**Boolean Algebra**

Boolean Algebra is used to analyze and simplify the digital (logic) circuits. It uses only the binary numbers i.e. 0 and 1. It is also called as **Binary Algebra** or **logical Algebra**. Boolean algebra was invented by **George Boole** in 1854.

Rule in Boolean Algebra

Following are the important rules used in Boolean algebra.

* Variable or literals used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
* Complement of a variable is represented by an overbar (-). Thus, complement of variable A is represented as , Thus if A = 0 then  = 1 and A = 1 then  = 0.
* Logical ORing of the variables is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as A + B + C.
* Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometime the dot may be omitted like A.B.C.

**Boolean Laws**

There are six types of Boolean Laws.

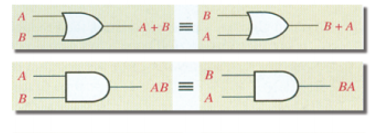
**Commutative law**

Any binary operation which satisfies the following expression is referred to as commutative operation w.r.t addition and multiplication.

**Commutative Law**

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **A+B** | **B+A** |
| **0** | **0** | **0** | **0** |
| **0** | **1** | **1** | **1** |
| **1** | **0** | **1** | **1** |
| **1** | **1** | **1** | **1** |

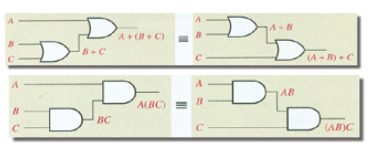
Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

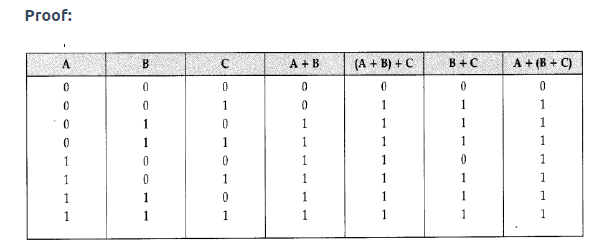


**Associative law**

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

Associative Law





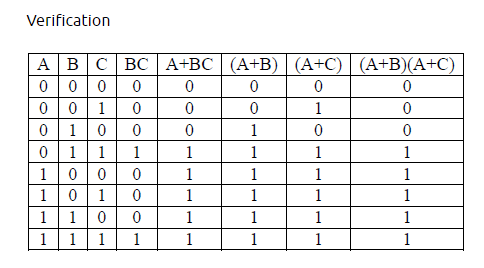
**Distributive law**

Distributive law states the following condition.

Distributive Law

A+BC=(A+B)(A+C)





**AND laws**

These laws use the AND operation, therefore, they are called as **AND** laws.

AND Law

OR laws

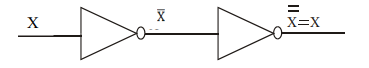
**These laws use the OR operation, Therefore, they are called as OR laws.**

OR Law

**INVERSION law**

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

**NOT Law**



**Absorption Law**

**A + AB = A**

Proof; L.H.S = A + AB = A(1+B)

But 1+B = 1

Therefore, L.H.S = A.1 = A

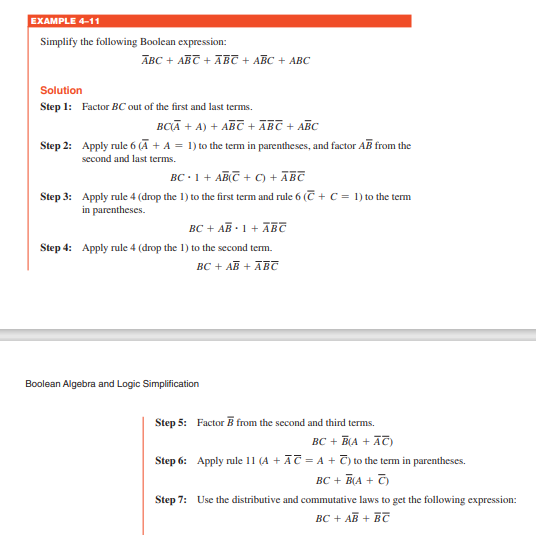
**Other Important law**

**A + B = A + B**

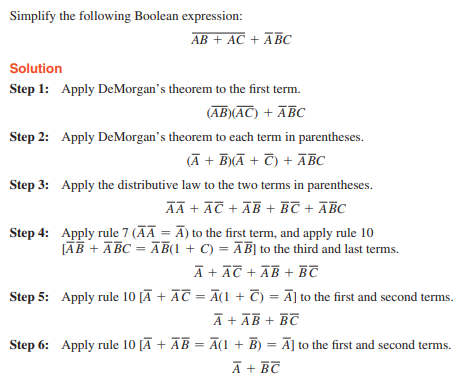
Proof L.H.S = **A + B**

**As A**= A + AB

So L.H.S = = A + AB + **B = A + B(1 + = A+B**



Example



### De-Morgan’s First Theorem

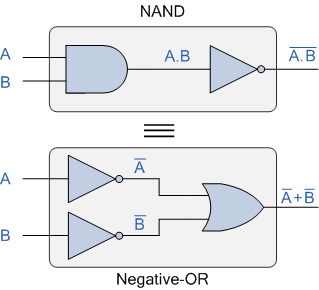
De-Morgan’s First theorem proves that when two (or more) input variables are AND’ed are complemented, they are equivalent to the OR of the complements of the individual variables. Thus, the equivalent of the NAND function will be a negative-OR function, proving that  = + We can show this operation using the following table.

Table showing verification of the De Morgan's first theorem −



We can also show that  = +   using logic gates

### De Morgan’s First Law Implementation using Logic Gates



### De Morgan’s Second Theorem

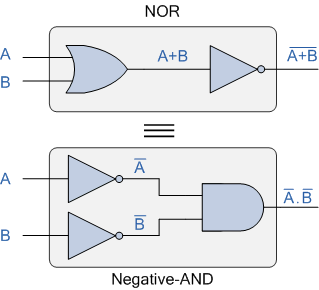
De Morgan’s Second theorem proves that when two (or more) input variables are OR’ed complemented, they are equivalent to the AND of the complements of the individual variables. Thus the equivalent of the NOR function is a negative-AND function proving that  =   and again we can show operation this using the following truth table.

Table showing verification of the De Morgan's second theorem −



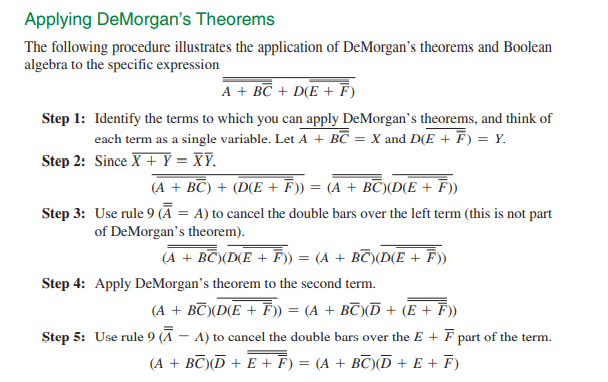
We can also show that A+B = A.B using the following logic gates example.

### De Morgan’s Second Law Implementation using Logic Gates

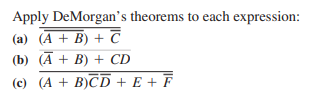


Example

Develop the truth table for 3-input of De-Morgan laws.

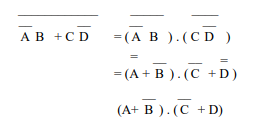


**Examples**

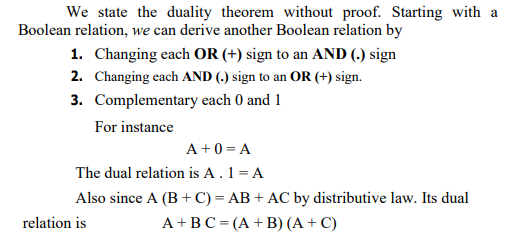


****

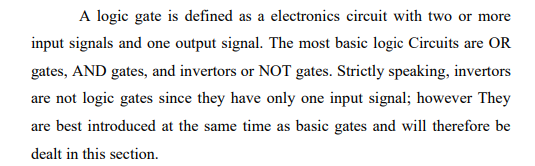
**Solution**

****

Duality Theorem



**Logic Gates**



The logic gates perform ON/OFF operation.

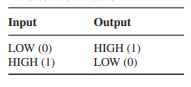
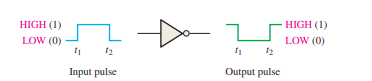
Three types of logic gates

1. Basic Gates (NOT gate, AND gate, OR gate)
2. Universal Gates (NAND gate, NOR gate)
3. Exclusive OR gate, Exclusive NOR gate.

**NOT gate**

The inverter (NOT gate) performs the operation called inversion or complementation. The inverter changes 0 to a1 and 1 to a 0 in terms of bit.

The truth table and symbol are

This shows when input is LOW, the output is HIGH or when input is HIGH, the output is LOW.

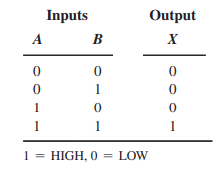
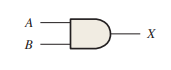
The logic expression for inverter is X =

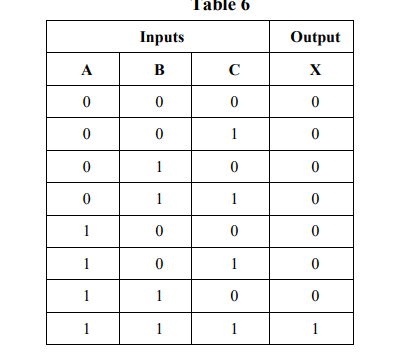


**AND Gate**

The AND gate performs the logical multiplication, it is multiple input and single output component. The logical equation is X = A . B

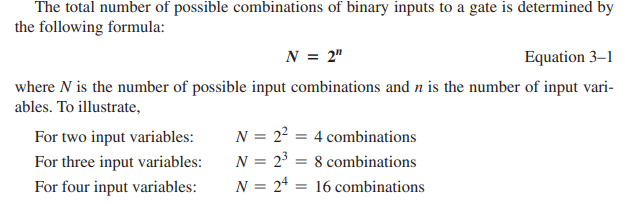
The truth table and logical symbol for AND gate is





Logical Operation

From the truth table we see that if both or any one input is at LOW logic, the output is at LOW logic. The output is at HIGH logic only if both or all inputs are at HIGH logic.



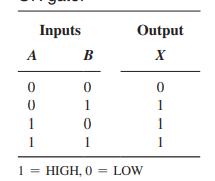
Example

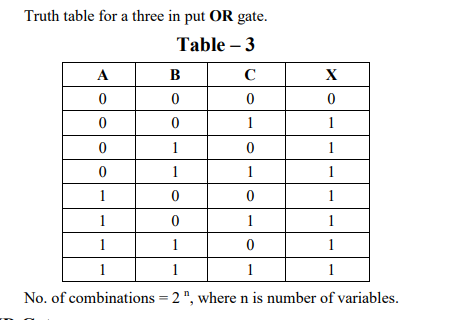


**OR gate**

OR gate is a multiple input and single output component, it is used to perform logic addition. The logic equation is X = A + B

The truth table and logic symbol is



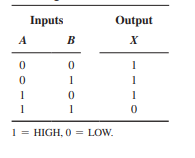


From the truth table, the output is at LOW logic only when all inputs are at LOW logic. The output is HIGH logic when all inputs or any one of the input is HIGH.

**NAND Gate**

The NAND gate is a universal gate and it is the combination of AND gate and NOT gate. The logical equation is

The truth table and logical symbol is

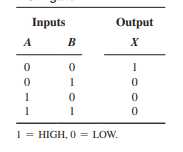
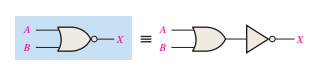


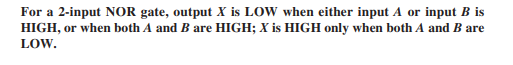


**NOR Gate**

The NOR gate is a universal gate and composed of OR and NOT gate. The logic equation is

The truth table and logic symbol is



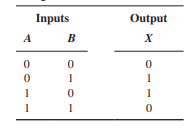


**The Exclusive OR gate**

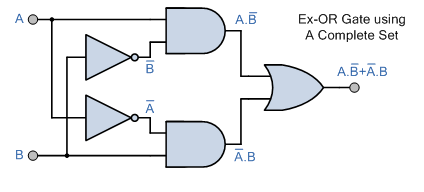
The Exclusive OR is used for logical comparison. The output is HIGH logic if both inputs are different and output is LOW logic if both inputs are same. The logical equation is

**X= (A ⊕ B) = A + B**

The truth table and logic symbol is





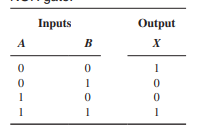


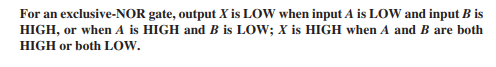
**Exclusive NOR gate**

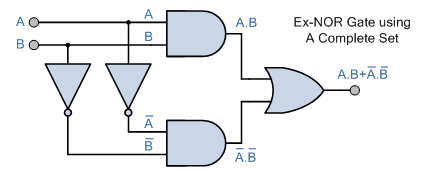
The logical equation for Exclusive NOR gate is

**X = = AB +**

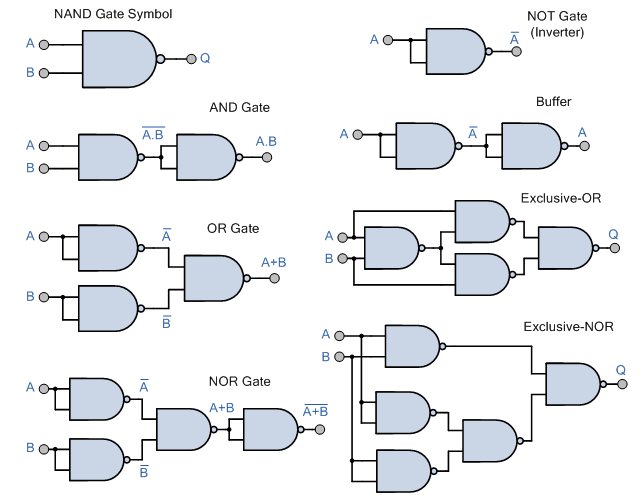
The truth table and logic symbol is







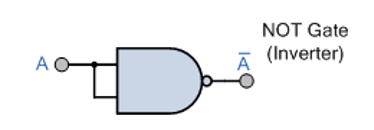
## Implementation of Logic Functions Using Only NAND Gates



**NOT gate using NAND gate**

As the logic equation for NOT gate is X =

So both inputs of NAND gate are joined



**AND gate using NAND**

As the logic equation of AND gate is X = A . B

Take the double inversion of R.H.S

X =

## 

## OR gate using NAND gate

## As the logic equation for OR gate X = A + B

## Take double inversion of R.H.S

## X =

## Apply De Morgan theorem, we have X =

## 

## NOR Using NAND gate

## As the logic equation for NOR gate is X =

## Apply De Morgan Theorem X = .

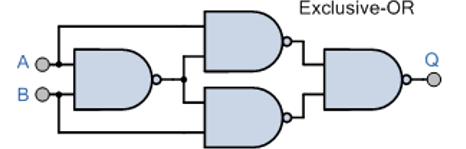
## Take double inversion X =

## 

## Ex-OR gate using NAND gate

**As the logic expression for Ex-OR gate is X = = A + B**

Take the double inversion X = = . =

****

## Ex-NOR gate using NAND gate

## 

## Implementation of Logic Functions Using Only NOR Gates