# UNIT-2

## GATES AND BOOLEAN ALGREBA

- Gates.
- Boolean algebra, Truth Tables.
- Preparing a Truth Table using circuit.
- Preparing circuit for given table (SOP & POS)
- De Morgan's Theorems, Use of De Morgan's theorems to implement and Gate Minimizations.

## GATES

***A logic gate*** is an electronic circuit which implements logical operation produces a typical output signal depending on its input signal. The output signal of a gate is a simple Boolean operation of its input signal.

Binary information is represented in digital computers using electrical signals.
These signals can be represented by voltage to specify one of two possible states.

The manipulation of binary information in a computer is done using logic circuits called ***gates.***

Electrical signals such as voltage throughout the computer in either one of two recognizable states. Binary information is represented in digital computers by physical quantities called ***signals***. Gates are the basic logic elements that produce signals of binary 1(high) or 0(law).

We can represent any Boolean function in the form of gates. The manipulation of binary information is done by logic circuit called ***gates***. Digital systems are said to be constructed by using logic gates.

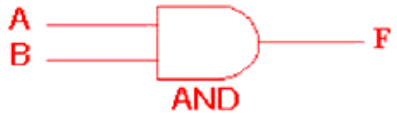
A variety of logic gates are commonly used in digital computer system. Each gate has a distinct graphic symbol and its operation can be described by means of an algebraic expression.

The notations and truth -tables for different logic gates are given below.

### AND gate

The AND gate is an electronic circuit that gives a **high** output (1) only if **all** its inputs are high. A dot (.) is used to show the AND operation i.e. (A.B) or (AB).

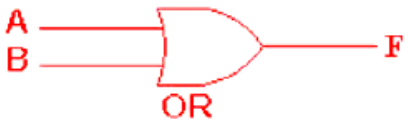
The output F is true if both input A, AND input B are both true, (F = A and B).

| Symbol | Truth Table | | |
|---|---|---|---|
|  2 – input AND Gate | A | B | F |
| | 0 | 0 | 0 |
| | 0 | 1 | 0 |
| | 1 | 0 | 0 |
| | 1 | 1 | 1 |
| Boolean Expression F = A.B | Read as A AND B gives F | | |

➔ **OR gate**

The OR gate is an electronic circuit that gives a high output (1) if **one or more** of its inputs are high. A plus (+) is used to show the OR operation

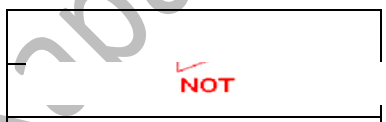
The output F is true if either input A, OR input B is true, (F = A or B).

| Symbol | Truth Table | | |
|---|---|---|---|
|   **2 – input OR Gate** Boolean Expression F = A+B | A | B | F |
| | 0 | 0 | 0 |
| | 0 | 1 | 1 |
| | 1 | 0 | 1 |
| | 1 | 1 | 1 |
| | Read as A OR B gives F | | |

➔ **Inverter / NOT gate**

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an *inverter*. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs.

The output F is only true when the input is NOT true, the output is the inverse or complement of the input (Q = NOT A).

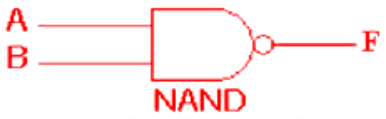
| Symbol | Truth Table | |
|---|---|---|
| | A | F |
| | 0 | 1 |
| | 1 | 0 |
|   **Inverter or NOT Gate** | | |

**Boolean Expression F = NOT A or A** **Read as inverse of A gives F**

➔ **NAND gate**

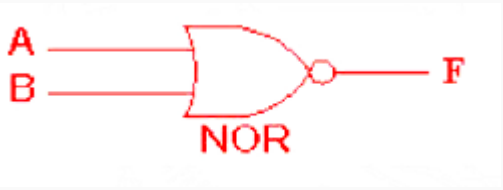
This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if **any** of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

The output F is true if both input A and input B are not true, (Q = not(A and B)).

| Symbol | Truth Table | | |
|---|---|---|---|
|   2 – input NAND Gate | A | B | F |
| | 0 | 0 | 1 |
| | 0 | 1 | 1 |
| | 1 | 0 | 1 |
| | 1 | 1 | 0 |
| Boolean Expression F = A.B | Read as NOT A or NOT B gives F | | |

➔ **NOR gate**

This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if **any** of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion.
The output F is true if both input A and input B are not true, (Q = not(A or B)).

| Symbol | Truth Table | | |
|---|---|---|---|
|   2 – input NOR Gate | A | B | F |
| | 0 | 0 | 1 |
| | 0 | 1 | 0 |
| | 1 | 0 | 0 |
| | 1 | 1 | 0 |
| Boolean Expression F = A+B | Read as NOT A and NOT B gives F | | |

➔ **Exclusive – OR gate (EXOR or XOR)**

The '**Exclusive-OR**' gate is a circuit which will give a high output if **either, but not both**, of its two inputs are high. An encircled plus sign (⊕) is used to show the EOR operation.

The output F is true if either input A or if input B is true, but not both (Q = (A and NOT B) or (NOT A and B)).

| Symbol | Truth Table | | |
|---|---|---|---|
|   2 – input EX- OR Gate Boolean Expression F= A⊕B | A | B | F |
| | 0 | 0 | 0 |
| | 0 | 1 | 1 |
| | 1 | 0 | 1 |
| | 1 | 1 | 0 |

➔ **Exclusive – NOR gate (EXNOR or XNOR)**

The '**Exclusive-NOR**' gate circuit does the opposite to the EOR gate. It will give a low output if **either, but not both**, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.

The output F is true if both input A and input B are the same, either true or false, (F = (A and B) or (NOT A and NOT B)).

| Symbol | Truth Table | | |
|---|---|---|---|
|   2 –input EX – NOR Gate Boolean Expression F = A B | A | B | F |
| | 0 | 0 | 1 |
| | 0 | 1 | 0 |
| | 1 | 0 | 0 |
| | 1 | 1 | 1 |

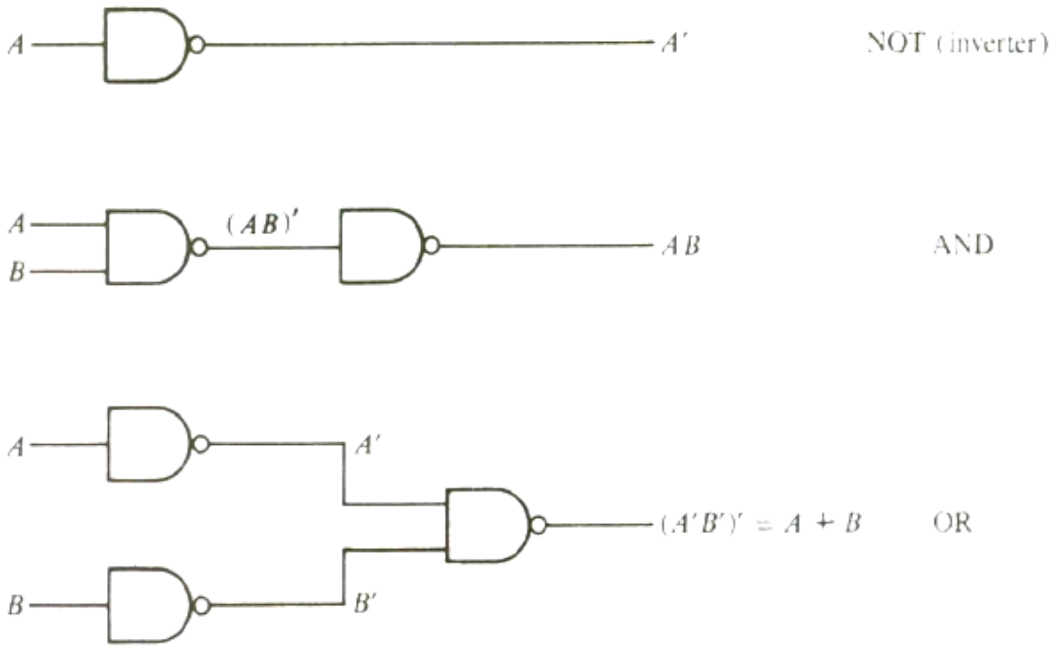
## Universal Gates

NAND and NOR gates are Universal gates because any digital circuit can be implemented with it. Combinational circuits and sequential circuits as well can be constructed with these gates because the flip-flop circuit (the memory element most frequently used in sequential circuits) can be constructed from two NAND gates or NOR gates connected back to back.

To show that any Boolean function can be implemented with NAND or NOR gates, it is needed to show that the logical operations AND, OR, and NOT can be implemented with these gates.

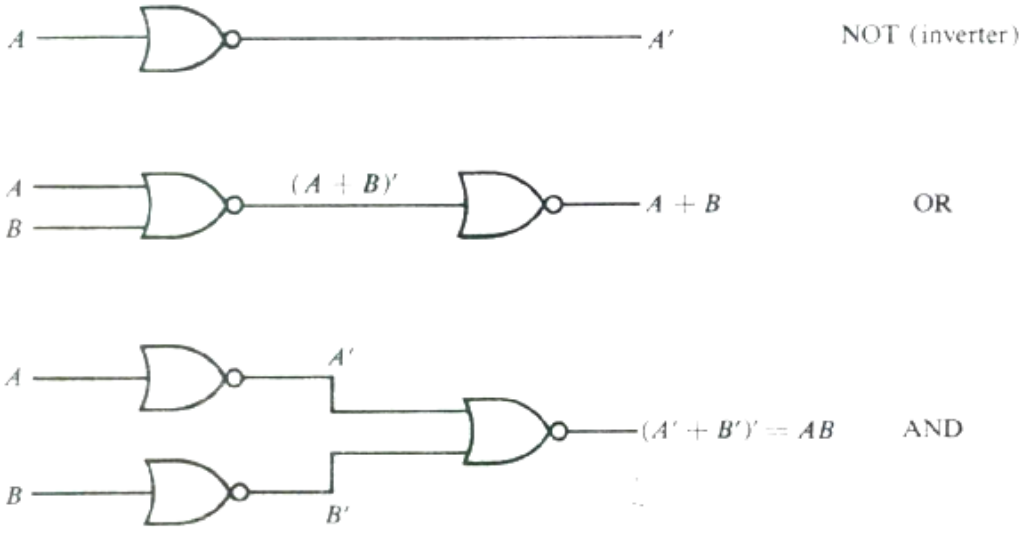
## NAND gate as a Universal Gate

**Implementation of NOT, AND, OR by NAND gates**




## NOR gate as a Universal Gate

**Implementation of NOT, OR, AND by NOR gates**



**BOOLEAN ALGEBRA**

One of the primary requirements when dealing with digital circuits is to find ways to make them as simple as possible. This constantly requires that complex logical expressions be reduced to simpler expressions that nevertheless produce the same results under all possible conditions. The simpler expression can then be implemented with a smaller, simpler circuit, which in turn saves the price of the unnecessary gates, reduces the number of gates needed, and reduces the power and the amount of space required by those gates.

One tool to reduce logical expressions is the mathematics of logical expressions, introduced by George Boole in 1854 and known today as Boolean Algebra.

A Boolean function can be represented by either:

Truth tables
Logic diagrams
Algebraic expression

**Boolean algebra** is an algebra that deals with binary variables and logic operations. Variables are designated by letters such as A, B, x, and y.

A **Boolean function** can be expressed algebraically with binary variables, the logic operation symbols, parentheses, and equal sign.

The result of a Boolean function is either 0 or 1.

The rules of Boolean Algebra are simple and straight-forward, and can be applied to any logical expression. The resulting reduced expression can then be readily tested with a Truth Table, to verify that the reduction was valid.

Boolean algebra is an algebra that deals with binary variables and logic operations. The variables are designated by letters such as A,B, x and y.

**Truth Table**

Table of all possible combinations of variables showing the relation between the values that the variable may take and the result of the operation. The relationship between a function and its binary variables can be represented in a truth table.

**Boolean Function**

It is an expression formed with binary variables, two binary operators (+) OR, (.) AND and unary operator NOT, parenthesis and equal sign.

⇨ **For example the Boolean function**

$F = x + y'z$

The function F is equal to 1 if x is 1 or if both y' and z are equal to 1, F is equal to 0 otherwise.

A Boolean function can be transformed from an algebraic expression into a logic diagram composed of AND, OR and inverter gates.

In logic diagram, there is an inverter for input y to generate complement y'. There is an AND gate for the y'z and an OR gate is used to combine the two terms.

The purpose of Boolean algebra is to facilitate the analysis and design of digital circuit.

## Boolean Operations

There are three basic logical operations:

**AND**: This operation is represented by a dot or by the absence of an operator. For example, x.y = z or xy = z is read "x AND y is equal to z." The logical operation AND is interpreted to mean that z=1 if and only if x=1 and y=1; otherwise z=0.

**OR**: This operation is represented by a plus sign. For example, x+y = z is read "x OR y is equal to z", meaning that z=1 if x=1 or if y=1 or if both x=1 and y=1. If both x=0 and y=0, then z=0.

**NOT**: This operation is represented by a prime (sometimes by a bar). For example, $x' = \bar{z}$ (or $x = \bar{z}$) is read "x not is equal to z", meaning that z is complement of x. In other words, if x=1, then z=0, but if x=0, then z=1.

### BASIC IDENTITIES / BASIC THEOREMS OF BOOLEAN ALGEBRA

| | |
|---|---|
| **a. x + 0 = x** | **b. x . 0 =0** |
| **a. x + 1 =1** | **b. x . 1 = x** |
| **a. x + x = x** | **b. x . x = x** |
| **a. x + x' = 1** | **b. x . x' = 0** |
| **(x')' = x** | |

One part may be obtained from the other if the binary operators and the identity elements are interchanged. The important property of Boolean algebra is called the ***duality principle***.

If the dual of an algebraic expression is desired, we simply interchange OR and AND Operators and replace 1's by 0's and 0's by 1's.

**Commutative Law**

**a. x + y = y + x** **b. xy=yx**

**Associative Law**

**a. x + (y + z) = (x + y) + z** **b. x(yz) = (xy)z**

**Distributive Law**

**a. x(y + z) = xy + xz** **b. x+yz = (x+y)(x+z)**

**DeMorgan's Theorem**

**a. (x+y)' =x'y'** **b. (xy)' = x' + y'**

**PREPARING CIRCUIT FROM GIVEN BOOLEAN FUNCTION**

In a logic circuit, the variables coming on the left hand side of Boolean expression are inputs to circuit and the variable function coming on the right hand side of expression is taken as output. A Boolean function can be implemented into a logic circuit using the basic gates:- AND , OR & NOT

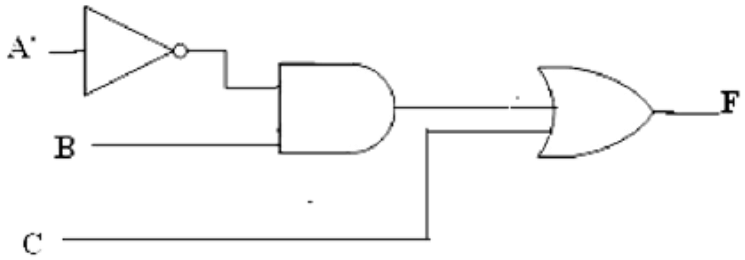
❖ **EXAMPLE NO 1 :** Prepare Truth table and circuit using given Boolean function

**BOOLEAN FUNCTION: - F (A,B,C) = A'. B + C**

The relationship between this function and its binary variables A, B, C can be represented in a truth table as shown below:

➢ **TRUTH TABLE**

| Inputs | | | Intermediates | | Output |
|---|---|---|---|---|---|
| A | B | C | A' | A'B | F |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 |



➢ **LOGIC DIAGRAM / LOGIC CIRCUIT**

There are two methods for converting truth tables to boolean expressions.

1. The Sum of Products
2. Product of Sum

- A binary variable may appear either in its normal form (x) or in its complement form (x'). Now consider two binary variables x and y combined with an AND operations. Since each variable may appear in either form, there are four possible combinations : x'y', x'y, xy' and xy. Each of these four AND terms called a **minterm** or a **standard product**.
- N variables can be combined to form $2^n$ minterms.
- Similarly n variables forming an OR terms, with each variable being primed or unprimed, provide $2^n$ possible combinations, called **maxterm** or **standard product**.

❖ **SUM OF PRODUCT (SOP)**

- A simple method for converting a truth table in a standard form of Boolean expression called the **Sum-of-Products (SOP)** form.
- SOP expressions can easily be implemented as a set of AND gates feeding into a single OR gate
- An SOP expression is literally a sum of Boolean terms called minterms. A minterm is a multiplicative combination of Boolean variables whose output equals 1.
- An example of an SOP expression is ABC + BC + DE, where ABC, BC, and DE are minterms. SOP expressions may be generated from truth tables using the following steps:

1. Determine which rows of the table have an output of 1.
2. Derive each row's minterm, such that the output is 1 given that row's input state.
3. Sum the minterms.

**EXAMPLE NO 1 :** A Truth Table conversion to an SOP expression. Prepare circuit and Boolean expression from given truth table.

- **TRUTH TABLE**

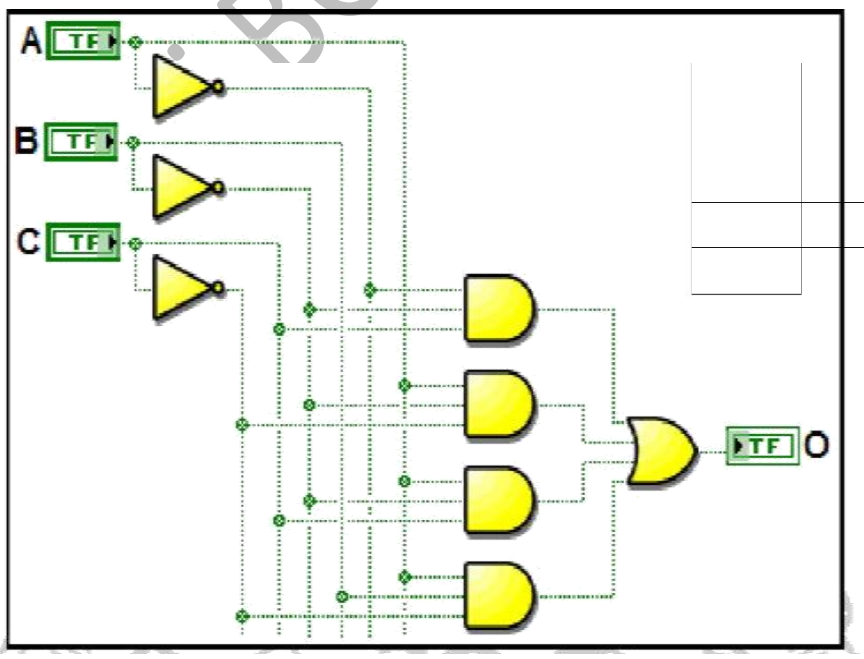
| A | B | C | O | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 1 | A'B'C |
| 0 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | AB'C' |
| 1 | 0 | 1 | 1 | AB'C |
| 1 | 1 | 0 | 1 | ABC' |
| 1 | 1 | 1 | 0 | |

O = A'B'C + AB'C' + AB'C + ABC'

**BOOLEAN FUNCTION O = A'B'C + AB'C' + AB'C + ABC'**

**BOOLEAN FUNCTION O = A'B'C + AB'C' + AB'C + ABC'**

- **LOGIC DIAGRAM / LOGIC CIRCUIT**



❖ **EXAMPLE NO 2: PREPARE BOOLEAN FUNCTION AND CIRCUIT FROM GIVEN BOOLEAN FUNCTION**

| X | Y | Z | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

**EXPLANATION**

- The input variables are X, Y and Z, and the function output is F.
- Let's examine this function in some detail. The only non-zero entries are at:

$$X = 0, Y = 1, Z = 0$$

and

$$X = 1, Y = 0, Z = 1$$

The function is 1 for those two input conditions and zero for all other input conditions.

Now, lets' think about how we can implement this function. Here's a description of what we want to implement:

- We want the output to be 1 whenever we have either
  - X=0 AND Y=1 AND Z=0
- OR when we have
  - X=1 AND Y=0 AND Z=1.

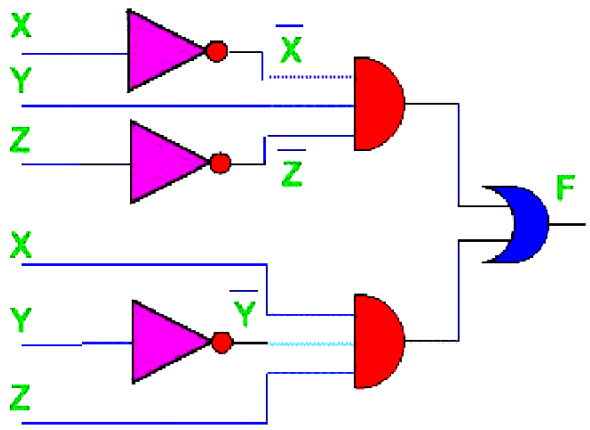
Here's the function:

$$F = \overline{X} \bullet Y \bullet \overline{Z} + X \bullet \overline{Y} \bullet Z$$

This function is read as (NOT-X AND Y AND NOT-Z) OR (X AND NOT-Y AND Z) and when we read NOT-X that means we have to have X=0 to make the three terms ANDed together work out to 1.

Now, let's look at a circuit that will implement this function. Here's the circuit. Notice how the inputs

are grouped into groups of 3, ANDed together (after taking inverses where appropriate) and the results ORed at the end.




In producing our circuit we had to use the form:

$$F = \overline{X} \bullet Y \bullet \overline{Z} + X \bullet \overline{Y} \bullet Z$$

This form is composed of two groups of three. Each group of three is a minterm. What the expression minterm is intended to imply it that each of the groups of three in the expression takes

on a value of 1 only for one of the eight possible combinations of X, Y and Z and their inverses. Important points about minterms include the following.

- In a minterm, each variable, X, Y or Z appears once, either as the variable itself or as the inverse.
- Each minterm corresponds to exactly one entry (row!) in the truth table.

A truth table gives a unique sum-of-products function that follows directly from expanding the ones in the truth table as minterms.

❖ **PRODUCT OF SUM (POS)**

**Product-of-Sums (POS)** expressions are another way of representing truth tables. A POS expression is a product of Boolean terms called maxterms. A maxterm is a summation of Boolean variables whose output equals 0.

POS expressions can easily be implemented as a set of OR gates feeding into a single AND gate

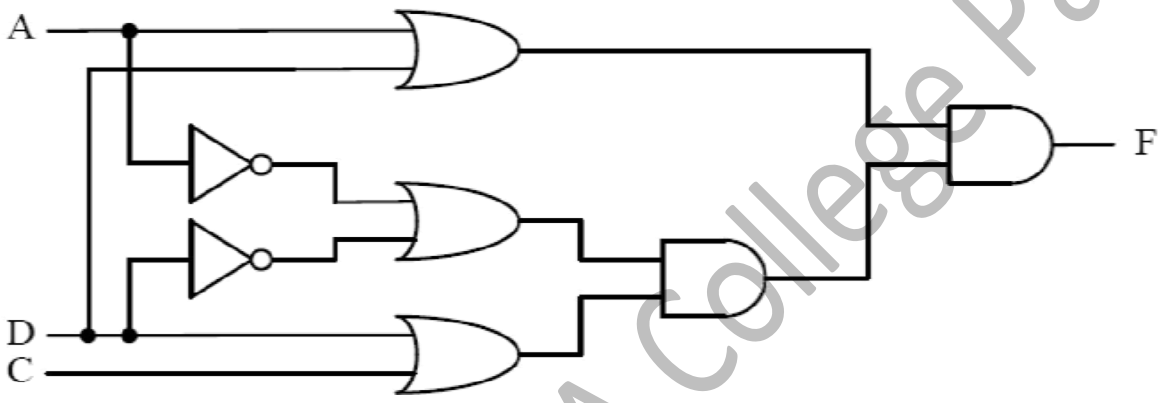
To generate a POS expression from a truth table, perform the following steps:

1. Determine which rows of the table have an output of 0.
2. Derive each row's maxterm, such that the output is 0 given that row's input state.
3. Multiply the maxterms.

**EXAMPLE NO 1 :** Prepare Circuit from given boolean function

**BOOLEAN FUCTION : F(A,B,C,D) = (A + D) . (A' + D') . (C + D)**

➢ **LOGIC DIAGRAM / LOGIC CIRCUIT**



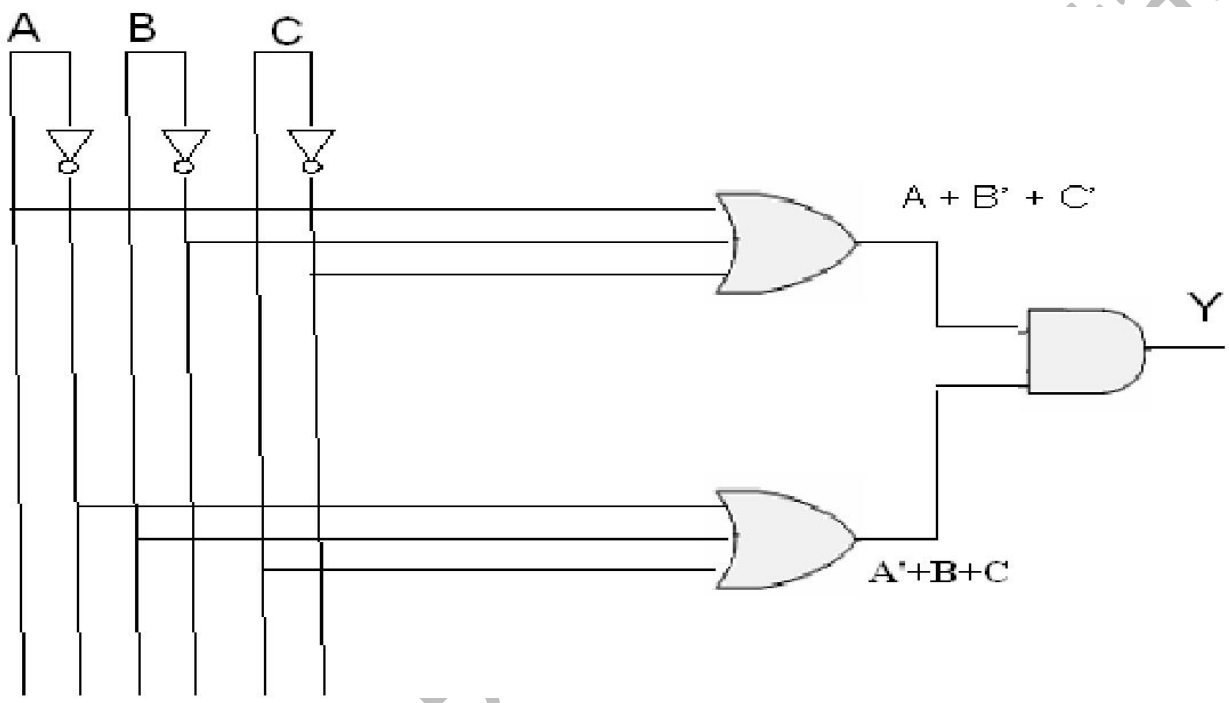

**EXAMPLE NO 2** : A Truth Table conversion to an POS expression. Prepare circuit and Boolean function using given truth table.

➢ **TRUTH TABLE**

| Inputs | | | Output |
|---|---|---|---|
| A | B | C | Y |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| | | | |

**BOOLEAN FUNCTION : Y = (A +B'+C') . (A' + B +C)**

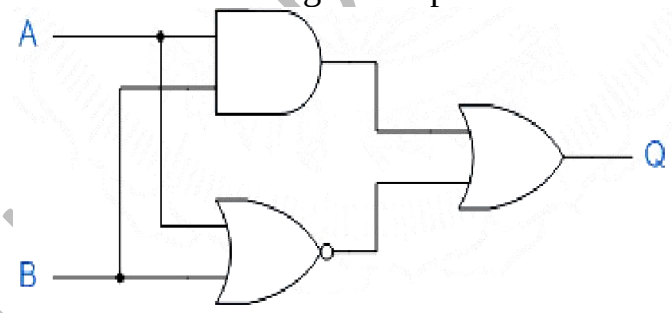
- **LOGIC DIAGRAM / LOGIC CIRCUIT**




- **PREPARING A TRUTH TABLE FROM CIRCUIT**

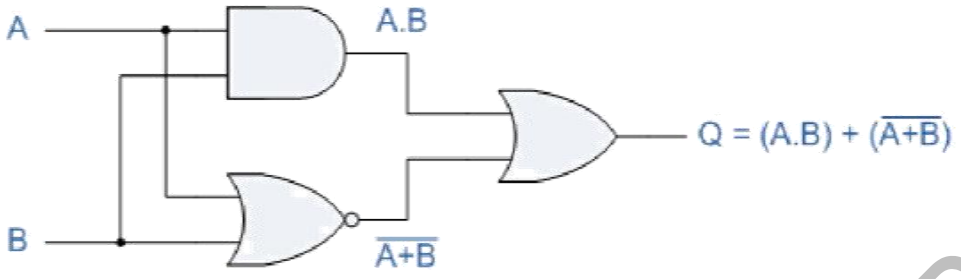
**EXAMPLE NO 1**

Find the Boolean algebra expression and truth table for the following circuit.




The circuit consists of an AND Gate, a NOR Gate and finally an OR Gate.

The expression for the AND gate is A.B, and the expression for the NOR gate is A+B. Both these expressions are also separate inputs to the OR gate which is defined as A+B. Thus the final output expression is given as:




**The output of the system is given as**

**BOOLEAN EXPRESSION / BOOLEAN FUNCTION**

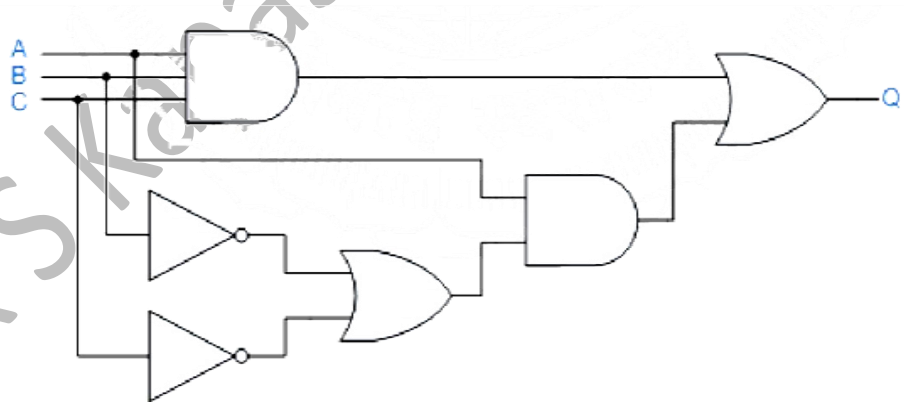
**Q = (A.B) + (A+B)**

**TRUTH TABLE**

| Inputs | | Intermediates | | Output |
|---|---|---|---|---|
| B | A | A.B | A + B | Q |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

❖ **EXAMPLE NO 2**

Find the Boolean algebra expression and truth table for the following circuit.



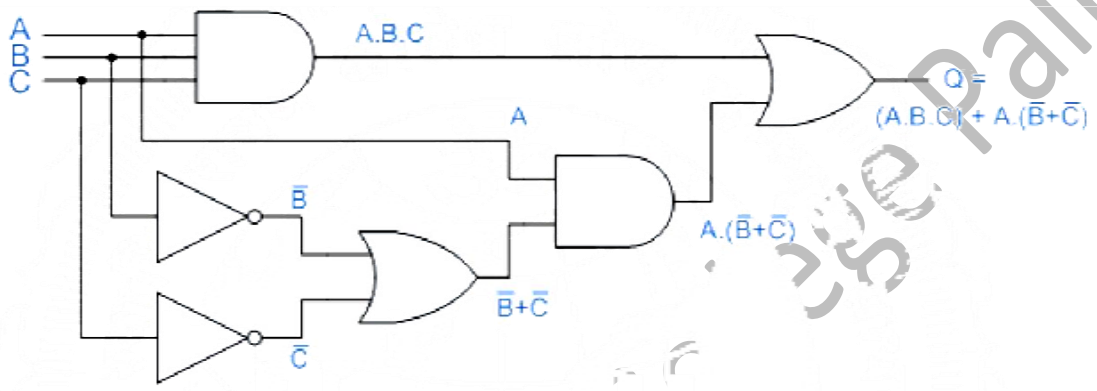

The Circuit consists of simple AND, OR and NOT gates.
The output from the 3-input AND gate is only a "1" when **ALL** the inputs are at logic level "1" (A.B.C). The output from the lower OR gate is only a "1" when one or both inputs B or C are at logic level "0".

The output from the 2-input AND gate is a "1" when input A is a "1" and inputs B or C are at "0".

Then the output at Q is only a "1" when inputs A.B.C equal "1" or A is equal to "1" and both inputs B or C equal "0", A.(B+C).

Thus the final output expression is given as:

➢ **LOGIC DIAGRAM / LOGIC CIRCUIT**




**BOOLEAN EXPRESSION / BOOLEAN FUNCTION**

**Q = (A.B.C) + A. (B'+C')**

➢ **TRUTH TABLE**

| Inputs | | | Intermediates | | | | | Output |
|---|---|---|---|---|---|---|---|---|
| C | B | A | A.B.C | B | C | B+C | A.(B+C) | Q |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

❖ **LAWS OF BOOLEAN ALGEBRA**

Every law has two expressions, (a) and (b). This is known as *duality*. These are obtained by changing every AND(.) to OR(+), every OR(+) to AND(.) and all 1's to 0's and vice-versa.

⇨ **Commutative Law**

(a) $A + B = B + A$

(b) $A\,B = B\,A$

➔ **Associate Law**

(a) $(A + B) + C = A + (B + C)$

(b) $(A\,B)\,C = A\,(B\,C)$

**Distributive Law**

(a) $A\ (B + C) = A\ B + A\ C$
(b) $A + (B\ C) = (A + B)\ (A + C)$

**Identity Law**

(a) $A + A = A$
(b) $A\ A = A$

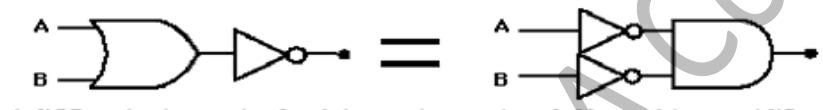
## De Morgan's Theorem

The most important logic theorem for digital electronics, this theorem says that any logical binary expression remains unchanged if we

1. Change all variables to their complements.
2. Change all AND operations to ORs.
3. Change all OR operations to ANDs.
4. Take the complement of the entire expression.

### First De Morgan's Theorem

**(A+B)' = A'B'**



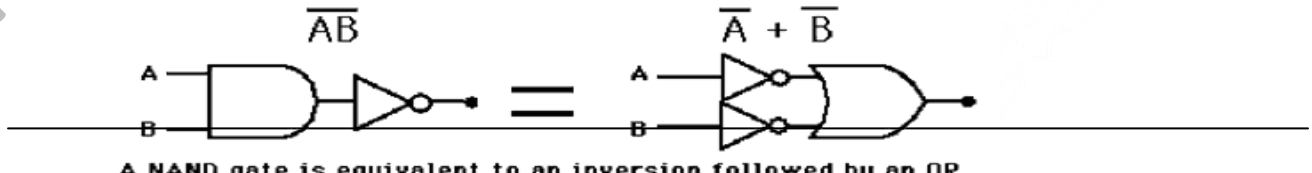

A NOR gate is equivalent to an inversion followed by an AND

**TRUTH TABLE**

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | A+B | (A+B)' | A' | B' | A'B' |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |

### Second De Morgan's Theorem

**(AB)' = A' + B'**




A NAND gate is equivalent to an inversion followed by an OR

NAND

➢ **TRUTH TABLE**

| A | B | AB | (AB)' | A' | B' | A'+B' |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Because of these combination of gates are used often, there are special symbols to represent them
The De Morgan's Laws generalize to n variables :

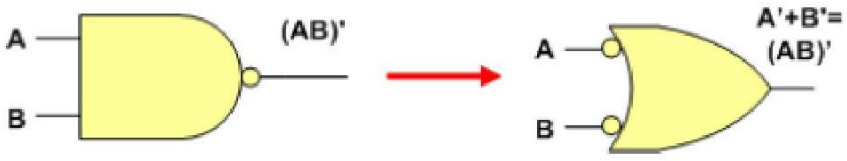
$(X_1 + X_2 + X_3 + \ldots.. + X_n) = X_1'X_2'X_3'\ldots.X_n'$

$(X_1X_2X_3\ldots.X_n)' = X_1' + X_2' + X_3' + \ldots.. + X_n'$

## Equivalent Gates

The shown figure summarizes important cases of gate equivalence. Note that bubbles indicate a complement operation (inverter).

A NAND gate is equivalent to an inverted-input OR gate




An AND gate is equivalent to an inverted-input NOR gate


A NOR gate is equivalent to an inverted-input AND gate


An OR gate is equivalent to an inverted-input NAND gate


## Application of De Morgan's Theorem

- De Morgan's Theorem is useful in the implementation of the basic gate operations with alternative gates, particularly with NAND and NOR gates
- **Universal Gate :** A universal gate is a gate which can implement any Boolean function without need to use any other gate type.
- The NAND and NOR gates are universal gates.

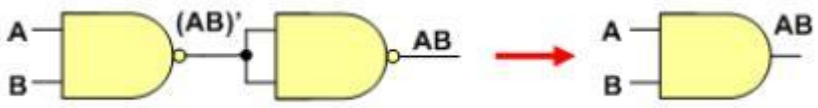
### NAND Gate Operations

**NAND GATE IS A UNIVERSAL GATE:**

To prove that any Boolean function can be implemented using only NAND gates, we will show that the AND, OR, and NOT operations can be performed using only these gates.

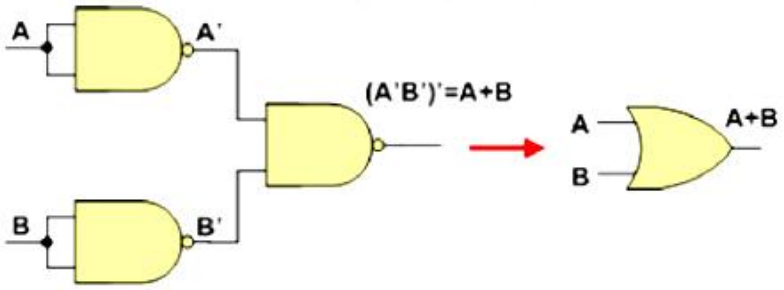
✓ **Implementing AND Using only NAND Gates**

**An AND gate** can be replaced by NAND gates as shown in the figure (The AND is replaced by a NAND gate with its output complemented by a NAND gate inverter).



✓ **Implementing OR Using only NAND Gates**

**An OR gate** can be replaced by NAND gates as shown in the figure (The OR gate is replaced by a NAND gate with all its inputs complemented by NAND gate inverters).




**Thus, the NAND gate is a universal gate since it can implement the AND, OR and NOT functions.**

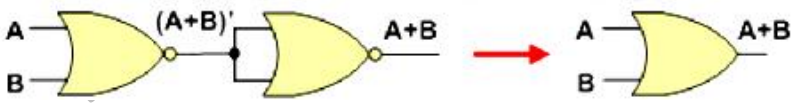
## NOR Gate Operations

**NOR GATE IS A UNIVERSAL GATE**

To prove that any Boolean function can be implemented using only NOR gates, we will show that the AND, OR, and NOT operations can be performed using only these gates.

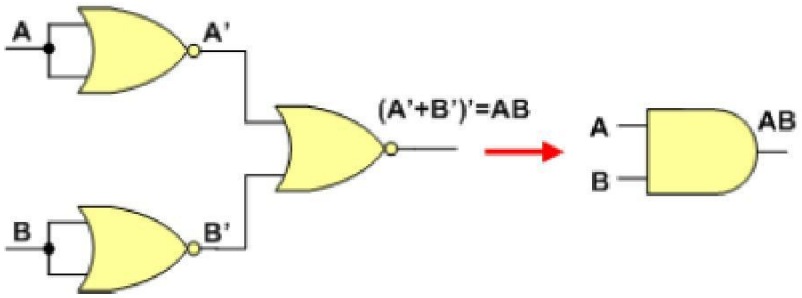
✓ **Implementing OR Using only NOR Gates**

**An OR gate** can be replaced by NOR gates as shown in the figure (The OR is replaced by a NOR gate with its output complemented by a NOR gate inverter)




✓ **Implementing AND Using only NOR Gates**

**An AND gate** can be replaced by NOR gates as shown in the figure (The AND gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters)




**Thus, the NOR gate is a universal gate since it can implement the AND, OR and NOT functions.**

**SOP USING NAND GATE**

Any logic function can be implemented using NAND gates. To achieve this, first the logic function has to be written in Sum of Product (SOP) form. Once logic function is converted to SOP, then is very easy to implement using NAND gate. In other words any logic circuit with AND gates in first level and OR gates in second level can be converted into a NAND-NAND gate circuit.

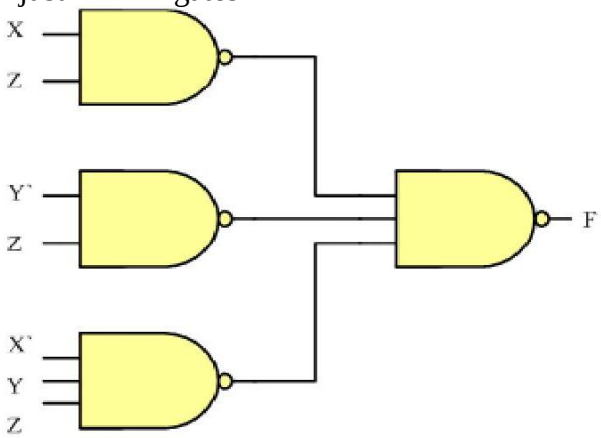
- An SOP expression can be easily implemented using
  - **2-level AND-OR circuit**
  - **2-level NAND circuit**

**<u>EXAMPLE NO 1</u> : Implement the following SOP function**

**F = XZ + Y'Z + X'YZ**

**2 – LEVEL NAND CIRCUIT**

Gates with input bubble with the NAND gate. Now we have circuit which is fully implemented with just NAND gates

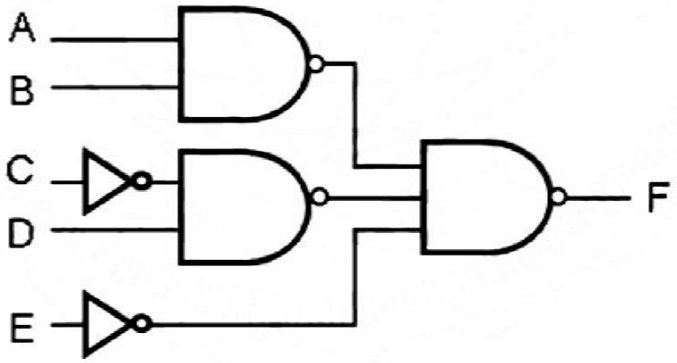


**EXAMPLE NO 2 : Implement the following SOP function**

**F = A B + C' D + E**

Gates with input bubble with the NAND gate. Now we have circuit which is fully implemented with just NAND gates

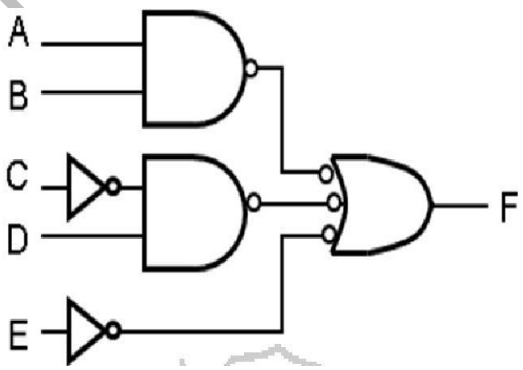
**2 – LEVEL NAND CIRCUIT**




**OR**

**2 – LEVEL AND – OR CIRCUIT**

If bubbles are introduced at AND gates output and OR gates inputs the circuit becomes as shown in figure.




**POS USING NOR GATE**

Any logic function can be implemented using NOR gates. To achieve this, first the logic function has to be written in Product of Sum (POS) form. Once it is converted to POS, then it's very easy to implement using NOR gate. In other words any logic circuit with OR gates

in first level and AND gates in second level can be converted into a NOR-NOR gate circuit.

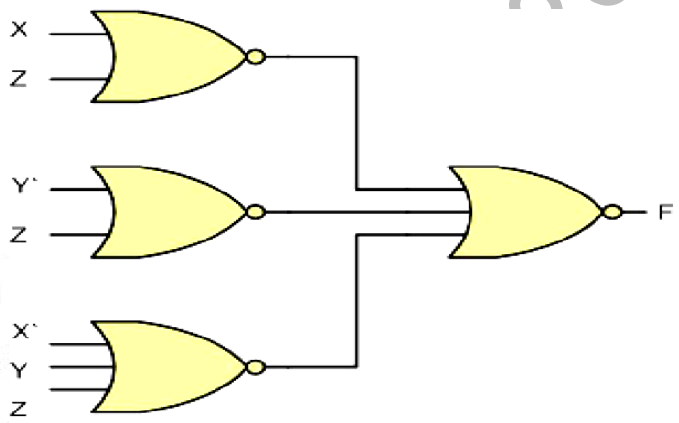
- A POS expression can be easily implemented using
  - 2-level OR-AND circuit
  - 2-level NOR circuit

**EXAMPLE NO 1 : Implement the following POS function**

**F = (X+Z) (Y'+Z) (X'+Y+Z)**

**2 – LEVEL NOR CIRCUIT**

Gates with input bubble with the NOR gate. Now we have circuit which is fully implemented with just NOR gates

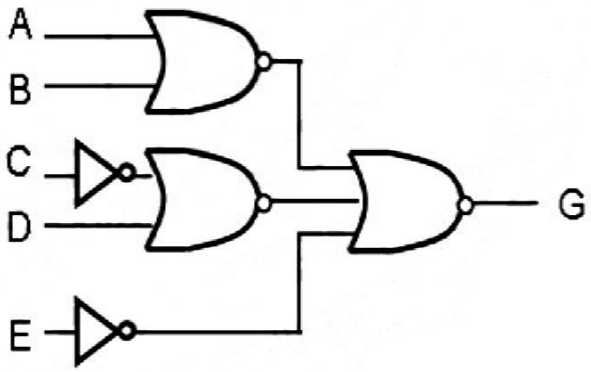



**EXAMPLE NO 2 : Implement the following POS function**

**G = (A+B) (C'+D) E**

**2 – LEVEL NOR CIRCUIT**

Gates with input bubble with the NOR gate. Now we have circuit which is fully implemented with just NAND gates



**OR**

**2 – LEVEL OR – AND CIRCUIT**

If bubbles are introduced at OR gates output and AND gates inputs the circuit becomes as shown in figure