HEATHKIT
CONTINUING
EDUCATION

Individual Learning Program

In

DIGITAL TECHNIQUES

9 DIGITAL DESIGN

Heathkit
ZENITH Educational Systems

UNIT 9

DIGITAL DESIGN

CONTENTS

UNIT 9

DIGITAL DESIGN

INTRODUCTION

In this unit, you are going to learn how to design digital circuits. You will apply your knowledge of digital circuit operation and application to develop a digital circuit to solve a particular problem or meet a particular need. First, we will show you how to establish a design criterion for the equipment under consideration. Then we will consider procedures for designing both combinational and sequential logic circuits. These step-by-step procedures will lead you from a problem definition to a completed electronic circuit. The emphasis will be on implementing your design with modern integrated circuits. A variety of design examples will be given to show you some of the many applications for digital circuits.

Refer to the unit objectives that completely define the content and purpose of this unit. Then go to the unit activity guide and follow the steps indicated there to complete this unit.

UNIT OBJECTIVES

When you complete this unit you will be able to:

1. State the major criterion for the design of digital circuits.
2. List several design criteria for digital circuits.
3. Define a digital design problem and write a set of specifications for the circuit required to solve that problem.
4. Develop a truth table defining the circuit design.
5. Write the logic equation expressing the operation of a combinational logic circuit from the truth table.
6. Use Karnaugh maps to minimize Boolean equations.
7. Select the appropriate circuitry to implement the design equations.
8. List several important trade-offs regarding the selection of SSI, MSI, ROMs, and PLAs.
9. Design a combinational logic circuit for a given application.
10. Design special counter and controller circuits for implementing sequential logic applications.

UNIT ACTIVITY GUIDE

	Completion Time
☐ Play record 7, side 1	_____
☐ Read section Design Criteria	_____
☐ Answer Self Test Review questions 1 and 2.	_____
☐ Read section Combinational Logic Circuit Design.	_____
☐ Answer Self Test Review questions 3 and 4.	_____
☐ Perform Experiment 23.	_____
☐ Read section Sequential Logic Circuit Design.	_____
☐ Answer Self Test Review Question 5.	_____
☐ Perform Experiment 24.	_____
☐ Complete Unit Examination.	_____

DESIGN CRITERIA

The first step in designing a digital circuit is to define what the circuit must accomplish. You can do this by outlining the circuit specifications. These details will accurately specify the purpose of the circuit and the desired performance. The remaining design steps will convert this set of specifications into a practical working circuit that meets the design objectives. In designing the circuit there must be some standard for evaluating your design. In other words, there must be some criterion for determining whether you have adequately met your design objectives. In digital equipment design, as in the design of almost any type of electronic equipment, the primary design criterion is to achieve maximum performance for the lowest cost. This broad general criterion is made up of many parts which define what we mean by maximum performance and lowest cost. Together these make up the design criteria for the equipment. Let's see what each of these important considerations mean.

Maximum Performance

The term maximum performance can have a variety of meanings depending upon the circuit or equipment being designed. The definition of maximum performance therefore is a direct function of the application. Some of the factors that make up maximum performance include operating speed, accuracy, size, power consumption, reliability, and the number of unique features.

If we are going to market this equipment then we want to make it as desirable as possible in terms of cost, performance, and unique features. We want it to be marketable and competitive. These are only a few of the factors that define what maximum performance is. Each of these must be defined by itself to match the specific application. As you can see, what we want to do is to develop the best piece of equipment possible for our investment in design and production time and money.

Lowest Cost

Our design objective is to achieve the maximum performance possible for the lowest cost. The cost means both materials and time. A low cost design will have fewer parts. Fewer parts means a smaller printed circuit board on which to mount them and lower cost. The fewer the number of parts in a design the greater the reliability.

Another important factor is design and production time. The cost of any product includes the time required to design and produce it. The faster and easier the circuit is to design, the less it costs. In addition, the fewer the number of hours required to construct and test that unit the lower its cost.

Trade–Offs

A design that achieves maximum performance for lowest cost is an efficient design. The time and money spent in developing the unit is minimal while we achieve the benefit of maximum performance. By letting our criterion of maximum performance for lowest cost be our primary design goal, you must realize that in practical situations trade offs are generally necessary. This means it is not always possible to achieve the very highest of performance for the lower cost. As a general rule, high performance costs more money. If very high performance standards must be met, then we must accept the fact that the penality we will pay is higher cost. For example, to achieve the highest operating speed, higher cost digital integrated circuits must be used. This higher speed also generally requires a higher power consumption. Therefore, to achieve the highest possible speed, we are sacrificing cost and power consumption. Increased accuracy generally requires higher quality circuits and in some cases a greater quantity of circuitry. Again, higher costs, higher power consumption will be sacrificed. Increasing the number of components also increases the production time and reduces the reliability. High performance designs in addition, are generally more sophisticated and complex. This, in turn, means a greater design time. Therefore, an attempt to achieve maximum performance will invariably increase cost. For that reason you must be ready to trade off higher performance and features in order to obtain a lower cost. The design procedure is basically one of juggling the performance requirements and the costs to achieve a desired performance level for the lowest possible cost. You are seeking an efficient, middle-of-the-road solution.

There are many situations where it is desirable to select high performance as the single, most desirable design characteristic. It may be absolutely necessary to achieve the desired level of performance regardless of the cost. In another application, just the opposite may be true. Instead of optimizing your design for high performance you might want to optimize it for lowest possible cost. In order to achieve the low cost, high performance and features will naturally be sacrificed.

While optimizing your design for maximum performance or lowest cost is sometimes necessary, most design projects will be in that middle ground where your job will be to balance performance and cost to achieve an acceptable level in both.

Self Test Review

1. The primary design criterion for digital circuits is to achieve highest _____ for lowest _____.
2. List five characteristics of digital equipment that are usually affected by the trade-offs made to achieve an optimum design.
   a.
   b.
   c.
   d.
   e.

Answers

1. performance, cost
2. a. cost
   b. speed
   c. accuracy
   d. power consumption
   e. reliability

Also, size, weight, features, design and production time.

COMBINATIONAL LOGIC CIRCUIT DESIGN

In this section we are going to give you a step-by-step procedure that can be used to design virtually any combinational logic circuit. The procedure will lead you from the basic problem definition to the completed circuitry using modern integrated circuits. The steps in this design procedure are:

1. Problem definition
2. Truth table development
3. Writing logic equations
4. Minimizing the logic equations
5. Selecting the circuitry and implementing the design

Each of these steps will be described in detail. Design examples will be used to illustrate these steps. In addition, you will learn an important technique for minimizing logic equations. This technique involves the use of a Karnaugh map. This is a method of putting the logic equations into a graphical form that permits rapid circuit minimization without the use of Boolean algebra. Finally,a variety of design problems will be given to permit you to practice this procedure yourself. Now, let's take a look at the steps in the design procedure.

Problem Definition

The first step in the design of any combinational logic circuit is complete problem definition. This means that you must thoroughly identify all functions of the circuit. You will know from the specific application what operations the circuit must perform. Your initial job will be to outline them completely.

The best and most thorough method of problem definition is to write out a complete description of the application and the desired functions to be accomplished. While it may seem unnecessary to put this information in writing, by doing so it forces you to completely identify and explain what is going to take place. In this description you will identify the types and number of input signals to the circuit. You will also identify the type and number of outputs to be produced by the circuit. Make your circuit description as complete as possible. The exact form of the description is not important. It can be a descriptive narrative in paragraph form. Alternately it can simply be a list of inputs, outputs and functions to be performed.

Once you have completed the functional description of your circuit, make up a table of specifications. This table of specifications will duplicate some of the information contained in your circuit description. However, the information will be more concisely stated. The specifications will list number and type of inputs, number and type of outputs, desired operating speed, desired power consumption, a cost objective, a size and weight objective, types of integrated circuits to be used, interface requirements for both the inputs and outputs including logic level specifications, and any other information pertinent to the operation of the circuit. Regardless whether your design is to be a circuit within a larger system or a complete digital system itself; the functional description, problem definition and complete set of specifications will give you all of the information necessary to complete the design.

The process of writing down the circuit description and its specifications is a valuable excercise. It forces you to think through the problem and to identify it as carefully as possible. In preparing this information you will discover many things that you may not have thought of initially. Problem definition is more than just a busy work exercise. It is a vital first part of the design process and the success or failure of the design can be traced almost directly to the thoroughness of this problem definition.

Another benefit of complete problem definition and specification outlining is that you will have a complete set of documentation for your circuit or system that can be used later in preparing instructional manuals for the equipment, engineering reports, journal articles, and other requirements for this information.

To illustrate this concept of problem definition, and the other steps of a design procedure, we are going to take a typical design example and follow it through each of the design steps. A circuit example will be a simple one to start with in order to help you easily grasp the concepts involved. Later, several more detailed examples will be given. In addition, practice problems are provided to permit you to practice these steps yourself.

Example Problem. The design objective is to develop a combinational logic circuit that will monitor a four bit binary word and generate a binary 1 output signal when any one of the six invalid four bit states in the 8421 BCD code occur.

Specifications. The detailed specifications for the circuit described above are as follows:

1. Four bit parallel binary input word
2. Signal output that will be a binary 1 state when any one of the six invalid BCD code numbers occur.

3. Use TTL integrated circuitry with standard TTL logic levels of binary 0 = +0.4 volts and binary 1 = +3.5 volts.
4. Speed of operation: Propagation delay of this circuit shall be less than 100 nanoseconds. In other words, the output will become binary 1 in 100 nanoseconds or less from the time the input code is any one of the six invalid BCD values.
5. Minimize cost, size and power consumption. It is desirable to have the entire circuit contained within a single dual inline package IC.

Truth Table Development

The next step in the design procedure is to convert your problem description into a truth table. The truth table as you will recall is a chart that completely identifies all possible input combinations and the corresponding logic output states. The truth table completely defines the operation of the circuit. The truth table can be developed directly from your problem description and specifications.

The first step in developing the truth table is to determine the number of inputs to the logic circuit. This, of course, is a function of the application and this information should have been defined in the problem description. The number of inputs will determine the maximum number of input states that can occur with this number of variables. The total number of states that can occur is equal to 2^n where n is the number of inputs. For example, if you have defined four circuit inputs, there are a total possible number of $2^4 = 16$ different states that can occur. Depending upon the application, all or possibly only some of the total number of possible states may occur. In the problem description these should be identified.

Begin the construction of the truth table by writing down all possible binary input states. You can do this by simply listing the binary numbers from zero through the maximum upper limit which is a function of the number of input states. For four inputs and a maximum of 16 possible states, you will simply list all four bit binary numbers 0000 through 1111 in sequence. This completely defines all possible input states.

In columns adjacent to the input states in the table, record the output variables specified by the problem. In these columns, identify the desired output states for each input combination. If some of the input states are not used, identify them as being invalid or "don't care" states even though they are not used or needed.

Figure 9-1 shows the truth table for the BCD invalid code detector. The input is a parallel four bit word. We label each of the input bits with a letter for identification. Remember that any input lettering or numbering scheme can be used to suit the application. Short mnemonic names designating the signal or its function can be used. Alpha-numeric combinations can also be used. For this application, the letters A, B, C, and D are adequate. The output of the circuit is designated F. This is the logic signal that will be a binary 1 if an invalid code is detected.

INPUTS	OUTPUT
A B C D	F
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	0
0 1 0 0	0
0 1 0 1	0
0 1 1 0	0
0 1 1 1	0
1 0 0 0	0
1 0 0 1	0
1 0 1 0	1
1 0 1 1	1
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1

Figure 9-1
Truth table for BCD invalid code detector.

Notice in the truth table that all sixteen possible combinations of four bits are listed. The first ten states 0000 through 1001 are the valid 8421 BCD codes. Since these are valid, the output F will be binary 0. For the inputs 1010 through 1111, the output F is binary 1 signaling an invalid code. There are no unused or "don't care" states. The truth table completely defines the operation of the circuit.

While our example here has only a single binary output, other combinational logic circuits may have multiple outputs. In this case the truth table will define these as well. A separate column for each output will be provided in the truth table.

Develop the Logic Equations

The next step in the design process is to write the Boolean logic equations from the truth table. This will put the logic function into a form where it can be manipulated with Boolean algebra. This will allow you to reduce the logic equation using Boolean techniques and thus minimize the amount of circuitry required to implement it. For some applications it may not be necessary to minimize the equation with Boolean algebra. Instead the equation will simply be used as a guide in implementing the function depending upon the types of circuits to be used.

To write the logic equation from the truth table, you observe the outputs column in the truth table and write a product term of the inputs for each output where a binary 1 state occurs. The result will be a sum-of-products logic equation. This process will lead to a single Boolean equation for each output.

Observing the BCD invalid code detector truth table in Figure 9-1 you can write the output equation.

$$F = A\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,C\,D + A\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,D + A\,B\,C\,\overline{D} + A\,B\,C\,D$$

At this point it is possible to implement the Boolean equation directly with logic circuits. AND and OR gates can be combined to perform this function. However, in most cases it is desirable to use Boolean algebra or other means to reduce the equation to a simpler form. This can have the

result of minimizing the number of gates and integrated circuit packages used in a design. Minimization generally leads to lower cost, smaller size, and reduced power consumption.

Circuit Minimization

By using the Boolean algebra techniques described in a previous unit, the logic equations you developed from the truth table can be reduced.

Shown below is the step-by-step procedure for using Boolean algebra to minimize the logic equation developed in the previous step. Keep in mind that this is only one of several approaches that can be used. Depending upon how you group the various logic terms, the individual steps may be different. In any case, your resulting reduced logic equation should be the same.

$$F = A\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,C\,D + A\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,D + A\,B\,C\,\overline{D} + A\,B\,C\,D$$

Reduce by factoring

$$F = A\,\overline{B}\,C\,(\overline{D} + D) + A\,B\,\overline{C}\,(\overline{D} + D) + A\,B\,C\,(\overline{D} + D)$$

Reduce by Law of Complements and Law of Intersection

$$F = A\,\overline{B}\,C + A\,B\,\overline{C} + A\,B\,C$$

Reduce by factoring

$$F = A\,\overline{B}\,C + A\,B\,(\overline{C} + C)$$

Reduce by Law of Complements and Law of Intersection

$$F = A\overline{B}C + AB$$

Reduce by factoring

$$F = A(\overline{B}\,C + B)$$

Reduce by the Law of Absorption $\overline{B}C + B = B + C$

$$F = A\,(B+C)$$

Expand by multiplying

$$F = AB + AC$$

As you can see a significant reduction in the equation takes place. It is obvious that it requires much less circuitry to implement the reduced version than it does the original equation derived from the truth table. This minimization step is very important to the design of the circuit.

The use of Boolean algebra is time consuming and burdensome. For some logic equations, a reduction can take place quickly without a lot of work. However, for large complex equations, the minimization process can require a substantial amount of time. You may have to rearrange the equation and regroup the terms several times before you arrive at a minimum result. In addition, the Boolean algebra procedure described in an earlier unit does not always lead to the optimum minimization. Because of the subtlety of logic circuits, some methods of circuit reduction do not show themselves in the equation reduction process. For that reason, Boolean algebra has its limitations. Other forms of minimization have been developed to provide the maximum amount of minimization possible and to do so quickly and conveniently. One of these techniques is known as Karnaugh maps.

Karnaugh Maps

A Karnaugh map is a graphical method of minimizing logic equations. The equations describing a digital logic function can be broken up and arranged in such a way that they form a map or illustration that permits rapid reduction or simplification. The Karnaugh map is an alternative to the use of Boolean algebra for minimizing logic expressions. In fact, the Karnaugh map is preferred over Boolean algebra because it makes the reduction process faster, easier, and more effective. This technique completely eliminates the need for using Boolean algebra and allows you to translate the logic function directly from the truth table into a Karnaugh map that then leads to the simplified form. With this technique, it is not always necessary to write the equations from the truth table first.

Karnaugh maps effectively replace Boolean logic equations in the sum-of-products form. For design purposes, these equations are derived from the truth table as we described earlier. Each of the product terms in the equation is referred to as a minterm. Each minterm is the product of the various input variables which are called literals. In the truth table describing the logic function, all possible input combinations are listed. For example, for a 2-input logic circuit, there are four possible input combinations. They are 00, 01, 10, and 11. If the input literals are given the names A and B, then the minterms are $\overline{A}\,\overline{B}$, $\overline{A}\,B$, $A\,\overline{B}$, and $A\,B$.

Instead of writing the product term with respect to the literals, they are often simply expressed as the letter *m* with a subscript equal to the decimal value of the binary number representing that minterm. For example, the product term $\overline{A}\,\overline{B}$ represents input states of 00. The minterm designation then would be m_0. The product term $A\,B$ represents the input states 11. The decimal equivalent of this number is 3, therefore, the

minterm designation would be m_3. Figure 9-2 lists the product terms for a two input logic circuit, their binary and decimal equivalents and the minterm designation.

DECIMAL	BINARY A	BINARY B	PRODUCT TERM	MINTERM DESIGNATION
0	0	0	$\overline{A}\ \overline{B}$	m_0
1	0	1	$\overline{A}\ B$	m_1
2	1	0	$A\ \overline{B}$	m_2
3	1	1	$A\ B$	m_3

Figure 9-2
Minterm designations for logical products of two literals.

A Karnaugh map takes this information and translates it into graphical form. Figure 9-3 shows a Karnaugh map for a two input logic circuit. Since there are two input variables, there are four possible product terms. Each product term is represented by a cell or square in the map.

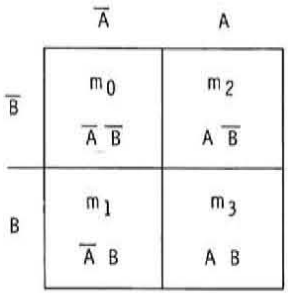


Figure 9-3
Two-variable Karnaugh map.

To show how the map and the equation are related, let's take several examples of converting from the equation to the map and from a map to the equation.

Consider the Boolean equation: $C = \overline{A}\ B + A\ \overline{B}$

This is the equation for an exclusive OR circuit. Note that it contains two product terms of the two variables A and B. To plot this equation on the map, we simply place a binary 1 in those squares representing the product terms in the equation. This is shown in Figure 9-4. The minterm designations are not generally included within the squares on the map. Instead, the squares themselves can be identified by referring to the designations above and to the left of the map. At the top of the map are the designations $\overline{A}$ and A which refer to the two vertical columns. At the left of the map are the designations $\overline{B}$ and B which designate the two rows of squares in the map. The product term corresponding to a square is identified by simply reading upward and to the left for the letter designation defining that term and using these letters to form the product term.

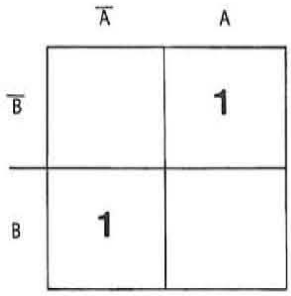


Figure 9-4
Karnaugh map for the exclusive OR function $A\overline{B} + \overline{A}B$.

In this example, we translated a known equation into Karnaugh map form. Keep in mind that the map can also be developed directly from the truth table. The output column of the truth table is observed and those input states corresponding to binary 1 outputs are translated into product terms that can then be plotted on the map.

As an example of translating from a Karnaugh map into the equivalent Boolean equation, consider the map shown in Figure 9-5. To write the output expression corresponding to this map, you develop a minterm for each square containing a binary 1. These minterms are then ORed together to form a sum-of-products Boolean equation. The map in Figure 9-5 designates the exclusive NOR function expressed by the equation:

$$C = \overline{A}\,\overline{B} + A\,B$$

We can also write this equation using the minterm designations:

$$C = m_0 + m_3$$

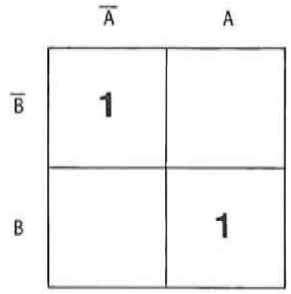


Figure 9-5
Karnaugh map for exclusive OR function $\overline{A}\overline{B} + AB$.

Figure 9-6 shows the Karnaugh map for a three variable logic circuit. Since there are three input variables, there are eight possible input combinations. Each input state is represented by a square in the map. The minterm for each square in the map is designated. The relationships between the product terms, their binary and decimal equivalents and the minterm designation are given in Figure 9-7. Note that the columns and rows in the map are designated by the letters corresponding to the inputs.

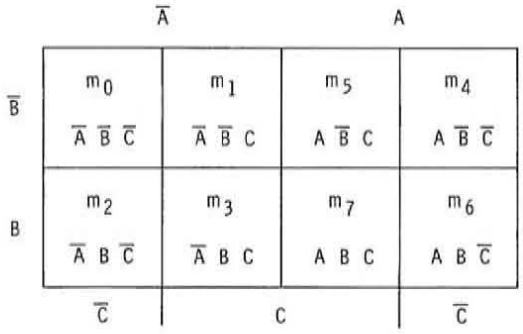


Figure 9-6
Three—variable Karnaugh map.

DECIMAL	BINARY A	B	C	PRODUCT TERM	MINTERM DESIGNATION
0	0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$	m_0
1	0	0	1	$\overline{A}\,\overline{B}\,C$	m_1
2	0	1	0	$\overline{A}\,B\,\overline{C}$	m_2
3	0	1	1	$\overline{A}\,B\,C$	m_3
4	1	0	0	$A\,\overline{B}\,\overline{C}$	m_4
5	1	0	1	$A\,\overline{B}\,C$	m_5
6	1	1	0	$A\,B\,\overline{C}$	m_6
7	1	1	1	$A\,B\,C$	m_7

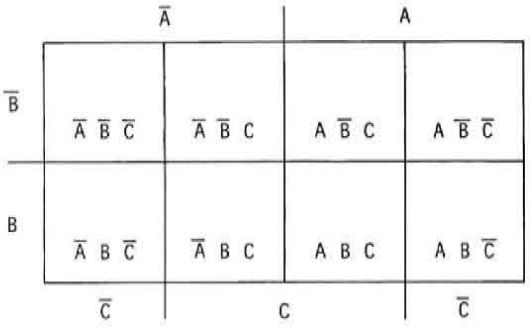


Figure 9-7
Minterm designations for logical products of three literals.

The two right-hand vertical columns are designated A, the two left-hand vertical columns are designated $\overline{A}$. Note that these vertical columns are designated in a different form by the input variable C. The two center vertical columns represent C while the two outside columns represent $\overline{C}$. The two horizontal rows of four squares are designated $\overline{B}$ and B. The minterm represented by each square can be determined by simply writing a product term made up of the three letters designating that square in its row and column position. It takes three input designations to define the coordinates of a square.

The method of recording a given Boolean equation in the Karnaugh map for three variables is similar to that for two variables. Consider the equation:

$$X = A\,\overline{B}\,C + A\,\overline{B}\,\overline{C} + \overline{A}\,B\,C$$

Each three-variable term is designated by a binary 1 in the appropriate square. See Figure 9-8.

The Karnaugh map in Figure 9-9 shows how an equation can be written from the map. The minterm represented by each square where a binary 1 appears is summed (logically ORed) with the other terms to produce the equation:

$$M = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,B\,C$$

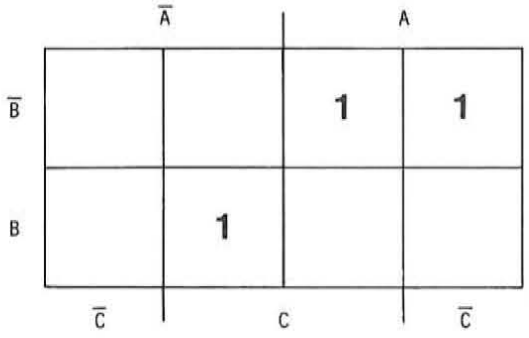


Figure 9-8
Karnaugh map for the equation
$X = A\overline{B}C + A\overline{B}\overline{C} + \overline{A}BC$

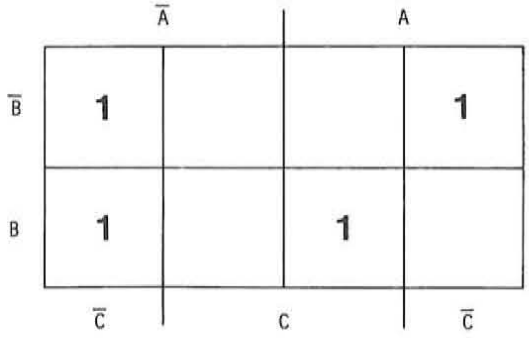


Figure 9-9
Karnaugh map of the equation
$M = \overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$

In minterm form the equations are:

$$M = m_0 + m_2 + m_4 + m_7$$

These same concepts can be applied to logic expressions involving four variables. The table in Figure 9-10 shows all sixteen possible combinations of four-input variables. All product terms with their binary and decimal equivalents are given.

DECIMAL	BINARY A B C D	PRODUCT TERM	MINTERM DESIGNATION
0	0 0 0 0	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$	m_0
1	0 0 0 1	$\overline{A}\,\overline{B}\,\overline{C}\,D$	m_1
2	0 0 1 0	$\overline{A}\,\overline{B}\,C\,\overline{D}$	m_2
3	0 0 1 1	$\overline{A}\,\overline{B}\,C\,D$	m_3
4	0 1 0 0	$\overline{A}\,B\,\overline{C}\,\overline{D}$	m_4
5	0 1 0 1	$\overline{A}\,B\,\overline{C}\,D$	m_5
6	0 1 1 0	$\overline{A}\,B\,C\,\overline{D}$	m_6
7	0 1 1 1	$\overline{A}\,B\,C\,D$	m_7
8	1 0 0 0	$A\,\overline{B}\,\overline{C}\,\overline{D}$	m_8
9	1 0 0 1	$A\,\overline{B}\,\overline{C}\,D$	m_9
10	1 0 1 0	$A\,\overline{B}\,C\,\overline{D}$	m_{10}
11	1 0 1 1	$A\,\overline{B}\,C\,D$	m_{11}
12	1 1 0 0	$A\,B\,\overline{C}\,\overline{D}$	m_{12}
13	1 1 0 1	$A\,B\,\overline{C}\,D$	m_{13}
14	1 1 1 0	$A\,B\,C\,\overline{D}$	m_{14}
15	1 1 1 1	$A\,B\,C\,D$	m_{15}

Figure 9-10
Minterm designations for all possible combinations of four literals.

A four variable Karnaugh map is shown in Figure 9-11. Each square in the map represents one of the four-variable minterms. The process of recording a given sum-of-products equation on the map is similar to the process described earlier for two- and three-input variables. In addition, the

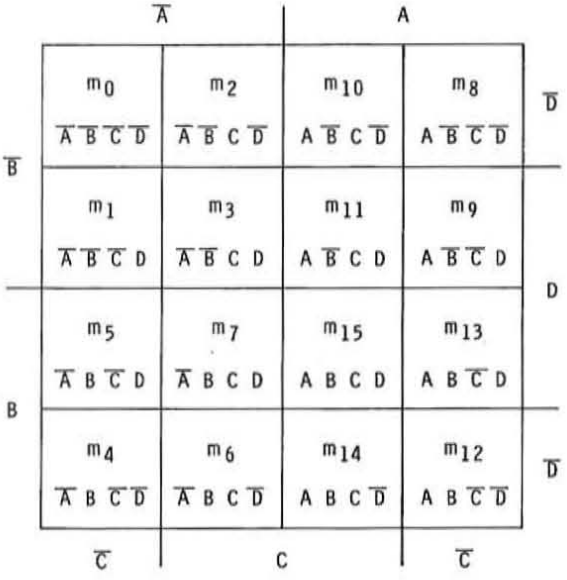


Figure 9-11
Four-variable Karnaugh map.

procedure for writing the equation from the map is also similar to that described before. As an example, the map shown in Figure 9-12 represents the equation below:

$$X = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,C\,D + \overline{A}\,B\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{C}\,\overline{D} + A\,\overline{B}\,C\,D$$

$$X = m_0 + m_3 + m_4 + m_8 + m_{11}$$

Match each term in the equation to the appropriate square in the map to be sure that you understand how the two are related.

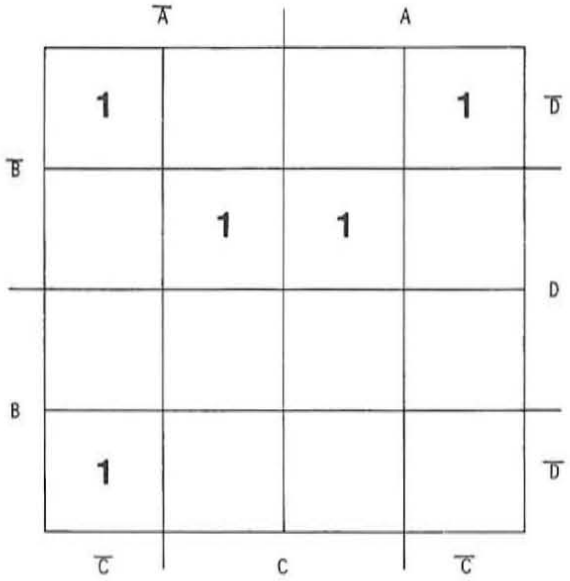


Figure 9-12
Karnaugh map for the equation $X = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}CD + \overline{A}B\overline{C}\overline{D} + A\overline{B}CD + A\overline{B}\overline{C}\overline{D}$

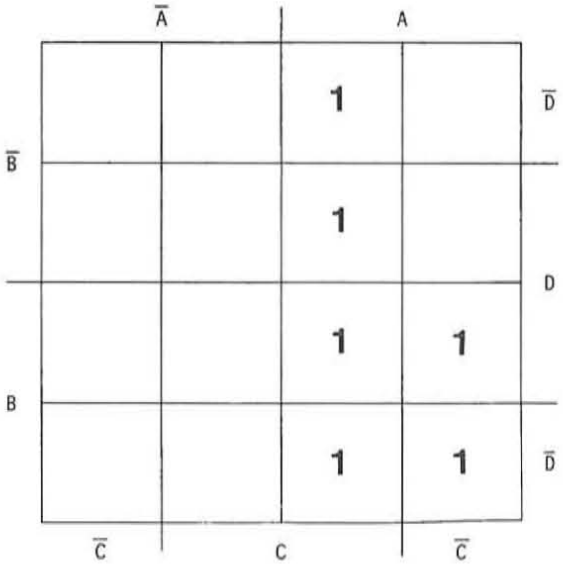


Figure 9-13
Karnaugh map for the BCD invalid code detector.

In designing the BCD invalid state detector circuit, you wrote the Boolean equations from the truth table. These are:

$$F = A\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,C\,D + A\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,D + A\,B\,C\,\overline{D} + A\,B\,C\,D$$

$$F = m_{10} + m_{11} + m_{12} + m_{13} + m_{14} + m_{15}$$

These can be placed on a Karnaugh map as show in Figure 9-13.

Now that you know how Boolean equations are plotted on Karnaugh maps and how to read a Karnaugh map and translate it into a Boolean equation, you are ready to see how these maps can be used for circuit minimization.

The reduction of logic equations with Boolean algebra is largely by use of the law of complements ($A + \overline{A} = 1$). Putting the logical function in sum-of-product equation form, the minterms can then be grouped to permit the factoring out of common variables. This procedure generally produces law of complement expressions for one of the input variables. Thus, one of the input variables is eliminated from the group of minterms from which it was factored thereby simplifying the expression. Nearly all of the simplification that results from the use of Boolean algebra comes from the use of factoring and the reduction by the law of complements. The Karnaugh map effectively implements this technique in a graphical form.

Refer to the four-variable Karnaugh map in Figure 9-14. If you will study this map carefully, you will see that adjacent cells or adjacent minterms differ by only one of the input variables. In other words, only one variable will change in moving from one adjacent cell to the next, horizontally or

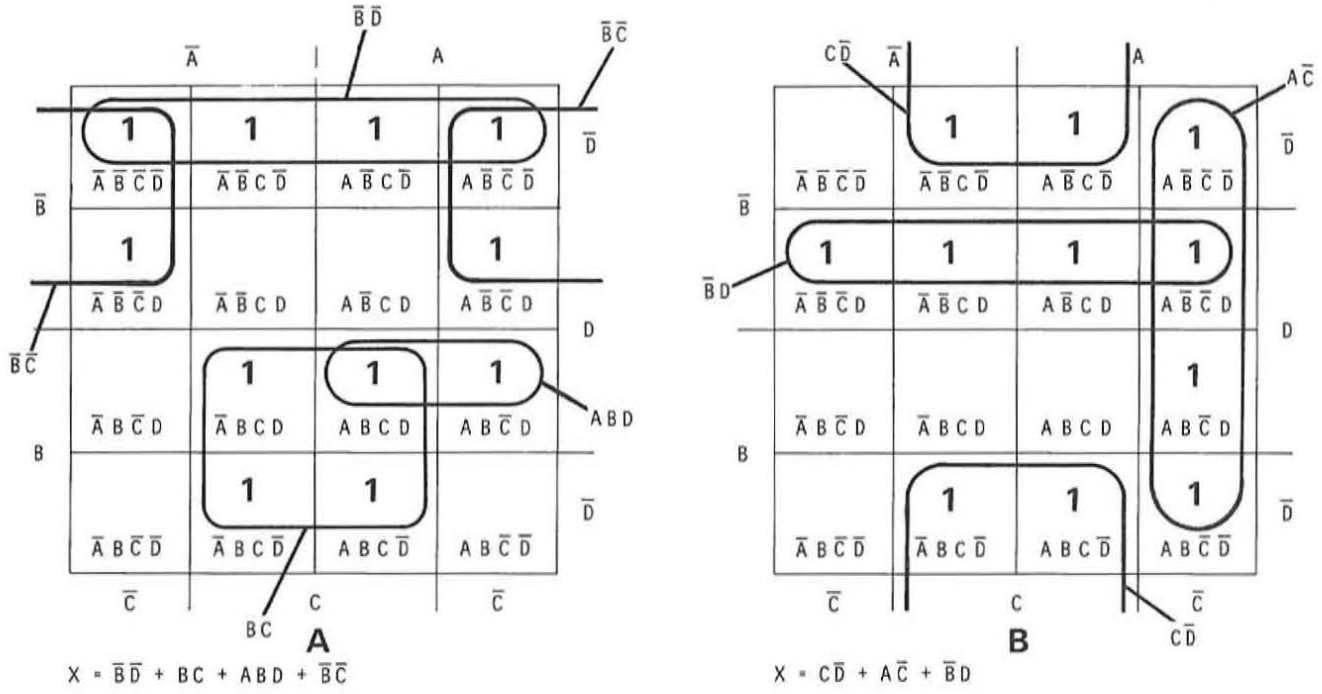


Figure 9-14
Four-variable Karnaugh map.

vertically. For example, consider the two vertical minterms identified by binary 1's in the map of Figure 9-15. In moving from the upper cell to the lower cell, only the B variable changes, from $\overline{B}$ to B. The A and C variables do not change. These two adjacent minterms specify a simplification that can be made. You can see this by writing the Boolean equation of the minterms recorded. Assume that the sum-of-product equations of these two minterms is equal to the function Y. The equation from the map in Figure 9-15, then is $Y = A\,\overline{B}\,C + A\,B\,C$.

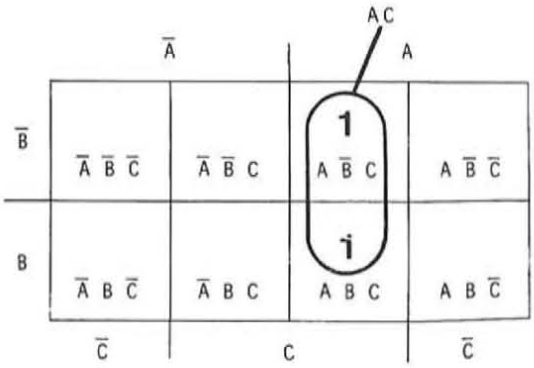


Figure 9-15
Adjacent cells in a Karnaugh map differ by only one literal.

Now, use Boolean algebra to simplify this expression. This is done as shown below.

$$Y = A\,\overline{B}\,C + A\,B\,C$$
$$Y = A\,C\,(\overline{B} + B)$$
$$Y = A\,C\,(1)$$
$$Y = AC$$

Note that AC was factored out of each minterm leaving an expression equal to $(\overline{B} + B)$. Since this is equal to 1, the expression is considerably simplified. The B literal simply drops out leaving the expression Y = AC. Logic equation minimization with a Karnaugh map is based on this concept.

The basic procedure for reducing a logic equation by a Karnaugh map is to first map the expression by putting a binary 1 in each cell representing the minterms in the sum-of-products logic equation. Then, horizontal and vertical adjacencies in groups of two or four are identified. We then note which variables change from one cell to the next in each set of grouped adjacent terms. These inputs then drop out of the expression. The remaining input terms are regrouped in sum-of-product forms to produce the simplified expression. An example will illustrate this process.

Consider the logic expression:

$$Y = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C + \overline{A}\,B\,\overline{C} + A\,\overline{B}\,C$$
$$Y = m_0 + m_1 + m_2 + m_5$$

To simplify this logic expression we first record the minterms on a Karnaugh map. Since there are three-input variables, an eight-cell Karnaugh map will be used. Binary 1's are entered in those cells representing the minterms in the equation. This is indicated in Figure 9-16.

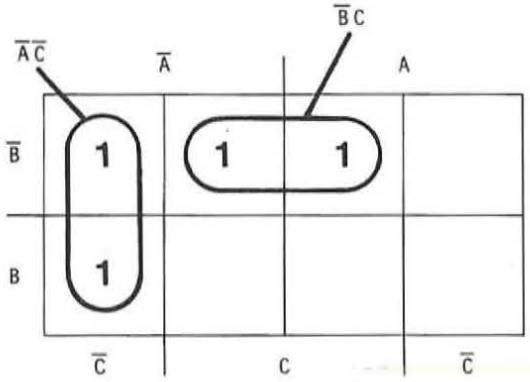


Figure 9-16
Using a Karnaugh map to minimize the expression $Y = m_0 + m_1 + m_2 + m_5$

Next, adjacent minterms are grouped by some power of 2 (2, 4, 8, etc.). Each group of two or four minterms is identified by a circle enclosing the binary 1's on the map as indicated in Figure 9-16.

Each circled group is then observed to determine which variable changes in moving from one adjacent cell to the next in the group. In the vertical group of Figure 9-16, the variable B changes in moving from the upper cell to the lower cell. This indicates then that the B term drops out leaving only the $\overline{A}$ and $\overline{C}$ terms. Therefore, this group of two adjacent variables represent the logic expression $\overline{A}\,\overline{C}$.

Observing the horizontal grouping of two variables in Figure 9-16, we see that the variable that changes in moving from one cell to the next is the A variable. The A term therefore drops out leaving the $\overline{B}$ and C terms. Once the variable that changes has been identified, a new shorter minterm is developed from the variables that have not changed. A product term of the variables that do not change is formed, in this case $\overline{B}\,C$. These shorter product terms for each group are then summed (ORed) to produce the reduced expression. Therefore, the reduced expression from the map in Figure 9-16 is:

$$Y = \overline{A}\,\overline{C} + \overline{B}\,C$$

The ability to use a Karnaugh map to produce a minimum equation reduction results from being able to properly group the minterms and recognize all of the adjacencies or combinations of adjacencies.

Figure 9-17 shows two additional examples of mapping logic equations, grouping minterms and generating the reduced expression. In Figure 9-17A, the Boolean equation is:

$$X = A\,\overline{B}\,C + A\,\overline{B}\,\overline{C} + A\,B\,\overline{C}$$

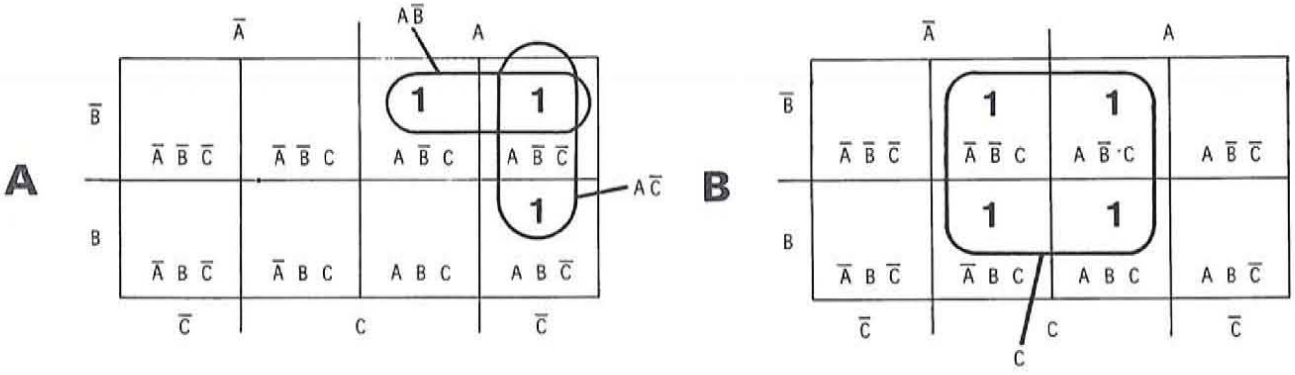


Figure 9-17
Examples of reducing three variable logic equations with maps.

The three minterms are recorded in the appropriate cells in the map by marking 1s in the appropriate cells. Adjacent minterms are then grouped to identify the redundant input terms. Note that the minterm $A\,\overline{B}\,\overline{C}$ is used twice. Any given minterm may be used as many times as needed to form adjacent groups of two or four.

To determine the redundant input variables, you note which variable changes when moving from one cell to the next within the groups you formed. The horizontal group of two identifies the change of variable $\overline{C}$. In moving from one cell to the next the A and $\overline{B}$ terms do not change but the C term does. This means that the C term drops out. We then form a new product term made up of the variables that did not change, in this case $A\,\overline{B}$.

Next, we observed the vertical group of two to determine the redundant variable. In this case the redundant variable is B since in moving from one cell to the next within that vertical group the B term changes from B to $\overline{B}$. The A and $\overline{C}$ terms do not change therefore they represent the new product term for use in the minimized expression. The new product terms are then logically summed or ORed to produce the output expression:

$$X = A\overline{B} + A\,\overline{C}$$

As you can see from Figure 9-17A the original and reduced expressions are considerably different. The reduced expression is far more economical in the use of circuitry.

Figure 9-17B illustrates another example of minimizing a three-variable Boolean equation. This equation is:

$$X = \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + A\,\overline{B}\,C + A\,B\,C$$
$$X = m_1 + m_3 + m_5 + m_7$$

Groups of two or four adjacent functions are then formed as shown. It is desirable to form the largest grouping possible with the minterms recorded on the map. The larger the grouping, the greater the reduction that will take place. Note in the group of four that, in moving from one adjacent cell to the next, the A and B terms change. In moving vertically, the B terms change. Moving horizontally, the A terms change. The only input variable that does not change from one of these four cells to the next is the C input term. This means that A and B inputs are redundant and can be factored out of the equation and dropped. This 4-bit grouping then results in a substantial minimization and simply represents the input variable C. What this grouping tells us is that the output expression will be affected only by the variable C regardless of the A and B input states. The reduced output expression then is:

$$X = C$$

The power of the Karnaugh map is evident from these examples. With a little practice in mapping and grouping the minterms, you can quickly reduce logic expressions to their minimum form. The map provides a visual means of recognizing patterns in the minterm groupings so that redundancies in the input variables can be easily recognized and eliminated, thereby leaving only the essential input terms to implement the function.

The benefit of the Karnaugh map in speeding up and simplifying logic equation reduction becomes more evident as more input variables are used. Four-input variables produce sixteen different input states. These can be combined in a variety of ways to form logic equations. Expressions involving minterms of four variables or more are difficult to work with by using standard Boolean algebra techniques. But by mapping them, you automatically group the related minterms so that the redundancies can be readily identified.

Figures 9-18A through 9-18D show several examples of the use of four-variable Karnaugh maps. The reduced equations for each example are given. Study the various groupings of minterms to be sure you understand how the reduced expression is obtained for each group. As you study each example, keep in mind these important facts:

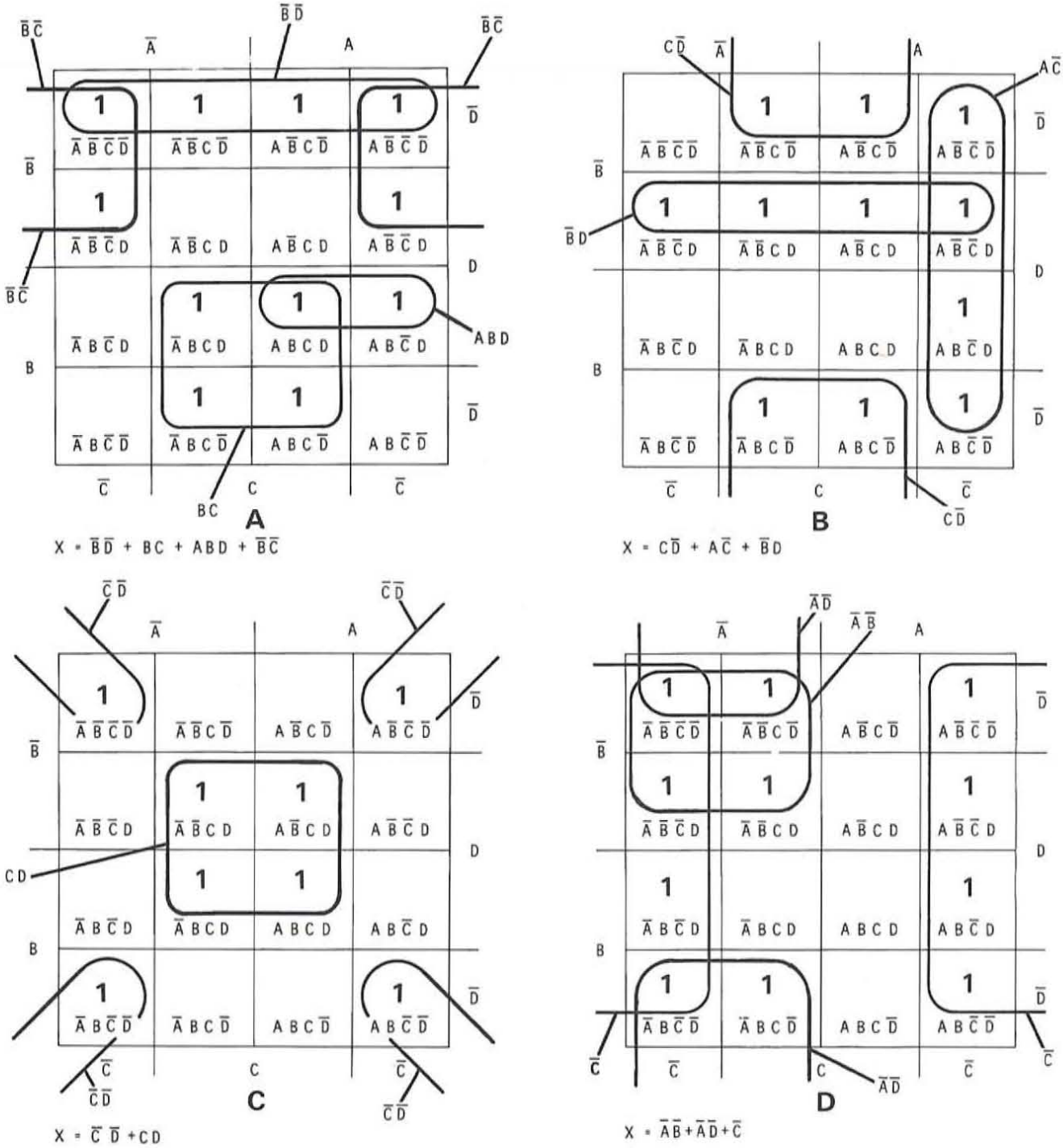


Figure 9-18
Examples of Karnaugh map usage.

1. Adjacent minterms are grouped by twos, fours, eights, and higher powers of 2 as required. The groupings may be horizontal or vertical and may involve adjacent terms that do not "appear" adjacent in the map. In a sixteen cell, four-variable Karnaugh map, adjacencies or redundancies can occur in the cells in the left and right most columns. Assume that the map is formed into a cylinder where the left and right edges are made adjacent. In the same way a cylinder can be formed by causing the upper and lower edges of the map to be made adjacent. The term "redundant" simply means that in moving from one cell to an adjacent cell only one of the input variables change.
2. Try to use each minterm in a group of two or four. After you have made your initial groupings, go back and study them to be sure that you have not overlooked various combinations of two, for example, which would be made into a single logic grouping of four. The larger the number of minterms enclosed within a loop the greater the reduction that results. There will be some occasions where a minterm cannot be included in a group of two, four or eight. In such cases no reduction is possible and the minterm must be treated by itself.
3. In each group of two or four variables, simply move from one cell to the next noting which variables change. The variables that change are redundant and can be dropped out of the minterm. Form a new product expression using the variables that do not change.
4. The minimized expression is formed by producing a sum-of-products expression made up of the reduced product expressions resulting from each group of minterms.

Keeping in mind the rules and characteristics just discussed, the examples in Figure 9-18 should be self-explanatory. There is one special case, however, that you may not easily recognize. In figure 9-18C the minterm in each corner of the map is marked. Since minterms on the left and right columns and in the upper and lower rows are considered to be adjacent, the minterms in each corner of the map can be considered as a single group of four. This is more easily seen by determining which variables change in moving from one corner cell to the next. As you can see, the A and B variables change in moving between these four cells. The $\overline{C}$ and $\overline{D}$ variables, however, are common to these cells. This permits a reduction of this group to the simple two-variable term $\overline{C}\,\overline{D}$.

Another special case that may occur is when all squares in the map are marked. In this case the function represented by the map is simply a binary 1.

Summary of Karnaugh Map Usage. The list below is a summary of the rules and procedures for using Karnaugh maps in reducing logic equations.

1. Study the truth table or the logic equations for the function to be minimized. Determine the number of input variables and construct a Karnaugh map containing a number of cells equal to two raised to a power equal to the number of input variables.
2. Map the minterms directly from the truth table. If you constructed a truth table as part of your design procedure there is no need to translate the truth table into a Boolean equation first. If you have an equation instead of a truth table, plot the minterms in the map from the equation itself.
3. Group the minterms in the map in units of two, four, and eight terms. Try to include each minterm in the largest group possible to ensure a minimum solution. Each minterm should be used at least once and can be used as many times as necessary to form the groups to produce a minimum result. Identify each group of two, four or eight terms by enclosing them within a loop or circle.
4. Note the input variables that change in moving from one minterm to the next in each group. The variables that change are redundant and drop from the expression. Another way of looking at this is to observe the variables within each group that remain the same in moving from one adjacent cell to the next within that group. Use these variables to form a product expression that will appear in the reduced equation.
5. Once you have determined a product expression for each group of minterms on the map, write the final output expression by ORing together the product terms developed for each group.

In Figure 9-19 we show the Karnaugh map used in reducing the Boolean expression for our BCD invalid code detector. Note that we can identify two groups of four variables. The reduced equation is much simpler than the equation we wrote from the truth table given earlier.

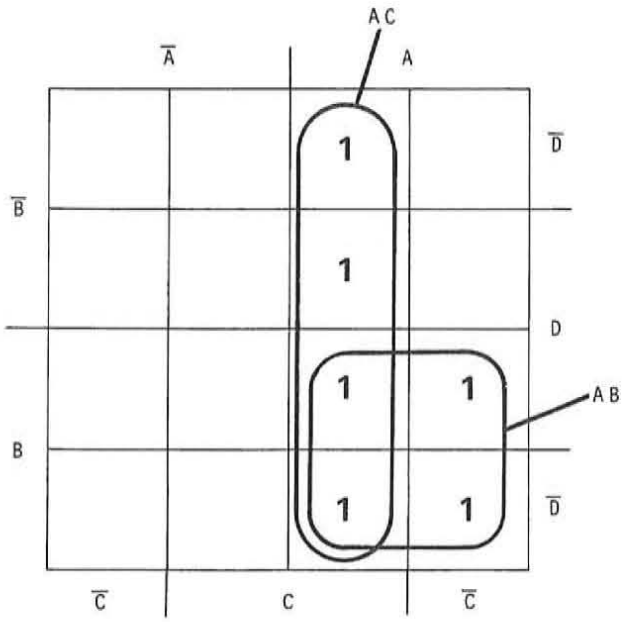


Figure 9-19
Karnaugh map for reducing the equation for a BCD invalid code detector.

Don't Care" States. There are some design situations where all combinations of the input variables will not occur. For example, you may have identified the need for four input variables and your design calls for the use of only seven of the sixteen possible input states. You can usually determine by the application which input combinations can never occur. In other situations there are various combinations of input states which will not effect the operation of the circuit and therefore you do not care whether they occur or not. It is useful to identify these "don't care" states. In most applications you should have no difficulty in determining what these "don't care" states are. Such states are of value in minimizing the logic expression through the use of a Karnaugh map. The "don't care" states are plotted on the Karnaugh map along with the minterms specified by the truth table or the equation. In most cases they will aid in the reduction of the circuitry required to implement the desired function.

To illustrate this concept, assume that your design calls for four-input variables and the output function is indicated by the equation below.

$$M = \overline{A}\,\overline{B}\,C\,D + \overline{A}\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,C\,D + A\,B\,C\,\overline{D} + A\,B\,C\,D$$

Figure 9-20A shows how this function is plotted on a Karnaugh map. The variables are grouped to reduce the amount of circuitry required to implement the function. This greatly simplifies the function as you can see by the reduced equation below.

$$M = \overline{A}\,\overline{B}\,C + \overline{B}\,C\,D + A\,B\,C$$

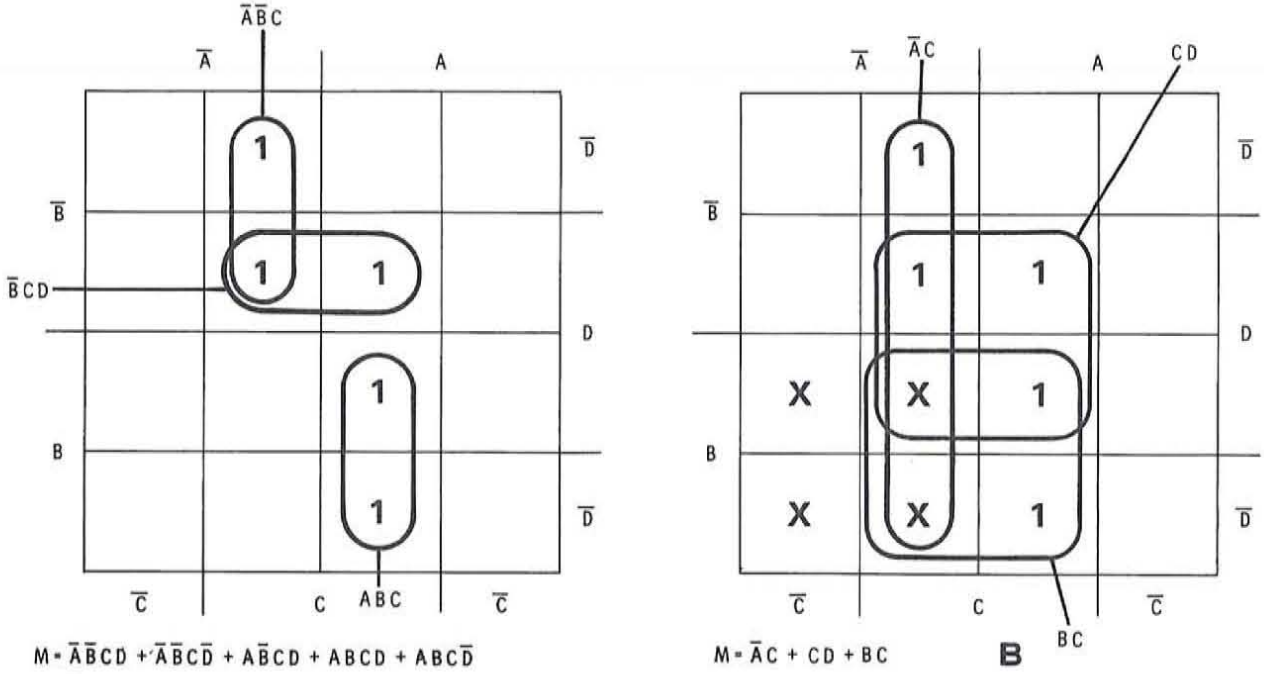


Figure 9-20
Example of the use of "don't care" states in minimizing logic functions.

In your design process assume you determine that there are four "don't care" states. These are: $\overline{A}\,B\,\overline{C}\,\overline{D}$, $\overline{A}\,B\,\overline{C}\,D$, $\overline{A}\,B\,C\,D$, and $\overline{A}\,B\,C\,\overline{D}$. These four "don't care" states can then be plotted on the map as shown in Figure 9-20B. The "don't care" states are indicated by X's instead of binary 1's used to designate the required minterms. You can now use the "don't care" states along with the designated minterms to produce further circuit reductions. The X's can be grouped along with the 1s to form larger loops. The more minterms that you can include within a group, the greater the resulting minimization. As you can see, three groups of four can be formed. This resulting equation is:

$$M = \overline{A}\,C + C\,D + B\,C$$

This equation is far simpler than either the original equation or the first reduced version. When designing a combinational logic circuit, don't forget to make every attempt to identify these "don't care" states since significant reductions in circuit size and complexity can result.

Implementing the Logic Equations

You are now ready to select the circuitry to implement your design. Remember that your basic goal is to perform the desired function for the lowest possible cost. This means you will select the lowest price available circuitry. You will attempt to minimize the number of components in the design. This not only lowers the cost, but will also reduce power consumption, size and weight and increase reliability.

In all new equipment design situations you will be using integrated circuits. There are very few if any applications where benefits can be derived by using discrete component circuitry. Therefore, our discussion here is limited to selecting the types of integrated circuits appropriate to your design.

There are four practical ways to implement a combinational logic function with integrated circuits. They are:

1. SSI
2. MSI
3. ROM
4. PLA

Each of these approaches has its own benefits and limitations. In the sections to follow we will discuss each of these methods of implementation. We will use the BCD invalid code detector circuit as an example in evaluating each of these methods.

SSI Implementation. The most direct method of implementing your logic equations is to use SSI logic gates. Then by working from the equation derived from the truth table or the minimized version from the Karnaugh map, implement the circuit with available NAND and NOR gates. This literal approach is best employed when the function to be implemented is simple. For larger more complex functions some of the other techniques should be used.

To illustrate the use of SSI circuits in implementing our BCD invalid code detector, consider the original Boolean equation:

$$F = A\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,C\,D + A\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,D + A\,B\,C\,\overline{D} + A\,B\,C\,D$$

This expression is readily implemented with SSI logic circuits as shown in Figure 9-21A. Each product term requires a four-input gate. Dual four-input TTL gates such as the 7420 can be used. One gate will be used for each minterm expression in the equation. All of these product terms will then be ORed together to produce the output function F. And since there are six terms, a six-input OR gate is required. An eight-input TTL gate such as the 7430 can be used for this purpose. Two of the inputs will not be used and can be simply connected to one of the other inputs. Note that the four-input variables must come from a source where both the normal and complement signals are available. If the complements are not available then they can be generated with inverters as shown in Figure 9-21B. A standard TTL circuit is the 7404 hex inverter containing six inverter circuits. Only four of these are needed in this application. As you can see, it requires a minimum of four and possibly five TTL integrated circuit packages to implement this function. These ICs must be mounted on a printed circuit board and the interconnection pattern on the circuit board must be designed. This will take a substantial amount of time, and the printed circuit board required to hold these circuits will be fairly large.

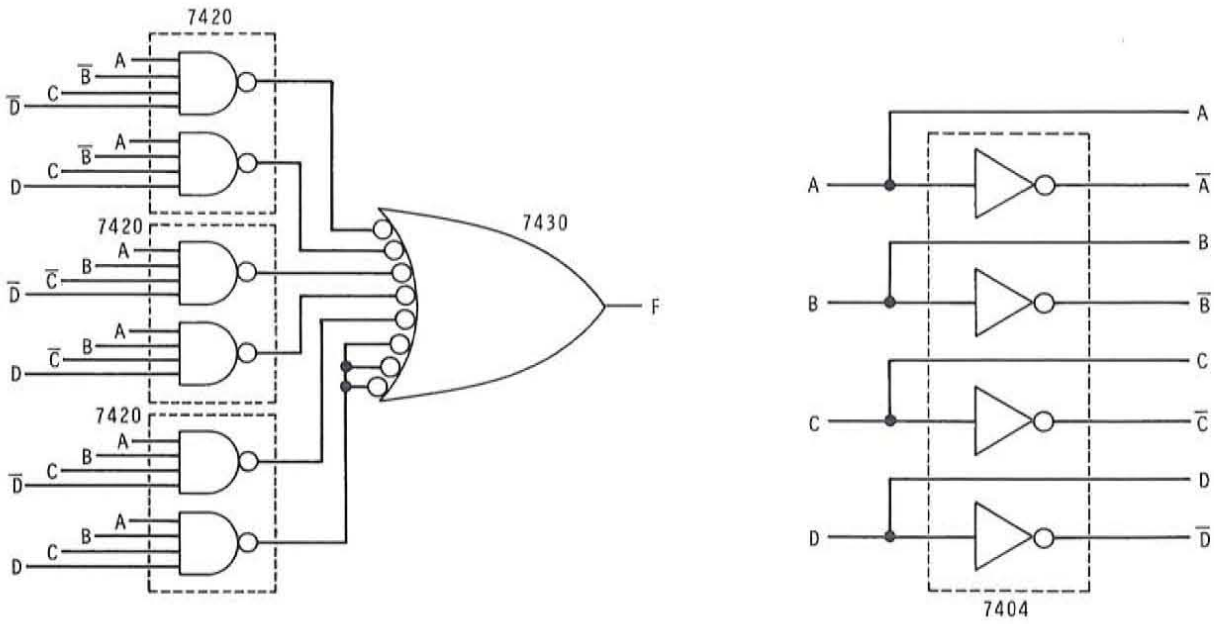


Figure 9-21
BCD invalid code detector circuit using SSI implementation of the equation.

Of course it is ridiculous to implement our BCD invalid code detector circuit as we have indicated. You have already shown that by the use of the Karnaugh map you can reduce the original equation to the simplified expression:

$$F = A\,C + A\,B$$

Figure 9-22 shows how this equation can be implemented with SSI circuits. Only three two-input logic gates are required. This means that a standard quad two-input NAND gate such as the TTL 7400 can be used to implement this expression. As a result we have implemented our detector circuit with a single integrated circuit. This reduces the package count, power consumption and printed circuit board design time as well as size and weight. Note that the input variable D is not even required in implementing this function. Yet, this simple circuit will generate the same truth table as the more complex circuit in Figure 9-21. The value of minimization by the use of a Karnaugh map is evident in this example.

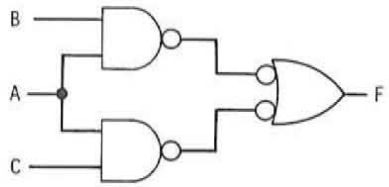


Figure 9-22
BCD invalid code detector circuit.
Minimum SSI circuit.

MSI Implementation. There are a variety of MSI functional circuits that can be used to implement combinational logic circuit designs. While these MSI circuits are designed to perform common combinational logic functions, they can often be adapted to perform other functions. Their use can result in a simplified, low cost method of implementing a logic expression. It is an alternative which should be thoroughly considered when designing digital circuits.

The two most useful MSI circuits for implementing logic equations are the decoder and the multiplexer or data selector. A decoder or 1 of N detector circuit accepts a number of logic inputs and recognizes all possible combinations of the input states. This is done by using AND or

NAND gates to detect each of the possible input conditions. Figure 9-23A shows a one-of-sixteen decoder circuit. It features four inputs which are then decoded by NAND gates to produce 16 outputs. A typical commercial version of this circuit is the 74154 TTL decoder. It is housed in a 24-pin dual in-line package. You can see from the figure that this decoder is a minterm generator. All 16 possible states are generated within the single IC thereby eliminating the need to interconnect external SSI gates and inverters. Note that in this circuit active low outputs are generated. To obtain active high outputs, inverters can be used on each output or the decode gates can be combined with other logic gates to produce the proper logic levels. When inputs E1 and E2 are low, all

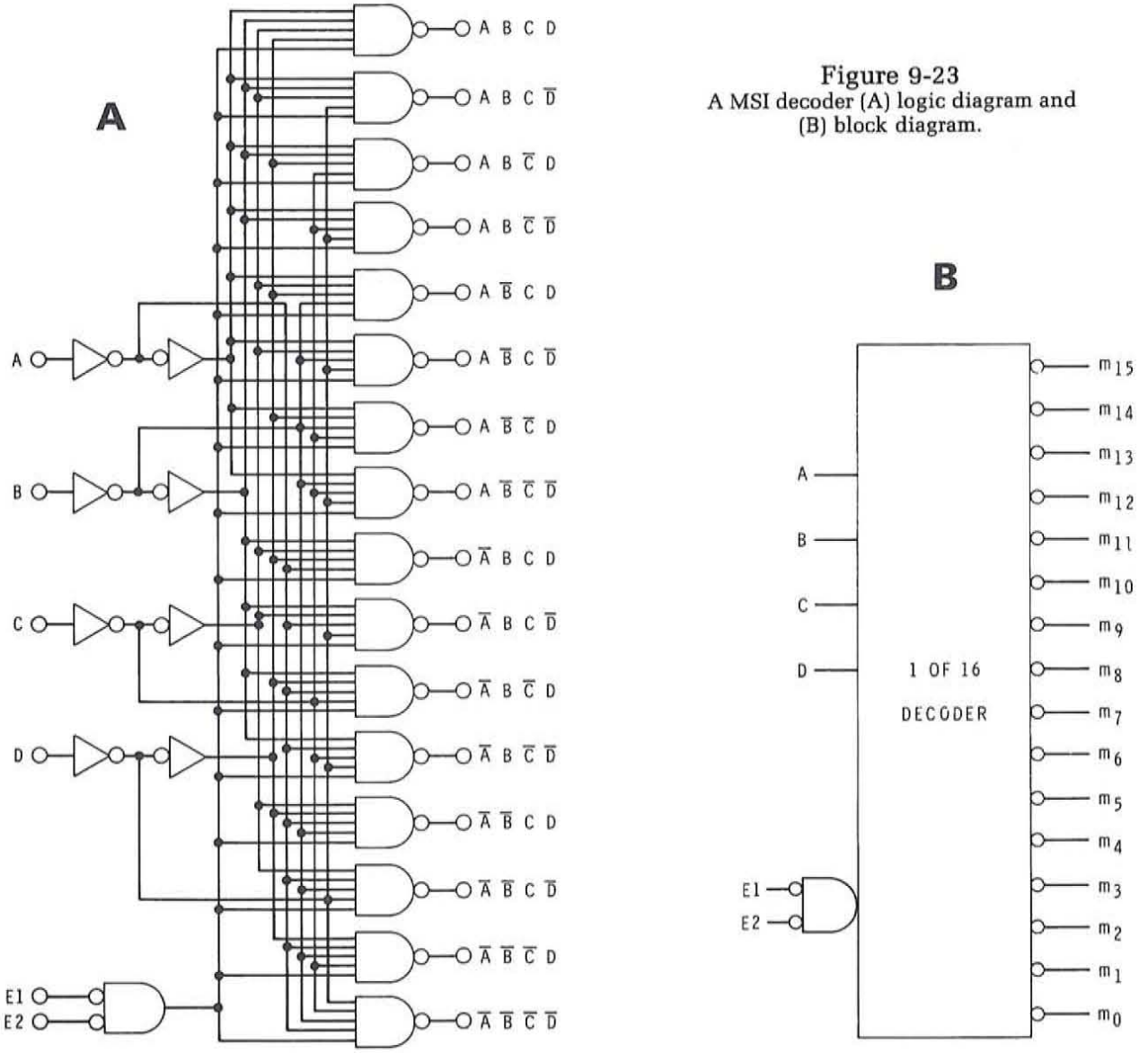
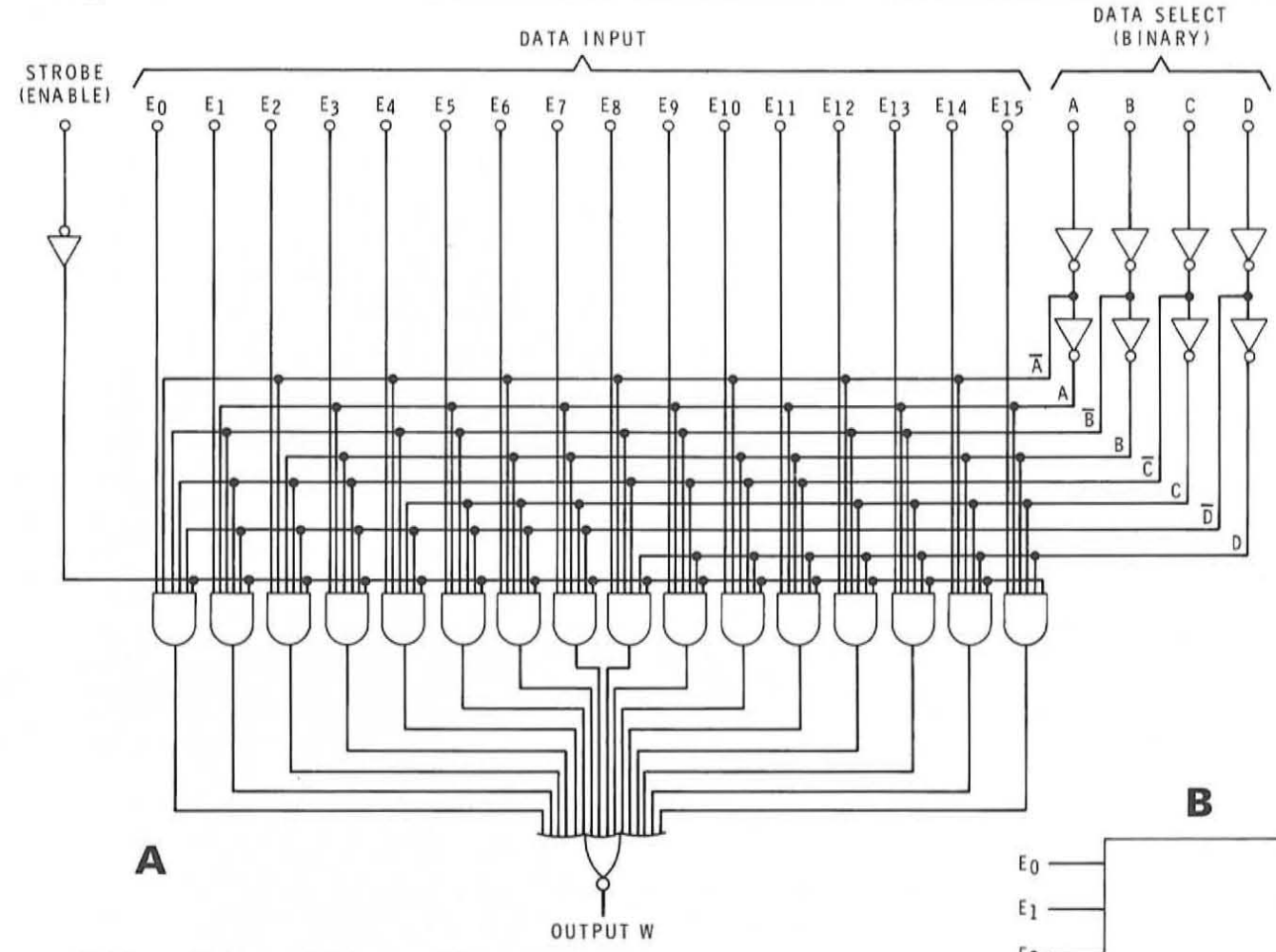


Figure 9-23
A MSI decoder (A) logic diagram and (B) block diagram.



$$W = S(\overline{A}\overline{B}\overline{C}\overline{D}E_0 + A\overline{B}\overline{C}\overline{D}E_1 + \overline{A}B\overline{C}\overline{D}E_2 + AB\overline{C}\overline{D}E_3 + \overline{A}\overline{B}C\overline{D}E_4 + A\overline{B}C\overline{D}E_5 + \overline{A}BC\overline{D}E_6 + ABC\overline{D}E_7 + \overline{A}\overline{B}\overline{C}DE_8 + A\overline{B}\overline{C}DE_9 + \overline{A}B\overline{C}DE_{10} + AB\overline{C}DE_{11} + \overline{A}\overline{B}CDE_{12} + A\overline{B}CDE_{13} + \overline{A}BCDE_{14} + ABCDE_{15})$$

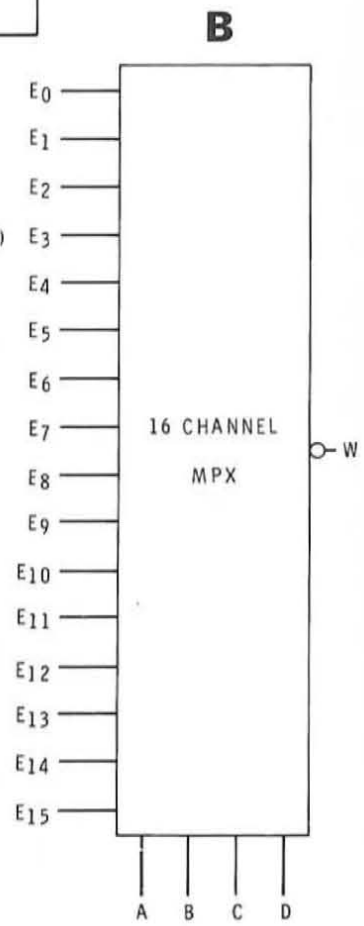


Figure 9-24
7415 TTL MSI multiplexer or data selector used to implement combinational logic circuits. (A) logic diagram, (B) block diagram.

sixteen gates are enabled. Figure 9-23B shows the block diagram used to represent this MSI decoder. Only the inputs and outputs are shown to simplify the drawing of the circuit. The outputs here are designated by the minterm numbers.

Another MSI circuit widely used for implementing combinational logic functions is the multiplexer or data selector. Figure 9-24 shows a logic diagram of a typical sixteen channel multiplexer circuit. Any one of sixteen input signals may be routed to the single output. Each input is applied to an AND gate that is enabled by the four-bit input code. The input code selects the input that is routed to the output.

Investigation of the data selector circuit in Figure 9-24 shows that all four-variable minterms are generated by the AND gates in the circuit. These are ORed together to produce a single output. The data selector circuit itself then implements a logic equation that is a sum of all possible minterms. An additional fifth input variable E can be accomodated by connecting it to one or more of the sixteen input lines. For example, any one of the sixteen four-bit minterms can be added to the output by applying a binary 1 to the appropriate data input. If a minterm is not needed in the output, the associated input line can be connected to binary 0. A fifth input code bit can be implemented by connecting it or its complement to the appropriate data inputs. A single strobe or enable line(s) is also used to enable or inhibit the entire circuit. Note that the output is active low. An inverter or other logic gate can be used to provide the complement if needed.

We can readily illustrate the use of MSI decoders and data selectors by showing how our BCD invalid code detector circuit can be implemented with them.

Figure 9-25 shows the 74154 one-of-sixteen MSI decoder used to generate the BCD invalid code detection function. Instead of working with the simplified Boolean equation for this function we work with the complete equation derived from the truth table. Each of the six four-variable minterms are defined. These minterms are generated by the decoder. The proper decoder outputs are then fed to a TTL NOR gate to produce the desired output function. Note that with this method of implementation two integrated circuits are required. Although we are able to implement this function more economically than by the previously described brute force SSI method, the result does not produce a minimum package count or the lowest cost design. MSI circuits are considerably more expensive than SSI circuits. In addition, this implementation requires two integrated circuits. The MSI implementation shown in Figure 9-25 is not the most desirable approach. In most cases it will not produce the most efficient design.

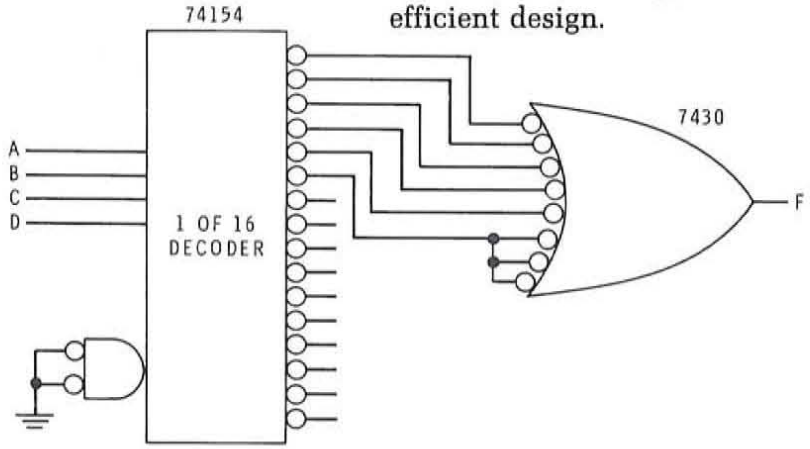


Figure 9-25
BCD invalid code detector function implemented with a 1-of-16 MSI decoder.

Figure 9-26 shows how the BCD invalid code detection function can be implemented with a sixteen channel data selector. A 74150 multiplexer circuit is used. Note that binary 1 (+5 volts) is applied to the six higher order inputs thereby enabling the gates which generate the proper minterms. All other inputs are connected to binary 0 to inhibit the remaining minterms from being applied to the output. Again we work from the expanded version of the logic equation rather than the simplified version. The implementation does result in the use of the single integrated circuit. But again this is an MSI device that is larger and more expensive than the simple SSI circuit we developed earlier. Note also that the circuit in Figure 9-26 has an active low output. Whenever we detect one of the six invalid BCD codes, the output of this circuit ($\overline{F}$) will go low instead of high as we indicated earlier. For many applications this is no disadvantage since a low output is just as valid an indication of the incorrect code as a high output. Nevertheless, our initial requirements stated that the output must be high. This may require the use of an external inverter thereby adding an additional IC package and further increasing cost, power consumption, and waste of space.

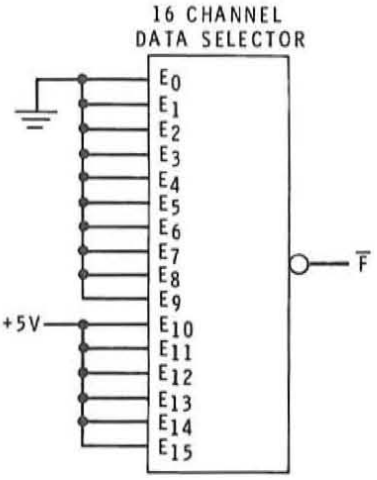


Figure 9-26
BCD invalid code detector circuit implemented with a MSI data selector.

While the use of MSI decoders and data selectors did not lead to a minimum implementation of our example problem, there are many situations where these devices will result in the minimum, lowest cost design. Every design will be different and you must evaluate each of the four basic alternatives before you find the one that meets your design criteria.

ROM Implementation. One of the easiest ways to design digital circuits is to use a read only memory. Virtually no design time or effort is required to use such a device. However, there are restrictions or limitations on its use. ROMs are generally large scale integrated circuits and must be custom manufacturered to your specifications. Therefore, they are expensive. In order to justify their use, the logical functions being implemented must require that degree of sophistication.

Here are some guidelines for determining whether a ROM should be used to implement a given logic function:

1. ROMs are used primarily for multiple input and multiple output logic circuits. The use of a ROM becomes practical and economically feasible only when the number of inputs and number of outputs are equal to or exceed four. Logical circuits having fewer inputs and outputs are generally more economically implemented with SSI or MSI logic circuits. Because of this restriction, a ROM is not applicable to our BCD invalid code detector. While the circuit requires four inputs, it has only a single output. Naturally, a ROM could be employed but more of its capabilities would be wasted than used.

2. ROMs are best employed where all possible input combinations are specified by the logic design. For example, in a four input variable circuit a ROM is economical only if all or most of the input states are used.

If the circuit you are designing has four or more inputs and outputs you should consider the use of a ROM. Evaluate your design by studying the truth table to see if it meets the criterion indicated in the two steps above. If it does, the implementation of the ROM can be taken directly from the truth table itself as the input signals specify the ROM address states while the output states specify the memory contents at each of the address locations. No further design procedure is required.

PLA Implementation. The fourth and final alternative available to the digital designer in implementing combinational logic circuits is the programmable logic array. This LSI device is used primarily for implementing large complex logic functions. Such devices do not become practical or economically feasible until the complexity of the design reaches a very high level. PLAs are used primarily for multiple input multiple output circuits. If your design calls for five or fewer inputs and outputs, a PLA will not result in a minimum design. The cost involved in programming the circuit to produce the desired function during manufacturing will make the cost unusually high. In such cases MSI circuit implementation should be employed. If the number of inputs and outputs exceeds five or six, then PLAs should be considered. Like any of the other alternatives, the PLA implementation should be evaluated carefully from a cost standpoint. Size, power consumption, and reliability should also be considered as usual. For both PLAs and ROMs, high volume of usage will greatly reduce the cost and make these alternatives more practical. For our BCD invalid code detector, the PLA would certainly not be applicable.

To design a logic circuit using a PLA, you use the procedure outlined previously. The design is first tabulated in a truth table. From the truth table the logic equations are written. Then by using Boolean algebra or Karnaugh maps, the equations are then minimized. The minimized equations are given to the manufacturer who will in turn design the mask that will properly interconnect the gates within the PLA device. The result will be a custom integrated circuit for your design.

All four of these design alternatives for implementing digital circuits depend directly upon the available commercial integrated circuits. Your ability to meet your design objectives is a function of the type, cost, performance, flexibility, and quality of the integrated circuits available to you. For that reason it is imperative that you become familiar with the literature of all of the manufacturers of digital integrated circuits. Order

their catalogs and ask for individual device data sheets. Study these to determine what circuits are available, what their specifications are and how they can be used. In addition, most integrated circuit manufacturers supply application notes which describe the ways in which their component can be used. Most manufacturers also offer engineering and design assistance to help their customers in selecting the correct device. There are also many manufacturers that do custom design work. If the commercially available devices do not meet your applications then it is possible that special custom devices can be designed and implemented for you. We simply cannot overemphasize the importance of being familiar with and working with the manufacturers of the digital integrated circuits that you will use in your designs.

Multiple Output Combinational Circuits

The applications that we have considered involve circuits with a single output. Multiple input states are monitored and a single logic signal is developed to indicate when specific states occur. There are many applications, however, that require multiple outputs as well as multiple inputs. All of the design procedures that we have described so far are fully applicable to combinational circuits with multiple outputs. Only minor variations are necessary to achieve a correct design.

The methods of defining the problem and stating the design objectives are similar. You will completely specify the type and number of inputs and the type and number of outputs.

Your problem statement is then converted into a truth table that will completely define the operation of the circuit. The number of inputs will determine the total number of states that can exist. Then, instead of defining a single output based on these inputs, you will define all of the outputs required by the application. Simply, this means creating a separate column in your truth table for each circuit output. In each column a binary 1 is recorded adjacent to the set of input conditions necessary to produce that output. Don't forget to note the states that won't occur or states that have no meaning for this application. These "don't care" states will greatly aid in reducing the amount of circuitry required. Once the truth table is complete you will have thoroughly defined the circuit to be designed.

Next, you will observe the output columns in the truth table and write a separate Boolean equation for each. Use a Karnaugh map to minimize these output equations. This will result in a minimized or reduced output equation for each of the outputs required by the circuit. It is these minimized equations that you will implement in your final design.

When choosing the integrated circuits to implement your design there are several important points to consider. First, depending upon the complexity of the circuit, ROMs and PLAs should be given first consideration. These will generally result in the simplest and smallest circuits. MSI logic circuits should then be considered if ROMs and PLAs are too complex and sophisticated for the application. For many common functions, a standard MSI circuit may already exist making the design unnecessary. Finally, SSI circuits should be considered for multiple output circuits of minimum complexity.

When implementing the multiple output function with SSI circuits, it is a good idea to study the minimized output equations derived from the Karnaugh maps to determine if common product terms exist. If the same product happens to occur in two or more of the output expressions then it is only necessary to generate this product once. This will further reduce the amount of circuitry required.

Design Examples

We have now described the procedure for designing combinational logic circuits. Virtually any logic design problem can be handled with these procedures. However, because of the wide range of applications there will be many variations. The only way to illustrate the use of this procedure is to provide information on several different types of applications. Your own ability to design digital circuits will come from practice. The design examples in this section will help to give you the experience necessary to achieve competence. The primary purpose of the examples in this section is to illustrate the many ways in which the procedures described can be used. Additional practice problems are given in the Self Test Review following this section.

INPUTS	OUTPUT
A B C D	F
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	0
0 1 0 1	1
0 1 1 0	1
0 1 1 1	0
1 0 0 0	0
1 0 0 1	1
1 0 1 0	1
1 0 1 1	0
1 1 0 0	1
1 1 0 1	0
1 1 1 0	0
1 1 1 1	0

Figure 9-27
Truth table for Design Example #1: Two-of-four detector

Design Example #1. Design a two-of-four input detector circuit. The circuit has four inputs A, B, C, and D, and we want a binary 1 output condition F to occur whenever only two of these inputs are binary 1. Develop the truth table for this circuit, write the output equation, minimize it, and select a method for implementing it.

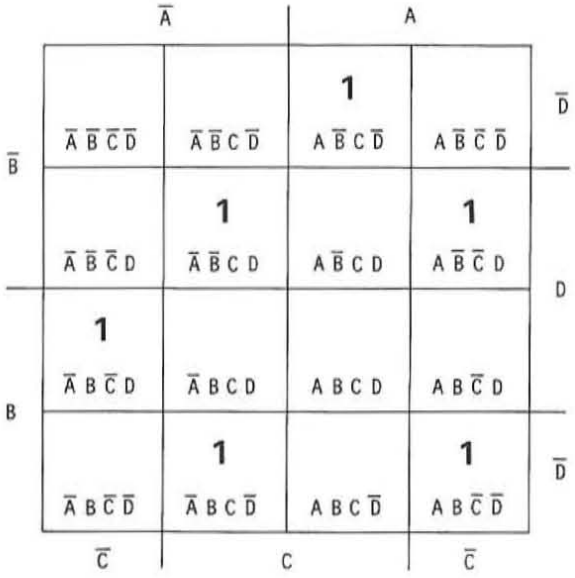
Solution to Design Example #1. The truth table for this circuit is shown in Figure 9-27. With four inputs there are sixteen possible combinations that can occur. Our design requirements stated that we wanted the output F to be binary 1 when only two of the inputs were binary 1. By observing the binary states of each of the sixteen possible input conditions, you can quickly identify those where only two of the inputs are binary 1. These states are indicated by a binary 1 in the F output column.

You can go directly from the truth table to a Karnaugh map to attempt simplification of this logic function. However, it is usually a good idea to write the logic equation from the truth table first. This step only takes a short time and helps you to visualize the function better. Writing the equation from the truth table gives us:

$$F = \overline{A}\,\overline{B}\,C\,D + \overline{A}\,B\,\overline{C}\,D + \overline{A}\,B\,C\,\overline{D} + A\,\overline{B}\,\overline{C}\,D + A\,\overline{B}\,C\,\overline{D} + A\,B\,\overline{C}\,\overline{D}$$

$$F = m_3 + m_5 + m_6 + m_9 + m_{10} + m_{12}$$

Next, using the logic equation or the truth table, plot the function on a Karnaugh map. This is done in Figure 9-28. A binary 1 is marked in those cells identified by the minterms specified by the truth table and the equation.



$$F = \overline{A}\,\overline{B}\,C\,D + \overline{A}\,B\,\overline{C}\,D + \overline{A}\,B\,C\,\overline{D} + A\,\overline{B}\,\overline{C}\,D + A\,\overline{B}\,C\,\overline{D} + A\,B\,\overline{C}\,\overline{D}$$

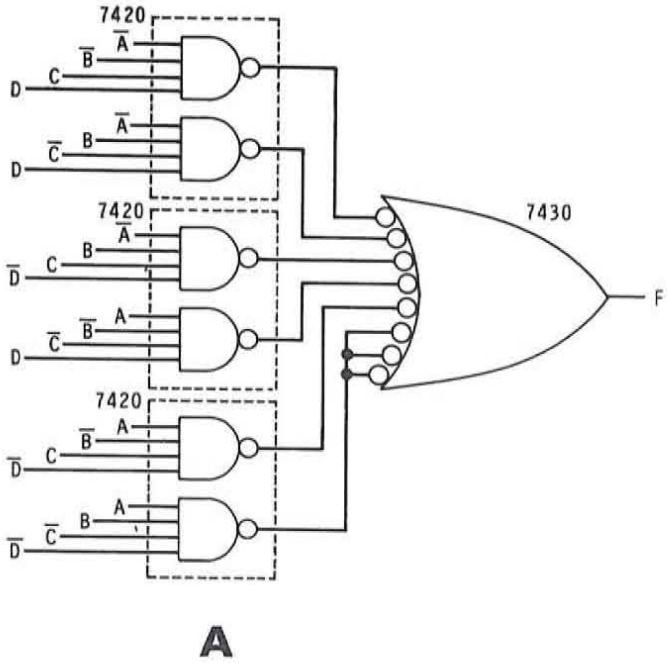
$$F = m_3 + m_5 + m_6 + m_9 + m_{10} + m_{12}$$

Figure 9-28
Karnaugh map for Design Example #1.

Observing the Karnaugh map should immediately tell you that absolutely no simplification of this logic function is possible. As you can see, the variables are widely spaced and separated. There are no two minterms that can be grouped together. Since no simplification is possible, the logic equation must be implemented directly.

An initial consideration of the four methods of implementing the logic function will quickly rule out the use of the ROM and the PLA. Since only a single output is required, the circuit implementation will be either by SSI logic elements or MSI functional devices. Your job is to evaluate these alternatives and select the best form of implementation for your design.

There are several ways that we can implement our two-of-four detector circuit. We can use SSI logic gates and implement the equation directly as shown in Figure 9-29. Here TTL SSI gates are used. Type 7420 dual four-input gates are used to form products of the inputs. A 7430 eight-input gate is used to produce the output sum. Depending upon the source of the inputs, the 7404 hex inverter IC may be needed to generate the complements of the input signals. With this circuit a total of five integrated circuits are required. While the cost of such circuits is extremely low (approximately 15 cents each in large quantities) they do take up a lot of space. A significant amount of time is required to lay out a printed circuit board to interconnect these devices. Therefore, it is desirable to investigate the methods of implementing this circuit with MSI functional devices.



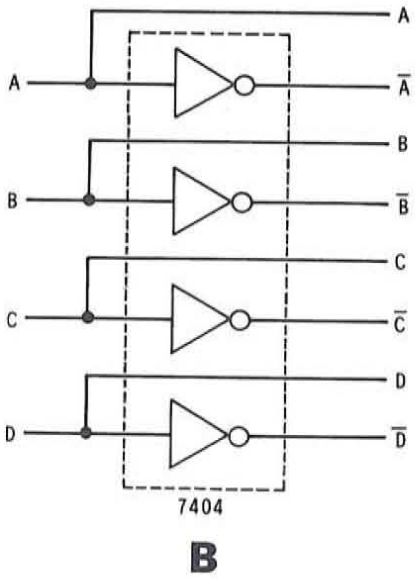


Figure 9-29
Two-of-four detector circuit.

Figure 9-30 shows how the two-of-four detector circuit can be implemented using a 74154 one-of-sixteen decoder and a 7430 eight-input gate. The one-of-sixteen decoder is used as a minterm generator and the appropriate outputs are ORed together in the 7430 gate. The size of the circuit is somewhat less than the SSI implementation mentioned earlier. The layout is simpler and the two circuits occupy much less space.

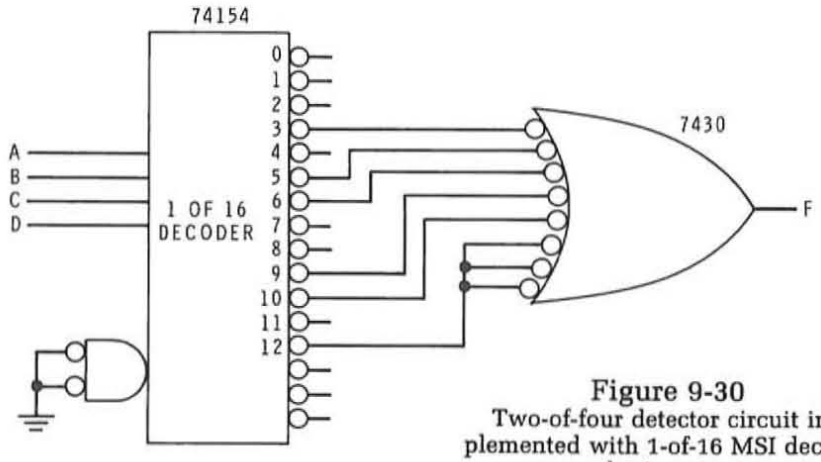


Figure 9-30
Two-of-four detector circuit implemented with 1-of-16 MSI decoder and 8-input gate.

A third alternative is to use an MSI data selector. The two-of-four detector circuit can be implemented with a 74151 multiplexer as shown in Figure 9-31. This single 16-pin dual-in-line IC seems to offer the most promising method of implementing the circuit.

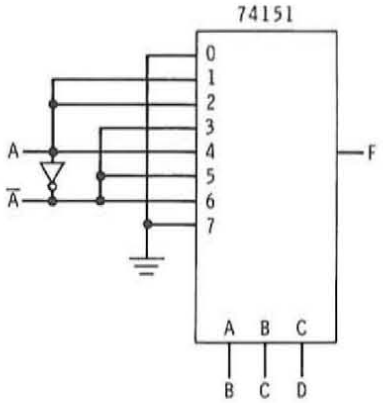


Figure 9-31
Two-of-four detector circuit implemented with MSI data selector.

The 74151 data selector or multiplexer circuit was described in detail in an earlier unit. The logic diagram of this circuit is repeated in Figure 9-32. Each of the gates in the multiplexer are enabled by the A, B, and C inputs. The B, C, and D inputs of our logic circuit will be applied to these lines. These are the least significant bits of the four-bit words. To see how the circuit works, consider the decimal value of these three least significant bits alone and analyze the truth table to determine which inputs on the multiplexer will be used. You should find that six of the sixteen possible states of these three inputs are used. The decimal values of the BCD input for each output F are 3, 5, 6, 1, 2 and 4. The unused input states are $\overline{B}$, $\overline{C}$, $\overline{D}$ and B, C, D, or 0 and 7. These correspond to the 0 and 7 inputs on the multiplexer. Since they will not be used, these two inputs are connected

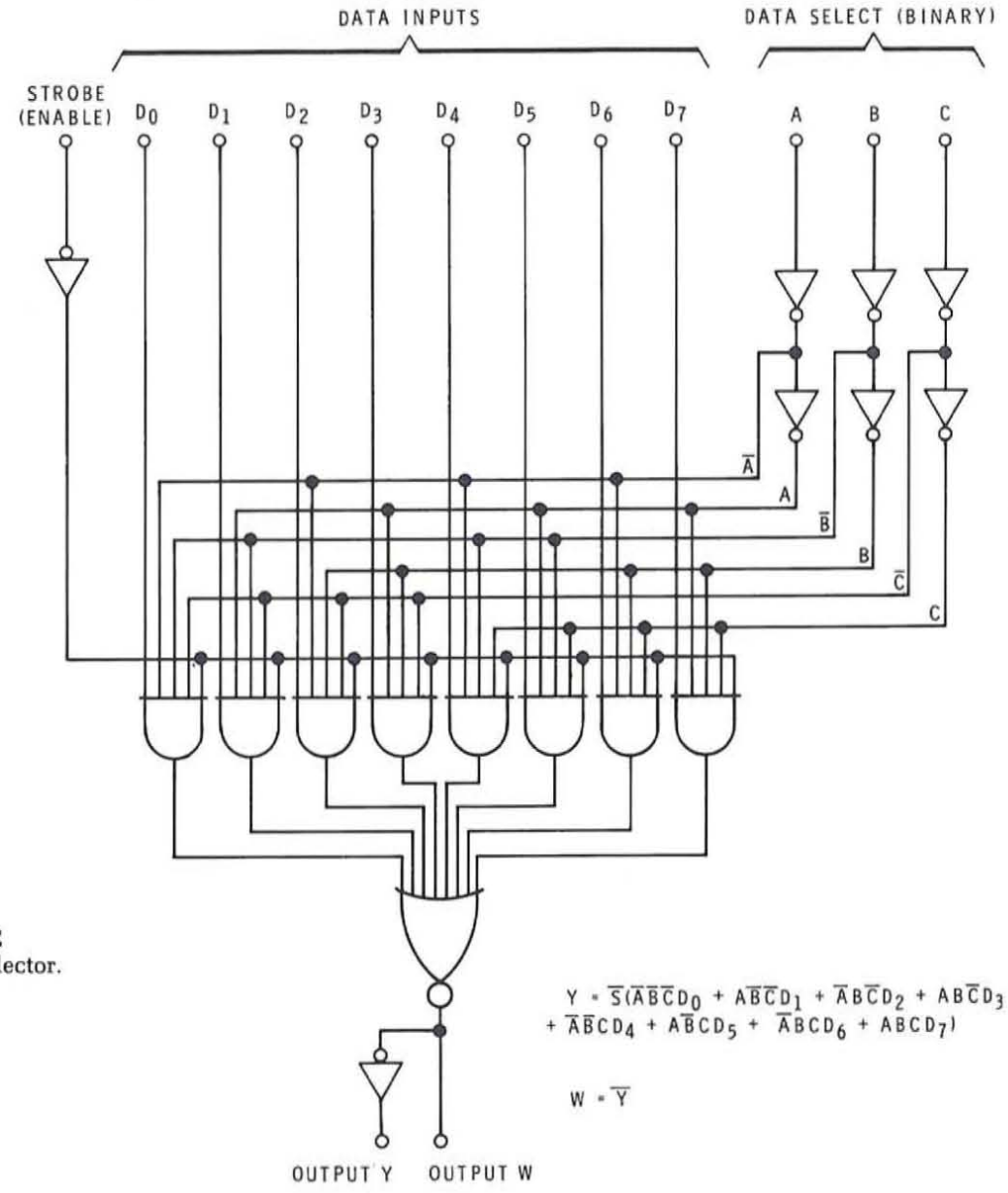


Figure 9-32
74151 TTL data selector.

to ground or binary 0 thereby disabling them. To the other six inputs we connect either the A or $\overline{A}$ signal in order to form the appropriate four-bit product terms. Depending upon the source of the A input, an external inverter may or may not be required as shown in Figure 9-31.

In evaluating our alternatives now we can see that the simplest and easiest to use is the 74151 multiplexer. It results in a single integrated circuit and design time and PC board layout time are at a minimum. However, this MSI device is more expensive compared to the SSI devices used and shown in Figure 9-29. The MSI multiplexer costs approximately twice as much as all of the integrated circuits in the SSI version. For that reason from a cost standpoint we may be inclined to select the SSI implementation method. However, keep in mind that the PC board layout time for the SSI circuit will be significant. In most cases it will be great enough to offset the extra cost of the 74151. For that reason the multiplexer implementation of the circuit probably offers the best solution to this particular design problem.

Design Example #2. This next example of combinational logic circuit design is more complex but is also more representative of the types of circuits that you will be designing. The techniques for designing multi-input multi-output logic circuits are demonstrated here.

Design a code converter circuit that will change the 8421 BCD code into the four-bit excess 3 code. Parallel inputs and outputs are required. (NOTE: Since the 8421 BCD input code is used, the six invalid states are considered as "don't care" states.)

Solution to Design Example #2. The first step in the design procedure is to develop a truth table. Since the inputs are the 8421 BCD code, four input lines are required. These are labeled A, B, C, and D. The excess 3 code also has four bits. These will be labeled W, X, Y, and Z. The truth table for this circuit is shown in Figure 9-33.

INPUTS 8421 BCD A B C D	OUTPUTS X S 3 W X Y Z
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 1 0 0
1 0 1 0	DON'T CARE
1 0 1 1	
1 1 0 0	
1 1 0 1	
1 1 1 0	
1 1 1 1	

Figure 9-33
Truth table of 8421 BCD to XS3 code converter circuit.

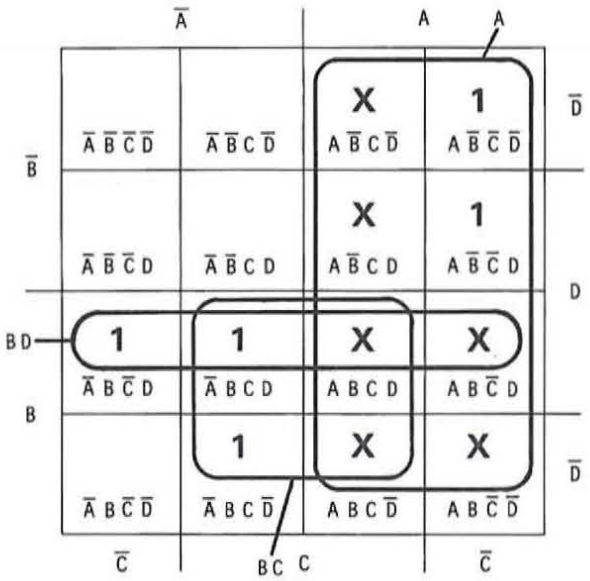
The next step is to write the Boolean equations from the truth table. Since there are four outputs from the circuit, you will develop an output equation for each. This is done by observing the positions of binary 1's in each output column. Then you write a sum-of-products expression involving the related minterms. The output equations for this circuit are:

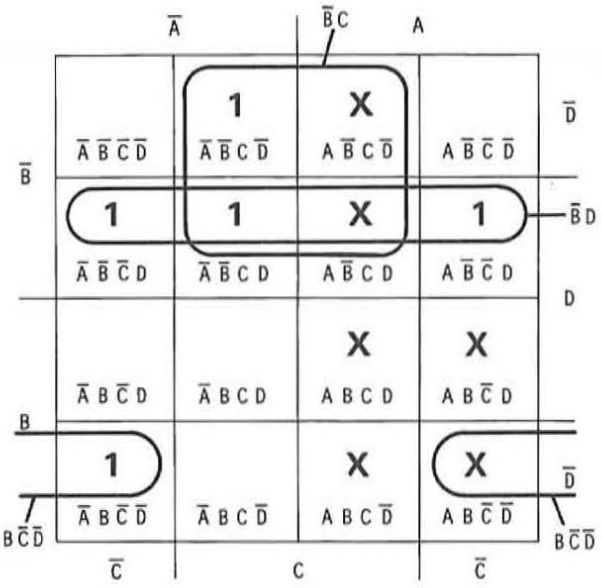
$$W = \overline{A}\,B\,\overline{C}\,D + \overline{A}\,B\,C\,\overline{D} + \overline{A}\,B\,C\,D + A\,\overline{B}\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{C}\,D$$
$$X = \overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,\overline{B}\,C\,\overline{D} + \overline{A}\,\overline{B}\,C\,D + \overline{A}\,B\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{C}\,D$$
$$Y = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,C\,D + \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,B\,C\,D + A\,\overline{B}\,\overline{C}\,\overline{D}$$
$$Z = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,C\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,B\,C\,\overline{D} + A\,\overline{B}\,\overline{C}\,\overline{D}$$

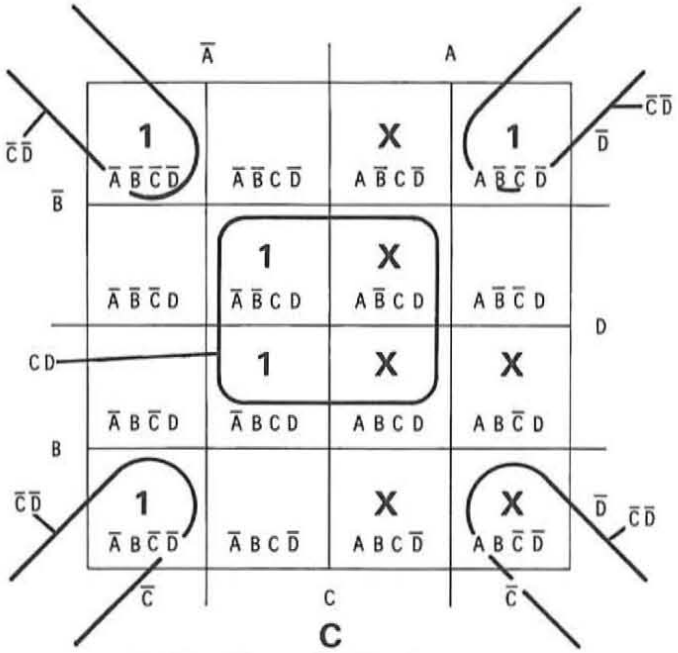
The next step is to map the output equations. A sixteen-cell Karnaugh map is used for each output. You can map each output function directly

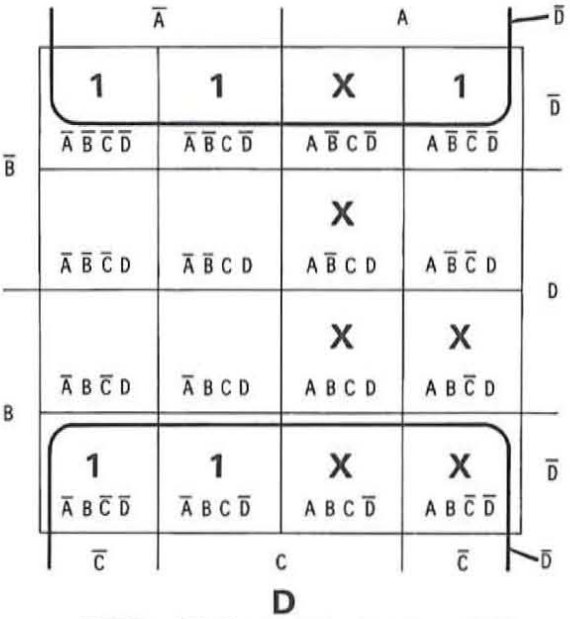
Figure 9-34
Karnaugh maps for BCD to XS3 code converter.

from the truth table or from the equations you derived from the table. Don't forget to mark the "don't care" states with X's. Combine these X's with the binary 1s on the map to help in reducing the equations. Finally, minimize the equations by grouping the variables on the map and from those groupings write the reduced logic equations. Figure 9-34 shows the four output maps and the reduced equations.









As with any combinational logic circuit, there are several ways in which it can be implemented with hardware. Consider the various techniques described earlier and apply them to this problem to determine the optimum method of implementation.

Figure 9-35 shows the circuit implemented with SSI logic circuits. This circuit implements the minimized equations from the Karnaugh maps. Only four SSI packages are required. Assuming the use of 7400 TTL circuits, the following circuits are required:

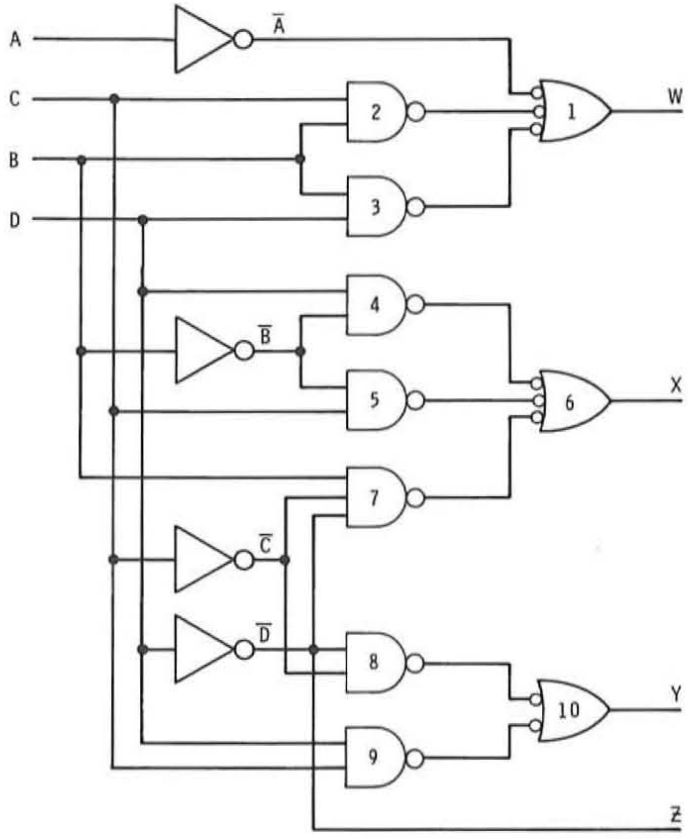


Figure 9-35
BCD to XS3 code converter implemented with SSI circuits.

1—7410 triple 3-input gate (gates 1, 6, and 7)
2—7400 quad 2-input gates (gates 2, 3, 4, 5, and gates 8, 9, and 10)
1—7404 hex inverter

The circuit is simple and straight forward. Trace the input lines and compare each of the circuits with the logic equations to be sure you see how the circuit is implemented.

Figure 9-36 shows how the BCD to XS3 code converter can be implemented with MSI data selectors. Type 74150 TTL MSI data selectors are used to implement the output equations for W, X, and Y. These sixteen-input multiplexers are driven by the four-line 8421 BCD input. The input lines of the multiplexers corresponding to the minterms appearing in the output equations are connected to +5 volts to enable them. The unused inputs are connected to ground to disable them. Output Z is implemented with an inverter connected to the D input. Note that this method of implementation requires four IC packages, three of which are 24 pin MSI devices. This method of implementation is larger and more expensive than the SSI implementation described earlier. It is not an efficient method of implementation.

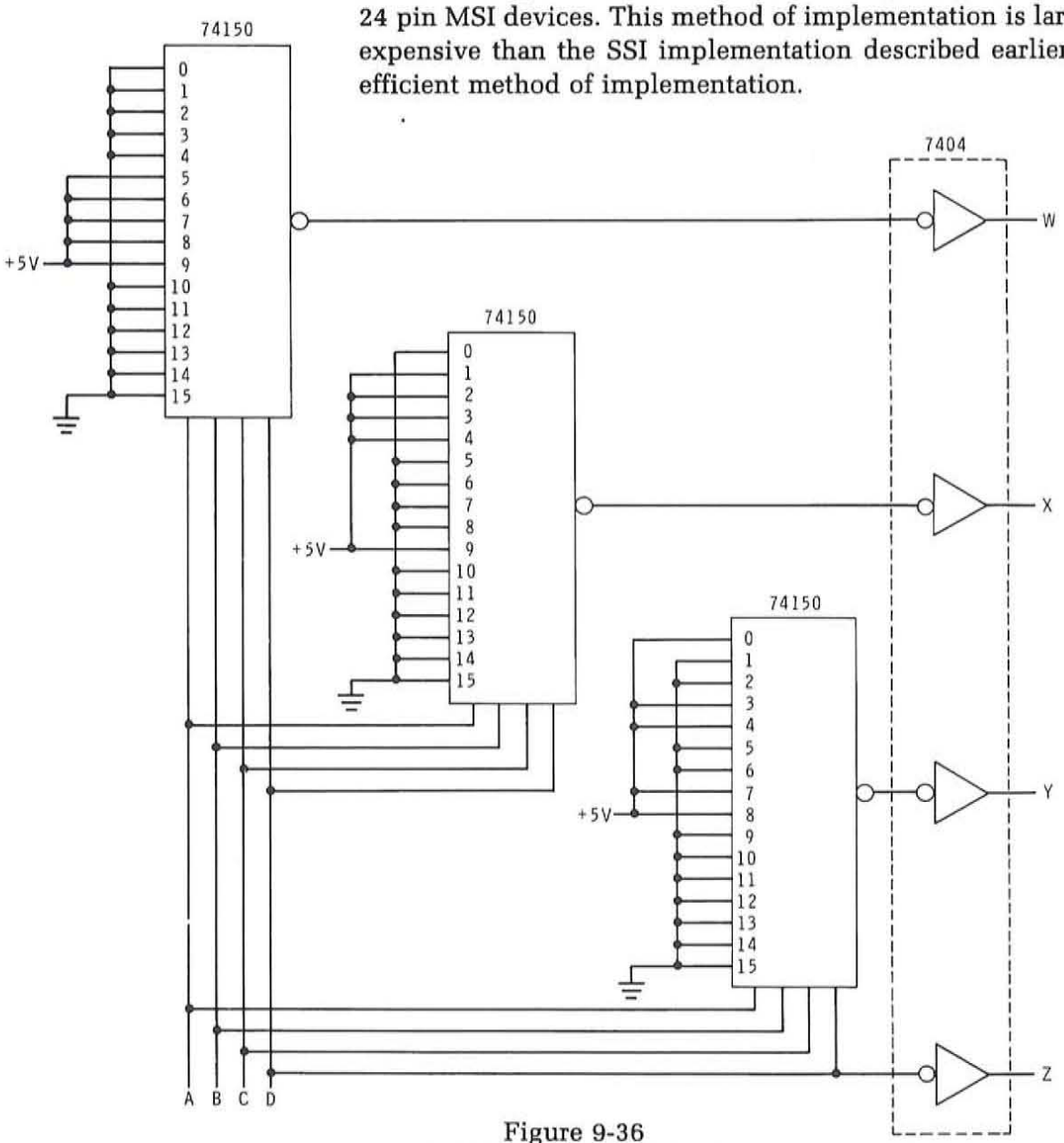


Figure 9-36
BCD to XS3 code converter implemented with MSI data selectors.

Perhaps the easiest way to implement this code converter circuit is to use a ROM. The 8421 BCD input code can be applied to the address lines of the ROM. The corresponding XS3 output code can be stored in the memory location specified by the input address. Since there are ten input states and ten output states, ten memory locations are required. The four-bit output code means that a total of forty bits are required in the ROM to implement this function.

Our guidelines for determing the applicability of a ROM to a combinational logic design is that the circuit have four or more inputs and outputs. Such a criterion specifies a minimum 64-bit ROM. Four input lines can specify a total of sixteen memory locations. Four output lines specifies four bits per memory word or $4 \times 16 = 64$ bits. Commercial ROMs this small are not available. The smallest available commercial ROM is a 256 bit unit organized as 32 eight-bit words. Such a ROM could be used for implementing the BCD to XS3 code converter.

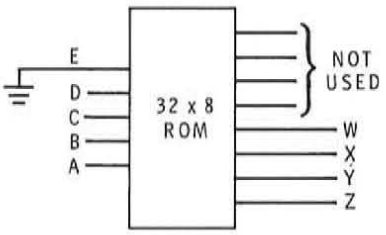


Figure 9-37
BCD to XS3 code converter implemented with a ROM.

Figure 9-37 shows the block diagram of a 32×8 ROM used to implement this function. The 32 word memory is addressed by five address input lines. The fifth or E input line is not required so it is simply connected to ground. The BCD input code is applied to the A, B, C, and D input lines. Each memory location can hold up to eight bits. Therefore, there are eight output lines. Only four of these are required for this application. These are labeled W, X, Y and Z to correspond to the desired output code signals. When the ROM is manufactured, the XS3 code will be stored in the memory location specified by the 8421 BCD input code.

With this arrangement only forty of the total possible 256 bits are used. This means that a significant amount of the memory is wasted. However, if this circuit is to be used in high volume, the cost of this device can be very low. Since it requires only a single 16-pin dual-in-line package, it may be the most desirable means of implementing this function.

The other method of implementing this logic equation is with a PLA. This is not a practical means in this application since the requirement is not large enough or complex enough to warrant the use of a PLA. Therefore it should not be considered.

In considering the various means of implementing the circuit we have outlined, the two most desirable means appear to be the SSI implementation of the minimized equations or a ROM. The SSI implementation is the lowest cost approach but does require four integrated circuit packages and the associated printed circuit board design. The ROM method is more expensive but occupies less space. Depending upon the quantities used and the size and space limitations of the project, the ROM method should be carefully considered.

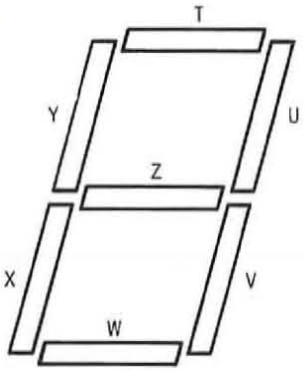


Figure 9-38
Seven-segment display.

Self Test Review

3. Design a three-input majority detector circuit that generates a binary 1 output if two or more of the inputs are binary 1 indicating a majority.
4. Design a BCD to 7-segment decoder circuit that converts the 8421 BCD code into the 7 logic signals to drive a 7-segment LED display with the corresponding decimal digits 0 through 9. Assume that the output of the circuit must be binary 1 in order to enable the segments in the display. The BCD input signals are designated A, B, C, and D while the 7-segment output signals are designated T through Z as indicated in Figure 9-38. Assume also that the 6 invalid BCD states can be used as "don't care" states.

NOTE: In both of the above problems, design the circuit using the procedures described earlier and select the smallest and most economical method of implementing the circuit. In problem 4, do not use "tails" on the 6 and 9 digits.

Answers

3. The design of the majority detector circuit calls for three inputs which we can label A, B, and C. The output, which we can call M, is to be a binary 1 whenever two or more of the three inputs are binary 1 at the same time. The signal therefore, indicates that a majority is present.

   The truth table for this circuit is shown in Figure 9-39. All eight possible combinations of the input signals are accounted for. Scanning down the input state we note those states where two or more of the inputs are binary 1. In the output column M where we record a binary 1 adjacent to those input states where the correct condition occurs, four of the eight input states represent a majority condition.

INPUTS	OUTPUTS
A B C	M
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

Figure 9-39
Truth table for the majority circuit.

Figure 9-40 shows the eight-cell Karnaugh map used to map and reduce this function. The original equation from the truth table is indicated. The function may be mapped from the equation or from the truth table itself. The loops of adjacent minterms are shown on the Karnaugh map. The resulting minimized equation M is indicated.

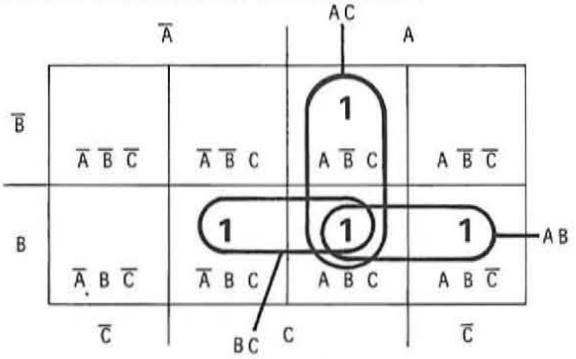


Figure 9-40
Karnaugh map for reducing the majority detector circuit.

$M = \overline{A}BC + A\overline{B}C + ABC + AB\overline{C}$

$M = BC + AC + AB$

On first glance this circuit appears to be best implemented with SSI logic circuits. Figure 9-41 shows one method of implementation. A type 7400 quad two-input IC is used to implement the input product terms. One of the gates in this IC is not used. A type 7410 triple three-input gate is used to implement the sum (OR) part of the result. The product terms developed by the input gates are ORed together in the three-input gate to produce the output M. Note that two of the three input gates in this IC are not used. Despite the fact we have used Karnaugh maps to minimize the number of inputs and number of output terms, there is still some waste of the circuitry because of the standard configuration in which ICs are available. This clearly indicates that standard minimization techniques do not always result in the lowest parts count or the minimum number of circuits when using ICs.

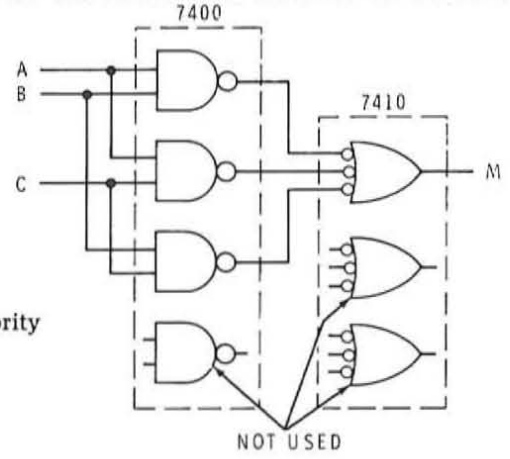


Figure 9-41
SSI implementation of the majority detector.

Figure 9-42 shows another way this function could be implemented. Shown here is a type 7454 TTL AND-OR-invert gate. This circuit implements the sum-of-products for four groups of two inputs. By using three of the input gates and properly connecting the inputs to the variables, the majority detector function can be implemented. The unused inputs are simply disabled by connecting them to ground. In this IC the output is active low meaning that the output equation developed by this circuit would be the complement of the desired equation. Many times the active low or complement output can be used as well as the normal version of the signal. The exact voltage level of the output signal depends on the application. By using this IC only a single package is required to implement the function. The purpose of this example is to indicate the importance of knowing the types of integrated circuits available. To design optimum circuits you must know which types of ICs are available. Be sure that you have access to all of the manufacturer's literature, data sheets, and application notes. When you are designing a circuit study the IC types available and mentally make note of those that could be of value to you.

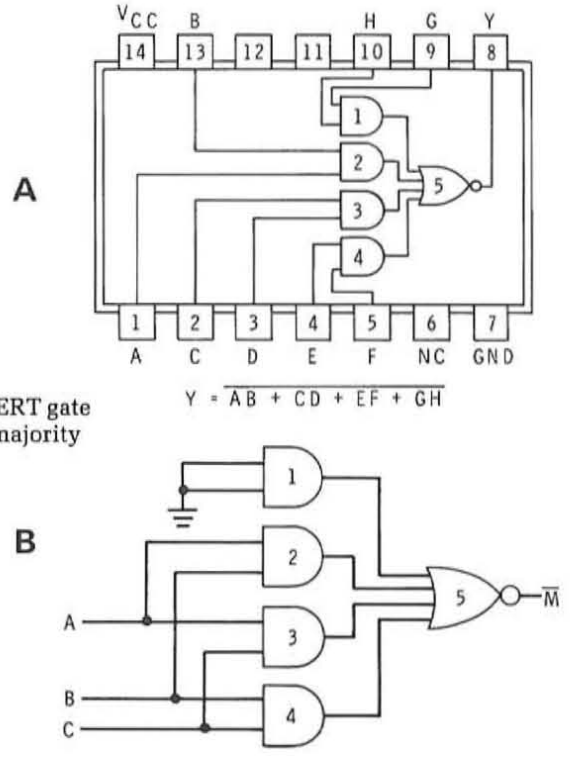
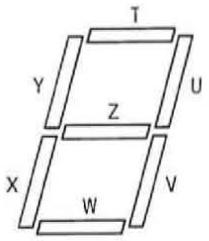


Figure 9-42
Type 7454 TTL AND-OR-INVERT gate (A) used to implement the majority detector (B).



BCD INPUT A B C D	DISPLAY	OUTPUT-SEGMENTS T U V W X Y Z
0 0 0 0	0	1 1 1 1 1 1 0
0 0 0 1	1	0 1 1 0 0 0 0
0 0 1 0	2	1 1 0 1 1 0 1
0 0 1 1	3	1 1 1 1 0 0 1
0 1 0 0	4	0 1 1 0 0 1 1
0 1 0 1	5	1 0 1 1 0 1 1
0 1 1 0	6	0 0 1 1 1 1 1
0 1 1 1	7	1 1 1 0 0 0 0
1 0 0 0	8	1 1 1 1 1 1 1
1 0 0 1	9	1 1 1 0 0 1 1
1 0 1 0	DON'T CARE	
1 0 1 1		
1 1 0 0		
1 1 0 1		
1 1 1 0		
1 1 1 1		

Figure 9-43
Truth table for BCD to 7 segment decoder.

4. Figure 9-43 shows the truth table for the BCD to 7-segment decoder. This circuit is a code converter for changing the 8421 BCD code into a special 7-bit output code to turn on the correct segments in a 7-segment LED display device to read out the decimal digits 0 through 9.

Figure 9-44 shows the output equation in minterm form for each output segment. These equations are then plotted on the 7 Karnaugh maps and minimized as indicated. The minimized output equations are also shown.

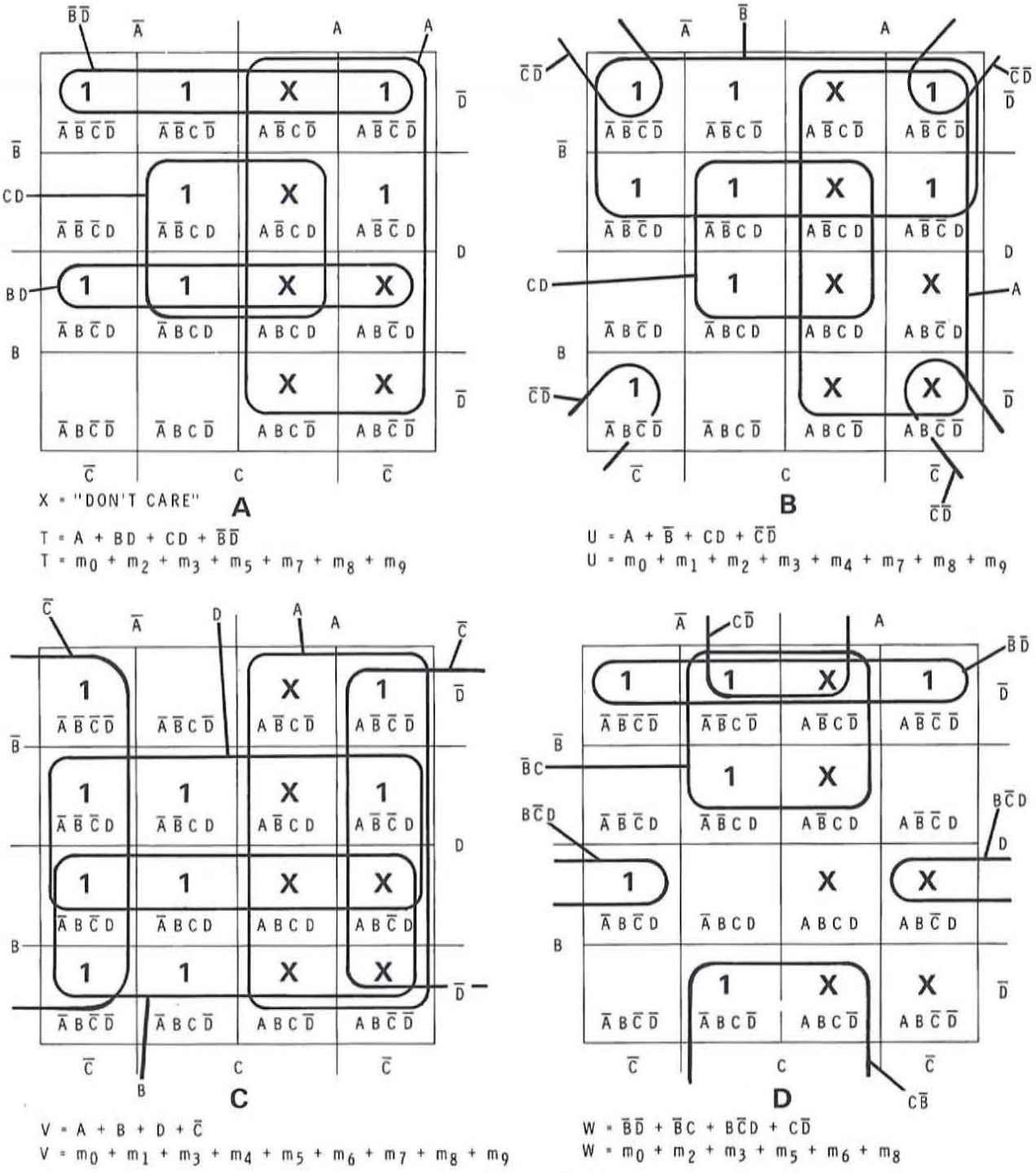
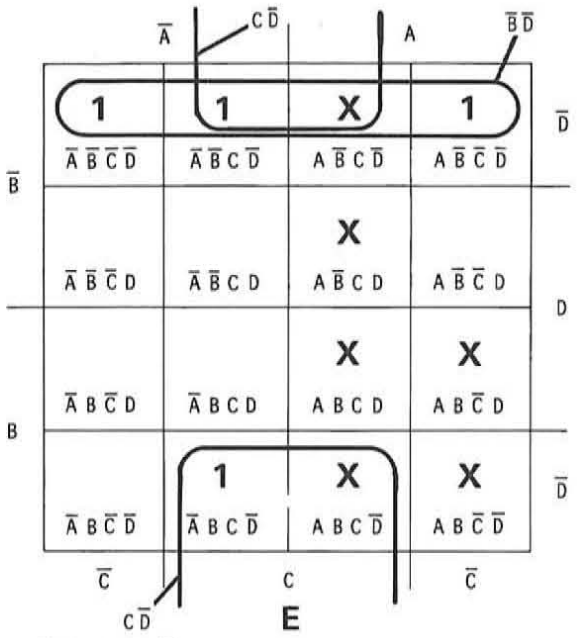
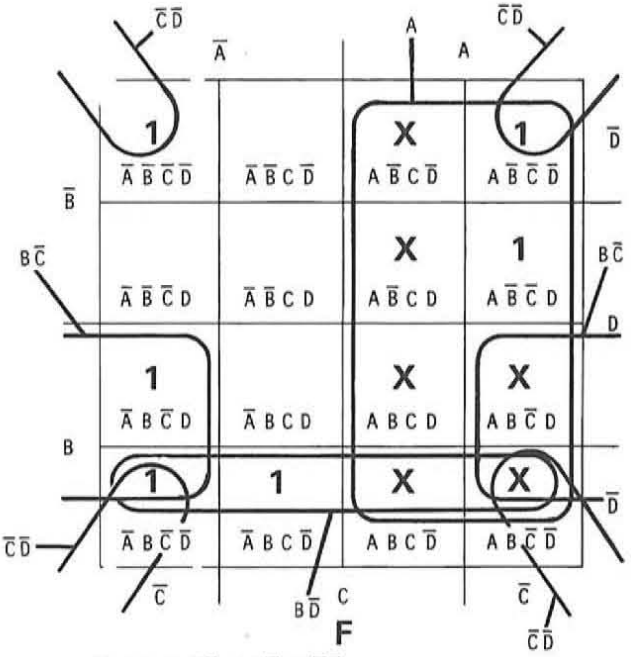


Figure 9-44
Karnaugh maps for minimizing the BCD to 7-segment code converter.



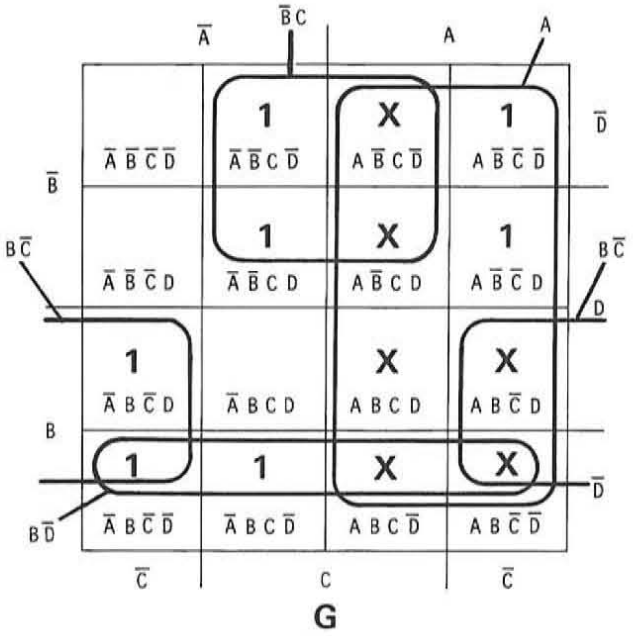
$X = B\bar{D} + C\bar{D}$

$X = m_0 + m_2 + m_6 + m_8$



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

$Y = m_0 + m_4 + m_5 + m_6 + m_8 + m_9$



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

$Z = m_2 + m_3 + m_4 + m_5 + m_6 + m_8 + m_9$

Figure 9-44
(continued)

Figure 9-45 shows how the circuit can be implemented with SSI logic devices. Compare this circuit to the minimized equations in Figure 9-44. It is important to note that the product terms in some output equations are common to several other equations (gates 2, 3, 5, 7 and 9). When a common term is found, that product can be generated only once and then used in several sum outputs. This eliminates the necessity to duplicate that product term with other logic circuitry. This further adds in the reduction of the amount of circuitry required to implement the desired function.

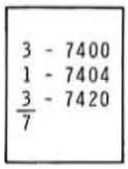


Figure 9-45
SSI implementation of a BCD to 7-segment decoder.

While MSI circuitry could be used to implement this equation it should generally be obvious to you that the cost and number of circuits required would be far greater than that shown for the SSI implementation in Figure 9-45. Since seven outputs are required, seven multiplexer circuits would be required. Since MSI devices are always more expensive than SSI devices, such a method of implementation would be wasteful. A BCD decoder such as the 7442 could be used to generate the minterms and OR gates used to produce the output sums. Again, this would take more circuitry than the SSI version described.

This application is a prime candidate for implementation with a ROM. It has four inputs and seven outputs. The four BCD inputs define ten memory locations each containing a seven-bit word. This means a seven-bit ROM is required. The smallest standard ROM available is 256 bits organized as 32 eight-bit words. This device would be perfect for implementing the BCD to seven-segment decoder. In fact, many commercial BCD to seven-segment decoders are implemented in this way. See Figure 9-46.

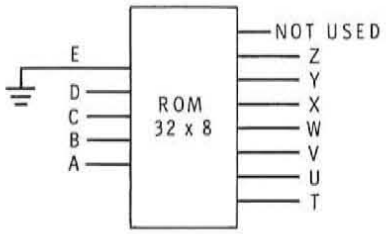


Figure 9-46
ROM used to implement the BCD to 7-segment decoder.

While the SSI logic circuit shown in Figure 9-45 or a ROM could be used to implement this function, you should recall that this device is already available as a single MSI circuit. Whenever a particular function has been defined, it is desirable to check the manufacturer's literature to be sure that the circuit isn't already available as an MSI package before beginning to design it. Today, it is simply not necessary to design a BCD to seven-segment decoder circuit since so many commercial versions are available. Such a device was discussed in detail in a previous unit and a typical device has been supplied with this program.

EXPERIMENT 23

DESIGNING COMBINATIONAL CIRCUITS

OBJECTIVES

To verify the design of the examples given in the text and to design typical circuits using the ICs supplied with this program.

Material Required

Heathkit Digital Design Experimenter ET-3200
All of the ICs supplied with this program.

Procedure

1. Construct the BCD invalid code detector circuit shown in Figure 9-22. Use a typical 7400 IC (443-1). Assign the pin numbers yourself, wire the circuit and verify its operation. Use data switches SW1 - SW4 as your input and one of the LED indicators to observe the output. Record your data in Table 1.
2. Show how you could use the 7442 BCD to decimal decoder IC to implement the BCD invalid code detector circuit. Draw the logic diagram and construct the circuit using any additional ICs that you need. Veryify the operation of the circuit. Review the operation and characteristics of the 7442 in Unit 8 and attempt to determine any special trait that will permit you to use this IC.
3. Show how the 74151 data selector IC can be used as a BCD invalid code detector. Draw the circuit, then construct it and verify its operation.

TABLE I				
INPUTS				OUTPUT
SW1 A	SW2 B	SW3 C	SW4 D	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Discussion

Figure 9-47 shows one way that the gates in a 7400 IC could be interconnected to perform the BCD invalid code detection function. This is the simplest method of implementing this function since it requires a single SSI package. When you applied the 16 four-bit words to the circuit you should have found that LED L4 indicated a binary 1 during the 6 invalid BCD states 1010 through 1111. For all other input codes which represent the ten states of the 8421 BCD code, the output of the circuit was binary 0 indicating that these are valid codes. Note that SW4, which represents the D input or LSB of the input word is not required by the circuit.

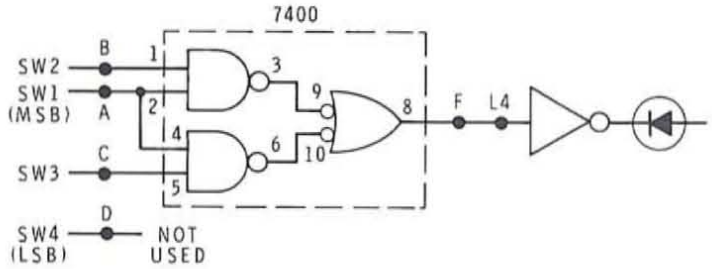


Figure 9-47
Minimum SSI circuit for BCD invalid code detector.

In Step 2 you were asked to determine a method for using the 7442 BCD to decimal decoder IC to implement the BCD invalid code detector. While the simplest and least expensive circuit for implementing this function has already been determined, the idea of this step is to give you practice in designing with MSI circuits. As you will see, many MSI circuits can be used for functions other than those for which they were originally designed. It is important to keep this in mind as many cost and time saving benefits can result.

If you studied the characteristics of the 7442 circuit as detailed in Unit 8, you should have found that the 7442 accepts the standard 8421 BCD input code and enables one of ten outputs. The enabled output goes low while all unselected outputs remain high. The truth table for this circuit indicates that if any of the six invalid BCD codes are applied to this device, all ten outputs will go high simultaneously. For any of the valid codes at least one of the ten will be low. This should give you a hint as to how you may use this device to detect one of the six invalid BCD codes.

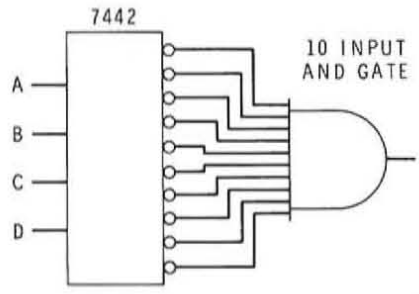


Figure 9-48
Using the 7442 to implement the BCD invalid code detector.

Figure 9-48 shows one method of doing this. All ten outputs of the 7442 are connected to a ten-input AND gate. As long as a valid BCD code is applied to the input, one of the ten outputs will be low and will disable the AND gate. However, if an invalid input code should be applied, all outputs will go high. The output of the AND gate will go high at this time and thereby indicate an invalid condition.

Figure 9-49 shows how you might have implemented this circuit with the ICs available in this program. Since a ten-input AND gate is not available, other IC gate packages can be properly interconnected to perform this function. In Figure 9-49, the 7420 IC contains two four-input NAND gates. These gates, labeled 1 and 2, when combined with inverters 3 and 4 form two four-input AND gates. The outputs of these two circuits are then ANDed in gate 5. Gate 5 and inverter 7 form another AND gate. The two remaining outputs of the 7442 are ANDed in gate 6. The outputs of inverters 7 and 8 are then again ANDed in gate 9. The output of inverter 10 is the output of our 10-input AND gate. Trace through this network of gates to be sure that you understand how they perform as a single 10-input AND gate.

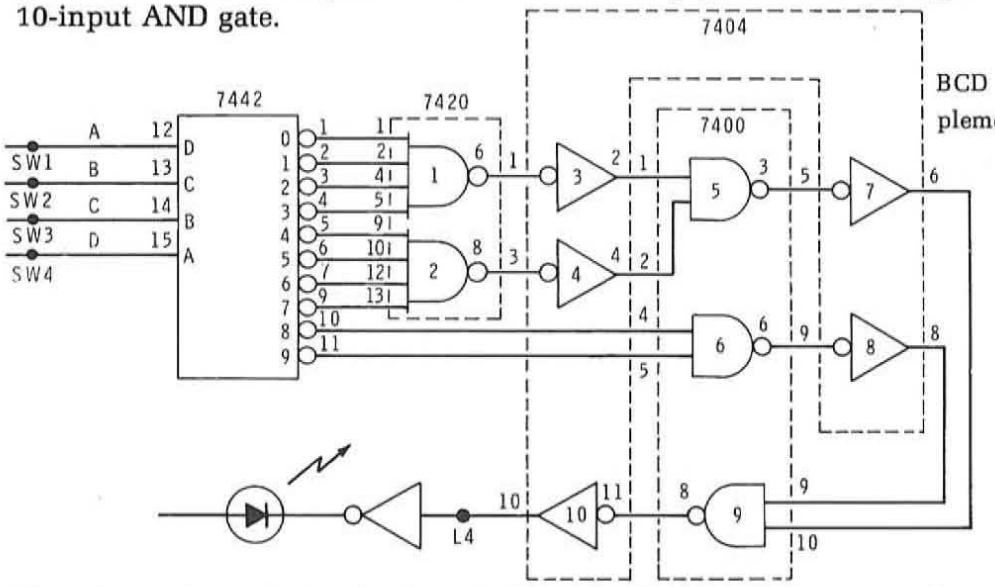


Figure 9-49
BCD invalid code detector implemented with 7442, 7420, 7404, and 7400.

There is one important point to make here. In all of our design examples in this unit we have used the letter designations A, B, C, and D to define our input states. We find A as the MSB and D as the LSB. The manufacturer's literature and our previous text coverage use the same letters A, B, C, and D to define the inputs, however, A was assumed to be the LSB and D the MSB. This is just the opposite from the assignments we made in this unit. The assignments are arbitrary and can be changed around to fit whatever designations you desire. However, the reversal in assignments may have led to some confusion for you. The purpose of doing this was to show you how the input assignments can be changed around in any way that fits your particular application. The most direct approach is to give the circuit whatever designations you desire, then match these to the input designations on the IC. Change the input designations on the IC to match your design. In Figure 9-49, our input definition assignments are shown adjacent to the input lines. The letter designations inside the outline for the integrated circuit are those assigned by the manufacturer. No difficulty will arise as long as you keep the definitions straight. However, if you should have accidentally reversed the connections so that you made the assigned input designations match those on the IC, the circuit would not have performed properly.

In the design example in the text we showed you how to implement the BCD invalid code detector circuit with a 74150 16-input data selector circuit. In this experiment you are asked to show how a 74151 8-input data selector could also be used to perform this function. The technique for doing this was described in the text. The 74151 has a three-bit address input code which selects one of the 8 inputs. Three of our four BCD input bits will be applied to these address lines. The fourth BCD input will be connected to the appropriate data selector inputs in order to produce the correct function. In order to determine the connections for the address lines and the input lines to the data selector, the truth tables can be used. Table II shows the six invalid BCD states with the letter designations as we assign them in the design example. Note that A is the MSB and D is the LSB. Table III shows the input states for the 74151 data selector. The three-bit address and the enabled input designations are given. Note that the input address here defines A as the LSB. To determine which inputs must be applied to the 74151, you simply match portions of the truth tables for the desired output states and for the device itself. Note how the least significant bits of the invalid states correspond to the six address input states designated in the table for the 74151. Since these two match, the B, C, and D inputs should be connected to the C, B, and A pins of the 74151 IC. By matching the truth tables we have also designated which inputs of the data selector are to be connected. Since the first two input states are not used, inputs D_0 and D_1 can simply be connected to ground. Inputs D_2 through D_7 will be used. These will be connected to the A input from our BCD word. This is the MSB of the input.

The completed circuit showing all connections is shown in Figure 9-50. You can wire this circuit and verify it's operation yourself. While this approach does lead to a design implemented with a single integrated circuit, the 74151 is an MSI device and therefore much more expensive than the 7400 SSI device used earlier to implement this function.

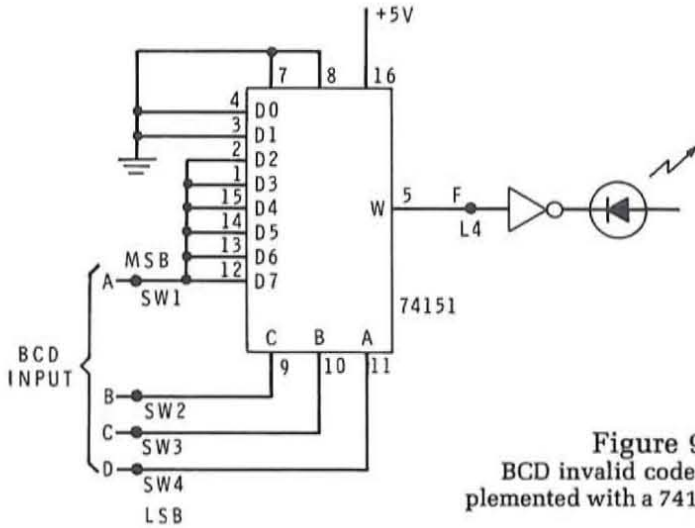


TABLE II

INVALID BCD STATES

A	B	C	D
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

TABLE III

74151 INPUT DEFINITIONS

INPUT ADDRESS C	B	A	ENABLED INPUT
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7

Figure 9-50
BCD invalid code detector implemented with a 74151 data selector.

Procedure (Continued)

4. Using a 74151 data selector IC, implement the two-of-four detector circuit described in the text. Use Figure 9-31 as a reference. Determine the appropriate pin connections yourself, wire the circuit and verify it's operation. Use data switches SW1 through SW4 as the input and one of the LED indicators to designate the output. Record your data in Table IV.
5. Using the integrated circuits supplied with this program, implement the three-input majority detector circuit described in the text. Use Figure 9-41 as a guide. Sketch the logic diagram, implement the circuit and verify its operation. Record your data in Table V.

TABLE IV

A	B	C	D	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

TABLE V

A	B	C	M
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Discussion

Figure 9-31in the text shows how to implement the two-of-four detector circuit with a 74151 data selector. However, some additional anaylsis is desirable in order to verify the operation of this circuit. Refer to Tables VI and VII. Table VI is the truth table for our circuit which defines the output states F for the sixteen possible four-bit input states. A binary 1 output occurs when any two of the four inputs are binary 1. If you will look closely at the truth table you will see that the three least significant bits of the input (B, C and D) repeat themselves. This three-bit 8-state pattern occurs twice in sixteen possible states, once when the input A is binary 0 and the other when the input is binary 1. To determine which data selector inputs are connected to the most significant bit of the input word (A) you analyze the input truth table. For example, you want the output F to be binary 1 when input states 1001, 1010 and 1100 occur. You study the three least significant bits of these input states to determine what input on the data selector they will enable. In this case they will enable inputs D_1, D_2 and D_4. Therefore, to these inputs you will apply the A input from the four-bit input word. When A is binary 1, the output of the data selector will be 1 when the D_1, D_2, or D_4 states are enabled.

In the same way you want the output F to be binary 1 when the input states 0011, 0101 or 0110 occur. Studying the three least significant input bit you see these correspond to data selector inputs D_3, D_5 and D_6. To these inputs you will apply a binary 1 signal when the A input is binary 0. This signal can be developed from the A input with an inverter.

TABLE VI
TWO OF FOUR DETECTOR

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

TABLE VII

74151 INPUTS

C	B	A	ENABLED INPUT
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Translating this analysis into an actual circuit we arrive at the configuration shown in Figure 9-51. The four-bit input word is derived from the data switches and a 7404 inverter IC can be used to obtain the complement of the most signficant bit A. The output of the data selector is monitored on LED indicator L4. This output F will be binary 1 when two of the four input bits are binary 1. You should have constructed the circuit and verified its operation. The operation should correspond to the truth table in Figure 9-27 (and Table VI).

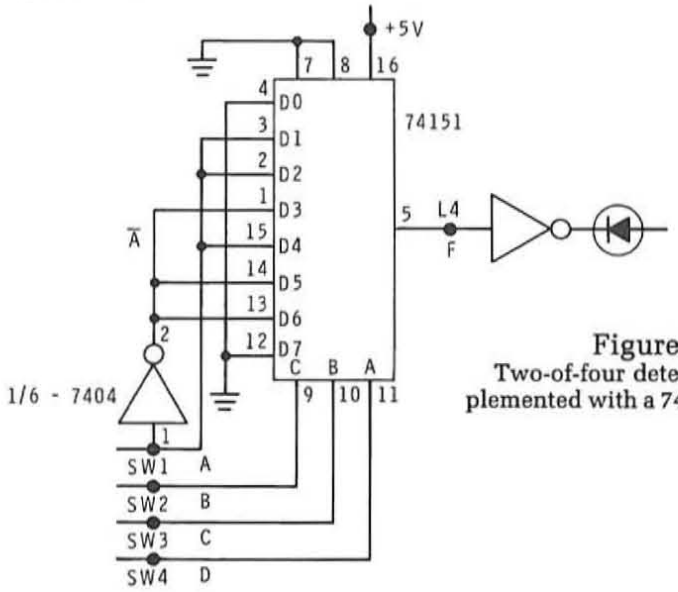


Figure 9-51
Two-of-four detector circuit implemented with a 74151 data selector.

Figure 9-52 shows how to implement the majority detector circuit with available integrated circuits. This circuit is similar to that shown in Figure 9-41. A type 7420 four-input NAND gate is used as a three-input gate by simply connecting two of the inputs together. In verifying the operation of the circuit, you should have found it to be similar to that described in the truth table of Figure 9-39.

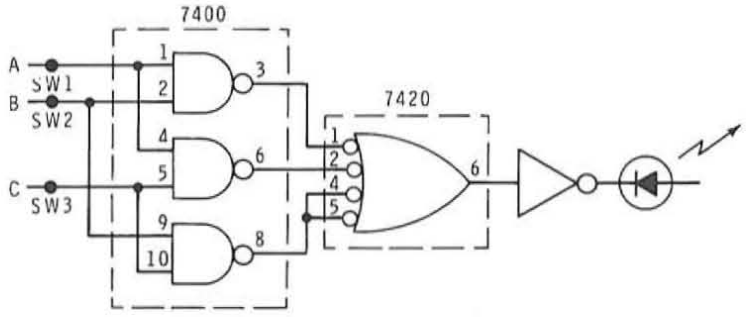


Figure 9-52
Majority detector circuit.

Procedure (Continued)

6. Design a combinational logic circuit that will convert the XS3 BCD code into the standard 8421 BCD code. After you have designed the circuit, implement it with the integrated circuits supplied with this program. Use the data switches as the input source and monitor the outputs on your 7-segment LED display. Use the 9368(443-694) BCD to 7-segment decoder driver IC and LED to monitor the BCD output of your code converter circuit. Implement the circuit and verify its operation. Use the design procedure described earlier in this program.

Discussion

Your design should have exactly followed the procedure given earlier in this unit for designing combinational logic circuits. In this problem the definition is very clear: convert the XS3 BCD input code into the standard 8421 BCD output code. This automatically defines the circuit as having four inputs, four outputs, and ten discrete states. From this definition you can go immediately to the truth table defining this problem. Refer to Table VIII. Here the inputs designated A, B, C, and D are the standard XS3 BCD code as we considered it earlier. The output is the standard 8421 BCD output code.

The next step is to write the output equations in terms of the inputs. This will result in four output equations W, X, Y, and Z in terms of the inputs. Since you are going to use Karnaugh maps to reduce these equations it is not absolutely necessary to write out each equation. Instead we can transfer the information directly from the truth table to the Karnaugh map.

	TABLE VIII								
	EXCESS 3 INPUT					8421 BCD OUTPUT			
DECIMAL	A	B	C	D		W	X	Y	Z
0	0	0	1	1		0	0	0	0
1	0	1	0	0		0	0	0	1
2	0	1	0	1		0	0	1	0
3	0	1	1	0		0	0	1	1
4	0	1	1	1		0	1	0	0
5	1	0	0	0		0	1	0	1
6	1	0	0	1		0	1	1	0
7	1	0	1	0		0	1	1	1
8	1	0	1	1		1	0	0	0
9	1	1	0	0		1	0	0	1
	0	0	0	0					
	0	0	0	1					
	0	0	1	0	"Don't Care"				
	1	1	0	1	INPUTS				
	1	1	1	0					
	1	1	1	1					

Figure 9-53 shows the Karnaugh maps for minimizing the XS3 to BCD code converter. Notice that X's are used to mark the six "don't care" input states. There is a map for each of the four output lines. Both the original and minimized equations are given for each map.

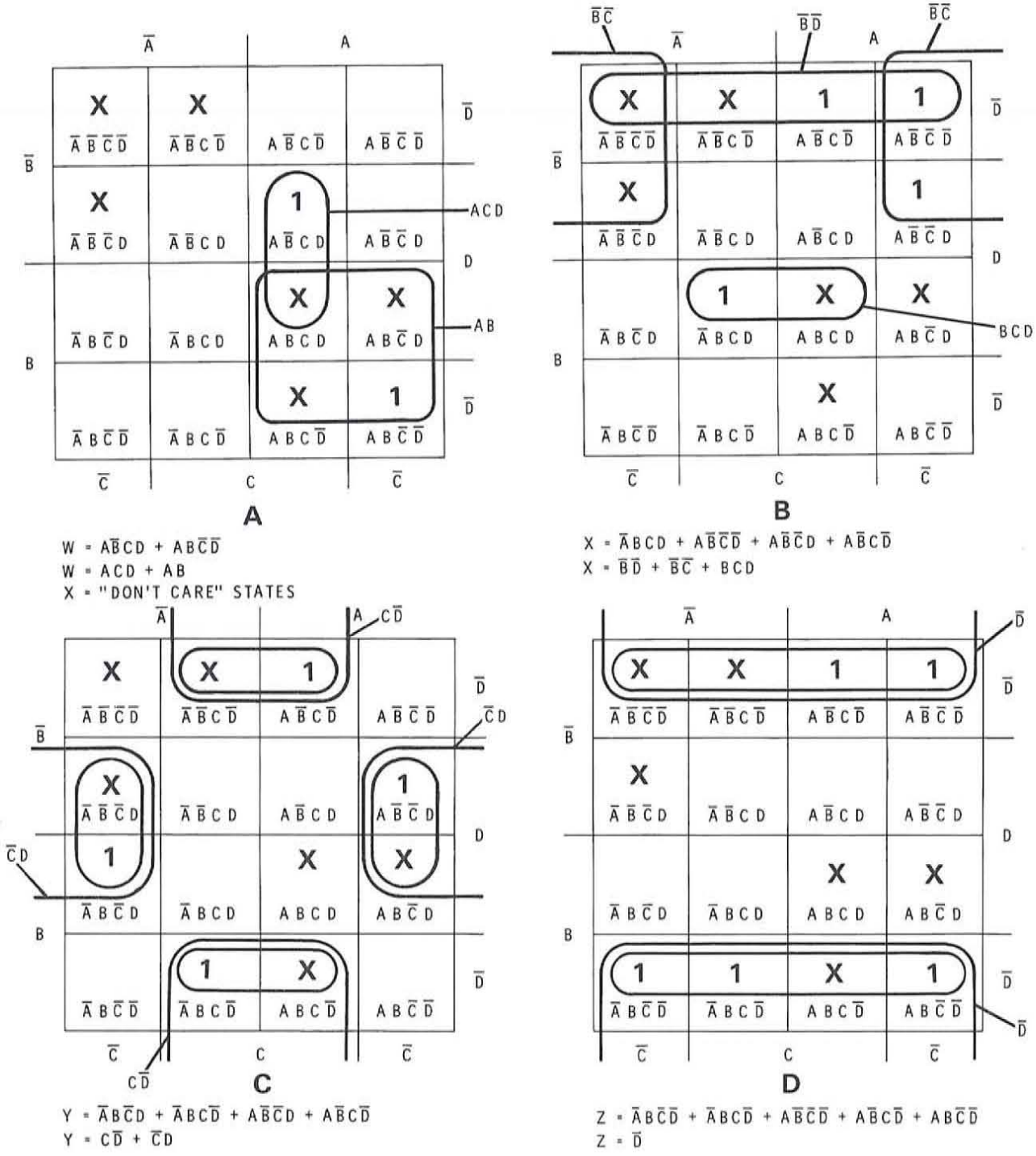


Figure 9-53
Karnaugh maps for minimizing the XS3 to BCD code converter.

There are any number of practical means of implementing this function with integrated circuits. However, you were restricted by the integrated circuits supplied with this program. The easiest approach is to use the SSI circuits available. Figure 9-54 shows how the simplified equations are implemented. A total of five integrated circuits are required. The implementation is straight forward and follows the equations directly. There are two special cases that require explanation however. To implement equation X, three product terms must be ORed together as you can see by referring to the X equation. A three-input OR gate is required. It can be implemented with available 2-input gates. The outputs of gates 4 and 5 are ORed together in gate 9. The output of gate 9 is then inverted to produce a negative OR function. The results are then ORed together in gate 8 with the BCD term from gate 3.

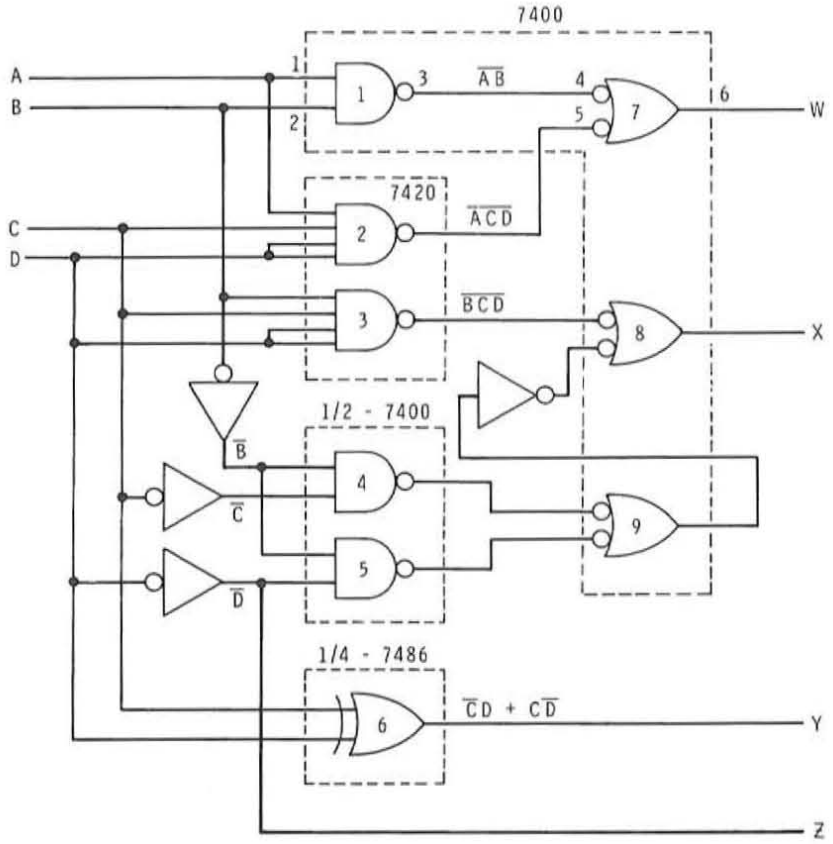


Figure 9-54
XS3 to BCD code converter.

The output Y as you should have identified from the equation is the exclusive OR function. Therefore, one of the exclusive OR gates in the 7486 IC can be used to implement this function directly. The Z output is simply equal to $\overline{D}$ as determined by the simplified equation.

To verify the operation of the circuit you should have then applied the XS3 input code from the data switches SW1 through SW4. The output of the code converter circuit is then connected to the 9368 BCD to seven-segment decoder driver IC. This IC then, in turn, drives the 7-segment display. The decimal output on the LED display should correspond to the SX3 input code as determined in your original truth table. See Figure 9-55.

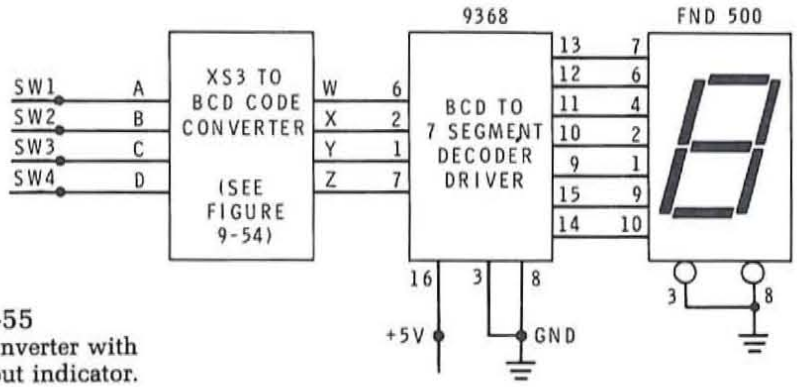


Figure 9-55
XS3 to BCD code converter with 7-segment LED output indicator.

SEQUENTIAL LOGIC CIRCUIT DESIGN

A sequential logic circuit is one designed to deal with a sequence of logic operations occuring over a period of time. The circuit may generate a sequence of timing pulses for use in controlling the operation of other circuits. In such an application the sequential logic circuit is used to **automate** a particular function. It will carry out a certain programmed sequence of events in the proper order and in the proper time sequence. Alternately, a sequential logic circuit may **process** logic signals occuring in a particular sequence. A sequential circuit for example, may be designed to detect a certain sequence of events and respond by generating output signals to other circuits.

In order to carry out these functions, a sequential logic circuit must in some cases be capable of making logical decisions. Logical decision-making of course is carried out by combinational logic circuits. This can be a simple logic gate or a more complex functional logic circuit. Most sequential circuits will contain some form of combinational logic circuit for decision making purposes.

The key feature of a sequential logic circuit is its ability to store data. In order to generate a desired sequence of output pulses, the sequential circuit must have some type of memory so that it can keep track of its sequence. The major element in any sequential logic circuit is the flip-flop. Flip-flops are interconnected with the combinational logic circuit to perform the desired function.

A sequential logic circuit responds to the various inputs applied to it. In return it generates specific output pulses depending upon its function. The output signals are a function not only of the input states, but also the current state of the sequential circuit as stored in the flip-flop memory.

Figure 9-56 shows a general block diagram of a sequential logic circuit. The heart of the circuit is the flip-flop memory where the state of the circuit is determined. The flip-flops are generally controlled by clock pulses. The flip-flop outputs drive the combinational logic circuits. The combinational logic circuits are also driven by various input signals. The combinational logic outputs in turn drive the flip-flops and can also drive external circuits as required.

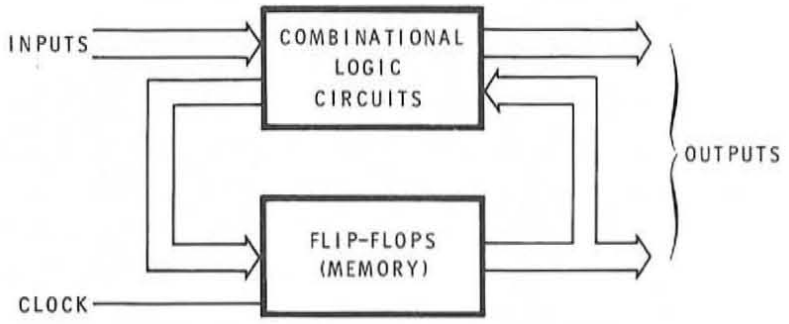


Figure 9-56
General block diagram of a sequential logic circuit.

The number of flip-flops in the sequential logic circuit determines the total number of different states in which the circuit can exist. These states are defined by a particular binary code which is stored in the flip-flops. As the inputs are applied and clock pulses occur, the state of the sequential logic circuit will change. Flip-flops will become set or reset according to a particular desired pattern. The flip-flop states may be interpreted as a special form of binary code. These states, in turn, control the combinational logic circuits to cause the correct sequence of events to occur and to generate the sequence of output pulses.

You have already studied the most common forms of sequential logic circuits. These are counters and shift registers. Standard binary and BCD counters are made up of flip-flops and in some cases combinational logic circuits. Shift registers are another form of sequential logic circuit. In most design situations a standard counter or shift register can generally be used to meet the need for a sequential logic circuit. However, there are many applications where special sequential circuits can result in benefits. Unusual applications and special functions are readily implemented with special forms of sequential logic circuits. When unique codes and sequences are required, special sequential logic circuits can often result in a more efficient design using fewer parts which operate at higher speeds.

The most common way of classifying sequential logic circuits is the process by which the circuit changes from one state to the next. The state transition can occur synchronously with the application of an external clock pulse or asynchronously. In a synchronous sequential circuit, the speed of operation is a function of the clock frequency. The states of the circuit change in step with the clock pulses. In an asynchronous circuit, the state changes occur as a result of previous changes. The speed of operation is strictly a function of the propagation delay times in the circuit and the rate of occurence of any external signals. Because most IC logic circuitry has such short propagation delays, very high speed operations can be obtained with asynchronous circuits. However, because of the unequal propagation delays of the various gates and circuit paths, unreliable operation can occur. False triggering and invalid states can prevent circuits from operating correctly. The design of an asynchronous circuit is also more difficult since it requires an analysis of the possible fault conditions that can occur and some means of correcting these faults. Generally, synchronous circuits are easier to design, implement and control, and therefore are recommended over asynchronous designs. This section emphasizes the design and application of synchronous sequential circuits using JK flip-flops.

Design Procedure

Most of the sequential logic circuits in use today are some form of special counter. Like the standard binary and BCD counters you have studied before, these special counters will sequence through a number of states in response to the input signals applied. The flip-flops in the counters will set and reset according to a specific count sequence and generate the output pulses called for by the application. Such sequential circuits can be counters that implement a specific code or frequency dividers that have a required number of states. When sequential circuits like this are used for control purposes, the counters are normally referred to as sequencers or controllers. A wide variety of different code types can be implemented in this application, but in controllers and sequencers the codes most often used are cyclical codes where only one flip-flop in the counter changes state at a time.

The design procedure to be described here shows one method of designing special counters and sequencers. This procedure will provide you with a means of designing such circuits for most of the applications that you will encounter. Keep in mind that it is only one of many different techniques. Alternate design methods are available but most applications can be manipulated such that the type of circuit described here and the procedure to design it can be used.

In describing and illustrating this design procedure we will assume the use of synchronous circuits using JK flip-flops. This is the most commonly used approach and the one that will result in the most versatile and reliable design.

The design procedure for sequential logic circuits is as follows:

1. State the problem and completely define the design objectives.
2. Develop a state table from the problem definition.
3. Develop Karnaugh maps for the flip-flop inputs from the state table.
4. Write the input equations for the flip-flops from the Karnaugh maps.
5. Draw the complete logic diagram from the logic equations.
6. Implement the circuit using standard integrated circuits.

Defining the Problem. The first step in designing a sequential logic circuit is to define the required objective. This is best done by writing down a complete but concise description of the function to be performed. State explicitly what operations are to occur. There are many different ways of expressing the logic function to be performed. There is no standard method for doing this, but the most important part of it is to include all possible conditions. As part of the problem definition you should specify the characteristics of the input and output signals and specify or determine the number of states that the circuit must assume.

The circuit inputs and outputs can be expressed in several different ways. These may take the form of logical waveforms that define the sequence of functions to be performed. Alternately, these may be logic levels that occur at specific times which cause certain operations to take place. The inputs and outputs can also be expressed in the form of a truth table or state table. A state table is similar in format to a truth table, in that the input and output signals are expressed in terms of 1's and 0's in table form that shows the sequence of change at each of the desired steps or states defined by the problem.

Specifying the input and output signals and writing out a description of the function to be performed will generally decide or specify the number of states in which the circuit can exist. The number of steps in the sequence of operations to take place will generally determine the number of circuit states. When the number of states have been determined, this will tell you the number of flip-flops that the memory section will contain. These flip-flops will define a binary word. The sequence of the steps designates how the flip-flops change state. This in turn defines a specific code sequence. The code sequence may be the standard binary code or it could be any one of a number of special codes such as XS3 or Gray Code. Of course, any binary sequence can be selected and implemented if it is required by the application.

Developing a state or flow table. Having determined the number of states required by the application, you should now be able to develop a state or flow table that completely defines all of the states in the circuit. A state or flow table is simply a truth table that expresses the outputs of the flip-flops in the circuit for each of the states required by the application. Knowing the number of states required, the number of flip-flops can be calculated using the techniques described earlier. For example, an application requiring seven states would require a 3-bit counter. A 3-bit counter will produce $2^3 = 8$ states. Two flip-flops are insufficient since they will produce a total of only $2^2 = 4$ states. Three flip-flops will handle the application properly. One of the eight states will not be used.

Figure 9-57A shows a state table or flow table for an application requiring seven states. The states have been specified by the application, and are numbered 0 through 6. The eighth state (7) whose binary output is 100 is not used but is generally included in the table and labeled as not being used. Note that the circuit contains three flip-flops labeled A, B, and C. The binary code for each state bears no relationship to the decimal state assigned to it. The flip-flop outputs in such a case are simply treated as a bit pattern rather than a binary number.

A

	FLIP FLOP OUTPUTS	
STATE	A B C	
0	0 0 0	
1	1 1 0	
2	0 1 0	
3	1 0 1	RECYCLE
4	0 0 1	
5	0 1 1	
6	1 1 1	
7	1 0 0	NOT USED

B

	t	t + 1
STATE	A B C	A B C
0	0 0 0	1 1 0
1	1 1 0	0 1 0
2	0 1 0	1 0 1
3	1 0 1	0 0 1
4	0 0 1	0 1 1
5	0 1 1	1 1 1
6	1 1 1	0 0 0
7	1 0 0	NOT USED

Figure 9-57
State or flow table for a 7-state counter or controller.

To interpret the flow table in Figure 9-57A, you simply observe the flip-flop states as the circuit changes from one state to the next sequentially moving from top to bottom. The state column indicates that the 0 state is the initial condition state and the other states occur sequentially as shown. When state six occurs (binary output 111) the circuit will then recycle back to the initial zero state upon the application of the next clock pulse.

Figure 9-57B shows another method of constructing a flow or state table. The state column and the section labelled t are similar to the flow table in Figure 9-57A. An additional section labelled t + 1 is also included. The t indicates the state of the flip-flop outputs **before** the application of a clock pulse. The section t + 1 indicates the state of the flip-flops **after** the occurrence of one clock pulse. The information in this table is exactly the same as that in the flow table of Figure 9-57A. Only the format is different.

The flow tables shown in Figure 9-57 illustrate one of many possible special codes that can occur as the result of developing a sequential circuit for a specific application. It is possible that the code illustrated was derived strictly from the desired flip-flop output states at each state in the circuit. This table may have been developed by observing output waveforms that were originally specified as required by the application to produce the specific timing and sequencing required. Studying the table we see that no recognizable standard code exists. It is simply a random or special code that meets the particular application.

There will be other applications where a standard code may be used or specified. The problem may call for the pure binary code, a BCD code, excess 3 code, or the Gray code. In still other applications no standard code will be specified or required. In addition, a particular application may not be code sensitive, that is any code can be used. In such situations it is generally desirable to use some form of standard code. The conventional binary code is desirable since a counter can be easily constructed. This results in hardware simplifications.

For most sequential circuit applications where a special counter or sequencer is required, the most desirable approach is to implement a circuit in which only one flip-flop changes state at a time when the circuit is stepped from one state to the next. The Gray code is an example of such a code. Such a circuit can be made to operate faster than other types of counters. Unequal propagation delays in the various circuit components will not cause false triggering or spurious undesirable output pulses known as **glitches**. If more than one flip-flop changes state at a time, the unequal propagation delays of the flip-flops can cause momentary false states. Gates used to decode the various counter states can then produce very short duration pulses equal in length to the difference between the propagation delay time changes in the various flip-flops. These glitches can cause false circuit triggering. By changing only one flip-flop at a time such pulses are eliminated or very greatly minimized. In addition, it is also desirable to use a synchronous circuit so that all flip-flops in the counter are clocked at the same time. This too helps to minimize glitches.

The Gray code is only one of many cyclical codes in which only one bit of the code word changes at a time from one state to the next. There are many different combinations possible and you may use any combination to achieve the desired end result.

Another useful guideline in developing a special code for a counter or controller is that the initial state be assigned 0. All flip-flops should be reset for the 0 or initial state of the circuit. Most circuits will have a reset, resting or wait state from which all operations begin. By making this state equal to 0, it becomes easy to identify. The direct clear inputs on most JK flip-flops can also be tied together to generate this initial or reset state by simply bringing this line low. This line can then be controlled from a pushbutton or by a special circuit that will automatically place the counter in the 0 state when power is applied.

Developing a Karnaugh Map For The Counter. Another way of showing the states of a special counter or sequencer is to use a Karnaugh map. Each cell or square in the map indicates a specific state. The decimal number corresponding to each state can be written into the cell corresponding to the binary code produced by the circuit. Arrows can then be drawn on the table to indicate the sequence of flow as defined by the state table. The Karnaugh map produces a visual means of indicating the states of the sequential circuit. Figure 9-58 shows a Karnaugh map that plots the seven-state controller defined by the tables in Figure 9-57.

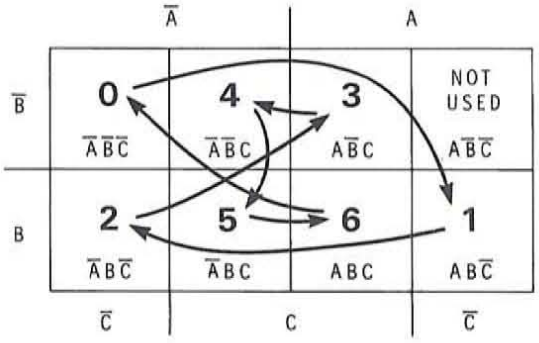


Figure 9-58
Karnaugh map for the seven-state counter defined by the table in Figure 9-58.

Besides helping you visualize what your counter or sequencer is doing, it can also help you in selecting a suitable cyclical code where only one bit changes from one state to the next. Recall from our earlier discussion of Karnaugh maps that adjacent cells in the map represent a change in only one of the variables. In moving from one cell to the next only one of the variables will change. Therefore to create a special code all that is necessary is to choose an initial starting point and then move from one cell to the next as many times as required to generate the special code.

Figure 9-59 shows several examples of how the Karnaugh map can be used to develop a special code. In all cases the initial starting point is 000 or $\overline{A}\,\overline{B}\,\overline{C}$. In each example, the counter or sequencer has 6 states. Six of the cells in the map, therefore are involved. Note that in moving from one cell to the next only one variable changes at a time. Note particularly the state change involved in recycling from the sixth state to the initial state. Remember that the Karnaugh map can be treated as a cylinder where the left and right edges of the map are considered to be adjacent.

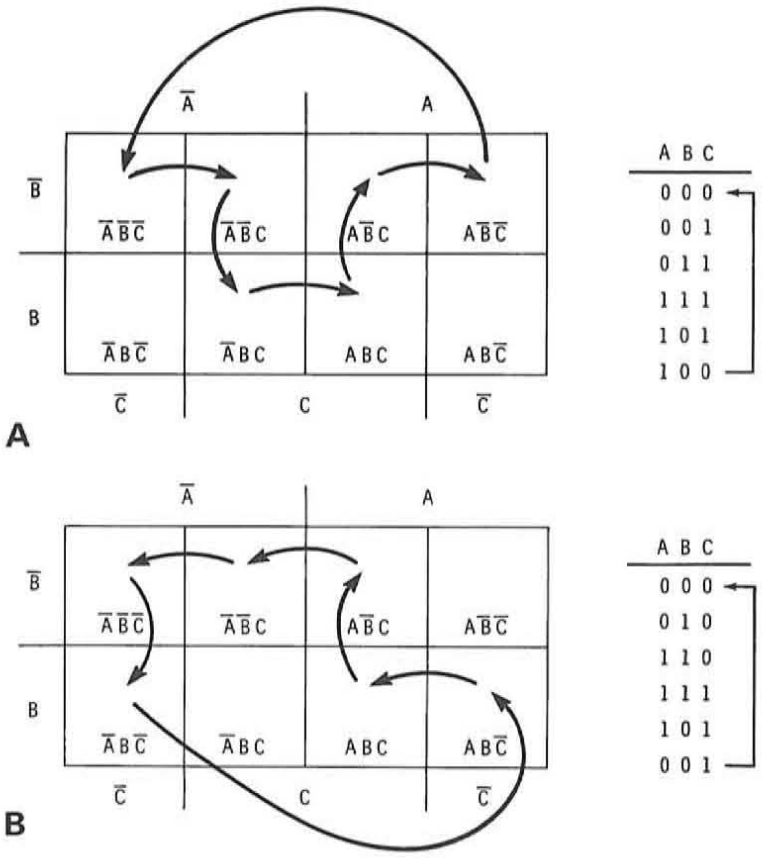


Figure 9-59
Examples of Karnaugh maps defining six-state cyclical counters.

While we have indicated six states in these examples, controllers of any number of states can be developed using this same technique. It is not always possible to generate a code where only one bit changes from one state to the next. This is particularly true of counters or sequencers with an odd number of states. It may be possible to generate a code that changes from only one state to the next, but in recycling more than one bit may have to change in order to have the code return to its initial state.

Figure 9-60 shows a five-state code where only one bit changes from the initial state through the five code states. But in recycling from the last state 110 to the initial state 000, two bits change. Generally, such conditions are not detrimental. Where they are, an even number of states could be introduced. The extra or unneeded state, called a **dummy** state, would not be used by the application but would serve only as a means of recycling the counter by having only one flip-flop change state at a time.

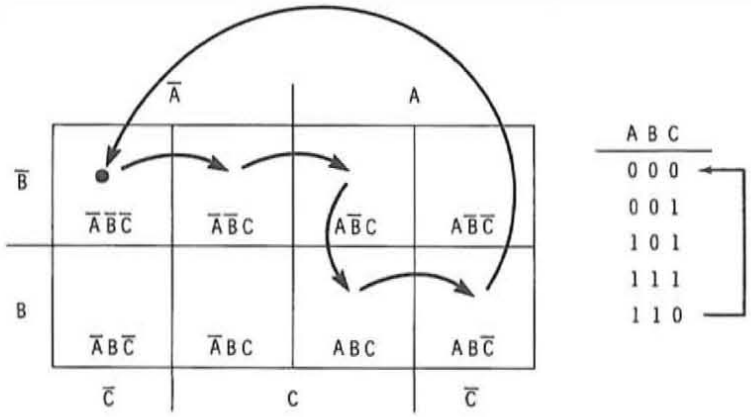


Figure 9-60
Examples of a Karnaugh defining a five-state code.

The most important application of a Karnaugh map in designing a counter or sequencer is in determining the input states to the flip-flops required to produce this special code specified. In using JK flip-flops, certain states must be applied to the J and K inputs to cause the flip-flops to set and reset in the desired code sequence. To determine these inputs, we observe the state table and indicate which flip-flop must be set or reset in changing from one state to the next. The set or reset conditions are then plotted on the Karnaugh map. Then, by properly grouping these plotted input states, the input logic equations for the various flip-flops can be found.

To plot the state change Karnaugh maps for each flip-flop in the counter, you will mark each cell in the map with a symbol that designates the state change that is to take place. There are five possible conditions that can occur. These are:

1. Flip-flop changes from reset to set.
2. Flip-flop changes from set to reset.
3. Flip-flop is set and remains set.
4. Flip-flop is reset and remains reset.
5. Don't care.

To indicate these five state changes, or conditions, the symbols below are used.

1 Flip-flop changes from reset to set.
/ Flip-flop is in the set state and remains in the set state.
ϕ Flip-flop changes from set to reset.
0 Flip-flop is initially reset and remains reset.
X Input conditions do not occur or a "don't care" state exists.

These state changes are summarized by the table in Figure 9-61. The left-hand column shows the symbol used to indicate the state change. The t column represents the state of the flip-flop prior to the application of a clock pulse. The t + 1 column indicates the state of the flip-flop after the clock pulse.

SYMBOL TO REPRESENT STATE CHANGE	t	t + 1
1	0	1
/	1	1
ø	1	0
0	0	0

t = BEFORE CLOCK PULSE
t + 1 = AFTER CLOCK PULSE

Figure 9-61
Symbols representing flip-flop state changes.

The state table is then analyzed to determine how each flip-flop change changes from one state to the next. These state changes are then plotted on the Karnaugh maps, one map for each of the flip-flops in the counter.

The symbol to be plotted in each cell of the Karnaugh map designates the state change that must take place in the output variable associated with the map in moving from one state to the next. For example, if flip-flop A is presently reset as indicated by its condition in the cell of interest and must set in order to transfer to the next state, a 1 will be marked in that cell. **The symbol in a cell represents the state change that must occur to move to the next state.**

This is best illustrated by developing the Karnaugh maps for the flip-flops in the special 7-state counter discussed earlier. The maps for this circuit are shown in Figure 9-62.

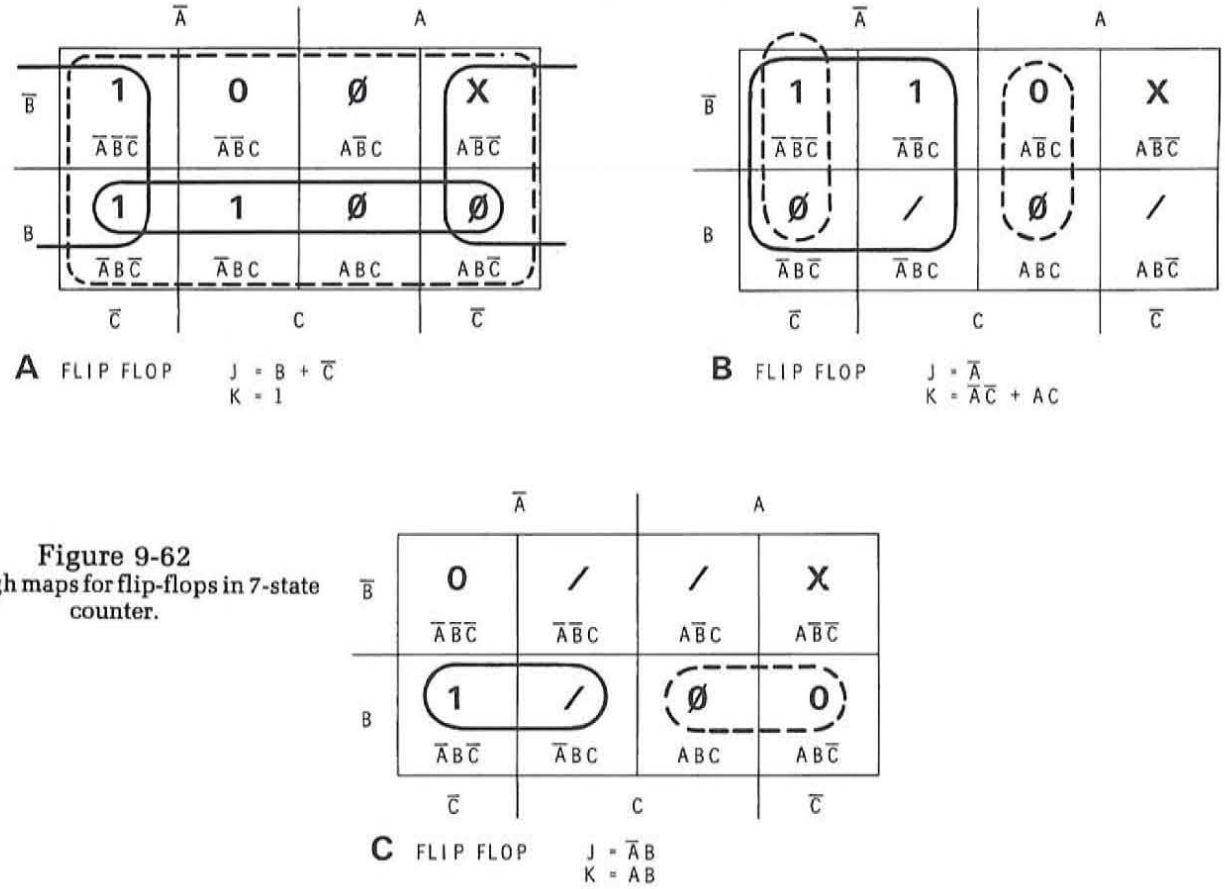


Figure 9-62
Karnaugh maps for flip-flops in 7-state counter.

In each cell is recorded the symbol that designates the state change that will take place in moving from the current state to the next state. Using the state tables in Figure 9-57 as a guide, verify the use of the correct symbol in each case. As an example, consider state 2 for the B flip-flop. State 2 is defined by the code 010 or $\overline{A}$ B $\overline{C}$. Locate this cell on the Karnaugh maps for the B flip-flop. To move from this state to the next state, state 3 (Code 101 or A $\overline{B}$ C) the B flip-flop will change from its current set state to the reset state. Therefore, in cell $\overline{A}$ B $\overline{C}$ you will record the symbol indicating that a reset condition must occur. This is the ø symbol.

As another example consider the condition of flip-flop C in going from state 4 to state 5. Here you see that the C flip-flop is set in state 4 (code 001 or $\overline{A}$ $\overline{B}$ C). Locate state 4 on the Karnaugh maps for the C flip-flop. To move to the next state, state 5 (011 or $\overline{A}$ B C), the C flip-flop does not change state, it remains set. Therefore, a / is recorded in the fourth state cell. Be

sure to verify that the proper symbol is used in each cell of each Karnaugh map to obtain practice in reading and developing these maps.

Once you have completely plotted the maps for each flip-flop, you will use them to develop the J and K input logic equations for each. To do this you will group the various terms in the maps together in groups of 2, 4, 8, or higher powers of 2 as you did in minimizing combinational equations. There are some special rules that you must follow with regard to grouping the variables in the maps. For a JK flip-flop these rules are as indicated below.

J input equation

1 Each 1 square must be accounted for in the J equation.
/ Optional
Ø Optional
0 Must not be used
X Optional

K input equation

1 Optional
/ Must not be used
Ø Each Ø square must be accounted for in the K equation.
0 Optional
X Optional

In developing the J input logic equation you must consider each 1 term marked in the cells of the map. Each 1 must be used in some way to account for all of the necessary input states. Cells marked with a 0 must not be used. All other input symbols such as the /, Ø, and X can be used in the same way as "don't care" states in any other Karnaugh map.

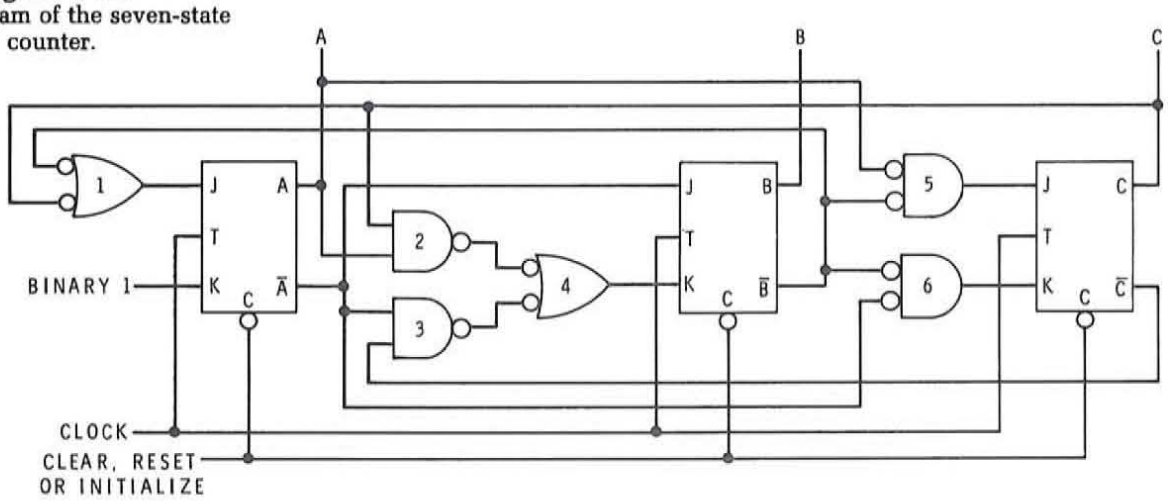
In determining the K input logic equations, all Ø terms must be accounted for and used in one grouping or another. The / cells must not be used. All other cells marked with symbols 1, 0 and X can be used as "don't care" states.

The Karnaugh maps in Figure 9-62 show the proper groupings of the terms of both the J and K input equations. The J input equations are identified by groups marked with solid lines. The K input equations are designated by the dashed lines. The Karnaugh map is read in the same way as you read Karnaugh maps for combinational logic circuits. Simply look at each marked group and determine which variable or variables does not change when moving from one cell to the next within that group. Make a product term with these variables. Then OR together each of these terms, one term for each group. The equations corresponding to the J and K input states are designated adjacent to each map. Using the rules given earlier, verify the correct grouping of the variables. Then derive the input equations for the J and K inputs of each flip-flop yourself from the map to check the equations.

A special case in the map of flip-flop A is that all 8 cells are valid for the K input. This means that we can form one single large group of eight representing the input term for the K input of the A flip-flop. When all terms in a map can be looped together, it simply indicates a binary 1 condition. In other words, input A,B, or C could be in either the binary 1 or binary 0 state and the output would still be binary 1.

Drawing the Logic Diagram. Knowing the number of flip-flops in the counter and having the equations of the J and K inputs for each you can draw a logic diagram for the counter or sequencer. Keep in mind that we are dealing strictly with the JK flip-flops and a synchronous circuit. Figure 9-63 shows the logic diagram of the seven-state counter. The T inputs of each flip-flop are connected together to a common clock input line. This indicates a synchronous circuit. Note that all direct clear inputs of the flip-flops are connected together so that all of these flip-flops can be cleared, reset or initialized from a common input line if desired. The remaining logic circuitry represents logic gates that implement the J and K input equations for each flip-flop. Gate 1 through 4 are positive NAND negative NOR gates such as the type 7400 quad two-input NAND gate. Gates 5 and 6 are positive NOR negative NAND circuits such as type 7402 IC. JK flip-flops like the dual flip-flop 7476 could be used. The resulting circuit is a counter that will sequence itself in the given code with the minimum amount of hardware.

Figure 9-63
Logic diagram of the seven-state counter.



Design Examples

To further illustrate the design procedure for sequential logic circuits, the example designs given here are presented. They show that a counter can be designed with any number of states and to sequence or step from one arbitrary binary bit pattern to another. Special codes or arbitrary sequences are equally easy to design and implement.

Two-Bit Gray Code Counter. Assume that we wish to design a four-state Gray Code Counter. In attempting to achieve high speed we choose the Gray Code since only one bit changes from one state to the next. The fact that we need a Gray Code counter is determined by the application itself. Perhaps the application simply calls for a four-state counter and based on your knowledge of the application you determined that a Gray Code sequence was the most desirable for high speed operation. The Gray Code sequence could also have been determined by the desired output waveforms.

STATE	A	B
0	0	0
1	0	1
2	1	1
3	1	0

RECYCLE

Figure 9-64
State table for 2-bit Gray Code counter.

Once the problem has been completely specified and stated, a state table is developed. Figure 9-64 shows a state table for a two-bit Gray Code counter. The states are labeled 0 through 3 and as usually desirable the initial or first state, state 0, is made equal to 00. It takes two bits to define four states. Note that in changing from one state to the next only one of the two bits changes at a time. This includes the state change from the fourth state, state 3, back to the initial state, state 0.

The next step of the design procedure is to plot the state changes for each flip-flop on a Karnaugh map. To do this you examine the state changes that must occur in each flip-flop in moving from one state to the next. These state changes are then plotted in the appropriate cells on the Karnaugh map. In each cell you plot a symbol that designates the transition that must take place to move to the next state. Use the symbols given earlier for designating the state changes.

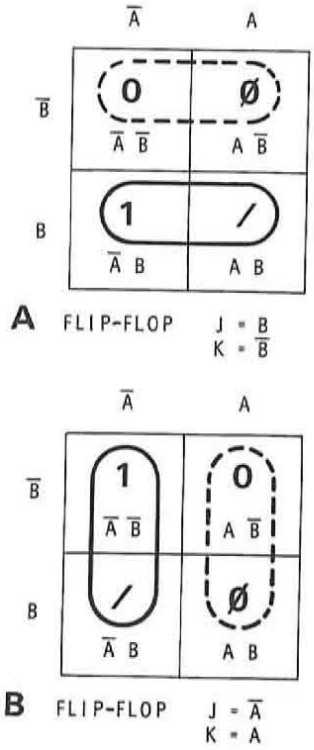


Figure 9-65
Karnaugh maps for flip-flops in 2-bit Gray Code counter.

Figure 9-65 shows the Karnaugh maps for the A and B flip-flops with the appropriate state changes plotted. The various symbols in this table are then grouped according to the directions given earlier to determine the input expressions for the J and K inputs on each flip-flop. The minimized input expressions are given adjacent to the Karnaugh maps.

From the information derived from the Karnaugh maps, a logic diagram can be drawn. This is illustrated in Figure 9-66. The two JK flip-flops are interconnected as specified by the J and K input expression given in Figure 9-65. Since this is a synchronous circuit, the toggle inputs to the flip-flops are tied together to the clock circuit.

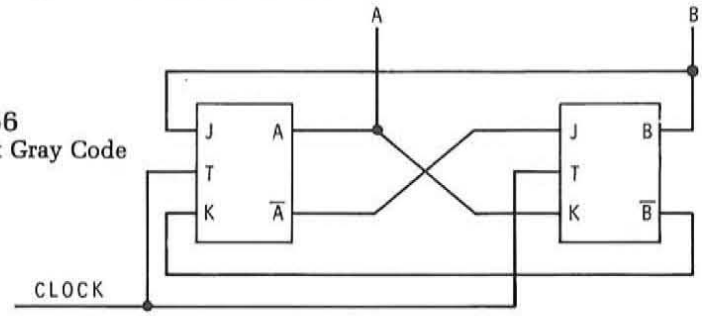


Figure 9-66
Logic diagram of 2-bit Gray Code counter.

Be sure to work through this example problem yourself. This includes developing the initial state table, plotting the state changes on the Karnaugh maps, developing the input equations and sketching the logic circuit. Verify each step of the example to be sure that you understand what was done.

	OUTPUTS	
STATE	A B C D	
0	0 0 1 1	←
1	0 1 0 0	
2	0 1 0 1	
3	0 1 1 0	
4	0 1 1 1	RECYCLE
5	1 0 0 0	
6	1 0 0 1	
7	1 0 1 0	
8	1 0 1 1	
9	1 1 0 0	←

Figure 9-67
State Table for Excess 3 counter.

XS3 Code BCD Counter. Design a counter circuit that will count in the standard XS3 BCD Code. We know from the statement of the design problem that a ten-state counter is required. A BCD counter is a decade counter with ten states. The code is also defined for us. The standard XS3 code is specified. From this information we can immediately develop a state table. This is shown in Figure 9-67. The initial or zero state is 0011. From there the counter steps in a standard binary code sequence until the tenth state (1100) is reached. The counter then recycles on the tenth input pulse.

The next step in the design process is to translate the flip-flop transitions in the state table into the symbols that can be plotted on the Karnaugh map. Use Figure 9-61 as a guide.

Figure 9-68 shows four sixteen-cell Karnaugh maps used to plot the state changes for each of the four flip-flops. Since the counter has only ten states, six of the four states will not be used and therefore can be treated as "don't care" conditions. These are states 0000, 0001, 0010, 1101, 1110, 1111. X's are placed in the appropriate squares in all four Karnaugh maps so that these "don't care" conditions can be used in minimizing the input equations for each flip-flop.

Next, the transitions of each flip-flop from one state to the next are analyzed and plotted on the appropriate Karnaugh map. Go through each of these yourself to see how the various transition changes were determined.

Using the rules given earlier about grouping the symbols in the Karnaugh maps, develop the input expressions for each flip-flop. The Karnaugh maps indicate the appropriate groupings and the resulting logic equations for the JK inputs. Solid line groups are for the J inputs while dashed line groups are for the K inputs.

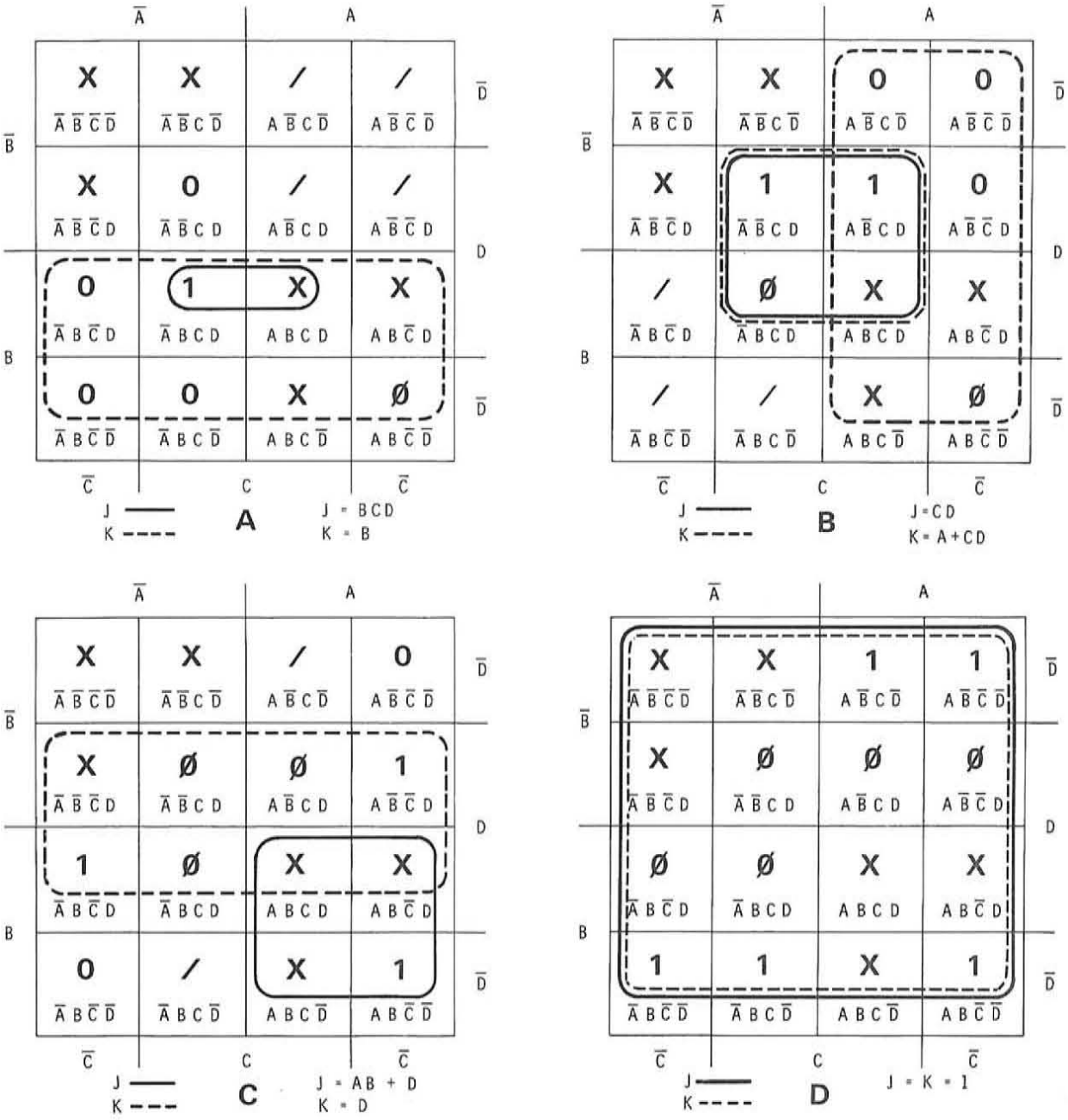


Figure 9-68
Karnaugh maps for the XS3 counter.

From the input equations developed from the Karnaugh maps, the logic diagram can be sketched. Figure 9-69 shows one method of implementing this circuit. All of the T inputs to the flip-flops are connected together to form a synchronous circuit. The J and K inputs to each flip-flop are specified by the input equations. This gating assumes the use of standard SSI logic packages.

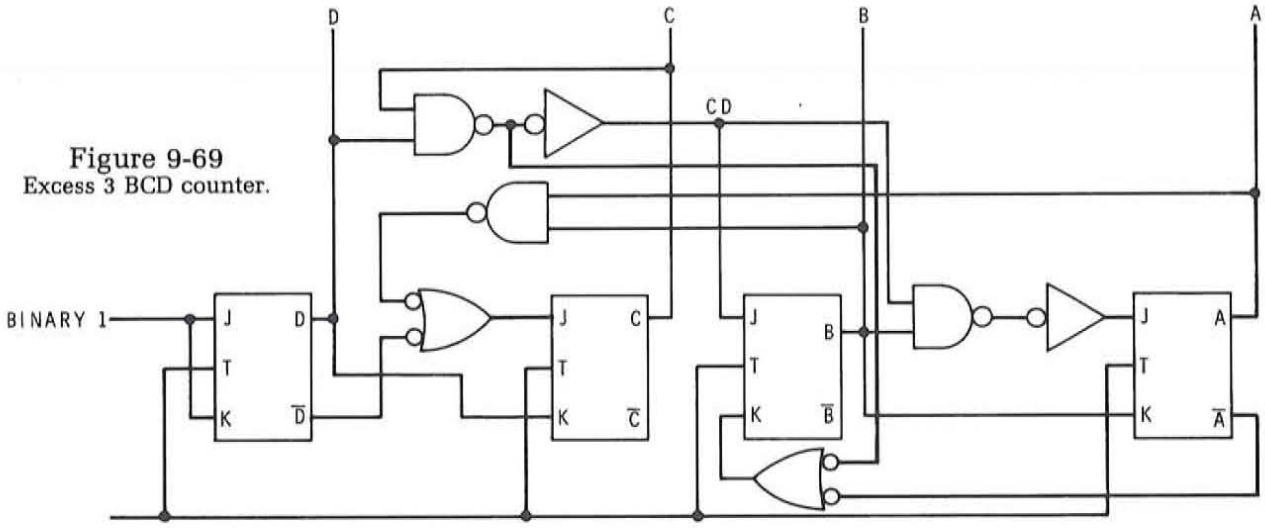


Figure 9-69
Excess 3 BCD counter.

Design Variations

The sequential circuits that we have considered are special counters that can have any number of states and any special code sequence. All of these circuits have a single input, the clock. However, there are other sequential circuits where external control signals are used to control the counter. These external inputs essentially determine when a counter or sequencer steps from one state to the next. The only modification needed in our design procedure to handle external inputs is to include these signals as variables in the JK input expressions. In order to cause a circuit to change from one state to the next, the JK inputs of the various flip-flops must have the appropriate input signals as determined by the count sequence of the counter. If an external input signal is to have control over the change from one specific state to the next, then that external input signal becomes one of the product terms in the expressions for the JK inputs on each of the relevant flip-flops.

This concept can be simply illustrated with the two-bit Gray Code counter discussed earlier. Assume that we wish to have an external start signal to control the counter. In other words we wish the counter to remain in its initial 00 state until it receives a binary 1 signal on the START input line. Once the START signal goes high, the counter will be incremented by the clock pulses from one state to the next. This sequence will continue until the START line is brought low. Then the counter will stop counting.

Figure 9-70 shows one way that this circuit could be implemented. An AND gate is connected to the J input of the B flip-flop. Normally this J input is connected directly to the $\overline{A}$ output. The $\overline{A}$ output is used as one input to the control AND gate. A START signal is also applied to the AND gate. Now in order for the B flip-flop to set, the $\overline{A}$ output must be high and the START input must be high. With the counter initially in its 00 or reset state, the JK inputs on each flip-flop are 1 and 0 respectively. As clock pulses are applied both flip-flops will continue to be reset. When the START input line goes high, the AND gate output goes high making the J input to the B flip-flop high. The K input to the B flip-flop is low as it is connected directly to the normal output of the A flip-flop. The conditions are now correct for the B flip-flop set on the occurrence of the next clock pulse. When this happens, the normal count sequence of the Gray code counter will begin. The counter will continue to count as indicated by the waveforms in Figure 9-71.

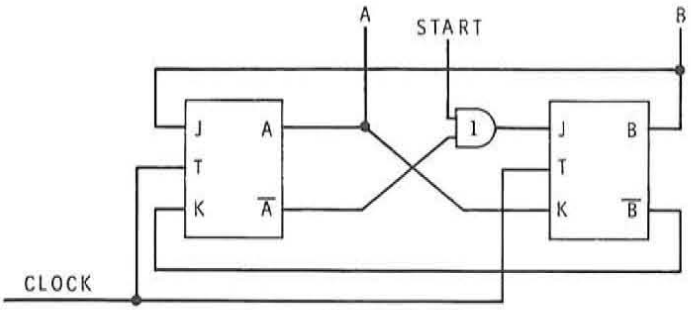


Figure 9-70
Two-bit Gray Code counter with external control input.

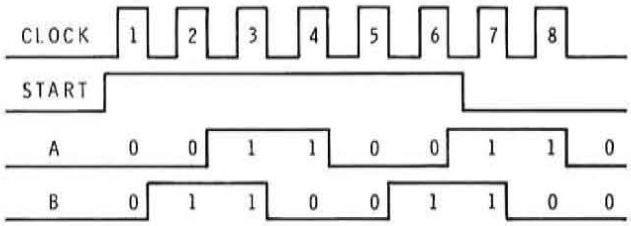


Figure 9-71
Waveforms for two-bit Gray Code counter with external control input.

If the START line should go low during the count sequence as indicated in the waveforms of Figure 9-71, the counter will continue to run until the 00 state is reached at which time the counting sequence will stop. The counter will remain reset until the next start pulse is applied. While this is a simple example, it does illustrate the concept of using external signals to control the occurrence of the state changes in a special counter or sequencer.

In using these special counter or sequencer circuits, we will often be able to use the flip-flop outputs directly to control external circuits. In such a case no additional circuitry is required. It is possible to define the control waveforms required and then design a counter to produce the desired count sequence thereby minimizing the circuitry. Another approach to obtain a sequence of timing pulses is to decode the state of the special counter. AND gates can be connected to the flip-flop outputs to recognize each unique state produced by the counter. The outputs of these decode gates can then be used to control the sequence of operations in external circuits.

Figure 9-72 shows how all four states of the two-bit Gray Code counter are decoded. The output signals produced by the decoder gates are illustrated in Figure 9-73. Note that as the counter steps from one state to the next, a sequence of timing pulses is generated. These pulses are then used to control the external circuits. In some applications all states of the counter must be decoded to create the necessary timing pulses. In other applications only specific states may be required thereby minimizing the number of decode gates required. For three- and four-bit counters, MSI decoder circuits can be used to reduce the amount of circuitry required to decode the desired states.

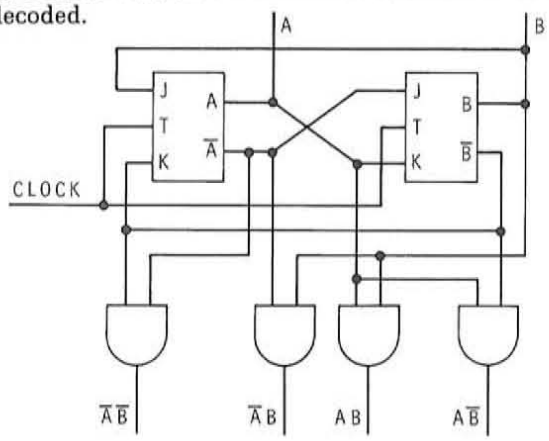


Figure 9-72
Two-bit Gray Code counter with all states decoded.

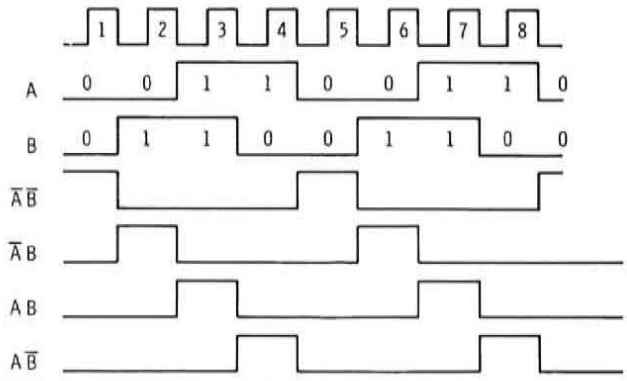


Figure 9-73
Waveforms for two-bit Gray Code counter and decoder.

Self Test Review

The purpose of these self test review problems is to give you practice in designing digital circuits. The example problems given here combine both the techniques of combinational and sequential logic design. As in any design situation the problems are wide open to interpretation based on your knowledge and experience. There is no single perfect way of designing and implementing a given circuit. For most applications a variety of methods are suitable. For the problems here, however, we will give you hints to guide you in developing a circuit based on the design techniques you have learned here. Keep in mind that our design emphasizes high performance for minimum cost, size and power consumption. We will also emphasize the use of digital integrated circuits, primarily the TTL type which are supplied with this program. The examples given here will permit you to breadboard these circuits to verify your designs.

5. Design a digital die. Many games use dice to randomly select a number that is used in determining the outcome of the game. It is possible to design and build digital dice where the marks on the dice

can be simulated by indicator lamps. Figure 9-74 shows an arrangement of indicator lamps labeled T through Z. When the appropriate lamps are illuminated, the numbers 1 through 6 will be represented in standard die format. The objective of this design problem is to develop the circuitry necessary to randomly select a number 1 through 6 and display it. Assume that the indicator lights in the die are light-emitting diodes (LEDs) and are driven by a saturated transistor switch as shown in Figure 9-75A. The type of transistor and the values of R1 and R2 are not important except to indicate that the value of R2 is such that Q1 will saturate and the indicator LED will turn on when a standard TTL binary 1 level is applied to R2. An open collector TTL inverter can also be used as indicated in Figure 9-75B.

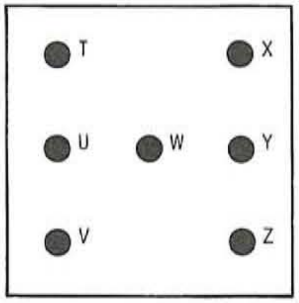


Figure 9-74
Standard die format.

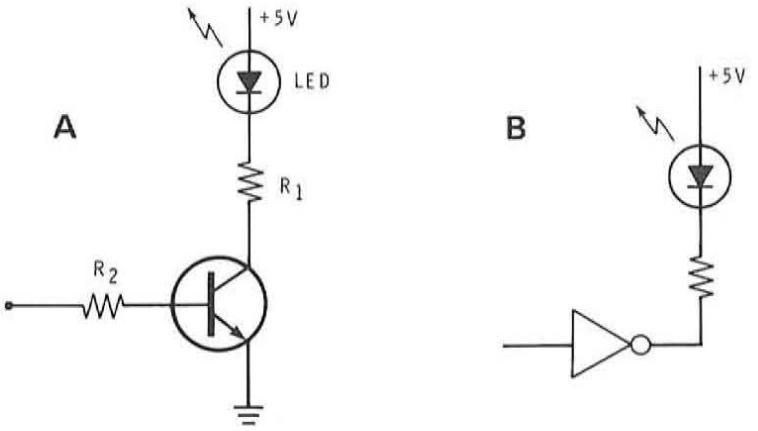


Figure 9-75
LED driver circuits (A) discrete component (B) TTL IC open collector inverter.

As a hint in starting you on this design, assume that the random nature of the circuit is derived from the use of a high speed clock oscillator. When the clock oscillator is enabled, it will step the logic circuit rapidly through the necessary states. The random depressing and release of the control push button for the clock oscillator will randomly determine when the clock starts and stops and what state the die circuit is in when it begins and when it ends. This will produce sufficiently random results for fair die operation. Using these guidelines develop the necessary circuit.

Answers

5. An analysis of this design problem shows that the circuit can be broken down into four basic parts. These are a six-state counter, a code converter, the die display and the clock circuit. These sections are shown properly interconnected in the simplified block diagram of Figure 9-76.

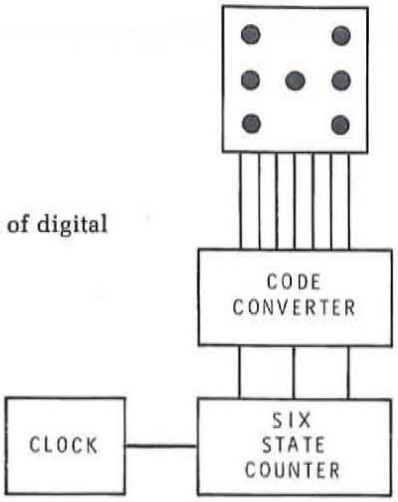


Figure 9-76
Simplified block diagram of digital die.

Since there are six possible die display states, a six state counter is required for the sequential circuit. A clock circuit is used to step the counter. The clock speed can be anything that is high enough to prevent the operator or user from selecting the desired outcome. If the clock is slow enough the user can observe the state changes and stop the counter at a desired state. Anything above approximately 50 Hz is satisfactory.

The six-state counter generates a specific binary code. This code can be almost any desired sequence of binary states. Three bits are required to represent the six states. With three bits a maximum of $2^3 = 8$ states will be produced. Two of these states will not be used or can be considered as "don't care" states.

The output of the counter drives a code converter. The code developed by the six-state counter is converted into proper logic output signals used to drive the LED indicators in the die display.

As you can see from Figure 9-76, this is a two part design problem. First you will need to design a six-state counter and then an appropriate code converter. The die display configuration has already been specified. The driver circuit for the LED was shown in Figure 9-75. The clock

circuit can be any astable or free-running multivibrator with some type of pushbutton switch used to start and stop it.

We are going to describe two possible solutions to the problem. Your solution may or may not be like either of these. The first solution to be presented follows the design procedures described in this unit. The second solution uses these procedures also but deviates somewhat in order to minimize the logic circuitry required for implementation.

The first part of the design procedure is to completely define the problem. As before this is best done by illustrating the inputs and outputs. Figure 9-77 shows the standard die format. These are the six discrete output states that we

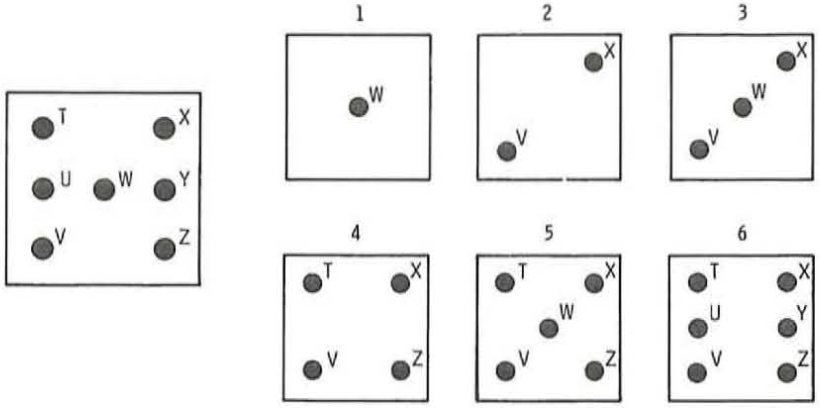


Figure 9-77
Standard die format.

wish to achieve with our circuit. Each spot on the die is implemented with an LED indicator. As you can see there are a total of seven outputs required by the circuit. These seven outputs or die segments are labeled T through Z. It is the purpose of our code converter circuit to convert the three-bit code from the counter into these seven outputs.

The truth table in Figure 9-78 shows the six die states, the counter states, and the die segment outputs. For this design we have chosen the standard binary code for the counter states. The standard binary counting sequence is generally easy to implement and therefore, it is chosen for this design. However, keep in mind that any sequence of code states could be used. Nothing in the design restricts us to one particular code.

DIE STATE	COUNTERSTATE A B C	DIE SEGMENTS T U V W X Y Z
1	0 0 0	0 0 0 1 0 0 0
2	0 0 1	0 0 1 0 1 0 0
3	0 1 0	0 0 1 1 1 0 0
4	0 1 1	1 0 1 0 1 0 1
5	1 0 0	1 0 1 1 1 0 1
6	1 0 1	1 1 1 0 1 1 1
	1 1 0 1 1 1 } DON'T CARE	

Figure 9-78
State and flow tables for digital die.

Associated with each counter state is the outputs for the die segments. A binary 1 in the T through Z columns indicates that the associated LED indicators will be on. Verify this truth table by referring to the die formats in Figure 9-77. This truth table in Figure 9-78 completely defines the design problem. Note that the 110 and 111 states for the counter are not used and therefore can be considered as "don't care" states.

The first part of our design then is to implement a six-state counter that steps in the standard binary code shown by the truth table. Using the procedure described in the text, we develop our design using Karnaugh maps as shown in Figure 9-79. Here a separate map for each flip-flop is shown. In each cell of the map is the appropriate symbols used to indicate the state change specified by the counter state table. Once all of the tables have been completely marked, the various cells can be grouped to specify the J and K inputs. From the marked Karnaugh map the input equations for each JK flip-flop are developed. The input equations for each are designated in Figure 9-79. The equations in Figure 9-79 can be implemented with JK flip-flops and SSI logic gates.

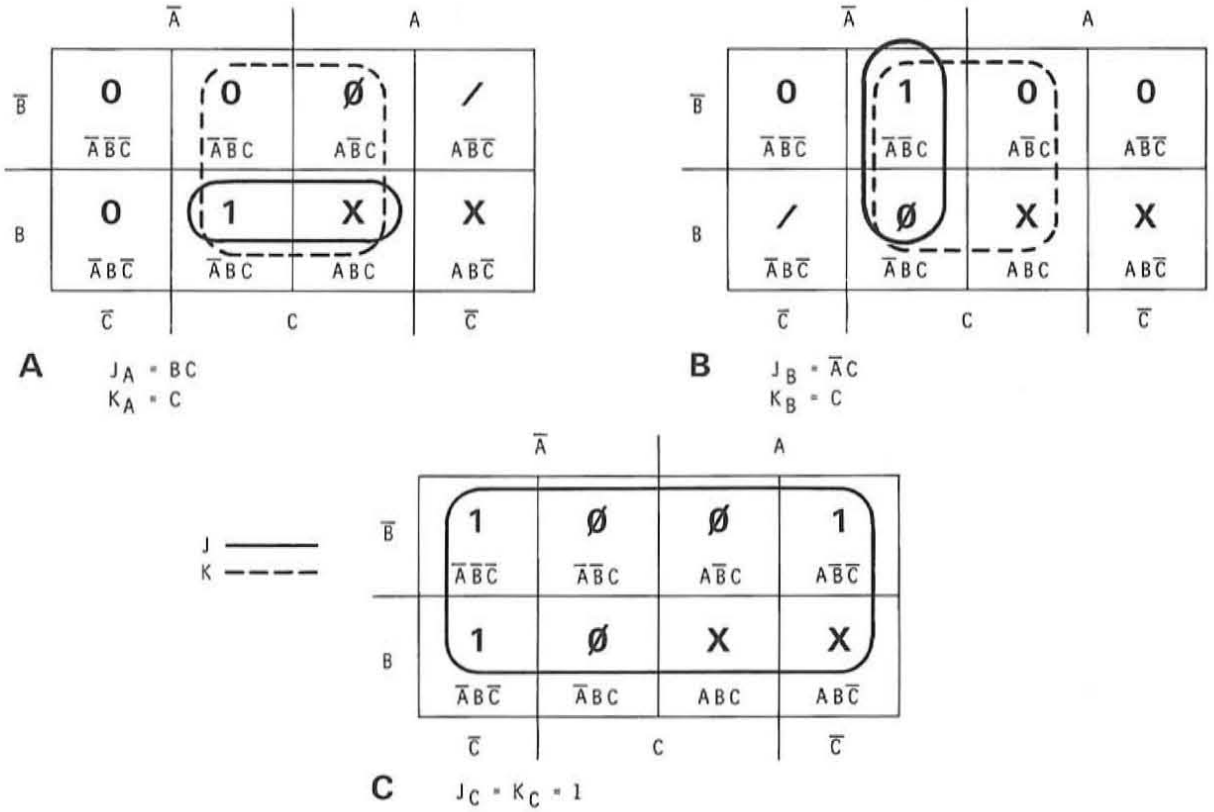


Figure 9-79
Karnaugh map for six-state counter.

The next step is to design a code converter circuit that will translate the six-state binary code into the output code specified by the die segments in Figure 9-78. To do this we effectively write the equation for each of the outputs T through Z and then implement it. However, before we do this it is desirable to study the truth tables to determine what simplifications they suggest. Studying the table we see that outputs T and Z are equal. Outputs U and Y are also equal. Outputs V and X are the same.Since these various outputs are equal, the output equations and the resulting circuitries are also the same. This means that the total number of output equations for this circuit then becomes four instead of seven. It is always a good idea to study the truth table of any design problem to see if such simplifications exist.

Instead of writing the output equation from the truth table we can go directly to Karnaugh maps for minimizations.

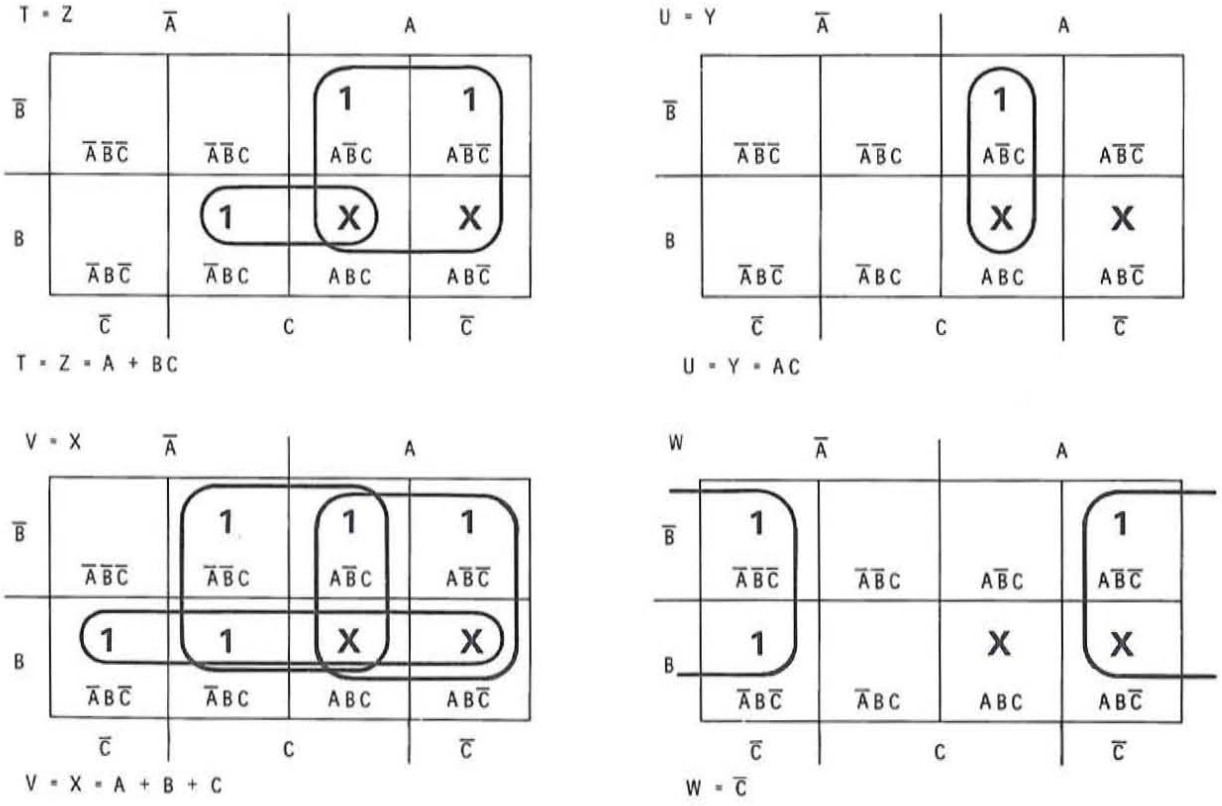


Figure 9—80
Karnaugh map for code converter.

These truth tables are shown in Figure 9-80. Note that the "don't care" states are marked with X's. Each of the output expressions is minimized and the minimum output equation for each die segment is given.

Figure 9-81 shows the complete logic diagram of the circuit. JK flip-flops A, B, and C are used to implement the 6-state counter. Gates 1 and 3 and inverters 2 and 4 are used to implement the logic inputs specified by the counter design. Gates 5, 6, 7, and 9 and inverter 8 are used to implement the logic equations for the code converter. The solid triangle represents the LED driver circuits shown in Figure 9-75.

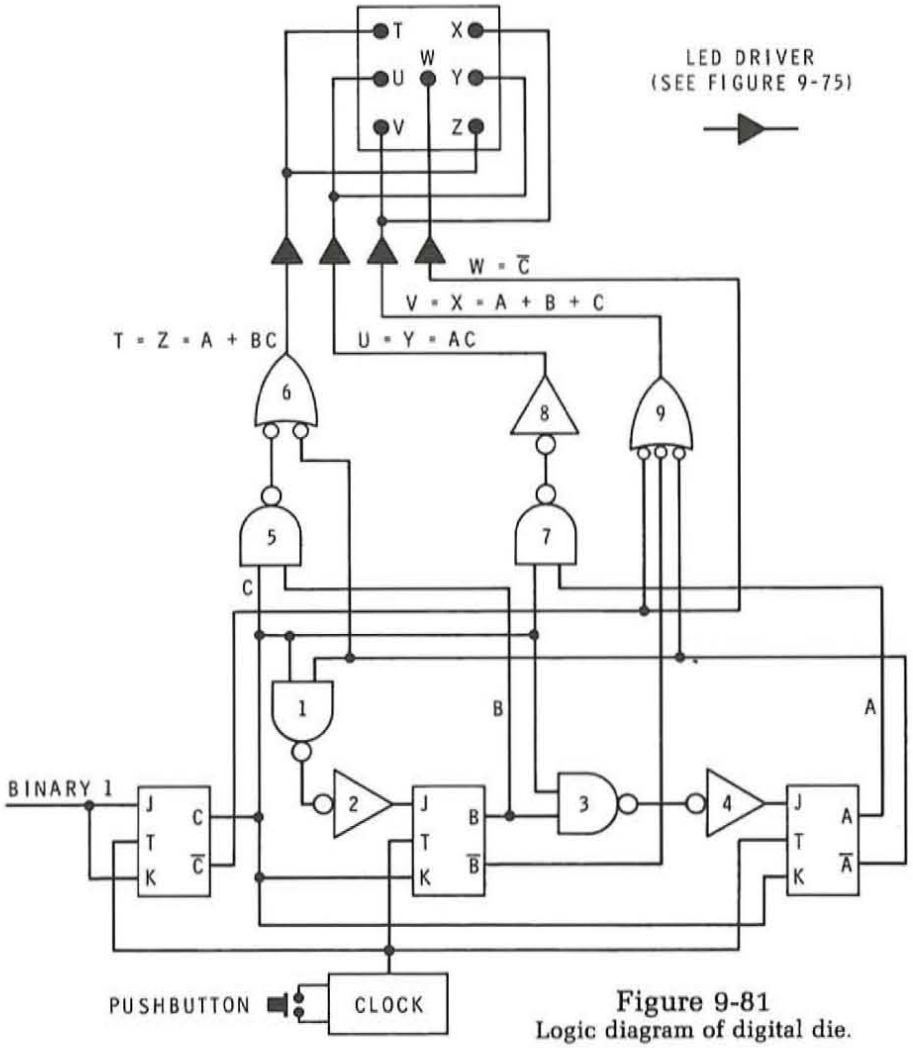


Figure 9-81
Logic diagram of digital die.

The design arrived at by the procedure outlined in the text is a workable design and uses a minimum number of components. However, no design procedure is perfect and there are many additional techniques and approaches that can be applied to further reduce the amount of circuitry required to implement the function. This will bring about a reduction

in the number of components, the cost, the power consumption and should increase reliability. Such further reductions in circuitry come about as a result of experience in working with digital logic circuits and in knowing the minimum forms of various types of circuits. Simplifications and reductions can also come about as a result of being familiar with the integrated circuits available from the various manufacturers. We can readily illustrate these two improvements on the circuits just designed.

A familiarity with the most commonly used counter and frequency divider circuit might lead you to develop the six-state counter circuit shown in Figure 9-82. Here, three

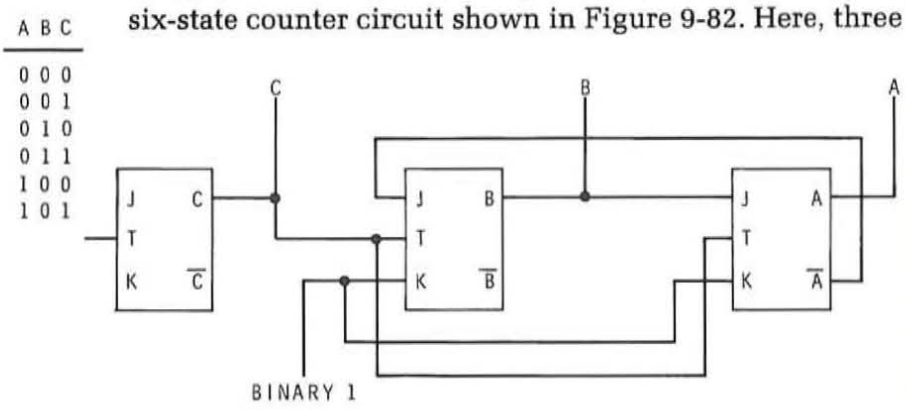


Figure 9-82
Simple six-state counter.

flip-flops are used but note that no external gating is required. The circuit counts in the standard binary code are identical to that used by the circuit in Figure 9-81. A close look at this circuit in Figure 9-82 shows that it is simply a binary divide-by-three circuit cascaded with an additional JK flip-flop to provide an additional divide-by-two function. Flip-flops A and B make up the count-by-three circuit. This is identical to the circuit discussed in Unit 7. This counter could be used to replace the counter circuit shown in Figure 9-81. It would eliminate gates 1 and 3 and inverters 2 and 4 providing a significant savings in size, cost and power consumption.

A familiarization with the various MSI integrated circuits available from the various manufacturers can also lead you

to an even simpler design. Figure 9-83 shows a logic diagram of a type 7492 integrated circuit IC. This circuit is designed as a 12-state counter or divide-by-12 frequency divider. A close look at the circuit shows that flip-flops B and C are connected as a count-by-three circuit similar to that shown in Figure 9-82. By connecting the output of the A flip-flop to the clock inputs of the B and C flip-flops, the circuit shown in Figure 9-82 is automatically available as a single MSI integrated circuit. Flip-flop D, although connected, is not used. By using the 7492, further reduction in cost and size can generally be obtained. These are only two examples of how design experience and component familiarization can lead to an efficient design.

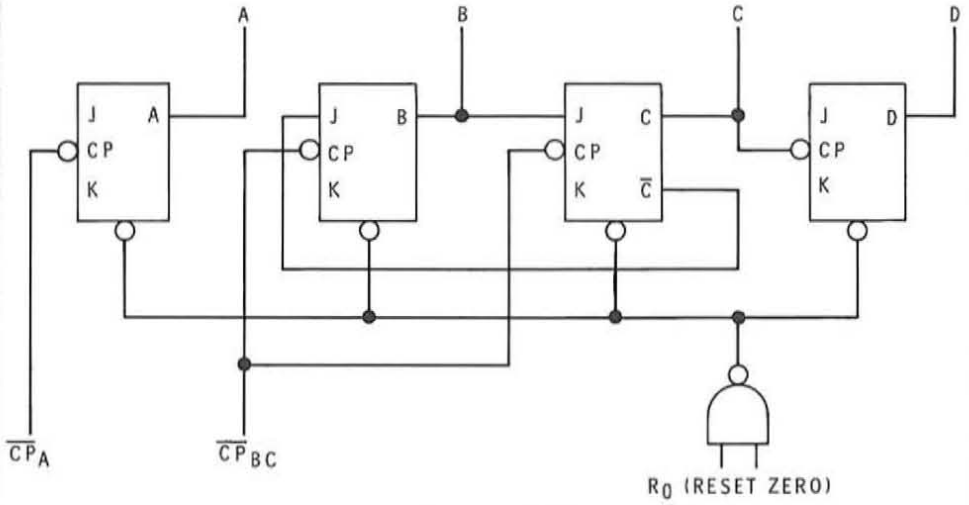


Figure 9-83
Type 7492 count-by-twelve MSI TTL IC.

EXPERIMENT 24

DESIGNING COMBINATIONAL LOGIC CIRCUITS

OBJECTIVES:

To construct and verify the operation of the sequential logic circuits designed in this unit and to provide practice in designing sequential logic circuits for specific applications.

Materials Required:

Heathkit Digital Design Experimenter ET-3200
All of the integrated circuits supplied with this program

TABLE IX

STATE	A	B	C
0			
1			
2			
3			
4			
5			
6			

Procedure:

1. Construct the seven-state counter circuit designed in the text and illustrated in Figure 9-63. From the integrated circuits available with this program, select those appropriate to the implementation of this circuit. Make the pin number assignments then wire the circuit and verify its operation. You can use the A logic switch as the clock input and the B logic switch as the reset input. Connect the A, B, and C outputs of the counter to the LED indicators. To test the counter, clear it to the 0 state with the B logic switch, then step the counter through its states and record your results in Table IX. Your result should correspond to the code shown in the table of Figure 9-57.

Discussion

To implement the seven-state counter in Figure 9-63, you should have used two 7476 dual JK flip-flops. Gates 1, 2, 3 and 4 in this circuit can be implemented with a type 7400 IC. Gates 5 and 6 can be implemented with

a type 7402 positive NOR IC. The assignment of pin numbers and gate connections is arbitrary and depends upon your own particular physical layout. Your output code should correspond to that given in Figure 9-57. If for some reason your counter does not work, the most likely cause is a wiring mistake. This could be because of improper use of a gate or flip-flop or by an actual wiring error. Double check all your connections. Several important items to check are to see that + 5 volts and ground are applied to each of the integrated cicuits you use. Second, the reset input line should remain high except when the circuit is being cleared to 0. This means that the clear input line should be connected to the $\overline{B}$ output of logic switch B. This line normally rests high but goes low when the switch is depressed.

Procedure

2. Construct the two-bit Gray Code counter shown in Figure 9-66. Select the proper type of IC and connect it making your own pin number assignments. Step the circuit with the A logic switch and monitor the outputs on two of the LED indicators. The count code sequence should be like that shown in Figure 9-64. Record your results in Table X.
3. Once you have verified the operation of the two-state Gray Code counter, modify your circuit so that it conforms to that shown in Figure 9-70. You can implement the AND gate using two sections of a type 7400 IC. One of the data switches can be used as the start input. Connect the clock input to the circuit to CLK and set the frequency to 1 Hz. It is also desirable to connect the direct clear inputs of the JK flip-flops to the $\overline{B}$ logic switch output so that you can conveniently reset the counter. To start the experiment, set the START input line to binary 0 and clear the counter to 0 state. Let the clock step the counter and note the effect on the A and B outputs. Next, set the START input line to binary 1 and note the effect. At some arbitrary point in the count sequence, switch the START input to binary 0. Note the effect on the A and B outputs.

TABLE X

STATE	A	B
0		
1		
2		
3		

Discussion

The two-bit Gray Code counter circuit is easily constructed using a single 7476 dual JK flip-flop. The clock input can come from the A or $\overline{A}$ output of the A logic switch. Output lines A and B can be monitored on any two

LED indicators. Code sequence should be like that shown in Figure 9-64 where only one bit at a time changes as clock pulses are applied. You should check the circuit several times stepping it through its cycle to monitor this effect.

With the control AND gate installed on the two-bit Gray Code counter, you should be able to control the start and stop operation of the counter. With the counter reset to 0 and the START input equal to binary 0, and clock pulses applied, no counter operation will take place. When the START line is switched to the binary 1 state, the counter will begin incrementing at a 1 Hz rate. The count sequence will continue as long as the start input line is high. When the START input line is switched to low, the counter will continue counting in its normal sequence until the 0 state is reached at which time it will stop and remain in this state until the START line is again brought high.

Procedure (Continued)

4. Construct the excess 3 BCD counter circuit shown in Figure 9-69. Again select the types of integrated circuits required and implement the circuit. As before, you can step the counter from the A logic switch. The outputs can be monitored on the LED indicators. Step the counter through its ten states and record your outputs in Table XI

TABLE XI

STATE	A	B	C	D
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				

NOTE: There is a possibility that the excess three counter can initially come up in one of the six invalid states. If this occurs the count sequence for this circuit will not be correct. Normally, if the counter is incremented six or more times, the circuit will generally recover and eventually be stepped into one of the ten valid excess 3 code states. When one of these valid code states is achieved, the counter will from that point on count in the correct sequence. Once you recognize one of the correct code states, simply increment the counter until it reads 0011, the initial 0 state for the excess three code. Starting at this point increment the counter with the A logic switch and complete Table XI.

Discussion

Your counter circuit should have sequenced through the ten excess 3 BCD code states as you applied clock pulses with the A logic switch. An incorrect result is probably due to a wiring error or the incorrect use of an IC. Be sure power is applied to all ICs. Check your wiring for mistakes if you have trouble.

Procedure (continued)

5. Using the integrated circuits supplied with this program, design a digital die circuit. You can use the 1 KHz clock to step the counter. By using a NAND gate and one of the logic switches, you can control the application of the clock pulses to the die counter circuit. For the output use the seven-segment LED indicator instead of the standard die display format discussed in the text. Use the 9368 (443-694) IC supplied with the circuit to drive the LED indicator. Select the codes so that the seven-segment display indicates the numbers 1 through 6. Design an appropriate six-state counter using the integrated circuits supplied with this program. Also, design the necessary code converter combinational logic circuit to convert the counter code into the appropriate code necessary to drive the BCD to seven-segment decoder driver circuit. After you design the circuit, draw a complete logic diagram. Assign pin numbers to the various integrated circuits, then build your circuit and test it.

Discussion

As with most applications there are many possible solutions. There are a variety of ways to implement the digital die function as defined by the problem. However, a close evaluation of the various alternatives will generally lead to only one or two designs which can be considered the most efficient. You should have found this to be the case with this problem.

The seven-segment LED will display the die numbers 1 through 6. The LED will be driven by the 9368 decoder driver IC. This IC accepts the standard 8421 BCD input code. Therefore, it is obvious that we wish our sequential circuit to generate the code necessary to sequence through the states that will display the decimal numbers 1 through 6. For this problem the combinational logic circuitry is automatically taken care of by the 9368 decoder driver circuit. No further design is necessary. However, you must define the input code required to cause the display to indicate the numbers 1 through 6. These are indicated in Table XII. The standard 8421

TABLE XII

DECIMAL OUTPUT	DECODER/DRIVER INPUTS		
	A	B	C
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
	0	0	0
	1	1	1

"don't care"

BCD codes for each desired decimal output is required. The most significant bit for the decimal outputs 1 through 6 will be binary 0 and therefore it has been omitted. The most significant input to the 9368 decoder-driver can simply be grounded. The three least significant bits will be used. The desired input code is a three-bit word with the standard binary number equivalents for the decimal outputs as indicated by the Table. The 000 and 111 states are not required and therefore, can be treated as "don't care" conditions.

The code sequence required by the application defines the sequential logic circuit operation. We must design a six-state three bit counter that will count in the standard binary sequence 1 (001) through 6 (110) and then recycle.

With the information in the state table, we develop the Karnaugh maps for each flip-flop in the counter. These maps are shown in Figure 9-84. The symbols marked in each cell designate the state change that must

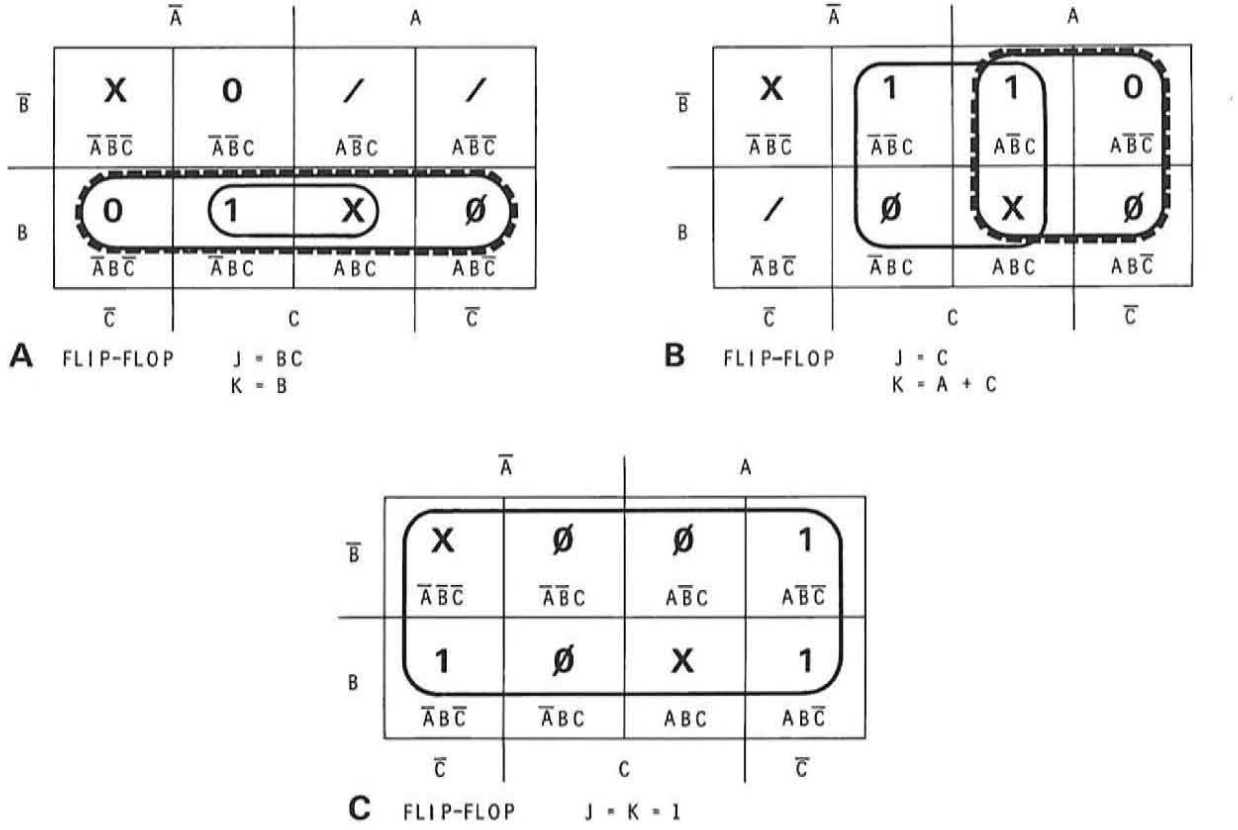


Figure 9-84
Karnaugh maps for deriving input equations for flip-flops in six-state digital die counter.

occur to transfer to the next state in sequence. These various symbols are then grouped according to the rules given earlier to derive the J and K input equations for the flip-flops in the counter. The completed counter circuit is shown in Figure 9-85. This counter is readily implemented with two dual JK flip-flops type 7476. A type 7400 IC can be used to implement the gating functions. Gate 1 is used to switch the clock off and on with logic switch A. Gates 2, 3, and 4 are used to implement the logic equation expressed by the Karnaugh maps. Gate 4 is simply connected as an inverter.

To operate this circuit, you simply depress the A logic switch for an arbitrary length of time. During this time the 1 KHz clock will increment the counter rapidly. All seven segments of the display will appear to be lighted during this period of time. When you release the A logic switch, the counter will stop at one of the six states. The arbitrary starting and stopping of the counter will produce a random result.

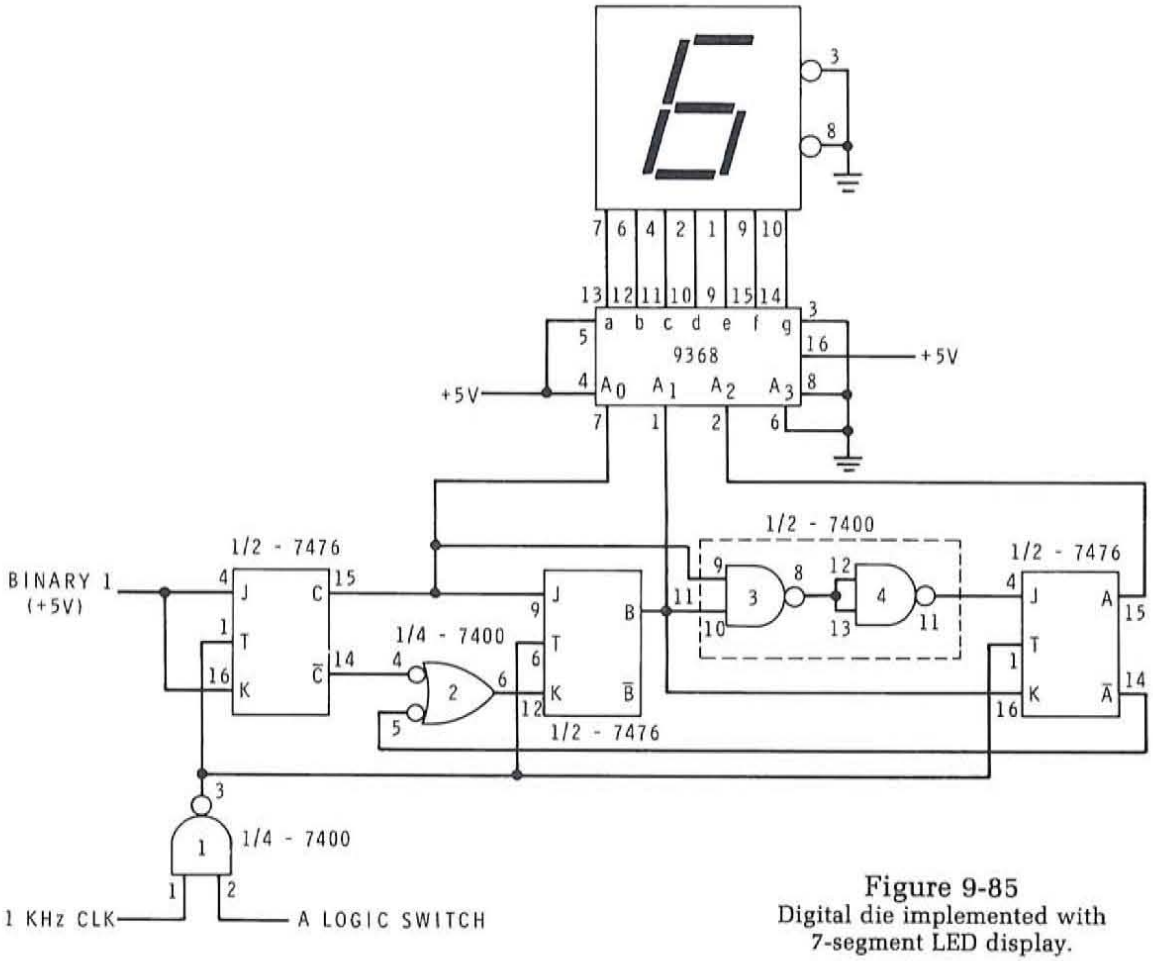


Figure 9-85
Digital die implemented with 7-segment LED display.

EXAMINATION

UNIT 9

DIGITAL DESIGN

The purpose of this exam is to help you review the key facts in this unit. The problems are designed to test your retention and understanding by making you apply what you have learned. This exam is not so much a test as it is another learning method. Be fair to yourself and work every problem first before checking the answers.

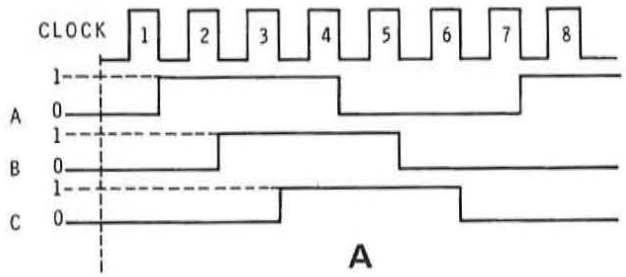
1. Design an invalid code detector circuit for the excess 3 BCD code. Use the design procedure for combinational circuits given in the text showing all truth tables and maps. Select the most efficient method of implementation with integrated circuits and draw the final circuit diagram. Construct your circuit on your Digital Design Experimenter using the ICs supplied with the program. Verify the operation of the circuit.

2. Assume that you wish to use a standard six-state binary counter such as the circuit shown in Figure 9-82 to implement a digital die. Assume further that you want to use the 7-segment LED display and 9368 decoder-driver IC. Design a code converter circuit that will make the counter and display compatible. Follow the design procedure in the text and draw a complete logic diagram of your circuit. Construct and test your design.

3. Design a 6-state counter that will generate the timing waveforms shown in Figure 9-86A. Use three JK flip-flops and assume that the waveforms come directly from the normal flip-flop outputs. Develop the state table from the waveforms using the table provided in Figure 9-86B. Then use the design procedure for sequential circuits described in the text. Draw your final logic diagram. Construct and verify the operation on your circuit.

A	B	C

B



A

Figure 9-86
Timing waveforms for Exam Problem 3 (A) and a state table to be used in developing the code sequence from the waveforms (B).

ANSWERS

UNIT 9 EXAMINATION

DIGITAL DESIGN

1. The first step in designing an excess 3 BCD invalid code detector is to develop the truth table. The excess 3 BCD code was described in a previous unit and is a four-bit code whose decimal equivalent value is three greater than the standard binary equivalent value for the related decimal number. The decimal numbers 0 through 9, therefore, are represented by the binary numbers 0011 through 1100. The BCD excess 3 code is shown in Figure 9-87. As you can see from this table there are six invalid states. These invalid four-bit codes have no meaning in the excess 3 system and therefore, in some applications it is desirable to detect these invalid codes. The circuit you are designing will detect these codes and generate a binary 1 output if any of the six should occur. The circuit required is a combinational logic circuit with a single output F and inputs A, B, C and D.

DECIMAL	INPUTS	OUTPUT
(* INVALID)	A B C D	F
*	0 0 0 0	1
*	0 0 0 1	1
*	0 0 1 0	1
0	0 0 1 1	0
1	0 1 0 0	0
2	0 1 0 1	0
3	0 1 1 0	0
4	0 1 1 1	0
5	1 0 0 0	0
6	1 0 0 1	0
7	1 0 1 0	0
8	1 0 1 1	0
9	1 1 0 0	0
*	1 1 0 1	1
*	1 1 1 0	1
*	1 1 1 1	1

Figure 9-87
Excess 3 BCD invalid code detector truth table.

Rather than write the Boolean equations of the output expression F, we can transfer the truth table information directly to a sixteen-cell Karnaugh map. This map is shown in Figure 9-88. The designated cells are grouped to minimize the logic function. The minimum Boolean equation for this function is given adjacent to the map.

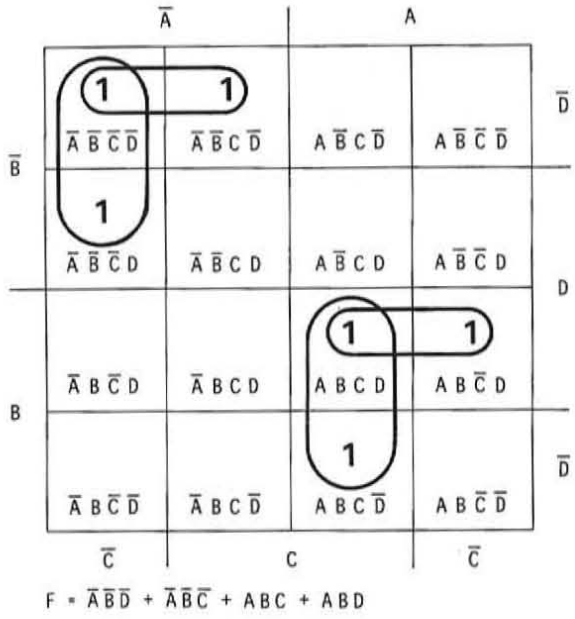


Figure 9-88
Karnaugh map for minimizing the XS3 BCD invalid code detector circuit.

SSI logic gates can be used to implement the minimum equation. However, it will require four three-input AND gates and a four-input OR gate. From what you have learned in this unit you should have determined that the most economical method of implementing this circuit is to use a data selector. An eight-input data selector such as the 74151 can be used to implement the function. The three least significant bits of the input code (B, C, and D) will be used to drive the A, B, and C inputs of the data selector. The decimal equivalents of the three least significant bits in Figure 9-87 define the data selector inputs which will be used. These correspond to D0, D1, D2, D5, D6, and D7. Inputs D3 and D4 are not required, therefore they are connected to 0. The A input from the excess 3 code signal or its complement will applied to the D0, D1, D2, D5, D6, and D7 inputs as required. When the A input is binary 1, we want the D5, D6, and D7 inputs enabled. Therefore, we will apply the A input directly to these three input lines on the data selector. The complement A will be applied to the D0, D1, D2 inputs through an inverter so that these

terms will be enabled when A is binary 0. The resulting circuit is shown in Figure 9-89. While your own design procedure may have lead to a slightly different result, the circuit shown here is the most efficient method of implementing this function.

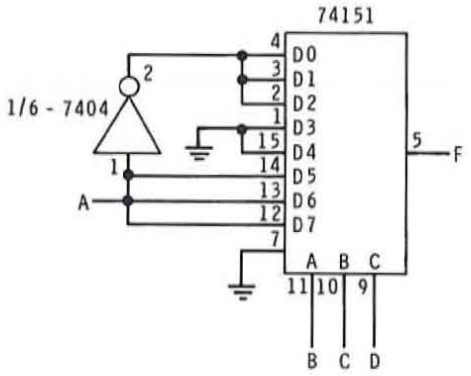


Figure 9-89
Excess 3 BCD invalid code detector implemented with a 74151 TTL data selector.

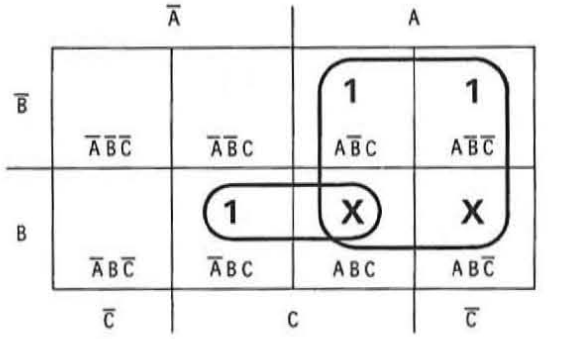
2. Your design goal in this problem is to make the output of a standard binary counter compatible with the input to a BCD to seven-segment decoder-driver for the purpose of implementing a digital die. The digital die will have six states and will display the numbers 1 through 6. The problem specified that a standard binary counter such as the one shown in Figure 9-82 would be used. This counter counts in the standard weighted binary code of 000 through 101. The outputs of this counter will be the input to your code converter.

INPUTS	OUTPUTS
A B C	D E F
0 0 0	0 0 1
0 0 1	0 1 0
0 1 0	0 1 1
0 1 1	1 0 0
1 0 0	1 0 1
1 0 1	1 1 0

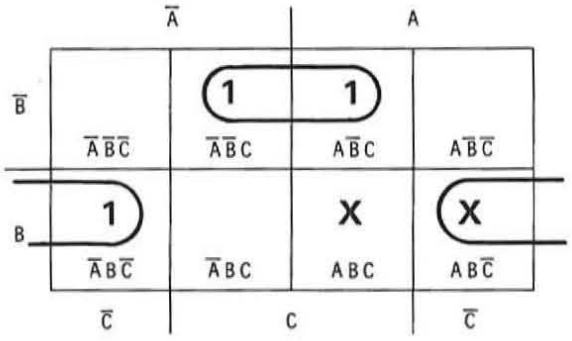
Figure 9-90
Truth table for die code converter.

In order to display the digits 1 through 6, the standard 8421 BCD code for these decimal numbers must be applied to the 9868 BCD to seven-segment decoder-driver circuit. Since we are only using the numbers 1 through 6, only the three least significant bits of the standard 8421 BCD code are valid. The most significant bit can be binary 0. Therefore, this pin on the decoder driver is simply grounded. The three least significant bits of the BCD codes for the numbers 1 through 6 are the required outputs for the code converter. With this analysis you have effectively defined your design problem. It can be summed up by the truth tables shown in Figure 9-90. The inputs can be labeled A, B, and C and represent the counter output. The code converter outputs can be labeled D, E, and F and are as indicated to implement the six states of the digital die.

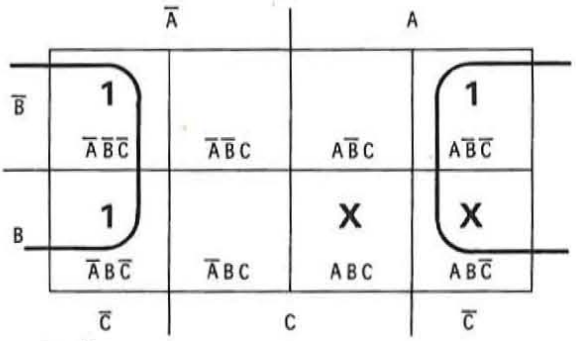
We have constructed a Karnaugh map for each of the outputs in Figure 9-90. The Karnaugh maps for the code converter circuit are shown in Figure 9-91. Note that the "don't care" states are marked and are used to reduce the equations to their minimum form. The minimum equations for each output are given with the map. Check to be sure that you see how the information in the truth table corresponds to the data in the Karnaugh maps.



$D = A + BC$ **A**



$E = B\overline{C} + \overline{B}C$ **B**



$F = \overline{C}$ **C**

Figure 9-91
Karnaugh maps for digital die code converter.

The minimum equations for this circuit can be readily implemented with SSI logic circuits. Two type 7400 integrated circuits containing two input NAND gates can be used. The completed logic diagram is shown in Figure 9-92. One of the gates in the 7400 IC can be used as an input control for the clock to the counter. When the A logic switch is depressed, the 1 KHz clock will be applied to the counter and will step it rapidly. Releasing the switch will stop the counter at some arbitrary state and display one of the numbers 1 through 6.

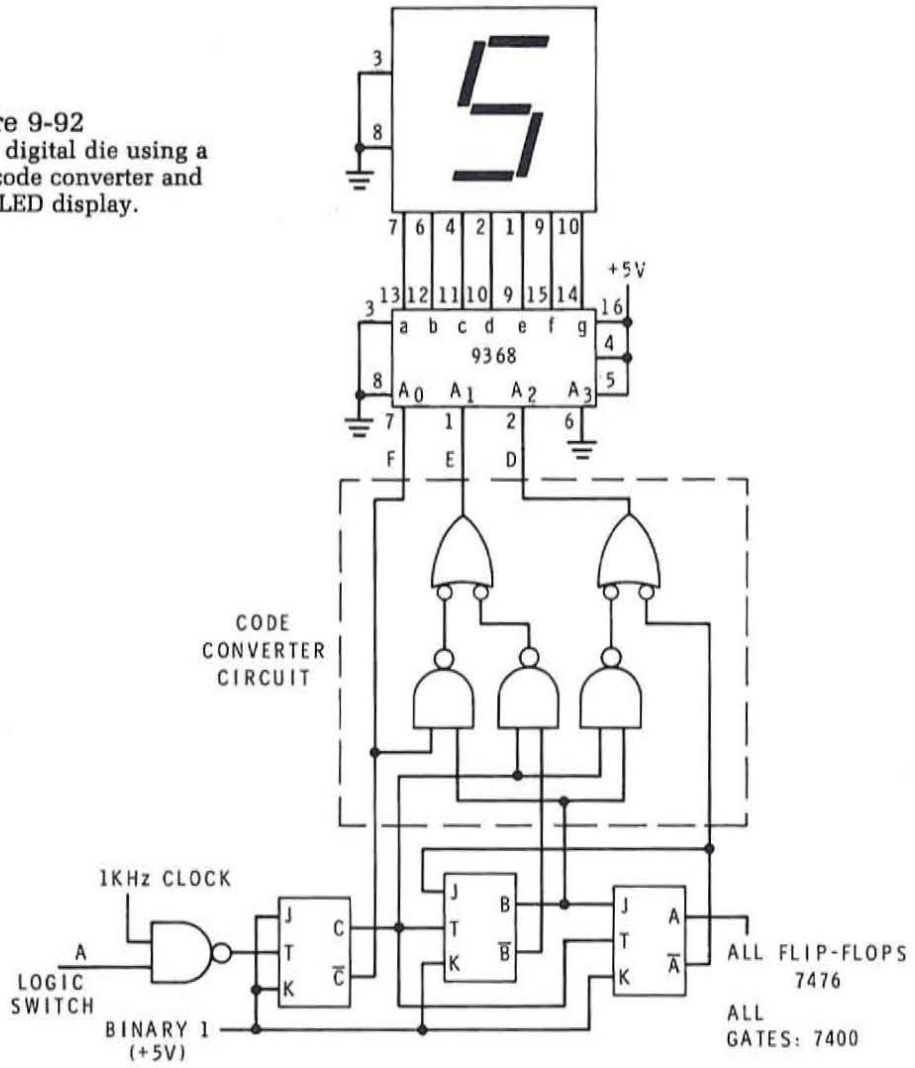


Figure 9-92
Logic diagram of digital die using a 6-state counter, code converter and 7-segment LED display.

3. In this problem the application begins with a set of timing waveforms that fulfill some specific function. The design job is to implement a logic circuit that will produce this sequence of timing signals. A sequential circuit is required to fulfill this application. The waveforms are studied and from them a flow table is produced. As

specified by the problem, each of the waveforms represents the output of a flip-flop in the circuit. A six-state sequential circuit will result. You can see this by studying the waveforms and putting binary 1s and 0s on them as shown in Figure 9-93A. The initial state

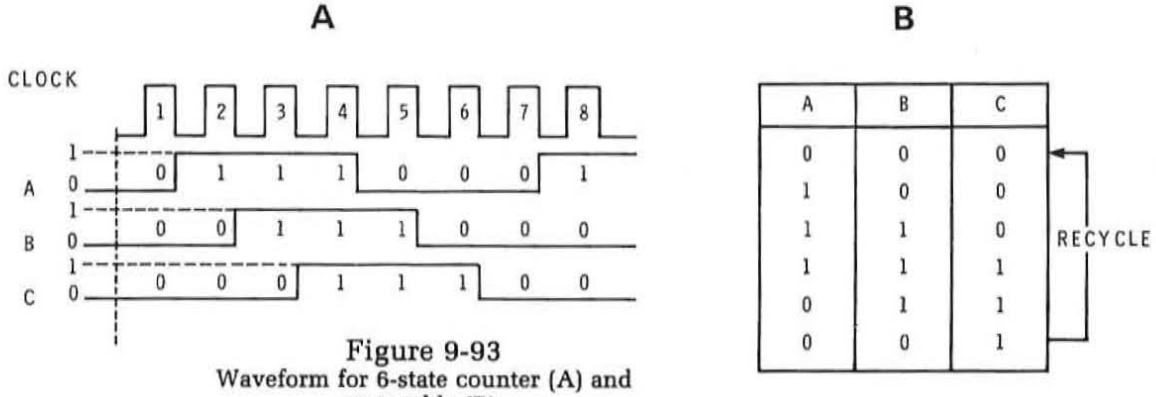


Figure 9-93
Waveform for 6-state counter (A) and state table (B).

of the counter is 000 as you can see by the waveforms. After the application of six-input pulses, the sequential circuit recycles to the 000 state. The various states of the flip-flops can be transferred to the state table as shown in Figure 9-93B. From this table you can develop the Karnaugh maps that will indicate the J and K inputs states for each flip-flop in order to generate this code. These maps are shown in Figure 9-94. Study the state table and relate it to the Karnaugh maps.

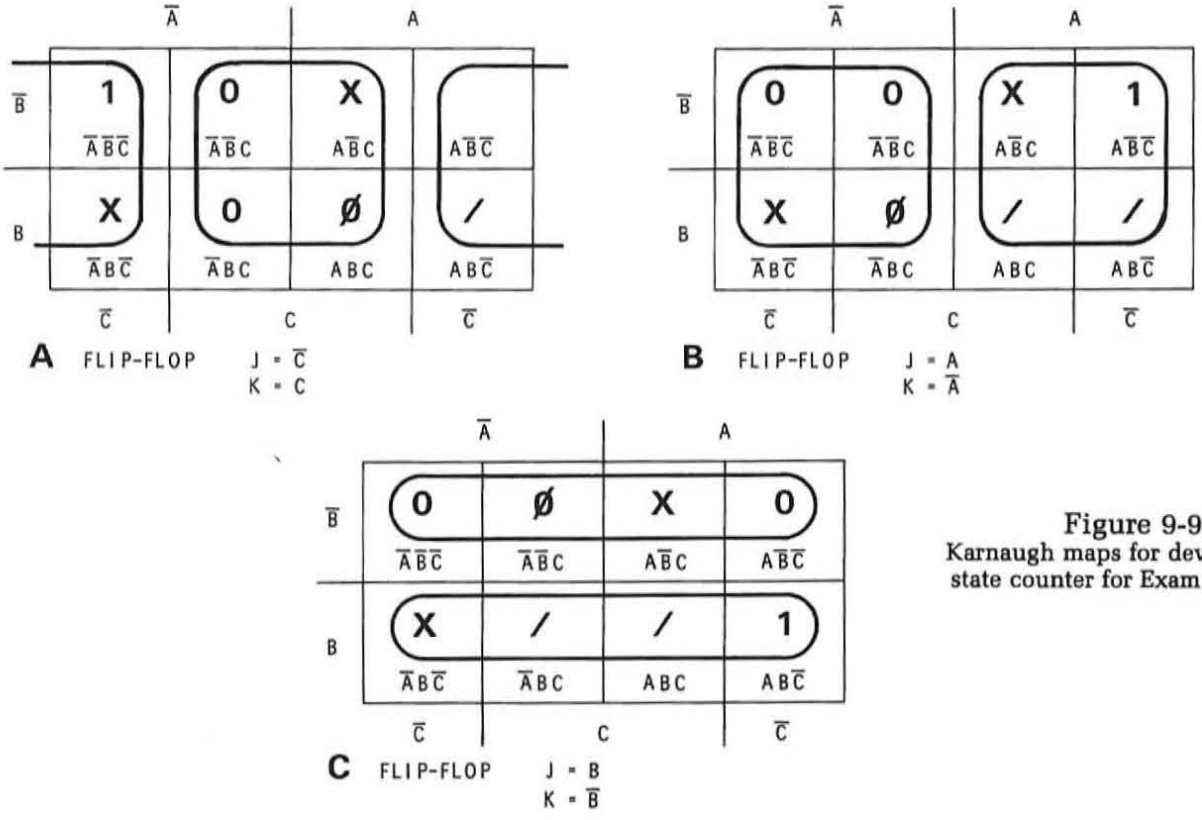


Figure 9-94
Karnaugh maps for developing 6—state counter for Exam question 3.

The various cells in the map are grouped according to the rules given in the design procedure. The J and K input states are then determined. These can then be translated directly into a logic diagram. The circuit is shown in Figure 9-95. If you study this circuit carefully, you will see the six-state counter is really a shift register. It is a three flip-flop six-state switch tail ring shift register similar to that discussed in a previous unit. This problem demonstrates the fact that the design procedure described in the text for sequential circuits often results in shift register circuits as well as special counters. The circuit can be readily implemented with 7476 JK flip-flops.

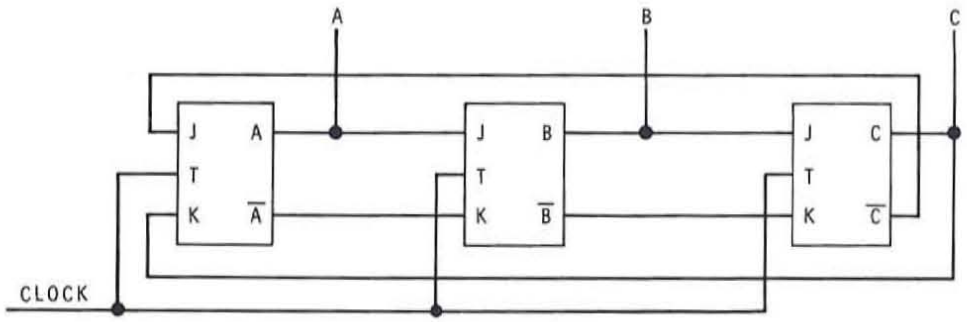


Figure 9-95
Six-state counter or switch tail ring shift register.

HEATHKIT
CONTINUING
EDUCATION

Individual Learning Program

In

DIGITAL TECHNIQUES

703-8

Heathkit
ZENITH Educational Systems

UNIT 10

DIGITAL APPLICATIONS

CONTENTS

INTRODUCTION

In this final unit of your individual learning program in digital techniques, you are going to study some typical applications for digital techniques. Throughout the program we have emphasized component operation and circuit details, but in this unit we will put all of this information together and show you examples of how digital circuitry performs useful functions.

Perhaps the greatest application for digital techniques is in computers. The development of computers and digital circuits has been a parallel effort, each area being a benefit to the other. Another area of electronics affected by digital technques has been test/measurement equipment. Most of the high quality, precision test equipment available today is based on digital techniques.

It is impossible to cover the complete spectrum of digital applications in a unit this size. For that reason we are selecting what we feel are two of the most important applications for digital techniques. These are test and measuring equipment and computers. In this unit we will give you typical examples of each. These examples clearly illustrate the power and benefits of digital techniques.

The unit objectives state specifically the things that you will learn in this unit. Review them now, then refer to the unit activity guide.

UNIT OBJECTIVES

When you complete this unit, you will be able to:

1. Name the largest application area for digital techniques in electronics.
2. List three benefits of the use of digital techniques in test and measurement applications.
3. List two test instruments that use digital techniques.
4. Draw a block diagram of a digital counter and explain the operation of each major section.
5. Name the three basic operating modes of a digital counter and explain how each works.
6. Explain how a digital computer operates.
7. Given a computer instruction set, analyze a simple program.
8. Define the terms minicomputer, microcomputer, and microprocessor.
9. Write a simple program, given the problem and the instruction set.
10. Given a digital design problem, determine the applicability of a microprocessor.
11. List the four primary benefits of using a microprocessor to replace hard-wired logic systems.

UNIT ACTIVITY GUIDE

		Completion Time
☐	Play Audio Record 7, Side 2	_____
☐	Read Section: Digital Test Equipment: The Frequency Counter	_____
☐	Answer Self Test Review Questions 1-7	_____
☐	Read Sections: Modes of Operation and Counter Specifications	_____
☐	Answer Self Test Review Questions 8-19	_____
☐	Read Section: A Typical Digital Counter	_____
☐	Answer Self Test Review Questions 20-27	_____
☐	Read Sections: What is a Digital Computer? and How Computers Are Classified	_____
☐	Answer Self Test Review Questions 28-31	_____
☐	Read Sections: Digital Computer Organization and Digital Computer Operation	_____
☐	Answer Self Test Review Questions 32-40	_____
☐	Read Sections: Programming, Writing Programs and Example Programs	_____
☐	Answer Self Test Review Questions 41-47	_____
☐	Read Section: Software	_____
☐	Answer Self Test Review Questions 48-51	_____
☐	Read Section: Microprocessors	_____
☐	Answer Self Test Review Questions 52-57	_____
☐	Complete Unit Examination	_____

DIGITAL TEST EQUIPMENT: The Frequency Counter

Digital test equipment has greatly improved the resolution and accuracy of electronic measurements. At the same time, digital techniques have made them faster and more convenient. The quantity being measured is displayed directly with numerical digits, thereby eliminating the interpolation associated with meter-type analog instruments.

While digital instruments have been designed to measure almost every electronic quantity, there are several quantities which are more commonly measured than others. These are voltage, current, resistance and frequency. Two basic types of digital test instruments have been developed to measure these quantities. These are the digital multimeter (DMM) and the frequency counter.

A digital multimeter is an electronic test instrument that is used for measuring voltage, current, and resistance. The most commonly measured quantity is voltage. A digital instrument used to measure voltage is referred to as digital voltmeter (DVM). A digital multimeter is used like any analog multimeter in that the test leads are connected to the circuit under test. The quantity being measured is displayed on a decimal readout instead of being indicated by the position of a pointer on a meter scale. Both the resolution and accuracy of the digital display is better than that associated with even the highest quality analog test instruments.

Figure 10-1 shows a low cost digital multimeter for measuring current, voltage and resistance. Special single function digital meters are also available to replace standard analog panel meters for either current or voltage measurements. These are referred to as digital panel meters (DPM). Any digital multimeter is essentially an analog-to-digital converter. The meter converts analog quantities of voltage, current or resistance into an equivalent BCD word so that the quantity can be displayed in decimal form. The analog-to-digital conversion technique employed in digital multimeters can also be used in measuring other electronic quantities such as capacitance, inductance, reactance, impedance, power and others.

Figure 10-1
Typical digital multimeter for measuring voltage, current and resistance. (Heathkit IM-1212)

The other widely used digital test instrument is the frequency counter. This instrument is designed primarily to measure the frequency of a periodic input signal. It displays the frequency in Hz, KHz, or MHz in decimal form. A frequency counter literally counts the number of pulses or cycles of an input signal that occurs in a known period of time and displays the frequency directly. Figure 10-2 shows a typical frequency counter.

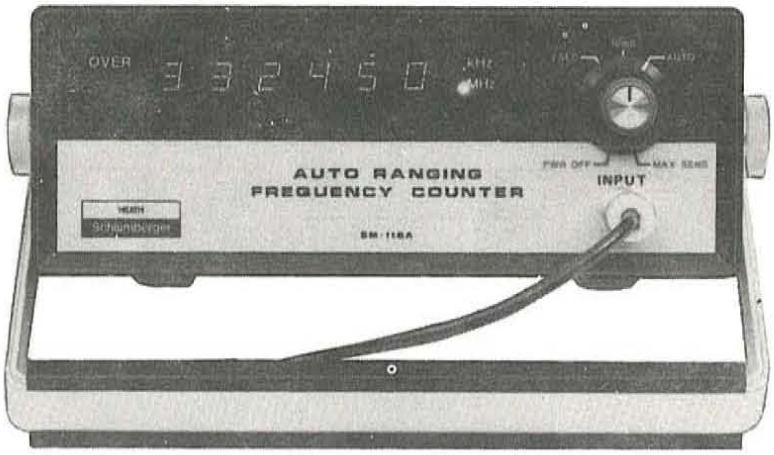


Figure 10-2
Typical digital frequency counter.
(Heath/Schlumberger SM-118A)

While frequency measurement is the basic function of a digital counter, many counters can make other time related measurements. Some general purpose counters can also measure the period of an input signal. Frequency counters can also be used for totalizing operations where the counter functions to keep tally of input events that occur. Some counters also permit frequency ratio measurements where two input frequencies can be compared and their ratio displayed. Time interval measurements can also be made with some counters.

The digital counter is one of the most versatile electronic test instruments available. Time and frequency measurements are vital to the proper testing and evaluation of electronic circuits and equipment. The digital counter has greatly improved the resolution and accuracy of frequency and time measurements, and at the same time has made them faster and more convenient.

The digital counter is an excellent example of the use of digital circuitry to perform a useful measurement function. For that reason, we're going to analyze the operation of a typical digital counter. Many of the digital circuits described in this program are used to implement the various counting functions.

A general block diagram of a digital counter is shown in Figure 10-3. It consists of four main sections; the input circuit, the gate and control circuits, the time base, and the decimal counter and display. These circuits work together in various ways to provide measurement of frequency or time interval as determined by the application. Let's take a detailed look at each of these major sections.

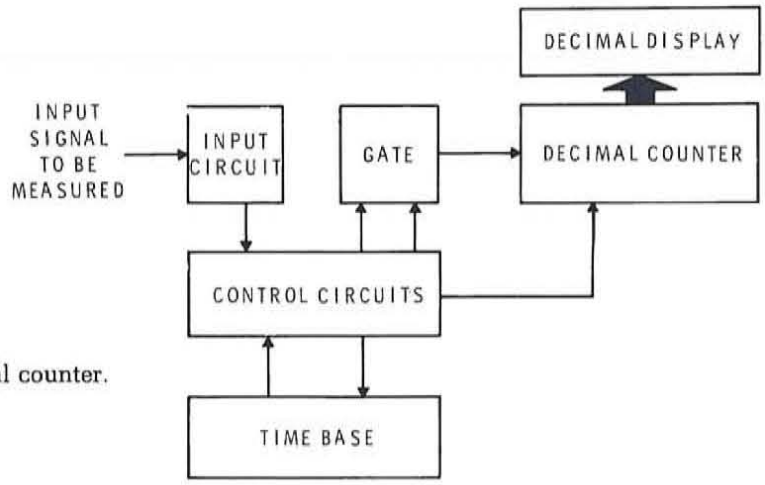


Figure 10-3
Block diagram of a digital counter.

Input Circuit

The input circuit is a signal conditioner designed to make the signal to be measured compatable with the digital circuitry in the counter. The input circuit must be flexible enough to permit the counter to accept signals whose amplitude and wave shape vary considerably.

The input circuit incorporates some form of amplification and isolation. The isolation is obtained with a high input impedance follower stage to minimize the loading on the circuit generating the input signal. Amplification is used to improve the sensitivity of the counter so that very low amplitude signals can be measured.

The input circuitry also incorporates an attenuator and protection circuitry for handling large amplitude signals. A resistive attenuator is used to reduce high amplitude signals to a level compatable with the input circuitry. Diodes are also incorporated to clamp or limit the input voltage. This protects the circuitry from excessively high amplitude signals.

Perhaps the most important part of the input circuitry is the wave shaper. This is a circuit that converts the input signal into a rectangular pulse whose output logic levels are compatable with those of the rest of the counter circuitry. This circuit is used to shape and square sine wave and other non-rectangular input signals. A typical circuit used for such squaring operations is the Schmitt trigger. The Schmitt trigger is a special form

of bistable multivibrator circuit. The binary output state is a function of the input amplitude. When the input is below a certain threshold level the output is binary 0. When the input exceeds this threshold level the output switches to binary 1. If the input voltage then drops below another threshold level, the output switches back to binary 0. Schmitt trigger circuits are available in TTL, ECL and CMOS form. The output of the Schmitt trigger is a signal whose frequency is identical to that of the input signal but whose shape and amplitude are compatable with the remaining counter circuitry. The output of the input circuit is applied to the control circuitry which then determines how that input signal is used in the measurement process.

Gate and Control Circuits

The gate is nothing more than a standard AND circuit that controls the application of the input pulses or a precision timing signal to the decimal counter. The control circuit determines how the input circuitry and precision timing signals from the time base are used to control the gate. The control circuits are also used to operate the decimal counter. The control circuits generate all of the pulses required to permit the counter to operate in a variety of modes.

Time Base

The time base consists of a precision crystal oscillator and a series of frequency dividers that generate highly accurate and stable timing signals. These signals are used as the reference or standard in making the frequency and time measurements. The crystal oscillator which usually runs at 1 MHz or 10 MHz provides this standard. BCD counters used as frequency dividers generate decade sub-multiples of the crystal oscillator frequency. The precision time signals generated by the time base are then used to control the gate for frequency measurements or are counted by the counter in period and time interval measurements.

The quality and accuracy of the time and frequency measurements made by the counter is a direct function of the accuracy and stability of the crystal oscillator. Most good counters use a temperature compensated crystal oscillator (TCXO) to insure that the frequency remains stable over a wide range of ambient temperature variations. In higher quality counters the crystal oscillator itself is contained within a small oven or temperature controlled chamber where the temperature is maintained at a constant level.

In lower quality, less accurate counters, other signal sources can be used as a time base. For example, the ac power line voltage can be used as a

reasonably accurate time and frequency standard. The frequency of most ac power line signals is controlled to better than .1 percent. The normal 60 Hz power line signal can be divided down to produce timing signals of 100 ms, 1 second, and 10 seconds.

Decimal Counter and Display

The heart of the digital counter is the decimal counter circuit and its display. The decimal counter consists of a number of cascaded BCD counter stages. The counter counts the pulses received from the gate. These pulses can be the shaped input signal or a reference signal from the time base. The decimal counter accumulates these pulses and stores them as a multiple digit BCD word. The output of each BCD counter is fed to a storage register where the BCD word can be stored. The output of these registers are then used to drive BCD-to-decimal decoder-driver circuits. These operate the display readout elements. Seven-segment light emitting diodes and gas discharge readouts are the most commonly used displays in digital counters.

The operation of the decimal counter is determined by the control circuits. The control circuit generates signals to reset the counter and to transfer the contents of the BCD counters into the storage registers.

Self Test Review

1. List three benefits of digital techniques to electronic measuring instruments.
   a. _____
   b. _____
   c. _____
2. The digital test instrument used to measure resistance is called a _____.
3. A digital counter is most often used to measure _____.
4. The four basic digital counter sections are:
   a. _____
   b. _____ c. _____ d. _____
5. The circuit used to "square" the input signal is called a(n) _____ _____.
6. The _____ circuit controls the application of input pulses to the counter.
7. What circuit is used as the time standard in most digital counters?
   a. decoder driver
   b. BCD counter (decade divider)
   c. Schmitt trigger
   d. crystal oscillator

Answers

1. a. accuracy
   b. resolution
   c. convenience
2. digital multimeter
3. frequency
4. a. input
   b. gate/control
   c. time base
   d. decimal counter
5. Schmitt trigger
6. gate
7. d. crystal oscillator

Modes of Operation

1 sec → 1 hz
1 ms → 1 Khz
1 μs → 1 Mhz

Digital counters are capable of operating in a variety of modes, each permitting a different type of time and frequency measurement. The most commonly used mode of operation is frequency measurement. Many counters are wired to perform only this function. This is true of the counter shown in Figure 10-2. Other more flexible counters permit a variety of time and frequency measurements to be made. Such counters are widely used in laboratory testing and development applications. A front panel switch selects the desired mode of operation. In this section, we discuss each of the most commonly used digital counter modes.

Totalized Mode. The simplest form of operation for a digital counter is the totalized mode. This is an events counting mode where pulses appearing at the input are counted with the sum being accumulated in the counter and displayed. Figure 10-4 shows the interconnections in the counter for the totalize mode. The electrical events to be counted are applied to the input circuit. The output of the input circuit is connected to the count input of the gate. A front panel pushbutton is used to reset the counter and clear the display to zero. As the input signals occur they will be counted and the total displayed.

In some counters the count control input to the gate can be enabled or inhibited by an external gating pulse. The totalized mode provides a simple means of counting or keeping tally of the input events that occur. In this mode the time base section of the counter is not used.

An application example of the counter used in the totalized mode will more clearly illustrate its operation. Consider the situation in a manufacturing company where it is desired to count the number of items transported along a conveyor belt. As the products come off a production line they are placed on to a conveyor belt approximately one foot apart. At one point in their trip down the conveyor belt, they will pass between a light source and a photo cell. The photo cell is connected to the input circuit of the counter. As each item passes between the light source and the photo cell, it breaks the light beam. The photo cell generates an input pulse that is used to increment the counter. The counter will be incremented as each unit passes the photo cell. By using the counter in this way, an accurate tally of the number of items coming off the production line is maintained in the counter. The counter automates the counting function thereby eliminating the time, effort and error of a human operator.

Frequency Measurement. The most commonly used counter function is frequency measurement. The counter circuit configuration for frequency measurement is given in Figure 10-5. The signal whose frequency

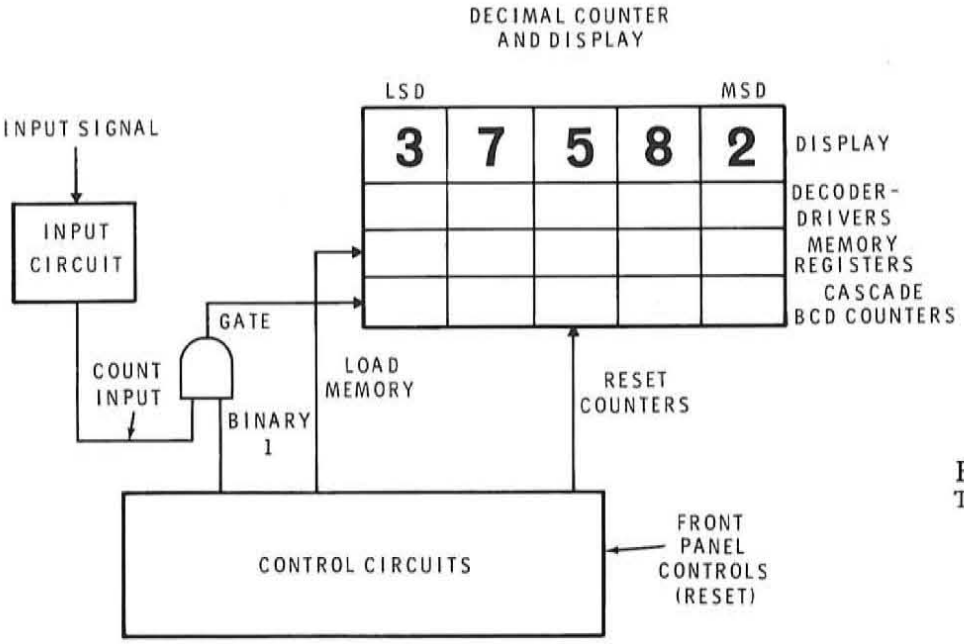
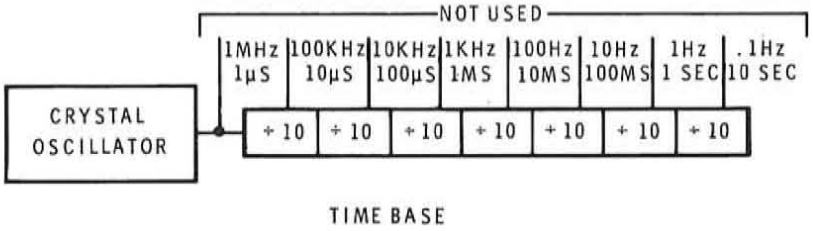


Figure 10-4
Totalize Mode



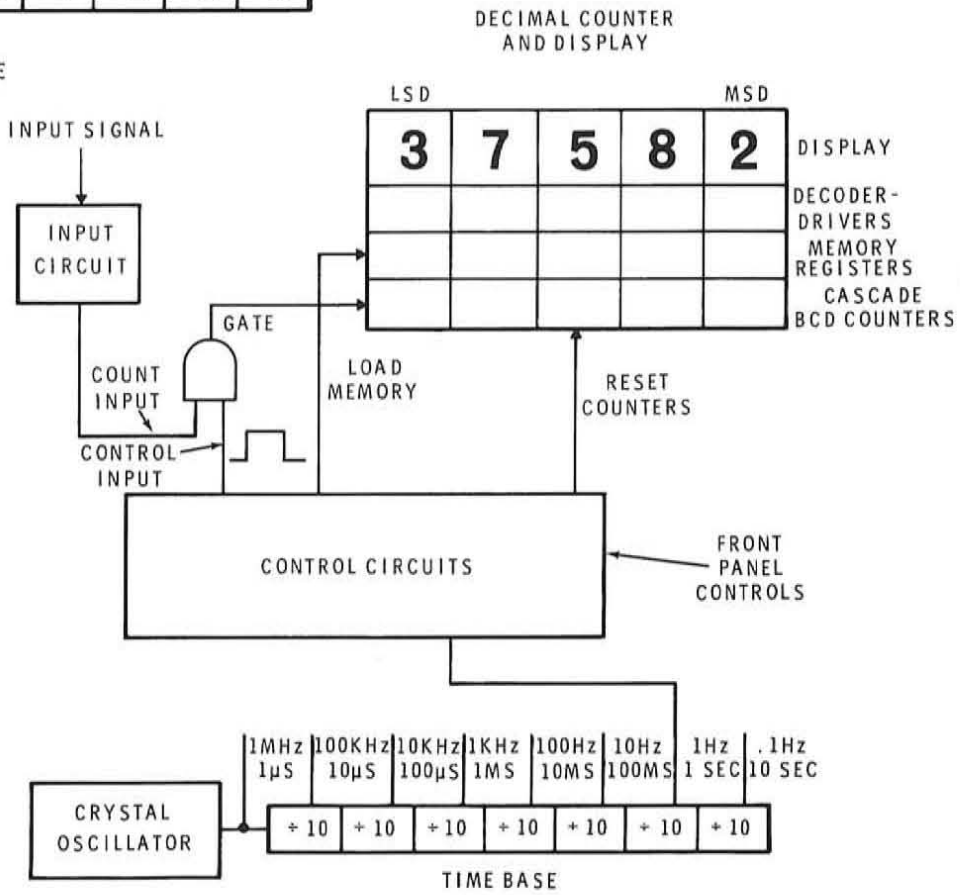


Figure 10-5
Frequency Measurement

is to be measured is applied to the input circuit. The output of the input circuit is then applied to the count input of the gate. The control input to the gate is a pulse derived from the time base. In order to perform frequency measurement, the control input to the gate is an accurate time interval. For example if the control input is a pulse that is exactly one second in duration, the counter will count the number of input pulses or cycles that occur during the one-second time interval that the gate is enabled. This will cause the display to read the frequency in cycles or pulses per second. Assuming a one second gate control pulse on the counter in Figure 10-5, the frequency shown on the display is 28573 Hz. The one-second pulse for the control input on the gate is derived from the 1 Hz output of the time base.

Higher frequencies can be measured and read out on the same display by using a gate control input of shorter duration. For example, by using a gate control pulse with a duration of 1 ms, the display will read frequency in kilohertz (kHz). The counter counts the number of input pulses that occur in one millisecond of time. Assuming a one-millisecond gate pulse for the circuit in Figure 10-5, the frequency being measured by the counter as indicated by the display is 28,573 kHz or 28.573 MHz. If the counter counts 28573 pulses in one millisecond (1/1000th second) then the number of pulses that occur in one second is 28573 × 1000 = 28573000 Hz or 28.573 MHz. The one millisecond pulse is derived from the 1 kHz output of the time base. Other time base frequencies can be used to generate control pulse intervals that are some multiple of a power of ten.

The selection of the time base output frequency determines the measurement resolution and how the frequency is displayed. In some counters the time base frequency is selected with a rotary switch or a series of pushbuttons. By making all of the time base frequencies available, much flexibility is obtained. However, optimum resolution as well as wide range can be obtained by using only two time base frequencies: 1 Hz and 1 kHz. Several counters provide a two position switch to select these two time bases. When the time base frequency is switched, the decimal point in the display is switched to the appropriate position to provide the corrected readout frequency.

Some counters such as the one in Figure 10-2 have an auto-ranging feature. Here the time base selection is made automatically, based on the input signal frequency. The auto-ranging circuitry automatically determines the correct time base frequency for maximum measurement resolution without over-ranging. Over-ranging refers to exceeding the count capability of the counter. The number of digits in a counter determines the over-ranging point for a given time base. Five and six digit counters are the most common.

Period Measurement. Another commonly used counter function is period measurement. In this mode of operation, the counter measures and displays the amount of time that it takes for the input signal to complete one cycle. Figure 10-6 shows the counter arrangement for this mode of operation. The input signal after being shaped is applied to the control circuits. The control circuits permit the input signal to operate the control input of the gate. One of the outputs of the time base is then connected to the count input of the gate through the control circuitry.

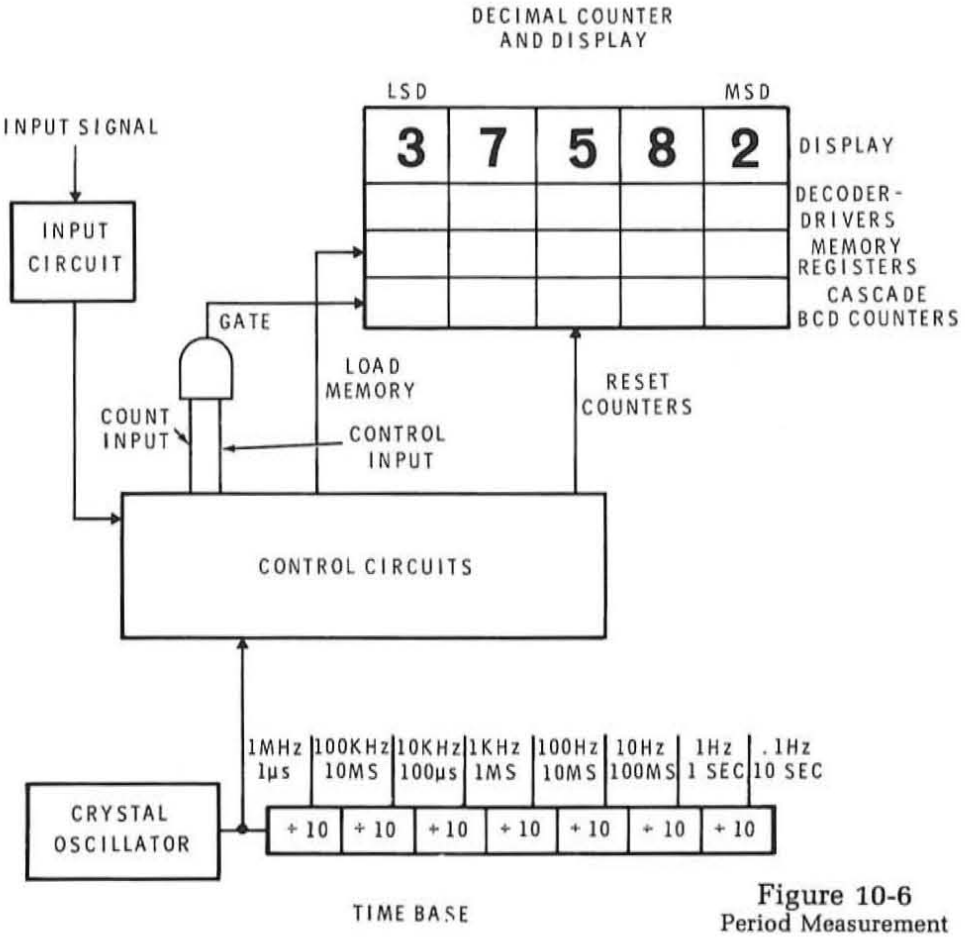


Figure 10-6
Period Measurement

The control circuitry causes the counter to be initially reset. Then the gate is opened or enabled by one period of the input signal. During this time interval, the counter is incremented by the time base signal. As indicated in Figure 10-6, the counter counts the 1 MHz pulses from the time base. According to the display in Figure 10-6, the period (t) of the input signal for this example is 28,573 microseconds. This corresponds to a frequency of $f = 1/t = 1/_{28573} \times 10^{-6} = 34.99$ Hz. The control circuit generally causes the period measurement to be made repeatedly during the time the input signal is applied.

The purpose of the period mode is to provide a means of measuring and displaying the time that it takes for an input signal to complete one cycle. However the primary benefit of the period mode is to provide improved resolution and accuracy of measurement of low frequency signals. It is difficult to obtain a highly accurate measurement of low frequency signals with most counters. For example in measuring the 60 Hz power line frequency with a counter whose gate time is 1 second, the count displayed on the counter would be 60. Since the least significant digital is 1Hz, then the resolution of the measurement is no better than 1 out of 60. This is not a very accurate measurement of the power line frequency and some means must be provided to improve the resolution.

There are several ways that the accuracy of low frequency measurements can be improved. The most direct way is to increase the measurement interval, that is increase the gate pulse duration in the frequency mode. For example, the gate pulse could be made equal to ten seconds. The counters would then count the number of cycles of the input signal that occur over a ten second interval. The display on the counter would then read 600 for a 60 Hz input. In other words, the least significant digit would then indicate tenths of a cycle rather than one cycle. By increasing the measurement interval, we provide greater resolution of measurement. The trade off, of course, is in the length of time involved in making the measurement. While ten seconds doesn't sound like a long time, for an electronic measurement it is extremely long. It is a distinct disadvantage to have to wait this length of time for a measurement to take place. For example, an increase in gate time to 100 seconds would provide an improvement in the measurement resolution and thereby the accuracy. However the measurement time is further increased.

To improve the resolution and accuracy of frequency measurements at low frequencies, the period mode can be used. The period of the input signal is measured quickly and provides much greater resolution. For example the period of the 60 Hz power line frequency is 16.667 ms. By counting the number of 1 MHz pulses of the time base that occur during one period, the counter display would read 16666. Then by taking the reciprocal of the period, the frequency is obtained. This method of measurement is much more accurate and faster than measuring the frequency directly. However, measuring the period does necessitate the computation of the frequency once the measurement has been made. This can be quickly accomplished with an electronic calculator.

There are many situations where it is desirable to obtain a highly accurate measurement of some low frequency ac signal. Audio frequency applications involving filters and musical instruments often require accurate low frequency measurements. In the field of geophysics and the vibration

testing of equipment, accurate low frequency measurements are often needed. The period mode of a standard counter can produce these low frequency measurements. However to improve the readout convenience, special low frequency counters have been developed. A typical unit is shown in Figure 10-7. This is a computing counter that is designed primarily for making accurate low frequency measurements. This counter operates in the period mode. However, once the period is measured, it automatically computes the frequency and displays it directly. The counter circuitry contains a MOS LSI calculator integrated circuit that is set up to perform the period-to-frequency computation automatically. To the operator, the computing counter appears to be measuring frequency directly. This counter provides a maximum resolution of .00001 Hz. To obtain the same resolution with a standard frequency counter would require a gate period greater than 27 hours.

Figure 10-7
A computer frequency counter measures period then computes frequency to provide high measurement resolution at low frequencies.

A further improvement in the resolution and accuracy of low frequency measurements can be made with a multiple period measurement. In this mode of operation, the time base pulses are counted for a duration equal to some multiple of the period of the input signal. This is done by feeding the signal whose period is to be measured into a frequency divider that produces division by 10, 100, 1000 or higher power of 10. This causes the control circuitry to generate a gate interval equal to 10, 100, 1000 or more times the period of the input signal. The counter counts the accurate pulses from the time base during this interval. The effect is to produce a period averaging measurement. For example if the input signal were applied to a divide by 100 circuit, the counter would accumulate pulses over a period of time equal to 100 times the period of the signal. The actual time for one cycle then would be equal to the display number divided by 100. In some counters a special period averaging mode is included. The counter display is made to read the period directly by simply shifting the decimal point the correct number of positions to compensate for the selected period interval.

Time Interval Measurements. The time interval measurement mode of a digital counter is a variation of the period mode. In this application, the count input to the gate is derived from the time base. The resolution of measurement of the time interval is determined by the time base frequency. For example, with a 1 MHz time base signal, the accuracy or resolution of measurement is 1 μs. The time resolution is the reciprocal of the time base frequency ($t = 1/f$). To the control input of the gate is applied a signal that will determine the length of time that the gate is open. This can come from a variety of sources depending upon the application. One application may be the measurement of pulse width. The signal whose pulse width is to be measured is applied to the input circuit. It causes the gate to be enabled for a period of time equal to the pulse duration. During this time interval, the counter counts the time base pulses. The display reads the pulse duration directly.

The counter can also be used to measure the time interval occurring between two independent events. A typical application is drag racing where the measurement of a car's performance is its elapsed time over a quarter mile distance. Light beams and photo cells are used at the start and finish lines to control the counter that will measure the elapsed time. The car is initially at rest at the starting line where its front wheel breaks the light beam. When a start signal is given to the driver, the car moves forward and the light strikes the photo cell. This generates an electrical impulse that is used to start the counter. The counter is initially reset and then begins to count the pulses from the time base. A 1 kHz signal is normally used to provide a measurement resolution of one millisecond. The car then traverses the quarter mile distance and breaks the light beam as it crosses the finish line. This generates another pulse that is used to close the gate and stop the time interval measurement. In this application there are two independent input pulses, one from the starting line photo cell and the other from the finish line photo cell. These start-stop pulses can be used as the inputs to a set-reset flip-flop. The starting line photo cell will set the flip-flop. The flip-flop then enables the gate. When the car crosses the finish line, the pulse from the finish line photo cell will reset the flip-flop and inhibit the gate. The count in the counter is the elapsed time.

There are many applications for the time interval mode of measurement. Another typical electronic application involves the measurement of the pull-in or release time of a relay. The counter could also be used to measure shutter opening intervals in a camera. In this mode of operation the counter in effect acts as a high resolution stop watch.

Frequency Ratio Measurement. Another mode of operation for the digital counter is frequency ratio. In this mode, the counter is used as a means of comparing two external frequencies. One of the frequencies is

used to control the duration of time that the gate is enabled while the other signal is fed to the gate to be counted by the counter. The number displayed on the readout is the ratio of the two frequencies. In this mode of operation, the internal counter time base is not used. Instead, one of the external signals acts as the time base. To provide high resolution of the ratio of two frequencies that are close to one another, one of the frequencies is fed to the time base frequency divider. This allows the gate to remain open for a period of time equal to 10, 100, 1000 or more times longer than the period of the reference input frequency.

Counter Specifications

When selecting a digital counter for a specific application or in comparing counters there are certain important specifications which must be considered. In this section we define and explain these important specifications.

Input Sensitivity. Input sensitivity is a specification that refers to the amount of input voltage required to cause the counter to operate properly. This specification usually calls out the minimum value of the input voltage required for the counter to trigger reliably. Most good counters have an input sensitivity of less than 100 mV. Higher quality counters have input sensitivity as low as 1 mV. This means that the frequency or period of a signal whose amplitude is as low as a few millivolts can be determined. The input circuit in the counter generally incorporates some amplification that permits such low level signals to be used.

Generally a counter with high sensitivity is most desirable. However, the greater the sensitivity the more susceptible the counter is to noise problems. Noise pulses riding on top of a signal to be measured can cause false triggering of the counter and an inaccurate reading. For this reason it is generally desirable to use a counter with a sensitivity level to match the application. If ultra-high sensitivity is not required the cost of the counter will be less and there will be less susceptibility to the effects of noise. Many counters have an adjustable sensitivity level that permits the input triggering level to be varied over a wide range to match a given application.

Input Impedance. The input impedance of a counter is the impedance looking into the input circuit of the counter. This is the impedance that is connected across the signal source whose frequency or period is to be measured. Most commercial counters have an input impedance of one megohm of resistance in parallel with a small value of capacitance in the 10 to 100 picofarad range. This high input impedance is chosen to minimize loading effects on the circuit under test. At high frequencies, the most important part of the input impedance becomes the shunt capacity. At frequencies above 1 MHz it is also desirable to take into consideration the

effect of any cable impedance. Normally, a coaxial cable is used to connect the signal being measured and the counter input. Such cables generally have a high capacity (above 30 pf per foot) which can significantly affect the amplitude and characteristics of the signal being measured. A high input impedance, low capacity attenuator probe normally used with oscilloscopes can also be used with most general purpose counters to minimize the effect of capacitive loading at the expense of input signal attenuation.

Frequency Range. The upper and lower frequency limits of a given counter define the counters frequency range. Because of circuitry limitations, the upper and lower frequency limits are restricted. For example a typical medium priced counter has a frequency range of 5 Hz to 30 MHz. The lower frequency limit is generally determined by the size of the input coupling capacitor and input resistance. In most counters, the input signal is ac coupled to the input circuit through a series capacitor. At the lower frequencies, the reactance of this capacitor increases and, with the input impedance, forms a voltage divider. The higher the input impedance and the larger this capacitor, the better the lower frequency response. Most typical counters have a lower frequency limit in the 1 to 5 Hz range.

The upper frequency limit of a counter is determined by the high frequency response of the input amplifier circuit, the propagation delay of the gate, and the upper frequency counting limit of the input decade counter. Some simple low cost counters have an upper frequency limit of only several megahertz. Standard direct counters are available for measuring frequencies as high as one gigahertz (1 GHz). Special counting techniques permit counters to measure frequencies up to several hundred GHz. (1 GHz = 10^9 Hz.)

Display Digits. The number of read-out display digits is an important counter characteristic. The greater the number of digits, the higher the resolution. Keep in mind that the resolution is also a function of the gate time and time base frequencies available in the counter. Most low and medium priced counters have a minimum of five digits in the readout. Counters for measuring very high frequencies in the GHz range have as many as nine digits.

Time Base. An important characteristic of any counter is the time base characteristics. This includes the time base frequencies and intervals as well as information on the stability of the time base oscillator. As a general rule, the greater the number of time base signals available, the greater the flexibility in making time and frequency measurements. The higher the frequency, the greater the resolution that can be obtained in period and time interval measurements.

The accuracy and stability of the crystal oscillator in the unit is an important factor in the quality of the counter. The crystal oscillator frequency is generally made adjustable over a narrow range to permit setting the counter to the exact frequency against an accurate known standard. From this point the stability of the counter will determine how much this frequency changes due to temperature changes and aging. Both long term and short term stability is considered.

Short term stability is affected mainly by the inherent losses in the oscillator circuit itself. Temperature variations greatly affect the short term stability of the oscillator. Temperature compensating techniques are used to help stabilize the frequency changes with temperature.

The long term stability of a crystal oscillator is due to aging of the crystal. Most crystal aging takes place in the first several months of operation causing frequency drift. After this, it stabilizes to a very low level. A wide range of stability characteristics are available in commercial counters. The degree of stability desired depends upon the application.

Modes. An important characteristic in selection of a counter is the availability of the various measurement modes. The large percentage of available commercial counters have only the frequency measurement mode. Since this is the most common mode of operation it is no disadvantage. If your application requires only the measurement of frequency such a counter will suffice. However if the counter is to be used as a general purpose laboratory or test instrument, it is desirable to incorporate other modes of measurement such as period, time interval, totalizing and frequency ratio.

Self Test Review

8. When the input signal to a counter controls the gate interval, the counted mode is:
   a. Period
   b. Frequency
   c. Totalize
   d. Ratio
9. In what counter modes is the time base *not* used?
   a. frequency
   b. period
   c. totalize
   d. time interval
   e. ratio

10. What mode is often used to improve the accuracy of low frequency measurements?
    a. frequency
    b. period
    c. totalize
    d. time interval
    e. ratio
11. In the time interval measurement mode of a computer, a time base frequency of 10 kHz is selected. What is the time resolution?
    a. 1 μs
    b. 10 μs
    c. 100 μs
    d. 1 ms
12. In the frequency mode of a digital counter, the gate pulse interval is 10 μs. The 5 digit display reads 706. What frequency does this represent?
    a. 70.6 kHz
    b. 706 kHz
    c. 7.06 MHz
    d. 70.6 MHz
13. The maximum frequency that can be indicated by a 6-digit counter with a 1 ms gate interval is _____ MHz.
14. A computing counter usually measures _____ then computes _____.
15. True or False. A counter that can reliably respond to a 5 mV signal is more sensitive than a counter that will respond to a 25 mV signal. _____
16. At very high frequencies, the input impedance to a counter is primarily
    a. capacitive
    b. inductive
    c. resistive
    d. one megohm
17. Which of the following does *not* affect the upper frequency limit of a counter?
    a. input sensitivity
    b. gate propagation delay
    c. bandwidth of input circuit
    d. speed of input BCD counter
18. The accuracy of time and frequency measurements in a counter is a direct function of the quality of the time base. The time base frequency is, in turn, greatly affected by changes in _____.
19. A 5.273 MHz signal is applied to a 5-digit counter with a 1-second gate interval. The display will read _____ .

Answers

8. a. period
9. c. totalized and e. ratio
10. b. period
11. c. 100 μs t = 1/f = 1/10000 = .0001 = second = 100 μs
12. d. 70.6 MHz
    There are 1,000,000 microseconds per second and 100,000 10 μs intervals in one second. Therefore if a counter counts 706 pulses in 10 μs it would count 706 × 100,000 = 70,600,000 pulses in one second. This is a frequency of 70,600,000 pulses per second or 70.6 MHz.
13. 999.999 MHz
14. period, frequency
15. True
16. a. capacitive
17. a. sensitivity
18. temperature
19. 73000
    With an input of 5.273 MHz, 5,273,000 pulses will occur during the one second gate interval. The 5-digit counter has a capacity of 99999. Therefore it will read only the 5 least significant digits of the input or 73,000.

A Typical Digital Counter

The best way to learn the operation of a digital counter is to analyze a typical unit. In this section we describe in detail the operation of the Heathkit Model IM-4100 Digital Counter. This is a low cost digital counter with a five digit display and a frequency range of 5 Hz to 30 MHz. It incorporates the frequency, period and totalize modes of operation. It has a sensitivity of 15 mV rms and an input impedance of 1 megohm shunted by less than 35 pF.

General Circuit Description. Refer to the block diagram in Figure 10-8 and the schematic diagram in Figure 10-9 as you read the following sections.

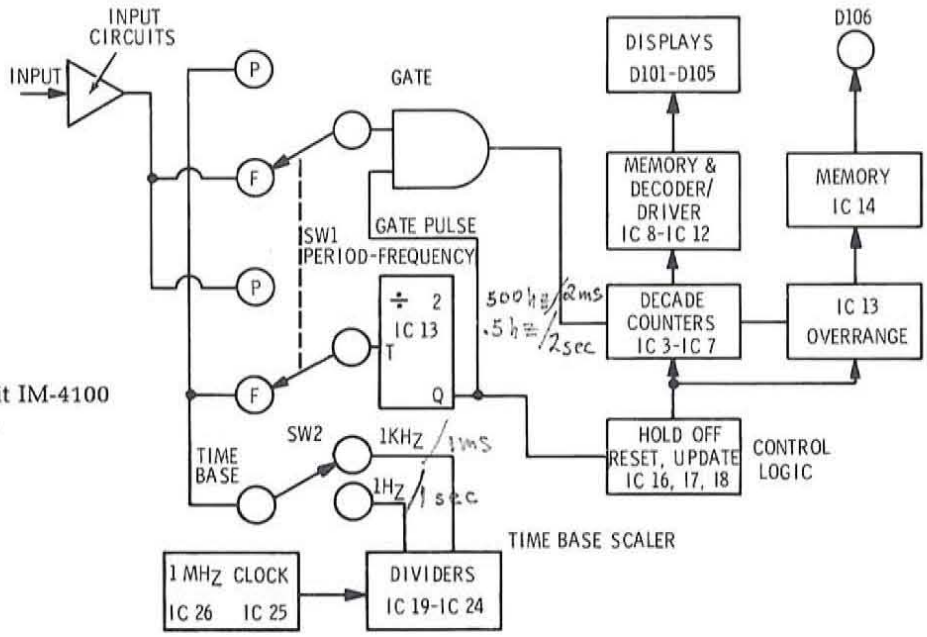


Figure 10-8
Block diagram of Heathkit IM-4100 digital counter.

The signal to be counted is applied to the input circuits which consist of the input attenuator, impedance converter, and the Schmitt trigger. There the signal is squared and applied to the GATE. During the time the gate pulse from IC 13 is also present, the GATE is open and the frequency is counted by the decade counters. At the end of the GATE pulse, the count in the decade counters is transferred to the memories by a transfer pulse. At this time, proper segments of the display units turn on and the frequency is displayed. A reset pulse then clears the decade counter so they are ready for the next time the GATE is open. The duration of the GATE pulse is determined by the position of the POWER/TIME BASE switch (SW2). The pulse is of one second duration in the kHz position and of one millisecond duration in the MHz position. A one megahertz signal is also produced for the period measurements.

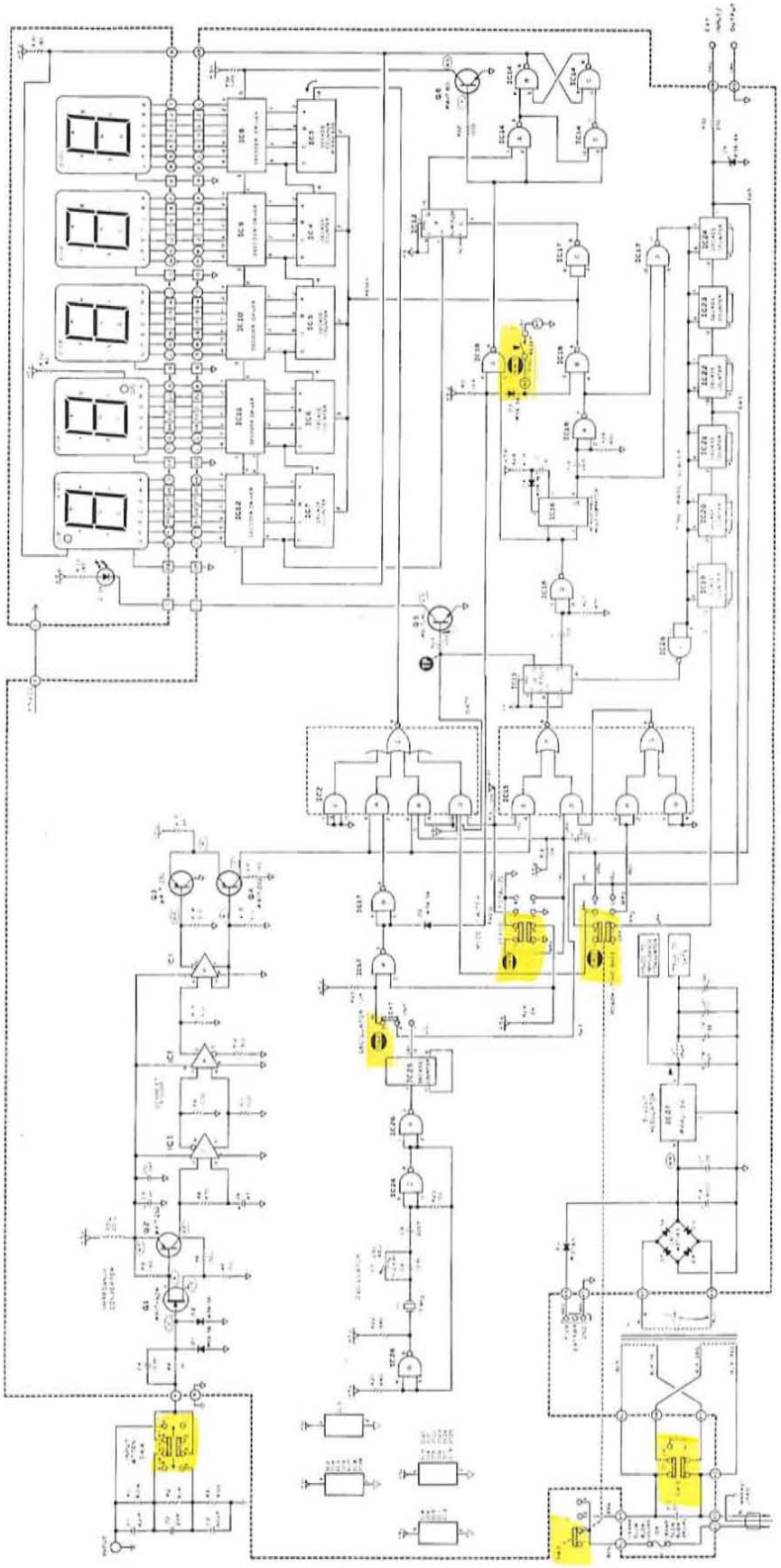


Figure 10-9
Schematic of the Heath Model IM-4100/SM-4100 frequency counter.

IC13 converts either the time base or the input period into a gate pulse depending on the position of the mode switch (SW3). By counting the input signal gated by the time base signal, frequency is displayed. By counting the time base signal gated by the input signal, the period is displayed. In the totalize mode, the counter simply counts the input signal without any gating.

Input Circuit and Schmitt Trigger. Refer to Figure 10-9. The input signal is applied to an input circuit that consists of a switchable voltage divider (R1, R2, R3) that is frequency compensated by C1, C2, and C3. This attenuates the input signal by a factor of 1, 10, or 100 depending upon the position of SW4. The signal is then coupled through C4 and R4 to D1 and D2 which provide over-voltage protection for Q1.

Transistors Q1 and Q2 are direct coupled with 100% negative feedback to form a unity gain follower circuit. The transistors provide wide bandwidth, high input impedance, low output impedance, and a gain of one. IC1C and IC1A then amplify the signal up to the input trigger threshold limits of IC1B which is wired as a Schmitt trigger. IC1 is an integrated circuit containing three differential amplifier circuits. Each amplifier has high gain and wide bandwidth. IC1B drives Q3 and Q4 which translate the signal into TTL levels making it compatible with the remaining logic circuitry. The conditioned input signal appears at the collector of Q4 and is applied to the gating circuitry.

10 MHz Clock and Scaler. A 10 MHz crystal and gates B and C of a type 7400 TTL gate (IC26) form a TTL-compatible clock oscillator. Capacitors C7, C8 and C9 provide the proper capacitive load for the crystal. C7 is variable to permit precise calibration of the oscillator. Resistors R21, R22, and R23 bias the IC and ensure efficient starting of the oscillator. Gate A of IC26 provides buffering action between the oscillator and the first decade divider of the time base scaler, a type 7490 (IC25) BCD counter. The 10 MHz clock signal is then further divided by other 7490 decade dividers, IC19 through 24, to provide appropriate gate times for the frequency mode. This divider chain is referred to as the time base scaler.

The 1 MHz time base output signal from decade counter IC25 is applied to the control circuitry and to the decade dividers in the time base scaler to provide the timing pulses to operate the counter in the various modes. This 1 MHz signal passes through switch SW6 when it is in the internal (INT) position to the input of IC24, the first decade divider in the time base scaler. The scaler then develops the 1 kHz and 1 Hz signals that will produce the 1 ms and 1 s gate pulses for frequency mode operation. The 1 kHz and 1 Hz pulses appear on the terminals of switch SW2. When the kHz position is selected, the 1 Hz signal output from IC19 is connected to gate IC15A. In turn it passes through gates IC15C and D. In the frequency mode, gate

IC15D is enabled and the 1 Hz signal passes through IC15F to the gate flip-flop IC13B. Setting SW2 to the MHz position causes the 1 kHz signal from IC22 to be connected to gate IC15A and therefore transmitted to the gate flip-flop IC13B.

In the period mode gate IC15D is inhibited by the binary 0 applied to pin 13 from SW3. This prevents the time base signal from IC15C from reaching the gate flip-flop. Instead, the input signal whose period is to be measured will operate gate flip-flop IC13B. The input signal from Q4 is applied to gate IC15E which is enabled during the period mode.

In the totalized mode the internal time base circuitry is disabled entirely. This is done by inhibiting gates IC2B and D and IC15D and E. In this mode the input signal passes through gates IC2A and E directly to the counter.

Decade Counter and Display. Figure 10-9 indicates that the main counting unit of the IM-4100 counter is made up of five integrated circuit decade counters, IC3 through IC7. IC3 is the least significant digit (LSD). The pulses to be counted are applied to pin 14 of this counter. Each of the IC's in the counter is a type 7490 TTL BCD counter similar to the one you have worked with earlier in this program. The counters are cascaded and can achieve a maximum count of 99999. Note that all of the reset pins (pin 2) are connected together so that the counter can be reset by one of the pulses from the control circuitry.

The four-line BCD output of each decade counter is applied to a memory-decoder/driver integrated circuit. These are IC8 through IC12. Each of these devices is a type 9368 decoder-driver circuit similar to the one supplied with this program. As you can recall, this device contains a 4-bit storage register and a BCD to 7-segment display device. At some time during the operation of the counter, the BCD data stored in the decade counter is transferred to the storage registers in the decoder-driver IC's. It is the BCD data stored in the registers of IC8 through IC12 that is displayed on the LED readouts. The loading of the registers in the decoder-driver IC's is accomplished by a control pulse applied to pin 3 of these circuits.

Control Circuitry. Now let's take a look at the operation of the control circuitry for the counter in each of its three operating modes. We will discuss the gating, memory and reset functions for each mode.

Assume that the mode switch SW3 is in the frequency mode and the power-time base switch SW2 is in the MHz position. The input signal from the collector of Q4 is applied to gate B of IC2. This gate acts as the main gate for the counter in the frequency mode. The output of gate B is coupled through gate E in IC2 and is applied to the input of the LSD decade counter IC3. Gate IC2B is enabled by a one second or one millisecond pulse from pin 11 of IC13B. IC13B is a J-K flip-flop that is set and

reset by the control circuitry. With IC13B set, transistor Q5 is turned on. This enables D106, the gate LED indicator. Flip-flop IC13B remains set for one millisecond or one second depending upon the setting of the time base switch. During this time the counter counts or accumulates the pulses applied to the input.

When IC13B resets, its $\overline{Q}$ output at pin 10 will go high. This causes the output of IC18D to go low momentarily. IC18D is a TTL gate connected as a pulse generator. The 470 ohm resistor R27 holds both inputs of the gate at a low enough potential so that to the gate it appears that a binary 0 is applied. This keeps the output of IC18D normally high. When the input to the capacitor C11 goes high, the gate inputs also go high. This forces the output low. However capacitor C11 charges quickly through R27 thereby leaving little voltage across R27. As a result the input again appears to be a binary 0 and the output switches back to its original high state. As you can see IC18D generates an output pulse that switches from high to low and back again. The duration of the pulse is a function of the time constant of C11 and R27. This pulse is produced when IC13B resets.

The output from IC18D is a negative going pulse that is applied to IC18C which drives Q6. This momentary pulse is applied to pin 3 of the decoder/driver ICs, IC8 through IC12. This causes the count stored in the decade counters to be transferred to the storage registers. The LED's then display the count.

The trailing edge of the pulse from pin 11 of IC18D triggers IC16. This monostable multivibrator produces a 200 ms output pulse.

The termination of the 200 ms output pulse from IC16 triggers IC18A. This circuit generates a negative going pulse which is used to reset the decade counters IC3 through IC7. This pulse is applied to the counters through IC18B. The reset pulse is also used to reset flip flop IC13A through gate IC17C. Flip-flop IC13A is used in the over-range circuit which will be discussed later.

The complement output of the monostable (IC16) and the pulse from IC18A are also used to gate IC17D. This gate causes all of the decade dividers in the time base scaler to be set to 1001. The next input pulse from the time base oscillator will then cause all of these time base decade dividers to be cycled to zero (0000). The signal from IC17D that sets the time base dividers to 1001 is also used to reset flip-flop IC13B through gate IC26D. The timing pulses for the frequency mode are shown in Figure 10-10.

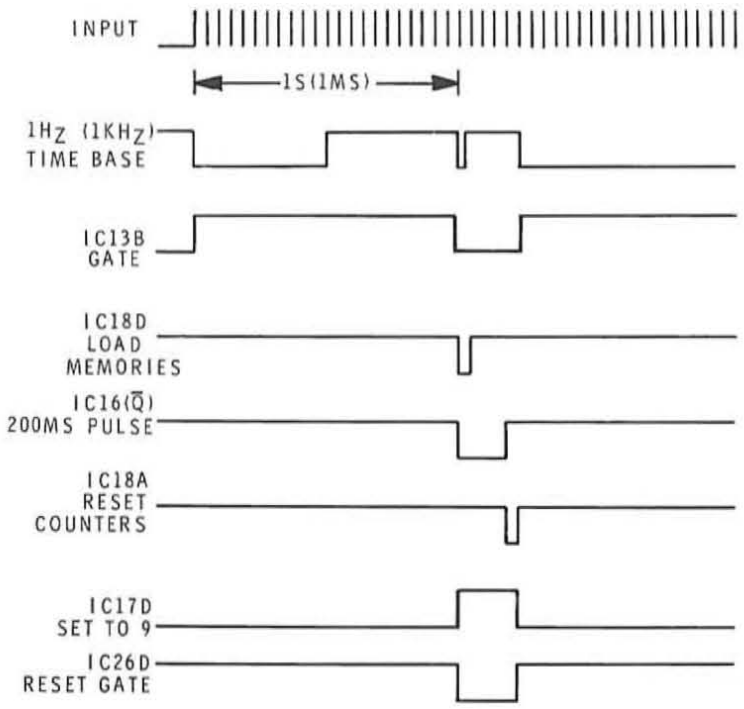


Figure 10-10
Timing diagram of control pulses of IM-4100 counter in the frequency mode.

In the period mode it is the input signal that controls the GATE flip-flop IC13B and therefore determines the period of time during which the GATE is open. During this time the counter is incremented by the 1 kHz or 1MHz time base signals. This permits period measurements with a resolution of 1 millisecond or 1 microsecond to be made.

Assume that the mode switch SW3 is in the period position as shown in Figure 10-9. This inhibits gates IC2B and IC15D which are used in the frequency mode. Gate IC17A is also inhibited. At this time gate IC15E is enabled. The input signal from the collector of Q4 is coupled through this gate and IC15F to the gate flip-flop IC13B. The 1 kHz or 1MHz signal from the time base is applied to gate IC2D pin 11. Here gate IC2D is enabled by the gate flip flop IC13B. The time base signal is allowed to pass through gates IC2D and E to the counter for a period of time equal to one cycle of the incoming signal.

In the period mode, the control circuitry for the memory transfer and reset operation is similar to that for the frequency mode. At the end of the gate interval, a pulse is generated by IC18D as before to cause the data in the decade counter to be transferred to the storage latches and a display. The termination of the 200 millisecond pulse generated by IC16, in turn, causes the reset pulse to be generated by IC18B. See Figure 10-11 for the control pulses in the period mode.

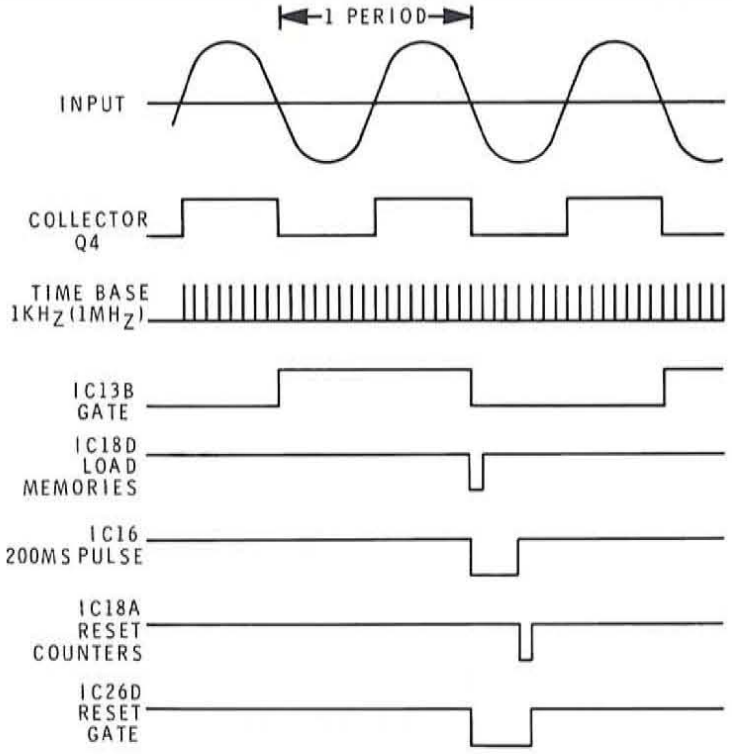


Figure 10-11
Timing diagram of control pulses of IM-4100 counter in the period mode.

In the totalized mode, the input pulses to be counted are passed through the input circuitry and appear at the collector of Q4 as in the other modes. The input signal is then applied to gate IC2A. With the mode switch SW3 in the totalized mode, gates IC2D and IC15E are inhibited. This prevents the gate function of the counter from operating. Gate IC2A is enabled at this time so that the signals to be counted are applied directly to the decade counter without any control.

When the totalized mode is used, both inputs to IC17A are high thereby holding the output of this gate low. This low level is coupled through diode D3 to the input to gate IC18C. This holds the output of IC18C high thereby causing the output of Q6 to go low. What this does is to enable the memory storage latches in IC8 through IC12 so that the pulses being counted in the decade counters will be transferred directly to the display. As the input pulses occur you can see the count change on the LED readout.

The only control available in the totalize mode is the front panel reset switch SW5. When this manual pushbutton is depressed, it forces the cathode end of D5 low. This generates a reset pulse through IC18B which clears the decade counters. In operating the counter in the totalized mode, the counter should be initially reset to zero prior to the counting of any input events.

In the totalize mode, it is possible to gate the input signal to the counter by using an external input signal connected to the input normally associated with the time base. The gate pulse is coupled through resistor R33 and switch SW6 to gate IC17A. If the external control signal is low, the output of IC17B will be low and will therefore inhibit gate A and prevent the input signals from appearing at the counter input. Figure 10-12 shows the control pulses in the totalize mode with an external gating signal.

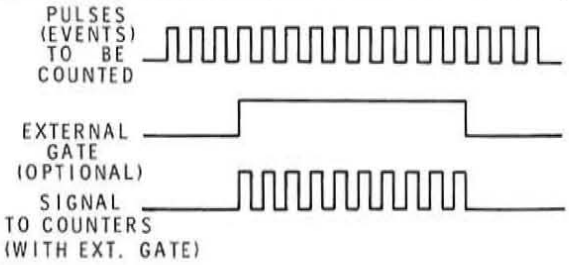


Figure 10-12
Timing diagram of control pulses of IM-4100 counter in the totalize mode.

Over-Range Detection. The IM-4100 counter has an over-range circuit that will give the user an indication of when the count input exceeds the count capability of the counter. If the gate duration and the time base signals are such that the count passes from 99999 to 00000, a carry output pulse occurs at the D output of the most significant digit decade counter IC7. This will toggle flip-flop IC13A. When this flip-flop is set, it indicates that an over-range condition has occurred. This flip-flop will remain in the binary 1 state indicating an over-range condition until the normal reset pulse is generated by the control circuitry. At this time IC13A is reset via gate IC17C.

The over-range condition is transferred to a d-type flip-flop (data latch) made up of the gates in IC14. When the control circuitry generates the pulse to transfer the data from the decade counter to the storage registers, data latch input gates IC14A and D will be enabled. If an over-range condition exists at this time as determined by the condition of IC13A, the data latch will be set. The output of the data latch at IC14 pin 6 normally keeps the over-range indicator LED shorted. This LED indicator is the decimal point element in the seven-segment LED display D105. It is normally lighted by the supply voltage through the 180 ohm resistor R101. If the data latch is reset, its output at IC14 pin 6 is low and therefore the over-range indicator is off. However when the data latch is set, the decimal point element is lighted thereby indicating an over-range condition.

Self Test Review

20. Which integrated circuit (Figure 10-9) generates the gate pulse?
    a. IC2
    b. IC13
    c. IC15
    d. IC19
    e. IC22
    f. IC26
21. The purpose of LED D106 is to indicate when:
    a. power is on
    b. an over-range condition occurs
    c. a decimal point is needed in the display
    d. the gate is open
22. The proper operating sequence of the decade counter is:
    a. reset, count, transfer to memory
    b. count, reset, transfer to memory
    c. transfer to memory, count, reset
23. The time base frequencies used in the frequency mode are _____ Hz and _____ Hz.
24. The time base frequencies used in the period mode are _____ Hz and _____ Hz.
25. The gate for the frequency mode is IC _____ .
26. Frequency ratio measurements can be made with the Heath IM-4100 counter. To do this, which mode should be selected?
    a. frequency
    b. period
    c. totalize

    One input will be applied to the regular counter input. The other input will be applied to the external input/output connector (see Figure 10-10). SW6 must then be set to the _____ position.

    In the frequency ratio mode, the clock oscillator _____ used.
    is/is not
27. The time base frequency is 1 MHz. The display reads 8521. The _____ mode is being used and the input frequency is _____ Hz.

Answers

20. b. IC13
21. d. indicates when the gate is open (enabled)
22. a. reset, count, transfer to memory
23. 1000 Hz (1 KHz) and 1 Hz
24. 1000 Hz (1 KHz) and 1000000 Hz (1 MHz)
25. IC2B
26. b. period
    EXT position
    is not
27. period, 117.3 Hz ($f = 1/8521 \times 10^{-6}$)
    (The 1 MHz time base signal is used *only* in the period mode).

DIGITAL COMPUTERS

The greatest application for digital circuits today is in digital computers. Digital circuits were originally developed to provide a means of implementing digital computers. As new circuits and techniques have been developed, computer performance has improved. The greatest impact on the digital computer has been the development of integrated circuits. IC's have greatly reduced the size and cost of digital computers and has made them more powerful. Over the years digital computers have continued to decrease in price. Their size and power consumption have also decreased significantly. At the same time, their performance and sophistication have increased, making them practical for a wider range of applications.

Recent technological advances in semiconductor techniques have created a unique digital product. Large scale integration of digital circuits has permitted the semiconductor manufacturers to put an entire digital computer on a single chip of silicon. These computers are known as microprocessors. We normally think of digital integrated circuits as being the gates and flip-flops used to implement a computer. Now, the computer itself is a single low cost integrated circuit. But the power of this device is significant, and for many applications it can replace hundreds of SSI and MSI circuits. This significant development will further broaden the applications for digital computers. Best of all, it will increase the sophistication and capabilities of the electronic equipment that uses them.

Today digital computers (microprocessors) are as much a part of digital techniques and digital design as any of the smaller and simpler circuits. While it is impossible to cover all aspects of this exciting field in this unit, we will introduce you to the digital computer and related techniques as they apply to implementing digital systems. Our primary emphasis will be on the microprocessor and its ability to replace standard hard-wired control logic systems.

What is a Digital Computer?

A digital computer is an electronic machine that automatically processes data by the use of digital techniques. Data refers to any information such as numbers, letters, words, or even complete sentences and paragraphs. Processing is a general term referring to a variety of ways in which the data can be manipulated. The computer processes the data by performing arithmetic operations on it, editing and sorting it, or evaluating its characteristics and making decisions based upon it. In addition to being able to manipulate data in a variety of ways, the computer contains an extensive memory where data is stored. The key characteristic of a digital

computer is its ability to process data automatically without operator intervention.

The manner in which the data is manipulated is determined by a set of instructions contained within the machine. These instructions form a program that tells the computer exactly how to handle the data. The instructions are executed sequentially to carry out the desired manipulations. Most computers are general purpose in that the instructions can be assembled into an almost infinite variety of application programs. Each computer has a specific instruction set. These instructions are then put into the proper sequence to perform the required calculation or operation. The process of writing the desired sequence of instructions is called programming.

How Computers are Classified

There are many different types of digital computers and a variety of ways in which they can be classified. One method of classifying computers is by size and computing power. There is a broad spectrum covering all types of computers. At one end of the spectrum are large scale computers with extensive memory and high-speed calculating capabilities. These machines can process huge volumes of data in a short period of time and in any desired manner. At the other end of the spectrum are the small scale, low cost digital computers such as the microprocessor whose application and computing power is more limited.

Computers are also classified by function or application. The most commonly known digital computer is the electronic data processor that is used by most business, industry and government organizations for maintaining records, performing accounting functions, maintaining an inventory, and providing a wide variety of other data processing functions. Then there are the scientific and engineering computers that are used primarily as mathematical problem solvers. They greatly speed up and simplify the calculations of complex and difficult scientific and engineering problems.

Another way to classify digital computers is general purpose or special purpose. General purpose machines are designed to be as flexible as possible. This means that they can be programmed for virtually any application. Special purpose computers, on the other hand, are generally dedicated to a specific application. They are designed to carry out only a single function. General purpose computers with a fixed program become special purpose computers.

Most digital computers are of the general purpose type. Most have versatile instruction sets which permit the computer to be programmed to

perform almost any operation. With the proper program, a general purpose computer can perform business data processing functions, scientific and mathematical calculations, or industrial control functions.

The most widely used computers are the small scale machines. These include the minicomputer, the microcomputer, the programmable calculator, and the microprocessor. While all of these small scale machines together account for less than 10% of the total computer dollar investment, they represent more than 95% of the unit volume of computers. Small scale computer systems are very low priced and are within the reach of almost everyone. Today a complete computer system can be purchased for less than the price of a new automobile. Microprocessors and programmable calculators are even less expensive. There are many thousands of small computers in use today. Your own personal contact with a digital computer will no doubt be through some type of small scale computer.

Minicomputers. The largest of the four types of small computers is the minicomputer. A minicomputer is a general purpose digital computer usually constructed of TTL or ECL bipolar logic circuits. It is supported with software and peripheral units. Software refers to the programs supplied with the computer that make it easy to use. Peripheral units are the input-output devices that allow an operator to communicate with the computer. Typical peripheral units are typewriters, card readers and printers. Minicomputers are similar to the larger digital computers, but their memory capacity, speed and applications are more limited.

A complete but minimum minicomputer can be purchased for less than $1,000. This does not include peripheral equipment. However, such machines are often purchased to be built into a larger piece of equipment or a system for use as a controller. The users of such computers are referred to as original equipment manufacturers (OEM). A complete stand-alone minicomputer with sufficient memory, peripheral devices and software to be used for general purpose computing can be purchased for less than $5,000.

Microcomputers. A microcomputer is similar in many respects to a minicomputer in that it is a general purpose machine that can be programmed to perform a wide variety of functions. However, the microcomputer is normally smaller and more restricted in its application. Its speed and memory capacity is less than a minicomputer. As a result, microcomputers are substantially less expensive than minicomputers. Microcomputers are more often used in dedicated,

single function applications. Software and peripheral support is at a minimum. Most microcomputers are implemented with MOS LSI circuitry.

Programmable Calculator. A programmable calculator can be classified as a special purpose microcomputer. These machines are similar in many respects to the hand-held and desk top electronic calculators widely available. The programmable calculator has an input keyboard for entering data and a decimal display for reading out the results of calculations. In a standard calculator, an operator enters the numbers to be manipulated and the functions to be performed by depressing keys on the keyboard in the proper sequence. The solution to the problems then appears on the display. A programmable calculator can be used this way also, but it contains a memory and control unit that is used to automate the problem solving process. The data to be operated upon and the functions to be performed are entered via the keyboard and stored in the memory in the proper sequence. When enabled, the programmable calculator will then automatically solve the problem stored in its memory without operator control. Programmable calculators offer the advantage of improved speed and convenience over standard calculators when the same problem must be computed several times with different data. Long problems requiring complex data and many mathematical operations are also best solved by a programmable calculator as they relieve the operator from the tedious work and greatly minimize errors. Another advantage of the programmable calculator over other types of digital computers is its ability to communicate directly with the operator through the keyboard and decimal readout display.

Microprocessors. A microprocessor is the smallest and least expensive type of digital computer that still retains all of the basic features and characteristics of a computer. It can be implemented with standard digital integrated circuits or it is available as a single large scale integrated (LSI) circuit. While the capabilities of a microprocessor are limited when compared with a microcomputer or minicomputer, this device is still a very powerful unit. It extends the applications of computer techniques to many areas where minicomputers and microcomputers are not economically feasible.

Microprocessors are generally designed to perform a dedicated function. These devices are built into electronic equipment that will be used for some specific application. Some typical dedicated applications include traffic light controllers, electronic scales and cash registers, and electronic games. In addition, engineers are finding that low cost microprocessors can be used to replace standard hard-wired digital logic. Design time and cost can be significantly reduced in the design of a digital system when microprocessors are used. A microprocessor can be used

economically if the design is equivalent to thirty or more standard TTL integrated circuit packages. Such hard-wired logic designs are replaced by a microprocessor with a stored program. The program stored in a read only memory permits the microprocessor to carry out the same functions as a hard-wired logic controller. Microprocessors can also be used as the main component of a minicomputer or microcomputer.

Self Test Review

28. The two types of binary information stored in the memory of a digital computer are _____ and _____ .
29. A list of computer instructions for solving a particular problem is called a _____ .
30. Both a digital computer and an electronic calculator can solve mathematical problems, but the unique feature of the computer is its ability to solve the problem _____ .
31. The simplest digital computer is called a _____ .

Answers

28. instructions and data
29. program
30. automatically
31. microprocessor

Digital Computer Organization

All digital computers are made up of four basic units. These are the memory, the control unit, the arithmetic-logic unit (ALU), and the input-output (I/O) unit. These major sections and their relationship to one another are illustrated in Figure 10-13. An understanding of digital computer operation starts with a knowledge of how these sections operate and how they affect one another.

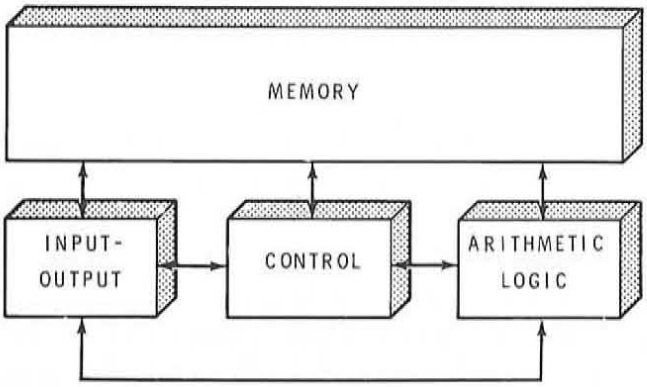


Figure 10-13
General block diagram of a digital computer.

Memory. The heart of any digital computer is its memory. It is in the memory where the program and data are stored. As indicated earlier, the program is a series of instructions that are stored and executed in sequence to carry out some specific function. The instructions cause the computer to manipulate the data in some way.

Computer memories are organized as a large group of storage locations for fixed length binary words. A computer instruction is nothing more than a binary word whose bit pattern defines a specific function to be performed. The data to be processed by the computer is also a binary word. A computer memory is an accumulation of storage registers for these instruction and data words. Most computers have memories capable of storing many thousands of words.

Digital computers typically have a fixed word size. A 32-bit word is common for many large computers. Minicomputers usually have a 16-bit word. Microprocessors widely use an 8-bit word. Memory sizes range from approximately several hundred words to several hundred thousand words of storage. A typical minicomputer may provide 4096 16-bit words. A microprocessor may use 1024 words of 8-bit memory. The number of words in memory is generally some power of two.

Each memory location appears to be like a storage register. Data can be loaded into the register and retained. The word can also be read out of memory for use in performing some operation. Each memory word is given a numbered location called an address. The address is a binary word used to locate a particular word in memory. The normal procedure is to store the instruction words in sequential memory locations. The instruction word generally contains an address which refers to the location of some data word to be used in carrying out the operation specified. The instructions stored in the sequential memory locations are executed one at a time until the desired function is performed.

Most modern digital computers use semiconductor memory. These are MOS LSI circuits where data is stored in latch flip-flops or as the charge on a capacitor. Semiconductor memories are small, fast, and inexpensive. Many computers, however, still use magnetic core memories. In these memories, binary data is stored in tiny donut-shaped magnetic cores. By magnetizing the core in one direction a binary zero is stored. Magnetizing the core in the opposite direction causes it to store a binary one. Electronic circuitry associated with the cores is used to store data into the memory and read it out. The advantage of core memories over semiconductor memories is their non-volatility. When power is removed from a semiconductor memory, all of the data is lost. Removing the power from a magnetic core memory has no effect on the data contents. Because the cores are permanently magnetized in one direction or the other, all data is retained.

The typical organization of a computer memory is shown in Figure 10-14. It consists of the semiconductor or magnetic core elements that retain the binary data. The memory used in a digital computer is generally referred to as a random access read/write memory. Random access refers to the ability of the computer to directly seek out and access any specific word stored in the computer memory. Read/write refers to the ability of the memory to store data (write) or to retrieve data for use elsewhere (read).

As you can see from Figure 10-14, the access to a specific word in memory is achieved through the memory address register (MAR) and memory address decoder. The memory address register is a flip-flop register into which is placed a multi-bit binary word that designates the location of a desired word in memory. If the address 00010011 is stored in the MAR, the content of memory location 19 is referenced. The address word may

refer to the location of an instruction or a data word. The size of the address word determines the maximum memory size. For example, if the memory address word is 12-bits in length, the maximum number of words that the computer memory can contain is $2^{12} = 4096$ words. (called a 4K memory)

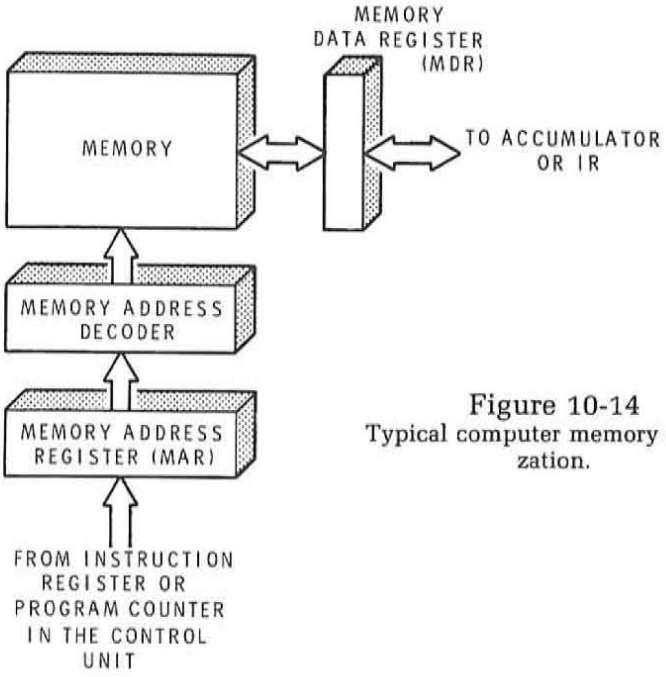


Figure 10-14
Typical computer memory organization.

The output of the memory address register drives the memory address decoder which recognizes one unique memory address word at a time and enables the appropriate location. In semiconductor memories, the memory address decoder is generally a fixed part of the integrated circuit memory itself. When an address word is loaded into the MAR, the specific location in memory designated by that address is enabled. Data can then be written into or read out of that memory location.

The access to the addressed memory location is made through a memory data register (MDR) or memory buffer register (MBR). This is a flip-flop register into which the data or instruction word is stored on its way into or out of the memory. A word to be stored in memory is first loaded into the MDR and then stored in the addressed memory location. If a read operation is being carried out by the memory, the data stored in the addressed location is first loaded into the MDR. From there it is sent to other portions of the computer as needed. Many computers do not use an MDR. Instead the data or instruction goes to or comes from another register in the computer.

Read Data from memory

Control Unit. The control unit in a digital computer is a sequential logic circuit. Its purpose is to examine each of the instruction words in memory, one at a time, and generate the control pulses necessary to carry out the function specified by that instruction. The instruction, for example, may call for the addition of two numbers. In this case, the control unit would send pulses to the arithmetic-logic unit to carry out the addition of the two numbers. If the instruction calls for the storage data in memory, the control unit would generate the necessary control pulses to carry out that storage operation. As you can see, it is the control unit that is responsible for the automatic operation of the digital computer.

Almost any type of sequential logic circuit can be used to implement the control unit. However, most modern digital computers incorporate a microprogrammed control unit using a ROM. Here special binary words known as microinstructions are stored in the read only memory. When an instruction is analyzed by the control unit, that instruction will cause a certain sequence of microinstruction words in the ROM to be executed. The result is the generation of logic signals that will carry out the operation designated by the instruction. The instructions set for any digital computer is defined by the operation of the control unit.

The exact logic circuitry used in the control unit varies widely from one machine to another. However, the basic elements are shown in Figure 10-15. The control unit consists of an instruction register, a program counter, an instruction decoder, a clock oscillator, and some type of sequential logic circuit used for generating the control pulses.

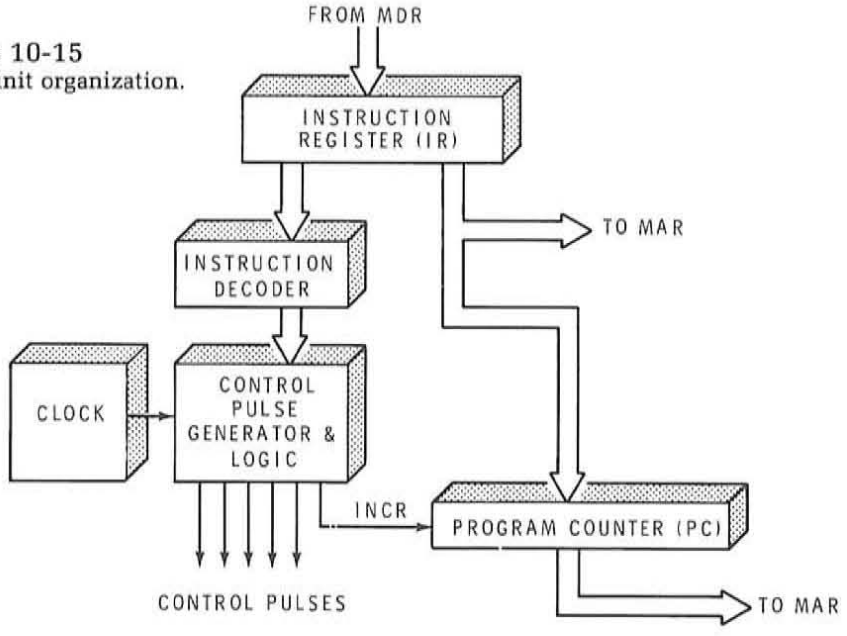


Figure 10-15
Typical control unit organization.

The instruction register is a multi-bit flip-flop register used for storing the instruction word. When an instruction is taken from memory, it passes through the MDR and then into the instruction register. From here the instruction is decoded by the instruction decoder. This logic circuitry recognizes which instruction is to be performed. It then sends the appropriate logic signals to the control pulse generator. Under the control of the clock oscillator, the control pulse generator then produces the logic signals that will enable the other circuitry in the machine to carry out the specified instruction.

The program counter is simply a binary up counter that keeps track of the sequence of instructions to be executed. The program consists of instructions that are stored in sequential memory locations. To begin a program, the program counter is loaded with the starting address. The starting address is the location of the first instruction in the program to be executed. The first instruction is then read out of memory, interpreted and carried out. The control circuitry then increments the program counter. The contents of the program counter is then fed to the memory address register that then permits the next instruction in sequence to be addressed. Each time an instruction is executed, the program counter is incremented so that the next instruction in sequence is fetched and executed. This process continues until the program is complete.

In Figure 10-15 you will notice a connection between the instruction register and the program counter. There are times when the instruction itself will modify the contents of the program counter. Some instructions specify a jump or branch operation that causes the program to deviate from its normal sequential execution of instructions. The instruction register will contain an address that will be loaded into the program counter to determine the location to which the program jumps.

Arithmetic-Logic Unit. The arithmetic-logic unit (ALU) is that portion of the digital computer that carries out most of the operations specified by the instructions. The arithmetic-logic unit performs mathematical operations, logical operations and decision making functions. Most arithmetic-logic units can perform addition and subtraction. Multiplication and division operations are generally programmed. The ALU can also perform logic operations such as inversion, AND, OR, and exclusive OR. In addition, the ALU can make decisions. It can compare numbers or test for specific quantities such as zero or negative numbers.

The arithmetic-logic unit and control unit are very closely related, so much so that it is sometimes difficult to separate them. Because of this, the ALU and control unit together are often referred to as the central processing unit (CPU). Most microprocessors are single chip LSI CPUs.

The arithmetic-logic unit in a digital computer varies widely from one type of machine to another. Figure 10-16 shows the ALU circuitry associated with a very simple, minimum digital computer. The heart of the arithmetic-logic unit is the accumulator register. It is in this register where most of the computer operations take place. Here the data is manipulated, computations are carried out, and decisions are made.

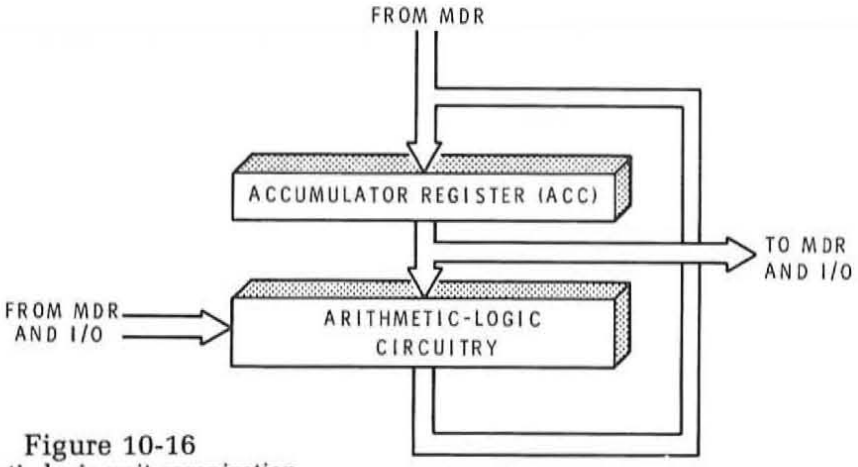


Figure 10-16
Arithmetic logic unit organization.

The accumulator register is a flexible unit that can usually be incremented or decremented. It can also be shifted right or shifted left. Many of the instructions define operations that will be carried out on the data stored in the accumulator register. The size of the accumulator register is generally determined by the basic computer word size, which is the same as the memory word size.

Associated with the accumulator register is the arithmetic-logic circuitry. For the most part, this circuitry is a binary adder. With the binary adder, both binary addition and subtraction can be accomplished. The arithmetic-logic circuitry is also usually capable of carrying out logic operations such as AND, OR, and exclusive OR on the data stored in the accumulator register.

The arithmetic-logic circuitry is capable of adding two binary words. One of the binary words is stored in the accumulator. The other binary word is stored in the memory data register. The sum of these two numbers appears at the output of the arithmetic-logic circuitry and is stored in the accumulator register replacing the number originally contained there. Most other operations with the arithmetic-logic circuitry is carried out in this manner. The two words to be manipulated are initially stored in the accumulator and the MDR with the results of the operation appearing back in the accumulator replacing the original contents.

Input-Output Unit. The input-output (I/O) unit of a computer is that section that interfaces the computer circuitry with the outside world. In order for the computer to communicate with an operator or with peripheral equipment, some means must be provided for entering data into the computer and reading it out. Data and programs to be stored in the memory are usually entered through the input-output unit. The solutions to calculations and control output signals are usually passed to the external equipment through the I/O unit.

The I/O unit is generally under the control of the CPU. Special I/O instructions are used to transfer data into and out of the computer. More sophisticated I/O units can recognize signals from extra peripheral devices called interrupts that can change the operating sequence of the program. Some I/O units permit direct communications between the computer memory and an external peripheral device without interference from the CPU. Such a function is called direct memory access (DMA).

DMA
Direct Memory access

The input/output section of a digital computer is the least clearly defined of all digital computer sections in that it can vary from practically no circuitry at all to very complex logic circuitry approaching the magnitude of the remainder of the computer itself. For our explanation of digital computer operation here, we will assume the simplest form of input/output circuitry. Data transfers between the computer and external peripheral devices take place via the accumulator register. Data to be inputted and stored in memory will be transferred a word at a time into the accumulator and then into the memory through the MDR. Data to be outputted is first transferred from the memory into the MDR, then into the accumulator, and finally to the external peripheral device. These data transfers into and out of the accumulator register take place under the control of the CPU and are referred to as programmed I/O operations. Special input/output instructions cause the proper sequence of operations to take place.

Most digital computers can also perform I/O operations at the request of an interrupt. An interrupt is a signal from an external device requesting service. The external device may have data to transmit to the computer or may require the computer to send it data. When an interrupt occurs, the computer completes the execution of its current instruction, then jumps to another program in memory that services the interrupt. Once the interrupt request has been handled, the computer resumes execution of the main program. Data transfers occurring in the interrupt mode can also take place through the accumulator.

Digital Computer Operation

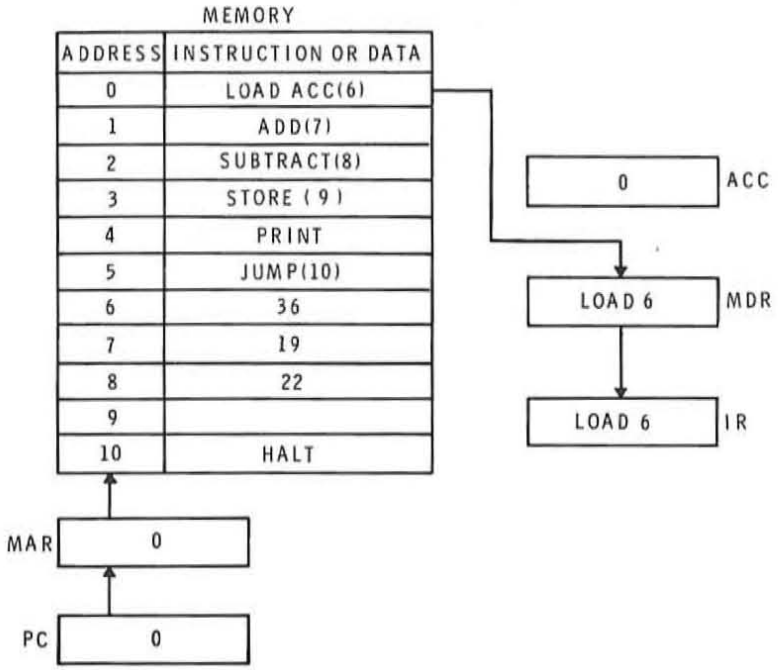
Now that you are familiar with the basic architecture of a digital computer, you are ready to see how the various sections operate together to execute a program. The units we described in the previous section, together, actually form a simple, hypothetical digital computer that we will use here to demonstrate how a computer operates. We will assume that a program for solving a specific problem is already stored in the computer memory. The computer will execute each instruction in sequence until a solution is reached. We will describe the operation of the computer and show the contents of each of the registers as the program is carried out.

Assume that the problem to be solved is a simple mathematical operation that tells us to add two numbers, subtract a third number, store the result, print the answer, then stop. The numbers that we will work with are 36, 19 and 22. The program calls for adding 36 and 19, subtracting 22, then storing and printing the answer.

The solution to this simple problem as it is solved step-by-step by the computer is illustrated in Figure 10-17. Here we show a simplified block diagram of the digital computer showing the memory and the major registers. The program is stored in memory. The contents of each memory location, either instruction or data, is shown adjacent to the memory address. To solve this problem, the computer sequentially executes the instructions. This is done in a two-step operation. First, the instruction is fetched or read out of memory. Second, the instruction is executed. This fetch-execute cycle is repeated until all of the instructions in the program have been executed. The figures in Figure 10-17 illustrate the contents of the various registers and memory locations for each fetch and execute operation.

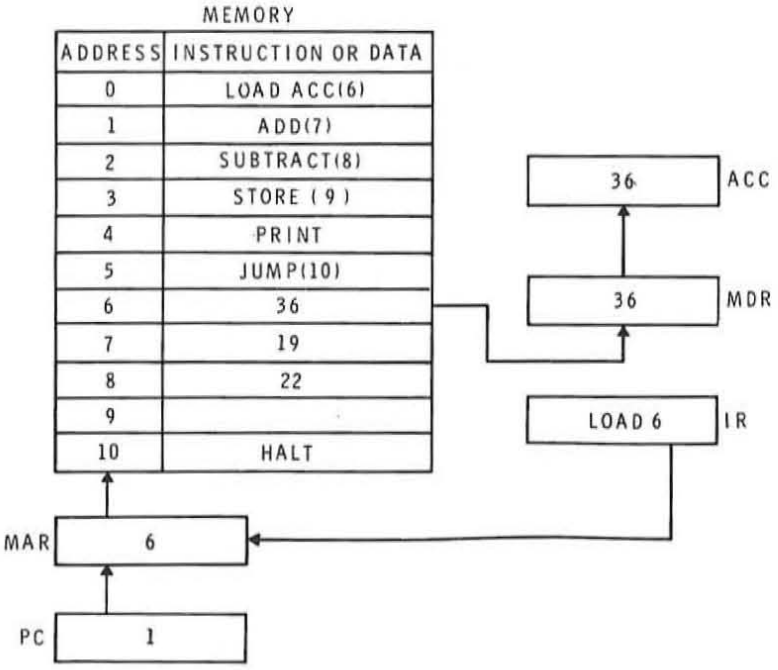
In Figure 10-17A, the first instruction of the program is fetched. The instruction word is read out of memory and appears in the memory data register (MDR). It is then transferred to the instruction register (IR) where it is interpreted. Note that the memory address register (MAR) contains 0, which is the address of the first instruction. The accumulator register (ACC) is set to 0 prior to the execution of the program.

Figure 10-17



(A) Fetch first instruction (LOAD)

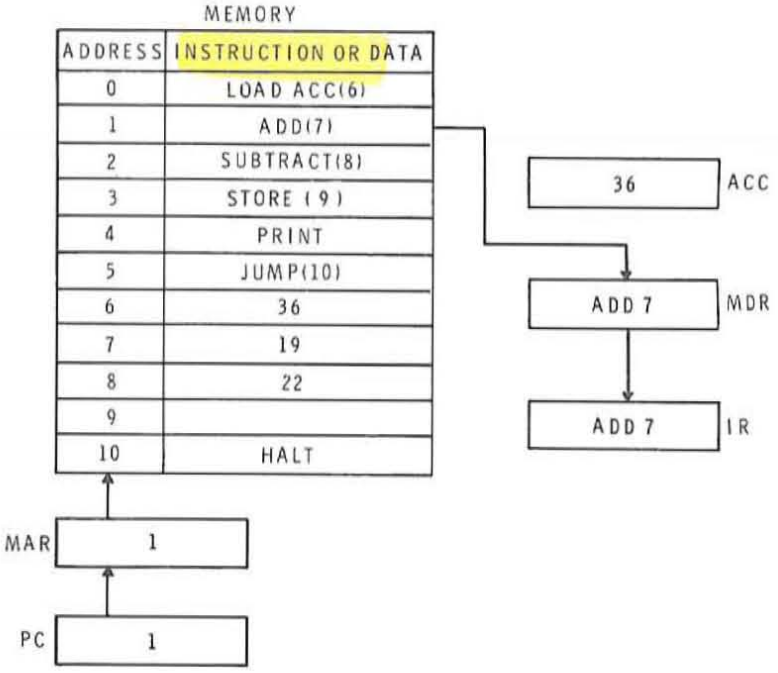
Figure 10-17B shows the execution of the first instruction. The first instruction LOAD ACC (6) tells us to load the accumulator with the data stored in memory location 6. In executing this instruction, the number 36 is transferred to the accumulator. Note how this is done. The address specified by the instruction word (6) is transferred from the instruction register to the memory address register (MAR). This causes the number 36 stored in that location to be transferred to the MDR and then to the accumulator. During this step, the program counter (PC) is incremented by one so the next instruction in sequence will be fetched.



(B) Execute first instruction (LOAD)

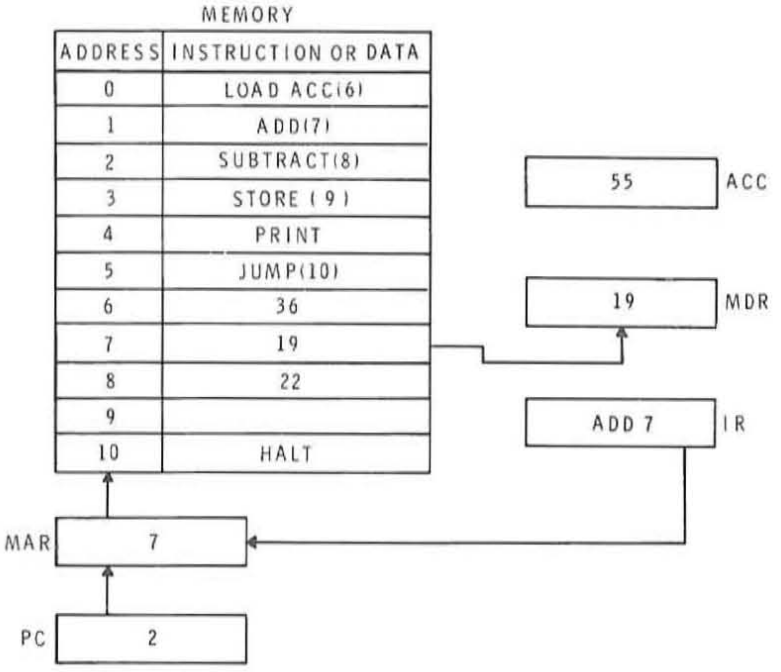
Figure 10-17C shows the fetch operation for the second instruction. The contents of the program counter is transferred to the MAR so that the ADD(7) instruction is fetched. This instruction passes through the MDR into the instruction register.

(C) Fetch second instruction (ADD)



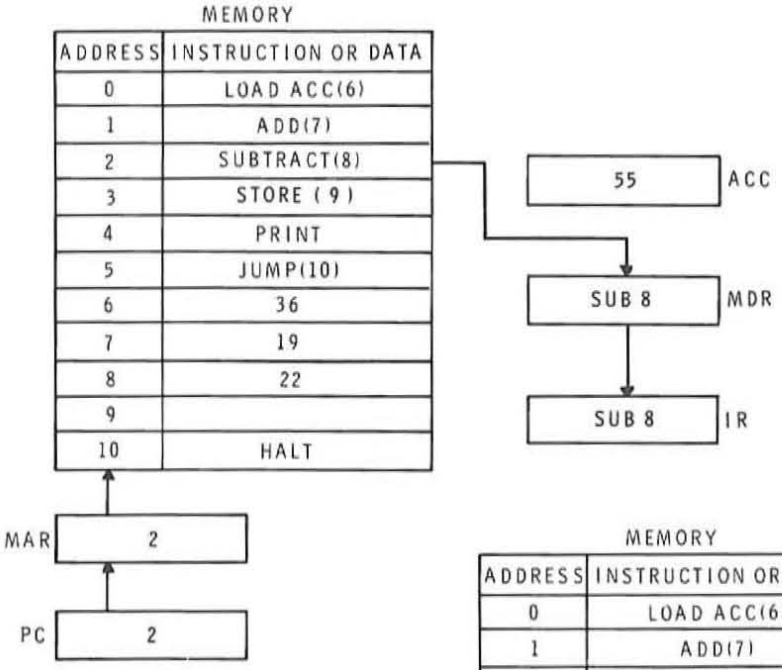
The execution of the add instruction is shown in Figure 10-17D. This instruction tells us to add the contents of memory location 7 to the contents of the accumulator. The address of the add instruction is trans-

(D) Execute second instruction (ADD)

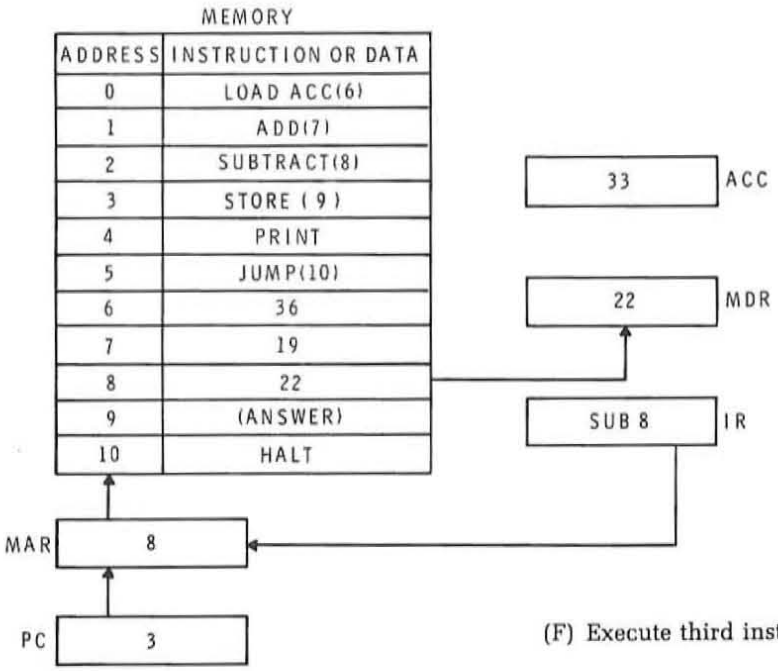


ferred to the MAR. This causes the contents of memory location 7, the number 19, to be transferred to the MDR. The contents of the MDR are added to the contents of the accumulator with the sum appearing back in the accumulator. As you can see, the sum of 36 and 19 is 55. Note that the program counter is again incremented so that the next instruction in sequence will be fetched.

The remaining instructions in the program are fetched and executed in a similar manner. The third instruction, a subtract, causes the memory contents of location 8 to be subtracted from the contents of the accumulator with the resulting remainder appearing in the accumulator. This produces an answer of 33. This fetch-execute sequence is shown in Figure 10-17E and F.



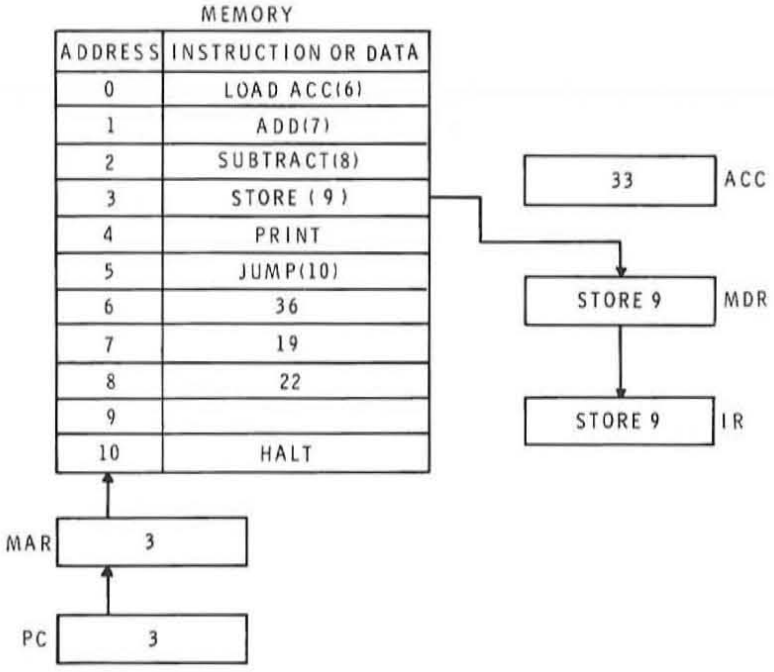
(E) Fetch third instruction (SUB)



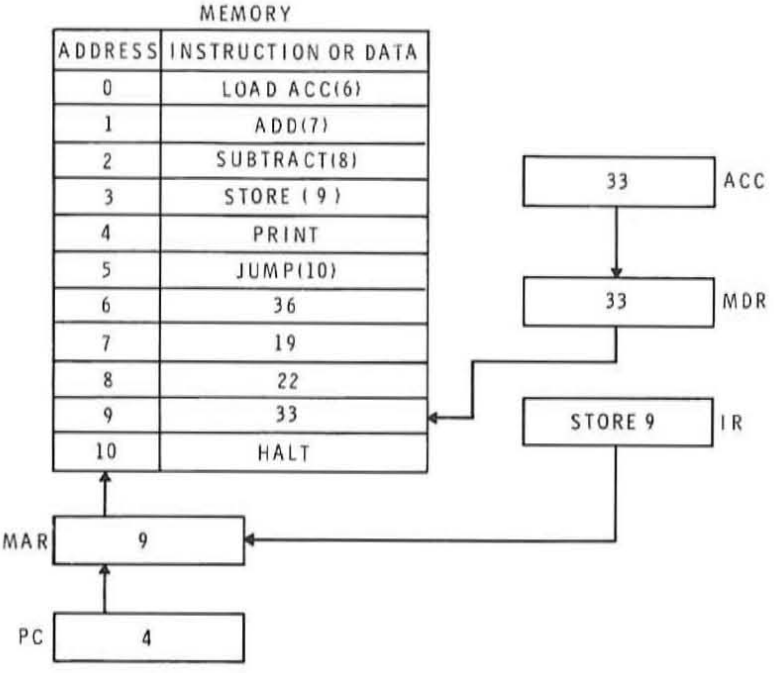
(F) Execute third instruction (SUB)

The next instruction in sequence, STORE(9), tells us to store the contents of the accumulator in memory location 9. The number 33 in the accumulator is transferred to the MDR and stored in location 9, as indicated in Figure 10-17G and H.

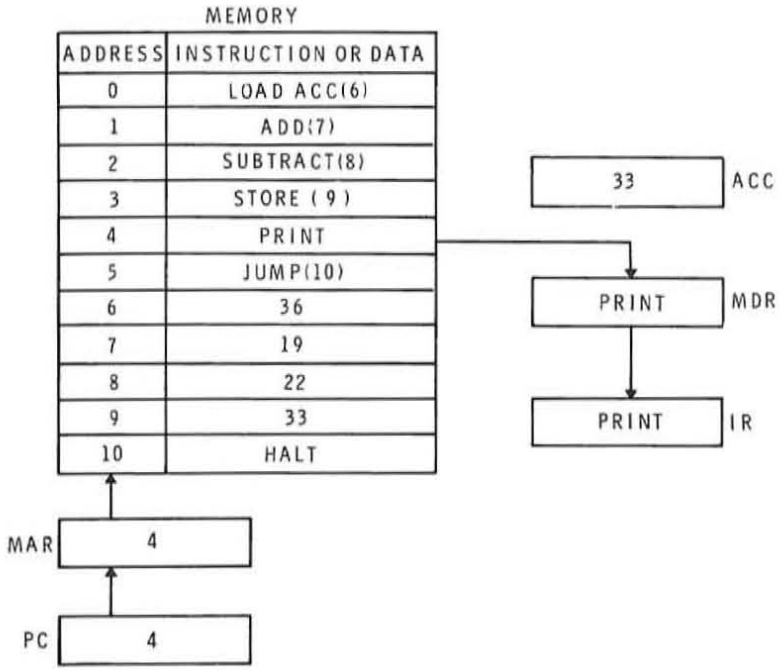
(G) Fetch fourth instruction (STORE)



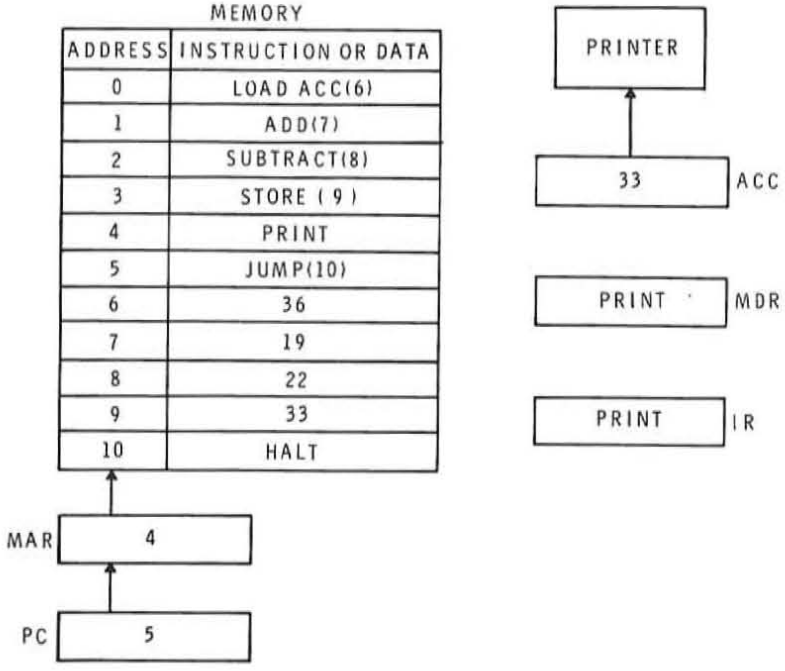
(H) Execute fourth instruction (STORE)



The fifth instruction in the program, PRINT, tells us to print the contents of the accumulator on the external printer. The number stored in the accumulator will then be transferred to a printer where it is printed. The fetch-execute cycle for this operation is shown in Figures 10-17I and J.

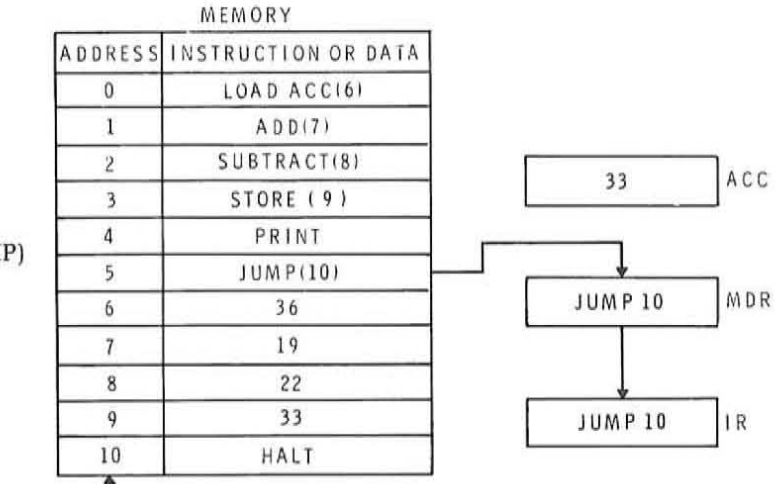


(I) Fetch fifth instruction (PRINT)

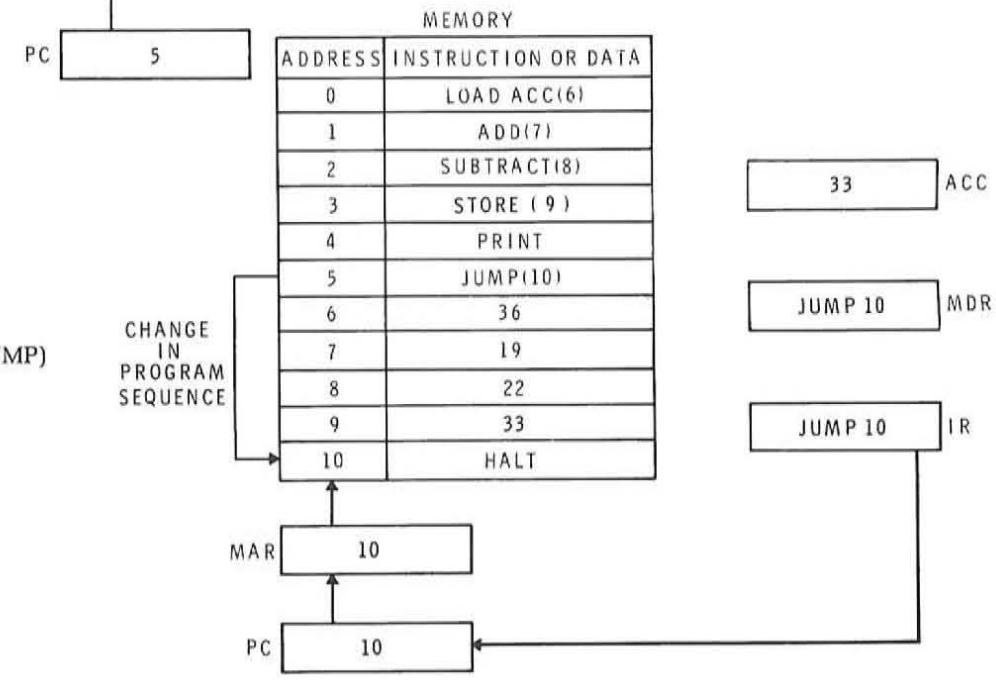


(J) Execute fifth instruction (PRINT)

The sixth instruction in sequence is a JUMP(10) instruction that causes the normal sequence of program executions to change. The jump instruction tells us not to execute the contents of the next memory location in sequence. Instead it tells us to take the next instruction from memory location 10. You can see by referring to Figure 10-17K that the contents of the next memory location in sequence (address 6) contains a data word. The computer, being a dumb machine, would simply interpret a data word as an instruction and attempt to execute it. If this ever happens, the result of the computation will be erroneous. The purpose of the jump instruction in our program is to jump over the data words in the program stored in locations 6, 7, 8, and 9. The program is continued in location 10, where a

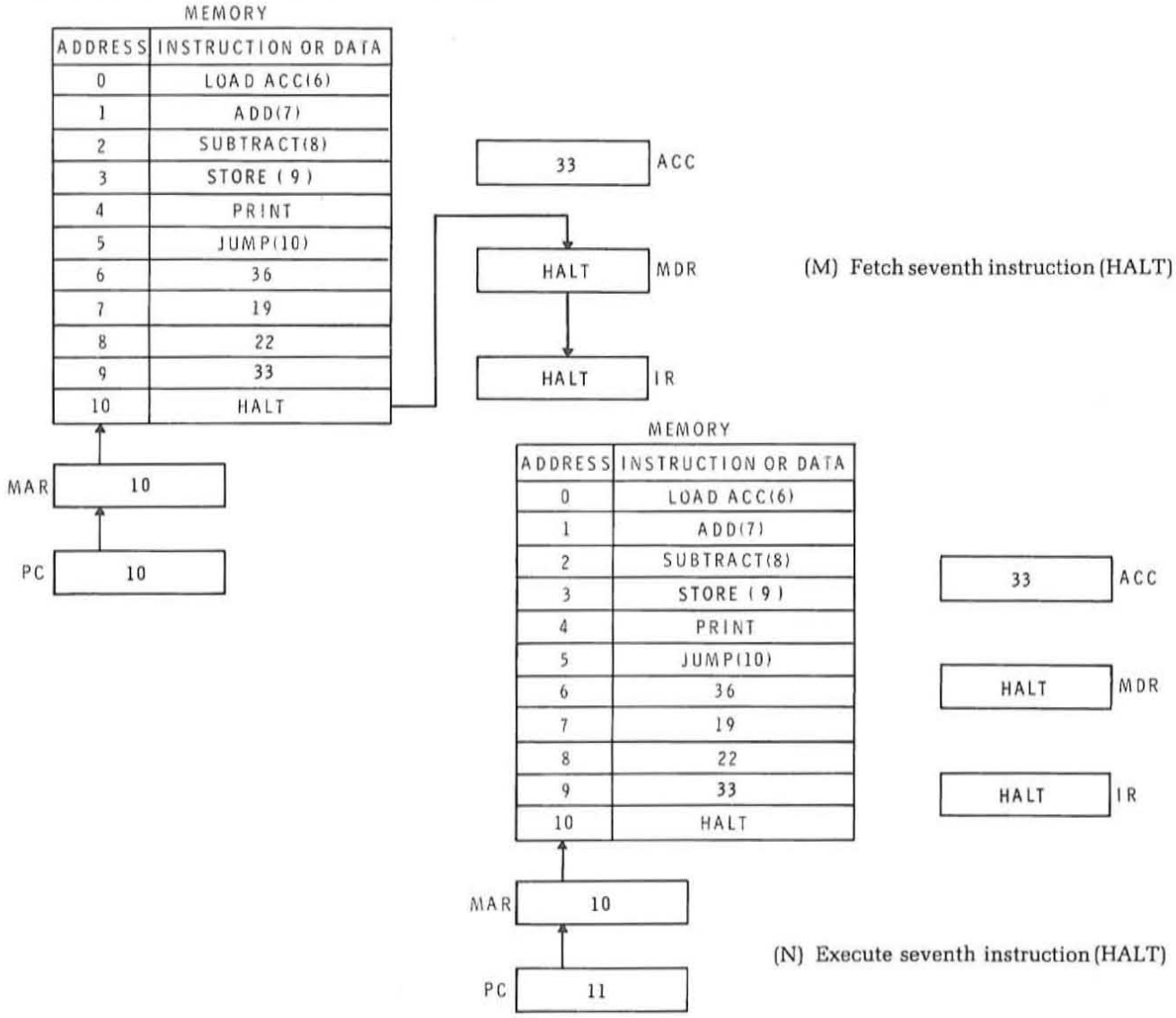


(K) Fetch sixth instruction (JUMP)



(L) Execute sixth instruction (JUMP)

HALT instruction is stored. By executing the jump instruction, the program counter is loaded with the address portion of the jump instruction (10) instead of being incremented as it normally is. See Figure 10-17L. This causes the computer to fetch and execute the instruction stored in location 10. This is illustrated in Figures 10-17M and N.



The last instruction in the program is a HALT. This instruction has no effect other than to stop the operation of the machine. Note in Figure 10-17N that the program counter was incremented so that it contains the memory location (11) of the next instruction in sequence to be fetched.

Study the program shown in Figure 10-17. Trace through each of the fetch and execute cycles for each instruction to be sure that you fully understand the operation. All digital computers operate in this same way with minor variations.

Self Test Review

32. The four major sections of a digital computer are:
    a. _____
    b. _____
    c. _____
    d. _____
33. The section of the computer that interprets the instructions is the:
    a. Memory
    b. Control
    c. ALU
    d. I/O
34. An 8-bit microprocessor has a 14-bit memory address word. What is the maximum number of memory words it can have?
    a. 256
    b. 4096
    c. 16,384
    d. 65,536
35. The address of the memory word indicates its
    a. Content
    b. Location
    c. Size
36. The main computational and data manipulation register in a computer is the
    a. MAR
    b. MDR
    c. IR
    d. Accumulator
37. What register indicates the location of the next instruction in sequence in a program?
    a. MDR
    b. IR
    c. PC
    d. Accumulator
38. The ALU and control unit combined are referred to as the _____.
39. Two numbers are to be added by the ALU. These numbers are initially stored in the _____ and _____ registers. The sum appears in the _____.
40. In carrying out a program, the computer repeats a series of _____ and _____ operations on the instructions in memory.

Answers

32. a. Memory
    b. Control
    c. Arithmetic-Logic
    e. Input/Output
33. b. Control
34. 16,384 = 2^{14}
35. b. Location
36. d. Accumulator
37. c. PC (Program Counter)
38. CPU or Central Processing Unit
39. MDR and Accumulator, Accumulator
40. Fetch, Execute

Programming

A digital computer without a program is a useless piece of electronic hardware. The logic circuitry making up the computer is incapable of performing any useful end function without a program. It is this characteristic of a digital computer that sets it apart from other types of digital circuitry. And it is this characteristic that makes the digital computer the versatile machine that it is. For this reason, a discussion of digital computers is not complete without information on programming.

The process of using a digital computer is mainly that of programming it. Whether the computer is a simple microprocessor or a large scale system, it must be programmed in order for it to perform some useful service. The application of the computer will define the program. Programming is the process of telling the computer specifically what it must do to satisfy our application.

Programming is a complex and sophisticated art. In many ways it is almost a field apart from the digital circuitry and the computer hardware itself. There are many different levels of programming and many unique methods that are employed. For that reason it is impossible to cover them all here. The purpose of this section is to give you an overview of the process of programming a computer. As indicated earlier, our emphasis will be on the programming of small scale digital computers such as the microprocessor.

Programming Procedure. There are many different ways to program a digital computer. The simplest and most basic form of programming is machine language programming. This is the process of writing programs by using the instruction set of the computer and entering the programs in binary form, one instruction at a time. Programming at this level is difficult, time consuming, and error prone. It also requires an in-depth understanding of the computer organization and operation. Despite these disadvantages, however, this method of programming is often used for short simple programs. While most computer applications do not use machine language programming, it is desirable to learn programming at this level. It helps to develop a thorough knowledge of machine operation and generally results in the shortest, most efficient programs. Many microprocessors and minicomputers are programmed in machine language.

To illustrate the concepts of programming in this unit, we will use machine language programming. Other more sophisticated methods of programming will be discussed later.

Programming a digital computer is basically a seven step process. These seven steps are: (1) define the problem; (2) develop a workable solution; (3) flow chart the problem; (4) code the program; (5) enter the program into the computer; (6) debug the program; (7) run the program. Let's take a look at each of these steps in detail.

The first and perhaps the most important step in programming a digital computer is defining the problem to be solved. The success of the program is directly related to how well you define the operation to be performed. There is no set standard for the problem defining procedure, and any suitable method can be employed. The definition can be a written statement of the function to be carried out, or it may take the form of a mathematical equation. In some cases, the problem may be more easily defined by graphical means. For control applications, the problem may be expressed in the form of a truth table. The form in which you place the definition is strictly a function of the application.

Once the problem is analyzed and defined, you can begin thinking in terms of how the computer may solve the problem. Remember that a computer program is a step-by-step sequence of instructions that will lead to the correct results. You should think in terms of solving your problem in some step-by-step sequential manner. What you will be doing in this phase of the programming procedure is developing an algorithm. An algorithm is a method of procedure of solving a problem.

An important point to remember is that there is usually more than one way to solve a given problem. In other words, there is more than one algorithm suitable for achieving the goal that you have set. Much of the job of programming is in determining the alternatives and weighing them to select the best suitable approach. The simplest and most direct algorithms are usually the best.

The next step is to flowchart the problem. A flow chart is a graphical description of the problem solution. Various symbols are used to designate key steps in the solution of the problem. Figure 10-18 shows the basic flowcharting symbols. An oval defines the starting and finishing points. A rectangular box defines each individual computational step leading to the solution. Each rectangle contains some basic operation or calculation that is to take place. The diamond shaped symbol represents a decision point. It is often necessary to observe the intermediate results in a problem solution and make a decision regarding the next step to be taken. There are generally two exits to the diamond shaped decision making symbol. These represent a yes or no type of decision.

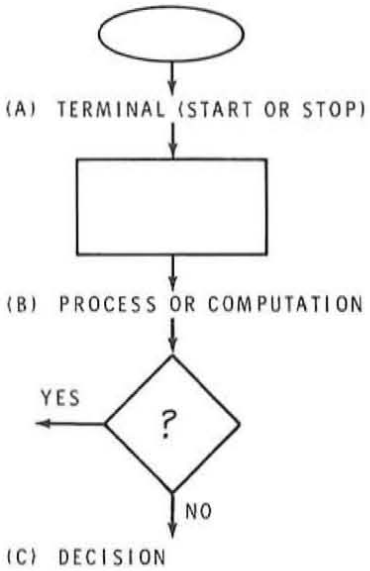


Figure 10-18
Basic flowcharting symbols.

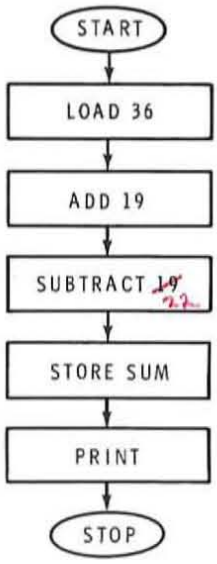


Figure 10-19
Flow chart for the problem 36 + 19 − 22 = 33 and print.

Figure 10-19 shows a simple flow chart for the problem solved in Figure 10-17. No decision was made in this program.

As you can see, the flow chart is a graphical representation of the basic method used to solve the problem. The flow chart permits you to visualize the algorithm you developed. In many cases, the flowcharting of a problem helps to determine the best approach to solving a problem since it forces you to think in a logical sequence and express the solution in a step-by-step form.

At this point in the programming procedure, your problem is quite well defined and a basic method of solving the problem has been determined. You are now ready to convert your flow chart and algorithm into a machine language program. This process is called coding. Coding is the procedure of listing the specific computer instruction sequentially to carry out the algorithm defined by the flow chart. This requires a familiarity with the instruction set of the computer you plan to use.

The next step in the programming procedure is to load the program into the computer memory. Once you have written the program with the computer instructions, you have all of the information necessary to enter that program into the computer memory. If you are dealing with machine language programming, you will convert the instruction words into their binary equivalents and then load them into the computer. If the program is a simple one, it can be loaded by using the binary switches on the front panel of the computer. However, for long, complex programs, this manual loading procedure is difficult and time consuming.

Most computers make it easy for the programmer to enter his program. Because of the availability and use of support programs residing within the machine, the program can usually be entered automatically. One of the most common ways of entering data into a computer is by the use of a keypunch machine. This is a typewriter-like machine that punches a standard computer card with the instructions to be entered. Teletype input/output machines using perforated paper tape are also commonly used for program entry. The instruction designations are typed on the machine and as they are typed, a paper tape is punched.

Once the cards or paper tape are punched, they are then fed into a tape reader or card reader and loaded into the computer memory. A special program residing within the computer memory called a loader causes the program to be loaded automatically.

With the program now in the computer memory, you can begin to run it. However, before you use it to obtain your final answer, it is often necessary to run through the program slowly a step at a time to look for programming errors and other problems. This process is called debugging. You test the program to see that it produces the desired results. If the program produces the correct result, it is ready to use. Often, programming mistakes are encountered and it is necessary to modify the program by changing the instruction steps. Often the entire program may be discarded and a new one written, using a different algorithm.

Once the program has been debugged, it is ready for use. With the program stored in memory, your problem can be solved. The computer is started and the desired results are produced.

Writing Programs

Before you can begin coding programs, you must be familiar with the instruction set of the computer you are using. Most digital computer instruction sets are basically alike in that they all perform certain basic functions such as addition, branching, input/output and the like. But each instruction set is different because the logic circuits unique to each computer carry out these operations in different ways. To code the program properly, you must know exactly what each instruction does. You can get this information by studying the instruction set as it is listed and explained in the computer's operation and programming manuals. By studying the instruction set you will learn how the computer is organized and how it operates. The insight you gain from this will be valuable to you not only in coding the program but also in developing the best solution to a problem with a given machine.

Computer Instructions. A computer instruction is a binary word that is stored in the computer memory and defines a specific operation that the computer is to perform. The instruction word bits indicate the function to be performed and the data which is to be used in that operation.

There are two basic types of computer instructions: memory reference and non-memory reference. A memory reference instruction specifies the location in memory of the data word to be used in the operation indicated by the instruction. A non-memory reference instruction simply designates an operation to be performed. Non-memory reference instructions generally refer to internal housekeeping operations to be performed by the computer and manipulations on data stored in the various registers in the computer.

Figure 10-20 shows typical instruction word formats for an 8-bit microprocessor. The format shown in Figure 10-20A is a memory reference instruction. The instruction is defined by three 8-bit words which are stored in sequenctial memory locations. The first 8-bit word is the op code or operations code which is simply a binary bit pattern specifying some operation. The second and third 8-bit words specify the memory address of the data or operand to be used. The 8-bit op code defines 256 possible operations or functions. It is the op code that designates the operation that is to be performed. The 16-bit address specifies the memory location of the data to be operated upon. The size of the address generally indicates the maximum memory size of the computer. With 16 bits of address information, $2^{16} = 65{,}536$ words can be directly addressed. We usually say that the maximum memory size is 65K.

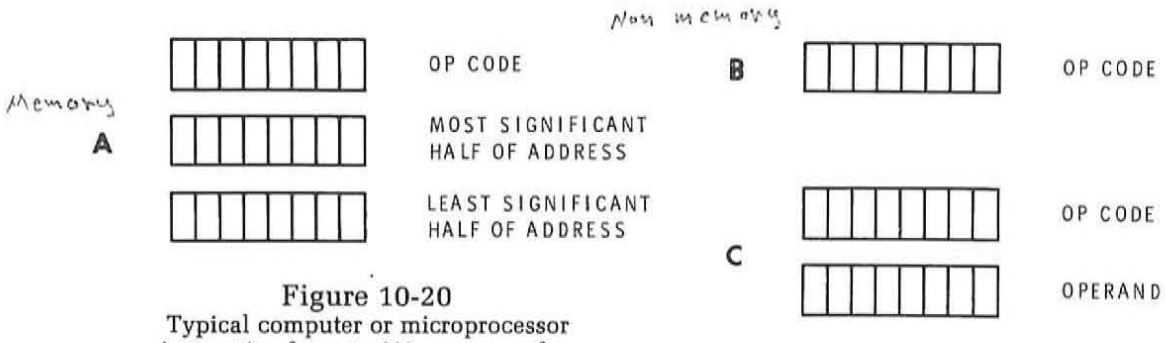


Figure 10-20
Typical computer or microprocessor instruction formats. (A) memory reference, (B) non-memory reference, (C) immediate.

The word format in Figure 10-20B is the typical format for non-memory reference instructions. Only an 8-bit op code is used. In this type of instruction, an address is not needed since we do not reference a location in memory where data is stored. Instead, the bits in this field are used to specify various operations that are to take place within the CPU. For example, such an instruction might call for the resetting (clearing) of a register or the transfer of data from one register to another. Certain types of input-output instructions have this format.

Another instruction type is the immediate instruction which is widely used in microprocessors. The format for this instruction is shown in Figure 10-20C. It consists of an 8-bit op code that specifies the operation. The second 8-bit part of this instruction is the data or operand to be used in the operation called for. The immediate instruction is like the memory reference instruction in that it specifies the use of some data word. The data to be used is in the instruction word itself rather than being referenced by an address in the instruction word. Immediate instructions save memory space and shorten instruction fetch and execution times.

Another method of classifying computer instructions is to group them according to the type of functions that they perform. These include arithmetic and logic, decision-making, data moving, and control. Let's consider each of these in more detail.

An arithmetic instruction defines a specific mathematical operation that is to take place. The most commonly used arithmetic instructions are add and subtract. In larger computers the multiply and divide functions are also included. Multiply and divide operations in smaller computers such as minicomputers and microprocessors are carried out by special subroutines. As an example, multiplication can be performed by repeated addition. Division can be programmed by the use of repeated subtractions. Arithmetic instructions are generally of the memory reference type.

Logical instructions specify digital logic operations that are to be performed on computer data. These include the standard logic functions of AND, OR and invert(complement). Many computers include the exclusive OR function. Other logic instructions include shift right and shift left operations. The AND, OR and XOR logical instructions are usually memory reference type. The shifting and inversion instructions are of the non-memory reference type as they generally refer to operations carried out on data stored in one of the computer's registers.

A decision making instruction is one that permits the computer to test for a variety of results and based upon these tests make a decision regarding the next operation to be performed. It is the decision-making instructions that set the computer apart from the standard calculator. Decision-making instructions allow the computer to automate its operations. A decision-making instruction generally follows a sequence of other instructions that perform some arithmetic or logical operation. Once the operation is performed, the decision-making instruction tests for specific results. For example, decision-making instructions test for positive or negative numbers, zero, odd or even numbers or equality. These tests are generally made on the data stored in various registers in the machine. If the test for a specific condition exists, the computer is usually instructed to deviate from its normal sequential execution of instructions. Jump or branch instructions are memory reference instructions that test for certain conditions and then specify a memory location where the next instruction to be executed is located. Skip instructions also change the computing sequence. These instructions test for a specific condition and then, if that condition exists, direct the computer to skip the next instruction in sequence. Skip instructions are non-memory reference types.

A data moving instruction is one that causes data words to be transferred from one location to another in the computer. It is these instructions that are used to take data from memory and load it into one of the operating registers in the computer. Other data moving instructions cause data stored in a register to be stored in a specific memory location. These are memory reference instructions. Other data moving instructions specify the transfer to data words between registers in the machine. These are non-memory reference instructions. The data moving instructions provide a flexible means of transferring data within the machine to prepare it to be processed as required by the application. A special class of data moving instructions are the input/output instructions. I/O instructions cause data to be transferred into and out of the computer. These non-memory reference instructions often specify one of several input/output channels or a specific peripheral device. Input/output operations can be programmed to take place through the operating registers of the machine, or in some computers, directly between the memory and the peripheral unit.

A control instruction is a non-memory reference instruction that does not involve the use of data. Instead, it designates some operation that is to take place on the circuitry in the computer. Clearing a register, setting or resetting a flip-flop or halting the computer are examples of control instructions.

A Hypothetical Instruction Set. A typical but hypothetical instruction set for a minicomputer or microprocessor is shown in Table I. Only a few of the most commonly used instructions are listed so that you can become acquainted with them quickly. Real instruction sets are far more extensive. Nevertheless, the instruction set in Table I is representative. We will use it to demonstrate the writing and coding of programs.

The instruction set in Table I can apply to the hypothetical computer described earlier or a typical microprocessor. For the instructions listed here, we assume that the computer has an 8-bit word length and 65K of memory. The accumulator and memory data registers are 8 bits in length. The program counter and MAR are 16 bits in length. I/O transfers take place through the accumulator. A single instruction may occupy one, two or three consecutive memory locations depending upon its format as shown in Figure 10-20. Study the instructions in Table I so that you will be familiar with the operation each performs. Note that each instruction is designated by a three letter mnemonic. The type of instruction is designated by the letters R (memory reference), N (non-memory reference), A (arithmetic-logic), T (data moving or transfer), D (decision) and C (control). Despite the simplicity of this instruction set, it can be used to program virtually any function.

HYPOTHETICAL COMPUTER INSTRUCTION SET

Table I

MNEMONIC	TYPE OF INSTRUCTION	OPERATION PERFORMED
LDA	R, T	Load the data stored in the specified memory location (M) into the accumulator register.
STA	R, T	Store the data in the accumulator register in the specified memory location (M).
ADD	R, A	Add the contents of the specified memory location (M) to the contents of the accumulator and store the sum in the accumulator.
SUB	R, A	Subtract the contents of the specified memory location (M) from the contents of the accumulator and store the remainder in the accumulator.
AND	R, A	Perform a logical AND on the data in the specified memory location (M) and the contents of the accumulator and store the results in the accumulator.
OR	R, A	Perform a logical OR on the data in the specified memory location (M) and the contents of the accumulator and store the results in the accumulator.
JMP	R, D	Unconditionally jump or branch to the specified memory location (M) and execute the instruction stored in that location.
JMZ	R, D	Jump to the specified memory location if the content of the accumulator is zero (reset). Execute the instruction stored in that location. If the accumulator is not zero, continue with the next instruction in normal sequence.
CLA	N, C	Clear or reset the accumulator to zero.

CMP	N, A	Complement the contents of the accumulator.
SHL	N, A	Shift the contents of the accumulator one bit position to the left.
SHR	N, A	Shift the contents of the accumulator one bit position to the right.
INP	N, T	Transfer an 8-bit parallel input word into the accumulator.
OUT	N, T	Transfer the contents of the accumulator to an external device.
HLT	N, C	Halt. Stop computing.
INC	N, C	Increment the contents of the accumulator.
DCR	N, C	Decrement the contents of the accumulator.
SKO	N, D	If the number in the accumulator is odd (LSB = 1), skip the next instruction and execute the following instruction. If the accumulator is even (LSB = 0), simply execute the next instruction in sequence.

Example Programs. The following examples illustrate the use of the instruction set in writing programs. The program description, flowchart and instruction code are given in each example. Study each program, mentally executing the instructions and imagining the outcome. The format of the instruction coding is shown below.

3 ADD (7)

The number on the left is the memory address. The mnemonic specifies the instruction. The number in parenthesis is the address of the operand called for by a memory reference instruction. This line of instruction coding tells us that in memory location 3 is an add instruction that tells us to add the content of location 17 to the content of the accumulator.

The program shown below is a repeat of the program given in Figure 10-17. The only differences are the memory location numbers of the instructions, the use of mnemonics, and the substitution of the OUT instruction for the PRINT instruction.

```
 0  LDA  (16)
 3  ADD  (17)
 6  SUB  (18)
 9  STA  (19)
12  OUT
13  JMP  (20)
16  36
17  19
18  22
19  ANSWER
20  HLT
```

The difference in memory addresses is the result of the assumption that our computer uses an 8-bit word and that memory reference instructions occupy three sequential memory locations. In the program of Figure 10-17 we assumed that one memory address contained one instruction. In the program above, the LDA (16) occupies memory locations 0, 1 and 2. The op code is in 0, the most significant part of the address (0000 0000) is in location 1, and the least significant part of the address (0001 0000) is in location 2. The ADD, SUB, STA and JMP memory reference instructions each occupy three sequential locations. The OUT and HLT instructions do not reference memory so they occupy only a single location.

The program below illustrates the use of the logical instructions.

Assume that the only logical instructions that your computer has are AND and CMP (complement). We need to perform the OR function on the two words A and B stored in locations 16 and 17. The flow chart in Figure 10-21 and the program below illustrates how this is done. By DeMorgan's theorem we know that $\overline{A+B}=\overline{A}\,\overline{B}$. Complementing both sides gives us the OR function.

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

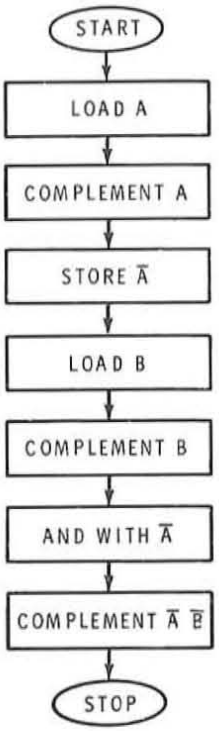


Figure 10-21

Flow chart illustrating a method of performing the OR function with AND and invert instructions.

It is the right-hand part of this equation that is our algorithm.

```
 0  LDA (16)
 3  CMP
 4  STA (18)
 7  LDA (17)
10  CMP
11  AND (18)
14  CMP
15  HLT
16  A
17  B
18  Intermediate storage of Ā
```

Note the use of memory location 18 as temporary storage for an intermediate result ($\overline{A}$). This frees the accumulator to process other data.

The next program illustrates several important concepts. First, it shows how the computer makes decisions. Second, it demonstrates the use of a program loop. A loop is a sequence of instructions that is automatically repeated. The sequence is executed once and a jump instruction causes the program to branch back (loop) to the beginning of the sequence and repeat it again.

The program below is designed to enter two 4-bit BCD numbers and store them in a single 8-bit memory location. The desired memory format is shown in Figure 10-22. The BCD digits are entered, one at a time, into the four least significant bit positions of the accumulator as shown in Figure 10-23. The first digit entered must be moved to the four most significant bit positions of the accumulator before the second digit can be entered. This is accomplished with a series of shift left instructions.

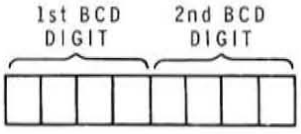


Figure 10-22
Memory format for two BCD digits.

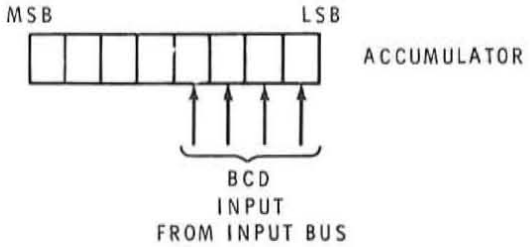


Figure 10-23
Loading the accumulator from the data bus.

The flow chart in Figure 10-24 shows how this is accomplished. The program is given below.

```
 0  CLA
 1  INP
 2  SHL
 3  STA (25)
 6  LDA (24)
 7  DCR
 8  STA (24)
11  JMZ (20)
14  LDA (25)
17  JMP (2)
20  LDA (25)
21  INP
22  STA (26)
23  HLT
24  4
25  Temporary storage
26  Storage for two BCD digits
```

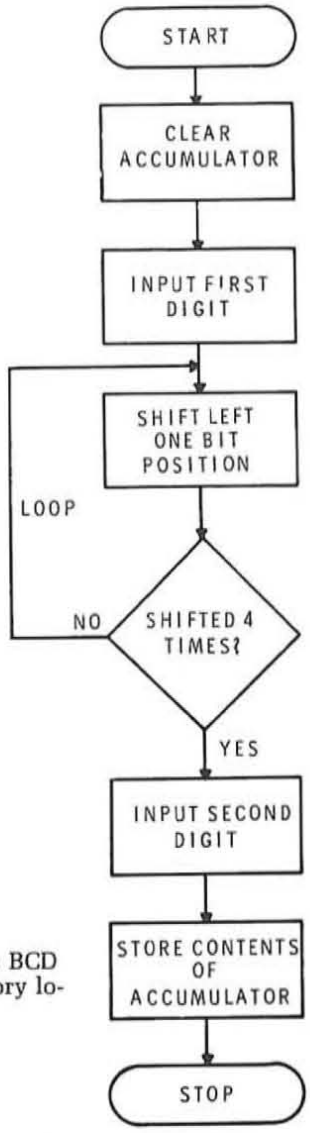


Figure 10-24
Flow chart showing how two BCD words are stored in one memory location.

The first instruction (CLA) clears the accumulator. The second instruction (INP) loads the first BCD digit. This digit is then shifted left one bit position. We need to shift it four positions to the left. The sequence of instructions in locations 2, 3, 6, 7, 8, 11, 14 and 17 form a loop and a decision-making test to accomplish this. Stored in memory location 24 is a number that tells us how many times to shift. That number is loaded into the accumulator decremented by one and restored each time a shift occurs. We test that number with a jump on zero instruction (JMZ). When the BCD digit has been shifted four times, the number in location 24 has been reduced to zero. The JMZ instruction detects this condition and branches the program to location 20 where the data word is retrieved from temporary storage and the second BCD digit is inputted.

Let's consider the loop and decision-making process in more detail. After the first input digit is loaded, it is shifted left once. We then store it temporarily in location 25. This is to prevent loss of the data while we are in our decision-making loop. Next, the number of desired shifts is loaded into the accumulator. We decrement it by one, indicating that we have shifted left once. Next, we restore this number (now 3) in location 24. This number, which is still in the accumulator, is now tested with the JMZ instruction. At this time the accumulator is not zero so the program does not branch. Instead, the next instruction in sequence is executed. This is a load accumulator instruction that retrieves the data words which we temporarily stored in location 25. Then the JMP instruction is executed. This instruction returns us to location 2 to produce another shift. It is the jump instruction that creates the loop.

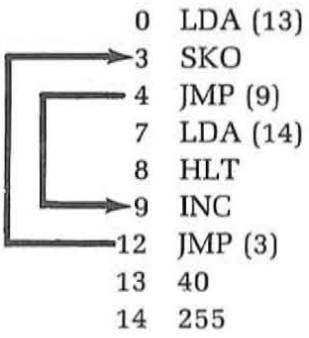
The loop is then repeated three more times. On the fourth pass through the loop, the number in location 24 is decremented to zero. The JMZ instruction detects this condition and causes the program to branch to location 20. We have now escaped from the loop. Next we reload the shifted data from location 25. Finally, we load the second BCD digit. Both digits are now in the accumulator so we can store them in location 26 with the STA (26) instruction. The program is now complete and the HLT instruction terminates it.

These examples show how a computer performs its work. It does it laboriously, one step at a time. The only thing that makes it practical is its high speed operation. With each instruction taking only microseconds, even long complex programs are executed quickly. To an operator, the execution appears almost instantaneous.

Self Test Review

41. An algorithm is a
    a. Flow Chart
    b. Program
    c. Procedure for solving a problem
    d. Decision-making step

42. The content of the accumulator is 10111010. The CMP instruction is executed. The new accumulator content is _____.
43. Which instruction would you use to down-count the accumulator?
    a. DCR
    b. INC
    c. SUB
    d. CLA
44. The content of the accumulator is 45. A JMZ (34) instruction in location 25 is then executed. The next instruction executed is in location
    a. 0
    b. 28
    c. 34
    d. 45
45. Program loops are implemented with the _____ and _____ instructions.
46. The number 0110 0101 is stored in the accumulator. The number in memory location 18 is 1111 0000. The AND (18) instruction is executed. The content of the accumulator becomes
    a. 0110 0101
    b. 1111 0000
    c. 1111 0101
    d. 0110 0000
47. Study the program below. At the completion of the program, the content of the accumulator is:
    a. 14
    b. 40
    c. 41
    d. 255



Answers

41. c. An algorithm is a step-by-step procedure for solving a problem.
42. 01000101
43. a. DCR
44. b. 28 The JMZ (34) instruction tests for a zero accumulator. The accumulator content is 45, therefore the program does not branch to location 34. Instead, it executes the next instruction in sequence which begins in location 28. Remember that the JMZ (34) instruction and its reference address occupies locations 25, 26 and 27.
45. JMP, JMZ
46. d. 0110 0000 Consider the two words to be ANDed as inputs to an AND gate as they would appear in a truth table. Then AND each corresponding pair of bits to get the result.
47. d. 255 This program uses the skip on odd accumulator instruction (SKO) to test the content of the accumulator. It first loads the content of location 13 into the accumulator. This is the number 40. The SKO then tests for an odd condition by monitoring the LSB. Since 40 is even, the program does not skip. The next instruction in sequence is executed. This is a JMP (9) instruction which causes the program to branch to location 9. Here the INC instruction is executed. The accumulator then becomes 41. The next instruction JMP (3) loops the program back to the location 3 where the SKO instruction again tests the accumulator. This time the content is odd. The instruction in location 4 is now skipped and the next in sequence is executed. This is an LDA (14) which loads 255 into the accumulator. The program then halts.

Software

The steps that we have just described make up the procedure typically used in developing application programs for the digital computer. The program may be solving a mathematical equation, sorting and editing a large volume of data, or providing some type of automatic control to an external machine. These application programs fall into a larger category of computer programs called software. Software is a broad general term used to describe all of the programs used in a digital computer. Besides the specific applications programs, there are many special programs supplied by the computer manufacturer which are used to simplify and speed up the use of the computer. These support programs eliminate much of the drudgery from programming and using a computer. It was determined early in the development of the digital computer that the computer itself with special internal control programs could assume much of the responsibility for the detailed translation of a problem into the binary language of the computer.

Subroutines. Many digital computer manufacturers supply what are called software libraries of subroutines and utility programs. A subroutine is a short machine language program that solves a specific problem or carries out some often used operation. For example, typical subroutines in many minicomputers and microprocessors are the multiply and divide programs. Instead of using the multiply subroutine each time it is required in a problem, the subroutine is stored in the computer memory only once. This saves a substantial amount of memory space. Each time the multiply subroutine is required, a jump instruction in the program causes the program to branch to the multiply subroutine. Once the multiplication has been performed, the computer jumps back to the normal program sequence.

There are many different types of commonly used subroutines. Multiplication and division are two of the most commonly used. Other subroutines include binary to BCD and BCD to binary code conversions. To communicate with external peripheral devices which use the decimal number system and the alphabet, a code such as ASCII is used. Data is entered into the computer in the ASCII code. In order for the computer to process this data, it must first be converted into pure binary numbers. Solutions to computer programs are in the pure binary form. A subroutine is used to convert the binary numbers into the ASCII format and then they are sent to an external peripheral device such as the printer.

Utility Programs. Utility programs refer to the short routines used to run the computer. Input/output programs for specific types of peripheral devices fall into this category. A loader is another utility program. This is a short sequence of instructions that allows data to be loaded into the computer. In order to operate, a computer must be programmed. But to load a program into the computer automatically requires that the loader program exist in the memory to begin with. Such loader programs are often entered manually from the computer front panel. The short loader program in memory then permits longer, more complicated programs to be loaded automatically.

There are many different types of utility programs used in digital computers. These also include editor programs for manipulating data and moving it from one part of the memory to another. These programs also permit you to conveniently modify a program through a peripheral device such as a teletypewriter. Utility programs often include diagnostic programs which provide a means of testing all of the functions of the computer. Diagnostic routines test each computer instruction and all memory locations. Other diagnostic programs permit peripheral devices to be exercised and their operation verified.

Assembler. The most sophisticated software supplied with most computers are large complex conversion programs called assemblers and compilers. These programs allow the computer to be programmed in a simpler language. Machine language programming is completely impractical for many modern applications. In order to simplify computer programming and remove the necessity for a dependence upon a knowledge of binary numbers and the computer architecture, computer manufacturers have developed easier methods of programming the computer. These methods involve higher level languages, which are special systems for speeding up the programming process. The higher level language permits someone with no computer expertise whatsoever to use the computer. The higher level language permits the programmer to express his problem as a mathematical equation or in some cases as an English language statement. These equations and statements are then fed to the computer, which then automatically converts them into the binary instructions used to solve the problem.

The simplest form of higher level programming language is called assembly language. This is a method of programming the computer an instruction at a time as you do in machine language programming. However, instead of binary designations for each instruction, short multiletter

names called mnemonics are given to each of the computer instructions. These are then written sequentially to form the program. Mnemonics are also given to memory addresses to avoid the use of specific memory locations.

Once the computer program is written in the assembly language, it is then entered into the machine along with an assembler program. The assembler program resides in the computer memory and is used to convert the mnemonics into the binary instruction words that the computer can interpret. As you can see, the assembler is a program that eliminates the necessity of dealing with binary numbers in the digital computer. However, since the machine is still programmed an instruction at a time, it provides wide flexibility in solving a given problem.

Compiler. The compiler, like the assembler, is a complex conversion program that resides in the computer memory. Its purpose is to convert a simplified statement of the program into the binary machine code that the computer can understand. The difference between the compiler and the assembler is that the compiler is capable of recognizing even simpler problem statements.

In one type of compiler programming language known as Fortran, the program can be written as an algebraic equation. This algebraic equation is then entered into the computer through a teletypewriter or via punched cards. The compiler program then analyzes the formula and proceeds to construct a binary program to solve this equation at some location in memory.

Another higher level programming language known as Cobol uses English language statements to describe the problem. These English language statements are punched into cards and then read into the computer memory. The compiler interprets them and converts them into the binary program. Unlike an assembly language program which has a one-to-one correspondence of instruction steps with machine language, a single compiler language program statement often causes many binary instructions to be generated.

There are many different types of higher level programming language used with computers. All of them have the prime function of simplifying the programming procedure. They greatly speed up and expedite the communications with the computer. They allow anyone who is capable of defining his problem to use the computer.

Cross Assemblers and Compilers. There are several special types of higher level programming language that have been developed to aid in programming microprocessors. For simple applications, microprocessors are programmed at the machine language level. However, when longer or more complex programs are required, it is desirable to use an assembler or compiler if it is available. For minicomputers and larger scale computers, compilers and assemblers that reside within the computer memory itself are available to aid in the programming process. However, most microprocessors do not have sufficient memory to accommodate such large complex programs. In addition, the microprocessor is generally to be dedicated to a specific application and therefore its memory will only be large enough to hold the application program required. In order to simplify the development of programs for use in a microprocessor, special programs called cross-assemblers and cross-compilers have been developed. These are special programs that reside in the memory of a larger general purpose digital computer. The application programs are written in these higher level languages and the larger machine then converts the application program into the binary machine language required by the microprocessor. The output of the larger scale computer is generally a paper tape containing the binary program, which is later loaded into the microprocessor memory.

Some of the larger more sophisticated microprocessors have been used as the primary component in a microcomputer that can be used as a software development system for that microprocessor. A large random access memory is added to the microprocessor along with appropriate peripheral devices. Resident assembler programs have been developed for these machines. In this way, the microcomputer based on the microprocessor can be used to develop application programs that will be used later in another system employing the same microprocessor.

Self Test Review

48. The program used to enter a program into the computer memory is called a:
    a. subroutine
    b. loader
    c. compiler
    d. assembler
49. The program used to convert an instruction-by-instruction mnenomic program into binary machine language is called a(n):
    a. subroutine
    b. loader
    c. compiler
    d. assembler
50. A program used to compute the square root of a number would be classified as a:
    a. subroutine
    b. loader
    c. assembler
    d. utility
51. A knowledge of binary numbers and computer operation is not required if the computer can be programmed in a higher level compiler language.
    a. True
    b. False

Answers

48. b. loader
49. d. assembler
50. a. subroutine
51. a. True

Microprocessors

As we indicated earlier, a microprocessor is the simplest and least expensive form of digital computer available. However, now we want to be more specific. In this section, we are going to discuss exactly what a microprocessor is, the types that are available, and how they are used.

Types of Microprocessors. Most microprocessors are the central processing unit (CPU) of a digital computer. That is, the microprocessors usually contain the arithmetic-logic and control sections of a small scale digital computer. Most of these microprocessors also contain a limited form of input-output circuitry which permits them to communicate with external equipment. To make the microprocessor a complete computer, external memory and input-output devices must be added. An external read only memory is normally used to store the program to be executed. Some read/write, random access memory may also be used. The external input-output circuitry generally consists of registers and control gating that buffer the flow of data into and out of the CPU.

Microprocessors come in a wide variety of forms. However, the most popular and widely used microprocessor is a MOS LSI circuit. These circuits are made with both P-channel or N-channel enhancement mode MOS devices. The entire CPU is contained on a single chip of silicon and mounted in either a 16, 24 or 40-pin dual in-line package. Such microprocessors are available with standard word lengths of 4, 8 and 16 bits. Other more sophisticated types of microprocessors are contained within two or more integrated circuit packages. When combined, they form a complete, small scale digital computer.

While most microprocessors are of the single chip MOS variety, there are numerous bipolar microprocessors available. These are inherently faster than the MOS devices but occupy more chip space and consume more power. Where high speed is required, these bipolar devices can be used. A recently developed integrated circuit technology, referred to as integrated injection logic (I^2L), combines both the speed of bipolar devices and the high density characteristics of MOS devices. These new I^2L LSI circuits offer many benefits, and their potential for microprocessor applications is great.

Microprocessors can also be constructed with the standard TTL or ECL integrated circuits. Standard MSI packages can be combined to construct a small CPU. Figure 10-25 shows a computer of this type. While this kind of microprocessor takes more circuitry and consumes more power, it generally offers several advantages. First, the microprocessor can be constructed to execute a special instruction set designed specifically for the

application. With a standard off-the-shelf CPU, the instruction set is fixed. Special instruction sets are often necessary for some applications and they can be readily optimized with a special TTL or ECL microprocessor design. Another advantage of a MSI TTL or ECL microprocessor is high speed. A standard MOS microprocessor may be too slow for the application. The fastest available MOS microprocessor can execute a single instruction in approximately 2 microseconds. The simpler and less sophisticated MOS microprocessors have instruction execution speeds in the 10 to 50 microsecond region. With a special TTL or ECL MSI microprocessor, execution speeds in the nanosecond region are easily obtained.

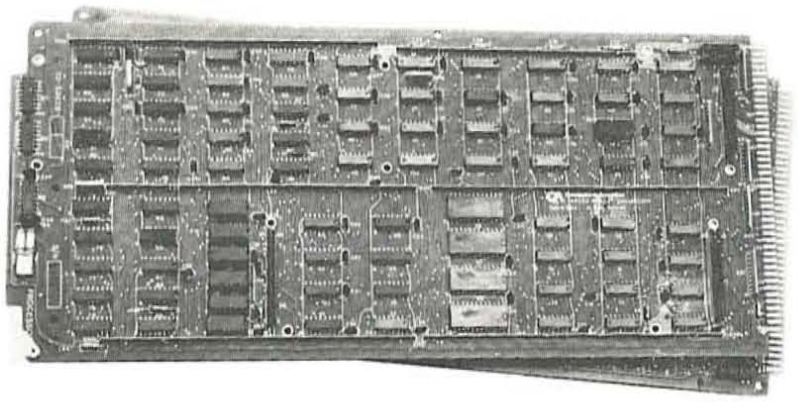


Figure 10-25
A microprocessor made with TTL MSI and SSI integrated circuits. This machine is more powerful than the typical LSI microprocessor but less powerful than a full minicomputer. (Photo courtesy Computer Automation Inc.).

In order to use a standard single chip microprocessor, some form of external memory must be used. The program to be executed by the microprocessor is generally stored in a ROM. Data is stored in RAM. Other external components needed to support a microprocessor are an external clock circuit, input-output registers, and peripheral devices.

ROM
RAM

All single chip microprocessors incorporate a data bus through which all external data transfers take place. This may be a 4 or 8-bit bi-directional bus over which all data transfers between the memory and input-output devices communicate with the CPU. A bus design of this type greatly minimizes the number of interconnections required to connect the microprocessor to the external devices. The limiting factor of such interconnections is the number of pins on the integrated circuit package. The bus organizations keeps the pin count to a minimum, but at the same time requires time sharing of the bus. Since all data transfers between the memory and CPU and between the CPU and the peripheral devices must use the same input-output lines, each operation must take place at a different time.

The input-output devices used with most microprocessors are quite different from those used with larger digital computers. Most larger computers are connected to input-output devices like CRT terminals, teletypewriters, paper tape readers and punches, card readers and line printers. On the other hand, microprocessors are interfaced to devices such as keyboards, 7-segment LED displays, thumbwheel switches, relays, analog to digital and digital to analog converters, temperature sensors, and other such components.

Applications of Microprocessors. Microprocessors are used primarily for dedicated functions. Rarely are microprocessors used to implement a general purpose digital computer. The program of a microprocessor is usually stored in the read only memory. This means that the program is fixed and dedicated to the specific application.

There are two broad general applications for modern LSI microprocessors. They can be used as replacements for minicomputers or as replacements for random hard-wired logic. The development of the minicomputer enabled many engineers to design digital computers into special control systems. The minicomputer was dedicated to the control application and its programmable flexibility offered many benefits. But its cost was very high. Some microprocessors have nearly as much computing power and capability as a minicomputer and can replace the minicomputer in many systems. A microprocessor has the advantage of smaller size, lower cost, and lower power consumption.

Another common use for the microprocessor is as an alternative to standard hard-wired digital logic circuits. Equipment customarily constructed with logic gates, flip-flops, counters, and other SSI and MSI circuits can often be implemented with a single microprocessor. All of the standard logic functions such as Boolean operations, counting and shifting can be readily carried out by the microprocessor through programming. The microprocessor will execute instructions and sort subroutines that perform the same logic functions.

Many benefits result from using the microprocessor in replacing hard-wired random logic systems. Some of these advantages are: (1) reduced development time and cost; (2) reduced manufacturing time and cost; (3) enhanced product capability; (4) improved reliability.

Development time and cost can be significantly reduced when a microprocessor is used. The design procedures used with standard logic circuits are completely eliminated. Much of the breadboarding, cut-and-try and prototype construction is completely eliminated. Design changes can be readily incorporated and new functions implemented by simply changing the program. With a microprocessor, the logic and control

functions are implemented with programs. The program can be written and entered into memory and then tested. System changes are easy to make by simply rewriting the program. Unique functions can be readily added by increasing the size of the program. In many cases, the system can be made self-checking by programming special diagnostic routines. Development time is further reduced because usually a single integrated circuit microprocessor replaces many other integrated circuits. This reduces wiring and interconnections and simplifies printed circuit board layout. Often the printed circuit board will be significantly smaller with a microprocessor system. Power comsumption and cooling are also usually simplified. The benefit of reduced development time and cost, of course, is that the product can come to market or be applied sooner.

Manufacturing costs are also reduced as a result of replacing random logic with a microprocessor. Fewer integrated circuits and smaller printed circuit boards are required to construct the system. Therefore, less time and materials are required to assemble the equipment. The programmed nature of the microprocessor system also makes it easier to test and debug than an equivalent hard-wired system.

Enhanced product capability is another benefit of using the microprocessor to replace hard-wired logic.The power of a digital system implemented with a microprocessor is limited strictly by the imagination of the designer. Many unique features and capabilities can be incorporated into the design by simply adding to the program. The incremental cost for adding such features to a microprocessor system is small compared to that of a hard-wired logic system. The ROM used to store the program usually contains extra room for program additions. Therefore, it is very easy to add special features. Many of these special or unique features would be difficult to incorporate in a random hard-wired logic design because of the extra design time, the complexity, and the additional cost. When a microprocessor is used, no additional parts or significant amount of design time are required to add them. The more unique and special features that a product can incorporate the better it performs and the more competitive it will be in the marketplace.

Another benefit of using the microprocessor to implement digital systems is increased reliability. Whenever the number of integrated circuits and wiring interconnections are reduced in a system, reliability increases significantly. Most system failures result from the failure of an integrated circuit or from an interconnection. The number of integrated circuits and interconnections are reduced many orders of magnitude in going from a standard hard-wired logic system to a microprocessor system. Increased reliability means fewer failures and leads to a corresponding reduction in both warranty and service costs.

The benefits of using a microprocessor are so significant that they will soon replace most random hard-wired logic designs. But the biggest present disadvantage of using a microprocessor is the designer's lack of programming knowledge. Very little circuit or logic design is required to implement a system with a microprocessor. Instead, the primary skill required is digital computer programming. Most engineers and digital designers were not trained in this subject, and therefore, initial design attempts with microprocessors may be slow and frustrating. However, as microprocessors are more widely used and their benefits recognized, engineers and designers will learn programming and begin to implement their systems with these devices.

Where are Microprocessors Used? There are so many applications for microprocessors that it is difficult to classify and list them. However, to give you a glimpse at the many diverse uses for these devices, consider some of the applications where they are now being used or being considered.

1. Electronic Cash Registers
2. Electronic Scales
3. Electrical Appliance Controls
4. Automotive Controls
5. Traffic Signal Controllers
6. Machine Tool Controls
7. Programmable Calculators
8. Automatic Test Equipment
9. Data Communications Terminals
10. Process Controllers
11. Electronic Games
12. Data Collection

These are only a few of the many applications presently implemented with microprocessors. Just keep in mind that the microprocessor can be used in any other application where hard-wired standard logic systems are now used. In addition, microprocessors can also be used as the CPU in a small general purpose microcomputer or minicomputer.

A typical application for a microprocessor is illustrated in Figure 10-26. Here the microprocessor is used to implement an electronic scale for use in a grocery market. The item to be weighed is placed on the scale. A transducer and analog-to-digital converter convert the weight into a binary word that is read into the CPU under program control. A clerk enters the price per pound via the keyboard. This too is read into the CPU. Then the CPU computes the price by multiplying the weight by the

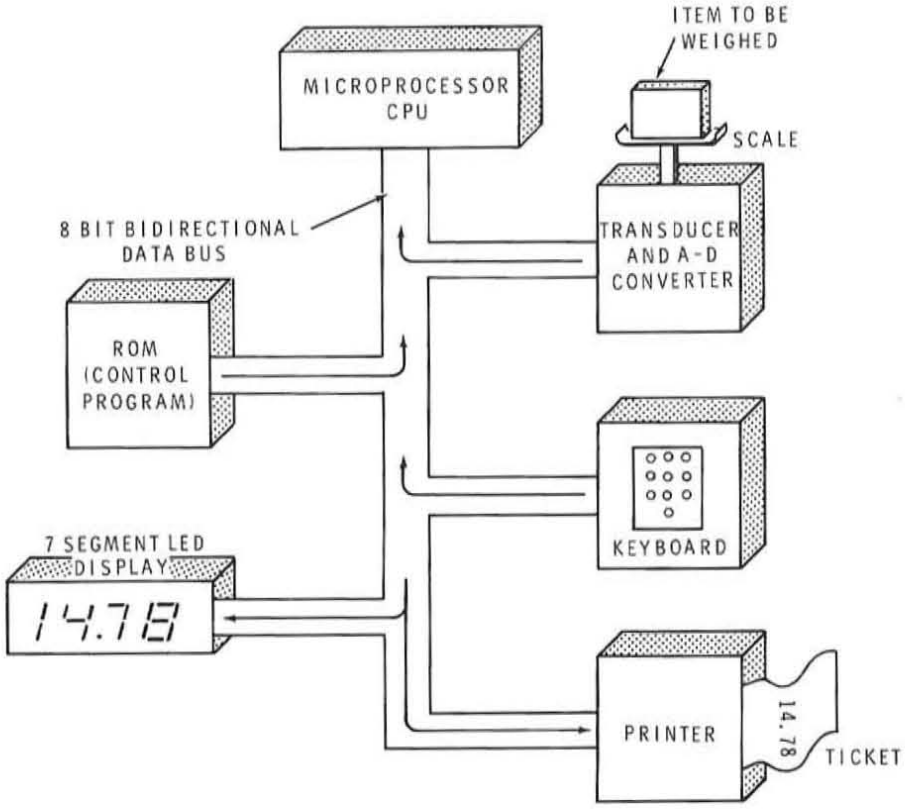


Figure 10-26
Typical application of a microprocessor in an electronic scale.

price per unit of weight. Then the total price is displayed on a 7-segment LED readout and printed on a ticket. All of this takes place under the control of the dedicated program stored in the ROM. Note the single 8-bit bi-directional data bus over which all data transfers take place.

Designing with Microprocessors. As indicated earlier, microprocessors can be used in two general ways. First, they can be used to replace minicomputers for dedicated control functions. Second, microprocessors can be used to replace standard hard-wired random logic systems. This section provides you with some guidelines to help you decide when and where a microprocessor should be used.

Microprocessors are generally much slower and less sophisticated than the typical minicomputer. But despite these limitations, microprocessors can often be used to replace minicomputers in some systems. The reason for this is that most minicomputers used in control systems are not used to their full capability. In a sense, they are a case of over-kill. Many control systems used the minicomputer simply because of the ease with which the control can be changed by modifying the program. The significantly higher cost has been traded off for the convenience of system modification. In these applications, the microprocessor can usually handle the control functions as well as the minicomputer. A careful study must be made in such designs to see when a microprocessor can replace a minicomputer. There are many trade-offs to consider (speed, cost, etc.). Keep in mind that microprocessor development is in its infancy. Many technological improvements will be made over the years causing the microprocessor to futher approach the capabilities of today's minicomputer.

The microprocessor is a design alternative which should be considered in the early design stages of any digital system. The benefits of a microprocessor over standard hard-wired designs is significant in the larger, more sophisticated digital systems. As a general guideline, a microprocessor can be used beneficially if it will replace from thirty to fifty standard MSI and SSI TTL integrated circuits. If a preliminary design indicates that this many TTL integrated circuits must be used a microprocessor should be considered. Unless the speed limitation of the microprocessor is a factor, all of the benefits mentioned earlier will result from the use of the microprocessor.

Another way to equate a microprocessor design with the more conventional hard-wired logic design, is to compare the number of gates in a hard-wired design with the number of bits of memory required by a microprocessor system. It has been determined that it takes approximately 8 to 16 bits of memory in a microprocessor system to replace a single gate. Since most read only memories used to store the program for a microprocessor can contain as many as 16,384 bits, such a memory can replace from 1000 to 2000 gates. Depending upon the number of gates per SSI or MSI package, this can represent a replacement of hundreds of integrated circuit packages. A 16,384 (16K) bit ROM in a single 40-pin IC package, for example, can replace one hundred to four hundred 14, 16, or 24-pin SSI and MSI packages. This is a significant saving.

At this point you may still have some doubts about the ability of a microprocessor to replace standard hard-wired logic functions. It may be difficult for you to imagine how a microprocessor can perform the functions you are so used to implementing with SSI and MSI packages. To ease your mind about this, let's consider all of the standard logic functions and illustrate how a microprocessor can perform them.

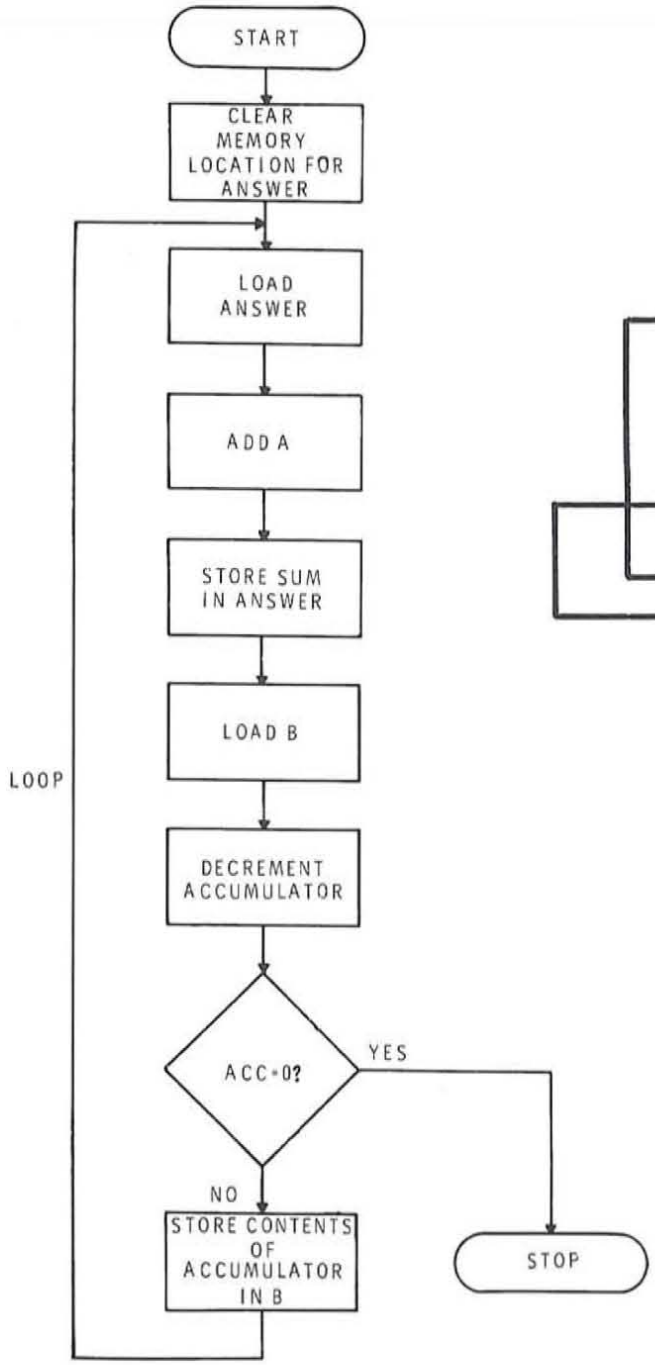
The microprocessor can readily perform all of the standard logical functions such as AND, OR, and Exclusive OR. It usually does this by executing the instructions designed for this purpose. Logical operations are generally performed on data stored in memory and in the accumulator register, with the result appearing in the accumulator. Suppose that you wanted to perform the NAND function on two 8-bit words. Using the instruction set in Table I, we could write the following program. Assume that the two words to be NANDed are stored in locations 8 and 9.

```
0    LDA  (8)
3    AND  (9)
6    CMP
7    HLT
```

The first instruction loads the first word into the accumulator. The second instruction performs the AND function with the word in the accumulator and the word in location 9. The result appears in the accumulator. Finally, this result is complemented to form the NAND function. This simple example illustrates the procedure you use to implement any Boolean function.

Arithmetic operations are also readily performed by a microprocessor. Special adders, subtractors and other arithmetic circuits are not required because all microprocessors can perform arithmetic operations through programming. Multiplication and division operations are carried out by subroutines. Even the higher math functions such as square root, trigonometric functions, and logarithms can be computed with subroutines. Many special algorithms have been developed for solving these higher mathematical functions with digital computers. To handle very large or very small numbers or to improve the accuracy of computation, multiple precision arithmetic subroutines are also available. Number size is limited by the number of bits in the basic computer data word. However, several computer words can be used to represent a quantity as large or as small as needed. Special programs can then be written to manipulate this data just as if it were represented by a single smaller word.

An example of a programmed arithmetic operation is shown in Figure 10-27. This flow chart illustrates the procedure for multiplying two positive numbers, A and B, by repeated addition. A is added B times to produce the product. The program to implement this algorithm is given below. The numbers to be multiplied are stored in locations 31 and 32. The product or answer is stored in location 33.




```
 0    CLA
 1    STA  (33)
 4    LDA  (33)
 7    ADD  (31)
10    STA  (33)
13    LDA  (32)
16    DCR
17    JMZ  (26)
20    STA  (32)
23    JMP  (4)
26    HLT
31    A
32    B
33    ANSWER
```

Figure 10-27
Flow chart of multiplication subroutine.

The first two instructions are used to clear the memory location where the ANSWER is to be stored. CLA resets the accumulator to zero and the STA instruction writes zeros in memory location 33. Next, the LDA instruction loads the contents of 33 (zero) into the accumulator. Then we add A to it with the ADD instruction. We then restore the partial product in location 33. We then load the content of location 32 (B) into the accumulator and subtract one from it with the decrement instruction DCR. We use a jump on zero (JMZ) instruction to see if the accumulator is zero. If it is not, we restore the accumulator content in location 32. The jump instruction creates a loop that returns us to the LDA (33) instruction. The entire sequence is then repeated. This continues until A has been added B times. Each time we add A to the answer, we subtract one from B. This permits us to keep track of how many times A has been added. When A has been added B times, the content of location 32 is again reduced by one producing a zero result. The JMZ instruction tests for zero. The correct product is contained in location 33 at this time. The JMZ causes the program to branch to location 26 where the HALT instruction is executed to stop the program.

Microprocessors can also be used to make decisions. For example, the microprocessor can compare two binary numbers and determine if they are equal or if one is greater than or less than another. This decision making function permits the microprocessor to evaluate information as it is developed and to modify its operation according to the values of the data.

The flow chart in Figure 10-28 illustrates one algorithm for comparing two binary numbers. Here, one number is subtracted from the other. A test for zero is then made. If the remainder is zero, of course, the numbers are equal. The program below implements this algorithm. The numbers to be compared are stored in locations 15 and 16.

```
 0    LDA  (15)
 3    SUB  (16)
 6    JMZ  (23)
 9    next instruction (A ≠ B)
23    next instruction (A = B)
```

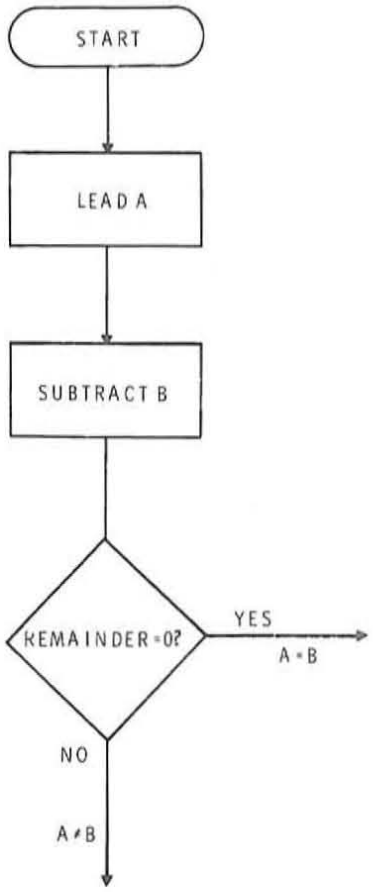
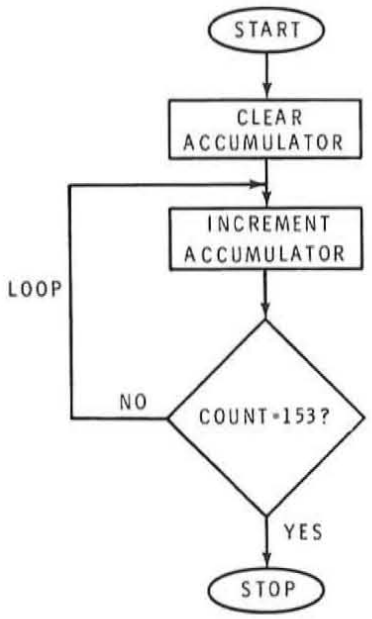


Figure 10-28
Flow chart of a procedure for comparing two numbers.

If the numbers are equal, the program branches to location 23. If the numbers are not equal, the program continues in its normal, sequential manner.

Another common logic function that is readily implemented with a microprocessor is counting. The microprocessor can count external events or a frequency standard. External events are counted by applying them to the interrupt line on the microprocessor. As each event occurs, an interrupt is generated with the microprocessor. This causes the microprocessor to jump to a subroutine that will increment the accumulator register or add one to some memory location. Up or down counters are readily implemented with the increment and decrement accumulator instructions. Decision-making techniques can be used to detect when a specific count is reached or to count quantities larger than the computer word size permits. For example, with an 8-bit data word in a microprocessor, the maximum count that the accumulator can handle is 1111 1111 or 255. To count to higher values, a program can be written to indicate each time the counter overflows.

The program below illustrates a method of detecting a count of 153. The flow chart in Figure 10-29 shows the approach.




```
 0   CLA
 1   INC
 2   STA  (18)
 5   SUB  (19)
 8   JMZ  (21)
11   LDA  (18)
14   JMP  (1)
18   Count
19   153
21   LDA  (18)
24   HLT
```

Figure 10-29
Flow chart showing a method of detecting a count of 153.

The first instruction clears the accumulator. The accumulator is then incremented by the INC instruction, and the count is stored in location 18. The count is then compared by subtracting 153 and testing for zero. If a non-zero result occurs, the count is retrieved with the LDA instruction and the program loops back to the increment instruction. This loop continues until a count of 153 is reached. When the JMZ instruction detects the zero condition, the program branches to location 21 where the count is loaded and the program halts.

To count to numbers higher than 255, the program below can be used. See the flow chart in Figure 10-30 for an explanation of the procedure. This program counts in multiples of 256. Note that the program has two loops. The inner loop determines when a count of 256 occurs, while the outer loop determines the number of times that the inner loop occurs. The total count then is the product of the number of times the inner loop occurs and the count in location 37, in this case 5. The program halts on a count of 5 × 256 or 1280.

```
                0    CLA
                1    STA   (36)
          ┌─►    4    INC
          │     5    JMZ   (11)
          │     8    JMP   (4)
inner     │    11    LDA   (36)
loop      │    14    INC
          │    17    STA   (36)
          │    20    SUB   (37)
          │    23    JMZ   (30)
          │    26    CLA
          │    27    JMP   (4)
outer     │    30    HLT
loop      │    36    COUNT
          └─   37    5
```

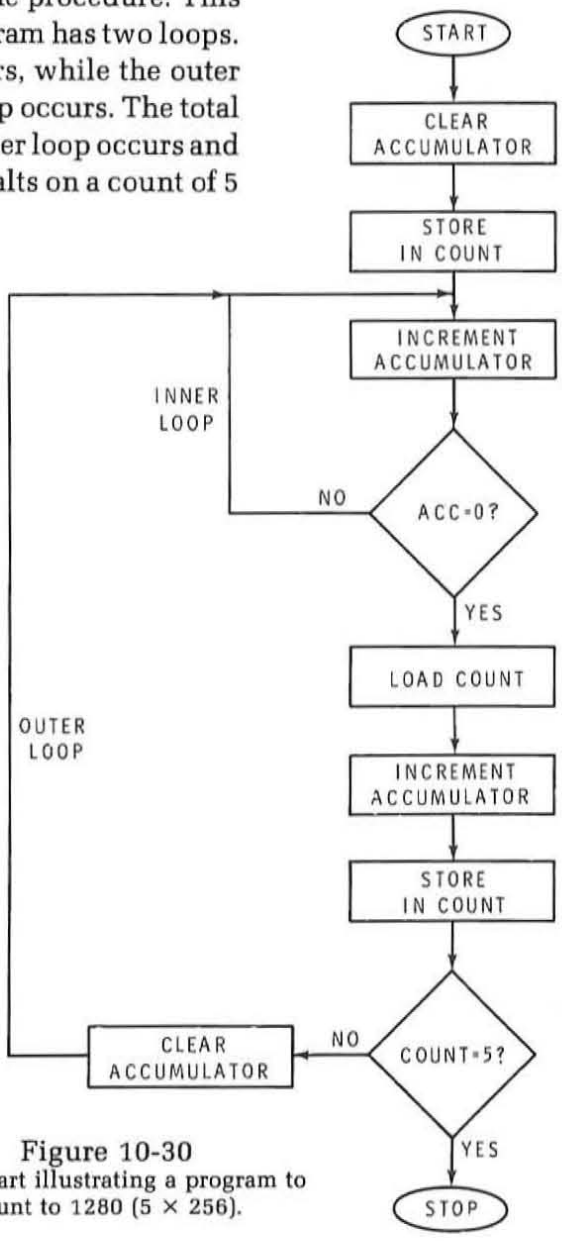


Figure 10-30
Flow chart illustrating a program to count to 1280 (5 × 256).

The first instruction clears the accumulator. The next instruction writes zero into memory location 36 which we call COUNT. The content of location 36 tells us how many times the inner loop is repeated. These first two instructions initialize the circuitry prior to starting the count. The program then begins the count by executing the increment instruction in location 4. The JMZ (11) then tests for zero. If the accumulator is not zero, the JMP instruction is executed creating a loop that returns the program to the INC instruction. The loop is repeated 255 times at which time the accumulator is 1111 1111. The INC instruction is executed a 256th time and the accumulator recycles to 0000 0000. Then the JMZ (11) instruction again tests for zero. This time the program branches to location 11 where the LDA (36) instruction is performed. This loads COUNT (which is initially zero). COUNT is then incremented to indicate that a count of 256 has occurred. COUNT is then restored by the STA (36) in location 17. Next, the program tests to see if COUNT is 5. It subtracts 5 from COUNT. If the remainder is not zero, the accumulator is cleared and the program loops back to the beginning where the inner loop is again repeated. When COUNT becomes 5, the inner loop has been repeated 5 times indicating a count of 1280. The program then branches via the JMZ (30) instruction to a HLT.

The microprocessor can also generate timed output pulses. It can do this in several ways. The simplest method is to store a series of binary numbers in sequential memory locations with the proper bit designations. These can then be read out of memory, one at a time, and sent to the output data bus. As the binary words change, the output bits change and generate any desired sequence of timing pulses. The rate of occurence of these pulses depends upon the speed of the microprocessor. Longer timed output pulses can also be generated by producing internal timing delays. This can be done by programming counting loops like the ones just illustrated. The time delays are a product of the instruction execution speed and the desired count. With a count of 153 and an instruction execution speed of 12.5 microseconds, the total delay would be $153 \times 12.5 = 1912.5$ microseconds. The binary words in memory could be outputted every 1.9125 milliseconds.

As you can see, microprocessors can perform all the same functions as hard-wired digital logic systems. The logic designer no longer spends his time in designing circuits or in minimizing Boolean expressions. Instead, he writes programs. For that reason, it is highly desirable for the digital designer to learn programming. As microprocessors become more widely used, the digital engineer will become more of a programmer than a circuit or logic designer.

Self Test Review

52. Write a program that performs the exclusive OR function on two 8-bit words, A and B, stored in locations 41 and 42. Use the instruction set in Table I. Start your program in location 0.
53. Write a program showing how you would multiply a number in location 22 (X) by 8. (Hint: A shift-left operation multiplies by 2.) Start your program in location 0.
54. Which of the following best describes a microprocessor?
    a. General purpose digital computer
    b. Special purpose digital computer
55. The memory used with a microprocessor is usually a
    a. semiconductor RAM
    b. semiconductor ROM
    c. magnetic core
56. List the four benefits of using microprocessors to replace hard-wired logic.
    a. _____
    b. _____
    c. _____
    d. _____
57. A hard-wired logic system using 40 CMOS SSI and MSI packages is to be redesigned. Would a microprocessor be a good alternative to consider?
    a. True
    b. False

Answers

52. The exclusive OR function is $C = A\overline{B} + \overline{A}B$. The flow chart of this function is shown in Figure 10-31 and the program is given below.

```
 0    LDA  (42)
 3    CMP
 4    AND  (41)
 7    STA  (43)
10    LDA  (41)
13    CMP
14    AND  (42)
17    OR   (43)
20    HLT
41    A
42    B
43    Intermediate Results AB̄
```

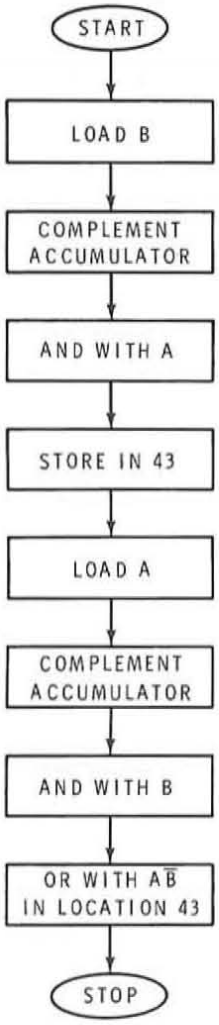


Figure 10-31
Flow chart of a program to perform the exclusive OR function.

The first instruction loads B into the accumulator. It is the complemented to product $\overline{B}$. $\overline{B}$ is then ANDed with A by the third instruction. The result, $A\overline{B}$, is stored in location 43 for later use. Next, A is loaded with the LDA (41). A is then complemented to produce $\overline{A}$. $\overline{A}$ is ANDed with B by the AND (42) instruction. The result $\overline{A}B$ appears in the accumulator. Finally, the contents of the accumulator is ORed with the content of 43 ($A\overline{B}$) to produce $\overline{A}B + A\overline{B}$ which appears in the accumulator.

53. The flow chart in Figure 10-32 shows one method of multiplying the content of location 22 (X) by 8. The program is given below.

```
0    LDA  (22)
3    SHL
4    STA  (22)
7    LDA  (23)
10   DCR
11   JMZ  (24)
14   STA  (23)
17   JMP  (0)
20
22   X
23   COUNT (3)
24   LDA  (22)
27   HLT
```

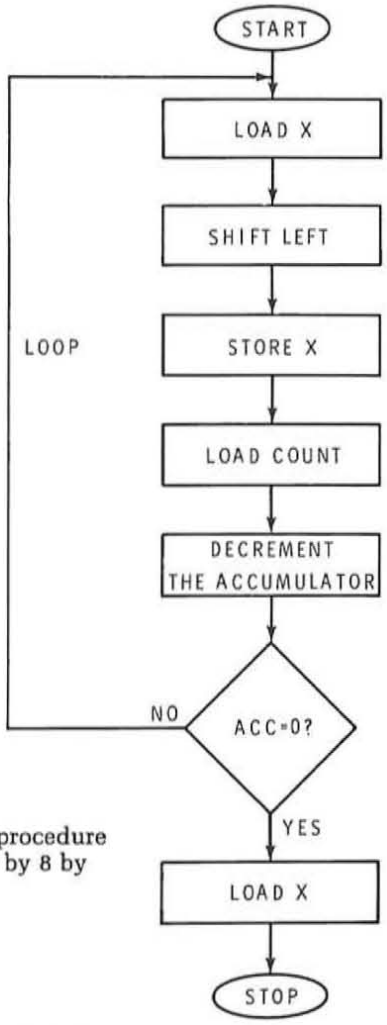


Figure 10-32
Flow chart illustrating the procedure for multiplying a number by 8 by shifting.

Each time the number X is shifted left, it is effectively multiplied by 2. Three shifts produce multiplication by 8. A counter and decision-making loop determine when three shifts occur.

54. b. Special purpose digital computer
    The microprocessor is usually dedicated to a specific application.

55. b. semiconductor ROM

56. a. Less design time and cost
    b. Less manufacturing time and cost
    c. Enhanced product capability
    d. Greater reliability

57. a. True

EXAMINATION

UNIT 10

DIGITAL APPLICATIONS

The purpose of this exam is to help you review the key facts in this unit. The problems are designed to test your retention and understanding by making you apply what you have learned. This exam is not so much a test as it is another learning method. Be fair to yourself and work every problem first before checking the answers.

1. A frequency counter cannot measure:
   A. frequency
   B. voltage
   C. period
   D. time
2. In the frequency mode of operation of a digital counter, the gate interval is controlled by the:
   A. Input signal
   B. Schmitt trigger
   C. Time base
   D. Decade counter
3. Period measurements are made on low frequency signals to improve the measurement:
   A. resolution
   B. accuracy
   C. speed
   D. convenience
4. A frequency counter counts 1715 pulses during a 100 microsecond interval. This represents a frequency of:
   A. 1715 Hz
   B. 171.5 KHz
   C. 1.715 MHz
   D. 17.15 MHz
5. Using the instruction set in Table I, write a program that compares two binary numbers, A and B, that are stored in memory locations 41 and 42. Do *not* use the compare algorithm of subtracting one quantity from another as discussed in the text. Develop a new algorithm. Start the program in location 0.

6. Write a program that will determine (count) how many times the number 5 can be subtracted from the number 215. The number 5 is in location 31 and the number 215 is in location 32. Start your program in location 0. Store the count in location 33. End the program with the count in the accumulator.
7. A hard-wired logic system uses 30 ECL MSI and SSI integrated circuits. Could this system be replaced by a single chip microprocessor?
   A. Yes
   B. No
8. Most single chip microprocessors contain which of the following major sections of a digital computer? Check all that apply.
   A. Memory
   B. ALU
   C. Control
   D. I/O
9. What register in the CPU of a digital computer would you look at to determine the address of the next instruction to be fetched?
   A. Accumulator
   B. Program counter
   C. Instruction register
   D. Memory data register
10. External signals that change the normal program sequence are called a:
    A. bus
    B. branch or jump
    C. interrupt
    D. fetch
11. Study the program below and determine which algebraic expression is being solved.

```
 0   LDA  (15)
 3   ADD  (16)
 6   ADD  (17)
 9   SUB  (18)
12   SHL
13   SHL
14   HLT
15   W
16   X
17   Y
18   Z
```

    A. $W + X + Y - Z$
    B. $W + X - Y + Z$
    C. $4(W + X + Y - Z)$
    D. $(W + X + Y - Z)/4$

12. The program stored in a large scale digital computer that is used to convert an instruction-by-instruction higher level language program into the binary code used in a microprocessor is called a(n):
    A. Compiler
    B. Assembler
    C. Subroutine
    D. Cross assembler

ANSWERS

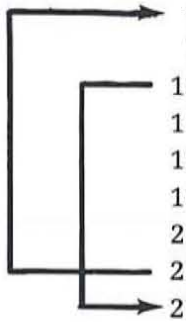
UNIT 10

DIGITAL APPLICATIONS

1. B - Voltage
2. C - Time base
3. A - Resolution
4. D - 17.15 MHz. To determine the frequency in pulses per second or Hz, you multiply the number of counts in 100 microseconds (1715) by the number of 100 microsecond intervals in one second (10,000). The frequency then is 17150000 or 17.15 MHz.
5. The exclusive OR function can be used to compare two binary numbers. Remember that an exclusive NOR is a single bit comparator. The algorithm is simply to perform an exclusive OR on the two words to be compared. If they are equal, the accumulator will be zero. A JMZ instruction can then test for zero. The program below is identical to that described in the answer to Self Test Review Question 52 except for the JMZ instruction.

```
 0    LDA  (42)
 3    CMP
 4    AND  (41)
 7    STA  (43)
10    LDA  (41)
13    CMP
14    AND  (42)
17    OR   (43)
20    JMZ  (30)
23    next instruction if A ≠ B
30    next instruction if A = B
41    A
42    B
43    intermediate results A B̄
```

6.



```
 0    CLA
 1    STA  (33)
 4    LDA  (32)
 7    SUB  (31)
10    JMZ  (26)
13    STA  (32)
16    LDA  (33)
10    INC
20    STA  (33)
23    JMP  (4)
26    LDA  (33)
29    INC
30    HLT
31    5
32    215
33    ANSWER (COUNT)
```

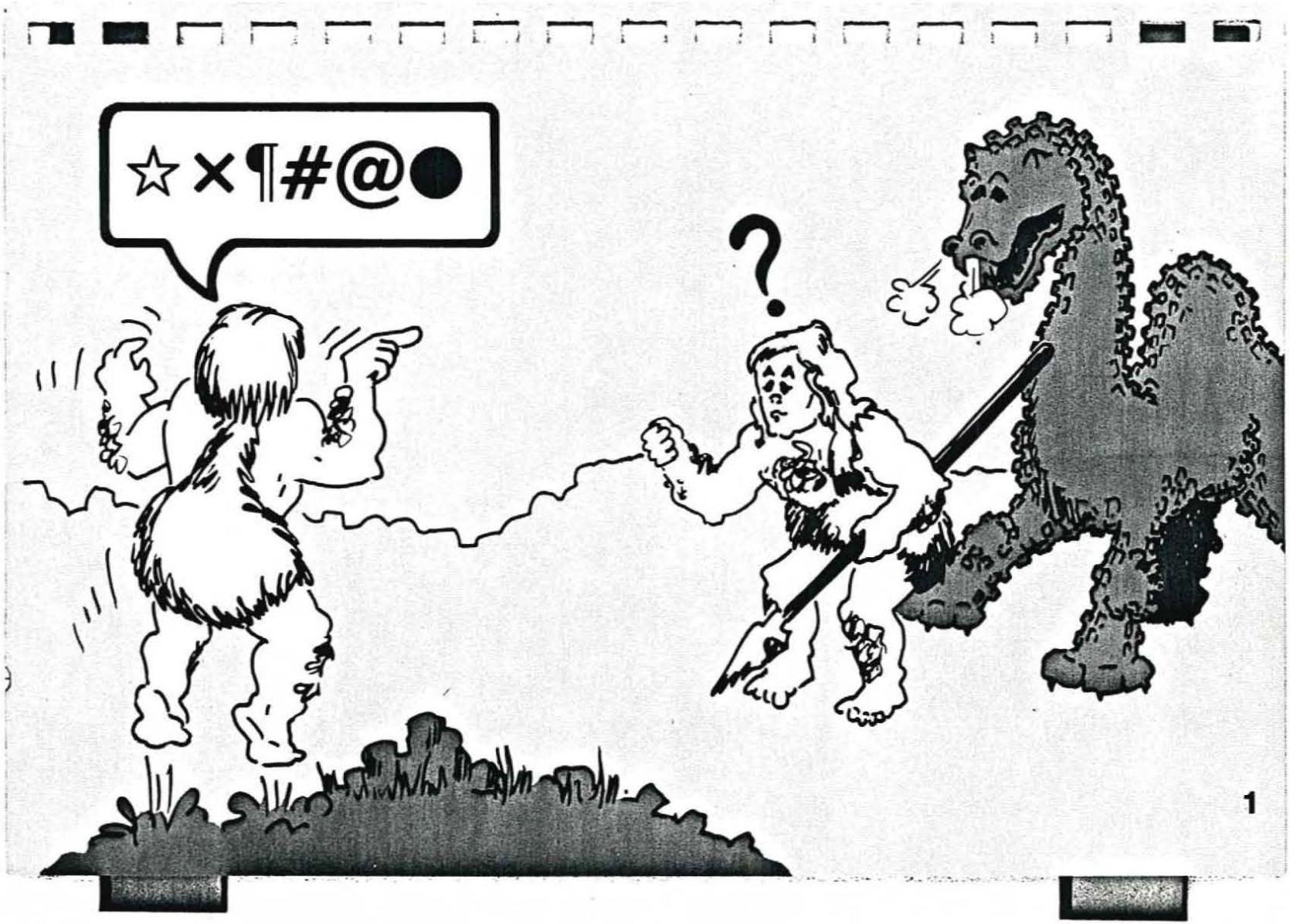
The first two instructions clear the accumulator and load zero into location 33 where the answer will be stored. Next, the LDA (32) instruction loads 215 into the accumulator. The next instruction subtracts 5. We then test for zero. If the accumulator is not zero, we store its content back into location 32. Next, we load the COUNT (initially zero) and increment it, thereby indicating that we subtracted 5 once from 215. The count is then restored in location 33. Next, the program loops back to again subtract 5 and test for zero. The loop repeats and the COUNT is incremented each time a subtraction occurs. When the number in location 32 is reduced to zero, the JMZ instruction branches to location 26 where the COUNT is loaded and incremented once more to indicate the final subtraction.

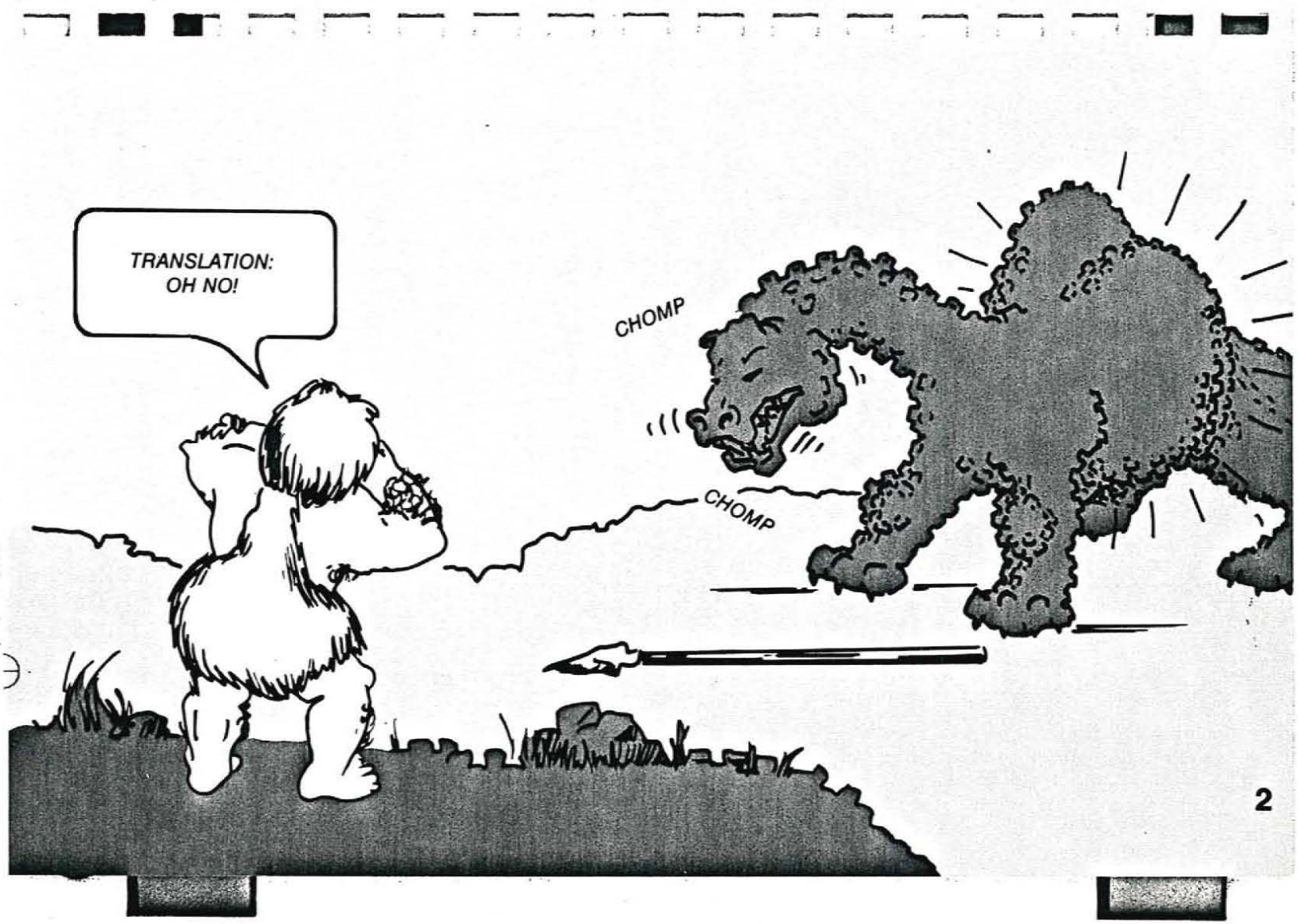
If you will study the solution closely, you will see that it is actually a simple division subroutine. The divide algorithm is repeated subtraction. The quotient is in location 33. The program will not work for numbers that do not result in an integer quotient.

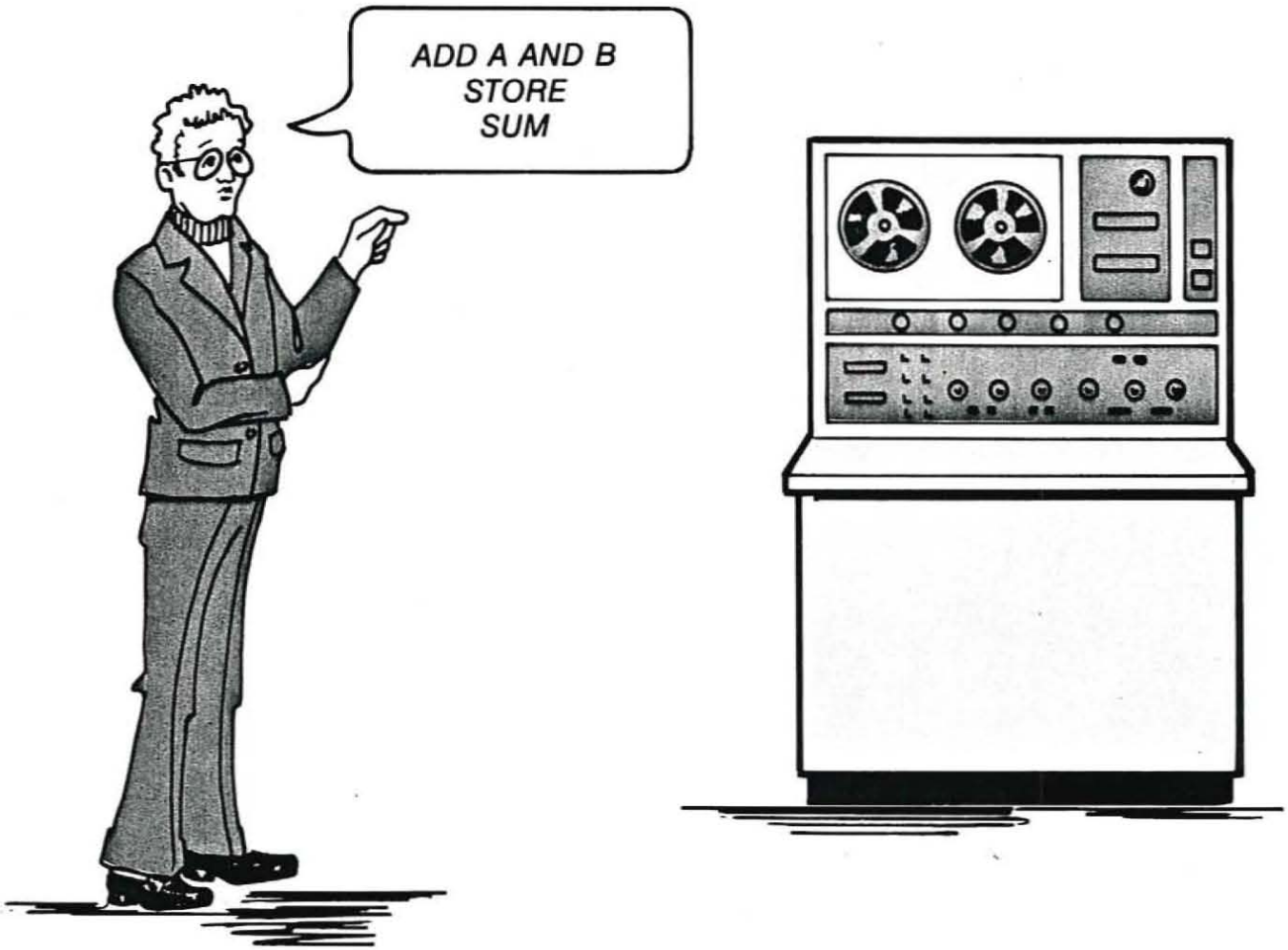
7. B - No. Because of the package count a microprocessor would ordinarily be considered. But most single chip microprocessors are probably not fast enough to achieve the desired speed if ECL is used in the original design.

8. B, C - ALU and Control
9. B - Program counter. This register keeps track of the addresses of the instructions being executed.
10. C - Interrupt
11. C - 4(W + X + Y + Z). The two shift-left operations multiply the result by 4.
12. D - Cross assembler

INTRODUCTION TO PROGRAMMING

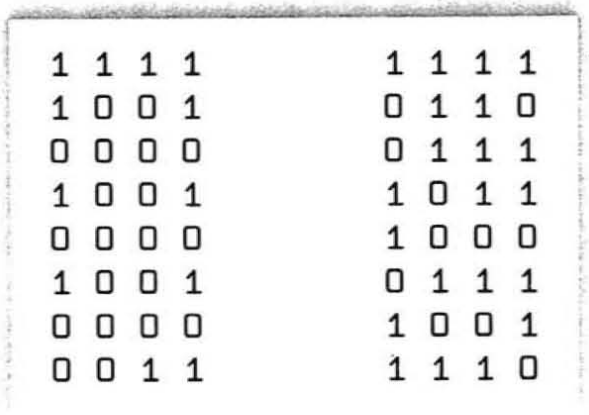

☆×¶#@●
?
1


TRANSLATION:
OH NO!
CHOMP
CHOMP
2

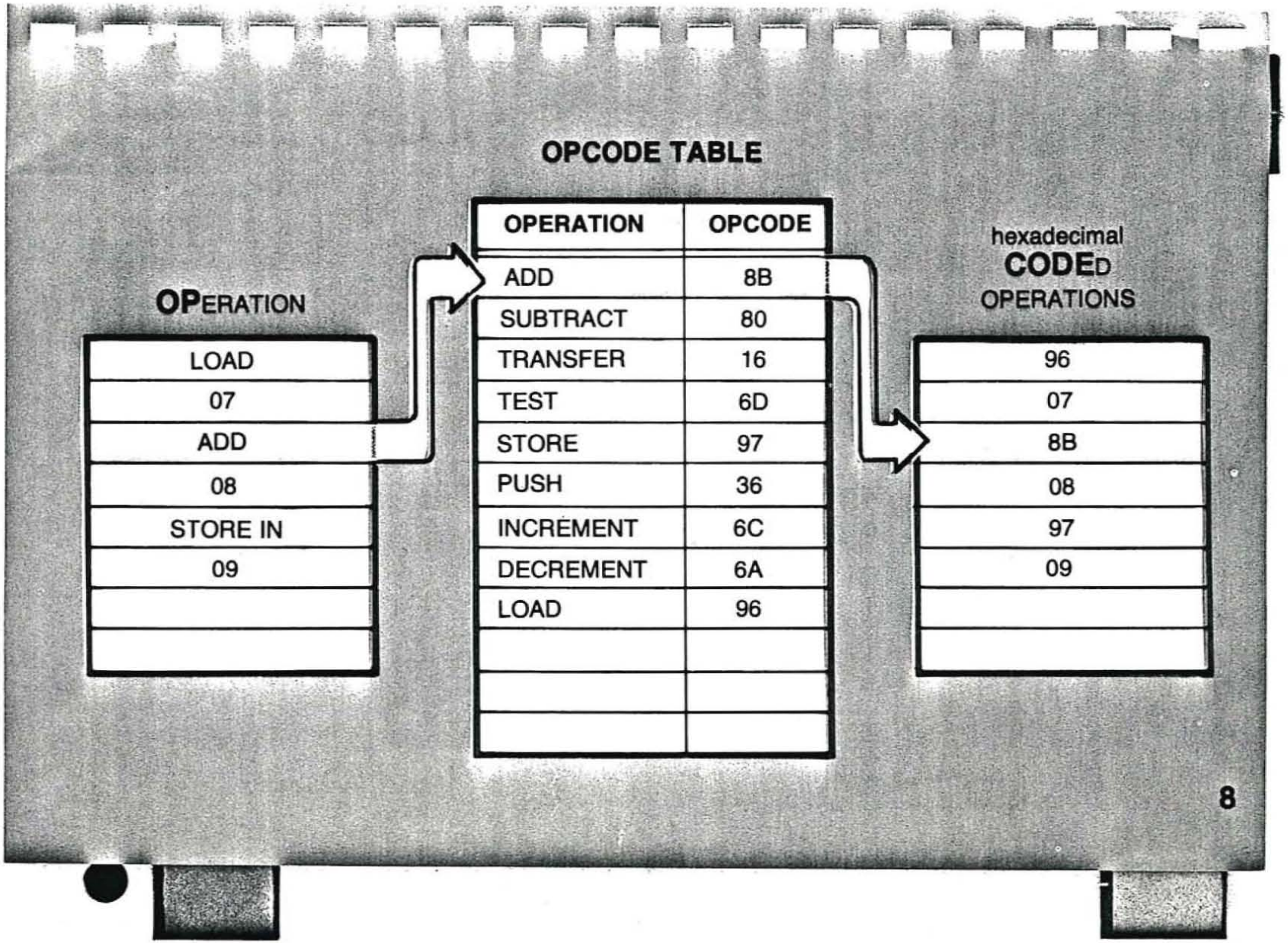

ADD A AND B
STORE
SUM

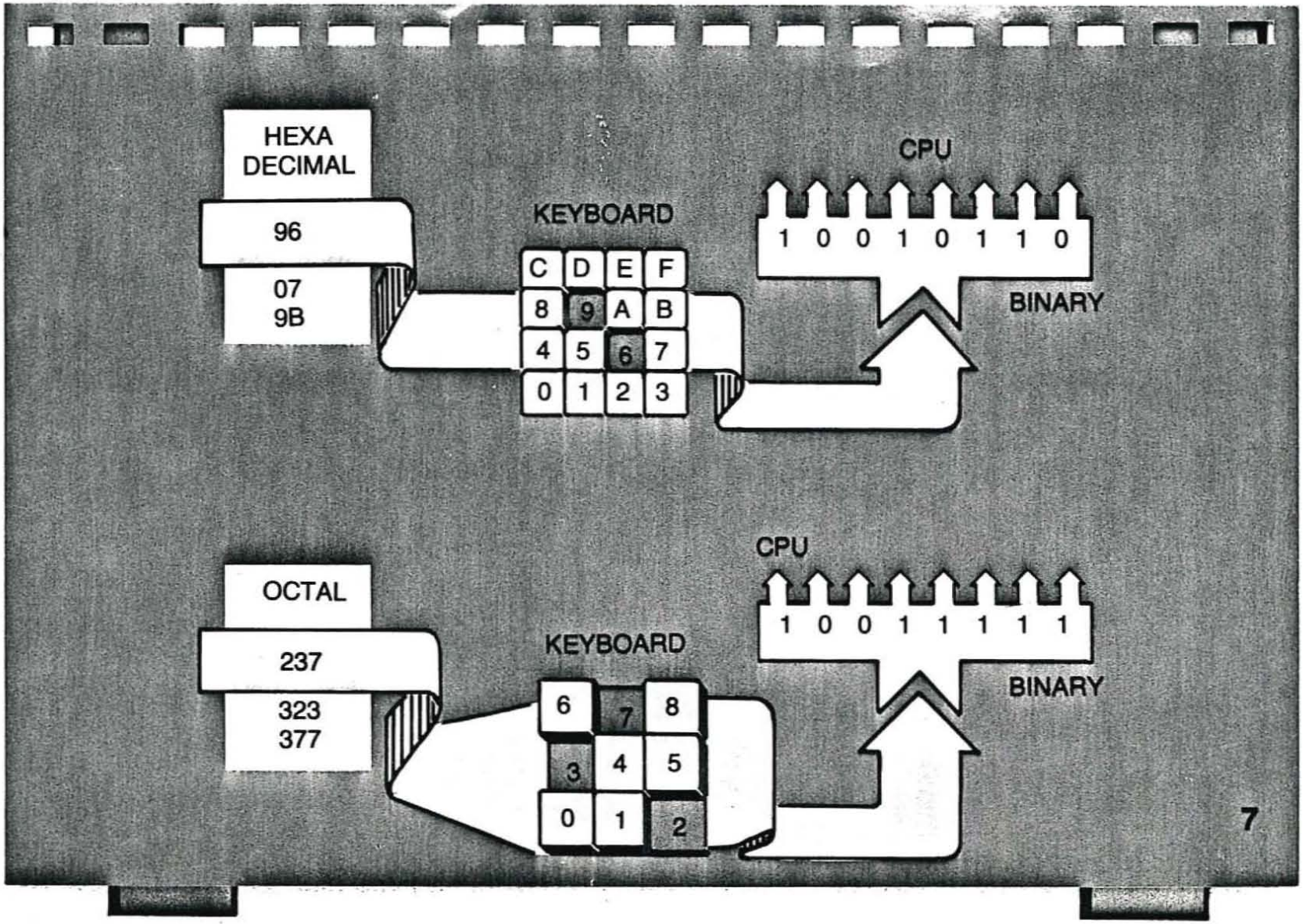
MACHINE LANGUAGE

ADD
A
AND
B
STORE
SUM
STOP

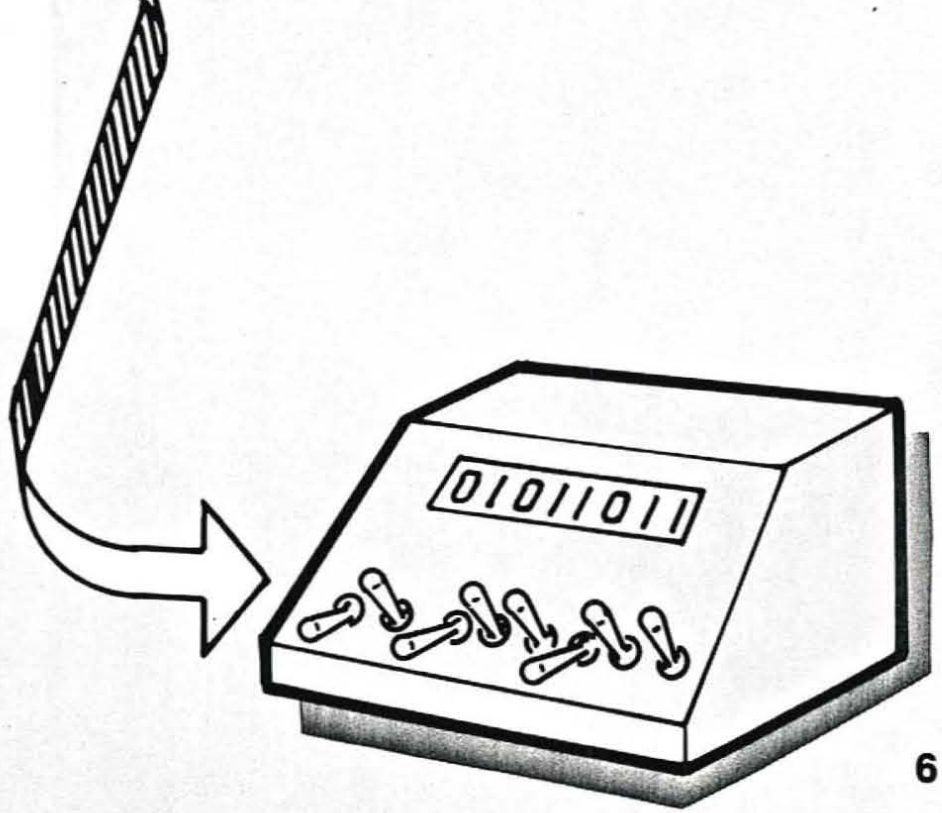


MACHINE LANGUAGE


OPCODE TABLE
OPERATION
LOAD
07
ADD
08
STORE IN
09
OPERATION
OPCODE
ADD
8B
SUBTRACT
80
TRANSFER
16
TEST
6D
STORE
97
PUSH
36
INCREMENT
6C
DECREMENT
6A
LOAD
96
hexadecimal
CODED
OPERATIONS
96
07
8B
08
97
09
8


HEXA
DECIMAL
96
07
9B
KEYBOARD
C D E F
8 9 A B
4 5 6 7
0 1 2 3
CPU
1 0 0 1 0 1 1 0
BINARY
OCTAL
237
323
377
KEYBOARD
6 7 8
3 4 5
0 1 2
CPU
1 0 0 1 1 1 1 1
BINARY
7

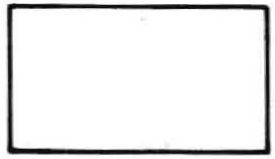
0 1 0 1	1 0 1 1
0 0 0 0	0 1 1 1
1 0 0 1	1 0 1 1
0 0 0 0	1 0 0 0
1 0 0 1	0 1 1 1
0 0 0 0	1 0 0 1
1 1 0 1	0 0 0 1
0 0 1 1	1 1 1 0



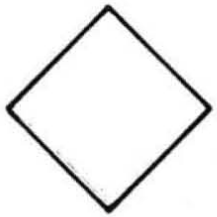
MACHINE LANGUAGE

ENTERED VIA
BINARY SWITCHES

FLOWCHART SYMBOLS



OPERATIONS



DECISION

TERMINAL

CONNECTOR

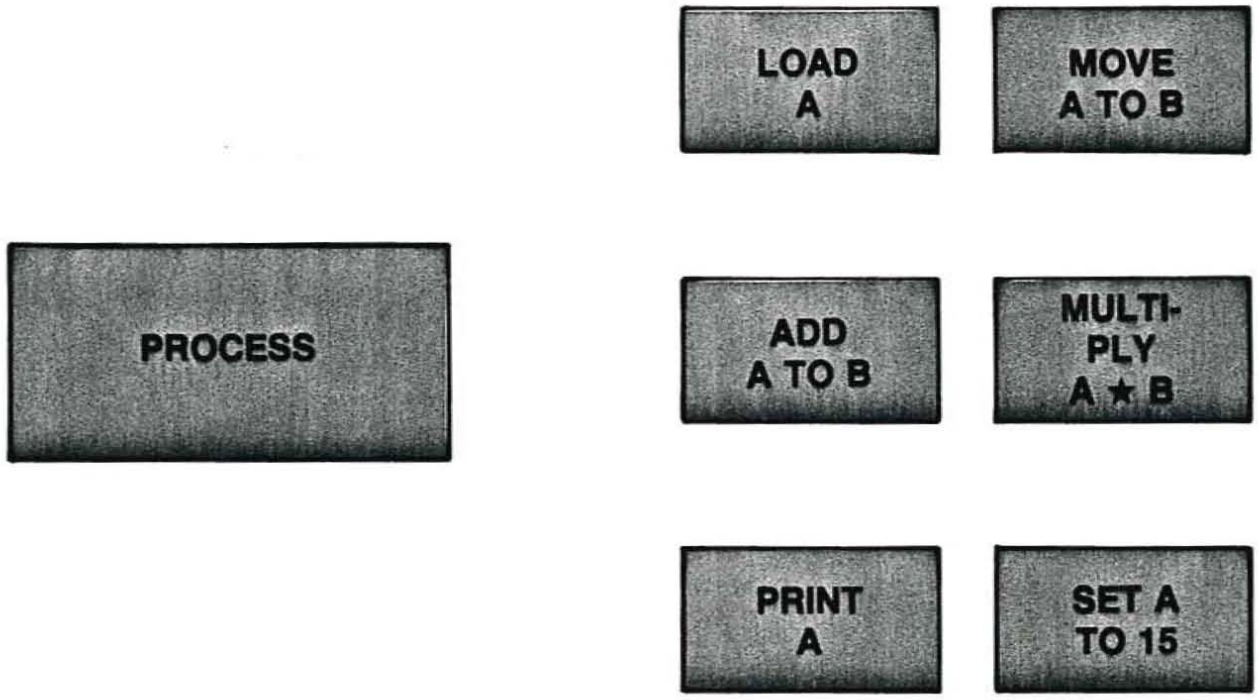
FLOW LINES

TERMINAL SYMBOL

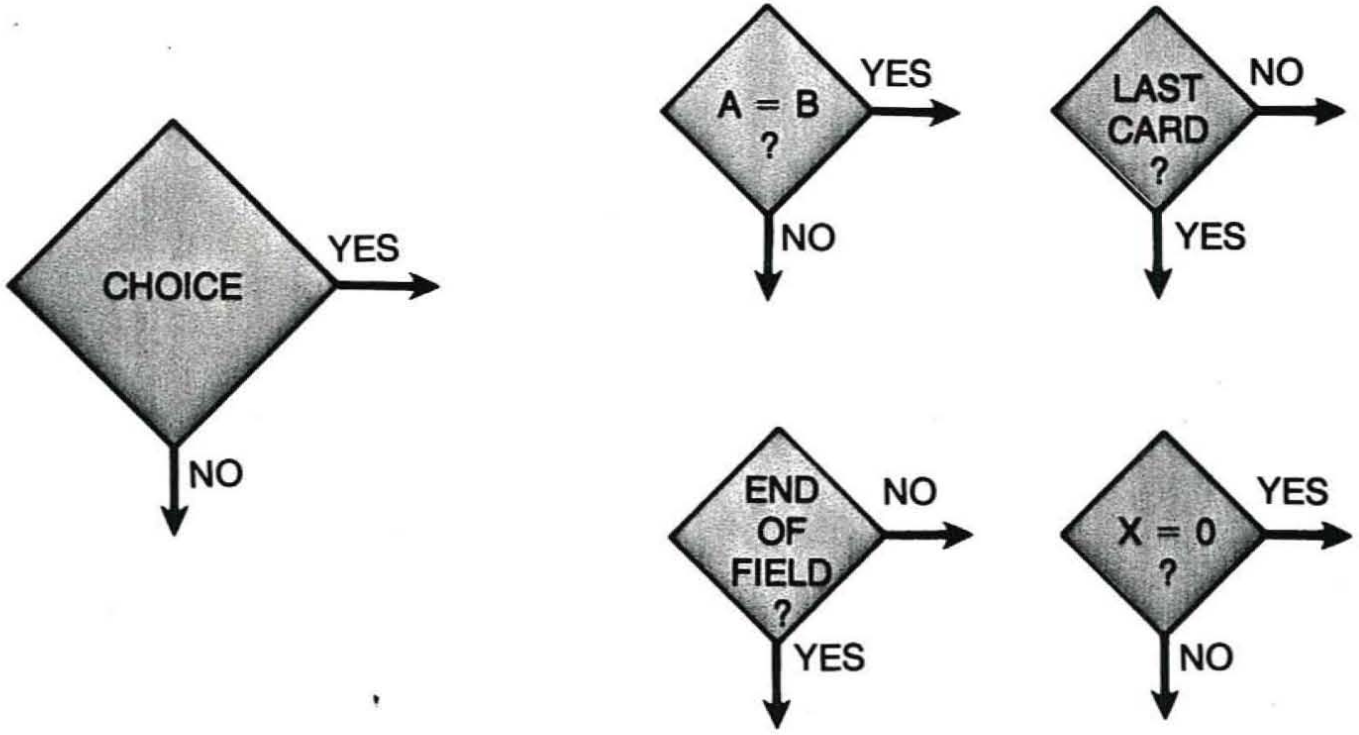
START

STOP

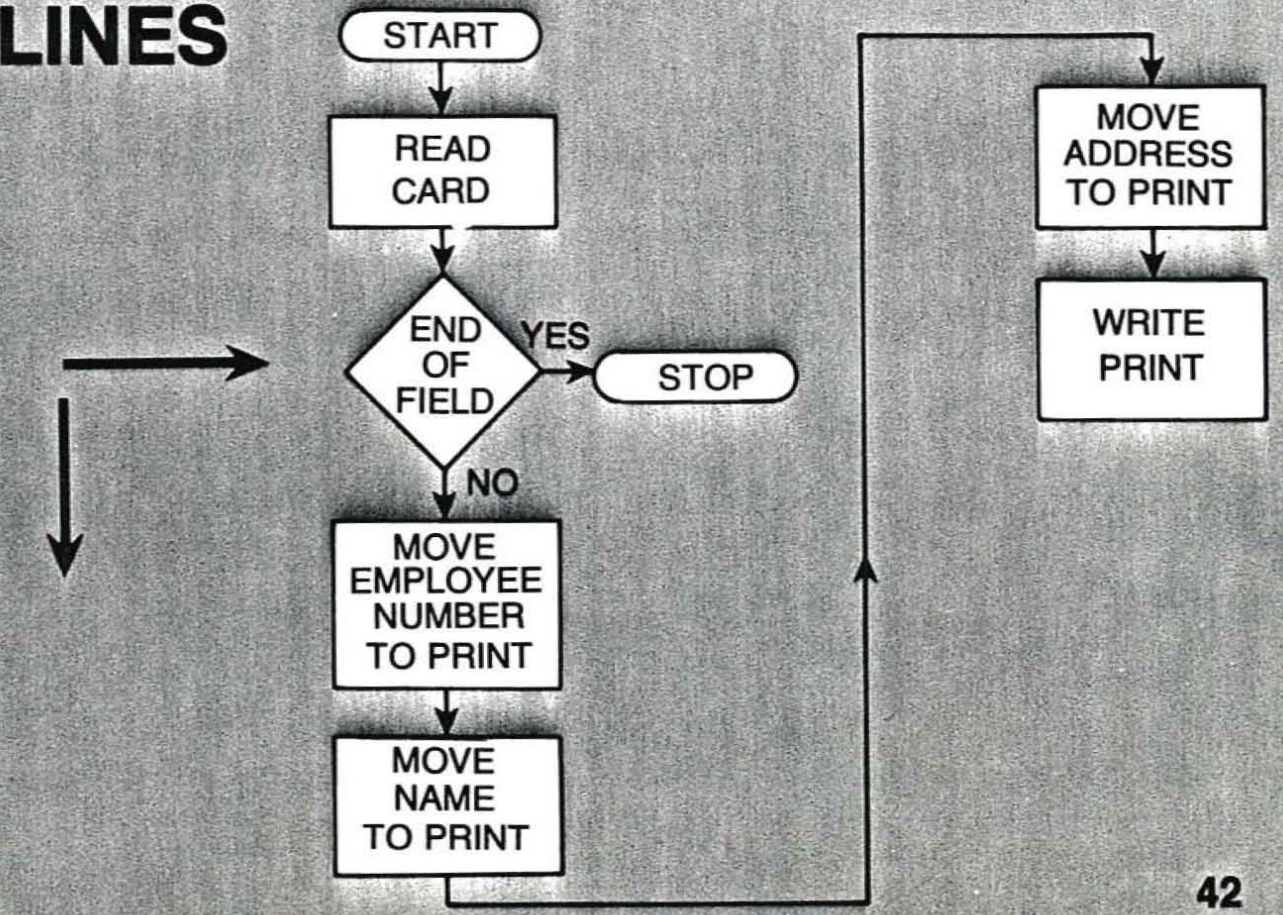
OPERATION BOX



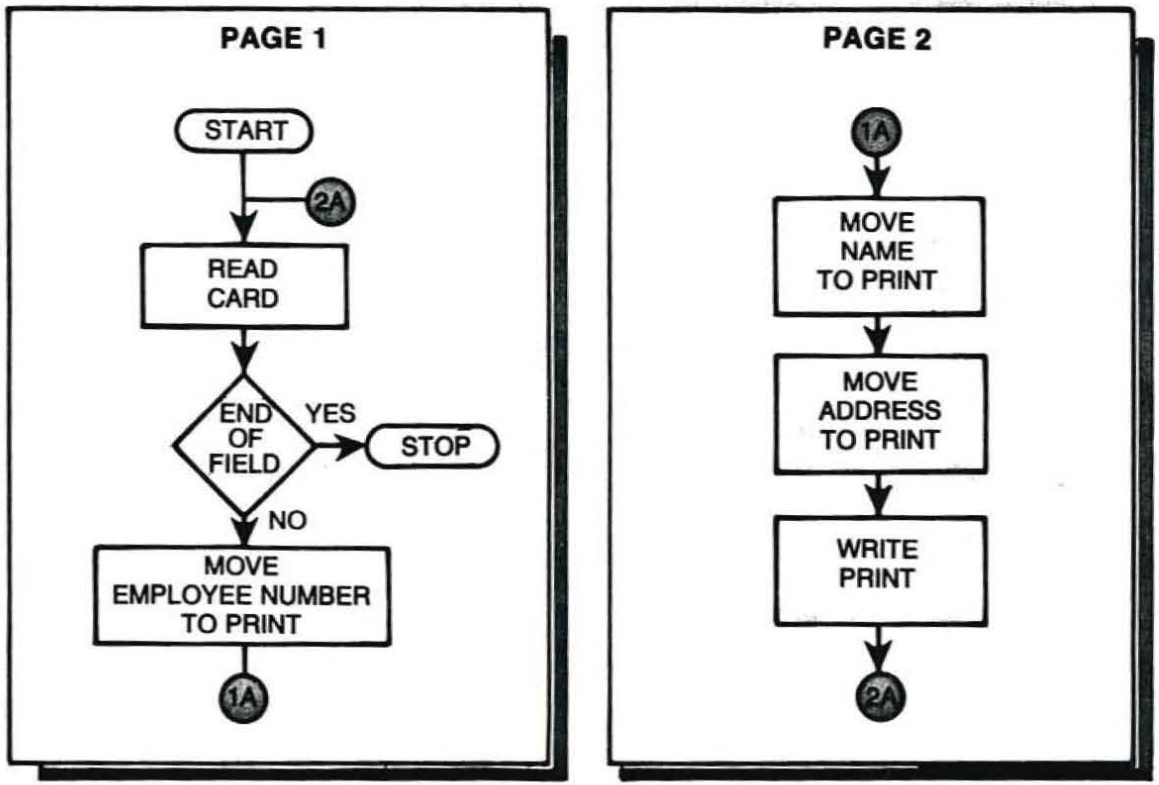
DECISION BOX



FLOW LINES


START
READ CARD
END OF FIELD
YES
STOP
NO
MOVE EMPLOYEE NUMBER TO PRINT
MOVE NAME TO PRINT
MOVE ADDRESS TO PRINT
WRITE PRINT

CONNECTOR


PAGE 1
START
2A
READ CARD
END OF FIELD
YES
STOP
NO
MOVE EMPLOYEE NUMBER TO PRINT
1A
PAGE 2
1A
MOVE NAME TO PRINT
MOVE ADDRESS TO PRINT
WRITE PRINT
2A

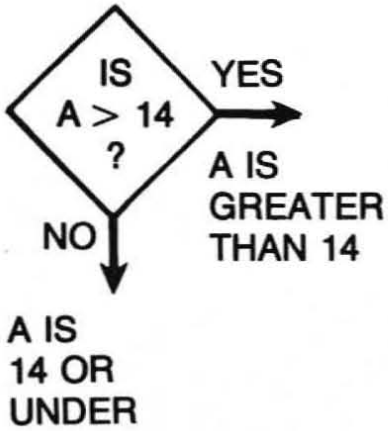
DECISION-MAKING SYMBOLS

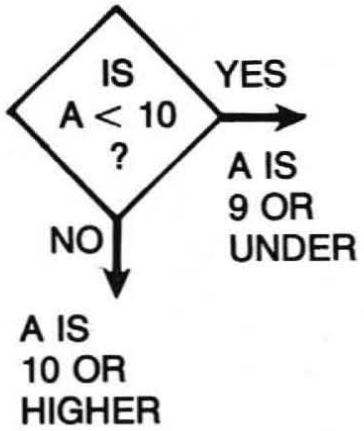
	SYMBOL	EXAMPLE	DESCRIPTION
1.	$=$	$A = B$	A is equal to B.
2.	$>$	$A > B$	A is greater than B.
3.	$<$	$A < B$	A is less than B.
4.	$\geqq$	$A \geqq B$	A is greater than or equal to B.
5.	$\leqq$	$A \leqq B$	A is less than or equal to B.

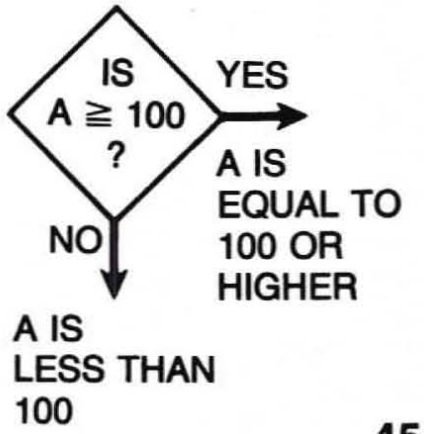
DECISION-MAKING SYMBOLS

ARE GENERALLY USED IN

DECISION BOXES.





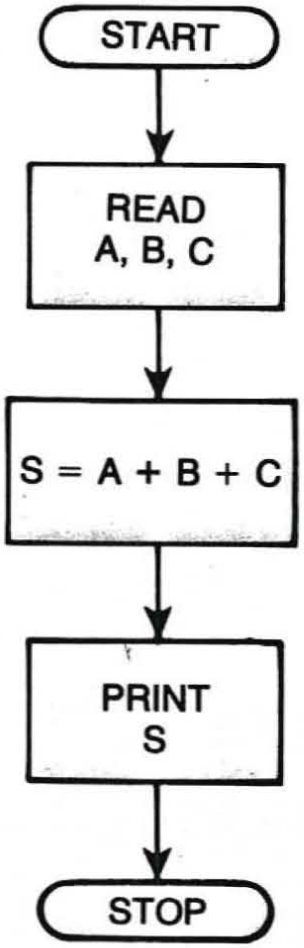


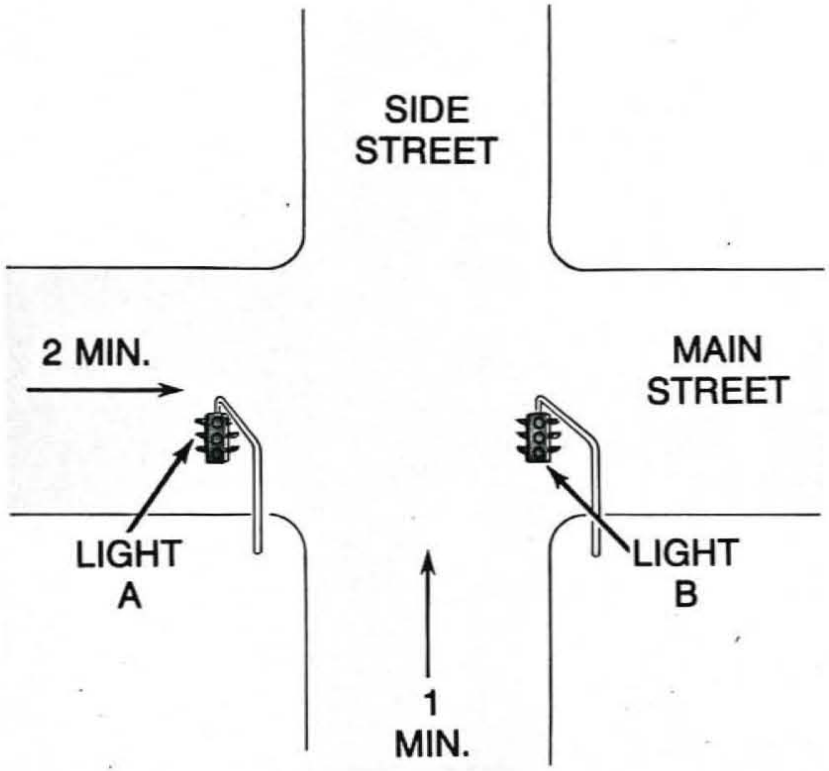
PROBLEM:

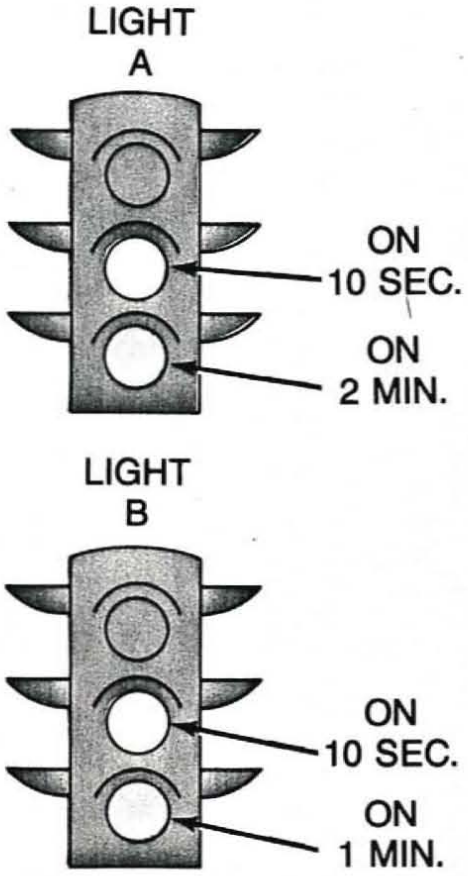
1. Add 3 numbers:
   A, B, and C producing sum, S.

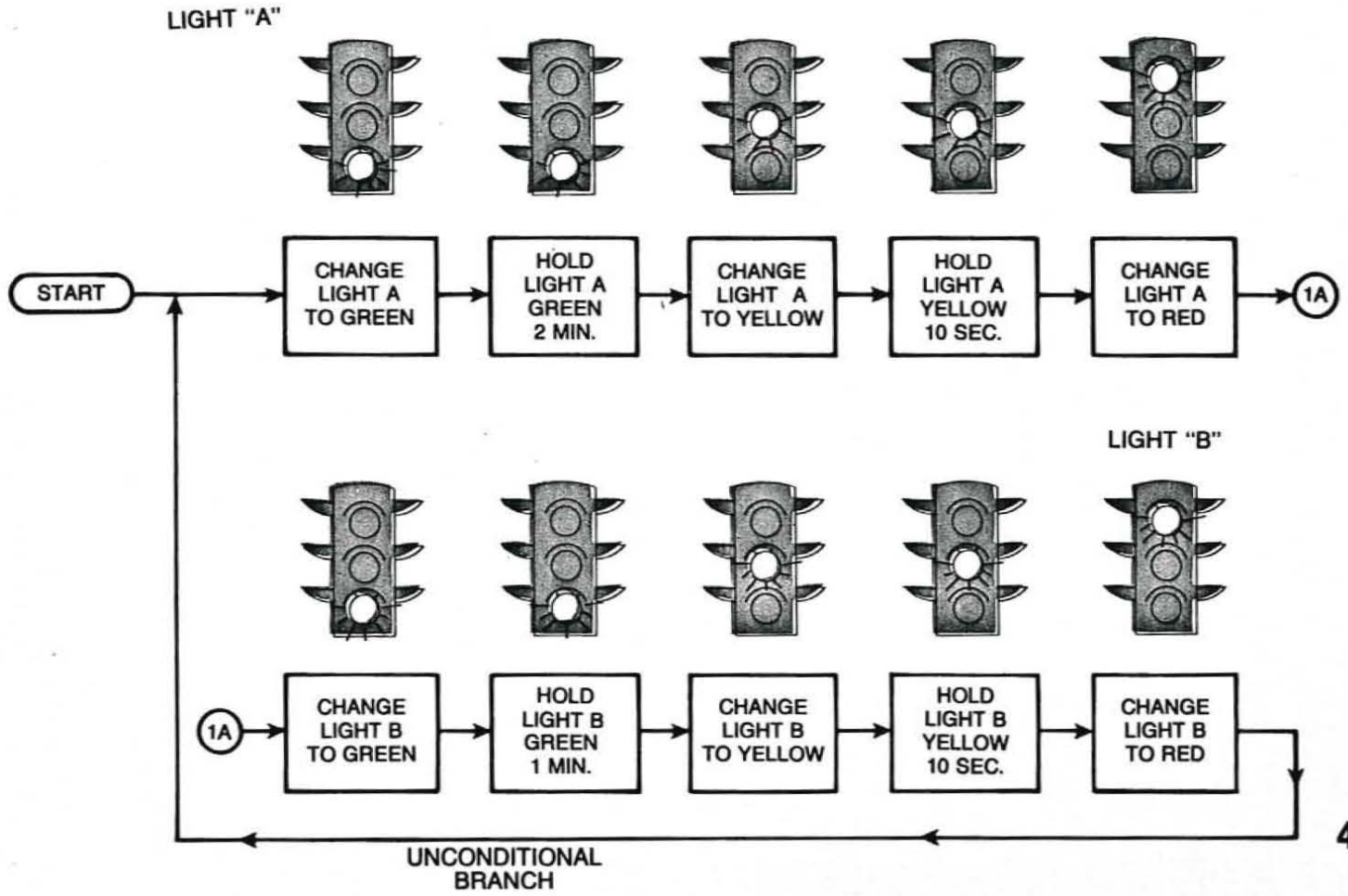
2. Print sum.

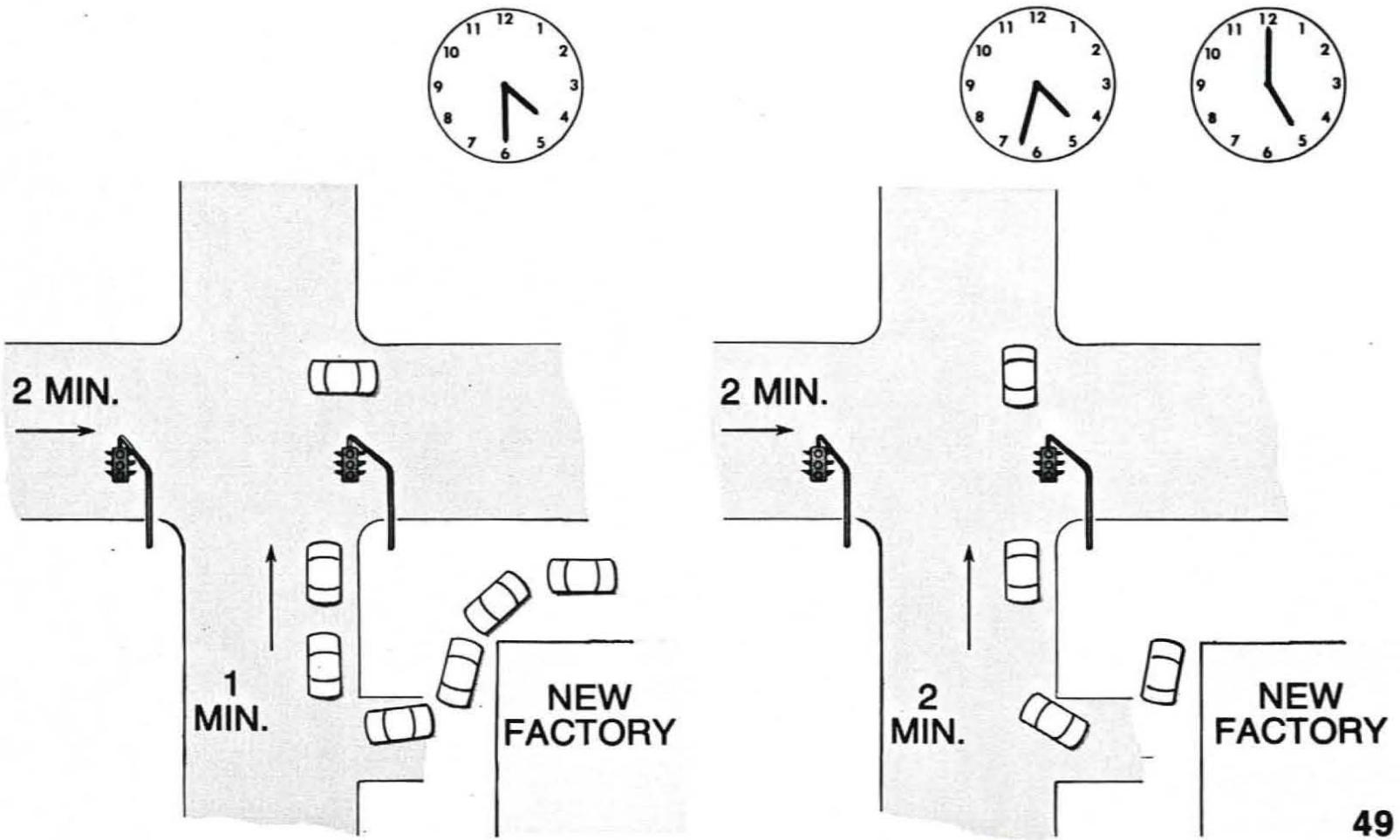
S = A + B + C




SIDE
STREET
2 MIN.
MAIN
STREET
LIGHT
A
LIGHT
B
1
MIN.


LIGHT
A
ON
10 SEC.
ON
2 MIN.
LIGHT
B
ON
10 SEC.
ON
1 MIN.


LIGHT "A"
START
CHANGE LIGHT A TO GREEN
HOLD LIGHT A GREEN 2 MIN.
CHANGE LIGHT A TO YELLOW
HOLD LIGHT A YELLOW 10 SEC.
CHANGE LIGHT A TO RED
1A
LIGHT "B"
1A
CHANGE LIGHT B TO GREEN
HOLD LIGHT B GREEN 1 MIN.
CHANGE LIGHT B TO YELLOW
HOLD LIGHT B YELLOW 10 SEC.
CHANGE LIGHT B TO RED
UNCONDITIONAL BRANCH


2 MIN.
1
MIN.
NEW
FACTORY
2 MIN.
2
MIN.
NEW
FACTORY

PROBLEM DEFINITION:

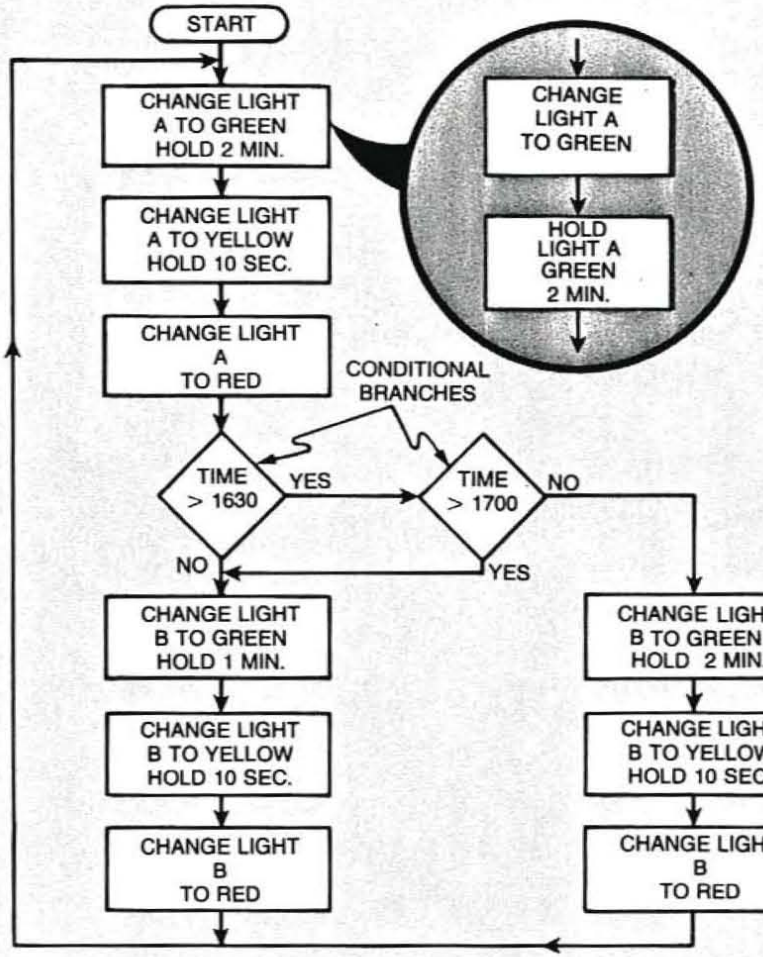
1. Normal light timing works well until factory closes at 4:30 PM.

2. Traffic study shows that if Light "B" is retimed to stay green for 2 minute intervals between 4:30 PM and 5:00 PM congestion is relieved.

3. After 5:00 PM traffic lights should be returned to the "NORMAL" timing cycle.

4. Light "A's" green timing cycle is never changed.

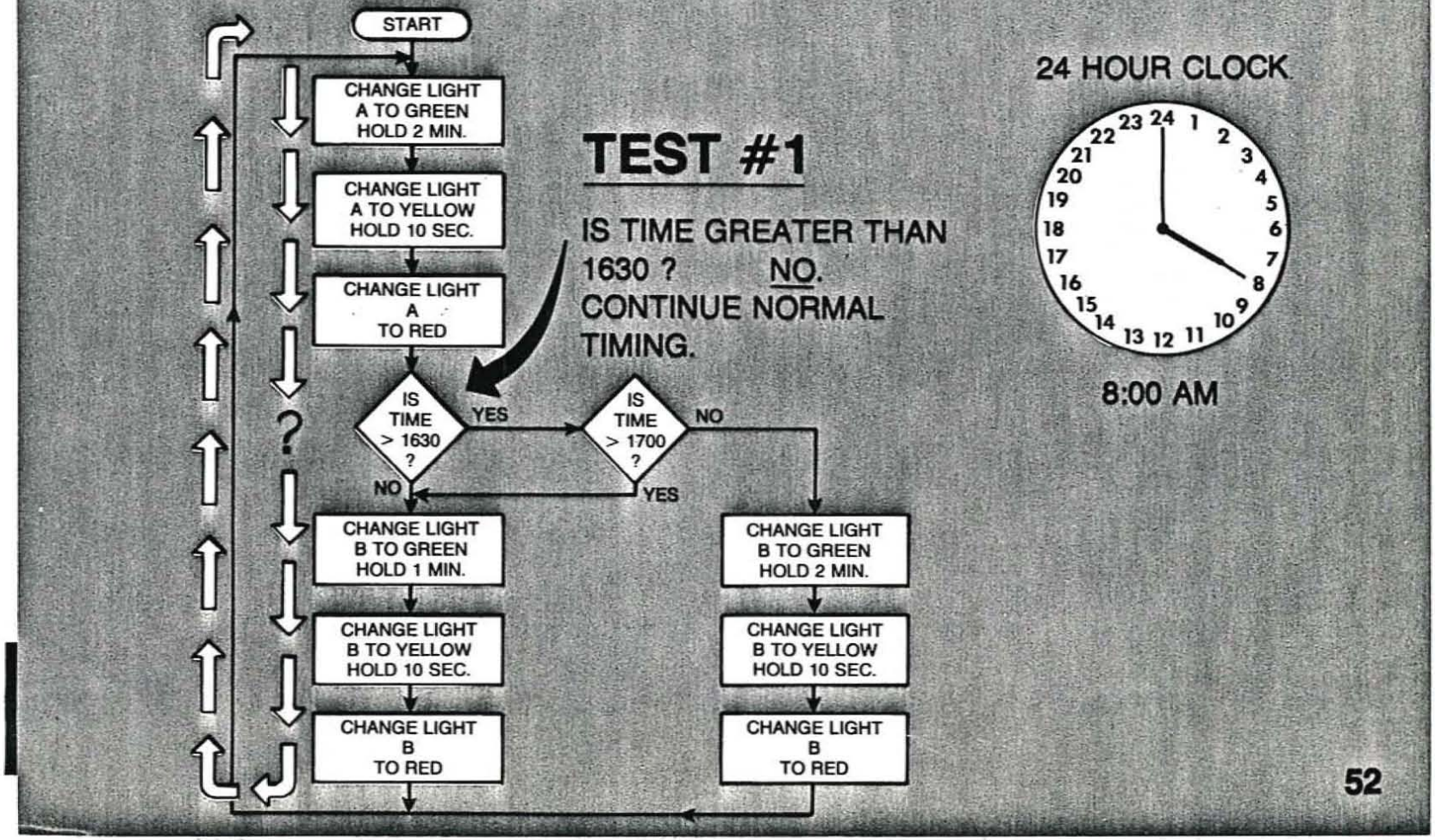
5. Light "B's" timing cycle is changed to 2 minutes at 4:30 PM and returned to 1 minute at 5:00 PM.

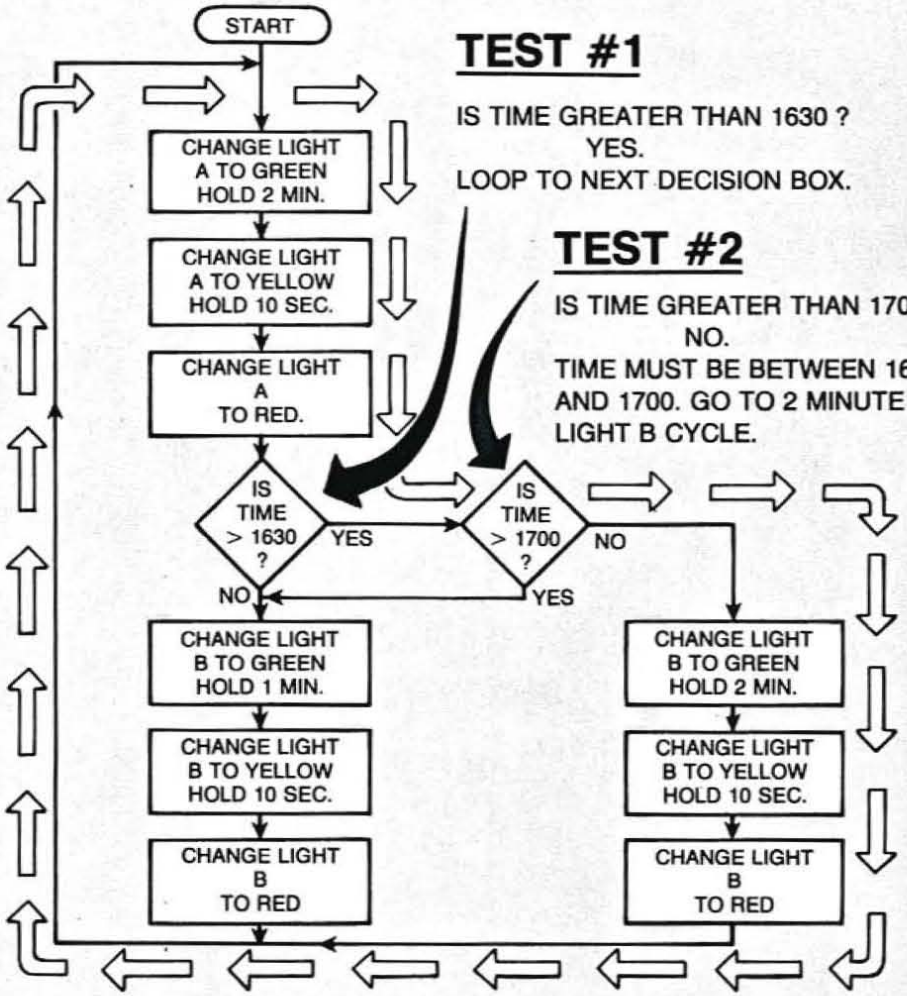


24 HOUR CLOCK

4:30 PM = 1630

5:00 PM = 1700


START
CHANGE LIGHT A TO GREEN HOLD 2 MIN.
CHANGE LIGHT A TO YELLOW HOLD 10 SEC.
CHANGE LIGHT A TO RED
IS TIME > 1630 ?
YES
IS TIME > 1700 ?
NO
YES
NO
CHANGE LIGHT B TO GREEN HOLD 1 MIN.
CHANGE LIGHT B TO YELLOW HOLD 10 SEC.
CHANGE LIGHT B TO RED
CHANGE LIGHT B TO GREEN HOLD 2 MIN.
CHANGE LIGHT B TO YELLOW HOLD 10 SEC.
CHANGE LIGHT B TO RED
TEST #1
IS TIME GREATER THAN 1630 ? NO.
CONTINUE NORMAL TIMING.
24 HOUR CLOCK
8:00 AM

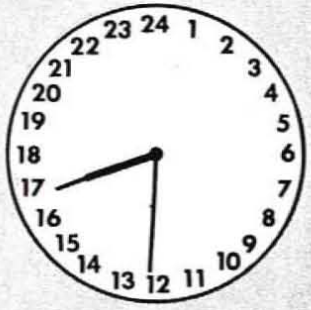


TEST #1

IS TIME GREATER THAN 1630 ?
YES.
LOOP TO NEXT DECISION BOX.

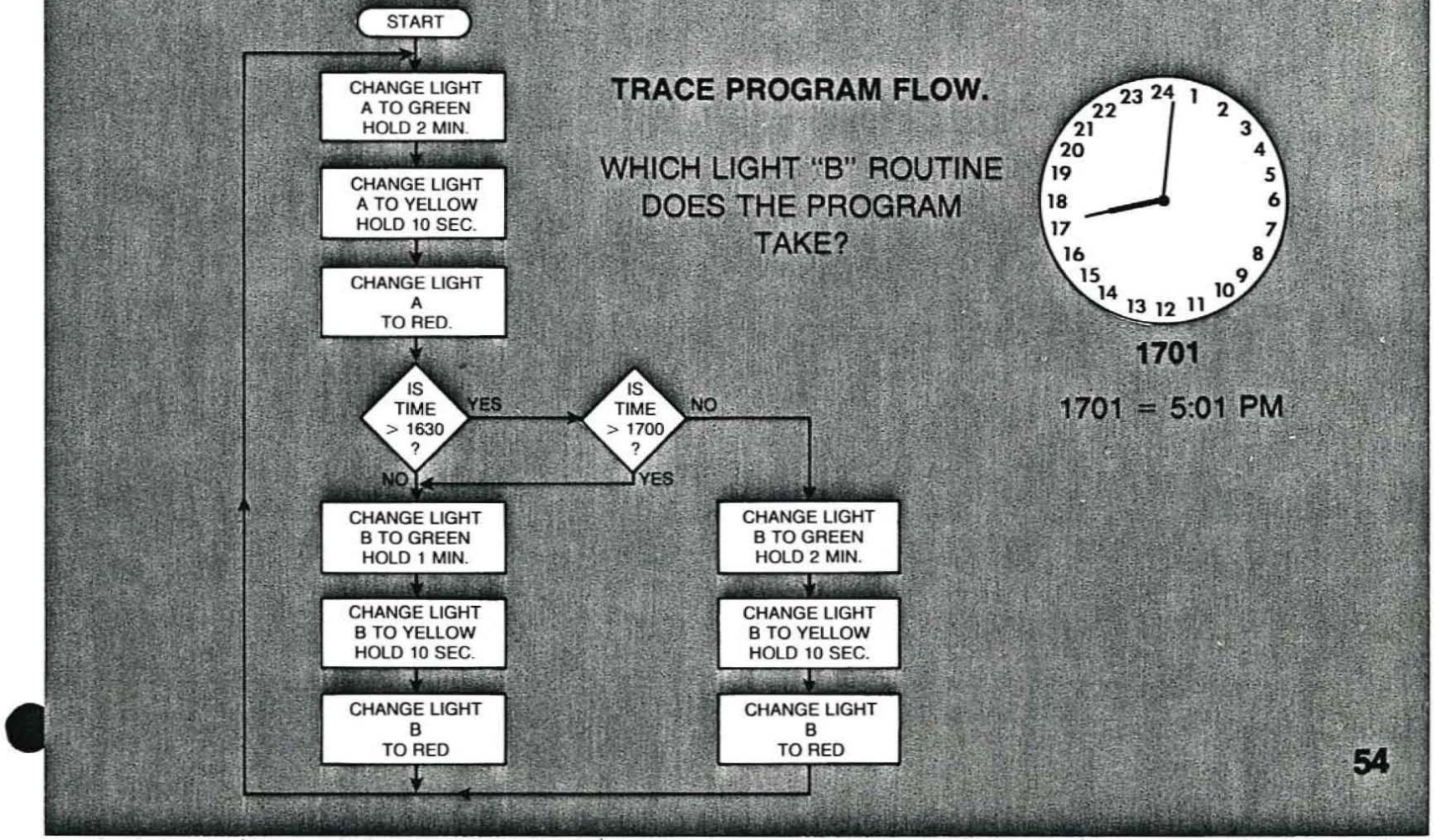
TEST #2

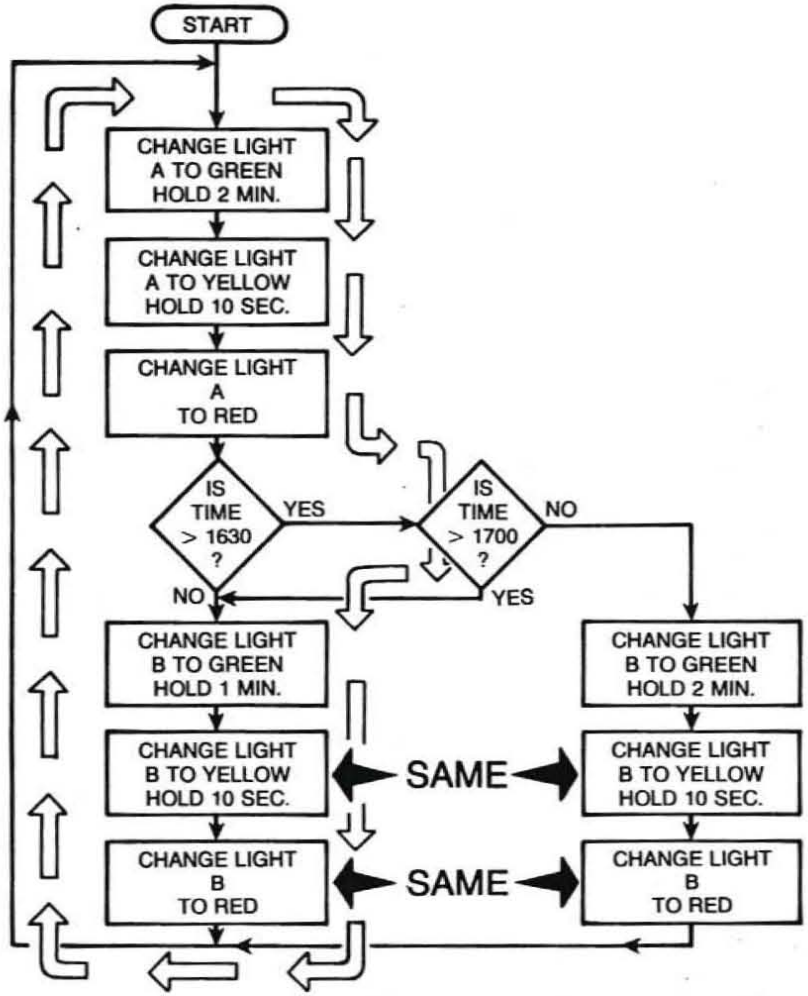
IS TIME GREATER THAN 1700 ?
NO.
TIME MUST BE BETWEEN 1630 AND 1700. GO TO 2 MINUTE LIGHT B CYCLE.



1631

1631 = 4:31 PM
1700 = 5:00 PM


START
CHANGE LIGHT A TO GREEN HOLD 2 MIN.
CHANGE LIGHT A TO YELLOW HOLD 10 SEC.
CHANGE LIGHT A TO RED.
IS TIME > 1630 ?
YES
NO
IS TIME > 1700 ?
NO
YES
CHANGE LIGHT B TO GREEN HOLD 1 MIN.
CHANGE LIGHT B TO YELLOW HOLD 10 SEC.
CHANGE LIGHT B TO RED
CHANGE LIGHT B TO GREEN HOLD 2 MIN.
CHANGE LIGHT B TO YELLOW HOLD 10 SEC.
CHANGE LIGHT B TO RED
TRACE PROGRAM FLOW.
WHICH LIGHT "B" ROUTINE DOES THE PROGRAM TAKE?
1701
1701 = 5:01 PM
54

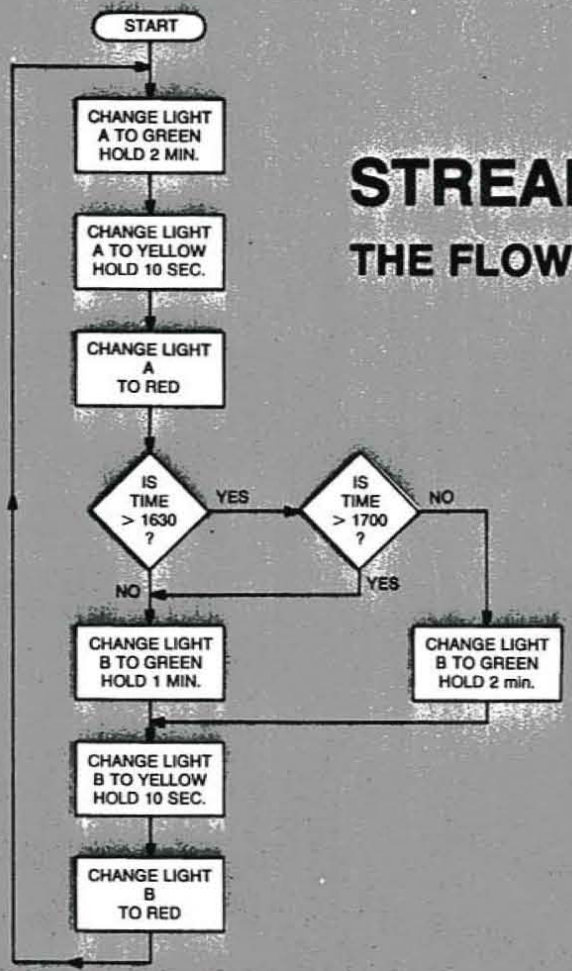

START
CHANGE LIGHT A TO GREEN HOLD 2 MIN.
CHANGE LIGHT A TO YELLOW HOLD 10 SEC.
CHANGE LIGHT A TO RED
IS TIME > 1630 ?
YES
IS TIME > 1700 ?
NO
NO
YES
CHANGE LIGHT B TO GREEN HOLD 1 MIN.
CHANGE LIGHT B TO GREEN HOLD 2 MIN.
CHANGE LIGHT B TO YELLOW HOLD 10 SEC.
SAME
CHANGE LIGHT B TO YELLOW HOLD 10 SEC.
CHANGE LIGHT B TO RED
SAME
CHANGE LIGHT B TO RED


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

1701

STREAMLINING

THE FLOWCHART



INFORMATION SUMMARY

Mechanical alternating current generators combine circular mechanical motion with magnetism to produce an alternating current electromotive force (voltage).

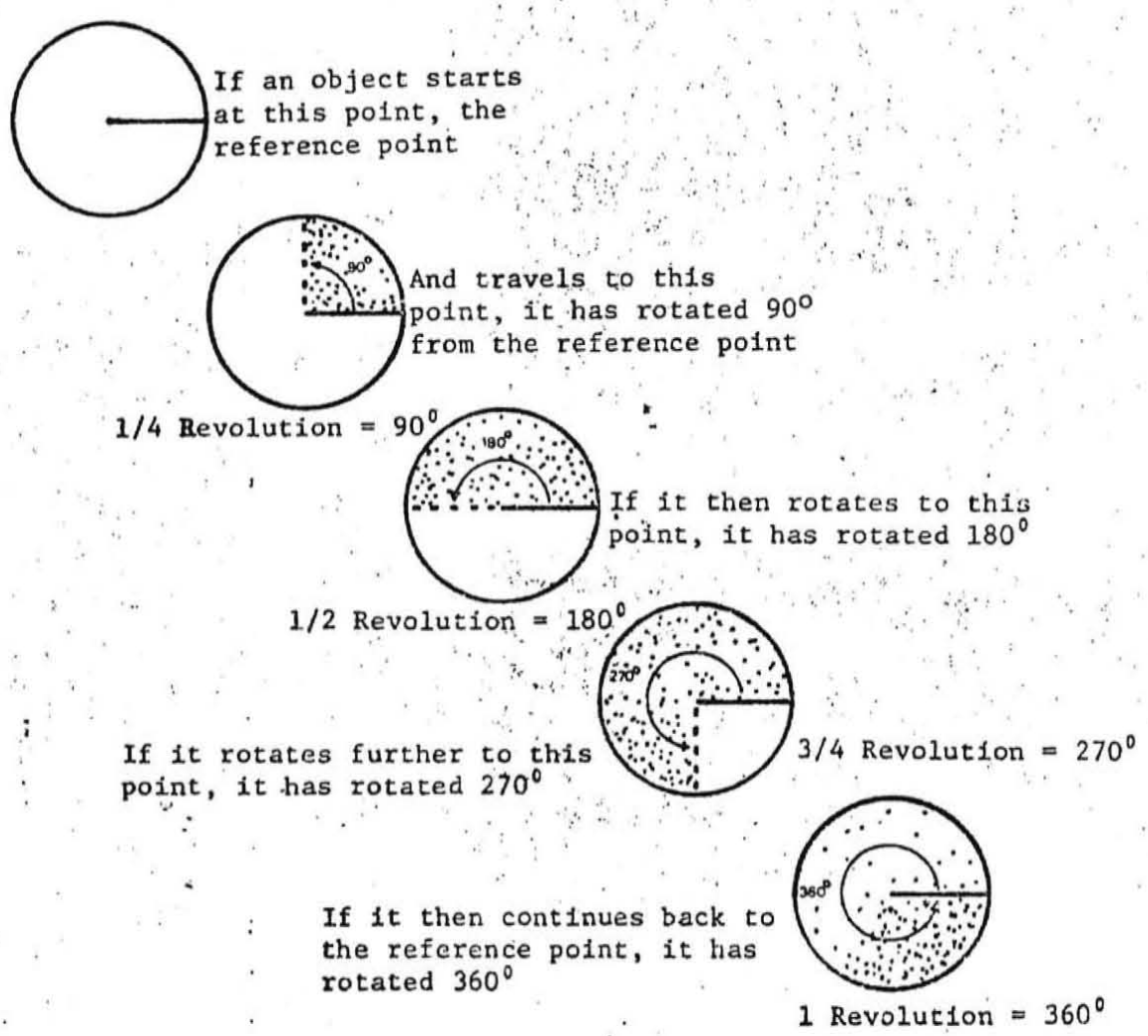
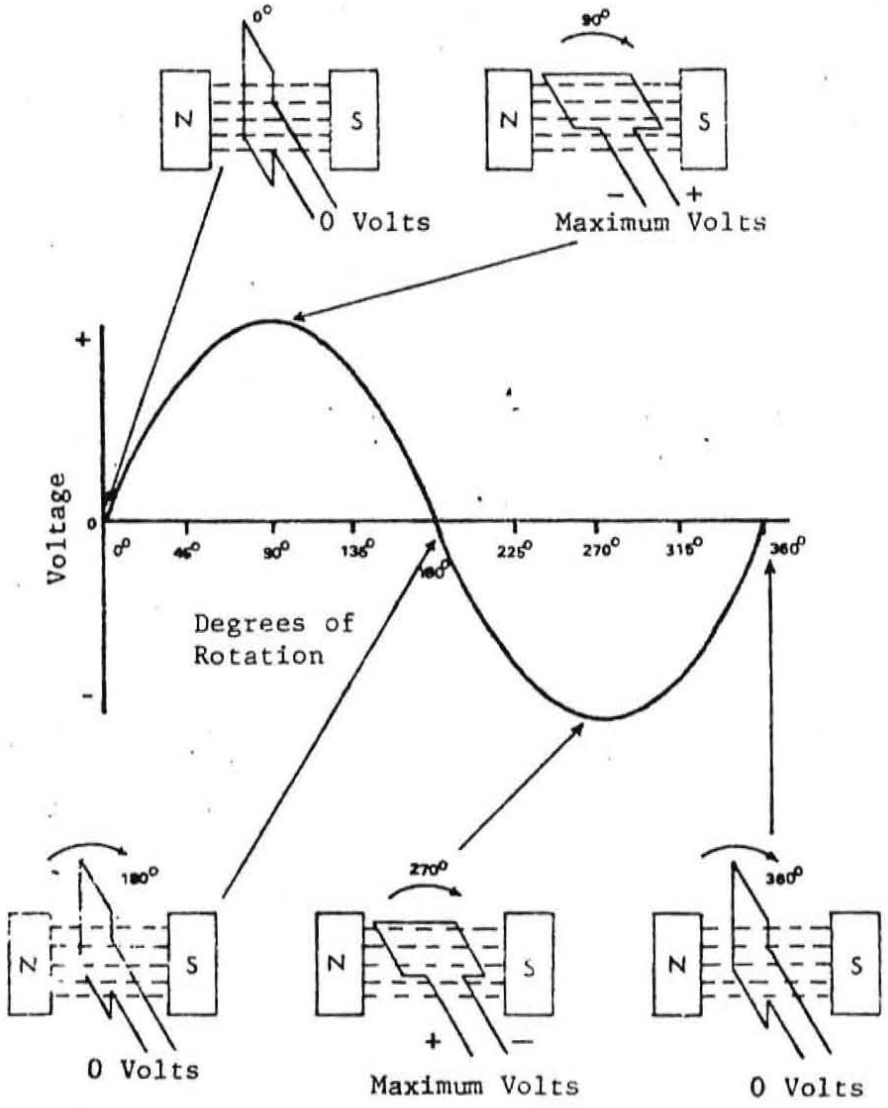


Figure 1 Each cycle can be considered a complete sequence of events--in the above case, from 0° through 180° and on around to 360° or 0°.

A loop of wire or armature starting from any position and rotated at a uniform speed in a clockwise or counterclockwise direction is forced to cut the lines of force in a magnetic field. See Figure 2. As the armature rotates, it cuts a varying number of lines of force. As the number of lines of force being cut increases, the induced electromotive force increases. As the armature rotates and less lines of force are cut, the induced electromotive force decreases. This action is repeated again and again as the generator continues to rotate. A cycle or a complete series of events is accomplished each time the armature makes a full revolution.



As the generator armature rotates, the magnitude and polarity of the voltage produced follows the pattern of the sine wave

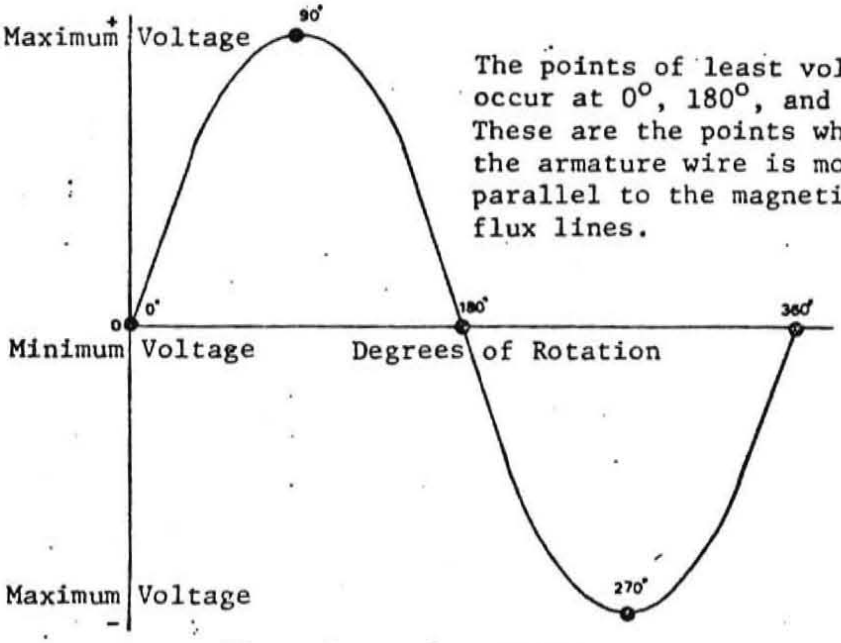
Figure 2

Information Summary, continued

The voltage produced by the simple alternating current generator has a characteristic waveform that is important to alternating current theory. The waveform describes the output voltage of the generator. See Figure 3.

Since a sine wave corresponds to one cycle and there are 360° in one cycle, there are 360° in one complete sine wave.

SINE WAVE



The points of least voltage (0 volts) occur at 0°, 180°, and 360°. These are the points where the armature wire is moving parallel to the magnetic flux lines.

Figure 3

The points of greatest voltage occur at 90° 270°. The armature wires are moving at right angles to the magnetic flux lines and the rotating wires are cutting the maximum number of lines of flux per degree of rotation.

Information Summary, continued

It should be remembered that alternating current reverses in polarity from plus to minus to plus each half cycle. Note Figure 3. The waveform that describes this variation is called a sine wave which means that the voltage generated at any point in the armature wire or coil travel is proportional to the angle between the magnetic flux lines and the direction of motion of the armature coil. See Figure 4.

Waveforms can be plotted for either current or voltage. This graph shows how the current or voltage varies with time.

For a-c power sources, waveforms can also show how the output voltage varies with one complete cycle of 360°.

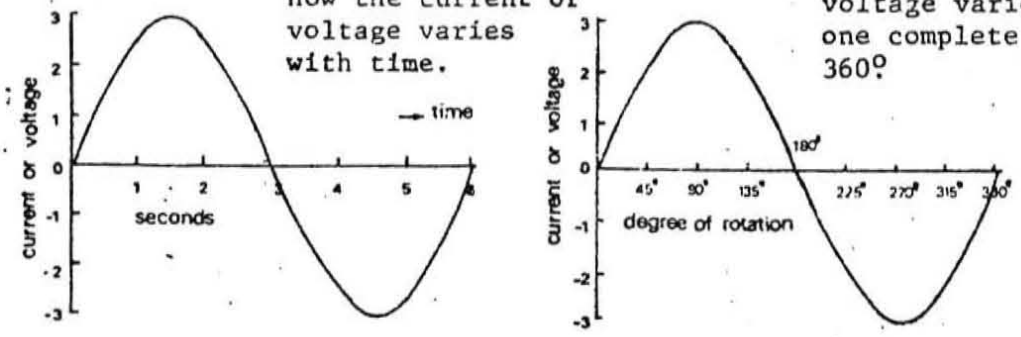


Figure 4

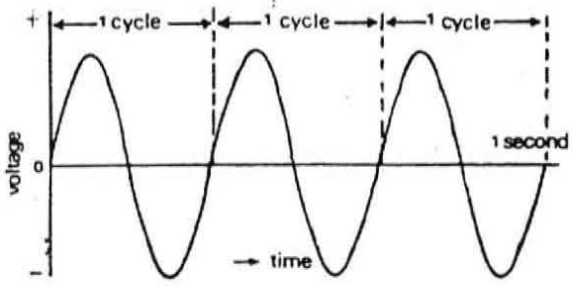
Since alternating current has a sine wave as its characteristic waveform, it is necessary to understand the meaning of the terms used to describe the sine waveform.

In the alternating current waveform, the voltage or current changes from zero to a maximum value and back to zero in a positive direction. The same change or pattern takes place in a negative direction. Increasing from zero to a maximum positive value, then decreasing to zero and increasing to a maximum negative value, and then decreasing again to zero describes one complete cycle of the sine wave.

Information Summary, continued

The number of complete sine waves generated in one second is the frequency of the voltage or current and is expressed in cycles per second (cps) or the currently used standard, Hertz (Hz).

The greater the number of cycles generated per second, the higher the frequency. A generator rotating at a speed sufficient to produce sixty alternations of current each second is producing electrical current or voltage at a frequency of 60 Hz. If the generator were speeded up to produce 120 alternations, the frequency would be 120 Hz. See Figure 5.



The frequency of a voltage or current is the number of cycles generated each second. The frequency of this voltage is, therefore, 3 cps.

Figure 5

The following chart indicates some of the units commonly used to indicate frequency.

1 Kilocycle/sec (1 kc) - 1 KiloHertz (1 kHz) = 1000 cps or 1000 Hz

1 Megacycle/sec (1 Mc) - 1 MegaHertz (1 MHz) = 1,000,000 cps or 1,000,000 Hz

1 Gigacycle (1 Gc) - 1 GigaHertz (1 GHz) = 1,000,000,000 cps or 1,000,000,000 Hz

Hz is now an internationally accepted standard for measuring frequency and will be used herein.

Information Summary, continued

Electric current in a circuit travels at the speed of light, (186,000 miles per second or 300,000,000 meters per second) and since the speed is constant, current can only travel a given distance in a given period of time. Since frequency is a measure of the number of cycles for a given period of time, the distance traveled by the current during one cycle can be calculated. This distance is called WAVELENGTH. See Figure 6. It takes 1/60th of a second to complete one full cycle of AC power line voltage. The wavelength of 60 Hz voltage is 1/60 x 186,000 or 3100 miles. The wavelength of 60 KHz voltage is $\frac{186,000}{60K}$ or 3.1 miles.

Wavelength is equal to velocity of current divided by the frequency.

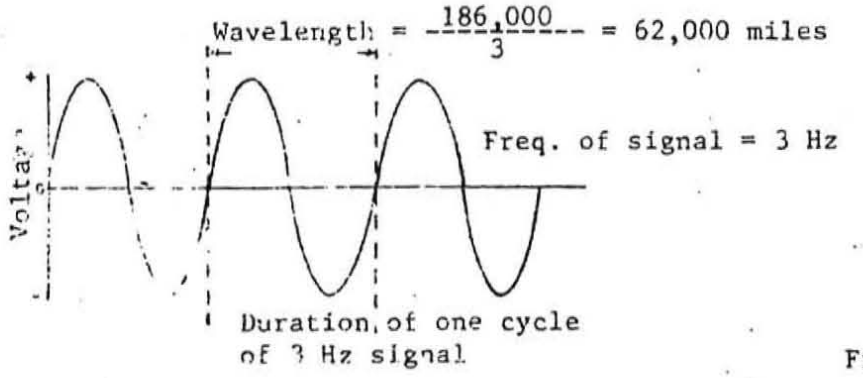


Figure 6

If the frequency is 3 Hz, and the velocity of the electricity along the wire is 186,000 miles per second, the wavelength is the velocity divided by frequency: $\frac{186,000}{3}$ = 62,000 miles.

The output of an AC generator varies as a sine wave. Two generators of the same type and size will each generate at least one full sine wave for each full revolution. If they are started at the same time and are run at the same speed, the two waveforms begin together and end together.

Information Summary, continued

They will reach their maximum values and pass through zero at the same time. The two waveforms are in step and can be superimposed upon each other. The two output currents or voltages represented by such in-step sine waves are said to be in phase. See Figure 7.

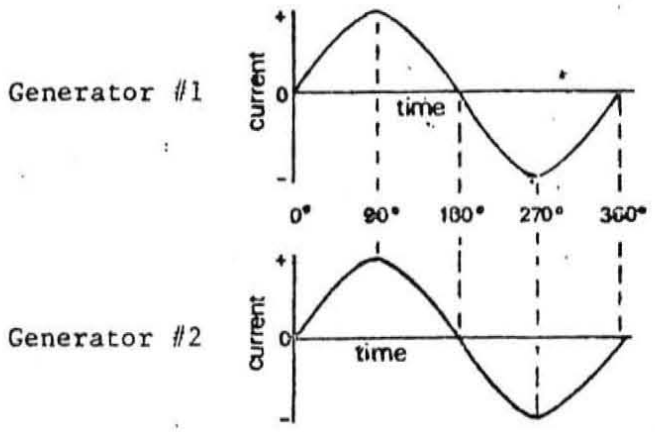


Figure 7

Two currents or voltages are in phase if they reach their peak magnitudes at the same instant. Their peak magnitudes may be of different values however. See Figure 8.

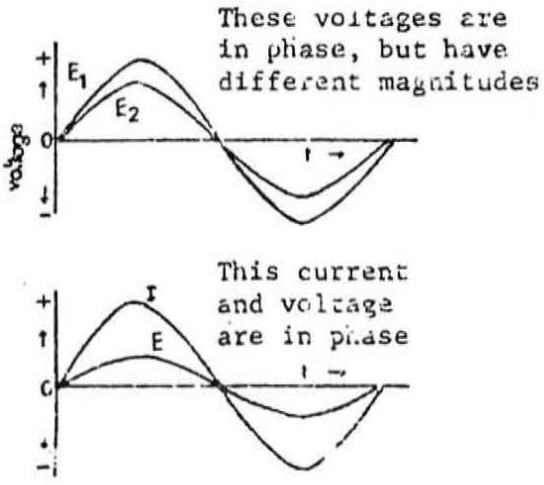
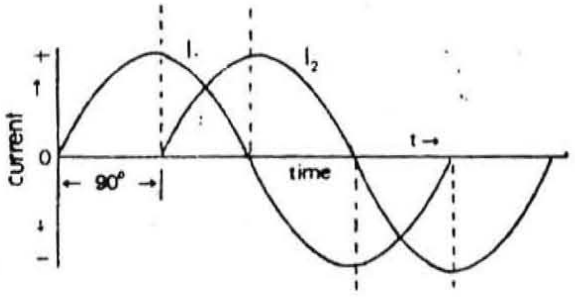


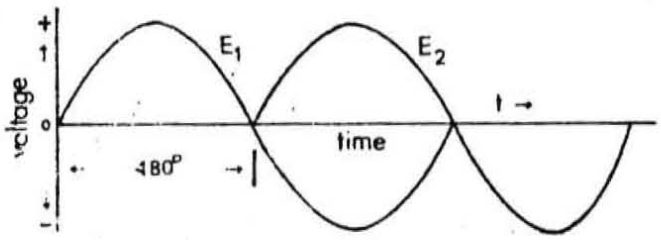
Figure 8

Information Summary, continued

If one generator is started after the other, maximum and minimum output values will occur after corresponding values of the other generator. The two outputs are said to be out of phase or that a phase difference exists. See Figure 9.



These two currents have the same frequency but I_1 leads I_2 by 90° or I_2 lags I_1 by 90°



These two signals have the same amplitude and frequency but E_1 leads E_2 by 180° or E_2 lags E_1 by 180°

Figure 9

The terms "lead" and "lag" are used to indicate the position of the current or voltage with respect to time.

Information Summary, continued

An alternation is, in general, one-half a cycle, or 180°.

The amplitude, or peak value is the maximum positive and negative values of an a-c voltage or current.

The period is the time required by one full cycle of a-c voltage or current.

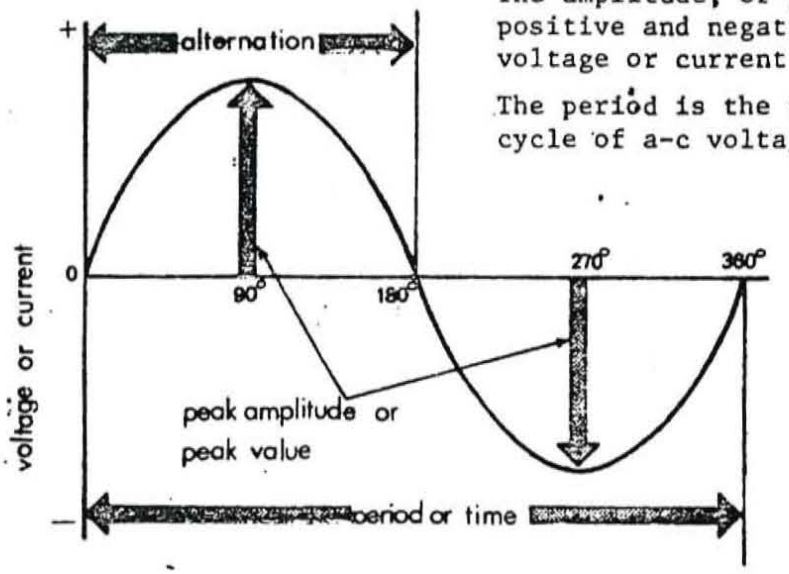


Figure 10

The above diagram describes the terms "alternation", "amplitude", and "period". The peak value of the amplitude is obtained by measuring the current or voltage with a peak reading meter. The period is determined by dividing the frequency into one.

$$\text{period} = \frac{1}{\text{frequency}}$$

Information Summary, continued

The period is in seconds and the frequency is in cycles per second.

The values of AC voltages and currents are constantly changing. The most obvious value is the peak value or a measure of the maximum amplitude of the sine wave. On a waveform, the distance from maximum positive value to maximum negative value is the peak-to-peak value. See Figure 11.

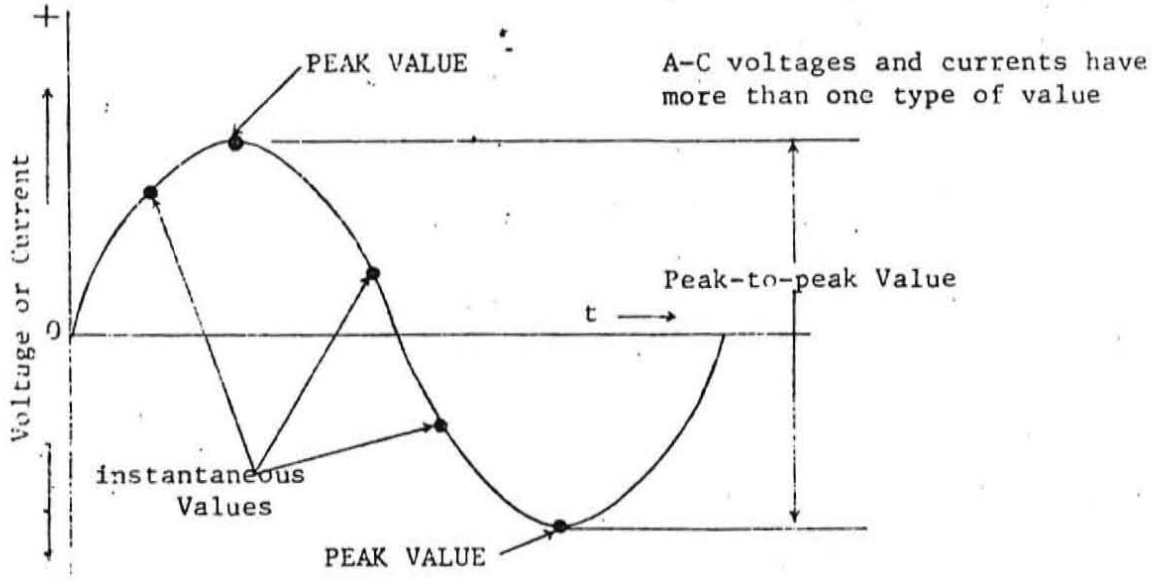


Figure 11

The instantaneous values are the values of the current or voltage at the instant they are measured (as seen on oscilloscope). This value would vary

Information Summary, continued

from zero to peak value in both the positive and negative direction. As a rule, peak-to-peak, peak, and instantaneous values are not satisfactory for expressing values of voltage or current. Instead, the average value and the effective values are generally used. See Figure 12.

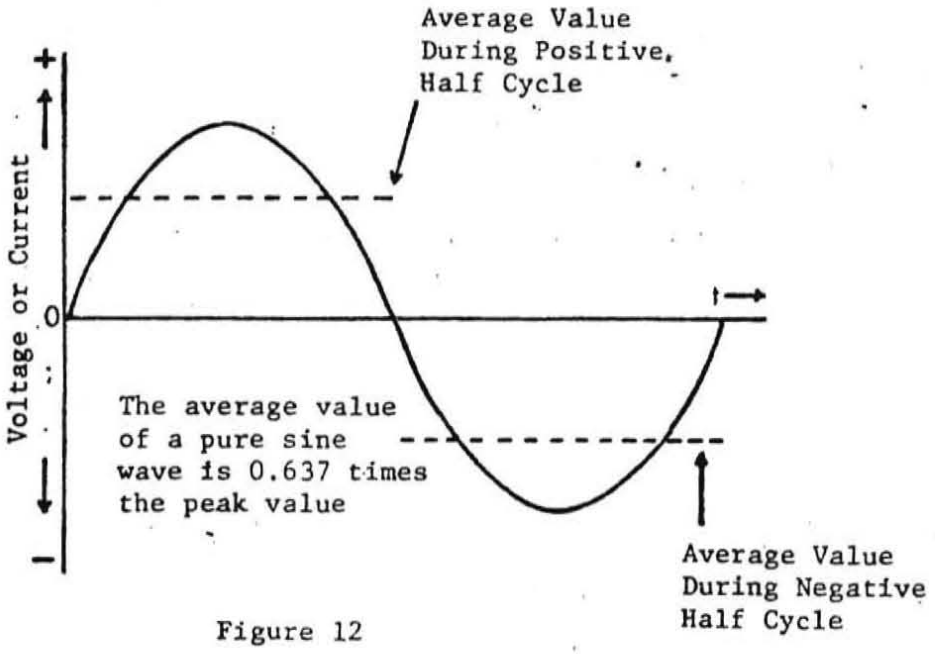


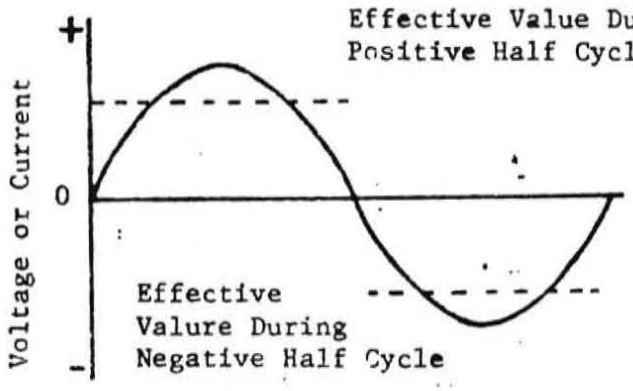
Figure 12

The average value of AC voltage or current is an average of all the instantaneous values during one half cycle or alternation.

Average voltage is determined by multiplying peak voltage by a factor of 0.637.

A circuit with a peak voltage of 100 volts has an average voltage of 63.7 volts.

Information Summary, continued



The effective (rms) value of a pure sine wave is 0.707 times the peak value

Eeff= 0.707 x E peak
or
Ieff= .707 x I peak

Figure 13

The effective value of an AC voltage or current is useful in expressing AC currents and voltages in values related to DC. The effective value of an AC voltage or current is that value which will produce the same amount of heat in a circuit containing only resistance that would be caused by a DC voltage or current of the same value.

The effective value of an AC voltage or current is expressed as .707 times the peak value. The effective value of a 100 volt AC circuit is 70.7 volts.

Information Summary, continued

The following table is used to convert one AC value to another.

To Convert From	To	Use the Equations	
Peak	Average	$E_{AV} = 0.637\ E_{PK}$	$I_{AV} = 0.637\ I_{PK}$
Peak	Effective	$E_{EFF} = 0.707\ E_{PK}$	$I_{EFF} = 0.707\ I_{PK}$
Average	Peak	$E_{PK} = 1.57\ E_{AV}$	$I_{PK} = 1.57\ I_{AV}$
Average	Effective	$E_{EFF} = 1.11\ E_{AV}$	$I_{EFF} = 1.11\ I_{AV}$
Effective	Peak	$E_{PK} = 1.414\ E_{EFF}$	$I_{PK} = 1.414\ E_{EFF}$
Effective	Average	$E_{AV} = 0.9\ E_{EFF}$	$I_{AV} = 0.9\ I_{EFF}$

INFORMATION

INFORMATION SHEET #1

THE LABORATORY OSCILLOSCOPE

The lab scope uses the same basic principles as the general purpose scope. It has been designed to provide greater accuracy and stability, wider frequency response, less distortion, and more flexibility in use. Many special features are commonly included. In this information sheet, you will begin to learn some of the typical lab scope features. It is assumed that you have studied the package on "General Purpose Oscilloscopes."

1. Set-up controls for the trace.

   A. The focus,intensity, horizontal and vertical position controls serve the same function as in the general purpose scope.

   B. Astigmatism--This is a new control. Few general purpose scopes provide this. This control provides a sharper focus at all parts of the screen.

2. Horizontal features.

   A. Triggered sweep
   The sweep circuit in the lab scope is switched on or triggered by the incoming signal. This provides a much more stable display. The general purpose scope requires an adjustment of the frequency of the sweep generator to a value near that of the frequency of the input signal. This is not necessary in the lab scope. As a result the display can be more quickly stabilized.

      a. Controls--There are usually two controls, often called TRIGGER LEVEL and STABILITY, although other names are used. The details of these and other controls are in your scope's instruction manual. These two controls adjust the sensitivity of the trigger circuit and the voltage at which it triggers.

Information Sheet #1, con't

b. Trigger selection--Most lab scopes have provisions for selecting various trigger signals. Some of these are positive or negative, internal, external, high frequency, TV line or frame.

B. Calibrated time base

a. The general purpose scope does not readily provide time information about the signal being observed. The sweep speeds in the lab scope are calibrated in time per centimeter. This allows the time of a certain waveform to be measured. The sweep speed selector setting multiplied times the length, in centimeters, of the waveform gives the time.
NOTE: Many lab scopes provide a variable sweep speed control. This must be in the CAL (calibrated) position for the sweep speeds to be calibrated and to make time measurements.

b. Many lab scopes have auxiliary switched sweep speed controls. These may increase or decrease the sweep speed by a definite factor. Time measurements are still possible providing the position of these switches is considered. Sweep magnifiers are examples of this. These increase the sweep speed to "magnify" the display and show fine details more clearly.

3. Vertical Features
Sweep speeds can be as slow as seconds per centimeter and as fast as a few nanoseconds per centimeter.

A. Calibrated Vertical Amplifiers

a. The general purpose scope must be calibrated with an external signal or else an internal switched signal. If the vertical gain control is changed, the scope is no longer calibrated. As long as any variable control is in the calibrated position, the scope need not be calibrated for different sensitivities. The vertical sensitivity of a lab scope is measured in volts per centimeter. For example,

Information Sheet #1, con't

A scope might have these sensitivities or ranges: 10 mv/cm, 50 mv/cm, 100 mv/cm and so on up to perhaps 50 v/cm. The maximum sensitivity, sometimes called the deflection factor, in this case would be 10 mv/cm.

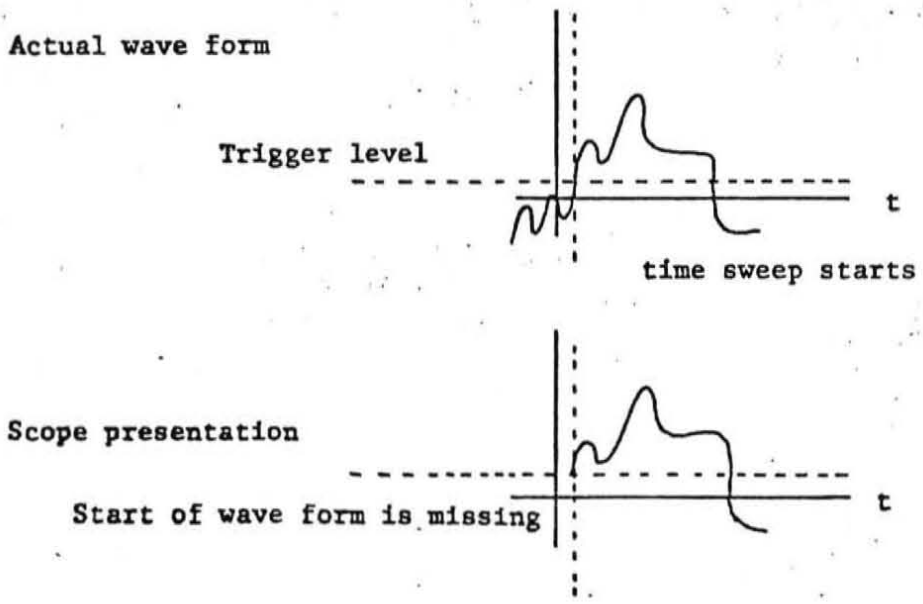
The maximum sensitivity is related to the frequency response or bandwidth. Generally a scope with two sensitivity ranges will have a narrower frequency response at its highest sensitivity. Here is an example:

Sensitivity	Frequency Response
100 mv/cm	DC - 15 MHz
10 mv/cm	DC - 8 MHz

b. Plug-in vertical amplifiers are features of some lab scopes. These allow great flexibility in using the basic scope. Amplifiers with different sensitivities, frequency responses, and special features are available.

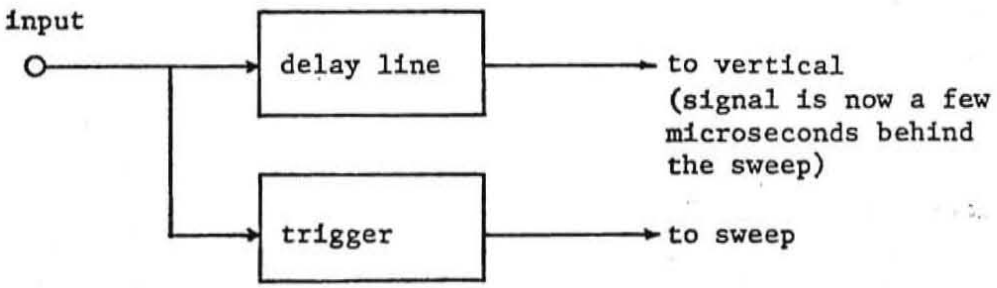
c. Another specification of the vertical amplifier is the "rise time". This is important in scopes which are to be used to measure the time a rapidly rising signal takes to rise. This can be in the nanosecond range!

4. Other Features
   A. Delayed presentation
   The sweep, which is triggered by the signal, starts after the signal has already reached the trigger level. The result is shown below:

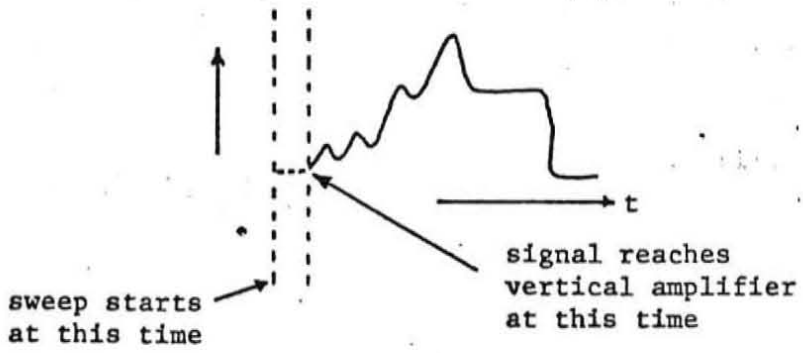


Information Sheet #1, con't

A delay line, which causes the input signal to delay a small time between the time it triggers the sweep and the time the signal starts being displayed eliminates this.



The presentation now becomes:



Information Sheet #1, con't

B. Dual Trace and Dual Beam

a. A dual beam scope has two separate beams with separate vertical amplifiers. It may have a common sweep or separate sweeps.

b. A dual trace scope has one beam which is electronically switched back and forth between two separate inputs. This is done; rapidly during the sweep (chopped) or at the start of each sweep(alternate).

C. Single sweep

For observation, often photographic, of a non-repetitive wave or a wave which varies its characteristics, a single sweep feature is useful. This allows "cocking" the sweep trigger which is "fired" when the wave appears. The scope sweeps once only and must be reset for another sweep.

D. Scope Photography

A permanent record is often made of a waveform with a camera. The usual camera is a specially adapted Polaroid camera. This allows the user to examine his photographs immediately and retake them if necessary.

E. Storage Oscilloscopes

The storage scope allows a waveform to be kept on the screen while it is being studied or until a second waveform is available for comparison. Then the waveform can be erased electronically. The waveforms can be stored for up to several hours.

F. Spectrum Analyzers

These devices display the energy distribution of a signal vertically against the frequencies within the signal on the horizontal axis. This is extremely useful in pulse and radio frequency work.

G. Miscellaneous

Space does not permit description of the many more special features which are available. You may wish to learn more about them on your own. Some of these are:

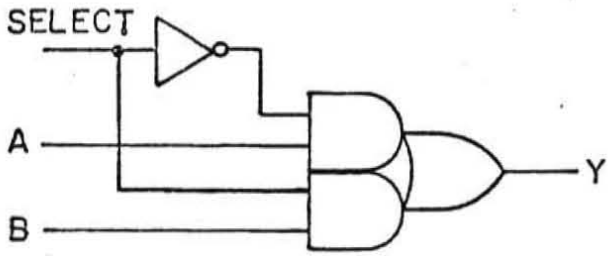
1. differential vertical amplifiers
2. time domain reflectrometry
3. phosphor types
4. beam finders
5. voltage and current probes

6.5 MULTIPLEXORS

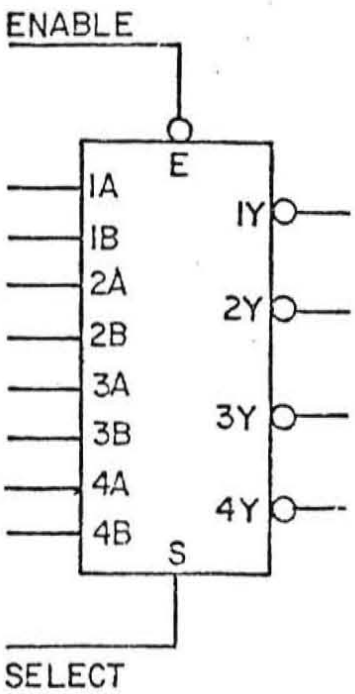
Multiplexors are defined as circuits which select one of two or more inputs to appear on the output. Multiplexors are equivalent to electronic rotary switches and are available in 2 to 1, 4 to 1, 8 to 1, and 16 to 1 varieties.

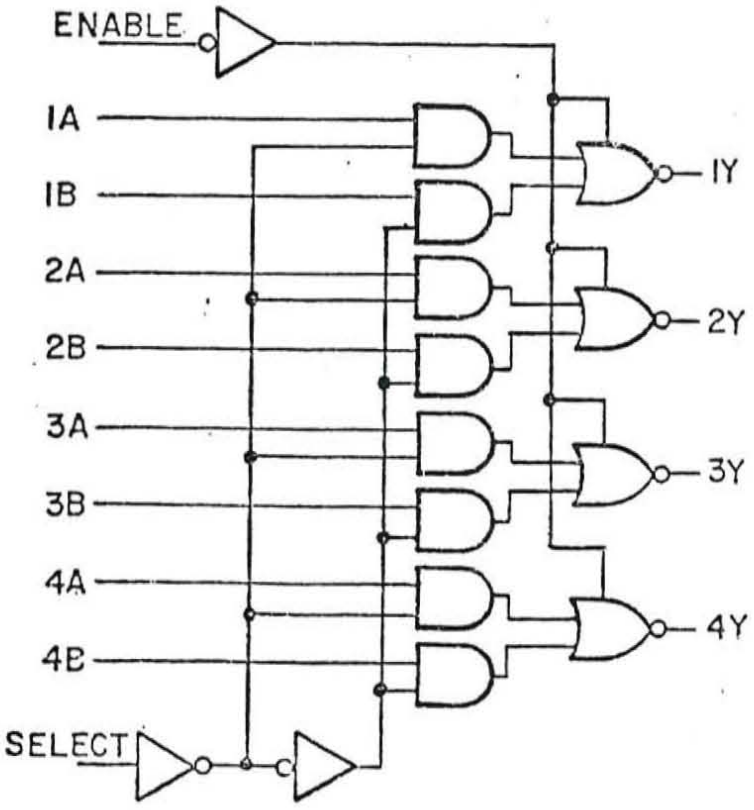
6.5.1 Two-to-One Multiplexors

A two-to-one multiplexor which selects one of two inputs to appear at the output is shown below.



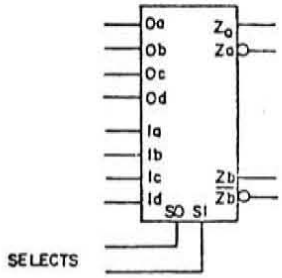
Four 2 to 1 mux's are contained in one IC, and the entire IC logic is illustrated below.

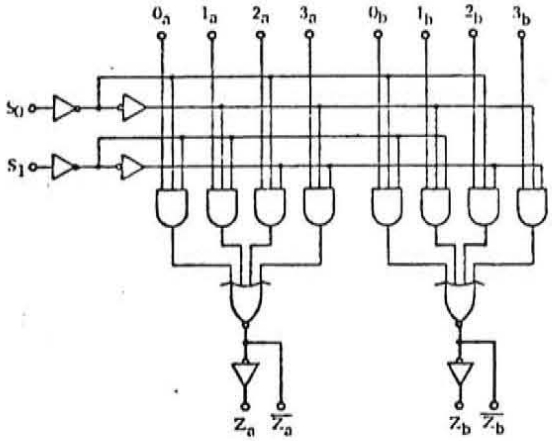




6.5.2 Dual 4 to 1 Multiplexor

The dual 4 to 1 multiplexor illustrated below selects one of four inputs to appear at the output.





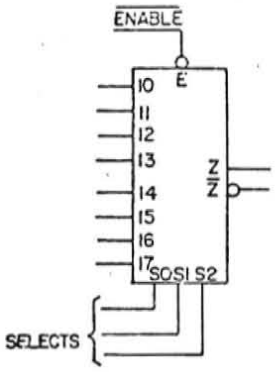
Truth Table

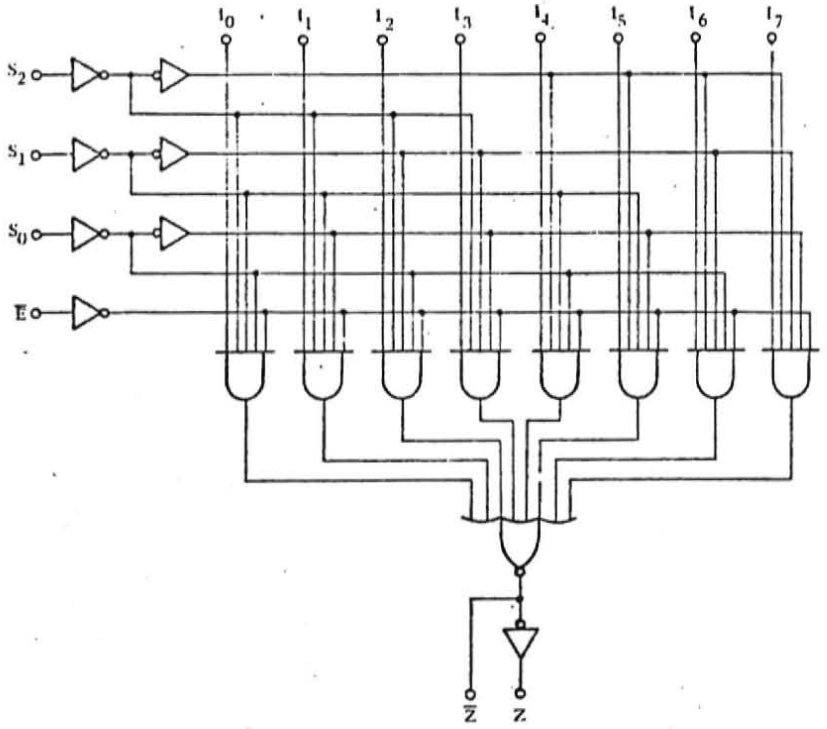
Select Inputs		Inputs				Outputs	
S_0	S_1	I_{0a}	I_{1a}	I_{2a}	I_{3a}	Za	$\overline{Z}a$
L	L	L	X	X	X	L	H
L	L	H	X	X	X	H	L
H	L	X	L	X	X	L	H
H	L	X	H	X	X	H	L
L	H	X	X	L	X	L	H
L	H	X	X	H	X	H	L
H	H	X	X	X	L	L	H
H	H	X	X	X	H	H	L
S_0	S_1	I_{0b}	I_{1b}	I_{2b}	I_{3b}	Zb	$\overline{Z}b$
L	L	L	X	X	X	L	H
L	L	H	X	X	X	H	L
H	L	X	L	X	X	L	H
H	L	X	H	X	X	H	L
L	H	X	X	L	X	L	H
L	H	X	X	H	X	H	L
H	H	X	X	X	L	L	H
H	H	X	X	X	H	H	L

L = LOW Voltage Level
H = HIGH Voltage Level
X = Either HIGH or LOW Logic Level

6.5.3 8 to 1 Multiplexor

The 8 to 1 multiplexor illustrated below selects one of eight inputs to appear at the output.





Truth Table

$\overline{E}$	S_2	S_1	S_0	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	$\overline{Z}$	Z
H	X	X	X	X	X	X	X	X	X	X	X	H	L
L	L	L	L	L	X	X	X	X	X	X	X	H	L
L	L	L	L	H	X	X	X	X	X	X	X	L	H
L	L	L	H	X	L	X	X	X	X	X	X	H	L
L	L	L	H	X	H	X	X	X	X	X	X	L	H
L	L	H	L	X	X	L	X	X	X	X	X	H	L
L	L	H	L	X	X	H	X	X	X	X	X	L	H
L	L	H	H	X	X	X	L	X	X	X	X	H	L
L	L	H	H	X	X	X	H	X	X	X	X	L	H
L	H	L	L	X	X	X	X	L	X	X	X	H	L
L	H	L	L	X	X	X	X	H	X	X	X	L	H
L	H	L	H	X	X	X	X	X	L	X	X	H	L
L	H	L	H	X	X	X	X	X	H	X	X	L	H
L	H	H	L	X	X	X	X	X	X	L	X	H	L
L	H	H	L	X	X	X	X	X	X	H	X	L	H
L	H	H	H	X	X	X	X	X	X	X	L	H	L
L	H	H	H	X	X	X	X	X	X	X	H	L	H

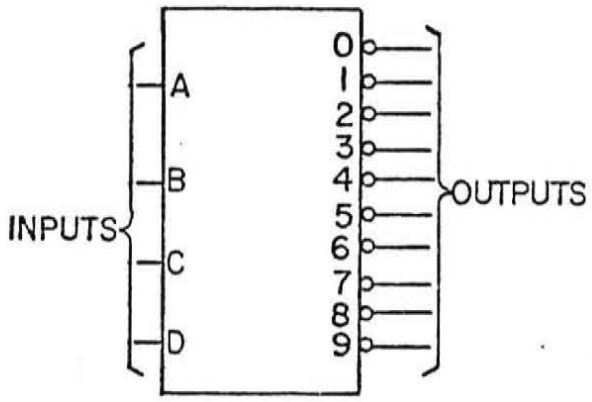
H = HIGH voltage level
L = LOW voltage level
X = Level does not affect output.

6.6 DECODERS/DEMULTIPLEXORS

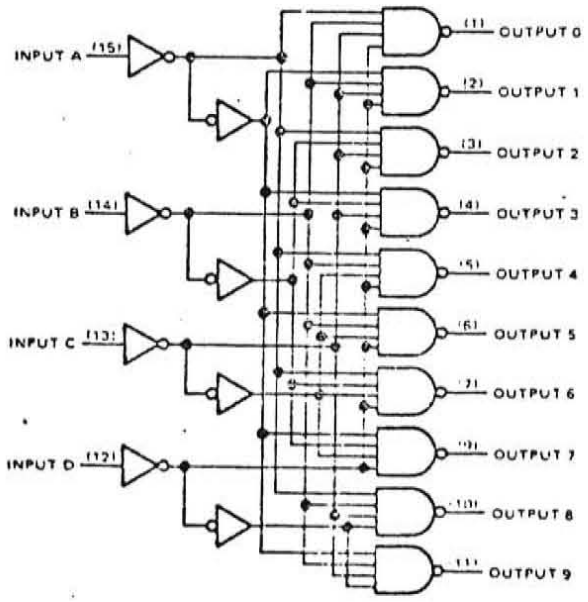
Decoders/Demultiplexors are defined as circuits which convert from a binary code to a decimal system or from one binary code to another. There are various decoders/demultiplexors available. We will examine only one, the BCD-to-Decimal Decoder.

6.6.1 BCD-To-Decimal Decoder

This decoder takes a four bit binary digit input and converts it into one of ten possible outputs. The circuit design insures that all outputs are high when binary codes greater than nine (9) are applied to the inputs.



Logic Diagram



Function Table

	Inputs				Outputs									
No.	D	C	B	A	0	1	2	3	4	5	6	7	8	9
0	L	L	L	L	L	H	H	H	H	H	H	H	H	H
1	L	L	L	H	H	L	H	H	H	H	H	H	H	H
2	L	L	H	L	H	H	L	H	H	H	H	H	H	H
3	L	L	H	H	H	H	H	L	H	H	H	H	H	H
4	L	H	L	L	H	H	H	H	L	H	H	H	H	H
5	L	H	L	H	H	H	H	H	H	L	H	H	H	H
6	L	H	H	L	H	H	H	H	H	H	L	H	H	H
7	L	H	H	H	H	H	H	H	H	H	H	L	H	H
8	H	L	L	L	H	H	H	H	H	H	H	H	L	H
9	H	L	L	H	H	H	H	H	H	H	H	H	H	L
Invalid	H	L	H	L	H	H	H	H	H	H	H	H	H	H
	H	L	H	H	H	H	H	H	H	H	H	H	H	H
	H	H	L	L	H	H	H	H	H	H	H	H	H	H
	H	H	L	H	H	H	H	H	H	H	H	H	H	H
	H	H	H	L	H	H	H	H	H	H	H	H	H	H
	H	H	H	H	H	H	H	H	H	H	H	H	H	H

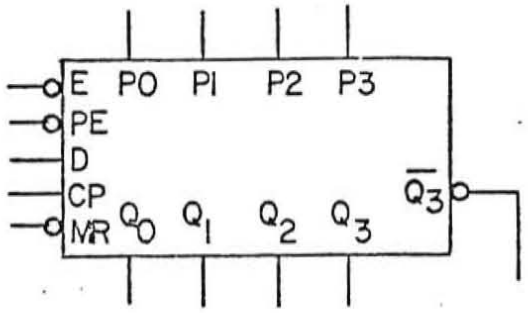
H = high level (off); L = low level (on).

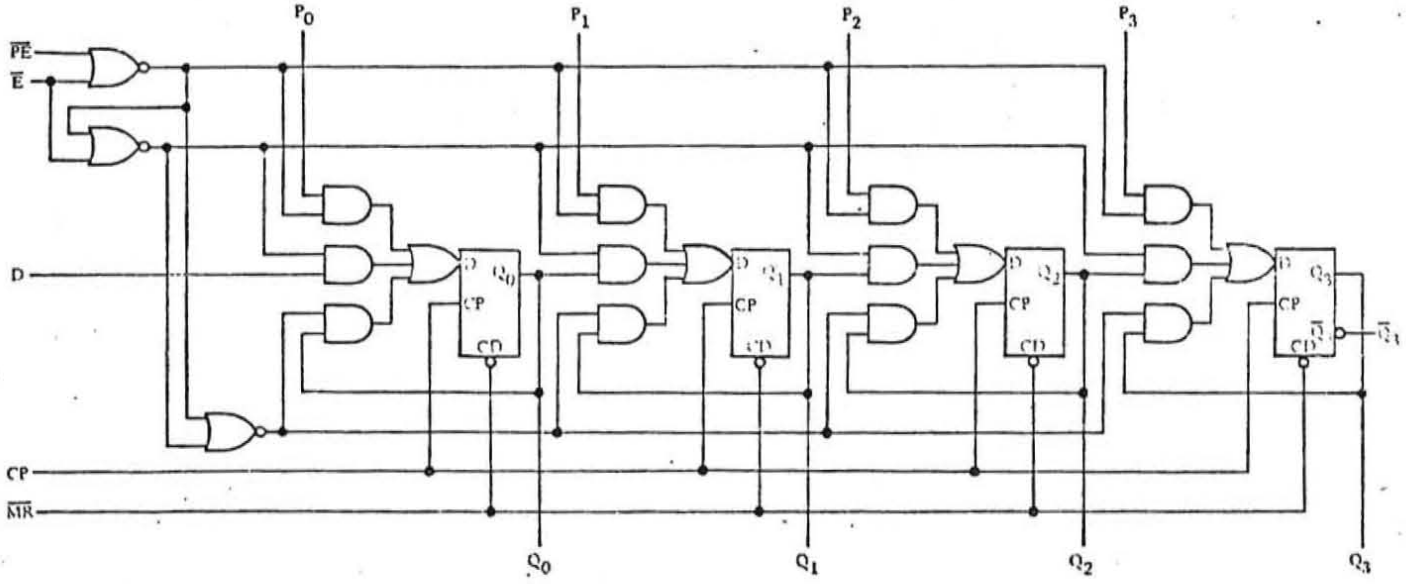
6.7 REGISTERS

A register is an electrical device which receives information upon command, and holds that information without modification until another command is received. Registers are nothing more than a number of flip-flops used on conjunction with each other.

6.7.1 Load/Shift Register (100-000180)

There is one 4 bit shift register IC which is used in a wide variety of register applications. This IC is illustrated below and its operation is covered in detail. The DGC part number is identificd by the 100-xxxxxx number shown above.

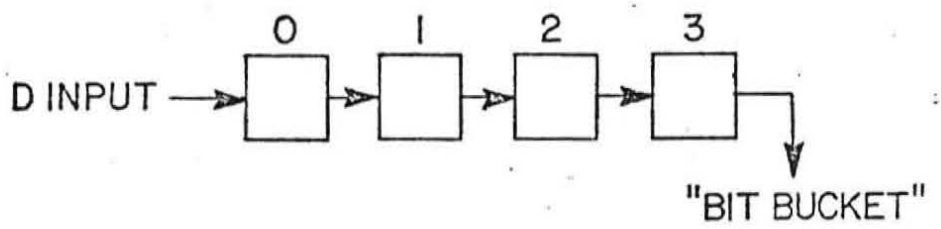




This IC has 3 synchronous modes of operation: parallel load, shift, and hold (do nothing). The hold capability permits information storage in the register which is independent of clock.

In order for any operation to be performed, the enable (E) input must be low. Once the IC is enabled with E low, the IC must be told whether to parallel load or shift. This is accomplished by controlling the PE input. The IC is programmed to parallel load the P0-P3 inputs when PE is low. The parallel load takes place on the positive going edge of the clocking signal (CP).

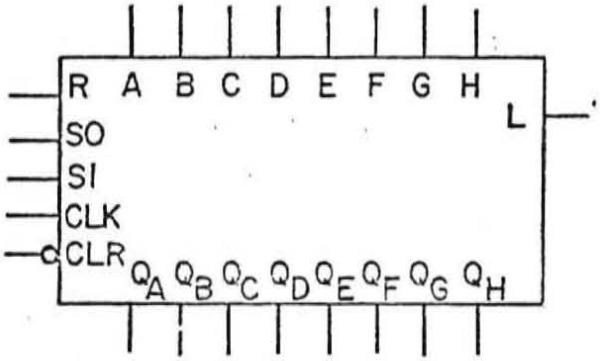
When PE is high the IC is programmed to shift one bit position on the positive going edge of clock. The shift operation shifts the bit contained in positions 0, 1 and 2 of the register into the next higher position. Bit position 3 is shifted out and bit position 0 receives the incoming bit which is controlled by the D input. When D is high a one is shifted into position 0 and a zero is shifted in when D is low.



The MR input clears the register when low. This master reset overrides all inputs, and does not require a clock signal.

6.7.2 8-bit Shift Register (100-000101)

Another shift register IC used in DGC logic provides parallel loading and shifts in either direction.



This register has four distinct modes of operation:

Parallel Load
Shift Right (QA towards QH)
Shift Left (QH towards QA)
Hold (do nothing)

All operations, except clear, are synchronous with the positive going edge of clock. The S0-S1 inputs are mode control inputs and control the register operation as follows:

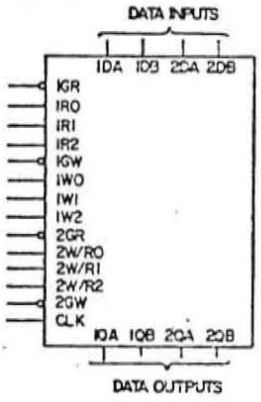
Operation of Mode Control		
Inputs		
S_1	S_0	Mode
L	L	Hold
H	L	Shift Left
L	H	Shift Right
H	H	Parallel Load

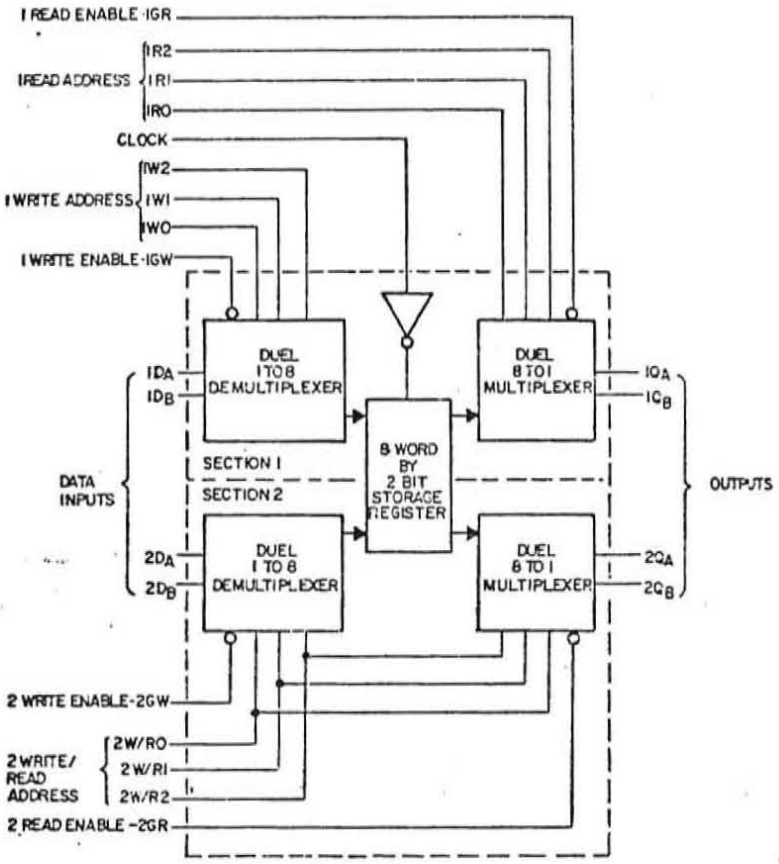
During a shift right operation, serial data (R input) is shifted into Q_A and Q_A through Q_G are shifted toward Q_H one bit position. A "one" bit is entered into Q_A when the R input is high. During a shift left operation serial data (L input) is shifted into Q_H and Q_H through Q_B are shifted toward Q_A one bit position.

6.7.3 16-bit Multiple-Port Register File (100-000171)

The 16-bit register file IC is organized as eight words of two bits each. The register file is similar in operation to a random access memory (RAM) of eight locations containing two bits each. That is, any one of the eight locations can be addressed and read from, without destroying the contents of the location, and the contents can be altered by generating a write signal. Multiple-port inputs and outputs allow two different locations to be read from or altered at the same time.

The register file logic symbol and simplified block diagram are shown below.





Referring to the simplified block diagram you will notice the register file is divided into two (2) sections. All inputs and outputs except CLOCK are prefixed with a 1 or 2 indicating which section they are to be associated.

Any one of the 8 word by 2 bit storage registers, shown in the center, can be read from either section one outputs ($1Q_A$, $1Q_B$) or section two outputs ($2Q_A$, $2Q_B$). Likewise any one of the 8 words can be altered by either section one inputs ($1D_A$, $1D_B$) or section two inputs ($2D_A$, $2D_B$).

In order to read from any word via section one output, the register file must have its section one 8 to 1 output multiplexor, selected to the word desired, and enabled. This is accomplished by applying the correct control levels to the section one read address lines (1R0-2), and applying a low control signal to the section one read enable line (1GR).

To read any word via section two output, the register file must have its section two, 8 to 1 output multiplexor, selected to the word desired and enabled. This is accomplished by applying the correct control levels to the section two address lines (2W/R0-2) and applying a low control signal to the section two read enable line (2GR).

No clock signal is required to read from the register file.

The register file is a tri-level output device (high, low or high-impedance). When the read enable line is applied high to the output multiplexor the associated outputs remain in the high-impedance state and neither significantly load or drive the line connected.

In order to write into any word the register file must have the input demultiplexor, associated with the desired section, addressed to the word to be altered (1W0-2 or 2W/R0-2) and the demultiplexor must be enabled (1GW or 2GW). The input data, associated with the desired section (1DA-B or 2DA-B) is then applied to the storage register and will be written into the storage register on the positive going edge of clock (CLK).

In register file applications, the same word could be selected to be read to both section outputs or two different words could be selected to appear on the section outputs. Two different words can be selected for alteration at the same time. Using section one of the register file, it is possible to read from one word while altering a different word. Section two has common write-read address lines, therefore, in applications it is possible to read from and alter the same word.

NOTE: Since the two sections are independent, it is possible for both write functions to be activated with both write addresses selecting the same word location. If this occurs and the information at the data inputs is not the same for both sections, the low-level data will predominate and be stored.

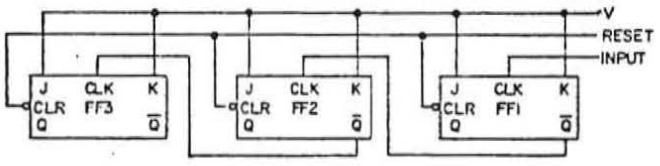
The register file IC is used by DGC computers to store the four accumulators, the program counter, and three other 16-bit registers. Since each register file contains two bits it is necessary to use 8 register file ICs to provide the 16-bit storage required. The 8 register file ICs will have their addresses, enables, and clock signal inputs connected in parallel.

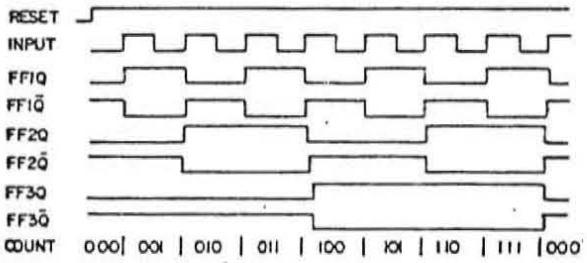
6.8 COUNTERS

A counter is an electrical device which records the number of events which have occurred. Counters are flip-flops connected together in cascade and depending upon the manner of connections, can be either up-counters or down-counters, as illustrated and described in the following paragraphs.

6.8.1 Up-Counters

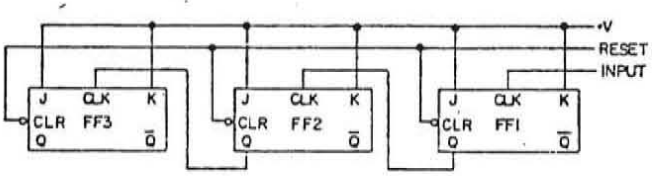
The binary up-counter illustrated below is incremented by one each time the INPUT signal has a positive going transition (⌐¬). In paragraph 6.4.2, you found when both the J and K inputs are high as in this illustration (+V), a JK flip-flop toggles to the opposite state when the clock signal is asserted. Therefore, FF1 will toggle on every positive going transition of INPUT, FF2 will toggle every other INPUT transition (FF1$\overline{Q}$ going high) and FF3 toggles every fourth INPUT transition (FF2$\overline{Q}$ going high).

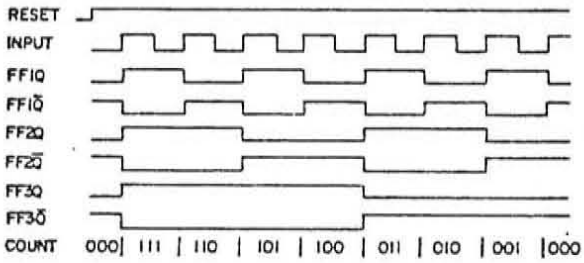




6.8.2 Down-Counters

The binary down-counter illustrated below is decrmented by one each time the INPUT signal goes high. Notice the only difference between the down-counter and the up-counter is one uses the Q output (down) of the JK flip-flops while the other used the $\overline{Q}$ output (up).





6.8.3 Presettable Counters

Counters can also be set to a specified value and the count continues from that value. This is accomplished by using the preset input to the flip-flops.

6.8.4 Counter ICs

There are many various types of counters contained within IC packages. Some of these are listed below:

1. 4-bit Binary Counter - Counts 0 to 15
2. BCD Decade Counter - Counts 0 to 9
3. Up/Down Binary Counter
4. Up/Down Decade Counter

Counter ICs are available in either asynchronous or synchronous operation. The up/down counters described in paragraphs 6.8.1 and 6.8.2 are examples of asynchronous operation, where the count is rippled through all the stages. Synchronous counters have additional gating which provides having all flip-flops clocked simultaneously so that the outputs change coincident with each other when instructed by the count input. A synchronous counter is described in paragraph 6.8.5.

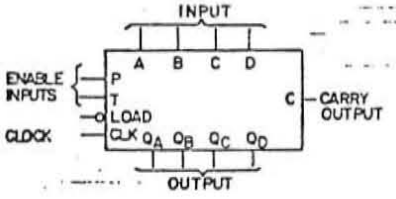
6.8.5 - Synchronous 4-Bit Counter

The synchronous 4-bit counter illustrated below is fully programmable; that is, the outputs may be preset to either level. As presetting is synchronous, setting up a low level at the load input disables the counter and causes the outputs to agree with the setup data after the next clock pulse regardless of the levels of the enable inputs.

The clear function is synchronous and a low level at the clear input sets all four of the flip-flop outputs low after the next clock pulse, regardless of the levels of the enable inputs.

This counter also provides an internal carry look-ahead (Ripple Output) which can be generated at the 15 (1111) count.

Both count-enable inputs (P and T) must be a high level to count and the T enable input also enables the carry output.



BLOCK DIAGRAM