how to build a working digital computer

Edward Alcosser
James P. Phillips
Allen M. Wolk

HAYDEN BOOK COMPANY, INC., NEW YORK

Second Printing, 1968

Printed in the United States of America

Preface

The new and exciting field of digital computers has expanded so much in recent years that almost everyone is now affected by it. The need to understand computers experienced by people of widely diversified vocational and academic backgrounds has presented one of the most challenging jobs of teaching ever to be tackled by authors, lecturers, and teachers alike.

As with any new and complicated subject, the first texts to be introduced are usually effective at the college and postgraduate levels. This book, *How to Build a Working Digital Computer,* is the result of an endeavor to present to the non-engineer the basic facts concerning digital computers, their uses, and how they work. It is specifically aimed at a reader with an interest or need to understand this subject but with no formal training or education in computer technology. *How to Build a Working Digital Computer* is equally aimed at hobbyists, technicians, secondary school students, and college students with no computer background.

In presenting the topic of basic computer technology, a unique method of "learning by doing" is employed. Since there is no effective substitute to proving out theory in the laboratory, this book shows the reader how to construct a working model of a digital computer, using simple inexpensive components usually found around the house or in a neighborhood electrical parts store. This computer is divided into basic units corresponding to the actual working sections of a computer. The design and operating principles of each computer section are explained in detail using examples and simple experiments and the corresponding model computer unit is then constructed and used to illustrate the theory discussed in the text.

In this manner, a complete working digital computer, able to add, subtract, multiply and perform many other complicated functions at the reader's discretion, is constructed. The text then discusses the topic of programming and several basic computer programs are developed for use with the model computer. The reader may write his own programs, load them on his own working computer, and run the programs, providing experience and proving the principles expounded in the text.

With this text, hobbyists, students, technicians, and just ordinary people will find a simplified, enjoyable, and effective approach towards learning and really understanding the basic principles of modern digital computers.

We would like to acknowledge the work done by Scalor Publications in helping to produce the art and tables used in the text.

June, 1967

THE AUTHORS

Table of Contents

Chapter 1
INTRODUCTION

THE DIGITAL COMPUTER

The progress of man is directly related to the tools and implements he uses in his work. The extremely large and rapid strides being made today in the development of our technology may be attributed in part to the coming of age of the electronic, programmable, digital computer. This is a "tool" that can be taught or programmed to solve a variety of complex problems rapidly and with great accuracy. In this text, we will detail the construction of a working model of a digital computer and demonstrate the operation of this important machine through discussion and experiment.

Modern computers can perform billions of basic arithmetic operations such as addition or subtraction each second without error or fatigue. They are able to manipulate extensive amounts of data at speeds that operators of calculating machines cannot match. Accuracies achieved are typically rated in terms of one part in a million. Such awesome capabilities have led to use of computers in countless areas. As of April, 1965, there were approximately 18,000 computer installations in this country serving the government, science, schools, and industry.

The Internal Revenue Service has estimated that tax revenues were increased by 52.5 million dollars in 1964 because computers were used to process Federal income tax returns. Using computers to make blood tests in laboratories enables technicians to perform tests in two minutes that would ordinarily take two to three hours. A publishing firm that now uses a computer to process book orders estimates that it has reduced its order-processing time by 75 percent. The spectacular achievements of our various space programs can be directly related to the computer control of different phases of each operation. The highly satisfactory performances of the Ranger, Gemini, and Mariner missions would not have been possible without computers.

The influence the digital computer is exerting on our society is demonstrated every day on the front pages of our newspapers and in countless articles and columns relating new computer applications. The potential future applications of the digital computer will make this present widespread use seem small. Before proceeding with our study of the computer, it is well to understand the history of the computer's development and the relationships between this development and the culture and technology of these times.

A BRIEF HISTORY OF NUMBERS AND CALCULATORS

In earliest times, man most likely counted on his fingers. If he wanted to indicate that he had killed two tigers, he would hold up two fingers. His earliest counting methods were probably limited to totals below five. In time, sticks and pebbles were used to extend the range of counts, but the simplicity of the existing social organization kept such usage at a very low level of sophistication. Number symbols and methods of computation were unknown and not needed.

As the farmer and tradesman superseded the hunter, man's requirements for counting and recording his counts for future use became more complex. Simple tallying methods were replaced by systems in which different symbols were used to represent different counts and these symbols were added and subtracted. The early Egyptians (circa 3400 B.C.) devised such a number system using symbols called *hieroglyphics*. The hieroglyphic symbols representing the numbers 1 through 10 are shown in Fig. 1-1. It is interesting to note the relationship between these written symbols and counting tallied on the fingers on a hand.

The Romans developed a system of numbers, also illustrated in Fig. 1-1, which should be familiar to you because of its use on clock faces, as chapter numbers, and so on. Calculating with this number system was slow and awkward since large numbers required many symbols. For example, the number 9876 in Roman numerals is MMMMMMMMMDCCCLXXVI (M = 1000, D = 500, C = 100, L = 50, X = 10, V = 5, I = 1).

The symbols of the numbering system that we use today, the decimal, are probably of Hindu-Arabic origin and are therefore referred to as *Arabic numerals*. Figure 1-1 shows early Hindu and Arabic number symbols for the period about the year 900 A.D. The most important mathematical development attributed to the Hindus is the origin of the concept of the zero, 0.

MODERN	EARLY EGYPTIAN	EARLY ROMAN	EARLY HINDU	EARLY ARABIC
1	𓏺	I	꣑	١
2	𓏺𓏺	II	꣒	٢
3	𓏺𓏺𓏺	III	꣓	٣
4	𓏺𓏺𓏺𓏺	IV	꣔	٤
5	𓐄	V	꣕	٥
6	𓐄𓏺	VI	꣖	٦
7	𓐄𓏺𓏺	VII	꣗	٧
8	𓐄𓏺𓏺𓏺	VIII	꣘	٨
9	𓐄𓏺𓏺𓏺𓏺	IX	꣙	٩
10	𓎆	X		

Fig. 1-1. Early number symbols.

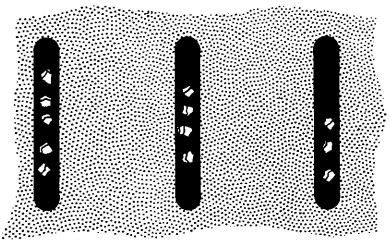


Fig. 1-2. An Egyptian sand calculator.

The development of various number symbols and counting systems was matched by the introduction of counting and calculating aids other than sticks or pebbles. For example, to perform calculations the Egyptians developed a sand calculator consisting of columns of grooves in the sand, as illustrated in Fig. 1-2. The right-hand groove represented quantities from 0 to 9, depending upon the number of pebbles contained in the groove. For example, if four pebbles were contained in the groove, the number represented was 4. The next groove to the left represented 10 times the number of pebbles in the groove. Thus, if there were three pebbles in this groove and one in the right-hand groove, the number 31 was represented. Each succeeding groove to the left represented numbers 10 times that of the preceding groove. Hence, the grooves represented the 1's, 10's, 100's, and 1000's columns, respectively, and so on. This system of counting, the *decimal system,* is the basic system we employ today.

The Egyptians were not alone in the development of simple calculators. Around 600 B.C., the Chinese developed a calculator called an *abacus* (Fig. 1-3). This device consists of a wooden frame strung with beads on wire columns. Each column represents an order or place in the decimal numbering system in the same way the grooves represented decimal places in the sand calculator. The abacus is still used today and skilled operators manipulate it at speeds comparable to those of a mechanical or electric desk calculator.

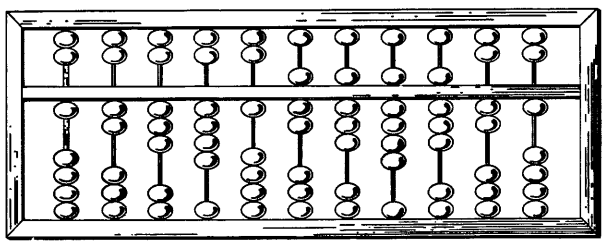


Fig. 1-3. An abacus.

As civilization developed, calculating devices became more sophisticated and, correspondingly, demanded a higher degree of intelligence and knowledge from their users. The design theory and potential range of applications of some computing machines developed before the 19th century were similar to that of present-day computers, but the efficiency and speed of the machines was very low.

The first truly mechanical calculator, invented by Blaise Pascal, a French philosopher and mathematician, around 1642, was an adding machine that operated in the decimal numbering system (Fig. 1-4).

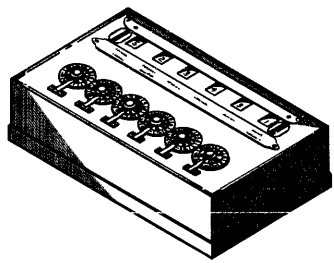


Fig. 1-4. Pascal's calculator.

It consisted, essentially, of a series of wheels which added and carried 10's. Shortly after Pascal's invention, another philosopher and mathematician, Gottfried von Leibnitz, developed a gear-operated calculator, the *stepped reckoner,* which allowed for the carry operation between orders or places (Fig. 1-5). In Pascal's machine, multiplication was performed by repeated addition; in Leibnitz's machine, multiplication could be performed directly. Both machines were operated manually.

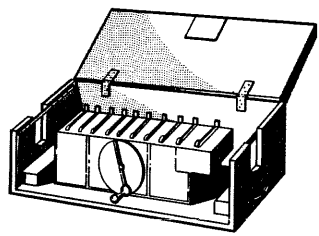


Fig. 1-5. Von Leibnitz's stepped reckoner.

In the early 1800's, an English mathematician, Charles Babbage, worked on an "engine," or calculator, that would solve complex mathematical problems automatically. However, the technology of his time was not capable of producing the accurate parts required to make the calculator work.

Around 1890 an American scientist working for the Census Bureau, Herman Hollerith, devised a card containing punched holes that could be read or interpreted by an electrical device (Fig. 1-6). Each hole punched into the card represented a different piece of information. After the cards were punched, the data contained on them were read by an

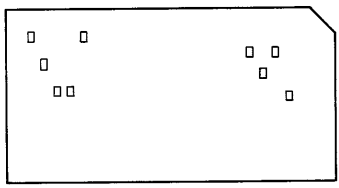


Fig. 1-6. A punched card.

electrical scanner which then was able to sort, count, and tabulate the information. This punch card method is used extensively today in data processing applications and was applied successfully to the tabulation of data gathered during the Census of the United States in 1890.

COMPUTERS IN THE 20th CENTURY

After 1890, the development of various types of calculators and data-processing machines proceeded rapidly. However, the machines remained relatively crude as late as the 1920's. They were not capable of performing the long sequences of operations required to solve complex problems and were operated by fairly constant human control.

The first successful computer capable of performing these operations was developed during the period between 1939 and 1944 by the International Business Machines Corporation in cooperation with Harvard University. It was known as the *Automatic Sequence-Controlled Computer* or the *Harvard Mark I.* It is believed that the development of this machine was based upon the work done earlier by Charles Babbage and was permitted by the advance of technology by the 1940's which allowed the accuracy in fabrication not available in Babbage's time.

This first, large-scale computer was essentially mechanical in nature using gears, cams, and shaft rotations as the calculating devices, and, therefore, it was relatively slow and bulky. Since then, the use of electronic components has allowed the building of computers that are much faster and less bulky. The first electronic computer was completed in 1946 by the Moore School of Electrical Engineering in Philadelphia and was called ENIAC (*Electronic Numerical Integrator and Calculator*). The successful design and fabrication of ENIAC represented a major accomplishment, as the technology of that time was such that it was thought vacuum tubes in a large-scale computer would fail more quickly than they could be found and replaced. Engineers were then having difficulty keeping radar sets containing 100 tubes operating

properly. However, in spite of this, ENIAC did operate successfully with approximately 18,000 tubes.

The next major step in the evolution of the digital computer was the advent of the *stored-program machine.* The word *program* refers to the series of operations that the computer is instructed to perform. A stored-program machine is built with a facility for storage that allows the insertion of a program at the option of the operator. In this way the solutions to many different problems may be "programmed" and saved for use at appropriate times. Thus, if a problem must be solved for which a program already exists, the operator need only insert the existing program into the computer to obtain the required solution. The idea of the stored-program machine was introduced in 1945 by Dr. John von Neumann. The first stored-program machine was built by a group headed by M. V. Wilkes at Cambridge University in England.

Today's computers are capable of extremely high speeds and accuracy and are used in applications ranging from payroll processing to the guidance of spacecraft. As our technology advances, great strides are being made to increase still further the capability of our computers in terms of information storage capacity, accuracy, and speed. This increase in speed and accuracy allows the computers to solve larger and more complex problems at economical cost and within the most stringent time requirements. Almost every day, major developments are applied toward the creation of computers that can solve greater and more challenging problems. Problems that would require lifetimes to be solved manually are now being solved quickly and efficiently by modern computers. The excitement of our times is certainly aided by the extension given to our minds by computers.

ORGANIZATION OF THE COMPUTER

A digital computer is composed of five basic functional units, as illustrated in Fig. 1-7, the input, storage, arithmetic, control, and output

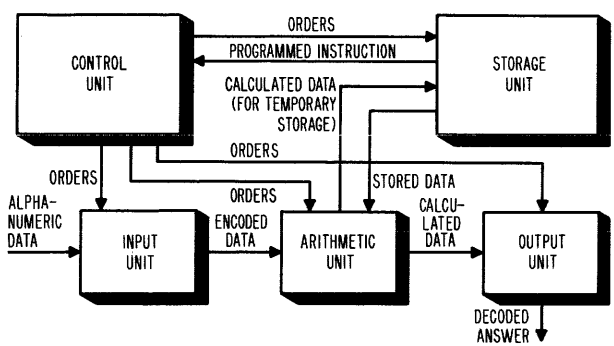


Fig. 1-7. A typical computer's functional diagram. The arrows indicate the direction in which information travels through the computer.

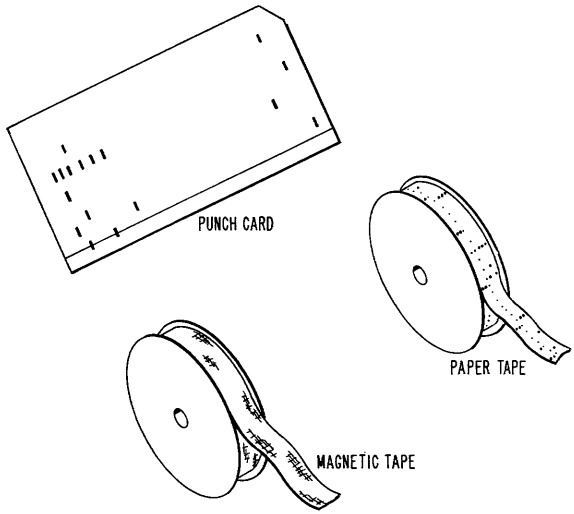


Fig. 1-8. Input methods: (A) paper tape; (B) punched cards; (C) magnetic tape.

units. They will be described briefly here and considered in detail in the chapter in which the corresponding unit is constructed for our own computer model. The function of the computer is to accept alphabetical-numerical (alphanumeric) data, perform calculations and other processing of these data, and produce specific alphanumeric outputs.

The alphanumeric input data are usually prepared in the form of coded, punched paper tape or cards (Fig. 1-8), or magnetic tape. This information is then converted to electrical signals coded in the binary format (or computer language) used by the computer. The computer unit that receives the alphanumeric data and converts them into the binary format, a process called encoding, is called the *input unit.* (This unit is described completely in Chapter 2.) The input unit stores the data until the computer is ready to perform as instructed.

Once the input data have entered the input unit, their paths through the computer and the operations performed upon them are completely determined by the stored program, which indicates the sequence of operations required to perform the desired computations. The program is stored in coded form in the computer's *storage unit.* Each operation is read from this storage unit, in sequence, and used by a *control unit* to issue instructions (orders) to the remaining units in the computer. These instructions cause the computer to perform the programmed operation read from the storage unit. Hence, a stored operation such as "ADD $x + y$" will cause the control unit to generate instructions to the arithmetic unit causing that unit to add the quantity x to the quantity y.

The storage unit usually consists of three storage areas: program storage; permanent data storage; and scratch-pad storage. The first area, *program storage,* is used to store the program for the particular computer application. Programmed operations are stored in a particular fixed sequence so that they may be read from storage automatically in the proper order. The second area, *permanent data storage,* is used to store various constants and other data required by the program for solution of the problem. The third area, *scratch-pad storage,* is a temporary storage area used by the program to store intermediate results developed during the course of the problem solution.

The various types of memory devices are described in Chapter 5 and a simulated magnetic drum and a core memory unit are designed and constructed. The control unit for our computer is described and constructed in Chapter 6.

The *arithmetic unit* performs all of the basic arithmetic operations, such as addition, subtraction, multiplication, and division, under program control. Complex mathematical techniques and methods such as the use of calculus are implemented by reducing these techniques to sequences of basic arithmetic operation. (The arithmetic theory of digital computers is discussed in Chapter 4, when the arithmetic unit for our computer is constructed.) The output of the arithmetic unit is either stored in scratch-pad storage or read out through the *output unit.*

The output unit is just the opposite of the input unit in that it performs the function of converting the alphanumeric data from the computer's internal format to that required by the outside environment. Output data are fed to peripheral units such as magnetic tape recorders, printers, typewriters, or card punches, depending on the use these data are put to. (Our computer's output unit is discussed and constructed in Chapter 2.)

A SAMPLE COMPUTER PROGRAM

Any computer is built specifically to perform a particular set of operations. The list of these operations is called the computer's *instruction repertoire,* as it lists the computer's capabilities in terms of the operations it can perform. The instructions contained in the computer's repertoire can be arranged (programmed) in many different sequences to perform all sorts of complex calculations. To illustrate this process, we shall assume that we have a computer that is capable of performing the following instructions:

1. Add the contents stored at location A to those stored at location B.
2. Subtract the contents stored at location B from those stored at location A.

3. Read in data to the input unit.
4. Read out data to the output unit.
5. Store data in location *A* or in location *B*, whichever is indicated.

Let us arrange these instructions to develop a program that calculates the solution to the following problem:

$$T = x + y - z$$

where x, y, and z are any three numbers. For example, if $x = 5$, $y = 3$, and $z = 4$, the solution to the problem is:

$$\begin{aligned} T &= 5 + 3 - 4 \\ &= 4 \end{aligned}$$

For illustrative purposes we will use these three numbers in our sample program.

First, we must read in and store our numerical inputs, 5 and 3:

1. Read in 5 to the input unit. (*Instruction 3*)
2. Store 5 in storage location *A*. (*Instruction 5*)
3. Read in 3 to the input unit. (*Instruction 3*)
4. Store 3 in location *B*. (*Instruction 5*)

Next, we must add 5 and 3 and store the result:

5. Add the contents of *A*, 5, to the contents of *B*, 3, for a total of 8. (*Instruction 1*)
6. Store 8 in location *A*. (*Instruction 5*) This process writes over the number previously stored in *A* so that only the new number remains.

Finally, we must subtract 4 from 8 and read the numerical answer out:

7. Read in 4 to the input unit. (*Instruction 3*)
8. Store 4 in location *B*. (*Instruction 5*)
9. Subtract the contents of location *B*, 4, from the contents of location *A*, 8, for a remainder of 4 ($T = 4$). (*Instruction 2*)
10. Read out 4 to the output unit. (*Instruction 4*)

With this 10-step program inserted in program storage, using x, y, and z instead of set numbers, the computer will automatically calculate T for any values of x, y, and z, within its capabilities.

A diagrammatic representation of the manner in which the data used in this program move through the computer is shown in Figs. 1-9 through 1-14. When the computer is started, the first instruction in the program is read from program storage. This causes the control unit to order the input unit to read 5 (or x) into its own temporary storage facility (Fig. 1-9). If the input data are in decimal form, the input unit encodes these data into the binary form used by the computer. (Encoding is covered in detail in Chapter 2.)

The program's second instruction causes the data, 5 (or x), to

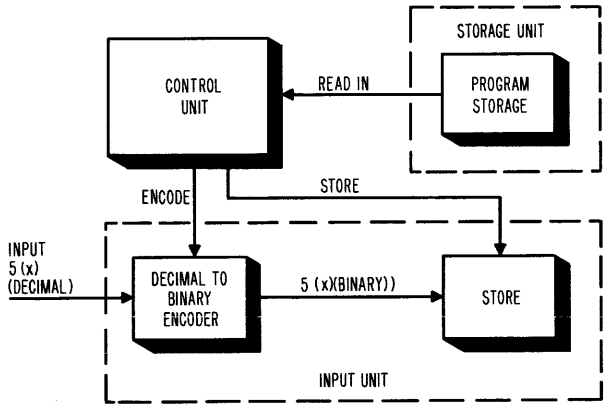


Fig. 1-9. Read in x.

be transferred from the input unit to scratch-pad storage location *A* (Fig. 1-10). Sometimes this transfer may be accomplished via the arithmetic unit for convenience.

Instructions 3 and 4 of the program cause the computer to perform in the same manner as with 1 and 2, reading in and storing 3, or *y*, in location *B* of scratch-pad storage.

The program's fifth instruction causes both the contents of *A* and *B* to be transferred to the arithmetic unit where they are added and their sum stored in the accumulator, a short-term storage element contained in the arithmetic unit (Fig. 1-11). The sixth instruction causes the sum to be transferred to scratch-pad storage location *A* (Fig. 1-12).

Instructions 7 and 8 of the program read in and store data, 4 (*or z*), in location *B* in the same manner as instructions 1 and 2 or 3 and 4.

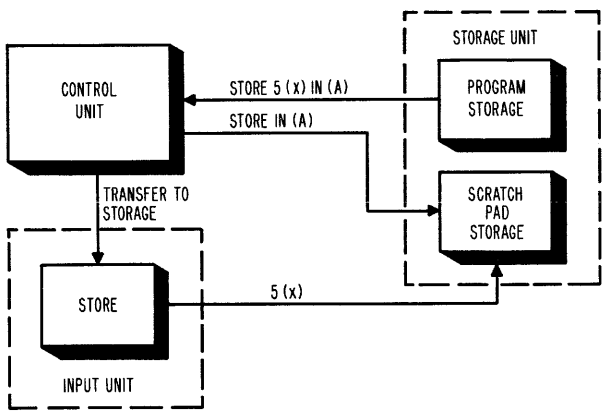


Fig. 1-10. Store x in A.

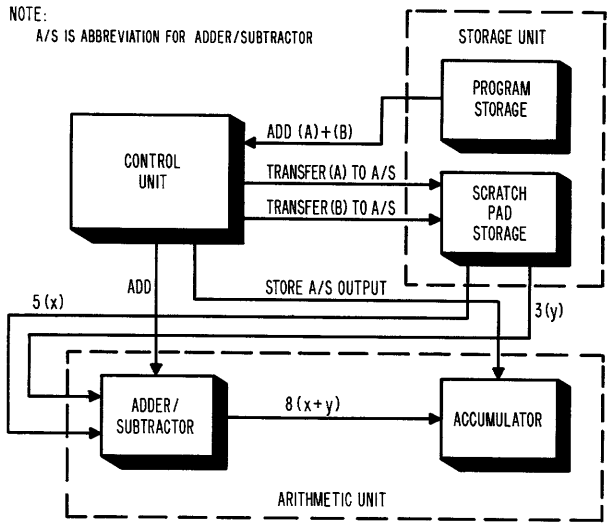


Fig. 1-11. Add A to B.

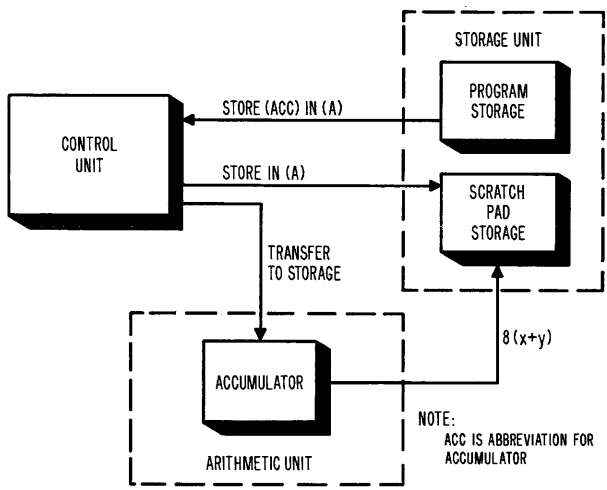


Fig. 1-12. Store in A.

Instruction 9 causes the contents of B to be subtracted from those of A with the difference 4, T, being temporarily stored in the accumulator (Fig. 1-13). Finally, the tenth and final instruction in the program causes the computer to read out the solution through the output unit (Fig. 1-14). In this case we are assuming that output data are decoded to decimal form.

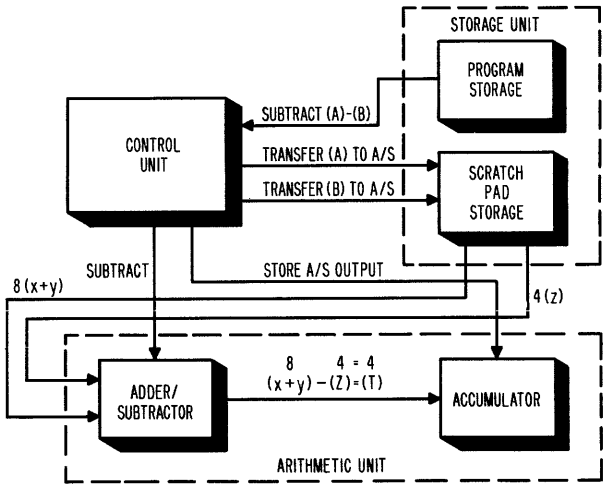


Fig. 1-13. Subtract B from A.

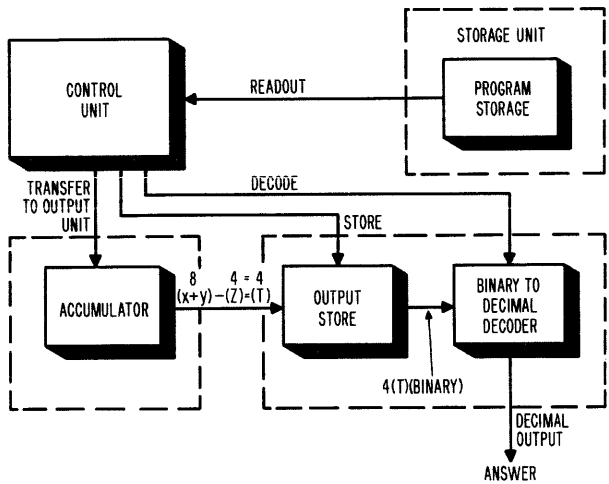


Fig. 1-14. Read out.

Chapter 2
COMMUNICATING WITH THE COMPUTER

The only "language" that the computer uses is numerical. All of the information fed into it must be converted to number form. However, as indicated in Chapter 1, the digital computer does not operate in the decimal number system. Any numerical data given to it must be translated from our decimal language into the binary language of the computer. All information taken from the computer must be translated from the machine's language into our own. In a like manner, alphabetic data must be converted to a binary numerical code for use in the computer. In this chapter, only numerical data will be considered.

Our computer has two translating devices, the *encoder* and the *decoder*. The encoder enables us to translate our problem data from decimal numbers to binary numbers. The decoder enables us to understand the answers the computer obtains by translating its binary response into decimal numbers. To understand how these units function, it is necessary to examine the relationship between the decimal and binary number systems.

NUMBER SYSTEMS

In the decimal number system, there are ten different numeral symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. When standing alone, each of these symbols represents a fixed number. In combination, however, these symbols can represent any number in the system. They do so by assuming different values when placed in different positions in a number. For example, in the number 4444, the numeral symbol is the same but it has four different values: 4000, 400, 40, and 4. In the decimal number 255,252, two numeral symbols are used, but each has more than one value (200,000, 200, and 2; 50,000, 5,000, and 50).

Such a system, where the value of the numerical symbol is determined by its position, uses *positional notation.* The significance of positional notation in the decimal number system is shown in Fig. 2-1. The first column to the left of the decimal point is the 10^0 or unit's column. Any numeral that appears in this position is weighted by the factor 1. The numeral 6 in this column is the equivalent of 6×1, or 6×10^0. In the second column to the left of the decimal point, the 10^1 column, a numeral is weighted by a factor of 10. Here, 6 would

HUNDREDS COLUMN (10^2)	TENS COLUMN (10^1)	UNITS COLUMN (10^0)	DECIMAL POINT	TENTHS COLUMN (10^{-1})	HUNDRETHS COLUMN (10^{-2})	
0	0	6	.	0	0	= 6
0	6	0	.	0	0	= 6 x 10 = 60
6	0	0	.	0	0	= 60 x 10 = 600

Fig. 2-1. Positional notation in the decimal number system.

attain a value of 6×10, or 6×10^1. In the third column, the 10^2 column, 6 would attain a value of 6×100, or 6×10^2.

Each of the numeral symbols in the decimal system will change its value in the same manner as its position changes in a number. Each shift to the left increases a numeral's value by a factor of 10. Each shift to the right decreases a numeral's value by the same factor. To illustrate this further, we will break down several decimal numbers:

$$\begin{aligned} 357.1 &= 300 + 50 + 7 + 0.1 \\ &= 3 \times 100 + 5 \times 10 + 7 \times 1 + 1 \times 1/10 \\ &= 3 \times 10^2 + 5 \times 10^1 + 7 \times 10^0 + 1 \times 10^{-1} \end{aligned}$$

$$\begin{aligned} 13402.01 &= 10{,}000 + 3000 + 400 + 2 + .01 \\ &= 1 \times 10{,}000 + 3 \times 1000 + 4 \times 100 + 2 \times 1 \\ &\quad + 1 \times 1/100 \\ &= 1 \times 10^4 + 3 \times 10^3 + 4 \times 10^2 + 0 \times 10^1 \\ &\quad + 2 \times 10^0 + 0 \times 10^{-1} + 1 \times 10^{-2} \end{aligned}$$

Many practical number systems use positional notation. They differ from the decimal system only in the number of numeral symbols they use and the weighting factor related to each position. The *octal* system, for example, uses 8 symbols: 0, 1, 2, 3, 4, 5, 6, and 7. Numeral values in this system change by a factor of 8 as the numeral's position changes. For example, the octal number 136 is the equivalent of the decimal number 94:

$$\begin{aligned} 136 &= 1 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 \\ &= 1 \times 64 + 3 \times 8 + 6 \times 1 = 94 \end{aligned}$$

Because these systems and others use the same numeral symbols, it is difficult at times to tell what system a number is written in. If someone were to tell you that 136 really represented 94 you wouldn't believe them unless there was some indication that 136 was an octal number and 94 was a decimal number. For this reason, when numbers in different systems are compared in this text they will be identified by a key number. This key number indicates the number of numeral symbols

used in the system and is referred to as the system's *base* or *radix*. The base for the decimal system is 10; the base for the octal system is 8. Using this coding, for example, $(100)_{10}$ and $(100)_8$ represent, respectively:

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

and:

$$1 \times 8^2 + 0 \times 8^1 + 0 \times 8^0$$

Also, $(100)_8 = (64)_{10}$.

Any number in any positional notation system may be represented by the following formula:

$$N = S_nB^n + \ldots + S_3B^3 + S_2B^2 + S_1B^1 + S_0B^0 + S_{-1}B^{-1} + S_{-2}B^{-2} + \ldots + S_{-m}B^{-m}$$

where N is the number, S_n through S_{-m} are the numeral symbols used in the system, and B is the base of the system. The base, raised to a power, represents the factor by which the numeral symbol is multiplied in each position. In these number systems, the least significant numeral is the rightmost digit and the most significant numeral is the leftmost digit.

THE BINARY NUMBER SYSTEM

Like the octal and decimal systems, the binary system uses positional notation. Two symbols, only, are used, 0 and 1, so the value of a numeral increases or decreases by a factor of 2 as it shifts to the left or right. The base of the binary system is 2. The significance of positional notation in the binary system is shown in Fig. 2-2. To illustrate this further, we will break down several binary numbers.

$$\begin{aligned}(1101)_2 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 \\ &= (13)_{10}\end{aligned}$$

$$\begin{aligned}(10011)_2 &= 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 1 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 \\ &= (19)_{10}\end{aligned}$$

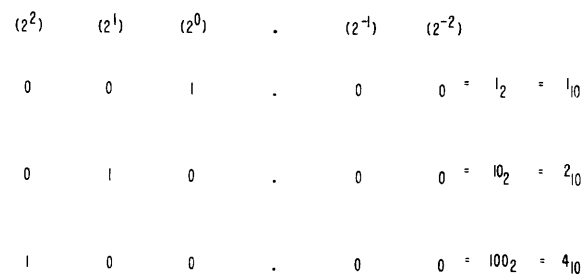


Fig. 2-2. Positional notation in the binary number system.

TABLE 2-1

BINARY	DECIMAL	BINARY	DECIMAL
1	1	10001	17
10	2	10010	18
11	3	10011	19
100	4	10100	20
101	5		
110	6	11110	30
111	7	110010	50
1000	8	1000110	70
1001	9	1011010	90
1010	10	1100100	100
1011	11		
1100	12	1111101	125
1101	13	10010110	150
1110	14		
1111	15	111110100	500
10000	16	1111101000	1000

Each numeral in a binary number is referred to as a *bit* (*bi*nary digi*t*). The binary number 100, which is $(4)_{10}$, is a three-bit number, and 10100, or $(20)_{10}$, is a five-bit number. Computers are rated by the number of bits that can be stored in their memory sections. The capacity for bit storage is directly related to the computer's precision and ability to handle complex calculations. Larger capacities enable the computer to work with greater precision and more complex calculations.

As indicated in Table 2-1, which lists some binary numbers and their decimal equivalents, binary numbers use more digits than decimal numbers to express the same quantities. However, the advantages this may give to decimal numbers are offset, in a computer, by the ease with which arithmetic is performed in the binary system. This will be covered in more detail in Chapter 4.

Because the decimal and binary number systems both use positional notation, conversion from one system to the other is relatively simple. It is done by translating the numeral values of one system into those of the other.

ENCODING

The process of conversion from a generally known system to a generally unknown system is called *encoding*. The unit in our computer

that performs this function accepts data in one form and transmits them in another. Its output is the properly-coded equivalent of its input.

The simple alphabet codes you may have used in childhood involved an encoding process. Converting such meaningful phrases as:

The ball was thrown from first to third.

into:

Ftg nmzz ime ftdaiz rday rudef fa ftudp.

used a conversion system in which a generally known arrangement of letters was converted into a generally unknown arrangement of letters. A relationship was established between both letter systems so that any sentence could be converted and still be understandable when decoded. (In this case, the known letters, a, b, c, . . . , n, o, p, . . . , z are the equivalent of the coded letters m, n, o, . . . , z, a, b, . . . , l, respectively.)

Encoding in a computer follows the same kind of fixed relationship. Each known decimal number corresponds to a "coded" or binary number.

DECIMAL TO BINARY CONVERSION

Decimal numbers can be converted to binary numbers by using the general number equation for the binary number system:*

$$(N)_2 = S_n 2^n + \ldots + S_3 2^3 + S_2 2^2 + S_1 2^1 + S_0 2^0$$

To do so, you would set N in the equation equal to the decimal number and solve for all the terms. If you wish to work this out for any particular decimal number, do so, but you will find that the process is quite lengthy and time-consuming.

There is a short-hand method for decimal to binary conversion that simplifies the exercise considerably. In it, the decimal number to be converted is repeatedly divided by 2. The remainder of each division, which can only be 1 or 0, will indicate a digit of the binary number. The remainder of the first division is placed in the 2^0 column; the remainder of the second is in the 2^1 column; etc. For example, to convert $(128)_{10}$ into its binary equivalent, the following steps are followed:

Division	*Remainder*	*Position*
$2\overline{)128}$	0	2^0
$2\overline{)64}$	0	2^1
$2\overline{)32}$	0	2^2
$2\overline{)16}$	0	2^3

* Fractional portions are not included for simplicity.

Division	*Remainder*	*Position*
$2\overline{)8}$	0	2^4
$2\overline{)4}$	0	2^5
$2\overline{)2}$	0	2^6
$2\overline{)1}$	1	2^7

Therefore, $(128)_{10} = (10000000)_2$.

Let us try this with a different number. For instance, $(61)_{10}$:

Division	*Remainder*	*Position*
$2\overline{)61}$	1	2^0
$2\overline{)30}$	0	2^1
$2\overline{)15}$	1	2^2
$2\overline{)7}$	1	2^3
$2\overline{)3}$	1	2^4
$2\overline{)1}$	1	2^5

Therefore, $(61)_{10} = (111101)_2$.

THE ENCODER

The encoder for our computer will convert the decimal numbers 0 through 9 into their binary equivalents. The construction details for this unit are given at the end of this chapter. At this point we will consider its design aspects.

The encoder unit receives a decimal input and produces an equivalent binary output. Switches will be used to represent and manipulate the decimal input. To monitor its output and demonstrate its function, the encoder will have a display that indicates what binary numbers are being used. Lamps will be used to indicate this binary output. Since a lamp has two states—on and off—and a binary number has two symbols —0 and 1, we will indicate a 1 symbol by lighting the lamp and a 0 symbol by extinguishing it.

We know that the largest decimal number we will encode is 9 and that its binary equivalent is a 4-bit number, (1001). Four lamps will therefore be sufficient for the binary output, one for each bit position. The lamps will be labeled according to their column heading: 2^3, 2^2, 2^1, and 2^0.

The switch connections used to encode decimal numbers are determined by examination of a conversion chart, such as that shown in Table 2-2. This reveals that there is some pattern to the manner in which the four output lamps indicate 0's or 1's. In the 2^0 column (rightmost column), the 0's and 1's alternate. For an even number there is a 0, for an odd there is a 1. In the next column, the 2^1 column, the 0's and 1's run down the column by twos. In the 2^2 column the 0's and 1's run down the column by fours. In the 2^3 column, the first eight rows have 0's and it is only because we have limited our chart to ten

TABLE 2-2

DECIMAL NUMBER	2^3	2^2	2^1	2^0
0	(0×2^3)	$+ (0 \times 2^2)$	$+ (0 \times 2^1)$	$+ (0 \times 2^0)$
1	(0×2^3)	$+ (0 \times 2^2)$	$+ (0 \times 2^1)$	$+ (1 \times 2^0)$
2	(0×2^3)	$+ (0 \times 2^2)$	$+ (1 \times 2^1)$	$+ (0 \times 2^0)$
3	(0×2^3)	$+ (0 \times 2^2)$	$+ (1 \times 2^1)$	$+ (1 \times 2^0)$
4	(0×2^3)	$+ (1 \times 2^2)$	$+ (0 \times 2^1)$	$+ (0 \times 2^0)$
5	(0×2^3)	$+ (1 \times 2^2)$	$+ (0 \times 2^1)$	$+ (1 \times 2^0)$
6	(0×2^3)	$+ (1 \times 2^2)$	$+ (1 \times 2^1)$	$+ (0 \times 2^0)$
7	(0×2^3)	$+ (1 \times 2^2)$	$+ (1 \times 2^1)$	$+ (1 \times 2^0)$
8	(1×2^3)	$+ (0 \times 2^2)$	$+ (0 \times 2^1)$	$+ (0 \times 2^0)$
9	(1×2^3)	$+ (0 \times 2^2$	$+ (0 \times 2^1)$	$+ (1 \times 2^0)$

rows that the pattern is not apparent for this column. If we had extended the chart to sixteen rows, we would have eight rows of 1's.

A ten-position rotary switch is used in the encoder. This switch is similar to the channel selector on a TV set where each channel is a particular position on the switch. A rotary switch has two parts: the arm and the contacts. Usually, one end of the arm is connected to a single point on a common terminal and rotates about this point. As it rotates, it contacts connection points that are positioned in a circle around the common terminal. A schematic of a ten-position rotary switch is shown in Fig. 2-3. You can see that the arm serves to connect the contact terminals to the common terminal, one at a time.

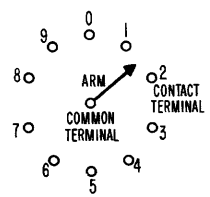


Fig. 2-3. A ten-position rotary switch.

From the conversion chart we know that the 2^0 lamp will light for every odd decimal number. Therefore, electricity is routed from the power source to the lamp when the switch is in a position to encode an odd number. This wiring is shown in Fig. 2-4. At positions 1, 3, 5, 7, and 9, the lamp will light because B+ is connected from one of the contact terminals through the arms to the common terminal and the

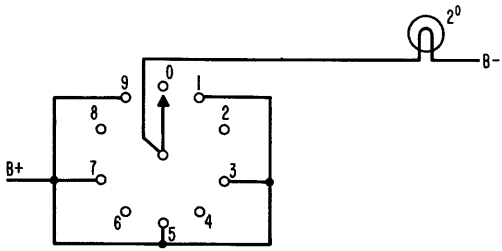
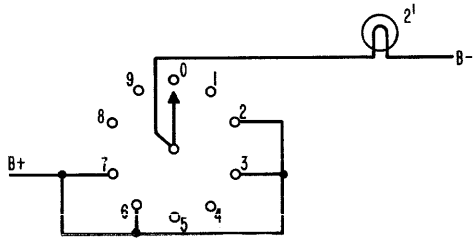


Fig. 2-4. A rotary switch wired to light 2^0 with every odd number.

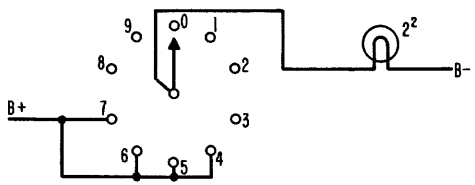
lamp. To form a complete circuit, the other side of the lamp is connected directly to the power source return.

The 2^1 lamp will light only when the decimal numbers 2, 3, 6, or 7 are encoded. The rotary switch connections for this lamp are shown in Fig. 2-5A.

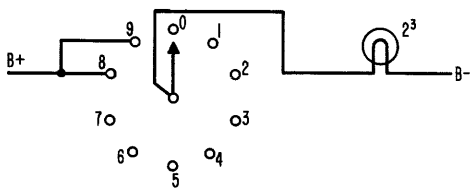
The 2^2 lamp will light only when the decimal numbers 4, 5, 6, or 7 are encoded. The 2^3 lamp lights only when the decimal numbers 8 or 9 are encoded. The rotary switch connections for these lamps are shown in Fig. 2-5B and C, respectively.



(A) ROTARY SWITCH SCHEMATIC, ENCODE 2^1



(B) ROTARY SWITCH SCHEMATIC, ENCODE 2^2



(C) ROTARY SWITCH SCHEMATIC, ENCODE 2^3

Fig. 2-5. The rotary switch wiring to lamp (A) 2^1; (B) 2^2; (C) 2^3.

To encode the decimal numbers 0 through 9, therefore, four ten-position rotary switches are needed. However, since this means that we must run each switch to the same position to encode a number, the construction is a bit cumbersome. Therefore, the switches are ganged; i.e., they are combined so that the arm is common to all four switches.

Our encoder will encode two decimal numbers simultaneously. Therefore, it will use two ganged rotary switches and two output displays. The wiring diagrams and a picture of the finished unit are included in the construction details at the end of this chapter.

BINARY TO DECIMAL CONVERSION

In the computer, decoding is necessary because the response is in binary form and must be converted to decimal form to be understood by the user. The unit that performs this function in our computer is the decoder, or output unit.

Arithmetically, converting binary numbers to decimal numbers is not too difficult. The process is one of simple addition:

$$\begin{aligned}(11011)_2 &= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 \\ &= 16 + 8 + 2 + 1 \\ &= (27)_{10}\end{aligned}$$

$$\begin{aligned}(10101)_2 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 \\ &= 16 + 4 + 1 \\ &= (21)_{10}\end{aligned}$$

If we limit our consideration to numbers under $(10)_{10}$, a simple decoder may be designed in a manner similar to that of the encoder. However, since our computer will be capable of producing numerical outputs exceeding $(10)_{10}$, but less than $(100)_{10}$, it will be necessary to decode a maximum of a 7-bit number into a 2-digit decimal number. One method of performing this conversion consists of converting the 7-bit binary number into two 4-bit binary numbers such that each 4-bit binary number represents a decimal digit. For example, the 7-bit number $(0101011)_2$ or $(43)_{10}$, may be converted to the following 4-bit numbers.

0100 and 0011 (or $(4)_{10}$ and $(3)_{10}$)

This conversion is usually performed by a programmed sequence of operations and is discussed in Chapter 7. The 4-bit numbers each represent a decimal digit and are called *binary-coded decimal numbers.* These numbers may readily be converted to decimal numbers by a decoder.

TABLE 2-3

BINARY SYMBOLS				DECIMAL SYMBOLS
2^3	2^2	2^1	2^0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

THE DECODER

The decoder for our computer will convert the binary numbers 0000 through 1001 to their decimal equivalents. As in the encoder, switches will be used to manipulate and represent the unit's binary input and lamps will be used to display the decimal output. Table 2-3 indicates the binary-to-decimal conversions possible in our computer.

Because our computer output will consist of a maximum of two binary-coded decimal digits it will be necessary to have two conversion units and two output displays; one for each four-bit binary output. The two units are composed of four double-throw multipole switches. Each switch represents a digit in a four-bit binary number. When the switch is thrown to the left it represents a 0, when it is thrown to the right it represents a 1. The switches are connected to ten lamps representing the ten decimal numbers 0 through 9. The binary numbers "written" by the switches will light the bulb representing their equivalent decimal number.

The wiring diagrams and a picture of the finished decoder unit are included in the construction details at the end of this chapter.

CONSTRUCTION DETAILS—ENCODER

COMPONENTS: *Chassis, rotary switches, display circuit*

MATERIALS

Chassis:

2	5 × 8 × ⅛ in. composition boards
6	⅜ in. dowels, 2⅛ in. long (height of spool + ¼ in.)
12	½ in. wood screws (no. 4)

Rotary switches (2):

2	thread spools, empty (1½ in. diameter, 1⅞ in. height)
2	¼ in. dowels, 3⅛ in. long (height of spool + 1¼ in.)
2	1 in. wood screws (no. 4)
	5 ft. uninsulated hook-up wire (20 gage)
	Adhesive tape, 2 in. wide

Display circuit:

8	#48 or 41 lamps (2 v, .06 a)
16	paper clips (large)
2	7 × ½ in. tin strips (from can or sheet)
	5 ft. insulated hook-up wire (20 gage)
	1 ft. uninsulated hook-up wire (20 gage)
4	½ in. machine screws (6-32)
2	1 in. machine screws (6-32)
12	nuts (6-32)

SPECIAL TOOLS:

Tin snips
Drill (1/16, 3/32, ⅛, ¼, ⅜)
Razor blade
Steel-edged ruler or straightedge
Round file

Chassis Construction

One of the 5 × 8 in. boards will hold the lamp display and switch controls. The other will hold the two rotary switches in position underneath the top panel. The dimensions for this unit are controlled by the size of the spools used for the rotary switches. If the recommended spool (1½ in. diameter, 1⅞ in. height) is not available, the placement and length of the supporting dowels will have to be altered to accommodate the larger or smaller spool size.

1. Select your top panel and mark and start the drill holes indicated in Fig. A-1.
2. Drill the ⅜-in. and ⅛-in. holes in the top panel. (The ⅜-in. holes will hold the lamps for the encoder display. The ⅛-in. holes will hold machine screws for circuit and terminal connections.)
3. Clamp the top panel over the second 5 × 8-in. board, lining up the sides, and drill the 3/32-in. and ¼-in. holes, making sure that the drill passes

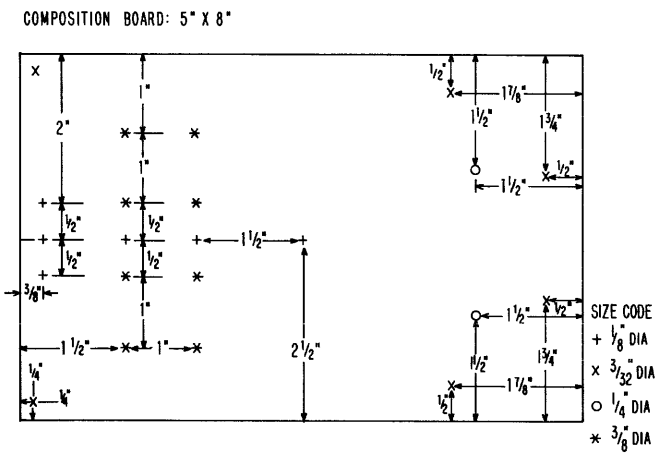


Fig. A-1.

cleanly through both boards. (The 3/32-in. holes will hold the wood screws securing the dowel supports. The ¼-in. holes will secure the axes of the rotary switches.)

4. With a round file, open up the ¼-in. holes in both boards so that a ¼-in. dowel will turn freely in them. Smooth off all rough edges around drilled holes.

Rotary Switch Construction

The construction details for both switches are the same.

1. Notch the spools as shown in Fig. A-2(a).
2. Drill a 1/16-in. hole through the diameter of the spool at the head and base, starting at the foot of each notch (Fig. A-2(a)).
3. On the head of the spool, drill a 1/16-in. hole opposite the notch into the head so that the drill comes out close to the hole drilled through the spool's diameter (Fig. A-2(b and c)). Drill a second hole down into the head ¼ in. away.
4. Cut a 3⅛-in. length of ¼-in. doweling and insert it through the spool's center hole. Allow at least ½ in. to protrude from the base (Fig. A-2(d)). (If the dowel does not fit snugly in the spool, wrap tin foil around it for a tight fit. It may help to file the inside of the spool's center hole to remove any burrs created by the drill. It is important that the dowel does not move independently of the spool. If desired, secure it with wood cement or wedges.)
5. When the dowel is secured properly, drill a 1/16-in. hole through the doweling at the bottom of the spool, using the hole drilled through the base of the spool in Step 2 as a guide.
6. Insert one end of the uninsulated wire into the hole opposite the notch on the spool's bottom and push it through so that it protrudes slightly from the other side.
7. Without cutting the wire, wind it around the spool body, making sure there are no separations between windings. After every three turns, pull the wire taut to make the windings as tight as possible. See

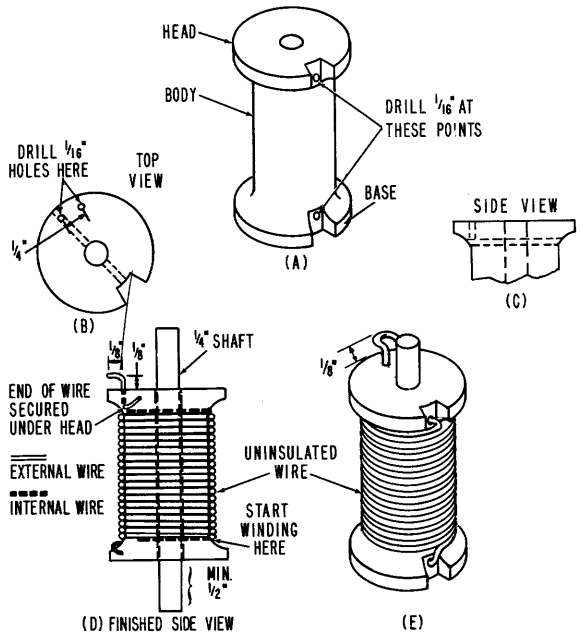


Fig. A-2.

Fig. A-2(d). Before the spool body is completely covered by uninsulated wire, drill a 1/16-in. hole through the doweling at the top of the spool, using the hole drilled in Step 2 through the head of the spool as a guide.

8. When you have covered the spool body to the hole drilled in Step 7, cut the wire, leaving approximately 3 in. to the end.
9. Insert the end of the wire into the top diameter hole at the notched side and push it through. Pull the wire so that all the windings are taut against the spool body. Then, insert the end of the wire into the 1/16-in. hole under the head of the spool and push it through so that it comes out on top of the spool.
10. Insert the end of the wire into the second hole in the spool's head and push it through until a ¼-in. loop is left on the top of the spool. See Fig. A-2(d). Secure the end of the wire by bending it under the head of the spool. Cut off any excess wire protruding beyond the head. Trim off the wire protruding at the bottom of the spool.
11. Bend the ¼-in. loop outward as in Fig. A-2(d).
12. Construct a second rotary switch.

Chassis Supports

1. Cut 6 2⅛-in. lengths from the ⅜-in. doweling. (If your spools are not the recommended size, these supports should be ¼ in. longer than the height of the spool.)
2. Start a drill hole in the centers of the ends of each support. Drill approximately ¼ in. into each end with the 3/32-in. drill.

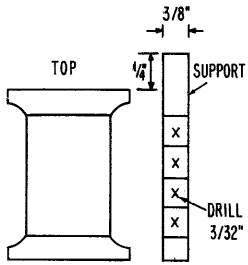


Fig. A-3.

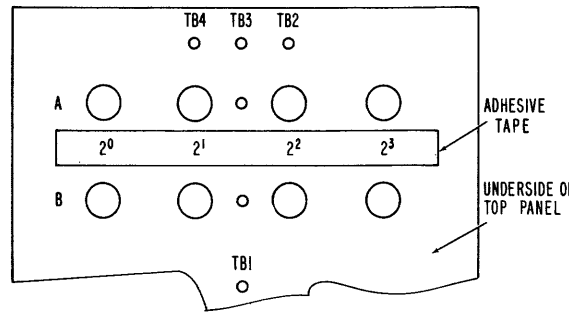


Fig. A-4.

3. Mark off four of these supports as shown in Fig. A-3.
4. Drill through the diameter of these four supports, where marked, with a 3/32-in. drill.
5. Secure the six supports to the bottom of the chassis with ½-in. wood screws. The four supports drilled in Step 4 should be placed next to the ¼-in. holes. The spacer should be at the top.

Display Lamp Construction

1. The eight display lamps will be placed in the ⅜-in. holes in the top panel so that their glass envelopes protrude approximately ⅛ in. to ¼ in. above the board. Test each hole with a bulb, and file it as needed to fit the bulb properly.
2. Label the holes on the underside of the panel as indicated in Fig. A-4. This can be done by placing strips of adhesive tape on the board and printing the proper labels on the tape.
3. For each bulb, construct a base terminal connector as follows. Straighten out a paper clip as shown in Fig. A-5(a) and cut it as indicated. Secure the paper clip around the screw base of the lamp as in Fig. A-5(b) and crimp it as in Fig. A-5(c) to hold the bulb firmly. With your long-nose pliers, turn down the cut ends of the paper clip as shown in Fig. A-5(d). When connecting a lead to the bulb, place the stripped end of the lead in the angle formed by the clip ends and bend them until the wire is pinched between them (Fig. A-5(e)). This type of base connection will be used in all lamp displays on all units of the computer.

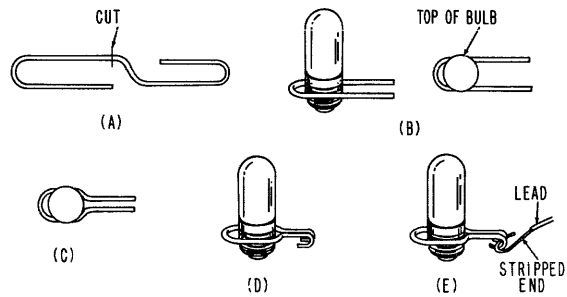


Fig. A-5.

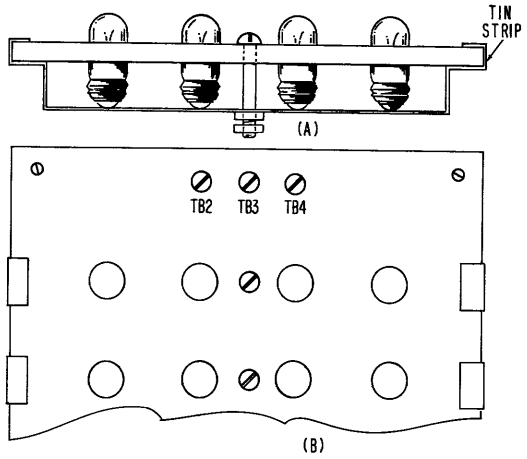


Fig. A-6.

4. Place the eight lamps, with the base terminal connectors attached, in the ⅜-in. holes on the panel top so that their glass envelopes protrude approximately ⅛ in. to ¼ in. above the panel.
5. Position the lamps so that the base connectors point towards the ¼-in. holes at the foot of the panel.
6. Cut two 7-in. strips of tin, ½ in. wide. Buff both sides of each strip.
7. Shape each strip as indicated in Fig. A-6(a). When attached to the top panel, these strips should contact only the terminal on the bottom of the lamp. They should not touch the threaded section of the base or the base terminal connector.
8. Fit each tin strip onto the board, running them beneath the lamps, as shown in Fig. A-6(a).
9. Drill through the center of each strip between the 2^1 and 2^2 lamps, using the ⅛-in. holes on the panel as a guide.
10. Insert 1-in. machine screws through these holes in the top panel and strips and secure the strips to the lamp bottom terminal with two nuts (Fig. A-6(a)).
11. Screw ½-in. machine screws into the 3 ⅛-in. holes at the top of the panel (TB2, TB3, and TB4). Secure these screws with nuts beneath the panel. Connect insulated leads from TB3 to the 1-in. screw securing the A strip and from TB4 to the 1-in. screw securing the B strip.
12. Cut 8 20-in. lengths of insulated wire. Strip ¼ in. of insulation from each end of each wire.
13. Label the wires by attaching small pieces of adhesive tape to each end of each wire. Mark the labels with the lamp designations given in Fig. A-4 ($A2^0$, $B2^0$, etc.).
14. Attach the wires to the paper-clip lamp connectors. Lead the wires from the A strip to the left and the wires from the B strip to the right.

Tape Preparation

1. Wrap a blank piece of paper around the body of one of the wired spools. Mark off the height of the body and its circumference.

2. Using the paper with the body dimensions marked on it, divide the height into four equal levels and the circumference into ten equal levels. See Fig. A-7(a).

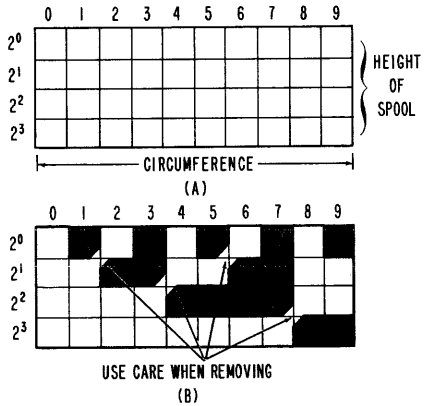


Fig. A-7.

3. Mark the ruled paper in the fashion shown in Fig. A-7(b). This is the template pattern for the adhesive-tape insulation that will cover the uninsulated wires wrapped around the spool.
4. Place a length of 2-in. wide adhesive tape that is slightly longer than the circumference of the spool on a metal surface, sticky side down, and cover it with a second layer of 2-in. tape.
5. Lay the paper template on top of the tape layers and cut out the template and tape with a sharp razor and steel-edged ruler along the heavy lines indicated in Fig. A-7(b).
6. Peel both layers of tape from the metal surface as one unit. Be careful that the tape does not tear at the narrow points.
7. Repeat these steps to make a double-layered covering for the second rotary switch.

Placing Tape on Spool Body

Care must be taken when selecting where the tape is to be placed on the spool body. This depends on which spool is to be used for which input (A or B). Input A will be on the left side of the chassis. The design for A is such that 0 is at 12 o'clock and the numbers run clockwise to 9. The switch contact point, however, is between 7 and 8 o'clock.

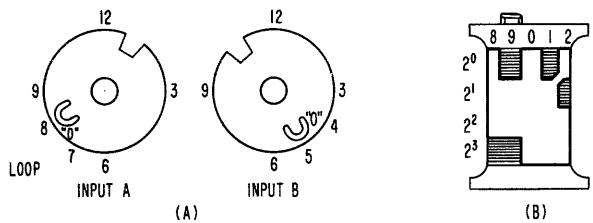


Fig. A-8.

1. Insert the spool shafts in the ¼-in. holes drilled in the bottom board so that the spool bottom is flush with the board. Place the top panel so that the top shafts of the spools run through the ¼-in. holes and the panel rests on the dowel supports.
2. Rotate the spool to be used as the A input (on the left) so that the wire loop on the top is positioned at about 8 o'clock. Rotate spool B so that the loop is positioned at 5 o'clock. See Fig. A-8(a).
3. On spool A, mark the area between 7 and 8 o'clock "0." On spool B, mark the area between 4 and 5 o'clock "0."
4. Remove both spools and attach the tapes to them. Position the 0 part of the tape where you marked 0. The 2^0 row, which alternates tape and no tape, is to be at the top of each spool. See Fig. A-8(b).

Switch Connections

1. Place the rotary switches on the bottom board. File off any spool shaft that protrudes below the board.
2. Position the panel top over the switches, with shafts in the proper place, and fasten the top to the chassis supports with ½-in wood screws.
3. Straighten 8 large paper clips and cut each one so that it is approximately 3 in. long.
4. Run the paper clips through the holes drilled in the dowels next to the switches. Position the dowels so that the clips appear as in Fig. A-9.
5. Bend the clips for contact on the switch surface as shown in Fig. A-9. Each clip should make contact with the switch approximately in the center of a horizontal row on the patterned insulation. The contacts should be aligned vertically as well as horizontally.
6. Tighten and secure the switch contacts by binding the clip ends around the dowels.
7. Connect the wires from the lamp displays to the switches by attaching the stripped ends to the ends of the paper clips.
8. Cut two 6-in. insulated leads. Strip ¼ in. of the insulation from both ends of each and coil each wire.
9. Attach one end of one wire to the loop on top of spool A. Attach one end of the other wire to the loop on top of spool B. (See Fig. A-10.)
10. Put a ½-in. machine screw through the drill hole for TB1 and attach the free ends of the coiled wire to it with a nut.
11. Run a 4-inch insulated lead between TB1 and TB2.

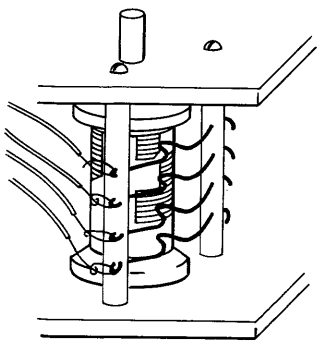


Fig. A-9.

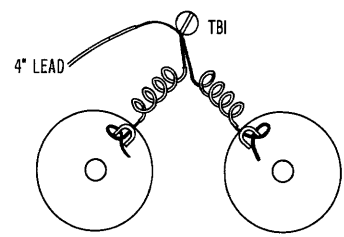


Fig. A-10.

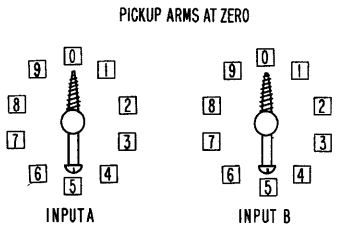


Fig. A-11.

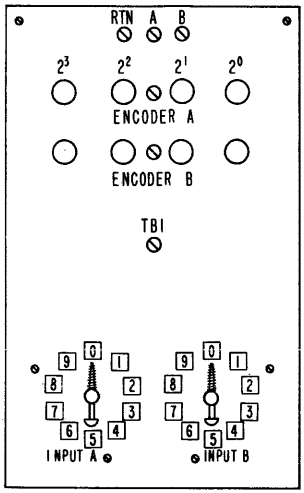


Fig. A-12.

Knobs and Labels

1. Position switches A and B so that the contacts rest on 0. With the switches held in this position, drill a 3/32-in. hole through each dowel protruding above the top panel. Drill each hole toward the top of the panel so that a 1-in. wood screw screwed into the hole will point to the top of the panel. See Fig. A-11.

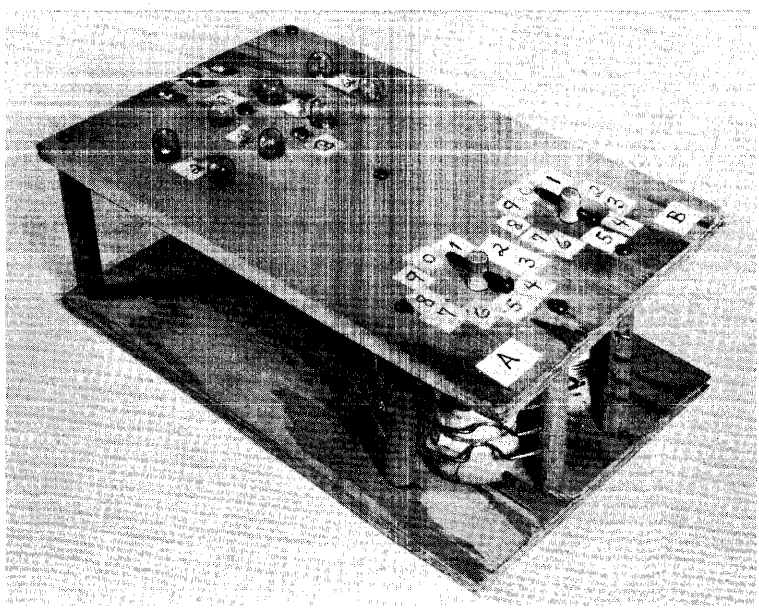


Fig. A-13.

2. Put a 1-in. wood screw through the holes drilled in Step 1.
3. Label the top panel as shown in Fig. A-12.
4. To prevent the switch pick-up from fouling, put mechanical stops between 0 and 9 of each switch. These can be 1-in. brads nailed into the top of the chassis. Do not drive the nails in too far.

Encoder Checkout Procedure

1. Connect a 1½-volt flashlight battery between the RTN and A terminals.
2. Set switch A to each position and check lamps to see that they light as indicated below (0 = off, 1 = on).

Switch Position	*Lamps*			
	2^3	2^2	2^1	2^0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

3. Repeat Steps 1 and 2 for switch B, connecting the battery between the RTN and B terminals.

CONSTRUCTION DETAILS—DECODER

COMPONENTS: *Switch chassis, switches, display chassis, display circuit*

MATERIALS

Switch chassis and switches:

1	15 × 15 × ⅛ in. composition board
1	15 × 4 × ¼ in. composition board
4	5 × ½ × ⅛ in. composition board
4	3 × ½ × ⅛ in. composition board
	20 ft. insulated hook-up wire (20 gage)
12	DPDT switches
2	SPDT switches
32	½-in. wood screws (no. 4)
2	½-in. machine screws (6-32)
8	¾-in. machine screws (6-32)
12	nuts (6-32)

Display chassis and circuit:

1	4¾ × 10 × ⅛ in. composition board
1	3 × 10 × ¼ in. composition board
20	#48 or 41 lamps (2 v, .06 a)
20	paper clips

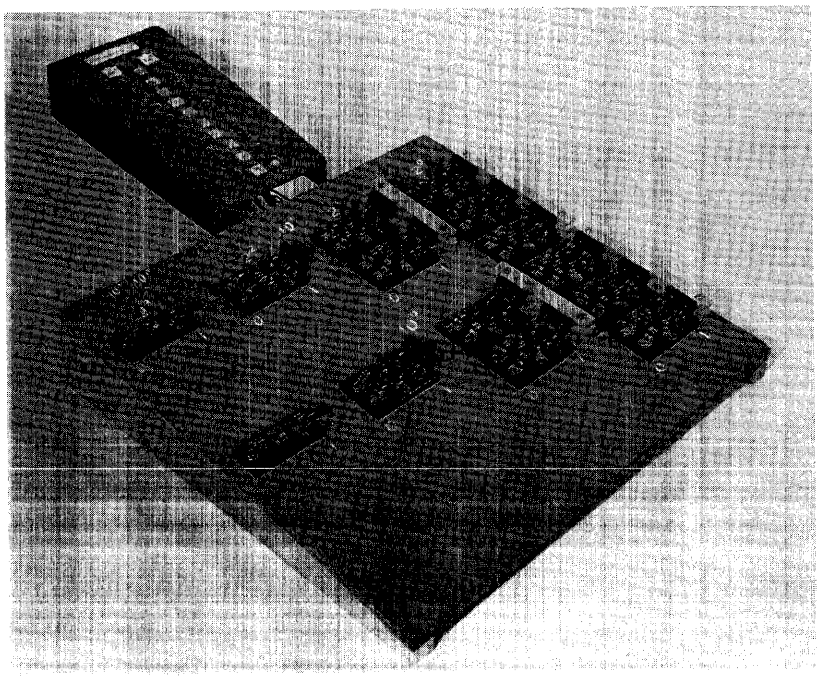


Fig. B-1.

4	½ in. machine screws (6-32)
4	1 in. machine screws (6-32)
10	nuts (6-32)
6	½ in. wood screws (no. 4)
50	1 in. wire brads (approx. count)
2	9½ × ¾ in. tin strips
	25 ft. insulated hook-up wire (20 gage)
	2 in. uninsulated hook-up wire (20 gage)

SPECIAL TOOLS:

Tin snips
Drill (1/16, 3/32, ⅛, ⅜)
Pencil and paper
Ruler or straightedge

Switch Chassis Construction

This panel will hold an array of single-pole, double-throw (SPDT) and double-pole, double-throw (DPDT) switches. The unit is designed for knife switches mounted on a bakelite base. Manufactured switches are used to insure switch reliability and durability, which are difficult to maintain with

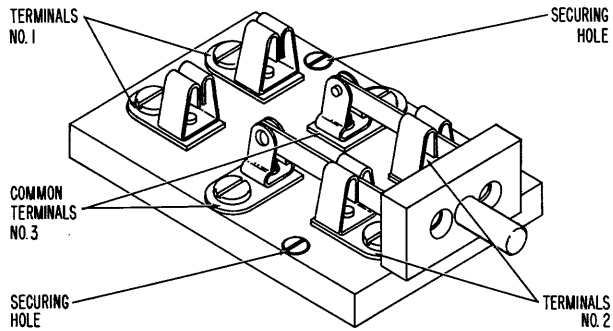


Fig. B-2.

simple, home-made switches. The double-pole, double-throw model of this switch is shown in Fig. B-2.*

1. The switches are arranged on the panel as shown in Fig. B-3. Mark the 15 × 15 × ⅛-in. panel on its top (smooth side) as shown in Fig. B-3. Then, using the figure as a guide, position a DPDT or SPDT switch where indicated and mark the position of the switch-securing holes with a sharp pencil.

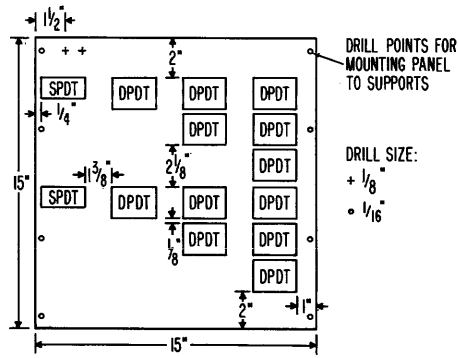


Fig. B-3.

2. Start a drill hole at each pencil mark, then drill through the panel with a 3/32-in. drill at each point.
3. Start drill holes at all other points indicated in Fig. B-3 and drill through the panel with the designated drill.
4. Mark the 15 × 4 × ¼-in. board as in Fig. B-4 and cut along the diagonal line. These angle supports will support the switch panel.
5. Placing the wide ends of the supports at the top of the panel, drill 1/16-in. holes along the top edge of each support, using the holes drilled at the panel side edges in Step 3 as guides.
6. Secure the angle supports to the panel with ½-in. wood screws.
7. Attach all switches, using ½-in. wood screws.

* If you wish to build your switches, a workable switch design is detailed in the Appendix. Of course, when you substitute home-made switches for manufactured, the units will have to be redesigned.

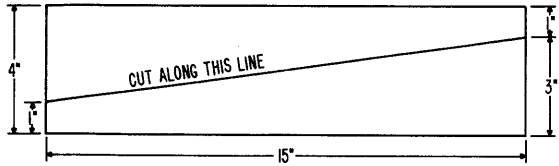


Fig. B-4.

8. Mark for drill holes alongside each switch terminal of each switch. Drill these holes through the panel with the ⅛-in. bit.
9. As different sets of switches are to be thrown simultaneously, their handles must be joined. Connect the switch indicated in Fig. B-5(a) in the manner detailed in Fig. B-5(b), using the ½-in. strips of composition board and ¾-in. machine screws and nuts.
10. Using ½-in. machine screws, insert in two ⅛-in. holes at upper left side of panel. Secure with two nuts.

Panel Wiring

1. Turn the switch panel over and label the underside as shown in Fig. B-6. These labels will guide you when wiring the switches together.
2. The wiring list for the panel is shown in Chart A. The leads designated "harness" will be made and connected when the display panel is con-

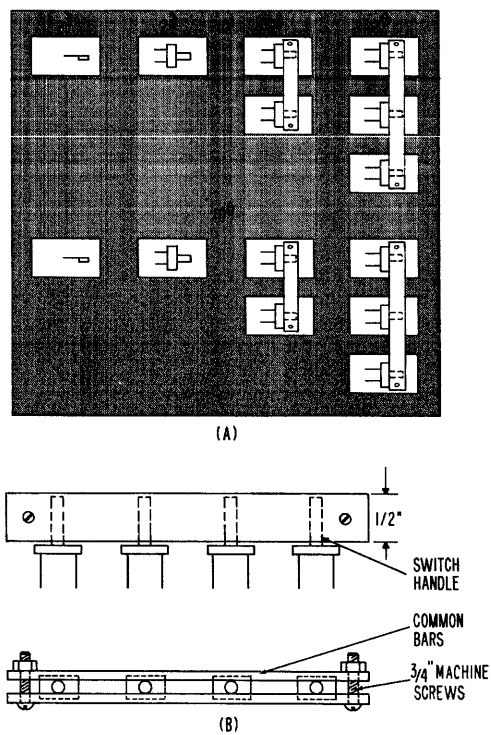


Fig. B-5.

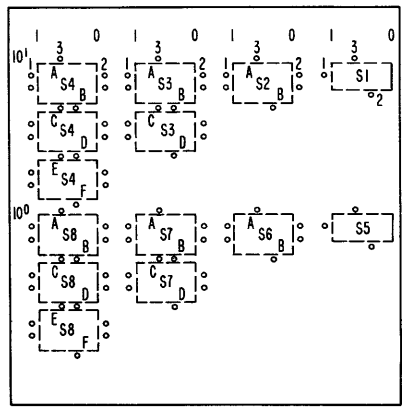


Fig. B-6.

nected to the switch panel. Following Chart A, use 20 gage insulated hook-up wire to connect the designated terminals. Do this by running uncut wire from terminal to terminal to the determine the needed length; cutting the wire ¾ in. longer than this length; stripping ¼ in. insulation from each end of the cut wire; and connecting wire ends to the proper terminal. As some terminals have more than one connection, do not secure wires until all connections have been made.

CHART A

10^1 Switches		*10^0 Switches*	
From	To	From	To
TB1	S1-3	TB1	S5-3
S1-2	S2-A3	S5-2	S6-A3
S2-A2	S3-A3	S6-A2	S7-A3
S3-A2	S4-A3	S7-A2	S8-A3
S4-A2	Harness (0)	S8-A2	Harness (0)
S1-1	S2-B3	S5-1	S6-B3
S2-B2	S3-C3	S6-B2	S7-C3
S3-C2	S4-E3	S7-C2	S8-E3
S4-E2	Harness (8)	S8-E2	Harness (8)
S2-A1	S3-B3	S6-A1	S7-B3
S3-B2	S4-C3	S7-B2	S8-C3
S4-C2	Harness (4)	S8-C2	Harness (4)
S3-A1	S4-B3	S7-A1	S8-B3
S4-B2	Harness (2)	S8-B2	Harness (2)
S3-B1	S4-D3	S7-B1	S8-D3
S4-D2	Harness (6)	S8-D2	Harness (6)
S4-A1	Harness (1)	S8-A1	Harness (1)
S4-B1	Harness (3)	S8-B1	Harness (3)
S4-C1	Harness (5)	S8-C1	Harness (5)
S4-D1	Harness (7)	S8-D1	Harness (7)
S4-E1	Harness (9)	S8-E1	Harness (9)
TB2	Harness (B—)		

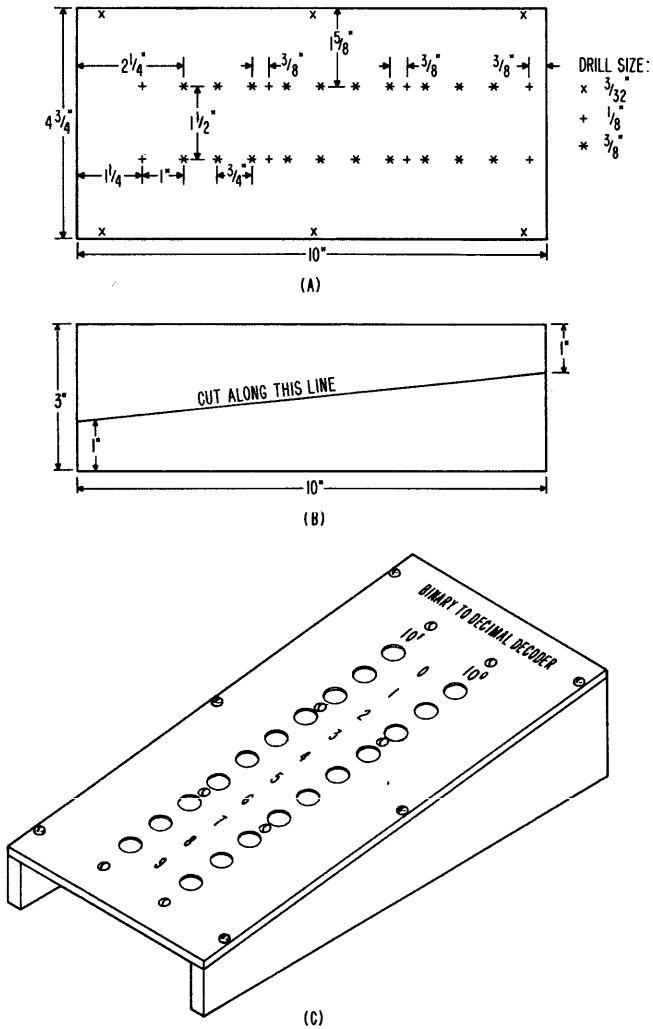


Fig. B-7.

Display Chassis Construction

1. Mark the 4¾ × 10 × ⅛-in. board as indicated in Fig. B-7(a) and start a drill hole at each indicated point. This is the top panel of the display chassis.
2. Drill holes through the chassis at each point with the indicated bit size.
3. File each ⅜-in. hole so that the bulb of a lamp will protrude approximately ⅛ in. above the panel when the lamp is fitted into the hole.
4. Mark the 3 × 10 × ¼-in. board as shown in Fig. B-7(b) and cut along the diagonal line to form two angle supports.
5. Placing the support pieces as shown in Fig. B-7(c), drill 6 3/32-in. holes into the supports, using the 3/32-in. holes in the top panel as a guide.

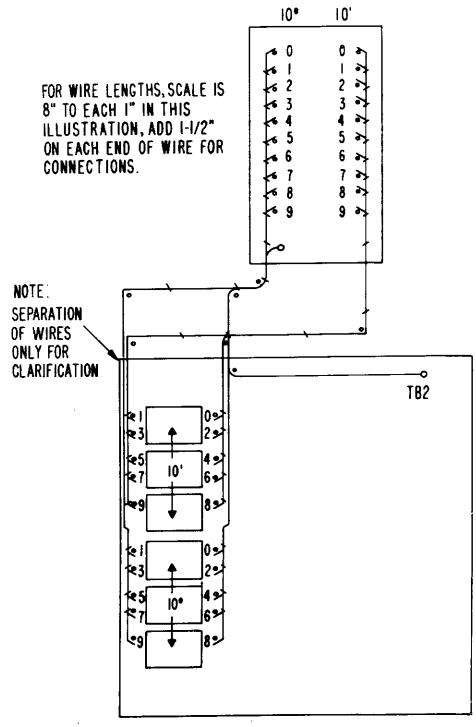


Fig. B-8.

6. Secure the top panel to the supports with ½-in. wood screws.

Wire Harness and Display Circuit Construction

1. To streamline the connections between the display lamps and switches, the wire runs will be ganged in harnesses. The harnesses are to be laid out on patterns and transferred to the unit when finished, as detailed below. They should be carefully constructed and tied securely at all indicated points to minimize adjustments when they are moved from pattern to chassis.

   Figure B-8 is a representation of the harness pattern, drawn to scale. Reproduce this pattern on a large sheet of paper at actual size, following the dimensions indicated in Fig. B-8.
2. Place the pattern on a wooden bench or work area into which nails may be driven. Drive a 1-in. brad into each point indicated by a dot on the pattern, so that the brad is secure and at least ¾ inches of it protrude from the pattern.
3. Both harnesses are identical except for length. Figure B-9 illustrates the shortest harness. Following Chart B, measure and cut the indicated lengths of 20 gage, insulated hook-up wire. Strip ¾ inch of insulation from each end of each wire and label each wire at each end with the lamp number (0-0, 0-1, 0-2, etc., for 10^0 lamps; 1-0, 1-1, 1-2, etc., for 10^1 lamps).

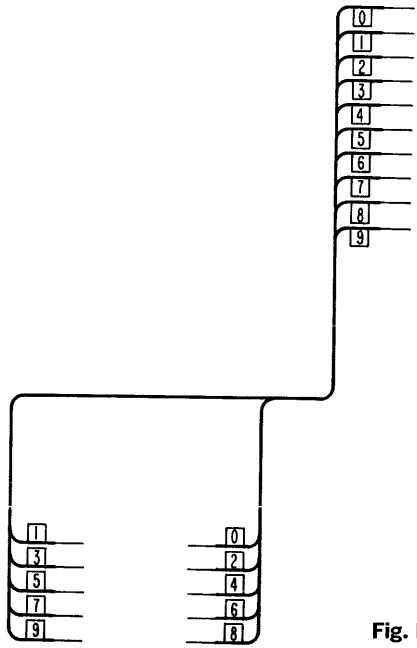


Fig. B-9.

4. When all lengths are cut and labeled, construct wire harnesses as follows:
   (a) Start each harness with the wire connecting the lamp furthest from the switch chassis (the 0 and 1 lamps) and complete each harness with the lamps nearest the switch chassis (the 8 and 9 lamps).
   (b) Place the proper wire in position, running it through the brads opposite the lamp hole, and bending it so that it will run through the brads at the bottom of the chassis. Run the wire as shown in the harness diagram until it is opposite its proper switch terminal. Place each wire for each harness on the pattern in this fashion. At the

CHART B

	10^0			10^1	
Wire	*Length*	*Connect To*	*Wire*	*Length*	*Connect To*
0–0	27	S8–A2	1–0	25	S4–A2
0–1	29	S8–A1	1–1	29	S4–A1
0–2	27	S8–B2	1–2	25	S4–B2
0–3	29	S8–B1	1–3	29	S4–B1
0–4	28	S8–C2	1–4	26	S4–C2
0–5	30	S8–C1	1–5	30	S4–C1
0–6	28	S8–D2	1–6	26	S4–D2
0–7	30	S8–D1	1–7	30	S4–D1
0–8	29	S8–E2	1–8	27	S4–E2
0–9	31	S8–E1	1–9	31	S4–E1
			B–	25	TB2

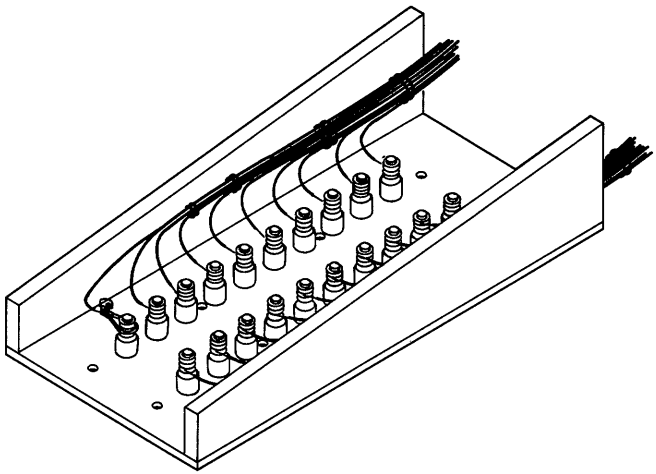


Fig. B-10.

point where each wire joins the others in the harness, secure it to the previously positioned wires with thin, short strips of adhesive tape. Also, tape the completed harness every few inches to hold the wires firmly in the position in which they were laid out.

(c) The finished harnesses should appear as in Fig. B-9. At the end where the wire contacts the lamp, bend the stripped portion at an angle of 90° so that it lies parallel to the main branch of the harness.

5. Construct 20 base terminal connectors with paper clips for the 20 lamps as was done for the lamps on the encoder panel. Test each bulb to see that it works.
6. Place each lamp in position on the top panel with the connectors pointed in the directions indicated in Fig. B-10.
7. Cut 2 9½-in. tin strips, each ¾ in. wide.
8. Secure these strips to the underside of the panel as shown in Fig. B-11.

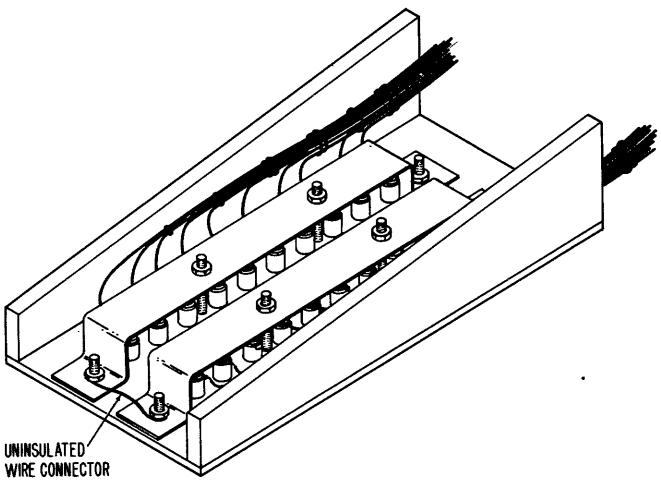


Fig. B-11.

Use the ⅛-in. holes in the panel as a guide for drilling holes through the strips. Secure each strip at both ends with nuts and ½-in. machine screws and at each interior point with a nut and 1-in. machine screw. Be certain that the strips are in contact with the bottom of each lamp and no other lamp part. Connect both strips with a short piece of uninsulated wire.

9. Lay the harnesses in position alongside the rows of lamps. Connect each wire to its proper bulb, securing the wire firmly by squeezing the terminal connector wires tightly together. Connect B– to tin strip screw.
10. When all bulbs are connected, place the harnesses in the proper positions on the underside of the switch panel and connect the wire ends to the proper terminals, following Chart B as a guide.
11. Label the switch panel as shown in Fig. B-5(a) and the lamp display panel as shown in Fig. B-7(c).

Decoder Checkout Procedure

1. Connect a 1½-volt flashlight battery between the RTN and ANS terminals.
2. Set all 10^1 and 10^0 switches to 0. Check that the 0 lamps for the 10^1 and 10^0 digits are lighted.
3. In order, set each switch (10^1 and 10^0) to each of the positions listed in the left-hand columns below and check to see that the indicated lamps light.

Switch Position				*Decimal Representation*
2^3	2^2	2^1	2^0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Chapter 3
COMPUTERS AND LOGIC

Although highly complex in appearance, the circuitry of the digital computer is composed of a handful of basic switching circuits that are repeated thousands of times. These circuits control the movement of electrical signals representing the bits (binary digits) used in computer operations. When designing computers or establishing a computer's problem-solving abilities it is necessary to determine the manner in which these circuits are to be interconnected. Also, it must be determined what circuit combinations are appropriate for each type of problem.

Such design and problem-solving relationships can easily be worked out by analyzing the circuits in terms of *symbolic logic*. These terms may not be familiar to you, but you use the theory behind them every day. Let us examine the basic terms of logic in light of our everyday experience and then prescribe some basic rules for their application.

First, let us consider a *statement*. From our studies of English we are all familiar with the definition of this term: *a statement is an assertion*. The previous sentence contains two statements separated by a colon. It is important to note that the fundamental property of any statement is that it is either true or false but never both true and false.

The next definition to consider is that of a *compound statement*. Again from our English studies, we know that a compound statement consists of two or more simple statements connected by conjunctions such as *and* and *or*. Compound statements also possess the property that they are either true or false. However, evaluating the truth of a compound statement is an exacting science and is the heart of the study of logic.

The study of compound statements can be divided into two facets:

1. In how many different ways can statements be compounded.
2. How do we determine the truth value of a compound knowing the truth value of its component simple statements?

In the study of logic, there are two basic conjunctions frequently used to compound statements: AND and OR. To illustrate the use of these conjunctions, let us make the following simple statements:

(a) It is raining.
(b) We are getting wet.

If we wish to make a compound statement asserting that both simple statements are true, we use AND.

It is raining AND we are getting wet.

This statement is only true if both simple statements are true. On the other hand, we may make a weaker assertion using the conjunction OR.

It is raining OR we are getting wet.

In this case, it is only necessary for a minimum of one of the simple statements to be true in order for the compound to be true. Hence, we can summarize the rules for evaluating compound statements as follows:

AND Rule

A compound joined by the conjunction AND is true only if all of its component simple statements are true. In all other cases, the compound is false.

OR Rule

A compound joined by the conjunction OR is true only if one or more of its component simple statements is true. It is false only if all of its component simple statements are false.

A third logical modifier is used to indicate a negative. This modifier is *NOT*. For example:

It is NOT raining.

The rules governing truth and falsity of statements using this modifier are obvious.

SYMBOLS

In order to work with and evaluate compound statements, it is convenient to establish a set of shorthand symbols to represent the various statements and conjunctions. Let us use lowercase letters to indicate statements, for example:

Let $a =$ It is raining.
Let $b =$ We are getting wet.

Also let us use the symbols $+$ and $\cdot$ to indicate OR and AND as follows:

Let OR $= +$
Let AND $= \cdot$

Finally, let us use a bar over the statement to indicate NOT as follows:

Let $\overline{a}$ = It is *not* raining.
Let $\overline{b}$ = We are *not* getting wet.

To further illustrate the symbols just established, consider the following list of symbolic statements and their English counterparts:

SYMBOLIC STATEMENT	ENGLISH EQUIVALENT
1. $a \cdot b$	1. It is raining and we are getting wet.
2. $a + b$	2. Either it is raining or we are getting wet (or both).
3. $(a \cdot b) + (\bar{a} \cdot \bar{b})$	3. Either it is raining and we are getting wet, or it is not raining and we are not getting wet.
4. $\bar{a} \cdot (b + \bar{b})$	4. It is not raining and either we are getting wet or we are not getting wet.
5. $(a + \bar{a}) \cdot \bar{b}$	5. Either it is raining or it is not raining, and we are not getting wet.
6. $\left[a \cdot (b + \bar{b})\right] + \left[b \cdot (a + \bar{a})\right]$	6. Either it is raining and either we are getting wet or we are not getting wet, or we are getting wet and either it is raining or it is not raining.

Examining these statements, especially 5 and 6, we can notice some redundancy. For example, in statement 6, look at the terms $(a + \overline{a})$ and $(b + \overline{b})$. Since there are only two possible conditions (either it's raining or it's not raining, and either we are getting wet or we are not getting wet), these sets of terms are meaningless. After all, the terms $(a + \overline{a})$ (either it is raining or it is not raining) must be true and goes without saying since there are no more possible conditions. This being the case, statement 6 in the list could just as meaningfully reduce to the statement $(a + b)$, either it is raining or we are getting wet. Reducing logic terms in this fashion is called *minimization* and plays a prominent role in the efficient design of digital computers. Minimization will be discussed later.

TRUTH TABLES

Sometimes it is convenient to draw up a table that lists the truth or falsity of a logic statement (simple or compound) for every possible condition or combination of conditions for that statement. As an example, let us consider the compound statement $(a \cdot b)$. There are four possible conditions for this statement since each simple statement

can assume one of two conditions, true or false. If we tabulate these conditions and then tabulate the truth value of the compound for each set of conditions we have:

a	b	a · b
T	T	T
T	F	F
F	T	F
F	F	F

where T indicates true and F indicates false. This table is called a *truth table* since it indicates the truth value of the compound for every possible variation of individual truth values of the component simple statements. An important theorem concerning truth tables is that *statements having identical truth tables are equal.*

A sample of a truth table for a higher-order compound statement is provided below:

a	b	c	d	$a \cdot b \cdot (c + \bar{d})$
T	T	T	T	T
T	T	T	F	T
T	T	F	T	F
T	F	T	T	F
F	T	T	T	F
T	T	F	F	T
T	F	F	T	F
F	F	T	T	F
T	F	T	F	F
F	T	F	T	F
F	T	T	F	F
T	F	F	F	F
F	T	F	F	F
F	F	T	F	F
F	F	F	T	F
F	F	F	F	F

LOGIC CIRCUITS

The question, now that the basic rules of logic have been defined, is "How can they be used?" We could, of course, use them as part of a simple mental exercise to test our powers of reasoning or we could build machines to solve problems for us logically. If we wish to build machines of this sort, however, we must learn to build devices that can evaluate the truth of compound statements. Devices of this sort are called *logic circuits* because they are electrical devices that simulate the basic rules of logic we have learned.

The first rule discussed was the AND rule, which stated that a compound statement is true only if all of its component simple statements are true. The compound $a \cdot b \cdot c$ (a AND b AND c) is true only if a is true and b is true and c is true. Therefore, we would like a device that provides a true indication only when all of the simple statements are individually true. The circuit illustrated in Fig. 3-1 is such a device.

In this circuit, the lamp lights only if switches A, B, and C are all closed. Switch A is closed when simple statement a is true; switch B is closed when simple statement b is true; and switch C is closed when simple statement c is true. Hence, the lamp lights only when the compound statement ($a \cdot b \cdot c$) is true.

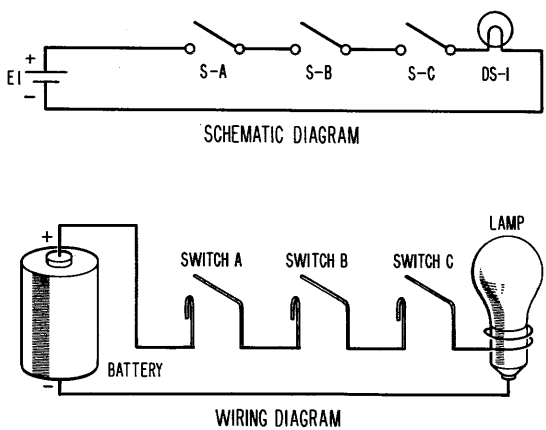


Fig. 3-1. An AND circuit.

The second rule discussed, the OR rule, asserted that a compound statement is true if one or more of its component simple statements is true. The compound $a + b + c$ (a OR b OR c) is true if either a is true or b is true or c is true, or any combination of the three simple statements is true. Therefore, we would like a device that provides a true indication only when one or more of the simple statements is individually true. The circuit illustrated in Fig. 3-2 is such a device.

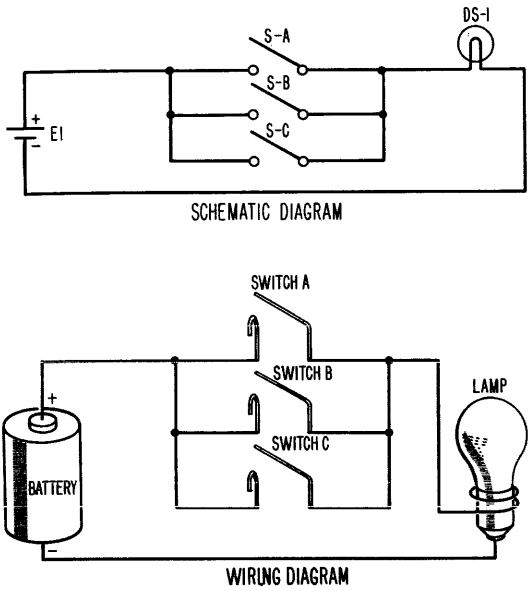


Fig. 3-2. An OR circuit.

In this circuit, the lamp lights only if any single switch or combination of switches A, B, or C is closed. As before, switch A is closed when simple statement a is true; switch B is closed when simple statement b is true; and switch C is closed when simple statement c is true. Hence the lamp lights only when the compound statement $(a + b + c)$ is true.

The third rule discussed was the NOT rule, in which the statement $\overline{a}$ (NOT a) is true only if the statement a is not true. Figure 3-3 illustrates a circuit that mechanizes the NOT rule. In this circuit, the switch is placed in the $\overline{a}$ position when the statement a is not true thereby

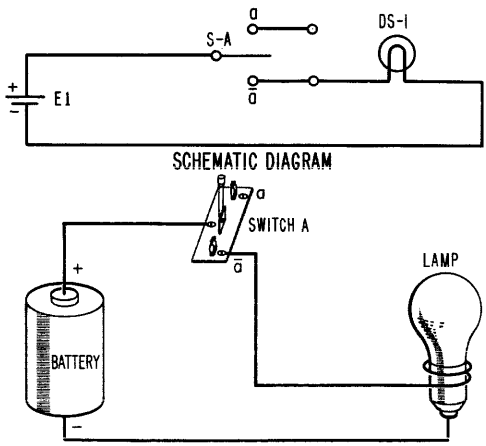


Fig. 3-3. A NOT circuit.

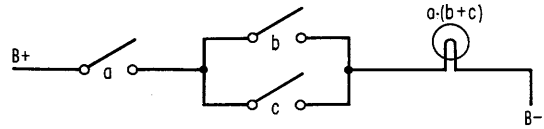


Fig. 3-4.

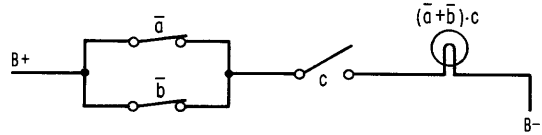


Fig. 3-5.

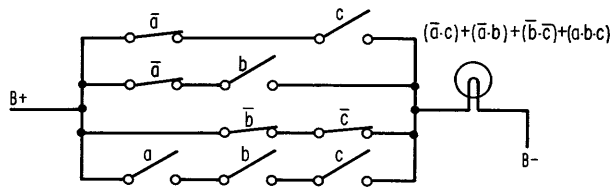


Fig. 3-6.

lighting the lamp. When the statement *a* is true, the switch is placed in the *a* position, causing the lamp to go out. In this case, the statement $\bar{a}$ is not true. It is important to note that in logic, as in English, a double negative results in a positive. Therefore, if the statement $\bar{a}$ is *not* true, the statement *a* is true. When using the NOT circuit in conjunction with the AND and OR circuits let us use the convention that the NOT switches will be shown normally closed, while all other switches will be shown normally open.

Now let us examine the use of logic circuits to instrument various compound statements. These circuits are shown in the indicated figures.

COMPOUND STATEMENT	FIGURE
$a \cdot (b + c)$	3-4
$(\bar{a} + \bar{b}) \cdot c$	3-5
$(\bar{a} \cdot c) + (\bar{a} \cdot b) + (\bar{b} \cdot \bar{c}) + (a \cdot b \cdot c)$	3-6

TRUTH EVALUATOR

Our study of truth tables and their significance in logic can be aided considerably by the use of a simple mechanism that will illustrate and evaluate the truth values of compound statements. Although this truth evaluator must be capable of simulating any combination of logical conjunctions, we will limit the number of simple statements to three. Refer to the construction details at the rear of this chapter and construct the truth evaluator. It will be used in the following experiments.

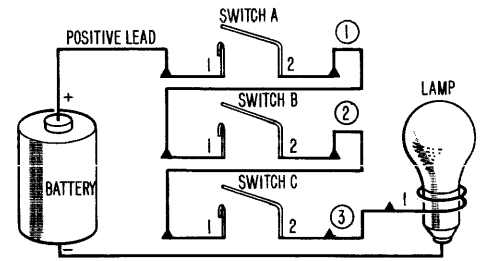
EXPERIMENTS WITH TRUTH TABLES

1. Set up the evaluator as illustrated in Fig. 3-7 for $a \cdot b \cdot c$ and check all eight conditions of this statement against Truth Table A.

2. Set up the evaluator as illustrated in Fig. 3-8 for $a + b + c$ and check all eight conditions of this statement against Truth Table B.

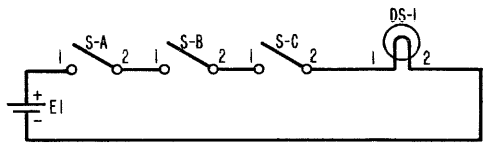
3. Set up the evaluator as indicated in Fig. 3-9 for $(a + b) \cdot c$ and check all eight conditions of this statement against Truth Table C.

4. Set up the evaluator as illustrated in Fig. 3-10 for $(a \cdot b) + \bar{c}$ and check all eight conditions of this statement against Truth Table D.



WIRE	FROM	TO
POSITIVE LEAD	BATTERY (+)	SWITCH A (1)
1	SWITCH A (2)	SWITCH B (1)
2	SWITCH B (2)	SWITCH C (1)
3	SWITCH C (2)	LAMP (1)

WIRING DIAGRAM



SCHEMATIC DIAGRAM

Fig. 3-7. The evaluator set-up for a · b · c.

TRUTH TABLE A. (a · b · c)

a	b	c	a · b · c
T	T	T	T
T	T	F	F
T	F	T	F
F	T	T	F
T	F	F	F
F	F	T	F
F	T	F	F
F	F	F	F

TRUTH TABLE B. $(a + b + c)$

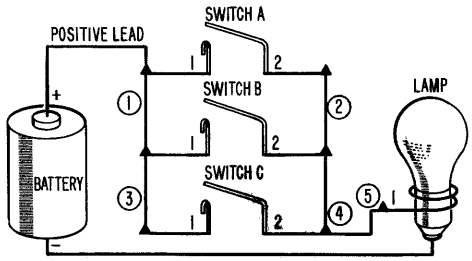
a	b	c	$a+b+c$
T	T	T	T
T	T	F	T
T	F	T	T
F	T	T	T
T	F	F	T
F	F	T	T
F	T	F	T
F	F	F	F

TRUTH TABLE C. $(a + b) \cdot c$

a	b	c	$(a+b) \cdot c$
T	T	T	T
T	T	F	F
T	F	T	T
F	T	T	T
T	F	F	F
F	F	T	F
F	T	F	F
F	F	F	F

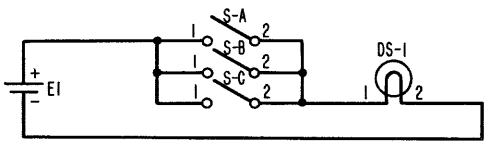
TRUTH TABLE D. $(a \cdot b) + \bar{c}$

a	b	c	$(a \cdot b) + \bar{c}$
T	T	T	T
T	T	F	T
T	F	T	F
F	T	T	F
T	F	F	T
F	F	T	F
F	T	F	T
F	F	F	T



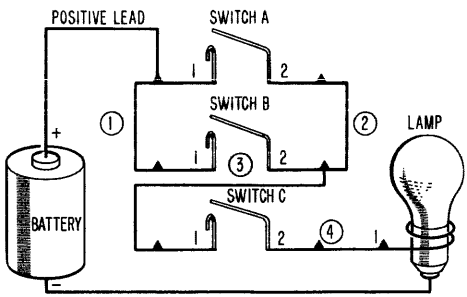
WIRE	FROM	TO
POSITIVE LEAD	BATTERY (+)	SWITCH A (1)
1	SWITCH A (1)	SWITCH B (1)
2	SWITCH A (2)	SWITCH B (2)
3	SWITCH B (1)	SWITCH C (1)
4	SWITCH B (2)	SWITCH C (2)
5	SWITCH C (2)	LAMP (1)

WIRING DIAGRAM



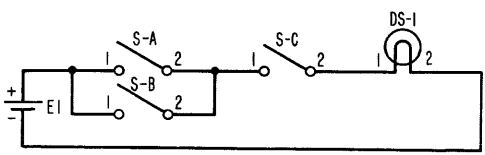
SCHEMATIC DIAGRAM

Fig. 3-8. The evaluator set-up for a + b + c.



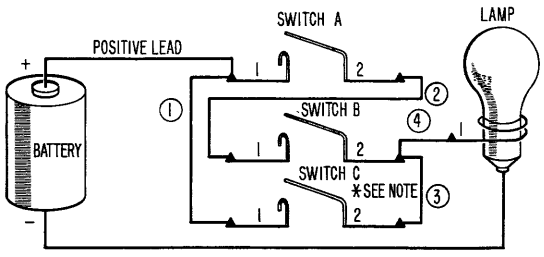
WIRE	FROM	TO
POSITIVE LEAD	BATTERY (+)	SWITCH A (1)
1	SWITCH A (1)	SWITCH B (1)
2	SWITCH A (2)	SWITCH B (2)
3	SWITCH B (2)	SWITCH C (1)
4	SWITCH C (2)	LAMP (1)

WIRING DIAGRAM



SCHEMATIC DIAGRAM

Fig. 3-9. The evaluator set-up for (a + b) · c.



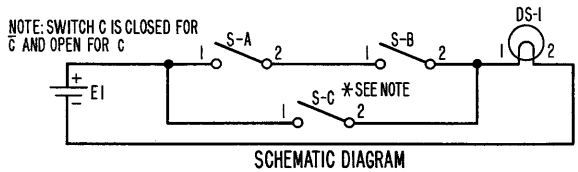


Fig. 3-10. The evaluator set up for (a · b) + $\overline{c}$.

BOOLEAN ALGEBRA

As demonstrated, the circuit combinations used in computers can be represented and analyzed in terms of symbolic logic. The different configurations we used with the truth evaluator experiments are good examples of the manner in which logic circuits combine. Such circuit combinations in computers, however, are much more complicated, although the basic circuit elements remain the same. Planning these circuit combinations with logic statements is an adequate method, but it is not entirely satisfactory. This is so because, in many cases, the combinations of logic circuits can be simplified, thereby eliminating unnecessary components and reducing expense and construction time. This simplification is carried out by the use of a form of algebra called Boolean algebra, which is named after its inventor, George Boole. By applying the rules of Boolean algebra to logic statements, we can simplify the statements by minimizing terms.

Boolean algebra uses the same logical conjunctions that we have already described: AND, OR, and NOT. However, instead of using T and F to indicate whether a statement is true or false, the binary numerals 1 and 0 are used. Using this convention, the following rules can be derived from the basic AND, OR, and NOT rules given at the beginning of this chapter.

AND	OR	NOT
$1 \cdot 1 = 1$	$1 + 1 = 1$	$\overline{1} = 0$
$1 \cdot 0 = 0$	$1 + 0 = 1$	$\overline{0} = 1$
$0 \cdot 1 = 0$	$0 + 1 = 1$	$\overline{\overline{1}} = 1$
$0 \cdot 0 = 0$	$0 + 0 = 0$	$\overline{\overline{0}} = 0$

Further expansion using 1 and 0 in place of T and F results in these additional rules:

AND	OR	NOT
$A \cdot 0 = 0$	$A + 0 = A$	$\overline{\overline{A}} = A$
$A \cdot 1 = A$	$A + 1 = 1$	
$A \cdot A = A$	$A + \overline{A} = 1$	
$A \cdot \overline{A} = 0$	$A + A = A$	

In regard to the algebraic manipulation of terms, Boolean algebra is very similar to regular algebra. For example the following statements represent rules for simple AND and OR statements.

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

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

The relative positions of the terms are of no significance in AND or OR statements (commutative and associative laws). Also (distributive law):

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

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

These equations can easily be proved by constructing truth tables for each side of the equation and comparing your results. If both statements in each equation produce the same truth table, they are equal. The required truth tables are:

a	b	c	$a \cdot b + a \cdot c$	=	$a \cdot (b + c)$
1	1	1	1	=	1
1	1	0	1	=	1
1	0	1	1	=	1
0	1	1	0	=	0
1	0	0	0	=	0
0	0	1	0	=	0
0	1	0	0	=	0
0	0	0	0	=	0

a	b	c	a + b · c	=	(a+b)·(a+c)
1	1	1	1	=	1
1	1	0	1	=	1
1	0	1	1	=	1
0	1	1	1	=	1
1	0	0	1	=	1
0	0	1	0	=	0
0	1	0	0	=	0
0	0	0	0	=	0

From this discussion, we can see that it is possible to reduce the number of terms and conjunctions in a Boolean equation (statement) by applying the basic rules, factoring, rearranging, etc., while still maintaining equality. This leads us to our next topic, minimizing terms.

MINIMIZING TERMS

In our study of Boolean algebra, we saw how two equal statements or equations might have different numbers of terms and conjunctions. The process of determining the smallest equal logical statement, given an original statement, is called *minimization*. To illustrate the effect of the minimization process, consider the statement $a + (a \cdot b)$. This statement can be instrumented by a logic circuit such as that shown in Fig. 3-11. Note that this logic circuit contains three switch poles. How-

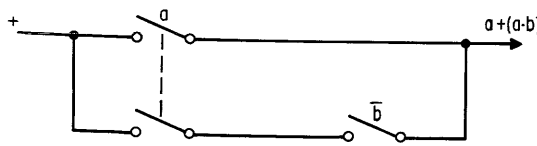


Fig. 3-11. The logic circuit for a + (a · b).

ever, by applying some of the rules of Boolean algebra we can reduce the number of terms as follows:

OPERATION	BOOLEAN RULE
$a \cdot \left[1 + (1 \cdot b)\right]$	a · 1 = a
a · (1)	1 + b = 1
a	a · 1 = a

Therefore, $a + (a \cdot b)$ equals a. The circuit configuration for this statement, Fig. 3-12, needs only one switch pole. Hence, by the process

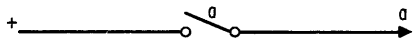


Fig. 3-12. The logic circuit for a.

of minimization, the circuit complexity has been reduced by two-thirds.

As a further example, take the statement:

$$a \cdot \overline{b} \cdot c + a \cdot b \cdot c + c \cdot (a \cdot \overline{b} + \overline{a} \cdot \overline{b}) + \overline{c} \cdot (a \cdot \overline{b} + \overline{a} \cdot \overline{b} + \overline{a} \cdot b)$$

The circuit for this statement in its present form is illustrated in Fig. 3-13.

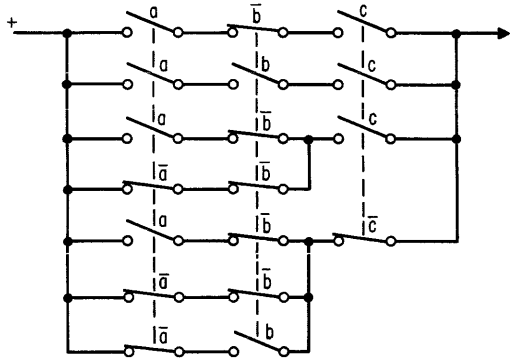


Fig. 3-13. The logic circuit of $a \cdot \overline{b} \cdot c + a \cdot b \cdot c + c \cdot (a \cdot \overline{b} + \overline{a} \cdot \overline{b}) + \overline{c} \cdot (a \cdot \overline{b} + \overline{a} \cdot \overline{b} + \overline{a} \cdot b)$.

This circuit requires 18 switch poles. Now, apply the rules of Boolean algebra to minimize terms:

1. Factor $a \cdot c$ from the first two terms:
$$a \cdot c \cdot (\overline{b} + b) = a \cdot c$$
since $(\overline{b} + b)$ must equal 1.
2. Factor $\overline{b}$ from the third term:
$$c \cdot \overline{b} \cdot (a + \overline{a}) = c \cdot \overline{b}$$
3. Factor $\overline{c} \cdot \overline{b}$ from the last term:
$$\overline{c} \cdot \overline{b} \cdot (a + \overline{a}) + \overline{c}\overline{a}b = \overline{c} \cdot \overline{b} + \overline{a}b\overline{c}$$
4. Now, the statement is:
$$a \cdot c + c \cdot \overline{b} + \overline{c} \cdot \overline{b} + \overline{a}b\overline{c}$$
5. Factor $\overline{b}$ from the second and third terms:
$$a \cdot c + \overline{b} \cdot (c + \overline{c}) + \overline{a}b\overline{c} = a \cdot c + \overline{b} + \overline{a}b\overline{c}$$
6. Since the b term in the last term is meaningless, because of $\overline{b}$ in the middle term (i.e., if $b = 0$, the statement is 1 due to the $\overline{b}$ term, and if b is 1 the statement is $a \cdot c + \overline{a} \cdot \overline{c}$), the b term can be dropped and the minimized statement becomes:
$$a \cdot c + \overline{b} + \overline{a} \cdot \overline{c}$$

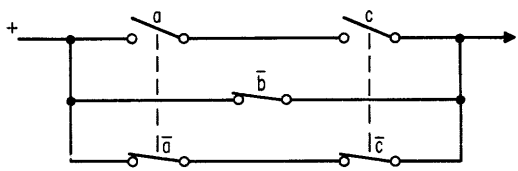


Fig. 3-14. The logic circuit for $a \cdot c + \overline{b} + \overline{a} + \overline{c}$.

The minimized statement, $a \cdot c + \overline{b} + \overline{a} \cdot \overline{c}$, is equal to the original statement as illustrated by their truth tables, which follow. This statement may be instrumented using the circuit of Fig. 3-14. Notice that now only 5 switch poles are required instead of the original 18.

a	b	c	$a \cdot \overline{b} \cdot c$	$a \cdot b \cdot c$	$c \cdot (\overline{a \cdot b} + \overline{a \cdot b})$	$\overline{c(a \cdot b + a \cdot b + a \cdot b)}$	$a \cdot \overline{b} \cdot c + a \cdot b \cdot c + c \cdot (\overline{a \cdot b} + \overline{a \cdot b}) + c(a \cdot \overline{b} + \overline{a} \cdot \overline{b} + \overline{a} \cdot b)$
1	1	1	0	1	0	0	1
1	1	0	0	0	0	0	0
1	0	1	1	0	1	0	1
0	1	1	0	0	0	0	0
1	0	0	0	0	0	1	1
0	0	1	0	0	1	0	1
0	1	0	0	0	0	1	1
0	0	0	0	0	0	1	1

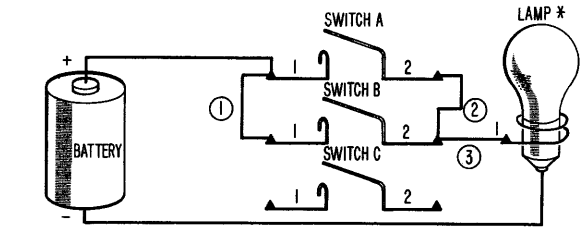
a	b	c	$a \cdot c$	$\overline{b}$	$\overline{a \cdot c}$	$a \cdot c + \overline{b} + \overline{a \cdot c}$
1	1	1	1	0	0	1
1	1	0	0	0	0	0
1	0	1	1	1	0	1
0	1	1	0	0	0	0
1	0	0	0	1	0	1
0	0	1	0	1	0	1
0	1	0	0	0	1	1
0	0	0	0	1	1	1

DE MORGAN'S THEOREM

A very valuable tool in the study and application of logic and Boolean algebra is De Morgan's theorem. This theorem says that the statement "NOT: *a*-or-*b*-or-*c*" is equivalent to the statement "not-*a* and not-*b* and not-*c*," or, in Boolean algebra:

$$\overline{(a + b + c)} = \overline{a} \cdot \overline{b} \cdot \overline{c}$$

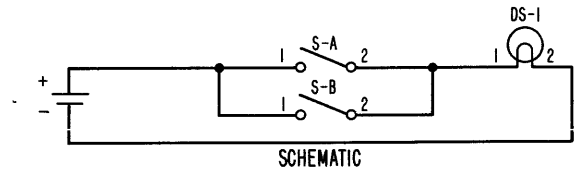
This theorem provides a relationship between the AND and OR connectives and thereby allows interchanging connectives for convenience. A further reason for the importance of this theorem is its widespread use in transistorized logic circuits. Many circuit designs that use transistors employ basic circuit units that instrument the logic $\overline{a \cdot b}$ or $\overline{a + b}$;



* NOTE: IN THIS CASE, WHEN LAMP LIGHTS, INDICATES A ZERO; WHEN LAMP STAYS OUT, INDICATES A ONE

WIRE	FROM	TO
POSITIVE LEAD	BATTERY (+)	SWITCH A (1)
1	SWITCH A (1)	SWITCH B (1)
2	SWITCH A (2)	SWITCH B (2)
3	SWITCH B (2)	LAMP (1)

WIRING DIAGRAM



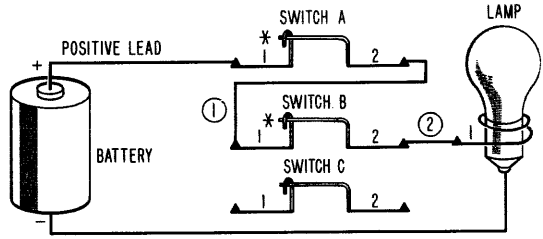
SCHEMATIC

Fig. 3-15. The evaluator set-up for $\overline{a + b}$.

i.e., NAND (not-and), or NOR (not-or). With this logic, to instrument the statement $\bar{a} \cdot \bar{b}$, you may use a NOR circuit $(\overline{a + b})$ since by De Morgan's theorem, $a + b$ equals $\bar{a} \cdot \bar{b}$.

To prove the equivalence of the two parts of De Morgan's theorem, set up the truth evaluator in Figs. 3-15 and 3-16 to determine that both statements have the following truth table:

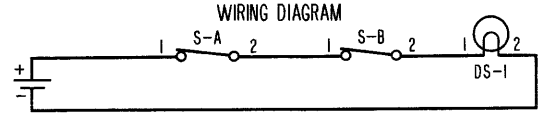
Fig. 3-16. The evaluator set-up for $\bar{a} \cdot \bar{b}$.



*NOTE: SWITCHES MUST NORMALLY BE CLOSED FOR THIS SET-UP. WHEN a IS TRUE, SWITCH A IS OPENED AND VICE VERSA.

WIRE	FROM	TO
POSITIVE LEAD	BATTERY (+)	SWITCH A (1)
1	SWITCH A (2)	SWITCH B (1)
2	SWITCH B (2)	LAMP (1)

WIRING DIAGRAM



SCHEMATIC DIAGRAM

a	b	$\overline{a+b}$	$\bar{a} \cdot \bar{b}$
1	1	0	0
1	0	0	0
0	1	0	0
0	0	1	1

THE UN PROBLEM—AN EXPERIMENT IN LOGIC

This problem consists of devising a means for instantaneously registering the results of a secret vote taken in an organization built along the lines of the United Nations Security Council. The organization consists of five members, each of whom may vote yes or no independently. In addition, two of the five members have a veto power; i.e., if either or both of these members exercise their veto the result is no, regardless of the votes of the other members.

Each member of the organization has a switch assigned to him which he must close for a yes vote or open for a no vote. If the members are designated A, B, C, D, and E, the following statements may be defined:

Statement	*Meaning*
a	Member A votes yes
b	Member B votes yes
c	Member C votes yes
d	Member D votes yes
e	Member E votes yes

Finally, let us assign the previously mentioned veto power to members A and B.

The problem has thus been completely defined. The first step in the logical design of this device is to write the logical statement of the problem which, in this case, is the compound statement corresponding to a yes vote. From the logic of the problem:

$$a \cdot b \cdot (c + d + e) = \text{YES}$$

Both *a* and *b* must be true to satisfy the veto condition, and at least one of statements *c, d,* or *e* must be true to provide a majority.

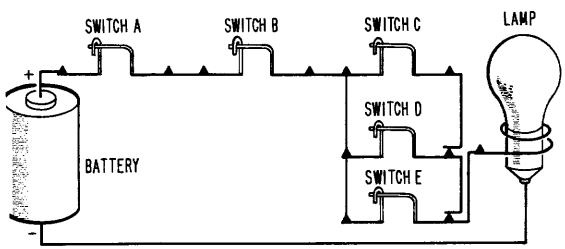


Fig. 3-17.

The logic circuit which instruments this logical statement so that a lamp lights to indicate a YES result is illustrated in Fig. 3-17. Refer to the construction details at the rear of this chapter and build the voting machine.

CONSTRUCTION DETAILS—TRUTH EVALUATOR

COMPONENTS: *Panel, switches, battery holder*

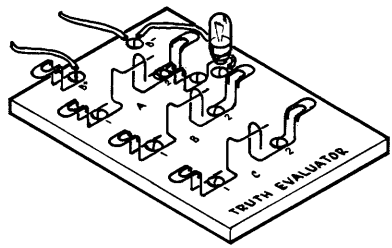


Fig. C-1.

MATERIALS

1	8 × 10 × ⅛ in. composition board
1	battery holder (construction details follow)
3	½ in. machine screws (6-32)
3	nuts (6-32)
7	½ in. wood screws (no. 4)
	9 ft. insulated hook-up wire (20 gage)
39	paper clips
1	#48 or 41 lamp (2 v, .06 a)

SPECIAL TOOLS:

Drill (⅛, 3/32)

Truth Evaluator Construction

1. Mark the 8 × 10 × ⅛-in. composition board as in Fig. C-2.
2. Start drill holes at each point and drill as indicated. (The ⅛-in. holes will hold machine screws for switch and terminal connections.)

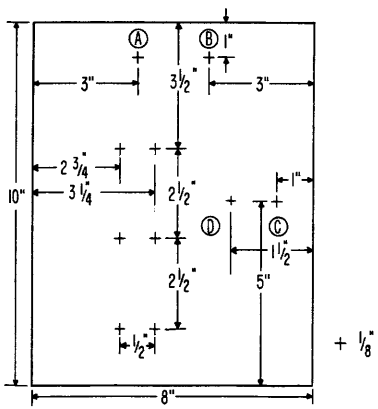


Fig. C-2.

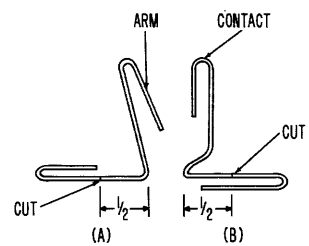


Fig. C-3.

3. Each SPST switch is made from 2 paper clips. Bend 3 paper clips as in Fig. C-3(a). Bend 3 other paper clips as in Fig. C-3(b).
4. Cut each paper clip for each configuration as shown in Fig. C-3. The ½-in. section will be crimped around a wood screw to secure the switch parts on the panel in an upright position.
5. The quick-disconnect terminals are also made with paper clips. Their configuration and use are shown in Fig. C-4. Construct 8 of these terminals.

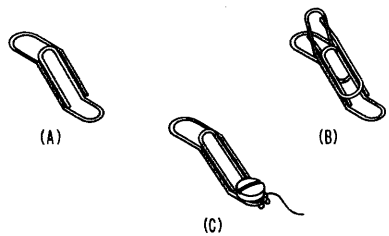


Fig. C-4.

6. Install 6 quick-disconnect terminals and 3 SPST switches on the panel top as shown in Fig. C-5. Use ½-in. wood screws. Note that each arm or contact of the SPST switches has a quick-disconnect terminal attached to it. Put the switch arms in the first vertical row on the left side of the panel (as illustrated) with the arms pointing toward the right side of the panel. Place the contacts so that the open loop is towards the top of the panel. As each quick-disconnect terminal and switch contact or arm is secured in place, crimp the arm or contact around the screw to hold it upright on the board.
7. When the switches are in place, cut each arm so that it extends past the contact about ⅛ in.
8. Cut each contact so that it is about ½ in. long on the open loop.
9. Bend the switch arms to the outside of the contact so that there is enough tension to spring the arm away from the contact when it is not locked in place. The make or break motion is similar to the opening or closing of a safety pin.
10. Cut three 7-in. lengths of insulated wire and strip ½ in. of insulation from each end of each length.

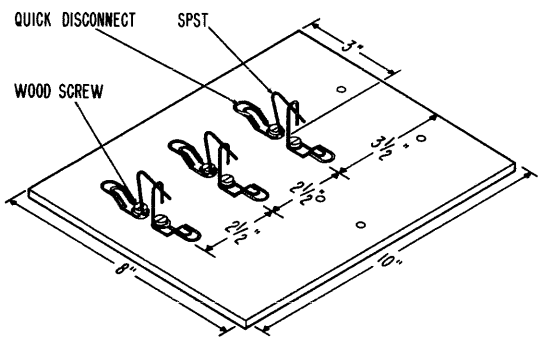


Fig. C-5.

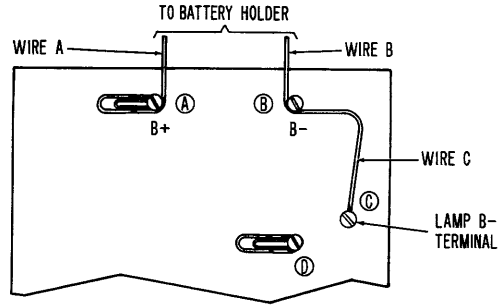


Fig. C-6.

11. As in Fig. C-6, place ½-in. machine screws in holes A, B, and C and attach nuts. Do not tighten the nuts.
12. Attach one 7-in. wire and a quick-disconnect terminal to the screw in hole A and attach wires to B and between B and C, as indicated in Fig. C-6.
13. Construct a lamp socket by wrapping a paper clip around the base of the lamp, as illustrated in Fig. C-7(a).

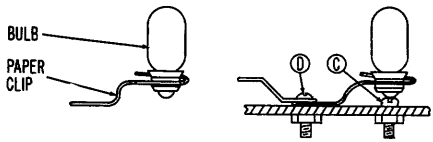


Fig. C-7.

14. Attach the socket constructed in Step 13 and a quick-disconnect terminal to point D (Fig. C-7(b)) with a ½-in. wood screw. *Caution:* The socket should be positioned directly over the machine-screw head acting as the B— terminal. The socket, however, should never touch this terminal, but it must be close enough to it so that the base of a bulb screwed into the socket will contact the terminal.
15. Label the switches and terminals as shown in Fig. C-8.

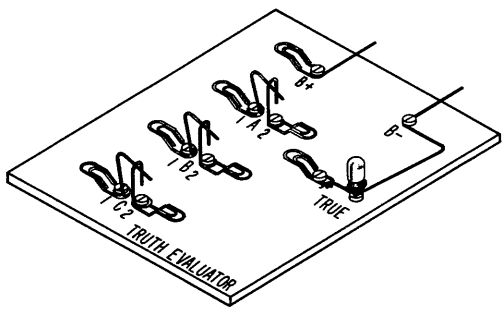


Fig. C-8.

Fig. C-9.

16. Cut 6 9-in. lengths of insulated hook-up wire and 6 5-in. lengths. Strip ¾ in. of insulation from each end of each wire.
17. Attach each end of each wire to a paper clip as shown in Fig. C-9. Use these wires to make the various connections in the experiments involving the truth evaluator by clipping the wires to the appropriate quick-disconnect terminals. Always make certain that the connections are solid before proceeding with the experiment.

CONSTRUCTION DETAILS—BATTERY HOLDER

COMPONENTS: *Chassis, batteries*

MATERIALS

1	4 × 5 × ½ in. block of wood
8	1½ in. wire brads
2	1½ volt flashlight batteries
2	1 in. machine screws (6-32)
8	nuts (6-32)
	1 ft. insulated wire (20 gage)
	Adhesive tape

Chassis Construction

1. Place the batteries side by side, facing them in opposite directions. Tape them together.
2. Put the batteries in the center of the 4 × 5 × ½-in. block and secure them on both sides with brads driven into the block. See Fig. C-10.

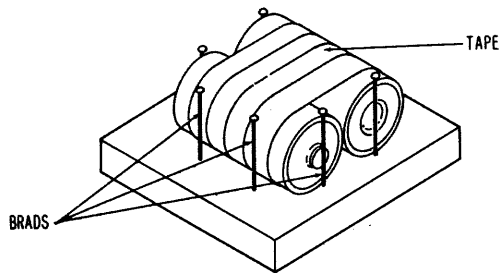


Fig. C-10.

3. Drive brads into the block at both ends of each battery, making sure that contact is made between each battery terminal and the brad in front of it. See Fig. C-10.
4. At one end of the block, jump a short length of insulated wire, stripped at its ends, between the two brads.
5. At the other end of the block, connect a single wire to each brad.
6. Drill two 3/32-in. holes at the front edge of the wood block, immediately in front of the brads that have single wires connected to them. See Fig. C-11.
7. On the bottom of the block, enlarge the holes drilled in Step 6 so that

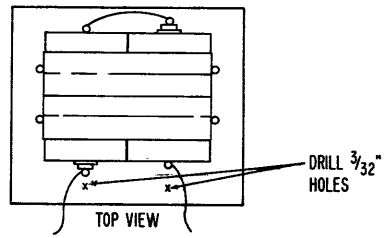


Fig. C-11.

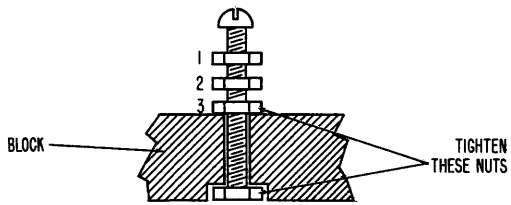


Fig. C-12.

they will accommodate the nuts for a 1-in. machine screw. See Fig. C-12.

8. Place three nuts on each of the 1-in. machine screws. Advance each nut so that the last one screwed on is approximately ½ inch from the end of the screw.
9. Insert the screws into the drilled holes from the top of the block. Secure them with a fourth nut on the bottom. As in Fig. C-12, tighten the bottom nut and the top nut nearest the block to make the screw rigid.
10. Attach the free end of the wire connected in Step 5 from the brad to the screw, placing it between the nuts marked 2 and 3 in Fig. C-12. Do this with both of the free wires. Tighten nut 2 down onto nut 3.
12. Use additional tape to ensure that the brads continually make contact with the battery terminals.
13. To use the batteries to power computer units, connect the power leads to the screw between nuts 1 and 2 and tighten nut 1 down onto nut 2.

CONSTRUCTION DETAILS—UN PROBLEM

COMPONENTS: *Panel, switches, battery holder*

MATERIALS

1	8 × 10 × ⅛ in. composition board
12	½ in. machine screws (6-32)
12	nuts (6-32)
	5 ft. insulated hook-up wire (20 gage)
10	paper clips
1	battery holder (see truth evaluator construction details)
1	#48 or 41 lamp (2 v, .06 a)

UN Problem Construction

1. Mark the 8 × 10-in. board as shown in Fig. D-1.
2. Start drill holes and drill through the board with indicated bit size.

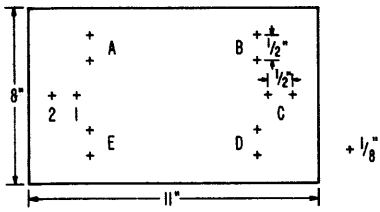


Fig. D-1.

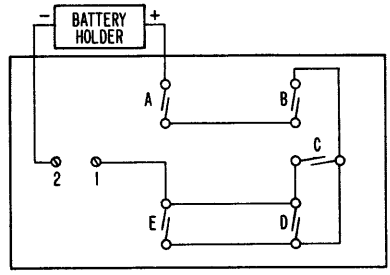


Fig. D-2.

3. Make 5 SPST paper-clip switches. Refer to the truth evaluator construction procedures for construction details.
4. Install the SPST switches at points A through E using ½-in. machine screws and nuts.
5. Make a bulb socket following the details given in the truth evaluator construction details.
6. Install a machine screw at point 2 and secure it with a nut.
7. Attach the socket constructed in Step 5 to a machine screw at point 1 (Fig. D-2). The socket should be positioned directly over the machine-screw head at point 2. The socket, however, should not touch the machine screw. Place it so that the base of the bulb it holds will maintain contact with the machine screw.
8. Wire the board as shown in Fig. D-2 with insulated hook-up wire.
9. Label the panel as shown in Fig. D-2.

Chapter 4
COMPUTER ARITHMETIC

The computer performs all of its arithmetic operations in its arithmetic unit. This unit is the central or focal point of the computer, for all calculations and processing are carried out in it. The arithmetic unit of a computer generally consists of several storage devices called *registers,* plus circuits that perform addition and subtraction. The basic register in the arithmetic unit is called the *accumulator* since the results of almost all computer operations "accumulate" in this register. The number and size of the registers and other circuits in the arithmetic unit are dependent on the complexity and versatility of the overall computer and its primary function. In this chapter, computer arithmetic will be discussed and a basic arithmetic unit described and constructed for use in our computer.

BINARY ARITHMETIC

In Chapter 2, we learned about the binary number system, which is the system used by computers. The relationship between binary and decimal numbers (and vice versa) was discussed and units that performed decimal-to-binary and binary-to-decimal number conversions were constructed. These units allow raw rata to be inserted into our computer in decimal numbers and processed data to be returned in decimal numbers. However, internally, the computer performs all of its operations using binary numbers. In the following paragraphs, the basic rules of binary addition and subtraction will be described. Then, using these rules, we will construct an arithmetic unit capable of performing operations of addition and subtraction automatically. The arithmetic operations of multiplication and division (which we will be able to do on our computer) will be accomplished by performing multiple additions or subtractions in sequence under control of a stored program. These operations are discussed in Chapter 7.

BASIC RULES

As with any numbering system, there are a certain number of simple, basic rules in the binary system on which the more complicated arithmetic operations rest. These rules are the basic rules of single-column addition, subtraction, multiplication, and division. Since the

binary system contains only two numerals, the rules for this system are both simple and few. These rules are contained in the following tables.

TABLE 4-1. BINARY RULES OF ADDITION

ADDEND		AUGEND		SUM	CARRY TO NEXT COLUMN
0	+	0	=	0	0
0	+	1	=	1	0
1	+	0	=	1	0
1	+	1	=	0	1

TABLE 4-2. BINARY RULES OF SUBTRACTION

MINUEND		SUBTRAHEND		DIFFERENCE	BORROW FROM NEXT COLUMN
0	-	0	=	0	0
0	-	1	=	1	1
1	-	0	=	1	0
1	-	1	=	0	0

TABLE 4-3. BINARY RULES OF MULTIPLICATION

MULTIPLICAND		MULTIPLIER		PRODUCT
0	×	0	=	0
0	×	1	=	0
1	×	0	=	0
1	×	1	=	1

TABLE 4-4. BINARY RULES OF DIVISION

DIVIDEND		DIVISOR		QUOTIENT
0	÷	0	=	Undefined
0	÷	1	=	0
1	÷	0	=	Undefined
1	÷	1	=	1

These basic rules define the binary arithmetic functions for single-column operations. However, since we hardly require the assistance of

a computer for numbers of single-column magnitude, let us proceed to discuss and analyze the arithmetic functions as applied to higher-order numbers. This chapter concerns itself with this analysis and with reducing our findings to a series of logical statements that can be readily converted, using basic logic circuits (see Chapter 3), into a device to be used in our computer as the arithmetic unit.

ADDING BINARY NUMBERS

The addition of higher-order binary numbers using the basic rules tabulated earlier can best be illustrated by an example, as follows:

Decimal	*Binary*									
		1					1	1		Carry
		↓					↓	↓		
210		0	1	1	0	1	0	0	1	0	Augend
+135		+0	1	0	0	0	0	1	1	1	Addend
345		1	0	1	0	1	1	0	0	1	Sum

The process of addition illustrated in the foregoing examples consists of adding each column in sequence starting from the least-significant column (right-hand) and working towards the most-significant column. Note that, in accordance with our rules for addition, when the addend and augend are both 1, a 1 is carried to the next column. The algebra of addition for a single column of binary numbers is expressed as follows:

Sum = Augend + Addend + Carry (from the previous column)

We can now analyze this process logically by means of a truth table (Table 4-5) which uses binary 1 to represent True and binary 0 to represent False.

TABLE 4-5. ADDITION TRUTH TABLE

AUGEND	ADDEND	CARRY (Previous column)	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
1	0	0	1	0
0	1	1	0	1
1	1	0	0	1
1	0	1	0	1
1	1	1	1	1

From this table we can gather the compound statements for Sum = 1 and Carry = 1 and write the following statements, which cover the addition of a single column of binary numbers:

$$S = (\overline{A}_u \cdot \overline{A}_d \cdot C_{pc}) + (\overline{A}_u \cdot A_d \cdot \overline{C}_{pc}) + (A_u \cdot \overline{A}_d \cdot \overline{C}_{pc}) + (A_u \cdot A_d \cdot C_{pc}) \quad (1)$$

$$C = (A_u \cdot A_d \cdot C_{pc}) + (\overline{A}_u \cdot A_d \cdot C_{pc}) + (A_u \cdot \overline{A}_d \cdot C_{pc}) + (A_u \cdot A_d \cdot \overline{C}_{pc}) \quad (2)$$

where: S = Sum
A_u = Augend
A_d = Addend
C_{pc} = Carry (previous column)
C = Carry

At this point, let us refresh our understanding of Boolean algebra by referring back to Chapter 3. Using the basic rules of Boolean algebra and logic, we can rewrite statements (1) and (2) so that they contain fewer component simple statements, thereby reducing the number of switches necessary for their instrumentation. For the summation statement:

1. *Process:* Group the terms containing C_{pc} and factor C_{pc}.
   Result: $S = C_{pc} \cdot [(A_u \cdot A_d) + (\overline{A}_u \cdot \overline{A}_d)] + (A_u \cdot \overline{A}_d \cdot \overline{C}_{pc}) + (\overline{A}_u \cdot A_d \cdot \overline{C}_{pc})$
2. *Process:* Group the terms containing $\overline{C}_{pc}$ and factor $\overline{C}_{pc}$.
   Result: $S = C_{pc} \cdot [(A_u \cdot A_d) + (\overline{A}_u \cdot \overline{A}_d)] + \overline{C}_{pc} \cdot [(A_u \cdot \overline{A}_d) + (\overline{A}_u \cdot A_d)]$ (3)

By counting terms it can be seen that the number of statements comprising the summation statement has been reduced from 12 to 10.

Now let us minimize the logical statement for "carry" (2).

1. *Process:* Group the terms containing $(A_u \cdot A_d)$ and factor $(A_u \cdot A_d)$.
   Result: $(C_{pc} \cdot \overline{A}_u \cdot A_d) + (C_{pc} \cdot A_u \cdot \overline{A}_d) + (A_u \cdot A_d)(C_{pc} + \overline{C}_{pc})$
2. *Process:* Since $(C_{pc} + \overline{C}_{pc}) = 1$.
   Result: $(C_{pc} \cdot \overline{A}_u \cdot A_d) + (C_{pc} \cdot A_u \cdot \overline{A}_d) + (A_u \cdot A_d)$
3. *Process:* Group the terms containing C_{pc} and factor C_{pc}.
   Result: $C_{pc}[(\overline{A}_u \cdot A_d) + (A_u \cdot \overline{A}_d)] + (A_u \cdot A_d)$ (4)

In this case, we have reduced the number of terms from 12 to 7.

At this point, logical statements have been derived for S and C that, together, cover the entire process of addition for a single column of binary numbers. However, prior to mechanization of these statements to form an adder, we must also derive the statement $\overline{C}$ since this statement is used for summation. To derive $\overline{C}$, we again refer to the truth table for addition and gather the compound statements corresponding to $C = 0$ $(\overline{C} = 1)$.

$$\overline{C} = (A_u \cdot \overline{A}_d \cdot \overline{C}_{pc}) + (\overline{A}_u \cdot A_d \cdot \overline{C}_{pc}) + (\overline{A}_u \cdot \overline{A}_d \cdot C_{pc}) + (\overline{A}_u \cdot \overline{A}_d \cdot \overline{C}_{pc}) \quad (5)$$

The "not-carry" statement, (5), is minimized by the following process:

1. *Process:* Group the terms containing $\overline{C}_{pc}$ and factor $\overline{C}_{pc}$.
   Result: $(C_{pc} \cdot \overline{A}_u \cdot \overline{A}_d) + \overline{C}_{pc}\,[(A_u \cdot \overline{A}_d) + (\overline{A}_u \cdot A_d) + (\overline{A}_u \cdot \overline{A}_d)]$
2. *Process:* In the $\overline{C}_{pc}$ term, factor $\overline{A}_u$.
   Result: $(C_{pc} \cdot \overline{A}_u \cdot \overline{A}_d) + \overline{C}_{pc}\,[\overline{A}_u(A_d + \overline{A}_d) + (A_u \cdot \overline{A}_d)]$
3. *Process:* Since $A_d + \overline{A}_d = 1$.
   Result: $(C_{pc} \cdot \overline{A}_u \cdot \overline{A}_d) + \overline{C}_{pc}\,[\overline{A}_u + (A_u \cdot \overline{A}_d)]$
4. *Process:* Referring to $\overline{C}_{pc}$ terms, if $A_u = 1$, term in brackets equals $\overline{A}_d$; if $A_u = 0$, then $[\overline{A}_u + (A_u \cdot \overline{A}_d)] = \overline{A}_u$.
   Result: $(C_{pc} \cdot \overline{A}_u \cdot \overline{A}_d) + \overline{C}_{pc}\,[(\overline{A}_u + \overline{A}_d)]$ (6)

Since statements (4), (5), and (6) apply to the addition of a single column, it is necessary to use a complete set of these three statements for each column of the numbers to be added.

For example, if we have a four-column number (i.e., 0101), we must mechanize each set of three statements (S, C, $\overline{C}$) four times. For example:

S_0	C_0	$\overline{C}_0$	*First column* (least significant)
S_1	C_1	$\overline{C}_1$	*Second column*
S_2	C_2	$\overline{C}_2$	*Third column*
S_3	C_3	$\overline{C}_3$	*Fourth column*

The answer to an addition problem involving two four-digit numbers thus consists of four individual summations, S_0 through S_3, arranged in the following order:

$$S_3 \quad S_2 \quad S_1 \quad S_0$$

The circuit used to mechanize each set of three statements corresponding to a single column is called a *stage*. In the above case, four stages are required to mechanize the number completely. The design of each stage is identical since it must mechanize the same logical statements. However, it is possible to simplify the mechanization of the first stage since there are no C_{pc} statements involved. The general statements for a stage are:

$$S = C_{pc} \cdot [(A_u \cdot A_d) + (\overline{A}_u \cdot \overline{A}_d)] + \overline{C}_{pc} \cdot [(A_u \cdot \overline{A}_d) + (\overline{A}_u \cdot A_d)]$$
$$C = (A_u \cdot A_d) + C_{pc} \cdot [(\overline{A}_d \cdot A_u) + (A_d \cdot \overline{A}_u)]$$
$$\overline{C} = (C_{pc} \cdot \overline{A}_u \cdot \overline{A}_d) + \overline{C}_{pc}\,(\overline{A}_u + \overline{A}_d)$$

If we substitute a 0 for C_{pc} in every case:

$$S = (A_u \cdot \overline{A}_d) + (\overline{A}_u \cdot A_d) \quad (C_{pc} = 0; \text{ therefore } \overline{C}_{pc} = 1)$$
$$C = (A_u \cdot A_d)$$
$$\overline{C} = \overline{A}_u + \overline{A}_d$$

Note: We previously defined binary 1 as being the equivalent of True and binary 0 as being equivalent to False. Therefore $0 \cdot N = 0$ and $1 \cdot N = N$.

Prove the statement derived for $\overline{C}$ by using the statement C and De Morgan's Theorem.

To summarize:

First Stage

$S = (A_u \cdot \overline{A}_d) + (\overline{A}_u \cdot A_d)$

$C = (A_u \cdot A_d)$

$\overline{C} = \overline{A}_u + \overline{A}_d$

Remaining Stages

$S = C_{pc} \cdot [(A_u \cdot A_d) + (\overline{A}_u \cdot \overline{A}_d)] + \overline{C}_{pc} \cdot [(A_u \cdot \overline{A}_d) + (\overline{A}_u \cdot A_d)]$

$C = (A_u \cdot A_d) + C_{pc} \cdot [(\overline{A}_d \cdot A_u) + (A_d \cdot \overline{A}_u)]$

$\overline{C} = (C_{pc} \cdot \overline{A}_u \cdot \overline{A}_d) + \overline{C}_{pc} \cdot [\overline{A}_u + \overline{A}_d]$

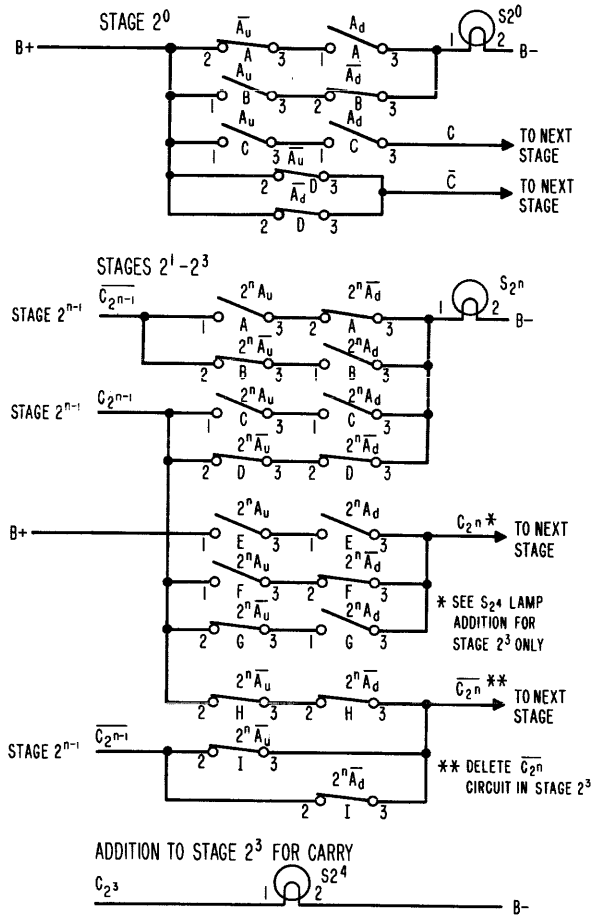


Fig. 4-1. The circuits of a four-column binary adder.

There is one remaining consideration; the possibility of an $(n + 1)$-digit sum resulting from the addition of two n-digit numbers. However, in this case, the $(n + 1)$ digit is merely the carry from the n^{th} column. Keeping this in mind, let us refer back to Chapter 3 for the discussion of basic circuits and then compare the circuits illustrated in Fig. 4-1 for a four-column binary adder with the statements summarized above. These circuits implement the statements for addition and hence provide an automatic means for adding any two four-digit numbers accurately and instantaneously.

At this point, it is not necessary to build this adder since a complete arithmetic unit will be devised and constructed, using these statements, later in the chapter. However, at this point, we can appreciate a basic advantage of the computer as a practical device. Note that the design of the adder just discussed prevents it from making an error (unless a malfunction develops). Over and over again, this adder will unerringly calculate the sum of two four-digit numbers at great speed without feeling fatigue and without losing efficiency. Even at the relatively slow speed of our adder due to the use of mechanical switching, the calculations are performed many times faster than the best human calculator. If we consider the electronic switching speeds of modern computers, which are of the order of billionths of a second, the magnitude of the computer's capability becomes apparent.

SUBTRACTION

The process of subtraction can be illustrated by the following example:

Decimal		*Binary*									
210	← Minuend →	0	1	1	0	1	0	0	1	0	
−135	← Subtrahend →	0	1	0	0	0	0	1	1	1	
						↓	↓	↓	↓		
11	← Borrow →					1	1	1	1		Borrow
075		0	0	1	0	0	1	0	1	1	

The borrow term is thought of and used in several different ways, each equally valid. Some people speak of borrowing a 1 from the minuend digit of the next most significant column by adding 10 (in the case of decimal numbers) to the minuend digit of the column being processed and subtracting 1 from the minuend digit of column borrowed from. For our purposes, however, we will adapt the method consisting of borrowing 10 from the next most significant column by adding 10 to the minuend digit of the column being processed and adding 1 to the subtrahend digit of the column borrowed from.

As in addition, the process of subtraction consists of subtracting each column in sequence, starting from the least significant column (right-hand) and working towards the most significant column. The

algebra of subtraction for a single column of binary numbers is expressed as follows:

$$\begin{array}{lcl} \text{Difference} = \text{Minuend} - & [\text{Subtrahend} + & \text{Borrow (previous column)}] \\ D = M - & [S + & B_{pc}] \end{array}$$

We can logically analyze this process as before, using a truth table (Table 4-6).

TABLE 4-6. SUBTRACTION TRUTH TABLE

MINUEND	SUBTRAHEND	BORROW (pc)	DIFFERENCE	BORROW
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
1	0	0	1	0
0	1	1	0	1
1	1	0	0	0
1	0	1	0	0
1	1	1	1	1

From this truth table we can write the following logical statements covering the subtraction of a single column of binary numbers:

$$D = (M \cdot S \cdot B_{pc}) + (M \cdot \overline{S} \cdot \overline{B}_{pc}) + (\overline{M} \cdot \overline{S} \cdot B_{pc}) + (\overline{M} \cdot S \cdot \overline{B}_{pc}) \tag{7}$$

$$B = (M \cdot S \cdot B_{pc}) + (\overline{M} \cdot S \cdot B_{pc}) + (\overline{M} \cdot \overline{S} \cdot B_{pc}) + (\overline{M} \cdot S \cdot \overline{B}_{pc}) \tag{8}$$

$$\overline{B} = (M \cdot S \cdot \overline{B}_{pc}) + (M \cdot \overline{S} \cdot B_{pc}) + (M \cdot \overline{S} \cdot \overline{B}_{pc}) + (\overline{M} \cdot \overline{S} \cdot \overline{B}_{pc}) \tag{9}$$

Using the same methods employed earlier for addition, statements (7), (8), and (9) may be reduced to their minimum form.

$$D = B_{pc} \cdot [(M \cdot S) + (\overline{M} \cdot \overline{S})] + \overline{B}_{pc} \cdot [(M \cdot \overline{S}) + (\overline{M} \cdot S)] \tag{10}$$

$$B = (\overline{B}_{pc} \cdot \overline{M} \cdot S) + B_{pc} \cdot (\overline{M} + S) \tag{11}$$

$$\overline{B} = (B_{pc} \cdot M \cdot \overline{S}) + \overline{B}_{pc} \cdot (M + \overline{S}) \tag{12}$$

These are the general statements used in the design of the stages of a subtractor. However, as for addition, the first stage may be simplified

since there can be no borrow (from a previous column). Therefore, setting B_{pc} equal to 0, the first stage statements are:

$$D = (M \cdot \overline{S}) + (\overline{M} \cdot S)$$
$$\underline{B} = \overline{M} \cdot S$$
$$\overline{B} = M + \overline{S}$$

Should the subtrahend exceed the minuend, the last stage of the subtractor will generate a borrow. This borrow must be displayed as a negative sign since it will occur only when the difference is a negative number, in which case, the number:

$$D_n D_{n-1} \ldots D_1 D_0$$

is in complemented form. Complementary numbers are discussed in the

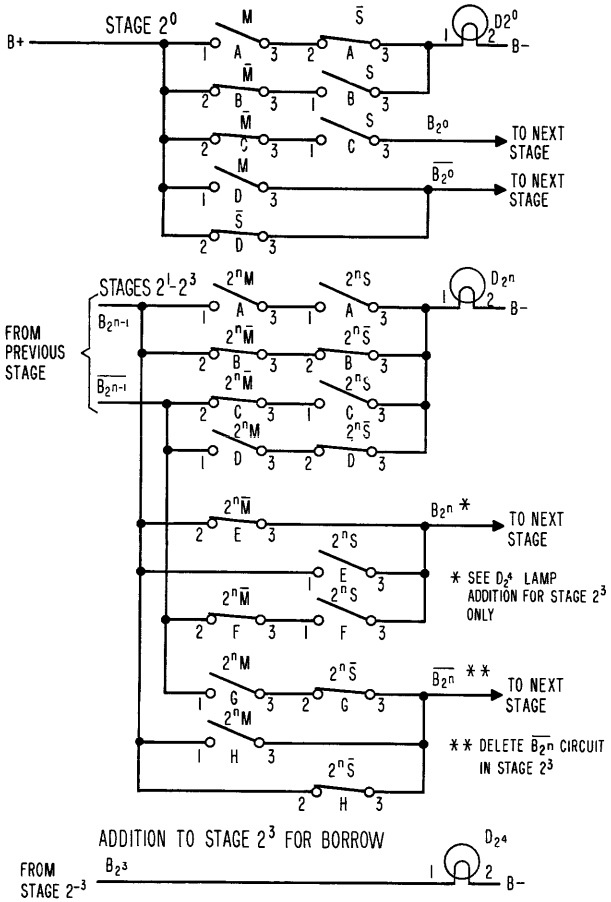


Fig. 4-2. The circuits of a binary subtractor.

next section. For the present let us confine our study to those cases that produce positive differences and limit our study of negative differences to recognizing the sign in the $(n + 1)^{th}$ digit.

Figure 4-2 illustrates the circuit required to instrument the statements for subtraction. The subtractor illustrated will subtract any two four-digit numbers and display the difference (in complemented form for negative differences). Again, it is not necessary to build the subtractor since our arithmetic unit will be capable of both addition and subtraction and will be based on the logical statements derived in this chapter.

COMPLEMENTARY NUMBERS

Complementary numbers are those numbers that, when added to the number being complemented, result in zero (neglecting the $(n + 1)^{th}$ digit). For example, in the decimal numbering system, the complement of 3579 is 6421:

$$\begin{array}{r} 3579 \\ +6421 \\ \hline 10000 \end{array}$$

In the binary numbering system, the complement of 0101 is 1011:

$$\begin{array}{r} 0101 \\ +1011 \\ \hline 10000 \end{array}$$

In this manner, one can consider the complement as being the equivalent negative number:

$$\begin{array}{r} 0101 \\ +1011 \\ \hline 10000 \end{array} \qquad \begin{array}{r} 5 \\ +\ (-5) \\ \hline 0 \end{array}$$

To obtain a complemented number, subtract the number to be complemented from 2^N, where N is the number of digits in the number. For example, if we wish to obtain the complement of 0110, the number of digits is 4 and, hence, 2^4 equals $(10000)_2$. Subtracting 0110 from 10000:

$$\begin{array}{r} 10000 \\ -\ 0110 \\ \hline 1010 \end{array}$$

Therefore, 1010 is the complement of 0110 ($(0110)_2 = (6)_{10}$ and $(1010)_2 = (-6)_{10}$). The following additional examples illustrate the process for complementing a number, x.

2^N	10000	10000	10000	10000
$-\ (x)$	0001	1001	0111	0100
$(-x)$	1111	0111	1001	1100

A "short-cut" method of obtaining a complement of a number is as follows:

1. Reverse all digits in the number to be complemented; i.e., all 0's become 1's and all 1's become 0's. For example, 0111 becomes 1000.
2. Add a 1 to the least significant digit of the reversed number: $1000 + 1 = 1001$.
   The result is the complement number.

To illustrate the equivalence of this method, the examples below are the same numbers complemented by the method in the earlier examples:

x	0001	1001	0111	0100
Reverse	1110	0110	1000	1011
Add 1	+ 1	+ 1	+ 1	+ 1
$(-x)$	1111	0111	1001	1100

In many digital computers, complementary numbers are used to represent negative numbers. If three-digit numbers are required to solve the problems for which a computer is designed, a fourth digit is added to be used as a sign digit. For example, if the magnitude of a number is 3 (011) the number is represented as 0011 with the 0 in the most significant column representing a positive sign. The number —3 is then represented by the complement 1101 with the 1 in the most significant column representing the negative sign.

The significance of this method will become clearer after the arithmetic unit is constructed and various mathematical operations involving negative numbers are performed on it.

MULTIPLICATION AND DIVISION

Multiplication and division of binary numbers are performed by a programmed series of additions and/or subtractions. Since we perform multiplication and division as programming exercises in Chapter 7, these processes are not discussed here.

ARITHMETIC UNIT

Now that we have discussed and analyzed the logic of binary arithmetic, it is necessary to design the unit that performs the actual computation, the arithmetic unit. First, of course, let's state what our arithmetic unit must be capable of doing.

The arithmetic unit must be capable of performing addition or subtraction of two numbers stored within its own registers and of storing

the result in its own registers. Inputs to the arithmetic unit will be 4-bit numbers.

The basic functional units required for an arithmetic unit usually are:

Add/Subtract Circuit
Accumulator
X-Register
Extension Register

In addition, we require a display for indicating the contents of the accumulator and extension register. Figure 4-3 illustrates the resulting functional configuration of the arithmetic unit.

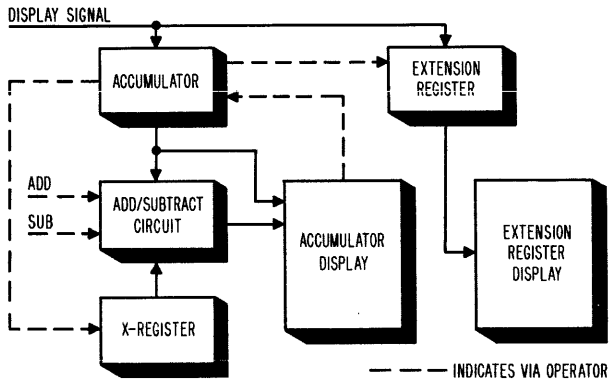


Fig. 4-3. A block diagram of the arithmetic unit.

The accumulator is the basic register of the computer and consists of four stages (one stage for each digit of the 4-digit number) plus a sign or carry stage for a total of five stages. In addition, an extension register of four additional stages is provided to allow for larger numbers obtained when performing multiplication. The contents of all nine register stages will be displayed automatically when a DISPLAY signal is fed to the registers. In addition to accumulating the results of computer operations, the accumulator is used to store the augend or the minuend prior to an addition or subtraction operation, respectively.

A second register, called the X-register, is also provided in the arithmetic unit. This register is used basically to store the addend or subtrahend just prior to an addition or subtraction operation, respectively. Data may be transferred from the accumulator to the X-register, if required. The X-register is a four-stage register.

The add/subtract circuit is capable of adding the 4-digit numbers in the accumulator and X-register together and displaying the result on

the accumulator display. The result is then entered back into the accumulator. The add/subtract circuit can also subtract the contents of the X-register from the contents of the accumulator and display the difference on the accumulator display. In this case, also, the result is entered into the accumulator. With this general outline of the arithmetic unit, let us proceed with the logical analysis required for the design.

Earlier in this chapter, equations were derived for addition and subtraction.
These equations are:

For addition:
1st stage—

$$S = (A_u \cdot \overline{A_d}) + (\overline{A_u} \cdot A_d)$$
$$\underline{C} = \underline{A_u} \cdot \underline{A_d}$$
$$\overline{C} = \overline{A_u} + \overline{A_d}$$

2nd through 4th stages—

$$S = C_{pc} \cdot [(A_u \cdot A_d) + (\overline{A_u} \cdot \overline{A_d})] + \overline{C_{pc}} \cdot [(A_u \cdot \overline{A_d}) + (\overline{A_u} \cdot A_d)]$$
$$\underline{C} = (A_u \cdot \underline{A_d}) + \underline{C_{pc}} \cdot [(\overline{A_d} \cdot \underline{A_u}) + (A_d \cdot \overline{A_u})]$$
$$\overline{C} = (C_{pc} \cdot \overline{A_u} \cdot \overline{A_d}) + \overline{C_{pc}} \cdot [\overline{A_u} + \overline{A_d}]$$

5th stage—

$$S = C_{pc}$$

For subtraction:
1st stage—

$$D = (M \cdot \overline{S}) + (\overline{M} \cdot S)$$
$$\underline{B} = \overline{M} \cdot \underline{S}$$
$$\overline{B} = M + \overline{S}$$

2nd through 4th stages—

$$D = B_{pc} \cdot [(M \cdot S) + (\overline{M} \cdot \overline{S})] + \overline{B_{pc}} \cdot [M \cdot \overline{S}) + (\overline{M} \cdot \underline{S})]$$
$$\underline{B} = (\overline{B_{pc}} \cdot \overline{M} \cdot \underline{S}) + \underline{B_{pc}} \cdot (\overline{M} + \underline{S})$$
$$\overline{B} = (B_{pc} \cdot M \cdot \overline{S}) + \overline{B_{pc}} \cdot (M + \overline{S})$$

5th stage—

$$D = B_{pc}$$

If we examine the above equations for addition and subtraction for the first stage, and we let $A = A_u = M$ and $X = A_d = S$ (i.e., we let the statement for the augend in addition equal the statement for the minuend in subtraction, thereby allowing the augend and the minuend to be stored on the same set of switches with a separate device being used to distinguish between addition and subtraction), we can see the following similarities:

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

Thus, for the first stage, we see that addition and subtraction produce the same results if the augend is set equal to the minuend and the addend is set equal to the subtrahend. The following examples illustrate this point:

$$\begin{array}{cccc} 1 & 1 & 1 & 1 \\ +\ 0 & -\ 0 & +\ 1 & -\ 1 \\ \hline 1 & 1 & 0^* & 0 \end{array} \qquad \begin{array}{cccc} 0 & 0 & 0 & 0 \\ +\ 1 & -\ 1 & +\ 0 & -\ 0 \\ \hline 1 & 1^* & 0 & 0 \end{array}$$

If we set S = D = T, for the first stage:

$$T = (A \cdot \overline{X}) + (\overline{A} \cdot X)$$

is the required equation for both addition and subtraction.

The carry and borrow equations for the first stage are:

$$C = \underline{A} \cdot X \qquad \overline{C} = \overline{A} + \overline{X}$$
$$B = \overline{A} \cdot X \qquad \overline{B} = A + \overline{X}$$

Since there are no similarities here, it is necessary to write these equations in complete form, setting C = B and adding the statements ADD and SUB:

$$C = ADD \cdot (\underline{A} \cdot X) + SUB \cdot (\overline{A} \cdot X)$$
$$\overline{C} = ADD \cdot (\overline{A} + \overline{X}) + SUB \cdot (A + \overline{X})$$

Thus, the first stage equations can be mechanized by the circuit illustrated in Fig. 4-4. Notice that wafer D has been added to the display switch so that the content of switch A (accumulator) can be displayed on order from the control unit. This switch wafer allows the value of A

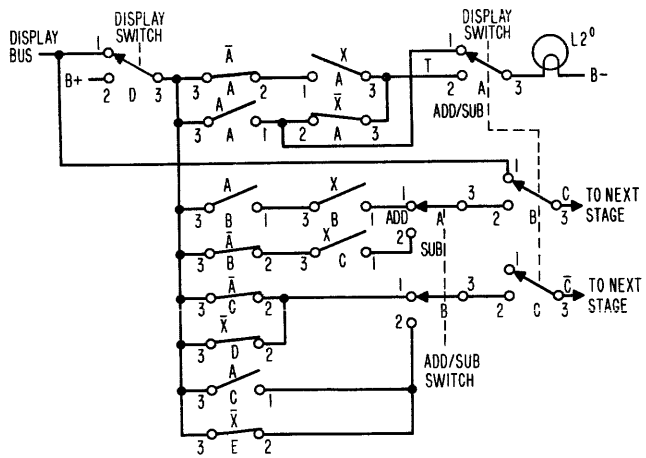


Fig. 4-4. The first stage of the arithmetic unit.

* Neglecting borrow or carry.

from wafer A of the A switch to be displayed through wafer A of the display switch on the 2^0 lamp when the DISPLAY order arrives from the control unit.

Examining the second through fourth stages in the same manner, one sees for the sum and difference:

$$T = C_{pc} \cdot [(A \cdot X) + (\overline{A} \cdot \overline{X})] + \overline{C}_{pc} \cdot [(A \cdot \overline{X}) + (\overline{A} \cdot X)]$$

For the carry and borrow:

$$C = ADD \cdot [(A \cdot X) + C_{pc} \cdot [(\overline{A} \cdot X) + (A \cdot \overline{X})]] + SUB \cdot [(\overline{A} \cdot X) + C_{pc} \cdot [(\overline{A} \cdot \overline{X}) + (A \cdot X)]]$$

$$\overline{C} = ADD \cdot [(\overline{A} \cdot \overline{X}) + \overline{C_{pc}} \cdot [(\overline{A} \cdot X) + (A \cdot \overline{X})]] + SUB \cdot [(A \cdot \overline{X}) + \overline{C_{pc}} \cdot [(A \cdot X) + (\overline{A} \cdot \overline{X})]]$$

These equations can be instrumented in the same manner as before, using the circuit of Fig. 4-5. Notice that the display signal is again passed to the next stage through the same wafer as C. In the fourth stage, however, it is not necessary to mechanize the $\overline{C}$ term. The C term

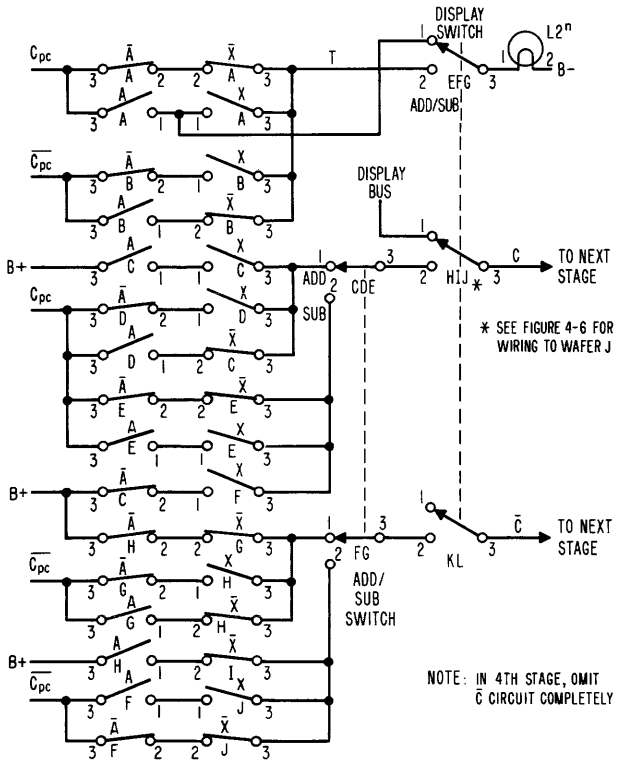


Fig. 4-5. The second through fourth stages of the arithmetic unit.

from the fourth stage is fed to the fifth stage as illustrated in Fig. 4-6.

Now, the add/subtract portion of our arithmetic unit is complete along with the X-register and four stages of the accumulator. Since it is necessary also to store a 2^4 bit due to an addition or subtraction, the fifth stage of the arithmetic unit must be equipped with a switch as illustrated in Fig. 4-6.

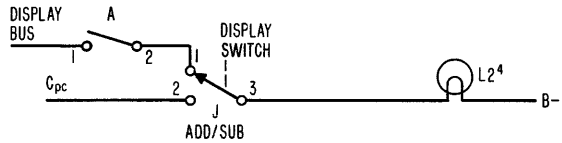


Fig. 4-6. The fifth stage of the arithmetic unit.

Now, the add/subtract and display switches on our arithmetic unit must be operated manually. When the control unit provides a visual indication to add or subtract, the operator must throw the display switch to the ADD/SUB position and then the ADD-SUB switch to the appropriate position. At all other times, the switch must be kept in the DISPLAY position.

At this point we have accounted for the add/subtract circuit, accumulator, accumulator display, and X-register of our arithmetic unit. It is necessary now to construct an extension register whose function is to handle the least-significant portion of numbers greater than four bits. This condition will arise due to the multiplication of two 4-bit numbers. The extension register and display need only comprise four independent switches wired to a lamp display as indicated in Fig. 4-7.

Wire and construct the arithmetic unit in accordance with the instructions contained at the end of this chapter.

Remember that this unit now requires the following rules of interconnection and operation in our overall computer scheme:

1. Upon visual instruction (to be provided by the control unit) the operator must throw the add/subtract and display switches to their appropriate position. Unless indicated otherwise by the

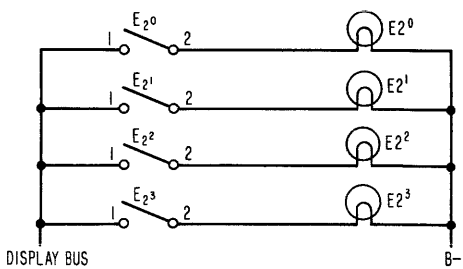


Fig. 4-7. The extension register.

control panel, the display switch is normally left in the DISPLAY position.

2. A display (ACC) wire from the control unit must be connected to a display input bus in the arithmetic unit and must provide B+ whenever it is required to display the accumulator's contents.
3. Entry of data to the extension register is a manual operation to be directed visually by the control unit.

CONSTRUCTION DETAILS—ARITHMETIC UNIT

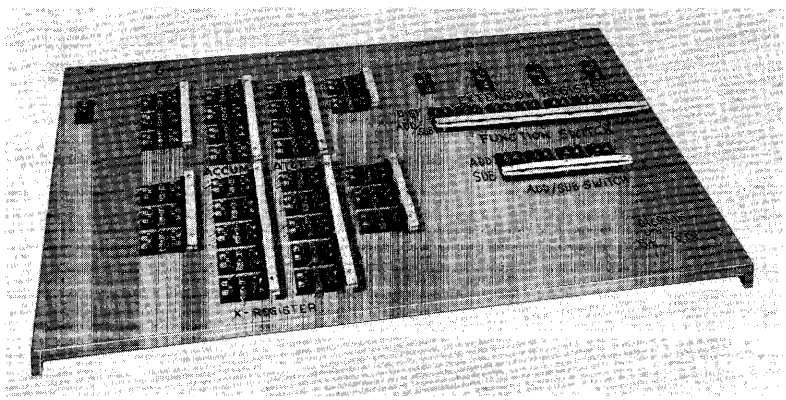


Fig. E-1.

COMPONENTS: *Chassis, switches, display circuit*

MATERIALS

1	3 ft. × 2 ft. × ⅛ in. composition board
2	½ × 11 × ⅛ in. composition board
4	½ × 9 × ⅛ in. composition board
6	½ × 7 × ⅛ in. composition board
6	½ × 5 × ⅛ in. composition board
2	½ × 3 × ⅛ in. composition board
1	4 ft. × 1 in. × 4 in. pine strip
1	6 ft. × 1 in. × 1 in. pine strip
39	DPDT switches
5	SPST switches
9	#48 or 41 lamps (2 v, .06 a)
	150 ft. insulated hook-up wire (20 gage)
144	½ in. wood screws (no. 4)
23	¾ in. machine screws (6-32)
29	nuts (6-32)
1	tin strip (large juice can or sheet)

SPECIAL TOOLS:

Tin snips
Drill (1/16, 3/32, ⅛, ⅜)

Chassis Construction

1. Mark the smooth side of the 3 × 2-ft. composition board with rules as shown in Fig. E-2.
2. Cut the 4-ft. 1 × 4-in. pine into two 24-inch lengths. Rip each length as shown in Fig. E-3 to obtain four angle supports. Leave two of these lengths at 24 inches and cut one of the others down to 22 inches. Discard the fourth length.

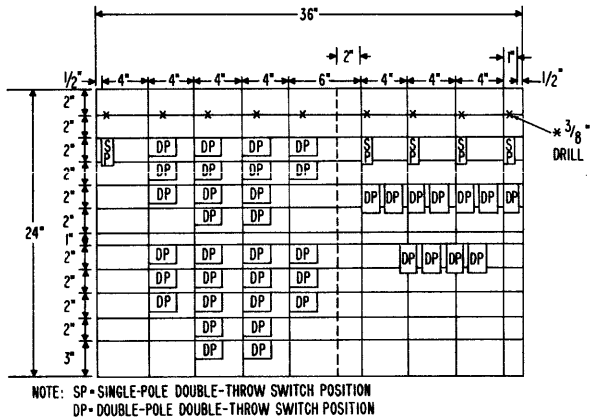


Fig. E-2.

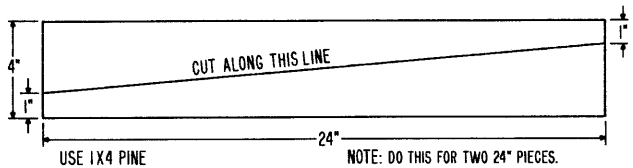


Fig. E-3.

3. Mark the edges and center of the 3 × 2-ft. board as shown in Fig. E-4. Drill 3/32-in. holes at each point marked.
4. Following Fig. E-5, clamp the angle supports to the panel and use the panel holes drilled in Step 3 as guides to drill holes into the support edges.
5. Attach the side and center supports to the panel with ½-in. wood screws.
6. Attach the two 34-in. 1 × 1-in. strips to the front and rear of the bottom panel using ½-in. wood screws.
7. On the panel face, use a DPDT or SPDT switch, whichever is necessary, as a guide to mark the location of switch-securing holes at each switch location. Be certain each switch will be positioned as indicated in Fig. E-9.
8. Start a drill hole at each mark made in Step 7 and drill through the panel with a 3/32-in. drill.
9. At each lamp position indicated in Fig. E-2 start a drill hole and drill through the panel with a ⅜-in. drill.

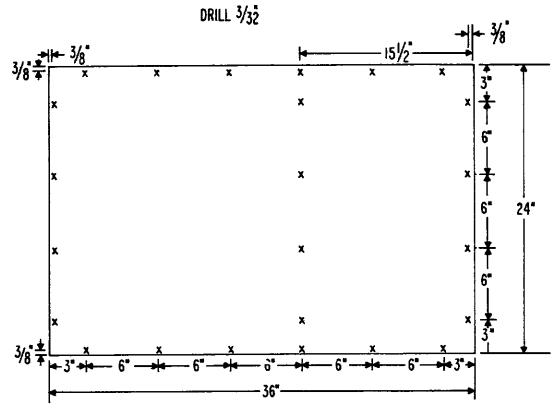


Fig. E-4.

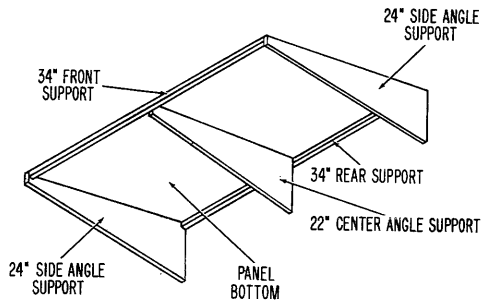
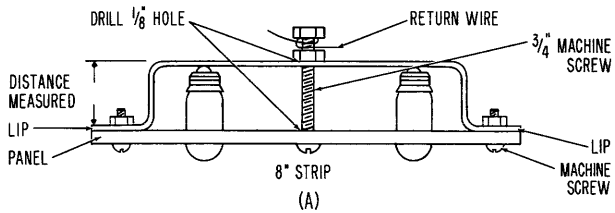


Fig. E-5.



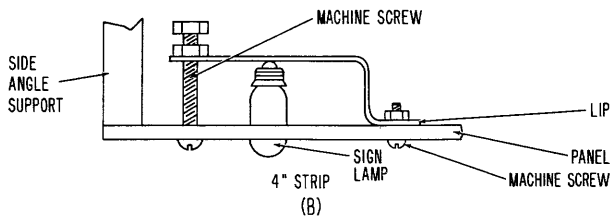


Fig. E-6.

10. Test each of the holes drilled in Step 9 to see that the envelope of a lamp will protrude approximately ⅛ inch above the panel face when the lamp is placed in the hole. File or sand each hole for a snug bulb fit.
11. Attach all switches with ½-in. wood screws.
12. Mark each switch next to each of its terminals. Start a drill hole and drill through the panel at each of these points with a ⅛-in. drill.

Power Terminal Strip

1. Drill three ⅛-in. holes in the lower-right corner of the panel as indicated in Fig. E-9.
2. Place ¾-in. machine screws in the holes drilled in Step 1 and attach 3 nuts to each screw from the bottom of the panel.

Display Circuit

1. Construct base terminal connectors for the 9 lamps as detailed in the construction procedures for the encoder. Test each lamp.
2. Cut tin strips for the lamp bases, each ¾ in. wide: one 4-in. strip; four 8-in. strips. The 4-in. strip is for the sign lamp, the 8-in. strips are for pairs of the remaining 8 lamps.
3. Buff the tin strips so they have good surface continuity.
4. Set the lamps in their holes with the terminal connectors pointed towards the foot of the panel. Measure the distance from the panel to the lamp contact.
5. Referring to Fig. E-6, bend each tin strip as shown.
6. Drill a ⅛-in. hole in the center of each 8-in. strip.
7. Drill a 3/32-in. hole at each end of each 8-in. strip.
8. Place the 8-in. strips in position, one by one, drilling holes in the panel using the center and end holes as guides.
9. Secure the 8-in. strips in the center with ¾-in. machine screws and two nuts. Secure them at the ends with ½-in. wood screws. Tighten the ¾-in. machine screw to ensure good contact between the strip and the lamp.
10. Secure the 4-in. strip as indicated in Fig. E-6(b), following the procedure detailed above.
11. Figure E-7 shows the final placement of the strips and the RTN wiring. Wire together the securing nuts of each tin strip as shown in this figure.

Wiring

1. Turn the panel over and mark the switches on the underside as shown in Fig. E-8.
2. Following the wiring lists given in Appendix B, connect the switch terminals and lamps, following the procedure outlined in the wiring section of the decoder construction details. Check your wiring after every few connections. When connecting wires to the lamps, crimp the lamp connectors around the wire.

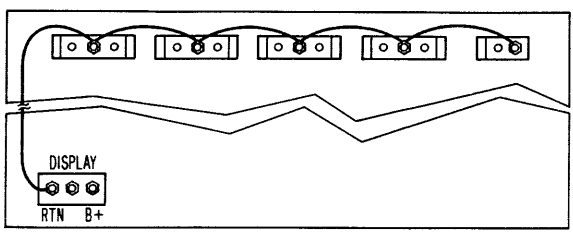


Fig. E-7.

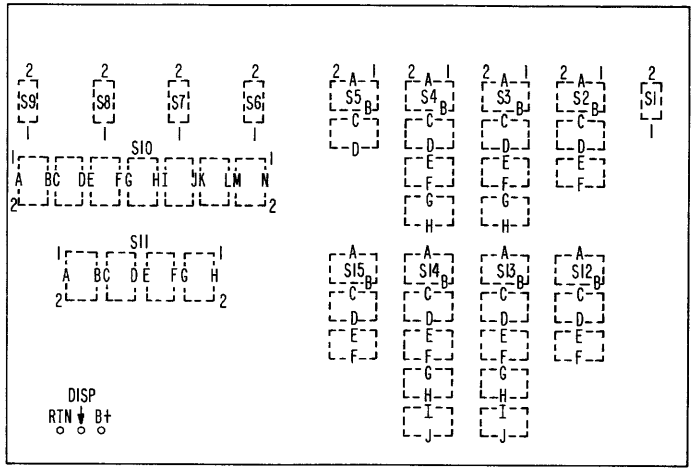


Fig. E-8.

3. Using ½-in. strips and ¾-in. machine screws and nuts, attach the common switch bars to the switch handles. Chart A shows which switches are to be connected in this fashion.
4. Connect the B+, Display, and RTN to the junction box.
5. Label the unit as shown in Fig. E-9.

Arithmetic Unit Checkout Procedure*

1. Connect one 1½-volt flashlight battery between the B+ and RTN terminals. Connect a second 1½-volt flashlight battery between Display and RTN.
2. Set all switches to 0 and set the Function Selector to Display.
3. In order, set the 2^4 (Sign), 2^3, 2^2, 2^1, and 2^0 switches of the Accumulator and the 2^{-1}, 2^{-2}, 2^{-3}, and 2^{-4} switches of the Extension Register to 1. Note that each associated lamp is lighted.
4. Set the Function Selector at Add/Sub. All lamps should remain lighted except the 2^4 (Sign) lamp.
5. Set the Add/Sub switch at Sub.
6. In order, set the 2^0, 2^1, 2^2, and 2^3 switches of the X-Register at 1. Note that the lamp corresponding to each switch goes out when the switch is set to 1.
7. In order, set the 2^3, 2^2, 2^1, and 2^0 switches of the Accumulator at 0. Note that the lamp corresponding to each switch goes on when the switch is set to 0, but goes out when the next switch is set to 0. Note that the Sign lamp is on.
8. In order, set the 2^3, 2^2, 2^1, and 2^0 switches of the X-Register to 0. Note that the lamp corresponding to each switch lights when the switch is

* *Note:* This checkout procedure represents a reasonable check of the arithmetic unit. However, it does not cover all possibilities for adding and subtracting binary numbers. A complete check would consist of the addition of every possible combination of 4-bit binary numbers and the subtraction of all such combinations. This type of check is left to the reader should he wish to follow it out.

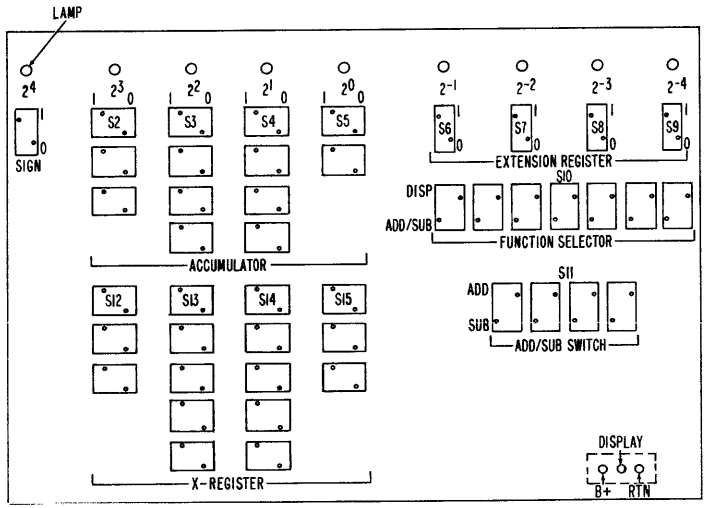


Fig. E-9.

set to 0 and remains lighted when switching the next switch. Note that after switch 2^0 is set, all lamps are extinguished.

9. Set the Add/Sub switch to Add.
10. In order, set the 2^0, 2^1, 2^2, and 2^3 switches of the X-Register to 1. Note that the lamp corresponding to each switch is lighted.
11. In order, set the 2^0, 2^1, 2^2, and 2^3 switches of the Accumulator to 1. Note that after 2^0 is set, only the 2^4 lamp is lighted. After 2^1 is set, the 2^1 and 2^4 lamps are lighted. After 2^2 is set, the 2^1, 2^2, and 2^4 lamps are lighted. After 2^3 is set, all lamps are lighted except 2^1.
12. In order, set the 2^3, 2^2, 2^1, and 2^0 switches of the X-Register to 0. Note that after 2^3 is set, the 2^4, 2^2, and 2^1 lamps are lighted. After 2^2 is set, the 2^4 and 2^1 lamps are lighted. After 2^1 is set, only the 2^4 lamp is lighted. After 2^0 is set, all lamps are lighted except 2^4.

Chapter 5
STORAGE DEVICES

One of the reasons why computers are called "electronic brains" is their ability to store and "remember" instructions and data. Because it has a memory, the computer is released from the common calculating machine's dependence on human, external, step-by-step control. Once it has received its instructions and the proper data, the computer can work without interruption and take advantage of the high computational speeds that have made it such an invaluable tool.

Memory units take several forms, according to their function, and are found in all parts of the computer. Some are used continuously and others are referred to only at specific moments during a calculation. Because they receive and hold information, these memory units are also called *storage devices.* A computer's computational ability is related directly to the amount of information it can store, its *storage capacity,* and the speed with which it can obtain information from these storage devices.

There are four types of storage employed in present-day computers: the *register store,* the *internal memory,* the *auxiliary internal memory,* and the *external memory.* The frequency of use of these units is dependent on the ease with which data can be recovered from them.

Register stores are used continuously during computer operation. Partial answers, digits being operated on, and actual calculations are registered in these devices. Essentially, they are two-position switches that store 1 in one position and 0 in the other. The double-throw switches used in our arithmetic unit represent this type of storage device. Because information passes quickly through them they are considered to be short-term storage units.

The internal memory is the major storehouse in the computer. It holds all of the information supplied by the programmer: both the instructions to be followed, in sequence, and the data to be used. The auxiliary memory store, also internal, is used to back up the internal memory in the event that its storage capacity is not great enough for the program and data being inserted. The magnetic drum form of this unit is not as rapid a transmitter of material as the magnetic core form. Although the speed differential is in millionths of a second, which may be insignificant to us, this is quite meaningful in most computer operations.

The external memory store is physically outside the computer and serves as a library of programs and less often-used data, or as a second auxiliary memory. Such storage devices as punched cards, perforated

tape, magnetic tape, and magnetic disks are used in external memory units.

Our primary consideration in this chapter will be the internal memory stores—the magnetic drum memory and the core memory—as these are the two most common types of memory store used in computers today and are used in our computer.

BASIC CONCEPTS

The storage capacity sets a limit on the range of computer calculations and the number of digits a computer can handle. This fixes the number of bits the computer can process at any given moment. In each computer this fixed number of bits is called the computer's *word length.* (Computer word length can be variable, but for most computers it is fixed.) For example, in a computer that uses 7-bit words, the number $(25)_{10}$ would appear as $(0011001)_2$, the number $(9)_{10}$ would appear as $(0001001)_2$, or the number $(2)_{10}$ would appear as $(0000010)_2$.

When a computer is designed, its word length is set by determining the largest number the computer will handle to achieve the required accuracy. For example, if the computer must handle integers of the order of magnitude 10^4 (10,000), the number of bits required is 14 ($10^4 \cong 2^{13}$) plus a bit to indicate sign (+ or −) for a total of 15 bits.

In some cases, the requirements for instruction word lengths differ from those for data words (instruction words are described in the following paragraphs). The instruction word length is determined by the number of different instructions and addresses required by the design as well as the overall design criteria. If, for example, there are 32 different instructions (2^5) and 9,000 possible addresses (approximately 2^{13}), the instruction word requires a minimum of 20 bits. The instruction word may be less than the data word length; or it may be set equal to the data word length, if possible, for convenience and ease of design.

COMPUTER WORDS

Basically, the computer uses two different word types. The first type, a straight numerical or data word, is a binary number. The second type is an instruction word.

The *instruction word* is that part of a program that causes the computer to perform a particular operation. It has two parts: the *order* and the *address.* In Chapter 1 we learned that a computer has an instruction repertoire that consists of such orders as Add, Subtract, etc. Each order is represented in the instruction word by a binary code that the computer is designed to recognize. Each order usually is directed to data stored in a computer memory section. The data are designated by a numerical

code indicating their location in the memory section. This code is called an *address.*

The complete instruction word, therefore, breaks down as follows:

Order	*Address*
Add A to	B
Subtract A from	B
Store A in	B

Let us assign a word length of 16 bits to a computer with a 20 order instruction repertoire and a 1500 word memory capacity. The instruction word would have the format illustrated below:

$$\underbrace{2^{15}\,2^{14}\,2^{13}\,2^{12}\,2^{11}}_{\textit{Order}}\;\underbrace{2^{10}\,2^{9}\,2^{8}\,2^{7}\,2^{6}\,2^{5}\,2^{4}\,2^{3}\,2^{2}\,2^{1}\,2^{0}}_{\textit{Address}}$$

The five most-significant bits contain the order code and the remaining bits contain the address code.

Every storage device capable of storing a word is assigned an address. Hence, if we have 1500 storage devices, the address code will range from 00000000001 to 10111011100. Therefore, if the order "Add A to" is represented by the code 01010, the instruction word

0101000110110010

means "add the contents of A to the contents of address 434." A series of instruction words may be written to perform any operation by keeping track of the addresses in which the pertinent data are stored.

THE CORE MEMORY

The core memory unit is composed of many small magnetic cores. Each core is a doughnut-shaped ring whose magnetic characteristics are used to store bits. Hundreds of thousands of these cores are used in the average computer core memory.

Bits are stored in a core by changing its magnetic state. The magnetic state of the core is affected by magnetic lines of force or flux in the core material. This flux polarizes the molecules in the material and determines the direction of magnetization in the core. The direction of magnetization can be reversed.

In the magnetic core this is accomplished by wires coiled around the core—the input and sense windings. Electrical current passing through the input coil will generate a magnetic field that affects the flux in the core. If the generated field is strong enough it will reverse the direction of magnetization in the core. By assigning one magnetic state the value of 1 and the other the value of 0 (Fig. 5-1), these cores can be used as register stores and, in combination, internal memories. The sense winding, when energized, will sense the state of the core and pass

Fig. 5-1. Magnetic cores are used as memory stores.

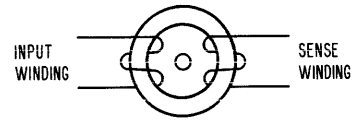


Fig. 5-2. The core is read into and out of on sense windings.

on an electrical pulse indicating what it is. In this fashion the core can be read into on the input winding and read out of on the sense winding (Fig. 5-2).

Combinations of cores can store computer words. One computer word in memory is wired as in Fig. 5-3. A signal that is strong enough is introduced on the word line. If a core is in the 0 state the input signal will have no effect. If, however, it is in the 1 state, the input signal causes it to change to the 0 state. The resultant change in flux imparts a signal to the sense line that is amplified and interpreted as a binary 1. When no change occurs, the sense winding develops no signal and the core's state is read as a binary 0.

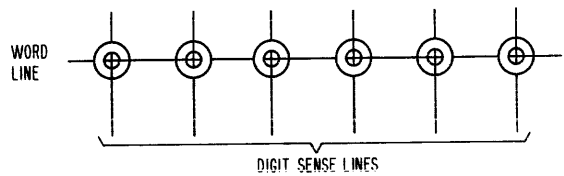


Fig. 5-3. One computer word in memory.

The core memory capable of storing many complete computer words is a series of planes stacked one above the other, as in Fig. 5-4. Each plane has an assembly of cores that equals the number of words in a stack of planes. The number of planes in a stack is generally equivalent to the number of bits in a word. Each plane is wired in such a way that only one core is addressed at any particular time. The entire stack is wired so that the same core location in each plane is addressed at any particular time. In this fashion, one computer word can be read in or out at one time.

The core memory of our computer has a capacity of five 4-bit words. It will use, therefore, the equivalent of 20 magnetic cores in four planes of five cores. As we are eliminating the automatic feature of the computer, memory switches will be used instead of magnetic cores. This memory unit will store 20 bits (5×4 bits). Thus, each core may be loaded with any 4-bit word from 0000 to 1111. The contents of a single address will be displayed upon order. Construction details are given at the end of the chapter.

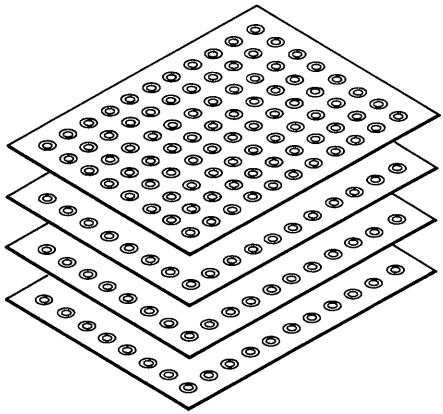


Fig. 5-4. A core memory capable of storing many words.

THE DRUM MEMORY

The drum memory is another storage device that alters the magnetic state of a medium to record binary 1's and 0's. In this case, the magnetized medium is a block of parallel strips or tracks that ring the circumference of a cylinder or drum (Fig. 5-5). A magnetic head similar to that used on a tape recorder reads data into the track as current switches back and forth through its windings (Fig. 5-6). Each current reversal alters the magnetic state of the head and the portion of the track immediately beneath it. The direction of current passing through the head coils affect the direction of magnetic flux.

When the magnetized portions of recorded track pass below another magnetic head, the read head, the magnetic lines of flux create a current in the head's windings. The direction of flow of the induced current indicates whether a 0 or 1 is recorded on that portion of the track.

Each portion of the track that contains a bit is called a *cell.* All of the cells in a line parallel to the memory drum's axis constitute a slot or

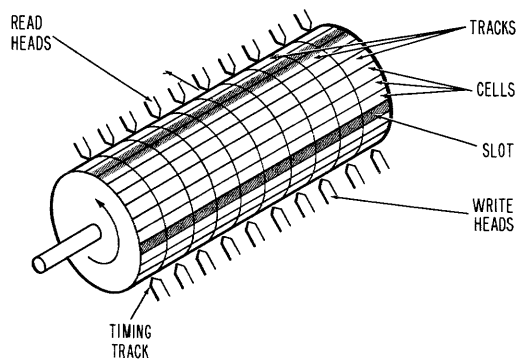


Fig. 5-5. A memory drum.

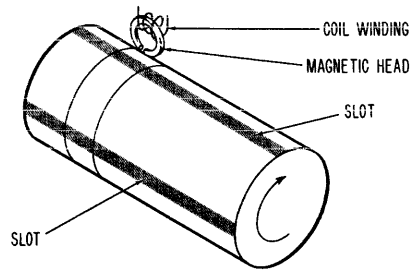


Fig. 5-6. A magnetic head "reads" or "writes on" the drum track.

sector. Often a slot will contain one computer word, so the number of cells in it is equal to the number of bits in a word for that computer.

The drum storage capacity depends on its size and speed of rotation. Drum sizes vary from 15 to 400 tracks with a rotational speed of from 120 rpm to 75,000 rpm. The number of bits per track inch varies from 200 to 300 bits per inch. In these ranges, capacities run from 25,000 to 15,000,000 bits.

The magnetic drum used in our computer has 29 tracks and will store 65 computer words. Its rotational speed is controlled by the computer operator. This drum reads out in parallel; i.e., the bits comprising a computer word in each slot will read out simultaneously. Construction details for this unit follow those for the core memory at the end of this chapter.

CONSTRUCTION DETAILS—CORE MEMORY

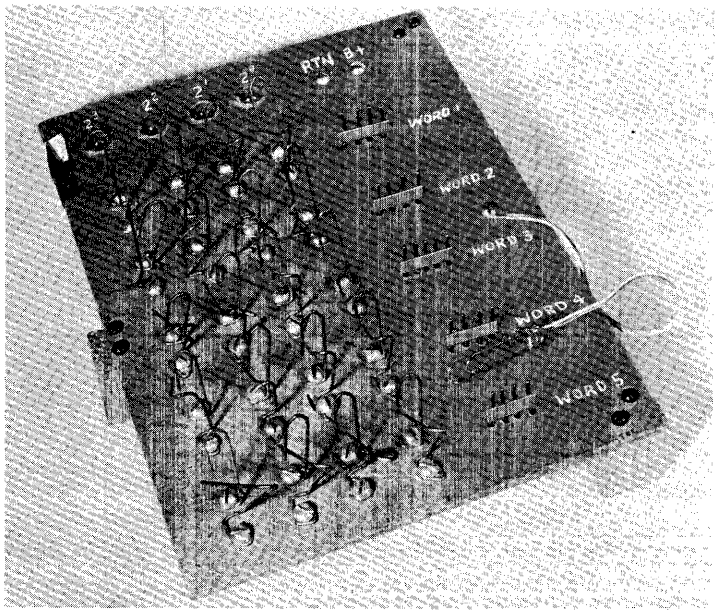


Fig. F-1.

COMPONENTS: *Chassis, switches (paper clip), display circuit*

MATERIALS

Chassis:

1	8 × 6 × ⅛ in. composition board
3	¾ × ½ × ½ in. supports (scrap wood)
6	½ in. wood screws (no. 4)

Display Circuit (including paper-clip, SPST switches):

44	paper clips
42	½ in. machine screws (6-32, roundhead)
42	nuts (6-32)
4	#48 or 41 lamps (2 v, .06 a)
1	6¼ × ½ in. tin strip (can or sheet)
	3½ ft. uninsulated hookup wire (20 gage)
	6 ft. insulated hookup wire (20 gage)
	Adhesive tape

SPECIAL TOOLS:

Tin snips
Drill (3/16, ⅛, 3/32)

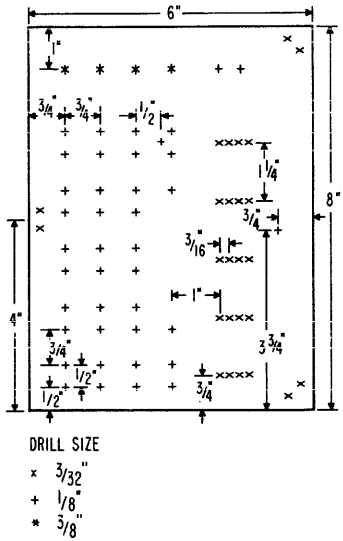


Fig. F-2.

Chassis Construction

1. Mark and start drill holes indicated in Fig. F-2 on the 8 × 6½ × ⅛ in. board.
2. Drill holes through the board, following the key in Fig. F-2. The ⅜-in. holes will hold the display lamps. The ⅛-in. holes will hold machine screws that secure paper-clip SPST switches. The 3/32-in. holes will hold the wood screws securing the chassis to its supports and the contact wires for each row of switches.
3. Test each ⅜-in. hole with a lamp bulb and file, if necessary, to fit bulb snugly with approximately ⅛ in. to ¼ in. of the glass envelope protruding above the surface of the panel.
4. Attach the three chassis supports, holding them in position and drilling through the 3/32 in. holes in the panel. Secure the supports with the ½-in. wood screws.

Switches

1. These switches were first built for the truth evaluator (Chapter 3). Bend 20 paper clips as in C-3(a). Bend the 20 other paper clips as in Fig. C-3(b).
2. Cut each paper clip for each configuration as indicated in Fig. C-3. The ½-in. section will be crimped around a machine screw to secure the switch parts on the panel in an upright position.
3. Secure the switch parts to the panel top with ½-in. machine screws. Put the switch arms in the first vertical row on the left side of the panel (lamp holes to the right) with the arms pointing toward the lamp holes. Place the contacts so that the open loop is towards the top of the panel. Alternate between arms and contacts as you move to the right. As each switch part is secured with a ½-in. machine screw, crimp the paper clip around the screw and tighten it.

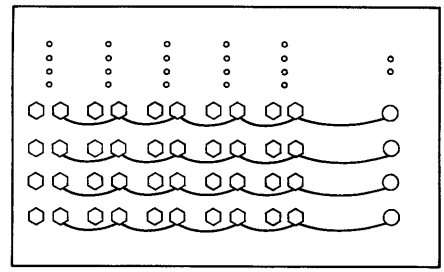


Fig. F-3.

4. When switches are in place, cut each arm so that it extends past the contact about ⅛ in.
5. Cut each contact so that it is about ½ in. long on the open loop.
6. Bend the switch arms to the outside of the contact (toward the top of the panel) so that there is enough tension to spring the arm away from contact when it is not locked in place. The make or break motion is similar to the opening or closing of a safety pin.

Switch Wiring

1. When all switches are secure, turn board upside down with lamp holes to the right. Connect each horizontal row of five contact screws with an 8-in. length of uninsulated wire, as in Fig. F-3. Loop the wire around each contact screw and secure the loop with a nut, terminating the run at the lamp hole opposite that row.
2. Cut the insulated wire into the following lengths: five 4-in. lengths, five 3½-in. lengths, five 3-in. lengths, and five 2½-in. lengths. Strip ¼ in. of insulation from one end of each wire and ½ in. from the other end.
3. With the board upside down, attach the 2½-in. lengths to each arm screw on the bottom horizontal row as in (A) of Fig. F-4. The end with ¼ in. of insulation stripped from it is connected to the screw. Secure this end with a nut. Thread the other end of each lead through the bottom-most 3/32-in. hole, as shown in the figure.
4. Repeat Step 3 for the 3-in., 3½-in., and 4-in. leads, as shown in (B), (C), and (D), respectively, of Fig. F-4. Part (E) of the figure indicates how each vertical row of arm screws should look with all wires in place.
5. Turn the board right side up. Pull the arm contact wires up through the 3/32-in. holes so that they are taut. Trim each set of four wires evenly so they extend approximately ½ in. above the panel.

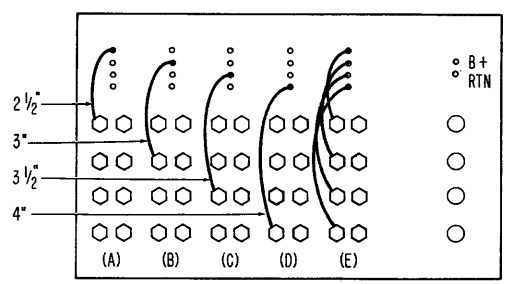


Fig. F-4.

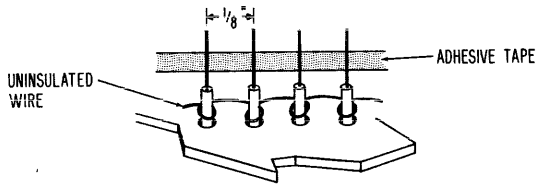


Fig. F-5.

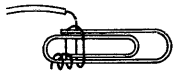


Fig. F-6.

6. Take four short pieces of uninsulated wire and loop one around each set of four wires to secure them in place (Fig. F-5).
7. Position the loose ends of wire approximately ⅛ in. apart and bind them with a thin strip of adhesive tape, as in Fig. F-5.
8. Strip 1½ in. of insulation from a 10-in. length of insulated wire. Strip ¼ in. of insulation from the other end of this wire.
9. Wrap ½ in. of exposed wire around one side of the curved end of a paper clip, as shown in Fig. F-6. Wrap the remaining exposed wire around the entire end of the clip and secure it in place with adhesive tape.
10. Fasten the other end of this wire to the panel with a ½-in. machine screw in the ⅛ in. hole at the top of the panel, to the right of the lamp holes. Secure this screw underneath the panel with a nut. Label this terminal B+.

Lamp Display

1. Fit each lamp with a paper clip base connection. (See Encoder Construction Details for procedure.) Check each bulb to see that it works properly.
2. Fit bulbs into ⅜-in. holes with connector facing the switches.
3. Connect the uninsulated leads from the contact screws to the lamp at the end of each horizontal row of switches. Crimp each lead securely in the bend of the paper clip.
4. Bend the 6¼ × ½ in. tin strip as shown in Fig. F-7 to secure it to the panel's edge. Drill through the other end using the ⅛-in. hole in the panel as a guide.
5. Secure the tin strip to the panel with a ½-in. machine screw, making

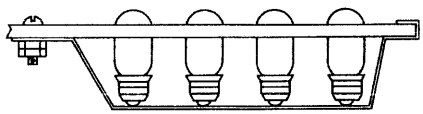


Fig. F-7.

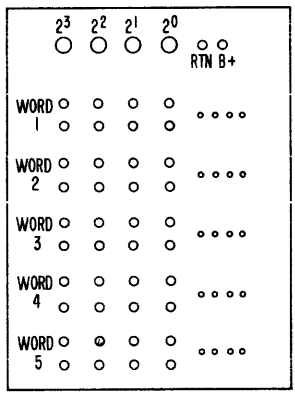


Fig. F-8.

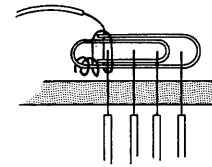


Fig. F-9.

sure that the lamp contacts are firm. Place a nut on each screw underneath the panel. Label the terminal on the right side RTN.

6. Label the panel top as shown in Fig. F-8.

Core Memory Operation

1. Close switch for a 1, open for a 0.
2. To read memory, use clip lead and simultaneously connect all four exposed wires for address desired. See Fig. F-9.

Core Memory Checkout Procedure

1. Connect a 1½-volt flashlight battery between the B+ and RTN terminals.
2. Close all toggle switches.
3. Connect the switch for word 1 and check to see that each lamp lights.
4. Open the 2^3 toggle of word 1 and note that the 2^3 lamp is out.
5. Open the 2^2 toggle of word 1 and note that the 2^2 lamp is out.
6. Open the 2^1 toggle of word 1 and note that the 2^1 lamp is out.
7. Open the 2^0 toggle of word 1 and note that the 2^0 lamp is out.
8. Repeat Steps 3 through 7 for words 2 through 5.

CONSTRUCTION DETAILS—DRUM MEMORY

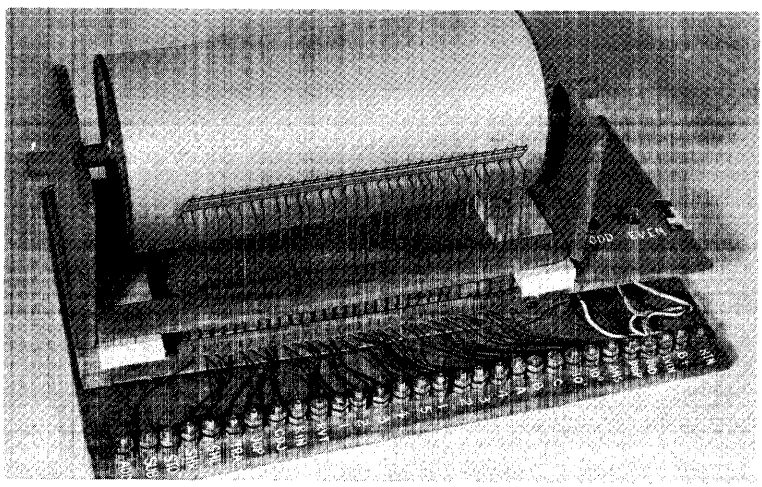


Fig. G-1.

COMPONENTS: *Sub-base and supports, drum, paper-clip contacts, base, display*

MATERIALS

Sub-base, supports, drum:

1	tin can (large juice size)
2	4 × 3 × ¼ in. composition board
1	5 × 12 × ¼ in. composition board
2	2 × 1½ × ⅛ in. composition board
2	¾ × ¾ × 9 wood blocks
1	12 in. dowel (⅜ in. diameter)
8	½ in. wood screws (no. 4)
2	½ in. molly bolts

Paper-clip contacts:

28	paper clips
2	1 × 3 × ¼ in. wood blocks
1	9 × ½ × ⅛ in. composition board
1	9 × ¾ × ⅛ in. composition board
8	½ in. wood screws (no. 4)
	10 ft. insulated hook-up wire (20 gage)
	adhesive tape (½ in. wide)
	wood glue

Base and display:

1	6 × 12 × ⅛ in. composition board
1	2½ × 1½ × ⅛ in. composition board
4	wood supports for lamp chassis (¾ × ¾ × ¾)
1	tin strip (4 × ¾ in.)
28	½ in. machine screws (6-32)
56	nuts (6-32)
6	½ in. wood screws (no. 4)
2	#48 or 41 lamps (2 v, .06 a)

SPECIAL TOOLS:

Tin snips
Drill (1/16, 3/32, ⅛, ¼, ⅜)
Emery cloth or sandpaper

Sub-base and Side Support Construction

1. Drill ¼-in. holes through each side support as indicated in Fig. G-3. The 4 × 3 × ¼-in. composition boards are the side supports.

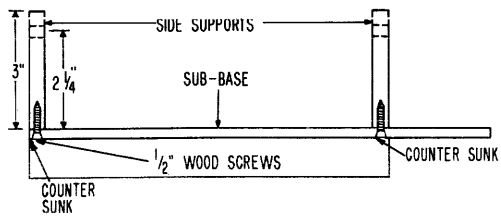


Fig. G-2.

2. Notch the side supports as shown in Fig. G-3.
3. Clamp one support to the left side of the 5 × 12 × ¼-in. sub-base and drill through from the bottom of the sub-base up into the side support with a ⅛-in. drill ½ in. in from the top and bottom edges of the sub-base. Countersink the drilled holes. Secure the support to the sub-base with ½-in. wood screws. See Fig. G-2.
4. Measure the length of the can to be used as the drum memory. Add 2 inches to this figure. The length obtained is the distance the second support should be from the first.
5. Clamp the second support at the proper distance from the first, as determined in Step 3. Drill through the sub-base into the support, as in Step 2, countersink the drilled holes, and secure the support to the sub-base with ½-in. wood screws.
6. Stand the sub-base on the ¾ × ¾ × 9-in. wood blocks with the 9-in. length under both side supports, one to the rear of the sub-base, the other to the front.
7. Drill through the sub-base into the wood blocks from the top about ¼ in. inside the side supports. Countersink the drilled holes.
8. Attach the sub-base to the wood blocks with four ½-in. wood screws.

Memory Drum Construction

1. Remove the paper label, if any, from the can to be used as the drum. Clean off any glue. If the can is painted, use sandpaper or emery cloth to remove all paint from the surface.
2. Puncture or drill a ⅜-in. hole in the center of each end of the drum. *Note:* It is important that these holes be exactly in the center, for, as the drum revolves, a constant pressure must be exerted on the read heads. Otherwise, it will be most difficult to adjust the head tension properly. To find the centers of the ends, do the following:
   (a) Notch the edge of the can slightly and sharply.
   (b) Stretch a length of cord across the diameter of the can, holding one end of the cord in the notch made in Step (a).
   (c) Using the notch as a pivot, swing the string back and forth over the opposite edge of the can and determine at which point on the can the string is longest. Notch the edge of the can at that point.
   (d) Using a straightedge, score a line across the can's end from one notch to the other.

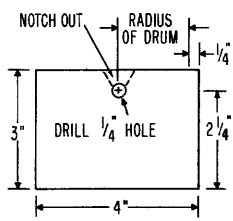
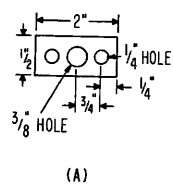


Fig. G-3.



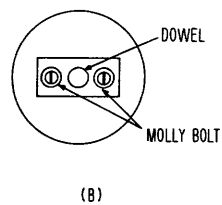


Fig. G-4.

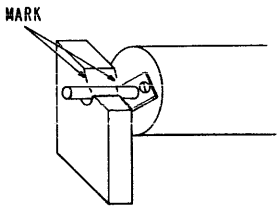


Fig. G-5.

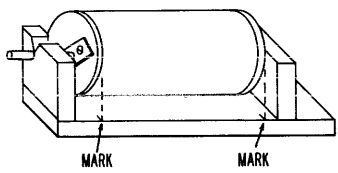


Fig. G-6.

(e) The center of the can is at the center of the line scored in Step (d). Use an awl and dimple the center point before drilling.

3. Drill a ⅜-in. hole through the center of each of the two 2 × 1½ × ⅛-in. pieces of composition board. Drill the ends of these boards as shown in Fig. G-4(a).
4. Place the boards drilled in Step 3 on each end of the drum so that their ⅜-in. holes are over the drum centerholes. Using the two other holes in the boards as guides, drill through the ends of the drum.
5. Attach the boards drilled in Steps 3 and 4 to each end of the drum with molly bolts (Fig. G-4(b)).
6. Run the 12-in. dowel (⅜-in. diameter) through the drum so that the projecting ends are of equal length. Use white glue to secure the dowel to the pieces at each end of the drum.
7. Lay the projecting dowel ends on the side supports and mark the dowels as in Fig. G-5.
8. With sandpaper, trim the dowels between the marks made in Step 7 until they fit in the hole drilled in the supports.
9. With the drum in place, mark the sub-base where the ends of the drum are, as in Fig. G-6.
10. Remove the drum from the supports.

Drum Contacts Construction

1. Straighten out 29 paper clips and cut each one so that it is approximately 4½ in. long.
2. Bend each clip to the configuration shown in Fig. G-7.
3. On the front edge of the sub-base drill a row of 29 1/16-in. holes to the dimensions indicated in Fig. G-8, centered within the drum end marks.
4. Mark and drill 29 1/16-in. holes on the 9 × ¾ × ⅛-in. and the 9 × ½ × ⅛-in. composition boards. These holes should line up, from end to end, with those drilled in the sub-base in Step 3.

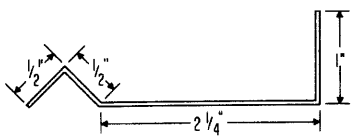


Fig. G-7.

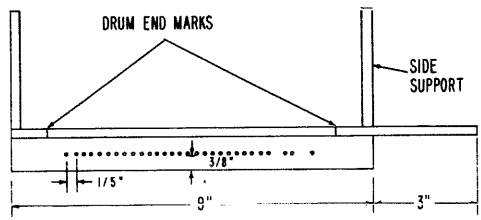


Fig. G-8.

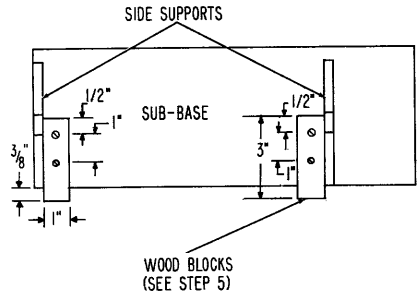


Fig. G-9.

5. Mark and drill the 1 × 3 × ¼-in. wood blocks as shown in Fig. G-9. Place these blocks inside each support, with their front ends extending ⅜ in. beyond the front of the base. Using the drill holes in the blocks as guides, drill 1/16-in. holes in the base and secure the blocks with ½-in. wood screws.
6. Insert the paper clips in the holes in the 9 × ¾-in. strip drilled in Step 4. See Fig. G-10(a).
7. Along the top of the clips, place two strips of ½-in. adhesive tape, sticky side to sticky side, with the clips in between. See Fig. G-10(b).
8. Cut 29 4-in. lengths of hook-up wire. Strip ¾ in. of insulation from one end of each wire and ½ in. of insulation from the other end of each wire.
9. Wrap the ends with ¾ in. of insulation stripped from them tightly around the 1-in. end of each clip. See Fig. G-11(a).
10. Insert the clip end with the wire on it in the hole on the edge of the sub-base as in Fig. G-11(b). Smear a little white glue on the wire when it is almost completely inserted to ensure a good fit. *Caution:* Be sure the glue is only at the front. Do not smear the entire end. If glue gets between the wire and paper clip, there will be no electrical contact.
11. Feed each wire, in order, through the corresponding hole in the 9 × ½-in. strip. Attach the strip to the sub-base by drilling 1/16-in. holes

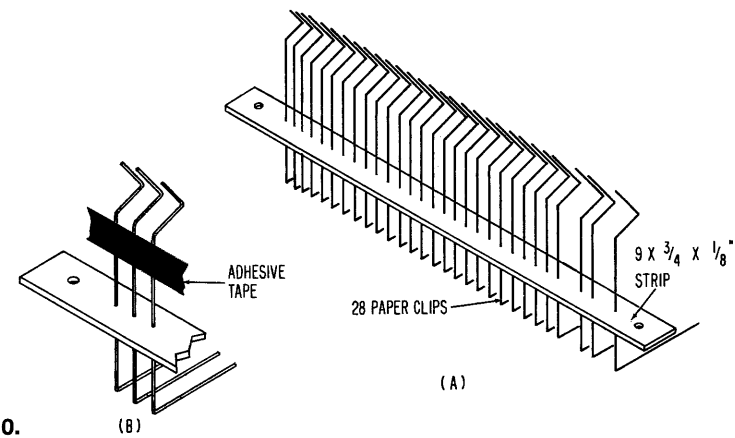


Fig. G-10.

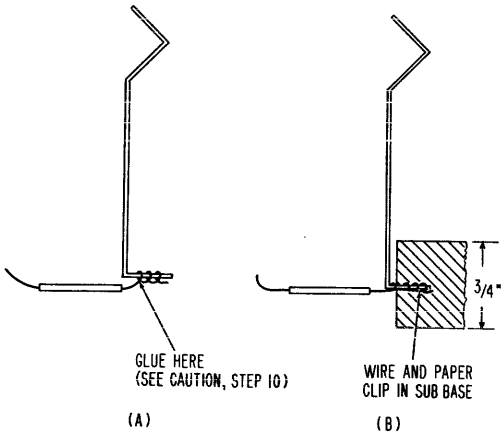


Fig. G-11.

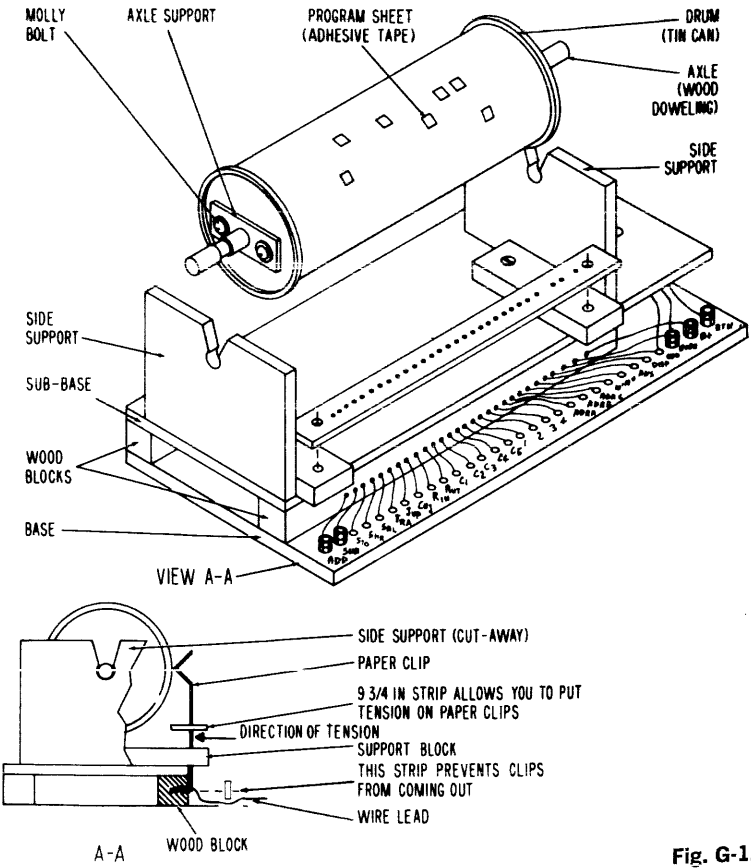


Fig. G-12.

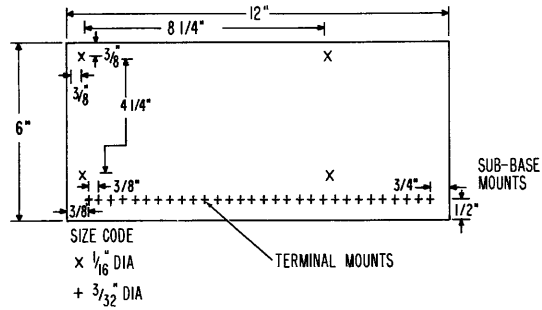


Fig. G-13.

through the strip and base at each end. Attach the strip to the sub-base with ½-in. wood screws. This strip prevents the clips from pulling out of the base.

Establishing Contact Tension

1. Replace the drum on the supports.
2. Push the 9 × ¾-in. strip against the clips and line up each contact point as best as possible. See Fig. G-12.
3. Slide the strip toward the drum along the supporting block until sufficient tension is obtained on all contacts against the drum. Secure the strip to the supporting blocks with ½-in. wood screws.

Base and Lamp Display Construction

1. The 7 × 12 × ⅛ composition board is the base for this unit. Mark and drill it as indicated in Fig. G-13.
2. Place ½-in. machine screws from the bottom in each of the 3/32-in. holes drilled in the base in Step 1 and place two nuts on each screw. Do not tighten the nuts.
3. Place the sub-base on the base with the row of terminal screws in the front. Using the 4 holes drilled at the sides of the base as guides, drill 4 holes into the sub-base. Secure the base to the sub-base with ½-in. wood screws.

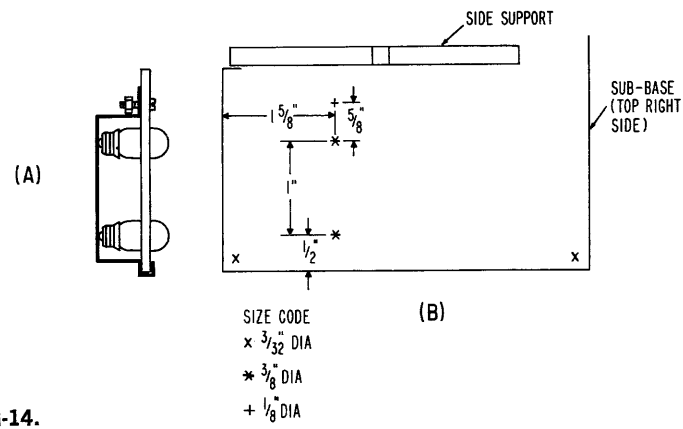


Fig. G-14.

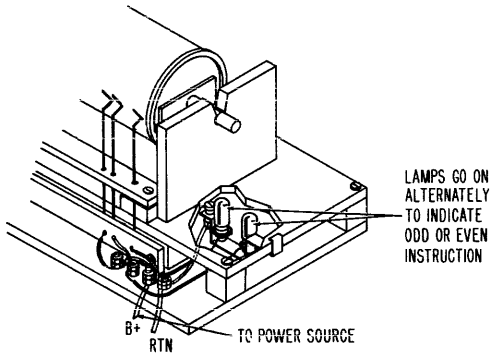


Fig. G-15.

4. Connect the wires from the paper clips to the corresponding terminal screws by tightening the wire under the bottom nut. Note that the terminals are slightly spread out. Therefore, start at one end first and work across.
5. The lamp display for this unit is on the side of the drum sub-base. It is constructed as follows:
   (a) Mark and drill the right end of the sub-base as shown in Fig. G-14(a). File the ⅜-in. holes to accommodate the lamps.
   (b) Attach paper-clip, base terminal connections to the lamp bases, as was done for the encoder lamps and those in other units.
   (c) Cut a 4-in. long, ¾-in. wide tin strip. Bore a ⅛-in. hole at one end.
   (d) Place the lamps in their holes and attach the tin strip as in Fig. G-14(b).
   (e) Fasten supports (small wooden blocks) to the corners of the sub-base with ½-in. wood screws as shown in Fig. G-15, and wire the lamps as shown.
6. Label the unit as shown in Fig. G-12.

Final Operations

1. Using graph paper, lay out your program as indicated in Chapter 7.
2. Wrap three 2-in. pieces of adhesive around the can and next to each other. Place the program layout on top of the adhesive.
3. Using a single-edge razor blade or X-acto knife, cut out each square marked with an X.

Chapter 6
COMPUTER CONTROL

We have constructed all of the necessary calculating units but we do not yet have a functioning computer. In this chapter we will examine and build the unit that unifies all of the computer elements that we have, the control. Externally, the programmer directs the computer's sequence of operations. The internal control, however, must come from the computer itself.

This internal computer regulation is executed by countless control circuits that decode program instructions and generate electrical signals telling the various computer sections what to do. These circuits function as a central command post that prevents the computer from operating in a haphazard fashion. Although the control circuits may be spread throughout the computer, they can be understood more easily by treating them as an isolated section. The control unit for our computer will be isolated, too, to emphasize its operation.

THE COMPUTER UNITS—A REVIEW

A review of the function of the various completed computer units will help us develop an understanding of the operations the control unit directs. In the operating computer these units will act only upon command of the control unit.

The Input Unit. The input unit reads data into the computer. Using two 10-position rotary switches, it accepts two 1-digit decimal numbers, encodes each from decimal to binary, and provides an intermediate, buffer storage of the encoded bits in its display (the input buffer display).

The Arithmetic Unit. Using mechanical switches as register stores, our arithmetic unit adds or subtracts two 4-bit binary numbers and stores the results in its own register. This unit is composed of four basic functional units: the add/subtract circuit, the accumulator, the X-register, and the extension register. Data in the arithmetic unit's accumulator can be transferred to the X-register or displayed for the benefit of the computer operator.

The Storage Unit. Our computer uses two storage units: the core memory and the drum memory. The core memory, a temporary storage unit, stores five 4-bit words and is capable of receiving or releasing data. As the drum memory stores the computer program, the control unit is given direct access to it. The drum holds 26-bit instruction words.

The Output Unit. In our computer we are assuming that a program automatically converts binary numbers to binary-coded decimal numbers. Therefore, the output unit functions as a BCD-to-decimal decoder and displays the answers to the computer's calculations.

THE CONTROL UNIT

The control unit examines the program instructions and relays them to the proper units at the proper time. In our computer the chain of command will be altered somewhat because the operator works the computer manually. For this reason, the control unit will display its commands to the operator rather than send them directly to the unit involved. The operator will function as the link between control and the input, arithmetic, storage, and output units.

We have reviewed the functions of the various units to see the range of operations the control unit directs. The various instructions the control uses to initiate these operations are referred to as its *instruction repertoire*. Our control unit's instruction repertoire is set up in relation to each unit as follows.

The Input Unit. The input unit requires an instruction to *read in* data. Therefore, our first order in the instruction repertoire will be "Read in from the input buffer display (IBD) to accumulator." To simplify the writing out of instructions we will use code names for each one. For "Read in from IBD to accumulator" we will write RIN, a shortened form of *read in.*

There are two possible inputs in this unit, A and B, each at a different location, or address. Therefore, two addresses are required along with the RIN instruction, Address A and Address B.

The Arithmetic Unit. The arithmetic unit obviously needs the instructions *Add* and *Subtract,* the shorthand codes for which will be ADD and SUB, respectively. The ADD instruction will cause the contents of the accumulator to be added to the contents of the X-register. The sum obtained will be indicated automatically on the accumulator display. The SUB order will cause the contents of the X-register to be subtracted from the contents of the accumulator. The difference obtained will be indicated automatically on the accumulator display.

Besides these two arithmetic instructions, other instructions are necessary to manipulate data within the arithmetic unit. Instructions that will allow us to multiply or divide easily by powers of 2 and use the extension register are needed: "shift right" and "shift left." The notation for these instructions will be SHR (shift right) and SHL (shift left). When these instructions occur in the program, a signal must be sent to the arithmetic unit so that the accumulator contents are displayed to the operator. This "display accumulator contents" instruction will be written as DISPLAY (ACC). The operator must also be given an indication of the number of places to shift, so four addresses are

required along with the SHR or SHL: "1 place," "2 places," "3 places," and "4 places."

The instructions for the arithmetic unit are numerous because of the many operations performed in arithmetic calculations. An instruction is needed to indicate that the contents of the accumulator should be transferred to the X-register. This order, "transfer accumulator contents to X-register," or TRA, will also require the DISPLAY (ACC) address so that the accumulator's contents will be displayed to the operator.

Thus far we have defined the need for the following instructions and addresses:

Unit	*Instruction*	*Address*
Input	RIN	Address A Address B
Arithmetic	ADD	
	SUB	
	SHR	1 place
	SHL	2 places
		3 places
		4 places
	TRA	
		DISPLAY (ACC)

The Output Unit. In the output unit it is necessary to have an instruction for data to be read out from the accumulator to output display. The order, "read out accumulator to output" or RUT, also requires the DISPLAY (ACC) address so the operator is shown the accumulator contents. Since the RUT operation requires setting a BCD word into two output unit registers, two addresses are required with the instruction: 10^0 and 10^1. Finally, the output unit is designed to display results only upon receipt of an answer command, "display answer" or ANS.

The Storage Units. No instructions are associated with the drum unit as it is the source of the computer program and originates the information that the control unit uses. There are special program steps that affect the drum unit which will be considered later in this chapter.

Instructions for the core memory are needed to read data in and out. To read data into the core memory, which can store five 4-bit words, we will use a store order, STO, with five addresses. As this instruction requires transfer of the word indicated on the accumulator display to the core address indicated by control, the DISPLAY (ACC) address is also required.

To read data out of the core memory to the accumulator, the RIN instruction is used with an Address C and the five core addresses. This instruction will command the operator to determine the contents of the indicated memory core and read them into the accumulator.

Program Instructions. The fundamentals of programming for our

computer are developed in the next chapter. However, it is necessary to consider two program aspects here as they directly affect the instruction repertoire of our control unit.

Often during a program, it is necessary to make a decision based on the results of a certain computation or operation. The decision made will determine whether the program continues along its regular sequence of steps or "jumps" to a new subprogram or routine. The instruction required to call for this decision is called "conditional jump" or COJ. When a COJ instruction occurs, the operator checks the most-significant bit (MSB) in the accumulator. If the MSB is 0 the operator continues to the next instruction. If the MSB is 1, the operator follows an alternate program of instructions.

It is also necessary to jump sometimes from one place in a program to another to avoid the repetition of subroutines. A "jump" instruction, JUP, is used in this case.

THE COMPLETE INSTRUCTION REPERTOIRE

The complete list of instructions and addresses used by the control unit is given in Table 6-1. As shown, there are 10 orders and 16 addresses. Therefore, our computer word will require 26 bits as a minimum.

TABLE 6-1. THE INSTRUCTION REPERTOIRE

ORDERS	ADDRESSES
RIN	Address A
ADD	Address B
SUB	Address C
SHR	Core 1
SHL	Core 2
TRA	Core 3
RUT	Core 4
STO	Core 5
COJ	1
JUP	2
	3
	4
	DISPLAY (ACC)
	10^0
	10^1
	ANS

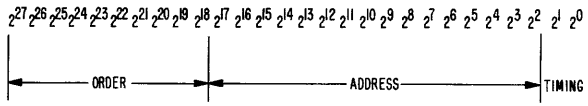


Fig. 6-1. The instruction word format.

The format for our word is shown in Fig. 6-1. Bits 2^2 through 2^{17} are used for addressing and bits 2^{18} through 2^{27} are used for instructions. Bits 2^0 and 2^1 are used for power and timing, as discussed in the chapter on storage units.

The coding for the orders and addresses is shown in Tables 6-2 and 6-3.

The control unit in our computer is a panel of coded lamps that displays instructions and addresses. The unit commands the operator by interpreting program signals in terms of computer operations and lighting the coded lamps to guide the operator in his manipulation of the computer units. The construction procedures for this unit are given at the end of the chapter.

The mechanical manipulation of the operational procedures indicated by the control unit will be detailed at the end of the chapter after the unit is completed and the computer parts are wired together. Figure 6-2 shows what the completed computer should look like.

WIRING THE SYSTEM

The various units of the computer should be arranged on a table or workspace. See Fig. 6-2. All intraunit wiring should be completed and

TABLE 6-2. THE ORDER CODE

ORDER	2^{27}	2^{26}	2^{25}	2^{24}	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}
ADD	1	0	0	0	0	0	0	0	0	0
SUB	0	1	0	0	0	0	0	0	0	0
STO	0	0	1	0	0	0	0	0	0	0
SHR	0	0	0	1	0	0	0	0	0	0
SHL	0	0	0	0	1	0	0	0	0	0
TRA	0	0	0	0	0	1	0	0	0	0
JUP	0	0	0	0	0	0	1	0	0	0
COJ	0	0	0	0	0	0	0	1	0	0
RIN	0	0	0	0	0	0	0	0	1	0
RUT	0	0	0	0	0	0	0	0	0	1

TABLE 6-3. THE ADDRESS CODE

ADDRESS	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}
Core 1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Core 2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Core 3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Core 4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Core 5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Address A	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Address B	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Address C	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
10^1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
10^0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
ANS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
DISPLAY(ACC)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

checked out in accordance with the instructions for the individual units. Once this is done, the power supply switch on the junction box is opened and the following power wiring is accomplished.

TABLE 6-4

FROM		TO	
UNIT	TERMINAL	UNIT	TERMINAL
Junction Box	B+ Bus	Arithmetic Unit	B+
Junction Box	B+ Bus	Drum Memory	B+
Junction Box	B+ Bus	Core Memory	B+
Junction Box	RTN	Input Unit (Encoder)	RTN
Junction Box	RTN	Arithmetic Unit	RTN
Junction Box	RTN	Drum Memory	RTN
Junction Box	RTN	Core Memory	RTN
Junction Box	RTN	Output Unit (Decoder)	RTN
Junction Box	RTN	Control Unit	RTN

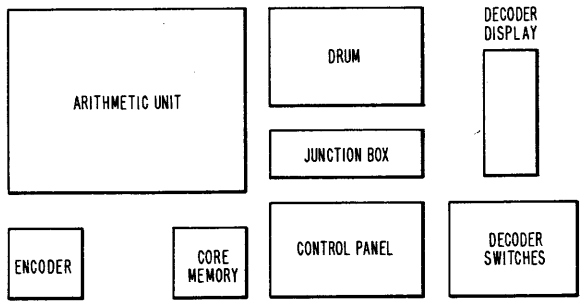


Fig. 6-2. The completed computer.

After the power wiring is installed, the following signal wiring should be installed.

TABLE 6-5

FROM		TO	
UNIT	TERMINAL	UNIT	TERMINAL
Junction Box	DISP	Drum Memory	DISP (2^2)
Junction Box	DISP	Arithmetic Unit	DISP
Junction Box	INP A	Drum Memory	INP A (2^8)
Junction Box	INP A	Input Unit (Encoder)	A
Junction Box	INP B	Drum Memory	INP B (2^9)
Junction Box	INP B	Input Unit	B
Junction Box	ANS	Drum Memory	ANS (2^3)
Junction Box	ANS	Output Unit (Decoder)	ANS

After completing this wiring, wire the drum memory and control units, starting at the left side of the drum with the terminal marked ADD. Wire all 26 order and address terminals.

CHECK-OUT PROCEDURES

Periodically, and prior to operating the computer, each unit should be tested individually according to the instructions presented for testing in the various chapters of this book. In addition, the control panel should be checked by using a test lead connected at one end to the B+ terminal of the drum. The test lead should then be touched to each read head, 2^2 through 2^{27}, in turn and the following results obtained:

TABLE 6-6

READ HEAD	NORMAL INDICATION
2^2	Accumulator display indicates contents of accumulator
2^3	ANS lamp lights and output unit register contents are displayed.
2^4	10^0 lamp lights
2^5	10^1 lamp lights
2^6	Input C lamp lights
2^7	IBD indicates Input B switch setting, Input B lamp lights
2^8	IBD indicates Input A switch setting, Input A lamp lights
2^9	4 lamp lights
2^{10}	3 lamp lights
2^{11}	2 lamp lights
2^{12}	1 lamp lights
2^{13}	Core 5 lamp lights
2^{14}	Core 4 lamp lights
2^{15}	Core 3 lamp lights
2^{16}	Core 2 lamp lights
2^{17}	Core 1 lamp lights
2^{18}	RUT lamp lights
2^{19}	RIN lamp lights
2^{20}	COJ lamp lights
2^{21}	JUP lamp lights
2^{22}	TRA lamp lights
2^{23}	SHL lamp lights
2^{24}	SHR lamp lights
2^{25}	STO lamp lights
2^{26}	SUB lamp lights
2^{27}	ADD lamp lights

If this test is successfully completed a program may be loaded (see Chapter 7) and operation may begin.

OPERATING PROCEDURES

To operate the computer, the operator must become familiar with the manual operations he must perform under direction of program control. Remember, our computer does not have active components and hence the human operator simulates many normally automatic functions. It is suggested that a chart be made of the required operator's procedures as prescribed in Table 6-7 and displayed conveniently near the computer for ready reference.

The operator, after loading a program, sets the drum to its 0 position and closes the power switch on the Junction Box. The operator then advances the drum, one slot at a time, performing the required operations as indicated by the control panel and the following table.

TABLE 6-7

LAMP	OPERATION
ADD	Place the DISPLAY switch in the ADD-SUBTRACT position and the ADD-SUBTRACT switch in the ADD position. Register answer as displayed on accumulator display into the accumulator. Set the DISPLAY switch back to the DISPLAY position. Set X-register to zero.
SUB	Place the DISPLAY switch in the ADD-SUBTRACT position and the ADD-SUBTRACT switch in the SUBTRACT position. Register answer as displayed on the accumulator display into the accumulator. Set the DISPLAY switch back to the DISPLAY position. Set X-register to zero.
STO	Transfer the word on the accumulator display to the core address indicated.
SHR	Note the word on the accumulator display; reset the accumulator switches (including Extension register) so that the display word is shifted the indicated number of digits to the right. For example, if the displayed word is 00101 0000 and the core 2 lamp is lighted, the word is shifted two places to the right as follows: 00001 0100.
SHL	Same as SHR except shift to the left.
TRA	Transfer the contents of the accumulator as displayed to the X-register by setting the X-register switches.
JUP	Note whether the 1, 2, 3, or 4 lamp is lighted. If the 1, 2, or 3 lamp is lighted, rotate the drum forward until the same lamp and JUP lamp light again. If the 4 lamp is lighted, rotate the drum backwards until the 4 lamp and JUP lamp light again. Perform the indicated order.
COJ	Same as for JUP, but perform only when accumulator sign or 2^4 bit lamp indicates 1. Otherwise, proceed to next instruction.

TABLE 6-7 (Cont.)

LAMP	OPERATION
RIN	If IBD display lights, set the accumulator switches to reflect the contents of the IBD. If a core address lights, throw display switch for that core address and then set the contents of the core display in the accumulator.
RUT	Transfer the displayed accumulator contents into the indicated output buffer register 10^0, or 10^1 or read the computer output if the ANS lamp lights.

CONSTRUCTION DETAILS—CONTROL PANEL

COMPONENTS: *Chassis (panel and support), display circuit, terminal board*

MATERIALS

Chassis:

1 12 × 8½ × ⅛ in. composition board
1 5 × 8½ × ¼ in. composition board
4 ½ in. wood screws (no . 4)

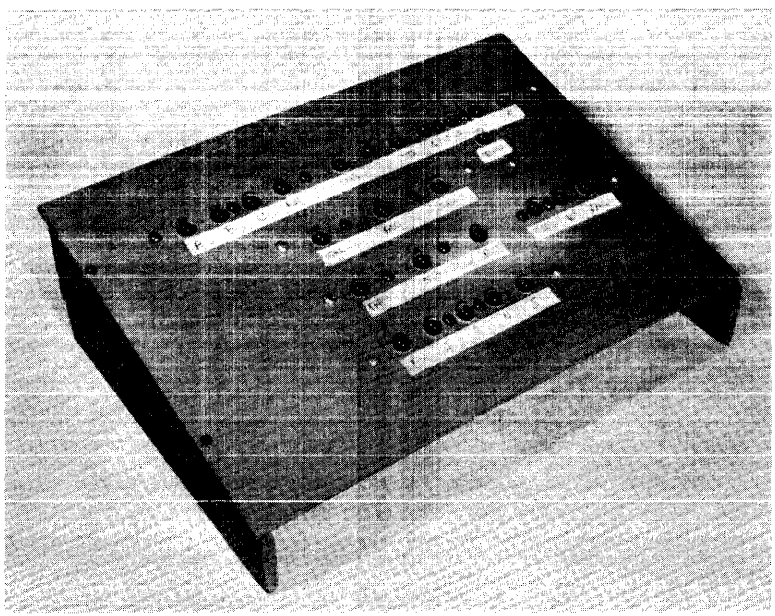


Fig. H-1.

Display circuit:

26	#48 or 41 lamps (2 v, .06 a)
7	tin strips, dimensions in construction details (can or sheet)
	23 ft. insulated hook-up wire (20 gage)
	1 ft. uninsulated hook-up wire (20 gage)
8	½ in. machine screws (6-32)
14	1 in. machine screws (6-32)
30	nuts (6-32)
	1 sheet of 8½ × 11 in. blank paper
	Adhesive tape
4 dozen	1 in. brads

Terminal board:

1	12 × 2½ × ¼ in. composition board
27	½ in. machine screws
54	nuts (6-32)
4	½ in. wood screws (no. 4)

SPECIAL TOOLS:

Drill (⅜, ⅛, 3/16, and 3/32)

Chassis Construction

1. Mark the 12 × 8½ × ⅛ in. board as shown in Fig. H-2 and start drill holes.
2. Drill all holes through the panel, following Fig. H-2 as a guide for proper drill size. Smooth rough areas with sandpaper.
3. Rule the 5 × 8½ × ¼ in. board as shown in Fig. H-3 and saw through board along ruled line. The resulting tapered pieces are the supports for the control board panel.
4. Assemble the chassis by fastening the panel to its supports as follows:

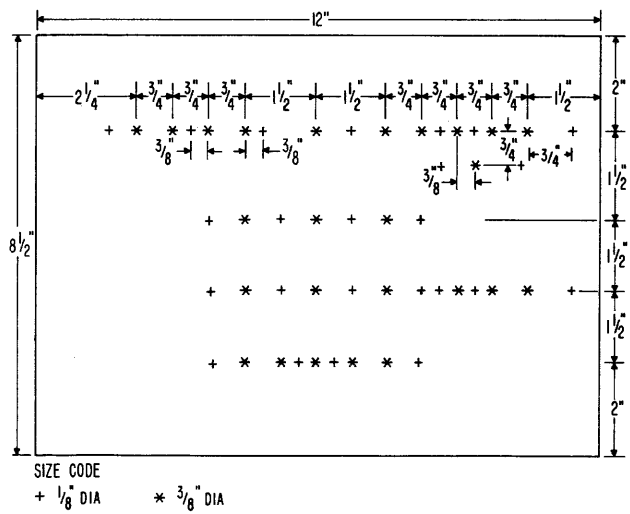


Fig. H-2.

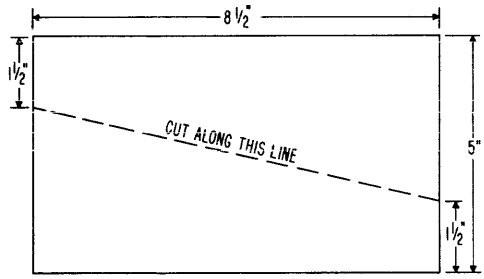


Fig. H-3.

(a) Start four drill holes on the chassis panel, one in each corner, approximately ⅛ in. from the outside edge and 2 in. from the top or bottom edge.
(b) Hold the panel and supports together as shown in Fig. H-4 with the longest row of holes in the panel at the top. Drill 3/32-in. holes through the panel and support at the points where the drill holes of Step 4(a) were started.
(c) Fasten the panel and supports together with ½-in. wood screws.

Display Circuit Construction

A. Wiring

1. Turn the control panel over (supports up) and place it on top of a sheet of 8½ × 11 in. paper. Line up the edges of the sheet with the top and bottom edges of the panel.
2. With a pencil, trace the circumference of each ⅜-in. hole so that a circle is made on the paper underneath. When each hole is marked you will have a pattern that represents the position of the lamps on the panel.
3. Remove the control panel and place the pattern on a flat working surface that you can drive nails into. This pattern is the same type of harness wiring guide that was used when harnesses were made for the Decoder.
4. Along the top edge of the pattern, mark off ⅜-in. segments starting

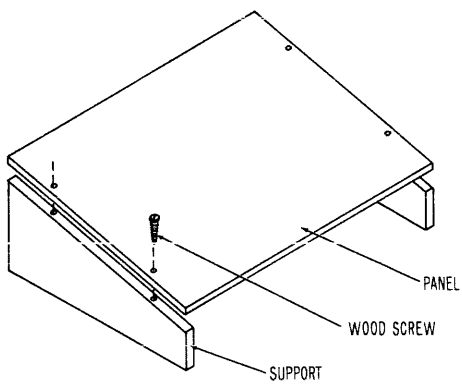


Fig. H-4.

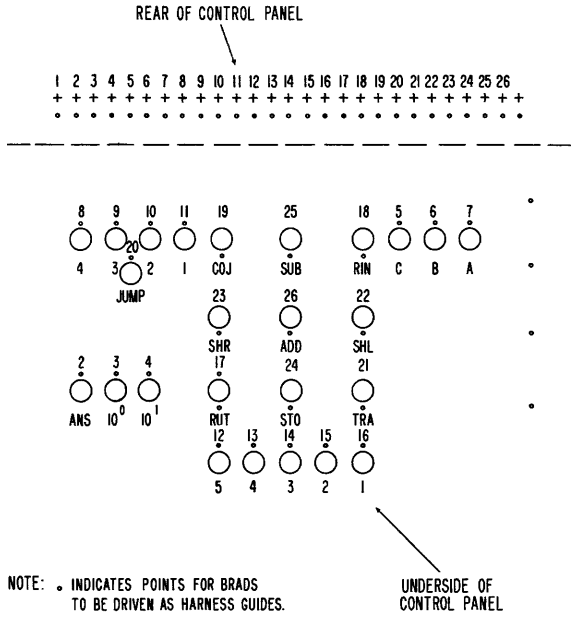


Fig. H-5.

¾ in. in from the side, as in Fig. H-5. You should have 26 such marks when finished. Number each mark, starting at the left with number 1. These marks represent the terminal connections for each lamp. Label and number each lamp hole as indicated in Fig. H-5.

5. Drive 1-in. brads into each mark at the top of the pattern. Place brads, also, at the circled points indicated in Fig. H-5. They should be driven in only as far as is necessary to hold them securely.
6. The five wiring harnesses to be constructed on this pattern will run between the lamp bases and terminal connections. Each harness, as described in the following steps, should be laid out in the same fashion as those made for the Decoder. Run wires through the guides. Use adhesive tape strips at tie points, run longest wires first, etc.

   Cut insulated wire to the lengths indicated in Chart A. Strip ½ in. of insulation from each end of each wire and tag wires at each end with identifying number indicated in the chart.
7. *Harness 1:* Connect terminals 2, 3, and 4 with holes 2, 3, and 4 as indicated in Fig. H-6. (On all figures representing harnesses, X's indicate wire guide-nails and diagonal lines indicate tie points.)
8. *Harness 2:* Connect terminals 5, 6, and 7 to their correspoding holes as shown in Fig. H-7.
9. *Harness 3:* Connect terminals 8, 9, 10, 11, and 20 to their corresponding holes as shown in Fig. H-8.
10. *Harness 4:* Connect terminals 12, 13, 14, 15, and 16 to their corresponding holes as shown in Fig. H-9.
11. *Harness 5:* Connect terminals 17, 18, 19, 21, 22, 23, 24, 25, and 26 to their corresponding holes as shown in Fig. H-10.

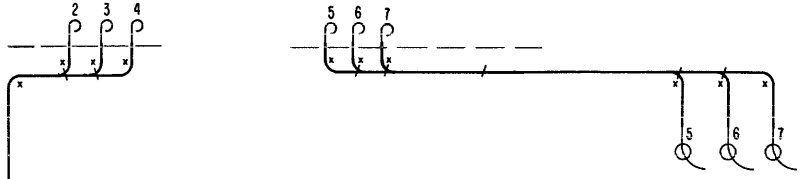


Fig. H-7.

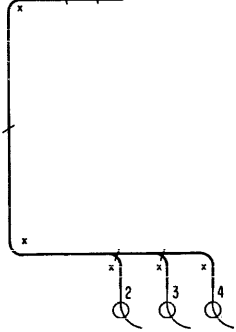


Fig. H-6.

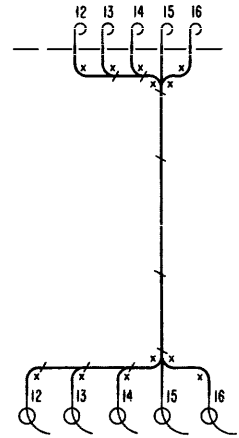


Fig. H-9.

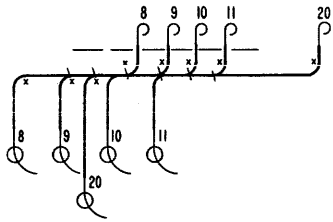


Fig. H-8.

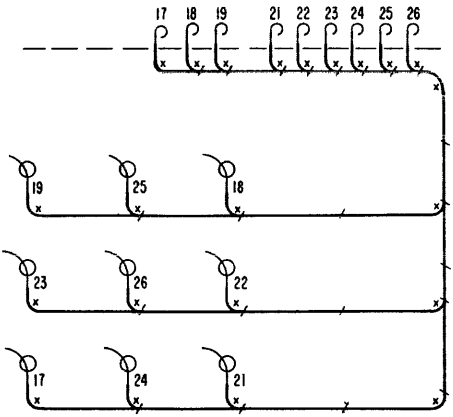


Fig. H-10.

B. Lamps and lamp connections

1. Prepare paper clips for lamp bases as was done for the Encoder lamps and secure them to the 25 control panel lamps.
2. Test each lamp hole to be sure that each lamp protrudes approximately ⅛ in. to ¼ in. above the panel face when inserted from the underside of the panel. Adjust lamps to proper height by filing holes.
3. Cut ½-in. wide tin strips in the following number and lengths: one 7-in.

CHART A

Terminal	*Length (in.)*	*Order of Use*	*Terminal*	*Length (in.)*	*Order of Use*
2	7	1	15	7½	16
3	8	2	16	8½	17
4	9	3	17	18	18
5	7	7	18	11½	19
6	9½	5	19	14	20
7	9½	6	20	5	11
8	4¾	8	21	13	21
9	4¼	9	22	12	22
10	3½	10	23	14	23
11	3⅛	12	24	14	24
12	11½	13	25	10¼	25
13	10¾	14	26	11	26
14	9	15			

CHART B

Lamps	*Type*	*Strip Designation*
8, 9, 10, 11, 19	1	A
25, 18, 5, 6, 7	3	B
20	5	C
23, 26, 22	2	D
2, 3, 4, 17	1	E
24, 21	4	F
12, 13, 14, 15, 16	1	G

strip; four 6-in. strips; one 5-in. strip; one 3-in. strip.

4. Punch and drill each strip as shown in Fig. H-11. Use a 3/16-in. drill. Drill three strips as Type 1.
5. Insert the lamps in the holes and attach the tin strips to the lamp tips using 1 in. machine screws and nuts. Chart B indicates which strip type to use with each set of lamps. Figure H-12(a) details the manner in which the strips are attached.
6. Depress ends of strips A, D, E, and G to panel and drill through panel with a ⅛-in. bit, using end holes in strip as a guide. Angle the strip so that it does not contact the lamp (see Fig. H-12(b)). Secure both ends of strips D and G with ½-in. machine screws and nuts. Secure the left end of strip A with 1-in. machine screws and the left end of strip E with ½-in. machine screws and right ends of same strips with 1-in. machine screws.
7. Connect adjoining ends of strips A and B with same machine screw. Drill through panel and secure right end of strip B with a 1-in. machine screw.
8. Connect adjoining ends of strips E and F with same machine screws. Depress right end of strip F, drill through panel, and secure with ½-in. machine screw.

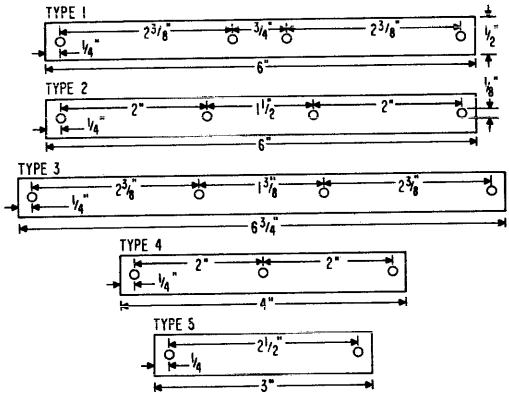


Fig. H-11.

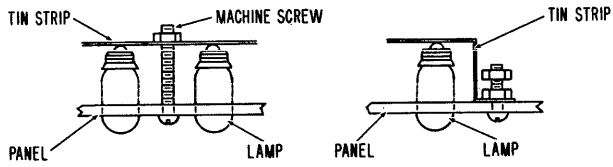


Fig. H-12.

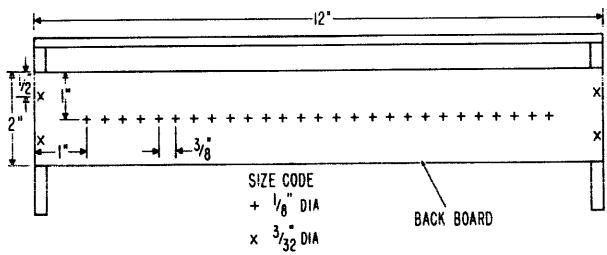


Fig. H-13.

9. With 3 short lengths of uninsulated wire, connect the machine screws securing strips A and C, C and D, D and F, and F and G. Secure these connections with a second nut.

Terminal Board Construction and Harness Installation

1. Starting 1 in. in from the end on the 12 × 2½ in. board, mark off 26 points, ⅜ in. apart, along the center of the board, as shown in Fig. H-13.
2. Start a drill hole at each point and then drill through the board with the ⅛-in. bit.
3. At each corner of the board, ⅛ in. from the edge and ¼ in. from the top or bottom, start a drill hole and drill through with the 3/32-in. bit.
4. Insert a ½-in. machine screw in each ⅛-in. hole. Position the screw with two nuts, as indicated in Fig. H-14.
5. Lay the harnesses out on the underside of the panel, one by one. For

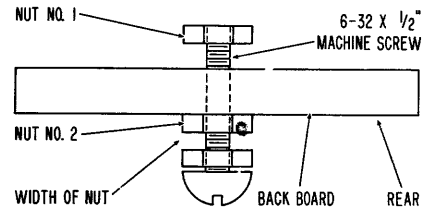


Fig. H-14.

each harness, connect the stripped ends to the lamp bases and crimp for a secure connection.

6. Attach the free ends of the harness wires to the screws on the terminal board at the back of the board. The wires should run in numerical order from left to right. Secure each connection by tightening the nut. There will be no wire for terminal 1 at the far left.
7. Secure the terminal board to the back of the control panel supports by placing it ½ in. below the top of the board and drilling through the holes on its edge into the supports. Fasten the board to the supports with ½-in. wood screws.
8. Attach a short, uninsulated wire to terminal 1 and the screw holding the left end of strip A.
9. Label the panel and terminal board as shown in Fig. H-15.

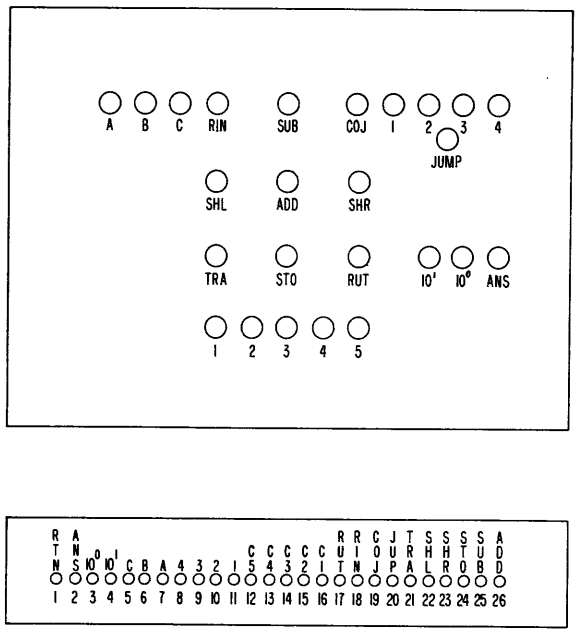


Fig. H-15.

CONSTRUCTION DETAILS—JUNCTION BOX

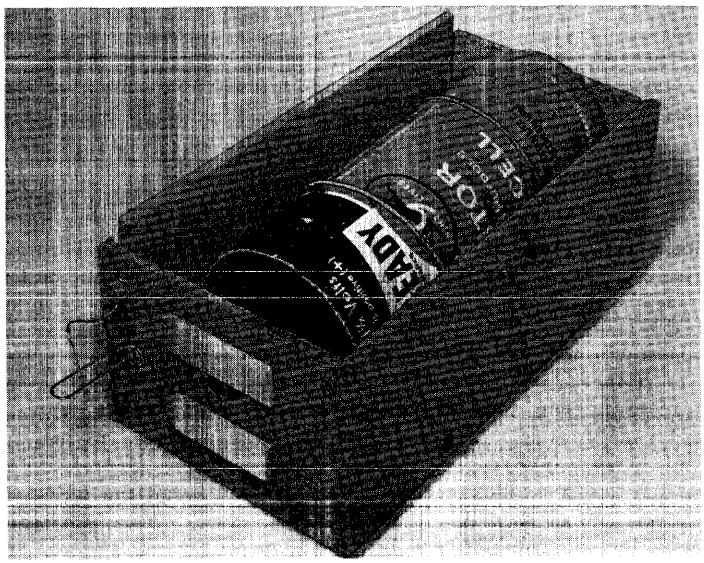


Fig. I-1.

MATERIALS

1	4 × 6 × ½ in. block of wood
2	2 × 4 × ⅛ in. composition boards
2	2 × 5½ × ⅛ in. composition boards
1	6 volt battery
1	paper clip
24	½ in. wood screws (no. 4)
8	½ in. machine screws (6-32)
8	nuts (6-32)
	2 ft. insulated hook-up wire (20 gage)
	1 ft. insulated wire (18 gage)

Construction

1. Attach battery to block of wood using uninsulated wire and wood screws. See Fig. I-2.
2. Assemble the 4 pieces of composition board to the block of wood as shown in Fig. I-3. After drilling screw holes, attach the board with ½-in. wood screws.
3. Construct two 2-in. long common tie-point terminal strips (see construction details below). Mount them on the front side of the junction box as in Fig. I-3(a) with ½-in. machine screws and nuts.
4. Construct a single-pole switch with the paper clip and two wood screws and attach at side of terminals as shown in Fig. I-3.

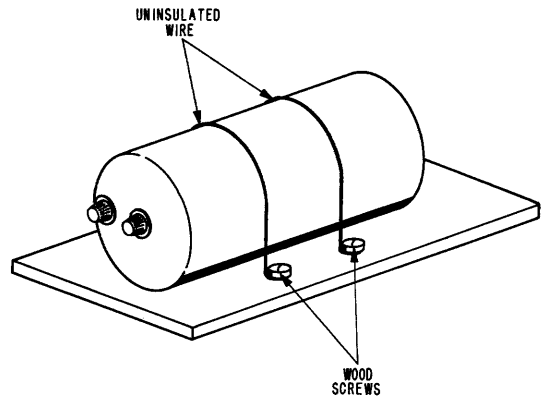

UNINSULATED
WIRE
WOOD
SCREWS

Fig. I-2.

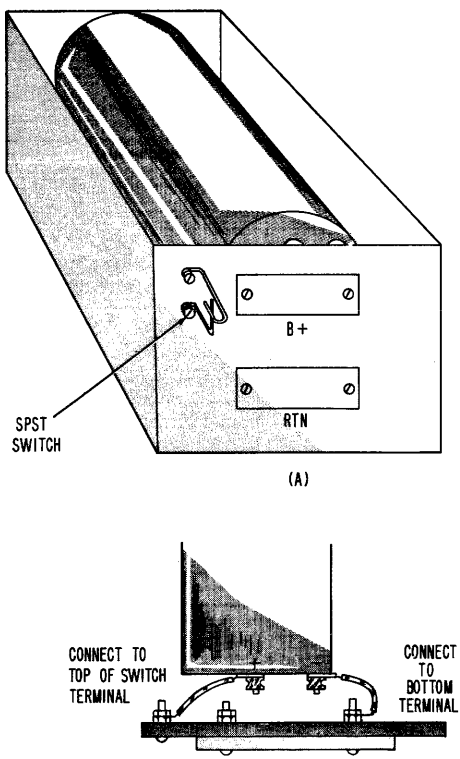

B+
RTN
SPST
SWITCH
(A)
CONNECT TO
TOP OF SWITCH
TERMINAL
CONNECT
TO
BOTTOM
TERMINAL
(B)

Fig. I-3.

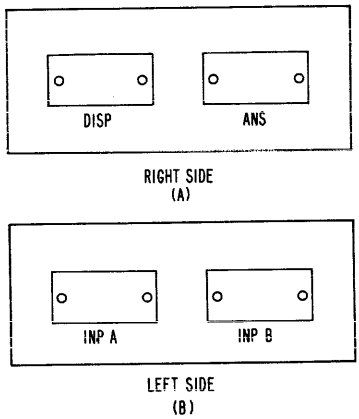


Fig. I-4.

5. Connect the negative battery terminal to one strip as shown in Fig. I-3 with insulated hook-up wire. Connect the positive battery terminal to one side of switch. Connect other side of switch to remaining common tie-point terminal strips.
6. Label the strip connected to switch B+ BUS and the other RTN BUS. When wiring into the computer system, use separate wood screws for each connection whenever possible.
7. Construct four 1-in. long common tie-point terminal strips and mount two on each side of the junction box with ½-in. machine screws and nuts (Fig. I-4).
8. Insert two ½-in. wood screws in each terminal strip mounted in Step 7. Do not screw them down tight.
9. Label the four terminal strips mounted in Step 7 as follows: DISP, ANS, INP A, INP B.

CONSTRUCTION DETAILS—COMMON TIE POINT TERMINAL STRIPS

MATERIALS

Construction board (1-in. wide, ⅛-in. thick, length to suit application)
Fine wire mesh or aluminum foil
½-in. wood screws

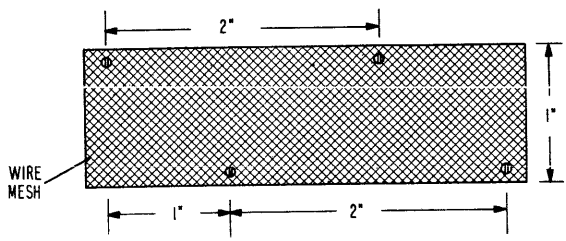


Fig. I-5.

Construction

1. Cut 1-in. construction board to desired length.
2. Cut wire mesh to same size as construction board cut in Step 1.
3. Attach the wire mesh to the board as illustrated in Fig. I-5 using ½-in. wood screws approximately every inch along the edges.
4. To connect wire to the mesh, place the stripped end over the mesh or aluminum foil and secure it with a wood screw or thumb tack. Make sure that the wire is in firm contact with the mesh or aluminum foil.

Chapter 7
PROGRAMMING OUR COMPUTER

Up to now we have concerned ourselves with the problem of devising and constructing a machine with a certain "built-in capability" to perform such basic operations as addition, subtraction, shifting, etc. The subject of this chapter is how the computer is directed so that it uses its built-in capabilities to solve the problem or problems at hand. This subject is known as *programming.*

To make a computer solve a problem, the method by which the computer is to solve this problem must be defined and related, in step-by-step sequence, to the instructions available in the computer's instruction repertoire. This repertoire was discussed in Chapter 6.

THE PROGRAMMING PROCESS

The process of creating a program for a particular computer resolves itself into the following basic steps:

1. Define the problem and what data are available for its solution; define the form of the solution.
2. Outline the method of solution in graphic form using a "flow chart."
3. Write the entire program, step by step, using the instructions available in the instruction repertoire and the appropriate address codes.
4. Try the program on the computer, using enough different sets of raw input data to provide confidence in the general applicability of the program.

Defining the problem is a very important step in devising a program, since an efficient program can only be written if it is properly organized at the start. The problem definition phase may result in several subproblem definitions. In this case, several smaller programs, called *routines* or *subroutines,* may be devised along with an overall "executive" routine that directs the use of the subroutines. A typical program consisting of an executive routine plus several subroutines is indicated in Fig. 7-1. This diagram is called a *flow chart,* since it indicates the "flow" of the program.

Once the overall configuration of the program is defined by flow chart, each subroutine is outlined using detailed flow charts. A typical

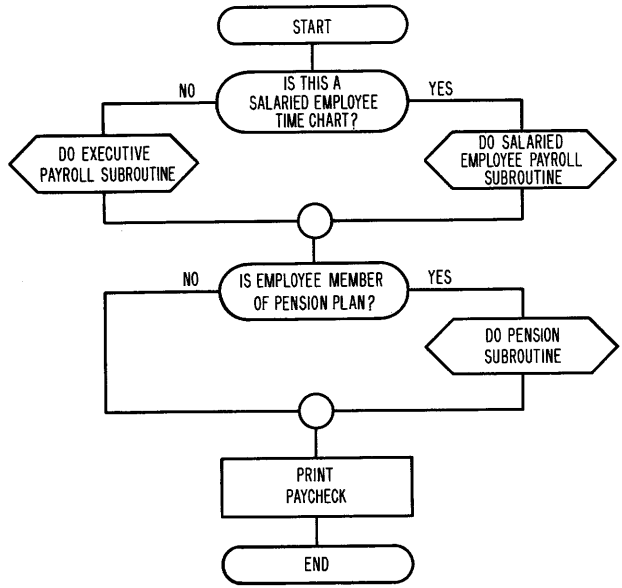


Fig. 7-1. A typical overall flow chart.

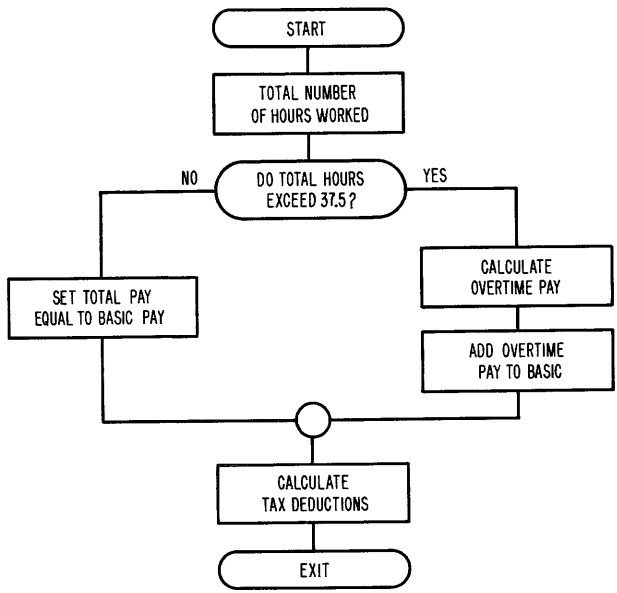


Fig. 7-2. A typical detailed flow chart.

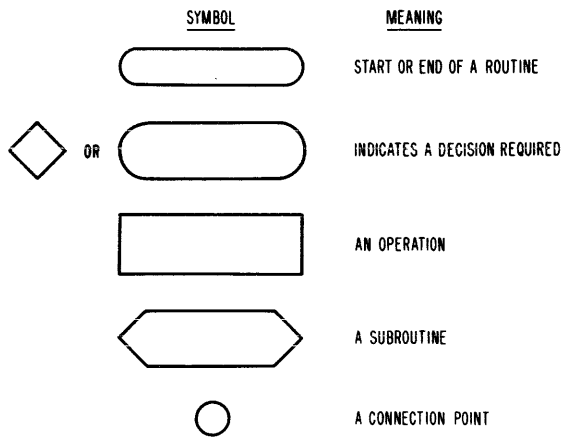


Fig. 7-3. Flow chart symbols.

detailed flow chart is illustrated in Fig. 7-2. This chart shows both the sequence and logic of the routine.

At this point, you may have noticed that boxes and symbols of different shapes are used on the flow charts. There are several standards currently in use in the computer industry for such symbols. We will use those shown in Fig. 7-3.

TABLE 7-1. TYPICAL PROGRAM CODING SHEET

STEP	MNEMONIC	INSTRUCTION WORD		REMARKS
		ORDER	ADDRESS	
00	RIN	04	210	Hours worked Monday
01	RIN	04	211	Hours worked Tuesday
02	RIN	04	212	Hours worked Wednesday
03	RIN	04	213	Hours worked Thursday
04	RIN	04	214	Hours worked Friday
05	ADD	01	300	Total hours
06	TRA	02	100	37.5 to X-register
07	SUB	11	300	Total -37.5
10	COJ	21	015	<37.5 jump to 015
11	MUL	43	300	Overtime Pay
12	TRA	02	100	Basic pay to X-register
13	ADD	01	300	Total pay
14	JUP	30	016	Jump to tax deductions

After the detailed flow chart is prepared, the program is coded, using the appropriate instructions, addresses, and codes. Table 7-1 illustrates a typical program coding sheet.

USING OCTAL CODES

Because of the number of digits required to specify a complete instruction word (28), a shorter code is used when programming. This code uses octal (base 8) numbers, instead of binary numbers. It is a simple substitution of a symbol used in the octal system for a set of three consecutive binary digits. Use of the octal code cuts the space required for writing instruction words to one-third. The table below shows the correspondence of octal and binary numbers.

Octal	*Binary*		
8^n	2^2	2^1	2^0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

To convert from binary to octal, the binary word is separated into groups of three consecutive digits, starting at the right. Each discrete group of three digits is then compared with the corresponding octal symbol. For example, the code for SHR is 0001000000. To convert to octal, the binary word is separated as follows:

xx0 001 000 000

Then, for each group of three digits, the corresponding octal symbol is selected. Therefore:

0 1 0 0

This short-cut method of writing out long binary words is possible because 2^3 is equal to 8^1.

Table 7-2 provides the octal codes for the various instructions and addresses used in our computer. Refer to Chapter 6 for details of the instruction repertoire and the corresponding binary codes. The "Remarks" column of the table uses the following shorthand conventions:

( ) = contents of
$\longrightarrow$ = automatic routing of data
$\xrightarrow{M}$ = manual routing of data

TABLE 7-2. COMPUTER INSTRUCTIONS

OPERATION CODE	OCTAL CODE	ADDRESS CODES	REMARKS
ADD	1000	000000	(Acc) + (x) ⟶ Acc Display
			(Acc Display) $\xrightarrow{M}$ Acc
			(x) $\xrightarrow{M}$ zero
			ADD lamp on
SUB	0400	000000	(Acc) - (x) ⟶ Acc Display
			(Acc Display) $\xrightarrow{M}$ Acc
			(x) $\xrightarrow{M}$ zero
			SUB lamp on
STO	0200		(Acc) ⟶ Acc Display
			(Acc Display) $\xrightarrow{M}$ CORE ADDRESS
			STO (store) lamp on
		400004	Core address 1 lamp on
		200004	Core address 2 lamp on
		100004	Core address 3 lamp on
		040004	Core address 4 lamp on
		020004	Core address 5 lamp on
SHR	0100		(Acc) ⟶ Acc Display
		400004	(Acc Display) 2^{-1} $\xrightarrow{M}$ Acc (Shift one place)
			SHR (shift right) lamp on
		200004	Shift two places
		100004	Shift three places
		040004	Shift four places
SHL	0040		(Acc) ⟶ Acc Display
			(Acc Display) 2^{1} $\xrightarrow{M}$ Acc
			SHL (shift left) lamp on
		400004	Shift one place
		200004	Shift two places
		100004	Shift three places
		040004	Shift four places

TABLE 7-2. COMPUTER INSTRUCTIONS (Cont.)

OPERATION CODE	OCTAL CODE	ADDRESS CODES	REMARKS
TRA	0020	000004	(Acc) ⟶ Acc Display (Acc Display) $\xrightarrow{M}$ x TRA (transfer) lamp on
JUP	0010		JUP (jump) lamp lights To identify jump sequence the following addresses are used:
		010000	Jump to 1
		004000	Jump to 2
		002000	Jump to 3
		001000	Jump to 4 (opposite direction)
COJ	0004		COJ (conditional jump) lamp lights To identify jump sequence the following addresses are used:
		010000	Jump to 1
		004000	Jump to 2
		002000	Jump to 3
		001000	Jump to 4 (opposite direction)
RIN	0002	000400	RIN (read in) lamp lights INPUT A displayed on IBD (IBD) $\xrightarrow{M}$ Acc
		000200	Same as above, but INPUT B
		000100	Input C lamp lights Core⟶Core display by throwing (address) display switch manually corresponding to lighted address lamp (Core Display) $\xrightarrow{M}$ Acc
		400100	Core address 1
		200100	Core address 2
		100100	Core address 3
		040100	Core address 4
		020100	Core address 5

TABLE 7-2. COMPUTER INSTRUCTIONS (Cont.)

OPERATION CODE	OCTAL CODE	ADDRESS CODES	REMARKS
RUT	0001	000004	RUT (read out) lamp lights
			(Acc) ⟶ Acc Display
		000044	10^1 lamp lights
			(Acc Display) $\xrightarrow{M}$ 10^1 ORD
		000024	10^0 lamp lights
			(Acc Display) $\xrightarrow{M}$ 10^0 ORD
		000014	ANS lamp lights; output display matrix is energized

DEVISING A SIMPLE PROGRAM

At this point, we are ready to try our first program. A simple program is best to start with, so let us devise a program for subtracting decimal input B from decimal input A. The result of the subtraction must be displayed in decimal form.

We recall now that the input unit automatically converts decimal

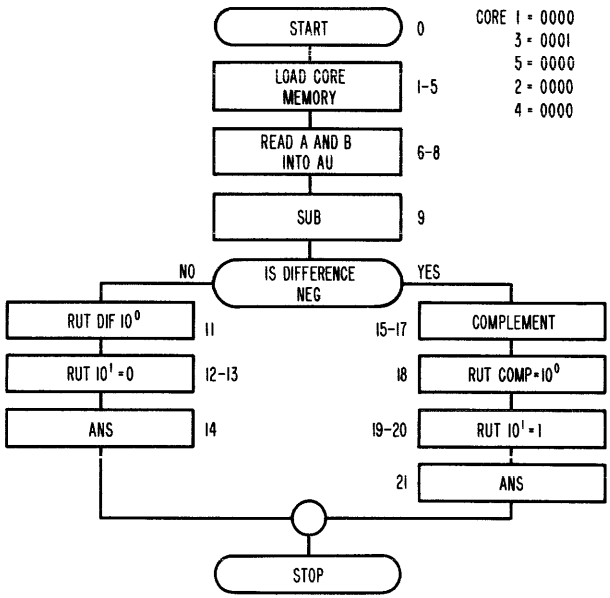


Fig. 7-4. Subtraction routine flow chart.

data to binary and that the output unit automatically converts binary-coded decimal data to decimal. However, since we are limiting ourselves to numbers less than 10, the binary numbers in the computer will be identical to BCD numbers and may be read directly into the output unit.

The first step, now that the problem has been defined, is to outline the program using a flow chart. The flow chart for the subtract program is illustrated in Fig. 7-4. Initially, the core memory is loaded with all zero's except for one location where 0001 is set for use in reading out the sign digit when the result of the subtraction is negative. After the core memory is loaded, input B is placed in the X-register and input A in the accumulator. The difference, A — B, is obtained by the SUB instruction. If the difference is positive, it is read out as 10^0, since this difference must be less than 10 and therefore already in BCD form. A 0 is then extracted from core memory and read out ($10^1 = 0$) to indicate a positive number.

If the difference is negative, it is complemented to obtain its absolute value. The complement is obtained by subtracting the difference from 2^4. This absolute value is read out as the magnitude of the difference (10^0 digit). The stored sign digit, 1, is then extracted from core address 3 and read out as 10^1, indicating a negative number.

Now, with the logic outlined, the actual program coding may be performed. Table 7-3 lists the coding for the entire subtract program.

PROGRAMMING MULTIPLICATION

Programming the multiplication process is a more complicated procedure than programming subtraction. Because of this, it is necessary to refamiliarize ourselves completely with the process of multiplication before we do any actual programming. Let us examine the basic problems encountered when paper-and-pencil multiplication is performed. For example, the problem 24 × 13:

24	*multiplicand*
× 13	*multiplier*
72	*partial product*
24	*partial product*
312	*product*

In this operation, we take the least significant numeral of the multiplier and mentally add the multiplicand to itself the number of times indicated by the multiplier numeral. The sum in this case, 72, is set down just below the multiplier in the proper position. We then take the next significant numeral of the multiplier and mentally add the multiplicand to itself the number of times indicated by this multiplier numeral. The sum in this case, 24, is set down below the previous sum but is shifted to the left by one column. Then both sums are added to obtain

TABLE 7-3. SUBTRACT ROUTINE CODING

STEP	MNEMONIC	INSTRUCTION WORD		REMARKS
		ORDER	ADDRESS	
1	RIN	0002	400000	Load core address 1 = 0000
2	RIN	0002	101000	Load core address 3 = 0001
3	RIN	0002	020000	Load core address 5 = 0000
4	RIN	0002	200000	Load core address 2 = 0000
5	RIN	0002	040000	Load core address 4 = 0000
6	RIN (B)	0002	000200	Read B into accumulator
7	TRA	0020	000004	Transfer B into X-register
8	RIN (A)	0002	000400	Read A into accumulator
9	SUB	0400	000000	A - B = difference
10	COJ	0004	010004	If difference negative, jump to 15
11	RUT (10^0)	0001	000024	Difference = 10^0 output
12	RIN (C)	0002	400100	Read core 1 = 0000 into accumulator
13	RUT (10^1)	0001	000044	Sign = 0 (pos) (10^1) output
14	ANS	0000	000010	Positive difference
15	TRA	0024	010004	Negative difference to X-register
16	RIN (C)	0002	400100	Read core 1 = 0000 into accumulator
17	SUB	0400	000000	Complement negative difference
18	RUT (10^0)	0001	000024	Complement = magnitude of negative difference = 10^0 output
19	RIN (C)	0002	100100	Read core 3 = 0001 into accumulator
20	RUT (10^1)	0001	000044	Sign (10^1) digit = 1 for negative sign
21	ANS	0000	000010	Negative difference

the final answer, 312. Note that multiplication, when broken down to its basic steps is actually an addition process.

Let us now examine multiplication in the binary system. Let us multiply 1011 and 1010.

1011	*multiplicand*
1010	*multiplier*
0000	*partial product* P_0
1011	*partial product* P_1
10110	
0000	*partial product* P_2
010110	
1011	*partial product* P_3
01101110	
01101110	*product*

This operation is exactly the same as in the decimal system with one small difference. Since the binary system has only two symbols, 0 and 1, the partial products can be equal to either all zeros or to the multiplicand.

With the above examples in mind, let us attempt to write down, in sequence, all the basic operations which go into the performance of a multiplication. To make this list of operations easier to work with, let us make the following algebraic definitions of terms:

1. The digits of the multiplicand are D^3, D^2, D^1, and D^0 with D^0 the least significant. The full multiplicand is referred to as $\overline{D}$.
2. The digits of the multiplier are R^3, R^2, R^1, and R^0 with R^0 the least significant. The full multiplier is referred to as $\overline{R}$.
3. The partial products are P_0, P_1, P_2, P_3 with P_0 being that partial product formed when multiplying the multiplier digit R^0 with the multiplicand.
4. P_{01} is the sum of partial products P_0 and P_1 when P_1 is properly shifted to the left. P_{0123} is the end product.

Now, in multiplying binary numbers, we notice that the partial products are either zero or equal to the multiplicand. Hence, it is merely necessary to examine the appropriate multiplier digit in order to determine partial product. Therefore, to start our multiplication, the first step is to examine the least significant digit (or bit) of the multiplier. If the bit is a 0, the partial product is 0; if the bit is a 1, the partial product is equal to the multiplicand. The flow chart of Fig. 7-5 indicates this procedure.

The next set of steps consists of examining the next bit to the left in the multiplier. If this bit is a zero, the partial product P_1 equals zero. If this bit is a 1, P_1 equals $\overline{D}$. The sum of the partial products, P_{01}, is then obtained by adding P_0 and P_1.

This process continues as illustrated in Fig. 7-5 until the most significant bit of the multiplier is processed. At the end of this process, the sum of the partial products, P_{0123}, is formed as the end product.

Now, let us apply our computer's instruction repertoire to the solution just described by Fig. 7-5. The first general step is to examine

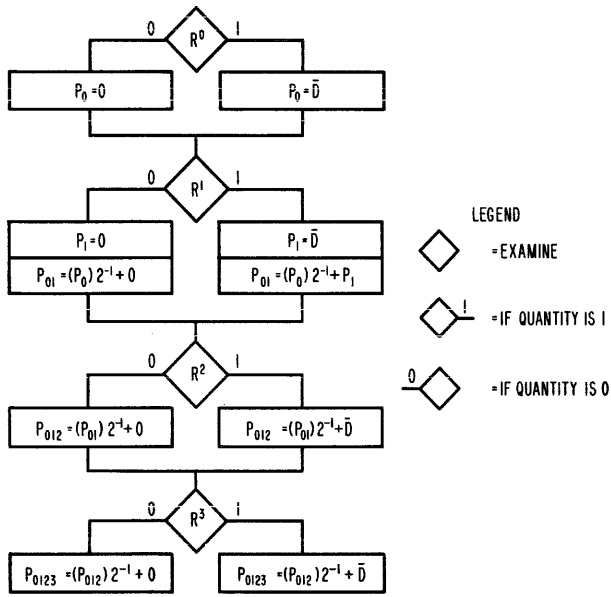


Fig. 7-5. Multiplication routine flow chart.

R^0, that is, determine if the least significant bit (LSB) of the multiplier is a 1 or a 0. This requires a decision on the part of the computer based on the value of the LSB. The instruction in our computer's repertoire covering this is COJ. However, COJ only examines the digit in the most significant stage of the accumulator. Therefore, the multiplier must be shifted to the left until the LSB is in the most significant stage of the accumulator. The instructions required to examine the multiplier bit form a subroutine, which we will refer to as the R^n subroutine. This subroutine comprises the following:

Read multiplier into accumulator (RIN)
Shift accumulator to left (SHL)
Observe MS place in accumulator, if a 1, jump to x (COJ)

Let us now examine the case when the condition is not satisfied, that is, the LSB of the multiplier is 0. The problem now is to form the first partial product based on the LSB being 0. We know from the previous discussion that this partial product is 0000. Since there is no way of insuring that 0000 is anywhere in the computer storage without actually having put this value someplace, we must load this particular constant in memory before we start. Assuming that this was done and that 0000 was loaded in core memory 3, then we can proceed as follows:

Read in from core 3 the value 0000 (RIN)
Shift to right once (SHR)

Store data in accumulator in core 4 (STO)
Jump to instruction x (JUP)

The first instruction reads in the partial product from core 3.

The second instruction shifts the partial product once to the right, causing the LSB of the partial product to be in the most-significant place of the extension register. This shift is not significant here, since the partial product is all zeros. However, adding it here allows this series of instructions to be used in the more general case, which will be described shortly. Remember when, with pencil and paper, the partial products were put down in particular locations, each being offset from the previous one by one place to the left.

The third instruction stores the accumulator data into core memory address 4, which we will use as a temporary storage space for the partial product.

The fourth instruction is necessary to jump over those instructions that concern the operation when the initial condition is satisfied. This entire set of instructions will be referred to as the $R^n = 0$ subroutine.

Now let us examine the case when the condition is satisfied. We assume that as part of the input portion of the program, the multiplier was read into core location 2 and the multiplicand was read into core location 1. We know from the previous discussion that the multiplicand, as it stands, becomes the first partial product. Therefore, we must instruct the computer to relocate the multiplicand in the location reserved for the partial product and we proceed as follows:

Read in partial product from core 4 (RIN)
Transfer to X-register (TRA)
Read in multiplicand from core 1 (RIN)
Add multiplicand and partial product (ADD)
Shift accumulator one place to right (SHR)
Store remaining accumulator data in core 4 (STO)

The first instruction enters the partial product into the accumulator and the next instruction transfers the partial product to the X-register.

The third instruction reads in the multiplicand from core 1. This is necessary, even though there is no partial product at this time, since now a subroutine is established that is usable whether or not there is a partial product. The fourth instruction adds the multiplicand and the partial product. The fifth instruction shifts the new partial product one place into the extension register. The need for this is the same as previously mentioned. The last instruction stores the three most-significant bits in core location 4 for use later on. This subroutine will be referred to as the $R^n = 1$ subroutine.

Now we have developed the three subroutines necessary to program completely for multiplication, except for instructions regarding input and output (see Table 7-4).

At this point, the program is still not complete. While the basic program is available, modifications must be made to account for shifting

TABLE 7-4. MULTIPLICATION ROUTINE—PRELIMINARY INSTRUCTION LISTING

FUNCTION	INSTRUCTION	STEP	DATA FLOW
Examine R^0	RIN	1	Read in multiplier
	SHL	2	Shift left 4 times (R^0 in MS place)
	COJ	3	If MSB = 1, Jump to 8
If 0 $P_0 = 0$	RIN	4	Read in core 3 (0000)
	SHR	5	Shift once to right
	STO	6	Store in core 4
	JUP	7	Jump to 14
If 1 $P_0 = \overline{D}$	RIN	8	Read in core 4
	TRA	9	Transfer to X-register
	RIN	10	Read in core 1 (multiplicand)
	ADD	11	Add Acc + X-reg $\xrightarrow{M}$ Acc
	SHR	12	Shift once to right
	STO	13	Store in core 4
Examine R^1	RIN	14	Read in core 2
	SHL	15	Shift left 3 times (R^1 in MS place)
	COJ	16	If MSB = 1, Jump to 21
If 0 $P_1 = (P_0)2^{-1}$	RIN	17	Read in core 4
	SHR	18	Shift once to right
	STO	19	Store in core 4
	JUP	20	Jump to 27
If 1 $P_1 = (P_0)+(\overline{D})2^1$	SHR		
	RIN	21	Read in core 4
	TRA	22	Transfer to X-register
	RIN	23	Read in core 1 (multiplicand)
	ADD	24	Add Acc + X-reg $\xrightarrow{M}$ Acc
	SHR	25	Shift once to right
	STO	26	Store in core 4
Examine R^2	RIN	27	Read in core 2
	SHL	28	Shift left 2 times (R^2 in MS place)
	COJ	29	If MSB = 1, Jump to 34
If 0 $P_2 = (P_1)2^{-1}$	RIN	30	Read in core 4 (0000)
	SHR	31	Shift once to right
	STO	32	Store in core 4
	JUP	33	Jump to 14

TABLE 7-4. MULTIPLICATION ROUTINE—PRELIMINARY INSTRUCTION LISTING (Cont.)

FUNCTION	INSTRUCTION	STEP	DATA FLOW
	SHR>		
	RIN	34	Read in core 4
	TRA	35	Transfer to X-register
If 1	RIN	36	Read in core 1
$P_2=(P_1)+(\overline{D})2^1$	ADD	37	Add Acc + X-reg $\xrightarrow{M}$ Acc
	SHR	38	Shift once to right
	STO	39	Store in core 4
	RIN	40	Read in core 2
Examine R^3	SHL	41	Shift left once (R^3 in MS place)
	COJ	42	If MSB = 1, Jump to 47
	RIN	43	Read in core 4
If 0	SHR	44	Shift once to right
$P_3 = (P_2)\ 2^{-1}$	STO	45	Store in core 4
	JUP	46	Jump to 53
	SHR>		
	RIN	47	Read in core 4
If 1	TRA	48	Transfer to X-register
$P_3 = (P_2)+(\overline{D})2^1$	RIN	49	Read in core 1
	ADD	50	Add Acc + X-reg $\xrightarrow{M}$ Acc
	SHR	51	Shift once to right
	STO	52	Store in core 4
Output Routine		53 ↓ N	

data in the extension register. Notice that when we examine the multiplier bit by shifting left we also shift data from the extension register into the accumulator. Then we insert new data into the accumulator and effectively lose those data that we originally shifted from the extension register. Therefore, we must add an additional SHR instruction to replace the data in the extension register prior to inserting new data into the accumulator. This instruction must be added after the COJ instructions in each of the following subprograms. For $R^n = 0$:

SHR—Shift right a number of places to replace extension data
RIN—Read in from core 3
SHR—Shift right once
STO—Store in core 4
JUP—Jump to required step

For $R^n = 1$:

SHR—Shift right a number of places to replace extension data
RIN—Read in from core 4
TRA—Transfer to X-register
RIN—Read in from core 1
ADD—Add multiplicand and partial product
SHR—Shift right once
STO—Store in core 4

Now, it is necessary to delete and combine steps to shorten the program because each unnecessary step takes time and memory space, which are the key limiting parameters of a computer. Note that at the end of the $R^n = 0$ routine a JUP exists and the two preceding steps are SHR and STO. At the end of the $R^n = 1$ routine an identical SHR and STO exist. By deleting SHR and STO in the $R^n = 0$ routine and jumping to the SHR and STO in the $R^n = 1$ routine, two instructions can be saved for each examination. Three steps may be saved in the examine R^0 routine by deleting RIN from core 4, TRA, and ADD. This can be done since there is no partial product in core 4 at the start of the program.

At this point, with the basic program completed, since the accumulator and extension register are both full and we have generated partial products four times, the final product is in the accumulator and the extension register. The most significant bits (MSB) are in the accumulator and the least significant bits (LSB) are in the extension register. It is necessary to read out the answer. First we must store:

Shift word in extension register to left
Store in core 5

Since the answer is in binary and the output unit uses BCD code, it is necessary to perform a separate readout routine described later in this chapter to convert binary to binary-coded decimal for readout. The addresses for the MSH and LSH (most significant half and least significant half) are, respectively, A and B. When the RUT order for the MSH is displayed, transfer the accumulator data to the A input on the encoder, and the LSH to the B input, then begin the readout routine.

Before writing down the entire program we must now determine the input portion of the program based on what is required in the multiplication portion of the program.

Recall the following requirements on our core memory:

1. Partial product in core 4
2. MS part of final product answer in core 4
3. LS part of final product in core 5
4. Multiplier in core 2
5. At start core 3 loaded with 0000
6. Multiplicand in core 1

The object of the input portion of the program is to ensure that all quantities required in the program are available. Therefore, we specify the following:

Read in multiplicand
Store multiplicand in core 1
Read in multiplier
Store multiplier in core 2

The preload data for the multiply program are as follows:

Core No.	*Data*
1	0000
2	0000
3	0000
4	0000
5	0000

Table 7-6 is a check-out routine of the program in Table 7-5. Use it to check your program after it has been cut and mounted.

TABLE 7-5. MULTIPLY PROGRAM

STEP	MNEMONIC	INSTRUCTION WORD: ORDER	INSTRUCTION WORD: ADDRESS	REMARKS	
1	RIN	0002	400000	0000 $\xrightarrow{M}$ core 1	Load Instructions
2	RIN	0002	100000	0000 $\xrightarrow{M}$ core 3	
3	RIN	0002	020000	0000 $\xrightarrow{M}$ core 5	
4	RIN	0002	200000	0000 $\xrightarrow{M}$ core 2	
5	RIN	0002	040000	0000 $\xrightarrow{M}$ core 4	
6	RIN	0002	000400	IBD-A $\xrightarrow{M}$ Acc	Input
7	STO	0200	400004	Acc $\xrightarrow{M}$ core 1	
8	RIN	0002	000200	IBD-B $\xrightarrow{M}$ Acc	
9	STO	0200	200004	Acc $\xrightarrow{M}$ core 2	
10	SHL	0040	040004	2^4 Acc $\xrightarrow{M}$ Acc	Examine R^0
11	COJ	0004	010004	If MSB = 1, Jump to 14	
12	RIN	0002	100104	Core 3 $\xrightarrow{M}$ Acc	$R^0=0$, $P_0=0$
13	JUP	0010	004000	Jump to 16	
14	RIN	0006	410104	Core 1 $\xrightarrow{M}$ Acc	$R^0=1$, $P_0=\overline{D}$
15	SHR	0100	400004	2^{-1} Acc $\xrightarrow{M}$ Acc	
16	STO	0210	044004	Acc $\xrightarrow{M}$ core 4	Read in and Store $(P_0)2^{-1}$
17	RIN	0002	200104	Core 2 $\xrightarrow{M}$ Acc	Examine R^1
18	SHL	0040	100004	2^3 Acc $\xrightarrow{M}$ Acc	
19	COJ	0004	002004	If MSB = 1, Jump to 23	

TABLE 7-5. MULTIPLY PROGRAM (Cont.)

STEP	MNEMONIC	INSTRUCTION WORD ORDER	INSTRUCTION WORD ADDRESS	REMARKS	
20	SHR	0100	100004	2^{-3} Acc $\xrightarrow{M}$ Acc	$R^1=0$
21	RIN	0002	040104	Core 4 $\xrightarrow{M}$ Acc	$P_1=(P_0)2^{-1}$
22	JUP	0010	010000	Jump to 28	
23	SHR	0104	102004	2^{-3} Acc $\xrightarrow{M}$ Acc	$R^1=1$
24	RIN	0002	040104	Core 4 $\xrightarrow{M}$ Acc	$P_1=(P_0)2^{-1}+\overline{D}$
25	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register	
26	RIN	0002	400104	Core 1 $\xrightarrow{M}$ Acc	
27	ADD	1000	000000	Acc + X-reg $\xrightarrow{M}$ Acc, X-register $\xrightarrow{M}$ 0	
28	SHR	0110	410004	2^{-1} Acc $\xrightarrow{M}$ Acc	Store $(P_1)2^{-1}$
29	STO	0200	040004	Acc $\xrightarrow{M}$ core 4	
30	RIN	0002	200104	Core 2 $\xrightarrow{M}$ Acc	Examine R^2
31	SHL	0040	200004	2^2 Acc $\xrightarrow{M}$ Acc	
32	COJ	0004	004004	If MSB = 1, Jump to 36	
33	SHR	0100	200004	2^{-2} Acc $\xrightarrow{M}$ Acc	$R^2=0$
34	RIN	0002	040104	Core 4 $\xrightarrow{M}$ Acc	$P_2=(P_1)2^{-1}$
35	JUP	0010	002000	Jump to 41	
36	SHR	0104	204004	2^{-2} Acc $\xrightarrow{M}$ Acc	$R^2=1$
37	RIN	0002	040104	Core 4 $\xrightarrow{M}$ Acc	$P_2=(P_1)2^{-1}+\overline{D}$
38	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register	
39	RIN	0002	400104	Core 1 $\xrightarrow{M}$ Acc	
40	ADD	1000	000000	Acc + X-reg $\xrightarrow{M}$ Acc, X-register $\xrightarrow{M}$ 0	
41	SHR	0110	402004	2^{-1} Acc $\xrightarrow{M}$ Acc	Store $(P_2)2^{-1}$
42	STO	0200	040004	Acc $\xrightarrow{M}$ core 4	
43	RIN	0002	200104	Core 2 $\xrightarrow{M}$ Acc	Examine R^3
44	SHL	0040	400004	2^1 Acc $\xrightarrow{M}$ Acc	
45	COJ	0004	010004	If MSB = 1, Jump to 49	
46	SHR	0100	400004	2^{-1} Acc $\xrightarrow{M}$ Acc	$R^3=0$
47	RIN	0002	040104	Core 4 $\xrightarrow{M}$ Acc	$P_3=(P_2)2^{-1}$
48	JUP	0010	004000	Jump to 54	

TABLE 7-5. MULTIPLY PROGRAM (Cont.)

STEP	MNEMONIC	INSTRUCTION WORD ORDER	INSTRUCTION WORD ADDRESS	REMARKS	
49	SHR	0104	410004	2^{-1} Acc $\xrightarrow{M}$ Acc	$R^3=1$ $P_3=(P_2)2^{-1}+\overline{D}$
50	RIN	0002	040104	Core 4 $\xrightarrow{M}$ Acc	
51	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register	
52	RIN	0002	400104	Core 1 $\xrightarrow{M}$ Acc	
53	ADD	1000	000000	Acc + X-reg $\xrightarrow{M}$ Acc, X-register $\xrightarrow{M}$ 0	
54	SHR	0110	404000	2^{-1} Acc $\xrightarrow{M}$ Acc	Store $(P_3)2^{-1}$
55	STO	0200	040004	Acc $\xrightarrow{M}$ core 4	
56	RUT	0001	000404	Acc = MSH of Answer	Readout
57	SHL	0040	040004	2^4 Acc $\xrightarrow{M}$ Acc	
58	STO	0200	020004	Acc $\xrightarrow{M}$ core 5	
59	RUT	0001	000204	Acc = LSH of Answer (Go into read out program)	

TABLE 7-6. MULTIPLY CHECKOUT

15 x 15 = 225

1111 x 1111 = 11100001

STEP	FUNCTION		SIGN	CONTENTS AFTER COMPLETION OF FUNCTION Acc	Ext	X-Reg	Core 1	2	3	4	5
6	RIN A	Read in A	0	1111	0000	0000	0000	0000	0000	0000	0000
7	STO 1	Store in 1		↓			1111				
8	RIN B	Read in B		1111							
9	STO 2	Store in 2		↓				1111			
10	SHL 4	Shift left 4x	1	0000							
11	COJ 14	If neg jump to 14		↓							
12	RIN 3	Read in 3									

TABLE 7-6. MULTIPLY CHECKOUT (Cont.)

STEP	FUNCTION		SIGN	CONTENTS AFTER COMPLETION OF FUNCTION							
				Acc	Ext	X-Reg	Core 1	2	3	4	5
13	JUP 16	Jump to 16									
14	RIN 1	Read in 1	0	1111							
15	SHR 1	Shift right 1x		0111	1000						
16	STO 4	Store in 4								0111	
17	RIN 2	Read in 2		1111							
18	SHL 3	Shift left 3x	1	1100	0000						
19	COJ 23	If neg jump to 23									
20	SHR 3	Shift right 3x									
21	RIN 4	Read in 4									
22	JUP 28	Jump to 28									
23	SHR 3	Shift right 3x	0	0111	1000						
24	RIN 4	Read in 4		0111							
25	TRA	Trans to X-reg				0111					
26	RIN 1	Read in 1		1111							
27	ADD	Add	1	0110		0000					
28	SHR 1	Shift right 1x	0	1011	0100						
29	STO 4	Store in 4								1011	
30	RIN 2	Read in 2		1111							
31	SHL 2	Shift left 2x	1	1101	0000						
32	COJ 36	If neg jump to 36	1	1101	0000	0000	1111	1111	0000	1011	0000
33	SHR 2	Shift right 2x									
34	RIN 4	Read in 4									
35	JUP 41	Jump to 41									
36	SHR 2	Shift right 2x	0	0111	0100						
37	RIN 4	Read in 4		1011							
38	TRA	Trans to X-reg				1011					
39	RIN 1	Read in 1		1111							
40	ADD	Add	1	1010		0000					
41	SHR 1	Shift right 1x	0	1101	0010						
42	STO 4	Store in 4								1101	
43	RIN 2	Read in 2		1111							
44	SHL 1	Shift left 1x	1	1110	0100						
45	COJ 49	If neg jump to 49									
46	SHR 1	Shift right 1x									

TABLE 7–6. MULTIPLY CHECKOUT (Cont.)

STEP	FUNCTION		SIGN	CONTENTS AFTER COMPLETION OF FUNCTION							
				Acc	Ext	X-Reg	Core 1	2	3	4	5
47	RIN 4	Read in 4									
48	JUP 54	Jump to 54									
49	SHR 1	Shift right 1x	0	1111	0010						
50	RIN 4	Read in 4		1101							
51	TRA	Trans to X-reg				1101					
52	RIN 1	Read in 1		1111							
53	ADD	Add	1	1100		0000					
54	SHR 1	Shift right 1x	0	1110	0001						
55	STO 4	Store in 4								1110	
56	RUT	Read out MSH		1110							
57	SHL 4	Shift left 4x	0	0001	0000						
58	STO 5	Store in 5									0001
59	RUT	Read out LSH		0001							

PROGRAMMING DIVISION

Before devising a divide program, let's examine the basic problems encountered when we divide using paper and pencil. Let's divide 52 by 4:

$$
\begin{array}{rl}
 & 13 \longleftarrow \textit{Quotient} \\
\textit{Divisor} \longrightarrow & 4\overline{)52} \longleftarrow \textit{Dividend} \\
 & \underline{4} \\
 & 12 \\
 & \underline{12} \\
 & 0
\end{array}
$$

In this operation we "guess" at a number which, when multiplied by the divisor, will give us a product less than the dividend. If the product is less, then we subtract this from the dividend. The remainder from the subtraction, if less than the divisor, is then treated as a new dividend and the process is repeated. However, if the remainder is greater than the divisor, then we increase the number initially "guessed at" and repeat the process. If the product is more than the dividend, then we decrease the guessed at number and repeat the process.

Using binary numbers, divide 0011 into 1001:

$$
\begin{array}{r}
0011 \\
0011\overline{)1001} \\
0 \\
\hline
10 \\
00 \\
\hline
100 \\
011 \\
\hline
0011 \\
0011 \\
\hline
000
\end{array}
$$

This method is the same as with decimal numbers. However, the guessing is simplified since there are only two symbols.

Figure 7-6 is a flow chart of the division program. The following conventions are used:

1. The digits of the dividend are D^3, D^2, D^1, and D^0.
2. $\overline{D}$ refers to the entire dividend.
3. The digits of the divisor are R^3, R^2, R^1, and R^0.
4. $\overline{R}$ refers to the entire divisor.
5. The digits of the quotient are Q^3, Q^2, Q^1, and Q^0.
6. The partial remainders are P_0, P_1, P_2, P_3, P_4, and P_5.

To develop the specific flow we must examine each function in the general flow and, recalling the characteristics of the computer, specify the particular instructions required.

The first general step is to determine if the MSB of the quotient is 0 or 1. If 1, jump to the specified instruction. If 0, continue with the next step. This requires a decision by the computer based on the divisor and the MSB of the dividend. The instruction covering the decision is COJ. COJ operates when the digit in the MS place of the accumulator display is a 1. Therefore, the dividend and divisor must be operated on to give the information we need. This can be accomplished by subtracting the divisor from the most significant digit of the dividend. We can now specify the examination routine.

Read in divisor
Transfer to X-register
Read in dividend
Shift dividend until MSB is in LS place of ACC
Subtract
Observe MS place in accumulator display, if a 1 jump to x($Q^3 = 0$)

The first three instructions are, respectively, RIN, TRA, and RIN. These read in the divisor and dividend and set up these quantities for an arithmetic operation. It is assumed here that the dividend and

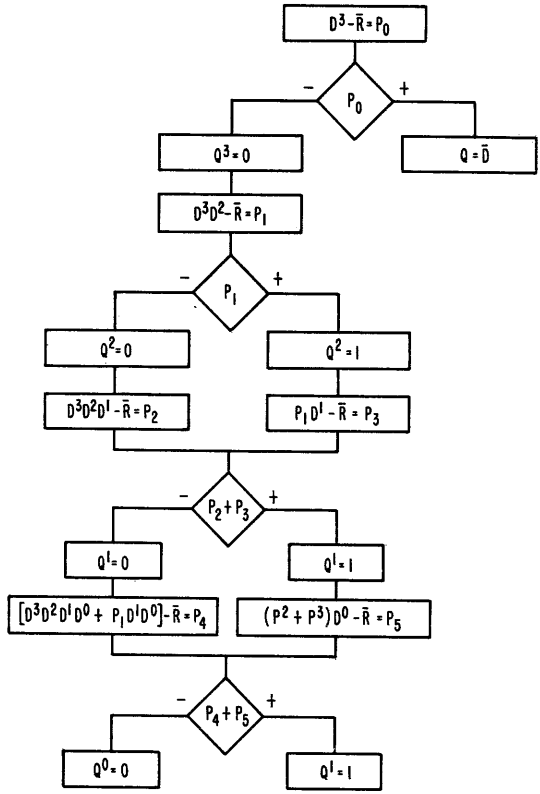


Fig. 7-6

divisor had previously been read into memory as part of the input routine. The fourth step shifts the MSB of the dividend until the bit is in the least significant place of the accumulator. For instance, divide 1010 by 0101: 0101 (divisor) is in the X-register and 1010 (dividend) is in the accumulator. Shifting the dividend as specified sets 0001 into the accumulator; i.e., we have 0001010, where 0001 is in the accumulator and 010 is in the extension register.

The fifth instruction, SUB, is subtraction, and the sixth is the conditional jump, COJ. If, as a result of subtraction, a 1 is in the MS place of the accumulator display, then the condition is satisfied and the jump is made to the particular step further in the program. Under this condition, Q^3 is a 0 and can be set into memory. The portion of the program detailing this is given later on. When a 0 occurs in the MS place, the program continues to the $Q^n = 1$ routine.

In the $Q^n = 1$ routine, two things should be considered: (1) In the paper and pencil method, a "new" dividend is developed by "bringing

down" part of the original dividend and tacking it on to the remainder. (2) A 1 must be stored in the proper location each time $Q^n = 1$.

We will now develop a generalized program for any $Q^n = 1$ and then modify each set for $Q^3 = 1$, $Q^2 = 1$, $Q^1 = 1$, and $Q^0 = 1$. The generalized $Q^n = 1$ program takes the following form. It comprises two portions, generation of the new dividend and generation of the new partial quotient Q^n.

New Dividend

STO—Store remainder in accumulator in core 4
RIN—Read in dividend from core 1
SHR—Shift dividend to right x times
RIN—Read in remainder from core 4
SHL—Shift left x times
STO—Store new dividend in core 1

New Partial Quotient

RIN—Read 0001 from core 5
SHL—Shift 0001 left x times
TRA—Transfer to X-register
RIN—Read in old partial quotient from core 3
ADD—Add
STO—Store new partial quotient in core 3
JUP—Jump to next examination routine

The first step, STO, stores the remainder from the previous subtraction order in core memory location 4. The next four steps, RIN, SHR, RIN, SHL, bring in the dividend, shift the lesser significant digits into the extension register, bring in the remainder, and then shift those lesser significant digits of the dividend out of the extension register. This effectively "brings down" the original dividend and tacks it onto the remainder. The value of x varies for each $Q^n = 1$ routine. For $n = 3$, $x = 3$; for $n = 2$, $x = 2$; therefore, $x = n$. The next order, STO, replaces the original dividend in core 1.

The next seven orders generate the partial quotient. The first order, RIN, brings 0001 in from core 5. This number is a pre-load requirement and must be loaded into the drum at the start of the program. The second order, SHL, shifts the 0001 to the proper location, x in this case being equal to n. The next three instructions, TRA, RIN, and ADD, combine the previous partial quotient with the new partial quotient. The next step, STO, stores the new quotient in core 3 and the following step, JUP, directs the program to the next examine routine.

In the $Q^n = 0$ routine, no action is required. Therefore, this routine does not exist and the COJ instruction of the examine routine specifies a jump to the examine routine.

Before calling the basic program final, it is necessary to see if it can be shortened. Examine the first time through the routine when the result of the examination is positive. Under this condition we note the following three possible conditions:

1. $D^3 = 1$ and $\overline{R} = 1$
2. $D^3 = 1$ and $\overline{R} = 0$
3. $D^3 = 0$ and $\overline{R} = 0$

The last two conditions indicate division by zero, which is an invalid operation. Therefore, the only valid condition is the first one, which is division by 1. This, of course, yields the dividend as the quotient. We then can substitute the $Q^3 = 1$ routine for the generalized $Q^n = 1$ routine, which will involve the following orders:

RIN—Read in dividend from core 1
RUT—Read out answer

In the $Q^n = 1$ routine where $n = 2$, we note that the Q^3 digit is 0 by virtue of the preceding discussion. Therefore, there is no need to add the old quotient to the new and we can rewrite the "generate new quotient" portion as follows:

RIN—Read in 0001 from core 5
SHL—Shift left twice
STO—Store partial quotient in core 3

The $Q^1 = 1$ routine remains as in the generalized $Q^n = 1$ routine. However, in the $Q^0 = 1$ routine we consider the following:

1. There is no need to store the remainder, except of course if you wish to continue the program to obtain fractions in the end quotient.
2. There is nothing left of the dividend to "bring down."
3. The prestored data 0001 is in the proper position before adding to the old quotient.

The program can be revised as follows:

RIN—Read in 0001
TRA—Transfer to X-register
RIN—Read in "old" quotient from core 3
ADD—Add
STO—Store "new" quotient in core 3

Now we have developed the subroutines necessary to program completely for division, except for input and output. Table 7-7 contains the full list of the previously discussed instructions.

The input instructions for the divide program consist of reading in the divisor and dividend. The divisor is stored in core 2 and the dividend in core 1. The following sequence is suggested:

Read in divisor
Store in core 2
Transfer to X-register
Read in dividend
Store in core 1

With these five instructions we disregard the first three instructions in the basic program to shorten the program as much as possible.

TABLE 7-7. BASIC DIVISION ROUTINE

FUNCTION	INSTRUCTION	STEP	DATA FLOW
Examine D^3	RIN	1	Read in divisor
	TRA	2	Transfer to X-register
	RIN	3	Read in dividend
	SHR	4	Shift right 3 times
	SUB	5	Subtract
	COJ	6	If MSB = 1, jump to 9
$D^3 = 1$	RIN	7	Read in dividend
	RUT	8	Read out answer
$D^3 = 0$ Examine D^2	RIN	9	Read in divisor
	TRA	10	Transfer to X-register
	RIN	11	Read in dividend
	SHR	12	Shift right twice
	SUB	13	Subtract
	COJ	14	If MSB = 1, jump to 24
$D^2 = 1$ Generate new dividend	STO	15	Store remainder in core 4
	RIN	16	Read in dividend
	SHR	17	Shift right twice
	RIN	18	Read in remainder from core 4
	SHL	19	Shift left twice
	STO	20	Store "new" dividend in core 1
$D^2 = 1$ Generate partial quotient	RIN	21	Read in 0001 from core 5
	SHL	22	Shift left twice
	STO	23	Store Partial Quotient in core 3
$D^2 = 0$ Examine D^1	RIN	24	Read in divisor
	TRA	25	Transfer to X-register
	RIN	26	Read in dividend
	SHR	27	Shift right once
	SUB	28	Subtract
	COJ	29	If MSB = 1, jump to 42
$D^1 = 1$ Generate new dividend	STO	30	Store remainder in core 4
	RIN	31	Read in "dividend" from core 1
	SHR	32	Shift right once
	RIN	33	Read in remainder from core 4
	SHL	34	Shift left once
	STO	35	Store new "dividend" in core 1

TABLE 7-7. BASIC DIVISION ROUTINE (Cont.)

FUNCTION	INSTRUCTION	STEP	DATA FLOW
	RIN	36	Read in 0001 from core 5
$D^1 = 1$	SHL	37	Shift left once
Generate	TRA	38	Transfer to X-register
partial	RIN	39	Read in partial quotient from
quotient			core 3
	ADD	40	Add
	STO	41	Store new partial quotient in
			core 3
	RIN	42	Read in divisor
	TRA	43	Transfer to X-register
$D^1 = 0$	RIN	44	Read in "dividend" from core 1
Examine D^0	SUB	45	Subtract
	COJ	46	If MSB = 1, jump to 52
			(output routine)
	RIN	47	Read in 0001 from core 5
$D^0 = 1$	TRA	48	Transfer to X-register
	RIN	49	Read in partial quotient from
Generate			core 3
end	ADD	50	Add
quotient	STO	51	Store end quotient in core 3

The output instructions consist of reading out the answer. The single instruction, "Read out accumulator," is the only one required. This instruction takes the form RUT and the address code would correspond to ANS. Also, as part of the output, the COJ instruction in step 46 of the basic program causes a jump to step 52, "Read in from core 3." Note the last step in the basic routine, "Store in core 3," becomes unnecessary.

The loading for the complete divide program shown in Table 7-8 is:

Core 1 → 0000
2 → 0000
3 → 0000
4 → 0000
5 → 0001

Table 7-8 lists the entire division program including load data, input/output, and the basic routine. Table 7-9 is a checkout for the divide routine. Use it to verify the program after it is cut and mounted on the drum.

TABLE 7-8. DIVIDE PROGRAM

STEP	MNEMONIC	INSTRUCTION WORD		REMARKS
		ORDER	ADDRESS	
1	RIN	0002	400000	Core 1 $\xrightarrow{M}$ 0000
2	RIN	0002	100000	Core 3 $\xrightarrow{M}$ 0000
3	RIN	0002	021000	Core 5 $\xrightarrow{M}$ 0001
4	RIN	0002	200000	Core 2 $\xrightarrow{M}$ 0000
5	RIN	0002	040000	Core 4 $\xrightarrow{M}$ 0000
6	RIN	0002	000404	IBD-A $\xrightarrow{M}$ Acc (Divisor)
7	STO	0200	200004	Acc $\xrightarrow{M}$ core 2
8	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register
9	RIN	0002	000204	IBD-B $\xrightarrow{M}$ Acc (Dividend)
10	STO	0200	400004	Acc $\xrightarrow{M}$ core 1
11	SHR	0100	100004	2^{-3} Acc $\xrightarrow{M}$ Acc
12	SUB	0400	000000	Acc - X-register $\xrightarrow{M}$ Acc, X-register $\xrightarrow{M}$ 0
13	COJ	0004	010004	If 2^4 is 1, jump to indicated instruction
14	RIN	0002	400104	Core 1 $\xrightarrow{M}$ Acc
15	RUT	0001	000034	Acc $\xrightarrow{M}$ 10°, Read Answer
16	RIN	0006	210104	Core 2 $\xrightarrow{M}$ Acc
17	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register
18	RIN	0002	400104	Core 1 $\xrightarrow{M}$ Acc
19	SHR	0100	200004	2^{-2} Acc $\xrightarrow{M}$ Acc
20	SUB	0400	000000	Acc - X-register $\xrightarrow{M}$ Acc, X-register $\xrightarrow{M}$ 0
21	COJ	0004	004004	
22	STO	0200	040004	Acc $\xrightarrow{M}$ core 4
23	RIN	0002	400104	Core 1 $\xrightarrow{M}$ Acc
24	SHR	0100	200004	2^{-2} Acc $\xrightarrow{M}$ Acc
25	RIN	0002	040104	Core 4 $\xrightarrow{M}$ Acc
26	SHL	0040	200004	2^2 Acc $\xrightarrow{M}$ Acc
27	STO	0200	400004	Acc $\xrightarrow{M}$ core 1
28	RIN	0002	020104	Core 5 $\xrightarrow{M}$ Acc
29	SHL	0040	200004	2^2 Acc $\xrightarrow{M}$ Acc
30	STO	0200	100004	Acc $\xrightarrow{M}$ core 3

TABLE 7-8. DIVIDE PROGRAM (Cont.)

STEP	MNEMONIC	INSTRUCTION WORD		REMARKS
		ORDER	ADDRESS	
31	RIN	0006	204104	Core 2 $\xrightarrow{M}$ Acc
32	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register
33	RIN	0002	400104	Core 1 $\xrightarrow{M}$ Acc
34	SHR	0100	400004	2^{-1} Acc $\xrightarrow{M}$ Acc
35	SUB	0400	000000	Acc - X-register $\xrightarrow{M}$ Acc, X-register $\xrightarrow{M}$ 0
36	COJ	0004	002004	
37	STO	0200	040004	Acc $\xrightarrow{M}$ core 4
38	RIN	0002	400104	Core 1 $\xrightarrow{M}$ Acc
39	SHR	0100	400004	2^{-1} Acc $\xrightarrow{M}$ Acc
40	RIN	0002	040104	Core 4 $\xrightarrow{M}$ Acc
41	SHL	0040	400004	2^{1} Acc $\xrightarrow{M}$ Acc
42	STO	0200	400004	Acc $\xrightarrow{M}$ core 1
43	RIN	0002	020104	Core 5 $\xrightarrow{M}$ Acc
44	SHL	0040	400004	2^{1} Acc $\xrightarrow{M}$ Acc
45	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register
46	RIN	0002	100104	Core 3 $\xrightarrow{M}$ Acc
47	ADD	1000	000000	Acc + X-register $\xrightarrow{M}$ Acc, X-register $\xrightarrow{M}$ 0
48	STO	0200	100004	Acc $\xrightarrow{M}$ core 3
49	RIN	0006	202104	Core 2 $\xrightarrow{M}$ Acc
50	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register
51	RIN	0002	400104	Core 1 $\xrightarrow{M}$ Acc
52	SUB	0400	000000	Acc - X-register
53	COJ	0004	010004	
54	RIN	0002	020104	Core 5 $\xrightarrow{M}$ Acc
55	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register
56	RIN	0002	100104	Core 3 $\xrightarrow{M}$ Acc
57	ADD	1000	000000	A + X-register $\xrightarrow{M}$ Acc, X-register $\xrightarrow{M}$ 0
58	STO	0200	100004	Acc $\xrightarrow{M}$ core 3
59	RIN	0006	110104	Core 3 $\xrightarrow{M}$ Acc
60	RUT	0001	000034	Acc $\xrightarrow{M}$ 10°, Read Answer

TABLE 7-9. DIVIDE PROGRAM CHECKOUT

DIVIDE PROGRAM CHECKOUT
PROCEDURE DIVIDE 12 BY 3
B = 12 = 1100
A = 3 = 0011

STEP	FUNCTION	SIGN	CONTENTS AFTER COMPLETION OF FUNCTION							
			Acc	Ext	X-Reg	Core 1	2	3	4	5
	START		0000	0000	0000	0000	0000	0000	0000	0001
6	RIN A	0	0011							
7	STO 2						0011			
8	TRA				0011					
9	RIN B		1100							
10	STO 1					1100				
11	SHR 3		0001	1000						
12	SUB	1	1110		0000					
13	COJ 16									
14	RIN 1									
15	RUT ANS									
16	RIN 2	0	0011							
17	TRA				0011					
18	RIN 1		1100							
19	SHR 2		0011	0010						
20	SUB		0000		0000					
21	COJ 31									
22	STO 4								0000	
23	RIN 1		1100							
24	SHR 2		0011	0000						
25	RIN 4		0000	0000						
26	SHL 2		0000	0000						
27	STO 1					0000				
28	RIN 5		0001							
29	SHL 2		0100	0000						
30	STO 3							0100		
31	RIN 2	0	0011	0000	0000	0000	0011	0100	0000	0001
32	TRA				0011					
33	RIN 1		0000							
34	SHR 1		0000	0000						

TABLE 7-9. DIVIDE PROGRAM CHECKOUT (Cont.)

STEP	FUNCTION	SIGN	CONTENTS AFTER COMPLETION OF FUNCTION							
			Acc	Ext	X-Reg	Core 1	2	3	4	5
35	SUB	1	1101		0000					
36	COJ 49									
37	STO 4									
38	RIN 1									
39	SHR 1									
40	RIN 4									
41	SHL 1									
42	STO 1									
43	RIN 5									
44	SHL 1									
45	TRA									
46	RIN 3									
47	ADD									
48	STO 3									
49	RIN 2	0	0011							
50	TRA				0011					
51	RIN 1		0000							
52	SUB	1	1101		0000					
53	COJ 59									
54	RIN 5									
55	TRA									
56	RIN 3									
57	ADD									
58	STO 3									
59	RIN 3	0	0100							
60	RUT ANS		0100		ANSWER IS 0100 = 4					

READOUT

Before devising a readout program to convert binary data to BCD for use by the output unit, let's examine the basic problems encountered when we attempt to convert a binary number into a binary-coded decimal number. First, let's re-examine what a binary-coded decimal number is.

The symbols 0 through 9 represent all the decimal numbers. A combination of any one or more of these ten symbols in positional notation will represent a particular numerical value. This numerical value also can be represented using the binary number system symbols

0 and 1. However, if we take each individual decimal symbol and represent it with four binary symbols, we have converted each decimal symbol to its binary-coded decimal (BCD) equivalent.

For example, the number 528 is represented by 0101 0010 1000 in the BCD system. Note the four binary symbols on the left are equal to 5 in the binary notation, the four middle binary symbols are equal to 2, and the four binary symbols on the right are equal to 8. Some representative examples of binary coded numbers, with their decimal and binary equivalents, follow:

TABLE 7-10

DECIMAL NUMBER	BCD NUMBER	BINARY NUMBER
6	0000 0000 0110	000000000110
29	0000 0010 1001	000000011101
57	0000 0101 0111	000000111001
84	0000 1000 0100	000001010100
114	0001 0001 0100	000001110010
143	0001 0100 0011	000010001111
175	0001 0111 0101	000010101111
199	0001 1001 1001	000011000111
212	0010 0001 0010	000011010100
246	0010 0100 0110	000011110110
287	0010 1000 0111	000100011111

Note the binary number in the right-hand column of Table 7-10. There is a significant difference in notation between BCD and pure binary.

Before writing the program, let's examine the limitations of our computer and set up some rules to make the development of the program a little less difficult. The limitations of the computer are:

1. Input to the computer is limited to numbers less than or equal to 9. Therefore, in a multiply program the highest number would be 81; arrived at by multiplying 9×9.
2. Maximum number of bits in a word to be read out is 8. (*Note*: 81 requires only 7 bits; therefore the eighth bit (MSB) is always 0.)
3. Maximum number of instructions on the drum is 65.
4. Maximum number of core locations is 5.
5. Instruction repertoire.

With paper and pencil we could proceed as follows:

1. Put the binary number down in a single column.
2. Assign the proper decimal weight to each 1.

3. Add the total.
4. Put the total down across the page.
5. Place the binary equivalent below each of the decimal symbols.

0	0		
1	64		
0	0		
0	0	6	9
0	0	0110	1001
1	4		
0	0		
1	1		
	69		

The paper and pencil method does not seem to lend itself to programming simply because there is a requirement to convert the binary number to the decimal number, and then convert each decimal number to a BCD number. This is not logical since we wish to obtain the decimal number in the first place. What is required is a direct conversion from binary to BCD without obtaining the decimal number.

Consider the flow chart in Fig. 7-7. In this method we determine whether the number is greater than a particular multiple of 10, and if not, we continue to examine until we have established between what two multiples of 10 our number lies.

The binary to BCD program is a loop-type program that is repeated until the answer is obtained. The program successively subtracts multiples of 10 from the number. The number is read into core memory locations 4 and 5. The program is started by subtracting 80 from

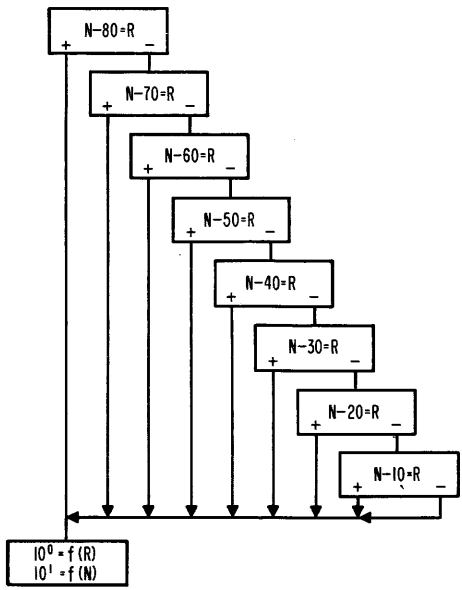


Fig. 7-7

the number and then examining the sign of the remainder. The remaining three core memory locations must be loaded with the following data:

Core	1	1000
	2	0101
	3	0000

Core 1 is equivalent to 8 in binary notation. As the program continues, the core data are reduced by 1 each time through the loop.

Core 2 contains the MSH of 80 and as the program continues core 2 is updated to contain the MSH of the number being subtracted. Core 3 contains the LSH of 80 and is also updated to contain the LSH of the number being subtracted. The flow chart for this program is shown in Fig. 7-8.

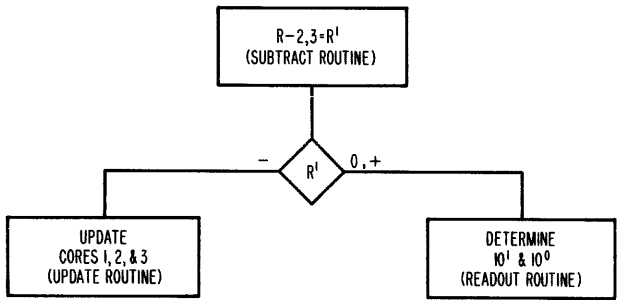


Fig. 7-8

The first action would be to subtract 80 from the number in memory. Since this number is in two core locations, it is not a straightforward subtraction. We must subtract the LSH of 80 from the LSH of the number, examine for a carry and then subtract the MSH of 80 from the MSH of the number. With paper and pencil we accomplish this as follows: let the number in memory be 0100/0101 with 0100 in core 4 and 0101 in core 5. Since the binary equivalent of 80 is 0101 0000, then core 2 is 0101 and core 3 is 0000. Proceed as follows:

0101	*core 5*
— 0000	*core 3*
0101	*ans. in accumulator*

Subtract:

0100	*core 4*
— 0101	*core 2*
1.1111	*ans. is negative, indicating the number is less than 80*

Therefore write the program:

1—Read in from core 3

2—Transfer to X-register
3—Read in from core 5
4—Subtract
5—Shift ans. right 4 times
6—Transfer
7—Read in from core 2
8—Add
9—Transfer
10—Read in from core 4
11—Subtract
12—If MSB is a 1, jump to X
13—Read in core 1
14—Read out as 10^1
15—Shift left $4x$
16—Read out as 10^0

The first four steps set up for subtracting the LSH of 80 from the LSH of the number. The fifth through ninth instructions take into account any carry from the previous subtraction and set up for subtraction the MSH of 80 from the MSH of the number. In the preceding example there was no carry. However, in subsequent subtractions, a carry will be indicated by a 1 in the 2^4 place of the accumulator display. Therefore, by shifting the accumulator four times to the right, this 1 is moved to the LSB of the accumlator, giving 0001 in the accumulator. This 0001 is added to the MSH of the current multiple of 10, which is then subtracted from the MSH of the number. When there is no carry, the accumulator, after being shifted, would contain 0000.

The tenth and eleventh instructions subtract the MSH of the number. If the answer is negative, then instruction 12 directs the program for the case when the number is less than 80. If the answer is positive, then the number is greater than 80 and we can read out the 10^1 and 10^0 results via instructions 13, 14, 15, and 16.

If the number is less than 80, the contents of cores 2 and 3 must be changed and the same subtraction process repeated. The binary equivalent of 70 is 0100 0110 and to set these numbers into core memory one need only subtract 10 from 80.

The readout routine is quite simple when we have core 1 data. Core 1 is loaded initially with 1000 and continually updated to equal the tens digit of the number we are subtracting from the number in cores 4 and 5. Therefore if a number proves to be greater than the number being subtracted, core 1 is read out as the 10^1 BCD number.

Obtaining 10^0 requires shifting the contents of the extension register into the accumulator and reading this out as the 10^0 BCD number.

The update routine advances the values of core 1, core 2, and core 3, so the program may have the loop format. See Table 7-11 for the required data in cores 1, 2, and 3 for each loop of the program.

TABLE 7-11

LOOP	SUBTRACT	CORE 1	CORE 2	CORE 3
1	80	1000	0101	0000
2	70	0111	0100	0110
3	60	0110	0011	1100
4	50	0101	0011	0010
5	40	0100	0010	1000
6	30	0011	0001	1110
7	20	0010	0001	0100
8	10	0001	0000	1010
9	0	0000	0000	0000

The most simple to update is core 1, since this requires a subtraction of 0001 each time through the loop. However, the lack of a 0001 any place in core memory complicates the routine. The only way to get a 1 is to read in any number, shift left, and examine for a 1 in the 2^4 place. If a 1 is present, shift right 4 places. If not, repeat the left shift and examination. Therefore, we program as follows:

1—Read in core 1
2—Shift left one place
3—If MSB is a 1, jump to 5
4—Jump to 2
5—Shift right four places
6—Transfer to X-register
7—Read in core 1
8—Subtract
9—Store in core 1

The updating of core 2 and core 3 is a special program of multiplication. Core 1 is multiplied by ten (1010) as follows:

1—Read in core 1
2—Shift right two places
3—Transfer to X-register
4—Read in core 1
5—Add
6—Shift right one place
7—Store accumulator in core 2
8—Shift left four places
9—Store accumulator in core 3

Core 1 is read in and shifted right. Therefore, the 2^0 and 2^1 bits of the word are in the extension register and the 2^2 bit is in the LS position of the accumulator. This is set into the X-register and core 1 is reread into the accumulator. An addition is made and a single shift right. After this the MSH of the product is in the accumulator

TABLE 7-12. READOUT PROGRAM

STEP	MNEMONIC	INSTRUCTION WORD		REMARKS
		ORDER	ADDRESS	
1	RIN	0002	410000	Core 1 $\xrightarrow{M}$ 1000
2	RIN	0002	100000	Core 3 $\xrightarrow{M}$ 0000
3	RIN	0002	020000	Core 5 $\xrightarrow{M}$ 0000
4	RIN	0002	205000	Core 2 $\xrightarrow{M}$ 0101
5	RIN	0002	040000	Core 4 $\xrightarrow{M}$ 0000
6	RIN	0002	000404	IBR-A $\xrightarrow{M}$ Acc (MSH→Acc)
7	STO	0200	040004	Acc $\xrightarrow{M}$ core 4, Acc→Acc
8	RIN	0002	000204	IBR-B $\xrightarrow{M}$ Acc (LSH→Acc)
9	STO	0200	020004	Acc $\xrightarrow{M}$ core 5, Acc→Acc
10	RIN	0002	100104	Core 3 $\xrightarrow{M}$ Acc
11	TRA	0030	002004	Acc $\xrightarrow{M}$ X-register
12	RIN	0002	020104	Core 5 $\xrightarrow{M}$ Acc
13	SUB	0400	000000	Acc - X-register $\xrightarrow{M}$ Acc, X-register $\xrightarrow{M}$ 0
14	SHR	0100	040004	2^{-4} Acc $\xrightarrow{M}$ Acc
15	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register
16	RIN	0002	200104	Core 2 $\xrightarrow{M}$ Acc
17	ADD	1000	000000	Acc + X-register → Acc, X-register $\xrightarrow{M}$ 0
18	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register
19	RIN	0002	040104	Core 4 $\xrightarrow{M}$ Acc
20	SUB	0400	000000	Acc - X-register → Acc, X-register $\xrightarrow{M}$ 0
21	COJ	0004	010004	If Acc 2^4 = 1, jump as instructed
22	RIN	0002	400104	Core 1 $\xrightarrow{M}$ Acc
23	RUT	0001	000044	Acc $\xrightarrow{M}$ OBR-10^1
24	SHL	0040	040004	2^4 Acc $\xrightarrow{M}$ Acc
25	RUT	0001	000034	Acc $\xrightarrow{M}$ OBR-10°
26	RIN	0006	410104	Core 1 $\xrightarrow{M}$ Acc
27	SHL	0050	401004	sub loop: 2^1 Acc → Acc
28	COJ	0004	004004	sub loop: If 2^4 = 1, jump out
29	JUP	0010	001000	sub loop: Return to start of sub loop
30	SHR	0104	044004	2^{-4} Acc $\xrightarrow{M}$ Acc

TABLE 7-12. READOUT PROGRAM (Cont.)

STEP	MNEMONIC	INSTRUCTION WORD ORDER	INSTRUCTION WORD ADDRESS	REMARKS
31	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register
32	RIN	0002	400104	Core 1 $\xrightarrow{M}$ Acc
33	SUB	0400	000000	Acc - X-register $\xrightarrow{M}$ Acc, X-register $\xrightarrow{M}$ 0
34	STO	0200	400004	Acc $\xrightarrow{M}$ core 1
35	SHR	0100	200004	2^{-2} Acc $\xrightarrow{M}$ Acc
36	TRA	0020	000004	Acc $\xrightarrow{M}$ X-register
37	RIN	0002	400104	Core 1 $\xrightarrow{M}$ Acc
38	ADD	1000	000000	Acc + X-register $\xrightarrow{M}$ Acc, X-register $\xrightarrow{M}$ 0
39	SHR	0100	400000	2^{-1} Acc $\xrightarrow{M}$ Acc
40	STO	0200	200004	Acc $\xrightarrow{M}$ core 2
41	SHL	0040	040004	2^{4} Acc $\xrightarrow{M}$ Acc
42	STO	0200	100004	Acc $\longrightarrow$ core 3
43	JUP	0010	002000	Return to start of loop

TABLE 7-13. READOUT PROGRAM CHECKOUT

STEP	FUNCTION	SIGN	CONTENTS AFTER COMPLETION OF FUNCTION Acc	Ext	X-Reg	Core 1	2	3	4	5
10	RIN 3	0	0000	0000	0000	1000	0101	0000	0100	1101
11	TRA		↓	↓	0000	↓	↓	↓	↓	↓
12	RIN 5		1101	↓	↓	↓	↓	↓	↓	↓
13	SUB		1101	↓	↓	↓	↓	↓	↓	↓
14	SHR 4		0000	1101	↓	↓	↓	↓	↓	↓
15	TRA		↓	↓	0000	↓	↓	↓	↓	↓
16	RIN 2		0101	↓	↓	↓	↓	↓	↓	↓
17	ADD		0101	↓	↓	↓	↓	↓	↓	↓
18	TRA		↓	↓	0101	↓	↓	↓	↓	↓
19	RIN 4		0100	↓	↓	↓	↓	↓	↓	↓
20	SUB	1	1111	↓	0000	↓	↓	↓	↓	↓
21	COJ 26		↓	↓	↓	↓	↓	↓	↓	↓

TABLE 7-13. READOUT PROGRAM CHECKOUT (Cont.)

STEP	FUNCTION	SIGN	CONTENTS AFTER COMPLETION OF FUNCTION							
			Acc	Ext	X-Reg	Core 1	2	3	4	5
22	RIN 1									
23	RUT 10^1									
24	SHL 4									
25	RUT 10^0									
26	RIN 1	0	1000							
27	SHL 1	1	0001	1010						
28	COJ 30									
29	JUP 27									
30	SHR 4	0	0001	0001						
31	TRA				0001					
32	RIN 1		1000							
33	SUB		0111		0000					
34	STO 1					0111				
35	SHR 2		0001	1100						
36	TRA				0001					
37	RIN 1	0	0111	1100	0001	0111	0101	0000	0100	1101
38	ADD		1000		0000					
39	SHR 1		0100	0110						
40	STO 2						0100			
41	SHL 4		0110	0000						
42	STO 3							0110		
43	JUP 11									
11	TRA				0110					
12	RIN 5		1101							
13	SUB		0111		0000					
14	SHR 4		0000	0111						
15	TRA				0000					
16	RIN 2		0100							
17	ADD		0100							
18	TRA				0100					
19	RIN 4		0100							
20	SUB		0000		0000					

TABLE 7-13. READOUT PROGRAM CHECKOUT (Cont.)

STEP	FUNCTION	SIGN	CONTENTS AFTER COMPLETION OF FUNCTION							
			Acc	Ext	X-Reg	Core 1	2	3	4	5
21	COJ 26		↓							
22	RIN 1		0111							
23	RUT MSH 10^1		0111	↓						
24	SHL 4		0111	0000						
25	RUT LSH 10^0	↓	0111	↓	↓	↓	↓	↓	↓	↓

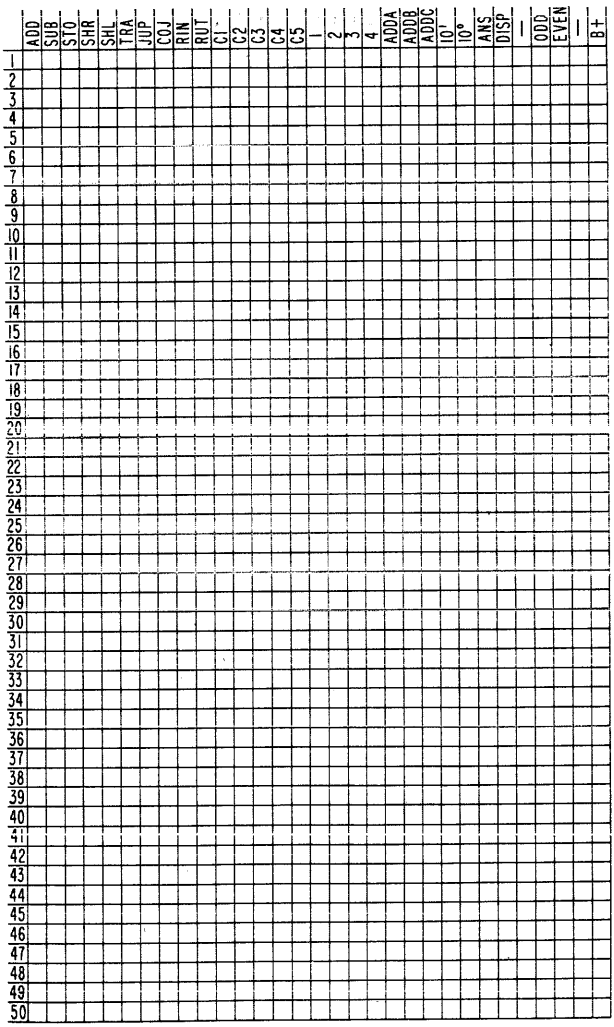


Fig. 7-9

and the LSH is in the extension register. The last three instructions read this data into core 2 and core 3.

Now the program may be written in its entirety as shown in Table 7-12.

Table 7-13 is a check-out chart for the readout routine. The number 77 is loaded into cores 4 and 5 in binary form (0100 and 1101). Use this chart to check out your routine, since it gives step by step results as you step through the program.

FORMULATING THE PROGRAM FOR TRANSFER TO DRUM

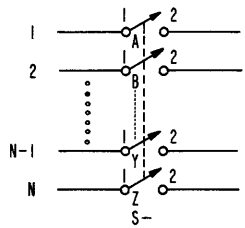
Select an 11 $\times$ 14 in. piece of graph paper with five squares per inch and lay out as shown in Fig. 7-9. Using the octal address in the listed program, score an x in the proper square.

When complete, cut each square and wrap the form around the drum (see Chapter 5).

APPENDIX—BUILD YOUR OWN SWITCHES

The multipole switches required in the computer consists of single-throw and double-throw.

The multipole, single-throw switch, Fig. A, is used to open and close a number of circuit lines simultaneously. The term single-throw indicates that only one terminal can be connected to each contact arm when the switch is in the closed position. Each pole of the switch has an alphabetical designation; i.e., A, B, C, etc. The terminal connected to the contact arm is numerically designated as terminal 1. The other terminal is designated as terminal 2.



WHERE N = THE NUMBER OF CIRCUIT LINES AND POLES.

NOTE: SWITCHES ARE SHOWN IN THE OPEN POSITION

Fig. A.

Construction of the single-throw, multipole switch requires the following material:

Wood screws

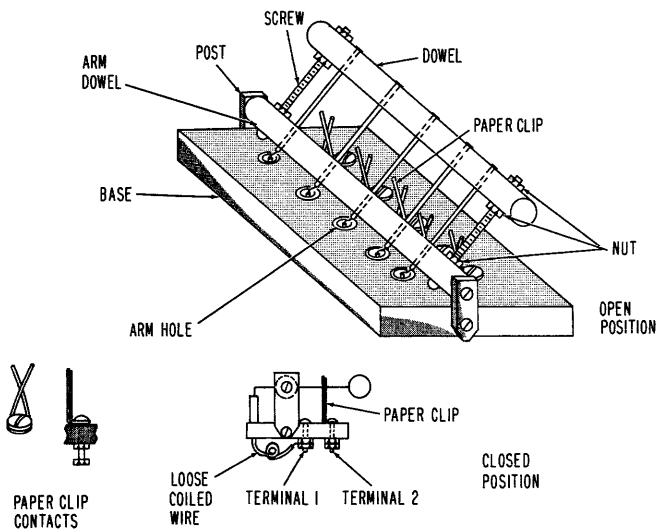


Fig. B.

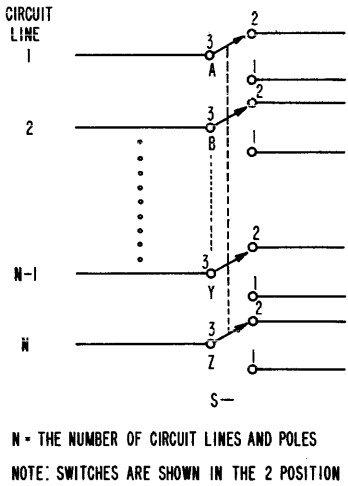


Fig. C.

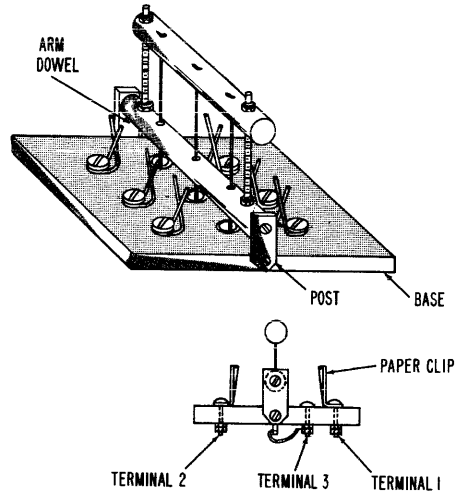


Fig. D.

¼-in. round doweling
Paper clips
Assorted hardware
Insulated hook-up wire.

Refer to Fig. B and build your switch. Note the following:

1. The arm dowel must rotate, therefore use small wood screws with a smooth shaft below the head and make your holes in the posts slightly larger than the wood screw diameter.
2. Make the armholes in the base equal in length to the distance between the center of the arm dowel and the free end of the paper clip.

The multipole, double-throw switch, Fig. C, is used to connect a single terminal to either of two other terminals. The term *double-throw* indicates that either one of two terminals can be connected to each contact arm. Each pole of the switch has an alphabetical designation; i.e., A, B, C, etc. The terminal connected to the contact arm is numerically designated as terminal 1, the other terminals are designated 2 and 3. To be consistent throughout construction of your computer, always designate that terminal to the left or top as terminal 2, and the terminal to the right or bottom as terminal 3. If this is kept in mind, then wiring and interpretation of schematics will be simplified. Also, when wiring double-throw switches to select 0 or 1 in the computer units, designate terminal 2 as the 0 position and terminal 3 as the 1 position.

Construction of the multipole, double-throw switch is similar to the single-throw construction, except the switch now must be swung to either side of the posts. Refer to Fig. D and build your switches.

ARITHMETIC UNIT WIRING LIST

S1-1 to Disp
S1-2 to S10J-1

S2A-1 to S12A-1
-2 to S12A-2
-3 to S2D-3
B-1 to S12B-2
-2 to S12B-1
-3 to S10L-3
C-1 to S12C-1
-2 to S12F-1
-3 to B+
D-1 to S12C-2
-2 to S12D-1
-3 to S2A-3 S2E-3
E-1 to S12E-1
-2 to S12E-2
-3 to S2D-3, S10I-3
F-1 NC
-2 NC
-3 NC

S3A-1 to S13A-1
-2 to S13A-2
-3 to S3D-3
B-1 to S13B-2
-2 to S13B-1
-3 to S3F-3
C-1 to S13C-1
-2 to S13F-1
-3 to B+
D-1 to S13C-2
-2 to S13D-1
-3 to S3A-3 S3E-3
E-1 to S13E-1
-2 to S13E-2
-3 to S3D-3 S10H-3
F-1 to S13J-1
-2 to S13J-2
-3 to S3B-3 S3G-3
G-1 to S13H-2
-2 to S13H-1
-3 to S10K-3
H-1 to S13I-2
-2 to S13G-2
-3 to B+

S4A-1 to S14A-1
-2 to S14A-2
-3 to S4D-3
B-1 to S14B-2
-2 to S14B-1
-3 to S4F-3
C-1 to S14C-1
-2 to S14F-1
-3 to B+
D-1 to S14C-2
-2 to S14D-1
-3 to S4A-3 S4E-3
E-1 to S14E-1
-2 to S14E-2
-3 to S4D-3 S10B-3
F-1 to S14J-1
-2 to S14J-2
-3 to S4B-3 S4G-3
G-1 to S14H-2
-2 to S14H-1
-3 to S4F-3 S10C-3
H-1 to S14I-2
-2 to S14G-2
-3 to B+

S5A-1 to S15A-2
-2 to S15A-1
-3 to S5B-3
B-1 to S15B-3
-2 to S15C-3
-3 to S5A-3 S5C-3
C-1 to S15E-2
-2 to S15D-2
-3 to S5B-3 S15D-3

S6-1 to $L2^{-1}$
-2 to B+

S7-1 to $L2^{-2}$
-2 to B+

S8-1 to $L2^{-3}$
-2 to B+

S9-1 to $L2^{-4}$
-2 to B+

S10A-1 to S15A-2
-2 to S15A-3
-3 to $L2^0$
B-1 to DISP
-2 to S11A-3
-3 to S4E-3
C-1 NC
-2 to S11B-3
-3 to S4G-3
D-1 to DISP
-2 to B+
-3 to S15E-3
E-1 to S14A-1
-2 to S14B-3
-3 to $L2^1$
F-1 to S13A-1
-2 to S13B-3
-3 to $L2^2$

S10G-1 to S12A-1
-2 to S12B-3
-3 to $L2^3$

S10H-1 to DISP
-2 to S11C-3
-3 to S3E-3
I-1 to DISP
-2 to S11D-3
-3 to S2E-3
J-1 to S1-2
-2 to S11E-3
-3 to $L2^4$
K-1 NC
-2 to S11F-3
-3 to S3G-3
L-1 NC
-2 to S11G-3
-3 to S2B-3

S11A-1 to S15B-1
-2 to S15C-1
-3 to S10B-2
B-1 to S15D-2
-2 to S15E-2
-3 to S10C-2
C-1 to S14D-3
-2 to S14F-3
-3 to S10H-2
D-1 to S13D-3
-2 to S13F-3
-3 to S10I-2
E-1 to S12D-3
-2 to S12F-3
-3 to S10J-2
F-1 to S14H-3
-2 to S14J-3
-3 to S10K-2

S11G-1 to S13H-3
-2 to S13J-3
-3 to S10L-2

S12A-1 to S2A-1 S10G-1
-2 to S2A-2
-3 to S12B-3
B-1 to S2B-2
-2 to S2B-1
-3 to S12A-3 S10G-2
C-1 to S2C-1
-2 to S2D-1
-3 to S12D-3
D-1 to S2D-2
-2 NC
-3 to S12C-3 S11E-1
E-1 to S2E-1
-2 to S2E-2
-3 to S12F-3
F-1 to S2C-2
-2 NC
-3 to S12E-3 S11E-2

S13A-1 to S3A-1 S10F-1
-2 to S3A-2
-3 to S13B-3
B-1 to S3B-2
-2 to S3B-1
-3 to S13A-3 S10F-2
C-1 to S3C-1
-2 to S3D-1
-3 to S13D-3
D-1 to S3D-2
-2 NC
-3 to S13C-3 S11D-1
E-1 to S3E-1
-2 to S3E-2
-3 to S13F-3
F-1 to S3C-2
-2 NC
-3 to S13E-3 S11D-2
G-1 NC
-2 to S3H-2
-3 to S13H-3
H-1 to S3G-2
-2 to S3G-1
-3 to S13G-3, S11G-1
I-1 NC
-2 to S3H-1
-3 to S13J-3
J-1 to S3F-1
-2 to S3F-2
-3 to S13I-3 S11G-2

S14A-1 to S4A-1 S10E-1
-2 to S4A-2
-3 to S14B-3
B-1 to S4B-2
-2 to S4B-1
-3 to S14A-3 S10E-2
C-1 to S4C-1
-2 to S4D-1
-3 to S14D-3
D-1 to S4D-2
-2 NC
-3 to S14C-3 S11C-1
E-1 to S4E-1
-2 to S4E-2
-3 to S14F-3
F-1 to S4C-2
-2 NC
-3 to S14E-3 S11C-2
G-1 NC
-2 to S4H-2
-3 to S14H-3
H-1 to S4G-2
-2 to S4G-1
-3 to S14G-3, S11F-1

S14I-1 NC
-2 to S4H-1
-3 to S14J-3
J-1 to S4F-1
-2 to S4F-2
-3 to S14I-3 S11F-2

S15A-1 to S5A-2
-2 to S5A-1 S10A-1
-3 to S10A-2
B-1 to S11A-1
-2 NC
-3 to S5B-1
C-1 to S11A-2
-2 NC
-3 to S5B-2
D-1 NC
-2 to S5C-2 S11B-1
-3 to S5C-3 S15E-3
E-1 NC
-2 to S5C-1 S11B-2
-3 to S15D-3 S10D-3

Index