UNDERSTANDING DIGITAL COMPUTERS
SIEGEL

UNDERSTANDING DIGITAL COMPUTERS

BY PAUL SIEGEL

■ A CAREFULLY DEVELOPED EXPOSITION OF THE FUNDAMENTALS OF OPERATING DIGITAL COMPUTERS AND DATA PROCESSORS ■ ■ ■

About the book . . .

This book has many remarkable features, but it is the last chapter which most demands special attention. In this chapter, which offers a detailed description of a specimen computer, the author ties together the concepts and principles that have been so clearly and carefully developed earlier in the book. It is, in a sense, the "capstone" of an unusually thorough, easy-to-follow, and well conceived exposition of the principles involved in operating and programming digital computers.

After an introduction which gives a broad explanation of what a computer is, the exposition is divided into three main sections: Section I discusses the basic logical elements; Section II summarizes the circuits which can be used as computer building blocks and at the same time reinforces the previously learned logical concepts; the last covers the functional parts of the computer and ends with the above-mentioned chapter on a specimen computer, which effectively puts across the **practical** meaning of the material presented earlier in the book. The emphasis throughout is on principles rather than hardware, and each principle is presented and developed with a host of illuminating examples and a minimum of complex mathematics. To aid the reader, there is a summary at the end of each chapter which includes a list of the important definitions contained therein.

About the author . . .

Paul Siegel is a systems engineer with the International Electric Corporation, an associate of the International Telephone and Telegraph Corporation, where he is presently studying exotic applications of digital computers to sophisticated command and control systems. At ITT he has also been involved in the coordination of systems testing and in the development of standards for easing the flow of information through a systems management organization.

At the Burroughs Research Center, with which he was associated previous to assuming his present position, the author worked on important research documents on magnetic and transistor digital circuit techniques to be used on SAGE. He has also been associated with Remington Rand Univac and has taught and lectured both here and abroad on electronic subjects. He holds a B.S. in physics from City College of New York and an M.S. in electrical engineering from Rutgers University.

LOGICAL DESIGN OF DIGITAL COMPUTERS

By **Montgomery Phister, Jr.,** Scantlin Electronics, Inc: Here is a sound, useful approach to the logical design of computer systems. Stressing general techniques rather than specific computers, the book is replete with worked-out examples and requires no previous experience on the part of the reader. Because of the importance of Boolean algebra to the designer, a full chapter is devoted to developing the subject. The various methods of simplifying Boolean functions are treated in a separate chapter, which covers the Quine methods and the Harvard method as well as the Veitch diagram method of simplification. The book represents the first published description of the logical-equation school of design. 1958. 408 pages.

PROGRAMMING BUSINESS COMPUTERS

By **Daniel D. McCracken,** Private Consultant in Computer Programming; and **Harold Weiss** and **Tsai-Hwa Lee,** both of the Computer Department of the General Electric Company. This book provides an elementary yet authoritative study of the commercial aspects of electronic computing. It has been specifically designed for readers who lack a background in mathematics but who are interested in the application of computers to business data processing.

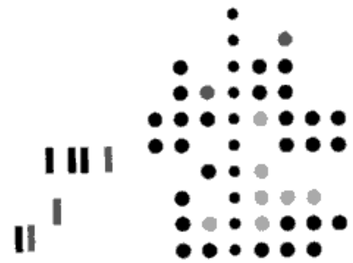
"Here is the first top-notch book on programming of electronic data processing machines. A number of books have been published on the subject, several of them recently, but the coverage in this new book by McCracken-Weiss-Lee is the best we have seen—comprehensive, authoritative, and up-to-date."—**Data Processing Digest.** 1959. 510 pages.

JOHN WILEY & SONS NEW YORK • LONDON

UNDERSTANDING DIGITAL COMPUTERS

SIEGEL

Understanding digital computers



UNDERSTANDING DIGITAL COMPUTERS

PAUL SIEGEL
International Electric Corporation
Associate of ITT

JOHN WILEY AND SONS, INC.
NEW YORK AND LONDON

Copyright © 1961 by John Wiley & Sons, Inc.

All Rights Reserved. This book or any part thereof must not be reproduced in any form without the written permission of the publisher.

Library of Congress Catalog Card Number: 61-15412

Printed in the United States of America

SECOND PRINTING, MARCH, 1963

To my wife, Evelyn,
without whose forbearance
I never could have found
the time to work on this book

Preface

This book is a comprehensive introduction to the fundamentals of digital computers and data processors. It tries to give the reader a firm understanding of these machines and their design. After finishing it the reader will be able "to talk the language."

The book is written primarily for the technician who is thinking of entering the digital computer field. But anyone with a basic understanding of electronics—a "ham" for instance—can read it with complete understanding.

Not a design handbook, it is about principles, and each principle is illustrated by means of examples.

The book has an introductory chapter and a body divided into three sections. The introductory chapter describes the digital computer and its functions. In a sense, it is a distillation of the entire book. It provides the reader with an appreciation—a feel—for digital computers and a craving to learn more about them.

The three sections of the book take the reader from the theoretical discussion of elemental logical elements to circuits which can be used as building blocks, and conclude with the building of a specimen computer.

None of the material found herein is original. For the most part it has been extracted from papers appearing in the *Proceedings of the Institute of Radio Engineers*, the *Journal of the Association for Computing Machinery*, and other technical journals. But I do believe that the material is presented in a fashion which facilitates learning. Basic ideas are presented first, and each succeeding chapter builds upon the facts presented in previous chapters. The essence of each chapter is put into a summary which includes a list of definitions of key words developed in that chapter. The exposition is made as simple and with as little mathematics as possible.

I gratefully acknowledge the constructive criticism of Al Meyerhoff and

Ed Veitch of Burroughs Corporation, and I am especially indebted to James Maginnis, Director of the Computing Laboratory of Drexel University, for his thorough critique of this work.

PAUL SIEGEL

Washington Township,
P.O. Westwood, New Jersey
June 1961

Contents

1	Introduction: What is a Digital Computer	1
	section I. LOGIC AND ARITHMETIC	**21**
2	Word and Number Languages	23
3	Arithmetic Processes	56
4	Machine Logic	79
	section II. BUILDING BLOCKS	**99**
5	Mechanical and Electromechanical Components	101
6	Vacuum-Tube and Related Components	120
7	Electromagnetic Components	150
8	Germanium Diodes and Transistors	185
9	Other Devices and Circuits	209
10	Logical and Functional Building Blocks	226
	section III. FUNCTIONAL UNITS OF A DIGITAL COMPUTER	**235**
11	Machine Language	237
12	Memory Unit	254
13	Input-Output System	274
14	Arithmetic Unit	299

15 Control Unit 326
16 A Specimen Computer 347

Bibliography 383
Glossary 385
Index 395

CHAPTER 1

Introduction: What is a digital computer?

What is a digital computer? Is it a lightning calculator which can multiply two 10-digit numbers in the time it takes a jet plane to fly 1 inch? Is it a data-processing machine which can automatically spew out yards of accurate and complete business reports? Or is it a device which can make decisions and check its own work? Or a device which accepts a set of thousands of instructions and then faithfully executes them in the appropriate sequence without outside help?

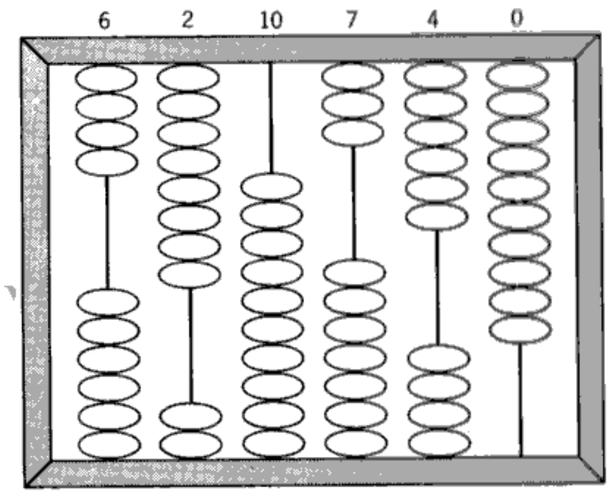
The digital computer is all these and more. It is more because its complicated mechanism is composed of childishly simple, basic elements; because it applies the rules of logic to simulate many of the functions of the human brain; and because of its fantastic accuracy, reliability, and flexibility.

The basic principles of operation of the remarkable electronic digital computer are portrayed in this introductory chapter. The more prosaic details of operation are presented in the remaining chapters.

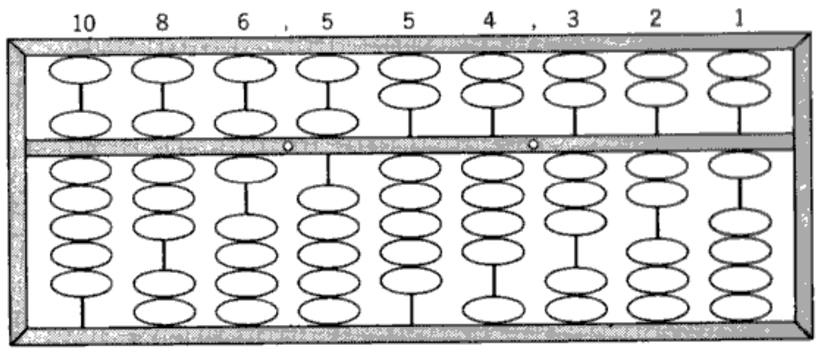
DIGITAL DEVICES

In what way is a digital computer different from the many calculating machines man has devised ever since he learned how to count? Before we can discuss the digital computer itself, we must know what is meant by a digital device.

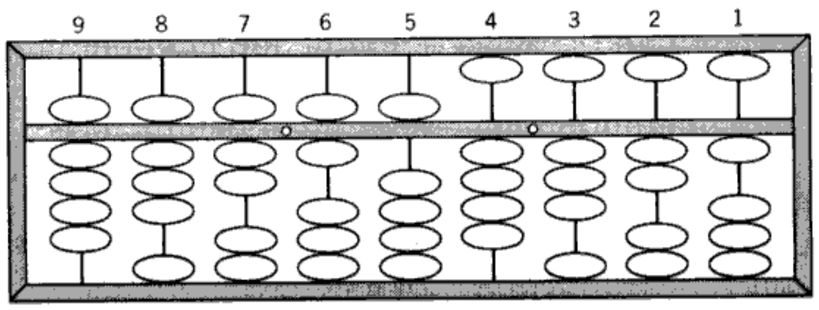
The simplest digital device is any device which can count. The simplest of all counting devices are the ten fingers. As a matter of fact, the word "digital" is derived from the latin *digitus*, which means finger. Another simple digital device is the clock. It ticks away, or counts, little bits of time called seconds.



(a)



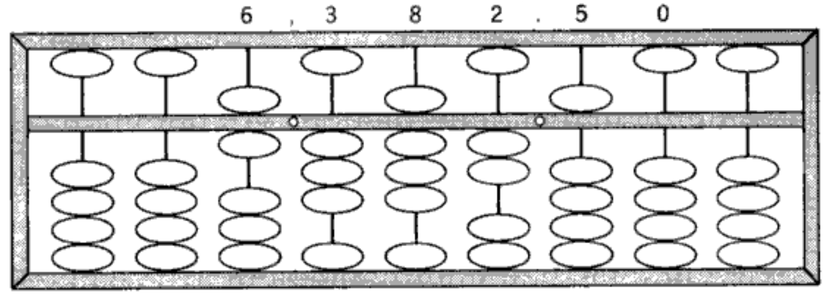
(b)



(c)

Fig. 1. The counting board and the abacus. (*a*) Counting board. (*b*) Abacus—Chinese. (*c*) Abacus—Japanese.

In ancient days man learned to substitute beads for fingers in order to help him count. He built a counting board which consisted of several stiff wires mounted on a wooden frame, with ten sliding beads on each wire (Fig. 1*a*). By moving a certain number of beads toward the bottom of the board, any digit from 0 to 9 can be placed on each of the wires. Even 10 can be placed on a wire, as is shown in the figure. This 10 is interpreted as a 0 in the indicated column plus a carry of 1 to the column to its left. The counting board in the figure is storing 630,740.



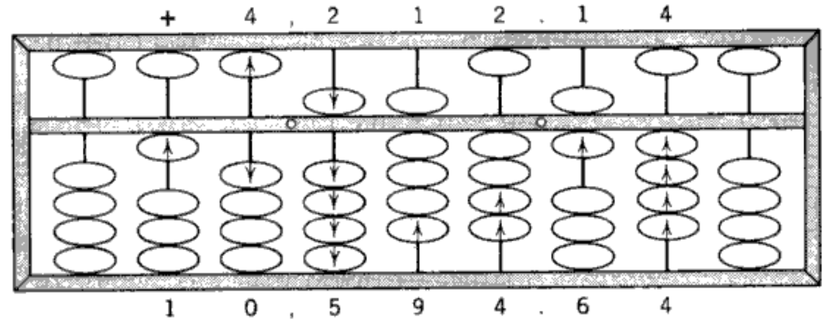


Fig. 2. Addition by means of abacus.

The ancient Chinese simplified the counting board into the abacus (Fig. 1*b*). Instead of ten beads on each wire, the abacus has seven. Two beads are mounted above a dividing wooden member, and five beads are mounted below the wooden member. All beads are in neutral position when they are furthest from the dividing member. Each of the lower beads has a value of one; each of the upper beads has a value of five. The number 1,086,554,321 is stored in the abacus of Fig. 1*b*.

The Japanese made the abacus more efficient. They used only five beads: four 1-unit beads below the dividing member and one 5-unit bead above. The number 987,654,321 is stored in the Japanese abacus of Fig. 1*c*.

Figure 2 illustrates how the Japanese abacus is used to add $6,382.50

and \$4,212.14. First \$6,382.50 are placed in the abacus as shown. The first period from the left on the dividing member is used as a comma and the second period is used as a decimal point. Four 1-unit beads are added to the rightmost column; one 1-unit bead is added to the 5-unit bead stored in column 2 (assuming that columns are numbered consecutively starting with the rightmost column as 1); two 1-unit beads are added to the two 1-unit beads in column 3; and a 1-unit bead is added to the representation for eight stored in column 4.

In the fifth column the operation is not so straightforward because two 1-unit beads must be added to the three already stored. This cannot be done since there are only four 1-unit beads to a column. So all the 1-unit beads are brought back to neutral and the 5-unit bead is brought down to the reading position. In column 6 we must add four 1-unit beads to a column where a 1-unit bead and the 5-unit bead are already stored. This is obviously impossible since there are only three 1-unit beads left in the column. So all the beads are brought to neutral and a carry is placed in column 7. The sum can be read from the abacus as \$10,594.64.

By extensions of the above process, subtraction, multiplication, and division may be performed on the abacus.

MECHANICAL CALCULATING MACHINES

A great step forward in the art of adding mechanically was made by Blaise Pascal in 1642. Pascal used wheels divided into parts instead of wires holding ten beads. The ten-part wheel can be compared to the wire with the ten beads bent so the two ends meet. The wheel represents a great improvement over the bead-holding wire, especially in the production of a carry. When going from 9 to 0 in any one column of the abacus, all the beads must be brought to neutral and a unit bead must be added to the column to its left. In the wheel-using machine just one turn of the wheel is all that is needed. The wheel moves to 0 and mechanical linkage is used to automatically cause a small carry movement in the wheel to its left.

Figure 3 shows 6431 stored in a wheel-using calculator. To add 7453 to this number, each wheel is caused to turn by the indicated number of wheel divisions. During the rotation of the fourth wheel from the right, the 0 division passes the reading position, causing a carry to the fifth wheel. By means of mechanical linkage, the fifth wheel moves from 0 to 1. The sum 13,884 is retained in the calculator.

Modern mechanical calculating machines also use wheels. Electric motors cause the wheels to turn at a rapid rate, yet the basic principle

behind modern calculators is the same as that used by Pascal in his machine. An important achievement has been made, namely, electric power is substituted for hand power. However, the drudgery of placing individual digits into the machine is still present.

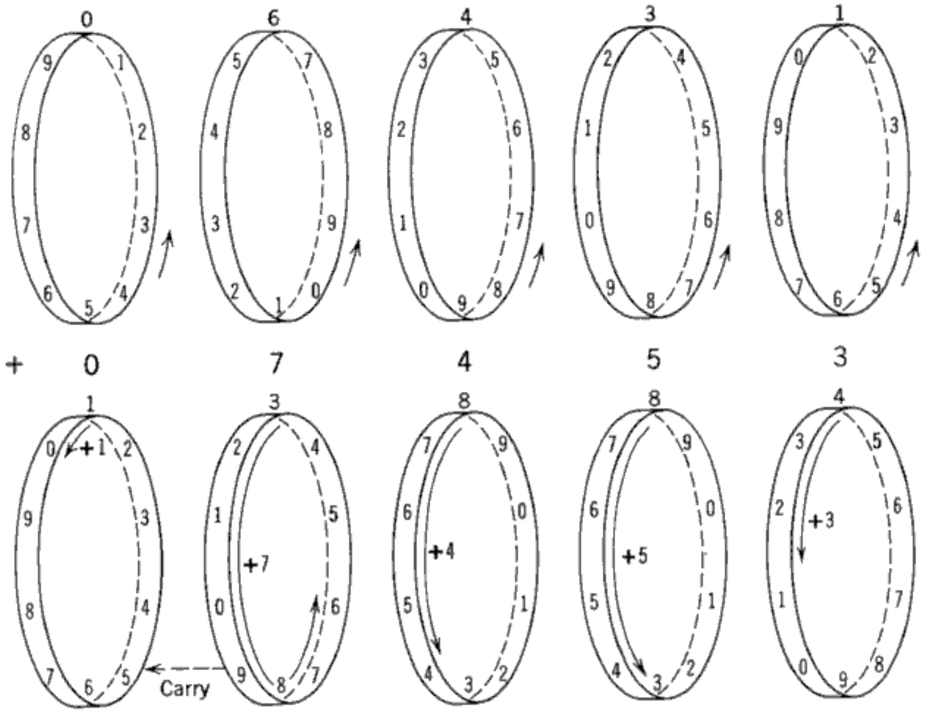


Fig. 3. Adding by means of wheel calculator.

ELECTRONIC CALCULATING MACHINES

To speed up the counting process, electronic devices may be substituted for mechanical wheels. By shifting electrons from one place to another for each unit of the digit, the same counting effect can be produced as by moving a bead at a time, or turning a wheel a division at a time. A diagrammatic representation of this idea is shown in Fig. 4. Instead of a column being represented by a wheel, it is represented by a circle of ten circuits shown as boxes in this figure. Originally the 0 box in each column is energized. Upon receipt of a bit of electric energy representing a unit, electrons flow from the energized box to the next box. Each successive bit of energy causes activation of the box next in line. As with the mechanical arrangement, the pulse which causes activation of the 0 box again, also

causes a carry pulse—a bit of electric energy—to be applied to the circle on the left.

In practice there is no need to put these "boxes" in a circle. It was done here only to show the analogy with mechanical wheels.

Why should a machine using moving electrons be so much faster than one using mechanical wheels? The answer to this question is very simple. The speed with which anything can be made to move depends on its weight, and the weight of an electron is practically nonexistent when

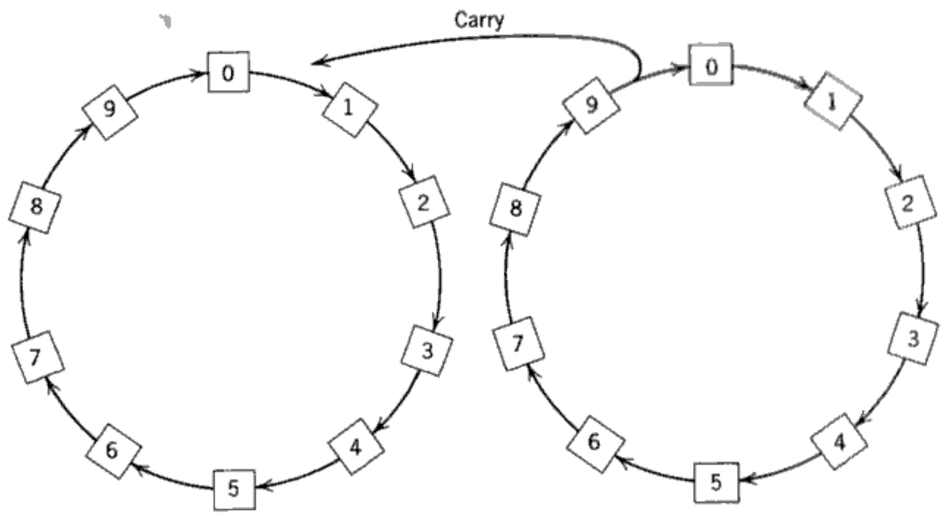


Fig. 4. Counting with electrons.

compared to any mechanical gadget. Thus, with very little energy applied, electrons can be made to travel at tremendous speeds. This fact enables us to build computers which can perform 500,000 additions in the time it takes a comptometer operator just to touch a key on her keyboard.

With machines of such lightning speed, it becomes ridiculous to have the operator feed it one problem at a time. Therefore, a logical control was built into the computer. This logical control enables the machine to accept a complete routine of instructions at one time, and then perform each instruction in the proper sequence. As a matter of fact, as will be seen later, the logical control enables the machine to accept generalized instructions which can cause it to perform more operations than are actually fed into the machine.

The tremendous speeds at which these machines can work, and the flexibility built into them due to the logical control, make modern electronic digital computers a thousandfold more powerful than mechanical calculators.

ANALOG VERSUS DIGITAL COMPUTERS

Even before the advent of the electronic digital computer, the analog computer was busy doing important jobs which could not be performed by desk calculators. For instance, instead of building a costly experimental model of an airplane and then running exhaustive wind-tunnel tests, the whole "experiment" can be performed by the analog computer. Instead of using an actual airplane, design data for an airplane are fed into the analog computer. Instead of using a wind tunnel, formulas representing the effects produced by varying flight conditions are fed into the computer. The output of the analog computer is usually a curve representing results for a complete set of conditions. Effectively, the tests are performed without any hardware. Upon completion of one "experiment," the computer is ready to perform a new "experiment."

What exactly is an analog computer? or, more fundamentally, an analog device? An analog device is any device whose operation is analogous to some physical quantity we want to measure. A thermometer is a simple analog device. It compares, or draws an analogy, between the expansion of mercury and temperature. A simple spring balance is another analog device. It compares the weight of an object with the force necessary to move a spring. Other examples are the oil-pressure gage in the car and the glass graduate used to prepare baby's formula.

If an analogy can be formed between the operation of a device and a mathematical process, this device can form the basis for an analog computer. Many physical phenomena may be used to construct analog computers. But since electrons are so mobile, most present-day analog computers perform mathematical operations with the aid of electronic circuits. Electronic circuits have been built which perform operations analogous to those of addition, subtraction, multiplication, division, and even higher mathematical operations such as integration. For instance to add 6 and 5, a 6-volt signal and a 5-volt signal are fed to an electronic adder. The circuit of the electronic adder is designed so that, under these circumstances an 11-volt signal is produced as output. Similar effects occur with the subtractor, multiplier, divider, and integrator.

Notice that in analog computers, measurement is always involved; in digital computers, numbers are used. Data in analog computers change smoothly; data in digital computers change in discrete steps. Analog computer results must be read off a scale. Results of digital computation are given by means of a representation for the digits of the number.

There is a limit to the accuracy obtainable from an analog computer. This is so because there is a limit to the accuracy with which one can read

a scale. The scale can be enlarged to increase the available accuracy. But even if this is done, the accuracy of the electronic circuits feeding the scale must be improved. Eventually the point is reached where it is difficult, if not impossible, to improve greatly the available accuracy.

However, the accuracy obtainable from a digital computer is theoretically unlimited. This is so because a separate circuit takes care of each column in a set of figures. Thus, 5 adders are needed to add a 5-digit number to another 5-digit number, and 50 adders are needed to add a 50-digit number to another 50-digit number. Theoretically, therefore, we can obtain answers accurate to any decimal place by merely providing the appropriate number of circuits.

FAST MORON VERSUS SLOW GENIUS

The digital computer is extremely fast, but it has the "brains" of a moron. Basically all the machine can do is add. If this is true, how can the computer perform problems in higher mathematics? The answer to this question is that for a respectable number of problems no matter how sophisticated they may appear, they can be shown to be related to addition.

Subtraction is the converse of addition. Addition of two digits is performed by first counting the units of one digit and then continuing to count the units of the second digit. Subtraction is performed in a similar way except that the counting for the second digit is in the opposite direction (see Fig. 5).

Multiplication is a rapid form of addition. For instance,

$$\begin{array}{r} 5 \\ \times 6 \\ \hline 30 \end{array} \quad \text{is equivalent to} \quad \begin{array}{r} 6 \\ 6 \\ 6 \\ 6 \\ 6 \\ \hline 30 \end{array}$$

Since division is the converse of multiplication, division may be considered to be an advanced form of subtraction.

Algebra is an extension of arithmetic processes. For instance, in order to find X in

$$2X = 8 + 4$$

both sides are divided by two, giving

$$X = \frac{8+4}{2} = \frac{12}{2} = 6$$

The answer is obtained by means of ordinary arithmetic.

Even the process of integration used in calculus can be shown to be equivalent to a series of additions. Integration is a means for finding the area included by a curve. Mathematicians have developed formulas and rules for performing integration. But finding the area included by a curve can be done in a much more simple manner.

For instance, to find the area of triangle ABC in Fig. 6, the length of the base (4) is multiplied by one-half the altitude (2). Another method of obtaining the same answer is to divide the triangle into four rectangles

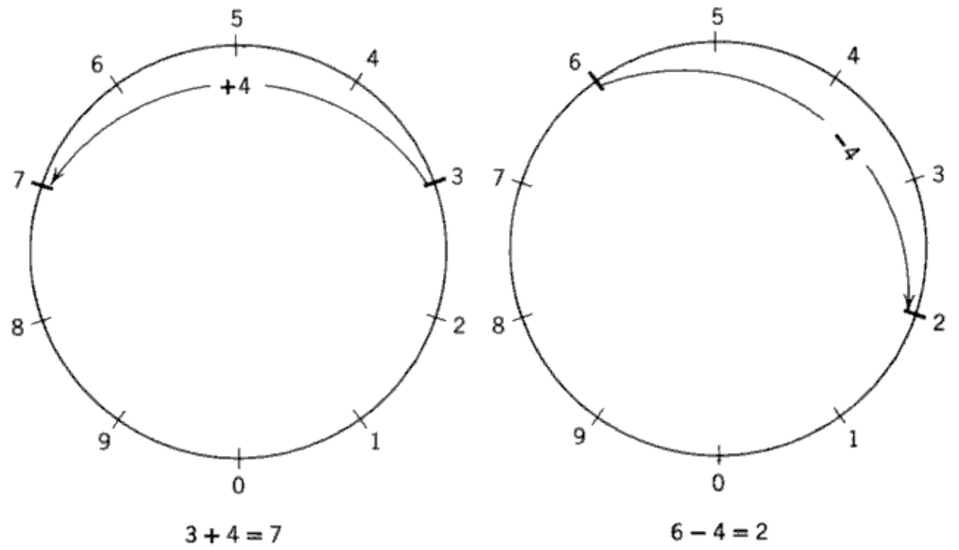


Fig. 5. Difference between addition and subtraction.

and then sum the areas of the rectangles. This gives the correct answer because the area outside of the triangle which each rectangle covers is just as large as the area inside the triangle which each rectangle does not cover (triangle DEF is equal to triangle EGH in Fig. 6). Dividing the triangle into many more rectangles and summing the areas provide the same answer.

Suppose the same process is performed on the circle of Fig. 7. If the circle is divided into three rectangles and the areas of the rectangles are summed, only the *approximate* area of the circle is obtained. This is so because area ABC is not exactly equal to area CDE, and the areas bounded by $EFGH$ and $E'F'G'H'$ are not counterbalanced by any other area. However, if the circle is divided into more and more rectangles (Figs. 7*b* and 7*c*), the sum of the rectangle areas gets closer and closer to the true area of the circle. By dividing the circle into very many rectangles

and summing their areas, an excellent approximation to the area of the circle may be obtained. This process can be used for finding the area included in any curve or sets of curves.

The mathematician—the genius—does not plod through all this work. He has developed formulas and rules which he uses similar to shorthand. With the use of this shorthand method he arrives at the correct answer

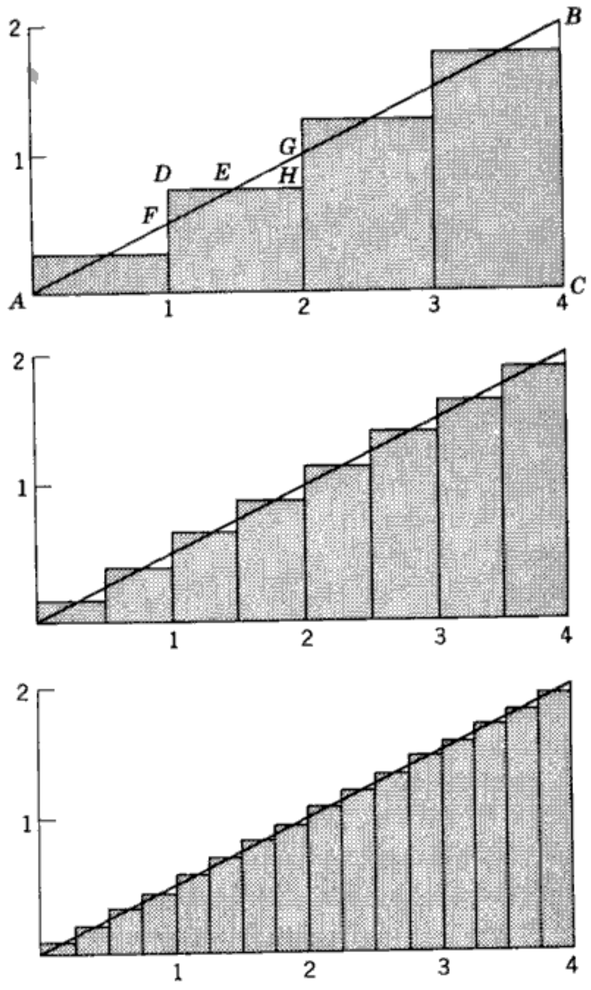


Fig. 6. Obtaining area of triangle.

quickly. The machine—the moron—does the problem the slow way by summing a large number of rectangles. But the machine is inherently so much faster that it arrives at the answer faster with the slow, stupid method than the mathematical genius with his supposedly short sophisticated method!

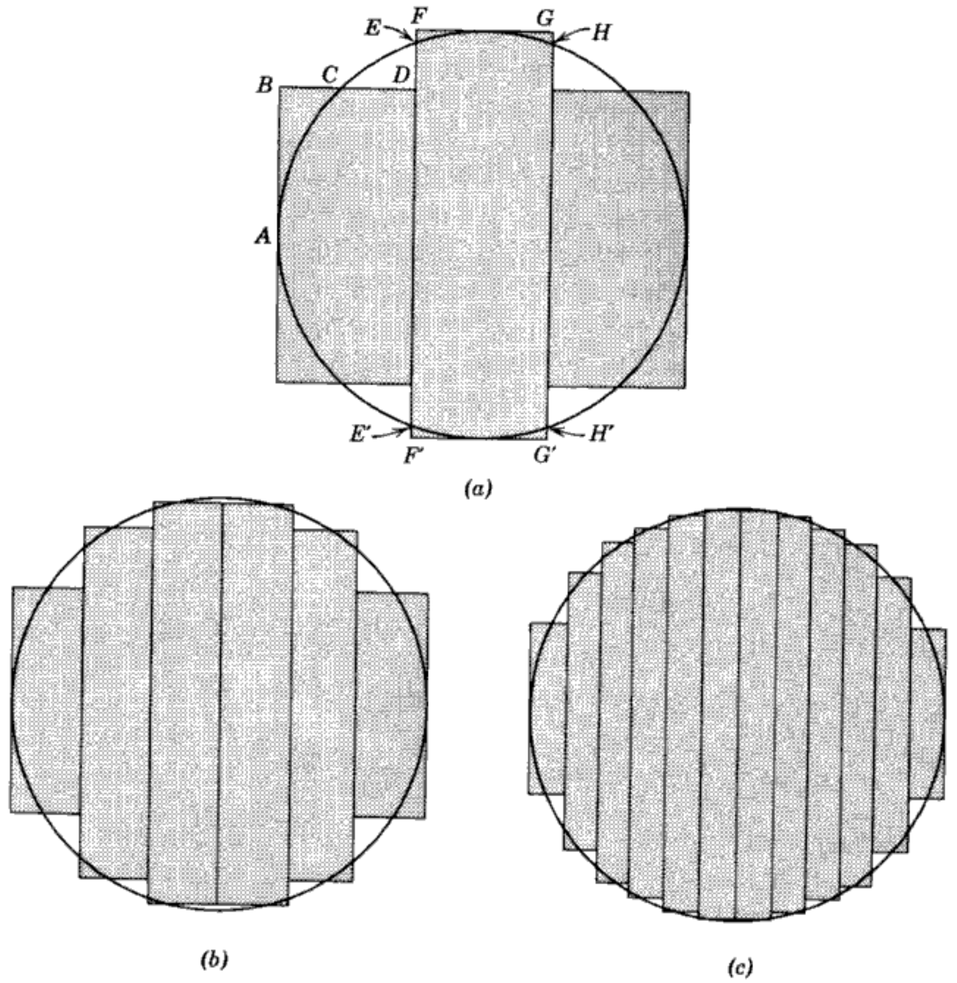


Fig. 7. Obtaining area of circle.

SIMPLE HARDWARE, COMPLICATED LOGIC

The computer, then, does only additions in a sequence determined by a logical control. This seems to imply that it is a simple machine. In a sense it is, since only a handful of different types of circuits are needed to build the most awesome machine in existence. What then produces the complication? The complication is produced by the many interconnections among these few basic circuits. The method of interconnection is called logic, and the few basic circuits are called logical elements.

Of course, you are wondering how the concept of logic fits into a discussion of computers. A digital computer processes information. In

that sense each part of it can be said to be communicating with or "speaking" to other parts of the machine. For this purpose the machine uses a "language." Whatever form this language may take, it must possess rules indicating how subjects are connected. These connectives are similar to connectives used in ordinary speech. Some of these grammatical connectives are "if, all, then, also, and, or not, only . . . ." These connectives are the cement of the language. They indicate the relationships among the different elements of a sentence. Because they are so important for indicating relationships, they are the fundamental building blocks of logic.

These connectives not only indicate the relationships among elements of a sentence but also among propositions in a logical syllogism. Take the following syllogism for instance:

If all men have two eyes,
and you are a man,
then you have two eyes.

What are the key words? The connectives "if," "and," "then." Take another syllogism:

Freedom is possible *only* in a democracy.
Russia is *not* a democracy.
There is *no* freedom in Russia.

The important connectives within the sentences, in this case, are "only," "not," "no." The key words which connect the three statements are understood to be "if" before the first sentence, "and" before the second sentence, and "then" before the third.

Now here is the important consideration. *All the possible connectives may be reduced to only three basic ones—"and", "or," and "not"*! This idea is discussed more fully in Chapter 4. Suffice it to say here that, if a circuit could be developed for each of the three logical connectives, we would have the necessary cement for the machine language. Then we only would need circuits to store the "propositions" before and after being applied through the connective circuits. These connective and storage circuits are called logical elements.

Logical elements may take on many forms. Originally, relays were used for this purpose. Early in the development of electronic computers, relays were replaced almost entirely by vacuum tubes. But a new era dawned when the magnetic core, the transistor, and many other solid-state devices were developed. These tiny components enabled manufacturers to reduce the size of electronic computers to the point where they became feasible for use in ordinary offices. In addition, the fact that only a handful of basic circuits are needed to construct a computer encouraged the development of plug-in packages. These plug-in packages simplified the maintenance problem tremendously.

FUNCTIONAL ORGANIZATION OF THE COMPUTER

To repeat, the logical elements built into the computer are simple. The big problem in understanding digital computers is the logic which ties the logical elements into a unit performing arithmetic and logical operations.

Fortunately, all computer operations can be grouped into five functional categories. The method in which these five functional categories are related to one another represents the functional organization of a digital computer. By studying the functional organization, a broad view of the computer is obtained, a view which will aid materially in the study of the detailed logic.

To gain an understanding of the functional organization, let us analyze what we do when confronted with a simple problem such as:

$$55 \times 35 + 67 \times 25 - \tfrac{24}{4}$$

To begin with, we pick up a pencil and record the problem on a sheet of paper. Next we decide which portion of the problem to do first. Let us assume we decide to do the first multiplication. We record the problem on another part of the paper and perform the calculation. As soon as the calculation is done, we store the product on another part of the paper. Next, we perform the second multiplication and store its product; then we do the division and store the quotient. Now we add the two products, and then subtract the quotient from the sum. The calculating steps in the procedure are summarized below.

(1)	(2)	(3)	(4)	(5)
55	67	$\frac{24}{4} = 6$	+1925	+3600
×35	×25		+1675	− 6
275	335		+3600	+3594
165	134			
+1925	+1675			

Notice, first, that we need a pencil, or a means of recording information —*input equipment*. Input equipment in computers takes on various forms such as modified typewriters which produce electric pulses when the keys are depressed; magnetic-tape units which convert information stored in the form of tiny, magnetized areas on tape into electric pulses; or punched

cards where information represented by holes punched in specific areas is converted into electric signals.

Second, the problem is recorded on paper. This is necessary because we can not do all parts of the problem at once. Therefore, we must have a *memory*—the paper—which retains the bits of information until we are ready to use them. In computers, memory is provided by magnetic drums which store many bits of information by magnetizing (or not magnetizing) tiny areas around their peripheries; by tanks filled with mercury through which sound waves representing the information are constantly being recirculated; by cathode-ray tubes which store information by charging tiny areas on the screen with electrons; by magnetic cores which store the information by being magnetized one way or another; by vacuum tubes, relays, and a host of other devices.

Third, we must decide which operation to perform first, and then choose the information needed for that operation. In the computer this is done by the *control* circuits, according to instructions placed in the computer by the operator. Incidentally, these instructions are usually placed into the machine through the input equipment.

Fourth, the rules of addition, subtraction, multiplication, and division must be applied to obtain the answers to the individual parts of the problem. In the computer, these operations are performed by the *arithmetic* unit.

Fifth, when the problem is done, the operator records this problem in a special place. If he is an accountant, he may put his answer in a ledger. This operation is called *output*. The *output* equipment in the computer is often similar to the input equipment since they both either transfer information from the "outside world" to the computer, or vice versa. Another type of output device is the printer which is used for the preparation of reports, bills, and other forms.

To summarize, the five major functional units of a digital computer are:

1. INPUT—To insert outside information into the machine.
2. MEMORY—To store information and make it available at the appropriate time.
3. ARITHMETIC—To perform the calculations.
4. OUTPUT—To remove information from the machine to the outside.
5. CONTROL—To cause all parts of the computer to act as a team.

Figure 8 shows how the five functional units of the computer act together. A complete set of instructions plus data is usually fed through the input equipment to the memory where it is stored. Each instruction is then fed to the control unit. The control unit interprets the instructions and issues commands (dashed lines on figure) to the other functional units

to cause operations to be performed on the data. Arithmetic operations are performed in the arithmetic unit, and the results are then fed back to the memory. Information may be fed from either the arithmetic unit or the memory through the output equipment to the outside world.

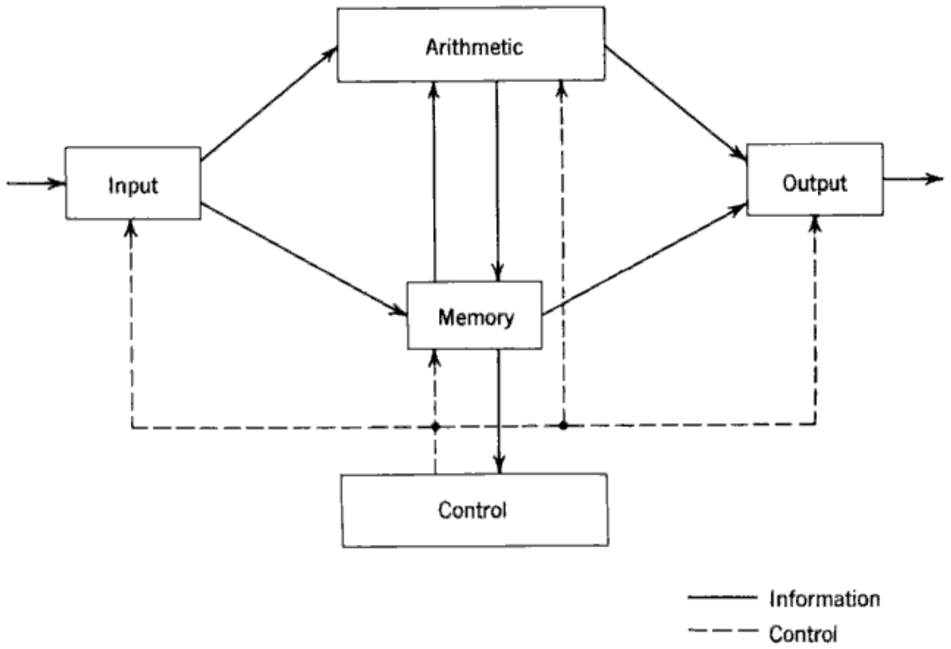


Fig. 8. Functional organization of a digital computer.

MACHINE LANGUAGE AND PROGRAMMING

The five units of the computer must communicate with each other. This they do by means of a machine language which uses a code composed of combinations of electric pulses. These pulse combinations are usually represented by ZEROS and ONES where the ONE may be a plus pulse and the ZERO a minus pulse; or the ONE may be a pulse and the ZERO a no-pulse. One possible code for the 10-decimal digits is

0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Numbers are communicated between one unit and another by means of these ONE–ZERO or pulse–no-pulse combinations. The input has the

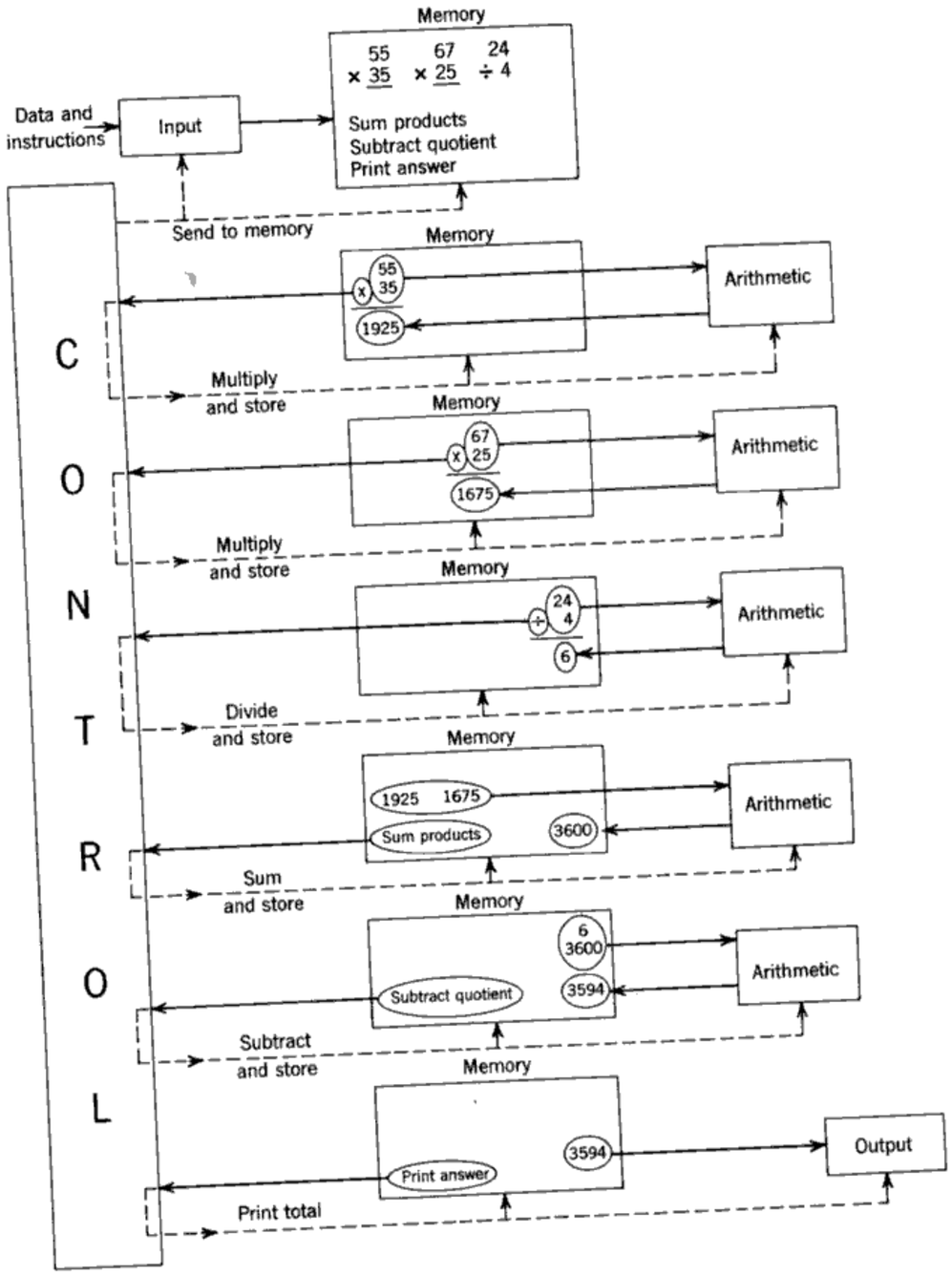


Fig. 9. Execution of simple problem by computer.

additional job of converting information fed in by the operator into machine language. In other words, it translates from our language into the pulse–no-pulse combinations understandable to the computer. The output has the additional job of converting the pulse–no-pulse combinations into a form understandable to us, such as a printed report.

In addition to data, the input translates instructions needed to perform a given problem into pulse-no-pulse combinations. These instructions are originated by a programmer whose job it is to translate a scientific or business problem presented in human language into a sequence of detailed instructions understandable to the machine.

What is involved in programming and the steps the computer goes through when it follows the program are shown in a very general way in Fig. 9. The data and the program of instructions necessary to perform the problem previously given are placed in the input system. During the first part of the execution of the problem, the control unit sends signals to the input and to the memory for the entire program to be stored in the memory. Then the control accepts each instruction in turn from the memory, interprets it, and sends the proper control signals to the other units for execution of these instructions. First it sends 55 to 35 to the arithmetic unit, and the multiplication instruction to the control for interpretation and execution. After the multiplication, the product is fed back to a designated spot in memory. Then the computer, following its program, repeats the above routine for the 67 and 25 factors and stores the product in another spot in memory. After this, the computer performs the division, $^{24}/_{4}$, and stores the quotient. Once these portions of the problem are completed, the two products are fed to the arithmetic unit and summed. Then the quotient is subtracted from the sum in the arithmetic unit and the grand total is stored in memory. The last instruction may call for a print-out of the answer. In this case, the control sends signals to memory and output for the answer to be printed.

It is easy to see that there are many detailed steps necessary to perform even a very simple problem. In some of the more complicated problems, the number of instructions necessary to specify the problem completely may take up about 200 pages of instructions!

Programming is tedious work. For instance, a straightforward way to cause the machine to sum 10,000 numbers would be to instruct the machine as follows:

Add the 1st number to the arithmetic contents (0 at the beginning).
Add the 2nd number to the arithmetic contents.
Add the 3rd number to the arithmetic contents.
Add the 4th number to the arithmetic contents, etc.

To accomplish this task the programmer must write 10,000 addition instructions!

But ways have been found whereby the programmer does *not* have to specify *every* instruction he wants the computer to execute. This is done by writing a generalized instruction sequence which is repeated over and over, each time with a slight modification. To see how this is done, first note that each location in memory is numbered and that the program for adding the 10,000 numbers really reads something like this:

Add the number in memory location No. 1 to the sum stored in arithmetic.

Add the number in memory location No. 2 to the sum stored in arithmetic.

Add the number in memory location No. 3 to the sum stored in arithmetic.

Add the number in memory location No. 4 to the sum stored in arithmetic, etc.

The only difference among these instruction is the number of the memory location referred to. Therefore, the routine could be written as follows:

Add the number in memory location No. 1 to the sum stored in arithmetic.

Add 1 to the memory location referred to in the addition instruction.

Repeat routine.

Of course, this is not the complete program. But enough is given to show the general idea.

SELF-CHECKING MACHINE

So far we have discovered that the computer possesses the following humanlike traits. The digital computer takes instructions from its environment (programmer), performs these instructions, and then communicates the results back to its environment. The computer is often called upon to exercise judgment in determining which of two possible alternatives (sets of instructions) to follow. The computer can do more than it is specifically asked to do: it can follow a generalized program.

Another human trait the machine possesses is the ability to check its own work. Although this sounds impossible, let us investigate how this is done. We saw that the computer has a language of its own based on pulse–no-pulse or ONE–ZERO combinations, and logical circuits which speak to each other in this language. This language possesses definite

rules just as English does. Suppose we received the following garbled English message over the radio:

"— — — time — all good men — — — — aid — — country."

Each blank space represents a missing word. Would we find it difficult to fill in these blank spaces? Of course not. Why not? Because of what we know about the English language. The language follows rules that relate certain words to other words, and anyone who knows these rules—who can speak the language—can check for mistakes in transmission. Here is the corrected sentence:

"Now is the time for all good men to come to the aid of their country."

Notice that we do not need the underlined words to convey meaning. These words are redundant. But because they bear a grammatical relationship to the other words in the sentence, they make it easy to check the sentence. They add readability, and improve the reliability of communication.

English word spellings have many redundancies which affect the reliability of the language in the same way. The garbled message could have been received in this form:

"N–w is t–e tim– f–r a–l g––d m–n to c–m– to the ai– of t–––r c–––try,"

Again we can reconstruct the original sentence because we know the relationships among the redundant letters and the other letters of the message.

The same principle of redundancy can be used in the machine language of ONE's and ZERO's to enable messages to be checked. If instead of writing each digit as a combination of four bits, we were to write it as a combination of five bits, where the fifth bit is related to the other bits according to some rule, the digits could be checked. For instance, the code for the digits presented before could be modified thus:

0	1 0000	5	1 0101
1	0 0001	6	1 0110
2	0 0010	7	0 0111
3	1 0011	8	0 1000
4	0 0100	9	1 1001

In this case, the leftmost bit is related to the others in that it makes the sum of all the ONE bits odd for every digit. A circuit which checks for this redundancy rule is all that is necessary to test this language.

In fact, it is easier for the machine to check for a mistake in transmission than it is for a person. English is full of idiomatic expressions,

duplication of meanings, grammatical exceptions, and many other vagaries which the machine does not have to contend with. The machine has its rules and they are rigorously followed.

Duplication is another method used to check work. When we add up a column of figures and we want to be sure the sum is correct, we add the figures again. If the two sums are equal, we assume the original sum is correct. If the sums are unequal, we search for a mistake. A machine can do the same thing. The problem is fed to two circuits each of which performs the same operation. Then the results are compared. If they are alike, the machine proceeds with the following instructions. If they are not the same, the machine stops and indicates an error.

Because the computer operates according to the rules of logic, these rules of logic may be used in the diagnosis of the fault. This is done similar to the way the doctor uses his knowledge of the human body to help him determine the source of a person's ailment. Like the doctor, too, the computer trouble shooter can perform tests and then note reactions. To a limited extent, the human body fights off disease without outside help by increasing the number of white blood cells, for instance. Some computers can perform a similar function. Not only do they find where the error occurred but they also correct for the error!

SUMMARY

1. The modern electronic digital computer is an extremely fast adding machine, built on logical principles. It can perform problems in higher mathematics faster than a human.

2. There are five major functional units to the computer: input, memory, arithmetic, output, and control. They communicate with each other by means of a language composed of coded combinations of ONE's and ZERO's.

3. To execute problems on the computer, the programmer breaks the mathematical problems down into instructions the computer can perform. The input system converts these instructions into the language of the machine.

4. The following are some of the advantages that electronic digital computers have over other mathematical machines:

a. *Speed:* The electronic machine is faster than the mechanical one because electrons travel much faster than any mechanical object.

b. *Accuracy:* The digital computer is capable of much greater accuracy than the analog computer. Another decimal place of accuracy can be added to a digital machine by merely inserting an extra circuit to take care of this decimal place.

c. *Reliability:* Machine reliability is the result of the strict logic on which the computer is based; circuit standardization; and the self-checking features of the machine.

d. *Flexibility:* This is obtained by the logical control; the use of many types of input and output equipment to communicate between the "outside world" and the computer; and the memory.

section I

ARITHMETIC AND LOGIC

There are many types of digital computers. There are special-purpose computers which are built to do a specific job, and general-purpose computers which are built to do many jobs. There are fixed-program machines which sequence the computer through the same operations over and over, and computers whose programs can be modified. There are computers built for business applications, others built for scientific applications, and still others for control of physical systems. But all the different types of computers are based on the same principles—the principles of arithmetic and logic.

In this first section of the book we discuss the principles of arithmetic and logic. In Section II these principles are used to develop building blocks; in Section III the building blocks are put together to form a specimen computer.

Section I is divided into three chapters. In the first chapter we show that number languages are similar to word languages and we develop several number languages. In the second chapter we discuss rules for performing arithmetic in several of the number languages. Since practically any calculating job can be broken down into a series of arithmetic operations, the rules of arithmetic provide a good foundation upon which to build our knowledge of computers. In the third chapter the fundamental elements needed to mechanize the rules of arithmetic are presented in the rules of logic.

CHAPTER 2

Word and number languages

Each man has his impressions of the world around him. These impressions are definitely unalike. How can anyone convey his inward impressions to another in a manner such that the second person can understand him? This can be done only by a set of symbols the meanings of which have been agreed upon by everybody. A set of such meaningful symbols put together according to commonly agreed rules is a language.

The language of ancient man was composed of pictures. The pictures were symbols and they were understood by all. But a picture can convey only subjective feeling; it is unsuitable for precise thinking.

The early picture languages developed into the ideographs of the Chinese and Japanese. The ideographs are a little less subjective, but still not quite objective enough.

Now most languages use words formed of letters. These words representing simple ideas are linked together with other words to form sentences representing more complicated ideas. The symbols used in modern languages are more abstract than the ideographs, and thus are better tools for orderly thought. But even in modern languages, words receive connotations which tarnish their usefulness for rigorous thought. This is the reason mathematicians and scientists like to use mathematical symbols for their more profound thinking.

We need language not only to communicate with our fellow men but also to communicate with ourselves, that is, to think. We can do some thinking without language. We can think of being hungry or of going fishing. But it is impossible for us to do abstract thinking without language. How would we think, "I want to go hunting next week," without a language. How would we think of "blue," "thin," "good," "better," "a mile"? We need to manipulate symbols in order to do any but the most primitive type of thinking.

LANGUAGE

Redundancy and Dependability

What are the rules of language? We usually have symbols tied together in a word. Each word consists of a different combination of symbols. The relationships among words are expressed by means of the logical rules of grammar. In order to understand a language, we must know the symbols, the rules of construction of words, and the grammar.

Because of the many relationships expressed by means of rules, not every symbol in a word, or every word in a sentence, conveys meaning or gives information. Some symbols are there only to take care of the rules. For instance, in the previous sentence why do we need to use four words, "to take care of," instead of one word, "care"? In the word "grammar," why do we need two m's? These extra symbols are called redundancies.

There is a relationship between redundancy and probability. Which would you expect to find more often in an English sentence, a t or an x? How often is t followed by h? What letter will almost inevitably follow *q*? Similarly, there are frequently occurring word combinations such as "in order," "and the," "there is," "because of," and "expect to." Because we have a good statistical idea of what symbols to expect in the above cases, these examples contain redundancies.

Redundancy is defined as the use of more symbols to encode a message than is strictly necessary.

These extra symbols, however, increase the dependability of a language. Because language rules tell us to some extent what to expect, garbled messages may be deciphered more readily. Redundancy thus makes a language easier to understand.

Brevity and Power

If redundancy increases the dependability of a language, it also makes language more cumbersome. To increase the informational efficiency of a language, we must remove redundancies—make the language tightly packed. This is the reason mathematicians use symbols instead of words to present their ideas.

There are many ways to abbreviate. One way is to designate a letter or a short group of letters to represent a larger combination of letters, for example, as in shorthand used by secretaries. St., Dr., and Mr. are other examples. Abbreviations enable us to transmit the message faster.

Another form of abbreviation—although it is never called that—is the use of one sophisticated word instead of a group of commonly known

words. Some examples of this are "ambiguous" instead of "not clearly understood"; "sadist" instead of "one who likes to inflict pain on others"; "atheist" instead of "one who does not believe in God." The word "God" represents a large variety of ideas. We have many such meaningful words: democracy, thermodynamics, logic, etc. There is little redundancy in these words. We can employ them to convey information compactly. They are powerful words, used by great thinkers, but difficult to understand.

Probably the most powerful way of expressing ideas is by means of mathematical symbols. Remember the age problem we had in high school?

> John is 15 years old.
> Five years ago Jack was twice as old as John.
> How old is Jack now?

To solve this problem we designated John's present age by X and Jack's present age by Y; then we expressed the whole problem compactly thus:

$$X = 15$$
$$Y = 2(X - 5) + 5$$

Once the problem was expressed in mathematical form, we did not care what the symbols represented. We operated with them according to definite rules. In this case we substituted the value for X into the second equation, and performed the indicated arithmetic, thus:

$$Y = 2(X - 5) + 5 = 2(15 - 5) + 5 = 2 \times 10 + 5 = 20 + 5 = 25$$

After the answer was obtained, we translated it back into English. Remembering that Y stood for Jack's present age, we drew the conclusion that Jack's present age was 25.

Mathematical symbols decrease the redundancy and greatly increase the efficiency of our thinking. But both the sender and the receiver of the message must completely understand the meaning of the symbols. The more the symbol represents, the harder it is to be sure that the meaning of the symbol will get across. Therefore we say that brevity increases the efficiency of communication at the expense of reliability.

POSITIONAL NUMBER SYSTEMS

Words and Numbers

We have been talking for the most part about languages based on words. We do have languages based on numbers. Word languages and number languages are similar in many respects. Word languages are built from letters which are combined into words. Relationships among these words

are expressed by means of the rules of grammar. Number languages are built from digits which are combined into numbers; relationships among these numbers are expressed by means of the rules of arithmetic (+, −, ×, ÷).

Some words are related to each other by rules of spelling or of grammar. But many words are unrelated. All numbers, however, are related by a scheme of counting.

Because number languages are built on a firmer foundation, they are much more reliable for reasoning than are word languages. "Twenty percent of the people are thieves means more than "some people are thieves"; or "Joe made a grade of 90% on the examination" expresses more than "Joe is a good student." However, an attempt has been made to place word language on just as rigorous a basis by means of the algebra of logic. The algebra of logic is discussed at greater length in Chapter 4.

Table 1 summarizes the similarities between word language and number language.

TABLE 1
Similarities Between Word and Number Languages

Item Compared	Word Language	Number Language
Fundamental unit	letter	digit
Coded combination	word	number
Rules for forming combinations	spelling and grammar	counting
Logic of language	grammar	arithmetic
Shorthand	mathematics	mathematics

What is Zero?

The decimal number system is in most common use today. To appreciate the decimal number system, let us look at ancient number systems, some of which are shown in Table 2. Note that in all these systems, numbers are built up in an *additive* manner: symbols having different weights are combined to produce a sum representing the number. For instance, in the Roman system symbols I, V, X, L, C, D, M representing 1, 5, 10, 50, 100, 500, and 1000 respectively are combined to produce:

$$\text{LXVIII} = 50 + 10 + 5 + 1 + 1 + 1 = 68$$
$$\text{MCXII} = 1000 + 100 + 10 + 1 + 1 = 1112$$

In some cases units are not added but subtracted, as in the following:

$$\text{CXLIV} = 100 - 10 + 50 - 1 + 5 = 144$$
$$\text{CMXCII} = -100 + 1000 - 10 + 100 + 1 + 1 = 992$$

It is very difficult to do arithmetic with these ancient number systems.

TABLE 2
Ancient Number Systems

Decimal	Roman	Mayan	Hieroglyphics (3400 BC)
1	I	•	∧
2	II	• •	∧∧
3	III	• • •	∧∧∧
4	IV	• • • •	∧∧∧∧
5	V	—	∧∧∧∧∧
6	VI	•—	∧∧∧∧∧∧
7	VII	••—	∧∧∧∧∧∧∧
8	VIII	•••—	∧∧∧∧∧∧∧∧
9	IX	••••—	∧∧∧∧∧∧∧∧∧
10	X	=	⋀
11	XI	•=	⋀∧
12	XII	••=	⋀∧∧
13	XIII	•••=	⋀∧∧∧
14	XIV	••••=	⋀∧∧∧∧
15	XV	≡	⋀∧∧∧∧∧
16	XVI	•≡	⋀∧∧∧∧∧∧
17	XVII	••≡	⋀∧∧∧∧∧∧∧
18	XVIII	•••≡	⋀∧∧∧∧∧∧∧∧
19	XIX	••••≡	⋀∧∧∧∧∧∧∧∧∧
20	XX		⋀⋀
40	XL		⋀⋀⋀⋀
100	C		C

Can you imagine trying to multiply CCCXXXIX by XLVII? These number systems are unwieldy because there is no simple way to set down the rules for counting. It is true that each symbol represents a certain value, but its relationship to the whole number depends upon the symbol before and the symbol after it. Furthermore, symbols in any relative position do not represent any particular range of numbers.

By contrast, the decimal system is simple to use. Any schoolboy can multiply 528 × 44 and obtain 23232 by following simple rules, thus

$$\begin{array}{r} 528 \\ \times 44 \\ \hline 2112 \\ 2112 \\ \hline 23232 \end{array}$$

Arithmetic in the decimal system is simplified because each digit represents a number between 0 and 9 or a 10's multiple of that number; the *position* of the digit in the number determines which 10's multiple the digit represents. In other words, the multiplicand in the above example is

$$528 = 500 + 20 + 8$$

the multiplier is

$$44 = 40 + 4$$

and the product is

$$23232 = 20000 + 3000 + 200 + 30 + 2$$

Because numbers in this positional enumeration system are put together by means of simple rules, the rules of arithmetic are made simple.

Why could not the ancients use positional notation? The answer is that they did not know about zero. Zero means more than nothing. Zero is a digit which indicates position without signifying value. The number 4000 is distinguished from the number 4 by the number of zeros. In the first case, the digit 4 is in the fourth column (from the right) and in the second case the digit 4 is in the first column. These facts are apparent when we use the following notation:

$$4. = 4 \times 10^0$$
$$4000. = 4 \times 10^3$$

The exponent indicates the number of zeros to the left of the decimal point.

The zero can be used to distinguish among fractional numbers too. The difference between .4 and .00004 is that in the first case the digit 4 is in the first column to the right of the decimal point, whereas in the second case the digit 4 is in the fifth column to the right of the decimal point. These facts are summarized in the following notation:

$$.4 = 4 \times 10^{-1}$$
$$.00004 = 4 \times 10^{-5}$$

The minus exponent indicates that the zeros are to the right of the decimal point.

Numbers in Exponential Form

As we have seen, the number 23232 can be expressed as

$$20000 + 3000 + 200 + 30 + 2$$

Instead of writing down all the zeros, the same numbers may be expressed in the following manner:

$$2 \times 10^4 + 3 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 2 \times 10^0$$

For the sake of consistency, the rightmost digit is considered to be multiplied by 10^0 which is equal to 1.

Note that in this positional enumeration system the exponent represents the number of the column in which the associated digit occurs, if we count columns from right to left and label the first column zero. We call the leftmost digit the Most Significant Digit (MSD) because a change in this digit has the greatest effect upon the value of the total number. We call the rightmost digit the Least Significant Digit (LSD) because a change in this digit has the least effect upon the value of the total number.

The same idea of positional enumeration can be applied to decimal fractions. The decimal fraction 0.6217453 can be expressed as

$$6 \times 10^{-1} + 2 \times 10^{-2} + 1 \times 10^{-3} + 7 \times 10^{-4} + 4 \times 10^{-5} + 5 \times 10^{-6} + 3 \times 10^{-7}$$

The minus sign in the exponent indicates that a multiple of one-tenth rather than of 10 is taken. The leftmost digit (6) is still the MSD and rightmost digit (3) is still the LSD.

Counting Process

Let us review the decimal counting process (Table 3). We start with each column holding a zero. In the first column from the right (column 0) we go through the sequence of digits from 1 to 9 and back to 0. As we return to 0, we sequence the digit in column 1 from 0 to 1. We follow through another cycle of ten digits in column 0 and, as we return to 0 again, we sequence the digit in column 1 from 1 to 2. We go through this cycle ten times or until the digit in column 1 is switched from 9 to 0. At this point we change the digit in column 2 from 0 to 1. This process is continued *ad infinitum*. This counting process can be performed by a group of rotating wheels. The wheel at the right makes a complete revolution for each ten digits. Each of the other wheels rotates at one-tenth the speed of the wheel to its right.

Because there are ten digits in this system of counting, each column produces a multiple of ten. We recognize this fact when we speak of the

TABLE 3

Decimal Counting

Columns:	3	2	1	0
Weight:	10^3	10^2	10^1	10^0
				0
				1
				2
				3
				.
				.
				.
				8
				9
			1	0
			1	1
			1	2
			1	3
			.	.
			.	.
			.	.
			1	8
			1	9
			2	0
			2	1
			2	2
			2	3
			.	.
			.	.
			.	.
			2	8
			2	9
			3	0
			3	1
			3	2
			.	.
			.	.
			9	9
		1	0	0
		1	0	1
		1	0	2
		.	.	.
		.	.	.
		1	0	9
		1	1	0
		.	.	.
		.	.	.
		1	9	9
		2	0	0
		.	.	.
		.	.	.
		9	9	9
	1	0	0	0

units, tens, hundreds, and thousands columns. Because we use ten digits in the decimal system, we say that it has a *base* or *radix* of 10. The base is defined as the number of digits used in a positional enumeration system.

Theoretically, we use an infinite number of columns, and we can go on counting forever. But for any physical device, there is always a finite number of columns used. The odometer of a car, for instance, has five columns (not including the one representing tenths of miles). This means that the largest number that can be represented by the ordinary odometer is 99999—one less than 100000.

What does this fact mean in practical terms? It means that if the odometer in a car reads 00500, we could not know for certain whether the car has traveled 500 or 100,500 or 200,500 miles. We say that the odometer reads numbers in modulus 100000, or 10^5 form. If it has 20 columns, it reads numbers in modulus 10^{20} form.

The modulus is a standard of measurement. It is one more than the last possible number that can be read from a practical counter. It is not the last number, but the number after it because we normally start counting with one and not with zero. If in the odometer the number 0 were used to represent the first mile, the reading of 99,999 would represent the 100,000th mile. As a matter of fact for most computer applications we do assume that zero is the first number. Reference to this point will be made later in this book.

Several Positional Number Systems

The almost universal use of the base of 10 for numbering developed only because we happen to possess ten fingers. But many bases are possible. Numbers in some of the bases used in computers are shown in Table 4.

The duodecimal system with a base of 12 consists of digits 0 to 9 of the decimal system plus two more—*A* and *B* for instance—representing the digits 10 and 11. In this system, whenever a digit in a column is sequenced from digit *B* to digit 0, the digit in the column to its left is sequenced to its next digit.

The digit in each column represents a multiple of 12; its position determines which multiple it is. For instance, 1*AB*6 is equivalent to

$$1 \times (12)^3 + A \times (12)^2 + B \times (12)^1 + 6 \times (12)^0$$
$$= 1 \times 1728 + 10 \times 144 + 11 \times 12 + 6 \times 1 = 3462$$

Some people believe that arithmetic is simpler in the duodecimal system. They base their contention on the fact that 10 is divisible by only two digits besides 1, i.e., 2 and 5; whereas 12 is divisible by four digits, i.e., 2, 3, 4, and 6.

The hexadecimal system has a base of 16. It consists of the ten decimal digits plus six additional symbols—*A*, *B*, *C*, *D*, *E*, and *F*—to represent the digits 10, 11, 12, 13, 14, and 15. Just as in the decimal system, a digit is worth 10 times more than a digit to its right, and in the duodecimal

TABLE 4

Positional Number Systems

Decimal 10	Hexadecimal 16	Duodecimal 12	Octal 8	Ternary 3	Binary 2
0	0	0	0	0	0
1	1	1	1	1	1
2	2	2	2	2	10
3	3	3	3	10	11
4	4	4	4	11	100
5	5	5	5	12	101
6	6	6	6	20	110
7	7	7	7	21	111
8	8	8	10	22	1000
9	9	9	11	100	1001
10	*A*	*A*	12	101	1010
11	*B*	*B*	13	102	1011
12	*C*	10	14	110	1100
13	*D*	11	15	111	1101
14	*E*	12	16	112	1110
15	*F*	13	17	120	1111
16	10	14	20	121	10000
17	11	15	21	122	10001
18	12	16	22	200	10010
19	13	17	23	201	10011
20	14	18	24	202	10100
40	28	34	50	1111	101000
100	64	84	144	10201	1100100

system a digit is worth 12 times more than a digit to its right, so in the hexadecimal system a digit is worth 16 times more than a digit to its right. Just as in the decimal system the placement of a zero to the right of the LSD multiplies the whole number by 10, and in the duodecimal system the placement of a zero to the right of the LSD multiplies the whole number by 12, so in the hexadecimal system the placement of a zero to the right of the LSD multiplies the whole number by 16.

Just as we may choose bases greater than 10, so may we choose bases less than 10. The octal system with a base of 8, the ternary system with a base of 3, and the binary system with a base of 2 are some commonly occurring schemes.

In the ternary system, only three digits—0, 1, and 2—are used. A number such as 210112 represents

$$\begin{aligned} &2 \times 3^5 + 1 \times 3^4 + 0 \times 3^3 + 1 \times 3^2 + 1 \times 3^1 + 2 \times 3^0 \\ &= 486 + 81 + 0 + 9 + 3 + 2 \\ &= 581 \end{aligned}$$

The digit in each column is worth three times more than a digit to its right.

In the binary system there are only two digits, 0 and 1. A number such as 100101011 is equivalent to

$$\begin{aligned} &1 \times 2^8 + 0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 \\ &\qquad + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 256 + 0 + 0 + 32 + 0 + 8 + 0 + 2 + 1 \\ &= 299 \end{aligned}$$

A digit in each column is worth twice as much as a digit in a column to its right. Evaluation in binary is much simpler than with any other base because no digit higher than 1 is used, and therefore no multiplication is needed.

The two digits of the binary system are called bits, which is a contraction of *bi*nary digi*ts*.

Advantages of Binary Number System

It is obvious that it takes many more digits to express a number in binary than it does in decimal. As was shown before, the binary 100101011 is equivalent to decimal 299. In this case, nine bits are needed for the binary representation as contrasted with three digits for the decimal representation. But this disadvantage is outweighed by the following advantages of the binary number system:

1. The binary ONE and ZERO can represent the YES and NO of formal logic.
2. Binary numbers are easy to mechanize reliably.
3. Binary arithmetic is very simple

Each of these advantages is discussed below.

Binary ONE and ZERO can represent the YES and NO of formal logic. In formal logic all propositions are either true or false; there is no middle ground. For instance, the following statement,

> Only animals which stand upright are human

implies that all animals may be divided into two mutually exclusive classes: humans and nonhumans. For all animals which stand upright we may draw the conclusion that they are human. This may be put in the form of a syllogism such as the following:

> Only animals which stand upright are human.
> Jack is an animal which stands upright.
> Therefore, Jack is human.

For all animals which do not stand upright, we may draw the conclusion that they are not human. This may be put in the form of a syllogism such as the following:

> Only animals which stand upright are human.
> The cat does not stand upright.
> Therefore, the cat is not human.

We can draw definite conclusions from our original premises because each premise must be true or false. If Jack is a baby who sometimes walks upright and sometimes crawls, we cannot draw *any* conclusion from the premises. For the sake of logical rigor, all animals *must* be divided into two mutually exclusive classes. Because each premise is either true or false, formal logic is often referred to as bivalued logic.

Of course, not everything can be specified in black and white terms. Many qualities are in between. Could we judge a man to be all evil or all good? Could we say at any time whether we are entirely happy or entirely unhappy? When is a piece of wood thin and when is it thick? When is a man bald and when is he not bald?

The answer is that these words are ambiguous. We cannot use them in formal logic unless we *define* them so that all objects referred to are separated into two mutually exclusive classes. We would set up a minimum list of characteristics necessary for a person to possess in order to be called "good." If he possessed anything less than these characteristics, he would be called "not good." We would define a man to be bald if at least 40% of the total area of the head as measured from a horizontal circumference drawn just above the ears was not covered with hair. This definition would divide all people into two mutually exclusive classes: the bald and the not bald.

The examples given are not perfectly restrictive definitions. But they serve to illustrate that the more rigorously we define words in bivalued logic, the more rigorous are the conclusions—the more definitely can we arrive at a "yes" or "no" answer. The binary ONE may be used to represent YES; and the binary ZERO may be used to represent NO. The implication of this statement are far-reaching, as will be seen later in the book.

Binary numbers are easy to mechanize reliably. It is much easier to develop a circuit which has only two stable states than it is to develop a circuit with ten stable states. For instance, to cause a vacuum-tube circuit to have ten stable states, we must design it so that it can conduct at ten distinct current levels. We must design it so that under steady conditions the current does not change enough to make the circuit change levels. We must design it to have equally spaced stable states; in other words, it should take just as much energy to go from one state to an adjacent state as it would to go from another state to an adjacent state.

We have no such problems designing 2-stable-state devices. Some bistable devices which have been used to store ONE's and ZERO's are relays which open and close; gas-discharge tubes which conduct or are cut off; magnetic materials which magnetize in one direction or the other; teletype tape in which holes represent ONE's and no holes represent ZERO's; and flip-flops which have one leg conducting and the other leg cut off, or vice versa. In each of these devices the two states are distinct, and it is easy to flip from one to the other without ambiguity.

Three-stable-state devices have been developed. Perhaps when these devices are made more reliable, a three-valued logic may be developed to enable us to design computers using the ternary system.

Binary arithmetic is very simple. To perform decimal arithmetic, we must first memorize addition and multiplication tables. We must do this also in the binary system. These tables are extremely easy, as shown in Tables 5*a* and 5*b*. The sum of 0 and 0 is 0; of 0 and 1, 1; and of 1 and 1, 0 with a 1 to carry. The multiplication of all 2-bit combinations

TABLE 5*a*
Addition Table

Augend / Addend	0	1
0	0	1
1	0	1 0 (carry)

TABLE 5*b*
Multiplication Table

Multiplicand / Multiplier	0	1
0	0	0
1	0	1

except 1 times 1 produces a 0; 1 times 1 produces 1. Can arithmetic rules be any simpler? The simple rules of binary arithmetic are reflected in the simple circuitry needed to accomplish this arithmetic electronically.

TRANSLATION BETWEEN NUMBER SYSTEMS

Decimal-to-Binary Translation

The following are some methods by which numbers in the decimal system may be translated into numbers in the binary system:

1. Dividing successively by the base.
2. Multiplying successively by the base.
3. Using meaning of binary ZERO.
4. Using rule of number formation.

Each of these methods is discussed below.

Dividing successively by the base. This method is applicable to integral decimal numbers. To translate an integral decimal number into binary, divide successively by 2 and record the remainder in each case. Continue this process until the original number is reduced to 0. The first recorded remainder bit is the LSD and the last recorded remainder bit is the MSD of the number in the binary system.

Example: Translate decimal 137 into binary, thus:

DIVIDE BY 2	QUOTIENT	REMAINDER
137/2	68	1—LSD
68/2	34	0
34/2	17	0
17/2	8	1
8/2	4	0
4/2	2	0
2/2	1	0
1/2	0	1—MSD

10001001 Binary

Multiplying successively by the base. This method is applicable to fractional decimal numbers. To translate a fractional decimal number into binary, multiply the fraction successively by 2. Each time the product becomes greater than 1, disregard the 1 and continue multiplying the fractional part of the number. For each multiplication that causes the product to become greater than 1—there is a carry to the units column—record a 1. For each multiplication that does not cause the product to become greater than 1—there is no carry to the units column—record a 0. Repeat this procedure until about three times as many binary digits as decimal digits are obtained. The first recorded bit is the MSD and the last recorded bit is the LSD of the number in binary.

Example: Translate decimal .413 into binary, thus:

MULTIPLY BY 2	PRODUCT	CARRY TO UNITS COLUMN
.413 × 2	.826	0—MSD
.826 × 2	1.652	1
.652 × 2	1.304	1
.304 × 2	.608	0
.608 × 2	1.216	1
.216 × 2	.432	0
.432 × 2	.864	0
.864 × 2	1.728	1
.728 × 2	1.456	1—LSD

0.011010011 Binary

Using meaning of binary ZERO. Writing a 0 to the right of the LSD of an integral binary number is the same as multiplying the whole number by 2. Writing a 0 to the left of the MSD (between the MSD and the binary point) of a fractional binary number is equivalent to dividing the whole number by 2. These facts can be used to translate from decimal to binary, as is shown by the following example.

Example: Translate decimal 855.0625 to binary. The integral portion and the fractional portion are best treated separately.

The integral portion, 855, may be considered to be composed of 800 + 40 + 15. The binary equivalents of 15 and 40 may be obtained from Table 4; they are:

$$15 = 1111$$
$$40 = 101000$$

The value for 800 may be obtained by recording the value for 100 and doubling it three times:

$$100 = 1100100$$
$$200 = 11001000$$
$$400 = 110010000$$
$$800 = 1100100000$$

The value of the number is now obtained by adding the three binary equivalents thus:

$$\begin{aligned} 800 &= 1100100000 \\ +40 &= \quad 101000 \\ \hline 840 &= 1101001000 \\ +15 &= \quad 1111 \\ \hline 855 &= 1101010111 \end{aligned}$$

The fractional part .625 is translated to binary as follows:

$$\begin{aligned} .5 &= .1 \\ .25 &= .01 \\ .125 &= .001 \\ .0625 &= .0001 \end{aligned}$$

Therefore, decimal 855.0625 equals 1101010111.0001 in binary.

Using rule of number formation. The following example illustrates this method.

Example: Express 972 in binary. This number can be expressed as

$$9 \times 10^2 + 7 \times 10^1 + 2 \times 10^0$$

If the binary equivalent to each of these numbers is obtained, and the indicated arithmetic is performed, the binary equivalent for the whole number is derived. From Table 4 we find that the following decimal numbers have the indicated binary equivalents:

9	1001
7	0111
2	0010
10	1010
100	1100100

The number 972 can therefore be expressed as

$$\begin{aligned} & 9 \times 10^2 + 7 \times 10^1 + 2 \times 10^0 \\ &= 1001 \times 1100100 + 111 \times 1010 + 10 \\ &= 1110000100 + 1000110 + 10 \\ &= 1111001100 \end{aligned}$$

Binary-To-Decimal Translation

The following are three methods which may be used to translate binary numbers into decimal numbers:

1. Using meaning of binary ZERO.
2. Dividing successively by binary equivalent of 10.
3. Assigning weights to bit positions.

The first two methods are good for translation of integers only. The third method may be used for fractional numbers as well. Each of these methods is discussed in the following.

Using meaning of binary ZERO. Start by recording a 1 for the MSD. Then double the number for each succeeding bit if it is a 0. Double the number and add 1 if the bit is a 1.

Example: Find the decimal equivalent of 10101100.

Binary number:	1	0	1	0	1	1	0	0	
Double numbers:	1	2	4	10	20	42	86	172	172 = Decimal equivalent
Add 1:			+1		+1	+1			
			5		21	43			

Dividing successively by binary equivalent of 10. The binary equivalent of decimal 10 is 1010. By dividing the given number and then successive quotients by 1010, each successive remainder translated into a decimal is a digit of the equivalent decimal number.

Example: Find the decimal equivalent of 10000100.

```
          1101
1010)10000100
      1010
       1101
       1010
         1100
         1010
           10 = 2——LSD——┐
         1                 │
1010)1101                  │
     1010                  │
       11     = 3          132  Decimal
                           ↑
1010)1        = 1——MSD——┘
```

Assigning weights to bit positions. This is the most straightforward way. Remembering that each bit to the left of the binary point is worth twice the value of the bit to its right, and that each bit to the right of the binary point is worth half as much as the bit to its left, we can assign weights to all bit positions as follows:

256	128	64	32	16	8	4	2	1 ·	1/2	1/4	1/8	1/16	1/32	1/64
x	x	x	x	x	x	x	x	x ·	x	x	x	x	x	x

To translate from binary to decimal, place the bits under the appropriate weights. Then add up all the weights in the positions in which 1's appear in the binary number.

Example: Find the decimal equivalent of 10101100.10101.
Place bits under appropriate weights:

Weights:	128	64	32	16	8	4	2	1 ·	1/2	1/4	1/8	1/16	1/32
Bits:	1	0	1	0	1	1	0	0 ·	1	0	1	0	1

Add weights in positions in which 1's appear:

$$128 + 32 + 8 + 4 + 1/2 + 1/8 + 1/32 = 172.6562$$

TABLE 6
Translating Decimal 540 to Binary, Ternary, Quarternary, and Octal

Binary			Ternary			Quarternary			Octal		
Divide by Base	Quotient	Remainder	Divide by Base	Quotient	Remainder	Divide by Base	Quotient	Remainder	Divide by Base	Quotient	Remainder
540/2	270	0—LSD	540/3	180	0—LSD	540/4	135	0—LSD	540/8	67	4—LSD
270/2	135	0	180/3	60	0	135/4	33	3	67/8	8	3
135/2	67	1	60/3	20	0	33/4	8	1	8/8	1	0
67/2	33	1	20/3	6	2	8/4	2	0	1/8	0	1—MSD→1034
33/2	16	1	6/3	2	0	2/4	0	2—MSD→20130			
16/2	8	0	2/3	0	2—MSD→202000						
8/2	4	0									
4/2	2	0									
2/2	1	0									
1/2	0	1—MSD→1000011100									

The translation can also be performed by treating the mixed number as a fraction: 1010110010101/100000.

Weights:	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Bits:	1	0	1	0	1	1	0	0	1	0	1	0	1

Numerator: $4096 + 1024 + 256 + 128 + 16 + 4 + 1 = 5525$

Weights:	32	16	8	4	2	1
Bits:	1	0	0	0	0	0

Denominator: 32

Fraction: $5525/32 = 172.6562$

Translation to and from Other Number Systems

One of the best methods of translating from decimal to binary is successive division (multiplication if the number is fractional) by 2, the remainder after each division being one of the bits of the resultant number. This method may be applied to translation from decimal to any number system with a base less than 10; division in each case is by the base of the number system to which we are translating. Table 6 shows how the decimal number 540 is translated into ternary, quarternary, and octal as well as binary.

TABLE 7
Translating Binary, Ternary, Quarternary, and Octal Equivalents of 540 to Decimal

Number in binary: 1000011100

Weights:	512	256	128	64	32	16	8	4	2	1
Bits:	1	0	0	0	0	1	1	1	0	0

Bits × weights: $1 \times 512 + 0 \times 256 + 0 \times 128 + 0 \times 64 + 0 \times 32 + 1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 5 + 0 \times 1$

Decimal equivalent: $512 + 16 + 8 + 4 = 540$

Number in ternary: 202000

Weights:	243	81	27	9	3	1
Digits:	2	0	2	0	0	0

Digits × weights: $2 \times 243 + 0 \times 81 + 2 \times 27 + 0 \times 9 + 0 \times 3 + 0 \times 1$

Decimal equivalent: $486 + 54 = 540$

Number in Quarternary: 20130

Weights:	256	64	16	4	1
Digits:	2	0	1	3	0

Digits × weights: $2 \times 256 + 0 \times 64 + 1 \times 16 + 3 \times 4 + 0 \times 1$

Decimal equivalent: $512 + 16 + 12 = 540$

Number in Octal: 1034

Weights:	512	64	8	1
Digits:	1	0	3	4

Digits × weights: $1 \times 512 + 0 \times 64 + 3 \times 8 + 4 \times 1$

Decimal equivalent: $512 + 24 + 4 = 540$

One of the best methods of translating from binary to decimal is by assigning weights to bit positions, multiplying the weights by the digits in the respective positions and summarizing. This method may be applied to translation from any number system with a base less than 10 to decimal. Table 7 shows how the ternary, quaternary, and octal as well as binary equivalents of 540 are translated into decimal.

TABLE 8
Binary-Octal Equivalents

Decimal	Binary	Octal
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7
8	001 000	10
9	001 001	11
10	001 010	12
11	001 011	13
12	001 100	14
13	001 101	15
14	001 110	16
15	001 111	17
16	010 000	20
17	010 001	21
18	010 010	22
19	010 011	23
20	010 100	24
40	101 000	50
100	001 100 100	144

Translation from binary to octal, and vice versa, is very simple because there is a direct relationship between the binary number system and the octal number system: if the bits of a binary number are divided into groups of three, the evaluation of each group provides the equivalent octal digit. This can be seen from the table of binary-octal equivalents (Table 8). The reason for this simple relation is this: there are only eight possible combinations for each group of three bits.

REFLECTED NUMBERS

When counting in a positional number system, occasionally all the digits of a number must be changed in order to obtain the following number in sequence. For instance, the number after 199999 is 200000. If the indicator of a mechanized counter should fall somewhere between these two numbers, the actual number read out may vary anywhere from 100000 and 299999.

TABLE 9
Reflected Numbers

Decimal	Reflected Decimal	Reflected Octal	Reflected Quarternary	Reflected Binary
0	0	0	0	0
1	1	1	1	1
2	2	2	2	11
3	3	3	3	10
4	4	4	13	110
5	5	5	12	111
6	6	6	11	101
7	7	7	10	100
8	8	17	20	1100
9	9	16	21	1101
10	19	15	22	1111
11	18	14	23	1110
12	17	13	33	1010
13	16	12	32	1011
14	15	11	31	1001
15	14	10	30	1000
16	13	20	130	11000
17	12	21	131	11001
18	11	22	132	11011
19	10	23	133	11010
20	20	24	123	11110
21	21	25	122	11111
22	22	26	121	11101
23	23	27	120	11100
24	24	37	110	10100
25	25	36	111	10101
26	26	35	112	10111
27	27	34	113	10110
28	28	33	103	10010

When we are interested in reducing the error which can be produced by a counter—as we are when converting an analog signal to digital form—a reflected form of the number system may be used. The reflected form possesses the quality that two adjacent numbers differ from each other in the value of only one of the digits. This means that regardless of how many digits a number possesses, only one of two possible digit combinations may be read at any one point.

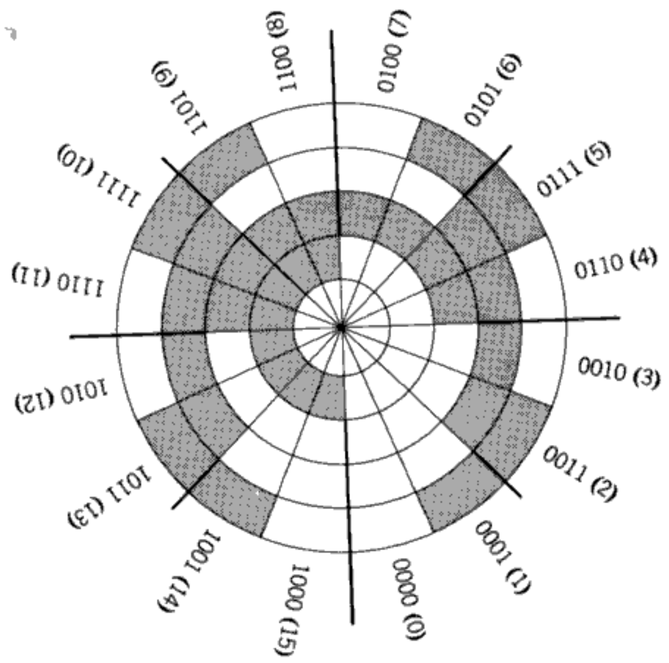


Fig. 10. Reflected binary in modulus 2^4.

Examples of reflected number systems are given in Table 9. Figure 10 is a pictorial illustration of the reflected binary system with a 2^4 modulus. Because the number after 15 is 0, the number system may be represented by a circle with 16 subdivisions as shown. A black area represents a bit of ONE and a blank area represents a bit of ZERO. Note that the pattern of the three outside bands on the left half of the circle is the mirror image of the pattern on the three outside bands on the right side of the circle. If the two outside bands are considered, the pattern in each quarter circle is the mirror image of the pattern in the adjacent quarter circle. If only the outside band is considered, the pattern in each eighth-circle is a mirror image of the pattern in the adjacent eighth-circle. Similar mirror images or reflected patterns exist in all reflected number systems.

CODED-DECIMAL NUMBERS

The decimal number system is universally used. But the binary number system possesses a simplicity that lends itself to machine use. These two advantages are available in one system, called the coded-decimal system. Numbers in the coded-decimal system are represented by decimal digits; but each digit is encoded by means of four bits. For instance, the binary representations for the digits 0 to 9 may be used to encode each digit. In this system the number 52963 is encoded as follows:

$$52963 = 0101\ 0010\ 1001\ 0110\ 0011$$

We may assign to each of the four bit positions (from right to left) the weights of 1, 2, 4, and 8; and then add the weights in positions holding 1's to evaluate each digit. The above binary-coded decimal representation may therefore be translated in the following manner:

Binary-Coded Decimal Digits	Weights 8	4	2	1	Decimal
0101	0 +	4 +	0 +	1	5
0010	0 +	0 +	2 +	0	2
1001	8 +	0 +	0 +	1	9
0110	0 +	4 +	2 +	0	6
0011	0 +	0 +	2 +	1	3

This type of representation is called a weighted-bit code. There are many other possible weights that can be given to each bit position to produce other coded-decimal systems. Some of these are listed in Table 10. Values for digits in typical 4-bit codes are given in Table 11.

In coded-decimal systems an ambiguity may arise in the use of the word "significant." The leftmost decimal digit (5 in the above example) may be called the most significant. Similarly, the leftmost bit comprising each digit may be called the most significant. To avoid confusion, we will always use "significant" when referring to digits, and "order" when referring to the bits of a digit. The leftmost decimal digit in the above example is the MSD and the rightmost is the LSD. The first-, second-, third-, and fourth-order bits of the MSD are 1, 0, 1, and 0 respectively.

There are 16 distinct possible combinations that can be obtained by the use of 4 bits to a digit. In the 8421 system, the first ten combinations in the binary number system are chosen. It is possible, however, to choose the middle ten combinations in the group of 16. This means that the binary

TABLE 10
Weighted 4-Bit Codes

Order of Bit				Order of Bit				Order of Bit			
4	3	2	1	4	3	2	1	4	3	2	1
5	2	1	1	6	2	2	1	7	3	2	1
4	3	1	1	3	3	2	1	4	4	2	1
5	3	1	1	4	3	2	1	5	4	2	1
6	3	1	1	5	3	2	1	6	4	2	1
4	2	2	1	6	3	2	1	7	4	2	1
5	2	2	1	2	4	2	1	8	4	2	1

TABLE 11
Values for Typical 4-Bit Codes

Decimal	Weighted				Nonweighted	
	2421	8421	5421	5211	Excess-3	Excess-6
0	0000	0000	0000	0000	0011	0110
1	0001	0001	0001	0001	0100	0111
2	0010	0010	0010	0011	0101	1000
3	0011	0011	0011	0110	0110	1001
4	0100	0100	0100	0111	0111	1010
5	1011	0101	1000	1000	1000	1011
6	1100	0110	1001	1001	1001	1100
7	1101	0111	1010	1100	1010	1101
8	1110	1000	1011	1110	1011	1110
9	1111	1001	1100	1111	1100	1111

representations for 3 to 12 would represent decimal digits from 0 to 9. This is shown in Table 11 as "Excess-3" code. The code is called excess-3 because the coded representation is the binary number for three more than the actual decimal number it represents. For instance, the excess-3 representation for 0 is 0011, or the binary equivalent for 3.

Similarly, in the excess-6 code, each coded representation is equal to the binary number for six more than the decimal number it usually represents.

NEGATIVE NUMBERS AND COMPLEMENTS

Up to now we have been dealing only with positive numbers, that is, we counted from a reference point in a forward direction. But it is just as easy to start at the reference point and count in the opposite direction.

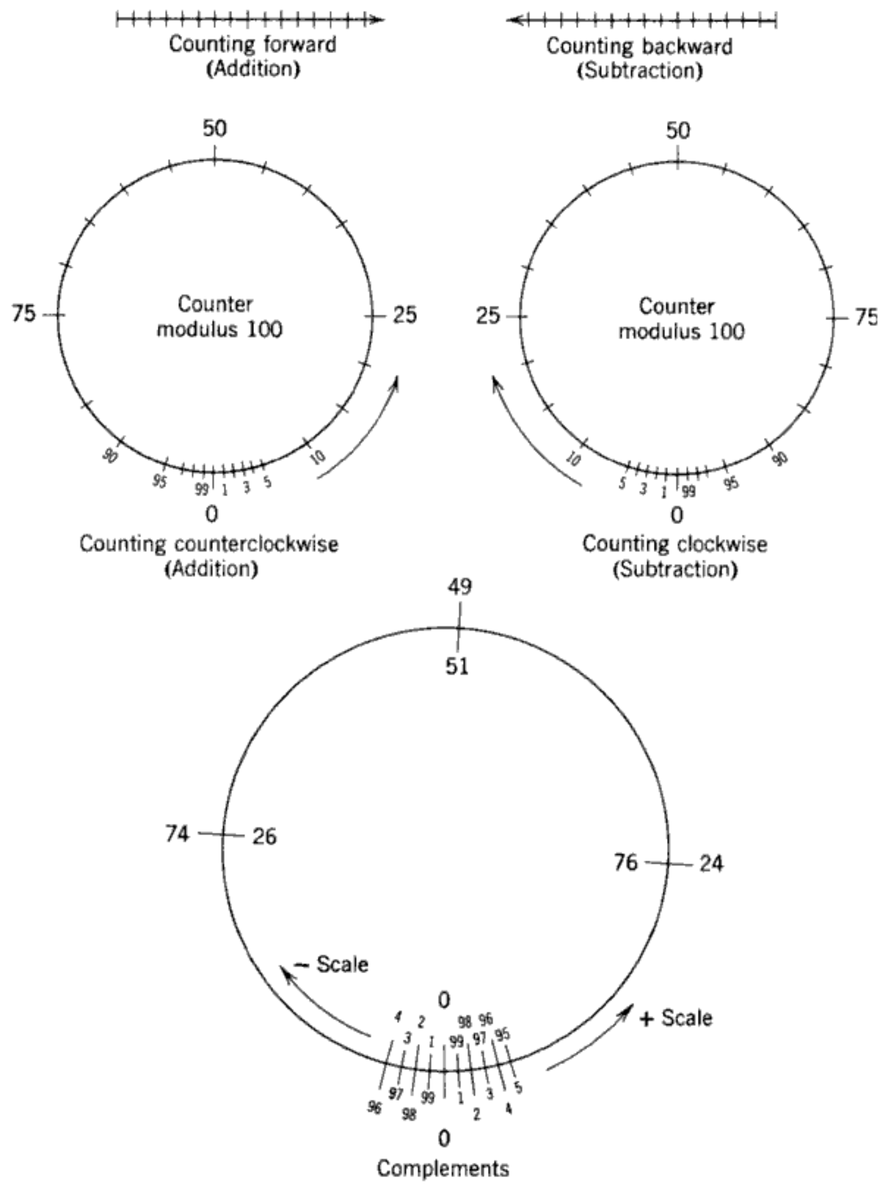


Fig. 11. Principle of complements.

If we assume that counting to the right is counting forward, then counting to the left is counting backward. When we count to the right, we add a one to obtain each succeeding number; when we count to the left we subtract one to obtain each succeeding number.

Because of its finite modulus, a practical counter is better represented by a circle, as shown in Fig. 11. In the circle representation, counting counterclockwise may be called forward counting, and counting clockwise may be called backward counting. If we place the positive numbers

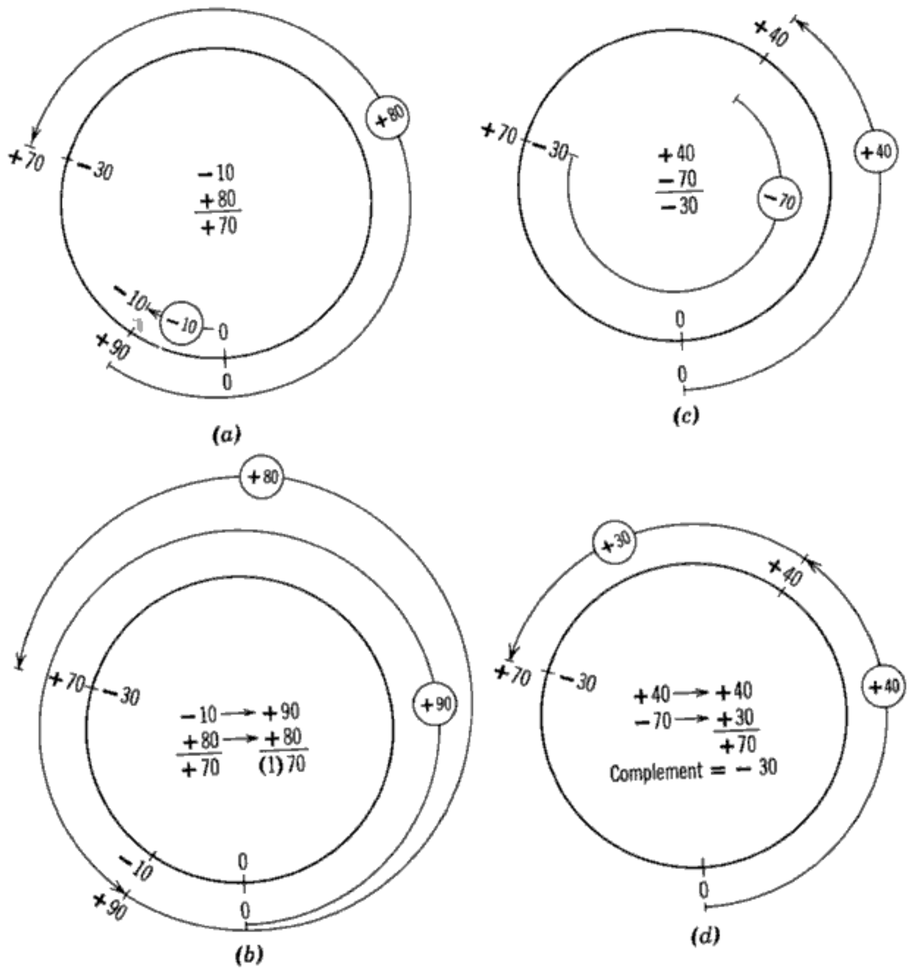


Fig. 12. Subtraction with the aid of complements.

obtained through forward counting and the negative numbers obtained through backward counting side by side around the circle, the corresponding numbers are complements of each other. A complement is a number derived from a given number according to a definite rule; the complement is used to facilitate arithmetic.

The complement concept is valuable in the mechanization of subtraction. Subtraction may be performed by reversing the direction of a wheel (or some other device which behaves conceptually like a wheel). By converting negative numbers to their complements, we can convert all subtraction problems into addition problems—we can have the "wheel" always moving in the same direction.

Figure 12 shows how the complement concept is applied in a counter with a modulus of 100. The numbers −10 and +80 are combined normally

in Fig. 12*a* to produce +70: Starting at the reference point, we subtract 10 by ticking off ten spaces clockwise; from the resultant we trace 80 spaces counterclockwise, which brings us back to the +70 point on the wheel. Figure 12*b* shows how the same problem is solved by means of complements; here we combine the complement of −10, which is +90, with +80; we count 170 divisions from the reference point, which brings us to the +70 point on the wheel. In Figs. 12*c* and *d*, +40 and −70 are combined. In Fig. 12*c* we count 40 divisions forward and then 70 divisions backward, which brings us to the −30 spot. In Fig. 12*d* we convert −70 to +30, and count first 40 divisions and then 30 divisions counterclockwise, which brings us to the +70 or −30 spot on the wheel.

If the modulus is given in the decimal system, the complement is called the 10's complement. The 10's complement of a number is defined as that number which when added to the original number gives the modulus. For instance, the 10's complement of 642 is 358 because

$$\begin{array}{r} 642 \\ +358 \\ \hline 1000 \end{array} = \text{modulus for a 3-digit decimal number system}$$

The 10's complement of the same 642 in modulus 10^5 is 99358 because

$$\begin{array}{r} 00642 \\ +99358 \\ \hline 100000 \end{array} = \text{modulus for a 5-digit decimal number system}$$

Another way of expressing the last statement is as follows: If we count 642 times backwards from 100000, we reach number 99358.

Instead of obtaining the 10's complement by subtracting the number from the modulus, the first digit (LSD) may be subtracted from 10 and all the other digits subtracted from 9. This process gives the same result as can be seen from the following:

100000	9 9 9 9 10	9 9 9 9 9	
−42135	−4 2 1 3 5	−4 2 1 3 5	
57865	5 7 8 6 5	5 7 8 6 4	9's complement
		+1	
		5 7 8 6 5	10's complement

Instead of obtaining the 10's complement directly, we may subtract each digit from 9 to obtain the 9's complement, and then add 1 to convert it into the 10's complement. The 10's complement is always equal to the 9's complement plus 1.

In the binary number system, the 2's and 1's complements are analogous to the 10's and 9's complements respectively. The 2's complement is defined as that binary number which must be added to the original binary number to obtain the modulus. The 1's complement is defined to be one less than the 2's complement. The following example shows the relationships between the 2's and 1's complements:

Example: Find the 2's complement of 10101

100000	1 1 1 1 1 10	1 1 1 1 1 1	
−10101	− 1 0 1 0 1	− 1 0 1 0 1	
01011	0 1 0 1 1	0 1 0 1 0	1's Complement
		+ 1	
		0 1 0 1 1	2's Complement

Note that in the binary number system, the 1's complement can be obtained by simply interchanging all 0's with 1's and all 1's with 0's.

ADVANCED CODES

Self-Complementing Codes

In several coded-decimal systems, replacing 0's with 1's and 1's with 0's produces the 9's complement. Codes which have this happy facility are called self-complementing codes. Several examples of self-complementing codes are given in Table 12.

TABLE 12
Self-Complementing Codes

	Coded Decimal					
Decimal	2421	5211	4311	4221	3321	Excess-3
0	0000	0000	0000	0000	0000	0011
1	0001	0001	0001	0001	0001	0100
2	0010	0011	0011	0010	0010	0101
3	0011	0110	0100	0011	0011	0110
4	0100	0111	0110	1000	0101	0111
5	1011	1000	1001	0111	1010	1000
6	1100	1001	1011	1100	1100	1001
7	1101	1100	1100	1101	1101	1010
8	1110	1110	1110	1110	1110	1011
9	1111	1111	1111	1111	1111	1100

Self-Checking Codes

Four bits can be combined in 16 different ways. Since only ten of these combinations are used in a coded-decimal system, six combinations are extra or redundant. This redundancy increases the reliability of the code because the machine can be made to detect the six forbidden combinations. The forbidden combinations in the 8421 system, for instance, are 1010, 1011, 1100, 1101, 1110, and 1111. If one of these combinations occurs, we know that the machine has made an error. By introducing at the appropriate point in the machine a device which can detect any of these six forbidden combinations, we have an automatic means of detecting machine errors.

But the detection of forbidden combinations does not detect *all* errors. Suppose the number 1000 is transmitted and the machine causes the second-order bit to invert, thus transmitting 1010. Since this is a forbidden combination, the machine can detect it. But suppose the *first*-order bit is inverted and 1001 is transmitted. This is not a forbidden combination—it is the representation for 9—and it may go undetected.

If we increase the redundancy further by introducing another bit to each coded combination, we could develop a system which could discover an error due to the change of any of the bits in the coded combination. To do this, we would have to make this bit related to the others. One way of doing this would be to form the rule that the sum of all the ONE bits in each 5-bit combination must always be odd. On the basis of this rule, the check bit (as the fifth-order bit is called) for the 8421 code is as follows:

Order of Bit / Weight / Decimal	5 Check	4 8	3 4	2 2	1 1
0	1	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	1	0	0	1	1
4	0	0	1	0	0
5	1	0	1	0	1
6	1	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	1	1	0	0	1

If a bit in any of these combinations should change, the sum of the ONE's would no longer be odd but even. The machine could be made to detect this. This system of checking is called *parity checking*.

A binary-coded decimal system with an added check bit is called a self-checking code. Other self-checking codes are shown in Table 13. In the other self-checking codes shown—the two out of five, modified bi-quinary and modified qui-binary systems—the rule used for checking is that each digit must have two and only two ONE's. If an error causes the number of ONE's in a digit to change, this fact can be detected.

TABLE 13
Self-Checking Codes

Decimal \ Order of Bit / Weight	8421 with Parity Check: 5 4 3 2 1 / Check 8 4 2 1	2 out of 5: 5 4 3 2 1 / 7 4 2 1 0	Modified Bi-Quinary: 7 6 5 4 3 2 1 / 5 0 4 3 2 1 0	Modified Qui-Binary: 7 6 5 4 3 2 1 / 8 6 4 2 0 1 0
0	1 0 0 0 0	1 1 0 0 0	0 1 0 0 0 0 1	0 0 0 0 1 0 1
1	0 0 0 0 1	0 0 0 1 1	0 1 0 0 0 1 0	0 0 0 0 1 1 0
2	0 0 0 1 0	0 0 1 0 1	0 1 0 0 1 0 0	0 0 0 1 0 0 1
3	1 0 0 1 1	0 0 1 1 0	0 1 0 1 0 0 0	0 0 0 1 0 1 0
4	0 0 1 0 0	0 1 0 0 1	0 1 1 0 0 0 0	0 0 1 0 0 0 1
5	1 0 1 0 1	0 1 0 1 0	1 0 0 0 0 0 1	0 0 1 0 0 1 0
6	1 0 1 1 0	0 1 1 0 0	1 0 0 0 0 1 0	0 1 0 0 0 0 1
7	0 0 1 1 1	1 0 0 0 1	1 0 0 0 1 0 0	0 1 0 0 1 0 0
8	0 1 0 0 0	1 0 0 1 0	1 0 0 1 0 0 0	1 0 0 0 0 0 1
9	1 1 0 0 1	1 0 1 0 0	1 0 1 0 0 0 0	1 0 0 0 0 1 0

Alphanumeric Codes

Many computers can process letters and editing marks as well as digits. To take care of all these characters, the number of bits in each coded combination is expanded to six plus a check bit, or to a total of seven bits. One such alphanumeric code is presented in Table 14.

The code used for teletype transmission accomplishes the same purpose with only five bits. Five bits can be used to represent 32 characters, which is not enough to take care of the 26 letters, 10 digits, and many editing marks. To get around this impasse, the teletype code uses the same codes for the 26 letters and for the digits and editing marks. To avoid confusion, one of the unused combinations is used to signal that the message which follows consists of alphabetic information; or, if receiving alphabetic information, the special code signals that the following information consists of digits and editing marks. The teletype code is shown in Table 14.

TABLE 14
Alphanumeric Codes

Character \ Order of Bit	7-Bit Code: 7 6 5 4 3 2 1	Teletype Code: 5 4 3 2 1	Alternate Meaning
0	1 0 0 0 0 1 1		
1	0 0 0 0 1 0 0		
2	1 0 0 0 1 0 1		
3	1 0 0 0 1 1 0		
4	0 0 0 0 1 1 1		
5	0 0 0 1 0 0 0		
6	1 0 0 1 0 0 1		
7	1 0 0 1 0 1 0		
8	0 0 0 1 0 1 1		
9	1 0 0 1 1 0 0		
A	1 0 1 0 1 0 0	0 0 0 1 1	–
B	0 1 0 0 1 0 1	1 1 0 0 1	?
C	0 1 0 0 1 1 0	0 1 1 1 0	:
D	1 0 1 0 1 1 1	0 1 0 0 1	$
E	1 0 1 1 0 0 0	0 0 0 0 1	3
F	0 1 0 1 0 0 1	0 1 1 0 1	!
G	0 1 0 1 0 1 0	1 1 0 1 0	&
H	1 0 1 1 0 1 1	1 0 1 0 0	£
I	0 1 0 1 1 0 0	0 0 1 1 0	8
J	1 0 1 0 1 0 0	0 1 0 1 1	'
K	0 1 0 0 1 0 1	0 1 1 1 1	(
L	0 1 0 0 1 1 0	1 0 0 1 0	)
M	1 0 1 0 1 1 1	1 1 1 0 0	.
N	1 0 1 1 0 0 0	0 1 1 0 0	,
O	0 1 0 1 0 0 1	1 1 0 0 0	9
P	0 1 0 1 0 1 0	1 0 1 1 0	0
Q	1 0 1 1 0 1 1	1 0 1 1 1	1
R	0 1 0 1 1 0 0	0 1 0 1 0	4
S	1 1 1 0 1 0 1	0 0 1 0 1	Bell
T	1 1 1 0 1 1 0	1 0 0 0 0	5
U	0 1 1 0 1 1 1	0 0 1 1 1	7
V	0 1 1 1 0 0 0	1 1 1 1 0	;
W	1 1 1 1 0 0 1	1 0 0 1 1	2
X	1 1 1 1 0 1 0	1 1 1 0 1	/
Y	0 1 1 1 0 1 1	1 0 1 0 1	6
Z	1 1 1 1 1 0 0	1 0 0 0 1	"

Note: 11011 on teletype means "switch to figures."
11111 on teletype means "switch to letters."

SUMMARY

1. The more redundant the language, the greater its reliability. The less redundant the language, the more powerful it is as an instrument for rigorous thought.

2. Ancient number systems were additive; present-day number systems are positional. The invention of zero made the positional notation possible. When numbers are written in exponential form, a positive exponent indicates the number of digits between the most significant digit and the decimal point to the right; a minus exponent indicates the number of digits to the right of the decimal point.

3. The binary number system with two digits (ZERO and ONE) is especially useful because of its similarity to bivalued logic, the ease with which it can be mechanized, and the resultant simplicity of binary arithmetic.

4. Integral (fractional) decimal numbers may be converted to numbers in a base less than 10 by a technique of dividing (multiplying) successively by the base. Numbers in systems having bases of less than 10 may be converted to decimal by multiplying each digit by a weight assigned to the digit position in question, and summarizing; the weights are multiples of the base of the number system from which the numbers are being converted.

5. There is a special relationship between binary and octal numbers. If bits of a binary number are grouped into threes, the decimal value of each 3-bit combination provides the octal equivalent of the binary number.

6. Four bits may be used to encode a decimal digit. By weighting each bit differently, several such encoding schemes can be produced. By using six bits, letters of the alphabet, editing marks as well as digits may be encoded.

7. Reflected number systems are useful in systems requiring the conversion of analog signals to digital form because only one digit changes when switching from one number to the succeeding number.

8. Addition is the process of counting two numbers in the same direction. Subtraction is the process of first counting one number in one direction, then counting the other number in the opposite direction. Negative numbers may be replaced by complements to convert subtraction problems into addition problems.

9. Checking is facilitated by redundancies. Any coded-decimal system may be converted to a self-checking code by the inclusion of a checking bit.

10. *Definitions to Remember*

DIGIT—A symbol representing one of the elements in a positional number system.

BIT—One of the two elements used in the binary number system. Also one of the two elements used in coded-decimal systems.

BASE OR RADIX—Number of digits (or bits) to a number system.

MODULUS—One more than the maximum number attainable in any practical counter.

SIGNIFICANCE—Rank of digit position in a number.

LEAST SIGNIFICANT DIGIT (LSD)—The rightmost digit of a number.

MOST SIGNIFICANT DIGIT (MSD)—The leftmost digit of a number.

BINARY-CODED DECIMAL NOTATION—A decimal number system in which each digit is represented by a group of bits.

ORDER—Rank of bits in a coded-decimal digit.

COMPLEMENT:

10's—The number which must be added to the given number to obtain the modulus in the decimal system.

9's—One less than the 10's complement.

2's—The number which must be added to the given number to obtain the modulus in the binary system.

1's—One less than the 2's complement.

REDUNDANCY—The use of more symbols to encode a message than are strictly necessary.

PARITY CHECK—Comparison of the sum of 1 bits in a coded representation to see if it is odd or even.

FORBIDDEN COMBINATION CHECK—A check which tests for occurrence of a nonpermissible code combination.

SELF-COMPLEMENTING CODE—A code in which all numbers may be converted to their 9's or 1's complements by inverting 1's and 0's.

SELF-CHECKING CODE—A code in which it is easy to check each character because of a redundancy in each character.

CHAPTER 3

Arithmetic processes

Almost all practical mathematical operations may be converted into arithmetic operations; and all arithmetic operations are extensions of the operation of addition. This means that if we can build a device which can add, we can, with a few modifications, convert it to a device which can also subtract, multiply, divide, take the square root, and integrate; and by proper programming, we can even cause it to perform problems in higher mathematics.

Before building this adding device, we review the rules of algebraic addition, and then go through several methods for multiplying, dividing, and taking the square root in the decimal system. This is followed by methods of performing the same arithmetic operations in binary and in coded-decimal systems.

ALGEBRAIC ADDITION

Algebraic addition refers to the summation of numbers some of which are positive and some of which are negative. There are several methods for algebraic addition in the decimal system. The addition may be performed by using the numbers given, or it may be done after the negative numbers are converted into their 10's or 9's complements. Variations of these techniques are possible. Some of the common methods of algebraic addition are discussed in the following.

Using Absolute Values

The rules of algebraic addition of absolute decimal numbers are simple and straightforward.

1. To add two numbers whose signs are alike—both plus or both minus—add the numbers and give the sum the sign of either number.

Example:

+372	−372	Augend
+198	−198	Addend
+570	−570	Sum

2. To add algebraically two numbers whose signs are unlike—plus and minus or minus and plus—subtract the numerically smaller number from the numerically larger number and give the sum the sign of the numerically larger number.

Examples:

+372	−372	Minuend
−198	+198	Subtrahend
+174	−174	Difference

The terms minuend, subtrahend, and difference are used to show that a subtraction is performed; augend, addend, and sum are perhaps more correct since the process is algebraic *addition.*

Using 10's Complement

Algebraic addition of two terms whose signs are unlike may be performed in the following manner. All negative numbers may be replaced by their 10's complements and the terms added. The treatment of the sum depends upon whether or not a decimal carry is produced from the most significant position.

1. If a decimal carry occurs, ignore the carry and make the sign plus.

Example:

+372 → 372
−198 —10's complement→ 802
+174 (1) 174 → +174 = Sum

2. If no decimal carry occurs, take the 10's complement of the tentative sum and make the sign negative:

Example:

−372 —10's complement→ 628
+198 → 198
−174 826 —10's complement→ −174 = Sum

Note that, if we change the sign of the number every time we complement, we automatically arrive at the correct sign of the answer. This is logical because when we take the complement we count in the opposite direction from that in which we have been counting before.

Using the 9's Complement

A similar set of rules applies for determining the sum of two unlike numbers when algebraic addition is performed with the aid of the 9's rather than the 10's complements. Again we take the complement of the negative term and add. The sum is treated in one of two ways depending upon whether or not a decimal carry is produced from the most significant position:

1. If a decimal carry occurs, add this 1 to the LSD of the tentative sum (end-around carry), and make the sign of the sum positive.

Example:

```
+372 ------------------------> 372
-198 ---- 9's complement ----> 801
-174                        (1) 173
                                 1   End-around carry
                                 174 ------------------> +174 = Sum
```

2. If no decimal carry occurs, take the 9's complement of the tentative sum and make the sign of the sum negative.

Example:

```
+198 ------------------------> 198
-372 ---- 9's complement ----> 627
-174                          825 ---- 9's complement ---> -174 = Sum
```

Adding the end-around carry when working with the 9's complement is equivalent to taking the 10's complement of the addend originally, because the 10's complement of a number is one more than the 9's complement of the number.

MULTIPLICATION

Pencil-and-Paper Multiplication

Every schoolboy knows the following method of multiplying 728 by 451:

728	Multiplicand
×451	Multiplier
728	First partial product
3640	Second partial product
2912	Third partial product
328328	Product

Why do we indent the second partial product (3640) one digit position to the left? and the third partial product another digit position to the left?

The answer is as follows. When we multiply 728 by the least significant digit of the multiplier (1), what we are actually doing is this:

$$\begin{array}{ccccc} 700 & & 20 & & 8 \\ \times 1 & & \times 1 & & \times 1 \\ \hline 700 & + & 20 & + & 8 = 728 = \text{First partial product} \end{array}$$

When we multiply 728 by the second digit (5), we are not really multiplying by 5 but by 50:

$$\begin{array}{ccccc} 700 & & 20 & & 8 \\ \times 50 & & \times 50 & & \times 50 \\ \hline 35000 & + & 1000 & + & 400 = 36400 = \text{Second partial product} \end{array}$$

We put the second partial product down as 3640, but indented to the left one place, making it equivalent to 36400. In the same way when we multiply 728 by 4, we are not really multiplying by 4 but by 400:

$$\begin{array}{ccccc} 700 & & 20 & & 8 \\ \times 400 & & \times 400 & & \times 400 \\ \hline 280000 & + & 8000 & + & 3200 = 291200 = \text{Third partial product} \end{array}$$

We put the third partial product down as 2912, but shifted one digit position to the left of the second partial product, making it equivalent to 291200.

Right- and Left-Hand Component Method

When multiplying pairs of digits in the pencil-and-paper method, we may obtain a carry; this carry must be added to the result of the multiplication of the next pair of digits. To avoid complication which such a routine may cause in a machine, the product of each digit pair may be separated into a right-hand or product digit, and a left-hand or carry digit. These digits are part of the left- and right-hand components respectively. The right- and left-hand components are summed separately. The sum of the left-hand components is then shifted one place to the left and added to the sum of the right-hand components to obtain the product.

Example:

```
                                  728
                                 ×451
                                 ----
                  ( 000          728 )
Left-hand components { 314          500 } Right-hand components
                  ( 203          882 )
Left-hand sum      23440         93928  Right-hand sum
                                23440  Left-hand sum, shifted left
                                328328  Product
```

Multiplication by Repeated Addition

By storing the multiplication table in the machine's memory, it is possible to adapt the pencil-and-paper method of multiplication to machine computation. To avoid occupying such valuable storage space, multiplication in most present-day machines is performed by a process of repeated addition. Very little extra storage is needed because the additions are performed by the same adder used for straightforward addition.

The method of repeated addition is based upon the meaning of multiplication. The multiplication symbol tells us to add the multiplicand to itself the number of times specified by the multiplier. For instance, 8 times 4 means that we should add four 8's together to get 32.

Example:

$$
\begin{array}{rlll}
728 = 728 & 728 & 728 & 728 \\
\times 451 & \times 4 \times 10^2 & \times 5 \times 10^1 & \times 1 \times 10^0 \\
\hline
 & 2912 \times 10^2 & 3640 \times 10^1 & 728 \times 10^0
\end{array}
$$

$$
\begin{array}{rlll}
 & 728 & 728 & 728 \\
 & 728 & 728 & \\
 & 728 & 728 & \\
= & 728 \quad + & 728 \quad + & \\
 & 728 & 728 & \\
 & & 728 & \\
\hline
 & 2912 \times 10^2 & + \ 3640 \times 10^1 & + \ 728 \times 10^0 \\
= & 291200 & + \ 36400 & + \ 728 \\
= & 328328 & &
\end{array}
$$

The following summation is equivalent to the above steps:

$$
\begin{array}{ll}
728 & \times 1 \\
728 \\
728 \\
728 & \times 5 \\
728 \\
728 \\
728 \\
728 & \times 4 \\
728 \\
\hline
328328
\end{array}
$$

The latter summation is easy to adapt to machine computation because only addition and shifting of digits are required.

DIVISION

Pencil-and-Paper Division

The following is an example of the pencil-and-paper method of division:

Example:

	1241	Quotient
Divisor 365	)452980	Dividend
	365	
	879	First intermediate remainder
	730	
	1498	Second intermediate remainder
	1460	
	380	Third intermediate remainder
	365	
	15	Remainder

If the understood zeros are filled in, the example looks like this:

$$
\begin{array}{r}
1241 \\
\hline
365)452980 \\
365000 \\
\hline
87980 \\
73000 \\
\hline
14980 \\
14600 \\
\hline
380 \\
365 \\
\hline
15
\end{array}
$$

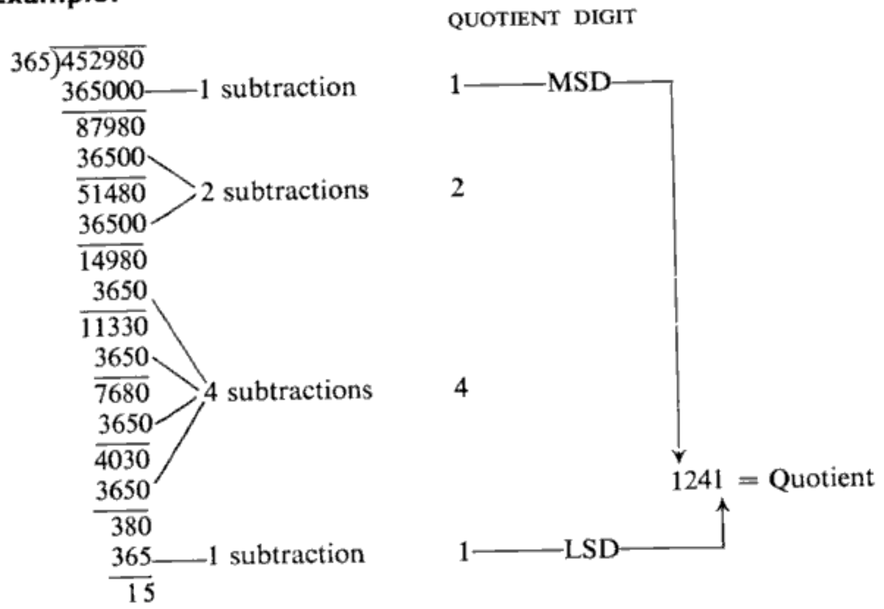
Whereas in multiplication each successive partial product is shifted one position to the left of the previous partial product, in division each successive intermediate remainder is shifted to the right of the previous intermediate remainder.

Division by Repeated Subtraction

The pencil-and-paper method of division is hard to mechanize because of the comparisons and trial-quotient digits needed. A more easily mechanized way of performing division uses a system of repeated subtraction. In this method the divisor is repeatedly subtracted from the

dividend until an intermediate remainder is obtained which admits of no further subtraction (without obtaining a negative result). The number of subtractions performed is the value of the MSD of the quotient. Next, the divisor is shifted one place to the right and the subtractions are repeated. The number of subtractions required to produce a remainder which admits of no further subtractions determines the next most significant digit of the quotient. This procedure is repeated until the required decimal places have been obtained.

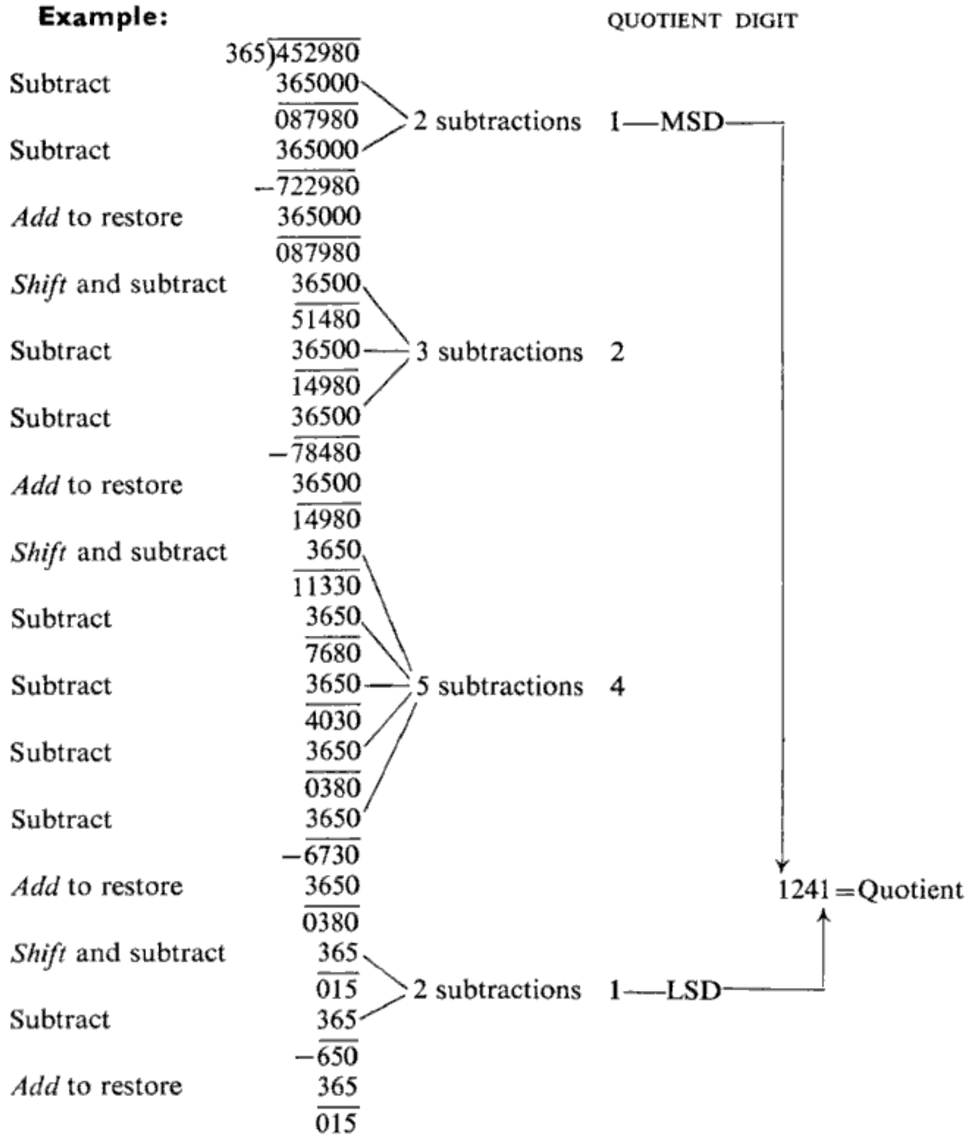
Example:



Restoring Method of Division

From the standpoint of the machine, division by repeated subtraction has one flaw: the machine cannot tell by looking at it whether an intermediate remainder admits of further subtraction or not. To get around this objection we make the machine repeatedly subtract the divisor until a *negative* intermediate remainder is obtained. The negative remainder indicates that the divisor was subtracted one more time than necessary. Therefore, as soon as a negative intermediate remainder is detected, the divisor is added to it to restore it to the correct intermediate remainder. This is why this method is called the restoring method of division. After the first intermediate remainder is developed, the next digit is brought down from the dividend, the divisor is shifted to the right one place, and the next series of subtractions is performed.

Example:

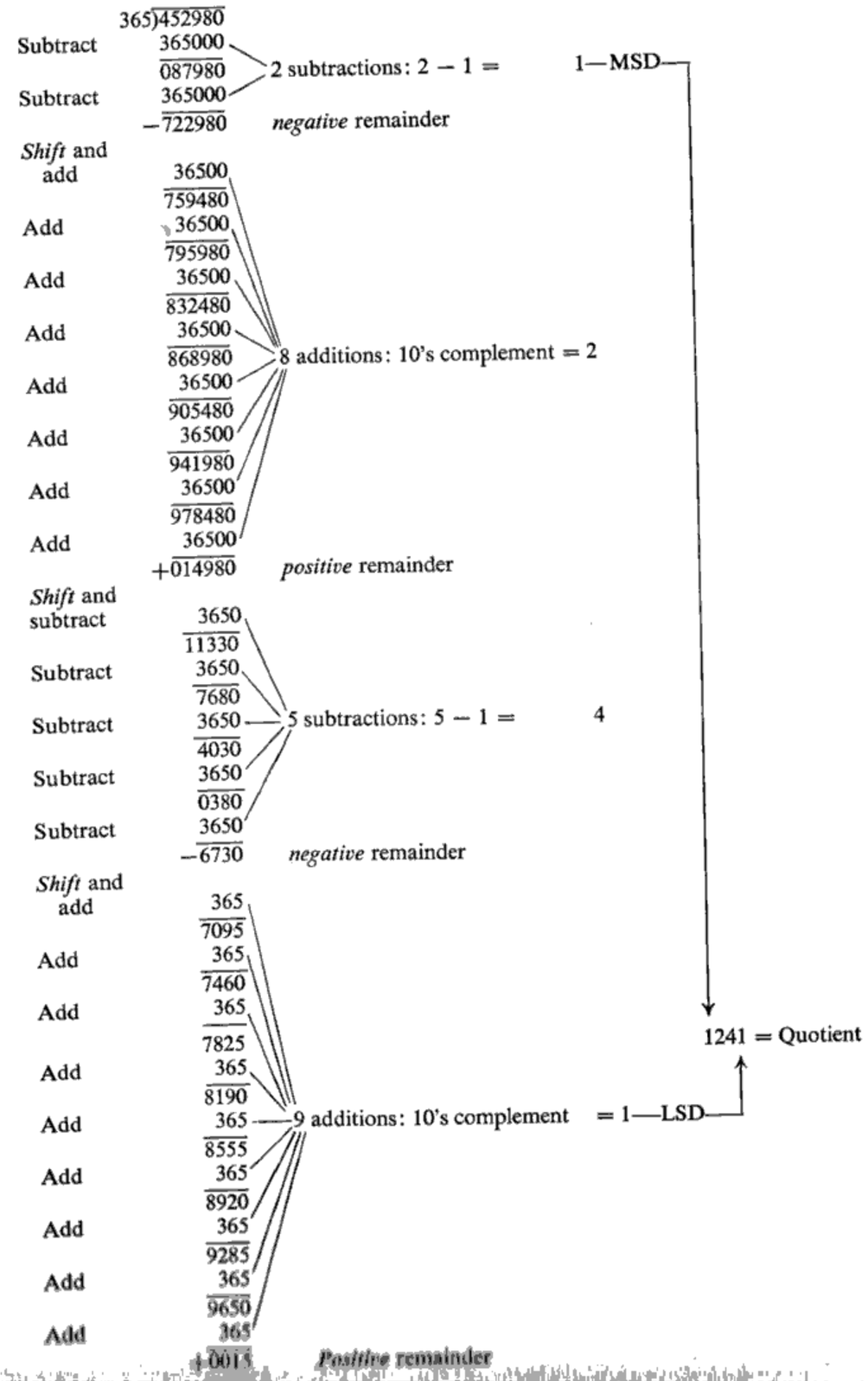


Note that the quotient digit in each case is one less than the number of subtractions required to yield a negative intermediate remainder.

Nonrestoring Method of Division

The nonrestoring method of division is similar to the restoring method except that, after the first series of subtractions, the next quotient digit is obtained by performing a series of additions, the following quotient digit by performing a series of subtractions, and so on. The method is called nonrestoring because intermediate remainders are not restored as in the previous method.

Example:



Here is the method. The divisor is repeatedly subtracted from the dividend until a *negative* remainder is produced. One less than the number of subtractions performed is the MSD of the quotient. The divisor is shifted one place to the right and added successively until a *positive* intermediate remainder is produced. The 10's complement of the number of additions performed is the second most significant quotient digit. The divisor is shifted to the right again and subtracted successively until a *negative* intermediate remainder is produced. One less than the number of subtractions performed is the next quotient digit. The divisor is shifted right again and added successively until a *positive* intermediate remainder is produced. The 10's complement of the number of additions performed is the fourth quotient digit, and so on.

Note that the negative remainders are the same as the negative remainders obtained from the corresponding series of subtractions in the restoring method; and that the positive remainders are the same as the corresponding remainders obtained after the restoring addition in the restoring method. So evidently the series of additions in the nonrestoring method accomplishes the same thing as the series of subtractions plus the restoring addition in the restoring method. Why is this so?

To find the answer to this question, let us compare the two methods starting at the subtraction that produces the first negative remainder. In the restoring method, after subtracting 365000, the entire 365000 is added back; then the divisor is shifted right to convert it to 36500, which is repeatedly subtracted to obtain the second most significant quotient digit. In the nonrestoring method, instead of completely restoring the 365000, the divisor is shifted right and the resultant 36500 is added successively. This means that 10 times 36500 is subtracted and 8 times 36500 is added; in effect, 36500 is subtracted twice. The total effective number of subtraction is thus equal to the 10's complement of the number of additions.

It is possible to perform the nonrestoring method of division by means of additions alone—by the addition of complements.

SQUARE ROOT

Three methods of obtaining the square root are discussed here: the pencil-and-paper method, the subtractive method, and the approximation method.

Pencil-and-Paper Method

The pencil-and-paper method of obtaining the square root of a number can be illustrated best by an example.

Example:

```
                 3  7   Square root
               √13 69   Square
                 9
Trial divisor  67) 4 69  First intermediate remainder
                   4 69
```

Note that before any calculation is performed the digits of the number whose square root is to be obtained are grouped in pairs. The most significant digit of the square root is then obtained by a process of trial and error: we look for a digit whose square is as close as possible to 13 without being greater than 13. After subtracting the square of 3—which is 9—from 13 and then bringing down the next pair of digits (69), we double 3 and record the 6 with a space to its right as a trial divisor. Again, by a process of trial and error, we seek a digit which can be placed to the right of the 3 under the square root sign and also to the right of the 6 in the trial divisor. This digit—in this case 7—must be so chosen that when it is multiplied by the trial divisor—in this case 67—we will get a number smaller than or equal to the intermediate remainder 469.

Subtractive Process

This method is easy to mechanize because it depends upon a simple rule and the steps in the rule are very much alike. The method is based upon an unusual relationship which is shown in Table 15.

TABLE 15

Relationship Between Equivalent Series of Squares and Square Roots

Square	Equivalent Series	Square Root (Number of Terms)
1	1	1
4	1 + 3	2
9	1 + 3 + 5	3
16	1 + 3 + 5 + 7	4
25	1 + 3 + 5 + 7 + 9	5
36	1 + 3 + 5 + 7 + 9 + 11	6
49	1 + 3 + 5 + 7 + 9 + 11 + 13	7

The number of terms combined is equal in every case to the square root of the number in question. This relationship has been proved to be universally true. This principle may be applied in practice, as shown by the following examples.

Example: Find the square root of 25

```
                25
Subtract         1
                24
Subtract         3
                21
Subtract         5 ——— 5 subtractions; the square root is 5
                16
Subtract         7
                 9
Subtract         9
                 0
```

Example:

```
                        √13 83. 84          SQUARE ROOT DIGIT
Subtract                  1
                         12
Subtract                  3 ——— 3 subtractions  3—MSD—
                          9
Subtract                  5
                          4 83
  Add 1 to last subtra-
    hend (5) and shift;
  Repeat odd series in
    units position
Subtract                 61
                         4 22
Subtract                 63
                         3 59
Subtract                 65
                         2 94
Subtract                 67 ——— 7 subtractions  7
                         2 27
Subtract                 69
                         1 58
Subtract                 71
                          87
Subtract                 73
                          14 84
  Add 1 to last subtra-
    hend (73) and shift;
  Repeat odd series in                                  37.2 = Square
    units position                                          root
Subtract                 7 41
                         7 43  2 subtractions  2—LSD
Subtract                 7 43
                           00
```

When the number whose square root is to be found consists of more than two digits, the digits are marked off into groups of two, just as is done in the pencil-and-paper method. The odd numbers 1, 3, 5, etc., are subtracted successively from the most significant pair of digits until a remainder is reached from which cannot be subtracted without producing a negative term. The number of subtractions performed is the MSD of the square root. The next pair of digits of the square is brought down. Then a 1 is added to the last subtrahend used; with this digit as the 10's digit and the series 1, 3, etc., as the units digits, we subtract successively again until a remainder is produced from which cannot be subtracted without producing a negative number. The number of subtractions performed is the second significant digit of the square root. The next pair of digits of the square is brought down, a 1 is added to the LSD of the last subtrahend, and with the series 1, 3, etc., placed to the right of this number, we again subtract successively to obtain the next digit in the square root.

Instead of subtracting, we may add complements. A machine can easily execute this process by performing a series of additions and shifts; a counter can keep track of the number of additions performed.

Approximation Formula

The square root of a number may be obtained by means of the following approximation formula:

$$y_{i+1} = y_i + .5\left(\frac{a}{y_i} - y_i\right)$$

New square root = previous square root

$$+ \text{ half } \left(\frac{\text{square}}{\text{previous square root}} - \text{previous square root}\right)$$

At first a guess of the square root is made, and this value is substituted for the "previous square root" in the above formula. The calculated "new square root" is then substituted for the previous square root, and the formula is applied once more. This procedure is repeated, and each time the new square root becomes closer and closer to the actual value of the square root.

Example: Find $\sqrt{64}$; $a = 64$; first guess for $y_i = y_0 = 2$.

First approximation

$$y_1 = y_0 + .5\left(\frac{a}{y_0} - y_0\right)$$

$$y_1 = 2 + .5(\tfrac{64}{2} - 2) = 2 + .5(30) = 17$$

Second approximation

$$y_2 = y_1 + .5\left(\frac{a}{y_1} - y_1\right)$$

$$y_2 = 17 + .5(\tfrac{64}{17} - 17) = 17 + .5(-13.235) = 10.382$$

Third approximation

$$y_3 = y_2 + .5\left(\frac{a}{y_2} - y_2\right)$$

$$y_3 = 10.382 + .5\left(\frac{64}{10.382} - 10.382\right) = 10.32 + .5(-4.218) = 8.273$$

Fourth approximation

$$y_4 = y_3 + .5\left(\frac{a}{y_3} - y_3\right)$$

$$y_4 = 8.273 + .5\left(\frac{64}{8.273} - 8.273\right) = 8.273 + .5(-.5370) = 8.004$$

Fifth approximation

$$y_5 = y_4 + .5\left(\frac{a}{y_4} - y_4\right)$$

$$y_5 = 8.004 + .5\left(\frac{64}{8.004} - 8.004\right) = 8.004 + .5(-.008) = 8$$

Sixth approximation

$$y_6 = y_5 + .5\left(\frac{a}{y_5} - y_5\right)$$

$$y_6 = 8 + .5(\tfrac{64}{8} - 8) = 8 + .5(0) = 8$$

This procedure for finding the square root is not usually built into a computer. Any computer which can add, subtract, multiply, and divide may be programmed to perform this operation.

BINARY ARITHMETIC

The rules of binary addition are similar to those for decimal addition, except that a binary carry instead of a decimal carry is obtained.

Example:

```
  1 1 0 1
  1 1 0 0
  1         Carry
-------
1 1 0 0 1
```

The subtraction required as part of the algebraic addition of binary numbers of unlike signs is similar to decimal subtraction. When subtracting a larger bit (1) from a smaller bit (0), borrows are used as in decimal subtraction.

Example:

```
1 0 1 0
0 1 0 1
1   1     Borrow
-------
0 1 0 1
```

Subtraction by the use of complements is made easy since the 1's complement of a number is obtained by converting all 0's to 1's and all 1's to 0's.

Example:

```
+1010 ------------------------> 1010
-0101 --1's complement--------> 1010
                        1 1     Carry
                       -----
                       10100
                        ____
                        |  |
                        |  |
                        |  |
```

Example:

	NO. OF PLACES POINT IS FROM RIGHT
110110.11	2
×.111	+3
11011011	
11011011	
11011011	
101111.11101	5

In division the rule for determining the decimal (binary) point depends upon the relative values of the divisor digits and of an equivalent number of digits at the most significant end of the dividend. If the value of the divisor digits is greater than the value of an equivalent number of digits at the most significant end of the dividend, do the following:

1. Count the number of places from the left to where the decimal (binary) point appears in the dividend.
2. Count the number of places from the left to where the decimal (binary) point appears in the divisor.
3. Subtract the latter from the former. The difference is the number of places from the left where the decimal (binary) point should be placed in the quotient.

Example:

Dividend 324.0 } Divisor digits
Divisor 4886 } greater

NO. OF PLACES POINT IS FROM LEFT

.0x
4886.)324.000

Dividend	3
Divisor	−4
Quotient	−1

Minus one means that the point is not one place to the right of the most significant nonzero digit but one place to the left of it.

Example:

Dividend 1010 10. } Divisor digits
Divisor 1110 } greater

NO. OF PLACES POINT IS FROM LEFT

1x.xxx
1110.)101010.000

Dividend	6
Divisor	−4
Quotient	2

If the value of the divisor digits is smaller than the value of an equivalent number of digits in the dividend, the subtraction is performed as before, but this time the remainder *plus one* is the number of places the point in the quotient is from the left.

Example:

Dividend 488 648. } Divisor digits
Divisor 244 } smaller

NO. OF PLACES POINT IS FROM LEFT

xxxx.
244.)488648.000

Dividend	6
Divisor	−3
	3
	+1
Quotient	4

Example:

Dividend 11.000 } Divisor digits
Divisor 10101 } smaller

NO. OF PLACES POINT IS FROM LEFT

.00x
10101.)11.0000

Dividend	2
Divisor	−5
	−3
	+1
Quotient	−2

The decimal (binary) point need not be moved directly. We may express numbers with the aid of the base and exponent as follows:

$$3000. = .3 \times 10^4$$
$$.00324 = .324 \times 10^{-2}$$
$$654.127 = .654127 \times 10^3$$

The decimal (binary) point may be located indirectly by properly changing the exponent of the base. The rules for dealing with the exponents are as follows.

To add two numbers follow these rules.

1. Make both exponents the same.
2. Add the significant digits.
3. The new sum consists of the sum of the significant digits multiplied by the base to the common exponent.

Example:

$$\begin{aligned} 654.127 &= .654127 \times 10^3 \\ 3.120 &= .00312 \times 10^3 \\ \hline 657.247 &= .657247 \times 10^3 \end{aligned}$$

Example:

$$\begin{aligned} 11101.101 &= .11101101 \times 2^5 \\ .0011 &= .000000011 \times 2^5 \\ \hline 11101.1101 &= .111011101 \times 2^5 \end{aligned}$$

Multiplication is fairly easy. Multiply the digits in the ordinary way, but add the exponents.

Example:

$$\begin{aligned} 65.4 &= .654 \times 10^2 \\ \times .31 &= .31 \times 10^0 \\ \hline 654 & \quad 654 \\ 1962 & \quad 1962 \\ \hline 20.274 &= .20274 \times 10^2 \end{aligned}$$

Example:

$$\begin{aligned} 11.011 &= .11011 \times 2^2 \\ \times .011 &= \times .11 \times 2^{-1} \\ \hline 11011 & \quad 11011 \\ 11011 & \quad 11011 \\ \hline 1.010001 &= .101001 \times 2^1 \end{aligned}$$

In division the digits are divided normally but the exponent of the divisor is subtracted from the exponent of the dividend.

Example:

$$\frac{488648}{200} = 2443$$

$$\begin{aligned} 488648 &= .488648 \times 10^6 \\ 200 &= .200 \times 10^3 \\ \hline .488648 &= 2.443 \times 10^3 \\ .200 & \end{aligned}$$

Example:

$$\frac{10101.010}{10.1} = 1000.1$$

$$\begin{aligned} 10101.010 &= .10101010 \times 2^5 \\ 10.1 &= .101 \times 2^2 \\ \hline .10101010 &= 1.0001 \times 2^3 \\ .101 & \end{aligned}$$

CHECKING

The forbidden combination check, the parity check, and other types of redundancy checks are based on the rules of the language. It is also possible to perform checking based on arithmetic. Several arithmetic checking schemes utilized by computers are discussed in the following.

Duplication

We all do this type of checking, especially when adding columns of digits. We repeat the addition. If the sum obtained the second time is the same as the sum obtained the first time, we assume the addition is correct. If the second sum is different from the first sum, we assume we made an error. This procedure is applicable to any arithmetic operation.

Inverse Arithmetic

An arithmetic operation may be checked by performing its inverse. Subtraction is the inverse of addition, division is the inverse of multiplication, and taking the square root is the inverse of squaring. The method is obvious from the following examples.

Example:

SUBTRACTION		CHECK BY ADDITION	
7	Minuend	3	Difference
−4	Subtrahend	+4	Subtrahend
3	Difference	7	Minuend

Example:

DIVISION		CHECK BY MULTIPLICATION	
Dividend	$\frac{100}{5} = 20$ Quotient	20	Quotient
Divisor		×5	Divisor
		100	Dividend

Example:

SQUARE ROOT		CHECK BY SQUARING	
5	Square root	5	Square root
		×5	Square root
25	Square	25	Square

Casting Out 9's and 7's

Casting out 9's is a checking scheme which has been used by bookkeepers for years. In this scheme every number taking part in an arithmetic

operation is divided by 9; the remainder is therefore a number in modulus 9. This remainder is called the Residue Modulus Nine or RMN.

Example: Find RMN of 236.

$$\frac{236}{9} = 26\tfrac{2}{9} \qquad \text{RMN} = 2$$

Example: Find RMN of 421.

$$\frac{421}{9} = 46\tfrac{7}{9} \qquad \text{RMN} = 7$$

Checking is accomplished by performing the same operations on the RMN's as are performed on the numbers. The following rules apply to the RMN's:

1. To check addition, add the RMN's of the numbers. The result should be equal to the RMN of the sum.

Example:

NUMBERS		RMN
2534		5
+1273		+4
3807	RMN of sum =	0 = Sum of RMN's

Note that in the modulus 9 scheme, the digit after 8 is not 9, but 0.

2. To check subtraction, subtract the RMN's of the numbers. The result should be equal to the RMN of the difference.

Example:

NUMBERS		RMN
2534		5
−1273		−4
1261	RMN of difference =	1 = Difference of RMN's

3. To check multiplication, multiply the RMN's of the numbers. The result should be equal to the RMN of the product.

Example:

NUMBERS		RMN
253		1
×75		×3
18975	RMN of product =	3 = Product of RMN's

4. To check division, multiply the RMN of the quotient by the RMN of the divisor and add the RMN of the remainder. The result should be the RMN of the dividend.

Example:

$$\frac{6479}{21} = 308\tfrac{11}{21}$$

NUMBER		RMN
308	Quotient	2
×21	Divisor	×3
6468		6
+11	Remainder	+2
6479	RMN of dividend =	8 = (RMN of quotient) × (RMN of divisor) + RMN of remainder

Binary numbers may be checked by a scheme of casting out 7's. To obtain the Residue Modulus Seven (RMS) of a binary number, the number is divided by binary 7 or 111; the remainder is the RMS.

Example: Find the RMS of 1010110.

$$\begin{array}{r} 1\ 100 \\ 111\overline{)1\ 010\ 110} \\ \underline{111} \\ 011\ 110 \\ \underline{11\ 1} \\ 010 \end{array} = \text{RMS}$$

The rules for checking in the RMS system are similar to those used in the RMN system.

Example of Addition:

NUMBERS		RMS
1 010 110		010
+010 111		010
1 101 101	RMS of sum =	100 = Sum of RMS's

SUMMARY

1. In algebraic addition, if the signs are the same, the terms are added normally. If the signs of the augend and addend are different, the complement of the addend is added to the augend.

a. If a carry is produced from the most significant digit position, it is ignored if the 10's or 2's complement has been taken; an end-around carry is performed if the 9's or 1's complement has been taken.
b. If no carry is produced, the complement—10's, 9's, 2's, or 1's—of the sum is taken.

2. Most systems of multiplication are based upon the pencil-and-paper method. In the left-and-right-hand component method, the product of each pair of digits is divided into two components: left and right; then the components

are summarized separately and combined to give the product. In the method of repeated addition, the multiplicand is repeatedly added a number of times as determined by the multiplier digits. Each successive group of additions is shifted one position to the left.

3. Division is usually performed by a system of repeated subtraction. In the restoring method, each group of subtractions is taken to the point where the remainder changes sign, then an addition is performed to restore the remainder. At that point the divisor is shifted right, and the next series of subtractions is performed, etc. In the nonrestoring method, first a series of subtractions is performed; when the remainder is negative, a series of additions is performed; when the remainder turns positive a series of subtractions is performed, etc.

4. Arithmetic in binary is very simple because of the simple addition and multiplication tables.

5. Arithmetic may be checked by a system of casting out 9's or 7's. RMN or RMS of each factor is obtained by dividing by 9 or 7. The RMN and RMS are used as follows:

a. The sum of the RMN (RMS) of two numbers is equal to the RMN (RMS) of the sum.
b. The difference of the RMN (RMS) of two numbers is equal to the RMN (RMS) of the difference.
c. The product of the RMN (RMS) of two numbers is equal to the RMN (RMS) of the product.
d. If the RMN (RMS) of the quotient is multiplied by the RMN (RMS) of the divisor and added to the RMN (RMS) of the remainder, the result should be the RMN (RMS) of the dividend.

CHAPTER 4

Machine logic

Logic is the cement of language. Formal logic is the cement holding the propositions of syllogisms together. In the more general case bivalued logic applies to the relationships among classes. By using letters of the alphabet to designate bivalued classes and mathematical symbols to represent logical connectives, an algebra of logic can be developed.

This logical algebra can be applied to mechanize the rules of arithmetic (or anything else) in the following way. The conditions of the problem are summarized in what is known as a truth table. From this truth table are found the algebraic equation or equations expressing the relationships among all the classes of the problem. This equation or set of equations is then converted to a logical block diagram, which is essentially a plan for the construction of the device.

But there is more to the plan of a practical device than what can be specified by the rules of algebra of logic. In the computer, propositions are represented by signals, and amplifiers are needed to maintain the strength of these signals. In formal logic time is not considered; in practical machine logic time is a very important element, and its consideration leads to the use of storage devices. Practical machine logic therefore consists of amplifiers and storage elements in addition to elements representing the fundamental logical connectives.

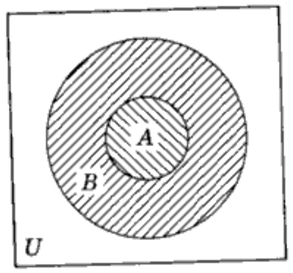
BASIC ELEMENTS OF FORMAL LOGIC

Graphical Logic

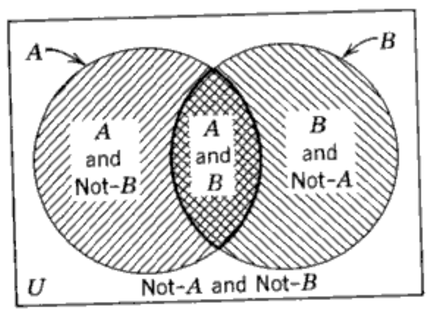
Logic deals with relationships among classes. For instance:

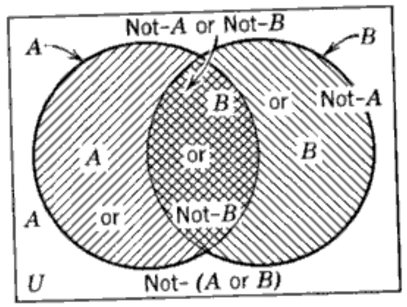
All men who go to college have a high IQ.
Jack is a man who is attending college.
Therefore, Jack has a high IQ.

The above is a syllogism having three classes: Jack, men who attend college, and men with high IQ. This simple syllogism may therefore be expressed graphically as shown in Fig. 13. The illustration shows that class *A*, or Jack, is a member of class *B*, or the class of men attending college, which in turn is a member of *U*, the universal class consisting of all men with high IQ.



U = Universal class of men with high IQ
B = Class of men attending college
A = Jack





U = Men
A = Men attending college
B = Men wearing gray hats

Fig. 13. Graphical logic.

If only one property such as college attendance is considered, the universal class may be divided into two classes: those who attend and those who do not attend—class *A* and class Not-*A*. If two properties of the universal class—such as college attendance (*A*) and wearing of gray hats (*B*)—are considered, then several classes are obtained as shown in the illustration. By combining the fundamental classes *A*, Not-*A*, *B*, and Not-*B*, by means of the AND connectives we obtain:

1. *A* and Not-*B*—The class of men who attend college and who do not wear gray hats.
2. Not-*A* and *B*—The class of men who do not attend college and who wear gray hats.

3. *A* and *B*—The class of men who attend college and who wear gray hats.
4. Not-*A* and Not-*B*—The class of men who do not attend college and of men who do not wear gray hats.
5. Not-(*A* and *B*)—The class of men who are not both attending college and wearing gray hats.

These classes are shown in the figure. *A* and Not-*B* is represented by the portion of the *A* circle to the left of the *B* circle boundary; Not-*A* and *B* is represented by the portion of the *B* circle to the right of the *A* circle boundary; *A* and *B* is represented by the cross-hatched portion; Not-*A* and Not-*B* is represented by the area outside the *A* and *B* circles; and Not-(*A* and *B*) is represented by everything outside the center cross-hatched area.

Note that the AND connective is restrictive: as a rule, there are fewer members in a class containing an AND element than there are in any of the fundamental classes.

Combining the fundamental classes by means of an OR connective we obtain:

1. *A* or Not-*B*—The class of men who attend college or who do not wear gray hats.
2. Not-*A* or *B*—The class of men who do not go to college or of men who wear gray hats.
3. *A* or *B*—The class of men who attend college or of men who wear gray hats.
4. Not-(*A* or *B*)—The class of men who are not either attending college or wearing gray hats.
5. Not-*A* or Not-*B*—The class of men who are not attending college or of men who are not wearing gray hats.

These classes are also shown in Fig. 13. *A* or Not-*B* is represented by everything within circle *A* plus everything outside both circles; Not-*A* or *B* is represented by everything within circle *B* plus everything outside both circles; *A* or *B* is represented by the area covered by both circles; Not-(*A* or *B*) is represented by everything outside both circles; Not-*A* or Not-*B* is represented by everything outside the cross-hatched area.

Note that the OR relation used here is the INCLUSIVE OR as distinguished from the EXCLUSIVE OR. The difference between the two can be seen from the following interpretations of *A* or *B*:

A or *B* (INCLUSIVE OR)—*A* or *B* or both *A* and *B*.
A or *B* (EXCLUSIVE OR)—*A* or *B*, but not both *A* and *B*.

In this book, OR refers to INCLUSIVE OR, unless specifically stated otherwise.

Mathematical Logic

From the above graphical presentation it appears that relationships among classes may be presented by means of three logical functions:

AND, OR, and NOT

The same is true not only for two fundamental classes but also when three or more fundamental classes are combined. The more fundamental classes there are, the more difficult it is to express the many logical relationships graphically. But the same ideas may be expressed easily mathematically, as the following equalities show:

$$A \text{ AND } B \text{ AND } C = A \cdot B \cdot C$$
$$A \text{ OR } B \text{ OR } C = A + B + C$$
$$\text{Not-}A = \bar{A}$$

With the aid of the above symbols, we may form logical expressions and operate on them mathematically. However, a few simple identities and laws must first be presented. The following are some identities involving AND, OR and NOT functions:

$A \cdot 0 = 0$ A class with no members (0) can not intersect a class with members (A).

$U \cdot A = A$ Members of class A and of the universal class are the same only in class A.

$A \cdot \bar{A} = 0$ It is impossible for a class to intersect its complement .

$A + A = A$ Class A or class A equals class A.

$0 + A = A$ The class composed of members of the empty class or of class A is class A.

$U + A = U$ The class composed of members of the universal class or of class A is the universal class.

$A + \bar{A} = U$ The universal class consists of members of a class or of members of the complement of this class.

$\bar{\bar{A}} = A$ The complement of the complement of a class is the same as the class itself.

Note that both identity $A \cdot 0 = 0$ and identity $A + \bar{A} = U$ follow directly from the meaning of bivalued logic. Identity $\bar{\bar{A}} = A$ is analogous to the grammatical rule which states that two negatives make an affirmative.

In addition to the above identities, there are three laws which all logical expressions follow. These laws are illustrated by means of examples as follows:

1. *The commutative law:*

$$A + B = B + A$$
$$A \cdot B = B \cdot A$$

2. *The associative law:*

$$A + (B + C) = (A + B) + C$$
$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

3. *The distributive law:*

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$
$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

These laws are similar to those used in ordinary arithmetic—all except the last one: $2 + (3 \times 4)$ is not equal to $(2 + 3) \times (2 + 4)$.

Logical Relationships

There are other logical functions besides AND, OR, and NOT. But all other connectives can be shown to be made up of combinations of these three logical functions. Several such functions are discussed.

1. EXCLUSIVE OR—The EXCLUSIVE OR may be expressed as "A or else B." This phrase is interpreted to mean A or B but not both A and B, or mathematically,

$$(A + B) \cdot \overline{(A \cdot B)}$$

2. IF AND ONLY IF—"A if and only if B" means that, if we have A, we also have B, but if we do not have A, we also do not have B, or mathematically,

$$(A \cdot B) + (\bar{A} \cdot \bar{B})$$

3. IF, THEN—"If A then B" means that, if we have A, we must have B too, but if we do not have A, we may or may not have B, or mathematically,

$$(A \cdot B) + (\bar{A} \cdot B) + (\bar{A} \cdot \bar{B})$$

4. INHIBIT—"A is inhibited by B" means that A occurs providing B is not present, or Not-B *is* present; mathematically this is expressed as

$$A \cdot \bar{B}$$

Not even the three functions AND, OR, and NOT are independent of each other. There is a relationship among these three which can be developed from the graphs shown in Fig. 13. The area covered by both circles is $A + B$. This means that everything outside this area is $\overline{(A + B)}$. On the other hand, everything outside circle A is $\bar{A}$, everything outside circle B is

$\bar{B}$, and, therefore, everything outside both circles is $\bar{A} \cdot \bar{B}$. Thus we arrive at the conclusion that

$$\overline{(A + B)} = (\bar{A} \cdot \bar{B})$$

Now let us look at the area labeled $A \cdot B$. Everything outside this area is $\overline{(A \cdot B)}$. On the other hand, everything outside A is $\bar{A}$, and everything outside B is $\bar{B}$. $\bar{A}$ includes the entire area outside both circles plus the part of B not intersected by A. $\bar{B}$ includes the entire area outside both circles plus the part of A not intersected by B. Therefore everything outside the overlap portion is $\bar{A} + \bar{B}$. Thus we arrive at the conclusion that

$$\overline{(A \cdot B)} = \bar{A} + \bar{B}$$

From the above two relationships we arrive at the following rule:

To obtain the inverse of a function, replace all AND's with OR's and all OR's with AND's, and invert the terms.

$$\overline{(A + B)} = \bar{A} \cdot \bar{B} = \text{INVERTED OR}$$
$$\overline{(A \cdot B)} = \bar{A} + \bar{B} = \text{INVERTED AND}$$
$$\overline{(A \cdot \bar{B})} = \bar{A} + B = \text{INVERTED INHIBIT}$$

As will be shown in Section II, the inverted functions occur often, especially in circuits which use amplifiers that invert the phase of the signal.

The inverse rule can be applied in another manner. If only one property A is considered, the universal class may be divided into two classes.

$$U = A + \bar{A}$$

If two properties A and B are considered, the universal class may be divided into the following four classes:

$$U = (\bar{A} \cdot \bar{B}) + (A \cdot B) + (\bar{A} \cdot B) + (A \cdot \bar{B})$$

To find the value of $(\bar{A} \cdot \bar{B}) + (A \cdot B)$, we may say that it is equal to the complement of the remaining terms; this is similar to saying that in a modulo-10 system, 3 is equal to the complement of the remaining digits, or 3 equals the complement of 7. Therefore,

$$(\bar{A} \cdot \bar{B}) + (A \cdot B) = \overline{(\bar{A} \cdot B) + (A \cdot \bar{B})}$$

By replacing on the right-hand side all AND symbols with OR symbols and all OR symbols with AND symbols, and at the same time inverting each individual term, we obtain

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

Similarly,

$$(\bar{A} \cdot B) + (A \cdot \bar{B}) = \overline{(A \cdot B) + (\bar{A} \cdot \bar{B})}$$

By replacing on the right-hand side all AND symbols with OR symbols and all OR symbols with AND symbols, and at the same time inverting all the individual terms, we obtain

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

Therefore, AND terms linked together by means of OR symbols may be converted to OR terms linked together by means of AND symbols.

Truth Tables

To apply the rules of the algebra of logic to a problem, we make use of a truth table. A truth table is a statement of all the conditions of a problem in tabular form. To facilitate the formation of this table, follow the convention that ONE stands for the YES and ZERO stands for the NO of a proposition. Stated differently, a ONE indicates the presence of a certain class, A, and ZERO represents the presence of the opposite class, $\bar{A}$. From the truth table it is easy to develop the logical expressions stating all the conditions in algebraic terms.

To understand fully the meaning of a truth table, let us write one which represents the operation of a binary comparator. A binary comparator is a device which looks at two bits and produces the answer to the following question: "Are the two bits unlike?" A ONE output indicates "Yes, they are unlike"; a ZERO output indicates "No, they are not unlike, they are alike." If the input bits are labeled A and B, and the output bit K, then the conditions of the problem are summarized in Table 16.

TABLE 16

Truth Table for Binary Comparator

Bit (A)	Bit (B)	Comparator Output (K)
0	0	0
0	1	1
1	0	1
1	1	0

To find the logical equation for the binary comparator, we ask ourselves "Under what conditions does the comparator produce a ONE?" The answer is given on lines two and three: when A equals ZERO and B equals ONE or $\bar{A} \cdot B$; or when A equals ONE and B equals ZERO or $A \cdot \bar{B}$. Therefore, the algebraic expression for the binary comparator is

$$K = (\bar{A} \cdot B) + (A \cdot \bar{B})$$

As another example, let us take a half-adder. A half-adder is a binary adder that has two inputs, augend and addend. It is called a half-adder because it does only half the job of addition; a full adder adds three bits, the augend, addend, and carry from the last bit position. The half-adder addition table is presented in Table 17.

TABLE 17

Truth Table for Half-Adder

Augend (A)	Addend (B)	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

To express the operation of a half-adder, we need two equations: one for the sum and one for the carry. From the table we see that the sum is ONE under the same conditions as for the binary comparator. Therefore the sum is

$$S = (\bar{A} \cdot B) + (A \cdot \bar{B})$$

The table also shows that a carry is produced or $C =$ ONE only when both A and B are ONE, or

$$C = A \cdot B$$

Since the equation for the 2-bit sum represents roughly half the job performed by the half-adder, a circuit which performs this operation is often called a quarter-adder. Now, we know that

$$S = K = (\bar{A} \cdot B) + (A \cdot \bar{B})$$

We know also that

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

and that

$$(\bar{A} + \bar{B}) = (\overline{A \cdot B})$$

The equation for the 2-bit sum then reduces to

$$S = K = (A + B) \cdot (\overline{A \cdot B})$$

which is exactly the same as the equation for the exclusive OR function. Therefore, the binary comparator, quarter-adder, and EXCLUSIVE OR are all one and the same.

The truth table for a full 3-input binary adder is given in Table 18.

TABLE 18

Truth Table for Binary Adder

Augend (A)	Addend (B)	Previous Carry (C')	Sum (S)	Carry (C)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

There are eight lines to this table since these three variables may be combined in eight ways. The associated sum and carry bits are shown for each combination. The second, third, fifth, and eighth lines indicate separate conditions under which the sum S is equal to ONE. Therefore,

$$S = (\bar{A} \cdot \bar{B} \cdot C') + (\bar{A} \cdot B \cdot \bar{C}') + (A \cdot \bar{B} \cdot \bar{C}') + (A \cdot B \cdot C')$$

or S equals ONE when the three inputs are

0, 0 and 1 *or* 0, 1 and 0 *or* 1, 0 and 0 *or* 1, 1 and 1

The fourth, sixth, seventh, and eighth lines indicate alternative conditions under which the carry C is equal to ONE. Therefore,

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

or C equals ONE when the three inputs are

0, 1 and 1 *or* 1, 0 and 1 *or* 1, 1 and 0 *or* 1, 1 and 1

As another example, the 9's complements of numbers in the 5421 system are to be found. Table 19 is a summary of all the conditions that must be met. The 9's complement of each number represented by the four bits in the left-hand column is given by the number represented by the four bits in the right-hand column and vice versa.

To find the logical expressions summarizing the information in this table we determine the conditions in the left-hand columns under which a ONE appears in each of the four right-hand columns. Four logical expressions are needed, one for each bit position.

TABLE 19

Truth Table for 5421 Complementer

Number					Complement				
Order of Bit Weight / Decimal	A_4 5	A_3 4	A_2 2	A_1 1	A'_4 5	A'_3 4	A'_2 2	A'_1 1	Order of Bit Weight / Decimal
0	0	0	0	0	1	1	0	0	9
1	0	0	0	1	1	0	1	1	8
2	0	0	1	0	1	0	1	0	7
3	0	0	1	1	1	0	0	1	6
4	0	1	0	0	1	0	0	0	5

From the truth table we see that A'_1 is equal to ONE whenever A_1 is equal to ONE, and at no other time. Therefore,

$$A'_1 = A_1$$

A'_2 is equal to ONE on lines two and three. On these two lines on the left side we note that the bits in the first two orders are either ZERO and ONE or ONE and ZERO. Expressed algebraically,

$$A'_2 = \bar{A}_2 \cdot A_1 + A_2 \cdot \bar{A}_1$$

A'_3 is equal to ONE on line one. On this line bits A_1, A_2, A_3, and A_4 are all ZERO. Therefore,

$$A'_3 = \bar{A}_4 \cdot \bar{A}_3 \cdot \bar{A}_2 \cdot \bar{A}_1$$

If $\bar{A}_4$ is removed from this expression, we still have a unique expression for A'_3. The $\bar{A}_4$ is thus redundant, and it is enough to state

$$A'_3 = \bar{A}_3 \cdot \bar{A}_2 \cdot \bar{A}_1$$

Lastly, A'_4 is equal to ONE whenever A_4 is equal to ZERO or

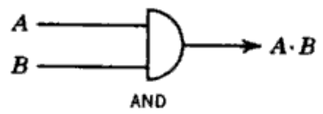
$$A'_4 = \bar{A}_4$$

The four logical equations derived above express all the conditions necessary to produce a 9's complement for all numbers between 0 and 4. The same equations hold for the complementation of numbers between 5 and 9, as can readily be seen by inspecting the truth table.

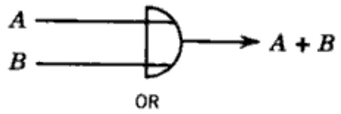
Logical Block Diagrams

Logical expressions relate symbols representing classes by means of AND, OR and NOT functions. These mathematical expressions may be represented by means of equivalent AND, OR, and NOT blocks in a logical

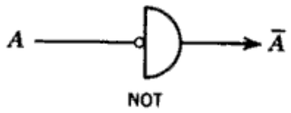
A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1



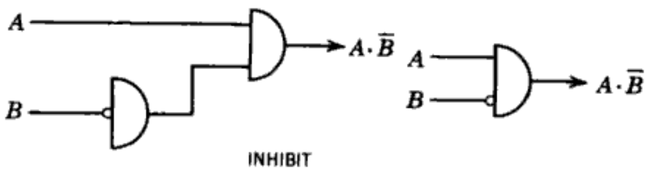
A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1



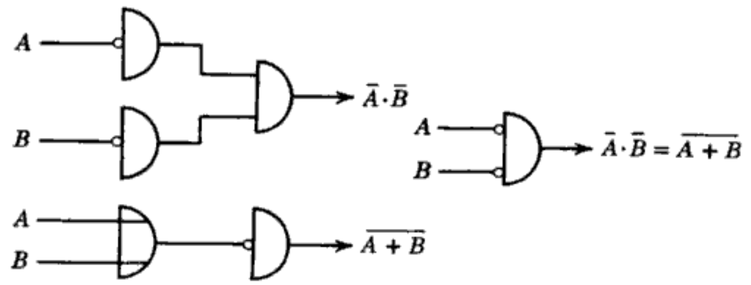
A	$\bar{A}$
0	1
1	0



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



A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0



A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

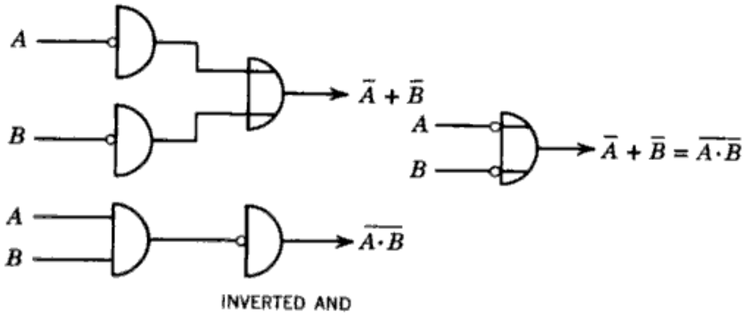


Fig. 14. Fundamental logical blocks.

block diagram. As will be seen in Section II, several physical devices are available which correspond closely in their operations to these logical blocks. Thus, the logical block diagram is a plan from which the actual hardware may be designed and built.

Figure 14 shows the relationships among several logical blocks and their associated truth tables and logical expressions. The first three blocks are primary: the most complicated logical expressions may be built from them. The INHIBIT function $A \cdot \bar{B}$ may be represented by a NOT block which

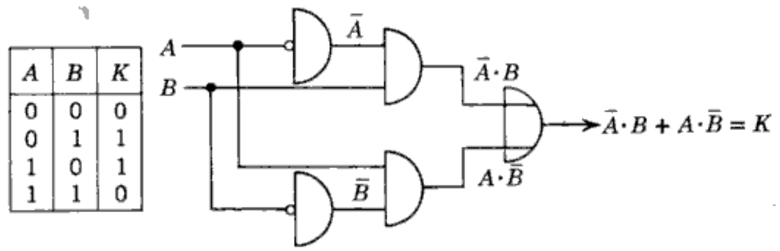


Fig. 15. Binary comparator.

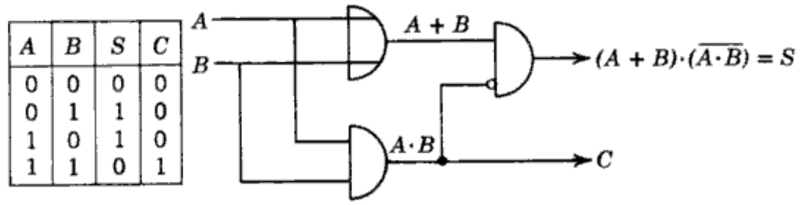


Fig. 16. Half-adder.

converts B to $\bar{B}$ and an AND block which combines A and $\bar{B}$. These two blocks are equivalent to the one block shown. The circle at the end of input line B represents an inhibit signal: whenever B is present, it inhibits the flow of signal A.

The INVERTED OR may be drawn in two ways. In the first diagram both A and B are inverted and then fed to an AND block; this produces $\bar{A} \cdot \bar{B}$. In the second diagram, A and B are ORed and then fed through an inverter; this produces $\overline{A + B}$. Both block diagrams are equivalent and may be represented compactly by two inhibit signals applied to an AND block as shown.

The INVERTED AND may similarly be drawn in two ways. In the first diagram, A and B are individually inverted and then fed through an OR to produce $\bar{A} + \bar{B}$. In the second diagram, A and B are ANDed and then inverted to produce $\overline{A \cdot B}$. Both block diagrams are equivalent and may be represented compactly by two inhibit signals applied to an OR block as shown.

With the aid of these fundamental logical blocks we can convert any logical expression into a logical block diagram. The binary comparator, quarter-adder, or EXCLUSIVE OR is shown in block diagram form in Fig. 15. By substituting for A and B, in turn, each of the four combinations shown and following the signals through, it can be seen that this block diagram does perform everything required of it by the truth table.

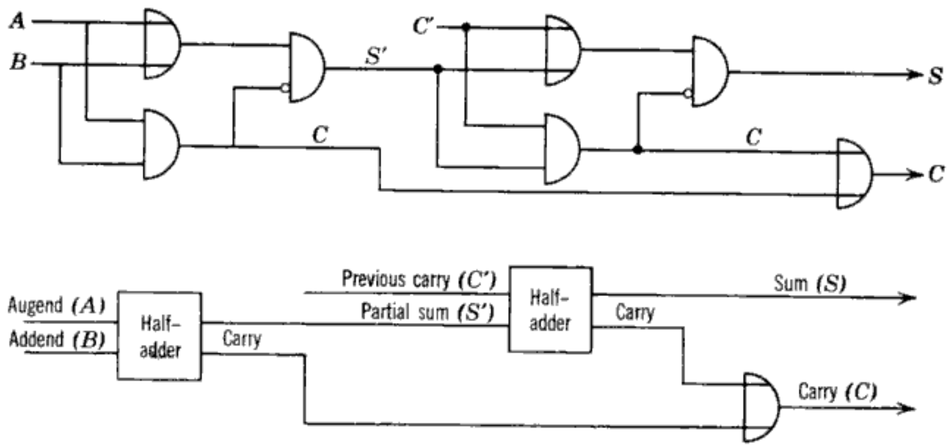


Fig. 17. Binary adder composed of two half-adders.

The block diagram of the half-adder (Fig. 16) illustrates why the following expression for the sum:

$$S = (A + B) \cdot (\overline{A \cdot B})$$

is preferable to

$$S = (\bar{A} \cdot B) + (A \cdot \bar{B})$$

In the former, the carry ($C = A \cdot B$) is produced as part of the formation of the sum. Therefore, there is a saving in the number of logical blocks needed.

Figure 17 shows how two half-adders may be combined to form a full binary adder. One half-adder adds the augend and addend to produce a partial sum and carry. The other half-adder adds the partial sum and the previous carry to produce the sum and a carry. The carrys from both half-adders are combined in an OR block to produce a carry which is applied to the addition of bits in the following bit position.

If we proceed in a straightforward manner to develop a 3-input adder, we develop the truth table shown in Fig. 18. From the resultant algebraic expressions, we form the block diagram shown. Although it looks entirely different from the block diagram in Fig. 17, the two are equivalent in function.

Figure 19 shows the block diagram for the 5421 complementer.

A	B	C'	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

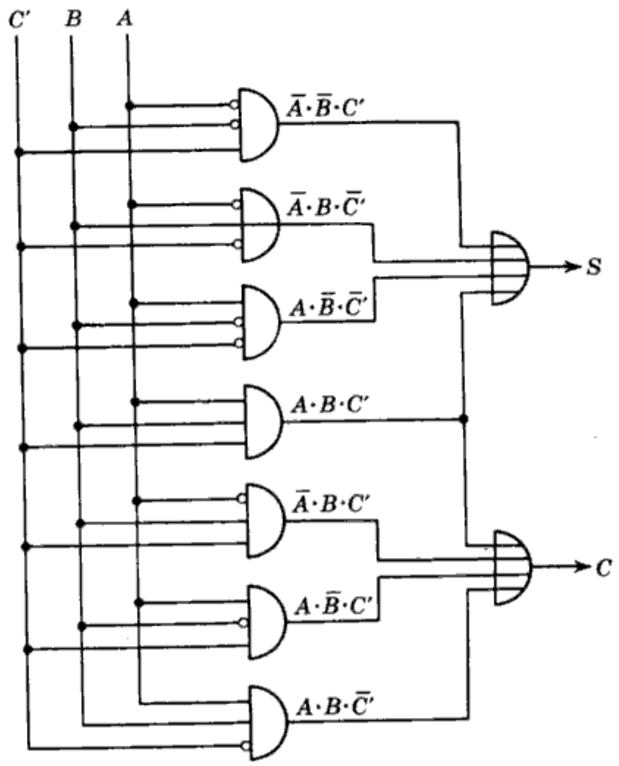


Fig. 18. Binary adder.

A_4	A_3	A_2	A_1	A'_4	A'_3	A'_2	A'_1
0	0	0	0	1	1	0	0
0	0	0	1	1	0	1	1
0	0	1	0	1	0	1	0
0	0	1	1	1	0	0	1
0	1	0	0	1	0	0	0

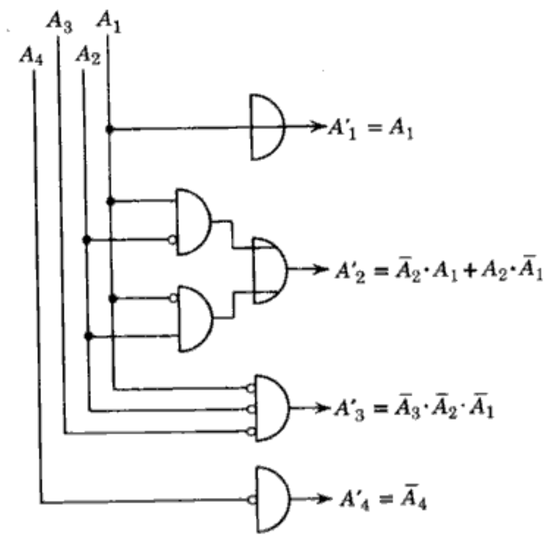


Fig. 19. 5421 Complementer.

PRACTICAL LOGICAL ELEMENTS

In a practical machine, propositions or classes are represented by signals. As these signals progress from one stage to the next, they lose energy. Amplifiers restore this energy.

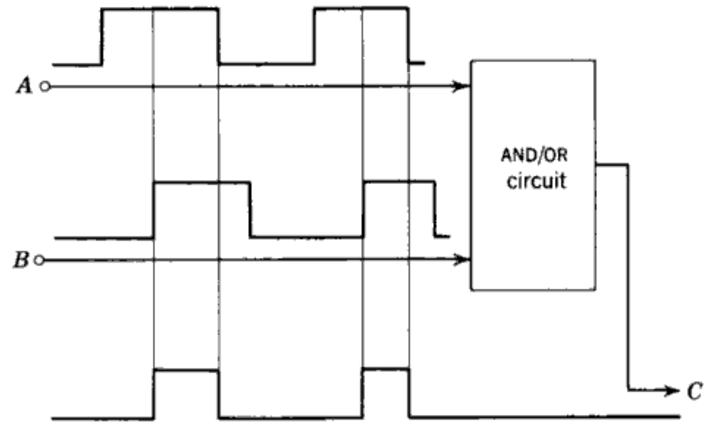
If a signal is produced now but must be applied to a certain circuit a little later, the signal waits in a storage device.

Amplification

There are in general two types of amplifiers: those that invert the polarity of the input signal and those that do not. The first is called an inverting amplifier and the second is called simply an amplifier. The amplifier and inverting amplifier blocks are shown in Fig. 21. Amplifiers perform various other jobs, especially in relation to memories. Symbols may be placed within the triangles to distinguish among several types of amplifiers (as will be shown in Section II).

Storage

The importance of time in computer logic may be illustrated by means of a diagram which also shows how the same circuit may at times be an AND circuit and at other times an OR circuit. This illustration is given in Fig. 20.



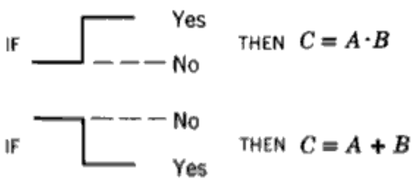


Fig. 20. The AND/OR circuit.

Applied to terminals A and B of the device are two waveforms, each varying differently with time. The device produces an output waveform at C which is positive only when both A and B are positive and negative at all other times. If the positive portions of the waves are taken to represent ONE and the negative portions ZERO, then the device is an AND circuit. If the negative portions of the waves are taken to represent ONE and the positive ZERO, then the device is an OR circuit.

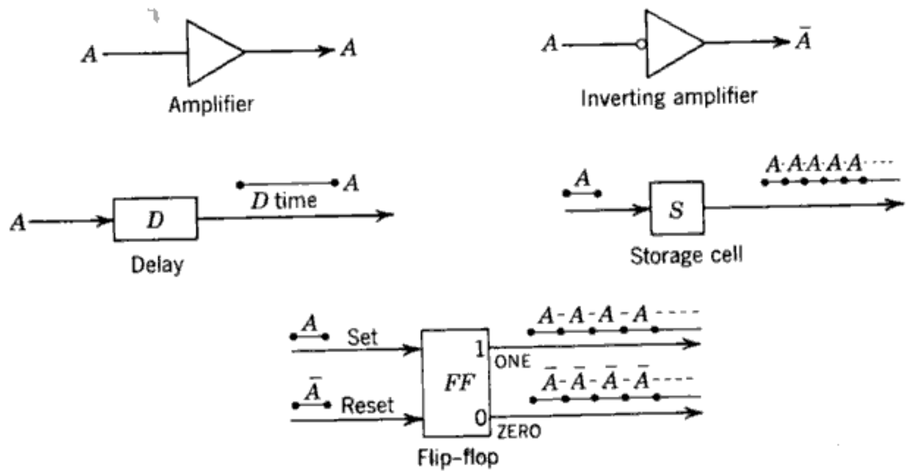


Fig. 21. Amplifying and storage elements.

Time is taken care of by storage. Storage may be represented in block diagrams in several ways. Some of the more common blocks are the time delay, the storage cell, and the flip-flop. Time delay implies that the signal is delayed for a definite amount of time before being applied to another point in the circuit, whereas a storage cell holds a bit for an indefinite time, until called for. Logical blocks representing delay and the storage cell are shown in Fig. 21.

Also shown in Fig. 21 is the flip-flop. The flip-flop is a versatile storage cell. It has two input lines labeled "set" and "reset"; and two output lines labeled ONE and ZERO. A signal applied to the set line produces a continuous ONE output. A signal applied to the reset line removes the information from the ONE output line and causes the ZERO output line to produce a continuous signal. In the first instance, the flip-flop is set; in the second case it is reset. The flip-flop remains in one of these two stable states until a signal flips the flip-flop to the opposite state. But in either state, the two output lines produce signals that are out of phase with each other. This means that the flip-flop produces a signal and its complement simultaneously.

Examples of Practical Logic

As a rough indication of the value of these practical logical elements the adder (composed of two half-adders) and the binary comparator are redrawn in Fig. 22. Part (*a*) shows the use of the amplifiers and delay elements. As the signal passes through each block, it deteriorates. Therefore, after every few basic logical blocks, the signal is fed through an

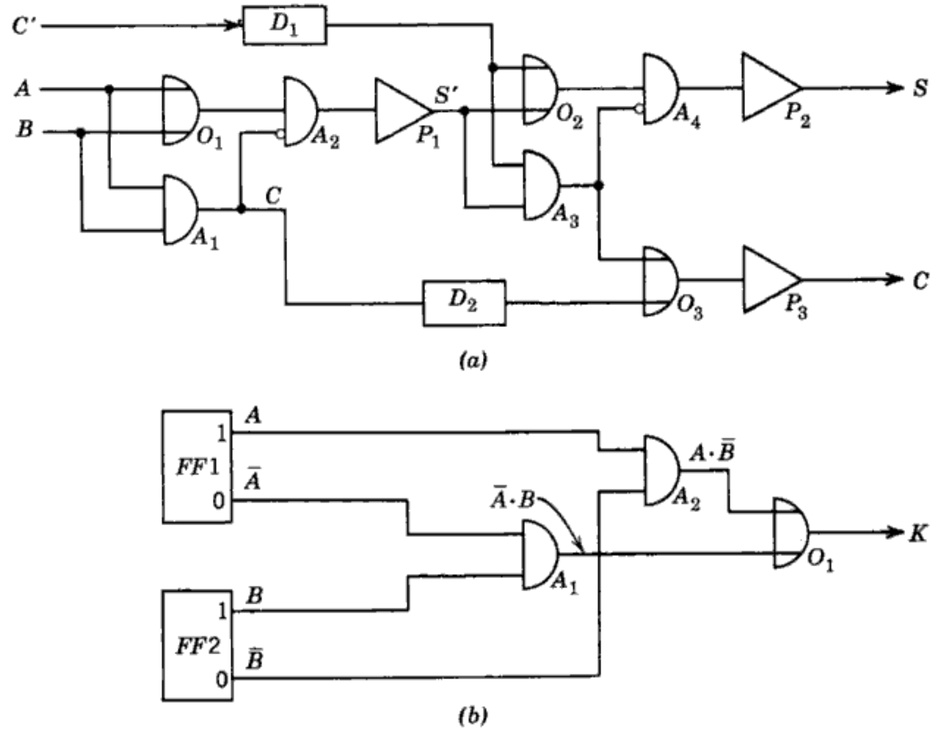


Fig. 22. Use of amplifying and storage elements. (*a*) Binary adder. (*b*) Comparator.

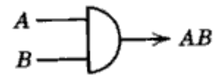
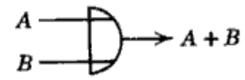
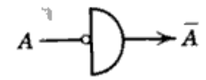
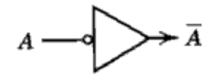
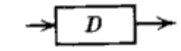
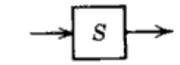
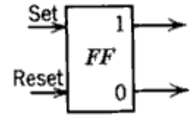
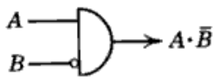
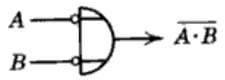
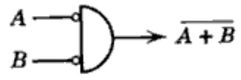
amplifier as shown. If signals A, B, and C' occur at the same time, and if it is assumed that it takes a finite time for a signal to pass through each logical element—as it must—then the signal C' reaches A_3 before S' does. Time delay D_1 is inserted to make sure that both signals are applied to A_3 at the same time. Similarly, the carry C from A_1 is delayed by D_2 to make sure it is applied to the next stage (0_3) at the same time as the output of A_3.

Part (*b*) shows the use of flip-flops in a binary comparator. A_1 produces a pulse only when flip-flop $FF1$ is reset and flip-flop $FF2$ is set. A_2 produces a pulse only when $FF1$ is set and $FF2$ is reset. The output of 0_1 is therefore

$$K = (A \cdot \bar{B}) + (\bar{A} \cdot B)$$

The output K remains indefinitely at either ONE or ZERO until $FF1$ or $FF2$ reverses states.

TABLE 20
Summary of Logical Rules

Name	Block	Type Output or Conditions Under Which Output Produced
(a) Fundamental Elements		
AND		Only if A and B signals present
OR		If either signal A or B or both present
NOT		If no signal at input
Amplifier		Amplified signal
Inverting amplifier		Amplified signal out of phase with input
Delay		Delayed in time by an amount D
Storage cell		Store bit for indefinite time
Flip-flop		Set pulse stores ONE Reset pulse stores ZERO
(b) Combinatorial Elements		
INHIBIT		If A is present and B is not present
INVERTED AND		Complement of AND circuit
INVERTED OR		Complement of OR circuit

(c) Laws

Commutative: $A+B=B+A$; $A\cdot B=B\cdot A$

Associative: $A+(B+C)=(A+B)+C$; $A\cdot(B\cdot C)=(A\cdot B)\cdot C$

Distributive: $A\cdot(B+C)=(A\cdot B)+(A\cdot C)$; $A+(B\cdot C)=(A+B)\cdot(A+C)$

(d) Identities

$A\cdot 0=0$
$U\cdot A=A$
$A\cdot\bar{A}=0$
$A+A=A$
$0+A=A$
$U+A=U$
$A+\bar{A}=U$
$\bar{\bar{A}}=A$

(e) Important Relations

$\overline{(A+B)}=\bar{A}\cdot\bar{B}$
$\overline{(A\cdot B)}=\bar{A}+\bar{B}$
$(\bar{A}\cdot B)+(A\cdot\bar{B})=(\bar{A}+\bar{B})\cdot(A+B)$
$(\bar{A}\cdot\bar{B})+(A\cdot B)=(A+\bar{B})\cdot(\bar{A}+B)$

SUMMARY

1. There are three basic logic elements: AND, OR, and NOT. But for practical machine logic, five more logical elements must be included: amplifier, inverting amplifier, time delay, storage cell, and flip-flop.

2. The three fundamental logical elements may be manipulated according to simple algebraic rules. Amplifying and storage elements are kept track of with the aid of logical block diagrams.

3. To develop a device to perform a certain job, a truth table is drawn from which logical algebraic expressions are derived. The algebraic expressions are converted to block diagrams, to which then are added amplifying and storage blocks. The final logical block diagram is a plan for the construction of the device.

4. Some of the more important blocks, algebraic laws, identities, and relationships are summarized in Table 20.

5. *Definitions to Remember*

EXCLUSIVE OR = BINARY COMPARATOR = QUARTER-ADDER—A device which compares two bits. If the two bits are alike, a ZERO output is produced. If the two bits are unlike, a ONE output is produced.

TRUTH TABLE—A statement of all conditions of a problem in tabular form.

HALF-ADDER—A device which produces the binary sum and carry of two bits. (The sum portion may be obtained by a quarter adder.)

BINARY ADDER—A device which produces the binary sum and carry when the augend, addend, and carry from previous bit position are applied.

section II

BUILDING BLOCKS

In the last chapter of Section I we developed the logical blocks which form the basis for all computer functions. In this section we translate each of the theoretical logical functions into hardware—we form practical logical building blocks. These logical building blocks are built of many kinds of components. Several components which have been used in practical computers and several which have been investigated only in the laboratory are discussed. Among the building blocks represented are those using electromechanical, vacuum-tube, electromagnetic and semiconductor principles. Emphasis in every case is placed not so much on the circuit principles involved—which are in the realm of electronics for the most part—but on how each device or circuit performs a logical function.

The principles behind each building block are presented first. Then the laws of logic are applied to develop several functional building blocks such as adders, complementers, and registers. In the process the interrelationships among logical expressions, logical block diagrams and schematics are made clear.

CHAPTER 5

Mechanical and electromechanical components

The logical, amplification, and storage functions of machine logic find their counterparts in the following logical building blocks: switching circuits, amplifiers, and storage circuits. Switching circuits produce AND, OR, and NOT functions. Amplifiers invert (produce NOT) as well as amplify. Often switching and amplifying functions are performed by one device. Storage circuits include those which produce delay as well as those which store information for an indefinite time.

The three types of logical building blocks—the switch, amplifier, and storage device—may take on various forms. In this chapter we look at some of the simplest devices, those using mechanical and relay types of mechanisms.

SWITCHING CIRCUITS

The fundamental elements of logical algebra may be expressed practically by means of switches (see Fig. 23). If the switches are wired in series, then all switches—*A and B and C*—must be turned on to light the lamp. This is an AND circuit. If the switches are wired in parallel, then the turning on of either switch *A or B or C or* any combination of switches causes the lamp to light. This is an OR circuit. If a normally closed switch keeps the lamp lit except when the switch is depressed, then the circuit is a NOT circuit.

In this arrangement, depressing the switch represents the input signal, and a voltage or current flow lighting the lamp represents the output signal of the switching circuit. It would be much more satisfactory from the point of view of mechanization to have the input and output signals alike. This is easily accomplished by substituting relays for switches, as shown

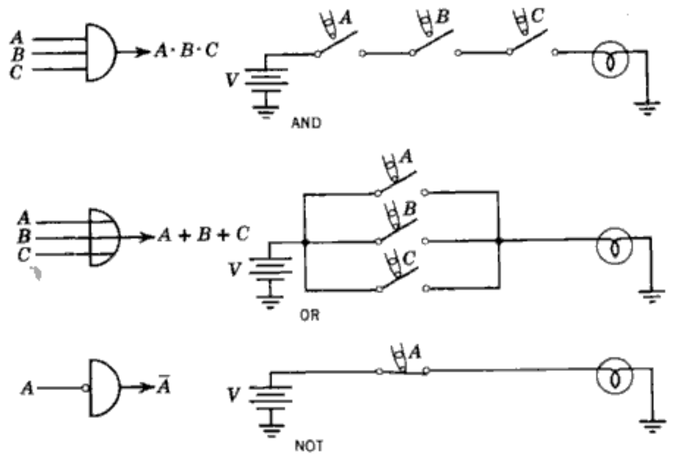


Fig. 23. Mechanical switching circuits.

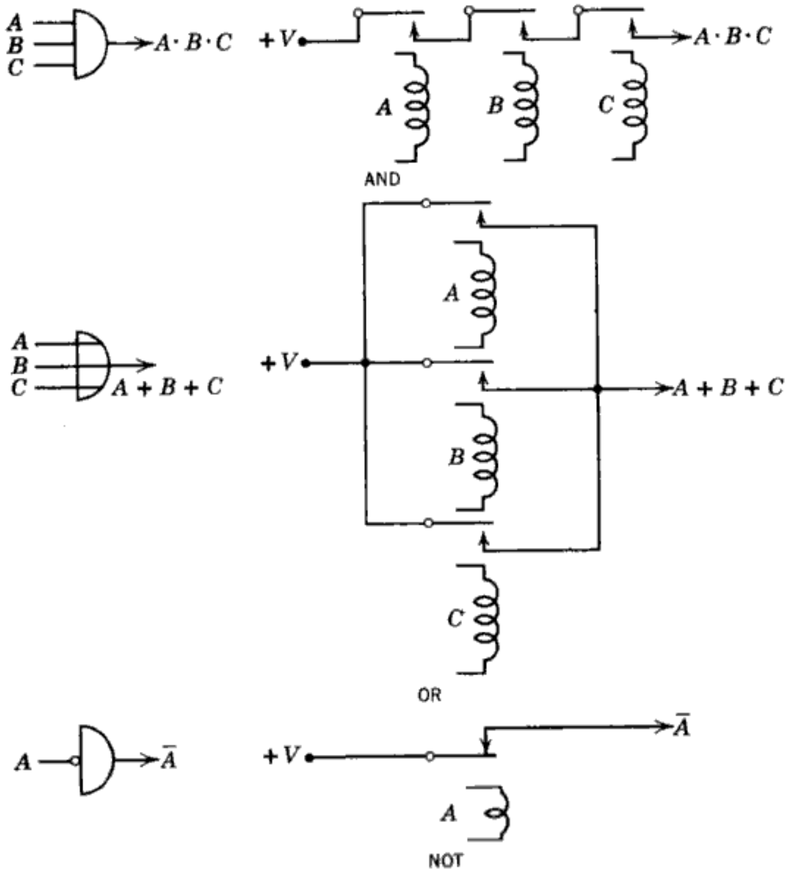


Fig. 24. Relay switching circuits.

in Fig. 24. Here voltage (or current) signals activate the relay coils, and voltage (or current) signals are produced at the output lines. The output of each relay switching circuit, may therefore be fed to another switching circuit.

Encoders and Decoders

Switching circuits may be combined to produce encoders and decoders. An encoder is a device which translates one of a group of signals into a

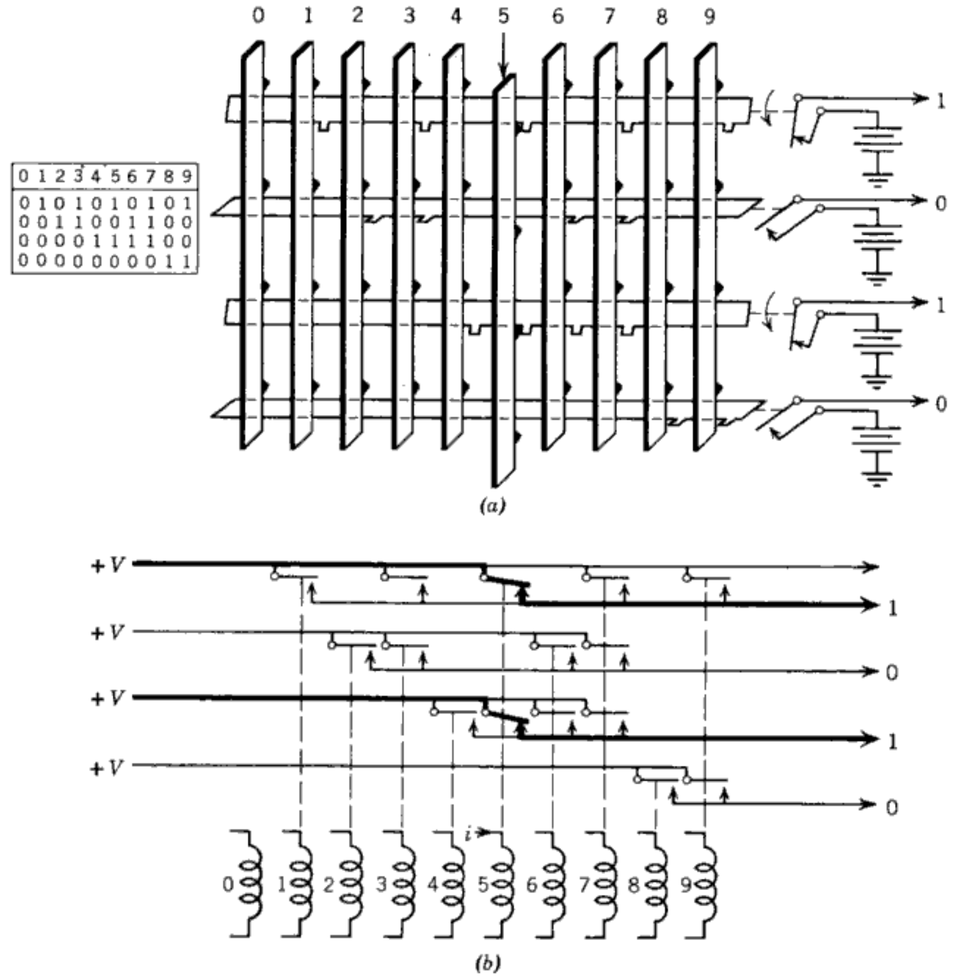


Fig. 25. Encoders. (*a*) Mechanical encoder. (*b*) Relay encoder.

combination of signals. A decoder is a device which translates a combination of signals into one of a group of signals.

Figure 25 shows a mechanical encoder and a relay encoder. Both translate effectively from decimal (one of a group of ten signals) to

binary-coded decimal (a combination of four signals). The mechanical encoder consists of ten vertical bars each having four evenly spaced ledges; and four horizontal bars possessing ledges in a pattern determined by the binary code for each decimal number. The input signal consists of

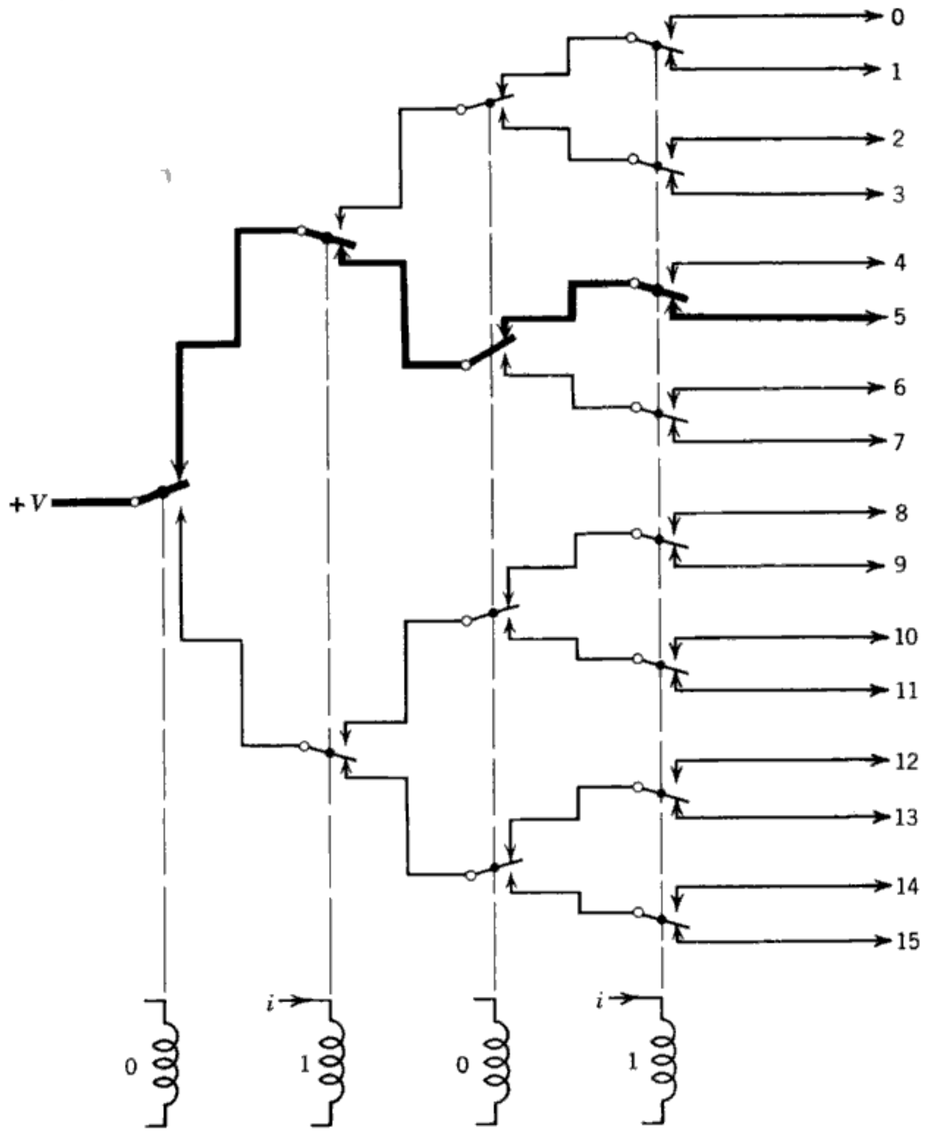


Fig. 26. Relay tree decoder.

mechanical pressure applied in a downward direction to one of the vertical bars. Wherever the ledges on this bar strike the ledges on the horizontal bars, these horizontal bars are made to move in an arc. The horizontal bars are connected mechanically to a pair of electric contacts in such a

way that, if the bar is moved, the contacts close, causing current to flow in the associated output line. In the illustration, bar 5 is depressed. The ledges on this bar hit the ledges on horizontal bars 1 and 3. Therefore, current flows through lines 1 and 3. If we consider the flow of current a ONE and the absence of current a ZERO, then the output is 0101, the binary-coded decimal equivalent of 5.

The relay encoder works in a similar manner. Instead of ten vertical bars it has ten relays. Instead of four horizontal bars it has four sets—or rather four possible sets—of contacts on each relay. These contacts are arranged analogously to the way the ledges are arranged on the horizontal bars. When a signal is applied to relay 5, the first and third lines are activated as before to produce 0101.

In Fig. 26 is shown an example of a relay tree decoder which chooses one out of 16 output lines when a 4-bit combination is applied to it. The input signals are applied to the relay coils; a ONE is represented by a current pulse, a ZERO is represented by the absence of a current pulse. The movable contact is forced downward for each of the relays to which a current pulse is applied; all other movable contacts remain in their upward positions. For each combination of ONE's and ZERO's applied, a different closed-circuit path is established. For instance, in the illustration 0101 is applied. As a result, the first and third (counting from the right) relays are activated. The $+V$ is therefore fed through the path indicated by the bold lines to choose line number 5.

Binary Comparator

The logical expression for the binary comparator is

$$K = A \cdot \bar{B} + \bar{A} \cdot B$$

Two versions of the relay binary comparator are shown in Fig. 27. In the first version, one signal is applied to the two coils on the left, the other signal to the two coils on the right. If A equals ONE and B equals ZERO, the A contacts are energized and the B contacts are de-energized; current flows along the bold line as shown to produce an output pulse. If A equals ZERO and B equals ONE the contacts would be arranged so that current flowed through the lower branch to produce a pulse. On the other hand, if both A and B are energized ($A \cdot B$) or both de-energized ($\bar{A} \cdot \bar{B}$), then no output pulse is produced.

The lower version of the circuit performs the same job with only two relays. The bold line indicates the path of current flow when A equals ZERO and B equals ONE.

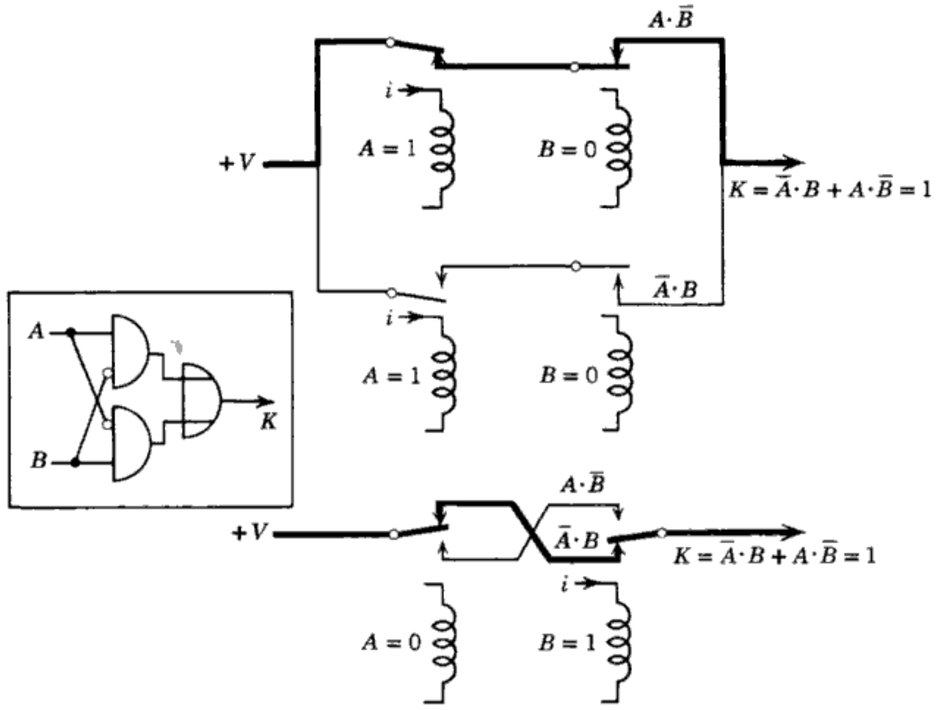


Fig. 27. Relay binary comparator.

Half-Adder

A circuit of the half-adder constructed of relays is shown in Fig. 28. Two relays ($RY1$ and $RY2$) are required to manufacture the carry:

$$C = A \cdot B$$

and three additional relays ($RY3$, $RY4$, and $RY5$) are required to manufacture the sum:

$$S = (A + B) \cdot (\overline{A \cdot B})$$

The $A \cdot B$ signal produced by $RY1$ and $RY2$ is applied to $RY3$. Since $RY3$ is normally closed, the output of this relay is $\overline{A \cdot B}$. This $\overline{A \cdot B}$ signal is in series with the parallel circuit producing $A + B$. Therefore the output is equal to $(A + B) \cdot (\overline{A \cdot B})$.

The illustration shows what happens when A equals ZERO and B equals ONE. $RY1$ and $RY4$ are de-energized; $RY2$ and $RY5$ are energized. No current flows through the carry line:

$$C = 0$$

Because of the absence of carry current, $RY3$ is de-energized. As a result, current flows through the bold line to produce a sum:

$$S = 1$$

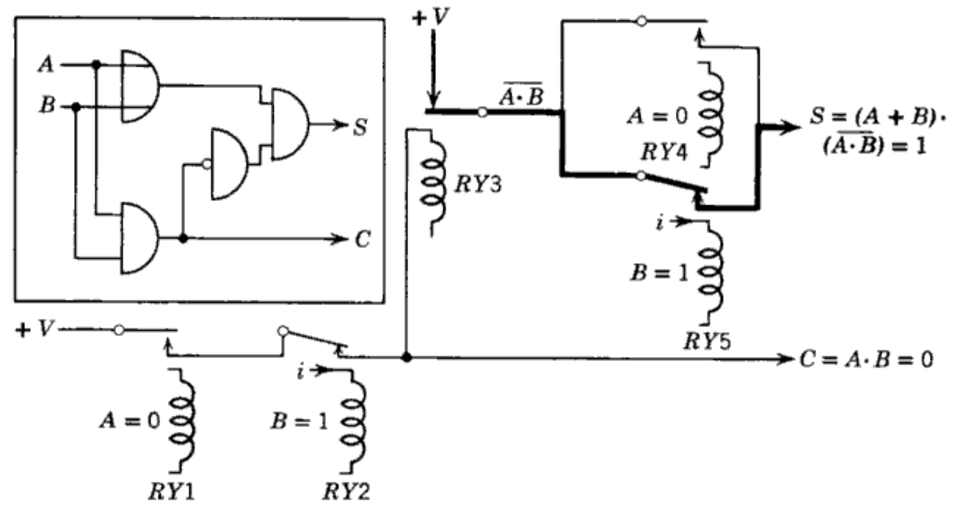


Fig. 28. Relay half-adder.

Binary Adder

The logical equations for a binary adder were found to be:

$$S = \bar{A} \cdot \bar{B} \cdot C' + A \cdot B \cdot C' + \bar{A} \cdot B \cdot \bar{C}' + A \cdot \bar{B} \cdot \bar{C}'$$
$$C = A \cdot B \cdot C' + A \cdot B \cdot \bar{C}' + \bar{A} \cdot B \cdot C' + A \cdot \bar{B} \cdot C'$$

With the aid of the rules presented in Chapter 4 these equations may be simplified; and the simplified equations may reduce the number of components—in this case relays—required to build this circuit. The C' and $\bar{C}'$ may be factored out of the sum equations as follows:

$$S = C'(\bar{A} \cdot \bar{B} + A \cdot B) + \bar{C}'(\bar{A} \cdot B + A \cdot \bar{B})$$

From

$$U = \bar{A} \cdot \bar{B} + A \cdot B + \bar{A} \cdot B + A \cdot \bar{B}$$

we see that $(\bar{A} \cdot \bar{B} + A \cdot B)$ is the complement of $(\bar{A} \cdot B + A \cdot \bar{B})$, or

$$\bar{A} \cdot \bar{B} + A \cdot B = (\overline{\bar{A} \cdot B + A \cdot \bar{B}})$$

Therefore

$$S = C'(\overline{\bar{A} \cdot B + A \cdot \bar{B}}) + \bar{C}'(\bar{A} \cdot B + A \cdot \bar{B})$$

Similarly, the carry equation may be simplified by factoring as follows:

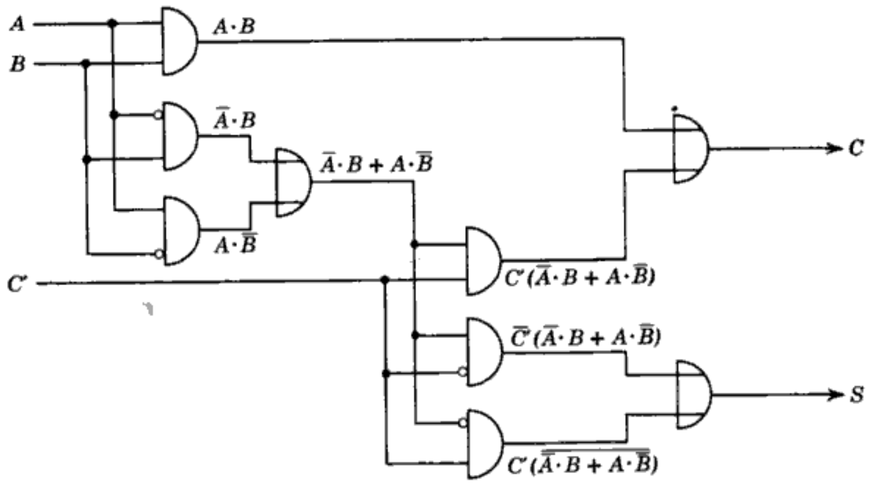
$$C = A \cdot B(C + \bar{C}') + C'(\bar{A} \cdot B + A \cdot \bar{B})$$

Since

$$C' + \bar{C}' = U$$

then

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



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

$$S = \bar{A}\cdot\bar{B}\cdot C' + A\cdot B\cdot C' + \bar{A}\cdot B\cdot\bar{C}' + A\cdot\bar{B}\cdot\bar{C}'$$
$$= C'(\bar{A}\cdot\bar{B} + A\cdot B) + \bar{C}'(\bar{A}\cdot B + A\cdot\bar{B})$$
$$= C'\overline{(\bar{A}\cdot B + A\cdot\bar{B})} + \bar{C}'(\bar{A}\cdot B + A\cdot\bar{B})$$

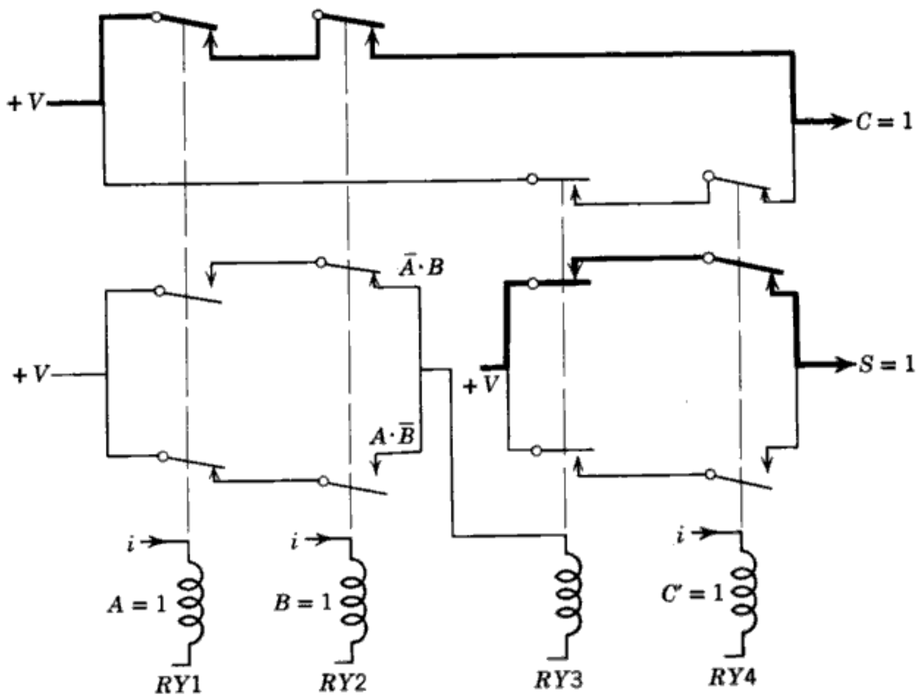


Fig. 29. Relay binary adder.

The logical block diagram and a relay schematic for a half-adder are shown in Fig. 29. First $A \cdot B$ and $\bar{A} \cdot B$ and $A \cdot \bar{B}$ are produced. Then $\bar{A} \cdot B$ and $A \cdot \bar{B}$ are sent through an OR circuit to produce $\bar{A} \cdot B + A \cdot \bar{B}$. This factor in turn is combined with C' several ways to produce all the other necessary terms:

$$C'(\bar{A} \cdot B + A \cdot \bar{B})$$

$$\bar{C}'(\bar{A} \cdot B + A \cdot \bar{B})$$

$$C'\overline{(\bar{A} \cdot B + A \cdot \bar{B})}$$

The schematic shows the positions of the relay contacts when A equals ONE, B equals ONE, and C' equals ONE. Current flows through the bold lines to produce a sum and a carry, or

$$S = 1$$

$$C = 1$$

AMPLIFYING CIRCUITS

In relay switching circuits, a ONE is represented by the presence of current flow, and a ZERO is represented by the absence of current flow. In practical terms, since no conductor or insulator is perfect, the ONE is represented by a certain high level of current and the ZERO by a certain minute level of current. But the amount of current flowing in a line decreases as the number of contacts in series is increased. It is easy to see that in some cases the current in a line carrying a ONE may decrease to the point where another logical building block may mistake it for a ZERO. To avoid this problem, we need amplifiers.

In relay switching circuits, the relay itself can act as an amplifier. A relay can amplify because a small current applied to the relay coil can cause a much larger current to flow through the relay contacts. Note in Fig. 29 that the $(\bar{A} \cdot B + A \cdot \bar{B})$ signal is fed to the coil of $RY3$ and this coil in turn causes current to flow through the contacts as shown. In effect then the $(\bar{A} \cdot B + A \cdot \bar{B})$ output is fed through an amplifier before being applied to other switching circuits. In Fig. 28, relay $RY3$ acts as an inverting amplifier.

So we see that by using relays we not only produce logical outputs, but we also produce them in a manner which allows them to activate other relay circuits. Many other logical components possess both switching and amplifying properties, as is shown in this section.

STORAGE DEVICES AND CIRCUITS

A device which has two stable states can store information. To be a useful storage device for computers, it should be easy to write information into the device and read information from it. The three important elements in storage devices are then:

1. Storing.
2. Writing.
3. Reading.

Several mechanical and electromechanical storage devices are now discussed with respect to these three elements.

Punched Paper Tape

Information is stored on punched paper tape by means of patterns of holes as shown in Fig. 30. Each spot where a hole may appear is in effect

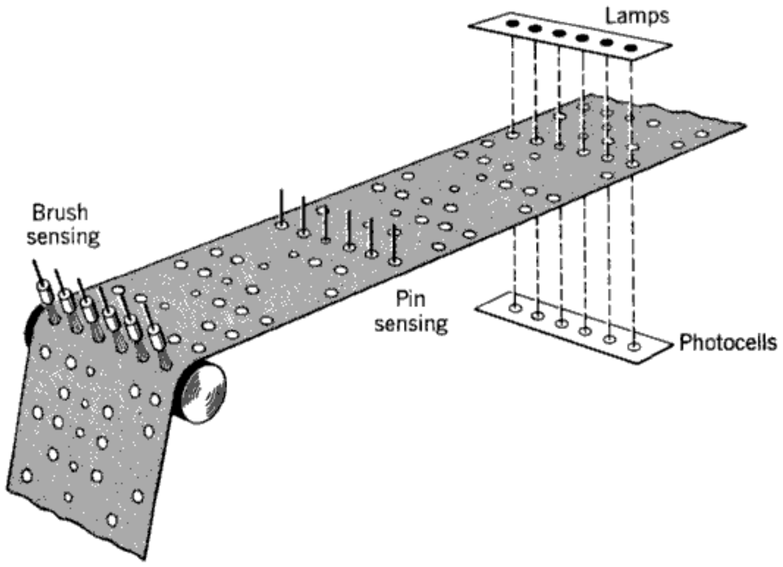


Fig. 30. Methods of reading punched paper tape.

a binary storage cell. It may store a hole (ONE) or a no-hole (ZERO). Writing is accomplished by means of a hole-punching machine. Reading may be performed by means of pins, brushes, or photocells. In the pin-sensing method, as the paper moves, a group of spring-loaded pins pushes against the paper. Where a hole is present, the pin falls through the hole; as the pin falls, it causes a switch to close and produce a pulse of current

(ONE). The pin meeting a no-hole does not fall and does not cause a pulse of current (ZERO).

The holes and no-holes may be read by means of brushes. Wherever a hole is present, the brush closes the circuit with the contact on the opposite side to produce a ONE. Where a no-hole is present, no contact is made and a ZERO is produced.

In the photocell method a bank of six lamps shines at a row of code on the paper. On the opposite side of the paper is a bank of photocells. At a point where a hole is present light gets through and impinges on the associated photocell to produce current (ONE). At a point where a no-hole is present, no light passes and no current is produced by the associated photocell (ZERO).

The smaller holes are called sprocket holes. Their function is to allow the sprocket of a punched paper tape reader to move the paper at a constant rate.

Punched Cards

A punched card is another storage device which stores information by means of punched holes. The main difference between punched paper

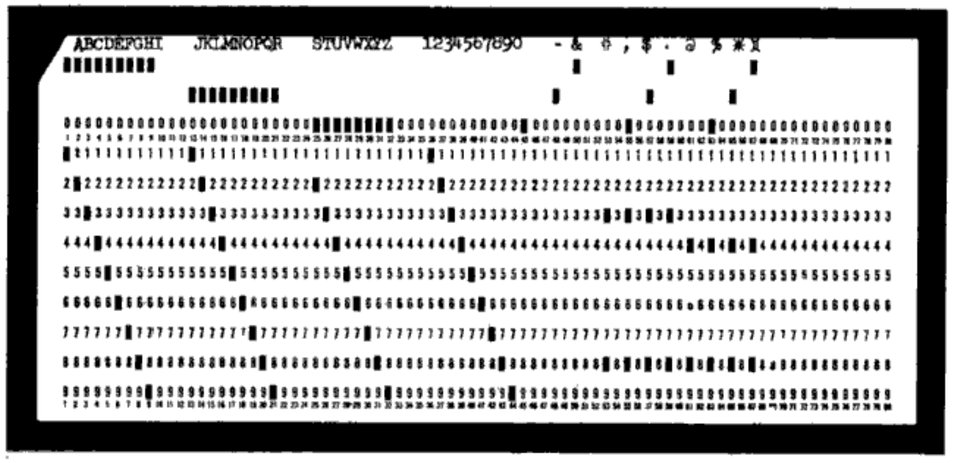


Fig. 31. Eighty-column punched-card alphanumeric code.

tape and punched cards is that the former is a continuous record and the latter is a discrete record. This difference is significant in the selection of data-processing systems.

The most common punched card has 80 columns with 10 positions labeled 0 to 9 (Fig. 31). There are two unlabeled positions in the space above the 0-digit position: *X* (nearest the 0 position) and *Y*. Decimal digits may be written into each column by punching the appropriate spot.

Letters may be written by punching two holes in each column; one hole is punched in either the 0, *X*, or *Y* row and the other hole is punched in a digit position. For some of the editing symbols three holes are necessary.

One method of writing is performed by means of machines which punch holes into appropriate positions when keys are depressed.

Reading may be accomplished by the brush-sensing or photocell methods. In the former method a roller moves the card along its breadth across a set of 80 brushes. At each hole a circuit closes; at each point where a no-hole exists the circuit stays open. How does the reader distinguish among pulses representing 9, 8, or 0? This is done by means of a clock which is activated when the card starts moving and counts 12 pulses. If the card is moved from its 9 position toward its *Y* position a pulse produced by a brush during count 1 represents a 9; a pulse produced by a brush at count 10 represents a 0.

Printing Devices

Printing is a means of "writing" on paper in a form where direct communication to, or reading by, human beings is possible. Printers, therefore, play an important role in communication between machine and man. Several types of mechanical printing mechanisms are available. Four generic types are discussed briefly.

Figure 32 shows a simple system in which all the characters are aligned in a vertical rack. Characters are chosen for printing by pushing the racks upward against springs and stopping each rack in its movement at the appropriate time by a stop pawl. After all the racks are in position, printing plungers force the keys against carbon and paper. The racks are then brought back to their neutral position. The paper and carbon are moved a little, the racks are adjusted again, and the plungers cause the next line to be printed. Several adding and calculating machines print in this manner.

Figure 33 illustrates the type-wheel printer. In this type of printing mechanism, several wheels, one for each print position on a line, are mounted on a steadily rotating axle. Each type-wheel has embossed around its periphery all the characters which may be printed. At any one time during the rotation of the type-wheels, the same character appears in printing position at each wheel. Printing is accomplished by striking a hammer onto carbon and paper at each printing position along the line at the appropriate time. This means that all the a's in a line are printed first, then all the b's, all the c's, etc. One revolution of the type-wheels is needed to print one line of information. As soon as a line is printed, the paper and carbon are moved and the characters of the next line are printed during the next revolution of the type-wheels.

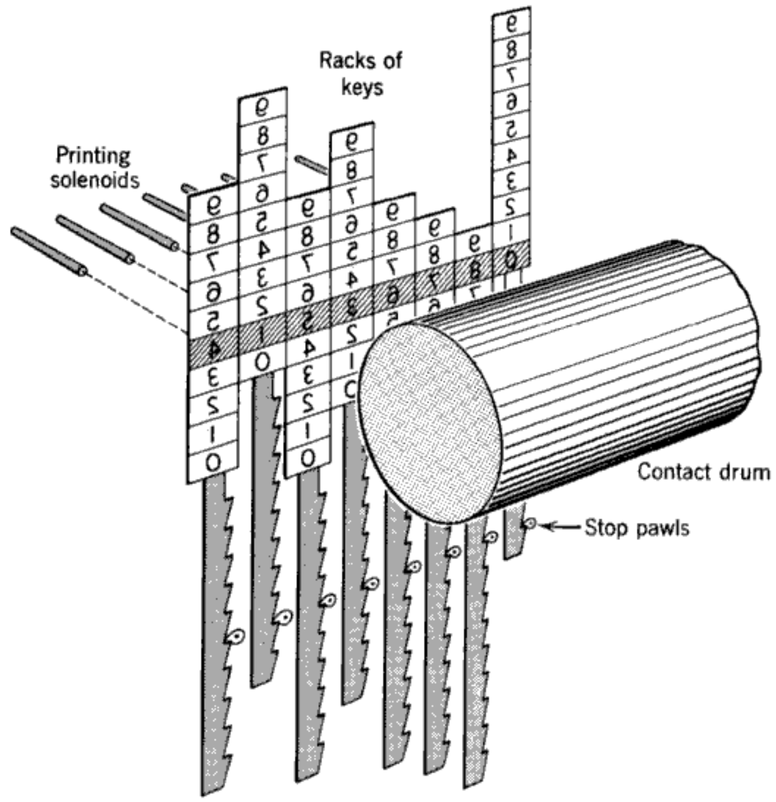


Fig. 32. Rack-type printing mechanism.

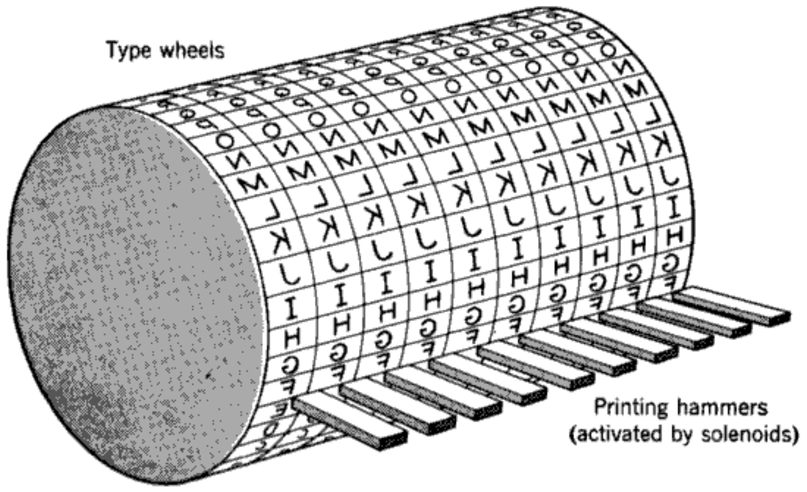


Fig. 33. Type-wheel printing mechanism.

The spinning-disk type of printing mechanism is shown in Fig. 34. In this case, all the characters are embossed around the periphery of one horizontal disk. The paper is fed vertically and follows the curvature of the disk. Arranged in an arc on the opposite side of the paper are the solenoids, one for each printing position along a line. As the disk rotates, different characters appear at different points along the line. As the appropriate character appears in a desired printing position, the associated print hammer is activated by a solenoid. One revolution of the disk is

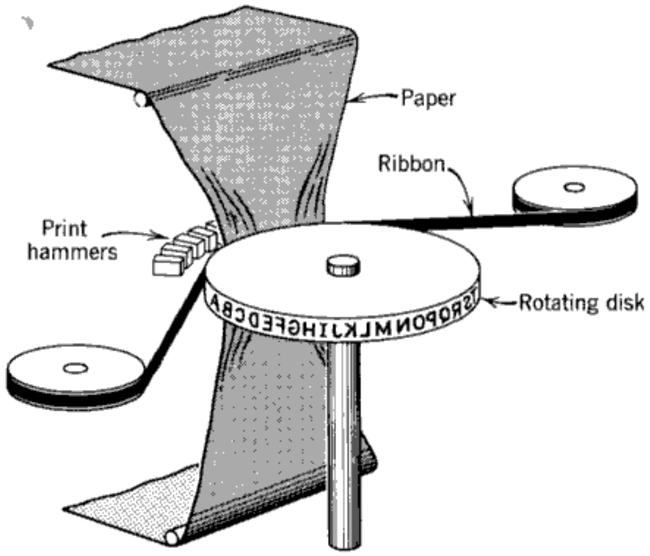


Fig. 34. Spinning-disk printing mechanism.

needed to print a line. If a series of a's is desired across the line, the solenoids are activated in sequence. If all the characters are desired in sequence across the line, all the hammers are activated at once.

Another type of printing mechanism uses a matrix composed of pins. Different characters are printed by producing impressions on paper with different configurations of pins. Figure 35 shows character impressions produced when a matrix of 35 pins, 5 columns of 7 pins each is used.

Relay Register

A register is a circuit for storing several bits of information temporarily. One of its many functions in the computer is the storage of addend, augend, and other terms in arithmetic operations.

An ordinary relay is a switching and amplifying device. But it cannot be used as a storage device because the removal of current from the coil causes the associated contacts to open. A relay with holding contacts,

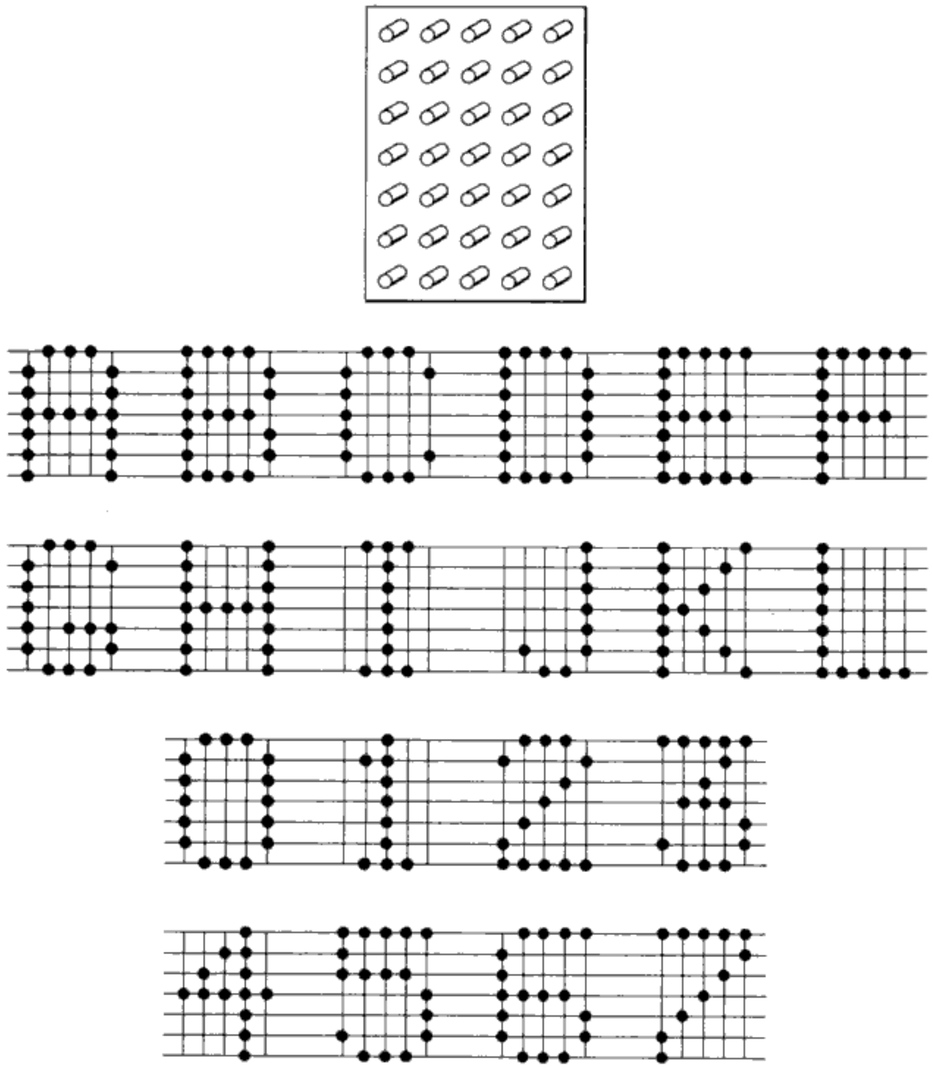


Fig. 35. Matrix-type printing.

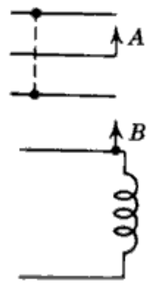


Fig. 36. Relay with holding contacts.

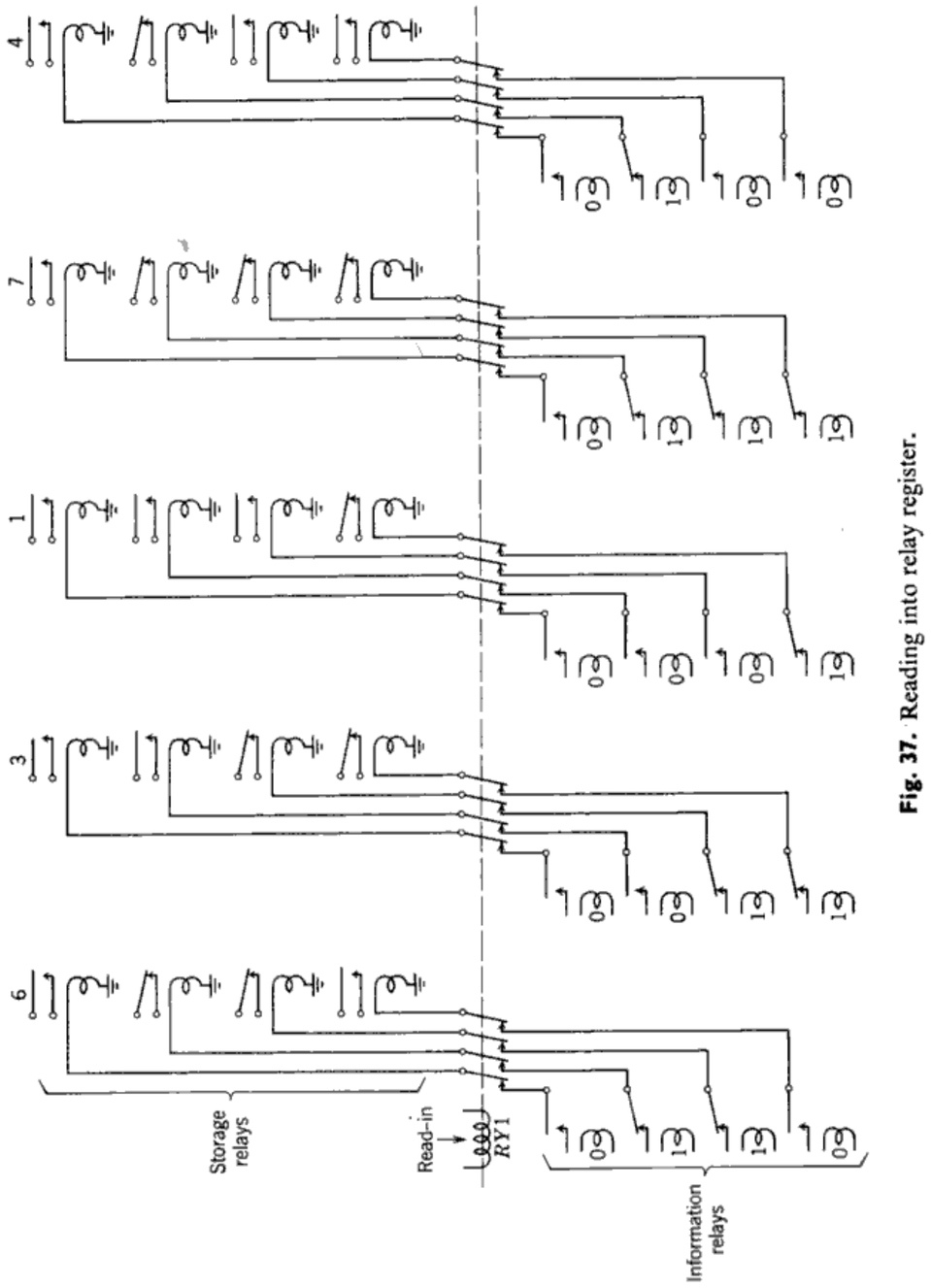


Fig. 37. Reading into relay register.

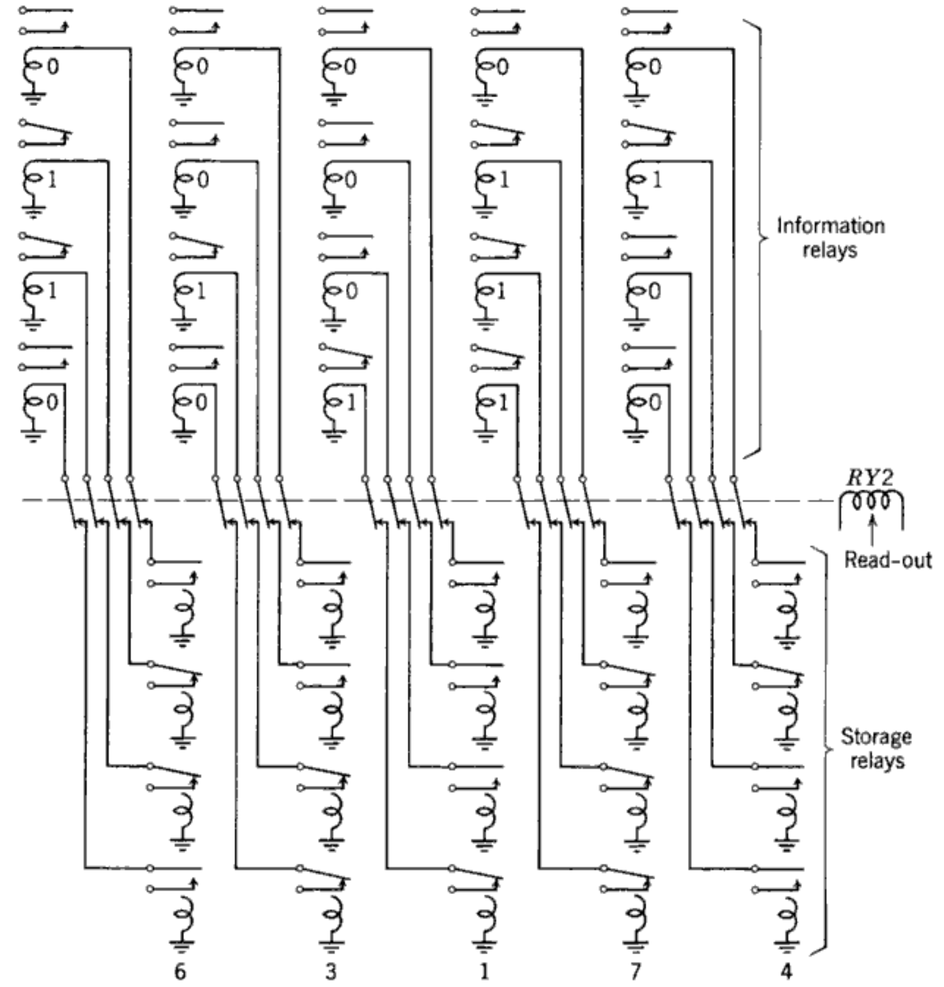


Fig. 38. Reading out of relay register.

however, may be used to store information. An example of a relay with holding contacts is shown in Fig. 36. Contacts *A* are the normal contacts. Contacts *B* are the holding contacts. Once the coil is energized, current is applied to the coil through the *B* contacts to keep the coil energized even if the initial pulse is removed from the coil. A switch in series with the *B* contacts may be used to deenergize the relay coil. A pulse of current applied to the coil closes contacts *A* to store a ONE. If no pulse of current is applied to the coil contacts *A* remain open to store a ZERO.

Figures 37 and 38 show a relay register consisting of 20 relays and capable of storing a number consisting of 5 binary-coded decimal digits. Only contacts *A* are drawn for each relay; contacts *B* are assumed. Figure 37 shows how information is transferred to the register, and Fig. 38 shows how information is transferred from the register. Transferring

information into the register is performed by read-in gates. Transferring information from the register is performed by read-out gates.

In Fig. 37, relay $RY1$ performs the read-in function. When $RY1$ is unoperated, all the storage relays are storing ZERO's. When a signal is applied to $RY1$, the contacts of its 20 poles close. Each of these poles together with the pole of the relay in the same line constitutes an AND circuit. Whenever both sets of contacts are closed, current flows to the associated storage relay to cause it to store a ONE. Whenever the information relay is open, no current flows to the associated storage relay; the storage relay therefore stores a ZERO. The illustration shows 63174 read into the relay register.

When information stored in the register is required in another location, a read-out pulse is applied to $RY2$ (Fig. 38). This relay closes 20 circuits as shown. Relays in series with storage relays storing ONE's are activated, others are not. In this way 63174 may be fed to another part of the computer.

SUMMARY

1. A component which acts like a switch may be used to form a switching circuit. A device which increases the energy in a signal may be used for amplification. A device which has two easily entered and easily distinguished stable states may be used for storage (see Fig. 39).

2. The relay makes a good switching component. An AND circuit can be built by connecting the relay contacts in series; an OR circuit by connecting the relay contacts in parallel. A NOT may be achieved by using a normally closed relay (see Fig. 39).

3. A relay may be used for amplification since the current flowing through the relay contacts may be greater than the current flowing through the coil.

4. A relay with holding contacts may be used as a storage cell. By using several such relays, a register for storing numbers may be built (see Fig. 39).

5. A few prominent mechanical storage devices are the punched paper tape, punched card, and several types of printers. The first two store coded information, the last store actual words.

6. *Definitions to Remember*

SWITCHING CIRCUIT—A circuit which produces an output for certain combinations of input.

ENCODER—A network (or mechanical device) of circuits in which only one input is excited at a time, and each input produces a combination of outputs.

DECODER—A network of circuits which converts a combination of signals into an output on one of several output lines.

REGISTER—A device or circuit which stores temporarily numbers consisting of several digits or bits.

WRITE—To place bits into a storage device or circuit.

READ—To remove bits from a storage device or circuit.

READ-IN GATE—A switching circuit which allows information to be transferred into a register.

READ-OUT GATE—A switching circuit which allows information to be transferred from a register.

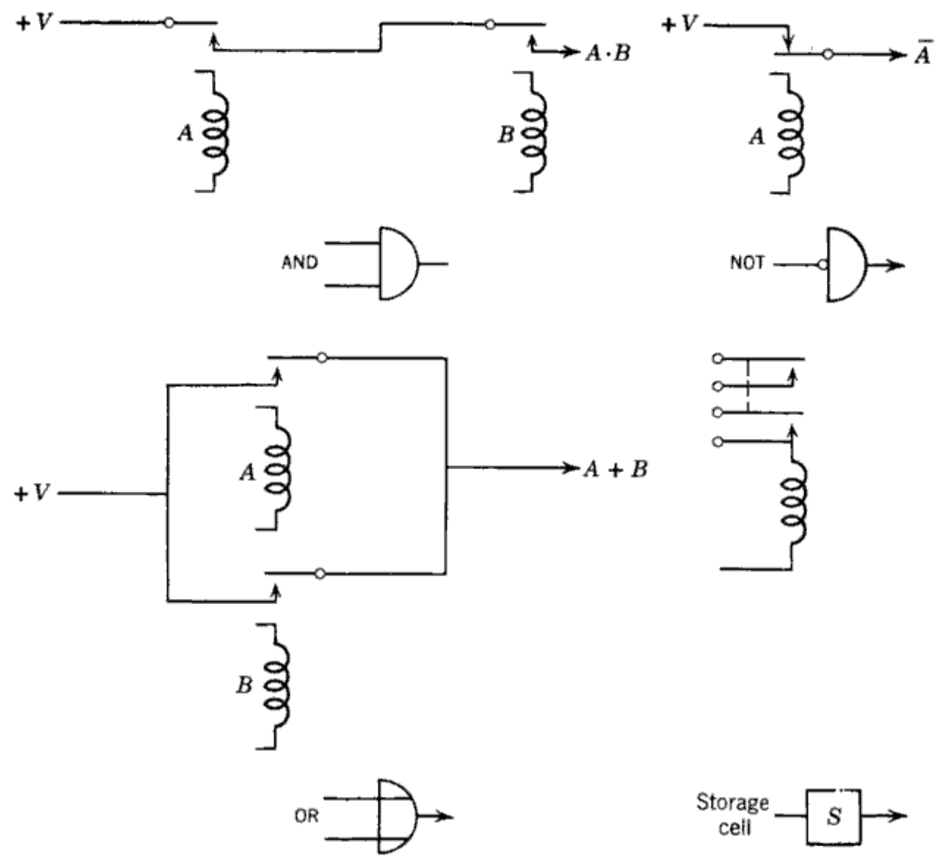


Fig. 39. Relay building blocks.

CHAPTER 6

Vacuum-tube and related components

Switching, amplification, and storage may be achieved with electronic components such as resistors, capacitors, inductors, vacuum tubes, gas tubes, and cathode-ray tubes. Resistors and diodes make good switching components. Capacitors, inductors, gas tubes, and cathode-ray tubes make good storage components. The vacuum-tube triode and tetrode may be used in switching, amplifying, and storage circuits. By themselves the triode and tetrode are amplifying devices. With judicious use of their characteristics, they may be made to act as switches. By introducing feedback between two vacuum-tube amplifiers the flip-flop is formed.

ELECTRONIC LOGICAL ELEMENTS

Resistors, Capacitors, and Inductors

A resistor is a very simple electronic element. When current flows through a resistor, a voltage drop is produced across it. When a potential is applied across two or more resistors in series—a voltage divider—the voltage across each resistor is proportional to the value of the resistor. These facts may be used to build a switching circuit of resistors.

In the broader sense, a switch is a device which allows a signal to come through under one set of circumstances and does not allow the signal to come through under another set of circumstances. The relay switch is a device which under one set of circumstances produces a high *current flow*, and under another set of circumstances produces a low *current flow*. A device which under one set of circumstances produces a high *voltage* and under another set of circumstances a low *voltage* may also be called a switch. Such a device may be built of resistors.

Figure 40 illustrates the voltage-switching effect of resistor voltage dividers. With +6 v applied to each resistor, there is no potential drop across either resistor, and +6 v appears at the output. With 0 v applied to each resistor, there is no potential difference across the divider and therefore 0 v appears at the output. With 0 v applied to one resistor and +6 v to the other, a potential difference of +3 v is produced at the output. If +6 v represents a high voltage, and a voltage of +3 v or less a low voltage, then the switch is shown in the "closed" position in (*a*) and in the "open" position in (*b*) and (*c*).

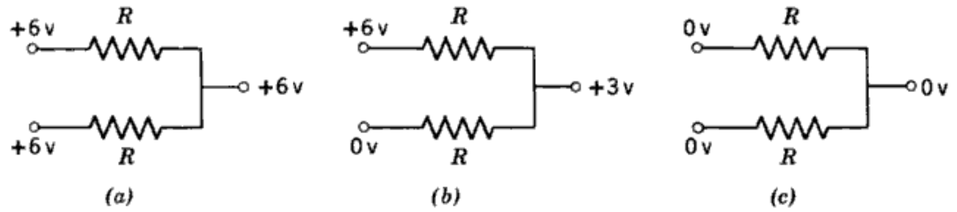


Fig. 40. The resistor as a switching component.

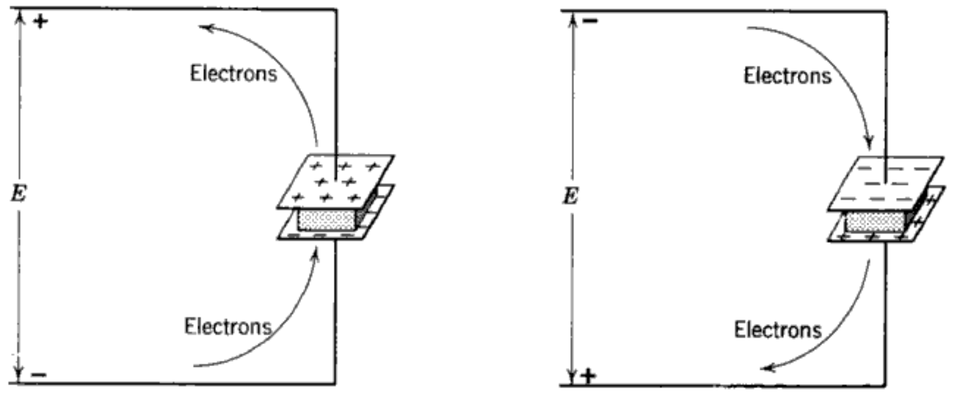


Fig. 41. The capacitor as a storage component.

The capacitor stores electric charge. When a voltage is applied across a capacitor, the capacitor charges positively or negatively depending on the polarity of the applied voltage (Fig. 41). When a short circuit is placed across a charged capacitor, the capacitor discharges. Both the charge and discharge of the capacitor take a definite amount of time. Therefore, the capacitor may be used to produce time delay. If discharge can be prevented, the capacitor may be used as a storage cell.

The charging time of a capacitor may be controlled by a series resistor. With a large resistance in series, the charging time is long; with a small resistance in series, the charging time is short. A resistor-capacitor combination with a short charging time may be used as a differentiator.

A differentiator is a circuit which converts a square wave into two narrow pulses, one of opposite polarity from the other.

Briefly, the differentiator (Fig. 42) works as follows. When the voltage across the resistor-capacitor combination rises to $+E$, this voltage appears across R because the voltage across a capacitor cannot change suddenly. The charge on the capacitor builds up exponentially. As the charge builds up, so does the voltage across the capacitor. As the voltage across the capacitor rises, the voltage across the resistor must fall to maintain the voltage across the combination at $+E$. When the capacitor is fully

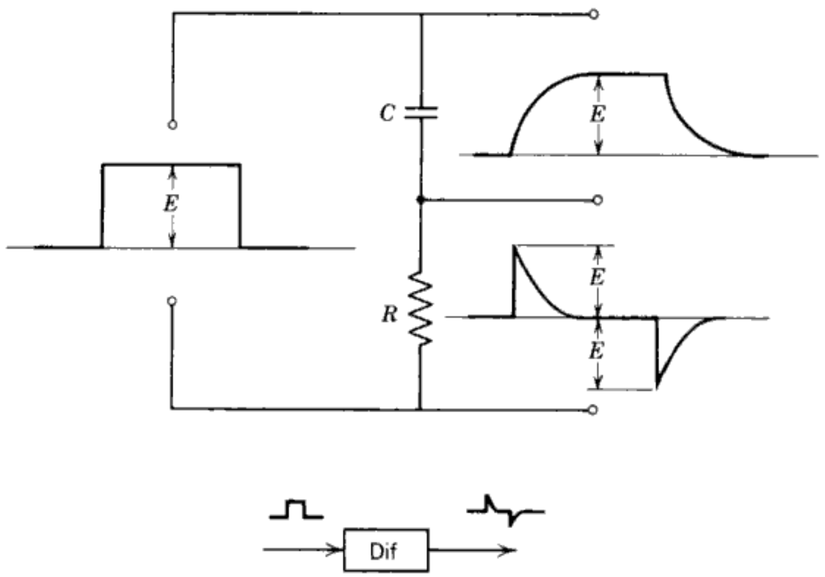


Fig. 42. The differentiator.

charged, no current flows through the resistor and thus the voltage across it is zero. At the terminating edge of the applied square wave, the resistor-capacitor combination acts the same way but in an opposite sense to produce a negative pulse. The shorter the charging time, the narrower the pulses produced by the differentiator.

Just as the capacitor stores electric energy, the inductor stores magnetic energy. When current i is fed to a coil in one direction, a clockwise magnetic field is formed. When current i is fed to the coil in the opposite direction, a counterclockwise magnetic field is produced (Fig. 43). As with the capacitor, it takes a definite amount of time to charge and discharge an inductor; so an inductor may therefore be used to obtain delay. By placing a piece of soft iron inside the magnetic field, the iron may be made to store magnetic energy. This phenomenon is discussed in greater detail in the following chapter.

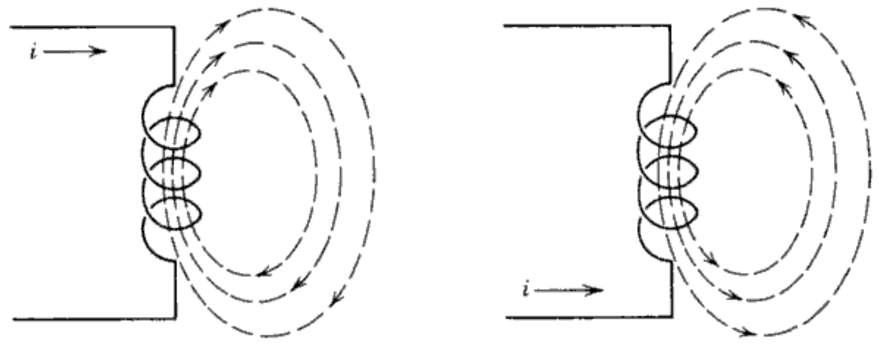


Fig. 43. The inductor as a storage component.

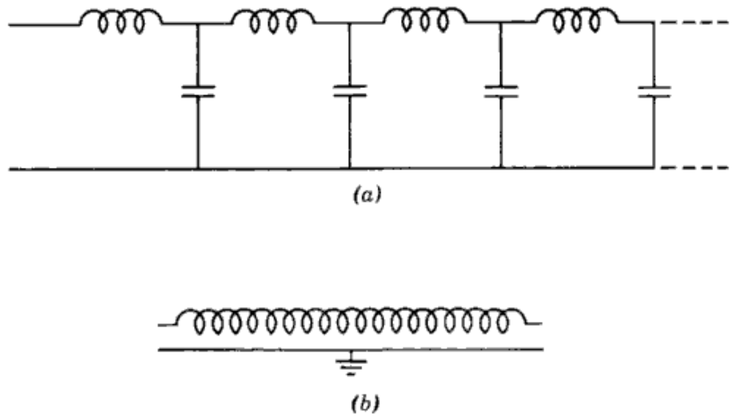


Fig. 44. Electric delay lines. (*a*) Lumped inductance and capacity. (*b*) Distributed inductance and capacity.

Capacitors and inductors may be connected together as shown in Fig. 44*a* to produce an artificial transmission line. Because it takes a definite amount of time for a pulse to travel to the end of this artificial transmission line, this circuit may be used as a delay element.

In the above circuit the inductance and capacity of a transmission line are lumped into individual inductors and capacitors. An artificial transmission line may also be derived by a cable consisting of an inner helical conductor insulated from a surrounding conductor. The inner conductor provides an inductance distributed throughout its length; and the inner and outer conductors acting together provide a capacity distributed throughout their length. The symbol for this type of artificial transmission line is given in Fig. 44*b*.

Vacuum Tubes and Gas Tubes

The diode consists of a filament, cathode, and plate in an evacuated bulb. When current is fed to the filament, it heats the cathode and

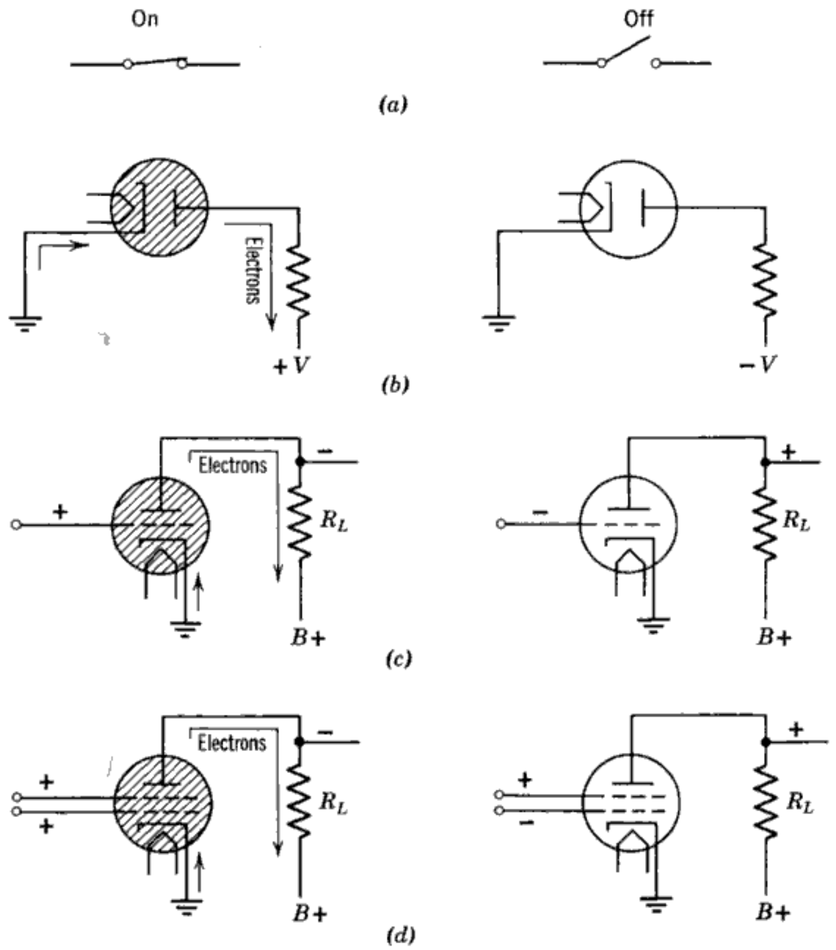


Fig. 45. Switching characteristics of diodes, triodes, and tetrodes. (*a*) Switch. (*b*) Diode. (*c*) Triode. (*d*) Tetrode.

causes electrons to evaporate from it. If the plate potential is positive relative to the cathode, the electrons from the cathode flow toward the plate. If the plate is negative relative to the cathode, the electrons do not flow toward the plate. The diode thus acts like a switch controlled by the potential applied to the plate. A positive plate potential closes the switch; a negative plate potential opens the switch (compare Figs. 45*a* and 45*b*).

By placing a grid between cathode and plate, a triode is formed. The triode can amplify because a small variation in potential at the grid may cause a larger variation in potential across the plate load R_L. The triode may be considered to be a switch too, because a highly positive potential on the grid causes lots of electrons to flow between cathode and plate, whereas a highly negative potential on the grid causes the tube to open, to be cut off (Fig. 45*c*).

A tetrode has two grids which may control the flow of electrons. Both the control grid and screen grid must be relatively positive to cause high conduction through the tube; if either grid is sufficiently negative, the tube is cut off (Fig. 45*d*). Similar considerations apply to vacuum tubes with more than two grids.

The vacuum tube by itself cannot be used as a storage cell because, once the signal is removed from the grid, the tube returns to its quiescent state. However, if a tube such as a triode is filled with gas, then, as soon as a positive trigger is applied to the grid, the tube conducts and remains conducting even after the trigger is removed because of the ionization of the gas. It stays conducting until the plate potential is brought down to below the ionization potential.

Cathode-Ray Tube

The ordinary cathode-ray tube stores information on its screen. The cathode-ray tube consists of an electron gun, vertical and horizontal deflection plates, an accelerating anode, and a phosphor screen. The electron gun shoots out electrons in a narrow beam. These electrons are accelerated toward the screen by the highly positive accelerating anode. Before reaching the screen, a potential between the vertical plates deflects the beam either up or down; and a potential between the horizontal plates deflects the beam either to the right or to the left. The deflection plates are thus capable of directing the narrow beam toward a designated tiny area on the screen.

The cathode-ray tube may be modified in a number of ways for binary storage. One method is as follows (see Fig. 46). The screen is replaced by an insulator mounted on a conducting back plate. On the opposite side of the insulator is a collector grid. The conductor, insulator, and grid form a capacitive arrangement. A source of voltage and a load resistor are connected between the conducting plate and the collector grid. The insulator is divided into 1024 squares, each of which is effectively a small storage cell. The voltage applied to the vertical deflection plates determines to which of 32 positions along the vertical the beam is deflected; the voltage applied to the horizontal deflection plates determines to which of 32 positions along the horizontal the beam is deflected. In this manner one of 1024 spots is chosen.

The potential applied on the collector grid determines whether a negative (ZERO) or positive (ONE) charge is stored on the chosen spot on the insulator. If the collector grid is negative, the electrons from the gun strike the insulator and charge it negatively. The insulator remains charged negatively because any secondary electrons escaping from it are repelled back to it by the negative charge on the collector grid. If the

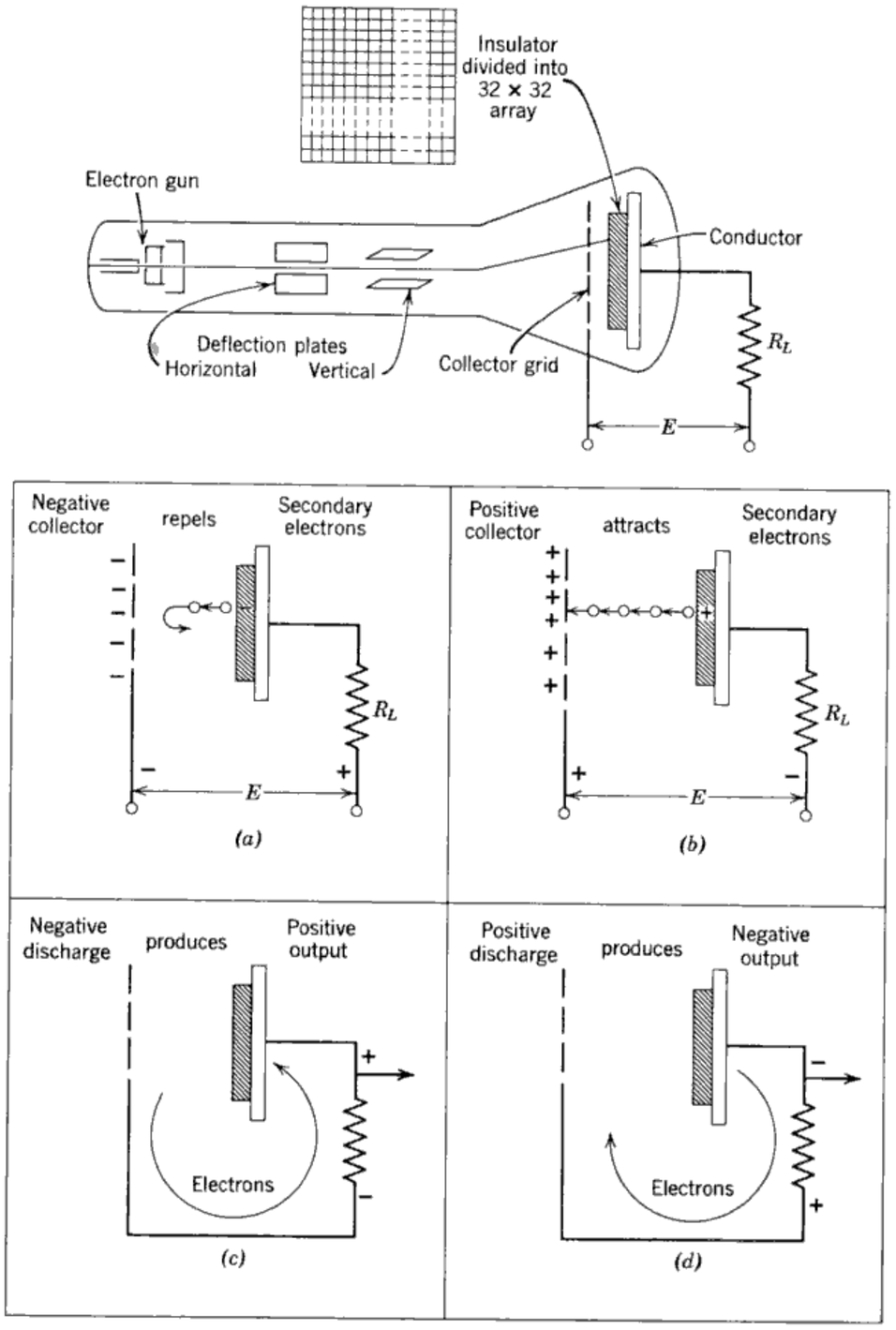


Fig. 46. Cathode-ray tube storage. (*a*) Write ZERO. (*b*) Write ONE. (*c*) Read ZERO. (*d*) Read ONE.

collector grid is positive, it attracts the secondary electrons, thus leaving the insulator positively charged. Therefore, application of a negative potential to the collector grid at the same time that a spot is irradiated causes the storage of a ZERO in that spot. Application of a positive potential to the control grid causes the storage of a ONE.

Reading is accomplished by removing the potential from the collector grid and using R_L to discharge the spot on the insulator. The spot to be read is chosen as before. If the area was storing a ZERO, it discharges and produces a positive pulse. If the area was storing a ONE, it discharges in the opposite direction to produce a negative pulse.

Cathode-ray tubes have been modified to project letters and numerals onto the screen. Each of the characters is stenciled out at a certain location on a thin sheet of metal. A pair of deflection plates directs the electron beam toward one of the characters. The beam shaped by the stenciled character may then be placed anywhere on the screen by means of another pair of deflection plates. In some systems a camera photographs the words beamed onto the screen.

Marginal Checking

Reliability is one of the most important considerations when choosing components for machine logic, and it may be greatly increased by preventive maintenance. One way of performing preventive maintenance on

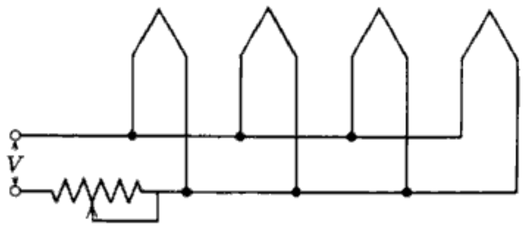


Fig. 47. Marginal checking.

systems composed of vacuum tubes is by marginal checking—checking of components at a tolerance extreme. For instance, a vacuum tube may be operational with its filament drawing current over a certain range. If the tube is in good condition, a decrease in filament current within this range does not affect the operation of the tube. But if the tube is on the verge of breakdown, bringing the filament current to its operating margin may cause the tube to become nonoperational. A simple potentiometer controlling the flow of current through several filaments may therefore be used for marginal checking (Fig. 47). All tubes which become nonoperational when filament current is reduced toward the margin of

tolerance are replaced before the equipment is placed into operation. Thus, possible causes of future trouble are removed before they can make the equipment inoperative.

SWITCHING CIRCUITS

Resistors

A simple resistor switching circuit is shown in Fig. 48. This 3-resistor combination may be considered to be an AND circuit for positive or negative signals, or an OR circuit for positive or negative signals, depending upon what the signals represent. If +6 v represents a class equal to ONE, and a signal between 0 and +4 v represents a class equal to ZERO, the circuit performs the AND function for positive signals. The circuit performs the OR function for negative signals if a voltage between 0 and +4 v defines a class equal to ONE and +6 v a class equal to ZERO. If 0 v defines a class

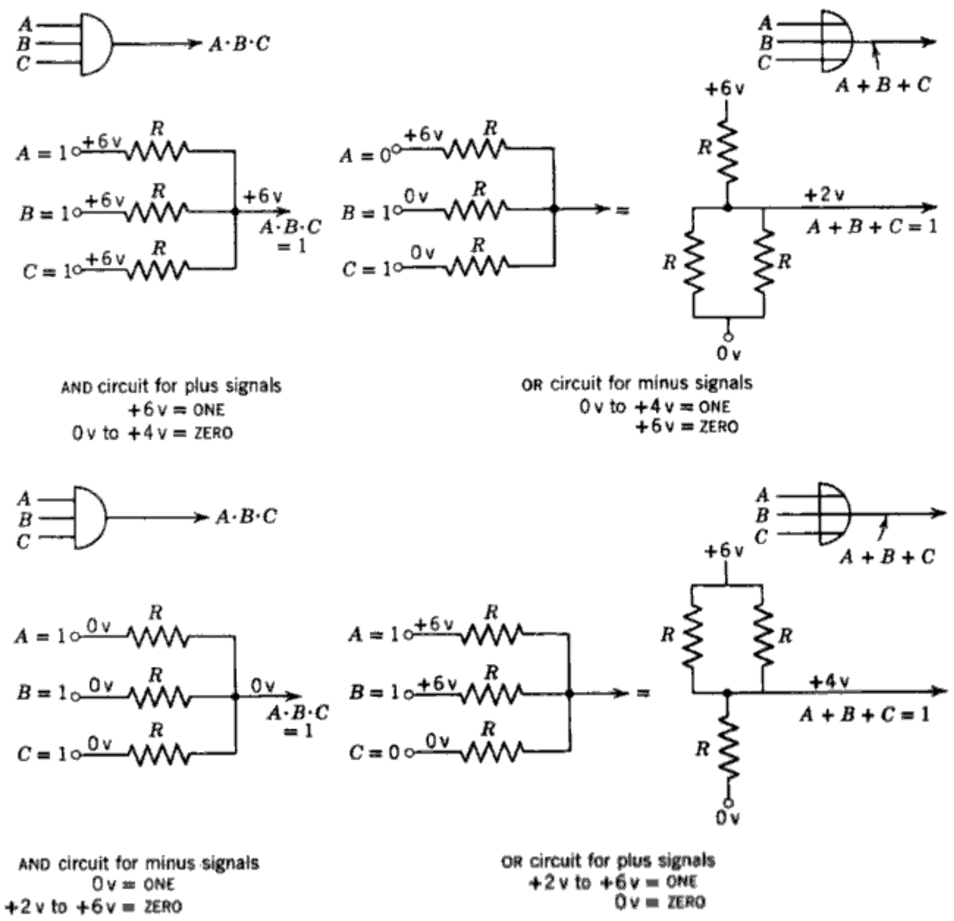


Fig. 48. Resistor switching circuits.

equal to ONE and a signal between +2 and +6 v a class equal to ZERO, the circuit performs the AND function for negative signals. It performs the OR function for positive signals if a signal between +2 and +6 v defines a class equal to ONE and 0 v a class equal to ZERO. All these facts follow from the voltage-divider action of the resistors, as shown in the illustration.

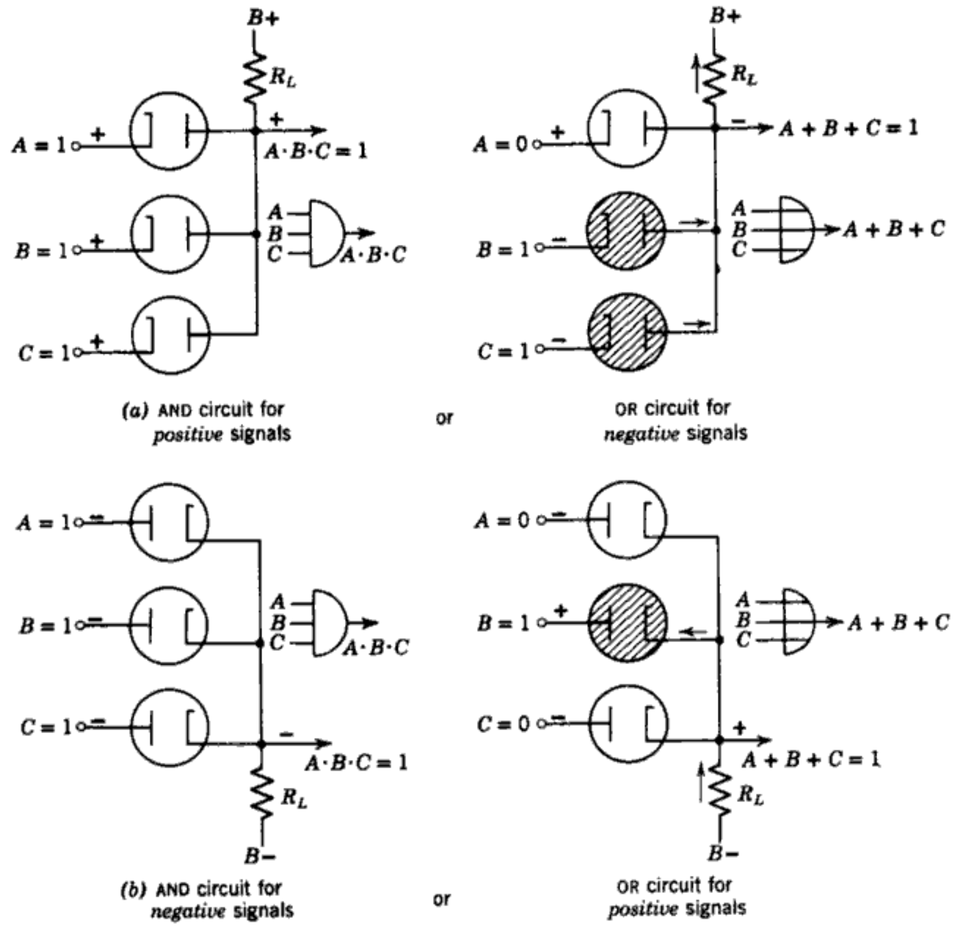


Fig. 49. Diode switching circuits. (*Note:* Hatching indicates conducting tubes.)

Diodes

A simple diode switching circuit which may be an AND or an OR is shown in Fig. 49*a*. The plates of the three diodes are connected through a load resistor to a positive supply voltage, and the cathodes are connected to either positive or negative potentials. If one of the cathodes is negative, electrons flow through the diode and the load resistor; the voltage drop across the load resistor makes the output voltage negative. If all cathodes

are positive, none of the diodes conduct and no electrons flow through the load resistor; the output voltage is positive. If a positive potential is considered to be ONE and a negative potential ZERO, this is an AND circuit. If a negative potential is considered to be ONE and a positive potential ZERO, this same circuit is an OR circuit.

Figure 49*b* shows an OR circuit for positive signals or an AND circuit for negative signals. When any input is positive, the associated diode conducts and the resultant voltage across the load resistor is positive. If all plates are negative, none of the diodes conduct and the output is negative.

In almost all cases AND circuits for positive signals are equivalent to OR circuits for negative signals, and OR circuits for positive signals are equivalent to AND circuits for negative signals. For simplicity, all switching circuits are discussed under the assumption that positive signals are applied, unless otherwise stated.

Triodes and Tetrodes

The circuit shown in Fig. 50*a* is an INVERTED AND. The two grids are normally biased to cutoff. Only when the potentials applied to both grids are *positive* do electrons flow to produce a *negative* output voltage. This circuit produces the inverse of an AND circuit or $\overline{A \cdot B}$. The illustration shows circuit operation when $A = 1$ and $B = 1$. Under these circumstances, both grids are positive, the tube conducts, the electron flow produces a negative output. Therefore, the output is

$$\overline{A \cdot B} = 1$$

A triode connected as a voltage amplifier produces a NOT. Two triodes sharing a common load resistor produce an INVERTED OR (Fig. 50*b*). Both tubes are biased to cutoff. If a positive signal is applied to either grid, electrons flow through the load resistor and a negative output voltage is produced. This means that this circuit produces $\overline{A + B}$. The illustration shows that with A equal to ONE and B equal to ZERO, tube $T2$ conducts and $T3$ is cut off. Electrons from $T2$ flowing through R_L produce a negative output potential. Therefore, the output is

$$\overline{A + B} = 1$$

A common load resistor may be used to combine the outputs of several AND circuits by means of the OR function. The 3-tube circuit shown in Fig. 50*c*, for instance, produces $\overline{A \cdot B + C \cdot D + E}$. The combination of signals A and B produces electron flow through $T4$ and through the load resistor. The combination of signals C and D produces electron flow through $T5$ and through the load resistor. Similarly, a signal E

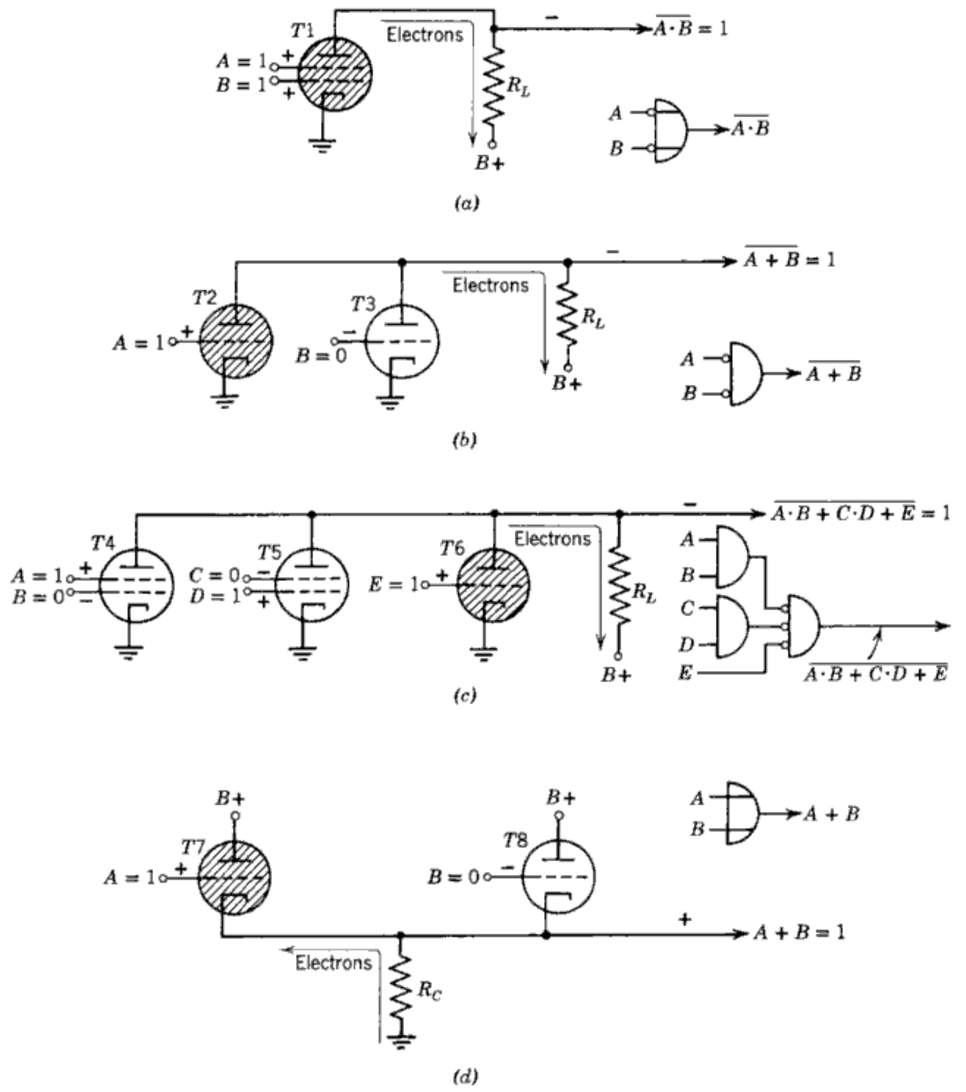


Fig. 50. Vacuum-tube switching circuits. (*a*) INVERTED AND. (*b*) INVERTED OR. (*c*) AND INVERTED OR combination. (*d*) OR circuit—cathode follower.

applied to $T6$ causes electron flow through $T6$ and the load resistor. Any one of these three combinations, therefore, produces a negative output. In the illustration $A = 1$, $D = 1$, and $E = 1$; $T6$ conducts, and the electrons flowing through R_L produce a negative output. Therefore, the output is

$$\overline{A \cdot B + C \cdot D + E} = 1$$

Cathode followers with a common cathode resistor produce the OR function without inversion. If either grid is positive, a positive potential

appears at the output. In the circuit of Fig. 50*d*, $A = 1$, and $T7$ conducts. As a result, electrons flowing through R_C produce a positive potential. The output voltage is

$$A + B = 1$$

Binary Adder

In the last chapter the expressions for the sum and carry of a binary adder were simplified to

$$S = C'(\overline{\bar{A} \cdot B + A \cdot \bar{B}}) + \overline{C'}(\bar{A} \cdot B + A \cdot \bar{B})$$
$$C = A \cdot B + C'(\bar{A} \cdot B + A \cdot \bar{B})$$

The expression for the sum may be modified. If we make the following substitutions,

$$X = \bar{A} \cdot B + A \cdot \bar{B}$$
$$\bar{X} = \overline{\bar{A} \cdot B + A \cdot \bar{B}}$$

we may write the S equation as

$$S = C' \cdot \bar{X} + \overline{C'} \cdot X$$

Since the universal class U is equal to

$$U = C' \cdot \bar{X} + \overline{C'} \cdot X + C' \cdot X + \bar{C}' \cdot \bar{X}$$

then

$$C' \cdot \bar{X} + \bar{C}' \cdot X = \overline{C' \cdot X + \bar{C}' \cdot \bar{X}}$$

Substituting back the value for X and $\bar{X}$, the sum equation becomes

$$S = \overline{\bar{C}' \cdot (\overline{\bar{A} \cdot B + A \cdot \bar{B}}) + C' \cdot (\bar{A} \cdot B + A \cdot \bar{B})}$$

The vacuum-tube binary-adder circuit which produces this expression for S and the former expression for C is shown in Fig. 51. The output of $T4$ and $T5$ is $\overline{\bar{A} \cdot B + A \cdot \bar{B}}$. This signal is applied to the screen of $T7$ and to one resistor of the resistive OR circuit. $\bar{C}'$ is applied to the other resistor of the switch. The output of the resistor switch is inverted by $T6$ to produce

$$\overline{\bar{C}' + (\overline{\bar{A} \cdot B + A \cdot \bar{B}})} = C' \cdot (\bar{A} \cdot B + A \cdot \bar{B})$$

The output of $T3$, $\bar{C}'$, is also fed to the grid of $T7$. Across the common load resistor for $T7$ and $T8$ is produced the sum S.

The output of $T6$ is applied to $T9$ while A and B are applied to $T10$. The output from across the common load resistor for $T9$ and $T10$, after an inversion, is

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

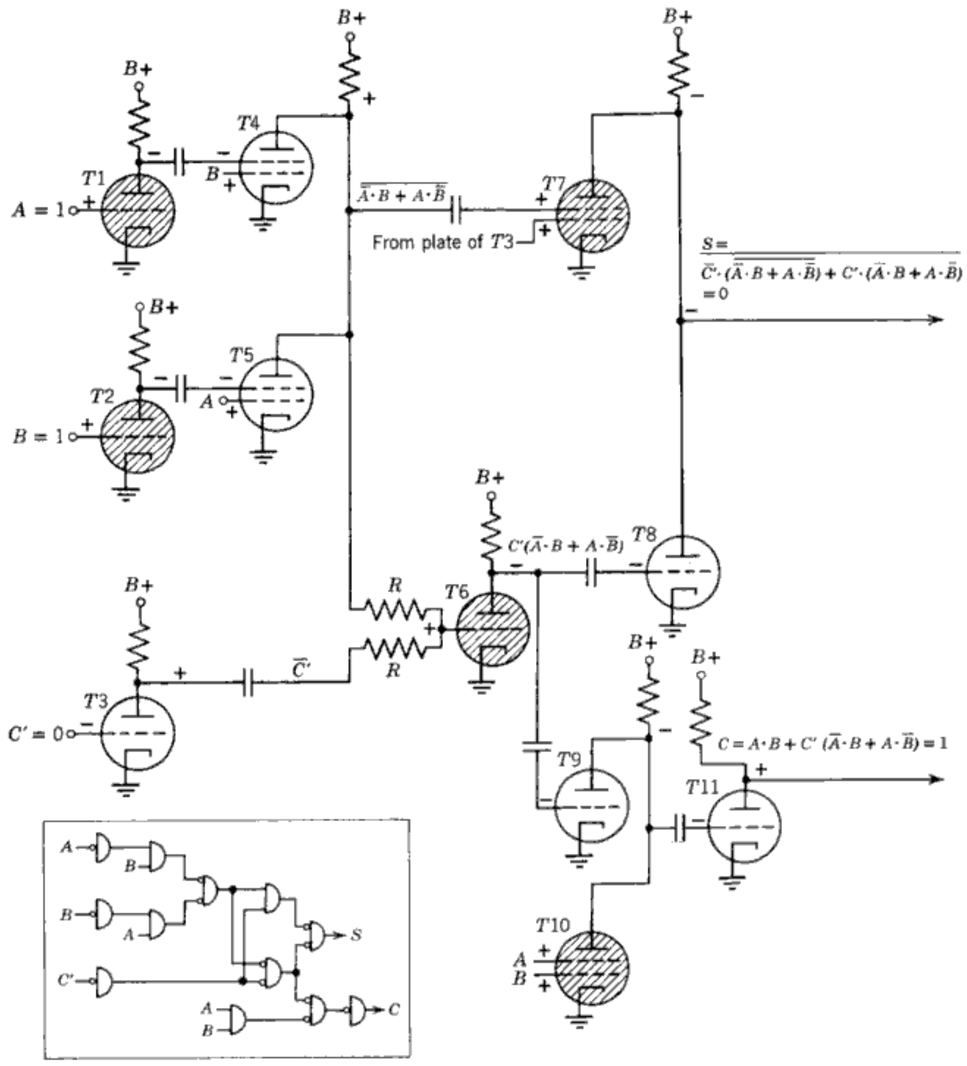


Fig. 51. Vacuum-tube binary adder.

The schematic illustrates the operation of the circuit when $A = 1$, $B = 1$, and $C' = 0$. $T1$ and $T2$ conduct, and $T4$ and $T5$ do not. The resultant positive output of $T4$–$T5$ is applied to the screen of $T7$. Since $C' = 0$, $T3$ is cut off, and a positive potential is applied to the grid of $T7$ too. Tube $T7$ conducting produces a negative output pulse for S, or

$$S = 0$$

The positive $\bar{C}'$ and the positive signal from $T4$–$T5$ cause $T6$ to conduct. The resultant negative potential keeps $T8$ and $T9$ cut off. But since both

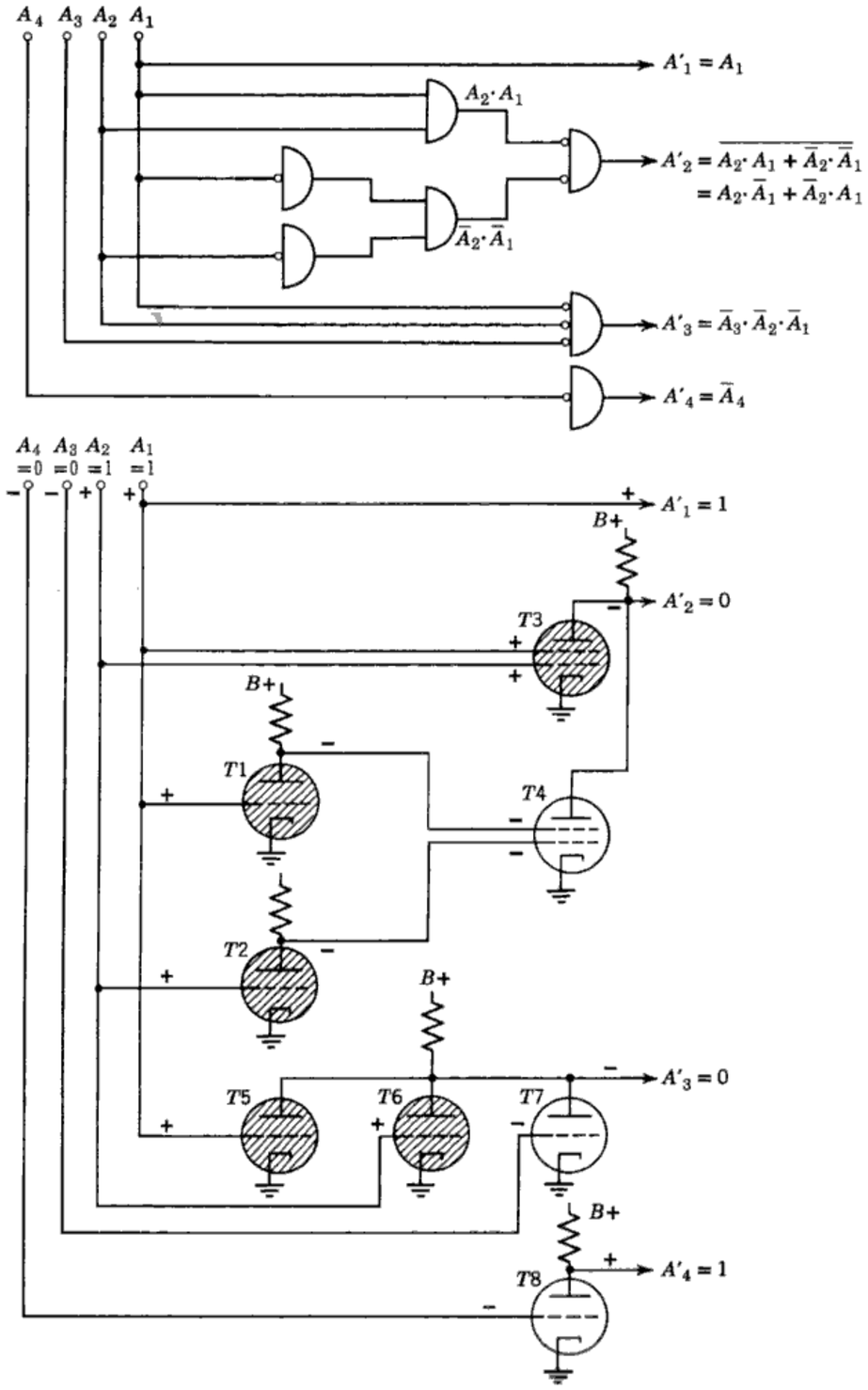


Fig. 52. 5421 Complementer.

A and B are positive, $T10$ conducts to produce a negative pulse which keeps $T11$ cut off. $T11$, therefore, produces a positive output for C, or

$$C = 1$$

5421 Complementer

The equations previously derived for the 9's complementer in the 5421 coded-decimal system are:

$$A'_1 = A_1$$
$$A'_2 = A_2 \cdot \bar{A}_1 + \bar{A}_2 \cdot A_1$$
$$A'_3 = \bar{A}_3 \cdot \bar{A}_2 \cdot \bar{A}_1$$
$$A'_4 = \bar{A}_4$$

A vacuum-tube circuit which performs these functions is shown in Fig. 52. A_1 and A_2 are fed to $T3$; and $\bar{A}_1$ and $\bar{A}_2$ are fed to $T4$. The output of $T3$–$T4$ is

$$\overline{A_2 \cdot A_1 + \bar{A}_2 \cdot \bar{A}_1} = A_2 \cdot \bar{A}_1 + \bar{A}_2 \cdot A_1 = A'_2$$

The output of INVERTED OR $T5$–$T6$–$T7$ is

$$\overline{A_3 + A_2 + A_1} = \bar{A}_3 \cdot \bar{A}_2 \cdot \bar{A}_1 = A'_3$$

A'_4 is obtained by feeding A_4 through a simple NOT, $T8$.

The illustration shows circuit operation when 0011 or decimal 3 is fed to it. The grids of $T1$, $T2$, $T5$, and $T6$ are made positive; and those of $T7$ and $T8$ negative. The first output line A'_1 is obviously positive. The A'_2-output line is negative since both grids of $T3$ are positive, causing the tube to conduct. The A'_3-output is negative because of the current flowing through $T5$ and $T6$. A'_4 is positive because of the negative potential on the grid of $T8$. The output thus represents 1001 or decimal 6, which is the 9's complement of 3.

AMPLIFYING CIRCUITS

If information is conveyed by means of voltage levels, simple voltage amplification is needed to maintain the separation between the two d-c levels. If a ONE is conveyed by a pulse and a ZERO by no-pulse, then feedback amplifiers are needed to re-form the pulse with regards to time as well as to amplitude.

D-C Level System

Because the voltage at the output of a resistor switching circuit varies so much, the switch is usually connected directly to an amplifier (as in *T*6, Fig. 51). By making the amplifier operate between cutoff and saturation, two distinct voltage levels are produced. At saturation the tube is a short circuit, and the potential on the plate is approximately ground. At cutoff the potential at the plate is approximately at *B*+.

If diodes are used for switching, after several stages of diode circuits an amplifier must be introduced to increase the distance between the two signal levels. The level separation at the output of the amplifier is fairly high. But as the signal passes through several other diode-circuit stages, the d-c level separation becomes small again and another amplifier must be introduced.

For switching circuits which are also amplifiers, not as much voltage amplification is needed. In this case it is best to have the voltage-level separation at the output practically the same as the voltage-level separation at the input. Operating the tube between cutoff and saturation is one way of accomplishing this end.

Pulse–No-Pulse System

A pulse traveling through any kind of network eventually loses its shape and may be delayed, as shown in the first waveform of Fig. 53*c*. To restore the shape and also to retime it—bring it in step with a master clock—a pulse reshaper is used.

A simple pulse reshaper using feedback is shown in Figs. 53*a* and 53*b*. A signal input has no effect until a clock pulse is fed to the AND circuit. Once the clock pulse arrives, the tube conducts, and the feedback voltage from the output transformer back to the input keeps the tube conducting until the end of the clock pulse. Without the clock pulse the feedback voltage cannot pass through the AND circuit to the grid. The output of the pulse reshaper, therefore, is a pulse of proper amplitude accurately timed with the clock pulse.

Naturally, if a no-pulse arrives at the input, the tube remains cut off and no pulse is produced at the output.

STORAGE CIRCUITS

Diode-Capacitor Storage Cell

The capacitor can store a positive or a negative charge, but by itself it cannot be a storage cell. It needs to be combined with two diodes which

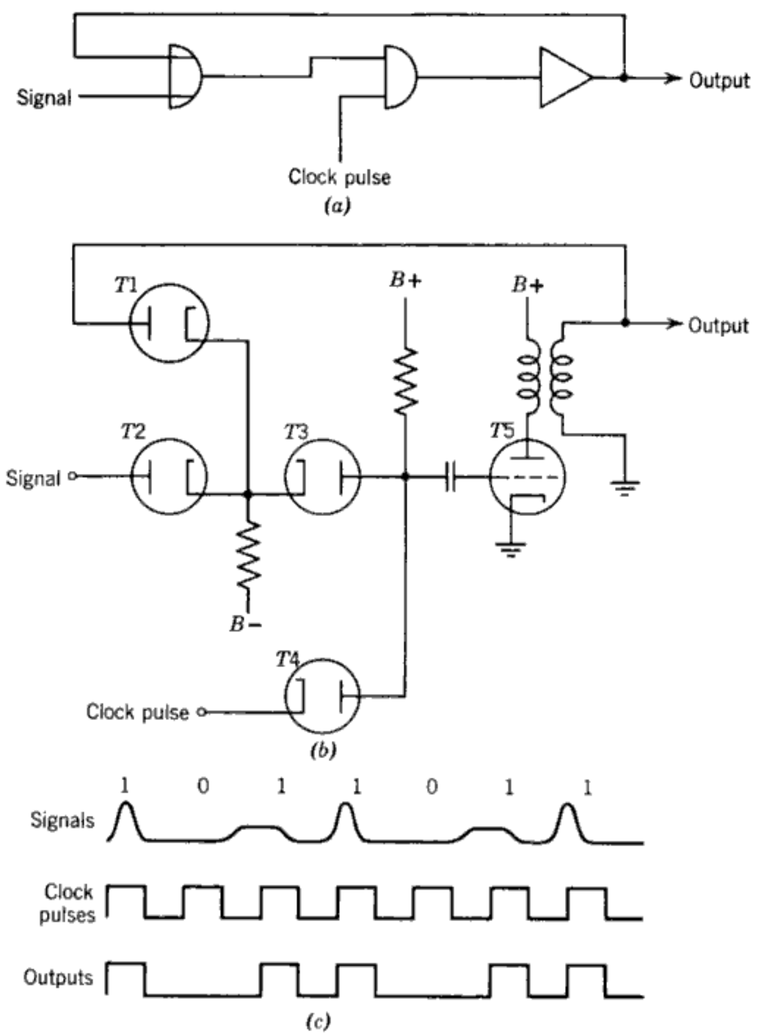


Fig. 53. Pulse reshaper. (*a*) Block diagram. (*b*) Schematic. (*c*) Waveforms.

prevent the capacitor from discharging when holding information, and which allow writing and reading when voltages are applied to them (Fig. 54). A diode-capacitor storage cell is useful in an information scheme where a positive pulse represents ONE and a negative pulse represents ZERO.

Writing a ZERO is accomplished by applying a negative pulse across R_L and no voltages to the diodes. As a result, electrons flow through *T*1 to charge the capacitor as shown. Writing a ONE is accomplished by applying a positive pulse across R_L and no voltages to the diodes. As a result, electrons flow through *T*2 to charge the capacitor in the opposite direction, as shown. After writing, a positive potential is applied to the cathode of

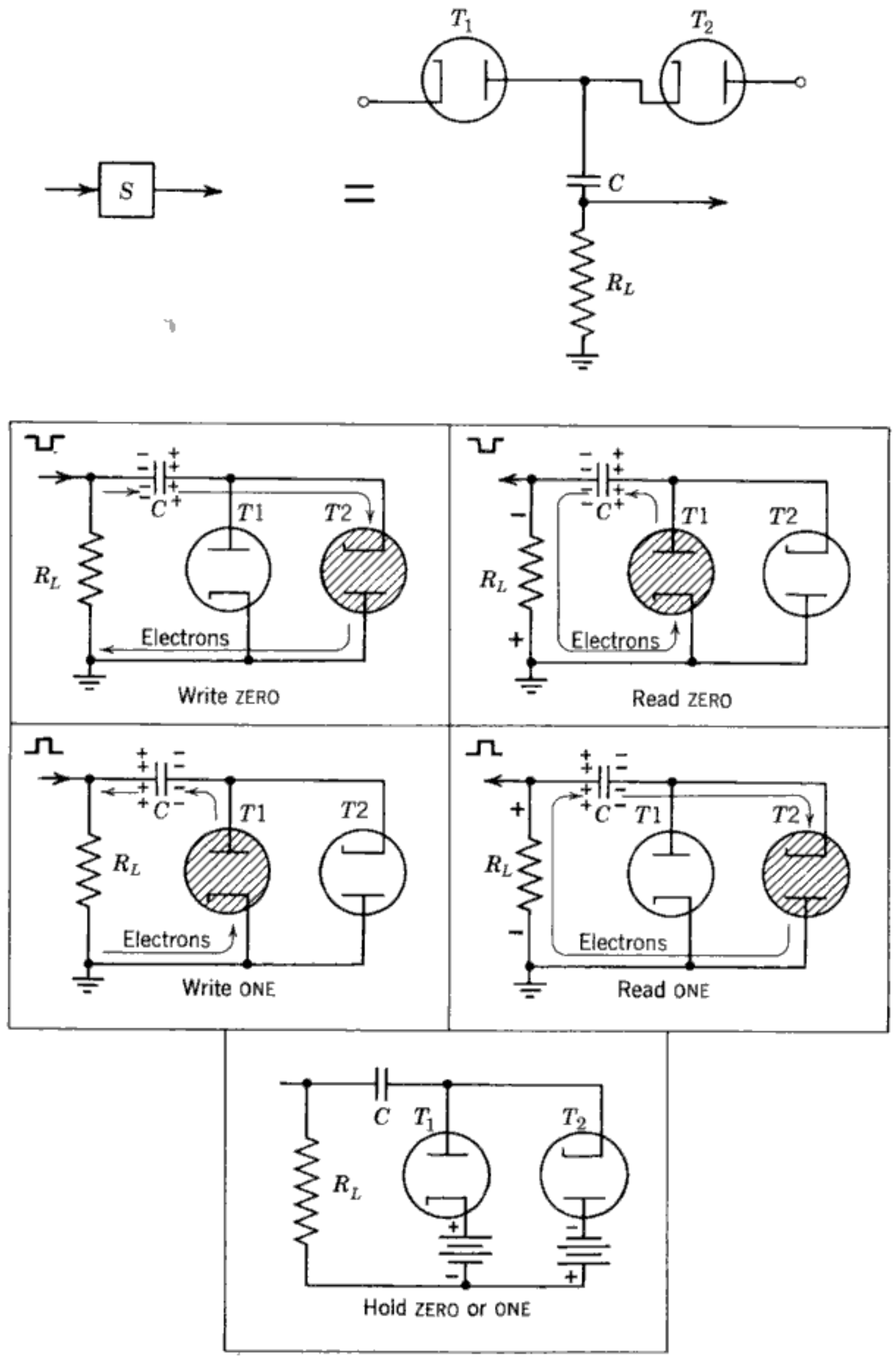


Fig. 54. Capacitor-diode storage cell.

$T1$ and a negative potential to the plate of $T2$ to prevent the charge on the capacitor from leaking off.

Reading is accomplished by removing the potentials from the diodes and allowing the capacitor to discharge. If the cell was storing a ZERO, the electrons discharge through $T2$ and produce a negative pulse across R_L. If the cell was storing a ONE, the electrons discharge through $T1$ to produce a positive pulse across R_L.

Flip-Flop

The flip-flop is a storage cell with two inputs and two outputs. A pulse applied to the set input flips the circuit to the ONE state; a pulse applied to the reset input flops the circuit to the ZERO state.

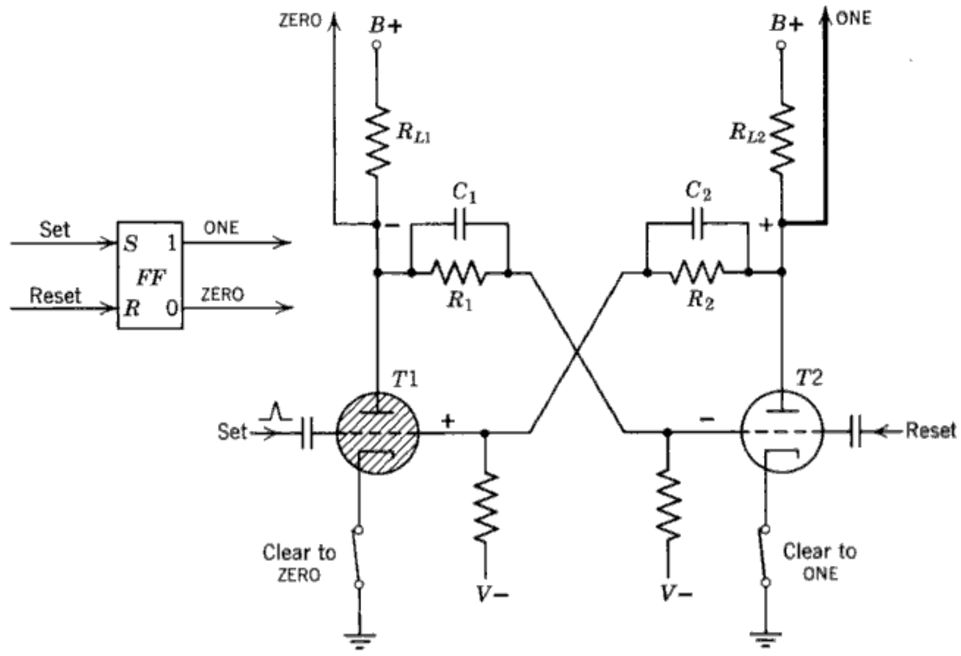


Fig. 55. Triode Flip-flop.

A simple triode flip-flop is shown in Fig. 55. At equilibrium one tube is conducting and the other is cut off. If $T1$ is nonconducting and $T2$ is conducting, a positive pulse applied to the grid of $T1$ causes it to conduct. The resultant plate current makes the plate negative. This negative potential when applied to the grid of $T2$ causes $T2$ to conduct less. As a result the voltage across R_{L2} decreases causing the plate of $T2$ to become more positive. This positive potential when fed back to the grid of $T1$ causes $T1$ to conduct more. This process is cumulative until $T1$ reaches saturation and $T2$ is cut off.

If now a positive pulse is applied to the grid of $T2$, $T2$ conducts and, as a result, the voltage at the plate becomes more negative; this negative potential fed to the grid of $T1$ causes $T1$ to conduct less. This process is cumulative until $T1$ is cut off and $T2$ reaches saturation.

A positive pulse applied to the grid of $T1$ is a set pulse. It places the flip-flop in the ONE state: $T1$ conducting and $T2$ cut off, the ONE output line positive and the ZERO output line negative. Figure 55 shows the flip-flop in the set or ONE state. A positive pulse applied to the grid of $T2$ is a reset pulse. It places the flip-flop in the ZERO state: $T2$ conducting and $T1$ cut off, the ZERO output line positive and the ONE output line negative.

Negative pulses produce opposite effects from positive pulses. A negative pulse applied to the grid of $T1$ *resets* the flip-flop. A negative pulse applied to the grid of $T2$ *sets* the flip-flop.

The flip-flop may be placed manually in the ZERO state by opening the switch in the cathode of $T1$. Since $T1$ cannot conduct, $T2$ does conduct and the ZERO output line becomes positive. We say that the switch *clears* the flip-flop to ZERO. Similarly, opening the switch in the cathode of $T2$ clears the flip-flop to ONE.

Shift Register

A shift register is a register consisting of storage cells, in which bits may be shifted from one cell to the next by means of shift pulses. Flip-flops and modified delay elements may be used as storage cells.

Several flip-flops may be combined by means of AND circuits and delay elements to produce a shift register such as that shown in Fig. 56. Information is applied to this register one bit at a time through the input gate A_1. Each bit is first applied to $FF1$. Each succeeding shift pulse advances this bit to the following flip-flop, at the same time bringing a new bit to $FF1$. The AND circuits between each pair of flip-flops allow a positive pulse from the ONE output line to set the following flip-flop, and a positive pulse from the ZERO output line to reset the following flip-flop. The delay element makes sure that a flip-flop does not flip before the flip-flop has had a chance to transfer its information to the following stage.

The illustration shows a 3-stage shift register storing 101. When a shift pulse is applied, the positive potential from the $T2$ plate is fed to the grid of $T3$ to turn on $T3$. As a result $FF2$ stores a ONE. At the same time, the negative potential from the plate of $T4$ is being applied to the grid of $T5$ to cut off $T5$. As a result $FF3$ stores a ZERO. The ONE of $FF3$ is lost.

If the output of $FF3$ is applied through an AND circuit to the input of $FF1$, the shift pulse causes the ONE from $FF3$ to be shifted into $FF1$. Upon completion of the operation, the register stores 110.

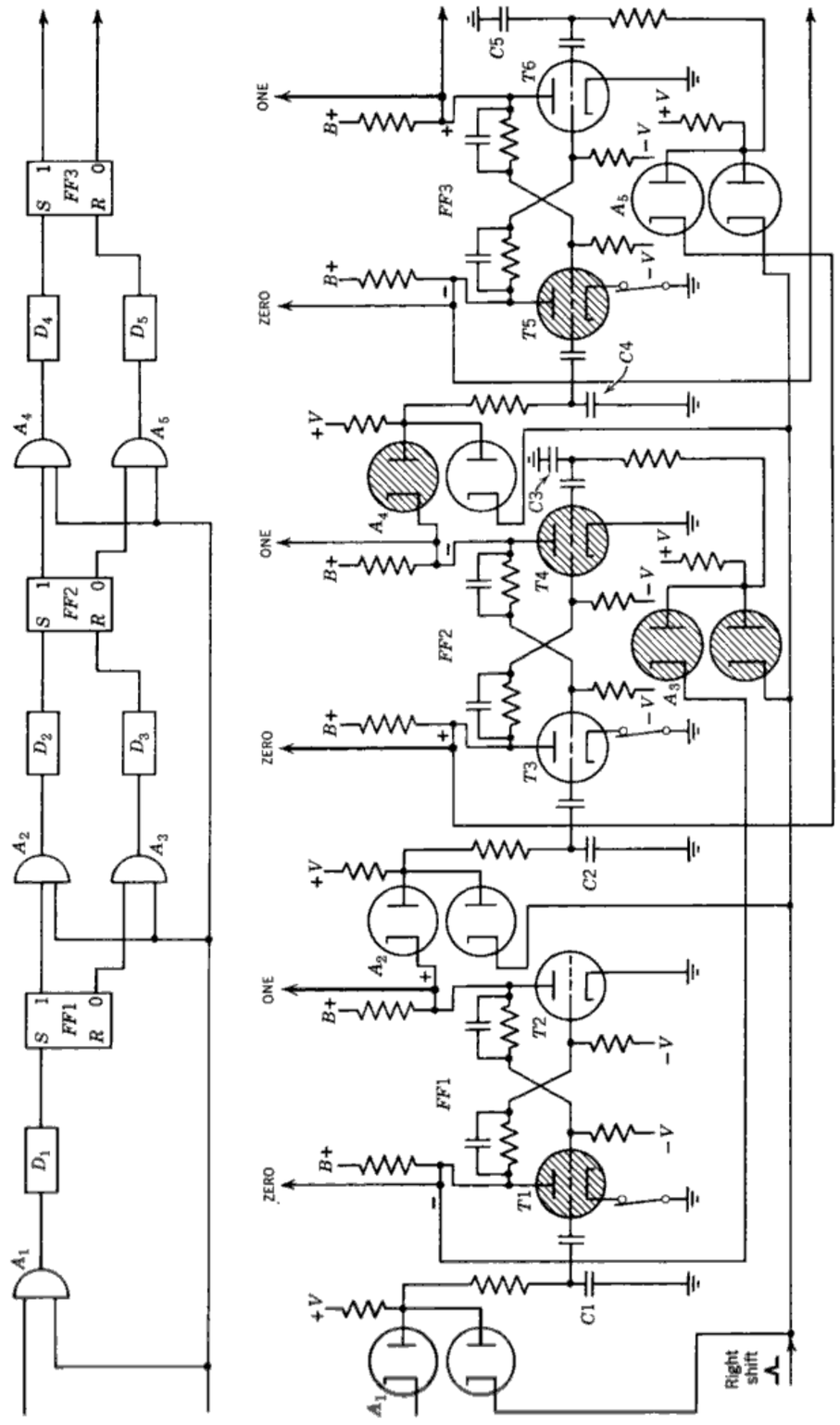


Fig. 56. Shift register composed of flip-flops.

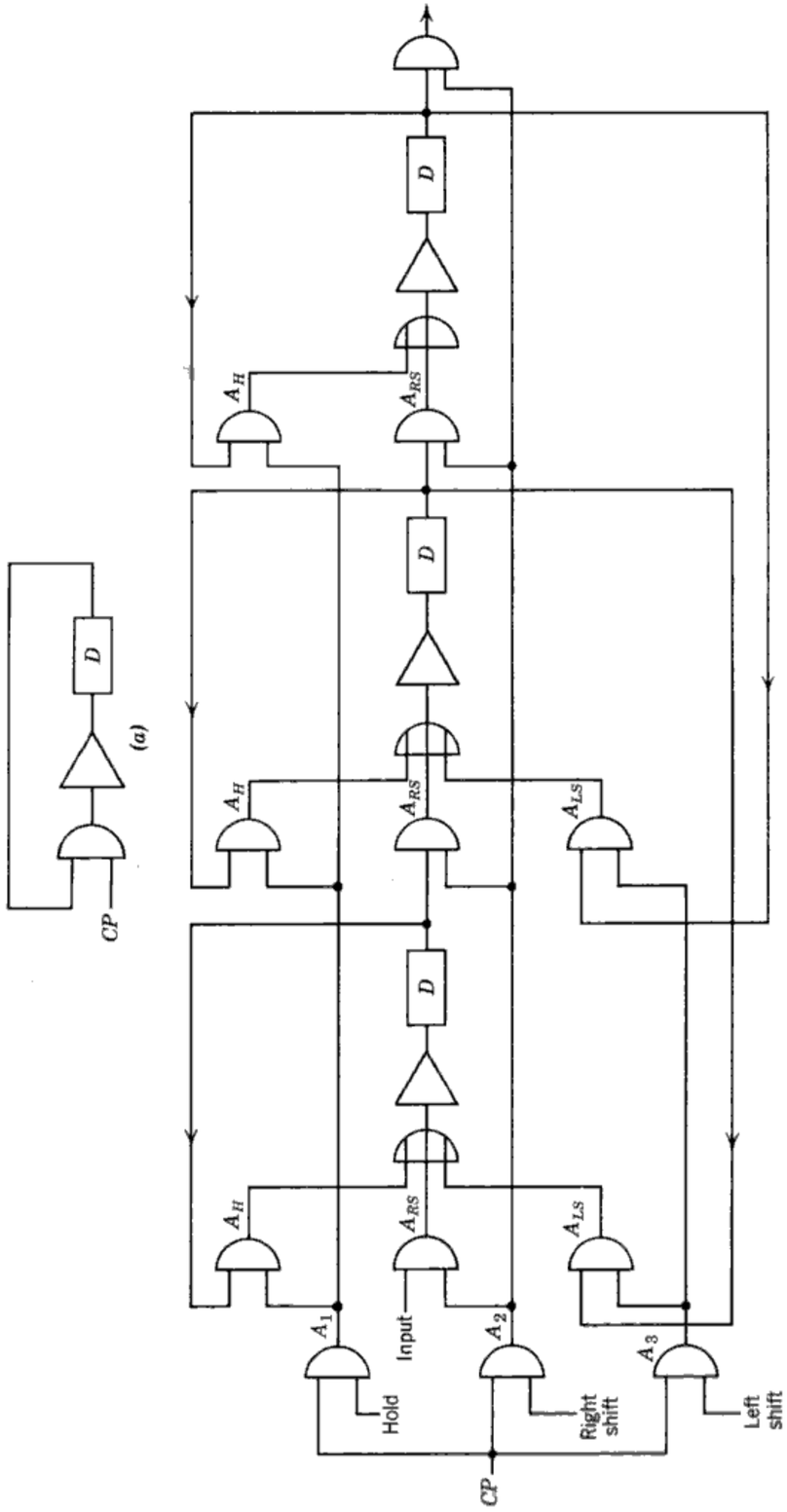


Fig. 57. Shift register composed of delay elements with feedback. (*a*) Storage cell. (*b*) Shift register.

In the foregoing circuit all the information is shifted to the right. It is just as easy to connect outputs from right-hand flip-flops to inputs of left-hand flip-flops. In this case a shift pulse would cause the information to be shifted to the left.

A shift register may be built of storage cells composed of delay elements and feedback. One such storage cell is shown in Fig. 57*a*. It is composed of an AND circuit, amplifier, and delay element. Each time the signal reaches the end of the delay element, it is fed back to the gate. A clock pulse allows the pulse through the gate to recirculate once more. As long as clock pulses are applied, the signal goes round and round the loop—it is stored.

A shift register built of these storage cells and capable of shifting information in and out, and to shift bits to the right or to the left is shown in Fig. 57*b*. Clock pulses (*CP*) are applied at a constant rate to A_1, A_2, and A_3. To keep information stored in the register the hold signal is energized. As a result AND circuit A_1 allows the *CP* to be applied to all the A_H switches, and the signal in each storage cell goes round and round in its loop in synchronism with the applied *CP*.

To shift information to the right, the right-shift signal is energized. As a result AND circuit A_2 allows the clock pulses to be applied to all the A_{RS} switches. Instead of recirculating, the information leaving each delay line is applied to the storage cell to its right. Similarly if the left-shift line is energized, A_3 allows clock pulses to be applied to all the A_{LS} switches. As a result, the information reaching the end of each delay element is fed to the storage cell to its left. Upon completion of the shift, right or left, the hold signal must be reapplied to keep the information in the register.

Note that no *CP* are needed with the flip-flop register. A shift pulse may be applied at any time to shift the information. In the delay type of register *CP* are applied constantly.

Counters

With a slight modification the flip-flop may be converted to a complementing flip-flop, a circuit which always reverses states upon application of an input pulse. The complementing flip-flop plus a differentiator form a building block for a binary counter.

The binary counter building block is shown in Fig. 58. Except for diodes *D*1 and *D*2 and differentiator R_1-C_1, the circuit is an ordinary flip-flop. Input pulses are applied to the two diodes simultaneously. If *T*1 is conducting and *T*2 is cut off—the flip-flop is storing ONE—the positive input pulse has no effect on the grid of *T*1 which is already positive. But the positive potential does increase the potential at the grid of *T*2, causing *T*2 to start conducting. The resultant lowered potential on the

plate of $T2$ makes the grid of $T1$ more negative, causing $T1$ to conduct less. The effect is cumulative: eventually $T2$ reaches saturation and $T1$ is cut off—the flip-flop is in the ZERO state.

The next positive pulse applied does not affect the grid of $T2$ but does raise the potential at the grid of $T1$. As a result $T1$ conducts, causing $T2$ to cut off—the flip-flop is brought back to the ONE state.

If each time the flip-flop is flipped to ZERO a positive pulse is fed to another complementing flip-flop, this second flip-flop also flips back and

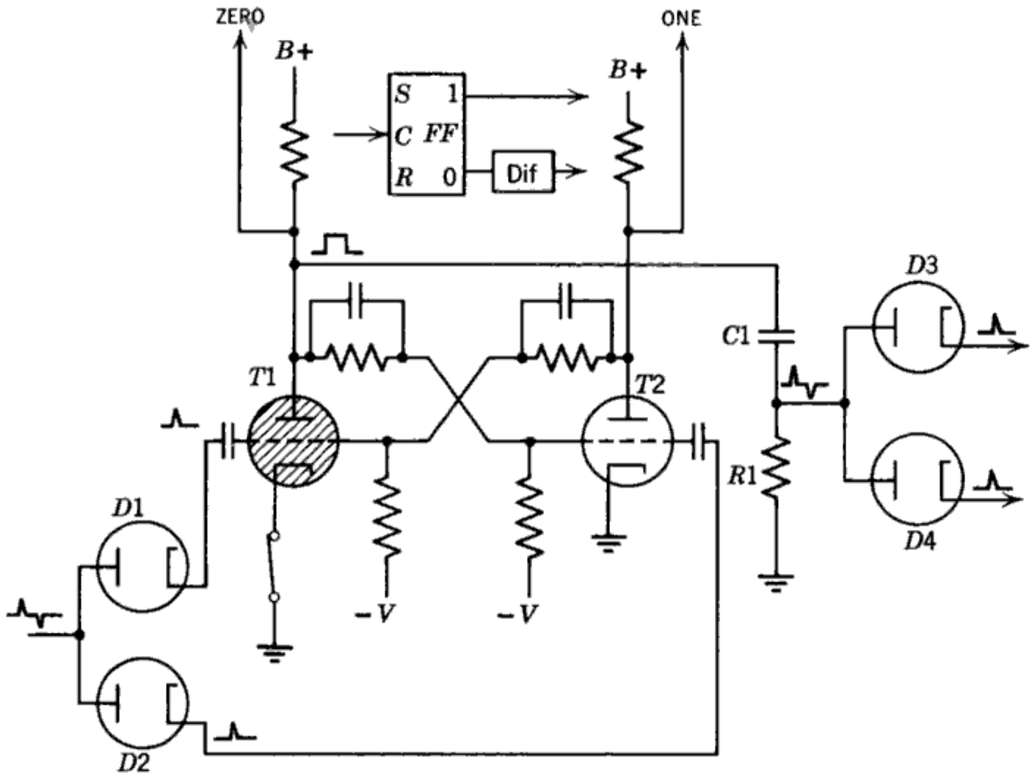


Fig. 58. Binary counter building block. (*Note:* Hatching indicates tube conducting before application of input pulse.)

forth but half as frequently. If each time the second flip-flop is flipped to ZERO a positive pulse is applied to still another complementing flip-flop, this third flip-flop also flips back and forth but one-quarter as frequently. The result of placing in tandem such a group of complementing flip-flops is a binary counter (see Fig. 59).

The positive square-wave output of $T1$ produced when the flip-flop is flipped to ZERO is not a good trigger. It is not definite in its action to flip a following flip-flop because the positive potential is applied to both grids.

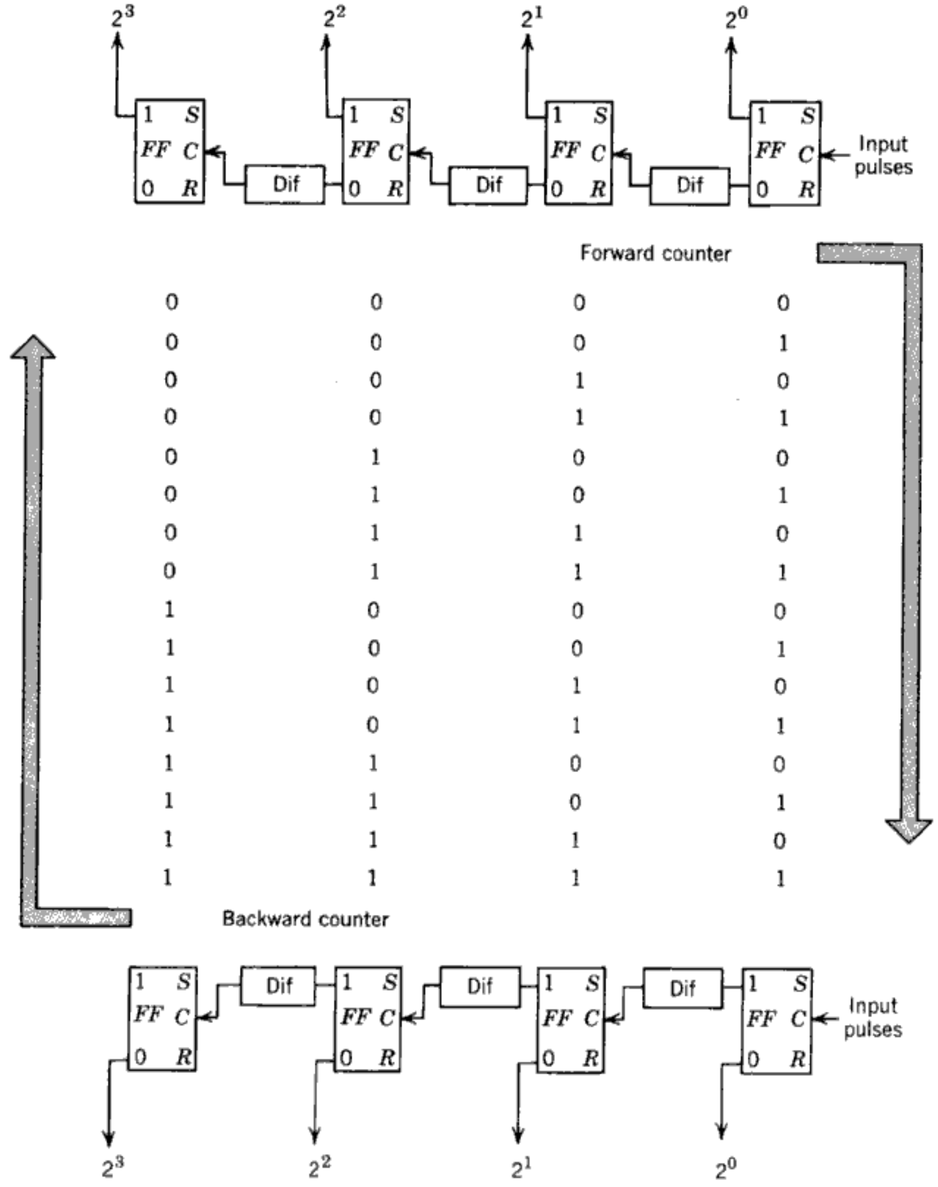


Fig. 59. Binary counter.

When one tube of the following flip-flop starts to conduct and tries to pull the grid of the other tube negative, this applied positive potential counteracts this effect. A narrow positive pulse produces a surer flipping action. This narrow pulse is obtained by feeding the square wave to differentiator $C1$-$R1$. $D3$ and $D4$ allow only the positive pulse to trigger the following flip-flop.

The modulus of a 2-stage counter is 2^2; of a 3-stage counter 2^3; of a

4-stage counter 2^4. In other words, the modulus is 2 to an exponent given by the number of counting stages.

The positive pulse produced by the differentiator when the associated flip-flop flips to ZERO is called a carry; the carry adds a ONE to the succeeding column. If the flip-flops are connected in such a way that a carry pulse is produced when flipping from ZERO to ONE, the result is a counter which counts backward (see bottom of Fig. 59).

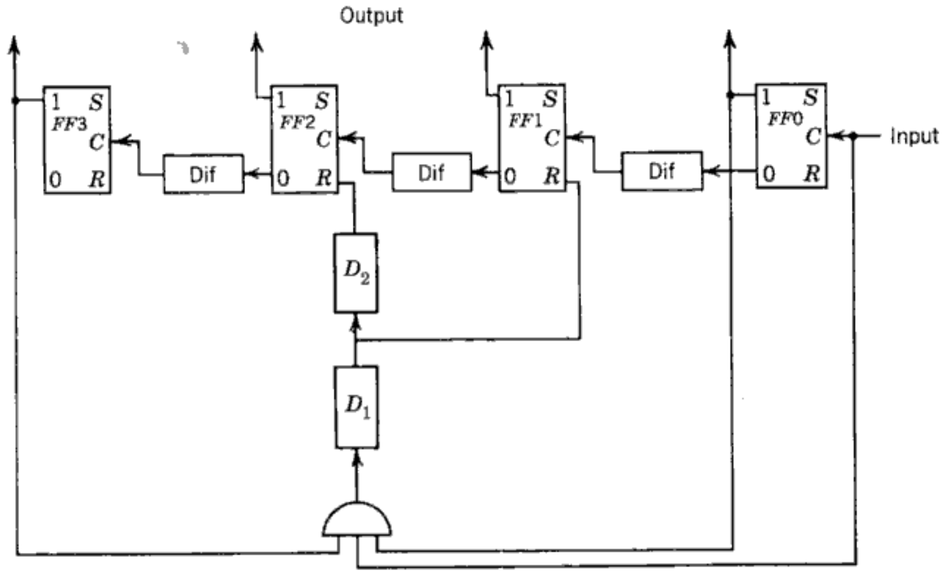


Fig. 60. Decimal counter.

A 4-stage counter counts to 16. By removing the last six binary combinations in the sequence, this counter can be made to count from 0 to 9. The last six binary combinations are removed by means of the AND circuit and delay elements shown in Fig. 60. The counter operates normally until it reads 1001 or binary 9. When this occurs, *FF*0 and *FF*3 are storing a ONE—they prime the AND circuit. When the next count pulse arrives, *FF*0 is flipped to ZERO, and the AND circuit allows *FF*1 and *FF*2 to be reset to ZERO. Delay D_1 makes sure *FF*1 is flipped to ZERO after the carry from *FF*0 had flipped *FF*1 to ONE. Delay D_2 makes sure *FF*2 is flipped to ZERO after the carry from *FF*1 had flipped *FF*2 to ONE. Flipping *FF*2 to ZERO produces a carry which flips *FF*3 to ZERO too. The tenth input pulse, therefore, makes the counter read 0000. Further counting is in normal binary until 1001 is again reached.

Another type of counter is the ring counter. A ring counter is a device consisting of a ring of interconnecting bistable elements, only one of which can be in a specified state at any one time. The ring counter may be

likened to a commutator where each successive pulse causes a new line to be chosen.

An 8-stage ring counter composed of thyratrons is shown in Fig. 61. The grid of each stage is connected to the cathode of the previous stage through a voltage divider. The voltage divider in the cathode of *T*8 connects back to the grid of *T*1. Thus all the stages are in a continuous ring.

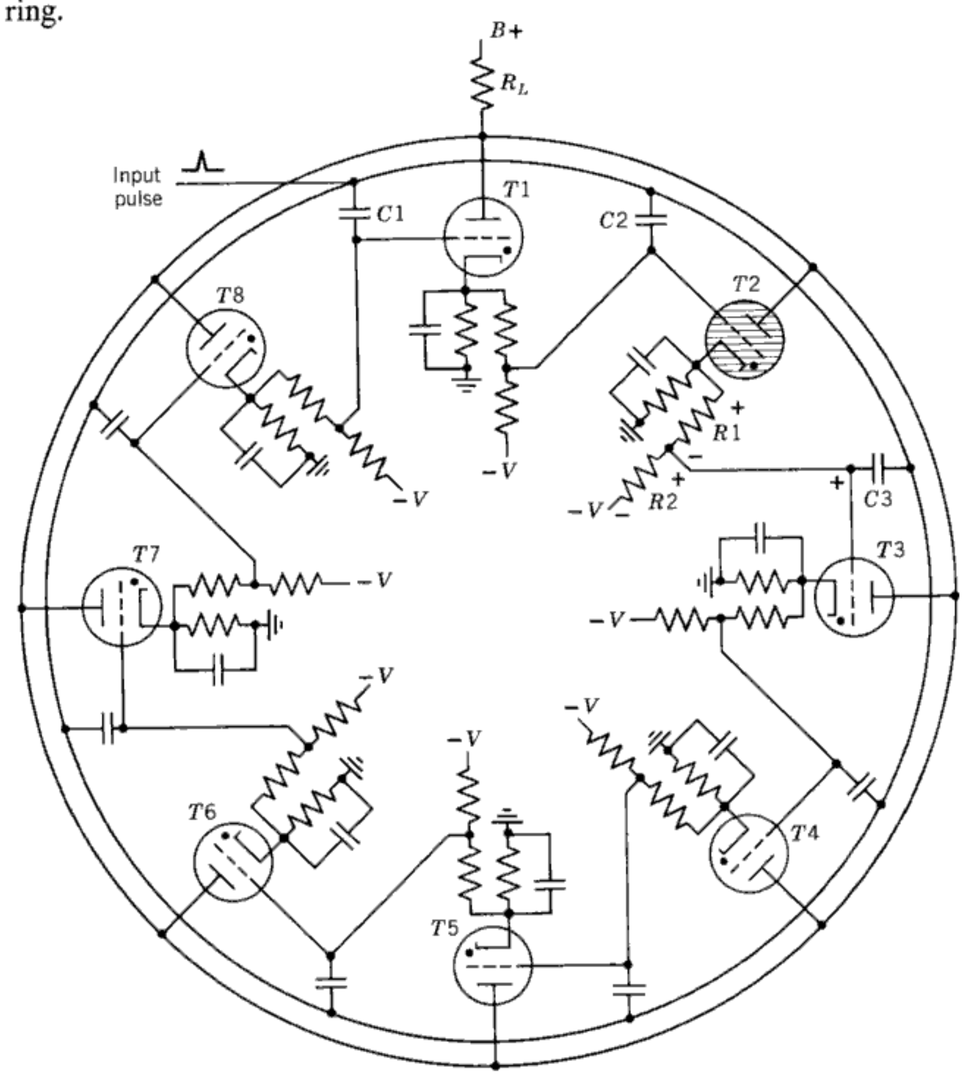


Fig. 61. Ring counter.

Only one tube conducts at any one time. Current flowing through the voltage divider in the conducting stage maintains a positive potential on the grid of the following stage. This positive potential is not enough to cause the following stage to conduct, but it is enough to prepare it for

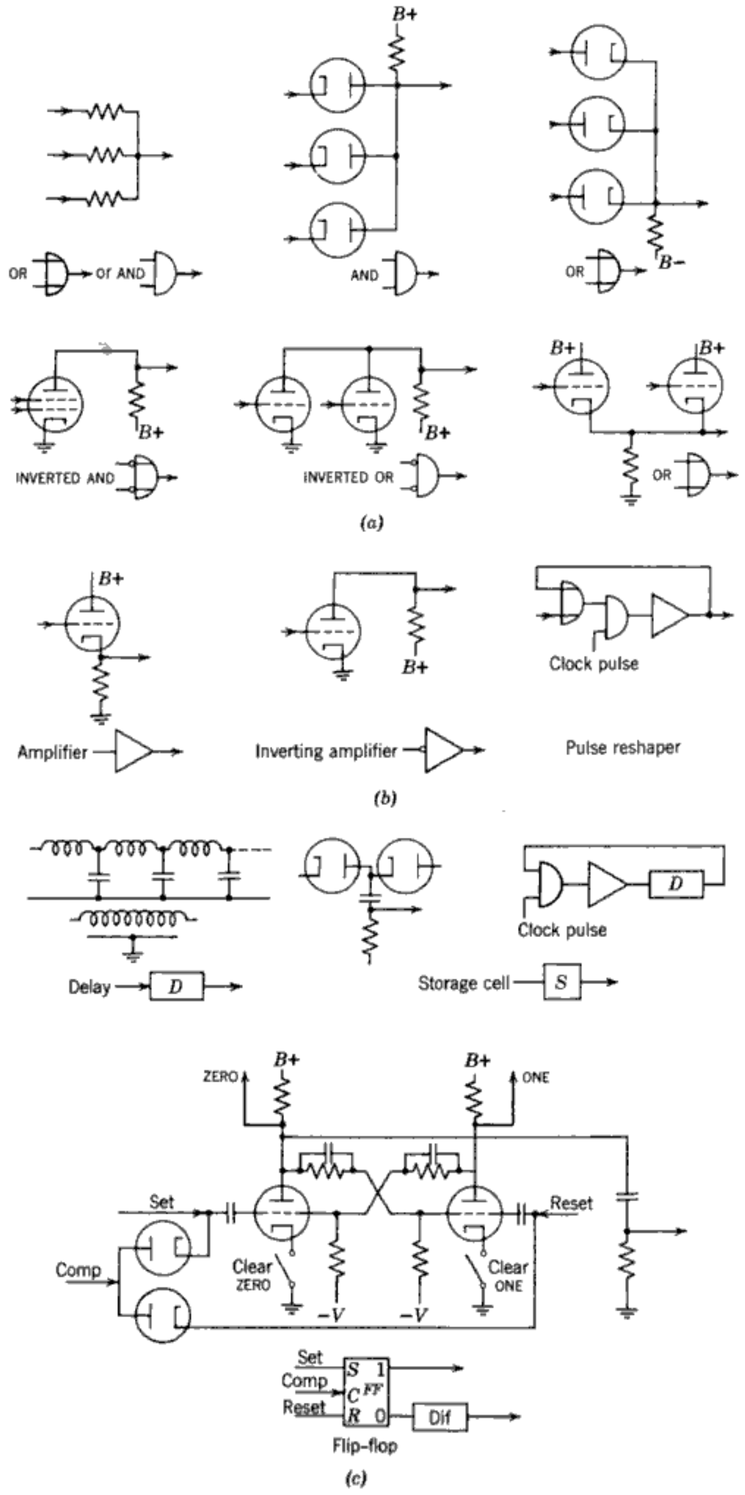


Fig. 62. Electronic building blocks. (*a*) Switching. (*b*) Amplifying. (*c*) Storage.

conduction—to prime it. When the next positive input pulse is applied, it has no effect on any stage except the one which is primed. The primed stage conducts and lowers the potential across R_L enough to cause the previous stage to stop conducting. The voltage divider in the newly conducting stage now primes the succeeding stage so that the next input pulse will make the following tube conduct.

The illustration shows $T2$ conducting. The positive potential across $R2$ primes $T3$. The next positive pulse applied through $C3$ to the grid of $T3$ increases its potential enough to cause $T3$ to conduct.

SUMMARY OF ELECTRONIC BUILDING BLOCKS (Fig. 62)

1. The resistor voltage divider acts as a switching circuit by controlling the voltage output. The diode circuit acts as a switching circuit by controlling the flow of current.

2. Because of the relationship between AND and OR circuits, an AND circuit for positive signals is equivalent to an OR circuit for negative signals.

3. The INVERTED AND function is produced by a vacuum tube with several grids; the INVERTED OR function is produced by a common load resistor for several vacuum tubes.

4. A simple voltage amplifier inverts the polarity of the signal. A cathode follower does not invert the polarity, and does not increase the amplitude of the signal. A pulse reshaper standardizes the amplitude and width of a pulse signal.

5. Inductance and capacity may be used as delay elements. The capacitor may form the basis for a storage cell if the charge and discharge are controlled by diodes. A delay line may be converted to a storage cell by means of feedback. Feedback can also convert two amplifiers into a flip-flop.

6. *Definitions to Remember*

FLIP-FLOP—A 1-bit storage device with two input terminals and two output terminals. The device remains in either state until caused to change to the other state by application of the corresponding signal. In either state the signal on one output terminal is out of phase with the signal on the other output terminal.

COMPLEMENTING FLIP-FLOP—A flip-flop which always reverses states upon application of an input signal.

CLEAR—To restore a storage device to a prescribed state, usually ZERO.

SHIFT REGISTER—A register consisting of discrete storage cells, and in which bits may be advanced from one cell to the other by means of applied shift pulses.

DIFFERENTIATOR—A circuit which produces narrow pulses coincident with and in the same direction as the leading and trailing edges of an applied square wave.

BINARY COUNTER—A device or circuit which counts in the binary number system—produces numbers consisting of ONE-ZERO combinations—when pulses are applied to its input.

RING COUNTER—A device consisting of a loop of interconnecting bistable elements, only one of which can be in a specified state at any one time. Each successive applied pulse causes another element in the loop to switch to the specified state.

CHAPTER 7

Electromagnetic components

Instead of electric energy, magnetic energy may be stored. Instead of turning an electric circuit on and off, an electromagnetic circuit may be turned on and off. Instead of changing impedance by applying a voltage to the grid of a tube to swing it into and out of saturation, the impedance of a magnetic core may be changed by applying a pulse of current to a primary to swing the core into and out of magnetic saturation. Instead of using electrostatic coupling in a vacuum tube to produce amplified signals in the plate circuit, electromagnetic coupling may be used in a transformer to produce amplified signals in the secondary winding.

In other words, logical circuits may be built of electromagnetic components as well as of vacuum-tube components. But there is one important difference between the two types of components: electronic tube components may operate with direct current; electromagnetic components cannot. Direct current flowing through a vacuum tube produces a voltage drop across a load which can be detected by another circuit. Not so with magnetic energy flowing through a magnetic core. Only when magnetic energy is *changed*, is a voltage induced across a winding. This means that only dynamic techniques, pulse–no-pulse or plus-pulse–minus-pulse combinations, may be used with electromagnetic components.

ELECTROMAGNETIC LOGICAL ELEMENTS

Electromagnetic Fundamentals

Two principles form the basis for almost everything that follows: transformer action and the hysteresis loop. The first shows the relationships among current, magnetic fields, and induced voltages. The second shows how magnetic energy is stored.

Figure 63 shows a transformer: an iron core around which are wound primary and secondary windings. If a pulse of current is applied to the

primary, the magnetic field varies, thus inducing a voltage across the secondary. As the current increases, the lines of flux increase and a positive potential is produced across the secondary. During the flat portion of the applied current wave the current is not changing and neither is the magnetic flux. As a result, no voltage is induced across the secondary. When the current drops, the lines of flux collapse, and a negative potential is produced across the secondary.

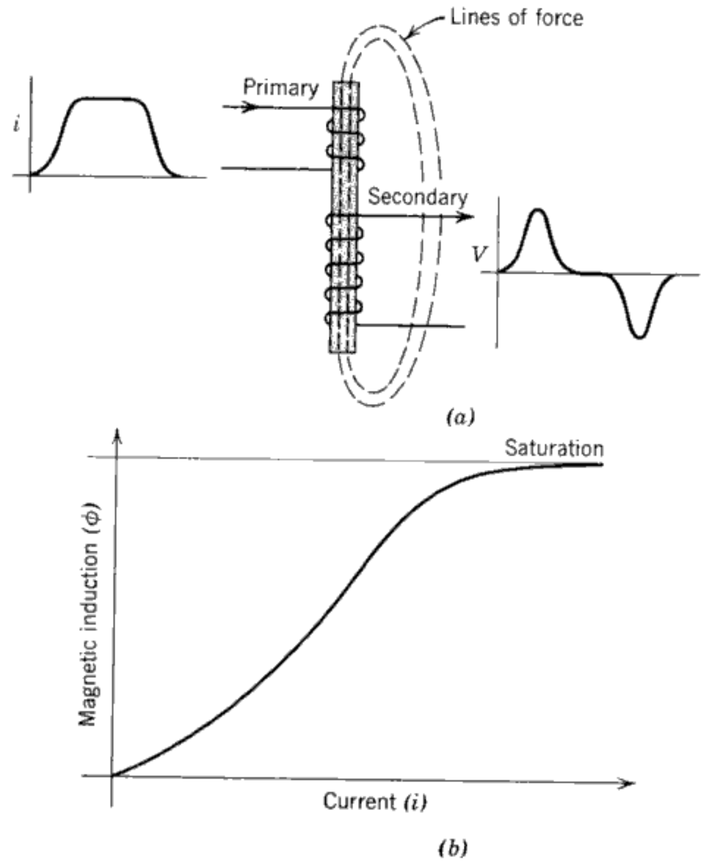


Fig. 63. Transformer action. (*a*) Transformer. (*b*) Magnetic induction curve.

This description of transformer action assumes that the magnetic core has a linear characteristic, that is, the magnetic induction of the core at all times varies with the applied magnetizing force which, in turn, varies with the applied current. But the magnetization curve is actually as shown in Fig. 63*b*. The magnetic induction varies directly with the magnetizing force up to a point. After this point, any increase in applied magnetizing force has a very small effect on the magnetic induction. This point on the curve is called the point of magnetic saturation. At magnetic saturation

very few lines are cut by the secondary and very little voltage is induced.

This nonlinear curve represents the behavior of a magnetic core when subjected to a magnetic field for the first time. After it has once become magnetized, changes in current flow through a winding cause the magnetic induction of the core to follow the hysteresis loop shown in Fig. 64. If after the core reaches magnetic saturation in the positive direction (point

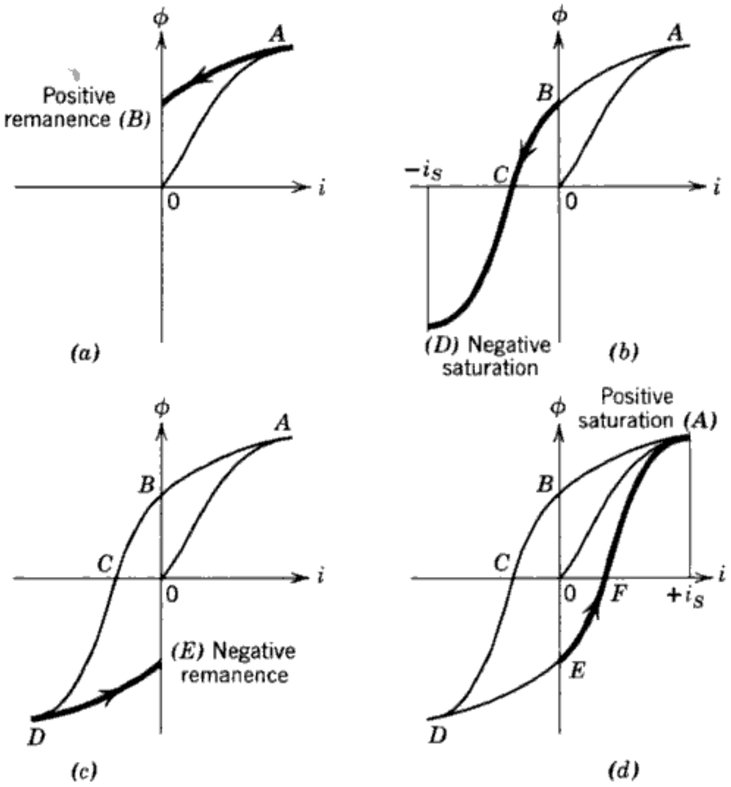


Fig. 64. The hysteresis loop. (*a*) Positive current removed. (*b*) Negative current applied. (*c*) Negative current removed. (*d*) Positive current applied.

A, Fig. 64), the current through the winding is removed, then the magnetic induction does not drop to zero but to the positive remanence point (point *B*). If now current is allowed to flow through the winding in the opposite direction, the resultant magnetizing force makes the magnetic induction of the core follow loop *BCD* to the saturation point in the negative direction (point *D*). When the current is stopped, the magnetic induction falls to the negative remanence (point *E*). Reversing the current flow now causes the magnetic induction to vary along *EFA*. Thus, instead of following the same curve upward and downward, hysteresis makes the core follow one curve upward and another curve downward.

The shape of the hysteresis loop depends on the type of material used for a core. Iron cores such as those used for transformers have a nonsquare loop (Fig. 65*a*); ferrite and molypermalloy cores such as those used for

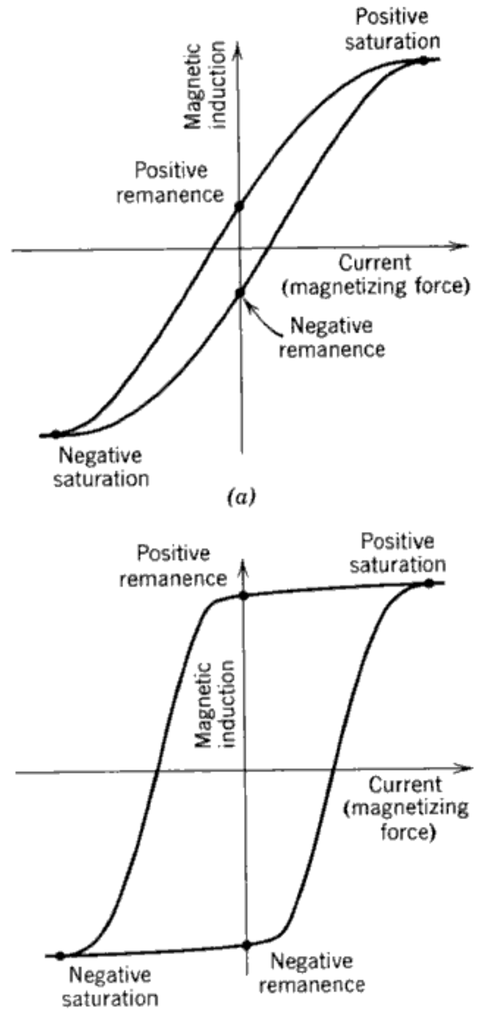


Fig. 65. Difference between square-loop and nonsquare-loop cores. (*a*) Nonsquare-loop core. (*b*) Square-loop core.

many computer applications have a square loop (Fig. 65*b*). Note that the core with the nonsquare loop does not have a well-defined saturation line and the remanence points are fairly high.

Because it is needed for an understanding of practically all electromagnetic components, the hysteresis loop is explained by means of the domain theory. In brief, the domain theory states that a magnetic material is composed of many tiny magnets. With no magnetizing force

applied, these tiny magnets are oriented in a random manner, causing no total field in any direction. With a magnetic force applied, the magnets are forced to line up. Once the magnets are fully aligned, further application of magnetic force has no effect on the magnetization of the core—saturation is reached.

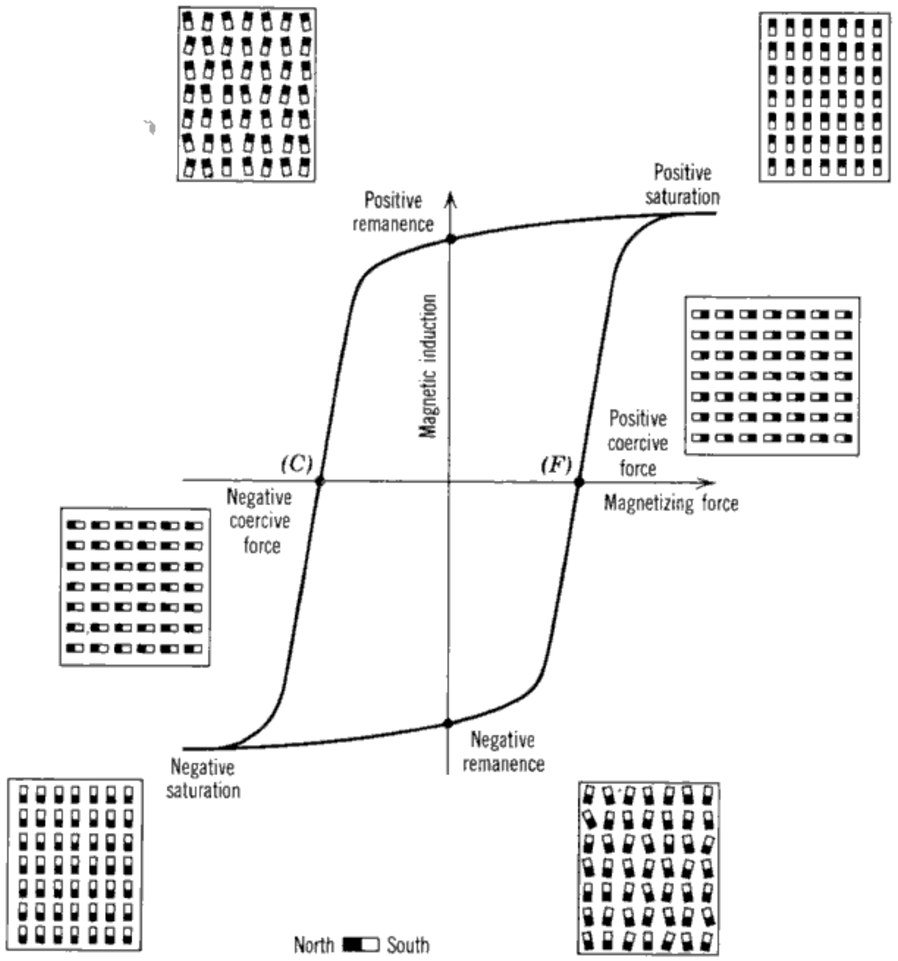


Fig. 66. Explanation of square hysteresis loop.

The operation of a square-loop core is explained in Fig. 66. Assume the core is originally at negative remanence. At this point the tiny magnets are aligned, but not perfectly, with the south poles facing upward, let us say. If to the input winding is applied a pulse of current of such amplitude as to produce a magnetizing force smaller than the positive coercive force (point *F*), the magnets are rotated but not quite enough. When the pulse is removed, the magnets fall back to their original position. But if the pulse of current is of such magnitude as to produce a magnetizing force greater

than the positive coercive force, then the magnets rotate past their horizontal orientations and fly toward positive saturation—with the north poles facing upward. This operation is analogous to that of a swinging door: if the door is pushed past the center point, it completes the swing even if the force is removed. The flipping pulse must be applied for a long enough time to supply the energy necessary to reorient each magnet. When the pulse is removed, the magnets become slightly disoriented—they fall to positive remanence. Flipping from the positive side to the negative side is similar except that it is accomplished with an opposite polarity pulse.

Doughnut-Shaped Magnetic Cores

Most magnetic cores used in computers are doughnut-shaped. The advantage of the doughnut shape, or the toroid, is that very few lines of flux are lost; a complete circular path is provided for the lines of flux.

Doughnut-shaped magnetic cores with square hysteresis loops upon which are wound input and output windings make excellent storage cells. The positive remanence point may be called the ONE state and the negative remanence point the ZERO state. A ONE may be written into the core by applying a pulse of current in such a direction as to bring the core to positive saturation. When the pulse is removed, the core falls to positive remanence and remains there indefinitely. To write a ZERO into the core a pulse of current is applied to the input winding in the opposite direction, thus bringing the core to negative saturation. When the pulse is removed, the core falls to negative remanence and remains there indefinitely.

To avoid ambiguity, a dot is placed near one end of the input winding to indicate whether a ONE or ZERO is written according to the following rule (see Fig. 67*a* and *b*).

> Current (i_0) fed to the dot side flips the core to the ZERO state; current (i_1) fed to the nondot side flips the core to the ONE state.

A dot is also placed near one end of the output winding to indicate whether a plus or minus output voltage is induced across it according to the following rule:

> When flipping the core to the ZERO state the dot end of the output winding becomes positive; when flipping the core to the ONE state the nondot end of the output winding becomes positive.

Reading is accomplished as follows. A pulse which tends to flip the core to the ZERO state is applied to the input winding. If the core was storing a ZERO, the pulse causes the core to move along the negative saturation line. Since there is only a slight change in the lines of force,

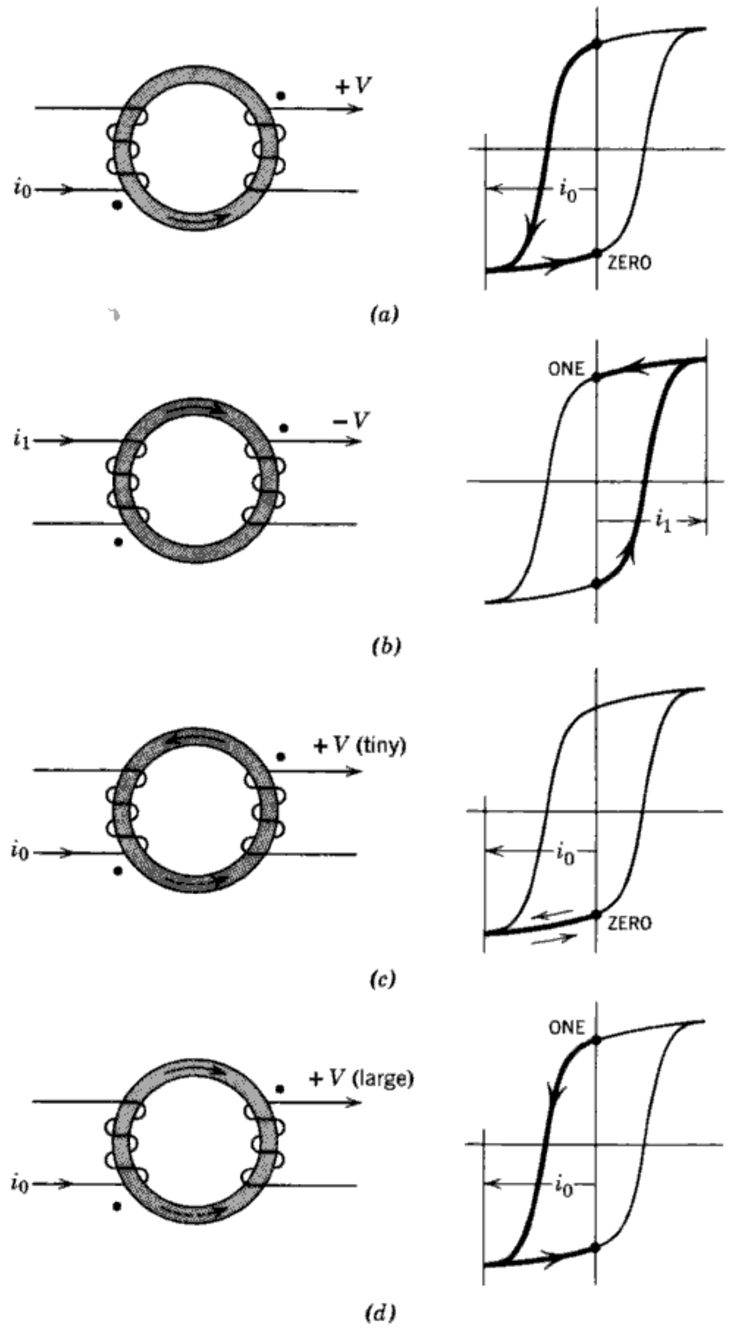


Fig. 67. Doughnut-shaped magnetic core as storage element. (*a*) Writing ZERO. (*b*) Writing ONE. (*c*) Reading ZERO. (*d*) Reading ONE.

only a tiny voltage is induced across the secondary (Fig. 67*c*). If the core was storing a ONE, this pulse flips the core to the ZERO state. The change in lines of force induces a large voltage across the output winding (Fig. 67*d*). Therefore, no output voltage (or a tiny voltage) indicates the core was storing a ZERO, and an output voltage indicates the core was storing a ONE. Note that, after reading, the core is in the ZERO state in both cases.

Although the doughnut-shaped square-loop core is primarily a storage cell, it may be used as a switch. It may be a switch from the voltage standpoint or from the impedance standpoint.

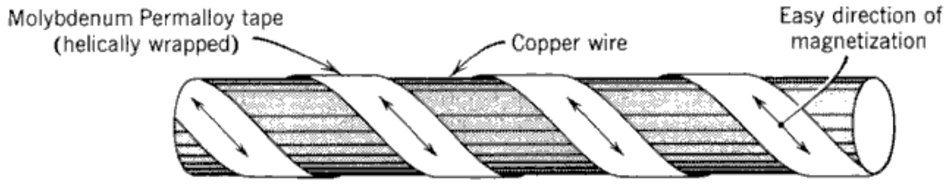
The core may be viewed as a voltage-switching device if voltage signals applied to the input winding or windings control the voltage signals at the output winding or windings. The core may be considered to be a closed switch when a high voltage appears across the output winding; this occurs when the core reverses states. The core is an open switch when no (or a tiny) voltage appears across the output; this occurs when the core does not change states.

The core may be viewed as an impedance-switching device if signals applied to input windings control the impedance of a circuit connected to the output winding. The core is a closed switch if a pulse applied to the output winding sees a short circuit; this occurs when the core is previously set so that the applied pulse does not flip the core. Since little magnetic energy is consumed by the core, the pulse is used almost entirely to cause current to flow in the secondary. The core is an open switch if a pulse applied to the output winding sees a very high impedance; this occurs when the core is previously set so that the applied pulse does flip the core. Since almost all the energy of the pulse is expended in reversing the state of the core, very little is left to produce current in the secondary.

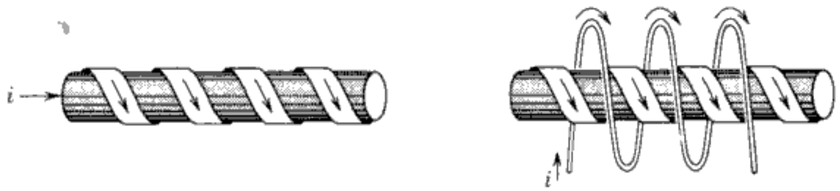
The doughnut-shaped square-loop core may also be used as an amplifier. A pulse applied to the input winding determines the state of the core. A pulse applied to the output winding causes current to flow in the load in accordance with what was stored in the core. The amount of energy flowing into the load is determined by the power applied by the pulse. This secondary pulse plays the same part in the magnetic amplifier that $B+$ plays in the vacuum-tube amplifier.

The Twistor

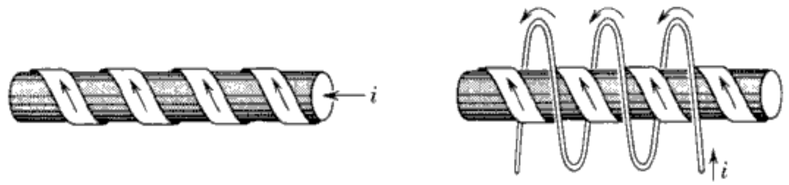
A device which behaves like a magnetic core but which possesses an entirely different physical form is the twistor. A twistor element consists of a copper wire upon which is helically wrapped Molybdenum Permalloy tape, as shown in Fig. 68*a*. Because of the way this magnetic tape is manufactured, it is easier to magnetize it along its length than in any other direction. This fact forms the basis for the twistor effect.



(a) Twistor element



(b) Current into left of center conductor } produces same magnetic field as } Current into left of encircling solenoid



(c) Current into right of center conductor } produces same magnetic field as } Current into right of encircling solenoid

Fig. 68. The twistor effect.

Current i flowing through the center conductor produces a magnetic field which magnetizes the wrapped tape. If the current enters the conductor from the left (Fig. 68*b*), the magnetic field induced in the tape is as shown. This can be easily seen by application of the right-hand rule which states that, if the wire is held by the right hand so that the thumb points in the direction of current flow, the fingers point in the direction of the resultant magnetic field; this field induces a magnetic field in the tape in the direction of easy magnetization. If the current enters the conductor from the right, the direction of magnetization of the wrapped tape is reversed, as shown in Fig. 68*c*.

The same results may be achieved by sending current through a solenoid surrounding the twistor element. If current enters the left side of the solenoid (Fig. 68*b*), a magnetic field is produced which points from left to right along the axis of the conductor. The wrapped tape is magnetized along the path of easy magnetization, as shown. If current enters the right side of the solenoid Fig. 68*c*, a magnetic field is produced which

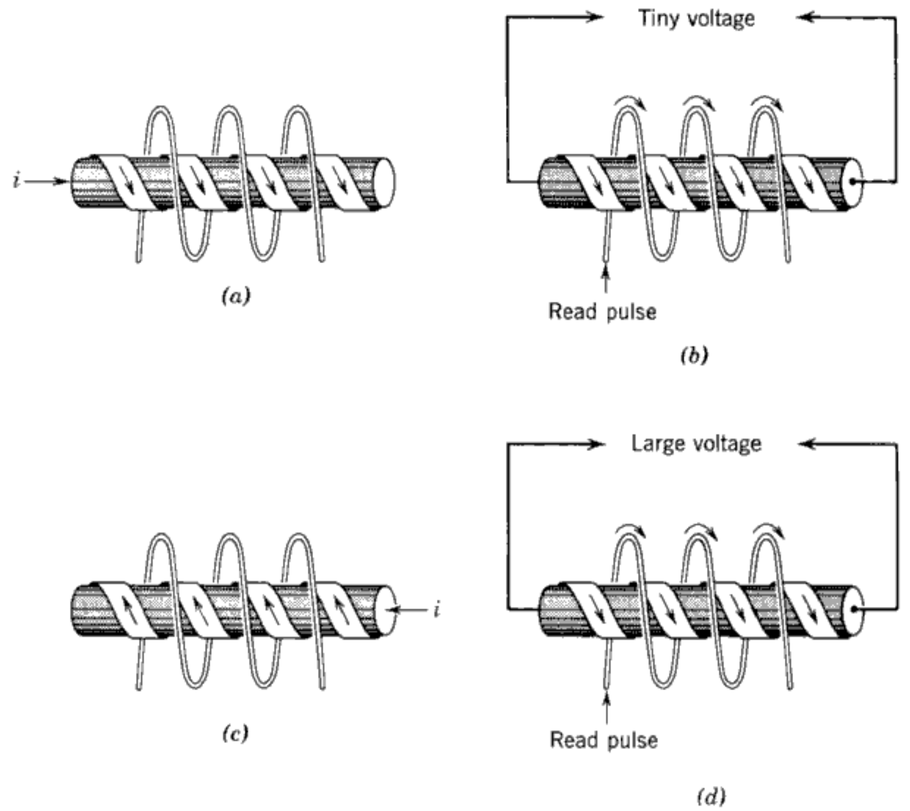


Fig. 69. Twistor as storage element. (*a*) Writing ZERO. (*b*) Reading ZERO. (*c*) Writing ONE. (*d*) Reading ONE.

points from right to left along the axis of the conductor. In this case, the wrapped tape is magnetized in the opposite direction.

Figure 69 shows the operation of the twistor as a storage element. By applying current i to the conductor (or the solenoid) in one direction, we write a ZERO. By applying current to the conductor (or the solenoid) in the opposite direction, we flip the magnetic state of the wrapped tape and thus write a ONE. Reading is accomplished by applying a ZERO-flipping read pulse current to the solenoid. If the wrapped tape was storing a ZERO, little change in magnetic field occurs and no (or little) voltage is induced across the center conductor. If the wrapped tape was storing a ONE, the reversal of the field induces a large voltage across the center conductor.

Marginal Checking

With magnetic cores, marginal checking is accomplished by varying the width of pulses applied to the cores. For any one type of core the width

of the needed pulse is specified within certain limits. If the width is reduced to its lower limit and a core refuses to function, this core is replaced.

Marginal checking is not needed as much with magnetic components as it is with vacuum-tube components. This is because magnetic cores are inherently much more reliable and rugged than vacuum tubes.

Other Magnetic Components

A square-loop magnetic mixture may be deposited on an acetate or Mylar tape to form magnetic tape. Thousands of bits of information may be stored on this magnetic tape, each bit being located in a small square on the tape. In effect, the tape is divided into many storage cells.

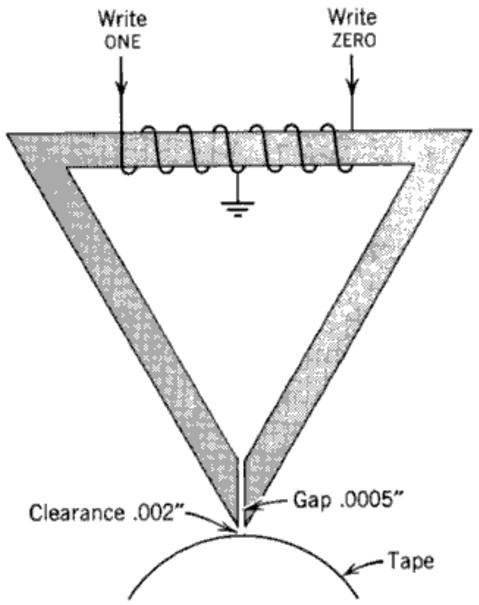


Fig. 70. Magnetic write head.

Writing is accomplished by means of a magnetic head constructed as shown in Fig. 70. The triangular material has an air gap of about .0005 in. When current is applied to the winding, a magnetic field flows through the head and across this tiny air gap. If the magnetic tape is close to the head—about .002 in. distant—the lines of flux affect the magnetic state of the tape near the point of the head. Sending current through the winding in one direction causes the simulated storage cell on the tape to saturate positively—store a ONE. Sending current through the winding in the opposite direction causes the simulated storage cell to saturate negatively —store a ZERO. By moving the tape and feeding successive information pulses to the winding, bits may be stored in successive locations on the

tape. Figure 71 shows 4327928 recorded on the tape in binary-coded decimal plus parity-check bit for each digit.

Reading is accomplished by means of a magnetic read head similar to the write head. As the magnetic tape with its stored bits is moved under the head, the magnetic field of each area induces a magnetic field in the head which induces a voltage across the output winding. The direction of the induced voltage depends on the direction of magnetization of the effective storage cell on the tape.

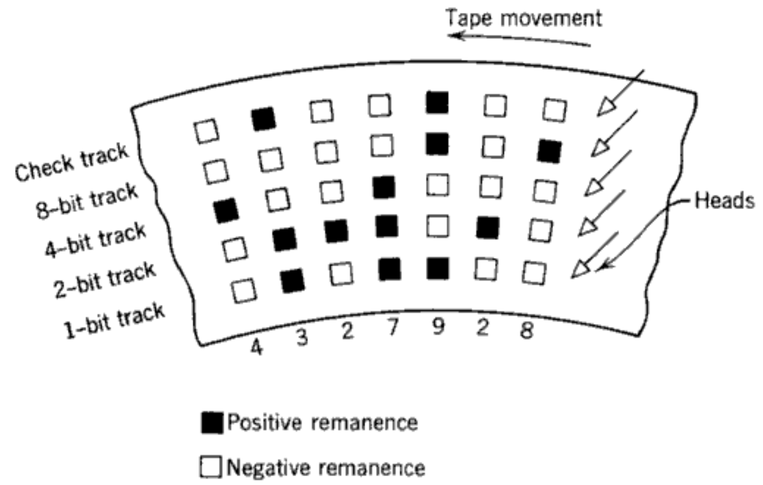


Fig. 71. Tape recording.

Several heads may be used to write and read information to and from several tracks along a strip of magnetic tape. A wider storage surface can be achieved by coating a drum with the square-loop magnetic material. With this arrangement many more tracks can be written into or read from simultaneously. Another variant of this technique is to coat a disk with the magnetic material, and write onto it and read from it juke-box fashion. Many other variations are possible.

SWITCHING CIRCUITS

For encoders, decoders, and other circuits which select certain lines under certain circumstances, the transformer action of magnetic-core components may be used. The magnetic core may have a nonsquare loop or a square loop. The wired-core encoder is an example of a switching circuit using nonsquare loop cores. The magnetic-core coincident-current selector is an example of a switching circuit using square-loop cores.

For the bulk of the logical functions which must be performed by a computer, transformer action as well as storage action is used. In these circuits the switching functions are performed as the bits are shifted from core to core. In essence the cores, each of which delays the bit a little, represent a modified shift register. Logical functions easily performed by these modified shift registers are OR and INHIBIT. With only these two circuits as building blocks, all other logical functions may be produced.

The wired-core encoder, the magnetic-core coincident-current selector, and the OR and INHIBIT circuits are presented below.

Wired-Core Encoders

Figure 72 shows a wired-core encoder which produces a different 3-bit binary number for each line chosen. Eight wires are threaded through

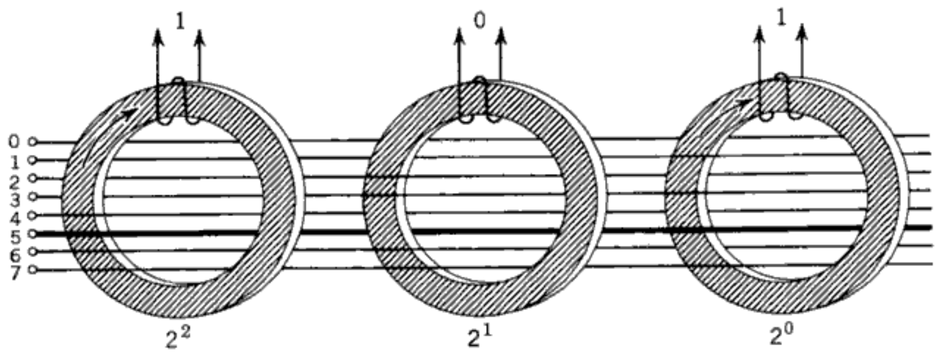


Fig. 72. Wired-core encoder.

three nonsquare loop cores, each wire threaded differently. Line 0 does not go through any core. Line 1 threads only the 2^0 core, line 2 threads only the 2^1 core, and line 7 threads all cores. When a pulse of current is applied to one of these eight input lines, a magnetic field is produced in the cores that are threaded by the chosen wire. The changing magnetic field produces an output voltage in the secondaries of these cores. No voltages are produced at secondaries of cores not threaded by the chosen line. The resultant output is a different binary number for each line selected.

The illustration shows what happens when a pulse of current is applied to line 5. Cores 2^0 and 2^2 are threaded and, therefore, become magnetized. An output pulse is produced in positions 2^2 and 2^0, thus giving an output of 101, or the binary combination for 5.

Magnetic-Core Coincident-Current Selector

The magnetic-core coincident-current selector is a device which activates one out of several output lines when signals are applied to two input lines

simultaneously. The selector consists of a group of square-loop cores arranged in an array of vertical columns and horizontal rows. A vertical input line threads all the cores in a column; and a horizontal input line threads all the cores in a row. A separate output winding is wound around each core in the array. Current pulses, each of which is only half the value needed to flip a core, are applied to one vertical line and one horizontal line. Only the one core in the array through which both activated lines pass receives enough magnetic energy to reverse states; a pulse is produced across its output winging.

Figure 73 shows a 3-column by 3-row coincident-current selector. The horizontal input lines are labeled X_0, X_1, and X_2. The vertical input lines are labeled Y_0, Y_1, and Y_2. Assume all cores in the array are originally at negative remanence. If now a half-current pulse—a pulse half the amplitude needed to bring a core to positive saturation—is applied to X_2, the cores in this row reach point A on the hysteresis loop (Fig. 73*b*). Likewise, a half-current pulse applied to Y_1 causes the cores in this column to reach the same point on the loop. Only the cross-hatched core receives enough magnetic energy to flip completely to positive saturation. When the half-current pulses are removed, the cross hatched core falls to the positive remanence point; all other cores fall to the negative remanence point.

Before being used again, the cores in the selector must be reset to negative remanence.

Note that, if the strength of the half-current pulse is increased slightly, the core's coercive force may be exceeded, and the core may flip completely with only one half-current pulse applied. For greater reliability of operation it is best to improve the distinction between the effects of single half-current pulses and double half-current pulses. This distinction may be improved by applying a d-c bias to all cores in the coincident-current selector. Core operation with d-c bias applied is indicated in Fig. 73*c*. The direct current places each core at negative saturation. One half-current pulse brings the core to the origin; two half-current pulses bring the core to positive saturation.

OR and INHIBIT Circuits

As was stated above, logical functions may be developed as bits are shifted from core to core in what may be considered to be a modified shift register. To understand this shifting action better, refer to Fig. 74*a*. A pulse applied to the input of core 1 sets it to the ONE state. A *CP* fed to the lower winding flips core 1 back to the ZERO state, and during flipping a voltage is developed across the output winding. This output voltage sends current through the input winding of core 2 and sets *it* to the ONE state. In other words, the ONE in core 1 is shifted to core 2. The

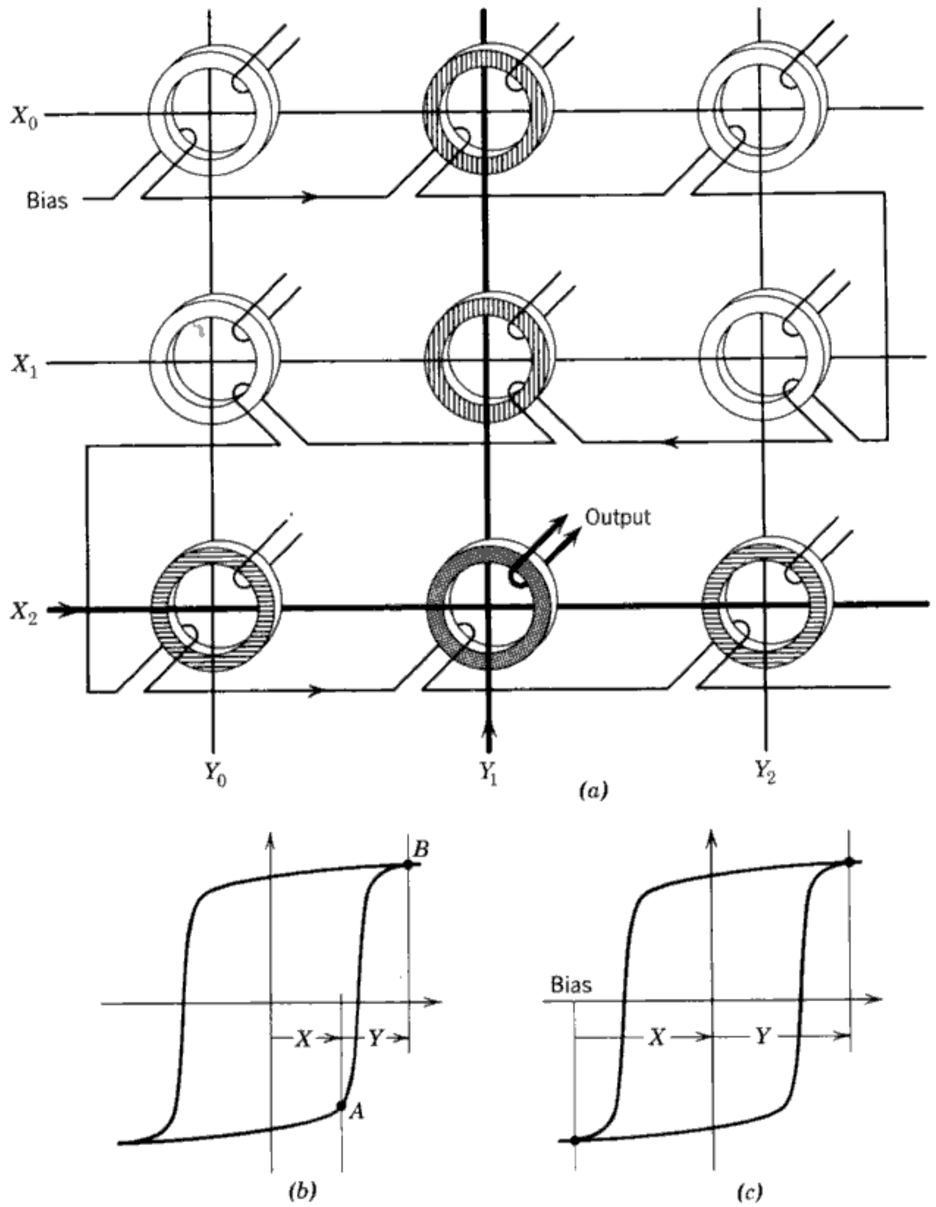


Fig. 73. Magnetic-matrix selector. (*a*) Selector. (*b*) Loop without bias. (*c*) Loop with bias.

next *CP* flips core 2 to the ZERO state, and the resultant output voltage sends current to the input winding of core 3. Of course, if a core is storing ZERO, the *CP* has no effect. The diode between cores prevents induced voltages from sending current backward—from core 2 to core 1, for instance.

This circuit may be satisfactory for shifting one bit from core to core.

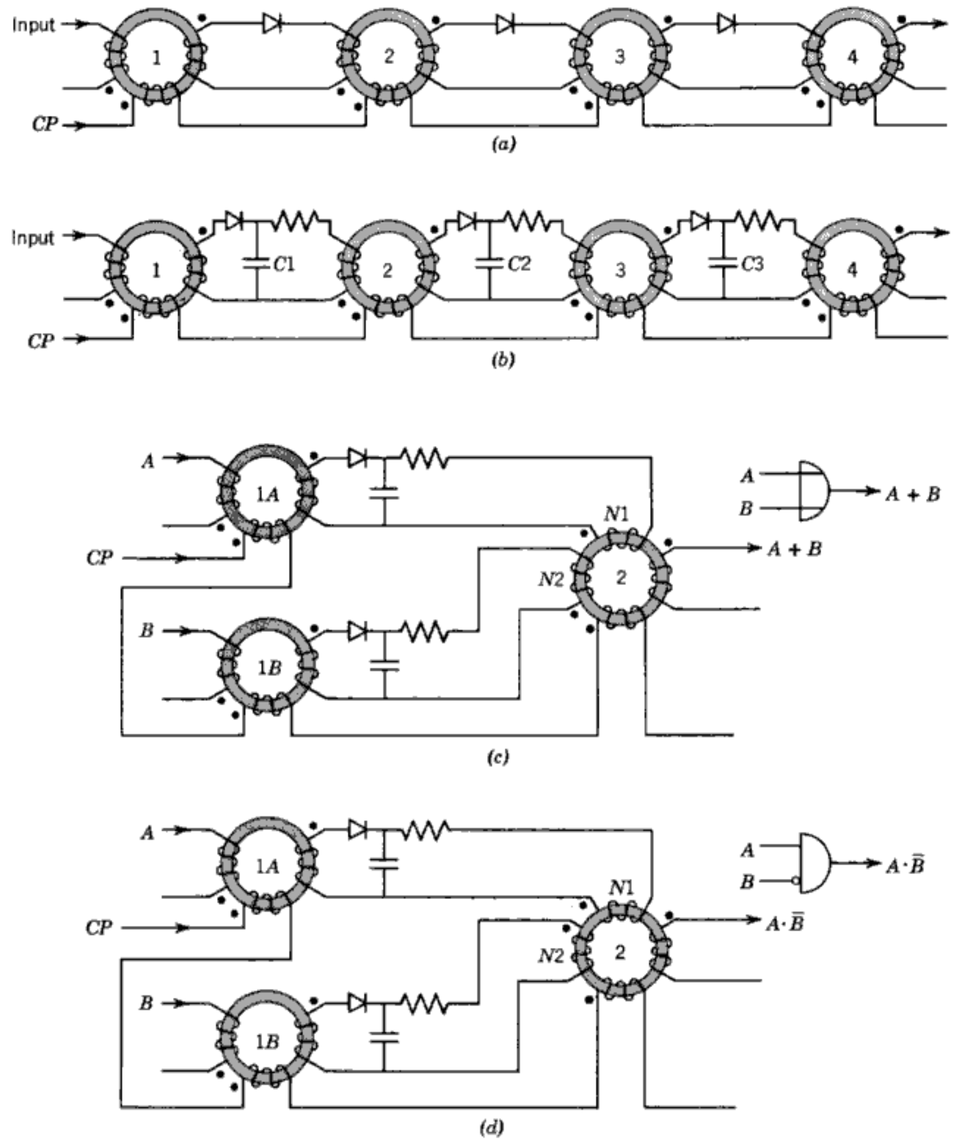


Fig. 74. Magnetic-core switching circuits. (*a*) Shifting bits. (*b*) Temporary storage between storage cells. (*c*) OR circuit. (*d*) INHIBIT circuit.

But usually, as one bit is being shifted from core 2 to core 3, another bit must be shifted from core 1 to core 2. To make sure that core 2 is completely flipped before the contents of core 1 are applied to core 2, a delay circuit is placed between cores, as shown in Fig. 74*b*. In this configuration when the *CP* is applied, if core 1 is storing ONE, it flips and an output pulse flows through the diode to charge the capacitor *C*1. After a delay determined by the charging of capacitor *C*1, the capacitor discharges its current into the input winding of core 2. While a ONE in core 1 is charging

$C1$, a ONE in core 2 is charging $C2$. While $C1$ is discharging into core 2, $C2$ is discharging into core 3. Thus the delay between storage cells prevents interaction between storage cells.

A simple OR circuit is shown in Fig. 74*c*. A signal A is applied to the input winding of core $1A$, and signal B is applied to the input winding of core $1B$. The output winding of each of these cores is connected to one of the two input windings ($N1$ and $N2$) on core 2. The CP's are applied in series to a winding on each core.

The circuit operates as follows. A pulse of current applied at A flips core $1A$ to the ONE state. The next CP flips core $1A$ back to the ZERO state, and induces a positive output voltage. This output voltage enters the nondot side of $N1$ to flip core 2 to the ONE state. The next CP flips core 2 to the ZERO state and produces an output voltage. If a current pulse is applied to B instead of to A, the first CP induces an output voltage which causes current to flow to the nondot side of $N2$ to flip core 2 to the ONE state; the following CP produces an output voltage as before. Since a pulse applied at either A or B (or both) produces a pulse at the output, this is an OR circuit.

The INHIBIT function is produced by the circuit shown in Fig. 74*d*. On the surface, the OR and INHIBIT circuits look alike. But there is this important difference between them: in the OR circuit winding $N2$ is wound in the same relative direction as $N1$, whereas in the INHIBIT circuit $N2$ is wound in the opposite direction from $N1$. In the INHIBIT circuit a pulse at A sets $1A$ to the ONE state, and the following CP flips core 2 to the ONE state. The next CP flips core 2 to ZERO and produces an output pulse. If at the same time a pulse is applied at A a similar pulse is applied at B, core $1B$ is set to the ONE state. While the first CP is shifting the ONE from core $1A$ to $N1$, it is also shifting the ONE from core $1B$ to $N2$. Since $N1$ and $N2$ are wound in opposite directions, their magnetic fields cancel each other. The result is that core 2 does *not* flip to the ONE state and the following CP does *not* produce an output pulse. In other words, an output pulse is produced when A is present unless inhibited by a B pulse. The circuit produces $A \cdot \bar{B}$.

Other logical functions may be developed from these two logical blocks. The NOT function may be developed by making the A applied to an INHIBIT circuit always equal to 1:

$$A \cdot \bar{B} = 1 \cdot \bar{B} = \bar{B}$$

If this $\bar{B}$ is applied to another INHIBIT circuit, an AND function is produced:

$$A \cdot \bar{\bar{B}} = A \cdot B$$

A very important fact to remember about these switching circuits is that

the output occurs later in time than the inputs. In other words, there is a delay inherent to the magnetic-core switching circuit.

AMPLIFYING CIRCUITS

The square-loop toroidal core may be used for amplification, as shown in Fig. 75. Essentially the magnetic amplifier consists of an input winding

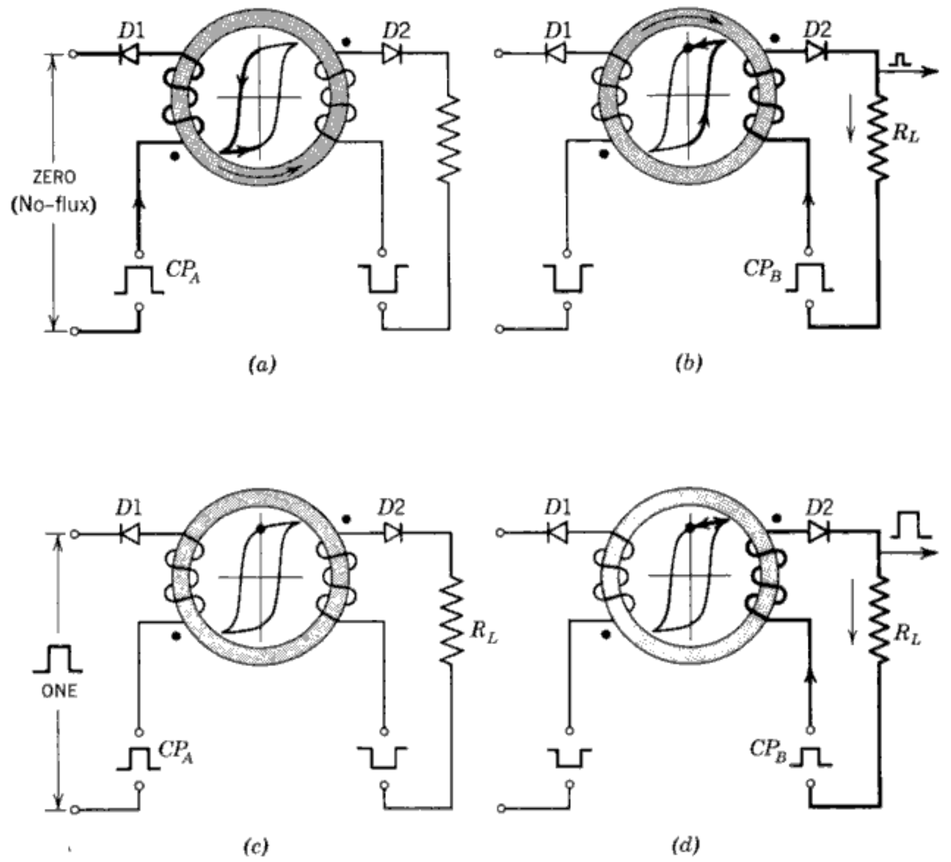


Fig. 75. Magnetic amplifier. (*a*) ZERO flips core to ZERO state. (*b*) CP_B flips core to ONE state. (*c*) ONE keeps core in ONE state. (*d*) CP_B keeps core in ONE state.

to which are fed a clock pulse A (CP_A) and an input pulse; and an output winding to which is applied a clock pulse B (CP_B) across which the amplified output is produced.

Information is fed to the amplifier during CP_A—when A is positive. Information is removed from the core during CP_B—when B is positive.

When CP_A is positive, CP_B is negative, and when CP_A is negative, CP_B is positive.

If during the application of CP_A a ZERO in the form of no-pulse is applied to the input winding, diode $D1$ allows current to enter the input winding through the dot side to bring the core to the ZERO state (Fig. 75*a*). Upon completion of CP_A, CP_B is applied to the output winding. Current enters the output winding through the nondot side to flip the core to the ONE state. Since most of the energy is consumed flipping the core, very little current flows to the load, and only a tiny output voltage is produced (Fig. 75*b*).

Upon completion of CP_B, CP_A is applied to the input core. If now a ONE in the form of a positive pulse is applied to the input winding, $D1$ does not conduct and the state of the core cannot be changed. It remains in the ONE state. The next CP_B sends current through the nondot side of the output winding to send the core from positive remanence to positive saturation. Since this small change in flux requires little energy, almost all the current flows through the load to produce a large voltage drop.

This circuit is an amplifier because the amplitude of the output pulse may be greater than that of the input pulse. The output amplitude is determined mainly by the size of the clock pulse and by the load. The function of the clock pulse in the magnetic amplifier is analogous to the function of $B+$ in vacuum-tube amplifiers.

STORAGE CIRCUITS

The bistable magnetic core can be used for amplification and for switching. We have seen that during switching the core acts as a delay element too. This delay is determined by the time between clock pulses. If clock pulses are withheld, the core may store information for an indefinite period of time. This great versatility of the magnetic core makes it an ideal component for a variety of circuits which are essentially storage devices: the storage array which is a permanent storage device; the magnetic shift register and counter which are used for temporary storage; and the dynamic storage cell and dynamic flip-flop. The magnetic drum, tape, and disk are associated devices which are used for bulk storage.

Magnetic-Core Storage Arrays

Square-loop magnetic cores may be placed in a rectangular storage array (Fig. 76) similar to the array used for selection. As in the selector,

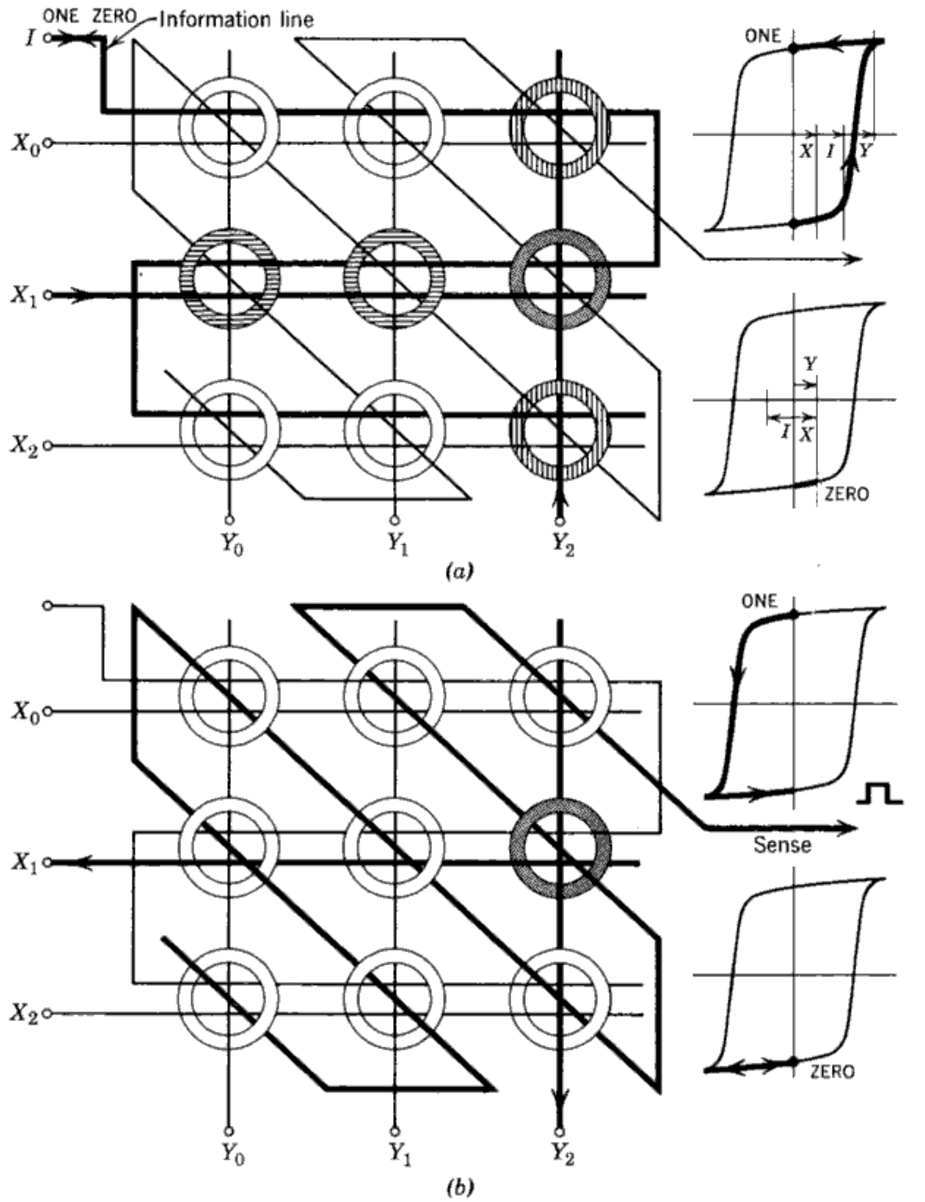


Fig. 76. Coincident-current storage plane. (*a*) Writing. (*b*) Reading.

vertical lines thread all the cores of a column and horizontal lines thread all the cores of a row. In the selector there is a separate output winding for each core; in the storage circuit all the output (sense) windings in the plane are connected in series. In the selector a bias winding inhibits selection unless two half-current pulses are present; in the storage array an information winding adds or subtracts magnetic flux as needed to store ONE or ZERO.

Selection of a core during writing is accomplished by applying coincident pulses to a horizontal line (X) and to a vertical line (Y) in a manner similar to that used in the selector. At the same time that these two pulses are applied, a pulse is applied to the information line I, in one direction for a ONE, in the opposite direction for a ZERO. If a ONE pulse is applied at the chosen core, the I, X, and Y fields add to bring the core to the ONE state. If a ZERO pulse is applied at the chosen core, the I and X pulses cancel each other and not enough flux is produced to flip the core; it stays in the ZERO state. All other cores definitely do not receive energy to flip.

Reading is accomplished by applying a half-current pulse to a vertical and to a horizontal line in such a direction as to cause the chosen core to flip to the ZERO state. If the chosen core was storing a ONE, the core flips to ZERO and, during this reversal of flux, produces a voltage across the sense winding. If the chosen core was storing a ZERO, there is no flux reversal and very little voltage is produced across the sense winding.

The above scheme is called coincident-current storage, and the plane of nine magnetic cores is called a 3-by-3 (three rows by three columns) coincident-current storage plane. This scheme of storage may be expanded to include several planes, coincidence occurring simultaneously at similar points in each plane. In Fig. 77 is shown how three 3-by-3 coincident-current storage planes are connected to form a storage scheme capable of storing combinations of bits. Three current drivers X_0, X_1, and X_2 feed the rows of all three planes; and three column drivers Y_0, Y_1, and Y_2 feed the columns of all three planes. A separate information driver sends either a ONE or a ZERO pulse into the information winding of each plane; and a separate sense amplifier picks up the signal produced during reading across the sense winding of each plane.

The illustration shows what happens when the binary combination 101 is written into a group of cores. The cores are chosen by activating drivers X_1 and Y_2. The drivers send current flowing through the row and column lines shown in bold in each plane, selecting core A in plane 0, core B in plane 1, and core C in plane 2. At the same time, information drivers 0 and 2 are caused to send ONE-producing current through the cores in plane 0 and 2 respectively. The X_1, Y_2, and information pulses add to flip the cores A and C to the ONE state. Information driver 1, on the other hand, sends ZERO-producing current through the cores of plane 1. This current cancels the effect of the X_1 current in core B, thus leaving only Y_2 to produce flux. This flux is not enough to flip core B. Upon completion of writing, therefore, cores A, B, and C are storing the combination 101.

To read this combination, half-current pulses are produced by X_1 and

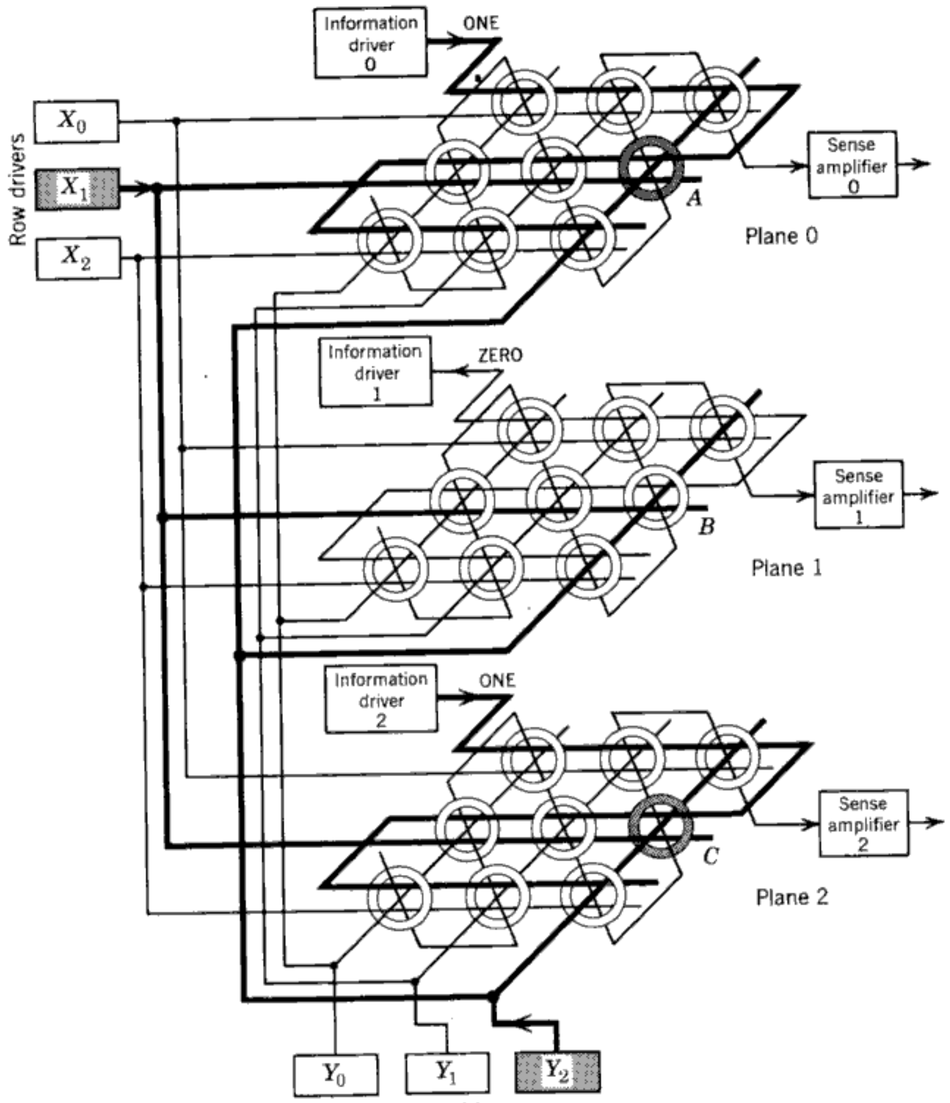


Fig. 77. Three planes of coincident-current storage.

Y_2, but in the opposite direction. The two pulses in coincidence are strong enough to flip a core to the ZERO state. Cores A and C flip from the ONE state to the ZERO state; the reversals in flux produce output voltages in the sense windings of planes 0 and 2. The respective sense amplifiers amplify these output pulses. Core B does not flip since it is already in the ZERO state; no voltage is produced across the sense winding of plane 1, and nothing is amplified by sense amplifier 1.

Many variations of the core storage array have been tried. One such variation is the ferrite apertured plate. This device is composed of a sheet

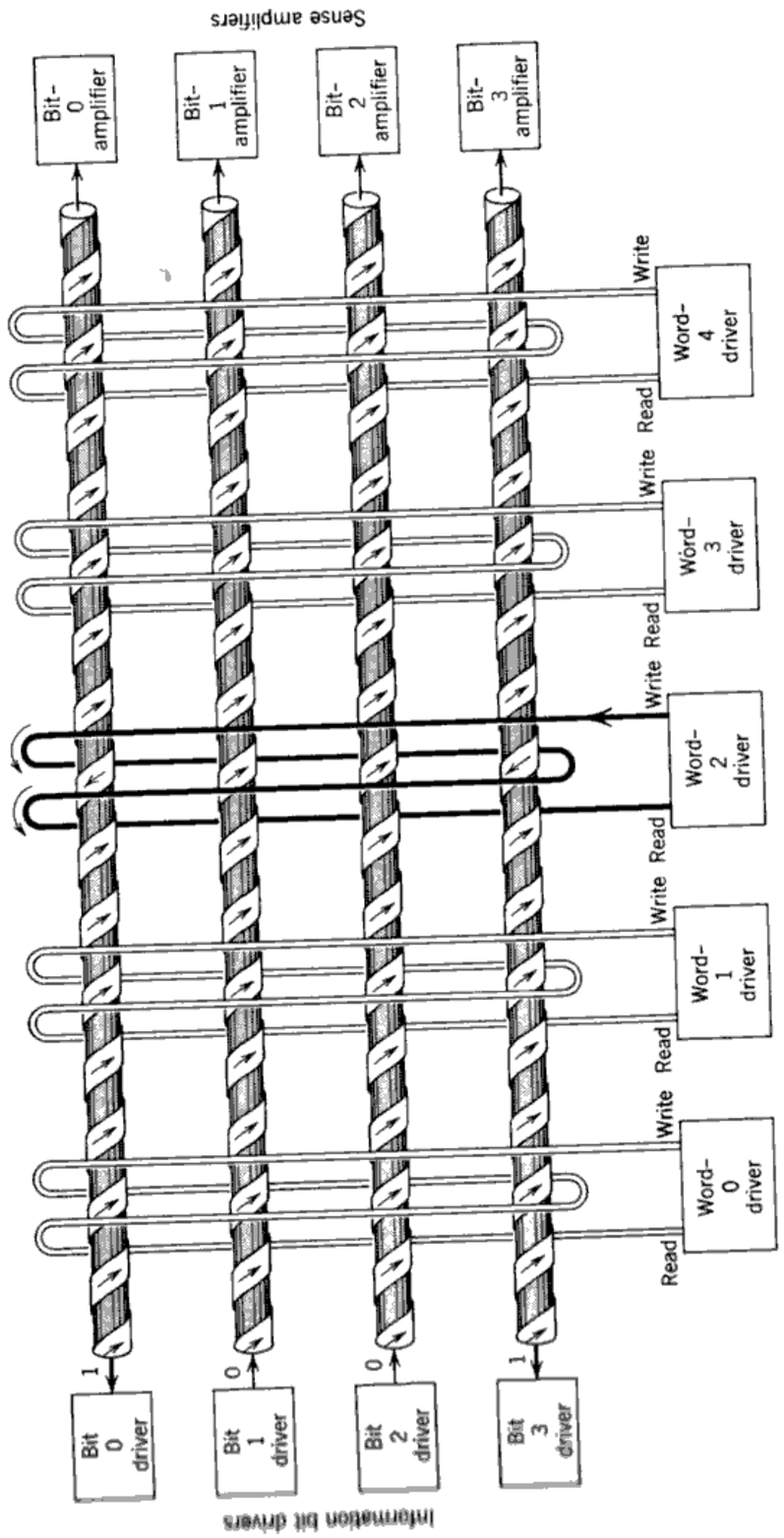


Fig. 78. Twistor storage array.

of ferrite drilled with holes. The small area around each hole behaves like a toroid. Each toroid is activated by windings which are applied by printed-circuit techniques. The big advantage of the ferrite apertured plate is ease of fabrication.

Research has also been done on thin films of magnetic material deposited on glass by printed-circuit techniques.

Twistor Storage Arrays

Coincident magnetic-core storage may be obtained in a slightly different way: by a word-selection technique. It is called word selection because in this scheme only one pulse is needed to read a complete word. In the following word-selection storage is described as it applies to twistor elements. The reader can easily visualize how this technique can be applied to magnetic cores.

A twistor storage array capable of storing five 4-bit words is shown in Fig. 78. It consists of four twistor elements and five solenoids, each solenoid surrounding all four twistor elements at a certain location. Each spot in the twistor element that is enclosed by a solenoid may be considered to be a storage element similar to a toroid. Feeding each twistor element is an information bit driver which sends current through the center conductor in one direction for writing a ZERO and in the opposite direction for writing a ONE. Feeding each solenoid is a read-write driver which sends current through the solenoid in one direction for writing and in the oppositie direction for reading. At the output end of each center conductor is a sense amplifier which amplifies any voltage induced in the conductor when a read pulse is applied to a solenoid.

Writing is accomplished by a coincidence of half-currents through a chosen solenoid and through a center conductor. If the bit current flowing through a center conductor is in such a direction as to produce a magnetic field in the wrapped tape in the same direction as that produced by the solenoid write current, the wrapped tape in this location flips to the ONE state. If the bit current is in such a direction as to produce a magnetic field in the wrapped tape opposing the field produced by the solenoid write current, the wrapped tape in this location remains in the ZERO state. The illustration shows 1001 being written into location 2. In the uppermost and lowermost twistor elements the fields unite to write a ONE. In the middle two twistor elements the fields cancel each other to leave a ZERO.

No coincidence is needed for reading. A read current pulse large enough to flip the state of the wrapped tape is applied by a read-write driver to a chosen solenoid. This pulse flips the magnetic state of the wrapped tape to ZERO at each of the four spots enclosed by the solenoid.

If a spot was storing ONE, the flipping produces an output voltage which is amplified by the associated sense amplifier. If a spot was storing a ZERO, no flipping takes place and nothing is picked up by the sense amplifier. In the example given, a read pulse applied by word-2 driver would flip the state of the wrapped tape in the uppermost and lowermost twistor elements. The amplifiers would thus read 1001, or what was stored in the given location.

Registers and Counters

In Fig. 74 we have seen how a bit may be shifted from one core to the next by means of clock pulses; and we have also seen how a delay between storage cores increases the reliability of bit shifting. A circuit such as that shown in Fig. 74*b* may be used as a shift register. Instead of introducing a delay circuit between cores, the same delaying effect may be achieved by using two out-of-phase clock pulses in an arrangement using two cores for each stored bit, as shown in Fig. 79*a*. CP_A is applied to one core of each pair, and CP_B is applied to the other core. A bit is fed to the input winding of core $1A$. CP_A shifts this bit to core $1B$. Then CP_B shifts the bit from core $1B$ to $2A$. At the same time the next bit is applied to core $1A$. The following CP_A shifts the bits in cores $1A$ and $2A$ to cores $1B$ and $2B$. Then CP_B shifts the information from core $1B$ to $2A$ and from cores $2B$ to $3A$, and so on. The clock pulses are applied at a steady rate which determines the delay through each core. Figure 79*b* shows the same shift register in block diagram form.

Figure 79*c* shows a 4-bit magnetic recirculating shift register with five gates: read-in, read-out, recirculating, shift right, and shift left. A 4-bit number may be fed to the register by applying a read-in signal to the read-in gate for the length of time it takes four CP_A and four CP_B pulses to occur. As each succeeding bit is shifted into S_{3A}, bits further along in the register are shifted into other cores with A subscripts. After four CP_A and four CP_B pulses, the first bit is stored in S_{0B} and the last bit is stored in S_{3B}.

If now the read-in signal is removed and the recirculation signal is applied, succeeding clock pulses cause bits from S_{0B} to be shifted into $S_{3A'}$. After four more CP_A and CP_B, all the bits are back to the same relative positions they were in upon completion of read-in.

If now the right-shift signal is applied, the total number of delay elements in the loop is reduced from eight to six. After the advent of three CP_A and CP_B, the first bit appears in S_{0B}. If now the right-shift signal is removed, the following CP_A and CP_B annihilate the first bit, and the rest of the number appears in the three rightmost positions of the register. The leftmost position contains ZERO.

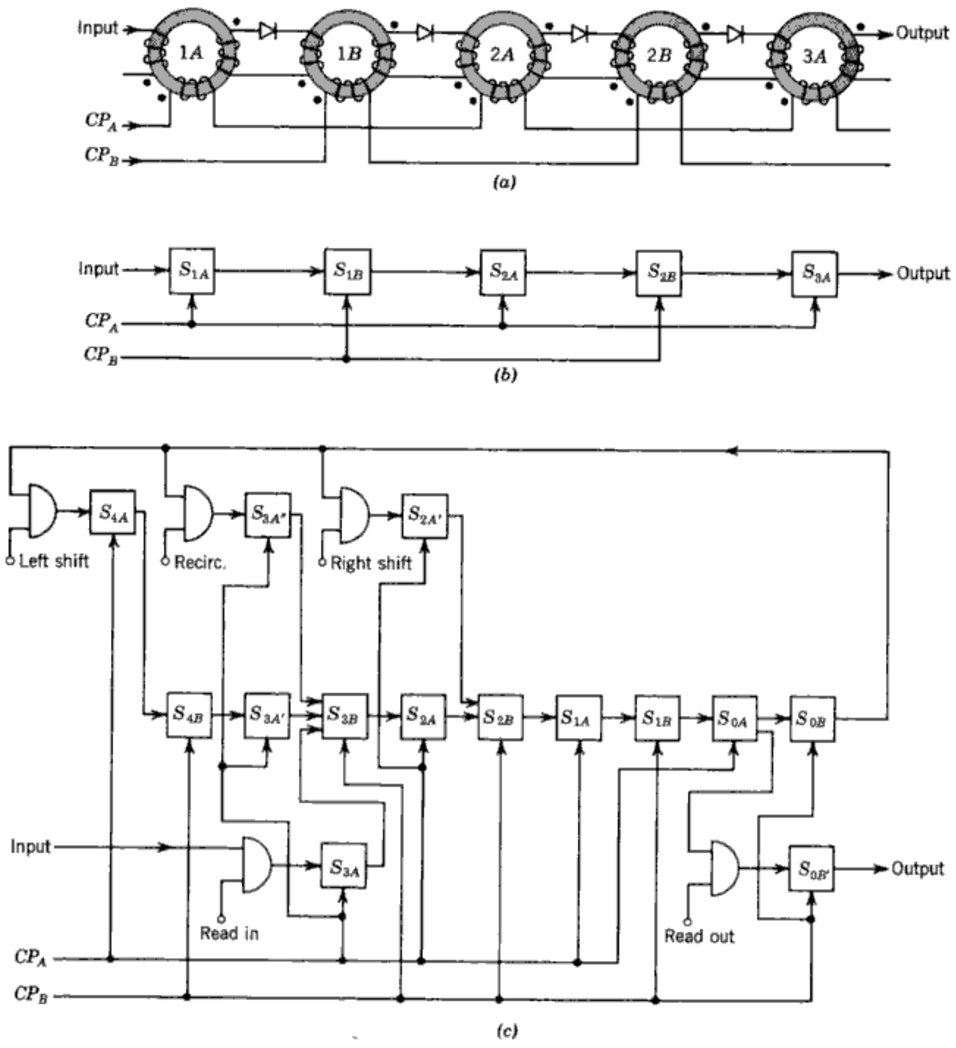


Fig. 79. Magnetic shift registers. (*a*) Shifting with two clock pulses. (*b*) Equivalent block diagram. (*c*) Recirculating shift resister.

Just as the removal of delay from the loop produces a right shift, the addition of delay produces a left shift. When the left-shift is applied and four CP_A and CP_B are applied, the bits wind up in S_{1B}, S_{2B}, S_{3B}, and S_{4B}. If now S_{4A} and S_{4B} are cut off from the register, only the three leftmost bits are being stored in the register—S_{3B}, S_{2B}, and S_{1B}. The rightmost position, S_{0B}, is storing ZERO

To feed information out of the register, the read-out signal is applied for the time it takes four CP_A and CP_B to pass.

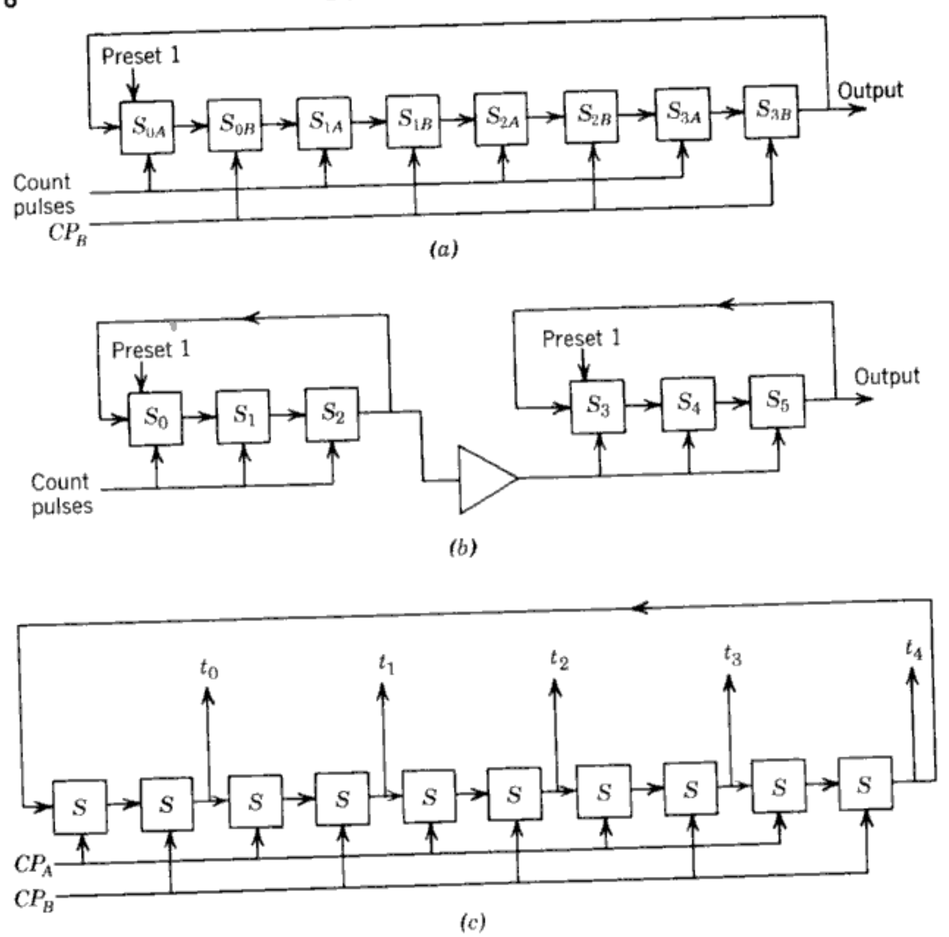


Fig. 80. Modulus counters. (*a*) Modulo-4 counter. (*b*) Modulo-9 counter composed of two modulo-3 counters. (*c*) Timing-pulse generator.

A modulus counter is essentially a recirculating shift register, as indicated in Fig. 80*a*. A preset pulse is placed in S_{0A} before counting begins. Instead of CP_A, half the cores receive the count pulse. The other cores receive CP_B as before. Each count pulse shifts the ONE to the following core; the next CP_B shifts the ONE to the next core in line.

The illustration shows a modulo-4 counter: after four count pulses are applied, a pulse is produced at the output. At the same time the ONE is shifted back to the first core in order that counting may be resumed.

Figure 80*b* illustrates how a modulo-3 counter can feed another modulo-3 counter to produce a modulo-9 counter. The first counter produces a pulse after three counts. When this pulse is applied to the second counter, the ONE in the second counter moves to the next position. After two more counts of three, the ONE in the second counter is shifted to the output. Thus an output is produced after three times three or nine counts.

A magnetic-core modulus counter may be modified to produce an output at each core—a ring counter. Each count pulse causes a pulse to be produced in the following core in the line. This ring counter may sometimes be used as a timing-pulse generator, such as shown in Fig. 80*c*. Here ten storage cells are connected in a recirculating loop to produce five timing pulses, t_0, t_1, t_2, t_3, and t_4. Note that the time between the occurrence of t_0 and the occurrence of the following t_0 is always five CP_A and CP_B. The same is true of the other timing pulses. If five CP_A and CP_B are considered to be a cycle, timing pulses always occur at the same point of the cycle. Timing pulses are used for control in digital computers.

Dynamic Storage Cells

A dynamic storage cell—one producing a train of pulses when storing ONE and no pulses when storing ZERO—may be built of three square-loop magnetic cores, as shown in Fig. 81*a*. The dynamic storage cell has an input core 1*A*, an output core 1*B*, and a reset core 2*A*. A positive pulse applied to the input core stores a ONE in it. CP_A shifts the ONE to the output core; CP_B shifts the ONE back to the input core. The ONE pulse is thus continuously shifted back and forth between cores 1*A* and 1*B* as long as clock pulses are applied. During each CP_B an output pulse is produced.

This situation continues as long as no signal is applied to reset core 2*A*. But if a pulse is applied to 2*A*, the next CP_A sends current through *N*2 to oppose the field produced by *N*1, thus preventing core 1*B* from flipping to the ONE state. As a result no output pulses are produced until a set pulse is once again fed to core 1*A*.

By combining two elementary dynamic storage cells as shown in Fig. 81*b*, a dynamic flip-flop is produced. The dynamic flip-flop has four cores: a set core (1*A*) and a ONE output core (1*B*); a reset core (2*A*) and a ZERO output core (2*B*). The output of the set core inhibits the ZERO output core; the output of the reset core inhibits the ONE output core.

Operation is as follows. A positive set pulse causes core 1*A* to store a ONE. CP_A shifts this ONE to the core 1*B*; CP_B shifts it back to the core 1*A*. As long as clock pulses are applied, the ONE bounces back and forth between cores 1*A* and 1*B*, and a ONE output pulse is produced during each CP_B.

Every time the ONE is shifted to core 1*B*, an inhibit pulse is applied to core 2*B* causing it to store ZERO. The result is that the ZERO output produces no pulses while the ONE output core produces a train of pulses.

If now a positive pulse is applied to the reset core 2*A*, it stores a ONE. CP_A shifts it into core 2*B*; CP_B shifts it back to core 2*A*. While this ONE is bouncing back and forth between cores 2*A* and 2*B*, core 1*B* is being

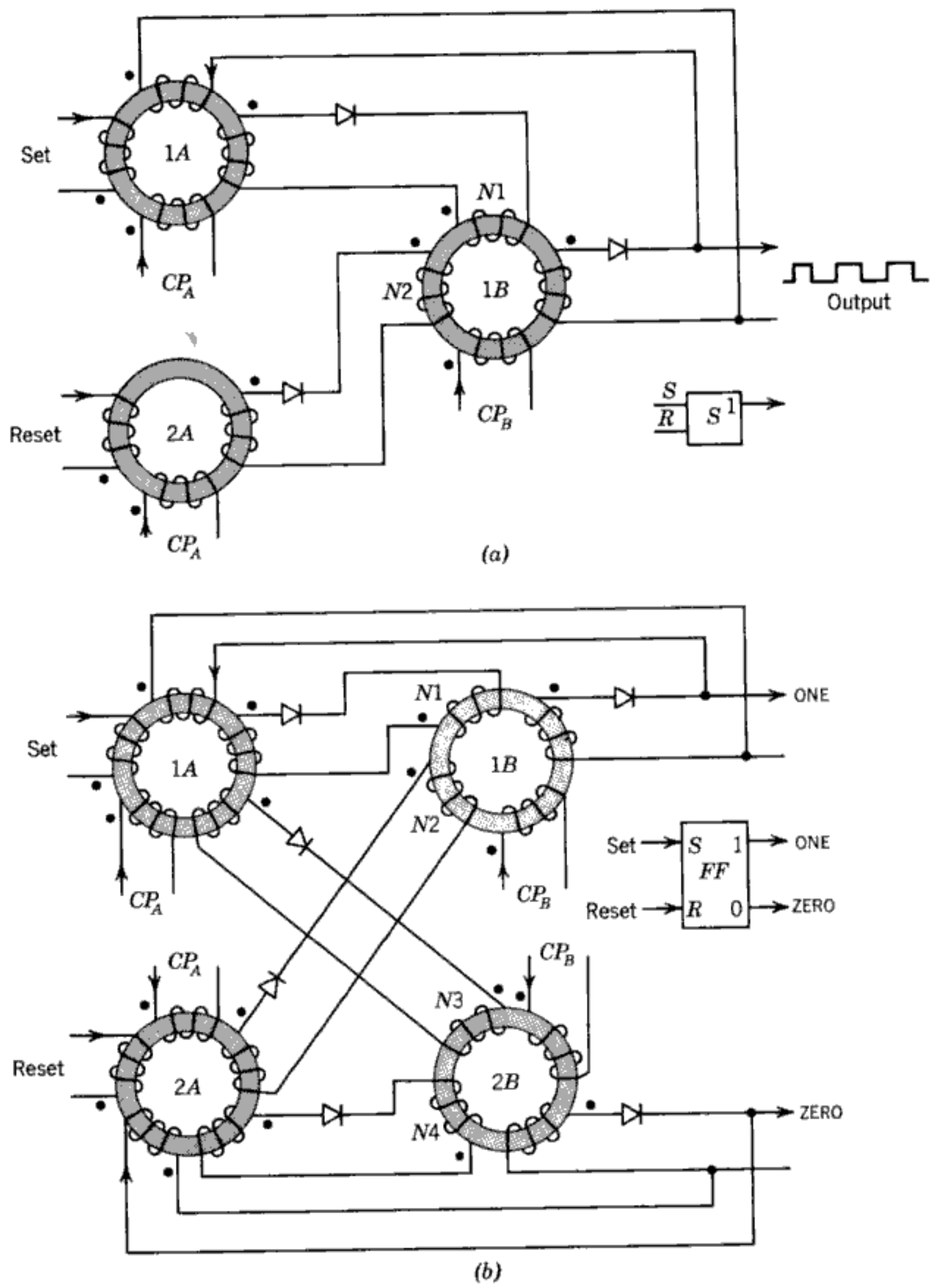


Fig. 81. Dynamic storage. (*a*) Dynamic storage cell. (*b*) Dynamic flip-flop.

inhibited. The result is that the ZERO output core produces a series of pulses while the ONE output core produces no pulses.

Note that, if the output of the dynamic flip-flop is viewed only during CP_B, the output of the flip-flop is essentially the same as that of a static flip-flop.

Magnetic Tape, Drum, and Disk

Magnetic tape may be used to store many binary-coded characters in compact form. In the usual magnetic-tape handler, the tape, which may be several thousand feet long, unwinds from a source reel onto a take-up reel. As it unwinds, it is made to pass under a group of write-read heads at a constant rate. As the tape passes by the heads, each head writes a bit into the cell beneath it during each instant. The bits recorded by any cell make up a track. Figure 71 (p. 161) shows five recorded tracks produced by five magnetic heads. The bits on each track vary according to the character to be represented, as shown.

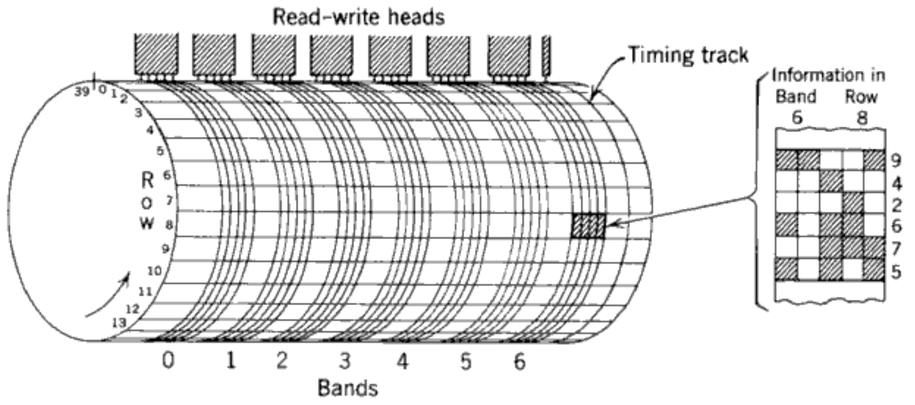


Fig. 82. Magnetic-drum storage.

Reading is accomplished by unwinding the information tape and allowing the heads now activated for reading to pick up the signals from all five tracks at once.

Because tape is long, it takes time to write it or to read it. To speed writing and reading, several tapes may be wrapped around a rotating cylinder, and several more stationary heads may be added to take care of the extra tracks. This is essentially the description of a magnetic drum.

Figure 82 illustrates a magnetic drum with seven bands of five tracks each. A group of five read-write heads services each band. All bands are divided into 40 rows, numbered 0 to 39. A spot on the drum is specified by row and band. The bold area, for instance, is specified by band 6, row 8. At this location is stored the number 942675 in binary-coded decimal form, where positive saturation (hatched area) represents a ONE and negative saturation (clear area) represents a ZERO.

The drum is rotating constantly. To select a location on the drum for either writing or reading, the proper band is chosen by activating the appropriate group of read-write heads. However, it must be activated only

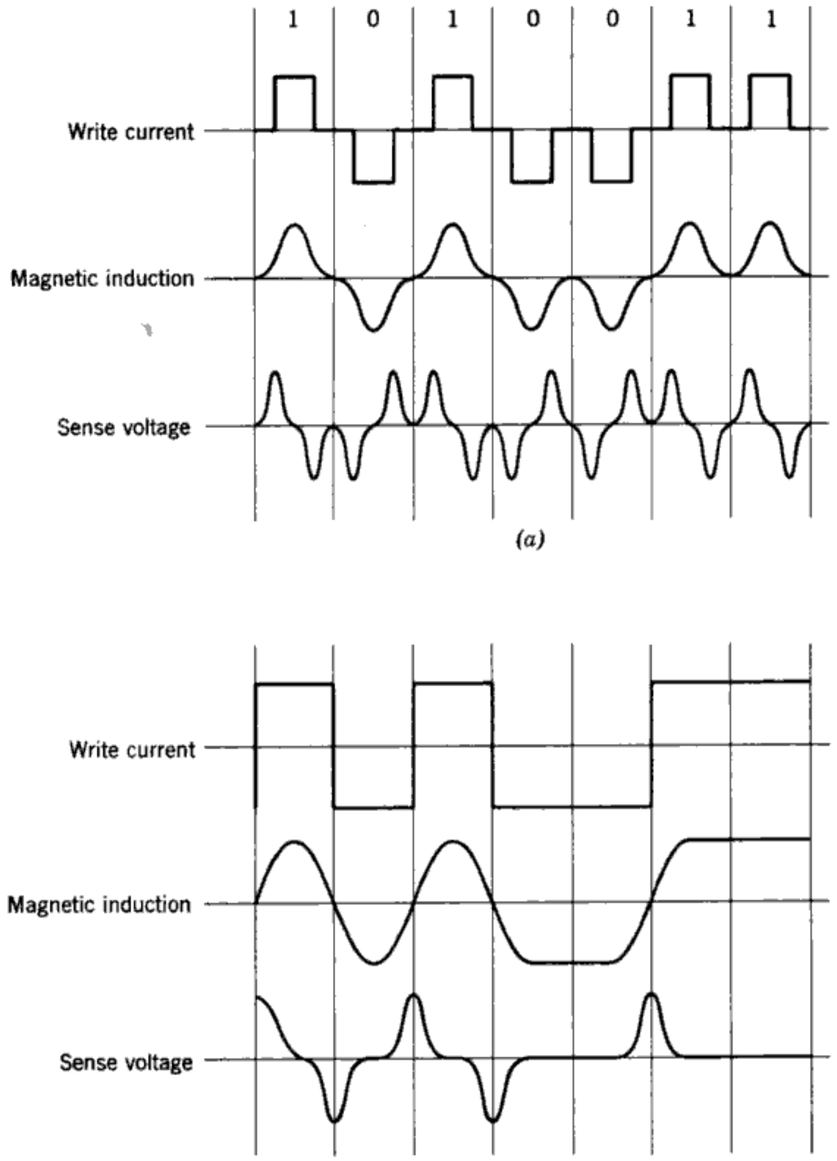


Fig. 83. Types of magnetic recording. (*a*) Return-to-ZERO. (*b*) Nonreturn-to-ZERO.

when the appropriate row is underneath the heads. For instance, in Fig. 82, to read the information in the area shown in bold, the heads servicing band 6 are activated as soon as row 7 comes under the heads.

Evidently the row must be chosen by some sort of timing arrangement. A timing track storing a succession of ONE's helps in this selection.

Several recording schemes are in use for magnetic drums. Two of these are the return-to-ZERO and the nonreturn-to-ZERO schemes. In the

return-to-ZERO recording scheme, a ONE is recorded by feeding a positive current pulse through the write coil, and a ZERO is recorded by feeding a negative current pulse through the write coil. It is called return-to-ZERO because the current always passes through the ZERO point between pulses. Figure 83*a* shows the flux pattern on the drum as the combination 1010011 is recorded with the return-to-ZERO scheme. It also shows the voltage induced into the read coil as it passes under this track during reading. A ONE induces a positive pulse followed by a negative pulse; a ZERO

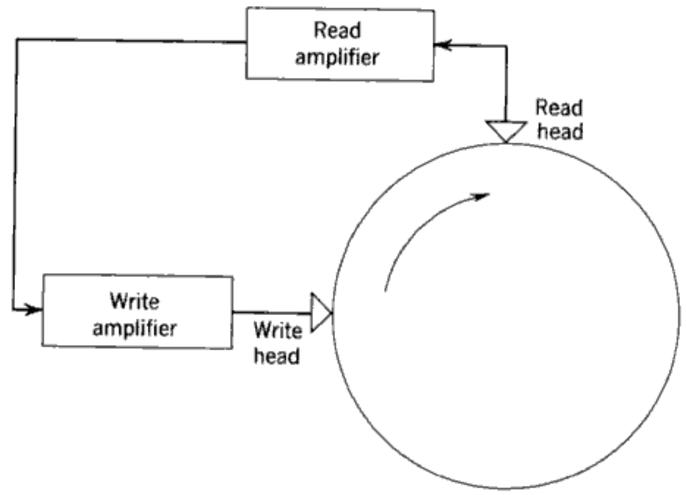


Fig. 84. Drum recirculating register.

induces a negative pulse followed by a positive pulse. To make sense of the pulses read, they are fed to a gate which opens only during the first half of each pulse period. All ONE's come through as positive pulses and all ZERO's as negative pulses.

The current, flux, and voltage waveforms in the nonreturn-to-ZERO recording scheme are depicted in Fig. 83*b*. Here a ONE is represented by a flow of positive current, and a ZERO by a flow of negative current. But the currents do not fall back to zero in every case. When reading a track storing 1010011 in this form, the induced voltage waveform is as shown. Pulses are not produced for each bit of information. Pulses are produced only when the information switches from ZERO to ONE or from ONE to ZERO. A positive pulse is produced when switching from ZERO to ONE, and a negative pulse is produced when switching from ONE to ZERO. To translate these pulses, a storage device—such as a flip-flop—is needed. A positive pulse is made to set the flip-flop to ONE. Each clock pulse gates the ONE output out. A negative pulse is made to reset the flip-flop. The ZERO output is then gated through by clock pulses.

Interestingly enough, a track on a magnetic drum may be used as a recirculating shift register. The arrangement which accomplishes this is shown in Fig. 84. It consists of a read head scanning a track and feeding the bits through an amplifier back to a write head which writes the information further along on the same track. After a group of bits are read and rewritten, the read head appears at the point on the track where it can read the bits once more. This sequence may continue as long as the drum rotates. Several such recirculating registers may be built into one track.

The magnetic tracks may be placed on disks as well as on drums and tapes. Devices have been developed which move the heads to the chosen disk and read the appropriate track when coded signals are applied to them. Other variations are the combination of tape and drum in one system, and the tape file in which a magnetic head may be shifted from one tape to another, thus allowing the storage of millions of bits in one system.

SUMMARY OF ELECTROMAGNETIC LOGICAL BUILDING BLOCKS (Fig. 85)

1. The magnetic toroidal core with square hysteresis loop has two states and may, therefore, be used as a storage cell. A ONE may be stored in one of several cores in an array by causing the magnetic field in the chosen core to be strong enough to flip the core to positive remanence. A ZERO is stored by keeping the core at negative remanence. Reading is accomplished by flipping the core to minus remanence and looking at the sense winding for a voltage (ONE) or no voltage (ZERO).

2. The twistor is similar in action to a toroid. The twistor may be used in a storage array where writing is accomplished by a coincidence of half-currents through the center conductor and through an enclosing solenoid; and reading is accomplished by sending ZERO-flipping current through the enclosing solenoid.

3. By using a clock pulse to set a toroidal core and another clock pulse to shift the information out of the core, a storage cell is effectively converted to a delay element. The length of delay is determined by the time between clock pulses.

4. With magnetic cores, the dynamic form of the storage cell and the flip-flop are commonly used.

5. The OR and INHIBIT circuits are easily formed with magnetic cores. In the case of the OR, two input windings are wound in the same direction. For the INHIBIT, the two input windings are wound in opposite direction, thus producing cancellation of magnetic flux when current is applied to both windings. These switching circuits possess inherent delay.

6. Other switching circuits may be built by utilizing the transformer action of the nonsquare hysteresis loop core.

7. The toroid may be used for amplification. The polarity of the output pulse depends on the signal applied to the input. The strength of the output may be greater than that of the input. The power is obtained from the clock pulse which has a function similar to that of $B+$ in vacuum-tube amplifiers.

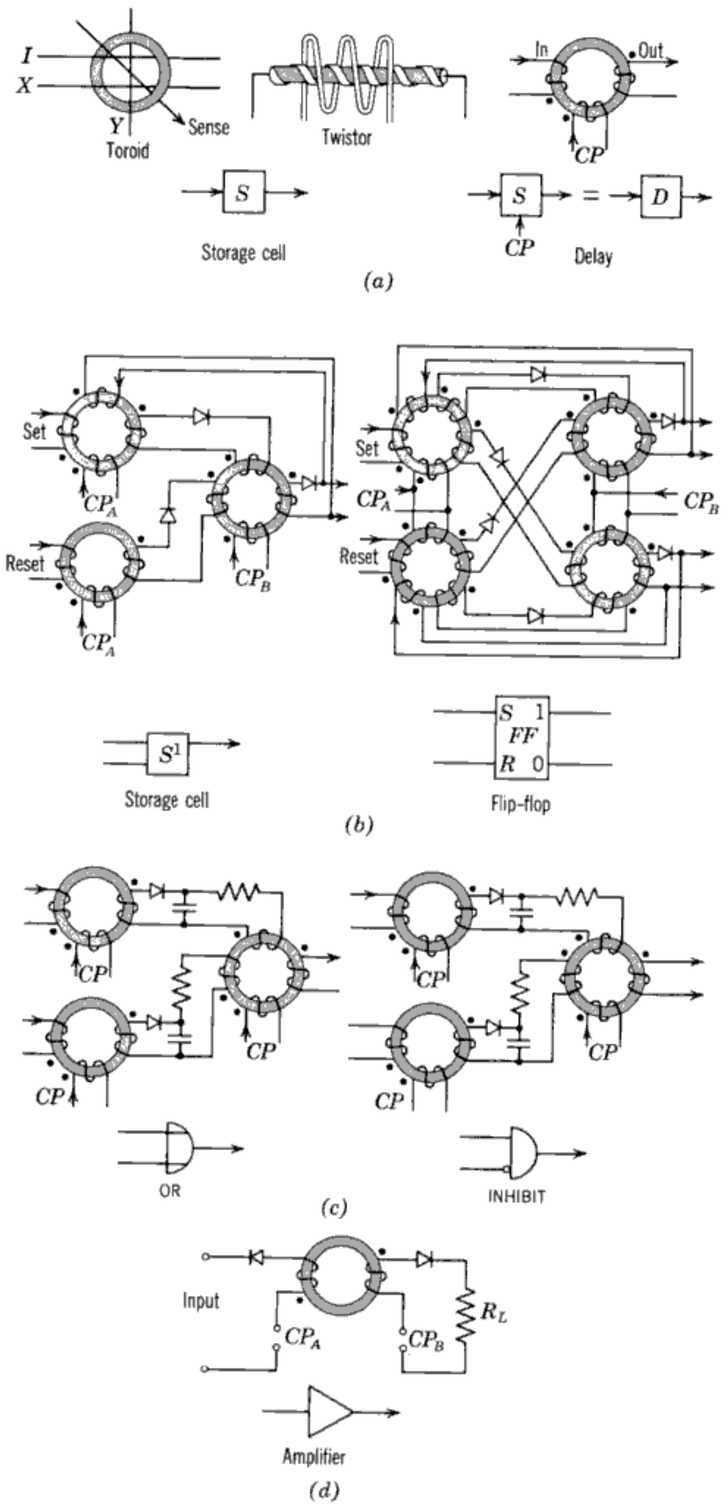


Fig. 85. Electromagnetic logical building blocks. (*a*) Storage. (*b*) Dynamic storage. (*c*) Switching. (*d*) Amplifying.

8. The principles of operation of the magnetic drum, magnetic tape, and disk are similar to those of the magnetic core. Whereas clock pulses shift information out of the core, movement of the tape or drum relative to the heads cause output pulses to be induced in read heads.

9. *Definitions to Remember*

HYSTERESIS—The property of a magnetic material which prevents it from coming back to its original magnetic state after the magnetizing force is removed.

MAGNETIC SATURATION—The point where any further increase in the magnetizing force applied to a magnetic material causes an almost negligible change in the magnetic state of the material. A material may be magnetically saturated in the positive or negative direction.

SQUARE HYSTERESIS LOOP—A loop whose upper and lower saturation lines are fairly flat.

COINCIDENT CURRENT STORAGE—A storage device composed of storage cells in a rectangular array. Information is written into and read from a location by sending current coincidentally to the column and row lines in which a location occurs.

DYNAMIC STORAGE CELL—A logical element which has two possible outputs; a continuous series of pulses (ONE) or of no pulses (ZERO). A set pulse flips the cell into the ONE state; a reset pulse flips the cell into the ZERO state.

DYNAMIC FLIP-FLOP—A circuit with two inputs and two outputs. The set input pulse produces a series of pulses on the 1-output line and no pulses on the 0-output line. The reset input pulse produces the reverse: no pulses on the 1-output line and a series of pulses on the 0-output line.

TRACK—That portion of a moving-tape storage medium—tape, drum—which is accessible to a given writing or reading station.

BAND—A group of tracks which are often written onto or read from at one time.

MODULUS COUNTER—A device which produces an output pulse after a certain number of input pulses or multiples of this number are applied.

REMANENCE—The magnetic state to which a magnetic material falls when the magnetizing force is removed. Materials have positive and negative remanence states.

RETURN-TO-ZERO MAGNETIC RECORDING—Magnetic recording in which a ONE is written by a pulse of current rising to a level and then returning to zero; and a ZERO is written by a pulse of current rising to a level in the opposite direction and then returning to zero.

NONRETURN-TO-ZERO MAGNETIC RECORDING—Magnetic recording in which a ONE is written when current flows in one direction and a ZERO is written when the current flows in the opposite direction. Current passes through the zero point only when a ONE is followed by a ZERO or a ZERO is followed by a ONE.

CLOCK PULSE (CP)—One of a stream of pulses occurring at a constant frequency.

TIMING PULSE—A pulse always occurring at a definite point in a repeated cycle.

CHAPTER 8

Germanium diodes and transistors

Early computers were built of relays and vacuum tubes. These machines were monstrous in size, consumed a great deal of power, and were usually installed in large, completely air-conditioned rooms. These computers were followed by computers built of magnetic-core components. The magnetic core reduced the size and power consumption, and made the computers decidedly more reliable. In recent years the quality of computers has been raised to a new level. Computers have been produced which are desk size, extremely reliable, fast, and consume little power. These computers are built of germanium diodes and transistors. The transistor is tinier than the miniature tube; it has no filament to draw power; it weighs less than a piece of candy-covered chewing gum; and it can perform almost all the functions associated with vacuum tubes, and several the vacuum tube cannot do.

SEMICONDUCTOR LOGICAL ELEMENTS

Basic Semiconductor Physics

From the standpoint of conductivity, there are three types of elements: conductors, nonconductors, and semiconductors. A conductor has free electrons which may be made to flow from point to point by the application of electric charge. A nonconductor has its electrons tightly bound, making electron flow very difficult. A semiconductor is not normally a conductor but may be made conductive under certain circumstances.

A typical semiconductor is germanium. As shown in Fig. 86, an atom of germanium consists of a nucleus with a +32 charge surrounded by 32

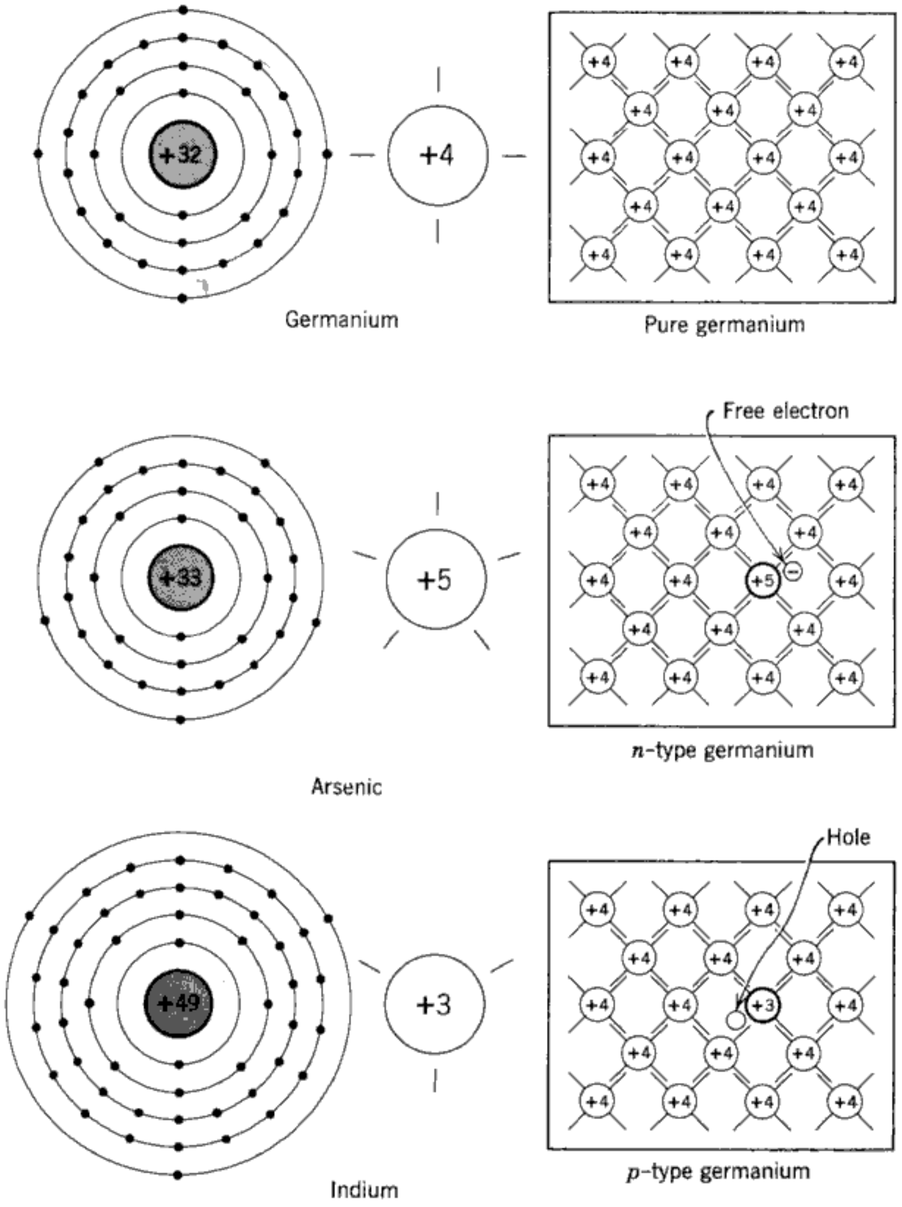


Fig. 86. Pure, *p*-type, and *n*-type germanium.

negatively charged electrons arranged in four orbits. The electrons in the three inner orbits are tightly bound to the nucleus. The four electrons in the outer orbit are loosely bound. This picture may therefore be replaced by the equivalent simpler picture shown: a +4 core (representing the nucleus plus the tightly bound electrons) and four minus electrons.

In a crystal of germanium the atoms arrange themselves neatly in the structure shown in Fig. 86. Loose electrons from adjacent atoms pair off to form what are known as covalent bonds. Since practically all outer orbit electrons are paired off, there is a negligible number of free electrons available for conduction.

If a small impurity is embedded in the germanium, either an *n*-type or a *p*-type crystal may be formed. The *n*-type crystal allows the conduction of negative charges; the *p*-type crystal allows the conduction of positive charges.

The *n*-type crystal may be formed by adding a tiny amount of an element such as arsenic which has five electrons in its outer orbit (Fig. 86). Four of the outer electrons in each atom of arsenic form covalent bonds with electrons from atoms of germanium, thus leaving one electron free to wander about the crystal lattice. These negative free electrons can be made to flow through the crystal by application of a potential across the crystal. Because the arsenic atoms donate electrons, they are called donors. Donor impurities introduce negative free electrons to produce *n*-type germanium; by donating negative electrons, the donor becomes positively charged.

The *p*-type crystals may be formed by introducing into the germanium tiny impurities of an element such as indium which has three electrons in its outer orbit (Fig. 86). The three outer electrons in each atom of indium form covalent bonds with three electrons from atoms of germanium. There remains one electron in a nearby atom of germanium that has no electron in the indium atom with which to pair off. We say the indium atom has a hole with a positive charge. If a potential is applied across the crystal, a negative electron is attracted to this positive hole. This electron leaves a hole in the atom from which it is taken. Then another electron fills the last hole formed, and so on. Even though electrons are actually moving, it is easier to explain semiconductor phenomena by saying that there is a flow of positive holes. Because the indium atoms accept electrons, they are called acceptors. Acceptor impurities introduce positive holes to produce *p*-type germanium. The acceptors by accepting electrons become negatively charged.

Germanium diodes and transistors are formed by combining *p*-type and *n*-type semiconductor crystals.

Germanium Diode

A germanium diode is formed by placing a *p*-crystal next to an *n*-crystal, as shown in Fig. 87*a*. With no potential applied across this *p-n* junction, the negative acceptors in the *p*-type material repel the negative free electrons in the *n*-type material; and the positive donors in the *n*-type material repel the positive holes in the *p*-type material. The result is that the electrons and holes move toward the outer edges—but not entirely to the edges—of the diode.

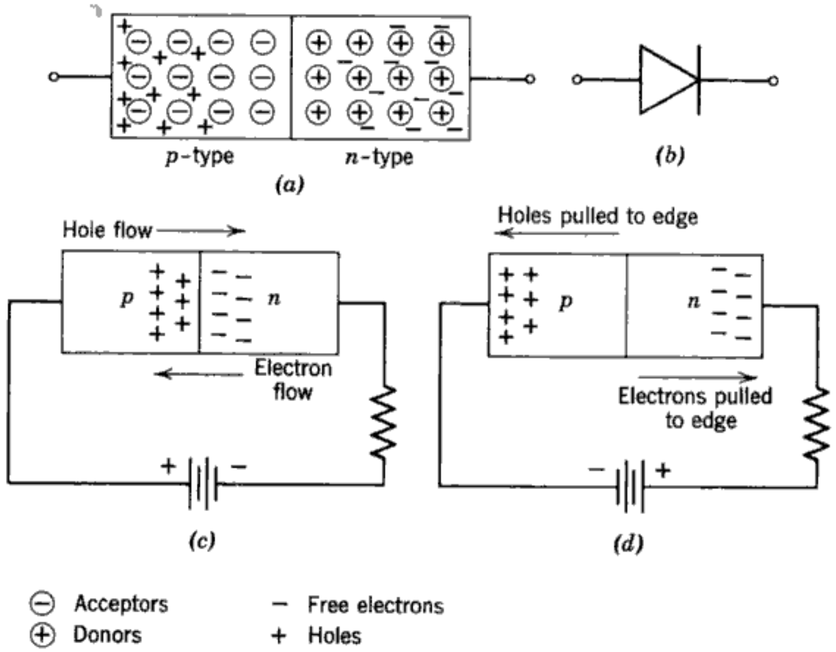


Fig. 87. Germanium diode as a switching component. (*a*) At equilibrium. (*b*) Symbol. (*c*) Diode on. (*d*) Diode off.

If a potential is applied across the diode in such a direction as to make the *p*-crystal positive and the *n*-crystal negative, current flows. The positive potential on the *p*-crystal repels the positive holes toward the junction, and the negative potential on the *n*-crystal repels the electrons toward the junction. Positive or hole current flows from the *p*-crystal to the *n*-crystal (Fig. 87*c*). This direction of current flow is indicated by the arrow of the symbol for the semiconductor diode (Fig. 87*b*).

If a potential is applied across the diode in such a direction as to make the *p*-crystal negative and the *n*-crystal positive, current does not flow. The negative potential on the *p*-crystal attracts the positive holes to the edge of the *p*-crystal; and the positive potential on the *n*-crystal attracts the negative electrons to the edge of the *n*-crystal (Fig. 87*d*).

The semiconductor diode is thus a switch which is similar in operation to the vacuum-tube diode.

Junction Transistors

By forming two *p-n* junctions, as shown in Fig. 88*a*, an *n-p-n* junction transistor is produced. The *n*-crystal at the left is the emitter; the thin *p*-crystal in the middle is the base; the *n*-crystal at the right is the collector. The symbol for the *n-p-n* junction transistor is shown in Fig. 88*b*.

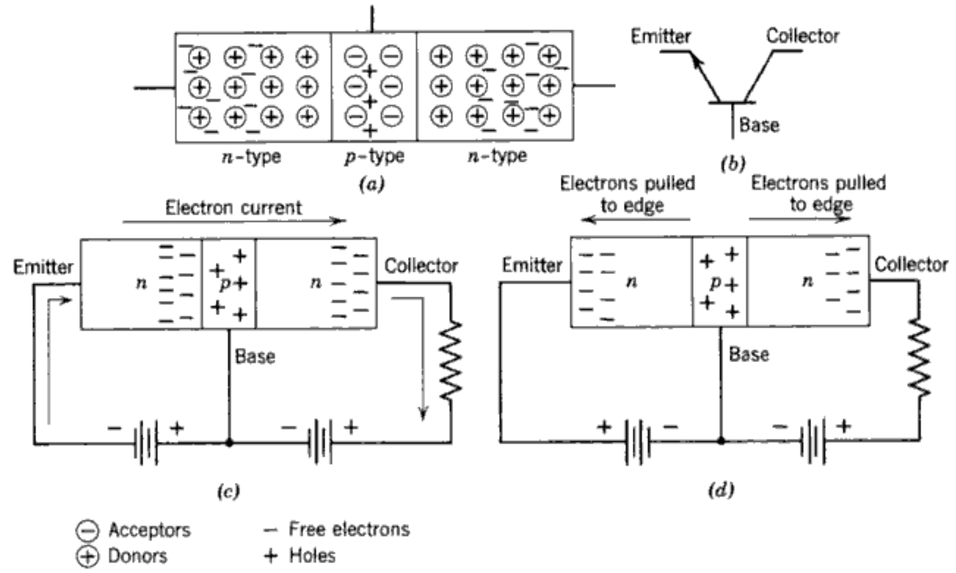


Fig. 88. A *n-p-n* junction transistor as a switching component. (*a*) At equilibrium. (*b*) Symbol. (*c*) Transistor conducting. (*d*) Transistor cut-off.

Figure 88*a* shows the *n-p-n* junction transistor when no potentials are applied to it. The positive holes are repelled to the center of the base by the positive donors in both *n*-crystals. The negative acceptors in the base repel the free electrons toward the edges of both *n*-crystals.

If the emitter is made negative with reference to the base, and the collector is made positive with reference to the base (Fig. 88*c*), electron current flows from emitter through the base to the collector. The negative potential applied to the *n*-crystal emitter repels electrons toward the base; and the positive potential applied to the *n*-crystal collector pulls these electrons out of the base toward the collector. The extreme thinness of the base does not allow the electrons much time to combine with holes in the base. Note that electrons flow opposite to the direction of the arrow on the emitter of the transistor symbol; in other words, the arrow indicates the direction of *positive* current flow.

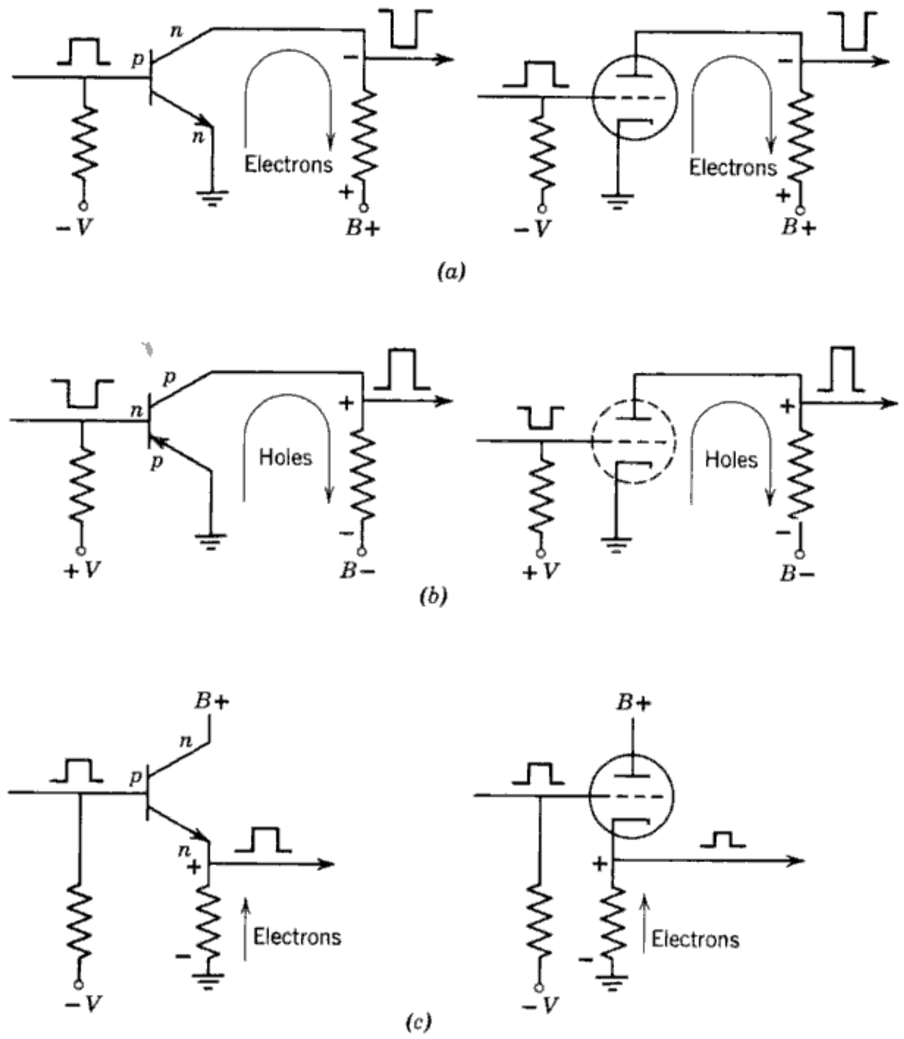


Fig. 89. Junction transistor as an amplifying component. (*a*) An *n-p-n* inverting amplifier and an analogous triode amplifier. (*b*) An *p-n-p* inverting amplifier and an analogous fictitious triode amplifier. (*c*) An *n-p-n* emitter follower and an analogous cathode follower.

If the emitter is made positive with reference to the base, and the collector is made positive with reference to the base (see Fig. 88*d*), then no electrons flow from emitter to collector. The positive potential applied to the *n*-crystal emitter pulls electrons over to the left edge; and the positive potential applied to the *n*-crystal collector pulls electrons over to the right edge. The transistor is cut off.

Because the *n-p-n* transistor has two stable states—cutoff and conduction—it may be used as a switching component. It is also an amplifying component, as can be seen from Fig. 89*a*. The base and emitter are in the

input circuit; the emitter, base, and collector are in the output circuit. The more positive the base is made relative to the emitter, the more electrons flow from emitter through the base to the collector. The potential applied between base and emitter may be small, but a larger potential may be developed across the collector load. Electrons flowing toward *B+* produce a voltage drop across the load in such a direction as to invert the phase of the amplified signal.

The above circuit of the *n-p-n* inverting amplifier is analogous in operation to the vacuum-tube amplifier. The emitter is analogous to the cathode; the base to the grid; and the collector to the plate. A positive potential is required on the collector just as it is on the plate of a triode to keep the transistor operating. Cutoff of the transistor is obtained by applying a negative bias to the base, just as it is obtained in the vacuum tube by making the grid negative. A positive signal on the base of the transistor causes conduction which produces an inverted amplified signal at the collector, just as a similar signal applied to the grid produces an amplified inverted signal in the plate of a triode.

By making the emitter and collector of *p*-crystals and the base of *n*-crystals, a *p-n-p* junction transistor is formed. The operation of the *p-n-p* is similar to the operation of the *n-p-n* transistor, except that operating potentials and current flow are in reverse directions. At equilibrium the holes are repelled by the positive donors in the base toward the edges of the emitter and collector. With a negative collector and a negative emitter (with reference to the base), these holes are repelled further to the edges: no holes flow from emitter to collector. If a negative pulse is applied between the base and the emitter—thus making the emitter relatively positive—the holes in the emitter are repelled to the base and then attracted to the collector. Positive hole current flowing through the collector load produces an inverted amplified signal at the collector.

The operation of the *p-n-p* junction transistor is analogous in operation to a fictitious vacuum-tube triode containing a cathode which emits holes instead of electrons (see Fig. 89*b*).

Note that the arrow in the symbol for the *p-n-p* transistor points in the direction of positive current flow.

Figure 89*c* shows an emitter-follower circuit using an *n-p-n* which produces an in-phase output in a manner analogous to the way the cathode-follower circuit operates. An emitter-follower circuit may also be formed of a *p-n-p* transistor.

Marginal Checking

The transistor has no filament. Therefore, marginal checking must be performed by varying the potential applied to another electrode, usually

the collector. In a good transistor, lowering the potential on the collector to the lower tolerance margin has no effect on its operation. If the transistor does become inoperative as a result of lowering the collector potential, it means that it is wearing out and should be replaced.

SWITCHING CIRCUITS

Diode Switching Circuits

Germanium diode switching circuits are formed the same way vacuum-tube diode switching circuits are formed. An AND and an OR circuit are shown in Fig. 90. Only when all signals applied to the AND circuit are

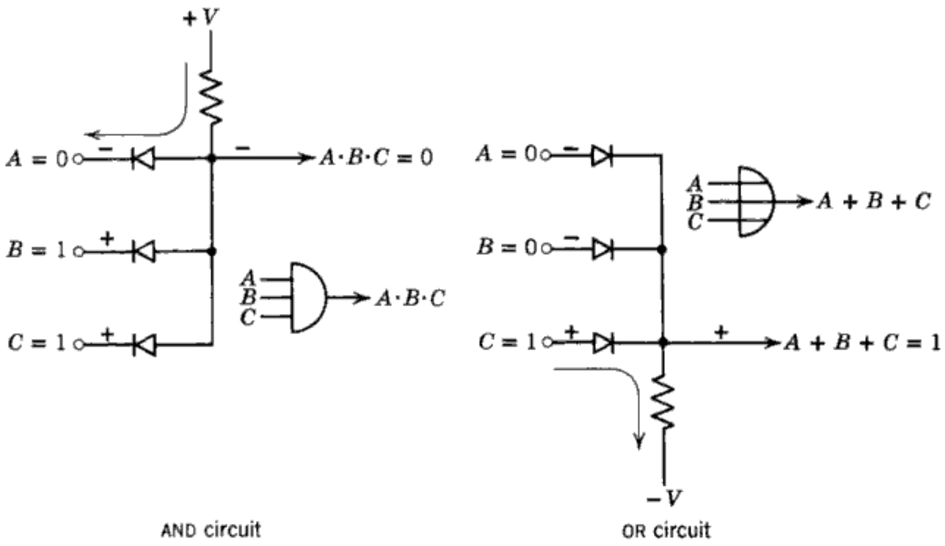


Fig. 90. Diode switching circuits.

positive, do none of the diodes conduct and produce a positive voltage at the output. If any signal to the OR circuit is positive, the associated diode conducts and produces a positive output voltage.

Several double-diode AND circuits may be combined in a rectangular-diode decoding matrix such as that shown in Fig. 91 to produce any or all possible combinations of signals. Two variables A and B are combined in four different ways in the four AND circuits at the top; two other variables C and D are combined four different ways in the four AND circuits at the left. The four A–B combinations are then combined with the four C–D combinations in the sixteen AND circuits in the rectangle. With this arrangement, each combination of four input signals chooses one of sixteen possible output points.

From the above illustration it is obvious that a decoder consists of a combination of AND circuits. The AND circuits of a decoder may be arranged differently. Figure 92 shows a circuit which decodes numbers in excess-3 code to decimal. Essentially this decoder consists of ten AND

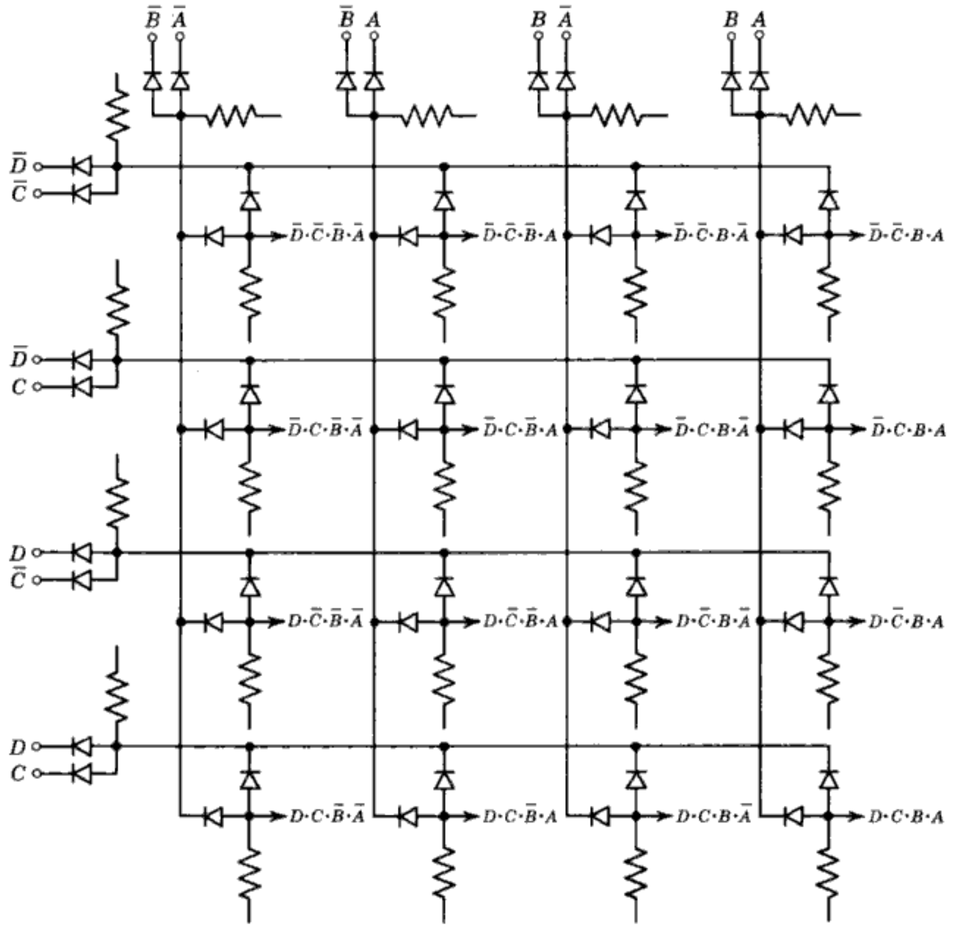


Fig. 91. Rectangular-diode decoding network.

circuits, each AND circuit being composed of four diodes attached to a horizontal line, and the resistor in the horizontal line. Four input bits are applied to each of the ten AND circuits by sending a positive pulse to either the ONE or ZERO line of each of the four pairs of vertical lines. All other vertical lines are negative. Each horizontal line—AND circuits—which has a diode connected to a negative vertical line produces a negative output since the diode conducts. The one horizontal line which is connected through diodes to the four positive vertical lines produces a positive output since none of the diodes conducts.

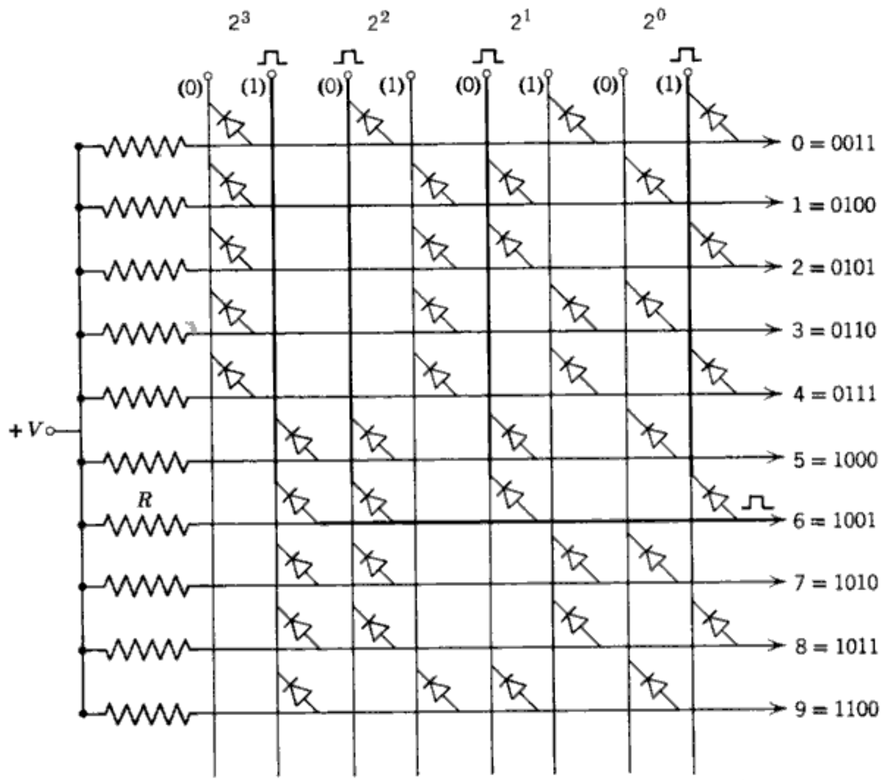


Fig. 92. Diode decoding matrix—excess-3 code to decimal.

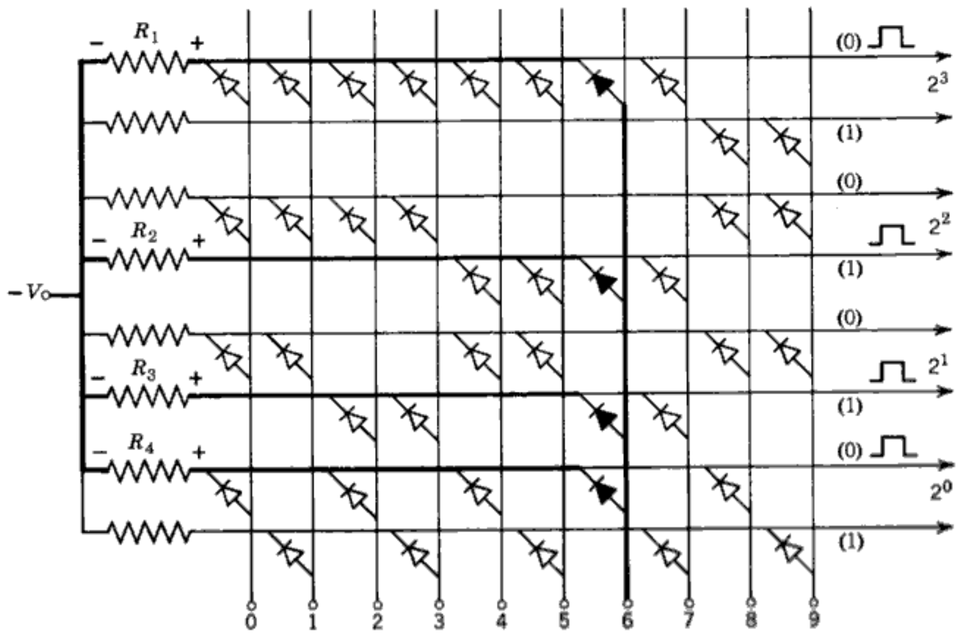


Fig. 93. Diode encoding matrix—decimal to binary-coded decimal.

The illustration shows 1001 decoded into 6. The bold, vertical lines are positive; all other vertical lines are negative. A positive potential applied to all four diodes attached to horizontal line 6 causes all of these diodes to be open. No voltage drop is produced across R and a positive signal is developed at the output.

Just as a decoder consists of a group of AND circuits, an encoder may be considered to consist of a group of OR circuits. A circuit which encodes decimal numbers to binary-coded decimal is given in Fig. 93. The diodes along each horizontal line together with the resistor in each horizontal line compose an OR circuit. A signal applied to each of the ten vertical input lines feeds a positive potential to a different combination of four OR circuits. For instance, a positive potential applied to line 6 feeds current through the diodes shown to the four horizontal bold lines. As a result, positive signals are produced across $R1$, $R2$, $R3$, and $R4$. The output is therefore 0110 or binary 6.

Both the decoder and the encoder do translation. The decoder translates a combination of signals into one of a group of signals; the encoder from one of a group of signals into a combination of signals. The general case of translating from a group of signals to a different group of signals may be accomplished by combining a decoder with an encoder. For instance, to translate from excess-3 to binary-coded decimal, the excess-3 signals may be first applied to the decoder of Fig. 92. The decimal line outputs of the decoder are connected to the inputs of the encoder of Fig. 93. The output of the second circuit is the required result. In the example used, the 1001 of excess-3 chooses line 6 in the decoder. This line is connected to line 6 of the encoder and thus causes 0110 to be produced by the encoder.

By means of logic, the decoder and encoder may be merged into a translator matrix using fewer diodes.

Transistor Switching Circuits

Switching circuits may be produced by connecting transistors in series or in parallel. When connected in series, as shown in Fig. 94*a*, the circuit is an INVERTED AND. Only when a positive potential is applied to both bases does hole current flow as indicated. The hole current flowing through the load resistor produces a negative output. When connected in parallel, as shown in Fig. 94*b*, the circuit is an INVERTED OR. If the base of either transistor is made positive, the transistor conducts, and the current flowing through the common load resistor produces a negative output.

Remembering that an INVERTED AND is formed by a series connection and that an INVERTED OR is formed by a common load resistor, almost any logical function can be easily duplicated. Figure 94*c*, for instance,

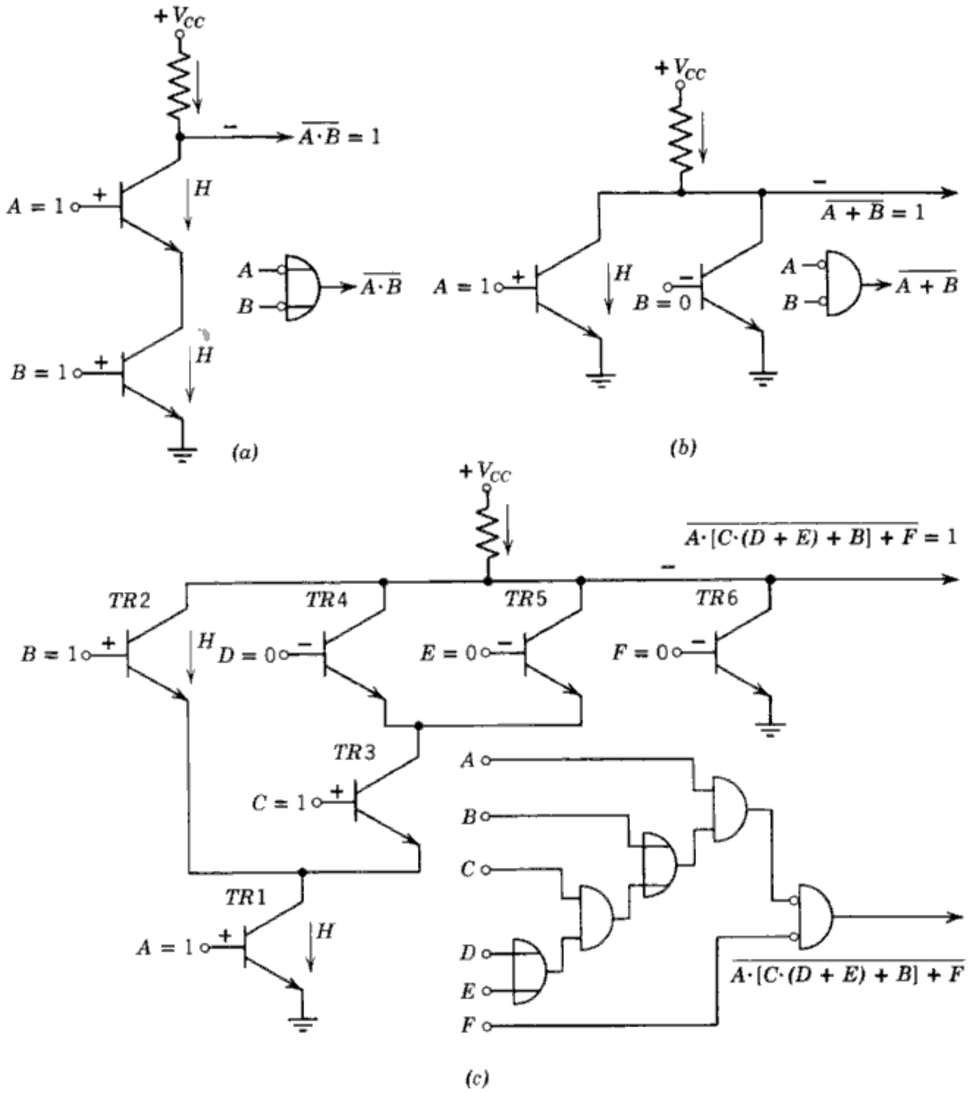


Fig. 94. Junction transistor switching circuits. (*a*) INVERTED AND. (*b*) INVERTED OR. (*c*) Combination of switching circuits.

illustrates a circuit which produces

$$\overline{A \cdot [C \cdot (D + E) + B] + F}$$

If for the moment, polarity inversions are disregarded, the parallel combination of *TR*4-*TR*5 produces $D + E$. By placing these transistors in series with *TR*3, the expression $C \cdot (D + E)$ is produced. Paralleling all this with *TR*2 produces $C \cdot (D + E) + B$. Placing *TR*1 in series with this combination produces $A \cdot [C \cdot (D + E) + B]$. Connecting all of this in parallel with *TR*6 produces $A \cdot [C \cdot (D + E) + B] + F$. This expression

indicates how the currents in the various transistors combine. When the current flows through the load resistor, it produces a negative output voltage. The output of this circuit is therefore

$$\overline{A \cdot [C \cdot (D + E) + B] + F}$$

The illustration indicates what happens when $A = 1$, $B = 1$, and $C = 1$. The bases of *TR*2, *TR*1, and *TR*3 are positive; the others are negative. Though the base of *TR*3 is positive, this transistor does not conduct since the bases of both *TR*4 and *TR*5 are negative thus maintaining an effective open circuit. However, since the bases of both *TR*2 and *TR*1 are positive, hole current can flow through the path given by the load resistor, *TR*2 and *TR*1 to ground to produce a negative output. Therefore, the output is equal to

$$\overline{A \cdot [C \cdot (D + E) + B] + F} = 1$$

Another example of a transistor switching circuit is the 5421 coded-decimal forbidden-combination detector illustrated in Fig. 95. A forbidden-combination detector determines if the coded bits of a number are forbidden by the code in use. For instance, in the 5421 system, the six combinations

0101
0110
0111
1101
1110
1111

have no meaning, and their presence anywhere in the computer indicates an error has occurred.

The forbidden-combination detector may take on two forms. It may either test for proper combinations or it may test for forbidden combinations. In the first case, the output of the detector allows further transmission of the information. In the second case, the output of the detector is used to prevent further transmission.

The first version of the circuit is obtained by noticing that all proper combinations have $C = 0$ or else $C = 1$ and $B = 0$ and $A = 0$, and that this is never true for forbidden combinations. The logical expression for the proper combination is therefore

$$\bar{C} + C \cdot \bar{B} \cdot \bar{A}$$

This is produced by the *p-n-p* transistor circuit shown in Fig. 95a. If the combination 0100, for instance, is fed to the circuit, the bases of transistors *TR*2, *TR*3, and *TR*4 are made negative (in this circuit, negative means ONE and positive means ZERO) and the others positive. As a result, electrons

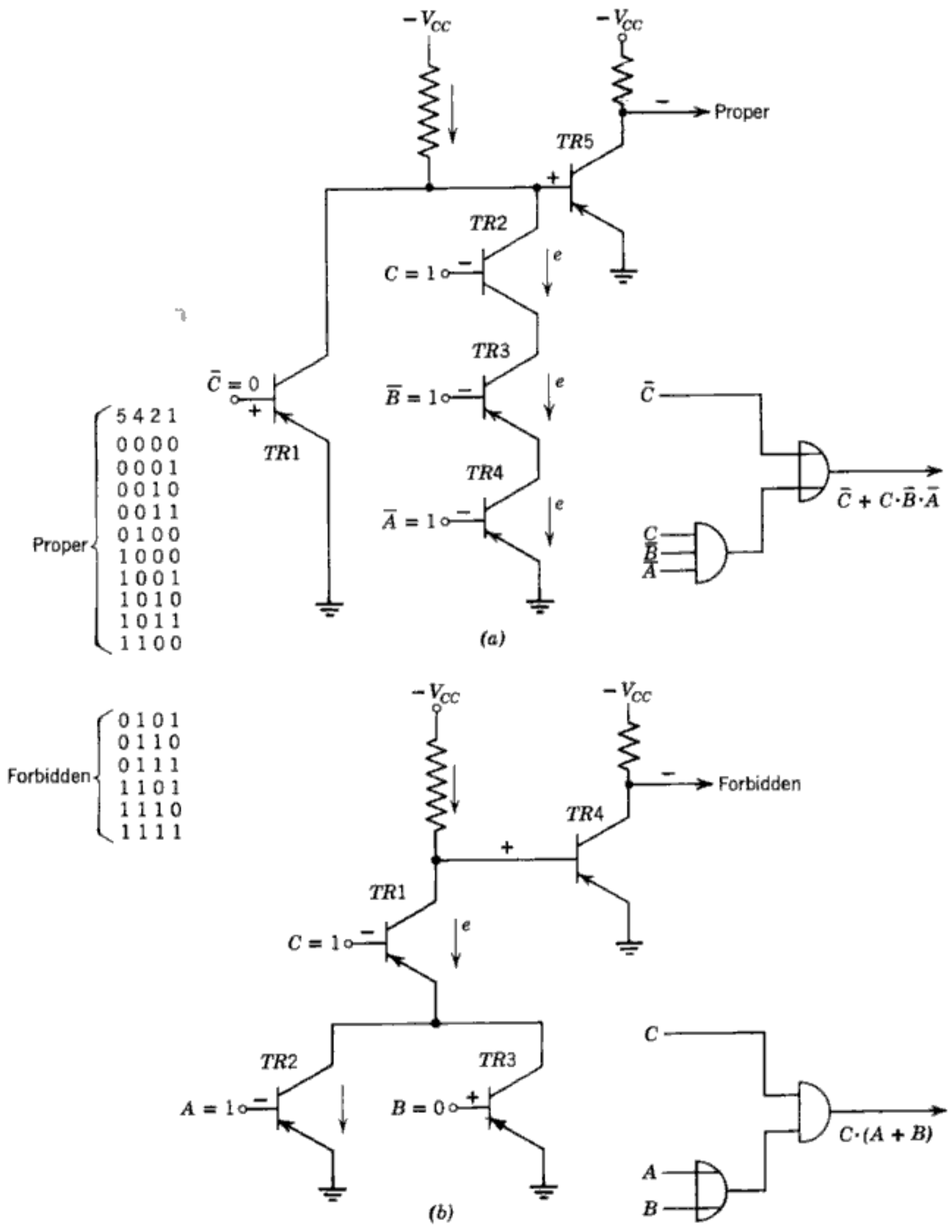


Fig. 95. Forbidden-combination detector in 5421 coded decimal. (*a*) Based on proper combinations. (*b*) Based on forbidden combinations.

flow through *TR*2, *TR*3, and *TR*4, and a positive output is produced across the load resistor. Transistor *TR*5 inverts the signal to make it negative.

The second version of the circuit is obtained by noting that all forbidden combinations have either *A* and *C* equal to ONE, or *B* and *C* equal to ONE,

and that this is never true of a proper combination. The logical expression for the forbidden combination is therefore

$$A \cdot C + B \cdot C = C \cdot (A + B)$$

which is produced by the circuit shown in Fig. 95b. If the forbidden combination 1101 is applied to this circuit, the base of *TR*1 and *TR*2 are negative and the base of *TR*3 is positive. Electrons flow through *TR*1 and *TR*2 to produce a positive output which is reversed by *TR*4.

AMPLIFYING CIRCUITS

Like the triode and tetrode, the junction transistor may be either an amplifier or an inverting amplifier. With the emitter grounded and the output taken from the collector, inversion is produced. With the collector tied to a potential and the output taken from a load in the emitter—an emitter-follower circuit—no inversion is produced. In the first case voltage amplification is achieved, and in the second case current amplification.

Figure 96*a* shows how an *n-p-n* transistor inverting amplifier may be used between groups of diode circuits. The circuit shows a level of AND circuits (diodes *D*1 to *D*5) followed by a level of OR circuits (*D*6 to *D*9), and another AND circuit (*D*10 and *D*11). This is the general arrangement of practically all logical diode circuits. The output of the AND circuit is fed to the transistor amplifier to restore the signal energy. But the transistor inverts the phase too, and the NOT function is produced.

Suppose we do not want the NOT function but the function itself. Must the signal be reinverted? Not necessarily. For the next group of logical circuits we may form the convention that a negative pulse is ONE and a positive pulse is ZERO. If this is done, the same configuration of diode circuits as shown in the illustration represents the three logical levels of OR-AND-OR. Thus by alternating definitions for each successive group of three diode levels, an unbroken succession of AND-OR-AND-OR-AND . . . levels may be obtained.

If the circuit illustrated accepts positive signals (Fig. 96*b*), then the polarities shown at the inputs represent

$$\begin{array}{ll} A = 1 & D = 1 \\ B = 1 & E = 1 \\ C = 0 & F = 1 \\ & G = 0 \end{array}$$

With these signals applied, a positive pulse—a ONE—is produced at the

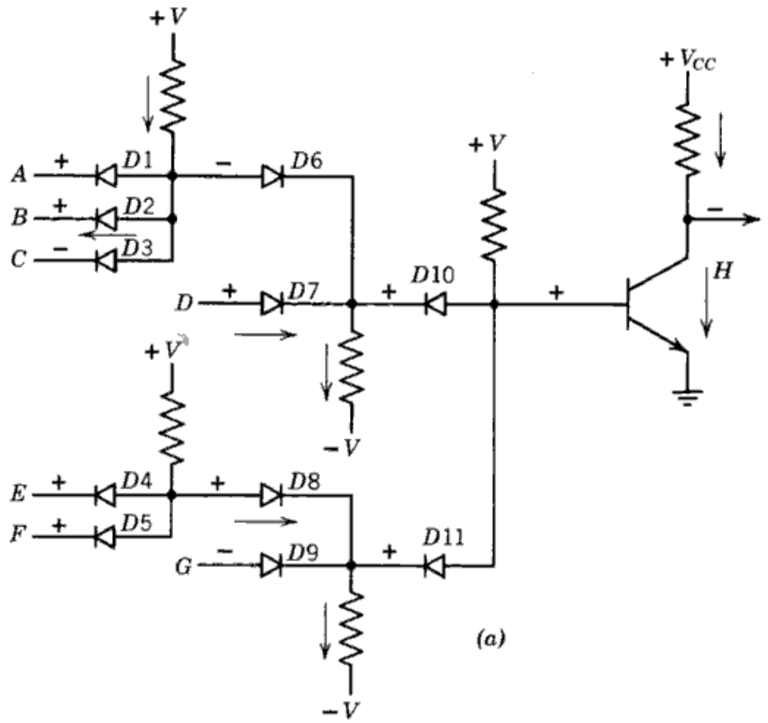


Fig. 96. Diode switching—transistor amplifier combination. (*a*) Schematic. (*b*) Logic representation for *positive* signals. (*c*) Logic representation for negative signals.

base of the transistor. The transistor inverts the potential to make it negative.

If the same circuit is used to accept negative signals (Fig. 96*c*), then the polarities at the inputs represent

$A = 0$ $\quad D = 0$
$B = 0$ $\quad E = 0$
$C = 1$ $\quad F = 0$
$\quad\quad G = 1$

Obviously the same positive potential is produced at the base of the transistor. But this positive potential represents a ZERO this time. From the logical block diagram it is obvious why the output should be ZERO. The transistor may once more invert the phase so that positive signals may be assumed to represent ONE, and the process continues.

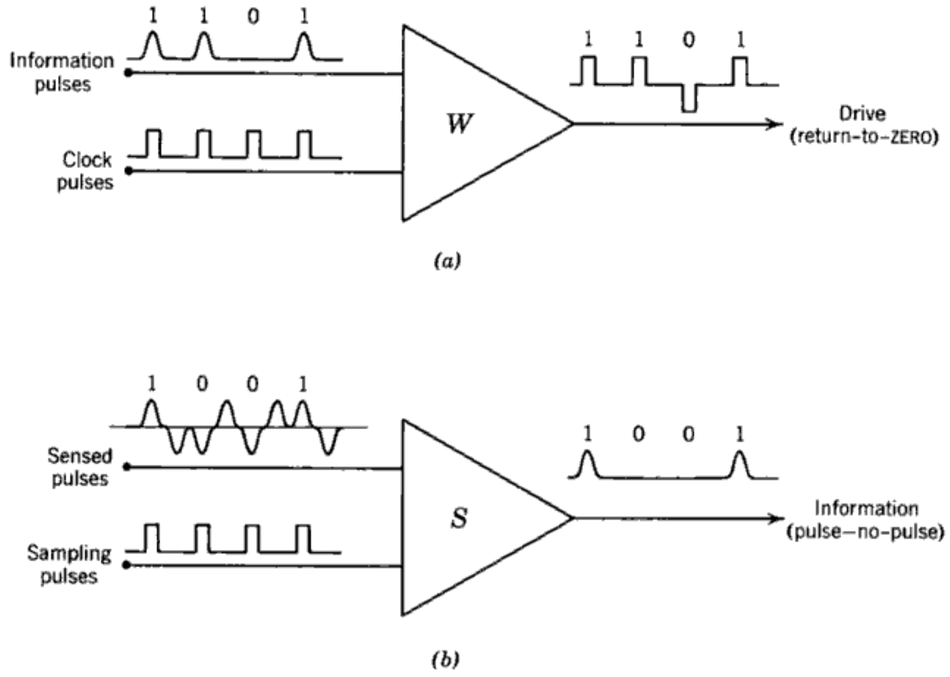


Fig. 97. Write and read (or sense) amplifiers. (*a*) Write amplifier. (*b*) Read (or sense) amplifier.

This illustration shows the significance of polarities in the determination of logical circuits. In other combinations the polarities may influence the logic in a different way. As long as definitions of ONE's and ZERO's are kept firmly in mind, no difficulty need occur in our understanding of the logic.

Some amplifiers used in computers are not as straightforward as those mentioned up to now. This is especially true of amplifiers used in the writing and the reading of storage devices. The additional complication in these circuits arises from the fact that the write and read amplifiers do more than amplify. The write amplifier associated with a drum, for instance, translates computer informational pulses into pulses which are of proper polarity, width, and power to magnetize appropriately the drum areas when these pulses are applied to the write head. Similarly, the drum read amplifier translates the pulses sensed by the read head into information pulses that the remainder of the computer can understand. Figure 97 shows in a very simplified form what the write and read amplifiers

do in a computer using a pulse–no-pulse informational system and a magnetic drum with return-to-ZERO recording.

The write amplifier operates as follows. The pulse–no-pulse combination for the information is fed to a gate-type circuit at the same time that

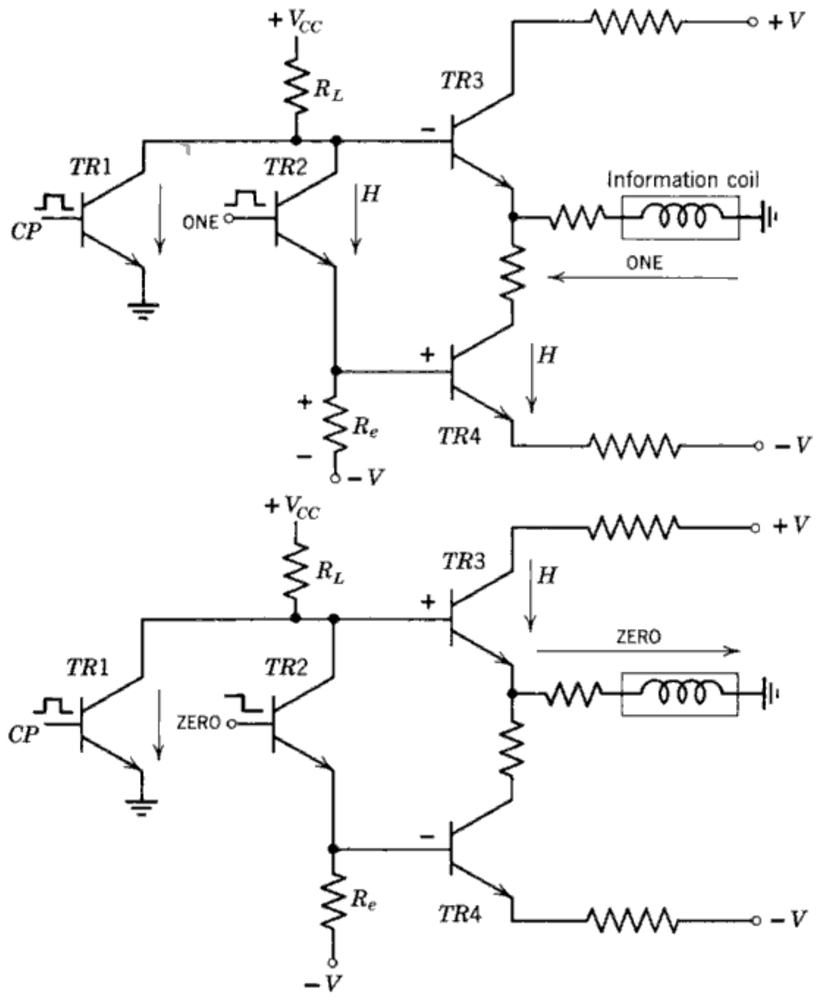


Fig. 98. Write amplifier for core memory.

clock pulses are applied to this gate. The circuit is arranged so that, if a ONE (a pulse) is applied, an amplifier is activated to send current through the write coil on the head in one direction; if a ZERO (no pulse) is applied, another amplifier is activated to send current through the write coil in the opposite direction.

A simplified schematic of a circuit which accomplishes this operation is shown in Fig. 98. With a positive clock pulse on the base of *TR*1 and a

positive pulse (ONE) on the base of *TR*2, both these transistors conduct to make the base of *TR*3 negative enough to keep it cut off. At the same time, the current across R_e produces a positive potential which turns on *TR*4. Transistor *TR*4 sends hole current through the writing coil in the direction shown. With no pulse (ZERO) applied to *TR*2 at the same time *CP* is applied to *TR*1, transistor *TR*1 conducts while *TR*2 does not. Current produced by only one of these transistors is not sufficient to make the base of *TR*3 negative enough to open it. The base of *TR*3 is relatively positive while the base of *TR*4 is negative. As a result, *TR*3 conducts and sends hole current through the writing coil in the direction shown.

The read amplifier accepts the waveform sensed by the sense winding. With a return-to-ZERO recording scheme, a ONE is sensed as a positive pulse followed by a negative pulse, and a ZERO is sensed as a negative pulse followed by a positive pulse. To avoid confusion, a sampling pulse allows only the first pulse of the pair to pass through an AND circuit and to be applied to an amplifier circuit.

These write and read amplifiers are merely examples of the circuitry required. For different informational systems and different storage systems, other types of write and read amplifiers are required. Since write and read amplifiers are more a problem for the electronics expert than for the computer expert, they are not discussed any further.

STORAGE CIRCUITS

Flip-Flop

The junction transistor flip-flop is similar in operation to the vacuum-tube flip-flop. A simple transistor flip-flop is drawn in Fig. 99. One transistor or the other is conducting at any one time. If *TR*1 is conducting, the negative potential produced at the *TR*1 collector by R_{L1} is fed to the base of *TR*2 to keep *TR*2 cut off. The nonconducting *TR*2 keeps its collector and, therefore, the base of *TR*1 positive. As a result, *TR*1 is maintained at high conduction and *TR*2 at cutoff. This is the set or ONE state. To flip the flip-flop to the ZERO state, a positive pulse may be applied to the base of *TR*2 to cause it to conduct; *TR*2 conducting, in turn, cuts off *TR*1. The same result can be obtained by applying a negative pulse to the collector of *TR*2 and to the base of *TR*1 to cause *TR*1 to cut off and *TR*2 to conduct.

The flip-flop may be flipped back to the ONE state by a negative set pulse applied to the collector of *TR*1 and to the base of *TR*2.

In the set state a positive signal appears at the ONE output line and a negative signal at the ZERO output line. In the reset state a positive signal appears at the ZERO output line and a negative signal at the ONE output line.

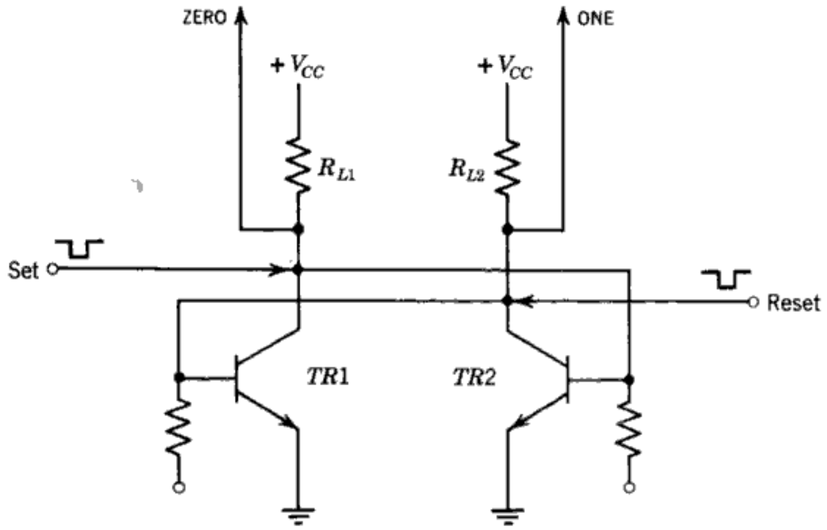
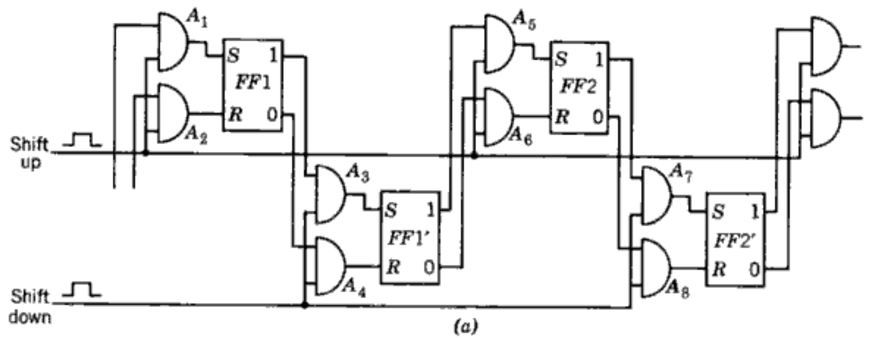


Fig. 99. Transistor Flip-flop.

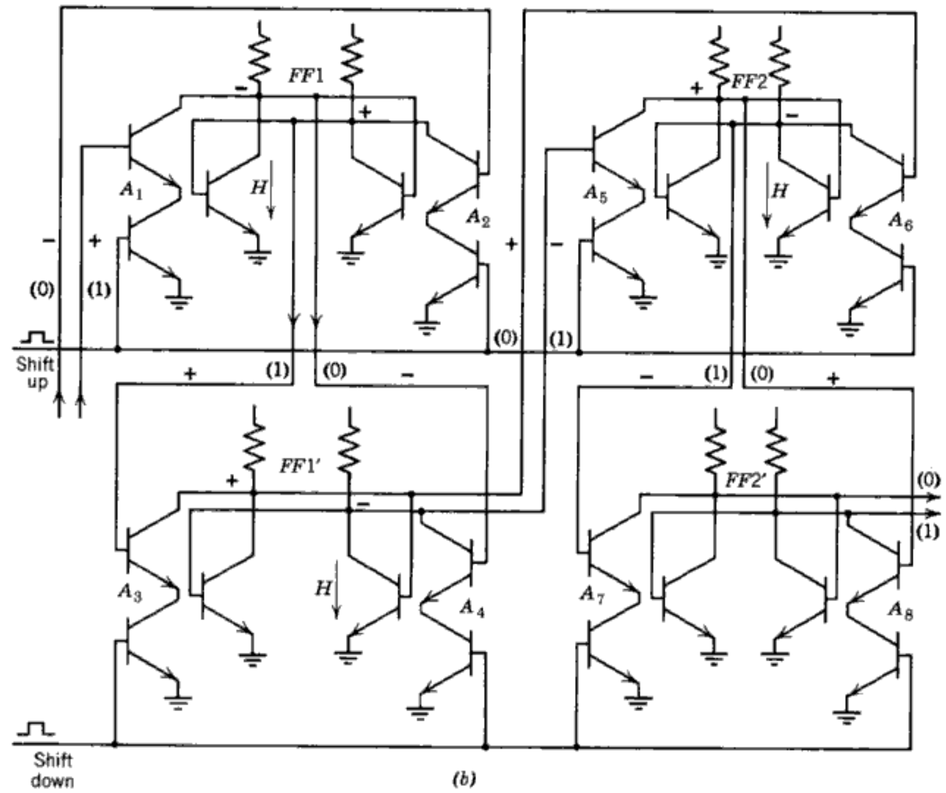
Shift Register

Transistor flip-flops may be used as storage cells in a shift register. As was shown in the previous chapters, to reduce ambiguity of operation, a delay must be introduced between storage cells. Even more reliable operation can be obtained by means of a double-rank shift register. A double-rank shift register (Fig. 100*a*) has two storage cells for each bit. One storage cell (*FF*1, *FF*2, etc.,) stores the bit; the other cell (*FF*1′, *FF*2′, etc.,) acts as temporary storage during the transfer of the bit from one storage cell to the next. The block diagram shows the two storage levels connected by means of AND switches to produce right shifts. A right shift is accomplished by applying a shift-down pulse and then a shift-up pulse. The shift-down pulse primes the AND circuits to the temporary storage cells; bits stored in the information flip-flops *FF*1, *FF*2, etc., are fed to flip-flops *FF*1′, *FF*2′, etc. The following shift-up pulse primes the AND circuits to the information storage cells; information stored in the temporary storage cells are fed to the information flip-flops. Each succeeding pair of shift pulses shifts the information one bit position to the right.

Another set of AND circuits may be connected between flip-flops in such a manner as to shift bits from information cells to temporary storage cells, and then to information cells to the left to produce left shifts.



(*a*)



(*b*)

Fig. 100. Double-rank transistor shift register. (*a*) Block diagram. (*b*) Schematic.

A circuit of a transistor double-rank shift register is shown in Fig. 100*b*. Information flip-flops *FF*1 and *FF*2 are shown on the upper level; temporary storage flip-flops *FF*1′ and *FF*2′ are shown on the lower level. Set pulses are fed to the left-hand collector of each flip-flop through an AND circuit consisting of two transistors in series. The negative set pulse is produced only when a ONE is applied to one transistor and a shift pulse to the other transistor of the AND circuit. Similarly, reset pulses are fed to the right-hand collector of each flip-flop through a series AND circuit. The negative

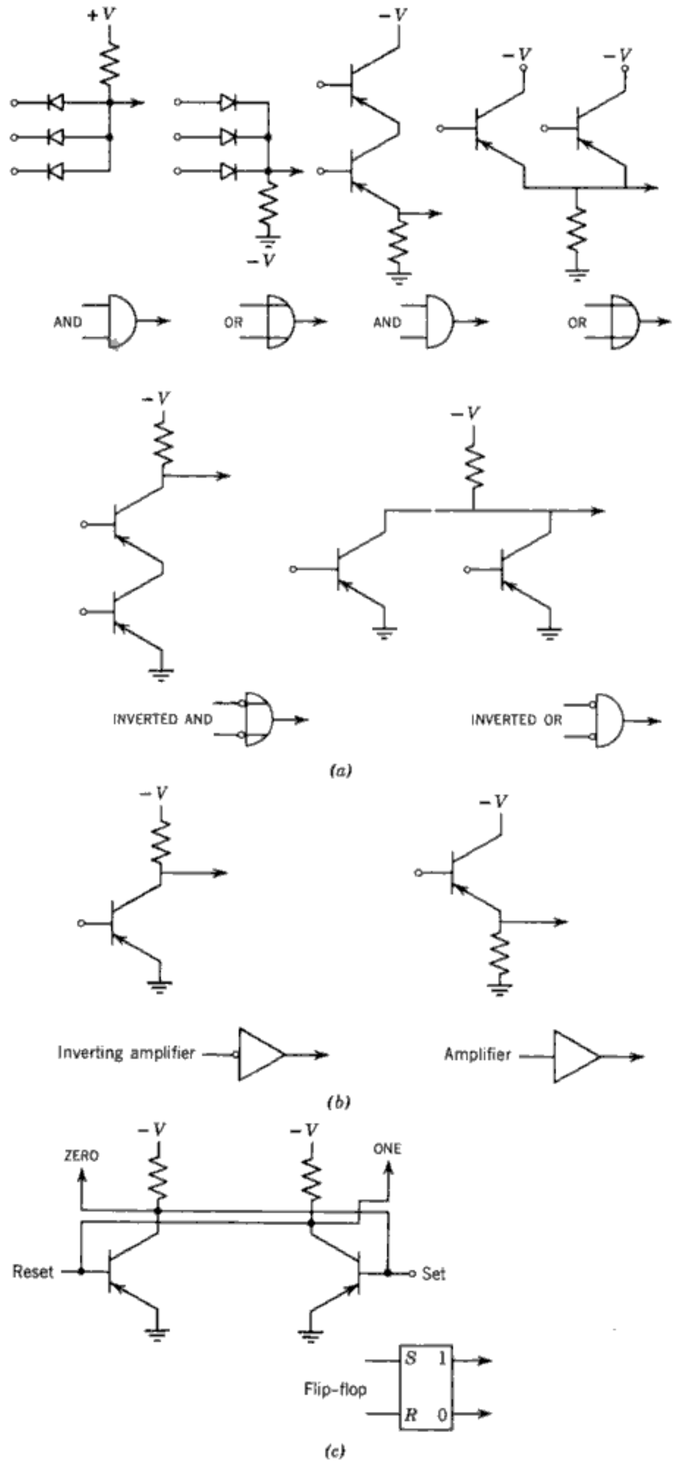


Fig. 101. Semiconductor logical building blocks. (*a*) Switching. (*b*) Amplifying. (*c*) Storage.

reset pulse is produced only when a ZERO is applied to one transistor and a shift pulse to the other transistor of the AND circuit.

The illustration shows *FF*1 storing a ONE and *FF*2 storing a ZERO. Since *FF*2 received its ZERO during the last shift from *FF*1′, *FF*1′ is also shown storing a ZERO. Similarly, the temporary storage cell feeding *FF*1 is storing a ONE. If now a shift-down pulse is applied, the base of both transistors of A_3 are made positive, causing these transistors to conduct. The negative output pulse of the AND circuit sets *FF*1′ to ONE, reversing the polarities shown on the two output lines. When the next positive shift-up pulse is applied, the bases of both transistors of A_5 are made positive, causing these transistors to conduct. The negative output pulse of A_5 sets *FF*2 to ONE. The total result is the shifting of ONE from *FF*1 to *FF*2.

At the same time that bits are shifted from *FF*1 to *FF*1′ and then to *FF*2, other bits are being shifted from other information cells to temporary storage cells and then to information cells to their right.

SUMMARY OF SEMICONDUCTOR LOGICAL BUILDING BLOCKS (Fig. 101)

1. AND and OR switching circuits may be formed of germanium diodes and a load resistor. The diodes in the OR circuit are connected in the opposite direction from diodes in the AND circuit. The supply voltage for the OR circuit is opposite in polarity to the supply voltage of the AND circuit.

2. Two *p-n-p* junction transistors connected in series form an AND circuit. Two transistors connected in parallel form an OR circuit. This is true if the load is between emitter and ground. If the load is in the collector, an inversion occurs to make the series circuit an INVERTED AND and the parallel circuit an INVERTED OR. The *n-p-n* junction transistor produces the same functions in similar circuits except that it handles negative pulses instead of positive pulses.

3. The junction transistor with collector load acts as an inverting amplifier. The emitter-follower transistor acts as an amplifier; it does not increase the amplitude but it does increase the power of the output signal.

4. A transistor flip-flop may be obtained by connecting two *p-n-p* transistors in a circuit very similar to a vacuum-tube flip-flop. By connecting the set and reset points through diodes to a common input point, the circuit is converted to a complementing flip-flop.

5. *Definitions to Remember*

TRANSLATOR—A network to which are applied signals representing information in a certain code, and the output of which are signals representing the same information but in a different code.

DOUBLE-RANK REGISTER—A shift register consisting of two rows of storage cells: one row for actual storage of the bits, the other row for temporary storage of bits while they are being shifted to another stage.

WRITE AMPLIFIER—An amplifier plus the associated circuitry required to convert computer information signals into the form needed for driving a storage device.

READ AMPLIFIER—An amplifier plus the associated circuitry required to convert storage output signals into a form which makes them understood by the remainder of the computer.

CHAPTER 9

Other devices and circuits

The relay, vacuum-tube, magnetic, and semiconductor circuits and devices presented up to this point have been successfully incorporated into practical digital computers. Components operating on principles other than those previously discussed have also been tried. In addition, many industrial laboratories and universities have been for some time searching for smaller, faster, cheaper, more reliable, and easier to maintain computer components. Several of these components and their principles of operation are discussed briefly.

CHARACTERISTICS OF A COMPONENT FOR COMPUTER LOGIC

A component for computer logic may be based on almost any physical phenomenon. But regardless of the phenomenon utilized, to be able to be employed for storage or switching, the component must possess certain general characteristics. It must either delay the occurrence of a physical phenomenon, possess two distinct stable states, or allow one of two levels of a certain phenomenon to take place during any one set of conditions.

An example of the delay type of component previously discussed is the electric delay line. Here a signal in the form of an electric wave is converted into an electromagnetic wave which travels down the line at a definite rate. At the end of the delay line the wave is converted back to its original form. Any other device which can delay any other phenomenon may be useful as a delay element.

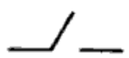
Examples of bistable devices previously used are many: relays, gas tubes, square-loop magnetic cores, and transistors. All these components are similar in the sense that they have two stable states, it is easy to enter one or the other of these states, and it is easy to recognize each state. Any

other component which possesses these qualities may be a useful storage or switching component.

If we review carefully what has been said in previous chapters, we will realize that practically any storage component may be operated as a switching component. This is so because any storage component to be practical must be able to be read, and when it is read an output signal is produced. And a binary switching component is defined as a device which produces one output or another depending upon the conditions present. Thus a square-loop magnetic core is a storage device which may also be used as a switch.

Examples of components which allow only one of two levels of a phenomenon to take place during any one set of conditions are the diode and the resistor. These components may be used only for switching, not for storage because they produce outputs only as long as input signals are applied.

The binary switching device may be represented graphically as



and the binary storage and switching devices may be represented as follows:



The amplifier presents no special problem because many of the components investigated can be made to amplify sufficiently for use in machine logic. Components which are not able to amplify may be used with other standard amplifiers such as those using transistors.

ACOUSTIC DELAY DEVICES

Delay can be produced by several devices which transmit acoustic energy. Among such devices are:

1. The mercury tank.
2. Fused-quartz crystal.
3. Magnetostrictive ferrite or nickel ribbon.

Mercury Tank

The mercury delay-line form of storage was used early in the development of computers. The mercury-tank delay line consists of a tank filled with mercury with a quartz crystal attached at each end. Signals consisting of an alternating wave for a ONE and no wave for a ZERO are applied to the

quartz crystal at one end of the tank. The crystal converts the electric energy into mechanical vibrations which, in turn, cause acoustic waves to travel down the tank. After a definite period of time determined by the length of the tank and other parameters of the mercury, the acoustic vibrations impinge upon the second quartz crystal. The resultant mechanical vibrations cause the crystal to produce an electric wave. By amplifying this energy and feeding it back to the front crystal on the tank, the information may be kept recirculating.

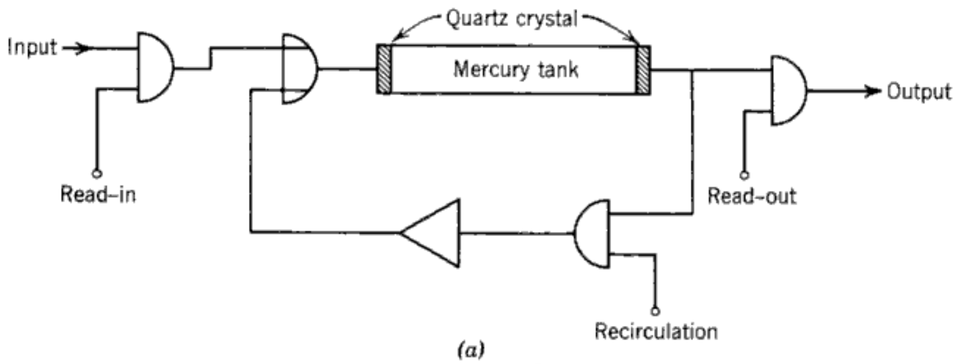


Fig. 102. Acoustic storage. (*a*) Mercury delay-line storage. (*b*) Magnetostriction.

The mercury delay line may be used as a register if operated as indicated in Fig. 102*a*. Information is allowed to enter the register if a read-in signal is applied. As soon as all the bits are in the delay line, the read-in signal is removed and the recirculation signal is applied to keep information recirculating through the amplifier and back to the tank input crystal. To remove information from the register, a read-out signal is applied for the length of time necessary to remove all pulses.

Note that information may be read out and recirculated at the same time. In other words the mercury delay line possesses nondestructive read-out.

Fused Quartz

The fused-quartz delay acts in a manner similar to the operation of the mercury tank. Piezoelectric quartz crystals are mounted at the ends of a

piece of fused quartz. One crystal converts electric waves into acoustic waves, and the other crystal converts the acoustic waves back into electric waves. With the fused quartz the acoustic waves take a definite amount of time to travel from one end to the other. Great delays have been achieved in a small volume by shaping the sides of the fused-quartz crystal so that the wave bounces back and forth many times and in many directions before striking the output crystal.

Magnetostrictive Ferrite or Nickel Ribbon

Delay may be produced by using the magnetostrictive property of certain materials such as ferrite or nickel. A magnetostrictive material is one which converts magnetic energy into a mechanical movement, and vice versa. A magnetostrictive delay line composed of two coils wound on a piece of ferrite or nickel ribbon is shown in Fig. 102*b*. A pulse applied to the input coil produces a magnetic field which causes a mechanical deformation in the rod or ribbon. After a definite time, the mechanical deformation reaching the area within the output coil causes a change in magnetic field and thus induces a voltage in the output coil. This voltage may be amplified and fed back to the input coil to cause recirculation just as is done in the mercury delay line.

BISTABLE DEVICES

Superconductive

Bistable devices may be formed of superconductive elements. A superconductive element is one whose resistance is so low that it cannot be measured with present-day instruments when its temperature is brought below a certain critical point—which usually occurs at a temperature close to absolute zero. An interesting aspect of superconductive elements is that this threshold of superconductivity decreases with the strength of a magnetic field surrounding the element. Therefore, if the element is kept at a constant temperature, it can be made resistive with the application of a magnetic field and superconductive with the removal of the field.

These principles form the basis for a bistable component called the cryotron (Fig. 103). The cryotron consists of a piece of hair-thin (0.009-in. diameter) tantalum wire surrounded by a control winding of even thinner (0.003-in. diameter) niobium wire. In an atmosphere cooled to 4.4° above absolute zero, both niobium and tantalum are superconductive. With no current applied to the control winding, the central niobium wire remains superconductive. With a current pulse applied to the control winding, the

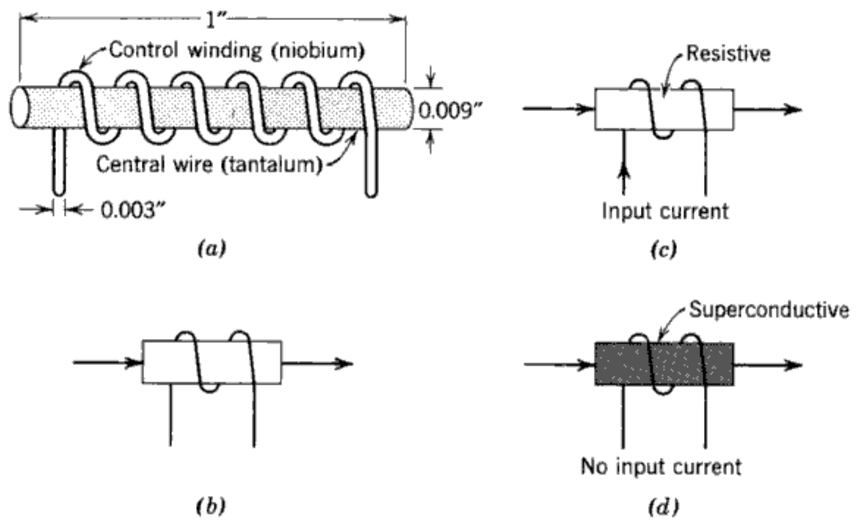


Fig. 103. The cryotron. (*a*) Construction. (*b*) Symbol. (*c*) Store ZERO. (*d*) Store ONE.

magnetic field increases to the point where the central wire becomes resistive. The cryotron has been recently applied to switching and storage circuits in practical computers.

If the presence of current flow indicates ONE and the absence of current flow ZERO, then the simple cryotron is a NOT circuit. A pulse of current representing A applied to the control winding shuts off the flow of current through the central wire to produce $\bar{A}$ (Fig. 104*a*). The cryotron is thus similar in operation to the normally closed relay.

By connecting the central wires of several cryotrons in series (Fig. 104*b*) with a voltage source, an INVERTED OR circuit is produced. A pulse of current applied to any one of the control windings causes the associated central wire to become resistive and to impede the flow of current through the central line.

By connecting the central lines of several cryotrons in parallel with a voltage source (Fig. 104*c*), an INVERTED AND circuit is produced. A pulse of current applied to only one control winding makes its associated central winding resistive; but current flows freely through the central windings of the other cryotrons. Only when pulses are applied to all the control windings is there no current flow through the cryotrons.

Figure 105 shows a full binary adder built of cryotron switches. The eight vertical lines, each connecting the central wires of three cryotrons, are eight AND circuits. The inputs to these AND circuits are either A or $\bar{A}$, B or $\bar{B}$, C or $\bar{C}$. Note that the same signals are applied to several control windings by connecting them in series. Remembering that

$$\overline{A + B + C} = \bar{A} \cdot \bar{B} \cdot \bar{C}$$

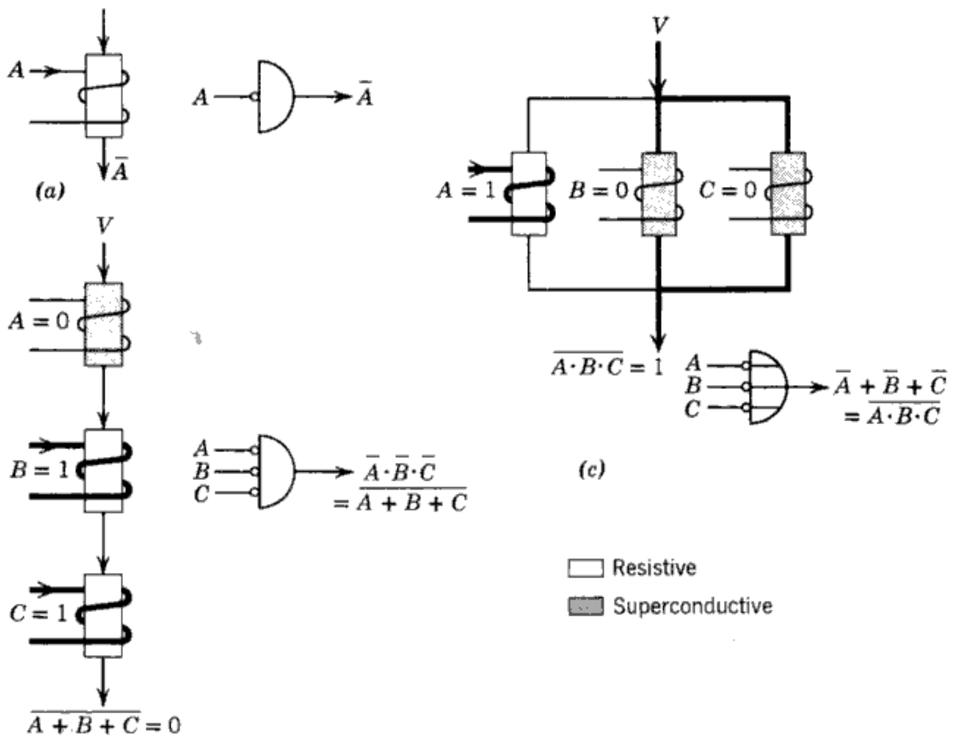


Fig. 104. Cryotron switching circuits. (*a*) NOT. (*b*) INVERTED OR. (*c*) INVERTED AND.

we see that the eight AND circuits produce the following eight terms:

$$\begin{array}{ll} \bar{A} \cdot \bar{B} \cdot \bar{C} & A \cdot B \cdot \bar{C} \\ A \cdot \bar{B} \cdot \bar{C} & A \cdot \bar{B} \cdot C \\ \bar{A} \cdot B \cdot \bar{C} & \bar{A} \cdot B \cdot C \\ \bar{A} \cdot \bar{B} \cdot C & A \cdot B \cdot C \end{array}$$

Each of these signals is then fed to the control winding of a cryotron in the sum circuit and to the control winding of a cryotron in the carry circuit. Each sum and carry circuit consists of two 4-input INVERTED OR circuits, one producing a ONE, the other producing a ZERO.

The operation of this adder can be readily seen by tracing currents through the circuit when $A = B = C = 1$. Under these circumstances, current flows through the three bold, horizontal lines linking the eight INVERTED OR circuits, causing the clear cryotrons to become resistive. Only the right-hand vertical line conducts to produce $A \cdot B \cdot C$. This current flows through the two right-hand control windings in the sum and carry circuits, causing these two cryotrons to become resistive. This means that

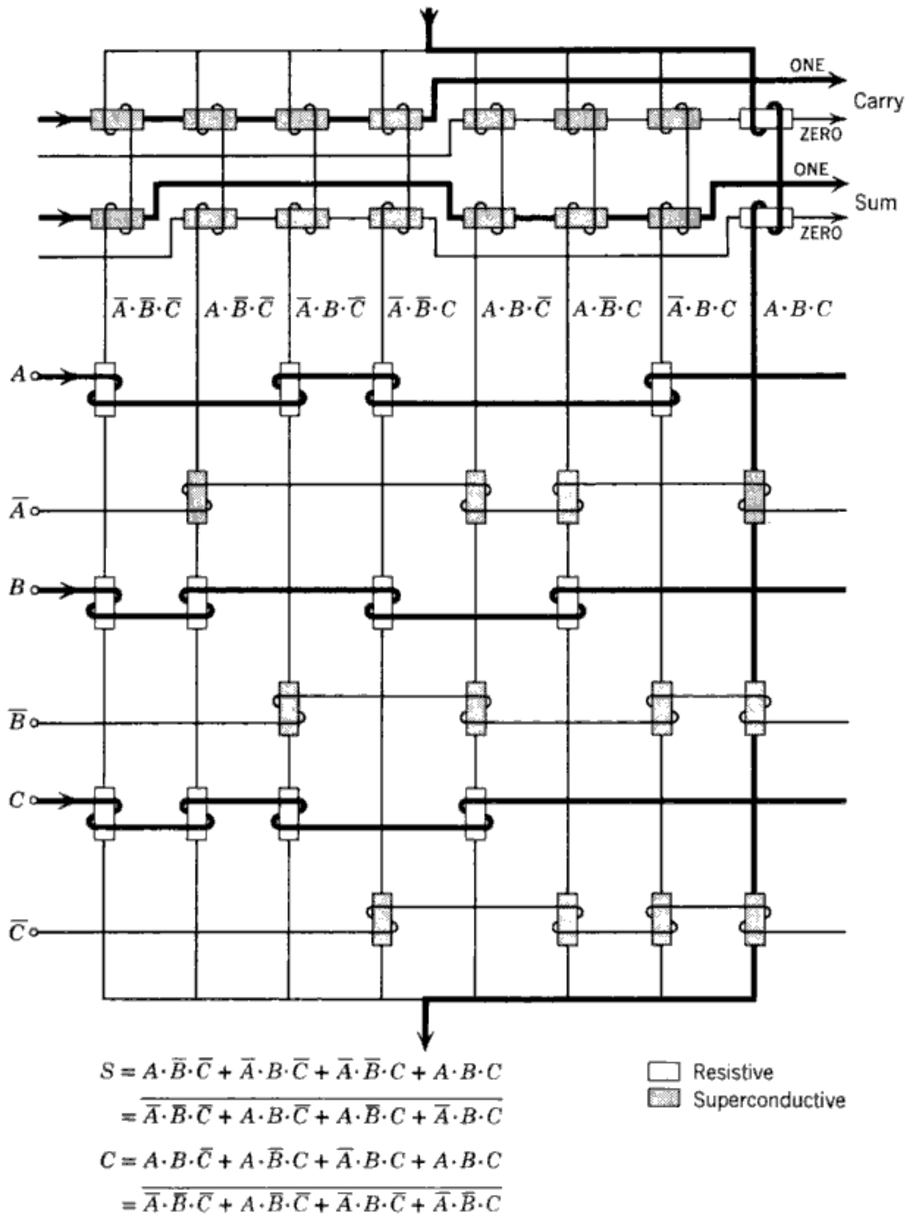


Fig. 105. Cryotron binary adder.

current can flow only through the ONE output lines of the sum and carry circuits.

By connecting the central wire of one cryotron to the control winding of the first, a flip-flop is produced. *CR*1 and *CR*2 in Fig. 106 represent this flip-flop. If current flows through the control winding of *CR*2, this cryotron is resistive and thus prevents current from flowing through the

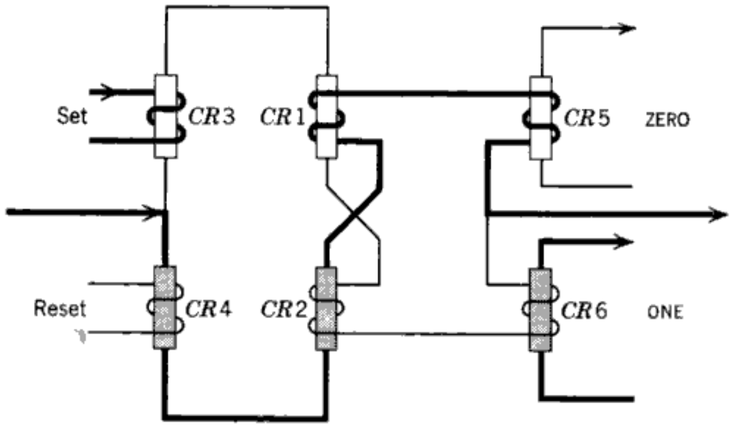


Fig. 106. Cryotron flip-flop in ONE state.

control winding of *CR*1. The central wire of *CR*1 therefore conducts and keeps the control winding of *CR*2 conducting.

A pulse of current fed to the central wire of *CR*1 sends current to the control winding of *CR*2. This tends to make *CR*2 resistive. The more resistive *CR*2 becomes the less current flows through the control winding of *CR*1. This, in turn, makes *CR*1 conduct more. The effect is cumulative. *CR*1 flips to the superconductive state and *CR*2 to the resistive state. A pulse of current fed to the central wire of *CR*2 can flip the circuit back to its original state. When *CR*1 is superconductive and *CR*2 resistive, the flip-flop is storing ZERO. When *CR*2 is superconductive and *CR*1 resistive, the flip-flop is storing ONE.

Cryotrons *CR*3 and *CR*4 are the input circuit to which the set and reset pulses are applied. A set pulse applied to the control winding of *CR*3 makes *CR*3 resistive; therefore current flows through the central wire of *CR*4 to *CR*2 and to the control winding of *CR*1. *CR*2 is thus made superconductive and *CR*1 resistive. A reset pulse applied to the control winding of *CR*4 makes *CR*4 resistive. The current therefore flows through *CR*3 to *CR*1 and to the control winding of *CR*2. *CR*1 is thus superconductive and *CR*2 resistive.

The output circuit consists of *CR*5 and *CR*6. When the flip-flop is storing ONE, *CR*2 is superconductive, sending current to the control winding of *CR*1. This control winding is in series with the *CR*5 control winding, making *CR*5 resistive. No current flows through the ZERO output line. Since no current flows through the *CR*2 and *CR*6 control windings, *CR*6 is superconductive, producing current at the ONE output line. Similarly, when the flip-flop is in the reset state, *CR*6 is resistive and *CR*5 conductive, producing current flow in the ZERO output line and no current flow in the ONE output line.

Several wires may be run through a control winding in a scheme to produce a decoder. One such cryotron decoder capable of decoding three bits into eight outputs is shown in Fig. 107. Each of the eight central wires is fed through a different combination of control windings. Each input bit is fed to one or the other of a pair of control windings depending upon whether the bit is ONE or ZERO. The cryotrons whose control windings receive current become resistive. The result is that the path through one of the lines is entirely superconductive. In the illustration, 101 is fed to the decoder, making the cross-hatched cryotrons superconductive. The result is that line 5 is chosen.

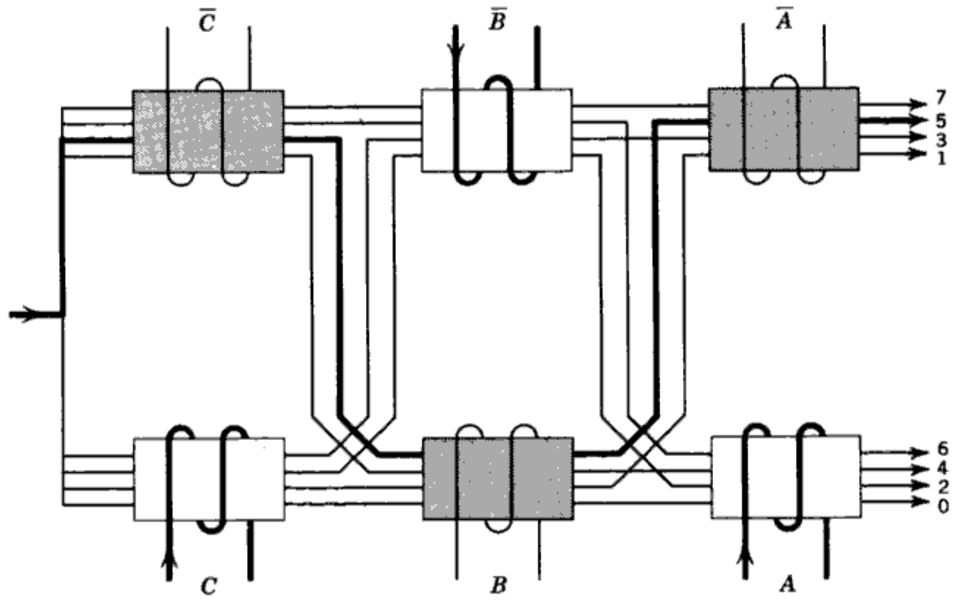


Fig. 107. Cryotron decoder.

The cryotron is capable of amplification because a signal applied to the control winding may cause a greater flow of current through the central wire that is fed from a different power source.

The superconductive component possesses several important advantages. It is tiny in volume, polarity of current flow is unimportant, and it consumes a minute amount of power because of the lack of resistance in superconductive elements. The main drawback to the widespread use of superconductive components has been the need for refrigeration. But recently, a device capable of maintaining the necessary cold temperatures has been developed. The device, called a cryostat, produces and maintains helium liquid for an indefinite period of time. A versatile digital computer using cryotrons has been built within a volume of only one cubic foot,

complete with cryostat. By using a printed-circuit technique to etch onto an insulator base the central conductor and the control winding (as an adjacent conductor), it is expected that the volume may eventually be made small enough so that the computer may be carried about in a vest pocket.

Ferroelectric

The magnetic induction of the core of the ordinary transformer varies with the current applied to one of the windings. If the core is made of

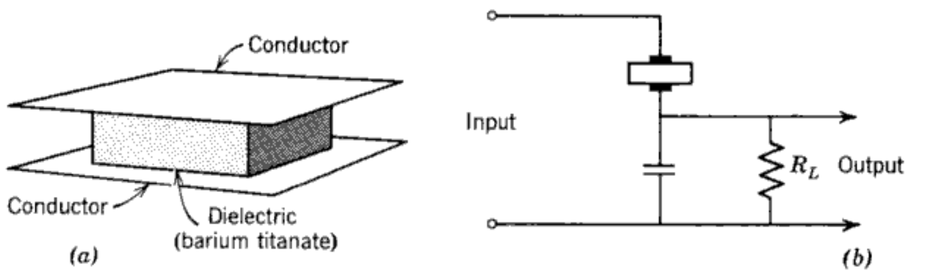


Fig. 108. Ferroelectric storage. (*a*) Ferroelectric capacitor. (*b*) Storage cell. (*c*) Electric hysteresis loop. (*d*) Coincident electric storage.

material such as ferrite, the magnetic-induction-applied-current curve is a square loop. Similarly, the polarization of certain dielectrics sandwiched between two conductors of a capacitor varies with the voltage applied across the plates. If the dielectric is made of a material such as barium titanate, the polarization-applied voltage characteristic is a square loop, as shown in Fig. 108*c*.

Materials such as barium titanate which possess this square electric hysteresis loop are called ferroelectric because of the similarity of these hysteresis loops to those of their magnetic counterparts.

A ferroelectric capacitor and a ferroelectric bistable storage circuit are shown in Figs. 108*a* and *b*. A positive pulse applied to the ferroelectric capacitor brings the electric polarization up to the positive saturation area. When the pulse is removed, the polarization falls slightly to the positive remanence point; it is storing ONE. If a negative pulse of sufficient amplitude is now applied to the ferroelectric capacitor, the capacitor flips states—changes polarity—and brings the electric polarization to the negative saturation region. As soon as the electric pulse is removed, the dielectric falls to the negative remanence point; it is now storing ZERO.

Reading is accomplished by applying a negative pulse across the ferroelectric capacitor. If the ferroelectric was storing a ZERO, the electric polarization varies between *C* and *D*; when the pulse is removed, the polarization falls back to *C*. With such a small change in electric polarization, little current flows in the output, and negligible voltage is produced across the load resistor R_L. If the ferroelectric was storing a ONE, the ferroelectric flips from *B* to *D* and then falls slightly to *C*; in the process, lots of current flows through the load to produce a large output voltage across R_L.

Small areas in a ferroelectric crystal may be polarized electrically independent of the polarization of adjacent areas. This means that many bits may be stored in one large crystal of barium titanate. Figure 108*d* indicates a coincident *electric* storage device capable of storing 49 bits. It consists of a large crystal of barium titanate. Seven strips of metal are attached horizontally across the bottom face of the crystal. Seven other strips of metal are attached vertically across the top face of the crystal. The 49 spots in the crystal which are sandwiched between two conductors represent separate storage cells. When half-voltages—similar to half-currents in the coincident-current arrays—are applied to a vertical and to a horizontal strip, only the one cell where coincidence of voltage occurs receives enough voltage to flip to the ONE state. Read-out is accomplished by flipping the chosen cell to the ZERO state.

Electro-optical

Binary information may be stored on a piece of film in the form of light areas for ONE's and dark areas for ZERO's. The film may be written onto by allowing light to fall on the film through a mask. Those film areas exposed to light become dark and those not exposed remain clear.

Reading may be accomplished by shining light through the film. Beams of light pass through the clear areas but not through the dark areas. Figure 109 shows a simple method for reading the film. It consists of a group of neons, a lens system, the film, and a group of photocells. When one of the neons is lit, the beam of light is directed by the lens system

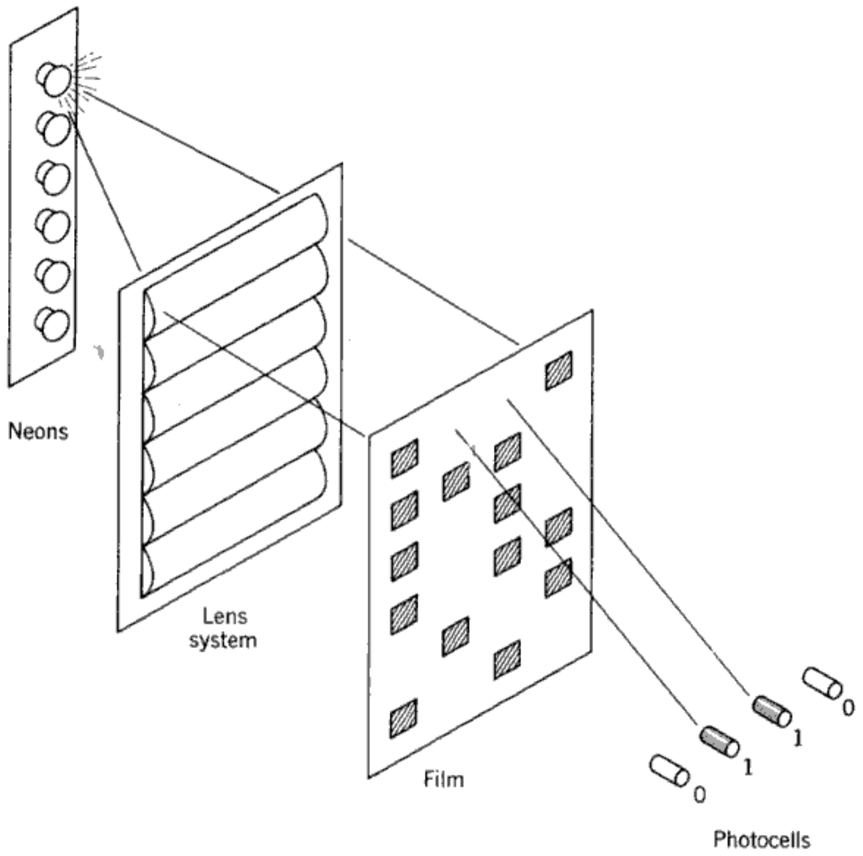


Fig. 109. Photographic storage.

across one of the lines on the film. Light passes through clear areas and is stopped by opaque areas on the film. Photocells associated with clear areas are activated; photocells associated with dark areas are not activated. In the illustration the combination 0110 is being read off the film.

The film may be pasted onto the periphery of a transparent drum with a source of light on the inside to choose the areas to be illumined, and a group of photocells on the outside to pick up the information. The film may also be coated on a glass disk. Here the black and white spots are recorded in annular rings. The optical drum and disk are counterparts to the magnetic drum and disk.

An important advantage of film as a storage device is that both words and coded information may be recorded on the same piece of film. Another important advantage of film is the great density of information storage possible. By making each of the effective storage cells 0.001 in. by 0.003 in., over 300,000 bits may be stored on a 1-in.^{2} piece of film. If we assume there are five bits to a character, this is equivalent to 50,000

characters. With such density the entire contents of the Library of Congress may be recorded on film which would occupy only one drawer of an ordinary filing cabinet—with lots of room to spare. On the other hand, film has the disadvantage of being nonerasable. The only way information may be changed is by exposing new film.

Another electro-optical storage device consists of a tube with a photocathode and a phosphor anode. The photocathode consists of silver-oxygen-caesium; when irradiated with light, it emits electrons. The phosphor anode consists of a manganese-activated zinc-silicate-phosphor screen; when electrons hit its surface, it emits light. The tube operates as

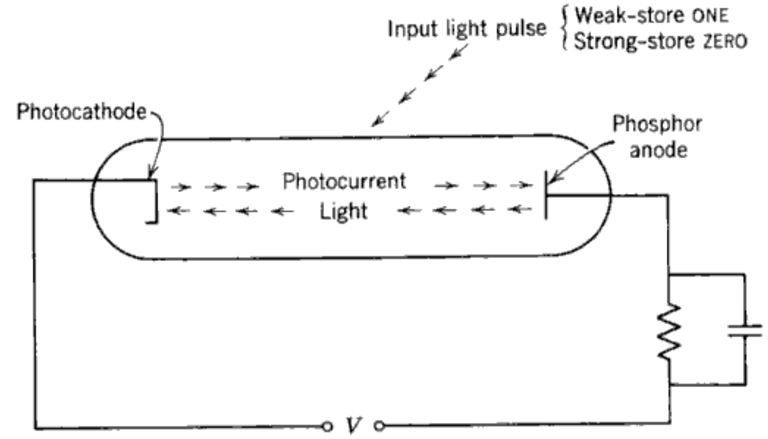


Fig. 110. Electro-optical binary storage cell.

follows. When a pulse of light strikes the photocathode, the photocathode emits electrons which are attracted to the phosphor anode if the anode potential is positive. Electrons striking the anode cause it to emit light which is fed back to the photocathode to keep it emitting electrons. This light emission and conduction is the stable state representing ONE. If the potential between photocathode and phosphor anode is reduced, conduction and light emission stop to put the device in the ZERO state.

Fortunately, the device may be flipped from one state to the other purely by means of applied light pulses if the device is connected as shown in Fig. 110. A weak pulse of light applied to the tube causes the photocathode to emit electrons which, in turn, cause the anode to emit light. This light increases photocathode current which, in turn, increases phosphor-anode light. The circuit stabilizes at the ONE state.

A strong pulse of light applied to the tube flips the device back to the ZERO state. The strong pulse of light causes the phosphor anode to conduct heavily. The heavy current flow produces such a great voltage drop

across the load that the potential between photocathode and phosphor anode falls below the value necessary to maintain conduction. Both conduction and light emission fall.

Another device recently developed is the light amplifier. The light amplifier can increase the energy present in a light signal just as an electronic amplifier can increase the energy in an electric signal. A simplified sketch of a basic light amplifier is shown in Fig. 111. It consists of photoconductive and electroluminescent layers sandwiched between pieces of glass. An alternating voltage is applied across the two layers. Light is

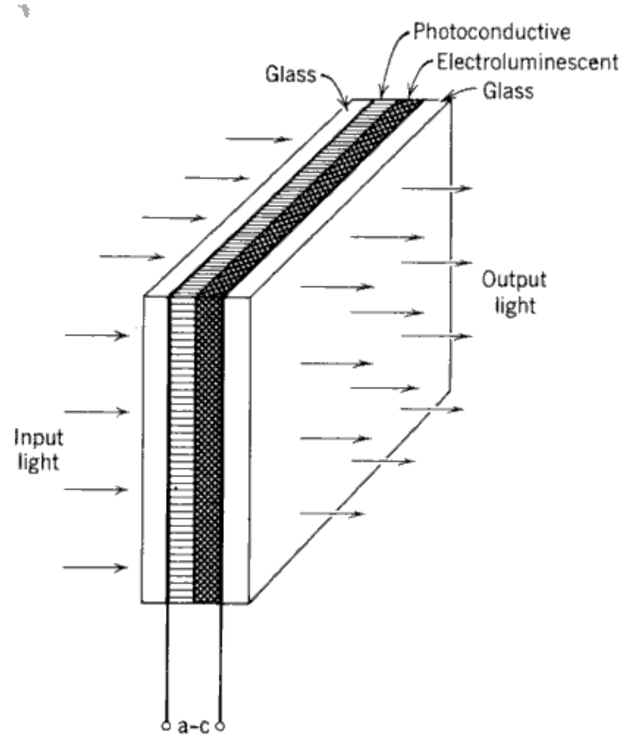


Fig. 111. Light amplifier.

applied to the photoconductive layer. The greater the light, the lower the impedance of the photoconductive layer and, therefore, the greater the alternating current flowing through it and through the electroluminescent layer. The greater the current flowing through the electroluminescent layer, the more light it emits. The device amplifies; the extra power for the amplified light comes from the a-c source.

The photocathode–phosphor-anode tube, photographic storage, and light amplifier may be teamed up in the future to produce a new kind of computer where information flows from point to point, not by means of wires but through the air by means of light waves!

Chemical

A bistable element may be simulated chemically. There are dyes which are dark when subjected to short-wave blue light and colorless when subjected to longer-wave yellow light. The dark state may be called the ONE state and the colorless state may be called the ZERO state. Recently a method has been devised for enclosing tiny bundles of this liquid dye—as small as a millionth of an inch—in capsules which may be painted onto paper.

Writing may be accomplished by scanning the "bubble-painted" paper with light alternating between shorter and longer wavelengths according to the pattern of ONE's and ZERO's we want to store. Reading may be accomplished by scanning the paper with a light-sensitive cell.

Experimentation is also going on to find dyes which react in a bistable manner to infrared and other invisible waves.

Microwave

Transistor logical building blocks have been built which operate 10,000,000 times and more a second. Recently, transistor components have been announced which can operate in the frequency range of hundreds and even thousands of megacycles; and building blocks which can operate even faster are soon to come. The microwave region will be entered and many of the techniques presently used on radar equipment are going to be adapted to digital computers.

It is difficult to visualize a switching device which can turn on and off at a 10,000-megacycle rate. Probably the computer of the future will use a microwave carrier which is flipped toward one frequency to represent a ONE, and toward another frequency to represent a ZERO.

MULTISTABLE DEVICES

Several devices capable of having more than two stable states have been developed. Many of these devices are decade counters operating on magnetic or electrostatic principles. One electrostatic type of beam-deflection decade counter is discussed in the following.

The electrostatic decade counter (Fig. 112) consists of an electron gun, a set of deflection plates, a slotted screen, an anode, and a reset electrode. A highly positive $B+$ is applied through a load resistor (R_L) to the anode. Because of this high potential, negative electrons shot from the electron gun are attracted to the anode. The high positive potential is also applied to the right deflection plate. Therefore the electron beam is deflected toward the extreme right.

As soon as electrons strike the anode, current flows through R_L to

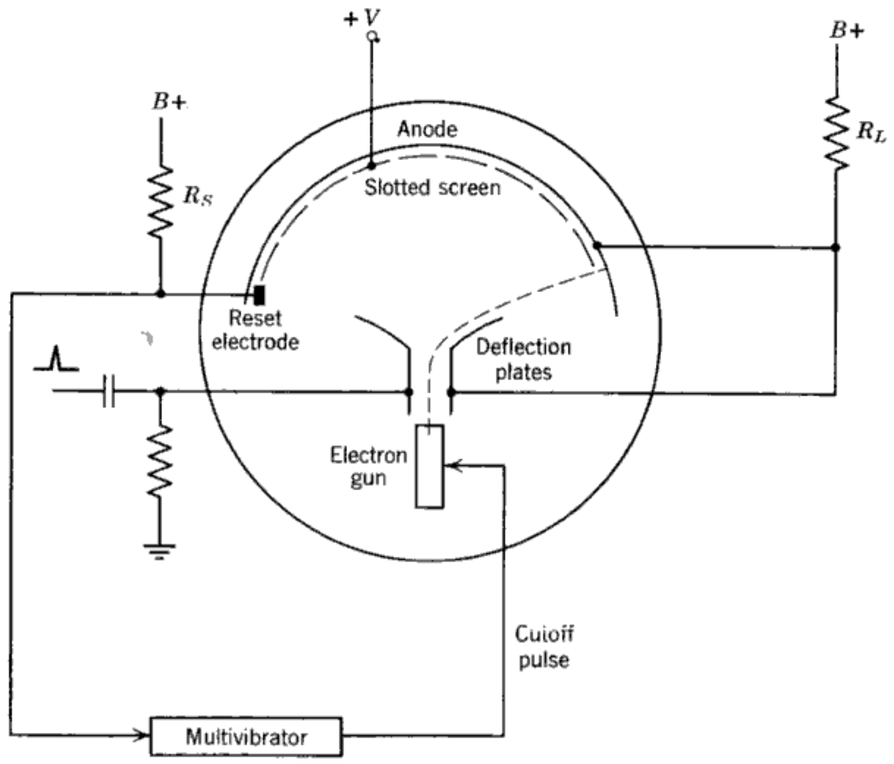


Fig. 112. Electrostatic decade counter.

produce a voltage drop which makes the right deflection plate more negative. The lowered deflection plate potential tends to move the electron beam toward the left until the electrons strike the screen. Since the screen is now diverting some of the current flow, the potential on the deflection plate remains steady and the beam strikes the anode at the point shown in the illustration.

The tube remains in this stable state until a positive pulse is applied to the left deflection plate. The fast positive pulse pulls the electron beam over so it travels through the first right-hand slot in the screen. Again, current flowing through R_L produces a potential on the right deflection plate, tending to move the beam to the right again. So the beam remains at its second stable point.

Successive positive pulses pull the electron beam to succeeding screen slots until the tenth pulse arrives. The tenth pulse causes electrons to strike the reset electrode. Current flowing through resistor R_S produces a negative pulse. This pulse triggers a multivibrator which applies a negative pulse to the control grid of the electron gun to shut off the beam of electrons. There being no anode current, no potential drop is produced across R_L, thus raising the right deflection plate potential to $B+$. Upon

completion of the negative multivibrator pulse, the electron gun turns on, and the beam is deflected to the extreme right in position for a new counting sequence.

SUMMARY

1. A component for computer logic may possess one of the following characteristics:

a. Ability to delay a physical phenomenon.
b. Ability to enter, remain, and be recognized in two or more stable states.
c. Ability to develop at least two different output signals which are distinguishable from each other.

2. In most components discussed in previous chapters, the variable is either voltage, current, or impedance. These variables are changed by modifying one of the following phenomena:

a. Electrostatic field—vacuum tubes, cathode-ray tube.
b. Magnetic field—relay, magnetic core.
c. Semiconductor action—germanium diode, transistor.

3. Delay-type action may be achieved by converting electric waves to acoustic waves which take a definite amount of time to travel through a mercury tank, a piece of fused quartz, or through a ribbon of nickel. The electric waves are converted to acoustic waves, and vice versa, by means of:

a. Piezoelectricity—quartz crystal.
b. Magnetostriction—nickel ribbon.

4. Bistable action may be achieved by the utilization of the following phenomena:

a. *Superconductivity*—A tiny wire of tantalum has almost zero resistance at temperatures close to absolute zero; when a magnetic field is applied, the tantalum becomes resistive. The cryotrons must be stored inside a refrigeration unit which usually takes the form of a helium liquifier.
b. *Ferroelectricity*—A piece of barium titanate crystal may be electrically polarized in the positive direction or in the negative direction. The crystal possesses a square electric hysteresis loop analogous to the square magnetic hysteresis loop of the magnetic core.
c. *Electro-optics*—Dark and light areas on film may represent ONE's and ZERO's which may be read by means of light-photocell arrangements. The film may be placed on drums and disks. Another electro-optical binary storage device consists of a photocathode and phosphor anode in a tube. The photocathode converts light to current, and the phosphor anode converts current to light. A beam of light indicates ONE, and no light indicates ZERO.
d. *Chemistry*—Some dyes are capable of being dark when irradiated with light of one frequency, and colorless when irradiated with light of another frequency.
e. *Microwave transmission*—Bistability may be achieved by building devices which may hold one frequency or another.

5. Decade counters have been built which deflect an electron beam by means of electrostatic or electromagnetic fields through ten stable states. These devices at present are used mainly for counting.

CHAPTER 10

Logical and functional building blocks

From an analysis of the fundamental requirements of machine logic, we have developed logical blocks. The actual forms the logical building blocks take in practice, depend on the types of components from which they are built. Nevertheless, once the circuitry of the logical building blocks is decided, little attention need be paid to the electronics. The logical building blocks themselves may be interconnected to form functional building blocks—blocks which produce the many functions a computer must perform. The complete computer may then be described by relating functional building blocks.

LOGICAL BUILDING BLOCKS

Several logical blocks follow directly from the rules of machine logic. These are shown in Fig. 113.

The flip-flop is a complicated form of the storage cell. It is considered to be a fundamental block because it is commonly used in basic ways throughout a digital computer.

These fundamental logical building blocks may be combined to produce other blocks which are in fairly common usage. Two of these building blocks are the binary comparator and the half-adder.

When converting theoretical logic into practical logic, a modular approach is often followed to facilitate building and maintaining the computer. Components are arranged on plug-in units in such a fashion that with a minimum of wiring almost any logical configuration may be easily obtained. This technique is best if only a small number of different types of plug-in units is needed and if the units may be easily replaced.

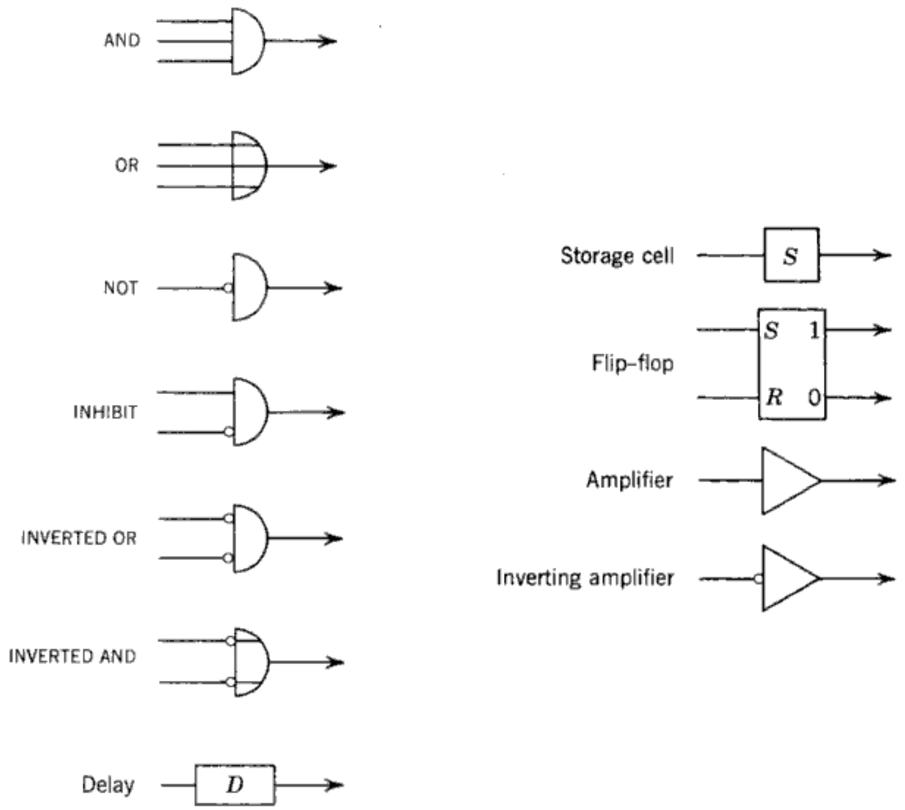


Fig. 113. Fundamental logical building blocks.

An example of such a plug-in unit is shown in Fig. 114. It is a printed-circuit board storing several transistor AND and OR circuits. It consists of a phenolic board upon which are mounted resistors, diodes, and transistors

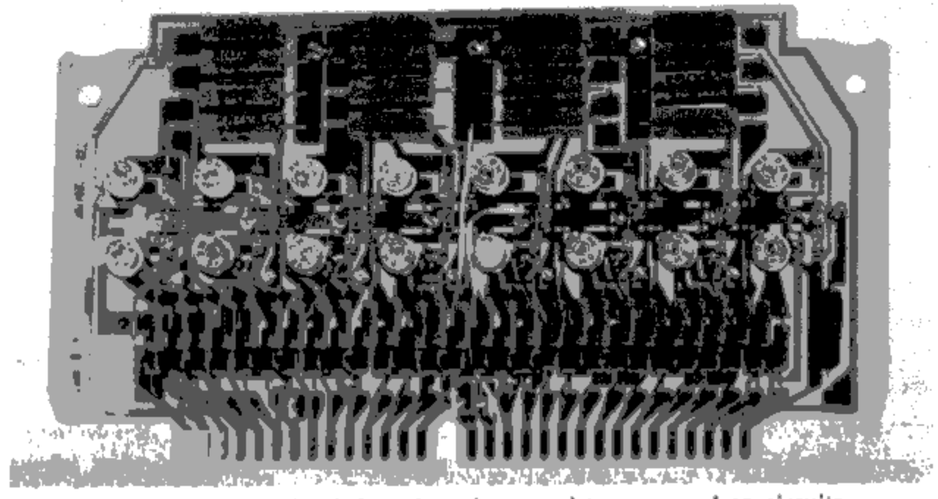


Fig. 114. Printed-circuit board storing transistor AND and OR circuits.

connected together into AND and OR circuits by means of printed wiring. Inputs to and outputs from these circuits are connected by printed wiring to the terminals at the lower end of the board; these terminals mate with an associated connector.

FUNCTIONAL BUILDING BLOCKS

Many different functions are performed by a typical digital computer. All prominent functions may be classed under one of seven categories; and probably most of the unusual functions of special computers may also so be classed. The seven functional categories are as follows:

1. Gates.
2. Registers.
3. Counters.
4. Selectors.
5. Generators.
6. Detectors.
7. Comparators and adders.

Each of the typical functional building blocks is discussed in the following.

Gates

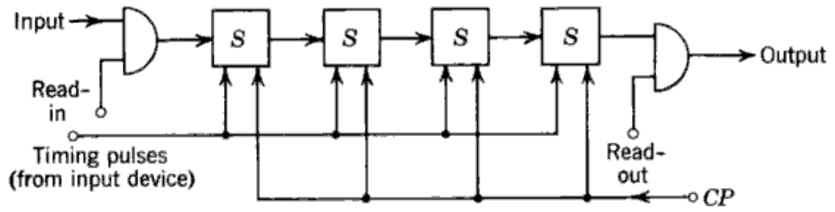
A gate is a simple switching circuit which allows information to flow through if certain signals are present, and which prevents information from flowing through if other signals are present. A gate thus establishes conditions under which numbers, control signals, and other meaningful signals may be applied to other functional building blocks in the computer.

Registers

Registers are temporary storage devices. Information is applied to registers through read-in gates; information is removed from registers through read-out gates. The read-in and read-out gates are usually considered to be part of the register. They are drawn separately, however, in order to show better the information flow.

Registers may be used for one of two purposes: storing words before and after they are combined arithmetically, and as buffers. When used for the former purpose, they are called arithmetic registers. Arithmetic registers must be capable of accepting numbers and applying them at appropriate times to an adder. They must also be capable of shifting, a process which is very important in multiplication, division, and square root.

A buffer is a register which ties together two systems operating at different speeds, or which handles their information bits differently. A speed adjusting buffer is shown in Fig. 115*a*. Bits are fed serially into the



(*a*)

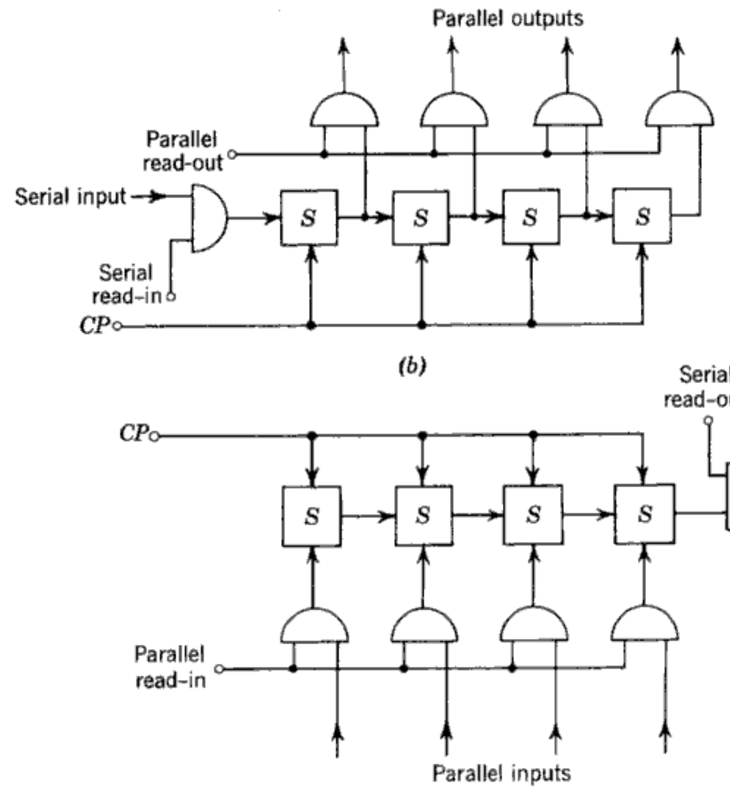


Fig. 115. The buffer register. (*a*) Change-speed-of-information flow. (*b*) Serial-to-parallel flow. (*c*) Parallel-to-serial flow.

register from an outside device whenever a read-in signal is present on the read-in gate. With each bit is applied a timing pulse which shifts the bits from cell to cell in the register. After the information is stored in the register, clock pulses (*CP*) may be applied to shift the bits from cell to cell while the read-out signal allows shifted bits to leave the register.

The input information may be applied at a slow rate, at a fast rate, or even in a random manner. Once the information is present in the register, it can be read out at a different rate as determined by the *CP*.

A buffer register may be used to convert a group of bits following each other serially in time to a group of bits all occurring simultaneously—in parallel. A register which does this is shown in Fig. 115*b*. A serial read-in pulse allows information to be applied to the input bit by bit. With each bit is applied a *CP* to shift the bits from cell to cell in the register. Once the register is full, a parallel read-out pulse is applied to the output gates and all the bits are read out at the same time.

Similarly, the buffer register may be used to convert a group of signals applied in parallel to a group of signals following each other serially in time, as shown in Fig. 115*c*. A parallel read-in pulse applied to all the input gates allows the bits to be read into the respective storage cells. Once the bits are in the register, *CP* may be applied to shift the bits from cell to cell; and a serial read-out pulse allows the bit to leave the register.

Counters

The following types of counters have been discussed:

1. Ring.
2. Modulus.
3. Forward counting.
4. Backward counting.

The ring counter alternately activates one of a series of elements in a closed ring and, thus, is similar in operation to a commutator. It is useful when choosing items in a definite sequence. The modulus counter produces an output pulse after a definite number of input pulses have been applied. It is useful for signifying the end of an operation which always consists of a definite number of steps.

One of the most useful types of counters is the forward-counting binary or decimal counter. In many computers a binary or decimal counter is used as an arithmetic register: the augend is placed in the counter, and then the counter is made to count forward by the amount given by another register. A forward-counting counter is also used to sequence operations in a machine. Ordinarily the counter is cleared to zero before starting an operation. But it need not be. The counter may be set at any number desired, thus modifying the counting as required.

A backward-counting counter is useful for signifying the end of an operation where the number of steps in the operation is variable. The number signifying the number of steps required is fed to the counter. Upon completion of each step, a pulse is applied to the counter to reduce

its reading by one. After the required number of steps is performed, the counter reaches zero and produces a pulse signifying the end of the operation. Such a backward-counting counter is in common use for counting the number of shifts performed by an arithmetic register.

Selectors

A selector is a circuit which chooses a certain location in a storage device, or the performance of a certain function. The selection may occur in space or in time. When performed in space, the selector usually takes

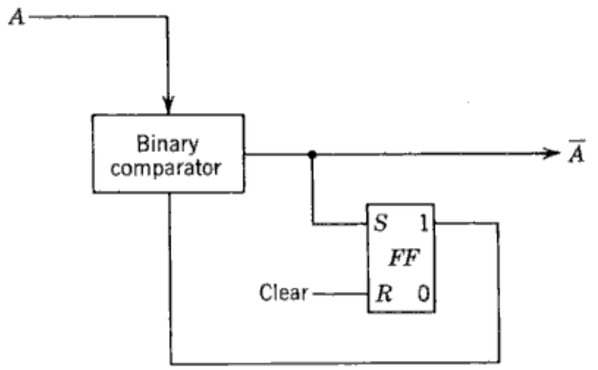


Fig. 116. The 2's complement generator.

the form of a translator: an encoder or decoder. The translator translates an applied code into a signal or group of signals which then perform the selection.

Selection in time may be performed by a sequencer. The sequencer may take the form of a ring counter. Each time another element in the ring flips states, another operation may be begun. Or the sequencer may take the form of a timing-pulse generator where each timing pulse activates another operation. Selectors of both kinds are in common use for sequencing computer operations.

Generators

A generator is a circuit which takes an input or a series of inputs and converts it to a new bit combination which possesses a certain significance. For instance, a complement generator generates the complement of a number applied to it. A parity-check bit generator generates ONE or ZERO depending on whether the sum of the ONE's is even or odd.

An example of a 2's complement generator for the situation where the bits are applied sequentially in time is shown in Fig. 116. This generator produces the 2's complement by following this procedure:

Inspect each bit in turn, starting with the least significant bit. Do not change all zeros to the left of the first significant bit and the first significant bit. Change remaining ONE's to ZERO's and ZERO's to ONE's.

The circuit works as follows. The flip-flop is first cleared to ZERO, thus making the lower input to the binary comparator ZERO. The bits of the number are applied to the other input to the binary comparator. If the bit is ZERO, a ZERO appears at the output. When the first ONE appears, a ONE appears at the output. It also sets the flip-flop to ONE and thus keeps a ONE at the lower input to the comparator. If the following input bit is a ONE, the two ONE's at the comparator produce a ZERO output. If a following bit is a ZERO, the ONE and ZERO combination produces a ONE output. The output is therefore the 2's complement of the input.

Other generators are the clock-pulse generator and the timing-pulse generator. The clock-pulse generator may be an electronic oscillator producing pulses at a steady rate. The timing-pulse generator is different from the clock-pulse generator in that its output pulses are labeled with reference to a point in a time cycle.

It seems that the timing-pulse generator may be considered either a generator or a selector. As a generator its function is general; it produces pulses which may time many different portions of the computer. As a selector it is used for sequencing.

Detectors

The inverse of a generator is a detector. The detector detects bit representations and produces a pulse which causes some operation to be performed. An example of a detector is an end-block detector. Information is stored on magnetic tape in blocks, each block ending with an end-block symbol (*EB*). When reading from the tape into the machine, the *EB* symbol informs the machine that the last symbol of the block has been read.

The *EB* symbol may be one of the forbidden bit combinations. A simple AND circuit may be activated when this combination is present. The resultant output of the AND circuit is the detector output.

Another type of detector is the forbidden-combination detector which is used for checking.

Comparison and Adder Circuits

A comparison circuit is one which compares two or more input signals or groups of signals and produces an output according to the result of the comparison. A binary comparator is a simple example. The binary

comparator compares two bits: if they are alike, it produces a ZERO, if they are unlike it produces a ONE. The principle may be extended to compare two binary numbers consisting of any number of bits.

Another comparison circuit—although it is not usually called that—is the adder. A half-adder compares two bits and produces a sum or carry or any combination of sum and carry according to the results of the comparison. A binary adder does the same job with three bits; it performs a 3-way comparison among augend, addend, and previous carry bits. The principle of the binary adder may be extended to compare numbers containing many bits, and to binary-coded numbers.

Comparators are also used for checking. The same arithmetic operation may be performed by two separate adders. The results are then compared in a comparator. If the two are alike, a signal is produced which allows the computer to continue its work. If the two sums are unlike, the computer stops and a light goes on indicating an arithmetic error.

There may be some overlapping of classes. For instance, the binary comparator is used as a block in the 2's complement generator. But the classification presented here does help categorize functions performed in the computer.

section III

FUNCTIONAL UNITS OF A DIGITAL COMPUTER

In this section the logical and functional building blocks developed in the last section are combined into the five functional units of a typical computer: memory, input, output, arithmetic, and control. Each of these functional units is discussed in detail in a separate chapter, except for input and output. The last two are discussed in one chapter because of their fundamental similarities.

By way of introduction into the functional description of computers and their components, the first chapter of this section describes practical machine language. After all the functional units are described, all the pieces are combined into a specimen computer in the last chapter. This specimen machine is described in enough detail to provide an integrated picture of a digital computer. The last chapter includes a brief discussion of operation, programming, and maintenance of the specimen computer.

CHAPTER 11

Machine language

A digital computer is composed of five functional units, each performing its own function. A machine language is needed for each functional unit to communicate with the others. This same language is required by man to instruct the machine what to do. Man translates his problems into machine language by means of a program of instructions.

LANGUAGE STRUCTURE

The language structure of a digital computer is fairly simple. Bits are encoded into digits or characters which are, in turn, encoded into words. There are two kinds of words: the instruction word and the data word. The instruction word is active. It operates upon the passive data word. A group of instruction words forms a program.

Instruction Word

In general, the instruction word consists of two parts: the command portion and the address portion. The command specifies the operation to be performed by the machine. The address portion specifies the location where the operand or operands are stored.

Most of the operations of the computer are performed in the arithmetic unit. The command portion of the instruction causes the arithmetic unit to perform addition, subtraction, multiplication, division, square root; or to bring operands into the arithmetic unit; or to remove results from the arithmetic unit. Some commands rearrange the information stored in the arithmetic registers. Other commands allow the computer to choose among different sets of instructions according to some criterion such as whether the result of a previous operation is positive or negative.

The other part of the instruction word is the address. Instruction codes differ significantly according to the number of addresses specified

by the instruction. One-, two-, three-, and four-address instructions have been used in digital computers. These instruction codes are described in the following.

One-address instruction. When only one address is specified, the machine executes the given command upon the operand found in the location specified by the address. Since most arithmetic operations require at least two operands—augend and addend, multiplicand and multiplier—the computer must be built so that one operand is brought into the arithmetic unit during one instruction, and the other operand is brought into the arithmetic unit during the following instruction; during the second instruction the arithmetic operation is performed. The sequencing of the instruction words in the program is performed by a program counter. As soon as one instruction is completed, the program counter is advanced by one. The new reading of the program counter specifies the address of the next instruction to be executed. The machine thus automatically chooses instructions stored in successive memory locations.

Two-address instruction. The two-address instruction consists of a command, an address specifying the operand, and an address specifying the location of the next instruction word to be executed. The two-address machine does not need a program counter.

Three-address instruction. The three-address instruction consists of a command and three addresses. The first two addresses specify two operands; the third address specifies the location in memory where the result should be stored. Machines using the three-address instruction need a program counter to sequence instructions in the program.

Four-address instruction. The four-address instruction has a command and four addresses. The first two addresses specify two operands. The third address specifies the location of the result. The fourth address indicates the location of the next instruction to be executed. Machines using the four-address instruction need no program counter.

The above descriptions are true for the majority of computers. However, computers have been built which interpret their address codes differently.

Data Word

The data word is the operand. It may be given in binary or in coded-decimal form. It may be given in integral or fractional form. When integral numbers are specified, the decimal point is assumed to be located to the right of the LSD. When fractional numbers are specified, the decimal point is assumed to be to the left of the MSD.

If the size of the data word is fixed, then there is a definite range of numbers. If there are ten decimal digits to a word in integral form, the number range is between 0 and 9,999,999,999. If the number is given in

fractional form, the range of numbers is between 0 and 1 regardless of the number of digits per data word.

The sign of a data word may be specified by a digit, a 1 representing a minus sign and a 0 representing a positive sign. In an integral number system consisting of plus and minus numbers of ten decimal digits plus sign digit, the number range is −9999999999 to +9999999999. In a fractional number system of equivalent digits, the number range is −.9999999999 to +.9999999999. A computer designating data words in this way is considered to specify them in absolute value plus sign form.

Data words may also be written in complement form. A decimal computer may specify them in 9's complement form: each negative number is replaced by the 9's complement and each positive number is unchanged. The number range of a fractional computer in this system is:

ABSOLUTE VALUE FORM	COMPLEMENT FORM
−.9,999,999,999	= 1.0000000000
0000000000	= 0.0000000000
+.9,999,999,999	= 0.9999999999

Here a 0 to the left of the MSD indicates a positive number and a 1 to the left of the MSD indicates a negative number in complement form.

Computers handling digits with the decimal or binary point assumed to be on the left end or at the right end are called fixed-point computers. Computers using the significant digits multiplied by a base to an exponent are called floating-point computers. A data word in a floating point computer is composed of two parts: the mantissa and the characteristic. The mantissa consists of a group of significant digits with the decimal point to the left of the MSD. The characteristic specifies the exponent. The base is not recorded; it is understood that the base is 10 in a decimal machine, and 2 in a binary machine.

To distinguish between minus and positive exponents, the following characteristic encoding system may be used in decimal machines. An exponent of 0 is encoded as 50. Positive exponents are encoded as numbers above 50; negative exponents are encoded as numbers below 50.

In a 10-digit data word the eight most significant digits may be used for the mantissa and the two least significant digits for the characteristic.

The following are several examples of floating-point representation of data words:

		Floating Point	
Fixed Point	Exponent Form	Mantissa	Characteristic
54000000	$.54 \times 10^{8}$	54000000	58
.0000000843	$.843 \times 10^{-7}$	84300000	43
360743000000	$.360743 \times 10^{12}$	36074300	62

Obviously, the arithmetic unit for the floating-point machine is different from the arithmetic unit required for a fixed-point machine. In the floating-point machine one register is required for handling the mantissa and another register is required for the characteristic.

Word Size

Computer words come in different sizes depending upon the following factors:

1. Complexity of the Instruction.
2. Precision required in data word.

The complexity of the instruction depends, in turn, upon the following:

1. Variety of possible commands.
2. Capacity of memory.
3. Type of instruction code.

The greater the number of possible commands, the greater the number of bits or digits required to encode them. Sixteen commands may be obtained with four bits; 32 commands with five bits. Ten commands may be obtained with one digit; 100 commands with two digits. The greater the capacity of the memory, the greater the number of bits or digits required to encode all the addresses. Four digits are required to provide a different address for each location in a 10,000-word memory. Everything else being equal, it obviously requires many more digits to encode a four-address instruction than a one-address instruction.

Precision determines the size of data word required. The greater the number of significant digits in a data word, the more precise the word. The number 12347563 is obviously more precise than the number 12340000 (if the four right-hand zeros are not significant). The first may be stored as is in a computer possessing 8-digit data words. The four most significant digits of the second may be stored in a computer possessing 4-digit data words, provided the programmer records the fact that the four zeros are excluded.

From the point of view of mechanization, it is best to have the instruction word and data word of the same size. The size of computer words is thus usually arrived at by a compromise of the precision requirements of data words against the complexity of instruction words.

For instance if precision requires six digits be used for the data word, the instruction word may be composed of two command digits and four address digits in a one-address instruction code. This means that 100 commands may be encoded, and that the memory may have a capacity of

10,000 words. If essentially the same machine is required to handle data words of seven digits, the instruction word could be made seven digits long too, with one digit space not being used. This is not too wasteful. But if the same machine must be built to handle data words 12 digits long, it is obviously very wasteful to make the instruction word 12 digits long too. This problem may be neatly solved by combining two 6-digit instructions into one instruction word of 12 digits. The machine must be built to execute each half of the instruction word in turn.

There are occasions when, instead of combining two instructions into one word, we may use two words to store one instruction. Suppose the required precision is obtainable from a 6-digit data word. And suppose the memory has a capacity of 100,000 words, a two-address instruction is wanted, and 50 commands are needed. From this data we arrive at the conclusion that 12 digits are needed to encode an instruction: 2 for the command, 5 for the first address, 5 for the second address. An instruction may be encoded in two 6-digit instruction words as follows:

FIRST WORD		SECOND WORD	
Command (MSD)	First Address	Command (LSD)	Second Address
X	XXXXX	X	XXXXX

Many other variations are possible.

The above discussion applies to machines using data words of fixed length. Greater flexibility may be obtained by making the size of data words variable. This flexibility is obtained at the expense of extra complication in the computer circuitry. In essence, the computer must be built to handle a character at a time. A special mark called the end - of - word symbol, is required to distinguish the end of a word. In some computers the size of the word may be indicated by a portion of the command.

Instruction words are almost always fixed since, for any machine, they always take a more or less standard form.

Size of words is also controlled by machine timing and control considerations as will soon become apparent.

INTERNAL COMMUNICATION

In general, the digital computer is composed of five functional units; memory, input, output, arithmetic, and control. The memory stores instruction and data words, and makes them available when needed. Instruction and data words are fed into the machine through the input unit,

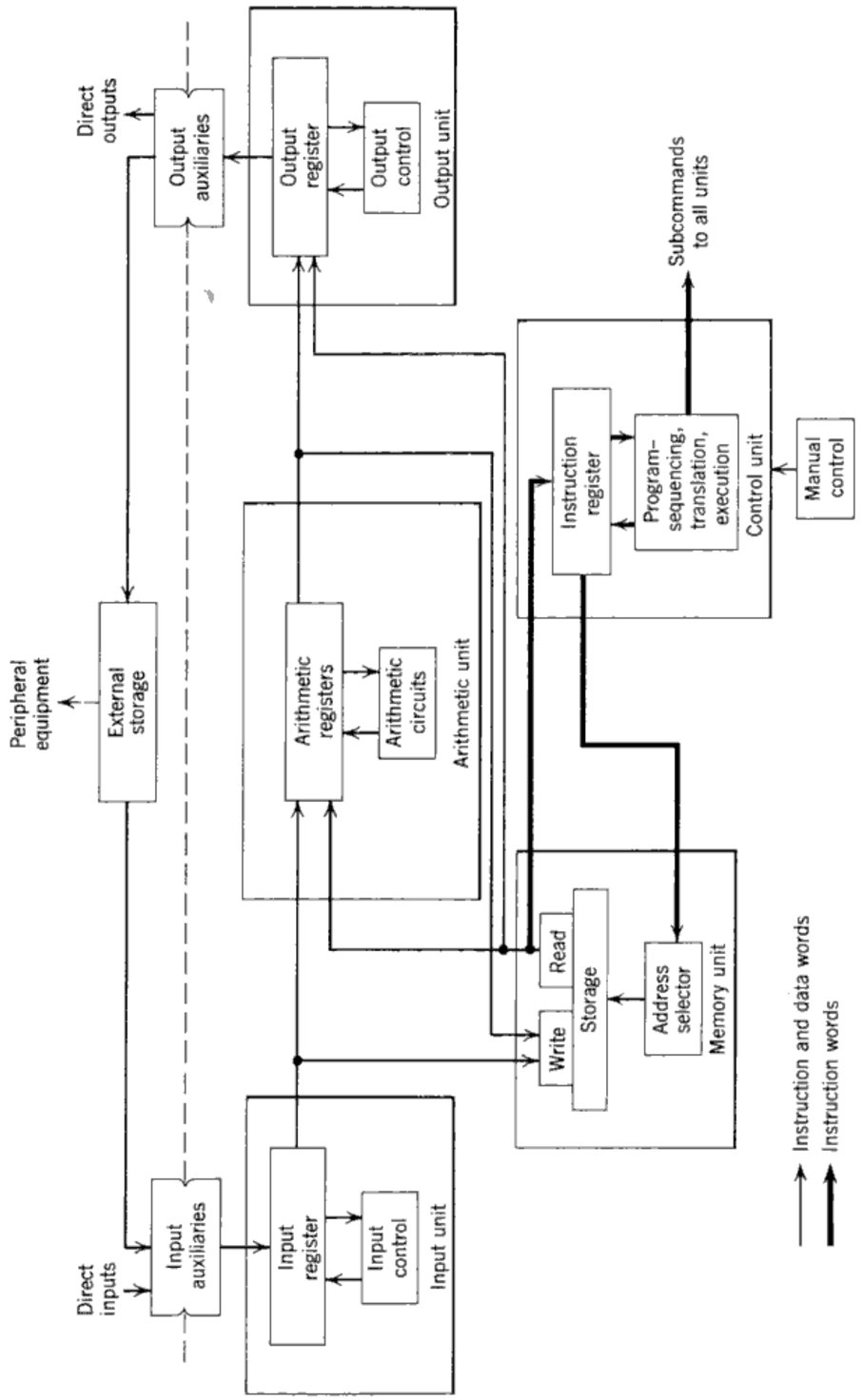


Fig. 117. The functional units of a digital computer.

and out of the machine through the output unit. Most arithmetic and other operations are performed in the arithmetic unit. The control unit sees to it that instructions are properly sequenced and executed.

Each area in the machine does its job and then communicates with another area by means of the machine language.

Computer Information Paths

The five functional units of a digital computer and the information paths connecting these units are shown in Fig. 117. Each of the functional units has a register which receives and transmits information and thus acts as the link between the rest of the machine and the functional unit of which it is a part. This register acts as a buffer. A buffer register is seldom needed in the memory unit since the latter contains plenty of storage. The arithmetic unit contains several registers for storing operands.

Both data and instruction words are loaded into the machine through input auxiliaries, such as paper- or magnetic-tape readers or punched-card readers. The words are fed to the input register. Input control circuits operate on these words before they are fed to other parts of the machine. Data words and instruction words follow slightly different paths.

From the input register the data word is fed to the memory. After a group of data words is stored in memory, each word is called into the arithmetic unit as it is needed for computation; results are fed back to the memory. After a complete batch of results is obtained, this group of data words is fed to the output auxiliaries through the output unit.

In some cases, each data word may be applied directly from the input register to the arithmetic unit for computation. The result of the computation may then be fed either to the memory unit or to the output auxiliary through the output unit.

The instruction word is fed as part of a program, from the input register to the memory. In some machines it is stored in the same section of the memory as the data words. In other machines the two types of words are stored separately.

A portion of the control unit causes a group of instructions in a program to be taken out of the memory in a definite sequence. Each instruction word is fed to the instruction register and then interpreted by the associated control circuits. The command is converted into a group of subcommands —signals which are fed to various gates and circuits throughout the machine to cause the indicated command to be performed. The address portion of the instruction is interpreted, and signals are fed to the memory unit to allow the appropriate operands to leave the memory or to enter it.

Because in many machines, instruction words look like data words, instruction words may be fed to the arithmetic unit and modified arithmetically. As is explained later (Chapter 15), this arithmetic modification enables one instruction to perform the work of several instructions.

Many modifications of this arrangement of the five functional units are possible. Instead of storing the program in the memory, the program may remain outside the computer—on punched cards for instance. In this case the control unit ascertains that instructions are read at the appropriate time by the input auxiliary and fed to the control unit for interpretation. The input and output registers may be combined into one register serving both functions. As a matter of fact, in most cases, the input and output units are so similar in function that they are performed by one unit called the input-output unit. In many cases, one of the arithmetic registers may double as the input-output register.

Methods of Information Transmission

Words may be transmitted from point to point within the machine serially, in parallel, in serial-parallel, or parallel-serial. The method of information transmission depends for the most part on the number system used and the speed of machine operation required.

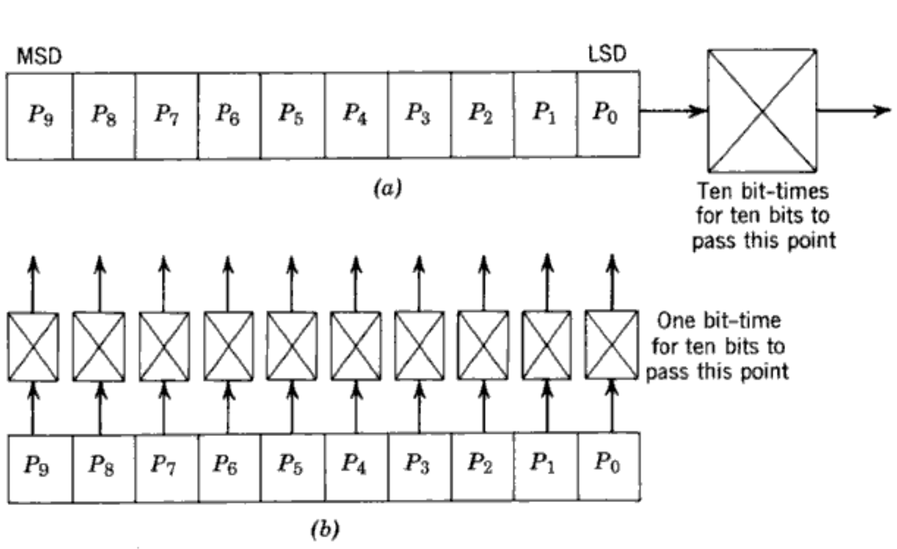


Fig. 118. Methods of transmission of binary words. (*a*) Serial. (*b*) Parallel.

If a binary number system is used, the bits composing the word may flow through the machine in a serial or in a parallel manner. In serial flow, bit follows bit. A basic register consists of a string of binary storage cells

equivalent to the number of bits to a word (Fig. 118*a*). Bits enter the most significant end of the register. The least significant bit is applied first, followed by other bits in order, until the most significant bit is entered. As each bit is entered, the previous bits are shifted within the register.

In parallel flow, all bits of a word are applied simultaneously to all the storage cells in the register (Fig. 118*b*). If a word consists of ten bits P_0 to P_9 ten bit-times t_0 to t_9 are required to shift all the bits into the register when information is transmitted serially. Similarly, ten bit-times are required to shift all the bits out of the register to some other register or to another circuit. In parallel flow, one bit-time is required to bring all ten bits into the register or out of it.

Obviously, parallel information transmission is speedier than serial information transmission. Note, however, that in parallel transmission a separate channel is required for each bit, whereas only one channel is required for serial flow regardless of the number of bits per word.

If a coded-decimal number system with 4-bit digits is used, 40 bits are needed to encode a 10-digit word. In serial flow, 40 bit-times are required to bring a word into or remove it from a register. In parallel flow, 40 channels are required to transmit all the bits of the word. The serial transmission is too slow. The parallel transmission is too costly in required equipment. A good compromise between transmission time and number of channels may be obtained by using serial-parallel or parallel-serial information flow.

In serial-parallel flow (Fig. 119*a*) the digits of a word are transported in serial; and the bits comprising each digit are transported in parallel. Effectively there is a separate channel for each order of bits. To handle the 4-bit, 10-digit word, we may use a serial-parallel register consisting of four subregisters, each subregister handling in serial one bit of each digit. The subregister may be labeled according to the order of bit flowing through it. In the illustration, the top subregister is the first-order subregister and the bottom one is the fourth-order subregister. The information transmission path consists of four channels, and ten bit-times are required to transmit a word.

In parallel-serial flow (Fig. 119*b*), the digits of a word are transmitted in parallel; and the bits comprising the digits are transmitted in series. In this system all the bits of the first order are transmitted during the first bit-time, followed by succeeding orders in succeeding bit-times. To handle the 4-bit, 10-digit word, we may use a parallel-serial register consisting of ten subregisters, each subregister handling in serial four bits. The subregister may be labeled according to the significance of the digit it handles. In the illustration, the top subregister is the least significant subregister and the bottom subregister is the most significant subregister. The information

transmission path consists of ten channels, and four bit-times are required to transmit a word.

The serial-parallel method of information transmission is popular. The parallel-serial method is not in common use. One important disadvantage

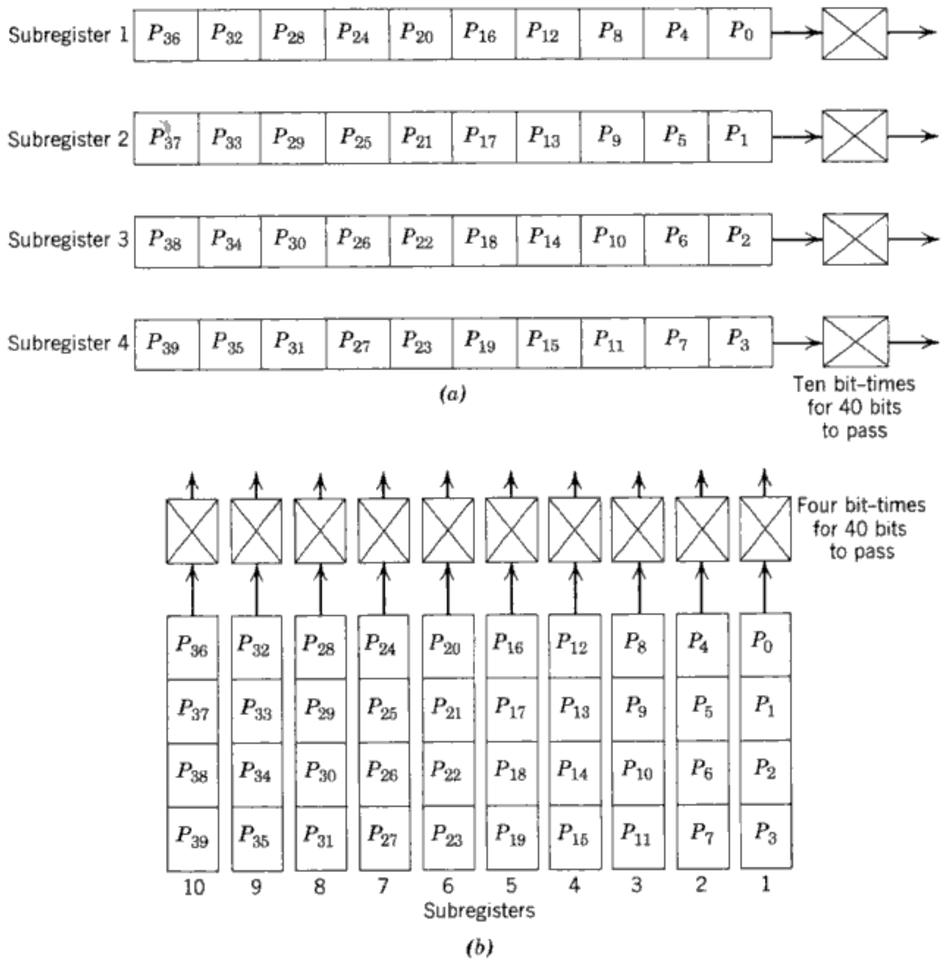


Fig. 119. Methods of transmission of coded-decimal words. (*a*) Serial-parallel. (*b*) Parallel-serial.

of the parallel-serial method is the fact that digits are not transmitted as units.

The sign of a data word possesses special significance since it determines operations to be performed by the arithmetic unit. It is therefore often fed to a special flip-flop. The flip-flop stores ONE to indicate a minus sign, and ZERO to indicate a plus sign. The sign is usually the first bit in a sequence, although it can be transmitted at other times.

Checking

Every computer has definite language rules and definite rules for transmission of information. To increase the reliability of the computer, words are periodically fed through checking circuits.

In a binary computer, a parity-check bit may be attached to one of the words and transmitted as part of the word. The parity-check bit is chosen so that the sum of all the ONE's in the word is odd. At strategic locations in the machine, parity-check circuits look at the word to determine whether the sum of the ONE's is odd or even. If the sum is even, the machine knows a mistake has occurred. It can then stop the machine or merely warn the operator by lighting a light on the control panel.

In a coded-decimal machine, parity bits may be added to each digit combination. In effect, each digit in a word is tested.

Other redundancy checks such as the forbidden-combination check may be used to test the correctness of information flow throughout the machine.

EXTERNAL COMMUNICATIONS

Communication between man and machine may be accomplished automatically or manually. In the automatic method a complete program of instruction words and a list of required data words are prepared beforehand. This information is then loaded into the machine which then executes the instructions in proper sequence. The manual system of communication is accomplished by means of switches and controls on the front panel of the machine. Instruction words and data words may be fed individually into the machine; and instruction sequences may be modified. The machine keeps the operator up to date on events within the computer by means of display lights which indicate the informational content of various registers and whether any arithmetic errors or component malfunctions have occurred.

Types of Instructions

In keeping with the general organization of the computer, several different types of instructions are used by man to communicate with the machine.

Most of the actual arithmetic work of the machine is performed by the arithmetic unit. We inform the machine to do these operations by means of arithmetic instructions which include add, subtract, multiply, square root, and shift right. Some of these instructions such as add and subtract

refer to addresses in memory where operands are located. Other instructions, such as shift right, do not refer to memory. The first type of instruction requires an address portion. The second type does not require an address.

Before operations can be performed by the arithmetic unit, operands must be transferred to it. After the arithmetic unit has performed its work, results must be removed from it. Information must also be fed into and out of other registers in the computer. Man instructs the machine to perform these operations by means of transfer instructions. Transfer instructions may be of two types: read or write. In the first, a word is read out of memory into another register in the machine. In the second, a word is taken from another register and written into the memory. The memory location in both cases is given by the address portion of the instruction.

As long as the computer is performing simple arithmetic operations, transfer and arithmetic instructions are sufficient for communication. But the great forte of the digital computer is that it can follow one sequence of instructions under one set of conditions and another sequence of instructions under another set of conditions. An instruction which asks the computer to do this is called a branch instruction. The branch instruction enables man to communicate with the computer's control unit to modify the sequencing of computer instructions. There are several types of branch instructions among which are the unconditional jump, the conditional jump, and the comparison. The unconditional jump instructs the computer to jump to an instruction word that is out of the regular sequence. The conditional jump instruction orders the computer to jump to a certain instruction word only under certain conditions—if the result of a previous calculation is negative, for instance. A comparison instruction instructs the machine to compare two words and then follow one set of instructions if one result is produced and another set of instructions if another result is produced.

There are instructions which enable man to communicate with input and output auxiliaries. These input-output instructions may be used to control operation of tape units, punched-card equipment, and printers.

To summarize, here is a tentative list (others are described in the following chapters) of the types of computer instructions:

1. *Arithmetic*
   a. Memory.
   b. Nonmemory.
2. *Transfer*
   a. Read—memory to register.
   b. Write—register to memory.

3. *Branch*
   a. Unconditional jump.
   b. Conditional jump.
   c. Comparison.
4. *Input and output.*
5. *Miscellaneous.*

Preparation of Programs

The program of instructions tells the computer what to do. To make sure the computer does exactly what we want, we must detail every step in the process, and we must detail each step by instruction words following the format understood by the machine.

To make each instruction explicit, we must be sure of the address of each required data word, and the address of each instruction word. Therefore, before computation begins, the data words are loaded into one group of addresses in memory, and the instruction words are loaded into another group of addresses.

The basic method of preparation of programs can best be understood by following a simple example. Assume we have a machine with a one-address instruction code of six digits, two digits for the command and four digits for the address as follows:

COMMAND	ADDRESS
XX	XXXX

Part of the command code is as follows:

01 Add the data word stored in the location specified by the address to the word stored in the arithmetic register.
02 Subtract the data word stored in the location specified by the address from the word stored in the arithmetic register.
03 Read the contents of the location specified by the address out of the memory and send it to the arithmetic register (after first clearing it of its previous contents.)
04 Write the contents of the arithmetic register into the memory location specified by the address.
05 Transfer a block of four data words from external storage to the four consecutive locations in memory beginning with the one specified by the address.
06 Transfer to external storage the word stored in the memory location specified by the address.
07 Halt the computer.

Suppose we want to perform the following simple problem:

$$450000 + 380000 - 240000 - 650000$$

First we enter these operands on punched paper tape or whatever external storage means is used. Then the following program is prepared:

Instruction		
Command	Address	Explanation
05	1000	Enter the four data words into the locations specified by the following addresses:
		DATA WORDS ADDRESSES
		450000 1000
		380000 1001
		240000 1002
		650000 1003
03	1000	Read the contents of the location specified by address 1000 (which is 450,000) into the arithmetic register.
01	1001	Add the contents of the location specified by address 1001 to the contents of the arithmetic register; store sum in the arithmetic register.
		450000 (In arithmetic register at start)
		380000 (In address 1001)
		830000 (In arithmetic register at end)
02	1002	Subtract the contents of the location specified by address 1002 from contents of the arithmetic register; store difference in arithmetic register.
		830000 (In arithmetic register at start)
		−240000 (In address 1002)
		590000 (In arithmetic register at end)
02	1003	Subtract the contents of the location specified by address 1003 from the contents of the arithmetic register; store difference in arithmetic register.
		590000 (In arithmetic register at start)
		−650000 (In address 1003)
		−60000 (In arithmetic register at end)
04	1004	Write the result of the calculation into the location specified by address 1004
		−60000 (In address 1004)
06	1004	Transfer the contents of address 1004 to external storage.
07	0000	Halt the computer.

After this program is prepared, it is loaded into the machine so that the instructions in the sequence are in consecutive memory locations starting with the location given by address 0001, as follows:

	Instruction	
Location of Instruction	Command	Address
0001	05	1000
0002	03	1000
0003	01	1001
0004	02	1002
0005	02	1003
0006	04	1004
0007	06	1004
0008	07	0000

If now the start button is pushed, each instruction is performed in the sequence given by the program.

Manual Communication

In addition to the automatic communication possible by means of the program, manual communication between man and machine is also possible. Switches on the control panel are available to start and stop the machine, or to stop the machine under certain circumstances. Switches are available for manually feeding data words or instruction words to various registers in the machine. Switches are available for modifying the instruction sequencing.

On the other side of the communication line the computer informs the operator of what is going on within the machine by means of panel indicators. Indicators show the contents of the various registers at any one time. They show whether errors have occurred and, if they have, of what type.

GENERAL- AND SPECIAL-PURPOSE COMPUTERS

Up to now we have been discussing communication problems relating to the general-purpose computer. The general-purpose computer is a machine capable of performing many different kinds of problems. It is sufficiently flexible so that it can communicate with all parts of the machine. It can

execute a long list of instructions. A program may be prepared at will, stored in the memory, and then executed.

Not so with the special-purpose computer. As its name implies, the special-purpose computer is built to perform only one job or a group of related jobs. Internal communication is restricted. It may possess a shorter instruction repertoire. Its program may be fixed; in other words, the same sequence of events may occur during a repetitive cycle.

The advantage of the general-purpose computer is that it can be programmed to perform many different problems; it is flexible. The advantage of the special-purpose computer is that it can be tailored to do a specific job in the most straightforward way.

Most computers are neither completely general-purpose machines nor completely special-purpose machines. Many so-called general-purpose computers have features which restrict their use to certain general problem areas. The programs of many special-purpose computers may be modified by switches on control panel, by plugboards, or by other means.

In the following chapters most of the discussion concerns itself with general-purpose computers. The general-purpose machine is as a rule more complicated. It should be easy to apply the principles of general-purpose machines to the understanding of special-purpose machines.

DEFINITIONS TO REMEMBER

WORD—A set of bits or digits treated by the computer as a unit.

DATA WORD—A data word consists of numbers or characters to be processed. It may consist of digits and sign.

INTEGRAL COMPUTER—A computer dealing with whole numbers, that is, numbers of 1 or above in magnitude.

FRACTIONAL COMPUTER—A computer dealing with fractional numbers, that is, numbers smaller than 1 in magnitude.

FIXED-POINT COMPUTER—A computer in which the decimal or binary point is assumed to be in a fixed location.

FLOATING-POINT COMPUTER—A computer which carries data words in exponential form. It can deal directly with fractional as well as integral numbers.

FIXED WORD—A word possessing a fixed number of digits or characters.

VARIABLE WORD—A word composed of a variable number of digits or characters.

INSTRUCTION WORD—A word used to inform the machine what operation to perform. It consists of a command and one or several addresses.

COMMAND—The portion of the instruction which tells the machine what to do.

ADDRESS—An expression which designates the location of a character or word in a storage device.

ONE-ADDRESS INSTRUCTION—An instruction with only one address. The address indicates one of the operands upon which the operation is to be performed.

TWO-ADDRESS INSTRUCTION—An instruction with two addresses. One address specifies an operand and the other specifies the address of the next instruction.

THREE-ADDRESS INSTRUCTION—An instruction with three addresses. These addresses specify the two operands and the address of the result.

FOUR-ADDRESS INSTRUCTION—An instruction with four addresses. These addresses specify the two operands, the result, and the address of the instruction to be executed next.

CHANNEL—A path along which information may flow.

SERIAL TRANSMISSION—All bits of a word are transmitted sequentially.

PARALLEL TRANSMISSION—All bits of a word are transmitted simultaneously.

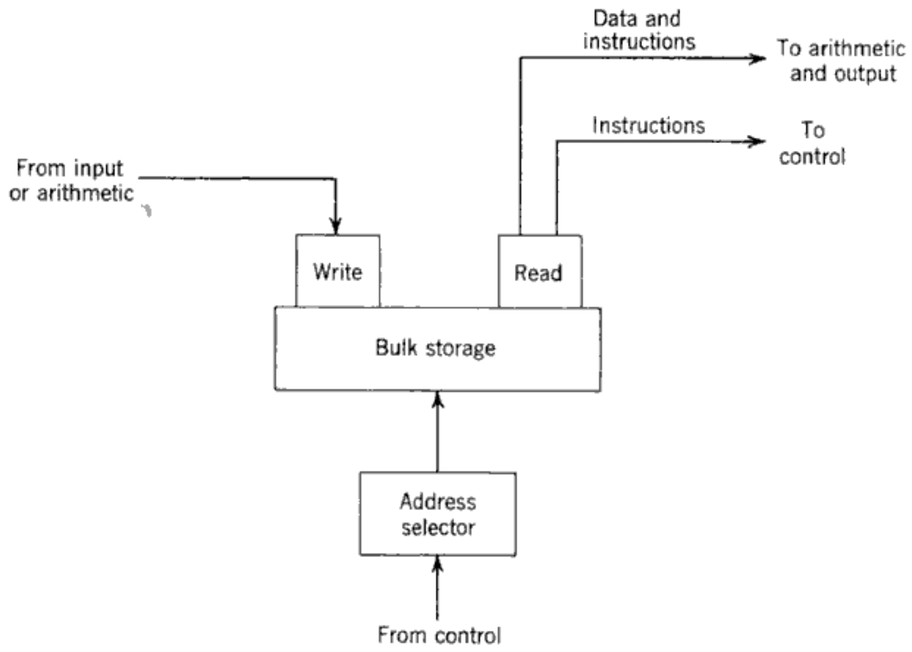
SERIAL-PARALLEL TRANSMISSION—All digits are transmitted serially; but the bits which make up the digits are transmitted in parallel.

PARALLEL-SERIAL—All digits are transmitted in parallel; but the bits of the digits are transmitted serially.

SUBREGISTER—A portion of a register handling a group of bits flowing serially in serial-parallel or in parallel-serial transmission.

GENERAL-PURPOSE COMPUTER—A computer which can be programmed to solve many types of problems.

SPECIAL-PURPOSE COMPUTER—A computer which performs one type of problem or a group of related problems.



CHAPTER 12

Memory unit

The memory unit consists of: a bulk storage which stores data and instruction words; an address selector which chooses the location in memory to be written into or read out of; and write and read circuits which perform the actual writing and reading.

The memory serves two functions. It stores the program of instructions, and it stores the data words upon which the instructions operate. The instructions are read out of memory into the control unit by the part of the control unit concerned with instruction sequencing; the sequencer chooses the appropriate address and activates the read circuits at the correct times.

Aside from the sequencing, words are written into or read out of memory as determined by the instruction words. There are two kinds of instruction words which refer to memory: the write and the read. In the write-type instruction, the address selector is activated to choose the address, and signals are applied to the write circuits to allow the information to be stored. In the read-type instruction, the address selector is activated, and signals are applied to the read circuits to allow information to be read.

WHAT IS MEMORY

Memory consists of three elements: storage, read and write circuits, and address selection. But any storage system requires these three elements. What distinguishes a memory from other types of storage?

To answer this question, let us look at different classes of storage. Storage may be classified according to size. There is bit storage, as exemplified by the flip-flop; there is word storage, as exemplified by the register and counter; and there is multiword storage which is exemplified by systems of bulk storage.

Storage may be classified according to whether it is external or internal. External storage exists outside the computer; it is not a fundamental part of the computer. Internal storage is an integral part of the machine; without it the computer cannot function.

Storage may be classified according to whether it is fixed or erasable. The contents of fixed storage cannot be changed. The contents of erasable storage can be changed—it can be written into.

To be considered memory, storage must be:

1. Bulk storage.
2. Internal.
3. Erasable.

Obviously one bit or one word cannot constitute memory; we need a device which can store many words. Internal storage implies that there is ready and quick communication between memory and the other functional units of the computer. Erasability means 2-way access to words in memory, in as well as out.

To summarize, we may say that memory is accessible bulk storage which may be written into as well as read from.

External storage is used in the input and output systems. Fixed storage may be used in the control unit for storage of nonchangeable programs in special-purpose computers. More about these forms of storage in following chapters.

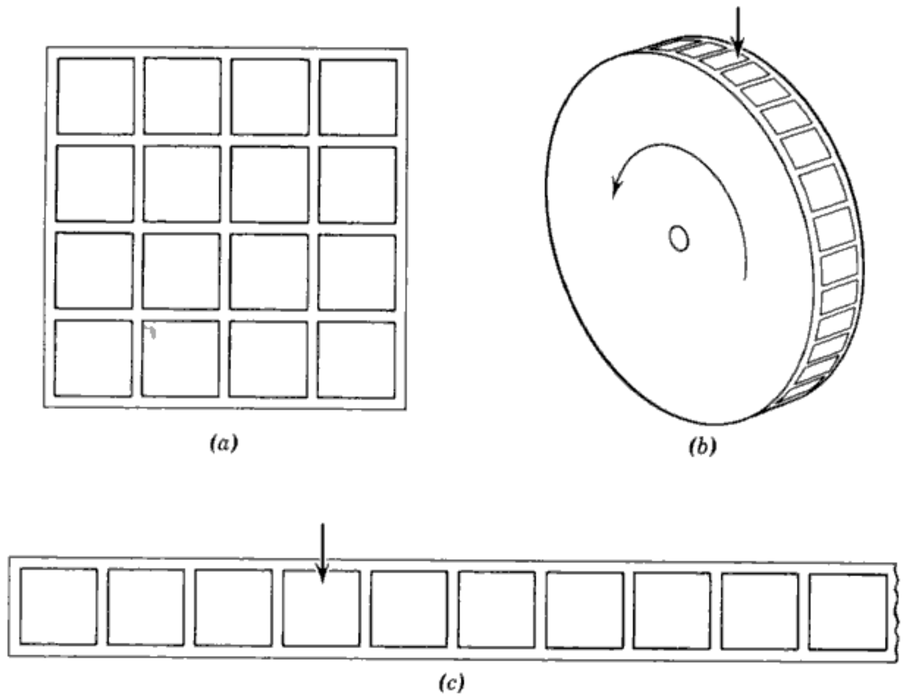


Fig. 120. Types of storage access. (*a*) Coordinate access. (*b*) Cyclic access. (*c*) Sequential access.

STORAGE ACCESS

Types of Access

Storage systems may be categorized according to the method of access to any one of the many storage addresses. There are three types of access as follows:

1. Coordinate.
2. Cyclic.
3. Sequential.

The three types of storage are described pictorially in Fig. 120.

In the coordinate access system the storage cells are arranged in an array in space. To gain access to any one location, the address selector opens a path to one of the storage cells. Coincident-current magnetic-core storage is an example of coordinate access storage.

In the cyclic access system a group of storage cells is arranged in time in such a manner that access may be gained to each cell once during each period of time called a cycle. Storage cells may be rotated past a stationary access device; or an access device may be rotated past a stationary group

of storage cells; or the bits may be circulated through a closed-loop delay circuit. Several storage loops may be operated in parallel. Examples of storage devices using cyclic access are the magnetic drum and the mercury tank.

In the sequential-access system a long group of storage cells moves in time with reference to an access device. Relative movement of storage cells with respect to the access device may be started and stopped at will, and each storage cell may appear only once at the access device during an operation. Examples of storage devices using sequential access are magnetic tape, punched paper tape and punched cards.

Evidently it takes longer to gain access to a storage cell in a sequential storage system than to a storage cell in either the coordinate or the cyclic storage system. Sequential storage is therefore rarely used in memory units; because of its greater storage capacity, it is in common use as external storage in input-output systems. Coordinate and cyclic storage devices, however, make good memories.

Latency, Access Time, and Cycle Time (Fig. 121)

The speed with which words may be written into or read from a storage device depends on the cycle time. The cycle time is defined as the time between successive references to storage.

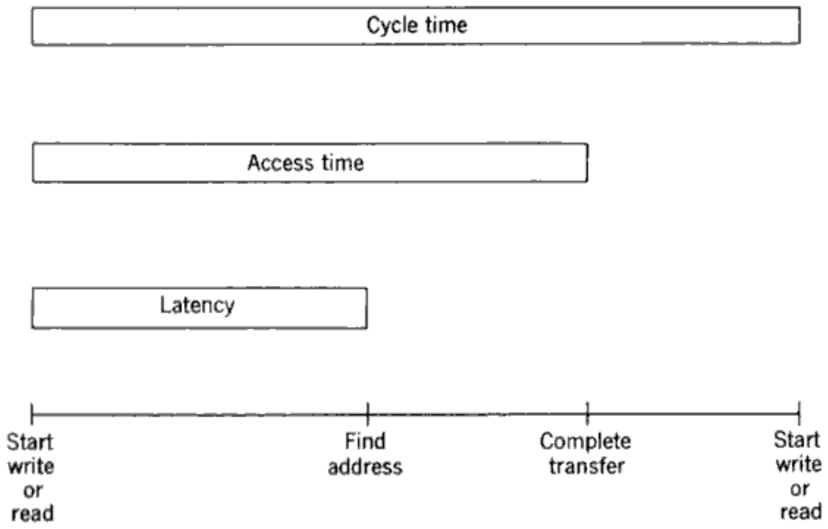


Fig. 121. Latency, access time, and cycle time.

The cycle time consists of the access time plus a waiting time. The access time is the time between arrival of the signal activating the memory and the completion of write-in or read-out. The waiting time is the time the

storage device needs to settle sufficiently so that the next reference to storage may be performed reliably.

The access time consists of the latency plus the transfer time. The latency represents the amount of time it takes to find the chosen address. Latency is a factor in cyclic and sequential storage devices. Latency may be considered to be zero for coordinate systems. The transfer time is the time it takes to transfer a word from storage to some other location, or from some other location to storage.

Random Access

An important class of storage devices possesses the property of random access. Random access storage refers to a device in which it takes just as long to obtain access to one address as it does to obtain access to another address. The coincident-current magnetic-core storage possesses random access. Magnetic tape, on the other hand, does not possess random access, since it takes a short time to gain access to a location at the start of the tape and a much longer time to gain access to a location at the end of the tape. The same is true of the magnetic drum: access time depends on what point in the cycle reference to an address is made.

The main advantage of a random access device is that it allows information to be stored and retrieved quickly. Since in any practical situation it is the relative speed that is important, the definition of random access has been distorted by some to refer to a device which allows access at a rate faster than the information is needed by the equipment served by the storage. For instance, a drum with a cycle time of 20 ms may be considered to be a random access device if it serves a device which cannot operate faster than once every 30 ms.

ADDRESS SELECTION

Address Formation

An addressing scheme is a means for labeling each storage cell in memory. The simplest way to form an address is by giving a number to each storage cell or group of cells in memory. Another way is by labeling each storage location by the number of the column and row in which it is found. In this case the address is effectively divided in two: a row address and a column address.

The row address and column address schemes may be used for coordinate as well as cyclic access devices. In the coordinate access device the row and column both represent addresses in space. In the cyclic access device the

column address is an address in space, whereas the row address is an address in time.

A third coordinate may be used. The third coordinate may represent another dimension in space or another storage device. This general addressing scheme may be extended in many directions.

A special type of address is the relative address. A relative address does not give the location in memory directly; it gives the location in relation to the address in which is stored the instruction being executed. For instance, if the instruction being executed is in column 5, row 10, and the relative column address is given as 2 and the relative row address as 4, the actual address referred to is in column $5 + 2 = 7$ and row $10 + 4 = 14$.

Selection of Space Addresses

A space address may be chosen by a simple decoder (Fig. 122*a*). The bits representing the address portion of the instruction are fed from the instruction register to the address decoder. In the decoder a different output line is activated for a different combination of address bits. The activated output line gates information either into or out of storage, depending upon whether a read or a write operation is called for by the command part of the instruction.

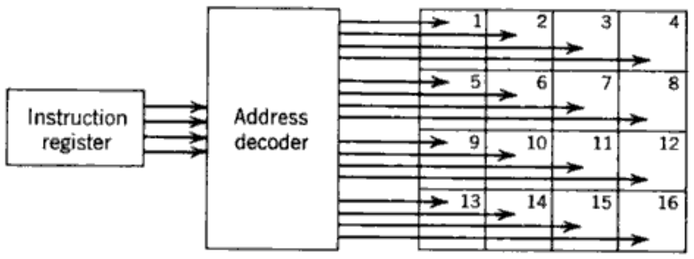
When row and column addresses are used in a coordinate access memory, two address decoders are provided, one for the column address and one for the row address (Fig. 122*b*). The two chosen lines then produce coincidence at one storage location in memory. Coincidence may occur in the read or write direction, depending upon whether a read or write operation is called for by the command. In this scheme the memory itself performs part of the selection.

To select a column address when given in relative form, the relative column address is added to the address of the instruction being executed, and the sum is then decoded.

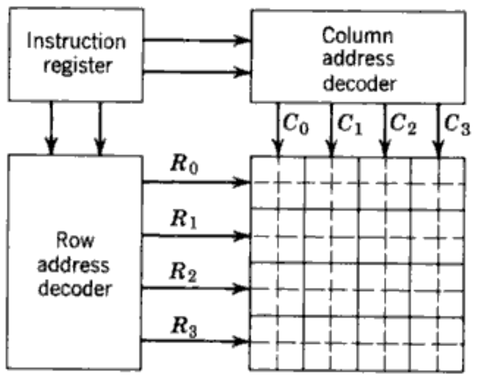
Finding Time Addresses

In a cyclic access device it takes a certain number of word-times for all the rows to pass through the access device. If one row is labeled as reference, all the other rows may be labeled in terms of the number of word-times needed to travel from the reference point to the designated row. This word-time number is the time address.

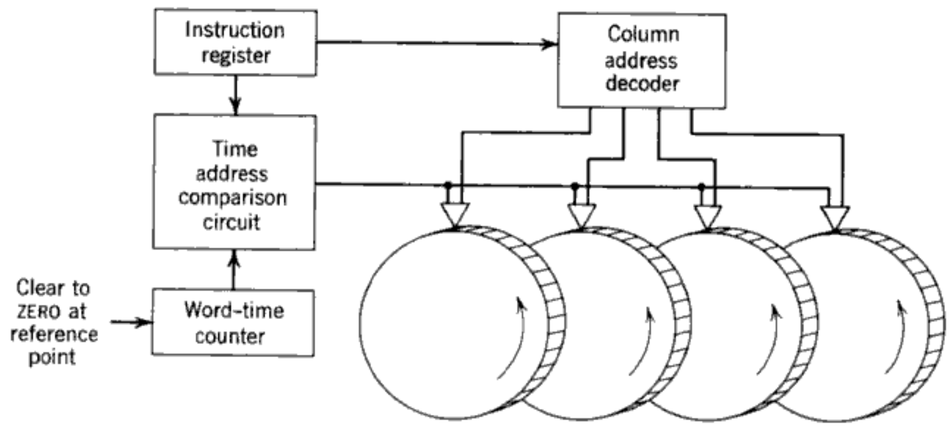
To find a time address, the row address is compared with the output of a clock ticking in synchronism with the storage device. When the two are alike, it signifies that the chosen row is passing through the access device. Figure 122*c* shows a cyclic access memory in which the columns are chosen



(a)



(b)



(c)

Fig. 122. Address selection. (*a*) Space-array address selection. (*b*) Coincident-storage address selection. (*c*) Space-time storage address selection.

by a column address decoder, and the rows are chosen by a time address selector. At the reference point on the memory a pulse is applied to the word-time counter to clear it to 0. The word-time counter thus stores 0 during the period row 0 is passing through the access device. The word-time counter advances to 1 at the same time that row 1 passes through the access device. If the wanted row address is 1, the time-address comparison circuit produces an output pulse which allows writing or reading as determined by the command. If the wanted row address is not 1, the word-time counter is stepped again. This operation continues until comparison is achieved.

To find the actual row when a relative row address is specified, the relative row address is added to the row in which the present instruction is located. The sum is then treated as above.

WRITE AND READ INSTRUCTIONS

Word Transfers

Figure 123 shows the relationships between the memory and the registers to and from which words are transferred. All words may be fed to a bus, and from the bus to all other storage points, or vice versa. Words on the bus enter only registers whose input gates are opened; on the other hand, output gates allow words from registers to be returned to the bus. The address selector acts as a group of input gates for the memory.

If instruction sequencing is disregarded for the moment, instructions may be divided into memory and nonmemory instructions. The nonmemory instruction makes no reference to memory. Examples are square root and shift right. The memory instruction, on the other hand, does make reference to memory.

In the one-address machine, memory instructions may be divided into two classes: write and read. In the write instruction, a word stored in one of the registers and fed to the bus is allowed to be written into the memory by activating the write circuits and by applying the proper signals to the address selector. In the read instruction, one of the words stored in memory, as determined by signals applied to the address selector, is fed to the bus; from the bus the word enters the register whose gates are open. The write or read circuits are activated by subcommands. The address selector is activated by the address.

In the two-address instruction machine, the command and operand address part of the instruction may be classified in a manner similar to the one-address instruction. Naturally, since the second address indicates the

location of the next instruction to be executed, the second address must refer to memory. When this part of the instruction is executed, the address selector is activated by the second address and the appropriate instruction word is fed to the instruction register.

In the three- and four-address instructions, memory instructions may be simultaneously of the write and read types. Two operands as determined

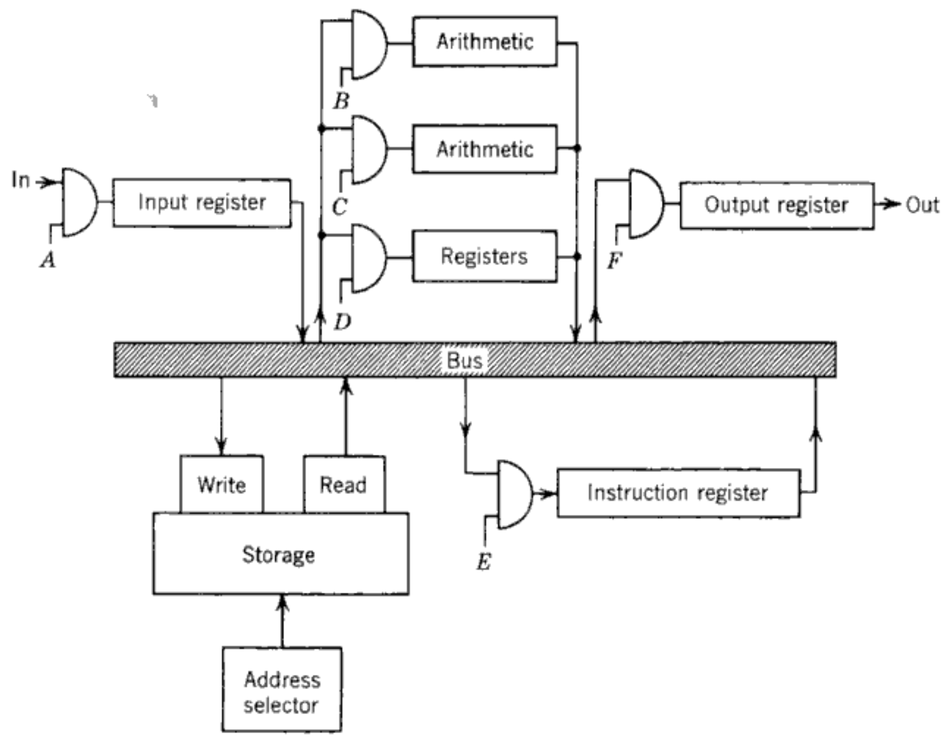


Fig. 123. Information transfer by means of bus.

by the first two addresses are read from memory. After the appropriate operation as determined by the subcommands is performed, the result is written back into the address in memory given by the third address. In this system, writing and reading are performed at different times.

In more complicated computers, separate write and read buses may be used.

Translation

Just as the word-transfer mechanism depends upon the instruction code, the function performed by the write and read circuits depends on the type of storage used.

For instance, when using return-to-ZERO recording in a magnetic drum,

the write circuits convert ONE's and ZERO's into standard-size pulses of positive and negative polarity. These pulses then magnetize locations on the drum to store information. When using cathode-ray-tube storage, the write circuits convert ONE's and ZERO's to voltages which cause the tube to store dots and nondots on the screen. In mercury-tank storage, the write circuits convert ONE's and ZERO's into bursts of acoustic waves or no-waves, which are then fed to the input crystal of the mercury tank.

The read circuits do the reverse of the write circuits. They convert the stored signals into ONE's and ZERO's. In the return-to-ZERO magnetic-drum recording, a ONE is picked up by the head as a positive pulse followed by a negative pulse, and a ZERO as a negative pulse followed by a positive pulse. The read circuits convert the positive pulse followed by a negative pulse to a ONE, and a negative pulse followed by a positive pulse to a ZERO. In the cathode-ray-tube storage, the dot on the screen is converted to a ONE, and the nondot to a ZERO. In the mercury tank, the acoustic signal is detected and converted to a ONE, and the absence of an acoustic wave is converted to a ZERO.

Some storage devices such as the cathode-ray tube cannot maintain their information unless the information is periodically rewritten. To do this, the bits are read at regular intervals by the read circuits and then fed back to the write circuits.

Some storage devices possess destructive read-out. Therefore, each time information is read from storage, it is stored temporarily in a register and then applied to the write circuits. After the rewriting the location is storing the same information as before read-out.

Timing Problems

Coordinate and cyclic access storage devices present different timing problems. Bits may be written into or read out of a coordinate access storage device in parallel. Reference to memory may be made in just one bit-time. Bits are usually written into or read out of a cyclic access device in a serial or serial-parallel manner. Reference to memory requires one word-time. Coordinate access storage is therefore suited to a machine using parallel transmission, and cyclic access storage is suited to a machine using serial or serial-parallel transmission.

Obviously, cyclic access storage slows down the operation of a machine using parallel transmission. However, if a cyclic access storage device is wanted, a buffer register is required in the memory unit. During writing a complete word is transferred from another register to this buffer register; then the address selector and the write circuits are activated to allow serial writing into the cyclic device. During reading, the address selector and

read circuits are activated to allow a word to be read in serial into the buffer register; as soon as the complete word is in the buffer register, the whole word is transferred in parallel to another register.

By the same token, coordinate access storage can be most efficiently used in a parallel machine. However, if it is used in a serial or serial-parallel computer, a buffer register is required in the memory unit. During writing, the bits of a word are transferred in serial from another register to this buffer register; as soon as the complete word is in the buffer, the address selector and write circuits are activated to write the word in parallel into memory. During reading the address selector and read circuits are activated to read a word out of memory in parallel into the buffer register; then the word is transferred serially to another register.

Storage devices have a certain cycle time, which determines how frequently they may be referred to. This cycle time should be compatible with the speed of operation of the arithmetic and control units of the computer.

The problem is more intense with storage devices requiring rewriting because of either fade-out or destructive read-out. In the first case, information is being regenerated at a certain constant rate and words may be written or read only during a certain portion of the cycle. In the second case, the rewrite time must be included to increase the effective cycle time.

Transfer of a Block of Words

In some computers, especially those using magnetic tape for external storage, a complete block of words may be read off the tape and written into memory. Usually the instruction specifying such an operation designates the address in which the first word is to be written; successive words are written into successive addresses.

For a block consisting of a definite number of words, the block transfer is performed as shown in Fig. 124. In addition to the transmission path from tape handler to input register through write circuits to storage, we need an instruction register, a counter, an adder and a comparison circuit. Subcommands activate the tape reader, open the input register gates, operate the write circuits, and clear the counter to 0. The address portion of the instruction is fed to the instruction register.

As the first word is being transferred into the input register, the initial address stored in the instruction register and the contents of the counter (0) are added and then applied to the address selector. The first word is written into this location in memory. When the first word is transferred, an end-of-word signal (*EW*) is fed from the tape handler to step the counter to 1. This 1 is added to the initial address to obtain the next address to be

written into. In this way, each successive word read from tape is written into a successive memory location.

At the same time the counter contents are fed to the adder, they are also fed to the comparison circuit where they are compared with a number

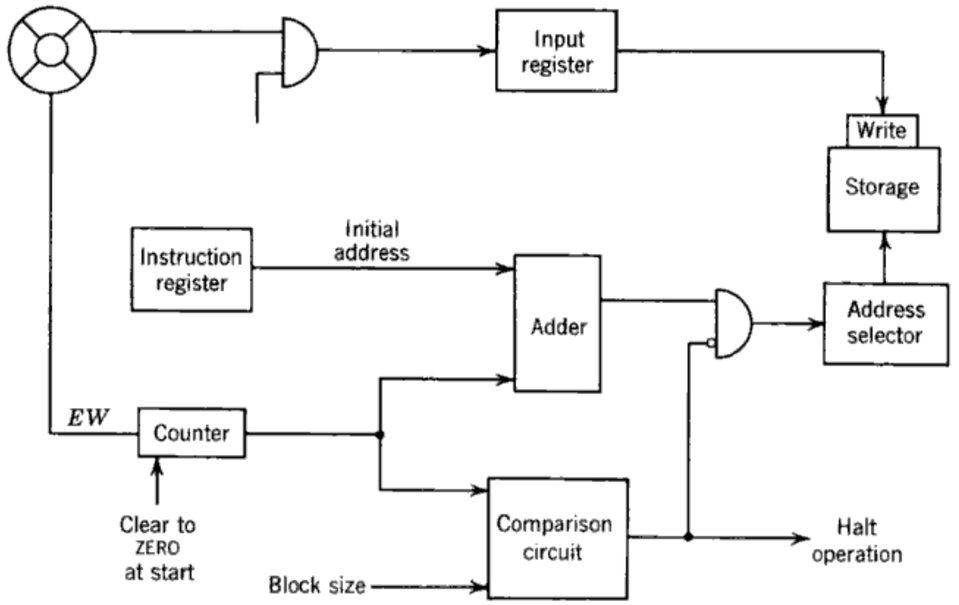


Fig. 124. Block transfer. (*a*) Standard-size block. (*b*) Variable-size block.

representing the size of the block. No output is produced by this comparison circuit until the number in the counter is equivalent to the size of the block. When this happens, the comparison circuit produces a pulse which halts the operation.

If the block does not consist of a definite number of words, it carries an end-of-block signal (*EB*). In this case the comparison circuit is replaced by an *EB* symbol detector. Operation is as described above until

an *EB* symbol arrives. The symbol is detected by the detector which then produces a pulse which halts the operation.

The same type of arrangement may be used to read a block of information out of memory and transfer it through the output register to external storage.

TYPES OF MEMORIES

Characteristics of Storage Components

As we have seen, the construction of the memory depends a great deal on the type of storage component used. Storage components may be classified according to four different criteria:

1. Static or dynamic
2. Erasable or nonerasable
3. Volatile or nonvolatile
4. Destructive read-out or nondestructive read-out.

These classes are defined as follows:

A *static* storage component stores bits by staying in one of several stable states. Examples of static storage components are the magnetic core, diode-capacitor circuit, magnetic drum, magnetic tape, and cathode-ray tube.

A *dynamic* storage component stores information by continuous recirculation of a wave, pulse, or other physical quantity. Examples of dynamic storage components are the mercury delay line, the dynamic flip-flop, and the recirculating register.

In an *erasable* storage component the bits may be removed and other information may be written in the same place. Examples of erasable storage components are the magnetic core, cathode ray tube, diode-capacitor circuit, magnetic drum, and magnetic tape.

In a *nonerasable* storage component the information cannot be erased. Examples are punched paper tape and film. Nonerasable storage is not used for memory. It may be used for external storage or for fixed program storage in special-purpose computers.

A *volatile* storage component loses the stored information with time or if the power is turned off. Examples are the cathode-ray tube and the diode-capacitor circuit.

A *nonvolatile* storage component holds the information even after the power is removed. Examples are the magnetic core, magnetic drum, ferroelectric devices, and film.

A storage component with *destructive read-out* loses the information stored in it when the information is read out. A storage component with *nondestructive read-out* keeps the information stored in it intact even when read out.

Coincident-Current Magnetic-Core Memory

A good example of a coordinate access memory is the simplified coincident-current magnetic-core memory shown in Fig. 125. It consists

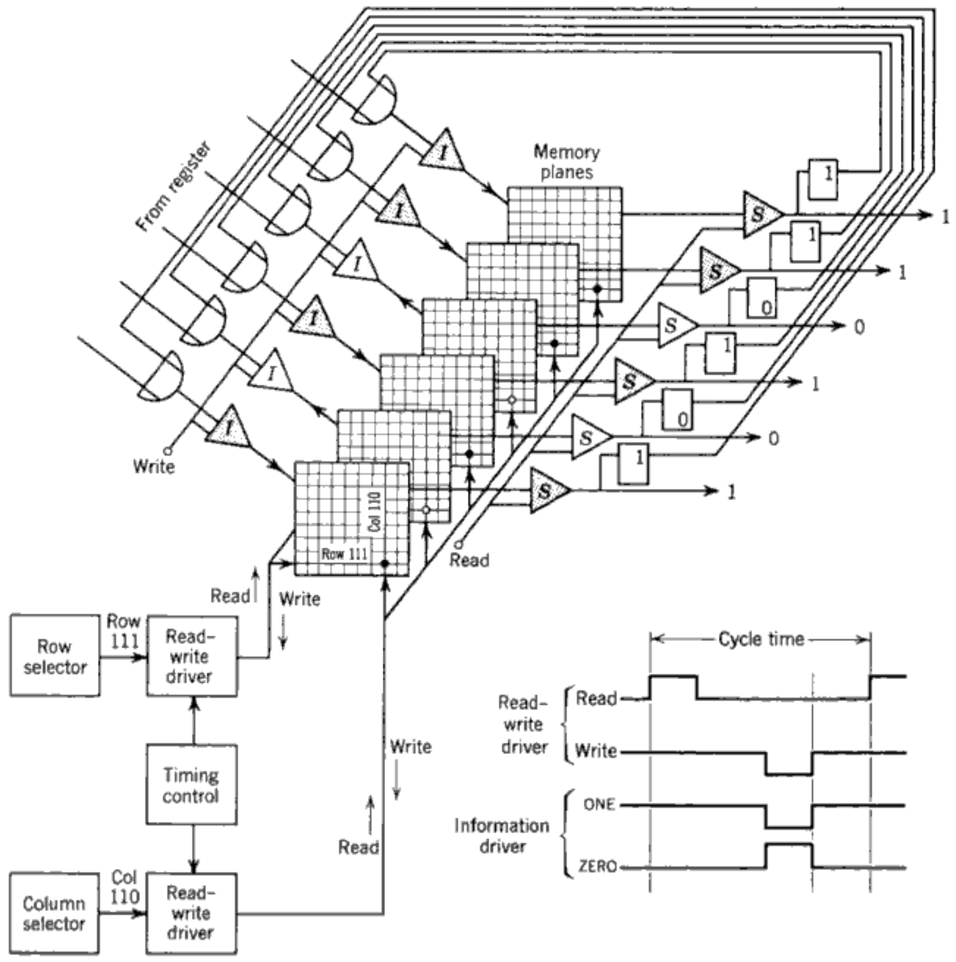


Fig. 125. Coincident-current memory.

of six storage planes, each plane consisting of eight columns and eight rows of cores. Each core has column, row, information and sense windings threading through it according to the pattern shown previously in Fig. 77.

Column address and row address decoders select the core in each plane to be written into or to be read from. Each decoder converts three address bits into eight lines. The drivers supply the necessary power and the proper polarity of current required in the address selection process. Information drivers (I) convert the individual ONE's and ZERO's into pulses which either add to or subtract from the flux in the chosen core of each storage plane. Sense amplifiers (S) pick up voltages during read-out and convert them to ONE's and ZERO's. Flip-flops are used to feed ONE's back to the information drivers to rewrite them into the cores from which they have just been read out.

The basic timing cycle of the memory is determined by the timing control. The timing control produces read-phase pulses followed by write-phase pulses at a definite frequency. The time between the advent of one read-phase pulse and the next is the cycle time.

During the read phase, half-current pulses are applied to the chosen column and row in a direction which would cause the core to flip to the ZERO state. If a read control pulse is present at the sense amplifiers, the voltages induced across the sense windings in each plane are fed through the sense amplifiers to the output. At the same time the ONE's set the associated flip-flops.

During the write phase, half-current pulses are applied to the chosen column and row in a direction which would cause the core to flip to the ONE state. If this is the rewrite portion of a read operation, a write pulse appears at all the information drivers and all 1-bits are rewritten into memory. A write pulse also appears at all the information drivers during an ordinary write instruction. In this case, the entering bits are written into the chosen cores in each storage plane.

The illustration shows a "read 110111", where 110 is the column address and 111 is the row address being performed. The read command is converted to a read pulse followed by a write pulse. The column address 110 is decoded to choose column 6; the row address 111 is decoded to choose row 7. The timing control sends a read-phase pulse through the decoders to cause reading. The word 101011 is stored in the chosen cores. During reading the cores storing ONE's produce output pulses whereas the others do not. The sense amplifiers convert these pulses to standard pulses representing ONE's. Thus at the output is found 101011. The ONE's also set the flip-flops shown. When the timing control switches to the write phase, these ONE's are rewritten into the cores from which they were read out.

The cathode-ray-tube memory works in a similar fashion except that information must be rewritten constantly because of the volatility of the cathode-ray tube.

Magnetic-Drum Memory

An example of a cyclic access memory is the magnetic-drum memory shown in Fig. 126. The drum is divided into five bands, each band consisting of five tracks and storing 20 words. The words are nine 5-bit digits long; but ten digit spaces are provided in each row to allow space

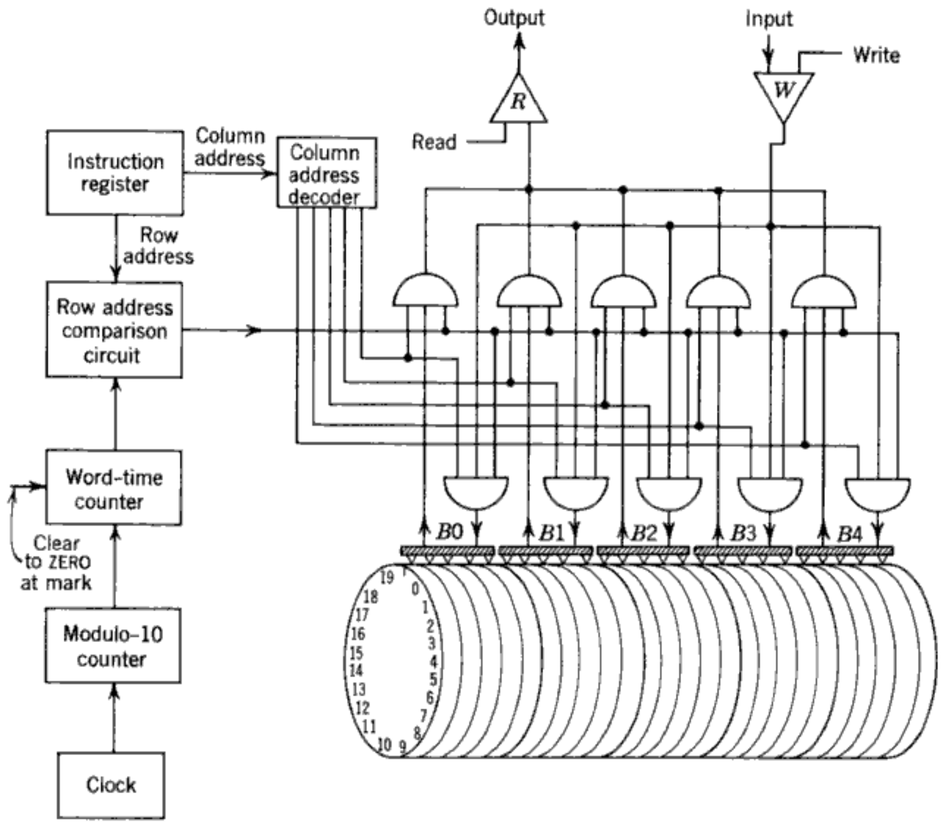


Fig. 126. Magnetic-drum memory.

between words (Fig. 127). It takes 20 word-times to complete one revolution of the drum. In other words, every 200 bit-times the same digit space appears at the heads of any one band.

The column address decoder chooses one of the five bands as determined by the column address digits in the instruction. The row address comparison circuit together with the modulo-10 counter and the word-time counter determines the point in time when the chosen band is opened for writing or reading.

Each symbol for read-write head, gate, and amplifier represents five

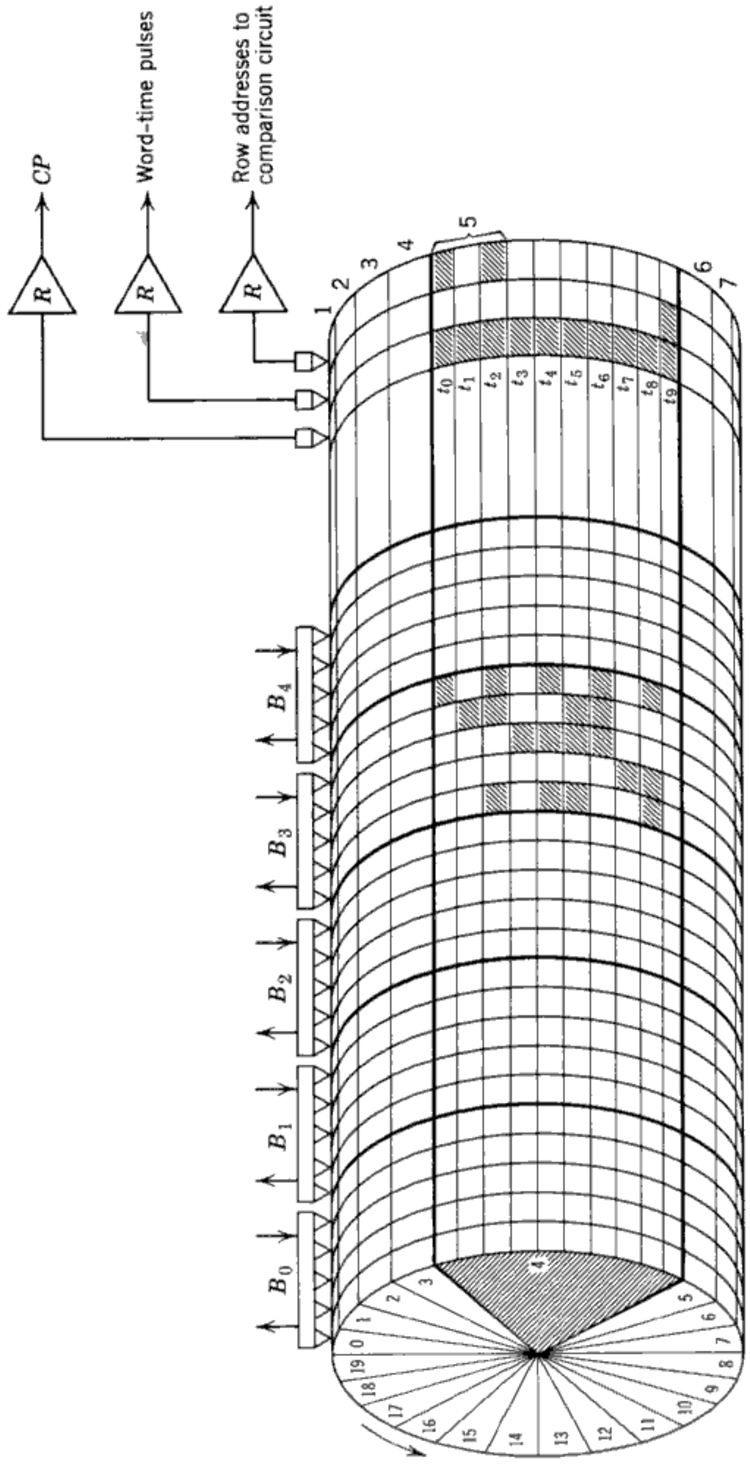


Fig. 127. Magnetic-drum with timing tracks.

such components, one for each track in a band. The read lines are connected to gates (in practice these gates also amplify) and the outputs of all the gates are fed to a group of five read amplifiers. Similarly one group of five write amplifiers feeds the gates to each of the bands.

The memory operates as follows. When the command portion of the instruction calls for a write operation, a write pulse opens the five write amplifiers for a period of a word-time. At the same time, the column address decoder decodes the column address and places an alerting signal to both the read and write gates of the chosen band.

The row address is fed to the row address comparison circuit where it is compared with the output of a word-time counter. The word-time counter is fed from a modulo-10 counter which, in turn, is fed by a clock. Starting with the reference point on the drum, the clock produces one pulse per bit-time in synchronism with the appearance at a head of each successive bit on a drum track. After each group of 10 bits, the modulo-10 counter produces a pulse indicating that a word-time has passed. This pulse steps the word-time counter which thus always stores the word-time representing the row presently under the heads. When the contents of the word-time counter are the same as the contents of the row address portion of the instruction register, the row address comparison circuit produces a signal which opens the chosen band write gate to allow the nine digits of the word to be written into the memory.

Reading is performed in a similar manner except that the read amplifiers are activated by a read subcommand.

It is possible to simplify the row address circuits considerably by properly utilizing extra tracks on the drum. By permanently storing 1-bits in each bit space of a timing track, these pulses can be read as clock pulses. No separate clock circuit is needed. By storing a ONE pulse in each row of another track, these may be read as word-time pulses. No separate modulo-10 counter is needed. It is even possible to do away with the word-time counter by storing the word-time on each row of a separate track. The clock pulses and word-time pulses may be used for control functions too.

A drum constructed in this manner is shown in Fig. 127. Please note that the number actually stored in each word space of the word-time track does not represent the word-time for the row in which it appears but for the following row. The reason for this is that it takes a word-time for the comparison to take place. Since the drum rotates during this word-time, at coincidence the drum is ready for writing into or reading from the following row.

The illustration shows the drum storing 987654321 in band 3, row 4. This word is read by placing into the machine an instruction reading

"read 304" where the first digit represents the band and the next two digits the row. The band address decoder alerts the gates feeding the band-3 heads. Now if row 0 is under the head when band selection occurs, word-time number 1 is fed to the comparison circuit. Since no coincidence occurs, no reading takes place. No coincidence occurs until row 3 passes under the heads. As soon as coincidence occurs the gates of band 3 open to allow the reading of row 4. The gates are open for nine bit-times.

Another cyclic access storage device, the mercury tank, is similar in operation.

SUMMARY

1. The memory is a multiword storage device which is internal and erasable. It is accessible storage.

2. Memory consists of a storage device to store many words; address selection circuits to find word locations in storage; write circuits to convert machine bits into pulses which can activate the storage cells; and read circuits which convert storage output pulses to machine bits.

3. Storage devices are best classified according to method of access as follows:
a. Coordinate.
b. Cyclic.
c. Sequential.

Coordinate and cyclic access storage devices are used for memories. Sequential access storage devices are usually used for external storage as part of the input and output system.

4. Storage cells in the coordinate access system are arranged in space. Only address decoders are needed to locate words. Storage cells in the cyclic access system are located in the time as well as space. The address in time is located by means of counting and comparison circuits.

5. The coordinate access memory is best for use with a machine using parallel transmission of information. The cyclic access memory is best for use in a machine using serial or serial-parallel transmission of information. If a coordinate access memory is used with a serial machine, a buffer register is placed between the other registers and the memory.

6. A complete block of words may be transferred from the input unit to memory. A counter is used to keep tab of the number of words transferred and to see that each successive word is fed to a succeeding address.

7. *Definitions to Remember*

STATIC STORAGE—A storage device which stores bits by staying in one of several stable states.

DYNAMIC STORAGE—A storage device which stores bits by continuous recirculation of a wave or other physical phenomena.

ERASABLE STORAGE—Storage in which bits may be replaced by other bits.

NONERASABLE STORAGE—Storage which does not allow bits to be erased.

VOLATILE STORAGE—Storage which loses its information with time or when the power is turned off.

NONVOLATILE STORAGE—Storage which does not lose its information even when the power is turned off.

DESTRUCTIVE READ-OUT—Storage in which the information is changed when it is read out.

NONDESTRUCTIVE READ-OUT—Storage in which bits remain stored even after they are read out.

CYCLE TIME—The time between references to a storage device.

LATENCY—The time it takes to find an address in a storage device.

ACCESS TIME—The time it takes to complete a transfer to or from a storage device.

RANDOM ACCESS DEVICE—A device in which it takes as long to obtain access to one address as it does to any other address.

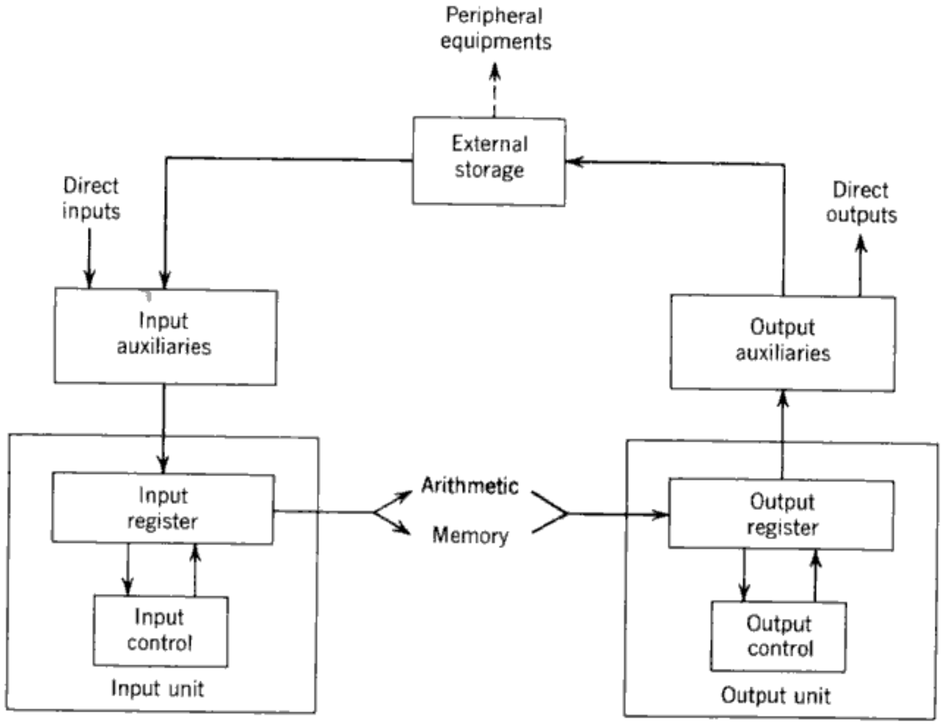
CLOCK—A circuit or some physical arrangement which produces pulses at a steady rate.

WORD-TIME—The time it takes for a word space to pass by a certain point.

COORDINATE ACCESS STORAGE—A storage device whose individual locations are arranged in space.

CYCLIC ACCESS STORAGE—A storage device whose individual locations are arranged in time (as well as in space) in such a manner that access to any one location may be obtained repeatedly at a certain rate.

SEQUENTIAL ACCESS STORAGE—A storage device whose individual locations are arranged in time in such a manner that access to any one location may be normally obtained only once during an operation.



CHAPTER 13

Input-output system

The input-output system consists of three parts:

1. Input and output units.
2. Auxiliary units.
3. Peripheral equipment.

The input and output units are internal functional parts of the computer. The auxiliaries are links between the computer and the outside world. The peripheral equipment is needed to make the operation of the computer more efficient and more meaningful.

The input unit consists of an input register plus associated input control circuits. The input unit receives signals from the control unit whenever a

word or a group of words is to be entered into the machine. The input unit also makes incoming information more palatable to the rest of the computer.

The output unit consists of an output register plus associated output control circuits. The output unit receives signals from the control unit whenever a word or a group of words is to be removed from the machine. The output unit also modifies information so it can be more readily applied to the output auxiliaries.

Auxiliary devices either feed information into the computer or take information out of the computer. These auxiliary devices are of two general types:

Direct—analog as well as digital.
Linked with external storage.

The direct digital device communicates directly between man and machine. It may take the form of a keyboard for manually entering instruction or data words. It may be a printer which prints results of computations onto paper or a group of display lights on the control panel indicating the contents of internal registers.

The analog input auxiliary accepts analog information and converts it to digital form for entry into the input register. The analog output auxiliary receives digital information from the output register and converts it to analog form. The first device is called an analog-to-digital converter; the second is called a digital-to-analog converter.

The input auxiliary linked with digital storage reads information from external digital storage and feeds it into the input register. The digital output auxiliary accepts information from the output register and writes it onto external storage. Examples of such input and output auxiliaries are magnetic tape, punched paper tape, and punched-card readers and writers.

The peripheral equipment is concerned mainly with external digital storage. Recorders are used by operators to place information into external storage. An example is a device which writes words onto magnetic tape when keys of a keyboard are operated. Converters transform information stored in one form to information stored in another form. An example is the punched-card-to-magnetic-tape converter. Communication devices are used to send information to or from external storage. An example is teletype.

INPUT AND OUTPUT UNITS

The input and output units are integral portions of the digital computer. Under direction of the control unit they determine the shape, sequence,

speed, and time of arrival of bits and words as they are transferred between the computer and auxiliary devices.

Input and Output Registers

Input and output registers are essentially buffers between the computer and auxiliary devices. Auxiliary devices operate at much slower and more erratic speeds than do computers. The buffers make them compatible.

Words are transmitted from input auxiliaries to the buffer and then from the buffer to other points in the computer. This operation may be performed in one of two ways: either entirely under control of the program or partially under control of the program. In the first case, the instruction sends a signal to activate the auxiliary unit which transmits information to the buffer; as soon as the buffer is full, its contents are emptied into the address given by the instruction. In the second case, words may be entered into a register either automatically or by hand; an instruction in the program tests the status of the register and, if it is full, transfers its contents to the given address. Similar remarks apply to output auxiliaries.

When entering information from external storage or removing information to be written onto external storage, transfer of more than one word at a time may occur. In this case, one word is transferred to the buffer and then removed; this is followed by a transfer of another word in a similar manner. Some type of counter is needed to monitor this operation.

Most large, general-purpose digital computers possess many auxiliary devices. This would imply that a different buffer is needed for each auxiliary. This is not necessarily so. If the auxiliaries are used at different times, they may all utilize the same buffer.

Input and Output Controls

The input and output control circuits make the input and output registers act as more than mere buffers. They cause the input and output units to perform the following functions:

1. *Synchronization*—Ascertaining that all bits from auxiliaries enter the computer at a time which will not cause interference with computer operation.
2. *Ordering of information*—Reversing the order of bit flow from LSD first to MSD first. The former is the normal way of entering bits by keyboard; the latter is the manner in which the computer handles bits internally. Rearrangement in the opposite sense may occur on output.
3. *Deletion or insertion of information*—Extra bits appearing on a punched card to aid in punched-card processing, for instance, are removed before entry into the computer. Extra bits, an *EB* symbol on magnetic tape, for instance, are inserted before writing the information onto external storage.
4. *Matching auxiliaries to computer*—Prevention of the loss of time which would occur if the fast computer were to wait for the slow auxiliaries.

Each of these functions is discussed in more detail in the following.

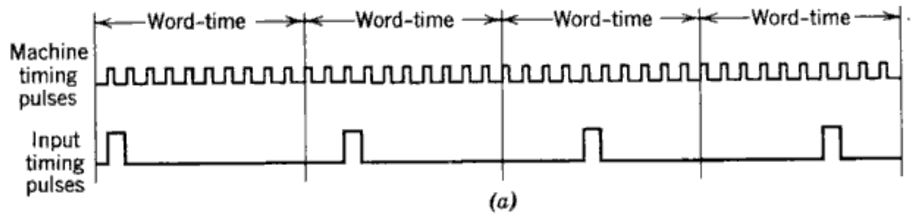
Synchronization

An external reader operates at a certain frequency. While it is reading information bits, it may also be reading input timing pulses at the frequency of reading. These input timing pulses are needed to cause the bits to be shifted into the input register. The input timing pulses and the internal machine timing pulses occur at different frequencies. The input timing pulse may occur at any point in time with reference to a computer word-time, and worse, it may occur at a different point in time during each word-time. As shown in Fig. 128a, the input timing pulse may occur once near machine timing pulse t_0 and the next time near machine timing pulse t_2. The input timing pulse is usually also of different amplitude and width. The purpose of the synchronizer is to produce one sync pulse for each input timing pulse, to make this pulse occur always at the same point in a word-time, and to have it the size of a machine timing pulse. The sync pulse, instead of the input timing pulse, is used to shift the bits into the input register.

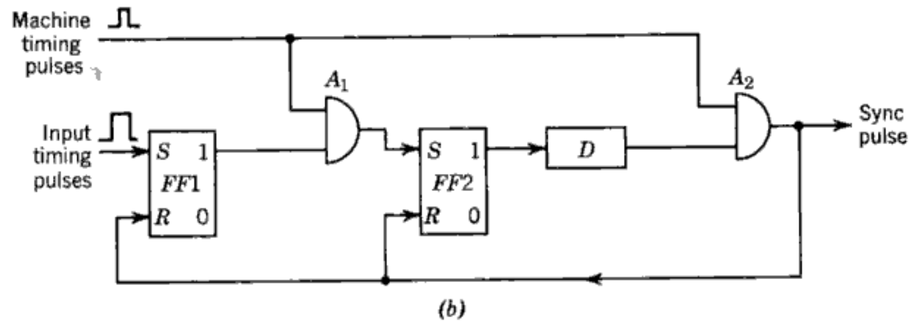
A logical diagram of a synchronizer is shown in Fig. 128*b*. It consists of two flip-flops, a time delay, and two AND circuits. When the input timing pulse arrives during the reading of a digit, it sets *FF*1 to ONE, and this output primes A_1. When the machine timing pulse t_0 arrives, A_1 opens and allows *FF*2 to be set to ONE. One bit-time later, as determined by the delay element, A_2 is primed. When during the next word-time t_0 occurs again, a sync pulse is produced. Thus, slightly over a word-time after the arrival of the input timing pulse, a sync pulse is produced, which occurs always at time t_0, and which has the same amplitude and width as a machine timing pulse. (See Fig. 128*c*.)

When a sync pulse is produced, it is used to reset *FF*1 and *FF*2 to ready the synchronizer for the next input timing pulse.

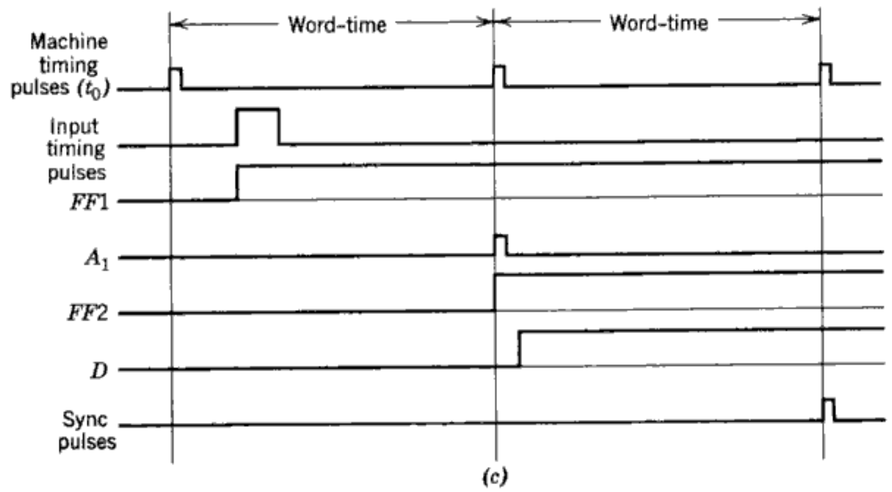
Why do we need *FF*2 and the delay element to delay development of the sync pulse for a word-time? Does not the output of A_1 meet all the requirements of a sync pulse? It does, except that we cannot be positive that one sync pulse would be produced for each input timing pulse. For instance, if the trailing edge of the machine timing pulse should arrive at



(a)



(b)



(c)

Fig. 128. Synchronizer. (*a*) Comparison of clock and input timing pulses. (*b*) Block diagram. (*c*) Timing chart.

A_1 at the same time that the leading edge of the input timing pulse is applied to *FF*1, the output of A_1 may be too small to be effective. No sync pulse would occur.

To be sure of producing one sync pulse for each input timing pulse, we allow the output of A_1 a word-time to flip *FF*2 to ONE. If the A_1 output is large enough, *FF*2 flips; during the following word-time, t_0 produces a sync pulse. If the A_2 output is too small, *FF*2 does not flip during the word-time and no sync pulse is produced. But during the following

word-time *FF*2 does flip and a sync pulse is produced during the word-time after that. Thus at least one and never more than one sync pulse is produced for each input timing pulse.

In this system the input timing pulse must be between two widths: the minimum is somewhat larger than the width of a machine timing pulse, and the maximum is the size of the interval between applied machine timing pulses. In the above example, this interval is a word-time. The interval may be doubled by applying the machine timing pulse during every other word-time.

Ordering the Information

Words are brought into the computer as shown in Fig. 129. As the reader operates, it sends information pulses to the input gate and input timing pulses to the synchronizer. At about the time the information reaches the input register, the sync pulse shifts it in. After each succeeding bit is shifted in, the bits that have entered before are shifted one place to the right. As this shifting operation occurs, the sync pulses are counted. After ten counts representing the ten bits of a word, the counter produces a pulse which sets the read-out flip-flop. The output of this flip-flop allows the computer clock pulses to shift the bits out of the register through the output gate.

The counter in this arrangement counts the number of bits per word and then signals when read-out may occur. The input register itself may be made to perform the counting. Before information enters the input register, a ONE is inserted in the MSD of the register. As each bit enters the register and is shifted right, so is this ONE bit. When the last bit is shifted in, the ONE is shifted out to set a flip-flop indicating that the register is full. (See Fig. 129*b*.)

The above method of reading in information is satisfactory when the digits are arriving with the LSD first and the MSD last. But when digits arrive with the MSD first and the LSD last—this is the order in which humans write—the order in the register must be reversed. This may be accomplished by performing a left shift each time a bit enters the register. The procedure is shown graphically in Fig. 129*c*. The MSD is applied to the most significant end of the register. After a circular left shift (considering the register as a recirculating device), this bit appears in the LSD. The second bit is fed to the MSD, and after a left-shift operation, it appears in the LSD. At the same time, the previous bit appears in the next to least significant position. This procedure continues until the last bit enters the MSD and is shifted to the LSD, and all the other bits are arranged in the proper order in the register.

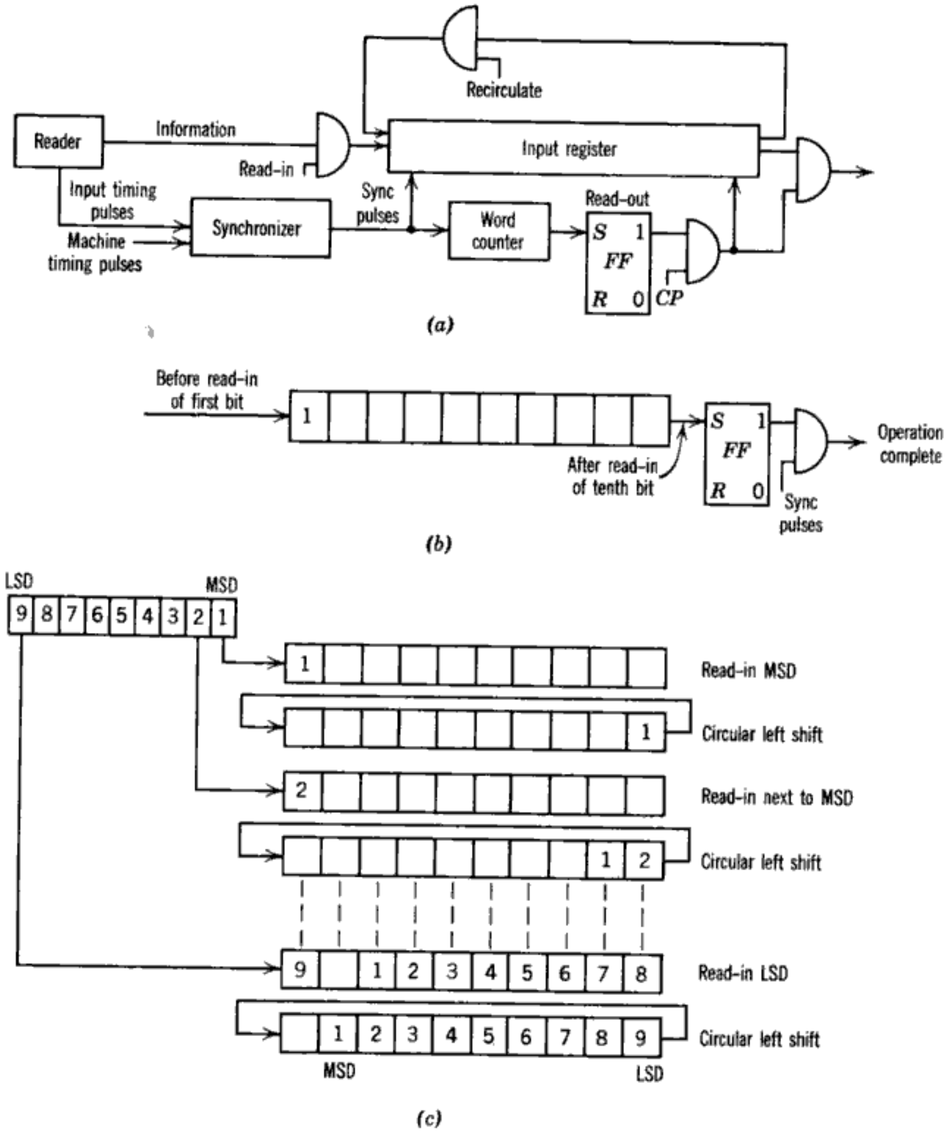


Fig. 129. Entering and ordering input information. (*a*) Reading into and out of input register. (*b*) Using register as word counter. (*c*) Rearranging the information.

Matching Input-Output to Computer

Since input and output devices operate at a slower frequency than the computer, means must be found to prevent waste of time while the external devices are operating. The following are several methods which have been used:

1. Programmed delay.
2. Independent input-output operation.

3. Multiplexing.
4. Reading in a block at a time.

Programmed delay. A good part of the lost time occurs because of the warm-up time required by auxiliary devices. To reduce the time lost, reading and writing of auxiliary devices are accomplished by means of two instructions: one to turn on the device, the other to read or write. Other instructions are sandwiched in during the warm-up interval.

Independent operation. A more efficient matching of input-output auxiliaries to the computer may be obtained by operating the auxiliaries independently of the computer. At the same time that the computer is performing instructions calling for calculations, the auxiliaries may be transferring information to and from the machine. This technique is effective provided interlocks are included to prevent the two routines from interfering with each other.

One method of obtaining independent input operation without interference is shown in Fig. 130. The circuit consists of an input register into which the reader output is shifted by means of sync pulses, and from which the bits are shifted out by means of clock pulses. The read-in is accomplished when the read-in flip-flop is set; and the read-out is accomplished when the read-out flip-flop is set. The register-full and the register-in-use flip-flops are the interlocks.

The input operation is accomplished by means of two instructions. The first activates the reader and reads a word into the input register. As soon as the reader is activated, an end-of-operation pulse is produced to signal the computer to continue with its program. The second instruction, which occurs later in the program, is a sampling-type instruction. It tells the computer to look at the input register. If the register is full, the register is emptied into the location specified by the address portion of the instruction. If the register is not full, the machine is made to sequence to the next instruction. Another similar sampling instruction may be included later in the program to transfer the information if the first instruction did not perform the transfer.

The circuit works as follows. The read-in instruction produces a pulse which primes A_6 and A_7. If the register-in-use flip-flop is set, a signal is fed to the control unit to cause the machine to sequence to the next instruction. If the register-in-use flip-flop is reset, A_7 produces an output which sets the read-in flip-flop; the read-in flip-flop, in turn, operates the reader, and sends a signal to the control unit telling it to continue instruction sequencing.

When the reader is in operating order, information pulses are applied to A_2 and input timing pulses to A_1. The input timing pulses combine

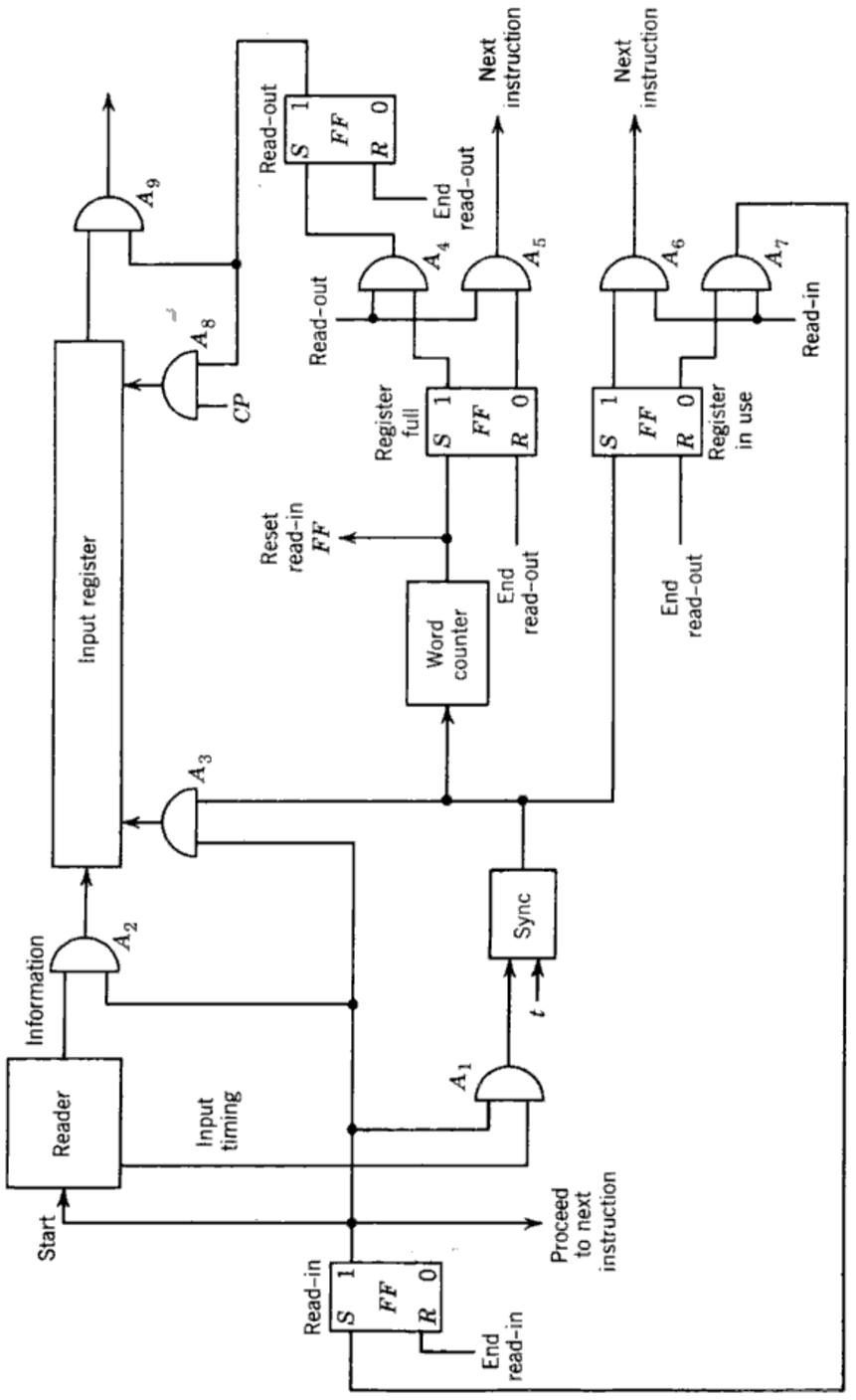


Fig. 130. Preventing interference between input-processing unit and rest of computer.

with machine timing pulses in the synchronizer to produce sync pulses. As soon as the first sync pulse is produced the register in use flip-flop is set to make sure that no other read-in instruction may be effective. The sync pulses shift the information bits into the input register; at the same time they are counted by the word counter. As soon as a complete word is in the register, the word counter produces a pulse which resets the read-in flip-flop to stop the read-in operation, and sets the register full flip-flop to inform the machine that now the register is ready to transfer out a word.

The read-out instruction produces a signal which is applied to A_4 and A_5. If the register-full flip-flop is reset, A_5 produces a pulse which causes instruction sequencing. If the register-full flip-flop is set, A_4 produces a pulse which sets the read-out flip-flop. The setting of the read-out flip-flop primes A_8 to allow clock pulses to shift the information out of the register. Upon completion of this operation, the two interlock flip-flops and the read-out flip-flop are reset in preparation for the next read-in instruction.

In some machines a register is provided which is filled automatically from an outside source at a certain slow rate. A sampling instruction similar to the instruction discussed above is then used to feed the word into memory. Many other variations of this basic scheme are possible.

Multiplexing. Another way to increase the effective rate of input and output is by connecting several auxiliaries of a kind to one computer. This arrangement is often made when magnetic tape is the external storage medium. While one tape handler is rewinding tape, another tape handler may be reading and feeding information into the input register; and still another tape handler may be writing computer output information.

Instead of multiplexing auxiliary devices, several input or output registers may be used to speed external communication. A word may be fed from a reader into an input register. When this register is full, it is emptied into memory. While being emptied, the following word is fed from the reader into the second input register. As soon as read-in of the first register is complete, the second register is emptied into memory while the first register is filled with the next word from the reader, etc.

Read-in of block. To reduce the relative time consumed by warm-up of an auxiliary such as magnetic tape, information is usually fed to and from tape handlers in block form. Successive words of the block are fed to successive memory locations. Now, if computer operation is to go on independently of the block transfer, an interlock must be provided to prevent instructions from referring to addresses in memory that are between the address currently being used and the final address to be used. Such an interlock circuit is shown in Fig. 131. It consists of current and final address registers, two comparison circuits, and a flip-flop. The address programmed into a possibly interfering instruction is fed to both comparison

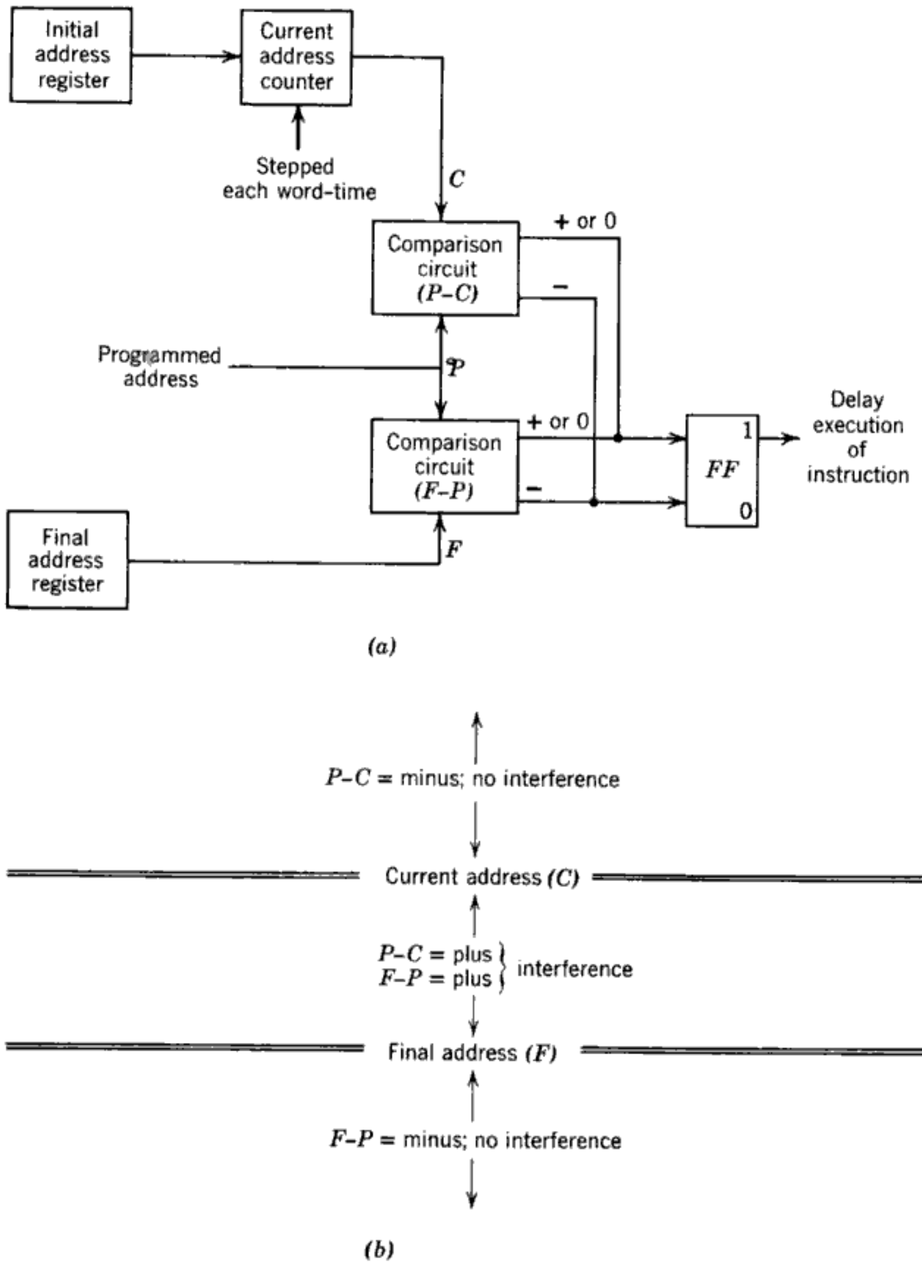


Fig. 131. Memory interlock during block transfer. (*a*) Block diagram. (*b*) Relationships among programmed, current, and final addresses.

circuits. In the first, the current address C is subtracted from the programmed address P. If the difference $P - C$ is negative, it means that the programmed address is smaller, and that it occurs outside the area of possible interference. The flip-flop is reset, allowing the instruction to be performed. If the difference $P - C$ is plus (0 is considered to be plus)

it means the programmed address occurs within the area of possible interference. The flip flop is set, delaying the performance of the instruction. In the second comparison circuit, the programmed address P is subtracted from the final address F. If the difference $F - P$ is negative, the programmed address is larger and thus occurs outside the area of possible interference; the flip-flop is reset. If the difference $F - P$ is positive, it means that the programmed address occurs within the area of possible interference; the flip-flop is set, and execution of the instruction is delayed until the block input instruction is completed.

Insertion or Deletion of Information

Information may be inserted by means of a generator. A pulse applied to the generator causes the generator to produce an appropriate signal combination. This technique is often used when an *EB* symbol must be added onto the end of a group of words leaving the computer to be read into the output auxiliary.

Information may be removed by means of a detector. A detector looks for a certain combination or a certain group of combinations of signals. If these appear, the detector produces a pulse which inhibits the flow of these coded combinations.

EXTERNAL STORAGE

External digital storage is different from the type of storage required for memory. In the memory the emphasis is on accessibility. In external storage the emphasis is on capacity. As a rule, the greater the capacity, the longer the access time. So the choice between the two characteristics is always a compromise.

Because the emphasis is on capacity even at the expense of access time, most storage devices used for external storage are of the sequential access type.

Sequential access storage devices may be of two kinds: those storing continuous records and those storing unit records. Examples of continuous-record and unit-record storage devices are given in Table 21.

The unit record lends itself to manual filing and sorting as well as to automatic sorting. The continuous record, on the other hand, cannot be sorted manually, only automatically: words are read, compared, and then rewritten onto another continuous record. The continuous record, however, lends itself to faster feeding of information into and out of the computer.

TABLE 21

Sequential Access Storage Devices

Continuous Record	Unit Record
Magnetic tape	Punched cards
Punched paper tape	Magnetic cards
Film	Microfilm cards

INPUT AND OUTPUT AUXILIARIES

External Storage Readers and Writers

Readers act as links between external storage and the input register. Writers act as links between the output register and external storage.

After an operator puts a roll of tape or a bunch of punched cards into the reader, the computer may be started. Instructions in the program, or as determined by the control panel, activate the reader and allow it to feed words into the input register and from there to the arithmetic or memory units. Instructions are also available for feeding in blocks or groups of blocks, or the information stored in only certain fields on punched cards or the information stored on a group of punched cards.

Similarly, after the operator places a set of blank cards into the punch, or a roll of magnetic tape into the magnetic-tape writer, the computer may be operated. Instructions in the program, or as determined by the control panel, activate the writer or punch to allow it to accept words coming from the memory or arithmetic units via the output register.

When magnetic tape is used for external storage, one handler is utilized for writing and for reading. In both cases the magnetic tape is moved with reference to the heads. Write heads are activated for writing and read heads are activated for reading. Most other auxiliary devices have separate readers and writers.

Magnetic Drum

Although it is a cyclic access device, the magnetic drum is occasionally used as an input-output device because it is capable of high-capacity storage. The drum does not possess items, either continuous records or unit records, which may be detached and filed. In essence the drum acts as an extension of the internal memory.

Direct

By means of a keyboard, words or parts of words may be fed directly into the computer. As any one key is depressed, a mechanical encoder causes an electric-code combination representing the chosen character to be fed to the input register. As each successive key is depressed, the bit combination enters the input register and previous character combinations are shifted further along in the register. After a complete word is entered, the computer transfers the information to memory or to the arithmetic unit. Transfer of this information to another part of the machine may be under control of a switch on the panel or under control of the program.

Results of computations are usually stored in memory. Output instructions transfer these words to the output register and from there to an output printer. At the printer the words are decoded and solenoids are activated to cause the characters to be printed on paper.

There are usually groups of display lights on the computer panel to indicate the contents of the important registers at any time. Each display light may be a neon lamp connected to one of the storage cells of a register. When the storage cell to which the neon is connected is storing a ONE, the neon lights; when it is storing a ZERO, the neon does not light. Thus the neons flicker on and off according to the changing pattern of the bits running through the machine. When machine operation halts, the final contents of the registers are shown by the display lights. This feature is useful for checking out a program of instructions and for trouble shooting.

ANALOG-TO-DIGITAL CONVERTERS AND DIGITAL-TO-ANALOG CONVERTERS

Some inputs to the computer may arrive in analog form. This analog information arrives from natural phenomena being studied by scientists, or from physical processes occurring in industry. Some analog quantities which we may want to measure and apply directly to a digital computer are pressure, temperature, humidity, flow of a liquid, stress, speed of movement or of rotation. To feed such information directly into a computer we need an analog-to-digital converter.

At the output end of the computer, a digital-to-analog converter may be needed to convert digital results of calculations into graphical form, or into continuous power for controlling an output mechanism.

Analog-to-Digital Converters

Analog-to-digital converters may be of several types. Herein only a typical electrically operated and a typical mechanically operated converter are discussed.

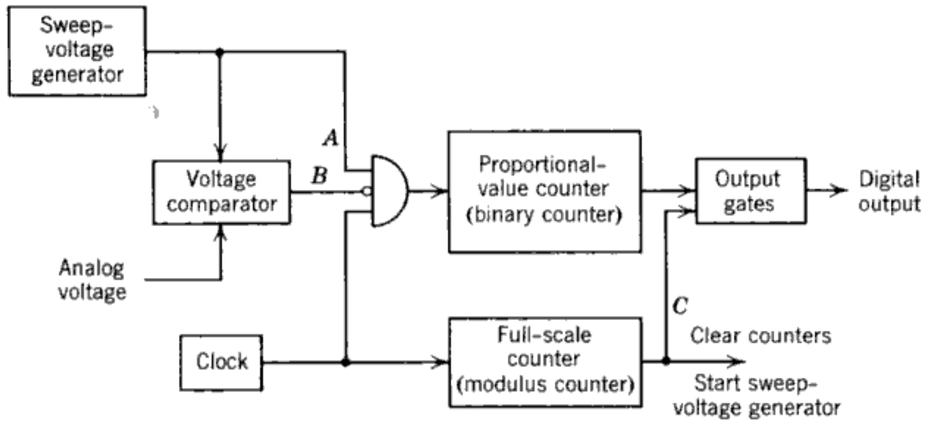


Fig. 132. Analog-to-digital converter, electrically operated. (*a*) Block diagram. (*b*) Waveforms.

The electrically operated encoder to be discussed uses a time-encoding technique. In this technique the amplitude of an electric signal is converted into an equivalent time. The time is then converted into an equivalent binary number.

A simplified block diagram of such a time-encoding converter is shown in Fig. 132. It consists of a linear sweep-voltage generator which forms the basis for our timing; a clock for counting bits of time; a voltage comparator for comparing the analog voltage with the sweep voltage; a binary counter for determining the binary number equivalent to the analog voltage; and a modulus counter for triggering the next sweep.

The conversion of an analog voltage to a digital number starts at the sweep's baseline (point *A*). As long as a sweep voltage is present at the

gate, clock pulses go through it to the proportional-value counter. When coincidence is achieved between the sweep voltage and the analog input voltage (point *B*), the voltage comparator produces a pulse which inhibits the gate, thus preventing clock pulses from further stepping the proportional-value counter. This counter thus stores a binary number whose value is proportional to the time between *A* and *B*, or whose value is proportional to the input analog voltage.

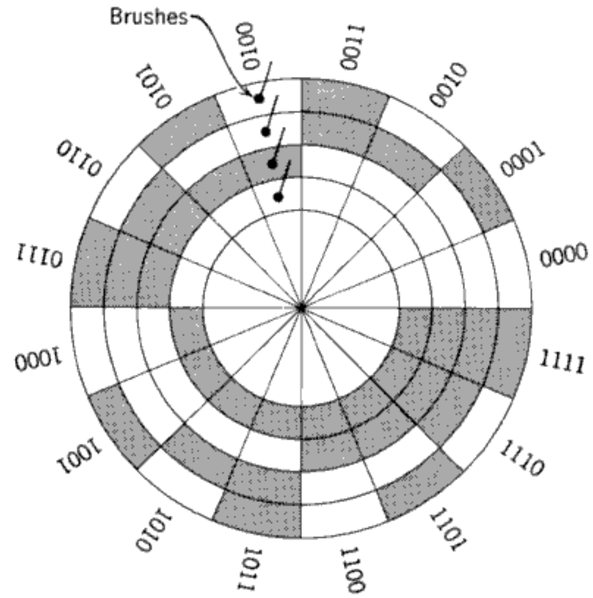


Fig. 133. Analog-to-digital converter, mechanically operated.

The full-scale counter is a modulus-type counter, and is stepped each time the proportional-value counter is stepped. After point *B*, the full-scale counter continues to be stepped until it reaches a count representing the full scale of the sweep. At this point, the modulus counter produces a pulse which allows the contents of the proportional-value counter to be read out. This pulse also clears both counters and causes the sweep-voltage generator to start the next sweep. During the following sweep, the analog voltage is sampled again and is converted to a binary number. The frequency of sampling may be determined by the frequency of the sweep.

A simple mechanically operated converter may be built by encoding a physical representation of a code onto a disk rotated by a shaft. Figure 133 shows a disk with such a binary pattern. The dark areas represent conducting surfaces and the light areas insulating surfaces. An electric brush connected to a source of power is in contact with each channel. As the shaft rotates, current flows through some channels and not through others, depending on the configuration of conducting and insulating areas.

The illustration shows the disk stopped in a position to cause the brushes to read 0100: only the brush in the second inner channel is on a conducting surface and thus conducts current.

Digital-to-Analog Converters

A digital-to-analog converter may be built by a process of weighting. This process is outlined in Fig. 134 for a 3-bit number. Switches are used in each bit position to connect a battery or not, depending upon whether

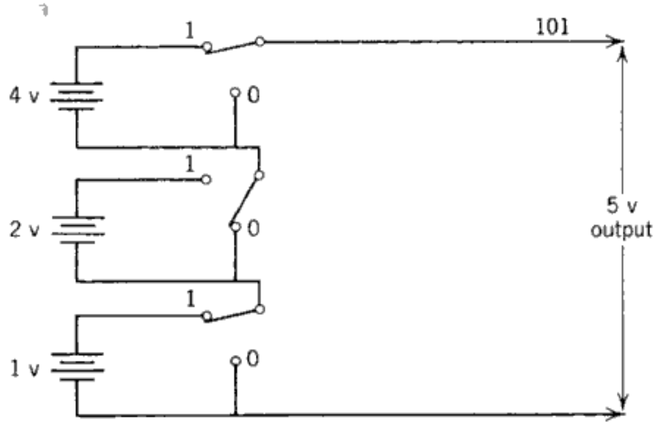


Fig. 134. Digital-to-analog converter.

there is a ONE or a ZERO in that position. The battery voltages are proportional to the weights of the associated bit positions. The illustration shows that, for the binary combination 101, the 4-v battery and the 1-v battery are in series to produce an analog output of 5 v.

Of course, practical circuits are not built this way. The bits may be stored in flip-flops. The switches may be transistor switches activated by the outputs of the flip-flops. The actual summing of voltages may be obtained by the proper use of resistor networks.

PERIPHERAL EQUIPMENT

All the external devices discussed up to now are operated on line. By "on line" we mean that the devices are tied in directly with the computer. Information flows in an unbroken stream from the auxiliary input devices to the computer, and from the computer to the auxiliary output devices.

Some operations are best performed off line. An off-line operation is one performed at a time which bears no relationship with the computer

operation. For instance, after the computer goes through a program and places the results on magnetic tape, the magnetic tape may be removed and information on the magnetic tape may be transferred onto punched cards. On-line operation is related to continuous processing; off-line operation is related to batch processing.

The peripheral equipment used in off-line operation falls into several categories as follows:

1. Recorders.
2. Converters.
3. High-speed printers.
4. Communication devices.

Each is discussed briefly below.

Recorders

Means are required to record information onto the external storage medium before entry into the computer. For magnetic tape, typewriter-like devices are used to write coded information on the tape. As each key is depressed, a mechanical encoder produces the correct combination of pulses to be fed through write amplifiers to write the coded combinations on the tape. A similar arrangement is used to write on punched paper tape, except that here a coded mechanical output punches holes in the appropriate spots on the paper.

In a typical practical application such as the summarizing of department store sales, clerks transcribe information from cash register tapes onto punched paper or onto magnetic tape by means of recorders. Once a tape is prepared, it is mounted on the tape handler for feeding into the computer for processing. To avoid the slow, manual transcribing step, point-of-sale recorders have been developed. At the same time that the saleslady rings up a purchase on the cash register, the record of the sale is automatically punched on paper. At the end of the day, the information on the punched paper tape is fed directly to the computer by means of a paper-tape reader. The point-of-sale recorder introduces an extra degree of automation into data processing.

Converters

Converters are available which can convert information stored on almost any external storage medium to information stored on any other. One of the best input-output auxiliary devices developed up to now is the magnetic tape handler. However, many business firms have complete punched-card systems. By means of a punched-card-to-magnetic-tape converter, the information on the cards may be transferred to the magnetic

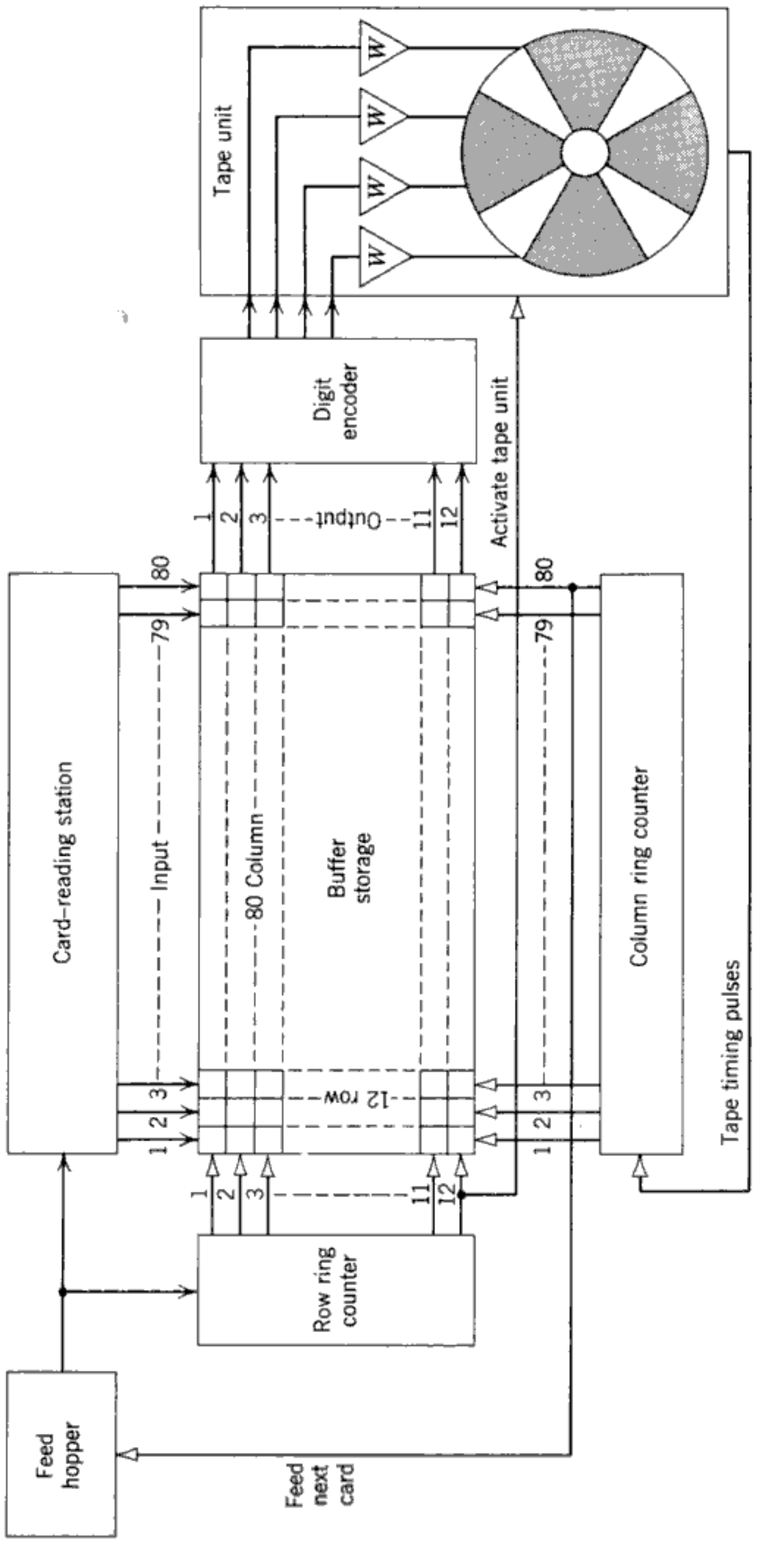


Fig. 135. Punched-card-to-magnetic-tape converter.

tape. Similarly, there are punched-card-to-punched-paper-tape converters, checks-to-paper-tape converters, checks-to-magnetic-tape converters, magnetic-tape-to-punched-cards converters, etc.

Only the punched-card-to-magnetic-tape converter is discussed here. The general principles of operation of other converters are similar.

In broad outline, the punched-card-to-magnetic-tape converter works as follows. A punched-card reader transfers the information stored on the punched card to buffer storage. A tape-handling device then transfers the information stored in the buffer to magnetic tape. A plugboard attached to the buffer storage enables us to rearrange, delete, or insert information to be written onto the tape.

A simplified block diagram of a punched-card-to-magnetic-tape converter is shown in Fig. 135. It has an 80-column 12-row buffer storage, a profile of a punched card. A row counter at the input side regulates the read-in operation a row at a time. A column counter at the output side regulates the read-out operation a column at a time.

The converter operates as follows. From the feed hopper punched cards are fed to the reading station one at a time. Each card is applied so that the top side hits the row of 80 reading brushes first. When the card is in the first-row reading position, the row counter produces a pulse which allows this row of information to be fed to the first row of 80 storage cells in buffer storage. As each succeeding row of the card is read, the row counter is stepped to produce a signal which allows the information to be applied to a succeeding row of storage cells in the buffer.

After the 12th row is read, a signal activates the tape handler. A cycling device in the tape handler produces tape timing pulses which step the column counter. As the column counter is stepped, the information in another column of buffer storage cells is fed through a digit encoder and write amplifiers to the magnetic tape. After the 80th column is written onto the tape, a signal applied to the feed hopper starts the next cycle of operation.

Note that the buffer storage in this case effectively rotates the punched card by 90°. Read-in is by row since less time is required to read 12 rows than would be required to read 80 columns. But read-out must be column by column since digits are stored in columns.

To provide flexibility to the converter, a plugboard is connected between buffer storage and the digit encoder. The plugboard enables the operator to modify the sequence of writing onto the magnetic tape. By failing to wire certain buffer cells to the digit encoder, words may be deleted from the tape. By wiring to the digit encoder signals generated from outside sources, extra words may be inserted onto the tape.

High-Speed Printers

A printer which reads the computer output directly usually prints a character at a time. A printer which is used for off-line operation may read a complete line at a time or at least during a small cycle of operation. The off-line printer operates at high speed. Mechanical printers which can print up to 5000 lines per minute have been developed. The off-line printer is actually a converter. It converts information stored on the external storage medium to the printed word.

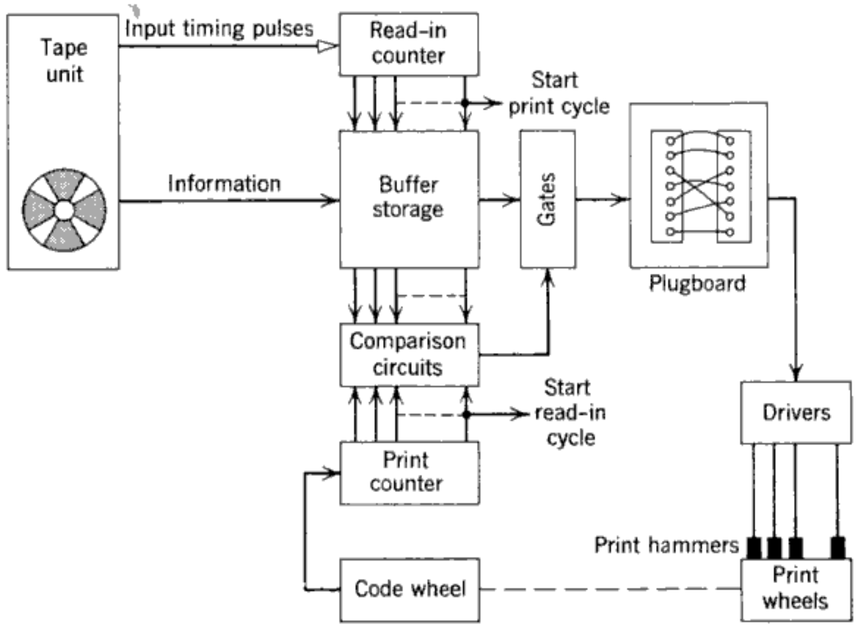


Fig. 136. Printer synchronization.

Since it is a converter, the high-speed off-line printer must possess buffer storage which is filled at a rate compatible with external storage, and is emptied at a rate determined by the printing device. In addition, the high-speed printer which prints a complete line at a time needs a special synchronizing arrangement to assure that each character is printed in the proper spot across the line.

Only the block diagram of the type-wheel printer is discussed. General operation and synchronization problems of the spinning-disk and matrix-type printers are related to those present in the type-wheel printer.

A simplified block diagram of a magnetic-tape-fed high-speed type-wheel printer is shown in Fig. 136. It consists of a buffer storage with a counter controlling input from the tape, and a code-wheel–counter–comparison-circuit combination controlling the output from buffer

storage. A plugboard and a set of drivers which cause print hammers to strike the type wheels complete the block diagram.

At the start, input timing pulses read from the tape step the counter, which then allows each character to enter a succeeding address in the buffer storage. Upon completion of read-in, the counter produces a pulse which starts the print cycle.

The print cycle must be synchronized with the rotation of the type wheels and code wheel. To be sure of the character we are printing, we must start the print cycle at a point in the rotation of the wheels which is considered as reference—just before the appearance of the letter *a* at printing position, for instance. Starting at this point, the counter is cleared. As each character on the print wheels reaches printing position, the code wheel produces a pulse which steps the counter; the counter then produces a binary-coded combination representing the character apposing the print hammers. This coded combination is applied to a comparison circuit. To the other side of the comparison circuit are applied the contents of each character address in the buffer. In reality, there is a separate comparison circuit for each character address. For every location in the buffer which is storing the same character as that coming from the counter, a pulse is transmitted to the associated driver to activate the desired hammer. If comparison occurs for more than one location in the buffer, more than one hammer is struck. As the print wheels make one revolution, all the characters have been compared and all the characters on one line have been printed.

The plugboard may be used to rearrange the location of any of the characters on the printed line.

Communication Devices

Information in external storage may be sent over wire or over radio. One of the most common forms of wire communication is teletype. Five-hole punched paper tape may be run through teletype which transmits the information over wire and causes paper tape at distant points to be punched with the same information. Because of the prevalence of teletype communication, teletype is often called the common language. Any computer which can accept teletype may "speak" to many other devices in other parts of the world.

Many firms have been storing and using punched cards because they are easy to file and can be made understandable to humans as well as to machines. To solve the long-distance communication problem, a punched-card transmitter-receiver combination has been developed. Punched cards are inserted into the transmitter. The transmitter converts the stored information into electric pulses which are transmitted over telephone

wire to a distant receiver. The receiver picks up the electric signals and causes equivalent holes to be punched into cards.

Similarly, a magnetic-tape transmitter-receiver combination operating over telephone wires has been developed.

CHECKING

As was indicated previously, all transfers of information may be checked by means of the parity check or by a forbidden-combination check.

When fixed blocks of information are transfered, checking may take the form of counting the number of words in the block. If a full block is not present, or if more than the appropriate number of words is being transferred, an indicator lamp lights.

A recorder may be checked by means of a verifier. A verifier is a device which compares the information recorded with the same information being mechanically entered by the operator a second time. In magnetic-tape recording, for instance, the operator records information on tape and then allows the verifier to read this information and to store it in its buffer. The operator then operates the keys of the verifier in the same way he would operate the keys of the original recorder to enter the same information. The verifier compares the two sets of information and lights a warning lamp if any discrepancy occurs.

Converters, printers, and all other devices using buffer storage may be checked by means of a comparison check. After the information is read into the buffer, the same information is read once more. The second input information is compared with the information stored in the buffer from the first reading. If the two are the same, the rest of the process continues. If the two are not the same, the machine halts and lights an error lamp.

A modification of this comparison check is used by some punched-card-to-magnetic-tape converters. After the information is recorded onto the tape, it is read from the tape and compared with the information previously in the buffer. This check is called an echo check.

In addition to the above checks, there are checks used in connection with certain devices. For instance, pulses may not record reliably at certain points on magnetic tape. These bad spots may be distinguished by punching a hole before and after the bad spots. A light-photocell arrangement on the tape handler may be used to find the holes. The first hole sets a flip-flop which inhibits tape reading; the second hole resets the flip-flop to allow normal operation to resume.

A checking arrangement commonly used in punched-card readers is called blank-column and double-punch detection. This checking scheme

is good only when reading digits. Since the digits in each column of a card are represented by one punch, the absence of a punch or the presence of two punches indicate an error. Blank-column and double-punch detection may be easily obtained by having a complementing flip-flop associated with each column. At the start of reading, all the flip-flops are reset to ZERO. The presence of a punch in a column flips the flip-flop to ONE and allows the information to be read out. The absence of a punch in a column causes the flip-flop to remain reset. The presence of two punches causes the flip-flop to set to ONE, and then to be reset back to ZERO. In both error conditions, read-out is prevented and an alarm lights.

SUMMARY

1. The input-output system consists of input-output units which order and direct the information within the machine; input-output auxiliaries which form a link between the external environment and the computer; and peripheral equipment which processes information outside the computer.

2. The input-output units consist of input-output registers and input and output control circuits. The registers operate as buffers, and the input-output control circuits perform functions which vary with the application. Functions often performed are synchronization, rearrangement of information, insertion or deletion of information, and matching the input-output devices to the computer.

3. Matching of input-output devices to the computer is necessary because auxiliary devices are usually much slower in operation. To speed up the effective communication rate, the following methods have been used: programmed delay, independent input-output device operation, multiplexing, and reading in a block at a time.

4. Whenever the input-output system is made to operate at the same time that the computer is performing calculations, an interlock for preventing interference is needed. This interlocking is accomplished by means of flip-flops, counters, and comparators. Flip-flops indicate whether a device is in operation or not; counters count digits or words to keep track of a block transfer and to indicate when the transfer is completed; comparators are used to indicate whether a specified address is in an area of potential interference or not.

5. Inputs may be of two kinds: digital and analog. The digital input may be directly applied by manipulation of panel controls; or it may be applied from external storage. Analog inputs are applied through analog-to-digital converters.

6. Outputs too may be required in digital or analog form. Digital outputs may be direct in the form of display lights or printed characters; or written onto external storage. Analog outputs are obtained by means of digital-to-analog converters.

7. Peripheral equipment is used to communicate between the external storage of the computer, on the one hand, and man, other external storage, and long distance devices, on the other hand. Communication between man and external storage is performed by recorders; between two different types of external

storage by converters; and long-distance communication by telephone or radio transmitters and receivers.

8. Redundancy is the main form of check in devices which are part of the input-output system. Verifiers compare a newly written tape with one previously written. In converters or other devices possessing buffer storage, the information in storage may be compared with the information read once more. Information recorded may also be compared with the same information still present in buffer storage.

9. *Definitions to Remember*

EXTERNAL STORAGE—Information stored on a medium that is not permanently connected to the computer.

READER—A device which translates the information stored on an external storage medium into electric pulses which may then be fed to the input register of the computer.

WRITER—A device which translates the electric pulses from the output register of a computer into information stored on an external storage medium.

SYNC PULSE—A pulse produced upon coincidence of an input timing pulse from external storage and a machine timing pulse. The sync pulse is used to gate the associated digit into the input register.

UNIT RECORD—A record where a small group of characters is stored as one item. Unit records are easy to file. Punched cards are a good example.

CONTINUOUS RECORD—A record where a long string of information is stored on one unit. A good example is magnetic tape.

BLOCK—A group of words considered or transported as a unit, particularly with reference to input and output.

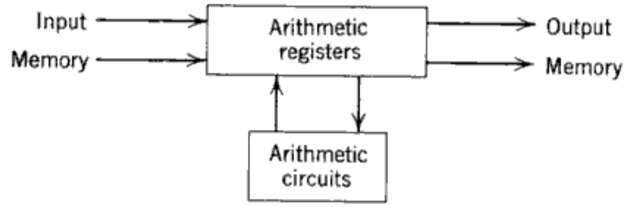
INTERLOCKING—The process of making sure that when two processes are proceeding at the same time, no interaction can occur between the two parts of the machine performing each of the two processes. In case of conflict one of the processes is made to stop.

RECORDER—A device which is used to write information manually onto a document or storage medium.

CONVERTER—A device for converting information stored in one form into information in another form.

ON-LINE OPERATION—Operation which is tied directly to computer operation.

OFF-LINE OPERATION—Operation which may proceed concurrently with computer operation.



CHAPTER 14

Arithmetic unit

The majority of computer operations are performed by the arithmetic unit. The arithmetic unit consists of several registers for holding operands while they are being processed, and of arithmetic circuits working in conjunction with the registers to perform addition, subtraction, multiplication, division, and other operations.

The typical arithmetic unit possesses three registers, labeled *AR*, *XR* and *DR* for convenience. The *A* register (*AR*) is usually the link between the arithmetic unit and the rest of the computer: it receives operands and transmits results of operations. Read instructions transfer to *AR* the contents of the memory location specified by the address. Write instructions transfer words present in *AR* to the memory location specified by the address. Other operations such as shifting and taking the square root are performed on operands stored in *AR*. *XR* receives the second operand in an arithmetic operation. *DR* takes part in the performance of more complicated operations such as multiplication and division.

In some machines inputs may be fed directly to *AR*. Results may be fed directly from *AR* to the output.

The arithmetic circuits consist of an adder which performs addition; sign or complementing circuits which modify adder operation to perform subtraction when required; and control circuits for modifying adder operation to adapt it to lengthier operations such as multiplication and division. It may also possess checking circuits.

ADDITION

Adders, Accumulators, and Counters

An adder is a device which accepts an augend and addend and produces a sum. Several general methods are available for accomplishing this

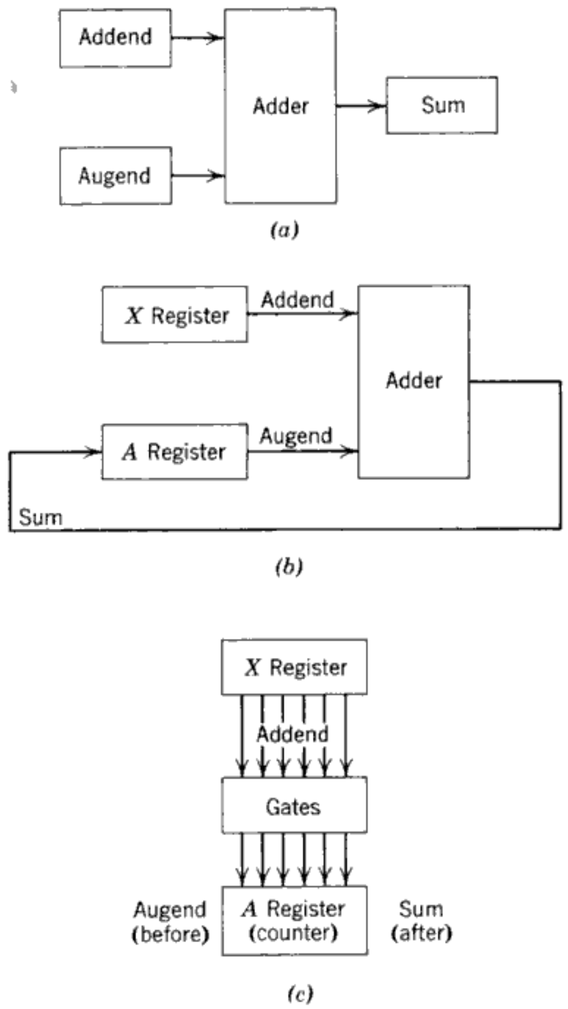


Fig. 137. Adders. (*a*) Adders, coincidence type. (*b*) Accumulator. (*c*) Adder, counter type.

task, among them being the coincidence-type adder and the counter-type adder. In the coincidence-type adder (Fig. 137*a*), pairs of augend and addend bits are fed serially into a binary adder such as that explained in Chapter 4. The coincidence of each pair of bits produces an output bit

which is a ONE or a ZERO depending upon the rules of binary addition. As it is manufactured, each bit is applied to a sum register.

The adder output may be fed back to the register from which came the augend. If this is done, the latter register is called an "accumulator" because, as new numbers are added, the cumulative total may be built up in this register. The accumulator is also called the *A* register or *AR* (Fig. 137*b*).

Another method of performing addition is by making *AR* a counter. The augend is applied to *AR* and causes it to count to this input value. The addend is then applied through gates to the counter, causing it to step an equivalent number of times to bring the final count to a value equal to the sum (Fig. 137*c*). The operation of this type of adder is analogous to the operation of accumulators in mechanical adding machines.

Serial and Parallel Addition

When binary numbers are involved, addition of the bits may be accomplished serially or in parallel. Examples of each method are shown in Fig. 138.

In serial addition only one bit adder is needed. During the first bit-time, the least significant augend and addend bits are applied to the bit adder to produce a sum bit and a carry bit. The carry bit is applied through a 1-bit-time delay circuit to the input side of the bit adder. During the second bit-time, the bit adder adds the second least significant augend and addend bits plus the carry bit from the least significant position. In each succeeding bit-time the following bits are added plus any carry from the previous position. After a word-time, a complete sum is produced.

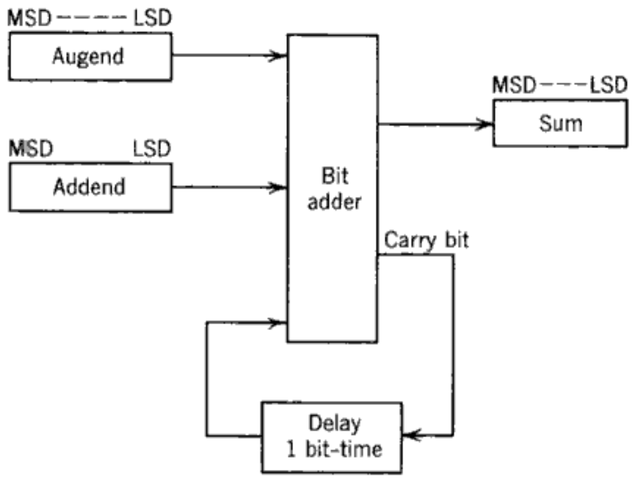
In parallel addition, as many bit adders are needed as there are bits to a word. The augend and addend bits in each position are added separately to produce all the bits of the sum and carries where appropriate. The carries are then applied to following columns to modify the sum.

If the carry produced in any one column affected only the sum bits in the following column, the complete parallel addition could be accomplished during two bit-times, one for producing the sum and one for modifying the sum with the carries. However, carries may propagate as can be seen from the following example:

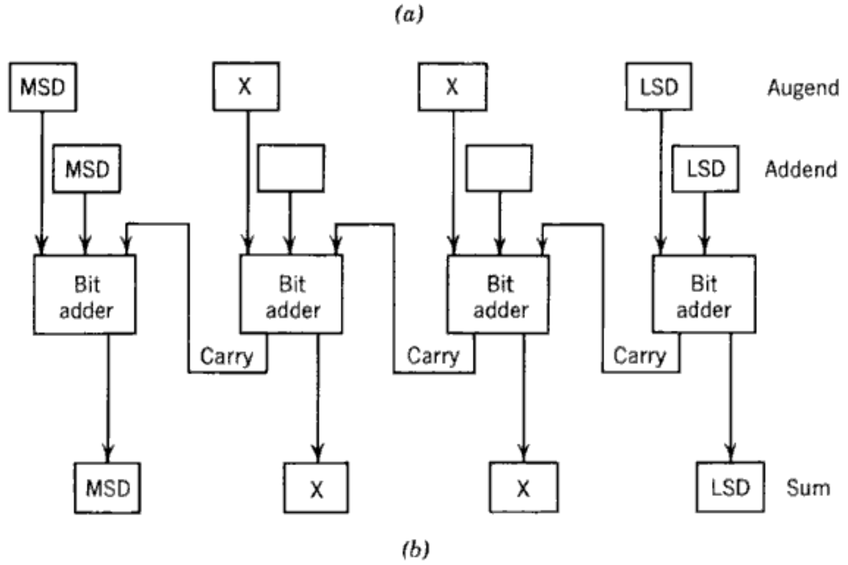
```
 0111111   Augend
+0000001   Addend
 111111    Carries
-------
 1000000
```

Although it is much faster, parallel addition is not quite as fast in

operation as one would normally suppose. Time is needed for carries to propagate. An example of parallel addition with automatic propagation of carries is shown in Fig. 139. It consists of an upper row of complementing flip-flops representing stages of the accumulator; a lower row



(a)



(b)

Fig. 138. Serial and parallel addition. (*a*) Serial. (*b*) Parallel.

of flip-flops representing stages of the addend register; a series of addition gates A_1 controlled by an add pulse; a series of carry manufacturing gates A_3 controlled by a carry pulse; and a series of delays D and gates A_2 for controlling automatic carry propagation.

The adder works as follows. After a number is applied to the addend register and another number to the accumulator, an add pulse is applied

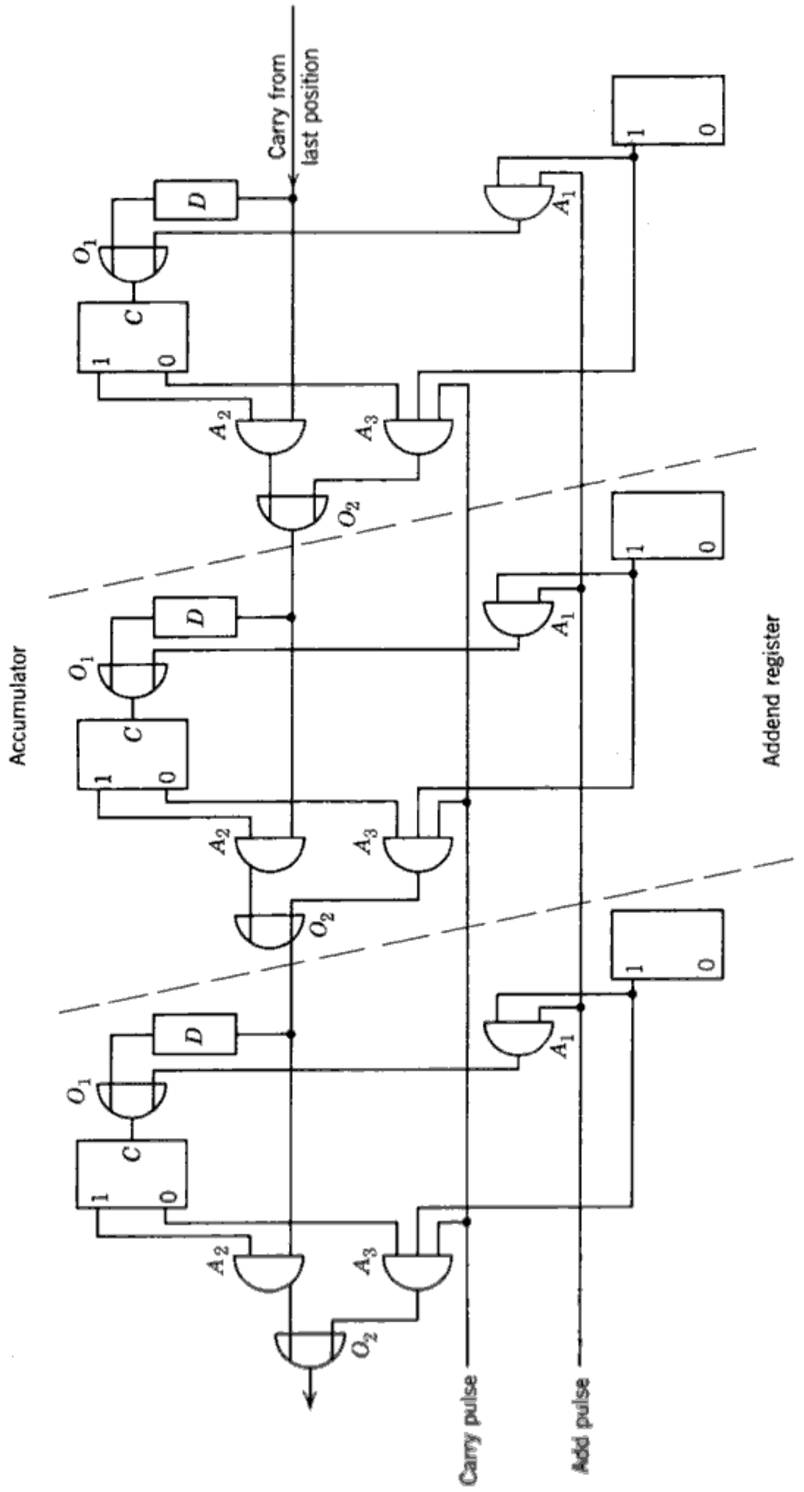
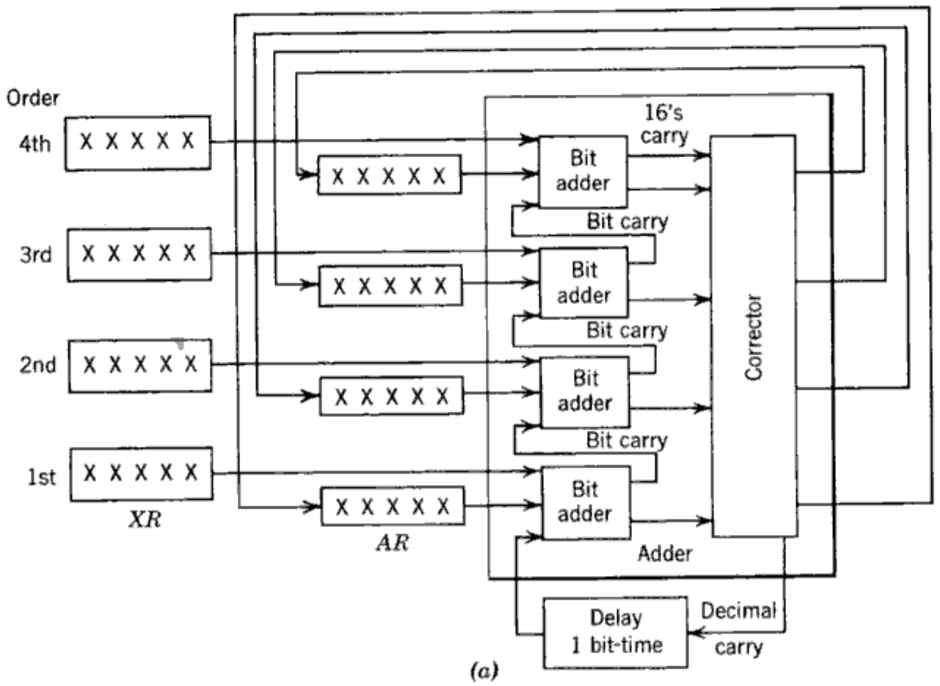
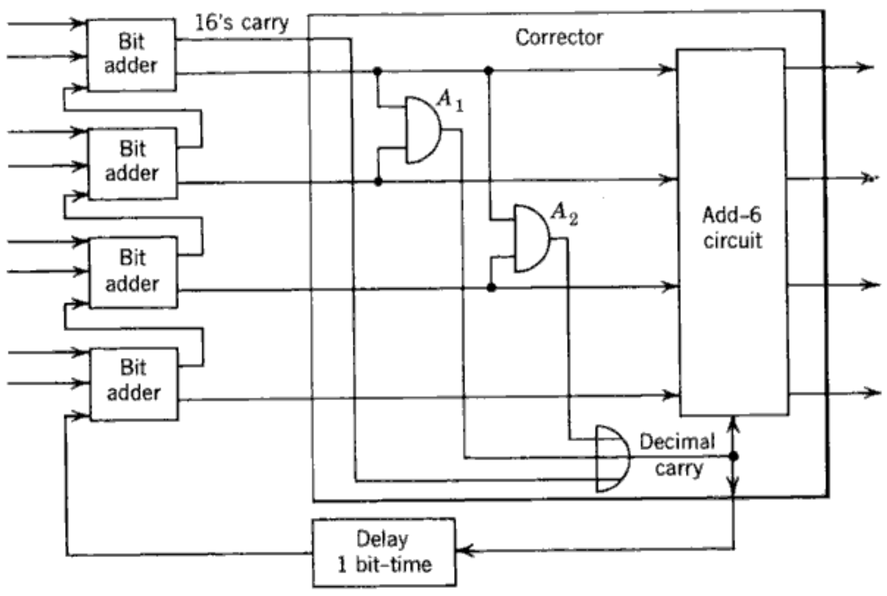


Fig. 139. Parallel addition with automatic carry propagation.



(a)



(b)

Fig. 140. Serial-parallel addition of binary-coded decimal numbers. (*a*) Serial-parallel addition. (*b*) Corrector.

to all gates A_1. Those addition gates associated with addend flip-flops storing ONE produce output pulses which flip the associated accumulator flip-flop to produce a tentative sum.

As soon as this tentative sum is produced in the accumulator, a carry pulse is applied to all gates A_3. If the addend flip-flop is storing ONE and the associated accumulator flip-flop is storing ZERO, signifying that 1 was added to 1, a carry pulse is produced by A_3 and fed to the succeeding accumulator flip-flop through 0_2, delay D and 0_1.

If an accumulator flip-flop is storing a ONE and a carry is being applied to it, obviously the flip-flop must be flipped to ZERO and a carry must be propagated to the following stage. These two actions are performed independently. The carry is applied to A_2 which propagates it to the following stage. After this occurs, the carry applied through D and 0_1 flips the accumulator stage. Delay D assures this independent action.

Serial-Parallel Addition and Correctors

In machines using coded decimal numbers, addition is usually performed in a serial-parallel manner. The outline of the method for adding two numbers consisting of binary-coded decimal digits is shown in Fig. 140. The addend register (XR) and the accumulator (AR) each consist of four subregisters, one for each of the four bit orders of a digit. The adder consists of four bit adders, one for each bit in a digit, and a corrector. The carry is applied through a bit-time delay to the lowest order bit adder.

The four bits of the least significant digit of the augend and addend are applied simultaneously to the four bit adders. After carries have propagated through the bit adders, a 4-bit sum digit plus a possible carry bit from the fourth-order adder are produced. These bits are modified by the corrector; the resultant 4-bit sum is sent to the accumulator, and any carry is transmitted through the delay element to arrive at the first-order bit adder at the same time that the four bits of the following digit are being applied to the bit adders. Succeeding digits are added in a similar manner. After a word-time the addition is complete.

The corrector is required because the rules of binary addition are not completely obeyed when adding binary-coded decimal numbers. In the addition of binary-coded decimal numbers, the following three general situations may occur:

1. The sum is less than 10: no carry occurs, no carry is needed.
2. The sum is between 10 and 15: no carry occurs, but a carry is needed.
3. The sum is 16 or more: a 16's carry occurs, but a 10's carry is needed.

Three examples representing the above three cases are given below:

1.	2.	3.
$0100 = 4$	$0110 = 6$	$0111 = 7$
$0011 = 3$	$1000 = 8$	$1001 = 9$
$\overline{0111} = \overline{7}$	$\overline{1110} = \overline{14}$	$1\ \overline{0000} = \overline{16}$

In case 1, the result of addition is correct, but in cases 2 and 3 the results are incorrect. The corrector is needed to take care of these two cases. An example of such a corrector is shown in Fig. 140*b*.

In case 2, no decimal carry is produced but one is needed. To correct for this situation, means must be present to detect when the tentative sum is between 10 and 15. This can be done if we note that for each of these digits a ONE is present simultaneously in the fourth- and second-order bits or in the fourth- and third-order bits; and this situation is not true for any other digit. Gate A_1 in the corrector detects the first situation, and A_2 detects the second situation. If either of these situations occurs, a decimal carry pulse is produced and applied through the delay circuit to the first-order bit adder. Since a decimal carry is thus added to the following digit, 10 must be subtracted from the present digit. The subtraction of 10 may be performed by adding 6 and ignoring the carry which occurs from the fourth-order bit. Ignoring this carry is equivalent to subtracting 16. This correction is accomplished by the add-6 circuit as soon as the developed decimal carry is applied to it. The case-2 example given above is mechanized as follows:

$0110 =$	6	
$+1000 =$	8	
$\overline{1110} =$	$\overline{14}$	Binary
$+0110 =$	$+6$	Correction
$1\ \overline{0100} =$	$\overline{14}$	Binary-coded decimal

In case 3, a 16's carry is produced but a 10's carry is needed. We correct the sum by adding binary 6. The 16's carry is applied to the decimal carry line to be returned to the first-order bit adder through the delay, and also to activate the add-6 circuit. The case-3 example given above is mechanized as follows:

$0111 =$	7	
$+1001 =$	9	
$1\ \overline{0000} =$	$\overline{16}$	Binary
$+0110 =$	6	Correction
$1\ \overline{0110} =$	$\overline{16}$	Binary-coded decimal

Overflow

Occasionally a carry occurs from the most significant position. Since the sum now possesses more digits than can be contained in a register, this carry is called an overflow.

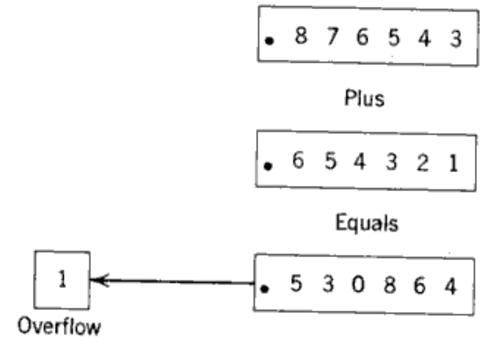


Fig. 141. Production of overflow.

The occurrence of overflow indicates that the machine has manufactured a number which is greater than the maximum allowable number. If the machine handles fractional numbers, the occurrence of overflow indicates that a number 1 or greater has been manufactured (Fig. 141).

The overflow is usually fed to a flip-flop. This overflow is used to control arithmetic operations, especially those involving the use of complements as discussed in the following.

ALGEBRAIC ADDITION

The adder produces the absolute-value sum of the augend and addend. To perform algebraic addition, the signs of the operands must be taken into account. As we have learned in Chapter 3, when the signs of the operands are alike, the operands are added. When the signs are unlike, the smaller operand is subtracted from the larger, and the result is given the sign of the larger. The same result may be obtained by adding the complement of the negative number to the true value of the positive number. A second way of performing algebraic addition is by converting all negative numbers into complement form and applying the rules of algebraic addition to these numbers. Simple methods for performing these two types of addition are given below.

Algebraic Addition of Numbers in Absolute Value Plus Sign Form

Algebraic addition may be performed by means of a circuit similar to that shown in Fig. 142. The circuit consists of a binary comparator, an adder, and two subtractors. At the start of algebraic addition, the sign

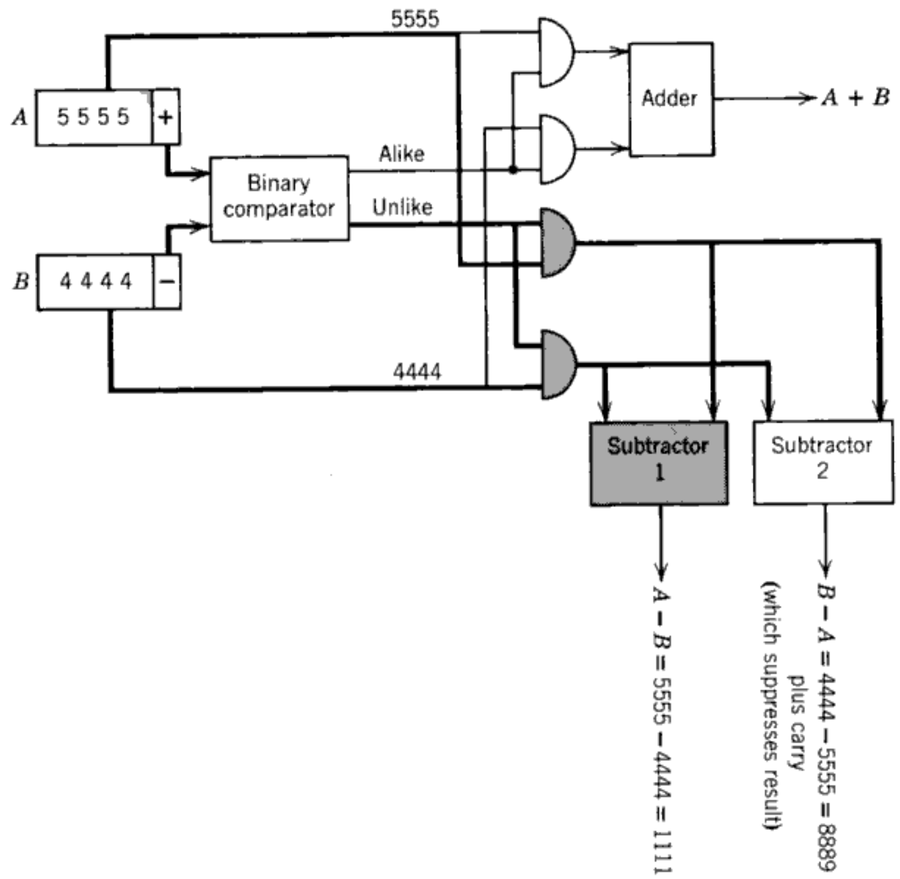


Fig. 142. Algebraic addition, absolute value plus sign form.

bits of the two operands A and B are fed to the binary comparator, while the digits themselves are applied to a group of gates. If the two signs are alike, the operand digits are allowed to flow to the adder. If the signs are unlike, the switches allow digits A and B to be fed to the two subtractors. Subtractor 1 subtracts the addend B from the augend A, and subtractor 2 subtracts augend A from addend B. The subtractor that subtracts the larger number from the smaller number produces a carry; this carry inhibits the result of this subtractor.

For example, assume

$$A = +5555$$
$$B = -4444$$

The comparator activates the unlike control line, and the two numbers are fed to the subtractors. Subtractor 1 produces $5555 - 4444 = 1111$ and subtractor 2 produces $4444 - 5555$ which gives 8889 and a carry. The carry inhibits this last result.

Algebraic Addition by the Use of Complements

A better way of accomplishing algebraic addition on numbers given in absolute value plus sign form is to complement negative numbers and then add. Positive results are produced in absolute-value form. Negative results, on the other hand, are given in complement form and should be recomplemented when removed from the arithmetic unit.

A circuit which performs this operation is shown in Fig. 143. Figure 143*a* shows the input and adder circuits, and Fig. 143*b* the output circuits. Complementer 2 takes the 2's complement of the augend arriving to A register if the sign is negative. Complementer 1 complements the addend arriving to the adder if the sign is negative. The output is transmitted through complementer 3 which is under control of the sign circuits consisting of four flip-flops and two binary comparators. The presence of a signal to complement the output depends on the relative signs of augend and addend, and the presence or absence of overflow.

This algebraic adder operates as follows. First the augend is transmitted to AR through complementer 2 by means of a transfer instruction. The first bit arriving at the complementer is the sign bit which acts as a control signal for the remaining bits. If the sign bit is plus, the complementer allows the bits of the number to pass through it unchanged; if the sign is negative, the number is complemented as it passes through.

Next, an add instruction pulls the addend from the location in memory specified by the address and sends it through complementer 1 to the adder. Again, complementation occurs only if the sign is negative. The addend and augend hit the adder at the same time. The adder produces the sum which is returned to AR; in some cases it also produces an overflow which is applied to the sign circuits.

The result of addition is transmitted from AR through complementer 3 to the memory or to the output register by means of another transfer instruction. Complementation or no complementation of the output is determined by the sign circuits, which operate as follows. When the augend is originally applied to AR, the sign bit sets or resets the A sign flip-flop depending upon whether it is minus or plus. Similarly, when the

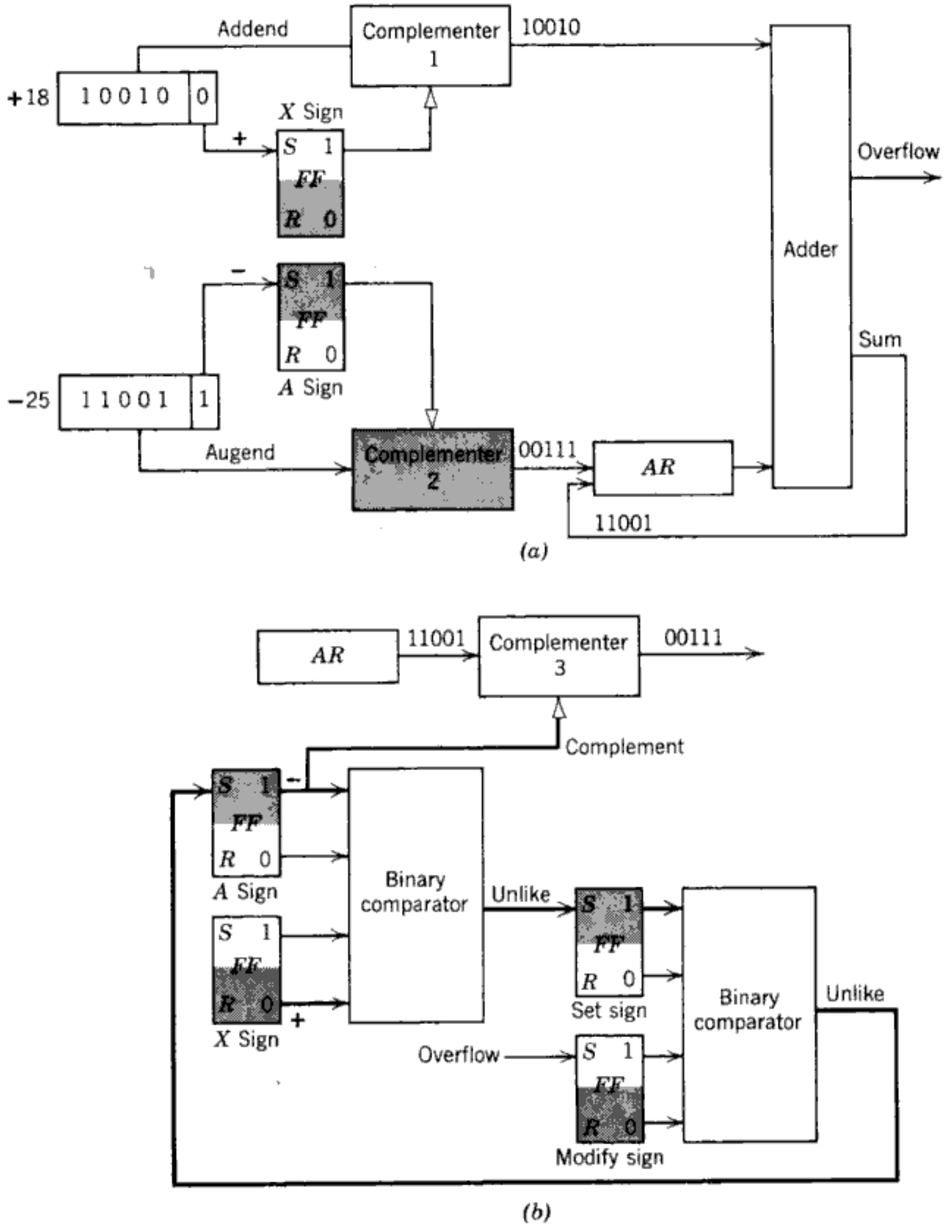


Fig. 143. Algebraic addition by adding complements of negative numbers. (*a*) Complementation on input and addition. (*b*) Complementation on output.

addend enters the adder, the sign bit sets or resets the *X* sign flip-flop, depending upon whether it is minus or plus. If the outputs of the two sign flip-flops are alike, the set sign flip-flop is reset; if unlike, it is set. If no overflow occurs upon completion of the addition, the modify sign flip-flop is reset; if overflow occurs, it is set. The outputs of the set sign and modify sign flip-flops are compared. If they are alike, the *A* sign flip-flop is reset; if they are unlike, it is set.

The illustration shows +18 and −25 being added algebraically. The sign bits appear first. Because there is a ONE (signifying minus) in the least significant place of the augend, the number 11001 is complemented to 00111 which is applied to *AR*. Next, the addend, 10010 is fed directly through because a ZERO signifying plus, is present in the least significant position. These two numbers are added in the adder to produce

$$\begin{array}{r} 10010 \\ 00111 \\ \hline 11001 \end{array}$$

which is fed to *AR*.

When the augend and addend were applied, the *A* sign flip-flop was set and the *X* sign flip-flop was reset. As a result, the unlike output of the comparator was activated to set the set sign flip-flop. Since no overflow was produced by the adder, the modify sign flip-flop is reset. The unlike output of the comparator causes complementer 3 to complement the output as it is transferred, and it also sets the *A* sign flip-flop to feed a minus signal out.

Not all three complementers are needed. By proper gating it is possible to use one complementer for several operands. It is also possible to combine the summing and complementing functions in one circuit. Many other arrangements are possible.

If negative numbers are stored in the machine in complement form and positive numbers in absolute-value form, no sign circuits such as those described above are needed. For instance, numbers may be stored according to the following rules.

DECIMAL	ABSOLUTE VALUE PLUS SIGN	2's COMPLEMENT	MACHINE FORM
+4	+0100		0 0100
+2	+0010		0 0010
−2	−0010	1110	1 1110
−4	−0100	1100	1 1100

A ZERO in the most significant position represents a positive number and a ONE in the most significant position represents a negative number in complement form.

The use of this extra bit simplifies addition as can be seen from the following examples.

DECIMAL	ABSOLUTE VALUE PLUS SIGN	MACHINE FORM
+3	+0011	0 0011
+7	+0111	0 0111
+10	+1010	0 1010 = +1010
+3	+0011	0 0011
−7	−0111	1 1001
−4	−0100	1 1100 = −0100
−3	−0011	1 1101
+7	+0111	0 0111
+4	+0100	(1) 0 0100 = +0100 (Disregard carry)
−3	−0011	1 1101
−7	−0111	1 1001
−10	−1010	(1) 1 1010 = −0110 (Disregard carry)

MULTIPLICATION

Binary Multiplication

The following simple example of binary multiplication was given in Chapter 3.

1011	Multiplicand
1001	Multiplier
1011	First partial product
0000	Second partial product
0000	Third partial product
1011	Fourth partial product
1100011	Product

Each partial product is determined by the corresponding multiplier bit: if the multiplier bit is ONE, the multiplicand is set down as a partial product; if the multiplier bit is ZERO, zeros are set down for the partial product. Each successive partial product is shifted one position to the left before it is added.

To mechanize this multiplication, each successive multiplier bit is detected in turn, and the bit is used to open a gate to allow the multiplicand to be added to the contents of the accumulator. After each multiplication

the accumulator contents are shifted right before the following partial product is added; this is equivalent to shifting the following partial product to the left and then adding. The accumulator builds up accumulated partial products until finally it is storing the product. The process is summarized as follows:

1011|	
1001|	Add
1011|	
101|1	Shift right
0000|	Add
0101|1	
010|11	Shift right
0000|	Add
0010|11	
001|011	Shift right
1011|	Add
1100|011	

A 4-bit binary multiplication circuit which operates in this manner is shown in Fig. 144. It consists of four registers, one for storing the multiplier, one for the multiplicand, and two for the product (since 4 bits × 4 bits = 8 bits); an adder; two flip-flops, one to store the multiplier bit being considered at a certain time, the other to control shift and add operations; and a counter to count the number of times shift and add operations are being performed.

Details of operation are as follows. At the start the multiplicand is stored in *DR* and the multiplier in *XR*. The contents of *XR* are shifted right, and the least significant bit of the multiplier is fed to multiplier bit flip-flop. If the bit is ONE, the flip-flop is set; if it is ZERO, the flip-flop is reset.

During the next step, if the multiplier bit flip-flop is reset, the multiplicand is not allowed to enter the adder. If the flip-flop is set, the multiplicand is allowed to be applied to the adder. Simultaneously, the contents of *AR* (0000 at the start) are also applied to the adder. The first partial product is developed in *AR*.

The next step is to shift the partial product in the accumulator to the right. At the same time, the next multiplier bit is shifted into the multiplier

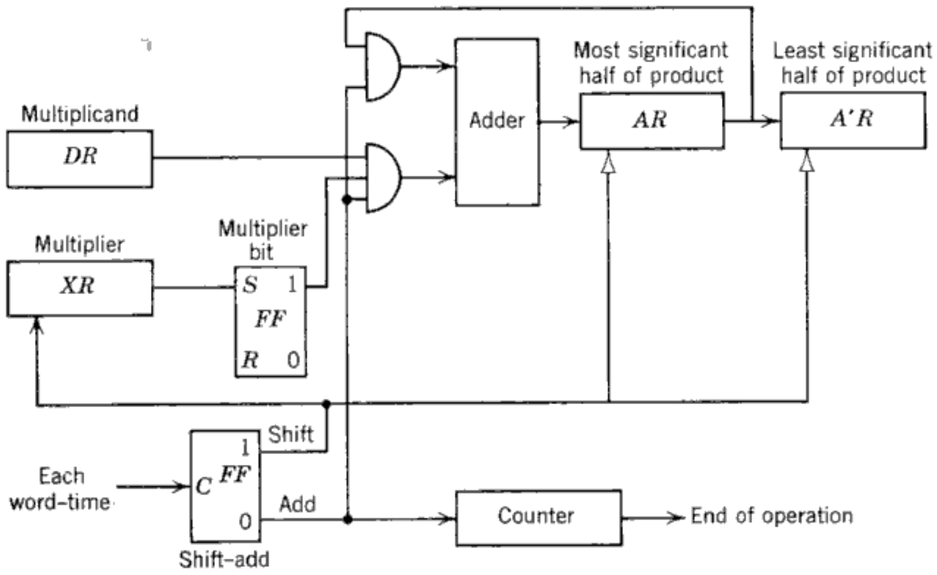


Fig. 144. Mechanization of binary multiplication.

bit flip-flop. During the following step, the multiplicand is added to the first shifted partial product if the multiplier bit flip-flop is set; nothing is added if the flip-flop is reset.

This sequence of shift and add steps is under control of the shift-add flip-flop. During each step a pulse is applied to complement its output. At the start it is set and produces a shift pulse which is applied to *XR* and *AR*. The following pulse resets the flip-flop to produce an add pulse. The add pulse is applied to the adder input gates as shown.

Every time the shift-add flip-flop produces an add pulse, the pulse steps a counter. After a count of 4 signifying that all multiplier bits have been taken care of, the counter produces an end-of-operation pulse.

Figure 145 shows the contents of each register during each step of a multiplication operation.

Note that, during the step where a bit at the least significant end leaves *XR*, a bit enters the most significant end of the *A'* register. When all the bits in *XR* have been shifted out, the *A'* register is filled with bits. Because

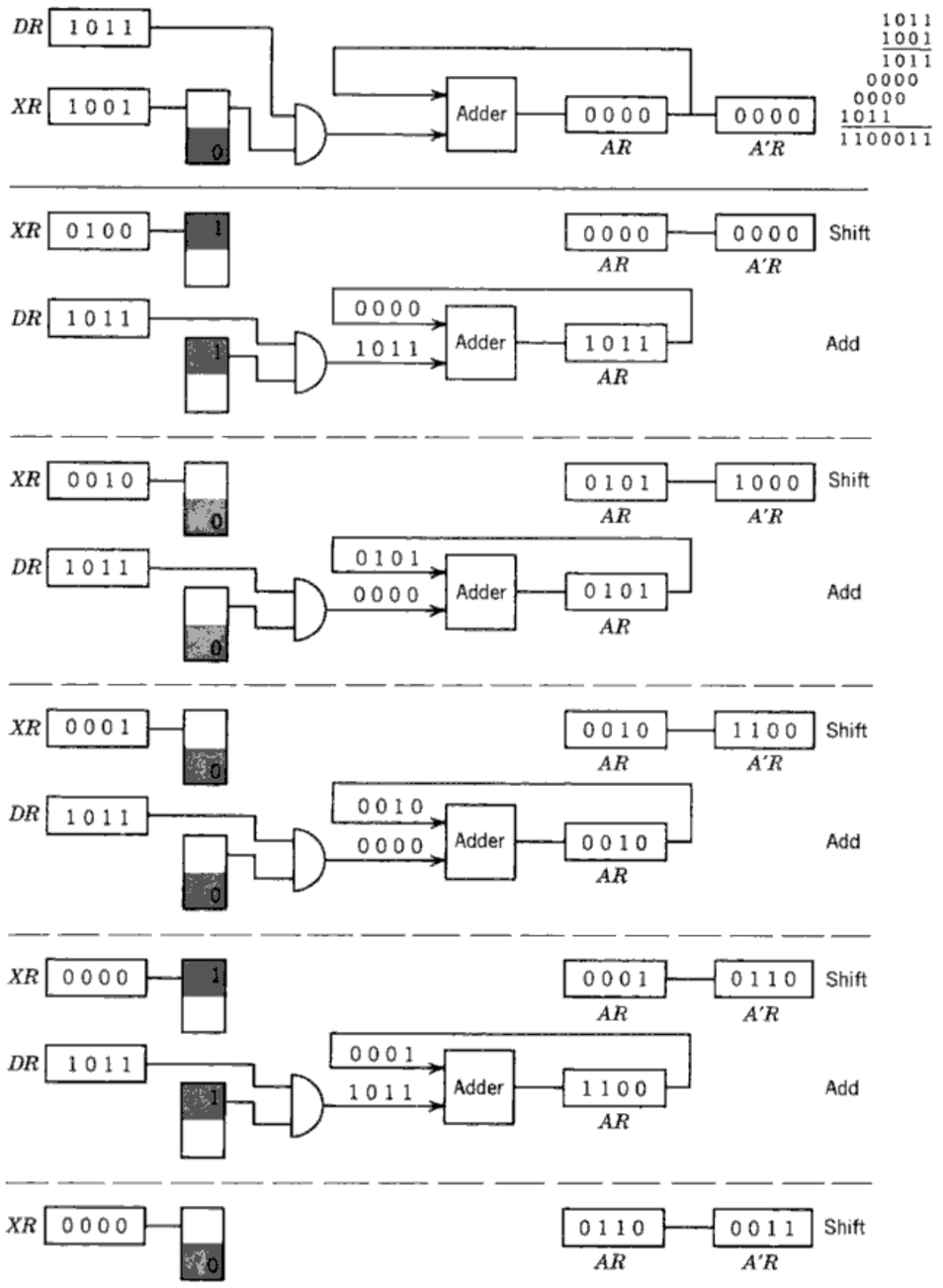


Fig. 145. Steps in binary multiplication.

of this relative timing, XR may double as the A' register. When this is done, only three registers are needed: AR, XR, and DR. Upon completion of the multiplication operation, the most significant half of the product is stored in AR and the least significant half in XR. The simplified multiplication circuit thus looks as shown in Fig. 146.

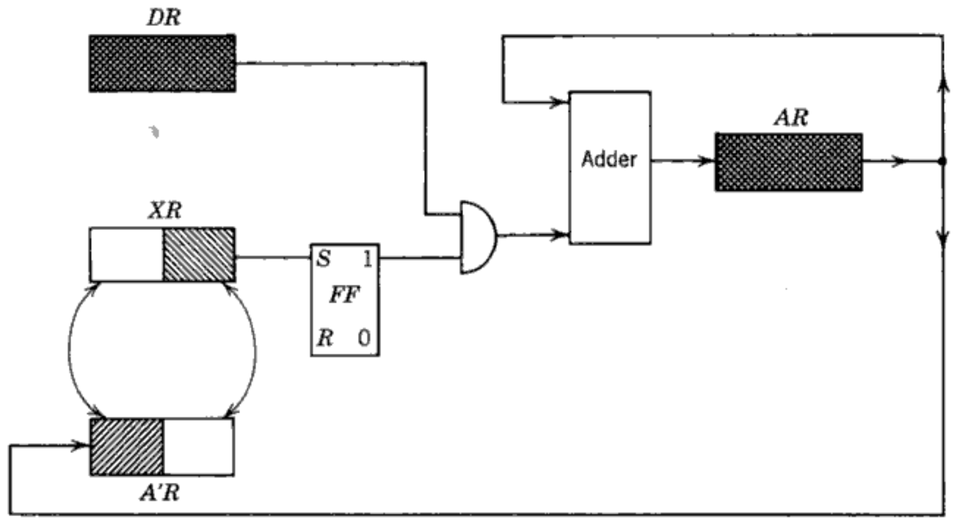


Fig. 146. Storing least significant half of product in XR.

Decimal Multiplication

One of the most common machine methods for performing decimal multiplication is by a process of repeated addition.

Example: $728 \times 451 = 328328$.

MULTIPLIER BIT	REPEATED ADDITION		PARTIAL PRODUCTS
1	728	=	728
5	728 728 728 728 728	=	3640
4	728 728 728 728	=	2912
	328328		328328

When mechanized, the machine performs essentially the same process except that the partial products are accumulated as we go along, and that

right shifts of accumulated partial products rather than left shifts of multiplicands are performed. As in binary multiplication, each multiplier digit, in turn, starting with the LSD, determines whether the multiplicand should be added or not. But there is this difference. In binary multiplication only one addition is involved for each multiplier bit. In decimal multiplication the multiplicand must be added to the accumulated partial

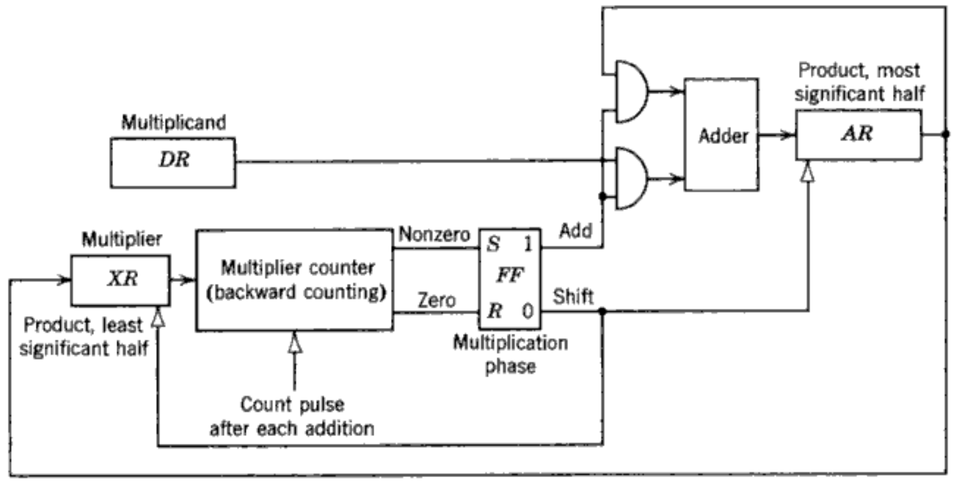


Fig. 147. Mechanization of decimal multiplication.

product a number of times as given by the value of the multiplier digit. In decimal multiplication a counter is needed to keep track of the number of additions performed.

Figure 147 is a block diagram of a decimal multiplication circuit. Note that, except for the backward-counting counter, this circuit is practically the same as the binary multiplication circuit.

This circuit operates as follows. First the multiplicand is stored in DR, and the multiplier in XR. Secondly, the contents of the AR (0 at the start) are shifted right, and the rightmost digit is shifted into the most significant end of XR. At the same time the contents of XR are shifted right, and the LSD is fed to the backward-counting counter.

If this counter is storing any digit but 0 during the next step, it allows the multiplicand (in DR) to be added to the accumulated partial product (in AR). During this addition the counter is stepped down 1. If the counter still does not read 0, the multiplicand is added again to the accumulated partial product; again a pulse applied to the counter causes it to step down 1. In this way the multiplicand is added the number of times given by the multiplier digit.

As soon as the counter reads 0, it resets the multiplication phase flip-flop

which then produces a shift pulse. The shift pulse causes *AR* and *XR* to shift right, and the next multiplier digit to be transferred to the counter.

Now that there is a number other than 0 in the counter, an add phase is introduced. Add and shift phases alternate in this way until a counter (not shown) which is counting shift phases produces an end-of-operation pulse.

DIVISION

The restoring method of division was explained in Chapter 3. Briefly, the method consists of successively subtracting the divisor from the most significant digits of the dividend until a negative remainder (indicating an overdraft) occurs. The divisor is then added to the negative remainder to restore the balance. The number of subtractions, not including the restored one, is the most significant quotient digit. Then the divisor is shifted right and successively subtracted again until a negative balance is produced, after which it is restored. The number of subtractions not including the restored one determines the next most significant quotient digit, etc.

For ease in mechanization, the subtractions are replaced by the addition of complements. The example of restoring division shown on page 319 is the same as the one presented in Chapter 3 except that the addition of complements is shown at each step where a subtraction is called for.

Note that a carry is produced from the most significant position when the remainder is positive, and no carry is produced from the most significant position when the remainder is negative. During the addition of a 10's complement each time a carry is produced from the most significant position, a 1 is added to the value of the quotient digit. Each time no carry is produced, a restoring division followed by a shift operation is introduced. The presence or absence of a carry from the most significant position may thus be used to control the division operation.

A simplified block diagram of a circuit operating on these principles is shown in Fig. 148. It consists of three registers, *AR*, *DR*, and *XR* for storing the dividend, divisor, and quotient; a complementer flip-flop and complementer for complementing the divisor when required; a shift-add flip-flop for flipping between the shift and add phases; and a counter for accumulating each quotient digit. The steps in the division process are as follows:

1. The dividend is fed to *AR*, and the divisor to *DR*.
2. Division is started by resetting the shift-add flip-flop and setting the complementer flip-flop. The first allows addition to take place; the second causes 10's complementation of the divisor before it is added to the dividend. The resultant sum is stored in *AR* as a partial accumulated remainder.
3. If an overflow is produced, it steps the counter and allows another addition of the 10's complement to take place.
4. The counter continues counting until no overflow is produced. The no-overflow condition resets the complementer flip-flop. During

Example of restoring division

```
      1241
365)452980
```

		OPERATION	QUOTIENT DIGITS
	452980		
	635	Add complement	(1)
	(1) 087980		
	635	Add complement	
No carry	722980		
	365	Add to restore	
	(1) 087980		
	635	Shift and add complement	1
	(1) 51480		
	635	Add complement	(2)
	(1) 14980		
	635	Add complement	
No carry	78480		
	365	Add to restore	
	(1) 14980		
	9635	Shift and add complement	1
	(1) 11330		
	9635	Add complement	2
	(1) 07680		
	635	Add complement	3
	(1) 4030		
	635	Add complement	(4)
	(1) 0380		
	635	Add complement	
No carry	6730		
	365	Add to restore	
	(1) 0380		
	635	Shift and add complement	(1)
	(1) 015		
	635	Add complement	
No carry	650		
	365	Add to restore	
	(1) 015		

1241

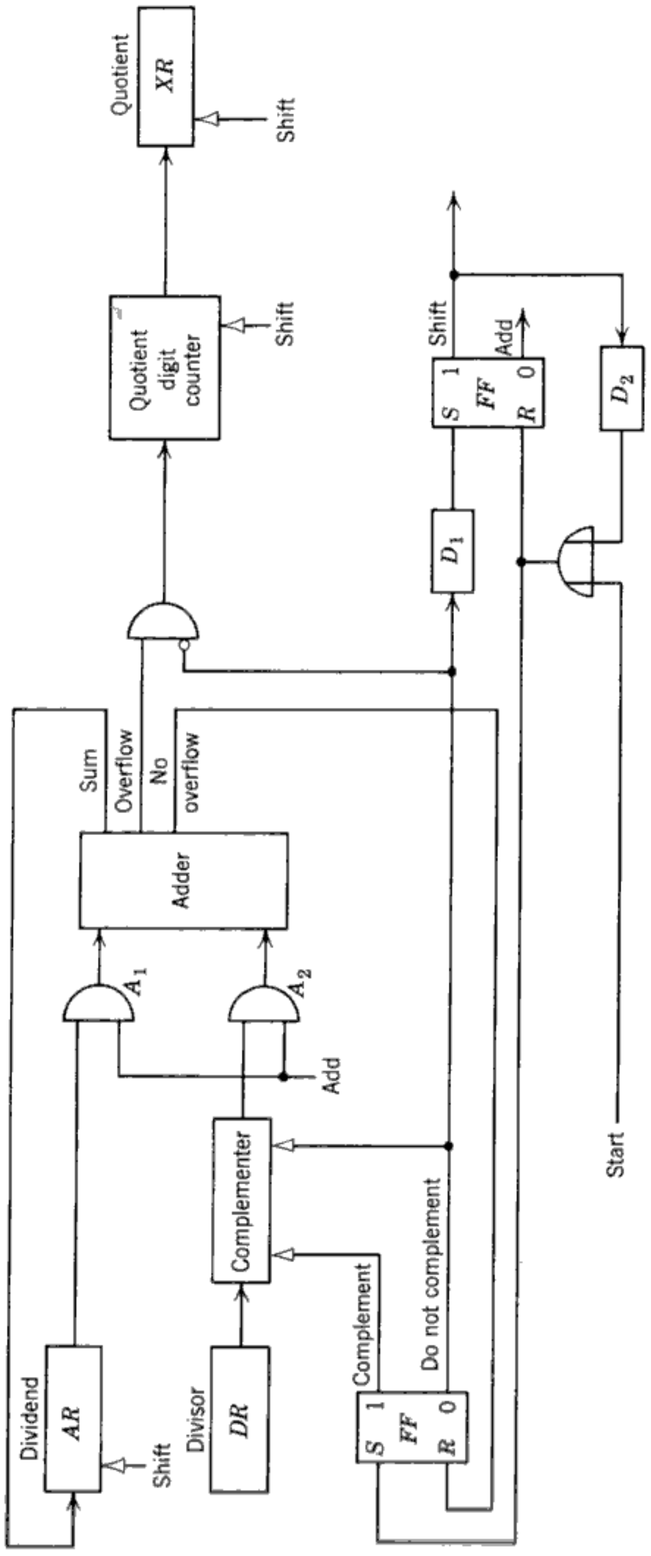


Fig. 148. Mechanization of restoring division.

the next step, therefore, the divisor is not complemented; it is simply added. This restoring addition produces a carry. However, A_3 is inhibited by the reset output of the complementer flip-flop and, therefore, does not step the counter.

5. The reset output of the complementer flip-flop is delayed a word-time by D_1 after which it sets the shift-add flip-flop. The set output causes the quotient in the counter to be shifted into XR, and the accumulated partial remainder in AR to shift left. Shifting the accumulated partial remainder left produces the same relative results as shifting the divisor to the right.

6. One word-time after the shift pulse is produced, the output of D_2 resets the shift-add flip-flop to cause the adder to add again, and it sets the complementing flip-flop to cause the divisor to be complemented before addition. The sequence for developing the next most significant quotient digit is thus introduced.

7. A separate counter (not shown) may be used to count the number of quotient digits developed. When a complete word is produced, the operation stops. At the end, the remainder is stored in AR and the quotient in XR. The last step in some machines is the interchanging of the contents of AR and XR. This is done in order to be consistent with the usual requirement of having useful results available in the accumulator.

EXTRACT OPERATION

The extract operation is a powerful tool for modifying words or for extracting digits from a word. The extract operation is sometimes called a logical multiply instruction because it depends on the characteristics of an AND circuit. What it is and what it does can best be seen by means of examples.

If one register stores an extractee and another register stores the extractor, when an extract instruction is given, the output is as follows:

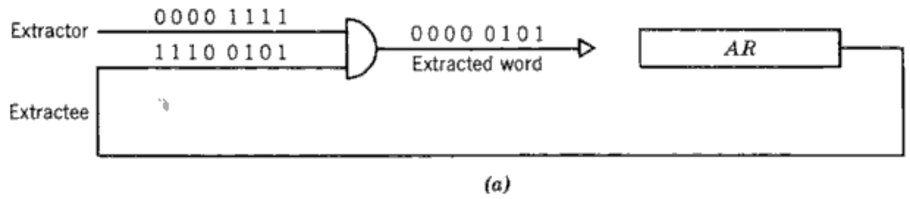
1001001110	Extractee
1010101010	Extractor
1000001010	Result

Note that in each column where a ZERO appears in the extractor, the resultant digit is always 0; in each column where a ONE appears in the extractor, the resultant digit is the same as the corresponding extractee digit. In essence, we have extracted the digit in every other column.

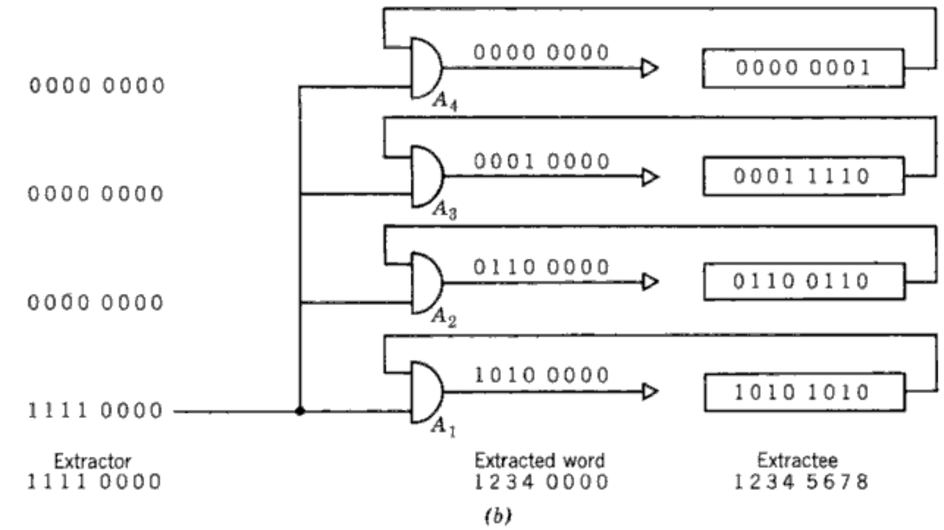
The extract instruction can be used to extract the least significant half

of a word by placing ONE's in the least significant positions of the extractor word:

1001001110	Extractee
0000011111	Extractor
0000001110	Result



(a)



(b)

Fig. 148. Mechanization of extract. (*a*) Binary. (*b*) Decimal.

Similarly, the extract instruction can be used to extract the most significant half of a word by placing ONE's in the most significant positions of the extractor word:

1001001110	Extractee
1111100000	Extractor
1001000000	Result

The extract operation for binary numbers is performed with the aid of a simple AND circuit, as shown in Fig. 149a. The extractee is stored in the *A* register. An extract instruction feeds the bits of the extractor to one side of the AND circuit at the same time that the contents of the *A* register are fed to the other side of the AND circuit. The result—a ONE in all positions where both bits are ONE and a ZERO in all other positions—is fed back to *AR*.

The extract instruction may also be applied to coded decimal numbers. In one arrangement the extracted word consists of all those digits of the extractor in positions where the extractor word has odd digits. In all other positions zeros are present. The reason for this is that, in binary-coded decimal, a ONE appears in the lowest order of all odd digits and a ZERO in the lowest order of all even digits. These lowest order bits may be fed to four AND circuits, one for each subregister, in a manner shown in Fig. 149*b*, in order to perform the extraction.

CHECKING

Transmission of information to and from the arithmetic unit or among arithmetic registers may be checked by redundancy checks, such as the parity check and the forbidden-combination check. The forbidden-combination check may also be used to check the results of arithmetic operations. Obviously, if a forbidden-combination is produced as a result of an arithmetic operation, something has gone wrong with the operation.

Arithmetic operations themselves may be checked in several ways. One way is by means of casting out 9's or casting out 7's. In the casting-out-9's method, as each operand is fed to an arithmetic register, it is also applied to a modulo-9 counter, which thus stores the RMN. At the same time that an arithmetic operation is performed on the numbers, the same operation is performed on the RMN's. On completion of the arithmetic operation the result is fed to a modulo-9 counter, and then comparisons of RMN's are made as follows:

1. In addition: RMN of sum = sum of RMN's.
2. In subtraction: RMN of difference = difference of RMN's.
3. In multiplication: RMN of product = product of RMN's.
4. In division: RMN of dividend = (RMN of quotient) × (RMN of divisor) + RMN of remainder.

Instead of introducing redundancy in the words used, redundancy may be introduced into the machine itself. By duplicating the adder, for instance, we may have the machine perform an addition twice. Now if the two sums are compared, we may determine whether the results of the two operations are alike. If they are, it indicates that no error has occurred. If the sums are different, it indicates that a mistake must have occurred.

Redundancy may be achieved in another manner. The two operations performed may not be the same, but related. For instance, if the operation called for is an addition such as 5 + 4 = 9, then the machine would also

perform a subtraction: $9 - 5 = 4$. The advantage of this arrangement is that the second related operation may be performed after the actual operation is completed; no extra equipment is needed. Machines which are not built to do this sort of checking automatically may be programmed to do it.

Another type of test found in arithmetic units protects the machine against the performance of forbidden operations. For instance, in a machine handling fractional numbers, we cannot divide a larger number by a smaller number because the result would be greater than 1. But suppose such a problem *is* programmed. The machine can be made to detect this condition fairly easily because overflow is always produced when a result turns out to be 1 or greater. This overflow may be used to set an exceed-capacity-overflow flip-flop to warn the user that an illicit operation has been performed.

SUMMARY

1. The arithmetic unit usually consists of a group of registers (*AR*, *DR*, and *XR*) for storing operands, and arithmetic circuits such as adders, complementers, sign-determining circuits, and counters to perform the arithmetic operations.

2. Simple addition of bits may be performed by a coincidence-type adder which produces the appropriate sum and carry bits when an augend, addend, and carry bit are applied. Addition may also be performed by a counter-type adder which operates in a manner analogous to the operation of mechanical calculating machines.

3. Addition of words may be accomplished serially or in parallel. In serial addition, the bits of the augend and addend are applied sequentially to the adder. In parallel addition, a separate bit adder is used to obtain the sum of each pair of bits. In serial addition only one bit adder is required and one word-time is needed to complete the operation. In parallel addition, we require as many bit adders as there are bits to be summed, but the time required is reduced. Parallel addition is not as fast as one would normally suppose because of the time required for carry propagation.

4. Coded decimal numbers are usually added in serial-parallel. Because a carry from the last order bit of a digit is not necessarily a decimal carry, a corrector is made part of the adder.

5. Algebraic addition may be accomplished by converting all negative numbers to their complement, and adding. Sign circuits control the complementing process.

6. Binary multiplication may be accomplished by a succession of add-shift operations. Decimal multiplication may be accomplished in a similar manner, except that a group of additions is followed by a shift; then another group of additions occurs, and then another shift. Decimal division may be accomplished by a succession of subtractions, followed by a restoring addition and a shift.

7. In addition to the parity and forbidden-combination checks, the arithmetic unit may be checked by duplicating circuits, repeating operations by programming, by casting out 9's or 7's, and by detecting forbidden occurrences such as exceed-capacity overflow during division.

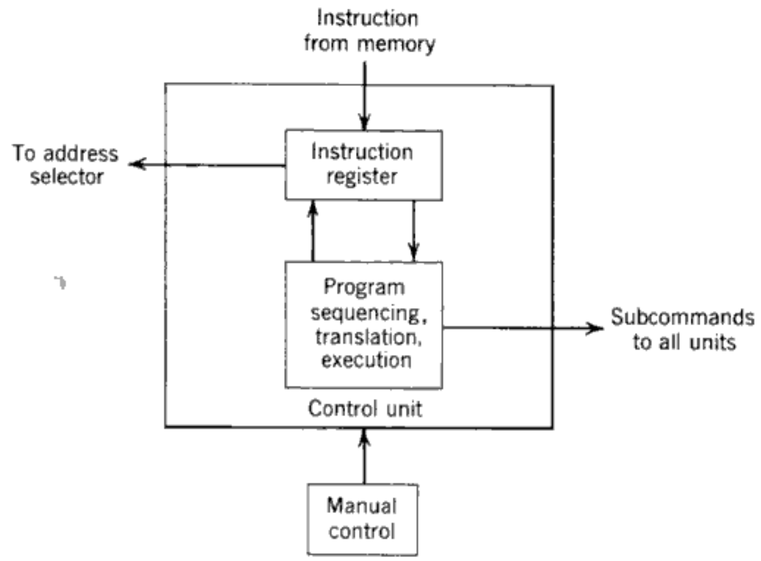
8. *Definitions to Remember*

OVERFLOW—The occurrence of a carry from the most significant position.

EXCEED-CAPACITY OVERFLOW—An overflow that implies that the result of an operation is a number greater than the capacity of a register.

ACCUMULATOR—A register which accumulates totals and stores the results of most operations.

EXTRACT—The process of removing a portion of a word by combining another word with it according to the rules of logical multiplication (AND).



CHAPTER 15

Control unit

The control unit consists of an instruction register which accepts instruction words from memory, and control circuits which sequence, translate, and execute these instructions by means of a group of subcommands.

Sequencing is accomplished by a partnership of the memory and control units in a basic machine cycle which varies with the instruction code used. For one and three-address machines, a program counter determines the address of the next instruction. For two- and four-address machines, the instruction register itself stores the address of the next instruction to be executed. The address is applied to the address selector which, in turn, causes the appropriate instruction to be read out to the instruction register.

Translation is performed by the control unit. Commands are translated into subcommands which are fed to all parts of the machine.

Execution depends on timing pulses as well as subcommands. Control may be transferred to the arithmetic control or to the input-output control circuits for more complicated instructions.

The control unit not only determines the operation of the other functional units; it also determines and can modify its own operation. For instance, sequencing may be modified by means of a branch instruction. The control unit is also capable of modifying an instruction before it is executed.

Manual control is available to allow the operator to intervene, monitor, or modify the automatic operation of the computer.

BASIC MACHINE CYCLE

Automatic control of a digital computer may be divided into three functions as follows:

1. Sequencing of instructions.
2. Translation of commands and addresses.
3. Execution of subcommands.

These functions are performed as part of a basic machine cycle which depends for the most part on the instruction code.

Sequencing of Instructions

In a computer using a one-address or a three-address instruction code, sequencing is accomplished by means of a program counter which provides the address of the next instruction word to be executed, and a flip-flop which divides the sequence into a FETCH phase and an EXECUTE phase. The other major parts of the control unit are the instruction register which stores the command and address of the instruction word, and the address selector (actually part of memory) and command decoder which translate the two parts of the instruction into signals which cause execution of the instruction.

A block diagram of such a sequencing unit is shown in Fig. 150. The following is the sequence it follows. Upon completion of an instruction, the instruction mode flip-flop is set to the FETCH state. In this state, it allows the address stored in the program counter to choose the next instruction word from memory to be transferred to the instruction register. As soon as the word is completely in the instruction register, the instruction mode flip-flop is reset to the EXECUTE state. In this state, gates A_2 and A_3 allow the address part of the instruction to be transmitted to the address selector and the command to the command translator. The address selector chooses the location to be read from or written into. The command translator translates the command into subcommands which execute the

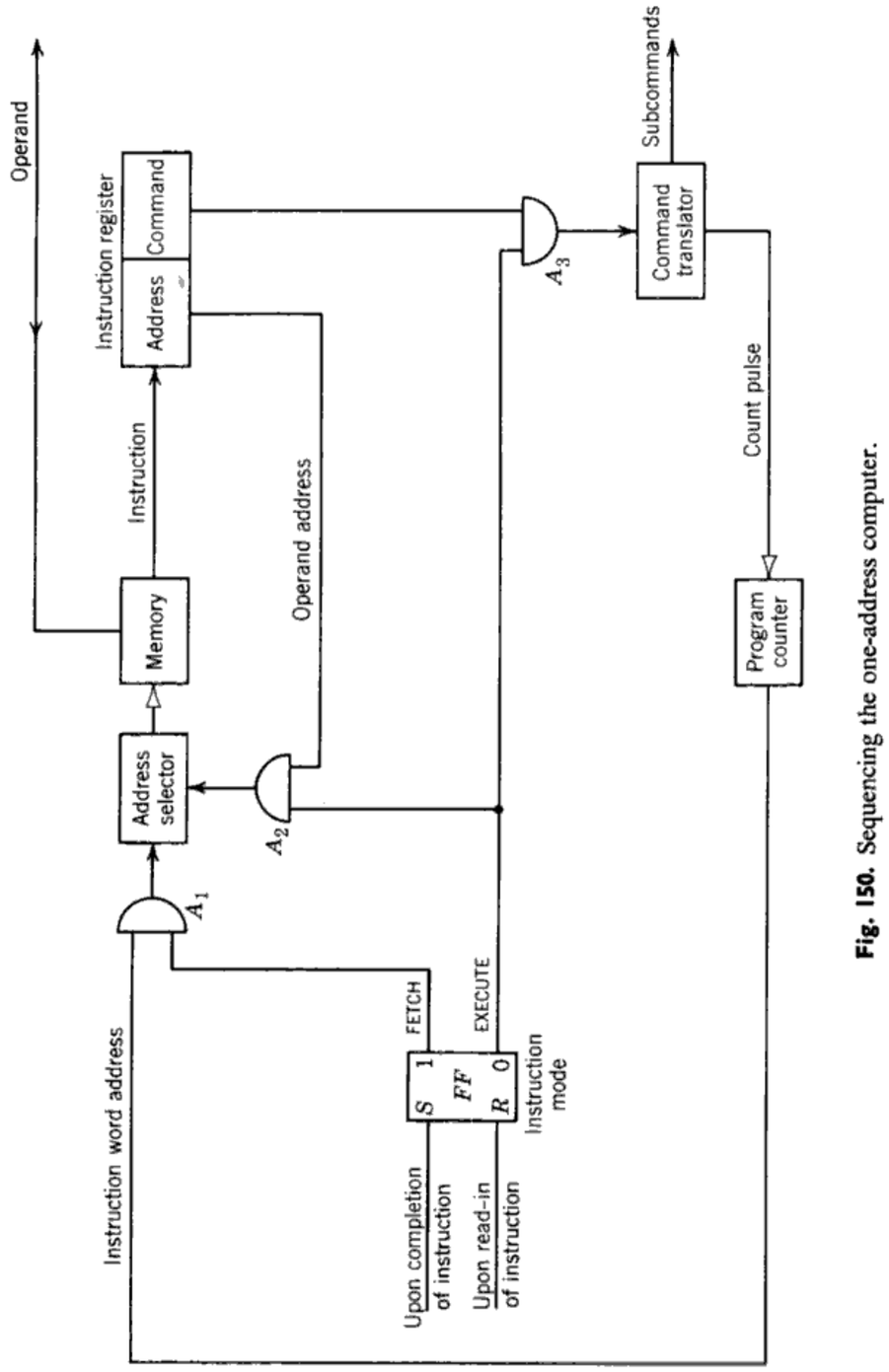


Fig. 150. Sequencing the one-address computer.

instruction on the operand specified by the address. One of the subcommands is a count pulse which steps the program counter to the next address. After execution of the instruction, the instruction mode flip-flop is set to the FETCH state to start a new cycle.

In a machine using a two-address instruction code, no program counter is needed; the address of the next instruction to be executed is given in the instruction word. The two-address instruction code is usually specified with a cyclic access memory. Since in this type of memory time is required for finding an address, the basic machine cycle consists of four steps instead of two. These four steps are:

1. Search for address of operand.
2. Translate and execute subcommands upon operand.
3. Search for address of next instruction.
4. Read next instruction into instruction register.

A modulo-4 cycle counter may be used to sequence through these four steps.

A sequencing unit to be used with a drum memory and a two-address instruction is shown in Fig. 151. Everything above the horizontal dashed line is part of the memory. Everything below the dashed line is in the control unit. Sequencing is performed by the two units working closely together.

The sequence is as follows. When the cycle counter reads 1, the data address is fed to the memory control circuits. The column address is applied to the column address selector which primes the appropriate drum heads. The row address is transmitted to the row address register where it is applied to one side of a comparator.

Upon completion of read-in of row address into the row address register, the cycle counter is stepped to 2. Every word-time the row address of the row currently under the heads is applied to the other side of the comparator. When the two addresses are alike, the comparator sends an activating signal to A_5 and A_6. Gate A_5 allows the operand to be read off the chosen head and to be applied to the arithmetic circuit (information may also go from the arithmetic unit to the memory). At the same time A_6 allows the subcommands to be transmitted to the arithmetic unit where they perform the operation called for.

As soon as the memory's part of the job is complete, the cycle counter is stepped to 3. After the entire operation is completed, an operation-complete pulse is applied to gates A_3 and A_4, and the instruction column address is fed to the column address selector and the instruction row address to the row address register to begin the search for the next instruction.

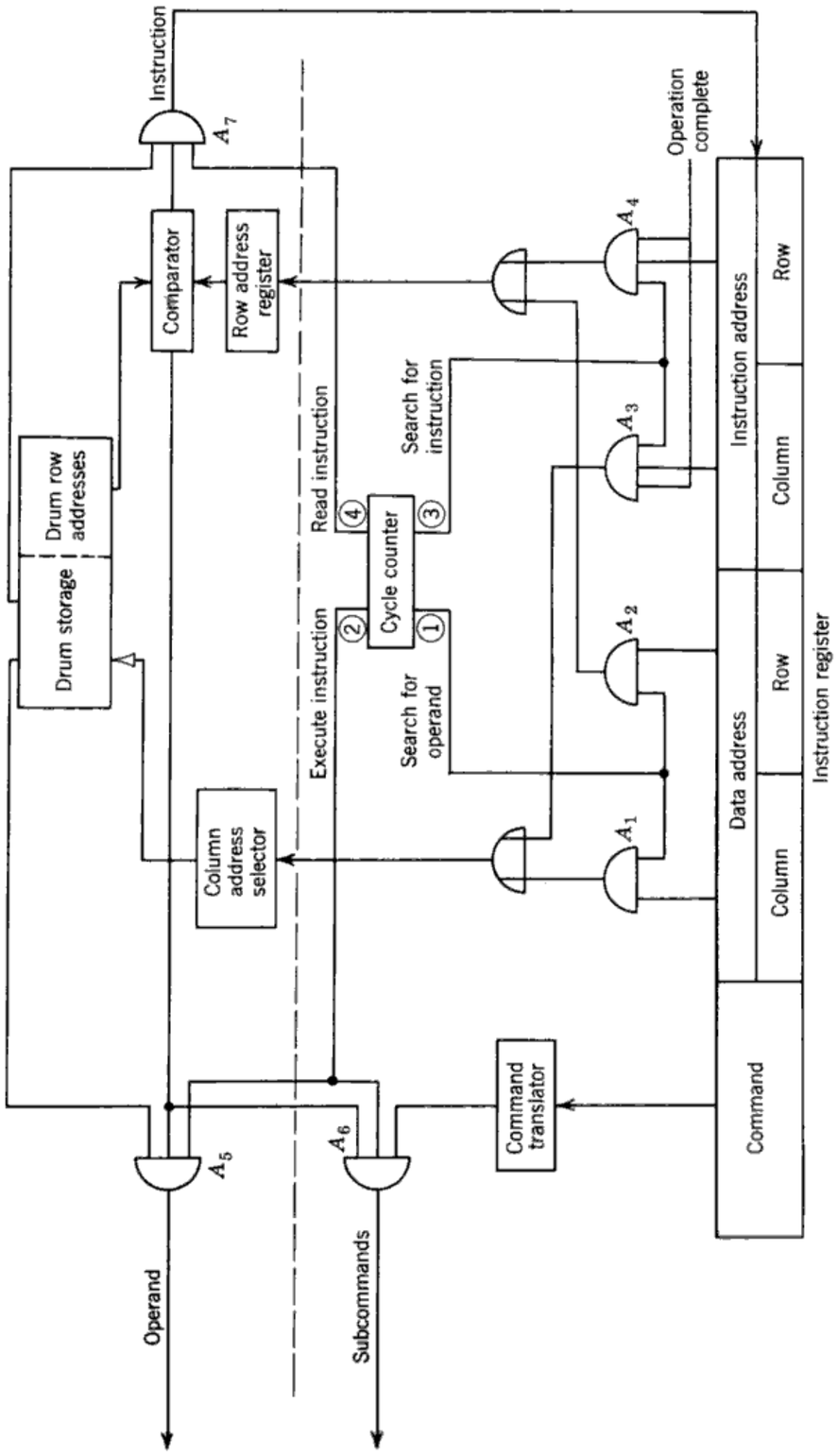


Fig. 151. Sequencing the two-address computer. (Above dashed line—memory; below dashed line—control.)

Upon completion of read-in of instruction row address into the row address register, the cycle counter is stepped to 4. After the comparator produces a coincidence pulse, the instruction word is fed through A_7 to the instruction register. The cycle counter is then stepped back to 1, and the sequence is repeated for the new instruction.

The first two steps in the cycle are concerned with the operand; the second two steps are concerned with the instruction.

Machines using three-address instruction codes are sequenced by means of a program counter. Machines using four-address instruction codes are sequenced by a cycle counter. In both the three- and four-address machines, more steps must be included to take care of the two operands and the result.

The machine cycles given above apply for simple transfer, arithmetic, and input-output instruction. If the command calls for a lengthy operation such as multiplication, the execution step may be extended by transferring control to arithmetic control. If the command calls for a lengthy input-output instruction such as the writing of a whole block of words into memory, the execution step may be extended by transferring control to the input-output control. In both cases, upon completion of the execution sequence control is transferred back to the control unit which continues in a normal manner. (See Fig. 152.)

Translation of Commands and Addresses

As soon as the instruction word appears in the instruction register, the command signals are applied to a command decoder which produces one output line for each code combination. This output line is then fed to an encoder which produces the subcommands needed to perform the instruction.

Different subcommands are encoded for different instructions. For instance in a "read from memory to *AR*" instruction, the following subcommands may be needed:

1. Open read amplifiers.
2. Open input gate to *AR*.
3. Shift word into *AR*.
4. Reset *A* register sign flip-flop (to read ZERO unless a minus pulse flips it to ONE).
5. Operation-complete pulse,
6. Block recirculation gate of *AR*.

Any one subcommand may be used by several commands. For instance, a right-shift subcommand may be part of a right-shift instruction, and it may also be part of a multiplication instruction.

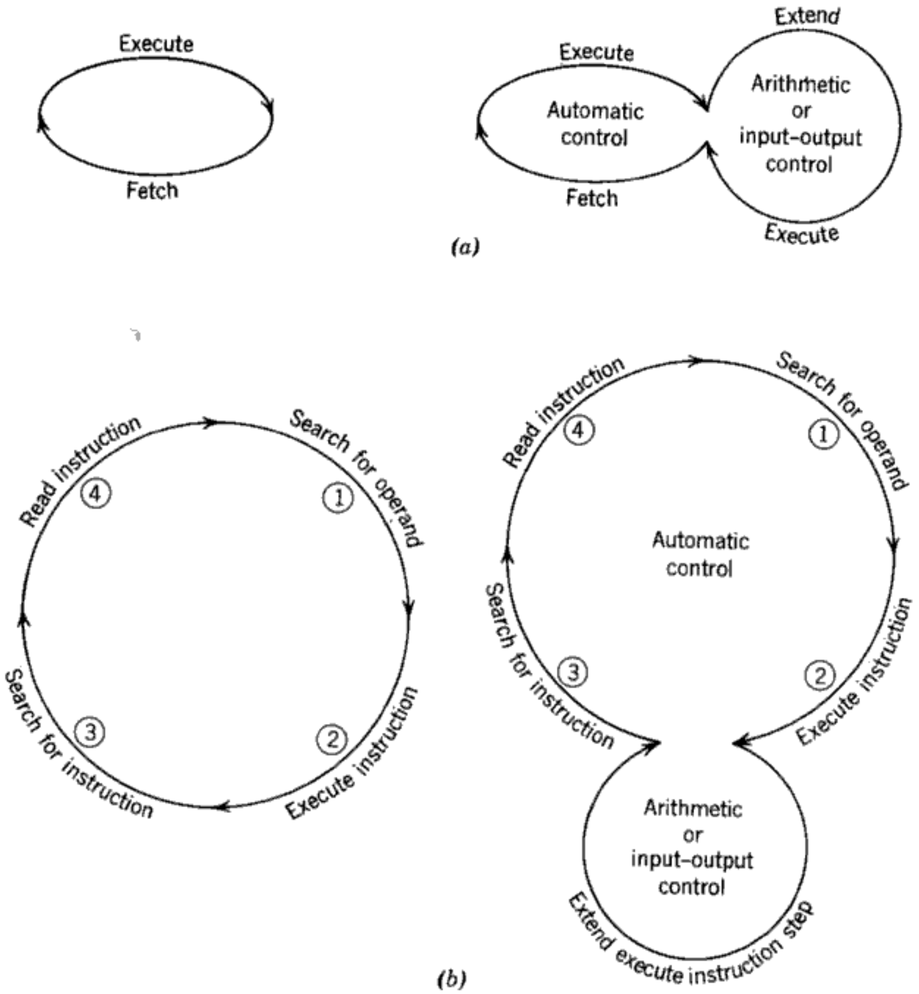


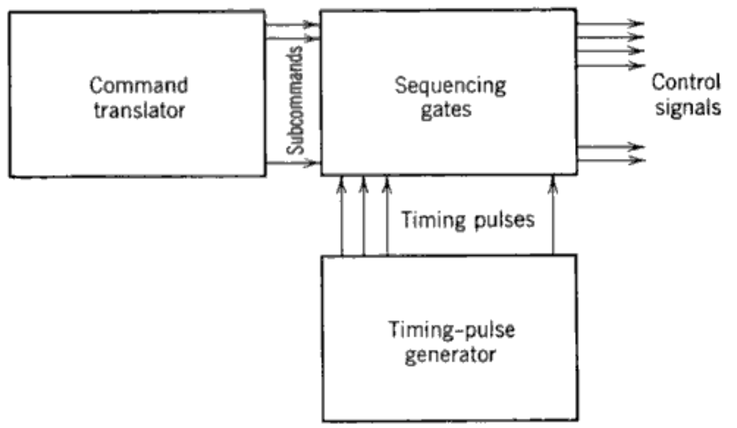
Fig. 152. Extending the machine cycle. (*a*) One-address instruction. (*b*) Two-address instruction.

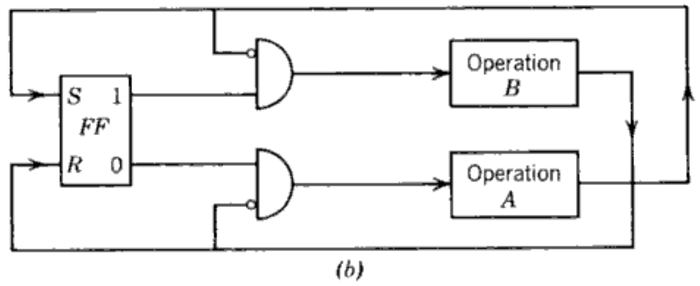
To summarize, the list of subcommands represents the available atomic machine operations, and each command is translated into the group of subcommands needed to perform the indicated operation.

At the same time that the command translator is producing subcommands, the address selector is translating the address part of the instruction into a signal which activates the appropriate location in memory.

Execution of Subcommands

To be effective, subcommands must be executed at the proper times. For instance, in the above example, if the sign bit is the first one





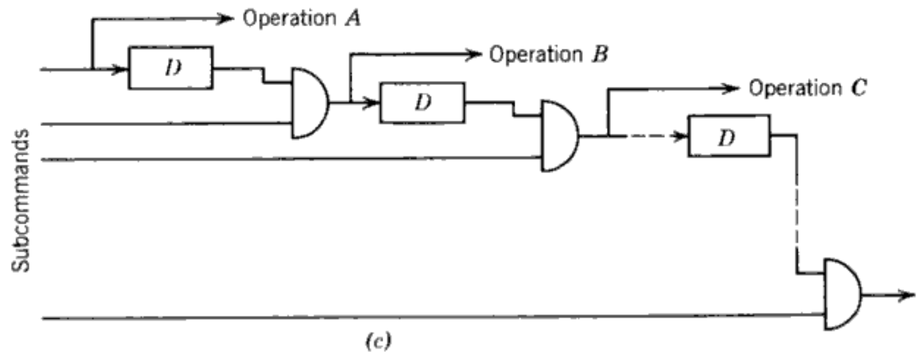


Fig. 153. Synchronous and asynchronous operation. (*a*) Synchronous. (*b*) Asynchronous with flip-flops. (*c*) Asynchronous with delays.

transferred, the sign bit would be allowed to enter the sign flip-flop at time t_0. But the gate to AR would not be opened until time t_1, allowing all the other bits to enter AR. Similarly, the operation-complete pulse is usually not allowed to operate until the end of the word-time.

Execution of subcommands may be performed in one of two ways: synchronously or asynchronously. In the synchronous machine (Fig. 153*a*)

a timing-pulse generator produces a set of timing pulses for each cycle. A subcommand is gated with a timing pulse in a sequencing gate to produce a control signal effective at the appropriate time. These sequencing gates may be scattered throughout the machine. In the asynchronous machine a central timing-pulse generator is not used. As soon as one operation is completed, the next is allowed to take place. Asynchronous operation is easily accomplished by means of flip-flops in circuits such as that shown in Fig. 153*b*. With the flip-flop reset, operation *A* is performed. Upon completion of operation *A*, a signal sets the flip-flop to allow operation *B* to be performed. Asynchronous operation may also be accomplished by means of delay elements in a circuit such as that shown in Fig. 153*c*.

Very few machines are entirely synchronous or entirely asynchronous. Most of them use a combination of both techniques. In many machines it is difficult to separate the subcommands from the control signals.

In many computers, there are several instructions which may be performed in one word-time or in some other standard length of time. The timing is usually set up to take care of these situations easily. When a more complicated instruction such as a multiplication requiring repeated addition is called for, one of the subcommands may transfer machine control to the arithmetic control unit. Basic subcommands are fed to the arithmetic control, where a counter, flip-flops, complementing and other circuits are in control of sequencing the steps of the operation. Upon completion of the operation, the operation-complete pulse transfers control back to the control unit. A similar operation occurs when executing a block transfer into or out of the computer.

Nonmemory Instruction

A nonmemory instruction refers to an instruction which does not call for an operand to be applied to or removed from the memory. The bit positions of the instruction word usually holding the address may contain bits to modify the command part of the instruction. For instance, in a right-shift instruction, the address part of the instruction may be used to specify the number of shifts to be performed. During translation of "right shift," the machine produces a subcommand which prevents the contents of the address part of the instruction from entering the address selector and causes it to be applied to a counter instead. This counter controls the number of places to be shifted.

The nonmemory instruction does not alter the basic machine cycle. It merely interprets some of the signals in a different way.

MODIFICATION OF BASIC MACHINE CYCLE

The basic machine cycle may be modified by a branch or halt instruction. The branch instruction may cause the following instruction not to be in the normal sequence. The halt instruction may stop the instruction sequencing entirely.

Branch Instructions

The simplest type of branch instruction is the unconditional jump. The unconditional jump instructs the computer to choose the next instruction from the address specified in the instruction word. The mechanization is fairly easy. When an unconditional jump command is translated by the machine, a subcommand is fed to the address selector to prevent the address portion of the instruction word from affecting the memory. Instead, a different subcommand gates the address into the program counter. After the operation-complete pulse is received, the computer sequences to the instruction stored at the address present in the program counter. Instructions sequence normally from this new address until another branch instruction is executed.

An unconditional jump is used in one- and three-address machines. It is not needed in two- and four-address machines since, in effect, an unconditional jump is part of every instruction word.

A more sophisticated branch instruction is the conditional jump which allows the computer either to sequence normally or to jump to an instruction out of sequence based upon conditions present within the machine. In other words, the conditional jump chooses among alternatives, and it makes its decision on the basis of some important fact.

Some conditions commonly used by digital computers as criteria for deciding whether to perform a jump or not are as follows:

1. Occurrence of a minus accumulator as a result of a previous arithmetic operation.
2. Occurrence of overflow as a result of a previous arithmetic operation.
3. The presence of a special symbol such as an *EB* (end of block).
4. The presence of an interlock.

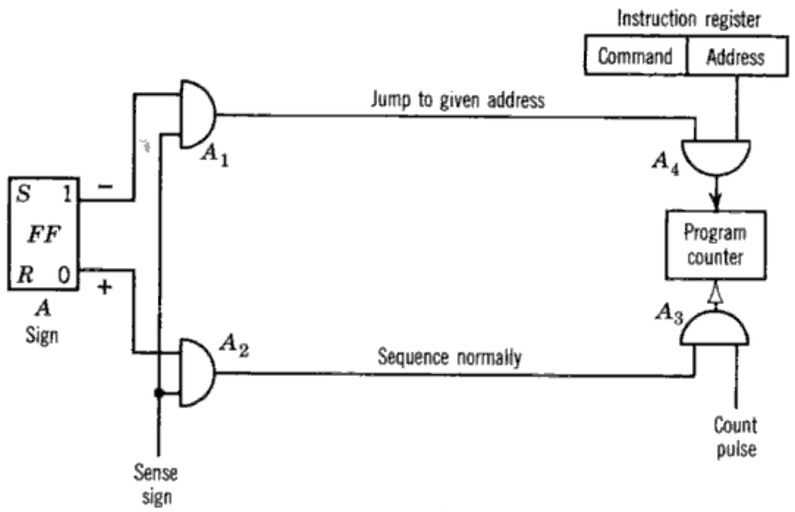
As a matter of fact, any yes-no condition which may be stored in a flip-flop may form the basis for a decision to perform or not to perform a jump.

In a one- or three-address computer, the conditional jump instruction may read as follows:

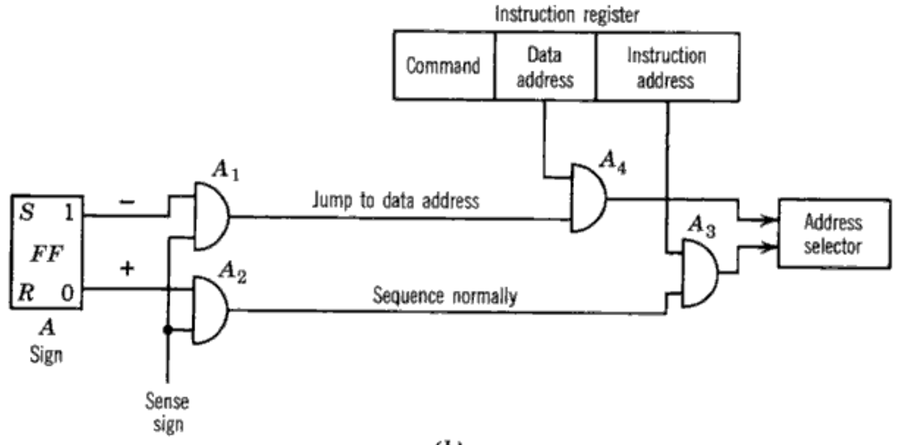
Look at the accumulator sign flip-flop. If it is storing plus (flip-flop

reset), sequence normally. If it is storing minus (flip-flop set), sequence to instruction located at address given by instruction word.

The mechanization is accomplished as shown by the simplified block diagram in Fig. 154*a*. The command is translated into several subcommands



(a)



(b)

Fig. 154. The jump instruction. (*a*) One-address instruction. (*b*) Two-address instruction.

one of which is the sense sign. This subcommand senses the state of the accumulator sign flip-flop. If it is plus, A_2 sends a signal to A_3 to allow a count pulse to step the program counter to the next address. If it is minus A_1 sends a signal to A_4 to allow the address part of the instruction to be fed to the program counter.

In a two- or four-address computer, the conditional jump may read as follows:

> Test the contents of the accumulator sign flip-flop. If it is storing plus (reset), choose next instruction from location specified by instruction address. If it is minus (set), jump to instruction location specified by data address.

A simplified block diagram of a possible mechanization of this instruction is shown in Fig. 154*b*. Again, a sense sign subcommand looks at the state of the accumulator sign flip-flop. If it is reset, A_2 sends a signal to the group of gates A_3 to allow the instruction address to be fed to the address selector. If the flip-flop is set, a signal from A_1 opens group of gates A_4 to allow the data address to be fed to the address selector.

The conditional jump instruction may be used together with a subtraction instruction to compare numbers. One number B is subtracted from the other number A. If the difference $A - B$ is negative, it implies that B is the larger. The subtraction instruction is followed by a conditional jump—a jump if minus—instruction. Thus a jump occurs if B is larger than A.

The subtraction and jump on minus instructions may be combined into one comparison instruction. For a one-address machine, the comparison instruction may read as follows:

> Compare the word in the accumulator with the word stored in the location given by the address. If the operand in the address is less than or equal in magnitude to the word present in the accumulator, perform the next instruction in sequence. If the operand is greater in magnitude than the value present in the accumulator, jump to the second following instruction.

The skip of one instruction is easily accomplished by feeding two count pulses to the program counter instead of one.

In a two-address comparison instruction, the data address specifies the operand, and the instruction address specifies the location of the next instruction if the operand is smaller or equal in magnitude to the accumulator contents. If the operand is larger than the contents of the accumulator, the machine jumps to a specified address.

The sampling input-output instruction referred to previously acts in a manner similar to the conditional jump instruction. The instruction looks at the state of an interlock flip-flop in the input-output. If the flip-flop indicates, for instance, that the input register is not full, no transfer occurs and sequencing continues in a normal way. If the flip-flop indicates the register is full, a transfer occurs followed by a jump.

Halt Instructions

The basic machine cycle may be modified by a simple halt instruction. A halt instruction is needed because, if left to its own devices, the computer would sequence indefinitely. Usually when a halt instruction stops computer operation, a lamp on the panel lights up to indicate this fact.

A more sophisticated type of halt instruction may be built similar in operation to the conditional jump. The instruction senses the state of a flip-flop. If it is reset, sequencing occurs normally. If it is set, the computer halts.

INSTRUCTION MODIFICATION

The instruction word itself can be modified within the machine. Since the instruction word has the same appearance as a data word, words may be added to or subtracted from the instruction word to produce new instruction words. Two common means for modifying instructions are add address instructions and *B* Register instructions.

The Add Address Instruction

The purpose of the add address instruction and its mechanization can be made apparent by means of an example. Suppose that a jump instruction is to be executed. However, the address of the instruction to which we want to jump is unknown at the time of preparation of the program; this address is determined as a result of a previous calculation.

The required jump instruction is constructed by feeding a pseudo-instruction word into the accumulator, such as

11 0000

where 11 represents a jump command, and 0000 means that the address part of the instruction is unspecified. To this word is then added a word such as 00 1001. The result is

$$\begin{array}{r} 11\ 0000 \\ +00\ 1001 \\ \hline 11\ 1001 \end{array}$$

which means "jump to address 1001." This new instruction word may be stored at any point in the program by means of a write instruction.

The add address instruction may be used with other instructions besides

jump. It may be used together with *one* add instruction to cause the computer to accumulate a long list of numbers. For instance, if a group of 500 numbers to be accumulated is stored in addresses 5001 to 5500, it is not necessary to write

Add 5001
Add 5002
Add 5003
Add 5500

We may write "add 5000" in a certain spot in memory. Before this instruction is executed, however, it is fed to *AR* where 000001 is added to it. Then this new instruction is stored back in memory so it can be executed next. The result of this operation is that the contents of 5001 are added to the accumulator contents. Next the "add 5000" is returned to the accumulator and 000002 is added to it, after which the revised instruction is fed back to a location in memory so that it can be executed next. Upon completion of the second accumulation, the process is repeated except that now 000003 is added to the original instruction, etc.

Two questions arise in our minds with reference to the above program: (1) How do we obtain each time a number to be added which is one greater than the previous number? (2) How do we stop this repetition?

The first problem is solved by means of a few instructions which cause a 1 to be added to a number in memory and then the new number is fed back to the same location. In effect, the memory location acts as a counter, with each count being performed by means of an add instruction.

The second problem, that of stopping the constant repetition of this group of instructions, is accomplished by means of a comparison instruction. The comparison instruction compares the contents of the counter with the number of operations required to be performed. If the counter value is less than the number required, the program jumps back to the first instruction in the routine. If the counter and required values are the same, the instruction located in the following address is performed.

The simplified program is as shown in Fig. 155. This type of routine is called a program loop since the same instructions are repeated over and over. A comparison instruction is used to jump out of the loop.

BR Instructions

The counter simulated by the program above may be replaced by an actual counter, which is commonly called an index register, a *B*-box, or a *B* register (*BR*). Each time the add instruction is read by the control, it automatically adds the contents of *BR* to the address part of the instruction before the instruction is translated and executed.

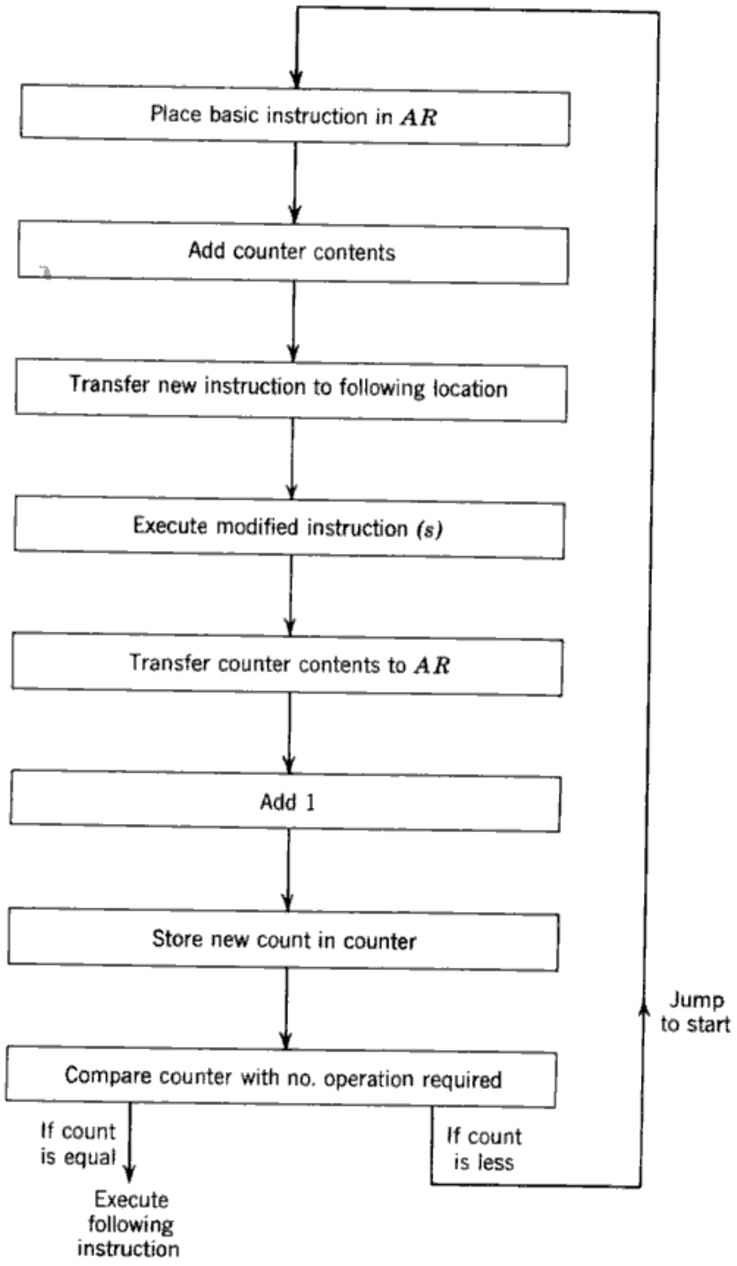


Fig. 155. Simplified program loop.

Later, instead of a comparison instruction, we have a test *BR* instruction which states:

> Compare contents of *BR* with contents of memory location specified by the address part of the instruction. If the contents of *BR* are less than the contents of the memory location, add 1 to the count in *BR* and jump to start of the routine. If the contents of *BR* are equal to the contents of the memory location, execute the following instruction.

The test *BR* instruction performs the same function as the comparison instruction plus the instructions required to produce the count.

The mechanization of a *BR* modifiable instruction is simple. An instruction which is *BR* modifiable may be distinguished from ordinary instructions by means of a code. A simple way of distinguishing it is by placing a ONE in an extra bit position to indicate a *BR* modifiable instruction. When read by the computer, the control unit uses this ONE to open gates to allow the address bits of the instruction register and the bits of *BR* to be applied to an adder. The result—the modified address—is then fed back to the instruction register. After this, the instruction is translated and executed in a normal manner.

Before a program loop is entered, the counter must be set. When constructing a loop with the aid of the add address instruction, the counter may be set to 0 by clearing the accumulator, and then feeding the zeros to the counter in memory. When constructing a loop with the aid of *BR* modifiable instructions, a new instruction called the "set *BR*" instruction is required. The set *BR* states:

> Transfer the contents of the location specified by the address into *BR*.

The *BR* is set to 0 by transferring a 0 to it. By the same token *BR* may be set to any other value.

In any one loop any number of instructions may be made *BR* modifiable, and instructions which are not *BR* modifiable may be interleaved with instructions that are. Several *BR* may be built into one machine, and any one instruction may be made modifiable by any one of the *BR*.

MANUAL CONTROL

The automatic control unit may be monitored and modified by manual controls and indicators located on the control panel. These controls and indicators may be classified according to function as follows:

1. Operational.
2. For loading.

3. For program debugging.
4. For trouble shooting.

Controls and indicators fitting each of these classifications are discussed briefly below.

Operation Controls and Indicators

The most obviously needed controls are those associated with turning the equipment on and off. Separate controls are usually provided for turning on and off the main a-c power; for turning on and off the d-c power supplies feeding the switching, storage and amplifying circuits; and for starting and stopping the actual sequencing and execution of the program.

During the turn-on procedure, indicator lamps are available to warn the operator against activating certain switches until after appropriate delays. Other lamps indicate which components of the equipment are receiving needed voltages. Fuses and circuit breakers provide needed protection for the various circuits.

Controls are available for controlling the operation of external devices. Together with these controls are indicators showing which input or output device is in use.

An important group of controls and indicators is concerned with manual input and visual output display. By means of switches it is possible to feed any combination of bits into almost any of the computer registers. Neon displays show the operator the word he is entering into the machine. Selector switches enable him to choose the register to which the word is transferred. Also, normal computer operation may be halted and the contents of the arithmetic registers displayed.

Another important group of controls is concerned with the determination of the mode of operation of the computer. In some machines there is a control placing the computer in the test mode. In this condition, all the parts of the machine may be tested to make sure the computer is ready to execute programs. The computer may be set to accept inputs only from manual switches, or it may be set to work automatically from the program. There may be other modes of operation which depend on the design of the individual machine.

Loading

A stored program computer sequences and executes instructions stored in memory. The process of bringing the program into memory is called loading. The loading procedure starts with the transcription of the

program onto the external storage medium. This is followed by a transfer of all the instructions from external storage to appropriate addresses in memory.

But how can the computer transfer words into memory when no instructions are present in the machine for accomplishing this transfer? The transfer is accomplished with the aid of panel switches which may be used to simulate instruction words. Instruction words are constructed which cause the program to be entered into memory.

A more efficient method of loading is to place a loading routine at the beginning of the program to be stored in memory. The instructions simulated by means of the panel switches bring the loading routine into memory and then transfer control to this routine. The loading routine then proceeds to load the rest of the program at machine speed.

Program Debugging

There is so much detail work in the preparation of a useful program that rarely is the program correct the first time it is tried. This is why most computers have controls whose purpose it is to expedite debugging—the finding and removal of mistakes from the program. Among the more commonly used controls are the one-instruction, breakpoint, and halt switches.

The one-instruction switch allows only one instruction of the program to be executed at a time. It causes an instruction to be fed to the instruction register and then to be executed. The machine stops immediately after execution of this instruction. Upon completion of this instruction, the contents of the arithmetic and instruction registers are displayed on the panel to enable the operator to see whether the instruction has performed as intended. As soon as the one instruction switch is activated again, the next instruction is fed to the instruction register, executed, and the results of that operation are displayed. In this way, the program may be executed in slow motion.

Two of the most useful controls for testing a program are the breakpoint switch and the jump–no-jump switch. A breakpoint switch allows the computer to operate normally until the program reaches a key instruction at which point the computer stops. The key instruction may be of any kind, but it is most often a branch instruction. At a branch instruction a decision must be made whether to follow one set of instructions or to follow another set. It is a spot where mistakes in programming can be easily spotted. It is a convenient point for breaking the program to view register contents.

A breakpoint switch may have two positions—stop and proceed.

When the switch is placed in the stop position, it sets a flip-flop; when in the proceed position, it resets the flip-flop. When the branch instruction occurs, it produces a subcommand which looks at this flip-flop. If it sees the flip-flop set, it causes the computer to stop; if it sees it reset, it performs a normal branch operation.

The jump–no-jump switch affects the jump flip-flop. In the jump position, the jump flip-flop is set. In the no-jump position, the jump flip-flop is reset. The switch overrides whatever is placed in the flip-flop as a result of normal operation.

During program debugging the breakpoint switch is used in conjunction with the jump–no-jump switch as follows. The breakpoint switch is put in the stop position, and the computer is placed into operation. The computer performs the program until it reaches a branch instruction, at which point it stops. The operator then looks at the contents of the registers as displayed on the panel. The displayed results indicate whether the machine will jump or not during the next instruction. However, the operator may modify the operation by activating either the jump or no-jump positions of the switch. In the first case the machine continues with a jump, and in the second case it continues with no jump regardless of what the branch instruction says. In effect, the computer can be made to execute a different program from the one originally written. A machine may have several breakpoint switches, each able to control a different group of instructions.

Another control which is useful in program debugging is the halt switch. The halt switch can be made to act like a breakpoint switch. At various points in the program a conditional halt instruction is placed. The conditional halt instruction senses in what position the halt switch is placed—by looking at an associated flip-flop. If the control panel switch says halt, the computer halts; if it does not say halt, computer operation continues normally.

Greater flexibility may be achieved by using several halt switches: halt 1, halt, 2, halt 3, and so on. Each halt instruction may be encoded to be sensitive to only one of the halt switches. Thus a halt-2 instruction causes the computer to halt only if the halt-2 switch is activated.

Trouble Shooting

All the controls and indicators used for operation, loading, and program debugging are useful for trouble shooting. Power-operational controls and indicators enable the operator to see if there is any trouble in the power lines or in the power supplies. Fuses and circuit breakers give further indication. Program-operational controls and indicators enable the operator to judge whether the program is being executed properly. If he has any doubts, he may use some of the loading controls to feed information into various places in the machine, operate on them, and then view results. With the program-debugging controls he may dissect the program and view the results at any point.

Several fault and error lamps are available on most control panels. If an error occurs, a lamp lights, indicating in what part of the computer the trouble has occurred. If something goes wrong with one of the external devices, a lamp lights, indicating which device is inoperative.

In some machines a marginal checking device is available for checking all major active elements. Marginal checking is more of an aid in trouble prevention than in trouble shooting.

Because the computer can read, write, and manipulate numbers, the cause of trouble may be obtained by means of a program called a "diagnostic routine." In essence, this program goes through the many operations that the machine is capable of and tests results. Some computers have such diagnostic routines built into the hardware; these routines are entered into automatically when an error signal occurs.

SUMMARY

1. The control unit consists of an instruction register and of automatic and manual control circuits.

2. The automatic control determines the basic machine cycle which consists of sequencing the instructions, translation of commands and addresses, and execution of subcommands. Sequencing is accomplished in cooperation with the memory unit. For complicated instructions the execution stage may be extended by the arithmetic or input-output controls.

3. A one-address computer is sequenced by a program counter. A flip-flop may be used to step from the FETCH phase, in which the instruction word is read from the memory to the instruction register, to the EXECUTE phase, in which the address and command are translated and the subcommands are executed on the operands. Sequencing of the three-address computer is similar.

4. A two-address computer is sequenced by means of a four-stage cycle counter. The four stages are as follows:

a. Search for operand.
b. Execute instruction.
c. Search for next instruction.
d. Read next instruction into instruction register.

Sequencing the four-address computer is similar.

5. The instruction word consists of a command and an address or a group of addresses. In the translation stage the address portion(s) of the instruction is fed to an address selector which allows the path to the chosen location in memory to be opened; and the command is fed to a command translator which produces a subcommand for each of the small operations required to execute a complete instruction.

6. During the execution phase the subcommands are converted to control signals occurring at the appropriate times for the execution of the required instruction.

7. The basic machine cycle assumes an instruction requiring reference to memory. If the instruction does not require such reference, the address part of the instruction may be vacant or its contents may be sent to a special encoder which produces additional subcommands.

8. The basic machine cycle may be modified by a branch instruction. In the one- or three-address computer, a jump to an instruction out of sequence occurs by sending the address part of the instruction to the program counter. In the two- or four-address computer, a jump occurs by feeding the data address, instead of the instruction address, to the address selector.

9. An instruction may be modified by programming the addition of a number to the address part. An instruction may be modified more easily by having it *BR* modifiable, which means that before the instruction is executed the contents of a *BR* are added to the address part of the instruction.

10. Manual controls and indicators are provided to allow the operator to start and stop computer operation, to load the computer with data or with a new program, to help during the debugging of the program, and for trouble shooting.

11. *Definitions to Remember*

SUBCOMMAND—One of the many operations that must be performed in order to complete a command. *Example:* opening a gate, or resetting a flip-flop, or causing a register to shift its contents to the right.

SYNCHRONOUS COMPUTER—A computer whose subcommands are executed at times controlled by a master clock.

ASYNCHRONOUS COMPUTER—A computer whose subcommands are not executed according to times from a master clock; one operation begins after the previous operation is completed.

BRANCH INSTRUCTION—An instruction which may cause the machine to choose the next instruction out of the usual sequence. *Example:* the unconditional and conditional jumps and comparison instructions.

UNCONDITIONAL JUMP—An instruction which always causes the machine to execute a specified instruction out of the normal sequence.

CONDITIONAL JUMP—An instruction which causes the machine to execute a specified instruction out of the normal sequence, only if certain conditions are met.

COMPARISON INSTRUCTION—An instruction which compares two numbers to see which is larger or smaller and then performs a jump on the basis of the results.

CONDITIONAL HALT INSTRUCTION—An instruction which causes the machine to come to a halt only if a halt switch on the control panel is thrown.

SAMPLING INSTRUCTION—An instruction which looks at the state of the input register. If it is not full, the next instruction in sequence is executed. If it is full, the contents are transferred to the memory and an instruction out of sequence is executed next.

BREAKPOINT—A point in a routine where the computer may be stopped by the operator for a check of the routine.

B REGISTER—A counter used to modify address portions of instructions before the instructions are executed.

CHAPTER 16

A specimen computer

We now know of what building blocks computers are built. We know now how building blocks may be combined to perform an array of computer functions. We know how to combine functional building blocks into the five major functional units that any digital computer must possess. Now let us see how the functional units work together by describing an integrated specimen computer.

The specimen computer presented is not a complete machine; it is a skeleton. However, the skeleton is good enough so that if enough flesh is added to the bones the computer can be made operable.

GENERAL DESCRIPTION

Computer Characteristics

The specimen computer is a binary fractional fixed-point machine possessing 13 commands and a memory with a capacity of 64 words. The one-address instruction word is 12 bits long: 4 bits to encode the 13 commands; 6 bits to encode the 64 memory addresses; 1 sign bit to specify whether the instruction should be modified by *BR*; and 1 parity bit for checking. To simplify the logic of the machine, the data word is also 12 bits long: 10 magnitude bits, 1 sign bit, and 1 parity bit.

A coincident-current magnetic-core type of storage is utilized for the memory. This type of storage requires parallel read-in and read-out. The arithmetic unit, on the other hand, operates serially. As is shown later, a buffer register is utilized to make the memory and arithmetic units compatible.

The external storage medium is punched paper tape encoded in octal. A reader reads sequentially groups of three bits representing octal digits. Similarly, on the output side, the information is transmitted to the punch in groups of 3 bits.

TABLE 22

Characteristics of Specimen Computer

Language	
Number system	Binary
Number range	Magnitude less than one
Number designation	Fixed point
Bit representation	ONE = 0-v level; ZERO = −6 v level
Type of transmission	Serial; writing and reading are in parallel
Word size	12 bits; 10 information bits, 1 sign bit, 1 parity bit; parity bit not present in arithmetic and input-output units
Instruction code	One-address
Composition of instruction word	Address: Parity, Column, Row, Command, Sign — X XXX XXX XXXX X
Composition of data word	Parity, Magnitude, Sign — X XXXXXXXXXX X
Machine timing	Synchronous
	Word-time = 14 bit-times
Instruction repertoire	0001 *Store; 0010 *Clear add; 0011 *Add; 0100 *Subtract; 0101 *Multiply; 0110 *Shift right; 0111 Input; 1000 Punch; 1001 Jump; 1010 Set *BR*; 1011 Test *BR*; 1100 Jump on Minus; 1101 Halt
	* *BR* modifiable instructions if sign of instruction word is negative
Checking scheme	Parity-bit detector
Memory	
Type	Coincident-current magnetic core
Capacity	64 words
Construction	12 8 × 8 core planes
Input-output	
Type	Punched paper tape reader and punch
Number system	Octal
Arithmetic	
Type of algebraic addition	Negative numbers converted to 2's complement and added
Type of multiplication	Sequence of alternate additions and shifts
Speed	Addition: 1 word-time Multiplication: 22 word-times *BR* modification: 1 extra word-time
Building blocks	
Basic logical elements	Diode AND's and OR's; transistor amplifiers for NOT
Amplification	Transistor amplifiers
Storage	Direct-coupled transitor flip-flops Square-loop magnetic cores (bulk storage)

In the paper tape the bits are stored as holes for ONE's and no-holes for ZERO's. In the inside of the computer, ONE's are stored as 0-v levels, and ZERO's as −6-v levels. Machine timing signals are pulses of 6-v amplitude.

The ONE's and ZERO's are operated upon by means of diode AND and OR circuits. Since after four levels of logic the information signal is deteriorated, a transistor amplifier is used to restore its energy. Two types of amplifiers are provided, one to reverse the polarity in order to produce the NOT function, and one merely to amplify and not change the polarity, For delay and control functions, direct-coupled transistor flip-flops are used.

These basic building blocks are combined into gates, registers, counters, selectors, generators, detectors, and adders which are needed to breathe life into the machine.

The essential characteristics of the computer are given in Table 22.

Computer Organization

A system block diagram showing the major functional blocks within each of the functional units of the specimen computer is drawn in Fig. 156.

The memory consists of 12 magnetic-core coincident-current planes driven by column and row address selectors. Information arrives at the memory unit serially and leaves it serially. Access to the coincident-current planes, however, is in parallel. The *MR* acts as a buffer to convert parallel read-out information to serial form, and serially arriving information into parallel form for writing into the cores. A parity-bit detector checks the information as it is removed from memory.

A parity-bit generator generates an extra bit according to the rules of parity just before a word is written into memory.

Machine words are in binary. However, instruction and data words are written in octal form onto punched paper tape. These are then fed to memory via the input-output register. The three bits of each octal digit are applied directly to the appropriate storage cells of the input-output register. The read-in is under control of a word counter. Output from the input-output register to memory is in serial. Words are sent serially into the input-output register and then fed from the separate storage cells to the punch under control of a word counter.

The arithmetic unit executes the arithmetic operations. Words are fed from memory directly to the adder or to the adder via the arithmetic registers, *AR*, *DR*, and *XR*. The adder performs algebraic addition, algebraic subtraction, and the addition steps in the multiplication process. Results are stored in *AR* from which point they are returned to the memory. As negative words enter the arithmetic unit, they are converted to

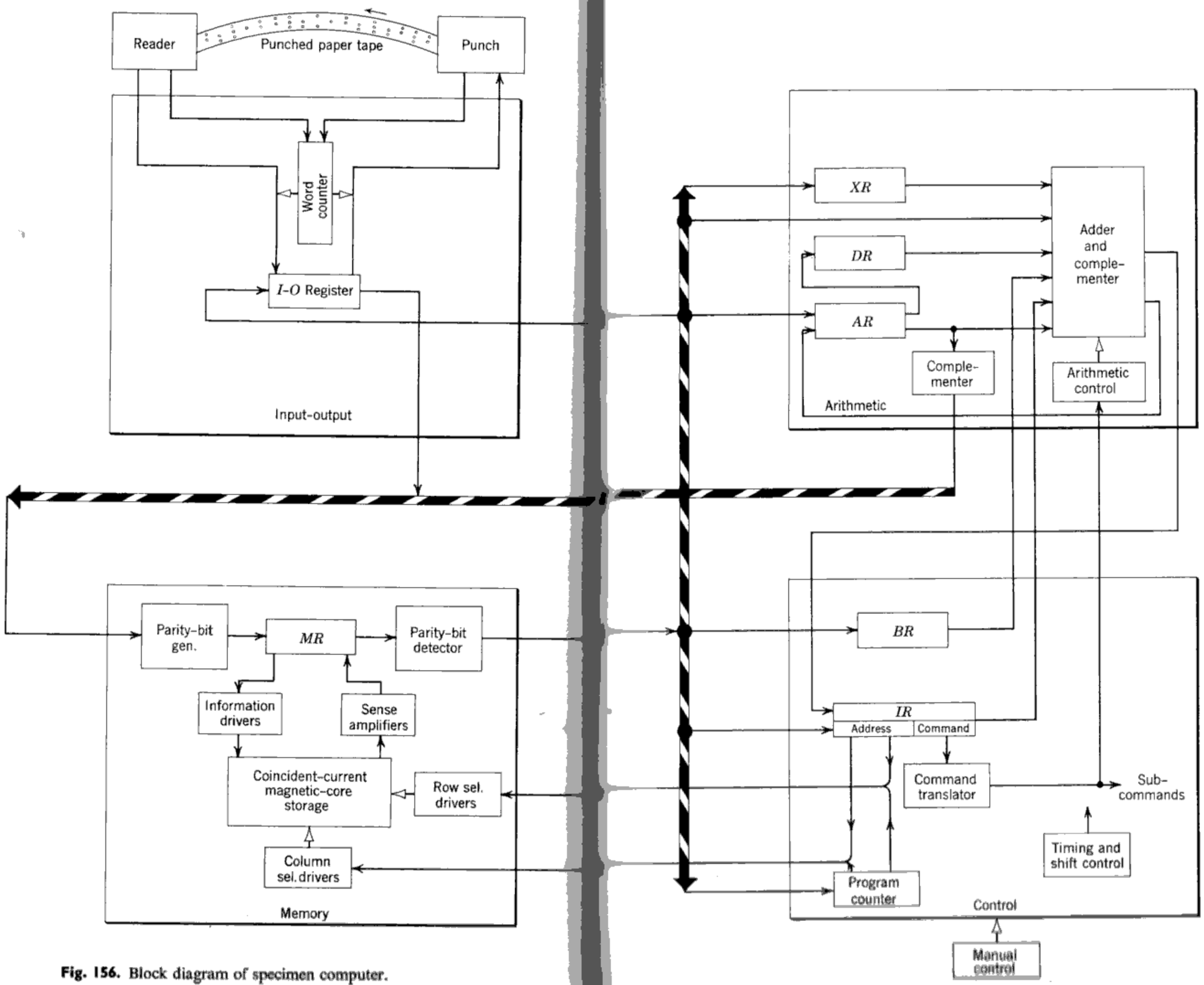


Fig. 156. Block diagram of specimen computer.

their 2's complement by the adder and complementer; positive words are untouched. Similarly, as words in 2's complement form are removed from *AR*, they are recomplemented by the complementer to bring them back to absolute value plus sign form.

The control unit determines the sequencing, translation, and execution of the instructions. From memory the instructions are transmitted to *IR*, where the instruction bits remain while being translated by the command translator and the column and row address selectors. The command translator produces subcommands which activate the appropriate gates to perform the necessary pieces of the operation. The timing and shift control determines in what sequence the subcommands are performed. At the end of an operation the program counter is stepped; its contents are fed to the column and row selectors to choose the next instruction to be executed. *BR* is used to modify instructions: before execution of an instruction the contents of *BR* are added to the contents of *IR* in the adder; the new instruction is returned to *IR* and translated and executed. The automatic control of the machine may be modified by the manual controls on the panel of the machine.

MEMORY (Fig. 157)

The memory consists of 12 64-bit magnetic-core planes, one plane for each of the 12 bits to a word. Column and row selectors and drivers (D_{WR}) choose the appropriate address by a coincidence of column and row currents. Information write amplifiers (*I*) cause ONE's and ZERO's to be written into the chosen core in each plane. Sense amplifiers (*S*) amplify output signals leaving each plane during read-out. Reading and writing are accomplished in parallel. The *MR* acts as a buffer to convert parallel information into serial form for outward transmission, and serially arriving information into parallel form for writing into the core planes. A parity detector is in the serial read-out path. A parity-bit generator is in the serial read-in path.

Read-Write Cycle

The memory operates in a read-write cycle which takes one word-time, as shown in Fig. 158. Parallel reading occurs during the first bit-time, serial transmission to and from *MR* occurs during the next 12 bit-times, and parallel writing occurs during the last bit-time.

During a read-type instruction—clear add, add, subtract, multiply, punch—read and rewrite subcommands are produced by the control

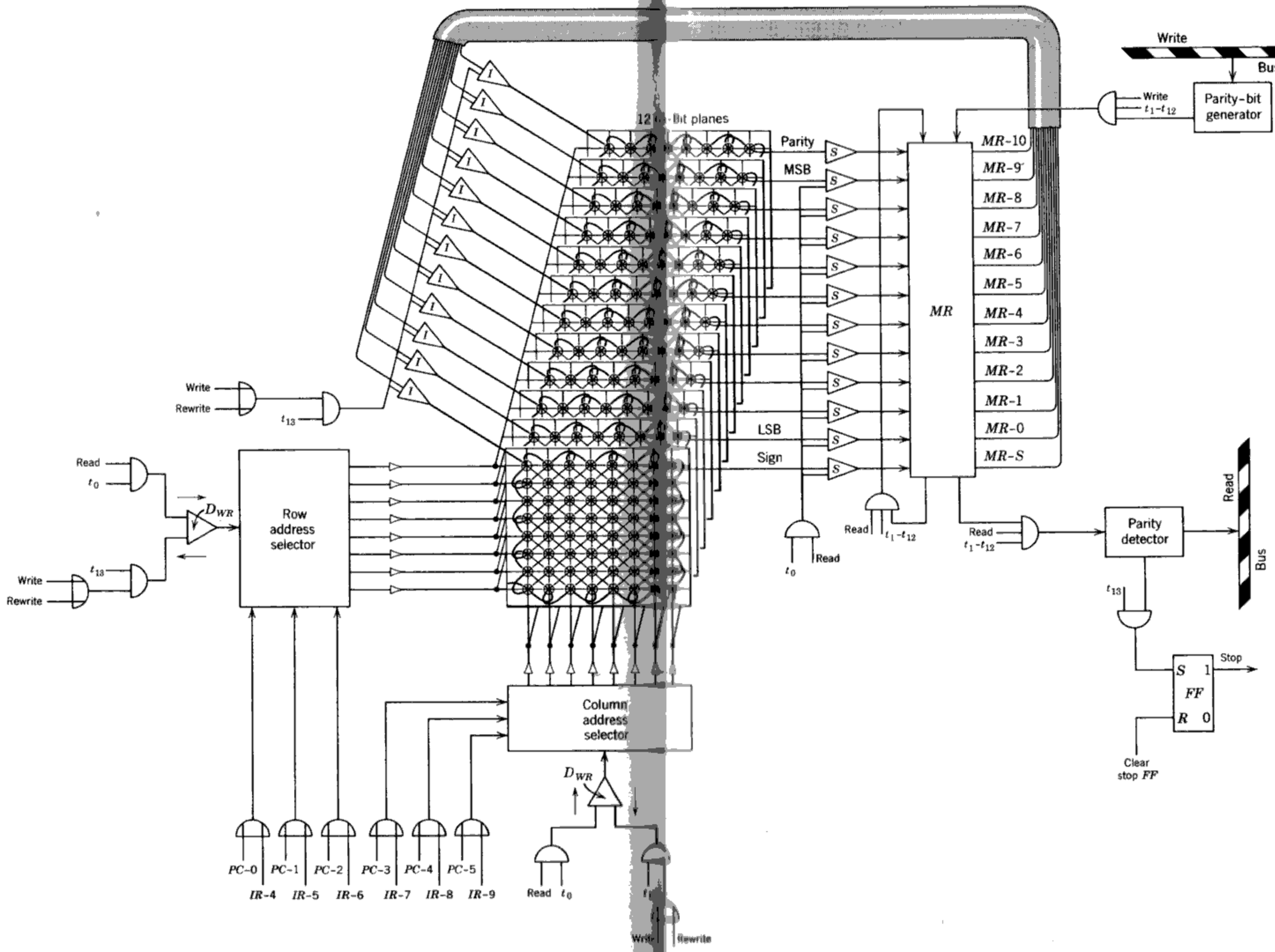


Fig. 157. Memory of specimen computer.

unit. The read subcommand operates at t_0 to transmit the 12 bits of the word in parallel to MR. At t_1 to t_{12}, the bits are transmitted serially through the parity detector to the arithmetic or output units. Because of the destructive read-out of magnetic cores, at t_{13} the bits of the word are removed from MR and rewritten into memory.

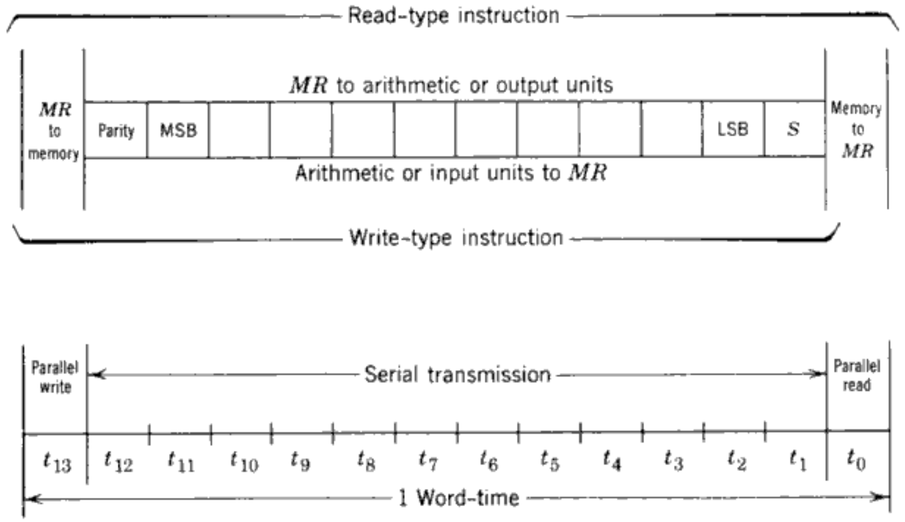


Fig. 158. Read-write cycle.

During a write-type instruction—store, input, set BR—no read subcommand occurs; nothing happens during t_0. During t_1 to t_{11}, the bits of the word are transmitted serially from the arithmetic or output units to MR through the parity generator. At time t_{12} a parity bit is generated. At time t_{13} the 12 bits of the word including the recently produced parity bit are written in parallel into memory.

Address Selection

The address is chosen by a 6-bit combination arriving from the program counter or from the address part of the word in the instruction register. In the first case, an instruction is read out of memory into the instruction register. In the second case an instruction word or data word is read from or written into memory.

The address is divided into a 3-bit column address and a 3-bit row address, each of which is fed to a diode selector. The selector decodes the bits into a signal on one of 3 output lines. The activated line opens the associated driver. During a write operation, current is made to flow in one direction; during a read operation, current is made to flow in the opposite direction through this driver.

Writing and Reading

One bit of each word is stored in one core in each plane. The lowest plane stores the sign, the next plane the least significant bit, followed by the other bits in order, and the topmost plane stores the parity bit.

Writing of a bit in the chosen core of each plane is accomplished by a coincidence of three currents: the column current, the row current, and the information current. One column driver feeds current through the chosen column of each plane, and one row driver feeds current through the chosen row in each plane. Separate information drivers feed each plane. In planes where we want to write a ONE, the information current is in a direction aiding the row and column currents. In planes where we want to write a ZERO, the information current is in a direction opposing the row and column currents. In the first case the chosen core flips; in the second case it does not.

Reading is performed by reversing the row and column drive currents. The outputs are introduced into the sense amplifiers, one for each plane. The sense amplifiers convert the signals into pulses of standard amplitude before they are transmitted to *MR*.

M Register

The *MR* is a shift register capable of accepting information in serial or in parallel, and transmitting it in serial or in parallel. During reading (t_0), the 12 bits from the 12 sense amplifiers are fed through 12 gates to each of the 12 storage cells of *MR*. Once in the register, timing pulses t_1 to t_{12} shift the bits from one storage cell to the next, and from the LSD through a parity bit-detector to the bus. During this same time a recirculation gate is opened to allow the bits to feed back to the other end of the register; this is done to maintain the bits in storage. At t_{13} the bits are applied through the information drivers to the same cores in memory from which the bits were removed.

If writing is taking place, bits are transmitted serially at times t_1 to t_{11} to the parity-bit generator. Out of the generator the eleven bits plus a parity bit are transmitted during t_1 to t_{12} to *MR*. This time the recirculation gate is inhibited, so that the new bits replace the old. Again, at t_{13}, the 12 bits are applied in parallel through the information drivers to the chosen cores.

Parity-Bit Detection and Generation

The 10 magnitude bits of the word may contain an odd or an even number of 1-bits. If the number of ONE's is odd, a ZERO is recorded for the parity bit; if the number of ONE's is even, a ONE is recorded for the parity bit. Thus the sum of all the ONE's (excluding the sign bit) should always be odd.

The parity-bit detector tests for this condition. In essence it is nothing more than a complementing flip-flop which reverses states with each ONE. After all the bits pass through, a timing pulse t_{12} samples the state of the flip-flop. If it reads ONE, everything is alright. If it reads ZERO, indicating an even count of ONE's, the stop flip-flop is set. This flip-flop, in turn, stops the computer.

INPUT-OUTPUT (Fig. 159)

The external storage medium for the specimen computer is punched paper tape. Instruction and data words are entered onto punched paper in octal form: the 10 bits plus sign bit are represented by 5 octal digits, as shown in Fig. 160. An input instruction causes the bits of each octal digit to be entered into the appropriate storage cells in the input-output (*I-O*) register under control of the word counter; when the register is full, the bits are shifted out serially into *MR*. During a punch instruction, the bits are shifted serially from *MR* to the *I-O* register; when the register is full, the bits of each octal digit are removed under control of the word counter and are punched on paper tape.

Input Instruction

An input instruction produces the input command signal which sets the read-in flip-flop and turns on the paper tape reader. At the same time that the reader reads the three bits of each octal digit, it also reads an input timing pulse. The input timing pulse is applied to the synchronizer which produces a sync pulse occurring at t_0. The sync pulse steps the word counter to 1. The 1-output of the word counter plus the "start read-in" signal from the read-in flip-flop allow the most significant bit to enter the most significant storage cell of the *I-O* register. As the three bits of the next octal digit are read, the associated input timing pulse becomes a sync pulse which steps the counter to number 2. The 2-output of the word counter plus the "start read-in" signal allow the next most significant three bits to enter into the three next most significant storage cells of the *I-O* register.

As each octal digit is read, the associated sync pulse steps the counter to cause the bits to be entered into progressively less significant storage cells until, at count 5, the sign is entered in the least significant position. Also at the count of 5, the read-out flip-flop is set. The set output resets the read-in flip-flop whose output in turn shuts off the reader.

The read-out flip-flop is turned on at time t_0. During t_1 to t_{11} the bits

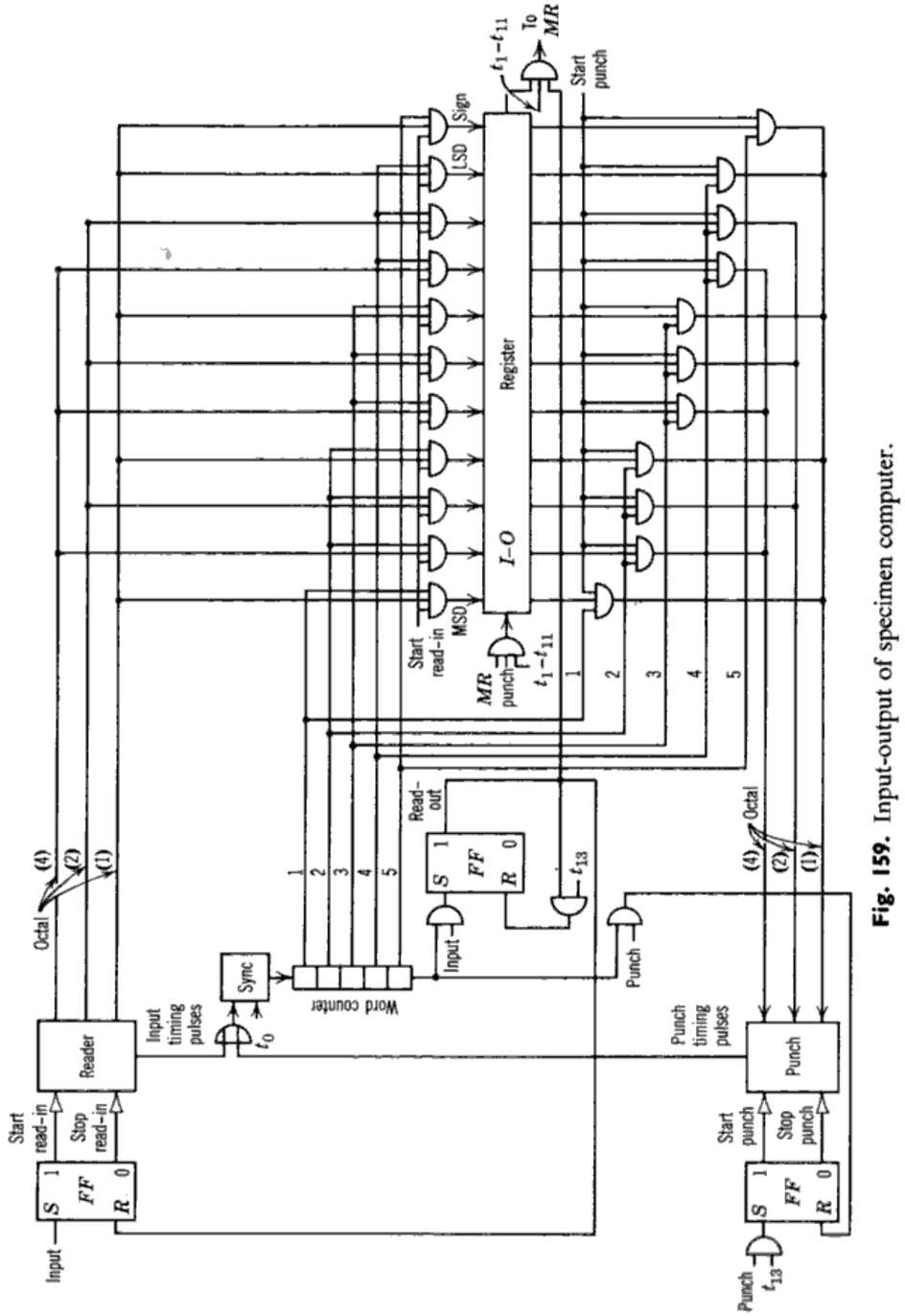


Fig. 159. Input-output of specimen computer.

in the *I-O* register are shifted through the output gate to *MR*. During t_{12} a parity bit is generated, and during t_{13} the whole word is written into the address in memory specified by the instruction word.

At t_{13}, too, the read-out flip-flop is reset. The *I-O* register is clear and ready for another input or punch instruction. Figure 160 shows the relationships among the number to be entered, its octal representation, and its representation on paper tape, and in the *I-O* register.

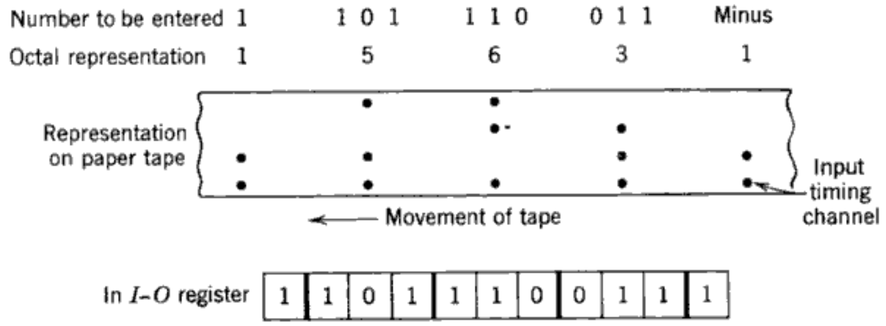


Fig. 160. Punched paper tape format.

Punch Instruction

During t_0 of a punch instruction, a read subcommand causes read-out of the word from memory into *MR*. During t_1 to t_{11}, the bits are shifted out of *MR* into the *I-O* register. At t_{12} the parity detector generates a pulse to stop the computer if the number does not meet the parity rule. The word transferred from memory remains in the *I-O* register. At t_{13} the "start punch" flip-flop is set to turn on the punch, and to halt computer sequencing until the punch finishes its job.

The punch produces punch timing pulses which are converted to sync pulses at time t_0. These sync pulses step the word counter in the same way they do during the input operation. In the 1-output position the most significant bit is applied to the punch. In the succeeding positions of the counter, lower significant bits are transferred in groups of three to the punch. During step 5, the sign bit is transferred and the start punch flip-flop is reset to stop the punch. The stop punch signal also allows the end-of-operation signal to be produced to enable instruction sequencing to continue.

ARITHMETIC (Fig. 161)

The arithmetic unit executes the add, subtract, shift-right and multiply instructions. Before these instructions may be executed, a clear-add

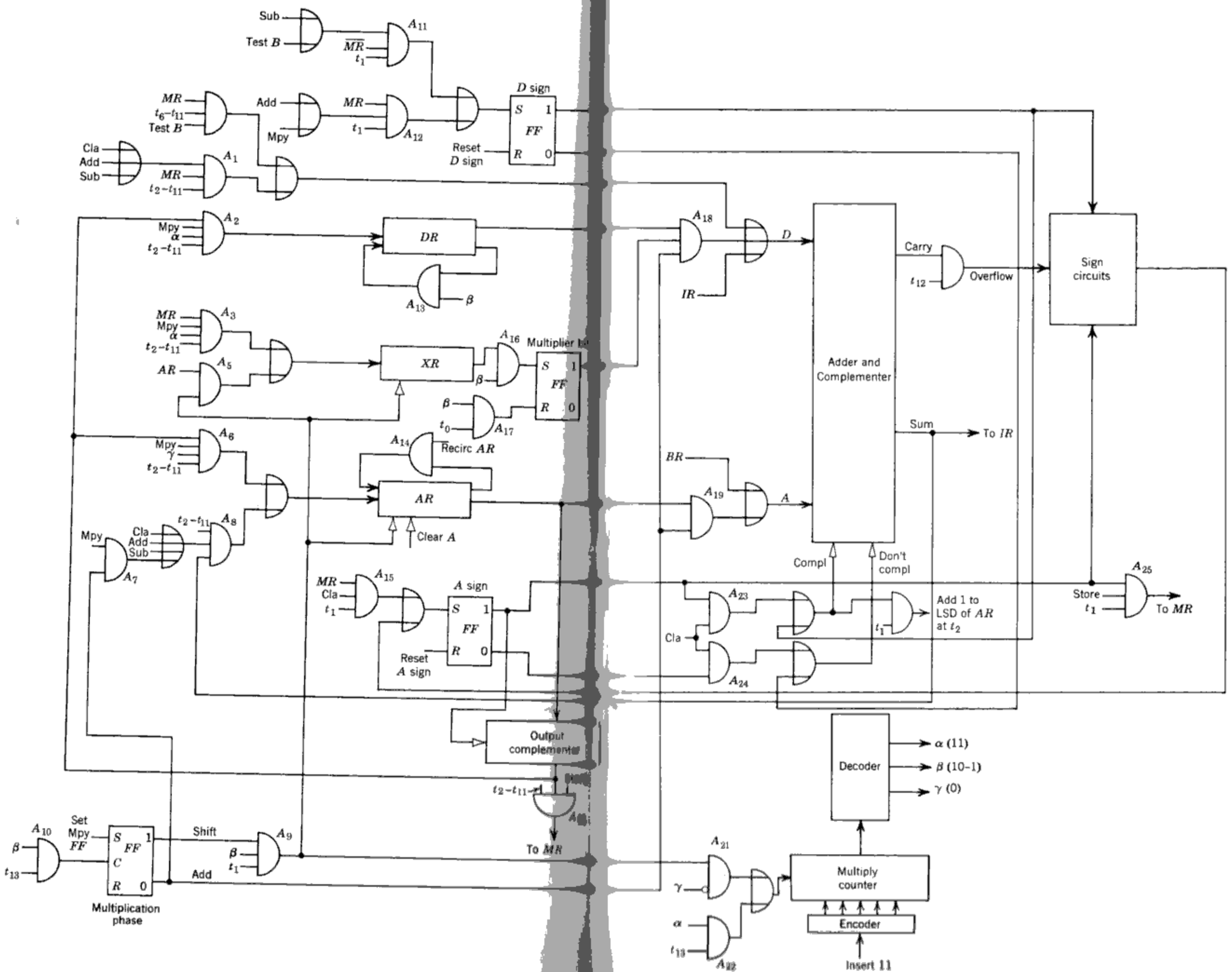


Fig. 161. Arithmetic unit of specimen computer.

instruction brings an operand into *AR*. The second operand is entered by the arithmetic instruction. After the arithmetic operation is performed, the result may be returned to memory by means of a store instruction. For all operations except multiply, negative operands are stored in arithmetic registers and are operated upon in 2's complement form. This means that complementation may occur during the input or output of a negative word.

General Description

The arithmetic unit consists of three arithmetic registers—*AR*, *DR*, and *XR*—an adder and complementer, an output complementer, sign circuits, multiply counter, and several flip-flops.

AR and *DR* are 10-bit shifting recirculating registers. *AR* is the accumulator, and *DR* stores the multiplicand. Shift pulses are under control of the timing and shift control. (Shift pulses are not shown at the registers.) Both *AR* and *DR* have separate flip-flops for storing the *A* and *D* signs.

XR is a 10-bit shifting register for storing the multiplier. Associated with *XR* is a multiplier-bit flip-flop which controls the addition of the multiplicand to the accumulated partial product during the repeated addition and right-shift process used in multiplication.

The adder and complementer produce the sum or difference of the two terms applied to them and return the result to *AR*. Whenever necessary, the adder complements and adds at the same time. This it does by the judicious use of gates, as shown below.

The output complementer converts results that are in 2's complement form back to absolute-value form while they are being transmitted to memory.

The sign circuits determine whether the sign of the result, as stored in the *A* sign flip-flop, should be left alone or reversed. This it does under control of the *A* and *D* sign flip-flops and the presence or absence of overflow.

The multiply counter together with its associated encoder and decoder, and the multiplication-phase flip-flop control the more extensive sequencing required to execute a multiply instruction.

The Adder

The adder is of the serial type. The augend and addend bits plus the carry bit from the previous position are combined in this circuit, and the sum bit is fed to the accumulator. The carry bit is delayed one bit-time and returned to the input of the adder in time to be combined with the next most significant augend and addend bits.

The augend is applied to the adder from *AR* which is storing negative words in complement form. The addend, on the other hand, is applied directly to the adder in absolute-value form. If the addend is positive, the adder behaves as any normal adder. If the addend is negative, the adder complements the addend and then adds. To convert the sum from 1's com plement to 2's complement, a 1 is added to the *LSD*.

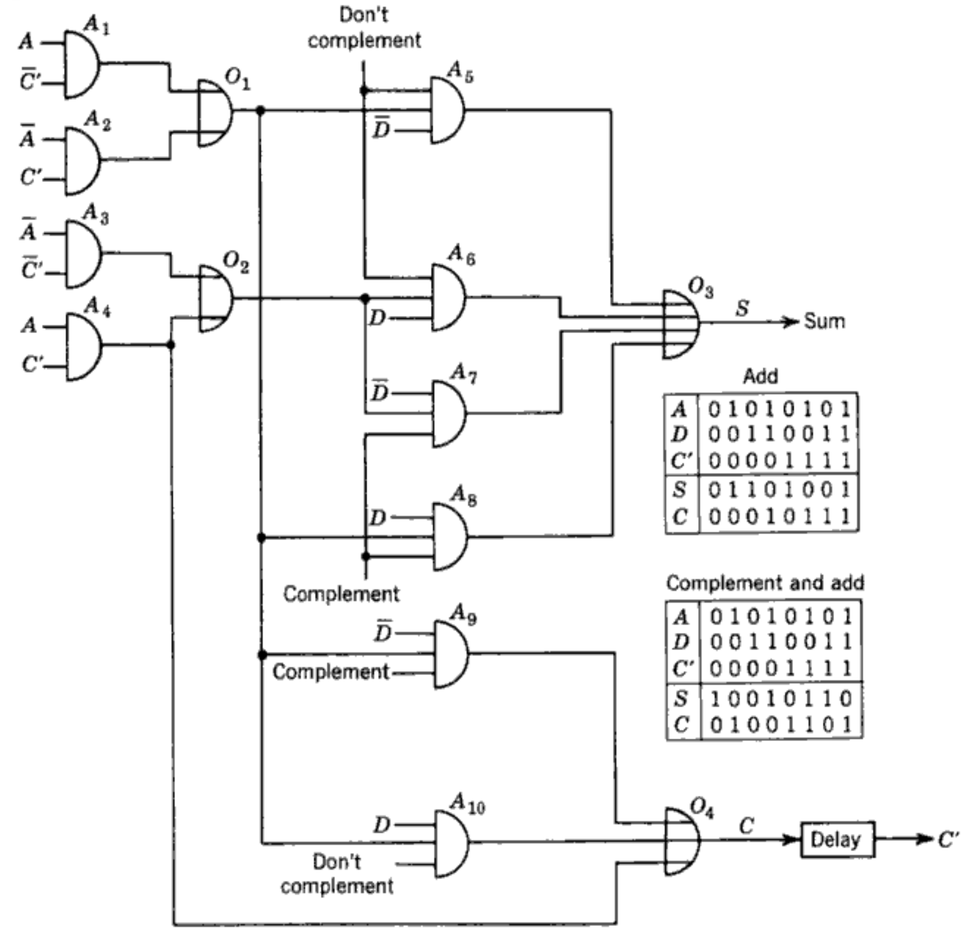


Fig. 162. Adder and complementer of specimen computer.

The adder circuit is shown in Fig. 162. Here *A* is the augend, *D* the addend, *C'* the carry from the previous position, *S* the sum bit, and *C* the output carry bit. The sum circuit and the carry circuit produce the normal adder outputs when the "don't complement" signal is applied; they produce outputs which take into account the complementation of the addend when the complement pulse is applied.

The add table in the illustration is a truth table showing the conditions

under which sum and carry bits are produced during simple addition. These conditions may be summarized by the following logical expressions:

$$S = A \cdot \bar{D} \cdot \bar{C}' + \bar{A} \cdot D \cdot \bar{C}' + \bar{A} \cdot \bar{D} \cdot C' + A \cdot D \cdot C'$$
$$C = A \cdot D \cdot \bar{C}' + A \cdot \bar{D} \cdot C' + \bar{A} \cdot D \cdot C' + A \cdot D \cdot C'$$

The complement and add table is a truth table showing the conditions under which sum and carry bits are produced if the addend bit is assumed to be complemented first. Note that the S and C bits are obtained by assuming a value of 1 for each D bit which is 0, and a value of 0 for each D bit which is 1. This truth table may be summarized by the following logical expressions:

$$S_C = \bar{A} \cdot \bar{D} \cdot \bar{C}' + A \cdot D \cdot \bar{C}' + A \cdot \bar{D} \cdot C' + \bar{A} \cdot D \cdot C'$$
$$C_C = A \cdot \bar{D} \cdot \bar{C}' + \bar{A} \cdot \bar{D} \cdot C' + A \cdot \bar{D} \cdot C' + A \cdot D \cdot C'$$

If D and $\bar{D}$ are factored out, the expression for the simple sum may be rewritten:

$$S = \bar{D} \cdot (A \cdot \bar{C}' + \bar{A} \cdot C') + D \cdot (\bar{A} \cdot \bar{C}' + A \cdot C')$$

If D and $\bar{D}$ are similarly factored out of the expression for the sum assuming complementation of the addend bit, the sum expression may be rewritten:

$$S_C = \bar{D} \cdot (\bar{A} \cdot \bar{C}' + A \cdot C') + D \cdot (A \cdot \bar{C}' + \bar{A} \cdot C')$$

By a similar factoring procedure, the carry expressions may be rewritten:
No complementation

$$C_C = D \cdot (\bar{A} \cdot C') + D \cdot (A \cdot \bar{C}' + \bar{A} \cdot C' + A \cdot C')$$
$$= A \cdot C' + D \cdot (A \cdot \bar{C}' + \bar{A} \cdot C')$$

Complementation

$$C_C = \bar{D} \cdot (A \cdot \bar{C}' + \bar{A} \cdot C' + A \cdot C') + D \cdot (A \cdot C')$$
$$= \bar{D} \cdot (A \cdot \bar{C}' + \bar{A} \cdot C') + A \cdot C'$$

By studying the above expressions, several important facts come into view:

1. $(A \cdot \bar{C} + \bar{A} \cdot C')$ is common to several terms.
2. $(\bar{A} \cdot \bar{C}' + A \cdot C')$ is common to several terms.
3. $A \cdot C'$ produces a carry regardless of whether D or $\bar{D}$ is present, or whether the addend is complemented or not.

Because of these conditions, duplications may be avoided by manufacturing $A \cdot \bar{C}'$, $\bar{A} \cdot C'$, $\bar{A} \cdot \bar{C}'$, and $A \cdot C'$ by means of gates A_1 to A_4, and

combining the first two in O_1 and the second two in O_2. Gates A_5 and A_6 produce the two terms of the sum when no complementation occurs. Gates A_7 and A_8 produce the two terms of the sum when complementation does occur. The output of O_1 is also fed to the carry circuit gates A_9 and A_{10}. Gate A_9 produces $\bar{D} \cdot (A \cdot \bar{C}' + \bar{A} \cdot C')$ when a complement pulse is present; gate A_{10} produces $D \cdot (A \cdot \bar{C}' + \bar{A} \cdot C')$ when a "don't complement" pulse is present. The $A \cdot C'$ is fed to the carry output line under all conditions.

Add, Clear-Add and Subtract Instructions

Before an add instruction is given, the augend is present in absolute-value form in AR, and the A sign flip-flop is storing its sign.

When the add command is translated, it produces read and rewrite subcommands; at the same time, the address part of the instruction chooses the appropriate location in memory to be read. The word is read-out of memory in parallel into MR during t_0. During t_1 the sign bit is transferred from MR to the D sign flip-flop via gate A_{12}. If the sign is minus, the D sign flip-flop is set; if the sign is plus, the D sign flip-flop remains reset (this is the state in which the reset D sign signal placed it during t_0). During t_2 to t_{11}, the 10 magnitude bits are transferred from MR to the adder via A_1. These are the bits labeled D in the adder circuit.

At the same time that the magnitude bits of the addend (D bits) are being applied to the adder, the 10 augend bits stored in the A register (A bits) are also applied to the adder; at t_1 a pulse is produced which causes a 1 to be added to the LSD of AR at t_2.

If the D sign flip-flop is set, the adder complements and adds. If the D sign flip-flop is reset, the adder produces simply a sum. As each sum bit is produced, it is fed back to AR. Thus, after time t_{11}, the complete sum is stored in AR.

The sign of the sum is determined by the sign circuits. The sign of the sum depends upon the following factors:

1. The sign of D.
2. The sign of A.
3. The presence or absence of a carry from the most significant position (overflow).

The signs of D and of A are compared immediately after the D sign flip-flop has received the addend sign bit, and the result of the comparison is stored in a flip-flop. If overflow occurs, at t_{12} it enters the sign circuits to modify its output. At t_{13}, the sign circuits set the A sign flip-flop under the conditions given in Table 23. Under all other conditions, the A sign flip-flop remains reset because of the reset A sign signal applied to it during t_0.

TABLE 23
Determination of Sign of Sum

Input			Output
A Sign	*D* Sign	Overflow	*A* Sign
−	−	Yes	−
+	−	Yes	+
−	+	Yes	+
+	+	Yes	−
−	−	No	+
+	−	No	−
−	+	No	−
+	+	No	+

A clear-add instruction is executed in a manner similar to the add instruction except that a clear-*A* subcommand clears *AR* to 0 during t_0, and the sign is transferred during t_1 to the *A* sign flip-flop instead of to the *D* sign flip-flop. During t_2 to t_{11}, the magnitude bits applied to the adder through A_1 are added to zero coming from *AR*. The sum which is the same as the input word is sent to *AR*. Thus a simple transfer occurs from memory to *AR*, and it occurs this way if the *A* sign flip-flop is reset. If the *A* sign flip-flop is set, the complement of the word is sent to *AR*.

During algebraic subtraction, the adder operates the same way as during algebraic addition. The only difference is that the *D* sign flip-flop is set when the sign is *plus*, and *reset* when the sign is *minus*. Thus complementation during algebraic subtraction occurs at opposite times to its occurrence during algebraic addition.

Right Shift and Multiply Instructions

AR, *DR*, and *XR* are of the shifting type; they shift only when shift pulses are applied to them from the timing and shift control. When ten shift pulses are applied, all ten bits in the register are shifted out. If at the same time the recirculating gates in *AR* and *DR* are activated, the ten bits return to their original positions in the register after shifting.

To obtain a one-place right shift of a word stored in *AR*, only one shift pulse is applied. To obtain a right shift of two pulses, two shift pulses are applied, and so on. A right-shift instruction has a number in the address part specifying the number of shifts. This number is decoded to produce a signal which is applied to the timing and shift control; this circuit, in turn, produces the appropriate number of shift pulses to be applied to *AR*.

The ability to recirculate the contents of *AR* and *DR* and to perform a right shift in *AR* and *XR* makes these registers adaptable to the addition and right-shift process of multiplication. Also important for the multiplication process are: the multiplier-bit flip-flop which determines whether the least significant bit of the multiplier is 1 or 0; the multiplication-phase flip-flop which causes alternate add and shift stages; and the multiply counter with its encoder and decoder for controlling the sequencing of the multiplication instruction.

A multiplication is translated into several subcommands, one of which is "insert 11." At t_0 the insert-11 signal is applied to the encoder which inserts the binary equivalent of 11 into the multiply counter. With the aid of the decoder, the counter controls the 22 steps of the multiplication process, as follows.

During the first word-time, the number 11 is decoded into an α signal which controls rearrangement and transfer of words to take part in the multiplication operation. During the following 20 word-times, the counter is stepped *down* once every alternate word-time, and the decoder produces a β signal to allow the multiplication-phase flip-flop to control the shift and add stages. During the 22nd word-time, the counter is stepped down to 0, which is decoded as γ. The γ signal rearranges the words so that the product appears in *AR*.

The actual steps in the multiplication process are as follows. At t_0, α is produced. At t_0, also, the multiplier stored in memory is read into *MR*. During t_1, the sign is transferred from *MR* to the *D* sign flip-flop. During t_2–t_{11}, the word is transferred to *XR* via A_3. Note that the word enters *XR* in absolute-value form. At the same time that the multiplier magnitude bits are entering *XR*, the previously stored multiplicand bits in *AR* are transferred to *DR* via the output complementer and A_2. The output complementer converts the word into absolute-value form. Thus at the end of the first word-time of the multiplication, *DR* is storing the multiplicand, and *XR* is storing the multiplier.

During α, the sign circuits compare the two signs. If they are unequal, they set the *A* sign flip-flop. If they are equal, the *A* sign flip-flop stays reset.

At t_{13} the multiply counter is stepped down and the decoder produces β. At t_1 the coincidence of β and the shift signal produced by the multiplication-phase flip-flop, which was set at the beginning of the multiply operation, produce a pulse at A_9 which causes *AR* and *XR* to shift one place to the right. The rightmost bit in *XR* enters the multiplier-bit flip-flop. If it is a ONE, it sets the flip-flop. If it is ZERO, the flip-flop remains reset because of the pulse applied to it at t_0. At t_{13}, A_{10} produces a pulse which complements the multiplication-phase flip-flop and brings it

to the add state. During the next word-time, A_{18} is open if the multiplier-bit flip-flop is set, and closed if the flip-flop is reset. If gate A_{18} is open, the multiplicand is shifted into the adder at the same time that the contents of AR (0 to start) are also shifted into the adder via A_{19}. The sum—the first partial product—is returned to AR via A_8. The multiplicand remains in DR since the recirculation gate is open. At the end of this word-time (at t_{13}), the multiplication-phase flip-flop is flipped to the shift state. As soon as this happens, the multiply counter steps down to 9.

Since the output of the decoder is still β, the shift and add stages are repeated as before. This shifting and adding under control of the least significant multiplier bit are performed for each of the ten multiplier bits. After 20 word-times the most significant 10 bits of the product appear in AR. (The least significant 10 bits are lost through shifting). The product is in absolute-value form.

To simplify mechanization as far as other instructions are concerned, during γ the absolute-value form of the product in AR is converted into complement form if the A sign flip-flop is set—storing a minus. This is easily accomplished by feeding the product from AR through the output complementer back to AR via A_6.

Store Instruction

The store is a write-type instruction. At t_1, the sign is transferred from the A sign flip-flop via A_{25} to MR. During t_2–t_{11}, the contents of the register are transferred via the output complementer, which converts the word to absolute-value form, through A_{20} to MR. During t_{12}, the parity generator generates the appropriate parity bit, and during t_{13} the entire word is written into the location in memory specified by the address.

CONTROL (Fig. 163)

The control unit accepts instruction words from memory, modifies the address if the instruction is BR modifiable, and produces the subcommands and other control signals needed to execute the instructions. After the instruction is executed, a program counter (PC) is stepped to choose the location in memory from which the next instruction is read. The sequencing may be modified by a jump instruction.

Instruction Sequencing

Instruction sequencing of non-BR modifiable instructions is under control of the instruction-phase flip-flop. Upon completion of an instruction, the flip-flop is reset to the FETCH state. When the instruction-phase flip-flop is in this state, the contents of the program counter are applied to the column and row selectors to read the next instruction into IR.

At the end of the word-time, at t_{13}, the instruction-phase flip-flop is flipped to the EXECUTE state. In this position, the address part of the instruction is applied to the column and row selectors to choose the operand, and the command decoder and subcommand encoder translate the command into subcommands. The subcommands plus the timing and shift pulses control the execution of the instruction.

The execution takes one word-time for all instructions except multiply. In the latter instruction an insert-11 subcommand is applied to the multiply counter which then takes over sequencing.

A subcommand produced for every instruction is the operation-complete pulse. This signal steps PC by one. At the end of the word-time, or for multiply at the end of the last word-time, the operation-complete pulse resets the instruction-phase flip-flop to the FETCH state, and the sequence repeats.

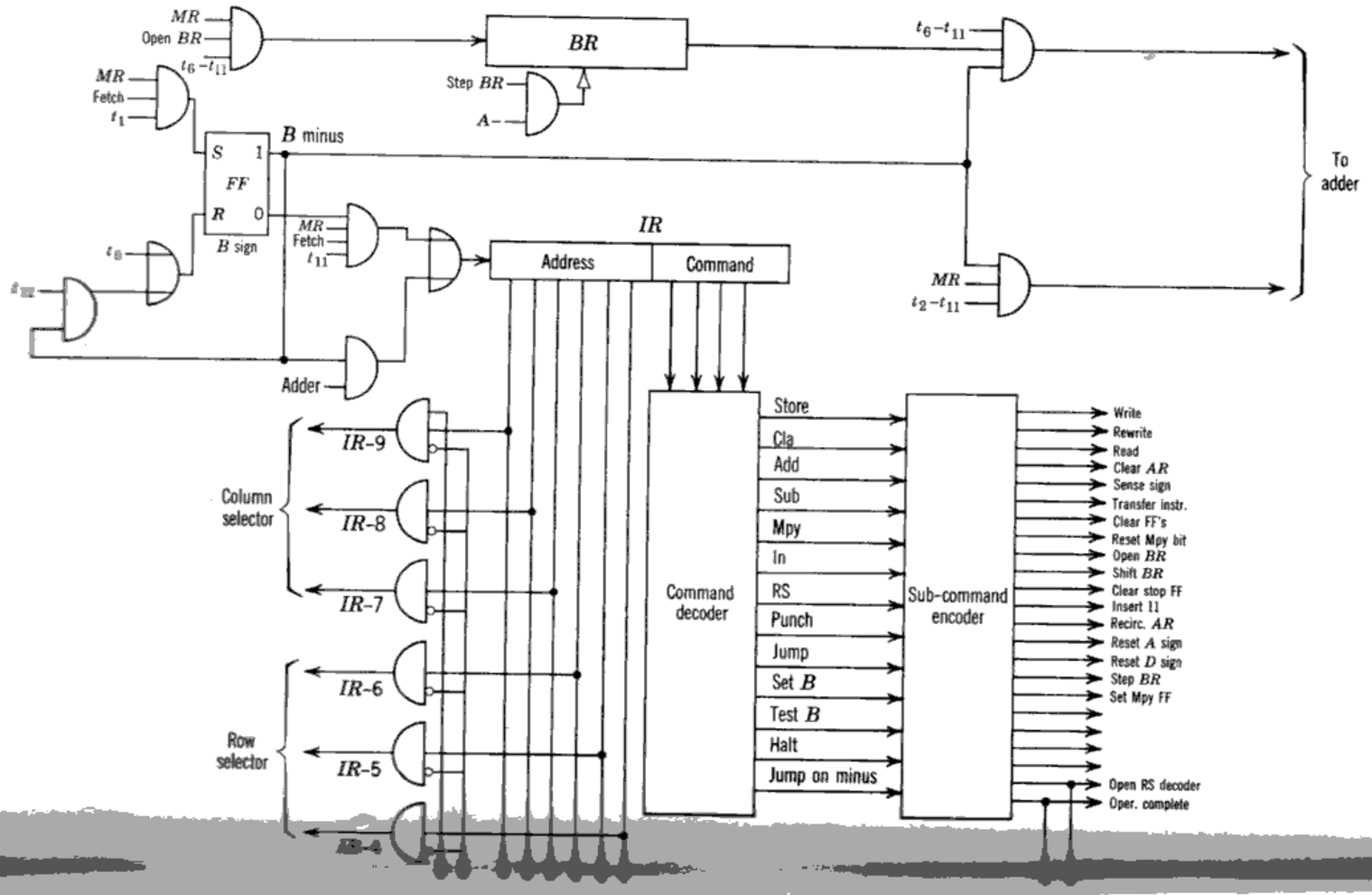
If the sign of the instruction is minus, the B sign flip-flop introduces an extra operation into the sequence. At time t_0, the B sign flip-flop is reset. At time t_1, the minus sign sets the flip-flop. As a result, the instruction bits entering from MR during t_2 to t_{11} are applied to the adder. During t_6 to t_{11}, the BR bits are also applied to the adder. The sum is immediately fed to IR.

While the addition is taking place, the instruction-phase flip-flop remains in the FETCH state since the B-minus signal inhibits the set input. As soon as the addition is completed at t_{12}, the B sign flip-flop is reset, and at t_{13} the instruction-phase flip-flop is set to EXECUTE. The remainder of the sequence is as before.

Memory and Nonmemory Instructions

Nonmemory instructions are executed slightly differently from memory instructions. During the EXECUTE phase of a memory instruction, the address bits are applied to the column and row selectors while the shift control feeds pulses to the appropriate register or registers to accomplish the transfer of information.

During a nonmemory instruction, such as a right shift, no address is present in the six most significant positions of IR. The "open RS decoder" subcommand inhibits the inputs and the selectors, and sends the six most significant bits of IR to the right-shift decoder. The right-shift decoder decodes the bits representing the number of places to be shifted into a signal which causes the shift control to produce the number of shift pulses required to accomplish this shift.



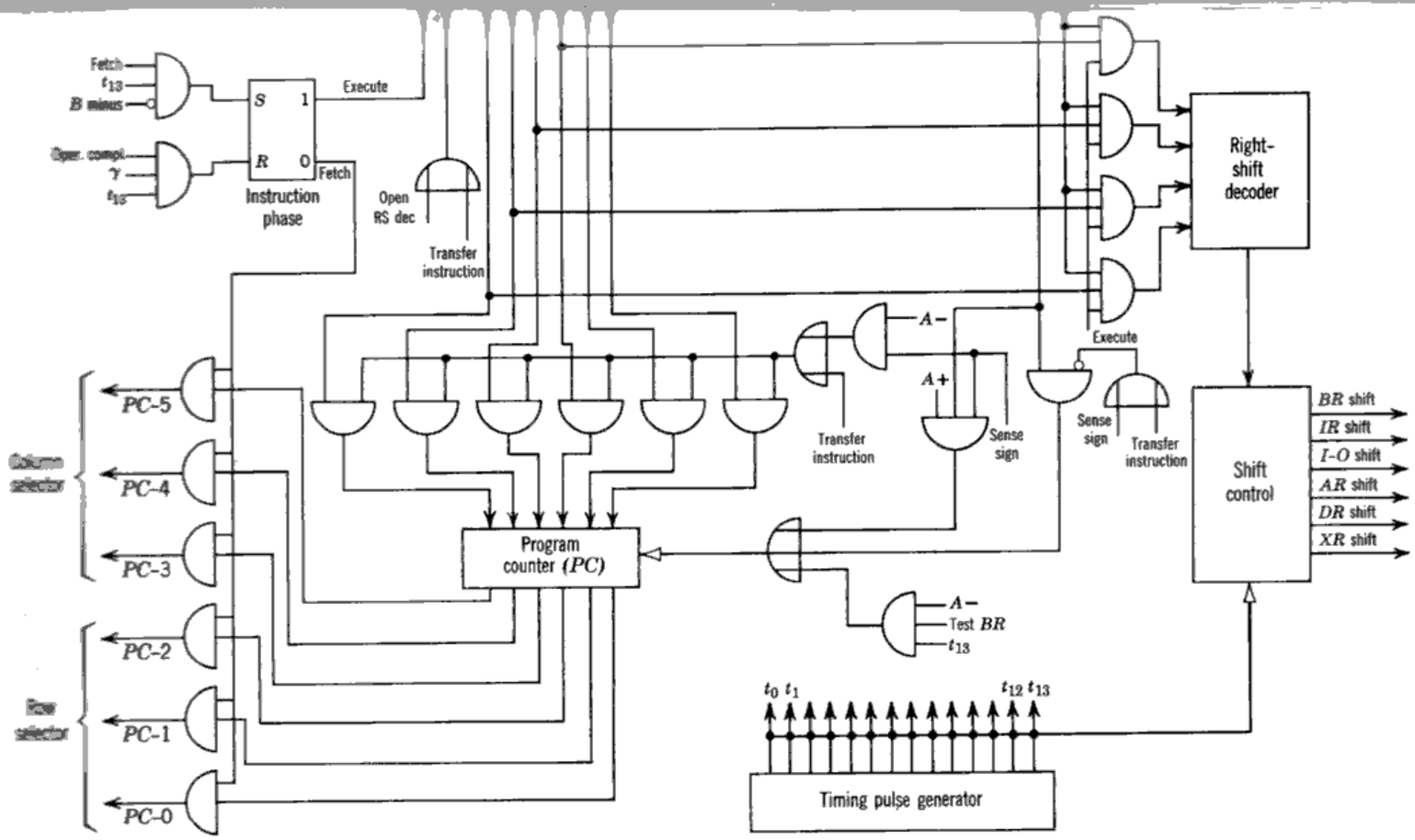


Fig. 163. Control unit of specimen computer.

Modifying the Sequence

The basic machine sequence may be modified by a jump or a jump-on-minus instruction. One of the subcommands produced during the translation of the jump command is "transfer instruction." This signal inhibits the selectors, and causes the address to be transmitted to the program counter. The operation-complete pulse is prevented from stepping the program counter so that during the next FETCH phase the next instruction to be executed is obtained from the address in the program counter.

The jump-on-minus instruction transfers the address to *PC* only if the sign of *AR* is minus. The sense sign subcommand and the *A* minus signal open the gates to *PC*. At the same time the sense sign subcommand inhibits the operation-complete pulse from stepping the *PC*. During the presence of *A*+ and sense sign signals, the operation-complete pulse steps the program counter normally.

Use of *BR*

To use *BR* in a program loop requires two instructions: Set *BR* and test *BR*. The first stores a number in *BR* before the start of a loop. The second tests the loop each time the series of operations is performed to see if the loop should be broken. If no jump should occur out of the loop, a count pulse is applied to *BR* and the loop is repeated.

The set *BR* command is translated into several subcommands, among which are read, rewrite, and open *BR*. The read subcommand causes the word given by the address part of the instruction to be read out into *MR* during t_0. During t_1 to t_{11}, the bits of the word are transmitted out of *MR*. However, the open *BR* and t_6–t_{11} signals allow only the six most significant bits to be shifted into *BR*.

The test *BR* instruction acts as a combination comparison and count *BR* instruction. The comparison is performed by means of the following subtraction:

Contents of *BR* — contents of test location

where the test location is given by the address part of the instruction. The difference is stored in *AR*. The relative value of the two terms is determined by the *A* sign flip-flop. Since both terms are positive, a negative sign for the difference indicates the *BR* count has not reached the test number. Under these conditions *BR* is stepped once (by the step *BR*, subcommand) the program counter is stepped twice. The program counter is stepped once normally by the operation-complete pulse, the second time at t_{13}. If the *A* sign flip-flop is positive, the program counter is stepped only once.

USING THE COMPUTER

Now for a few words on how the specimen computer may be used. Before the computer may solve a problem, the problem must be programmed. The program must then be loaded into the machine and debugged before actual operation. The loading, debugging, and operation, as well as maintenance, are under control of switches on the console.

The Console (Fig. 164)

The console stores controls and indicators for the performance of the following five functions:

1. Power control.
2. Operation control.
3. Program test
4. Manual input.
5. Register display.

The logic associated with each of these groups is discussed briefly below. (This logic is not included in the logical block diagrams in order to keep these diagrams simple.)

Power control. In this group are included the a-c power and d-c power switches and the fuses. The switches are merely for turning on the power and activating the machine's power supplies, and the fuses are for the protection of the several power supplies required for a digital computer.

Operation control. In this group are included the start, stop, and mode switches; and the operate, paper-punch-on, and tape-reader-on indicators.

The start and stop switches control machine sequencing. The start push button clears the program counter to 0, and then allows the operation-complete pulse to step the program counter to maintain instruction sequencing. As long as the operation-complete pulse is allowed to step *PC*, the operate lamp is lit. When the stop push button is depressed, the operation-complete pulse is inhibited from stepping *PC* and sequencing stops.

The mode switch has three positions: Automatic, test, and load. In the automatic position, normal sequencing occurs. In the test position the program test switches are made operative. In the load position, the number 40 is introduced into the program counter, and then automatic sequencing begins. If the location 40 in memory is storing the first instruction of the loading routine, a program stored on paper tape may be entered into memory.

The paper-punch-on and tape-reader-on indicators are in series with the on-off switches of the respective devices. They therefore light when these equipments are operating.

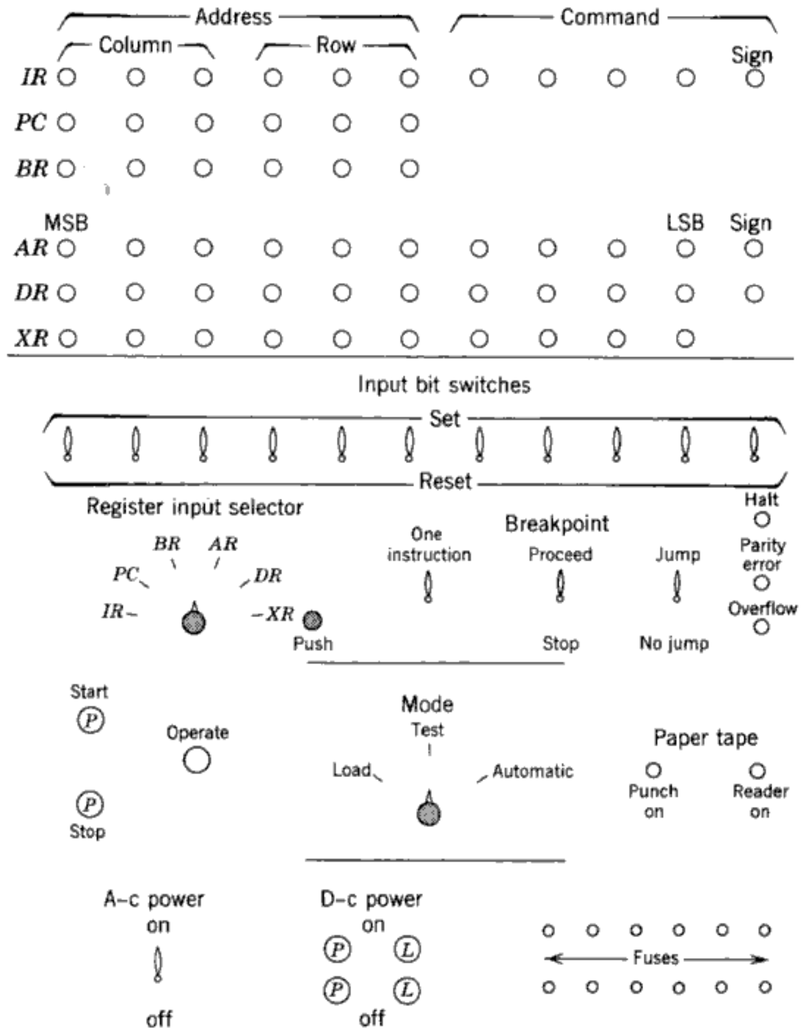


Fig. 164. Control panel of specimen computer.

Program test. In this group are included the one-instruction, breakpoint, and jump–no-jump switches; and the halt, parity error, and overflow indicators.

The one-instruction switch executes one instruction of the program at one time. Activation of the switch produces a simulated operation-complete pulse which steps the program counter. The instruction stored in this new location given by the program counter is then executed. The normally produced operation-complete pulse is inhibited, and sequencing stops until the one-instruction switch is depressed again.

The breakpoint and jump–no-jump switches are used together. The breakpoint switch causes the computer to stop at key points in the program being tested. The jump–no-jump switch may then be used to continue normal machine sequencing or to modify sequencing. In the stop position the breakpoint switch causes the computer to stop at the occurrence of a conditional jump instruction: the combination of the breakpoint stop signal plus the jump signal inhibits the operation-complete pulse from stepping the program counter, and inhibits gates between the address part of the instruction register and the program counter. When the breakpoint switch is placed in the proceed position, one of the above paths is opened: the gates between *IR* and *PC* if the jump–no-jump switch is in the jump position; the gate allowing the operation-complete pulse to step the counter if the jump–no-jump switch is in the no-jump position.

The halt, parity error, and overflow indicators light whenever the indicated conditions occur. The overflow lights only when the overflow represents an exceed-capacity.

Manual input. Manual input is controlled by the input-bit switches and the register input selector. Bits are entered by placing the input bit switches in the ONE or ZERO positions. The setting of the register input selector determines to which register this number is transferred. The actual shifting of the bits into the chosen register starts when the push button is depressed.

Register display. The register indicator lamps are connected to the outputs of each of the flip-flops composing the registers. An indicator is lit when a ONE is stored; unlit when a ZERO is stored.

Programming

To show more definitely how the computer operates, a simple program will be prepared, loaded into the specimen computer, debugged and performed. For ease of reference when discussing the program, the command code of the specimen computer is summarized in Table 24.

The program to be developed is one for finding the smallest number in a group of 60 numbers. Basically the program is as follows. The 60 numbers are stored on the punched paper tape, and the routine asks the tape reader to read-in one word at a time and store it in a location in memory called the "latest number location." This latest number is compared with the smallest number obtained from the previous comparison. If the latest number is the smaller of the two, the latest number is transferred to the smallest number location. If the previous smallest

TABLE 24

Specimen Computer Command Code

Command Code			
Octal	Binary	Name	Description
01	0001	Store	*STORE *AR* contents in location given by address.
02	0010	Transfer	*TRANSFER TO *AR*, contents of location given by address.
03	0011	Add	*ADD algebraically contents of location given by address to *AR* contents; leave sum in *AR*.
04	0100	Subtract	*SUBTRACT algebraically contents of location given by address from *AR* contents; leave difference in *AR*.
05	0101	Multiply	*MULTIPLY *AR* contents by contents of location given by address; leave product in *AR*.
06	0110	Shift right	SHIFT RIGHT *AR* contents a number of places given in address part of instruction.
07	0111	Input	INPUT a word from punched paper tape to location given by address.
10	1000	Punch	PUNCH onto paper tape word stored in location given by address.
11	1001	Jump	JUMP to instruction stored in location given by address.
12	1010	Set *BR*	SET *BR* by transferring to it address bits of word stored in location given by address.
13	1011	Test *BR*	TEST *BR* by subtracting contents of location given by address from *BR* contents. If difference is plus, sequence normally; if minus, step *BR* and jump to instruction stored in location following next location.
14	1100	Jump on minus	TEST sign of *AR* and JUMP ON MINUS to instruction at location given by address; if it is plus, sequence normally.
15	1101	Halt	HALT computations.

* These commands are modified by *BR* when the sign of the instruction word is minus.

number is still the smallest, the next word is read-in from the tape for another comparison. This loop is repeated until 60 numbers have been compared. *BR* keeps count of the number of comparisons. After the 59th comparison, signifying that 60 words have been compared, the smallest number is punched onto paper tape.

TABLE 25

Number Comparison

Location of Instruction	Instruction Code		Explanation
	Command	Address	
01	12	16	SET *BR* to 1.
02	07	21	INPUT first word from paper tape to smallest number location.
03	07	20	INPUT (second) word from paper tape to *latest number* location.
04	02	21	TRANSFER smallest number to *AR*.
05	04	20	SUBTRACT latest number from smallest number.
06	14	11	JUMP ON MINUS to instruction stored in 11. If *AR* is plus, sequence normally.
07	02	20	TRANSFER latest number to *AR*.
10	01	21	STORE latest number as smallest number.
11	13	17	TEST *BR* to see if 59 comparisons have been made. If so, sequence normally. If not, add one to *BR* contents and jump to instruction in 13.
12	11	14	JUMP to 14.
13	11	03	JUMP to 03 to repeat loop.
14	10	21	PUNCH smallest number.
15	15	00	HALT.
16	00	01	Constant.
17	00	73	Octal for 59 (for testing count).
20			Latest number location.
21			Smallest number location.

The program is shown in Table 25. In addition to storage locations 20 and 21 which store the latest number and smallest number respectively, two other locations, 16 and 17, are reserved for the storage of constants. The steps of the routine refer to these locations (shown at end of Table 25).

The routine is as follows. Before the program loop is begun, *BR* is set to 1, and the first word is read from the tape and stored in the smallest

number location. The first instruction word of the loop is stored in location 03. The next number (labeled second in the table) is read in from the tape to the latest number location and the latest number is subtracted from the smallest (steps 4 and 5). The next step is a jump-on-minus instruction. If the result of subtraction is a plus, the latest number is transferred to the smallest number location (steps 7 and 10). If the result of the subtraction is minus, a jump occurs to 11. At this location the *BR* contents are tested against the test count of 0073, which is octal for 59. Since fewer than 59 comparisons have been made, *BR* is stepped to 2, and the routine jumps to the instruction in 13 which, in turn, calls for a jump to location 03 to repeat the loop. During the second time through the loop, the third word enters from the tape to the latest number location, and it then takes part in the comparison. After each time through the loop, the test is made and the loop is repeated until the 59th comparison is accomplished. At this point the computer sequences to the instruction in 12 which, in turn, calls for a jump to 14 where the smallest number is punched out after which the computer is halted.

Loading

The words of the previous program should be stored in locations 01 to 21. How do we go about entering these words in their proper locations? In other words, how do we load the program?

Loading is accomplished by means of a punched paper tape reader and a loading routine stored in memory; this routine is called into operation when the mode switch is placed in the load position. The location in which the first instruction of the program is to be stored is entered manually by means of the input-bit switches and the register input selector.

The sequence of operations required to load a program is as follows. First the program is punched onto paper tape; in addition to the actual program, the number of words in the program—17 or octal 21 in the previous illustration—is punched as the *first* word. This tape is mounted on the reader which is then placed into operation. By means of the input-bit switches, the location of the first word of the program—01 in our example—is entered. The register input selector is placed in the *AR* position and the push button is depressed. This causes the number stored on the panel to be transferred to *AR*. After this is accomplished, the mode switch is placed in the load position, causing the computer to start sequencing with the instruction in location 40, the first instruction of the loading routine.

The loading program stores an input-instruction constant which is continuously modified to cause entry of succeeding words on tape to successive memory locations. At the start of the routine, the address of

TABLE 26

Loading Routine for Specimen Computer

Location of Instruction	Instruction Code		Explanation
	Command	Address	
40	01	55	STORE "location of first program word" (previously transferred from panel to *AR*).
41	07	56	INPUT first word from tape into "number of program words" location.
42	02	54	TRANSFER input-instruction constant to *AR*.
43	03	55	ADD "location of first program word" to input-instruction constant. *Result:* −INPUT 01.
44	01	57	STORE in input-instruction word location and
45	01	47	STORE in 47.
46	12	53	SET *BR* to 0.
47	−07	(01)	INPUT (first) instruction word of program into the (first) location of the program.
50	13	56	TEST *BR* to see whether the specified number of program words have been transferred to memory.
51	15	00	HALT computer if all program words have been entered.
52	11	47	JUMP to 47 to repeat loop if all program words have not yet been entered.
53	00	00000	Constant.
54	−07	00000	Input-instruction constant.
55			Address of first instruction word stored here.
56			Number of program words location.
57			Input-instruction word location.

the first word of the program is added to the input-instruction constant and, when it is executed, the first instruction word is transferred to the first address reserved in memory for the program. *BR* keeps count of the number of words entered, and also modifies the input-instruction constant each time the loop is repeated.

The entire loading program is shown in Table 26. In step 40, the address of the first program word is stored. In step 41, the first word on the tape,

specifying the number of program words, is stored. In steps 42 and 43, the input-instruction constant is added to the "location of the first word,"

−07 00000	Input instruction constant
00 00001	"Location of first program word"
−07 00001	Input instruction

and the resultant input instruction is stored for later use. In step 46, *BR* is set to 0 in preparation for the program loop occupying steps 47 to 50. Step 47 stores the manufactured input instruction. The first time it is executed, the first program word is transferred from the tape to the first location in the program. This is immediately followed by a test to see whether the specified number of program words has been entered. During the first time through the loop, the computer jumps to location 52, at which point a jump to the instruction in 47 causes the loop to repeat.

During the second time through the loop, *BR* stores a 1, thus sending the second program word to the following location in memory. Each successive program word is entered into a successive memory location until all words have been entered. At this point the test-*BR* instruction causes the computer to sequence to 51, and the machine halts.

Program Debugging

Programming is the art of describing a task so completely that a machine can perform it by following blindly the step-by-step procedures given. Because of this requirement for meticulous detail, the first results of a programmer's efforts are rarely correct. To test out or debug the program, the programmer has program test controls available to him. In the specimen computer, he has the one-instruction, breakpoint, and jump–no-jump switches. These program test switches are used together with the register display lights to find mistakes in the program.

The number-comparison program may be debugged with the aid of the one-instruction switch and the register indicators. After the program is loaded, and a group of 60 punched numbers is mounted on the paper tape reader, the one-instruction switch is depressed. The first instruction of the program is executed after which the contents of *PC*, *IR*, *AR*, and *BR* are viewed on the panel. The one-instruction switch is depressed again, the second instruction in the program is executed, and the contents of the same registers are viewed once more. In this way, the results of each instruction are apparent, and the programmer can tell if there is a mistake in the program. A group of numbers to be compared and the contents of the registers during such a step-by-step operation are shown in Table 27.

The above procedure may become tedious, especially in long, complicated

TABLE 27

Debugging Number-Comparison Routine

Numbers to be Compared		Contents of Registers During Each Step of Routine				
		PC	*IR*	*AR*	*BR*	
1st	1200	01	1216		1	
2nd	1300	02	0721		1	
3rd	0500	03	0720		1	
4th	0700	04	0221	1200	1	
5th	1400	05	0420	−0100	1	
.	.	06	1411	−0100	1	Breakpoint
.	.	11	1317	−0072	2	Breakpoint
.	.	13	1103	−0072	2	
.	0200	03	0720	−0072	2	
60th	0600	04	0221	1200	2	
		05	0420	0700	2	
		06	1411	0700	2	Breakpoint
		07	0220	0500	2	
		10	0121	0500	2	
		11	1317	−0071	3	Breakpoint
		13	1103	−0071	3	
		03	0720	−0071	3	
		04	0221	0500	3	
		05	0420	−0200	3	
		06	1411	−0200	3	Breakpoint
		11	1317	−0070	4	Breakpoint
		13	1103	−0070	4	
		03	0720	−0070	4	
		04	0221	0500	4	
		05	0420	−0900	4	
		.	.	.	.	Breakpoint
		.	.	.	.	
		.	.	.	.	
		.	.	.	.	Breakpoint
		03	0720	−0002	72	
		04	0221	0500	72	
		05	0420	+0300	72	
		06	1411	+0300	72	Breakpoint
		07	0220	+0200	72	
		10	0121	0200	72	
		11	1317	−0001	73	Breakpoint
		13	1103	−0001	73	
		03	0720	−0001	73	
		04	0221	0200	73	
		05	0420	−0400	73	
		06	1411	−0400	73	Breakpoint
		11	1317	0000	73	Breakpoint
		12	1114	0000	73	
		14	1021	0000	73	
		15	1500	0000	73	

programs. Faster debugging may be obtained by placing the breakpoint switch on stop. The computer sequences until it reaches a branch instruction, and then stops to allow the programmer to view the contents of the registers. After he is satisfied, he flips the breakpoint switch to the proceed position, and the computer sequences until the next breakpoint is reached. Table 27 shows the contents of the registers at each breakpoint in the above program.

When the computer stops at a breakpoint, a negative number in *AR* signifies that a jump will be executed next. If the programmer feels that the result of the previous operation is in error—the contents of *AR* should be +, for instance—he may test his belief by throwing the jump–no-jump switch to the no-jump position before throwing the breakpoint switch to proceed. If the machine now produces expected results, his hunch was correct.

Operation and Maintenance

Once a program is loaded and debugged, the mode switch is placed in the automatic position, and the computer performs the program. During operation, the register display lights flash on and off to indicate the changing contents of the registers. All is well, provided the following lights are on:

1. Operate.
2. Paper punch on.
3. Tape reader on.

The halt, parity, and overflow indicators are normally not on. Upon completion of a routine, the halt indicator goes on. If it goes on before the routine is completed, it indicates that a machine malfunction has occurred:

1. In the memory or arithmetic units if the parity indicator goes on at the same time.
2. In the arithmetic unit if the overflow indicator goes on at the same time.

To forestall the occurrence of machine breakdown during operation, the specimen computer may be tested beforehand by means of a test routine. A test routine simply makes the machine go through its paces. It consists of a group of instructions utilizing all the variations of machine commands and transferring words to and from every address in memory. At the end of the routine, the final result is compared with what the result should be if all the operations have been properly performed. If the two are alike, the machine is in operating order. If the two are unlike, something is wrong with the machine and it should be debugged before a program is run on it.

A more sophisticated form of trouble-finding routine is the diagnostic routine. This routine not only tests the machine to see if it is operating but also determines in what area the trouble is! The operation of this routine is based on the fact that every instruction utilizes only certain functional blocks, some functional blocks being shared by several types of instruction. The routine may therefore determine the cause of trouble by a process of elimination.

In outline, the diagnostic routine may operate as follows. First, since a comparison of numbers is required for practically every test, the adder is tested first. This is done by performing a series of additions on a set of standard numbers and then comparing the final result obtained with the correct sum. If the two are unlike, something is wrong with the adder. If the two are alike, the adder is tested further by means of a series of subtractions.

DR and *XR*, the multiplier-bit flip-flop, and the multiply counter are tested by performing several multiplication instructions on standard numbers and then comparing the results obtained with the correct results. If the results are unlike, the actual location of arithmetic malfunction may be pinned down further by performing right-shift instructions.

If the arithmetic unit has been tested and found to be in operating order, a series of store and transfer instructions transmitting a definite number between *AR* and every address in memory is performed. At the end of this series, the word finally found in *AR* is compared with the original word. If they are alike, the memory is in operating order; if not, the memory is not functioning properly.

If both the arithmetic and memory units are functioning properly, the control unit may be tested by performing jumps and *BR* modifiable instructions. Again, a comparison may be used to determine if the control is functioning properly.

Next the reader and input unit are tested by entering a word from the reader into memory, and then comparing this word with the same word stored in another location in memory. If the two are unlike, a malfunction is present in either the reader or the input unit. If the two are alike, the input system is in operating order.

Naturally, the routine is more devious than described. By a judicious choice of instructions and factors to be transferred and operated upon, it is possible to develop a diagnostic routine which would narrow the malfunction to a very small area in the machine. The routine may be written so that, when a malfunction is discovered, a code number specifying the functional area at fault is punched out on the paper tape.

Bibliography

The following is a selected bibliography of publications in the digital computer field. Only publications of about the same order of difficulty as this book are included.

Books

1. Booth, A., and Booth, K. H. V., *Automatic Digital Calculators*, Butterworth Scientific Publications, London, 1953.
2. Bowden, *Faster than Thought*, Pitman and Sons, Ltd., London, 1953.
3. Canning, R. G., *Electronic Data Processing for Business and Industry*, John Wiley and Sons, Inc., New York, 1956.
4. Eckert, W. J., and Jones, R., *Faster, Faster*, McGraw-Hill Book Co., Inc., New York, 1956.
5. Engineering Research Associates, *High Speed Computing Devices*, McGraw-Hill Book Co., Inc., New York, 1950.
6. Gotlieb, C. C., and Hume, J. N. P., *High Speed Data Processing*, McGraw-Hill Book Co., Inc., New York, 1958.
7. Irwin, W. C., *Digital Computer Principles*, D. Van Nostrand Co., Inc., New York, 1960.
8. Ivall, T. E., *Electronic Computers, Principles and Applications*, Philosophical Library, Inc., New York, 1960.
9. Kozmetsky, G., and Kircher, P., *Electronic Computers and Management Control*, McGraw-Hill Book Co., Inc., New York, 1956.
10. McCracken, D. D., *Digital Computer Programming*, John Wiley and Sons, Inc., New York, 1957.
11. Phister, M., *Logical Design of Digital Computers*, John Wiley & Sons, Inc., New York, 1958.
12. Richards, R. K., *Arithmetic Operations in Digital Computers*, D. Van Nostrand Co., Inc., New York, 1955.
13. Richards, R. K., *Digital Computer Components and Circuits*, D. Van Nostrand Co., Inc., New York, 1957.
14. Wilkes, M. V., *Automatic Digital Computers*, John Wiley and Sons, Inc., New York, 1956.
15. Wilkes, W. G., *Preparation of Programs for an Electronic Digital Computer*, Addison-Wesley Publishing Co., Inc., Cambridge, 1951.

Periodicals

1. *Computers and Automation*, Edmund C. Berkeley & Associates, New York.
2. *IBM Journal of Research and Development*, New York.

3. *Journal of the Association of Computing Machinery*, New York.
4. *Transactions of the Professional Group on Electronic Computers of the Institute of Radio Engineers*, New York.
5. *Datamation*, Ridgefield, Conn.

Bibliographies

1. *Bibliography on Office Automation*, Controllership Foundation, 1 E. 42nd St., New York.
2. *Electronic Data Processing in Industry*, American Management Association, 330 W. 42nd St., New York.
3. *Transactions of the Electronic Computer Group of the Institute of Radio Engineers.* March issue of each year gives list of papers printed during previous year.

Glossary

ABSOLUTE ADDRESS—The label assigned to a storage location by the machine designer.

ACCESS TIME—The time it takes to complete a transfer to or from a storage device.

ACCUMULATOR—A register which accumulates totals and stores the results of most arithmetic operations.

ADDRESS—An expression which designates the location of a character or word in a storage device.

AMPLIFYING ELEMENT—A device which increases or maintains the energy of signals representing bits or digits.

AND ELEMENT—A device which produces an output only when all input signals are present.

ASYNCHRONOUS COMPUTER—A computer whose subcommands are not executed according to times from a master clock; one operation begins after the previous operation is completed.

B-REGISTER—A counter used to modify address portions of instructions before the instructions are executed. Also called *B*-Box and index register.

BAND—A group of tracks which are often written onto or read from at one time.

BASE—Number of digits (or bits) to a number system.

BINARY—Involving the integer two.

BINARY ADDER—A device which produces the correct binary sum and carry bits with each combination of augend, addend, and previous carry bits.

BINARY CODED-DECIMAL NOTATION—A decimal number system in which each digit is represented by a group of bits (for instance, 8421, excess-3 systems).

BINARY COMPARATOR—See quarter adder.

BINARY COUNTER—A device or circuit which counts in the binary number system; produces numbers consisting of 1-0 combinations when pulses are applied to its input.

BINARY POINT—The point which marks the separation between integral powers of 2 and fractional powers of 2.

BIT—One of the two elements used in the binary number system. Also one of the two elements used in coded-decimal systems.

BIT-TIME—The time for the passage of a bit in a serial system, for the passage of a word in a parallel system, and for the passage of a digit in a serial-parallel system.

BLOCK—A group of words considered or transported as a unit, particularly with reference to input and output.

BRANCH INSTRUCTION—An instruction which may cause the machine to choose the next instruction out of the usual sequence. *Examples:* Unconditional and conditional jumps and comparison.

BREAKPOINT—A point in a routine where the computer may be stopped by the operator for a check of the routine.

BREAKPOINT SWITCH—A switch which causes the computer to stop at certain instructions in the program; for instance, at conditional jumps or at conditional halts.

BUFFER STORAGE—A storage device used to accommodate differences in the rate of flow or in method of transmission between two devices when transmitting information from one to the other.

CARRY—The digit to be added to the next higher column when the sum of the digits in one column is greater than the radix.

CHANNEL—A path along which information may flow.

CHARACTER—A digit, letter, or punctuation symbol, or their coded representations.

CHARACTERISTIC—That part of a floating-point representation which indicates the size of the exponent.

CLEAR—To restore a storage device to a prescribed state, usually ZERO.

CLOCK—A circuit or some physical arrangement which produces pulses at a steady rate.

CLOCK PULSE—One of a stream of pulses occurring at a constant frequency.

COINCIDENT-CURRENT STORAGE—A bulk storage device composed of storage cells in a rectangular array. Information is written into and read from a location by sending current coincidentally to the column and row lines in which the location occurs.

COMMAND—The portion of the instruction which tells the computer what to do (add, transfer, etc.).

COMMON LANGUAGE MACHINE—A machine whose output is understood by many other machines. Usually referred to machines using 5-hole punched paper tape.

COMPARATOR—A functional building block which determines whether two input pulse combinations are the same or different.

COMPARISON INSTRUCTION—An instruction which compares two numbers to see which is larger or smaller and then performs a jump on the basis of the result.

COMPLEMENT—10's—The number which must be added to the given number to obtain the modulus in the decimal system.
9's—One less than the 10's complement.
2's—The number which must be added to the given number to obtain the modulus in the binary system.
1's—One less than the 2's complement.

COMPLEMENTING FLIP-FLOP—A flip-flop which always reverses states upon application of an input signal.

CONDITIONAL HALT—An instruction which causes the machine to come to a halt only if a halt switch on the control panel is thrown.

CONDITIONAL JUMP—An instruction which causes the machine to execute a specified instruction out of the normal sequence, only if certain conditions are met.

CONTINUOUS RECORD—A record where a long string of information is stored on one unit. A good example is magnetic tape.

CONVERTER—A device for converting information stored in one form into information in another form. This device usually operates externally to the computer itself.

COORDINATE ACCESS STORAGE—A storage device whose word locations are arranged in space.

CYCLE TIME—The time between references to the storage device.

CYCLIC ACCESS STORAGE—A storage device whose word locations are arranged in time (as well as space) in such a manner that access to any one location may be obtained repeatedly according to a definite rate.

DATA WORD—A data word consists of numbers or characters which are to be processed. A number word may consist of digits and sign.

DECIMAL COUNTER—A counter which counts in the decimal number system—produces numbers consisting of combinations of digits 0 to 9—when pulses are applied to its input.

DECODER—A network of AND circuits which is used to convert a combination of signals into an output on one of several output lines.

DESTRUCTIVE READ-OUT—Storage in which the information is changed when it is read out.

DETECTOR—A functional building block which produces a pulse when certain combinations of pulses are applied to it.

DIAGNOSTIC ROUTINE—A routine placed in the computer which is used to determine the cause of a malfunction in the computer or a mistake in coding.

DIFFERENTIATOR—A circuit which produces narrow pulses coincident with and in the same direction as the leading and trailing edges of an applied square wave.

DIGIT—A symbol representing one of the elements in a positional number system.

DOUBLE-RANK REGISTER—A shift register consisting of two rows of storage cells: one row for actual storage of the bits, the other row for temporary storage of bits while they are being shifted to another stage.

DYNAMIC FLIP-FLOP—A circuit with two inputs and two outputs. The set input pulse produces a series of pulses on the 1-output side and no pulses on the ZERO output line. The reset input pulse produces the reverse.

DYNAMIC STORAGE CELL—A cell which has two possible outputs: a continuous series of pulses (ONE) or of no pulses (ZERO). A set pulse flips the cell into the ONE state; a reset pulse flips the cell into the ZERO state.

ENCODER—A network of OR circuits in which only one input is excited at a time and each input produces a combination of outputs.

ERASABLE STORAGE—Storage in which bits may be replaced by other bits.

EXCESS-3 CODE—A number code in which the decimal digit is represented by the binary equivalent of the number plus 3.

EXCLUSIVE OR CIRCUIT—A circuit which produces an output only when either one or the other of two inputs but *not both* are present.

EXTERNAL READER—A device which translates the information stored on external storage medium into electric pulses which may then be fed to the input register of the computer.

EXTERNAL STORAGE—Information stored on a medium that is not permanently connected to the computer. External storage is understood by the computer.

EXTERNAL WRITER—A device which translates the electric pulses from the output register of the computer into information stored on the external storage medium.

EXTERNALLY PROGRAMMED COMPUTER—A computer whose instructions are fed to it from the outside, as from a plugboard, pinboard, or punched card.

EXTRACT—The process of removing a portion of a word by combining another word with it according to the rules of logical multiplication (AND).

FIXED-POINT COMPUTER—A computer in which the decimal or binary point is assumed to be in a fixed location.

FIXED WORD—A word possessing a fixed number of digits.

FLIP-FLOP—A 1-bit storage device with two input terminals and 2 output terminals. The device remains in either state until caused to change to the other state by application of the corresponding signal. In either state the signal on one output terminal is out of phase with the signal on the other output terminal.

FLOATING-POINT COMPUTER—A computer which carries data words in exponential form. It can deal directly with fractional as well as integral numbers.

FORBIDDEN-COMBINATION CHECK—A check which tests for occurrence of a nonpermissible code combination.

FOUR-ADDRESS INSTRUCTION—An instruction with 4 addresses. These addresses specify the two operands, the result, and the address of the next instruction to be executed.

FRACTIONAL COMPUTER—A computer dealing with fractional numbers, that is, numbers smaller in magnitude than one.

GATE—Another name for a switching circuit.
Read-in—switching circuit which allows information to be written into a register.
Read-out—switching circuit which allows information to be read from a register.

GENERATOR CIRCUIT—A circuit which generates a certain combination of pulses under certain circumstances.

HALF-ADDER—A device which produces the binary sum and carry of two bits.

HYSTERESIS—The property of magnetic materials which prevents them from coming back to their original magnetic state after the magnetizing force is removed.

HYSTERESIS LOOP—A graphical representation of the variation in the state of a magnetic material as the magnetic force applied to it is varied in the plus and minus directions.

INHIBIT ELEMENT—A device which allows a signal to pass through only when another signal is not present.

INSTRUCTION WORD—A word used to inform the machine what operation to perform. It consists of a command and one or several addresses.

INTEGRAL COMPUTER—A computer dealing with whole numbers, that is numbers above 1 in magnitude.

INTERLOCKING—The process of making sure that when two operations are proceeding at the same time, no interaction can occur between the two parts of the machine performing each of the two operations. In case of conflict one of the operations is made to stop.

JUMP—An instruction which conditionally or unconditionally directs the computer to the next instruction. It usually alters the normal instruction sequencing.

LEAST SIGNIFICANT DIGIT—The rightmost digit of a number.

LOOP—Repetition of a group of instructions in a routine.

MAGNETIC SATURATION—The point where any further increase in the magnetizing force applied to a magnetic material causes an almost negligible change in the magnetic state of the material. A material may be magnetically saturated in one of 2 possible directions.

MALFUNCTION—A failure in operation of the computer hardware.

MANTISSA—That part of a floating point representation which specifies the significant digits of the number.

MARGINAL CHECKING—Preventive maintenance procedure in which certain operating conditions, such as supply voltage, are varied about their normal values in order to detect incipient defective units.

MEMORY—Internal storage forming an integral part of the computer and directly controlled by the computer.

MEMORY INSTRUCTION—An instruction which makes reference to the memory.

MODULUS—One more than the maximum number attainable in any practical counting device.

MODULUS COUNTER—A device which produces an output pulse after a certain number of input pulses or multiples of this number are applied.

MOST SIGNIFICANT DIGIT—The leftmost digit of a number.

NONDESTRUCTIVE READ-OUT—Storage in which bits remain stored even after they are read out.

NONERASABLE STORAGE—Storage which does not allow the bits to be erased.

NONVOLATILE—Storage which does not lose its information even when the power is turned off.

NONRETURN-TO-ZERO MAGNETIC RECORDING—Magnetic recording in which a ONE is written when current flows in one direction, and a ZERO is written when current flows in the opposite direction. Current passes through the zero point only when a ONE is followed by a ZERO, or a ZERO is followed by a ONE.

OCTAL NOTATION—A system for representing numbers in the radix of 8.

OFF-LINE OPERATION—Operation of a device which may proceed concurrently with computer operation.

ON-LINE OPERATION—Operation of a device which is tied directly with computer operation.

OR ELEMENT—A device which produces an output if one or all of its input lines are energized. This circuit performs the inclusive OR function.

ORDER—Rank of bit position in a coded decimal system.

ONE-ADDRESS INSTRUCTION—An instruction word with only 1 address. The address indicates one of the operands upon which the operation is to be performed.

OVERFLOW—The occurrence of a carry from the most significant position of a word.

PARALLEL-SERIAL TRANSMISSION—All digits are transmitted in parallel; but the bits of the digits are transmitted serially.

PARALLEL STORAGE—Storage in which all bits, characters or words are essentially equally available in space. Time is not one of the coordinates.

PARALLEL TRANSMISSION—All bits of a word are transmitted simultaneously.

PARITY CHECK—Comparison of sum of 1 bits in a coded representation to see if it is odd or even.

PROGRAM—Detailed plan for solution of a problem.

PYRAMID OR TREE—A decoding network which looks like a pyramid.

QUARTER-ADDER—A device which produces a ONE when its two input bits are unequal and a ZERO when equal. A binary comparator.

RADIX—See **BASE**.

RANDOM ACCESS DEVICE—A device in which it takes as long to obtain access to one address as it does to any other address.

READING—The process of determining what bits or digits are stored in a storage device. The information may or may not be erased during reading.

RECORDER—A device which is used to manually write onto a document or storage medium.

RECTANGLE—A modified matrix.

REDUNDANCY CHECK—A check which uses extra digits to help detect malfunctions and mistakes.

REGISTER—A device or circuit used to store temporarily numbers consisting of several digits.

REMANENCE—The magnetic state in which a magnetic material falls when the magnetizing force is removed. Materials have positive and negative remanence states.

RING COUNTER—A device consisting of a loop of interconnecting bistable elements, only one of which can be in a specified state at any time. Each successive applied pulse causes another element in the loop to switch to the specified state.

RETURN-TO-ZERO MAGNETIC RECORDING—Magnetic recording in which a ONE is written by a pulse of current rising to a level and then returning to zero; and a ZERO is written by a pulse of current rising to a level in the opposite direction and then returning to zero.

RELATIVE ADDRESS—An address which is easily converted to an absolute address by adding or subtracting a factor.

ROUTINE—A set of instructions arranged in sequence which directs a computer to perform a certain operation or series of operations.

SAMPLING INSTRUCTION—An instruction which looks at the state of the input register. If it is not full, the next instruction in sequence is executed. If it is full, the contents are transferred to the memory and an instruction out of sequence is executed next.

SELECTOR—A circuit which chooses instructions to be executed.

SELF-CHECKING CODE—A code in which it is easy to check each character because of a redundancy in each character.

SELF-COMPLEMENTING CODE—A code in which all numbers may be converted to their 9's or 1's complements by inverting 1's and 0's.

SENSE AMPLIFIER—An amplifier plus the associated circuitry required to convert storage output signals into a form which makes them understood by the rest of the computer.

SEQUENTIAL ACCESS STORAGE—A storage device whose word locations are arranged in time in such a manner that access to any one location may normally be obtained only once during an operation.

SERIAL-PARALLEL TRANSMISSION—All digits are transmitted serially; but the bits which make up the digits are transmitted in parallel.

SERIAL STORAGE—Storage in which time is one of the coordinates used to locate any given bit, character or word.

SERIAL TRANSMISSION—All bits of a word are transmitted sequentially.

SIGNIFICANCE—Rank of digit position in a number.

SQUARE HYSTERESIS LOOP—A hysteresis loop where the upper and lower saturation lines are fairly flat.

STATIC STORAGE—A storage device which stores bits by staying in one of several stable states.

STORAGE ELEMENT—A device which is used to store bits or digits either temporarily or permanently.

STORED PROGRAM COMPUTER—A computer whose program is stored in the memory of the machine. The program is usually prepared on an external storage medium and then fed to the memory.

SUBCOMMAND—One of the many operations that must be performed in order to complete a command. *Examples;* opening a gate, resetting a flip-flop, causing a register to shift its contents to the right.

SUBROUTINE—A subunit of a routine. A portion of the routine which causes the computer to perform a well defined operation.

SWITCHING CIRCUIT—A device which produces an output for certain combinations of input.

SYNC PULSE—Pulse produced upon coincidence of input timing pulse from external storage and timing pulse from computer. This pulse is used to gate the associated digit into the input register.

TEMPORARY STORAGE COMPONENT—A component which stores bits for a short time; a delay element.

TEST ROUTINE—A routine which tests for proper functioning of the computer.

THREE-ADDRESS INSTRUCTION—An instruction with 3 addresses. These addresses specify the two operands and the address of the result.

TIMING PULSE—A pulse always occurring at a definite point in a repeated cycle.

TRACK—That portion of a moving type storage medium (tape, drum, etc.) which is accessible to a given writing or reading station.

TRANSLATOR—A network to which are applied signals representing information in a certain code and the outputs of which are signals representing the same information but in a different code.

TRUTH TABLE—A statement of all conditions of a problem in tabular form.

TWO-ADDRESS INSTRUCTION—An instruction word with 2 addresses. One address indicates one of the operands upon which the operation is to be performed; the other address indicates where in memory can be found the next instruction.

UNCONDITIONAL JUMP—An instruction which always causes the machine to execute a specified instruction out of the normal sequence.

UNIT RECORD—A record where a small group of characters is stored as one item. Unit records are easy to file. Punched cards are an example.

VARIABLE WORD—A word composed of a variable number of characters.

VOLATILE STORAGE—Storage which loses its information with time, or if the power is turned off.

WRITING—The process of placing bits or digits into a storage device.

WORD—A set of bits or digits which is treated by the computer as a unit.

WORD SPACE—The space (usually in storage) occupied by a word, plus the space between words if present.

WORD-TIME—The time it takes for a word space to pass by a certain point. Used especially in reference to serial storage devices.

WORD-TIME ADDRESS—Numerical value given to the word space in storage with reference to a certain zero point in time. Used especially in reference to serial storage devices.

WRITE AMPLIFIER—An amplifier plus the associated circuitry required to convert computer information signals into the form needed for driving a storage device.

Index

Abacus, 3
Access, coordinate, 256
  cyclic, 256, 273
  random, 258, 273
  sequential, 257, 273
  time, 257, 273
Accumulator, 301, 325
Accuracy of computer, 7
Acoustic delay, 210
Addend, 57
Adder, abacus, 3
  binary, *see* Binary adder
  coincidence, 300
  counter type, 300
  half, 86, 91, 97, 106
  parallel, 301
  quarter, 86, 91, 97
  serial, 301
  serial-parallel, 305
  specimen computer, 360
Addition, algebraic, 56, 307
  binary, 35
Address, 241, 252
Address selection, 258, 332, 353
Algebra of classes, 82
Alphanumeric code, 52, 111
Amplification, 93
Amplifier, magnetic, 167
Amplifying circuits, electromagnetic, 167
  relay, 109
  transistor, 199
  vacuum tube, 135
Analog computer, 7
Analog-to-digital converter, 287
AND element, cryotron, 212
AND element, expression, 82
  inverted, 84
  logical block, 89
  mechanical, 101
  relay, 101
  semiconductor diode, 213
  truth table, 90
  vacuum tube, 130
AND-OR relationship, 93
Approximation formula for square root, 68
Arithmetic, binary, 35
Associative law, 83
Asynchronous operation, 334, 346
Augend, 57
Auxiliaries, input and output, 286

B box, 339
B register, 339, 346, 370
B register instruction, 339, 370
Band, 179
Base, 29
Binary adder, block diagram, 91, 92, 95
  cryotron, 213
  definition, 97
  expressions, 87, 91, 107, 132
  relay, 107
  truth table, 87
  vacuum tube, 132
Binary arithmetic, 35, 69
Binary comparator, definition, 85, 97
  expression, 85
  logical block diagram, 90, 95
  relay, 105
  truth table, 85
Binary number system, 33

Binary-octal equivalents, 42
Binary point, location of, 71
Bistable devices, as components for logic, 209
Bit, 33, 45
Block, 283, 298
Branch instruction, 335, 346
Breakpoint, 343, 346
Building blocks, functional, 228
  logical, 226
Bus, memory, 261

Calculating machines, 4
Carry, end-around, 58, 70
  propagation of, 302
Casting out 7's, 75
Casting out 9's, 75
Cathode follower, 131
Cathode ray tube, 125
Channel, 245, 253
Characteristic, of a number, 239
Checking, blank column and double punch, 296
  casting out 7's, 75
  casting out 9's, 75, 323
  comparison, 296
  echo, 296
  forbidden combination, 51, 54, 197
  forbidden operation, 324
  inverse arithmetic, 323
  language, 247
  parity, 51
  self-, 18
  specimen computer, 354
Circuit, adder, 107, 132, 301, 305, 360
  amplifying, 109, 135, 167, 199
  AND, 101, 131, 213
  binary comparator, 105
  complementer, 135
  counter, 143, 146, 176, 230
  decoder, 104, 162, 193, 217
  delay line, 123, 210
  differentiator, 122
  encoder, 103, 162, 194
  flip-flop, 94, 139, 143, 177, 203, 215
  half adder, 106
  INHIBIT, 101, 130, 166, 190
  OR, 101, 130, 163, 192, 213
  pulse reshaper, 136
Circuit, register, 114, 140, 174, 181, 204
  selector, 162
  storage, 136, 168, 173, 177, 203, 210, 218
  switching, 128, 130, 161, 192, 195, 213
Clock pulse, 167, 184, 273
Code, alphanumeric, 52
  command, 249
    for specimen computer, 374
  Excess 3, 46
  Excess 6, 46
  modified bi-quinary, 52
  modified qui-binary, 52
  nonweighted 4-bit, 46
  self-checking, 51
  self-complementing, 50
  7-bit, 52
  teletype, 53
  two out of five, 52
  weighted bit, 45
Coded-decimal notation, 45, 54
Coincident-current storage, 168, 173, 184
Coincident electric storage, 218
Column address, 258
Command, 241, 252
Command code, *see* Instructions
Command translator, 331
Communication devices, 295
Commutative law, 83
Comparison instruction, 337, 346
Complement form for numbers, 239
Complementer, 5421 code, expression, 87
  truth table, 88
  vacuum-tube circuit, 135
Complements, principle of, 46
  1's and 2's, 50, 54, 82, 231
  9's and 10's, 49, 54
Computer, analog, 7
  digital, 7
  fractional, 239, 252
  functional organization of, 13
  general-purpose, 251
  integral, 239, 252
  logic, characteristics of component for, 209

Computer, special purpose, 251
  specimen, characteristics of, 347
Conditional halt, 338, 346
Conditional jump, 335, 346
Console of specimen computer, 371
Continuous record, 285, 298
Control, arithmetic, 314, 317
  asynchronous, 333
  input, 276
  output, 276
  of specimen computer, 366
  synchronous, 333
  unit, 326
Converter, analog-to-digital, 288
  definition, 298
  digital-to-analog, 290
  magnetic tape to high-speed printer, 294
  mechanical encoding, 289
  punched card to magnetic tape, 291
  time encoding, 288
Coordinate access storage, 256, 273
Core, magnetic, 155
Core storage array, 168
Corrector, for binary-coded decimal adder, 305
Counter, backward counting, 231
  binary, 143
  cycle, 329
  decade, 223
  decimal, 146
  electromagnetic, 176
  forward counting, 230
  modulus, 176, 230
  program, 327
  ring, 146, 230
  vacuum tube, 143
Counting, 29
Counting board, 3
Cryotron, 212
Crystal, germanium, 185
Cycle counter, 329
Cycle, machine, 327, 331, 335, 352
Cycle time, 257, 273
Cyclic access storage, 256, 273

Data word, 238
Decade counter, 223
Decimal point, location of, 71
Decoder, cryotron, 217
  definition, 103, 118
  diode matrix, 192
  electromagnetic, 162
  excess-3 to decimal, 193
  relay tree, 103
Delay lines, acoustic, 210
  electric, 123
Detector, definition of, 232
  end-of-block, 232, 265
  forbidden combination, 5421 code, 197
Diagnostic routine, 345
Difference, 57
Differentiator, 122
Digit, 1, 54
Digital-to-analog converter, 290
Diode, germanium, 188
Diode translator, 195
Disk, magnetic, 182
Distributive law, 83
Division, mechanization of restoring method, 318
  nonrestoring method, 63
  pencil and paper method, 61
  repeated subtraction method, 61
  restoring method, 62
Domain theory, 153
Dot convention for magnetic cores, 155
Double-rank shift register, 204, 207
Doughnut-shaped core, 155
Drum, magnetic, 179, 269, 286
Duodecimal notation, 32
Duplication, checking by, 75
Dynamic flip-flop, 177, 184
Dynamic storage, 177, 184

Electro-optical devices, 219
Electrostatic decade counter, 223
Emitter follower, 191
Encoder, decimal-to-binary-coded decimal, 194
  definition, 103
  diode matrix, 194
  relay, 103
  wired-core, 162
End-around carry, 58, 70

Excess-3 code, 46
Excess-6 code, 46
EXCLUSIVE OR, definition, 81, 97
  logical block diagram, 90
  truth table, 86
Execute phase, 327, 367
Exponential numbers, 29, 239
External storage, 255, 285, 298
Extract, 321, 325

Ferrite-apertured plate, 171
Ferrite core, 153
Ferroelectric storage, 218
Fetch phase, 327, 366
Fixed-point computer, 239, 252
Fixed word, 240, 252
Flip-flop, complementing, 144
  cryotron, 215
  dynamic, 177
  standard, 94
  transistor, 203
  vacuum tube, 139, 144
Floating point representation, 239, 252
Forbidden combination check, 51, 55
Forbidden combination detector, 5421 code, 197
Formal logic, 79
Fractional computer, 239
Functional building blocks, adder, 233
  comparison circuit, 233
  counter, 230
  detector, 232
  gate, 228
  generator, 231
  register, 228
  selector, 231

Gates, 228
General-purpose computer, 251
Generator, clock pulse, 232
  timing pulse, 232
  two's complement, 231
Germanium, 185
Germanium diode, 188
Graphical logic, 79

Half adder, expression, 86
  logical block diagram, 90
  relay, 106
Half adder, truth table, 86, 97
Half-current pulses, coincidence with, 163
Halt instruction, 338
Head, magnetic, 160
Hexadecimal notation, 32
High-speed printer, 294
Hysteresis loop, definition, 152, 184
  ferroelectric, 218
  magnetic core, 152, 184
  nonsquare, 153
  square, 153

Impedance switching devices, 157
Index register, 339
Induction, magnetic, 150
Information rearrangement, 279
Inhibit, electromagnetic, 166
  inverted, 84
  logical block, 89
  relay, 101
  transistor, 190
  truth table, 89
  vacuum tube, 130
Input and output auxiliaries, 286
Input-output, independent operation of, 281
  of specimen computer, 355
Input register, 276
Input unit, 275
Instruction modification, 338
  in specimen computer, 370
Instruction sequencing, 1-address, 327
  2-address, 329
  3-address, 331
  4-address, 331
  in specimen computer, 366
Instruction types, add, 363
  add address, 338
  B register, 339
  branch, 335
  clear add, 363
  comparison, 337
  conditional halt, 338, 346
  for specimen computer, 348
  grouping of, 248
  halt, 338
  input, 355
  jump, 335

Instruction types, memory, 261, 367
  multiply, 364
  nonmemory, 261, 334, 367
  punch, 357
  read, 261
  right shift, 364
  sampling, 281, 337
  set *B* register, 341, 370
  store, 366
  subtract, 363
  test *B* register, 341, 370
  transfer, 261
  write, 261
Instruction word, 1-address, 238, 252
  2-address, 238, 253
  3-address, 238, 253
  4-address, 238, 253
Integral computer, 239, 252
Interlocking, 281, 283, 298
Inverse arithmetic, 75
INVERTED AND, cryotron, 213
  element, 84
  expression, 84
  logical block, 90
  transistor, 195
  truth table, 90
  vacuum tube, 130
INVERTED INHIBIT, element, 84
  expression, 84
  logical block, 90
  truth table, 90
INVERTED OR, cryotron, 213
  element, 84
  expression, 84
  logical block, 90
  transistor, 195
  vacuum tube, 130
Inverter, cryotron, 213
  electromagnetic, 166
  logical block, 89
  relay, 101
  transistor, 199
  truth table, 89
  vacuum tube, 130
Iteration, square root, 68

Jump, unconditional, 335, 346
Jump-no-jump switch, 343
Junction transistor, *n-p-n*, 189
  p-n-p, 191
Language, machine, 15
  structure, 237
  word and number, 23
Latency, 257, 273
Least significant digit (LSD), 29, 45, 54
Left shift, 175
Light amplifier, 222
Loading controls, 342
Loading program for specimen computer, 376
Logic, basic elements of, 79
  blocks, 88
  connectives, 12, 80
  electromagnetic, 161
  graphical, 79
  identities, 82
  laws, 83
  math, 82
  relay elements, 101
  semiconductor elements, 185
  vacuum-tube elements, 128
Logical algebra, 79

Machine cycle, 327
Machine language, 15
Magnetization curve, 151
Magnetostrictive delay, 212
Maintenance of specimen computer, 380
Mantissa, 239
Manual control, 341
Manual input switch for specimen computer, 373
Marginal checking, electromagnetic, 159
  transistor, 191
  vacuum tube, 127
Matching input-output to rest of computer, 280
Mathematical logic, 82
Memory of specimen computer, 352
Mercury tank, 210
Microwave techniques, 223
Minuend, 57
Modified bi-quinary code, 52
Modified qui-binary code, 52

Modular approach, 226
Modulus, 31, 54
Modulus counter, 176
Molybdenum permalloy tape, 157
Molypermalloy core, 153
Most significant digit (MSD), 29, 45, 54
Multiplexing, 283
Multiplication, binary table, 35
  circuit for binary, 312
  circuit for decimal, 316
  method of repeated addition, 60
  pencil and paper method, 58
  right- and left-hand component method, 59
Multistable devices, 223
Mylar tape, 160

Negative numbers, 46
Nonmemory instruction, 334
Nonrestoring method of division, 63
Nonreturn to zero recording, 180, 184
Nonsquare hysteresis loop, 153
Nonweighted 4-bit code, 46
NOT, cryotron, 213
  element, 80
  expression, 82
  electromagnetic, 166
  logic block, 89
  relay, 101
  transistor, 199
  truth table, 90
  vacuum tube, 130
Notation, binary, 33
  binary-coded decimal, 45, 54
Number-comparison program, 373
Numbers, additive, 26
  ancient systems of, 27
  binary, 33
  duodecimal, 32
  exponential, 29, 239
  hexadecimal, 32
  octal, 40, 42
  positional, 25, 31
  precision of, 240
  quaternary, 40
  reflected, 43
  roman, 26
  ternary, 33
Octal-binary equivalents, 42
Off-line operation, 290, 298
On-line operation, 290, 298
One-instruction switch, 343
Operation of specimen computer, 380
Operational controls, 342
OR, cathode follower, 131
  cryotron, 213
  electromagnetic, 163
  element, 81
  emitter follower, 191
  exclusive, 81, 83
  expression, 82
  inclusive, 81
  inverted, 84
  logical block, 89
  relay, 101
  semiconductor, 192, 195
  truth table, 89
  vacuum tube, 131
Order, 45, 54
Ordering of information, 276
Organization of specimen computer, 349
Output register, 276
Output unit, 275
Overflow, exceed capacity, 307, 325
  in division process, 319

Parallel transmission, 245, 253
Parallel-serial transmission, 245, 253
Parity bit detector for specimen computer, 354
Parity bit generator for specimen computer, 354
Parity checking, 51, 55
Pascal, Blaise, 4
Peripheral equipment, 290
Photographic storage, 219
Piezoelectric crystals, 211
Plug-in unit, 227
Plugboard, for converter, 293
Positional number systems, 25, 31
Printed circuit board, 227
Printer synchronization, 294
Printing devices, 112
Program, counter, 327, 336
  debugging, 343

Program, debugging of specimen computer, 378
  examples, 375, 377
  loop, 339
  preparation, 249
Programming, 15, 373
Pulse reshaper, 136
Punched-card-to-magnetic-tape converter, 293
Punched cards, 111
Punched paper tape, 110
  in specimen computer, 357

Quarter adder, definition of, 97
  logical block diagram, 90
  truth table, 86
Quartz crystal delay, 211
Quaternary, 40

Radix, 29, 54
Random access, 258, 273
Read amplifier, transistor, 201, 208
Reader, 277, 286, 298
Recirculating register, drum, 182
  electromagnetic core, 174
Read-write cycle, of specimen computer, 352
Record, continuous, 285
  unit, 285
Recorder, 291, 298
Recording, magnetic, 181
Rectangular diode decoding network, 192
Rectangular storage array, 168
Redundancy, 19, 24, 51, 55
Reflected numbers, 43
Register, arithmetic, 229, 299
  buffer, 229
  definition of, 114, 118, 228
  display, specimen computer, 373
  double rank, 204
  drum recirculating, 182
  electromagnetic, 174
  index, 339
  input, 276
  instruction, 327
  output, 276
  relay, 114
  shift, 140, 163, 204
Register, transistor, 204
  vacuum tube, 140
Relative address, 259
Remanence, 152
Residue modulus nine (RMN), 76
Residue modulus seven (RMS), 77
Restoring method of division, 62
Return to zero recording, 180, 184
Right shift, 174
Roman number system, 26
Row address, 258

Sampling instruction, 337, 346
Saturation, magnetic, 151, 184
Selector, 162, 231
Self-checking codes, 51, 54
Self-checking machine, 18
Self-complementing codes, 50, 55
Semiconductors, 185
Sense amplifier, transistor, 201
Sequencing of instructions, 327
Sequential access storage, 257, 273
Serial-parallel transmission, 245, 253
Serial transmission, 245, 253
Set *B* register instruction, 341, 370
Significance, 29, 45, 54
Space address, 259
Special-purpose computer, 252
Specimen computer, add, clear add, and subtract, 363
  address selection, 353
  arithmetic, 357
  characteristics of, 347
  command code, 374
  control, 366
  debugging, 378
  input-output, 355
  instruction sequencing, 366
  loading, 376
  M register, 354
  manual input, 373
  memory, 352
  number comparison routine, 375
  operation and maintenance, 380
  organization, 349
  parity bit detector and generator, 354
  program, 373
  punch instruction, 357
  read-write cycle, 352

Specimen computer, register display, 373
right shift and multiply, 364
store instruction, 366
use of *B* register, 370
using computer, 371
Speed of computation, 5
Square hysteresis loop, 153, 184
Square root, approximation formula, 68
pencil and paper method, 65
subtractive process, 66
Storage, access, 256
characteristics of components for, 266, 285
coincident current, 168, 173, 267
coincident electric, 218
destructive read out, 267, 273
dynamic, 266, 272
erasable, 266, 272
external, 255, 285, 298
need for, 93
nondestructive read out, 267, 273
nonerasable, 266, 272
nonvolative, 266, 273
static, 266, 272
volatile, 266, 273
Storage devices, cathode ray, 125
chemical, 223
diode-capacitor, 136
dynamic storage, 177
electromagnetic, 168
electromechanical, 110
electro-optical, 219
ferrite, 212
ferrite-apertured plate, 171
ferroelectric, 218
fused quartz, 211
magnetic disk, 179
magnetic drum, 179
magnetic tape, 179
magnetostrictive, 212
mercury tank, 210
microwave, 223
nickel ribbon, 212
photographic, 219
transistor, 203
twistor array, 173
Subcommand, 331, 346
Subtraction with the aid of complements, 48, 309
Subtrahend, 57
Sum, 57
Superconductive devices, 212
Switching circuits, electromagnetic, 161
electromechanical, 101
resistor, 128
semiconductor diode, 192
superconductive, 213
tetrode, 130
transistor, 195
triode, 130
vacuum diode, 129
Syllogism, 12, 34, 79
Sync pulse, 277, 298
Synchronization, 277, 294
Synchronizer, 277
Synchronous operation, 333, 346

Tape, magnetic, 179
Tape recording, 160
Teletype code, 52
Ternary, 40
Test accumulator contents, 335
Test *B* register instruction, 341
Time, access, 257
Time address, 259
Timing problems, input-output, 277
memory, 263
Timing pulse, 180, 184
Toroid, 155
Track, 179
Transfer instruction, 261
Transfer of block of words, 264, 283
Transformer action, 150
Translation, binary to decimal, 38
binary to octal, 42
decimal to binary, 36
of commands and addresses, 331
write and read circuits, 262
Translator, semiconductor diode, 193
Transmission, parallel, 244
parallel serial, 245
serial, 244
serial parallel, 245
Transmission line, 123
Tree encoder, relay, 103
Trouble-shooting controls, 344

Truth table, binary adder, 87
binary comparator, 85
complementer, 5421 code, 88
EXCLUSIVE OR, 85
half-adder, 86
purpose of, 85, 97
quarter-adder, 85
Twistor, 157, 173
Two out of five code, 52

Unconditional jump, 335, 346
Unit record, 285, 298
Universal class, 82
Use of specimen computer, 371

Variable word, 241, 252
Verifier, 296
Voltage switching devices, 157

Waiting time, 257
Weighted 4-bit codes, 45
Weights, octal, 41
of bit positions, 33, 39
of decimal digit positions, 30
of quaternary digit positions, 41
of ternary digit positions, 41
Wired core encoder, 162
Word, data, 238, 252
instruction, 237
size, 240
space, 271
time, 271, 273
Word selection technique, 173
Write amplifier, transistor, 201, 208
Writer, 286, 298
Writing, in specimen computer, 354

ZERO, 26