves the time required to separately clear the delay line.

6.1.2.3. *Use of a Shift Register for Augend–Sum Storage*

One or both inputs as well as the output of a serial adder may be stored in a circulating memory. However, the use of a shift register for at least one of the two numbers permits the sum to be stored for further manipulation if desired. In this way it can serve as an accumulator, since as the bits of the augend are read into the adder, room is provided for the sum bits.

In Fig. 6.16, the bits of one number $B_1 \ldots B_n$ are stored in a static register, the contents of which can be shifted, one bit at a time, to the right on receipt of clock pulses if the signal for a shift is present on the line marked "shift." Nothing occurs except when the shift gates 1 through $n-1$ and the gates s, c are pulsed. Then the sum is produced by the adder

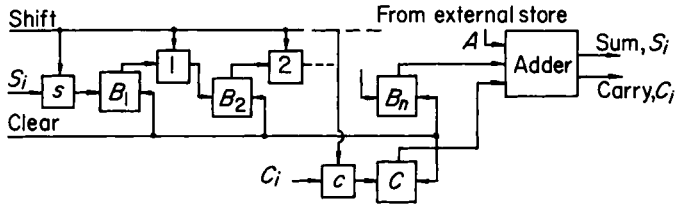


FIG. 6.16. A serial adder utilizing a shift register for augend–sum storage

bit by bit upon receipt of successive clock pulses. Each bit of the sum is successively shifted into flip-flop B_1 and the carry for the next stage, if any, is entered into the carry flip-flop C. The carry flip-flop is cleared prior to an addition.

The scheme of Fig. 6.17 is a variation of the arrangement in Fig. 6.16.

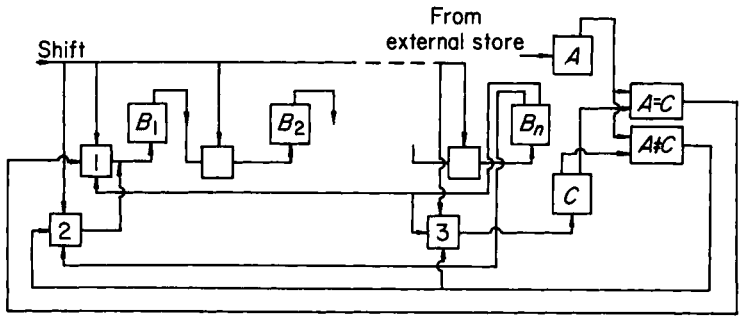


FIG. 6.17. An alternate logical scheme for serial addition utilizing a shift register for augend–sum storage

It is based on considering the sum and carry in terms of the variables B_i and C_{i-1}, as described in Table 6.4. From Table 6.4, it is evident that if $A_i = C_i$, then the sum and carry are simply B_i and C_{i-1}, respectively; if $A_i \neq C_i$ they are $\bar{B}_i$ and $\bar{B}_i$ respectively. In Fig. 6.17, the flip-flop A receives the bits of one addend $A_1 \ldots A_n$ from an external store, and flip-flop C stores the carry bits. Upon receipt of shift pulses, the contents of the B register are shifted right and the output of B_n is sent either directly via gate 1 or inverted via gate 2 into B_1. Concurrently, the output of B_n is sent directly to C via gate 3, when gate 2 is actuated, i.e., if $A \neq C$.

TABLE 6.4.

A_i	C_{i-1}	B_i	Sum	Carry
0	0	0	$0 = B_i$	$0 = C_{i-1}$
0	0	1	$1 = B_i$	$0 = C_{i-1}$
1	1	0	$0 = B_i$	$1 = C_{i-1}$
1	1	1	$1 = B_i$	$1 = C_{i-1}$
1	0	0	$1 = \bar{B}_i$	$0 = B_i$
1	0	1	$0 = \bar{B}_i$	$1 = B_i$
0	1	0	$1 = \bar{B}_i$	$0 = B_i$
0	1	1	$0 = \bar{B}_i$	$1 = B_i$

6.1.2.4. *A Serial Accumulator*

The arrangement shown in Fig. 6.18 indicates how bistable counter

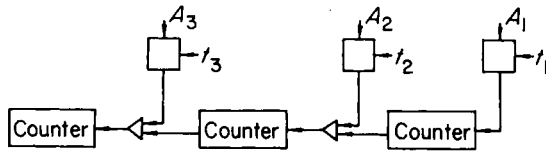


FIG. 6.18. A serial accumulator

circuits can be utilized to form an accumulator. The circuits interconnecting the counters do not function as logical OR gates, but only as buffers, since there is never a signal on both inputs simultaneously. The bits of the incident number, $A_1 \ldots A_n$, are applied to each stage sequentially. Each counter accomplishes two functions. First, upon receipt of an input signal it adopts a state representing the value of the sum bit for that stage. Secondly, it produces an output signal if there is a carry. Before an incident bit is applied to the input of any stage, time must be allowed for any carry produced in the next less significant stage to have passed the succeeding stage.

6.1.2.5. *Parallel Binary Adders*

In a parallel adder all the bits of a word are accepted by a static register at the same time. There are as many sets of input lines as there are bits, and an adding circuit is associated with each bit. To allow the description of some specific parallel adders, certain assumptions will be made. Assume that the addend is stored in flip-flops A_i, the augend

in flip-flops B_i, and that it is desired to store the sum in the flip-flops B_i, (flip-flops A_i, B_i are assumed to be of the trigger or complement type). The input equations to the flip-flops B_i may be derived by noting in Table 6.5 the states of A_i and C_{i-1} at time t, for those cases where there is to be a change in B_i at time $t + 1$.

TABLE 6.5.

Time t			Time $t + 1$	
A_i	B_i	C_{i-1}	C_i	B_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The result is: $b_i = \bar{A}_i C_{i-1} + A_i \bar{C}_{i-1}$. The combinational circuit for C_i is formed from an expression which states the set of values of A_i, B_i, C_{i-1}, for which the variable C_i has the value 1. One form for this expression is

$$C_i = A_i B_i + A_i C_{i-1} + B_i C_{i-1}$$

Since there is no carry in the least significant place, i.e., $C_0 = 0$, the expression for C_1 is simply $C_1 = A_1 B_1$.

If the input equations to the flip-flops B_i are examined, it is apparent that each b_i is a function of C_{i-1} (or $\bar{C}_{i-1}$) which, in turn, is a function of $A_1 \ldots A_{i-1}$, $B_1 \ldots B_{i-1}$. From this it is evident that if the input equations to a flip-flop B_i were made an explicit function of $A_1 \ldots A_i$, $B_1 \ldots B_i$, there would, in general, be so many terms that the corresponding gating circuit would be impractical to construct. One solution is to insert some power amplifying device, e.g., a cathode or emitter follower, for each C_i and $\bar{C}_i$. Even so, a long time would be required for the transients to die out after a new addend and augend appear in the A and B registers. The maximum length of this transient will determine the maximum time interval from arrival of the operands until the sum is available and b_i can be made to operate. The duration of this transient is proportional to the number of bits in each operand. The transient may be eliminated by storing the bit-by-bit sum in B_i, the carry in A_i, and then proceeding in the next

bit time as if A_i and B_i still contained an addend and an augend. Each step may then be considered as a half-addition. The addition of 11 and 6 would be as shown in Example 6.2.

Example 6.2

Decimal number	Contents of A_i, B_i		Clock period
11	01011	A_i	
6	00110	B_i	1
	01101	B_i	
	1	A_i	2
	01001	B_i	
	1	A_i	3
	00001	B_i	
	1	A_i	4
17	10001	B_i	
	00000	A_i	5

The truth table for the half adder is shown in Example 6.3.

Example 6.3

Time t		Time $t+1$	
A_i	B_i	A_i	B_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

From Example 6.3 the following difference equations may be derived

$$(B_i)_{t+1} = (A_i\bar{B}_i + \bar{A}_iB_i)_t$$

$$(A_i)_{t+1} = (A_iB_i)_t.$$

Input equations to the flip-flops A_i, B_i that will cause these relations to be satisfied are

$$b_i = A_i$$

$$a_i = A_i\bar{B}_i$$

These input equations are derived from the preceding truth table by noting the states of both A_i and B_i at time t which lead to a change in A_i and B_i, respectively, at time t_{i+1}.

The end of the addition is indicated when all $A_i = 0$, i.e., there are no more carries. The maximum time for an addition is, in terms of clock periods, equal to the number of bits in the operands, and requires the same time as a serial adder. The average time, however, is less for a series of additions on operands that may be considered random numbers. The average number of successive carries that will occur in the addition of two 40-bit numbers containing random bits is $\approx$ 4.6.* Although its logic is simple, this adder is not very efficient when the large number of components that has been added for a slight decrease in the time required for an addition is considered.

We will now consider a parallel adder in which the maximum and average addition times are reduced further by means of a more complex logic. The operation of this adder is based on considering each operand as $n/2$ adjacent groups of two bits each (where n = the total number of bits in each operand). The first step in the addition process consists of storing the sum (modulo 2) of the operands within each two-bit group in the two A flip-flops within each group. Also, each odd-numbered (i.e., less significant) B flip-flop in a group of two, is set to 1 if the less significant two-bit group produces a carry. Each even-numbered B flip-flop is reset to 0. Then the process is continued with a series of half-additions. The logical sequence from time t to $t + 1$ is described in Table 6.6.

---

* See Burks, Goldstine, and von Neumann. "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument. Princeton Institute for Advanced Study, 1947.

TABLE 6.6 Successive states of flip-flops in a parallel adder

Time t				Time $t+1$			
A_i	A_{i+1}	B_i	B_{i+1}	A_i	A_{i+1}	B_{i+2}	B_{i+1}
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0
0	1	0	0	0	1	0	0
0	1	0	1	0	0	1	0
0	0	1	0	1	0	0	0
0	0	1	1	1	1	0	0
0	1	1	0	1	1	0	0
0	1	1	1	1	0	1	0
1	0	0	0	1	0	0	0
1	0	0	1	1	1	0	0
1	1	0	0	1	1	0	0
1	1	0	1	1	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	0	1	0
1	1	1	0	0	0	1	0
1	1	1	1	0	1	1	0

From Table 6.6 the following equations may be derived

$$(A_i)_{t+1} = (\bar{A}_i B_i + A_i \bar{B}_i)_t$$

$$(A_{i+1})_{t+1} = \bar{A}_{i+1}(\bar{A}_i \bar{B}_i B_{i+1} + \bar{A}_i B_i B_{i+1} + A_i \bar{B}_i B_{i+1} + A_i B_i \bar{B}_{i+1}) + A_{i+1}(\bar{A}_i \bar{B}_i \bar{B}_{i+1} + \bar{A}_i B_i \bar{B}_{i+1} + A_i \bar{B}_i \bar{B}_{i+1} + A_i B_i B_{i+1})$$

$$(B_{i+2})_{t+1} = (A_{i+1}\, B_{i+1})_t + [A_i B_i\, (A_{i+1} + B_{i+1})]_t$$

$$(B_{i+1})_{t+1} = 0.$$

If the flip-flops A_i, B_i are of the set-reset type, the input equations that cause these relations to be satisfied are

$$b_{i+2} = A_{i+1}B_{i+1} + A_i B_i (A_{i+1} + B_{i+1})$$

$$\bar{b}_{i+2} = (\bar{A}_i + \bar{B}_i)\,(\bar{A}_{i+1} + \bar{B}_{i+1}) + \bar{A}_{i+1}\bar{B}_{i+1}$$

$$\bar{b}_{i+1} = 1 \qquad b_{i+1} = 0$$

$$a_i = \bar{a}_i = B_i$$

$$a_{i+1} = \bar{a}_{i+1} = B_{i+1}\overline{(A_i B_i)} + \bar{B}_{i+1} A_i B_i.$$

These input equations indicate that it would be appropriate to use two-input (set–reset) flip-flops for the B_i and single-input (trigger) flip-flops for the A_i. The maximum time for an addition is, in terms of clock periods, equal to $n/2$ where n is the number of bits in each operand. The average time is considerably lower. The end of an addition is indicated by all odd-numbered B_i being equal to zero. The maximum time may be reduced to n/x clock periods by dividing each operand into groups of x bits each. However, the formulation for the carry becomes more involved as x increases. The arithmetic process is clarified in Example 6.4.

Example 6.4

Decimal equivalent							Clock period
11	0	1	0	1	1	A_i	1
6	0	0	1	1	0	B_i	
13	0	1	1	0	1	A_i	2
4	0	0	1	0	0	B_i	
1	0	0	0	0	1	A_i	3
16	1	0	0	0	0	B_i	
17	1	0	0	0	1	A_i	4
0	0	0	0	0	0	B_i	

6.1.2.6. A Parallel Adder with Carry Flip-flops

This discussion on parallel adders will be concluded with a description of a parallel binary adder which can readily be modified to serve as a parallel decimal adder (as described in Section 6.1.3.2). As in Section 6.1.2.5, assume that the addend is stored in flip-flops A_i, the augend in flip-flops B_i, and that their sum will be stored in flip-flops B_i. It is also assumed that there is one carry flip-flop C_i after every fourth bit of the operands. More frequent carry flip-flops would reduce only slightly the number of gating elements required, at the cost of increasing the time required for an addition. The B_i flip-flops are assumed to be of the single input complement type and the C_i flip-flops of the set–reset type.

The mode of operation of this adder may be outlined as follows: (1) Upon the arrival of the initiating clock pulse t_0 there is an input signal to each B_i flip-flop if the corresponding A_i flip-flop is in the 1 state. For example, for the first group of four bits (B_0, B_1, B_2, B_3)

$$(b_0)_{t=0} = A_0 t_0 \qquad (b_1)_{t=0} = A_1 t_0$$

$$(b_2)_{t=0} = A_2 t_0 \qquad (b_3)_{t=0} = A_3 t_0.$$

This step actually consists of forming the sum (modulo 2) in each column and, for the time being, ignoring the generation and propagation of any carries. (2) Upon the arrival of the next clock pulse t_1, each flip-flop B_i receives an input signal if the original addend and augend are such that a carry would have been propagated to stage B_i from B_{i-1}, where both B_i and B_{i-1} are within a given group of four bits. A carry can be propagated to a given stage only if: (a) both operands are 1 in the preceding stage, in which case the value of the operands in less significant places is of no consequence; or (b) there is an uninterrupted sequence of less significant columns in which the value of at least one of the operands is 1, immediately followed by a lesser significant column in which both operands have the value 1. The Boolean algebraic statement of the original conditions capable of producing carry input signals $(b_i)_c$ to the flip-flops B_i are, for the first group of four bits (B_0, B_1, B_2, B_3)

$$\begin{aligned}
(b_1)_c &= A_0B_0t_0 \\
(b_2)_c &= [A_1B_1 + (A_1\bar{B}_1 + \bar{A}_1B_1)A_0B_0]t_0 \\
(b_3)_c &= [A_2B_2 + (A_2\bar{B}_2 + \bar{A}_2B_2)A_1B_1 \\
&\quad + (A_2\bar{B}_2 + \bar{A}_2B_2)(A_1\bar{B}_1 + \bar{A}_1B_1)A_0B_0]t_0.
\end{aligned} \tag{6-5}$$

Remember that at time t_1, the B_i flip-flops no longer contain the original augend, for each B_i flip-flop was set to its complementary state wherever the corresponding A_i flip-flop was in the 1 state. However, this presents no problem since at t_1 the A_i still contain the original addend, and to express the input signals to the B_i flip-flops at time t_1, it is only necessary to complement the value of the B_i as given in Eqs. (6-5) wherever the corresponding A_i is in the 1 state, and to replace t_0 by t_1. The input equations b_i at time $t = 1$ are, accordingly

$$\begin{aligned}
(b_1)_{t=1} &= A_0\bar{B}_0t_1 \\
(b_2)_{t=1} &= [A_1\bar{B}_1 + (A_1B_1 + \bar{A}_1B_1)A_0\bar{B}_0]t_1 \\
&= (A_1\bar{B}_1 + B_1\bar{B}_0A_0)t_1 = A_1\bar{B}_1t_1 + B_1(b_1)_{t=1} \\
(b_3)_{t=1} &= [A_2\bar{B}_2 + (A_2B_2 + \bar{A}_2B_2)A_1\bar{B}_1 \\
&\quad + (A_2B_2 + \bar{A}_2B_2)(A_1B_1 + \bar{A}_1B_1)A_0\bar{B}_0]t_1 \\
&= (A_2\bar{B}_2 + B_2\bar{B}_1A_1 + B_2B_1\bar{B}_0A_0)t_1 \\
&= A_2\bar{B}_2t_1 + B_2(b_2)_{t=1}.
\end{aligned} \tag{6-6}$$

(3) Upon arrival of clock pulse t_1 each carry flip-flop is to be set if a carry would be propagated out of a given group of four bits upon addition

of the original addend and augend bits. The conditions for the propagation of a carry to a given stage were stated in item (2). The set signal for the first carry flip-flop C_3 is

$$\begin{aligned}(c_3)_{t=1} &= (A_3\bar{B}_3 + B_3\bar{B}_2A_2 + B_3B_2\bar{B}_1A_1 + B_3B_2B_1\bar{B}_0A_0)t_1 \\ &= A_3\bar{B}_3t_1 + B_3(b_3)_{t=1}.\end{aligned}$$

(4) Up to this point each group of four adjacent bits has been considered almost as if it were isolated from the others. The carry flip-flops, C_{i+3}, provide the necessary links. In general, the carry produced by one group of four bits may affect both the sum and carry bits in more significant places. Accordingly, additions must be made to the B_i, B_{i+1}, B_{i+2}, B_{i+3}, and C_{i+3} flip-flop input equations for all groups except the least significant one. Therefore, for $i = 4, 8, 12 \ldots (n-3)$, where $n = 39$ for an assumed register of 40 bits (and where $t_{2/10}$ designates the interval from the appearance of t_2 through t_{10}) the flip-flop input equations are:

$$\begin{aligned}b_i &= A_it_0 + C_{i-1}t_{2/10} \\ b_{i+1} &= A_{i+1}t_0 + A_i\bar{B}_it_1 + B_iC_{i-1}t_{2/10} \\ b_{i+2} &= A_{i+2}t_0 + A_{i+1}\bar{B}_{i+1}t_1 + B_{i+1}(b_{i+1})_{t=1} \\ &\quad + B_{i+1}B_iC_{i-1}t_{2/10} \\ b_{i+3} &= A_{i+3}t_0 + A_{i+2}\bar{B}_{i+2}t_1 + B_{i+2}(b_{i+2})_{t=1} \\ &\quad + B_{i+2}B_{i+1}B_iC_{i-1}t_{2/10} \\ c_{i+3} &= A_{i+3}\bar{B}_{i+3}t_1 + B_{i+3}B_{i+2}B_{i+1}B_iC_{i-1}t_{2/10}\end{aligned} \tag{6-7}$$

where $t_{2/10}$ represents the logical sum of t_2 through t_{10}. Flip-flops C_3, C_7, $C_{11}, \ldots C_{35}$ are reset by t_2, t_3, $t_4, \ldots t_{10}$, respectively.

The end of the addition process is indicated by sensing completion of all carries, i.e., by detecting the condition $\bar{C}_3\bar{C}_7\bar{C}_{11}\bar{C}_{15} \ldots \bar{C}_{35}t_{2/11}$.

This permits an addition to be performed in an asynchronous manner. For a large number of additions this allows an average time for addition which is considerably less than that required for synchronous operation, where a maximum time interval must be assigned for each addition. The speed of addition could be increased further if fewer carry flip-flops were used. However, this would be very costly in the number of additional gating elements required. If more carry flip-flops were used, the speed would be decreased appreciably while the number of gating elements required would be reduced only slightly.

6.1.2.7. Parallel Adders with Full Length Anticipatory Carry Chains

The adder shown in Fig. 6.19, becomes operative upon application of

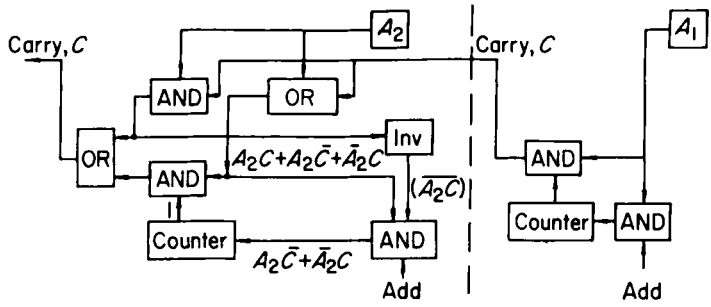


FIG. 6.19. A parallel adder with bistable counter storage and an anticipatory carry chain

an "add" command, simultaneously applied to the inputs of all stages. The flip-flops A_i may comprise either a buffer register or a shift register with serial or parallel read-in. Carry pulses generated in each stage are propagated only through gates. The accumulator is comprised of a set of bistable counters B_i. Each B_i is triggered only after a carry has passed that stage (assuming a carry is propagated to it). The maximum time to produce the sum is dependent on the speed of the carry propagation circuit.

For a fast addition operation in a parallel computer, carry pulses should be passed through as few gates and other circuits as possible. Also, these gates and other circuits should be fast operating. The adder shown in Fig. 6.20 operates in two major steps: First, carries are gen-

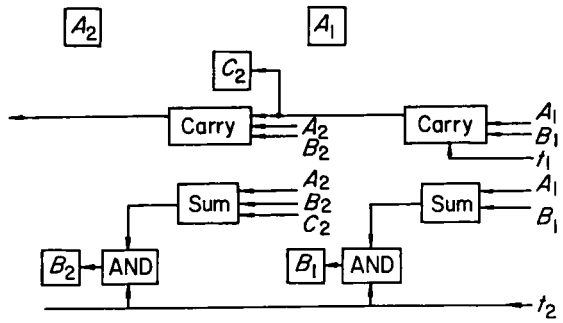


FIG. 6.20. A parallel adder with flip-flop storage and an anticipatory carry chain

erated and propagated through carry gates from less to more significant stages; as a carry pulse reaches any particular stage it is also stored by means of a carry flip-flop C. Secondly, when all carries have been propagated, a pulse t_2 is applied to all B_i input gates simultaneously, as a result of which the arithmetic sum of A and B is stored in B. The maximum time to produce the sum is dependent on the speed of the carry propagation circuitry.

6.1.2.8. A Parallel Adder with a Fast Carry

Even though, in a parallel adder, the addends are applied simultaneously to all stages, there will be a delay before the most significant stage of the adder assumes its final value, because of the serial nature of carry propagation. Most synchronous computers employ carry circuits in which the full length carry time must be allowed in each addition. The required time interval is provided by a separate timing device, with a safety margin to allow for tolerances in both the carry circuit and timing device. By using the carry circuit to time its own full length carry time, an improvement may be effected in the timing reliability of the carry system.

A significant decrease in the time required for carry propagation can be achieved if advantage is taken of the fact that on the average the length of a one's carry sequence is only a small fraction of the maximum sequence (being 4.6 stages in a 40 bit addition).

A simple carry circuit is shown in Fig. 6.21(a). It cannot provide its own timing because of the carry interruptions and starts caused by columns where $\bar{A}\bar{B}$ and AB exist, respectively. The arrangement shown in Fig. 6.21(b) provides for two carry lines into and out of each stage.

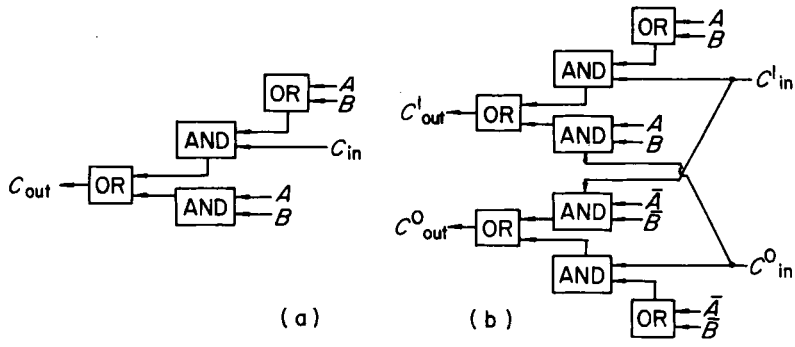


FIG. 6.21. (a) A simple carry circuit, (b) A circuit to propagate 0 and 1 carries

One line C^1, corresponds to the usual carry line. The other line C^0, has a signal when there is no carry from the next less significant stage. By providing for 0 carries as well as 1 carries through the use of separate carry chains, there results a circuit that can provide its own timing. At the beginning of an addition, both carry lines are off. This condition will be met if both carry inputs to the least significant stage are held off. The carry propagation is begun by applying a signal to the C^0 line of the least significant stage. This carry will then proceed down the 0 chain until it reaches a stage where AB is 1, at which point the carry switches over to the 1 chain. Similarly, it will proceed down the 1 chain until it reaches a stage where $A\bar{B}$ is 1. Finally, it will emerge from the most significant stage as either a C^0 or C^1 to signal the end of the n-stage carry. The carry will always pass serially through all n stages.

If the truth table for the formation of a carry is arranged as shown in Table 6.7, it becomes apparent that if either AB or $A\bar{B}$ exists, C_{out} is independent of C_{in}

TABLE 6.7. Truth table for carry generation

C_{in}	A	B	C_{out}	
0	0	1	0	
0	1	0	0	$= C_{in}$
1	0	1	1	
1	1	0	1	
0	0	0	0	
1	0	0	0	$= AB$
0	1	1	1	
1	1	1	1	

As a result, the cross coupling shown in Fig. 6.21(b) may be eliminated, and it becomes apparent now that there are, in general, places within an n-stage addition operation where the sum and carry bits may be formed independently of information in preceding bit places. In constructing an adder* based on this observation, the state of the carry lines should be

* Gilchrist, B., Pomerene, J. H., and Wong, S. Y. [1955], Fast carry logic for digital computers, *IRE Trans. El. Computers*, **4**, 133-136.

viewed as off or 1 for C^1 and off or 0 for C^0. Both carry lines are off at the start of an addition. This is enforced for the interior stages by an explicit parallel inhibition on the lines or by operating on the AB and $\bar{A}\bar{B}$ inputs. Carries are begun by releasing the inhibitions on all stages, including the selected carry into the least significant stage. At this time, carry sequences will arise from the selected input carry, and from every interior stage having $\bar{A}\bar{B}$ or AB. Thus the serial aspect of the carry is restricted to sequences of stages for which $A \neq B$.

Feeding the $C^1{}_{\text{out}}$ and $C^0{}_{\text{out}}$ lines in each stage to the input of an OR gate, and the output of all such OR gates to an n-input AND gate, permits a carryless determination of the equality of two addends, this mode being obtained by not releasing the parallel carry inhibitions. In this case, an output is obtained from the carry completion gate if, and only if, two addends are equal. (Other comparators are described in Section 6.1.4.3).

6.1.3. Decimal Addition

6.1.3.1. Serial Decimal Adders

When designing a decimal adder, one must choose first of all a binary code group to represent each decimal digit. Any decimal digit can be represented by the states of a group of four or more bistable elements. A number so represented is said to be in a binary-coded decimal form. There are many forms of binary-coded decimal representation, differing in the number of bits per group (usually four or five) and the weights assigned to each position in the group. A total of 70 weighted four-bit codes, including those with negative weights have been found. The most obvious binary-coded decimal representation is referred to as the straight binary, or (8-4-2-1), decimal code. It is shown in Table 6.8. In this code, the representation of 694 is

6	9	4
(0110)	(1001)	(0100).

Whenever binary-coded decimals are operated upon, attention must be paid to the fact that each group is distinct. For example, consider the binary-coded decimal representation of 694. If it were multiplied by two, simply by a single shift left, there would result

(0110) (1001) (0100). $\times$ 10. = (1100)* (0010) (1000). $= ?\ 28$

The error in the result arises from neglecting the fact that, if the operation

* In the (8-4-2-1) code (1100), the binary equivalent of 12, is not defined.

TABLE 6.8. The straight binary decimal code

Decimal	(8-4-2-1) Decimal code			
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

on a particular binary-coded decimal digit produces a number greater than nine, a carry is produced which must be added to the next most significant digit, and the digit producing the carry must also be adjusted. Table 6.9 shows that a multiplication by two can be accomplished by a shift left operation only if a digit is $\leqslant 4$.

TABLE 6.9.

$2 \times 0000 =$	$0000 =$	0
$2 \times 0001 =$	$0010 =$	2
$2 \times 0010 =$	$0100 =$	4
$2 \times 0011 =$	$0110 =$	6
$2 \times 0100 =$	$1000 =$	8
$2 \times 0101 = 1$	$0000 = 1$	0
$2 \times 0110 = 1$	$0010 = 1$	2
$2 \times 0111 = 1$	$0100 = 1$	4
$2 \times 1000 = 1$	$0110 = 1$	6
$2 \times 1001 = 1$	$1000 = 1$	8

The use of binary-coded decimals can relieve the user from the task of converting numbers from decimal to binary representation prior to inserting them into a machine, and of converting the binary output of the machine. For example, ten numerical keys can be provided on an input device, so wired that when the operator presses any one of them the corresponding binary-coded decimal signal is inserted into the machine.

Similarly, each of the ten binary-coded decimal signals can be used to trip, say, a corresponding numerical key in an output typewriter. If the internal arithmetic units of the computer operate in the binary system, binary-coded decimal inputs to the machine must be converted to true binary numbers before being operated upon by the arithmetic unit. In Section 6.4 the subject of conversion between binary and binary-coded decimal representation is considered in more detail.

Though there are many binary-coded decimal representations, in practice the choice is usually confined to one of a small set of four and five bit codes. In the discussion following, only four-bit decimal codes will be considered.

Assuming a particular code has been chosen, the general approach for testing its suitability in an adder is to form a truth table. In this case, there would be 200 input conditions, corresponding to all possible combinations of values of the two addends and the carry from the preceding stage. For each possible input combination, the value of five output bits will be defined, namely, the four bits of the sum digit and the carry bit. A simpler procedure may be used for certain special choices of the four bit code. Two of the most commonly used codes will be described next.

In the straight binary decimal code, four binary places are assigned weights of 1, 2, 4, and 8 respectively. A decimal adder to operate on these decimal code groups can be formed from a binary adder with the following simple modification. When the sum digit produced in any decimal place is ten or greater, indicating that a carry must be propagated from one code group to the next, the carry must be generated and the sum digit itself corrected to yield a value less than ten.

Another commonly used four bit decimal code is referred to as the excess-three code. It is a nonweighted code wherein each decimal digit d is represented by a code group which, if interpreted as a conventional binary number would represent $d + 3$. This is shown in Table 6.10.

The excess-three code has certain useful properties. First of all, it is self-complementing, i.e., the nine's complement (see Section 6.1.4.2.3.) can be obtained simply by interchanging ones and zeros. Also, because there is at least one 1 in the representation of each digit, the conditions of zero or no digit can be readily distinguished. Another advantage is that a carry bit occurs automatically out of the most significant position. This follows because, if the sum of two decimal numbers is $\geqslant 10$, the sum of their excess-three code representation must be $\geqslant 16$. However, it is necessary to correct the sum digit whether or not there is an overflow. Specifically, if the sum digit is $\geqslant 16$, three must be added to it, (since the sum does not have an excess-three bias), if it is < 16, three

TABLE 6.10

Decimal	Binary excess-three
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

must be subtracted (since an excess three has been added twice). This necessitates additional storage elements for storing the sum of each excess-three code group, so that it may be corrected. Finally, since for every entry in Table 6.10 there is another entry where the 1's and 0's are interchanged, a decoder used to obtain a 1 out of 10 representation will place equal loading on both outputs of the flip-flops holding the excess-three code.

The logical design of a serial excess-three code decimal adder will now be described. Assume that the bits of the two addends appear serially at successive bit times t in two flip-flops, A, B. Successive sum and carry bits are produced and stored in flip-flops S and C respectively. The times at which the least significant bits of each code group appear in A, B will be indicated by a timing signal t_1. Four additional flip-flops S_1, S_2, S_3, S_4, are provided in which to form the corrected sum. These flip-flops actually serve two distinct purposes. First, they act as a shift register to which successive values formed in S are sent, and also as a correction register in which the corrected sum of each code group is formed. When the corrected sum is formed, these flip-flops again act as a shift register, transmitting their contents back to the main storage unit. At time t_1 any given code group will be stored in flip-flops S, S_1, S_2, S_3. This number must be corrected according to the value of flip-flop C at time t_1. However, the corrected value must be placed in flip-flops S_1, S_2, S_3, S_4, since at the next bit time S must be used to store the sum bit of the next two bits appearing in flip-flops A, B. The configurations to be assumed by the flip-flops S_1,S_2, S_3, S_4 are shown in Table 6.11.

TABLE 6.11.

	Time t_4				Time t_1			
	S	S_1	S_2	S_3	S_1	S_2	S_3	S_4
				Subtract 3				
	0	1	1	0	0	0	1	1
	0	1	1	1	0	1	0	0
	1	0	0	0	0	1	0	1
	1	0	0	1	0	1	1	0
$C = 0$	1	0	1	0	0	1	1	1
	1	0	1	1	1	0	0	0
	1	1	0	0	1	0	0	1
	1	1	0	1	1	0	1	0
	1	1	1	0	1	0	1	1
	1	1	1	1	1	1	0	0
				Add 3				
	0	0	0	0	0	0	1	1
	0	0	0	1	0	1	0	0
	0	0	1	0	0	1	0	1
	0	0	1	1	0	1	1	0
$C = 1$	0	1	0	0	0	1	1	1
	0	1	0	1	1	0	0	0
	0	1	1	0	1	0	0	1
	0	1	1	1	1	0	1	0
	1	0	0	0	1	0	1	1
	1	0	0	1	1	1	0	0

If S_1, S_2, S_3, S_4 are two-input R–S–T flip-flops, their input equations are

$$
\begin{aligned}
s_1 &= (S_2S_3\bar{C} + SC)t_1 + S\bar{t}_1 \\
\bar{s}_1 &= (\bar{S}\bar{C} + \bar{S}_2\bar{S}_3C)t_1 + \bar{S}\bar{t}_1 \\
s_2 &= [\bar{S}_1\bar{C} + (\bar{S}_1S_3 + S_1\bar{S}_3)C]t_1 + S_1\bar{t}_1 \\
&= (\bar{S}_1\bar{C} + \bar{S}_1S_3 + S_1\bar{S}_3C)t_1 + S_1\bar{t}_1 \\
\bar{s}_2 &= [(S\bar{S}_1S_3 + S_1\bar{S}_3)\bar{C} + S_1C]t_1 + \bar{S}_1\bar{t}_1 \\
s_3 &= [(\bar{S} + S_2)\bar{C} + \bar{S}_2C]t_1 + S_2\bar{t}_1 \\
\bar{s}_3 &= [(\bar{S} + S_2)\bar{C} + \bar{S}_2C]t_1 + \bar{S}_2\bar{t}_1 \\
s_4 &= (\bar{S}_3\bar{C} + \bar{S}_3C)t_1 + S_3\bar{t}_1 = \bar{S}_3t_1 + S_3\bar{t}_1 \\
\bar{s}_4 &= (S_3\bar{C} + S_3C)t_1 + \bar{S}_3\bar{t}_1 = S_3t_1 + \bar{S}_3\bar{t}_1.
\end{aligned}
\tag{6-8}
$$

In these equations the terms associated with t_1 indicate the input signals required for the correction operation, and the other terms cause the flip-flops to act as a shift register at all other times, i.e., when $\bar{t}_1$ is true.

6.1.3.2. Parallel Decimal Adders

Parallel decimal adders are more complex. The carry propagation time is the important factor as in other parallel adders. A greater variety of decimal parallel adders is possible than in the case of binary adders since combination series–parallel adders may be designed. For example, each digit may be represented bv four bits in parallel while successive digits are operated upon serially, or all the digits may be operated upon in parallel while the bits comprising the binary representation of each digit are operated upon serially.

As an example of a decimal parallel adder, let us consider a relatively simple decimal adder employing the excess-three code and operating in a manner similar to the binary parallel adder described in Section 6.1.2.6. As stated there, an advantage of having a carry flip-flop after every fourth bit in the binary adder is that decimal operation can then be obtained with only a small amount of additional equipment.

To yield correct excess-three code decimal representation, the binary sums formed at time t_1 must be modified, +3 being added to each stage if it generates a carry and −3 if it does not. This can be done by appending the signals, b_d, b_{d+1}, b_{d+2}, b_{d+3} in Eqs. (6-9) to the expressions for b_i, b_{i+1}, b_{i+2}, b_{i+3}, respectively, in Eqs. (6-7).

$$\begin{aligned} b_d &= t_2 \\ b_{d+1} &= B_d\bar{C}_{d+3}t_2 + \bar{B}_dC_{d+3}t_2 \\ b_{d+2} &= (\bar{B}_{d+1} + \bar{B}_d)C_{d+3}t_2 + (B_{d+1} + B_d)\bar{C}_{d+3}t_2 \\ b_{d+3} &= \bar{B}_{d+2}\bar{B}_{d+1}\bar{C}_{d+3}t_2 + \bar{B}_{d+2}\bar{B}_d\bar{C}_{d+3}t_2 \\ &\quad + B_{d+2}B_{d+1}C_{d+3}t_2 + B_{d+2}B_dC_{d+3}t_2. \end{aligned} \tag{6-9}$$

Eqs. (6-7) must be modified further: $t_{2/10}$ replaced throughout by $t_{3/11}$, the term $B_{i+3}\,B_{i+2}\,C_{i-1}\,t_{3/11}$, indicating a stage has the value 9 and there is a carry from the preceding stage, substituted for $B_{i+3}\,B_{i+2}\,B_{i+1}\,C_{i-1}\,t_{2/10}$ in c_{i+3}, and appended to the logic of b_{i+1}, b_{i+2} and b_{i+3}. The carry flip-flops are reset by t_3, t_4, t_5, . . . t_{11}, respectively, and in the carry completion sensing logic, $t_{2/11}$ is replaced by $t_{3/12}$.

6.1.3.3. Parallel Accumulators with Automatic Carry Propagation

If an accumulator's design is such that it utilizes a step-by-step

carry process in arriving at a result, its operating speed would not be adequate for a parallel machine. The speed may be increased by arrangements wherein a carry produced in any stage is automatically propagated to higher order stages. There are two widely used methods in mechanizing such operation. In one, the operation is under the control of two input commands, an "add" and a "carry" command which are applied in sequence to all stages simultaneously. In the other, the operation is initiated and completed simply by the application of an "add" command to all stages simultaneously. In the descriptions that follow, A_i refers to the bistable elements of an addend register, and B_i refers to the bistable counter circuits of the accumulator.

The circuit shown in Fig. 6.22(a) functions as follows. Operation is

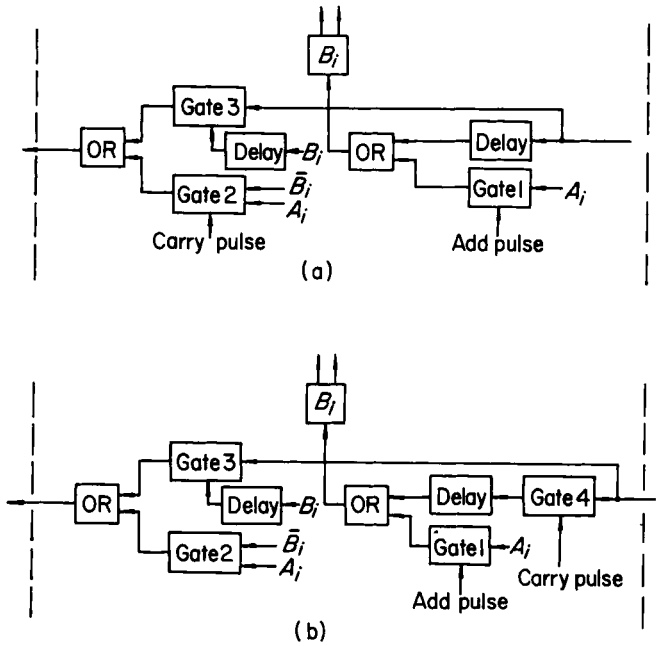


FIG. 6.22. Typical stage of a parallel accumulator. (Either one but not both delay units may be required in these two arrangements.)

begun by application of the "add" pulse. In each stage in which A_i is 1, the corresponding B_i is triggered. If B_i is 0 after being triggered, and A_i is 1, this indicates that a carry should be propagated to the next more significant stage. Therefore, when the carry command pulse is applied, gate 2 allows it to pass through a stage where a condition for a carry

propagation has been met. If the next more significant stage of the accumulator holds a 1, the carry is allowed to pass through via gate 3. It passes through successive gates until a stage is reached where the accumulator holds a 0. The delay element is included only if it is essential to insure that a carry does not pass through a stage where it should not. This could occur if a carry from a preceding stage arrived at the next more significant stage of the accumulator before it changed from 1 to 0.

The circuit shown in Fig. 6.22(b) produces a faster propagation of the carry. It differs from the arrangement of Fig. 6.22(a) mainly in that there is no command pulse input to gate 2. As a result, the condition $\bar{B}_i A_i$ produces a steady state signal that is applied to gates 3 and 4 of the next more significant stage before application of the "carry" command. If the next more significant stage of the accumulator holds a 1, the steady state signal is passed through to the stages beyond via gate 3. After time has been allowed for the steady state signal to pass through the maximum possible number of stages, the "carry" command is applied to gate 4. This causes B_i to be triggered in each stage to which a signal was transmitted from a preceding stage.

6.1.3.3.1. ACCUMULATORS WITH SEPARATE CARRY STORAGE. If it is difficult to obtain signals from the addend register for carry purposes after the addend has been entered into the accumulator, a carry storage device may be incorporated in each stage. The carry flip-flop C_i is set to 1 whenever B_i changes from 1 to 0.

One arrangement utilizing a carry flip-flop is almost identical to that shown in Fig. 6.22(a). The only change, outside of the addition of C_i, is the replacement of the signal $\bar{B}_i A_i$ at the input to gate 2 by the signal C_i. Another arrangement is similar to that of Fig. 6.22(b). The only change, outside of the addition of C_i, is the elimination of gate 2 and the substitution of the signal C_i as the lower input to the OR gate.

In Fig. 6.23 two types of accumulators are depicted. Circuitry common to both is in the center of the figure while that peculiar to circuit (a) and (b) is shown in dashed enclosures. In circuit (a) the addend is entered upon application of the "add" command. Upon application of the "carry" command pulse, B_i is triggered if the next less significant stage produced a carry as indicated by the signal C_{i-1}. If B_i held a 1 just prior to receiving a carry, it changes to 0 and causes the signal C_i to be produced. The signal C_i is sent to the next more significant stage to which the "carry" command has already been applied, and a pulse is passed via gate 4 to trigger B_{i+1}. The carries will be propagated as far as necessary, provided the carry command signal is applied for a sufficient period. This method, though generally not as rapid as the preceding two,

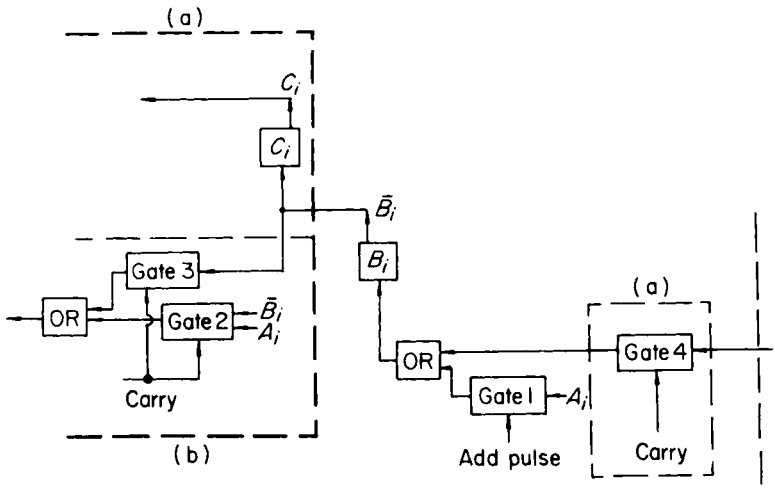


FIG. 6.23. Typical stage of a parallel accumulator

has the advantage of requiring fewer components and no delay elements because of the manner in which the carries are propagated. The principles of the arrangement just described can be adapted to the carry propagation method of Fig. 6.22(a) to gain an important advantage. The resulting arrangement is circuit (b) in Fig. 6.23. The addend is entered as usual. When the "carry" command signal is activated, a carry signal is transmitted to the next more significant stage via gate 2, if the condition $A_i\bar{B}_i$ exists. A pulse will also be sent to the next more significant stage if a carry pulse from the next less significant stage arrives. This is because a carry from the next less significant stage would trigger B_i from 1 to 0, thereby causing a pulse to be transmitted to gate 3. The advantage of this accumulator is that no pulse or steady-state signal has to pass through more than one stage of gates.

6.1.3.3.2. ACCUMULATORS WITH NO "CARRY" COMMAND INPUTS. Figure 6.24 illustrates a simple form of accumulator requiring no carry command. When any B_i changes from 1 to 0, a pulse is sent to the next more significant stage via a delay. Operation is slow because in the carry propagation circuit, all the delays are in series. The arrangement of Fig. 6.25 is faster because a delay D_1 is introduced only in the stage where a carry originates. Operation is as follows: The addend is entered in the usual manner. If B_i is caused to change from 1 to 0, a pulse is applied to gate 3 which allows the "add" pulse to pass to the next more significant stage, via delay element D_1, (which allows B_{i+1} to recover in the event it was

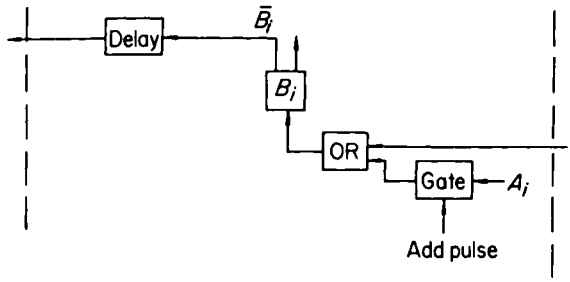


FIG. 6.24. Typical stage of a parallel accumulator with no carry command input

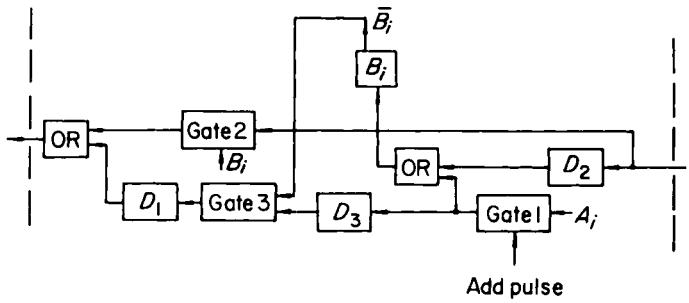


FIG. 6.25. Typical stage of a parallel accumulator with no carry command input

triggered upon application of the "add" command). A delay element D_3 may be necessary if the switching time of a B_i is appreciable. A carry from the next less significant stage will pass without delay via gate 2 through any stage where a B_i has been triggered to 1. If a B_i is triggered from 1 to 0 upon receipt of a carry from the next less significant stage, another pulse will not pass via gate 3 because the "add" pulse will no longer be present. Delay D_2 is a short delay introduced to insure that the carry pulse does not switch B_i before the carry itself can pass through gate 2. If the inherent switching delay of B_i were large enough, D_2 would not be required.

The arrangement in Fig. 6.26 allows the carry signals to be generated before the addend is entered into the accumulator. In each stage the signal for a carry, $(A_iB_i + A_iC_{i-1} + B_iC_{i-1})$, is produced by a set of AND and OR gates, which form the equivalent expression $[(A_i + B_i)C_{i-1} + A_iB_i]$. In the other accumulators that have been described, each B_i was triggered twice if A_i were 1 and there was also a carry from the next less

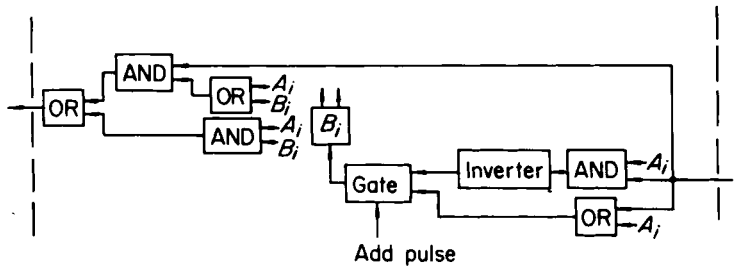


FIG. 6.26. Typical stage of a parallel accumulator with no carry command input

significant stage. This produced no net effect on B_i. The arrangement of Fig. 6.26 causes B_i to be triggered only if either of these events, but not both, occurs, i.e., by the signal $A_i\bar{C}_{i-1} + \bar{A}_iC_{i-1}$. In Fig. 6.26, an equivalent expression, $(A_i + C_{i-1})\overline{A_iC_{i-1}}$, is formed. If this type of accumulator is modified to function as a subtractor, a carry signal from the highest stage can be used to indicate whether the sign of the difference will be positive or negative. In a trial-and-error division process (see Section 6.1.6.1.1), the subtraction can be prevented from taking place if an indication is provided of a negative difference.

The accumulators with automatic carry propagation which have been described in this section do not have means for indicating the end of carry propagation. A multi-input AND gate for sensing a carry in any stage could be used with some of them, but is more suited for step-by-step carry systems. The arrangement of Fig. 6.21(b) has a fast carry propagation circuit which can readily be provided with means to yield a signal when the carry process is completed. The sum is entered into the accumulator by application of an add pulse to the input gates, as shown in Fig. 6.27. This pulse causes each B_i to be triggered if $(AC^0_{in} + \bar{A}C^1_{in})$ is 1. Note that this corresponds to a conventional half-adder sum signal: $(A\bar{C} + \bar{A}C)$.

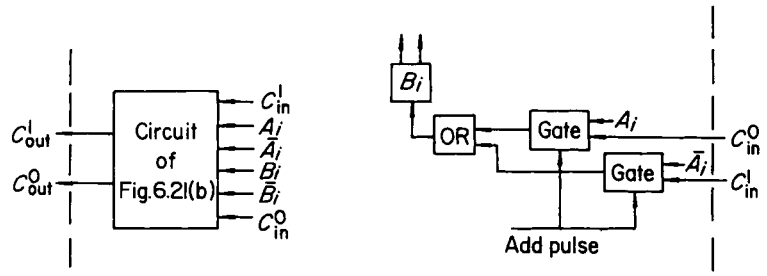


FIG. 6.27. Typical stage of an accumulator with a 0 and 1 carry propagation circuit

6.1.4. The Representation of Negative Numbers and the Subtraction Process

In a digital computer provision is made for representation of negative as well as positive numbers by using one of the following two schemes. One of these schemes is commonly encountered in everyday usage of numbers. It uses a common grouping of symbols to represent a given magnitude, and a special symbol to indicate whether the value is positive or negative. In other words, each number is represented in terms of an absolute value plus a sign, e.g., + 703, − 703. The other method of representation relies on the use of a so-called complementary number system. The nature of complementary numbers, as well as a description of the merits of absolute and complementary representation in a digital computer will be described in the sections following.

6.1.4.1. Representation of a Negative Number by an Absolute Value Plus Sign

In this representation, a negative number is distinguished from a positive one by an arbitrary symbol (usually, but not necessarily) preceding the number. For example, either of the following absolute value plus sign designations could be employed

Designation 1	Designation 2
$0.11 = +3/4$	$1.11 = +3/4$
$1.11 = -3/4$	$0.11 = -3/4$

Because of its wide everyday use, this type of representation in a computer simplifies the preparation of input data, as well as the visual interpretation of output data and data stored in the computer. Also, it simplifies multiplication and division in a machine. To obtain the correct sign for a product or quotient, only a simple circuit is required to compare the signs of the operands, yielding a positive value for the sign if they are alike, and a negative value if different, (see the discussion of comparators in Section 6.1.4.3). For addition, operations are more complex than with complementary representation. If the signs are alike, addition is performed normally, but if they are different, subtraction must be performed, with means to insure that the smaller absolute value is subtracted from the larger, and that the correct sign is attached to the result. Negative quantities must be in complemented form before being sent to an adder. If the result of an operation is negative, and, therefore, in complement form it must be uncomplemented before it is stored, and the correct sign bit attached. However, complements may be avoided completely by using a subtractor when the signs of the operands are not alike.

Just as there are half-adders, there are also half-subtractors. The input-output relations for a half-subtractor are shown in Table 6.12. The minuend, subtrahend, difference, and borrow bits are represented by A_i, B_i, D_i, and C_i, respectively.

TABLE 6.12. Truth table for a half-subtractor

A_i	B_i	D_i	C_i
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The input–output relations for a full subtractor are shown in Table 6.13.

TABLE 6.13. Truth table for a full subtractor

A_i	B_i	C_{i-1}	D_i	C_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

In Table 6.13, C_{i-1} represents the borrow produced in the less significant binary place and C_i is the borrow resulting from the combination of C_{i-1}, the minuend A_i and the subtrahend B_i. The equations for the difference and borrow bits are

$$\begin{aligned} D_i &= A_i\bar{B}_i\bar{C}_{i-1} + \bar{A}_iB_i\bar{C}_{i-1} + \bar{A}_i\bar{B}_iC_{i-1} + A_iB_iC_{i-1} \\ C_i &= \bar{A}_iB_i\bar{C}_{i-1} + \bar{A}_i\bar{B}_iC_{i-1} + \bar{A}_iB_iC_{i-1} + A_iB_iC_{i-1} \\ &= \bar{A}_iB_i + \bar{A}_iC_{i-1} + B_iC_{i-1}. \end{aligned} \tag{6-10}$$

These equations are similar to Equations (6-1) and (6-2) and show that the sum for an adder and the difference for a subtractor are of the same form. In reference to the production of a negative result by a

subtractor, it should be remarked that a negative result appears in complementary form. However, it is possible to use the subtractor itself as a complementer, by subtracting the number to be complemented from zero. In passing, note that a subtractor may be considered more fundamental than an adder, since an addition may be performed by two subtractions, e.g., $(A + B) = A - (0 - B)$.

6.1.4.2. Representation of a Negative Number by a Complement

There are two important complementary representations of a negative number. One is referred to as the radix complement and the other as the (radix − 1) complement. In the decimal system these are known as the true (or ten's) complement and the nine's complement, respectively. In the binary system, they are the true (or two's) complement, and the one's complement, respectively. Complements in the binary system only will be discussed here. Similar statements apply to the decimal system.

The use of complements to represent negative numbers makes it unnecessary to build into a computing machine the ability to both add and subtract. Actually, a subtraction takes place when a complement is formed, but the advantage derives from the fact that the forming of a complement is easier to mechanize than a subtraction.

6.1.4.2.1. Two's Complement Representation. Complementary representation of negative numbers is similar to the notation used for handling logarithms of numbers less than 1. The two's complement of a number is obtained by subtracting the number from 2. An easy way to mechanize this operation is to interchange all 1's and 0's and add 1 to the least significant binary place. For example

7/16 = 0.0111	
Interchanging 1's and 0's	1.1000
Adding 1 in the least significant place	1
Result	1.1001 = −7/16

Any two binary numbers may be added (replacing negative numbers by their two's complement) if the following two rules, analogous to those in the decimal system are used. First, discard any carry into the two's (2^1) column, since $M_1 - M_2 = M_1 + (2 - M_2) - 2$. Secondly, if there is a 1 in the one's (2^0) column, the sum is interpreted as a negative number and its magnitude is obtained by taking its two's complement. Because of the latter rule, the digit in the one's (i.e., 2^0) column is referred to as the sign digit. Example 6.5 illustrates the use of these rules

Example 6.5

	Binary number	Decimal equivalent
	0.0100	4/16
	0.1001	9/16
	0.1101	13/16
	0.0100	4/16
	1.0111	− 9/16
	1.1011	− 5/16
	1.1100	− 4/16
	1.0111	− 9/16
	1 | 1.0011	− 13/16
Carry into 2^1 column is discarded	|	
	| 1.1100	− 4/16
	| 0.1001	9/16
	1 | 0.0101	5/16

When the bit to the left of the binary point is used to indicate the sign of a number x the range of numbers within the machine is restricted to the interval $-2^\circ \leqslant x < 2^\circ$. Machines of this type are accordingly referred to as fractional machines.

6.1.4.2.2. ONE'S COMPLEMENT REPRESENTATION. The one's complement of a number is formed by interchanging all ones and zeros. This is easily mechanized since each bit can be altered without reference to other bits. Addition or subtraction can be executed as above, with the following exception: whenever an overflow bit is produced at the most significant end, this bit must be added in at the least significant end. Consider the addition of two negative numbers, $-4/16$ and $-9/16$, as in Example 6.6.

Example 6.6

	Decimal representation	One's complement representation
Addend	− 4/16	| 1.1011
Addend	− 9/16	| 1.0110
		|
Result of addition	− 13/16	1 | 1.0001
End around carry		↳ 1
Sum	− 13/16	| 1.0010

In case of an overflow in subtraction, an end-around borrow would be subtracted from the difference. When a positive sum is obtained in adding a positive number to a negative one, a carry is generated in the highest order. This carry indicates a change in sign from the previous balance and must be added to the lowest order to restore the true indication of the sum, in one's complement form.

The advantage of the one's complement is that conversion to the true number is so simple. This is important if the results of a computation are to be displayed, and to facilitate certain schemes of multiplication and division. The one's complement is used in several parallel binary machines. An objection to it is that representation of zero is not unique. Two representations of zero may occur, namely, the normally encountered one, 0.000 . . . 0 (sometimes referred to as "plus zero") and 1.111 . . . 1 (referred to as "minus zero"). Whenever subtraction of two like quantities occurs in a subtractive accumulator, "plus zero" is produced and when a number and its 1's complement are added in an additive accumulator, a "minus zero" is produced. It is possible for either form to be produced in any system. For example, the rounded-off product of two very small quantities can result in either a "plus zero" or "minus zero" depending on whether the signs of the factors are alike or opposite, respectively. In any case, the machine design should be such that the programmer need not distinguish between the two forms of zero in the use of conditional test instructions.

Factors that influence the final choice in the representation of negative numbers include: (1) The machine's facilities for complementing. (2) The expected relative frequencies of the different arithmetic operations. (3) The relative convenience and need of examining numbers in storage for servicing the computer. (4) The form of numbers to be transmitted to the computer from external inputs, and the form in which output quantities must appear.

6.1.4.2.3. SUBTRACTION OF BINARY CODED DECIMALS. Subtraction of one binary coded decimal, d_1, from another, d_2, may be accomplished by adding the ten's complement of d_1, *i.e.*, $(10^n - d_1)$ to d_2. Whenever $d_1 \leqslant d_2$, indicating a positive or zero remainder, a carry into the 10^n column is produced. This carry is discarded so that the net effect of the operation is as follows

$$d_2 + (10^n - d_1) - 10^n = d_2 - d_1.$$

If $d_1 > d_2$, indicating a negative remainder, no carry into the 10^n column is produced. Therefore, the result of the operation is

$$d_2 + (10^n - d_1) = 10^n - (d_1 - d_2).$$

The absence of a carry into the 10^n column can be used to indicate that the difference is negative and in complementary form. If the minuend is negative, the addition of the complement of the subtrahend would produce $(10^n - d_1) + (10^n - d_2)$. If the carry into the 10^n column is discarded, the difference is $10^n - (d_1 + d_2)$ indicating a negative result.

An advantage of the nine's complement over the ten's complement is that conversion between it and true representation is more straightforward. The use of the nine's complement requires an end-around carry, but this is a negligible complication in parallel systems and not always a serious one in serial systems. If the nine's complement of d_1 is added to d_2, the result is $d_2 + (10^n - 1) - d_1$. Whenever $d_1 < d_2$, indicating a positive or zero remainder, a carry is produced which is then added in end-around fashion to the least significant digit. The difference then is equal to $[d_2 + (10^n - 1) - d_1] - 10^n + 1 = d_2 - d_1$. If $d_1 \geqslant d_2$, no carry is produced so the result of the operation is $(10^n - 1) - (d_1 - d_2)$. As in the ten's complement system, the absence of a carry into the 10^n column is used to indicate that the difference is negative and in complementary form.

In the nine's complement system, addition of a number and its complement produces the minus representation of zero, as in any (radix − 1) system. To obtain the difference of two like numbers directly as a plus zero, a subtractor may be used. To add, the nine's complement of a number is subtracted, and to subtract, the number itself is subtracted. When a subtractor is used, end-around borrows rather than carries must be considered. Example 6.7 illustrates the addition of 23 to 0 in a subtractor.

Example 6.7

Decimal	Binary-coded decimal		
00	0000	0000	
−23	0111	0110	Nine's complement of 23
—	—	—	
	−1000	1010	
	⟵ 1		End-around borrow
	—	—	
	1000	1001	
	0110	0110	Correction
	—	—	
23	0010	0011	

The result 0010 0011 is the correct difference: $00 - (-23) = 23$.

The correction term is required because, when using a subtractor, the

difference must be corrected by subtracting six from each decimal code group where a borrow has been produced out of the column, as shown in Example 6.8, also.

Example 6.8

Decimal		Binary-coded decimal	
2		0010	
5		0101	
—		—	
	1	1101	
		0110	Correction
		—	
Difference (−) 3	(1)	0111	

The result, (1) 0111, represents the correct difference, −3, in ten's complement form.

Whenever a straight binary-coded decimal number is subtracted from another each four-bit decimal group that produces a borrow must be corrected by subtracting six from it, and the difference then would be in the form of a (10^n)'s complement, which is referred to simply as a ten's complement. In Example 6.9, this process is illustrated for the case of two three-digit binary-coded decimals. (The difference could be obtained in nine's complement form by use of an end-around borrow).

Example 6.9

Decimal		Binary-coded decimal			
257		0010	0101	0111	
− 465		0100	0110	0101	
—		—	—	—	
		1101	1111	0010	
		0110	0110		Correction
		—	—	—	
— 208	(−)	0111	1001	0010	

The result, (−) 792, represents the ten's complement of the correct difference, 208.

Of the 70 weighted four-bit codes, referred to in Section 6.1.2.1., 18 including the commonly used (2, 4, 2, 1) code are self-complementing, i.e., the nine's complement can be obtained simply by interchanging 1's and 0's. A disadvantage of the (8, 4, 2, 1) code is that it is not self-complementing: interchanging 1's and 0's yields the 15's complement. To obtain the nine's complement, either of the following methods can be used: adding 6 to the (8, 4, 2, 1) representation and then inverting,

or inverting first and then adding 10 to the result. In the latter case, the carry out of the eight's column is discarded, and effectively subtracts 16 from the result. That both of these procedures yield the nine's complement of the digit d may be seen as follows

$$15 - (d + 6) = (15 - d) + 10 - 16 = 9 - d.$$

The nine's complement may also be generated by means of a simple logical network that produces the bits c_i of the complement digit in accordance with the following equations

$$c_1 = \bar{d}_1$$

$$c_2 = d_2$$

$$c_4 = d_2\bar{d}_4 + \bar{d}_2 d_4$$

$$c_8 = \overline{(d_2 + d_4 + d_8)}$$

These relations may be obtained by considering the truth table for the nine's complement $(c_8c_4c_2c_1)$ as a function of the table defining the values of the binary-coded decimals $(d_8d_4d_2d_1)$. This scheme of conversion implies that the bits of the binary-coded decimal must be available in a parallel representation, for otherwise the functions c_4 and c_8 could not be obtained.

6.1.4.3. Comparators

A commonly encountered requirement in general purpose computers and data processing systems is a test for the relative magnitudes of two numbers, a and b. Three cases are possible: $a = b$, $a < b$, or $a > b$. If the absolute values of both numbers are presented serially, least significant bit first, the three cases can be distinguished by means of two flip-flops as follows. Assume there are two R–S type flip-flops, F_1 and F_2, both of which are reset to 0 prior to the comparison. The successive bits of each number are referred to as A and B. Whenever $\bar{A}B$ occurs, F_1 is set to 1. Whenever $A\bar{B}$ occurs, F_1 is reset to 0 and F_2 is set to 1. If each bit of a is equal to each bit of b, i.e., only the cases AB or $\bar{A}\bar{B}$ occur, neither F_1 nor F_2 is ever set to 1. To summarize, the input equations to F_1 and F_2 are

$$f_1 = \bar{A}B \qquad f_2 = A\bar{B}$$

$$f_1' = R + A\bar{B} \qquad f_2' = R.$$

At the end of the comparison process, one of the following three conditions will exist

$$\text{If } a = b, \qquad \bar{F}_1\bar{F}_2 = 1$$

$$\text{If } a > b, \qquad \bar{F}_1F_2 = 1$$

$$\text{If } a < b, \qquad F_1 = 1.$$

If it is only desired to distinguish between one of these cases and the other two, only one flip-flop is required. For example, whether $A > B$, or $A \leqslant B$ can be determined by a single flip-flop, F, with the following input equations

$$f = A\bar{B} \qquad f' = \bar{A}B + R.$$

If F is 1 at the end of a word, it implies that $A > B$, and if $\bar{F} = 1$, $A \leqslant B$. Another flip-flop, G, can be used to distinguish $A < B$ from $A = B$ by testing for $A = B$

$$g = A\bar{B} + \bar{A}B \qquad \bar{g} = R.$$

If $G = 1$ at the end of the comparison, it implies that $A \neq B$.

With dynamic flip-flops, a test for $A \geqslant B$ or $A < B$ may be performed as follows. A dynamic set–reset flip-flop F is initially set to 1 by a pulse S. The flip-flop stays in this condition unless $\bar{A}B$ occurs. Once put in the 0 condition it stays there unless $A\bar{B}$ occurs. Since, if the flip-flop is on, agreements, i.e., $AB + \bar{A}\bar{B}$, keep it on, it cannot distinguish between $A > B$ or $A = B$. At the end of the comparison process, F indicates $A \geqslant B$, and $\bar{F}$ indicates $A < B$. The input–output relation is

$$F_{i+1} = S + A\bar{B} + (AB + \bar{A}\bar{B})F_i.$$

A similar circuit could be used to indicate whether $A \leqslant B$, or $A > B$. A test for equality may be performed as follows: A flip-flop, G, is initially set to 1 by a pulse, S. The input $G(AB + \bar{A}\bar{B})$ causes the 1 state to be maintained as long as corresponding bits in A and B are equal. The first disagreement will interrupt regeneration of the 1 state which cannot then be attained regardless of agreement of subsequent bits. At the end of the comparison process, $G = 1$ indicates $a = b$, and $\bar{G} = 1$ indicates $a \neq b$. The input–output relation is

$$G_{i+1} = S + (AB + \bar{A}\bar{B})G_i.$$

6.1.5. MULTIPLICATION

In number systems other than the binary, multiplication is normally performed by: (1) Inspecting each digit of the multiplier in sequence and adding the multiplicand into the partial product a number of times corresponding to the multiplier digit. (2) Shifting the partial product by one digit place upon the completion of operation (1). Multiplication in

the binary system is simplified because, since a multiplier digit can only have the values 0 or 1, the number of additions between shifts will not vary. The binary multiplication table is shown in Table 6.14.

TABLE 6.14. Binary multiplication table

	0	1
0	0	0
1	0	1

There are a number of factors which must be taken into consideration when providing for multiplication by a single programmed instruction, e.g., the number of components that will be required, the maximum and average specified execution times (in an asynchronous system) or the standard execution time (in a synchronous system), round off procedures, the way in which negative numbers are to be treated, etc. Some of these items are considered briefly in the paragraphs following.

Multiplication is an operation consisting essentially of many additions. These may be performed slowly by successive additions in a single adder circuit, quickly by simultaneous additions in a large number of adders, or at an intermediate speed using an intermediate number of adders. The size of a multiplier increases with its speed and usually a compromise must be made in its design between speed and size, taking into consideration the size of the remainder of the computer.

As a general rule, the speed specified for a multiplication should be arrived at after considering the speed of execution of elementary operations (e.g., add, subtract, information transfers) and the estimated relative frequencies of multiplications and elementary operations in the class of problems to be solved by the computer. Since in general there will be considerably more additions than multiplications in programs chosen randomly, it is usually not economical to reduce the time required for a multiplication to below, say, k times the time required for an addition (where k is the ratio of additions or subtractions to multiplications). This is not a hard and fast rule, but calls attention to the fact that when considering increased complexity in equipment, the point of diminishing returns should not be overlooked, considering both the economics of the design and reliability of performance.

The product of two n-bit numbers may have as many as $2n$ significant bits. Therefore, if full accuracy is required, provision must be made not only to form the full $2n$ bit product in the multiplier, but also means must be provided so that it may be transferred to and stored in the memory. In those cases where it is neither convenient nor necessary for

the full $2n$ bit product to be retained, means must be provided for rounding off the product. Since the operands entering into a multiplication are obtained from n-bit capacity storage cells, and since the products formed must also be returned to these cells, it is convenient to round products to n binary places. It is desirable that any round-off procedure adopted produce a mean error of zero, and also a relatively small mean deviation. In other words, the errors introduced by the rounding process should cancel out over a large number of round-offs, and the maximum error introduced by one round-off operation should be relatively small. The production of a mean error of zero in round-off procedures depends upon the assumption that the remainders (i.e., the n least significant bits of a $2n$ bit product) can be considered to be random numbers. There are programs in which this assumption is not valid. The subject of round-off procedures is considered in more detail in Chapter 9.

If each number is represented in terms of an absolute value plus a sign, no difficulty is introduced when one or both of the factors is negative. The sign bits are merely ignored during the multiplication process, and the correct sign bit appended to the product in accordance with a simple comparison procedure which indicates whether the factors are of like or opposite sign. If negative numbers are stored in a one's complement form, they may readily be converted to a signed form before entering the multiplier. However, the signs of the operands must be known prior to multiplication, and in a serial computer this may cause a difficulty since least significant bits usually appear first. A negative product would be converted to its one's complement form before being transferred to other parts of the machine. If negative numbers are in a two's complement form, the complexity of conversion to a signed form usually leads to consideration of special multiplication methods that operate directly on numbers in this form. Three such methods are described in Section 6.1.5.1.6.

Another consideration that influences the design of a multiplier is whether there is a requirement for either or both of the operands to be available in the multiplier, after generation of the product, for use by the programmer.

6.1.5.1. Binary Multiplication

A commonly used method of performing multiplication consists of repeated addition of the multiplicand into appropriate orders of an accumulator. This simple process is applicable whether the operands are presented in serial or parallel. A description of multiplication by both serial and parallel accumulation is presented in the sections following.

6.1.5.1.1. MULTIPLICATION BY PARALLEL ACCUMULATION. Wherever a multiplier bit is 1, a partial product must be added, appropriately shifted, to the accumulated sum of the preceding partial products. For example, consider a multiplicand represented by $b_4\ b_3\ b_2 b_1$ and a multiplier $a_4\ a_3\ a_2\ a_1 = 1111$. The first partial product is $b_4\ b_3\ b_2\ b_1$ and the second, third, and fourth partial products are obtained by the process indicated in Example 6.10.

Example 6.10

$$
\begin{array}{l}
\quad b_4\ b_3\ b_2\ b_1 \\
\quad 1\ \ 1\ \ 1\ \ 1 \\
\hline
\quad b_4\ b_3\ b_2\ b_1 \\
\quad\ b_4\ b_3\ b_2\ b_1 \\
\hline
p_{26}p_{25}p_{24}p_{23}p_{22}p_{21} \\
\quad b_4\ b_3\ b_2\ b_1 \\
\hline
p_{37}p_{36}p_{35}p_{34}p_{33}p_{32}p_{31} \\
\quad b_4\ b_3\ b_2\ b_1 \\
\hline
p_{48}p_{47}p_{46}p_{45}p_{44}p_{43}p_{42}p_{41}
\end{array}
$$

There are, basically, two ways of controlling the addition of the partial products to the contents of the accumulator so that they are entered in the proper orders. One is to cause the partial products to be shifted before entry into the accumulator. The other is to always enter the partial products into the same order of the accumulator, and to shift the accumulated sum before entry of the next partial product. Although shifting is a relative term, as a matter of convenience we shall refer to the first scheme as shifting of the partial products, and the second as shifting of the accumulated sum. Each scheme will now be described in more detail.

In the scheme for shifting of the partial products, as many distinct timing signals are provided as there are orders in the multiplier, a one-to-one correspondence being established between a timing signal and an order of the multiplier. Each partial product is channelled to the appropriate orders of the accumulator by means of a group of gates controlled by these timing signals. The nature of these gates is indicated, for the case of a four-bit multiplier, by Eq. (6-11).

$$
\begin{aligned}
r_7 &= S_3B_4 \\
r_6 &= S_3B_3 + S_2B_4 \\
r_5 &= S_3B_2 + S_2B_3 + S_1B_4 \\
r_4 &= S_3B_1 + S_2B_2 + S_1B_3 + S_0B_4 \\
r_3 &= S_2B_1 + S_1B_2 + S_0B_3 \\
r_2 &= S_1B_1 + S_0B_2 \\
r_1 &= S_0B_1
\end{aligned}
\tag{6-11}
$$

In Eq. (6-11) r_i represents the 1 input to the ith stage of the accumulator, S_i a line on which a timing pulse appears signalling that the ith bit of the multiplier is to be inspected and the ith partial product added, and B_i represents the ith bit of the multiplicand. The number of terms in Eq. (6-11), and hence the amount of circuitry required, is a function of the number of bits in the operands. The number of AND gates equals the product of the number of bits in each operand, and the number of OR gate inputs is two less than this number.

An alternate way of shifting the partial products is indicated by Eq. (6-12)

$$
\begin{aligned}
r_7 &= S_2S_1B_4 \\
r_6 &= S_2(\bar{S}_1B_4 + S_1B_3) \\
r_5 &= \bar{S}_2S_1B_4 + S_2(\bar{S}_1B_3 + S_1B_2) \\
r_4 &= \bar{S}_2(\bar{S}_1B_4 + S_1B_3) + S_2(\bar{S}_1B_2 + S_1B_1) \\
r_3 &= \bar{S}_2(\bar{S}_1B_3 + S_1B_2) + S_2\bar{S}_1B_1 \\
r_2 &= \bar{S}_2(\bar{S}_1B_2 + S_1B_1) \\
r_1 &= \bar{S}_2\bar{S}_1B_1.
\end{aligned}
\tag{6-12}
$$

Here there is no individual timing control line for each shift command. Instead, an indication of the proper shift is provided by a binary-coded signal appearing simultaneously on a number of control lines. For example, the two control lines S_1, S_2 can provide the signals $\bar{S}_2\bar{S}_1$, $\bar{S}_2S_1$, $S_2\bar{S}_1$, and S_2S_1, indicating shifts of 0, 1, 2, and 3 binary orders, respectively. For practical lengths of the operands, this arrangement requires fewer components than the preceding one. However, if the equations are mechanized in the form shown, multi-level gates are required and if they are expanded more gating elements are required.

In the scheme for shifting the accumulated sum, assume that there is available an accumulator each stage of which can, upon command,

shift its contents one bit to the right. Assume also that the bits of the multiplicand $B_n \ldots B_2B_1$ are always entered into the same orders of the accumulator, as shown in Fig. 6.28. The multiplication process is then as

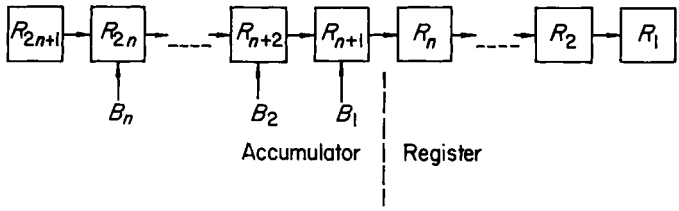


FIG. 6.28. A shifting accumulator

follows: Each multiplier bit is examined in succession, and the multiplicand is added into the orders shown whenever the multiplier bit is equal to 1. After each sum is produced, the contents of the accumulator are shifted right. Note that after entry of the first partial product into the accumulator, the least significant bit of the product is determined, and that, in general, after entry of the ith partial product the ith order bit in the product is determined. Therefore, the lowest n orders of the accumulator are not required for summation, but need only provide the functions of storage and shifting. Stage $2n + 1$ is needed to temporarily store carries from stage $2n$. The advantages of shifting in the accumulator are now apparent. First, far less combinational circuitry is required, e.g., in Fig. 6.28 the 1 input to an R-S flip-flop R_i is $R_{i+1}S$, where S is the shift command. Secondly, less equipment is required for addition, the number of accumulating orders being reduced from $2n$ to $n + 1$. The arrangement shown in Fig. 6.28 has another important feature in that the n least significant orders of the accumulator can be used to store the multiplier before the multiplication process begins. Each time the accumulated sum of the partial products is shifted to the right, so is the multiplier. Therefore, the addition of the partial products can be under control of the multiplier bit in the least significant stage of the accumulator. At the completion of the process the multiplier has been lost, but by that time it is no longer needed. If needed subsequently in some other operation it can be obtained from the position in storage from which it was copied into the accumulator.

6.1.5.1.2. MULTIPLICATION BY SERIAL ACCUMULATION. In a serial computer, the registers for storage of the operands and product can be either of the static or delay line type. In the latter case, shifts of information must be relative to standard timing signals, and may be produced

by the insertion of extra delays in the circulation path. A static storage register could be used in a serial mode by causing the bits to be sensed in sequence, automatically returning to the first bit in the sequence after the last bit has been sensed. In both cases, a zero reference timing signal is provided at the time the first bit of a number is to be read.

A particular scheme for serial multiplication using delay line registers is shown in Fig. 6.29. At the end of the multiplication, one n-bit delay

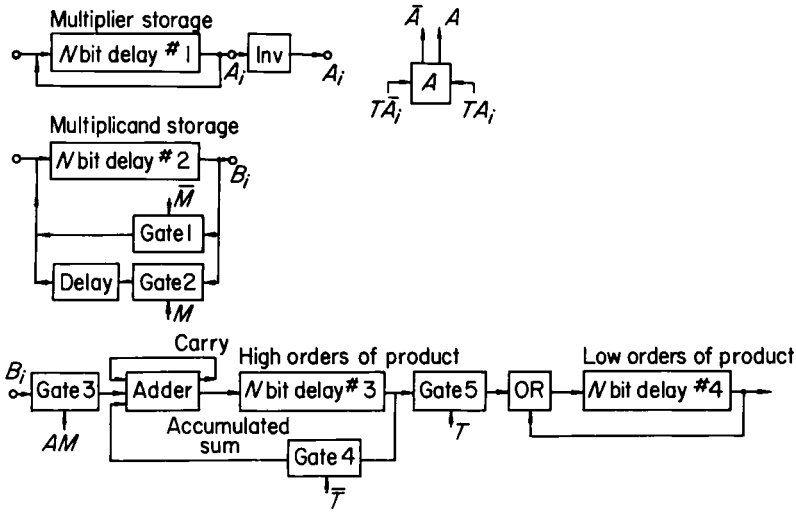


FIG. 6.29. A serial binary multiplier with delay line storage

holds the high order bits of the product and another the low order bits. The multiplication process is as follows: First it is assumed that the operands have been entered into their respective circulating registers in phase, i.e., corresponding orders of the two numbers are read or recorded at the same time. Upon application of the multiplication command signal M, a one bit delay is inserted into the recirculation loop of the multiplicand. A timing signal T, which lasts one bit period, and reappears after a period of $n + 1$ bits, is used to define the times at which successive bits of the multiplier are inspected. Whenever a multiplier bit is 1, the flip-flop A is set to state A, allowing the bits of the multiplicand to pass via gate 3 to the adder. Because of the additional one bit delay in the multiplicand recirculation loop, the bits of the multiplicand will become shifted one bit with respect to their original timed presentation each time the loop is traversed. This has the desired effect of causing a partial product to be always automatically and correctly shifted with respect to the accumulated sum. The least significant bit of the current accumulated sum always

appears at the output of delay no. 3 at time T. Future additions of partial products have no effect on this bit so it is sent to delay no. 4 via gate no. 5. Delay no. 4 stores and recirculates the $n/2$ low order bits of the product. Note that this arrangement is analagous to the multiplication procedure using a static shift register where an adder was also used only in conjunction with the upper half of the register. Since each shift, addition operation requires n pulse periods, the whole process is completed after n^2 pulse periods. At that time gate 3 is closed and the most significant half of the product can be read from delay no. 3.

Certain necessary details such as means for entering the operands, starting and stopping the multiplication process, rounding off and withdrawing the product have been omitted from the preceding descriptions for the sake of brevity, since there are a great many possible ways of achieving these functions. As far as round-off and storage of the product is concerned, the following comments are pertinent. First, each machine usually only has provisions for numbers of a fixed word length both in the main store and operand register of the arithmetic unit. Normally this is adequate, for one is usually interested in retaining only the most significant half of the product, even though the lower order bits are sometimes useful as, for example, in interpolation programs. However, there may be a requirement for temporary storage of the highest order bit of the least significant half of the product to accomplish a particular round-off procedure. Also, if a round-off scheme is used wherein an addition is made to the most significant bit to be dropped or the least significant bit to be retained, an additional time of n bit periods is required to produce the rounded product. (See Chapter 9 for a description of round-off procedures.)

6.1.5.1.3. SERIAL–PARALLEL MULTIPLIERS. In a serial machine the summing of the partial products is done successively by a single adding circuit. The time required to sum the partial products may be reduced by using several adders. The arrangement shown in Fig. 6.30 makes use of $n - 1$ adders and is referred to as a serial–parallel multiplier,

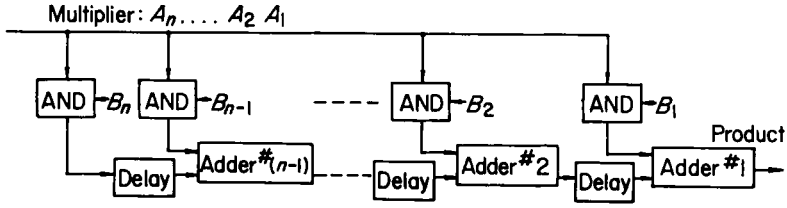


FIG. 6.30. A serial–parallel binary multiplier

since one operand, $B_n \ldots B_2B_1$, is presented in parallel and the other, $A_n \ldots A_2A_1$, is applied serially. It is one of the fastest types of multipliers. The question of which operand is the multiplier and which the multiplicand is arbitrary. For the purpose of explanation it will be assumed that $A_n \ldots A_2A_1$ is the multiplier. If $A_1 = 1$ the AND gates allow the multiplicand to pass through and the least significant bit of the multiplicand appears at the output of adder no. 1 as the least significant bit of the product. The other bits of the multiplicand must traverse a delay equal to the period between the appearance of successive bits of the multiplier. The chain of adders and delays is referred to as a multiplier chain. The delays in the chain serve to store each accumulated sum until the next partial product can be added, and also to shift it so that the partial product is added to the correct orders. For example, when A_2 appears, B_4, B_3, and B_2 of the first partial product will arrive at the inputs to adders no. 3, 2, and 1, respectively. If $A_2 = 1$, then B_3, B_2, and B_1 will also be applied to adders no. 3, 2, and 1, respectively. Note that B_1 of the first partial product, which is the first bit of the product, has already been transmitted out of the multiplier, and that the B_4 just entered is not required until the time of arrival of the next multiplier bit. Then it will be added to a newly entered B_3, provided $A_3 = 1$. The time required for a multiplication is a number of bit periods equal to the sum of the bits in the product.

The arrangement of alternate adding and delay circuits has an advantage in that the number of delay circuits required is small, and also that the delay inherent in each adding circuit can be compensated by a corresponding deficiency in the delay of the following delay circuit. An aperiodic form of delay circuit may be used which delivers its output pulse at the beginning of a bit period even if the input pulse occurs late in the preceding bit period. An interesting feature of this arrangement is that the amount of equipment required is determined by the number of bits in the multiplicand and is independent of the length of the multiplier. However, both operands are normally of the same length.

An interesting variation of the serial–parallel multiplier is employed in the University of Manchester computer. It makes use of the fact that the amount of equipment in a serial–parallel multiplier is dependent only on the number of bits in the operand arbitrarily termed the multiplicand. To save equipment, the multiplier of the University of Manchester computer is designed to accommodate only $n/2$ bits of the multiplicand at a time. This necessitates breaking the multiplication process into two major cycles and doubles the time required for multiplication. In the first cycle, the $n/2$ least significant bits of the multiplicand, and in the second cycle, the $n/2$ most significant bits, are used to control the output of the AND gates. The bits of the multiplier are applied as before. Each "half-product" is entered into an accumulator. The number of delay circuits between each

gate of the multiplier chain and the output of the chain is correct during the formation of the first half-product, but for the second half-product there are in each case $n/2$ too few delays. This is corrected by adding the second half-product to the $n/2$ most significant bits of the number in the accumulator, which is equivalent to shifting the second half-product by $n/2$ places.

Four or five word periods are required, in the University of Manchester computer, to execute simple instructions. This includes one word period for looking up an instruction, one for transferring it to the control unit, one for looking up the operand, and one or two for the actual execution. The multiplication process just described requires only 14 word periods, including look up of the multiplication instruction, extraction of both factors from storage, and adding the product to, or subtracting it from, the contents of the accumulator. A word consists of 20 bits of information (which may be used as an instruction or a number) plus a four-bit blank space. Since the clock rate of the computer is 100 kc, the word period is 240 μsec. Addition of a 40-bit number to the accumulator takes 1.2 msec, and addition of the product of two 40-bit numbers to the accumulator requires 3.36 msec (a speed slow by present standards). The high cost of a low ratio of multiplication to addition time is apparent from the fact that nearly one fourth of the vacuum tubes in the computer are in the multiplier.

In Fig. 6.31 there appears another scheme for reducing the equipment in a serial–parallel multiplier. One operand, B, is applied serially on the line shown. The quantities $2B$ and $3B$ are obtained from it by means of a delay unit and an adder as shown. The bits of the multiplier are grouped in pairs. Each pair can take on any one of four values, namely 0, 1, 2, and 3. Accordingly, either zero, B, $2B$, or $3B$ is added to the adder associated with each pair. Since the quantities applied to adjacent adders in the multiplier chain are separated by a quaternary order, which is equivalent to two binary orders, two-bit delays, $2D$, are inserted between adjacent adders. To illustrate how this multiplier operates, the multiplication of $B = 111011$ by $A = 101101$ will be described: The least and most significant bit of B appear at time t_1 and t_6, respectively. The bits of B, $2B$, and $3B$ appear as shown in the timing chart, Example 6.11.

Example 6.11

	t_8	t_7	t_6	t_5	t_4	t_3	t_2	t_1
B			1	1	1	0	1	1
$2B$		1	1	1	0	1	1	0
$3B$	1	0	1	1	0	0	0	1

Since $A = 10 \quad 11 \quad 01$

$A_2A_1 = 01 = 1$

$A_4A_3 = 11 = 3$

$A_6A_5 = 10 = 2$

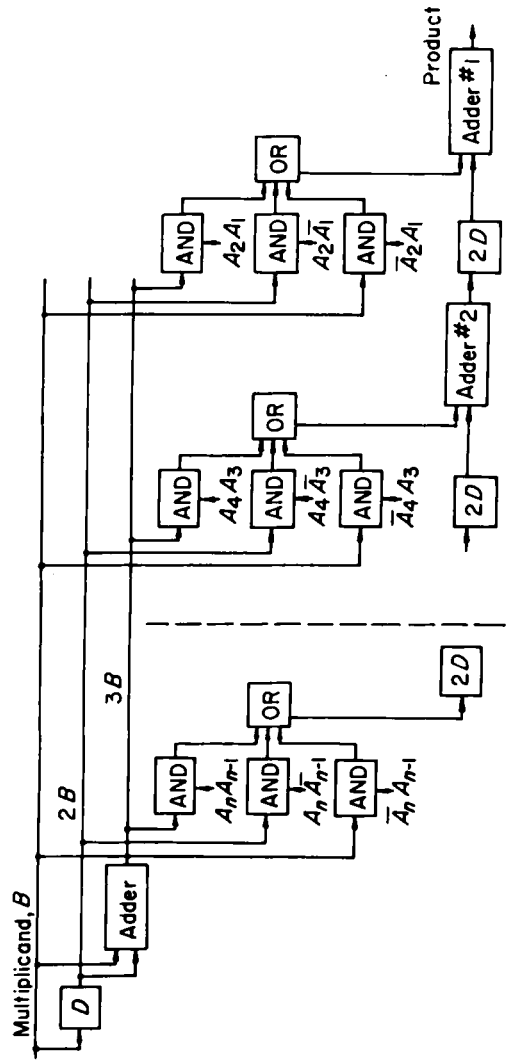


FIG. 6.31. A serial–parallel binary multiplier controlled by paired bits of the multiplier

Therefore, B, $3B$, and $2B$ are entered in adders no. 1, 2, and 3, respectively. The product is obtained by summing these quantities as follows

$$\begin{array}{r} 111011 \\ 10110001 \\ 1110110 \\ \hline 101001011111 \end{array}$$

The correctness of this result is readily verified.

The time required for multiplication using the scheme of Fig. 6.31 is about the same as for the scheme of Fig. 6.30. Comparison of the two arrangements shows that for a large number of bits in the operands, the saving essentially consists of replacing each of several adders with three three-input AND gates. By grouping the bits of the multiplier in even larger groups, and also operating on a group of the multiplicand simultaneously, the time for multiplication can be reduced further still. However, a substantial amount of additional equipment is then required.

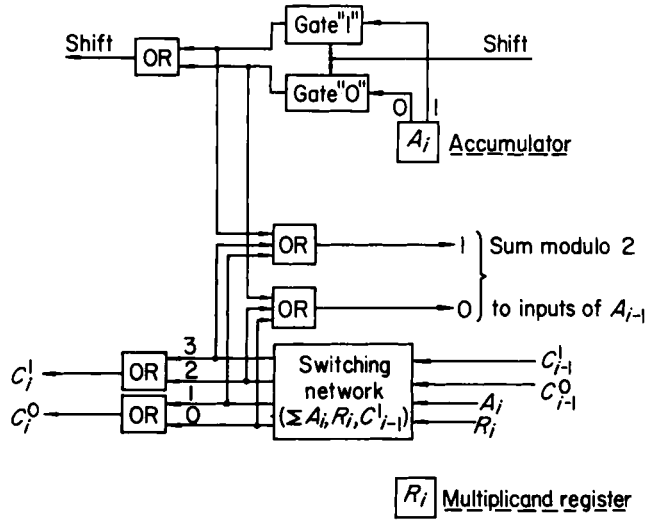


FIG. 6.32. Typical stage of an asynchronous binary multiplier

6.1.5.1.4. AN ASYNCHRONOUS MULTIPLIER. The asynchronous multiplier depicted in Fig. 6.32 is basically the asynchronous adder of Fig. 6.21(b) with auxiliary circuits. One of its important features consists of sending the bits of the accumulated sum directly to the next lower stage, thereby eliminating the command and time required for a separate shifting operation.

Whenever the multiplier bit is a 1, indicating that the contents of the multiplicand register are to be added to the accumulator, a pulse is applied to the no-carry input of the least significant stage. This pulse passes in succession through all stages. The actual path followed is determined by the current contents of the accumulator and of the multiplicand register. The switching network, composed of AND and OR gates, has four output lines labeled 0, 1, 2, 3. Only one of these is activated at a time, and in accordance with the number of the inputs A_i, R_i, and

C_{i-1} which have the value 1. If the total of these inputs is 2 or 3, a carry is sent to the next more significant stage. If the total of these inputs is 1 or 3, the value of their sum (modulo 2) is 1, otherwise it is 0. The value of this sum is sent directly to the next less significant stage of the accumulator. Small delays may be necessary in either the input or output lines of the bistable elements of the accumulator if the delay in the action of each of these elements is not sufficient in itself.

Whenever the multiplier bit is a 0, indicating that the only operation to be performed is that of shifting the contents of the accumulator, a pulse is applied to the shift input line of the least significant stage. This pulse is propagated to the next stage via either gate 0 or gate 1 and an OR gate, as shown in Fig. 6.32. In any stage except the least significant one, the output of either gate 1 or gate 0 is also fed back via one of the three-input OR gates to the input of the bistable element in the next less significant stage of the accumulator, thereby effecting the shift operation.

The completion of either an add and shift, or a shift operation alone, can be sensed by combining in an OR gate the outputs of gate 1, gate 0, and the switching network in the most significant stage. This signal, in turn, can be used to initiate the next operation. It is not always necessary, however, to await completion of one operation before starting the next. It is actually possible to initiate a new operation as soon as the effects of the preceding one have subsided in the least significant stage.

6.1.5.1.5. A SIMULTANEOUS MULTIPLIER. In a so-called simultaneous multiplier, steady state signals representing the operands are applied simultaneously, and after the decay of transients, signals representing the product are available at the output lines. To see how a multiplier of this type can be formed, consider the multiplication of a four-bit multiplicand, $a_4a_3a_2a_1$, by a four-bit multiplier, $b_4b_3b_2b_1$, shown in Example 6.12

Example 6.12

$$
\begin{array}{cccccccc}
 & & & & & a_4a_3a_2a_1 & & \\
 & & & & & b_4b_3b_2b_1 & & \\
\hline
 & & & & a_4b_1 & a_3b_1 & a_2b_1 & a_1b_1 \\
 & & & a_4b_2 & a_3b_2 & a_2b_2 & a_1b_2 & \\
 & & a_4b_3 & a_3b_3 & a_2b_3 & a_1b_3 & & \\
 & a_4b_4 & a_3b_4 & a_2b_4 & a_1b_4 & & & \\
\hline
P_8 & P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1
\end{array}
$$

When a multiplication is performed by a step by step procedure as in

the accumulation method, never more than two a_ib_j terms plus a possible carry from a less significant order have to be added in any order at any given time. In simultaneous multiplication, the maximum number of entries in any particular order depends on both the length of the operands and their value. For example, for $n = 4$, the maximum number of a_ib_j terms occurs in the order that generates P_4 and is equal to 4. If a_1, a_2, a_3, b_1, b_2, and b_3 each have the value 1 then two carries will be produced in the order where P_3 is generated, so that a total of six entries must be added to produce P_4.

If we employ only two and three-input adders, the sum of six entries can be obtained by the use of two three-input and one two-input adders. The maximum number of inputs for each column can be obtained by assuming all bits of both operands equal 1. The maximum number of entries in each order (which is equal to the number of a_ib_j terms plus the maximum number of carries from the next lower order) as well as the maximum number of carries that can be generated by each order, for $n = 4$, is shown in Fig. 6.33. The total equipment required consists

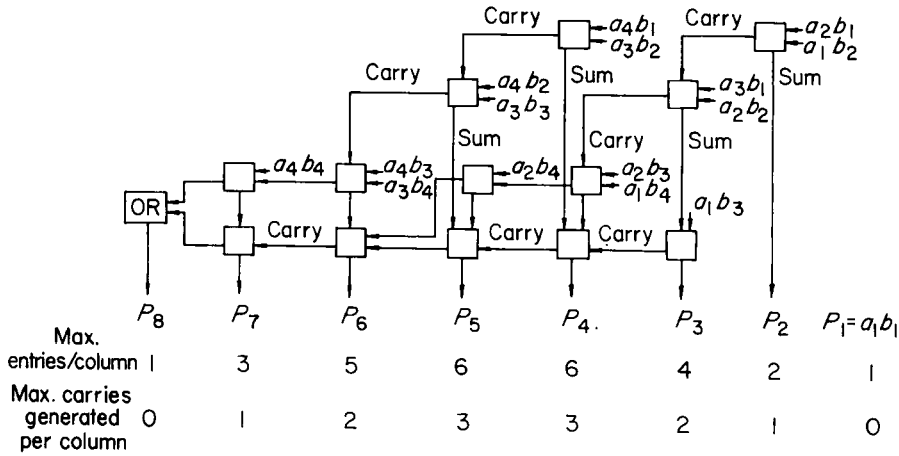


FIG. 6.33. A simultaneous binary multiplier

of a two-input AND gate for the mechanization of each a_ib_j term plus whatever adders are required to produce each P_i. Figure 6.33 indicates a particular interconnection of adders. However, any of a number of other arrangements could have been chosen. Note, too, that the two two-input adders in column P_7 and the OR gate in column P_8 are actually equivalent to the three-input adder of Fig. 6.12 with the deletion of the

one-bit delay feedback path. The time required for a multiplication depends on the number of gates and adders through which the input signals must pass before reaching the output. A reasonable estimate can be obtained by determining the longest path which any input may have to traverse. The price of high speed in this type of multiplier is the great number of components required for practical lengths of the operands, this number pyramiding as the operand length is increased.

6.1.5.1.6. MULTIPLIERS FOR OPERATING ON NEGATIVE NUMBERS IN TWO'S COMPLEMENT FORM. In this section three different schemes are described which can be used in mechanizing multipliers that can operate directly on operands expressed in a two's complement form. In the first of these, a pseudo-product is formed, and a particular correction is applied to it, in accordance with which of the operands is negative (all such corrections are referred to as end point corrections). The second scheme does not require such end point corrections. Instead, the normal multiplication algorithm is amended to compensate for negative operands, as the succeeding partial products are formed. The third scheme utilizes an algorithm for multiplication that is independent of the signs of the operands. Each of these schemes will now be described in detail.

In the first scheme*, multiplication is always performed as if both numbers were positive. Then corrections are applied to the result if one or both operands is negative. Consider first the multiplication of two positive numbers, as shown in Example 6.13.

Example 6.13

$x +$	0.1001	9/16
$y +$	0.0111	7/16
	1001	
	1001	
	1001	
	0.00111111	63/256

If the multiplier digit is 1, the multiplicand is added into the partial product and the result shifted one place to the right. If the multiplier digit is 0, only the shift is required.

* Goldstine, H. H. and von Neumann, J. [1947] *Planning and Coding of Problems for an Electronic Computing Instrument*, Institute for Advanced Study, Princeton, N.J. (U.S. Army Ordnance Contract W-36-034 ord 7481).

Consider next the case of a positive multiplicand and a negative multiplier, as shown in Example 6.14.

Example 6.14

$x+$	0.1001	x	9/16
$y-$	1.1001	$(2-y)$	7/16
	1001		
	0		
	0		
	1001		
	0.01010001	$(x-xy)$	
	1.0111	$-x$	
	1.11000001	$-xy$	$-63/256$

Note, first of all, that the multiplicand is not multiplied by the sign digit of y. This is equivalent to multiplying x by $(2-y)-1 = x-xy$. In order to obtain the correct product, i.e., $-xy$, $-x$ must be added to $x-xy$. Therefore, if on inspection the sign digit of y is found to be 1, indicating that y is negative, the complement of x is added to the product $x-xy$ already obtained to produce the correct product, $-xy$.

Now consider the case of a negative multiplicand and a positive multiplier, as shown in Example 6.15.

Example 6.15

$x-$	1.0111	$(2-x)$	$-9/16$
$y+$	0.0111	y	7/16
	0111		
	0111		
	0111		
	10000	Sign only of x	
	0.10110001		
	1.0001	Final correction	
	1.11000001	$-xy$	$-63/256$

If the sign digit of x is not included in the multiplication, the product $(1-x)y$ is produced. To obtain the correct product, i.e., $-xy$, $-y$ must be added to $(1-x)y$. However, in many computing machines, once a

digit of the multiplier has been examined it is shifted to the right and lost, i.e., no storage is provided for it in the arithmetic unit, for reasons of economy. However, even though y is not available at the end of the process for correcting the product, a procedure may be used whereby the necessary correction is accomplished as the partial product is being built up. This procedure is as follows: (1) Where $y = 0$, 1 is added to the uncorrected product. This 1 may automatically be added in the correct place if only the sign digit of x is added to the partial product when $y = 0$. This sign digit occupies a position in the product of the same order as the digit of y which was 0, and controlled its addition into the product. Since it is desired to correct the partial product by the complement of y, 0 should be added to the partial product where $y = 1$, and 1 where $y = 0$, except for the least significant place. The sign digit of x during any addition is entered into an order of the product equivalent to that of the digit of y which controlled its addition into the product. (2) Since the operation is halted before the sign of y is examined, a 1 in the sign position and a 1 in the units position is added to the partial product as a final correction. Inspection of the preceding example shows that the sum of the corrections added is equal to 1.1001, which is the complement of $y = 0.0111$.

Finally, there is the case of two negative operands, shown in Example 6.16.

Example 6.16

$x-$	1.0111	$2-x$	$-9/16$
$y-$	1.1001	$2-y$	$-7/16$
	0111		
	10000		(a)
	10000		(b)
	0111		
	0.10011111		
	1.0001		(c)
	1.10101111	$xy-x$	
	0.1001	$+x$	
	0.00111111	xy	63/256

Here the sign digit of neither x nor y is included in the multiplication, producing an uncorrected product equal to: $(1-x)(1-y) = 1-x-y-xy$. By adding the sign digit of x wherever $y = 0$, (lines (a), (b)), plus the term 1.0001 (line (c)), the complement of $-y$, *i.e.*, $+y$ is added to the product. Note that since y is negative, its complement would contain a 0 in the sign digit, but by writing the correction term with a 1

in the sign digit, the extra 1 in the product $1 - x - y + xy$, is also corrected for. This is due to the fact that, since all carries beyond the sign are neglected, adding 1 is equivalent to subtracting 1. For example

$$\begin{array}{ll} 0.0100 & \tfrac{1}{4} \\ 1.0000 & \\ \hline 1.0100 & = -\tfrac{3}{4} \end{array}$$

The process is completed by adding the complement of $-x$, i.e., $+x$.

We will consider now the second scheme. Again, multiplication is normal if both multiplier and multiplicand are positive. If either one or both are negative, the normal multiplication algorithm is amended as follows: (1) If only the multiplier is negative, then upon reaching the sign columnar position in the multiplier, add the complement of the multiplicand into the partial product instead of the value of the multiplicand (which would normally be done upon detection of a 1 in a columnar position of the multiplier). (2) If only the multiplicand is negative, whenever a 1 is detected in the multiplier, in addition to adding the multiplicand to the partial product, add the value of the sign digit of the multiplicand to all columnar positions of the partial product, extending through the sign position. For the case where both multiplier and multiplicand are negative, the multiplication algorithm includes both of these procedures. Specific examples illustrating this method are shown in Example 6.17.

Example 6.17

Multiplicand	0.1101	13/16		Multiplicand	1.0011	− 13/16	
Multiplier	0.1011	11/16		Multiplier	0.1011	11/16	
	01101				111110011		
	01101				11110011		
	00000				0000000		
	01101				110011		
	00000				00000		
	0.10001111	=	143/256		1.01110001	=	− 143/256
Multiplicand	0.1101	13/16		Multiplicand	1.0011	− 13/16	
Multiplier	1.0101	− 11/16		Multiplier	1.0101	− 11/16	
	01101				111110011		
	00000				00000000		
	01101				1110011		
	00000				000000		
	10011				01101		
	1.01110001	=	− 143/256		0.10001111	=	143/256

We will consider next a multiplication scheme that is independent of the signs of the operands. It is described in a paper by A. D. Booth.* At first glance, this scheme appears to be far removed from any of the well known methods of multiplication. Briefly, successive addends in the partial product are produced in accordance with the values of successive pairs of bits in the multiplier. Starting with the least significant bit, each bit in the multiplier is compared with the bit lying to the right of it (the bit lying to the right of the least significant bit is always considered to be zero). Each pair of bits can assume four different configurations. The operation performed in building up the partial product is determined according to which configuration exists at each step. If the bits of the multiplier are designated by $a_0.a_1\, a_2\, a_3 \ldots a_n$ and the bits of the multiplicand r by $r_0.r_1\, r_2\, r_3 \ldots r_n$, the algorithm for multiplication may be stated as in Table 6.15.

TABLE 6.15. Rules of multiplication based on values of successive pairs of bits in the multiplier: $a_0.a_1a_2a_3 \ldots a_n$.

Value of successive multiplier bits		Operation
a_i	a_{i+1}	
For $i \neq 0$:		
0	0	The current partial product is shifted one bit to the right.
0	1	The multiplicand is added to the partial product, and then the new partial product is shifted one bit to the right.
1	0	The two's complement of the multiplicand is added to the partial product and then the new partial product is shifted one bit to the right.
1	1	The current partial product is shifted one bit to the right.
For $i = 0$:		
0	0	Do nothing.
0	1	Add the multiplicand to the partial product.
1	0	Add the two's complement of the multiplicand to the partial product.
1	1	Do nothing.

The steps in the multiplication of $-13/16 = 1.0011$ by $-11/16 = 1.0101$

* Booth, A. D. [1951] A signed binary multiplication technique, *Quart. Journ. Mech. and Applied Math*, IV, Pt. 2, 236-40.

are shown in Example 6.18. It is assumed that the product is formed in an accumulator, the left half of which only can form sums, and both parts of which can shift their contents to the right.

Example 6.18

	Contents of accumulator									
	Left half					Right half				
$2-r$	0	1	1	0	1					
Shifted sum	0	0	1	1	0	1				
r	1	0	0	1	1					
Sum	1	1	0	0	1	1				
Shifted sum	1	1	1	0	0	1	1			
$2-r$	0	1	1	0	1					
Sum	0	1	0	0	1	1	1			
Shifted sum	0	0	1	0	0	1	1	1		
r	1	0	0	1	1					
Sum	1	0	1	1	1	1	1	1		
Shifted sum	1	1	0	1	1	1	1	1	1	
$2-r$	0	1	1	0	1					
Sum	0	1	0	0	0	1	1	1	1	

The product 0.10001111 = 143/256 appears on the bottom line. Two points should be noted in the procedure for producing the product. First, during the addition of addends to the partial products, that part of the partial product which is in the shift register is left undisturbed. Second, during a shift operation, the value of the sign bit in the accumulator remains unchanged.

6.1.5.2. Decimal Multiplication

Most stored program digital computers operate in the binary system internally. Usually, information is entered and read out in a binary-coded decimal form. Input and output conversion programs are used to effect the transition from binary-coded decimal to binary and vice versa, respectively. In some machines, especially those for business applications, it may be desirable for the machine to operate internally in the binary-

coded decimal system. Then data can be entered and read out without conversion. Decimal multipliers generally are more complex than binary multipliers and there are a great number of schemes on which such multipliers can be based. However, only a few of them will be described here.

6.1.5.2.1. MULTIPLICATION BY REPEATED ADDITION. In this scheme, the digits of the multiplier are inspected in sequence, and each partial product is formed by adding the multiplicand a number of times equal to the multiplier digit. A convenient way of controlling the process is to shift each digit of the multiplier into a decimal counter. If the number in the counter is not 0, the multiplicand is entered into an accumulator, and 1 is subtracted from the contents of the counter. This process is repeated until the counter contains 0. Then the next more significant digit is entered into the counter. Whenever a new multiplier bit is placed in the counter, a signal is also produced which causes the next partial product to be appropriately shifted when added to the accumulated sum.

One way of lessening the time required to produce the product is to use a subtractor as well as an adder. Then, the partial products called for by multiplier digits 6 through 9 can be obtained by subtracting the multiplicand a number of times equal to the ten's complement of the multiplier digit and then adding 1 to the multiplier digit in the next higher order. This scheme reduces the number of operations required, on the average, to generate the product. For example, consider the multiplication of some number, M, by 28. Instead of adding M eight times, and then adding $10M$ twice (a total of ten operations), M would be subtracted twice, and $10M$ would be added three times (a total of only five operations).

Another way to increase the speed of a repeated addition process is to generate double the multiplicand. If both $2M$ and M are made available, the generation of any multiple of M, from 2 through 9, can be made with fewer additions than if only M were used to build up the partial product. Generating $2M$ from M, when M is expressed as a straight binary-coded decimal, is relatively simple. The value of each order in the doubled digit and the carry can be obtained by relatively simple logical circuits, each of which generates the value of a particular order in $2d$ from the logical sum of all values of d that produce it. For example, from Table 6.16, the values of d that produce a 1 in the D_{24} position of $2d$ are given by the Boolean equation $D_{24} = \bar{D}_{14}D_{13}\bar{D}_{12}\bar{D}_{11} + D_{14}\bar{D}_{13}\bar{D}_{12}D_{11}$. A doubler can thus be formed from simple combinational circuits that produce the values of D_{25}, D_{24}, D_{23}, D_{22}. The case of D_{21} is trivial since it is always equal to 0.

Another multiple that is relatively easy to generate is the fifth. Inspection of Table 6.16 shows that the value of the least significant bit of d can be used to indicate whether the right hand part of $5d$ is 0 or 5, and the other three bits of d correspond to the left hand part of $5d$, i.e., to the value of the carry. Another way of producing the fifth multiple is to first divide d by 2 and then multiply the result by 10. The division by 2 is accomplished simply by a shift to the right. If d is odd, there is a carry to the next lower order digit. The carry always has the value 5, and is added to the lower order after the latter has been divided by 2. Finally, the multiplication by 10 is accomplished by a shift of one digit to the left.

TABLE 6.16. Second and fifth multiples of the straight binary-coded decimal

d $D_{14}D_{13}D_{12}D_{11}$	$2d$ $D_{25}D_{24}D_{23}D_{22}D_{21}$	$5d$
0000	0000	0000
0001	0010	0101
0010	0100	1 0000
0011	0110	1 0101
0100	1000	10 0000
0101	1 0000	10 0101
0110	1 0010	11 0000
0111	1 0100	11 0101
1000	1 0110	100 0000
1001	1 1000	100 0101

The generation of multiples other than the second and fifth introduces complications. This is because carries added into any given order may affect the value of the carry to be added to the next higher order. Such a situation cannot occur with doubling or quintupling, for the sum of any left and right hand part of $2d$ or of $5d$ cannot exceed 9. This is shown in Table 6.16.

If one provides two doublers and a quintupler any multiple may be formed as needed with only one addition operation by causing these units to be connected in various ways under control of the multiplier digit. The output of one doubler is used as an input to the other, thereby generating the fourth multiple. All multiples from 2 through 9 can be obtained by combining the multiplicand B, and the outputs of the doubler, $2B$, the quadrupler (i.e., the two cascaded doublers), $4B$, and the quintupler, $5B$, as shown in Example 6.19.

Example 6.19

Multiple desired	Required inputs to adder
B	B
$2B$	—
$3B$	$2B$, B
$4B$	—
$5B$	—
$6B$	$5B$, B or $4B$, $2B$
$7B$	$5B$, $2B$
$8B$	$4B$, $4B$
$9B$	$5B$, $4B$

Any of the schemes that have been described for generating multiples can be combined. The average number of operations per multiplier digit shown below is approximate for schemes using subtraction only in that any carry beyond the most significant digit is neglected. (Also, in determining this figure for such schemes the carry to the next higher order may be neglected because its effect over all values of a digit cancels out.

Addition only	4.5
Addition and subtraction	$\approx$ 2.5
Addition and doubling	2.5
Addition and quintupling	2.5
Addition, subtraction, and doubling	$\approx$ 1.5
Addition, subtraction, doubling and quintupling	$\approx$ 1.3
N-tupling (simultaneous generation of all multiples)	0.9

6.1.5.2.2. A SERIAL–PARALLEL MULTIPLIER. Serial–parallel binary multipliers are described in Section 6.1.5.1.3. An arrangement for handling numbers expressed in a decimal code is considerably more complicated. A particular one is shown in Fig. 6.34. It is assumed that the individual bits of each digit in both operands appear in parallel. Accordingly, each heavy line in Fig. 6.34 actually represents four lines. The digits, A_i, of the multiplier are presented in parallel, and the digits of the multiplicand appear serially. It is assumed, also, that all nine multiples of the multiplicand are available from some type of N-tupler arrangement. The proper multiple of the multiplicand, corresponding to the value of a multiplier digit, is channeled into a decimal adder via a 40-input, four-output many-to-one function table controlled by the multiplier digit. The inputs to each table consist of four lines carrying the bits of the multiplier digit, A_i, and 36 lines carrying the nine possible multiples

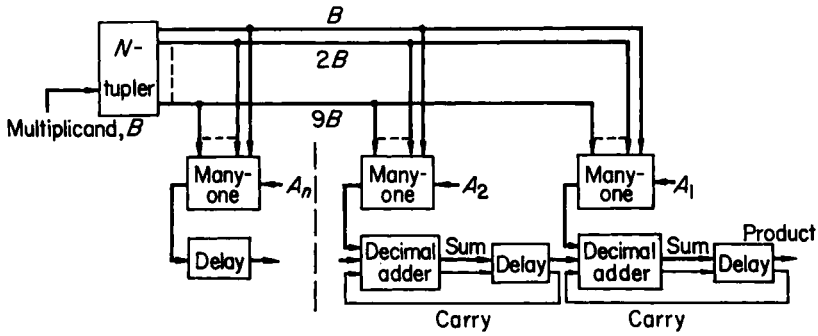


FIG. 6.34. A serial–parallel decimal multiplier

of the multiplicand digits. The one-digit delays associated with each adder serve two functions. They appropriately shift each partial product before it is added to the partial product in the next lower order, and also delay the carries generated in the summation of partial products. The digits of the product will appear serially at the point shown. The time required for generation of the product is the time required for serial transmission of the digits of the product.

6.1.5.2.3. MULTIPLICATION BY HALVING THE MULTIPLIER AND DOUBLING THE MULTIPLICAND. A multiplier of this type is shown in Fig. 6.35. One factor, say the multiplier, is repeatedly halved while the

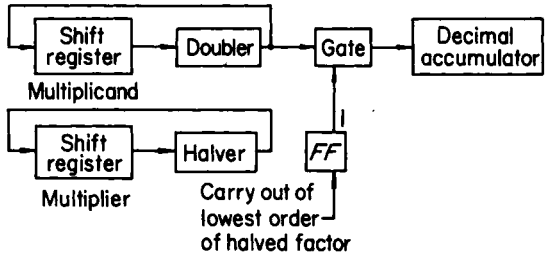


FIG. 6.35. Decimal multiplication by halving one factor, doubling the other

other is doubled. This process is continued for a number of cycles until the multiplier has been reduced to 0. Whenever a remainder of 1 is obtained from halving the multiplier, the product of the multiplicand and the appropriate power of 2 is entered into the accumulator. This remainder

can be anticipated, since it will always occur when the least significant digit is odd, indicated by the least significant bit of the lowest order binary-coded decimal digit being equal to 1. Essentially, the binary-coded decimal multiplier is converted to a pure binary number (see Section 6.4.3) in which the location of 1's is determined by where there are remainders.

Example 6.20 Multiplication of 36 by 13

Halved operand	Remainder	Doubled operand	Partial products
13			
6	1	36	36
3	0	72	
1	1	144	144
0	1	288	288
			468

Note that the sequence of the remainder bits forms the binary equivalent of the multiplier: 1101. Accordingly, the product is $(2^3 + 2^2 + 1)$ times the multiplicand.

Halving can be accomplished by shifting each binary-coded group one bit to the right. Whenever the least significant bit of a group is equal to 1, indicating there will be a remainder (or carry to the next lower order) after a shift, the number 5 must be added to the next lower order. The addition of the carry, 5, to any order cannot cause another carry because no code group can have a value greater than 4 after a shift, and before addition of the carry.

6.1.6. DIVISION

6.1.6.1. Binary Division

6.1.6.1.1. TRIAL AND ERROR OR RESTORING METHODS. In working out any division process, three important items must be taken into account. Each of these is described in the paragraphs following:

First of all there must be a determination of the correct orders of the dividend from which the divisor is to be subtracted initially. Because a computer does not ascertain relative magnitudes by observation, rules different from those used by humans must be used. One way to facilitate the initial subtraction problem is to place restrictions on the relative magnitudes of the dividend and divisor. In a fractional computer the restriction on the size of the operands is simple: the divisor must be larger than the dividend, or else the quotient would be greater than one, and beyond

the capacity of the machine. A simple way to test for this is to line up the binary points and subtract the divisor from the dividend. If the remainder is negative, the divisor is greater than the dividend. The initial steps of a division process may be modified to meet special situations, as for example, in a machine with built-in floating point representation of numbers. (See Section 6.3). In this case, an automatic shift, with corresponding adjustment of an exponent, can be actuated which will cause the mantissa of the dividend to be less than that of the divisor.

Another important item is the procedure to be followed when subtraction of the divisor from the dividend or a remainder produces a negative result. An obvious way to nullify the subtraction is to add the divisor back into the remainder. Whenever the subtraction of the divisor from the old remainder leaves a new, positive remainder, or when the divisor is added back to the old remainder to restore the previous remainder, the divisor is shifted one place to the right before being subtracted from the new remainder. The number of additions and subtractions required to complete a division can be reduced as follows: Instead of correcting a negative remainder by adding the divisor back, the remainder may be shifted one place to the left before adding the divisor. The operation of the first method can be expressed as $+y-y/2$ and that of the second as simply $+y/2$. If, after adding $y/2$, the remainder is still negative, it is known that the next quotient bit is 0, and the divisor is shifted another position to the right and added. The second procedure, which is equally applicable to serial or parallel machines, may be summarized as follows: (a) Test the dividend and each remainder. If it is $\geqslant 0$, subtract the divisor, otherwise add. Whenever, after an addition or subtraction, the remainder is $\geqslant 0$, a 1 is recorded in the quotient, otherwise a 0. (b) Shift the remainder one place to the left and repeat step (a).

A third procedure which has several variations is similar to the pencil and paper method of division. The divisor is compared with appropriate orders of the dividend or remainder and a subtraction is executed only when the comparison indicates the new remainder will be positive. If the digits appear with the less significant digits first, a comparator (see Section 6.1.4.3) may be used. The final setting of the comparator indicates whether a negative remainder will be produced by the subtraction of the divisor from the old remainder. An over-all system of operation can be devised wherein the comparison for the succeeding subtraction is performed at the same time that a given subtraction is being executed. Also, if the machine has a type of accumulator (Fig. 6.26) wherein the sign of the difference is available before the difference itself is formed, the sign may be tested and the subtraction aborted if a negative difference is indicated.

An example of binary division is given in Example 6.21. Since each digit can be only 1 or 0 more than one trial is never required before a restoration

Example 6.21

$\frac{x}{y} = \frac{7/16}{10/16}$:

		Quotient digits
	xxxx	
	0.1011	
0.1010	0.0111 0000	
	0.1010	
	Restore	
		0
	0.01110	
	0.01010	
	0.001000	1
	0.001010	
	Restore	
		0
	0.0010000	
	0.0001010	
	0.00001100	1
	0.00001010	
	0.00000010	1

The third major item to be considered in a division process is the disposition of the remainder after the quotient has been obtained to the precision desired. Each of the three procedures described for producing a quotient yields a value for the quotient that represents the largest multiple of the divisor equal to or less than the dividend. The final remainder is the difference between this largest multiple and the dividend. There are occasions when the final remainder may be of use to a programmer. To obtain the final remainder, it is necessary that the divisor be added back into the orders from which it was last subtracted. This is easily done if the first method of correcting a negative remainder is used. However, in the second method a special operation is required because the normal procedure is to shift the remainder left once before adding. Also, in some cases the final remainder must be obtained by a subtraction instead of an addition. The corrective steps which must be taken at the conclusion of the division process cause substantial complications in the design. Usually, however, the final remainder is of no interest and, therefore, discarded. Even then, it may be obtained when needed by subtracting

the product of the unrounded quotient and the divisor from the dividend.

The preceding description of the division process considered only positive operands. This does not introduce any loss of generality if the operands are in signed form. As in multiplication, the signed bits are ignored except for the simple comparison required to determine the sign to be attached to the result. Each of the division procedures described can also be used for operands expressed in a two's complement form. For example, the second procedure for correcting a negative remainder may be modified in the following way (a) Test the dividend or remainder. If it is zero or has the same sign as the divisor, subtract the divisor, otherwise add. Whenever after an addition or subtraction, the remainder is found to be zero or have the same sign as the divisor, a 1 is recorded in the quotient, otherwise a 0. (b) Shift the remainder one place to the left, and repeat step (a).

A negative quotient appears in complementary form. The binary point is after the first recorded bit. Complications arise in the determination of the least significant quotient digit. A possible round-off procedure is to always place a 1 in the least significant place of the quotient. When this procedure is used, the two's complement representation of operands is satisfactory. If a more accurate round-off procedure is required, it is preferable to convert the operands to a signed form before the division process.

6.1.6.1.2. THE NONRESTORING METHOD OF DIVISION. The methods of division described so far fall for obvious reasons into the category of so-called trial and error, or restoring methods. A different scheme, referred to as the nonrestoring method, has the following important advantages: First of all, it eliminates the operations of inspecting the remainder and restoring it when required. Secondly, it requires no extra correction operations if dividend and/or divisor are negative.

In a nonrestoring method, -1 should be used in the quotient if the divisor y is added to the remainder, and $+1$ if y is subtracted. Since an accumulator is to be used to hold the dividend and, subsequently, the remainders, it is more convenient to store the quotient digits in a register rather than an accumulator. However, a difficulty arises from the fact that the quotient register has no way of distinguishing -1 from $+1$. One solution is to place a 0 in the quotient wherever a -1 should appear, and to seek a simple relationship between this pseudo-quotient and the true quotient. Such a relationship will now be derived: We begin by writing an expression for the new remainder r_k in terms of the old remainder r_{k-1}, the pseudo-quotient digit p_k, and the divisor y.

$$r_k = 2r_{k-1} + (1-2p_k)y \qquad (6\text{-}13)$$

$$\text{if } p_k = 0, \quad r_k = 2r_{k-1} + y$$

$$\text{if } p_k = 1, \quad r_k = 2r_{k-1} - y$$

The factor of 2 appears in the recursion equation (6-13) since the old remainder was shifted to the left before the addition or subtraction of y. Multiplying Eq. (6-13) by 2^{-k} yields

$$2^{-k}r_k = 2^{-(k-1)}r_{k-1} + [2^{-k} - 2^{-(k-1)}p_k]y.$$

Setting $k = 1, 2$ and designating the initial remainder r_0 by the dividend x:

$$2^{-1}r_1 = x + (2^{-1} - 2^0 p_1)y$$

$$2^{-2}r_2 = 2^{-1}r_1 + (2^{-2} - 2^{-1}p_2)y$$

$$= x + [(2^{-1} + 2^{-2}) - (2^0 p_1 + 2^{-1}p_2)]y.$$

In general

$$2^{-n}r_n = x + \left[\sum_1^n 2^{-k} - \sum_1^n 2^{-(k-1)}p_k \right] y \qquad (6\text{-}14)$$

where

$$\sum_1^n 2^{-k} = 0.1 + 0.01 + 0.001 + \ldots = 0.111\ldots = 1 - 2^{-n} \qquad (6\text{-}15)$$

Substituting (6.15) in (6.14), and transposing

$$x = [-1 + 2^{-n} + \sum_1^n 2^{-(k-1)}p_k]y + 2^{-n}r_n.$$

Finally

$$\frac{x}{y} = [-1 + 2^{-n} + \sum_1^n 2^{-(k-1)}p_k] + \frac{2^{-n}r_n}{y}. \qquad (6\text{-}16)$$

In Eq. (6.16), the first digit, p_1 of the pseudo-quotient corresponds to the sign digit since for $k = 1$, $2^{-(k-1)}p_k = 2^0 p_1$, i.e., p_1 is in the 2^0 position, which is the sign position. From Eq. (6-16), it is clear that to convert the pseudo-quotient to the true quotient it is only necessary to add $(2^{-n} - 1)$ to the pseudo-quotient. Since carries beyond the sign position are discarded, subtracting 1 is the same as adding 1. Therefore, the

true quotient may be formed by transferring the pseudo-quotient from the register into the accumulator and adding units into the sign digit and the least significant digit.

Note that in doubling the remainder in the division process described, the sign digit is lost. This introduces no error. The addition or subtraction of the divisor is always such as to decrease the absolute value of the remainder. If it is specified to begin with that $|x| < |y|$, $|2x - y| < |y|$ and, in general $(|2r| - |y|) < |y|$. If also, $|x| < 1$, $(|2r| - |y|) < 1$, and therefore the result of the operation is always within the capacity of the registers.

The procedure of division just described is summarized by the following set of rules: (1) Compare the sign digit of the divisor with that of the remainder. (2) If the signs are alike, place a 1 in the pseudo-quotient, shift the remainder one binary place to the left, and add the complement of the divisor to the shifted remainder. If the signs are not alike, place a 0 in the pseudo-quotient, shift the remainder one binary place to the left, and add the divisor to the shifted remainder. (3) After the pseudo-quotient has been obtained through the $2^{-(n-1)}$th binary place, add $(1 + 2^{-n})$ to it to produce the true quotient. See Examples 6.22 and 6.23.

Example 6.22

$$\frac{x}{y} = \frac{7/16}{10/16} : \underline{0.1010} \overline{)\ 0.0111}$$

	$x = r_0$	
0.1110		$p_1 = 1$
1.0110		
0.0100	r_1	
0.1000		$p_2 = 1$
1.0110		
1.1110	r_2	
1.1100		$p_3 = 0$
0.1010		
0.0110	r_3	
0.1100		$p_4 = 1$
1.0110		
0.0010	r_4	

Pseudo-quotient	1.101
Correction	1.0001
True quotient	0.1011

The solution can be checked by substituting the result in Eq. (6.16)

$$\frac{x}{y} = [\text{True quotient}] + \frac{2^{-n}r_n}{y}$$

$$= 0.1011 + \frac{2^{-4}\ (0.0010)}{0.1010}$$

$$= 11/16 + \frac{2^{-4}(1/8)}{5/8}$$

$$= 0.6875 + 0.0125 = 0.7000$$

Example 6.23

$$\frac{x}{y} = \frac{7/16}{-10/16}:$$

$1.0110 \overline{)\ 0.0111}$

	$x = r_0$	
0.1110		$p_1 = 0$
1.0110		
0.0100	r_1	
0.1000		$p_2 = 0$
1.0110		
1.1110	r_2	
1.1100		$p_3 = 1$
0.1010		
0.0110	r_3	
0.1100		$p_4 = 0$
1.0110		
0.0010	r_4	

Pseudo-quotient	0.010
Correction	1.0001
True quotient	1.0101

Check

$$\frac{x}{y} = [\text{True quotient}] + \frac{2^{-n}r_n}{y}$$

$$= 1.0101 + \frac{2^{-4}\ (0.0010)}{1.0110}$$

$$= -11/16 + \frac{2^{-4}\ (1/8)}{-5/8}$$

$$= -0.6875 - 0.0125 = -0.7000$$

If a machine does not have a built-in facility for division, division may still be accomplished by means of a program based on an iterative formula not explicitly involving division (see Section 6.2.2).

6.1.6.2. Decimal Division

Decimal division presents the same general problems as binary division. Again, the correct orders of the dividend from which to subtract the divisor must be determined, because the equivalent of visual inspection is not readily mechanized. A simple automatic procedure, approximate but adequate to determine relative magnitudes of divisor and dividend is to sense the zeros in all orders higher than the highest order containing a nonzero digit. Then, the highest order nonzero digits in each operand can be lined up automatically. This procedure may sometimes cause the first quotient digit to be zero, but there is no objection to this. In floating point machines (see Section 6.3), automatic means are provided to cause the digits in each number to be shifted so that the highest order nonzero digit appears at the left end of the number.

Another problem is when to stop subtracting the divisor from one set of orders of the dividend, and start subtracting it from a less significant set of orders. The most straightforward solution is to use the restoring method of division wherein the divisor is repeatedly subtracted from one set of orders of the dividend until a negative remainder is produced. The actual number of operations (subtractions and additions) required to produce each digit of the quotient is then two greater than the digit, since an extra subtraction and a subsequent compensating addition will always be made. An exception occurs in the case when nine subtractions of the divisor are performed without producing a negative remainder. Then, unless an error has been made, it is known that the quotient digit must be nine, and the extra two operations can be suppressed.

Another general problem, that of round off of the quotient, is discussed in Section 9.4.

A number of schemes are available for increasing the speed of the division process. In one scheme, instead of restoring the remainder after it becomes negative, one shifts the divisor D to the right and repeatedly adds it to the remainder until it becomes positive. This method is based on recognizing that $10^nD = 10(10^{n-1})D$ and therefore $10^nD - j(10^{n-1})D$ can be replaced by $(10-j)(10^{n-1})D$ where n is the most significant order of the quotient, and j the number of iterations causing a negative remainder in the conventional restoring method. In this modified scheme the quotient digit q equals $j-1$ or $10-k$, where k is the number of additions.

Dividend:	1111 62	$j_3\ k_2\ j_1$
Divisor:	382	1
	729	
	382	2
	347	
	382	3
	−965 62	
	38 2	3 1
	003 82	
	3 82	3 1 2
Quotient:	$(j_3-1)(10-k_2)(j_1-1)$	= 2 9 1

Doubling or quintupling schemes may also be used for decimal division. When using quintupling, the first quotient digit is determined by subtracting the quintupled divisor from the dividend. Whether this or any subsequent subtraction of $5D$ is followed by an addition or subtraction depends on whether that operation changed the remainder's sign. In keeping

TABLE 6.17. Outline of division procedure utilizing quintupled values of the divisor

Quotient digit	Subtractions first	Additions first
0	$-5D+D+D+D+D$	$+5D+D+D+D+D$
1	$-5D+D+D+D+D$	$+5D+D+D+D+D$
2	$-5D+D+D+D$	$+5D+D+D+D$
3	$-5D+D+D$	$+5D+D+D$
4	$-5D+D$	$+5D+D$
5	$-5D-D$	$+5D-D$
6	$-5D-D-D$	$+5D-D-D$
7	$-5D-D-D-D$	$+5D-D-D-D$
8	$-5D-D-D-D-D$	$+5D-D-D-D-D$
9	$-5D-D-D-D-D$	$+5D-D-D-D-D$

with the procedure described in the preceding paragraph, whenever a negative remainder results from one of the sequences of operations shown in Table 6.17 (the case for digits 0, 5, 6, 7 and 8) it is not restored and the next sequence is added to it after a one digit shift to the right. On the average there are 3.8 operations per quotient digit.

If doubling is combined with quintupling, addition, and subtraction, as shown in Table 6.18, operations per quotient digit are reduced to 3.4.

TABLE 6.18. Outline of division procedure utilizing quintupled and doubled values of the divisor

Quotient digit	Subtractions first	Additions first
0	$-5D+2D+2D$	$5D+2D+2D$
1	$-5D+2D+2D-D$	$5D+2D+2D-D$
2	$-5D+2D+2D-D$	$5D+2D+2D-D$
3	$-5D+2D-D$	$5D+2D-D$
4	$-5D+2D-D$	$5D+2D-D$
5	$-5D-2D+D$	$5D-2D+D$
6	$-5D-2D+D$	$5D-2D+D$
7	$-5D-2D-2D+D$	$5D-2D-2D+D$
8	$-5D-2D-2D+D$	$5D-2D-2D+D$
9	$-5D-2D-2D$	$5D-2D-2D$

For the quotient digits 0, 1, 3, 5, and 7, the remainder after the indicated operations will be negative, and therefore addition should be used first to obtain the next quotient digit.

The number of operations per quotient digit may be reduced to 1.0 if all nine multiples of the divisor are provided. Then, each multiple can be compared with the dividend or remainder in a separate comparison circuit, and the quotient digit determined by the largest multiple which leaves a positive remainder after subtraction. Each comparison circuit can be simpler than a subtractor for only the sign of the difference is required. After the determination of the quotient digit is made, it is actually subtracted from the remainder. In a serial computer, the comparison process for determining the next quotient digit can proceed simultaneously with the subtraction of the divisor multiple corresponding to the quotient digit just determined.

6.2. Algebraic and Trigonometric Function Generation

6.2.1. DERIVATION OF A GENERAL ITERATIVE FORMULA

To find a root of the equation, $f(x) = 0$, begin by estimating a value x_n. If the point x_n is chosen at random, a point x_{n+1} closer to the root (see Fig. 6.36) may be found by means of the Newton–Raphson iteration procedure, which is based on use of the following equation

$$x_{n+1} = x_n - f(x_n) \cot \theta = x_n - f(x_n)/f'(x_n). \qquad (6\text{-}17)$$

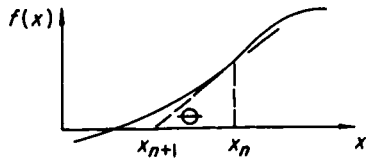


FIG. 6.36. Approximation to the root: $x_{n+1} = x_n - f(x_n)\cot\theta$

With an initial estimate of x_n, successive application of Eq. (6-17) will yield progressively better approximations to the root.

Let $f(x) = x^p - a$, then $f'(x) = px^{p-1}$ and

$$x_{n+1} = x_n - \left(\frac{x^p - a}{px^{p-1}}\right)_{x=x_n} = \left(\frac{px^p - x^p + a}{px^{p-1}}\right)_{x=x_n} = \left(\frac{a + x^p(p-1)}{px^{p-1}}\right)_{x=x_n} \tag{6-18}$$

Equation (6-18) is used in the sections following as the basis for the derivation of specific iterative equations to determine the value of reciprocals, square roots, and higher order roots.

6.2.2. COMPUTATION OF THE RECIPROCAL

If in the expression $f(x) = x^p - a$ (see Section 6.2.1), one substitutes $p = -1$, the result is

$$f(x) = (1/x) - a. \tag{6-19}$$

The root of this equation is $1/a$. Therefore, if $p = -1$ is substituted in Eq. (6-18), the following commonly used iterative equation for the reciprocal of a number, a, is obtained

$$x_{n+1} = \frac{a + x_n^{-1}(-2)}{-x_n^{-2}} = -x_n(ax_n - 2) = x_n(2 - ax_n). \tag{6-20}$$

The normalized difference $(x_{n+2} - x_{n+1})/x_{n+1} = (1 - ax_{n+1})$ is equal to $(1 - ax_n)^2$, the square of this difference at the preceding step. If the initial estimate of the reciprocal, i.e., x_0, is good to a precision of 2^{-5}, three iterations will suffice to give a final result good to 2^{-40}. Since there are two multiplications per iteration, a reciprocal can be obtained to the desired accuracy in six multiplication times, and a quotient in seven.

A small table of 2^4 entries can be used to provide the initial estimate of $1/a$. To avoid overflows $2^{-5}/a$ would be used, see Example 6.24.

Means for generating the reciprocal no longer has the importance within digital computers that it once had, when a built-in division operation was not common in general purpose computers.

Example 6.24

a	$2^{-5}/a$
.0001	.1
.0010	.01
.0011	.0010101
.0100	.001
.0101	.000110011
.0110	.0001010101
.0111	.0001001001
.1000	.0001
.1001	.0000111000111
.1010	.000011001100
.1011	.00001011101000101101
.1100	.0000101010
.1101	.0000100111011000
.1110	.00001001001
.1111	.0000100010001

6.2.3. Methods of Computing the Square Root

In this section a number of procedures for obtaining the square root of a number will be described. We will consider first a procedure referred to as simple iteration, based on the general iterative equation (6-18) derived in Section 6.2.1. If in the expression $f(x) = x^p - a$ (see Section 6.2.1), $p = 2$ is substituted the result is

$$f(x) = x^2 - a. \tag{6-21}$$

The roots of this equation are $x = \pm \sqrt{a}$. Therefore, if $p = 2$ is substituted in Eq. (6-18), the following iterative equation for the square root of a number a is obtained

$$x_{n+1} = \tfrac{1}{2}\left(x_n + \frac{a}{x_n}\right). \tag{6-22}$$

The initial estimate x_0 may be any plus or minus value, except zero. A good estimate of x_0, for small a's, is $a/2$. The number of iterations required is a function of the argument, increasing as the argument decreases. Note that a division operation is required for each iteration cycle. Equation (6-23) requires only one division, regardless of the number of times the equation is applied, but requires multiplications for each iteration cycle

$$x_{n+1} = \frac{3x_n}{2} - \frac{x_n^3}{2a}. \tag{6-23}$$

Equations (6-22) and (6-23) are both second order iterative equations. This means that once a moderately accurate approximation has been made, each application of the equation will double the number of signi-

ficant digits in the approximation. Equation (6-23) requires a more accurate first guess than Eq. (6-22) and the magnitude of x_0 must be $< (5a)^{1/2}$. Equations (6-24) and (6-25) are examples of third order iterative equations

$$x_{n+1} = \tfrac{1}{8}\left(3x_n + \frac{6a}{x_n} - \frac{a^2}{x_n^3}\right) \tag{6-24}$$

$$x_{n+1} = \frac{x_n}{8}\left(15 - \frac{10x^2_n}{a} + \frac{3x_n^4}{a^2}\right). \tag{6-25}$$

Another way of obtaining the square root would be to store a reasonably sized table of square roots and to use an interpolative procedure to obtain roots for which there were no entries in the table. Such a table would be cumbersome. A more desirable approach would consist of having a table with just a few entries to provide a good initial estimate, x_0, with which to start one of the iterative methods. The error at the $(n + 1)$th iteration, $(\text{error})_{n+1}$, is approximately equal to $(\tfrac{1}{2})\ ((\text{error})_n^2$ which converges rapidly for $(\text{error})_n < 1)$.

The procedure for extracting the square root to be described next is known as the odd series approximation. It makes use of the fact that the square root of the sum of a series of odd numbers: 1, 3, 5, . . . has a value that corresponds to the position of the highest term in the series. For example, the sum of 1, 3, 5, and 7 is 16 and the square root of 16, namely 4, corresponds to the position of 7 in the series 1, 3, 5, 7. The nature of the odd series relationship is shown below

Series	1	3	5	7	9	11	13	15 . . .
Sum of the series	1	4	9	16	25	36	49	64 . . .
Square root of the sum	1	2	3	4	5	6	7	8

The actual procedure used to extract a root is essentially a restoring division process utilizing artificial subtrahends based on the series of odd numbers. The procedure will be described for both the decimal and binary system.

In the decimal system, the square root may be obtained by the following procedure: (1) Separate the digits of the radicand into groups of two digits each, starting from the decimal point; (2) Diminish the first

group by 1, 3, . . . successively until a negative remainder is produced; (3) Restore the remainder to its last nonnegative value. Put down as the first digit of the root, the number of subtractions performed before the last nonnegative remainder was reached; (4) Bring down the next group of two digits, and form the new subtrahend by increasing the last subtrahend used (i.e., the one before that producing a negative remainder) by 1, shifting it one place to the right, and adding a 1 in the place to the right of it. These operations are continued until a remainder of 0 is reached, or to as many significant places as desired. The extraction of the square root of 489 is shown as Example 6.25.

Example 6.25

```
      2     2     1     1     3
√   4    89    00    00    00
    1
   ---
    3
    3
    0

          89
          41
         ---
          48
          43
         ---
           5
          45
         ---
       Restore
           5    00
           4    41
         -----------
                59
           4    43
         -----------
       Restore
                59    00
                44    21
              ---------------
                14    79
                44    23
              ---------------
              Restore
                14    79    00
                 4    42    21
              ---------------------
                10    36    79
                 4    42    23
              ---------------------
                 5    94    56
                 4    42    25
              ---------------------
                 1    52    31
                 4    42    27
              ---------------------
              Restore
```

Whenever there is a zero in the root, indicated by the need for a restoration after the first of a new series of subtrahends is subtracted, the next subtrahend is formed as follows: (a) increase the last subtrahend used by 1, (b) shift it two places to the right, (c) after it place a 0 and then a 1. Example 6.26 shows the use of this procedure for the extraction of the square root of 95,481.

Example 6.26

```
   3      0      9
√9     54     81
 1
 8
 3
 5
 5
 0
       54
       61
      Restore
       54     81
        6     01
       48     80
        6     03
       42     77
        6     05
       36     72
        6     07
       30     65
        6     09
       24     56
        6     11
       18     45
        6     13
       12     32
        6     15
        6     17
        6     17
```

In the binary system, the procedure is as follows: (1) Separate the bits of the radicand into groups of two bits each, starting from the binary point. (2) Begin the actual extraction operation at the first group of bits from the left that does not contain two zeros. Align a 1 with the right-hand bit of this group and subtract. The remainder will be nonnegative

and a 1 is entered in the root for this group. For each double 0 group to the left of this group, a 0 is entered in the root. (3) For all succeeding groups, the trial factor to be subtracted from the remainder is the expression $(4r_{n-1} + 1)$, (if fractional arguments and r_{n-1}, the approximate root already obtained, are treated as if they were integers). The right hand digit of the trial divisor is aligned with the right hand digit of the group for which it is used, and subtracted. If the remainder is nonnegative, a 1 is entered as the root for that group. If the remainder is negative, the root is 0 and the subtraction is restored. See Example 6.27 in which the square root of .10101001 is extracted.

Example 6.27

```
   .1      1      0      1
√.10     10     10     01
  1
  1      10
  1      01  = (4 × 1) + 1
         1      10
        11      01  = (4 × 3) + 1
        Restore
         1      10     01
         1      10     01  = (4 × 6) + 1
```

Check: .10101001 = 169/256
.1101 = 13/16

6.2.4. Computation of Higher Order Roots

To obtain the cube root of a positive number a, one can use any of a number of third order equations and corresponding iteration functions based on the Newton–Raphson method

Equation	Corresponding iteration function
$x^3 - a = 0$	$g_{31}(x)$: $x_{n+1} = 1/3(a/x_n^2 + 2x)$
$x^{3/2} - a^{1/2} = 0$	$g_{32}(x)$: $x_{n+1} = 1/3(2a^{1/2}/x_n^{1/2} + x)$
$x^{3/4} - a^{1/4} = 0$	$g_{33}(x)$: $x_{n+1} = 1/3(4a^{1/4}/x_n^{1/4} - x)$.

Since the function $g_{31}(x)$ does not involve any square root operation, it is more convenient to use in a machine without a built-in square root instruction. However, both $g_{32}(x)$ and $g_{33}(x)$ usually converge in fewer steps than $g_{31}(x)$.

Iterative equations for the fifth, seventh, and ninth roots are listed next.

For the fifth root

$$x^{5/4} - a^{1/4} = 0 \qquad g_5(x)\colon\ x_{n+1} = [(4a^{1/4}/x^{1/4}) + x] \div 5$$

For the seventh root

$$g_7(x)\colon\ x_{n+1} = [8(ax)^{1/8} - x] \div 7^*$$

For the ninth root

$$g_9(x)\colon\ x_{n+1} = [8(a/x)^{1/8} + x] \div 9^*$$

6.2.5. Generation of Trigonometric Functions

Storing an extensive table of trigonometric functions makes excessive requirements on the storage facilities of a computer. On the other hand, generating these quantities from a single Taylor series expansion is not desirable because the maximum error grows inordinately with the range of the interval and, therefore, the number of computations required becomes excessive.

Some simple methods superior to the ordinary Taylor series are available. They are:

(1) The use of expansions which have essentially the same accuracy over the entire interval, e.g., Chebyshev polynomials, continued fractions, or optimal rational approximations. For example, Hastings† has given the following series which has an accuracy of 10^{-6} over the first quadrant:

$$\sin\ (\pi/2)x = 1.570795x - 0.645921x^3 + 0.079488x^5 - 0.004362x^7$$

where: $-1 \leqslant x \leqslant 1$

$$\sin^{-1}x = \pi/2 - \sqrt{1-x}\,F(x)$$

where: $F(x) = 1.570788 - 0.214125x + 0.084666x^2 - 0.035757x^3 + 0.008649x^4$

and $0 \leqslant x \leqslant 1$.

(2) The technique of dividing the interval of interest into n small subintervals and using a Taylor series expansion about the center of each interval. This requires the storage of the sine and cosine of n arguments. A value of $n = 16$ is sufficient for an accuracy of 10^{-5} and facilitates entry into the table. For example, if the angle x lies in the ith interval and v_i is in the center of the interval, then to the accuracy desired

---

* Hammer, P. C. [1955] "Iterative procedures for taking roots based on square roots," *MATC*, **9**, 68.

† Hastings, C., Jr. [1955] *Approximations for Digital Computers*, Princeton Univ. Press.

$$\sin x = \sin v_i + \cos v_i(x - v_i) - \sin v_i \frac{(x - v_i)^2}{2} - \cos v_i \frac{(x - v_i)^3}{3!}$$

$$\cos x = \cos v_i - \sin v_i(x - v_i) - \cos v_i \frac{(x - v_i)^2}{2} + \sin v_i + \frac{(x - v_i)^3}{3!}$$

where v_i refers to the stored values of the argument.

(3) The sine and cosine of any argument, x, may be expressed in the form $\sin x = \sin(\omega + \delta)$ and $\cos x = \cos(\omega + \delta)$, and each of these expressions may be expanded to yield

$$\sin(\omega + \delta) = \sin \omega \cos \delta + \cos \omega \sin \delta$$

$$\cos(\omega + \delta) = \cos \omega \cos \delta - \sin \omega \sin \delta$$

The values of $\sin \omega$, $\cos \omega$ can be obtained from a small stored table, and $\sin \delta$, $\cos \delta$ may be obtained by the following series approximations

$$\sin \delta \approx \delta - \frac{\delta^3}{3!}$$

$$\cos \delta \approx 1 - \frac{\delta^2}{2}$$

Any desired accuracy may be obtained either by increasing the table of stored values or by increasing the number of terms in the expressions for $\sin \delta$, $\cos \delta$. For a table of 16 values of the sine and the cosine, and using the two term expansions for $\sin \delta$, $\cos \delta$, the values of $\sin x$, $\cos x$ may be obtained with a precision of 10^{-5}.

We will consider next some procedures for generating inverse trigonometric functions. The Taylor series expansion for an inverse trigonometric function, e.g., $\sin^{-1}x = x + x^3/3! + 3x^5/40 + \ldots$, suffers from poor accuracy, especially near $x = 1$. An interpolative scheme that is more desirable, and which can be used to obtain both the inverse sine and cosine is based on use of the following type of equation

$$\sin^{-1}(x + \epsilon) = \sin^{-1}x + \frac{\epsilon}{\sqrt{1 - x^2}} + \frac{x\epsilon^2}{2(\sqrt{1 - x^2})^3} + \ldots$$

If this equation were utilized, values of $\sin^{-1}x$ and $\sqrt{1 - x^2}$* could be stored. Although a higher order expansion is required for the inverse trigonometric functions than for the trigonometric functions, the method still is useful because only a few simple operations have to be performed

---

* Any subroutine involving the generation of the square root by an iterative method without a good first approximation provided by a small stored table consumes excessive time. Since the square root operation in this expression involves x (as opposed to $(x + \epsilon)$) no interpolation or iteration is required.

on stored values, thereby reducing round-off errors and the time required for computation.

Another scheme involves use of an interpolation formula of the form

$$\sin^{-1}(x_v + \epsilon) = \sin^{-1}x_v + a_1\sin^{-1}x_{v-1} + a_2\sin^{-1}x_{v-2} + \ldots + b_1\sin^{-1}x_{v+1} + b_2\sin^{-1}x_{v+2}.$$

In this equation a_1, b_1, and b_2 are all dependent on ϵ. In this procedure, a set of values of $\sin^{-1}x_v$ only must be stored. The inaccuracy of a small table of entries is compensated for by using several entries on either side of x_v.

Some of the methods for computing trigonometric and other transcendental functions in a general purpose type of digital computer are summarized as: (1) Computation of the function from its series expansion. (2) Generation of the function by means of a polynomial expression that adequately approximates the desired function to a required accuracy over the range required. (3) Use of a relatively small table of stored values and derivation of intermediate values by means of interpolation formulas. (4) Numerical integration of a set of difference equations whose solution represents the desired functions. It is not always readily apparent which method is the most desirable. Among the factors that must be considered are available storage space, storage access time, the time required for each operation in the program, and the relative difficulty in preparing different programs.

6.3. Scaling of Problems

6.3.1. Scaling for Fixed-Point Computation

While no binary point actually exists in a computer, one can imagine it to be between any two successive bits in a numerical representation. The position of the point is determined by the choice of scale factor. The determination of scale factors in a problem is referred to as scaling and involves the following: (1) Defining the position of the radix point in each of the input numbers based on the bounds of their magnitude. (2) Following the behavior of the point in all of the computational steps. (3) Knowing the position of the point in each of the final results. For proper scaling, complete and accurate information about the bounds on the magnitude of all numbers entering the computation should be available. This is necessary to simultaneously provide for : (1) Efficiency of scaling, i.e., minimization of leading zeros and, consequently, increased accuracy. (2) Accommodation of the largest numbers without overflow of any register. For convenience, the symbols for the numbers in a prob-

lem may be written in the following form which explicitly states the position of the radix point

$$x = r^b \bar{x}. \tag{6-26}$$

In Eq. (6-26), x is the true value, $\bar{x}$ termed the "scaled form" of x is an n place fraction, and in the scale factor, r^b, b is the smallest integral power of r which makes r^b greater than the maximum value of x.

A standard convention in fixed-point computation is to consider fixed-point numbers in a computer as fractions and to express the numbers in a problem as fractions multiplied by scale factors. Neither addition nor subtraction changes the position of the radix point. However, special attention must be given to the position of the radix point in a product or quotient. Usually, the radix point remains at the extreme left in both multiplication and division of fractions. For example, in multiplication, two n-bit fractions yield a product which is a $2n$-bit fraction, and in division, a $2n$-bit fractional dividend divided by an n-bit fractional divisor yields an n-bit fractional quotient (generally the dividend is an n-bit fraction and the $2n$-bit accumulator is used to shift the dividend the proper number of places to the right required to make it less than the divisor).

The steps to follow in scaling a problem are: (1) Ascertain the bounds on the absolute values of the numbers. (2) Set up the scaling relationship between true numbers and scaled fractions by determining the required scale factor. (3) By substitution, obtain from the true value formula the scaled value formula, and write the program directly from the latter. The scale factors which do not cancel specify the required machine shift operations.

To prevent an overflow in a summing process within an accumulator, it is not enough to scale the final sum according to its bound, for, in general, it must be scaled by the largest bound which applies to any element in the sum or partial sum, P_i, generated in the process of summing. Consider the sum

$$A = \sum_{i=1}^{n} a_i$$

with the following bounds given

$$\begin{aligned} |A| &< J \\ |a_i| &< K_i \qquad i = 1, 2, \ldots n \\ |P_i| &< L_i \qquad i = 1, 2, \ldots (n-2). \end{aligned}$$

The largest bound would be selected from J, K_i, and L_i to use as the effective bound for scaling both the sum A and the elements a_i. Where the partial sums are not known, the bound used for selecting the scale factor is $n\ |a_k|$, where n is the number of elements and a_k is the element with the greatest magnitude.

In setting up the order of computation steps, the programmer should attempt to determine the bounds on intermediate quantities in the computation. Often a smaller bound than the implied maximum bound may be used. For example, if $|A| < 12$, $|B| < 20$, the implied maximum bound of $|AB|$ would be 240. However, there may be other constraints on the system such that $|AB| < k$, where $k < 240$. When there are alternatives in the order of computation steps, that order should be chosen which makes use of known effective bounds to replace implied maximum bounds. The scaling of any problem is not necessarily unique, and several good approaches may be available.

6.3.2. Floating-Point Notation for Numbers

In this notation each number is expressed in the form aR^b. Thus, for any number system of radix R, where R may be any integer greater than unity, it is only necessary to specify the numbers a, b. The radix point is usually considered to be to the left of the highest order nonzero bit or digit. The coefficient or fractional part of the number, namely a, is constrained within specified limits by adjustment of the integral exponent b. The value of a is within the bounds $1/R \leqslant a < 1$, or else it is zero. In the binary system, $1/2 \leqslant a < 1$, or $a = 0$. In the decimal system $0.1 \leqslant a < 1$, or $a = 0$. When restricted to such an arbitrary range, the coefficient a is referred to as the mantissa, and the integer b as the exponent index of the floating binary or decimal number. The exponent may be zero or any integer, except that in any machine its magnitude must be bounded because of storage limitations.

The use of floating-point numbers affords a convenient method of computing with numbers which vary in magnitude over a relatively wide range. Since, after a floating-point operation, a specified number of significant digits is retained, and the magnitude of the number indicated by an exponent index, assurance is provided that all numbers (i.e., the mantissas) are fixed in magnitude within some predetermined scale. Some examples follow to show how numbers in floating-point form are manipulated. Consider first, multiplication and division using numbers in floating decimal form

$$(a_1 10^{b1})\ (a_2 10^{b2}) = (a_1 a_2) 10^{b1 + b2}$$

$$(a_1 10^{b1})/(a_2 10^{b2}) = (a_1/a_2) 10^{b1 - b2}.$$

If the values of a_1a_2 and a_1/a_2 do not fall within the range $0.1 \leqslant a_1a_2 < 1$, an additional step is required, namely the mantissas of the results, a_1a_2 and a_1/a_2, must be scaled to fall within the specified range, and the exponent index of the result adjusted to compensate for the scaling operation. For example

Operation	Unscaled result	Adjusted result
$(0.451 \times 10^{3})\ (0.207 \times 10^{-6})$	0.0934×10^{-3}	0.934×10^{-4}
$(0.905 \times 10^{4})/(0.231 \times 10^{14})$	3.92×10^{-10}	0.392×10^{-9}

The algebraic addition of numbers in floating-point form presents more of a problem since the exponent indices of the two numbers must be made equal before addition of the mantissas can take place. Therefore, the radix points of the two numbers must be aligned in some predetermined scale. This scale may be defined by the exponent index of the larger absolute number. Accordingly, the mantissa of the smaller absolute number is adjusted until its associated exponent index is equal to the exponent index of the larger. This adjustment is effected by dividing the mantissa of the smaller absolute number by ten raised to a power equal to the absolute value of the difference of the exponent indices. For example

Operation	Unscaled result	Adjusted result
$(0.324 \times 10^{-8}) + (0.984 \times 10^{-7})$	0.0324×10^{-7} 0.984×10^{-7} 1.016×10^{-7}	0.102×10^{-6}

6.3.3. Representation of Floating-Point Numbers Within a Computer

When using numbers in floating-point form within a computer, it is convenient to represent the mantissa and exponent index as a single number. However, a difficulty presents itself in that there is a sign associated with each, and these signs may not be the same. However, if the range of the index is arbitrarily limited, the sign of the index can always be made positive. For example, if it is desired to limit the range of the exponent to $-k \leqslant b < k$ (where, e.g., $k = 50$), the actual value of the exponent used would be $(b + k)$ since $0 \leqslant (b + k) < 100$, i.e., $(b + k)$ is always positive in sign. This bias of the exponent is readily removed in the interpretation of results. Figure 6.37 shows how a floating decimal number might be represented in a computer in which each storage location accommodates a sign plus ten digits. As an example, the floating decimal number $+0.54870623 \times 10^{-7}$ would appear as $+0.5487062343$. The exponent is 43, since $(b + k) = (-7 + 50)$.

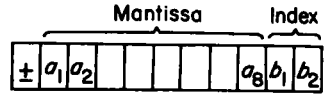


FIG. 6.37. Format for representing a number in floating point form

It is seen, then, that the use of floating point notation allows very large or very small numbers to be stored by means of a relatively small number of bits. In fixed-point notation, one has the alternatives of retaining all the zeros between the significant bits and the radix point or keeping track of the point throughout a lengthy computation.

6.3.4. A COMPARISON OF FIXED-AND FLOATING-POINT OPERATION IN A COMPUTER

Let us consider first some of the features of fixed-point operation. To code a problem with fixed-point numbers, the coder must know in advance the relative magnitude of the results of all arithmetic operations. This is necessary to insure that all numbers stay within range, and that a sufficient number of significant digits is retained. If it is anticipated that the result of some operation will produce an overflow, the coder will provide for shifting the operand(s) producing the result to the right before the operation, or after the operation if there are provisions for retaining the overflow digit. A left shift is called for if it is anticipated that results will not contain enough significant digits. The number of shifts necessary must be determined by estimating the magnitude of each result. The procedure of estimating the magnitude of operation results and including right and left shift commands to keep numbers in scale is part of the scaling problem.

From the preceding description it is apparent that the coder makes partial use of floating-point operations in order that results remain within the scale of the computer. If the analysis of the number and magnitude of shifts necessary is in error, and results go out of scale, it may be necessary to recode all or part of the problem. There are other disadvantages to fixed-point operation: (1) In preparing a general problem for machine solution, the coder attempts to account for the maximum and minimum results possible for all cases. Therefore, in some cases, significant digits may be lost. (2) In many problems it is not possible to predetermine the magnitude of the result of all computations. Here, the best course is trial and error running of the problem on the computer.

If floating-point notation is used for all numbers involved in arithmetic operations, the computer can be programmed to automatically handle the

bookkeeping job of normalizing (scaling) the mantissas and recording the exponent indices.

The principal advantages of floating-point operation are as follows: (1) A wider range of numbers may be handled. (2) Coding a problem in floating-point form is simpler, for no special attention must be given to scale factors during the course of the computation. This significantly reduces time needed for coding and checking, and is an important consideration for production computing, where time and energy required for scaling in fixed-point coding can be appreciable. (3) It allows the programming of specific problems for which the relative magnitudes of operation results are unknown, as well as the programming of general solutions to standard problems without consideration of individual cases. (4) It may be used in conjunction with fixed-point coding to determine the relative magnitude of operation results, thereby obtaining more accurate scaling for the fixed-point coding.

There are, also, certain disadvantages to floating-point operation, namely: (1) Fewer significant places are available since digit places that might otherwise be used to represent digits in a number must be reserved to represent the exponent index of each number. (2) The results of arithmetic operations may appear to have more significance than actually exists. This danger of falsely interpreting computed results arises because the bookkeeping associated with scaling is performed automatically within the machine, whereas in fixed-point operation the coder keeps track of the scaling involved, and therefore, knows how many significant digits are in the results of operations. For example, consider the following operation in floating-point notation

$$
\begin{aligned}
& (.2536475860^{*} - .2536475460) \times .5400022275 \\
= \; & .0000000460 \times .5400022275 \\
= \; & .4000000052 \times .5400022275 \\
= \; & .2160008977
\end{aligned}
$$

Actually the result does not have eight significant digits as the answer would indicate but, as a result of the subtraction, at most one. A loss of significance like this may be carried into other operations and amplified. For most engineering problems, the objection of unknown significance is not critical since intuitive and mathematical checks may be used to judge the correctness of results. However, it is also true that, because of incorrect scaling or the difficulty of proper scaling, fixed-point opera-

* The reader is reminded that in order to provide for negative exponents the indicated value of the index is biased by + 50, so that the actual index is + 10, not 60.

tions may result in a loss of significant digits as great or greater than that caused by floating-point operations. (3) A floating-point system introduces certain complexities. Multiplication and division are more difficult because, in addition to the normal operations on the significant bits, additive operations must be performed on the exponents. Also, a test must be made on the products and quotients for zeros to the left of the most significant bits and appropriate shifts and adjustments of the exponents performed, or else a gradual loss of significant bits may be introduced. Additive operations too are more difficult, because of the necessity of shifting to match exponents before an operation can take place. Overflows become more frequent, requiring a shift operation and a corresponding adjustment in the exponent. (4) Programmed floating-point operation consumes more time, and built in floating-point operation requires more equipment than fixed-point operation.

6.4. Binary, Decimal Conversion

6.4.1. Decimal to Binary Conversion

First, two methods of conversion will be illustrated using decimal notation. In the first method, a decimal integer is converted by repeated division by 2. Each time a remainder occurs, i.e., whenever the number being divided is odd, a 1 is entered in the appropriate order of the binary number being formed. A decimal fraction is converted by repeated multiplication by 2, any carry beyond the decimal point being discarded. Each time there is a carry into the units' order, a 1 is entered in the appropriate order of the binary number being formed. The process is continued until a desired number of significant places has been attained. Two examples of this method are shown in Example 6.28.

Example 6.28(a)

Integer: 219

219	
109	1
54	11
27	011
13	1011
6	11011
3	011011
1	1011011
0	11011011

Converted number: 11011011

Example 6.28(b)

Fraction: 0.8125

0.8125	
1.6250	0.1
1.2500	0.11
0.5000	0.110
1.0000	0.1101

Converted number: 0.1101

In the second method, powers of 2, in decimal notation, are subtracted from the decimal number in sequence, starting with the largest power of 2 equal to or less than the given number. Each power of 2 which would produce a negative difference is not subtracted. The corresponding bit in the binary number is 1 if there is a subtraction, otherwise 0. This process is continued until a remainder of 0, in the case of integers, or a desired number of significant places, in the case of fractions, is obtained. The binary number consists of 1's placed in the positions representing the powers of 2 contained in the number, and 0's in the remaining places. See Example 6.29.

Example 6.29(a)

Integer: 219

	219	
2^7	128	1
	91	
2^6	64	11
	27	
2^4	16	1101
	11	
2^3	8	11011
	3	
2^1	2	1101101
	1	
2^0	1	11011011
	0	

Converted number: 11011011

Example 6.29(b)

Fraction: 0.8125

		0.8125
2^{-1}	0.5	
		0.3125
2^{-2}	0.25	
		0.0625
2^{-4}	0.0625	
		0.0000

Converted number: 0.1101

This method is not attractive because powers of 2 in the decimal system are awkward to handle, and it is not simple to mechanize the determination of which powers of 2 should be subtracted.

The next two methods of conversion will be described using binary notation. In the first method, the digits in the decimal number are examined one at a time, starting with the highest order if the number is an integer. The binary equivalent of the highest order digit is recorded in the lowest four binary orders to the left of the radix point. This amount is then multiplied by 1010 (decimal 10) and the binary equivalent of the next decimal digit is added to the product. This process is repeated for each digit in the decimal number. For fractions, the digits are handled in opposite sequence, and the intermediate results are divided by 1010. See Example 6.30.

Example 6.30(a)

Integer: 219

Add	0010	(2):	0010
Multiply by	1010.	:	10100
Add	0001	(1):	10101
Multiply by	1010.	:	11010010
Add	1001	(9):	11011011

Example 6.30(b)

Fraction: 0.8125

Add	0.100000000000	(.5):	.100000000000
Divide by	1010.	:	.000011001101
Add	0.001100110011	(.2):	.01
Divide by	1010.	:	.000001100110
Add	0.000110011010	(.1):	.001
Divide by	1010.	:	.000000110011
Add	0.110011001101	(.8):	.1101

In Example 6.31 use is made of the binary equivalent of each decimal order (1, 10, 100, . . . in the case of integers and .1, .01, .001, . . . for fractions). The binary number is formed by accumulating each of these quantities as many times as specified by the value of the digit in each order. Neither multiplication nor division is required but there can be a large number of addition operations.

Example 6.31(a)

Integer: 219

Add	1100100 (dec 100)	twice :	11001000
Add	1010 (dec 10)	once :	1010
Add	1 (dec 1)	nine times:	1001

Example 6.31(b)

Fraction: 0.8125

Add	0.1	eight times:	0.110011001101
Add	0.01	once :	0.000000101001
Add	0.001	twice :	0.000000001000
Add	0.0001	five times :	0.000000000010
			0.110100000000

We will now reconsider the schemes of decimal to binary conversion shown in Examples 6.28(b) and 6.30(b) in terms of binary-coded decimal notation. We recall that in Example 6.28(b) the binary-coded decimal is

multiplied successively by 2, but if there is a carry into the unit's order, it is not included in the next multiplication. The successive coefficients that appear in the unit's order, i.e., to the left of the radix point, will be the bits of the converted number. Why this is so can be seen by considering the following equations

$$\text{If} \quad (a_1 \times 10^{-1} + a_2 \times 10^{-2} + a_3 \times 10^{-3} + \ldots)$$

$$= (b_1 \times 2^{-1} + b_2 \times 2^{-2} + b_3 \times 2^{-3} + \ldots)$$

$$\text{Then} \quad 2(a_1 \times 10^{-1} + a_2 \times 10^{-2} + a_3 \times 10^{-3} + \ldots)$$

$$= (b_1 \times 2^0) + R_1$$

$$\text{and} \quad 2 \times 2(a_1 \times 10^{-1} + a_2 \times 10^{-2} + a_3 \times 10^{-3} + \ldots) - b_1$$

$$= b_1 + (b_2 \times 2^0) + R_2$$

etc.

In these equations, the R_i refer to the fractional part of the binary-coded number after each multiplication. The conversion of (.1000) (0001) (0010) (0101) = .8125 to binary form is shown in Example 6.32.

Example 6.32

	.1000	0001	0010	0101	= $.a_1a_2a_3a_4$
$b_1 = 1$	0110	0010	0101	0000	
$b_2 = 1$	0010	0101	0000	0000	
$b_3 = 0$	0101	0000	0000	0000	
$b_4 = 1$	0000	0000	0000	0000	

Check: $.b_1b_2b_3b_4 = .1101 = 13/16 = .8125$

In a modification of this method, the digit right of the radix point is tested after each multiplication by 2 to determine if it is equal to or greater than 5. If it is, a 1 is placed in the binary number being formed, otherwise a 0. The two methods are equivalent since if the digit immediately to the right of the radix point is 5 or greater, a carry will be produced after the next multiplication by 2. The first procedure, that of listing the carries, is preferable since it requires no comparison operation.

The scheme of Example 6.30(b), using binary-coded decimal notation, is shown in Example 6.33. Each digit, starting with the least significant one, is added to the four most significant orders of a fractional accumulator, the result is divided by ten and the process repeated. After the most significant digit has been added, the result is divided by 10/16. Examination of this method shows that after each division by ten, the

four most significant binary orders must all have the value 0, so the new digit can simply be entered there without the need for an addition operation.

Example 6.33

Fraction: .(1000)(0001)(0010)(0101) = .8125

Add	0.0101	(5/16):	0.0101
Divide by	1010.	:	0.00001000
Add	0.0010	(2/16):	0.00101000
Divide by	1010.	:	0.00000100
Add	0.0001	(1/16):	0.00010100
Divide by	1010.	:	0.00000010
Add	0.1000	(8/16):	0.10000010
Divide by	0.1010	:	0.11010000

Check: {[((5/16) ÷ 10 + 2/16) ÷ 10 + 1/16] ÷ 10 + 8/16}
÷ 10/16 = 8/10 + 1/100 + 2/1000 + 5/10000 = .8125 = 13/16 = .1101

6.4.2. Binary to Decimal Conversion

The four methods of conversion to be described here are counterparts of the schemes in Section 6.4.1. In the first method, conversion of an integer (Example 6.34(a)) is begun by dividing it by ten. The quotient is divided by ten and this process is repeated until a zero quotient appears. The remainder after each division represents a digit of the decimal number. Conversion of a fraction (Example 6.34(b)) is begun by multiplying it by ten. The fractional part of the product is multiplied by ten, and this

Example 6.34(a)

Integer: 11011011 = 219

		Quotient	Remainder
Divide integer by	1010.:	10101	(1001)
Divide quotient by	1010.:	10	(0001)
Divide quotient by	1010.:	0	(0010)

Example 6.34(b)

Fraction: .1101 = .8125

Multiply fraction by	1010.:	(1000).001
Multiply fraction by	1010.:	(0001).01
Multiply fraction by	1010.:	(0010).1
Multiply fraction by	1010.:	(0101).0

process is continued until the fractional part of a product is zero. The removed integral values represent the digits of the decimal number. Since $10x = 2^3x + 2x$, multiplication can be avoided, the latter two terms being formed by shifting x right three places and one place, respectively. Also, if there is no provision for accepting digits to the left of the radix point, 10/16 may be used as a multiplier instead of ten, and the bits of the product starting with the 2^{-5} order are treated as the fractional part in Example 6.34(b). The four bits right of the radix point after each multiplication by 10/16 designate the binary coded digits.

The remaining three methods will be outlined briefly. In the counterpart of the method used in Example 6.29, binary equivalents of powers of ten are subtracted until there is a zero remainder. The same power of ten may have to be subtracted several times, depending on the value of the digit. In the next method, comparable to that used in Example 6.30(a), each digit of the binary number is examined, starting with the highest order. Its value (1 or 0) is added to an accumulator and the sum is doubled. This process is continued until the value of the least significant bit is added. In the method akin to that of Example 6.31, each bit position is examined and wherever there is a 1 the power of two it represents is accumulated. In Examples 6.29 and 6.31 the process for integers and fractions is the same, while in Examples 6.28 and 6.30 multiplication and division are interchanged. The same type of relationship applies in the schemes just described.

6.4.3. A Comparison of Binary and Binary-Coded Decimal Representation

The principal advantages of a binary-coded decimal compared to a binary representation of numbers for a computing system are: (1) Input–output equipment that accepts and generates binary-coded decimal numbers simplifies the task of the user in preparing input data and interpreting output data. (2) The interpretation of numbers within the computer is facilitated, thereby aiding in the detection of faults within the computer.

In pure binary representation, four bits can represent 16 different binary numbers. In a four bit binary-coded decimal system, six of these are not used. The relative efficiency of the binary-coded decimal number system depends on the range of numbers to be accommodated. For registers of practical length (say from 20 to 40 bits), numbers expressed in binary-coded decimal form require about 18% to 25% more bistable elements than binary numbers. However, the six unused numbers of a four bit binary-coded decimal group can be put to use in a redundancy

error checking scheme (see Chapter 9), since the presence of any of them indicates that an error has occurred.

When binary-coded decimal input and/or output devices are connected to a binary computer, so called input-output conversion programs, placed in the main store, are used to convert from one representation to the other.

LITERATURE

Ashenhurst, R. L. and Metropolis, N. [1959] Unnormalized floating point arithmetic, *J. ACM*, **6**, 415-428.

Ashenhurst, R. L. [1962] The MANIAC III arithmetic system, *Proc. AFIPS Spring Joint Computer Conference*, 195-202.

Avizienis, A., [1960] A study of redundant number representations for parallel digital computers, *University of Illinois Digital Computer Lab., Rept. No. 101.*

Avizienis, A., [1961] Signed digit number representations for fast parallel arithmetic, *IRE Trans. El. Comp.*, **10**, 389-400.

Booth, A. D. [1951] A signed binary multiplication technique, *Quart. J. Mech. Appl. Math.*, **4**, Pt. 2, 236-240.

Brigham, R. C. [1961] Some properties of binary counters with feedback, *IRE Trans. El. Comp.*, **10**, 699-701.

Carroll, W. N. [1960] High-speed counter requiring no carry propagation, *IBM J. Research and Develop.*, **4**, 423-425.

Chaplin, G. B. B., Hayes, R. E., Owens, A. R. [1955] A transistor digital fast multiplier with magnetostrictive storage, *Proc. Inst. Elec. Engrs., (London)* **102**, Pt. B, 412-425.

Couleur, J. F. [1958] BIDEC a binary-to-decimal or decimal-to-binary converter, *I.R.E. Trans. El. Comp.*, **EC-7**, 313-316.

Curtis, P. C., Jr. and Frank, W. L. [1959] An algorithm for the determination of the polynomial of best minimax approximation to a function defined on a finite point set, *J. ACM*, **6**, 395-404.

Estrin, G., Gilchrist, B., Pomerene, J. H., [1956] A note on high-speed multiplication, *IRE Trans. El. Comp.*, **5**, 140.

Fraenkel, A. S. [1961] The use of index calculus and Mersenne primes for the design of a high-speed digital multiplier, *J. ACM*, **8**, 87-96.

Frank, M. E. and Schy, S. T. [1961] Counting on a magnetic drum, *Control Engrg.*, **8**, No. 10, 75-79.

Freiman, C. V. [1961] Statistical analysis of certain binary division algorithms, *Proc. IRE*, **49**, 91-103.

Garner, H. L. [1959] A ring model for the study of multiplication for complement codes, *I.R.E. Trans. El. Comp.*, **EC-8**, 25-30.

Garner, H. L. [1959] The residue number system, *Proc. Western Joint Computer Conference*, 146-153.

Hendrickson, H. C. [1960] Fast high-accuracy binary parallel addition, *IRE Trans. El. Comp*, **9**, 465-469.

Jackson, R. C., Rhodes, W. H., Jr., Winger, W. D., Brenza, J. G., [1960] A built-in table lookup arithmetic unit, *Proc. Western Joint Computer Conference*, 239-250.

Kogbetliantz, E. G. [1959] Computation of sin N, cos N, and $\sqrt[m]{N}$ using an electronic computer, *IBM J. Research and Develop.*, **3**, 147-152.

Koons, F. and Lubkin, S. [1949] Conversion of numbers from decimal to binary form in the EDVAC, *MTAC*, **3**, 427-431.

Kilburn, T., Edwards, D. B. G., and Aspinall, D. [1959] Parallel addition in digital computers, a new fast carry circuit, *Proc. IEE*, **106**, Pt. B, 464-466.

Lehman, M. [1958] Short cut multiplication and division in automatic binary digital computers, *Proc. IEE*, **105**, Pt. B, 496-504.

Lehman, M. and Burla, N. [1961] Skip techniques for high-speed carry-propagation in binary arithmetic units, *IRE Trans. El. Comp*, **10**, 691-698.

MacSorly, O. L. [1961] High-speed arithmetic in binary computers, *Proc. IRE*, **49**, 67-91.

Metropolis, N. and Ashenhurst, R. L. [1958] Significant digit computer arithmetic, *I.R.E. Trans. El. Comp.*, **EC-7**, 265-267.

Minnick, R. C. [1957] Tchebysheff approximations for power series, *J. ACM*, **4**, 487-504.

Pope, D. A. and Stein, M. L. [1960] Multiple precision arithmetic, *Comm. ACM*, **3**, 652-654.

Rabinowitz, P. [1961] Multiple-precision division, *Comm. ACM*, **4**, p. 98.

Reitwiesner, G. W. [1960] The determination of carry propagation length for binary addition, *IRE Trans. El. Comp*, **9**, 35-38; correction, 261.

Robertson, J. E. [1958] A new class of digital division methods, *I.R.E. Trans. El. Comp.*, **EC-7**, 218-222.

Robinson, A. A. [1953] Multiplication in the Manchester University high speed digital computer, *Electron. Eng.*, **25**, 6-10.

Saltman, R. G. [1961] Reducing computing time for synchronous binary division, *IRE Trans. El. Comp*, **10**, 169-174; corrections, 461.

Shaw, R. F. [1950] Arithmetic operations in a binary computer, *Rev. Sci. Instr.*, **21**, 687-693.

Sklansky, J. [1960] An evaluation of several two-summand binary adders, *IRE Trans. El. Comp*, **9**, 213-226.

Sklansky, J. [1960] Conditional-sum addition logic, *IRE Trans. El. Comp*, **9**, 226-231.

Smith, J. L. and Weinberger, A. [1958] Shortcut multiplication for binary digital computers, *NBS Circular 591*, Section 1, U.S. Dept. of Commerce.

Spitzbart, A. and Shell, D. L. [1958] A Chebycheff filling criterion, *J. ACM*, **5**, 22-31.

Tocher, K. D. [1958] Techniques of multiplication and division for automatic binary computers, *Quart. J. Mech. Appl. Math*, **11**, 364-384.

Traub, J. F. [1961] Comparison of iterative methods for the calculation of nth roots, *Comm. ACM*, **4**, 143-145.

Traub, J. F. [1961] On a class of iteration formulas and some historical notes, *Comm. ACM*, **4**, 276-278.

Wadey, W. G. [1960] Floating-point arithmetics, *J. ACM*, **7**, 129-139.

Weinberger, A. and Smith, J. L. [1958] A logic for high speed addition, *NBS Circular 591*, Section 1, U.S. Dept. of Commerce.

Williams, F. C., Robinson, A. A., Kilburn, T. [1952] Universal high speed digital computers: serial computing circuits, *Proc. Inst. Elec. Engrs. (London)*, **99**, Pt. II, 94-106.

Wilson, J. B. and Ledley, R. S. [1961] An algorithm for rapid binary division, *IRE Trans. El. Comp*, **10**, 662-670.

7. System Design of GP Computers

7.1. Variants in Organization and Mechanization

In Chapter 2, the basic operational requirements of a stored program digital computer were presented. However, from that point on the reader had to accept as a matter of faith that a physical system satisfying these requirements could actually be constructed. Now that the topics of Boolean algebra, switching networks, storage systems, and arithmetic units have been presented, it is appropriate to consider, in some detail, the various ways in which a digital computer can be organized and mechanized.

In the early stages of planning a computer, a number of important decisions must be made upon which the eventual design will largely depend. Four important basic considerations are: (1) number and type of instructions to be included in the machine's repertory; (2) size and type of main store; (3) format of words for representing instructions and data; and (4) nature of the control unit, and how it is affected by the choice of serial or parallel, and synchronous or asynchronous operation, the choice of arithmetic unit, the number of addresses per instruction, special features such as index registers, the type and mode of operation of input–output equipment, the utilization of microprogramming, and, finally, the inclusion of program-interrupt control features valuable for efficient operation of a computer subject to concurrent demands. These items are described in Sections 7.2–7.5, respectively. The arithmetic unit will be considered only in relation to its influence on the control unit, since specific variants in its design, namely algorithms, logical designs, and circuitry for the basic arithmetic operations as well as ways of representing negative numbers are described in Chapter 6.

7.2. Number and Type of Instructions

In the design of a general purpose type of digital computer, one of the first decisions to be made relates to the choice of arithmetic, logical, and information transfer operations to be built into the machine, i.e., made available to the user as a single instruction. An important consideration here is the expected frequency of use of an operation in problems to be solved by the computer. This can only be estimated, since all the types of problems for which the computer will be used are not usually

known in advance. At one end of the design spectrum are machines that are relatively simple, inexpensive, slow, and difficult to program, and at the other end, machines that are complex, expensive, fast, and easily programmed. There are a number of intermediate areas, each of which is optimum for each of several specific classes of applications.

It is often preferable to generate relatively complex operations or functions, e.g., the extraction of roots, and the evaluation of trigonometric functions, by the use of subroutines instead of the additional hardware required to directly mechanize these operations. This would be the case if, after considering the speed and average frequency of use of these subroutines, it is ascertained that their use would not result in any serious impairment of computational work. Prior to the actual design of a computer, it is not accurately known what the addition of a particular instruction will cost in equipment. The addition of an apparenty simple instruction may add appreciably to the complexity and the required number of components, while an apparently complex instruction may be added at moderate cost. Why this is so may be explained by first considering the nature of an instruction. Each instruction can be considered as a directive to the computer's control unit, the effect of which is to cause a number of more or less elementary operations to be performed, e.g., obtaining an operand from storage, causing either an arithmetic or logical operation to be performed on it, transferring data from one part of the computer to another, etc. In Section 7.5.7, and following, it is shown how each instruction can be considered to be comprised of a number of commands. A machine is capable of executing a specified instruction, if all the elementary operations called for by the commands comprising the instruction are available and can be assembled in proper sequence. Thus, the cost of a new instruction depends on how many of the elementary operations needed to synthesize it are already available within the machine.

If certain instructions are established, e.g., by a prospective customer, as being necessary, the designer's chief concern is to design a computer that adequately meets the specified requirements. If the designer has freedom of choice in specifying the instructions, as well as the manner in which they are to be executed, he may, in general, produce a design that requires less equipment for mechanization. Thus economies can be effected if the instructions chosen are considered as a group. This is just one aspect of the economies that can, in general, be realized by considering a computer as a system.

If possible, it is desirable that no instructions be included which are not generally useful and yet add appreciably to physical requirements. The term "generally useful" must be defined. It is used in the sense of being

"generally useful for a specific type or group of applications." Any general purpose type of device suffers from the fact that compromises have been made in its design in order to be able to classify it as a general purpose type of device. The designer is continually confronted with a "Battle of the Bulge," and must grapple with all sides of the situation to keep things under control. The problem is a complex one, subject to several restraints: The number of instructions must not be allowed to grow too large or else the equipment will be cumbersome; it must not be too small or else the utility and speed of the computer are adversely affected. The design should permit as rapid a computation as possible without utilizing an excess of parallel equipment or frequencies of operation that promote unreliability. The fronts of a design battle are illustrated in Fig. 7.1. It

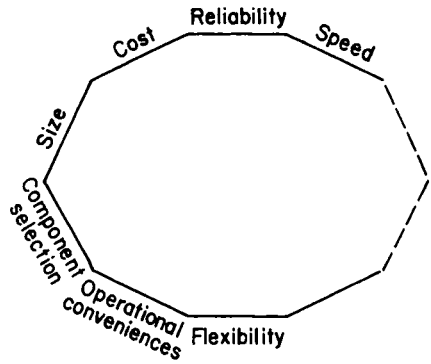


FIG. 7.1. Some representative computer design criteria

should be emphasized that the preceding remarks apply more to computational work or slow real time control systems, than to high speed control systems. In the latter case, the inclusion of special infrequently used instructions may make the difference between successful or non-successful operation.

The following list of instructions that have been used in existing computers provides a good indication of the wide variety possible, but also shows that there are only a few basic types:

Arithmetic instructions

1. Clear the accumulator and add (m) to it.*
2. Add (m) to the accumulator.

* By definition (m) means: Contents of storage location m.

3. Clear the accumulator and add absolute value of (m) to it.
4. Add absolute value of (m) to the accumulator.
5. Clear the accumulator and subtract (m) from it.
6. Subtract (m) from the accumulator.
7. Subtract absolute value of (m) from the accumulator.
8. Multiply the contents of the accumulator by (m), retaining the most significant half of the product in the accumulator.
9. Multiply the contents of the accumulator by (m), rounding off the most significant half of the product by a specified procedure.
10. Divide the contents of the accumulator by (m), forming the quotient in the multiplier–quotient register.

Logical instructions

11. Shift right by k binary positions.
12. Shift left by k binary positions.
13. Replace the address in a specified instruction in the main store with the address designated by specified bits in the accumulator.
14. Form the logical product (bit by bit) of a specified word in the main store and the contents of the accumulator.
15. Complement individual bits of the accumulator corresponding to positions in another register holding 1's.
16. Clear individual bits of the accumulator corresponding to positions in another register holding 1's.
17. Stop the machine (maintaining or recirculating all information).

Transfer of control instructions

18. Transfer control to storage location m.
19. Transfer control to storage location m if the sign of the accumulator is negative (or positive, or zero).
20. Compare the contents of the accumulator with some other specified register. According to whether the contents of the accumulator are less than, equal to, or greater than the contents of the other register continue in sequence, skip one, or skip two instructions, respectively.
21. If a breakpoint bit has the value specified, skip the instruction containing it and transfer control to a fixed location in the main store.

Sense instructions

22. Take the next instruction in sequence or from a specified address in

the main store according to whether a bistable device, Q^i, is in a set or reset state.

23. If a bistable device, Q^i, is reset, set it and continue in sequence.
24. If a bistable device, Q^i, is set, reset it and take the next instruction from a specified address in the main store.

Input–output instructions

25. Read a specified number of words, blocks, or cards from an input device directly to a specified register in the computer.
26. Write a specified number of words, or blocks on a specified output device from addresses in the main store starting at a specified point.
27. Advance (or reverse) a specified magnetic tape a specified number of blocks without altering their contents.
28. Rewind a specified magnetic tape to the beginning of a reel.
29. Print on a specified printer k lines from the main store, starting at a specified address.

If a computer has a number of auxiliary registers other than those comprising the basic arithmetic unit, a number of special instructions will be required for transferring information between each of these registers and other registers that may be specified as sources or destinations.

7.3. The Main Store

In a stored program computer, the instructions comprising a program as well as constants and intermediate data are stored in a large capacity storage system (like those described in Chapter 5) referred to as the main store. Three important decisions must be made in regard to this store, namely, the type of storage elements to be used, the total word capacity, and the manner in which access is gained to the store. The relative advantages of different types of large capacity storage systems are described in Chapter 5. In respect to the total word capacity to be employed, there is a practical upper limit to the size of the main store* because blocks of new data and/or instructions can be introduced from auxiliary storage units at adequate speeds. Since most computations will be highly repetitive in nature, the time required to complete the operations specified by one filling of the main store may be considerably greater than the time required for the filling itself. In this case, the over-all speed

* For commercial applications such as large inventory control, storage units of a different order of magnitude are required. Such machines are not so much computers as they are low access time filing units.

of the computer would be increased only slightly by use of a larger sized main store. This consideration alone would make main stores of 1,000-4,000 words adequate in most cases. However, more complex applications and accommodation of programs that automatically convert a problem-oriented language to instructions in machine code may call for 4,000 to 16,000 words and more. Selection circuitry for gaining access to the main store can take a variety of forms depending on the circuit characteristics of the storage elements and various schemes for recording in and reading the contents of the individual elements. As stated earlier (see Chapter 5) a basic characteristic of a storage system is whether there is access to all the bits of an instruction or number simultaneously, or whether there is access to each bit in sequence, i.e., whether the machine operates in a parallel mode or serially. Some large capacity storage systems are better suited for parallel operation, while others are better for serial operation. For example, acoustic delay lines are better suited for serial operation while a parallel mode is more applicable to cathode-ray tubes and magnetic cores. The bits of each word stored in a magnetic drum or disk store are usually recorded and read serially, although parallel arrangements can be used, too. The influence of the type of access on other parts of the computer will be described in Section 7.5.1.

7.4. Word Format

The term "word" denotes an assemblage of bits considered as an entity in a computer. A word may hold a coded representation of either a number or an instruction. In arriving at the number of bits to be used in a word, as usual, a number of compromises must be made. Let us first consider the items affecting the choice of word length to represent a number. If the word length is made adequate for even the largest numbers that might be used, much storage space and circuitry will be wasted when problems requiring less accurate solutions are solved. If too short a word length is chosen, there exists the possibility of using two or more words to store long numbers when necessary, but the multiple precision techniques required for operations on such numbers necessitate extra program steps, and consequently increase storage space requirements and solution time for a given problem. In a stored program computer, it is convenient, as well as economical of storage space, to represent an instruction by the same word length, or an integral multiple or submultiple of the word length, used for a number. The number of bits in an instruction depends on the following:

(1) The number of bits used to distinguish one instruction from another. Usually a binary code is used to represent the various instruc-

tions, so if there are n different built-in instructions, at least $\log_2(n + x)$ bits are required, where x is the smallest integer that makes $n + x$ an integral power of two. Sometimes additional bits may be reserved in order to accommodate codes for interpretive instructions that may be used. An instruction code wherein each instruction is represented by a binary number makes the most economical use of storage. Also, since it does not restrict the choice of code to represent a particular instruction, the use of a mnemonic code is facilitated. In some machines, however, particular bits are associated with particular classes of instructions. For example, one bit position can be used to distinguish between an addition or subtraction, another to distinguish between a recording or reading operation, etc. The effect of these two types of instruction codes on the main store and the control unit are described in Section 7.5.

(2) The size of the main store. Bits must be allotted in the instruction word format for controlling the selection of a particular word in the main store. The number of words, n, in storage is usually some integral power of two, so that the number of bits allotted for the selection of a particular address is simply $\log_2 n$.

(3) So far we have considered machines in which never more than one address is referred to in an instruction. Machines of this type are, accordingly, referred to as single-address machines. However, machines have also been built in which more than one address is referred to in a particular instruction. Additional bits must be provided to indicate these addresses, thereby increasing the number of bits required in an instruction word. The effect of multi-address instructions on the control unit is described in Section 7.5.4.

(4) The number of bits reserved for special control functions, if any, e.g., index registers (described in Section 7.5.5) or for the addresses of any other auxiliary registers.

The word lengths that have been chosen for presently operating general purpose electronic digital computers intended for scientific and engineering computations vary from 32 to 40 bits. Some studies have indicated that for an "average" scientific problem the optimum word length lies in this range.

7.5. The Control Unit

The usual mode of operation of a computer is as follows. After the machine is energized, a program for solving a particular problem is entered into the main store. The program is entered into the computer from an external storage medium such as punched cards, punched paper tape, or magnetic tape, which is moved and sensed by a suitable transport device and sensing mechanism. Once the machine has been filled and the

computer set to an active computing state, the sequence of states through which the machine progresses is determined by the stored program, and the number, type, and interconnection of its switching and storage elements. The advance from one internal state to the next is under the supervision of that part of the computer referred to as the control unit. In the case where the program to be executed requires more storage than available in the main store, large segments of the program may be entered at the appropriate time from an auxiliary store by the planting of input instructions at appropriate points in the master program.

The introductory description of the operation of an automatic stored program computer in Chapter 2 indicates the fundamental requirements of the control unit. Briefly, it must provide (1) means for inspecting the contents of each instruction word, and generating signals that cause the operations therein specified to be executed; and (2) means for causing the instructions located in the main store to be sensed in proper sequence. It is convenient, therefore, to divide the time interval for carrying out each step of the stored program into two major periods. The operations performed during the first period are said to comprise a search and acquisition cycle, and the operations performed in the second period comprise an execution cycle. To begin computation, means must be provided to locate the first instruction to be executed. If instructions are stored in sequentially numbered addresses in the main store, it is only necessary, after the machine has been switched to an active computing state, to always refer to a specific storage location, say that having the address 00 . . . 000. In machines with single-address instructions, the control unit is provided with a counter which is always set to some initial value when the machine is switched to an active computing state. Upon execution of each instruction, the control counter is advanced, usually by a single increment, so that it then specifies the address of the next instruction to be executed. An exception to this operation occurs for jump instructions, in a manner described at the end of the following paragraph.

Since, once acquired, the information in an instruction word must be used to control the execution process, it is necessary that the control provide a register for storing the contents of an instruction word. This register is referred to by various names in different machines, e.g., control register, instruction register. Actually the control register may also be considered as a group of separate registers. The ones that hold the operation code and address code are referred to as the operation and address registers, respectively. There may also be other registers for special functions. (For a jump instruction, the contents of the control counter are replaced by the contents of the address register).

If the main store is of the static type, the number in the address

register refers to a physical place only. As a matter of convenience, each address can be considered to be specified by two coordinate numbers, X and Y. For example in the coincident current magnetic core store (Section 5.3.2) the bits of a word occupy the same XY position in all planes and the planes are operated upon simultaneously. The X and Y parts of the address can each be applied directly to a many-to-one function table as shown in Fig. 7.2.

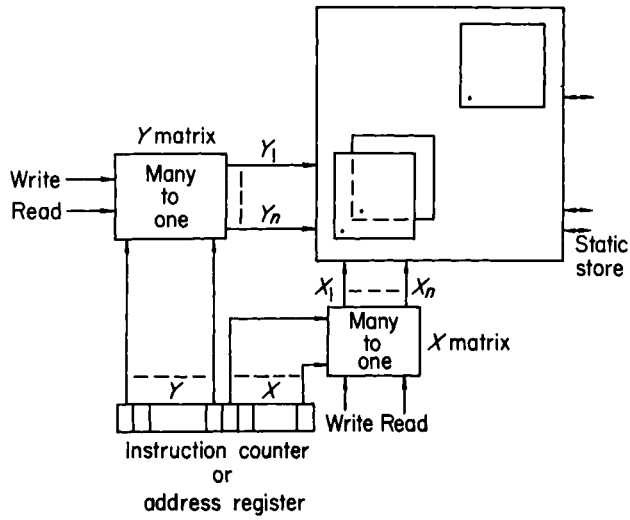


FIG. 7.2. Word selection in a static store

If the main store is of the dynamic type (see Fig. 7.3), part of the address refers to one of a number of storage areas, e.g., a particular track on a magnetic drum, or a particular acoustic tank among a set, and the other part of the address refers to the place which that storage location occupies along a particular track on the drum or in the delay line. One way of locating a particular word in a specific line or track in a dynamic store is by means of a counter whose contents are advanced by 1 after a time interval corresponding to that required to read, or record, a word. Each time the word, whose place in a line is specified by the Y part of the address, is about to emerge from the line, the number in the index counter will be identical with the bits in the time part (Y) of the address register. If the output from the latter unit and the counter are applied to a coincidence circuit, a timing signal will be produced which can be used to allow a writing or reading operation to occur at the proper time. The X output of the address register controls selection of a particular line. An input to

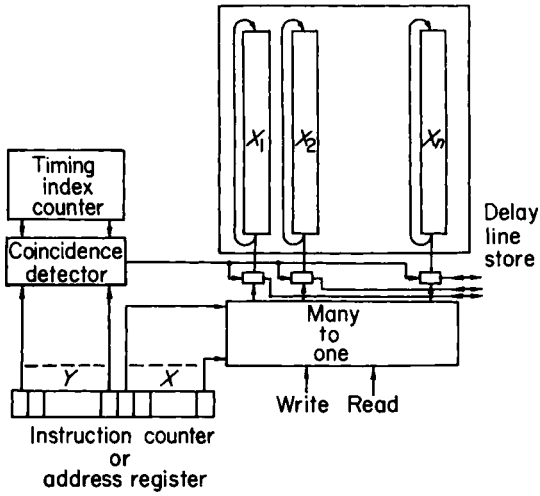


FIG. 7.3. Word selection in a delay line store

the many-to-one table from an order decoder specifies whether a writing or reading operation is to take place. Since it is necessary that the delay line input or output gates be open only during the time specified by the Y code in the address register, the output of the coincidence detector as well as the output of the many-to-one table is applied to the input-output gates.

We have already indicated that it is a usual convention to separate out groups of bits in an instruction word according to the functions to be represented or controlled by each group. For example, one group may be used to specify the address from which the operand or the next instruction is to be obtained. Another group is used to hold the operation code. If the machine has index registers, a special group of bits may be used to address the registers to be used with the instruction, while another group of bits may be reserved to hold numbers to be loaded into or added to the contents of an index register. Still other bits or groups of bits may be used for special purposes such as parity checking (see Chapter 9), breakpoint designation (see Section 7.6.3.), or for any other purpose that may be convenient and desirable for a particular machine. Though the number of bits in each group is usually determined by the number of bits required to specify any one of the total number of choices by means of a weighted binary code, this does not have to be the case. For example, a type of operation code may be used wherein particular bits are associated with particular classes of instructions. With such a system, the outputs of the operation register can be connected directly to the appropri-

ate gates in the machine, without intervening decoding and encoding function tables. The machine built at the Princeton Institute for Advanced Study has 10 bits allotted to specify an operation. Since not all combinations of these are used, more storage capacity is required to hold a program than if a binary weighted code with fewer bits were used. However, the word length was chosen on the basis of the length of numbers desired, and this was so large that more than enough bits were available for single address instructions. Whether the scheme would still be attractive if the conflict between the word length most suited to the storage of numbers and that most suited to the storage of instructions were resolved another way is debatable. It could be argued that the added expense of additional storage capacity would more than offset the degree of elimination of decoding and encoding function tables.

Regardless of the type of code used, the bits used to represent an instruction comprise the basic information from which other circuits must be activated for the execution of the indicated function. Therefore, in addition to the control counter and control register(s), the control unit must contain circuits which generate, from the information in the control register, the actual electrical signals for execution of each of the instructions. The circuits for accomplishing this will be referred to collectively as the main control circuits. These circuits are used to develop a sequence of detailed commands to control the switching of information within the arithmetic unit and between the arithmetic unit and other units of the computer. A method of control in which the sequence of commands is generated by means of a master clock followed by timing-pulse distributing circuits is referred to as synchronous or clock control. The timing of all operations is controlled or synchronized with the clock, and each operation requires an integral number of clock intervals. A method of control in which a start signal causes a certain action to be taken, and in which the successful completion of an action generates a signal to initiate the proper following action, is referred to as asynchronous or revertive control. In an asynchronous computer there is no fixed time reference, each operation being commenced as soon as the preceding one is completed. A further description of synchronous and asynchronous control is provided in Section 7.5.2.

To summarize, the control unit must produce as many different sequences of gating or switching signals as the number of different arithmetic, logical, or transfer operations the computer is required to perform, and it must be capable of assuming at least as many logical configurations (i.e., binary states) as there are different steps required for the execution of all these instructions. This sets a lower limit to the number of active storage elements required for the control unit. These

steps will be referred to as commands in discussions that follow, and it will become evident that some commands are commonly required for the execution of any instruction, and that of the remainder some are different only in a superficial way. The common steps performed during the acquisition cycle may be grouped under the heading of instruction look-up commands. An instruction counter (see Fig. 7.4) supplies the storage

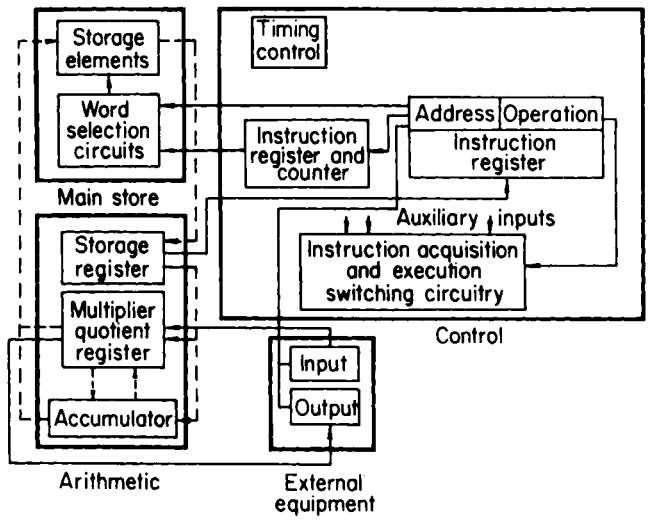


FIG. 7.4. Over-all arrangement for a single-address GP computer

selection circuits with the address of the next instruction and the main control circuits provide gating signals that cause an instruction to be read from the main store. The execution of an instruction is also performed by means of the main control circuits.

All instructions may be classified in even more general categories than those described in Section 7.2. Three basic categories are as follows. (1) Commands which control the transfer of information between parts of a system: the control of information transfers, as well as the selection of locations in the memory, depends largely on the particular type of storage used. (2) Conditional (or sequencing commands): This type of instruction makes the location from which the next instruction is obtained dependent upon whether the contents of some register, e.g., the accumulator, are less than, equal to, or greater than some number (usually taken, for convenience, to be zero). The control unit must arrange, therefore, to examine the sign and/or other bits of one or more registers, as well as cause any indicated transfers of control. (3) Commands which

control the manipulation of information—the arithmetic and logical operations called for by certain instructions. Though these commands may involve the selection of words to be read from or recorded in the main store, they are primarily concerned with the interconnections of various parts of the arithmetic unit.

The requirements for simple operations such as addition, subtraction, collation, may be very similar, differing perhaps in that part of their control which guides the operands to the proper destination in the arithmetic unit. More complicated instructions such as multiplication and division, require more complex control circuits to arrange for a number of different operations that may be required (depending on the algebraic algorithms employed), e.g., shifts of operand(s) and partial products, examination and/or comparison of bits in the operands, recording of the number of steps completed so that a signal may be provided indicating the completion of a multiplication or division, etc.

The details of a control unit's design depend on so many factors and differ so from one machine to another that it is not practical to discuss the details of several machines. Instead, the principal functions and operations common to most general purpose computer control units have been described. In the succeeding sections, the effects of five important variables on over-all machine operation will be considered. They are: serial or parallel operation, synchronous or asynchronous operation, the type of arithmetic unit, the use of single or multiple-address instructions, and the inclusion of special features such as index registers.

7.5.1. Serial or Parallel Operation

Though it is difficult to draw comparisons, because of various possible designs for both serial and parallel machines, some important distinctions can be made. First of all, because the complete addends are operated upon simultaneously in a parallel machine, rather than bit by bit as in a serial machine, its arithmetic unit must be larger: a separate register element (a flip-flop or bistable counter) as well as a separate adder circuit, must be provided for each bit (see Sections 6.1.2 and 6.1.3). The difference in equipment requirements is greater for machines with a longer word length. The control circuits can be simpler in a parallel machine; the timing is simpler since a number can be transferred from one section of the machine to another, or an addition performed, by the application of a single pulse to a set of gates. In a serial machine, a set of timing pulses corresponding to individual bit positions must be generated and applied to various gates.

The faster speed of the parallel machine is reflected in the fact that its

dominant unit of time is an addition time, i.e., the time interval required to add two numbers, while in a serial computer it is a word time, i.e., the time required to read or record a word (in both cases, the access time is not included). Certain devices may be employed to increase the speed of a serial computer. For example, the control unit may be modified so that the next instruction to be executed is selected from the main store and placed in a stand-by register (often referred to as a precommand register) concurrently with the actual execution of the current instruction. This, as well as other procedures that might be employed to increase the speed of a serial machine would have little effect on the speed of a computer with a parallel access memory and a parallel arithmetic unit. However, a higher degree of parallelism can be obtained by using several local information processors distributed throughout the area occupied by the memory, and a central processor to exercise over-all control. Such an arrangement could materially increase the speed of machines in the gigacycle range where transit time is a basic limitation (see Sections 4.3.4 and 4.7).

TABLE 7.1. A comparison of serial and parallel operation for certain figures of merit.

Figure of merit	Serial	Parallel
Circuitry		
Amount	Considerably less	
Complexity of control		Simpler
Speed of the arithmetic unit		Faster
Compatibility		
With main store	Compatible with serial or parallel storage	Compatible with parallel storage
With type of control	Compatible with synchronous control only	Compatible with synchronous or asynchronous control

7.5.2. SYNCHRONOUS OR ASYNCHRONOUS OPERATION

In a completely synchronous computer, the timing of all operations is controlled or synchronized with a clock, and each operation requires an integral number of clock intervals. The machine is controlled by a clock pulse oscillator whose successive output signals define the smallest time interval recognized in a machine, namely a bit period. Other pulses defining major and minor periods are derived either from the clock by

means of counters that produce an output signal after accumulating certain specified clock "counts" (see Sections 3.10 and 6.1.1) or from timing tracks (see Section 7.6.3). All switching waveforms rise and fall at times defined relative to the clock waveform, and the duration of the rise and fall times must be less than a specified time which is dependent on the nature of the switching elements (see Chapter 4).

In an asychronous machine, each operation is initiated upon completion of the one preceding. Not only is there a wide tolerance on the shape and amplitude of control pulses, but because the timing of a particular pulse is unimportant provided it does not occur until the preceding operation is completed, it need not be accurately phased relative to the clock. Although an asynchronous machine can be designed so that on the completion of each operation a pulse is emitted which initiates the next operation, as a matter of convenience the machine can be driven from a clock source to which it is, strictly speaking, synchronized.

7.5.2.1. Synchronous Control

In a synchronous computer there is a clock pulse generator which serves as the source for various timing signals. The period between successive clock pulses is the smallest interval of time defined in the machine. Other time intervals of importance in its operation are defined by the intervals between designated states of clock pulse counters. Two of the major time periods have already been referred to, namely that required to transfer an instruction from the main store to the control register(s), and that required to actually execute an instruction. In addition to these major periods for instruction acquisition and execution, there are other periods specified for the performance of various operations. There is also a requirement for the generation of signals that define specific points within these time intervals. The circuits that generate the various timing signals required are sometimes referred to collectively as a timing pattern generator.

As stated in Section 7.5, the execution of each instruction requires the performance of a number of operations in sequence. In a synchronous computer, the time for initiation of each minor or major operation is specified by a timing signal from the timing pattern generator. With synchronous control, no action can take place, i.e., there can be no advance from one internal state to the next unless a clock pulse is present. The number of clock pulse periods required for an instruction acquisition period is fixed, since the same operations are performed in all acquisition periods. The number allotted for the execution of each instruction is dependent on and determined from the operations of which it is comprised.

A typical arrangement for a synchronous control unit is shown in Fig. 7.5. All operations required in the execution of an instruction are

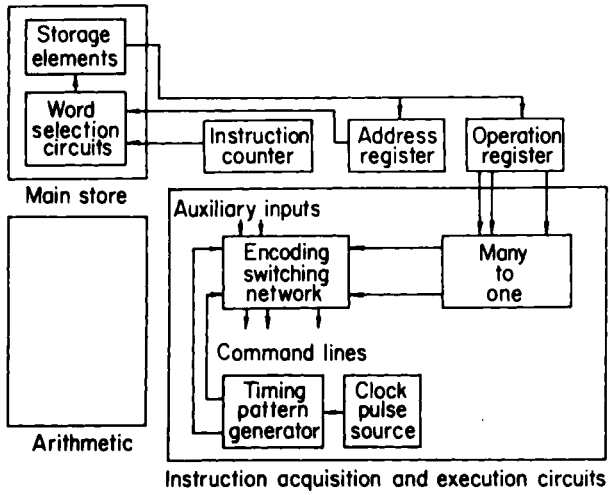


FIG. 7.5. Synchronous control for a single-address GP computer

dictated by the order code placed in the operation register during the acquisition period immediately preceding. The outputs of the operation register are applied to a many-to-one function table, causing one output line to be energized for each instruction. These output lines, as well as a number of auxiliary inputs and the timing signals from the timing pattern generator, are applied to an encoding switching network. One may think of the inputs from the many-to-one (decoding) table and the auxiliary inputs as controlling the connections made between the inputs from the timing pattern generator and the output command lines of the encoding switching network. Some typical auxiliary inputs are signals indicating: (1) whether the conditions of conditional type instructions have been satisfied, (2) whether the computer is currently in an acquisition or execution period, (3) which step is being performed in the execution of an instruction whose execution period consists of several steps. Whenever an instruction requires more than one step in its execution, the individual steps are indicated by a counter which is caused to advance after the completion of each step. When the required number of steps has been completed, a signal is produced to initiate a new instruction acquisition period.

For instructions referring to the main store, the storage selection circuits are controlled by the address register. During an acquisition

period, they are controlled by the instruction counter. This counter normally is advanced by one increment during the execution of each instruction, so that in the succeeding acquisition period it will cause the selection of the instruction in the next sequentially numbered address. If the instruction just executed called for a transfer of control to some other address, that address will have been transferred into the instruction counter from the address register.

7.5.2.2. Asynchronous Control

In an asynchronous machine, a control cycle normally begins by generation of a signal at a particular physical location. Branchings will then take place according to the order code of the instruction received from the main store. However, there is no precise control over the intermediate steps of each operation. One may think of the action as free running—each action in an operation triggers another action until the particular operation is completed. A signal, indicating the completion of a particular operation, is always generated at a specified location, and is used to initiate the next chain in a specified sequence of operations.

A typical arrangement for an asynchronous control unit is shown in Fig. 7.6. When the computer is switched to an active computing state,

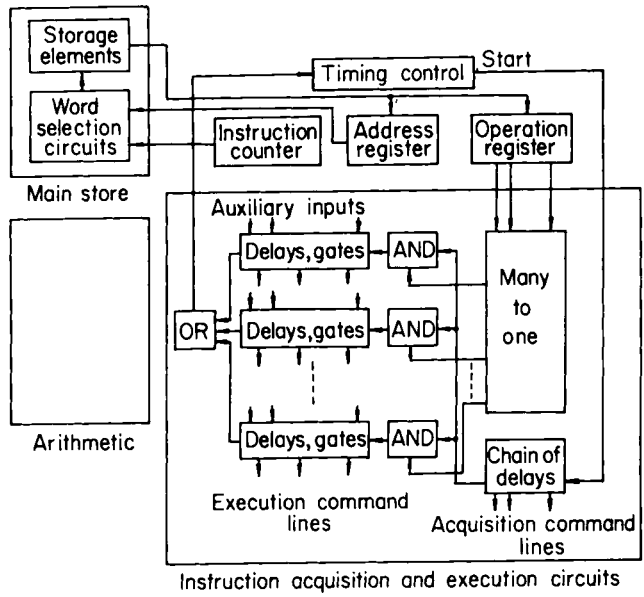


FIG. 7.6. Asynchronous control for a single-address GP computer

the timing control circuits initiate an operation by generating a "start" pulse. This pulse is applied to a chain of delay units from whose outputs the acquisition period timing pulses are obtained. The lines along which these timing pulses are directed are the command lines to circuits that must be activated to execute the operations required during the instruction acquisition period, namely selecting the next instruction from the main store and transferring it to the control register (i.e., the address and operation registers). The acquisition period is completed by the time the start pulse activates the instruction execution circuits.

The emergence of the pulse at the far end of the acquisition delay units signals the beginning of the execution period. The pulse is applied to the inputs of a number of AND gates, as shown in Fig. 7.6. There is one AND gate for each instruction in the computer's repertory, and the pulses will pass only through the one connected to the currently energized output line of the many-to-one function table, which in turn is determined by the order code in the operation register, i.e., the instruction to be executed. After passing any one of the AND gates, the pulse is applied to a chain of delays and gates which generate a sequence of signals which are directed along command lines to activate circuits that cause the execution of any given instruction. For some instructions, the pulse may be recirculated along some closed loop within a particular set of delays and gates until a specified condition is met, as signaled by one of the auxiliary inputs. For example, in operations such as multiplication, division, and shifting, a particular sequence of commands must be repeated a number of times. The completion of the required number is usually indicated by a counter which activates a command line. The signal on this command line causes the recirculating loop to be effectively broken, and also opens a gate allowing the pulse to pass on. When the execution of an instruction has been completed, the pulse returns to the timing control circuits so that they can initiate the next operation.

Before the instruction counter can be triggered to its next state, it must receive an indication, i.e., a revertive signal, that its last output signal was successful in activating certain specified circuits. If the operation were unsuccessful, the signal to advance the counter would not arrive, and the counter and therefore the whole computer would be stopped. This feature may be useful in locating failures within a computer.

7.5.2.3. Comparison of Synchronous and Asynchronous Control

In a synchronous system, the duration specified for an operational period is determined from the maximum time required by the longest operation to be performed in that period. This is not the case in an asyn-

chronous system. Asynchronous operation of regular networks (i.e., ones with exactly one equilibrium state for each possible combination of primary input values*) yields a speed advantage, in principle, because the average time to transmit a signal through a network is the average (rather than the maximum) delay per circuit times the number of circuits. With either type of control, overall speed is limited by the main store since addition (or subtraction), logical and transfer operations can be completed in less time than it takes to acquire a word in storage. By making serial multiplication and division asynchronous, the average time for execution of these operations can be significantly reduced.

The fact that by far most computers are synchronous machines may be ascribed to several factors, including the following. First, there are potential hazards in asynchronous networks due to variation in response time of active elements and in the transmission time of signals. An interlock system (see Section 4.3.4) for nonregular networks means additional circuitry which, in most cases, increases the delay per circuit to where there is no longer a clear advantage over synchronous operation. Finally, because of an indeterminate time for execution of operations and the complexities required, in general, for hazard free operation, asynchronous systems are considered more difficult to design, understand and service.

Table 7.2 provides a brief resume of certain figures of merit of interest in a comparison of synchronous and asynchronous operation. A principal consideration in reference to the computing element is the consistency of its operating time. For example, the operating time of a relay is a function of its exact adjustment, the tolerances of its components, etc. Because of a wide variation in operating time, relays are best suited for asynchronous control. Vacuum tube, transistor, and magnetic core circuits are suitable for either type of control.

7.5.3. Numerical Representation in the Arithmetic Unit

Two major considerations in the design of an arithmetic unit, namely serial or parallel operation, and synchronous or asynchronous control, have already been considered (in Sections 7.5.1 and 7.5.2). It was pointed out there that in a parallel asynchronous machine, a number can be transferred from one register to another upon receipt of a signal pulse whose time of occurrence, rise and fall times, and duration are all noncritical. In a serial synchronous machine, the timing of each waveform relative to the clock waveform, as well as its rise and fall times, and duration are all critical.

---

* Henney, F. C., III [1961] *Iterative Arrays of Logical Circuits,* M.I.T. and John Wiley & Sons, New York.

TABLE 7.2. A comparison of synchronous and asynchronous control for certain figures of merit

Figure of merit	Synchronous	Asynchronous
Compatibility of control With arithmetic unit	Compatible with sync. or async. arith. unit	Compatible with async. arithmetic unit
With storage	Compatible with serial or parallel storage	Compatible with parallel storage
Speed of operation	Determined by estimated speed of slowest element	Determined by average speed of all circuits
Component tolerance	Only limited degradation allowable because of requirements on wave-shapes and pulse coincidence at gates.	Speed independent circuits operable after considerable degradation of component characteristics.
Ease of maintenance	Simpler organization, easier to understand.	A dc coupled machine can be put in a state of static equilibrium
Ease of understanding	Operation of circuits is straightforward.	Hazard-free networks are generally more complex.

We will now consider the effects of certain choices in numerical representation upon the complexity of the arithmetic unit and, therefore, its control, i.e., all the circuits required to cause transfers of information into and out of the arithmetic unit as well as to supply signals that cause arithmetic and logical operations to proceed at the right time and in proper sequence. The items which we shall consider are (1) the number base, (2) means for positioning the radix point, (3) the representation of negative numbers. In most computers, numbers are represented internally either in a binary or some type of binary-coded decimal representation. In those cases where it is either convenient or necessary to enter and display data in decimal form, input–output data conversions would be required for a binary machine (see Sections 6.4.3, 6.4.4). On the other hand, arithmetic operations are more complex for a binary-coded decimal machine (see Sections 6.1.3, 6.1.4.2.3, 6.1.5.2, 6.1.6.2). Various means for representing negative numbers are described in Section 6.1.4. A comparison of fixed positioning of the radix point and floating point operation is provided in Sections 6.3.3. and 6.3.4. In a

computer with built-in floating-point operation, counters must be provided to store the exponent associated with each number. The shifts required prior to or after arithmetic operations are effected under control of these counters, and circuits that sense nonzero digits in a shift register. The exponent counter receives an input pulse after each shift. By accumulating them, it keeps track of the number of shifts.

The effects on the arithmetic unit and its control of the three major choices in numerical representation are summarized, for convenience, in Table 7.3.

TABLE 7.3. Effects of numerical representation on the arithmetic unit

Choice	Comment	
Number base		
Binary-coded decimal	General familiarity decimal input-output conversion simpler	Internal mechanization more complex than for binary
Binary	Simpler arithmetic	Requires conversions of input data and output data for decimal display
Radix point		
Fixed	Simple arithmetic unit	Limited range of numbers
Floating	Wide range of numbers	Complex arithmetic unit
Representation of negative numbers		
Absolute value and sign	Easy multiplication and division. Simpler input-output data conversion.	Addition and subtraction more complex than with complements. Two values of zero.
Two's complement	Addition and subtraction easy. Derivation of complement easy. One value of zero	Conversion to signed form is complex. Normal multiplication and division methods require corrections, special methods (Section 6.1.5.1.6) do not.
One's complement	Derivation of complement very easy, facilitating conversion to and from signed form before and after multiplication and division	Extra end point correction in addition and subtraction. Need to know signs prior to conversion for multiplication and division complicates serial machine (since l.s. bit is first). Two values of zero.

7.5.4. Number of Addresses in an Instruction

The instructions described in Chapter 2 and in Section 7.2 are of a class referred to as single-address instructions. This means simply that only one address is referred to in an instruction. However, there are a number of machines in which reference is made to more than one address per instruction. A brief description of different addressing systems that have been used follows:

A one-address system requires three instructions for most arithmetic operations: two instructions to bring the operands from storage and one instruction to transfer the result back to storage. Instructions are executed in sequence according to the contents of an instruction counter. The code bits reserved for specifying an address in the main store may be used for other purposes in the case of instructions requiring no internal address, e.g., shift, test, and input-output instructions.

In one type of two-address computer (e.g., the IBM 650) there is included in each instruction a second address which usually specifies from which storage location the next instruction is to be obtained. Consequently, the instruction counter (required in the one-address computer) may be replaced with a simpler unit, an instruction register. This system facilitates minimum access programming* for computers having a nonrandom access main store, because the next instruction can, in general, be placed in an address that will be accessible to the reading stations, shortly after execution of the current instruction. In another type of two-address machine (e.g., the ERA 1103) both addresses may refer to operands used in the execution of an instruction.

In the usual three-address computer, the three addresses specified in each instruction are those from which two operands are to be obtained and that to which the result of an operation on the two operands is to be transferred.

In a computer with a four-address instruction (e.g., the SWAC) each instruction contains the three addresses of a three-address computer, plus an additional address specifying the location from which the next instruction is to be obtained.

The control sequence of a machine using single address instructions proceeds in two stages: the acquisition of an instruction from the main

---

* Minimum access programming techniques can also be used with one-address machines. They depend on an internal addressing arrangement wherein successively numbered addresses are separated by a specified number of word lengths. This relative inflexibility makes the one-address machine less efficient in minimizing access time. See Knuth, D. E. [1961].

store and the execution of that instruction. Since the address of the next instruction is obtained by adding a constant to the address from which the present instruction was obtained, some facility must be provided to produce the new address at the proper time. Since the constant is usually 1, a counter is provided. In the execution of a jump instruction, the contents of the counter are replaced by the contents of the address register. The control sequence of a two or more address machine proceeds in two or more stages, depending on whether the addresses are dealt with concurrently or sequentially. As stated earlier, if each instruction contains the address of the next instruction, the sequence control counter can be replaced by a simpler register. This "next instruction register" may receive its information from the instruction register and one or the other may be eliminated.

As noted earlier, if each instruction specifies the address of the next instruction to be executed, there is greater programming flexibility. However, additional storage space may be required for these extra addresses. The amount of extra storage space consumed is difficult to ascertain precisely, since the length of a word is usually dictated by the maximum number of significant digits that are to be available for representing a number. The space consumed by extra addresses depends also on the size of the main store. On the other hand, a program of multiple address instructions requires fewer words of storage than an equivalent program of single address instructions. Another advantage of a multiple address system is that the main store does not have to be consulted as often, and therefore, the over-all computer speed is less dependent upon access time to the main store. The value of these advantages depends on how efficiently the multiple-address system is used. The advantage of the single-address system is that it permits a simpler control unit.

7.5.5. Inclusion of Special Control Features

An instruction may be converted to a new one by transferring it from the main store to the accumulator and then performing some operation on it. This procedure, though, is often inconvenient and wasteful of time and storage space. Therefore, in some computers a special register or group of registers referred to as index registers is provided. The contents of these registers can be automatically added to an instruction before it is executed (by means of special circuitry in the control unit) at the discretion of the programmer. A group of index registers is often referred to collectively as a B-box because this designation was used in the computer in which they were first used, namely that built at the University of Manchester. In a machine provided with index registers, extra bits

must be reserved in each instruction word for their addresses. For example, if four bits are assigned for a B-box address, as many as 16 index registers may be referenced. In most applications index registers are used only to modify the address part of an instruction. Therefore, they usually contain no more bits than required for an address in the main store.

Since the incorporation of index registers into a computer enables an instruction to be modified before it is executed, the following distinction must be made: An instruction in the form in which it appears on the programmer's sheet, on an input tape, or in the internal memory of the computer is termed a presumptive instruction. The instruction executed by the computer is termed the effective instruction. To facilitate the use of index registers, the control unit of the computer is modified so that the presumptive instruction may be automatically converted to the instruction to be executed. The additional capabilities incorporated into the control unit are as follows: (1) after an instruction has been selected from the main store, but before it is executed, the index register designated by the B-box address is inspected; (2) the contents of this index register are added to (or subtracted from) the presumptive instruction; (3) while B-box address bits in the actual instruction may be set to zero (as in the University of Manchester machine), the B-box address bits in storage are unchanged, however. The necessity of having to sequence these modification procedures complicates the control unit.

For a machine with index registers, other facilities must be added besides the capability for automatic modification of an instruction just described. These consist mainly of adding to the machine's instruction repertory, instructions relating to the B-box. Some basic instructions of this type are as follows: (1) an instruction for transferring data into the index registers; (2) an instruction for copying the contents of an index register into the main store; (3) at least one instruction that performs a simple arithmetic operation on the contents of an index register. Such an instruction is useful in permitting index registers to serve as auxiliary accumulators, e.g., for counting (from which the term "index register"), leaving the accumulator undisturbed during the main computation. A typical instruction might be: subtract the contents of storage location s from the contents of index register i. However, a difficulty arises in connection with instructions intended to operate on the contents of an index register. This is because such an instruction must specify the index register to be operated upon, yet the bits reserved for an index register address are normally used to indicate the index register whose contents are to be used to modify the instruction. Therefore, an additional type of B-box instruction, one which can never be modified by the

contents of an index register, must be provided. In this type of instruction, designated as non-B modifiable, the B-box address specifies the index register on which the instruction operates and nothing else. Each of the three types of B-box instructions described earlier in this paragraph may be of the B modifiable or non-B modifiable type.

It is also useful to have a transfer type of B-instruction conditional on the contents of a designated index register. The Manchester University computer has two instructions of this type, an absolute B-conditional transfer instruction and a relative B-conditional transfer instruction. The term absolute indicates that control is transferred to a specified address, and relative indicates that control is transferred a specified number of positions from the original position. An absolute instruction of this type might be of the form: (4) take control to the next instruction in order if the number in the index register is negative, otherwise transfer control to 1 plus the address which appears in the conditional transfer instruction. It should be noted that an additional rule is needed to specify which index register decides the behavior of the control. In the Manchester University computer, the decisive index register is the one last operated on, i.e., the index register appearing in the last actual instruction, prior to the B-conditional transfer instruction. On occasions when it is required to execute a B-conditional transfer instruction conditional on the contents of index register *i* while the last operation was performed on a different index register, a dummy instruction may be inserted which formally acts on *i* without having an effect on the program, i.e., without altering the number in index register *i,* e.g., "copy contents of index register *i* into storage location *s* where *s* is a spare," or "subtract the contents of *s* from index register *i* where *s* contains zero."

The economy to be realized from the use of a B-box will be demonstrated by programming a problem first without and then with the use of B-box instructions. Example 7.1 shows two such programs, each of which is designed to cause the numbers in storage locations 50 through 99 to be multiplied by the contents of storage location *n* and the products returned to these locations. Note: the instruction codes used in the first program are defined in Table 2.1 except for M m, which means "produce in the accumulator the product of the numbers in the accumulator and storage location m." In the second program, F m n is a non-B-modifiable form of instruction which fills index register n with the contents of storage location m. R m n (also non-B-modifiable) is a combination tally and conditional transfer instruction: 1) It subtracts* .00 . . . 01 from the

* The greater convenience of counting backwards (facilitating the use of the conditional transfer instruction) is one reason why subtraction may be preferred to addition for an arithmetic operation on the contents of an index register. Also, addition may be compounded from subtraction, but not vice versa.

contents of index register n, and 2) if the remainder is $\geqslant 0$, transfers control to location m, otherwise to the next consecutive location. The other instructions are B-modifiable forms of instructions defined earlier.

Example 7.1

Program without B-box instructions

Address	Instruction		Explanation
000	cA	n	Multiplies the contents of storage locations
001	M	099	050 through 099 by the contents of location n.
002	C	099	
003	cA	002	Converts the instruction C m (in storage loca-
004	S	102	tion 002) to C m − 1.
005	C	002	
006	cA	001	Address m in M m is used as a tally number
007	S	100	to detect when 50 numbers have been oper-
008	T	012	ated on, indicated by (M m − M 051) < 0.
009	A	101	M m is replaced by M m − 1, produced as
110	C	001	the net effect of instructions 007 and 009.
011	U	000	Stop
012	Z		
100	M	051	Constants are stored here.
101	M	050	
102	—	001	

Program with B-box instructions

Address	Instruction		B Address	Explanation
000	F	100	01	Fills index register 01 with contents of location 100.
001	cA	n	00	
002	M	050	01	
003	C	050	01	
004	R	001	01	Subtracts 1 from index register 01 and transfers control to location 001 if remainder $\geqslant 0$.
005	Z			Stop
100	—	049		The constant, 049, is stored at location 100.

For convenience in use of conditional transfer instructions, addresses are operated on in the order 99, 98, . . . 50 rather than the reverse order. It is assumed that index register 00 is cleared before the program is initiated (or that 00 is a fictitious address whose contents are interpreted to be zero). By reducing the number in index register 01, R 001 01 reduces the tally number and the address in locations 002 and 003.

Uses to which index registers may be put include the following:

(1) To reduce the amount of storage space required by programs containing computer modified commands. For example, when computed results have to be stored in sequential locations, the same instruction may be used repeatedly, with the address part being B-modified each time. Without the use of index registers, three separate instructions would be required to add an increment to the address part of an instruction during each traversal of an iteration loop. With index registers, the number of instructions is reduced to one.

(2) To dispose of or obtain information from stored tables. If the value of a function corresponding to a given argument is desired, a table may be stored having the value of $f(x)$ stored in location x. To obtain $f(x)$ it is then only necessary to plant the argument in an appropriate instruction. The B-box may be used to modify this instruction.

(3) In sorting processes.

(4) To modify instructions permanently stored in those channels of a dynamic magnetic store which are not provided with a record head, i.e., for altering so-called dead programs. For example, such dead programs may be used to provide a permanent input routine. They have the advantage of precluding the possibility of an accidental writing-over operation. Use of the B-box retains this advantage while eliminating the disadvantage of not being normally able to modify instructions within a dead program during use.

(5) To plant links in closed subroutines. A closed subroutine is preceded by instructions which insert, at the close of the subroutine, an instruction returning control to the main program. This may be done by storing the link in an index register at the beginning of the subroutine, using the same index register to modify the instruction which returns control to the main program.

(6) To change the internal addresses of a subroutine, thus enabling that routine to be placed in an arbitrary section of the main store. We recall that a subroutine contains both external addresses (which refer to fixed positions in the memory) and internal addresses (which refer to positions within the subroutine itself). If a subroutine is written to fit into a section of the memory starting at location zero, it may be made to fit into another section starting at location n if n is added to all internal addresses. If n is stored in an index register at the beginning of a subroutine,

and if all instructions having internal addresses are modified by the contents of that index register, the subroutine will operate properly.

The following computers, among others, have a B-box type of facility

The Ferranti (University of Manchester) Computer
The Whirlwind I (adapted)
The MIDAC (University of Michigan)
The Electrodata Datatron
The IBM-704

Some machines have a repeat counter that controls the number of times an instruction is repeated. In these machines a built-in instruction is provided for transferring a number from the main store to the repeat counter. The number in the counter is diminished by 1 each time the instruction is repeated, until the counter reaches 0, whereupon the process is ended. For example, the UNIVAC-1103A computer, which is a two-address machine, has a repeat instruction which states: repeat the next instruction n times, augment one address by i and the other by j (where the allowed values for i, j are (0, 1), (1, 0), (1, 1). A repeat counter facilitates such operations as adding a long list of numbers stored at sequentially numbered addresses, and transferring large blocks of information between the main store and input or output devices. In the latter case, the repeat counter is used to hold the number designating how many words are to be transferred. As indicated in the preceding discussion of index registers, it is often desirable to alter the address part of an instruction each time it is repeated. A simple way to provide this feature is to convert the control register that holds the address of an instruction to a counter. Then, the pulses that are sent to the repeat counter upon each execution of an instruction to be repeated may also be sent to this counter to change the address by 1.

7.5.6. Integration of Input–Output Equipment

Equipment used either for the preparation of data and/or its communication to the computer, or for producing a record of computed results is referred to generally as input–output equipment. An important distinction in the way this equipment is used is whether its operation is on-line or off-line. In on-line operation there is direct control of the equipment by the computer. In off-line operation there is no connection between the equipment and the computer: input equipment used off-line records data on an external storage medium for subsequent entry into the computer by on-line equipment; output equipment used off-line pro-

cesses data recorded on an external storage medium by on-line equipment.

The control console of a computer is usually provided with special keys and switches by means of which signals may be generated for such functions as controlling the internal operation (e.g., starting or stopping a computation), revising the contents of a selected storage location, or correcting small errors. Depression of a key actuates a mechanical switch, either directly or indirectly by energizing a solenoid. This action produces binary signals, on one or more wires, which are transmitted to the proper functional unit, e.g., one or more control flip-flops, a counter, control register, or storage register. Where it is convenient to have a typewritten record of data entered via a keyboard an electric typewriter may also be provided as an on-line data entry device. The signals generated by depression of a key are transmitted to the computer and used concurrently to actuate a typing bar. The use of a Flexowriter allows the production of a punched paper tape record as well.

For the entry of large amounts of data, e.g., complete programs and libraries of subroutines, it is more efficient to prepare the data off-line by means of data preparation machines, e.g., keyboard controlled paper-tape punches and card punches. After the data has been prepared it can be read into the computer at a rapid rate by means of higher speed readers, e.g., paper-tape readers, punched-card readers, or magnetic tape readers connected on-line. An additional advantage of off-line data preparation is that it permits the location and correction of most errors introduced by the operator prior to the entry of that data into the computer. In addition to various error checking procedures that may be used, verification may be obtained by means of standard or special equipment designed for this purpose. In either case, circuitry is provided that compares the recorded data keyed in by one operator with that prepared independently by another. If there is a disagreement an indication is provided, usually by locking of the keyboard, and the erroneous data can be removed. Of course, though this procedure greatly reduces the probability of an undetected operator error the same error made in corresponding characters of both records would go undetected. Such errors would then have to be picked up by other means.

Since a single keyboard can be connected electrically to more than one set of actuating devices, one used to produce typewritten records or documents such as checks or invoices can be modified to cause the recording of data on an input medium simultaneously.

In addition to its data entry devices, the control console is also usually provided with switches and indicators which allow the contents of registers in the arithmetic and control units or storage locations in the

main store, as well as the status of specified circuits, to be monitored. The data recorded by output equipment operating on-line may be in a form directly usable, such as the typewritten page produced by an electric typewriter, or in a form not directly usable such as data recorded on external storage media, e.g., punched-paper tapes, punched cards, or magnetic tape. A visual record can be produced from the external storage media by means of off-line electric typewriters activated by punched-paper tape and line-at-a-time printers activated by punched cards and magnetic tape. The principal reason for the use of off-line output equipment is that it allows a greater data output rate. This is because, in general, data can be accepted by a recording device at a greater rate than by a corresponding printer. The types and number of pieces of peripheral equipment used depends, generally, on the characteristics of the computer —its provisions for control of input–output equipment, its speed, the capacity and type of main store. The types of output equipment called for depend also on the output rates and form required of various classifications of data to be recorded on external storage media and/or printed. For example, because recording on magnetic tape is faster than the operation of a card punch or a mechanical printer, a magnetic tape unit would be used if large quantities of output data were to be recorded in a brief period. If punched-card records were also required they could be produced off-line by means of a magnetic tape-to-punched card converter. Printed records could be obtained either from a magnetic tape or punched-card controlled printer. An added benefit of producing intermediate records on external storage media is that subsequently they can be used as inputs to a computer as well as to a number of specialized units of peripheral equipment.

For applications involving long sequences of operations on relatively small amounts of input data and producing relatively little output data, the input–output data transfer rate is moderate, and a simple type of computer organization may be used in which input–output instructions are similar to internal instructions operating on single operands. For those applications where the ratio of input–output to internal operations is appreciable, (mainly in commercial record keeping and data processing applications), it may be necessary to provide higher speed input–output equipment or so design the computer's logical structure that it can control computing operations and certain input–output operations simultaneously. One way of accomplishing parallel internal and input–output operation is to provide instructions which can cause the transfer of a block of data between the main store and an auxiliary store. The latter must be provided with separate controls which allow it to accept or trans-

mit data between itself and external devices without the need for access to the main store. Thus, individual data transfers between the main store and external devices do not disrupt the main program.

Whenever two storage media of different rates must be interconnected, an intermediate store must be provided, referred to, for obvious reasons, as a buffer. An efficient way to utilize an input buffer is to cause the transfer of a block of data from the input medium to the buffer during the time the computer is processing the last previously received block of input data, and at the completion of this processing, to cause the new set of input data in the buffer to be transferred at high speed to the computer's main store. An output buffer would be used in a similar way.

In larger systems it is the usual practice to provide one appropriately large storage unit to act as buffer (and control) between the internal store and any one of several selectable external storage units of a particular type. For example, a single buffer may accommodate as many as ten magnetic tape units, (as in the UNIVAC-1103 A or IBM-704 computers), each of which is identified by an address code.

The storage capacity of a buffer depends primarily on the chosen unit of information transfer between the internal and external stores, the difference in their rates and the time allowed for transfer. The buffer must be capable of receiving information at one rate and transmitting it at another. This can be accomplished by means of static registers or a combination of static registers and a less expensive cyclical type of storage. In addition to being responsive to a wide range of input frequencies, static registers can also provide a completely asynchronous means of read-in and read-out.

A common characteristic of the external storage problem is the need for a very large volume of storage accessible to the computer, though not necessarily at high speeds. Therefore, the size of the external store, though varying with the application, is usually several times that of the internal store. An insurance company file, for example, may contain 10^{10} bits. Access times of the order of seconds rather than milli- or microseconds can be tolerated here. The most commonly used external storage media are punched cards, punched paper tape, and magnetic tape, disks, or drums. Programs held in these external stores must be transferred to the internal store before they can be executed.

As in the computer, an assemblage of bits in an external store is usually organized into groups called words. However, the form of a word or character stored here need not be the same as in the internal store. For example, a word on magnetic tape may be in serial–parallel form, a word of 36 bits, say, being arranged on six tracks with six bits

per track. In the computer the same word may be stored as 36 serial bits. A sequence of words in an external store is called a block. Its size need not have any relation to details of internal storage.

Because of the relatively long access time to a word in an external store, it is inefficient to use an instruction for the transfer of only one word. Accordingly, it is the practice to cause the transfer of information between the computer's fast access storage and consecutive locations on tapes, disks, or drums by an instruction that refers to a block of several words (called a block transfer instruction) rather than by instructions referring to single words.

Transfers of information from external to internal stores are most easily effected by addressing information in the internal store. During the execution of a program, if it is anticipated that the capacity of the internal store will be exceeded, a few internal storage locations are reserved for instructions which will record appropriate blocks of intermediate results on tapes, disks, or drums, and then call in blocks of more program steps or data, as required.

The control of the external store must supervise the tasks of handling the medium and causing the indicated locations to be selected. The associated problems vary with the particular storage medium. As an example, consider a magnetic tape handling unit. Here, a search operation may be initiated by an instruction from the computer which identifies the tape handling unit to be consulted, and gives the address of a particular block on the tape. These two numbers (i.e., that of the tape unit, and the block address) are stored by a separate tape control unit. This permits other instructions to be executed by the computer while the search progresses. At some point in the execution of the main program, the computer will require the execution of an instruction calling for the desired block of information to be read from the tape. Since the instant at which the read instruction may occur and that by which the search is completed will not, in general, coincide, it is necessary to provide some type of interlock in the system which will allow the read operation to occur only if there is an indication that the search for the desired block has been completed. The tape control unit provides the following facilities to accomplish the search: storage for the code identifying a tape unit and the address of a block, a means for distinguishing block address information from other information stored on the tape, controls to start the tape drive and accelerate the tape smoothly when a block is called for (in either a reading or recording operation), comparison circuits to detect when the desired block has been reached, controls to stop the tape drive, and other miscellaneous controls.

7.5.7. MICROPROGRAMMING

The operations called for by a single machine instruction are effected, in general, by a sequence of elementary commands, sometimes referred to as micro-operations. The number and sequence of these commands for the execution of each instruction are usually wired into the computer. Each instruction can be considered as being composed of a microprogram of commands, just as a normal computation is composed of a program of instructions. The concept of microprogramming is useful in designing the control circuits of a computer. Its primary purpose is to situate and interconnect the elementary commands in such a way that they are readily accessible either for the modification of old instructions or the formation of new ones. The use of microprogramming in the design of the control unit does not necessarily imply a more efficient final logical design. What it does provide is conceptual clarity and flexibility in respect to modification of the instruction repertory.

The design of a control unit based on the microprogramming technique will be described for a parallel computer. It is convenient to regard the control unit as consisting of a control register unit and a microcontrol unit. Both are shown in Fig. 7.7 and will be discussed in turn. The

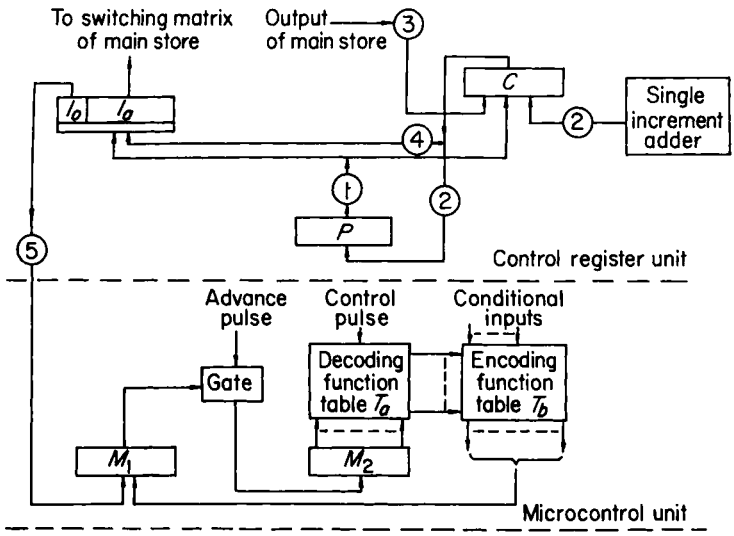


FIG. 7.7. A microprogrammed control unit

control register unit consists of a group of registers and an adder together with a switching system which enables transfers of information between

the registers. The names and functions of these registers are shown in Table 7.4. The incorporation of more registers into the unit would enable additional facilities, e.g., index registers, to be added to the microprograms.

TABLE 7.4. Registers in a control register unit

Register	Function
I_a	The contents of the address register, I_a, control the selection of a location in the main store.
I_0	The order code buffer register, I_0, holds the code of the instruction to be executed next by the microcontrol unit.
P	The program sequence control register, P, holds the address of the next instruction to be executed.
C	This register serves two functions: (1) it acts as an address counter, receiving an address from register P and augmenting it by 1, (2) it acts as an instruction buffer register, receiving first each instruction selected from the main store.

A number of preparatory commands must be carried out prior to the execution of any instruction. These instruction acquisition commands select an instruction or operand from the main store, transfer it to the control unit, and set up the control unit to repeat the look-up operation for successive instructions in the program. The numerical code for each of these commands, as well as a description of the operations performed by each is given in Table 7.5. At the conclusion of the instruction acquisition operations, the following conditions exist: (a) the address of the operand is in register, I_a; (b) the order code, defining the entry point to a microprogram, is in register, M_1.

TABLE 7.5. Instruction acquisition commands

Command No.	Operation
1	Transfers address of instruction to be executed from the register P to the register I_a, and also to the register C.
2	Adds 1 to the contents of register C and transfers the result, the next consecutive address, to the register P.
3	Selects from the main store the instruction whose address is specified by register I_a, and transfers this instruction to register C.
4	Transfers the order code and operand address of the instruction to be executed from register C to registers I_0 and I_a, respectively.
5	Transfers the order code, which defines the entry point to the microprogram of a given instruction, from register I_0 to register M_1.

It is the function of the microcontrol unit (see Fig. 7.7) to actually execute the instruction obtained from the main store by the control register unit. Application of an "advance" pulse to the gate causes the code of the entry point, which may be considered to be an address, to be transferred to register M_2. When a control pulse is applied to the many-to-one function table T_a, one, and only one, output line will be activated, according to the address in register M_2. The activation of any line causes gates to produce signals required for a particular command. The code specifying a particular instruction must cause a unique sequence of commands to be executed. Therefore, the activation of any output line of T_a can also be used to cause the address of the succeeding command in the microprogram to be entered next into register M_2 (via register M_1). This can be mechanized by use of a one-to-many table, T_b, which when activated produces a command address on its parallel output lines, (not, in general, unique to a particular input). The progression from one command of a microprogram to the next is achieved by alternate application of the control pulse to T_a and the advance pulse to the gate, the advance pulse being applied just before the control pulse.

To run a program, it is only necessary that the address of the first instruction to be executed be placed in register P, and that the address 00 . . . 01 be placed in register M_2. Then the application of successive pulses to the input of table T_a, and to the gate, will cause the first and succeeding instructions to be executed by a microprogram of elementary commands.

The steps involved in the execution of each instruction will now be described. We recall, first, that at the completion of any microprogram, control normally is advanced to the first in the sequence of commands necessary to select a new instruction from the main store and set up the control register unit for selection of the next instruction. This sequence consists of commands 1 through 5 (described in Table 7.5). However, at the conclusion of an unconditional transfer of control instruction and, it follows, the execution of a successful conditional transfer of control instruction, command 1 is skipped. The effect of this is twofold. First, it causes the instruction at the address specified in the transfer of control instruction to be selected from the main store, rather than the instruction at the address one greater than that from which the transfer of control instruction itself was obtained. Secondly, it places in the next instruction register P the address one greater than the address to which control was transferred by the transfer of control instruction. The effect of skipping command 1 is best illustrated by an example. In Table 7.6, the first column shows the effects of the operations dictated by commands

1 through 4 in the instruction acquisition phase of cycle $p + 1$, following the execution of a nontransfer of control type of instruction. The second column indicates the effects of commands 2 through 4 in cycle $p + 1$ following receipt of a transfer of control instruction in cycle p.

TABLE 7.6

Cycle	Command	Operations	Command	Operations
p	2	Augments the address j in C by one and transfers $j+1$ to P.	3	Transfers the instruction [U m] from the main store to C
	Receipt of any instruction except [U m] eventually transfers control to (1)		4	Transfers instruction in C, [U m], to I_0, I_a
			9	Transfers control to command 2.
$p+1$	1	Transfers contents of P, $j+1$, to I_a, and also to C.		
	2	Augments the address $j+1$ in C by one and transfers the result $j+2$ to P.	2	Augments the address m in C by one, and transfers the result $m+1$ to P
	3	The instruction whose address $j+1$ is in I_a is selected from the main store and placed in C.	3	The instruction whose address m is in I_a is selected from the main store and placed in C
	4	The operand address and the order code of the instruction in C are placed in I_0 and I_a, respectively	4	The operand address and the order code of the instruction in C are placed in I_a and I_0, respectively

An explanation of the microprogram for executing each of the instructions A m, cA m, S m, U m, T m, C m, and Z (described in Table 2.1) will now be given. Reference to Fig. 7.8, which indicates the sequence

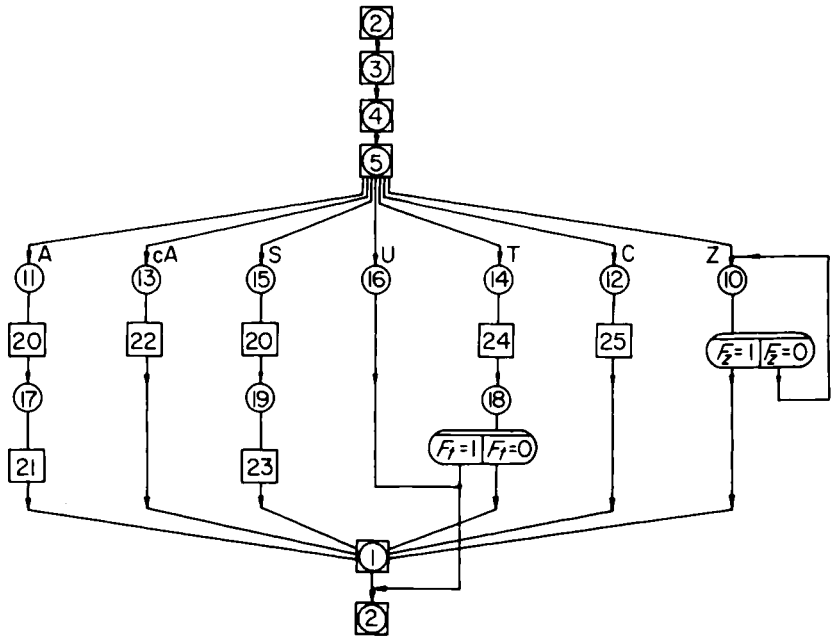


FIG. 7.8. Flow diagram of microprograms

of control and the operations performed for the execution of each instruction, will be helpful. The numbers within circles designate commands, and the numbers in squares, the operations called for by those commands. Commands 1 through 5 have already been discussed. Commands 11, 13, 15, 16, 14, 12, and 10 designate the entry points to the microprograms of the instructions A m, cA m, S m, U m, T m, C m, and Z, respectively. Every command designates not only what micro-operation will be performed next, but also to what command control will be transferred after completion of the micro-operation. The operation designated by each of the various operation codes is shown in Table 7.7.

The operations in Table 7.7 indicate use of a relatively simple arithmetic unit containing only a register R, an accumulator A, and an adder–subtractor unit. This restriction has been made only to simplify the description. Also, the switching system in the arithmetic unit may be designed either to permit a large variety of commands, or it may be restricted. It would seem preferable to have the more flexible system for a computer with a large instruction repertory, since then fewer commands would be required, in general, in a given microprogram. Similar remarks apply to the degree of flexibility to be provided when designing the switching system for the control register unit.

TABLE 7.7

Operation code	Operation
20	Transfers the operand, whose address is specified by I_a, from the main store to the register R in the arithmetic unit
21	Adds the contents of register R to the contents of the accumulator A
22	Transfers a word from the location in the main store specified by I_a to the accumulator A
23	Subtracts the contents of register R from the contents of the accumulator A
24	Sets the flip-flop F_t if the sign bit of the accumulator is 1.
25	Copies the contents of the accumulator A into the location in the main store specified by the address in I_a

The manner of execution of the microprograms for instructions A m, cA m, S m, and C m should be clear from reference to Fig. 7.8. Note that in the case of T m, control is transferred to the same place, (command 2), if the sign of the accumulator is negative, indicated by $F_t = 1$, as if a U m instruction were being executed. If the sign is positive, control is advanced to command 1. The only other instruction calling for some comment is instruction Z. If the flip-flop F_z is in the reset state, the only effect of command 10 is to cause control to be transferred back to itself. This effectively puts the computer in a blocked or "dynamic stop" state. The computer can be taken out of this state by activation of a restart switch, which sets the flip-flop F_z. Then control will be advanced to command 1.

All micro-operations will not, in general, take the same length of time to perform. For example, even in a parallel computer it may not be possible to reduce the carry propagation time in an adder to the point where an addition requires the same time interval as a transfer. Other operations, too, notably transfers between the main store and external equipment, may take many times the interval required for an ordinary command. Therefore, for longer micro-operations than the normal, the sequence of operations in the microprogram must be interrupted. One way of doing this for a long command is as follows. Associate a flip-flop, F_j, with the command and use the flip-flop output as a conditional input to table T_b. The flip-flop will be set only by the completion of the command being executed. As long as it is not set, the output of table T_b will be the address of the current command and, consequently, the contents of register M_2 are not altered. Upon completion of the command, F_j is set, allowing control to be advanced to the next com-

mand in the microprogram. The advantage of this arrangement is that it does not require any modification of the circuits supplying input signals to table T_a. Note that this type of operation is similar to that for recirculating the stop command (see Fig. 7.8).

The technique described in the preceding paragraphs is best adapted to a parallel type of computer. However, a serial computer may be designed along the same lines. The pertinent differences are as follows: In a parallel computer with an asynchronous arithmetic unit, every gate requires only one kind of waveform to operate it and the timing of that waveform is not critical. In a serial computer, different gates require different waveforms, the same gate may require different waveforms at different times, and all these waveforms must be critically timed. These complications may be handled by including in the micro-control unit a third function table, T_c, for selecting the appropriate waveform for each command. The main waveform routed by the decoding function table, T_a, opens a gate which is fed by a waveform selected by table T_c. This enables a waveform of correct duration to be applied to any selected gate in the arithmetic or control sections of the computer.

The early mechanization of the microprogramming technique was either by the use of accessible diode matrices, which are easily rewired, or plugboards. It is possible, however, to store the micro-instructions in a high speed random access storage unit. This allows new instructions to be generated even during the execution of a program, e.g., under the control of problem parameters. Thus, any given computer may have whatever instruction code its user desires, and even this may be changed for different programs, or during the course of a program.

7.5.8. Program Interrupt Control

Where a machine is used as an on-line data processor, it may be desirable to provide a means of interrupting the normal advance of control through the stored program in response to signals received from external sources. In an early form of mechanization, referred to as program interrupt control, the program being executed was interrupted whenever a signal was received from a source of input data indicating that it was ready to transmit data for storage in the computer's internal store. For computer systems applied to control of industrial processes, this concept can be expanded to allow the computer to respond to demand signals from different sources on a priority basis. Each source of such a signal is assigned a priority number in accordance with its role in the over-all program designed for the process being controlled. There

is a program in the main store corresponding to each of these priorities. Whenever an input signal is received, the program currently being executed is interrupted and control transferred to the program called for by the new input signal, provided it has a higher priority than the program being executed. All input demands are scanned in an orderly manner and all demands satisfied sequentially in accordance with their priorities.

A priority interrupt control feature is particularly useful in industrial control applications where it is important that a computer responds automatically and after a minimal delay to various critical situations indicated by off-limit values of variables being monitored or by other indicators such as the actuation of a control switch. It is similar on a programmed level to the operation of a dc coupled asynchronous system on a circuit level in that each operation is initiated by completion of the preceding one. In this case one of several programs which can be considered to be in parallel will be entered upon completion of a program executed in response to a demand of higher priority. This allows a machine's time to be utilized to the fullest and also allows a variety of demands to be satisfied without the necessity of assigning in advance time intervals for their performance. A wide variety of demands can then be accommodated within an over-all program design. Included in the hierarchy of programs may be those for the accumulation of signals from a clock, the execution of self-checking routines, special functions which may be demanded occasionally by manually-operated switches to which priorities have been assigned, the scanning of analog readings of various measuring devices and their conversion to digital form, etc.

Priority interrupt control is generally useful for efficient asynchronous use of a computer. For example, consider the case where a computer incorporated in a control system is idle for periods of varying duration throughout the day. In this case, programs for the solution of other problems can be made available for call-in and solution during these otherwise idle times by being assigned priorities lower than those of any of the demands imposed by the system being controlled.

The inclusion of a priority interrupt system does not markedly affect the complexity of the control unit. Briefly, it must provide for relatively simple operations of a bookkeeping nature in order to keep track of the demand currently receiving attention, and the priorities of the demands awaiting satisfaction. The priority interrupt control circuits operate concurrently and independent of other operations. Their main functions are to inspect periodically the status of stations at which demand signals may be present and to cause interruption of the program currently being executed whenever a higher level demand is detected. Upon detection of this higher priority demand, the control unit must cause a transfer of

control from one program to another subject to various constraints including the following: (1) the interruption is to be deferred until completion of the present instruction, and (2) before control is actually transferred to the entry of the new program, the return address to the next instruction that would have been obeyed (had the interruption not occurred) is planted in a specified location so that it may be used on exit from the new program to return to the one interrupted. Other special operations may also be desirable, depending on the nature of the particular computer in which the priority interrupt system is incorporated, and the degree of sophistication of the interrupt system.

7.6. Logical Designs of General Purpose Arithmetic Computers

Earlier in this chapter certain broad principles involved in the design of a general purpose arithmetic digital computer were considered. In Sections 7.6.2 and 7.6.3, the logical designs of two specific computer systems will be derived. Since there are a great many variables entering into the design of a digital computer, the total number of different designs possible is enormous. The criteria for the two particular designs described were chosen on the following bases. First of all, the instruction repertory was restricted to the seven basic single-address instructions described in Chapter 2. The inclusion of more instructions would not have contributed materially to the purpose of instruction, but would have added appreciably to the complexity of detail and perhaps even obscured fundamental points. To indicate specifically the influence of the type of main store on over-all computer design, both major types of main stores are considered—a parallel access static store in Section 7.6.2 and a serial access dynamic store in Section 7.6.3. In both cases, the size of the main store chosen was dictated by considerations of simplicity. Although the word lengths were chosen to be 16 bits in one machine and 32 in the other, the format of numbers and instructions in both machines is similar. In the machine with a static main store, the arithmetic unit is not described in detail because the machine's logical design is such that any of a number of arithmetic units, described in Chapter 6, could be used. Although synchronous operation is implied in the description of this machine, it could readily be adapted, in a manner to be described, to asynchronous operation. The machine with the dynamic main store must be a synchronous machine. In its design, advantage is taken of the opportunity to demonstrate certain ways to minimize the requirements for active storage units. For example, it is shown how the arithmetic and control functions can be achieved by means of circulating registers plus a small amount of active storage and

switching circuitry. Also, extensive use is made of the technique of time-sharing which is defined and described in Section 7.6.1.

7.6.1. TIME-SHARING

"Time-sharing" refers to a way of organizing the various operations to be performed by a machine in such a way that more efficient utilization of a storage element, such as a flip-flop, is obtained. It is best described by means of an example. Assume that a major operating cycle of a system is divided into a number of periods, say n, by time markers, t_0 through t_n. Assume that a flip-flop must be set by a signal W at time t_i, and reset by a signal X at time t_{i+a}, where $i < (i + a) < n$. Assume, too, that another flip-flop is to be set by a signal Y at time t_{i+b}, and reset by a signal Z at time t_{i+c}, where $(i + a) < (i + b) < (i + c) < n$. All of these requirements may be met by two flip-flops P, Q, having the following input equations

$$p = Wt_i \qquad \bar{p} = Xt_{i+a}$$

$$q = Yt_{i+b} \qquad \bar{q} = Zt_{i+c}.$$

If the periods during which each of the flip-flops P, Q, is used to control other circuits do not overlap, P and Q may be replaced by a single flip-flop R, with the following input equations

$$r = Wt_i + Yt_{i+b} \qquad \bar{r} = Xt_{i+a} + Zt_{i+c}.$$

Note that though a storage element (a flip-flop) is saved, two logical operators (OR gates) have been added.

There are two distinct ways of employing the time-sharing technique: (1) For a system with a given major cycle of operation, an inspection may be made of some given estimate of flip-flop requirements for the purpose of detecting whether separate flip-flops are being used for nonoverlapping functions as described above. Even if such a situation does not exist, it may often be forced by the designer through some minor changes in the operation of parts or all of the system (e.g., trial and error changes in various switching signals, in order to produce a larger percentage of input signals that are nonoverlapping). (2) This is a more fundamental method, and affects the basic design of a system. It consists of increasing the period of operation of the machine, defining new subintervals of time, and specifying that functions which might normally be performed concurrently be performed in different subintervals. Therefore, a single flip-flop may be used for several functions during a single major period. The extent of the time-sharing employed is limited by the speed requirements of a system.

Though the term time-sharing has been given to the technique de-

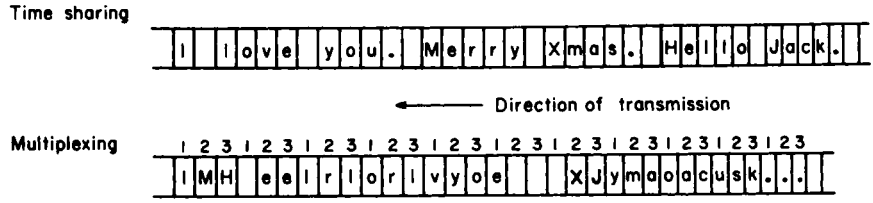
scribed, the reader will readily appreciate that no new concept is involved here. The first procedure recognizes that it is wasteful to use two transmission channels of equal capacity for the transmission of two messages in an interval Δt, if they can both be transmitted over one channel within the same interval. The second recognizes that, if a given message can be transmitted over a channel of bandwidth α in a time interval Δt, then the same message can be transmitted over a channel of bandwidth α/n (where $n > 1$) if a time interval $n\Delta t$ is allowed.

Often, in descriptions of digital computer design techniques, multiplexing is included under the heading of time-sharing. A distinction should be made. Consideration of a transmission channel will illustrate the difference: Time-sharing improves channel utilization by eliminating dead times, i.e., by minimizing the time intervals when the channel is not being used. Multiplexing, too, provides efficient channel utilization. However, instead of serially transmitting different whole messages, it samples corresponding bits of each message sequentially. For example, the second bit of the first message is not transmitted until the first bits of all the messages have been transmitted. At the receiving station, each message is reconstructed by diverting onto a separate path all bits of a particular message. The entry to each channel may readily be controlled by means of a gate with a timing signal input. Example 7.2 illustrates the basic difference between time sharing and multiplexing on an information channel. However, it is incomplete* in that it does not show why or how one method may be preferable to the other in the design of a digital computer. Examples of time-sharing are given in the design of the general purpose computers described in Sections 7.6.2 and 7.6.3. One example of multiplexing is provided in a digital differential analyzer, wherein the bits of the Y and R registers could be stored alternately, on one channel, at the cost of doubling the time required to process a single operational unit (see Chapter 8).

Example 7.2

Messages to be transmitted
One: I love you
Two: Merry Xmas
Three: Hello Jack

* Also, it does not take into account the repetitive nature of the signals generated by the flip-flops used for control purposes in a computer. There would, of course, be little point in transmitting these three messages repeatedly, whereas in a computer it is essential for control information to be maintained and/or generated as long as the system is in operation.



When there is time sharing or multiplexing of storage channels or active storage elements, distinct timing signals are associated with each term in the Boolean algebraic expressions of the read, record signals, or flip-flop input signals. Inspection of these equations, therefore, indicates what takes place at any given time. However, to satisfy some criteria of mechanization, like reducing the number of gating levels, or equalizing the load on certain variables, the original equations may be manipulated to yield forms that satisfy these criteria. Since the meaning of a rearranged expression may not be as apparent as that of the original, it is useful for the sake of clarity to list both in a description of the machine.

7.6.2. The Logical Design of a GP Computer with a Static Main Store

For convenience, we choose as the instruction repertory for this machine, the list of instructions described in Table 2.1. For ready reference, the instructions and their codes are listed again

Code	Instruction
cA m	Clear the accumulator, then add (m) to it
A m	Add (m) to the accumulator
S m	Subtract (m) from the accumulator
C m	Copy the contents of the accumulator into the main store
U m	Unconditional transfer
T m	Conditional transfer
Z	Stop

Other specifications are as follows: (1) All transfers of information within the machine are effected in parallel. (2) The main store is of the random access type with a capacity of 1024 words. (3) The length of words is 16 bits. (4) All numbers, x, used in the computer are normalized to lie in the range $-1 \leqslant x \leqslant (1 - 2^{-15})$. Negative numbers will have a 1 in the sign position and be in a two's complement form. (5) The same word length is used both for storage of a number or an instruction. The

format of a word representing a number is shown in Fig. 7.9(a).

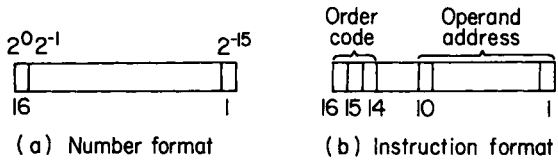


FIG. 7.9. Word format for a number (a) and an instruction (b)

Position 1 holds the least significant bit, (2^{-15}), and the sign bit is in position 16. The format of a word representing an instruction is shown in Fig. 7.9(b). The three-bit group (positions 14, 15, 16) is used to store the order codes. The ten-bit group (positions 1 through 10) is reserved for the address (from 1 to 1024) in the main store of the operand designated in all instructions, with the exception of the Z instruction.

We will now consider some general requirements for the control unit. It is assumed that, after a program has been placed in the main store, and upon activation of the computer, the control unit will cause the instruction stored in memory location 1 to be located first and subsequently will obtain instructions from consecutively numbered locations within the main store. The only exception to this procedure will occur in the event that a U m or a successful T m instruction is encountered. The other major function of the control unit is to initiate and assure completion of the sequence of operations necessary to execute the instruction obtained from the main store.

These two principle functions of the control unit, namely, causing reference to be made to a specified address in the main store and causing the actual execution of instructions so obtained, can be achieved by the use of flip-flops and switching networks. Specifically, the control unit will be comprised of ten flip-flops, C_1 through C_{10}, which we shall refer to as the control register, and four flip-flops, I_1 through I_4, which we shall refer to as the instruction register. The C flip-flops will serve two functions, namely as a storage selection (i.e., address) register (of both instructions and operands), and also as a program counter. When acting as an address register, the control register selects a word from the proper address in the main store by specifying two coordinate numbers, X and Y (see Fig. 7.2). The Y address is in positions 1 through 5 of an instruction word and the X address in positions 6 through 10. When an instruction word is obtained from the main store, the Y address is placed in flip-flops C_1 through C_5 and the X address in C_6 through C_{10}. These flip-flops are used as inputs to an X and a Y matrix, both of which are many-to-one function tables

(see Chapter 4). Consequently, one output line of both the X and the Y matrix is activated. Through the use of additional circuits (not shown) this selects a corresponding X, Y storage location in each plane for the purpose of either recording or reading. Details of a procedure for selecting all the bits of a word in parallel from a static store are provided in Chapter 5. Since a word in this machine has 16 bits, there are 16 planes and 16 output lines in the memory. When acting as a program counter, the control register functions as follows: The execution of each instruction causes its contents to be advanced by 1, so that it then indicates the next consecutive address in the main store from which an instruction word is to be obtained. The flip-flops I_1 through I_3 receive the order code from positions 14 through 16 of an instruction word at the same time that the operand address is received by the flip-flops $C_1 \ldots C_{10}$.

To summarize, upon activation of the computer, the control register will automatically be set to 1. By design, this will cause the operand address and order code of the instruction word stored in address $X = 0$, $Y = 1$, of the main store to be placed in the control register and the instruction register, respectively. The contents of the control register will then control the transfer of information to or from the specified address depending upon whether a recording or reading operation is specified by the order code in the instruction register. At the same time, a specified sequence of operations necessary to execute an instruction is generated in accordance with the contents of the instruction register. How this is accomplished will be described after considering the nature of certain other registers in the machine.

There are two registers in the arithmetic unit. One register, comprised of flip-flops, A_1, A_2, . . . A_{16}, accepts and temporarily stores the result of an arithmetic or logical operation. For example, if two numbers in the main store are to be added, the instruction cA would be used to bring one of these numbers into the register and then the instruction A(m) would cause the second number to be added to the first and the result placed in the register. In the execution of the latter operation, the flip-flops A_1, A_2, . . . A_{16} and another group of flip-flops, R_1, R_2, . . . R_{16} are used as the inputs to an adder. The output of the adder is stored in the flip-flops A_1, A_2, . . . A_{16}, replacing their previous contents. The flip-flops A_1, A_2, . . . A_{16} are referred to as an accumulator because the result of an operation stored there may, in turn, be operated upon to form a new result and, in general, the results of successive operations may be accumulated before the contents are transferred.

The other register in the arithmetic unit is referred to as the R register. It is comprised of flip-flops, R_1, R_2, . . . R_{16} which have already been referred to in connection with one of their major functions, namely to

store an operand selected from the main store in a form which can be used as an input to a switching circuit such as an adder. Briefly, it holds one of the operands when either of the instructions A(m) or S(m) is to be executed. As a matter of convenience, the transmission of a word from the main store will always be by way of this register. For example, when a word is to be transferred from the main store to the accumulator, it will first be transferred to the R register and from there to the accumulator. The R register is also used in conjunction with the control register as follows: One clock period after the R register receives an instruction word from the main store, the order code (in $R_{14}R_{15}R_{16}$) and the operand address (in $R_1 \ldots R_{10}$) are transferred to $I_1I_2I_3$ and $C_1 \ldots C_{10}$, respectively. Simultaneously, the contents of $C_1 \ldots C_{10}$ (the address from which the current instruction was obtained) are transferred to $R_1 \ldots R_{10}$. The latter transfer temporarily stores in the R register the address from which the current instruction was obtained until the contents of the control register have served to initiate a transfer of information to or from the main store. At that time, the address fhom which the current instruction was obtained can be retransferred from the R register to the control register so that the latter, acting as a program counter, can form the address of the next instruction to be executed.

Now that the nature of the control unit and arithmetic unit have been outlined, we will consider the specific requirements for the execution of the various instructions. Because execution of each instruction usually requires the performance of a number of operations, each instruction can be considered as a set of elementary commands. Though these commands differ in detail, all of them fall into one of two main categories, namely those that cause the transfer of information from one part of the computer to another, or cause information from two or more sources to be combined. Both types of operations can be performed by means of switching networks. A description of the machine's instructions in terms of more elementary commands is given in Table 7.7.

Before considering further the requirements for execution of particular instructions, we will consider what commands are required in the execution of any instruction. Three of these are: (1) A command which causes an instruction word (whose location is specified by the contents of the control register) to be transferred from the main store to the R register. (2) A command which causes the operand address part of the instruction word stored in $R_1 \ldots R_{10}$ to be transferred to $C_1 \ldots C_{10}$. (3) A command which causes the order code part of the instruction word, in $R_{14}R_{15}R_{16}$, to be transferred to $I_1I_2I_3$. The first of these three commands causes the instruction word specified by the control register to be obtained from the main store. The second supplies the control register with information

TABLE 7.7

Instruction Code	Commands
cA m	(a) Transfer the contents of the selected word to the *R* register (b) Transfer the contents of the *R* register to the accumulator
A m	(a) Transfer the contents of the selected word to the *R* register (b) Add the contents of the *R* register to the contents of the accumulator
S m	(a) Transfer the contents of the selected word to the *R* register (b) Add the complement of the contents of the *R* register to the accumulator
C m	Transfer the contents of the accumulator to the selected word in the main store
U m	Cause the next instruction word to be selected from memory location m
T m	(a) If the contents of the accumulator are negative, cause the next instruction word to be selected from memory location m (b) If the contents of the accumulator are not negative, carry out the operations necessary to advance to the state at which the program counter advances by a count of 1
Z	Stop, i.e., idle until the computer is activated again

necessary for it to cause information to be transferred into or out of the indicated storage location. The third supplies the instruction register with information which it uses to cause a particular sequence of commands to be obeyed, according to the instruction that is to be executed. The first of these two commands can be referred to as an instruction look-up command, and the latter two as set up commands for instruction execution, i.e., they set the control and instruction registers of the control unit to states which initiate the correct sequence of commands required to execute a specified instruction.

Other commands used to facilitate operation of the control unit are: (4) A command which causes the address in the control register to be transferred to the *R* register at the same time the operand address is transferred to the control register. (This command is required so that while the control register is being used to select a specified operand address in the main store, the address from which the current instruction was selected is not lost.) (5) A command which, after an operand address has been selected, returns to the control register (from the *R* register) the address of the current instruction being executed.* The control register

* This command is identical in its action to command (2), and therefore is not listed separately in Table 7.8.

will then be in a position (except in the case of U m, successful T m, or Z instruction) at the completion of execution of the current instruction, to obey command (6) which is described next. (6) A command which causes the contents of the control register to be advanced by a count of 1, thereby enabling the next instruction word to be selected from the main store.

For ease of reference, the five different commands just described will be referred to by command numbers. The numbers and an abbreviated statement of the action of each command are shown as the first five entries in Table 7.8.

TABLE 7.8

Command	Action
(1)	Transfer $(M_{xy})^*$ to $R_1 \ldots R_{16}$
(2)	Transfer $R_1 \ldots R_{10}$ to $C_1 \ldots C_{10}$
(3)	Transfer $R_{14}R_{15}R_{16}$ to $I_1I_2I_3$
(4)	Transfer $C_1 \ldots C_{10}$ to $R_1 \ldots R_{10}$
(6)	Add one increment to $C_1 \ldots C_{10}$
(7)	Add (R) to (A)
(8)	Transfer (R) to A
(9)	Add $(\bar{R})$ to (A)
(10)	Transfer (A) to M_{xy}

The additional commands required for the execution of specific instructions will now be described. First, note that command (a) of instructions A m, cA m, and S m are all alike and equivalent to command (1) in Table 7.8. In addition, provision must be made for the execution of part (b) of each of these three instructions. This calls for commands (7), (8), and (9), shown in Table 7.8. Instruction C m requires command (10), also shown in Table 7.8. The nine different commands listed in Table 7.8 permit the transfers of information between registers, and the arithmetic operations required for the execution of all instructions specified.

Now we can return to a further description of the operation of the instruction register. First, we will comment on why the instruction register has four stages which can indicate 16 different states, as shown in Table 7.9, when only three stages are required to distinguish seven different instructions, and each of the seven codes could be used to initiate at the

* M_{xy} refers to the storage location specified by the current contents of the x and y selection matrix.

proper time all elementary commands required for the execution of a given instruction.

TABLE 7.9. Defined states of the instruction register

I_4	I_3	I_2	I_1	Configuration
0	0	0	0	S_0
0	0	0	1	S_1
0	0	1	0	S_2
0	0	1	1	S_3
0	1	0	0	S_4
0	1	0	1	S_5
0	1	1	0	S_6
0	1	1	1	S_7
1	0	0	0	S_8
1	0	0	1	S_9
1	0	1	0	S_{10}
1	0	1	1	S_{11}
1	1	0	0	S_{12}
1	1	0	1	S_{13}
1	1	1	0	S_{14}
1	1	1	1	S_{15}

The reasons will be apparent from the following description of how the 16 states are utilized. First of all, seven of the states, as defined by I_1, I_2, and I_3 are used to indicate which of the seven instructions is about to be executed. The remaining nine states are used to indicate intermediate points. These intermediate points are simply the points between the successive commands used in the generation of each instruction. Thus, the instruction register acts in a way similar to the microcontrol unit described in Section 7.5.7. The code of an instruction placed in I_1, I_2, I_3 specifies the initial address of the microprogram of commands to be used in the execution of that instruction. Specifically, the instruction code placed in I_1, I_2, I_3 causes the first command, or commands if simultaneous operation is possible, to be executed. During the period of execution, the instruction register can be set to an intermediate state which is a function only of its preceding state. Similarly each new state of the instruction register can be used both to cause particular commands to be executed and to set the instruction register to another state. Any state of the instruction register causes a unique change in its contents. However, there are some intermediate states which could have been produced by any one

of several preceding states. Such states, e.g., S_{12}, S_{14}, S_{15}, are called common states because the execution of any instruction requires that the instruction register pass through these states.

A schematic indicating successive states assumed by the instruction register as well as different commands obeyed in the execution of each instruction is shown in Fig. 7.10. The symbols in the circles indicate con-

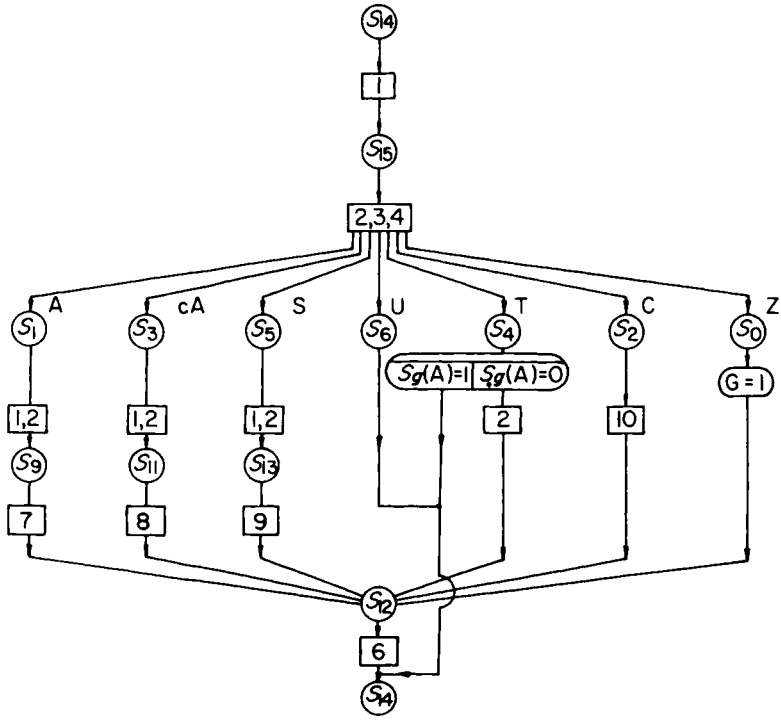


FIG. 7.10. Flow diagram of instruction execution

figurations of the instruction register. The numbers in the rectangles designate which commands are performed during the interval from the configuration above to that below. Only one clock period is required for the execution of any or all commands designated in a given box (with the exception of commands (7) and (9), for reasons to be explained). Also, the time interval between circles on a line is one clock period whether or not there is an intervening box, since the contents of the instruction register can be modified at the same time a command is being executed. The time required for the execution of each instruction, in terms of number of clock periods, is shown in Table 7.10.

TABLE 7.10

Instruction	Look-up and set-up time (i.e., from S_{12}, to S_0, S_1, S_2, S_3, S_4, S_5, or S_6)	Execution time (i.e., from S_0, S_1, S_2, S_3, S_4, S_5, or S_6 to S_{12})
cA m	3 Clock periods	2 Clock periods
A m	3	–
S m	3	–
C m	3	1
U m	3	1
T m	3	1
Z	3	–

The execution time for the stop instruction is not listed in Table 7.10 since its execution time does not fall strictly into the definition listed above, because all that is required for its execution is that the instruction register be in the state S_0. Of course, when the computer is set to an active state manually by means of a switch on the control panel (in a manner described in a succeeding paragraph), one clock period is required for the transition from S_0 to S_{12}. The execution times for the instructions A m and S m have not been specified because the times required to form the sum or difference of two numbers of given length depends on how we design the adder which is incorporated into the computer. This is subject to great variability, as indicated in Chapter 6. The adder may be serial and/or synchronous, or it may be serial–parallel and/or asynchronous. The choice hinges primarily on the price in equipment complexity one is willing to pay for increased speed. However, for the purposes of the present discussion, we need not be concerned with the internal logic of the adder. If it is synchronous, the time required for an addition or subtraction will be constant and predictable, and a timing signal can be provided to change the instruction register from configuration S_1 or S_5 to S_{12} at the completion of an addition or subtraction, respectively. If the adder is asynchronous, a signal may be derived from it indicating when the addition or subtraction is complete. This completion signal is then used to advance the instruction register. Such signals will be designated by the notation E_t in the Boolean equations describing the machine's logical structure.

Now that the general mode of operation of the computer has been considered, we will describe the input signals to the flip-flops in the instruction and control registers of the control unit and the accumulator and R register of the arithmetic unit. These flip-flops are all specified to be of the

T type. The input equations for flip-flops, I_1, I_2, I_3, I_4 can be derived by examination of Fig. 7.10 which shows all allowable configurations, as well as transition paths from one configuration to another, of the instruction register. These input equations are listed below in reduced form

$$i_1 = I_4 I_2 + (E_{i-m} + E_{i-m+1} \ldots + E_i + E_{i+1} + \ldots + E_n) + I_3 I_2 I_1 \bar{R}_{14}$$

$$i_2 = \bar{I}_3 I_2 (I_4 I_1 + \bar{I}_4 \bar{I}_1) + I_3 \bar{I}_2 \bar{I}_1 (I_4 + A) + I_3 I_2 I_1 \bar{R}_{15}$$

$$i_3 = \bar{I}_3 (I_4 + G\bar{I}_1) + I_3 I_2 I_1 \bar{R}_{16}$$

$$i_4 = \bar{I}_4 G + I_3 I_2 I_1.$$

The input signals to the control register may be derived simply by considering when and from what sources it receives input information, and when its contents have to be modified according to some prescribed rule. The preceding description of commands required by the computer shows that for the execution of command (2), which takes place if any of the configurations S_{15}, S_1, S_3, S_5, or $S_4\bar{A}$ exists (see Fig. 7.10) the control register must copy the contents of the R register. The other change required in the contents of $C_1 \ldots C_{10}$ occurs when the configuration S_{12} exists, at which time the contents of the control register should change by a count of 1 (achievable by execution of command (6)). Accordingly, the input equation to each flip-flop of the control register is of the form

$$c_i = (\bar{R}_i C_i + R_i \bar{C}_i)(S_{15} + S_1 + S_3 + S_5 + S_4\bar{A}) + C_{i-1} C_{i-2} \ldots C_1 S_{12}$$

for $i = 1, 2, \ldots 10$.

Because we are not specifying the type of adder to be used in this machine, the input signals, if any, to the R register when it is functioning as part of the adder will not be considered. Exclusive of this the R register can receive input information from two sources: the memory and the control register, in the execution of commands (1) and (4), respectively. Accordingly, the input equation to each flip-flop of the R register is of the form

$$r_i = [(\bar{M}_c)_i R_i + (M_c)_i \bar{R}_i](S_{14} + S_1 + S_3 + S_5) + (\bar{C}_i R_i + C_i \bar{R}_i) S_{15}$$

for $i = 1, 2, \ldots 10$

$$r_j = [(\bar{M}_c)_j R_j + (M_c)_j \bar{R}_j](S_{14} + S_1 + S_3 + S_5)$$

for $j = 11, 12, \ldots 16$.

Because we are not specifying the type of adder to be used, we will also neglect the input signals to the accumulator when it is functioning as part of the adder. Exclusive of this, the accumulator must be provided with

means to accept information from the R register (for the performance of command (8)). For this function the input equation to each flip-flop of the accumulator is of the form

$$a_i = (\bar{R}_i A_i + R_i \bar{A}_i) S_{11}$$
$$\text{for } i = 1, 2, \ldots 16.$$

We will now consider how the computer is started, i.e., set to an active computing state, or placed in an inactive or idle state. A single flip-flop, G, can be used for this purpose. When it is in the 1 state, the computer is defined as being in the active computing state. When the main power is switched on, the flip-flop G is first set to the idle state by means of a reset switch located on the control panel (a set switch also being provided). The computer is always set first to the idle state in order to prevent it from initiating the execution of instructions before a complete program has been inserted in the main store and checked, and the control circuits set to the desired initial conditions. Inspection of Fig. 7.10 shows also that once the computer has been set to state S_0, as a result of the execution of a stop instruction, it will remain in that state until flip-flop G is set to 1. (It is assumed that the I register is always set initially to the state S_0). The input signals to the flip-flop G (which is of the R–S type) are

$g = s$	Activation of the set switch
$\bar{g} = r$	Activation of the reset switch
$+ \bar{R}_{14}\bar{R}_{15}\bar{R}_{16}S_{10}$	Indication that a stop instruction is about to be executed, i.e., the I register will be put into configuration S_0 at the next clock pulse.

The reason for the signal $\bar{R}_{14}\bar{R}_{15}\bar{R}_{16}S_{10}$ rather than S_0 (which would be incorrect) becomes apparent when it is recalled that for the particular type of flip-flop being described, an effective delay of one cycle exists between the time it receives an input signal and the earliest time at which it can use that information to control a gating signal.

In conclusion, a few brief comments on the filling and initial setting of the computer, and the disposition of computed results. In the Appendix on input–output equipment there are descriptions of a number of devices that can be used to insert data into a computer. Essentially what the computer must provide for this function are signals to start and stop the input device and a buffer register to accept specified amounts of input data and from which this data can be transferred either to the arithmetic unit for

any required pre-storage transformation or directly to the main store. All that is required for initial setting of the machine is a set of clear switches on the console which upon activation transmit signals to the reset inputs of the appropriate flip-flops: $I_1, \ldots I_4$, C_0, $C_1, \ldots C_9$, R_0, $R_1, \ldots R_{15}$, and A_0, $A_1, \ldots A_{15}$. The results of computations can be made available in any of a number of forms described in the Appendix. Here provisions must be made similar to those for input data, namely: signals to start and stop the output device and buffer registers between the computer and the output device. With both input and output devices, the buffer register(s) may be either in the auxiliary device, the computer, or both, depending on the nature of each.

7.6.3. The Logical Design of a GP Computer with a Dynamic Main Store

The instruction repertory of the computer to be described next is practically the same as that of the machine described in Section 7.6.2. The only exception is that the stop instruction, Z, is replaced by a conditional stop or break point instruction Z_b, which reads: "If switch S_i is set, stop; otherwise continue." The S_i refer to a set of five two-position switches on the control panel, of which only one may be set before initiating or continuing execution of a program. Each bit of the address field used for one coordinate (the track number) of the operand address in other instructions is used in a Z_b instruction to refer to a particular break point switch. During execution of a Z_b instruction, the break point switches are inspected, and if the one designated in the address has been set, the machine is stopped, i.e., put into an idle state. The conditional stop instruction facilitates checking out a new program since it allows stops to be programmed at convenient points, while at the same time not requiring removal of these stop instructions from the program after check out. Deactivation of the break point switches on the control console effectively removes the stop instructions from the checked out program.

Other specifications are as follows: (1) All transfers of information within the machine as well as arithmetic operations are performed serially. (2) The main store is either a magnetic drum or disk memory with a capacity of 1024 words. (3) The length of words is 32 bits. (4) All numbers, x, used in the computer are normalized to lie in the range $-1 \leqslant x \leqslant (1 - 2^{-31})$. Negative numbers will have a 1 in the sign position and be in a two's complement form. (5) The same word length is used both for storage of a number or an instruction.

Other major differences between this machine and the one described in Section 7.6.2, outside of the different type of main store, will be de-

scribed next. First, the main store will be utilized not only for storage of instructions, problem parameters, and working storage, but also for other purposes. First of all, a set of channels will be provided with permanently recorded data from which timing signals useful for control purposes will be derived. Also, arithmetic and control registers will be mechanized, not by means of flip-flops, but from delay lines formed by appropriate positioning of record and read heads along a channel in the store (see Sections 5.1 and 5.2). Thus, this machine makes extensive use of passive storage elements not only for general storage, but also for information processing and control functions usually obtained by means of active elements. The other feature of this machine not present in the machine with the static main store is an extensive use of time-sharing. However, time-sharing could have been used in connection with the static store computer as well. It is utilized here, in conjunction with the dynamic main store, to emphasize how the requirements for active storage elements can be reduced.

The major functional units of this machine are the main store, the timing channels, the circulating arithmetic and control registers, the logic switching network, and the arithmetic and control flip-flops. The organization of these elements into a computer system is shown in Fig. 7.11.

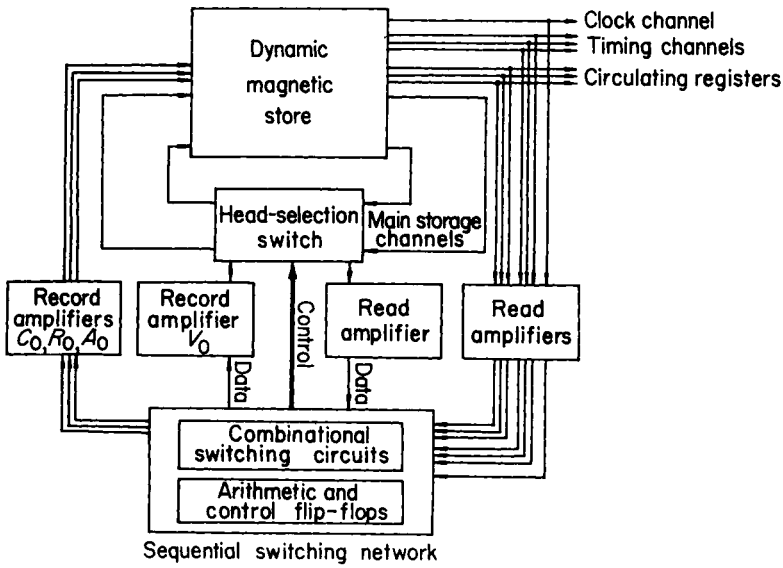


FIG. 7.11. Organization of a GP computer with a dynamic main store

As a matter of convenience, different storage areas in the machine can be classified on the basis of access time. The delay in reading or altering the contents of a flip-flop is at most one bit time, and for a circulating register one word time. The maximum access time to a location in the main store is the period of one revolution, and on the average, half a revolution. By the use of such devices as storage-address interlacing, and other minimum access coding techniques (see Section 7.5.4), the average access time can be reduced to just a few words. To summarize, these different storage areas represent immediate, quick, and slow access storage, respectively. Another class of storage, intermediate between the quick and slow access types, may be obtained by adding a number of circulating loops, each having a length of only a few words. Such loops, sometimes referred to as high speed loops, or revolvers, are useful for the purpose of serving as a working storage area.

The 1024 words of 32 bits each in the dynamic store are arranged in 32 tracks of 32 words each. Ten bits are adequate to specify the address of any of these storage locations, five being used to specify a sector number and five to specify a track number (see Fig. 7.12). The format of a word representing a number is shown in Fig. 7.13(a). The format of a

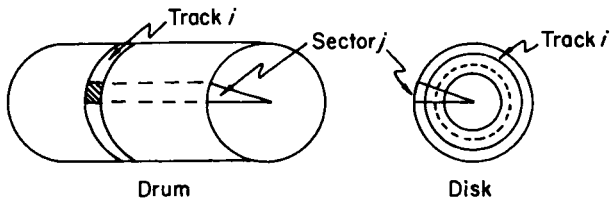


FIG. 7.12. Addressing systems in a magnetic drum or disk store

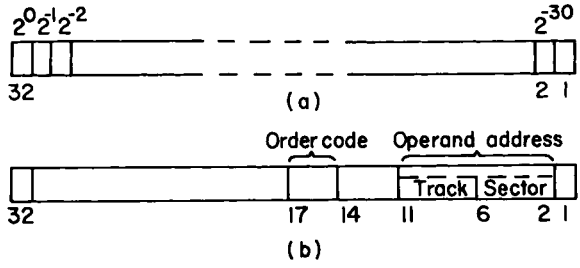


FIG. 7.13. Word format for a number (a) and an instruction (b)

word representing an instruction is shown in Fig. 7.13(b). This arrangement of information in an instruction word is to a certain extent arbitrary, but the utility of it will be apparent after the description of the computer's mode of operation. The positions 1 through 32 indicate the order in which information is read from or recorded in the store. These temporal positions can also be used to indicate spatial positions in a delay line type of store. Position 1 is not used, serving as a buffer zone between adjacent words. (This one-bit period provides an interval in which transients that may be introduced in initiating a recording operation can decay.) Positions 2 through 11 are reserved for the 10 bits of the address code. Positions 14 through 17 are reserved for the three-bit order code which specifies the instruction to be executed. Note that in this case the word length is dictated by the precision required for numbers. Less than half of the 32 bits are used in a word storing an instruction. For a more elaborate machine the unused bits could be used in various ways. For example, two complete single-address instructions could be stored in one word. Also, unused bits could be used for additional purposes such as the address of index registers, and other registers if such facilities were incorporated in the design. Some of the unused bits could also be utilized for order codes and storage addresses if additional instructions and storage facilities were added to the machine.

Since a recording and reading operation cannot occur simultaneously in the main store, a single magnetic head can be used for both purposes. Also, since at most a single word in the main store is referred to during the acquisition or execution of any instruction, it would be wasteful to provide a separate record and read amplifier for each head. Instead, a single record and read amplifier are provided for the main store together with a selection matrix which causes the appropriate amplifier to be connected to the head on the track containing the specified storage location. During the instruction acquisition period, the selection matrix causes the read amplifier to be connected to the address specified by the control register. During the instruction execution period, the selection matrix connects either the read or record amplifier to a particular head in accordance with the order code and operand address of the instruction to be executed. All heads other than the one selected are inactive and have no effect on the information circulating in their associated channels. Whenever new information is to be recorded, it is not necessary to first erase the old because erasure is implicit in the recording operation, i.e., it writes over old information. However, a separate erasure may be effected simply by recording all 0's in any storage location.

The basic source of timing signals in the computer is the clock channel. It has no record head, and a single read head. A uniformly spaced

series of signals, permanently recorded here, are read continually when the machine is in operation. The time interval between successive clock pulses defines, and is referred to as, a bit period. There are, also, three other permanently recorded timing channels, each of which has a single read head only. These channels, designated p_1, p_2, p_3, serve the following important functions: (1) They provide signals indicating time intervals of interest within a word period, namely that reserved for the order code, and the track and sector number of an address in an instruction, and the one bit period reserved for the sign bit of a number. (2) They provide signals indicating the sector number of words in the main store. Information is recorded on the permanent timing tracks as shown in Fig. 7.14.

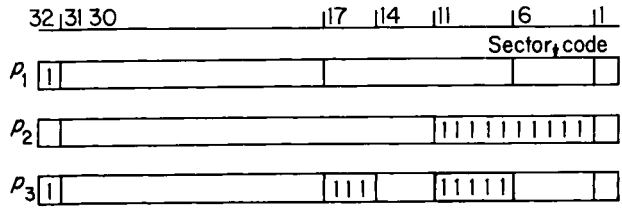


FIG. 7.14. One word length of information on the permanent timing tracks

In positions 2 through 6 of each word period, p_1 contains the sector number of all the words (as many as there are channels in the main store) in the next sector that will pass under the read-record heads, and in position 32, $p_1 = 1$; in all other positions $p_1 = 0$. The signals generated by p_2 and p_3 are the same in every word period. In positions 2 through 11, $p_2 = 1$; elsewhere $p_2 = 0$. In positions 7 through 11, 15 through 17, and in position 32, $p_3 = 1$; elsewhere $p_3 = 0$. Different word periods are distinguished by means of the sector code numbers recorded on channel p_1. The information from channels p_1, p_2, and p_3 are used to derive time duration signals as shown in Fig. 7.15. D_1 and D_2 define the positions of

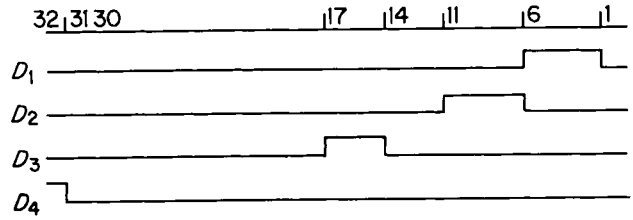


FIG. 7.15. Time duration signals derived from permanent timing tracks

the sector and track addresses, respectively, D_3 defines the position of the order code, and D_4 the position of the sign.

The registers of the arithmetic and control unit are actually short delay lines formed by appropriate placement of record and read heads along tracks of the dynamic store. Though these circulating registers (see Section 5.1) are physically part of the main store, logically they are parts of the arithmetic and control units and function analogously to the way they would if they were static registers. There are three circulating registers, each of a single word length: the control register, the R register, and the accumulator. The function of each of these will be described next.

The control register, C, holds the address of the storage location containing the next instruction to be read and executed. In the one-address machine being described, 1 is normally added to the contents of the control register after each instruction is read. This facilitates obtaining instructions sequentially from consecutively numbered storage locations, corresponding to the consecutively numbered steps of a written program of instructions. After an instruction has been located and read, it must be kept available until its operand has been located. The R register serves this function. Also, if a multiplication and/or division instruction were added to the instruction repertory, the R register could be used to store the multiplicand and make it available for incorporation into the partial products as they are formed, and to store and make available the denominator. The result of each arithmetic or logical operation appears first in the accumulator. This result may then either be transferred to the main store and/or retained for use in the succeeding operation. If a multiplication and/or division instruction were added, the accumulator could be extended to two word lengths. In multiplication, it could be used to hold the multiplier and the partial products. Since the bits of the multiplier can be discarded as the partial product grows, no more than two word lengths of storage are essential for the two numbers. In division, the accumulator could be used to hold the numerator (and subsequently the partial remainders), as well as the quotient. Since neither of these exceeds one word length, a total of two word lengths for the accumulator would be sufficient here, too.

As stated earlier, the logical design of a digital computer defines all the permissible states the computer can assume, as well as the rules governing the transition from one state to another. In the computer under consideration, the next active state is specified primarily by the state of a group of flip-flops, each of which maintains a constant setting throughout a bit period. The active state of the computer during any bit period may be considered as being defined by the following: (1) the current signals from the permanent timing tracks; (2) the current signals from the cir-

culating registers; (3) the current bit read from the main store (though such information is not present if recording is in progress); (4) the current state of the flip-flops. It is clear that each new active state is determined in part by the preceding active state, and in part by information presented by the main store* (including timing tracks and circulating registers). It is the function of the gating circuits that comprise the switching network to transform this input information and produce the following types of signals which will effect a transition to a new state: (1) input signals to the flip-flops; (2) signals specifying information to be recorded in the circulating registers; (3) a signal to the head selection matrix to indicate whether a bit is to be recorded in the main store and if so: (4) the signal to be recorded in the main store; (5) signals to control various output devices, e.g., an electric typewriter, magnetic tape, etc.

The flip-flops, which mainly determine an active state of the computer, represent storage to which there is practically immediate access either for the purpose of reading its current state or causing that state to be altered. The current state of each flip-flop is determined by its preceding state and the current input to it from the logic switching network. In this machine flip-flops are used for the following major functions: (1) They define various phases of operation in which characteristically different operations common to the execution of each instruction are performed. Continuous access to this information is required, since it controls the transition from one major operation to the next. The four phases of operation through which the computer passes in carrying out an instruction are defined in Table 7.11. (2) The flip-flops also store the order code of the instruction to be executed. This information serves to control the process of execution. (3) They indicate the address of the channel in the main store to be selected. Whenever information is to be read from, or recorded in, the main store, the address of the channel to be selected is stored in a group of flip-flops. This information is left undisturbed until the operation is completed, since it must be continuously available to control the selection of the appropriate head by means of the head selection matrix. (4) They hold, from one bit period to the next, a carry bit generated during any addition operation. (5) They compare two groups of bits from different sources, and indicate whether they are identical. Such a function is useful in search operations.

In addition to the functions described, flip-flops are also used for a number of specialized control or gating signals that must be generated during the course of operation of the computer. During times and phases of operations when some of the flip-flop functions already described are

* Also, in some cases, by external inputs.

TABLE 7.11

Phase	Principal operation performed
1	A search is made for the instruction word
2	The designated instruction is read into the R register
3	A search is made for the operand word
4	The designated operand is read and the instruction executed

not required, the same flip-flops may be utilized for these special functions.

Sometimes, more than the minimum number of flip-flops actually required are used to define various states. This can yield certain advantages. For example, the use of a few additional flip-flops makes it possible to reduce the complexity of the input signals to the flip-flops, thereby reducing the number of gating elements required and simplifying the description of the computer's operation. Also, as shown in Section 3.8.1, the use of additional flip-flops can eliminate the gating elements used for a many-to-one function table to decode the contents of a smaller number of flip-flops.

The four phases of operation through which the computer passes in carrying out an instruction will now be described in more detail: During phase 1, a search is made for the instruction word whose address is in the control register. Because of the varying access time to words in the main store, phase 1 may last from 1 to 32 word periods. During time interval D_1 of each word period, the sector number part of the address in the control register is compared with the sector code number from track p_1 by means of a flip-flop, K. The search, and phase 1, are concluded at the end of that word period during which there is complete coincidence. During the time interval D_2 the track number part of the address in the control register is stepped into a shift register comprised of five flip-flops, T_1 through T_5.

Phase 2 has a duration of only one word period, during which two principal events occur: (1) The instruction word at the selected address (which is the next instruction to be executed) is read into the R register This action is effected by means of the flip-flops T_1 through T_5 which control the head selection network. This network allows information to be read only from the head on that track of the main store whose numerical code is contained in T_1 through T_5. (2) In order that control may be advanced automatically to the instruction in the succeeding storage location after the one just selected has been executed, a 1 is added to the address in the control register.

During phase 3, a search is made for the operand whose address is specified in the instruction word which was read into the R register during phase 2. Therefore, the time duration of this phase may vary from 1 to 32 word periods. During time interval D_1 of each word period, the sector number part of the address in the R register is compared with the sector code number from track p_1, by means of flip-flop K. The search and phase 3 are concluded at the end of that word period during which there is complete coincidence. During time interval D_2, the track number part of the address in the R register is stepped into the shift register comprised of flip-flops T_1 through T_5. During time interval D_3 the order code of the instruction word in the R register is stepped into the shift register comprised of flip-flops I_1, I_2, I_3. For instructions Z, U m, T m phase 3 is only one word period in length since a search does not have to be made for a word in the main store.

During phase 4 each instruction is actually executed. Because of the simple nature of the instructions in the computer being described, an interval of only one word period is required to execute any of the instructions. Table 7.12 indicates the operations that are performed for each instruction.

TABLE 7.12

Instruction	Operation performed during phase 4
cA m	The contents of the address specified are copied into the accumulator
A m, S m	The contents of the address specified are added to or subtracted from the contents of the accumulator
C m	The contents of the accumulator are transferred to the address specified
U m	The address specified in the U m instruction is transferred to the control register
T m	Same as U m if the test is successful
Z_b	The computer is put into an inactive or idle state if the break-point condition is satisfied.

In Table 7.13 are listed the order codes, held in flip-flops I_1, I_2, and I_3 and also the information that would be recorded in the circulating registers and the main store during phase 4, for each instruction. C_0, A_0, and M_0 refer to the inputs to the record amplifiers associated with the control register C, the accumulator A, and the main store, respectively. During phase 4 data is never recorded in the R register.

TABLE 7.13. Data recorded in the dynamic store during phase 4 in accordance with the instruction being executed

I_1	I_2	I_3	Instruction	A_0	M_0	C_0
0	0	1	cA m	M		
1	1	0	A m	Sum: $(A+M)$		
1	1	1	S m	Diff: $(A-M)$		
1	0	0	C m	A	A	
0	1	0	U m	A		M
0	1	1	T m	A		M, for successful T m
0	0	0	Z_b	A		

Table 7.14 provides a summary of the functions of the various flip-flops during each of the four phases of operation.

Now that an over-all picture of the machine's organization and operation has been presented, the signals that cause the required actions to take place will be described by means of Boolean algebraic equations. These signals are classified into three main categories: (1) time duration signals that are derived from the permanent timing tracks, (2) input signals to the flip-flops, (3) input signals to the record amplifiers. (For brevity, not all logical and arithmetic functions are described. For example, logic necessary to initially load the central store is not included; also, the clock pulse input to each gate is not shown in these equations).

The time duration signals are as follows:

Signal defining time interval within a word allotted to the sector number part of the address: D_1

$$D_1 = P_2\bar{P}_3 \qquad \bar{D}_1 = \bar{P}_2 + P_3$$

Signal defining time interval within a word allotted to the track number part of the address: D_2

$$D_2 = P_2P_3 \qquad \bar{D}_2 = \bar{P}_2 + \bar{P}_3$$

Signal defining time interval within a word alloted to the order code: D_3

$$D_3 = \bar{P}_1\bar{P}_2P_3 \qquad \bar{D}_3 = P_1 + P_2 + \bar{P}_3$$

Signal defining time interval within a word allotted to the sign bit: D_4

$$D_4 = P_1P_3 \qquad \bar{D}_4 = \bar{P}_1 + \bar{P}_3$$

The input signals to the eleven flip-flops (listed in Table 7.14) will

TABLE 7.14. Summary of functions of flip-flops during the four phases of operation

Flip-flop	Phase 1	Phase 2	Phase 3	Phase 4
K	Searches for instruction whose address is in C register by comparing successive bits from C with the sector codes in track p_1.	Acts as carry flip-flop in the operation of adding 1 to the address in C register.	Searches for operand whose address is in R register by comparing successive bits from R with the sector codes in track p_1.	Acts as a carry (or borrow) flip-flop in the execution of an A m or S m instruction.
T_1, T_2, T_3, T_4, T_5	Receives track number of instruction address from C register.	Controls selection of channel whose address is in C register.	Receives track number of operand address from R register.	Controls selection of channel whose address is in R register.
I_1, I_2, I_3	I_2 indicates whether computer is in an active or idle state. I_1, I_3 not used.		Receives instruction code from R register,	Controls execution of instruction whose code it contains.
F_1, F_2	Define phases 1 through 4 as follows F_1 F_2 Phase 0 0 1 0 1 2 1 0 3 1 1 4			

be described next. As indicated in Table 7.14, the flip-flops, F_1, F_2, define the four phases of operation of the computer. The input signals which advance them from one phase to the next are (Note: in the remaining equations of this section, variables apparently missing have been eliminated by algebraic reduction—e.g., in an expression like $FX + \bar{F}XY$, F is eliminated).

$$f_1 = \bar{F}_1 F_2 D_4$$
$$\dot{f}_1 = F_1 F_2 D_4$$

$$f_2 = F_1 \bar{F}_2 \bar{K} D_4$$

Effects transition from phase 3 to phase 4, provided search for operand has been completed (indicated by $\bar{K}D_4$).

$$+ \bar{F}_2 \bar{K} D_4 I_2$$

Effects transition from phase 1 to phase 2, provided search for instruction has been completed (indicated by $\bar{K}D_4$) and computer is not blocked (indicated by I_2).

$$\dot{f}_2 = F_2 D_4$$

The flip-flop K serves four distinct functions, in accordance with which phase of operations is taking place. This accounts for the many terms in its input equations:

$$k = \bar{F}_1 \bar{F}_2 D_1 (P_1 \bar{C} + \bar{P}_1 C) + F_1 \bar{F}_2 D_1 (P_1 \bar{R} + \bar{P}_1 R)(I_1 + \bar{I}_2 I_3)$$

Indicates a disagreement between corresponding pulses of the sector code track and the sector number designated in the control register C or the register R during search phases 1 or 3 respectively.

$$+ \bar{F}_1 F_2 \bar{C} P_2$$

During the addition of 1 to the address in C during phase 2, state $\bar{K}$ is interpreted as 1 and state K is interpreted as 0. At the beginning of phase 2, the state $\bar{K}$ exists, and the flip-flop is set to state K by the first zero encountered in the address in C. (Thus, K acts like B in Section 6.1.1.5).

$+ F_1F_2MA\bar{D}_4I_1I_2\bar{I}_3$	Indicates production of a carry pulse during execution of instruction A(m) in phase 4.
$+ F_1F_2M\bar{A}\bar{D}_4I_1I_3$	Indicates production of a borrow pulse during execution of instruction S(m) in phase 4.
$\bar{k} = D_4$	Resets the flip-flop K at the end of each word period.
$+ F_1F_2\bar{M}\bar{A}I_1I_2\bar{I}_3$	Indicates no carry pulse has been produced and resets flip-flop K during execution of instruction A(m) in phase 4.
$+ F_1F_2\bar{M}AI_1I_3$	Indicates no borrow pulse has been produced and resets flip-flop K during execution of instruction S(m) in phase 4.

The flip-flops $T_1 \ldots T_5$ are used to hold the address of the track to be selected from the main store for a recording or reading operation. During phases 1 and 3 they act collectively as a shift register, information being stepped into them from the control register or the R register, respectively. During phases 2 and 4 they provide a signal to the head selection matrix. Since there are 32 tracks in the main store, five flip-flops (which have $2^5 = 32$ distinct configurations) are required. The input equations for these flip-flops are

$t_1 = \bar{F}_2D_2(\overline{F_1C + F_1R})$ $\bar{t}_1 = F_2D_2(\overline{F_1\bar{C} + F_1\bar{R}})$	Numbers indicating the tracks to be selected during phases 2 and 4 are stepped into flip-flop T_1 during phases 1 and 3 respectively. T_1 is the entrance to the shift register.
$t_2 = \bar{F}_2D_2T_1$ $\bar{t}_2 = F_2D_2\bar{T}_1$	Receives information shifted out of flip-flop T_1.
$t_3 = \bar{F}_2D_2T_2$ $\bar{t}_3 = F_2D_2\bar{T}_2$	Receives information shifted out of flip-flop T_2.
$t_4 = \bar{F}_2D_2T_3$ $\bar{t}_4 = F_2D_2\bar{T}_3$	Receives information shifted out of flip-flop T_3.
$t_5 = \bar{F}_2D_2T_4$ $\bar{t}_5 = F_2D_2\bar{T}_4$	Receives information shifted out of flip-flop T_4.

The flip-flops I_1, I_2, and I_3 are used to receive and hold the order code of the instruction which is about to be executed. During phase 3 they behave as a shift register, information being stepped into them from the R register, and during phase 4 their contents are used to control the execution of the specified instruction. Since these flip-flops are not required for the above function during phases 1 and 2, one of them, I_2, is used for another purpose, namely to put the counter into a blocked or idle state upon the completion of phase 4 if any of the following conditions exist: (1) An overflow of the accumulator is produced after the execution of instructions A(m) or S(m), indicating an improper addition or subtraction. (2) The one cycle of operations switch, O_1, is in a set condition. This switch enables the computer to be set to an idle state at the end of any cycle of operations so that its contents may be inspected. (3) The presence of the conditional stop instruction, Z_b, and the address upon which it is contingent. The input equations for these flip-flops are

$i_1 = F_1\bar{F}_2D_3R$
$\bar{i}_1 = F_1\bar{F}_2D_3\bar{R}$

Information indicating what type of instruction is to be executed is stepped into I_1 during phase 3. Flip-flop I_1 is the entrance to the shift register composed of I_1, I_2, I_3.

$i_2 = F_1\bar{F}_2D_3I_1$

Receives information stepped out of flip-flop I_1.

$+ F_1F_2D_4(I_1 + I_2 + I_3)\bar{O}_1$

$+ F_1F_2D_4(\overline{T_1S_1 + \ldots + T_5S_5})\bar{O}_1$

If the one cycle switch, O_1, has not been set and the conditional stop instruction, Z_b, is not being executed, the computer is put into an active state at the end of phase 4.

$+ E$

The activate switch, E, being set takes the computer out of an inactive or idle state.

$\bar{i}_2 = F_1\bar{F}_2D_3\bar{I}_1$

Receives information stepped out of flip-flop I_1.

$+ F_1F_2D_4[O_1 + (I_1 + I_2 + I_3)(T_1S_1 + \ldots + T_5S_5)]$

Puts the computer into an inactive state at the end of phase 4 because the one cycle switch, O_1, is set, or Z_b is being executed.

$$+ F_1F_2D_4\,[\bar{K}M(AI_1I_2\bar{I}_3 + \bar{A}I_1I_3) + K\bar{M}(AI_1I_3 + \bar{A}I_1I_2\bar{I}_3)]$$

The computer is put into an inactive state because an overflow has occurred on the execution of A m or S m.

$$i_3 = F_1\bar{F}_2D_3I_2$$

Receives information stepped out of flip-flop I_2.

$$\bar{i}_3 = F_1\bar{F}_2D_3\bar{I}_2$$

Receives information stepped out of flip-flop I_2.

$$+ D_4A\bar{I}_1I_2I_3$$

If the number in the accumulator is negative (indicated by D_4A) the instruction T m is converted to U m.

A few words of further explanation are in order for the third expression in the input equation $\bar{i}_2$. The four terms in this expression describe various conditions which indicate an overflow has occurred. Specifically, in addition, one knows that an overflow has occurred if the sum of two negative numbers produces a positive number, or if the sum of two positive numbers produces a negative number. These two conditions are indicated by the signals $\bar{K}MAD_4$ and $K\bar{M}\bar{A}D_4$, respectively. In subtraction, an overflow has occurred when the subtraction of a negative from a positive number produces a negative number or the subtraction of a positive from a negative number produces a positive number, as indicated by $\bar{K}M\bar{A}D_4$ and $K\bar{M}AD_4$, respectively.

Reference to Fig. 7.11 and the description of the machine's organization shows that there are only four record amplifiers: one for the 32 tracks of the central store and one for each of the circulating registers, C, R, and A. The input signals to these amplifiers, for recording a "1," M_0, C_0, R_0, and A_0, respectively. Recording current must be supplied to the recording circuit of the central store only when a C m instruction is to be executed. Accordingly, the signal used to energize this recording circuit is

$$W = F_1F_2I_1\bar{I}_2.$$

The information actually recorded in the central store is the contents of the accumulator A (see Table 7.13). Therefore,

$$M_0 = WA$$

The record "1" signals for the circulating registers are as follows

$C_0 = \bar{F}_2P_2C$	Recirculates address in control register C, during phases 1 and 3.
$+ F_2\bar{P}_2C$	Recirculates nonaddress information in C during phases 2 and 4.
$+ F_1P_2(I_1 + \bar{I}_2 + I_3)C$	Recirculates address in C during phases 3 and 4 unless instruction U m is about to be executed.
$+ \bar{F}_1F_2P_2(KC + \bar{K}\bar{C})$	Sum of the old address in C plus 1, formed during phase 2.
$+ F_1F_2\bar{I}_1I_2\bar{I}_3R$	Transfers the contents of the R register to the control register C during phase 4, thereby effecting execution of a U m instruction when called for.
$R_0 = \bar{F}_2R$	Recirculates contents of the R register during phases 1 and 3.
$+ F_1\bar{I}_2R$	Recirculates contents of the R register during phases 3 and 4 unless I_2 is true.
$+ \bar{F}_1F_2M$	Receives, during phase 2, contents of word selected from the main store.
$+ F_2I_2M$	Copies, during phases 2 and 4, the contents of the word selected from the main store if I_2 is true.
$A_0 = \bar{F}_1A$	Recirculates contents of accumulator A during phases 1 and 2.
$+ \bar{F}_2A$	Recirculates contents of A during phases 1 and 3.
$+ \bar{I}_1I_2A$	Recirculates contents of A during phase 4 if instructions U m or T m are being executed.

$+ I_1I_2I_3A$	Recirculates contents of A during phase 4 if instruction Z_b is being executed.
$+ F_1F_2I_1I_2I_3M$	During phase 4 copies the contents of the word selected from the main store into the accumulator if instruction cA m is to be executed.
$+ F_1F_2I_1I_2\,[K(AM + \bar{A}\bar{M} + \bar{K}(A\bar{M} + \bar{A}M)]$	During phase 4, records the sum or difference of the number in the accumulator and that in the word selected from the main store, according to whether the instruction to be executed is A m or S m respectively.
$+ I_1I_2A$	Recirculates contents of A during phase 4 if instruction C m is being executed.

7.7. Concluding Remarks

In earlier chapters we have considered various systems of circuit logic (Chapter 4), a description of how groups of these elements, interconnected for the purpose of generating various switching functions, could be conveniently described by means of Boolean algebraic statements (Chapter 3), the characteristics and means of access to large capacity storage systems (Chapter 5), and various schemes for interconnecting storage and gating elements in order to perform arithmetic and logical operations (Chapter 6).

In this chapter we have shown, first of all, some of the basic criteria to be considered in designing a digital computer. Next we have considered some of the problems encountered and various alternatives available in the design of the control unit whose function it is to generate the various signals required to coordinate the operations of the main store, arithmetic unit and input–output unit, and to cause them individually and in unison to execute the various operations required. Finally, two examples were presented to illustrate techniques for arriving at the logical structure of a digital computer as well as convenient means for describing that structure. It is our purpose here to review and amplify some basic points in this material.

A logical starting point in the design of a computer system is a con-

sideration of the kind and number of functions the system must perform. This leads to a specification of basic parameters such as word formats for instructions and data, the type and size of main store, the instruction repertory, a description of the arithmetic unit and its operation times, the type of control unit, etc.

Sometimes "logical design" is used to refer to the process by which one derives a minimal set of logical circuitry to perform specified functions. Logical design is here defined, in a broader sense, as synonymous with computer synthesis. It includes determination of the following: (1) the number and function of each building block: flip-flops, inverters, drivers, etc. (2) choice of a specific form of detailed logical structure—a statement of the organization of the various functional units, and a description of the interconnections of logical and nonlogical elements, taking into account limitations of a particular system of circuit logic which may place certain restrictions on permissible ways of interconnecting these elements.

In the computer designs presented in this chapter, three distinct means were employed to aid in a description of the logical structure of these machines, namely: (1) verbal statements, (2) block diagrams, (3) Boolean algebraic equations. Strictly speaking, logical design is an art. Though based on a knowledge of certain principles, it is dependent on the creative ability of the designer—his skill, intuition, and imagination. It is legitimate to use any tools which aid the creative process, and different designers use one or more to varying degrees according to their personal inclination. Also certain descriptions are convenient after a machine has been designed while others are more useful as aids in the synthesis process.

Verbal statements are useful, to begin with, in expressing the general structure of the machine. From this point on both block diagrams and Boolean algebraic statements may be helpful. Block diagrams are useful to indicate the paths of information flow between various parts of a system. At first, only relatively large functional blocks are delineated and as the design progresses, each block may be supplanted by several blocks, showing the logical structure in greater detail.

Different types of block diagrams, with varying degrees of detail may be useful in a number of ways. For example:

(1) To show the gross functions of various functional units.

(2) To show the arrangement and interconnection of these units for information flow.

(3) To show details of the internal design such as the number and locations of logical elements as well as nonlogical elements such as cathode or emitter followers, voltage clamps, pulse stretchers, etc.

(4) To indicate the manner of mating input–output devices to the

internal circuits of the computer by showing the details of matching devices such as buffer amplifiers, etc.

(5) (a) To show where physical wiring leads are to be placed, including the critical ones. The block diagram provides a one-to-one correspondence between logic and physical layout. The location of both local and remote terminals can be specified by coordinates.

(b) To show the physical interconnection of racks, registers, logic circuits, etc.

(c) To show, by means of different symbols on lines, the paths of pulse and dc signals.

(6) As a visual aid to checking for errors and unintentional redundancies in the design.

(7) As a visual aid in maintenance of the completed machine.

(8) To facilitate manufacturing and maintenance, since a block diagram can convey important design data to relatively untrained personnel.

In the conceptual development of a particular design, many gross arrangements may be considered before one is chosen. Block diagrams are useful as aids to both a spatial and temporal visualization of the arrangement of elements, aiding in the evaluation of alternate arrangements, and making apparent modifications that would improve the design.

A Boolean algebraic description of a computer can provide many of the functions performed by block diagrams. In addition, it is not as difficult to produce nor as cumbersome to manipulate as a block diagram, and offers other advantages because it constitutes a machine language. By definition, a machine language is a language which can be used to describe the structure of a digital computer, and which is of a form that enables it to be entered into and operated upon by any digital computer. A number of advantages accrue from this capability. First, after Boolean algebraic descriptions are entered into a computer, a number of useful listings can be obtained by sorting this data with respect to certain indices and tabulating the result. For example, the following tabulations are useful: (1) a tabulation of the inputs to all active storage elements, (2) a tabulation of all the loads on each active storage element, (3) a tabulation listing all physical interconnections of elements in the computer.

The first tabulation, called a logic tabulation, can describe inputs to all elements, whether of a logical nature or not, for the Boolean notation can be adapted to describe different types of circuits and gating arrangements. A computer can be programmed to inspect the logic tabulation to determine if any circuit restrictions have been violated, either in respect to nonallowable interconnections of elements in the permissible chains of logical elements and auxiliary circuits, the number of inputs to a gate

or the number of levels in a gate, or the number of gates which a flip-flop can drive.

The second tabulation is called a usage tabulation. The logic and usage tabulations facilitate maintenance since they indicate all elements that can affect a particular element as well as all elements that a particular element can effect. Also, from these two tabulations and a suitable program, a determination can be made of which elements are closely linked logically, and this can be used to facilitate physical layout specifications.

The third tabulation, which can be derived from the logic and usage tabulations, comprises a wiring tabulation for it lists all points that should be interconnected. It is useful for a number of purposes: (1) it will tell whether any wires have been left unconnected, thus providing a check on whether all required input signals have been specified; (2) it can also be updated more rapidly, in the event of changes either in the development or production stages, than wiring diagrams; (3) it is useful both in initially wiring a machine and in future maintenance, being easier to use than a set of wiring diagrams. If automatic wiring machines are to be used, they can be activated from the data in the wiring tabulation.

The logic, usage, and wiring tabulations describe the sources and destinations of all signals as well as the location of all components (and also enable a totalization of specified machine components to be readily obtained). While this adequately describes a machine, it is no assurance that the machine so described will function according to specifications. However, if the computer being synthesized is described in terms of a machine language, information relating to the actual operation of the machine can be obtained before construction by means of a simulation program. From this program, the logical operation of the computer being synthesized can be checked for various specified sets of problems and input data, and faults or omissions in the logical design can be detected. Also, simulation programs allow modifications or additions to the design to be checked readily, not only by the original designers, but by others less intimate with the structure of the machine. Finally, simulation aids in developing maintenance procedures, for specified faults can be simulated and their effect on the operation of the simulated machine observed. However, it must be pointed out that the cost incurred in producing these simulation programs is appreciable and should be justifiable on economic grounds or some other basis.

Some of the more important criteria to be considered when comparing different computer designs are as follows:

(a) Susceptibility of the computer to undetectable errors (inherent in its logical design).

(b) Reliability of operation (inherent in the electronic and mechanical design, and indirectly influenced by the logical design).
(c) Simplicity of design.
(d) Flexibility, both in respect to present operation and modification and expansion of the system.
(e) Compatibility of systems components.
(f) Speed and accuracy of computations.
(g) Automatic features.
(h) Convenience and flexibility of receiving input data from various sources.
(i) Facilities for presentation of output data.
(j) Provisions to retain intermediate results in case of power failure.
(k) Ease of servicing.
(l) Ease of training personnel.
(m) Economics of production.

The extensive use of time sharing, multiplexing, logical microprogramming and other schemes enable appreciable savings to be realized in the number of physical components in a system. However, this savings is achieved not without certain disadvantages. For one thing this type of design results in an almost complete loss of identity of functional units. This lack of a simple one to one correspondence between physical units and functions makes it difficult for any but highly skilled personnel to thoroughly understand the structure of a machine, and therefore makes maintenance more difficult. However, this disadvantage may be alleviated if the machine is built of reliable components and plug-in subassemblies which can readily be replaced.

Also, in developing schemes for equipment minimization it is important to consider not only the computer itself, but also the number, types, and mode of operation of input–output equipment that is to be utilized. Often, schemes that appear attractive for a machine with very limited input–output facilities lead to complications when additional terminal equipment is added. This is because a highly integrated system with little redundancy and flexibility does not have sufficient slack to allow additions to be squeezed in. As a result, modification of such systems may require unscrambling of many of the items originally integrated in the interior computer design.

In the last analysis, a particular design arrived at must be capable of justification on economic grounds. The choice will depend on whether one is interested mainly in solving differential equations, a computer capable of solving a wide variety of problems, in a computer for an automatic control system, or for other engineering, business or industrial applications. In all these cases, the particular information processing sys-

tem chosen must be justified either because it saves money, saves time, provides solutions not otherwise obtainable, enables a better product to be produced, increases operational efficiency, permits exploitation of new ideas, is more reliable than other methods, etc.

LITERATURE

Auerbach, A. A., Eckert, J. P., Jr., Shaw, R. F., Weiner, J. R. and Wilson, L. D. [1952] The Binac, *Proc. IRE,* **40,** 12-29.

Beckman, F. S., Brooks, F. P., Jr. and Lawless, W. J., Jr. [1961] Developments in the logical organization of computer arithmetic and control units, *Proc. IRE,* **49,** 53-66 (includes a bibliography of 60 entries).

Bigelow, J. H., *et al.* [1947] *Interim Progress Reports on the Physical Realization of an Electronic Computing Instrument,* Institute for Advanced Study, Princeton.

Blaauw, G. A. [1959] Indexing and control-word techniques, *IBM J. Research and Develop.,* **3,** 288-301.

Brooks, F. P., Jr. [1958] A program controlled interruption system, *Proceedings of the Eastern Joint Computer Conference,* 1957, 128-132.

Brooks, F. P., Jr. [1960] The execute operations—a fourth mode of instruction sequencing, *Comm. ACM,* **3,** 168-170.

Burks, A. W., Goldstine, H. H., Von Neumann, J. [1947] *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument.* Institute for Advanced Study, Princeton, N. J.

Burks, A. W., Copi, I. M. [1956] The logical design of an idealized general purpose computer, *Jour. Franklin Inst.,* **261,** 299–314; 421–436.

Carter, W. C., Ellis, M. [1954] A comparison of order structures for automatic digital computers, *Operations Research Soc. of America Journal,* **2,** 41-58.

Frankel, S. P. [1958] On the minimum logical complexity required for a general purpose computer, *IRE Trans. El. Comp.,* **EC-7,** 282-285.

Frankel, S. P. [1959] A logic design for a microwave computer, *IRE Trans. El. Comp.,* **EC-8,** 271-276.

Knuth, D. E. [1961] Minimizing drum latency time, *J. ACM,* **8,** 119-150.

Leiner, A. L. Notz, W. A., Smith, J. L., and Weinberger, A. [1959] PILOT—a new multiple computer system, *J. ACM,* **6,** 313-335.

Loberman, H. and Weinberger, A. [1957] Formal procedures for connecting terminals with a minimum total wire length, *J. ACM,* **4,** 428-437.

Lourie, N., Schrimpf, H., Reach, R. and Kahn, W. [1959] Arithmetic and control techniques in a multiprogram computer, *Proceedings of the Eastern Joint Computer Conference,* 1959, 75-81.

Staff of the Digital Computer Laboratory [1957] On the design of a very high-speed computer, *Digital Computer Lab., University of Illinois, Rept. No. 80.*

Van der Poel, W. L. [1951] A simple electronic digital computer, *Appl. Sci. Res.,* **2B,** 367-400.

Wilkes, M. V., Renwick, W. and Wheeler, D. J. [1958] The design of the control unit of an electronic digital computer, *Proc. Inst. Elec. Engrs.,* **105** (B), 121-128.

Wilkes, M. V., Stringer, J.B. [1953] Microprogramming and the design of the control circuits in an electronic digital computer, *Proc. Cambridge Phil. Soc.,* **49,** 230-238.

8. The Digital Differential Analyzer

8.1. Introduction

So far our attention has been directed mainly toward the general purpose type of digital computer. Utilization of such machines requires that an initial statement of a problem be reduced to a sequence of elementary arithmetic and logical steps, often involving numerical approximation algorithms. These steps are then reduced to a sequence of instructions (in machine code) selected from the computer's instruction repertory.

One of the most important and commonly encountered mathematical problems in the physical sciences and engineering is that of solving differential equations. An ordinary differential equation is, briefly, a mathematical representation of how an incremental change in one variable of a system affects the values of other variables in the system, and defines the way in which the system can change from one state to another. The equation, then, can be used to predict the effects of applying disturbing forces (referred to as forcing functions) to the system, provided the system's initial state, expressed by so-called boundary conditions, is known. These boundary conditions permit the evaluation of arbitrary constants that enter when integrations are performed. As many boundary conditions must be stated as there are arbitrary constants (in an ordinary differential equation) or arbitrary functions (in a partial differential equation).

The idea of building a machine in which the movement of functionally related parts would simulate the operation of defined, logical processes of the mind is usually credited to Leibnitz. Generally speaking, two systems are considered analogous if their elements have similar physical and/or abstract attributes, and the elements are similarly interrelated in each.

Differential analyzers are analog machines. They are based on the observation that, in general, a physical system can be represented by a group of elements interconnected so that disturbances of one or more elements are coupled to other elements. This is the basis for derivation of differential equations in the first place. The differential equation specifies the variables of interest in a system and the manner in which they are interrelated. It follows that if one has a set of idealized elements, analogous in behavior to the elements of a system to be investigated, one can use the differential equation of the system to indicate how these elements must be

interconnected. Assume then, that an "idealized" physical element is available which has the following properties. (1) It is capable of assuming a range of distinguishable states, and can indicate its state at all times. (2) It is capable of transmitting indications of changes in its states to other elements. Assume, further, that this element can accept increments of one variable, dx, as an independent variable input, and that of another variable as the dependent variable y, and that it can produce at its output an incremental change, $dz = ky\,dx$, where y represents the sum of increments accumulated from some initial time, t_0, and k is a constant. A device having these properties is referred to as an integrator. A commonly used functional schematic of an integrator is shown in Fig. 8.1. The first practical working differential analyzer used mechanical

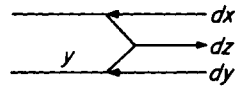


FIG. 8.1. Functional schematic of an integrator

wheel and disk integrators of the type shown in Fig. 8.2. Its mode of

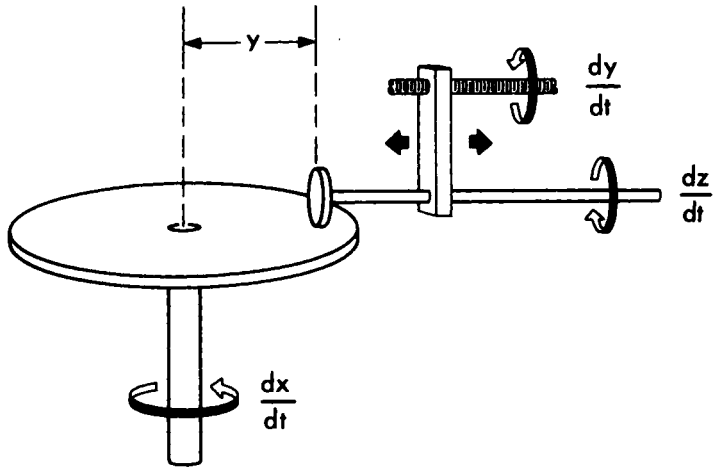


FIG. 8.2. Schematic of a mechanical integrator

operation is, briefly, as follows. The disk D is driven at a rate dx/dt, corresponding to the rate of change of an independent variable x. The

angular position of this disk represents the instantaneous value of x. The disk is geared by friction to a smaller disk. As a result, the smaller disk is forced to turn at some rate, say dz/dt. The ratio of the angular velocities of the two shafts is proportional to the distance y from the point of contact of the two disks to the center of the larger disk. Changes in the dependent variable y are effected by rotation of the lead screw. Any particular design will involve a constant, k, which is dependent on the relative sizes of the two disks. It is apparent then that $dz/dt = ky\,dx/dt$. If the dy/dt shaft is externally geared to the larger disk, so that the relation is independent of time, the following equations may be used to represent the action of the integrator

$$dz = ky\,dx \tag{8-1a}$$

or

$$z = k\int y\,dx \tag{8-1b}$$

Assume, also, that another element is available which can accept two or more variables in incremental form and produce an output equal to their algebraic sum. Such a device, suitable for use with mechanical integrators, is a differential gear assembly. It also allows the integration of the algebraic sum of a group of dependent variables to be obtained for its output can be used as the dy input of an integrator.

In principle, a machine composed of only integrators and adders is adequate to obtain the solution of differential equations. This is because:

(1) The action of these elements can be used to represent the action of each term in an equation. This action is in the form of shaft rotations, voltages, or pulse streams in the mechanical analog, electronic analog, and electronic digital differential analyzers, respectively.

(2) The individual elements can be interconnected so that the grouping of terms and the equality among the terms demanded by the equation are satisfied.

Inspection of a differential equation shows that there is always present an interdependence between values of derivatives and functions. This interdependence is satisfied by interconnecting integrators, as described in Sections 8.2 and 8.5.2. A characteristic of these interconnections is that there is always present at least one feedback path. It is the feedback connection which mechanizes the equal sign in the equation, for it imposes the constraint which forces the machine to operate so that the two sides of the equation are equalized. A standard procedure in setting up a problem is to anticipate that a feedback connection will be made. The feedback connection activates not only the element to which it is directly connected, but all others dependent on the activation of that element.

Examination of the ordinary differential equation

$$\frac{d^n y}{dx^n} = f\left[\frac{dx^{n-1}}{d^{n-1}}, \ldots, \frac{dy}{dx}, y, x\right] \tag{8-2}$$

shows that the feedback connection required always exists in at least one form. This follows since all terms on the right hand side can be developed from the independent variables and the nth derivative. Therefore, by supplying the independent variable, and anticipating an input providing the nth derivative, the right hand side of the equation can be produced, and this output fed back to supply the anticipated input carrying the nth derivative.

An alternate procedure is to use the highest-order derivative on the right hand side of the equation as the anticipated variable. From it and the dependent variable, all the remaining terms on the right can be derived. Once formed, the variable $d^n y/dx^n$ is integrated as many times as necessary to reach the order of the anticipated variable, where the feedback connection is made.

The conditions for solving an ordinary differential equation by means of a differential analyzer are outlined in the appendix to this chapter.

One may ask why an integrator is used as a basic element of a differential analyzer. It is obvious that an integrator can be used to generate lower-order derivatives from higher-order ones, in accordance with the relation

$$\frac{d^{n-1}y}{dx^{n-1}} = \int \frac{d^n y}{dx^n}\, dx. \tag{8-3}$$

However, an apparently equally useful relation, and one which would imply the use of a differentiator is

$$\frac{d^n y}{dx^n} = \frac{d}{dx}\left[\frac{d^{n-1}y}{dx^{n-1}}\right]. \tag{8-4}$$

Theoretically, the solution of differential equations could be mechanized with either of these devices. However, the integrator wins out on the basis of a very practical consideration. Differentiation requires essentially the subtraction of quantities of nearly equal magnitude. This implies that a differentiator would require a much greater precision for a given accuracy, since the subtraction of nearly equal quantities wipes out significant digits. The physical realization of such a device is, therefore, not as simple or practical.

Sir William Thomson [1876] was the first to suggest that mechanical integrators could be connected together in closed loops and constrained to produce solutions of differential equations. The first machine based on integrating devices was built at M.I.T. in 1925. It used a photo-

electric integraph (an integrating graph follower), and watt–hour meters as integrators. Its accuracy was only of the order of 1 part in 100. In 1930, Vannevar Bush developed at M.I.T. an all mechanical differential analyzer which utilized wheel and disk mechanical integrators. This machine could achieve an accuracy of 1 part in 1000, an accuracy of 1 part in 3000 being considered good for a mechanical integrator. In later machines where the mechanical integrator was placed in a servo loop, accuracies of 1 part in 30,000 could be obtained for the integrators with over-all accuracies ranging from below 1 part in 10,000 to 1 part in 25,000, depending on the nature of the problem.

Electronic analog differential analyzers, developed at a later date, use the integrating characteristic of a capacitor in conjunction with operational amplifiers to produce linearity. The advantage of the electronic analog machine over the mechanical is greater speed and compactness. The accuracy of the electronic analog devices is also limited, and involves long-term drift problems that are overcome only with highly engineered critical circuits. Also, unless special devices are employed the variable of integration must be time, and this places a restriction on the type of problems that can be solved.

The digital differential analyzer (DDA) is unique in that it provides certain desirable features of both analog differential analyzers and digital computation. The most important of these features are

(1) It can provide greater accuracy than is obtainable with an analog computer. Analog equipment is only as accurate as its components. In a mechanical analog differential analyzer, there is a limit to the accuracy of machined parts, and the accuracy decreases as wear continues. In the electronic analog type, there is a similar limit imposed by the stability as well as the precision of electronic components. In a digital computer, the components must be only capable of resolving two values, represented by two easily distinguishable voltages. Consequently, the digital differential analyzer is capable of yielding more accurate solutions as well as simplifying problems of maintenance.

(2) It inherently has greater logical capacity than analog machines.

(3) It provides greater flexibility than analog machines, because of its logical capacity, and also because changes can be effected by reprograming rather than equipment modification.

(4) It is more compact than analog machines (for systems beyond a certain minimum complexity).

(5) In an electronic analog integrator, the variable of integration must be time. The integrator in the digital differential analyzer can receive the output of any other integrator directly as its independent variable input.

This facilitates multiplication, division, and the solution of nonlinear equations without the use of special devices.

(6) It provides exact repeatability of problem solutions (not being subject to the variable drift of an analog computer).

(7) In a digital differential analyzer, differential equations may be solved without reducing them first to difference equations as required in an integral transfer (GP) type of digital computer.

(8) For the solution of certain types of problems, it can be mechanized with fewer components than required by an integral transfer type of digital computer.

Because an integrator can be used for a number of purposes, the number of integrators in a machine is not a reliable index of the maximum order of equation that can be solved. Specifically, in solving a differential equation, the value of a variable of interest is obtained by a process which includes integration of certain variables and generation of auxiliary functions usually of an algebraic or trigonometric nature. Integrators may also be used in the generation of these auxiliary functions, e.g., to generate the product of two variables. These auxiliary uses may, in some cases, consume more integrators than the reduction of derivatives. Thus, while only integrators and adders are essential to a differential analyzer (see Appendix, p. 516), as a matter of convenience, or economy, however, both mechanical and electronic differential analyzers have included other units. Mechanical differential analyzers may contain the following additional elements: gear boxes for multiplying a variable by a constant; resolvers for directly performing vector resolutions; multipliers for producing the product of two variables; input tables or function units which, given a variable, x, as an input constrain a second variable, y, to rotate as $y = f(x)$, where $f(x)$ is an arbitrary given function with only a finite number of finite discontinuities. Electronic analog differential analyzers may also contain special units for multiplication by a constant, multiplication of two variables, vector resolution, and function generation.

Digital differential analyzers may also be provided with units to facilitate execution of frequently encountered computing functions, e.g., output multipliers (Section 8.5.7.), decision units (Section 8.6), digital servos (Section 8.7) and more complex operational units for special purpose machines (Section 8.9). For machines to be used in computing laboratories, graph followers and plotters have also been developed to facilitate the insertion of empirical data, and to enable outputs to be displayed in graphical form. In certain types of control system applications (see Section 8.9), multiplication is called for more often than integration. This led to the development of an incremental multiplier, based on the

logical principles of the digital integrator. By the inclusion of appropriate control circuitry it then becomes possible merely by a simple programming procedure to cause the "multiplier" to provide the functions of both an integrator and a servo, simultaneously (see Fig. 8.27).

The differential analyzer came into being as a result of the deficiencies of classical methods of solving differential equations, especially nonlinear equations. The development of this machine to its present forms has been largely due to the following advantages which it offers, compared to analytic solutions, in the solution of problems in engineering: (1) The ease of changing parameters in a synthesis problem; this facilitates the investigation of many alternate configurations. (2) The completeness of solution; information can be generated which describes the states of a system at any specified number of points between an initial state and an arbitrary later time.

There is no single analytic method which provides the solution of differential equations in general. The differential analyzer provides a basic method applicable to the solution of all ordinary differential equations. Though it was invented to solve differential equations, it can also be used to solve arithmetic or algebraic equations. This can be done by causing it to solve the differential equations having the desired arithmetic or algebraic equations as solutions.

8.2. Generation of Functions in a Differential Analyzer

We will illustrate here how a number of commonly encountered functions may be generated by appropriate integrator interconnections. As indicated in the preceding section, the procedure consists of determining the differential equation(s) whose solution is the required function. An integrator hook-up that satisfies the differential equation relations will then generate the function. Often, there is available more than one integrator hook-up for generating a function. (See pages 476-480, and 511).

Integrator hook-ups for the generation of a number of commonly encountered functions are grouped together in Fig. 8.3 for convenience of reference. Most of these are self explanatory. However, some require simple mathematical preliminaries, or a few words of explanation. A statement of the functions generated in each of the entries in Fig. 8.3, as well as pertinent notes where necessary, are listed below.

(a) Generation of e^{kA}, given $d(kA)$, where k may be positive or negative.

(b) Generation of sin A, cos A, given dA

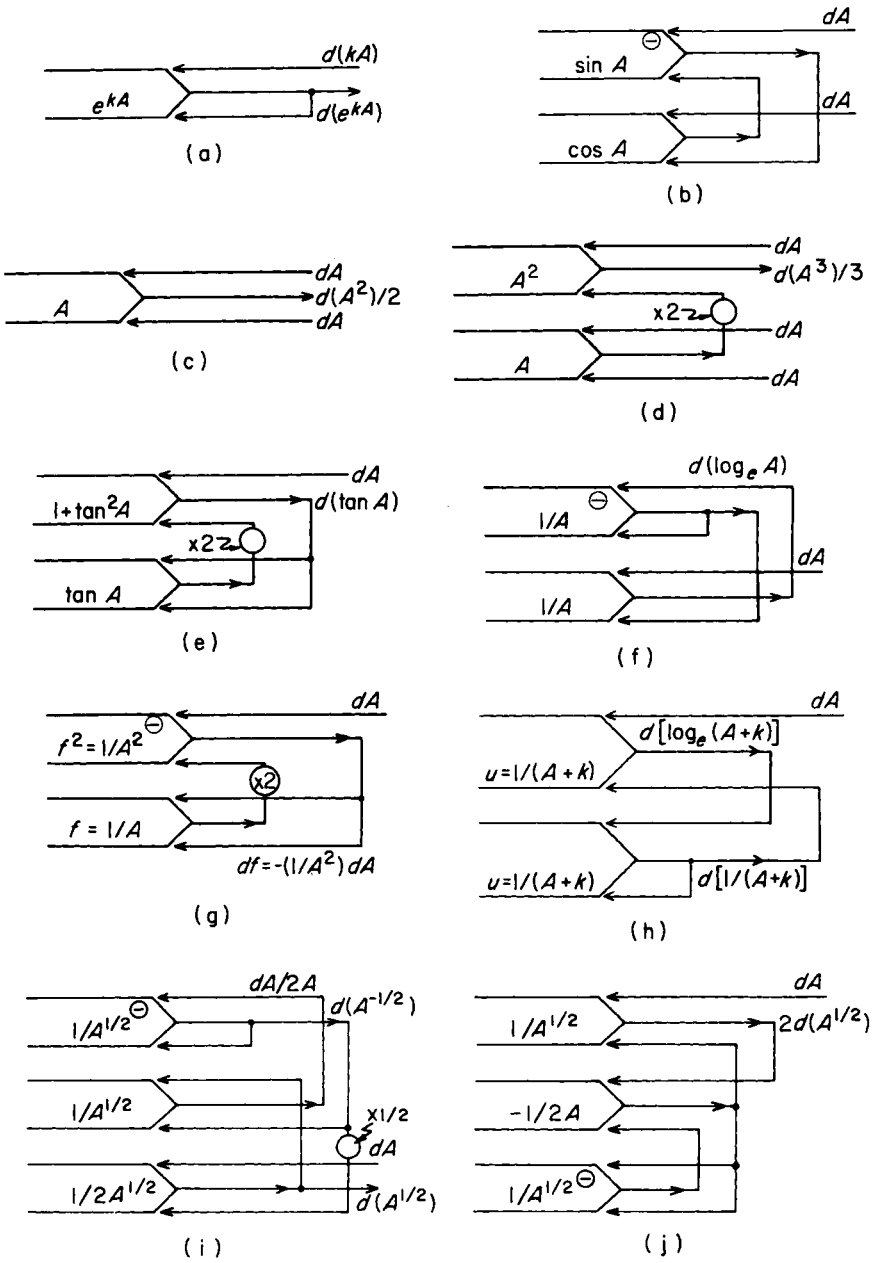


FIG. 8.3. Generation of functions by means of integrators.

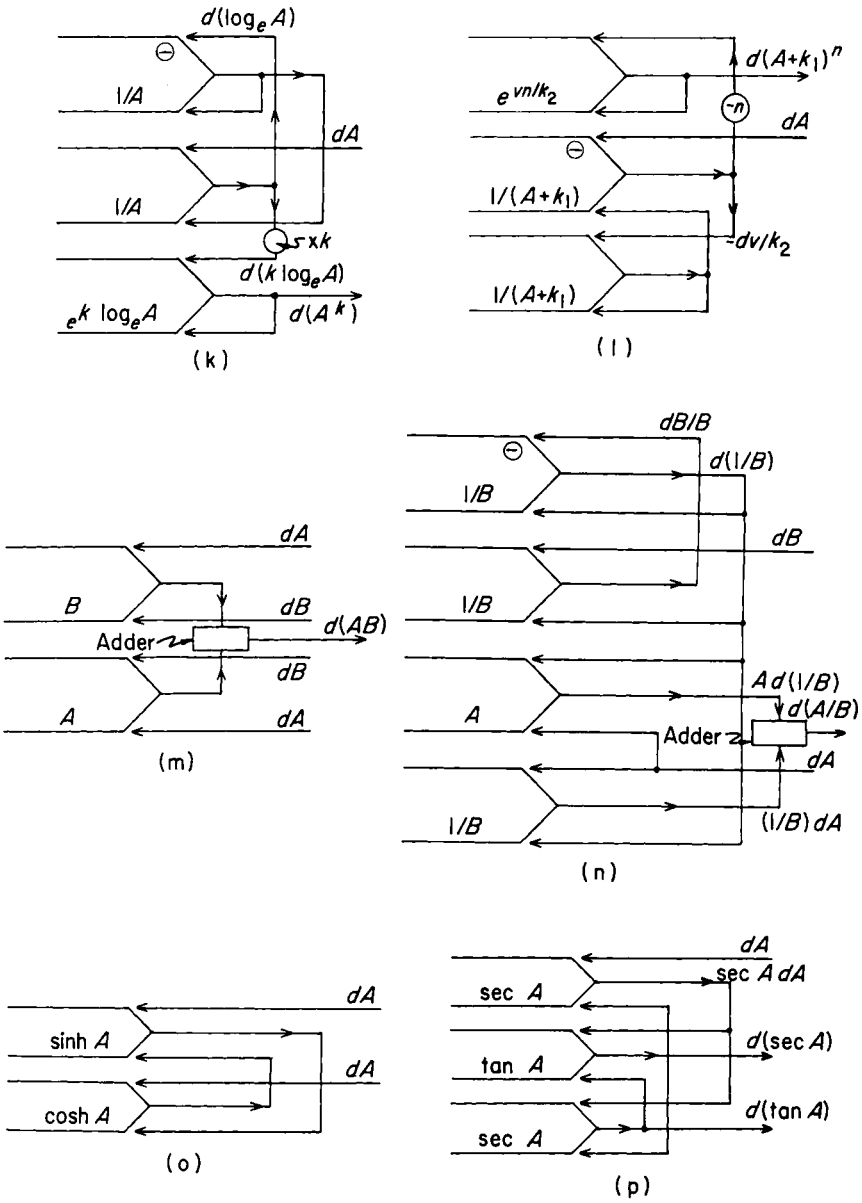


FIG. 8.3. Generation of functions by means of integrators.

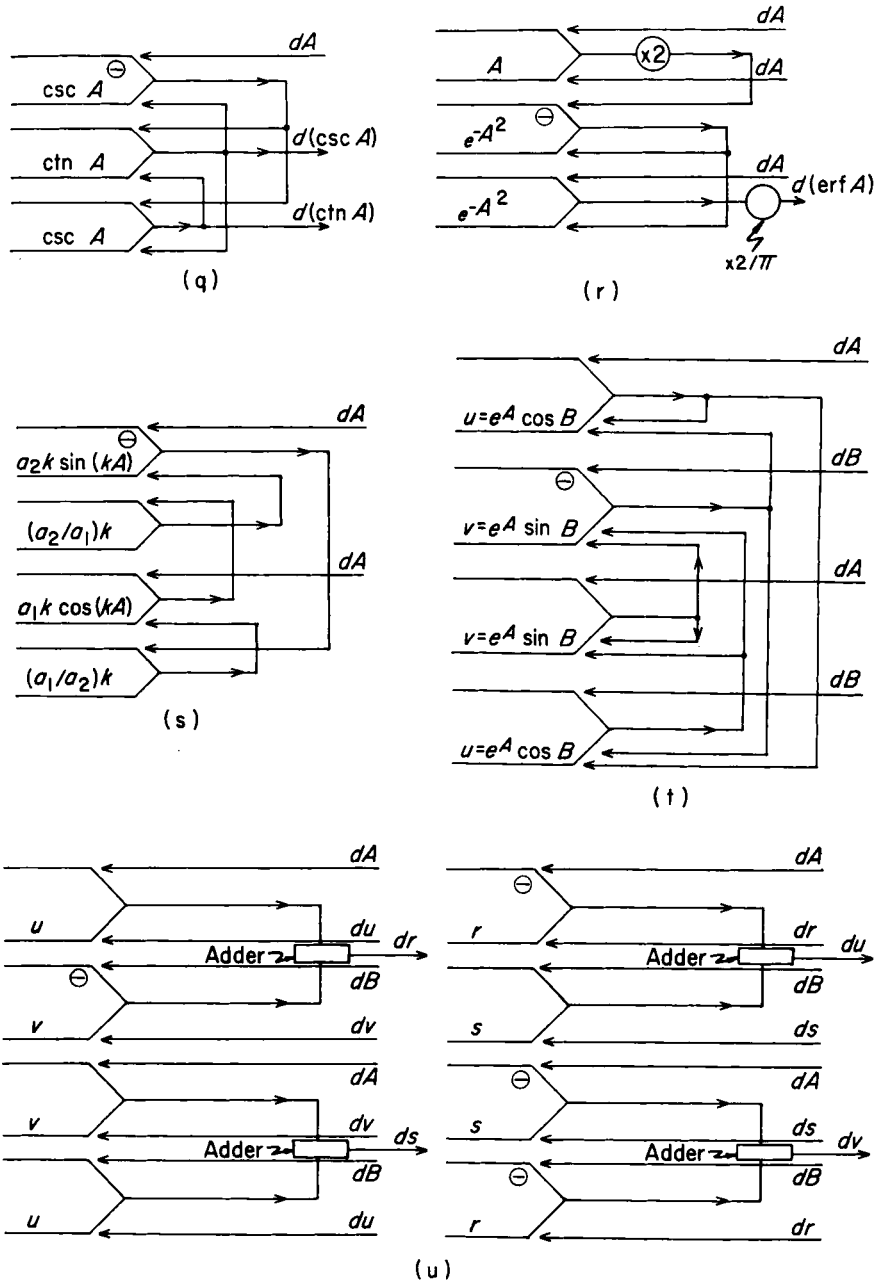


FIG. 8.3. Generation of functions by means of integrators.

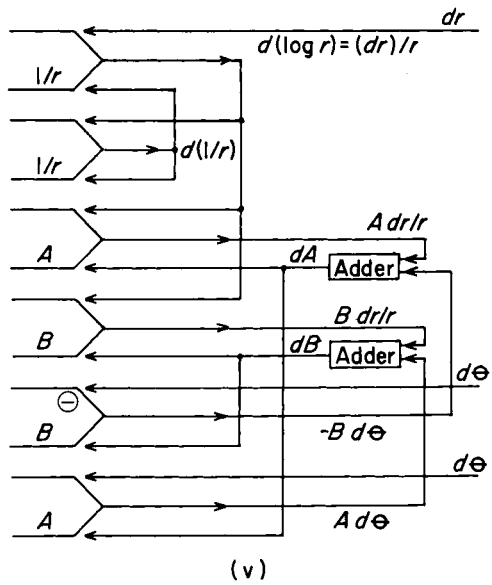


FIG. 8.3. Generation of functions by means of integrators

$$\text{Let } f_1 = \sin A,\ f_2 = \cos A$$

$$\frac{df_1}{dA} = \cos A, \quad \frac{df_2}{dA} = -\sin A$$

$$\frac{d^2f_1}{dA^2} = -\sin A, \quad \frac{d^2f_2}{dA^2} = -\cos A$$

$$= -f_1 \qquad = -f_2.$$

Therefore, an arrangement that satisfies the equation $d^2f/dA^2 = -f$ will generate both the sine and cosine.

(c) Generation of A^2, given dA

$$d(A^2) = 2A\,dA.$$

(d) Generation of A^3, given dA

$$d(A^3) = 3A^2\,dA.$$

(e) Generation of $\tan A$, given dA

$$1 + \tan^2 A = \sec^2 A$$

$$d(\tan A) = \sec^2 A\,dA.$$

For $A = 0$, the integrand register holding $(1 + \tan^2 A)$ is actually set to its maximum possible value $(1 - 2^{-n})$.

(f) Generation of $1/A$, $\log_e A$, given dA

$$d(1/A) = -(1/A)\, d(\log_e A)$$

$$d(\log_e A) = (1/A)\, dA.$$

(g) Generation of $1/A$, $1/A^2$, given dA

$$\text{Let } f = 1/A$$

$$df = -(1/A^2)\, dA$$

$$= -f^2\, dA.$$

Scaling difficulties are worse for this hook-up than that in (f), since the quantity $1/A^2$ in an integrand presents a greater scaling problem than $1/A$.

(h) Generation of $1/(A + k)$, and $\log_e(A + k)$, given k, da

$$\text{Let } u = 1/(A + k)$$

$$du = -(A + k)^{-2}\, dA$$

$$= -u^2\, dA.$$

The arrangement in (i) generates

$$du = uu\, dA = d[1/(A + k)]$$

The quantity $u\, dA$ is also generated in the process

$$u\, dA = dA/(A + k) = d[\log_e(A + k)]$$

(i) Generation of $A^{1/2}$, given dA

$$(1/2A^{1/2})\, dA = d(A^{1/2})$$

$$(1/A^{1/2})\, d(A^{1/2}) = dA/2A$$

$$-(1/A^{1/2})\, dA/2A = d(A^{-1/2}).$$

(j) Generation of $A^{1/2}$, given dA

$$(1/A^{1/2})\, d(1/A^{1/2}) = d(1/2A)$$

$$-(1/2A) \cdot 2 \cdot d(A^{1/2}) = d(1/A^{1/2})$$

$$(1/A^{1/2})\, dA = 2\, d(A^{1/2}).$$

(k) Generation of A^k, given dA

$$\text{Let } A^k = \exp(k \log_e A)$$

$$d(A^k) = \exp(k \log_e A)\, d(k \log_e A).$$

Note that the two-integrator arrangement for generating $d(\log_e A)$ is the same as in (f).

(l) Generation of $(A + k_1)^n$ where n may be integral or not, given k_1, n, dA.

The quantity $(A + k_1)^n$ is a general expression for an integral or nonintegral power of a function.

$$\text{Let } v = k_2 \log_e(A + k_1) = \log_e(A + k_1)^{k_2}$$

$$dv = k_2\, dA/(A + k_1)$$

$$vn/k_2 = n \log_e(A + k_1) = \log_e(A + k_1)^n$$

$$\exp(vn/k_2) = (A + k_1)^n.$$

Therefore, $(A + k_1)^n$ may be generated by generating $\exp(vn/k_2)$. If $n\, dv/k_2$ is given, only one integrator is required to generate $\exp(vn/k_2)$.

Note that the two-integrator arrangement for generating dv/k_2 is the same as the hook-up in (f).

(m) Generation of AB, given dA, dB

$$d(AB) = A\, dB + B\, dA.$$

(n) Generation of A/B, given dA, dB

$$d(A/B) = A\, d(1/B) + (1/B)\, dA.$$

The first two integrators are used to generate $d(1/B)$ from dB.

(o) Generation of $\sinh A$, $\cosh A$, given dA, dB

$$\text{Let } f_1 = \sinh A, \quad f_2 = \cosh A.$$

$$\frac{df_1}{dA} = \cosh A, \quad \frac{df_2}{dA} = \sinh A$$

$$\frac{d^2f_1}{dA^2} = \sinh A, \quad \frac{d^2f_2}{dA} = \cosh A.$$

Therefore, an arrangement that satisfies the equation $d^2f/dA^2 = f$ will generate both the hyperbolic sine and cosine.

(p) Generation of $\tan A$, $\sec A$, given dA. (For $-\pi/2 < A(rad) < \pi/2$)

$$d(\tan A) = (\sec A)^2\, dA$$

$$d(\sec A) = (\sec A)(\tan A)\, dA.$$

(q) Generation of cot A, csc A, given dA. (For $0 < A(\text{rad}) < \pi$)

$$d(\operatorname{ctn} A) = -(\csc A)^2\, dA$$

$$d(\csc A) = -(\csc A)(\operatorname{ctn} A)\, dA.$$

(r) Generation of probability (or error) function, erf A, given dA

By definition $$\operatorname{erf} A = \frac{2}{\sqrt{\pi}} \int_0^A \exp(-A^2)\, dA$$

$$d(\operatorname{erf} A) = \frac{2}{\sqrt{\pi}} \exp(-A^2)\, dA.$$

(s) Generation of $a_1 \sin kA$, $a_2 \cos kA$, given dA

Let $f_1 = a_1 \sin kA$, $f_2 = a_2 \cos kA$

$$\frac{df_1}{dA} = a_1 k \cos kA, \quad \frac{df_2}{dA} = -a_2 k \sin kA$$

$$\frac{d^2 f_1}{dA^2} = -k^2 f_1, \quad \frac{d^2 f_2}{dA^2} = -k^2 f_2.$$

This arrangement is identical to the one in Fig. 8.3(b) except for the use of two constant multipliers.

(t) Generation of the complex exponential, $e^{A + iB}$, given dA, dB

$$\text{Let } u + iv = e^{A + iB} = e^A e^{iB}$$

$$= e^A \cos B + ie^A \sin B$$

$$d(u + iv) = -e^A \sin B\, dB + \cos B\, e^A\, dA + ie^A \cos B\, dB + i \sin B\, e^A\, dA$$

$$du = -e^A \sin B\, dB + \cos B\, e^A\, dA = -v\, dB + u\, dA$$

$$dv = e^A \cos B\, dB + \sin B\, e^A\, dA = u\, dB + v\, dA$$

The arrangement shown in Fig. 8.3(t) does not produce the absolute value of the vector quantity $e^{A + iB}$ but only the value of the "real" and "imaginary" components, u and v, respectively.

(u) Generation of the complex sine and cosine, $\sin(A + iB)$, $\cos(A + iB)$, given dA, dB

$$\text{Let } r + is = \sin(A + iB) = \sin A \cos iB + \cos A \sin iB$$

$$= \sin A \cosh B + i \cos A \sinh B$$

$$d(r + is) = \sin A \sinh B\, dB + \cosh B \cos A\, dA + i(\cos A \cosh B\, dB - \sinh B \sin A\, dA)$$

$$\text{Let } u + iv = \cos(A + iB) = \cos A \cos iB - \sin A \sin iB$$

$$= \cos A \cosh B - i \sin A \sinh B$$

$$d(u + iv) = \cos A \sinh B\, dB - \cosh B \sin A\, dA - i(\sin A \cosh B\, dB + \sinh B \cos A\, dA)$$

$$dr = \sin A \sinh B\, dB + \cosh B \cos A\, dA = -v\, dB + u\, dA$$

$$ds = \cos A \cosh B\, dB - \sinh B \sin A\, dA = u\, dB + v\, dA$$

$$du = \cos A \sinh B\, dB - \cosh B \sin A\, dA = s\, dB - r\, dA$$

$$dv = \sin A \cosh B\, dB - \sinh B \cos A\, dA = -r\, dB - s\, dA.$$

The arrangement shown in Fig. 8.3(u) does not produce the absolute value of the vector quantities $\sin(A + iB)$, $\cos(A + iB)$, but only the value of the "real" and "imaginary" components, r, u, and s, v, respectively.

(v) Transformation from polar coordinates (r, θ) to rectangular coordinates (A, B).

$$A + iB = r e^{i\theta} = r(\cos\theta + i \sin\theta)$$

$$A = r \cos\theta$$

$$dA = \cos\theta\, dr - r \sin\theta\, d\theta$$

$$= (A\, dr)/r - B\, d\theta$$

$$B = r \sin\theta$$

$$dB = \sin\theta\, dr + r \cos\theta\, d\theta$$

$$= (B\, dr)/r + A\, d\theta.$$

8.3. Digital Integrators

The operations performed by a digital integrator are analogous to those performed by a mechanical integrator. An integrator may be considered as a device which receives two input rates and transmits one output rate, where the input and output rates for an ideal integrator (see Appendix of this chapter) are related as follows

$$\frac{dz}{dt} = k\left(\int \frac{dy}{dt}\, dt\right)\frac{dx}{dt} \tag{8-5}$$

Throughout this chapter an integrator's input lines will be referred to as dx and dy lines and its output line as the dz line; in schematics of integrator hook-ups, variables on these lines are shown as differentials, (as in Figs. 8.1 and 8.3). This allows an idealized statement of functions generated by particular integrator hook-ups without distinction between analog and digital integrators. However, for the case of digital integrators, quantities on these lines will be designated by the increments Δx, Δy, and Δz, respectively. The central store for the Δz outputs will be referred to as the Δz store or, sometimes, as the Z line (when a delay line store is used).

A block diagram of a basic digital integrator is shown in Fig. 8.4.

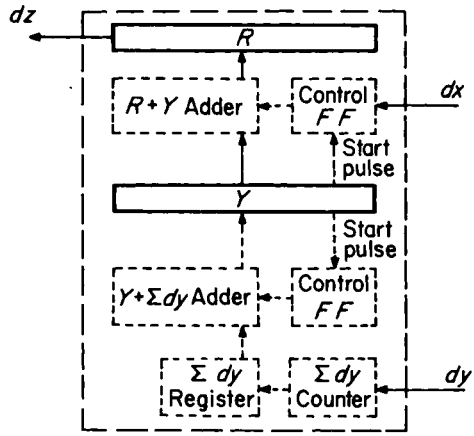


FIG. 8.4. Block diagram of a digital integrator

The digital integrator shown here consists of two accumulators, R and Y, together with associated adders and control circuitry. The block labeled "Σ dy register" receives from the "Σ dy counter" (a counter that can distinguish between and accumulate positive and negative increments), the algebraic sum of inputs appearing in sequence on an integrator's dy lines. (The number of dy lines is variable and is usually between 1 and 7). At a specified signal, the start pulse, the accumulated number in the $\Sigma\, dy$ register is shifted out and added to the contents of the Y accumulator, thus forming the new value of y. The number in the Y accumulator is added to or subtracted from the contents of the R accumulator according to whether Δx is positive or negative. The control flip-flops determine when the $(Y + \Sigma\, dy)$ and $(R + Y)$ additions start and stop (in accordance with a start pulse code usually placed in the Y accumulator in a position that is read just prior to the appearance of the least significant bit).

It is apparent that Y is added to R at a rate $\Delta x/\Delta t$. Eventually, the R accumulator will overflow, and this rate of overflow is taken as the output rate $\Delta z/\Delta t$. The Δz output signals are sent to a central store, where the current Δz outputs of all the integrators in the computer are held. The Δx and Δy inputs to the various integrators are obtained either from this central store or from external devices. The constant of proportionality between $\Delta z/\Delta t$ and $\Delta x/\Delta t$ for a digital integrator is determined by the numerical capacity of the accumulators. The maximum length of the accumulators is fixed in accordance with specified accuracy requirements.

It must be emphasized that the particular integrator unit illustrated is not the only one possible, but merely embodies the basic concepts of a digital integrator. Many variations within this basic design are possible. For example, (1) any of several number systems may be used in the Y accumulator, (2) an adder could be placed in the dx input line, (3) the capacity of the Σ dy register can be varied, subject to the anticipated requirements of the range of problems to be solved, and may even be eliminated by utilizing servo adders (see Section 8.7) to perform the summation of rates. A principal variant in the basic incremental computer design is the restriction placed on the allowable values of the increments produced and transmitted. In a machine with so called ternary transfer characteristics, each operational unit may produce any of three outputs: 0, + 1, − 1. In a machine with binary transfer, only two increment values are defined: + 1, − 1. At any step, Δz is defined to be + 1 if the R register overflows, and − 1 if it does not. The advantage of the binary transfer system is that it reduces the size of the Δz store by one half (since one binary storage element can represent either of two values, while two are required to represent three values) and also reduces the logical circuitry required. This economy is offset by a loss in precision and increased difficulty in comprehension of the machine's numerical operations. The peculiarities of the increment sequences and the means for producing them in the binary transfer system will now be compared to the more natural operation of the ternary system. In the latter, the average rate of change of a variable may be determined simply by summing the increments produced and dividing by the number of steps. For example, for the sequence + 1, 0, 0, − 1, − 1 the answer is $-2/6 = -1/3$. A zero output rate would be 0, 0, 0, . . . 0. In a binary transfer system, because of the lack of zero increments, special means must be provided to generate an average zero rate. This may readily be done by alternately generating positive and negative increments, in a machine where 0 represents − 1 and 1 represents + 1, simply by adding 1 to Δz every other step. The digital integrator described in Section 8.3.2. is designed for a machine with a binary transfer system and uses the number system for y shown in Fig. 8.18. With this system, the addition of zero (y = 1.000 . . . 0)

causes an overflow of the R register every other step. Thus, the various positive and negative output rates are comprised of the superposition of the "true" output rate on the zero-rate sequence. Returning to the loss of precision in the binary transfer system, referred to earlier in this paragraph, it is now apparent that in this system the least significant bit of the number in the Y accumulator actually has no significance. This is obvious if we consider the case where the dy input is a zero-rate sequence. Here, the value of the bit in the least significant position is determined only by whether it is inspected on an odd or even step of the iteration process. The loss of precision and the oscillation, too, are objectionable. In closing, it should be pointed out that the restriction on the size of increments to a single unit can cause considerable difficulty when the computer is used in applications where it is necessary to generate large magnitude changes within a short period. For this reason, variable increment computers have been designed which are capable of generating increments having any of several selectable values. The larger increments are used whenever it is necessary to generate a large change in a variable after a minimum number of steps, for example—in establishing new initial conditions (see Merz [1959] and Braun [1960]).

The particular integrator shown in Fig. 8.4 approximates integration by a simple rectangular summation operation, described in Section 8.3.2. The nature of this integration process is shown in Fig. 8.5. From the figure it is evident that the approximation can be improved by decreasing the size of increments in x and y, and that for a continuous

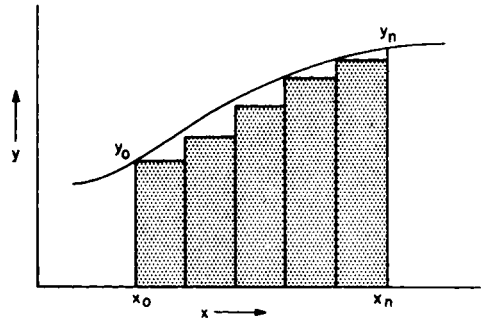


FIG. 8.5. Approximation to the integral by rectangular summation

monotonic curve the error of integration is $< (y_n - y_0)\Delta x$. A trapezoidal approximation to the integral is more precise and may be achieved by suitable modification of the scheme shown in Fig. 8.5. In a trapezoidal type of integration scheme, the area under a curve is approximated by

summing the areas of a series of trapezoids (see Fig. 8.6).

$$(y_t)_i = y_{i-1} + \tfrac{1}{2}(\Delta y)_i$$
$$\approx y_{i-2} + \tfrac{3}{2}(\Delta y)_{i-1}$$

The fundamental area of integration is $(y_t)_i(\Delta x)_i$, where $(y_t)_i$ is an

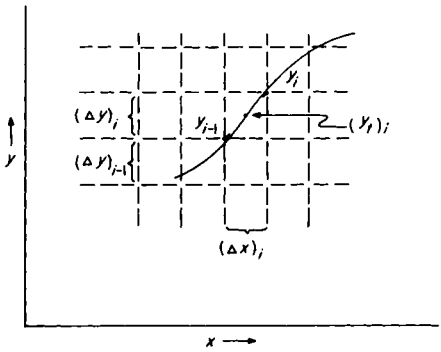


FIG. 8.6. Approximation to the integral by trapezoidal summation

estimate of the average value of y in the interval $(\Delta x)_i$. Theoretically, the integrator must take this into account. This may be done by utilizing for the rectangular scheme. The ratio of the error terms in the two methods is of the order of Δx.

In general, some of the successive Δx inputs to an integrator throughout a number of integration cycles will be zero. For trapezoidal integration, the error of approximation to the integral is less for the trapezoidal than an additional register as part of the integrator. This register, Y_t, accumulates the quantity

$$(y_t)_i = y_n + \tfrac{1}{2} \sum_{s=n+1}^{i} (\Delta y)_s \qquad (8\text{-}6)$$

where y_n is the value assumed by y_i when the last nonzero dx input, $(\Delta x)_{n+1}$, was received and $\Sigma_{s=n+1}^{i}\ (\Delta y)_s$ is the sum of all the dy inputs entering the integrator after y_n is set into the Y_t register. When a nonzero dx input occurs, the quantity in the Y_t register $(y_t)_i$ is either added to, or subtracted from, the contents of the R register, in accordance with the sign of the dx input. Thus, the value of y integrated is the average value of y in the region between the nonzero dx inputs rather than the value at either end-point. When an integrator's operation is described by Eq. (8-6), it is said to be operating in the interpolative mode, this name referring to the fact that the value of y used over an integration interval is obtained by interpolating between the values of y at the end-points.

8.3.1. Example of Operation of a Digital Integrator

The following example illustrates the operation of a digital integrator, utilizing a rectangular integration formula. It is assumed that the dz output of an integrator (with a sign reversal) is used as the dy input to the same integrator. In this case

$$dz = -z\,dx \tag{8-7a}$$

or

$$dy = -y\,dx. \tag{8-7b}$$

The solution of Eq. (8-7) is $y = e^{-x}$. Since, for $x = 0$, $e^{-x} = 1$, the initial value in the Y accumulator must be 1*. Table 8.1 shows the approximate values of e^{-x} generated by an integrator using a rectangular integration formula. (To simplify the description, it is assumed that the Y accumulator has a precision of only 1 part in 16.) Figure 8.7 shows the

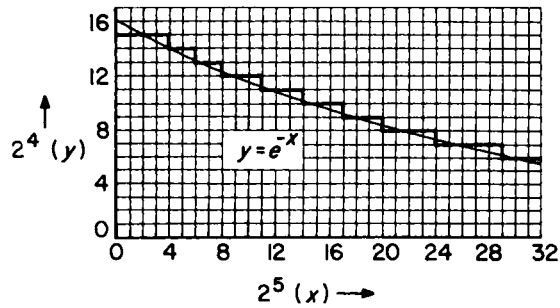


Fig. 8.7. Generation of e^{-x} by a digital integrator

function $y = e^{-x}$, and also the values as generated by the integrator. It is apparent that the error in the generated function is essentially less than the height of one increment. When accumulators of greater capacity are used, this increment becomes of less significance. In other words, the accuracy of the digital integrator can be extended indefinitely, theoretically.

8.3.2. Round-off Error in a Digital Integrator

The R register, whose capacity is $N = 2^m$, accumulates a sequence $n_1, n_2, \ldots n_k$ where $|n_i| < N/2$. In an integrator employing rectangular summation, Δz is the scaled down sum of the elemental areas $y_i\,\Delta x$ (see

* There is an initial error of magnitude $\frac{1}{16}$ since the largest number the Y accumulator can hold is $1 - 2^{-n}$ and in this case, $n = 4$.

TABLE 8.1.

2^5x	e^{-x}	2^4y	R	dy
0	1.00	15	0	
1	0.97	15	15	
2	0.94	15	30	
3	0.91	15	13	−1
4	0.88	14	27	
5	0.86	14	9	−1
6	0.83	13	22	
7	0.80	13	3	−1
8	0.78	12	15	
9	0.75	12	27	
10	0.73	12	7	−1
11	0.71	11	18	
12	0.69	11	29	
13	0.67	11	8	−1
14	0.65	10	18	
15	0.63	10	28	
16	0.61	10	6	−1
17	0.59	9	15	
18	0.57	9	24	
19	0.55	9	1	−1
20	0.54	8	9	
21	0.52	8	17	
22	0.50	8	25	
23	0.49	8	1	−1
24	0.47	7	8	
25	0.46	7	15	
26	0.44	7	22	
27	0.43	7	29	
28	0.42	7	4	−1
29	0.40	6	10	
30	0.39	6	16	
31	0.38	6	22	
32	0.37	6	28	

Fig. 8.5). In a machine with a binary transfer system Δz is considered positive in a period when register R overflows and zero otherwise. During the ith processing of an integrator, $[n_i + (N/2)]$* is added to register R, leaving in it some quantity r_i. If register R were of unlimited capacity, the number r_u which it would contain during the ith processing would be

* The particular type of integrator being described here is one in a system with binary communication between integrators (see page 464). This accounts for the term $N/2$.

$$r_u = r_o + \sum_{j=1}^{i} (n_j + N/2). \tag{8-8}$$

$$= r_o + \frac{iN}{2} + \sum_{j=1}^{i} n_j. \tag{8-9}$$

Letting $[\ ]^I$ stand for "integral part of," i.e., the sum of the overflows produced during i iterations

$$\left[\frac{r_u}{N}\right]_i^I = \left(\frac{r_u}{N}\right)_i - \frac{r_i}{N}. \tag{8-10}$$

The Δz contribution during the ith step, namely $[n_u/N]^I_i - [ru/N]^I_{i-1}$ is either 0 or + 1, and a convenient general expression for Δz is

$$\Delta z = 2\left\{\left[\frac{r_u}{N}\right]_i^I - \left[\frac{r_u}{N}\right]_{i-1}^I\right\} - 1. \tag{8-11}$$

Summing the unit increments, Δz_i

$$\sum_{i=1}^{k} \Delta z_i = 2\left\{\sum_1^k \left[\frac{r_u}{N}\right]_i^I - \sum_1^k \left[\frac{r_u}{N}\right]_{i-1}^I\right\} - \sum_1^k 1 \tag{8-12}$$

$$= 2\left\{\sum_1^k \left[\frac{r_u}{N}\right]_i^I - \sum_0^{k-1} \left[\frac{r_u}{N}\right]_i^I\right\} - k \tag{8-13}$$

$$= 2\left\{\left[\frac{r_u}{N}\right]_k^I - \left[\frac{r_u}{N}\right]_0^I\right\} - k. \tag{8-14}$$

Since $0 \leqslant (r_u)_0 = r_0 < N$, $[r_u/N]^I_0 = 0$. Also, substituting the right hand side of Eq. (8-10) into Eq. (8-14) yields

$$\sum \Delta z_i = 2\left\{\left(\frac{r_u}{N}\right)_k - \frac{r_k}{N}\right\} - k \tag{8-15}$$

$$= \frac{2}{N}\left\{(r_u)_k - r_k - \frac{kN}{2}\right\}. \tag{8-16}$$

Substituting the value of $(r_u)_k$ given by Eq. (8-8)

$$\sum \Delta z_i = \frac{2}{N}\left\{r_0 - r_k + \sum_{j=1}^{k} n_j\right\} \tag{8-17}$$

$$= \frac{2}{N} \sum_{j=1}^{k} n_j + \frac{2}{N} (r_0 - r_k) \tag{8-18}$$

$$= \frac{2}{N} \sum_{j=1}^{k} n_j + e \tag{8-19}$$

where $e = (2/N)(r_0 - r_k)$ is the round-off error. To determine its upper bound, consider its absolute value $|e|$

$$|e| = \frac{2}{N} |r_0 - r_k| \tag{8-20}$$

Since $|r_0 - r_k| < N$, $|e| < 2$. If initially the contents of register R are set equal to $N/2$

$$|e| = \frac{2}{N} \left| \frac{N}{2} - r_k \right|. \tag{8-21}$$

Since $0 \leqslant r_k < N$, $|N/2 - r_k| \leqslant N/2$, and

$$|e| \leqslant 1.$$

8.3.3. Choice of Integration Formulas

In a binary transfer machine, the numbers n_i which are added to the R register are given by $n_i = y_i \Delta x_i$ where $\Delta x = \pm 1$. During the ith iteration, the quantity added to register R is $(y_i \Delta x_i + N/2)$. If $\Delta x = +1$, the contents of Y are added to register R, but if $\Delta x = -1$, the N's complement of y is added to register R, i.e.,

$$r_i = r_{i-1} + (y_i + N/2) \text{ if } \Delta x_i = +1 \tag{8-22}$$

$$r_i = r_{i-1} + N - (y_i + N/2) \text{ if } \Delta x_i = -1$$

$$= r_{i-1} + (-y_i + N/2). \tag{8-23}$$

Equations (8-22) and (8-23) may be written in the generalized form

$$r_i = r_{i-1} + N/2 + y_i \Delta x_i \tag{8-24}$$

If in Eq. (8-19) n_j is replaced by $y_i \Delta x_i$

$$\sum \Delta z_i = \frac{2}{N} \sum y_i \Delta x_i + e. \tag{8-25}$$

Let us consider the nature of y_i, the number to be added to register R. If the initial value of y is y_0, and if during the ith cycle, there is an increment, Δy_i, as well as a unit increment, Δx_i, then the following general expression may be written

$$y_i = y_0 + \sum_{j=1}^{i-1} \Delta y_j + F(\Delta x_i)\ \Delta y_i \tag{8-26}$$

where the term $F(\Delta x_i)\Delta y_i$ indicates that y_i (the current value of y) may be a function of the increment in x, as well as the increment in y.

Four simple possible choices for $F(\Delta x_i)$ are

Case 1:

$$F(\Delta x_i) \equiv 0 \tag{8-27a}$$

Case 2:

$$F(\Delta x_i) \equiv 1 \tag{8-27b}$$

Case 3:

$$\left.\begin{aligned} F(\Delta x_i) &\equiv 1 \text{ if } \Delta x_i = +1 \\ &\equiv 0 \text{ if } \Delta x_i = -1 \end{aligned}\right\} \quad \text{i.e., } F(\Delta x_i) = \tfrac{1}{2}(1 + \Delta x_i) \tag{8-27c}$$

Case 4:

$$F(\Delta x_i) \equiv \tfrac{1}{2} \tag{8-27d}$$

If the value of $F(\Delta x_i)$ for each of these cases is substituted into Eq. (8-26), the following expressions for y_i are obtained

Case 1:

$$y_i = y_0 + \sum_{j=1}^{i-1} \Delta y_j \tag{8-28a}$$

Case 2:

$$y_i = y_0 + \sum_{j=1}^{i-1} \Delta y_j + \Delta y_i \tag{8-28b}$$

Case 3:

$$y_i = y_0 + \sum_{j=1}^{i-1} \Delta y_j + \tfrac{1}{2}(1 + \Delta x_i)\ \Delta y_i \tag{8-28c}$$

Case 4:

$$y_i = y_0 + \sum_{j=1}^{i-1} \Delta y_j + \tfrac{1}{2}\,\Delta y_i. \tag{8-28d}$$

For case 1, the value of y_i is that at the end of the $(i-1)$th iteration cycle (which may be referred to as "old y"). For case 2, the value of y_i is that at the end of the ith iteration cycle ("new y"). For case 3, y_i is equal to "new y" for $+\Delta x$ and "old y" for $-\Delta x$. Case 4 provides for linear interpolation.

8.4. Structure of a Digital Differential Analyzer

There are several levels of abstraction and organization on which to functionally describe a digital computer. For example, we may consider such a machine as a "black box" containing a set of switching elements and bistable-state storage elements. Its functional organization can be completely specified by stating how its elements are interconnected. To specify its functional state requires, in addition, a description of the current state of its storage elements.

The mechanics of solving a mathematical problem, or processing data, in general, consists of translating given initial information by prescribed rules to a more useful form. The computer in a sense does not create any new information, i.e., information not inherent in the original data. It may be considered both a computational operator, performing arithmetic and logical transformations, and also a communication system, since it accepts input data and transmits selected output information. The fundamental process of coding, by which information is translated by specified rules from one form to another, enters into both areas. In a computer, any of innumerable coding schemes may be used to convert input data to a form more appropriate for the machine's structure; also to convert results to a form called for by the end use. In any communication system, coding may be used to improve the probability of correct detection of information after transmission over a noisy channel, while within a computer special codes may be used to detect and correct errors generated either in arithmetic operations or data transfers (see Section 9.2).

In preceding descriptions, integrators were spoken of as distinct physical entities. However, though it is convenient to consider the integrator as an operational entity, in many digital differential analyzers, namely those with completely serial organization, an integrator does not exist as a distinct physical entity, as in mechanical and electronic analog differential analyzers. Nevertheless, even in this case, one can think of the machine

as having several units comparable in function to mechanical or electronic integrators. These units are capable of being interconnected as far as information transfer is concerned in such fashion that they will solve any problem that can be solved on a differential analyzer. The digital differential analyzer acts as a pulse coded analog of any physical system under investigation and is similarly constrained in its behavior. The rates of mechanical shaft rotations in a mechanical analog computer are represented in the digital differential analyzer by the repetition rates of electrical pulses, and angular or linear displacements by the contents of storage registers which accumulate these pulse inputs.

In a completely serial digital differential analyzer, all R and Y accumulators are mechanized as circulating registers. Specifically, the various integrands can be stored as magnetizable cells on the surface of a dynamic magnetic store. Each bit of each integrand is operated upon serially as it is read from the store. The circuits controlled by the variable of integration, which cause the transfer of data to the R accumulator, are time-shared among all the integrators. All of the arithmetic and control operations to be performed on an integrator are effected by signals from different parts of a gating network which, in turn, receive their signals from the control and read channel flip-flops. Only one arithmetic unit is required, for after each binary place of an integrator is operated upon, the result of that operation is sent to the store, leaving the computation and control circuits free to operate on new information being read from the store. The manner in which information is processed in a serial digital differential analyzer is presented in Fig. 8.8 and the discussion following.

The computation and control center, shown in Fig. 8.8, processes all information within the machine, and effectively achieves the only arithmetic functions required in a digital differential analyzer, namely, addition and subtraction. It receives signals from the storage synchronizing flip-flops, and also from the arithmetic and control flip-flops. From these, new signals are generated which are either recorded in the store or used to set arithmetic and control flip-flops. Of the many possible combinations of input signals, only those which represent the operational steps required in a differential analyzer are permanently wired into the gating network. As a result, it is not necessary to instruct the machine in the details of how to solve a problem, but only to insert the initial conditions into its store. A general purpose (GP) type of digital computer has a central arithmetic unit that must be used in solving a variety of problems. Therefore, to solve a specific problem, a program of instructions as well as initial numerical values, must be entered into its main store.

In a machine of this type, flip-flops are employed for several purposes: (1) to provide distinct time signals, (2) to facilitate the synchro-

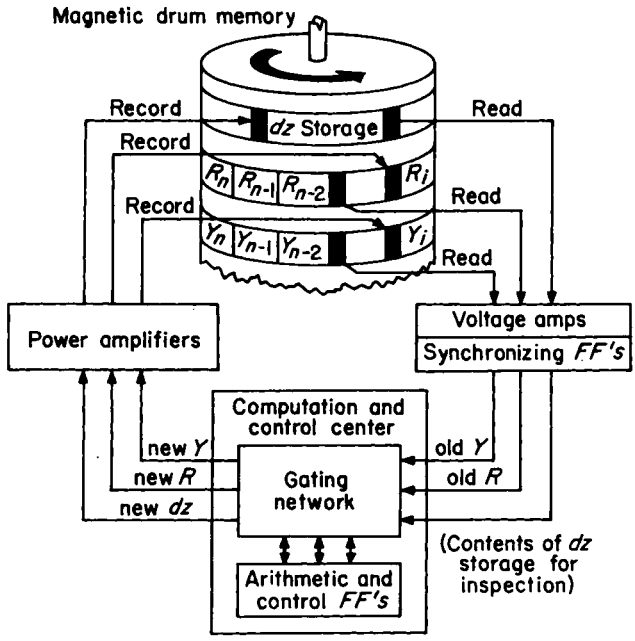


FIG. 8.8. Information flow through a completely serial digital differential analyzer (external inputs and outputs not shown)

nization of information from the memory with the computer's clock, (3) to control the arithmetic and logical operations performed in the computer, and (4) to provide time delays for carries in the adders.

In a completely serial machine, only one digit of one integrator is updated at a time in the computation and control center, and so it is necessary to provide a means for storing the other bits of that integrator, and the bits of all the other integrators. Each integrator is operated upon in sequence and in a cyclical manner: integrator $i + 1$ is always processed following integrator i. When this type of operation is used, the fixed delay or recirculation type of memory is practical for a digital differential analyzer. Completely serial DDA's have used magnetic drums or disks for the main store because of the combination of economy and reliability they afford. However, when the integrator registers are organized serially on a drum or disk, the iteration rate, i.e., the frequency of processing of each integrator, is limited by the rotational frequency (usually somewhere between 80 to 160 rev/sec). Also, there may be difficulties in addressing integrators, as brought out at the end of this section. When these limitations cannot be tolerated, DDA's can be mechanized with static stores and varying degrees of parallel operation (see Section 8.11).

Because the access time (i.e., the time required to obtain a specified word from storage) is variable in a dynamic magnetic store, depending on the word's position relative to a reading head at the time the word is called for, there is a distinct disadvantage in its use as the main store of a GP computer. However, the organization and control of the program in a DDA is such that a dynamic type of store can be used in what amounts to a zero access time mode of operation. This type of organization leads to a simple and elegant machine that consists mainly of passive storage elements (magnetizable cells) and in which the number of actual computing elements can be kept to a minimum. Actually, all that is required besides the storage registers are some relatively simple adders, counters, and some form of control. Because of the serial type of operation used, the adders can be relatively simple.

The signals read from the store are not of a suitable amplitude or shape to be processed directly by the gating network. Consequently, these signals are amplified and restored to rectangular pulses before entering the network. Each read channel consists of voltage amplification and shaping circuits and a flip-flop that synchronizes data read from the store with the clock. Each record channel has voltage amplification circuits, followed by a power amplifier which drives a magnetic recording head.

During each bit period of the time interval when the contents of a particular integrator (or similar type of operational unit that may be in a DDA, e.g., decision unit, multiplier, etc., described later in this chapter) are read out of the long delay lines, the last recorded output of some operational unit can be read out of the Z line. This line, we recall, is a short fixed delay store which holds the last recorded output of each operational unit. Information in the Z line is recirculated at all times except when new information generated by an operational unit replaces the preceding value.

If the number of operational units in the computer is equal to or less than the number of bits defining the length of an operational unit, a single addressing channel is sufficient to hold each marker required to indicate the pulse time when the output of a particular operational unit is available from the Z line. The information in the address channels is used as follows: When initially filling the computer, 1's are placed in those positions in the address channel which correspond to pulse times at which inputs to be routed to a particular operational unit are available from the Z line. This information is used to control coincidence gates which will admit information from the Z line at the specified time. The address channel achieves the same function as the physical wiring of a plugboard used to connect operational units in an analog differential analyzer.

A convenient way of utilizing the address channel is to read from it during each time interval the marker bits that will cause the pick up of

the dz's required as inputs to the operational unit that will be processed during the succeeding time interval. This is shown in Fig. 8.9.

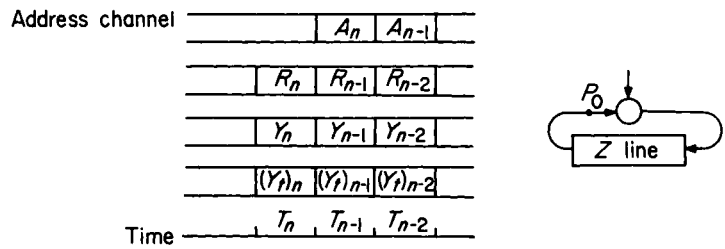


FIG. 8.9. Relative locations of address decoding information and other data in a serial DDA

A major difficulty arises in a completely serial DDA when there are more operational units than bits per register, because the marker system for Δz selection requires a marker position for each Δz. When several address lines are required, each can be associated with one of several read heads so stationed along the Δz line that they scan all Δz's in one word time. (For high-density recording the close head spacing required may not be realizable, so several Δz lines may be used). The possibility of simultaneous inputs from the Δz stations complicates matters too. (Also, the reader may reflect on the additional encoding and decoding required to channel a designated Δz to the proper input or combination of inputs of an operational unit). For further comments on this problem see Sections 8.10 and 8.11.

8.5. Preparation of Problems for a Digital Differential Analyzer

8.5.1. INTRODUCTION

A digital differential analyzer can be used to solve an ordinary differential equation of any order or degree, linear or nonlinear, or a simultaneous set of such equations. It can also be used to solve integral equations, transcendental algebraic equations, and simultaneous sets of such equations. Actually, any problem which has a solution that is also the solution of a set of nonsingular differential equations can be solved on a digital differential analyzer provided that the machine has a sufficient number of components of two types, adders and integrators. A principal advantage of the DDA over the general purpose arithmetic computer as an engineering computing aid is its more direct, simple, and intuitive approach. With a DDA, differential equations need not be reduced to

difference equations before computation can be initiated. Instead, programming a problem involves only three steps: mapping, scaling, and coding. Mapping, which consists of specifying how the operational units are to be interconnected, and scaling are discussed in the paragraphs following. Coding will not be discussed because it is routine and consists of detailed procedures which differ appreciably from one machine to another.

Any continuous function required in the solution of a problem can be generated within a DDA by suitably interconnecting a set of integrators to solve an auxiliary differential equation whose solution is the required function. Provision can also be made for inserting empirical or discontinuous data into a DDA.

8.5.2. MAPPING

Mapping consists of specifying how the operational units in a machine should be interconnected so that the variable or variables of interest are generated within the system. The value of these variables as a function of the independent variable represents the solution to the problem being investigated. The desired interconnections between operational units is usually expressed by means of marker code bits inserted in that part of the store reserved for this purpose (see Section 8.4). The actual interconnection is performed by logical decoding networks in the computer's control circuitry. During computation, information is transmitted between integrators in the form of incremental changes in variables. The source of the dx input may be the output of any integrator as well as the computer's internal clock, the empirical data received from external sources like paper tapes, magnetic tapes, etc. The dy input may also come from any of these sources. Also, a dx or a dy input may consist of the sum of outputs from several sources. Though adders can be provided to sum both dx and dy inputs, for reasons of design simplicity and practical considerations, dx inputs are limited usually to one or two, while several dy inputs are permitted. It can be shown that any ordinary differential equation can be so mapped that more than one input on a dx input line is never needed. On those occasions where it is convenient to use the sum of two variables as a dx input to a particular integrator, the summation may be performed prior to transmission to the integrator by means of a servo adder (see Section 8.7).

To establish the interconnections required to generate the solution of a given problem, a procedure like the following may be used: (1) If necessary, reduce the problem to a set of differential equations. (2) Isolate the highest derivative of each dependent variable by putting it on the left

side of the differential equation and all other terms on the right.* (3) Assume the highest derivative is known, and by integrating it, generate all lower derivatives required in the problem. (4) Combine the variables on the right side of the equation as indicated and use this sum as the source of the assumed highest derivative. This procedure is illustrated in Fig. 8.10 where the equation

$$a\,d^2x/dt^2 + b\,dx/dt + cx = 0 \tag{8-29}$$

is solved by double integration of the expression

$$d^2x/dt^2 = -(b/a)\,dx/dt - (c/a)x.$$

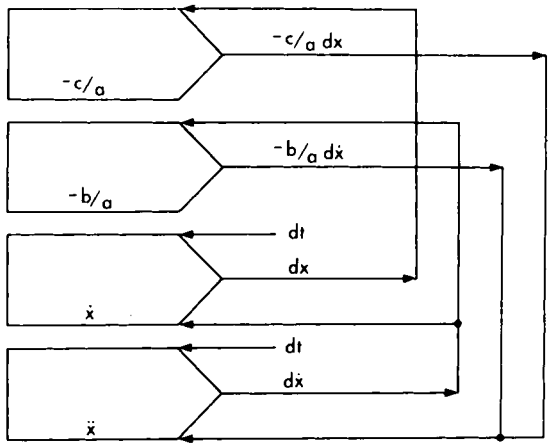


FIG. 8.10. A mapping for solution of the equation:
$a\,d^2x/dt^2 + b\,dx/dt + cx = 0$

In Section 8.3, a trapezoidal and a rectangular integration formula were considered, and in Section 8.3.3, certain variants were described. We will continue next with some comments on how the choice of a rectangular formula or an interpolative or extrapolative form of the trapezoidal formula is influenced by the assignment of integrator numbers, when integrators are processed sequentially in a common arithmetic unit.

* This may not always be possible, as, for example, in the case of certain transcendental types of equations, where the highest order derivative might appear alone on one side of the equation, and also be the argument of some function on the opposite side. Nevertheless, if the equation does have a solution, the machine can be programmed to find it.

(This assumes that a machine provides a programmer with the choice of integration formula to be used in each integrator—which is not always the case.)

The assignment of integrator numbers determines the order in which each integrator will operate in an iteration period of the computer. Each iteration begins with operations on integrator number 1, followed by operations on higher numbered integrators in ascending order. The assignment of the modes of integration is determined by the flow of information indicated by the mapping, taking into account the order in which the integrators are operated upon.

The interpolative mode gives greater accuracy only when the *dy* inputs are integrator outputs that have occurred in the same integration period. The solution of most problems requires the frequent use of integrator hook-ups in which some of the *dy* inputs are integrator outputs that have occurred in the preceding integration period. Therefore, when trapezoidal integration is to be performed and the *dx* input is $(\Delta x)_i$, the *y* value to be integrated must be predicted by an extrapolation of the $(\Delta y)_{i-1}$ input. The simplest extrapolation is a linear one, and an integrator performing such an extrapolation is said to operate in the extrapolative mode. An integrator operating in this mode functions as one in the interpolative mode, except that the quantity accumulated in the Y_i register during the *i*th period is

$$(y_i)_i = y_{n-1} + \tfrac{1}{2} \sum_{s=n}^{i-1} (\Delta y)_s + (\Delta y)_{i-1} \qquad (8\text{-}30)$$

where y_{n-1} is the value that was held in the *Y* register when the last nonzero *dx* input, $(\Delta x)_{n+1}$, was received.

The following rules may be used in assigning integrator numbers and modes: (1) The *dx* input to each integrator should be the machine independent variable, Δt, or the output of a smaller numbered integrator. (2) Whenever possible, the *dy* input should be Δt or the output of a smaller numbered integrator. Integrators so programmed should be assigned the interpolative mode. (3) When the *dy* input is the integrator's own output or the output of a higher numbered integrator, the extrapolative mode should be used. (4) For multiple *dy* inputs, all of the inputs should come either from sources as in rule 2 or, as in rule 3, i.e., unmixed, and the respective mode of integration should be used. (5) The inputs to an adder should come from smaller numbered integrators. (6) Use of a digital servo (see Section 8.7) may require violation of

some of the preceding rules, but the amount of violation should be kept to a minimum. (7) For multiplication of two variables, both integrators should receive their inputs from smaller numbered integrators.

8.5.3. SCALING

Any computing machine has a limitation in the magnitude of the numbers it can handle. A desk calculator, for example, has an accumulator register of fixed size. An electronic analog computer operates over some limited voltage range. A mechanical analog machine has physically limited motions. A primary purpose of scaling (see Section 6.3, also) in any computer is to assure that intermediate results stay within specified ranges during the running of a problem. In a digital machine, this means the capacity of the registers must not be exceeded. It is also important to prevent crowding of results into a small segment of the numerical range of the registers (usually centered about zero) with an accompanying loss of significant digits, and in a DDA we must prevent the output of any operational unit from being systematically limited to insignificant changes.

The problem of scaling a DDA is similar to that of scaling an analog differential analyzer. Control of the scale may be achieved in a number of ways: by providing a facility that allows a choice of the number of significant digits employed in any given integrator, the use of constant multipliers, and digital servos with gain (see Section 8.7), etc.

A logical first step in scaling a problem is to estimate the maximum values each of the variables is likely to attain during the course of a computation. The more accurate this estimate, the better the solution. If the estimate is too low, integrators will overflow, and the problem will have to be rescaled. If the estimate is too high, more significant places will be used than required and hence it will take longer than necessary to obtain a solution. Of course, it is desirable to have all scales as great as possible for maximum accuracy.

8.5.4. SCALING RELATIONSHIPS WITHIN AN INTEGRATOR

Although a scale factor can be any number within a machine's range, restricting scale factors to integral powers of the machine's radix allows the product of scale factors, or a scale factor and a problem value, to be obtained by summing exponents. For a binary machine, the characteristic equation of an integrator is $\Delta z = y\,\Delta x/2^{n_i}$, for an integrand register of n_i bits. Since each machine variable is a product of a problem variable and a scale factor, a similar relation holds between the scale factors

$$2^{S_z} = 2^{S_y} \cdot 2^{S_x}/2^{n_i} \qquad (8\text{-}31)$$

and $$S_z + n_i = S_y + S_x. \tag{8-32}$$

Let the estimated maximum value of the integrand be $< 2^{m_i}$, where m_i is the smallest integer satisfying the relation. Therefore, the capacity of the integrand must be $2^{S_y} \cdot 2^{m_i}$. Accordingly, the second relation that must be satisfied is

$$(S_y + m_i) \leqslant n_i \leqslant N \tag{8-33}$$

where N is the maximum number of bits per accumulator (a characteristic of a particular machine design).

Equation (8-32) may be written

$$S_x - S_z = n_i - S_y. \tag{8-34}$$

Equation (8-33) may be written

$$m_i \leqslant (n_i - S_y). \tag{8-35}$$

It follows from Eqs. (8-34) and (8-35) that

$$(S_x - S_z) \geqslant m_i. \tag{8-36}$$

All the variables contributing to a particular input must be at the same scale. For example, all the *dy* inputs to a particular integrator must have the same scale. When considering the interconnections between integrators, it is also apparent that if a *dz* output is used as an input to another integrator, then, in general, the two must be of equal scale. However, the same variable may be treated as having a different scale at input and output, for special purposes.

8.5.5. Scaling Relationships for a Set of Integrators

We may distinguish between two sets of criteria to be satisfied in scaling. The first consists of satisfying the relations in Eqs. (8-32), (8-33), and (8-36), for all integrators, and thereby obtaining relative magnitudes for all the scales (principally from Eq. (8-36). The second consists of establishing a definite value for one of the scales. This will depend principally on the accuracy and computing speed desired. Among other factors that may influence the choice is the fact that some variable may require a certain minimum scale for a required accuracy. The accuracy with which maximum values of the integrands are estimated and the efficiency of scaling are primary factors in determining how effectively the machine's precision and computing time are utilized.

Part of the difficulty in scaling results from the fact that the scaling relations involve inequalities, as shown in Eqs. (8-33) and (8-36). In general, there are a number of sets of scaling factors that will satisfy a particular hook-up of integrators. A trial and error approach to satis-

faction of the constraints on scaling imposed by the machine and the nature of a problem can be very tedious for problems involving large numbers of integrators. However, an optimal set of scales can be produced systematically by a method in which these constraints are organized in a matrix form (see Gill [1959]). Once the scaling has been performed, the scaling relations can be used to determine the initial value of each integrand.

There are two possible criteria for fixing the scaling, and sometimes it is impossible to satisfy both. On the one hand, one may require that a particular variable have a specified precision, thereby fixing its scale and establishing all others. On the other hand, one can fix the time of computation which, in general, fixes the scale of the independent variable. This points out a distinguishing feature of the DDA, namely the option it offers of a direct trade-off between precision and solution time, one being proportional to the other. For example, in increasing the precision from one part in 2^{10} to one part in 2^{11}, the computing time required would be doubled. The result of scaling a problem is a determination of the register lengths for every integrator in the machine. Therefore, once a problem has been correctly scaled, the computation may be stepped up in accuracy or in speed by readjustment of all integrator lengths by the same amount (within the capacity of the registers).

The steps involved in scaling a problem can be summarized as follows: (1) Estimate the value of m (see Eq. 8-33) for each integrand. (2) From Eq. (8-36) establish a set of inequalities for all integrators, involving scale differences. (3) Upon some basis, i.e., the accuracy requirement of a particular variable, establish the numerical value of a particular scale. This plus the information in item (2) should allow all scales to be determined. (4) Equation (8-32) now permits the integrand length in bits, n_i, to be determined for each integrator. (5) As a check, we see whether Eq. (8-32) can be satisfied for each integrator. It may be that it is not satisfied for all integrators, as a result of the fact that the relations formed in accordance with Eq. (8-36) are inequalities rather than identities. In this case, from one to all of the scale factors may have to be changed. If possible, the scale on the independent variable should not be changed, since the time required for the solution of a problem depends on the scale (and range of interest) of the independent variable. (6) Determine the proper scaled initial value of each integrand.

8.5.6. Normalization

Normalization insures that several of the critical integrands will be full at their maximum values. Since the output rates of these normalized integrators will be increased each pulse will represent a smaller value of the original variable, and so the accuracy is improved. In practice, the

scales of the final output variables usually determine the necessary scale of the over-all problem and thus the running time. Normalization permits a shorter running time with no decrease in accuracy by permitting a decrease in the ratio of the input to output scales.

A simple procedure for normalizing equations is as follows. Consider the differential equation

$$dy'' = A\,dy' + B\,dy + Cy\,dx + Dx\,dx. \tag{8-37}$$

Normalized values of y'' and x are

$$y''_n = \frac{y''}{|y''|_{\max}} \qquad x_n = \frac{x}{|x|_{\max}} \tag{8-38}$$

where the denominators in (8-38) represent maximum absolute values in the original units, and y_n and x_n refer to normalized units for which the maximum value will be unity.

We will derive next the equivalent normalized form of Eq. (8-37). The appropriate normalization for the lower derivatives is obtained from Eq. (8-38) by performing the indicated integrations

$$y'_n = \int y''_n\,dx_n = \int \frac{y''}{|y''|_{\max}}\,d\left(\frac{x}{|x|_{\max}}\right) = \frac{y'}{|y''|_{\max}|x|_{\max}} \tag{8-39}$$

$$y_n = \int y'_n\,dx_n = \int \frac{y'}{|y''|_{\max}|x|_{\max}}\,d\left(\frac{x}{|x|_{\max}}\right) = \frac{y}{|y''|_{\max}|x|^2_{\max}}. \tag{8-40}$$

Equation (8-37) may now be written in normalized form

$$dy''_n = A|x|_{\max}\,dy'_n + B|x|^2_{\max}\,dy_n + C|x|^3_{\max}\,y_n\,dx_n + D|x|^2_{\max}\,x_n\,dx_n. \tag{8-41}$$

The highest order integrand is now automatically full at its maximum value. At least one of the other integrands in the problem may be adjusted to optimum or full by suitable choice of $|x|_{\max}$. The problem will not necessarily be run to the maximum value assigned to the independent variable.

Normalization provides clarity and convenience in the scaling process. In some problems it may be applied differently to different sections of the problem, and usually enables constants and maximum values to be adjusted so that scaling can be improved further.

8.5.7. Output Multipliers

Assume that the variable appearing in a Y accumulator has a known upper bound. If the absolute value of this upper bound is given by $|\alpha|$,

where $0 < \alpha < 1$, then the output rate of the integrator will not be over α times the maximum rate. This causes a decrease in the efficiency of operation of the integrator. Increasing the output rate could result in better scaling. This can be done, within the limitations of the over-all scaling constraints, by restricting the capacity of the R register. In the Bendix D-12 digital differential analyzer, circuitry is provided which gives the user the option of multiplying the output of chosen integrators by 1, 2, or 5, for the cases in which the bounds on Y are 1.0, 0.5, and 0.2, respectively. For these cases the bounds and initial values of R are as shown below

Output multiplier	Bounds on R	Initial value of R
$\times 1$	$0 \leqslant r_i < 1.0$	$r_0 = 0.5$
$\times 2$	$0 \leqslant r_i < 0.5$	$r_0 = 0.25$
$\times 5$	$0 \leqslant r_i < 0.2$	$r_0 = 0.1$

8.6. Decision Units in a Digital Differential Analyzer

It has been shown (in Section 8.3) how digital integrators may be utilized to generate approximations (to any precision desired) of analytic functions. With a slight modification, one or both accumulators of a digital integrator may be utilized to generate nonanalytic functions. An operational unit used this way is sometimes referred to as a decision integrator. However, a term like decision unit is better suited, since integration is not actually performed. The output of a decision unit may be utilized to generate nonanalytic functions to be used to represent limiters, hysteresis, backlash, static friction, absolute values, inert zones, etc. It may also be utilized to act as an automatic switching control at some point or points in the course of a computation.

Two types of decision units will be considered. Figure 8.11 shows a

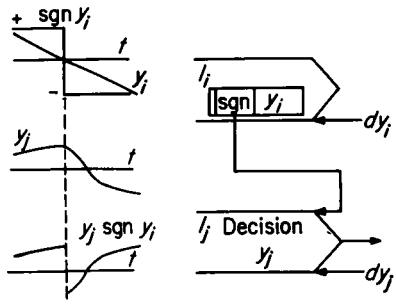


FIG. 8.11. Decision unit (type 1)

schematic of a decision unit which will arbitrarily be classed as type 1. This type of decision unit is identical to a digital integrator with one exception, namely, it has the capability of picking up as a *dx* input not only the normal incremental outputs of operational units or input devices, but also the signum function of the number in a *Y* accumulator. If each integrator is provided with this additional capability, it can be used either as an integrator or decision unit. A more general type of decision unit, referred to as type 2, is shown in Fig. 8.12; it does not use an *R* accumu-

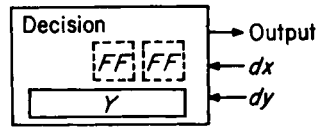


FIG. 8.12. Decision unit (type 2)

lator. Successive incremental inputs of the dependent variable are received by the *Y* accumulator, and in any iteration period, say the *i*th, the incremental output $(\Delta z)_i$, is determined by the incremental input of the independent variable, $(\Delta x)_i$, and the quantity, y_i, currently in the *Y* accumulator. The two flip-flops shown receive and indicate the three possible values of *dx*, namely + 1, 0, and − 1. The output $(\Delta z)_i$ is a function of y_i, in accordance with the following scheme

(1) If $y_i \geqslant 1$, $(\Delta z)_i = 0$

(2) If $y_i \leqslant -1$, $(\Delta z)_i = 0$

(3) If $y_i = 0$, $(\Delta z)_i = 0$

(4) If $0 < y_i < 1$, $(\Delta z)_i = (\Delta x)_i$

(5) If $-1 < y_i < 0$, $(\Delta z)_i = -(\Delta x)_i$.

Note that rules (1) and (2) enable the decision unit to be used as a limiter.

8.6.1. EXAMPLES OF USE OF A DECISION UNIT

We will consider first some examples of how nonlinear functions may be generated by the use of decision units: In Fig. 8.13, a decision unit is used to generate the absolute value of a function, *u*. Since, in any iteration period the same increment is used both as a *dx* and a *dy* input, the decision unit emits the absolute value of increments in *u* as its output, in accordance with rules (4) and (5) of the preceding paragraph.

FIG. 8.13.

In Fig. 8.14 two decision units are used to generate the sawtooth

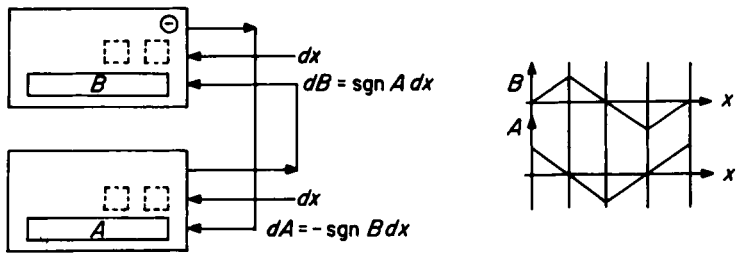


FIG. 8.14.

functions, A and B, also in accordance with rules (4) and (5). In Fig. 8.15 a decision unit is used to generate a clipped sine wave by use of

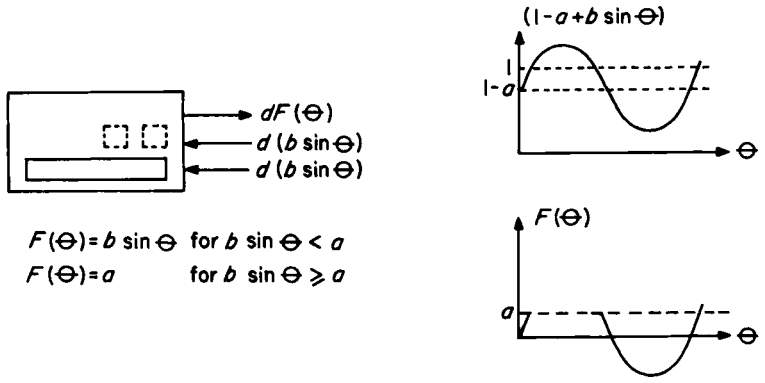


FIG. 8.15.

rule (1). The quantity 1 − a is entered initially into the Y accumulator so that y becomes equal to 1 when $b \sin \theta$ becomes equal to a.

Next we will describe examples of how decision units may be used for automatic switching operations. First, consider the case where it is desired to stop all or part of a computation when a variable, F, passes through zero, or exceeds specified limits. The decision unit acts as a switch, in that its output is used to stop computation when $F = 0$, or $F > k$. A mapping to achieve this is illustrated in Fig. 8.16. For $F < 0$,

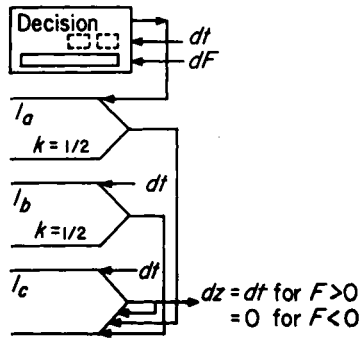


FIG. 8.16. Decision unit (type 2) controlled by a variable F

the output of $I_a = \Delta t/2$ since the output of the decision unit is $-\Delta t$. Since the output of $I_b = \Delta t/2$, the output of I_c will be $(-\Delta t/2 + \Delta t/2) = 0$. (This is because I_c is programmed to operate as a servo adder, therefore producing an output rate equal to the sum of its dy input rates; the internal operation of servo adders is described in Section 8.7). For $F > 0$, the output of $I_a = \Delta t/2$. Therefore, the output of I_c will be equal to Δt. If the output of I_c is used as the independent variable in certain parts or all of a problem, computation involving these parts will cease when F passes through zero. Next, consider the case where it is desired to stop certain parts of a computation and begin another (i.e., drop or add terms in an equation) when a variable, F, passes through zero, or exceeds certain limits; or when some variable F_2 becomes, say, greater than some other variable F_1. A mapping for the first case is shown in Fig. 8.17. From the figure one sees that the switching can be automatically achieved during the course of the computation if dt is used to generate functions to be used in all parts of the computation, and $dt - d\bar{t}$ to generate the functions to be used only during the time when $F < 0$.

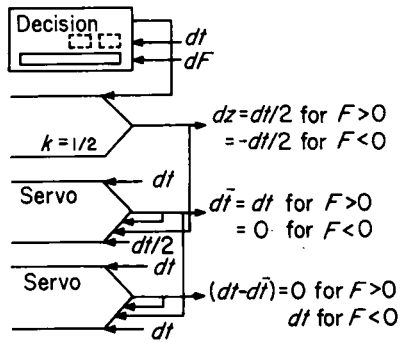


FIG. 8.17. Further example of use of decision units

8.7. Digital Servos

Though a separate accumulator or the Y accumulator of an integrator can accept two or more incremental inputs and accumulate their sum, it cannot generate a rate equal to the sum of the input rates. The addition of rates and the generation of values of implicit functions by function inversion may be performed by using an integrator programmed to function as a digital servo.

We will consider first how a digital servo can produce an incremental output equal to the sum of two inputs. An integrator may be programmed either as shown in Figure 8.19 (a), referred to as a "hard" or undamped servo, or as shown in Figure 8.19 (b), referred to as a "soft" or damped servo. The "hard servo" will be described first. Since its operation is governed by the number system used, the description will be related specifically to the number system usually employed in a DDA with binary communication. This number system, shown in Fig. 8.18, is described as circular because when an increment is added to the representation of the maximum positive number, $1 - 2^{-n}$, the result is the representation of the maximum negative number, -1, and conversely, subtracting a single increment from -1 produces $1 - 2^{-n}$. Ordinarily, the occurrence of overflows (at the two overload points) is avoided by accurately estimating the maximum value of each integrand in the scaling of a problem. Also, if an overflow of the Y register occurs, the machine is caused to stop. However, if one programs the machine to ignore the occurrence of an overflow, and utilizes the sensitivity of the circular number system at the overload points, an integrator can be used as a digital servo.

Assume that an integrand contains the representation of $1 - 2^{-n}$ and

Binary number				Decimal equivalent
Sign position	2^{-1}	2^{-2}	2^{-3}	
1.	1	1	1	7/8
1.	1	1	0	6/8
1.	1	0	1	5/8
1.	1	0	0	4/8
1.	0	1	1	3/8
1.	0	1	0	2/8
1.	0	0	1	1/8
1.	0	0	0	ZERO
0.	1	1	1	−1/8
0.	1	1	0	−2/8
0.	1	0	1	−3/8
0.	1	0	0	−4/8
0.	0	1	1	−5/8
0.	0	1	0	−6/8
0.	0	0	1	−7/8
0.	0	0	0	−1

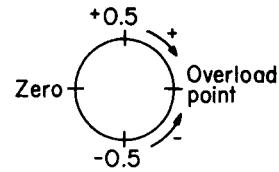


FIG. 8.18. Digital integrator's circular number system that facilitates servo operation

that the independent variable is time. Under these conditions, the integrator will produce a 1 at its output during each iteration period. The sequence of 1's thus generated represents the machine's maximum positive output rate. If a single bit is added in the least significant bit, the value of the integrand will change from $1 - 2^{-n}$ to -1 and the integrator will then generate a sequence of zeros, the maximum negative rate. As another example, assume the Y register is initially set to the value -1, and the difference of two quantities $(a - b)$ representing some error signal is accumulated in the register. If the value of the error is a small positive quantity, e, the quantity stored in the register will be $-1 + e$. The machine interprets this as being close to the maximum negative number, and the integrator output will be close to the maximum negative output rate. On the other hand, if the error is negative, the integrand value will be $1 - e$ which the machine interprets as being close to the largest positive number, and the integrator output is close to the maximum positive output rate. Thus, we see that a suitable initial setting of the contents of the Y register will cause the dz output to be the same as the signum function of y, and, by a small change in the integrand, the output of an integrator can be caused to vary from the largest positive to the largest negative rate, or vice versa. The initial setting is equal to $b - b^{-n}$ or

0.000 . . . 0, which represents in a complementary number system with a radix b the maximum positive and negative values, respectively.

We will illustrate now how to generate a dz output equal to the sum of two dy inputs, da and db. Assume these inputs are fed into a Y accumulator. Normally, under these conditions, the quantity $a + b$ would be accumulated. If, however, an initial value of $1 - 2^{-n}$ is set into the Y accumulator and the dz output of the integrator is fed back into its own integrand, as shown in Fig. 8.19(a), a considerably different result will

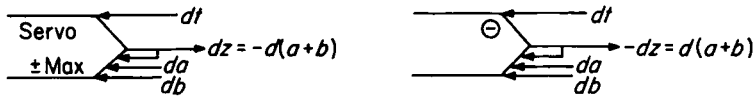


FIG. 8.19. (a) Servo adder, (b) Damped servo adder

be obtained. The dx input to the integrator is designated as dt (t is time) because it consists of a sequence of 1's occurring at the iteration frequency of the machine (which is also the maximum output rate that can be generated). Each occurrence of the dt input causes the contents of the Y register to be added to R. We will assume that initially the inputs have no effect on the integrand. (For a single input, in a system with binary communication, this would mean a sequence of alternate 1's and 0's; for the sum of two inputs to be zero in all iteration periods, their signs must always be opposite.) If the initial value of the integrand is $1 - 2^{-n}$, the dz output during the first iteration period will be positive. When fed back during the next iteration period it changes the integrand from $1 - 2^{-n}$ to -1. As a result, the next output of the integrator will be a negative increment which feeds back converting the value of the integrand to $1 - 2^{-n}$. The value of the integrand will, therefore, oscillate between -1 and $1 - 2^{-n}$, and the output of the integrator will be a zero rate. Now, assume that the da input and the db input each increase by a single increment. This will cause the contents of the integrand to change from $1 - 2^{-n}$ to -1 to $-1 + 2^{-n}$. The negative integrand results in a negative dz output. When fed back during the next iteration period, it changes the integrand from $-1 + 2^{-n}$ to -1 (assuming a net input of zero from da and db during this period). Under the same input conditions, the integrand is changed from -1 to $1 - 2^{-n}$ at the end of the next period. Thus, the feedback signals bring the integrand back to its initial condition, and a net output of two negative increments is produced, representing the sum of the da and db inputs during this period. Since the servo can-

not produce more than one output per iteration period, the correspondence of the sum of its outputs to the sum of the *da* and *db* inputs, over *n* iteration periods, depends on whether the sum is greater than *n*.

The output of the servo adder shown in Fig. 8.19(a) consists of bunched groups of pulses. The damped servo adder shown in Fig. 8.19(b) generates a steadier output rate at the cost of a delay between the input and output. Since the damped integrator servo causes an exponential decay of the error introduced, it may be used to smooth irregular inputs. This action is evident if we consider the schematic of Fig. 8.19(b) and assume the inputs on lines *da, db* to be temporarily zero. Then disregarding the input lines *da, db,* the schematic describes the familiar arrangement for generating e^{-t}, and, therefore, the value of the integrand (representing the current error) will decay exponentially. Since the output of the damped integrator servo is a function of the error rather than on–off (depending on the sign of the error) it may be used as a variable gain device in a servo loop. When a servo error exceeding a few pulses is allowable, the damped integrator servo should be used to decrease oscillations occurring in the servo system.

As mentioned previously, a digital servo can generate only a single output in an iteration period. The servo is said to be overloaded if the algebraic sum of incremental inputs received in any interval is greater than the maximum absolute value of outputs the servo can produce in that interval, namely the sum of a sequence of all positive or all negative increments occurring at the iteration rate of the computer. The net effect of an overload is a delay between the time by which a number of inputs are received and the time when the sum of the servo's outputs equals this number. The extent of the delay depends on the degree of overload and its duration.

If, for a particular problem, the sum of the outputs of the sources supplying a servo are greater than the full rate, the rate of the independent variable must be decreased, while the *dt* rate is left undisturbed. Then, the input rates to the servo will correspondingly decrease, allowing it to keep up. To facilitate changing the rate of the independent variable in all parts of the problem, the incremental sequence, *dt* can be used as the input to a constant multiplier. The output of this constant multiplier is then taken to be the independent variable, and its rate can be changed simply by changing the value of the constant in the multiplier.

It is at times useful to have a servo unit for the purpose of changing the scale factor of a variable in order to facilitate scaling a problem. Such a servo unit is often referred to as a servo with gain. An example is illus-

trated in Fig. 8.20. The sum of the dz outputs is equal to $1/k$ times the

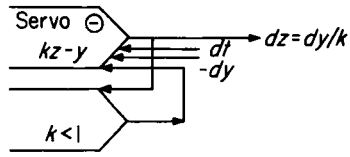


FIG. 8.20. Servo with gain

sum of the dy inputs, where k is an arbitrary constant less than one. It is clear from the figure that the servo unit will continue to generate outputs until $(kz - y)$ becomes equal to zero, at which time $dz = (kz - y)\,dt$ will become equal to zero. For this to occur, the sum of the dz outputs must be equal to $1/k$ times the sum of the dy inputs.

Digital servos are useful not only as adders to generate a pulse rate equal to the sum of two or more rates, but for performing such operations as function inversion and the solution of equations. Use of the basic digital servo for these more complex operations requires the inclusion of other elements into the feedback loop between the output of the servo and the input to its own integrand. As a first example, consider the generation of y as a function of x where y is defined implicitly as a function of x by $F(x, y) = 0$. Figure 8.21 shows the general operation of a servo in which

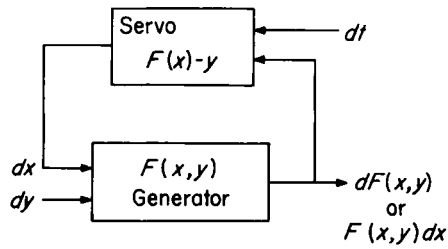


FIG. 8.21. Use of a digital servo for function inversion

$dF(x, y)/dy$ must be nonzero and the output sign of the servo must be the opposite of the sign of $dF(x, y)/dx$. Under these conditions, and if there is not excessive delay in the feedback network, when incremental changes in y cause $F(x, y) \neq 0$, the servo will generate incremental changes in x until $F(x, y) = 0$. The block labeled $F(x, y)$ generator, refers to one or more operational units that generate $F(x, y)$ from dy and the output of the servo, which is assumed to be dx. The functions may be

either algebraic or differential equations. The independent variable is shown as y, but considerable judgement will be required to determine which variable should be considered independent, and what sign should be used in the servo. The method may be extended to simultaneous equations in three or more variables. The decision as to choice of independent variable, sign of servo, form of schematic, etc. will be correspondingly involved, and one must also have an initial idea of the general form of the answer. Also, it is assumed that correct initial conditions are known.

A number of inverse functions, like the square root, inverse trigonometric quantities, a quotient, etc., may be generated more economically if digital servos are utilized in addition to conventional integrators. The next example demonstrates this point. Assume the following equation is to be solved for θ

$$a = b \cos \theta \tag{8-42}$$

where a and b are variables. Figure 8.22 shows an arrangement for

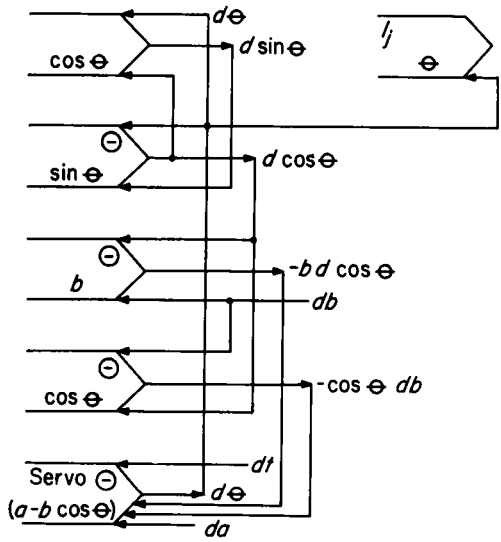


FIG. 8.22. Generation of $(\cos^{-1})$ utilizing a digital servo

generating θ. This is another example of the well known boot strap method: The output of the servo unit, assumed to be $d\theta$, is then used to generate $b \cos \theta$, which is fed back as one of the dy inputs to the servo unit. At some point $(a - b \cos \theta)$ becomes equal to zero, at which point the output of the digital servo becomes equal to zero. For this to have

occurred, the servo must, in fact, have generated the correct value of θ, which now can be read as the value in the Y accumulator of integrator, I_j. To illustrate the economies effected by this procedure, consider how the above problem can be solved without the use of servos. Equation (8-42) may be written

$$\cos\theta = a/b \tag{8-43}$$

$$\theta = \cos^{-1} a/b. \tag{8-44}$$

Taking derivatives on both sides of Eq. (8-44)

$$d\theta = \frac{-d\,(a/b)}{\sqrt{1-(a/b)^2}},\ 0 \leqslant \cos^{-1} a/b \leqslant \pi \tag{8-45}$$

$$d\theta = \frac{a\,db - b\,da}{b^2\sqrt{1-(a/b)^2}}. \tag{8-46}$$

The generation of $d\theta$ by means of Eq. (8-46) requires generation of several auxiliary functions (squares, square roots, quotients). As a result much more computing capacity is required than if a digital servo were utilized.

Figures 8.23(a), (b), (c), and (d) illustrate mappings which utilize digital servos to generate functions involving a division operation. In each of these examples, the inputs to the servo adders are like functions of opposite sign which are derived from different points in the system. The output of the servo, assumed to be the function required, is used in generating one or both of the servo's inputs.

Both "hard" and "soft" servo action can be obtained regardless of whether binary or ternary increments are used. Also, use of an integrator is not essential, for servo action can be obtained from a single accumulator by driving it to zero by means of a negative correction when it holds a positive value and vice versa. The accumulator is initially set to zero and during each iteration period it may receive one or more incremental inputs. These are accumulated and, during each iteration period, the output of the accumulator is a single increment whose sign is determined by the current contents of the accumulator. This type of operation can be programmed in a specific machine only if there is provision for it in the design of the machine. If decision units of the type shown in Fig. 8.12 are provided in the machine, they, too, may be programmed to operate as servos. The output of such a unit will be either positive or negative in accordance with the sign bit as long as the contents are nonzero, and zero when the contents of the accumulator have been driven to zero.

8.8. Error Analysis for an Incremental Multiplier

When feedback loops are present in an integrator network, its analysis becomes cumbersome because of the presence of remainders in the R

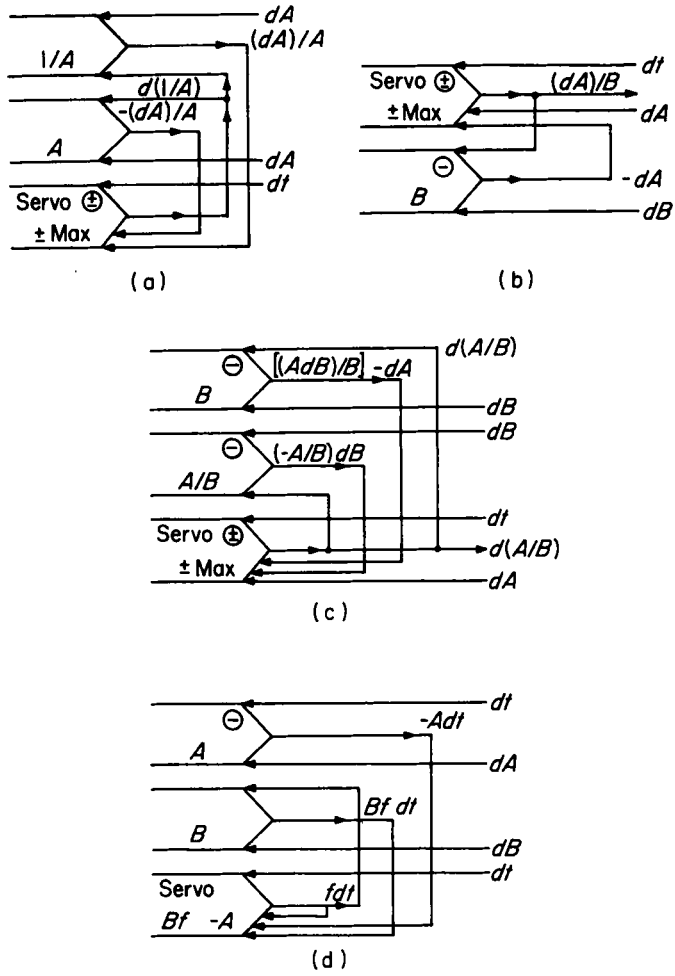


FIG. 8.23. Generation of (a) $d(1/A)$, (b) $(dA)/B$, (c) A/B, (d) A/B

registers. However, analysis is not too difficult for simple nonfeedback networks, e.g., the multiplication network shown in Fig. 8.24. Assume that dA

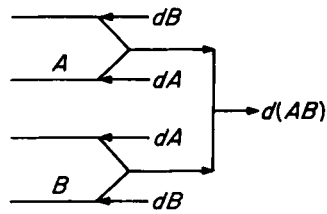


FIG. 8.24.

and dB are each restricted to unit increments, and that the scales of the dz outputs are the same (so that the dz outputs may be added directly), After k iterations, the value of the summed dz outputs is (Eq. (8-25))

$$\sum_{i=1}^{k} \Delta z_i = \frac{2}{N} \sum_{i=1}^{k} y_i \Delta x_i + e. \tag{8-47}$$

If in Eq. (8-26) one replaces $(y_0 + \sum_{j=1}^{i-1} \Delta y_j)$ by its equivalent y_{i-1}, one can obtain the following expression for $\sum_{i=1}^{k} y_i \Delta x_i$

$$\sum_{i=1}^{k} y_i \Delta x_i = \sum_{i=1}^{k} (y_{i-1} + F(\Delta x)_i \, \Delta y_i) \, \Delta x_i \tag{8-48}$$

The increment to the product $A_0 \, B_0$, which will be designated as ΔP, is the sum of contributions from both integrators (see Fig. 8-24).

$$\Delta P = \left(\sum_{i=1}^{k} \Delta z_i \right)_1 + \left(\sum_{i=1}^{k} \Delta z_i \right)_2 \tag{8-49}$$

$$= \frac{2}{N} \left\{ \sum_{i=1}^{k} (B_{i-1} + F(\Delta A_i) \, \Delta B_i) \, \Delta A_i + \sum_{i=1}^{k} (A_{i-1} + F(\Delta B_i) \, \Delta A_i) \, \Delta B_i \right\} + e_1 + e_2 \tag{8-50}$$

$$= \frac{2}{N} \left\{ \sum_{i=1}^{k} (B_{i-1} \, \Delta A_i + A_{i-1} \, \Delta B_i) + \sum_{i=1}^{k} (F(\Delta A_i) + F(\Delta B_i)) \, \Delta A_i \, \Delta B_i \right\} + e_1 + e_2 \tag{8-51}$$

From the equalities

$$\begin{aligned} A_i &= A_{i-1} + \Delta A_i \\ B_i &= B_{i-1} + \Delta B_i \end{aligned} \tag{8-52}$$

it follows that

$$A_iB_i = A_{i-1}B_{i-1} + B_{i-1}\Delta A_i + A_{i-1}\ \Delta B_i + \Delta A_i\Delta B_i \qquad (8\text{-}53)$$

Rearranging Eq. (8-53) yields

$$B_{i-1}\Delta A_i + A_{i-1}\Delta B_i = A_iB_i - A_{i-1}B_{i-1} - \Delta A_i\Delta B_i \qquad (8\text{-}54)$$

Substituting the right hand side of Eq. (8-54) into (8-51)

$$\Delta P = \frac{2}{N}\left\{\sum_{i=1}^{k}(A_iB_i - A_{i-1}B_{i-1} - \Delta A_i\Delta B_i) + \sum_{i=1}^{k}(F(\Delta A_i) + F(\Delta B_i))\ \Delta A_i\Delta B_i\right\} + e_1 + e_2 \qquad (8\text{-}55)$$

$$= \frac{2}{N}(A_kB_k - A_0B_0) + \sum_{i=1}^{k}(F(\Delta A_i) + F(\Delta B_i) - 1)\ \Delta A_i\Delta B_i + e_1 + e_2 \qquad (8\text{-}56)$$

The round off errors e_1 and e_2 are unavoidable but we are interested in considering what choices for $F(\Delta x_i)$ will reduce the term $\Sigma(F(\Delta A_i) + F(\Delta B_i) - 1)\Delta A_i\Delta B_i$, which may be considered as the error E in the computation of a product, to zero in the general case. Corresponding to the four choices for $F(\Delta x_i)$ described in Section 8.3.3, namely 1) $F(\Delta x_i) = 0$, 2) $F(\Delta x_i) = 1$, 3) $F(\Delta x_i) = \frac{1}{2} + \frac{1}{2}\ (\Delta x_i)$ and 4) $F(\Delta x_i) = \frac{1}{2}$ are the following expressions for E

$$\text{Case 1: } E = -\sum_{i=1}^{k}\ \Delta A_i\Delta B_i \neq 0$$

$$\text{Case 2: } E = \sum_{i=1}^{k}\ \Delta A_i\Delta B_i \neq 0.$$

Case 2 corresponds to a rectangular integration scheme using the new value of y, i.e., y_i. Two such integrators, interconnected as shown in Fig. 8.24, would generate

$$\Delta(AB) = [B_{i-1} + (\Delta B)_i](\Delta A)_i + [A_{i-1} + (\Delta A)_i](\Delta B)_i$$
$$= B_{i-1}\,(\Delta A)_i + A_{i-1}\,(\Delta B)_i + 2(\Delta A)_i\,(\Delta B)_i. \qquad (8\text{-}57)$$

From Fig. 8.25 it is readily seen that the error produced results from counting the area $(\Delta A)_i(\Delta B)_i$ twice.

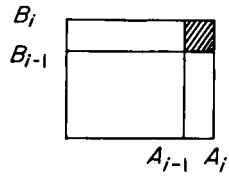


FIG. 8.25.

Case 3:

$$E = \tfrac{1}{2} \Sigma \Delta A_i^2 \Delta B_i + \tfrac{1}{2} \Sigma \Delta B_i^2 \Delta A_i$$

$$\text{Since } \Delta A_i^2 = \Delta B_i^2 = 1$$

$$E = \tfrac{1}{2} \Sigma \Delta B_i + \tfrac{1}{2} \Sigma \Delta A_i \neq 0$$

Case 4: $E = 0$.

Case 4 corresponds to a trapezoidal integration scheme. Two such integrators interconnected as shown in Fig. 8.24 would generate

$$\begin{aligned} \Delta(AB) &= (A_{i-1} + \tfrac{1}{2}\,\Delta A_i)\Delta B_i \\ &\quad + (B_{i-1} + \tfrac{1}{2}\,\Delta B_i)\Delta A_i \\ &= A_{i-1}\,\Delta B_i + B_{i-1}\,\Delta A_i + \Delta A_i\,\Delta B_i. \end{aligned} \tag{8-58}$$

The successive dy inputs are accumulated as before. However, during an iteration period in which the dx input differs from zero, say the ith, the quantity added to or subtracted from the contents of the R register is the current contents of the Y register plus $(\Delta y)_i/2$. At the end of the ith period, the Y register holds $y_i = y_{i-1} + (\Delta y)_i$, but the quantity added to, or subtracted from, the R register is $y_{i-1} + (\Delta y)_i/2$ when $(\Delta x)_i \neq 0$.

There is another case for which $E = 0$, namely, if $F(\Delta x_i) = 0$ (case 1) is chosen for one integrator and $F(\Delta x_i) = 1$ (case 2) is chosen for the other.

8.9. More Complex Operational Units

An important application of digital computers is the automatic generation of navigational data in vehicles traversing great distances under

conditions which make accurate determination of position, velocity, etc., at the rate required, difficult or impossible by other means. A computer can provide a high-speed link between measuring instruments and controls which enable it to change its position in accordance with current requirements. The computer receives its inputs from such sources as radars, gyro compasses, air speed indicators, roll and pitch indicators, barometric and radar altimeters, etc. It transforms this data to produce outputs in the form of control signals to the auto-pilot, automatic tracking signals to the radars, display signals, etc. Derived quantities include present position in terms of latitude and longitude, distance to destination, etc.

Essentially, the navigation problem consists of resolving a vehicle's motion, as sensed by its instruments, along the axes of some chosen coordinate system. The computational problem is, therefore, basically that of solving trigonometric and algebraic equations, plus the integration of certain variables. The mathematical operations that will enter into the computation may be classified as follows: (1) addition, subtraction, multiplication, and division; (2) generation of trigonometric functions (which may be restricted to generation of sines, cosines), and inverse trigonometric functions; (3) integration; (4) simple function generation: exponentials, absolute values, etc.; (5) analog–digital conversion, and digital–analog conversion.

The computations necessary to derive the desired output quantities may be performed either by an integral transfer (GP) computer or an incremental computer. Both types have been built for this function. However, the nature of the problem particularly lends itself to an incremental technique, since the variables involved are, in general, continuous and have limited rates of change.

Because of the high relative frequency with which the operation of multiplication is performed in this application, it is important that an efficient and accurate method of performing multiplication be provided. The method of performing multiplication depicted in Fig. 8.3 (m) takes two integrators and a total of four registers (two R and two Y registers) to generate the incremental change in the product of two numbers. An obvious way to increase the speed of multiplication by a factor of two would be to use two sets of integrators in parallel. This would require additional R line and an additional Y line. Other approaches may be more desirable. One alternative is to use a three line structure as shown in Fig. 8.26. This configuration may be arrived at by recognition of the fact that the use of two remainder registers in the generation of a product (or sum of squares) is redundant.

By providing for storage of the output of one of the Y registers (i.e.,

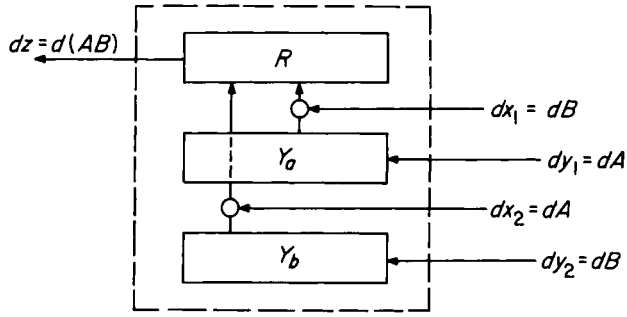


FIG. 8.26. Block diagram of a three-line structure programmed as a multiplier.

the value of the sign bit) as well as for that of the R register overflow, and by incorporating additional switching circuitry into the design, the three register unit may be programmed to provide (in addition to a product or sum of squares) both the integration of a variable and a servo operation, as shown in Fig. 8.27. Specifically, registers Y_a and R can be

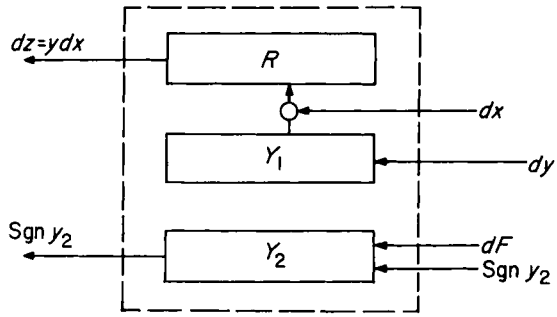


FIG. 8.27. Block diagram of a three-line structure programmed as an integrator and servo register.

organized to form a conventional integrator, and register Y_b can be used as a servo register. The servo register may be used to store the value of an input or output quantity in an analog-to-digital or digital-to-analog conversion loop, as well as operate in a servo computational loop.

A useful operation which is utilized frequently in a class of navigational problems is that of rotation of a vector (see Fig. 8.28). This operation may be written as

$$A' = A \cos \theta + B \sin \theta$$

$$B' = B \cos \theta - A \sin \theta$$

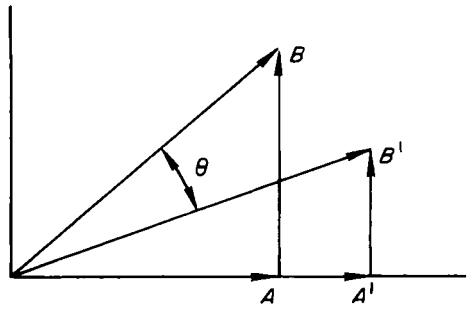


FIG. 8.28. Rotation of a vector

or

$$dA' = A\,d(\cos\theta) + \cos\theta\,dA + B\,d(\sin\theta) + \sin\theta\,dB$$

$$dB' = B\,d(\cos\theta) + \cos\theta\,dB - A\,d(\sin\theta) - \sin\theta\,dA.$$

If only the two-line integrator structure were used, four integrators would be required to generate the terms in each component of the vector. With the five-line structure depicted in Fig. 8.29 all four terms, plus an output rate equal to the sum of these four rates may be generated in one word time. The increased capacity of the three and five line structure is paid for by having to provide for more registers and associated circuitry in parallel. However, the efficiency is greater than that of the two-line structure because of the elimination of redundant remainder registers (see Fig. 8.29).

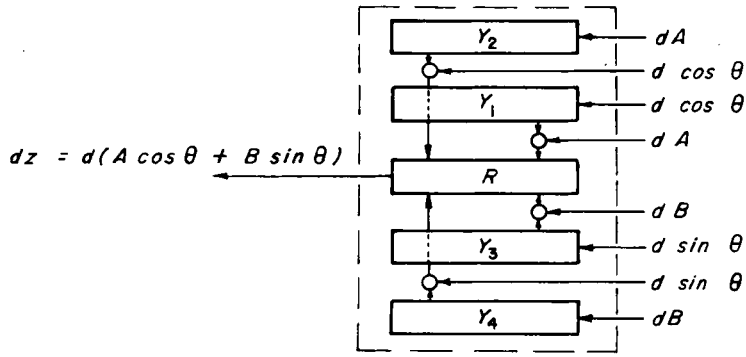


FIG. 8.29. Block diagram of a five-line operational unit.

If, in addition to providing for storage of the output of one of the Y registers (as in the three-line structure), some of the registers in the

five-line structure are made capable of operating either as an integrand register or a remainder register, and the required multiple incremental storage is provided for each operational unit, it becomes possible to program a single operational unit to operate also as follows: (1) to generate both a multiplication and an integration, (2) to integrate two functions and perform a servo operation, etc.

8.10. Limiting Communication in a Special Purpose DDA

For a machine designed to solve only a particular problem (e.g., a computer designed to operate in a specific control system as opposed to one designed to compute solutions to a variety of problems), a saving in equipment may be effected by restricting the communication between operational units. Since no operational unit receives inputs from more than a few, say five or six, of the other operational units, it is not necessary for each unit to be able to scan the outputs of all other units. However, the problem of assigning operational units for the solution of a large number of equations in such a way as to permit each operational unit to receive all its required inputs can be a cumbersome, trial and error task. Of course, the more limited the access to the Δz store, the probability of a particular desired input being available to an operational unit (assuming all inputs to be equally likely), and the more formidable the task of mapping (see Section 8.5.2).

In general, it may not be possible to assign operational units so that the inputs required by each are available from the Z line during the scanning time T_i. If this is the case, one solution is to use so-called repeater or relay stations as follows: Assume that after several "jugglings" of operational unit assignments, there is still some unit, j, which must receive an input from unit m, but that the output of unit m is not available from the Z line during the time interval T_{j-1} when unit j can receive its inputs. If there is some operational unit, r, which can receive an input from unit m, and whose output, in turn, can be picked up by unit j, then unit r (programmed as a constant multiplier with $k = 1$) can be used as a repeater link. Preferably, unit r should be where it can pick up the most recent output of unit m, and its most recent output should be available to unit j. Whenever it is considered economical to limit communication in a special purpose computer, both the specific mathematical formulation of the problem and programming requirements must be taken into consideration.

8.11. Applicability of the DDA

Various departures have been taken from the mechanization of a DDA depicted in Fig. 8.8 for the purposes of improving computing flexibility, ease of programming and computing speed. Some variants in operational unit structure for special purpose machines were described in Section 8.9. Use of a static store for the Δz's can alleviate addressing problems, described in the preceding section and on page 476. Some recent designs, with plugboard programming and higher iteration rates, will be mentioned briefly. The CORSAIR computer (Owen *et al.* [1960]) has an iteration rate of 500/sec, obtained by use of a ferrite core store for Y, R and Δz in conjunction with a single time-shared arithmetic unit. In the TRICE computer (Mitchell and Ruhman [1958]) an iteration rate of 100,000/sec is produced by use of separate active-element registers and an arithmetic unit for each integrator. In the SPEDAC-310 (Bradley and Genna [1962]) an iteration rate of 1,000,000/sec is achieved by use of separate active-element registers and a serial-parallel arithmetic unit for each integrator.

In control applications a computer must be able to analyze and generate data fast enough to keep up with changes in the physical environment which it is monitoring and aiding to control. A computer's maximum frequency of sine wave generation (in cycles/sec) is a convenient, though approximate, guide to the highest frequency of change in a computed function that can be accommodated for a second order system.

For a DDA, this upper frequency may be computed from the expression $RI/2\pi$ where R is the resolution and I the iterations/sec. (To compensate for round-off and truncation error, the number of bit positions used in the integrators may be greater than called for by the resolution.) For accuracies of .01% and .1% the upper frequencies are .0016 and .016, respectively, for a serial DDA with $I = 10^2$; .016 and .16 for a serial-parallel DDA with $I = 10^3$; 15.9 and 159 for a parallel DDA with $I = 10^6$.

For a GP machine of the IBM 7090 class, the upper frequency of sine wave generation for an accuracy of either .01% or .1% is about .1 to .3. These figures are based on generating $\sin\theta$ for all values of θ defined by the increments $\Delta\theta$ in the DDA. They are approximate since the multiplication time varies with the number of 1's in the multiplier and various options in programming details. The lower figures are for a three term Chebyshev polynomial approximation to $\sin\theta$ (see Section 6.2.5 and Hastings [1955]); the higher figure is based on computing increments to the sine and cosine, as in a DDA. For any significant problem the figures for a GP machine would be considerably less since the arithmetic unit would be available for the sine computation only a fraction of the total computational cycle.

For a high precision analog computer the upper frequency is about 3 at .01% accuracy and in the neighborhood of 100 for .1% accuracy. These figures are only approximate guides because the upper frequency varies with the nature of the over-all problem.

On pages 452-453, 476 and 499 certain capabilities of the DDA are described. At this point let us consider briefly the outlook for machines of this type. First of all, applicability of the DDA does not depend on whether integration appears explicitly in a problem. The distinguishing characteristic of this type of machine is that it operates in an incremental manner, with limits in the size of these increments. Serial and serial-parallel DDA's are economical to use in the control of systems where the variables change only in a limited and continuous manner, e.g., in airborne navigation and flight control, weapons control, missile guidance, the control of certain industrial plant processes. Outside of specialized military applications, the use of incremental machines has thus far been limited—for good reasons. In computing installations, much work does not require solutions over a continuum, but for a small set of values of the input variables; also, business accounting as well as other types of data processing problems for which a DDA is not well suited (see Braun [1960]) must usually be handled. For problems calling for continuous solutions, but not high accuracy, an inexpensive electrical analog computer is usually more economical. In the area of real time simulation of high performance systems, incremental machines competitive with the best analog computers for high speed, high accuracy integration were not available until recently. Now, a DDA with an iteration rate of 10^6/sec is even adequate for faster than real time (high speed) computation, so it can be used for making predictions sufficiently in advance to allow corrective action.

8.12. Sources of Error

In any computing machine, errors may arise from two distinct sources —those inherent in the nature of the machine and those peculiar to a particular problem. It is not always possible to isolate the source. Sources of error in a digital differential analyzer which are inherent in the nature of the machine are as follows: (1) round-off error (common to all computing machines; (2) truncation error; (3) the logical characteristics of computing algorithms used; (4) phase effect errors; (5) lags produced by feedback connections and serial processing of operational units. The relative importance of each type of error depends on the characteristics of individual machines, and special facilities, if any, that have been incorporated into its design to minimize the effects of certain errors. For example, a particular design may give the user the option of

using one of several integration formulas (see Section 8.3.3) according to which he judges best for the solution of a specific problem.

For a general discussion of round-off error (common to all computing operations because of the finite length of numbers carried) refer to Section 9.4. In an incremental computer there is a round-off error due to the fixed length of the Y register, and because the remainder in the R register is neglected in reading the current value of z. The magnitude of round-off error assumes greater importance in problems where two variables (u, v) are nearly equal and their difference is important to the problem. In these cases, the equations should be expressed in terms of u and u-v rather than in terms of u and v separately. (In servo system problems this may require solution of the open loop rather than the closed loop.) Truncation error arises because higher order terms, as expressed in a power series representation of y, are neglected in the integration formula. For example, the rectangular integration formula assumes y is constant during each increment of the independent variable, while the trapezoidal takes into account only the first-order difference.

Phase error and its relation to the computing algorithm, certain types of error resulting from the serial nature of the computer, and errors produced in the generation of certain functions will be treated later in this section. First, we will describe briefly certain errors which result from the nature of the problem and the particular way it is programmed.

Sometimes an idealized model of a physical system may result in a mathematical description which, in its deviation from a true physical situation, introduces troublesome anomalies. Examples of situations that cannot occur physically are representation of an acceleration by a step function, or a force by an impulse function. In the case of static systems discontinuities can arise, so generation of a solution may require introduction of certain devices and/or approximations. It is usually helpful to have information beforehand concerning the form of the solution and the nature of the variables involved in the problem. Also, since the form of equations can often be changed to better suit the characteristics of the DDA, it is important that the user of the computer be as familiar with the functional characteristics of the particular machine he is using as he is with the nature of the problem he is trying to solve.

The mathematics of a problem may be set up in different forms, arising from changes of variable and parametric methods. For example, dependent and independent variables may be interchanged by means of the relation

$$u\,dv = d(uv) - v\,du.$$

There is, in general, more than one mapping of an equation or set of

equations that will generate a solution. One will not always be able to predict whether one mapping has a significant advantage over another. A good choice depends on the user's familiarity with the nature of the problem, his mathematical intuition, his analysis of the mapping to detect sources of difficulty, and his ability to employ corrective measures to compensate for errors peculiar to the mapping or logical structure of a computer. There are cases where it is obvious that a particular mapping will give trouble. For example, a mapping in which an integrand appears containing $1/x$ cannot be used in the region where x is small or zero, since the integrand becomes infinite at $x = 0$. In such a case an alternate map may often be found which utilizes a digital servo, and which can be used in the region where x is small.

In a complex problem, a careful arrangement of the order of the integrators can be effective in reducing errors to much less than would be the case if the integrators were arranged at random (see Section 8.5.2).

A change in scale of the independent variable necessitates changing the lengths of all variable integrands and affects the scaling relationships between the computer and external devices. Increasing the scale factor of the independent variable (which decreases the size of the step in the integration formula) makes a more accurate solution possible. Not only is there a reduction in the integration step but also a reduction of round-off error in all variable integrands, and a lesser effect from errors introduced into the least significant digit position.

Consideration of Fig. 8.30 will be helpful in assessing the effect of increment size on the output of an individual integrator. It is assumed

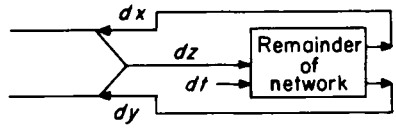


FIG. 8.30

that the dx and dy inputs of the integrator are the outputs of other operational units in the hook-up used to generate a problem solution, and that the dz output of this integrator feeds into the network. (Other inputs and outputs of the network are not shown.) The prime mover of the network is the independent variable input dt.

If the scale of dt is increased, the scale of dx and dy and also the length of the integrand register will be increased. If the length of the integrand register cannot be decreased, (and it is assumed that the scales of the integrators are such that the error contribution of each is the same)

little can be gained by other scaling adjustments. If only the scale of dy or dx were increased, when both inputs contribute a comparable error, there would be no appreciable effect. Since an integrand register should become as full as possible without producing an overflow, its length should be decreased if possible (implying that, before the change, the position in this register to the right of the binary point, at least, would always have been zero). Although the length of the integrand register indicates the degree of resolution to which a variable is accumulated, the computed function may not be this accurate because various sources of error may significantly degrade it. Nevertheless, precision can be used as a measure of relative accuracy since increasing it will increase the accuracy; also, where the contribution of error sources other than round-off and truncation error is negligible, the precision serves as a guide to the accuracy.

By efficient scaling (including such devices as dividing a solution into several parts, each with efficient scaling in a specified interval where one or more variables varies over a wide range), normalization and combining of constants, one can diminish the relative effect of errors. Devices which increase the rate of an incremental output improve the accuracy of the corresponding variable, though this is not generally true when servos are used. To prevent overloading of servo adders, without reducing overall accuracy, the rates of their inputs should be reduced, if possible, without reducing the scale of the independent variable.

There is a type of error peculiar to incremental computers, known as "phase" error, which results from the relationship between the dx and dy increment sequences. It may best be illustrated by specific examples. Assume that in a computer with binary transfer the function y is constant and $\Delta x/\Delta t = 0$. Since a zero-rate sequence in this type of machine consists of alternate positive and negative increments, y will be alternately augmented and diminished by a single increment even though, theoretically, it should stay constant. It at each step Δx and Δy are in phase (i.e., of the same sign), the following series of quantities will be added to the R register: $(y + \Delta y)\Delta x$, $-y\Delta x$, $(y + \Delta y)\Delta x$, $-y\Delta x$. . . . If Δx and Δy are out of phase the sign of each term in the series will be changed. The net result is a spurious introduction of $\Delta y\Delta x$ (or $-\Delta y\Delta x$) every two steps of the iteration, resulting eventually in a spurious dz output. The magnitude of Δy may be one or more increments depending on whether it is generated by one or more integrators. For the special case of both $\Delta x = 0$ and $\Delta y = 0$, phase error may be eliminated by the following correction scheme: If Δx_{i-1} is positive and Δx_i is negative, add $-y_{i-1}$ rather than $-y_i$ to the R accumulator. This scheme also tends to reduce phase error when $\Delta y \neq 0$. In another scheme, algorithm $y_i = y_{i-1} + (\Delta y_{i-1} + \Delta y_i)/2$ is used so when the current and preceding inputs are opposite in sign, the

value of y is left unchanged. The first of these schemes allows the generation of sin t, cos t without variation in the limits between which these functions oscillate. However, if the second scheme is used as well, phase error is introduced. Although various logical and programming devices may be used to reduce phase error, none has been found to be the most effective for all situations. Analysis of phase error is difficult because it requires knowledge of the behavior of the dx and dy inputs at all times.

In a machine with binary communication (see Section 8.3) if the R registers are left empty at the start of a problem negative increments are generated initially by all integrators and this bias may significantly affect the final result. A simple procedure that helps to compensate for this effect is to fill some average value into the R register. A more precise approach is to set R to an exact value. This value may be determined by recalling that the value of each integrand represents an accumulation of dz outputs from one or more integrators. If only one dz is involved, the initial condition of the integrand can be computed to more places than accommodated by the Y register, and the less significant bits placed in the R register whose overflow feeds it.

In a serially-organized DDA a start-up error is also produced because of a lag in the production of outputs by higher numbered integrators. In other words, any integrator with an input from a higher numbered integrator initially cannot pick up that output but receives from the Δz store instead a value which may not be valid (see Section 8.5.2). Thus, each integrand may have an effective initial value which will differ from the value called out in the coding of the problem. However, one may be able to compensate for this effect by running the problem for several iteration periods, and noting the value of the integrands at the end of each period. An effective initial value can be derived from a smooth curve fitted to these points. This information can be used to estimate the amount by which the initial value placed in an integrand register should be biased with respect to the integrand value on the coding sheet. The starting error can also be materially affected by the assignment of numbers to each of the operational units employed. Starting errors as well as phase errors assume greater importance when the dz output of an integrator drives other integrators which in turn may influence the first integrator's dy input. In this case, particular combinations of sign conditions can force the error farther in the same direction, producing a so called biased round-off error. Starting and round-off errors in the variable integrands are often of major importance in the overall error picture.

Often, errors may be produced which are peculiar to the functions being generated. For example, auxiliary functions well behaved throughout the interval of interest can be generated with greater accuracy than func-

tions with discontinuities or excessive rates of change in one or more regions. Another type of situation that can result in a sizeable error is where there is a product of two variables, and the variables have an inverse relationship to one another with the ratio of maximum to minimum values being large. A similar situation holds for the quotient of two variables which vary in a direct relation. In each case special corrective measures may be called for.

Functions of particular importance in navigation and guidance problems are the sine and cosine. We will consider next the nature of errors peculiar to generation of the sine. With the integrator hook-up of Fig. 8.3 (b) and a rectangular integration formula, the output of one integrator is

$$(\Delta \sin \theta)_r \approx \cos \theta \, \Delta\theta$$

However, the precise expression for $\Delta \sin \theta$ is

$$\begin{aligned} \Delta \sin \theta &= \sin (\theta + \Delta\theta) - \sin \theta \\ &= \sin \theta \, (\cos \Delta\theta - 1) + \cos \theta \sin \Delta\theta \end{aligned}$$

Replacing $\cos \Delta\theta$ and $\sin \Delta\theta$ in the expression above by their series expansions

$$\begin{aligned} \Delta \sin \theta &= \sin \theta(1 - (\Delta\theta)^2/2! + (\Delta\theta)^4/4! - \ldots - 1) \\ &\quad + \cos \theta \, (\Delta\theta - (\Delta\theta)^3/3! + (\Delta\theta)^5/5! - \ldots) \\ &= \cos \theta \, \Delta\theta - \sin \theta \, (\Delta\theta)^2/2 - \cos \theta \, (\Delta\theta)^3/6 + \ldots \end{aligned}$$

Thus, to the second order the error in $(\Delta \sin \theta)_r$ is

$$e = -(1/2)(\Delta\theta)^2 \sin \theta$$

Since this error grows regardless of the sign of $\Delta\theta$ it is cumulative, appearing as an apparent rotation of the vector and a growth in its amplitude. In problems where the range of θ is not too great, the error may be tolerable. In applications like the navigational problem described in Section 8.9, the drift in $\sin \theta$ (and $\cos \theta$) over a long period can render the computation invalid. The growth in amplitude of the vector may be removed in part by use of a trapezoidal integration formula, which yields

$$\begin{aligned} (\Delta \sin \theta)_t &= [\cos(\theta + \Delta\theta) + \cos \theta]\Delta\theta/2 \\ &= \cos \theta \, \Delta\theta - \sin \theta(\Delta\theta)^2/2 - \cos \theta(\Delta\theta)^3/4 + \ldots \end{aligned}$$

The initial statement of a differential equation may be in either a derivative or differential form, though the derivative form is prevalent. Any of a number of algebraic manipulations may be made to transform equations to a form more convenient for solution and/or interpretation—e.g., replacement of the original set of variables by a new set, defined in terms of the original ones, transformation of coordinates, etc. We will consider an important difference between derivative and differential expressions of a differential equation. For example, assume we have the equation

$$y^{(n)} = f(y^{(n-1)}, y^{(n-2)}, \ldots y^{(1)}, y, x) \tag{8-59}$$

where $y^{(n)}$ is equal to the nth derivative of the function y with respect to x, and f is a particular function of the variables in the parentheses. If we take the differential of both sides of Eq. (8-59)

$$dy^{(n)} = df(y^{(n-1)}, y^{(n-2)}, \ldots, y^{(1)}, y, x)$$

Since $y^{(n)} \equiv (d^n y/dx^n)$, it follows that

$$\int y^{(n)}\, dx = y^{(n-1)}$$

If we assume several inputs whose sum equals $dy^{(n)}$ are fed into an integrand register, and the independent variable is dx, the output for an ideal integrator would be $y^{(n)}\, dx$. If there is a bias β between given and effective initial values, the output is $(y^{(n)} + \beta)dx$. Accumulation of these increments yields $y^{(n-1)} + \beta x$. Thus instead of the true $(n-1)$st derivative we obtain a quantity which drifts from this value linearly with x.

Returning to Eq. (8-59), multiplying both sides by dx yields

$$y^{(n)}\, dx = f(y^{(n-1)}, y^{(n-2)}, \ldots, y^{(1)}, y, x)dx = dy^{(n-1)}$$

If the terms whose sum equals $dy^{(n-1)}$ are fed into an integrand register, accumulating them yields $y^{(n-1)}$. This indicates that use of the lower order derivative will not introduce the drift present in the higher order case. (The preceding comparison assumes that mappings of the two forms are comparable in the errors produced in other parts of the network). The differentiation method is less favorable because of other reasons, too: (1) In practice the lower order derivative is a velocity, whereas the higher order derivative is an acceleration. In a given period the latter can traverse a greater interval of its overall range. As a result, the scaling problem tends to be more difficult, (2) The mapping is complicated in cases where nonlinear terms must be differentiated. Finally, even where

the higher order derivative is required as part of the solution, the differentiation method may be avoided by use of servo differentiation.

In general, digital servos should not be used to generate algebraic functions that can be obtained directly. (This does not apply in on-line applications where a servo is used as a null seeking device that is part of a self-correcting system.) Limiting devices, for example an absolute value generator, must be carefully adjusted. Since the operation of a servo depends on feedback of errors to its input, a loop with appreciable attenuation in the feedback path results in poor control. Also, a servo adder, at best, introduces a one cycle phase leg.

In concluding this discussion on sources of error we point out that although schematics such as shown in Fig. 8.10 can be useful, they also can be responsible for the introduction of errors, e.g. by the inadvertent connection or failure to connect a pair of lines or by a functional notation becoming associated with the wrong line. The experienced programmer can save time and effort by replacing the schematics with a set of integrator input-output equations.

The characteristic equation of an integrator ($dz = ky\ dx$) may be stated in the form

$$d[\int \text{(integrand)}] = \text{[integrand]}\ [d(\text{independent variable})]$$

where the integrand y is the sum of the dependent variable inputs dy_i. The listing of input-output equations is started by assuming that increments of the terms whose sum equals the highest order derivative have been formed. These increments are accumulated in an integrand register for the purpose of generating the next lower order derivative, thus (see page 478)

$$\begin{aligned} dx/dt &= (d^2x/dt^2)dt \\ &= [-(b/a)dx/dt - (c/a)x]dt \end{aligned}$$

Since the term $-(b/a)\ dx/dt$ appears in the differential equation, a constant multiplier is used to produce the output $-(b/a)\ dx/dt$

$$-(b/a) \cdot dx/dt = -(b/a)dx/dt$$

Another integrator is required to generate dx from dx/dt

$$dx = (dx/dt)dt$$

Another constant multiplier converts dx to $-(c/a)dx$

$$-(c/a) \cdot dx = -(c/a)dx$$

The list of equations is complete since all terms originally assumed to exist have actually been generated. Each equation specifies the type of operational unit required for its mechanization.

8.13. Checking Results of Computations

Any of various verification procedures may be employed to aid in establishing a degree of confidence in the correctness of solutions. We will consider spot checks, which are generally useful in computational work, and two types of checks suitable for use with a DDA, namely running a problem with different scale factors, and running a problem in reverse.

In one type of spot check, solutions are compared with values already known for specific values of the independent variable. In physical problems these points are so chosen that values are readily obtained from physical considerations. In function generation these points are where the values of one or more factors is readily known (e.g., where $f(x) = 0$ or 1). In a type of spot check known as a substitution check certain sets of computed values are inserted into the original equations. An indication of error is provided by the degree to which an equation does not balance. To keep a running check of the discrepancy, one can employ an accumulator in which the terms on one side of an equation are subtracted from those on the other.

If a problem is run with different scale factors and certain digit positions in the solution are invariable, increased confidence may be placed in the accuracy of these positions. If this approach is to be practical, the additional runs should not greatly increase the time for a solution. This would not be the case if the problem were run first with a small scale factor and then with successively higher ones. An alternate approach is to rerun the problem with smaller scale factors; if each solution $(S_i)_j$ obtained for points i on run j is considered as the sum of a true solution S_i and an error E_i, then on the first run

$$(S_i)_1 = S_i + (E_i)_1 \tag{8-60}$$

and on the second run

$$(S_i)_2 = S_i + (E_i)_2 \tag{8-61}$$

If the scale factors on the second run are $1/n$ times those on the first run, then we may reasonably estimate that

$$(S_i)_2 \approx S_i + n(E_i)_1 \tag{8-62}$$

From Equations (8-60) and (8-62) a general expression for estimating the error on the first run is

$$(E_i)_1 \approx (S_i)_2 - (S_i)_1 / (n-1) \tag{8-63}$$

To lessen the probability of obtaining too low an estimate for the error on the first run because of a chance agreement of $(S_i)_1$ and $(S_i)_2$ at a selected point, i, $(E_i)_1$ should be evaluated for several values of i at widely separated points. If the error is too large, the original scale factor can be increased and a new estimate obtained.

This method also allows an estimate of the true solution to be made from two inaccurate solutions, in the event that a rerun with higher scale factors is not practical because of computing time requirements and the limited length of registers. Solving Equations (8-60) and (8-62) for S_i

$$S_i \approx [n(S_i)_1 - (S_i)_2]/(n-1) \tag{8-64}$$

In principle one can, by changing the sign of the independent variable, retrace a computation. At the end of this process, the deviation from the correct initial conditions indicates over-all error in the computation. (The retracing process is precise for a linear difference equation with constant coefficients only if the coefficients of the highest and lowest order terms are $+1$ or -1.) Also, sometimes the initial state cannot be recovered; e.g., after a damped oscillation has decayed completely.

8.14. Simulating the DDA with a GP Machine

The differential analyzer approach to the solution of a problem is direct, simple, and intuitive. A way in which this approach can be used with a GP arithmetic computer will be illustrated by a specific example. Consider the familiar second order differential equation

$$y'' + y = 0, \text{ where } y'' = \frac{d^2y}{dx^2}.$$

This equation is solved on a differential analyzer by specifying that two integrators be interconnected as shown in Fig. 8.3(b). The two integrators form a closed loop system, and when the driving function is applied, the system will oscillate in a manner defined by the given differential equation, since the integrator system is an analog of a physical system described by the equation. An analog differential analyzer produces a mechanical dis-

placement when driven; a digital differential analyzer produces trains of pulses representing numerical increments.

Each of the elements of a continuous differential analyzer can be simulated in a digital computer, of either the incremental (DDA) or integral transfer (GP) type. The digital differential analyzer has a fixed program that simulates the operation of integration, and only the interconnection of "integrators" and scaling has to be derived for an individual problem. Simulation of a differential analyzer can also be performed by a GP machine. However, in this case one must write a complete program to simulate the operation of integrators, in addition to specifying interconnections and scaling.

Common analytic functions (polynomial, trigonometric, logarithmic, etc.) are usually generated in a differential analyzer by interconnecting units in a system so that the system satisfies a differential equation whose solution is known to be the desired function (as shown in Fig. 8.3). These functions can be generated in a GP machine by the use of difference equations. For example, to generate the function e^{kx} for the equidistant discrete values of the argument $x = nh$ (where h is the constant interval and n an integer), one can utilize the difference equation

$$e^{knh} = e^{kh}\, e^{k(n-1)h}$$

which states the exponential function can be generated by simply multiplying the $(n-1)$th value by a constant to obtain the nth value. Sin x or cos x can be generated by the following trigonometric identities

$$\cos nh = 2(\cos h)\cos\ (n-1)h - \cos\ (n-2)h$$

$$\sin\ nh = 2(\cos h)\sin\ (n-1)h - \sin\ (n-2)h.$$

These functions may also be generated by power series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

or Chebyshev polynomials (see Section 6.2.5).

In a differential analyzer, integration is a basic operation, whereas multiplication (other than by a constant) is not and must normally be constructed from integration operations. In an integral transfer machine, multiplication and addition are basic operations (though multiplication

is usually constructed automatically from repeated addition operations) and integration must be constructed from multiplication and additions.

The integral of a function $f(x)$ between the limits a, b can be found by dividing the interval into n equal subintervals, approximating the function by some mean value f_i in each subinterval, multiplying each f_i by the width of the subinterval and adding up the areas of all the rectangles. The computation is simpler when fewer intervals are taken, but the accuracy suffers. For a high speed digital computer it is not impractical to take large values of n. When n is large, rectangular integration gives a sufficiently good approximation, thereby permitting the use of this simple process. A real difficulty lies in the fact that as n becomes large, the number of products to be summed becomes great. Since each product is rounded off, the error grows as n grows.

The characteristic equation of an integrator (see Section 8.1) may be written

$$w = k \int u \, dv.$$

An integrator can be simulated by assigning one register to store the output w, and a second register to store the input u. Once each cycle, the change in v, Δv (corresponding to the constant change h used in the preceding discussion), is determined. The product $u\Delta v$ is then added to the contents of the register containing w. Thus, identifying the various values of each variable with a subscript to denote the iteration period in which that value occurred (i.e., v_i is the value of v at the end of the ith period, v_0 being the initial value), the integration is performed by using the formula

$$w_i = w_{i-1} + u_i(v_i - v_{i-1})$$

or by a similar formula

$$w_i = w_{i-1} + u_{i-1}(v_i - v_{i-1})$$

To illustrate the differential analyzer approach to the solution of a differential equation, we will consider the equation

$$y'' = -y.$$

Solution of this equation requires two integrations

$$y' = \int y'' \, dx$$

$$y = \int y' \, dx.$$

The computation cycle is begun by performing the integration $y' = \int y'' \, dx$, making use of the rectangular integration formula

$$y'_i = y'_{i-1} + hy''_{i-1}$$

and the original differential equation

$$y''_i = -y_i.$$

The integration $y = \int y'\,dx$ is obtained by use of the rectangular integration formula

$$y_i = y_{i-1} + hy'_i.$$

The program shown in Table 8.2 will produce the solution from $x = 0$ to $x = nh$.

To summarize, this differential analyzer approach generates functions by the use of difference equations and performs integration by summing the areas of rectangles. It permits rapid writing of very short iterative programs for the solution of differential equations, and is intuitive, simple and easy to apply. Its primary value is in the preliminary investigations of ranges of values.

Appendix: Conditions for Generating Functions of One or More Variables, and for Solving Ordinary Differential Equations by Means of a Differential Analyzer

A general theory of the differential analyzer has been formulated to describe the conditions necessary for the solution of various types of problems on an "idealized" machine. Actual machines differ from the ideal in such physical limitations as time lags in the transmission of data within the system and the finite length of numbers that can be represented. An outline of the theory will be presented here. (For the complete treatise, see Shannon [1941], [1942.])

The assumptions underlying the theory are as follows: (1) All ordinary differential equations to be considered have unique solutions. (2) Formal processes of integration, differentiation, etc., are valid in the region of interest. (3) For total differential equations, it is not necessary that the equations be integrable but it is assumed that a solution exists along any curve in the region of interest. (4) There is available an unlimited number of idealized integrators and adders. The characteristics of an integrator are: Given two input variables du and dv, an output variable is constrained to be $dw = (u + a)dv$, where a, the initial setting of the integrand, is an arbitrary constant for all variations of u and v. (In any integrator, the maximum value of $|u + a|$ is limited, but by changing scale factors it can be made as great as desired so that except for poles of u the integration can be performed.) The characteristics of an adder

TABLE 8.2. Program for solution of the equation $y'' = -y$.

Address	Instruction	B-code	Explanation
001	F k + 3	01	Places iteration index in index register 01
002	A k + 4		Clears accumulator
003	S k + 1		Places $-y_{i-1} = y''_{i-1}$ in accumulator
004	M k		Forms hy''_{i-1}
005	A k + 2		Form $y'_{i-1} + hy''_{i-1} = y'_i$
006	C k + 2		Replaces y'_{i-1} (in storage location k + 2) by y'_i
007	M k		Forms hy'_i
008	A k + 1		Forms $y_{i-1} + hy'_i = y_i$
009	C k + 1		Replaces y_{i-1} (in storage location k + 1) by y_i
010	R 002	01	Subtract 1 from index register 01 and transfer control to location 002 if remainder is $\geqslant 0$.
12	Next instruction		

k	$h = \Delta x$	Constants and intermediate results are stored here
k + 1	y_0, y_i	
k + 2	y'_0, y'_i	
k + 3	Iteration Index $(n-1)\,2^{-80}$	
k + 4	Zero	

are: Given two input variables *du* and *dv*, an output variable is constrained to be $dw = du + dv$ for all variations of u and v. (5) A system of ordinary differential equations with independent variable x and dependent variables $y_1, y_2 \ldots y_n$ can be solved if, and only if, a set of connections can be found using the above elements and satisfying the source of drive assumption, such that when there is an increment in the independent variable x, increments in the dependent variables are constrained to vary in accordance with the equations for arbitrary given initial conditions. (6) A system of

total differential equations can be solved if a set of connections can be found such that when there is an increment in a set of independent variable inputs $x_1 \ldots x_n$, increments in the dependent variables are considered to vary in accordance with the equations for arbitrary given initial conditions.

We will list next five theorems pertaining to functions of a single variable. The first is referred to as the fundamental solvability condition:

Theorem 1. A necessary and sufficient condition for a system of ordinary differential equations to be solved using only integrators and adders is that they can be written in the form

$$\frac{dy_k}{dy_1} = \sum_{i,j=0}^{n} a_{ijk}\, y_i \frac{dy_j}{dy_1} \qquad k = 2, 3 \ldots n$$

where $y_0 = 1$, y_1 is the independent variable, and $y_2 \ldots y_n$ are dependent variables, among which are the dependent variables of the original system.

A function of a single variable $y = f(x)$ can be generated if there is an interconnection such that when there is an increment in the independent variable dx there is a dependent variable that is constrained to vary by an amount dy. It follows from Theorem 1 that if $f(x)$ can be generated there must exist a set of equations (1) such that if $y_1 = x$, then $y_2 = f(x)$. A function of n variables $F(x_1 \ldots x_n)$ can be generated if there is a set of interconnections such that for independent increments in $x_1 \ldots x_n$, a dependent variable will be constrained to generate an increment dF.

Theorem 2. A function of one variable can be generated if, and only if, the function is not hypertranscendental.

Theorem 3. If a function of one variable $y = f(x)$ can be generated, then its derivative $z = f'(x)$, its integral $w = \int_a^x f(x)dx$, and its inverse $x = f^{-1}(y)$ can be generated.

Theorem 4. If two functions f and g can both be generated, then the functional product $y = f\,[g(x)]$ can be generated.

Theorem 5 relates to the approximation of functions which cannot be generated exactly.

Theorem 5. Any function $f(x)$ which is continuous in a closed interval $a \leqslant x \leqslant b$, can be generated in this interval to within any prescribed allowable error $\epsilon > 0$ using only a finite number of integrators.

The next five theorems relate to functions of more than one variable.

Theorem 6. A function of m variables $y_{m+1} = f(y_1 \ldots y_m)$ can be generated if, and only if, it satisfies a set of total differential equations of the form

$$dy_k = \sum_{i,\, j\, =\, 0}^{n} a_{ijk}\, y_i\, dy_j \qquad k = (m+1), (m+2) \ldots$$

where $y_0 = 1$ and the a's are real constants.

Theorem 7. If two functions of several variables, $f(x_1 \ldots x_n)$ and $g(y_1 \ldots y_m)$ can both be generated, then it is possible to generate any functional product, for example $\phi(x_2, x_3 \ldots x_n, y_1, y_2 \ldots y_m) = f(g, x_2, \ldots x_n)$.

Theorem 8. Given any function of n variables $f(x_1 \ldots x_n)$, continuous in all variables in a closed region of n − space $a_k \leqslant x_k \leqslant b_k$, $k = 1, 2 \ldots n$, a function $F(x_1 \ldots x_n)$ can be generated using only a finite number of integrators and adders such that within the region $a_k \leqslant x_k \leqslant b_k$, $|f - F| < \epsilon$ where ϵ is an arbitrarily small prescribed positive number.

Theorem 9. If a function of n variables $f(x_1 \ldots x_n)$ can be generated, its partial derivative with respect to any one variable, say x_1, can be generated.

Theorem 10. If a function of n variables $y = f(x_1 \ldots x_n)$ can be generated, its inverse with respect to any one variable $x_1 = F(y, x_2 \ldots x_n)$ can be generated.

Finally, we list a general theorem which relates to systems of equations.

Theorem 11. The most general system of ordinary differential equations:

$$f_k(x;\, y_1, y'_1 \ldots y_1{}^m;\, y_2, y'_2 \ldots y_2{}^m \ldots y_n,\, y'_n \ldots y_n{}^m) = 0$$

where $k = 1, 2, \ldots n$, and which is of the mth order in n dependent variables can be solved on a differential analyzer using only a finite number of integrators and adders providing the functions f_k are combinations of non-hypertranscendental functions of the variables.

LITERATURE

Adams, C. W. [1950] The differential analyzer approach in digital computers, M.I.T., Project Whirlwind, Memorandum M-1036.

Amble, O. [1946] On a principle of connection for Bush integrators, *J. Sci. Instr.*, **23**, 284-287.

Bradley, R. E. and Genna, J. F. [1962] Design of a one-megacycle iteration rate DDA, *Proc. AFIPS Spring Joint Computer Conference*, 353-364.

Braun, E. L. [1954] Design features of current digital differential analyzers, *IRE National Convention Record*, **2**, Part 4, 87-97.

Braun, E. L. [1957] Digital computers in continuous control systems, *IRE National Convention Record*, **5**, Part 4, 127-135; *IRE Trans. El. Comp.*, **7**, (June 1958).

Braun, E. L. and Post, G. [1958] Systems considerations for computers in process control, *IRE National Convention Record*, **6**, Part 4, 168-181.

Braun, E. L. [1960] A comparison of integral and incremental digital computers for process control applications, *Control Engineering*, **7**, 113-118.

Bush, V. [1931] The differential analyzer, a new machine for solving differential equations, *J. Franklin Inst.*, **212,** 447-488.

Bush, V. and Caldwell, S. H. [1945] A new type of differential analyzer, *J. Franklin Inst.*, **240,** 225-326.

Crank, J. [1948] *The differential analyzer,* Longmans, Green, London.

Deering, C. S. and Shelman, C. B. [1961] An incremental computer technique for solving coordinate-rotation equations, *IRE Trans. El. Comp.*, **10,** 748-751.

Donan, J. F. [1952] The serial memory D.D.A., *MTAC* **6,** No. 38, 102-112.

Gill, A. [1959] Systematic scaling for digital differential analyzers, *IRE Trans. El. Comp.*, **8,** 486-489.

Hartree, D. R. [1946] [1948] The application of the differential analyzer to the evaluation of solutions of partial differential equations, *Proc. 1st Canadian Math. Congress,* Montreal, 1945. Univ. of Toronto Press, Toronto, pp. 327-337; *MTAC,* **2,** 56.

Hartree, D. R. [1940] The Bush differential analyzer and its applications, *Nature,* **146,** 319-323.

Henegar, H. B. [1961] New continuous path system uses DDA interpolator, *Control Engrg.*, **8,** No. 1, 71-76.

Maginniss, F. J. [1945] Differential analyzer applications, *Gen. Elec. Rev.*, **48,** 54-59.

Mendelson, M. J. [1954] The decimal digital differential analyzer, *Aeronaut. Engrg. Rev.*, **13,** 42-54.

Merz, D. M. [1959] GEVIC—A real time variable increment digital computer, *Automatic Control,* September 1959, 18 DC-26 DC.

Michel, J. G. L. [1948] Extensions in differential analyzer technique, *J. Sci. Instr.*, **25,** 357-361.

Mitchell, J. M. and Ruhman, S. [1958] The TRICE-A high speed incremental computer, *IRE National Convention Record,* **6,** Part 4, 206-216.

Nelson, D. J. [1962] DDA error analysis using sampled data techniques, *Proc. AFIPS Spring Joint Computer Conference,* 365-375.

Owen, P. L., Partridge, M. F. and Sizer, T.R.H. [1960] CORSAIR, A digital differential analyzer, *Electronic Engrg.*, **32,** 740-745.

Palevsky, M. [1953] The design of the Bendix digital differential analyzer, *Proc. IRE,* **41,** 1352-1356.

Reed, I. S. [1951] Some mathematical remarks on the Boolean machine, M.I.T., Project Lincoln, Technical Report No. 2.

Rowley, G. C. [1958] Digital differential analyzers, *Brit. Commun. & Electronics,* **5,** 934-939.

Shannon, C. E. [1941] Mathematical theory of the differential analyzer, *J. Math. and Phys.*, **20,** 337-354.

Shannon, C. E. [1942] Theory and design of linear differential equation machines, PBL 58240 (4-951) OSRD 411, January, 1942.

Shemeta, E. A. [1960] Accelerated programming for GEVIC in real time applications, *Automatic Control,* **12,** 26-44.

Sprague, R. E. [1952] Fundamental concepts of the digital differential analyzer method of computation, *MTAC,* **6,** No. 37, 41-49.

Thomson, Sir William (Lord Kelvin) [1876] Mechanical integration of the linear differential equations of the second order with variable coefficients, *Proc. Roy. Soc.*, **24,** 269.

Tierney, J. W., Homan, C. J., Nemanic, D. J., Amundson, N. R. [1957] The digital computer as a process controller, *Control Engineering,* **4,** 166-175.

Weiss, E. [1952] Applications of the CRC-105 decimal digital differential analyzer. *IRE Trans. El. Comp.*, **1,** 19-24.

9. The Detection and Correction of Errors

9.1. Introduction

This chapter deals with topics related to the validity of results produced by a digital computer. This includes techniques for anticipating, detecting, and locating the source of equipment failures; special codes for minimizing the effect of errors, and for detecting and/or self-correcting errors; mathematical techniques for checking results and minimizing errors due to the computation process itself. Techniques for improving the reliability of electronic circuits will not be discussed here. For a theoretical discussion of reliability in electronic circuits, the reader is referred to the references listed in the bibliography of this chapter.

The validity of results produced by a digital computer depends on many things including the adequacy of the mathematical formulation and numerical approximation procedure, the absence of mistakes in the programming and coding of the problem and its entry into the computer, and the correct operation of the computer during the time interval required for solution of the problem. Let us consider briefly the general question of reliability of the computer itself. A commonly used definition of reliability is as follows: the probability (expressed in percentage) that a system will perform its function without error for a specified length of time while in a specified environment. For example, if, whenever a machine is used, it operates without failure for the specified duration and in the specified environment, it is considered 100% reliable.

A useful device for indicating a machine's reliability is a histogram such as the one shown in Fig. 9.1, in which are charted the frequencies of

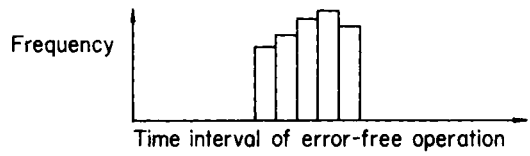


FIG. 9.1. History of error-free operating intervals

different error-free periods of operation, accumulated from an operating log which lists how long the computer functions properly each time it is

turned on. From this data one can also estimate the mean time to failure. Of course, the degree of confidence that can be placed in the reliability figure increases, theoretically, with the number of samples. In practice, this measure will also be influenced by the effects of aging (good or bad) and other factors such as improved maintenance techniques, replacement of parts, etc. A high mean time to failure figure is of importance in a laboratory computer, since it implies the computer will be available for useful work a large percentage of the time. For other applications, e.g., a computer used in a high-speed real-time control system, such as in a supersonic aircraft, a value for the minimum error free period of operation greater than the period of a mission is of more importance. Finally, it should be emphasized that any machine must be designed for operation in a particular environment and that a reliability figure for a particular machine must be based on operating experience in the environment specified.

Once a fault or error has occurred somewhere in the computer, it is, of course, desirable to detect the source of error in a minimal time. The fault-correcting time can be reduced by the use of techniques and devices designed to aid in disclosing the location of faults. (Regardless of these devices, the fault-correcting time will also be a function of the skill and ability of the operation and maintenance personnel.)

Before describing the checks most commonly used for detecting malfunctions of the computing equipment and tracing them to their source, let us consider the ways in which such malfunctions may be brought about, i.e., the ways in which components of the machine may fail. Some basic types of failures are shown schematically in Fig. 9.2. This is a

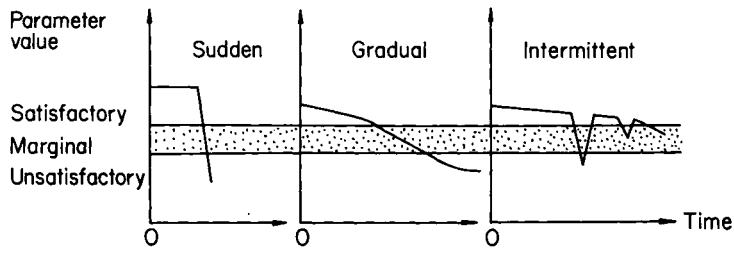


FIG. 9.2. Different types of failure

qualitative picture. The time scale will vary with the physical nature of the component and in fact will vary even with components of the same type. Unsatisfactory performance resulting from gradual deterioration or aging of a component is more common than a sudden complete failure.

The latter type of failure may occur when a component which has not been adequately underrated is subjected to a greater than normal load, i.e., when a component has been rated with respect to average rather than peak requirements. Intermittent failures may occur any time after the performance characteristics of a component have reached the marginal zone—whether that zone was entered suddenly or gradually.

The different types of failures described imply different degrees of difficulty in detection of faulty operation and location of the defective component. An error caused by a complete failure of a component is relatively easy to find.

9.2. Detecting and Locating Sources of Error

There are a number of possible sources of error in the results produced by a digital computer. These errors may be introduced by the particular mathematical formulation of the problem being solved, by the finite numerical processes employed, by mistakes in programming, and by malfunctioning of the computing equipment. We will consider in the succeeding sections the different means that may be employed to detect and locate the source of these failures.

When an error is caused by a complete failure of a component it is relatively easy to trace the source. Test programs (described in Section 9.2.3), which require the functioning of all components in the computer or in a suspected part of it are useful in detecting this source of error.

Tracing the source of an intermittent error is more difficult. To aid in such a trace it is desirable to provide some means for stopping the computer on the very step where the error occurs. This is because study of the contents of the various registers at this time sometimes permits the source of the error to be deduced. An advantage of error detection circuits, compared to programmed checks, is that they detect an error immediately upon occurrence. Test programs are helpful here, too, although not when the marginal component fails only rarely (see Section 9.2.3 for a discussion of how the frequency of failure may be increased by a marginal checking procedure). Actually, the best way to trace marginal components is by a preventive maintenance procedure.

Some intermittent errors cannot be traced by any simple procedure. Instead success depends on the ingenuity and experience of the troubleshooter. Examples of elusive sources of error are: (1) Defective connections in wiring, e.g., cold solder joints. (2) Components operating marginally, but to which marginal checking procedures cannot be applied because of their location in a circuit. (3) Places where the failure may be self healing for a relatively long time before partial failure occurs again.

Another significant source of errors arises from mistakes in wiring introduced either when the computer was built or modified, or during a trouble-shooting operation.

Checking to assure that the results of computations are correct has long been considered important. For a high-speed computer where enormous amounts of computation are performed without interruption there is an even greater need for checks. This follows because in large computing systems numerous components are subject to failure, and even a single failure may often completely vitiate the solution. It is desirable that a computer check itself during the course of a computation for the following principal reasons: (1) There are too many operations involved to permit a check of this type by a human operator. (2) Human checking would not be in keeping with the initial purpose of the computer—i.e., to relieve humans of routine computation. (3) The check should proceed in step with the computation in order that errors may be detected as they occur, thus preventing the loss of correct results obtained before the occurrence of an error. Special considerations in the checking of a computer functioning as part of a control system are discussed in Section 9.2.2.

There are a number of automatic checking methods available. They fall into two main categories, namely built-in checks, and programmed checks. These will be discussed in Sections 9.2.1 and 9.2.2, followed by a description of programs for testing and diagnosis in Section 9.2.3, and preventive maintenance in 9.2.4. It should be emphasized here that there is no substitute for basically reliable circuitry, since most practical error detection and correction schemes are effective only against single transient malfunctions resulting not from a faulty component but from a random disturbance.

9.2.1. Built-in Checks

9.2.1.1. Information Storage and Transfer Checks

This type of check is used to determine whether an error has been introduced in the process of writing information into or reading it from the store, or in the transfer of information between various sections of the computer. Any of various techniques may be employed for such checks. For example, there is incorporated in the MIT Whirlwind computer a special checking register which is used whenever the execution of an instruction requires the transfer of information from one set of registers to another. This register receives information from the source register and, also, via a different path, from the receiving register. Any discrepancy indicates an error has occurred. Another scheme consists of

simply storing each quantity in duplicate and comparing corresponding positions of each pair. Note that both these schemes depend on the introduction of some type of redundancy—in the first case a redundant operation, in the second redundant storage as well. Neither of them furnishes the information required to correct an error, although the latter scheme, which can detect a single error in any or all pairs of data words, can be extended one step further to the extreme of storing each quantity in triplicate and deciding on the correct value of any bit position on a majority principle.

We will now proceed to describe some schemes for error detection and correction based upon the use of special codes and coding techniques. When one or more bit positions in a coded group is changed as the result of a malfunction in the computer, a new value may be produced which is also meaningful, i.e., one of a set of admissible values. If this is the case, inspection of the new value is not sufficient to establish that an error has occurred. This implies that the capability for error detection depends on there being a greater number of possible values than admissible values. For example, in a four-bit binary-coded decimal group, the decimal values 0 through 9 are admissible, but the values 10 through 15 are not. Therefore, the occurrence of any of the values 10 through 15 is an indication that an error has occurred. However, with this system there is not complete assurance that an error, even in a single bit position, can be detected. This is because an error in an admissible value may produce another admissible value, e.g., the accidental changing of 6 to 7 (110 to 111). For assurance that an error in a single bit position can always be detected, the defined set of admissible values must be such that a change of an admissible value in any bit position produces a nonadmissible value. A code capable of detecting an error in any bit position of a decimal representation requires at least five bits. One such code represents the digits by the ten representations of five bits in which two of the bits have the value 1. Error detection is accomplished by inspection of the number of 1's present. A major drawback of this code is that it is not well suited for arithmetic manipulations.

A way of introducing nonadmissible values for error detection which does not preclude the use of the straight binary code or any other desired code, is by the inclusion of a parity bit with the group of bits to be checked. The value assigned to this bit is such that in a so-called even parity checking system the total number of 1's in the data and parity bit is even and in the odd parity checking system it is odd. The net effect is that $n + 1$ bits are used to represent 2^n admissible values and 2^n nonadmissible values. The parity bit is stored and transmitted with the group

of bits being checked. A subsequent discrepancy between the oddness (or evenness) of 1's in the number itself and the value of the parity bit indicates that one error or an odd number of errors has occurred.

The type of parity check just described may be considered as a single row or column check. In another type of parity check, referred to as an array check, several rows and columns of data are checked as a set, a parity bit being assigned to each row and column. The location of a single error is indicated by the intersection of the row and column corresponding to the row and column parity bits for which there is a discrepancy. In the array check, shown in Table 9.1, p stands for the parity bit, chosen to be 1 whenever the number of 1's in the collection being checked is even.

TABLE 9.1. Parity checking for an array of numbers

					p_r
	0	0	1	0	0
	0	0	1	1	1
	0	1	0	1	1
	0	1	0	0	0
	0	1	0	0	0
p_c	1	0	1	1	

If only one check fails, and it is assumed that only one error has occurred, then it may be assumed that the error is in the parity bit itself. By the addition of another parity bit, p_{cr}, to check the oddness or evenness of the number of 1's in the column parity bits, double errors may be detected. Also, the following characteristics will be exhibited for single failures in any of the parity check bits: (1) An error in a p_r bit causes the check in only that row to fail. (2) An error in p_{cr} causes a check failure in the bottom row only. (3) An error in a p_c bit causes the new check bit p_{cr} to fail as well as the corresponding column check to fail. If a response is obtained other than the three just listed or that produced by a single failure in one of the numbers checked, two or more errors must have occurred.

For a parity check to be capable of multiple error detection or the detection and correction of a single error, more than one parity bit is required. In other words, each admissible value must differ from every other in more than two positions. For example, the detection and correction of a single error in a group of bits requires that each admissible value differs in at least three bit positions from every other admissible value.

This is apparent if we consider the case where two admissible values differ in only two bit positions, e.g., 100 and 111. If the third bit of the first number is erroneously changed to 1 and the second bit of the second number is changed to 0, the same value 101 is produced. It is obviously impossible to correct such an error by inspection of the number 101 since it could have resulted from a single error in either 100 or 111.

Correction of a single error in addition to detection of a double error requires that each admissible value differs in at least four bit positions from every other admissible value. It is generally true that error detecting and correcting codes have the property of being able to trade correcting for detecting ability. For example, the double error detecting and single error correcting code can be used instead as a triple error detecting, non-correcting code.

The single-error correcting scheme that will be described next is essentially a special form of parity checking. It consists of producing a group of check bits in such a way that the value of this group of check bits, considered as a single check number, indicates which bit position, if any, is in error. Several parity bits are associated with the data bits, the number of parity bits being determined by the number of bits in the data. If m is the number of data bits, then the number of parity bits, k, must be such that the number of possible values of the k bits (i.e., 2^k) is adequate to indicate any of the $m + k$ bit positions in which an error can occur and, also, the occurrence of no error. In other words, $2^k \geqslant m + k + 1$. The set of bits which each parity bit checks is so chosen that a different value of the k check bits occurs for an error in any of the $m + k$ bits and for the case of no error. It is even possible to so select the group of bits checked by each parity bit that if an error occurs in any bit position, the value of the k check bits indicates directly the number of that position.

Assume that the m data bits are numbered from 1 through m and the k check bits are numbered from $m + 1$ through $m + k$. Then for the case of $m = 11$, $k = 4$, for example, the check bits F_i could be defined as the sum (modulo 2) of the value of the indicated bit positions

F_1 :	1	2	3	4	6	7	8	12
F_2 :	1	2	3	5	6	9	10	13
F_3 :	1	2	4	5	7	9	11	14
F_4 :	1	3	4	5	8	10	11	15

The actual checking of the $m + k$ bit positions takes place as follows. Since the value of the parity bit plus the bits it checks is defined to be always even (or odd), a check is made of the sum of the parity bits and the digits it checks to see whether this condition is satisfied. If so, the

corresponding check bit is assigned the value 0, otherwise 1. The bit positions entering into each parity check are so chosen that when a single error occurs, it will show up in one or more bits of the checking function. A one-to-one correspondence is thus established between the sources of an error (in either a data or parity bit) and the values of the checking function. Table 9.2 shows the value of the checking function for the cases of an error in any of positions 1 through 15 and for the case of no error. Note that a single error among the F_i indicates a parity bit in error, and that either two, three, or four failures among the F_i indicates a data bit in error.

TABLE 9.2. Values of a checking function $F_4F_3F_2F_1$ corresponding to a single error or no error in the 15 positions checked

Position of Error	F_4	F_3	F_2	F_1
None	0	0	0	0
1	1	1	1	1
2	0	1	1	1
3	1	0	1	1
4	1	1	0	1
5	1	1	1	0
6	0	0	1	1
7	0	1	0	1
8	1	0	0	1
9	0	1	1	0
10	1	0	1	0
11	1	1	0	0
12	0	0	0	1
13	0	0	1	0
14	0	1	0	0
15	1	0	0	0

Simply by rearranging the positions in which the data and parity bits are placed, it is possible to derive a checking function that automatically produces the number of the position in error, rather than an arbitrary number (which must be referenced to a particular position) as in the preceding example. This occurs if the parity bits are placed in positions 1, 2, 4, . . . 2^k, and each F_i is defined as the sum (modulo 2) of the values of the indicated bit positions

F_1 :	1	3	5	7	9	11	13	15
F_2 :	2	3	6	7	10	11	14	15
F_3 :	4	5	6	7	12	13	14	15
F_4 :	8	9	10	11	12	13	14	15

For example, if there is an error in position 13, $F_4F_3F_2F_1 = 1101$; if an error occurs in position 7, $F_4F_3F_2F_1 = 0111$; if there is no error, $F_4F_3F_2F_1 = 0000$.

Table 9.3 shows the number of data bits ($m \leqslant 2^k - k - 1$) that can be accommodated by a given number of check bits, and also the number of bits checked by each parity bit, p_i.

TABLE 9.3

Data bits, m	Check bits, k	Bits checked per p_i
1	2	1
2–4	3	3
5–11	4	7
12–26	5	15
27–57	6	31
58–120	7	63

We will now consider some hardware requirements of the parity checking schemes that have been described. Those for operating on what may be considered a single row or column of data can be mechanized for systems in which the bits appear either serially or in parallel. In a parallel system each parity bit can be formed by means of a combinational circuit whose inputs are the values of the bit positions being checked by the parity bit. In a serial system, the values of the bit positions being checked are entered sequentially into the input of a trigger (single-input) flip-flop. Either arrangement can indicate whether the number of 1's entered is even or odd. In the row and column checking scheme, if the bits in each row appear in parallel, and the bits in each column serially, it is necessary to store some indication of the row in which a parity check failed, so that when the column parity bit check fails the proper word can be referenced and the value of the bit position in error complemented.

In the error correction scheme utilizing a check number, each parity bit can be generated by the means already described: specifically, by a single input flip-flop if the data is in serial form, by a combinational circuit (with the number of terms indicated in column 3 of Table 9.3) if the

data is in parallel form. For each check bit, F_i, a single input flip-flop may be used for data in serial form and a combinational circuit (with one more term than for p_i) if the data is in parallel form. These circuits allow the detection of an error and its location. Automatic correction of the error (which is accomplished by complementing the value of the bit in the position designated by the checking number) requires the use of some temporary storage and delay elements. In the example described, if the bits appear serially as the successive states of a flip-flop, Q^1, then the correction can be effected by means of a delay line (or shift register) and ten flip-flops as follows. Data read from Q^1 is used to set each of four flip-flops F^4, F^3, F^2, F^1 in accordance with the rules for forming the check bits F_4, F_3, F_2, F_1. At the end of the word period, the check number is transferred from F^4, F^3, F^2, F^1 to four other flip-flops F^{14}, F^{13}, F^{12}, F^{11}. Q^1 also drives a delay line whose length is such that as one word is being read from Q^1 the bits of the preceding word are read from a flip-flop Q^2 driven from the output of the delay line. As each bit is read from Q^2, the contents of F^{14}, F^{13}, F^{12}, F^{11} are diminished by 1, so when they hold the value 0001, the bit read from Q^2 is the one that is to be corrected. The output of Q^2 is shifted into a flip-flop, Q^3, except when the contents of F^{14}, F^{13}, F^{12}, F^{11} equal 0001, at which time the complement of Q^2 is shifted to Q^3. Thus, bits of a word with one error appearing at Q^1, will appear in correct form at the output of Q^3 one word-time later. It should be stated at this point that because of the added cost they introduce, error detection and correction circuits are used sparingly.

The error detection and correction schemes described are all based on defining a set of admissible values and a corresponding set of inadmissible values. It follows that more bit positions are used than would normally be required to represent the data. A measure of the redundancy is the ratio of the additional bits used to the minimum number required to represent the data. For example, in the case where $m = 26$, $k = 5$, the redundancy is 0.19. In order to conserve storage elements, it is desirable to have an error detection and correction scheme that does not introduce too much redundancy. For a description of various error detection and correction schemes, see the papers listed in the bibliography.

9.2.1.2. Arithmetic Checks

In normal operation, a scale factor assigned by the programmer is associated with each number in the machine, and remains constant during the course of a problem. This scale factor is chosen so that every number used or generated during the course of a problem can be represented within the finite register length of the computer. Often it is quite difficult

to estimate accurate bounds on some of the partial results of a complicated problem. Therefore, one of the most important and commonly used built-in checking features is one that indicates whether a number has been produced that exceeds register capacity. This is usually accomplished by a circuit that detects an overflow of the accumulator (in either a positive or negative sense). This also sets the computer to an idle state, thereby allowing corrections to be made before the effects of the overflow can be propagated. In a control computer provided with automatic error correction routines, the overflow would not stop the machine but instead cause transfer of control to the correction routine.

Machines having a built-in divide instruction should have a built-in check to test whether the quotient will be less than 1. The check consists of ascertaining if the divisor is less than the dividend, in which event the quotient register would overflow. Checking circuits for adders and other arithmetic devices are not difficult if appropriate codes are chosen. However, it is difficult to devise practical checking circuits for all of a computer, the control circuits in particular being difficult to check thoroughly.

Another type of check that can be incorporated into the arithmetic circuits is based on a procedure often used to check manually-performed arithmetic. When used with the decimal system, this check is referred to as checking by casting out 9's. In this check, the result of each of the four basic arithmetic operations is checked by the use of the residue (modulo nine) of the operands. In the addition check, the residue of each addend is obtained* and the residue of the sum of these residues is compared with the residue of the sum of the addends. If the residue of the sum agrees with the residue of the sum of the addend residues, the sum is assumed to be correct. For example

	Addends		Residue of addends
	15941		2
	46897		7
Sum	62838	Sum of addend residues	9
Residue	0	Residue	0

The multiplication check is based on the fact that the residue of the product of two numbers should be equal to the residue of the product of the factor residues. For example

* It is not necessary to divide a number by the modulus to obtain its residue, since the residue of the sum of the digits in a number is equal to the residue of the number itself.

	Factors		Residue of factors
	12		3
	7		7
	—		—
Product	84	Product of factor residues	21
Residue	3		3

The division check consists of comparing the residue of the remainder with the difference formed by subtracting the product of the residues of divisor and quotient from the residue of the dividend. For example, if the dividend is 23 and the divisor 12, then

$$R_{\text{remainder}} = R_{\text{dividend}} - (R_{\text{divisor}} \times R_{\text{quotient}})$$
$$2 = 5 - (3 \times 1)$$

where R is the residue.

An analagous system of checking may be applied to binary numbers by considering such numbers as octal numbers (merely by considering groups of three bits each) and using a casting out 7's system. An obvious deficiency of any residue checking procedure is that it will not detect an error whose magnitude is an integral multiple of the number being cast out.

Circuits for residue checks would perform the following operations: (1) production of the residues from the operands, (2) operations of an arithmetic nature on these residues, (3) generation of the residues of these results, and (4) comparison of the residue of the results produced by operating on the operands with the residue of the result produced by operating on the residues.

Another built-in checking device, intended for use in a control computer, does not actually check an arithmetic operation but is designed to prevent a program from getting out of sequence. It consists of generating periodically a timing pulse which causes control to be transferred to a specified point in the program, the program being divided into sections each of which can be executed in less time than the period of the timing pulses. These timing pulses are also useful in applications where the computation must be synchronized with real time, facilitating the use of predictive and extrapolative formulas.

The use of built-in checking equipment adds considerably to the cost and complexity of a computing system. For this reason, and because more reliable components are becoming available, the use of built-in checks (including duplication of circuits) if used at all, is usually confined to a particular crucial part of the system. The emphasis is now being placed on improving the reliability of individual circuits, and the use of programming to detect errors due to malfunctions. These programs may be either ones that are written specifically for diagnostic purposes, or programmed

mathematical checks incorporated in a main program. For example, the residue checks described could be incorporated as part of a running program (at the expense of problem running time). In conclusion, it should be stated that each of the built-in or programmed error detection and correction procedures described has its deficiencies and that much work remains to be done in this area.

9.2.2. Programmed Error Detection and Correction

The over-all reliability of operation can be improved by supplementing reliable circuitry, including check circuits, with carefully designed error detection and correction programs. These programs not only can be employed where the extra cost of checking circuitry is prohibitive but also provide a greater variety of checks. The basic procedures are few: recomputation by the same, an inverse, or different process and comparison of the results, tests to see whether the results satisfy certain mathematical or physical criteria in the solution of problems involving physical systems, a check based on estimates of behavior of certain variables, and various special checks that may be possible with a particular process or machine.

In the various schemes for detection of errors by programmed checks, there is a basic difference between those designed for use with laboratory computers and those designed for use with computers that comprise part of a control system. In the former case, the programmed checks used in conjunction with various computational routines serve simply to detect an error, whereupon the computer is stopped. At this point, and at the discretion of the user, the problem may be rerun either completely or from the last point in the computation where the computer was known to be operating correctly, or recourse made to some fault locating technique like a diagnostic program (Section 9.2.3). In many applications a control computer cannot be stopped, and there is insufficient time available to repeat more than a small part of the computation upon detection of a malfunction. Thus, it would be desirable for the detection process, which can consume only a small percentage of computing time, to automatically actuate a process that corrects the error in a very short period.

The selection of programmed error detection and correction means to be employed must be based on several factors, including the probability of each type of malfunction and its detection and correction by a particular technique, the probable damage produced by different malfunctions, and the cost of additional storage and increased computing speed requirements. For utilization of many of the analytic checks, the programmer must establish tolerances on allowable discrepancies, and the degree of

confidence to be placed in a particular type of check. He must also decide upon the frequency with which various checks are to be applied, striking a compromise between machine running time consumed in checking operations and the relative ease with which an error may be traced to its source.

A point worth mentioning here is that even though many numerical approximation procedures tend to erase small errors which may accidentally be introduced, the nature of the convergence process requires careful analysis, for large errors may lead to convergence on another branch, producing a result which may or may not seem plausible.

9.2.2.1. Analytic Checks

One of the more obvious ways to check the result of a computation is by recomputation, the check being based on the assumption that if the two answers agree, the result is correct. However, this method has certain limitations. First of all, any error made in programming or coding will be common to both computations and therefore will not be detected. Also, this type of check is effective only against transient failures rather than steady state ones. If the machine fails systematically, there is an appreciable probability of the same error being made on the second run as on the first. This probability of identical systematic errors may be almost eliminated by running the problem on two computers. If only one computer is available, the time between successive runs should be as long as practicable, assuming that systematic failures of large computers will not persist over periods of one to two days. For increased confidence in the results of a recomputation check, the second computation should be made using a different mathematical method or program than that used in the first computation. Because of its limitations, this type of check should only be used as a preliminary one, in conjunction with other checks.

In a control application there may be certain computed quantities that are critical in the sense that an error in them would have a damaging effect on the system. Therefore, even though multiple computation (and storage of critical parameters) and inference of the correct answer on a majority basis may not be feasible for a whole computation, it may be warranted in the computation of these critical quantities. Such a routine will also detect and correct certain multiple errors in one quantity (since the value of each bit is inferred on a majority basis).

A type of check easily made where applicable is the use of known relationships between functions. For example, one can compute the value of a trigonometric function from some type of series expansion of the argument and then check this value by computing it again in terms of another function. Specifically, one could compute $\sin x$ from the relation

$\sin x \approx x - x^3/_3! + x^5/_5! - \ldots$ and then check it by means of the relationship $\sin^2 x = 1/(1 + \text{ctn}^2 x)$ after computing the value of ctn x. Another scheme is to use the same equation but to arrive at a particular value from two directions. For example, in the integration of a differential equation, one uses the given initial conditions and computes successive values from that point. Then a set of computed values is used to define new initial conditions and the process is reversed, values obtained in this way being compared with those obtained earlier. Also, the nature of a particular computation may often allow a useful functional check peculiar to it to be employed. A major limitation of functional checking as a general procedure is that no functional relationships are known for many of the functions which arise during the solution of a problem.

A test frequently employed, and which is a special type of functional relationship test, makes use of inverse operations. For example, if the addition of two quantities is called for, after the sum has been formed one of the quantities is subtracted from the sum and this difference compared with the value of the corresponding addend. This procedure will detect errors due to either transient or steady state and intermittent malfunctions of the machine (Fig. 9.1), but the lengthier program means increased computing time and storage requirements, and more time for preparation. A variation of this procedure, less costly in respect to the parameters mentioned, is one wherein the inverse operation of a group of operations rather than of an individual operation is performed. For example, after solution of a set of linear algebraic equations, or differential equations, the answers would be checked by substitution back into the original equations. An advantage of the group check is that it checks not only the arithmetic unit, but also various information transfers, in addition to parts of the program. However, an important disadvantage is that the source of error is more difficult to locate because of the variable number of steps that may take place after the malfunction. Therefore, careful consideration must be given to the number of operations to be covered by a single check. In using inverse checks for either individual or groups of operations, a tolerance should be provided on the differences that may occur between a direct and inverse operation not due to any machine malfunction but because of truncation and round-off errors normally introduced (see Section 9.4).

A differencing test is one used to determine whether the computed function is smooth in the sense that it has no discontinuous derivatives of low order. The test consists of determining the values of higher order differences of the function. Since the number of values available for inspection decreases by one with each higher order of difference, the

number of computed values of the function limits the order of difference that may be taken. Also, since round-off and truncation errors become more significant as the order of difference increases, a practical limit is set to the order of difference that can be considered significant. Though a difference check is sometimes useful in conjunction with the computation of the values of a function at equidistant intervals of its argument, it also has serious limitations. First of all, the check is not valid for those functions for which the differences do not decrease as their order increases, i.e., where the differences are inherently too large or too variable. It is not at all effective in detecting systematic errors that influence all computed values equally. For this reason an accompanying spot check of certain computed values is desirable. There is a possibility that even though all differences of a certain order are small an error has occurred. The values computed may even be completely incorrect, in that they represent a wrong function. Also, one cannot be sure that because the value of a certain order difference is greater than a certain magnitude that an error has occurred. Finally, smoothness checks cannot be relied upon for functions of more than one variable. Wherever the results of the test are questionable, the usual procedure is to assume an error has occurred and to attempt to verify this by a different type of check.

A similar reasonableness type of check which may be useful in the detection and correction of errors in real time control systems is to compare the value of a quantity computed directly from physical data with the value found by extrapolating from previous values, and if the computed value falls outside these bounds to use the extrapolated value. Using the known error bounds and physical bounds in the system, gross errors may be detected. Either direct Lagrangian extrapolation (if physical quantities are varying rapidly), or smoothed extrapolation (if physical and computational noise is the predominant type of error expected) may be used. For explicit descriptions of this procedure, including a description of how the extrapolated value itself may be checked by extrapolating a second time using a different formula and comparing results, and the effect on the extrapolation at the $n+1$st step if the extrapolated rather than the computed value at the nth step is used, see Ralston [1957].

The proper choice of scaling (Section 6.7.1) is important not only to the accuracy of the over-all computation, but also can be used to limit the magnitude of error produced by a malfunction. For example, assume that with one value of scaling, a variable, y, varies over the range 0.000001 (1/64) to 0.000100 (1/16). An error occurring in the most significant bit position could produce an error of 8 y_{max}. If the variable is scaled differently, so that it varies over the range 0.001000 to 0.100000

say, an error in a single bit position could not produce an error greater in magnitude than one half the maximum value.

9.2.2.2. *Sequencing Checks*

We will consider now a check devised to aid in preventing the accidental transfer of control to a storage location other than intended as a result of an error in computing the address. The method is based upon setting aside n storage locations, where n is the smallest power of 2 that exceeds $2x$, and x is the maximum number of locations to which control might be transferred from a given point in the program. There is the further restriction that the first address in the sequence be a multiple of $2n$ so that the higher order bits need not enter the computation. The x addresses are chosen from the set of n addresses so that they satisfy an even (or odd) parity check. A single error in computing any of the x entry addresses will produce one of the $n/2$ incorrect addresses. In each of these, the same instruction is stored, one which causes a transfer of control to a correction routine which recomputes the entry address.

A simple method for detecting erroneous entry to a table of constants is as follows: First of all the entries are separated into two groups, each entry in a group being of the same sign. Again a consecutive set of addresses is selected and its members assigned as the addresses of one group or the other according to whether they satisfy an odd or even parity check. The sign bit of an entry extracted from the table is tested and if found to be incorrect, a transfer of control is made to a correction routine which causes the address of the table entry to be recomputed.

To insure that a computational block is entered at the beginning, one set of instructions can be added at the beginning and another at the end of the block to check on whether those at the beginning were performed. If not, there is an automatic transfer of control to a correction routine. For example, a simple procedure is to place at the beginning of the block instructions that cause the contents of a specified storage location to be copied into another location, and at the end of the block to place instructions that transfer the contents of both locations into the accumulator, subtracting one from the other. Control is transferred to a correction routine if the difference is not zero.

9.2.2.3. *Data Transfer Checks*

Simple parity checks can also be programmed. Indication of failure could be used either to stop the computer or, in a critical control application, to transfer control to the entry of a correction routine, selected

from several in accordance with the contents of the program counter (indicating in what part of the program a failure occurred). For example, if the check fails on data read from the store, correction could be attempted by reading again, with a limit being set on the number of times this is done since continued failure indicates other than a transient error.

Another type of check, often used as a standard part of a machine's operation, is the memory sum check used to check a block of instructions and data entered into a machine's central store from an input storage medium. The check consists of reading from the store the contents of all storage locations into which the block of data was entered, producing the sum of these entries (ignoring overflows to the left of the radix point), and comparing it with a previously determined value.

9.2.3. Programs for Testing and Diagnosis

We will consider here the use of a number of special types of programs; namely, test, diagnostic, and tracing programs, which are useful in detecting computer malfunctions.

A test program causes various elements in a computer to function so that their responses may be tested for error. Specially designed test programs may be used to provide immediate indication of the approximate location of a fault. Final location and correction is then easy normally for failures of a definite nature, e.g., complete tube failures, nonoperating relays, open-circuited diodes, etc. A simple spot-check test program may be performed during maintenance time or even programmed for inclusion in the run of any given problem. A general test program may, for example, consist of all possible operations of which a computer is capable, listed sequentially. At the end of each step the computer compares the results against known answers. Upon detection of an error, the computer stops, indicating at what stage in the program something failed, and the approximate location of the fault. More specific test programs can then be used to check thoroughly the suspected parts. The source of trouble is finally located by checking the operation of individual circuits with the aid of an oscilloscope. Test programs are especially useful where the error is of an intermittent nature. However, not all parts of a machine may be tested by this method, e.g., the main control of the machine, where all fundamental wave forms are generated, must be tested by normal electronic techniques. However, a fault here means the computer will not obey the simplest instruction, thereby making the source of the fault relatively easy to locate.

A diagnostic type of program differs from a test program in that it is usually employed to locate the source of an error once it is known to

exist, whereas a test program is used to determine whether some part or all of the computer is functioning properly. The features of any specific diagnostic program depend on the engineering design of the computer for which it is intended, although certain requirements are essential to most. For a diagnostic program to work, the instructions must be executed properly. This implies certain parts of the computer must be in working condition. As a result, diagnostic programs are not generally useful to detect faults such as those in certain important control circuits, or in the power supply. These usually require test instruments for diagnosis. Also, it is not generally feasible to have one inclusive diagnostic program for a computer. Instead, a set of specialized programs, aimed at diagnosing the operation of parts of the computer are employed. In cases where the approximate location of the fault is known, the appropriate diagnostic routine may be selected to aid in quickly narrowing the possibilities. Where the location of the source is not known at all, the diagnostic programs are still of value, but the entire computer must be examined in some systematic manner.

In starting a problem, one or two special automatically-computed pilot problems may be used as test cases. If there are differences between expected results and these test cases, the program may readily be "traced" to detect the area of the program where trouble occurs by means of a tracing program. Once these test cases have been checked out, they may be stored on some input medium (punched cards, magnetic tape, etc.) together with the trace routines and thus be available for future testing of machine reliability for that problem. A trace routine is an interpretive type of program designed to assist the programmer in locating errors in a program. In this case, the interpretation causes instructions in addition to those in the main program to be executed. For example, a trace routine may cause each instruction or specified intermediate instructions to be printed upon execution. This record may be used for many purposes: to tell whether jump instructions were obeyed as expected, to indicate whether desired items of data entered into the computations at specified points, to provide a record of the contents of arithmetic registers at the end of a program step so results can be checked. Since the use of a trace routine increases the time required to execute the main program, the extent of its use is limited accordingly.

For intermittent errors which do not repeat during the course of a trace routine, tracing is unprofitable, and it becomes necessary to continue computation at the last point where results are known to be correct. To commence computation at such a point, a so-called roll back procedure must be programmed. Such a program stores all information necessary

to resume the computation in the particular addresses used in the standard program for a problem. The items of this information may be available in the input–output medium used at the beginning or middle of the computation or in computed results already printed. The programmer must foresee where a roll back will start (which may be arbitrarily selected) and then plan his standard program so as to compute and print or store on an input–output medium each item of such information. Once the necessary items are available, a standard roll back program is prepared on an input medium, which will assign the information to standard addresses in the problem program. This roll back program includes all data and instruction codes necessary to provide a configuration of storage standard to the problem program, such that a standard set of instructions can then be fed into the computer to continue the computation in a normal manner.

9.2.4. Preventive Maintenance

The simplest type of preventive maintenance procedure consists of methodically checking, by means of test instruments, the operation of the various circuits within a computer. This type of checking procedure consumes a large amount of time, and therefore is usually limited to a check of the fundamental waveforms only. A common procedure for this type of checking is to inspect a number of sections each day so that in a specified period, of the order of several days, all important waveforms will have been checked.

Another type of preventive maintenance procedure frequently used is referred to as marginal checking. It requires the inclusion of special features in the original design of the machine which enable a displacement of circuit operating conditions, by a variable amount from the normal, to be applied to various circuits within a machine. The difference in voltage between a specified nominal value and that at which the circuit fails is defined as an operating margin. A marked tendency of a lessening in this margin in a particular section of a computer is an indication that one or more components are deteriorating toward a point which would cause failure. Intermittent faults, caused by slow deterioration of components, can result in a circuit failing to operate correctly on certain pulse patterns. Variations of circuit conditions from the normal by means of a marginal check can cause a marginal circuit fault to be converted to one which is well defined and, therefore, more readily identifiable. This permits some warning to be obtained of the imminence of marginal conditions before they can cause errors in operation. Though circuits with incipient faults can be made to fail, others will operate satisfactorily. This type of check-

ing facility may be designed to be applied to the entire computer at once or to selected groups of circuits, as well as to individual circuits. A voltage variation scheme is used to vary supply voltages in any of various isolated sections of the computer to a point where steady failures occur. These voltages are varied slowly while the computer is executing some of its test programs. When a fault occurs, its origin may be traced as described in Section 9.2.3.

9.3. Error Minimizing Codes

We will present here an example of how selection of a particular code can influence the probability of error in interpretation of the data represented by the code. To illustrate the point we will consider a type of encoder widely used to produce a digital representation of a shaft position. In one of its forms, one or more so-called code disks are mounted on the shaft. There are several concentric bands on the disk, each corresponding to a particular bit position of the binary representation of the shaft position. The innermost band, which represents the most significant bit position, is divided into two segments and each band is divided into twice as many segments as the one radially inward from it. Each segment in a band is different in respect to a particular physical parameter than the segments next to it, e.g., electrically conducting or nonconducting, if the segments are to be sensed electrically, transparent or opaque if they are to be sensed optically. One property represents a 1 and the other a 0. The resolution of measurement can be no better than the width of the segments in the band for the least significant bit. The binary coded representation of the displacement of the shaft from a reference line is obtained by sensing the type of segment in each band on a radial line along which a set of sensors is located.

If the bands are sensed serially, an error can result from the motion of the disk from one defined position to the next during the sampling period. Thus, the number sensed will consist of the first few bits of the number present at the beginning of the process while the other bits are obtained from the succeeding number. This type of error may be avoided by making the sampling time less than the minimum time required for traversal of the disk from one defined position to the next. However, the source of error in which we are primarily interested here is common to both serial and parallel sampling methods. It presents itself when, at the time of sensing a particular band, the boundary of two segments is presented to a sensor. The sensor may then produce a signal corresponding to the value of the segment lying either to the left or the right of the boundary. The signal produced will depend on the sensor's sensitivity,

the precision of delineation of the segments, and the degree of alignment of the sensors and disk. In any event, if the two positions defined on either side of the boundary differ in several bit positions, an error may be produced in each of these positions, resulting in a value different from either of the positions by a large magnitude.

Large errors of this type may be avoided by use of codes known as Gray or cyclic codes which have the characteristic that any two coded representations defined to differ in value by only a single increment differ in the value of only one bit position. Thus, there can never be ambiguity in more than one bit position, so the error cannot be greater than a single increment. A difficulty with the use of a Gray code is that normal arithmetic operations cannot be performed on numbers expressed in this form, and conversion to a normal binary code of a number in serial form is somewhat complicated. However, the translation is simpler for a particular type of Gray code known as the reflected binary code (shown in Table 9.4). Inspection of Table 9.4 shows that the four right-hand bits of the representations of 16 through 31 are the same as that of 0 through 15 only in reflected order. Also, inspection of any two adjacent columns shows that the pairs of bits traverse the following sequence of values: 00, 01, 11, 10, 10, 11, 01, 00 (in which the last four values are in reflected order compared to the first four). Other cyclic codes may be obtained from Table 9.4 by interchanging any two columns.

TABLE 9.4. A comparison of the binary and reflected binary codes

Decimal	Binary	Reflected binary	Decimal	Binary	Reflected binary
0	0000	0000	16	10000	11000
1	0001	0001	17	10001	11001
2	0010	0011	18	10010	11011
3	0011	0010	19	10011	11010
4	0100	0110	20	10100	11110
5	0101	0111	21	10101	11111
6	0110	0101	22	10110	11101
7	0111	0100	23	10111	11100
8	1000	1100	24	11000	10100
9	1001	1101	25	11001	10101
10	1010	1111	26	11010	10111
11	1011	1110	27	11011	10110
12	1100	1010	28	11100	10010
13	1101	1011	29	11101	10011
14	1110	1001	30	11110	10001
15	1111	1000	31	11111	10000

Observation of Table 9.4 shows that the value of any bit, R_i, in the reflected code is a function of the corresponding and next more significant orders of the normal binary code. The following Boolean algebraic equations expressing the relationship are derivable from Table 9.4 and may also be checked by reference to the table.

$$\begin{aligned} R_n &= B_n \\ R_i &= B_{i+1}\bar{B}_i + \bar{B}_{i+1}B_i \qquad i \neq n. \end{aligned} \tag{9-1}$$

We will now consider how a number in the reflected binary code can be converted to a number in the normal binary code. At first glance it appears that the value of B_i depends on the values of R_i, R_{i+1}, . . . R_n. The conversion process may be simplified, however, if it is performed in a serial manner wherein B_n is generated first. In this process the value of B_i depends only on the values of B_{i+1} and R_i. Specifically

$$\begin{aligned} B_n &= R_n \\ B_i &= B_{i+1}\bar{R}_i + \bar{B}_{i+1}R_i. \end{aligned} \tag{9-2}$$

Thus, the conversion can be performed by a single flip-flop, B, whose state at any time t depends on its own value at time $t-1$ and the value of R_i at time t.

Actually, it is not absolutely necessary to convert from reflected binary to a normal binary code in order to perform arithmetic. A method for operating on numbers expressed in a modified reflected binary code is described in a paper by H. M. Lucal (see bibliography). The modification consists of adding an even parity check bit to the reflected code representation. A principal advantage of the modified code is that it can be used for error detection in arithmetic operations as well as in data transmission. A disadvantage is that the adder–subtractor circuitry required is somewhat more than twice that required for a conventional binary adder.

It should be pointed out, in passing, that nonambiguous reading is possible without the use of a Gray code by means of a technique which employs two sensors for each band except the one for the least significant bit position. See Susskind [1958].

9.4. Round-Off Errors

There are two sources of error that commonly occur in any numerical approximation procedure whether performed manually or by machine. These are due to the finite length of the numbers carried through the computation (round-off error) and the finite number of terms used in

certain operations (truncation error), e.g., computing the value of a function from the first few terms in an infinite series, or representation of an integral by a finite sum of terms (see Section 8.3 and Fig. 8.5). Specific techniques for reducing truncation error are outside the scope of this book and the reader is referred to texts on numerical analysis. However, in passing, we may mention that once a particular computing procedure has been selected, the error in the result can be limited to an acceptable size by adjusting the values of its parameters. The procedure is somewhat circular in that, if alternate procedures are available, error estimates can be made for each and selection made on this basis.

It is important to distingush between the absolute round-off error (determined by the number of places carried to the right of the radix point) and the relative round-off error which is the ratio of the absolute round-off error to the number itself and, therefore, a function of the number of significant bits retained. One of the most insidious sources of error in a computation is the large magnitude of relative error produced when the result of an algebraic addition is a number with fewer significant bits than either of the operands.

A round-off error in a machine can arise either when a number is initially entered and terminated at an intermediate point in order to be accommodated within the machine, or terminated after an arithmetic operation, e.g., after addition (subtraction) when the sum (difference) has to be shifted one place to the right to avoid an overflow, after multiplication of two n-bit fractional numbers, producing a $2n$-bit product, or a division resulting in a nonterminating quotient. Though round-off errors cannot be avoided completely, means can be utilized to keep them as small as possible.

If it is desired to carry more bits than the length of a single register will allow, and to reduce the round-off error in arithmetic operations like multiplication, division, square rooting, etc., recourse may be had to the use of multiple precision techniques, which will be described for double-precision addition and multiplication. As a matter of convenience, the less and more significant halves, l and m, respectively, of each double-precision number are stored in two consecutive storage locations. To add two double-precision numbers, $m_1 + l_1$ and $m_2 + l_2$, l_1 and l_2 are added first and the sum stored in a specified location. This is followed by addition of m_1 and m_2 and the carry (if any) produced in the addition of l_1 and l_2. In double-precision multiplication, the partial products l_1m_2, m_1l_2, and m_1m_2 are produced, the forming of l_1l_2 being omitted because of its insignificance, and the double-precision product formed by addition (according to the rules described) of the partial products. These examples show the

greatly increased cost in extra storage and increased computing time of multiprecision arithmetic.

In producing n-digit approximations to the products and quotients of two n-digit numbers, a round-off procedure should not introduce bias and the standard deviation (used as a measure of the dispersion of the error) should be minimal. In general, an n-digit number x entered into a machine already represents a true value x_t which has been rounded off:

$$x_t = (\cdot\ a_1 a_2\ \ldots\ a_n\ \ldots) \tag{9-3}$$

$$x = (\cdot\ b_1 b_2\ \ldots\ b_n) \tag{9-4}$$

where the difference $|x_t - x|$ depends on how x is formed from x_t. The product and quotient of two n-digit numbers is:

$$xy = (\cdot\ c_1 \ldots c_n c_{n+1} \ldots c_{2n}) \tag{9-5}$$

$$x/y = (\cdot\ d_1 \ldots d_n \ldots d_{2n} \ldots) \tag{9-6}$$

(although the length of registers in a computer limits x/y, as well as xy, to 2 n positions. Before considering how to round off xy and x/y to n-digit numbers, it is pertinent to inquire whether they are unbiased approximations of $x_t y_t$ and x_t/y_t, respectively. If the digits to be discarded are considered random variables with equally likely values (any two digits being treated as statistically independent), then

$$\overline{(xy)} = \bar{x}\bar{y} = \bar{x}_t \bar{y}_t \tag{9-7}$$

where the bars refer to mean values. However, if x and y are closely correlated, e.g., if $x = y$, there is a bias of the order of r^{-2n} (r being the radix). If the quotient x/y is considered in terms of xy^{-1}, it is apparent that if y is an unbiased estimate of y_t, y^{-1} is not an unbiased estimate of $y_t{}^{-1}$. In fact, $\overline{(1/y)} - 1/\bar{y} \approx y^3 \overline{(y - {}_t)^2}$.

Of the round-off procedures described in Section 9.4.1, the two prinpical classes are as follows: In one, all digits beyond the $n-1$st are ignored and the nth digit is always set equal to $r/2$. In the other, $r/2$ is added to the $n+1$st digit and the first n digits of the sum retained. When applied to nonterminating numbers $(\cdot\ a_1 a_2 \ldots a_n \ldots)$ the round-off error has the following characteristics (using our earlier assumption about the nature of discarded digits). In the former case, the error is over the interval $-r^{-n}$ to r^{-n}, yielding a mean of zero and a standard deviation of $(1/\sqrt{3})r^{-n}$. With the other procedure, the interval is $-r^{-(n+1)}$ to $r^{-(n+1)}$, also yielding a mean of zero, but a standard deviation of $(1/\sqrt{12})r^{-n}$. Because a $2n$ bit product does not satisfy the condition of being a nonterm-

inating number, the mean error may have a bias of the order of r^{-2n}.

Although the preceding discussion has shown biases may enter in various ways, they are small enough (of the order of r^{-2n}) to be considered negligible.

9.4.1. Mechanization of Round-Off Procedures

In this section we will consider first a number of round-off procedures for use with the binary system. This will be followed by a description of similar systems for use with the decimal system. The binary round-off procedures follow:

(1) The procedure described here is the most straightforward of all. It consists of adding a 1 in the highest order which is to be dropped. This is equivalent to adding 1 in the least significant bit to be retained if the bit in the highest order to be dropped is 1, for only then will a carry be propagated. However, if the 2's complement notation for negative numbers is used, round-off will occur in a direction opposite to that desired. This may be avoided by converting the number to its true representation before rounding. If the 1's complement notation for negative numbers is used, the desired rounding may be produced by subtracting 1 from the highest order to be discarded. The additional equipment required may be excessive especially for division, because the quotient may otherwise be formed in a register having no adding or carry propagating facility. The bias for this method is essentially zero, the variance is $1/12(2^{2n})$, and the standard deviation $1/\sqrt{12}(2^n) = 0.29$ times the last digit.

(2) This procedure, though simpler, retains the desirable characteristic of producing an unbiased result, i.e., the result has an equal probability of lying above or below the exact result. The procedure is to make the lowest order bit retained a 1 irrespective of the value of less significant bits. This method may produce an error twice as great as method (1), but is easily incorporated into a computer, and the average error over a large number of round-offs is usually sufficiently small. A minor disadvantage of this method is that zero is never produced after a round-off operation. The bias for this procedure is also essentially 0, the variance is $1/3\ (2^{2n})$, and the standard deviation is $1/\sqrt{3}\ (2^n) = 0.58$ times the last digit.

This method has certain advantages in a mechanized process. In multiplication, it can be used even though the $n+1$st place has been lost because of a right shift. This method is useful in nonrestoring division (see Section 6.1.6.1.2) because even though the quotient is formed in a

register without carry propagation capability, it allows the approximate quotient to be formed as soon as its first $n-1$ digits are known. Referring back to the nonrestoring division process, it is seen that the quantity 2^{-n} (in the correction term $[1+2^{-n}]$) corresponds to this round-off procedure. One may further justify using this scheme in division while using the one with a smaller dispersion for multiplication on the grounds that, generally speaking, division is a less frequent operation.

(3) In this procedure, a 1 is added to the lowest order bit retained when that bit is a 1, and nothing is done when it is 0. Consequently, it is possible to obtain a zero result. This method has the feature of not having to temporarily retain any bits that will be dropped subsequently, but requires an accumulator with carry propagating facilities.

(4) In this procedure a 0 or a 1 is added at random into the lowest order bit to be retained. This method has the feature of not having to temporarily retain any bits that will be subsequently dropped, but, besides requiring that the number to be rounded be placed in a unit with carry propagating facilities, it also requires a random number generator. An important objection to this method is that it is practically impossible to repeat computations exactly for checking purposes.

A number of round-off procedures for the decimal system, similar to the ones for the binary system are described below in corresponding order.

(1) This procedure consists of adding a 1 to the lowest order digit retained if the value of the highest order digit discarded is 5 or greater. The bias for this method is essentially zero, as for the corresponding round-off scheme with binary numbers, and the standard deviation is $(1/\sqrt{12})10^{-n} = .29)10^{-n})$. If it is inconvenient to sense the highest order digit to be discarded, an alternate, equivalent method may be used. The number to be rounded off is placed into an accumulator and a 5 is added to the highest order to be dropped or, what is equivalent, the highest order digit to be discarded is multiplied by 2. Either way a carry is produced that adds 1 to the least significant digit to be retained, if the most significant digit to be discarded is 5 or greater.

(2) The lowest retained digit is made equal to 5 regardless of other considerations. This procedure satisfies the requirement that the rounded number has equal probability of being greater or smaller than the exact value, i.e., the error averaged over a large number of round-off operations approaches zero. The bias for this method too is essentially zero and the standard deviation is $(1/\sqrt{3})10^{-n} = .58(10^{-n})$.

(3) A 1 is added to the lowest order digit retained if that digit is even, and 0 if it is odd. This is equivalent to making the last bit always equal to 1 for a decimal in the conventional binary code. A minor disadvantage

of this procedure is that 0 is never produced after a rounding operation. This disadvantage may be avoided by the following modification: a 1 is added to the lowest order digit retained when it is odd instead of even. However, it now becomes necessary to provide for the propagation of the carry which will occur when the lowest order bit is 9.

(4) A 0 or a 1 is added in a random manner into the lowest order retained regardless of other considerations. The average error here too approaches zero for a large number of operations.

Table 9.5 summarizes the procedures described.

TABLE 9.5. Listing of corresponding binary and decimal round-off procedures

Binary	Decimal
1. Add 1 to B_r if $B_d = 1$	1a. Add 1 to D_r if $D_d \geqslant 5$ 1b. Add 5 to D_d
2. Always set $B_r = 1$	2. Always set $D_r = 5$
3. Add 1 to B_r if $B_r = 1$	3a. Add 1 to D_r if D_r is even 3b. Add 1 to D_r if D_r is odd
4. Add 1 to B_r at random	4. Add 1 to D_r at random

B_d, D_d: most significant bit (digit) to be discarded
B_r, D_r: least significant bit (digit) to be retained

9.4.2. SEQUENCING ARITHMETIC OPERATIONS FOR MINIMUM ROUND-OFF ERROR

In a sequence of arithmetic operations involving addition, multiplication, and division one particular ordering may be preferable to another because the cumulative effect of round-off errors is less. That particular ordering will be considered superior which provides the least maximum error. In the discussion to follow (based on the treatment and notation of Von Neumann and Goldstine [1957]) our assumptions are as follows: 1) the digital numbers $a, b, c \ldots$ have n places to the right of the radix point, which is fixed, and may have either a positive or negative sign; 2) the rounded-off product (of n places), the so-called pseudo-product, will be denoted by $a \times b$ to distinguish it from the unrounded (true) product ab; the pseudo-quotient will be denoted by $a \div b$ to distinguish it from the true quotient a/b. No pseudo operations are involved in addition and subtraction.

First of all, it is apparent that the commutative law of multiplication applies to pseudo-multiplication as well:

$$a \times b = b \times a \tag{9-8}$$

However, the distributive and associative laws of multiplication are replaced in pseudo-multiplication by inequalities involving the round-off error $r^{-n}/2$ (where r, the radix, may be 2, 3, . . .). These are shown as Equations (9-12) and (9-15), respectively. Consider first the two basic inequalities:

$$|(a \times b) - ab| \leqslant r^{-n}/2 \tag{9-9}$$

$$|(a \div b) - a/b| \leqslant r^{-n}/2 \tag{9-10}$$

Using Eq. (9-9), it is apparent that

$$|(a + b) \times c - (ac + bc)| \leqslant 3(r^{-n}/2) \tag{9-11}$$

Since the left hand side of the inequality must be an integer multiple of r^{-n}, Eq. (9-11) reduces to

$$|(a + b) \times c - (ac + bc)| \leqslant r^{-n}/2 \tag{9-12}$$

Eq. (9-12) confirms our intuition about the best way to generate $ac + bc$ with the least upper bound for the round-off error. Since $|(a \times c) - ac| \leqslant r^{-n}/2$ and $|(b \times c) - bc| \leqslant r^{-n}/2$:

$$|(a \times c) - ac| + |(b \times c) - bc| \leqslant r^{-n} \tag{9-13}$$

(Also, adding a and b before multiplication produces the result with one less multiplication.)

The effect of round-off error on the associative law of multiplication will be considered next. To obtain an upper bound of the difference $|a \times (b \times c) - abc|$, we begin by adding $a(b \times c)$ and $-a(b \times c)$ to it and regrouping terms:

$$\begin{aligned} &|a \times (b \times c) - a(b \times c) + a[(b \times c) - bc]| \\ &\leqslant |a \times (b \times c) - a(b \times c)| + |a|\,|(b \times c) - bc| \\ &\leqslant r^{-n}/2 + |a|r^{-n}/2 \end{aligned} \tag{9-14}$$

Since $|a| < 1$, Eq. (9-14) reduces to

$$|a \times (b \times c) - abc| \leqslant r^{-n} \tag{9-15}$$

Two relationships will now be derived to show the effect on the total round-off error of the order in which a multiplication and division are performed. We start by considering the expression $|(a \div b) \times b - a|$. Adding $(a \div b)b$ and $-(a \div b)b$, and regrouping terms:

$$\begin{aligned} |(a \div b) \times b - (a \div b)b| + |[(a \div b) - a/b]b| \\ \leqslant r^{-n}/2 + |b|\, r^{-n}/2 \end{aligned} \tag{9-16}$$

For $|b| < 1$, the quantity on the right side of Eq. (9-16) is less than r^{-n} and, since the quantity on the left side must be an integer multiple of r^{-n}, it reduces to zero. (Also, for $|b| = 1$, the quantity $|(a \times b) \div b - a|$ must be zero.) Therefore:

$$|(a \div b) \times b - a| = 0 \tag{9-17}$$

In the next expression to be considered, namely $|(a \times b) \div b - a|$, the order of multiplication and division are reversed. Adding $(a \times b)/b$ and $-(a \times b)/b$, and regrouping terms:

$$\begin{aligned} |(a \times b) \div b - (a \times b)/b| + |[(a + b) - ab]/b| \\ \leqslant r^{-n}/2 + |b|^{-1} r^{-n}/2 \end{aligned} \tag{9-18}$$

Since $|b| < 1$,

$$|(a \times b) \div b - a| \leqslant |b|^{-1} r^{-n} \tag{9-19}$$

The maximum error as shown by Eq. (9-19) compares unfavorably with that of Eq. (9-17) and even with Eq. (9-16), especially for $|b| \ll 1$. Thus, it is advantageous to first divide and then multiply.

Let us now return to the subject of double-precision arithmetic introduced in Section 9.4 to consider how it may be effectively used to reduce round-off error. For an example, consider the generation of a sum of products:

$$\sum_{i=1}^{m} a_i b_i$$

If each $2n$ place product is rounded off to n places before addition to the partial sum:

$$\left|\sum_{i=1}^{m} a_i b_i - \sum_{i=1}^{m} a_i \times b_i\right| = \left|\sum_{i=1}^{m} (a_i b_i - a_i \times b_i)\right| \leqslant r^{-n}/2 \tag{9-20}$$

On the other hand, if instead, each $2n$ place product is retained intact and only the final sum is rounded-off

$$\left|\sum_{i=1}^{m} a_i b_i - \sum_{i=1}^{m} {}^{*}\, a_i b_i\right| \leqslant r^{-n}/2 \tag{9-21}$$

where $\sum_{i=1}^{m}{}^{*}$ denotes the $2n$ place sum after round-off. The maximum error shown in Eq. (9-21) is actually less than the probabilistic

(mean) error, using single precision arithmetic, namely: $.29m^{1/2}\, r^{-n}$.

We conclude this section by considering the effect of round-off error on scaling procedures (described in Section 6.3). First of all, we note that adjustment of a result by multiplication by an integer ($\pm 2, \pm 3, \ldots$) is not a pseudo-operation since it is equivalent to repeated additions or subtractions. However, division by an integer is a pseudo-operation. To eliminate a cumulative effect in successive divisions by a scale factor, the associative law should hold, namely:

$$(a \div s) \div m = a \div sm \tag{9-22}$$

Eq. (9-22) will hold provided only those values are used for s which are powers of a fixed integer $r(= 2, 3, \ldots)$, and by defining $a \div s$ not as the result of a single division of a by s, but of a p times iterated division of a by r:

$$a \div r^p = (\ldots ((a \div r) \div r) \ldots) \div r \qquad p \text{ times}$$

The smaller r, the more precise the adjustments of scale based on it. Since $r = 2, 3, \ldots$ this suggests the use of $r = 2$. Also, matters are simplified if r is set equal to the base of the number system employed. Use of the binary system satisfies both considerations, and this is an important argument in favor of its use in electronic computers.

LITERATURE

RELIABILITY

Bazovsky, I. [1961] *Reliability: Theory and Practice,* Prentice-Hall, Englewood Cliffs, N. J.

Benner, A. H. and Meredith, B [1954] Designing reliability into electronic circuits, *Proc. 1954 National Electronics Conference,* 137-145, National Electronics Conference, Inc., Chicago.

Bloch, R. M., Campbell, R. V. D. and Ellis, M. [1948] Logical design of the Raytheon computers, *MTAC,* 286-95, 317-23.

Brown, W. G., Tierney, J. and Wasserman, R. [1961] Improvement of electronic computer reliability through the use of redundancy, *IRE Trans. El. Comp.,* **10,** 407-416.

Dimsdale, B. and Weinberg, G. M. [1960] Programmed error correction in Project Mercury, *Comm. ACM,* **3,** 649-651.

Flehinger, B. J. [1958] Reliability improvement through redundancy at various systems levels, *IBM J. Research and Develop.,* **2,** 148-158.

Holden, P. [1962] The "rate of change factor" in reliability, *Electro-Technology,* **69,** No. 1, 53-55.

Läfgren, L. [1958] Automata of high complexity and methods of increasing their reliability by redundancy, *Information and Control,* **1,** 127-147.

Landauer, R. [1961] Irreversibility and heat generation in the computing process, *IBM J. Research and Develop.,* **5,** 183-191.

Liddell, D. W. [1962] Integration and automatic fault location techniques in large digital data systems, *Proc. AFIPS Spring Joint Computer Conference,* 213-224.

Lloyd, D. K. and Lipow, M. [1962] *Reliability: Management, Methods and Mathematics,* Prentice-Hall, Englewood Cliffs, N.J.

Pedelty, M. J. [1962] Neuron technology builds reliable circuits from unreliable components, *Control Engrg.,* **9,** No. 5, 115-119.

Schwartz, L. S. [1961] Reliability through redundancy and error-checking codes, *Electro-Technology,* **67,** No. 2, 123-130.

Scrivner, J. H. and Willey, J. R. [1962] Proving long term reliability (for alloy transistors) *Electronic Industries,* **21,** No. 5, 102-106.

Swanson, J. A. [1960] Physical versus logical coupling in memory systems, *IBM J. Research and Develop.,* **4,** 305-310.

Weinberg, G. M. [1961] Programmed error correction on a decimal computer, *Comm. ACM,* **4,** 174-175.

Von Neumann, J. [1956] Probabilistic logics and the synthesis of reliable organisms from unreliable components, "Automata Studies," Princeton Univ. Press.

Weiss, G. H. and Kleinerman, M. H. [1954] On the reliability of networks, *Proc. 1954 National Electronics Conference,* 128-136, National Electronics Conference, Inc., Chicago.

ERROR DETECTING AND CORRECTING CODES

Abramson, N. [1959] A class of systematic codes for non-independent errors, *IRE Trans. Information Theory,* IT-5, 150-157.

Balser, M. [1954] *IRE Trans. Information Theory,* PGIT-4, 50-63.

Brown, D. T. [1960] Error detecting and correcting binary codes for arithmetic operations, *IRE Trans. El. Comp.,* **9,** 333-337.

Calingaert, P. [1961] Two-dimensional parity checking, *J. ACM,* **8,** 186-200.

Elspas, B. and Short, R. A. [1962] A note on optimum burst—error—correcting codes, *IRE Trans. Information Theory,* **8,** 39-42.

Elias, P. [1954] Error free coding, *IRE Trans. Information Theory,* PGIT-4, 29-37.

Golay, M. [1954] Binary coding, *IRE Trans. Information Theory,* PGIT-4, 23-28.

Griesmer, J. H. [1960] A bound for error-correcting codes, *IBM J. Research and Develop.,* **4,** 532-542.

Hagelbarger, D. W. [1959] Recurrent codes: easily mechanized, burst-correcting, binary codes, *Bell Syst. Tech. Jour.,* **38,** 969-984.

Hamming, R. W. [1950] Error detecting and error correcting codes, *Bell Syst. Tech. Jour.,* **29,** 147-160.

Huffman, D. A. [1956] A linear circuit viewpoint on error correcting codes, *IRE Trans. Information Theory,* IT-2, 20-28.

Huffman, D. A. [1953] A method for the construction of minimum redundancy codes, *Communication Theory* (W. Jackson, ed.) pp. 102-110, Academic Press, New York.

Kilmer, W. L. [1959] An idealized over-all error-correcting digital computer having only an error-detecting combinational part, *IRE Trans. El. Comp.,* **8,** 321-325.

Laemmel, A. E. [1953] Efficiency of noise-reducing codes, *Communication Theory* (W. Jackson, ed.) pp. 111-118, Academic Press, New York.

Lloyd, S. P. [1957] Binary block coding, *Bell Syst. Tech. Jour.,* **36,** 517-535.

Lucal, H. M. [1959] Arithmetic operations for digital computers using a modified reflected binary code, *IRE Trans. El. Comp.,* **EC-8,** 449-458.

MacDonald, J. E. [1960] Design methods for maximum minimum-distance error-correcting codes, *IBM J. Research and Develop.*, **4**, 43-57.

Marcus, M. P. [1961] Minimum polarized distance codes, *IBM J. Research and Develop.*, **5**, 241-248.

Meggitt, J. E. [1961] Error correcting codes and their implementation for data transmission systems, *IRE Trans. Information Theory*, **7**, 234-244.

Meggitt, J. E. [1961] Error-correcting codes for correcting bursts of errors, Trans. AIEE, Pt. 1, 79, 708-711; *IBM J. Research and Develop.*, **4**, 329-334.

Melas, C. M. [1960] A new group of codes for correction of dependent errors in data transmission, *IBM J. Research and Develop.*, **4**, 58-65.

Ralston, A. [1957] Error detection and error correction in real-time digital computers, *Proc. 1957 Western Joint Computer Conference, Los Angeles*, 179-188.

Reed, I. S. [1954] A class of multiple-error-correcting codes and the decoding scheme, *IRE Trans. Information Theory*, PGIT-4, 38-49.

Sacks, G. E. [1958] Multiple error correction by means of parity checks, *IRE Trans. Information Theory*, IT-4, 145-147.

Shannon, C. E. [1948] A mathematical theory of communication, *Bell Syst. Tech. Jour.*, **27**, 379.

Siforov, V. I. [1956] On noise stability of a system with error correcting codes, *IRE Trans. Information Theory*, IT-2, 109-115.

Silverman, R. A. and Balser, M. [1954, 1955] Coding for constant data rate systems —Part I, A new error correcting code, *Proc. IRE*, **42**, 1428-1435; and Part II, Multiple error correcting codes, *Proc. IRE*, **43**, 728-733.

Slepian, D. [1956] A note on two binary signalling alphabets, *IRE Trans. Information Theory*, IT-2, 84-86.

Slepian, D. [1956] A class of binary signalling alphabets, *Bell Syst. Tech. Jour.*, **35**, 203-234.

Susskind, A. K. (ed) [1958] *Notes on Analog-Digital Conversion Techniques*, Wiley, New York.

NUMERICAL ANALYSIS

Abramowitz, M. and Stegun, I. A. [1956] Pitfalls in computation, *Jour. Soc. Indust. Appl. Math.*, **4**, 207-219.

Alt, F. L. [1958] *Electronic Digital Computers, Their Use in Science and Engineering*, Academic Press, New York.

Hartree, D. R. [1952] *Numerical Analysis*, Oxford Univ. Press, London and New York.

Hildebrand, F. B. [1956] *Introduction to Numerical Analysis*, McGraw-Hill, New York.

Householder, A. S. [1953] *Principles of Numerical Analysis*, McGraw-Hill, New York.

Lanczos, C. [1956] *Applied Analysis*, Prentice Hall, Englewood Cliffs, New Jersey.

Milne, W. E. [1949] *Numerical Calculus*, Princeton Univ. Press, Princeton, New Jersey.

Von Neumann, J. and Goldstine, H. H. [1947] Numerical inversion of matrices of high order, *Bull. Amer. Math. Soc.*, **53**, 1021-1099.

Appendix: Input–Output Equipment

There are many devices which may be used to enter data into a digital computer and to display or record the results of its computations. Small amounts of data may be entered by means of a keyboard and switches, and often by an electric typewriter, on a control console. The console, which may be situated either at the computer site or remotely, also usually contains indicator lights and other simple indicators which allow the operator to monitor certain basic items of information within the system. The control console is used primarily for monitoring and testing purposes since too much time would be consumed in entering all data into a computer manually. Higher speed data entry devices include electromechanical or photoelectric readers of punched paper tape, punched-card readers, magnetic-tape readers, automatic graph followers, etc. Data generated by the computer may also be recorded, printed, or displayed in a number of ways. Records such as punched cards, punched paper tape and magnetic tape are of a form that is not only suitable for retention and availability for future automatic processing by a computer, but can also be used to produce printed records by means of off-line equipment. Printed records can be produced either a character at a time (by means of a typewriter), a line at a time (by means of medium and high speed mechanical printers) and a page at a time (by means of electronic printers). Visual displays differ widely in the way characters are formed and illuminated. Graphical records may be produced by various types of automatic plotting devices.

When incorporated into a control system, a computer is also provided with input–output equipment capable of converting data from measuring instruments, in analog form, into digital signals, and converting the output of the computer into analog signals acceptable by the controllers and actuators of the system being controlled. For a survey of analog–digital conversion equipment, the reader is referred to Susskind [1957] and other entries in the bibliography of this appendix.

For commercial applications such as the automatic processing of bank checks and other documents, specially formed characters are printed on the document. These may be inspected visually and can be read automatically by specially designed reading heads that scan the characters and produce unique signals for each. The use of a special magnetic ink rather than printer's ink is used in order to lessen errors in the interpretation

process resulting from markings or obliterations of the printed characters in handling. For a general description of the automatic character recognition problem, see Chow [1957].

A.1. External Storage Media for Input–Output Functions

Where the medium of storage is readily separable from the sensing and transport mechanism, it is utilizable not only in conjunction with input and output devices, but also for external storage of unlimited capacity. Media that fall in this category are punched cards, punched paper tape, and magnetic tape. Moderate speed punches and mechanical readers for paper tape run about $1000, and photoelectric readers up to about $4000. Card punches and readers for tie-in to a computer vary in price by tens of thousands of dollars, depending on their complement of switching and storage circuits (though card preparation equipment is much less, e.g., a key punch is about $2000). High performance magnetic tape units designed for use with high speed data processors are about $20,000. These figures are only approximate, for prices of the newer units are changing, and there are many items such as the amount of buffer storage and special control circuitry that may or may not be included with some of these items that greatly affect the price.

A.1.1. Punched Paper Tape

Punched paper tape used as a data input or output medium for a digital computer commonly has 5, 6, 7, or 8 positions across the tape where circular holes may be punched, as well as a small sprocket hole for aiding the transport of the tape. A single row of bit positions across the tape is referred to as a character, and the individual bit positions comprising a character are designated as channels or levels. All bit positions in a row are punched or sensed simultaneously. Six levels are adequate for the coding of alphanumeric characters (either upper or lower case). Additional levels are used either when more than 64 characters are to be coded or more than 32 characters and a parity check bit are to be provided (see Chapter 9). Tape width varies slightly, from 11/16 in. for 5-level tape to 1 in. for 8-level tape. Holes are usually spaced 10 to the inch along the length of the tape.

A paper tape punch may be activated either from a keyboard or by control and data signals from a computer. Paper tapes produced by an operator for input to a computer are normally prepared off-line in order to allow the correctness of the data on the tape to be verified before entry into the computer, and to allow the data to be read in at a faster rate than an operator can type. When it is desirable, for any number of reasons,

to record output data from a computer on paper tape, the tape punch can be controlled by signals from the computer. Punched paper tapes may also be produced in conjunction with the preparation of data in other forms. For example, the keyboard used to produce a typewritten record on a typewriter or accounting machine may at the same time activate a paper tape punch. Though typewriter keyboards are primarily designed for the entry of alphanumeric data, they can also be used for the entry of binary data. This is accomplished by first dividing the binary numbers to be entered into groups of 5, 6, or 7 bits, according to the tape format used, and selecting a particular character to represent each binary coded group. Binary data may then be entered by depressing a corresponding sequence of keys, if switches are so chosen and placed that depression of each key causes a corresponding set of binary signals to be generated. (See Section A.2.1.)

Paper tape punches operate at speeds of from 10 to 300 characters/sec, with the majority operating at about 60 characters/sec. Two basic operations in a paper tape punch are the punching and the tape feed operation. Synchronization of the internal elements of the punch is obtained by built-in mechanical and electromechanical means. These internal interlocks assure that operations are performed in proper sequence with an adequate time interval between, e.g., the tape should not be advanced before the punch pins have been completely withdrawn from it, and the tape advance sufficiently completed before the tape is punched. Also, the input data must be so synchronized with the punch that the electromagnets that activate the punching mechanism are energized during the appropriate part of the punching cycle. Timing signals for this purpose are commonly obtained from some type of electrical pick-up placed on the drive-shaft.

A block diagram indicating the timing control for a paper tape punch

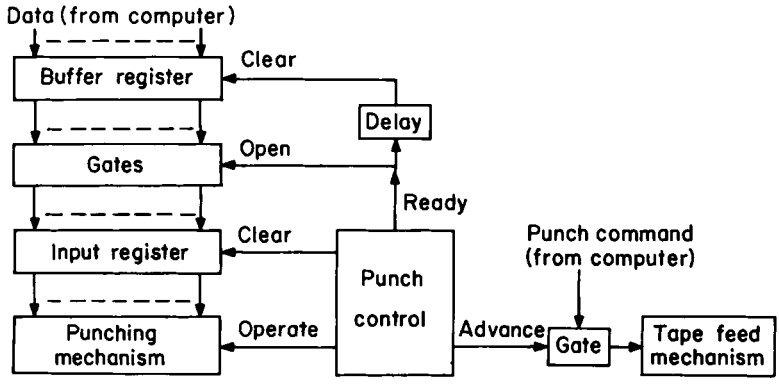


FIG. A.1. Block diagram of paper tape punch timing control

is shown in Fig. A.1. The punch control refers collectively to the internal source of timing signals. The tape feed mechanism is activated by a timing signal from the punch control, provided there is a punch command signal from the computer indicating that a new character is to be punched. The presence of a ready signal on the line shown indicates that it is the appropriate time in the cycle for the punching mechanism to be energized. This signal opens the gates shown, allowing the transfer of data from the buffer register to the input register, after which the ready signal is removed. Simultaneously, the operate signal causes activation of those punch plungers specified by the code in the input register. After the plungers are withdrawn a new advance signal is generated. There is buffering of the input data because of timing uncertainties between the times of its input and use by the punch. Inclusion of the buffering shown not only leaves the computer free for other operations between data insertions but allows the punch to operate near its maximum rate.

Punched paper tape readers are classified as mechanical or photoelectric, depending on the means used for sensing the presence of holes. The reading speed for mechanical readers varies from 20 to 60 characters/sec, and for photoelectric readers from 150 to 2000 characters/sec. Two basic operations performed by the reader are sensing and tape advancement. Synchronization of the internal elements of the reader is accomplished by its mechanical design. The tape feed mechanism advances the tape past a reading station where each character is sensed. The discrete times at which data may be read are distinguished from the times in between by an indexing mechanism which generates a signal when the row of hole positions is opposite the sensing mechanism. The indexing mechanism is associated with the small sprocket holes which may be sensed by means similar to that for sensing the data hole positions. The tape may be advanced either a character at a time or continuously. In discrete operation, when a signal from the reader indicates it is permissible to read from the tape, the data lines are sensed and the input gates to a buffer or static storage register in the computer are opened. At the same time, an advance signal is sent to the tape feed mechanism, to advance the next character to the reading station. Continuous operation allows higher reading speeds. In this case, pulses generated by the reader each time a character is sensed synchronize the computer to the reader. The duration of the reading process is controlled by start and stop signals from the computer.

A.1.2. Punched Cards

Punched cards differing in size and format were developed initially for use with different accounting machines. Only two of these cards have

been used to a great extent in digital computer systems. Both are a standard sized stiff paper card (3¼ in. × 7⅜ in. × 0.007 in.). The most widely used one, the IBM card, provides 12 row and 80 columnar positions where rectangular holes may be punched. The Remington Rand card provides 12 row and 45 columnar positions where circular holes may be punched.

Operating rates for card punches are in the range of 100–200 cards per minute. The operation of a card punch may be described in terms of its principal functional units, namely an input station (card hopper), a card feed and transport mechanism, an indexing mechanism, a punching mechanism, and an output station (card stacker). Synchronization of these elements is obtained by built-in mechanical and electro-mechanical means referred to collectively as the punch control. The events that take place in order to punch a single card are somewhat as follows: When a signal appears on the interlock line of the card indexing mechanism, indicating the punch is ready for a new cycle, the card feed mechanism is activated. This causes a single card to be extracted from the bottom of the stack in the hopper and advanced to the punching mechanism. When the card reaches it, a "card ready" signal appears which defines the period during which the punching operations for the whole card are completed. When the proper position on the card is reached, the appropriate punch plungers are actuated. The plunger drive signals are obtained from the output lines of a buffer register which holds the data to be punched in the first row or column, depending on the type of punch. Upon extraction of the plungers, the indexing mechanism generates a signal that allows the next row (or column) of data to be entered into the buffer register. After the last row has been punched, the "card ready" signal is removed. Completion of a full cycle causes the reappearance of a signal on the interlock line and the start of a new cycle.

A card reader may sense either a row or column or the entire card simultaneously. Reading rates for card readers are in the range of 100–1000 cards/min, the higher speed units having been specially developed for use with high speed computers. The functional units of the reader are similar to those of the punch. However, instead of a punching mechanism, there is a reading station where the presence or absence of holes in particular positions may be sensed either electromechanically by sets of brushes or photoelectrically. In either case, as many data output lines are provided as hole positions to be read simultaneously. When a card reaches the reading station, a "card ready" signal is generated by the sensing mechanism and when the card is properly positioned for sensing, a signal is generated by the indexing mechanism which can be used to gate signals from the data output lines to their destination. The indexing mechanism also generates an interlock signal which indicates that a card-

reading cycle has been completed and that a new card may be read.

When a card reader is used as an on-line input device to a digital computer, the computer must provide whatever signals are necessary to sequence the reader through its cycle of operation. To cause a new card to be read, it must generate a signal to actuate the card feed mechanism. This signal is generated only after it has been determined, by inspection of the interlock line, that the reading cycle for the last card has been completed. A signal indicating the presence of a card at the reading station is generated by the sensing mechanism. During the time the rows (or columns) are read, the indexing mechanism causes a serial train of indexing signals to be generated, each of which indicates a time at which there are signals on the data output lines, i.e., the output lines of the sensing stations. The computer must also be provided with logical circuitry to detect the failure of a card to arrive at the read station, indicated by absence of a card present signal, and to cause the indexing signals to be ignored. This action can be used to signal to the operator that either no cards remain in the hopper or the card reader is not operating properly.

A buffer is useful in conjunction with a card punch or reader for two main purposes. First, it is needed to compensate for the difference in data rates between these units and the computer. Secondly, it can be used to translate from computer code and format to one of a number of possible card codes and formats.

Punched cards are widely used because of the convenience or flexibility of the card itself, and, also, of the variety of equipment that has been developed for processing cards. The distinguishing feature of a card is that it provides a unit record readily separable from data on other cards. It also offers flexibility in that various codes may be used on a card to represent alphabetic characters, special symbols, or binary data. These features plus the variety of machines available for punching, verification, and duplication of cards facilitates the initial recording of data. To reduce the time required for data preparation, several key punches may be employed simultaneously, and the cards collected before entry of the data into the computer. The unit record feature may be used to advantage by punching one instruction to a card and using an assembly program to assemble them in the right order. If errors are detected or changes required, it is relatively simple for cards to be replaced or added, after which the corrected program is assembled.

There are a number of auxiliary card processing units which incorporate a punch or reader or both in conjunction with printing, sorting, duplicating, and limited computing devices. For example, a visual record of data on a punched card can be provided either by a unit called an interpreter, which reads a card and prints the corresponding data on the

same or different cards, or by a special key punch that prints data on a card concurrently with the punching operation.

To take advantage of the conveniences of preparation of data on cards while allowing data to be entered into a computer at the higher data transfer rates of magnetic tape reading, punched card to magnetic tape converters have been developed as an auxiliary unit of off-line equipment.

For detailed descriptions of punched card equipment, the reader is referred to Cemach [1951] and to various reference manuals of the IBM Corporation.

A.1.3. Magnetic Tape

Magnetic tape for computer applications usually has either a cellulose acetate or polyester (Mylar) base. The magnetic coating consists of about 80% iron oxide (Fe_2O_3 or Fe_3O_4 or a mixture) in an acetate or vinyl binder. Tape widths vary from ¼ to 1 in. Acetate tapes used have a base thickness of 1.5 mils while polyester base tapes of only 1.0 mil thickness can be used because of their greater tensile and yield strength. The coefficient of expansion with both humidity and temperature is also less for the polyester base tape. Another type of magnetic tape that has been used in computer applications has a magnetic coating consisting of a nickel–cobalt alloy which is plated onto a nonmagnetic metallic base having a highly polished conductive surface. However, it has not found as wide application, being used principally in the Remington Rand Uniservo tape unit. A major problem associated with magnetic tape is that of dropouts of recorded data resulting from tape imperfections. For this reason the quality of tape for computer usage must be much better than that for recording sound. A common size for reels is a diameter of 10½ in. Tape lengths usually vary between 2400 and 3600 ft.

It is common practice for words to be stored on magnetic tape in a serial-parallel fashion. For example, if a word in a binary machine has 36 bit positions, and 6 channels are available on the tape for data storage, the word would be stored in 6 parallel groups of 6 bits each. Other channels would also be provided for clock pulses, parity check bits, and block marker signals (defined later in this section).

In general, the considerations that enter into the design of record and read heads for use with magnetic tape are the same as those for heads to be used with magnetic drum or disk memories (see Chapter 5). One difference is that the heads are, as a rule, grouped together in a stack which may contain from 15 to 30 heads/in. This provides a convenient mounting arrangement while allowing a reasonable transverse packing density, i.e., number of channels per unit width of tape. The head stacks must be made

with great precision for misalignment of the individual heads with respect to the tape can cause various problems. For example, if the read heads were not adequately aligned along the channels with the record heads, the full recorded signal would not be picked up and the output voltage would vary accordingly. If the tape is placed on another unit whose heads are nct similarly aligned both within the stack and relative to a reference line external to the stack, there would be incomplete erasure of old data, which could result in the reading of unwanted data. In addition to the limitations on recording density imposed by the characteristics of the head and the recording medium, an additional limit is imposed by the precision with which the heads and tape can be aligned. For good longitudinal recording density, there must be precise alignment of track gap-center lines within the head stack. In high quality stacks, the gap scatter is held to a tolerance of 0.0001 in. The maximum longitudinal density obtainable is related to the amount of twist of the tape relative to the heads, for if the bits are packed too close, the head at one side of the tape may be picking up a bit from one row and that at the other side from another row. One way of reducing the effect of skew is to pack a given number of channels into a narrower section of tape. However, this also produces an alignment problem, for a smaller amount of tape slippage sidewise will now result in each head reading, erroneously, the contents of the adjacent channel. As a result of the tape riding in contact with the heads, the head laminations may be ground by the abrasive action of pigmented tapes, resulting in widening of the head gap. Also, as oxide particles accumulate on the head surface, the tape is lifted from the head gap, causing an appreciable signal loss. One manufacturer has alleviated this problem by providing recesses between the heads of a stack, in which loose oxide particles can collect between periodic head cleanings. Metal tapes are less abrasive and can be lubricated to reduce wear, but present other problems.

It is common practice to provide self-checking of the data on the tape. During recording, a parity bit is generated for each row and recorded in a channel provided for that purpose. In reading, the value of the parity bit is computed again and checked with the recorded value. Because a simple parity check like this detects only an odd number of errors in each row and tape drop-out errors are likely to be correlated, a longitudinal check is also provided in the form of a parity bit for each column of a block of data. The longitudinal parity bits are sometimes referred to as a check sum of the block of data. The use of both transverse and longitudinal parity bits allows a single error in a block to be corrected, since the bit in error will be the one at the intersection of the row and column where the parity bits indicate an error.

In utilizing magnetic tape in a computer system, it is necessary to be able to start and stop the tape at the command of the computer. Since the tape drive mechanism cannot stop the tape or accelerate it to full speed in a time short compared with the pulse repetition rate of recorded information, a certain amount of lead space, referred to as a gap, must be allowed between the blocks of recorded information. The length of the gap is the amount of tape moved from the issuance of a stop command until the tape is again moving at full speed after the issuance of a subsequent record or read command. Because of mechanical limitations, the deceleration and acceleration times are subject to slight variation. Therefore, after the issuance of a record command, a delay must be introduced before recording takes place adequate to insure that even under the worst conditions likely to be encountered the tape will have been accelerated to full speed by the end of the delay time. This provides assurance that whenever any record is consulted, data is read or recorded only when the tape is moving at the correct speed.

Because it is not only difficult to control exactly the start and stop times, but also to achieve constant tape speed, it is not practical to locate individual words by means of an indexing scheme based on counting pulses from an external clock source. A convenient way of locating a word is by its relative position in a block of data recorded by a single command. Each block can be located by a specific address or identifying tag stored within it. The length of blocks may be fixed or variable, depending on the associated tape control equipment. The extent of each block is defined by a block marker signal recorded at the beginning and end of each block. During subsequent reading operations these markers initiate reading and stopping of the tape.

It is desirable to record a large number of words per block for two important reasons. First, because the tape is normally at rest between recording and reading operations, each access to a block consumes a start and stop time during which no information is read or recorded. Therefore, the more data per block the lower the average access time to a unit of information. (For efficient operation, it is also desirable that a block of data transferred to the computer keeps it busy for a period large compared to the access time. This is the case, for example, when an iterative subroutine stored on the tape is called in.) Also, since the gaps between blocks contain no data, the longer a block the greater the ratio of used to unused space on the tape. It is apparent that a small start and stop time are desirable both to reduce access time and tape wastage. (The stop time is more wasteful of tape than the start time because the stop command that precedes and initiates mechanical action is given while the tape is running at full speed.)

Measures that may be employed to improve the start and stop times of various tape transports will be described briefly. We begin by considering first the basic elements of a tape transport, namely a drive unit that starts or stops the tape, moves it and controls its speed, and a tape reel assembly that supplies and takes up tape as required. The inertia of the loaded reels and their drive motors is considerably more difficult to overcome than the inertia of a small length of tape. Thus, to provide quick starts and stops, it is essential to maintain a slack loop of tape which can be accelerated faster than the reels. The speeds of the reel drive motors can be controlled by sensing the current amount of slack tape. Two principal types of slack loop systems are now commonly used. They differ both in the way the slack is maintained and its amount sensed. In one arrangement, the tape is formed into several zig-zag loops by means of guide rollers mounted on two movable arms. (See Fig. A.2.) The tape

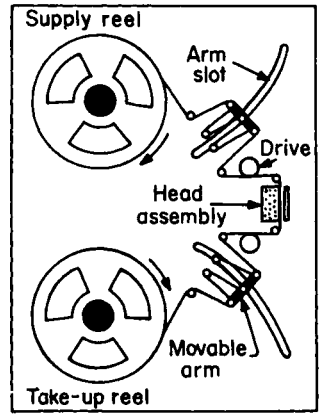


FIG. A.2. Magnetic tape transport with slack tape formed by zig-zag loops and movable arms

passing from the supply reel is formed into several loops on one arm, passes over a centrally located drive unit, is formed into loops on a second arm and finally wound on the take-up reel. When sudden changes in tape speed occur which tend to vary the tape tension, the arms move to either let out or take up slack, as required. This arm movement is sensed by a servo system that regulates the supply of tape by controlling the speed of the reel motors. In another widely used arrangement for maintaining slack, the tape is drawn into two columns by reduced air pressure (see Fig. A.3). Whether the slack supply is greater or less than a specified

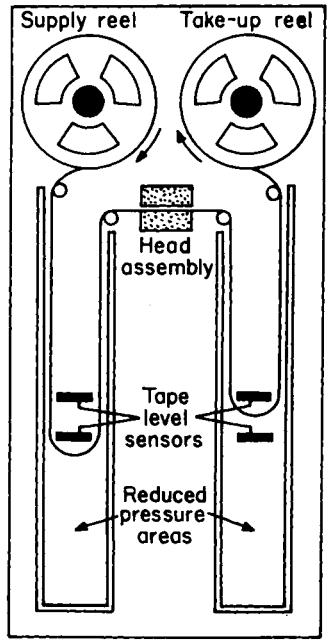


FIG. A.3. Magnetic tape transport with slack tape formed by air pressure

amount is sensed by pressure sensitive switches. The outputs of these switches are used as inputs to a servo system that controls the speed of the reel motors. Some speed characteristics of certain commercially available tape transports are listed in Table A.1.

TABLE A.1.

Model	Normal tape speed (in/sec)	Starting time (msec)	Stopping time (msec)
Ampex—FR-300	150	1.5	1.5
IBM—727	75	3.5	4.5
Potter—906	50–100	3	1.5
RCA—501	100	2	2
Sperry Rand—Uniservo	100	6.5	6.5

Of the three external storage media described, magnetic tape is the

most versatile. It provides the highest storage density and also the highest data transfer rate for both recording and reading. Also, both reading and recording functions are readily incorporated within a single unit. Finally, it is the only one of the three that provides the feature of erasibility.*

Because of its high data transfer rates, the use of magnetic tape for input–output operations reduces the ratio of time spent by the computer during these operations to time spent in other operations. For example, with other output recording media or high speed mechanical printers, the maximum data output rate is about 2000 characters/sec compared to a range of about 6000–60,000 characters/sec for present magnetic tape units. Several of its features make magnetic tape suitable as an external storage medium for the storage of libraries of programs (subroutines, compilers, etc.) and large files of data for temporary or permanent storage (where these terms may encompass a range of the order of minutes to years). The feature of erasibility is essential to its use as an auxiliary store in conjunction with either a medium speed internal store like a magnetic drum or disk, or a high speed store like a magnetic core array. Some pertinent points in comparing the use of a magnetic tape unit against a drum or disk for an auxiliary store are as follows: the tape offers a greater amount of storage (over 30 million bits for a 2400-ft reel, assuming a nominal recording density of 200 bits/in., 6 channels, and 10% tape wastage because of gaps) while the disk and drum are simpler mechanisms, and provide higher linear speeds and a shorter access time to a selected location.

Even though starting and stopping the tape is time-consuming, the relatively small access time per word obtainable when data transfer operations are limited to transfers of large blocks of data increases its usefulness as an auxiliary store. There are different ways for the computer to gain access to desired locations. For example, a program may be written so that blocks of information recorded on the tape can be counted. If tables of functions are stored, the arguments and values can be stored in alternate blocks. Then, through programming, the computer can examine, by a comparison operation, all arguments till it comes to the right one, at which time it senses the next value of the function. There are also ways to partially compensate for the relatively long time required by tape operations. For example, the control circuits of a computer may be so designed that the computer proceeds to other operations during the period between

* An added feature desirable in data recording and reduction is that, for moderate recorded pulse densities, data may be recorded or played back at any of several tape speeds with a speed ratio of 10:1 between one system and another being readily obtainable.

initiation of a command involving a tape unit and actual execution of the command. This type of operation calls for some independent control circuits for the tape unit as well as the inclusion of certain logical interlocks to prevent any undesirable interaction. Tape wastage may be reduced, too, by use of particular arrangements of data and programming devices, e.g., by back spacing at the end of a block in preparation for the next.

Though devices are available for entering data onto a magnetic tape directly from a keyboard, this procedure is seldom used, principally because the recording density thus obtainable is much less than that obtainable by other methods. Both paper tape to magnetic tape and punched card to magnetic tape converters have been built that read data from the specified medium and transcribe it in a continuous manner onto magnetic tape. Because of the speed differences in processing these media, buffer storage must be included in these converters. The need for the paper tape to magnetic tape converter also arises because data transmitted over long distances by teletype is available on punched paper tape (teletype tape). The need for punched card to magnetic tape converters arises because of the large amounts of data presently recorded on punched cards and the convenience of recording certain sources of data in this form. There is also a converter which punches cards in punched-card code from the code read from teletype tape.

There is available commercially a Computer Language Translator (produced by the Electronic Engineering Co. of California) that provides at high speed an efficient translation of data from any one of a number of input media (magnetic tape, punched paper tape, punched cards, an analog to digital converter) to the form required by one of a number of output units (magnetic tape, punched paper tape, punched cards, printers, digital plotters, or a digital to analog converter). In addition to format and media conversion, various types of code conversion can also be provided.

A.2. Printers

A.2.1. Character at a Time Printers

The simplest type of printing device usually employed in conjunction with a digital computer is an electric typewriter. The speed of these machines may vary from 6 to 20 characters per second, with an average figure of about 12 for most. As indicated in Chapter 7, an electric typewriter is often used as an on-line input–output device to handle limited amounts of data. A standard electric typewriter may be modified for use as an input device by the addition of switches so placed that movement

of the key lever (produced by depression of a key) not only moves a type bar, but also triggers the corresponding switch. Output signals from the switches may then be used as inputs to a computer. The typewriter may be modified for use as an output device by so locating a set of push rods, each controlled by an electromagnet, that energizing any magnet causes the push rod to activate a corresponding type bar in the same manner as if a key on the typewriter had been depressed. A disadvantage of this type of arrangement is that as many signal lines must be provided as there are different characters to be printed. To reduce this requirement, there is incorporated in some electric typewriters a mechanical decoder which, for each value of a binary-coded input signal, is so actuated (by the energizing of selected coils) that a particular type bar is depressed. The control signals required are those used to indicate that a printing operation is to be started, or that the typewriter is ready to accept another character, or to produce a carriage return and line advance operation. The latter control signal may be obtained: (a) automatically, when the carriage reaches a certain position, (b) by the inclusion of circuits which count the number of characters (and spaces) on a line and produce a signal when this count exceeds some specified number, (c) by having the computer generate, at appropriate places in the output data, a binary code group that causes a carriage return and line advance operation.

A.2.2. Line at a Time Printers

The term "line at a time" is used to distinguish printers which print a row of characters at a time from typewriters which produce only a single character at a time. There may be from about 25 to 120 character positions (columns) per row. A buffer store holds the complete line of characters which are to be printed at a time.

In the mechanical line at a time printers commonly used with punched card equipment, either type bars or print wheels are used. In the former case, there is a separate type bar (with a complete set of printing dies) for each column, and character selection is performed by positioning each bar vertically so that the desired character is opposite the paper. In the other type of printer, a separate print wheel is provided for each column and selection is performed by rotating the wheels till all the desired characters are opposite the paper. The speeds of these machines vary from about 100 lines/min for the type bar printer to about 150 lines/min for the type wheel printer. Format control may be provided either by plugboards or punched paper tape format control programs incorporated within the printer, or by means of a format control program in a computer that controls the printer.

A.2.3. High Speed (on the fly) Printers

The speed of the accounting machine type of line at a time printers is limited by the time required to accelerate the type bars or wheels. A high speed mechanical printer developed subsequently overcomes this limitation by the device of continuously keeping the print wheels rotating. A separate hammer for each wheel strikes the paper against the ribbon and wheel as the desired character passes underneath. Each character on a wheel is located by its relative position from a fixed reference point.

In a printer of this type manufactured by the Potter Instrument Company, there is only one print wheel, and it rotates in a horizontal rather than vertical plane, passing by as many hammers as there are columns per row. Though the mechanical arrangement is different, the operating principle is similar to that of the multiwheel printer. An important difference is that in this case the time at which a particular character arrives in front of a hammer depends not only on the fixed reference point but also on the columnar position of the particular hammer.

Printers of this type, which are usually operated off-line from magnetic tape inputs, are capable of printing from 300 to 900 lines of alphanumeric data per minute.

A.2.4. High Speed Matrix Printers

In matrix printers the characters are formed not by type font but according to the selection of a pattern of dots in a rectangular array, usually with five columnar and seven row positions. The actual printing operation takes place by means of small wires that transfer ink to the paper by striking it. These printers are intrinsically capable of greater speed than the type-bar or print-wheel printers because the character forming elements do not have to be moved into position, and the wires are of low mass. These units may be operated off-line from magnetic tape or punched card inputs and printing speeds of from 500 to 1000 lines per minute are obtainable.

In one type of arrangement, as many 35-wire matrices are provided as there are columns of data to be generated. This allows a complete row of data to be printed simultaneously. In another type of arrangement, only a single row of five wires is provided for each character column, these wires being actuated seven times (as the paper is advanced) to form a single row of characters.

At present, the quality of matrix printing is somewhat inferior to conventional printing, but for comparable printing speeds the registration is better than for on the fly printers.

A.3. Character-Generating Cathode-Ray Tubes with Light Sensitive Recorders

Three basic methods have been used for the generation of characters (numeric or alphanumeric) on the face of cathode-ray tubes. In the Lissajous method, each character is drawn by the simultaneous application of specified voltage waveforms to the horizontal and vertical deflection plates. In the raster scan method, control of the intensity of the cathode-ray beam during each sweep enables characters to be produced in matrix form. Both of these methods are unsatisfactory in regard to the quality of the characters, the resolution obtainable, and the complexity of circuitry and programming required to generate the pieces that combine to produce a single character.

A great improvement was provided by the development of special cathode-ray tubes that form any of a set of specified characters by means of a stenciled mask incorporated within the tube and placed in the path of the beam from the electron gun. Two of the most widely used character generating cathode-ray tubes are the Charactron, produced by the Stromberg-Carlson division of General Dynamics, and the Typotron (see Fig. A.4) produced by Hughes Aircraft Company. Both can generate characters

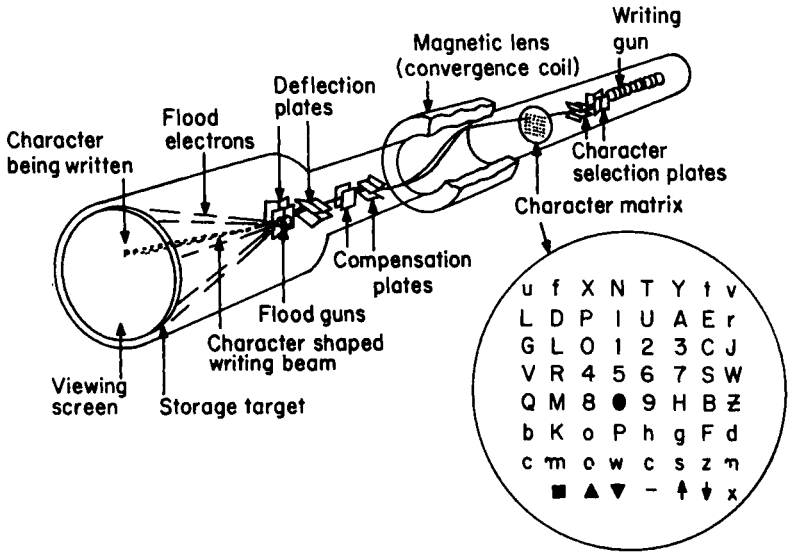


FIG. A.4. Simplified schematic of the Typotron (courtesy of Hughes Aircraft Co.)

at the rate of about 20,000 characters/sec. In each, a metal stencil

containing a full set (64) of characters to be used is built into the tube. To display any particular character the diffused cathode beam is first deflected, by voltages applied to electrostatic deflection plates, to the appropriate position on the stencil. The resulting image is focused with a magnetic lens and directed to a desired position on a long persistence screen by a second deflection system. Since one of the characters cut in the stencil may be a round hole, it is also possible to use these tubes as conventional cathode-ray tubes, and to generate data in graphical form. An added capability of the Typotron is provided by an electron flood gun mounted in the vicinity of the deflection plates that position the shaped beam on the screen. The low velocity electrons from this gun serve to regenerate, by means of secondary emission, the characters formed on the screen by the high velocity shaped beam. Thus, information on the screen can be held indefinitely or erased by disabling the holding function.

Although picture tubes may be viewed directly, such operation does not take advantage of the high potential data output rate of the tube because of the limit imposed by the rate at which the human observer can absorb or record the data. For this reason, when character-generating tubes are used for the generation of large quantities of data at high output rates (either on-line, or off-line with a magnetic tape input) means are provided for automatically recording the data displayed on the tube face. The most obvious method is to record each set of output data on successive frames of photographic film. One electronic printer, produced by the Stromberg-Carlson division of General Dynamics, projects the characters from the tube face onto 35-mm microfilm, and can record at rates up to 15,000 characters/sec. Hard copy can then be produced either by conventional film printing or by a high speed automatic film processor. Another printer produced by the same company uses the Charactron in conjunction with a Xerox printer (produced by Haloid Xerox, Inc.). In the Xerography process, a latent image produced on a specially prepared surface is developed by a dry developing process. The characters from the tube face are projected onto a section of the surface of a drum coated with a photoconductive material such as selenium. The surface is always charged just before exposure to light, and the effect of the exposure is to cause those areas exposed to lose their charge. Then the drum surface is brought in contact with a fine powder developer (electrically charged opposite to the initial charge on the drum surface) which carries a black thermoplastic toner which adheres to the areas of the surface where the light has discharged the selenium. The printing is produced by placing in contact with the drum a roll of paper with a charge of the same sign as the initial charge on the drum. This causes transfer to the paper of a

toner image which is fixed by heating and melting in a fuser. Both of the electronic printers described can be operated either on-line, or off-line with a magnetic tape input. Off-line operation permits better use to be made of computer time especially when each frame contains less than the maximum amount of data.

A.4. Display Devices for Small Quantities of Slowly Changing or Static Information

Methods for displaying individual characters (letters, numbers, or special symbols) can be classified according to the basic methods used for storage, selection, and display of the character.

A.4.1. Moving Indicator Displays

In the moving indicator methods, a particular character is selected and displayed by moving a drum or belt on which characters are imprinted until the character to be displayed is positioned behind a window.

A.4.2. Lamp Switching Displays

One of the earliest types of lamp switching displays provided as many characters and associated lamps for illumination as there were choices of characters. For example, to display any of the numerals 0 through 9, a separate character, lamp, and viewing window were provided for each of the 10 numerals, and selection performed by energizing a particular lamp. In a more compact arrangement, a single, ground-glass viewing screen is provided on which one of a set of characters is projected. Selection is obtained, as before, by energizing the lamp that illuminates the character to be displayed. In another scheme, no individual characters are provided but instead a group of window segments each with its own source of illumination. In this arrangement, a particular character is selected and formed by illuminating, by means of filamentary or neon lamps, a particular pattern of segments. In a similar scheme, instead of window segments and lamps, a group of segments of a fluorescent material is provided, and a fluorescent character is displayed by activation of selected segments.

A.4.3. The Edge-Lit Etched Plastic Display

This is a type of lamp-activated display which is widely used for the display of decimal digits. Each assembly has in a stack as many thin transparent plastic wafers as there are symbols to be displayed, a different

symbol being frost etched into each wafer. A separate filamentary lamp is provided to edge light each wafer. A particular symbol is made visible by activation of its associated lamp. Light transmitted through the plastic is diffused by the etched symbol which thereby becomes visible. (Symbols not selected can also be seen, though faintly.)

A.4.4. Neon Tube Displays

In the simplest type of neon tube used for digital display purposes, an indication of two voltage levels is provided by whether a glow discharge is produced in the tube or not. These tubes are useful where it is only necessary to give a binary indication of the voltage states of bistable elements.

A number of specialized neon display tubes have been developed to provide direct displays of decimal digits, letters of the alphabet, or other symbols. Such tubes hold a number of character-shaped wire elements in an atmosphere of neon. Any particular element is displayed by applying a negative voltage to it with respect to the other elements in excess of the breakdown voltage. This causes the element to act as a cathode and produces a glow discharge about it corresponding to its outline. (Symbols not selected can also be seen, though faintly.)

Another type of neon tube used for display is a combination counting and display device. Upon the application of each trigger pulse to a single input terminal, a glowing gaseous discharge spot is advanced from one element to another in a ring. There are usually 10 elements in the ring, and the tube is provided with a suitable mask which causes the glowing spot to illuminate a different decimal digit at each position. By means of appropriate circuitry, as many of these decade counters can be connected in cascade as desired.

Literature

Angel, A. M. [1957] A very high speed punched paper tape reader, *IRE 1957 Wescon Convention Record,* Pt. 4, 218-227.

Bauer, F. [1959] The Burroughs 220 high speed printer system, *Proc. 1959 Western Joint Computer Conf., San Francisco,* 212-217.

Baybick, S. and Montijo, R. E., Jr. [1957] An RCA high performance tape transport system, *Proc. 1957 Western Joint Computer Conf., Los Angeles,* 52-56.

Barker, R. A. [1960] Techniques of dynamic display, Part I: Cathode ray tubes, *Control Engrg.,* **7**, No. 2, 100-105.

Barker, R. A. [1960] Techniques of dynamic display Part II: Optics at work, *Control Engrg.,* **7**, No. 4, 121-125.

Bower, G. G. [1957] Analog-to-digital converters, *Control Eng.,* **4**, 107-118.

Burkig, J. and Justice, L. E. [1957] Magnacard—magnetic recording studies, *IRE 1957 Wescon Convention Record,* Pt. 4, 214-218.

Buslik, W. S. [1952] IBM magnetic tape reader and recorder, *Proc. Joint AIEE-IRE-ACM Computer Conference,* New York, 86-90.

Carroll, J. M. [1956] Trends in computer input-output devices, *Electronics,* **29,** 142-149.

Cemach, H. P. [1951] *The Elements of Punched Card Accounting,* Pitman, London.

Chow, C. K. [1957] An optimum character recognition system using decision functions, *IRE Trans. El. Comp.,* **EC-6,** 247-254. (Comments on this article by I. Flores appear on p. 180 of the June 1958 issue and by C. K. Chow on p. 230 of the June 1959 issue of the same journal.)

Davies, D. W. [1956] Sorting of data on an electronic computer, *Proc. Inst. Elec. Engrs.,* **103,** Pt. B, Supplement 1, 87-93.

Di Giulio, E. M. [1956] Burroughs G-101 high speed (matrix) printer, *1956 IRE National Convention Record,* Pt. 4, 94-100.

Ferguson, D. E. [1960] Input-output buffering and Fortran, *J. ACM,* **7,** 1-9.

Forgie, J. W. [1957] The Lincoln TX-2 input-output system, *Proc. 1957 Western Joint Computer Conf., Los Angeles,* 156-160.

Hayes, R. M. and Wiener, J. [1957] Magnacard—a new concept in data handling, *1957 IRE Wescon Convention Record,* Pt. 4, 205-209.

Josephs, J. J. [1960] A review of panel-type display devices, *Proc. IRE,* **48,** 1381-1395.

Kilburn, T., Hoffman, G. R., and Wolstenholme, P. [1956] Reading of magnetic records by reluctance modulation, *Proc. Inst. Elec. Engrs.,* **103,** Pt. B, Supplement 2, Convention on Digital Computer Techniques, 333-336.

MacDonald, D. N. [1956] Datafile—a new tool for extensive file storage, *Proc. 1956 Eastern Joint Computer Conf., New York,* 124-127.

Mee, C. D. [1958] Magnetic tape for data recording, *Proc. Inst. Elec. Engrs.,* **105,** Pt. B, 373-382.

Nelson, A. M., Stern, H. M., and Wilson, L. R. [1957] Magnacard—mechanical handling techniques, *I.R.E. 1957 Wescon Convention Record,* Pt. 4, 210-213.

Nordyke, H. W. [1952] Magnetic tape recording techniques and performance, *Proc. Joint AIEE-IRE-ACM Computer Conference, New York,* 90-95.

Partos, P. [1956] Industrial data-reduction and analogue-digital conversion equipment, *Brit. Inst. Radio Engrs.,* **16,** 651-678.

Pike, J. L. and Ainsworth, E. F. [1955] Input-output devices for NBS computers, *NBS Circular 551, Computer Development at the NBS,* 109-118.

Robinson, A. A., McAulay, F., Banks, A. H., and Hogg, D. [1955] A magnetic-tape digital-recording equipment, *Proc. Inst. Elec. Engrs.,* **103,** Pt. B, Supplement 2, Convention on Digital Computer Techniques, 346-353.

Rubinoff, M. and Beter, R. H. [1956] Input and output equipment, *Control Eng.,* **3,** 115-123.

Shaw, R. F. [1960] Techniques and equipment for digital data conversion, *Control Engrg.,* **7,** No. 3, 107-114.

Staff of Cresap, McCormick and Paget [1961] Punched card equipment for medium sized computers, *Control Engrg.,* **8,** No. 10, 108-112.

Staff of Cresap, McCormick and Paget [1961] Punched card equipment for intermediate and large size computers, *Control Engrg.,* **8,** No. 11, 115-121.

Staff of Crtsap, McCormick and Paget [1961] Punched paper tape equipment for medium, intermediate and large computers, *Control Engrg.,* **8,** No. 12, 105-109.

Staff of Cresap, McCormick and Paget [1962] Printing equipment for medium, intermediate, and large size computers, *Control Engrg.,* **9,** No. 1, 91-95.

Staff of Cresap, McCormick and Paget [1962] Magnetic tape equipment for medium size computers, *Control Engrg.*, **9**, No. 2, 124-128.

Staff of Cresap, McCormick and Paget [1962] Magnetic tape equipment for intermediate and large size computers, *Control Engrg.*, **9**, No. 3, 105-109.

Smith, H. M. [1955] The Typotron, a novel character display storage tube, *1955 IRE National Convention Record*, Pt. 4, 129-134.

Susskind, A. K. [1957] *Notes on analog-digital conversion techniques*, Technology Press of M.I.T., and Wiley, New York.

Taylor, J. H. W. [1958] A medium speed transistorized tape reader, *Brit. Com. and Elec.*, **5**, 149-151.

Taylor, K. [1961] Get maximum reliability from digital magnetic tape, *Control Engrg.*, **8**, No. 10, 113-115.

Welsh, H. F. and Lukoff, H. [1952] The Uniservo tape reader and recorder, *Proc. Joint AIEE-IRE-ICM Computer Conference, New York*, 47-53.

Wildanger, E. G. [1957] Tape recording systems for computers, *Automatic Control*, **7**, 36-42.

Wilkes, M. V. and Willis, D. W. [1956] A magnetic-tape auxiliary storage system for the EDSAC, *Proc. Inst. Elec. Engrs.*, **103**, Pt. B, Supplement 2, Convention on Digital Computer Techniques, 337-345.

Author Index

Numbers in italics show the page on which the full reference is listed.

Abbott, H. W., *261*
Abhyanker, S., *97*
Abramowitz, M., *552*
Abramson, N., *551*
Adams, C. W., *519*
Aiken, H., *259*
Ainsworth, E. F., *573*
Akers, S. B., *193*
Albers-Schoenberg, E., *259*
Alexander, M. A., *259*
Alexander, S. N., 122, *188*
Alphonse, G. W., *262*
Alrich, J. C., *258*
Alt, F. L., *552*
Amble, O., *519*
Amundson, N. R., *520*
Anderson, A. G., *189*
Anderson, H. A., *262*
Anderson, J. R., *264*, *265*
Angel, A. M., *572*
Angell, J. B., *189*
Arenberg, D. L., *264*
Arsenault, W. R., *261*
Ashenhurst, R. L., *260*, *370*, *371*
Aspinall, D., *371*
Auerbach, A. A., *447*
Auerbach, I. L., *189*, *191*, *259*, *264*
Avizienis, A., *370*

Bacon, G. C., *257*
Baker, R. H., *189*
Balser, M., *551*, *552*
Banks, A. H., *573*
Barkan, H. E., *256*
Barker, R. A., *572*
Barnes, R. C. M., *189*
Bartik, W. J., *259*
Bartky, W. S., 125
Bashkow, T. R., *189*
Bauer, E. W., *260*
Bauer, F., *572*
Bay, Z., *188*
Baybick, S., *572*
Bazovsky, I., *551*
Beatson, T. J., *97*
Beck, E. R., *193*, *263*
Beck, R. M., *265*
Beckman, F. S., *447*
Begun, S. J., *256*
Bell, P. R., *256*
Benfield, A. E., *264*
Benner, A. H., 521, *551*
Berry, D. L., *263*
Best, R. L., 222, *259*
Beter, R. H., *189*, *573*
Bethel, H. W., *188*
Beveridge, H. N., *264*
Beyer, R. T., *191*
Bigelow, J. H., *447*
Bindon, D. G., *260*
Birkhoff, G., *97*
Bittmann, E. E., *261*, *262*
Bivans, E. W., *258*
Blaauw, G. A., *447*
Blachman, N. M., *259*
Blattner, D. J., *193*
Bloch, R. M., *551*
Blois, M. S., Jr., *262*
Bobeck, A. H., *261*
Bonn, T. H., *191*, *259*
Boole, G., *97*
Booth, A. D., *258*, 334, *370*
Booth, G. W., *189*
Bothwell, T. P., *189*
Bower, G. G., *572*
Bozorth, R. M., *191*, *256*
Bradbury, E. H., *265*
Bradley, R. E., 503, *519*
Bradley, W. E., *189*
Bradspies, S., *192*, *260*
Braun, E. L., 465, 504, *519*
Bremer, J. W., *262*

Brenneman, A. E., *263*
Brenza, J. G., *370*
Brigham, R. C., *370*
Brillouin, L. N., *264*
Brooks, F. P., Jr., *447*
Brower, D. F., *257*
Brown, A. J., *265*
Brown, D. R., *188, 259*
Brown, D. T, *552*
Brown, G. W., *265*
Brown, R. B., *189*
Brown, R. M., *188*
Brown, W. G., *551*
Buck, D. A., *193, 260, 262, 264, 265*
Burkig, J., *572*
Burks, A. W., 289, *447*
Burla, N., *371*
Burns, L. L., *262*
Bush, V., *520*
Buslik, W. S., *573*
Butler, S. A., *193*

Caldwell, S. H., 69, *97, 189, 520*
Calingaert, P., *552*
Campbell, C. M., Jr., *189*
Campbell, D. S., *265*
Campbell, R. V. D., *551*
Carey, W. M., Jr., *191*
Carlson, A. W., *190*
Carne, E. B., 203, *258*
Carroll, J. M., *573*
Carroll, W. N., *189, 370*
Carter, I. P. V., *260*
Carter, W. C., *447*
Cemach, H. P., 559, *573*
Chance, B., *188*
Chao, S. C., *190*
Chapin, D. M., *259*
Chaplin, G. B. B., 130, *190, 191, 263, 265, 370*
Chen, W. H., *98*
Chien, R. T., *259*
Chow, C. K., 554, *573*
Chow, W. F., *193*
Christopherson, W. A., *259*
Chruney, M., *263*
Clapp, L. C., *256*
Clark, D. L., *257*
Clayden, D. O., *258, 264*
Coblenz, A., *190*
Cohen, A. A., *258*
Cohen, M. L., *263*
Constantine, G., Jr., *259*
Conway, A. C., *264*
Cooke-Yarborough, E. H., *189, 190, 257*
Cooper, J. N., *262*
Cooper, R. A., *189*
Copi, I. M., *447*
Couleur, J. F., *370*
Crane, H. D., *191, 192*
Crank, J., *520*
Critchlow, D. L., *193*
Crittenden, E. C., *262*
Crosby, R. L., *191*
Crowe, J. W., *262*
Crowther, T. S., *262*
Curtis, P. C., Jr., *370*

Darr, J. H., *264*
Davidsohn, U. S., *193*
Davies, D. W., *573*
Davies, P. M., *262*
De Barr, A. E., *260*
De Buske, J. J., *261*
Deering, C. S., *520*
DeLano, R. B., Jr., *263*
De Turk, J. E., *188*
Devenny, C. F., *191*
Dickinson, W. E., *258*
Dietrich, W., *262*
Di Giulio, E. M., *573*
Dimsdale, B., *551*
Di Nolfo, R. S., *259*
Disson, S. B., *191*
Dodd, S. H., *263*
Domenico, R. J., *190*
Donan, J. F., *520*

Eadie, D., *257*
Easley, J. W., *190*
Ebers, J. J., *190*
Eckert, J. P., Jr., *262, 264, 447*
Edwards, D. B. G., *262, 263, 371*
Edwards, H. H., *262*
Elbourn, R. D., 122, 188
Elias, P., *551*
Ellis, M., *551*
Elspas, B., *552*

Emslie, A. G., *264*
Epstein, G., *97*
Epstein, H., *264*
Esaki, L., 172, *193*
Estrin, G., *370*

Fagen, M. D., *264*
Fairclough, J. W., *265*
Fan, G. J., *257*
Felker, J. H., *190*
Ferguson, D. E., *573*
Fisch, E. A., *263*
Flehinger, B. J., *551*
Fleisher, H., *190*
Forbes, G. D., 256
Forgie, J. W., *573*
Forrester, J. W., *259*
Foss, E. D., *259*
Fraenkel, A. S., *370*
Frank, M. E., *370*
Frank, W., *260, 371*
Frankel, S. P., *447*
Freiman, C. V., *370*
Friedman, D. C., *264*
Friedman, M. J., *260*
Fruin, R. E., *262*
Fuller, H. W., *257*

Gact, J. K., *256*
Garner, H. L., *188, 370*
Garwin, R. L., *262*
Genna, J. F., 503, *519*
Ghazala, M. J., *97*
Giacoletto, L. J., *190*
Gibbons, J. F., *188*
Gilchrist, B., 296, *370*
Gill, A., 482, *520*
Ginsburg, S., *97*
Gluck, S. E., 111, *188*
Golay, M., *551*
Goldey, J. M., *190*
Goldstine, H. H., 289, 330, *447*, 548, *553*
Goodenough, J. G., *260*
Graham, M., *263*
Gray, H. J., 111, *188*
Greenwald, S., 122, *188*
Griesmer, J. H., *552*
Grimsley, J. D., *264*
Grisamore, N. T., *188*
Guterman, S. S., *191, 192*
Gutwin, O. A., *261*

Haeff, A. V., *263*
Hagelbarger, D. W., *552*
Hageman, D. H. A., *259*
Hammel, D. G., *261*
Hammer, P. C., 356
Hamming, R. W., *551*
Harris, J. R., *190*
Hartel, R. R., 203, 211, *257*
Hartree, D. R., *97, 520, 552*
Harwood, W. J., *192, 264*
Hastings, C., Jr., 356, 503
Haueter, R. C., 122, *188*
Hayes, R. E., *265, 370*
Hayes, R. M., *573*
Haynes, M. K., *259*
Hendrickson, H. C., *370*
Henegar, H. B., *520*
Henle, R. A., *190*
Henney, F. C., III, 390
Herndon, T. O., *262*
Hershberg, P. I., *256*
Heuer, A., *260*
Hildebrand, F. B., *552*
Hines, M. E., *263*
Hoagland, A. S., *258*
Hock, R. E., *188*
Hoffman, G. R., *259, 263, 573*
Hogg, D., *573*
Holden, P., *551*
Hollander, G. H., *258*
Holonyak, N., Jr., *190, 193*
Holt, A. W., *264*
Homan, C. J., *520*
Hong, K., 203, *258*
Householder, A. S., *552*
Howard, R. A., *258*
Howells, G. A., *189*
Huffman, D. A., 90, *98*, 124, *551*
Hughes, E. S., Jr., *258*
Hughes, V. W., *264*
Hunter, L. P., *190, 260*
Huntington, E. V., *98*
Huntington, H. B., *264*
Hurley, R. B., *188*

Husman, P. A., *257*
Hussey, L. W., III, *188*

Jackson, R. C., *370*
Jacobi, G. I., *99*
Jacoby, M., 210, *257*
Janik, J., Jr., *261*
Jaynes, E. T., *256*
Johnson, K. C., *192*, *264*
Johnson, R. F., *189*
Johnson, S. J., *265*
Josephs, J. J., *573*
Justice, L. E., *572*

Kahn, W., *447*
Kaiser, H. R., 160, *192*
Kamm, V. C., *193*
Kane, B., *260*
Karnaugh, M., 64, *98*, *192*, *260*
Kaufman, J., *188*
Kaufman, M. M., *264*
Kazan, B., *262*
Keiper, F. P., *189*
Keister, W., *98*
Keith, W. W., *264*
Kelner, R. C., *257*
Kilburn, T., *258*, *259*, *263*, *371*, *573*
Kilmer, W. L., *552*
King, G. W., *265*
Kiseda, J. R., *260*
Kittel, C., *256*
Kleene, S. C., *98*
Kleinerman, M. H., 521, *551*
Klemperer, H., *263*
Knoll, M., *262*
Knuth, D. E., 393, *447*
Kodis, R. D., *192*
Koelsch, A. C., *263*
Kogbetliantz, E. G., *370*
Koons, F., *371*
Kudlich, R. A., *190*

Laemmel, A. E., *552*
Läfgren, L., *551*
Lanczos, C., *552*
Landauer, R., *551*
Lawless, W. J., Jr., *447*
Leary, F., *193*
Leck, G. W., *262*
Ledermann, W., *98*
Ledley, R. S., *371*
Lee, C. Y., 69, *98*
Lehman, M., *260*, *371*
Leichner, G. H., *190*
Leiner, A. L., *447*
Leondes, C. T., III, *188*
Lesk, I. A., *193*
Lewin, M. H., *193*
Lewis, I. A. D., *98*
Lewis, W. D., *193*
Liddell, D. W., *552*
Lindquist, A. B., *262*
Linvill, J. G., *188*, *190*
Lipow, M., *552*
Litting, C. N. W., *263*
Lloyd, D. K., *552*
Lloyd, S. P., *552*
Lo, A. W., 163, *192*, *193*, *261*
Loberman, H., *447*
Loev, D., 155, *192*
Logue, J. C., *263*
Looney, D. H., *261*
Lourie, N., *447*
Lovell, C. A., *265*
Lubkin, S., *257*, *371*
Lucal, H. M., 543, *552*
Lukoff, H., *263*, *574*

McAulay, F., *573*
McCarthy, J. A., *263*
McCluskey, E. J., Jr., *98*
McCulloch, W., *98*
MacDonald, D. N., *573*
MacDonald, J. E., *553*
McDuffie, G. E., Jr., *264*
McGuigan, J. H., *258*
MacLane, S., *97*
MacNichols, E. F., Jr., *256*, *262*
MacSorly, O. L., *371*
Maginniss, F. J., *520*
Mapleton, R. A., *264*
Marcovitz, M. W., *190*
Marcus, M. P., *260*, *553*
Marolf, R., *193*
Mason, S. J., *188*
Maudsley, B. G., *258*
May, J. E., *264*
May, M., *258*
Meagher, R. E., *189*
Mealy, G. H., 90, *98*

Mebs, R. W., *264*
Mee, C. D., *573*
Meggitt, J. E., *553*
Meier, D. A., *262*
Melas, C. M., *553*
Mendelson, M. J., *520*
Menyuk, N., *260*
Meredith, B., 521, *551*
Merrill, L. L., *257*
Merry, I. W., *258*
Merwin, R. E., *260*
Merz, D. M., 465, *520*
Metropolis, N., *370, 371*
Michel, J. G. L., *520*
Michel, P. C., *257*
Miehle, W., *192*
Miller, G. H., *191*
Miller, G. L., *263*
Miller, G. P., *258*
Miller, S. L., *190*
Millership, R., *265*
Milne, W. E., *552*
Minnick, R. C., *192, 260, 371*
Minsky, M., 16
Mitchell, J. M., 503, *520*
Miyata, J. J., 203, 211, *257*
Moll, J. L., *190*
Montijo, R. E., Jr., *572*
Moore, E. F., *98*
Morgan, L. P., *191*
Morgan, W. C., *261*
Mueller, R. K., *99*
Muerle, J. L., *190*
Muller, D. E., *98*, 125

Nash, J. P., *189*
Naylor, R., *265*
Neff, G. W., *193*
Nelson, A. M., *573*
Nelson, D. J., *520*
Nelson, J. C., 125, *190*
Nelson, R. J., *98*
Nemanic, D. J., *520*
Netherwood, D. B., *98*
Newhouse, V. L., *192*, 227, *260, 262*
Newman, E. A., *264*
Nordyke, H. W., *573*
Notz, W. A., *447*
Noyes, T., 214, *258*

O'Connor, D. G., *259*
Olsson, J. K. A., 227, *260*
Osborne, C. F., *258*
Owen, P. L., 503, *520*
Owens, A. R., *265, 370*
Owens, H. L., *190*

Page, L. J., *258*
Paivinen, J., *189*, 192
Palevsky, M., *520*
Papian, W. N., *260*
Papoulis, A., *261*
Partos, P., *573*
Partridge, M. F., 503, *520*
Partridge, R. S., *259*
Pate, H. R., *263*
Paull, M. C., *98*
Pearson, R. T., *259*
Pedelty, M. J., *552*
Peil, W., *193*
Pennell, E. S., *265*
Perkins, R. L., *259*
Petersen, H. E., *260*
Petschauer, R. J., *262*
Pietenpol, W. J., *191*
Pike, J. L., *573*
Pitts, W., *98*
Pohm, A. V., *262*
Pollard, B. W., *259*
Pomerene, J. H., 296, *370*
Pope, D. A., *371*
Post, G., *519*
Potter, J. T., *257*
Pressman, A. I., *191*
Pressman, R., *189*
Preston, K., Jr., *261*
Proebster, W. E., *262*
Prom, G. J., *191*
Prutton, M., *265*
Prywes, N. S., *192*
Pucel, R. A., *193*
Puckle, O. S., *191*
Pullen, K. A., Jr., *188*
Pulvari, C. F., *265*

Quine, W. V., 61, *98, 99*

Rabinow, J., 214, *259*
Rabinowitz, P., *371*
Raffel, J. I., *192, 260, 262*

Rajchman, J. A., 163, *192, 257, 260, 261, 264*
Ralston, A., 536, *552*
Ramey, R. A., *192*
Rapp, A. K., *191*
Ratz, A. G., *189*
Reach, R., *447*
Reed, I. S., *520, 552*
Reichenbach, H., *99*
Reitwiesner, G. W., *371*
Renwick, W., *447*
Rhoades, W. T., *193*
Rhoderick, E. H., *262*
Rhodes, W. H., Jr., *370*
Ridenour, L. N., *265*
Ritchie, A. E., *98*
Ritchie, D. K., *189*
Robbins, R. C., *265*
Robertson, J. E., *189, 371*
Robinson, A. A., *260, 371, 573*
Rochester, N., *188*
Roger, G. H., *191*
Rogers, T. F., *265*
Rosenberg, M., *260*
Rosenfeld, J. L., *192*
Rosenheim, D. E., *189*
Rosin, R. F., *262*
Roth, J. P., *99*
Rothbart, A., *265*
Rowe, W. D., *191*
Rowley, G. C., *520*
Rubens, S., *259, 262*
Rubinoff, M., *99*, 111, *188, 189, 573*
Ruhman, S., *192*, 503, *520*
Rumble, W. G., *261*
Ryan, R. D., *265*

Sack, H. S., *191*
Sacks, G. E., *552*
Saltman, R. G., *371*
Salzberg, B., 111, *188*
Samusenko, A. G., *193*
Sands, E. A., *192*
Savitt, D. A., *263*
Scarrott, G. G., *192, 264, 265*
Schaffer, R. R., *258*
Schlaeppi, H. P., *260*
Schmidlin, F. W., *262*
Schrimpf, H., *447*
Schwartz, L. S., *552*
Schy, S. T., *370*
Scobey, J. E., 111, *188*
Scrivener, J. H., *552*
Seader, L. D., 213, 214, *258, 259*
Seeber, R. R., *262*
Seelbach, W. C., *260*
Seif, E., *190*
Serrell, R., *99*
Shannon, C. E., *99*, 516, *519, 552*
Shapiro, H., *264*
Shaw, R. F., *264, 371, 447, 573*
Shea, R. F., *191*
Sheffer, H. M., *99*
Shell, D. L., *371*
Shelman, H. B., *520*
Shemeta, E. A., *520*
Sheppard, C. B., *264*
Sherertz, P. C., *189*
Shevel, W. L., Jr., 235, *261*
Shifrin, G. A., *258*
Shockley, W., *191*
Shoenberg, D., *193, 257*
Short, R. A., *552*
Sidnam, R. D., *261*
Siforov, V. I., *552*
Silverman, R. A., *552*
Simkins, Q. W., *191, 261*
Simons, B. H., *261*
Sims, R. C., *193*
Sizer, T. R. H., 503, *520*
Sklansky, J., *371*
Slade, A. E., *263*
Slepian, D., 69, *99, 552*
Slutz, R. J., *264*
Smallman, C. R., *263*
Smith, D. O., *262*
Smith, H. M., *574*
Smith, J. L., *371, 447*
Smoliar, G., *263*
Spaeth, D. A., *265*
Sparks, M., *191*
Spinrad, R., *263*
Spitzbart, A., *371*
Sprague, R. E., *520*
Stabler, E. P., *193*
Staehler, R., *99*
Stegun, I. A., *552*
Stein, M. L., *371*

Stephen, J. H., *189, 257*
Stephenson, W. L., *191*
Stern, H. M., *573*
Sterzer, F., *193*
Straley, R., *260*
Stram, O., *264*
Stringer, J. B., *447*
Stuart-Williams, R., *260*
Suran, J. J., *261*
Susskind, A. K., 543, *552*, 553, *574*
Swanson, J. A., *257, 552*

Tanenbaum, M., *190*
Taylor, J. H. W., *574*
Taylor, K., *574*
Teal, G. K., *191*
Teig, M., *260*
Thompson, L. G., *191*
Thompson, P. M., *263*
Thompson, Sir William (Lord Kelvin), 451, *520*
Thorensen, R., *261*
Tierney, J., *520, 551*
Tillman, R. M., *261*
Tkach, G., *260*
Tocher, K. D., *371*
Traub, J. F., *371*
Trischka, J. W., *191*
Turing, A. M., *99*
Turner, R. J., *191*
Turnquist, R. D., *262*

Ulrich, W., 111, *188*
Unger, S. H., *98*, 124
Urbano, R. H., *99*

Van der Poel, W. L., *447*
Van Sant, O. J., *192*
Veitch, E. W., 63, *99*
Vinal, A. W., *261*
Vogelsong, J. H., *191*
Vogl, N. G., Jr., *260*
von Laue, M., *257*
von Neumann, J., *193*, 289, 330, *447, 548, 552, 553*
Vorndran, J. W., 160, *192*

Wadey, W. G., *371*
Wallace, R. L., Jr., *191*, 203, *258*
Walsh, J. L., *191*
Wang, A., *192, 259*
Wang, Hao, *99*
Wanlass, C. L., *261*
Wanlass, S. D., *261*
Warnock, J., *191*
Warren, C. S., *261*
Washburn, S. H., 69, *98, 99*
Wasserman, R., *551*
Weinberg, G. M., *551, 552*
Weinberger, A., *371, 447*
Weiner, J. R., *447*
Weiss, E., *520*
Weiss, G. H., 521, *551*
Weisz, R. S., *260*
Welsh, H. F., *574*
West, J. C., *258*
Wheeler, D. J., *447*
White, E. A., *191*
White, W. A., 111, *188*
Whiteside, A. E., *263*
Widrow, B., 227, *261*
Wiener, J., *573*
Wier, J. M., *261*
Wildanger, E. G., *574*
Wilkes, M. V., *447, 574*
Willey, J. R., 552
Williams, F. C., 130, *191, 258, 259, 263, 371*
Williams, R. C., *265*
Willis, D. W., *574*
Wilson, J. B., *371*
Wilson, L. D., *447*
Wilson, L. R., *573*
Winger, W. D., *370*
Witt, R. P., 122, *188, 264*
Wolf, P., *262*
Wolfendale, E., *191*
Wolstenholme, P., *259, 573*
Wong, S. Y., *191, 263*, 296
Woo, W. D., *192, 259*
Wright, M. A., *264*
Wylen, J., *192*

Yokelson, B. J., 111, *188*
Younker, E. L., *260*
Yourke, H. S., *191*
Youtz, P., *263*

Zimbel, N., *189*
Zimmerman, H. J., *188*

Subject Index

Abacus, 3, 4
  positional notation, 4–5
    definition of, 4
    importance of, 4, 5
Absolute computer, 20
Absolute value of a function
  generation in a DDA, 485–486
Access time
  general remarks, 195
  in a DDA, 475
  in a magnetic tape store, 562
  in multidisk stores, 216
  of different storage areas in a GP computer, 428
Accumulator
  block diagram of a shifting accumulator, 321
  functions of, 24
  in a digital integrator, 463
  in a simple GP computer with a static store, 417
  parallel accumulators with automatic carry propagation, 302–307
    accumulators with no carry command inputs, 305–307
    accumulators with separate carry storage, 304–305
    typical stage of an accumulator with a 0 and 1 carry propagation circuit, 307
    typical stage of a parallel accumulator, 303
    used as a digital servo in a DDA, 494
Acoustic delay line storage
  block diagram, 245
Acquisition period
  definition of, in asynchronous control, 389
Active state of a computer
  conditions that define, 431–432
  definition of, 425
Adder (see Addition)
  adder for use with recirculating type of main store, 279
  half-adders, 278–279
  logical equations for sum and carry, 278, 279, 280, 281
  parallel adder with bistable counter storage and an anticipatory carry chain, 294
  parallel adder with flip-flop storage and an anticipatory carry chain, 294
  serial adder utilizing a delay line for augend-sum storage, 284
  serial adder utilizing a shift register for augend-sum storage, 285
  serial excess-three code decimal adder, 300–302
  servo adder, 464, 477
  successive states of flip-flops in a parallel adder, 290
  three adders utilizing a carry flip-flop, 282
Addition (see Binary addition, 275–297 and Decimal addition, 297–307)
  of numbers in floating point notation, 361
  of rates by a digital servo in a DDA, 488
Address
  block address, 403
  decoding in a simple GP computer with a static store, 416–417
  field, functions of, 24, 25
  register (see Control register)
  return address (in priority interrupt control), 412
  symbolic address, 30
  use of literal symbols for, 29, 30
Address register (see Instruction register)
Addressing channel
  in a serial DDA, 475–476
  in a special purpose DDA, 502

Addressing system
in a GP computer
different addressing systems, 393
effect of number of addresses on the control unit, 393–394
in a magnetic drum or disk store, 428
Admissible values
in error detecting and correcting codes, 525, 530
Air bearing
for a magnetic head, 212
in IBM RAMAC disk memory, 216
Algebraic and trigonometric function generation, 349–358
derivation of a general iterative formula, 349–350
iterative formulas for the reciprocal, 350
Algorithms and logical designs for mechanization of basic arithmetic operations, 266–349
binary addition, 275–297
counting, 266–273
decimal addition, 297–307
division, 340–349
multiplication, 316–340
representation of negative numbers and the subtraction process, 308–316
Ambiguity
in shaft position encoding, 541–542
Amplifiers, in dynamic magnetic recording read circuits
DC, 205
wide band, 209
Analog computer
applications, 504
description of analog machines, 448–454
maximum frequency of sine wave generation, 503
Analogous systems, 448
Analytic checks, 534–537
Analytic function generation
in a differential analyzer, 454–462, 477, 484
Analytical engine (of Charles Babbage), 11
Apertured ferrite plate, 230–231
Applicability of the DDA, 502–504
Applications of computers, 15, 16, 17
business, industrial, and military, 16
Arithmetic
instructions, examples of, 374–375
operations in a computer, 266–371
algebraic and trigonometric function generation, 349–358
algorithms and logical designs for mechanization of basic arithmetic operations, 266–349
binary, decimal conversion, 364–371
scaling, 358–364
serial and parallel operations in a computer, 14
Arithmetic computer (see GP computer), 16–17, 19, 20, 22–32, 372–447, 503
Arithmetic processes (in a digital computer), 14
Assignment of integrator numbers in a serial DDA, 479–480
Associative law (multiplication)
effect of round-off error, 549
Asynchronous (revertive) control
comparison of synchronous and asynchronous control, 389–391
nature of, 382, 386, 388
schematic of asynchronous control for a single-address GP computer, 388
Asynchronous operation of a computer
effect on control unit, 385–391
Asynchronous systems
completion signals in, 123
decoupled circuits in, 124
degrees of asynchronous operation, 123
hazards in, 123
races, 124, 125
speed independence, 123–127
University of Illinois design techniques, 126, 127
Asynchronous use of a computer
priority interrupt control, 411
Automata
mechanical, 9, 10
Automatic computation
fundamental nature of, 20–23

Automatic switching operations
by a decision unit in a DDA, 487–488
Auxiliary circuits
signal level conversion, 188
trigger circuits, 186, 187
Auxiliary storage
magnetic tape, 565–566

B-box (see Index registers), 394–399
Bias (of round-off error), 545–547
Binary addition, 275–297
parallel adder with carry flip-flops, 291–293
parallel adder with fast carry, 295–297
parallel adders with full length anticipatory carry chains, 294–295
parallel binary adders, 286–291
serial accumulator, 286
serial binary adders, 275–283
addition by counting, 275–278
by analog summation of increments, 275–278
by digital summation of increments, 275–276
addition by use of logical operations, 278–283
use of a delay line for augend-sum storage, 284
use of a shift register for augend-sum storage, 284–285
Binary-coded decimals
definition of, 297
excess-three code, 299–300
second and fifth multiples of the straight binary-coded decimals, 337
straight binary code, 297, 298
weighted four-bit codes, 297
Binary codes
self complementing codes, 314
Binary communication (see Binary transfer), 13, 464, 468
Binary counter
nonsaturating DCTL circuit, 141
successive states in a three-stage unit, 267
diode gating networks for *R-S*, and *T* flip-flops, 268
Binary data
keyboard entry of, 556
Binary, decimal conversion, 364–371
binary to decimal, 368–369
comparison of binary and binary-coded decimal representation, 369–370
decimal to binary, 364–368
Binary division, 340–347
nonrestoring method, 343–347
derivation of division algorithm, 343–345
examples, 345–346
trial and error (restoring methods), 340–343
determination of correct orders of the dividend from which the divisor is to be subtracted initially, 340–341
disposition of the remainder, 342–343
example, 342
modifications required for numbers in a two's complement form, 343
procedure after a negative result, 341
Binary elements
reasons for use in electronic digital computers, 6
Binary multiplication, 318–335
asynchronous multiplier, 327–328
by parallel accumulation, 319–321
by serial accumulation, 321–323
multipliers for operating on negative numbers in two's complement form, 330–335
scheme independent of operand signs, 334–335
two schemes requiring special corrective action dependent on signs of the operands, 330–333
serial-parallel multipliers, 323–327
simultaneous multiplier, 328–330
Binary representation
advantages in a digital computer, 551
definition of bit, 5
Binary to decimal conversion, 368–369
Binary transfer, 13
in a DDA, 464, 468

Bit period, definition of, 430
Block address, in an external store, 403
Block diagrams
  uses of, 443–444
Block transfer instruction, with external storage devices, 403
Blocked state, of a computer, 409, 439
Blocking oscillator
  biased blocking oscillator, 187
Blocks of data
  on magnetic tape, 562
Boolean algebra
  associative law, 41
  commutative law, 41
  compared with number algebra, 38, 39
  conjunctive normal form, 43
  De Morgan's theorem, 43
  disjunctive normal form, 43
  distributive law, 41
  elemental form, 43
  logical functions of, 39, 40
  minimal set of independent operators, 49
  operators, 46
    exclusive or, 49
    NOR, 49
    NOT—AND (Scheffer stroke), 49
  primitive operators (AND, OR, NOT), 49
    derivation from universal switching functions, 49, 50
    sum (modulo 2), 50
  principle of dualization, 42
  product of sums, 43
  sum of products, 43
  tautologies, 43–45
  universal switching functions, 49
Boolean algebraic description
  advantages of, for a digital computer, 92–95
  as an aid to producing logic, usage and wiring tabulations, 444–445
  compared with block diagram descriptions, 444
Boolean algebraic equations, rearrangement and simplification, 52–69
  chart methods, 59–63
    chart for three variables, 59
    Harvard method, 59–61
    Quine simplification, 61–63
    simplified form of chart, 60
  converting a sum of products to a product of sums or vice versa, 54–59
    interchanging AND and OR functions by redefining values of the input variables, 58
    superfluous terms produced by double conversion, 57, 58
  expansion of terms in equation, 53, 54
  introduction of redundant terms, 54
  map methods, 63–69
    Karnaugh maps, 64–68
    selection of groups, 65–68
    Veitch maps, 64
  trial and error methods, 53, 54
Breakpoint
  instruction, 375, 426
  switches, 426
Buffer
  input, output buffers, 402
  for card punches and readers, 559
Buffer register
  location of, 426
Buffer storage
  shift registers, 180–186
Buffer zone
  between adjacent words in magnetic surface recording, 429

Carries
  average number of successive carries in addition of numbers, 289
  circuit to propagate 0 and 1 carries in a parallel adder, 295
  logic for sensing completion of (in parallel adders), 291, 293, 296, 307
  logical equation for, 278
  simple carry circuit in a parallel adder, 295
  transients in propagation of, 287
  truth table for carry generation, 296
  use of flip-flops to reduce gating elements, 291
Casting out 9's, checking by, 531–532

Cathode follower circuit, 118, 119
Cathode-ray tube
  display devices, 569–571
  storage, 239–243
Central store (see Store, main)
  in a DDA, 464
Channels
  in input-output media, 555
Character generating cathode-ray tube, 569–570
  charactron, 569–570
  typotron, 569–570
Character recognition, 554
Characters
  in input-output media, 555
Charactron, 569–570
Chebyshev polynomials
  function generation by, 514
  generation of trigonometric functions by, 356
Check bits
  for single error correction, 527
Checking
  residue checks, 531–532
  results of computations in a DDA, 512–513
    running a problem in reverse, 513
    varying scale factors on successive runs, 512–513
  spot check, 512
  substitution check, 512
  sum check of a memory, 538
Checking (automatic methods), 524–540
  (see also: Error detection and correction)
  built-in checks, 524–533
    arithmetic checks, 530–533
    storage and transfer checks, 524-530
  programmed checks, 533–538
    analytic checks, 534–537
    data transfer checks, 537–538
    diagnostic programs, 538–539
    roll-back programs, 539–540
    sequencing checks, 537
    test programs, 538
    trace programs, 539
Checking function
  in an error correction scheme, 528
Circuit descriptions of switching and storage elements
  introduction to, 100–101
Circuit logic
  general remarks, 101, 102
  magnetic core systems, 152–166
    computing elements for gigacycle operation, 171–173
    gating circuits, 158–160
    general remarks, 152–155
    multi-input core gate, transistor flip-flop systems, 160–162
    superconductive switching elements, 166–171
    transfer loops for coupling of magnetic circuits, 155–158
  NOR circuits, 137
  SBT NOR and Sheffer stroke logic, 139
  transistor systems, 127–152
    junction transistor circuits, 133–152
      DCTL (direct coupled transistor logic) circuits, 139, 143
      emitter followers, 135, 136
      gating circuits, examples of, 136, 137
      general remarks, 133, 134
      inverters, 134–135
      NOR logic circuits, 137, 138
      TRL (transistor-resistor logic) circuits, 138
    point-contact transistor circuits, 127–133
      semiconductor diode, point contact transistor system of circuit logic, 133
      single transistor flip-flops, 128–130
      two-transistor flip-flops, 130–132
  vacuum tube systems, 116–127
    AC system, 120–123
    diode gate, flip-flop system, 119
    general remarks, 116–119
    pentode gate system, 119–120
    synchronous DC system, 123–127
Circuits, auxiliary, 186, 188
Circular number system
  in a DDA, 489
Circulating loops, in a delay line store, 428

Circulating registers
arithmetic and control functions in a GP computer, 431
in mechanization of integrators for a serial DDA, 473
Clipping
by a decision unit in a DDA, 486–487
Clock channel (see timing channels), 429–430
Clock pulse generation
in synchronous control of a GP computer, 386
Clock pulse generators (see Timing signals), 95, 96
multiphase clocks, 95
in AC system of circuit logic, 122
Codes
error minimizing, 541–543
for error detection and correction, 525–530
basic codes for error detection, 525
Coding
nature of, 34
Coefficient of a number in floating-point notation, 360
Coercive force, 217
Coercivity (temperature coefficient of) for ferrite cores, 229
Coincident current magnetic core memory, 218–223
core storage cycle, 222–223
disturb signals, 221–222
drive system for, 223–226
schematic of a coincident current array, 218
selection of a word at a time, 220
selection ratios, 219
Coincidence gates
in a DDA, 475
Coincident current selection
disadvantages of, 219–220
Collection networks (see Switching matrix) 178–180
Column check (see Parity checking), 526
Combinational circuits
AND (Boolean multiplication), 40, 42
NOT (complementation or negation), 40
OR (inclusive or), 39, 42
reduction of the level or number of elements by use of storage, 113, 114
Commands
acquisition (look-up), 418–419
definition of, 383
elementary, 404
execution, 418–419
instruction look-up, 383
preparatory, 405
Common states
of an instruction register, 422
Communication
limiting in a special purpose DDA, 502
Commutative law of multiplication, 549
Comparators
carryless determination of equality by a circuit with 1 and 0 carry propagation chains, 297
logical equations of, 315–316
use in division, 341
Complementary current switches, 148, 150
Complementary numbers, 310–315
nine's complement, 313–315
one's complement, 311–312
self complementing codes, 314
two's complement, 310–311
Computation
by a stored program computer
example of, 23, 24
important characteristics of, 27
of higher order roots, 355–356
of trigonometric functions, 356–358
Computational checks, 512–513
Computer design
comparison criteria, 445–447
general remarks on representative design criteria, 374
Computer synthesis problem
general remarks on subdivision of, 96, 97
Computers
mechanical, 9
Computing aids
evolution of, 9–11
Conditional stop instruction
(see Break-point instruction), 375, 426

Conditional transfer instructions
effect on design of control unit, 383
examples of, 375
Conditions
for generating functions and solving ordinary differential equations by means of a differential analyzer, 516–519
Constant of proportionality in a digital integrator, 464
Constant frequency recording
in a magnetic disk store, 216
Control
circuits (main), 382
computer (programmed error detection and correction in), 533
console (of a computer), 400–401
counter (in a GP computer), 379
register (in a simple GP computer), 416–417
register (in a GP computer), 379–382
register (in multi-address computers), 393–394
Control register (see Control, register, 379–382, also 393–394, 416–417, 419–421)
Control unit, 378–412
criteria affecting minimum number of active storage elements required, 382–383
effect of
inclusion of special control features, 394–399
integration of input-output equipment, 399–403
microprogramming, 404–410
number of addresses in an instruction, 393–394
numerical representation in the arithmetic unit, 390–392
program-interrupt control, 410–412
serial or parallel operation, 384–385
synchronous or asynchronous operation, 385–390
for a magnetic tape store, 403
introductory remarks, 378–384
microcontrol unit, 406
modifications of for use of index registers, 395
of a serial DDA, 473
schematic of a microprogrammed unit, 404
Conversion, between
binary and decimal codes (see Binary, decimal conversion), 364–371
dynamic and static storage, 178–180
reflected and normal binary codes, 543
serial and parallel representation, 178–186
Conversion, format
on magnetic tape, 566, 567
Converters
paper tape to magnetic tape, 566, 567
punched card to magnetic tape, 560, 561, 566, 567
Correction, automatic
of a single error, 527–530
CORSAIR computer, 503
Counter
bistable circuits, 268–273
cascading circuits to form a multi-stage counter, 268
count-up and count-down logic, 84
dynamic binary counters, 271
multi-bit delay line counter, 271–273
multi-stage counters, 269–270
with anticipatory carry, 269–270
repeat, 399
Σdy (in a DDA), 463
Counting, 266–273
in a serial binary adder, 275–278
with set-reset flip-flops, 266–268
Critical magnetic field
curves for superconductors, 167
in superconductive thin film store, 236
Critical temperature, for a superconductor, 166
Cryotron, 167–171
crossed-film cryotron, 168–170
wire-wound, 168, 170
Curie temperature (of barium titanate), 253
Current sharing in a sequential network, 111
Cycle time, factors affecting in a random access memory, 223
Cyclic codes, 542–543

Data entry devices, 400
Data preparation, off line
advantages of, 400
Data preparation machines, 400
DDA (see Digital differential analyzer), 19, 20, 448–520
Decimal
addition, 297–307
parallel decimal adders, 302
serial decimal adders, 297–302
division, 347–349
multiplication, 335–340
a serial-parallel multiplier, 338–339
by halving the multiplier and doubling the multiplicand, 339–340
by repeated addition, 336–338
parallel adders, 302
parallel accumulators with automatic carry propagation, 302–307
serial adders, 297–302
Decimal, binary coded representation of (see Binary coded decimals)
Decimal, binary conversion, 364–371 (see Binary, decimal conversion)
Decimal to binary conversion, 364–368
examples using binary-coded decimal notation, 366–368
examples using binary notation, 365–366
examples using decimal notation, 364–365
Decision unit (in a DDA), 484–488
as a digital servo, 494
examples of use, 485–488
automatic switching, 487–488
nonanalytic function generation, 485–487
general uses, 484
type 1, 484–485
type 2, 485
Decoder (see Switching matrix)
Delay, in signal transmission, 125
Delay line storage, dynamic, 243–251
acoustic delay lines, 244–248
fuzed quartz delay lines, 247–248
mercury delay lines, 246–247
electrical delay lines, 244
general remarks, 243–244
magnetostrictive delay lines, 248–251
Delta noise, in a
coincident current core memory, 222
superconductive memory, 237
thin film memory, 235
Difference equations
to generate analytic functions in a GP machine, 514
Differencing test, 535–536
Differential amplifier, 148
Differential analyzer
conditions for generating functions and solving ordinary differential equations, 516–519
digital, 448–520
electronic analog, 452
auxiliary elements, 453
function generation, 454–462
idealized elements, 448–449
mechanical, 449–450, 451–452
Differential equations
advantages of machine solution, 454
general remarks, 448
mapping of, in a DDA, 477
solution of, by integrators and adders, 450–452
Differential gear, in a mechanical differential analyzer, 450
Digital differential analyzer (DDA), 448–520
addressing channels, 475
analysis of an integrator network for multiplication, 494–498
Appendix: conditions for generating functions and solving ordinary differential equations, 516–519
applicability of the DDA, 502–504
arithmetic and control operations in a serial DDA, 473
Bendix D-12, 484
central store in a serial DDA, 475
checking results of computations, 512–513
compared to a GP machine, 473, 475
CORSAIR, 503
decision units, 484–488
digital integrators, 462–472
digital servos, 488–494
distinguishing features, 452–453
functional structure of, 472–476

general capabilities, 452–453, 476–477, 499
general description of uses, 19, 20
generation of functions, 454–462
information flow in a serial DDA, 473–477
interconnection of elements
by an addressing channel, 475–476, 477
by a plugboard, 503
introductory remarks, 448–454
limiting communication in a special purpose DDA, 502
mapping, 477–480
mechanization of integrators in a serial DDA, 472–473
more complex (multi-register) operational units, 498–502
normalization of equations, 482–483
operation of a serial DDA, 474
organizational block diagram, 474
output multipliers, 483–484
preparation of problems for solution, 476–484
scaling, 480–482
simulating a DDA with a GP machine, 513–517
sources of error, 504–512
SPEDAC, 503
trade-off between precision and solution time, 482
TRICE, 503
variable increment machines, 465
Digital integrators, 462–472
basic design, 463
variations, 464
Digital servo (see Servo, digital), 488–494
Diode
microwave diode, 172
table of number required for different forms of a translational network, 177
tunnel diode
as a switching element, 172–173
in a memory, 238–239
Diode breakdown
silicon junction diodes
in DCTL circuits, 140
in NOR circuits, 137, 138
used to prevent saturation in a circuit, 140, 141
voltage-current, characteristics of, 141
Diode capacitor storage, 251–253
schematic of storage array, transformer AND gate selection matrix, 252
Diode recovery time, 111
Disks
for shaft position encoding, 541–542
magnetic (see Magnetic disk store)
Display devices (for individual characters), 571–572
etched plastic, 571–572
lamp switching, 571
moving indicators, 571
neon tube, 572
Distribution and collection networks (see Switching matrix), 178–180
Distributive law (multiplication)
effect of round-off error, 549
Disturb signals
in coincident current magnetic core memories, 221–222
Disturbed states (see, also, Disturb signals and partial selection) in a magnetic core, 221–222
Division
automatic test of whether quotient exceeds one, 531
binary, 340–347
decimal, 347–349
of numbers in floating-point notation, 360–361
Domain
in a ferromagnetic material, 226
rotation, 226, 227, 234
Dot notation, for magnetic core gates,154
Double precision
arithmetic, 550–551
numbers, 544–545
Drive systems for magnetic core memories, 223–226
Dynamic delay line storage (see Delay line storage, dynamic), 243–251
Dynamic magnetic storage (see Magnetic surface storage, 199–216 and Magnetic drum and disk store, 211–215, 473, 474), 197, 198

Dynamic storage
general description of, 197
magnetic disk, 214–216
magnetic drum, 211–213
synchronous and asynchronous types and selection schemes, 197–198
two principal types, 197
dz generation
in a DDA with binary transfer, 464
in a DDA with ternary transfer, 464
dz store (in a DDA), 464

Effective instruction, 395
Electronic digital computers
primary basis of utility, 22, 23
Electron storage (in an n-p-n transistor), 134
Emitter follower junction transistor circuits, 135, 136
Encoder (see Switching matrix)
Encoder, shaft position, 541–542
ambiguity in reading, 541–542
nonambiguous reading, 543
End around borrow
in operation on nine's complement, 313
in operation on one's complement, 312
ENIAC computer, 10
Equality, test for (see Comparators)
Equations (see Boolean algebraic equations, Flip-flop input equations, Logical design, Timing signals)
Erasability, 195
Error
in rectangular integrations, 465
sequencing operations to reduce round-off error, 548–551
sources of (in a DDA), 470, 504–512
affect of assignment of numbers to integrators, 506
general remarks, 504
in digital servos, 511
in generation of a higher order derivative, 510–511
in generation of specific functions, 508–509
in mathematical statement of problem and way it is programmed, 505–506
in schematic representation of mapping, 511–512
phase error, 507–508
round-off error, 470, 505
scaling of the problem, 506–507
start-up error, 508
truncation error, 505
truncation, 544
Error, detection and correction, 521–551
error minimizing codes, 541–543
general remarks, 521–523
round-off errors, 543–550
techniques for detecting and locating sources, 523–541
built-in checks, 524–533
data transfer checks, 537–538
diagnostic programs, 538–539
differencing (smoothness) test, 535–536
extrapolation checks, 536
functional relationships, 534–535
inverse checks, 535
multiple computation, 534
multiple error detection, 526–527
programmed checks, 533–540
residue checks, 531–532
roll-back programs, 539–540
sequence checks (built-in), 532
single error correction scheme, 527–530
single error detection, 526
sources of error, 523
test programs, 538
trace programs, 539
Excess-three code
description, 299–300
parallel decimal adder, 302
serial decimal adder, 300–302
Execution cycle, in a GP computer, 379
Exponent index (of a floating-point number)
definition of, 360
representation in a computer, 361–362
Externally programmed computer, 37
External store
function of, 194
External storage media, 400–402
organization of data in, 402–403

Extrapolative mode of operation (in a digital integrator), 479

Failure
basic types of, 522
Fault
intermittent, 540–541
location (utility of a revertive signal for), 389
Feedback
connections in differential analyzers, 450–451
Ferroelectric storage, 253–256
characteristics and examples of suitable materials, 253–254
dynamic capacitance, definition of, 253
partial selection in, 255
polarization hysteresis curve, 254
readout circuit, 254
schematic of a storage array, 255
storage cell, 254
File
external storage, 402
Filling a computer
logical requirements for, 425–426
Fixed-point computation, scaling for, 358–360
comparison with floating-point operation, 362–364
Flip-flop (circuits)
base gated DCTL, 141
DCTL flip-flop, 140
formed from a complementary current transistor switch, 149
formed from two cryotrons, 171
nonsaturating DCTL, 143
nonsaturating DCTL RC, 143
saturating DCTL emitter follower coupled flip-flop, 143
simplified transistor complementary current, 151
transistor complementary current, with four outputs, 152
Flip-flop (functional description), 72–87
application equation, general form, 76
derivation of a specific application equation, 76, 77
characteristic equations, derivation for T, R–S–T and R_T–S_T flip-flops, 79–82
difference equation, definition, 75, 76
dynamic, 85–87
R–S, T, R–S–T types, 86, 87
general description, 72–73
static, 73–75
R–S, T, R–S–T and R_T–S_T types, 74
Flip-flop (input equations)
derivation for R–S, T, R–S–T and R_T–S_T types, 77–79
derivation of specific equations for a particular application
from a Karnaugh map of application equation, 82, 83
from consideration of conditions preceding a change, 83–85
in a simple GP computer with a dynamic main store, 437–440
in a simple GP computer with a static main store, 423–425
Flip-flop (specific functions)
control flip-flops (in a DDA), 463
functions in a DDA, 473–474
major functions in a GP computer, 432
use of more than the minimum number, 433
Floating-point operation
comparison with fixed-point operation, 362–364
notation for numbers, 360–361
representation of floating-point numbers within a computer, 361–362
Flux, trapped
in a superconducting film, 236, 237
Fluxlok system, 227
Flux pattern (magnetic surface recording), 200
Format for representation of a floating-point number in a computer, 362
Format control
on a line-at-a-time printer, 567
Four-address system in a GP computer
description of, 393
Fractional computer
definition of, 311
restriction on size of operands in division, 340–341

Function generation
by Chebyshev polynomials, 514
by difference equations, 514
by power series, 514
errors peculiar to generation of certain functions, 508–509
function inversion by a digital servo in a DDA, 492–495
generation of e^{-x} by a digital integrator, 467–468
in a differential analyzer, 454–462, 477, 484
non-analytic function generation by decision units in a DDA, 485–487
Function table, encoding and decoding (see Switching matrix)
Functional relationships
as an aid to error detection, 534–535

Gates (see also Gating circuits)
functional representations of AND and OR gates, 103
general description of, 102–104
pyramid gates, 112
reduction of higher to lower level gates, 113
Gating circuits
diode gating circuits, 104–112
equivalent circuits of a semiconductor diode, 104
finite back resistance of diodes, effect of, 109, 110
load on gating circuit, effect of, 110
multi-level gating circuits, design of, 106–110
two-level diode gates, 107, 108
nonzero forward resistance of diodes, effect of, 109
single level gates, design of, 105, 106
switching speed, factors affecting, 111, 112
voltage and current requirements in, 110, 111
voltage current characteristic of a semiconductor diode, 104
formed from complementary transistor switches, 148, 149
magnetic gate, 153, 154
OR and AND gates, 159, 160
transfer loops for coupling gates, 155, 159
parametric oscillator circuits, 171–172
pulse-pedestal gate circuits, 114, 115
superconductive switching elements, 166–171
symbolic representation of magnetic core logic circuits, 154–159
tunnel diode circuits, 172, 173
General purpose computer (see GP computer), 16–17, 19, 20, 22–32, 372–447, 503
Gigacycle computer operation
increasing speed by means of local information processors, 385
Gigacycle, definition of, 171
gating circuits for operation at gigacycle frequencies, 171–173
GP computer (system design), 372–447
applications, 16–17
basic parameters in organization, 372
concluding remarks, 442–447
control unit, 378–412
fixed program type, 19
general description, 19, 22–32
instructions, number and type of, 372–376
logical designs of GP arithmetic computers, 412–442
main store, 376–377
sine wave generation, maximum frequency of, 503
system design, 372–447
word format, 377–378
Gray code (see Cyclic codes), 542–543

Half-selection (see also Partial selection)
minor hysteresis loop produced by half-select signals, 221
Harvard Mark I calculator, 10, 28
Hazards
in asynchronous networks, 390
Head selection matrix
in magnetic surface recording, 429
Head trailing effect, 211
Histogram, of error free operating intervals, 521

Hole storage (in a p-n-p transistor), 129, 134
Hysteresis loop
  minor, produced by half-select signals, 221
  of a ferromagnetic material, 152–153

Idle state
  automatic setting to, 531
  of a computer, 425, 439
Implicants, prime, 62
Inactive state (see Idle state)
Incremental analyzer (see Digital differential analyzer), 19, 20, 448–520
Incremental change
  in an integrator, 449
Incremental computer (see Digital differential analyzer), 19, 20, 448–520
  variable increment machines, 465
Incremental transfer computer (see Digital differential analyzer), 19, 20, 448–520
Increments
  restrictions on size in a DDA, 465
  variable, 465
Index registers, 394–399
  example of a program using index registers, 396–397
  means for addressing, 381
  modifications required in computer instruction repertory, 395
  representative computers in which incorporated, 399
  uses of, 398–399
Information processing systems
  applications of, 15, 16, 17, 498–501, 502–504
  elements of, 18
    central processors, 18
    information collecting devices, 18
    output terminals, 19
    transmission links, 18
  nature of, 18
Inhibit winding (in a coincident current magnetic core memory), 220
Inhibiting switching function, 86, 122
Inhibitor
  transistor complementary current, 151
Input equations (see Flip-flop input equations)
Input-output
  buffers, 402
  instructions, examples of, 376
  ratio of input-output to internal operations, effect of on computer organization, 401–402
Input-output equipment, 399–403, 554–572
  cathode-ray tubes, 569–571
  effect of on control unit, 399–403
    off-line operation, 399
    on-line operation, 399
  external storage media, 555–566
    magnetic tape, 560–566
    punched cards, 557–560
    punched paper tape, 555–557
  general remarks, 554–555
  individual character display devices, 571–572
  integration into a computer system, 399–403
  printers, 566–568
    character-at-a-time, 566–567
    high speed, 568
    line-at-a-time, 567
Instruction acquisition (look-up) commands, 383, 418–419
  in a microprogrammed control unit, 405
Instruction counter (see Control, counter, 379)
Instruction execution commands, 418–419
Instruction register (see Control, register, 379–382, 393–394, 416–417, 419–421)
Instruction repertory
  of a simple GP computer, 24, 415
Instructions
  B-box instructions
    B conditional, 396
    B modifiable, 395–396
    non-B modifiable, 395–396
  block transfer, 403
  dummy, in use of B-box instructions, 396

effective, 395
example of, 29, 30, 31
general types of, 23
modification of, 28
number and type of, 372–376
presumptive, 395
representative set for a GP computer, 374–376
tally, 396
three basic categories of, 383–384
Integral transfer computer (see GP computer), 16–17, 19, 20, 22–32, 372–447, 503
Integration formulas
choice of, 470–472
rectangular summation, 465, 467–470, 497, 515–516
example of function generation, 467–468
in multiplication, 497
trapezoidal summation, 465–466
in multiplication, 498
Integrator
bases for use, 451
digital (see Integrator, digital)
functional schematic, 449
general properties, 449
idealized equation, 450
mechanical, 449–450
Integrator, digital
assignment of a number to, 479–480, 502, 506
example of function generation, 467–468
interconnection of (mapping), 477–480
interpolative mode of operation, 466
mechanization of serial integrator, 472–473
simulation of in a GP machine, 515
sources and limits of inputs, 477
theory of, 467–472
used as a digital servo, 494
variations in design, 464
Intelligence (artificial), 16, 17
Interlock
for an external storage device, 403
system for nonregular networks, 390
Intermittent
errors, 523
faults, 540–541
Internal store
function of, 194
Interpolative formulas
in trigonometric function generation, 357–358
Interpolative mode of operation (in a digital integrator), 466, 479
Interrogation
of a magnetic core memory, 218
Interrupt control (see Program interrupt control), 410–412
Inverse check, 535
Inverter circuit, 116, 117
AND and OR gates formed from inverters, 117
junction transistor circuits, 134, 135
parallel inverter, 117
Iteration rate
in a serial DDA, 474
in parallel DDA's, 503
Iterative formulas, 349–352
derivation of a general formula, 349–350
formula for the reciprocal, 350
formulas for higher order roots, 355–356
formulas for the square root, 351–352

Junction transistor circuits
current switching circuits, 147–152
non-saturating complementary current switching and inhibiting circuits, 150–152
non-saturating complementary current switching systems, 147–150
dynamic pulse circuits, 143, 144
gated-pulse amplifier circuits, 144, 147

Language, machine (see Machine language), 444, 445
Latency time
figures for typical rotating magnetic stores, 215
in a magnetic disk store, 216
Level discrimination in adders, 277

Limiting of a function
in a DDA, 486–487
Linear select memory
(see Word organized memory), 228–230
Locating data
in a magnetic tape store, 562
Logic tabulation, 444–445
Logical
instructions, examples of, 375
product, type of instructions, 375
Logical design
aids in description of, 443
definition of, 443
general description of, 94
Logical design, modification of difficulties introduced by schemes for equipment minimization, 446
Logical designs, of GP arithmetic computers, 412–442
GP computer with a static main store, 415–426
arithmetic unit, 417–418
description of control unit, 416–417
elementary commands, 418–420
flip-flop input equations, 423–425
flow diagram of instruction execution, 422
instruction repertory, 415
other specifications, 415
word format, 416
GP computer with a dynamic main store, 426–442
block diagram of organization, 427
circulating registers, 431
flip-flop input equations, 437–440
instruction repertory, 426
other specifications, 426
permanent timing tracks, 430
recording equations, 440–442
time duration signals, 430–431
use of passive storage elements for information processing and control, 427
word format, 428–429
Logical elements (see Gates and Gating circuits)
Loops, high speed (see Revolvers), 428
Losses in dynamic magnetic recording systems, 203–204
gap loss, 204
self-demagnetization, 204
spacing losses, 203
thickness loss, 203–204

Machine language
definition of, 444
use in a computer simulation program, 445
Magnetic coatings
effect on read voltage variation with recording density, 210–211
on a drum surface, 212
Magnetic core logic circuits, 152–166
symbols for, 154, 159
Magnetic core storage, 216–229
coincident current core memory, 218–223
desirable characteristics of the core material for memory applications, 217
general remarks, 216–218
word organized core memory, 228–229
Magnetic core switches (for memory drive systems), 223–226
anti-coincident current switch, 225
biased coincident current switch, 224
biased multi-coincidence switch, 225, 226
multi-coincidence switch, 225
Magnetic disk store (see Magnetic surface storage, also 199–216)
general description of, 214–216
characteristics of typical multi-disk units, 215
in a serial DDA, 473, 474
Magnetic drum store (see Magnetic surface storage, also 199–216)
characteristics of typical units, 215
general description, 211–213
access time, means for improving, 213
head-to-surface spacing variation, 212
record-read system for one channel, 211

head selection matrix, 213
in a serial DDA, 473, 474
Magnetic heads
air bearings for, 212
in IBM-RAMAC memory, 216
design of, 200, 201
switching matrix for selection of, 213
Magnetic recording (see Magnetic surface storage), 199–216
Magnetic reluctance
definition of, 201
Magnetic surface (rotating) storage, 199–216
coding techniques, 204–211
comparison of NRZ and RZ recording, 208–209
general remarks, 204–205
NRZ recording, 207–208
RZ recording, 205–207
phase modulation recording, 209–210
efficiency of, 202–203
general remarks, 199
memory transfer function, 203–204
recording process, 200–201
Magnetic tape, 560–566
advantages of, 564–565
comparison with disks and drums, 565
composition, 560
head stack design, 560–561
organization of data, 560
storage capacity, 565
Magnetic tape control unit
facilities for searching operations, 403
Magnetic tape transport
start and stop times, 562–566
schemes for reducing, 563–564
Magnetostrictive delay line
schematic of, 248
typical amounts of delay, 251
Magnetostrictive effect, 248, 249
Main store (see Store, main), 376–377, 387–388
Maintenance
difficulties introduced by schemes for equipment minimization, 446
logic and usage tabulations as an aid to, 445
preventive, 540–541
Mantissa
definition of, 360
representation in a computer, 361–362
Many-to-many networks (see Switching matrix), 178
Many-to-one networks (see Switching matrix)
Mapping (in a DDA), 477–480, 506
errors producible by schematic representation of, 511–512
in a special purpose DDA, 502
Marginal
checking, 540–541
operation, 523
Marker bits
in an addressing channel, 475–476
Mass storage units
characteristics of typical multi-disk units, 215
Matrix printer, 568
Matrix switch (see Switching matrix)
Mean (round-off error), 545–547
Mean-time-to-failure, 522
Mechanical differential analyzers
accuracy, 452
auxiliary elements, 453
general description, 449–450
Meissner effect (in superconductors), 168
Memory (see Storage systems and Store, main)
advantages of storing numbers and instructions in a common unit, 28
differentiation between numbers and instructions, 27, 28
Memory cores (ferrite)
typical dimensions, 216, 217
Microcontrol unit, 406
Microprogrammed control unit
control register unit, 404–405
Microprograms
effect of on control unit, 404–410
execution of microprogrammed instructions, 406–410
flow diagram of, 408
means of access to, 410
Micro-operations (see Microprograms), 404

Microwave circuits, 171–173
parametric oscillators, 172
tunnel diode circuits, 172, 173
Minimization of equipment
disadvantages of, 446
Minimum access programming
for a nonrandom access main store, 393
Minimum translational network, 177
Minority carrier storage, 132, 134, 135
Mirror system of notation for magnetic core circuits, 225, 226
MIT Computer Laboratories
pentode gate system, 119, 120
Modulo checks (see Residue checks), 531–532
Multiaperture devices
transfluxor gates, 163–166
Multiple computation, 534
Multiple parity bits, 527
Multiple-precision
arithmetic, 550–551
numbers, 544–545
Multiplexing, 414–415
in a DDA, 414
Multiplication (see Binary multiplication, 318–335 and Decimal multiplication, 335–340)
analysis of an integrator network for multiplication in a DDA, 494–498
by special operational units in a DDA, 499–502
general remarks, 316–318
in computers for solving navigation problems, 499–502
incremental (in a DDA), 453–454
of numbers in floating-point notation, 360–361
table for binary numbers, 317
Multiplier (see Multiplication)
binary
serial binary multiplier with delay line storage, 322
serial-parallel binary multiplier, 323–325
serial-parallel binary multiplier controlled by paired bits of the multiplier, 325–327
typical stage of an asynchronous binary multiplier, 327
simultaneous multiplier, 328–330
decimal
serial-parallel multiplier, 339
incremental, 453–454
output multipliers in a DDA, 483–484

Nanosecond
definition of, 111
National Bureau of Standards
AC system of circuit logic, 120–123
Navigation
digital computers as an aid to, 498–501
Negative numbers, representation of, 308–315
by absolute value plus a sign, 308–310
complementary notation, 310–312
nine's complement, 313–315
one's complement, 311–312
ten's complement, 312–313
two's complement, 310–311
Negative remainder in division, 341
Networks, regular
asynchronous operation of, 390
definition of, 390
Noise
delta noise in a coincident current core memory, 222
delta noise in a thin film memory, 235
elimination of, in a superconductive memory, 237
reduction of, in a magnetic core array, 219
Nonanalytic function generation
by decision units in a DDA, 484
Nondestructive readout
in a magnetic core memory, 226–228
by elastic motion of domain walls, 227
quadrature field methods, 227
FLUXLOK, 227
RF sensing, 227
zero flux, 227, 228
in a transfluxor memory, 231
in a tunnel diode memory, 239
Normalization of equations
in a DDA, 482–483

NRZ recording, 207–208
read waveforms, 208
Numbers
binary coded, 11, 12
binary information transfer, 13
circular number system in a digital integrator, 489
floating point notation, 360–361
parallel representation, 11–13
serial representation, 11–13
sign designation, 13, 14
ternary information transfer, 13
unitary weighted, 11, 12
Numerals
cardinal, 1
display devices, 570–572
external storage media, 556–567
ordinal, 1
printers, 567–569
Numerical representation
in an electronic digital computer, 11, 13
in the arithmetic unit
effect on control unit, 390–392
Numerical symbols
development of, 2, 3
Roman numerals, 3
uses of, 1

Off-line
data preparation, 400, 555
operation of input-output equipment, 399
On-line
operation of input-output equipment, 399
One-address system in a GP computer
description of, 393
One cycle of operation, 439
One-shot delay circuit (delay multivibrator), 187
One-to-many-networks (see Switching matrix)
Operand storage register, 417–418
Operating speed (of a storage system), 194
Operating time
consistency of, in various computer elements, 390
Operation codes
basic types and their effect on decoding and encoding function tables, 381–382
Operation field, 24
Operation period
minor and major periods in synchronous control of a GP machine, 386–387
Operation time
in asynchronous circuits, 124
Operational units (in a DDA)
basic units, 475
more complex units, 498–502
Order codes (see Operation codes), 381–382
function of, 432
Order register (see Instruction register) 416–417
Ordering of arithmetic operations
for minimum round-off error, 548–551
Organization
of a serial DDA, 472–476
Orthogonal fields (see Nondestructive readout), 227
Output data
considerations in choice of output recording devices, 401
forms of, 401
Output devices (see Input-output equipment), 554–572
Output rate
of data in magnetic tape units, 566
Overflow (in an accumulator)
automatic checking of, 531
logical conditions describing, 440
of *R* accumulator in a DDA, 464, 468–469

Parametric oscillator, 171, 172
Parity checking, 525–530
array, 526
even parity, 525
hardware requirements, 529–530
multiple error detection, 526–527
multiple parity bits, 527
odd parity, 525
on magnetic tape, 562

single error correction scheme, 527–530
single error detection, 526
Partial selection
in a coincident current memory, 218–219, 221–222
Passive storage elements
in a serial DDA, 475
Pattern generators, 273–275
Peripheral equipment (see Input-output equipment), 553–571
Persistent-supercurrent storage elements, 236–238
Phase error (in a DDA), 507–508
Phase-locked subharmonic oscillator, 172
Phase modulation recording
record and read waveforms, 209
Phases of operation, in a computer, 432–436
Photoelectric readers
of paper tape, 557
Playback voltage (see Read voltage)
Plugboard controlled computer, 37
Point contact transistors
nonsaturating flip-flops, 132
saturating flip-flops, 128–132
Polish notation, 107
Positional notation
advantages of, 4, 5, 15
Post-write-disturb pulse (in a coincident current core memory), 222, 223
Power series
for function generation, 514
Precision
trade off with solution time in a DDA, 482
Preparation of problems
for a DDA, 476–484
general remarks, 476–477
Presumptive instruction, 395
Preventive maintenance, 540–541
Prime implicants, 62
Princeton Institute for Advanced Study, 127
type of operation code in IAS computer, 382
Printers, 566–568
character-at-a-time, 566–567
film, 570
high speed, 568
line-at-a-time, 567
Xerox, 57
Priority interrupt control (see Program-interrupt control), 410–412
Priority number, in program-interrupt control, 410–411
Problem preparation for a DDA, 476–484
assignment of integrator numbers in a serial DDA, 479–480
mapping (establishing interconnection of units), 477–480
normalization of equations, 482–483
scaling, 480–482
Program-interrupt control, 410–412
effect of on control unit, 411–412
examples of types of demands, 411
Programmed error detection and correction, 533–538
analytic checks, 534–537
data transfer checks, 537–538
general remarks, 533–534
basic procedures, 533
comparison of criteria for laboratory and control computers, 533
sequencing checks (programmed), 537
testing and diagnostic programs, 538–540
Programming
minimum access, 393
search operations on magnetic tape, 565–566
Programs
debugging routines, 34
flow diagrams
example of, 34–36
function of, 33
procedures in preparation of, 32–34
production running, 34
program for determining the highest factor of an integer, 25–27
program for simulating a DDA on a GP machine, 513–517
subprograms (subroutines), 31–34
tracing, 523, 539
test, 523

Propagation time
  importance of, 173
Pseudo-operations
  due to round off, 548–549
Pulse density (magnetic recording)
  effect on read voltage, 211
  in typical mass storage units, 216
Punched card
  advantages of, 559
  Hollerith, 10
  IBM, 558
  Remington Rand, 558
Punched paper tape readers and recorders, 555–557
Punches
  card, 558–560
  paper tape, 555–557
Pyrimidal switching matrix
  many-to-one, 175–177
  one-to-many, 177–178
Pyramiding
  in multi-input core gating circuits, 162

Quantization
  binary, 8
  nature of, 7, 8
  reasons for, 7, 8
Quartz (fused), velocity and attenuation figures for RF transmission, 247, 248

Read-around ratio
  (in a cathode-ray tube store), 242
Read voltage (magnetic surface recording)
  variation with head spacing, 201
  variation with recording density for different magnetic coatings, 211
  waveforms from induced positive and negative poles, 202
  waveforms in phase modulation recording, 209
Readers
  card, 558–559
  magnetic tape, 560–566
  punched paper tape, 557
Reciprocal, computation of, 350
Recomputation
  for error detection, 534
Recording flux pattern, 200
Recording, magnetic (see Magnetic surface storage, 199–216 and Magnetic drum and disk store, 211–215, 473, 474)
Rectangular switching matrix, 174–178
Redundancy
  definition of, 52
  in built-in storage and transfer checks, 525, 530
  in initial designs of new equipment, 15
  in numerical representation, 14
  in primitive notations, 15
Reflected binary codes (see Cyclic codes), 542–543
Regeneration cycle, 195
Register
  buffer register, location of, 426
  circulating registers, 431, 473
  control register, 416–417
  instruction register, 419–421
  operand storage register, 417–418
  Σdy (in a DDA), 463
  shift register, 85, 180–186
Relay computers, 10
Reliability, 521
Remanence, magnetic, 153
Remanent state, of a magnetic core, 222
Repeat counter, 399
Residue checks, 531–532
Return address
  in priority-interrupt control, 412
Return-to-bias magnetic recording, 209
Revertive signal, in asynchronous control, 389
Revolvers, in a dynamic memory, 428
Rollback program, 539–540
Roots, method of computing, 351–356
  higher order roots, 355–356
  square root, 351–355
Rotating magnetic memory (see Magnetic surface storage, 199–216 and Magnetic drum and disk store, 211–215, 473, 474)
Round off (of a product), 318
Round-off error, 543–550
  absolute, 544
  bias, 545–547

effect on multiplication, 549
effect on scaling, 551
in a DDA, 505
in a digital integrator, 470
mean, 545–547
multiple-precision operations, 544–545
relative, 544
standard deviation, 545–547
Round-off procedures
mechanization of, 546–548
in a binary system, 546–547
in a decimal system, 547–548
summary of binary and decimal schemes, 548
Row check (see Parity checking), 526
RZ magnetic recording, 205–207
head arrangement in, 205
read waveforms for various recording densities, 206

Sawtooth function generation
by decision units in a DDA, 486
Scale factors
in a DDA, 480–482
derivation of an optimal set, 482
in fixed-point operation, 359
Scaling
effect in limiting magnitude of errors, 536
in a DDA, 480–482
checking solutions by runs with different scaling, 512
effect on computational error, 506–507
effect on solution time, 482
scaling relationships for a set of integrators, 481–482
scaling relationships within an integrator, 480–481
summary of steps, 482
systematic generation of an optimal set of scales, 482
in a GP computer, 358–364
comparison of fixed and floating-point operation in a computer, 362–364
fixed point computation, 358–360
floating point notation for numbers, 360–361
representation of floating point numbers within a computer, 361–362
influence of round-off error, 551
Search and acquisition cycle
in a GP computer, 379
Search operation
in an external store, 403
in a GP computer, 433–434
Secondary emission coefficient of a phosphor surface, 240–241
Sector number
of an address in a dynamic store, 430
Selection
of a word in a dynamic store, 380–381
of a word in a static store, 380
Selection network for a static storage system
general remarks, 195–197
Selection ratio
in a coincident current memory, 219, 221, 222
Self checking
of data on magnetic tape, 561
Self demagnetization
in dynamic recording media, 204, 210, 551
Sense instructions, examples of, 375–376
Sense voltage
strobing of in a coincident current core memory, 221, 222
Sense winding
in a coincident current array, 219
Sequencing, automatic methods of, 37
Sequencing of arithmetic operations
for minimum round-off error, 548–551
Sequencing checks
built-in, 532
programmed, 537
Serial or parallel operation
comparison of, for certain figures of merit, 385
effect on control unit, 384–385
Servo adder (in a DDA), 464, 477
Servo, digital (in a DDA), 488–494
addition of rates by, 488–491
hard servo (integrator), 488–491
soft servo (integrator), 490–491

errors produced by, 511
function inversion, 492–495
servo action from an accumulator, 494
servo action from a decision unit, 494
servo with gain, 491–492
Set, determination of
elements in by counting, 2
Shift operations
in scaling of problems, 361–362
Shift register, 180–186
arrangements for vacuum tubes or transistors, 180–182
magnetic core shift registers, 182–186
transistor-magnetic core shift register, 185–186
Significant digits, loss of through incorrect scaling, 363–364
Signum function, 485
Simulation
of a continuous differential analyzer by a digital computer, 514
of a DDA by a GP machine, 513–517
of an integrator by a GP machine, 515
Simulation programs as an aid to computer design, 445
developing maintenance procedures, 445
Sine wave
maximum frequency of generation by different computers, 503
Single-address GP computer
schematic of over-all control arrangement, 383
Skew effect in magnetic tape, 561
Smoothness test, 535–536
Solution time
trade off with precision in a DDA, 482
Space domain storage (see also Storage systems)
general remarks, 195–197
Special purpose computer, 20
SPEDAC computer, 503
Speed, of
card punches, 558
card readers, 558
cathode-ray tube displays, 569
character-at-a-time printers, 566
electronic printers, 570
high speed printers, 568
line-at-a-time printers, 567
magnetic tape units, 564
paper tape punches, 556
paper tape readers, 557
Speed independence
in asynchronous circuits, 123–127
University of Illinois design techniques, 126, 127
Spot check, 512
Square root, methods for computation of, 351–355
interpolative formula, 352
iterative formulas, 351–352
odd series approximation, 352–355
in the binary system, 354–355
in the decimal system, 352–354
Standard deviation (or round-off error), 545–547
Starting a computer's operation
logical scheme for, 425
Start-up error (in a DDA), 508
State of a computer
conditions that define active states, 431–432
Static magnetic storage, 216–236
apertured ferrite plate, 230–231
magnetic core storage, 216–229
coincident current memory, 218–223
word organized memory, 228–229
thin film elements, 234–236
transfluxor, 231, 232
twistor, 232–234
Static storage (see also Storage systems)
general remarks, 195–197
Stibitz (Bell Telephone relay computer), 10
Storage cycle
in cathode-ray tube store, 242
in coincident current magnetic core memory, 222–223
Storage media
external (see also Input-output equipment), 400–402, 555–566
internal (see Storage systems)
Storage systems
access time, 195

criteria for evaluation, 194
erasibility, 195
general remarks, 194–199
operating speed, 194
regeneration cycle, 195
types currently dominant, 199–256
cathode-ray tube storage, 239–243
diode, capacitor storage, 251–253
dynamic delay line storage, 243–251
dynamic magnetic storage, 199–216
ferroelectric storage, 253–256
static magnetic storage, 216–236
superconductive storage, 236–238
tunnel-diode storage, 238–239
volatility, 195
Store (see Storage systems)
Store, main (see also Storage systems)
control of selection circuits, 387–388
general remarks, 376–377
Substitution check, 512
Subtraction, 308–315
by use of complements (see Complementary numbers, 310–315)
of binary coded decimals, 312–315
Subtractors
logical equations for difference and borrow digits, 309
truth table for a full-subtractor, 309
truth table for a half-subtractor, 309
Sum check (of a memory), 538
Summation
of rates in a DDA (see Servo adders)
rectangular (in a DDA), 465, 467–470
example of, 467–468
Superconductive elements
storage elements, 236–238
switching elements, 168–171
Superconductivity
general description of, 166, 167
Switches
complementary current, 148
simplified complementary current, 150
Switching
in a DDA program (by decision units), 487–488
Switching constant, 217
Switching functions
complementary functions, generation of, 115, 116
use of inverters in, 116
representation by Boolean algebra, 45–50
Switching matrix
decoder and encoder in a synchronous control unit, 387
diodes required for different forms of a translational network, 177
for magnetic core memory drive systems, 223–226
for magnetic head selection, 213, 429
general description, 173–178
pyramidal switching matrix, examples of, 175–178
rectangular switching matrix, examples of, 174–178
Switching networks, combinational
an adder as an example of a multiple-output network, 283
distribution and collection networks, 178–180
many-to-many networks, 178
miscellaneous form of (multiple level), 59
model of, 51
selection network for a static store
general remarks, 195–197
translational networks, 173–178
Switching networks, sequential
asynchronous operation, 90
equivalent states, definition of, 91
equivalent states, elimination of, 91
general description of, 87
general nature of, 72
interconnection of switching and storage elements general description, 72
minimization of storage elements in, 91
Huffman-Mealy method, 91
model of, 88
reasons for use of more than a minimum number of storage elements, 92
storage elements in, 69–72
superstates, general description of, 89
synchronous operation, 90

Switching time
in a magnetic core, 217
Symbols for magnetic core logic circuits, 154, 159
Synchronizer
definition of, 96
Synchronous (clock) control
comparison of synchronous and asynchronous control, 389–391
nature of, 382, 385–386
schematic of synchronous control for a single address GP computer, 387
Synchronous operation of a computer
effect on control unit, 385–391
effect on microprogrammed control, 410

Tabulations
logic, wiring and usage tabulations in a computer, 444–445
Tags, identifying
for data in a magnetic tape store, 562
Tally instruction, 396
Tally number, 397
Taylor series expansion, in trigonometric function generation, 356–357
Temperature coefficient
of coercivity for ferrite core materials, 229
Temperature range
transfluxor memory, 231
Ternary transfer, 13
in a DDA, 464
Test programs, 523
Thin film memory
continuous plane type of superconductive film memory, 238
superconductive film storage elements, 236–238
Three address system in a GP computer, 393
Time domain storage (see Dynamic storage), 197–198
Time sharing
example of use in a GP computer, 427 ff
of arithmetic unit in a serial DDA, 473
of storage elements, 413–415
Timing channels
in a dynamic magnetic memory, 429–430
Timing control
paper tape punch, 556–557
Timing pattern generator
for a synchronous control unit, 386, 387
Timing signals
binary counters for generation of, 95, 96
derivation from permanent timing tracks, 430–431
logical equations of, 435
multiple timing sources for magnetic core gating systems, 160
purpose of, 95, 96
synchronous control of a GP computer, 386
Timing tracks (in a rotating magnetic memory), 430–431
Tracing programs, 523, 539
Track number
of an address in a dynamic store, 430
Trade-off
between precision and solution time in a DDA, 482
Transfer checks, 537–538
Transfer loops
for magnetic gates, 155–159
Transfer of control instructions, examples of, 375
Transfluxor memory, 231, 232
Transfluxors
gating circuits, 165–166
general description, 163–165
Transients
in carrying propagation, 287
Transistor
alloy, 134
electron storage in an n-p-n transistor, 134
epitaxial, 134
hole storage in a p-n-p transistor, 129, 134
mesa, 134
micro-alloy, 134
minority carrier storage, 132, 134, 135

Transit time
  use of local information processors to alleviate effects of, 385
Transition temperature for a superconductor, 166
Translation, computer language, 566
Translational networks (see Switching matrix)
TRICE computer, 503
Trigger circuits
  relaxation oscillator circuits, 186, 187
  single-stable-state circuits, 187
  two-stable-state circuits, 187
Trigonometric function generation, 356–358
Truncation error
  description of, 544
  in a DDA, 505
Truth table, 39
Tunnel diode
  characteristic curve of, 239
  logic circuits, 172, 173
  memory, 238–239
Twistor memory, 232–234
Two-address system in a GP computer, 393
Typewriter, electric
  as an on-line data entry device, 400
Typotron, 569–570

University of Illinois Digital Computer Laboratory
  asynchronous computer design, 125–127
University of Manchester computer
  organization of serial-parallel multiplier, 324–325

Variable increment computers, 465
Verification of input data, means of, 400
Volatility (of a storage medium), 195

Williams storage system (cathode-ray tube storage), 239–243
Wiring tabulations
  uses of, 445
Word format in a computer
  criteria affecting, 377–378
Word organized memory, 228–229
  apertured ferrite plate, 230
  magnetic core, 228–229
  tunnel diode memory, 239
  twistor memory, 233

Xerox printer, 570

Z line
  in a DDA, 475
Zero access time
  in a DDA, 475
Zero rate
  generation in a DDA with binary transfer, 464–465
Zero, representations of in complementary number systems, 312