UNIT - I

Number System and Codes

NUMBER SYSTEMS

Decimal → A decimal code consits of 10 numbers (from 0-9). Each no is represented using these 10 digits.

A decimal no represented as

$(132)_{10} = (1 \times 10^2) + (3 \times 10) + (2 \times 10^0) = 132$

⇒ Numbers used are in decimal code.

Binary → A binary code consists of 2 no. (0 and 1). Each no. is represented using these 2 digits.

A binary number can be represented as

$(10101)_2$

$(10101)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 16 + 4 + 1 = 21$

Decimal to Binary Conversion.

1. Repeatedly divide the no by 2 and write the rem on the right.
2. Start writing the no from bottom to the top including the last quotient.

Example

example

2 | 21 → remainders
2 | 10 → 1
2 | 5 → 0
2 | 2 → 1
2 | 1 → 0
last quo.

∴ binary no is

$(21)_{10} = (10101)_2$

last quo. → remainders in reverse order.

Octal → An octal code consists of 8 numbers (0 to 7) which are used to represent another no in decimal code.

An octal no can be represented as

$(327)_8 = 3\times8^2 + 2\times8^1 + 7\times8^0$
$= 192 + 16 + 7$
$= 215$

Conversion ito Octal from ~~Binary~~ Decimal

Divide by 8 instead of 2.

8 | 215
8 | 26 → 7
3 → 2

∴ octal no is

$(215)_{10} = (327)_8$

Hexadecimal → A hexadecimal code consists of 16 numbers (0 to 15) 0 to 9 are represented by number digits whereas 10 to 15 are represented by alphabets.

0 → 0
1 → 1
2 → 2
3 → 3
4 → 4
5 → 5
6 → 6
7 → 7
8 → 8
9 → 9
10 → A
11 → B
12 → C
13 → D
14 → E
15 → F

A hexadecimal code can be represented as.

$(BA3) = 11 \times 16^2 + 10 \times 16^1 + 3 \times 16^0$
$= 2816 + 160 + 3$
$= 2979$

Decimal ~~Binary~~ Conversion form.

Divide by 16 and follow the same steps.

16 | 2979 → 3
16 | 186 → 10
11

Replace 11 and 10 by their alphabet codes.

11 → B, 10 → A

$\therefore (2979)_{10} = (BA3)_{16}$

Conversion from Binary to Hexa

0101 0101
5 5

$(55)_{16} = (01010101)_2$

1. Make pairs of 4.
2. Convert them to decimal
3. Write the corresponding no as hexa decimal codes.

For Hexa to Binary, Follow the reverse

$(55)_{16} \Rightarrow (01010101)_2$

(0101) (0101)

* For Binary to Octa Make pairs of 3.

Example

$(\underset{5}{0}\underset{2}{1}\underset{5}{0}\underset{2}{1}\underset{5}{0}\underset{2}{1})_2 = (525)_8$

For (Octa to Hexa) or (Hexa to Octa), Direct Conversion is not possible.

∴ either Octa → Binary → Hexa
or Octa → Decimal → Hexa
Is followed.

example → $(BA3)_{16} = (?)_8$

$(BA3)_{16} = (2979)_{10}$ (solved earlier)

8 | 2979		
8 | 372	3	$= (5643)_8$
8 | 46	4	
5	6	

$\therefore (BA3)_{16} = (5643)_8$

CODES

BCD code → BCD stands for Binary Coded Decimal.

It is a 4 bit code and each bit is weighed 8, 4, 2, 1 resp.

B_4 B_3 B_2 B_1

8 4 2 1

Decimal value of $B_4B_3B_2B_1$ is $(B_4 \times 8 + B_3 \times 4 + B_2 \times 2 + B_1 \times 1)$

The code can be written as.

Decimal	BCD Code.
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

8	1000
9	1001

Excess 3-Code → An Excess 3-Code has no particular weigh assigned. It is derived by adding 3 to the BCD code.

Decimal	Excess-3	
0	0011	(0+3=3)
1	0100	(1+3=4)
2	0101	(2+3=5)
3	0110	(3+3=6)
4	0111	(4+3=7)
5	1000	(5+3=8)
6	1001	(6+3=9)
7	1010	(7+3=10)
8	1011	(8+3=11)
9	1100	(9+3=12)

Difference in Binary conversion and Binary Codes.

During conversion the no is directly converted to binary

$(21)_{10} = (10101)_2$ is represent by 5 bits in binary conversion. whereas if we use a code 2 is converted separately and 1 is done separetly resulting in an 8 bit code.

BCD

21 → $(0010\ 0001)_{BCD}$

0010 0001

Excess-3

21 → $(01010100)_{excess-3}$

0101 0100

Gray Code → A gray code changes by only one bit from its preceding number. Due to this, its is preferd in cases where analog data is converted into descrete data.

Gray	Decimal
0 0 0 0	0
0 0 0 1	1
0 0 1 1	2
0 0 1 0	3
0 1 1 0	4
0 1 1 1	5
0 1 0 1	6
0 1 0 0	7
1 1 0 0	8
1 1 0 1	9
1 1 1 1	10
1 1 1 0	11
1 0 1 0	12
1 0 1 1	13
1 0 0 1	14
1 0 0 0	15

Sort of mirror image after every 2^n being the no of bit

ASCII Code → ASCII code is an alphanumeric code which represents upper/lower case letters, numbers and special characters using a 7-bit code.

ASCII → American Standard Code for Information Interchange.

EBCDIC → EBCDIC is an alphanumeric code which represents upper/lower case letters, numbers and special characters using a 8-bit code.

EBCDIC → Extended Binary Coded Decimal (BCD) Interchange Code.

Switching Theory

Postulates of Boolean Algebra (Huntington's Postulates)

① Closure → The end result after any operation belongs to the same set as the operands.

$a * b = c$

if $a \& b \in B$ then $c \in B$.

here $B = \{0, 1\}$

and $*$ is any binary operation

② **Identity** → There exists an element which when operated with x, such that $x \in B$, gives x.

$$x \cdot 1 = x \quad \text{and} \quad x + 0 = x$$

③ **Distributive** →

$$x \cdot (y+z) = x \cdot y + x \cdot z$$
$$x + yz = (x+y)(x+z)$$

④ **Commutative** →

$$x \cdot y = y \cdot x$$
$$x + y = y + x$$

⑤ **Inverse** → For every element x there exists an element x' (or $\bar{x}$) such that $x * x' = e$ where e is the identity element.

$$x \cdot \bar{x} = 0 \qquad x + \bar{x} = 1.$$

⑥ There exist atleast 2 elements x & y such that $x, y \in B$ and $x \neq y$.

Theorems of Boolean Algebra

I
$$x \cdot x = x$$
$$x + x = x$$

II
$$x \cdot 0 = 0$$
$$x + 1 = 1$$

III Associative

$(xy)z = x(yz)$

$(x+y)+z = x+(y+z)$

IV Involution

$(x')' = x$

V Demorgan's Theorem

$(A+B)' = A'B'$

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

Proof

Let $(A+B) = z$

$z' = (A+B)'$

$z \cdot z' = 0$ and $z + z' = 1$

If $(A+B) \cdot A'B' = 0$ and $(A+B) + A'B' = 1$

Then, $A'B' = (A+B)'$

$(A+B) \cdot A'B' = AA'B' + BA'B'$

$= (AA')B' + A'(BB')$ (associative)

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

$= 0 + 0$

$= 0$

$(A+B) + A'B' =$

$= (A+B)(1) + A'B'$ $[x \cdot 1 = x]$

$= (A+B)(A'+A) + A'B'$ $[A'+A = 1]$

$= A'A + A + A'B + A'B'$

$= 0 + A + A'(B+B')$ $[x \cdot x' = 0]$

$= A + A'(1)$ $[B+B' = 1]$

$= A + A' = 1$

Hence, $(A+B)' = A'B'$

VI Absorption

$X + XY = X$

$X(X+Y) = X$

VII $X + \bar{X}Y = X + Y$

Switching Functions

A switching function (boolean function) is formed by boolean variables, parantheses and boolean operators AND, OR and NOT.

A Boolean function can be represented in an algebric form and can also be represented using a truth table.

A truth table consists of all the possible inputs/outputs in a function

example → $F = XY + Z'$

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

all possible inputs (can only have values 0 or 1)

Each variable of a Boolean Function is known as a literal.

Switching Functions

Canonical Form

Every term of a canonical form contains all the literals. It is the form which is derived directly from a truth table.

Standard Form

This form consists of the simplified expression and is used to implements circuits of the switching functions.

Canonical Form

Max terms → All the possible outcomes when the the literals are connected using an OR gate are known as max terms or standard Sums.

Min terms → All the possible outcomes when the literals are connected using an AND gate are known as minterms or Standard products.

	X	Y	Z	MAXTERMS	MINTERMS
0	0	0	0	$M_0 = X+Y+Z$	$m_0 = X'Y'Z'$
1	0	0	1	$M_1 = X+Y+Z'$	$m_1 = X'Y'Z$
2	0	1	0	$M_2 = X+Y'+Z$	$m_2 = X'YZ'$
3	0	1	1	$M_3 = X+Y'+Z'$	$m_3 = X'YZ$
4	1	0	0	$M_4 = X'+Y+Z$	$m_4 = XY'Z'$
5	1	0	1	$M_5 = X'+Y+Z'$	$m_5 = XY'Z$
6	1	1	0	$M_6 = X'+Y'+Z$	$m_6 = XYZ'$
7	1	1	1	$M_7 = X'+Y'+Z'$	$m_7 = XYZ$

* To write maxterms
put X if X=0 and X' if X=1
and use '+' b/w literals.

* To write minterms
put X' if X=0 and X if X=1
and use '.' b/w literals.

Canonical POS → Product of maxterms

$$F = \Pi(0, 1, 2, 9, 13)$$

$$= M_0 \cdot M_1 \cdot M_2 \cdot M_9 \cdot M_{13}$$

To find Can. POS from Truth table, write all the maxterms whose output is 0.

Canonical SOP → Sum of minterms

$$F = \Sigma(1, 3, 9, 11) = m_1 + m_3 + m_9 + m_{11}$$

To find Can. SOP write all the minterms with output 1.

example

	X	Y	Z
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	0

$$Z = m_0 + m_2 = X'Y' + XY'$$

$$Z = M_1 \cdot M_3 = (X+Y')(X'+Y')$$

$$Z = \Sigma(0,2) = \pi(1,3)$$

SOP and POS forms are complimentary to each other i.e. no present in one are absent in another.

Standard Form

Standard Form can be generated from Cano. form by simplification.
There are 3 methods of simplification
(1) Algebric (long method but still used in some cases).
(2) K-Maps (used in most ques)
(3) Quine Mc-Clusky. (4 mark Ques for sure)

Algebric → Simplification using Theorems.

example

$$F = zxy + x'zy' + x'zy + z'xy$$

$$= xy(z+z') + zx'(y'+y)$$

$$= xy + zx' \quad [x+x'=1]$$

* Simplification can be done from non-canonical forms to standard forms as well.

example

$$F = xyz + x'y + xyz'$$

$$= xy(z+z') + x'y \quad [x+x'=1]$$

$$= xy \cdot (1) + x'y$$

$$= xy + x'y = y(x+x') \quad [x+x'=1]$$

$$= y$$

K-Maps

A K-Map represents the function in a combination of squares where each square represents a maxterm/minterm.

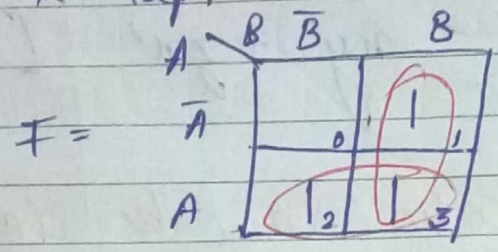
SOPs using K-Maps

no of sq $= 2^n$ where n is no of variables.

2 Variable

$$F(A, B) = \Sigma(1, 2, 3)$$

Step I Write the variables in a K-Map.



Step II Put 1 in places of the minterms specified in the ques.

Step III Make Pairs, Quads or octants possible. No, no should be left behind. And Octants are first preference followed by Quads followed by pairs.

In the above ques
Pairs (1, 3) and (2, 3)
are possible.

Step IV For the ans. consider the literals which are similar for a pair/quad/octant and write them with a '+' sign.

(1, 3) have B ✓ in common
(2, 3) have Ā in common

$\therefore \quad F = B + A$

example

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

$F =$

	$\bar{y}$	y
$\bar{x}$	1 (0)	1
x	1 (2)	1 (3)

$F = \bar{y} + x$

3-variable

$F(A, B, C) = \Sigma(0, 3, 4, 6, 7)$

Rest rules are similar just keep in mind the pattern of K-map

We don't follow 1, 2, 3, 4 rule. we follow 1, 2, 4, 3. i.e. the 3rd and 4th columns are interchanged.

$F =$

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

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

ex $F = \Sigma(0, 1, 2, 4, 6)$

Quads/Pairs/Octants can be formed at corners as shown

$F =$

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

Quad (0, 2, 4, 6)
Pair (0, 1)

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

4-Variable

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

Octant (0, 1, 4, 5, 8, 9, 12, 13)

Quad (0, 2, 4, 6)

Quad (4, 6, 12, 14)

$F =$

	$\bar{Y}\bar{Z}$	$\bar{Y}Z$	YZ	$Y\bar{Z}$
$\bar{W}\bar{X}$	1 0	1 1	3	1 2
$\bar{W}X$	1 4	1 5	7	1 6
WX	1 12	1 13	15	1 14
$W\bar{X}$	8	9	11	10

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

A quad/octant/pair formed should have atleast one-element which is not a part of any other quad/pair/octant.

5-Variable

Draw two 4 variable K-Maps side by side and assume them to be overlapping while considering pairs/quad/octants.

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

$F =$

	$\bar{D}\bar{E}$	$\bar{D}E$	DE	$D\bar{E}$		$\bar{D}\bar{E}$	$\bar{D}E$	DE	$D\bar{E}$
$\bar{B}\bar{C}$	1 0	1	3	1 2	$\bar{B}\bar{C}$	16	1 17	19	18
$\bar{B}C$	1 4	5	7	1 6	$\bar{B}C$	20	1 21	23	22
BC	12	1 13	1 15	14	BC	28	1 29	1 31	30
$B\bar{C}$	8	1 9	1 11	10	$B\bar{C}$	24	1 25	1 27	26

Oct (9, 11, 13, 15, 25, 27, 29, 31)

Quad (0, 2, 4, 6)

Quad (17, 21, 25, 29)

$\bar{A}$ A

$F = BE + \bar{B}\bar{E}\bar{A} + A\bar{D}E$

Calculating POS using K-maps

① Replace F by F' and plot 0s instead of 1's
② Take complement of the ans.

Example

$$F = \pi(0, 1, 2, 3, 4, 6, 12)$$
(A, B, C, D)

$F' = $	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	5	7	0
AB	0	13	15	14
$A\bar{B}$	8	9	11	10

Quad (0, 1, 2, 3) ; (0, 2, 4, 6)
Pair (4, 12)

$$F' = \bar{A}\bar{B} + \bar{A}\bar{D} + B\bar{C}\bar{D}$$

$$F = (\bar{A}\bar{B} + \bar{A}\bar{D} + B\bar{C}\bar{D})'$$

$$= (\bar{A}\bar{B})' \cdot (\bar{A}\bar{D})' \cdot (B\bar{C}\bar{D})' \quad [\text{De-morgan's}]$$

$F = (A+B)' \cdot (A+D) \cdot (\bar{B}+C+D)'$

(De morgans)

Don't Care Elements:

These are the elements which don't have a fixed output in a function. We don't consider this case as its value doesn't effect the output. It can either be 0 or 1.

They are represented by a X in K-map and are only considered if they help reducing the exp. or else are ignored.

example $F(w,y,x,y,z) = \Sigma(1,3,7,11,15) + d(0,2,5)$

	$\bar{y}\bar{z}$	$\bar{y}z$	yz	$y\bar{z}$
$\bar{w}\bar{x}$	X_0	1 (1)	1 (3)	X_2
$\bar{w}x$	4	X_5	1 (7)	6
$w x$	12	13	1 (15)	14
$w\bar{x}$	8	9	1 (11)	10

Quad (1, 3, 5, 7)
Quad (3, 7, 11, 15)
(only considered 5 ; 0, 2 were not occ.)

$F = yz + \bar{w}z$

Imp

Quine - Mc Cluskey

Can be used for simplification of func with more than 5 - variables.

* If the ques asks for Prime implicants / essential Prime implicants always use this method.

Explaination with example

$F(A,B,C,D) = \Sigma(1,3,5,7,9,10,11) + d\Sigma(12,13,14,15)$

Step 1 Write each no in decimal and binary form (include dont-care elements as well).

Decimal	Binary A B C D
1	0 0 0 1
3	0 0 1 1
5	0 1 0 1
7	0 1 1 1
9	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

Step 2 Make a table acc. to no of 1s.

no of 1s	minterms	ABCD
1	1	0001 ✓
2	3	0011 ✓
	5	0101 ✓
	9	1001 ✓
	10	1010 ✓
	12	1100 ✓
3	7	0111 ✓
	11	1011 ✓
	13	1101 ✓
	14	1110 ✓
4	15	1111 ✓

Step 3 Make Pairs by compairing two adjacent rows and replace unlike elements with a –.

no of 1s	minterm	ABCD
1	1-3	00-1 ✓
	1-5	0-01 ✓
	1-9	-001
2	3-7	0-11 ✓
	3-11	-011 ✓
	5-7	01-1 ✓
	5-13	-101 ✓
	9-11	10-1 ✓
	9-13	1-01 ✓
	10-11	101- ✓

	10-14	1-10 ✓
	12-13	110- ✓
	12-14	11-0 ✓
3	7-15	-111 ✓
	11-15	1-11 ✓
	13-15	11-1 ✓
	14-15	111- ✓

Step 4 Compare adjacent rows and make pairs of 4.

no of 1's	minterms	A B C D
1	1-3-5-7	0 - -1] ✓
	1-5-3-7	0 - -1]
	1-3-9-11	-0-1] ✓
	1-9-3-11	-0-1]
	1-5-9-13	--01] ✓
	1-9-5-13	--01]
2.	3-7-11-15	--11] ✓
	3-11-7-15	--11]
	5-7-13-15	-1-1] ✓
	5-13-7-15	-1-1]
	9-11-13-15	1--1] ✓
	9-13-11-15	1--1]
	10-11-14-15	1-1-]
	10-14-11-15	1-1-]
	12-13-14-15	11--]
	12-14-13-15	11--]

* If we get similar pairs, our ans is correct so far.

Step 5: Make further Pairs.

no of 1s.	minterms	A B C D
1	1-3-5-7-9-11-13-15	- - - 1
	1-3-9-11-5-7-13-15	- - - 1
	1-5-9-13-3-7-11-15	- - - 1

Step 6: Note down all the Pairs whose further pairing wasn't possible.

minterms	A B C D	Term
10, 11, 14, 15	1 - 1 -	AC
12, 13, 14, 15	1 1 - -	AB
1, 3, 5, 7, 9, 11, 13, 15	- - - 1	D

These are PIs (Prime Implicants)

Replace 1 by literal and 0 by its compliment

Step 7: Make the PI table.

(don't write don't care elements)

PI	minterms	1	3	5	7	9	10	11
AC	10, 11, 14, 15						X	X
AB	12, 13, 14, 15							
D	1, 3, 5, 7, 9, 11, 13, 15	X	X	X	X	X		X

① Put a cross against the min. present.

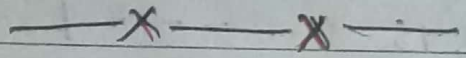
② Circle the X's which are one in the column

(8) Every PI with a circled x is an essential PI

(9) Write F as sum of essential PIs.

$F = AC + D$.

AC and D cover all the minterms, incase they don't, we include a PI in the ans. which consists of the remaining minterms

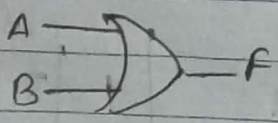


Logic Gates

AND

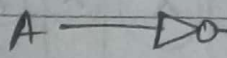
$F = A \cdot B$

OR



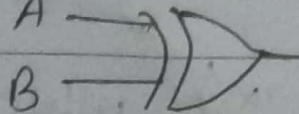
$F = A + B$

NOT



$F = \bar{A}$

EXOR



$F = A \oplus B$

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

EXNOR

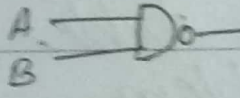
$F = A \odot B$

$= AB + A\bar{B}$

Universal Gates →

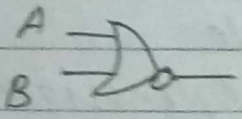
Any logical expression / circuit can be represented only in terms of NAND / NOR gates. Hence, they are known as universal gates.

NAND



$F = \overline{AB}$

NOR



$F = \overline{A + B}$

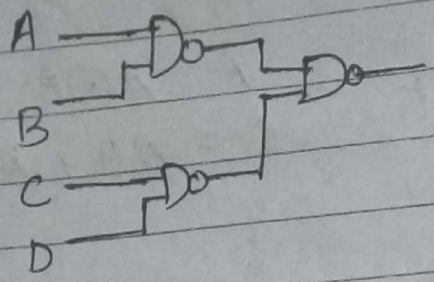
* To write any exp in form of NAND gates find simplified SOP and then take complement twice
* For NOR use POS.

ex $F = AB + CD$

$F = [(AB + CD)']'$

$= [(AB)' \cdot (CD)']'$ → in form of 3 nands.

$= NAND(NAND(A,B), NAND(C,D))$



Combinational Circuits

It is a logical circuit in which the output is independent of previous output.

Steps to form a Combinational Circuit.

1. Make a truth table using the word problem.
2. Write the expression using the truth table.
3. Simplify the expression to form a standard exp.
4. Implement it using logic gates.

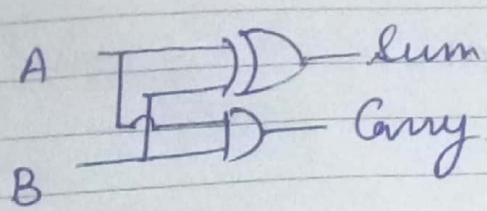
ADDER.

Half Adder → adds two 1 bit nos. and return Sum and Carry.

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

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

$Carry = AB$

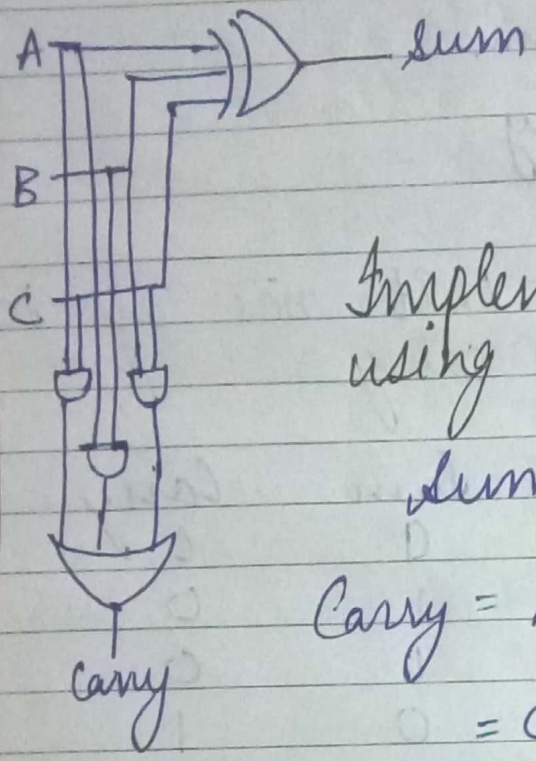


Adder → Adds 3 1bit nos are returns sum and carry.

A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$Sum = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$
$$= \bar{A}(\bar{B}C + B\bar{C}) + A(\bar{B}\bar{C} + BC)$$
$$= \bar{A}(B \oplus C) + A(B \odot C)$$
$$= \bar{A}(B \oplus C) + A\overline{(B \oplus C)} \quad [\text{Since } A \odot B = \overline{A \oplus B}]$$
$$= A \oplus (B \oplus C)$$
$$= A \oplus B \oplus C$$

$$Carry = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$
$$= \bar{A}BC + ABC + A\bar{B}C + ABC + AB\bar{C} + ABC \quad [X + X = X]$$
$$= (\bar{A} + A)BC + (\bar{B} + B)(AC) + (\bar{C} + C)(AB)$$
$$= AB + BC + CA \quad [\bar{X} + X = 1]$$



Implementing Full Adder using 2 Half-Adders

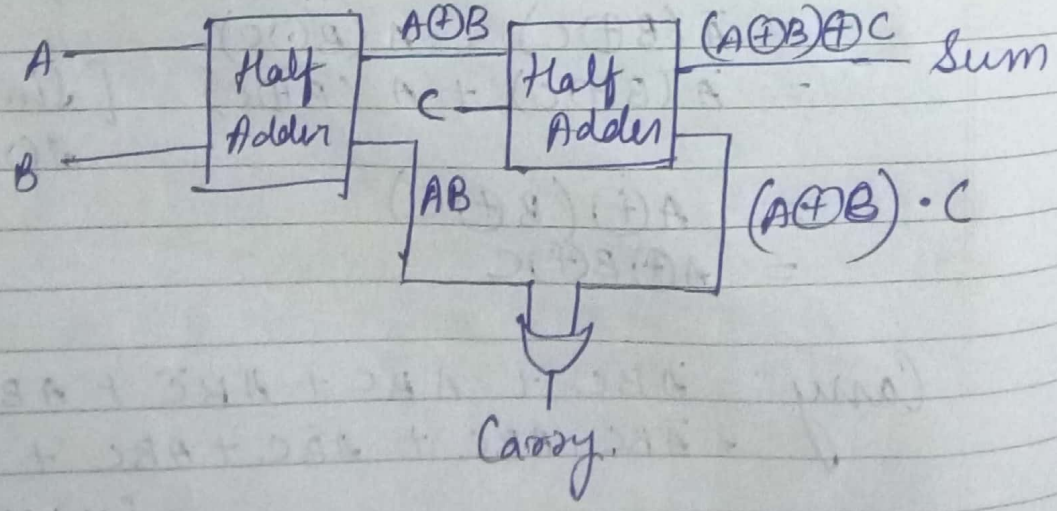
$$Sum = [A \oplus B] \oplus C$$

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

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

$$= (A \oplus B) \cdot C + AB.$$

Now, $A \oplus B$ is sum of half adder and AB is its carry.



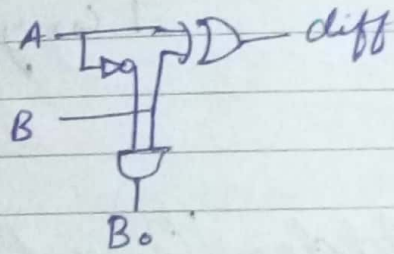
SUBTRACTOR

Half Sub → Subs two one bit nō, and returns diff and borrow.

A	B	Diff	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

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

$B_o = \bar{A}B$



Full Sub → Sub three one bit nō such as $(A - B) - C$.

A	B	C	Diff	B_o
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

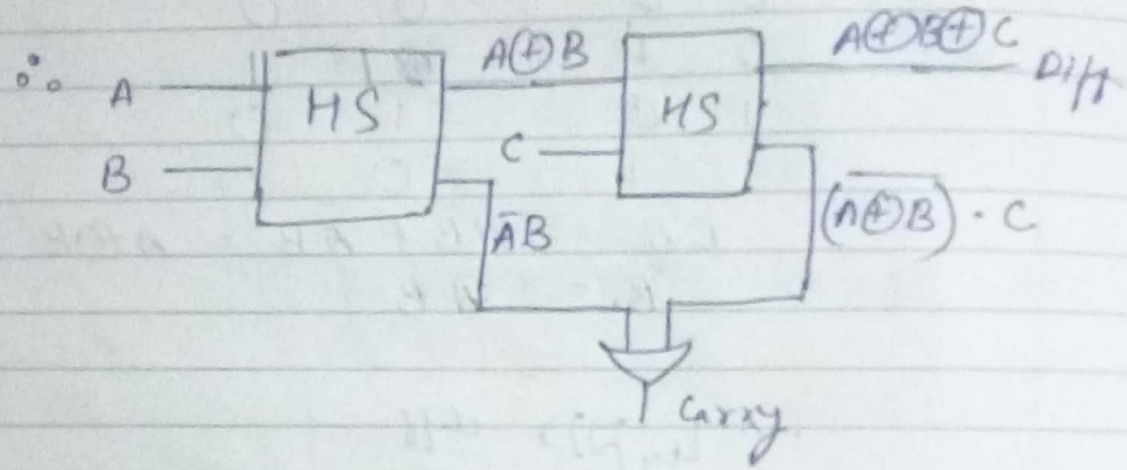
$Diff = A \oplus B \oplus C$ (same as Adder)

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

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

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

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



Parallel Binary Adder / Carry Propogate Adder

Adding 4 bit no

$C_3\ C_2\ C_1\ C_0 = 0$

$A_3\ A_2\ A_1\ A_0$

$B_3\ B_2\ B_1\ B_0$

$C_4\ S_3\ S_2\ S_1\ S_0$

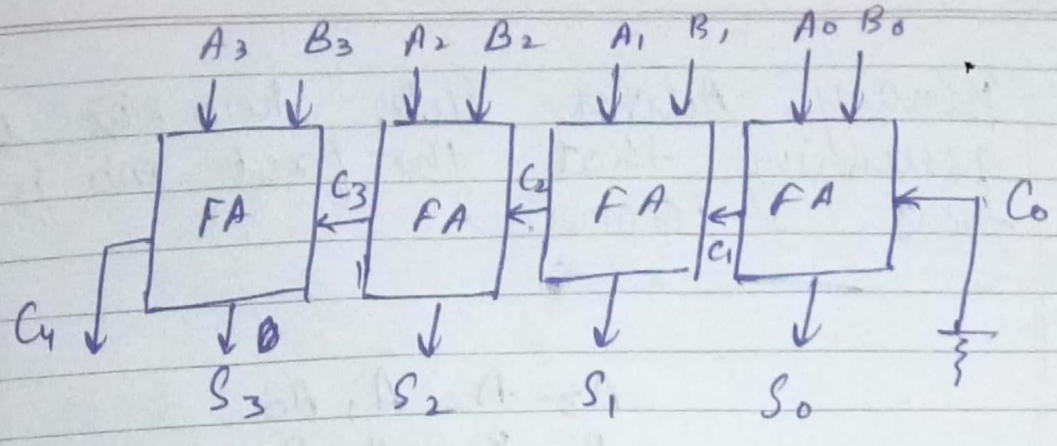
$C_1 \rightarrow$ Carry of $A_0 + B_0$

$C_2 \rightarrow$ Carry of $A_1 + B_1$

$C_3 \rightarrow$ Carry of $A_2 + B_2$

$C_4 \rightarrow$ Carry of $A_3 + B_3$

8 inputs are given to 2 Adders and the carry of one is passed as an input to another.



Binary Subtractor

* Subtraction using 2's Compliment

For A − B.

① Take 2's compliment of B.

② Add A and B.

③ If B < A; their addition is the final ans just neglect carry.

④ If B > A; Take two's compliment of the result and put a −ve sign.

Two's Compliment

① Replace 0s by 1s and 1s by 0s.

② Add '1' to the outcome.

1010 ⟷ 0101 + 1

= 0110

Binary Adder Sub. two nos. assuming that the first no is larger than second.

$$\begin{array}{l} A_3\ A_2\ A_1\ A_0 \\ -\ B_3\ B_2\ B_1\ B_0 \end{array}$$

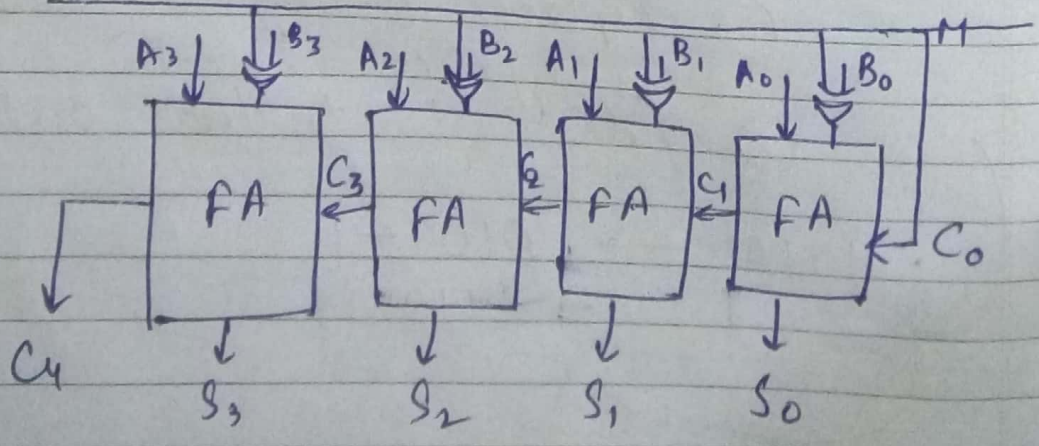
$$\Updownarrow$$

$$\begin{array}{l} C_3\ C_2\ C_1\ C_0 = 1 \\ A_3\ A_2\ A_1\ A_0 \\ +\ B_3'\ B_2'\ B_1'\ B_0' \\ \hline C_4\ \ S_3\ S_2\ S_1\ S_0 \end{array} \rightarrow \text{diff.}$$

B	$B \oplus 1$	$B \oplus 0$
0	1	0
1	0	1

$$B \oplus 1 = B'$$
$$B \oplus 0 = B$$

∴ A parallel adder/Sub can be made as.



If $M = 0$.

$C_0 = 0$

and $B \oplus M = B$

$\therefore$ It works as an adder.

If $M = 1$

$C_0 = 1$

and $B \oplus M = B'$

$\therefore$ It works as a sub.

Carry Look Ahead Adder.

Parallel Adder takes more time as each FA has to wait for the previous FA to complete its process. Thereby increasing time by 4 times.

$\therefore$ we use Carry Look ahead adder, which calculates Carry in advance.

$$S_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = (A_i \oplus B_i) C_i + A_i B_i$$

Put $A_i \oplus B_i = P_i$

and $A_i \cdot B_i = G_i$

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = P_i C_i + G_i$$

$C_0 = 0$

For $C_1 = P_0 \cdot C_0 + g_0$

$C_1 = g_0$ — (1)

$C_2 = P_1 \cdot C_1 + g_1$

$C_2 = P_1(g_0) + g_1$ (from (1))

$C_2 = P_1 g_0 + g_1$ — (2)

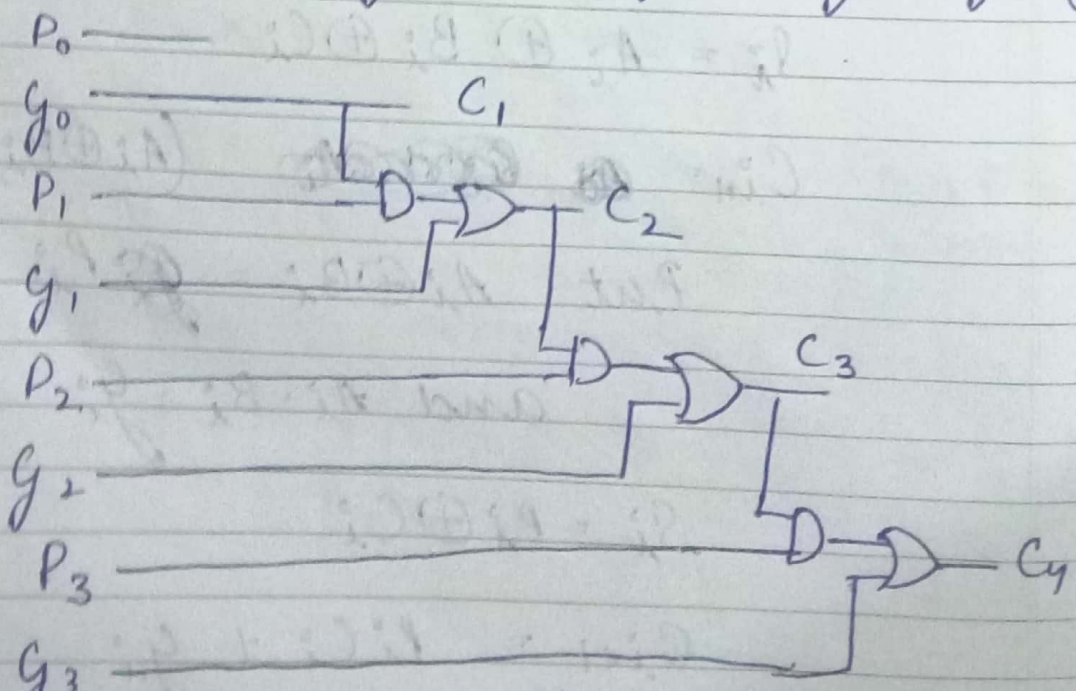
$C_3 = P_2 C_2 + g_2$

$C_3 = P_2(P_1 g_0 + g_1) + g_2$

$C_3 = P_2 P_1 g_0 + P_2 g_1 + g_2$ — (3)

$C_4 = P_3 C_3 + g_3$

$C_4 = P_3 P_2 P_1 g_0 + P_3 P_2 g_1 + P_3 g_2 + g_3$ — (4)



Circuit A

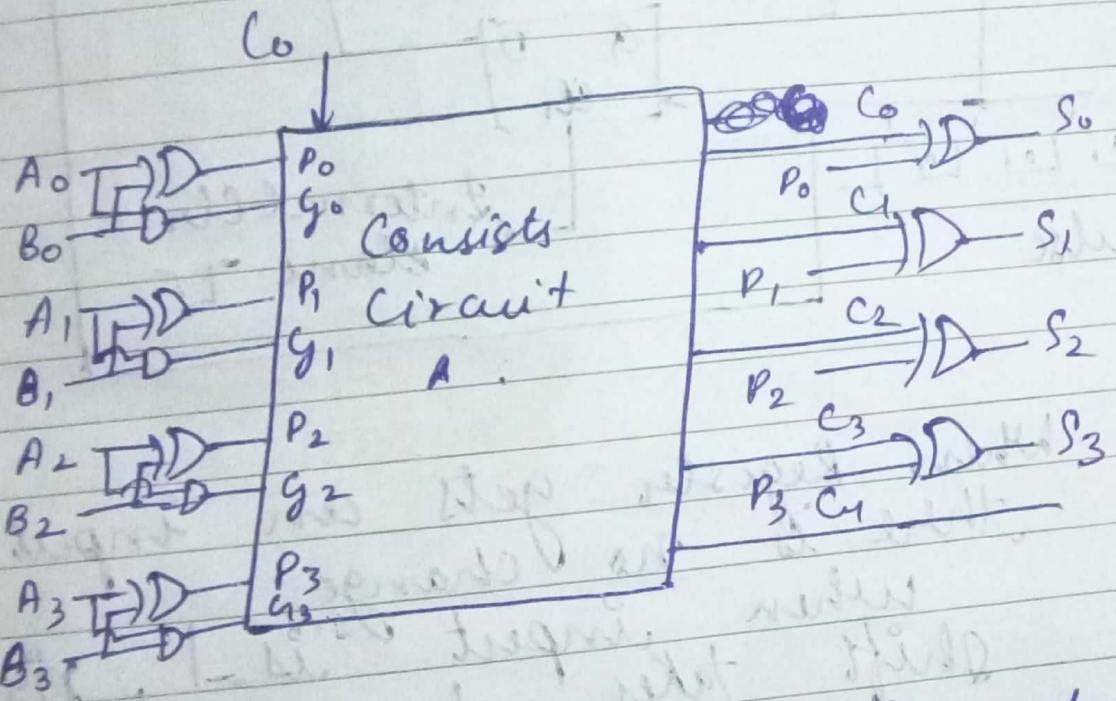
For Sum. $S_0 = P_0 \oplus C_0$.

$S_1 = P_1 \oplus C_1$

$S_2 = P_2 \oplus C_2$

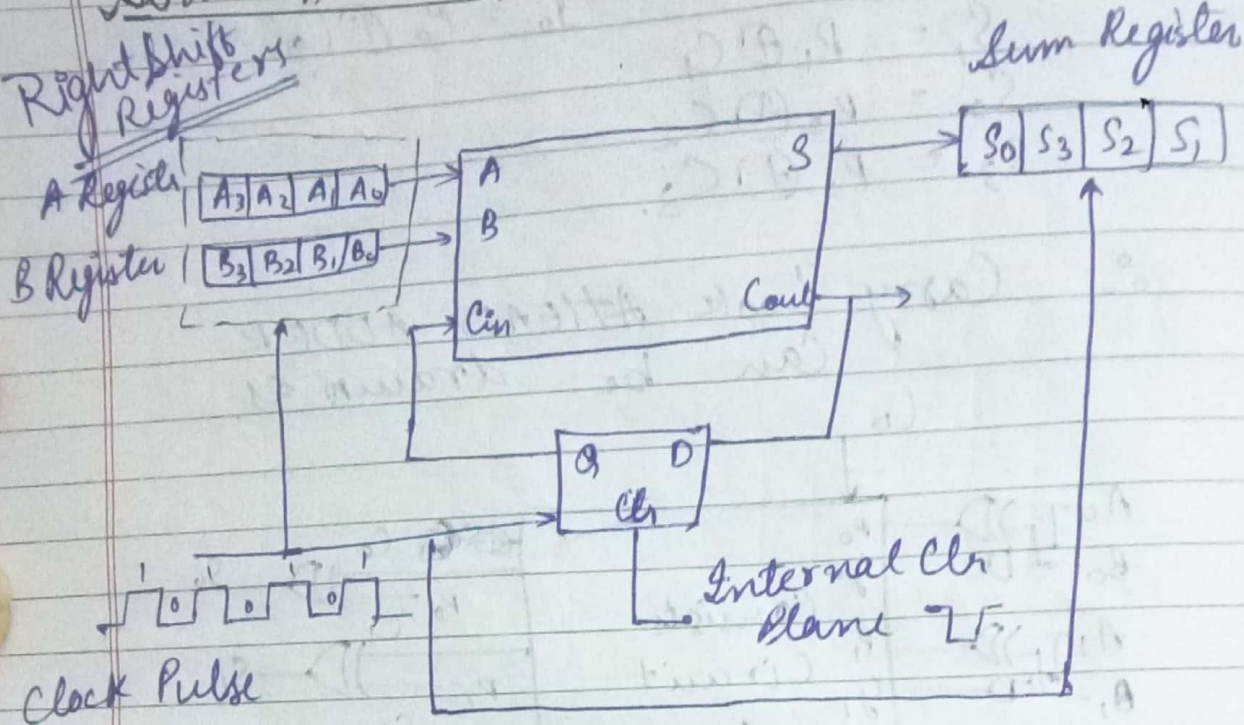
$S_3 = P_3 \oplus C_3$.

∴ Carry Look AHEAD ADDER Can be drawn as.



A Carry look Ahead Adder reduces time but increases circuitry. And that is its drawback.

Serial Adder



When Register gets an input 0 there is no change, when input is 1, right shift takes place.

When clock is 0

$A = A_0$ $\quad S = A \oplus B \oplus C_{in}$

$B = B_0$ $\quad Cout = Carry$

$C_{in} = 0$ $\quad$ Cout is stored in memory

When clock is 0

$S_0 = S$

registers shift.

A_0	A_3	A_2	A_1

B_0	B_3	B_2	B_1

S_1	S_0	S_3	S_2

When clock = 1

$A = A_1$

$B = B_1$

C_{in} = previous C_{out}

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

C_{out} = Carry

$S_1 = S_0$

C_{out} is stored in memory

And this continues till complete addition occurs.

Final Carry and $S_3 S_2 S_1 S_0$ give us the sum.

Carry Save Adder

Adds more than 2 no's simultaneously

First 3 no are given as 11 inputs and later new no are introduced on every step.

In the end a 11 adder is added to complete the addition.

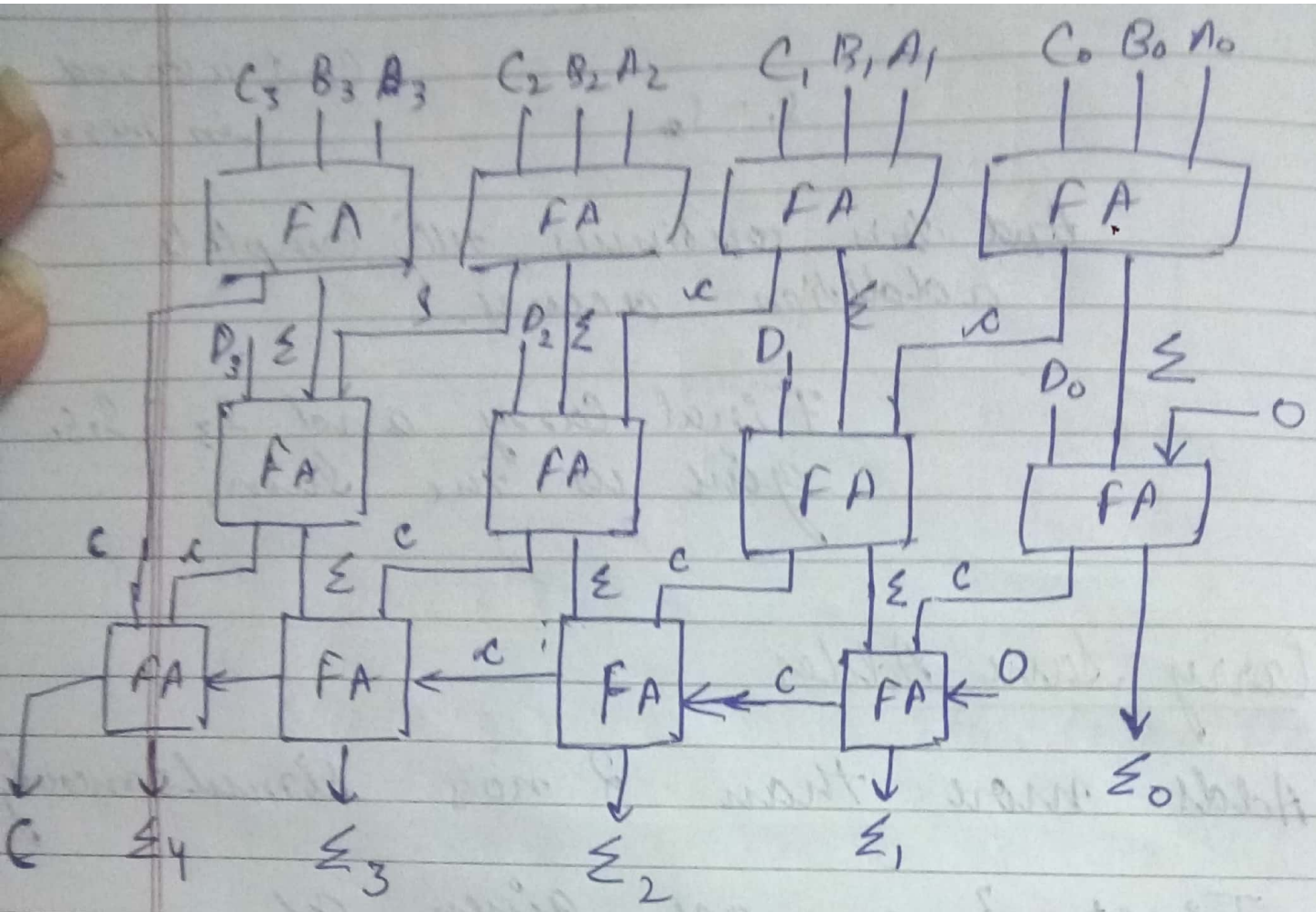
4 No Adder can be constructed with

$A = A_3 A_2 A_1 A_0$

$B = B_3 B_2 B_1 B_0$

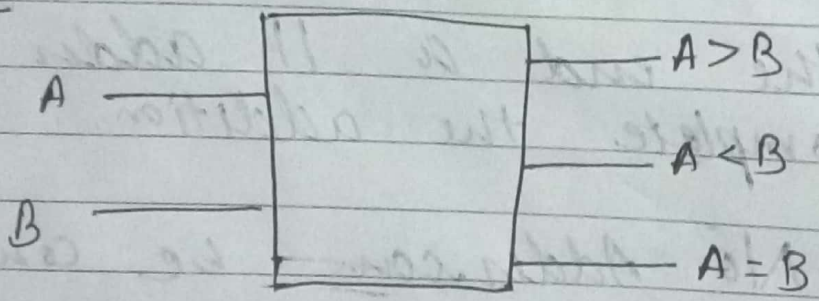
$C = C_3 C_2 C_1 C_0$

$D = D_3 D_2 D_1 D_0$



Comparitors

1-Bit



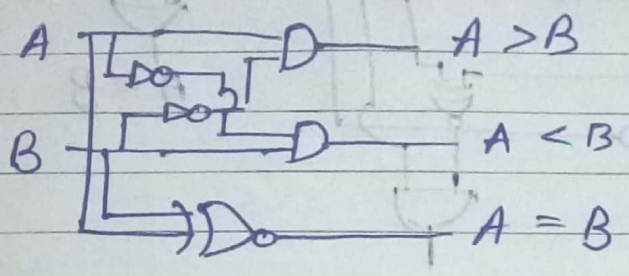
A>B

A	B	A>B	A<B	A=B
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

$A > B \Rightarrow A\bar{B}$

$A < B \Rightarrow \bar{A}B$

$A = B \Rightarrow A \odot B$



2-Bit Comparitor

$A = A_1 A_0$

$B = B_1 B_0$

$A_1 > B_1 \longrightarrow A > B$

$A_1 < B_1 \longrightarrow A < B$

$A_1 = B_1$

$A_0 > B_0 \longrightarrow A > B$

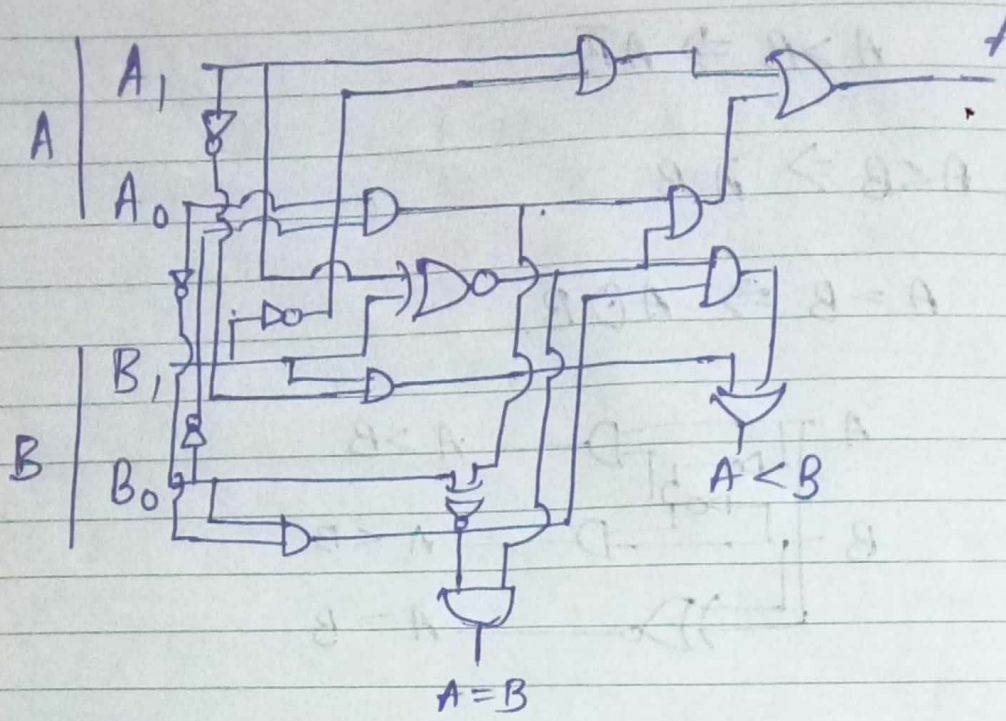
$A_0 < B_0 \longrightarrow A < B$

$A_0 = B_0 \longrightarrow A = B$

$A > B \Rightarrow A_1\bar{B_1} + (A_1 \odot B_1) A_0 \bar{B_0}$

$A < B \Rightarrow \bar{A_1} B_1 + (A_1 \odot B_1) B_0 \bar{A_0}$

$A = B \Rightarrow (A_1 \odot B_1)(A_0 \odot B_0)$



Parity Generators

Parity Generators and checkers are used to check is a signal transmitted is same as the signal recieved.

A parity bit is generated at the transmitting end and is cross-verified at the recieving end.

They are of two types

even bit parity generator

Parity bit added is such that no. of bits including the parity bit is even.

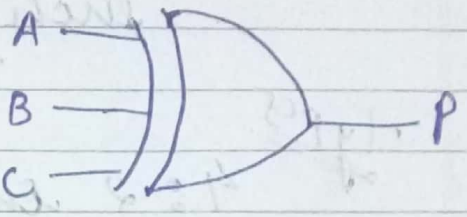
odd bit Parity generator

No. of bits including the parity bit should be odd.

Even Bit

A	B	C	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$P = A \oplus B \oplus C$



Odd Bit

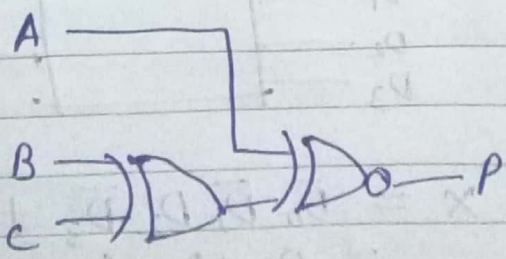
A	B	C	P
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

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

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

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

$= A \odot (B \oplus C)$



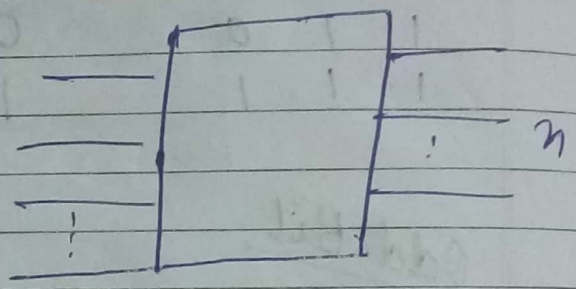
Encoder →

An encoder codes a message with m input channels into a message with n output channels such that

$$m \leq 2^n$$

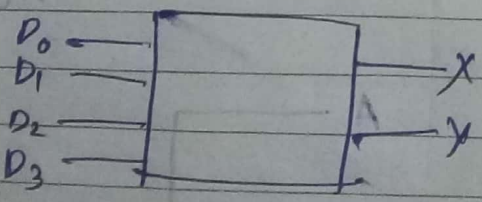
Types

4 : 2 encoder
8 : 3
16 : 4
⋮



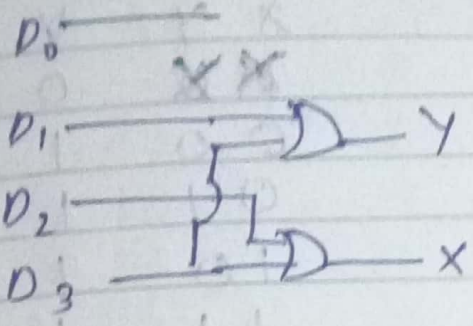
4 → 2 (only one bit can be one at a time)

D_0	D_1	D_2	D_3	X	Y
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1



$$X = \bar{D_0}\,\bar{D_1}\,D_2\,\bar{D_3} + \bar{D_0}\,\bar{D_1}\,\bar{D_2}\,D_3$$
$$= D_2 + D_3$$ (because when D_2 is 1 rest are automatically 0)

$$Y = D_1 + D_3$$



8 → 2

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

$X = D_4 + D_5 + D_6 + D_7$

$Y = D_2 + D_3 + D_6 + D_7$

$Z = D_1 + D_3 + D_5 + D_7$

Limitations

1. Only one bit can be 1.
2. 0000 gives same output as 1000.

Priority Encoder

It eliminates the limitations of encoder by introducing another output V.

When V is 1 → Inputs are valid

V is 0 → Incorrect inputs.

D_0	D_1	D_2	D_3
0	0	0	0
1	0	0	0
x	1	0	0
x	x	1	0
x	x	x	1

X	Y	V
~~X~~	~~X~~	0
0	0	1
0	1	1
1	0	1
1	1	1

$$X = D_3 + \bar{D_3} D_2 = D_3 + D_2$$

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

$$Y = D_3 + \bar{D_3} \bar{D_2} D_1$$

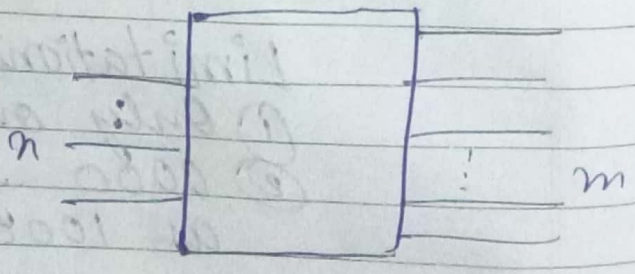
$$= D_3 + \bar{D_2} D_1$$

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

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

Decoder A decoder decodes a message with n input lines & m output lines such that

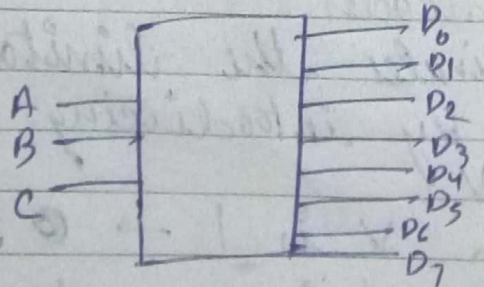
$$m \leq 2^n$$



Types

3 : 8
2 : 4
4 : 16

3 x 8



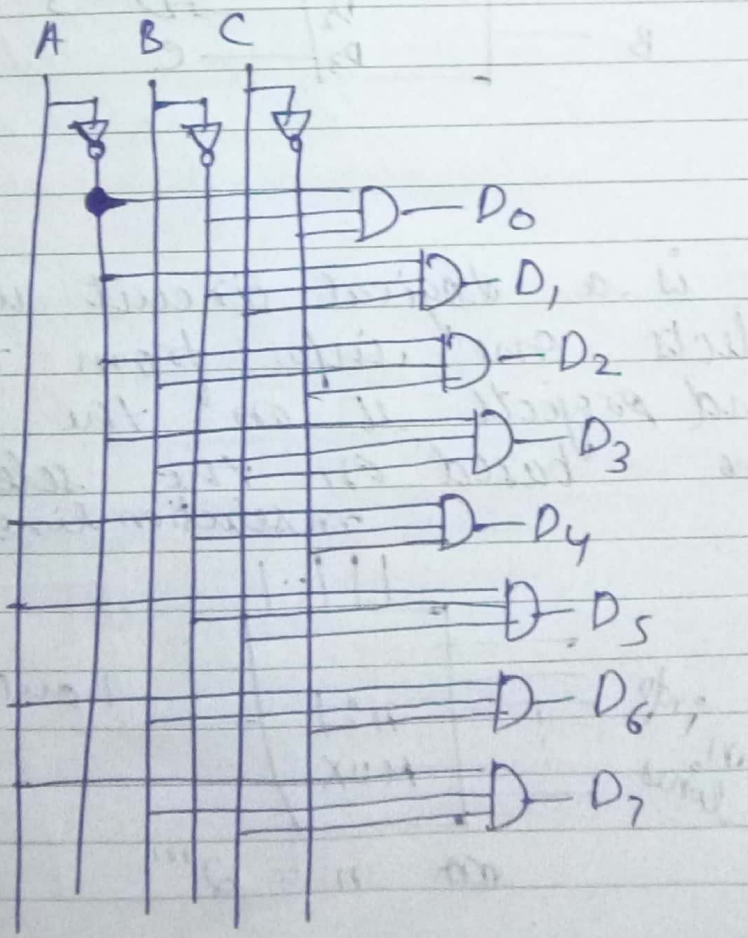
A B C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0 0 0	1	0	0	0	0	0	0	0
0 0 1	0	1	0	0	0	0	0	0
0 1 0	0	0	1	0	0	0	0	0
0 1 1	0	0	0	1	0	0	0	0
1 0 0	0	0	0	0	1	0	0	0
1 0 1	0	0	0	0	0	1	0	0
1 1 0	0	0	0	0	0	0	0	0
1 1 1	0	0	0	0	0	0	0	1

$D_0 = \bar{A}\bar{B}\bar{C}$ $D_4 = A\bar{B}\bar{C}$

$D_1 = \bar{A}\bar{B}C$ $D_5 = A\bar{B}C$

$D_2 = \bar{A}B\bar{C}$ $D_6 = AB\bar{C}$

$D_3 = \bar{A}BC$ $D_7 = ABC$.



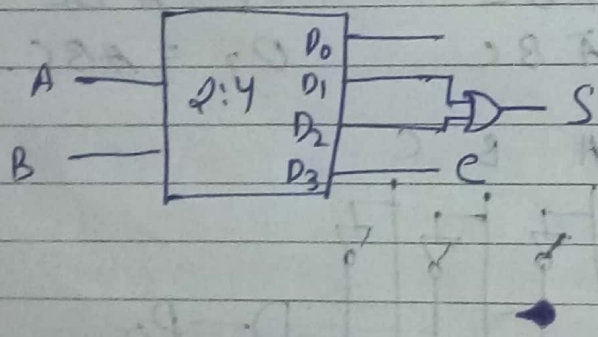
Implement a circuit using Decoder

1. Write the function in canonical form.
2. OR the outputs corresponding to minterms

Adder using Decoder

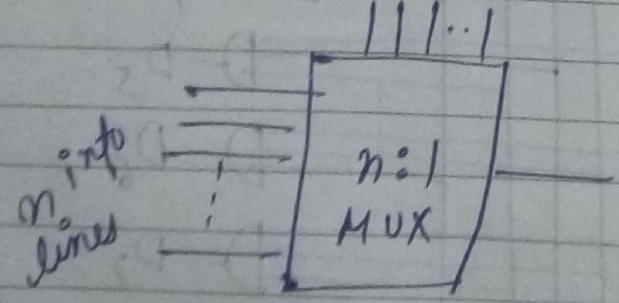
$S = \bar{A}B + \bar{B}A$

$C = AB$



MUX

It is a logical circuit which selects one inp. from the source and projects it on the output line based on the selection lines

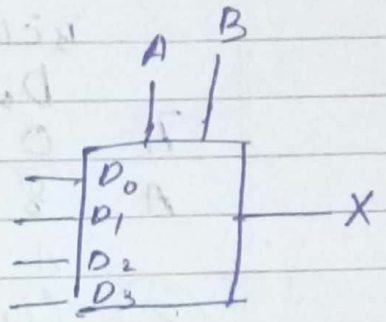


$n = 2^m$

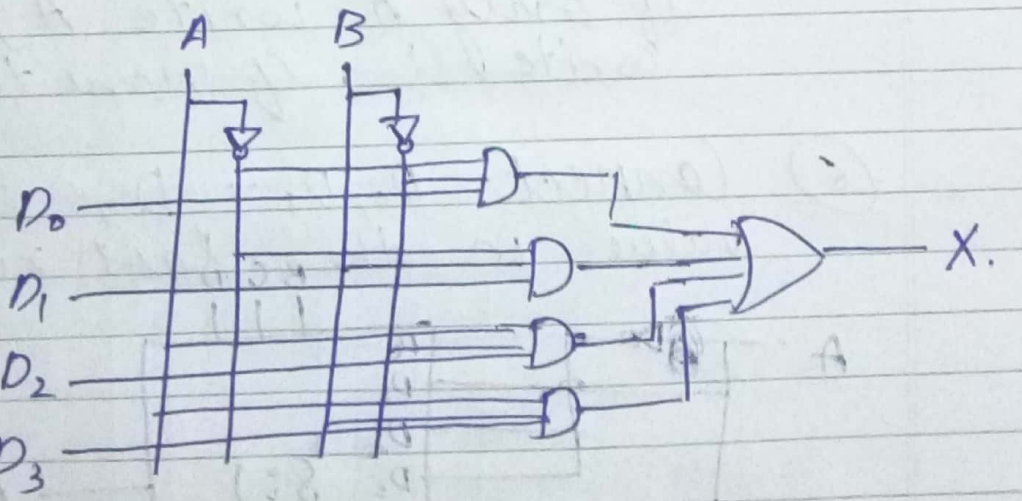
Types 4:1
8:1
16:1
:

4:1 MUX

A	B	X
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



$X = \bar{A}\bar{B}D_0 + \bar{A}BD_1 + A\bar{B}D_2 + ABD_3.$



Implement a circuit using MUX.
If no of variables = N
So, we use
$2^{n-1}:1$ mux.

Explaination with ex.

$(A,B,C,D)F = \Sigma(0,3,2,9,12,13,15)$

(1) Take BCD as selection lines.

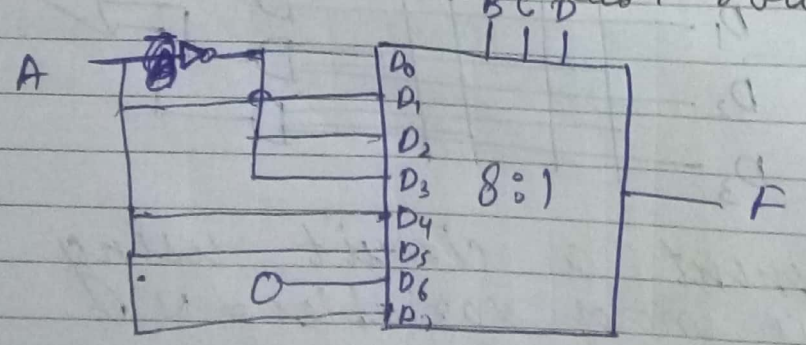
(2) Draw a table for minterms corresponding to BCD and A.

	$\bar{B}\bar{C}\bar{D}$	$\bar{B}\bar{C}D$	$\bar{B}C\bar{D}$	$\bar{B}CD$	$B\bar{C}\bar{D}$	$B\bar{C}D$	$BC\bar{D}$	BCD
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	$\bar{A}$	A	$\bar{A}$	$\bar{A}$	A	A	0	A

(3) Circle the minterms present in ques.

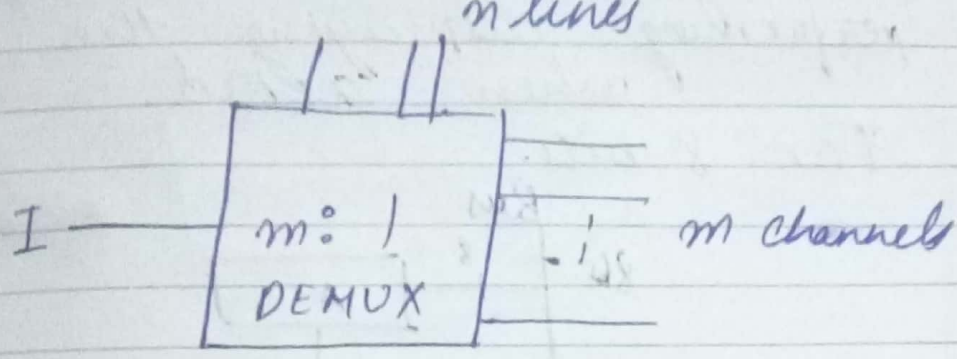
(4) If only $\bar{A}$ is circled write $\bar{A}$ if only A write A; if both write 1; if none the 0.

(6) Connect $D_0, D_1 \cdots D_7$ to the value in the last row.



DEMUX

A logical circuit which projects an information in one of the output channels according to the selection lines.



$m = 2^n$

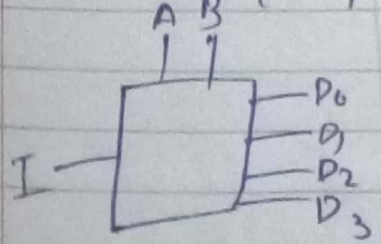
1 : 4

1 : 8

1 : 16

1 : 4

A	B	D_0	D_1	D_2	D_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

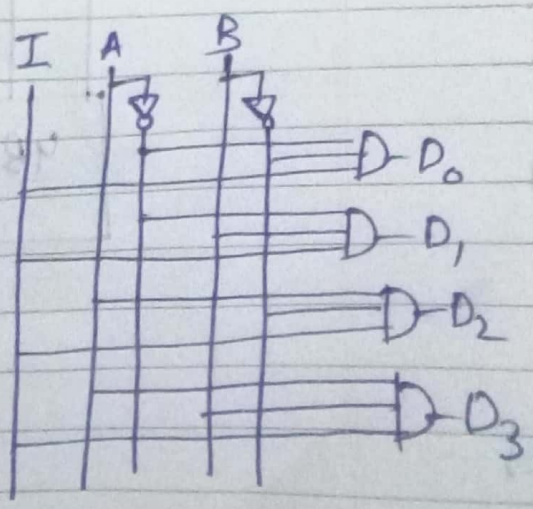


$D_0 = \bar{A}\bar{B}I$

$D_1 = \bar{A}BI$

$D_2 = A\bar{B}I$

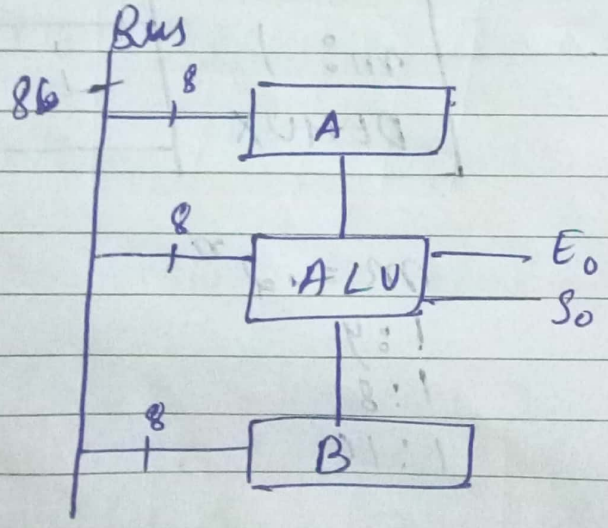
$D_3 = ABI$



ALU

ALU Performs arithmetic operations (addition/sub in this case) in a computer by taking the no from the registers (saved values) and ~~performing~~ displaying the output when asked.

For 8 bits.



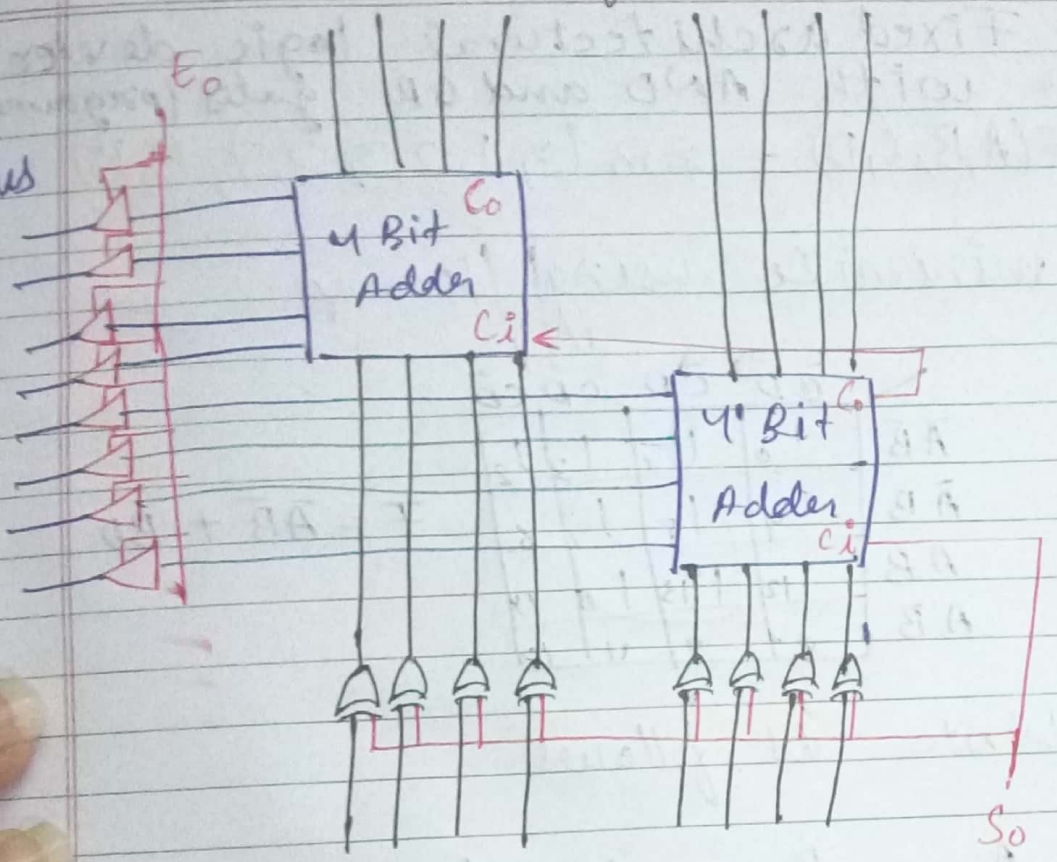
When $E_0 = 1$ → display

when $S_0 = 1$ → Sub

$S_0 = 0$ → add.

A register

A Register



B Register

PLA and PAL

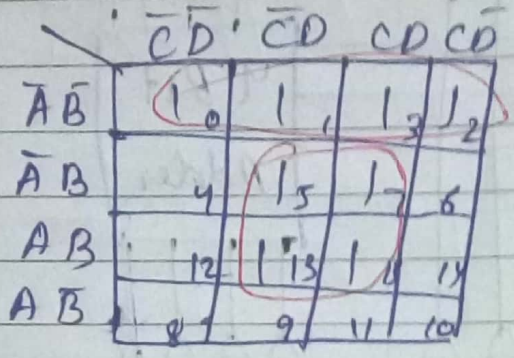
PAL → Programable Array Logics.
PALA → Programable Logic Arrays

PLA	PAL
① Shared Terms	① No shared terms
② Fully Programable	② Not fully programable.

PLA. Fixed architectural logic device with AND and OR gates (programable)

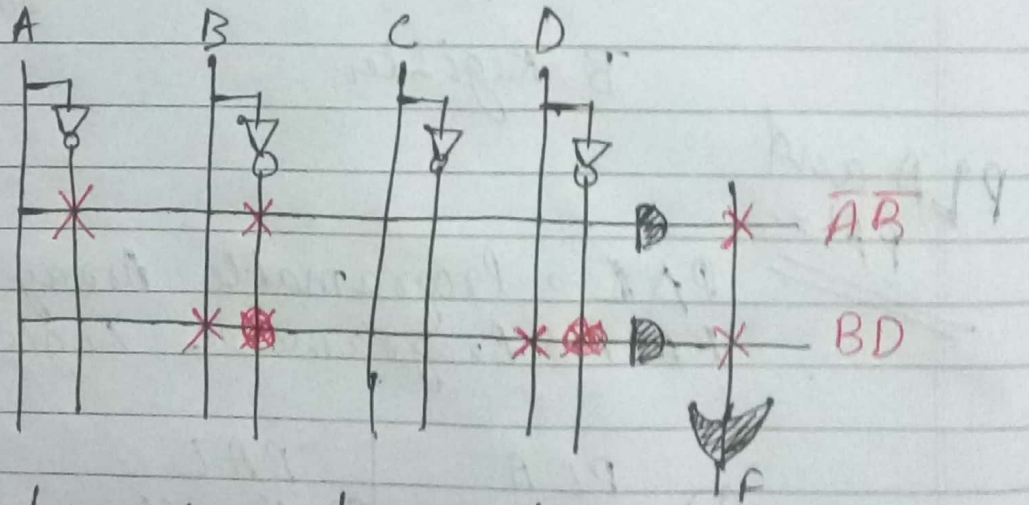
example. $F(A,B,C,D) = \Sigma m(0,1,2,3,5,7,13,15)$

(1) minimize using k map



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

(2) Write as follows.

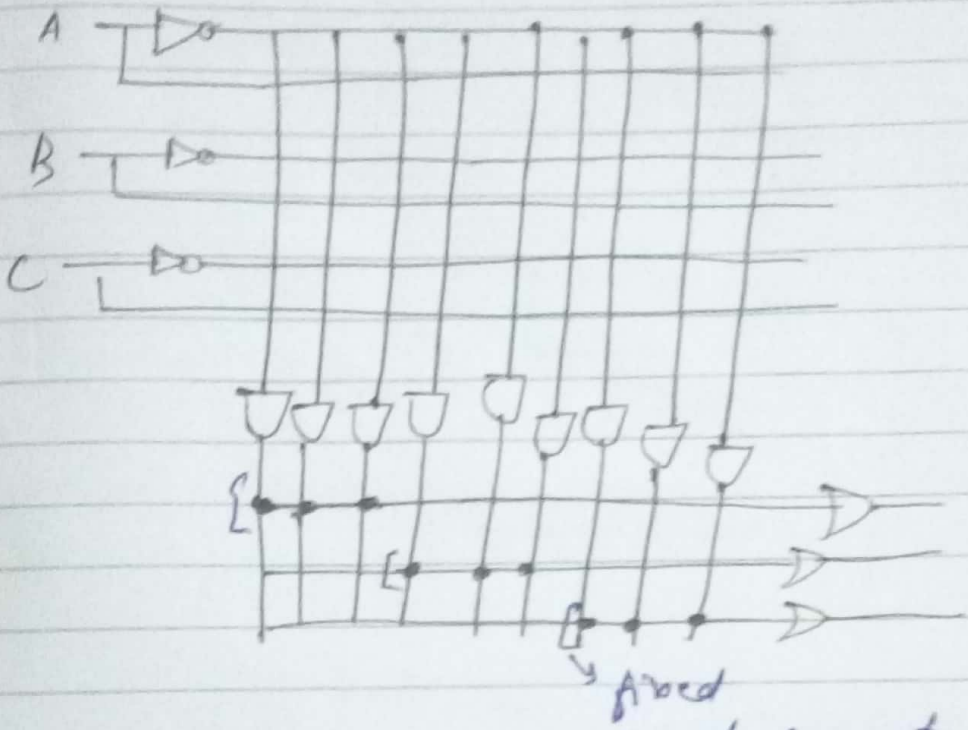


(3) We have two terms hence draw two horizontal lines with AND

(4) Cut the lines and draw an OR to represent OR in the F.

(5) Put crosses on the literals used.

PAL → Fixed Architectural logic device with programable AND and fixed OR gates.



we cannot put crosses acc to our will.

Rest is same as PLA

Point Conversions

Decimal → Binary

1. Separate the values bf and aft point.
2. For left side convert acc to normal rules. (diving by 2)
3. For right side, multiply by 2, on every step note the value of the no bf decimal point.
4. Right it from upper to lower.

ex → 3.75

11

.75 × 2 = 1.5 → 1

.5 × 2 = 1.0 → 1

11.11

ex → 13.25

1101

.25 × 2 = 0.5 → 0

0.5 × 2 = 1.0 → 1

1101.01

Binary to decimal. (-ve powers of 2 for right of point)

$(1101.01)_2 = (?)_{10}$

$(1\times2^3)+(1\times2^2)+(0\times2^1)+(1\times2^0)+(0\times2^{-1})+(1\times2^{-2}) = 3.25$

For octal and Hex we 8 and 16 resp.