# Combinational circuit

- Logic circuits can be devided into two types —
  - → Combinational logic circuit.
  - → Sequential logic circuit.

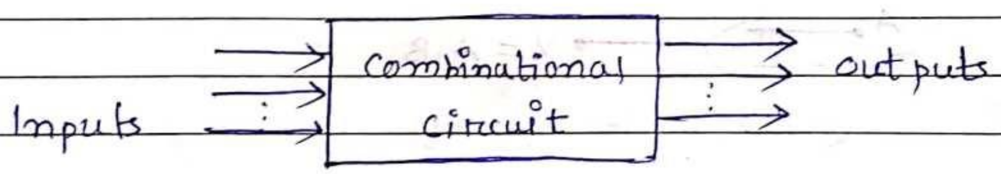
- **Combinational logic circuit:**

A combinational logic circuit consists a logic gates whose output is determined by the combination of current inputs.

→ It consists of Input variables, logic gate and output variable.

→ No feedback is required.

→ No memory is required.




Ex- Multiplexer, Encoder, Decoder, parallel adders, etc.

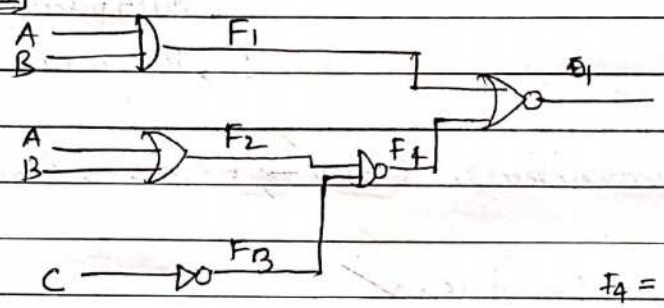
Combinational circuit

## • Anlysis procedure —

→ The Analysis of combinational circuit requires that we determine the function that the circuit Implements.

Logic diagram
↓
Boolean functions
↓
truth table.
↓
circuit operation.

**Ex-1**




$F = \overline{F_1 + F_4}$

$= \bar{C}\,(A \oplus B)$

$F = \overline{F_4 + F_1}$

$F_4 = \overline{F_2 \cdot F_3}$

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

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

→

| A | B | C | $F_1$ ($A\cdot B$) | $F_2$ ($A+B$) | $F_3$ ($\bar{C}$) | $F_4$ | F |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 ✓ |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 ✓ |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 ✓ |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |

$F = \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$ ~~correct~~

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

$\boxed{F = \bar{C}\,[(A \oplus B) + AB]}$ $= \bar{C}\,(A+B)$ (final Analysis procedure)

## • Design procedure :

Specification.
↓
Truth table.
↓
Boolean function
↓
Logic Diagram.

## • Combinational Circuits —

(I) Arithmetic Circuits —
- → Adders ✓
- → Subtractors. ✓
- → Multiplier. ✓
- → Magnitude comparator. ✓

(II) Code converters. (BCD → Excess-3 code)

(III) Encoder & Decoder. ✓

(IV) Multiplexer & Demultiplexer.

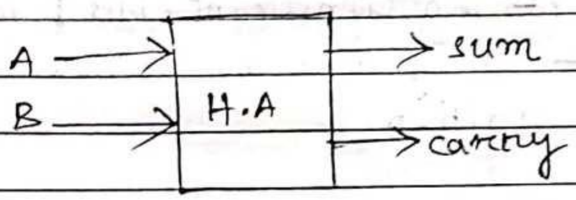
(V) programmable Logic Devices (PLDs)
- → PLA (programmable array logic)
- → PAL (programmable logic array)

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | (2) 10 |

## • ADDERS :

→ Half - adder.
→ Full - adder.
→ Binary parallel Adder
  (Ripple Carry adder)
→ Carry look-Ahead Adder.

### • Half adder :




→ Half adder is a combinational circuit that performs arithmetic sum of Two bits.

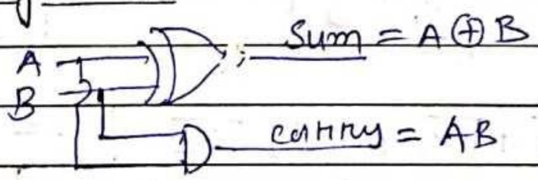
Truth table :

| A | B | S | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

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

$Carry = A \cdot B$.

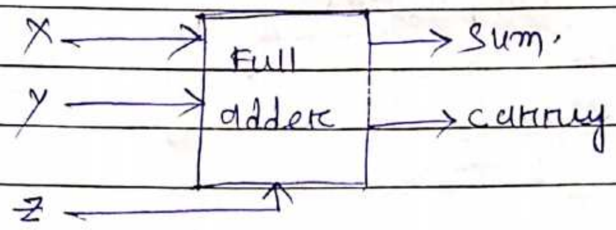
Logic diagram :




*→ required minimum no. of NAND and NOR gate to implement H.A = 5.

## • Full Adder:

→ A full adder is a combinational circuit that performs the arithmetic sum of three bits




here, x, y = Two significant bits to be added.
and z = carry from the previous lower significant position.

### Truth Table:

| x | y | z | sum | carry |
|---|---|---|---|---|
| 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 |

**S**

| x \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 (0) | (1) (1) | 0 (3) | (1) (2) |
| 1 | (1) (4) | 0 (5) | (1) (7) | 0 (6) |

**C**

| x \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

$S = x\bar{y}\bar{z} + \bar{x}\bar{y}z + xyz + \bar{x}y\bar{z}$

$= \bar{z}(xy + \bar{x}\bar{y}) + z(x\bar{y} + y\bar{x})$

$= \bar{z}(x \odot y) + z(x \oplus y).$

$C = yz + xz + xy$

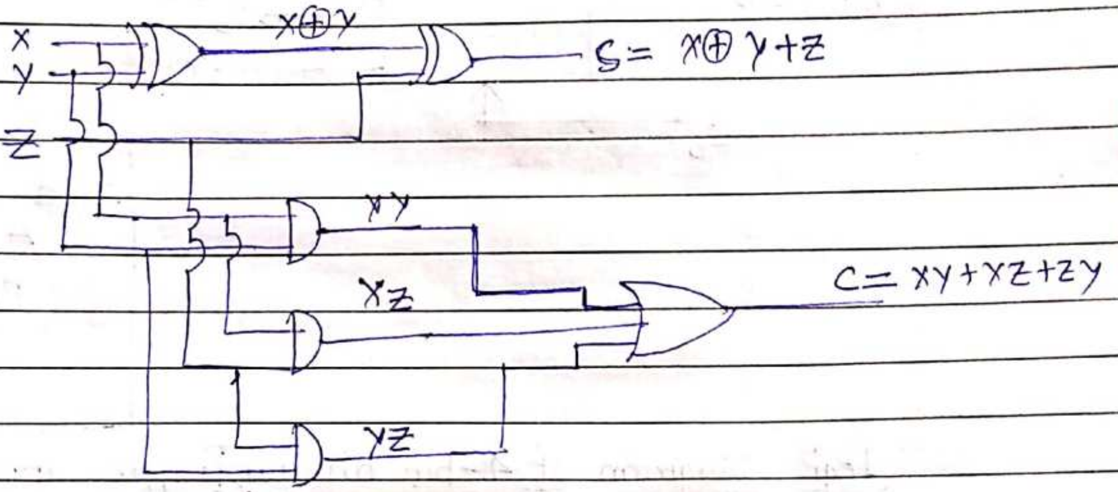
$\Rightarrow Z\,\overline{(x \odot y)} + \bar{Z}\,(x \oplus y)$

$\Rightarrow Z\,\overline{(x \oplus y)} + \bar{Z}\,(x \oplus y)$

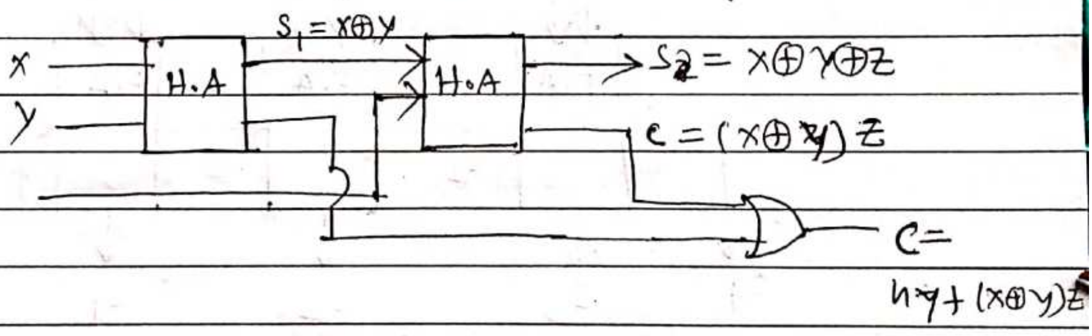
$\Rightarrow (x \oplus y) \oplus Z$

$S \Rightarrow x \oplus y \oplus Z$

Logic Diagram :



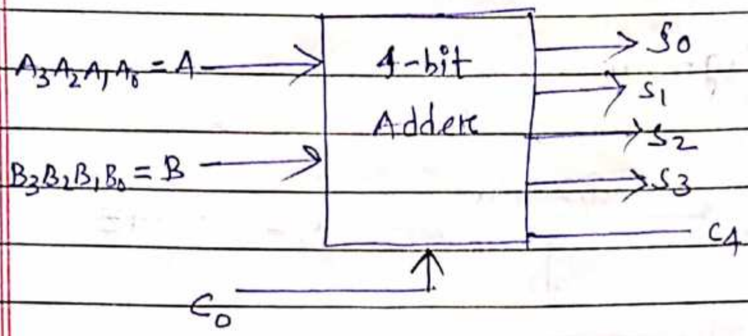

→ A full adder = 2 half adder + 1 OR gate.




→ *required minimum number of NAND and NOR gate to impliment F.A = 9.

## • Binary adder :

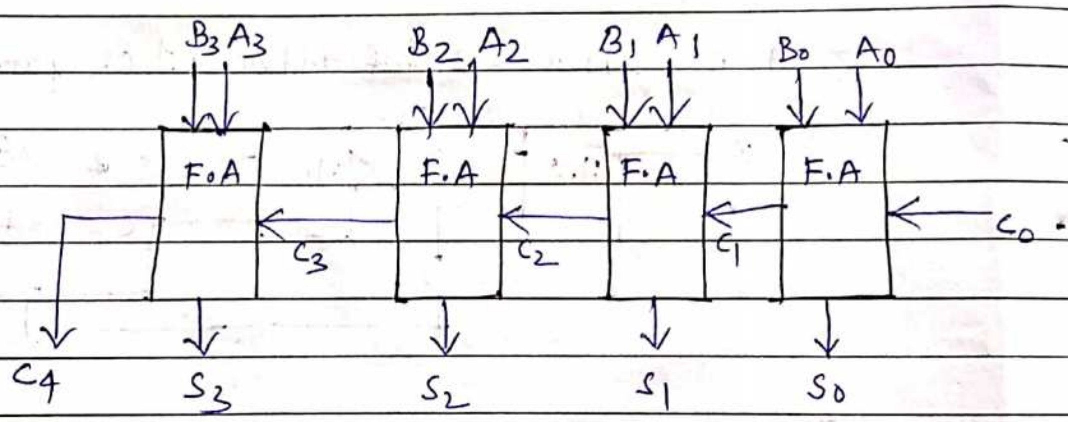
→ "Binary adder is a circuit that produces the arithmetic sum of two binary numbers."




Ex =

| | | | | |
|---|---|---|---|---|
| | $C_3$ | $C_2$ | $C_1$ | |
| | 0 | 0 | 1 | |
| A = | 1 | 0 | 0 | 1 |
| B = ($C_4$ = 0) | 0 | 1 | 0 | 1 |
| | 1 | 1 | 1 | 0 |
| | $S_3$ | $S_2$ | $S_1$ | $S_0$ |

### Logic Diagram of 4-bit binary ripple carry adder :




→ with 'n'-bit adder required 'n' no. of F.A.
(Ripple carry adder also called Binary parallel adder)

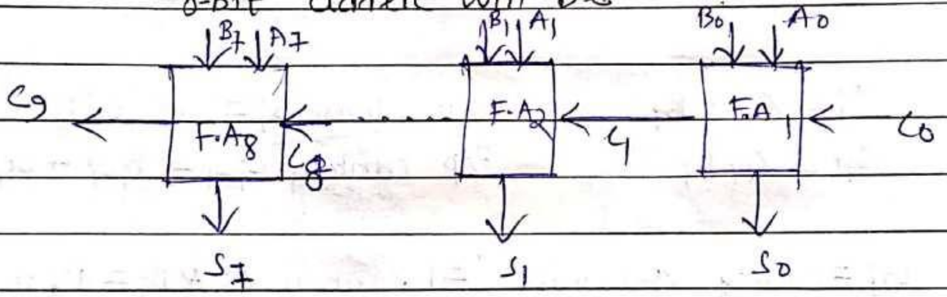
**Draw back:**

→ The longest propagation delay.

**→ In this adder, The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders.

**Example:** ①

A - 8 bit ripple carry adder. The carry propagation delay of each F.A is 10 ns & the sum propagation delay of each F.A is 15 ns. The worst case delay (in ns) of this 8-bit adder will be -




→ 8 × 10 ns

→ 80 ns. (Ans)

② If there is no $C_0$ then,

→ 7 × 10 ns + (time taken to produce $C_1$)

→ 70 ns + 15 ns

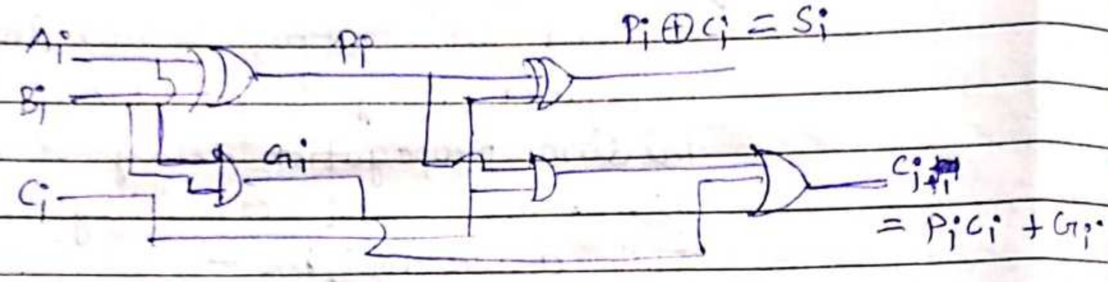
→ 85 ns.

## • CARRY LOOKAHEAD ADDER: (CLA)

→ It is reduces carry propagation delay time compared to ripple carry adder.

→ i.e it is a high speed adder.

→ But hardware complexity increases.




$P_i = A_i \oplus B_i$     O/P sum $S_i = P_i \oplus C_i$

$G_i = A_i B_i$     O/P carry $C_{i+1} = P_iC_i + G_i$.

$G_i$ = carry generate (=1, when $A_i$ & $B_i$ = 1, regardless of i/P carry $C_i$)

$P_i$ = Carry propagate, it determines whether a carry into stage 'i' will propagate into stage 'i+1'.

Boolean function for the carry outputs of each stage.

$C_0$ = i/P carry    [ $(C_{i+1} = P_iC_i + G_i)$

$C_1 = G_0 + P_0C_0$    $(G_0 = A_0B_0, P_0 = A_0 \oplus B_0)$

$C_2 = G_1 + P_1C_1 \Rightarrow G_1 + P_1 (G_0 + P_0C_0) \Rightarrow G_1 + P_1G_0 + P_1P_0C_0$.

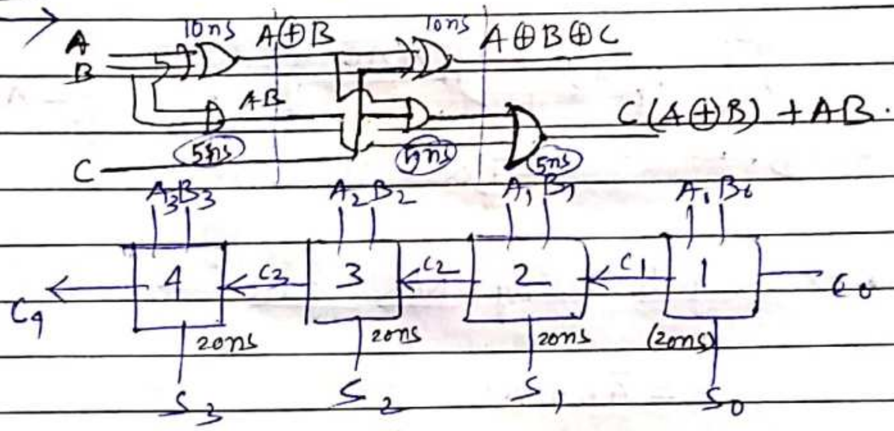
$C_3 = G_2 + P_2C_2 \Rightarrow G_2 + P_2G_1 + P_2P_1G_0 + P_2P_0P_1C_0$

$C_4 = G_3 + P_3C_3 \Rightarrow G_3 + P_3G_3 + P_3G_1P_2 + P_3P_2P_1G_0 + P_3P_2P_0P_1C_0$.

• Comparision of Hardware:

| | 4 bit RCA | 4 bit CLA |
|---|---|---|
| EX-OR → | 8 | 8 |
| AND → | 8 | 14 |
| OR → | 4 | 4 |
| | 20 gate req | 26 gate req. |

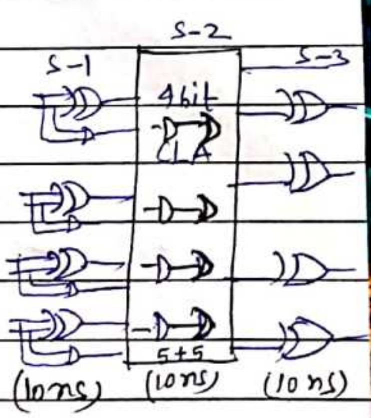
**Example:** Assume that the EX-OR gate has a propagation delay of 10 ns & that the AND (or) OR gates have a prop delay of 5 ns what is the total propagation delay time in the 4-bit RCA & CLA.




tpd → 20ns * 4 → 80 ns. (in RCA)

→ Stage-1 10ns, Stage-2 10ns, Stage-3 10ns

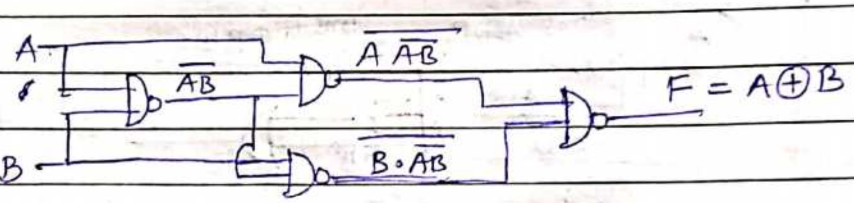
(Total propagation delay) tpd → 30ns (in CLA)



- ADDERS using universal gates :

  - EX-OR gate using NAND gates —

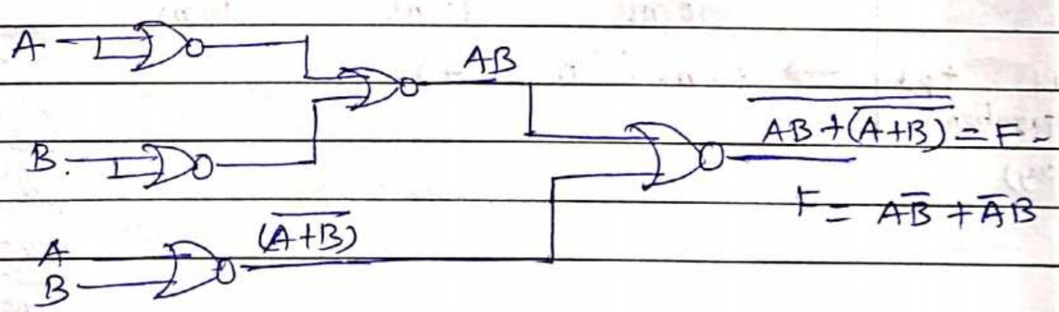
$$F = A\bar{B} + \bar{A}B$$
$$= (A+B)(\bar{A}+\bar{B})$$
$$= (A+B).\overline{AB}$$
$$= A\,\overline{AB} + B\,\overline{AB}$$
$$\bar{F} = \overline{A.\overline{AB} + B.\overline{AB}}$$
$$= \overline{A.\overline{AB}} . \overline{B.\overline{AB}}$$
$$F = \bar{\bar{F}} = \overline{\overline{A.\overline{AB}} . \overline{B.\overline{AB}}}$$




  - EX-OR gate using NOR gate —

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

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

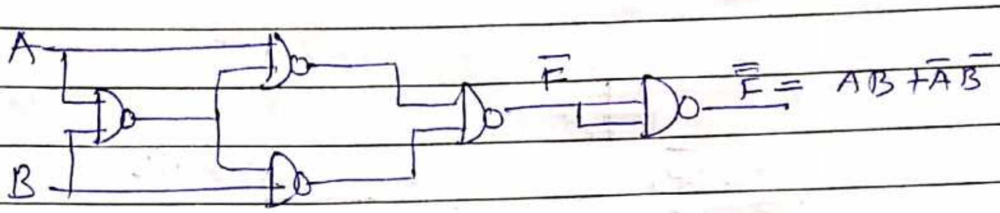


• EX-NOR using NAND gates =

$$F = AB + \bar{A}\bar{B}$$

$$F = \overline{A \oplus B}$$

$$F = \overline{A\bar{B} + \bar{A}B}$$




• EX-NOR using NOR gates =

$$F = AB + \bar{A}\bar{B}$$

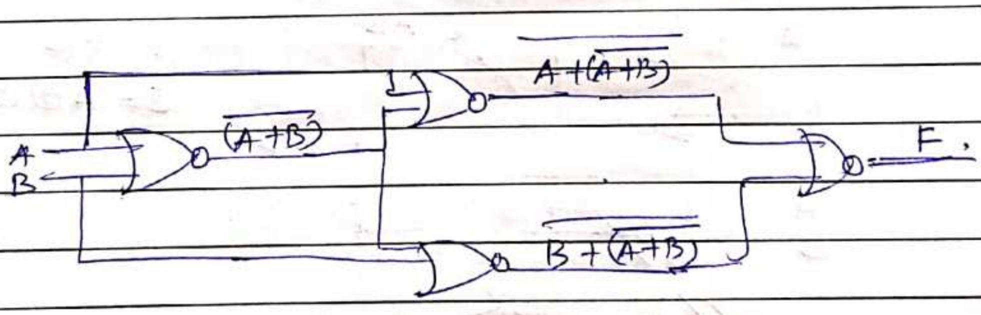
$$= (A + \bar{B})(\bar{A} + B)$$

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

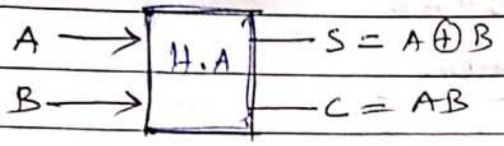
$$= (A + \overline{(A+B)})(B + \overline{(A+B)})$$

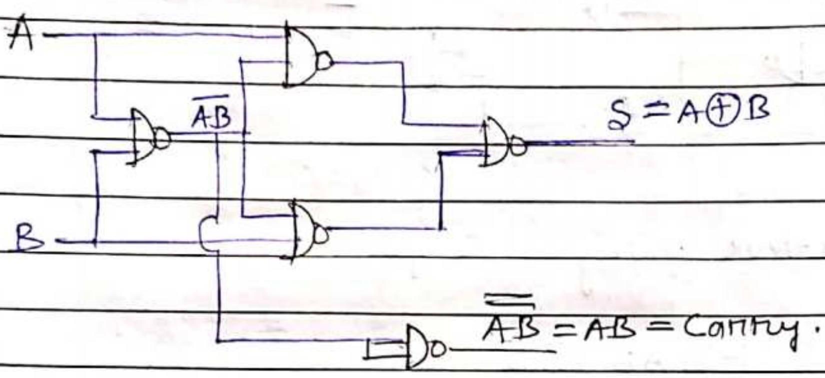
$$\bar{F} = \overline{A + \overline{(A+B)}} + \overline{B + \overline{(A+B)}}$$

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



- Half adder using NAND gates =




(5 - NAND gate required to Impremt H.A)

- Half adder using NOR gate =

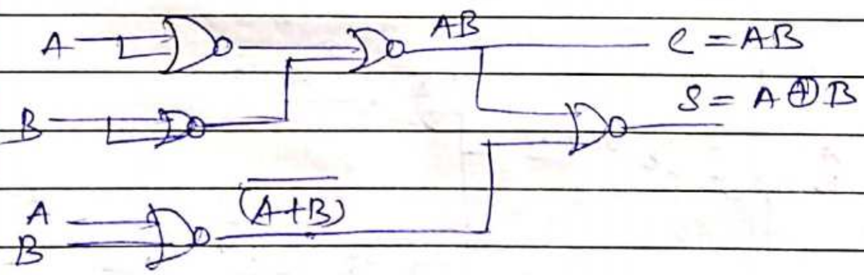
$S = A\overline{B} + \overline{A}B$

$C = AB$

$\overline{S} = \overline{A\overline{B}} \cdot \overline{\overline{A}B}$

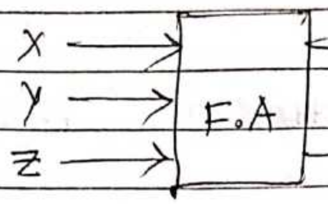
$S = \overline{(\overline{A}+B)(A+\overline{B})}$

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




(5 - NOR gate required to Implemented H.A)

## • Implementation of Full adder using NAND gates —




$S = x \oplus y \oplus z = \Sigma(1,2,4,7)$

$C = xy + yz + zx = \Sigma(3,5,6,7)$

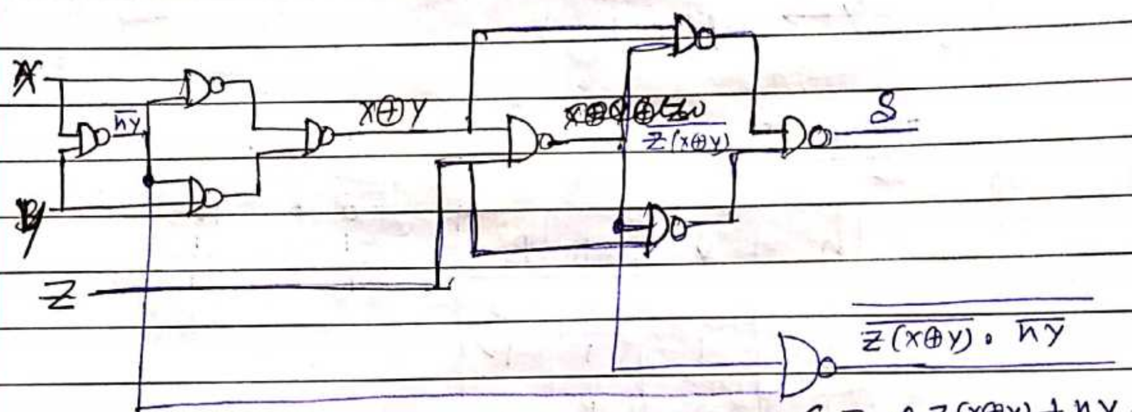
or

$= \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$

$= xy(z+\bar{z}) + z(x \oplus y)$

$C = xy + z(x \oplus y)$.

$S = x \oplus y \oplus z \qquad C = xy + z(x \oplus y)$.




$C = z(x \oplus y) + xy$.

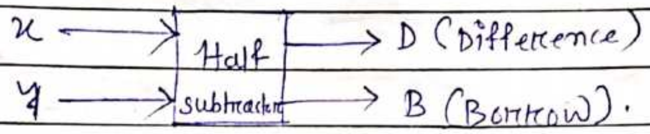
(9-NAND gate required to implement one full adder)

(9-NOR gate required to implement one full adder) → (every thing is same just change NAND by NOR)

## • Subtractors :

→ Half Subtractor.
→ Full Subtractor.
→ Binary subtractor, Adder cum subtractor.

• **Half Subtractor :** It performs subtraction of two bits.




x → minuend
y → subtrahend

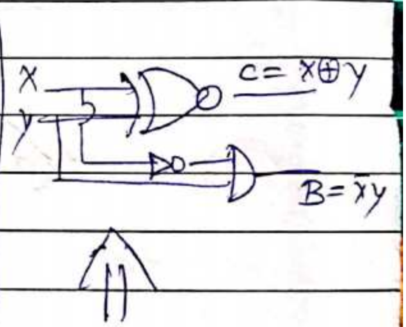
**Truth Table :**

| x | y | D | B |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1✓ |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

0 − 1 = 1; 0 − 0 = 0; (10 − 2) − 1 = 1

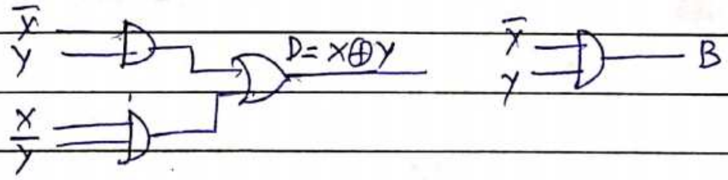
$D = x \oplus y$

$B = \bar{x}y$




**Logic diagram**

ⓐ sum-of-products form =

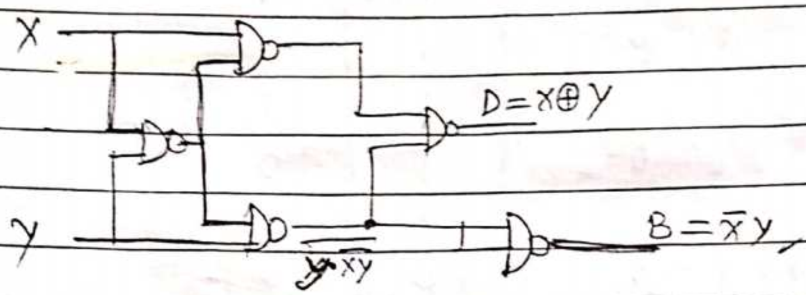


(H.S)

• Using only NAND gates — (5-NAND gate required)

$D = x \oplus y$

$B = \bar{x}y$




$\overline{xy}$

⇒ ~~$\overline{\bar{x}+y}$~~

$\overline{y(\bar{x}+\bar{y})}$

⇒ $\overline{\bar{x}y}$

⇒ $\bar{x}y$

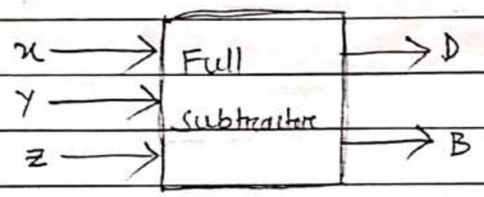
→ ~~5-NAND and NOR gate minimum req~~

minimum 5-NAND ~~and NOR~~ gate required to Implement one Half adder.

→ also 5-NOR gate required to Implement one H.A.

• Full subtractor —

→ It performs the subtraction of '3' bits.




x, y → two significant bits to be subtract

z → Borrow from next stage.

Truth table of full subtractor =

| | x | y | z | D | B |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 | 1 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 |
| 6 | 1 | 1 | 0 | 0 | 0 |
| 7 | 1 | 1 | 1 | 1 | 1 |

$1\ \overset{1}{0} - 2$

$\begin{array}{r} 0\ 1 \\ \hline 1\ 1 \end{array}$

$D = \Sigma(1,2,4,7) = A \oplus B \oplus C$

$= \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$

$B = \Sigma(1,2,3,7)$

B

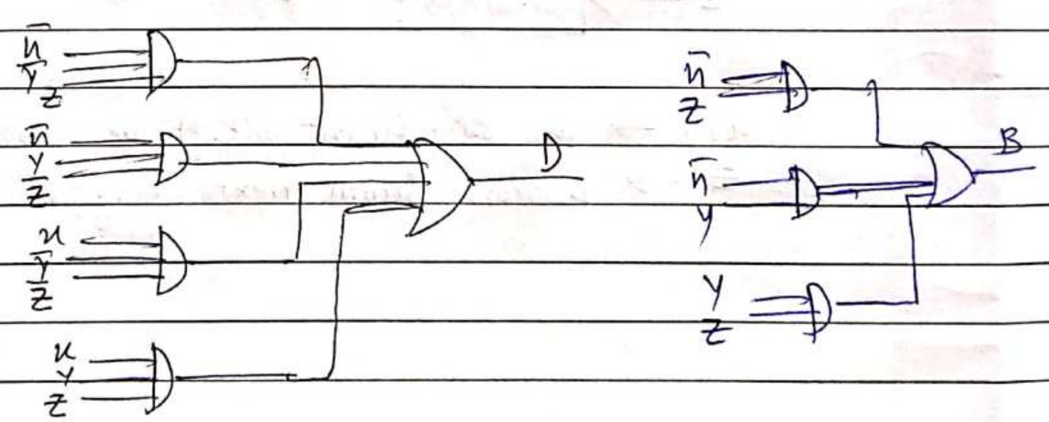
| x \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | $0_0$ | $1_1$ | $1_3$ | $1_2$ |
| 1 | $0_4$ | $0_5$ | $1_7$ | $0_6$ |

$B = \bar{x}z + \bar{x}y + yz$

$= \bar{x}y + (\overline{x \oplus y}) \cdot z$

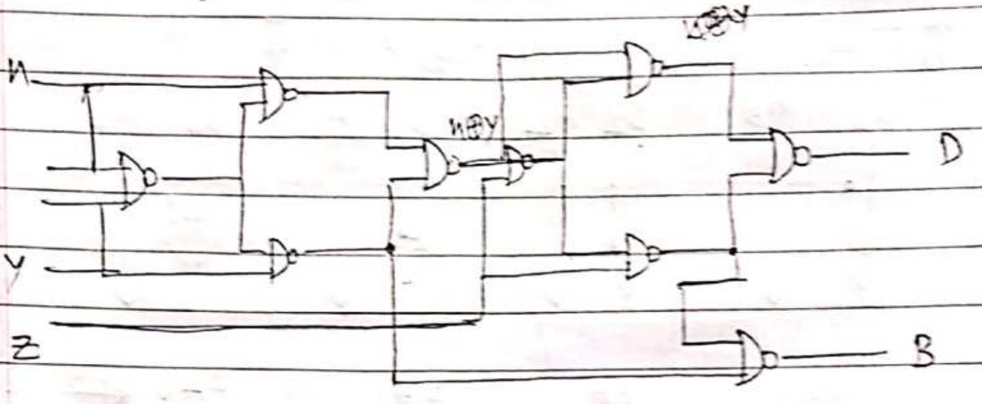
Logic diagram =

(a) Implimentation in sop form =




→ 9 NAND and NOR gate require to Implement full subtractor. → 2 Half sub + 1 OR gate = full sub.

(b) using only NAND gate =




(9-NAN gate required).

- **Binary Subtractor =**

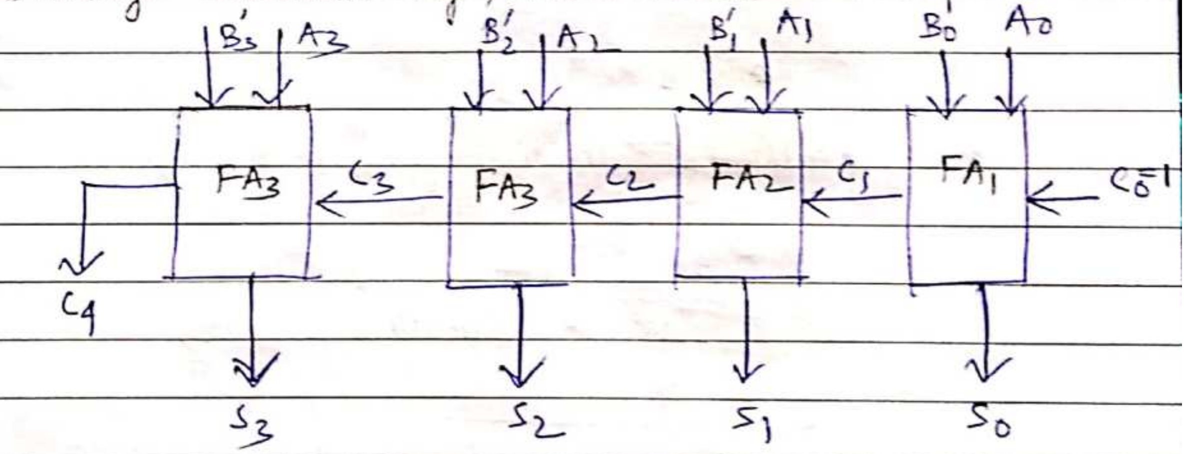
A + [2's complement of B]

$A_3 \quad A_2 \quad A_1 \quad A_0$

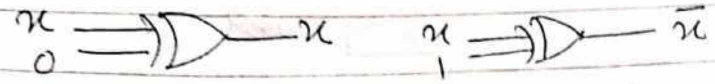
$B_3' \quad B_2' \quad B_1' \quad B_0' \rightarrow$ (1's complement of B')

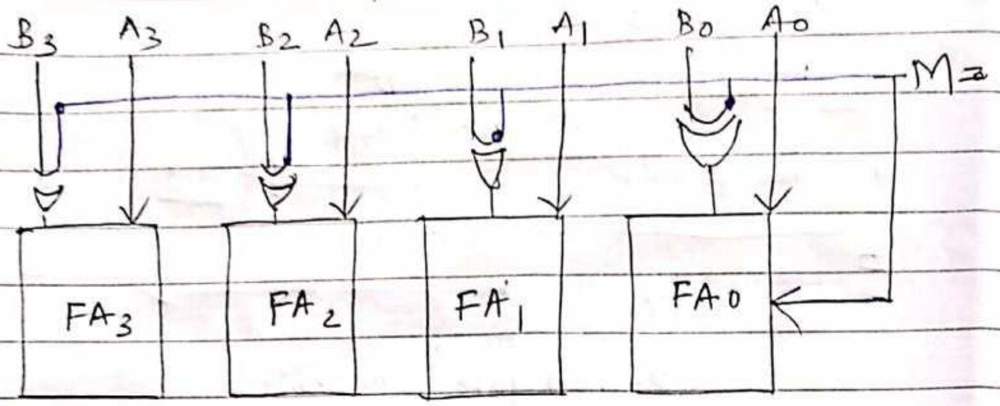
$+ 1$

Binary Subtractor using parallel Adder =



## 4-bit Adder - Subtractor =




M → Mode i/p controls the operation.

M = 0 → adder.

M = 1 → subtractor.

- **Binary Multiplier** =

→ It is a combinational circuit that performs multiplication of two binary Numbers".

A × B

multiplicand ← A, B → multiplier.

| | | | |
|---|---|---|---|
| | 2 | 3 | × 12 |
| | 4 | 6 | (partial product) |
| 2 | 3 | | |
| 2 | 7 | 6 | (final product) |

## 2-bit by 2-bit Binary multiplier =

$A = \quad A_1 \quad A_0$

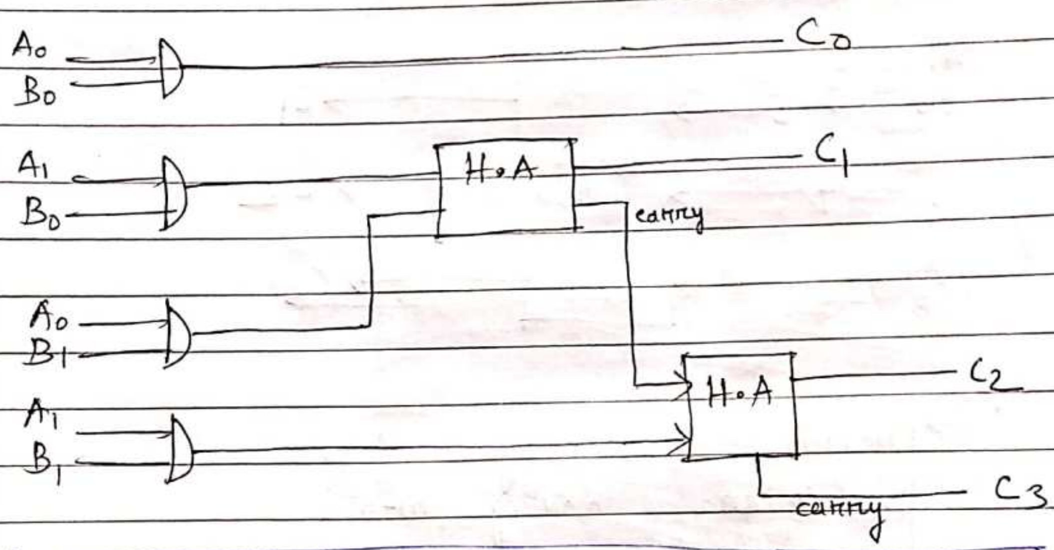
$B = \quad B_1 \quad B_0$

$n_1$(carry) $A_1B_0$ $A_0B_0$ (partial product) +

$c_3$ $A_1B_1$ $A_0B_1$

$C_3 \quad C_2 \quad C_1 \quad C_0$ (final product)

## Implimentation of Binary multiplier using Combinational circuit =




** → If multiplier (B) having n bits
multiplicand (A) having m bits

then it produces a product of (n+m) bits.
2 + 2 = 4 bit

** → AND gate required = (n × m)
= 2 × 2 = 4 NAND gate.

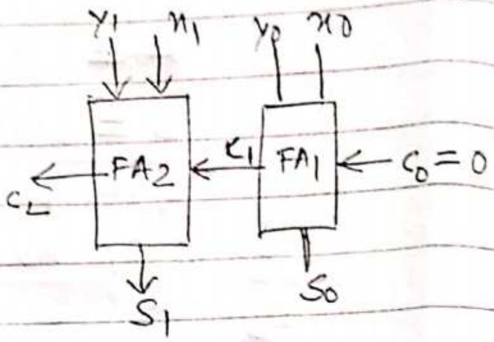
- **Binary multiplier using parallel Adder** =

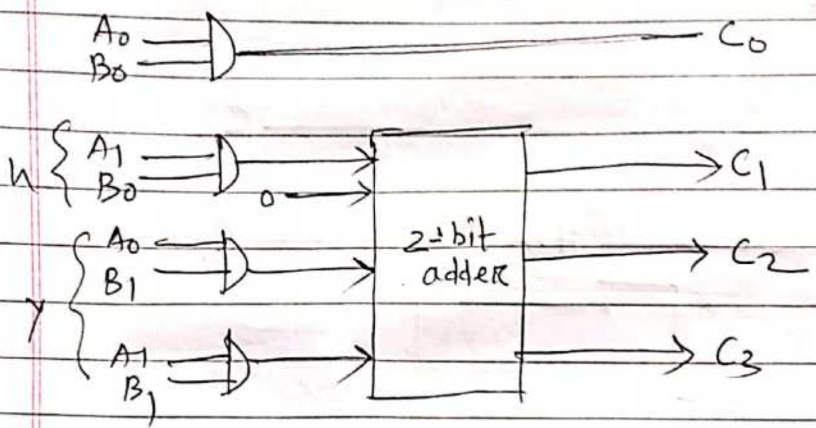
$A = A_1 \quad A_0$

$B = B_1 \quad B_0$

parallel adder $\begin{cases} x = 0 \quad A_1B_0 \quad A_0B_0 \\ y = A_1B_1 \quad A_0B_1 \end{cases}$

$\downarrow C_0$




**→ multiplier is having n-bits
multipliend is having m-bits

Then, (n-1) m-bit adders.

(2-1) 2-bit

= 2-bit adders required

→ **3-bit by 2-bit Binary multiplier** =

| | | | | |
|---|---|---|---|---|
| | $A \rightarrow$ | $A_2$ | $A_1$ | $A_0$ |
| | $B \rightarrow$ | | $B_1$ | $B_0$ |
| | | $A_2B_0$ | $A_1B_0$ | $A_0B_0$ |
| $A_2B_1$ | $A_1B_1$ | $A_0B_1$ | | |
| $C_4$ $C_3$ | $C_2$ | $C_1$ | | $C_0$ |

(1) final product contains 5 bits.
→ (3+2) = 5

(2) AND gate required 6
→ 3×2 = 6

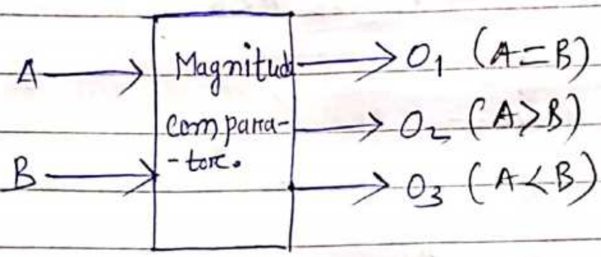
(3) how many binary parallel adder required
(n-1) m-bit
= (2-1)3 = 3 bit

not important for gate.

# Magnitude Comparator =

"A magnitude Comparator is a combinational ~~circuit~~ circuit that Compare two binary numbers A & B".

A = 1 1 0 0
B = 1 0 1 1




## 2-bit magnitude comparator =

$A = A_1 \quad A_0$
$B = B_1 \quad B_0$

**A=B** : all pairs of significant digits are equal.
$A_1 = B_1 \quad A_0 = B_0$

EX-NOR → (Both i/ps same)

$$x_i = A_i B_i + \overline{A_i}\, \overline{B_i}$$

$x_i = 1$, only if the pair of digits are equal in ith possition.

A = 1 1
B = 1 1

$$O_1(A=B) = x_1 \cdot x_0 \quad \text{—— (1)}$$
$= 1 \quad 1 = 1$ (yes both are equal)

**A>B**

$A_1 \; A_0$
A = 1 0
B = 0 1
$B_1 \; B_0$

$$O_2(A>B) = A_1 \overline{B_1} + x_1 A_0 \overline{B_0}$$
$1 \quad 1 + (\;)$
$= 1 + (\;)$
$= 1$ (Satisfy that A>B)

$A_1 \; A_0$

$A = 0 \; 1$

$B = 1 \; 0$

$B_1 \; B_0$

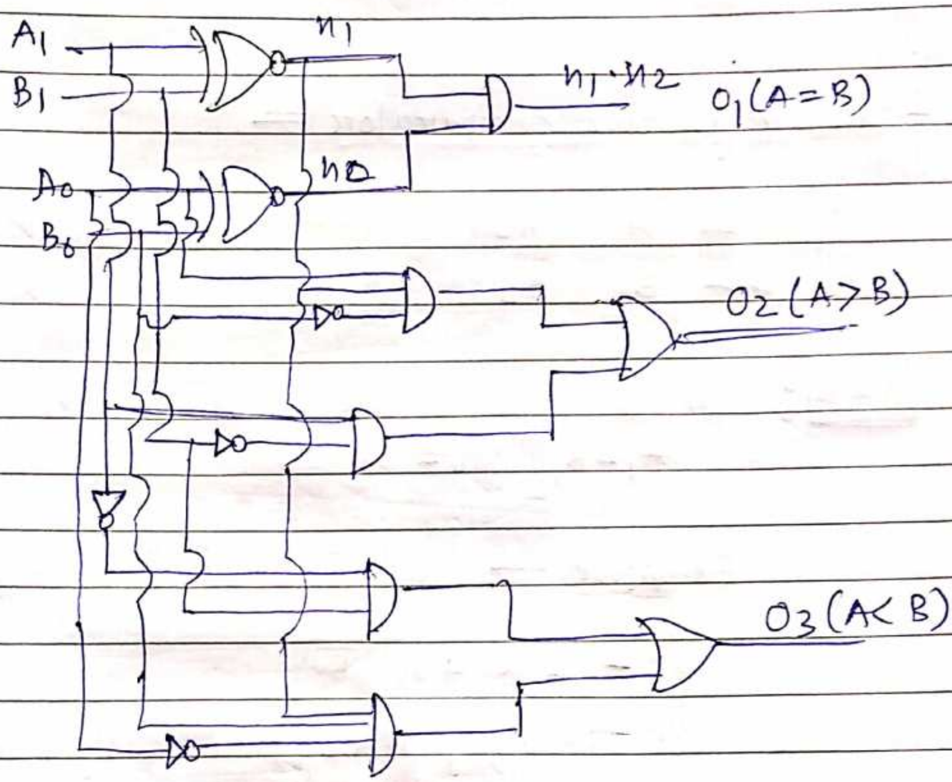
**A<B :**

$$O_3(A<B) = \bar{A}_1 B_1 + x_1 \bar{A}_0 B_0$$

$$= 1 + ()$$

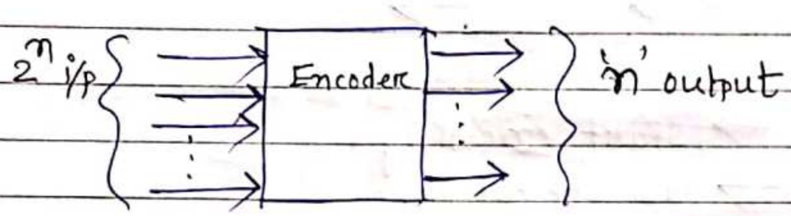
$$= 1 \text{ (yes } A<B)$$

logic diagram —



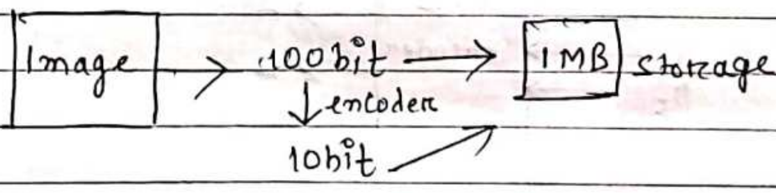
# • Encoders :

→ "It is a combinational circuit that takes multiple i/p and converts into a single binary code."

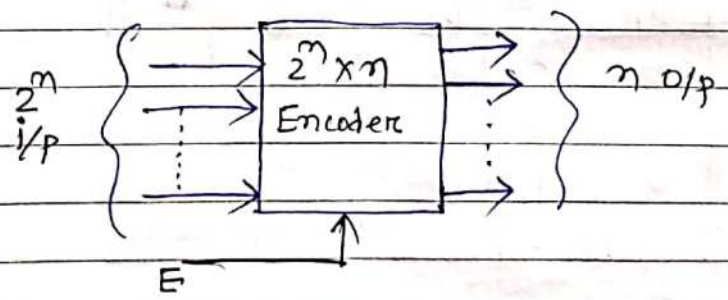



## Advantages :-

→ To store more data in a given space.
→ To send more data from source to destination.
→ To detect the errors easily.



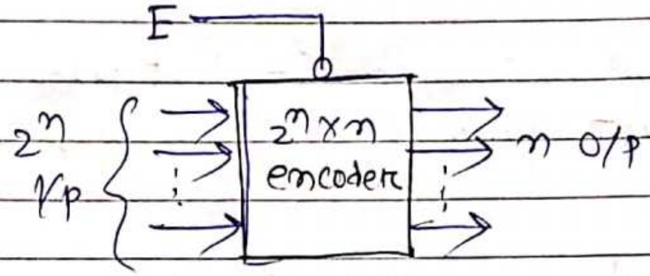

## Encoder with Enable input =




**Active-high**

E=1 → start Encoding (enable the circuit)
E=0 → no operation (Disable the circuit)

**Active-low**

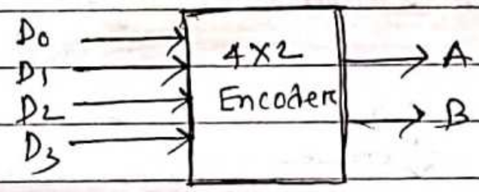



E=0 → start encoding.
E=1 → No operation.

**example** - 1

① 4 by 2 Encoder :

4×2 Encoder




Truth Table =

| $D_0$ | $D_1$ | $D_2$ | $D_3$ | A | B |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |

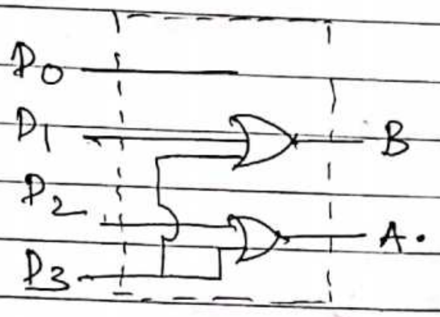
→ only one i/p has a value of '1' at any given time.

Logic expression

$$A = D_2 + D_3$$

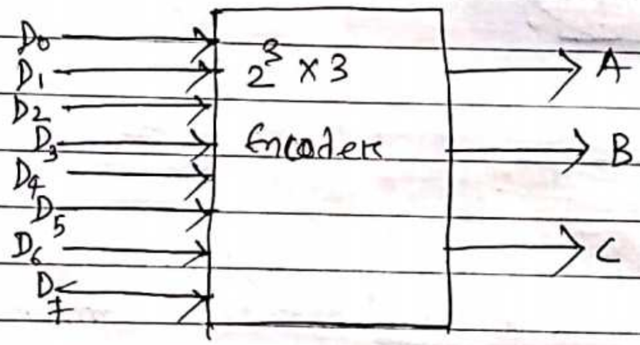
$$B = D_1 + D_3$$

## Logic Circuit =




## ② 8 X 3 Encoder (Octal to binary Encoder)

$8 = 2^3 = 3$ O/P




Truth table =

| D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | A | B | C |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

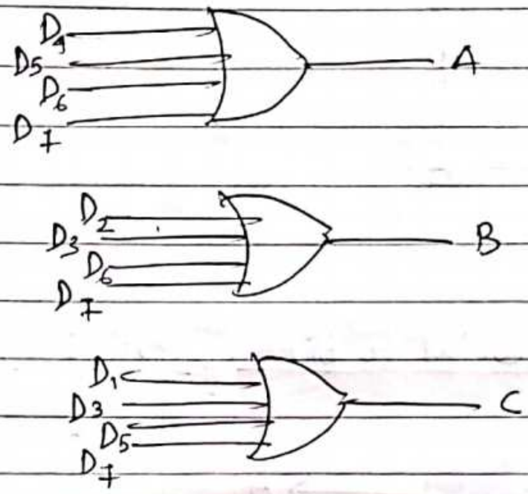
Logic expression —

$A = D_4 + D_5 + D_6 + D_7$

$B = D_2 + D_3 + D_6 + D_7$

$C = D_1 + D_3 + D_5 + D_7$

## Logic circuit —




• **Limitation** =

(1) If two Input are activate simultaneously the o/p of the encoder will be undefined combination.

ex- $D_2=1$, $D_4=1$      A B C

010    100      1 1 0 ($D_6$)

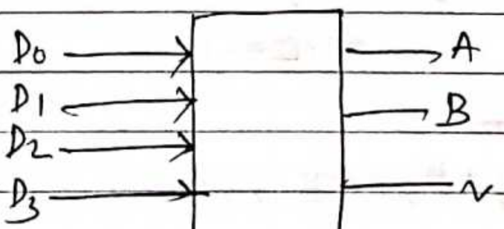
Solution → establish a I/p priority.

(II) whe all inputs are '0'

the o/p = 000 (A B C)

$D_0=1$ → 000      Solution →

• **4 X 2 priority Encoder** =

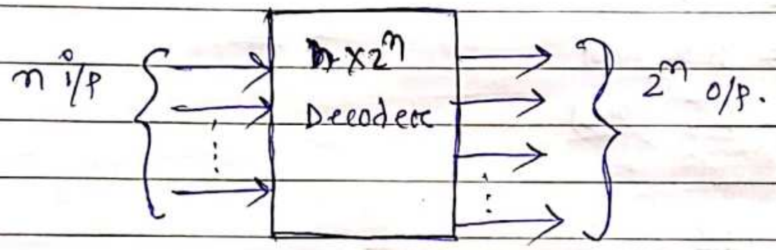



(valid bit indicator = 1 when one or more input are eavalute)

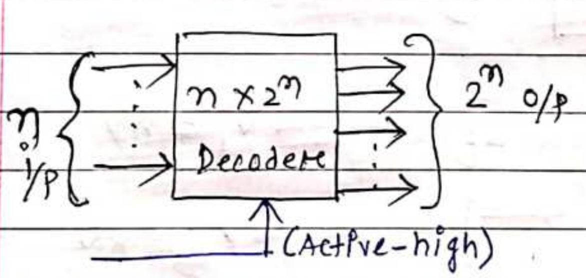
| $D_0$ | $D_1$ | $D_2$ | $D_3$ | A | B | V |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | X | X | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| X | 1 | 0 | 0 | 0 | 1 | 1 |
| X | X | 1 | 0 | 1 | 0 | 1 |
| X | X | X | 1 | 1 | 1 | 1 |

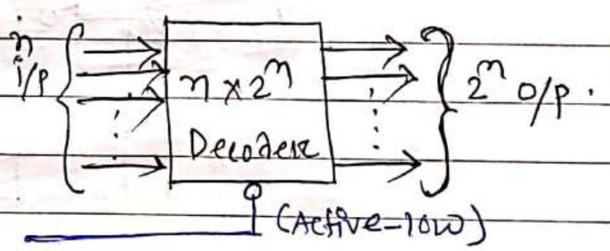
## • Decoder :

→ "A decoder is a combinational circuit that converts binary information from 'n' input lines to a max '$2^n$' unique output lines."




### • Decoder with enable input—

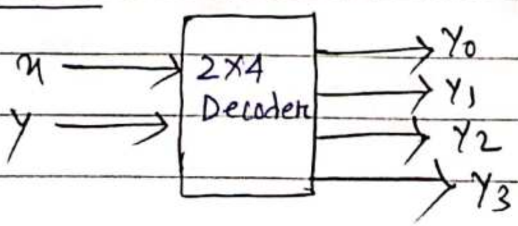



E=1 → enable the circuit
E=0 → disable the circuit

E=0 → enable the circuit
E=1 → disable the circuit.

### • 2 X 4 Decoder =



truth table –

| | x | y | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
|---|---|---|---|---|---|---|
| (0) | 0 | 0 | 1 | 0 | 0 | 0 |
| (1) | 0 | 1 | 0 | 1 | 0 | 0 |
| (2) | 1 | 0 | 0 | 0 | 1 | 0 |
| (3) | 1 | 1 | 0 | 0 | 0 | 1 |

logic expressions =

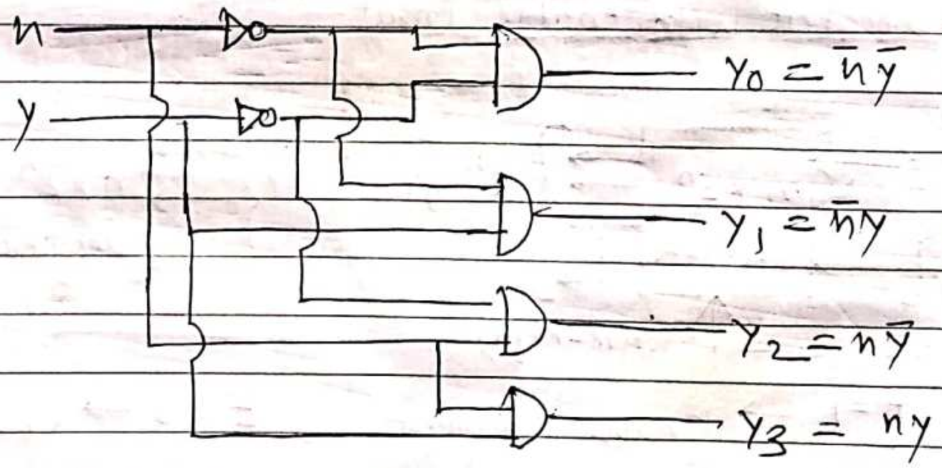
$Y_0 = \bar{x}\bar{y}$ $(m_0)$

$Y_1 = \bar{x}y$ $(m_1)$

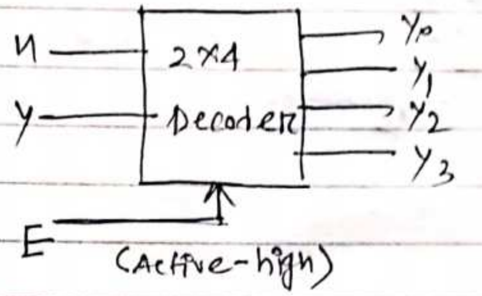
$Y_2 = x\bar{y}$ $(m_2)$

$Y_3 = xy$ $(m_3)$

logic diagram =



## 2×4 Decoder with Enable i/p –



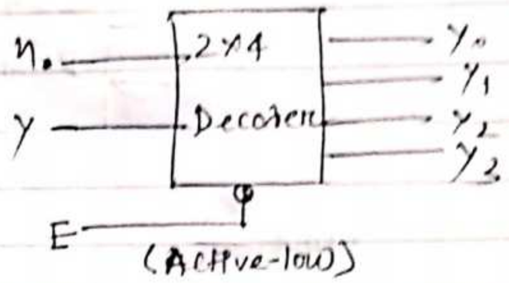




**table** (Active-high)

| E | x | y | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
|---|---|---|---|---|---|---|
| 0 | X | X | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

**table** (Active-low)

| E | x | y | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
|---|---|---|---|---|---|---|
| 1 | X | X | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |  | 1 |

logic expression –

$Y_0 = \bar{x}\bar{y} \cdot E$

$Y_1 = \bar{x}y \cdot E$

$Y_2 = x\bar{y} \cdot E$

$Y_3 = xy \cdot E$

(Active-low)

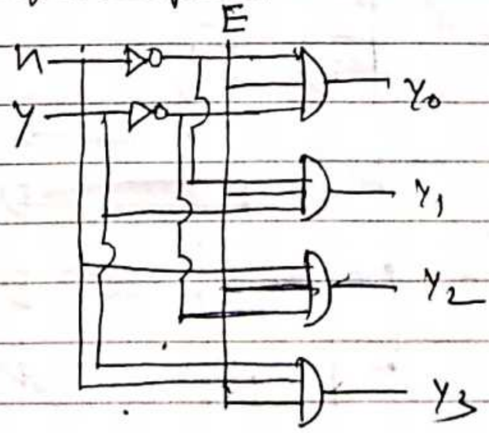
$Y_0 = \bar{x}\bar{y} \cdot \bar{E}$

$Y_1 = \bar{x}y \cdot \bar{E}$

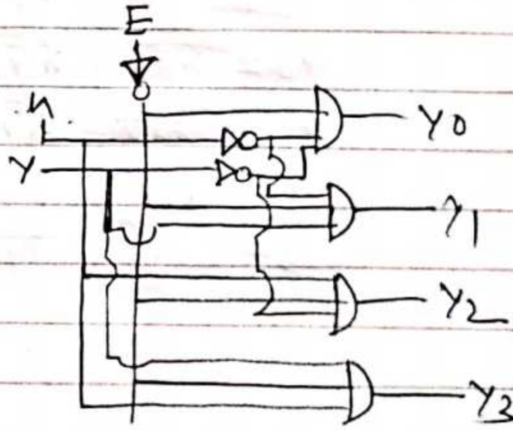
$Y_2 = x\bar{y} \cdot \bar{E}$

$Y_3 = xy \cdot \bar{E}$

logic diagram –




logic diagram –



## • 2×4 Decoder with active low output —

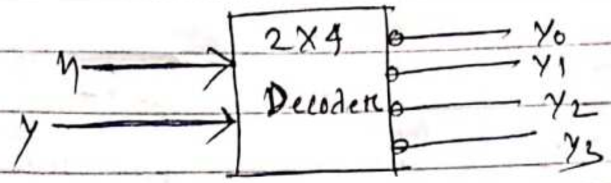



table —

| x | y | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

expression —

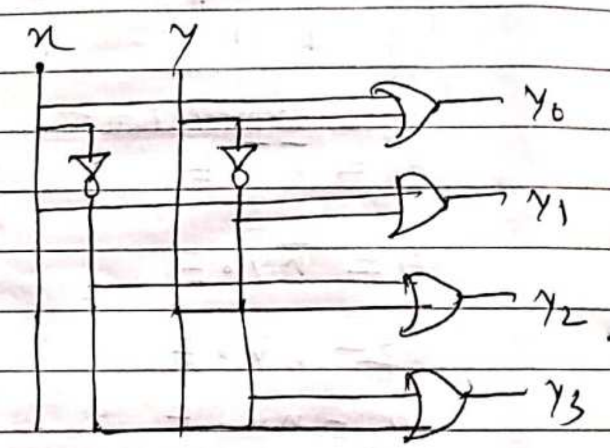
$Y_0 = x + y$

$Y_1 = x + \bar{y}$

$Y_2 = \bar{x} + y$

$Y_3 = \bar{x} + \bar{y}$

logic diagram —



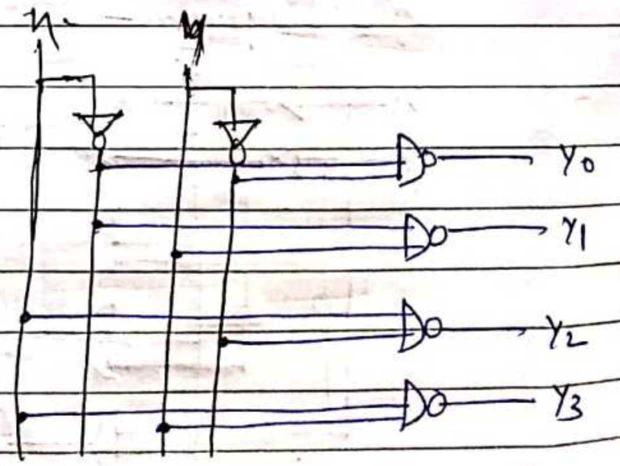

using NAND gates —

$Y_0 = \overline{\overline{x+y}} = \overline{\bar{x} \cdot \bar{y}}$

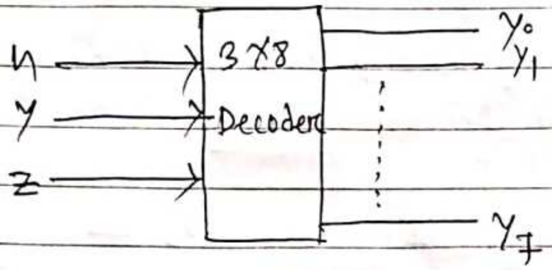
$Y_1 = \overline{\overline{x+\bar{y}}} = \overline{\bar{x} y}$

$Y_2 = \overline{\overline{\bar{x}+y}} = \overline{x \bar{y}}$

$Y_3 = \overline{\overline{\bar{x}+\bar{y}}} = \overline{x y}$



## 3X8 Decoder —




### Table —

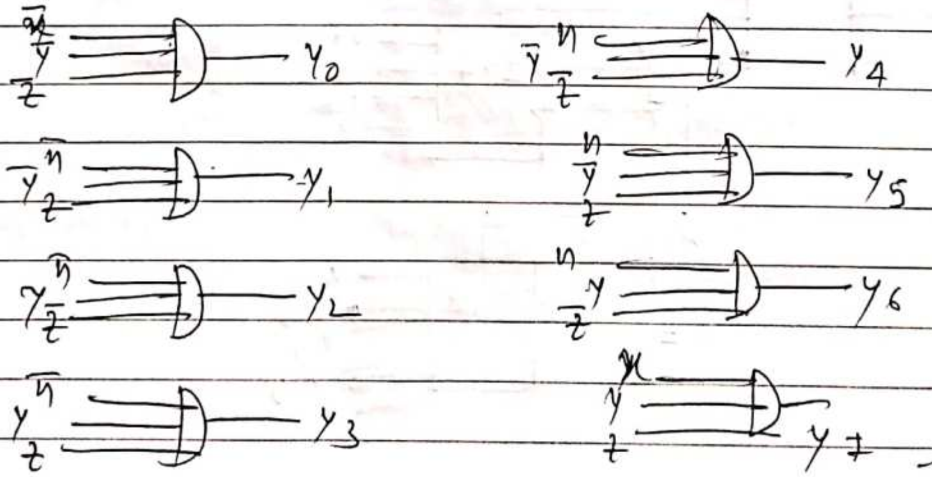
| | x | y | z | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | $Y_5$ | $Y_6$ | $Y_7$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| (0) | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| (1) | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| (2) | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| (3) | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| (4) | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| (5) | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| (6) | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| (7) | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

### logic expression —

$Y_0 = \bar{x}\bar{y}\bar{z}$   $Y_1 = \bar{x}\bar{y}z$   $Y_2 = \bar{x}y\bar{z}$   $Y_3 = \bar{x}yz$   $Y_4 = x\bar{y}\bar{z}$

$Y_5 = x\bar{y}z$   $Y_6 = xy\bar{z}$   $Y_7 = xyz$

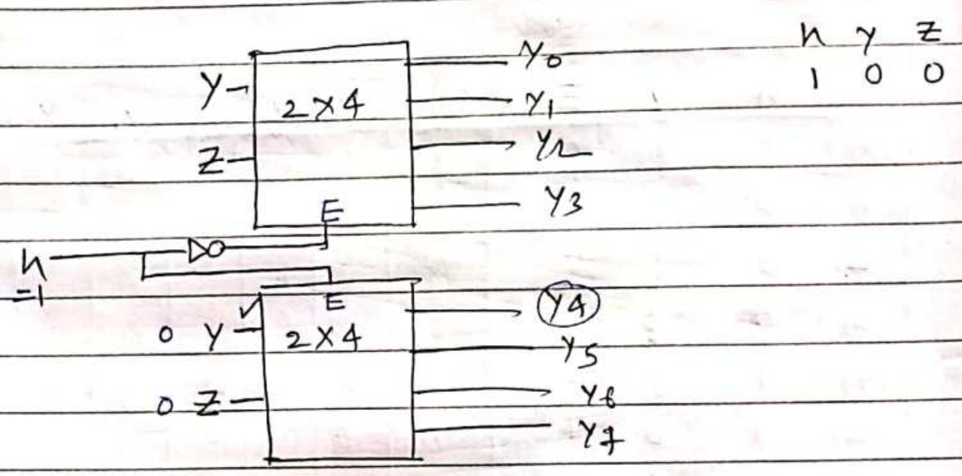
### logic diagram



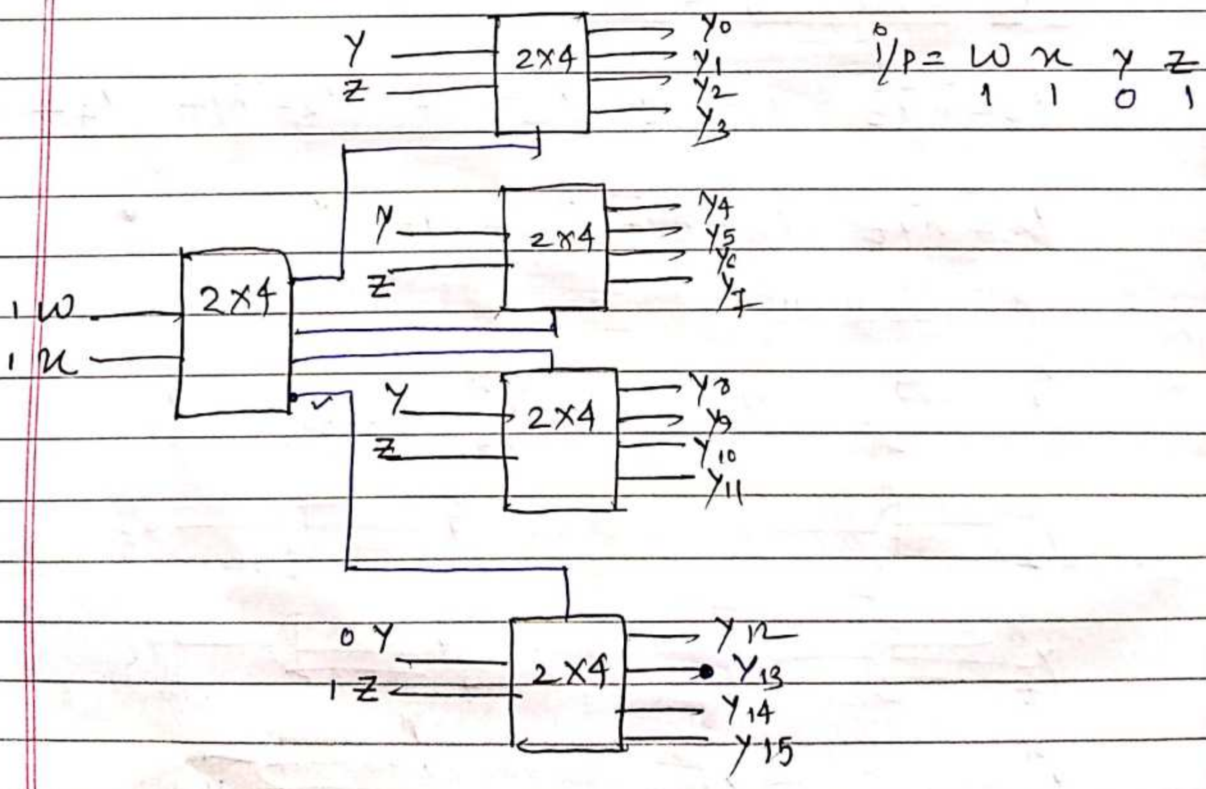
• Construction of larger Decoders using smaller decoders –

" Decoder with enable i/p can be connected together to form larger Decoder circuits.

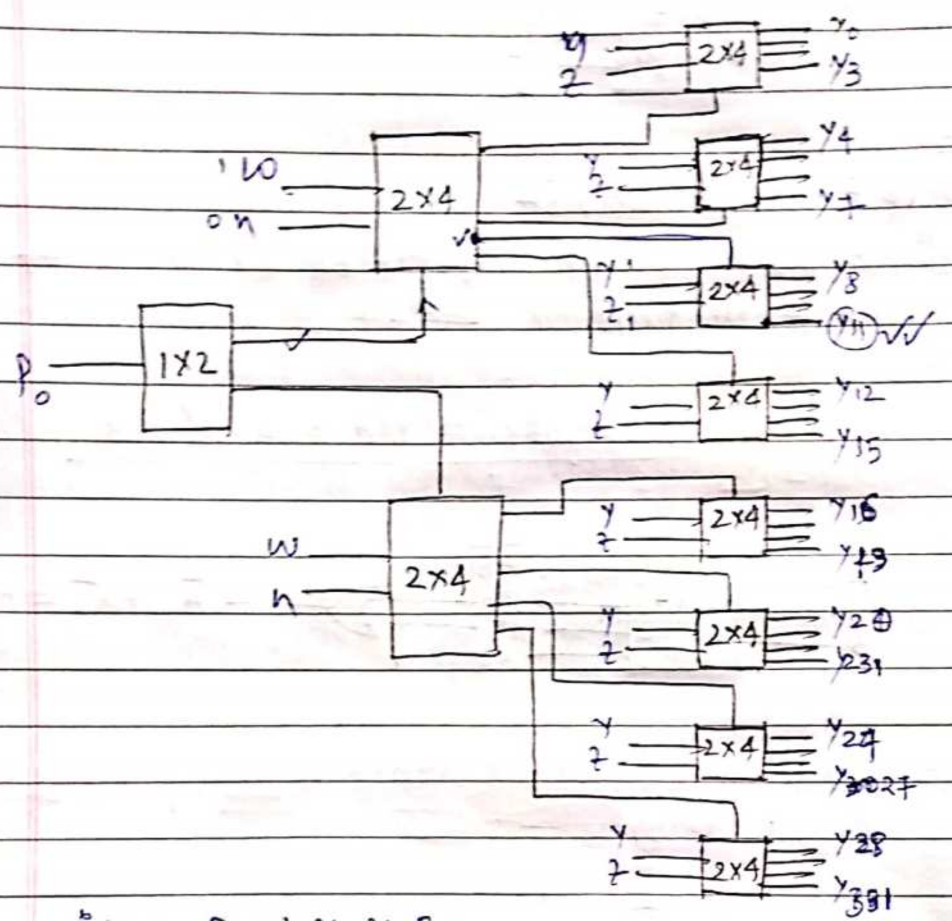
① 3×8 Decoder using 2×4 Decoders –




② 4×16 Decoder using 2×4 Decoder –

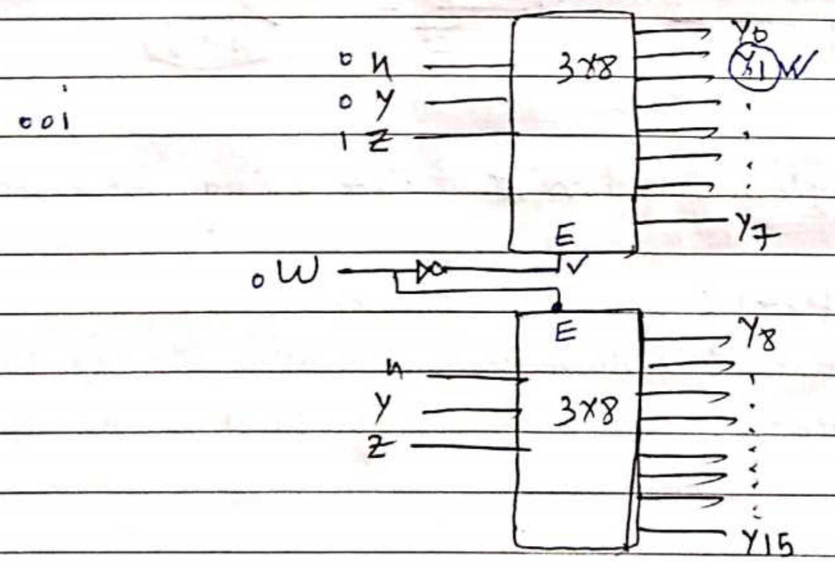


③ 5×32 Decoder using 2×4 Decoder — 3×8 / 4



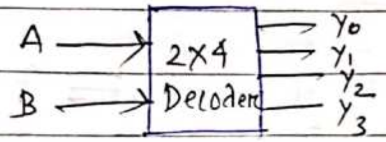

i/p = P W x Y Z
0 1 0 1 1

④ 4×16 Decoder using 3×8 Decoder — 16/8 = 2




i/p = W x Y Z
0 0 0 1

## • Implementation of Boolean functions using Decoders —

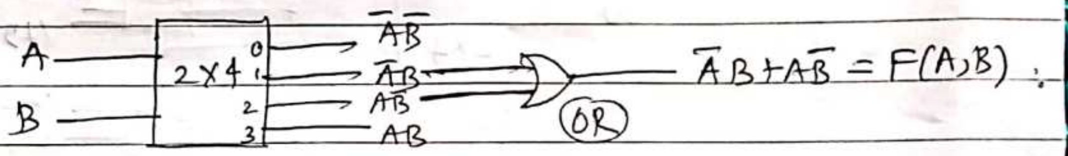



ex =

① $F(A,B) = B\bar{A} + \bar{B}A$

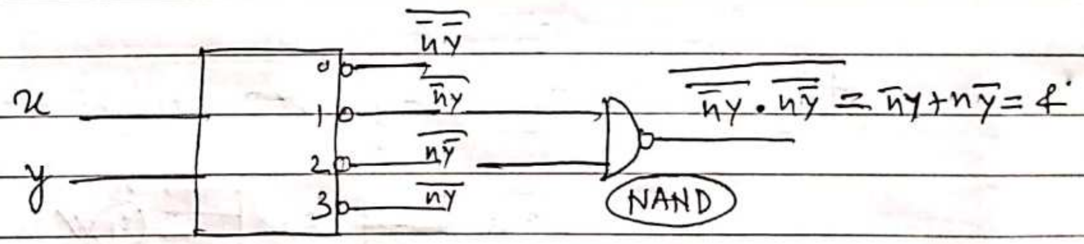
"Any Boolean function can be expressed ~~by~~ as sum of products or ~~product of~~ sum of minterms form".

Decoder + OR gate = f(A, B) or f(A, B, C)




** ② Implement the EX-OR function using active low output decoder —

→ $f = x\bar{y} + \bar{x}y$  $f = \Sigma(1,2)$




③ Implementation ~~of~~ of F.A using 3×8 Decoder —

Step-1 : write truth table.

Step-2 : Obtain Boolean function in SOP form.

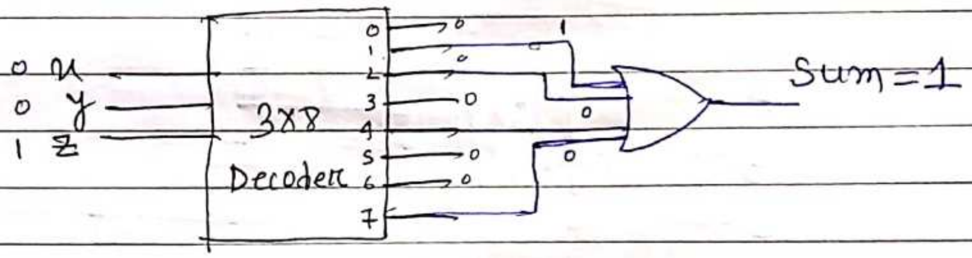
Step-3 : Implement using Decoder and OR gate.

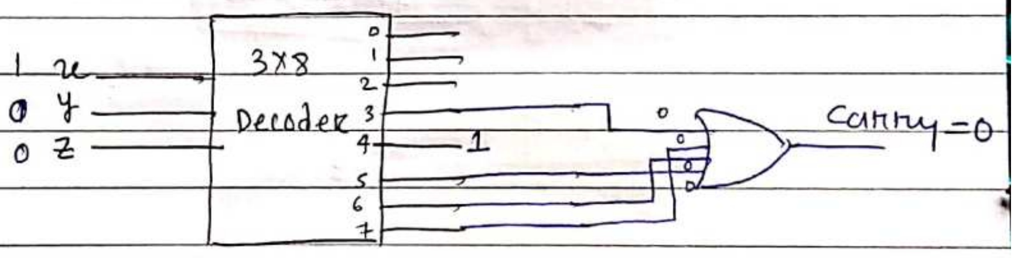
Scanned by CamScanner

table

| | x | y | z | sum | carry |
|---|---|---|---|---|---|
| (0) | 0 | 0 | 0 | 0 | 0 |
| (1) | 0 | 0 | 1 | 1 | 0 |
| (2) | 0 | 1 | 0 | 1 | 0 |
| (3) | 0 | 1 | 1 | 0 | 1 |
| (4) | 1 | 0 | 0 | 1 | 0 |
| (5) | 1 | 0 | 1 | 0 | 1 |
| (6) | 1 | 1 | 0 | 0 | 1 |
| (7) | 1 | 1 | 1 | 1 | 1 |

$sum = \Sigma(1, 2, 4, 7)$.
$carry = \Sigma(3, 5, 6, 7)$



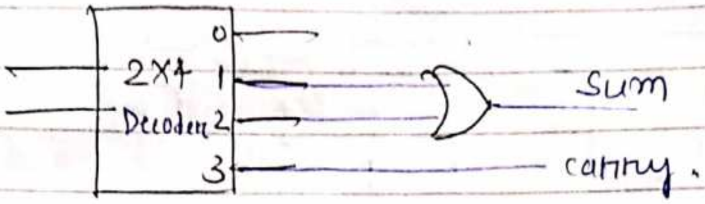

(worked)

④ Implimentation of H.A using Decoder—

table

| | A | B | S | C |
|---|---|---|---|---|
| (0) | 0 | 0 | 0 | 0 |
| (1) | 0 | 1 | 1 | 0 |
| (2) | 1 | 0 | 1 | 0 |
| (3) | 1 | 1 | 0 | 1 |

$Sum = \Sigma(1, 2)$
$carry = \Sigma(3)$.




⑤ Implement the following function using Decoder —

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

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

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

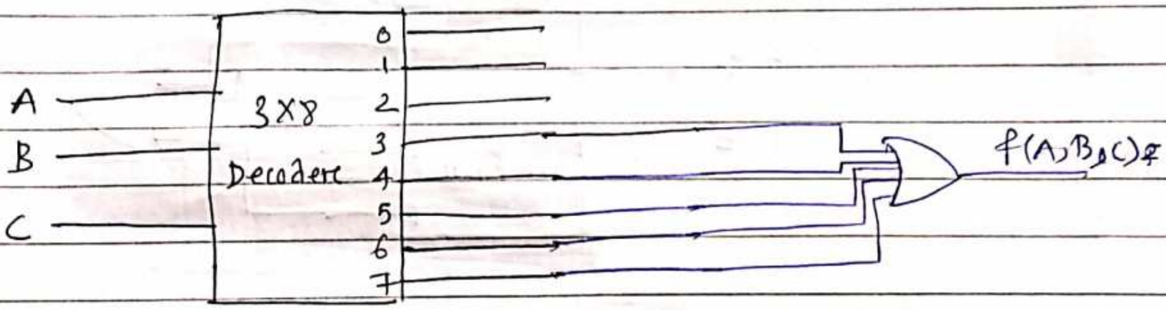
111   110   101   100   111   011

4 2 1

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

$(m_7)$   $(m_6)$   $(m_5)$   $(m_4)$   $(m_3)$

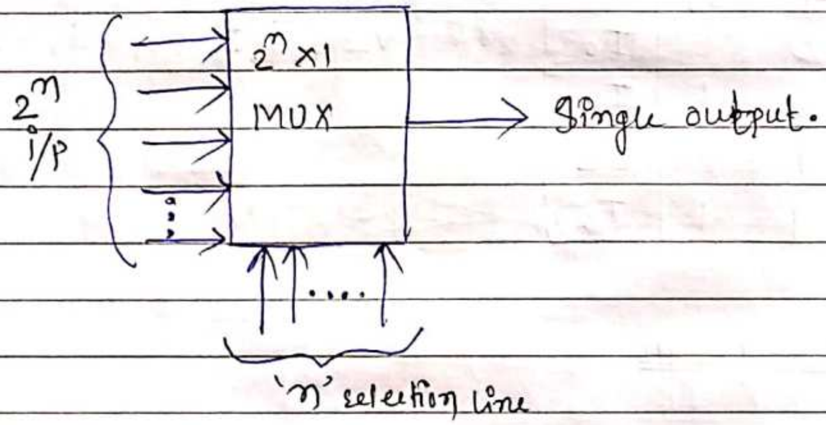
$= \Sigma(3,4,5,6,7)$



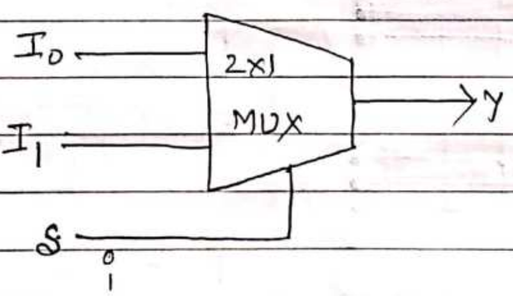
## ** • Multiplexer : (MUX) or (Data selector)

"A Multiplexer is a Combinational circuit that select binary information from one of many i/p lines and directs it to a single O/p line".

The selection of a particular i/p controled by a set of selection lines".




### • 2×1 Multiplexer (MUX) —

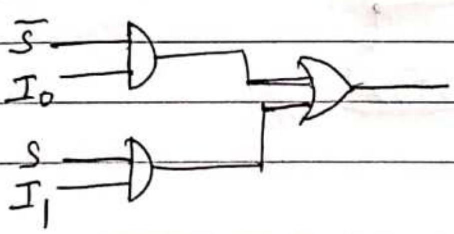



**Truth Table:**

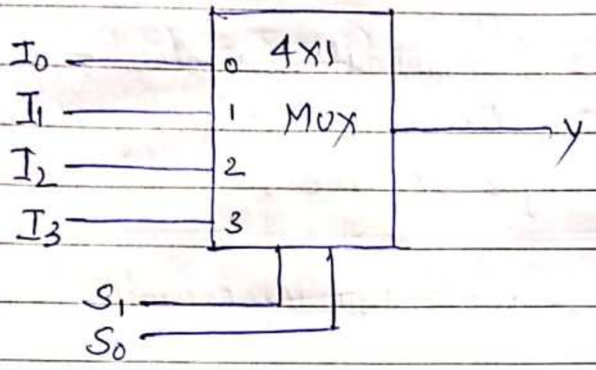
| S | Y |
|---|---|
| 0 | $I_0$ |
| 1 | $I_1$ |

$$Y = \bar{S} I_0 + S I_1$$

**logic Diagram:**



## • 4×1 MUX :



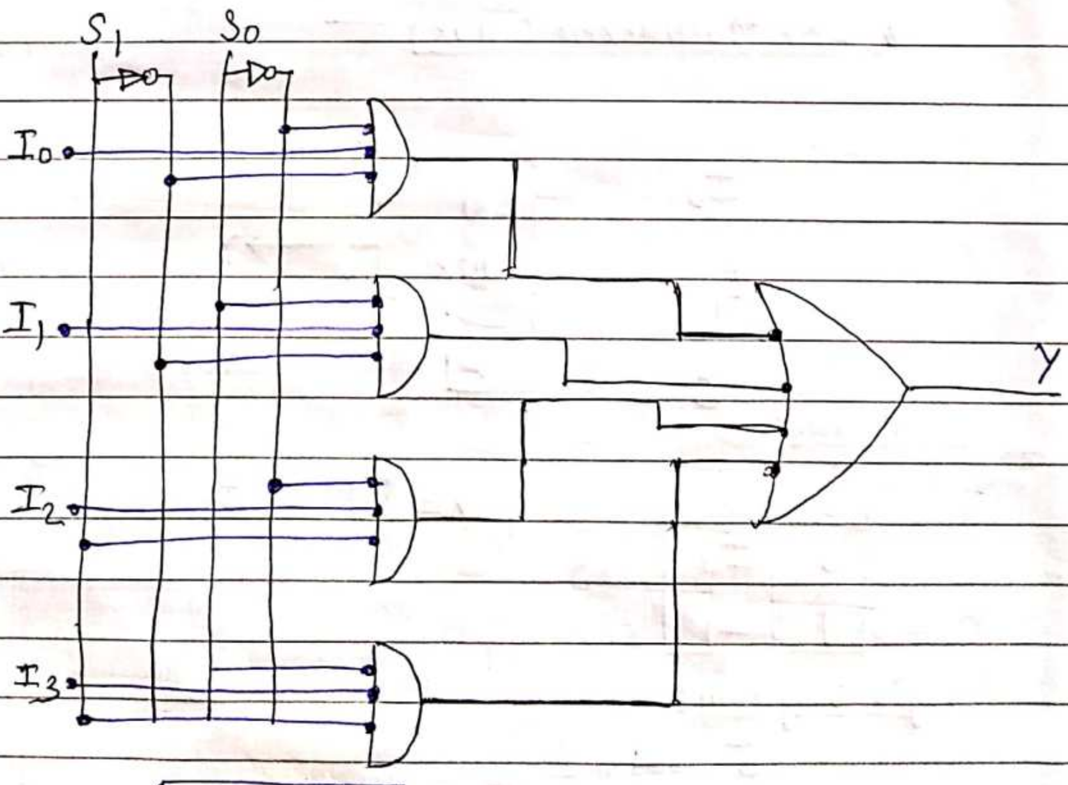

Table

| | $S_1$ | $S_0$ | $y$ |
|---|---|---|---|
| (0) | 0 | 0 | $I_0$ |
| (1) | 0 | 1 | $I_1$ |
| (2) | 1 | 0 | $I_2$ |
| (3) | 1 | 1 | $I_3$ |

expression –

$$y = \bar{S_1}\bar{S_0}I_0 + \bar{S_1}S_0I_1 + S_1\bar{S_0}I_2 + S_1S_0I_3$$

logic diagram

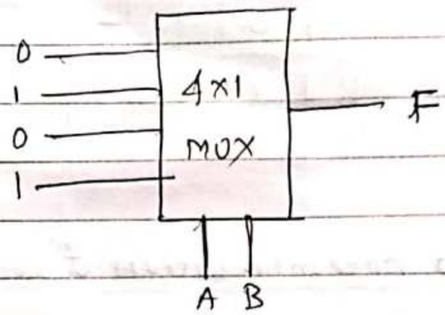



4×1 MUX

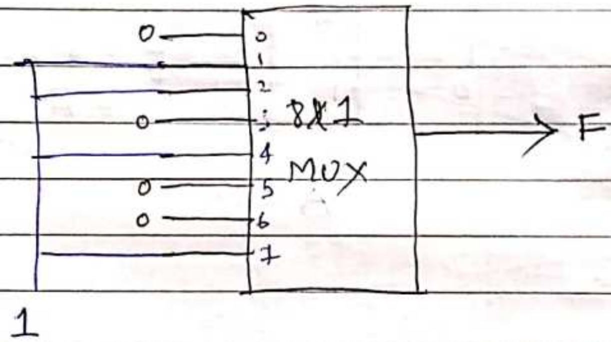
- **Implementation of Boolean function using MUX:**

① $F(A,B) = \Sigma(1,3)$ using 4×1 MUX.

| | A | B | F |
|---|---|---|---|
| (0) | 0 | 0 | 0 |
| (1) | 0 | 1 | 1 |
| (2) | 1 | 0 | 0 |
| (3) | 1 | 1 | 1 |

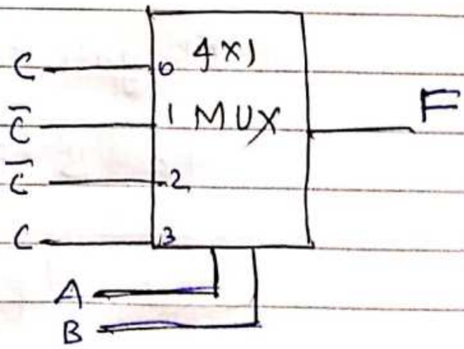



② $F(A,B,C) = \Sigma(1,2,4,7)$ using 8×1 MUX -




③ $F(A,B,C) = \Sigma(1,2,4,7)$ using 4×1 MUX -

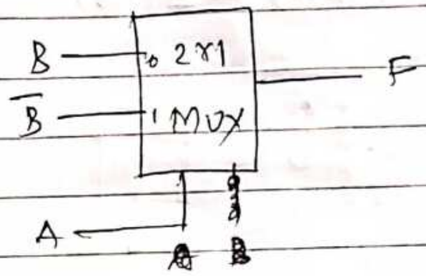
| | A | B | C | F | |
|---|---|---|---|---|---|
| (0) | 0 | 0 | 0 | 0 | |
| (1) | 0 | 0 | 1 | 1 | C |
| (2) | 0 | 1 | 0 | 1 | $\bar{C}$ |
| (3) | 0 | 1 | 1 | 0 | |
| (4) | 1 | 0 | 0 | 1 | $\bar{C}$ |
| (5) | 1 | 0 | 1 | 0 | |
| (6) | 1 | 1 | 0 | 0 | |
| (7) | 1 | 1 | 1 | 1 | C |




A B C'
0 0 0

(4) using 2X1 MUX –

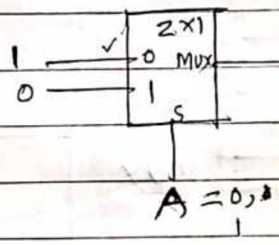
| A | B | F | |
|---|---|---|---|
| 0 | 0 | 0 | B |
| 0 | 1 | 1 | |
| 1 | 0 | 1 | $\overline{B}$ |
| 1 | 1 | 0 | |




• Implementation of all logic gates using 2X1 MUX –

(I) NOT gate –

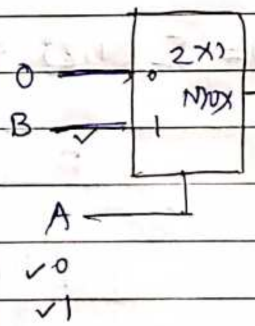
| A | F |
|---|---|
| 0 | 1 |
| 1 | 0 |




$F = \overline{A}.1 + A.0$
$= \overline{A}$

(II) AND gate –

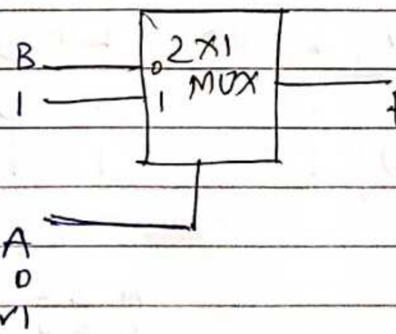
| | A | B | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| | 1 | 1 | 1 |




$F = \overline{A}.0 + AB$
$= AB$

(III) OR gate =

| A | B | F | |
|---|---|---|---|
| 0 | 0 | 0 | B |
| 0 | 1 | 1 | |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | |



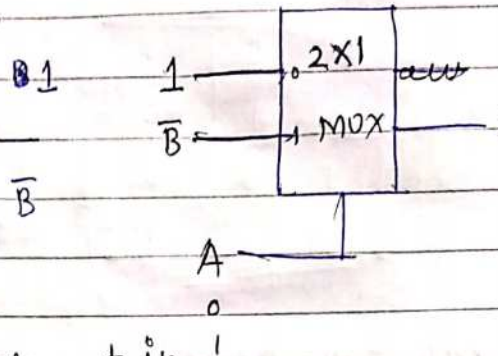

$F = \overline{A}B + 1A$
$= \overline{A}B + A$
$= A + B$

## (iv) NAND gate –

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

1

$\bar{B}$



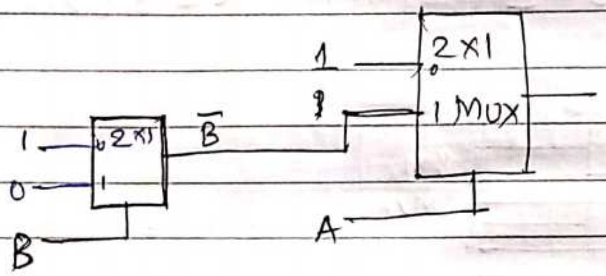

$F = \bar{A}\cdot 1 + A\cdot\bar{B}$

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

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

$= \overline{AB}$.

** using no complement i/p –

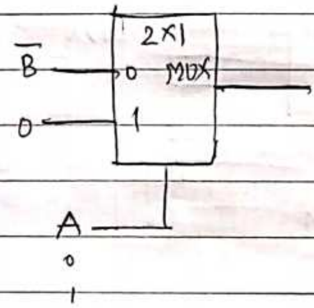



## (v) NOR gate –

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

0 → $\bar{B}$

1 → 0




$F = \bar{A}\,\bar{B} + 0\cdot A$

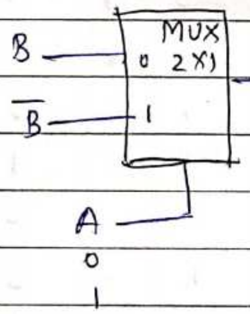
$= \overline{(A+B)}$.

## (vi) Ex-OR gate –

| A | B | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

B

$\bar{B}$

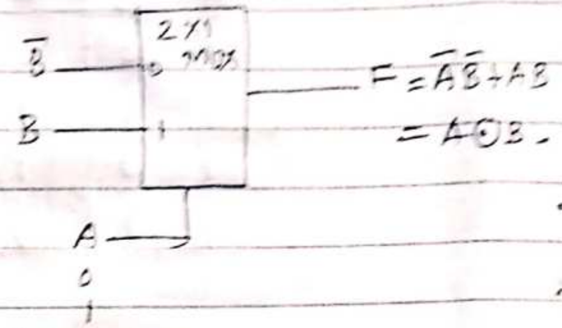



$F = \bar{A}B + A\bar{B}$

$= A \oplus B$.

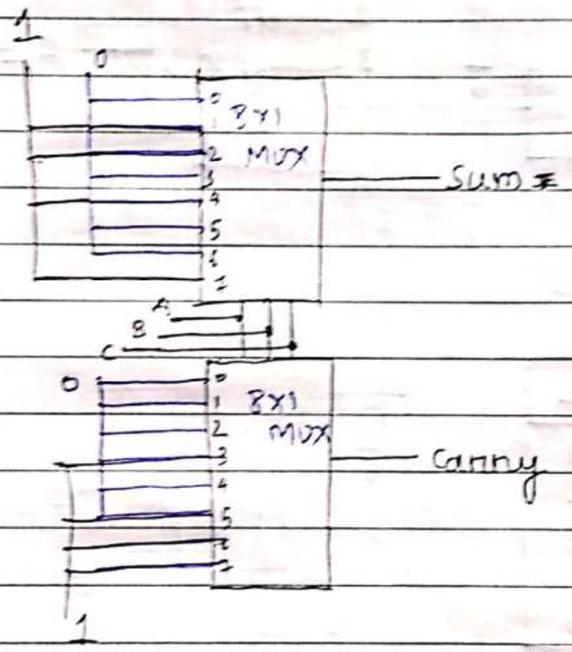
## (VII) EX-NOR gate —

| A | B | F | |
|---|---|---|---|
| 0 | 0 | 1 | $\bar{B}$ |
| 0 | 1 | 0 | |
| 1 | 0 | 0 | B |
| 1 | 1 | 1 | |




## (VIII) Full-Adder using 8×1 MUX —

$S = \Sigma(1,2,4,7)$ $C = \Sigma(3,5,6,7)$

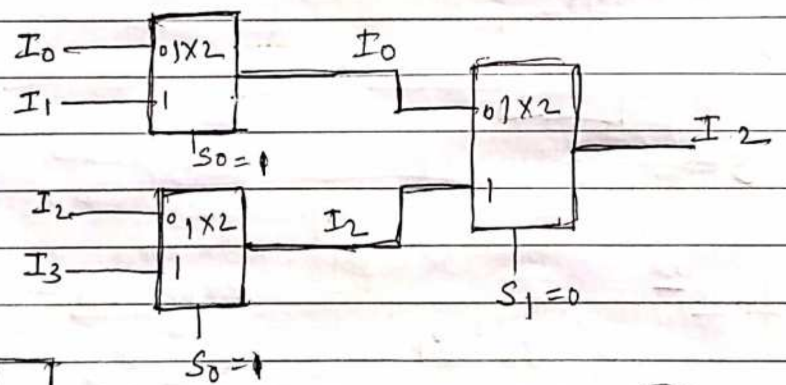



| A | B | C | S | Carry |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

## • Design of Large size multiplexer using Other MUX:

| Given MUX | To be Implemented | required no. of MUX |
|---|---|---|
| 2×1 | 4×1 | 3 |
| 2×1 | 8×1 | 7 |
| 2×1 | 16×1 | 15 |
| 2×1 | 64×1 | 63 |
| 2×1 | 256×1 | 255 |
| 2×1 | $2^n$×1 | $(2^n-1)$ |
| 4×1 | 16×1 | 4+1=5 |
| 4×1 | 64×1 | 16+4+1=21 |
| 8×1 | 64×1 | 8+1=9 |
| 8×1 | 256×1 | 32+4+1=37. |

① Implement 4×1 MUX using 2×1 MUX –




| $S_1$ | $S_0$ | F |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |

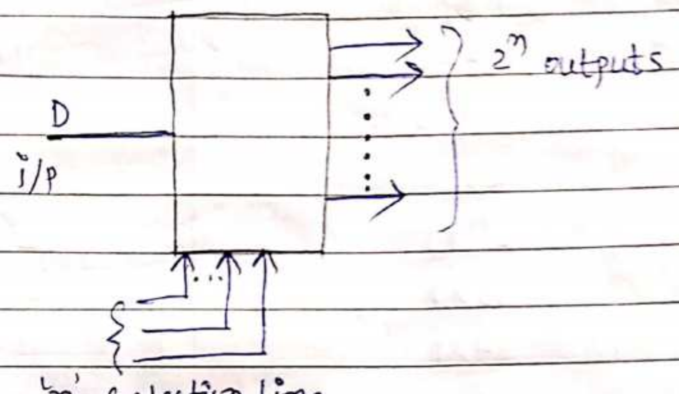
3 – (2:1) MUX required.

$S_1$ $S_0$

1 0 = $I_2$

## Demultiplexer : (De-MUX) (Data Distributor).

→ It performs the inverse of Multiplexer.
→ It take single i/p and distribute several outputs.




① 1x2 De-mux or 1 line to 2 line De-mux —

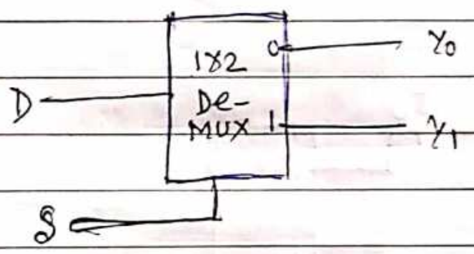



table —

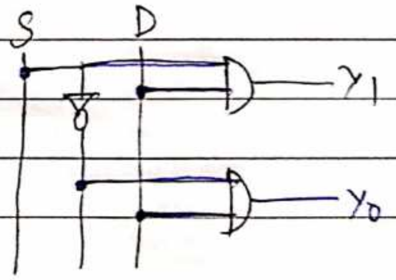
| S | $Y_0$ | $Y_1$ |
|---|---|---|
| 0 | D | 0 |
| 1 | 0 | D |

expression —

$Y_0 = \bar{S} D$

$Y_1 = S D$

logic diagram —



## ② 1x4 De-MUX with Enable i/p –

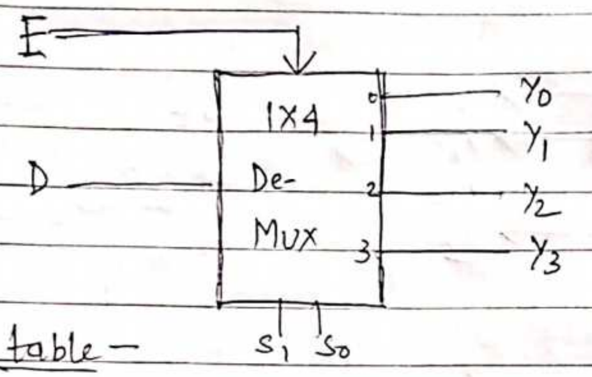


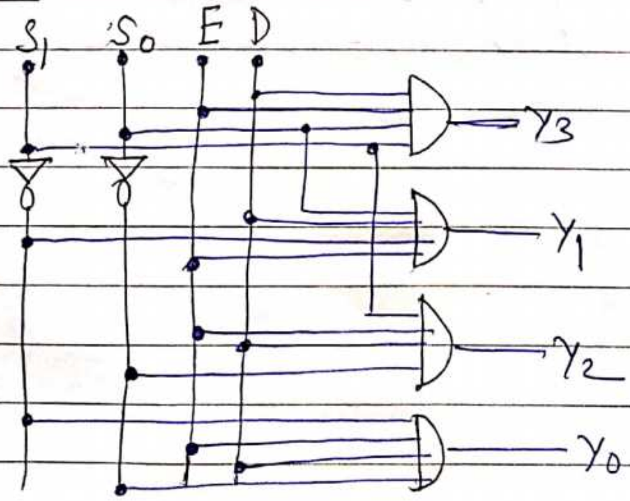

table –

| E | $S_1$ | $S_0$ | $Y_0$ | $Y_1$ | $Y_2$ | $Y_3$ |
|---|---|---|---|---|---|---|
| 0 | X | X | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | D | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | D | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | D | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | D |

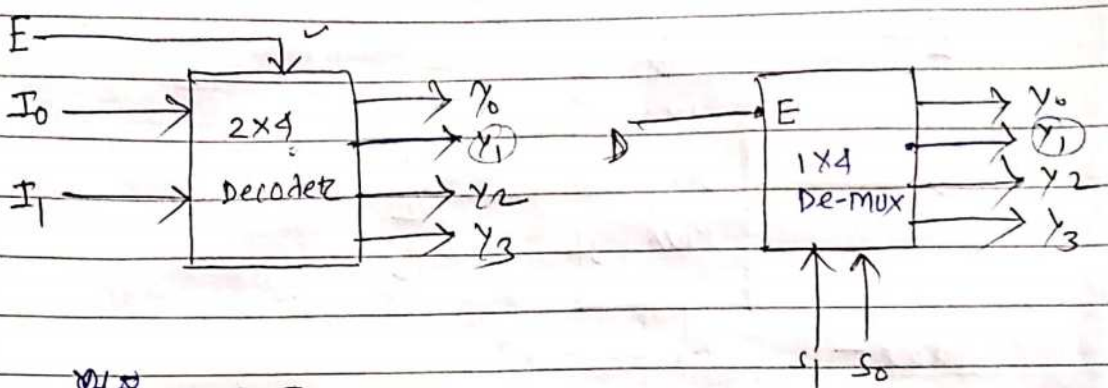
$Y_0$ expression –

$$Y_0 = E.\bar{S_1}\,\bar{S_0}.D$$
$$Y_1 = E.\bar{S_1}\,S_0.D$$
$$Y_2 = E.S_1\,\bar{S_0}.D$$
$$Y_3 = E.S_1\,S_0.D$$

Logic diagram –



③ Decoder with Enable pin can act as a De-MUX –

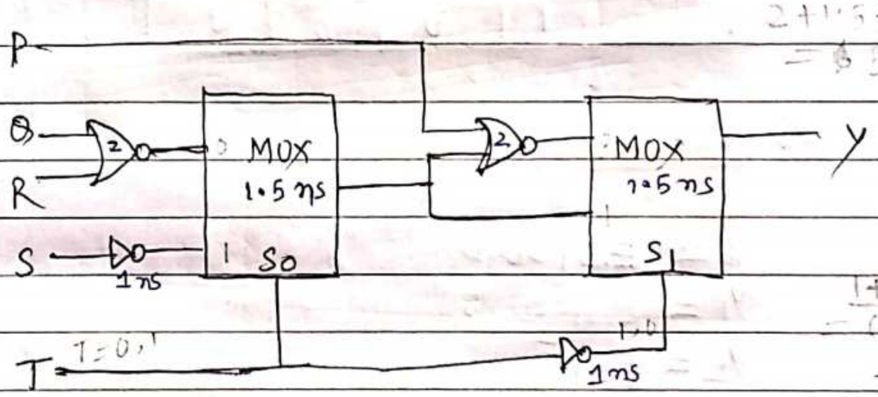



E $I_0$ $I_1$
1 0 1 = $Y_1$

D $S_1$ $S_0$
1 0 1 = $Y_1$

• GATE Questions on Combinational circuit –

Q–① 2016



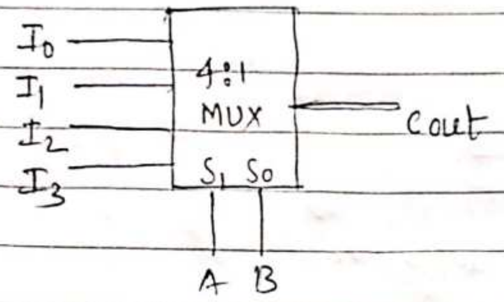

$T_0 = 0$
$2 + 1.5 + 1.5$
$= 5$ ns

$T_1 = 1$
$1 + 1.5 + \ldots$
$6$ ns.

The delay of NOR gates, MUX and inverters are 2 ns, 1.5 ns, and 1 ns respectively. If all the i/p P, Q, R, S applied at same time, the max propagation delay of the circuit is 6 ns.

2016-Gate

Q-(2)




Which one of the following statements correctly describes the choice of ~~singt~~ signals to be ~~converted~~ connected to the i/p's $I_0, I_1, I_2$ & $I_3$ so that the o/p is cout?

✓ (a) $I_0 = 0$, $I_1 = c_{in}$, $I_2 = C_{in}$, $I_3 = 1$,
(b) $I_0 = 1$, $I_1 = C_{in}$, $I_2 = c_{in}$, $I_3 = 1$,
(c) $I_0 = c_{in}$, $I_1 = 0$, $I_2 = 1$, $I_3 = c_{in}$,
(d) $I_0 = 0$, $I_1 = c_{in}$, $I_2 = 1$, $I_3 = C_{in}$,

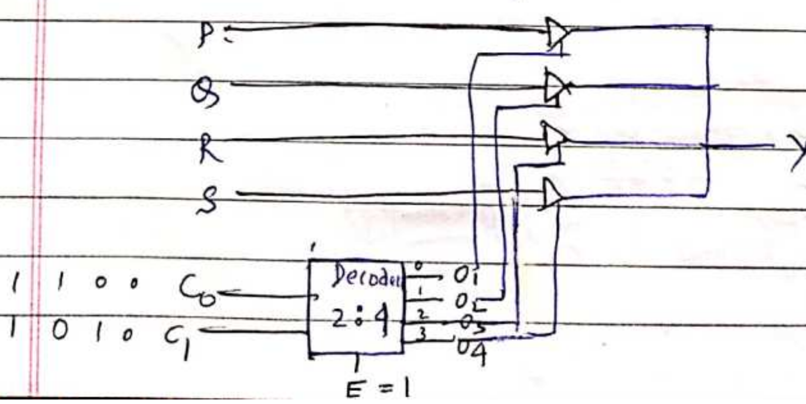
⇒

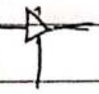
| A | B | $c_{in}$ | Cout (carry) | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 ($I_0$) |
| 0 | 0 | 1 | 0 | |
| 0 | 1 | 0 | 0 | $c_{in}$ ($I_1$) |
| 0 | 1 | 1 | 1 | |
| 1 | 0 | 0 | 0 | $c_{in}$ ($I_2$) |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | 1 | 1 ($I_3$) |
| 1 | 1 | 1 | 1 | |

G-16

Q-(3) The functionality implemented by the ckt below is

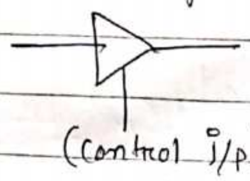
✓ (4:1 MUX)



is a tristate buffer.

**Three state gate** –



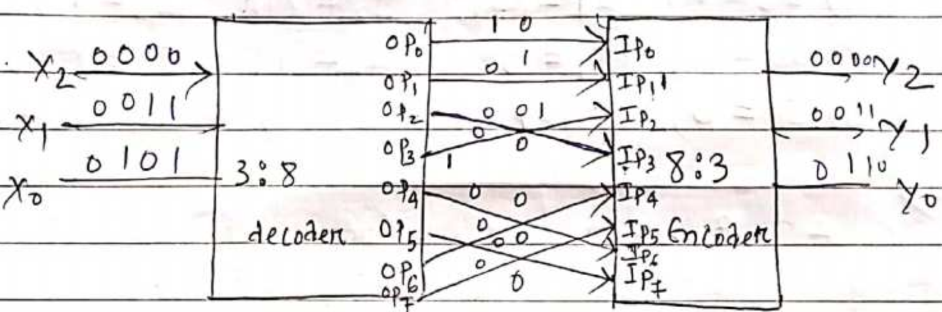
(control i/p)

if C=0, Y= high imp state
C=1, Y = A.

2016

**Q – ④**

identify the ckt below –




4 2 1
0 1 0

✓(a) Binary to Gray code covertor.
(b) Binary to $XS_3$ Converter.
(c) Gray to binary converter.

| $X_2$ | $X_1$ | $X_0$ | $Y_2$ | $Y_1$ | $Y_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

B → G

0 0 0 → 0 0 0
0 0 1 → 0 0 1
0 1 0 → 0 1 1
0 1 1 → 0 1 0

2014 Q-⑤

In a Half subtractor ckt with x & y as I/P, the Borrow (M) & Difference (N = x - y) are give by -

(a) $M = x \oplus y$, $N = xy$
(b) $M = xy$, $N = x \oplus y$
✓(c) $M = \bar{x}y$, $N = x \oplus y$
(d) $M = \bar{x}y$, $N = \overline{x \oplus y}$.

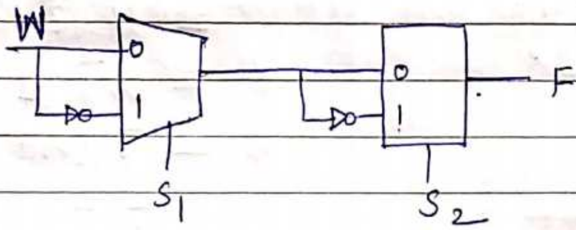
⇒

| x | y | D | B |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | (1) | (1) |
| 1 | 0 | (1) | 0 |
| 1 | 1 | 0 | 0 |

$B = \bar{x}y$

$D = \bar{x}y + x\bar{y} = x \oplus y$.

Q-⑥ Consider the MUX based ckt shown in fig -




Whic of the following Boolean function is realized by the ckt is ?

(a) $F = W\bar{S_1}\bar{S_2}$
(b) $F = WS_1 + WS_2 + S_1S_2$
(c) $F = \bar{W} + S_1 + S_2$
✓(d) $F = W \oplus S_1 \oplus S_2$

⇒

| $S_1$ | $S_2$ | F |
|---|---|---|
| 0 | 0 | $W\bar{S_1}\bar{S_2}$ |
| 0 | 1 | $\bar{W}\bar{S_1}S_2$ |
| 1 | 0 | $\bar{W}S_1\bar{S_2}$ |
| 1 | 1 | $WS_1S_2$ |

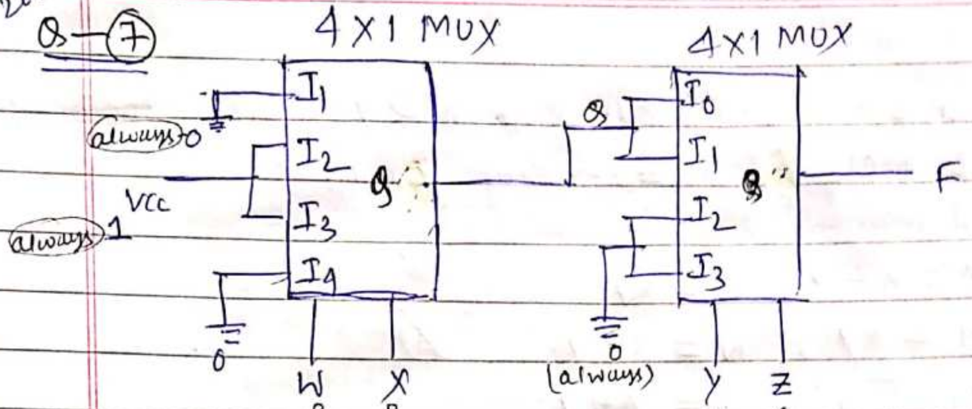
$F = W\bar{S_1}\bar{S_2} + \bar{W}\bar{S_1}S_2 + \bar{W}S_1\bar{S_2} + WS_1S_2$

$= W(\bar{S_1}\bar{S_2} + S_1S_2) + \bar{W}(\bar{S_1}S_2 + S_1\bar{S_2})$

$= W\,\overline{(S_1 \oplus S_2)} + \bar{W}\,(S_1 \oplus S_2)$

$= W \oplus S_1 \oplus S_2$.

2014
**Q—(7)**




The O/p is given by —

(a) $F = W\bar{X} + \bar{W}X + \bar{Y}Z$

(b) $F = W\bar{X} + \bar{W}X + \bar{Y}Z$

✓ (c) $F = W\bar{X}\bar{Y} + \bar{W}X\bar{Y}$

(d) $F = (\bar{W} + \bar{X})\,\bar{Y}Z$.

$\Rightarrow$ $Q = \bar{W}X\cdot 1 + W\bar{X}\cdot 1$

$= \bar{W}X + W\bar{X}$

$= W \oplus X$

$F = \bar{Y}\bar{Z}Q + \bar{Y}ZQ$

$= Q\bar{Y}(Z + \bar{Z})$

$= \bar{Y}(W \oplus X)$

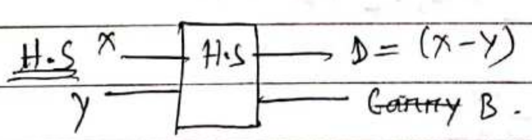
$= \bar{Y}[W\bar{X} + \bar{W}X]$

$F = W\bar{X}\bar{Y} + \bar{W}X\bar{Y}$

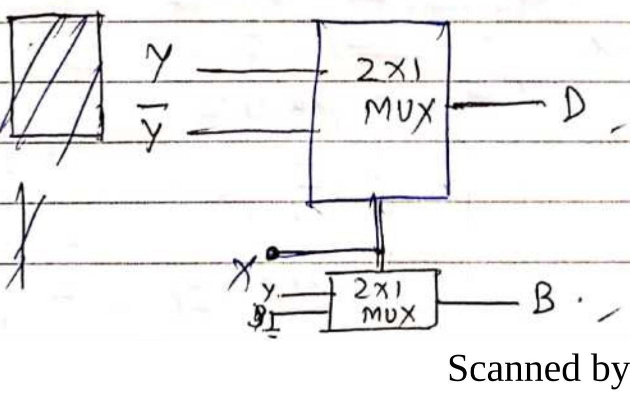
2014
**Q—8**

if x and y are i/ps and Difference (D) & the Borrow (B) are the O/ps. Implement it using 2×1 MUX

$\Rightarrow$ **H.S**

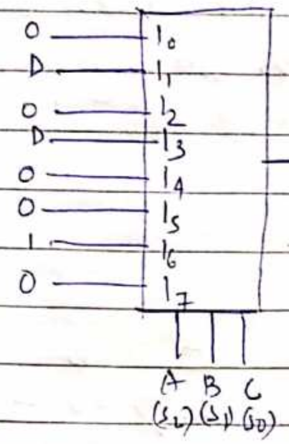



| x | y | D | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 (y) | 1 (y) |
| 1 | 0 | 1 ($\bar{y}$) | 0 |
| 1 | 1 | 0 | 0 |



2014

**Q-9** An 8-to-1 MUX is used to implement a logic y as shown in fig. The O/P.

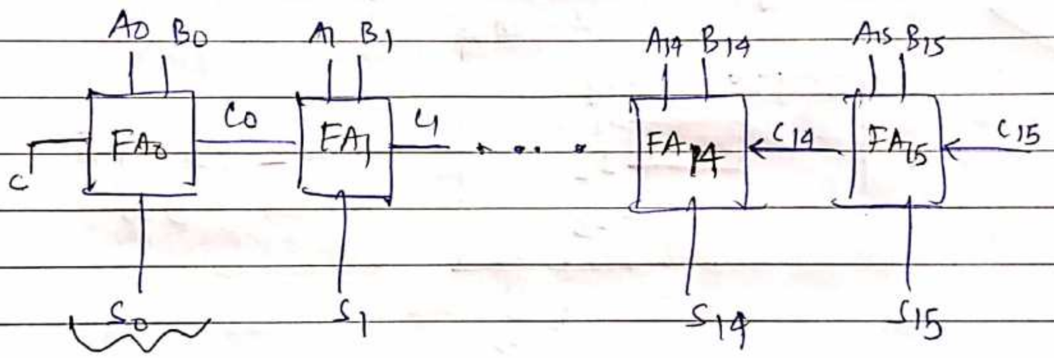



$y = \bar{A}\bar{B}CD + \bar{A}BCD + AB\bar{C}$

$= \boxed{\bar{A}DC + AB\bar{C}}$ ✓

**Q-10**

16 bit RCA is relized using 16 identical Full adder as shown in fig. The carry propagation delay of each FA is 12 ns and the sum propagation delay is 15 ns. The worst case delay (in ns) the 16 bit adder will be –




⇒ carry propagation delay = 12 ns
sum " " = 15 ns.

Total probagation delay = 15 ns + 15 × 12 ns
= 15 ns + 180 ns
= 195 ns ✓

Q-12
**Q-(11)**

The o/p 'Y' of a 2-bit comparator is logic '1' whenever the 2-bit i/p A is greater than 2-bit i/p B. The number of combinations for which the o/p is logic 1 is –

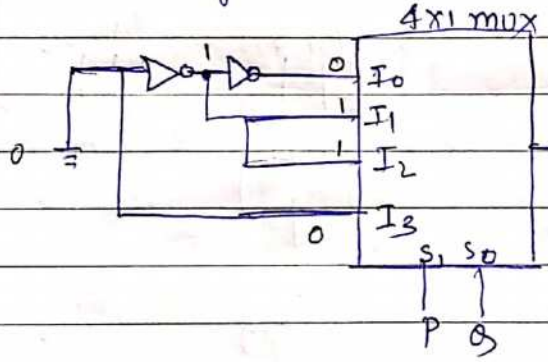
(a) 4 (b) 6 ✓
(c) 8 (d) 10.

⇒

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

| A | B | |
|---|---|---|
| 01 > | 00 – 1 | |
| 10 > | 00 – 1 | |
| | 01 – 1 | 6 |
| 11 > | 00 – 1 | |
| | 01 – 1 | |
| | 10 – 1 | |

Q-11
**Q-12** The logic function implemented by the ckt below is (ground implies a logic '0')

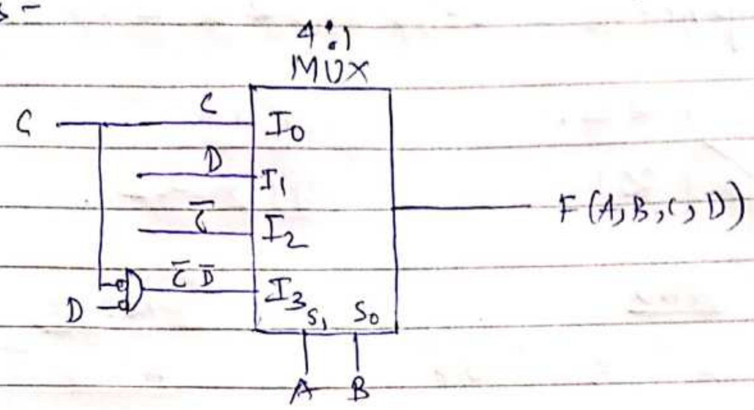



$F = \bar{P}Q + P\bar{Q}$
$= P \oplus Q$

(a) F = AND(P, Q) (b) F = OR(P, Q)
(c) F = XNOR(P, Q) (d) F = XOR(P, Q) ✓

Q-10

**Q-(13)** The Boolean function realized by the logic ckt shown is -




(a) $F = \sum m(0,1,3,5,9,10,14)$

(b) $F = \sum m(2,3,5,7,8,12,13)$

(c) $F = \sum m(1,2,4,5,11,14,15)$

(d) ✓ $F = \sum m(2,3,5,7,8,9,12)$.

⟹

| | A | B | C | D | F | | |
|---|---|---|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 | 0 (0) | | C |
| | 0 | 0 | 0 | 1 | 0 (1) | | |
| | 0 | 0 | 1 | 0 | 1 (2) ✓ | | |
| | 0 | 0 | 1 | 1 | 1 (3) ✓ | | |
| 01 | 0 | 1 | 0 | 0 | 0 (4) | | D |
| | 0 | 1 | 0 | 1 | 1 (5) ✓ | | |
| | 0 | 1 | 1 | 0 | 0 (6) | | |
| | 0 | 1 | 1 | 1 | 1 (7) ✓ | | |
| 10 | 1 | 0 | 0 | 0 | 1 (8) ✓ | | $\bar{C}$ |
| | 1 | 0 | 0 | 1 | 1 (9) ✓ | | |
| | 1 | 0 | 1 | 0 | 0 (10) | | |
| | 1 | 0 | 1 | 1 | 0 (11) | | |
| 11 | 1 | 1 | 0 | 0 | 1 (12) ✓ | | $\bar{C}\bar{D}$. |
| | 1 | 1 | 0 | 1 | 0 (13) | | |
| | 1 | 1 | 1 | 0 | 0 (14) | | |
| | 1 | 1 | 1 | 1 | 0 (15) | | |

cr-2

**Q-(14)** What are the min number of 2-to-1 MUX required to generate a 2-i/p AND & a 2-i/p EX-OR gate?

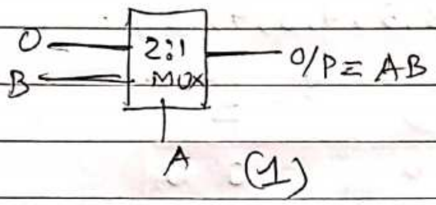
(a) 1 & 2.  (c) 1 & 3.
✓(b) 1 & 1.  (d) 2 & 2.

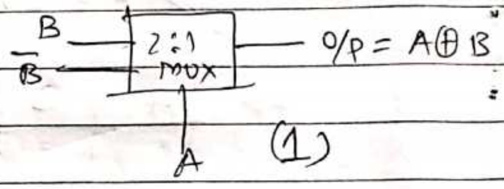
⟹ **AND**

| A | B | O/P |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**EX-OR**

| A | B | O/P |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




cr-3

**Q-(15)** without any additional circuitry, an 8:1 MUX can be used to obtain -

(a) Some but not all Boolean function of 3-variables.
(b) all functions of 3-variables but none of 4-variables
✓(c) " " " " and some but not all of 4-variables.
(d) all functions of 4 variables.