PDHonline Course E378 (4 PDH)

# Digital Logic Systems Volume II - Fundamental Logic Circuits

*Instructor: Lee Layton, PE*

**2020**

**PDH Online | PDH Center**

5272 Meadow Estates Drive
Fairfax, VA 22030-6658
Phone: 703-988-0088
www.PDHonline.com

An Approved Continuing Education Provider

# Digital Logic Circuits
# Volume II
# Fundamental Logic Circuits

*Lee Layton, P.E*

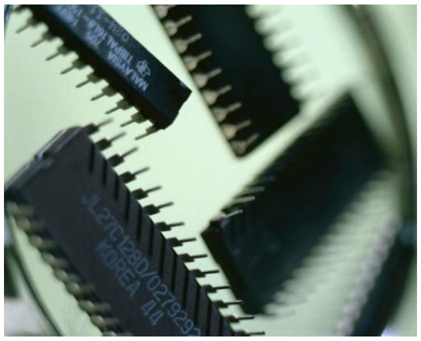
## Table of Contents

**Section** **Page**

This series of courses are based on the Navy Electricity and Electronics Training Series (NEETS) section on Logic systems. The NEETS material has been reformatted for readability and ease of use as a continuing education course. The NEETS series is produced by the Naval Education and Training Professional Development and Technology Center.

## Introduction

In the first course in this series, Digital Logic Circuits, Volume I, Introduction to Logic, we learned that the two digits of the binary number system can be represented by the state or condition of electrical or electronic devices. A binary 1 can be represented by a switch that is closed, a lamp that is lit, or a transistor that is conducting. Conversely, a binary 0 would be represented by the same devices in the opposite state: the switch open, the lamp off, or the transistor in cut-off.



In this, the second course in the series, we will study the four basic logic gates that make up the foundation for digital equipment. We will see the types of logic that are used in equipment to accomplish the desired results.

This course includes an introduction to Boolean algebra, the logic mathematics system used with digital equipment. Certain Boolean expressions are used in explanation of the basic logic gates, and their expressions will be used as each logic gate is introduced.

# Chapter 1
# Computer Logic

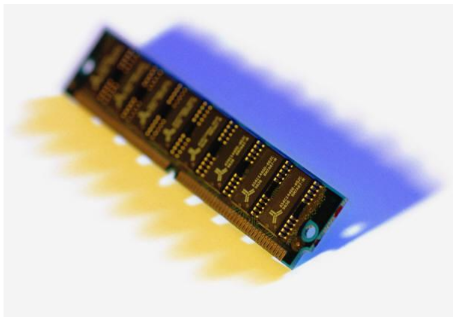
Logic is defined as the science of reasoning. In other words, it is the development of a reasonable or logical conclusion based on known information.

Consider the following example: If it is true that all non-penny, US coins are silver and the Nickel is a coin, then you would reach the logical conclusion that the Nickel is silver. To reach a logical conclusion, you must assume the qualifying statement is a condition of truth. For each statement there is also a corresponding false condition. The statement "A Nickel is a US coin" is true; therefore, the statement "A Nickel is not a US coin" is false. There are no *in-between* conditions.

Computers operate on the principle of logic and use the *TRUE* and *FALSE* logic conditions of a logical statement to make a programmed decision.

The conditions of a statement can be represented by symbols, called *variables*; for instance, the statement "Today is Tuesday" might be represented by the symbol T. If today actually is Tuesday, then T is TRUE. If today is not Tuesday, then T is FALSE. As we can see, a statement has two conditions. In computers, these two conditions are represented by electronic circuits operating in two *logic states*. These logic states are 0 (*zero*) and 1 (*one*). Respectively, 0 and 1 represent the FALSE and TRUE conditions of a statement.

When the TRUE and FALSE conditions are converted to electrical signals, they are referred to as *logic levels* called *HIGH* and *LOW*. The 1 state might be represented by the presence of an electrical signal (HIGH), while the 0 state might be represented by the absence of an electrical signal (LOW).



If the statement "Today is Tuesday" is FALSE, then the statement "Today is NOT Tuesday" must be TRUE. This is called the *complement* of the original statement. In the case of computer math, complement is defined as the opposite or negative form of the original statement or variable. If today were Tuesday, then the statement "Today is not Tuesday" would be FALSE. The complement is shown by placing a bar, or *vinculum*, over the statement symbol (in this case, $\overline{T}$ ). This variable is spoken as NOT T. Table 1 shows this concept and the relationship with logic states and logic levels.

**Table 1**
**Relationship of Digital Logic and Terms**

| Statement | Symbol | Condition | Logic State | Logic Level |
|---|---|---|---|---|
| Example 1: Assume today is Tuesday | | | | |
| **Original** Today is Tuesday | T | True | 1 | High |
| **Complement:** Today is Not Tuesday | $\overline{T}$ | False | 0 | Low |
| Example 2: Assume today is *not* Tuesday | | | | |
| **Original:** Today is Tuesday | T | False | 0 | Low |
| **Complement:** Today is Not Tuesday | $\overline{T}$ | True | 1 | High |

In some cases, more than one variable is used in a single expression. For example, the expression $AB\overline{C}D$ is spoken "A AND B AND NOT C AND D."

**Positive and Negative Logic**

To this point, we have been dealing with one type of *logic polarity*, positive. Let's further define logic polarity and expand to cover in more detail the differences between positive and negative logic.

Logic polarity is the type of voltage used to represent the logic 1 state of a statement. We have determined that the two logic states can be represented by electrical signals. Any two distinct voltages may be used. For instance, a positive voltage can represent the 1 state, and a negative voltage can represent the 0 state. The opposite is also true.

Logic circuits are generally divided into two broad classes according to their polarity - positive logic and negative logic. The voltage levels used and a statement indicating the use of positive or negative logic will usually be specified on logic diagrams supplied by manufacturers.

In practice, many variations of logic polarity are used; for example, from a high-positive to a low positive voltage, or from positive to ground; or from a high-negative to a low-negative voltage, or from negative to ground. A brief discussion of the two general classes of logic polarity is presented in the following paragraphs.

Positive Logic

Positive logic is defined as follows: If the signal that activates the circuit (the 1 state) has a voltage level that is more *positive* than the 0 state, then the logic polarity is considered to be *positive*. Table 2 shows the manner in which positive logic may be used.

**Table 2**
**Example of Positive Logic**

| | |
|---|---|
| Example 1 | Active signal – True, 1, High = +10 volts<br><br>Complement – False, 0, Low = 0 volts |
| Example 2 | Active signal – True, 1, High = 0 volts<br><br>Complement – False, 0, Low = -10 volts |

As we can see, in positive logic the 1 state is at a more positive voltage level than the 0 state.

## Negative Logic

As you might suspect, negative logic is the opposite of positive logic and is defined as follows: If the signal that activates the circuit (the 1 state) has a voltage level that is more *negative* than the 0 state, then the logic polarity is considered to be *negative*. Table 3 shows the manner in which negative logic may be used.

**Table 3**
**Examples of Negative Logic**

| | |
|---|---|
| Example 1 | Active signal – True, 1, High = +5 volts<br><br>Complement – False, 0, Low = +10 volts |
| Example 2 | Active signal – True, 1, High = -10 volts<br><br>Complement – False, 0, Low = -5 volts |

The logic level LOW now represents the 1-state. This is because the 1-state voltage is more negative than the 0-state.

In the examples shown for negative logic, you notice that the voltage for the logic 1 state is more negative with respect to the logic 0-state voltage. This holds true in example 1 where both voltages are positive. In this case, it may be easier for you to think of the TRUE condition as being less positive than the FALSE condition. Either way, the end result is negative logic. The use of positive or negative logic for digital equipment is a choice to be made by design engineers. The difficulty for the technician in this area is limited to understanding the type of logic being used and keeping it in mind when troubleshooting.

**Note:** Unless otherwise noted, the remainder of this course will deal only with positive logic.

## Logic Inputs and Outputs

As you study logic circuits, you will see a variety of symbols used to represent the inputs and outputs. The purpose of these symbols is to let you know what inputs are required for the desired output.

If the symbol A is shown as an input to a logic device, then the logic level that represents A must be *HIGH* to activate the logic device. That is, it must satisfy the input requirements of the logic device before the logic device will issue the TRUE output.

Look at view A of Figure 1. The symbol X represents the input. As long as the switch is open, the lamp is not lit. The open switch represents the logic 0 state of variable X.

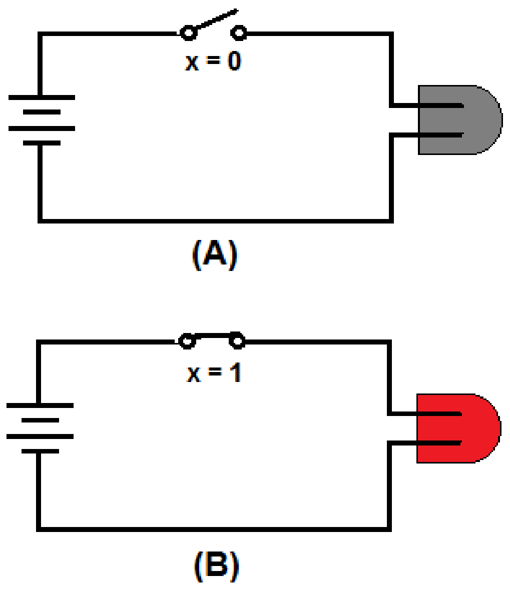


Figure 1

Closing the switch (view B), represents the logic 1 state of X. Closing the switch completes the circuit causing the lamp to light. The 1 state of X satisfied the input requirement and the circuit therefore produced the desired output (logic HIGH); current was applied to the lamp causing it to light.

If you consider the lamp as the output of a logic device, then the same conditions exist. The TRUE (1 state) output of the logic device is to have the lamp lit. If the lamp is not lit, then the output of the logic device is FALSE (0 state).

As you study logic circuits, it is important that you remember the state (1 or 0) of the inputs and outputs.

So far in this chapter, we have discussed the two conditions of logical statements, the logic states representing these two conditions, logic levels and associated electrical signals and positive and negative logic. We are now ready to proceed with individual logic device operations. These make up the majority of computer circuitry.

As each of the logic devices are presented, a chart called a *truth table* will be used to illustrate all possible input and corresponding output combinations. Truth Tables are particularly helpful in understanding a logic device and for showing the differences between devices.

The logic operations you will study in chapter two are the AND, OR, NOT, NAND, and NOR. The devices that accomplish these operations are called *logic gates*, or more informally, *gates*. These gates are the foundation for all digital equipment. They are the "decision-making" circuits of computers and other types of digital equipment. By making decisions, we mean that certain conditions must exist to produce the desired output.

In studying each gate, we will introduce various mathematical *symbols* known as *Boolean Algebra* expressions. These expressions are nothing more than descriptions of the input requirements necessary to activate the circuit and the resultant circuit output.

# Chapter 2
# Basic Logic Gates

The basic building blocks of digital logic systems are the AND gate, OR gate, NOT gate, NAND gate, and a NOR gate. In this chapter we will study each of these gates, looking at their output signals and truth tables.

**AND Gates**

The *AND* gate is a logic circuit that requires all inputs to be TRUE at the same time in order for the output to be TRUE.

The standard symbol for the AND gate is shown in Figure 2. Variations of this standard symbol may be encountered. These variations become necessary to illustrate that an AND gate may have more than one input.

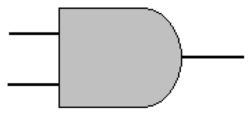


**Figure 2**

If we apply two variables, A and B, to the inputs of the AND gate, then both A and B would have to be TRUE at the same time to produce the desired TRUE output. . The symbol f designates the output function. The *Boolean expression* for this operation is f = A*B or f = AB. The asterisk, or lack of, indicates the AND function. The expression is spoken, "f = A AND B."

We can demonstrate the operation of the AND gate with a simple circuit that has two switches in series as shown in Figure 3. You can see that both switches would have to be closed at the same time to light the lamp (view A). Any other combination of switch positions (view B) would result in an open circuit and the lamp would not light (logic 0).

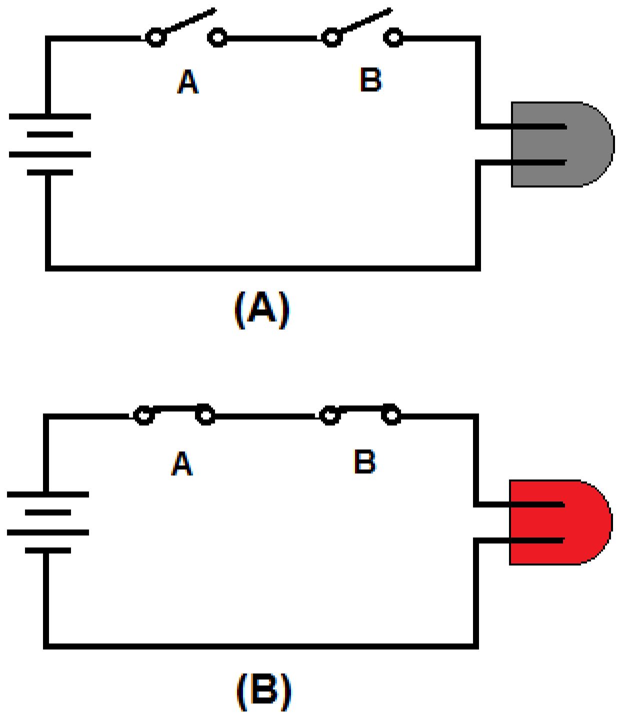


Figure 3

Now look at Figure 4. Signal A is applied to one input of the AND gate and signal B to the other. At time $T_0$, both inputs are LOW (logic 0) and f is LOW. At $T_1$, A goes HIGH (logic 1); B remains LOW; and as a result, f remains LOW. At $T_2$, A goes LOW and B goes HIGH; f, however, is still LOW, because the proper input conditions have not been satisfied (A and B both HIGH at the same time). At $T_4$, both A and B are HIGH. As a result, f is HIGH. The input requirements have been satisfied, so the output is HIGH (logic 1).

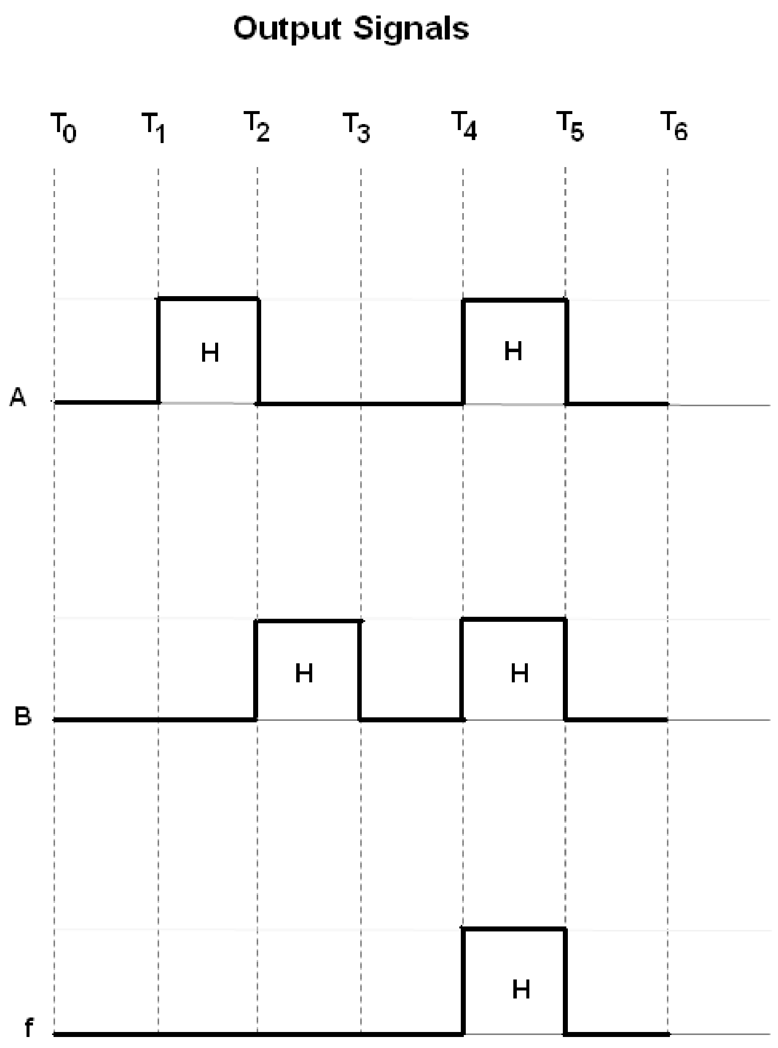


**Figure 4**

Now let's refer to Figure 5. As you can see, a *Truth Table* and a *Table of Combinations* are shown. The latter is a deviation of the Truth Table. It uses the HIGH and LOW logic levels to depict the gate's inputs and resultant output combinations rather than the 1 and 0 logic states. By comparing the inputs and outputs of the two tables, you see how one can easily be converted to the other (remember, 1 = HIGH and 0 = LOW). The Table of Combinations is shown here only to familiarize you with its existence; it will not be seen again in this course. As we mentioned earlier, the Truth Table is a chart that shows all possible combinations of inputs and the resulting outputs. Compare the AND gate Truth Table (Figure 5) with the input signals shown in Figure 4.

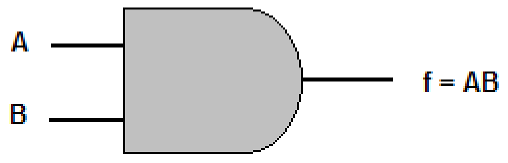


**Figure 5**

| **Truth Table** | | |
|---|---|---|
| **A** | **B** | **F** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| **Table of Combinations** | | |
|---|---|---|
| **A** | **B** | **f** |
| L | L | L |
| L | H | L |
| H | L | L |
| H | H | H |

The first combination (A = 0, B = 0) corresponds to $T_0$ in Figure 4; the second to $T_1$; the third to $T_2$; and the last to $T_4$. When constructing a Truth Table, you must include all possible combinations of the inputs, including the all 0's combination.

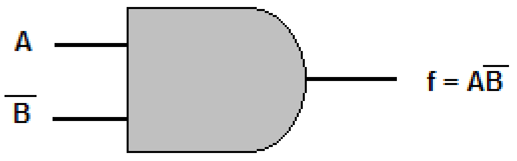
A Truth Table representing an AND gate with three inputs (X, Y, and Z) is shown below. Remember that the two-input AND gate has four possible combinations, with only one of those combinations providing a HIGH output. An AND gate with three inputs has eight possible combinations, again with only one combination providing a HIGH output. Make sure you include all possible combinations. To check if you have all combinations, raise 2 to the power equal to the number of input variables. This will give you the total number of possible combinations. For example: With inputs AB, the combinations are $2^2$, or 4 combinations. With inputs XYZ, the combinations are $2^3$, or 8 combinations.

| **Truth Table** | | | |
|---|---|---|---|
| **X** | **Y** | **Z** | **f** |
| 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 |
| **f = XYZ** | | | |

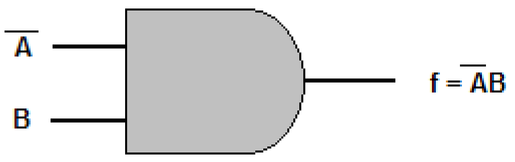
As with all AND gates, all the inputs must be HIGH at the same time to produce a HIGH output. Don't be confused if the complement of a variable is used as an input. When a complement is indicated as an input to an AND gate, *it must also be HIGH to satisfy the input requirements of the gate*. The Boolean expression for the output is formulated based on the TRUE inputs that give a TRUE output. Here is an adage that might help you better understand the AND gate:

**In order to produce a 1 output, all the inputs must be 1. If any or all of the inputs is/are 0, then the output will be 0.**

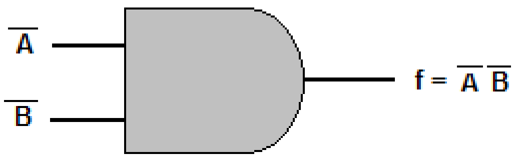
Referring to the following examples should help you cement this concept in your mind. Remember, the inputs, whether the original variable or the complement must be high in order for the output to be high. The three examples given are all AND gates with two inputs. Keep in mind the Boolean expression for the output is the result of all the inputs being HIGH.



| A | $\overline{B}$ | f |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| **f = $A\overline{B}$** | | |



| $\overline{A}$ | B | f |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| **f = $\overline{A}B$** | | |



| $\overline{A}$ | $\overline{B}$ | f |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| **f = $\overline{A}\,\overline{B}$** | | |

You will soon be able to recognize the Truth Table for the other types of logic gates without having to look at the logic symbol.

**OR Gates**

The OR gate differs from the AND gate in that only ONE input has to be HIGH to produce a HIGH output. An easy way to remember the OR gate is that any HIGH input will yield a HIGH output.

Figure 6 shows the standard symbol for the OR gate. The number of inputs will vary according to the needs of the designer.

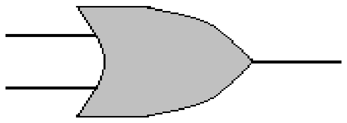


Figure 6

The OR gate may also be represented by a simple circuit as shown in Figure 7. In the OR gate, two switches are placed in parallel. If either or both of the switches are closed (view A), the lamp will light. The only time the lamp will not be lit is when both switches are open (view B).

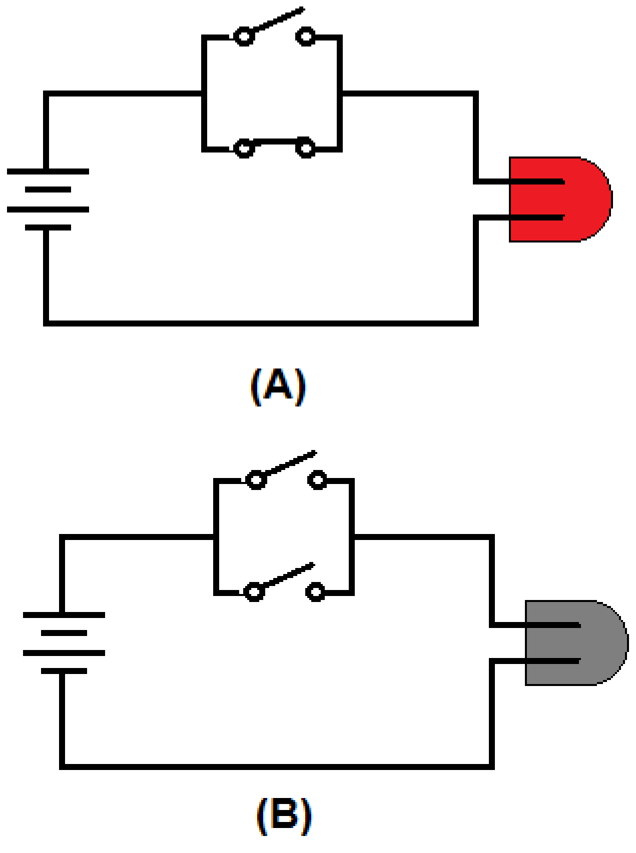


Figure 7

Let's assume we are applying two variables, X and Y, to the inputs of an OR gate. For the circuit to produce a HIGH output, either variable X, variable Y, or both must be HIGH. The Boolean expression for this operation is f = X+Y and is spoken "f equals X OR Y." The plus sign indicates the OR function and should not be confused with addition.

Look at Figure 8. At time $T_0$, both X and Y are LOW and f is LOW. At $T_1$, X goes HIGH producing a HIGH output. At $T_2$ when both inputs go LOW, f goes LOW. When Y goes HIGH at $T_3$, f also goes HIGH and remains HIGH until both inputs are again LOW. At $T_5$, both X and Y go HIGH causing f to go HIGH.

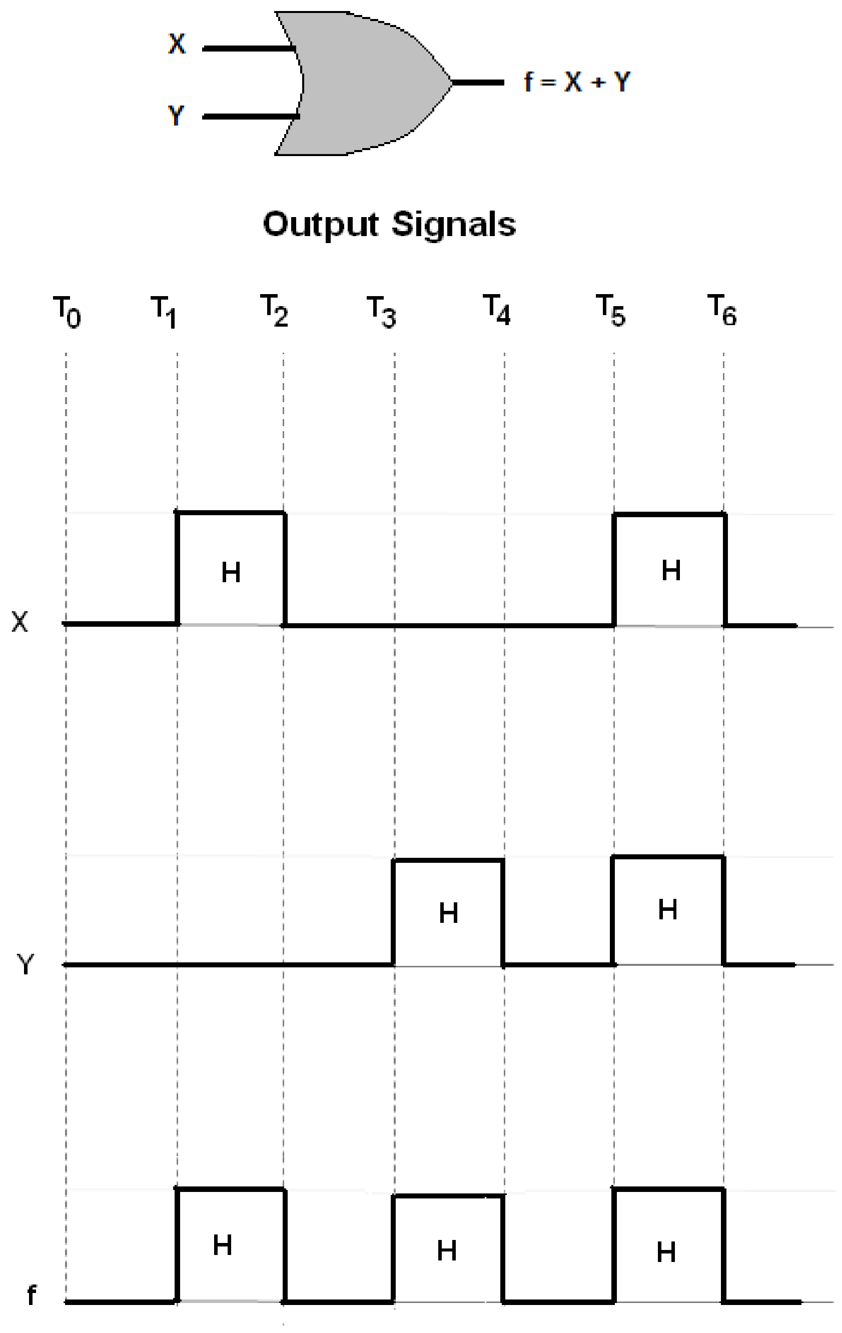


Figure 8

Using the inputs X and Y, let's construct a Truth Table for the OR gate. We can see from the discussion of Figure 8 that there are four combinations of inputs. List each of these combinations of inputs and the respective outputs and you have the Truth Table for the OR gate.

| **Truth Table** | | |
|---|---|---|
| **X** | **Y** | **F** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |
| **f = X + Y** | | |

When writing or stating the Boolean expression for an OR gate with more than two inputs, simply place the OR sign (+) between each input and read or state the sign as OR. For example, the Boolean expression for an OR gate with the inputs of A, B, C, and D would be:

$$f = A+B+C+D$$

This expression is spoken "f equals A OR B OR C OR D."

We can substitute the complements for the original statements as we did with the AND gate or use negative logic; but for an output from an OR gate, at least one of the inputs must be TRUE.

**Inverters**

The *INVERTER*, often referred to as a NOT gate, is a logic device that has an output opposite of the input. It is sometimes called a *negator*. It may be used alone or in combination with other logic devices to fulfill equipment requirements.

When an inverter is used alone, it is represented by the symbol shown in Figure 9 (view A). It will more often be seen in conjunction with the symbol for an amplifier (view B). Symbols for inverters used in combination with other devices will be shown later in the chapter.

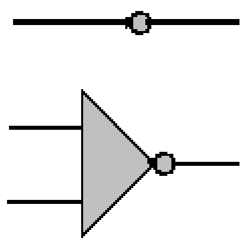


Figure 9

Let's go back to the statement "Today is Tuesday." We stated that T represents the TRUE state. If we apply T to the input of the inverter as shown in Figure 10, then the output will be the opposite of the input. The output, in this case, is T. At times $T_0$ through $T_2$, T is LOW. Consequently, the output ( T ) is HIGH. At $T_2$, T goes HIGH and as a result T goes LOW. T remains LOW as long as T is HIGH and vice versa. The Boolean expression for the output of this gate is f = T.

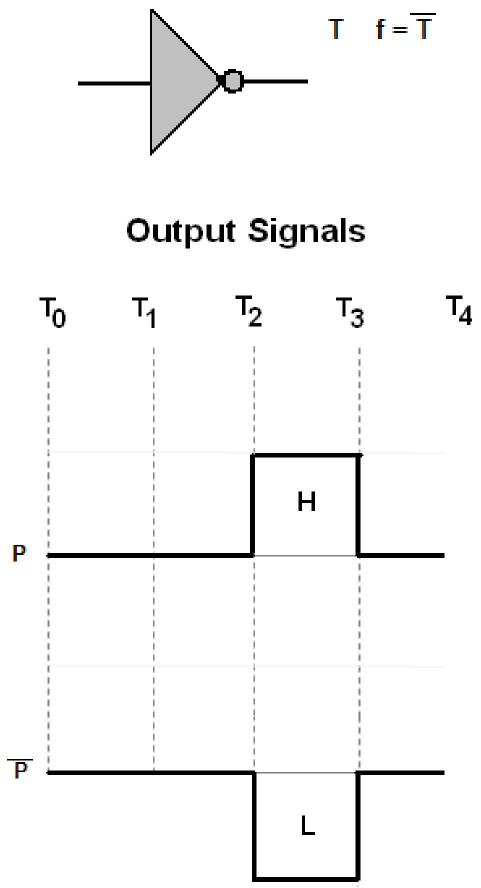
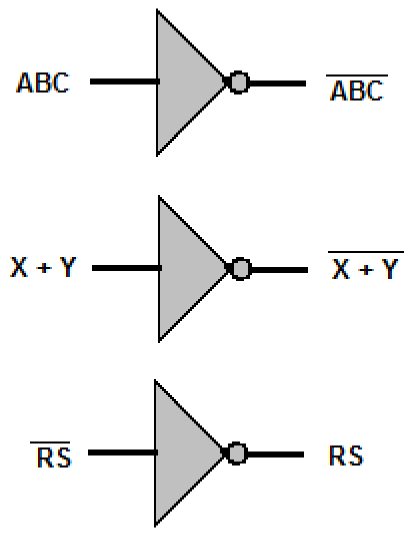


**Figure 10**

You will recall that $\overline{T}$ is the complement of T. The Truth Table for an inverter is shown below.

| **Truth Table** | |
|---|---|
| **T** | **F** |
| 0 | 1 |
| 1 | 0 |

The output of an inverter will be the complement of the input. The following examples show various inputs to inverters and the resulting outputs:



The *vinculum*, or NOT sign, is placed over the entire output or removed from the output, depending on the input. If we applied ABC to an inverter, the output would be $\overline{ABC}$ . And if we ran that output through another inverter, the output would be ABC.

**NAND Gates**

The NAND gate is another logic device commonly found in digital equipment. This gate is simply an AND gate with an inverter (NOT gate) at the output.

The logic symbol for the NAND gate is shown in Figure 11.

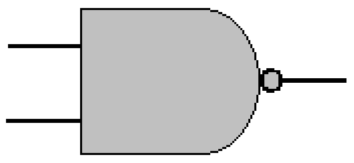


**Figure 11**

The NAND gate can have two or more inputs. The output will be LOW only when all the inputs are HIGH. Conversely, the output will be HIGH when any or all of the inputs are LOW. The NAND gate performs two functions, AND and NOT. Separating the NAND symbol to show these two functions would reveal the equivalent circuits depicted in Figure 12. This should help you better understand how the NAND gate functions.

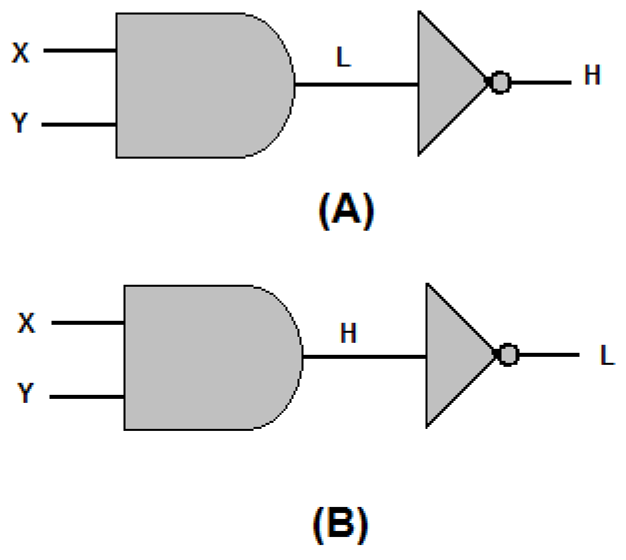
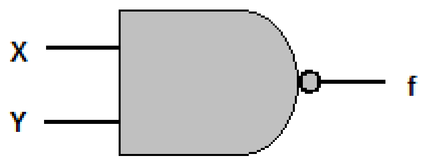


Figure 12

Inputs X and Y are applied to the AND gate. If either X or Y or both are LOW (view A), then the output of the AND gate is LOW. A LOW (logic 0) on the input of the inverter results in a HIGH (logic 1) output. When both X and Y are HIGH (view B), the output of the AND gate is HIGH; thus the output of the inverter is LOW. The Boolean expression for the output of a NAND gate with these inputs is f = $\overline{XY}$. The expression is spoken "X AND Y quantity NOT." The output of any NAND gate is the negation of the input. For example, if our inputs are X and $\overline{Y}$, the output will be $\overline{X\overline{Y}}$ .

Now, let's observe the logic level inputs and corresponding outputs as shown in Figure 13. At time $T_0$, X and Y are both LOW. The output is HIGH; the opposite of an AND gate with the same inputs. At $T_1$, X goes HIGH and Y remains LOW. As a result, the output remains HIGH. At $T_2$, X goes LOW and Y goes HIGH. Again, the output remains HIGH. When both X and Y are HIGH at $T_4$, the output goes LOW.

The output will remain LOW only as long as both X and Y are HIGH.



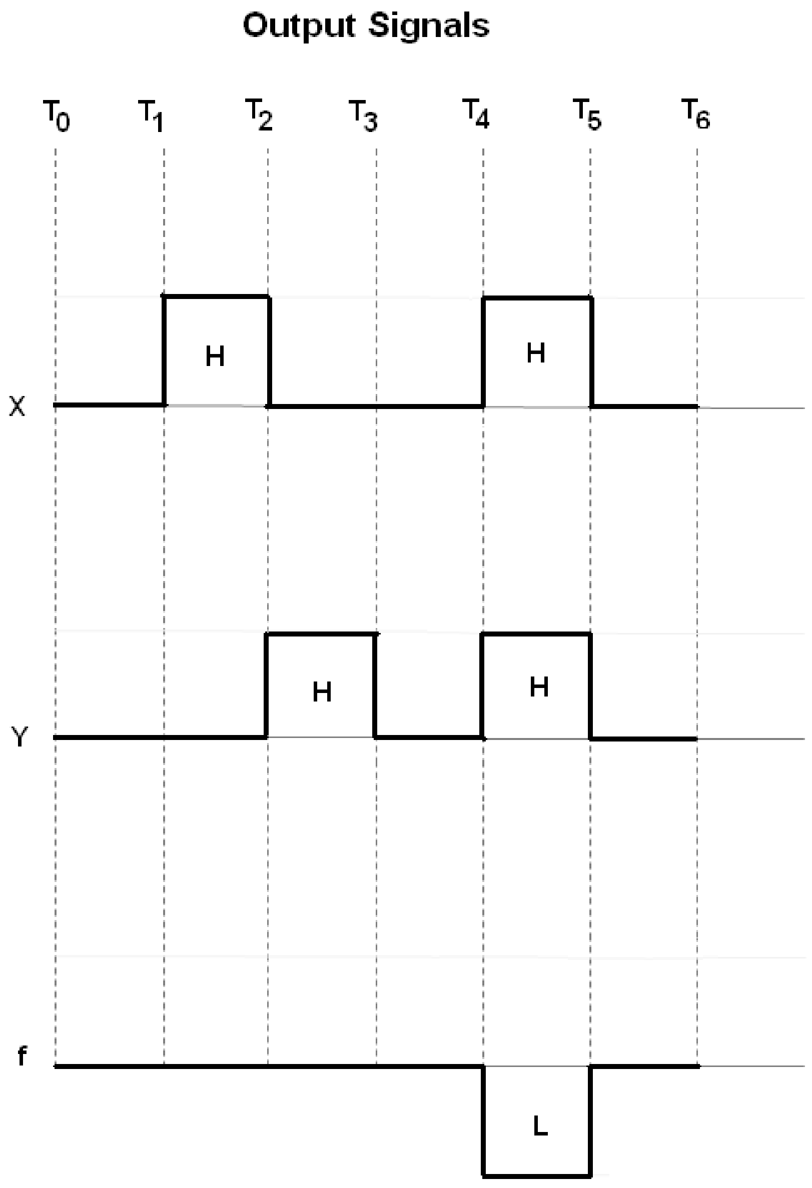


Figure 13

The Truth Table for a NAND gate with X and Y as inputs is shown below.

| Truth Table | | |
|---|---|---|
| **X** | **Y** | **F** |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| $\mathbf{f = \overline{XY}}$ | | |

**NOR Gates**

As we might expect, the NOR gate is an OR gate with an inverter on the output.

The standard logic symbol for this gate is shown in Figure 14. More than just the two inputs may be shown.

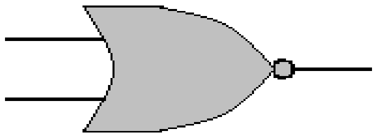


Figure 14

The NOR gate will have a HIGH output only when all the inputs are LOW. When broken down, the two functions performed by the NOR gate can be represented by the equivalent circuit depicted in Figure 15. When both inputs to the OR gate are LOW, the output is LOW. A LOW applied to an inverter gives a HIGH output. If either or both of the inputs to the OR gate are HIGH, the output will be HIGH. When this HIGH output is applied to the inverter, the resulting output is LOW. The Boolean expression for the output of this NOR gate is $f = \overline{K + L}$. The expression is spoken, "K OR L quantity NOT."

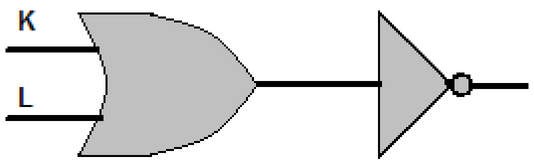
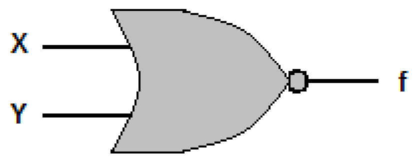


Figure 15

The logic level inputs and corresponding outputs for a NOR gate are shown in Figure 16. At time $T_0$, both K and L are LOW; as a result, f is HIGH. At $T_1$, K goes HIGH, L remains LOW, and f goes LOW. At $T_2$, K goes LOW, L goes HIGH, and the output remains LOW. The output goes HIGH again at $T_3$ when both inputs are LOW. At $T_4$ when both inputs are HIGH, the output goes LOW and remains LOW until $T_5$ when both inputs go LOW. Remember the output is just opposite of what it would be for an OR gate.



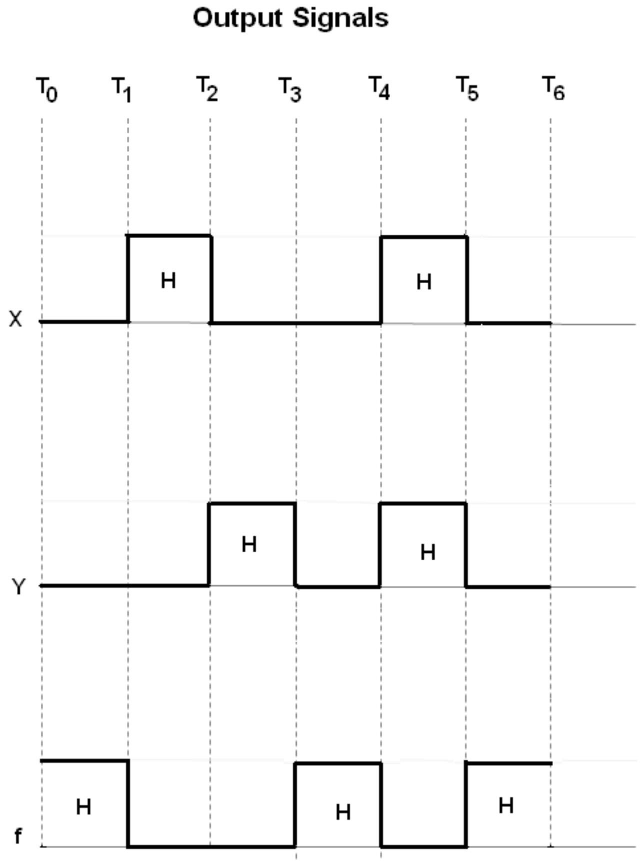


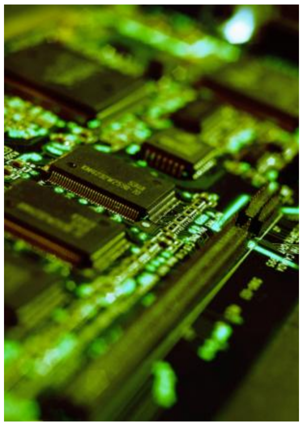
Figure 16

The Truth Table for a NOR gate with K and L as inputs is shown below.

| Truth Table | | |
|---|---|---|
| **K** | **L** | **f** |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| $f = \overline{K + L}$ | | |

# Chapter 3
# Variations of Fundamental Gates

Now that we are familiar with fundamental logic gates, let's look at some variations of these gates that we may encounter.



Up to now we have seen inverters used alone or on the output of AND and OR gates. Inverters may also be used on one or more of the inputs to the logic gates. Take a look at the examples as discussed in the following paragraphs.

**AND/NAND Gate Variations**

If we place an inverter on one input of a two-input AND gate, the output will be quite different from that of the standard AND gate.

In Figure 17, we have placed an inverter on the A input. When A is HIGH, the inverter makes it a LOW going into the AND gate. In order for the output to be HIGH, A would have to be LOW while B is HIGH, as shown in the Truth Table. If the inverter were on the B input, the output expression would then be f = AB.

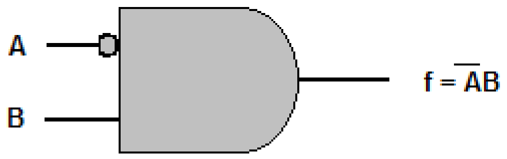


Figure 17

| Truth Table | | |
|---|---|---|
| **A** | **B** | **f** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| **f = $\overline{A}$ B** | | |

Now let's compare a NAND gate to an AND gate with an inverter on each input. Figure 18 shows these gates and the associated Truth Tables. With the NAND gate (view A), the output is HIGH when either or both inputs is/are LOW. The AND gate with inverters on each input (view B), produces a HIGH output only when both inputs are LOW. This comparison also points out the differences between the expressions f = AB (A AND B quantity NOT) and f = $\overline{A}\,\overline{B}$ (NOT A AND NOT B).

Now, look over the Truth Tables for Figures 17, 18, and 19; look at how the outputs vary with inverters in different positions.

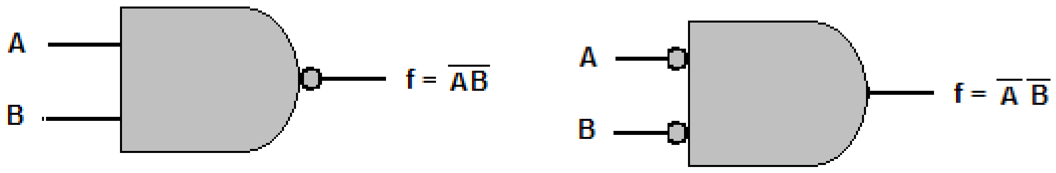


Figure 18

| A | B | f |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| A | B | f |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

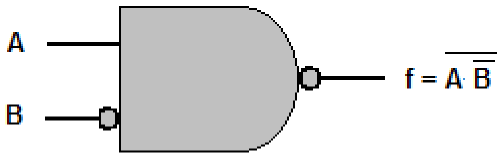


Figure 19

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| **$f = \overline{A\,B}$** | | |

**OR/NOR Gate Variations**

The outputs of OR and NOR gates may also be changed with the use of inverters.
An OR gate with one input inverted is shown in Figure 20. The output of this OR gate requires that A be LOW, B be HIGH, or both of these conditions existing at the same time in order to have a HIGH output. Since the A input is inverted, it must be LOW if B is LOW in order to produce a HIGH output.

Therefore the output is f = $\overline{A}$ +B.

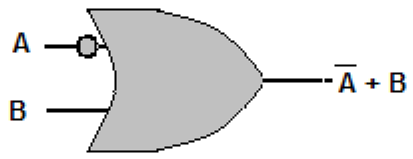
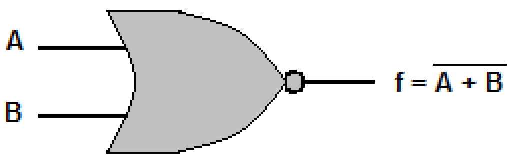


Figure 20

| Truth Table | | |
|---|---|---|
| **A** | **B** | **f** |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| **$f = \overline{A} + B$** | | |

Figure 21compares a NOR gate (view A), to an OR gate with inverters on both inputs (view B), and shows the respective Truth Tables. The NOR gate will produce a HIGH output only when both inputs are LOW. The OR gate with inverted inputs produces a HIGH output with all input combinations EXCEPT when both inputs are HIGH. This Figure also illustrates the differences between the expressions $f = \overline{A + B}$ (A OR B quantity NOT) and $f = \overline{A} + \overline{B}$ (NOT A OR NOT B).



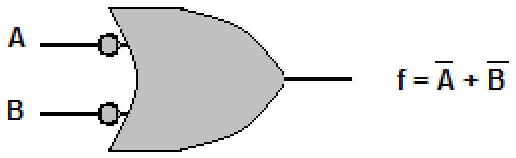


Figure 21

| A | B | f |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| **$f = \overline{A + B}$** | | |

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| **$f = \overline{A} + \overline{B}$** | | |

As with the NAND gate, one or more inputs to NOR gates may be inverted. Figure 22 shows the result of inverting a NOR gate input. In this case, because of the inversion of the B input and the inversion of the output, the only time this gate will produce a HIGH output is when A is LOW and B is HIGH. The output Boolean expression for this gate is $f = \overline{A + \overline{B}}$, spoken "A OR NOT B quantity NOT."

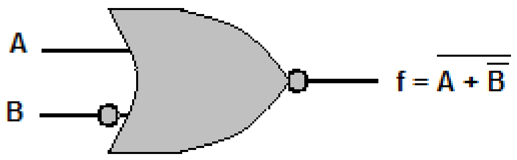


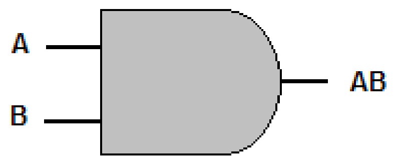
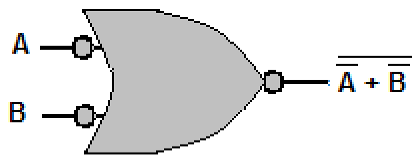
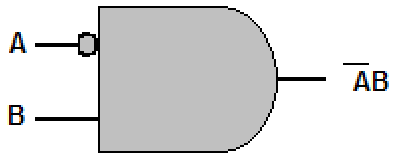
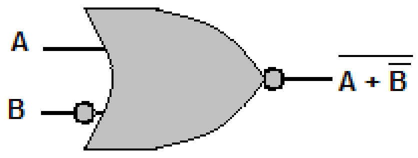
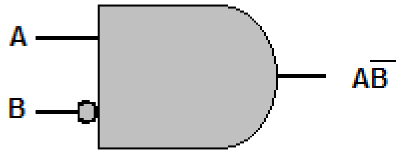
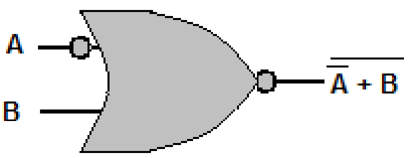
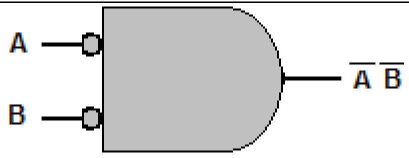
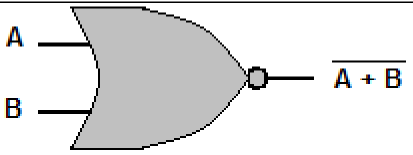
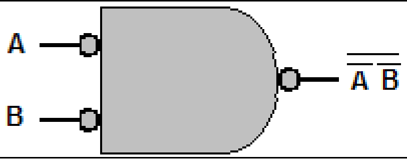
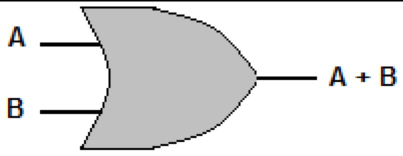
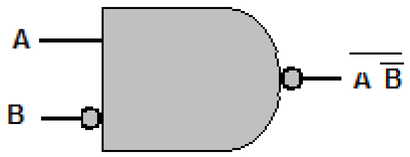
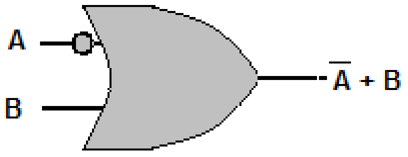
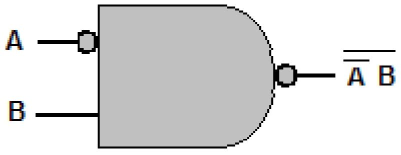
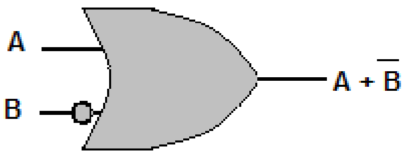
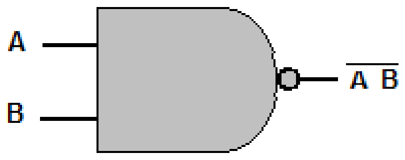
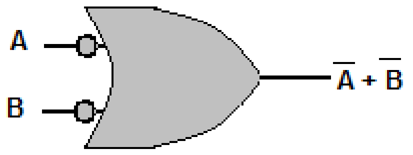
Figure 22

| A | B | f |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| **$f = \overline{A + \overline{B}}$** | | |

Table 4 illustrates AND, NOR, NAND, and OR gate combinations that produce the same output. We can see by the table that there is more than one way to achieve a desired output. Although the gates have only two inputs, the table can be extended to more than two inputs.

**Table 4**

**AND, NOR, NAND, and OR Gate Combinations**

| | AND Gates | OR Gates | A | B | f |
|---|---|---|---|---|---|
| 1 |  |  | 0<br>0<br>1<br>1 | 0<br>1<br>0<br>1 | 0<br>0<br>0<br>1 |
| 2 |  |  | 0<br>0<br>1<br>1 | 0<br>1<br>0<br>1 | 0<br>1<br>0<br>0 |
| 3 |  |  | 0<br>0<br>1<br>1 | 0<br>1<br>0<br>1 | 0<br>0<br>1<br>0 |
| 4 |  |  | 0<br>0<br>1<br>1 | 0<br>1<br>0<br>1 | 1<br>0<br>0<br>0 |
| | **NAND Gates** | **OR Gates** | | | |
| 5 |  |  | 0<br>0<br>1<br>1 | 0<br>1<br>0<br>1 | 0<br>1<br>1<br>1 |
| 6 |  |  | 0<br>0<br>1<br>1 | 0<br>1<br>0<br>1 | 1<br>1<br>0<br>1 |
| 7 |  |  | 0<br>0<br>1<br>1 | 0<br>1<br>0<br>1 | 1<br>0<br>1<br>1 |
| 8 |  |  | 0<br>0<br>1<br>1 | 0<br>1<br>0<br>1 | 1<br>1<br>1<br>0 |

# Chapter 4
# Logic Gates in Combination

When you look at logic circuit diagrams for digital equipment, you are not going to see just a single gate, but many combinations of gates. At first it may seem confusing and complex. If you interpret one gate at a time, you can work your way through any network. In this section, we will analyze several combinations of gates.

Figure 23 (view A) shows a simple combination of AND gates. The outputs of gates 1 and 2 are the inputs to gate 3. We already know that both inputs to an AND gate must be HIGH at the same time in order to produce a HIGH output.

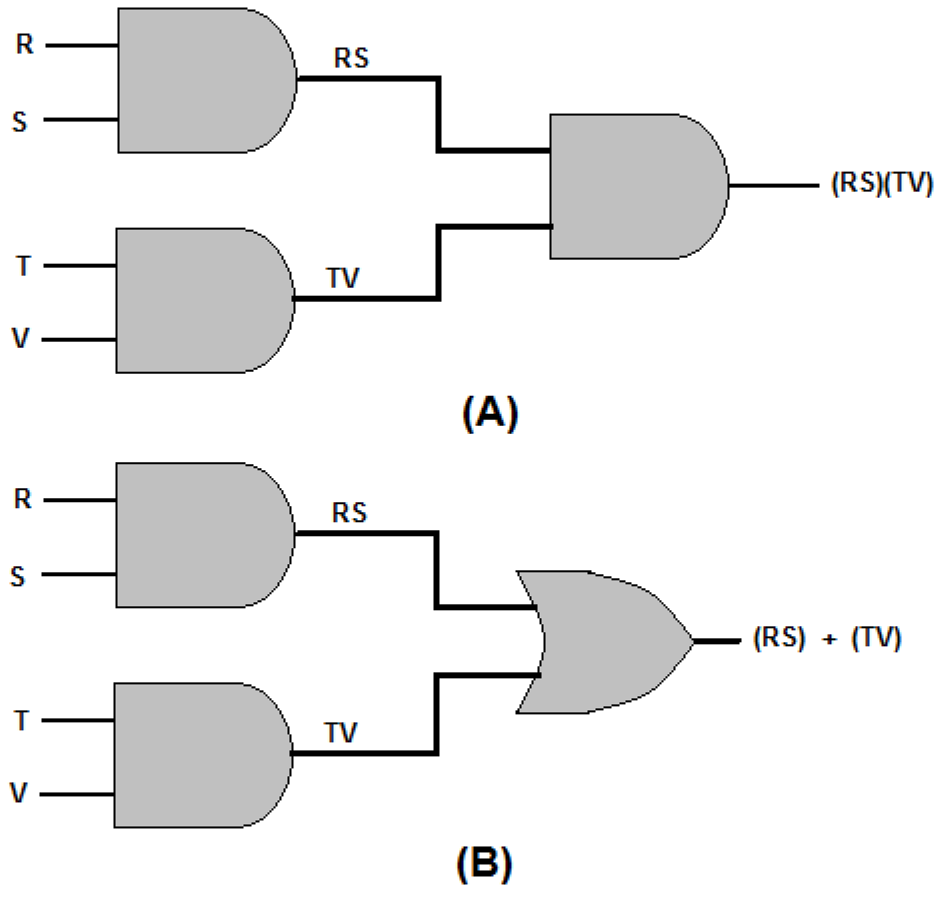


Figure 23

The output Boolean expression of gate 1 is RS, and the output expression of gate 2 is TV. These two output expressions become the inputs to gate 3. Remember, the output Boolean expression is the result of the inputs, in this case (RS)(TV); spoken "quantity R AND S AND quantity T AND V."

In view B we have changed gate 3 to an OR gate. The outputs of gates 1 and 2 remain the same but the output of gate 3 changes as we would expect. The output of gate 3 is now (RS)+(TV); spoken "quantity R AND S OR quantity T AND V."

In Figure 24 (view A), the outputs of two OR gates are being applied as the input to third OR gate. The output for gate 1 is R+S, and the output for gate 2 is T+V. With these inputs, the output expression of gate 3 is (R+S)+(T+V).

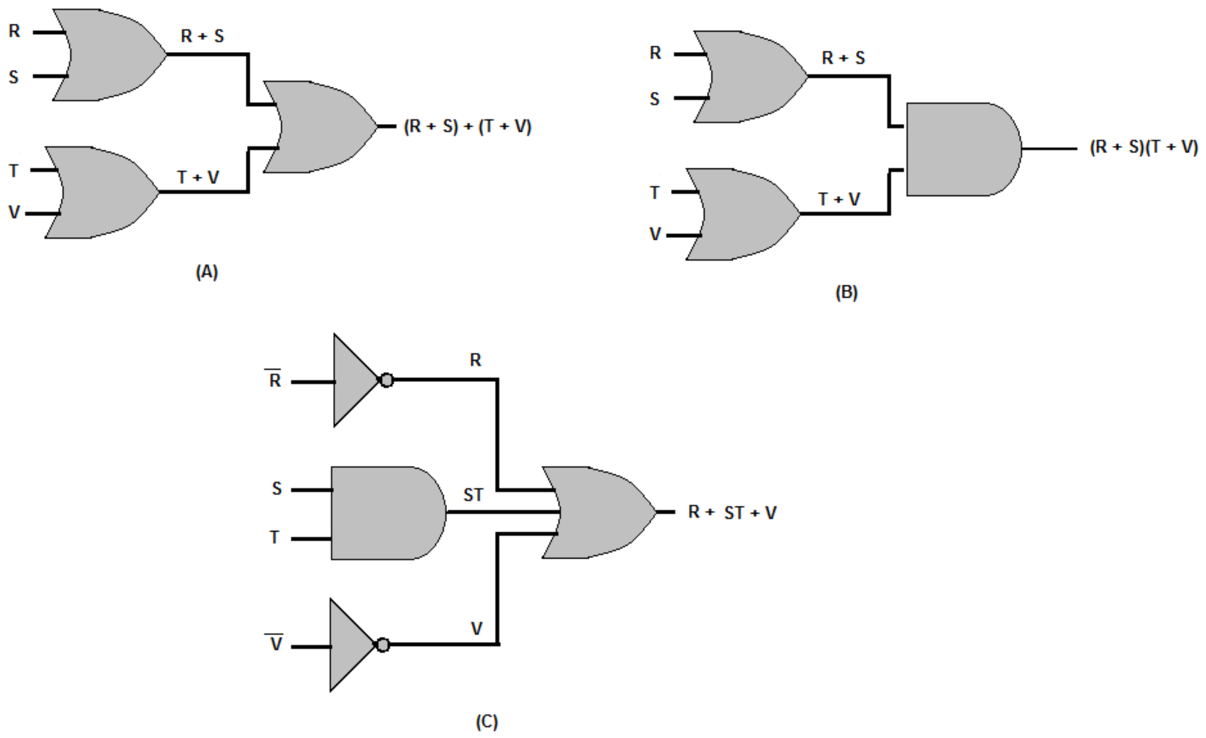


Figure 24

In view B, gate 3 has been changed to an AND gate. The outputs of gates 1 and 2 do not change, but the output expression of gate 3 does. In this case, the gate 3 output expression is (R + S)(T + V). This expression is spoken, "quantity R OR S AND quantity T OR V." The parentheses are used to separate the input terms and to indicate the AND function. Without the parentheses the output expression would read R + ST + V, which is representative of the circuit in view C. As we can see, this is not the same circuit as the one depicted in view B. It is very important that the Boolean expressions be written and spoken correctly.

The Truth Table for the output expression of gate 3 (view B) will help you better understand the output. When studying this Truth Table, notice that the only time f is HIGH (logic 1) is when either or both R and S AND either or both T and V are HIGH (logic 1).

**Truth Table**

| R | S | T | V | 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 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**$f = (R+S)(T+V)$**

Now let's determine the output expression for the NOR gate in Figure 25. First write the outputs of gates 1, 2, and 3:

| | | |
|---|---|---|
| $AB$ | $\overline{GH}$ | $X + Z$ |
| ⬆ | ⬆ | ⬆ |
| Gate 1 | Gate 2 | Gate 3 |

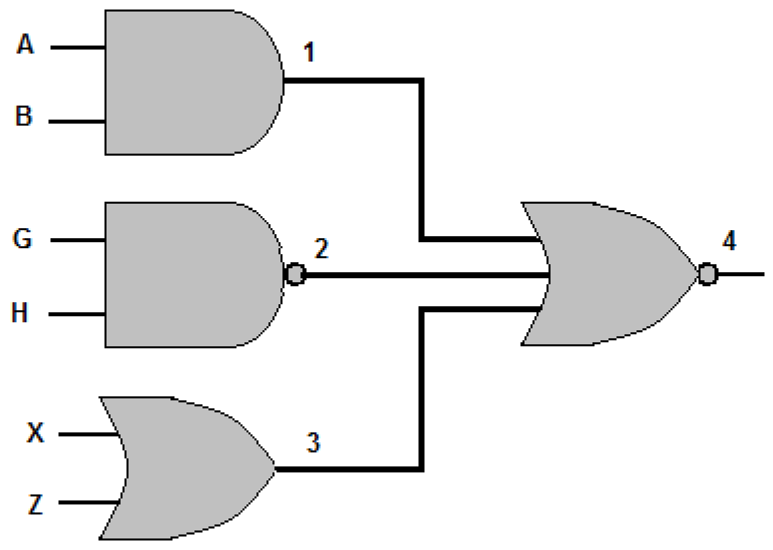


**Figure 25**

Since all three outputs are applied to gate 4, we may proceed as we would for any NOR gate. We separate each input to gate 4 with an OR sign (+) and then place a vinculum over the entire expression. The output expression of gate 4 is:

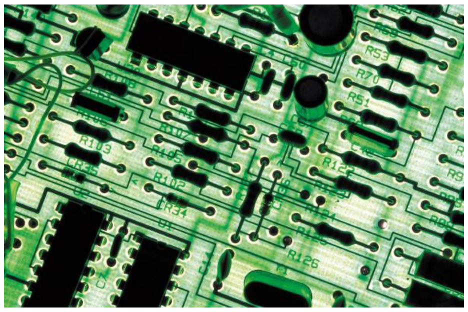
$$\overline{(AB) + (\overline{GH}) + (X + Z)}$$

When trying to determine the outputs of logic gates in combination, take them one gate at a time!

| **(AB)** | **(GH)** | **(X+Z)** | **f** |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 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 | 0 |
| $\mathbf{f = \overline{(AB) + (\overline{GH}) + (X+Z)}}$ | | | |

# Chapter 5
# Boolean Algebra

Boolean logic, or Boolean algebra as it is called today, was developed by an English mathematician, George Boole, in the 19th century. He based his concepts on the assumption that most quantities have two possible conditions - TRUE and FALSE. This is the same theory you were introduced to at the beginning of this course.



Throughout our discussions of fundamental logic gates, we have mentioned Boolean expressions. A Boolean expression is nothing more than a description of the input conditions necessary to get the desired output. These expressions are based on Boole's laws and theorems.

Boolean algebra is used primarily by design engineers. Using this system, they are able to arrange logic gates to accomplish desired tasks. Boolean algebra also enables the engineers to achieve the desired output by using the fewest number of logic gates. Since space, weight, and cost are important factors in the design of equipment, you would usually want to use as few parts as possible.

There are several Boolean laws and theorems that can be used to simplify the analysis of logic circuits. Each of the laws and theorems are explained below.

## Laws and Theorems

Each of the laws and theorems of Boolean algebra, along with a simple explanation, is listed below.

<u>LAW OF IDENTITY</u> - a term that is TRUE in one part of an expression will be TRUE in all parts of the expression (A = A or A = A).

<u>COMMUTATIVE LAW</u> - the order in which terms are written does not affect their value (AB = BA, A+B = B+A).

<u>ASSOCIATIVE LAW</u> - a simple equality statement A(BC) = ABC or A+(B+C) = A+B+C.

<u>IDEMPOTENT LAW</u> - a term ANDed with itself or ORed with itself is equal to that term (AA = A, A+A = A).

<u>DOUBLE NEGATIVE LAW</u> - a term that is inverted twice is equal to the term $\overline{\overline{A}} = A$.

<u>COMPLEMENTARY LAW</u> - a term ANDed with its complement equals 0, and a term O Red with its complement equals 1 ($A\overline{A} = 0$, $A+\overline{A} = 1$).

<u>LAW OF INTERSECTION</u> - a term ANDed with 1 equals that term and a term ANDed with 0 equals 0 (A*1 = A, A*0 = 0).

LAW OF UNION - a term ORed with 1 equals 1 and a term ORed with 0 equals that term ($A+1 = 1$, $A+0 = A$).

DeMORGAN'S THEOREM - this theorem consists of two parts: (1) $\overline{AB} = \overline{A} + \overline{B}$ and (2) $\overline{A + B} = \overline{A} * \overline{B}$ (Look at the fourth and eighth sets of gates in table 4).

DISTRIBUTIVE LAW - (1) a term (A) ANDed with an parenthetical expression (B+C) equals that term ANDed with each term within the parenthesis: $A* (B+C) = AB+AC$; (2) a term (A) ORed with a parenthetical expression ( $B \cdot C$) equals that term ORed with each term within the parenthesis: $A+(BC) = (A+B) * (A+C)$.

LAW OF ABSORPTION - this law is the result of the application of several other laws: $A*(A+B) = A$ or $A+(AB) = A$.

LAW OF COMMON IDENTITIES - the two statements $A*(\overline{A} +B) = AB$ and $A+\overline{A}\ B = A+B$ are based on the complementary law.

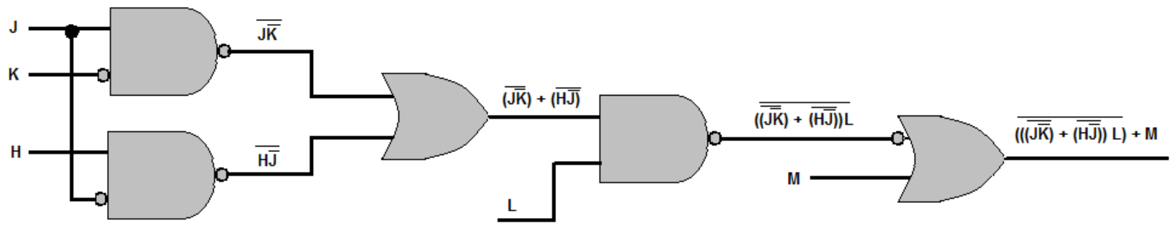
Table 5 on the next page summarizes these laws and theorems.

**Table 5**

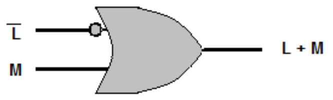
**Boolean Laws and Theorems**

| | | |
|---|---|---|
| 1. | Law of Identity | $A = A$<br>$\overline{A} = \overline{A}$ |
| 2. | Commutative Law | $A * B = B * A$<br>$A + B = B + A$ |
| 3. | Associative Law | $A * (B * C) = A * B * C$<br>$A + (B + C) = A + B + C$ |
| 4. | Idempotent Law | $A * A = A$<br>$A + A = A$ |
| 5. | Double Negative Law | $\overline{\overline{A}} = A$ |
| 6. | Complementary Law | $A * \overline{A} = 0$<br>$A + \overline{A} = 1$ |
| 7. | Law of Intersection | $A * 1 = A$<br>$A * 0 = 0$ |
| 8. | Law of Union | $A + 1 = 1$<br>$A + 0 = A$ |
| 9. | DeMorgan's Theorem | $\overline{AB} = \overline{A} + \overline{B}$<br>$\overline{A + B} = \overline{A}\,\overline{B}$ |
| 10. | Distributive Law | $A * (B + C) = (A * B) + (A * C)$<br>$A + (BC) = (A + B) * (A + C)$ |
| 11. | Law of Absorption | $A * (A + B) = A$<br>$A + (AB) = A$ |
| 12. | Law of Common Identities | $A * (\overline{A} + B) = AB$<br>$A + (\overline{A}\,B) = A + B$ |

Figure 26 (view A), shows a rather complex series of gates. Through proper application of Boolean algebra, the circuit can be simplified to the single OR gate shown in view B. Figure 27 shows the simplification process and the Boolean laws and theorem used to accomplish it.

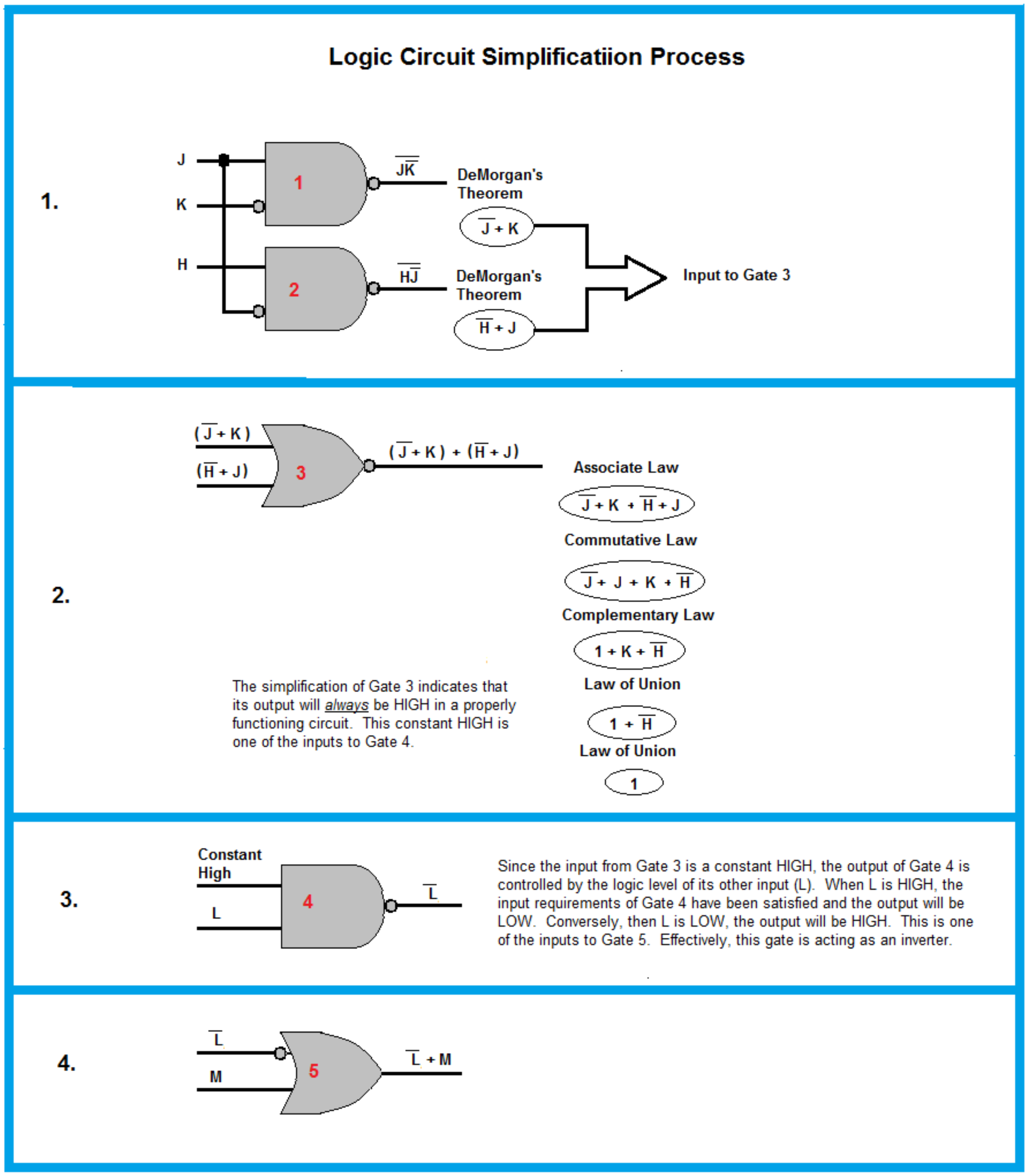

J
K
H
$\overline{JK}$
$\overline{HJ}$
$(\overline{JK}) + (\overline{HJ})$
L
$\overline{((\overline{JK}) + (\overline{HJ}))L}$
M
$\overline{((\overline{JK}) + (\overline{HJ})) L} + M$

(A)


$\overline{L}$
M
L + M

(B)

Figure 26



The simplification of Gate 3 indicates that its output will *always* be HIGH in a properly functioning circuit. This constant HIGH is one of the inputs to Gate 4.

Since the input from Gate 3 is a constant HIGH, the output of Gate 4 is controlled by the logic level of its other input (L). When L is HIGH, the input requirements of Gate 4 have been satisfied and the output will be LOW. Conversely, then L is LOW, the output will be HIGH. This is one of the inputs to Gate 5. Effectively, this gate is acting as an inverter.

Figure 27

## SUMMARY

This course has presented information on logic, fundamental logic gates, and Boolean laws and theorems. The information that follows summarizes the important points of this chapter.

*Logic* is the development of a logical conclusion based on known information.

Computers operate on the assumption that statements have two conditions - TRUE and FALSE.

*Positive logic* is defined as follows: if the signal that activates the circuit (the 1 state) has a voltage level that is more *positive* than the 0 state, then the logic polarity is considered to be *positive*.

*Negative logic* is defined as follows: if the signal that activates the circuit (the 1 state) has a voltage level that is more *negative* than the 0 state, then the logic polarity is considered to be *negative*.

In *digital logic* (positive or negative), the true condition of a statement is represented by the logic 1 state and the false condition is represented by the logic 0 state.

*Logic levels* high and low represent the voltage levels of the two logic states. Logic level HIGH represents the more positive voltage while logic level LOW represents the less positive (more negative) voltage. In positive logic, the high level corresponds to the true or 1 state and the low level corresponds to the false or 0 state. In negative logic, the high level corresponds to the false or 0 state and the low level corresponds to the true or 1 state.

A *Boolean expression* is a statement that represents the inputs and outputs of logic gates.

The *AND gate* requires all inputs to be high at the same time in order to produce a high output.

The *OR gate* requires one or both inputs to be high in order to produce a high output.

*INVERTER* (or NOT gate) is a logic gate used to complement the state of the input variable; that is, a 1 becomes a 0 or a 0 becomes a 1. It may be used on any input or output of any gate to obtain the desired result.

The *NAND gate* functions as an AND gate with an inverted output.

The *NOR gate* functions as an OR gate with an inverted output.

When deriving the output Boolean expression of a combination of gates, solve one gate at a time. Boolean algebra is used primarily for the design and simplification of circuits.

+++