

Programming Logic controller (PLC)

Programmable Logic Controller (PLC) is a microprocessor based system that uses programmable memory to store instructions and implement functions such as logic, sequencing, timing, counting and arithmetic in order to control machines and processes.

Advantage of PLC control systems

1. Flexible.
2. Faster response time.
3. Less and simpler wiring.
4. Solid-state-no moving parts.
5. Modular design-easy to repair and expand.
6. Ease of troubleshooting.
7. Less expensive.

Major Components of PLC

A PLC is essentially a microcomputer consisting of hardware and software. The major components are

1. Power Supply module.
2. Input module.
3. Central processing unit (CPU).
4. Output module.
5. Programming device.

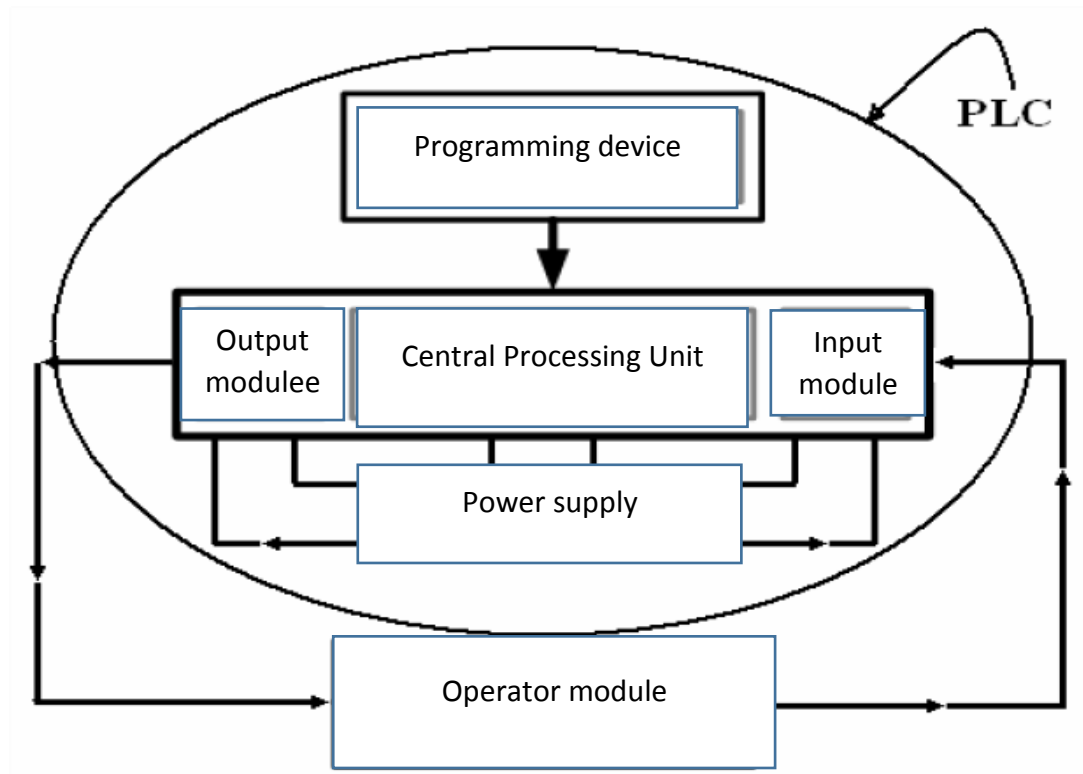


Figure (1): PLC Components

Factors of selecting switches

1. Contact type (e.g. 1-pole, 2-pole).
2. Current rating.
3. Voltage rating.
4. Method of operation.
5. Dielectric strength.
6. Electrical life and mechanical life.

Factors of selecting sensors switches

1. Type of sensor.
2. Design of sensor.
3. Sensing range.
4. Electrical data and connection.

5. Response speed.

Types of sensors

1. Proximity sensors.
2. Position sensors.
3. Inductive sensors.
4. Optical sensors.
5. Capacitive sensor.
6. Pressure sensors.

Number Systems

There are four systems of arithmetic which are often used in digital circuits. These systems are

1. decimal system.
2. binary system.
3. octal system.
4. hexadecimal system.

The Decimal Number System

The decimal number system has a base of 10 meaning that it contains ten unique symbols (or digits) .These are:1, 2, 3, 4, 5, 6, 7, 8, 9

$$2573 = 2 \times 10^3 + 5 \times 10^2 + 7 \times 10^1 + 3 \times 10^0$$

$$13432 = 1 \times 10^4 + 3 \times 10^3 + 4 \times 10^2 + 3 \times 10^1 + 2 \times 10^0$$

Again, the number 2573.469 can be written as

$$2573.469 = 2 \times 10^3 + 5 \times 10^2 + 7 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 6 \times 10^{-2} + 9 \times 10^{-3}$$

The Binary Number System

Its base or radix is two because it uses only digit is 0 and 1. All binary numbers consist of a string of 0 and 1.

Example1:- convert $(11001)_2$ to its equivalent decimal number.

Solution. The four steps involved in the conversion are as under

Step1. 1 1 0 0 1

Step2. 16 8 4 2 1

Step3. 16 8 0 0 1

Step4. 16+8+1=25 $(11001)_2 = (25)_{10}$

Or

$$11001 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (25)_{10}$$

Example2:- convert $(101.101)_2$ to its equivalent decimal number.

Solution.

$$(101.101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

$$(101.101)_2 = (5.625)_{10}$$

Decimal to binary convert

(a) Integers

As an example, let us convert $(25)_{10}$ into its binary equivalent

$$25 \div 2 = 12 \quad \text{remainder of } 1$$

$$12 \div 2 = 6 \quad \text{remainder of } 0$$

$$6 \div 2 = 3 \quad \text{remainder of } 0$$

$$3 \div 2 = 1 \quad \text{remainder of } 1$$

$$1 \div 2 = 0 \quad \text{remainder of } 1$$

$$(25)_{10} = (11001)_2$$

(b) Fractions

Example 4:- convert $(25.625)_{10}$ into its binary equivalent.

Solution:-

(a) Integers

$$25 \div 2 = 12 \quad \text{remainder of } 1$$

$$12 \div 2 = 6 \quad \text{remainder of } 0$$

$$6 \div 2 = 3 \quad \text{remainder of } 0$$

$$3 \div 2 = 1 \quad \text{remainder of } 1$$

$$1 \div 2 = 0 \quad \text{remainder of } 1$$

$$(25)_{10} = (11001)_2$$

(b) Fractions

$$0.625 \div 2 = 1.25 \quad \text{remainder of } 1$$

$$0.25 \div 2 = 0.5 \quad \text{remainder of } 0$$

$$0.5 \div 2 = 0 \quad \text{remainder of } 1$$

$$(0.625)_{10} = (0.101)_2$$

$$(25.625)_{10} = (11001.101)_2$$

Octal Number System

It has a base of 8 which means that it has eight distinct counting digits : 0, 1, 2, 3, 4, 5, 6, 7 these digits 0 through 7, have exactly the same physical meaning as in decimal system.

For example, decimal equivalent of octal 325 is

$$(325)_8 = 3 \times 8^2 + 2 \times 8^1 + 5 \times 8^0 = (234)_{10}$$

Similarly, decimal equivalent of octal 127.24 is

$$(127.24)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 2 \times 8^{-1} + 4 \times 8^{-2} = (87.3125)_{10}$$

Decimal to Octal Conversion

Let us see how we can convert $(175)_{10}$ into its octal equivalent.

$$\begin{array}{ll} 175 \div 8 = 21 & \text{with 7 remainder} \\ 21 \div 8 = 2 & \text{with 5 remainder} \\ 2 \div 8 = 0 & \text{with 2 remainder} \end{array}$$



Taking the remainders in the reverse order, we get $(257)_8$

$$(175)_{10} = (257)_8$$

Let us now take decimal fraction $(0.15)_{10}$. Its octal equivalent can be found as under

$$\begin{array}{ll} 0.15 \times 8 = 1.20 & \text{with 1 remainder} \\ 0.20 \times 8 = 1.60 & \text{with 1 remainder} \\ 0.60 \times 8 = 4.80 & \text{with 4 remainder} \end{array}$$



$$(0.15)_{10} = (0.114)_8$$

Hexadecimal Number System

It has a base of 16. Hence, it uses sixteen distinct counting digits 0 through 9 and A through F as detailed below :

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Decimal to hexadecimal conversion

As an example let us convert decimal 1983 into hexadecimal

$$\begin{array}{ll} 1983 \div 16 = 123 & \text{with remainder 15} \quad \text{F} \\ 123 \div 16 = 7 & \text{with remainder 11} \quad \text{B} \\ 7 \div 16 = 0 & \text{with remainder} \quad 7 \end{array}$$

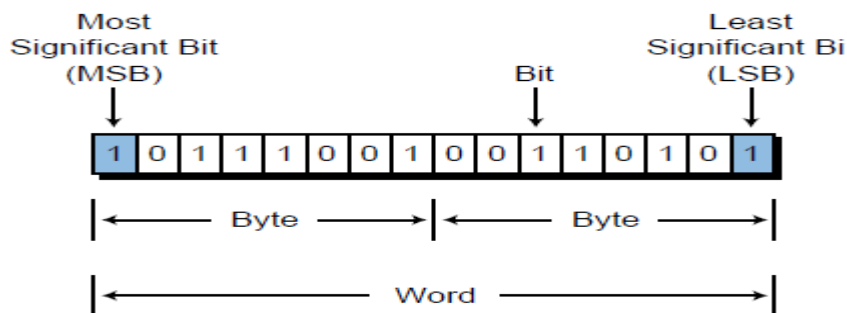


Hence, $(1983)_{10} = (7BF)_{16}$

Hexadecimal to decimal conversion

As an example, let us convert $(F6D9)_{16}$ to decimal

$$\begin{aligned}(F6D9)_{16} &= F \times 16^3 + 6 \times 16^2 + D \times 16^1 + 9 \times 16^0 \\ &= 15 \times 16^3 + 6 \times 16^2 + 13 \times 16^1 + 9 \times 16^0 = (63193)_{10}\end{aligned}$$



Byte =8 bits

Word=2 Bytes =16 bits

PLC PRODUCT APPLICATION RANGES

Figure 1-1 graphically illustrates programmable controller product ranges. This chart is not definitive, but for practical purposes, it is valid. The PLC market can be segmented into five groups:

1. Micro PLCs
2. Small PLCs
3. Medium PLCs
4. Large PLCs

5. Very large PLCs

ONE'S AND TWO'S COMPLEMENT

ONE'S COMPLEMENT

The number is positive if the sign bit is 0 and negative if the sign bit is 1. Using the one's complement method, +23 decimal is represented in binary, as shown here with the sign bit (0) indicated in bold:

$$\mathbf{0} \ 10111_2$$

The negative representation of binary 10111 is obtained by placing a 1 in the most significant bit position and inverting each bit in the number (changing 1s to 0s and 0s to 1s). So, the one's complement of binary 10111 is:

$$\mathbf{1} \ 01000_2$$

If a negative number is given in binary, its one's complement is obtained in the same fashion.

$$-15_{10} = \mathbf{1} \ 0000_2$$

$$+15_{10} = \mathbf{0} \ 1111_2$$

TWO'S COMPLEMENT

in the two's complement, each bit, from right to left, is inverted only after the first 1 is detected. Let's use the number +22 decimal as an Example:

$$+22_{10} = \mathbf{0} \ 10110_2$$

Its two's complement would be:

$$-22_{10} = \mathbf{1\ 01010}_2$$

Note that in the negative representation of the number 22, starting from the right, the first digit is a 0, so it is not inverted; the second digit is a 1, so all digits after this one are inverted.

$$-14_{10} = \mathbf{1\ 10010}_2$$

$$+14_{10} = \mathbf{0\ 01110}_2$$

$$+17_{10} = \mathbf{0\ 10001}_2$$

$$-17_{10} = \mathbf{1\ 01111}_2$$

$$+7_{10} = \mathbf{0\ 00111}_2$$

$$-7_{10} = \mathbf{1\ 11001}_2$$

$$+1_{10} = \mathbf{0\ 00001}_2$$

$$-1_{10} = \mathbf{1\ 11111}_2$$

The two's complement is the most common arithmetic method used in computers, as well as programmable controllers.

BINARY CODES

Several codes for representing numbers, symbols, and letters are standard throughout the industry. Among the most common are the following:

1. **BCD**
2. **Gray**
3. **ASCII**

Binary Coded Decimal (BCD)

Table 1-2 illustrates the relationship between the BCD code and the binary and decimal number systems.

Decimal	Binary	BCD
0	0	0000
1	1	0001
2	10	0010
3	11	0011
4	100	0100
5	101	0101
6	110	0110
7	111	0111
8	1000	1000
9	1001	1001

The BCD representation of a decimal number is obtained by replacing each decimal digit with its BCD equivalent. The BCD representation of decimal 7493 is shown here as an example:

BCD	→	0111	0100	1001	0011
Decimal	→	7	4	9	3

Example:- write the decimal number 369 in BCD code

Solution .

3=0011, 6=0110, 9=1001,

$369_{10} = 001101101001_{BCD}$

Example: - Find the equivalent decimal value for the BCD code number 0001010001110101.

Solution:-

0001=1, 0100=4, 0111=7, 0101=5,

Hence $0001010001110101_{BCD} = 1475_{10}$

Gray code

Table 1-3 shows this code with its binary and decimal equivalents for comparison.

Gray Code	Binary	Decimal
0000	0	0
0001	1	1
0011	10	2
0010	11	3
0110	100	4
0111	101	5
0101	110	6
0100	111	7
1100	1000	8
1101	1001	9
1111	1010	10
1110	1011	11
1010	1100	12
1011	1101	13
1001	1110	14
1000	1111	15

Table 1-3. Gray code, binary, and decimal counting.

ASCII

Alphanumeric codes (which use a combination of letters, symbols, and decimal numbers) These alphanumeric characters—26 letters (uppercase), 10 numerals (0-9), plus mathematical and punctuation symbols— can be represented using a 6-bit code (i.e., $2^6 = 64$ possible characters). The most common code for alphanumeric representation is **ASCII** (the American Standard Code for Information Interchange).

Figure 1-2 a shows the binary ASCII code representation of the letter Z (132).

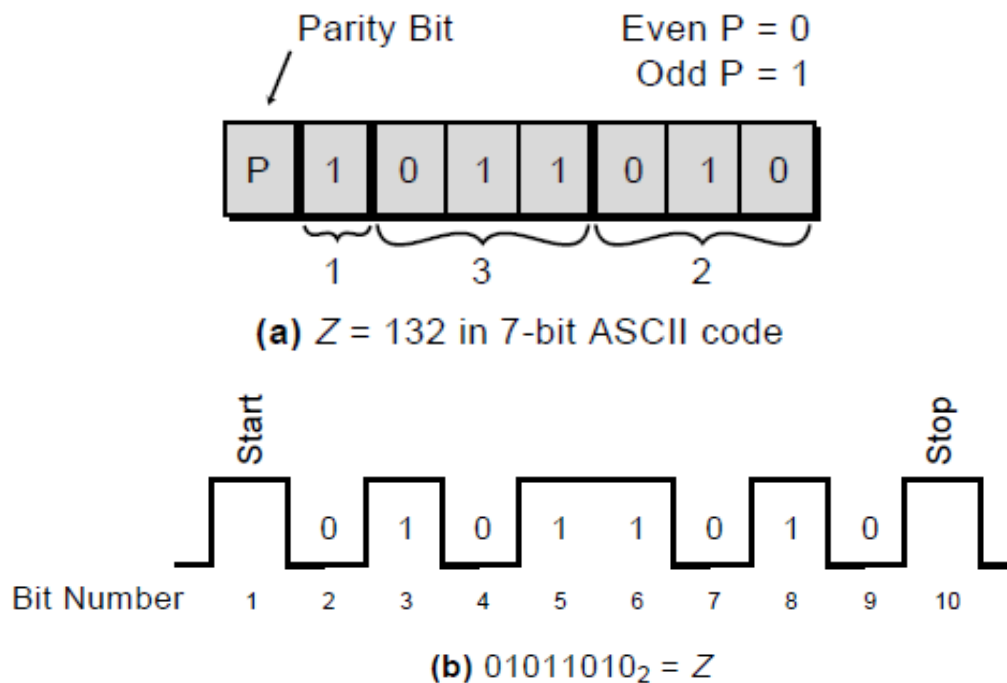


Figure 1-2. (a) ASCII representation of the character Z and (b) the ASCII transmission of the character Z.

BINARY CONCEPTS

In digital systems, these two states are actually represented by two distinct voltage levels, +V and 0V, as shown in Table 1-4.

1. Positive logic

1 (+V)	0 (0V)	Example
Operating	Not operating	Limit switch
Ringing	Not ringing	Bell
On	Off	Light bulb
Blowing	Silent	Horn
Running	Stopped	Motor
Engaged	Disengaged	Clutch
Closed	Open	Valve

Table 1-4. Binary concept using positive logic.

2. Negative logic

As illustrated in Table 1-5, uses 0 to represent the more positive voltage level, or the occurrence of the event. Consequently, 1 represents the nonoccurrence of the event, or the less positive voltage level.

1 (+V)	0 (0V)	Example
Not operating	Operating	Limit switch
Not ringing	Ringing	Bell
Off	On	Light bulb
Silent	Blowing	Horn
Stopped	Running	Motor
Disengaged	Engaged	Clutch
Open	Closed	Valve

Table 1-5. Binary concept using negative logic.

LOGIC FUNCTIONS

Operations performed by digital equipment, such as programmable controllers, are based on three fundamental logic functions—AND, OR, and NOT. These functions combine binary variables to form statements. Each function has a rule that determines the statement, outcome (TRUE or FALSE) and a symbol that represents it.

THE AND FUNCTION

Figure 1-3 shows a symbol called an AND **gate**, which is used to graphically represent the **AND** function. The AND output is TRUE (1) only if all inputs are TRUE (1).

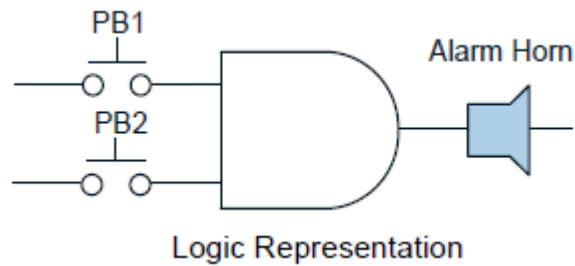


AND Truth Table		
Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

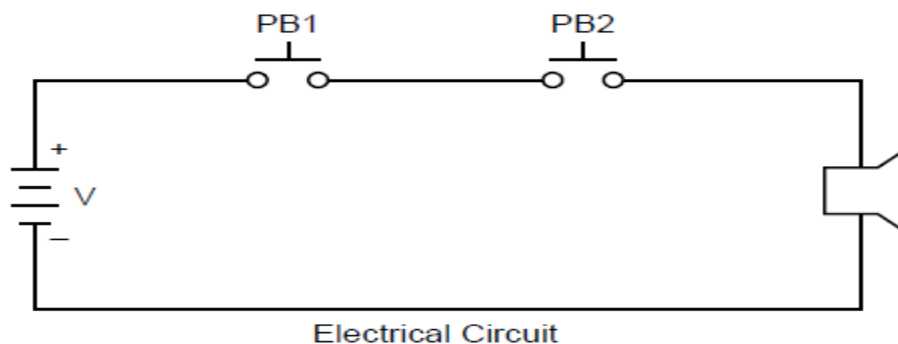
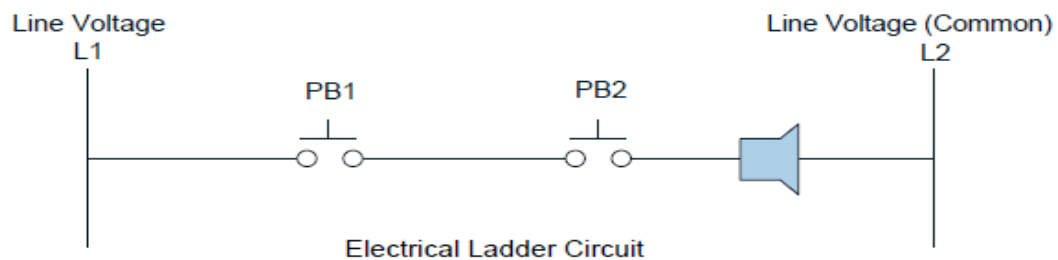
EXAMPLE 3-1

Show the logic gate, truth table, and circuit representations for an alarm horn that will sound if its two inputs, push buttons PB1 and PB2, are 1 (ON or depressed) at the same time.

SOLUTION



PB1	PB2	Alarm Horn
Not pushed (0)	Not pushed (0)	Silent (0)
Not pushed (0)	Pushed (1)	Silent (0)
Pushed (1)	Not pushed (0)	Silent (0)
Pushed (1)	Pushed (1)	Sounding (1)



THE OR FUNCTION

Figure 1-4 shows the OR gate symbol used to graphically represent the OR function. The OR output is TRUE (1) if one or more inputs are TRUE (1).

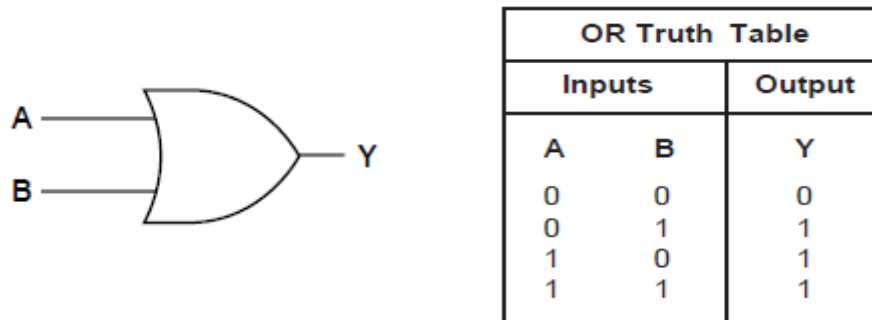
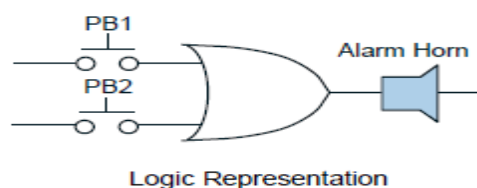


Figure 1-4. Two-input OR gate and its truth table.

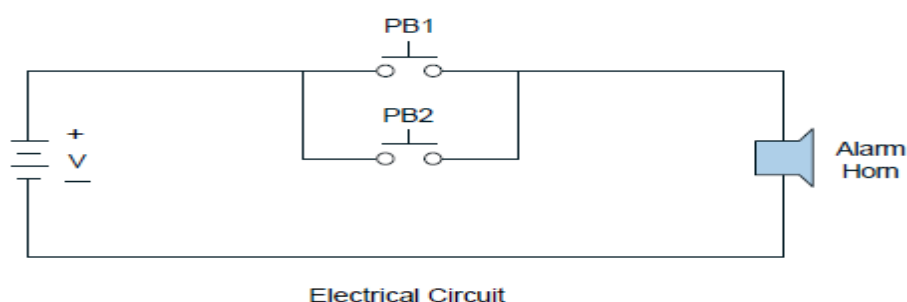
EXAMPLE 3-2

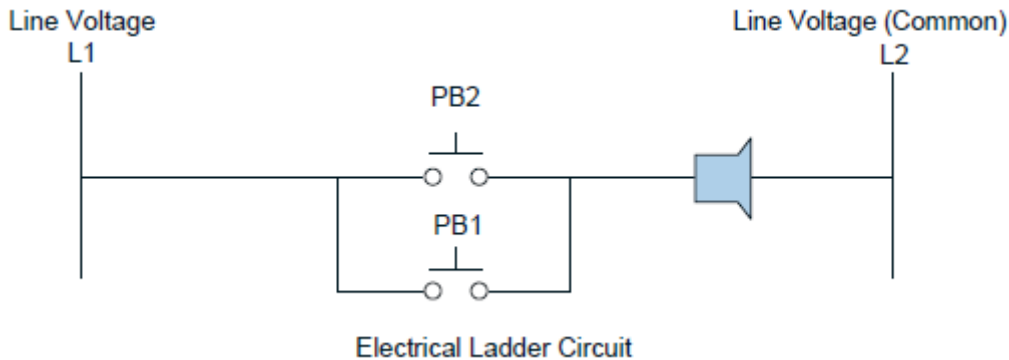
Show the logic gate, truth table, and circuit representations for an Alarm horn that will sound if either of its inputs, push button PB1 or PB2, is 1 (ON or depressed).

SOLUTION:-



PB1	PB2	Alarm Horn
Not pushed (0)	Not pushed (0)	Silent (0)
Not pushed (0)	Pushed (1)	Sounding (1)
Pushed (1)	Not pushed (0)	Sounding (1)
Pushed (1)	Pushed (1)	Sounding (1)



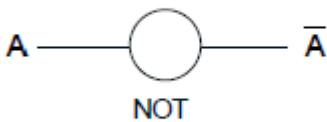


THE NOT FUNCTION

Figure 1-5 illustrates the NOT symbol, which is used to graphically represent the NOT function. The NOT output is TRUE (1) if the input is FALSE (0). Conversely, if the output is FALSE (0), the input is TRUE (1). The result of the NOT operation is always the inverse of the input; therefore, it is sometimes called an inverter.



Figure 1-5. Symbol for the NOT function.



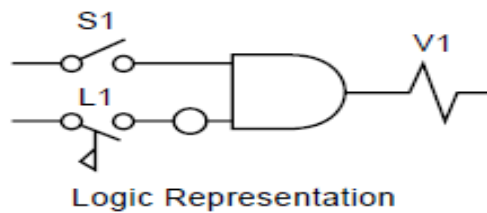
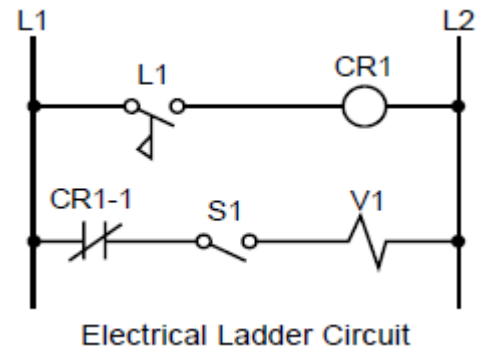
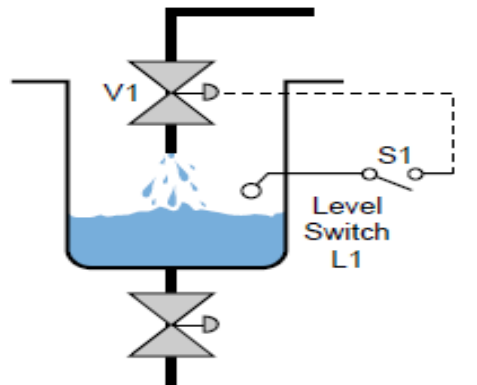
NOT Truth Table	
Input	Output
A	\bar{A}
0	1
1	0

Figure 1-6. NOT gate and its truth table.

EXAMPLE 3-3

Show the logic gate, truth table, and circuit representation for a solenoid valve (V1) that will be open (ON) if selector switch S1 is ON and if level switch L1 is NOT ON (liquid has not reached level).

SOLUTION:-



S1	L1 ($\overline{L1}$)		V1
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

Truth Table

EXAMPLE 3-4(Homework)

Show the logic gate, truth table, and circuit representation for an alarm horn that will sound if push button PB1 is 1 (ON or depressed) and PB2 is NOT 0 (not depressed).

3-3 PRINCIPLES OF BOOLEAN ALGEBRA AND LOGIC

Figure 1-8 summarizes the basic **Boolean operators** as they relate to the basic digital logic functions AND, OR, and NOT. These operators use capital letters to represent the wire label of an input signal, a multiplication sign (\bullet) to represent the AND operation, and an addition sign (+) to represent the OR operation. A bar over a letter represents the NOT operation.




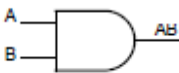
Logical Symbol	Logical Statement	Boolean Equation
	Y is 1 if A AND B are 1	$Y = A \bullet B$ or $Y = AB$
	Y is 1 if A OR B is 1	$Y = A + B$
	Y is 1 if A is 0 Y is 0 if A is 1	$Y = \bar{A}$

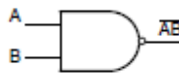
Figure 3-8. Boolean algebra as related to the AND, OR, and NOT functions.

1. **Basic Gates.** Basic logic gates implement simple logic functions. Each logic function is expressed in terms of a truth table and its Boolean expression.




A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

AND




A	B	\overline{AB}
0	0	1
0	1	1
1	0	1
1	1	0

NAND



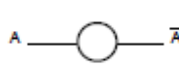
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

OR



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




NOR



A	\bar{A}
0	1
1	0

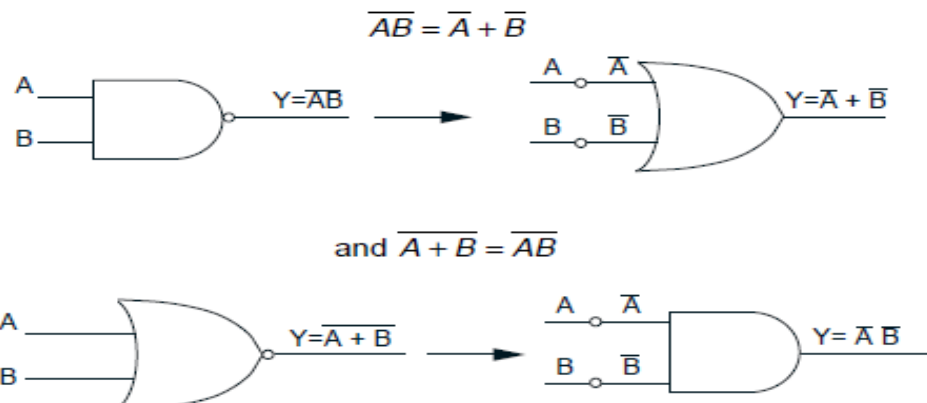
NOT

- 2. Combined Gates.** Any combination of control functions can be expressed in Boolean terms using three simple operators: (\bullet), ($+$), and ($-$).

Logical Symbol	Logical Statement	Boolean Equation
	Y is 1 if A AND B are 1	$Y = A \bullet B$ or $Y = AB$
	Y is 1 if A OR B is 1	$Y = A + B$
	Y is 1 if A is 0 Y is 0 if A is 1	$Y = \bar{A}$

- 4. Application of De Morgan's Laws.** De Morgan's Laws are frequently used to simplify inverted logic expressions or to simply convert an expression into a usable form.

According to De Morgan's Laws:



ADDRESSES USED IN PLCS

Figure 1-9 illustrates a simple electrical ladder circuit and its equivalent PLC implementation.

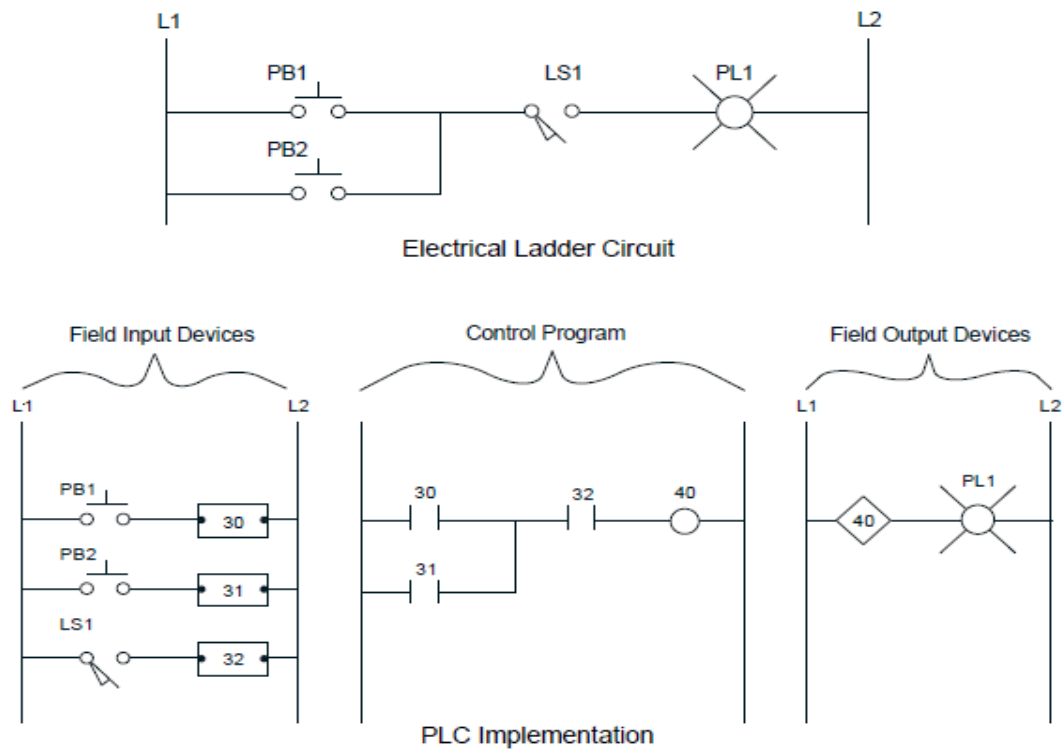


Figure 1-9. Electrical ladder circuit and its equivalent PLC implementation.

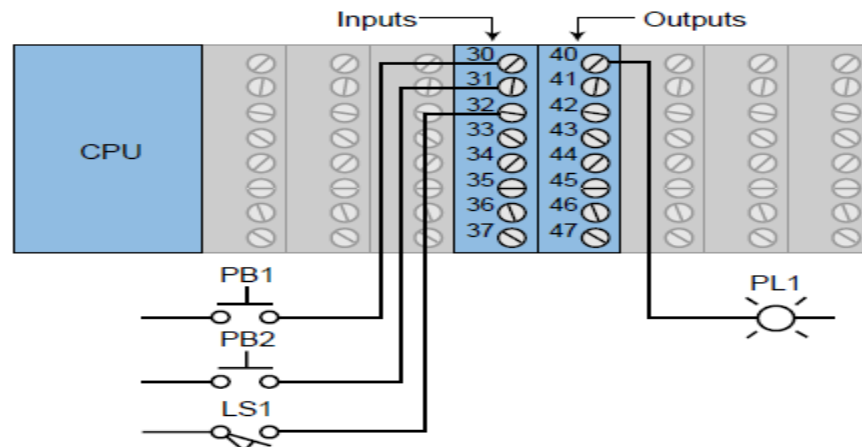
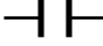



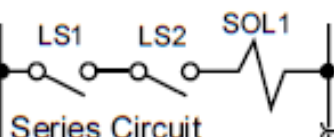
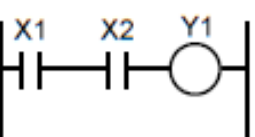
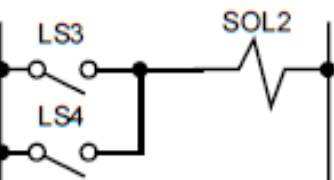
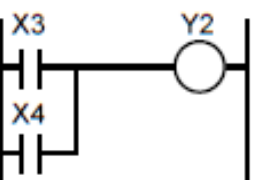
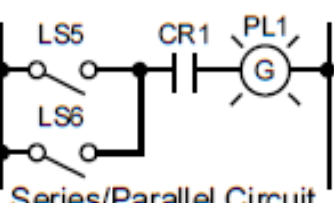
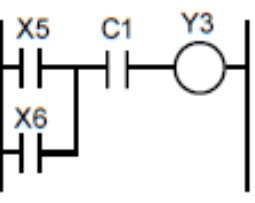
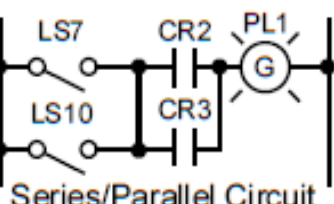
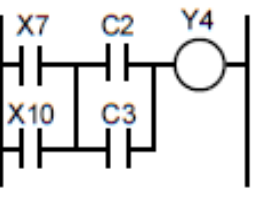
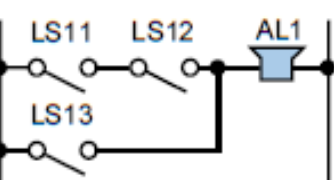
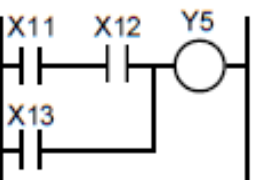
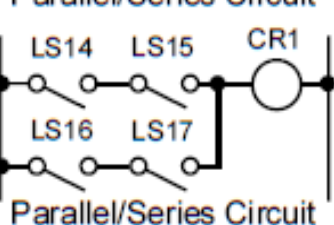
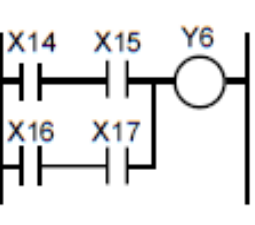
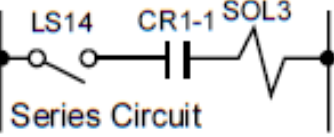
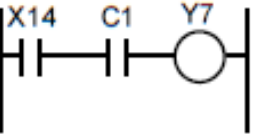
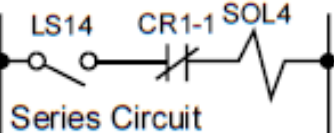
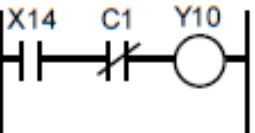


Figure 1-10. Field devices from Figure 1-9 connected to I/O module.

CONTACT SYMBOLS USED IN PLCs

The symbols in Table 3-5 are used to translate relay control logic to contact symbolic logic. These symbols are also the basic instruction set for the ladder diagram, excluding timer/counter instructions.

Symbol	Definition and Symbol Interpretation
	Normally open contact
	Normally closed contact.
	Output
	NOT output

i)	Relay Ladder Diagram	Contact or Ladder Diagram	Boolean Equation	Boolean Statements
(a)	 <p>Series Circuit</p>		$Y1 = X1 \cdot X2$	STR X1 AND X2 OUT Y1
(b)	 <p>Parallel Circuit</p>		$Y2 = X3 + X4$	STR X3 OR X4 OUT Y2
(c)	 <p>Series/Parallel Circuit</p>		$Y3 = (X5 + X6) \cdot C1$	STR X5 OR X6 AND C1 OUT Y3
(d)	 <p>Series/Parallel Circuit</p>		$Y4 = (X7 + X10) \cdot (C2 + C3)$	STR X7 AND X10 STR C2 OR C3 OUT Y4
(e)	 <p>Parallel/Series Circuit</p>		$Y5 = (X11 \cdot X12) + X13$	STR X11 AND X12 OR X13 OUT Y5
(f)	 <p>Parallel/Series Circuit</p>		$Y6 = (X14 \cdot X15) + (X16 \cdot X17)$	STR X14 AND X15 OR X16 AND X17 OUT Y6
(g)	 <p>Series Circuit</p>		$Y7 = X14 \cdot C1$	STR X14 AND C1 OUT Y7
(h)	 <p>Series Circuit</p>		$Y10 = X14 \cdot \overline{C1}$	STR X14 AND NOT C1 OUT Y10

Major Components of CPU(central processing unit)

The CPU forms what can be considered to be the “brain” of the system. The three components of the CPU are:

1. the processor
2. the memory system
3. the power supply

Figure 1-11 illustrates a simplified block diagram of a CPU.

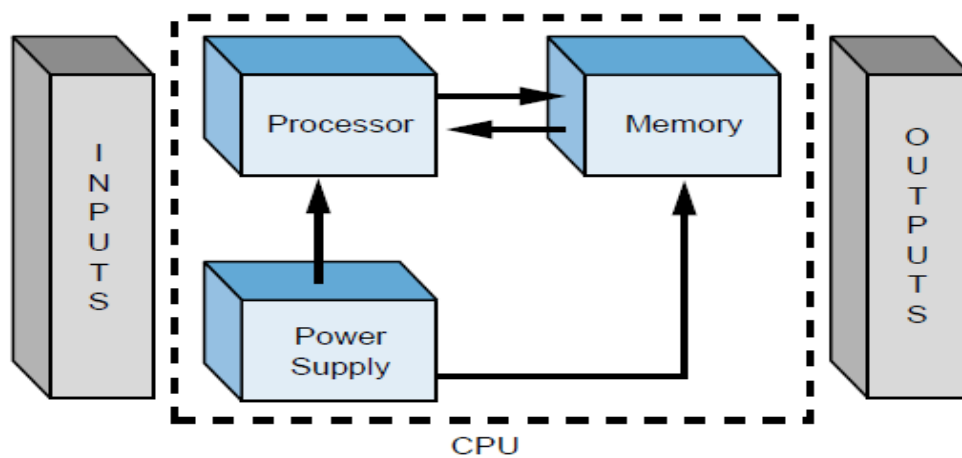


Figure 1-11. CPU block diagram.

Figure 1-12 illustrates the functional interaction between a PLC's basic components.

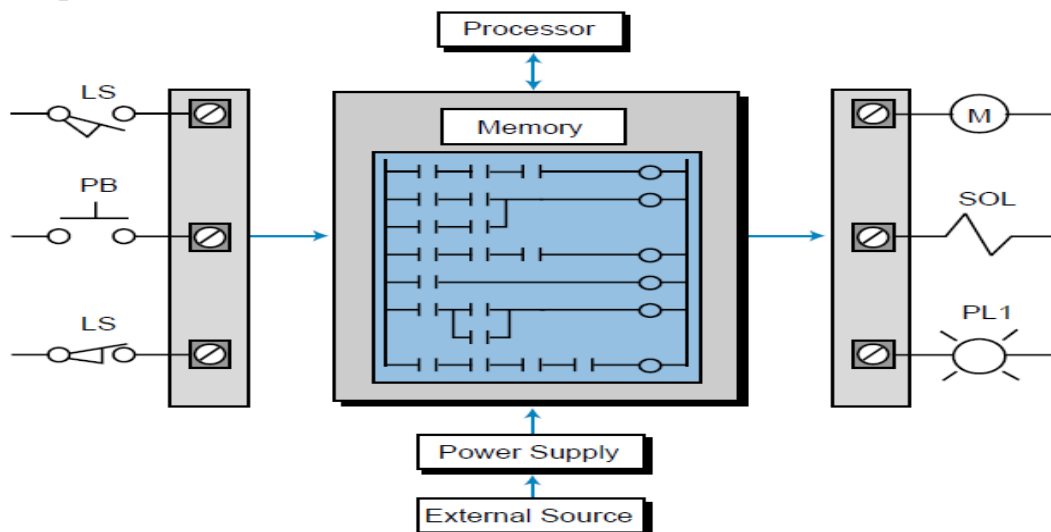


Figure 1-12. Functional interaction of a PLC system.

PROCESSORS

Very small **microprocessors** (or micros)—integrated circuits with tremendous computing and control capability—provide the intelligence of today's programmable controllers. They perform mathematical operations, data handling, and diagnostic routines that were not possible with relays or their predecessor, the hardwired logic processor.

The microprocessors used in PLCs are categorized according to their word size, or the number of bits that they use simultaneously to perform operations. Standard word lengths are 8, 16, and 32 bits. This word length affects the speed at which the processor performs most operations. For example, a 32-bit microprocessor can manipulate data faster than a 16-bit micro, since it manipulates twice as much data in one operation.

PROCESSOR SCAN

Figure 1-13 shows a graphic representation of the scan. The scanning process is repeated over and over in the same fashion, making the operation sequential from top to bottom. Sometimes, for the sake of simplicity, PLC manufacturers call the solving of the control program the **program scan** and the reading of inputs and updating of outputs the **I/O update scan**. Nevertheless, the total system scan includes both. The internal processor signal, which indicates that the program scan has ended, is called the *end-of-scan* (EOS) signal.

The time it takes to implement a scan is called the **scan time**. The scan time is the total time the PLC takes to complete the program and I/O updates scans.

The program scan time generally depends on two factors:

1. The amount of memory taken by the control program.
2. The type of instructions used in the program.

The time required to make a single scan can vary from a few tenths of a millisecond to 50 milliseconds.

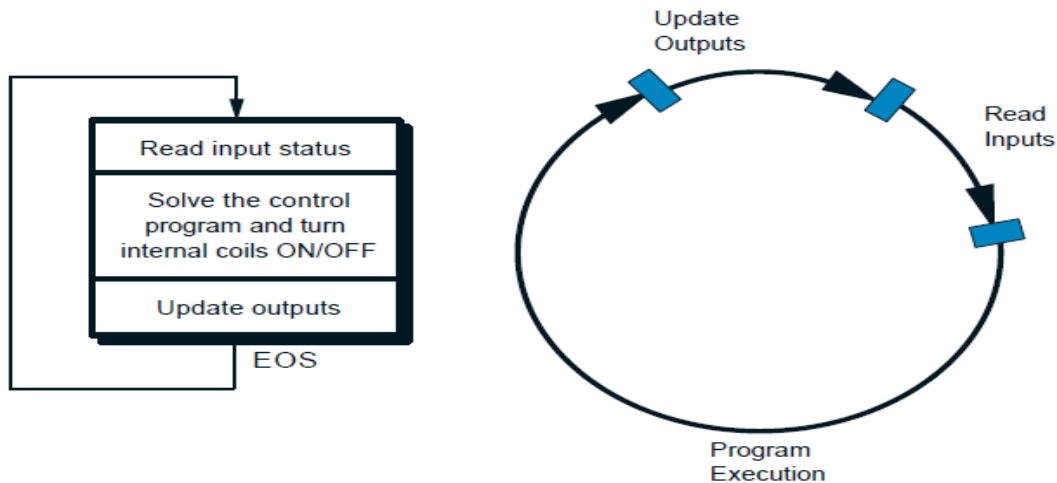


Figure 1-13. PLC total scan representation.

THE MEMORY SYSTEM

MEMORY SECTIONS

The total memory system in a PLC is actually composed of two different memories .

1. the executive memory
2. the application memory

The following discussion describes six types of memory and how their characteristics affect the manner in which programmed instructions are retained or altered within a programmable controller.

1. READ-ONLY MEMORY
2. RANDOM-ACCESS MEMORY
3. PROGRAMMABLE READ-ONLY MEMORY
4. ERASABLE PROGRAMMABLE READ-ONLY MEMORY

5. ELECTRICALLY ALTERABLE READ-ONLY MEMORY
6. ELECTRICALLY ERASABLE PROGRAMMABLE

MEMORY STRUCTURE AND CAPACITY

Bit : each cell is called a bit. A bit, then, is the smallest structural unit of memory.

Byte : a byte is the smallest group of bits that can be handled by the processor at one time. byte size is normally eight bits (byte=8 bit).

Word: a word is also a fixed group of bits that varies according to the controller; however, words are usually one byte or more in length.

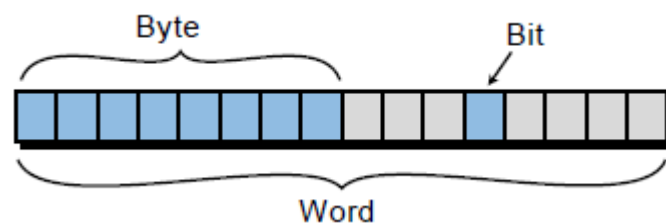


Figure 1-14. Units of PLC memory: bits, bytes, and words.

EXAMPLE:-

Referencing Figure 1-15, what happens to internal 2301 (word 23, bit 01) when the limit switch connected to input terminal 10 closes?

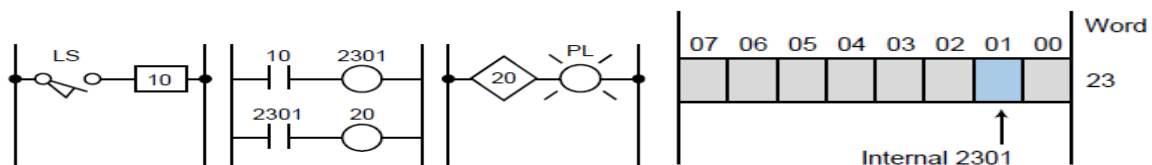


Figure 1-15. Open limit switch connected to an internal output.

SOLUTION:-

When LS closes (see Figure 1-16), contact 10 will close, turning internal output 2301 ON (a 1 in bit 01 of word 23). This will close

contact 2301 and turn real output 20 ON, causing the light PL to turn ON at the end of the scan.

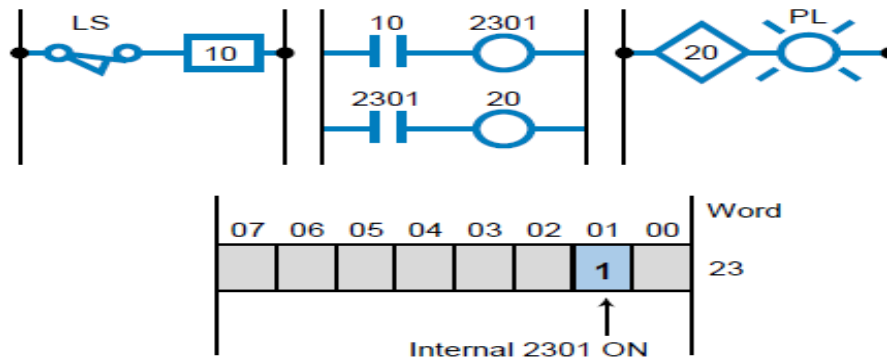


Figure 1-16. Closed limit switch connected to an internal output.

AC/DC INPUTS

Figure 1-17 shows a typical AC/DC input circuit. An AC/DC input circuit has two primary parts:

- The power section
- The logic section

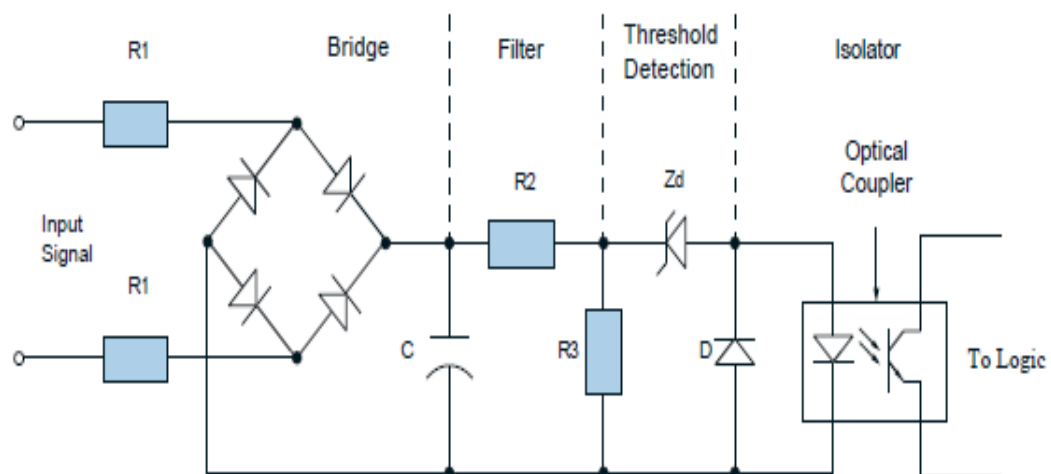


Figure 1-17. Typical AC/DC input circuit.

AC OUTPUTS

Figure 1-18 illustrates a typical AC output circuit.

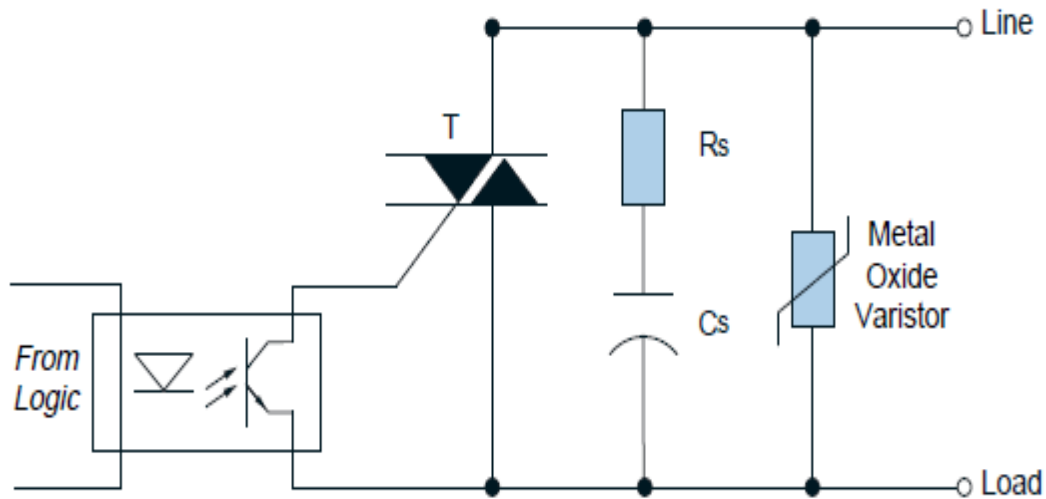


Figure 1-18. Typical AC output circuit.

TYPES OF PLC LANGUAGES

The three types of programming languages used in PLCs are:

1. ladder
2. Boolean
3. Grafcet (Graphe Fonctionnel)

EXAMPLE:-

Solve the logic rung shown in Figure 1-19 so that no reverse power flow condition exists. The reverse condition is not part of the required logic for the output to be energized.

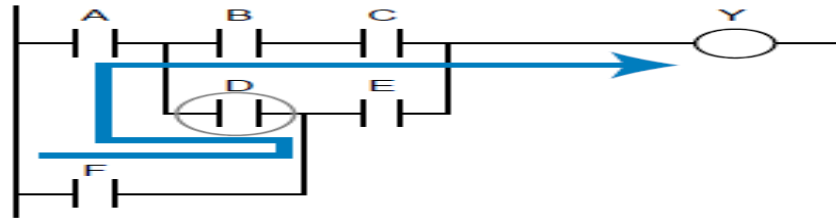


Figure 1-19 Reverse power flow at contact D.

SOLUTION:-

The forward power flow of the logic determines output Y. Let's implement it using logic concepts. The output Y is defined, using forward paths only, as:

$$Y = \overbrace{(A \bullet B \bullet C)}^{\text{1st line}} + \overbrace{(A \bullet D \bullet E)}^{\text{2nd line}} + \overbrace{(F \bullet E)}^{\text{3rd line}}$$

Which can be minimized, using Boolean algebra's distributed rule

$$Y = A \bullet (B \bullet C + D \bullet E) + (F \bullet E)$$

Figure 1-20 shows the implementation of this logic gate, while Figure 1-21 gives the ladder-equivalent solution.

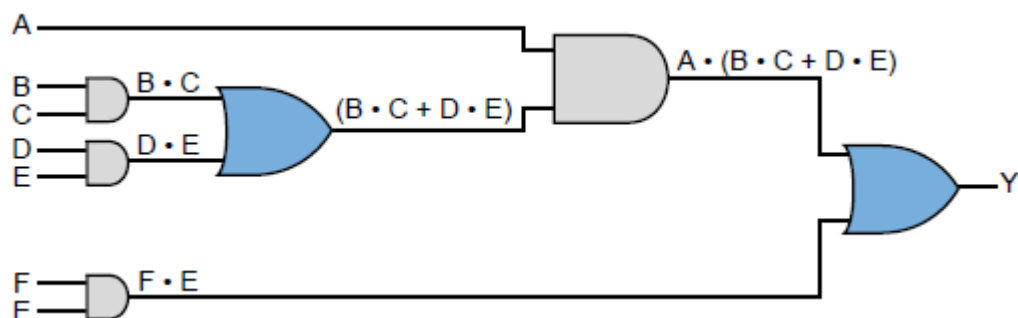


Figure 1-20. Logic solution for Example.

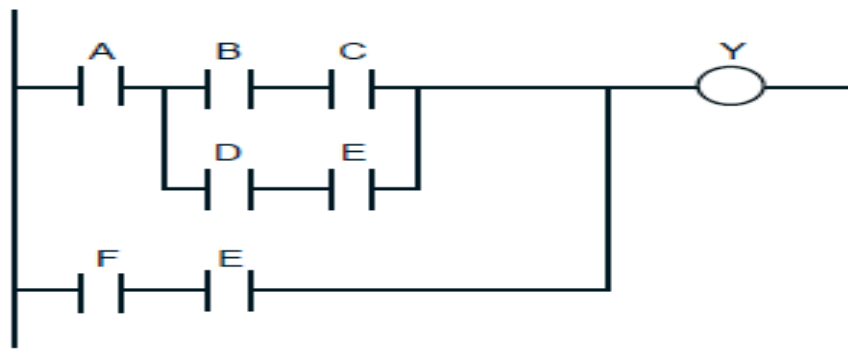


Figure 1-21. Ladder diagram implementation for Example