Number systems :

Any number can be expressed with help of its radix / base.

'4' Number systems Categorized :

1. Decimal Number System / radix = 10
2. Binary Number System / radix = 2
3. Octal Number System / radix = 8
4. Hexadecimal Number System / radix = 16

0 to 9 → Decimal Number System
0 & 1 → Binary Number System
0 to 7 → Octal Number System

(0 to 9) & (10 to 15)
Decimal A to F
10-A, 11-B, 12-C, 13-D, 14-E, 15-F } Hexadecimal Number System.

In general a number expressed in a base-r system has coefficients multiplied by powers of 'r'

$$a_n r^n \cdots + a_3 r^3 + a_2 r^2 + a_1 r^1 + a_0 + a_{-1} r^{-1} + a_{-2} r^{-2} + \cdots + a_{-m} r^{-m}$$

Ex : $(749)_{16}$, $(243)_8$, $(3A6)_{16}$, $(1011)_2$

Decimal	octal	hexadecimal
0	000	0000
1	001	0001
2	010	0010
3	011	0011
4	100	0100
5	101	0101
6	110	0110
7	111	0111
8		1000
9		1001
10		1010 (A)
11		1011 (B)
12		1100 (C)
13		1101 (D)
14		1110 (E)
15		1111 (F)

Octal - requires '3' bit in Binary Number System

$2^3 = 8$

Hexadecimal - requires '4' bits in Binary Number System

$2^4 = 16$

Conversions :

1. Binary to octal
2. Octal to Binary
3. Binary to Hexadecimal
4. Hexadecimal to Binary
5. Octal to hexadecimal
6. Hexadecimal to octal

7. Conversion of any radix to decimal

8. Conversion of decimal to any radix
   i) Successive division for Integer part
   ii) Successive Multiplication for fractional part.

1. Binary Number System to Octal Number System:

① $(10110001)_2 \rightarrow (131)_8$

MSB LSB

001 011 001

1 3 1

② $(111000111)_2 \rightarrow (707)_8$

7 0 7

2. Octal Number system to Binary Number System:

① $(745)_8 \rightarrow (\quad)_2$

$(111100101)_2$

② $(523.765)_8 \rightarrow (\quad)_2$

$(101010011.111110101)_2$

3. Binary Number System to Heradecimal Number System:

① $(0001\ 0110\ 1011)_2 \rightarrow (16B)_{16}$

1 6 B

② $(0011110101010.1110)_2 \rightarrow (1EAE)_{16}$

1 E A . E

4. Hexadecimal Number System to Binary Number System :

① $(A\ 9\ 5\ B)_{16} \rightarrow (\quad)_2$

A 9 5 B

1010 1001 0101 1011

$(1010100101011011)_2$

5. Octal Number System to Hexadecimal Number System:

① $(756)_8 \rightarrow (\quad)_{16}$

Step 1 : $(756)_8 \rightarrow (111101110)_2$

Step 2 : $(0\,0\,1\,1\,1\,1\,0\,1\,1\,1\,0)_2 \rightarrow (1EE)_{16}$

6. Hexadecimal number system to Octal Number System:

① $(C592)_{16} \rightarrow (\quad)_8$

Step 1 : $(C592)_{16}$

$(1100\,0101\,1001\,0010)_2$

Step 2 : $(142622)_8$

Weighted code :

Example

↳ Binary Code

18421 2421 , 5211

5311

$6 = 0110$

$= 8\times0 + 1\times4 + 1\times2 + 0\times1 = 6$

$8 = 1110 / 1101$

$9 = 1110 / 1101$

2 | 9

2 | 4 — 1

2 | 2 — 0

1 0

$9 = 1001$

5311 → 9 → 1110

$5\times1 + 3\times1 + 1\times1 + 1\times10$

$5+3+1+0 = 9$

4 → 0110 / 0101

Ex : 7 (what is Excess - 3 code for 7)

7 → 0111 +

(+3) 0011

1111

Binary address 1010

Sum=0 / Carry=1

1/1

1010

$2^2 + 0 2^2 +$

$8 + 2 = 10$

* Here adding 1 & 1, write sum also & Carry 1 to next significanG bit

Since it does not having any weighting factor so it Considered as unweighted.

ii) Gray Code :

i) Binary to Gray code conversion

→ Convert $(1100110)_2$ to Gray Code.

* Code
Binary Code ⟷ Gray code & XOR gate / Exclusive OR function

A	B	X OR O/P
0	0	0
0	1	1
1	0	1
1	1	0

* If we have n variable then we have 2^n Combination 2 in truth table

* $A \oplus B$ $AB' + A'B$

* For 3 variables, 8 combinations.

000	100
001	101
010	110
011	111

* Convert $(1100110)_2$ to Gray code

MSB ↓
1 1 0 0 1 1 0
performed ↓ XOR
1 0 1 0 1 0 1 ← This is the required Gray code

From start doing from MSB, & with corresponding succesing bildigit, perform XOR operation & copy that result, repeat until reach LSB.

(ii) Gray code to binary code:

MSB ↓ 1 1 1 1 1 0 1 ← Gray code.
1 0 1 0 1 1 0

⇒ Here also perform XOR operation in such that, first copy in as it as, new perform XOR operation to copied no & next bit of MSB, in this way we repeat until reach LSB.

1010110 → binary code
This is the equivalent binary code.

Ex:- 1 0 1 0 1 0 1 ← Gray
1 1 0 0 1 1 0

This is the equivalent binary code

* Self - Complement code
* Reflected codes
⇒ n-bit gray code is a member of a class called reflected codes

Rule for obtaining reflected Code :

⇒ n-bit gray code can be generated from (n-1) bit code

Suppose, we want 3-bit reflected code.

As per above slot, its can be obtained from 2-bit codes.

* Above axis MSB → 0

* below axis MSB → 1

1 bit → 2 bit → 3 bit → 4 bit.

0
1

MSB 00
01
MSB 11
10

000
001
011
010
110
111
101
100

0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000

* Alpha Numeric codes :

Ex : ASCII Code ⇒ American standard code for Implement Interchange.

Ex : EBCDIC Code

Extended Binary Code Decimal Interchange Code.

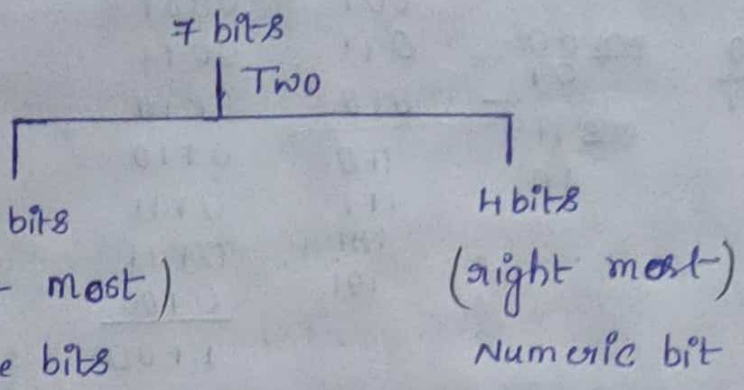
ASCII Code :

* This code used for representing characters, i.e., letters, numbers, special symbols, Mathematical symbols, Punctuation marks, Control Characters.

→ 7 bit code, i.e, that means 2^7 (128) different character.
(We can represent)

→ We are using ASCII code in our laptops

→ The ASCII 7 bit divides into 2 portions.
i.e., left portion having 3 bits. Here we called (zone bits)
right portion having 4 bits. Here we called (Numeric bits)

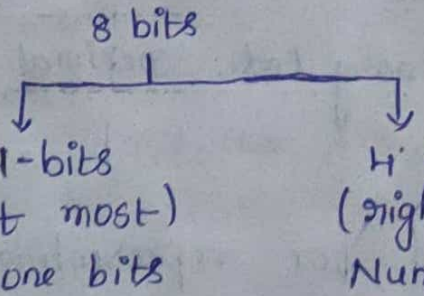


→ In ASCII, we have 8 bit also, (but we are using mostly 7-bit)

EBCDIC :

→ It is used to represent large Computer system.
Ex: mainframes.

→ using 8-bits representation.
8 bits → 256 characters.



Ex: Encode the word "Binary" in ASCII form using 7 bits.

B I N A R Y

42 49 4E 41 52 59

hexa decimal NO

3bit 4bit

100 0010 4I9 4IE 4I1 52

1000010 1001001 1001110 1000001 1010010

1000010

59

1011001

$2|14$, $2|2-0$, $1-0$

$2|2$, $1-0$

E → 14

→ Complements of numbers.

Binary, Decimal, octal, hexadecimal.

→ used to simplify substration operation.

Binary substraction:

A	B	Difference	Borrow
0	0=0	0	0
0	1=1	1	1
1	0	1	0
1	1	0	0

01 01101

0 1 1 0

(-) ———————

0 0 1 1

A	B	C	Difference	Borrow
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

Ex: 0 0 0 1
1 1 1 0
0 0 1 1

1 0 0 1
0 1 1 1
0 1 1 0

0 1 0 0 0 0
1 1 1 1 1 1
0 1 0 0 1 1

Ex: 1 0 0 1
0 1 1 0
0 0 1 1

0 0 0 0 1
1 1 1 1
0 1 0 0 1 0

→ $(10^3-1) - 497$
$- (1000-1)$
$(999) - 497$
→ $(502)_{10}$

$(5297)_{10}$ ← 9's Complement
→ $(10^4-1) - 5297$
→ $(9999 - 5297)$
→ $(4702)_{10}$

* 10's Complement :-

add 1 to 9's Complement

$((r^n-1) - N + 1) \rightarrow r^n - N$

Ex : 10's Complement of $(5555)_{10}$

$10^4 - 5555$
$10000 - 5555$
→ 4445.

Substraction using Complements :

→ The substraction of two n-digit unsigned numbers in base r can be done as follows

M, N are two numbers.

* M-N → two cases.

(i) If $m \geq N \rightarrow m-N = r$'s Complement (or) $(r-1)$'s Complement of $N+m$

r's Complement — end carry is added to the result.

end carry is neglected

ii) If $m < N$

$m-N = r$'s Complement (or) $(r-1)$'s Complement of $N+m = x$.

→ Again r's Complement (or) $(r-1)$'s Complement of x with placing - sign infront is the final result.

Ex: for given $x = 1010100$ & $y = 1000011$ find (i) $x-y$ & (ii) $y-x$ using 1's Complement.

(i) $x-y$ = 1's Complement of $y-x$.

↓

0111100 + 1010100.

```
 0111100
 1010100  (+)
 ------
 0010000  1+
 ------
 0010001
 ------
```

16+1 = 17

$x = 1010100 \quad -84$

$y = 1000011 \quad -67$

Sub 0010001 — 17

ii) $y-x$ = 1's of Complement of $x+y$.

0 1010 11 + 1000011

```
 0101011
 1000011  (+)
 ------
 1101110  ← x
 ------

 0010001  ← 1's Complement
```

$(1001)_2 = (17)_{10}$

1001 → 9
0110 → 6
0011 - 3

ex: 1 0 0 0 0 1
0 1 1 1 1 0
0 0 0 0 1 1

21 dec

Complements numbers :

two types of Complements for each base are System.

① The Radix Complement (r's Complement
② diminished radix Complement (r-1)'s Complement

→ most probably used techniques r's Complement.
→ 2's Complement & 10's Complement.
→ under (r-1)'s Complement is Complement 9's Complement.

→ 1'S Complement :

By changing is to 0 & 0's to 1's is called is Complement in Binary numbers.

ex: What is 1st Complement of.

10110110
01001001 → is Complement.

using formula :- $(r^n-1)-N$ / $r^n-1)-N+1$

2's 9's 2's or 10's

33
30
3

r - radix
n = no. of bits / digits
N = Given number

10110110.

$(2^8-1)-N$
$(256-1)-N$
$(255)-N$
↓
Binary form.

$(255)_{10} - (11111111)_2$

11111111
10110110
01001001

2	255
2	127-1
2	63-1
2	31-1
2	15-1
2	7-1
2	3-1
2	1-1
	0-1

→ 2's Complement :

By adding '1' to the 1's Complement result

→ what is 2's Complement of 10110110 ?

```
01001001   ← 1's Complement
11111111   1+1
--------
01001010
--------
01001010  → 2's Complement.
```

→ 9's Complement :

* $(r^n - 1) - N$

→ What is 9's Complement of $(497)_{10}$.

① $x = 10101101$ & $y = 11100100$ find (i) $x-y$ (ii) $y-x$ using 1's Complement.

② $x = 1111101$ & $y = 0100010$ find (i) $x-y$ (ii) $y-x$ using 1's Complement.

→ 10's Complement Substraction.

Ex : find 1888 − 1620 using 10's Complement.

let $m = 1888$, $N = 1620$.

(i) $m - N$ = 10's Complement of $N+m$. 10's Complement of N.

→ $r^n - N$

→ $10^4 - 1680$

→ $10000 - 1620$

→ 8380

```
  8380
  1888 (+)
  ----
  0268
```

Carry (= 10) ↓ sum.

```
  1888
  1620 (−)
  ----
   268
```

find 1620 - 1888 ?

→ 10's Complement of 1888

$10^4 - 1888$

→ 10000 - 1888

8112

1620 (+)

x=9732

10's Complement → 10000 − 9732

N-m ⇒ (268)

Self Complement Code :

If a weighted code is Self Complementing total weight must be '9'

→ BCD 1 8421 x

(15)x

→ 2421

(9) weighted Code / Self Complement Code.

→ 9's Complement in Binary.

Ex : 9's Complement of 7

$(10^1 - 1) - N$

$(10 - 1) - 7$

$9 - 7$

$= 2$

in 2421 Code

7 = 0111 / 1101

2 = 1000 / 0010

↓

1's Complement.

* Signed Binary numbers :

+9, -9 using sign bit a Computer can understand

20 → unsigned Number.

+20 → Signed Number.

+20 → 010100 - Binary form.

↑

Sign bit (left most bit is a sign bit)

→ To represent a +ve number, we have only one representation.

* Sign - Magnitude representation.

+9 = 0 0001001 (using 8 bits)

Magnitude

→ To represent a negative number we have 3 ways.

(i) Signed - Magnitude.

(ii) Signed - 1's Complement.

(iii) Signed - 2's Complement.

* Signed Arithmetic :

1. Addition :

Addition of two signed numbers using 8 bits and Signed 2's Complement representation.

Ex :- 6 & 13

```
+6  - 0 0 0 0 0 1 1 0
+13 - 0 0 0 0 1 1 0 1
-----------------------
+19   0 0 0 1 0 0 1 1
```

```
-6 → 1 0 0 0 0 1 1 0
     1 1 1 1 1 0 0 1   +
                   1
-----------------------
     1 1 1 1 1 0 1 0  ← 6' in 2's Complement
```

(ii) -6 → 11111010

+13 → 00001101

(1) 00001111

→ End carry neglected

-13 → 10001101

11110010 ← 1's Complement.

1 +

11110011

(iii) +6 → 0 0 0 0 0 1 1 0
-13 → 1 1 1 1 0 0 1 1 +
1 1
-7 → 1 1 1 1 1 0 0 1

(iv) -6 → 1 1 1 1 1 0 1 0
-13 → 1 1 1 1 0 0 1 1 +
1 1 1
-7 → 1 1 1 0 1 0 0 1
End carry neglected.

2. Subtraction :

Subtraction of two signed binary numbers when negative number are in 2's Complement form is very simple.

→ Take 2's Complement of subtrahend (Including Sign bit) and add it to minuend (Including Sign bit)

-6-13 = -6+ (-13)
↑
Take 2's Complemd.

-6 → 11111010 → 1 1 1 1 1 0 1 0
-13 → 1 1 1 1 0 0 1 1 → 0 0 0 0 1 1 0 1 +
1 1 1 1
0 0 0 0 1 1 0 0 +
1
0 0 0 0 1 1 0 1
1 0 0 0 0 0 1 1 1
End carry neglected.

* unsigned Arithmetic → addition
→ Subtraction
→ Multiplication
→ division

Multiplication :

Ex : multiplicant → 10110.1 → $(22.5)_{10}$
multiplier → 01001.1 → $(9.5)_{10}$

```
   10110.1 × 01001.1
   -----------------
        101101
       101101
      000000
     000000
    101101
   000000
   -----------------
   0110101010111
```

0110101011.11

1101010 ✓ Integer part
11 ✓ fractional part

Binary Division :

① 101010 ÷ 110 → divisor
dividend

```
        0111
110 ) 101010
      110
      ----
      1010
  (-)  110
      ----
       1001
        110
  (-) ----
       0110
      -----
        110
  (-) ----
          0
```

* Do it by actual division

101 < 110
5   6

So, the coefficient is 0

8 + 2 = 10
10 > 6
1
9 > 6

quotient 110
= 0111 = 111

Remainder = 0.

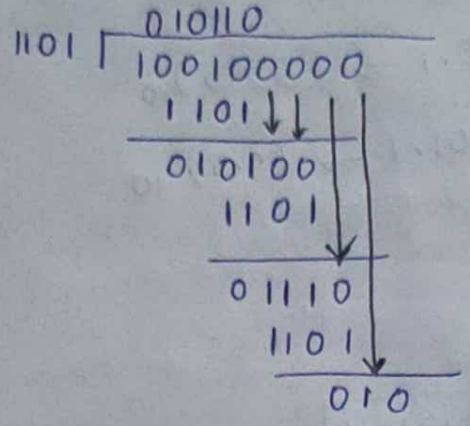
Ex : 1011010 ÷ 1010.

```
        1001
1010 ) 1011010
       1010
       -------
          1010
          1010
          ----
             0
```

1011 > 1010 → 1
10 < 1010
101 < 1010
1010 = 1010 → 1

quotient = 1001
remainder = 0.

* 100100000 ÷ 1101



quotient → 10110

remainder → 10

1101 > 1001

10010 1101

16 2

18 > 8 4 1

13

01010 < 1101

8

10

10100

4

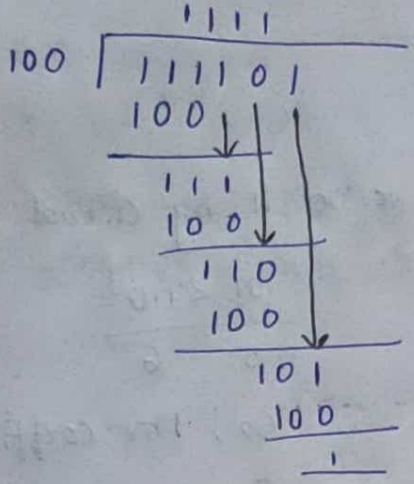
20 > 1101

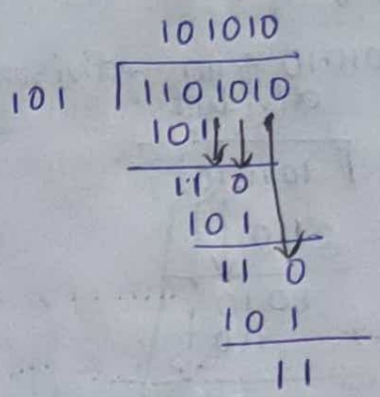
1110 > 1101

8 4 2

10 < 1101

* 11110 10101 ÷ 101





1. Given a = 10101001 & b = 1101

find (i) a+b (ii) a-b (iii) a.b (iv) a/b

2. find (3250 - 2532) using 102 Complement

①

Sol: a = 10101001

b = 1101

a+b ⇒ 10101001

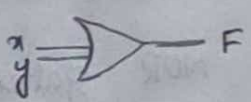
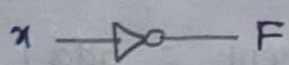
1101

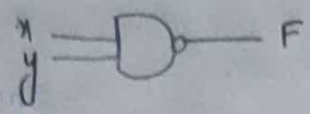
10110110

*Logic Gates :

(n basic logic gate have 2 input & 1 output)

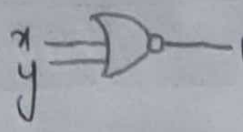
⇒ Allowing / retarding, particular signal / value.

Name	Graphic symbol	Algebric function	Truth table
1. AND		$F = x \cdot y$ (; dot product)	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
2. OR		$F = x + y$ (not binary addition) (logical addition) 1+1=1)	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
3. Inverter / NOT		$F = x' / \bar{x}$ (x Complement	x | F 0 | 1 1 | 0
4. Buffer		$F = x$ (It is opposite to inverter)	x | F 0 | 0 1 | 1

AND 

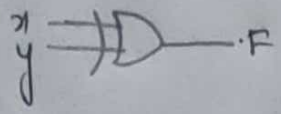
$F = (x \cdot y)'$
$= x \cdot y$
(Complement of dot product)

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

NOR 

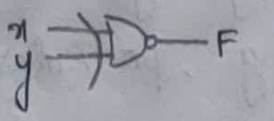
$F = (x+y)'$
(Complement of logical addition)

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

. Exclusive-OR (XOR) 

$F = xy' + x'y$
$= x \oplus y$
(Similar i/p → 0 (O/P)
different i/p → 1 (O/P)

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

. Exclusive-NOR XNOR 

$F = (x \oplus y')$
$= xy + x'y'$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

[For similar i/p → 1 (O/P)
For different i/p → 0 (O/P)]
Inversion of XOR.

There are 8 basic gates [Logic gates]

* NAND & NOR gates are "Universal gates"
* By using NAND & NOR gates, we can implement any logic gate function, that's why NAND & NOR gate are called universal gates.

10's Complement:

1) find its 9's Complement & add 1

Ex: 1423

1) 9's Complement

8576

10's Complement → 8576 + 1 = 8577

$x = 1010110$ $y = 11100100$ Using 1's Complement. (r-1)'s Complement)

$x - y$ is

Complement of $y + x$

$x = 10101101 - 173$

$2^7 2^5 2^3 2^2 2^0$

128 32 8 4 1

$y =$ 221

$x - y = -55$

→ 00011011

10101101

(+) 11111111

11001000

↓

→ It is not a +ve number

1's Complement

00110111

This is required Answer.

$-(110111)_2 = -(55)_{10}$

2^5

$y - x$

1's Complement of $x + y$

$228 - 173$

$= 55$

01010010

(+) 11100100

100110110

→ end carry +1

00110111

* NO Carry Means

→ It indicates -ve value.

$(110111)_2 = 55$.

Ex : $x = 1111101$ $y = 0100010$

Using 1's Complement.

$x - y$

= 1's Complement of $y + x$

$x = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 1$

$= 64 + 32 + 16 + 8 + 4 + 1$

$= 125$

⇒ 10111101

11111101

110111010

→ end carry +1

101010011

$y = 32 + 2 = 34$

$x - y = 125$

39 (-)

91

1011011 = 91

$y - x$

= 1's Complement of $x+y$

```
  0000010
  0100010
(+)    1
-------------
  0100100
-------------
NO end carry
```

$3x - 121$

$= -91$

So, 1's Complement of 0100100 = $-(1011011)$

$= -(91)$

29 Dec

Boolean Algebra :

It is defined with a set of elements, a set of operators & number of rules, laws, theorems & postulates

Postulates & theorems of Boolean Algebra :

① a) $x+0=x$ b) $x-1=x$

② a) $x+x'=1$ b) $x \cdot x'=0$

③ a) $x+x=x$ b) $x \cdot x=x$

④ $(x')' = x \longrightarrow$ Involution.

⑤ a) $x+y=y+x$ b) $x \cdot y = y \cdot x \rightarrow$ Commutative

⑥ a) $x+(y+z)=(x+y)+z$ b) $x(yz)=(xy.z) \rightarrow$ Associative.

⑦ a) $x.(y+z)=xy+xz$ b) $x+(y.z)=(x+y)(x+z) \rightarrow$ distributive

⑧ a) $(x+y)' = x' \cdot y'$ b) $(xy)' = x'+y' \rightarrow$ de-morgan's theorem

⑨ a) $x+xy=x$ b) $x.(x+y)=x \longrightarrow$ absorption.

Duality :

By interchanging AND & OR operators and replacing 1's by 0's & 0's by 1's

1. a) $x+x=x$?

$x+x=(x+x)\cdot 1$

$=(x+x)\cdot(x+x')$

$=x+(x\cdot x')$

$=x+0$

$=x.$

b) $x\cdot x=x$?

$x\cdot x=x\cdot x+0$

$=x\cdot x+x\cdot x'$

$=x\cdot(x+x')$

$=x\cdot 1$

$=x.$

2. a) $x+1=1$?

$x+1=(x+1)\cdot 1$

$=(x+1)(x+x')$

$=x+(x'+1)$

$=x+x'$

$=1$

b) $x\cdot 0=0.$

$x\cdot 0=x\cdot 0+0$

$=x\cdot 0+x\cdot x'$

$=x\cdot(0+x')$

$=x\cdot x'$

$=0.$

3. $x+xy=x$

$x+xy=x\cdot(x+y)$

$=x\cdot(1+y)$

$=x\cdot 1$

$=x.$

De-morgan's Theorem :

1. The Complement of a product of variables is equal to the Sum of the Complements of the variables

$(\overline{x\cdot y})=\overline{x}+\overline{y}$ / $(x\cdot y)'=x'+y'$

2. The Complement of Sum of variables is equal to the product of Complement of variables.

$\overline{x+y}=\overline{x}\cdot\overline{y}$ / $(x+y)'=x'-y'$

Truth table :

x	y	$x.y$	$\overline{x.y}$	$\overline{x}.\overline{y}$	$\overline{x}+\overline{y}$
0	0	0	1	1 1	1
0	1	0	1	1 0	1
1	0	0	1	0 1	1
1	1	1	0	0 0	0

Ex: Apply demorgan's theorem to the following & Simplify.

i) $\overline{(\overline{A+B}+\overline{C})}$

$$\overline{(\overline{A+B}+\overline{C})} = \overline{\overline{A+B}} \cdot \overline{\overline{C}} \qquad (\because (x')' = x)$$
$$= (\overline{\overline{A}} \cdot \overline{\overline{B}}) \cdot \overline{\overline{C}}$$
$$= (A \cdot B) \cdot C.$$

ii) $\overline{(\overline{A}+B)+CD}$

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

Find the Complement of following Expressions.

i) $F_1 = (A\overline{B}+C)\overline{D}+\overline{E}$

$$F_1 = (\overline{A\overline{B}}+C)\overline{D}+e$$
$$= \overline{(\overline{A\overline{B}}+C)\overline{D}+\epsilon}$$
$$= \overline{(A\overline{B}+C)\overline{D}} \cdot \overline{e}$$
$$= (\overline{A\overline{B}} \cdot \overline{C}+\overline{\overline{D}}) \cdot \overline{e}$$
$$= (\overline{A\overline{B}} \cdot \overline{C}+\overline{\overline{D}}) \cdot \overline{e}$$
$$= ((\overline{A}+\overline{\overline{B}}) \cdot \overline{C}+\overline{\overline{D}}) \cdot \overline{e}$$
$$= ((\overline{A}+B) \cdot \overline{C}+D) \cdot \overline{e}$$

ii) $F_2 = (x+\overline{y}+z)(\overline{x}+\overline{z})(x+y)$

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

iii) $F_1 = \bar{x}y\bar{z} + \bar{x}\bar{y}z$

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

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

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

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

iv) $x(\bar{y}\bar{z} + yz)$

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

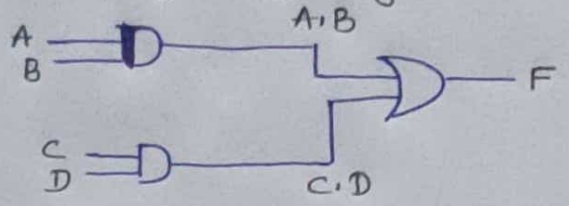
$$= \bar{x} + (\overline{\bar{y}\bar{z}})\cdot(\overline{yz})$$

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

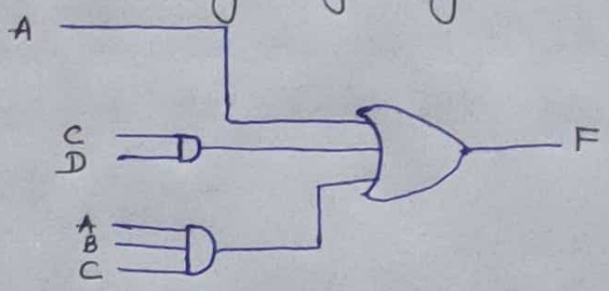
$$= \bar{x} + (y + z)\cdot(\bar{y} + \bar{z}).$$

Ex: $F = A\cdot B + C\cdot D$

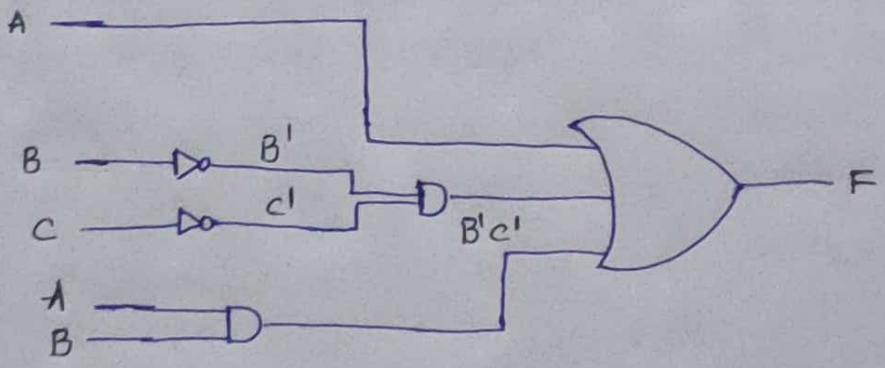
Draw using Logic gates.



$F = A + CD + ABC$ using Logic gates.



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



* Map Method:-

⇒ When we try to simplify Boolean functions in normal ways then it is a time consuming process and we have to re-write the simplified expressions after each step.

⇒ To overcome this difficulty, Karnaugh introduced a method for simplification of Boolean functions in an easy way. That is known as Map Method.

⇒ Map Method is also known as Karnaugh map or k-map (k represents karnaugh)

⇒ The k-map is a systematic way of simplifying Boolean Expressions

⇒ K-map is a pictorial method used to minimize Boolean Expressions without having to use Boolean algebra theorems and Equation manipulations.

⇒ k-map simplification is simples and less Error prone compared to the ~~sx~~ method of solving the logical Expressions using Boolean laws.

⇒ With the help of the k-map method, we can find the simplest POS and SOP Expression, which is known as the minimum Expression

⇒ The given Expression / function should be in canonical form which is either sum of Minterms or product of Maxterms.

⇒ Outcome of k-map is a standard form which means it may be SOP (sum of products form) or POS (product of sums form).

⇒ k-map method is used for Expressions containing 2, 3, 4 and 5 variables. For a higher number of variables, there is another method used for simplification called the Quine-McClusky method.

⇒ In k-map, the number of cells is similar to the total number of variable input combinations.

Ex:-

⇒ if number of variables is 3, the no. of cells is $2^3 = 8$

⇒ if number of variables is 4, the no. of cells is $2^4 = 16$ - - -

⇒ The k-map grid is filled using 0's and 1's.

⇒ The k-map is solved by making groups:-

→ Steps used to solve the Expressions using k-map are:-

① First, we find the k-map as per the number of variables.

② Find the maxterms and minterms in the given Expression.

③ Fill cells of k-map SOP with '1' respective to the minterms.

④ Fill cells of the block for POS with '0' respective to the maxterms.

⑤ Next, we create rectangular groups that contain total terms in the power of two like 2, 4, 8 --- and try to cover as many elements as we can in one group.

→ Grouping rules:- (Sum of Minterms)

① Group a two adjacent squares contains is that form a pair.

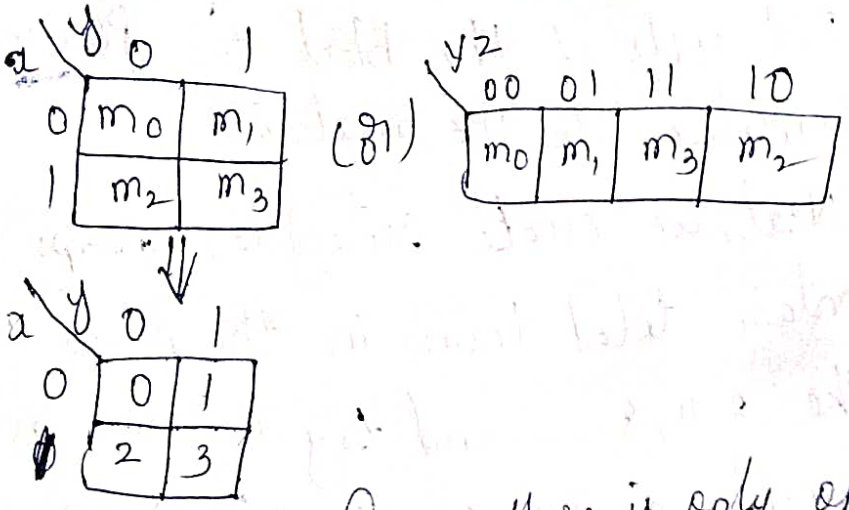
② Group a two adjacent pairs contains 1's that form a quad.

③ Group a two adjacent quads contains 1's that form an octet and so on - -

With the help of these groups, we find the product terms and sum them up for the sop form.

2 Variable k-map:- There is a total of 4 variables in a 2-variable k-map. There are two variables in the 2-variable k-map.

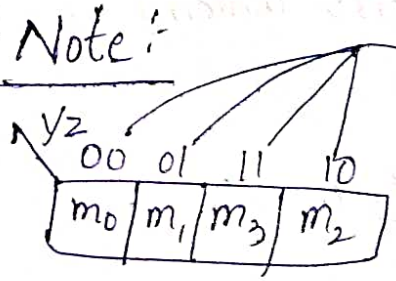
structure of the 2 variable k-map:-



⇒ In the above figure, there is only one possibility of grouping four adjacent minterms.

⇒ The possible combinations of grouping 2 adjacent minterms are {(m_0, m_1), (m_2, m_3), (m_0, m_2) and (m_1, m_3)}.

Note:-



here gray code is used. Based on gray code the k-map cells are numbered.

→ The gray code property is used in k-map to simplify Boolean functions. Due to this error probability is reduced. You can identify adjacent 1's & 0's and reduce it.

→ Because of gray code only changes one bit at a time as you move between adjacent states, so it makes the groupings of terms possible, because they are next to each other.

⇒ If you used binary code instead of gray code, the regions would be ~~disjointed~~ disjoined and the grouping of terms not so obvious.

⇒ That's why the cells in a k-map are numbered in gray code so that only one bit is changed and you can take variables of consecutive cells in common to cancel out the varying bit...!

Ex:- Minimize the Boolean Expression

$$F(x,y) = x'y + xy' + xy$$

(A) Given,

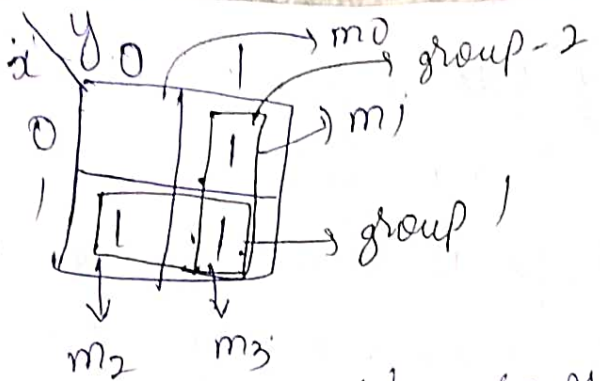
$$F(x,y) = x'y + xy' + xy$$

$x'y = 01 = 1$

$xy' = 10 = 2$

$xy = 11 = 3$

$\therefore F(x,y) = \Sigma(1,2,3)$



⇒ We know that, possible groups are

$(m_0, m_1), (m_2, m_3), (m_0, m_2), (m_1, m_3)$

Here, those groups are

$(0,1), (1,1), (0,1), (1,1)$

So, we won't consider $(0,1)$ here in this Example. So, we won't consider (m_0, m_1), (m_0, m_2). We consider, (m_2, m_3) and (m_1, m_3) whose values are $(1,1)$ and $(1,1)$.

∴ group-1 ⇒ $(m_2, m_3) = (1,1) = x$

group-2 ⇒ $(m_1, m_3) = (1,1) = y$

∴ $F(x,y) = x + y$

Verification:-

$F(x,y) = x'y + xy' + xy$

$= x'y + x(y + y')$ $(\because y + y' = 1)$

$= x'y + x$

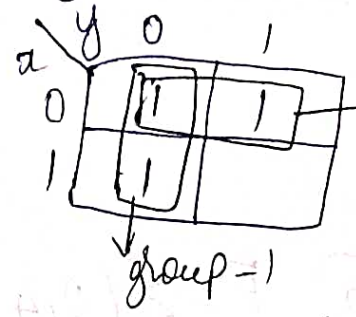
$= (x + x') \cdot (x + y)$ $(\because x + (yz) = (x+y)\cdot(x+z)$ distributive law

$= x + y$ ✓

Ex:

$F(x,y) = \Sigma(0,1,2)$

(A)



group-1 $\Rightarrow (1,1) = y'$

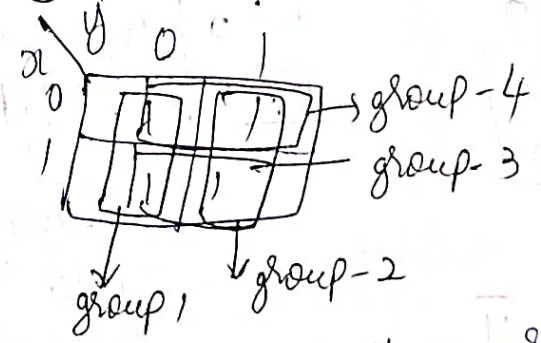
group-2 $\Rightarrow (1,1) = x'$

$F(x,y) = x' + y'$

Ex:

$F(x,y) = \Sigma(0,1,2,3)$

(A)



$F(x,y) = 1$. If there are also 1's then result will be 1.

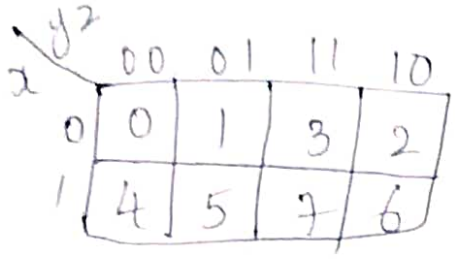
3 variable k-map:-

⇒ The 3-variables k-map is represented as an array of Eights cells.

i.e, $m_0, m_1, m_2, m_3, m_4, m_5, m_6$.

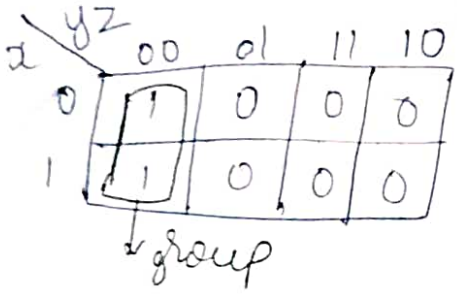
Structure:

x \ yz	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

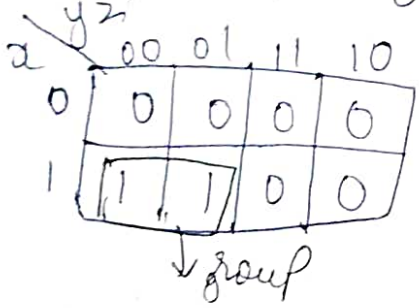


Few possible groups:-

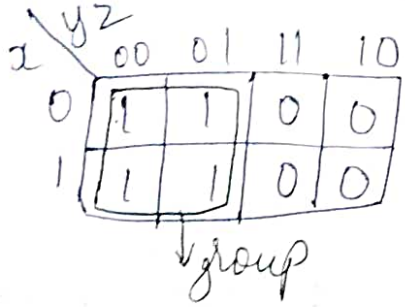
1) $F = \Sigma(0,4) = \bar{y}\bar{z}$



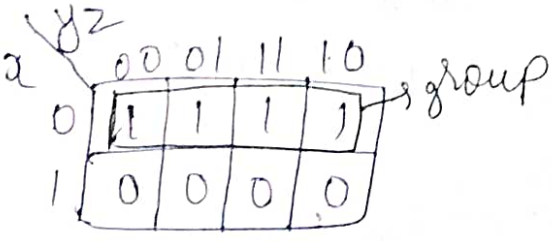
2) $F = \Sigma(4,5) = x\bar{y}$



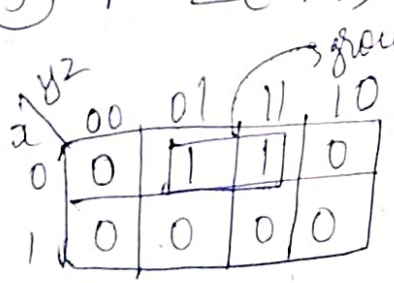
3) $F = \Sigma(0,1,4,5) = \bar{y}$



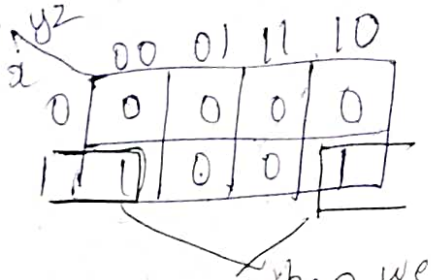
4) $F = \Sigma(0,1,2,3) = \bar{x}$



5) $F = \Sigma(0,4) = \bar{x}z$

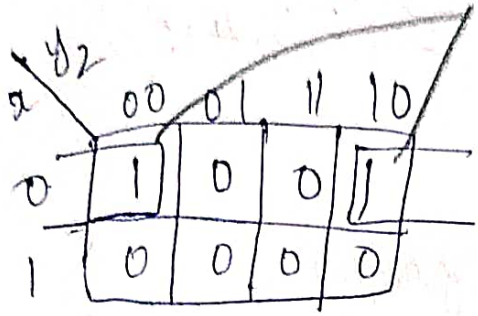


6) $F = \Sigma(4,6) = x\bar{z}$



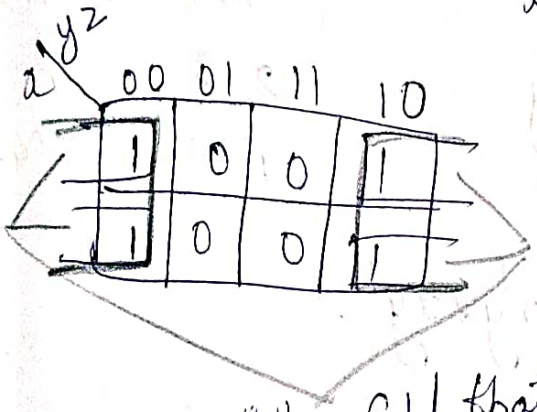
When we fold then the two will become adjacent and forms a group.

⑦ $f = \Sigma(0,2) = \bar{x}\bar{z}$



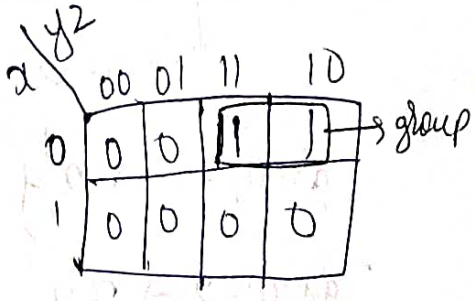
When it fold, they can be in one group.

⑧ $f = \Sigma(0,2,4,6) = \bar{z}$



When fold that four will form one group.

⑨ $f = \Sigma(3,2) = \bar{x}y$

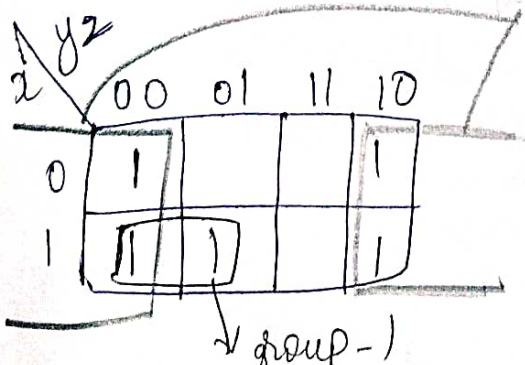


Ex:

Simplify $F(x,y,z) = \Sigma(0,2,4,5,6)$

(A) Given,

$F(x,y,z) = \Sigma(0,2,4,5,6)$



group-1 ⇒ $x.y'$

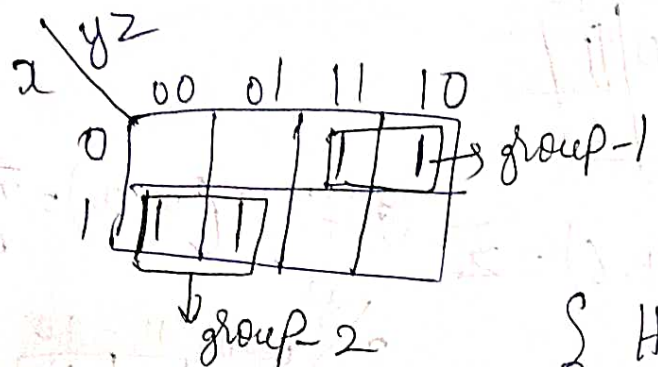
group-2 ⇒ $\bar{z}$,

$F(x,y,z) = x.y' + z'$

{Hint:- Refer ② & ⑧ in possible groups list}

Ex: $F(x,y,z) = \Sigma(2,3,4,5)$

(A) Given, $F(x,y,z) = \Sigma(2,3,4,5)$



{ Hint: Refer 2 & 4 in possible groups }

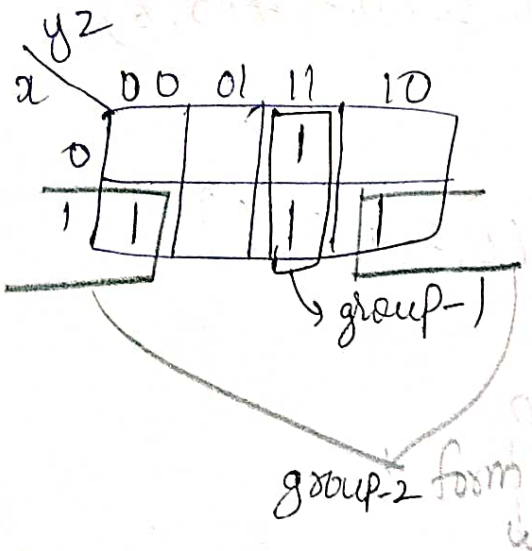
group-1 $\Rightarrow x'y$

group-2 $\Rightarrow xy'$

$\therefore F(x,y,z) = x'y + xy'$

Ex: ⑥ $F(x,y,z) = \Sigma(3,4,6,7)$

(A) $F(x,y,z) = \Sigma(3,4,6,7)$



form one group when we fold.

group-1 $\Rightarrow yz$

group-2 $\Rightarrow xz'$

$\therefore F(x,y,z) = yz + xz'$

{ Hint: refer diagram 6 in possible groups }

* Given the Boolean function

$$F = A'C + A'B + AB'C + BC$$

(a) Express it in sum of minterms

(b) Find the minimal of SOP

(A) a)

$$F = A'C + A'B + AB'C + BC$$

$$= A'C(B+B') + A'B(C+C') + AB'C + BC(A+A')$$

$$(\because (B+B') = 1,\ (A+A') = 1,\ (C+C') = 1)$$

$$= A'BC + A'B'C + A'BC + A'BC' + AB'C + ABC + A'BC$$

$$= A'BC + A'B'C + A'BC' + ABC + AB'C$$

$$\Rightarrow A'BC = 011 = 3$$

$$A'B'C = 001 = 1$$

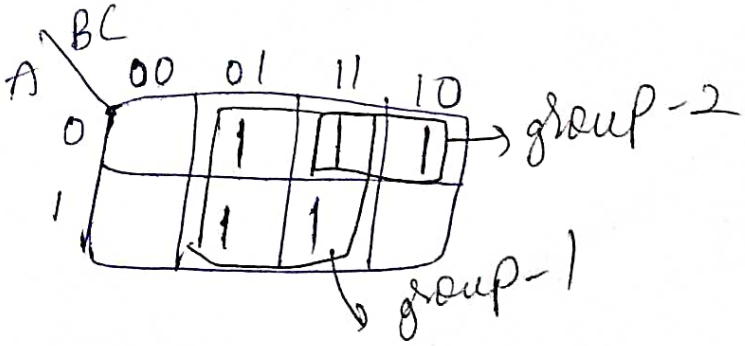
$$A'BC' = 010 = 2$$

$$AB'C = 101 = 5$$

$$ABC = 111 = 7$$

$$\Rightarrow \Sigma(1, 2, 3, 5, 7)$$

(b) $F(A, B, C) = \Sigma(1, 2, 3, 5, 7)$



group 1 ⇒ C

group 2 ⇒ $A'B$.

$\therefore F(A,B,C) = C + A'B$

group 1 ⇒ C

group 2 ⇒ $A'B$.

$\therefore F(A,B,C) = C + A'B$

4 variable k-map:-

⇒ The 4-variable k-map is represented as an array of 16 cells.

⇒ Binary values A and B are along the left side, and the values of C and D are across the top.

⇒ The binary value of the given cell is the binary values of A and B at left side in the same row combined with the binary values of C and D at the top in the same column.

structure:-

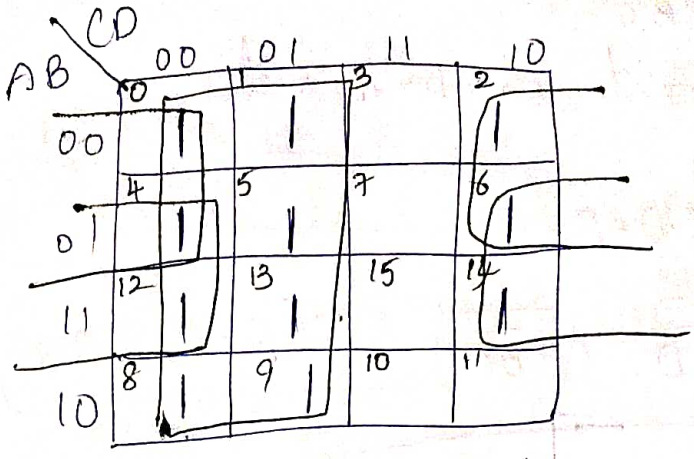
AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{14}	m_{15}
10	m_8	m_9	m_{11}	m_{10}

Ex:- Simplify the Boolean function,

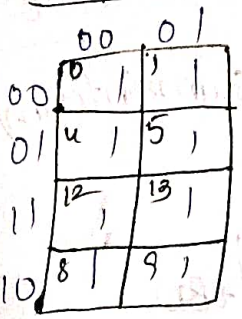
$F(A,B,C,D) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$

(A) Given,

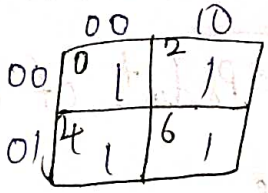
$F(A,B,C,D) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$



Group-1:-

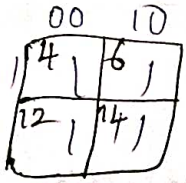


Group-2:-



$\Rightarrow A'D'$

Group-3



$\Rightarrow BD'$

$\Rightarrow C'$

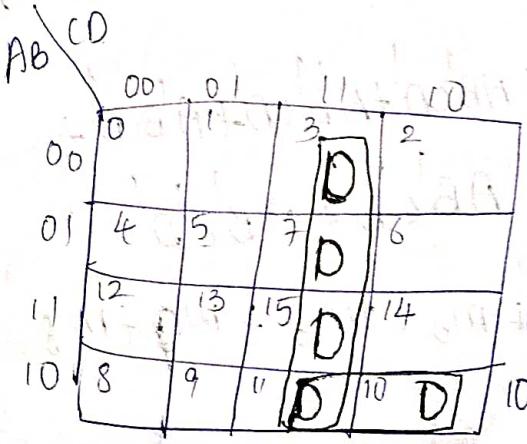
∴ Simplified Expression is,

$$F = C' + A'D' + BD'$$

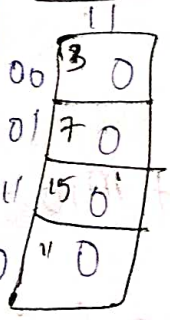
→ Sum of product terms (SOP)

In the above Example, the max terms are

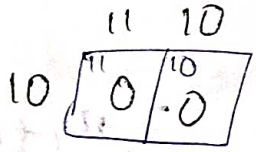
$$\Pi(3, 7, 10, 11, 15)$$



Group-1



Group-2:-



$\Rightarrow AB'C$.

$\Rightarrow CD$

$F' = CD + AB'C$

Do complement on both sides,

$(F')' = (CD + AB'C)'$

$F = (C\bar{D}) \cdot (A\bar{B}C)$ $\left[\because (A+B)' = A'B'\right]$

$\boxed{F = (\bar{C}+\bar{D}) \cdot (\bar{A}+B+\bar{C})}$ $\left[\because (AB)' = A'+B'\right]$

↳ Product of sum minterms

∴ For given question the

required sop is $F = C' + A'D' + BD'$.

required Pos is $F = (\bar{C}+\bar{D}) \cdot (\bar{A}+B+\bar{C})$

* Simplify the following Boolean Expression.

$F = A'B'C' + B'CD' + A'BCD' + AB'C'$

(A) Given,

$F = A'B'C' + B'CD' + A'BCD' + AB'C'$

$= A'B'C'(D+D') + B'CD'(A+A') + A'BCD' + AB'C'(D+D')$

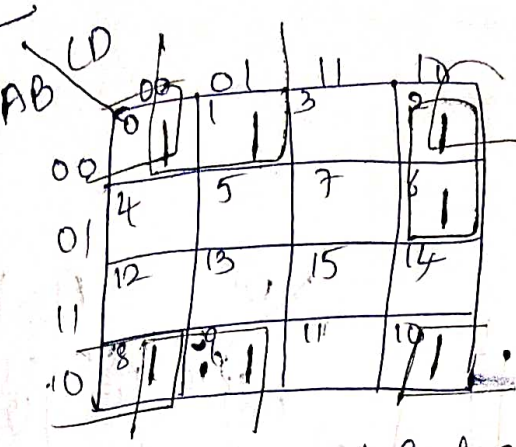
$= A'B'C'D + A'B'C'D' + AB'CD' + A'B'CD' + A'BCD' +$

$AB'C'D + AB'C'D'$

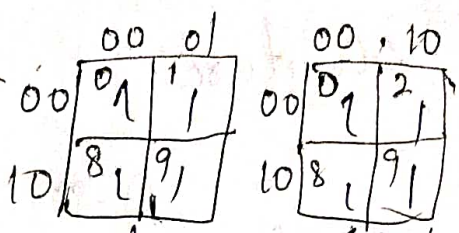
$= m_1 + m_0 + m_{10} + m_2 + m_6 + m_9 + m_8$

∴ $F = \Sigma(0, 1, 2, 6, 8, 9, 10)$

k-map:-

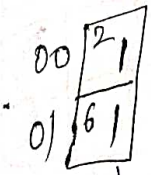


Group-1 Group-2



$\Rightarrow B'C'$ $\Rightarrow B'D'$

Group-3



$\Rightarrow A'CD'$

∴ The simplified Expression is

$$F(A,B,C,D) = B'C' + B'D + A'CD'$$

↳ SOP

5 variable k-map:- ~~This is~~

⇒ With the help of the 32-cell ($2^5 = 32$) k-map, the boolean Expression with 5 variables can be simplified.

⇒ For constructing 5 variable k-map, we use two 4-variable k-maps,

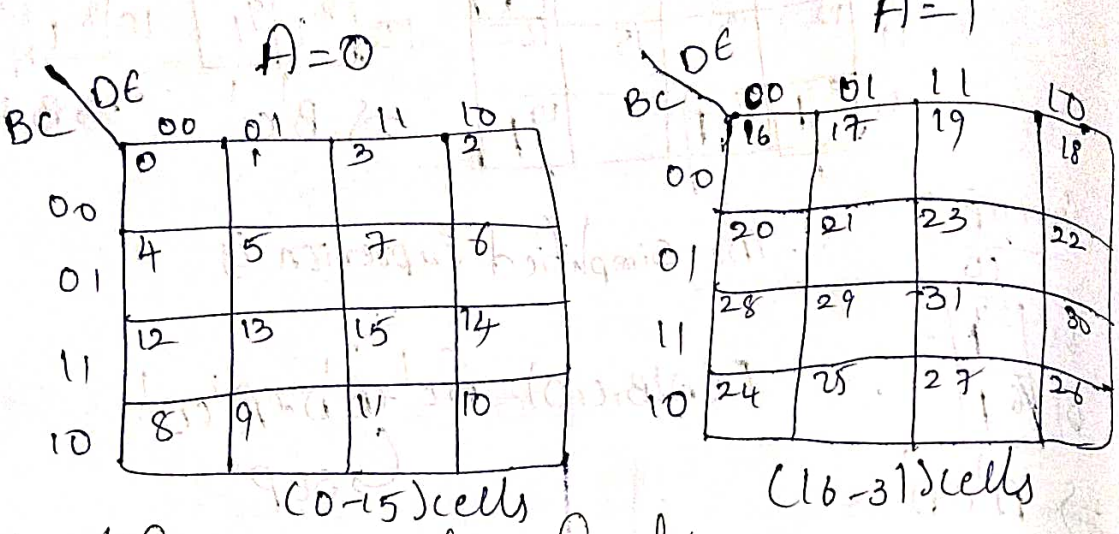
⇒ The cell adjacencies within each of the 4-variable maps for the 5-variable map are similar to the 4-variable map.

⇒ A k-map for five variables (ABCDE) can be constructed using two four variable maps.

⇒ Each map contains 16 cells with all combinations of variables B, C, D and E.

⇒ One map is for A=0, and the other is for A=1.

Structure:



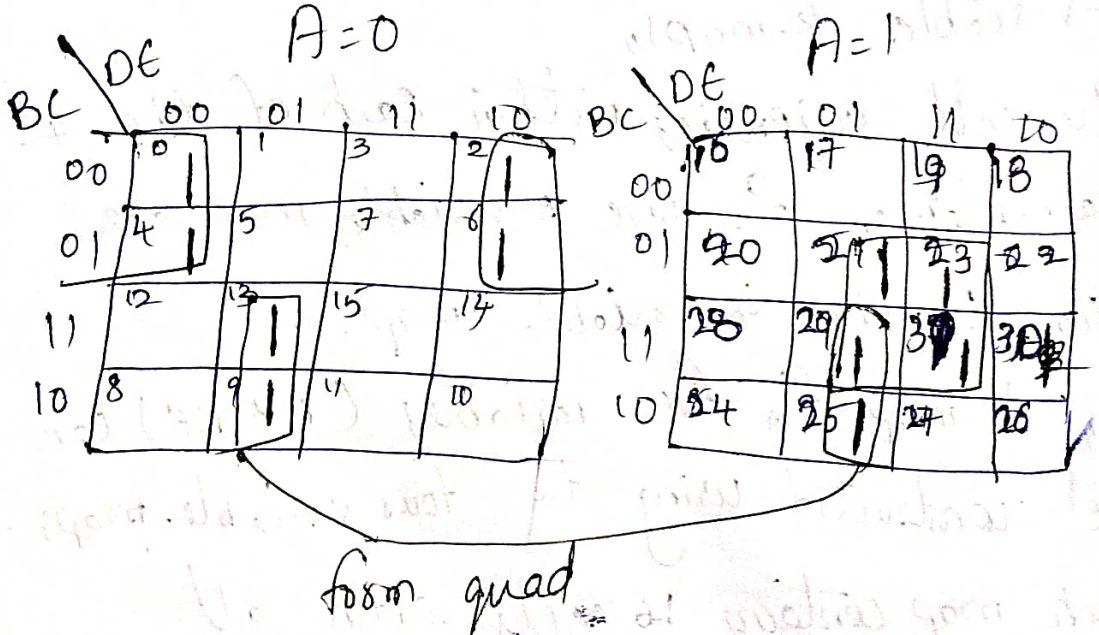
* Simplify the Boolean function

$F(A,B,C,D,E) = \Sigma(0,2,4,6,9,13,21,23,25,29,31)$

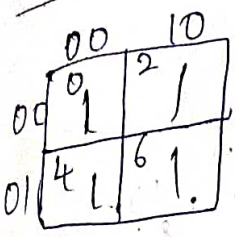
(A) Given,

$F(A,B,C,D,E) = \Sigma(0,2,4,6,9,13,21,23,25,29,31)$

K-map:

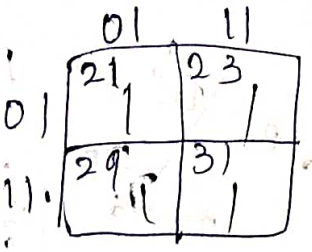


Group-1 :



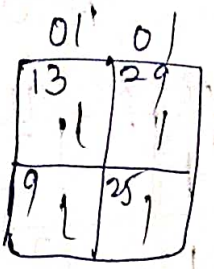
$\Rightarrow A'B'E'$

Group-2



$\Rightarrow ACE$

Group-3



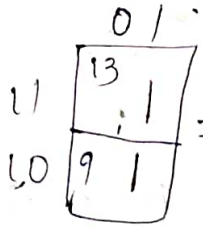
$\Rightarrow BD'E$

∴ the simplied Boolean Expression is,

$$F(A,B,C,D,E) = A'B'E' + ACE + BD'E$$

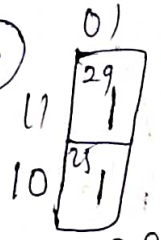
{ if we did normally, then group-3 can be written as two groups

A=1

① 

$\Rightarrow A'BD'E$

A=0

② 

$\Rightarrow ABD'E$

① + ② = $A'BD'E + ABD'E$

$= BD'E(A'+A)$

$= BD'E(1)$ ($\because A'+A=1$)

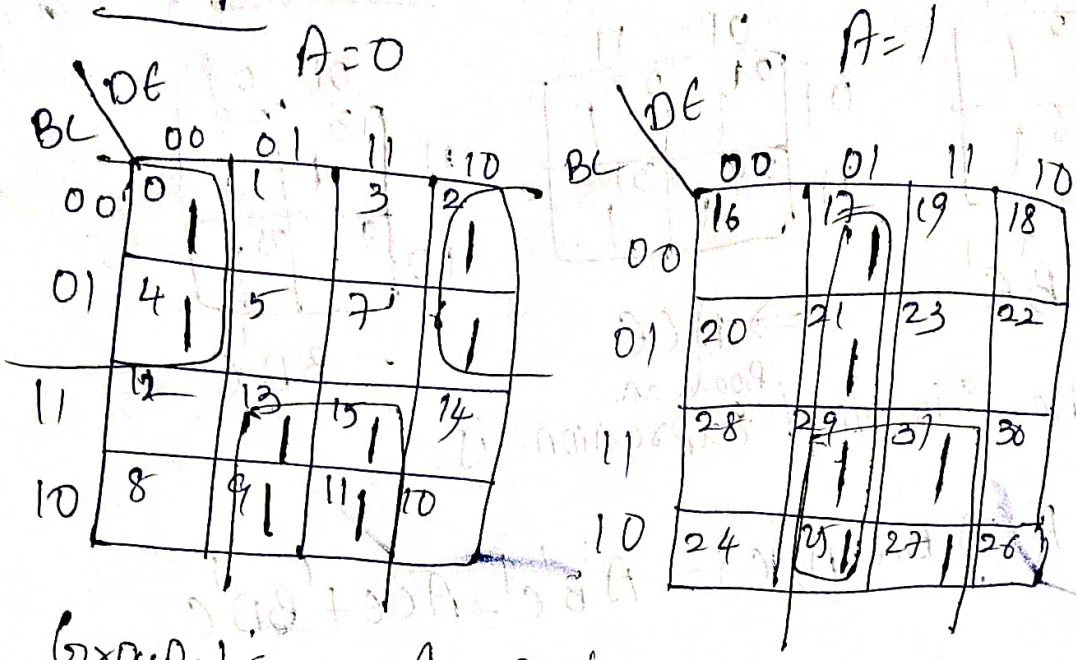
$= BD'E$ → same result we get in group 3 when we did directly)

*Simplify $F(A,B,C,D,E) = \Sigma(0,2,4,6,9,11,13,15,17,21,25,27,29,31)$.

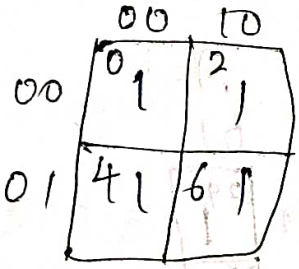
(A) Given,

$F(A,B,C,D,E) = \Sigma(0,2,4,6,9,11,13,15,17,21,25,27,29,31)$.

K-map:

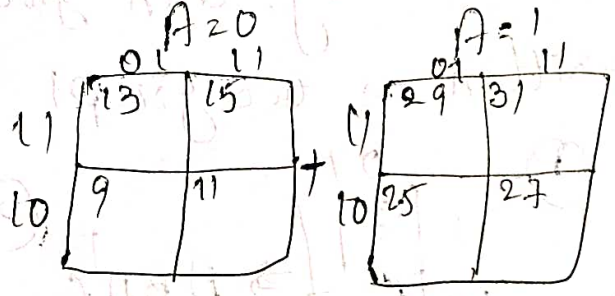


Group-1:



$\Rightarrow A'B'E'$

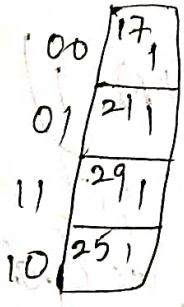
Group-2:



$\Rightarrow A'BE' + ABE \Rightarrow BE(A'+A)$

$\Rightarrow BE$

Group-3:



$\Rightarrow AD'E$

∴ The Simplified boolean Expression is,

$$F = A'B'E' + BE + AD'E$$

Don't Care Condition:

⇒ The "Don't care" condition says that we can use the blank cells of a k-map to make a group of the variables. We can simply say "unused inputs are don't cares".

⇒ To make a group of cells, we can use the "don't care" cells as either 0 or 1, and if required, we can also ignore that cell.

⇒ We mainly use the "don't care" cell to make a large group of cells.

⇒ The cross (X) symbol is used to represent the don't care" cell in k-map. We can use d or minus (-) or phi (ϕ) also to represent don't care cell.

⇒ The "don't care" in Excess-3 code are 0000 (0), 0001 (1), 0010 (2), 1101 (13), 1110 (14), and 1111 (15) because they are invalid combinations. Apart from this, the 4-bit ~~BD~~ BCD to Excess-3 code, the "don't care" are 1010 (10), 1011 (11), 1100 (12), 1101 (13), 1110 (14), 1111 (15).

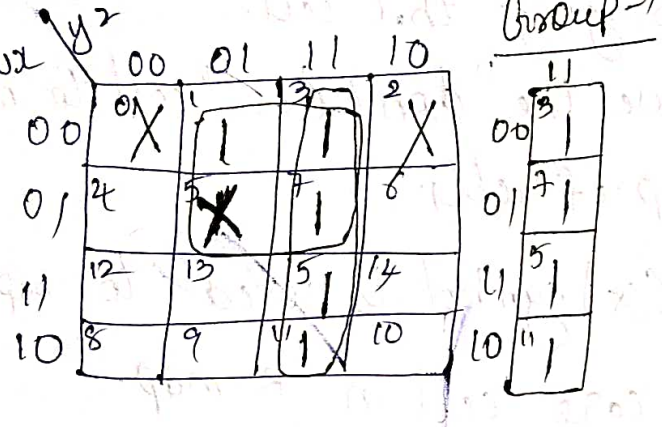
* Simplify the Boolean function $F(w,x,y,z) = \Sigma(1,3,7,11,15)$ which has the don't care conditions $d(w,x,y,z) = \Sigma(0,2,5)$

(A) Given,

$F(w,x,y,z) = \Sigma(1,3,7,11,15)$

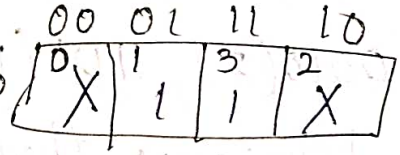
$d(w,x,y,z) = \Sigma(0,2,5)$

k-map:-

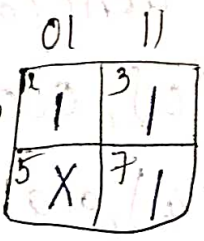


$\Rightarrow yz$

here we can make group 2 in 2 ways.

① 

$\Rightarrow w'x'$

(or) 

(or) $w'z$

$\therefore F = yz + w'x'$

(or)

$F = yz + w'z$

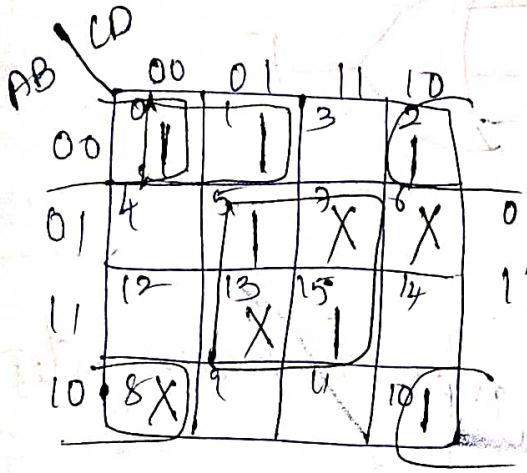
* $F(A,B,C,D) = \Sigma(0,1,2,5,10,15) + d(6,7,8,13)$

(A) Given,

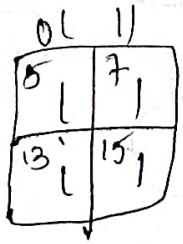
$F(A,B,C,D) = \Sigma(0,1,2,5,10,15)$

$d(A,B,C,D) = d(6,7,8,13)$

K-map:-

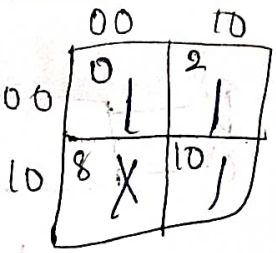


Group-1



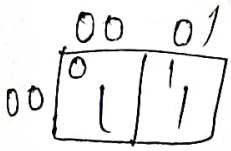
$\Rightarrow BD$

Group-2



$\Rightarrow B'D'$

Group-3:-



$\Rightarrow A'B'C'$

$\therefore$ The simplified boolean Expression is

$F = BD + B'D' + AB'C'$

Implementation

*

① Implementation using only NAND gates:-

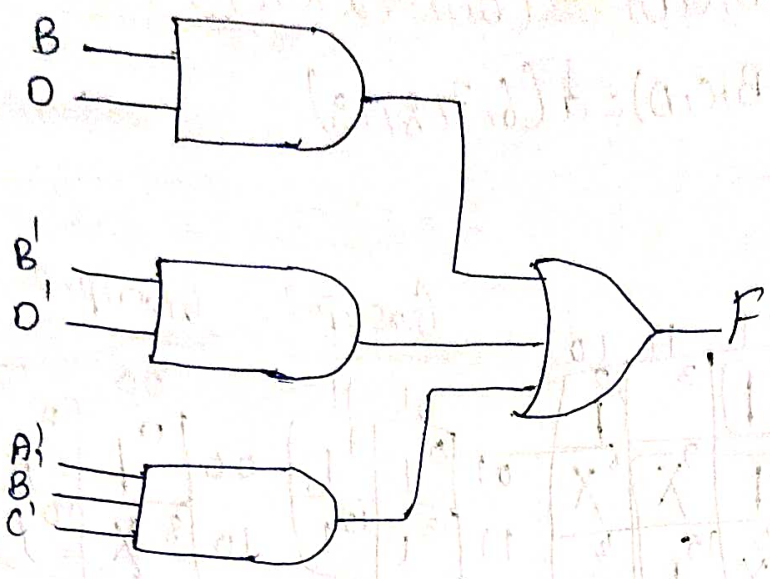
Ex:- Implement the boolean Expression

$F = BD + B'D' + AB'C'$ using only NAND gates.

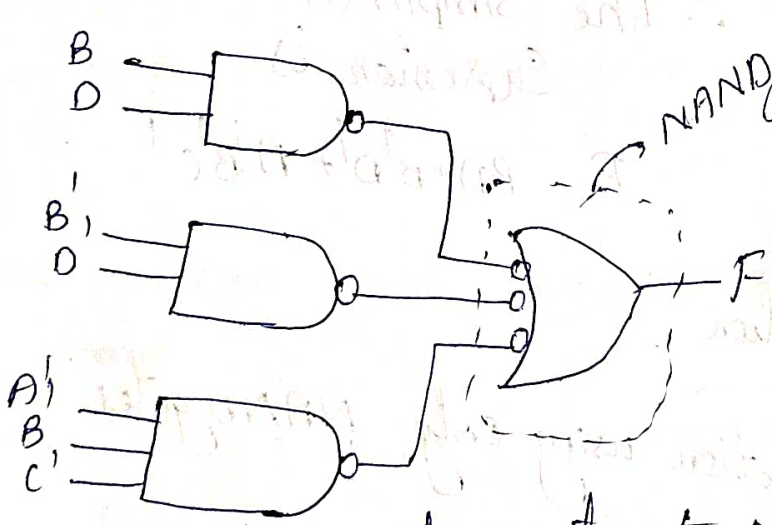
(A) Given,

$F = BD + B'D' + AB'C'$.

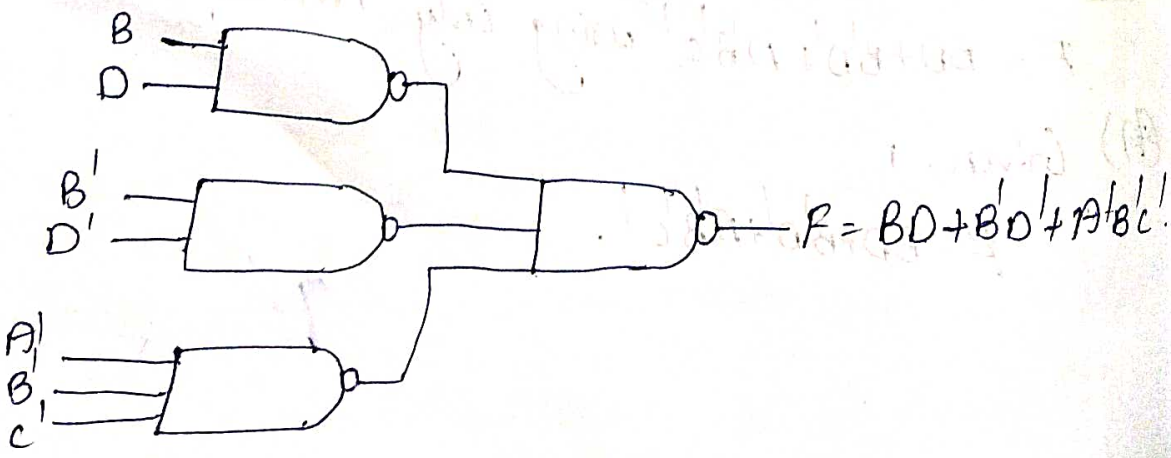
Step-1:- Draw general diagram



Step-2:- Convert all AND gates to NAND gates & OR gates to invert-OR gate



Step-3:- Convert invert-OR gates to NAND gates.



(2) Implementation using only NOR gates

(for POS)

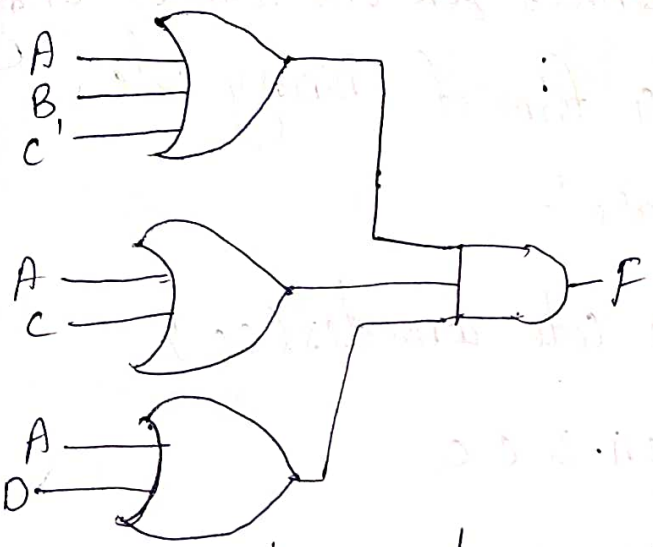
Ex:- Implement the Boolean Expression,

$F=(A+B+C').(A+C).(A+D)$ using NOR gates.

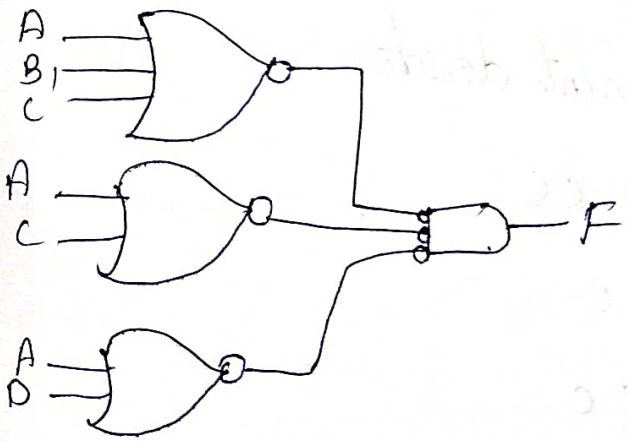
(A) Given,

$F=(A+B+C')(A+C)(A+D)$

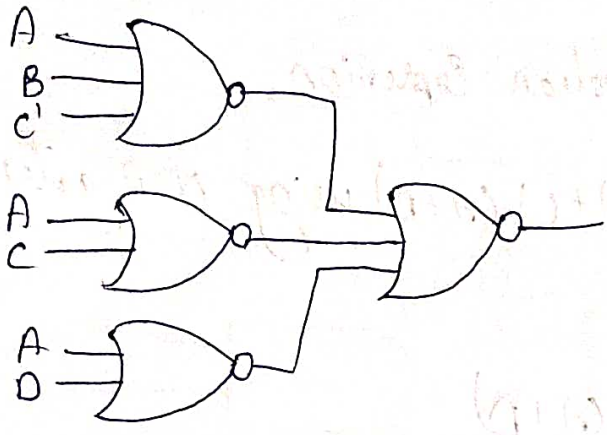
Step-1:- Draw general diagram



Step-2:- Convert OR gates to NOR gates & AND gates to invert AND gates.



Step-3:- Convert invert AND to NOR gates.



$F=(A+B+C')(A+C)(A+D)$

(NOR-NOR logic)

Code Converter (C.C):- It is a combinational circuit which converts one form of binary code to another form of binary code.

Code Converters are:-

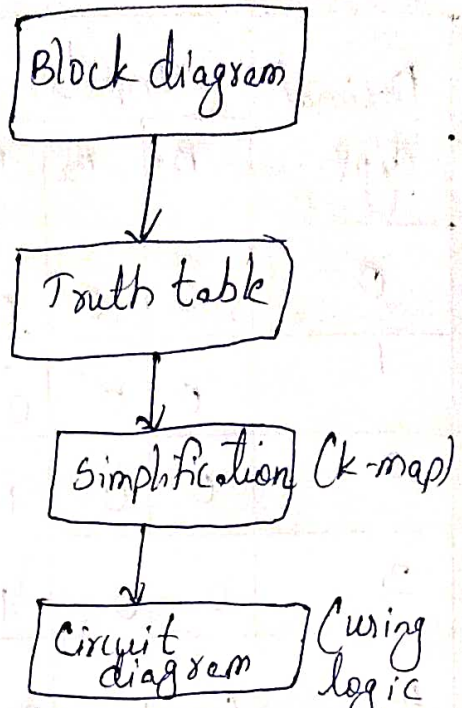
1. Binary to BCD code converter (cc)
2. Binary to Excess-3 cc
3. Binary to Gray cc
4. BCD to Gray cc
5. BCD to Excess-3 cc
6. BCD to seven-segment decoder
7. 8-4-2-1 to Excess-3 cc
8. 8-4-2-1 to BCD cc
9. 8-4-2-1 to Gray cc.

Process :-

step-1 : Draw block diagram for the given code converter.

step-2 : Draw truth table for that block diagram

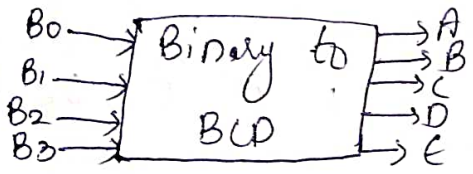
step-3 : Perform simplification using k-map.



step-4 : Draw circuit diagram using logic gates for that equation obtained from k-map.

① Binary to BCD CC :-

step-1 : Draw block diagram



Here number of inputs are 4 - B_0, B_1, B_2, B_3.

number of outputs are 5 - A, B, C, D, E

Step-2 :- Draw truth table

Decimal digit	Inputs				Outputs				
	B_3	B_2	B_1	B_0	A	B	C	D	G
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

Note:

We know that, for each digit in a given number we have to write BCD using 8-4-2-1. So, in the above table, for any write number

BCD using 8-4-2-1.

Ex:- $11 \Rightarrow$ 1 1

0001 0001

0001 0001 $\Rightarrow$ 1000 1 = 12

$12 \Rightarrow$ 1 2

0001 0010

0001 0010 $\Rightarrow$ 10010 = 12

By doing like this, we write outputs in the above table.

Step-3:- Simplification using k-map

$A = \Sigma(10, 11, 12, 13, 14, 15)$

$B = \Sigma(8, 9)$

$C = \Sigma(4, 5, 6, 7, 14, 15)$

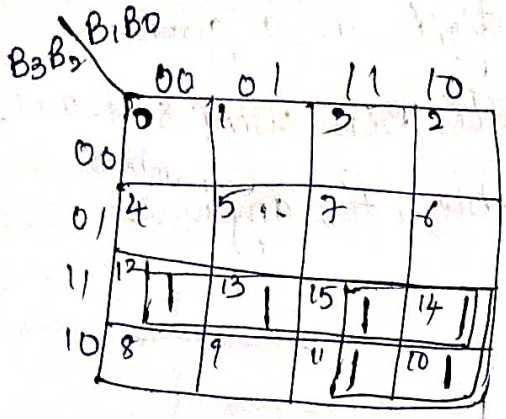
$D = \Sigma(2, 3, 6, 7, 12, 13)$

$E = \Sigma(1, 3, 5, 7, 9, 11, 13, 15)$

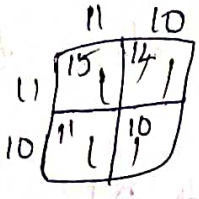
here

$E = B_0$

* k-map for $A = \Sigma(10, 11, 12, 13, 14, 15)$

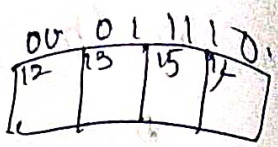


Group-1:



$\Rightarrow B_3B_1$

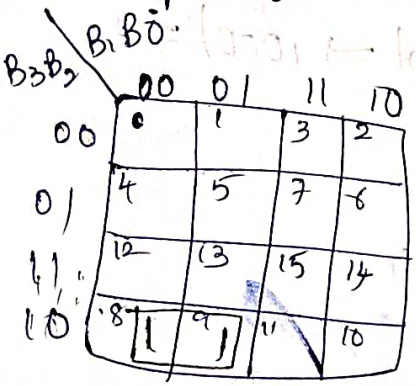
Group-2



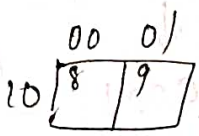
$\Rightarrow B_3B_2$

$\Rightarrow A = B_3B_1 + B_3B_2$

* k-map for $B = \Sigma(8, 9)$



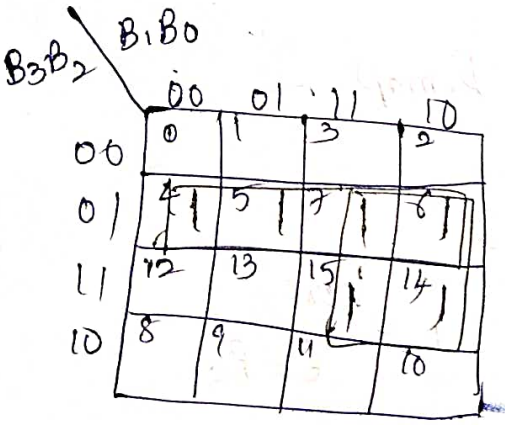
Group 1



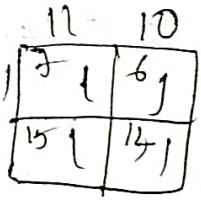
$\Rightarrow B_3B_2'B_1'$

$\therefore B = B_3B_2'B_1'$

* k-map for $C = \Sigma(4, 5, 6, 7, 14, 15)$

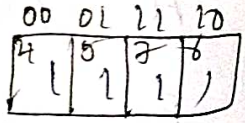


Group-1



$\Rightarrow B_2B_1$

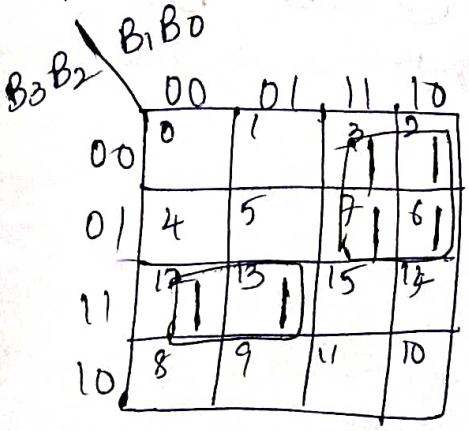
Group-2



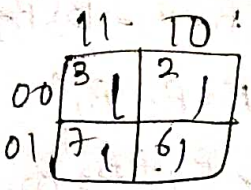
$\Rightarrow B_3'B_2$

$\therefore C = B_2B_1 + B_3'B_2$

*k-map for $D = \Sigma(2,3,6,7,12,13)$

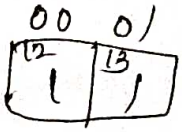


Group-1



$\Rightarrow B_3'B_1$

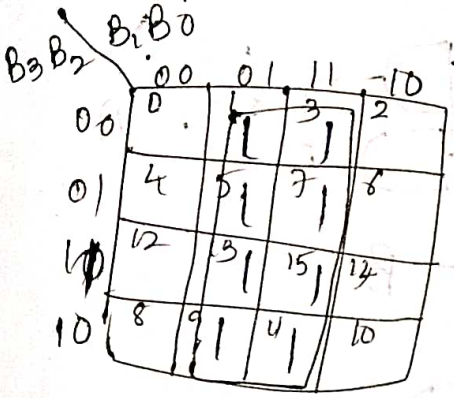
Group-2



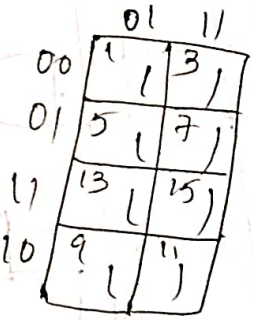
$\Rightarrow B_3B_2B_1'$

$D = B_3'B_1 + B_3B_2B_1'$

* k-map for $C = \Sigma(1,3,5,7,9,11,13,15)$



Group



$\Rightarrow B_0$

$C = B_0$ (we directly predict it already. It is ~~just for verification~~ verified and confirmed here).

Step-4:- Circuit diagram using logic gates.

Obtained Expressions are:

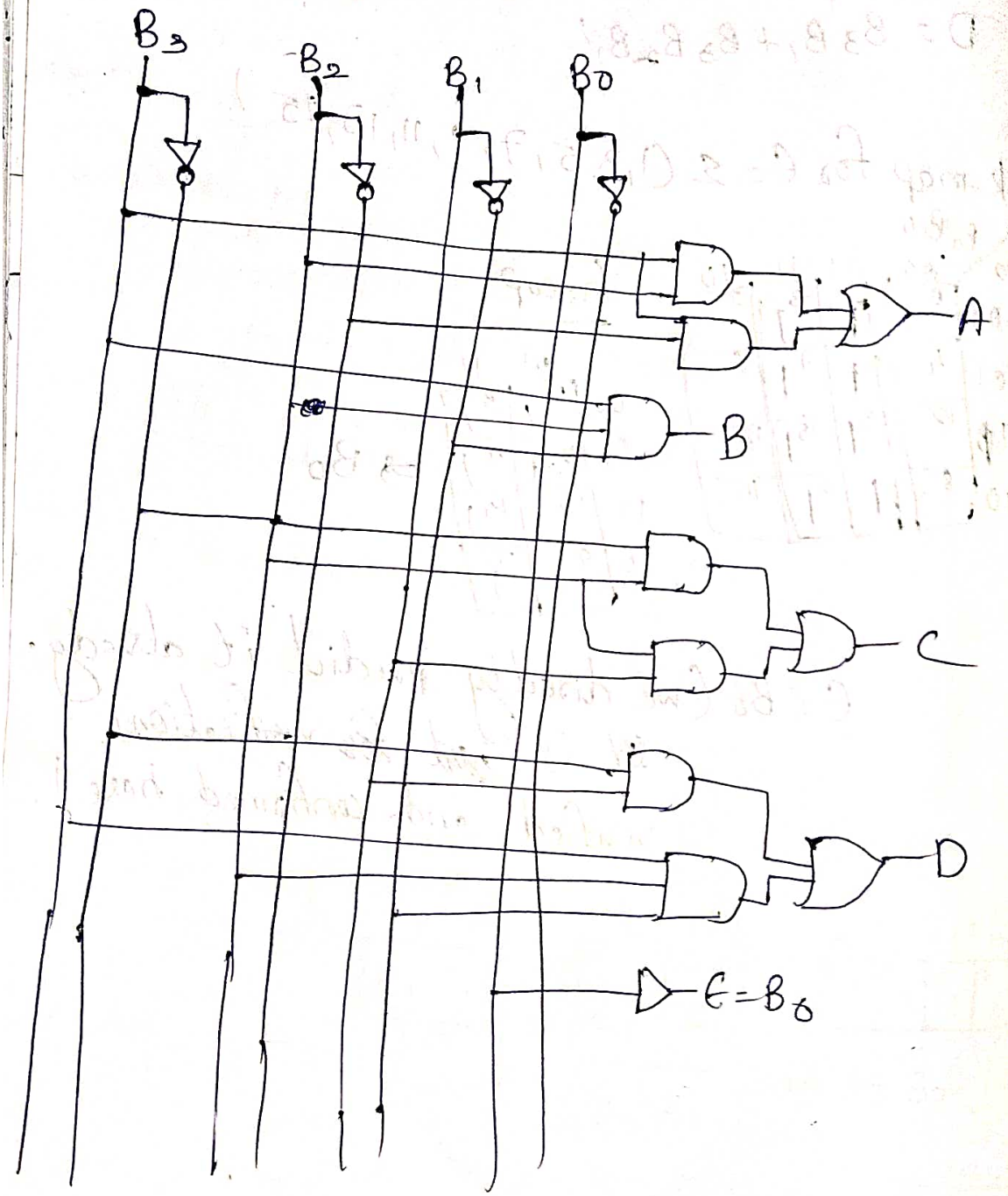
$A = B_3B_1 + B_3B_2$

$B = B_3B_2'B_1'$

$C = B_2B_1 + B_3'B_2$

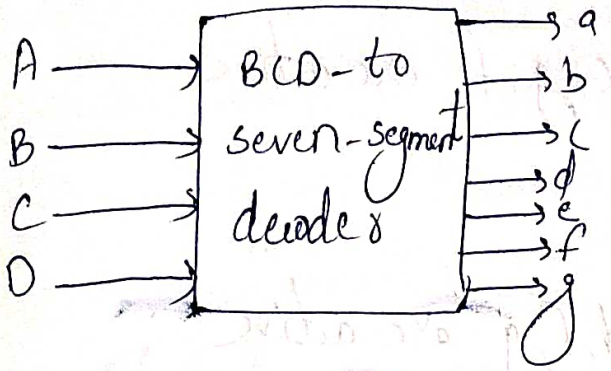
$D = B_3'B_1 + B_3'B_2B_1'$

$E = B_0$



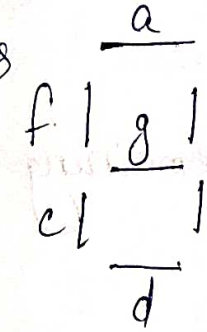
*BCD to seven-segment decoder:-

step-1: Draw block diagram.



here number of inputs are 4-ABCD number of outputs are 7-a,b,c,d, e,f,g

⇒



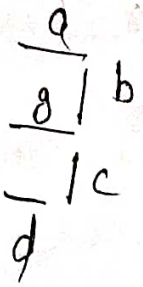
⇒ structure

From the above structure:-

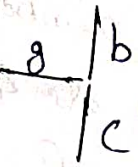
0:- a, f, b, e, c, d ⇒ g is absent, remaining are active only

1:- b, c ⇒ b, c are present/active

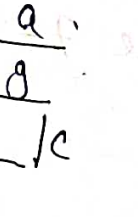
2:- a, g, b, e, d ⇒ a, b, d, e, g are present/active

3:- 

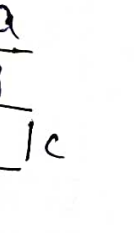
a, b, g, c, d are active

4:- 

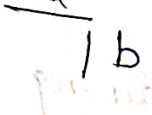
b, c, g, f are active

5:- 

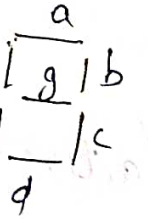
a, c, d, f, g are active

6:- 

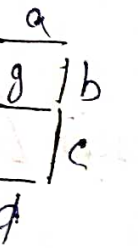
only b is absent, remaining are active

7:- 

a, b, c are active

8:- 

All are active

9:- 

only c is ~~obj~~ absent, remaining are active

Step-2: Construct truth table :-

Decimal digit	Inputs				outputs						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

$a = \Sigma(0,2,3,5,6,7,8,9)$

$b = \Sigma(0,1,2,3,4,7,8,9)$

$c = \Sigma(0,1,3,4,5,6,7,8,9)$

$d = \Sigma(0,2,3,5,6,8,9)$

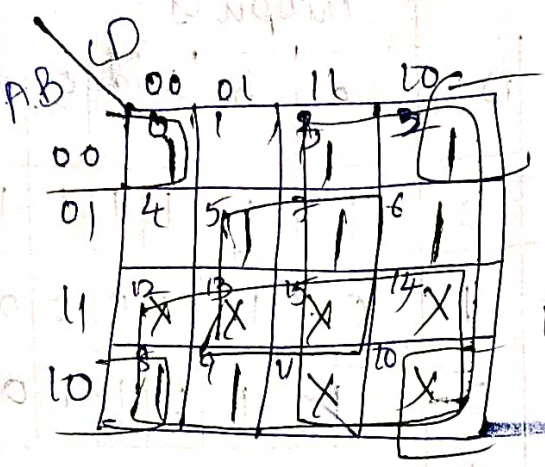
$e = \Sigma(0,2,6,8)$

$f = \Sigma(0,4,5,6,8,9)$

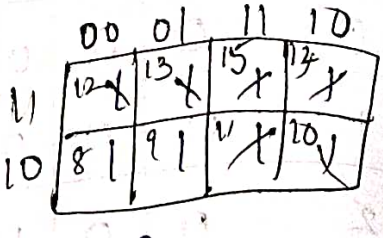
$g = \Sigma(2,3,4,5,6,8,9)$

Step-3:
Simplification using k-map.

* k-map for $a = \Sigma(0,2,3,5,6,7,8,9) + d(10,11,12,13,14,15)$

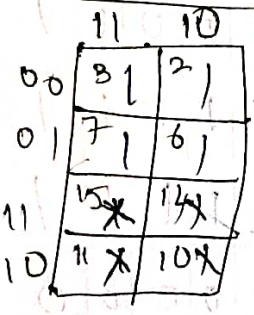


Group-1



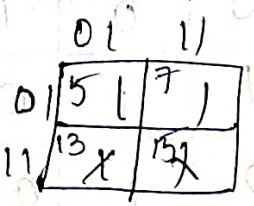
$\Rightarrow A$

Group-2:



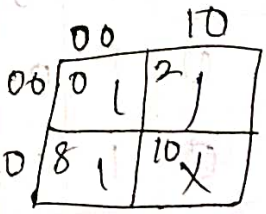
$\Rightarrow C$

Group-3



$\Rightarrow BD$

Group-4

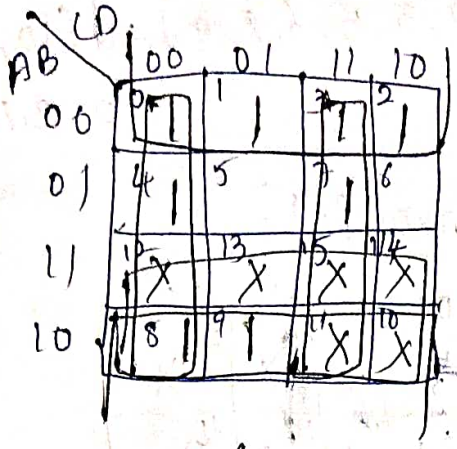


$\Rightarrow B'D'$

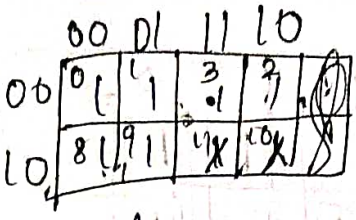
$\Rightarrow a = A + C + BD + B'D'$

{ Note: We have to use don't cares here. And don't cares for bcd are 10, 11, 12, 13, 14, 15. }

* k-map for ~~b = B + C~~ $b = \Sigma(0,1,2,3,4,7,8,9) + d(10,11,12,13,14,15)$

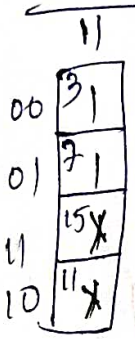


Group-1



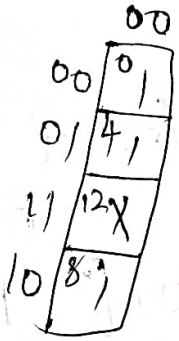
$\Rightarrow B'$

Group-2



$\Rightarrow CD$

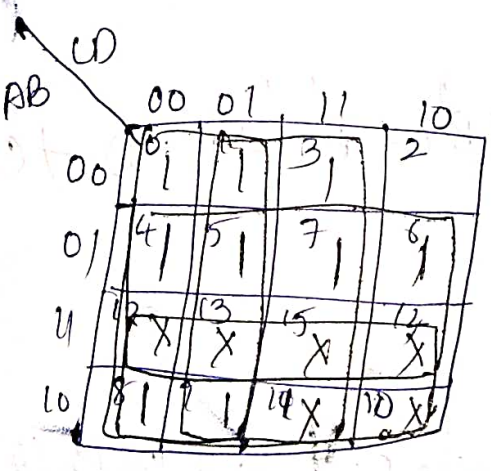
Group-3



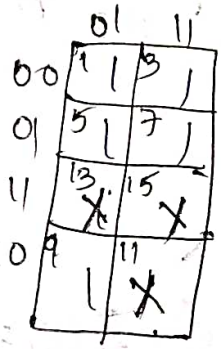
$\Rightarrow C'D'$

$\therefore b = B' + CD + C'D'$

* k map for $C = \Sigma(0,1,3,4,5,6,7,8,9) + d(10,11,12,13,14,15)$

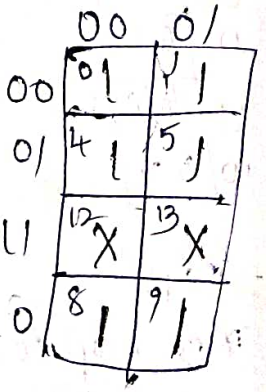


Group-1



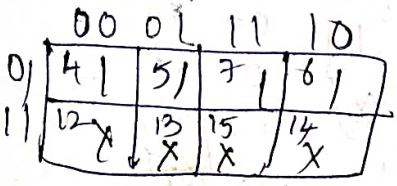
$\Rightarrow D$

Group-2



$\Rightarrow C'$

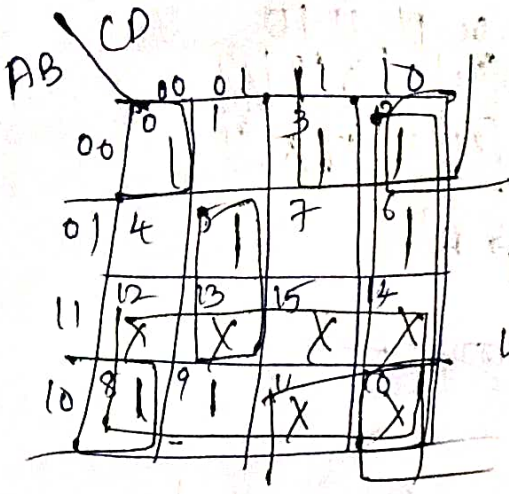
Group-3



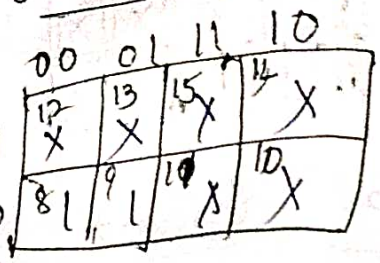
$\Rightarrow B$

$\therefore C = D + C' + B$

* k-map for $d = \Sigma(0,2,3,5,6,8,9) + d(10,11,12,13,14,15)$

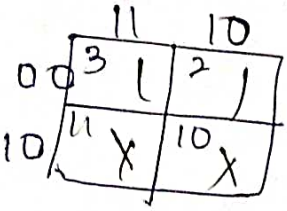


Group-1 :



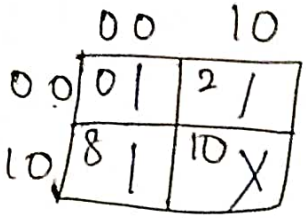
$\Rightarrow A$

Group-2 :



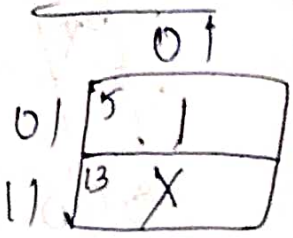
$\Rightarrow B'C$

Group-3



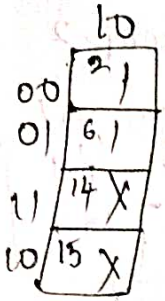
$\Rightarrow B'D'$

Group-4



$\Rightarrow BC'D$

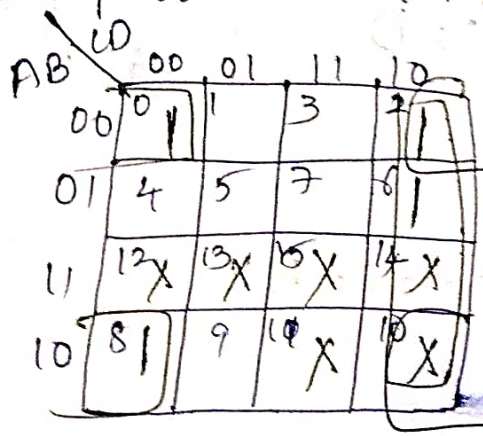
Group-5 :



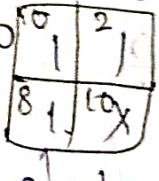
$\Rightarrow CD'$

$\therefore d = A + B'C + BD' + BC'D + CD'$

* k map for $e = \Sigma(0,2,6,8) + d(10,11,12,13,14,15)$

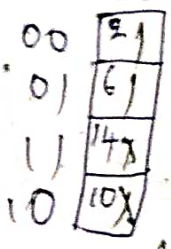


Group-1



$\Rightarrow BD'$

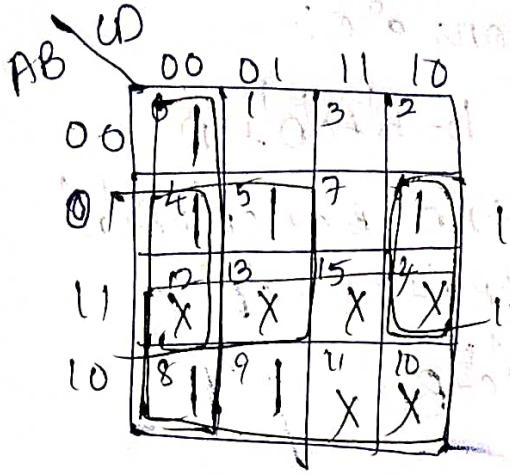
Group-2



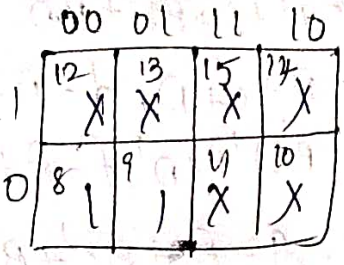
$\Rightarrow CD'$

$\therefore e = B'D' + CD'$

* k-map for $f = \Sigma(0,4,5,6,8,9) + d(10,11,12,13,14,15)$

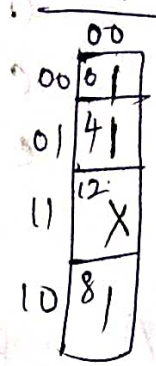


Group-1



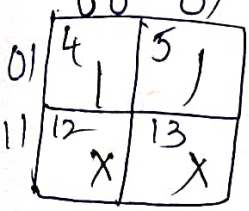
$\Rightarrow A$

Group-2



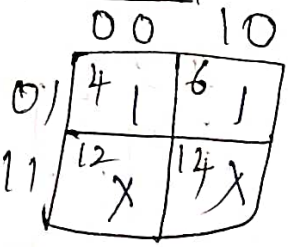
$\Rightarrow C'D'$

Group-3



$\Rightarrow BC'$

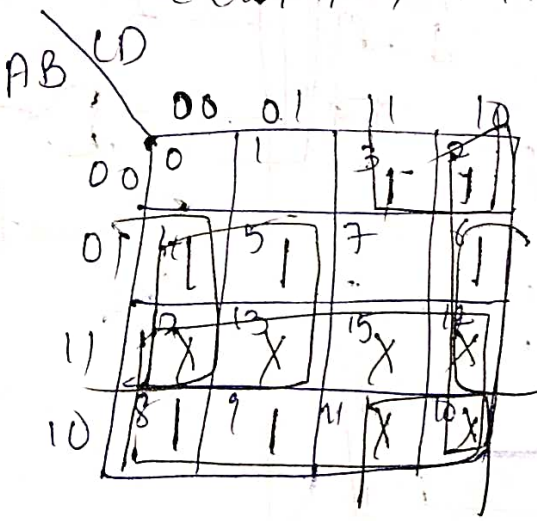
Group-4



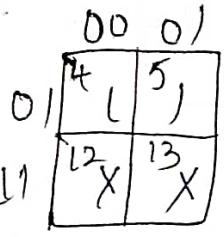
$\Rightarrow BD'$

$\therefore f = A + C'D' + BC' + BD'$

* k-map for $g = \Sigma(2,3,4,5,6,8,9) + d(10,11,12,13,14,15)$

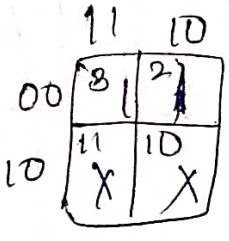


Group-1



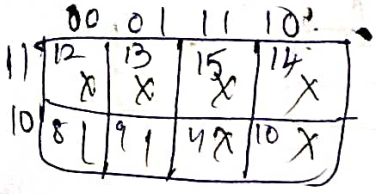
$\Rightarrow BC'$

Group-2



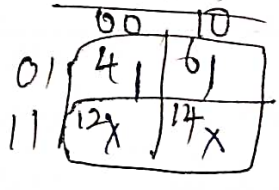
$\Rightarrow B'C$

Group-3



$\Rightarrow A$

Group-4



$\Rightarrow BD'$

$\therefore g = BC' + B'C + A + BD'$

Step-4 :- Circuit diagram using logic gates.

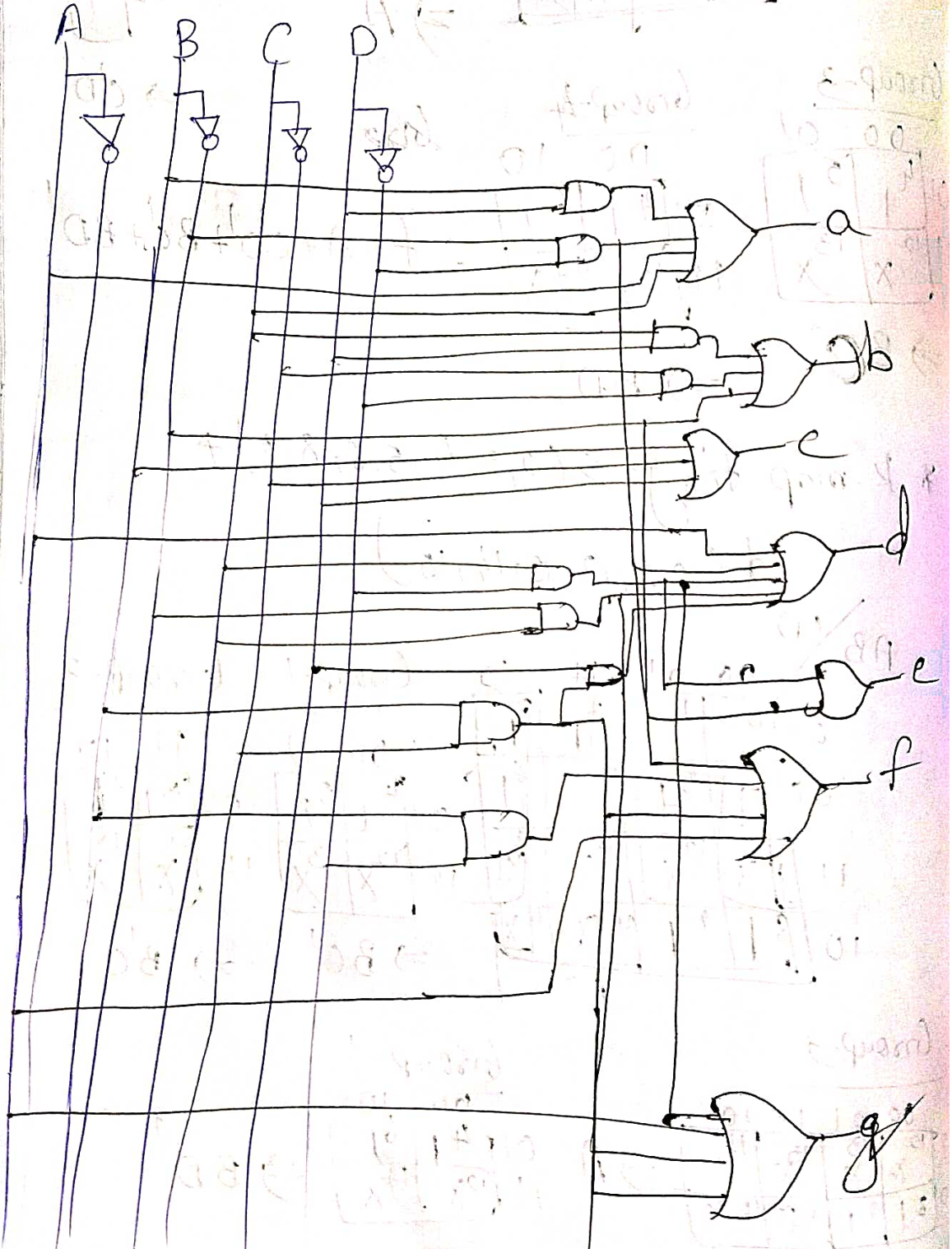
Obtained Expressions are :-

$a = A + C + BD + B'D'$, $b = B' + C'D' + CD$

$c = B + C' + D$, $d = A + B'D' + B'C + CD' + BC'D'$

$e = B'D' + CD'$, $f = A + BC' + BD' + C'D'$;

$g = A + BC' + B'C + BD'$.



$\therefore g = BC' + B'C + A + BD'$

Step-4:- Circuit diagram using logic gates

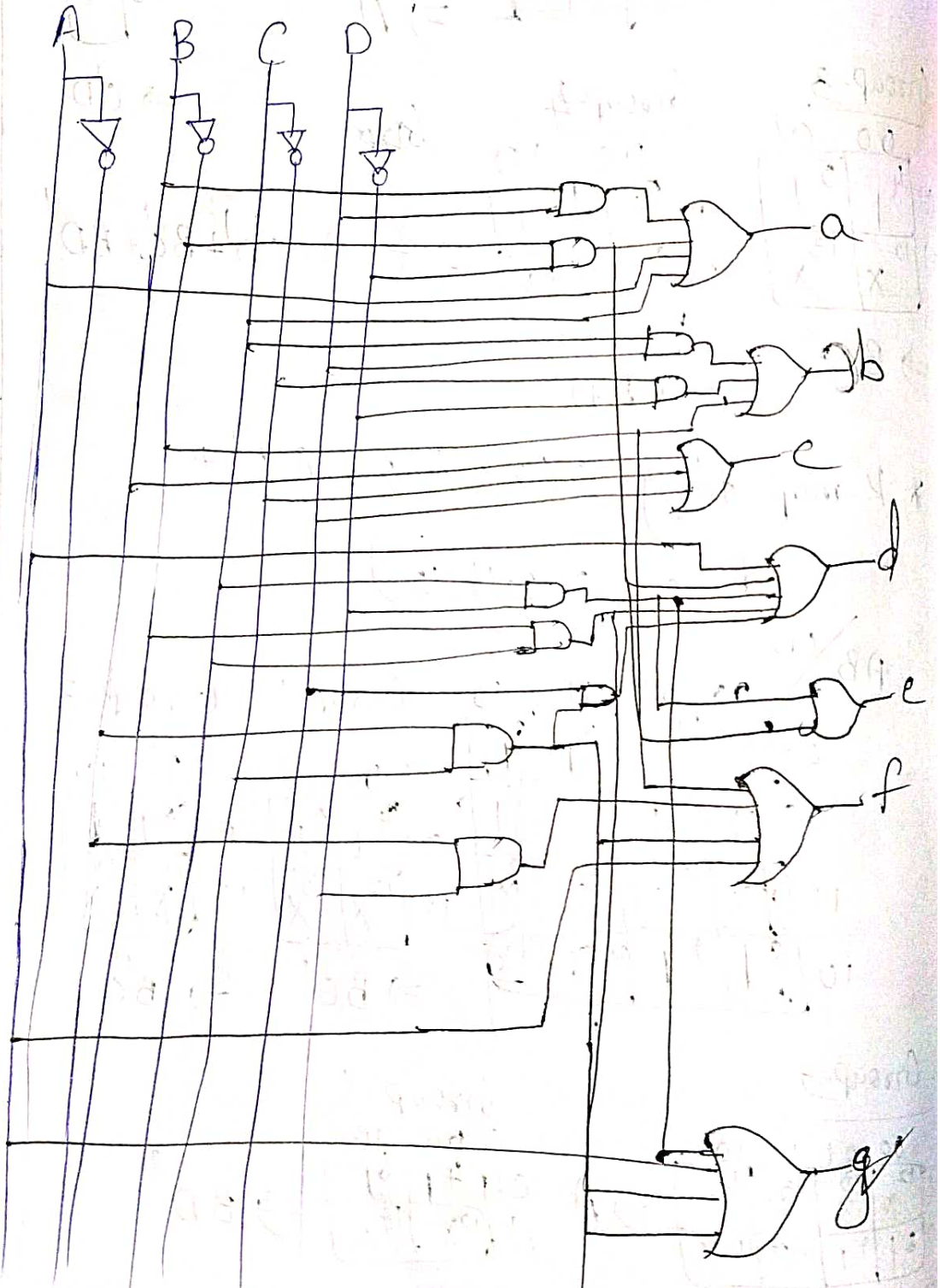
Obtained Expressions are:-

$a = A + C + BD + B'D'$, $b = B' + C'D' + CD$

$c = B + C' + D$, $d = A + B'D' + B'C + CD' + BC'D'$

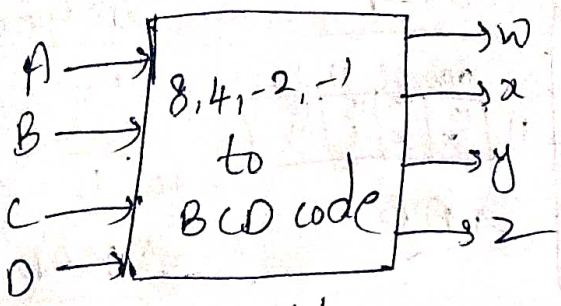
$e = B'D' + CD'$, $f = A + BC' + BD' + C'D'$;

$g = A + BC' + B'C + BD'$.



8,4,-2,-1 to BCD code:

step-1: Block diagram:-



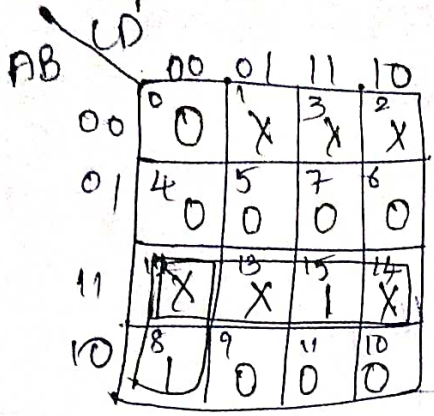
step-2: Truth table

Decimal digit	8	4	-2	-1	8	4	2	1
	A	B	C	D	w	x	y	z
0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	1
2	0	1	1	0	0	0	1	0
3	0	1	0	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	1	0	1	1	0	1	0	1
6	1	0	1	0	0	1	1	0
7	1	0	0	1	0	1	1	1
8	1	0	0	0	1	0	0	0
9	1	1	1	1	1	0	0	1

Step-3:

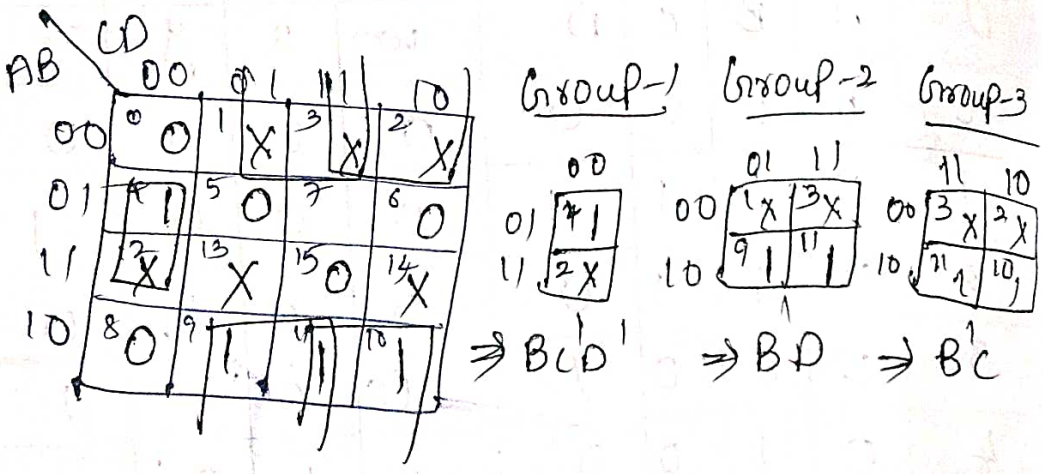
k-map for simplification.

* k-map for $W = \Sigma(8,15) + d(1,2,3,12,13,14)$



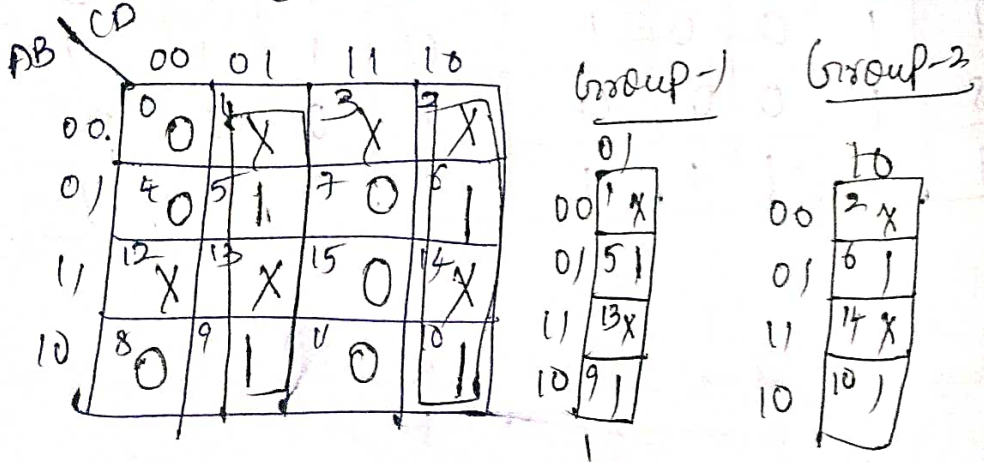
$\therefore W = AB + AC'D'$

* k-map for $x = \Sigma(4,9,10,11) + d(1,2,3,12,13,14)$



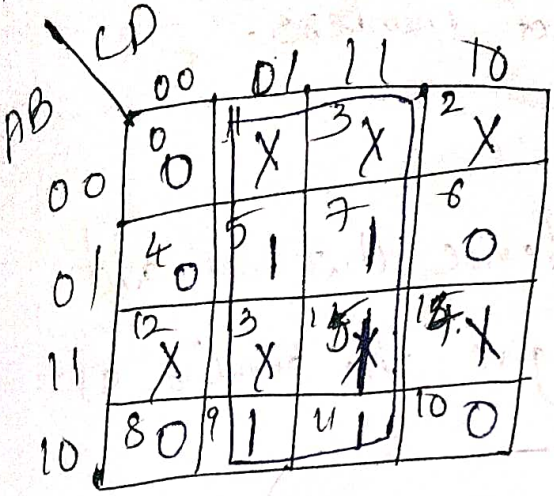
$\therefore x = B'D + B'C + BC'D'$

* Kmap for $y = \Sigma(5,6,9,10)$

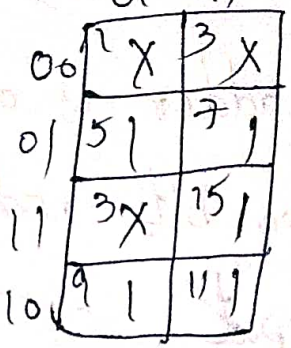


$\therefore y = c'D + cD' = c \oplus D$

* k-map for $z = \Sigma(5, 7, 9, 11, 15) + d(1, 2, 3, 12, 13, 14)$



Group



$\Rightarrow D$

$\therefore z = D$

* ADDERS/SUBTRACTORS:

ADDERS: An adder is a digital circuit that performs addition of numbers.

(or)

An adder is a device that will add together two bits and give the results as the output.

There are two kinds of adders:

1. Half adders
2. Full adders.

① Half adders (HA): A Half adder is a combinational logic circuit which is designed by connecting one EX-OR gate and one AND gate. The half adder circuit has two inputs: A and B, which add two input digits and generates a carry(C) and a sum(S). Thus, this is called Half Adder circuit.

Step-1: Block diagram

step-2:- Truth table

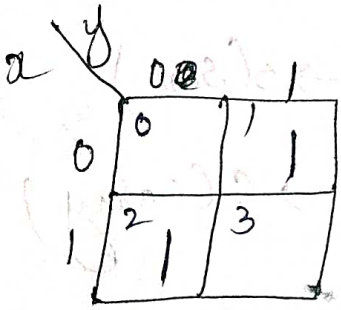
	Inputs		Outputs	
	x	y	S	C
0	0	0	0	0
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1

Here, $S = \Sigma(1,2)$

$C = \Sigma(3)$

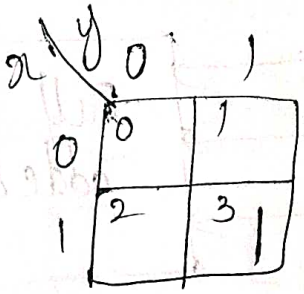
step-3:- Simplification using k-map:-

* k-map for $S = \Sigma(1,2)$



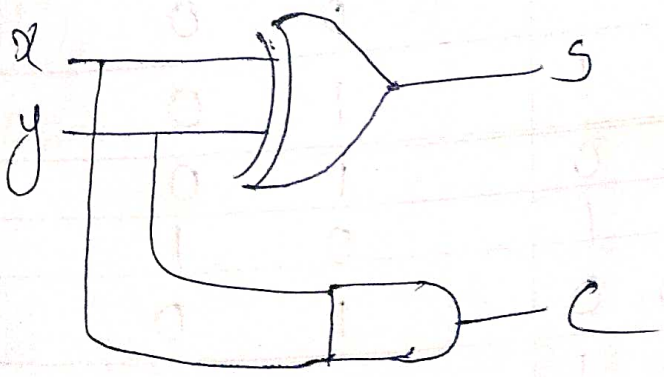
$S = x'y + xy'$

* k map for $C = \Sigma(3)$



$C = x.y$

step-4:- Circuit diagram using logic gates.

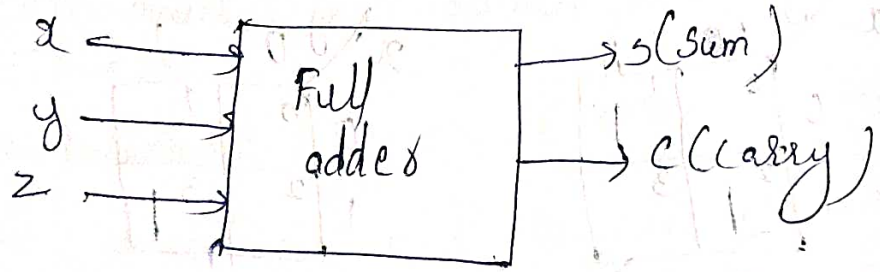


Full adder:- Full adder is the adder which adds three inputs and produces two outputs.

⇒ The first two inputs are A and B and the third input is an input carry. as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM.

⇒ Here three inputs are nothing but two bits significant & one is previous carry.

Step-1:- Block diagram



Step-2:- Truth Table

Decimal digit	Inputs			Outputs	
	x	y	z	S	C
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

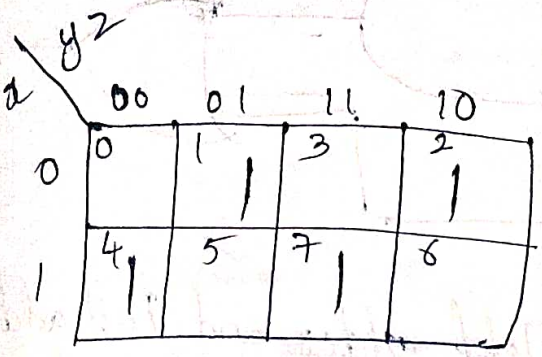
here we get,

$S = \Sigma(1,2,4,7)$

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

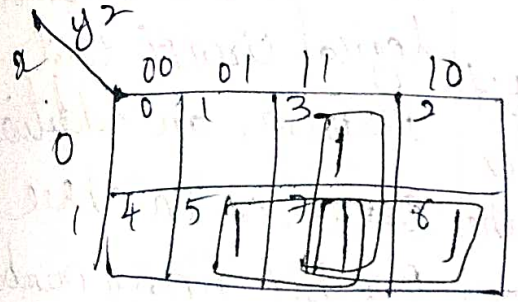
step-3:- Simplification using k-map

* k-map for $S = \Sigma(1,2,4,7)$

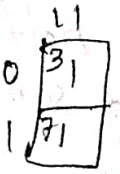


$S = x'y'z + x'yz' + xy'z' + xyz$

* k-map for $C = \Sigma(3,5,6,7)$

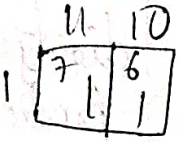


Group-1



$\Rightarrow yz$

Group-2



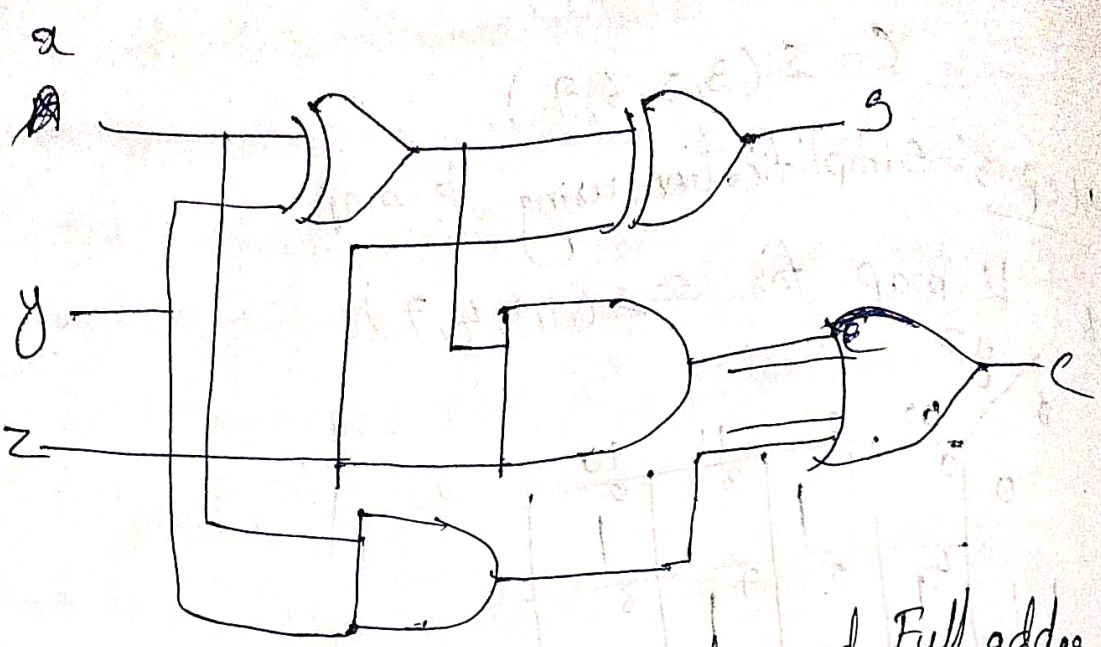
$\Rightarrow xy$

Group-3

$\Rightarrow xz$

$\therefore C = yz + xy + xz$

Step-4: Circuit diagram using logic gates.



* Differences between Half adder and Full adder

S.No	Half adder	Full adder
1	Half Adder is combinational logic circuit which adds two 1-bit digits. The half adder produces a sum of the two inputs.	Full adder is combinational logical circuit that performs an addition operation on three one-bit binary numbers. The full adder produces a sum of the three inputs and carry value.
2.	Previous carry is not used	Previous carry is used.
3	In Half adder there are two input bits (A,B)	In full adder there are three input bits (A,B,C-in).

S.NO	Half Adder	Full Adder
4	Logical Expression for half adder is $S = a \oplus b$; $C = a * b$.	Logical Expression for full adder is; $S = a \oplus b \oplus cin$; $Cout = (a * b) + (Cin * (a \oplus b))$.
5	It consists of one EX-OR gate and one AND gate	It consists of two EX-OR, two AND gate and one OR gate
6	It is used in calculators, computers, digital measuring devices etc.	It is used in Multiplied bit addition, digital processors etc.

Subtractor: A combinational circuit that performs the subtractions of bits is called a subtractor.

⇒ Subtractor circuits take two binary numbers as input and subtract one binary number input from the other binary number input. Similar to adders, it gives out two outputs, difference and borrow (carry-in the case of Adder). There are two types of subtractors. They are:-

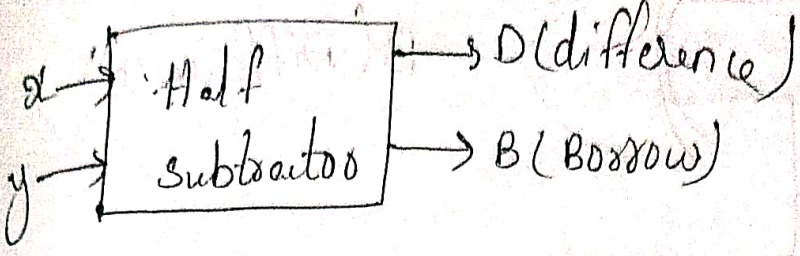
1. Half subtractor
2. Full subtractor.

① Half Subtractor: The half-subtractor is a combinational circuit which is used to perform subtraction of two bits.

⇒ It has two inputs, A (minuend) and B (subtrahend) and two outputs (Difference and Borrow)

⇒ In simple words, half subtractor is a combinational circuit (c.c) which subtracts the given two bits and produces difference.

step-1: Block diagram:



step-2: Truth Table:

Decimal digit	Inputs		Outputs	
	x	y	D	B
0	0	0	0	0
1	0	1	1	1
2	1	0	1	0
3	1	1	0	0

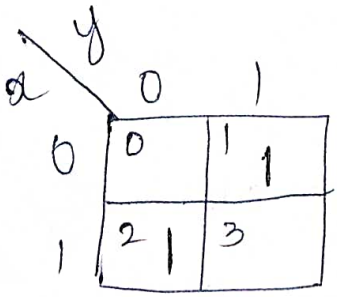
here,

$D = \Sigma(1, 2)$

$B = \Sigma(1)$

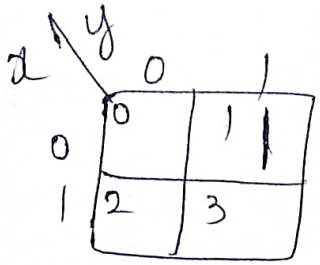
step-3: Simplification using k-map.

* k map for $D = \Sigma(1, 2)$



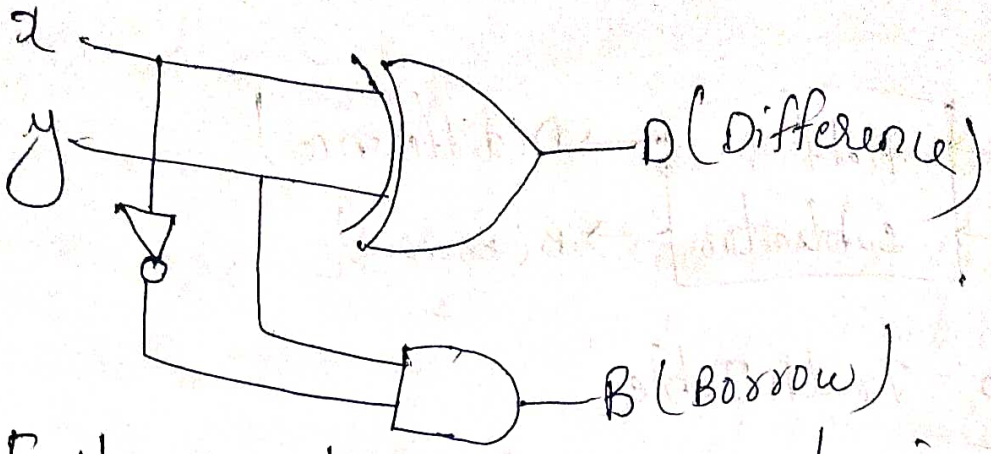
$\Rightarrow D = x'y + y'x = x \oplus y$

* kmap for $B = \Sigma(1)$



$\Rightarrow B = x'y$

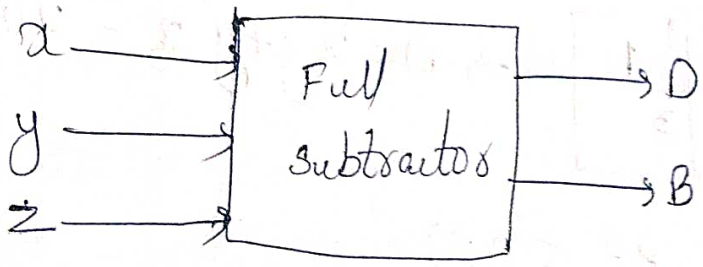
Step-4:- Circuit diagram using logic gates



Full Subtractor:- A full subtractor is a combinational circuit that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit.

→ This circuit has three inputs and two outputs.

Step-1:- Block diagram



step-2: Truth table.

Decimal digit	Inputs.			Outputs	
	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

Here, $D = \Sigma(1, 2, 4, 7)$

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

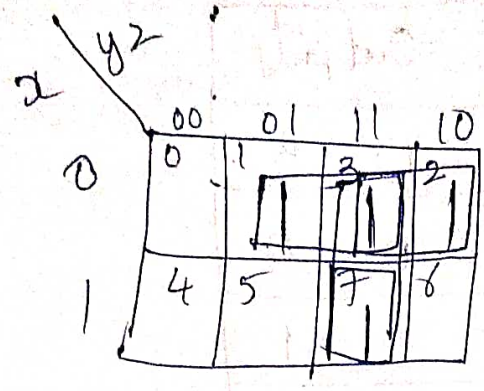
step-3: simplification using k-map.

* k map for $D = \Sigma(1, 2, 4, 7)$

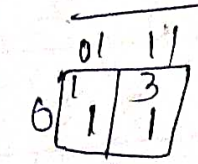
x \ yz	00	01	11	10
0	0	1 1	3	2 1
1	4 1	5	7 1	6

$D = xy'z' + x'y'z + xyz + x'yz'$

* k-map for $B = \Sigma(1,2,3,7)$

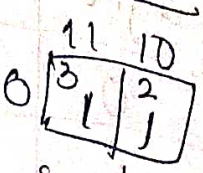


Group-1



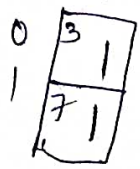
$\Rightarrow x'z$

Group-2



$\Rightarrow x'y$

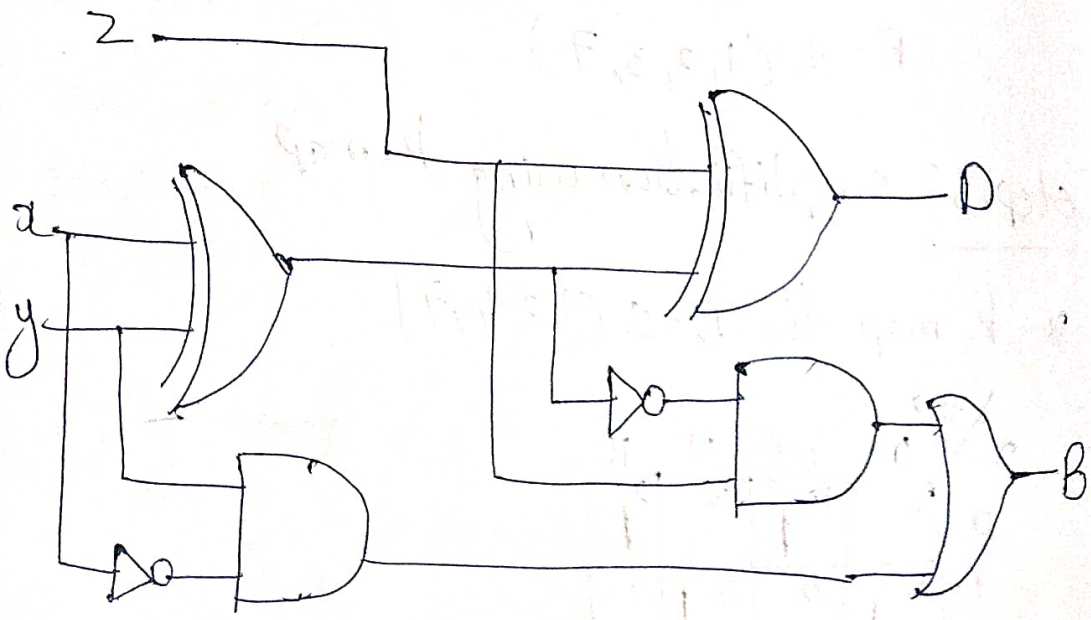
Group-3



$\Rightarrow yz$

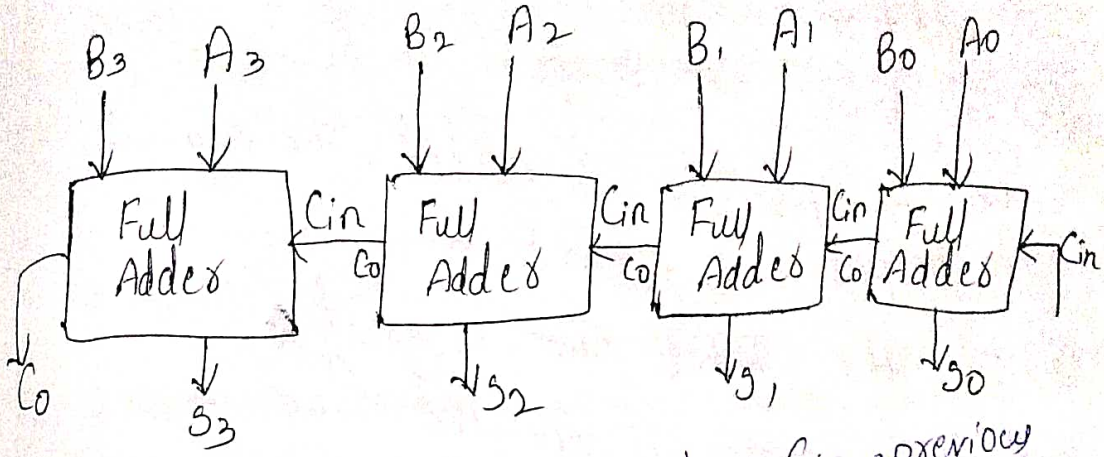
$\therefore B = x'z + x'y + yz$

Step-4 :- Circuit diagram using logic gates



* Multi-bits adder which perform addition more than two bits.

Eg: 4-bit Binary Adder (Parallel):-



For $A \rightarrow A_3 A_2 A_1 A_0$, here Cin → previous carry

$B \rightarrow B_3 B_2 B_1 B_0$ Co → Carry out

Here when we perform addition, firstly A_0 and B_0 are added, for this Cin is 0 initially. After adding it will give Co and Cin. Later we have to added $A_1 B_1$ and Cin that we get from the addition of $A_0 B_0$. Like this we perform.

* Here if we try to perform 4-bit Binary subtraction, we have to add inverter to B.

For subtraction,

$A - B \Rightarrow A + (-B)$ → 4-bit parallel subtraction.

for finding $-B$ do 2's complement of B. Then we can find $A-B$.

* BCD Adder:

⇒ BCD addition procedure as follows:-

① Add two BCD numbers using ordinary binary addition.

② If 4-bit sum is Equal to or less than 9, no correction is needed. The sum is in proper BCD form.

③ If the 4-bit sum is greater than 9 or if a carry is generated from 4-bit sum, the sum is invalid.

④ To correct the invalid sum, add $(0110)_2$ to the four-bit sum. If a carry results from this addition, add it to the next higher-order BCD digit.

Ex:

$$\begin{array}{r} 23 \\ +\ 44 \\ \hline 67 \end{array} \quad \begin{array}{r} 0010\ \ 0011 \\ +\ 0100\ \ 0100 \\ \hline 0110\ \ 0111 \end{array}$$

(2 → 0010, 3 → 0011, 4 → 0100, 4 → 0100, 0110 → 6, 0111 → 7)

67

Ex:

Ex: 36 – 0011 0110
45 – 0100 0101
81 1111 1
1011 → 10 → invalid sum. so add 0110

0110
1000 0001
8 1
81

Ex: 24 – 0010 0100
86 – 1000 0110
110 1010 1010
1010 = 10 invalid sum. so add 0110 on both sides
0110 0110
1001 0000
1 1 0
⇒ 110 (here we divide them into 4 bits)

⇒ To implement BCD adder we require:-
① 4-bit binary adder for initial addition
② Logic circuit to detect sum greater than 9
③ One more 4-bit adder to add $(0110)_2$ in the sum if sum is greater than '9' & carry is '1'.

⇒ The logic circuit to detect sum greater than '9' can be determined by simplifying the Boolean Expression of given truth table.

$$Y = S_3S_2 + S_3S_1$$

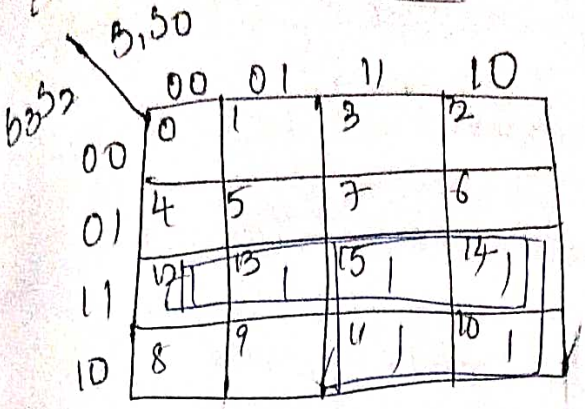
⇒ Y=1 ;indicates sum is greater than 9 we can put one more term like C_{out} in above Expression to check whether Carry is one. If any one condition is satisfied we add $6(0110)_2$ in the sum.

* Truth table:

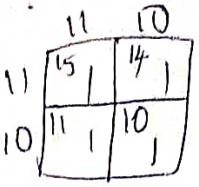
	Inputs				Outputs
Decimal digit	S_3	S_2	S_1	S_0	Y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

here, $Y = \Sigma(10, 11, 12, 13, 14, 15)$

* K-map simplification:

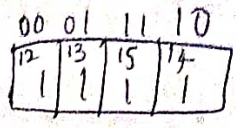


Group-1



$\Rightarrow S_3S_1$

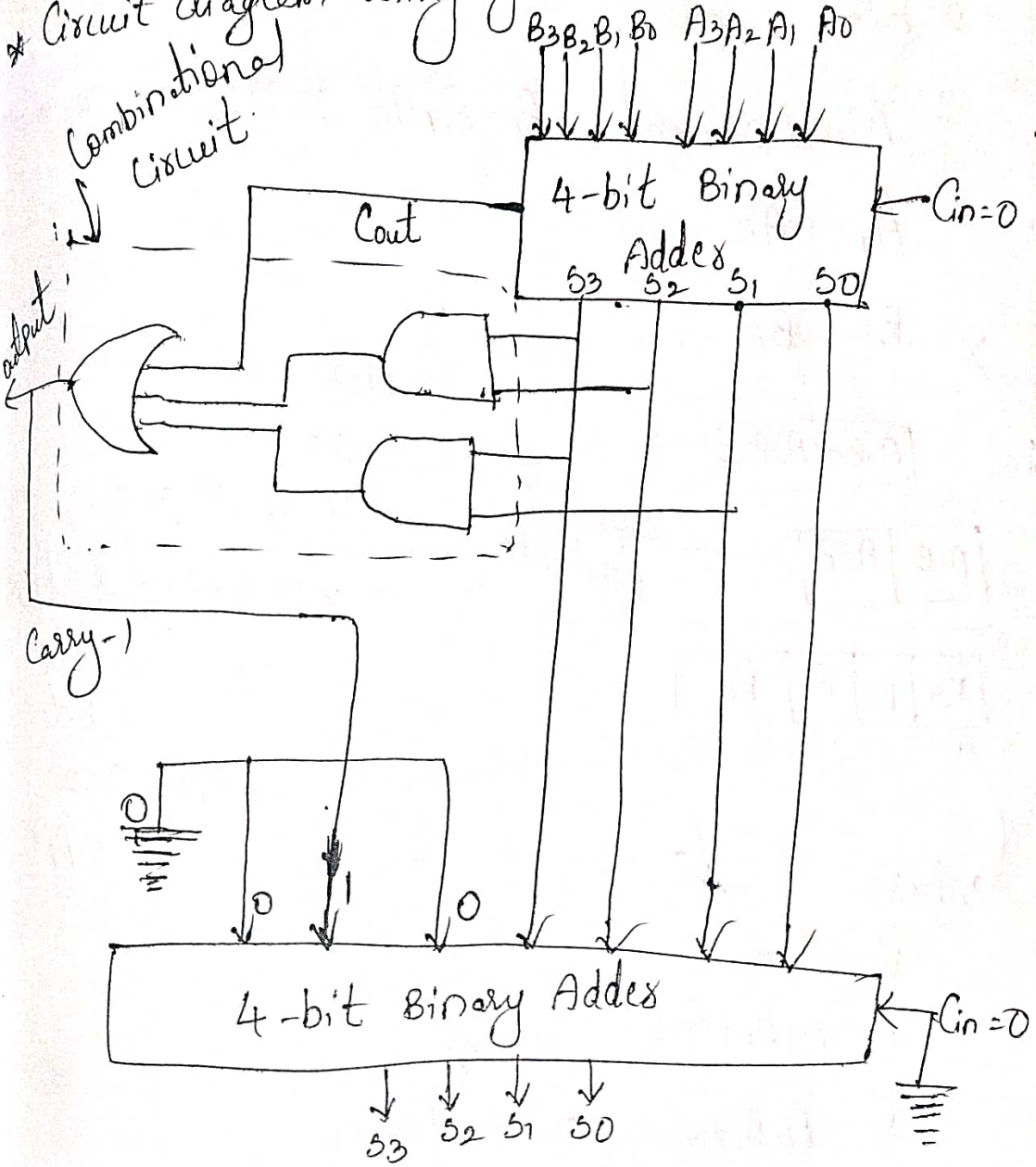
Group-2



S_3S_2

$\therefore Y = S_3S_2 + S_3S_1$

* Circuit diagram using gates.



* Implementation of BCD Adder.

* Binary Multiplier: It is a combinational circuit (c.c) in which the multiplicant is multiplied by each bit of a multiplier starting from LSB.

⇒ Let us generalize the multiplication process for a 2x2 multiplier for two unsigned 2-bit numbers.

Multiplicant $A = A_1A_0$ & Multiplier $B = B_1B_0$

A_1 A_0

X B_1 B_0

A_1B_0 | A_0B_0

A_1B_1 | A_0B_1 → partial products

P_3 | P_2 | P_1 | P_0

Note: If there are n×n bits it will give 2n partial products (PP)

Ex:- 2 bit X 2 bit

⇒ 2n = 2×2 = 4 PP

here

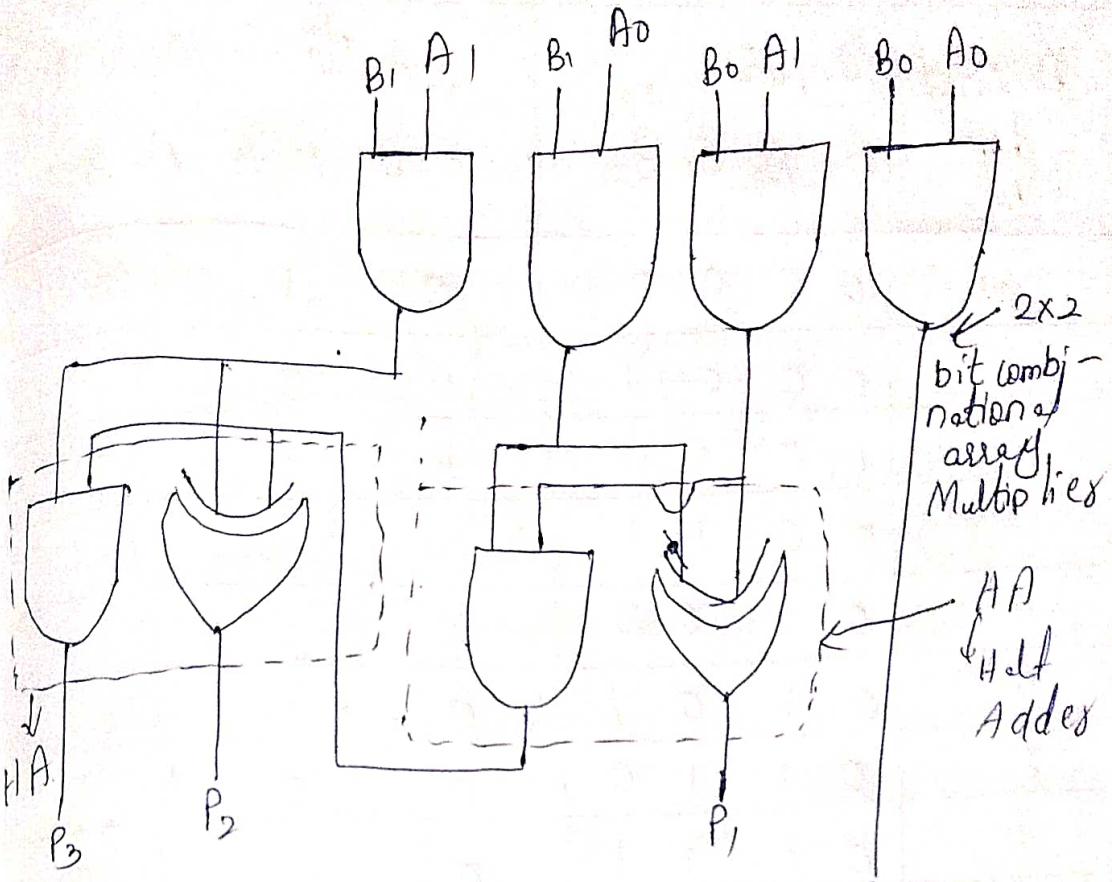
$P_0 = A_0B_0$

$P_1 = A_1B_0 + A_0B_1$

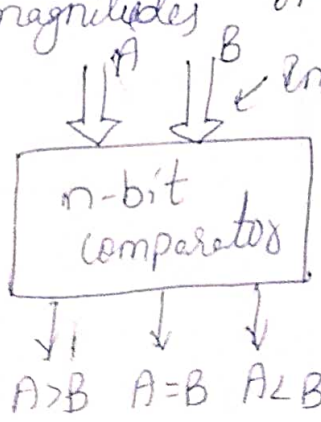
$P_2 = A_1B_1$ + Carry out of P_1

P_3 = Carry out of P_2

* Multiplication process:-



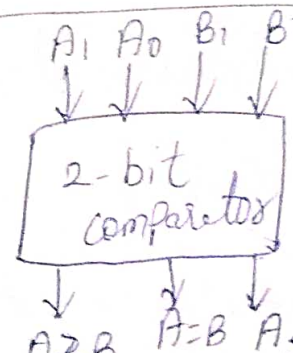
Magnitude Comparator:- It is a combinational circuit designed primarily to compare the relative magnitudes of two binary numbers.



A → (0, 1, 2, -- n)
B → (0, 1, 2, -- n)

* Design of 2 bit Magnitude Comparator.

Step-1:- Block diagram



A → A1 A0
B → B1 B0

Step-2: Truth table.

Decimal digit	Inputs A_1 A_0 B_1 B_0	Outputs $A>B$	$A=B$	$A<B$
0	0 0 0 0	0	1	0
1	0 0 0 1	0	0	1
2	0 0 1 0	0	0	1
3	0 0 1 1	0	0	1
4	0 1 0 0	1	0	0
5	0 1 0 1	0	1	0
6	0 1 1 0	0	0	1
7	0 1 1 1	0	0	1
8	1 0 0 0	1	0	0
9	1 0 0 1	1	0	0
10	1 0 1 0	0	1	0
11	1 0 1 1	0	0	1
12	1 1 0 0	1	0	0
13	1 1 0 1	1	0	0
14	1 1 1 0	1	0	0
15	1 1 1 1	0	1	0

Here

$A>B = \Sigma(4, 8, 9, 12, 13, 14)$

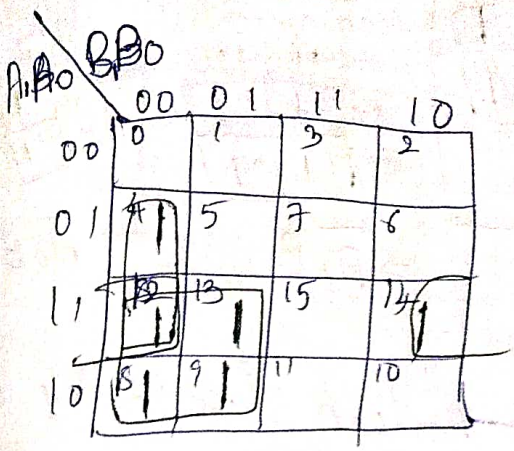
$A=B = \Sigma(0, 5, 10, 15)$

$A<B = \Sigma(1, 2, 3, 6, 7, 11)$

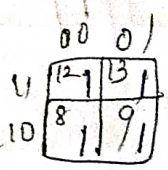
Step-3: Simplification using k-map.

* k-map for A>B:-

$A>B = \Sigma(4, 8, 9, 12, 13, 14)$

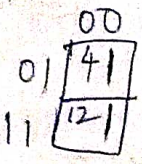


Group-1



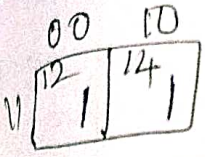
$\Rightarrow A_1\bar{B_1}$

Group-2



$\Rightarrow A_0\bar{B_1}\bar{B_0}$

Group-3



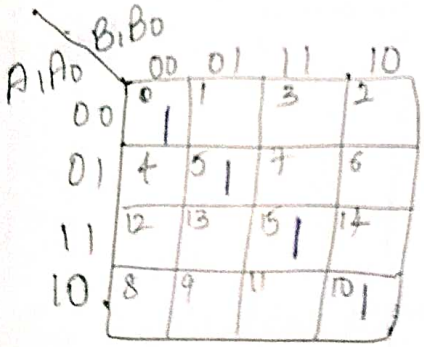
$\Rightarrow A_1A_0\bar{B_0}$

∴ ~~Simplified~~ The boolean Expressions is

$A>B = A_0\bar{B_1}\bar{B_0} + A_1\bar{B_1} + A_1A_0\bar{B_0}$

* k-map for A=B:-

$A=B \Rightarrow \Sigma(0, 5, 10, 15)$



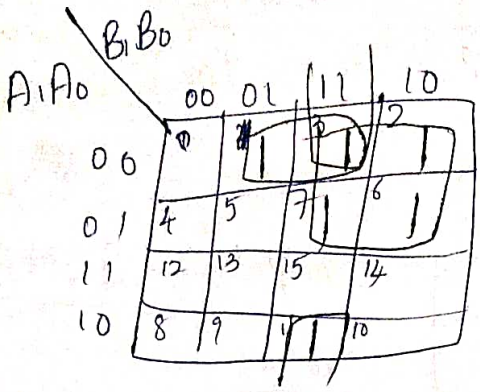
$A=B \Rightarrow \bar{A_1}\bar{A_0}\bar{B_1}\bar{B_0} + \bar{A_1}A_0\bar{B_1}B_0 + A_1A_0B_1B_0 + A_1\bar{A_0}B_1\bar{B_0}$

$= [(A_0 \text{ xnor } B_0) \cdot (A_1 \text{ xnor } B_1)]$

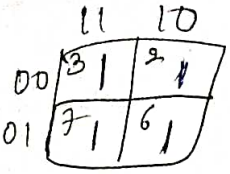
$[\because (x \oplus y)' = xy + x'y']$

*K-map for $A<B$:-

$$A<B = \Sigma(1,2,3,6,7,11)$$



Group-1

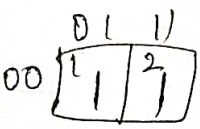


$\Rightarrow \overline{A_1}B_1$

Group-2

$\Rightarrow \overline{A_0}B_1B_0$

Group-3

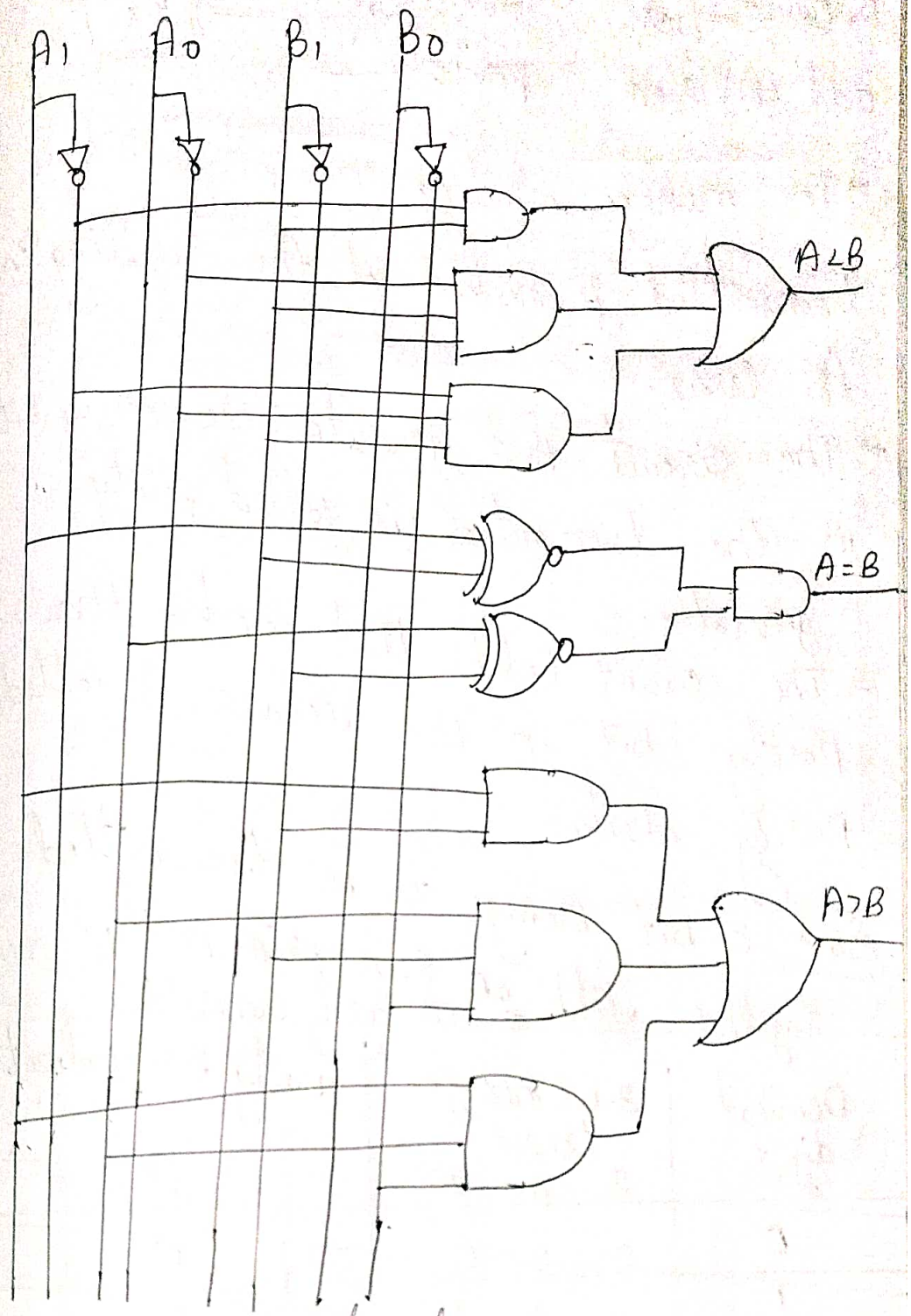


$\Rightarrow \overline{A_1}\,\overline{A_0}B_0$

The Boolean Expression of

$$A<B = \overline{A_1}B_1 + \overline{A_0}B_1B_0 + \overline{A_1}\,\overline{A_0}B_0$$

Step-4:- Logic Diagram

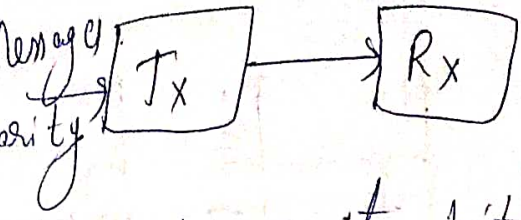


* Parity Generators / checkers :-

⇒ Parity bit is used to detect Errors during transmission of message.

⇒ A parity bit is an Extra bit included with a binary message to make

the number of ones either odd (or) Even



⇒ The message including the parity bit is transmitted & checked at the receiving end for Errors.

⇒ The circuit that generates the parity bit in the transmitted is called parity generator.

⇒ The circuit (ckt) that checks the parity bit in the receiver is called Parity checker.

Ex:- 3-bit message to be transmitted together with an Even parity:

⇒ Even-Parity Generator

Decimal digit	3-bit message (Input) x y z	Parity bit (output) P
0	0 0 0	0
1	0 0 1	1
2	0 1 0	1
3	0 1 1	0
4	1 0 0	1
5	1 0 1	0
6	1 1 0	0
7	1 1 1	1

⇒ Here it is the odd function.

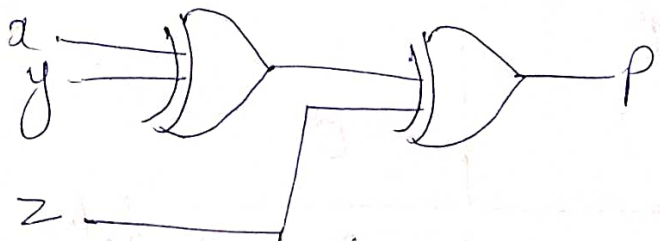
⇒ If there are odd numbers of ones in input then output is '1'.

⇒ If Even number of ones in input then output is '0'.

⇒ If we simplify using k-map for P, then we get

$$P = x \oplus y \oplus z$$

⇒ Circuit diagram is :-



*Even parity checker :-

Decimal digit	Four bits Received. x y z P (input)	Parity Error check C (output)
0	0 0 0 0	0
1	0 0 1 1	
3	0 1 0 1	
4	0 1 1 0	
4	1 0 0 1	
5	1 0 1 0	
6	1 1 0 0	
7	1 1 1 1	

Even parity checker:-

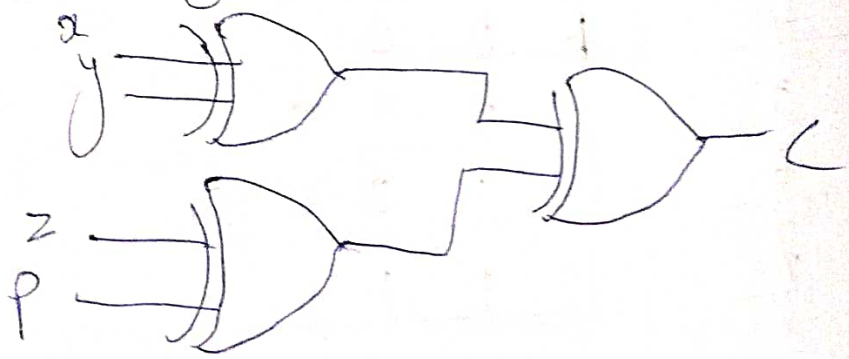
Four bits received				parity Error check
x	y	z	P	C
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

if Even parity, C = 1 → Received data is Error

C = 0 → Received data is correct

$$C = x \oplus y \oplus z \oplus P$$

Circuit diagram using gates:-

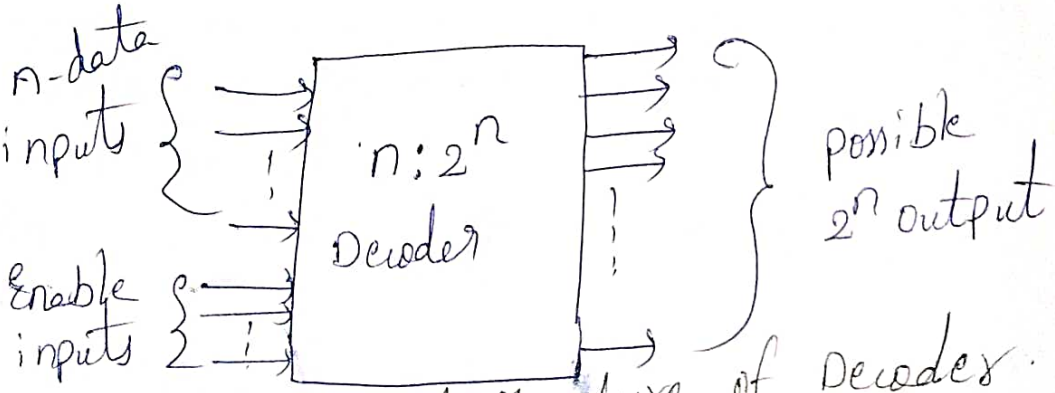


* Decoder:

→ Binary Decoder:- A decoder is a combinational circuit that converts binary information from 'n' input lines to a maximum of 2^n unique output lines.

(or)

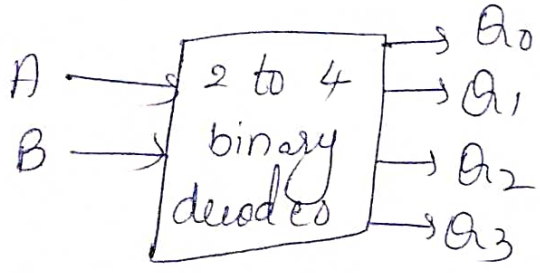
A decoder is a multiple input, multiple output logic circuit which converts code input to coded outputs, where input & output codes are different. The input code generally has less fewer bits than the output code.



* General Structure of Decoder.

Ex:- 2 to 4 Binary decoder:-

step-1:- Block diagram:



step-2: Truth table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

here 1 means active

0 means inactive

step-3:

If we simplify truth table using k-map then, we get boolean Expressions as follows,

$$Q_0 = \bar{A}\bar{B},\ Q_1 = \bar{A}B,\ Q_2 = A\bar{B}$$

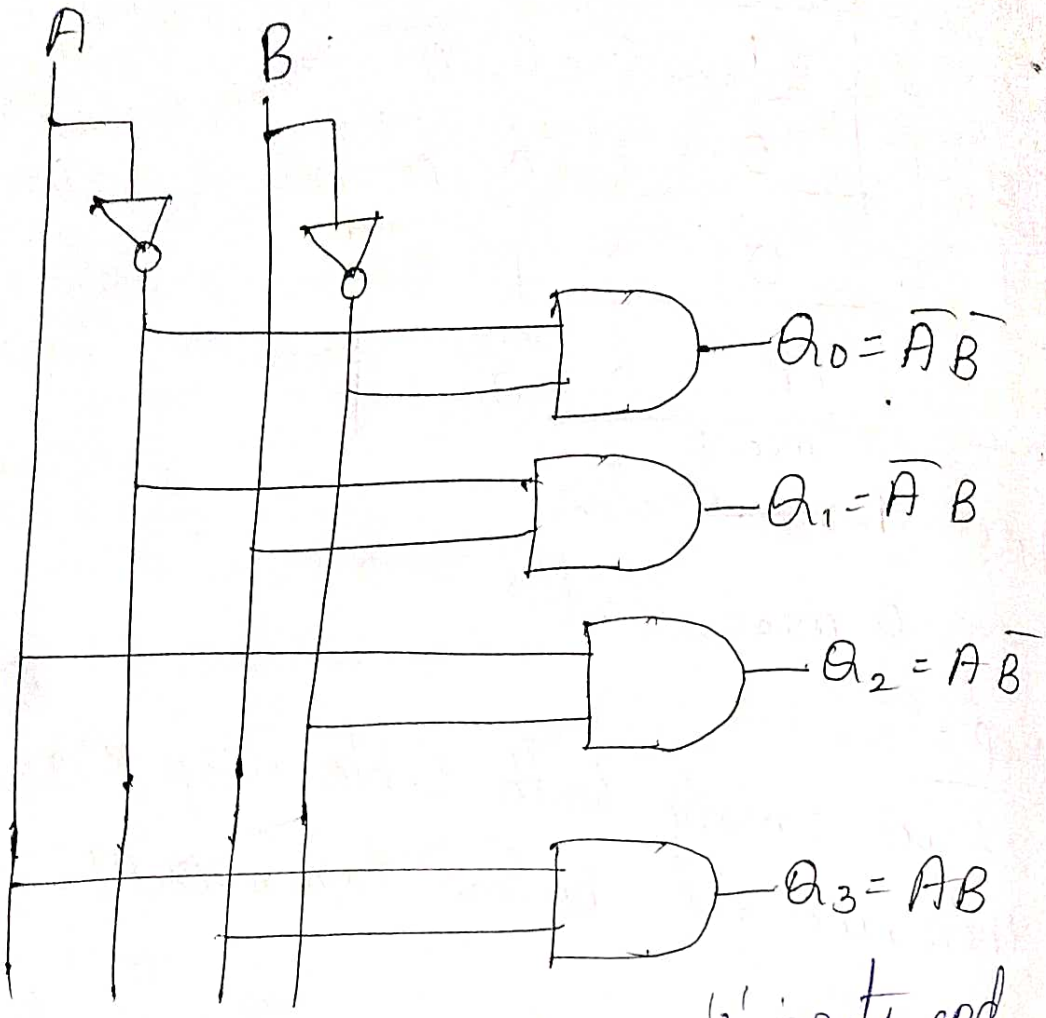
$$Q_3 = AB$$

⇒ Decoder will generates Minterms,

⇒ Using AND gates & NAND we can design decoder circuit

~~Circuit~~:

Step-4: Circuit diagram.



⇒ 3 to 8 decodes having '3' inputs and 1 Enable inputs and 8 outputs.

⇒ For 3 to 8, IC number is

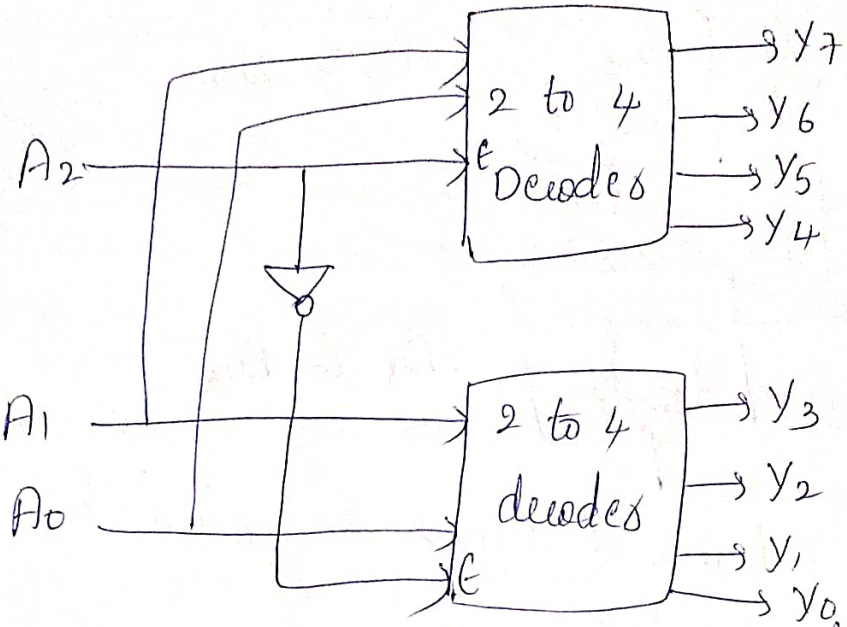
74LS138 / 74X138.

* Implement 4x16 decodes using '2' 3 to 8 decodes:

⇒ 4x16 decodes can be implement using '2' 3x8 decodes.

* Block diagram of 3 to 8 decoder:-

⇒ We required two 2 to 4 decoders for implementing one 3 to 8 decoder.



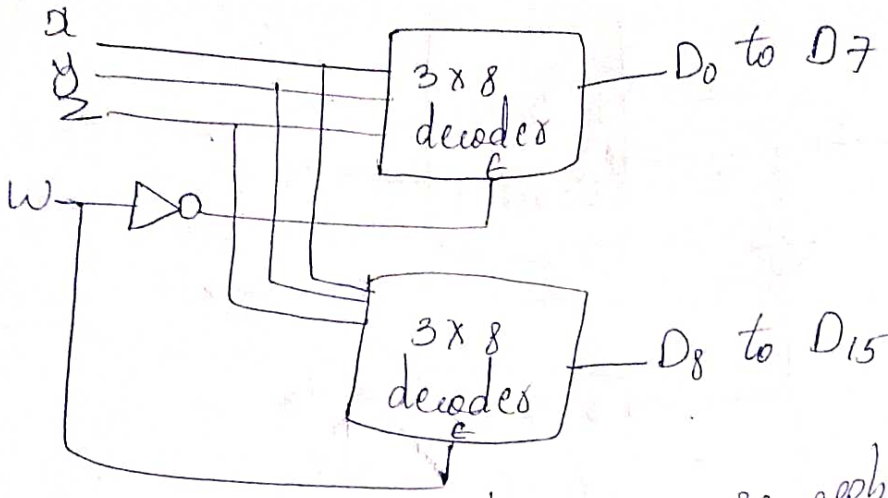
⇒ The parallel inputs A1 & A0 are applied to each 2 to 4 decoder. The complement of input A2 is connected to Enable, E of lower 2 to 4 decoder in order to get the outputs, Y3 to Y0.

* These are the lower four min terms.

⇒ The input, A2 is directly connected to Enable, E of upper 2 to 4 decoder in order to get the outputs, Y7 to Y4. These are the higher four min terms.

* Block diagram of 4 to 16 decoder:-

⇒ we require two 3 to 8 decoders for implementing one 4 to 16 decoder.



⇒ The parallel inputs x, y, z are applied to Each 3 to 8 decoder.

⇒ The complement of input w is connected to Enable, E of upper 3 to 8 decoder in order to get the outputs, D0 to D7. These are lower eight min terms.

⇒ The input, w is directly connected to Enable, E of lower 3 to 8 decoder in order to get the outputs D8 to D15. These are the higher eight min terms.

Ex:- if the values be like

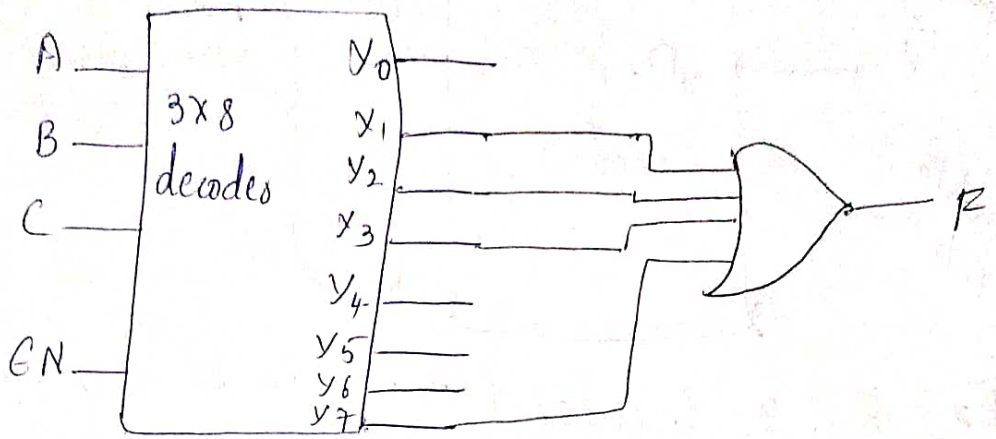
w z y x

0 0 0 1 ⇒ D1 = 1 and remaining outputs will be zero

1 1 1 1 ⇒ D15 = 1 and remaining outputs will be zero

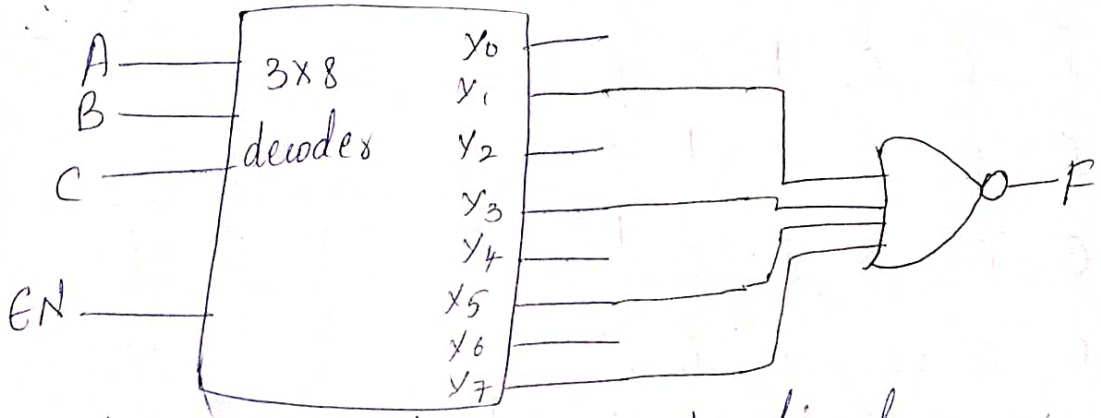
* Implement $F = \Sigma(1,2,3,7)$ using 3x8 decoder.

(A)

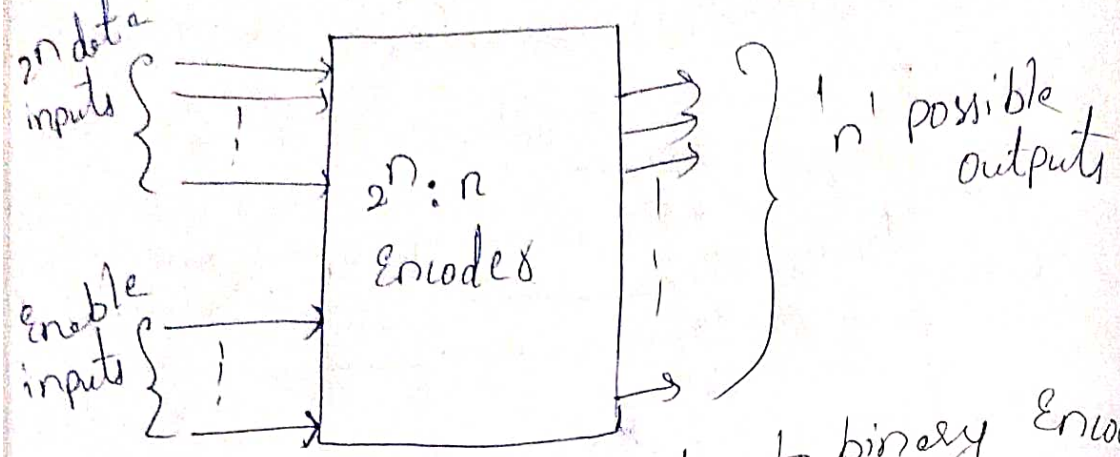


* Implement $= \pi M(1,3,5,7)$ using 3x8 decoder.

⇒ 1,3,5,7 are min terms. In order to get them as max terms, we need to perform complement for that minterms.



* Encoders :- An Encoder is a combinational circuit that performs the inverse operation of a decoder. An Encoder has 2^n input lines & 'n' output lines.



Ex: 8 : 3 Encoder (octal to binary Encoder)

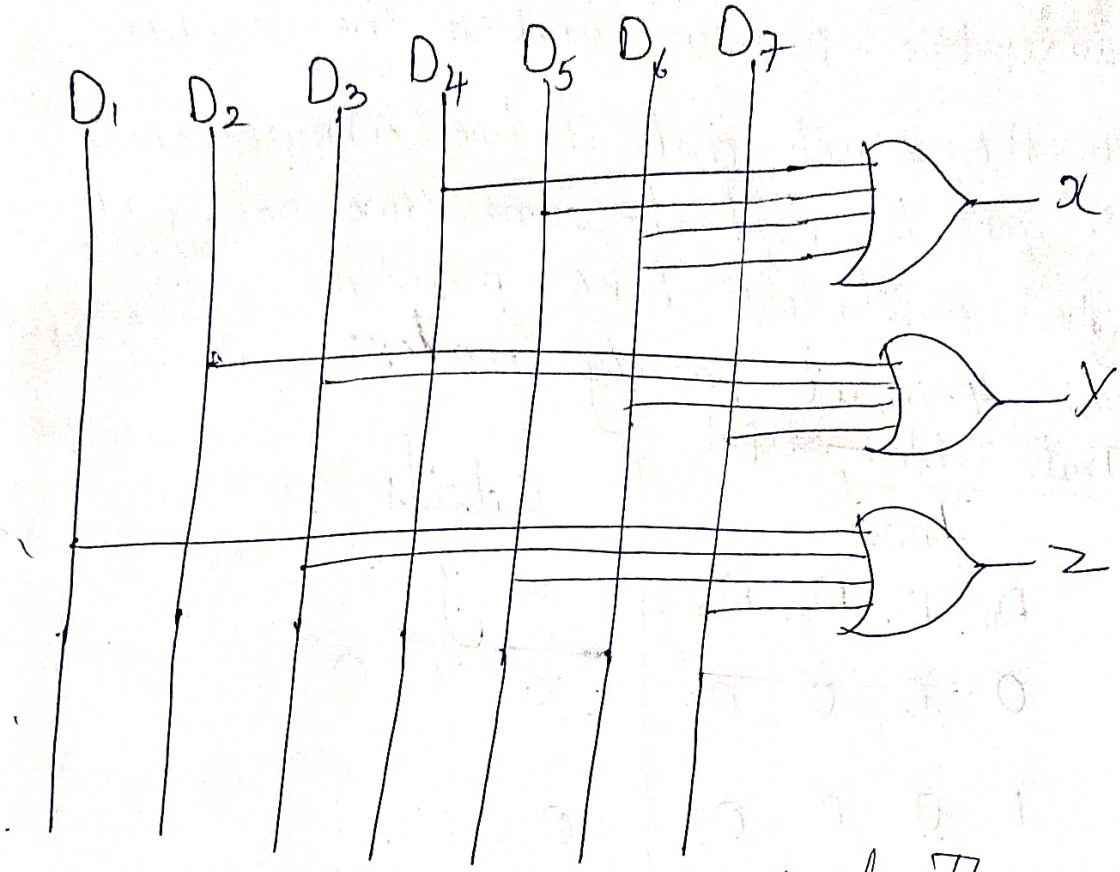
Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$x = D_4 + D_5 + D_6 + D_7$

$y = D_2 + D_3 + D_6 + D_7$

$z = D_1 + D_3 + D_5 + D_7$

Circuit Diagram :-



* 8:3 Encoders have some drawbacks. They are :-

(1) There is an ambiguity, when all outputs of Encoders are Equal to zero.

(2) If more than one input is active high, then the Encoders produces an output, which may not be the correct code.

So, to overcome these difficulties, we should assign "priorities" to Each input of the Encoder.

Priority Encoder (P.E):- A priority Encoder is an Encoder circuit that includes the priority function. The operation of P.E. is such that "if two (or) more inputs are equal to '1' at the same time, having the highest priority will take precedence.

Ex:- 4-input priority Encoder:-

(A) Truth Table ~~step-1~~

Input				Outputs	
D_0	D_1	D_2	D_3	x	y	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

⇒ Here invalid ~~by~~ inputs are recognized by 'V' output

⇒ if V=0 then it is invalid input.

V=1 → it is valid input.

⇒ In the above truth table 'X' refers to don't care term.

Ex:- if the input values are

D_0 D_1 D_2 D_3

X 1 0 0. Here D_0 is consists don't care terms. X values may be

Either there '0' or '1'. and D_1 values is ~~also~~ '1'. If 'D_0' value is 0 then D_1 priority will given to D_1. If 'D_0' value is also 1, then based on precedence higher priority will given to D_1 only. So output will be '1'. same thing will ~~be~~ apply to all inputs.

* Simplification using k-map:-

⇒ Simplification using k-map for ~~A~~. x

⇒ Let us consider the outputs of x.

don't care term

i) $x = X$ when ~~D~~$_0$ D_1 D_2 D_3

0 0 0 0

∴ For '0' result will be 'X'.

~~1 2 4 8~~ → binary values

ii) $x = 0$ when D_0 D_1 D_2 D_3

1 0 0 0

∴ For '1' result will be 0 because $x = 0$ are not considering that's why we ~~are~~ not considering them.

1 2 4 8 → binary values.

$x = 0$ when D_0 D_1 D_2 D_4

X 1 0 0

here D_2 & D_4 are zero's. So consider D_0 & D_1 only. D_1 is ~~*~~ '1' already.

And D_0 value will either '0' or '1'

So we have to consider the values by placing 0 & 1 in place of X.

Then the outcomes are:

	1 D_0	2 D_1	4 D_2	8 D_3	
(when X = 0)	0	1	0	0	= 2
(when X = 1)	1	1	0	0	= 3

So, for '2 & 3' value is '0' because

$x = 0$.

(iii) ~~x~~

$x = 1$, when

1 D_0	2 D_1	4 D_2	8 D_3
X	X	1	0

8 → binary value

Replace 'X' with both '0' & '1'.

The possible outcomes will be

1 D_0	2 D_1	4 D_2	8 D_3	
1	0	1	0	= 5
1	1	1	0	= 7
0	1	1	0	= 6
0	0	1	0	= 4

for these values, the value is '1' in k-map because $x = 1$. so we have to consider them

$x = 1$, when

1 D_0	2 D_1	4 D_2	8 D_3
X	X	X	1

replace 'x' with both 0 & 1.

possible outcomes are,

D_0	D_1	D_2	D_3	
0	0	0	1	⇒ 8
1	0	0	1	⇒ 9
0	1	0	1	⇒ 10
1	1	0	1	⇒ 11
0	0	1	1	⇒ 12
1	0	1	1	⇒ 13
0	1	1	1	⇒ 14
1	1	1	1	⇒ 15

value is '1' for all these values in k-map, because $x = 1$.

∴ k-map for x:-

Based on previous calculation results, we are going to construct the k-map.

D_3D_2 \ D_1D_0	00	01	11	10
00	0 X	1 0	3 0	2 0
01	4 1	5 1	7 1	6 1
11	12 1	13 1	15 1	14 1
10	8 1	9 1	11 1	10 1

Group-1:-

	00	01	11	10
01	4 1	5 1	7 1	6 1
11	8 1	9 1	11 1	10 1

⇒ D_2

Group-2:-

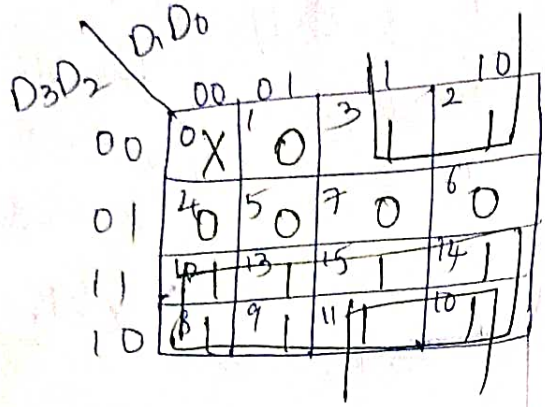
	00	01	11	10
11	12 1	13 1	15 1	14 1
10	8 1	9 1	11 1	10 1

⇒ D_3

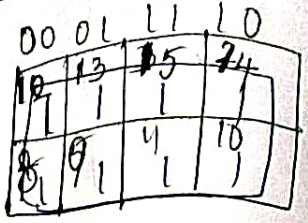
∴ $x = D_2 + D_3$

* k-map for y.

(i) ⇒ Follow the same process which is applied while calculating for 'x'; After calculation, k-map will became like following:-

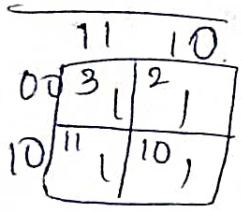


Group-1



⇒ D_3

Group-2:

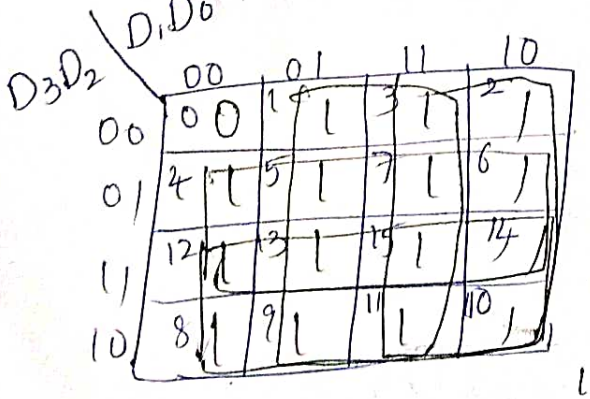


⇒ D_1D_2'

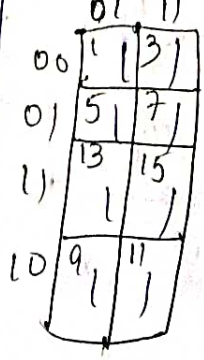
⇒ $y = D_3 + D_1D_2'$

* k-map for v.

⇒ Follow the same process and find values and construct k-map. we get k-map as,



Group-1



⇒ D_0

Group-2

⇒ D_1

Group-3:

	00	01	11	10
01	4 1	5 1	7 1	6 1
11	12 1	13 1	15 1	14 1

$\Rightarrow D_2$

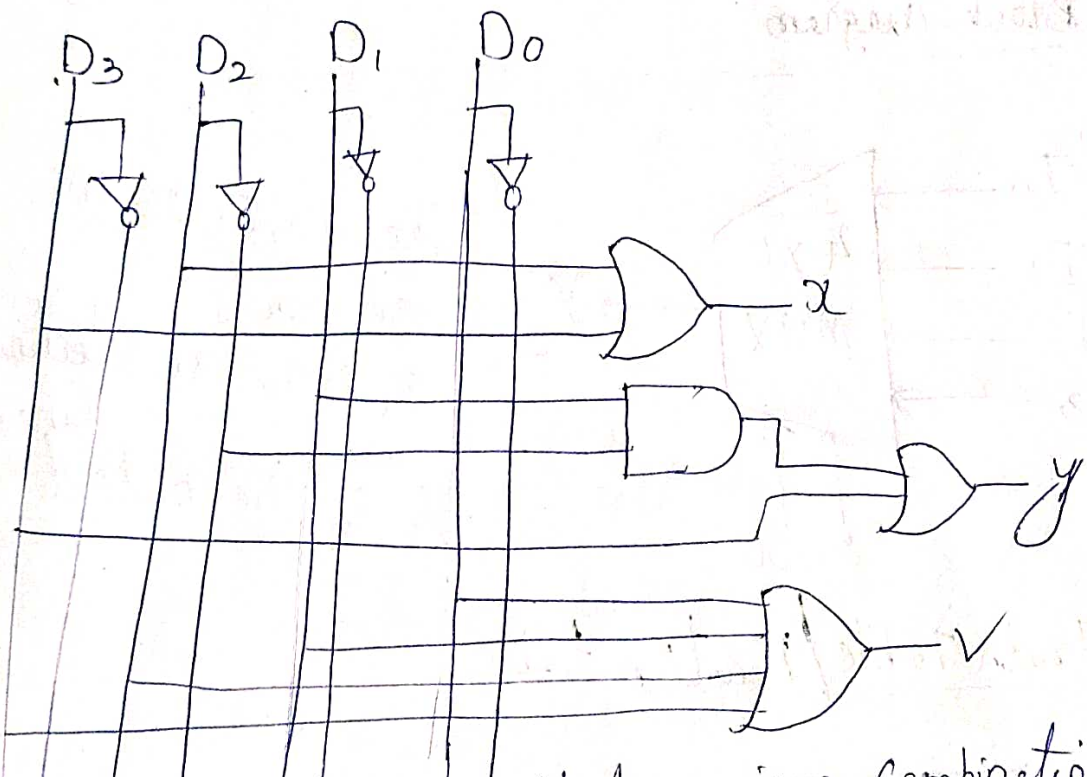
Group-4:

	00	01	11	10
01	12 1	13 1	15 1	14 1
10	8 1	9 1	11 1	10 1

$\Rightarrow D_3$

$\therefore V = D_0 + D_1 + D_2 + D_3$

Logic diagram using above Equations.



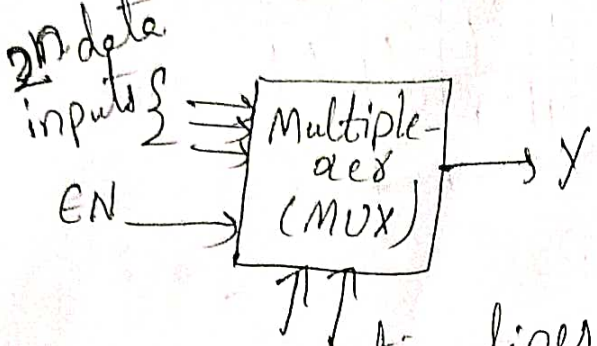
* Multiplexers: A multiplexer is a combinational circuit that selects binary information from "one of many inputs (lines) and ~~directs~~ directs it to a single output (line)."

⇒ The selection of particular input lines is controlled by a set of selection lines

⇒ Generally for a n-selection input lines

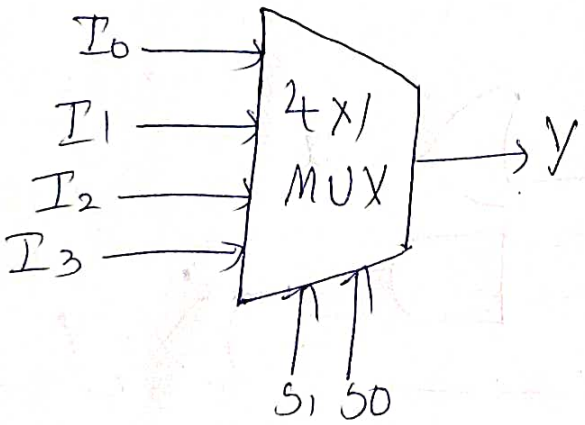
there are 2^n input lines.

Block diagram:



Ex:- 4-to-1 line Multiplexer (MUX)

Block diagram:

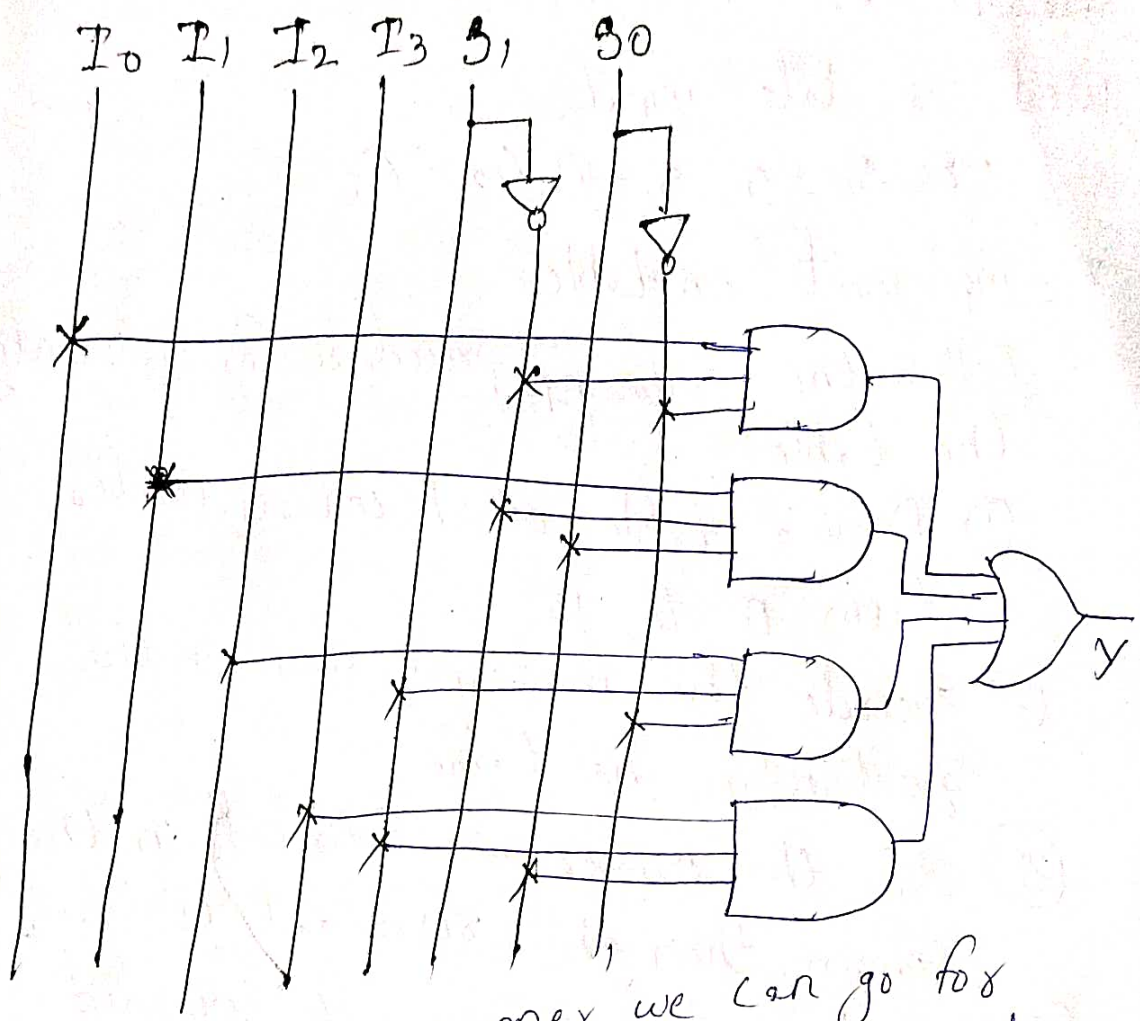


here I_0, I_1, I_2, I_3 are inputs.
S_1 & S_0 are selection lines
y is the output.

Truth Table / Function table

S_1	S_0	y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Logic diagram:-



2, I_3

selection lines

tput

* In the same manner we can go for 8x1 MUX which will have 8 data inputs, '3' selection lines/inputs & '1' output

* Implement the following Boolean functions using 8x1 MUX.

$$F(P, Q, R, S) = \sum m(0, 1, 3, 4, 8, 9, 15)$$

(A) Given,

$$F(P, Q, R, S) = \sum m(0, 1, 3, 4, 8, 9, 15)$$

⇒ we know that there are 3 selection lines in 8x1 MUX. i.e, $2^3 = 8$ (3) → selection lines.

S_0, in given P, Q, R, let us consider selection lies as Q, R, S and P can be used for data input.

i.e, $S_0 = S$, $S_1 = R$, $S_2 = Q$

* Implementation table:-

Follow the following procedure for implementing the table.

(1) Draw a table which consists numbers from 0 to 15.

(2) Encircle the minterms given in the question in the table.

(3) If the numbers are encircled in the column, then the resultant is '1'.

(4) If both numbers are not encircled then the resultant is '0'.

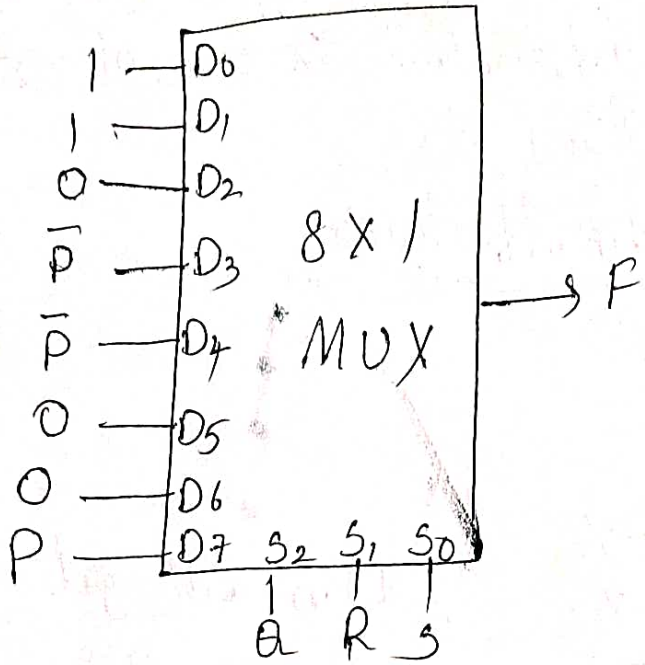
(5) If only one value is encircled, then the resultant will be its respective data input in that row i.e. $\overline{P}$ & P.

Table:-

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{P}$	(0)	(1)	2	(3)	(4)	5	6	7
P	(8)	(9)	10	11	12	13	14	(15)
	1	1	0	$\overline{P}$	$\overline{P}$	0	0	P

⇒ resultants

Block diagram:



Here Based on the values of Q, R, S output will come.

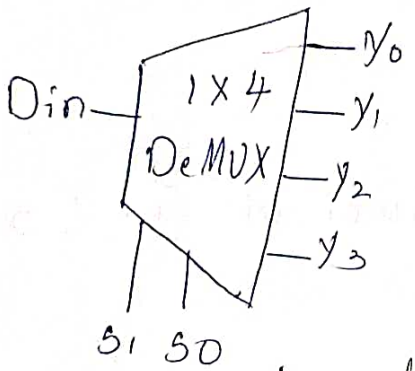
Ex:

Q	R	S	F(output)
0	0	0	$1 \Rightarrow D_0$
0	0	1	$1 \Rightarrow D_1$
0	1	0	$0 \Rightarrow D_2$
0	1	1	$\bar{P} \Rightarrow D_3$
1	0	0	$\bar{P} \Rightarrow D_4$
1	0	1	$0 \Rightarrow D_5$
1	1	0	$0 \Rightarrow D_6$
1	1	1	$P \Rightarrow D_7$

Demultiplexer: A demultiplexer is a combinational circuit that receives information on a single line and transmits this information to one of 2^n possible output lines.

Ex: 1 to 4 demultiplexer.

⇒ **Block Diagram:**



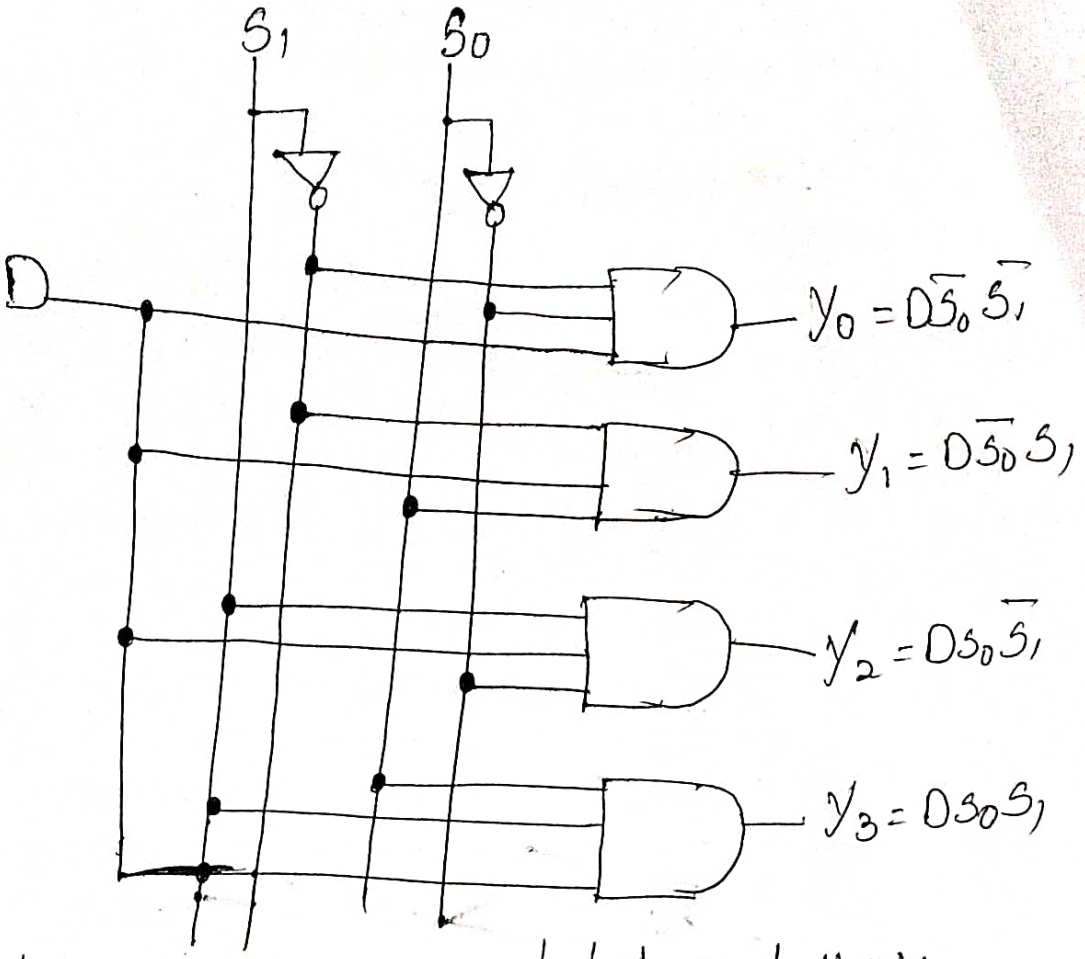
, Din will get from MUX. Here Din the output of MUX.

Truth table / Function table:

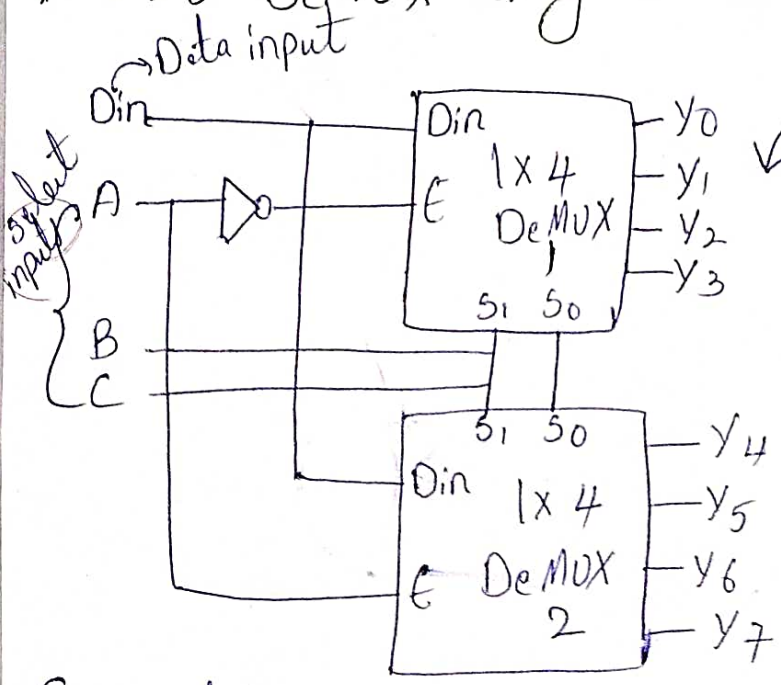
Inputs			Outputs			
Din	S_1	S_0	y_0	y_1	y_2	y_3
X	X	X	0	0	0	0
X	0	0	X	0	0	0
X	0	1	0	X	0	0
X	1	0	0	0	X	0
X	1	1	0	0	0	X

(Note:- Here we don't know the value of Din, because it will get from MUX. That's why we marked it as 'X' in both outputs and in Din.)

Logic diagram :-



* 1 X 8 DeMUX using '2' 1X4 deMUX



Cascading of DeMux (similar to Cascading of decoders) (DeMultiplexer tree)

C. In electronics, Cascading refers to a process where large multiplexers can be designed and implemented using smaller multiplexers)