

Introduction to Minicomputers

Logic and Hardware Basics



digital

1st Printing, June 1976
2nd Printing (Rev), October 1977
3rd Printing, August 1979

Copyright © 1976, 1977, 1979 by Digital Equipment Corporation

The reproduction of this workbook, in part or whole, is strictly prohibited. For copy information contact the Educational Services Department, Digital Equipment Corporation, Bedford, Massachusetts 01730.

Printed in U.S.A.

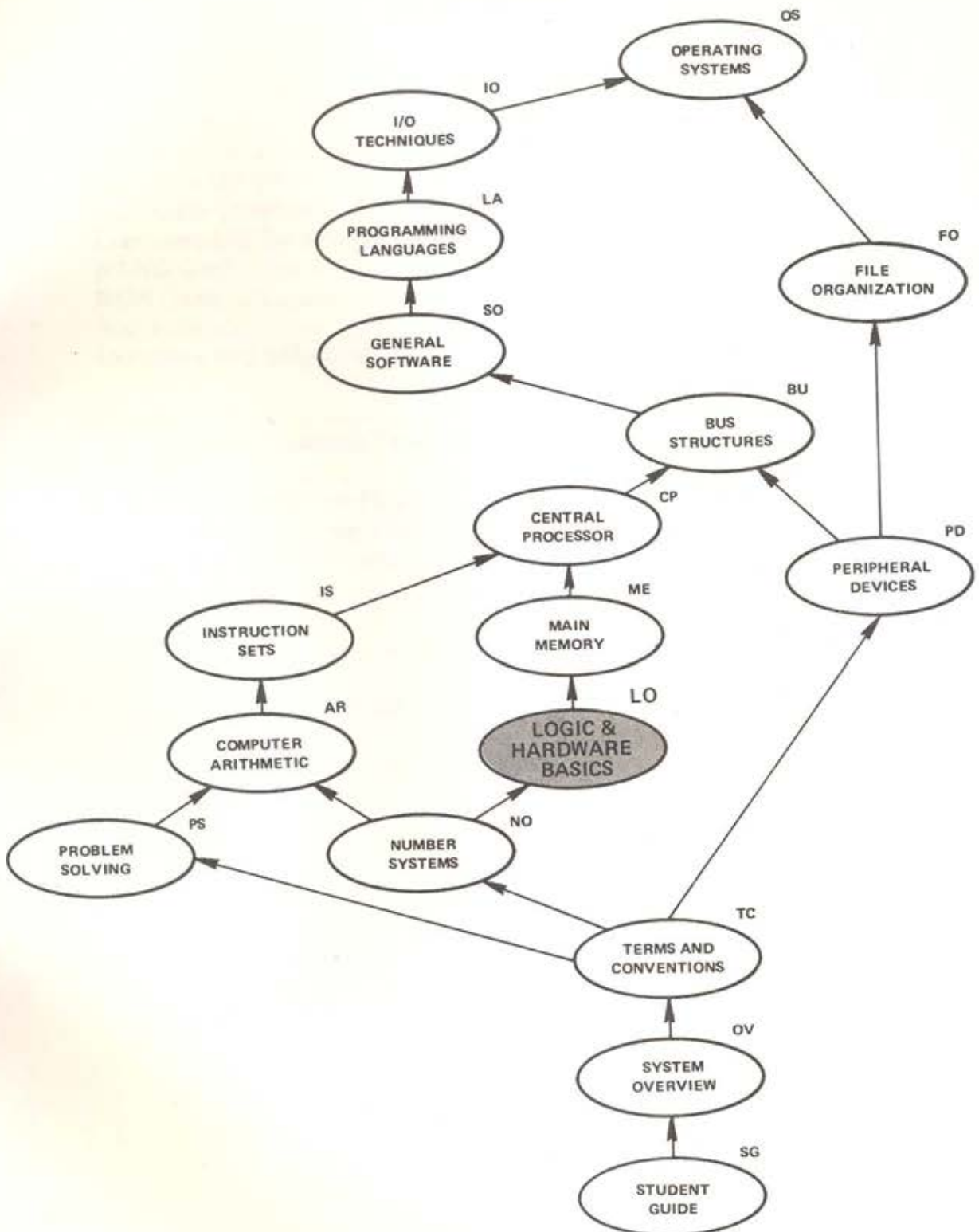
INTRODUCTION TO MINICOMPUTERS

Logic and Hardware Basics

Student Workbook

Audio-Visual Course by Digital Equipment Corporation

COURSE MAP



CONTENTS

Introduction.....	1
Logic Gates.....	3
Objectives and Sample Test Items.....	3
AND Gate.....	7
Inclusive OR Gate.....	10
Exclusive OR Gate.....	12
NOT Gate (Inverter).....	14
Exercises and Solutions.....	17
NAND Gate.....	21
NOR Gate.....	24
Summary.....	26
Exercises and Solutions.....	29
Basic Circuits.....	37
Objectives and Sample Test Items	37
Flip-Flops	41
Set-Reset (RS) Flip-Flops.....	42
Clock-Data (D-Type) Flip-Flops	43
Exercises and Solutions	45
Registers	49
Buffer Registers.....	49
Shift Registers.....	49
Counters	51
Exercises and Solutions	53
Appendix A More Complex Circuits.....	59

Logic and Hardware Basics

Introduction

Many people are awed by the apparent complexities of digital computer electronic circuits or "hardware." However, no matter how complex computer hardware may seem to be, it can be reduced to a small number of elemental components called "logic gates." These logic gates are the basic building blocks of any digital computer system – from the simplest to the most sophisticated computer in existence.

These basic logic gates, in themselves, perform extremely simple functions. However, by wiring them together in various combinations, they can be used to form more complex components such as adders and counters. These latter circuits can then be further combined to form extremely sophisticated logic elements.

It must be stressed that no matter how complex computer logic may appear at first glance, it can always be reduced to a combination of the basic logic gates that will be covered in this lesson. Therefore, it is quite important to obtain a thorough understanding of these basic gates.

This module contains two lessons. The first lesson describes the six basic *logic gates* used by computers. The second lesson describes how certain basic gates are used together to form such components as *flip-flops, registers, and counters*.

It should be noted that there are two basic approaches to describing logic gates. One is the conventional, theoretical approach – using 1 and 0 (true and false) to represent the two digital voltage levels in the hardware. The other approach, called Functional Boolean, does not uniquely assign true and false to digital voltage levels. Consequently, the Functional Boolean approach has many variations.

This module covers the conventional, theoretical approach. This approach provides the foundation for more advanced study.


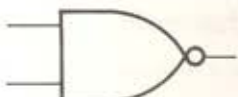

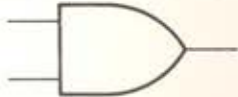
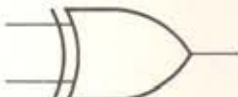

Logic Gates

OBJECTIVES

1. Given six logic symbols and six functions, be able to match each symbol with its function.
2. Given six truth tables and six logic functions, be able to match each truth table with its corresponding logic function.
3. Given a table of six logic gates and their inputs and outputs and a list of six mathematical expressions, be able to match each logic function with its mathematical expression.

SAMPLE TEST ITEMS

1. Match each of these logic symbols with the function it represents by writing the correct letter in the space provided.

Logic Symbol	Function
	
	
	
	
	
	

Functions

- | | |
|--------|---------|
| a. XOR | d. OR |
| b. NOR | e. NAND |
| c. AND | f. NOT |

SAMPLE TEST ITEMS

2. Six truth tables and six logic functions are given below. Match each truth table with the logic function it represents.

a.

Inputs		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

b.

Inputs		Output
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

c.

Inputs		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

d.

Inputs		Output
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

e.

Inputs		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

f.

Input	Output
0	1
1	0

Logic Function

AND

OR

NOR

NOT

NAND

XOR

Truth Table

SAMPLE TEST ITEMS

3. Determine the mathematical expression that illustrates the logical relationships between the inputs and output of the six logic gates listed below. Write the number of the correct mathematical expression in the space provided.

Gate	Inputs	Output	Mathematical Expression
AND	A,B	C	_____
OR	A,B	C	_____
NOR	A,B	C	_____
NOT	A	C	_____
NAND	A,B	C	_____
XOR	A,B	C	_____

Mathematical Expressions

- | | |
|--|-----------------------|
| 1. $C = \overline{A} \cdot \overline{B}$ | 4. $C = A \oplus B$ |
| 2. $C = A \cdot B$ | 5. $C = \overline{C}$ |
| 3. $C = \overline{A} + \overline{B}$ | 6. $C = A + B$ |

Mark your place in this workbook and view Lesson 1
in the A/V program, "Logic Gates."

Logic gates are simple electronic circuits that implement elementary logical functions. Logic gates serve as the basic building blocks for all digital computers. Computer operations, no matter how complex, are performed by using various combinations of these basic logic gates. The six most common types of logic gates are covered in this lesson. Each gate is named after the logical function that it performs. These gates are:

1. AND
2. OR
3. XOR (exclusive OR)
4. NOT (inverter)
5. NAND (not and)
6. NOR (not or)

Each of these gates, except the NOT gate (inverter), has *two or more* input lines but only *one* output line. The NOT gate has *one* input line and *one* output line.

When discussing logic gates, it is quite common to use the terms "true" and "false" when referring to the input and output states of the gate. In this workbook "true" is the equivalent of binary 1 and "false" is the equivalent of binary 0.

AND Gate

The AND gate provides a true output only when *all* of the inputs are true. In other words, the gate generates a binary 1 output when certain logic conditions are satisfied as indicated by a binary 1 on *every* input line. Thus, if the AND has two input lines, it can be stated that: $1 \text{ AND } 1 = 1$.

As an example, the output of an AND gate might be used to trigger some other logic element. The element is to be triggered only if certain conditions exist. If these conditions are set up as inputs to the AND gate, the element will be triggered only when the pre-defined conditions exist.

The standard logic symbol for an AND gate is shown in Figure 1. Notice that only two inputs are shown; however, an AND gate can have more than two inputs but never more than one output. The two input lines are labeled "A" and "B" while the output line is labeled "C."

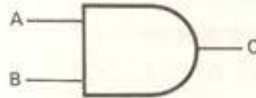


Figure 1 The AND Gate

A chart can be constructed to describe the operation of any logic circuit. This chart lists every possible combination of input values (binary 1s and 0s) and the output resulting from each combination of inputs. Some people substitute the words "true" and "false" for the values "1" and "0" when using a chart of this type. That's why this chart, or table, is commonly referred to as a "truth table" even though 1s and 0s are used.

Table 1 is a truth table for a 2-input AND gate. It lists *every* possible combination of input values and shows the appropriate output for each combination of inputs. Notice that there is an output only when *both* inputs are *true* (or *1s*). Thus, we can say that "C" (the output) is true when inputs "A" AND "B" are true.

Table 1 2-Input AND Gate – Truth Table

Inputs		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

When constructing truth tables for logic gates that have many inputs, it is useful to know how many input combinations are possible. This can be calculated by 2^n where "n" equals the number of input lines. For example, you have just seen a table for a 2-input AND gate. Using the formula 2^n , we substitute the number of inputs (2) for the "n" and come up with 2^2 . We know that $2^2 = 4$. Therefore, there are four possible input combinations in the truth table.

Table 2 below is a truth table for a 3-input AND gate. By using the formula 2^n , we know that the input combinations will be 2^3 (or 8). If you look at the table, you will see that there are 8 input combinations. Notice also that the output is true *only* when *all three inputs are true* . . . or 1s. We can say that "D" (the output) is true only when inputs "A" AND "B" AND "C" are true.

Table 2 3-Input AND Gate – Truth Table

Inputs			Output
A	B	C	D
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

Another method of describing the AND function is to use a mathematical formula. Using a formula provides a precise shorthand notation for easy representation of digital logic. Let's go back to the 2-input AND gate. We know that the output ("C") is true only when both inputs ("A" and "B") are true. Therefore, we can express this relationship by using the formula:

$$C = A \text{ AND } B$$

However, the relationship is normally expressed by using special logic notation. The AND function, for example, uses a large dot as the standard symbol. Thus, the AND function is normally expressed as:

$$C = A \cdot B$$

As another example, a 3-input AND gate is expressed by the formula:

$$D = A \cdot B \cdot C$$

where "D" represents the output of the gate, and "A," "B," and "C" represent the three inputs.

Inclusive OR Gate

The inclusive OR gate provides a true output if either one input *or* the other input *or both* inputs are true. This gate produces a true output (binary 1) when at least one input logic condition is satisfied as indicated by a binary 1 on one of the input lines. Thus, we can say that: $1 \text{ OR } 1 = 1$.

As an example, assume that a certain logic element is to be turned on when any one of a number of conditions exist. These conditions can be set up as OR gate inputs. The output of the gate is connected to the logic element. Therefore, whenever one or more of the conditions exist, the OR gate will provide an output that will turn on the desired logic element.

The standard logic symbol for an inclusive OR gate is shown in Figure 2. Notice that only two inputs are shown; however, an inclusive OR gate can have more than two inputs but never more than one output. The two input lines are labeled "A" and "B" while the output line is labeled "C."



Figure 2 The Inclusive OR Gate

This gate is called an *inclusive* OR because any *one or more* true inputs will provide a true output. In other words, if there were four input lines, a true input on lines 1 through 4, inclusive, would provide a true output. Whenever the term "OR" gate is used by itself, it refers to the "inclusive OR" gate.

Table 3 is a truth table for a 2-input inclusive OR gate. It lists *all* possible combinations of input values and shows the output for each combination of inputs.

Table 3 2-Input OR Gate – Truth Table

Inputs		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Notice in Table 3 that the *output* of an inclusive OR gate is zero only when *all* inputs are zero. On the other hand, the output is one if *either* or *both* inputs are one.

Table 4 is a truth table for a 3-input inclusive OR gate. Again, notice that the output is zero only when *all* inputs are zero.

Table 4 3-Input OR Gate – Truth Table

Inputs			Output
A	B	C	D
0	0	0	0
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	1

A mathematical formula can also be used to represent the inclusive OR function. For example, operation of a 2-input inclusive OR gate can be expressed by the formula:

$$C = A \text{ OR } B$$

However, the inclusive OR function is usually represented by a special symbol (+). The inclusive OR function is normally expressed as:

$$C = A + B$$

In the case of a 3-input OR gate, the mathematical formula would be:

$$D = A + B + C$$

The formula could be stated: "D" is true if "A" OR "B" OR "C" is true.

Exclusive OR Gate

The purpose of the *exclusive* OR gate is to provide a true output if either one input *or* the other input, *but not both*, is true. In other words, the gate generates a true output (binary 1) when any one, but not more than one, logic condition is satisfied.

As an example, assume that a certain logic element is to be turned on only if one of a number of conditions exist. The element is to remain off if none of the conditions exist, or if more than one condition exists. This problem could be satisfied by using an exclusive OR gate output as the input to the logic element.

The standard logic symbol for an exclusive OR gate is shown in Figure 3. Note that the exclusive OR gate is usually referred to as an XOR gate to differentiate it from an inclusive OR gate. Notice that only two inputs are shown in the figure; however, an exclusive OR gate can have more than two inputs but never more than one output. The input lines are labeled "A" and "B" while the output line is labeled "C."



Figure 3 The Exclusive OR (XOR) Gate

This gate is called an *exclusive* OR because any *one* true input, *exclusively*, will provide a true output. In other words, if there is *more than one* true input, the output will be *false*.

Truth tables for the exclusive OR (XOR) and inclusive OR functions are compared in Table 5.

Table 5 XOR Versus Inclusive OR – Truth Tables

XOR			Inclusive OR		
Inputs		Output	Inputs		Output
A	B	C	A	B	C
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	1	1	1

As you can see in Table 5, the XOR and inclusive OR gates are identical except for the *last* condition. With *inclusive* OR, the output is 1 when *both inputs are 1s*. However, with XOR, the output is 0 when *both inputs are 1s*.

In summary, the output of an XOR gate is 1 if *either* input, but not both, is 1. If both inputs are 0, or if both inputs are 1, the output is 0.

The XOR rules are true even when dealing with gates having more than two inputs as shown in Table 6. Notice that the output is 1 *only* when a single input is 1.

Table 6 3-Input XOR Gate

Inputs			Output
A	B	C	D
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

A mathematical formula can also be used to represent the XOR function. For example, operation of a 2-input exclusive OR (XOR) gate can be expressed by the formula:

$$C = A \text{ XOR } B$$

However, the XOR function is usually represented by a special symbol which is a plus sign with a circle (\oplus). Thus, the XOR function is normally expressed as:

$$C = A \oplus B$$

In the case of a 3-input XOR gate, the formula would be:

$$D = A \oplus B \oplus C$$

NOT Gate (Inverter)

The purpose of the NOT gate is to convert an input signal to its opposite (complementary) state. The NOT gate is often called an *inverter* because it inverts the input signal. For example, if the input signal is high, the NOT gate inverts it to a low signal.

The standard symbol for a NOT gate is shown in Figure 4. There are three important items to notice in the figure:

1. The NOT gate is the only gate with a *single input* and a *single output* line.
2. A bar drawn over the input or the output indicates the false state. If the input is "A," the output is "NOT A."
3. A small circle drawn on either the input or output line indicates the false or "not" state.

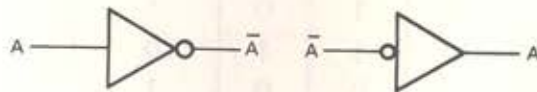


Figure 4 The NOT Gate

Table 7 is a truth table for the NOT gate. Notice that there are only two possible inputs to the gate.

Table 7 NOT Gate – Truth Table

Input	Output
0	1
1	0

We can also use a mathematical formula to represent the NOT function. We express the NOT function as:

$$A = \text{NOT } A$$

or

$$\text{NOT } A = A$$

The NOT function is usually represented by a standard symbol – a bar drawn over the false state. Thus, the NOT function is normally expressed as:

$$A = \overline{A}$$

or

$$\overline{\overline{A}} = A$$

EXERCISES

1. The following formula is used to indicate how many input combinations are possible for a given logic gate:

$$\text{Possible Inputs} = 2^n$$

If the logic gate has five input lines, how many input combinations are possible? (Circle the correct answer.)

- a. 16
 - b. 8
 - c. 32
 - d. 12
2. Using binary 1s and 0s, complete the truth table for each of the following logic gates:

AND

Inputs		Output
A	B	C

OR

Inputs		Output
A	B	C

XOR

Inputs		Output
A	B	C

NOT

Input	Output

SOLUTIONS

1. The following formula is used to indicate how many input combinations are possible for a given logic gate:

$$\text{Possible Inputs} = 2^n$$

If the logic gate has five input lines, how many input combinations are possible? (Circle the correct answer.)

- a. 16
 - b. 8
 - ☒ c. 32
 - d. 12
2. Using binary 1s and 0s, complete the truth table for each of the following logic gates:

AND

Inputs		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

OR

Inputs		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

XOR

Inputs		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

NOT

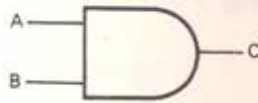
Input	Output
0	1
1	0

EXERCISES

3. Draw the gate that outputs a 1 only when both inputs are a 1.
4. Draw the gate that outputs the complement of the input.
5. Draw the gate that outputs a 1 when only one of its inputs is a 1.
6. Draw the gate that outputs a 1 when at least one of its inputs is a 1.

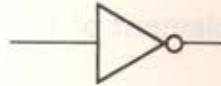
SOLUTIONS

3. Draw the gate that outputs a 1 only when both inputs are a 1.



AND gate

4. Draw the gate that outputs the complement of the input.



NOT gate

5. Draw the gate that outputs a one when any one of the inputs is a one.



XOR gate

6. Draw the gate that outputs a one when at least one of its inputs is a one.



OR gate

There are two additional gates to be covered in this module: the NAND gate and the NOR gate.

NAND Gate

The word "NAND" is a contraction for the words "not and." In order to understand the function of a NAND gate, it is necessary to review the AND and the NOT gates.

Figure 5 illustrates a standard AND gate. When both inputs are true (binary 1s), the output is also true (binary 1).

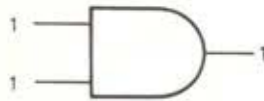


Figure 5 A Standard AND Gate

Figure 6 illustrates a standard NOT gate. Notice here that a true input (binary 1) results in a *false* output (binary 0).

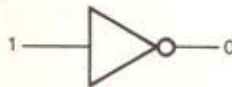


Figure 6 A Standard NOT Gate

Figure 7 illustrates what happens if the output of the AND gate is connected to the input of the NOT gate. Notice that the input to this *circuit* consists of true inputs (binary 1s) but the output is *false* (binary 0). This is the basic function that is performed by a NAND gate.

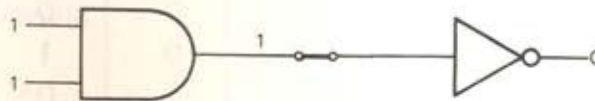


Figure 7 Connecting an AND and NOT Gate

The function of a NAND gate, therefore, is to provide a false output when *all* of the inputs are *true*. In other words, the gate generates a binary 0 output when all inputs are binary 1s. The NAND gate will produce a binary 1 output if *any* input line contains a binary 0.

As an example of NAND gate use, assume that some logic circuit is to be turned off whenever certain conditions exist simultaneously. Each of these conditions can be represented by one input line to the NAND gate. The output of the gate is then connected to the logic circuit. Thus, when all conditions are true (a 1 on each NAND gate input line), the NAND gate will produce a false (or binary 0) output which will turn off the logic circuit.

The standard logic symbol for a NAND gate is shown in Figure 8. Notice that only two inputs are shown. However, a NAND gate can have more than two inputs but *never more than one output*. The two input lines are labeled "A" and "B" while the output line is labeled "C."

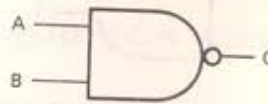


Figure 8 The NAND Gate

Truth tables for the NAND and AND functions are compared in Table 8. As shown in the table, the outputs of the NAND gate and AND gate are exactly *opposite*. With a NAND gate, the output is a 1 whenever there is a 0 on either or both input lines. The output is 0 only when there are 1s present on all input lines.

Table 8 NAND Versus AND – Truth Tables

NAND			AND		
Inputs		Output	Inputs		Output
A	B	C	A	B	C
0	0	1	0	0	0
0	1	1	0	1	0
1	0	1	1	0	0
1	1	0	1	1	1

Opposite Outputs

NOTE

When logic circuits are designed, AND and NAND gates are used to identify a specific set of input conditions. These gates output a unique value only when the specific inputs occur. In this respect, the AND gate and the NAND gate are alike – they both respond to the input condition of all 1s with a unique output. The difference is that the AND gate outputs the unique output value of one, and the NAND outputs the unique value of zero. The selection of a NAND gate or an AND gate depends on the circuit or device that is to receive its output.

A mathematical formula can be used to represent the NAND function. Although NAND is a contraction for NOT AND, it is interesting to note that the formula can be expressed in terms of inclusive OR. A 2-input NAND gate, for instance, is expressed by the formula:

$$C = \bar{A} + \bar{B} \text{ (C equals Not A or Not B)}$$

Thus, when both inputs are binary 0 (\bar{A} and \bar{B}), the output is binary 1. If either input is a binary 0 (\bar{A} or \bar{B}), the output is binary 1. However, when both inputs are binary 1 (A and B), the output (C) is binary 0. Therefore, if *either one or both inputs are false*, the output is true. Notice the similarities between the two formulas:

$$\text{OR} \quad C = A + B \text{ (C equals A or B)}$$

$$\text{NAND} \quad C = \bar{A} + \bar{B} \text{ (C equals Not A or Not B)}$$

The NAND gate can also be expressed in terms of the AND function by using the formula:

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

This formula states that C is false when A and B are true.

NOR gate

The word "NOR" is a contraction for the words "not or." In order to understand the function of a NOR gate, it is necessary to review the inclusive OR gate and the NOT gate.

Figure 9 illustrates the possible conditions of an inclusive OR gate. Notice that when either or both inputs are true (binary 1), the output is also true (binary 1).

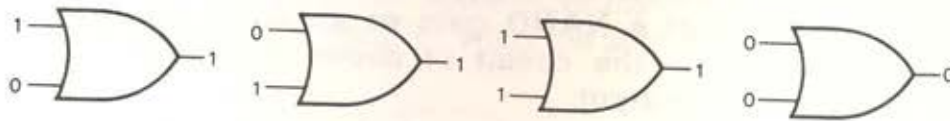


Figure 9 Inclusive OR gate – Possible Conditions

Figure 10 illustrates a standard NOT gate. Notice here that a true input (binary 1) results in a *false* output (binary 0).



Figure 10 A Standard NOT Gate

Figure 11 shows what happens if the output of the above OR gate is connected to the input of the above NOT gate. Notice that whenever either one or both inputs to the *circuit* are true (binary 1), the output of the *circuit* will be *false* (binary 0). This is the same basic function that is performed by the NOR gate.

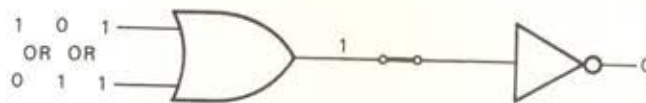


Figure 11 Connecting an OR and NOT Gate

The function of a NOR gate is to provide a *false* output when *either one or both* inputs are *true*. The opposite is also true. If no input line is true (both inputs *false*), then the NOR gate will generate a *true* output.

A NOR gate might be used to turn on some logic circuit only if certain conditions do not exist. If any one of these conditions do exist, the NOR gate produces a false output, preventing the logic circuit from being turned on.

The standard logic symbol for a NOR gate is shown in Figure 12. Although only two inputs are shown, a NOR gate can have more than two inputs but *never more than one output*. Notice also that the NOR gate is an inclusive OR symbol with a small circle on the output line to indicate that the output is false when the inputs are true (i.e., the circle represents the NOT function).



Figure 12 The NOR Gate

NOTE

As with AND and NAND, OR and NOR are also functionally the same – OR and NOR both have their desired output when at least one input is a one. The difference is that the desired output of an OR gate is one, while the desired output for a NOR gate is zero. The selection of an OR gate or a NOR gate depends on the circuit or device that is to receive the output.

Truth tables for the NOR and OR functions are compared in Table 9. As shown in the table, the NOR gate and OR gate outputs are exactly the *opposite*. With a NOR gate, the output is 0 whenever there is a 1 on either or both input lines. The output is 1 only when there are 0s present on all input lines.

Table 9 NOR vs OR – Truth Tables

NOR			OR		
Inputs		Output	Inputs		Output
A	B	C	A	B	C
0	0	1	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	0	1	1	1

Opposite Outputs

A mathematical formula can be used to represent the NOR function. Although NOR is a contraction for NOT OR, it is interesting to note that the formula is expressed in terms of AND. A 2-input NOR gate, for instance, can be expressed by the formula:

$$C = \overline{A} \cdot \overline{B} \text{ (C equals Not A and Not B)}$$

Thus, the only time the NOR gate produces a true (binary 1) output is when both inputs are binary 0 (\overline{A} AND \overline{B}).

The NOR gate can also be expressed in terms of the OR function by using this formula:






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

This formula states that C is true whenever the expression A or B is false. In other words, when both A and B are false, then C is true. However, if either A or B is true, then C will be false.

Summary

Table 10 is a composite truth table for five of the logic gates that have been covered in this lesson. Only the inverter (NOT gate) has been left out because it is a single input device. Notice that the table also includes the standard logic symbol for each gate as well as the appropriate mathematical formula.

Table 10 Basic Logic Gates – Truth Table

Inputs		Output (C)				
A	B	AND	OR	XOR	NAND	NOR
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	1	0
1	1	1	1	0	0	0
Logic Symbol						
Formula		$C = A \cdot B$	$C = A + B$	$C = A \oplus B$	$C = \overline{A + B}$ $C = \overline{A} \cdot \overline{B}$	$C = \overline{A \cdot B}$ $C = \overline{A} + \overline{B}$

EXERCISES

1. List the names of the six basic logic gates.

a.

b.

c.

d.

e.

f.

2. In the following formula, C is true only when A is true and B is _____.

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

3. In the following formula, C is a binary 0 and A is a binary 1. Therefore, B must be a binary _____.

$$C = A \oplus B$$

4. Using binary 1s and 0s, complete the truth table for each of the following logic gates:

NAND

Inputs		Output
A	B	C

NOR

Inputs		Output
A	B	C

SOLUTIONS

1. List the names of the six basic logic gates.
 - a. AND
 - b. OR
 - c. XOR (exclusive OR)
 - d. NOT (inverter)
 - e. NAND
 - f. NOR
2. In the following formula, C is true only when A is true and B is not true (false).

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

3. In the following formula, C is a binary 0 and A is a binary 1. Therefore, B must be a binary 1.

$$C = A \oplus B$$

4. Using binary 1s and 0s, complete the truth table for each of the following logic gates:

NAND

Inputs		Output
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

NOR

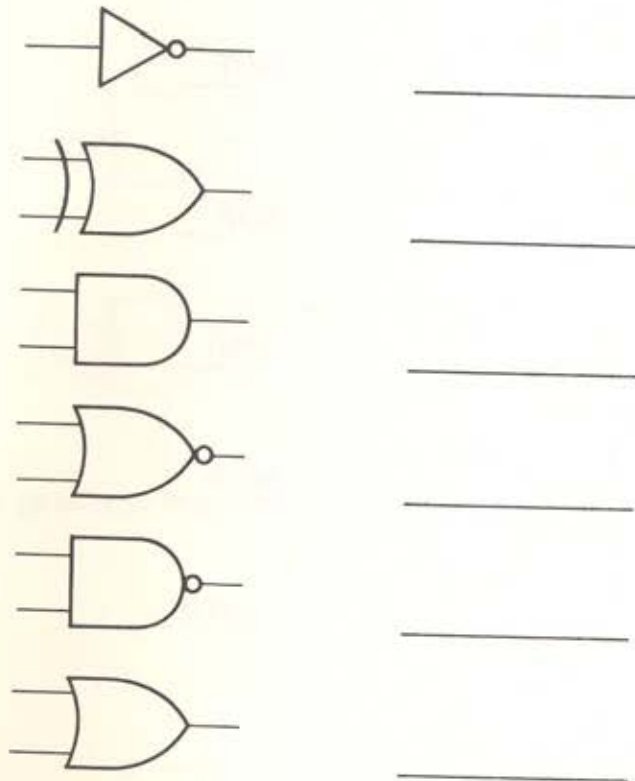
Inputs		Output
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

EXERCISES

5. Match each of the following logic functions with the correct mathematical formula:

- | | |
|---------|---|
| a. AND | () $C = \overline{A} \cdot \overline{B}$ |
| b. OR | () $C = A \oplus B$ |
| c. XOR | () $C = \overline{A} + \overline{B}$ |
| d. NOT | () $C = A \cdot B$ |
| e. NAND | () $C = A + B$ |
| f. NOR | () $C = \overline{C}$ |

6. Write the name of each logic gate represented by the symbols shown below:

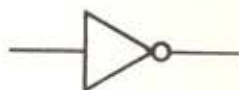


SOLUTIONS

5. Match each of the following logic functions with the correct mathematical formula:

- | | |
|---------|---|
| a. AND | (f) $C = \overline{A} \cdot \overline{B}$ |
| b. OR | (c) $C = A \oplus B$ |
| c. XOR | (e) $C = \overline{A} + \overline{B}$ |
| d. NOT | (a) $C = A \cdot B$ |
| e. NAND | (b) $C = A + B$ |
| f. NOR | (d) $C = \overline{C}$ |

6. Write the name of each logic gate represented by the symbols shown below:



NOT



XOR



AND



NOR



NAND



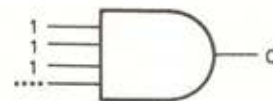
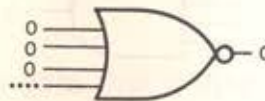
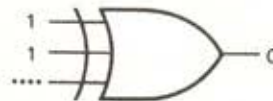
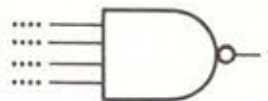
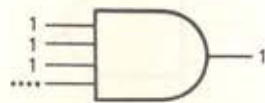
OR

EXERCISES

7. Write the mathematical formula for each of the following logic functions. Use "C" for the output and "A" and "B" for the inputs.

- a. XOR _____
- b. NAND _____
- c. OR _____
- d. AND _____
- e. NOR _____

8. Six logic gates are shown below. In each case write in a binary 1 or 0 for the missing input condition(s) that will produce the output condition indicated.



9. Draw the gate that outputs a 0 when at least one input is a 1.

SOLUTIONS

7. Write the mathematical formula for each of the following logic functions. Use "C" for the output and "A" and "B" for the inputs.

a. XOR $C = A \oplus B$

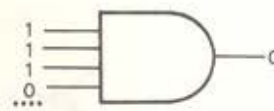
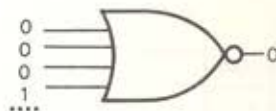
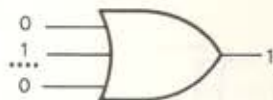
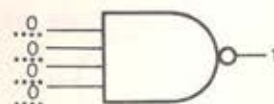
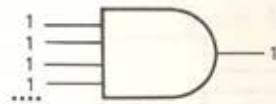
b. NAND $C = \overline{A + B}$ or $C = \overline{A \cdot B}$

c. OR $C = A + B$

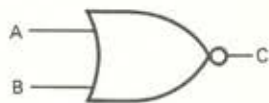
d. AND $C = A \cdot B$

e. NOR $C = \overline{A \cdot B}$ or $C = \overline{A + B}$

8. Six logic gates are shown below. In each case, write in a binary 1 or 0 for the missing input condition(s) that will produce the output condition indicated.



9. Draw the gate that outputs a 0 when at least one input is a 1.



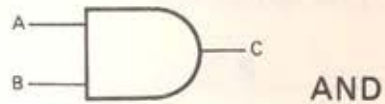
NOR

EXERCISES

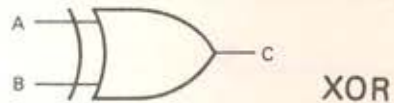
10. Draw the gate that outputs a 1 only when both inputs are a 1.
11. Draw the gate that outputs a 1 when only one of its inputs is a 1.
12. Draw the gate that outputs a 0 only when both inputs are a 1.
13. Draw the gate that outputs a 1 when at least one of its inputs is a 1.
14. Draw the gate that outputs the complement of the input.

SOLUTIONS

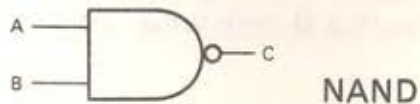
10. Draw the gate that outputs a 1 only when both inputs are a 1.



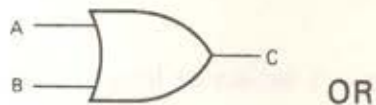
11. Draw the gate that outputs a 1 when only one of its inputs is a 1.



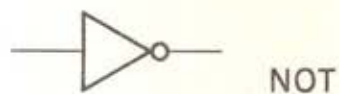
12. Draw the gate that outputs a 0 only when both inputs are a 1.



13. Draw the gate that outputs a 1 when at least one of its inputs is a 1.



14. Draw the gate that outputs the complement of the input.



Basic Circuits

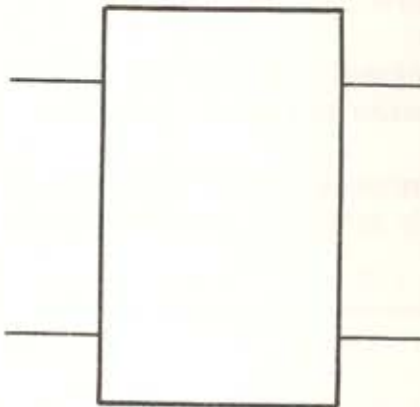
OBJECTIVES

1. Given blank diagrams and truth tables for both set-reset and D-type flip-flops, be able to: (1) label the input and output lines on the diagram, and (2) indicate in writing the elements of the truth table.
2. Given the terms "buffer register," "shift register," and "counter," and several functions for each, be able to select the one proper function of each term.
3. Given a logic circuit with four gates, one flip-flop, and a specific set of inputs, be able to write the circuit outputs.
4. Given six terms related to logic, logic circuits, and hardware and a list of six definitions, be able to match each term with its definition.

SAMPLE TEST ITEMS

1. Blank diagrams and truth tables for both set-reset and D-type flip-flops are given below. Label the input and output lines on the diagrams, and then complete the truth tables. (Note: Be sure to write in *headings* for the truth tables.)

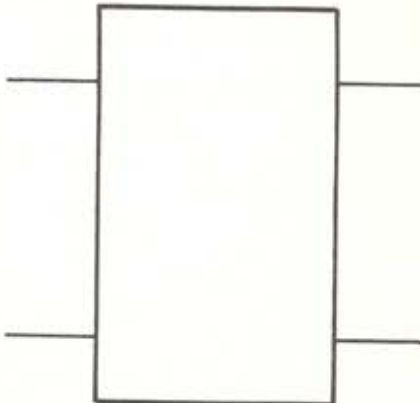
**SET-RESET
FLIP-FLOP
DIAGRAM**



TRUTH TABLE

INPUT		OUTPUT	

**D-TYPE FLIP-FLOP
DIAGRAM**



TRUTH TABLE

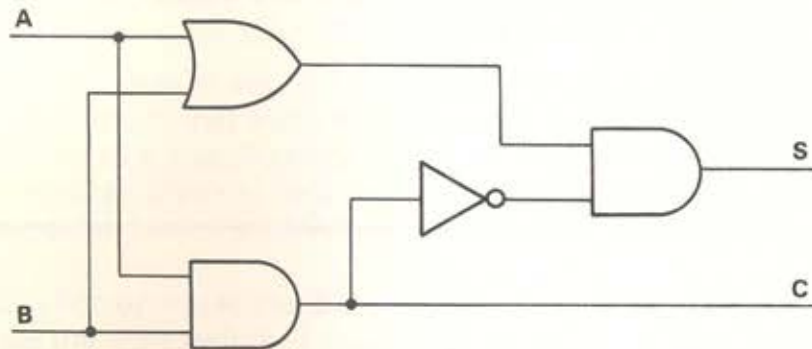
INPUT		OUTPUT	

SAMPLE TEST ITEMS

2. The buffer register:

- a. Permanently retains information for use in later programming.
- b. Momentarily slows down high-speed devices to allow lower-speed devices to handle output.
- c. Temporarily retains information until the selected unit is ready for it.
- d. Temporarily increases the speed of lower-speed devices to handle the output of faster devices.

3. For the logic circuits below, indicate the outputs for both Input A and Input B.



Input A

A = 1
B = 0
C = ____
S = ____

Input B

A = 1
B = 1
C = ____
S = ____

SAMPLE TEST ITEMS

4. Match each of the terms below with its definition.

Term	Definition
Truth Table	_____
Don't Care	_____
.	.
.	.
.	.

Definitions

- A particular output for a given set of inputs that is irrelevant.
- A means of expressing the input and output relationships of a logic circuit in tabular form.

.

.

.

Mark your place in this workbook and view Lesson 2 in the A/V program.

Flip-Flops

A *flip-flop* is the basic electronic circuit used to store information in digital computers. This component is found in every type of digital computer. The flip-flop has two independent *stable* states and is capable of storing one bit of information, such as a bit in a word.

The flip-flop is capable of *storing* and *remembering* information. Its outputs are based on *past* input data as well as *current* input data. The flip-flop differs from the other logic gates we have examined; it can *store* data and *utilize* the stored data as well as current input data to specify the output generated. For this reason, the flip-flop is widely used in digital computers. The output of a logic gate depends *only* on present input; the output of a flip-flop depends on past, as well as present inputs.

When a flip-flop is set to one of its two stable states by an input pulse, it remains in that state when the input pulse is removed. Once a flip-flop is set to a specific state, it can only be changed to the opposite state by *another* different input pulse. This principle allows a flip-flop to store a bit value until the bit value is modified by a different input pulse.

The flip-flop operates like a toggle light switch that has an *on* and *off* state. Like the light switch, it must settle into *one* of these two states. It can *never* assume both states simultaneously. After you set the light switch with your finger, it remains in that state until you change it again with your finger.

There are a number of different types of flip-flops. The set-reset (RS) flip-flop and the clock-data or D-type flip-flop are two commonly used types. Both of these will be discussed in this lesson. Other types of flip-flop devices include the JK and the RST. All of these flip-flop devices operate in much the same manner, differing basically in the number and type of input lines.

Set-Reset (RS) Flip-Flops

As illustrated in Figure 13, an RS flip-flop has two input lines, the *set* line and the *reset* line, and two output lines, a *ONE* and a *ZERO* line. The labels "one" and "zero" for the output lines of a flip-flop tend to be confusing because these lines can carry *either* a binary 1 or 0 value. However, this line labeling convention is commonly used. To minimize confusion, output values will always be written as 1 or 0, while the output lines themselves will be labeled with names (ONE, ZERO).

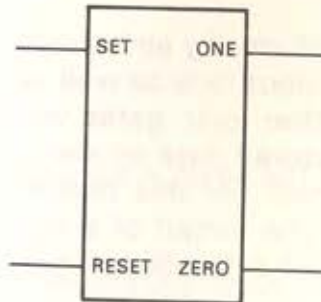


Figure 13 Set-Reset Flip-Flop

When a binary 1 is applied to the reset line, a binary 0 is produced on the ONE line. The ZERO line always outputs the *opposite* value of the ONE line. That is, if there is a binary 1 on the ONE line, a binary 0 will appear on the ZERO line and vice versa. Since the output of a flip-flop remains stable until a different input pulse is applied, repeating the same binary pulse on an input line will have no effect. The flip-flop will remain unchanged until a different value on the input line causes a change in state.

Table 11 gives the truth table for a set-reset flip-flop. As you can see, the output on the ONE line and the output on the ZERO line are always opposite. The RS flip-flop should *never* receive a 1 input on both the set and reset lines simultaneously as the resulting output would be uncertain.

Table 11 Truth Table for an RS Flip-Flop

INPUT		OUTPUT	
SET	RESET	ONE	ZERO
0	0	No change (Same as previous output)	
0	1	0	1
1	0	1	0
1	1	Uncertain	

NOTE

Unlike the truth tables for logic gates, the truth tables for flip-flops contain a new entry: "no change."

Clock-Data (D-type) Flip-Flops

D-type flip-flops save the binary value of a data line at some specific point in time. In order to accomplish this, the D-type flip-flop utilizes a control signal, called a *clock signal*, to specify when the binary data is to be stored. At that specified time, the data on the input line to the flip-flop is saved, regardless of the value already in the flip-flop.

The D-type flip-flop resembles the set-reset flip-flop, but as indicated in Figure 14, the two input lines to this type of flip-flop are the *DATA* line and the *CLOCK* line.

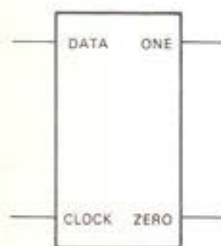


Figure 14 D-type Flip-Flop

When the *CLOCK* input goes from 0 to 1 (\uparrow), the value on the *DATA* input line becomes the output of the flip-flop on the *ONE* line. If there is a 1 on the *DATA* line at the time of the clock pulse, then the *ONE* line output of the flip-flop is a 1. If there is a 0 on the *DATA* line at the time of the clock pulse, the *ONE* line output is a 0. The flip-flop ignores any change in data between input pulses. Table 12 is the truth table for a D-type flip-flop.

Table 12 Truth Table for a D-type Flip-Flop

INPUT		OUTPUT	
CLOCK	DATA	ONE	ZERO
0	0	No change (Same as previous output)	
0	1	No change (Same as previous output)	
↑	0	0	1
↑	1	1	0

As with the RS flip-flop, the ZERO line output is always the opposite of the ONE line output.

EXERCISES

1. How does a flip-flop differ from a logic gate?
2. List at least two types of flip-flops.
3. In the space below, draw a simple diagram (logic symbol) of an RS flip-flop. Label all input and output lines.
4. For an RS flip-flop, given the input and output line values in the table below, fill in the output line values for the next output:

INPUT		OUTPUT	
SET	RESET	ONE	ZERO
0	0		
0	1		
1	0		
1	1		

SOLUTIONS

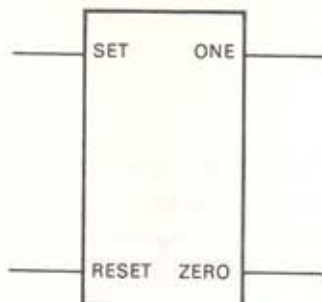
1. How does a flip-flop differ from a logic gate?

A flip-flop can store and remember information. A logic gate depends only on current input information.

2. List at least two types of flip-flops.

- a. Set-reset (RS)
- b. D-type, clock-data
- c. JK
- d. RST

3. In the space below, draw a simple diagram (logic symbol) of an RS flip-flop. Label all input and output lines.





4. For an RS flip-flop, given the input and output line values in the table below, fill in the output line values for the next output:

INPUT		OUTPUT	
SET	RESET	ONE	ZERO
0	0	No change	
0	1	0	1
1	0	1	0
1	1	Uncertain	

EXERCISES

5. What is the difference between an RS flip-flop and a D-type flip-flop?
6. Draw a simple diagram of a D-type flip-flop. Label all input and output lines.
7. For a D-type flip-flop, given the input and output line values in the table below, fill in the output line value for the next output.

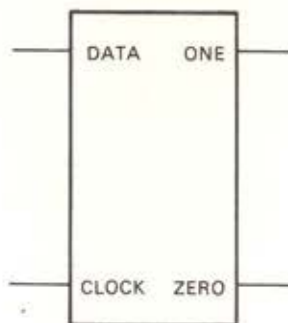
INPUT		OUTPUT	
CLOCK	DATA	ONE	ZERO
0	0	No change	
0	1		
	1		
	0		

SOLUTIONS

5. What is the difference between an RS flip-flop and a D-type flip-flop?

The D-type flip-flop assumes the state of the DATA input when a clock *pulse* appears at its CLOCK input. The RS flip-flop does *not* have a CLOCK input. Instead, the RS flip-flop is set or reset by a binary 1 *level* appearing at its SET input or RESET input. If the RS flip-flop receives a 1 at both its SET and RESET inputs, its state is uncertain.

6. Draw a simple diagram of a D-type flip-flop. Label all input and output lines.



7. For a D-type flip-flop, given the input and output line values in the table below, fill in the output line value for the next output.

INPUT		OUTPUT	
CLOCK	DATA	ONE	ZERO
0	0	No change	
0	1	No change	
↑	1	1	0
↑	0	0	1

Registers

Flip-flops can be interconnected to form more complex computer *circuits*. Some of these circuits will be discussed in Appendix A.

As we discussed, each flip-flop can store *only one bit of information*. However, more than one bit is required to make a computer word. Consequently, a group of flip-flops must be joined together to form a *register*.

A register is used to store the binary representation of a number, control information, or some other pattern of bits that the computer uses. Registers provide *temporary, fast-access storage* for data and addresses. Computers generally use a number of different registers. Each register performs a specific function in the data processing operation or in the control of the computer.

These registers can be made up of any type of flip-flop. Figure 15 illustrates a register using D-type flip-flops with an input and output line for each bit.

When the CONTROL (CLOCK) line changes from 0 to 1 (\uparrow), the value of each data output line is set to the value of the corresponding input line.

Buffer Registers

Some parts of the computer, such as the central processing unit, process information at extremely high speeds, while other units operate at lower speeds. In order to compensate for the lower speed and to avoid keeping a high-speed device waiting, special registers, called *buffer registers*, located in the individual units, *temporarily* retain information until the selected unit is ready to utilize it. Maximum utilization of the computer can be attained by using buffer registers.

Shift Registers

We have discussed registers that are used for storage and information transfer only. Another type of register is the shift register. Shift registers perform an important additional operation on stored information. A shift register can *shift* its entire contents one or more bit positions to the right or left. As a result of the shift operation, data can be fed into one end of the register, one bit at a time, and then shifted through the register as needed.

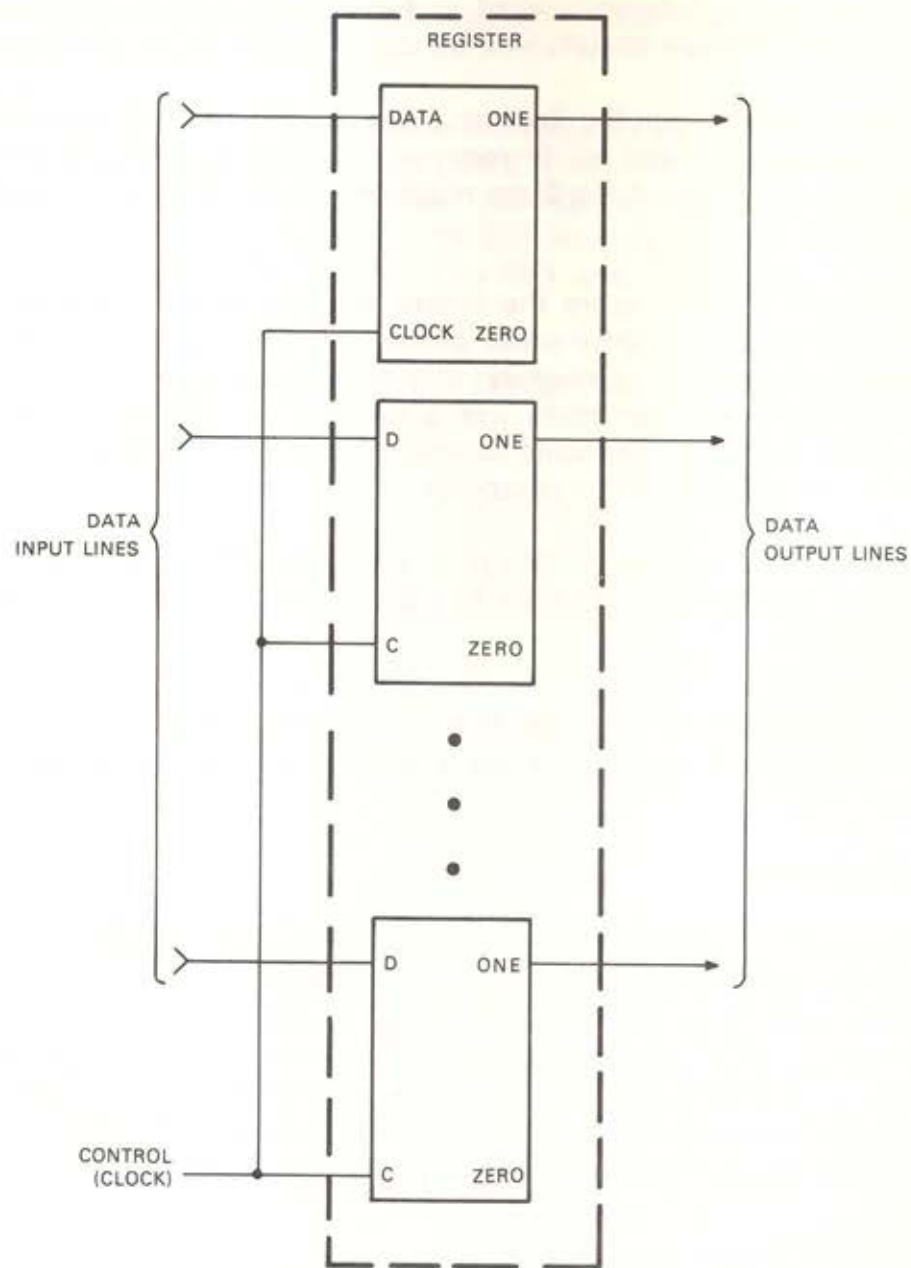


Figure 15 Register Using D-Type Flip-Flops

By shifting the contents of the shift register one place to the *left*, this register *multiplies* the contents of the register by 2. Conversely, shifting the contents of the shift register one position to the *right* is equivalent to *dividing* the entire contents by 2. Thus, the shift register performs mathematical operations simply by shifting the contents of the register. Positions at the right end of the register that are vacated as a result of the shift (least significant bits) are usually filled with zeros. Positions on the left are usually filled with the value that was in the leftmost (most significant) bit before the shift. Figure 16 illustrates the use of the shift register in multiplication and division. When used for division, the least significant bits are lost as a result of the right shift operation.

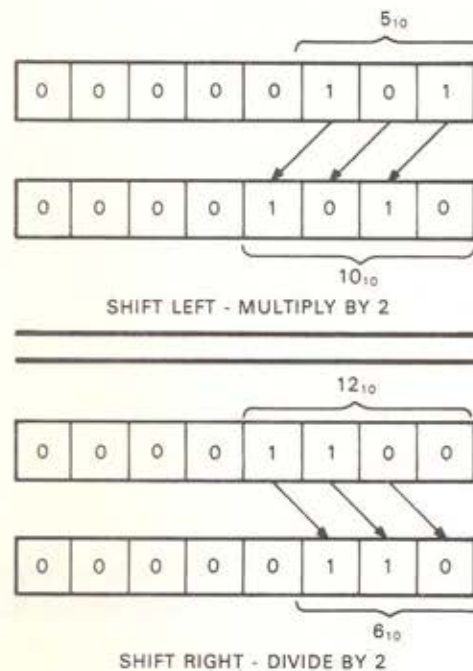


Figure 16 Shift Register Showing Multiply and Divide Arithmetic Operations

Like other types of registers, shift registers are constructed of flip-flops.

Counters

Counters are circuits that count the number of binary 1 inputs. They consist of combinations of flip-flops.

There are two kinds of counters: *up-counters* and *down-counters*. Up-counters start at zero and *increment* by one each time a binary 1 is input in the counter. The counter continues to increase as long as the binary 1s are received.

Down-counters work in the opposite direction. Here, the first number in the counter is the highest number, and the counter *decrements* until its contents read zero.

Counters are used to control various computer functions such as: counting the number of binary 1s on an input line to determine if the correct number has been received; controlling the number of shifts performed by the shift register; and maintaining an up-to-date record of an event, such as the number of words transferred from main memory to auxiliary storage.

EXERCISES

1. Match these terms with the appropriate definition by placing the letter of the term in the parentheses next to the correct definition.
 - a. Counter () Moves stored bits to the right or left.
 - b. D-type flip-flop () Compensates for time differences during data transfers within the computer.
 - c. Buffer register () Binary 1 input increments or decrements the contents by 1.
 - d. Shift register () Repeating the same binary value on the DATA line will have no effect.
2. Which of the following are the basic storage elements of a register?
 - a. AND gates
 - b. OR gates
 - c. XOR gates
 - d. Flip-flops
3. Which type of register is needed to store data, temporarily, from a high-speed unit of a computer until a lower speed unit is ready for it?
 - a. Shift register
 - b. Counter
 - c. Buffer register
 - d. Logic gate

SOLUTIONS

1. Match these terms with the appropriate definition by placing the letter of the term in the parentheses next to the correct definition.

a. Counter	(d) Moves stored bits to the right or left.
b. D-type flip-flop	(c) Compensates for time differences during data transfers within the computer.
c. Buffer register	(a) Binary 1 input increments or decrements the contents by 1.
d. Shift register	(b) Repeating the same binary value on the DATA line will have no effect.

2. Which of the following are the basic storage elements of a register?
 - a. AND gates
 - b. OR gates
 - c. XOR gates
 - ☒ d. Flip-flops

3. Which type of register is needed to store data, temporarily, from a high-speed unit of a computer until a lower speed unit is ready for it?
 - a. Shift register
 - b. Counter
 - ☒ c. Buffer register
 - d. Logic gate

EXERCISES

4. Given an 8-bit shift register with the initial data as shown, insert the data in the shift register as it would appear after the shifts indicated are performed.

1	0	0	1	1	0	0	0
---	---	---	---	---	---	---	---

Register

- a. Shift right two bit places
 - b. Shift left one bit place
5. The binary equivalent of the number 28 is stored in a shift register. A shift right two bit places is executed. The number in the shift register is now:
- a. 28
 - b. 14
 - c. 7
 - d. 56

SOLUTIONS

4. Given an 8-bit shift register with the initial data as shown, insert the data in the shift register as it would appear after the shifts indicated are performed.

1	0	0	1	1	0	0	0
---	---	---	---	---	---	---	---

- a. Shift right two bit places

0	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---

- b. Shift left one bit place

0	0	1	1	0	0	0	0
---	---	---	---	---	---	---	---

5. The binary equivalent of the number 28 is stored in a shift register. A shift right two bit places is executed. The number in the shift register is now:

- a. 28
- b. 14
- c. 7
- d. 56

EXERCISES

6. Select as many of the following as typify possible uses for a counter:
- a. Shifting data right or left
 - b. Maintaining a record of the number of bits transferred within the computer
 - c. Indicating the number of binary 1s on an input line
 - d. Keeping track of the number of times a shift is performed.
7. The current value in an up-counter is 6. Two more counts are made. What value is now present in the counter?
- a. 8
 - b. 6
 - c. 2
 - d. 4

SOLUTIONS

6. Select as many of the following as typify possible uses for a counter:
- a. Shifting data right or left
 - Ⓐ Maintaining a record of the number of bits transferred within the computer
 - Ⓒ Indicating the number of binary 1s on an input line
 - Ⓓ Keeping track of the number of times a shift is performed
7. The current value in an up-counter is 6. Two more counts are made. What value is now present in the counter?
- Ⓐ 8
 - b. 6
 - c. 2
 - d. 4

Take the test for this module and evaluate your answers before studying another module.

Appendix A

More Complex Circuits

Combining Logic Elements

Performing arithmetic and logic operations on data is one of the basic functions of a computer. These operations are done by electronic circuits using various combinations of *logic gates*, *flip-flops*, and *registers*. In this section, we will demonstrate the way in which the *logic elements* discussed thus far can be combined into *logic circuits* that can be used to perform some of these important logic and arithmetic operations.

In our examples, *only* logic gates AND, OR, XOR and NOT and the set-reset and D-type flip-flops will be used in the logic circuits. In practice, many different kinds of logic elements are used in logic circuits.

Adder Circuit

One of the principal operations in the digital computer is the addition of two numbers. With the computer, numbers are represented in *binary* form. To understand the method used by the computer to execute this basic function, we will design a simple logic circuit that will take two binary digits and perform an addition operation.

First, we will review binary arithmetic. The addition operation is quite simple because there are only *two* digits in the binary system. As shown in the following example, there are just four possible combinations:

$$\begin{array}{cccc}
 \begin{array}{r} 0 \\ +0 \\ \hline 0 \end{array} &
 \begin{array}{r} 0 \\ +1 \\ \hline 1 \end{array} &
 \begin{array}{r} 1 \\ +0 \\ \hline 1 \end{array} &
 \begin{array}{r} 1 \\ +1 \\ \hline 10 \end{array} \\
 & & & \uparrow \text{ carry}
 \end{array}$$

Three of the four possible combinations yield simple, single-digit sums. However, in the fourth case, a carry is generated to the *next* digit position.

These arithmetic examples can be expressed in tabular form known as a *truth table*. In the following truth table, A and B represent the two *inputs*, and S and C represent the sum and carry digits, respectively.

Table 13 Truth Table for Binary Arithmetic

Input		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Input		Output
A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

Examining the carry digit first, it becomes apparent that we have a carry *only* when *both* line A and line B have an input of 1. The logic element that produces this result is the AND gate. The carry circuit would be as shown in Figure 17.

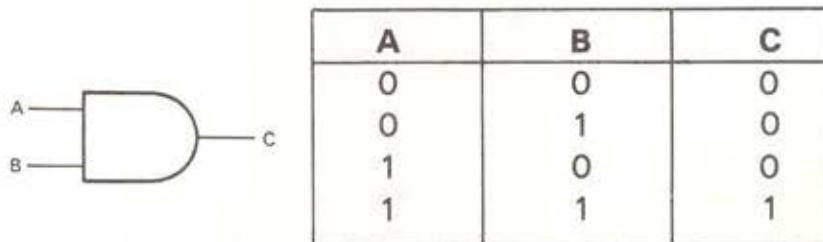


Figure 17 Carry Circuit

The sum circuit is more complex. The sum is 1 if *either* line A or line B has an input of 1, but *not* if both have an input of 1. The logic circuit to perform this function, therefore, must produce a 1 value when the A input is 1 and the B input is 0 or when the A input is 0 and the B input is 1.

Examining the truth table for the sum circuit, we can see that it is identical to the XOR gate. This logic circuit is shown in Figure 18.

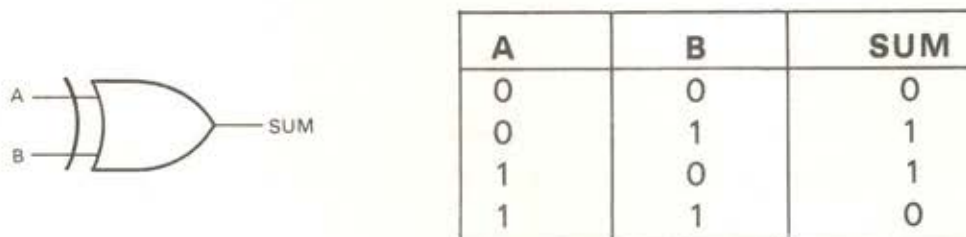


Figure 18 Sum Circuit for Binary Addition

The second circuit we designed, the sum circuit, reveals an interesting phenomenon. As you can see, when the *two* inputs are *identical*, the output is *always* a 0, and when they are *different*, the output is always a 1. This circuit is often called a binary *comparator* because it actually compares two values.

A combination of the carry circuit and the sum circuit, which is illustrated in Figure 19, is called a *half-adder* circuit. Two of these half-adder circuits are needed in order to perform binary addition with two binary digits as input; hence, the name *half-adder*.

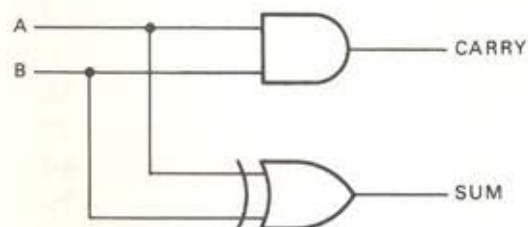


Figure 19 Half-Adder Logic Circuit

Tracing a Logic Circuit

To test or repair computers, it is frequently necessary to determine the output of a logic circuit for a given set of inputs. If the logic circuit is complex, this can be a very tedious and time-consuming task. This section discusses how to trace a logic circuit by showing simple examples of up to four logic gates and one flip-flop to demonstrate this principle.

Two Logic Gate Circuit

To trace a logic circuit completely, the truth table for this circuit should be constructed. Figure 20 illustrates the 3-input logic circuit with a 2-input AND gate and a 2-input XOR gate connected.

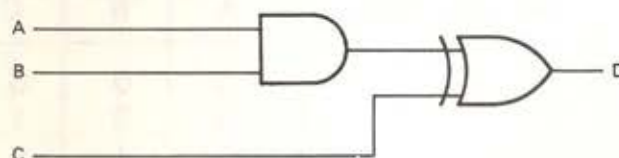


Figure 20 3-Input Logic Circuit

To *specify* this circuit completely, we must first list all possible input values. In this context, specify involves tabulating *all* the possible input and output values. Since there are three inputs to this circuit, each of which can be a 1 or 0, we have 2^3 (or 8) possible input values. Table 14 lists the number of possible input values for up to eight input lines.

Table 14 Input Values

Number of Input Lines	Number of Possible Input Values
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256

As this table illustrates, the number of input values *doubles every time* the number of input lines increases by *one*. There must be a row in the truth table for each value of the input as the number of input lines increases by one. This is another indication of how tedious it could be to attempt to construct a truth table for a logic circuit of any size or complexity.

In the example, eight rows are required for the truth table. There is a special method of listing the input values in a truth table to avoid accidentally omitting any input value. First, the row number is listed, beginning with 0. Then the binary number value for that row is entered as shown in Table 15. In this way, every possible input value is listed consecutively.

Table 15

Row Number	Input Value		
	A	B	C
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Next, we must apply the logic rules to the input for each gate until the final output is reached. In this example, we have A AND B, so the partial truth table after that gate is passed is:

Table 16 Partial Truth Table – A AND B

Row	Inputs			AND Gate Output 1 A AND B
	A	B	C	
0	0	0		0
1	0	0		0
2	0	1		0
3	0	1		0
4	1	0		0
5	1	0		0
6	1	1		1
7	1	1		1

(For the sake of clarity, the C inputs are omitted from the truth table.)

The output of the AND gate labeled ① is input to the XOR gate. The partial truth table is now given in Table 17.

Table 17 Partial Truth Table – C XOR ①

Row	Input			XOR Gate Output	
	A	B	C	① A AND B	C XOR ①
0			0	0	0
1			1	0	1
2			0	0	0
3			1	0	1
4			0	0	0
5			1	0	1
6			0	1	1
7			1	1	0

The output labeled D is the final output and the complete truth table is:

Table 18 Final Truth Table for Circuit Output

Row	Input			Output D
	A	B	C	
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Notice that each logic element requires an output column in the partial truth tables. In general, the size and complexity of the truth table for a logic circuit is determined by the number of inputs and the number of logic elements in the circuit.

Three Logic Gate and One Flip-Flop Circuit

For the next example, a D-type flip-flop and an AND gate will be added to the circuit. Now the circuit is as depicted in Figure 21.

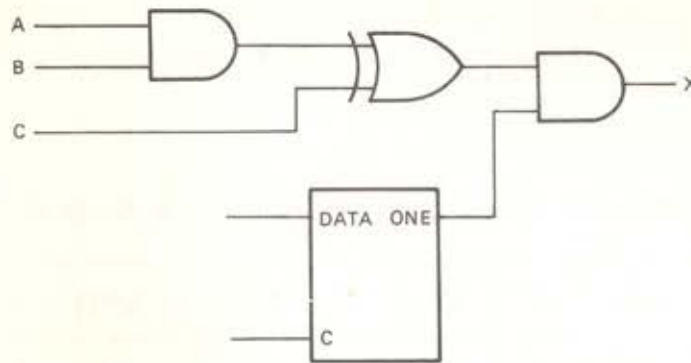


Figure 21 Three Logic Gate and One Flip-Flop Circuit

Since there are four inputs in this extended logic circuit (A, B, C, and the value of the flip-flop), the truth table must contain 16 rows as shown in Table 19.

Table 19 All Possible Input Values for a 4-Input Circuit

Row	A	B	C	ONE
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Again, applying the logic rules to the inputs entering the first AND gate, the partial truth table would be as shown in Table 20.

Table 20 Partial Truth Table – A AND B

Row	A	B	C	ONE	A AND B
0	0	0			0
1	0	0			0
2	0	0			0
3	0	0			0
4	0	1			0
5	0	1			0
6	0	1			0
7	0	1			0
8	1	0			0
9	1	0			0
10	1	0			0
11	1	0			0
12	1	1			1
13	1	1			1
14	1	1			1
15	1	1			1

Now, applying the XOR to the output of the first AND gate, labeled ①, the partial truth table is:

Table 21 Partial Truth Table – C XOR ①

Row	A	B	C	ONE	① A AND B	② C XOR ①
0			0		0	0
1			0		0	0
2			1		0	1
3			1		0	1
4			0		0	0
5			0		0	0
6			1		0	1
7			1		0	1
8			0		0	0
9			0		0	0
10			1		0	1
11			1		0	1
12			0		1	1
13			0		1	0
14			1		1	0
15			1		1	0

Finally, the output of XOR gate labeled ② must be ANDed with the ONE input of the flip-flop. The partial truth table for this logic operation is shown in Table 22.

Table 22 Partial Truth Table – ONE AND ②

Row	A	B	C	ONE	② C XOR ①	ONE AND ②
0				0	0	0
1				1	0	0
2				0	1	0
3				1	1	1
4				0	0	0
5				1	0	0
6				0	1	0
7				1	1	1
8				0	0	0
9				1	0	0
10				0	1	0
11				1	1	1
12				0	1	0
13				1	1	1
14				0	0	0
15				1	0	0

Since this AND is the last logical element, the final truth table (Table 23) can be written that specifies the circuit completely.

Table 23 Complete Truth Table for Extended Circuit

Row	A	B	C	ONE	X
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

As the preceding demonstration proved, developing the complete truth table by hand for even this relatively simple logic circuit (four input lines, three logic gates, and a single flip-flop) was a time-consuming and error-prone task.

Four Logic Gate and a Flip-Flop Circuit

On many occasions the complete specification of a logic circuit is unnecessary as all possible input values may not exist or may be irrelevant. When a particular output for a given set of inputs is irrelevant, it is known as a *don't care* output. In those cases, the number of rows in the truth table for the logic circuit can be substantially reduced, or the output value for a particular set of inputs is obvious without the truth table.

Figure 22 shows a simple logic circuit consisting of four AND gates, labeled 1, 2, 3, and 4, and one D-type flip-flop. This circuit *stores* (writes) one bit of data into the flip-flop, or *reads* a data bit from the flip-flop.

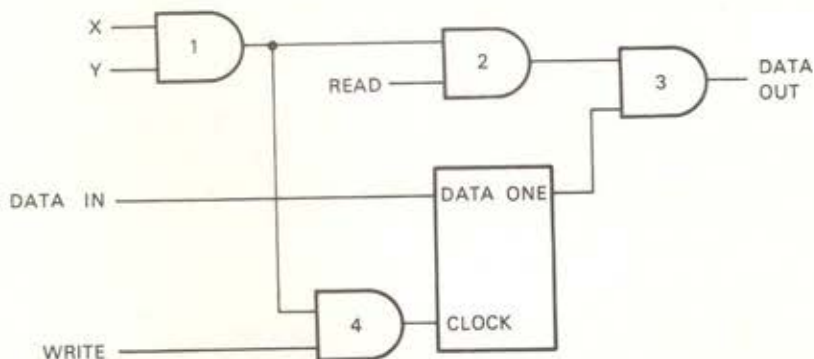


Figure 22 Four Logic Gate, One Flip-Flop Logic Circuit

To better comprehend the operation of this circuit, we will trace its operation with two different sets of inputs. First, we will examine the circuit with a 1 input to the X, Y, and WRITE lines and a 0 to the READ line. When we review this possibility, we will examine the same circuit with a 1 input to the X, Y, and READ lines, and a 0 to the WRITE line. Because the inputs in both cases are limited, a truth table will not be needed.

To initiate a *write* operation, the X and Y inputs must *both* be a 1, and must change to a 1. This causes the output of AND gate 4 to switch from a 0 to a 1. When this 0 to 1 transition (\uparrow) appears at the CLOCK input of the flip-flop, it *sets* the flip-flop (if DATA IN = 1), or *resets* the flip-flop (if DATA IN = 0). In other words, the flip-flop assumes the state of DATA IN when it is triggered by a 0 to 1 transition. During this write operation, the READ input remains a 0.

To initiate a *read* operation, the READ input must be switched to a 1 (X and Y must still be 1s). This causes AND gate 2 to output a 1 which enables gate 3. "Enable" means that data is allowed to pass through AND gate 3. In other words, if a 1 is stored in the flip-flop, a 1 will also appear on the DATA OUT line. Conversely, if a 0 is stored in the flip-flop, a 0 will appear on the DATA OUT line.

Inputs X and Y are used to *enable* or *disable* this circuit. When either or both inputs are 0, gate 1 outputs a 0. This 0, in turn, disables AND gates 2 and 4 so that an input of READ = 1 or WRITE = 1 has no effect on the circuit. Before a read or write operation can take place, X and Y must be a 1.

Logical Equations for Circuits

An alternate method to the truth table for representing a logic circuit is by means of a *logical equation*. The logical equation is a very compact representation for the circuit, but is not always as clear or easy to understand as a truth table. The logical equation for the circuit shown in Figure 20 is:

$$D = (A \cdot B) \oplus C$$

This equation simply states that the input must first be ANDed with the B input and the results XORed with the C input. The following equations are the logical equations for the other circuits presented as examples in this unit.

Circuit	Figure	Equation
Extended Logic	21	$X = ((A \cdot B) \oplus C) \cdot \text{ONE}$
Four Gate, Flip-Flop	22	$\text{DATA-OUT} = ((X \cdot Y) \cdot \text{READ}) \cdot \text{DATA}$

Logical Design in Practice

As we have seen, the design of logical circuits is a complex, time-consuming task. Fortunately, technological advances have almost eliminated the hand method described here. Computer programs are now used to design the complex circuits required by modern computers.

Although the computer has taken over the design work, the general procedure used is similar to the hand method discussed in this lesson. The computer must initially determine the truth tables for the specific circuits. Then the logical equations must be developed and examined by the computer program to determine the most efficient and economical circuit for the task. Ultimately, other computer programs draw the actual circuit diagram.

Manufacturing and Packaging of Logic Components

Today's computers are constructed of highly miniaturized components called *integrated circuits (IC)*. Modern computers owe their tremendous speed and efficiency to these integrated circuits. An entire circuit is placed on a tiny *silicon wafer* (Figure 23). These wafers, or *chips*, can contain hundreds of logic gates and flip-flops that are interconnected internally. Each chip performs specific functions within the computer. For example, a single chip can contain a number of registers or a complex decoder circuit. Technological advances in chip manufacturing and miniaturization have already developed to a high level of sophistication. This method of manufacturing is known as *large-scale integration* or *LSI*.

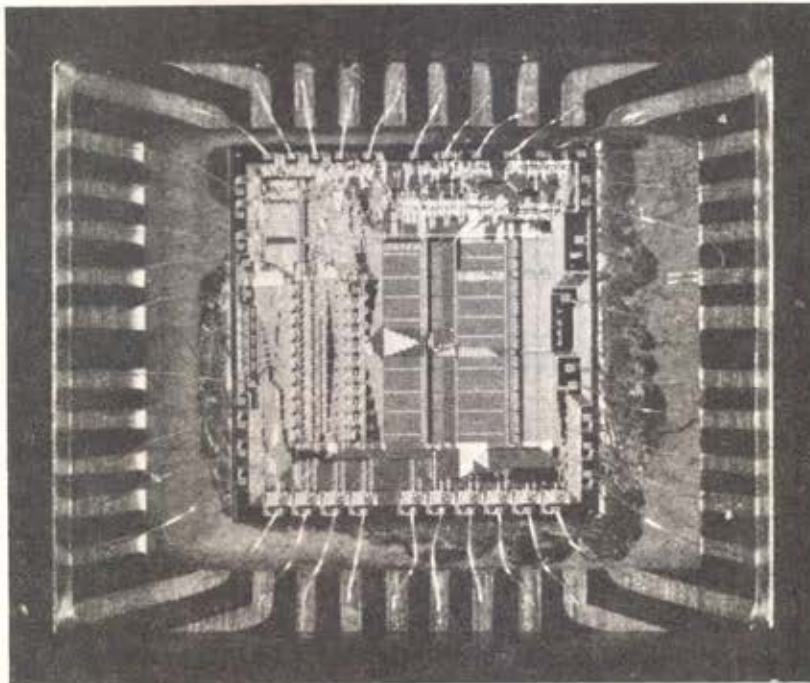


Figure 23 Chip

This silicon chip is enclosed or *packaged* in a plastic case. This type of packaging provides rugged protection and facilitates handling and installation. The external connections for the integrated circuit chips are called *pins*. Many of these integrated circuits can be mounted on a single printed circuit board. Figure 24 depicts one of these circuit boards with a number of integrated circuits.

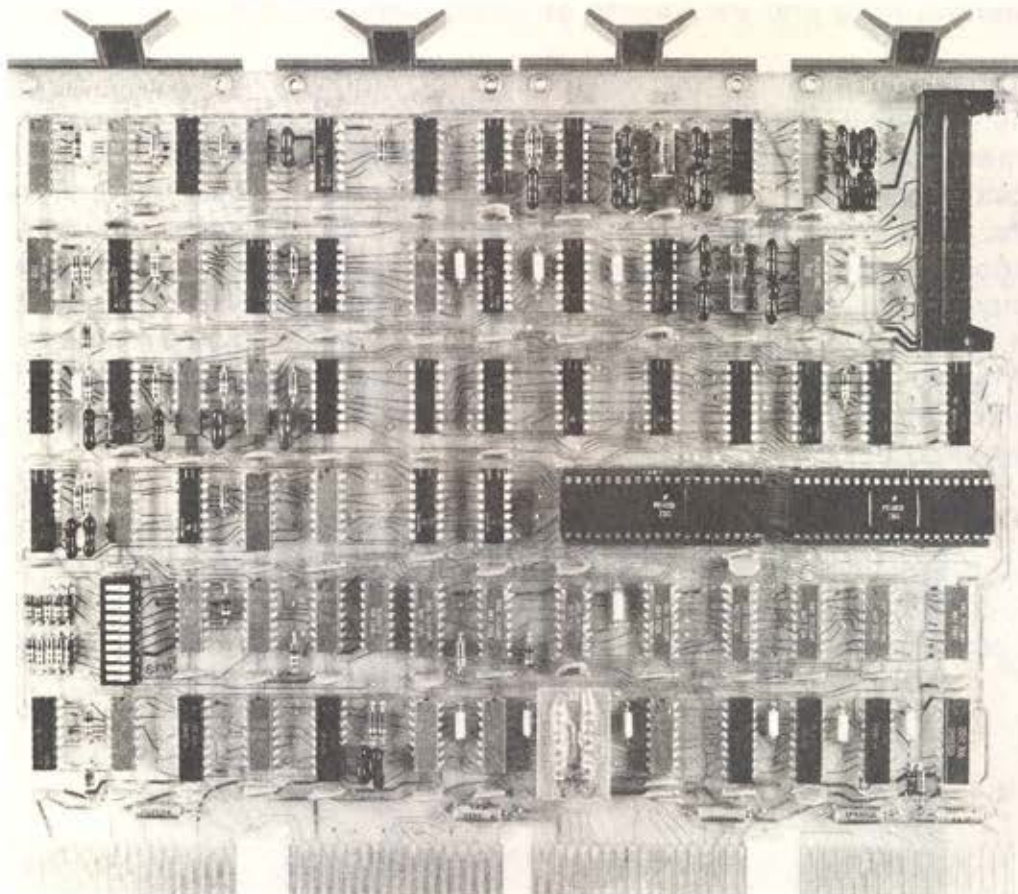


Figure 24 Circuit Board

Integrated circuits provide many advantages over previously developed computer components. Among these advantages are:

- Small size
- Reasonable cost
- Low power requirement
- Extremely high-speed execution of logic operations
- Low heat dissipation

Continued advances are expected in large-scale integration techniques, and logic for computers will become more compact, cheaper, and more efficient.

Take the test for this module and evaluate your answers before studying another module.