# LOGIC GATES

* Logic gates are the fundamental building blocks of digital systems.
* There are 3 basic types of gates AND, OR and NOT.
* Logic gates are electronic circuits because they are made up of a number of electronic devices and components.
* Inputs and outputs of logic gates can occur only in 2 levels. These two levels are termed HIGH and LOW, or TRUE and FALSE, or ON and OFF or simply 1 and 0.
* The table which lists all the possible combinations of input variables and the corresponding outputs is called a truth table.

##### LEVEL LOGIC:-

A logic in which the voltage levels represents logic 1 and logic 0. Level logic may be positive or negative logic. **Positive Logic:-**

A positive logic system is the one in which the higher of the two voltage levels represents the logic 1 and the lower of the two voltages level represents the logic 0.

##### Negative Logic:-

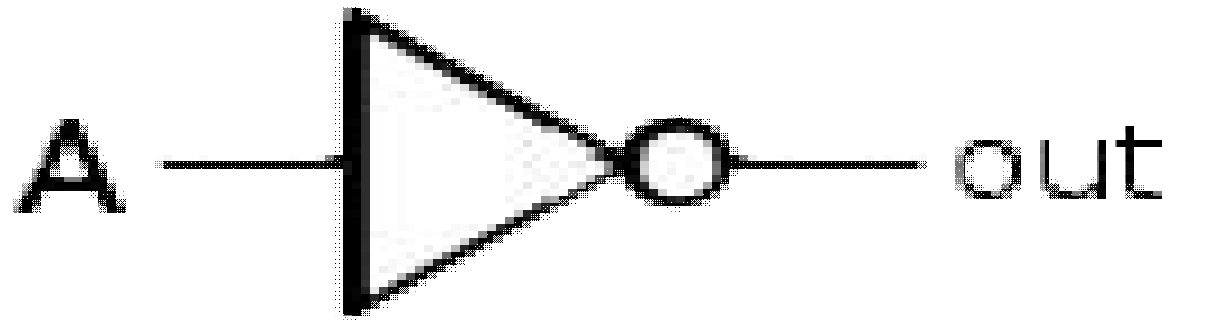
A negative logic system is the one in which the lower of the two voltage levels represents the logic 1 and the higher of the two voltages level represents the logic 0.

**DIFFERENT TYPES OF LOGIC GATES**:- **NOT GATE (INVERTER):-**

* A NOT gate, also called and inverter, has only one input and one output.
* It is a device whose output is always the complement of its input.
* The output of a NOT gate is the logic 1 state when its input is in logic 0 state and the logic 0 state when its inputs is in logic 1 state.

##### IC No. :- 7404

**Logic Symbol**



**Timing Diagram**

**1 0 0 1**

**Truth table**

|  |  |
| --- | --- |
| **INPUT**  **A** | **OUTPUT**  **A** |
| 0 | **1** |
| 1 | 0 |

**A**

|  |  |  |
| --- | --- | --- |
|  |  |  |

##### A

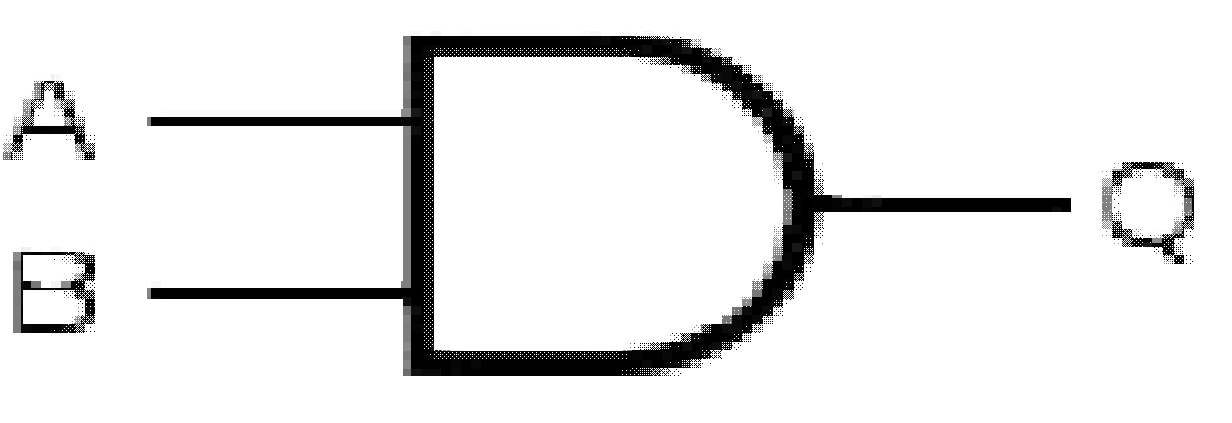
**0 1 1 0**

|  |  |  |
| --- | --- | --- |
|  |  |  |

### AND GATE:-

* An AND gate has two or more inputs but only one output.
* The output is logic 1 state only when each one of its inputs is at logic 1 state.
* The output is logic 0 state even if one of its inputs is at logic 0 state.

##### IC No.:- 7408

**Logic Symbol Truth Table**

|  |  |  |
| --- | --- | --- |
|  | | **OUTPUT** |
| **A** | **B** | **Q=A . B** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Timing Diagram**

**0 0 1 1**

##### A

**0 1 0 1**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

##### B

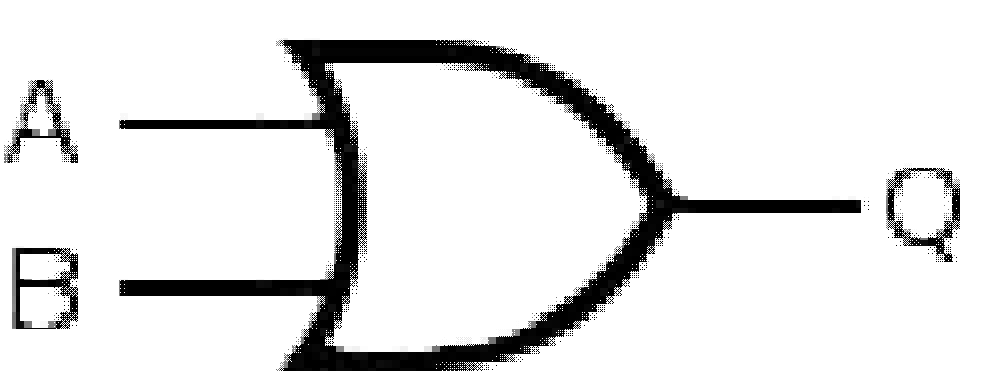
**0 0 0 1**

**Q**

### OR GATE:-

* An OR gate may have two or more inputs but only one output.
* The output is logic 1 state, even if one of its input is in logic 1 state.
* The output is logic 0 state, only when each one of its inputs is in logic state.

##### IC No.:- 7432

**Logic Symbol Truth Table**

|  |  |  |
| --- | --- | --- |
| **INPUT** | | **OUTPUT** |
| **A** | **B** | **Q=A + B** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Timing Diagram**

**0 0 1 1**

##### A

**0 1 0 1**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

##### B

**0 1 1 1**

**Q**

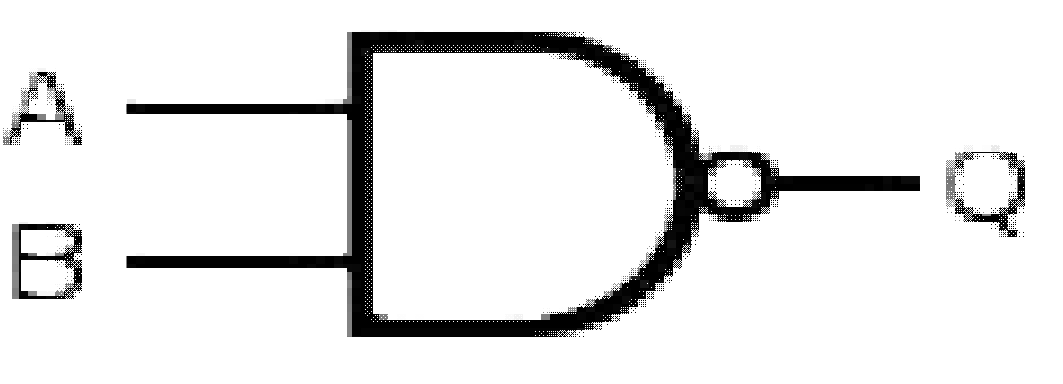
### NAND GATE:-

* NAND gate is a combination of an AND gate and a NOT gate.
* The output is logic 0 when each of the input is logic 1 and for any other combination of inputs, the output is logic 1.

##### IC No.:- 7400 two input NAND gate 7410 three input NAND gate 7420 four input NAND gate 7430 eight input NAND gate

**Logic Symbol Truth Table**

|  |  |  |
| --- | --- | --- |
| **INPUT** | | **OUTPUT** |
| **A** | **B** | **Q= A . B** |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



##### Timing Diagram

**0 0 1 1**

##### A

**0 1 0 1**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

##### B

**1 1 1 0**

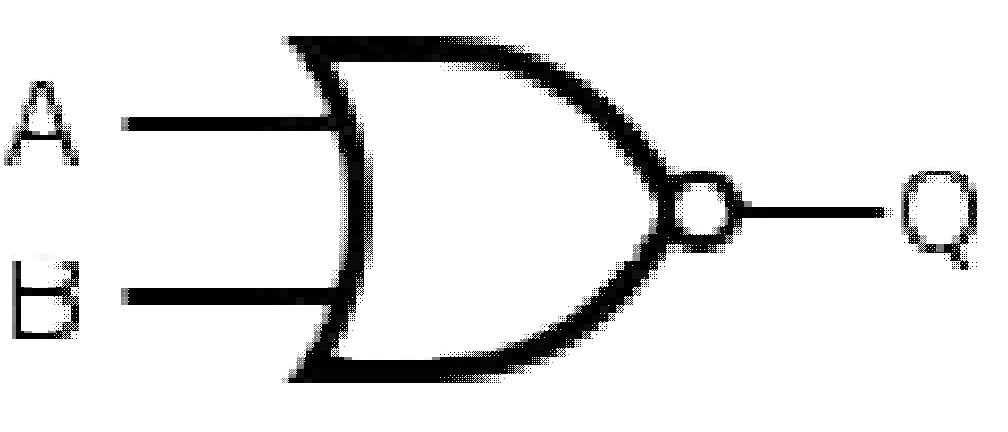
**Q**

### NOR GATE:-

* NOR gate is a combination of an OR gate and a NOT gate.
* The output is logic 1, only when each one of its input is logic 0 and for any other combination of inputs, the output is a logic 0 level.

##### IC No.:- 7402 two input NOR gate 7427 three input NOR gate 7425 four input NOR gate

**Logic Symbol Truth Table**



|  |  |  |
| --- | --- | --- |
| **INPUT** | | **OUTPUT** |
| **A** | **B** | **Q= A + B** |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**Timing Diagram**

**0 0 1 1**

##### A

**0 1 0 1**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

##### B

**1 0 0 0**

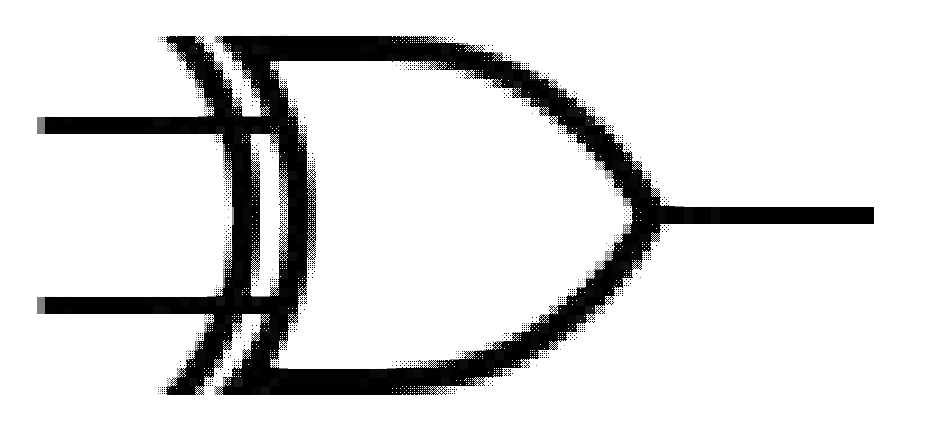
**Q**

### EXCLUSIVE – OR (X-OR) GATE:-

* An X-OR gate is a two input, one output logic circuit.
* The output is logic 1 when one and only one of its two inputs is logic 1. Whe**n** both the inputs is logic 0 or when both the inputs is logic 1, the output is logic 0.

##### IC No.:- 7486

**Logic Symbol**



INPUTS are **A** and **B** OUTPUT is **Q** = A  B

**=** A B + A B

**Timing Diagram**

**0 0 1 1**

##### Truth Table

|  |  |  |
| --- | --- | --- |
| **INPUT** | | **OUTPUT** |
| **A** | **B** | **Q = A**  **B** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**A**

##### 0 1 0 1

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

**B**

##### 0 1 1 0

**Q**

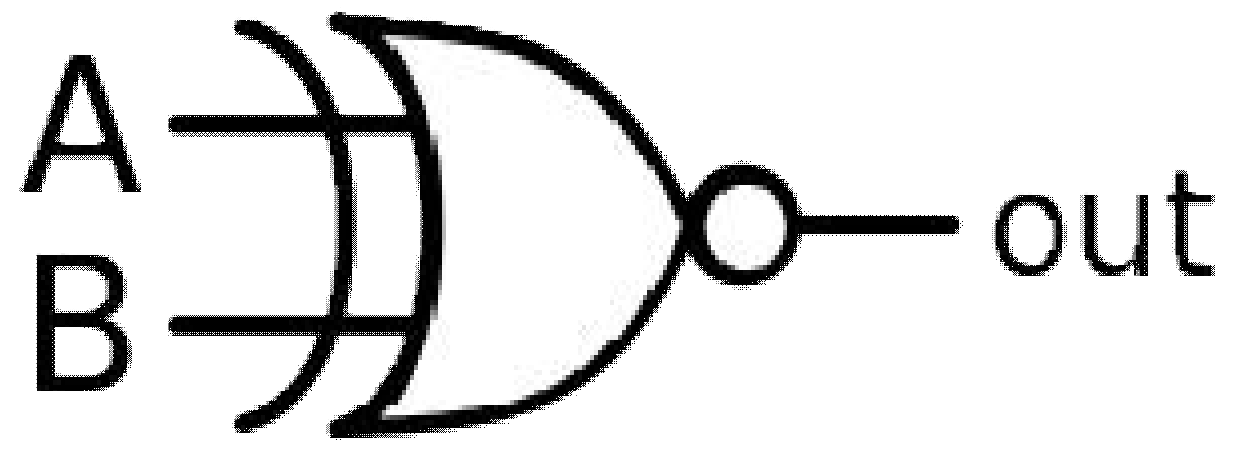
### EXCLUSIVE – NOR (X-NOR) GATE:-

* An X-NOR gate is the combination of an X-OR gate and a NOT gate.
* An X-NOR gate is a two input, one output logic circuit.
* The output is logic 1 only when both the inputs are logic 0 or when both the inputs is 1.
* The output is logic 0 when one of the inputs is logic 0 and other is 1.

##### IC No.:- 74266

**Logic Symbol**

|  |  |  |
| --- | --- | --- |
| **INPUT** | | **OUTPUT** |
| **A** | **B** | **OUT =A XNOR B** |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



##### OUT =A B + A B

**= A XNOR B**

**Timing Diagram**

**0 0 1 1**

##### A

**B**

##### OUT

**0 1 0 1**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

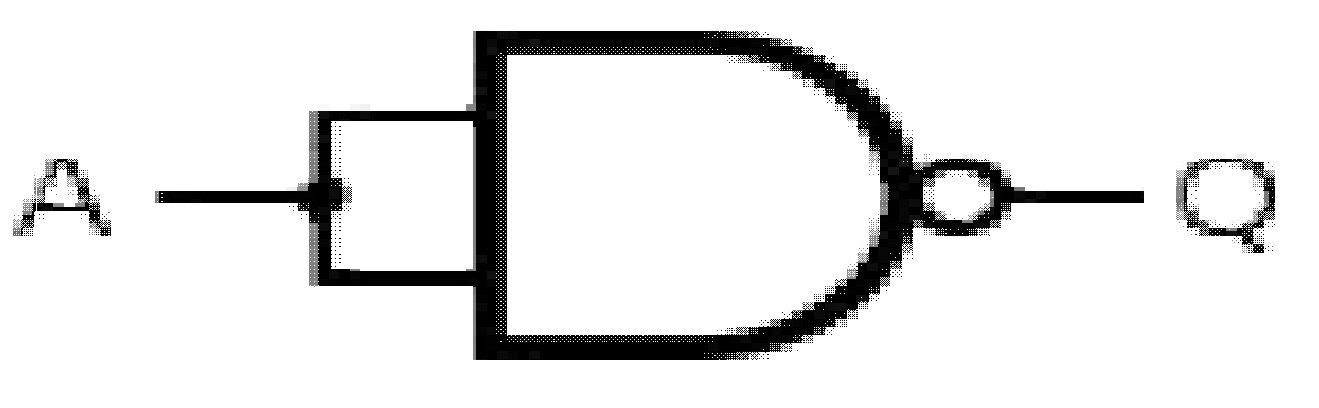
**1 0 0 1**

### UNIVERSAL GATES:-

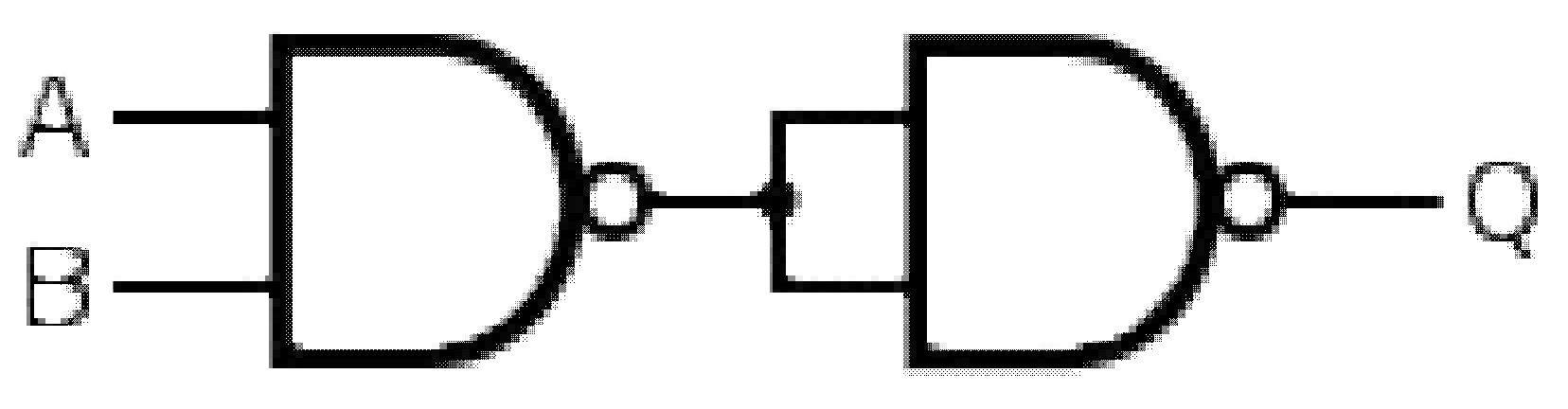
There are 3 basic gates AND, OR and NOT, there are two universal gates NAND and NOR, each of which can realize logic circuits single handedly. The NAND and NOR gates are called universal building blocks. Both NAND and NOR gates can perform all logic functions i.e. AND, OR, NOT, EXOR and EXNOR.

### NAND GATE:-

1. **Inverter from NAND gate**

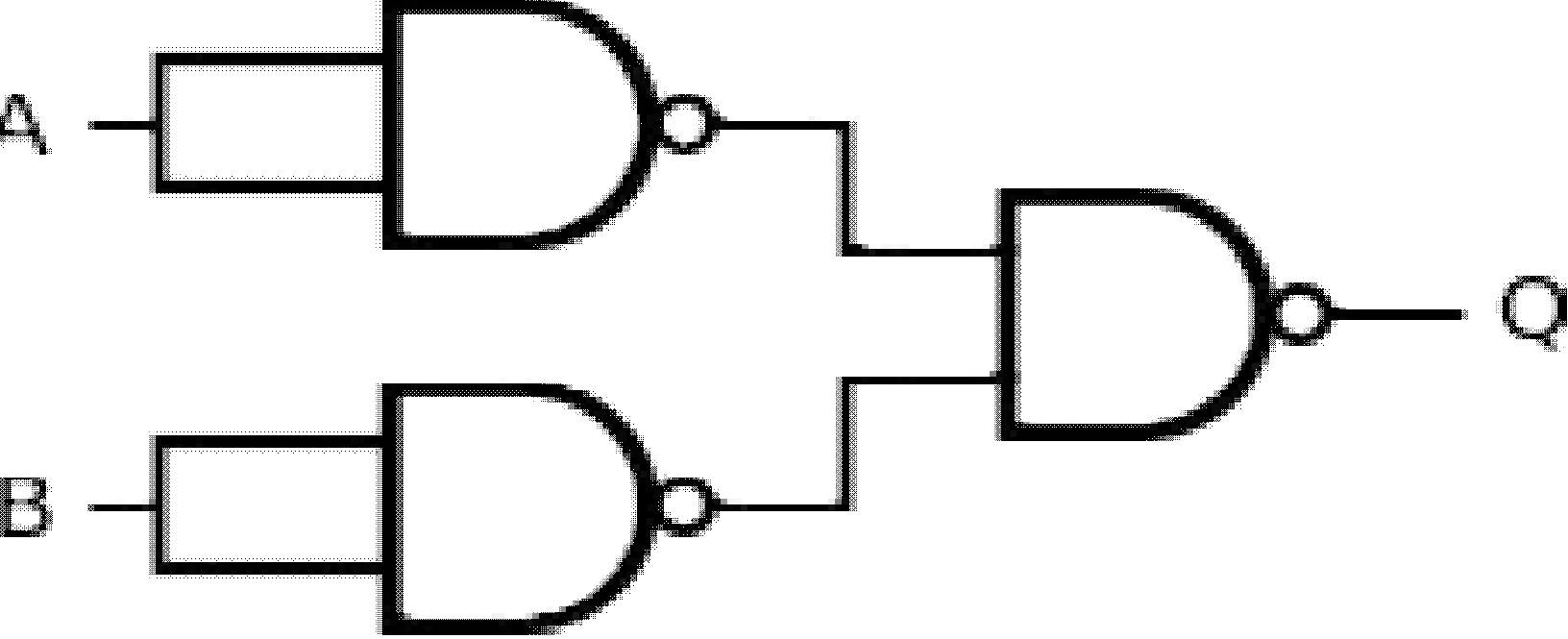
Input **= A ** Output **Q = A**

1. **AND gate from NAND gate**

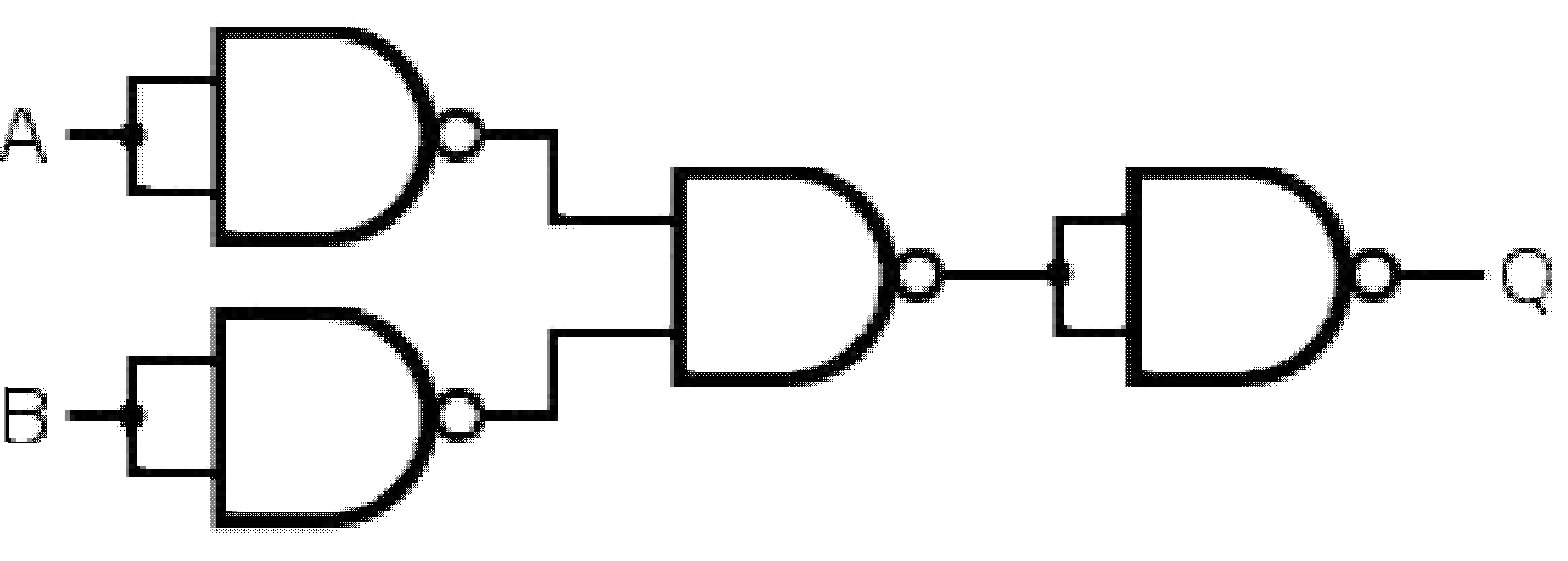
Input s are **A** and **B** Output **Q = A.B**

1. **OR gate from NAND gate**

Inputs are **A** and **B** Output **Q = A+B**

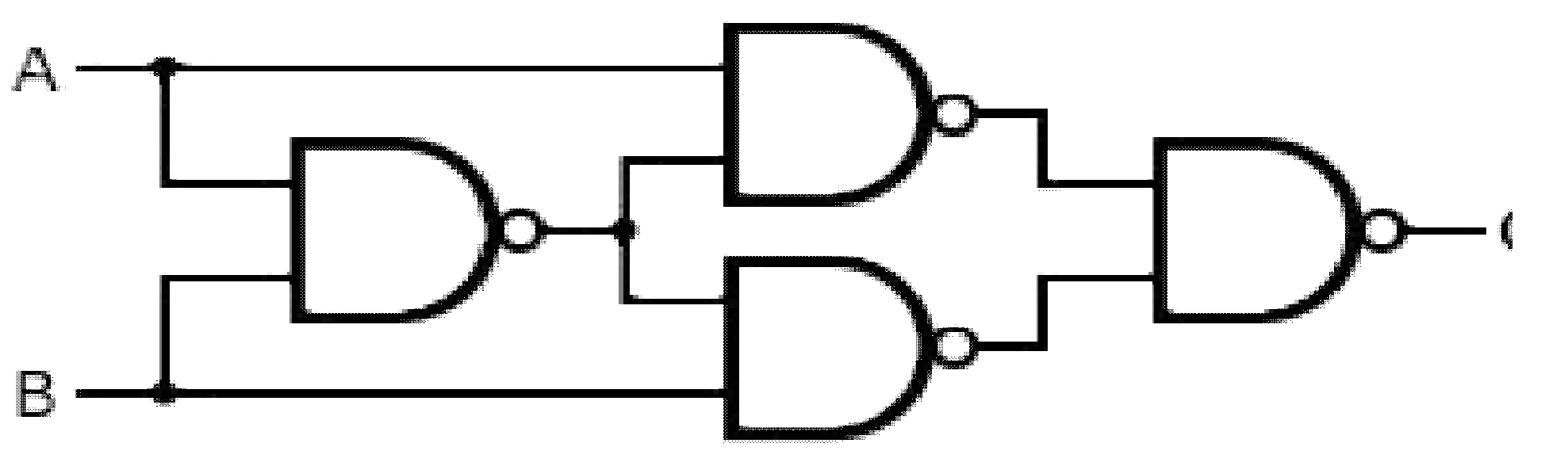


1. **NOR gate from NAND gate**

Inputs are **A** and **B** Output **Q = A+B**

1. **EX-OR gate from NAND gate**

Inputs are **A** and **B** Output **Q = A B + AB**



1. **EX-NOR gate From NAND gate**

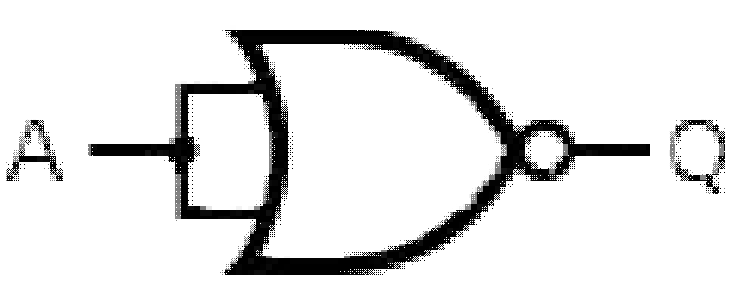
Inputs are **A** and **B**

Output **Q = A B + A B**

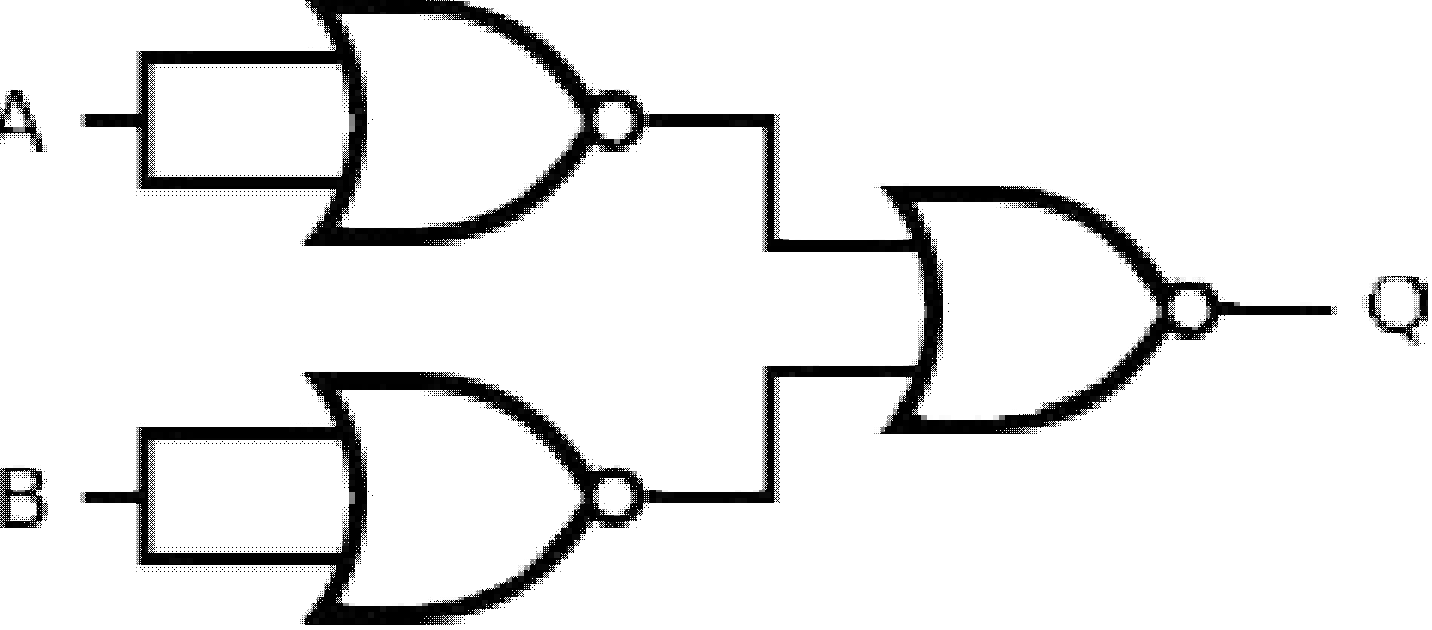


### NOR GATE:-

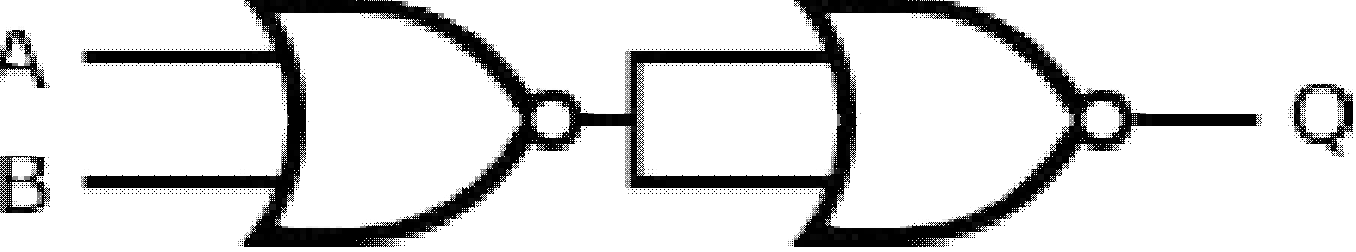
##### Inverter from NOR gate Input = A

Output **Q = A**

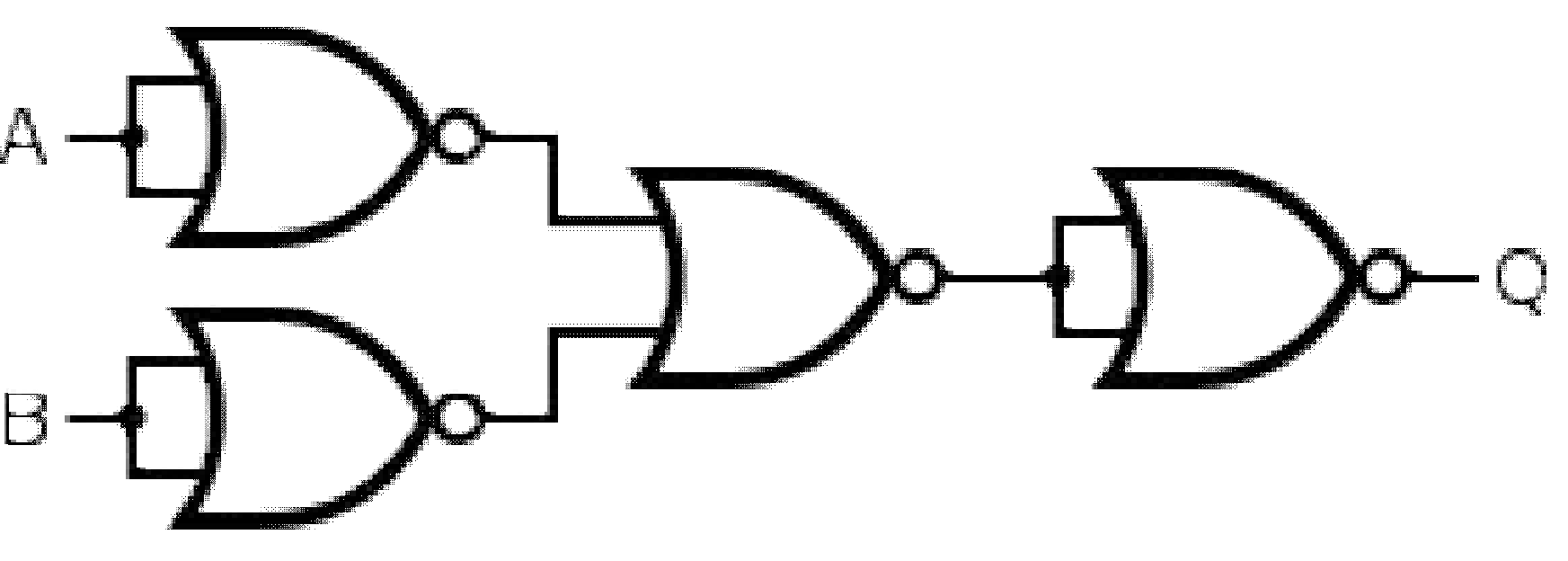
1. **AND gate from NOR gate** Input s are **A** and **B** Output **Q = A.B**



1. **OR gate from NOR gate**

Inputs are **A** and **B** Output **Q = A+B**

1. **NAND gate from NOR gate**

Inputs are **A** and **B** Output **Q = A.B**

1. **EX-OR gate from NOR gate**

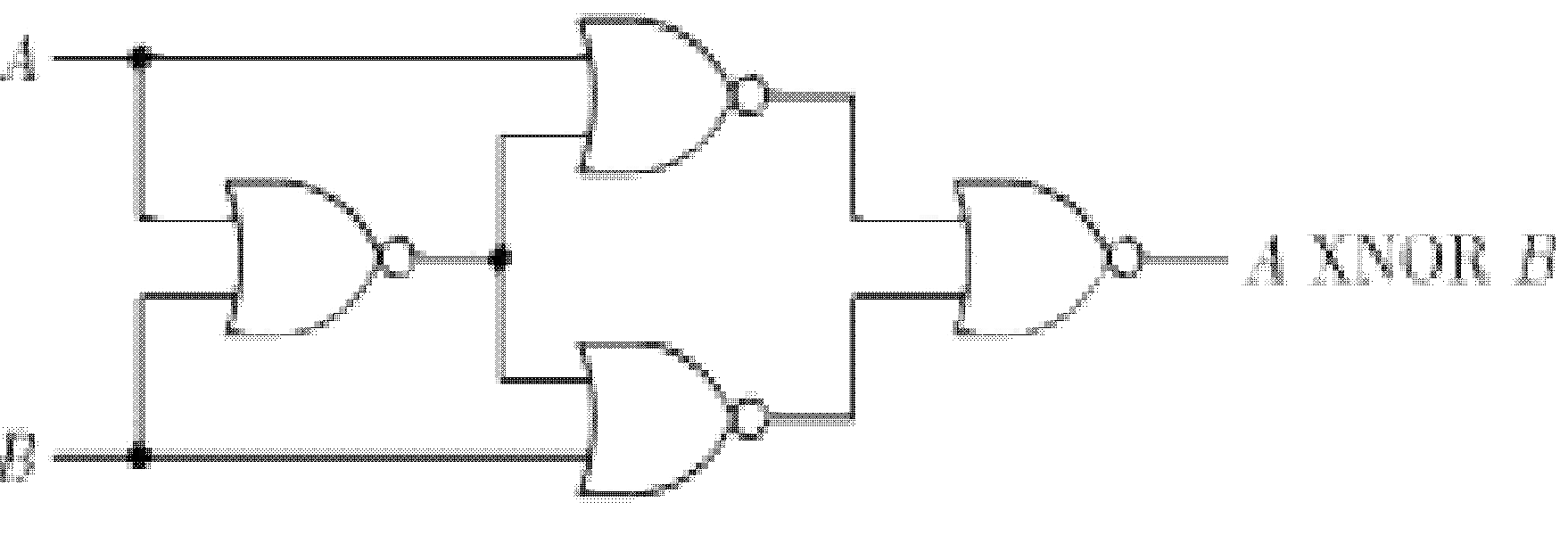
Inputs are **A** and **B** Output **Q = A B + AB**



1. **EX-NOR gate From NOR gate**

Inputs are **A** and **B**

Output **Q = A B + A B**



**THRESHOLD LOGIC**:-

### INTRODUCTION:-

* The threshold element, also called the threshold gate (T-gate) is a much more powerful device than any of the conventional logic gates such as NAND, NOR and others.
* Complex, large Boolean functions can be realized using much fewer threshold gates.
* Frequently a single threshold gate can realize a very complex function which otherwise might require a large number of conventional gates.
* T-gate offers incomparably economical realization; it has not found extensive use with the digital system designers mainly because of the following limitations.
  1. It is very sensitive to parameter variations.
  2. It is difficult to fabricate it in IC form.

# BOOLEAN ALGEBRA

### INTRODUCTION:-

* Switching circuits are also called logic circuits, gates circuits and digital circuits.
* Switching algebra is also called Boolean algebra.
* Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of elements (0,1), two binary operators called OR and AND and unary operator called NOT.
* It is the basic mathematical tool in the analysis and synthesis of switching circuits.
* It is a way to express logic functions algebraically.
* Any complex logic can be expressed by a Boolean function.
* The Boolean algebra is governed by certain well developed rules and laws.

### AXIOMS AND LAWS OF BOOLEAN ALGEBRA:-

Axioms or postulates of Boolean algebra are set of logical expressions that are accepted without proof and upon which we can build a set of useful theorems. Actually, axioms are nothing more than the definitions of the three basic logic operations AND, OR and INVERTER. Each axiom can be interpreted as the outcome of an operation performed by a logic gate.

|  |  |  |
| --- | --- | --- |
| **AND operation** | **OR operation** | **NOT operation** |
| Axiom 1: 0 . 0 = 0 | Axiom 5: 0 + 0 = 0 | Axiom 9: 1 = 0 |
| Axiom 2: 0 . 1 = 0  Axiom 3: 1 . 0 = 0  Axiom 2: 1 . 1 = 1 | Axiom 6: 0 + 1 = 1  Axiom 7: 1 + 0 = 1  Axiom 8: 1 + 1 = 1 | Axiom 10:0 = 1 |
| **1. Complementation Laws:-** |  |  |

The term complement simply means to invert, i.e. to changes 0s to 1s and 1s to 0s. The five laws of complementation are as follows:

**Law 1:** 0 = 1

**Law 2:** 1 = 0 **Law 3:** if A = 0, then A = 1 **Law 4:** if A = 1,thenA = 0

**Law 5:** A = 0 (double complementation law)

##### OR Laws:-

The four OR laws are as follows **Law 1:** A + 0 = 0(Null law)

**Law 2:** A + 1 = 1(Identity law) **Law 3:** A + A = A

**Law 4:** A +A = 1

##### AND Laws:-

The four AND laws are as follows **Law 1:** A . 0 = 0(Null law) **Law 2:** A . 1 = 1(Identity law) **Law 3:** A . A = A

**Law 4:** A .A = 0

##### Commutative Laws:-

Commutative laws allow change in position of AND or OR variables. There are two commutative laws.

**Law 1:** A + B = B + A

##### Proof

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

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

**=**

**Law 2:** A . B = B . A

##### Proof

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A . B** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| **B** | **A** | **B. A** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**=**

This law can be extended to any number of variables. For example

A.B. C = B. C. A = C. A. B = B. A. C

##### Associative Laws:-

The associative laws allow grouping of variables. There are 2 associative laws.

**Law 1:** (A + B) + C = A + (B + C)

##### Proof

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **B+C** | **A+(B+C)** |
| 0 | 0 | 0 | 0 | 0 |
| 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 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **A+B** | **(A+B)+C** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**=**

**Law 2:** (A .B) C = A (B .C)

##### Proof

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **B.C** | **A(B.C)** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **AB** | **(AB)C** |
| 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 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**=**

This law can be extended to any number of variables. For example A(BCD) = (ABC)D = (AB) (CD)

##### Distributive Laws:-

The distributive laws allow factoring or multiplying out of expressions. There are two distributive laws.

**Law 1:** A (B + C) = AB + AC

##### Proof

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **AB** | **AC** | **A+(B+C)** |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **B+C** | **A(B+C)** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**=**

**Law 2:** A + BC = (A+B) (A+C) **Proof RHS** = (A+B) (A+C)

= AA + AC + BA + BC

= A + AC + AB + BC

= A (1+ C + B) + BC

= A. 1 + BC ( 1 +C + B = 1 + B = 1 )

= A + BC

= **LHS**

1. **Redundant Literal Rule (RLR):- Law 1:** A + AB = A + B

##### Proof

**Proof**

A + AB = (A + A) (A + B)

= 1. (A + B)

= A +B

**Law 2:** A(A + B) = AB A(A + B) = AA + AB

= 0 + AB

= AB

1. **Idempotence Laws:-** Idempotence means same value.

**Law 1:** A**.** A = A

##### Proof

If A = 0, then A**.** A = 0**.** 0 =0 = A If A = 1, then A**.** A = 1**.** 1 = 1 = A

This law states that AND of a variable with itself is equal to that variable only.

**Law 2:** A + A = A

##### Proof

If A = 0, then A + A = 0 + 0 = 0 = A If A = 1, then A + A = 1 + 1 = 1 = A

This law states that OR of a variable with itself is equal to that variable only.

1. **Absorption Laws:-** There are two laws:

**Law 1:** A + A **∙** B = A

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

##### Proof

**Proof**

A + A **∙** B = A (1 + B) = A **∙** 1 = A

**Law 2:** A ( A + B) = A

A ( A + B) = A ∙ A + A **∙** B = A + AB = A(1 + B) = A **∙** 1 = A

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

##### Consensus Theorem (Included Factor Theorem):- Theorem 1:

AB +AC + BC = AB +AC

##### Proof

**Theorem 2:**

**LHS =** AB + AC + BC

= AB + AC + BC (A+A)

= AB + AC + BCA + BCA

= AB (1 + C) + AC (1+ B)

=AB (1) +AC (1)

= AB + AC

= **RHS**

##### Proof

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

LHS = (A + B) (A + C) (B + C)

= (AA + AC + BA + BC) (B + C)

= (AC + BC +AB) (B + C)

= ABC + BC + AB + AC + BC+ABC

= AC + BC +AB

RHS= (A + B) (A+C)

= AA + AC + BC +AB

= AC + BC +AB

= LHS

##### Transposition Theorem:- Theorem:

AB + AC = (A + C)(A + B)

##### Proof

**RHS=** (A + C) (A + B)

= AA + CA + AB + CB

= 0 +AC + AB + BC

= AC + AB + BC ( A+A)

= AB + ABC + AC +ABC

= AB + AC

= LHS

##### De Morgan’s Theorem:-

De Morgan’s theorem represents two laws in Boolean algebra. **Law 1:** A + B =A**∙** B

##### Proof

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

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

**=**

This law states that the complement of a sum of variables is equal to the product of their individual complements.

**Law 2:** A**∙** B = A + B

##### Proof

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **A . B** | **A . B** |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

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

**=**

This law states that the complement of a product of variables is equal to the sum of their individual complements.

### DUALITY:-

The implication of the duality concept is that once a theorem or statement is proved, the dual also thus stand proved. This is called the principle of duality.

[f (A, B, C,…..,0, 1, +, **∙**)]d = f( A, B, C, …., 1, 0, **∙,** +)

Relations between complement and dual

fc (A, B, C, …..) = f (A, B, C, …..) = fd (A, B, C,…)

fd (A, B, C, …..) = f (A, B, C,…) = fc ( A, B, C, …..)

The first relation states that the complement of a function f(A, B, C, …) can be obtained by complementing all the variables in the dual function fd (A, B, C, …..).

The second relation states that the dual can be obtained by complementing all the literals in f (A, B, C, ….).

##### DUALS:-

***Given expression Dual***

**1.** 0 = 1 1 = 0

**2.** 0 ∙1 = 0 1 + 0 = 1

**3.** 0 ∙0 = 0 1 + 1 = 1

**4.** 1 ∙1 = 1 0 + 0 = 0

**5.** A ∙ 0 = 0 A + 1 = 1

**6.** A ∙ 1 = A A + 0 = A

**7.** A ∙ A = A A + A = A

**8.** A ∙ A = 0 A + A = 1

**9.** A ∙ B = B ∙ A A + B = B+ A

**10.** A ∙ ( B ∙ C)=( A ∙ B) ∙ C A + ( B + C)=( A + B) + C

1. A ∙ (B + C) = AB + AC A + BC = ( A + B) (A + C)
2. A( A + B ) = A A + AB = A
3. A ∙ ( A ∙ B) = A ∙ B A + A + B = A + B
4. AB = A + B A + B = A B

1. ( A + B) ( A+ C) (B + C) = ( A+ B )(A + C) AB + AC + BC = AB + AC

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

1. (A+C)(A+B) = AB+AC AC+AB=(A+B) (A+C)

**18.** (A+B)(C+D) = AC + AD + BC + BD (AB+CD) = (A+C)(A+D)(B+C)(B+D)

**19.** A + B = AB + AB + AB AB =(A+B) (A+B) (A+B)

**20.** AB + A + AB = 0 A + B ∙ A ∙ (A + B) = 1

### SUM - OF - PRODUCTS FORM:-

* + This is also called disjunctive Canonical Form (DCF) or Expanded Sum of Products Form or Canonical Sum of Products Form.
  + In this form, the function is the sum of a number of products terms where each product term contains all variables of the function either in complemented or uncomplemented form.
  + This can also be derived from the truth table by finding the sum of all the terms that corresponds to those combinations for which ‘f ’ assumes the value 1.

For example

f( A, B, C) = AB + BC

= AB (C + C) + BC (A + A)

= A BC + ABC + ABC + ABC

* + The product term which contains all the variables of the functions either in complemented or uncomplemented form is called a minterm.
  + The minterm is denoted as mo, m1, m2 … .
  + An ‘n’ variable function can have 2n minterms.
  + Another way of representing the function in canonical SOP form is the showing the sum of minterms for which the function equals to 1.

For example

f ( A, B, C) = m1 + m2+ m3 + m5

or

f (A, B, C) =∑ m (1, 2, 3, 5)

where ∑m represents the sum of all the minterms whose decimal codes are given the parenthesis.

### PRODUCT- OF - SUMS FORM:-

* + This form is also called as Conjunctive Canonical Form ( CCF) or Expanded Product - of – Sums Form or Canonical Product Of Sums Form.
  + This is by considering the combinations for which f = 0
  + Each term is a sum of all the variables.
  + The function f (A, B, C) = ( A + B + C∙C) + ( A + B + C∙C)

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

* + The sum term which contains each of the ‘n’ variables in either complemented or uncomplemented form is called a maxterm.
  + Maxterm is represented as M0, M1, M2, …….

Thus CCF of ‘f’ may be written as f( A, B, C)= M0 ∙ M4 ∙ M6∙ M7

or

f(A, B, C) = ( 0, 4, 6, 7)

Where represented the product of all maxterms.

### CONVERSION BETWEEN CANONICAL FORM:-

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.

Example:-

f(A, B, C) = ∑m( 0,2,4,6,7)

This has a complement that can be expressed as

f (A, B, C) =∑ m(1, 3, 5) = m1 + m3 + m5

If we complement f by De- Morgan’s theorem we obtain ‘f’ in a form. f =(m1+ m3 + m5) = m1. m3. m5

= M1 M3 M5 =∏ M(1, 3 ,5)

##### Example:-

**Expand A (A + B) (A + B + C) to maxterms and minterms.**

##### Solution:-

In POS form

A( A + B) (A + B + C) A = A + B B + CC

= (A + B) ( A +B) + C∙C

= (A + B + CC) (A + B + C C)

= (A + B + C) (A + B +C) (A + B + C) (A + B + C) A + B = A + B + C∙C

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

Therefore

A( A + B)(A + B + C)

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

= (000) (001) (010) (011) (100) (101)

= M0 ∙ M1 ∙ M2 ∙ M3 ∙ M4 ∙ M5

=∏ M( 0, 1, 2, 3, 4,5)

The maxterms M6 and M7 are missing in the POS form. So, the SOP form will contain the minterms 6 and 7

### KARNAUGH MAP OR K- MAP:-

* + The K- map is a chart or a graph, composed of an arrangement of adjacent cells, each representing a

particular in

sum or product form.

combination of variables

* + The K- map is systematic method of simplifying the Boolean expression.

### TWO VARIABLE K- MAP:-

A two variable expression can have 22 = 4 possible combinations of the input variables A and B.

##### Mapping of SOP Expression:-

* + The 2 variable K-map has 22 = 4 squares. These squares are called cells.
  + A ‘1’ is placed in any square indicates that corresponding minterm is included in the output expression, and a 0 or no entry in any square indicates that the corresponding minterm does not appear in the expression for output.

##### B

0 1

0

##### A

|  |  |
| --- | --- |
| A B | A B |
| A B | A B |

1

##### Example:-

##### Map expression f= AB + AB Solution:-

The expression minterms is F = m1 + m2 = m( 1, 2)

##### B

0 1

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 1 | 1 |
|  | 2 |  | 3 |
| 1 |  | 0 |  |

0

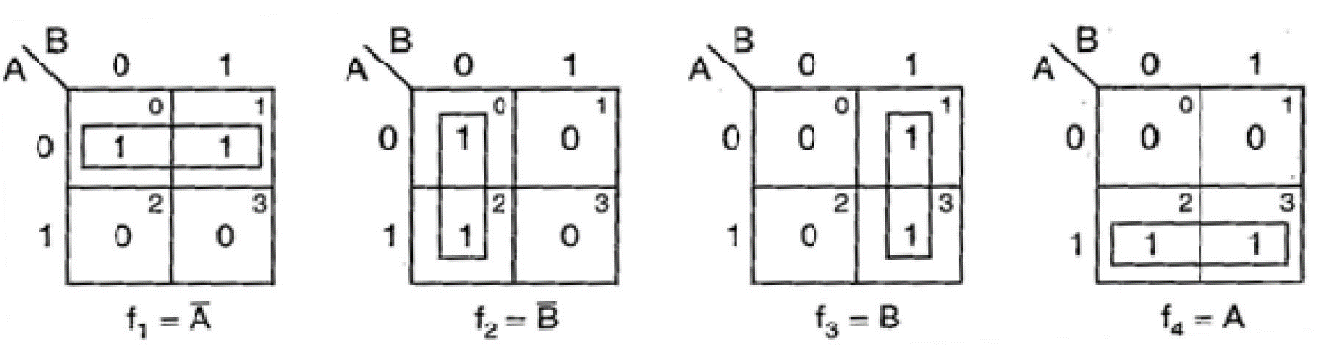
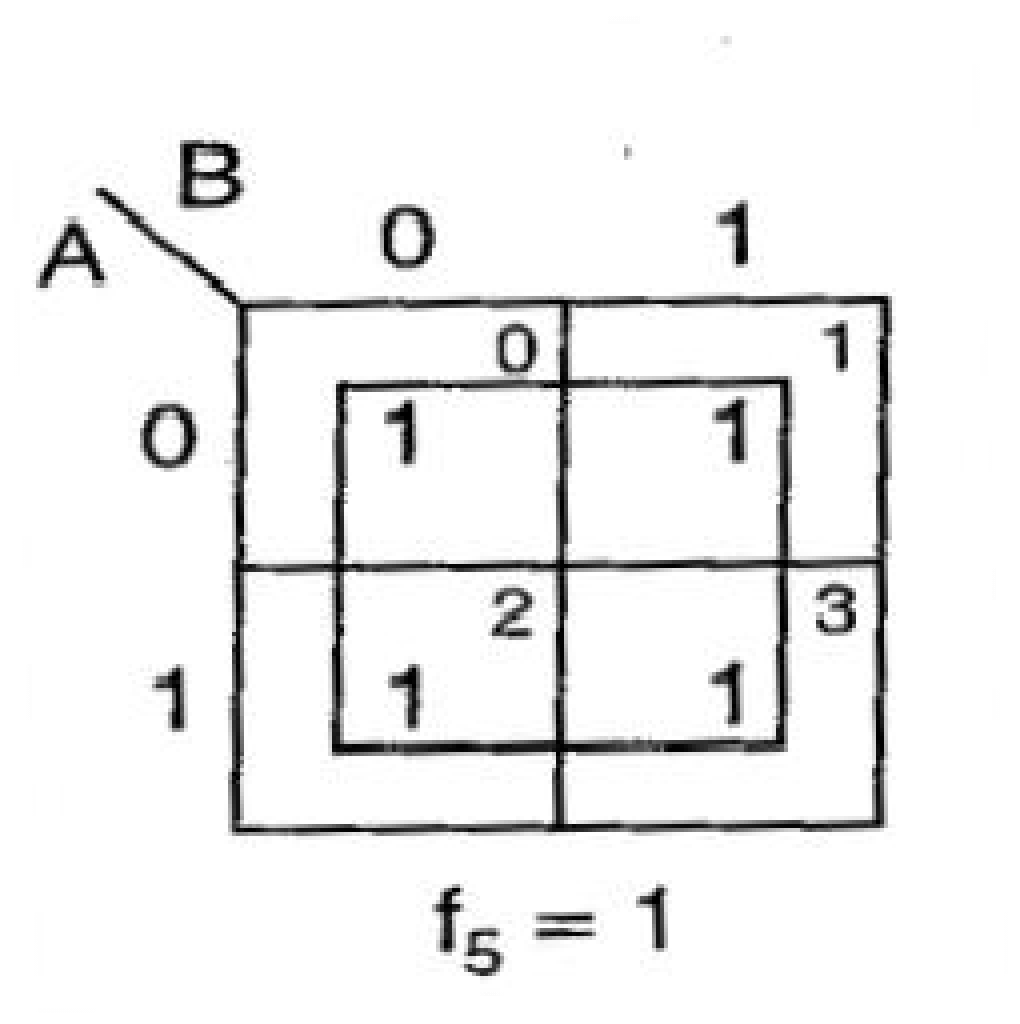
##### A

1

##### Minimization of SOP Expression:-

To minimize a Boolean expression given in the SOP form by using K- map, the adjacent squares having 1s, that is minterms adjacent to each other are combined to form larger squares to eliminate some variables.

The possible minterm grouping in a two variable K- map are shown below



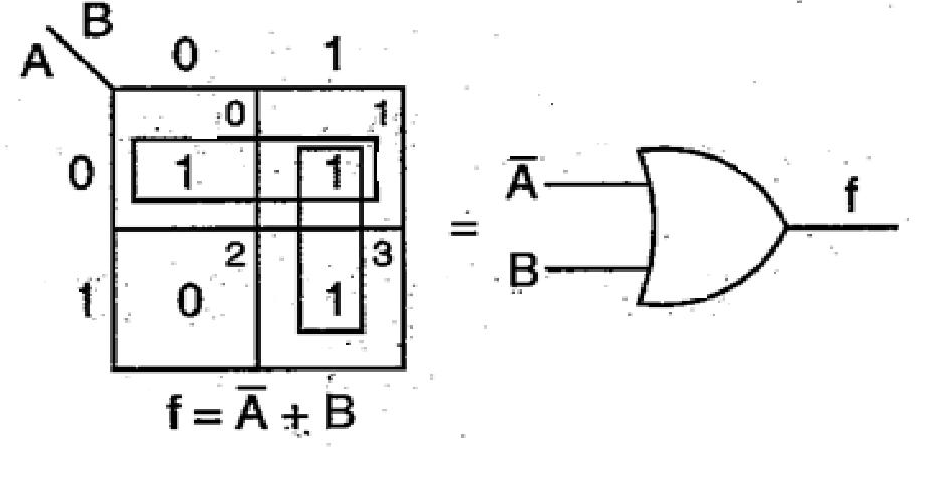
* + Two minterms, which are adjacent to each other, can be combined to form a bigger square called 2 – square or a pair. This eliminates one variable that is not common to both the minterms.
  + Two 2-squares adjacent to each other can be combined to form a 4- square. A 4- square eliminates 2 variables. A 4-square is called a quad.
  + Consider only those variables which remain constant throughout the square, and ignore the variables which are varying. The non-complemented variable is the variable remaining constant as 1.The complemented variable is the variable remaining constant as a 0 and the variables are written as a product term.

##### Example:-

Reduce the expression f= AB + A B + AB using mapping.

##### Solution:-

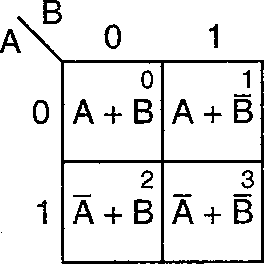
Expressed in terms of minterms, the given expression is f = m0 + m1 + m3 = ∑m ( 0, 1, 3)



##### F = A + B

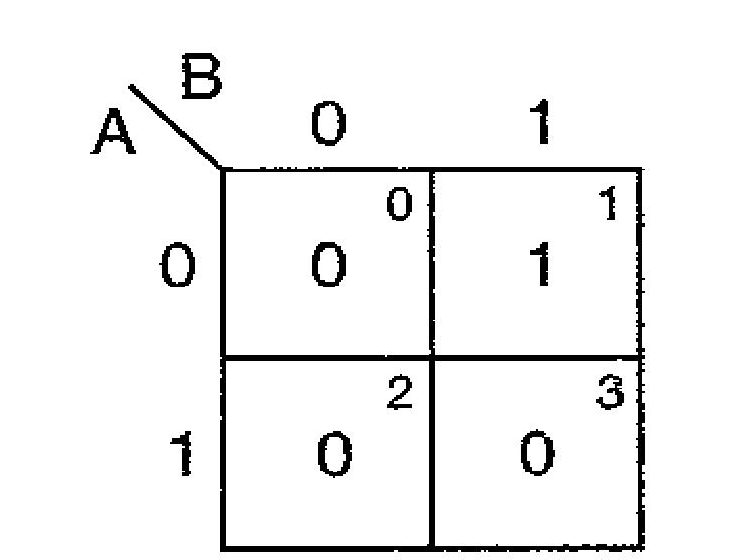
**Mapping of POS Expression:-**

Each sum term in the standard POS expression is called a Maxterm. A function in two variables (A,B) has 4 possible maxterms, A + B, A + B, A + B and A + B . They are represented as M0, M1, M2 and M3 respectively.



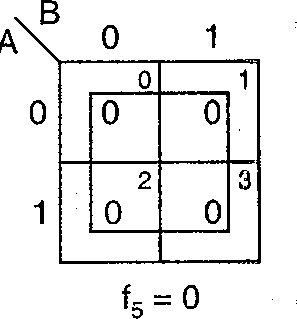
##### The maxterm of a two variable K-map Example:-

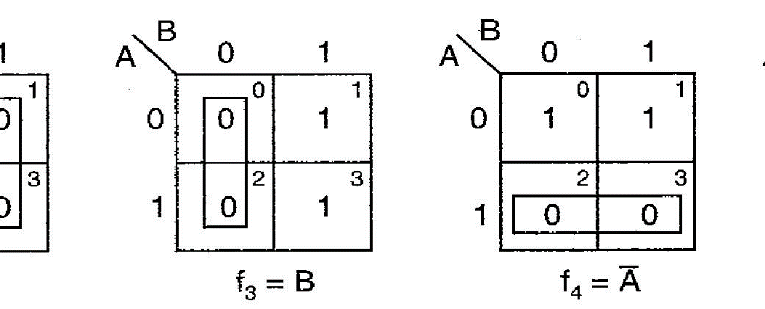
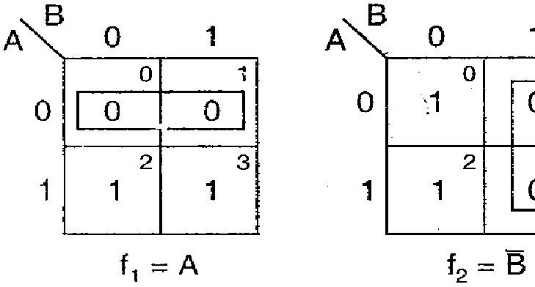
##### Plot the expression f= (A + B)(A + B )(A + B) Solution:-

Expression interms of maxterms is f = πM (0, 2, 3)

##### Minimization of POS Expressions:-

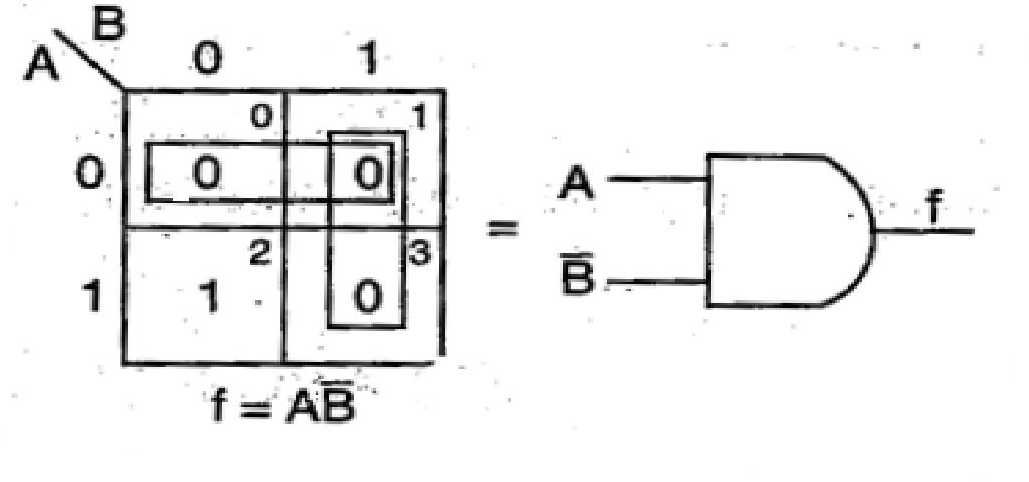
In POS form the adjacent 0s are combined into large square as possible. If the squares having complemented variable then the value remain constant as a 1 and the non-complemented variable if its value remains constant as a 0 along the entire square and then their sum term is written.

The possible maxterms grouping in a two variable K-map are shown below



##### Example:-

**Reduce the expression f = (A + B)(A + B)(A +B ) using mapping Solution:-**

The given expression in terms of maxterms is f = πM (0, 1, 3)

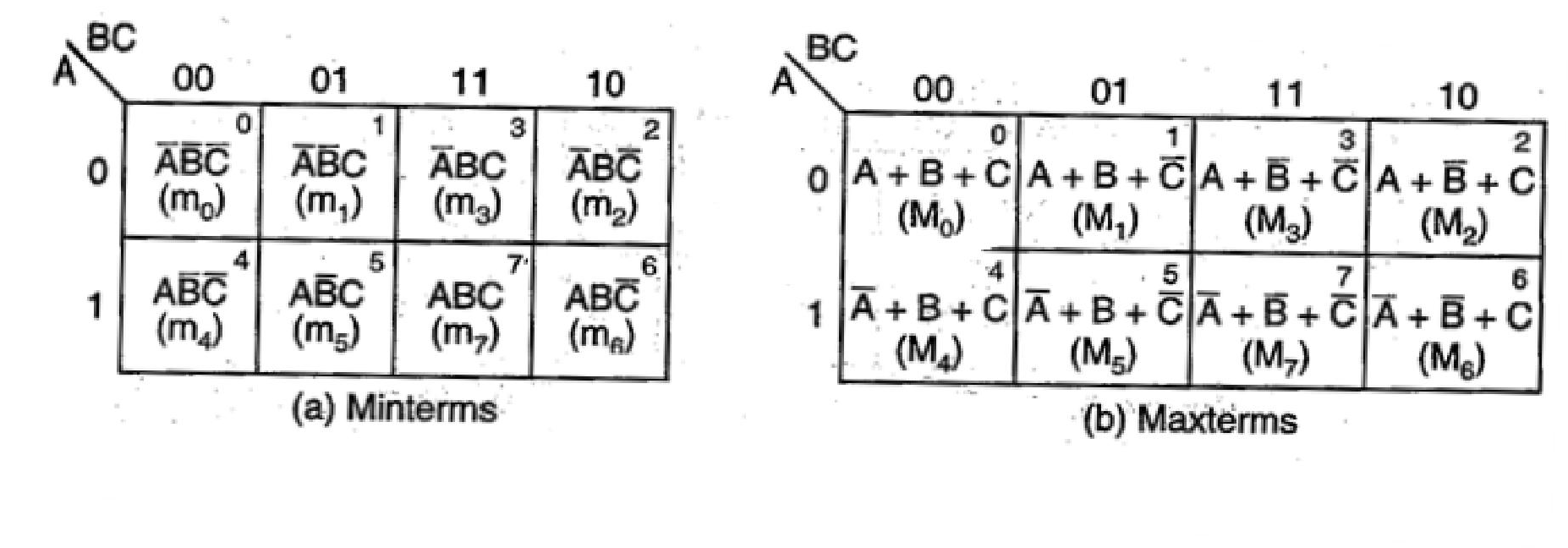
### THREE VARIABLE K- MAP:-

A function in three variables (A, B,

C) can be expressed in SOP and POS form having eight possible

combination. A three variable K- map have 8 squares or cells and each square minterm or maxterm is shown in the figure below.

on the map represents a



##### Example:-

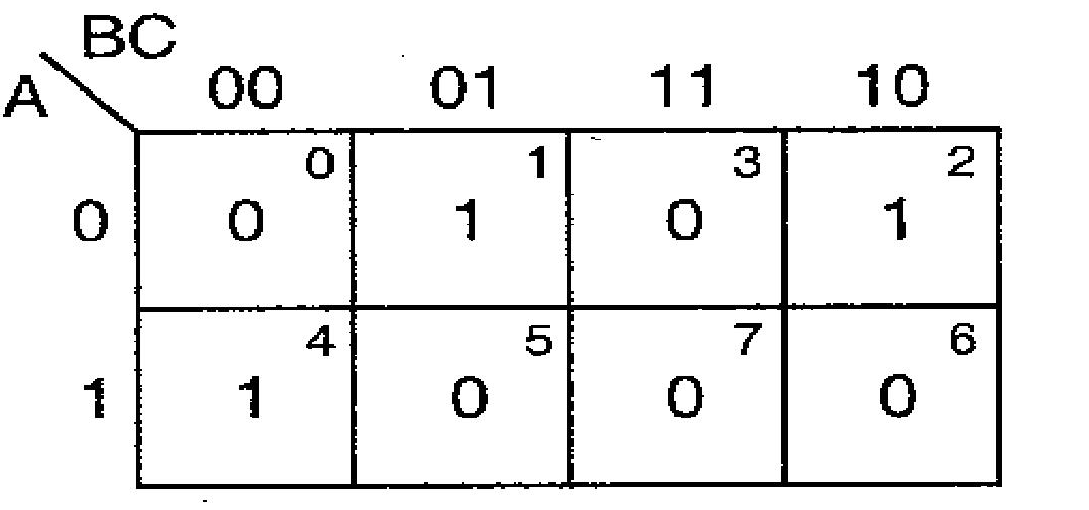
##### Map the expression f = ABC+ABC + ABC + ABC +ABC Solution:-

So in the SOP form the expression is f = ∑ m (1, 5, 2, 6, 7)

##### Example:-

##### Map the expression f = (A + B + C) (A + B+C) (A + B + C) (A + B + C) (A + B + C) Solution:-

So in the POS form the expression is f = π M (0, 5, 7, 3, 6)



### Minimization of SOP and POS Expressions:-

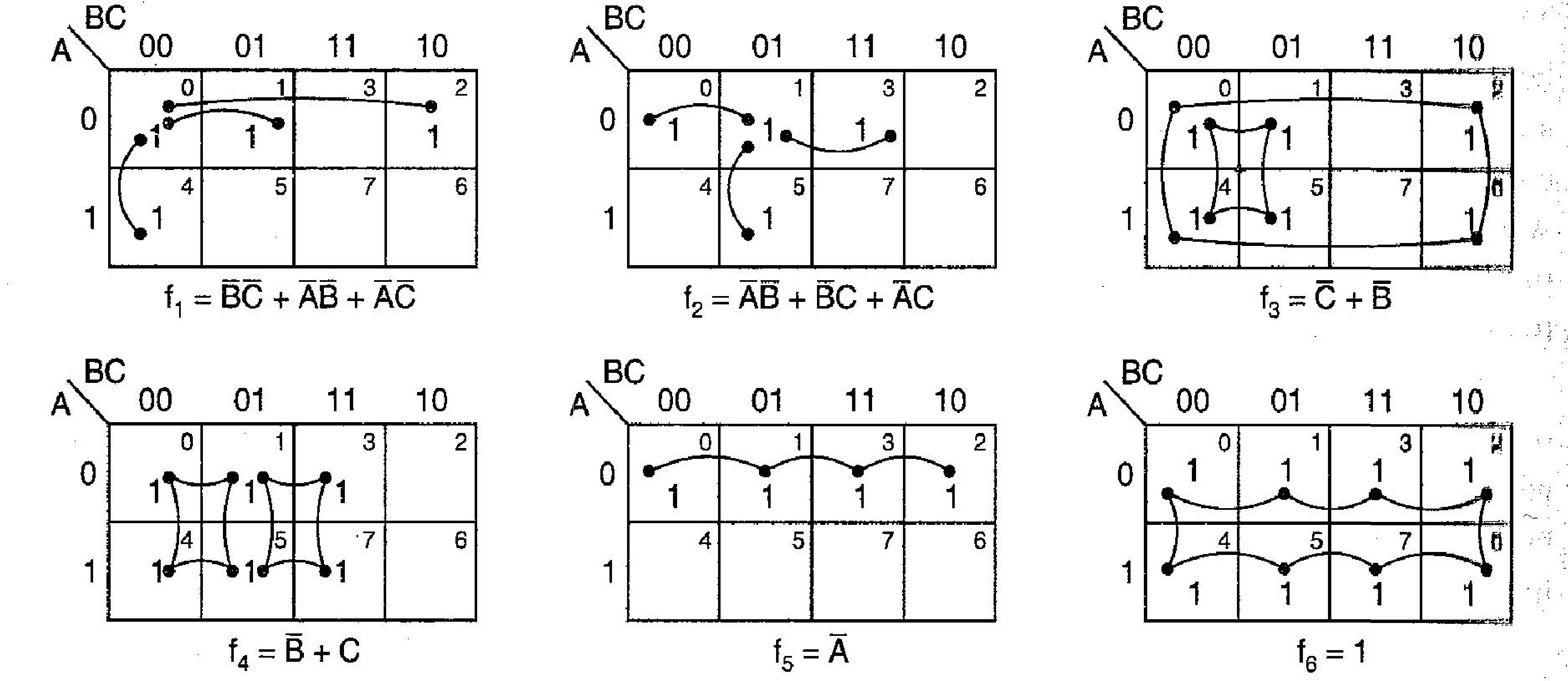
For reducing the Boolean expressions in SOP (POS) form the following steps are given below

* Draw the K-map and place expression.

1s (0s) corresponding to the minterms (maxterms) of the SOP (POS)

* In the map 1s (0s) which are not adjacent to any other 1(0) are the isolated minterms (maxterms). They are to be read as they are because they cannot be combined even into a 2-square.
* For those 1s (0s) which are adjacent to only one other 1(0) make them pairs (2 squares).
* For quads (4- squares) and octet (8 squares) of adjacent 1s (0s) even if they contain some 1s (0s) which have already been combined. They must geometrically form a square or a rectangle.
* For any 1s (0s) that have not been combined yet then combine them into bigger squares if possible.
* Form the minimal expression by summing (multiplying) the product (sum) terms of all the groups.

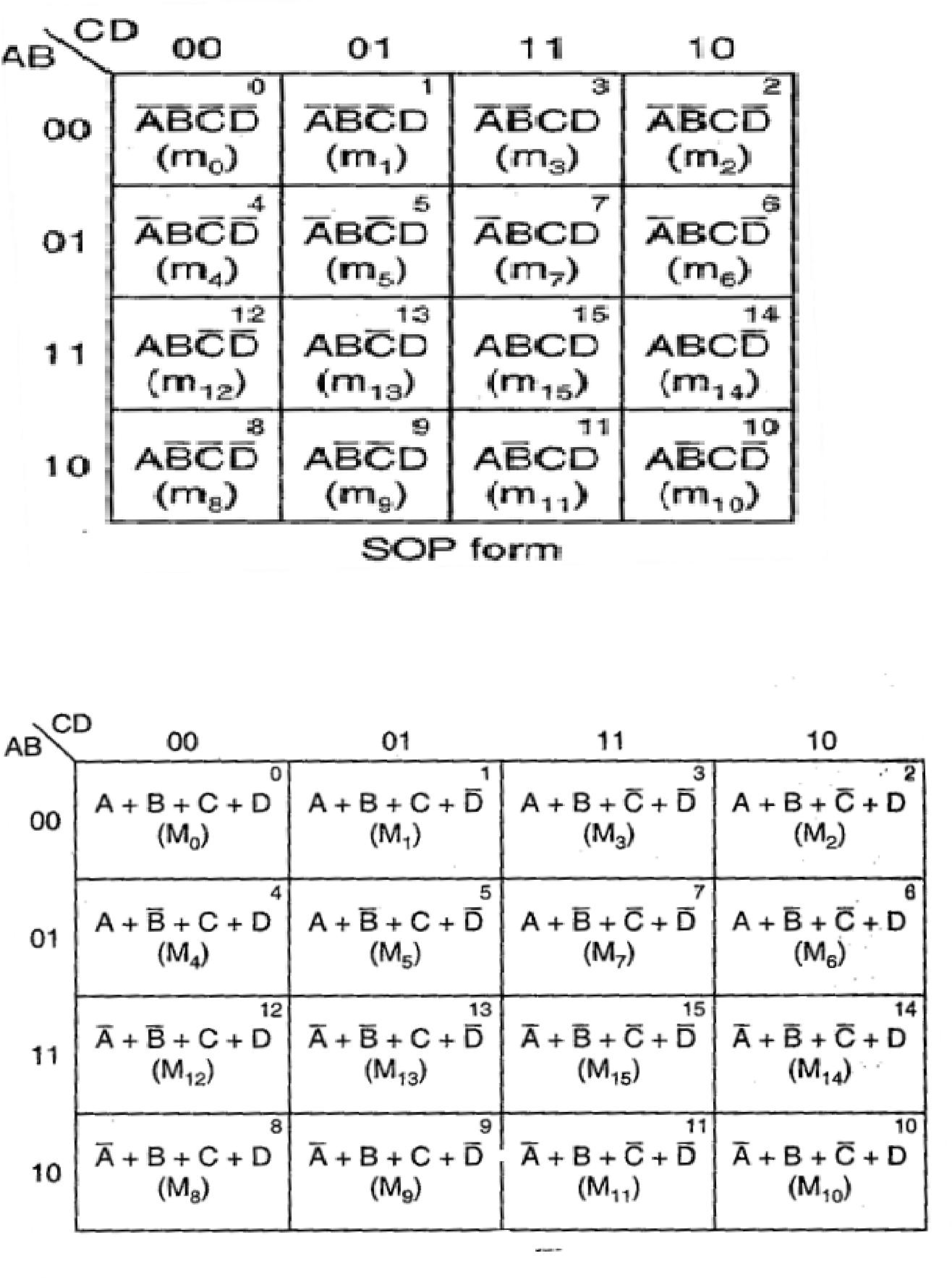
Some of the possible combinations of minterms in SOP form



These possible combinations are also for POS but 1s are replaced by 0s.

**FOUR VARIABLE K-MAP:-**

A four variable (A, B, C, D) expression can have 24 = 16 possible combinations of input variables. A four variable K-map has 24 = 16 squares or cells and each square on the map represents either a minterm or a maxterm as shown in the figure below. The binary number designations of the rows and columns are in the gray code. The binary numbers along the top of the map indicate the conditions of C and D along any column and binary numbers along left side indicate the conditions of A and B along any row. The numbers in the top right corners of the squares indicate the minterm or maxterm desginations.

**SOP FORM**

**POS FORM**

**Minimization of SOP and POS Expressions:-**

For reducing the Boolean expressions in SOP (POS) form the following steps are given below

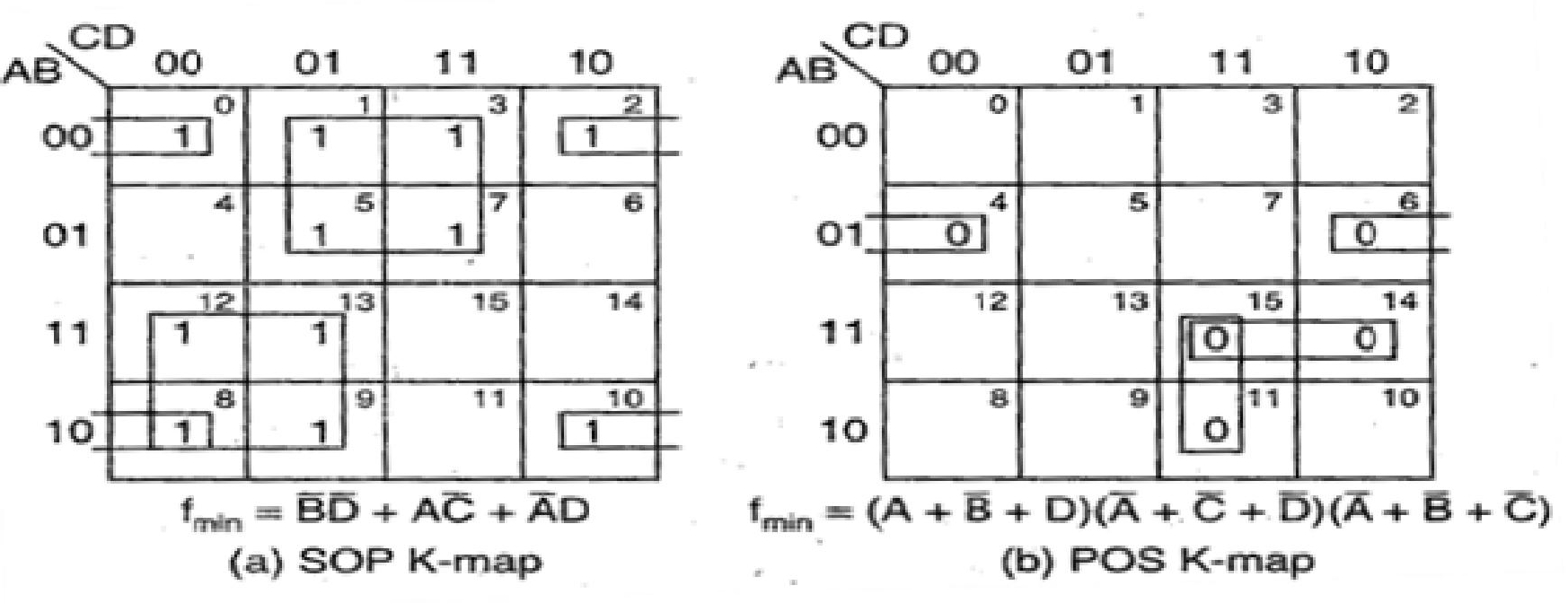
* Draw the K-map and place 1s (0s) corresponding to the minterms (maxterms) of the SOP (POS) expression.
* In the map 1s (0s) which are not adjacent to any other 1(0) are the isolated minterms (maxterms). They are to be read as they are because they cannot be combined even into a 2-square.
* For those 1s (0s) which are adjacent to only one other 1(0) make them pairs (2 squares).
* For quads (4- squares) and octet (8 squares) of adjacent 1s (0s) even if they contain some 1s (0s) which have already been combined. They must geometrically form a square or a rectangle.
* For any 1s (0s) that have not been combined yet then combine them into bigger squares if possible.
* Form the minimal expression by summing (multiplying) the product (sum) terms of all the groups.

##### Example:-

**Reduce using mapping the expression f = ∑ m (0, 1, 2, 3, 5, 7, 8, 9, 10, 12, 13) Solution:-**

The given expression in POS form is f = π M (4, 6, 11, 14, 15) and in SOP form f = ∑ m ( 0, 1, 2, 3, 5, 7, 8, 9,

10, 12, 13)



The minimal SOP expression is fmin= BD + AC + AD

The minimal POS expression is fmin =( A +B + D ) (A + C + D) (A + B + C)

## DON’T CARE COMBINATIONS:-

The combinations for which the values of the expression are not specified are called don’t care combinations or optional combinations and such expression stand incompletely specified. The output is a don’t care for these invalid combinations. The don’t care terms are denoted by d or X. During the process of designing using SOP maps, each don’t care is treated as 1 to reduce the map otherwise it is treated as 0 and left alone. During the process of designing using POS maps, each don’t care is treated as 0 to reduce the map otherwise it is treated as 1 and left alone.

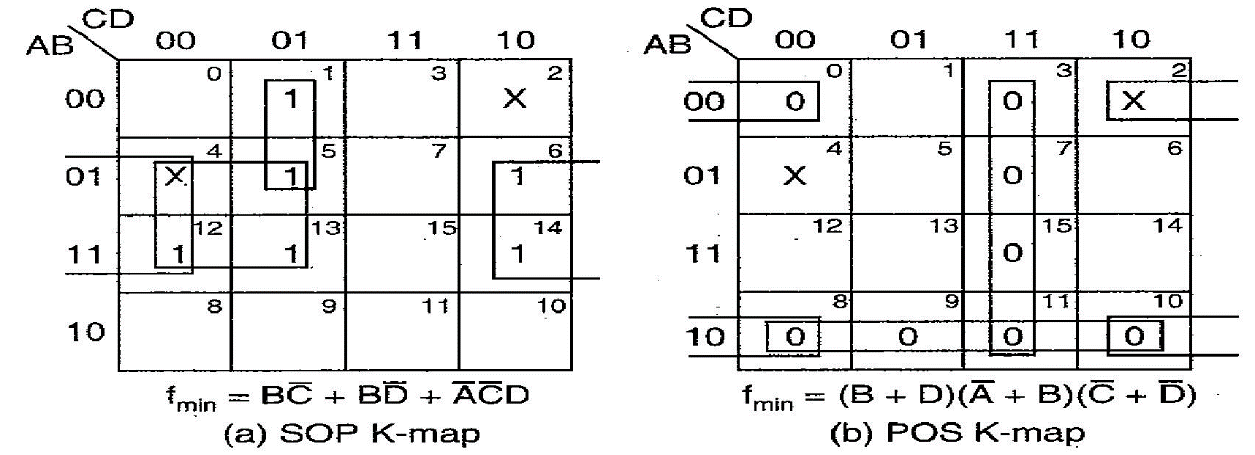
A standard SOP expression with don’t cares can be converted into standard POS form by keeping the don’t cares as they are, and the missing minterms of the SOP form are written as the maxterms of the POS form. Similarly, to convert a standard POS expression with don’t cares can be converted into standard SOP form by keeping the don’t cares as they are, and the missing maxterms of the POS form are written as the minterms of the SOP form.

##### Example:-

**Reduce the expression f = ∑ m(1, 5, 6, 12, 13, 14) + d(2, 4) using K- map. Solution:-**

The given expression in SOP form is f = ∑ m (1, 5, 6, 12, 13, 14) + d(2, 4)

The given expression in POS form is f = π M (0, 3, 7, 8, 9, 10, 11,15) + d(2, 4)



The minimal of SOP expression is fmin = BC + BD +ACD

The minimal of POS expression is fmin = (B + D)(A + B) (C + D)