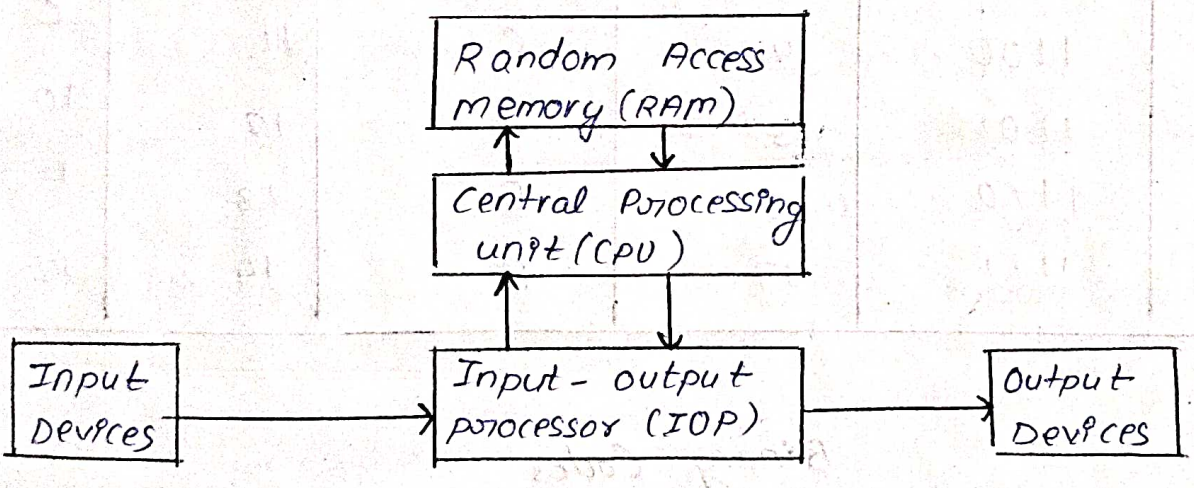
# Logic Gates and Circuit

## Digital system used in -

- Communication
- Business ~~transfer~~ transaction
- Traffic control
- Space guidance
- medical treatment
- Weather monitoring
- Internet
- Many Commercial, industrial and ~~syn~~ scientific enterprizes.




Block Diagram of Digital computer

## Number System/Representation -

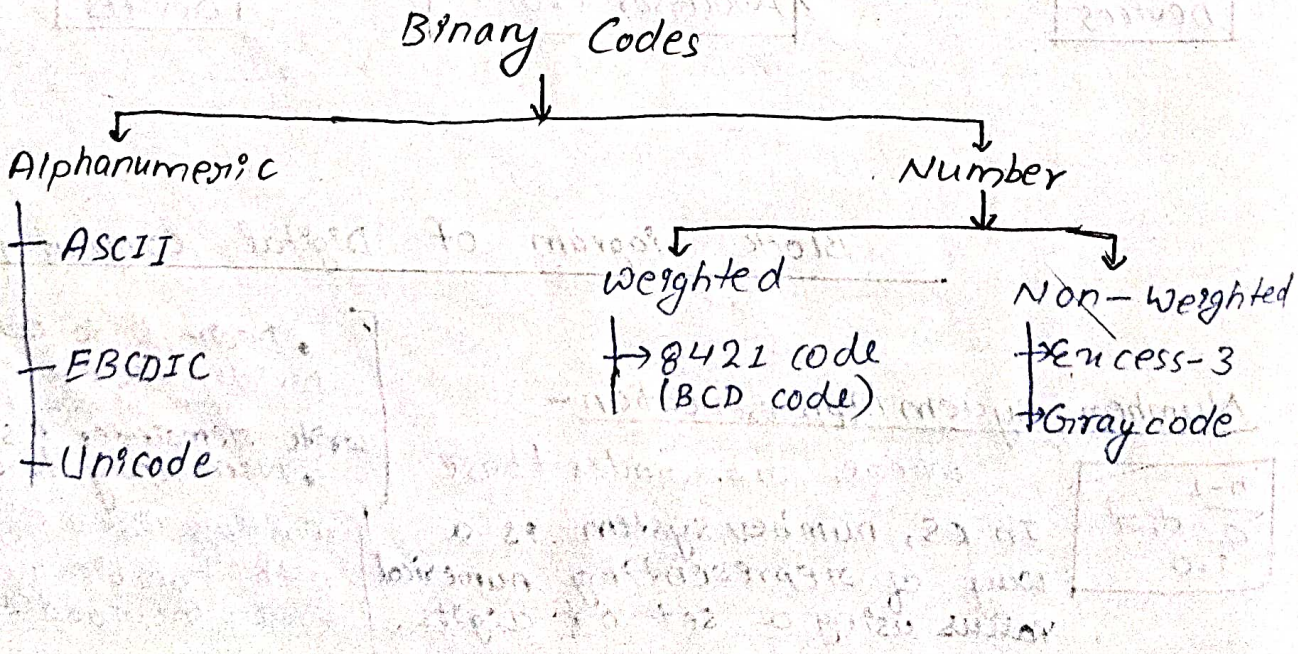
$$\sum_{i=0}^{n-1} d_i r^i$$

where, $r$ = radix / base

In CS, number system is a way of representing numerical values using a set of digits.

- Decimal to non-binary
Divide decimal no. with non-binary number & write remainder MSB to LSB.
- Non-binary to decimal
multiply each digit of no. with non-binary to the power $n$. & add the result

| Decimal ($r=10$) | Binary ($r=2$) | Octal ($r=8$) | Hexadecimal ($r=16$) | $r=11$ | $r=13$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 10 | 2 | 2 | 2 | 2 |
| 3 | 11 | 3 | 3 | 3 | 3 |
| 4 | 100 | 4 | 4 | 4 | 4 |
| 5 | 101 | 5 | 5 | 5 | 5 |
| 6 | 110 | 6 | 6 | 6 | 6 |
| 7 | 111 | 7 | 7 | 7 | 7 |
| 8 | 1000 | 10 | 8 | 8 | 8 |
| 9 | 1001 | 11 | 9 | 9 | 9 |
| 10 | 1010 | 12 | A | A | A |
| 11 | 1011 | 13 | B | 10 | B |
| 12 | 1100 | 14 | C | 11 | C |
| 13 | 1101 | 15 | D | 12 | 10 |
| 14 | 1110 | 16 | E | 13 | 11 |
| 15 | 1111 | 17 | F | 14 | 12 |



## BCD – Binary Coded Decimal :

BCD is a system of writing numerals that assigns a four-digit binary code to each 0 through 9 in decimal (base 10) number.

$33 \rightarrow (100001)_2 \rightarrow (00110011)_{BCD}$

3 3

0011 0011

$(110011)_{BCD}$

> BCD represents each decimal digit with a 4-bit binary no.

$78 \rightarrow (1001110)_2 \rightarrow$ 7 8

7 bit representation

0111 1000

$(1111000)_{BCD}$

8 bit representation

- In BCD, upto 9 input value is valid After 9 ie (1001) value is unused/invalid.

| | | |
|---|---|---|
| 0 | 00 | → Valid Input |
| 1 | 01 | |
| 2 | 10 | |
| 3 | 11 | |
| 4 | 100 | |
| 5 | 101 | |
| 6 | 110 | |
| 7 | 111 | |
| 8 | 1000 | |
| 9 | 1001 | |
| 10 | 1010 | → Unused/ Invalid Output |
| 11 | 1011 | |
| ⋮ | ⋮ | |
| | 1111 | |

### Addition in BCD –

- $2+6 \rightarrow$ 0010 + 0110 = $(1000)_2$ also $(1000)_{BCD}$ valid in BCD as sum is smaller than 9.

- $7+6 \rightarrow$ 0111 + 0110 = $(1101)_2$ → valid in Binary → Invalid in BCD (X) (Add 6 to make it valid in BCD)

1101 + 0110 = 10011 → (0001 0011) ⇒ $(10011)_{BCD}$

- $9+8 \rightarrow$ 1001
  + 1000
  $(10001)_2 \rightarrow$ valid in Binary

But for BCD add 6 (0110)

  00010001
+ 00000110
  00010111
  1 7

$9+8 = 17 = (10111)_{BCD}$

**Excess 3 (XS-3 code)** - Excess-3 code also known as the Stibitz code and a non-weighted code. In Excess-3 each decimal digit is represented by adding 3 to actual decimal value & then representing that value in binary.

Decimal $\longrightarrow$ 8-4-2-1 code (BCD) $\xrightarrow[0011]{\text{Add}}$ Excess-3

or

Decimal $\xrightarrow[3]{\text{Add}}$ 8-4-2-1 code $\longrightarrow$ Excess-3

- 5 9 2

  0101 1001 0010 (BCD)
+ 0011 0011 0011
  1000 1100 0101 (Excess-3)
  8 12 5

- 4 0 3
  + 3 3 3
  7 3 6

  0111 0011 0110 (Excess-3)

## Addition in Excess-3:-

- 8+6

  +3 +3

  11 9

  1011 + 1001

⇒ 1011
  1001
  ---
  (1)0100

→ Here, we get carry bit then add 3 to it

⇒ 0001 0100
  0011 0011
  ---
  0100 0111 ← 8+6 (In Excess-3)

  4 7
  -3 3 → To check if its correct, then minus 3
  ---
  1 4

- 4 + 1

  +3 +3

  7 4

  0111 + 0100

⇒ 0111
  0100
  ---
  [1]1011

→ No carry bit then Subtract 3 from it

⇒ 1011
  -0011
  ---
  1000 ⟷ 4+1 (Excess-3)

  8
  -3 → To check, if its correct, then subtract 3
  ---
  5

- 5 + 2
  ↓ ↓
  8 5

1000 + 0101

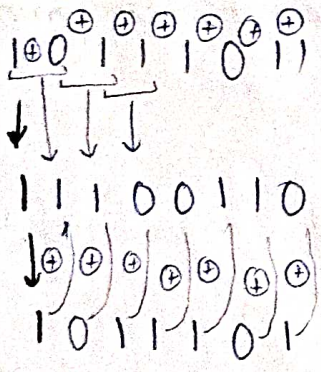
⇒ 1000
  0101
  1101
 -0011
  1010 ← 5+2 (Excess-3)

⇓

10
-3
7

(Distance code or cyclic codes), a non-weighted code

**Gray Code-** Also known as reflected binary code. It is a form of binary that uses a different method of incrementing from one number to the next.



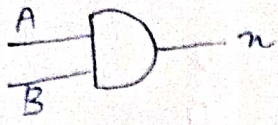

| ⊕ | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

**Logic Gates-**

It is a hardware use to design logic circuit. A device that performs a boolean function, a logical operation performed on one or more binary inputs that produces a single binary output.
Act as a building block for digital circuit.

**Gates-** Building block of a digital system and an electronic circuit that always have only one output. These gates can have one input or more than one input.

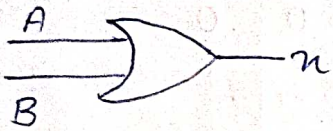
## ① AND Gate-




$x = A \cdot B$

| A | B | x |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

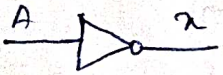
## ② OR Gate-




$x = A + B$

| A | B | x |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

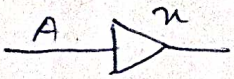
## ③ Not Gate-




$x = \bar{A}$

| A | x |
|---|---|
| 0 | 1 |
| 1 | 0 |

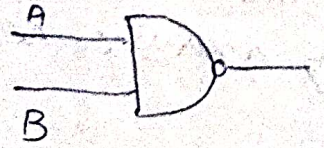
## ④ Buffer Gate-




$x = A$

| A | x |
|---|---|
| 0 | 0 |
| 1 | 1 |

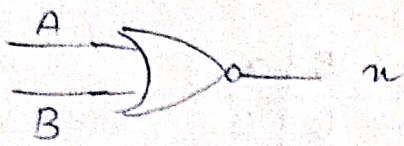
## ⑤ NAND Gate-




$x = \overline{(A \cdot B)}$

| A | B | x |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

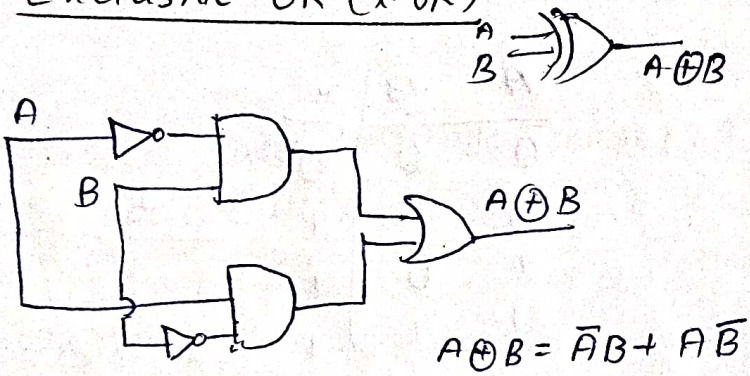
⑥ NOR Gate –




$x = (\overline{A+B})$

| A | B | x |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 0 |

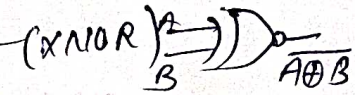
⑦ Enclusive OR (X-OR) –



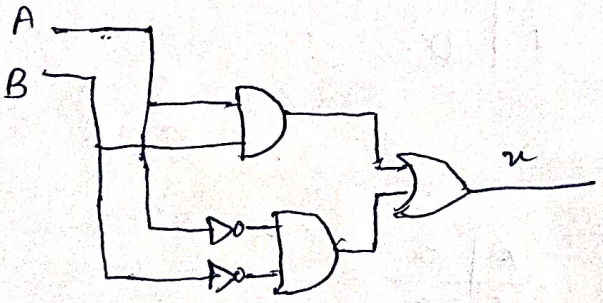

$A \oplus B = \bar{A}B + A\bar{B}$

| A | B | x |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

⑧ Enclusive NOR — (XNOR)




$\overline{A \oplus B} = AB + \bar{A}\bar{B} = A \odot B$




| A | B | x |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Universal gates →

In Boolean Algebra, the NAND & NOR gates are called universal gates because any digital circuit can be implemented by using any one of these two i.e. any logic gate can be created using NAND or NOR gates only.

# George Boole - 1854

## Boolean Theorem / Law -

① **Commutative law**

$A.B = B.A$

$A+B = B+A$

② **Associative law**

$(A.B).C = A.(B.C)$

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

③ **Distributed law**

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

④ **AND law**

$A.0 = 0$

$A.1 = A$

$A.A = A$

$A.\bar{A} = 0$

⑤ **OR Law**

$A+0 = A$

$A+1 = 1$

$A+A = A$

$A+\bar{A} = 1$

⑥ **Inversion**

$\bar{\bar{A}} = A$

⑦ $A+BC = (A+B)(A+C)$

**Demorgan's Law**

① $\overline{(A+B)} = \bar{A}.\bar{B}$

② $\overline{(A.B)} = \bar{A}+\bar{B}$

$\boxed{A+AB = A}$

$A.(1+B)$

$\Rightarrow A.1$ (1+B → 1)

$\Rightarrow A$

→ Simplification using Bodean Algebra

Q. $AB + A(B+C) + B(B+C)$

$\Rightarrow AB + AB + AC + BB + BC$ (Distributed law)

$\Rightarrow AB + AB + AC + B + BC$ $(BB = B) \rightarrow$ Rule 4

$\Rightarrow AB + AC + B + BC$ $(AB + AB = AB) \rightarrow$ Rule 5

$\Rightarrow \underline{AB + B} + AC + BC$

$\Rightarrow \underline{A + AC} + BC$

$\Rightarrow A + BC$

Q- $\overline{\overline{AB} + \overline{A} + AB}$

$\Rightarrow \overline{\overline{AB}} \cdot \overline{\overline{A}} \cdot \overline{AB}$

$\Rightarrow AB \cdot A \cdot (\overline{A} + \overline{B})$

$\Rightarrow AB \cdot (A\overline{A} + A\overline{B})$

$\Rightarrow AB \cdot (0 + A\overline{B})$

$\Rightarrow AB \cdot A\overline{B}$

$\Rightarrow AA\overline{B}\overline{B}$

$\Rightarrow 0$

Q- $(A + \overline{B} + AB)(\overline{A}B) = 0$

LHS $= (A + \overline{B} + AB)(\overline{A}B)$

$= A\overline{A}B + \overline{A}B\overline{B} + AB\overline{A}B$

$= 0.B + A.0 + 0.A$

$= 0$

$=$ RHS.

Q- $A + \overline{A}B + AB = A + B$

LHS $= A + \overline{A}B + AB$

$= A + AB + \overline{A}B$

$= B(A + \overline{A}) + \overline{A}B$

$= B.1 + A$

$= A + B$

$=$ RHS

Q- $A\overline{B} + \overline{A}B + AB + \overline{A}\overline{B}$

$\Rightarrow A\overline{B} + AB + \overline{A}B + \overline{A}\overline{B}$

$\Rightarrow A(\overline{B} + B) + \overline{A}(B + \overline{B})$

$\Rightarrow A.1 + \overline{A}.1$

$\Rightarrow A + \overline{A}$

$\Rightarrow 1$

# Circuit Designing Techniques (SOP, POS, K-Map)-

## Standard forms of Boolean Expression-

All Boolean expressions can be converted into either of two standard forms-

- Sum of Products (SOP) / Minterm {Denoted by $\Sigma m$}
- Product of Sum (POS) / Maxterm {Denoted by $\pi M$}

## SOP (Sum of Products)- [Minterm]

This form is also called Disjunction canonical form (DCF).

When two or more product terms are summed by boolean addition. In a SOP, only single bar is drawn over only one variable.

Ex→ $\bar{A}\bar{B}\bar{C}$ → Valid $\qquad$ $\overline{ABC}$ → Invalid

SOP ⇒ $AB + ABC + A\bar{C}B + \bar{A}\bar{B}\bar{C} + \bar{A}$

## Standard SOP form-

An expression in which all the variables in the domain appear in each product term in the expression.

Ex- $A\bar{B}CD + \bar{A}\bar{B}C\bar{D} + AB\bar{C}\bar{D} + ABCD$

- Standard SOP expressions are important in
  - Constructing Truth Tables
  - The Karnaugh map simplification method

- For converting Boolean expression into standard SOP form, multiply each non-standard product term by Sum of a missing variable and its complement {as $(A + \bar{A} = 1)$} and there will be no effect with multiplication by 1.

Ex- $A\bar{B}C + ABC\bar{D}$

$A\bar{B}C \Rightarrow A\bar{B}C(D + \bar{D}) \Rightarrow A\bar{B}CD + A\bar{B}C\bar{D}$

$A\bar{B}C + ABC\bar{D} \Rightarrow A\bar{B}CD + A\bar{B}C\bar{D} + ABC\bar{D}$

## POS (Product of sum) – [Maxterm] This format also called Conjunction Canonical Form (CCF)

When two or more sum terms are multiplied. In POS, single bar is drawn over only one variable.

Ex – $\bar{A}+\bar{B}+\bar{C}$ → Valid $\quad\overline{A+B+C}$ → Invalid

POS ⇒ $(\bar{A}+B)(A+\bar{B}+\bar{C})(\bar{A}+B+C)(A+\bar{C})$

## Standard POS Form –

An expression in which all the variables in the domain appear in each sum term in the expression.

Ex – $(\bar{A}+\bar{B}+C)(A+\bar{B}+C)(A+B+C)(\bar{A}+\bar{B}+\bar{C})$

- Standard POS expressions are important in
  - Constructing Truth Tables
  - K-map simplification
- For converting boolean expression into standard POS, add to each non-standard ~~product~~ sum term with multiplication of missing variable and its complement { as $A\cdot\bar{A}=0$, hence no effect by addition with 0 }

Ex – $(A+\bar{B}+C)(A+\bar{B}+\bar{C}+D)$

$A+\bar{B}+C \Rightarrow A+\bar{B}+C+D\bar{D}$

$\Rightarrow (A+\bar{B}+C+D)(A+\bar{B}+C+\bar{D})$ → Apply rule $A+BC = (A+B)(A+C)$

$(A+\bar{B}+C)(A+\bar{B}+\bar{C}+D) \Rightarrow (A+\bar{B}+C+D)(A+\bar{B}+C+\bar{D})(A+\bar{B}+\bar{C}+D)$

## Conversion (SOP / POS) –

- Standard SOP to Standard POS

SOP – $\bar{A}\bar{B}\bar{C}+\bar{A}B\bar{C}+\bar{A}BC+A\bar{B}C+ABC$

→ 000 + 010 + 011 + 101 + 111

→ ~~left one~~ As we know that there must be 8 possible combination as ($2^3=8$) there are 3 terms, so missing terms are ⇒ 001, 100 and 110

$\bar{A}\bar{B}C \rightarrow (\overline{\bar{A}\bar{B}C}) \rightarrow \bar{\bar{A}}+\bar{\bar{B}}+\bar{C} \Rightarrow (A+B+\bar{C}) \Rightarrow (1+1+0)$

→ POS = $(A+B+\bar{C})(\bar{A}+B+C)(\bar{A}+\bar{B}+C)$

- Standard POS to Standard SOP

POS = $(A+B+\bar{C})(\bar{A}+B+C)(\bar{A}+\bar{B}+C)$

→ $(1+1+0)(0+1+1)(0+0+1)$

→ missing combination :

$(1+1+1)$ $(1+0+1)$ $(1+0+0)$ $(0+1+0)$ $(0+0+0)$

↓

$(A+B+C) \rightarrow (\overline{A+B+C}) \rightarrow \bar{A}.\bar{B}.\bar{C} \rightarrow (0.00)$

Similarly,

→ SOP = $\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$

## Truth Table-

- SOP = $\bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$

| Inputs | | | Output | Product Term (Minterm) |
|---|---|---|---|---|
| A | B | C | x | |
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 1 | $\bar{A}\bar{B}C$ |
| 0 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | $A\bar{B}\bar{C}$ |
| 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 0 | |
| 1 | 1 | 1 | 1 | $ABC$ |

- POS = $(A+B+C)(A+\bar{B}+C)(A+\bar{B}+\bar{C})(\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+C)$

| Inputs | | | Output | Sum ~~Product~~ Term (maxterm) |
|---|---|---|---|---|
| A | B | C | x | |
| 0 | 0 | 0 | 0 | $(A+B+C)$ |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 0 | $(A+\bar{B}+C)$ |
| 0 | 1 | 1 | 0 | $(A+\bar{B}+\bar{C})$ |
| 1 | 0 | 0 | 1 | |
| 1 | 0 | 1 | 0 | $(\bar{A}+B+\bar{C})$ |
| 1 | 1 | 0 | 0 | $(\bar{A}+\bar{B}+C)$ |
| 1 | 1 | 1 | 1 | |

## Determining Standard Expression From Truth Table-

| I/P | | | O/P |
|---|---|---|---|
| A | B | C | x |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

SOP (four 1s) ⇒ $\bar{A}BC + A\bar{B}\bar{C} + AB\bar{C} + ABC$

011 → $\bar{A}BC$

100 → $A\bar{B}\bar{C}$

110 → $AB\bar{C}$

111 → $ABC$

POS (four 0s) ⇒ $(A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(\bar{A}+B+\bar{C})$

000 → $A+B+C$

001 → $A+B+\bar{C}$

010 → $A+\bar{B}+C$

101 → $\bar{A}+B+\bar{C}$

# The Karnaugh Map (K-Map) – Also called Veitch Diagram (KV-map).

K-Map is an array of cells in which each cell represents a binary value of the input variables. It provides a systematic method for simplifying boolean expressions and if properly used, will produce the simplest SOP or POS expression possible, known as the minimum expression.

- 3 Variable K-Map ($2^3 = 8$)

~~$\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}\bar{C}$~~

| A \ BC | 00 (0) | 01 (1) | 11 (3) | 10 (2) |
|---|---|---|---|---|
| 0 | $\bar{A}\bar{B}\bar{C}$ | $\bar{A}\bar{B}C$ | $\bar{A}BC$ | $\bar{A}B\bar{C}$ |
| 1 | $A\bar{B}\bar{C}$ | $A\bar{B}C$ | $ABC$ | $AB\bar{C}$ |

- 4 Variable K-Map ($2^4 = 16$)

| AB \ CD | 00 (0) | 01 (1) | 11 (3) | 10 (2) |
|---|---|---|---|---|
| 00 (0) | $\bar{A}\bar{B}\bar{C}\bar{D}$ | $\bar{A}\bar{B}\bar{C}D$ | $\bar{A}\bar{B}CD$ | $\bar{A}\bar{B}C\bar{D}$ |
| 01 (1) | $\bar{A}B\bar{C}\bar{D}$ | $\bar{A}B\bar{C}D$ | $\bar{A}BCD$ | $\bar{A}BC\bar{D}$ |
| 11 (3) | $AB\bar{C}\bar{D}$ | $AB\bar{C}D$ | $ABCD$ | $ABC\bar{D}$ |
| 10 (2) | $A\bar{B}\bar{C}\bar{D}$ | $A\bar{B}\bar{C}D$ | $A\bar{B}CD$ | $A\bar{B}C\bar{D}$ |

- SOP = $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + AB\bar{C} + A\bar{B}\bar{C}$

000 001 110 100

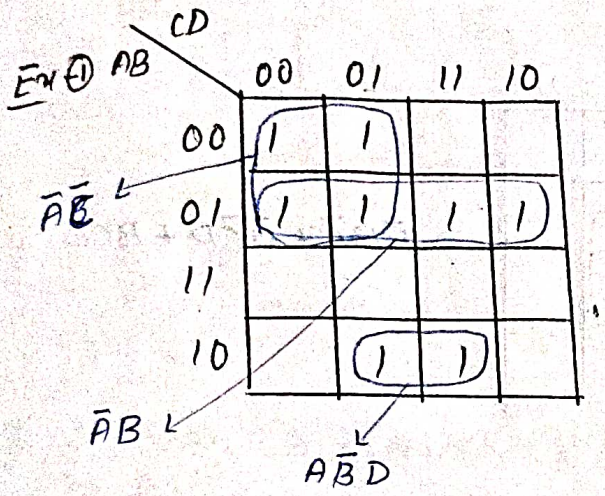
| A \ BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 1 | | 1 |
| 1 | 1 | | | |

## Grouping 1s ⇒

- You can group adjacent cells containing 1s on the K-map
- Goal is to maximize size of groups (i.e. no. of 1s) and to minimize number of groups (i.e. less pairing). In conclusion getting more 1s in less pairing.
- pairing will be of even no. of cells.

SOP

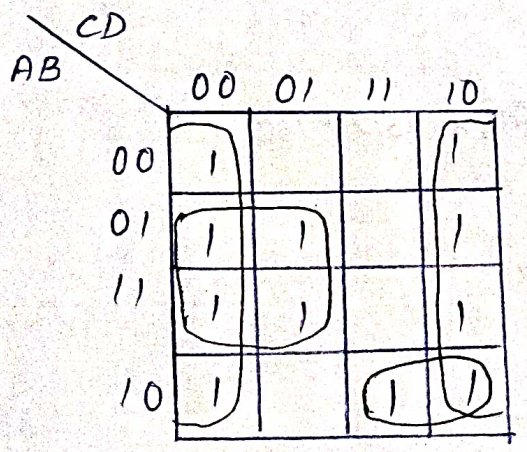
Ex ① 




minimum SOP expression
⇒ $\bar{A}\bar{C} + \bar{A}B + A\bar{B}D$

No. of pairing = 3

② $\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD$

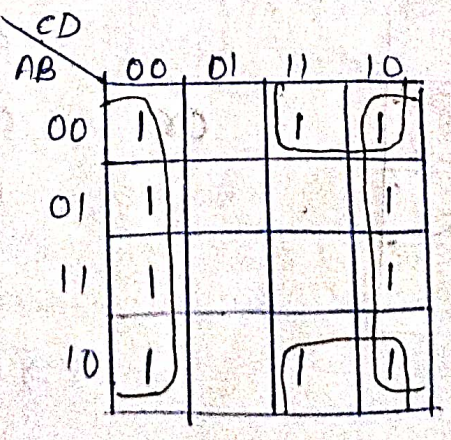



No. of pairing = 3

minimum SOP expression
⇒ $A\bar{B}C + B\bar{C} + \bar{D}$

③ $\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + \bar{A}\bar{B}CD + A\bar{B}CD + \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} + ABC\bar{D} + A\bar{B}C\bar{D}$
0100 1100 0000 1011 0010 0110 1110 1010

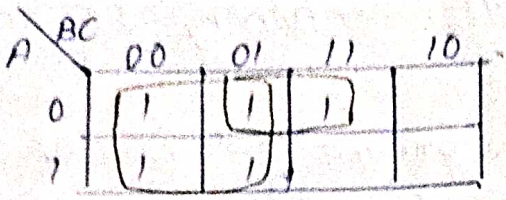
$\bar{B}\bar{C}\bar{D}(A+\bar{A}) = A\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D}$
1000 0000




⇒ $\bar{D} + \bar{B}C$

④ $A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C}$

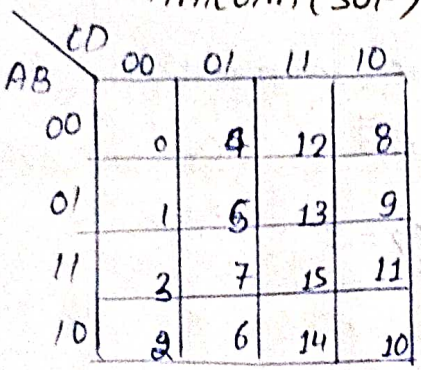
101 011 001 000 100

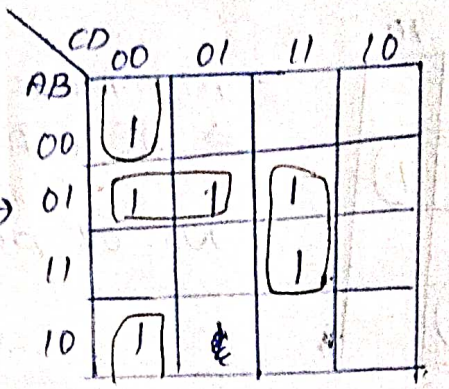



$\Rightarrow \bar{A}\bar{C} + \bar{B}$

⑤ $Y = \sum m(0,1,2,5,13,15)$

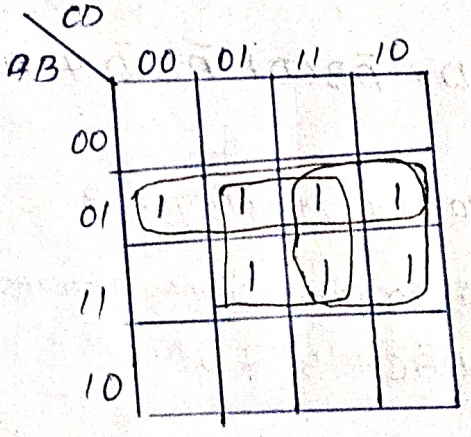
↓ minterm (SOP)



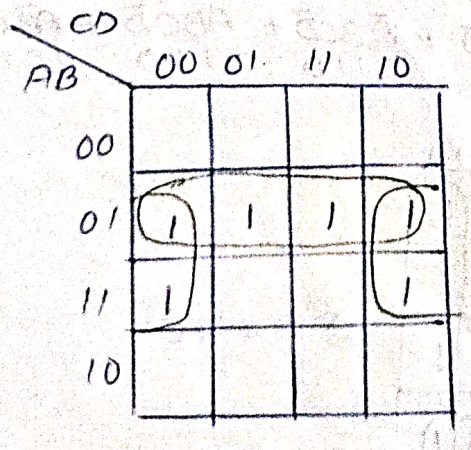

$\Rightarrow \bar{A}B\bar{C} + \bar{B}\bar{C}\bar{D} + BCD$

⑥ $Y = \sum m(1,5,7,9,11,13,15)$



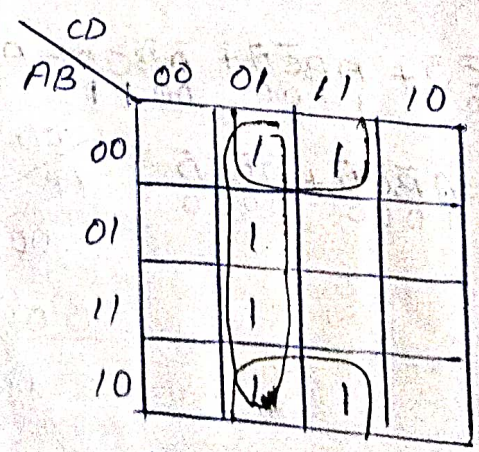

$\Rightarrow \bar{A}B + BD + BC$

⑦ $Y = \sum m(1,3,5,9,11,13)$




$\Rightarrow B\bar{D} + \bar{A}B$

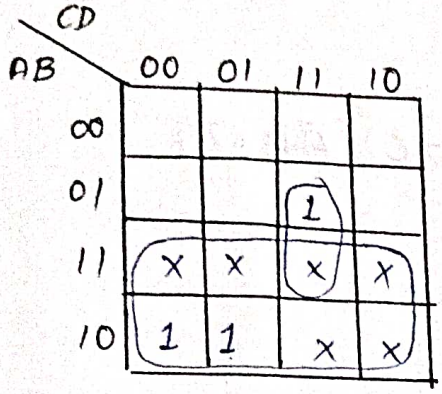
Or




$\Rightarrow \bar{C}\bar{D} + \bar{B}D$

# Don't Care Conditions –

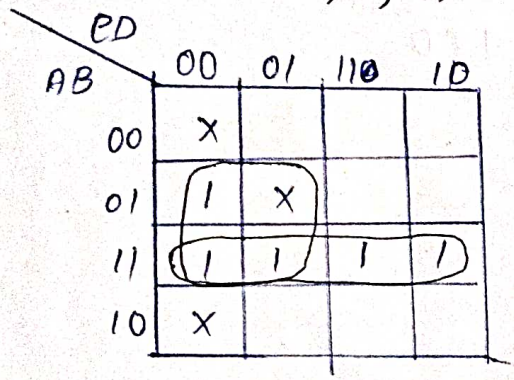
- Its symbol is 'x'.
- Use of don't care (x) in k-map is advantageous.
  - x → work as 1 for SOP
  - x → work as 0 for POS

① 


without "don't care"
$Y = A\bar{B}\bar{C} + \bar{A}BCD$

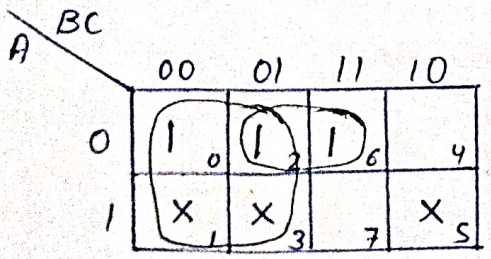
with "don't care"
$Y = A + BCD$

② $Y = \Sigma m(1,3,7,11,15) + d(0,2,5)$



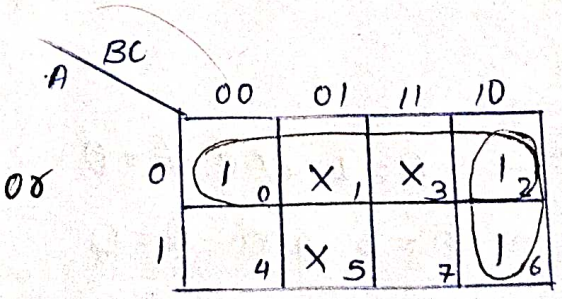

$\Rightarrow B\bar{C} + AB$

③ $F(A,B,C) = \Sigma m(0,2,6) + d(A,B,C) = \Sigma(1,3,5)$




$\Rightarrow \bar{B} + \bar{A}C$

or

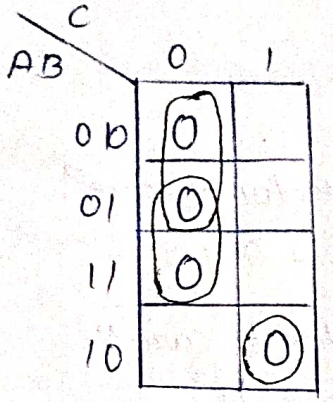



$\Rightarrow \bar{A} + B\bar{C}$

# K-Map POS minimization

① $(A+B+C)(A+\bar{B}+C)(\bar{A}+\bar{B}+C)(\bar{A}+B+\bar{C})$

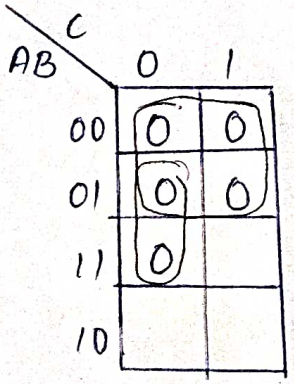
000 010 110 101




$\Rightarrow (\bar{A}+B)(\bar{B}+C)(\bar{A}B+\bar{C})$

② $(A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(A+\bar{B}+\bar{C})(\bar{A}+\bar{B}+C)$

000 001 010 011 110




$\Rightarrow (\bar{B}+C)A$

Note →

In SOP → $A=1$ and $\bar{A}=0$

In POS → $\bar{A}=1$ and $A=0$

# Unit - 2

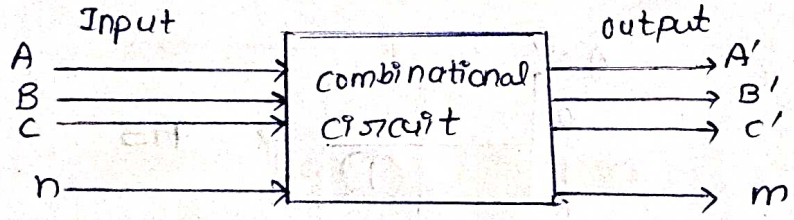
## Combinational and Sequential Building Blocks

Digital logic circuit are basically categorized in two ways-
- Combinational Circuit
- Sequential circuit

### Combinational circuit-

- A combinational circuit is a type of circuit in which output is only depended upon combination of inputs.
- No feedback is required for next output.
- No need of any memory.
- Logic gates are used as component.

Ex- Adder, subtractor, decoder, mux, demux.

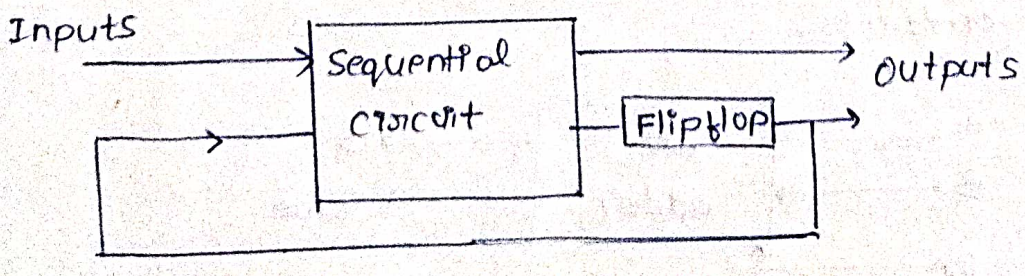



• Block diagram of combinational circuit •

### Sequential Circuit-

- It is a type of circuit in which output is depended upon present input as well as previous output.
- Feedback is required for next output.
- Need of memory for storing previous state/output.
- Logic gates + flipflop are used as component.

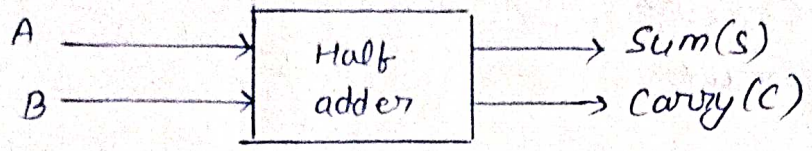
Ex- Registers, Counters.



## ↳ Half adder- (2-bit adder)

Half adder is a combinational circuit with two inputs and two outputs. It is basic building block for addition of two single bit number. Circuit has two output namely carry and sum.

① 



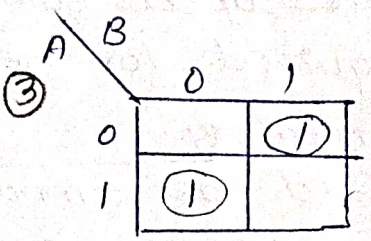

Block Diagram

② 

| Inputs | | Output | |
|---|---|---|---|
| A | B | S | Cout |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

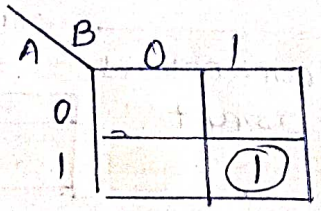
Truth Table

③ 




$\Rightarrow A\bar{B} + \bar{A}B \Rightarrow A \oplus B$

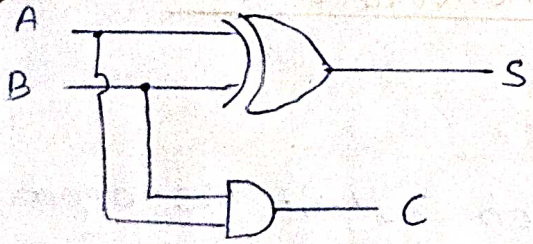
• Kmap for Sum •




$\Rightarrow AB$

• Kmap for Cout •

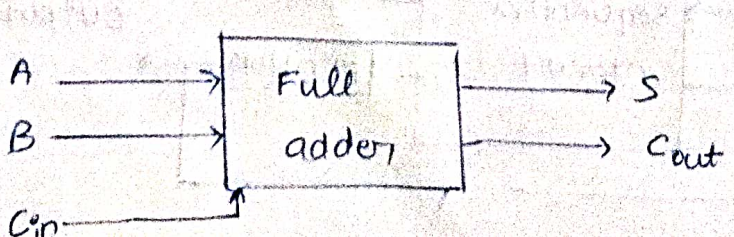
④ 




Logical diagram / Implementation of half adder / Circuit Diagram

## ↳ Full adder - (3-bit adder)

We can add 3 single bit numbers and it resulted in two outputs.

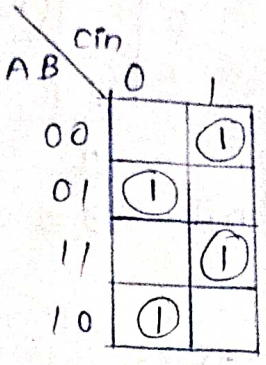


| A | B | Cin | S | $C_o$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Truth Table

Kmap for:

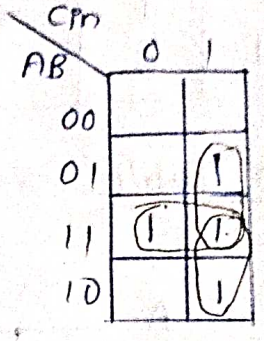
Sum




$\bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + AB\bar{C} + A\bar{B}\bar{C}_{in}$

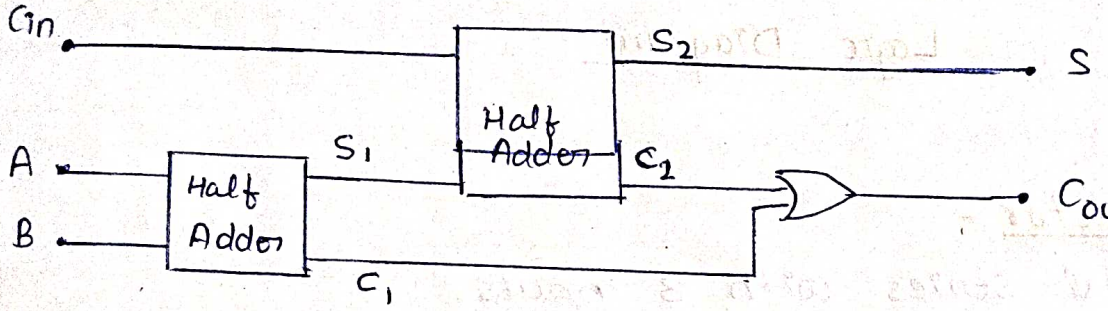
$A \oplus B \oplus C_{in}$

Carry




$AB + BC_{in} + AC_{in}$

$AB + C_{in}(A \oplus B)$



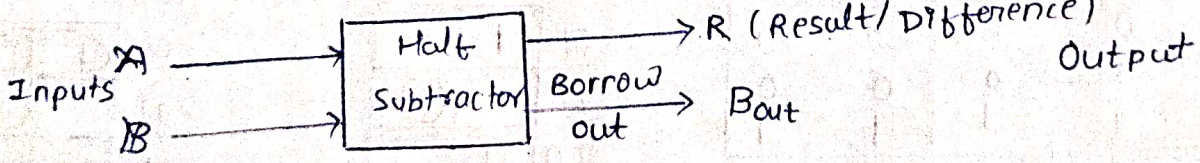

Logical diagram

---

## ↳ Half Subtractor–

Combinational circuit with two inputs and two outputs (i.e. Result/difference and borrow).

In subtraction, (A-B), A → minued and B → Subtrahend.

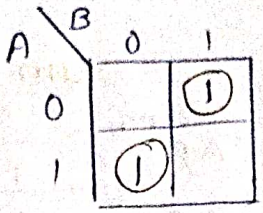



Block diagram

| A | B | R | Bout |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

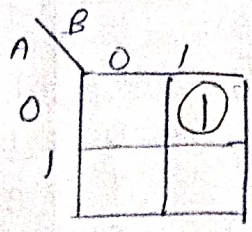
Truth Table

K-map for: Difference

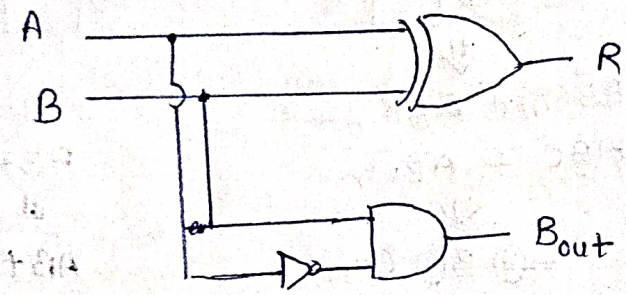



$A\bar{B} + \bar{A}B \Rightarrow A \oplus B$

Borrow



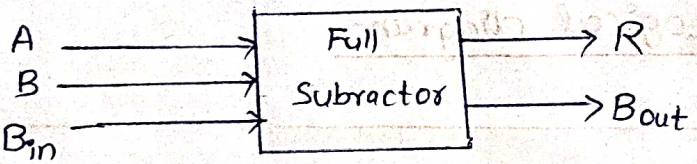

$\bar{A}B$




Logic Diagram

↳ **Full Subtractor** –

Combinational series with 3 inputs.

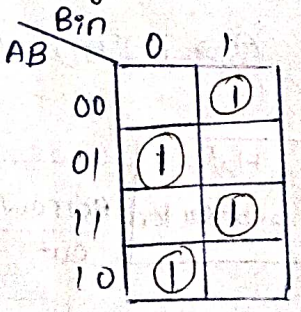



| A | B | Bin | R | Bout |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Truth Table

Kmap for:

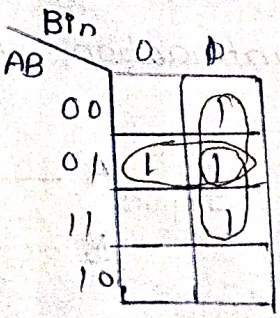
Difference




$\bar{A}\bar{B}Bin + \bar{A}B\bar{B}in + A\bar{B}\bar{B}in + ABBin$

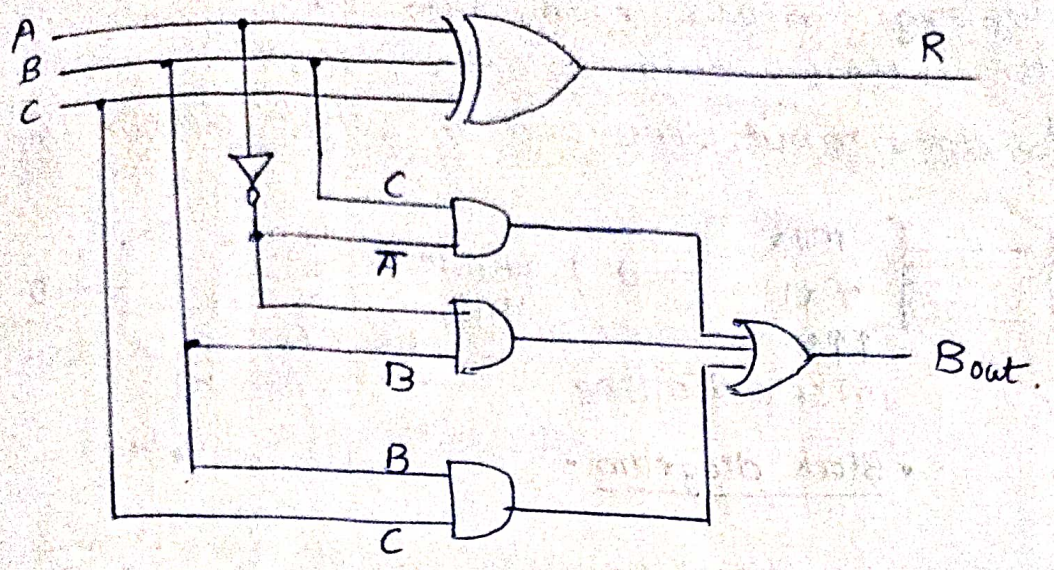
$A \oplus B \oplus Bin$

Borrow




$\bar{A}B + \bar{A}Bin + BBin$

$\bar{A}B + Bin(A \odot B)$




Logic Diagram

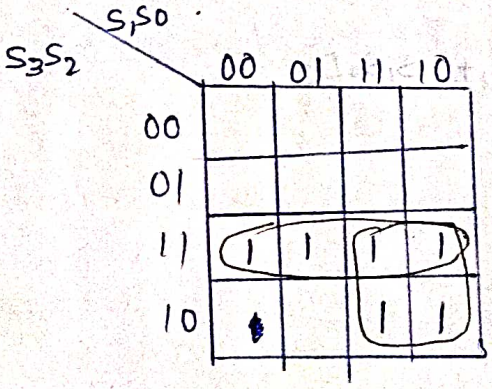
---

## BCD Addition- (4-bit adder)

| $S_3$ | $S_2$ | $S_1$ | $S_0$ | $(C_N)$ Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 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 |

Truth Table

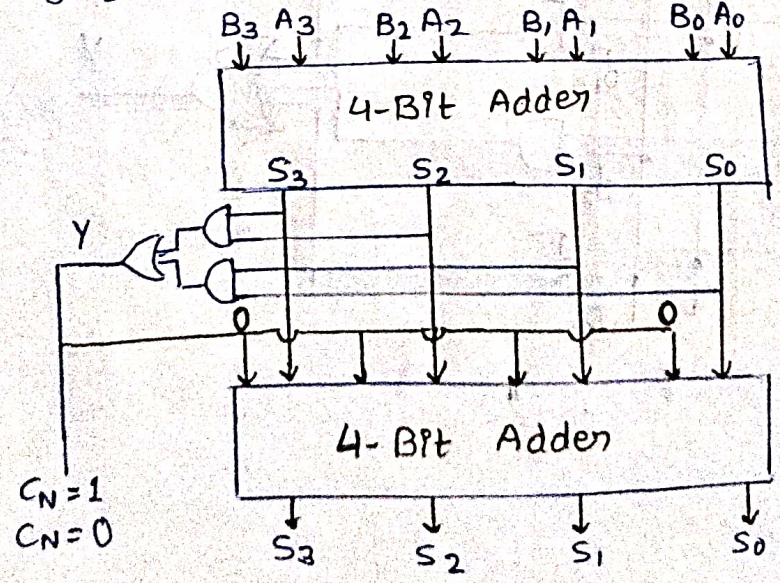
If $S_3+S_2+S_1+S_0 < 9$, then $Y=0$ (valid)
$S_3+S_2+S_1+S_0 \geq 9$, then $Y=1$ (Invalid)
and add 0110 $\{C_N=1\}$




$\Rightarrow S_3S_2 + S_3S_1$

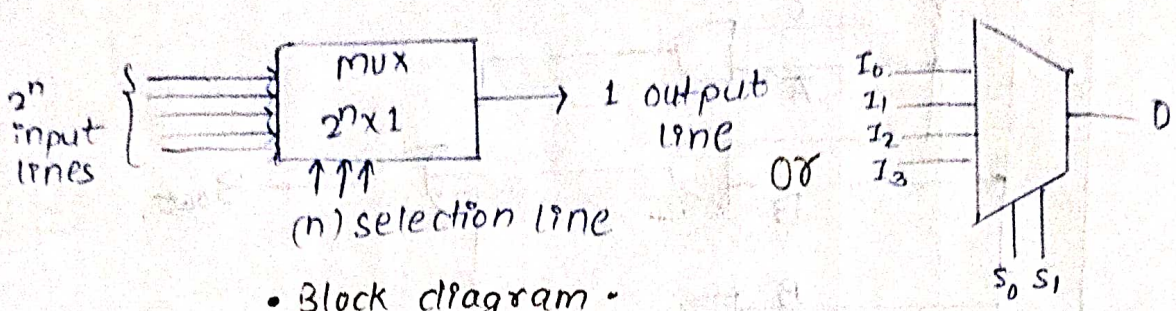
KMap

$A_3A_2A_1A_0$ = 1st NO. $B_3B_2B_1B_0$ = 2nd NO.



**Multiplexer** - Multiplexer means many into one.

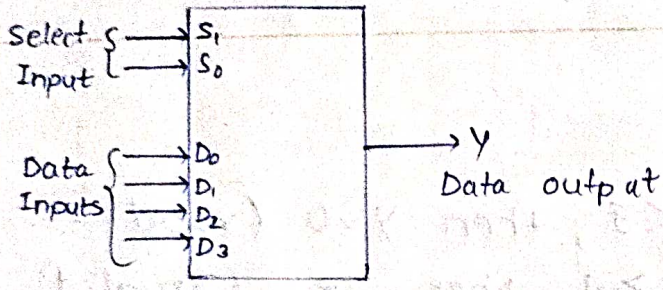
- Multiplexer also called data selector, mux.
- It has several data input line and a single output line.




- Block diagram -

- 4x1 MUX → Input=4, Output=1, selection line=2 {$2^2 \times 1$}

① Block Diagram



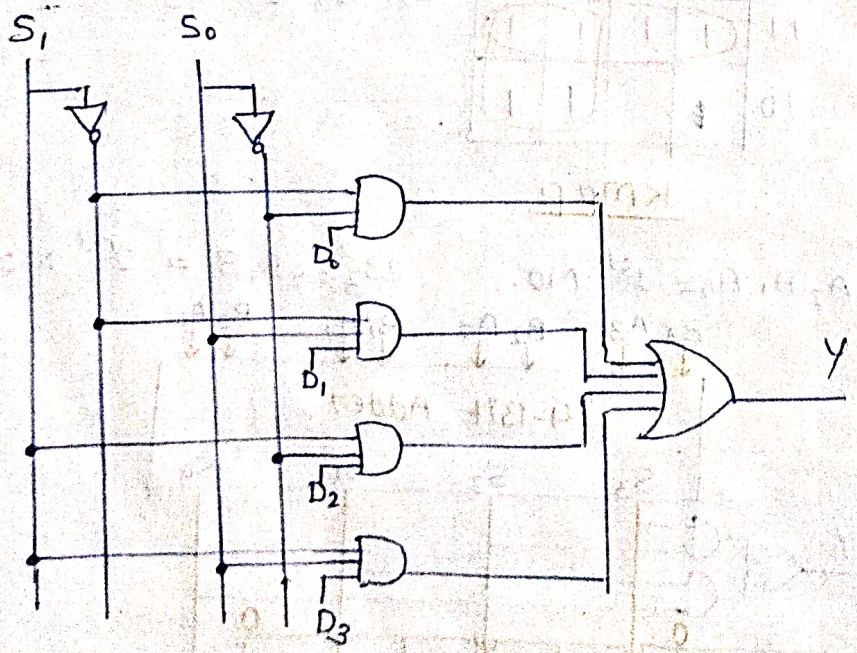

② Truth Table

| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | $D_0$ |
| 0 | 1 | $D_1$ |
| 1 | 0 | $D_2$ |
| 1 | 1 | $D_3$ |

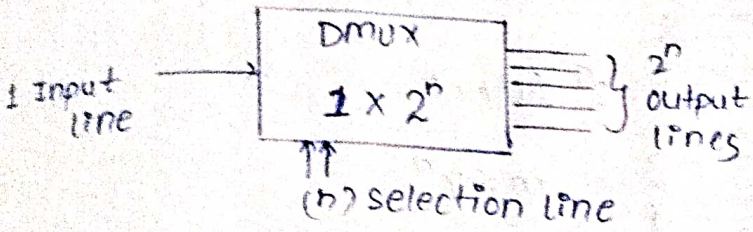
③ Boolean Expression

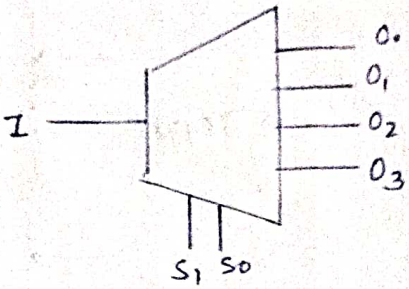
$$Y = D_0 \bar{S}_1 \bar{S}_0 + \bar{S}_1 S_0 D_1 + S_1 \bar{S}_0 D_2 + S_1 S_0 D_3$$

④ Logic Diagram



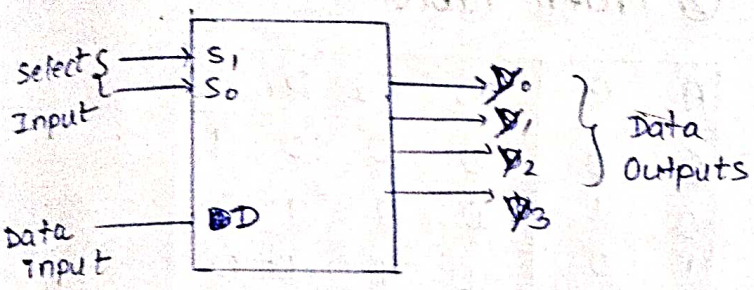
**Demultiplexer** – • It is a combinational logic circuit that performs the reverse operation of multiplexer.
• De-multiplexer means one to many
• It has one data input line and several output lines.




• 1 to 4 (1x4) Demultiplexer → Input = 1, selection line = 2, output = 4

① Block diagram



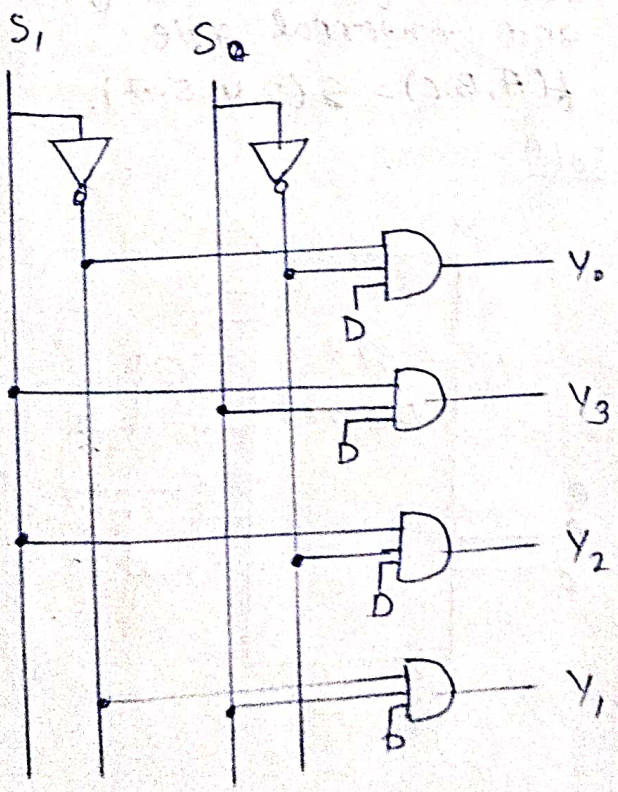

② Truth Table

| DI | SInputs | | Output | | | |
|---|---|---|---|---|---|---|
| D | $S_1$ | $S_0$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
| D | 0 | 0 | 0 | 0 | 0 | 1 |
| D | 0 | 1 | 0 | 0 | 1 | 0 |
| D | 1 | 0 | 0 | 0 | 0 | 0 |
| D | 1 | 1 | 1 | 0 | 0 | 0 |

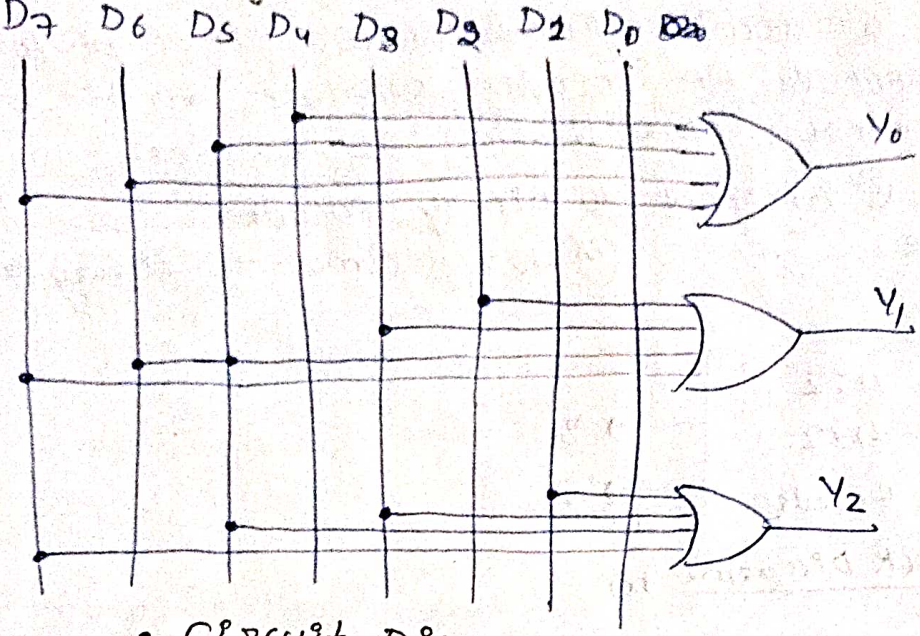
③ Boolean Expression

$Y_0 = D\bar{S}_1\bar{S}_0$ , $Y_1 = D\bar{S}_1 S_0$ , $Y_2 = DS_1\bar{S}_0$ , $Y_3 = DS_1S_0$

④ Logic Diagram



1) Logic Diagram

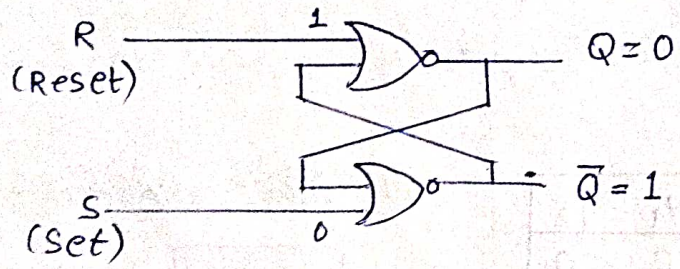



• Circuit Diagram.

## Sequential Circuit -

**Flipflop (Latch + Clock Pulse)** - A sequential circuit that consists of single bit (0,1).

**Latch using NOR -**




Truth Table NOR

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

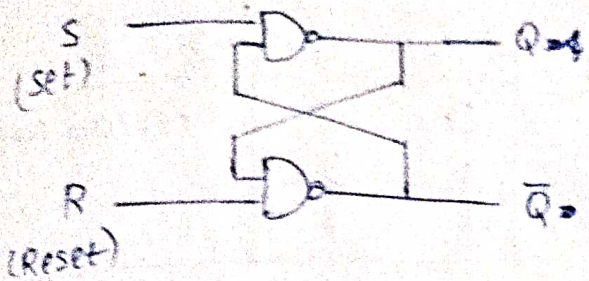
Case1: $S=0, R=1 \Rightarrow Q=0, \bar{Q}=1$

Case2: $S=1, R=0 \Rightarrow Q=1, \bar{Q}=0$

Case3: $S=1, R=1 \Rightarrow Q=0, \bar{Q}=0 \Rightarrow$ Not true/Invalid input i.e circuit will not work.

| S | R | Q | $\bar{Q}$ |
|---|---|---|---|
| 0 | 0 | Previous Result | |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | Invalid input | |

## Latch using NAND-




Truth Table NAND

| A | B | $\overline{A \cdot B}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Case1 : $S=0, R=0 \Rightarrow Q=1, \bar{Q}=1 \Rightarrow$ Invalid input

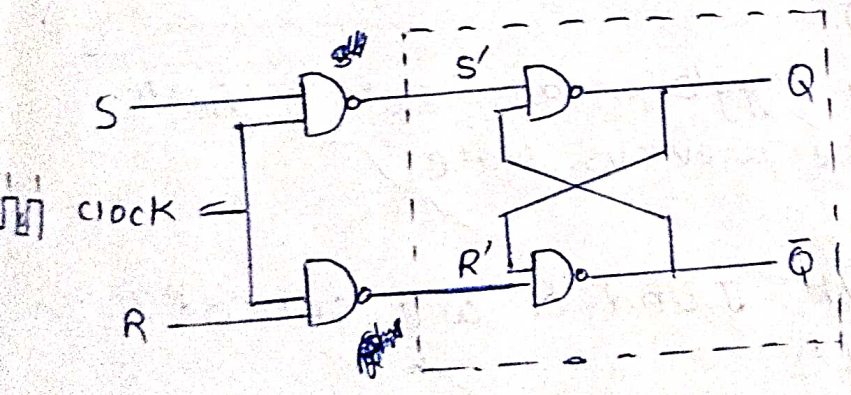
Case2 : $S=0, R=1 \Rightarrow Q=1, \bar{Q}=0$

Case3 : $S=1, R=0 \Rightarrow Q=0, \bar{Q}=1$

Case4 : $S=1, R=1 \Rightarrow Q=Q_n, Q=Q_{n+1}$ (Previous result)

| S | R | Q | $\bar{Q}$ |
|---|---|---|---|
| 0 | 0 | Invalid Input | |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | Previous result | |

- **SR- Flip Flop**- Flipflop with two inputs, one is S and other is R.




| CLK | S | R | Q | $\bar{Q}$ |
|---|---|---|---|---|
| 0 | X | X | Previous sta | |
| 1 | 0 | 0 | P. state | |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | Invalid | |

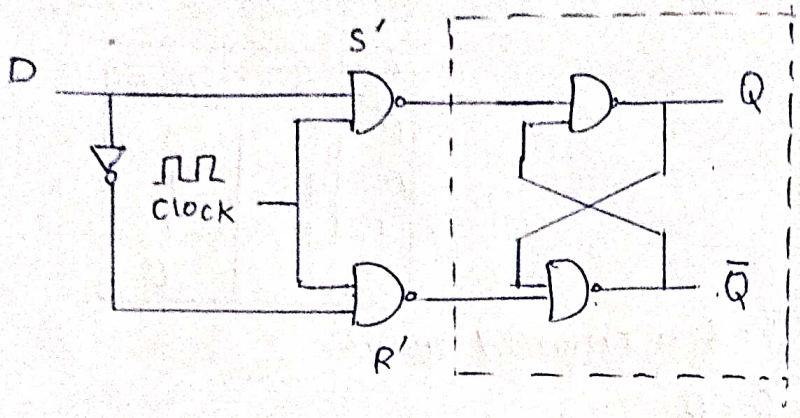
when CLK=1

$S' = \overline{S \cdot CLK} = \bar{S} + \overline{CLK} = \bar{S}$

$R' = \overline{R \cdot CLK} = \bar{R} + \overline{CLK} = \bar{R}$

Invalid state
when $Q = \bar{Q}$

- **D-Flip-flop-** Delay flipflop / Data flipflop




| Clock | D | Q $\bar{Q}$ |
|---|---|---|
| 0 | X | Previous Result |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

D flip flop eliminates the possibility of invalid states that can occur in an SR flip-flop.

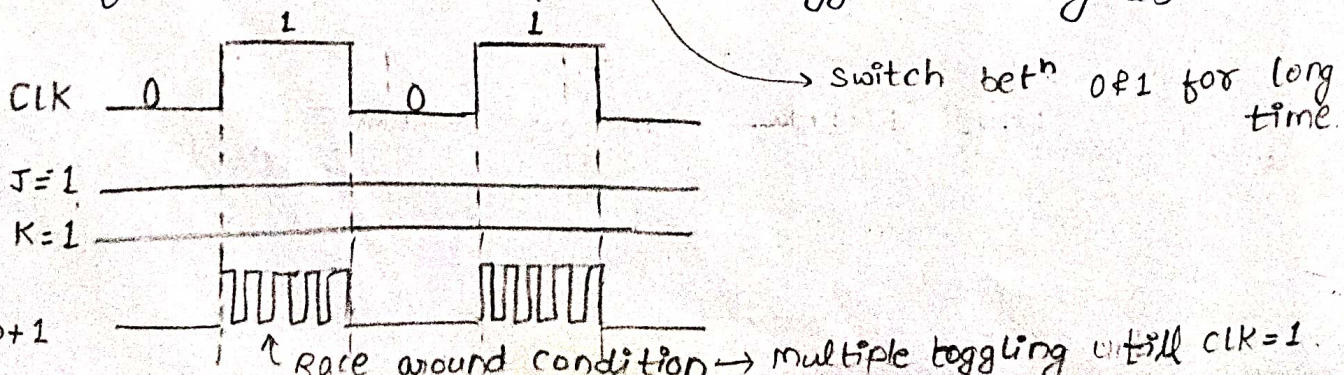
| Clk | S' | R' | Q | $\bar{Q}$ |
|---|---|---|---|---|
| 0 | X | X | Previous R. | |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

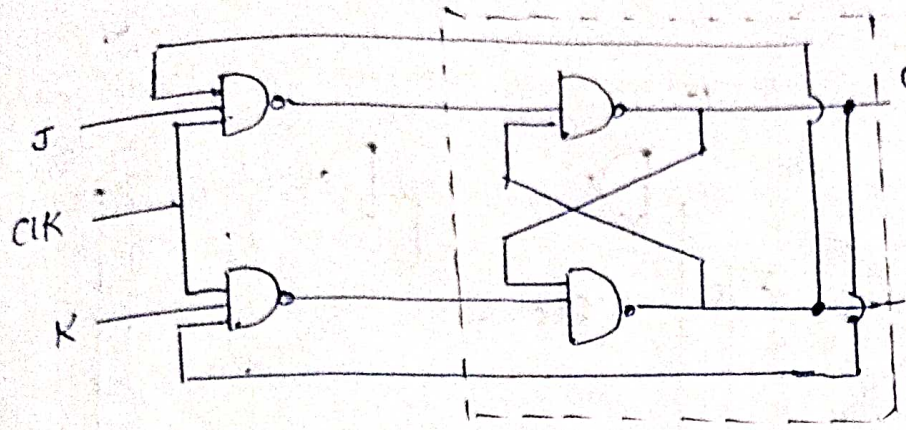
- **JK Flip-flop-** Jack Kilby Flip-flop

- It has overcome the limitations of SR flipflop. (i.e. condition of invalid state.)
- It has toggle capability, toggle means switch bet^n two state or inverting the previous state.

Toggle condition- when both J and K are set at 1. $\boxed{J=K=1}$ and clk is enabled(X).

Race around condition- If $\boxed{J=K=1 \text{ and } clk=1}$ for a long period of time, then output will toggle as long as clk = 1.






| CLK | J | K | $Q_{n+1}$ | $\overline{Q}_{n+1}$ |
|---|---|---|---|---|
| 0 | X | X | $Q_n$ | $\overline{Q}_n$ |
| 1 | 0 | 0 | $Q_n$ | $\overline{Q}_n$ |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | $\overline{Q}_n$ | $Q_n$ |

previous output at CLK = 0

or

| CLK | J | K | $Q_{n+1}$ |
|---|---|---|---|
| 1 | 0 | 0 | $Q_n$ → Previous |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | $\overline{Q}_n$ |

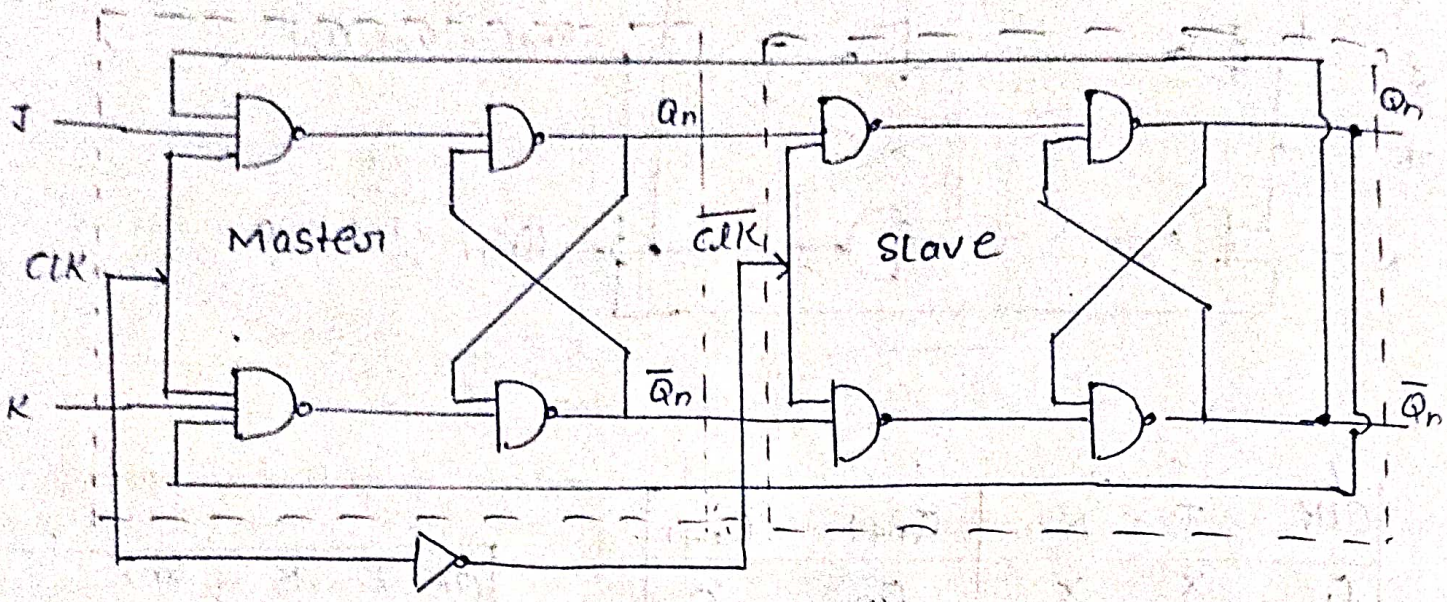
Complement of previous state

- **Master slave flip-flop** -

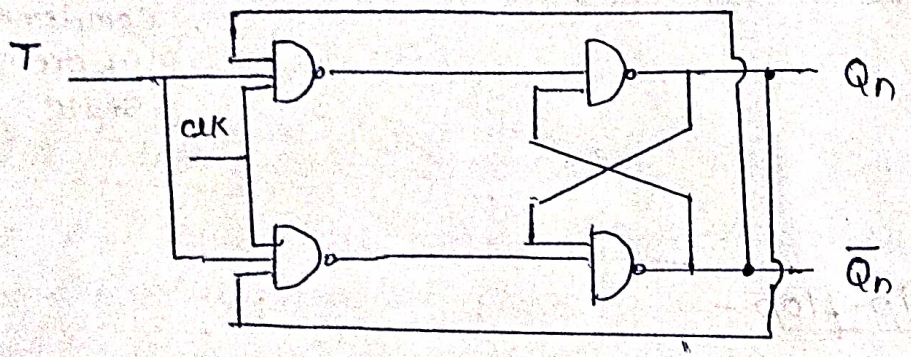
- It is used to eliminate the race around condition.
- It is constructed using two JK flip-flop.
- The first flipflop serves as 'master' and second one serve as 'slave'.

Note- For removing race around condition -
- use master slave JK flip flop
- make clock pulse < propagation delay

Propagation delay is amount of time it takes for a signal to travel from source to destination.




- T flip-flop – (Toggling Flip flop)




| Clk | T | Q |
|---|---|---|
| D | X | $Q_n$ |
| 1 | 0 | $Q_n$ |
| 1 | 1 | $\overline{Q}_n$ |

Complement of previous output

## Counter –

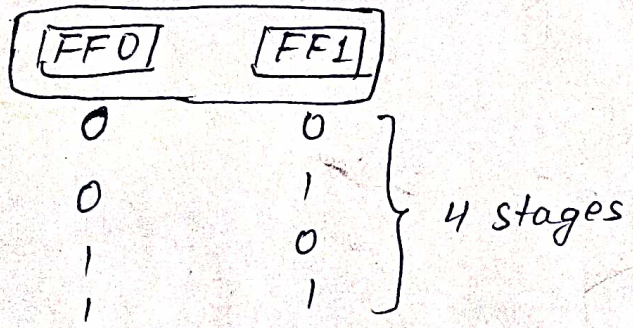
A sequential circuit, that is used to count the clock pulse. It is a special type of flip flop.

### Types of counter –

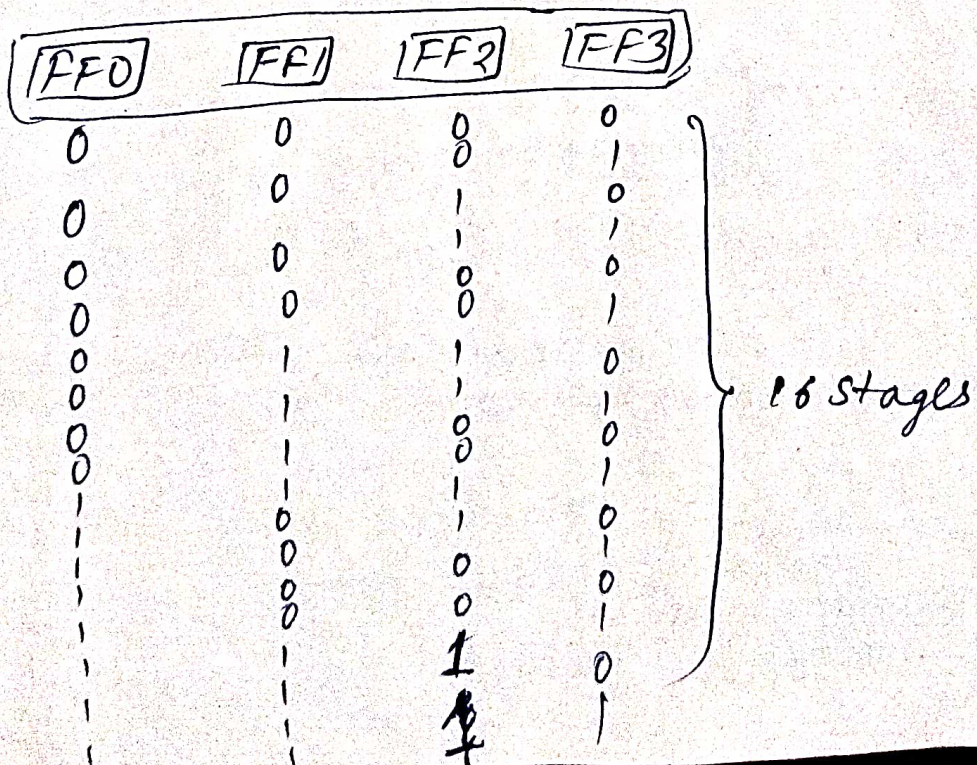
- Asynchronous (Ripple) counter (serial / non-synchronous) → 1 → 2 → 3
- Synchronous counter (Parallel) → 1 → 2 → 3

| Asynchronous Counter | Synchronous Counter |
|---|---|
| • In ripple counter flipflops are connected in that way so the output of first flipflop derives the clock of next flipflop. | • There is no connection between the first flipflop output and clock of the next output. The common clock input is connected to all |
| • Flipflop are not clocked. | • Flipflop are clocked. |
| • Circuit is simple with more no. of stages. | • Circuit becomes complicated when the no. of stages increases |
| • Speed is slow in asynchronous counter due to interstate propagation delay. | • Speed is high. |

→ 2 bit counter {mod 4 counter} ∴ $2^n$ ⇒ $n$ → Flipflop, $2^n$ → Stages




→ 4 bit counter → mod 16 counter {16 stages} and {4 flipflop}



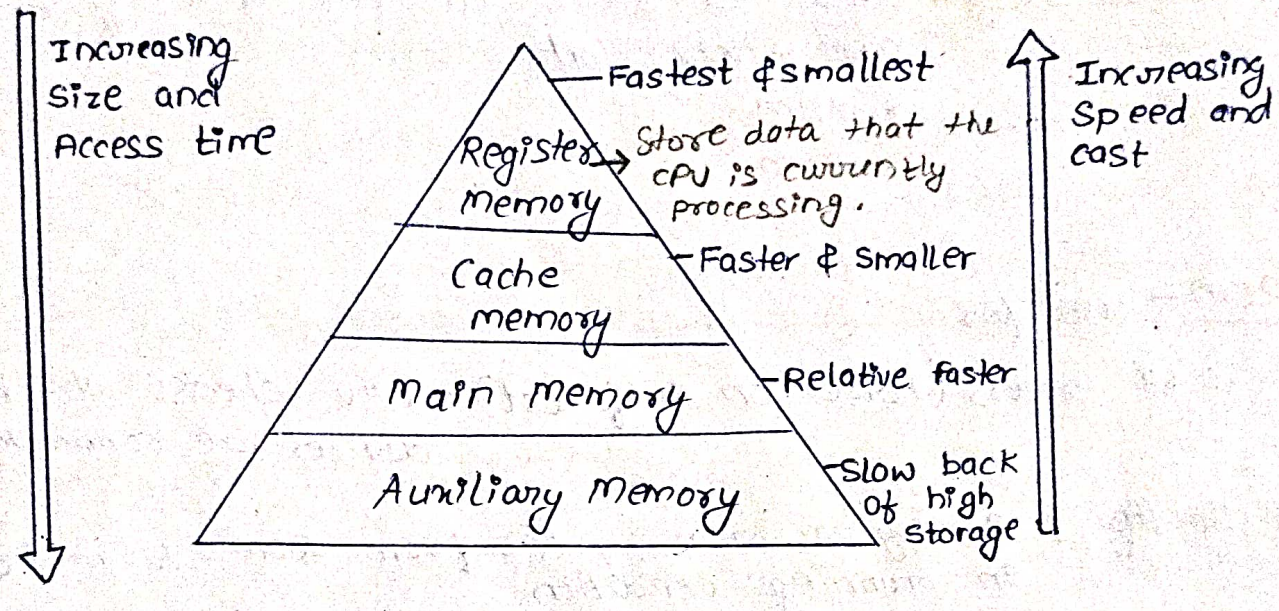
# UNIT-3

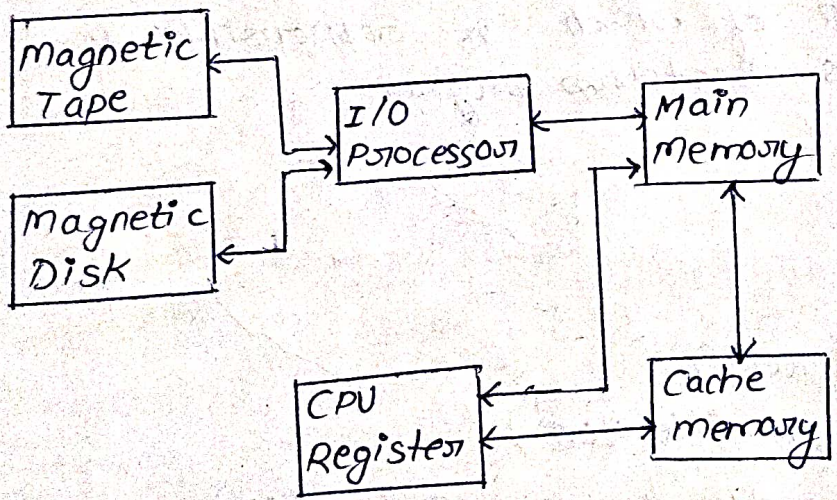
Default unit of storage → bits (b)

## Memories

In computer organisation, memory hierarchy is about arranging different kinds of storage devices in a computer based on their size, cost, access speed etc.

**Memory Hierarchy** - It is used when discussing performance issues.




**Memory representation** - No. of cells x 1 cell capacity
No. of memory locations x bits per location

4 cells { 8 bits, 8-bits, 8-bits, 8-bits } ⇒ 4 x 8 - bits

AB (Unidirectional) → Physical address in memory ← Address bus

DB (Bidirectional) → Data bus (transmit data among component)

4 cells { 00 8-bits, 01 8-bits, 10 8-bits, 11 8-bits }

$2^2 \rightarrow 4$

No. of cells = 4

Address size = 2 bits

Memory

Word addressable
1 cell capacity = 1 word

Byte addressable
1 cell capacity = 1 Byte

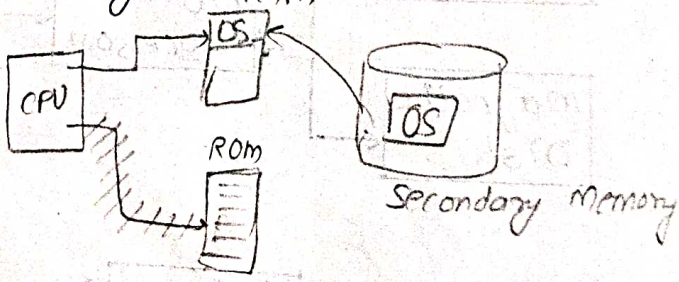
Main memory- used to store current running program and their data.
- RAM
- ROM

ROM- used for

① P.O.S.T Process (Power on self test): CPU checks which devices are connected to it

② Booting : A program run by CPU from ROM to make OS on running condition.

→ A program execution of which is responsible for booting is known as Bootstrap program.

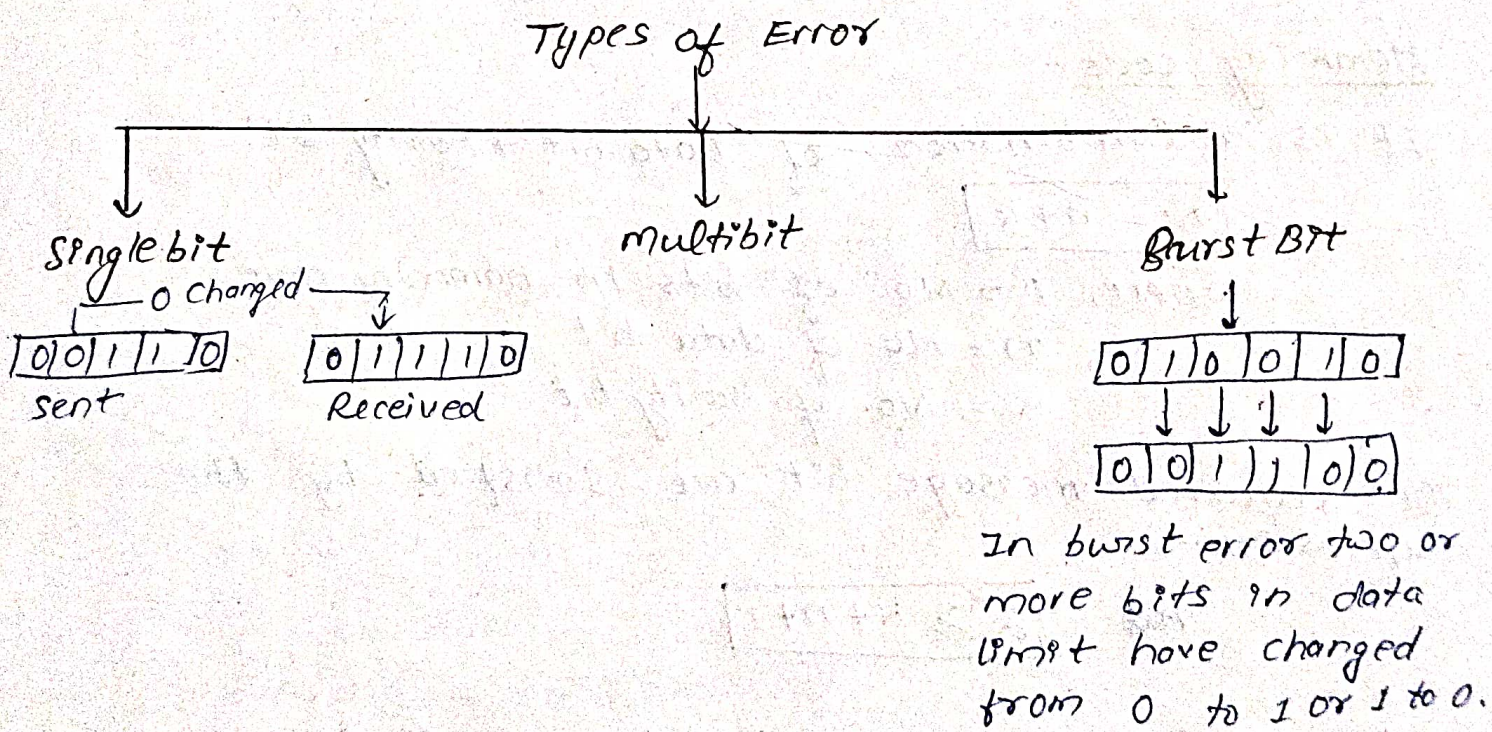



Types of ROM-
- MROM (masked ROM)
- PROM (Programmable ROM)
- EPROM (Erasable Programmable ROM)
- EEPROM (Electrically erasable programmeable ROM)

RAM - (Read write memory, Temporary, Volatile)
- SRAM (Static RAM)
- DRAM (Dynamic RAM)

| Static RAM | Dynamic RAM |
|---|---|
| • Implemented using flip-flops | • Implemented using capacitors. |
| • No refresh required. | • Periodic refresh is required. |
| • Faster Read / write | • Slow Read / write. |
| • Used for cache memory | • Used for main memory. |
| • Low idle power consumption. | • High idle power consumption. |
| • High operational power consumption. | • Low operational power consumption. |
| • Uses 6 transistor per data bit | • User 1 transistor per data bit. |

## Error Detection and correction -



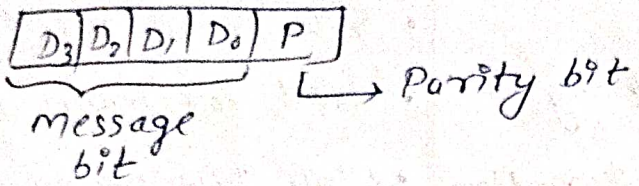

In burst error two or more bits in data limit have changed from 0 to 1 or 1 to 0.

## Detection Method -

- Parity check
- Cyclic Redundancy check (CRC)
- Check Sum

## Parity Check-

- Used to detect errors but it can't be corrected.
- Single bit error detection.




- If total No. of 1's are odd then known as odd parity (1101), and if no. of 1's are even then known as even parity (1100 or 1111)

## Error Correction-

Error correction method is Hamming code.

## Hamming Code-

It is a combination of Data bit + Parity bit.

$$N = n + K$$

where, $N$ = No. of bits in hamming code
$n$ = No. of data bit
$K$ = No. of parity bit

The required message bit are satisfied by the expression-

$$2^K \geq K + n + 1$$

Range of n for various value of K are-

| No. of Parity bits (K) | Range of data bits (n) |
|---|---|
| 3 | 2-4 |
| 4 | 5-11 |
| 5 | 12-26 |
| 6 | 27-57 |
| 7 | 58-120 |

Q- Encode the data bit 1001 into seven bit even parity Hamming Chart.

Soln- Data / message → 1001

$n = 4$

$N = 7$

$N = n + K$

$7 = 4 + K$

$\boxed{K = 3}$

Parity position are $P_1, P_2, P_4$ $\{2^n, n = 0, 1, 2 \ldots\}$

↳ 1, 2, 4 ..... }

Position for parity bit

Data positions → 3, 5, 6, 7

| | $P_4$ | $P_2$ | $P_1$ |
|---|---|---|---|
| 3 - | 0 | 1 | 1 |
| 5 - | 1 | 0 | 1 |
| 6 - | 1 | 1 | 0 |
| 7 - | 1 | 1 | 1 |

| (100) | | | (010) | (001) |
|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 1 | 0 | 0 | | 1 | | |
| $D_7$ | $D_6$ | $D_5$ | $P_4$ | $D_3$ | $P_2$ | $P_1$ |

∴ Parity bit are determined as calculating the value of $P_1, P_2, P_3$ -

$P_1$ : $(D_3, D_5, D_7) \Rightarrow P_1 : 1\ 0\ 1 \Rightarrow P_1 = 0$

$P_2$ : $(D_3, D_6, D_7) \Rightarrow P_2 : 1\ 0\ 1 \Rightarrow P_2 = 0$

$P_4$ : $(D_5, D_6, D_7) \Rightarrow P_4 : 0\ 0\ 1 \Rightarrow P_4 = 1$

or

$P_1$ : $D_3 \oplus D_5 \oplus D_7 \Rightarrow 1 \oplus 0 \oplus 1 \Rightarrow 0$

$P_2$ : $D_3 \oplus D_6 \oplus D_7 \Rightarrow 1 \oplus 0 \oplus 1 \Rightarrow 0$

$P_4$ : $D_5 \oplus D_6 \oplus D_7 \Rightarrow 0 \oplus 0 \oplus 1 \Rightarrow 1$

Final Hamming chart-

| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|
| $D_7$ | $D_6$ | $D_5$ | $P_4$ | $D_3$ | $P_2$ | $P_1$ |

→ Even parity hamming chart.

Q. 7-bit hamming code is received (1101100) check if is correct or not. Find the correct code if even parity is used.

Soln $N = 7$ (1101100)

So, $n = 4$, $K = 3$

| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|
| $D_7$ | $D_6$ | $D_5$ | $P_4$ | $D_3$ | $P_2$ | $P_1$ |

$C_1 : P_1 : D_3, D_5, D_7 = 0101 = 0$

$C_2 : P_2 \; D_3, D_6, D_7 = 0111 = 1$

$C_4 : P_4 \; D_5, D_6, D_7 = 1011 = 1$

$(P_4 \oplus D_5 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 \oplus 1 = 1)$

Error bit $= C_4 C_2 C_1 = 110$ (6)

Therefore at 6th place there is an error bit (i.e. 6th bit is error bit) and it is corrected by complementing.

Therefore correct code is 0.

| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Correct hamming code → 1001100

message → 1001

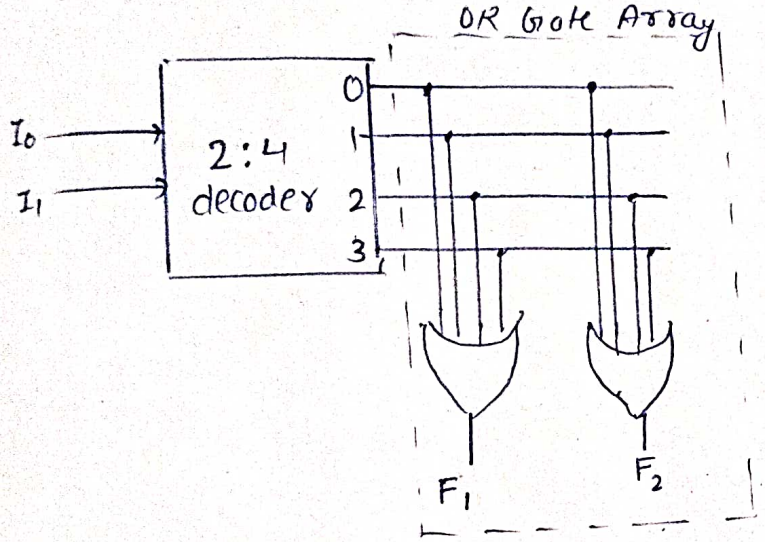
## Internal Structure of ROM -

Internal structure of ROM have two basic components -

- Decoder
- OR gates

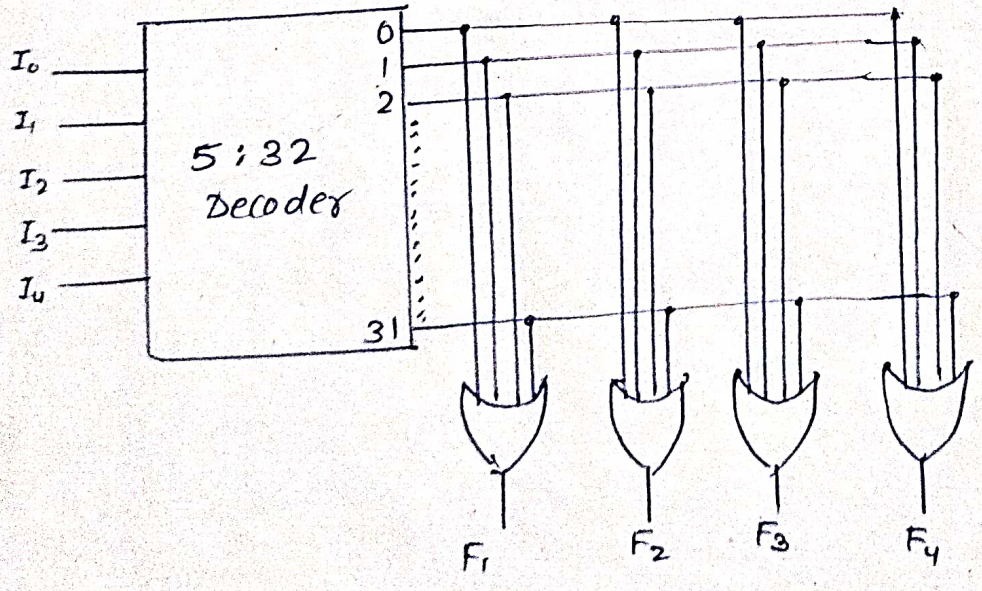
Ex - • 4 x 2 ROM

↳ 2:4 decoder and 2 OR gates




- 32x4 ROM

↳ 5:32 decoder and 4 OR gates




32x4 = 128 fuses

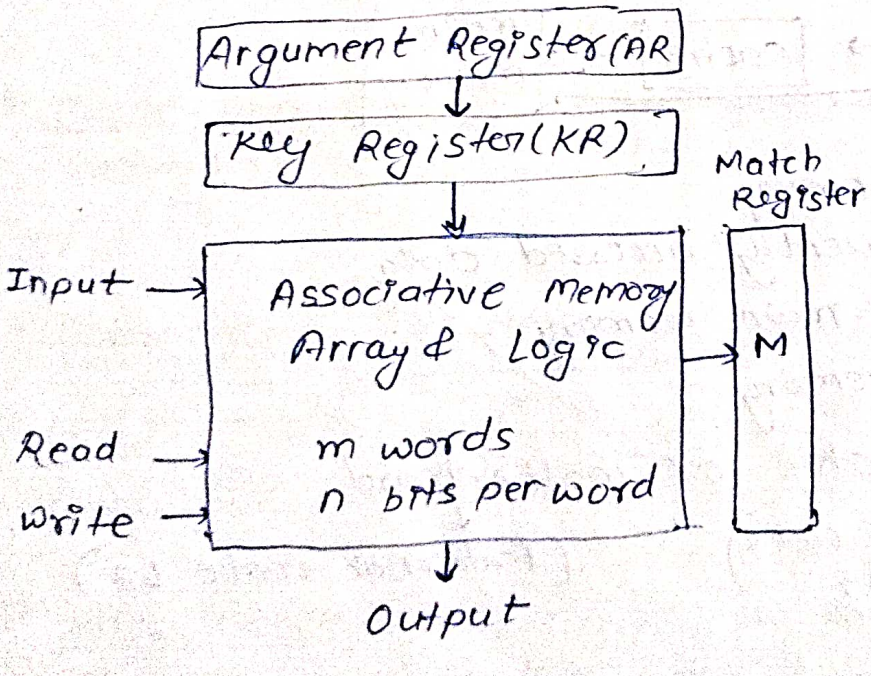
# Unit-4

## Memory Organizations

### Associative memory -

- A memory unit accessed by content is called associative memory or content addressable memory (CAM)
- It is accessed simultaneously and in parallel on basis of data content rather by specific address or location.
- When a word is written in it, no address is given.
- This Memory is capable of finding an empty unused location to store the word.
- When a word is to be "read" from an associative memory, the content of word or part of word is specified.
- Associative memory is more expensive, processing is very fast.

### Hardware organization of associative memory -




Ex -

AR 101 111000

KR 111 (masked) 00000 (unmasked)

| | | | M |
|---|---|---|---|
| Word 1 | 100 | 111100 | 0 |
| W2 | 101 | 000001 | 1 |
| W3 | 101 | 111100 | 1 |
| W4 | 110 | 001010 | 0 |
| W5 | 111 | 000111 | 0 |

0 → No match
1 → Match

**Argument Register** - Contains words to be searched.

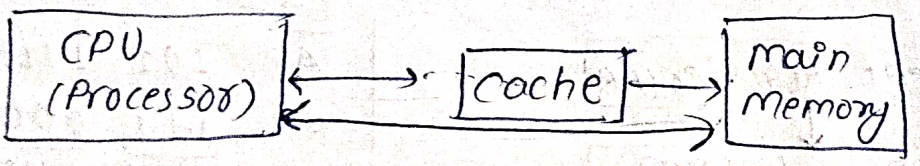
**Key Register** - Specifies which part of the argument word needs to be compared with words in memory. If all bits in KR are 1's, the entire word should be compared otherwise, only the bits having 1s in their corresponding position are compared.

**Associative memory Array** - Contains word that are to be compared with the argument word in parallel.

**match Register (M)** - After the matching process the bits corresponding to matching words in match register are set to 1.

→ Reading is accomplished by sequential access in memory for those words whose match bits are set (or 1).

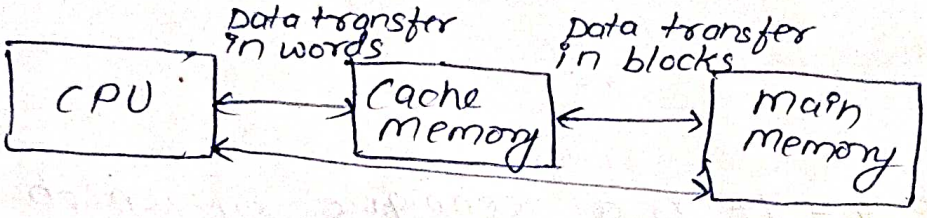
## Cache memory -




- Small sized fast memory
- contains most frequently accessed data.
- Placed b/w CPU and main memory.
- High-speed Volatile memory.
- Located inside CPU chip or motherboard.
  - (Internal cache) L1 & L2
  - (External cache L3)

## Working of cache-

- CPU initially looks in the cache for the data it needs.
- If data is there, it will retrieve it & process it.
- If data is not there, then CPU access main memory and then puts a copy of that data in cache before processing it.
- Next time if CPU need that data, it will directly retrieve cache memory.




## Cache Performance-

It is measured in terms of hit Ratio.

- **Cache Hit** - If required word is found in cache.

→ $$\text{Hit ratio (h)} = \frac{\text{Hits}}{\text{Hits + Miss}} = \frac{\text{No. of Hits}}{\text{Total no. of CPU cycle}}$$

→ **Cache Access Time** (Cache Hit Time) ($T_c$) - Time required to access word from the cache.

- **Cache miss** - If required word is not found in cache.

→ $$\text{Miss ratio (1-h)} = \frac{\text{Miss}}{\text{Hits + Miss}} = \frac{\text{No. of miss}}{\text{Total no. of CPU cycle}}$$ or miss ratio = 1 - Hit ratio

→ **Miss Penalty**: Time required to fetch the required block from main memory. (Cache miss time penalty)

- $$\text{Miss Penalty } (T_m) = \text{Cache access time} + \text{Main memory access time}$$

Average Access Time (AAT) → Read

Average Access Time = Hit ratio × cache access time + (1 − Hit ratio) × Miss penalty

(Hit ratio ↑ Miss ratio)

Miss penalty → Cache AT + Main memory AT

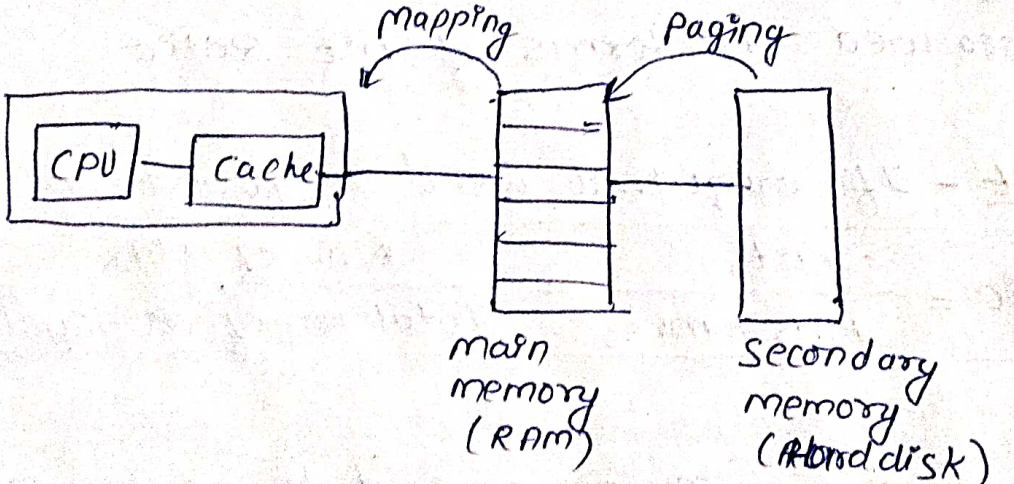
$$AAT = h \times T_c + (1-h) \times T_m$$

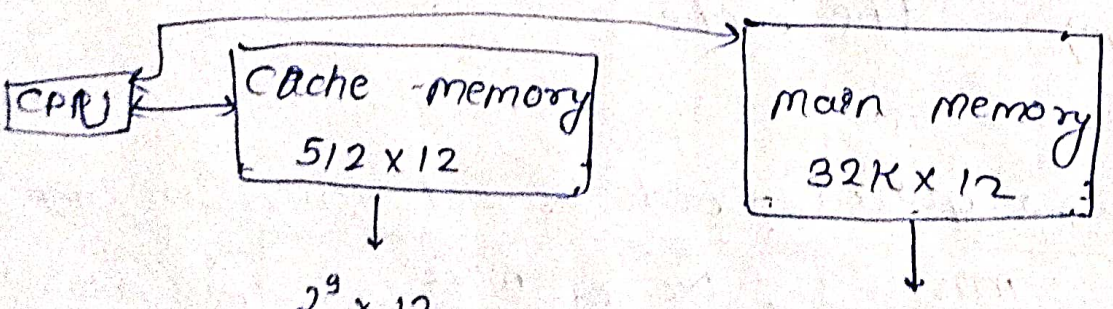
→ Write access time = 1 × max(cache memory access time, main memory access time)

(mapping or)

Cache Mapping - It is a technique by which content of main memory brought into the cache memory.

It is transformation of data from main memory to cache memory.




$2^9 \times 12$

CPU. Address = 9 bits

Data = 12 bits

$2^{15} \times 12 = 2^{15}$ words

CPU address line = 15 bits

Data = 12 bits

## Types of Mapping -

- Associative mapping
- Direct mapping
- Set-associative mapping

→ Uses Octal representation (3 bit → 1 digit)

## Associative Mapping -

- Fastest & most flexible cache organization uses associative memory.
- In Associative mapping, caches are made up of associative memory. Associative memory is used to store both the address and content (data) of the memory word.
- It permits any location in cache to store any word from main memory i.e. it enables any word from the main memory at any place in cache memory (which does not happens in other memory)

CPU address (15-bits)

↓

Argument Register

| Address (15-bits) | Data (12 bits) |
|---|---|
| 01000 | 3450 |
| 02777 | 6710 |
| 02345 | 1234 |
| | |

match Register

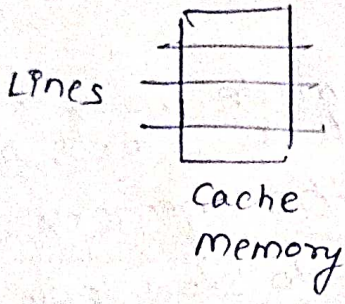
| |
|---|
| 1 |
| 0 |
| 0 |
| |
| |

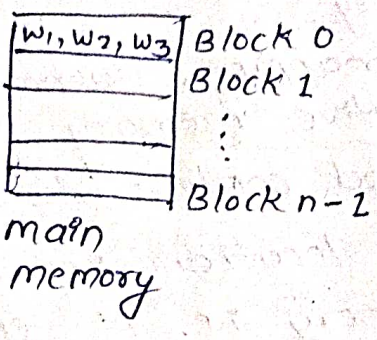
↳ use replacement policy FIFO, if space is not available in cache.

## Direct Mapping -

- Associative memory are expensive compared to RAM (because of added matching register (logic) attached with each cell)
- For RAM, direct mapping can be used.
- It maps each block of main memory into only one possible cache line / specific line in the cache.

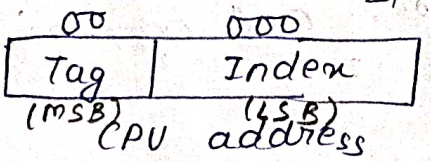
→ ~~CPU address~~




Blocks
(1 block contain 1 word to 16 words)

→ CPU address or address space is divided into 2 parts




Memory data

| | |
|---|---|
| 00000 | 1220 |
| 00777 | 2340 |
| 01000 | 3450 |
| 01777 | 4560 |
| 02000 | 5670 |
| 02777 | 6710 |

| Index | Tag | Data |
|---|---|---|
| 000 | ~~00~~ | ~~1220~~ |
| | 01 | 3450 |
| 777 | 02 | 6710 |

→ When miss occur the hit ratio will be drop b/c old value is replace by new value.

15 bit
6 bit | 9 bit
Tag | Index

00,000 → 000

Main Memory 32K x 12
Add. = 15
Data = 12

77 777

Cache 512 x 12
Add. = 9
Data = 12

↓
Octal Address

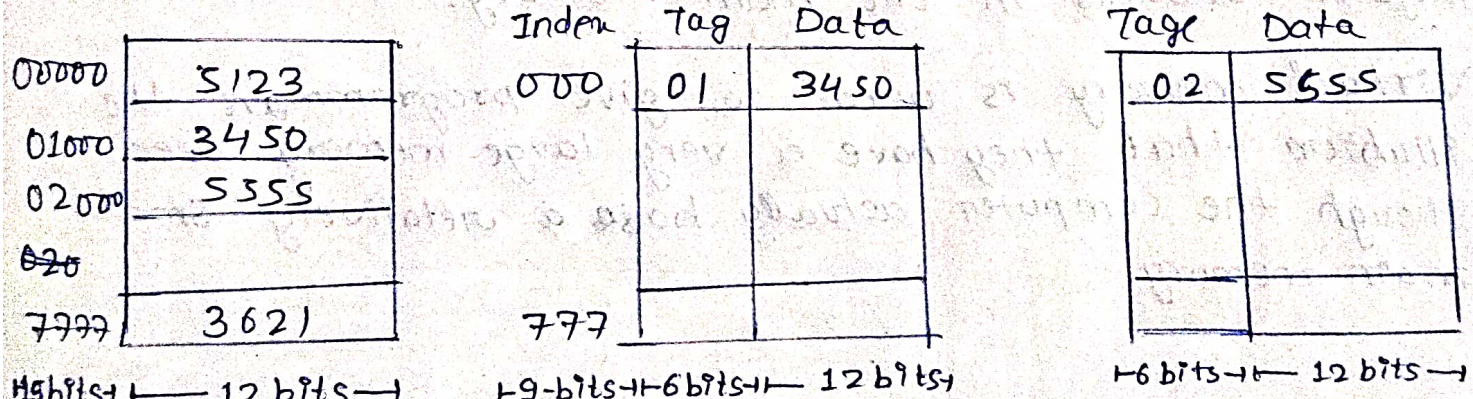
## Set-Associative Mapping – (combines direct & associative mapping)

- Improved form of direct mapping, where drawbacks of direct mapping is removed.
- Drawback of direct mapping → Two words with the same index in their address but with different tag values can't reside in cache memory at the same time.

So, for removing this drawback, set-associative mapping came into consideration., In set-associative mapping →

- Each word of a cache can store two or more words under the same index address but of different tag, by creating set.
- The no. of tag-data items in one word is said to form a set.

Ex- Two-way set associative mapping cache.




→ Set size = 2

→ Each index address refers to two data words and their associated tag, Tag 6 bits & data 12 bits, Index 9 bits

Word length = 2(6 + 12) = 36 bits

# Virtual memory -

## Address Space -

An address used by the programmer will be called a virtual address and the set of such addresses is called address space. → logical address (does not exist physically)

## Memory Space -

An address in the main memory is called physical address, the set of such allocations is called memory space.

## Virtual Memory -

Virtual memory is a concept used in some large computer systems that permit the user to construct the program as through a large memory space ~~for~~ were available. Each address that is referred by the CPU. goes thro an address mapping from so called virtual address to physical address in the main memory.

Virtual memory is used to give programmers the illusion that they have a very large memory even though the computer actually has a relatively small main memory.