3. Logic functions

Date 09/02/19

(1) Basic properties of ~~logic~~ switching algebra:-

→ There are only two var. 0 & 1 (also called Boolean var) in switching algebra.

Basic properties:-

(i) Idempotency:-

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

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

dual ↔ dual

(ii) Commutativity:-

→ $x + y = y + x$ → $xy = yx$

dual

(iii) Associativity:-

→ $(x+y)+z = x+(y+z)$

→ $(x \cdot y) \cdot z = x \cdot (y \cdot z)$

Both OR & AND are as well as left & Right associative

(iv) Complementation:-

→ $x + \bar{x} = 1$ → $x \cdot \bar{x} = 0$

(v) Distributivity:-

→ $x(y+z) = xy + xz$ ('.' is distributive over '+')

→ $x + yz = (x+y)(x+z)$ ('+' is distributive over '.')

→ Every pairs are dual, ie. $\cdot \rightarrow +$, $+ \rightarrow \cdot$, $1 \rightarrow 0$ & $0 \rightarrow 1$. in all the above properties.

Spiral

(2) Switching expressions & simplifications:-

Date.....

Switching expressions:- It is a finite no. of comb^n of switching variables & constants $\{0,1\}$ by means of switching oper^{n}s. $(+, \cdot, \text{NOT})$.

ex:- $x + \bar{x}yz + x\bar{z}$. (no. of literals = 6) (no. of var = 3)

Literals:- Any var. which is present in switching exp. either in its true form or complemented form.

→ Since these switching expressions are realised to circuits by using gates, thus our main aim is to simplify the switching expressions to make circuit eff^n.

Properties for simplifying switching expression:-

(i) **Absorption:-** $\boxed{x + xy = x}$ ***

$\Rightarrow x \cdot 1 + xy \Rightarrow x(1+y) = x$ ∵ it's dual

$\boxed{x \cdot (x+y) = x}$

(ii) $\boxed{x + x'y = x + y}$ *** $\xrightarrow{\text{dual}}$ $\boxed{x \cdot (\bar{x} + y) = xy}$

$\Rightarrow (x+x')(x+y) \Rightarrow (x+y)$ {using distributivity ie. '+' is distributive over '.'}

(iii) ******* **Consensus theorem:-**

$\boxed{xy + \bar{x}z + yz = xy + \bar{x}z}$

If an exp. 'x' ANDED with two exp. y & z once with its true form & other by it false form then (y·z) is redundant.

Spiral

Date.....

proof:-

$\Rightarrow xy + \bar{x}y + yz(x+\bar{x}) \Rightarrow$

$\Rightarrow xy + \bar{x}y + yzx + yz\bar{x} \Rightarrow xy(1+z) + \bar{x}z(1+y)$

$\Rightarrow \boxed{xy + \bar{x}z}$

ex:- simplify:-

(i) $(x'y'z + yz + xz) \Rightarrow z(\bar{x}\bar{y} + x + y)$

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

$\Rightarrow (z)\ \bar{y}\ \ y$

gn switching exp:- $\boxed{\text{if } x+y = x+z \not\Rightarrow (y=z)}$ X

$\boxed{\text{if } x\cdot y = y\cdot z \not\Rightarrow (x=z)}$

(3) Demorgans law & simplification:-

(i) $\boxed{(\overline{xy}) = (\bar{x} + \bar{y})}$ (ii) $\boxed{(\overline{x+y}) = (\bar{x}\cdot\bar{y})}$

Demorgan's → To get the complement of an exp.

$f(a, b, c, -z, 0, 1, \cdot, +)$

$\bar{f} = f(\bar{a}, \bar{b}, \bar{c}, -z, 1, 0, +, \cdot)$ ⇒ Complement of f

$\Rightarrow$ (dual of f) $\Rightarrow f_{dual} = H(a, b, c, -z, 1, 0, +, \cdot)$ ~~Only 1→0.~~

Apart from literals everything is complemented

ex: $f = \bar{x} + \bar{y} \Rightarrow \bar{f} \Rightarrow (x\cdot y)$

$\Rightarrow f_{dual} = \bar{x}\cdot\bar{y}$

***

Date.....

ex:- Simplify:

$\Rightarrow (x+y)(x'(y'+z'))' + x'y' + x'z'$

$\Rightarrow (x+y)(x'(\overline{x \cdot y})' + x'(\overline{yz})$

$\Rightarrow (x+y)((\overline{x + xy})' + (\overline{x+yz})$

$\Rightarrow (x+y)(x) + (x+yz)'$ ⇒ (1) ⇒ final answer

~~$x + (x+yz)'$~~

~~$x + ((x+y)(x+z))'$~~

~~$\Rightarrow x + \bar{x}(\overline{yz})$~~

~~$\Rightarrow x + \bar{x}(x \cdot (\bar{x}+y)$~~

~~$\Rightarrow x + \bar{x}(x+x) + x \cdot \overline{yz}$~~

$x(\overline{yz} + yz) + \bar{x}\,\overline{yz}$ ⇒ $\overline{yz} + x$

$\Rightarrow x(1) + \bar{x}\,\overline{yz}$

⇒

(9) Switching functions:-

→ Every ~~Boolean~~ switching expressions can act as switching func^n which gives some set of O/P for some set of i/P.

→ Switching function (or Boolean func^n) can have only two values 0 & 1.

Every func^n can be represented in cannonical form.

*** Cannonical form:- ORing of all the terms where func^n produces o/p 1.

ex:- $f = \bar{a}bc + ab\bar{c} + \cdots$

Date.....

(5) Cannonical sum of products:-

Min term:- A product term which contains each of 'n' variables as factors either in complemented or uncomplemented form is called a min term.

-> A minterm gives the value '1' for exactly one comb^n of the variables.

-> The sum of all minterms of 'f' for which 'f' assumes '1' is called Cannonical sum of products or disjunctive normal form.

OR

(It means each term is min-term in sum of product.)

ex:-

a	b	c	f
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	1
1	1	1	0

$$f = \bar{a}bc + \bar{a}b\bar{c} + a\bar{b}\bar{c} + ab\bar{c}.$$

Cannonical sum of Product

or

$$f = \Sigma(1,2,4,6)$$

(6) Cannonical product of sums:-

Max term:- A sum term which contains each of 'n' var. as factors either in complemented or uncomplemented form is called a max term.

-> A Max term gives the value '0' for exactly one comb^n of variables.

Spiral

Date.....

→ The product of all max terms of 'f' for which 'f' assumes '0', is called cannonical product of sums or (Conjunctive) normal form.

(AND)

ex:-
So, for previous ex:-

~~Canno~~ $f = (\bar{a}+\bar{b}+\bar{c})(\bar{a}+b+\bar{c})$

$f = (a+b+c)(a+\bar{b}+\bar{c})(\bar{a}+b+\bar{c})(\bar{a}+\bar{b}+\bar{c})$

Cannonical POS.

Write every term by complementing the variable
ex:- for $(0,0,0) \Rightarrow (a+b+c)$

because for $\bar{a}+b+\bar{c}$ it is $1+1+1=1$ not 0

or

$f = \Pi(0,3,5,7)$

(7) Examples of cannonical forms:-

ex:- $f(x,y,z) = x'y + z' + xyz$ Convert it to cannonical sop.

⇒ $f(x,y,z) = x'y(z'+z) + (x'+x)(y+y')z' + xyz$.

⇒ $x'yz' + x'yz + x'yz' + x'y'z' + xyz' + xy'z' + xyz$

Remove the repeating terms → Cannonical sop

⇒ $\Sigma(0,2,3,5,7)$ & $\Pi(1,5)$

Spiral

Date.....

(Q) By converting a minimal exp to cannonical form no. of literal ↑, why are we ↑ the no. of ~~minter~~ literal instead of ↓ it?

=> When we are using a already built circuit, ~~for use~~, then there must exist pins for all the minterms, while designing a new a circuit from the ~~scratch~~ scratch we need a minimized circuit.

(8) Functional properties:-

(i) The cannonical sop or pos form of a switching func^n is unique.

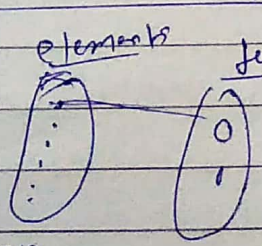
(ii) Two switching func^{n}s $f_1(x_1, x_2 \cdots x_n)$ & $f_2(x_1, x_2 \cdots x_n)$ are said to be logically equivalent iff both func^{n}s have same value for each & every comb^n of $(x_1, x_2, \cdots x_n)$

(iii) Two Swithing funct^n are equivalent if their cannonical pos or sop are identical.

(9) Number of functions:-

(Q) If there a 'n' boolean variables then how many boolean func^{n}s are possible.

=> So, total mapping:-



dec	1	2	3	4	—	n

2^n (rows)

2^n 2

***

total func^{n}s possible = 2^{2^n}

Date.....

(Q) no. of boolean functions for n-ternary variables:

⇒ Every variable can take 3-values:

⇒ total rows (or elements as i/p) $= 3^n$

so, $\boxed{\text{total boolean funcns} = 2^{3^n}}$

(Q) for 'n' k-ary variables then how many m-ary funcns are possible

⇒ $\boxed{\text{Total elements as i/p} = k^n}$

$\boxed{\text{total o/p possible} = m}$

so $\boxed{\text{total funcns} = m^{k^n}}$ ***

ex: for 2-boolean var. how many boolean funcns are possible?

⇒ $2^{2^2} \Rightarrow 16$ funcns are possible

ex:

a	b	f_1	f_2	f_3	...	f_{16}
0	0	0	0	0		1
0	1	0	0	0		1
1	0	0	0	1		1
1	1	0	1	0		1

→ 16 unique funcns

Neutral func^n:- func^n in which no. of minterms = maxterms

Date.....

(10) Counting no. of functions & neutral func^{n}s:-

xxx

(Q) How many boolean func^{n}s are possible with 3 var. such that there are exactly 3 Min terms?

⇒

a	b	c
2^3 elements		

Total functions possible = $2^{2^3} \Rightarrow 2^8 \Rightarrow 256$

→ exactly 3 - minterms = only 3 i/p comb^n should be given '1'- others '0'

So, no. of func^n = $^8C_3$

(Q) In the prev. question At most 3 min terms

⇒ $\left(^8C_0 + {}^8C_1 + {}^8C_2 + {}^8C_3\right)$ ⇒ function

⇒ no. of func^{n}s with at most 3 min terms

$^8C_0$ → exactly 0 min terms

$^8C_1$ → (exactly 1 min term)

(Q) How many neutral func^{n}s are possible with 2- boolean variables?

⇒

no. of elements = $2^2 = 4$

total neutral func^{n}s = $^4C_2 \Rightarrow 6$ (no. of min terms = max terms)

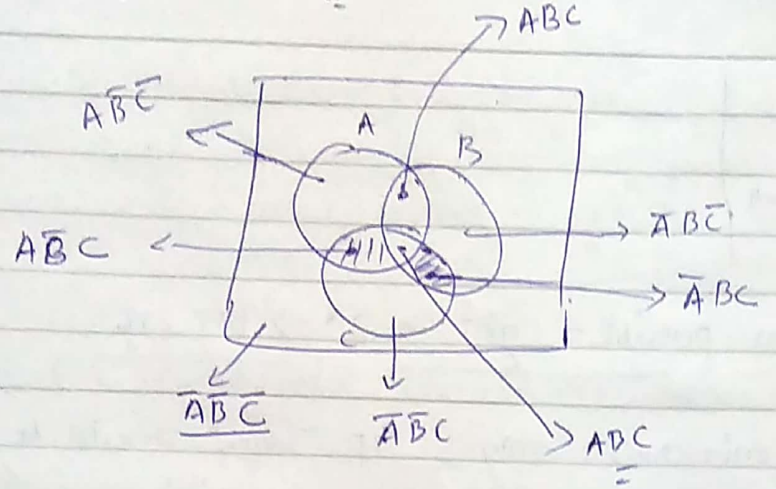
So, total no. of neutral function = $(2^n)C_{(2^n/2)}$ ⇒ with 'n' boolean var.

Spiral

Date.....

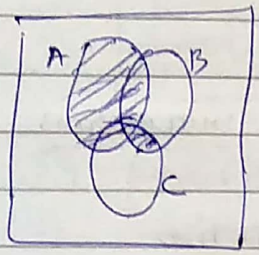
(11.) Venn diagram representation:-

(Q) → Boolean functions can be represented using venn diagram also.



{Total min-terms = 8 (for 3 vars)}

(Q) find the boolean func^n that following venn diagram represents.

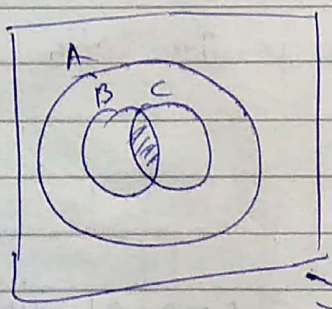


(a) $A+A'B'C$ (b) $A+BC$
(c) $A+A'BC$ (d) $AB+C$

→ $(A\bar{B}\bar{C}+ABC+AB+AB\bar{C})+(\bar{A}BC)$

→ $A+\bar{A}BC \Rightarrow A+BC$

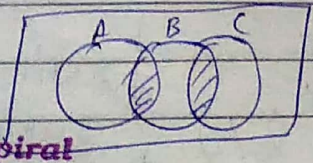
ex:-



⇒ Even though this venn diag. is meaningless it can be given in exams.

$F = ABC$

ex:-



⇒ $F = AB\bar{C}+\bar{A}BC$

{This venn diag. is also meaningless}

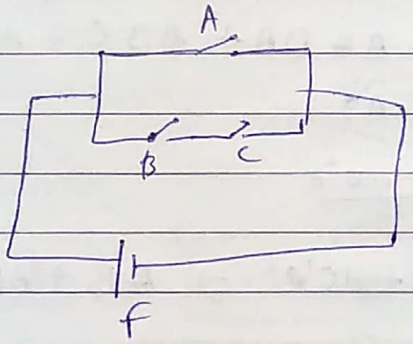
(12) Contact Representation:-

Date.....

→ Every boolean func^n can also be represented using Contact Representation (ie. serial or parallel contacts)

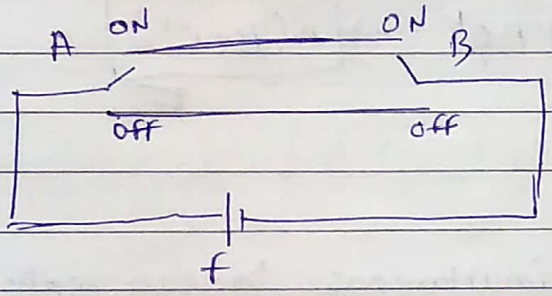
→ 'Serial' contacts performs 'AND' oper^n. While 'parallel' contact performs 'OR' oper^n.

ex:-



⇒ $f = A + BC$

ex:-



⇒ $f = (AB + \bar{A}\bar{B})$

***

→ Touching 'ON' means using the var. in its TRUE form & Touching 'Off' means using the var. in complemented form.

Procedure to solve complex questions:-

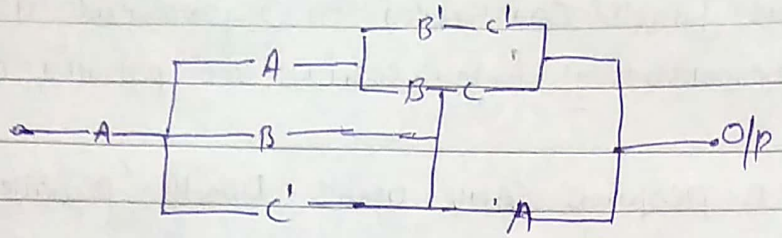
(1) Find the valid forward path
(2) perform 'OR' among them.

Forward path:- Any path starting ~~for~~ from i/p & ending at o/p w/o forming cycle.

Validity of forward path:- No path should contain a var. in both true & complemented form.

Date.....

ex:- Identify the boolean exp. given by following circuit.



⇒ $f = AAB'C' + AABC + AABA + ABC + ABA + AC'A + AC'CA$

↓ Not valid

⇒ $AB'C' + ABC + AB + AC'$ ⇒ $AB + (AB' + A)C'$

⇒ $AB + AC'$ ⇒ $A(B + C')$

(13) Nested function:-

(Q) In the following simultaneous boolean ~~exp.~~ equation, what are the values of w, x, y, z.

$x + y + z = 1$
$xy + w'z' = 0$
$xw' + yz' = 1$

(a) 0001 (b) 1101
(c) 0101 (d) 1000.

⇒ Use substitution from options

(Q) $f(A, B) = A' + B$ then find $f(f(x+y, y), z)$.

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

⇒ $\overline{(x+y)} = f(x+y, y)$ (inner fun^n)

$f(\overline{x+y}, z) \Rightarrow \overline{(\overline{x+y})} + z$

$\Rightarrow x \cdot \bar{y} + z$

Spiral

Date. 10-02-19

(14) Nand gate & properties :-

→ Complement of AND.

A	B	$A \uparrow B$ (or $\overline{A \cdot B}$) → NAND
0	0	1
0	1	1
1	0	1
1	1	0

Properties :-

$\overline{A \cdot A} \neq A$

(i) Identity :- $\overline{A \cdot A} = \overline{A}$ does not follow identity.

(ii) Commutative :- $\overline{A \cdot B} = \overline{B \cdot A}$

(iii) Associativity :- $\overline{A \cdot (B \cdot C)} \neq (\overline{A \cdot B}) \cdot C$ ⇒ It is not Associative.

(15) (NOR gate & properties) :-

A	B	$A \downarrow B$ (or $\overline{A+B}$)
0	0	1
0	1	0
1	0	0
1	1	0

properties :-

(i) Idempotent :- $A \downarrow A = \overline{A}$ { so it is not idempotent }

(ii) Commutative :- $A \downarrow B = B \downarrow A$ { It is Comm. }

(iii) Associative :- $A \downarrow (B \downarrow C) \neq (A \downarrow B) \downarrow C$ { It is not associative }

Spiral

(16) EX-OR gate & properties:-

Date.....

→ It is also called modulo '2' sum.

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

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

ex:- $1 \oplus 1 \oplus 1 = 1$, $1 \oplus 0 \oplus 1 = 0$

Properties:-

(i) $A \oplus A = 0$ ⇒ (It is not idempotent)

(ii) Commutative:-

$A \oplus B = B \oplus A$ ⇒ {It is commutative}

(iii) Associativity:-

$A \oplus (B \oplus C) = (A \oplus B) \oplus C$ {It is associative}

Check it by putting 0 & 1 in place of 'A'.

→ $\overline{B \oplus C} = \bar{B} \oplus C = B \oplus \bar{C}$ {Draw Truth table to find out}

Date.....

(17.) EX-NOR & properties:-

→ It is negation of EX-OR.

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

A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

properties:-

(i) Idempotent:- $A \odot A = 1$ { It is not idempotent }

(ii) Commutative:- $A \odot B = B \odot A$ { It is Comm. }

(iii) Associative:- $A \odot (B \odot C) = (A \odot B) \odot C$ { It is Associative }

xxx

A=0 { LHS ⇒ $0 \odot (B \odot C) \Rightarrow \overline{(B \odot C)}$

RHS ⇒ $(0 \odot B) \odot C \Rightarrow \overline{B} \odot C$

⇒ LHS = RHS

A=1 { LHS ⇒ $1 \odot (B \odot C) \Rightarrow (\overline{B \odot C})$

RHS ⇒ $(1 \odot B) \odot C \Rightarrow B \odot C$

⇒ LHS = RHS

So, it is Assoc. since for both A=0 & A=1 LHS = RHS

xxx → $\overline{B \odot C} = \overline{B} \odot C = B \odot \overline{C}$

$B \oplus C = \overline{B} \oplus C = B \oplus \overline{C}$

This is the way to fastly check whether L.H.S = R.H.S or not

Spirat

(18) Properties of EX-OR & EX-NOR:-

Date.....

→ ⊕ = 1 {for odd no. of 1's}
→ ⊙ = 1 {for even no. of 0's}

A	B	C	A⊕B⊕C	A⊙B⊙C
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

for three var.:

A⊕B⊕C = A⊙B⊙C

Since, when no. of 1's are odd then no. of 0's will be even & vice versa.

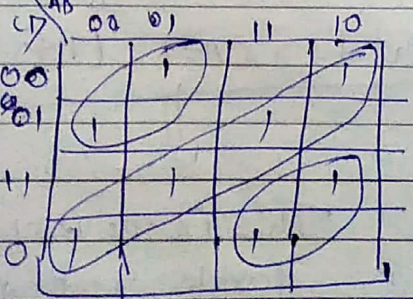
(because total no. of i/p are odd.)

→ In case even no. of i/ps ⇒ ⊕ = ⊙ ⇒ (complement of each other)

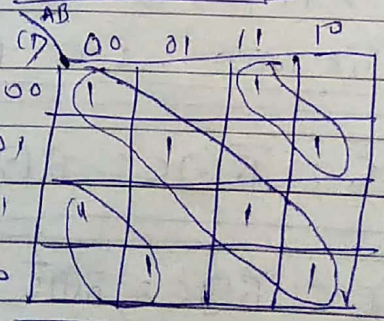
→ In case of odd no. of i/ps ⇒ ⊕ = ⊙ ⇒ (same)

→ In K-map's of EX-OR or EX-NOR 1's are only present at diagonals (ie. diagonal adjacency) & no. of min-terms = max-terms (ie. neutral funcns)

ex:- A⊕B⊕C⊕D :-



A⊙B⊙C⊙D :-



⇒ 1's at where no. of 0's are even

No. of min-terms = max-terms

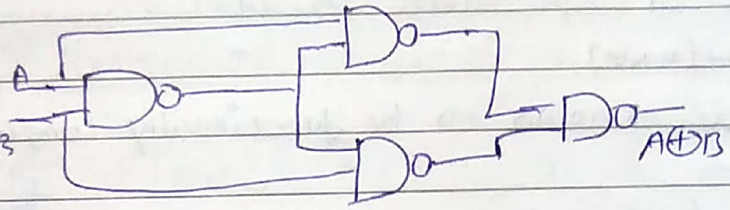
Date.....

(19) Minimum no. of gates req. for EX-OR & EX-NOR :-

(Q) Minⁿ no. of NAND & NOR gates req. to realise EX-OR and EX-NOR gates :-

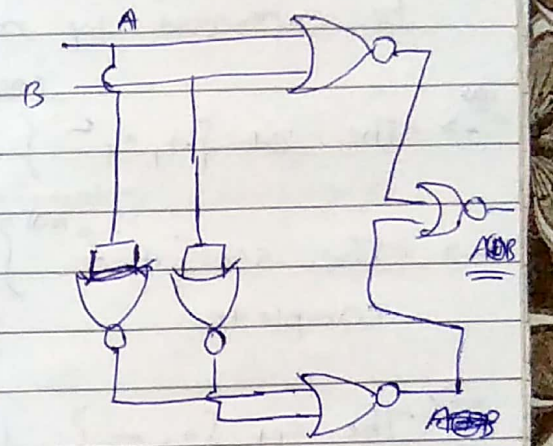
(Ans) EX-OR :-

(i) Using NAND :-



4 - NAND

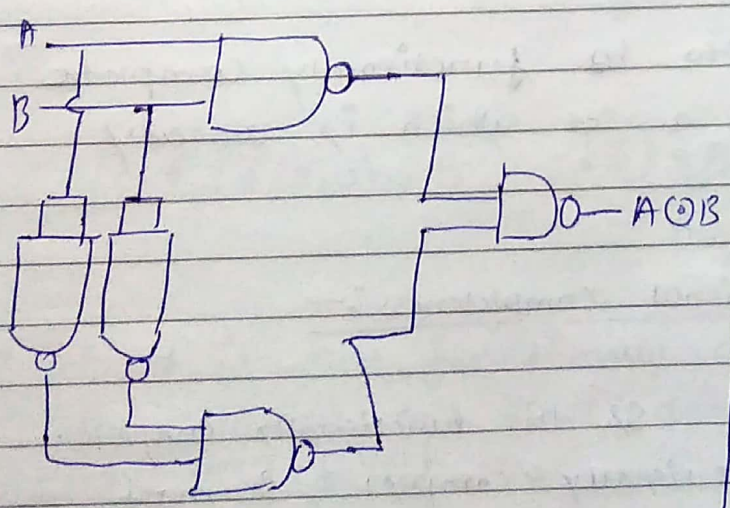
(ii) Using NOR :-



4 - NOR

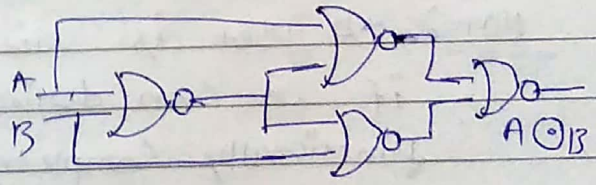
EX-NOR :-

(i) Using NAND :-



5 - NAND

(ii) Using NOR :-



4 - NOR

Date.....

(20) Functionally completeness :-

→ Manufacturers of circuits or gates try to make functionally complete circuits, because it by using those All the other functionality can be realised.

Functionally Complete opern :-

→ A set of operations said to be functionally complete (or universal) iff every switching funcn can be expressed by means of operations in it.

→ The set $\{+, \cdot, \overset{\text{complement (or NOT)}}{'}\}$ are known to be functionally complete.

→ The set $\{+, \overset{\text{NOT}}{-}\}$ (ie. NOR) is said to be functionally complete.

→ The set $\{\cdot, -\}$ (ie. NAND) is also functionally complete.

→ So, using NAND & NOR all the circuits can be designed.

Note:- A set is said to be functionally complete if we can derive a set which is already functionally complete.

(21) Example 1 on functional completeness :-

ex:- $f(A,B,C) = A' + BC'$ is this functionally complete.

⇒ for this to be functionally complete, it must derive (NOT) & either 'OR' or 'AND'.

⇒ $\boxed{f(A,A,A) = \overline{A}}$ (replacing B & C with A)

we got NOT

→ To get Complement or NOT by replacing all var. with one var. for each var. if NOT is obtained then proceed else it is not functionally complete (since NOT can't be obtained)

Date.....

Now, it seems deriving 'OR' is easy.

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

$f(A,A,A) \rightarrow \bar{A}$ (~~A~~)

$f(B,B,B) \rightarrow B'$

$\Rightarrow \boxed{A+B}$ ⇒ So, OR is derived.

→ So, 'f' is functionally complete. ~~(OR~~

(22) Example 2 on functional completeness :-

Ex:- $f(A,B) = \bar{A} + B$ Is it functionally complete?

⇒ $f(A,A) = \bar{A}$, $f(B,B) = 1$

~~$f(A,\bar{A})$~~

NOT is ~~not~~ not obtained (Not functionally complete)

$f(A,0) = \bar{A}$ { So, only 'f' is not enough to get 'NOT', it need '0'. ie. $(f,0) \Rightarrow$ NOT }

$f(f(A,0), B) = (\bar{A}) + B$

$\rightarrow A+B$ (OR)

→ So, it is not functionally complete because it needs the support of '0' to derive NOT. (It is partially functional comp).

~~xxx~~ Partially functionally complete :- A func^n which takes support ~~for~~ from '0' or '1' is called partially func. comp.

→ Complement form of a var can't be taken as i/p.
ex:- $f(\bar{A}, B)$

Date.....

Function

- functionally Complete
- partially functionally Complete
- Not functionally complete

(23) Example 3 on functionally Completeness :-

ex:- $f(A,B) = \bar{A}B$ is it func. Complete?

⇒ $f(A,A) = 0$, $f(B,B) = 0$

→ so it's not functionally complete directly

$f(A,1) = \bar{A}$ (NOT is obtained by the help of '1')

$f(f(A,1), B) = AB$ ('AND' is obtained)

So, this func^n is partially functionally complete

(29) Example 4 on functionally Completeness :-

ex: $f(A,B,C) = AB + BC + CA$

⇒ $f(A,A,A) = A$, $f(B,B,B) = B$, $f(C,C,C) = C$

⇒ This func^n can't be derived ~~from the ex~~ complement (or NOT) since complement is not present in the exp.

So, it is not functionally Complete.

Spiral

(25) Example 5 on functional completeness:

Date.....

ex: $f(x,y) = \bar{x}y + x\bar{y}$ (EX-OR)

$\Rightarrow f(x,x) = 0$, $f(y,y) = 0$ } $\Rightarrow$ Not directly functionally complete

$f(x,1) = \bar{x}$ { NOT is obtained with the help of '1' }.

$f(x,y) = xy + \bar{x}\bar{y}$
$f(x,\bar{y}) = \bar{x}\bar{y} + xy$
$f(\bar{x},\bar{y}) = x\bar{y} + \bar{x}y$
} we are not obtaining either 'AND' or 'OR'.

→ So, this func^n is not functionally complete (ie. EX-OR)

(26) Example 6 on functional Completeness:-

(Q) Any Boolean func^n can be defined with which of the following Opern.

(a) ⊕, NOT (b) (+), 1, OR (c) +, 1, NOT (d) ⊙, 1, NOT.

(+), 1 ⇒ NOT

⇒ By elliminating one var. by '1' or '0' we can only get 'NOT'. (because that will be only in one var.) but not 'OR' or 'AND'

ex: (+), 1 ⟹ We can't get OR or AND

⇒ (c) $f(x,y) = \bar{x}\bar{y} + xy$

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

$f(x,0) = \bar{x}$ (Using '0' we got NOT)

(⊙, 0) ⇒ NOT

Spiral

Date.....

$f(x,y) = \bar{x}y + x\bar{y}$, $f(x,\bar{y}) = \bar{x}y + \bar{y}\bar{x}$

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

→ so, it is also not functionally complete

→ EX-OR & EX-NOR are not even partially functionally complete.

(27) Self dual functions:- (If func^n & its dual are same)

→ A boolean func^n is self-dual if:-

(i) It is neutral (no. of minterms = maxterms)

(ii) The func^n does not contain two mutually excl. term.

ex:- $ABC \xrightarrow{\text{M.E term}} \bar{A}\bar{B}\bar{C}$

$AB'C \longrightarrow A'BC'$

ex:- $f(A,B,C) = AB + BC + CA$

$f_d(A,B,C) = (A+B)\cdot(B+C)\cdot(C+A) \Rightarrow AB + BC + CA$

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

$\Rightarrow ABC + ABC' + A'BC + AB'C$ } → no. of minterm = 4

($\bar{A}\bar{B}\bar{C}$ (M.E term), $\bar{A}\bar{B}C$, $A\bar{B}\bar{C}$, $\bar{A}BC$) → these are not in func^n

→ Total no. of minterms in self dual func^n = 2^{n-1}

(28) No. of self dual functions :- Date.....

ex:- for 3 variables :-

	A	B	C	f
0	0	0	0	
1	0	0	1	
2	0	1	0	
3	0	1	1	
4	1	0	0	
5	1	0	1	
6	1	1	0	
7	1	1	1	

M.E terms pairs => (0,7), (1,6), (2,5), (3,4)

total self dual func^n = 2 × 2 × 2 × 2

$\Rightarrow 2^4 = 16 \left(2^{2^{(3-1)}} = 2^4\right)$

$= \frac{2}{2}$

$\Rightarrow$ for 'n' variables total pairs possible $= (2^{n-1})$ & each pair has '2' choices

Total self dual func^{n}s $= 2^{2^{(n-1)}}$

(29) Self-dual functions are closed under Complementation :-

(Q) Which of the functions are self dual.

(i) $f(A,B,C) = \Sigma(0,2,3)$

(ii) $f(A,B,C) = \Sigma(0,1,6,7)$

(iii) $f(A,B,C) = \Sigma(0,1,2,4)$

(iv) $f(A,B,C) = \Sigma(3,5,6,7)$

$\Rightarrow$ Since all are 3 var. func^n so each self dual func^n should have atleast $\frac{2^n}{2}$ minterms (i.e. 4.)

(0,7) (1,6) (2,5) (3,4)

ME pair

(i) => no. of minterms are 3

(ii) => 1 & 6 are present.

~~(iii) &~~

(iii) & (iv) are complements of each other.

Date.....

→ Duality is closed under complementation.

(30) Introduction to electronic gates:-

Electronic gate NIUs:-

(i) Electronic gates generally recieve voltages as inputs & produces voltages as o/p.

(ii) The precise values of these voltages are not significant towards determination of logical operⁿ of gates.

(iii) The significant point is that voltages are restricted to two ranges of values, high & low.

(iv) Thus two valued variables may be used to represent these voltages.

(v) If we ~~assign~~ associate constant '1' with high voltage & '0' with low voltage, it is called +ve logic system.

(vi) If we associate '1' to low voltage & '0' to high voltage it is called -ve logic system.

→ By convention +ve logic system is used

Date.....

(31) positive & -ve logic systems:-

(Q) A gate which o/ps high voltage only when all i/ps are high & o/ps low voltage otherwise. How does this gate behaves in +ve logic & -ve logic.

⇒

A	B	A OR B	A ⊙ B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

→ (it behaves as OR gate) → -ve logic syst.

↳ in +ve logic syst. (it behaves as AND gate)

***

⇒ +ve & -ve logic systems are duals of each other.

NOT Complements **

(32) Gate 2016 question on boolean algebra:-

(Q) Let $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$ where x_1, x_2, x_3, x_4 are boolean var. and $\oplus$ which is true (always)?

(a) $x_1 x_2 x_3 x_4 = 0$ (b) $x_1 x_3 + x_2 = 0$

(c) $\bar{x}_1 \oplus \bar{x}_3 = \bar{x}_2 \oplus \bar{x}_4$ (d) $x_1 + x_2 + x_3 + x_4 = 0$

⇒ the exp. will have either 0 → 1's, 2 → 1's, 4 → 1's

***

$$\bar{x}_1 \oplus \bar{x}_3 = x_1 \oplus x_3$$

↓↓

(try with all 3 Comb^n)

(33) Gate 2016 question on gray code

Date.....

Func^n :-

(Q) Consider nos. represented in 4-bit gray code. Let $h_3 h_2 h_1 h_0$ be the gray code representation of a no. 'n' & $g_3 g_2 g_1 g_0$ be the " " of $(n+1)$ modulo 16 value of the no. which of the following func^n is correct?

(a) $g_0(h_3 h_2 h_1 h_0) = \Sigma(1,2,3,6,10,13,14,15)$

(b) $g_1(h_3 h_2 h_1 h_0) = \Sigma(9,9,10,11,12,13,14,15)$

(c) $g_2(h_3 h_2 h_1 h_0) = \Sigma(2,4,5,6,7,12,13,15)$

(d) $g_3(h_3 h_2 h_1 h_0) = \Sigma(0,1,6,7,10,11,12,13)$

2. Minimization

Date.....

(1.) Introduction to minimization of Boolean exp:-

→ While simplifying a switching $f(x_1, x_2 \cdots x_n)$, our aim is to find an expression $g(x_1, x_2 \cdots x_n)$ which is equiv. to 'f', which minimizes some cost criteria.

Criteria to determine minimal cost :-

(i) Minm no. of appearences of literals.
(ii) Minm no. of literals in sop or pos expression.
(iii) Minm no. of terms in sop expression, provided ~~that~~ there is no other such expression with the same no. of terms & fewer literals.

(2) Irredundant or irreducible expressions :-

ex:- $f(x,y,z) = x'yz' + x'y'z' + xy'z' + x'yz + xyz + xy'z$

method-1 ⇒ ~~x'yz'~~ $x'z' + xy'z' + yz + xy'z$.

$\Rightarrow (x' + xy')z' + (y + xy')z$

$\Rightarrow ((x'+x)(x'+y'))z' + (y+y')(y+x)z$ (distributing '+' over '.')

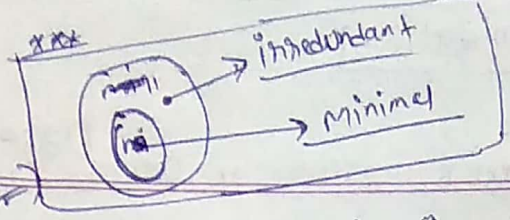
$\Rightarrow \boxed{x'z' + y'z' + yz + xz.}$ ⇒ Irredundant exp. or irreducible

method-2

$\Rightarrow$ ~~xyz~~ $\boxed{x'z' + xy' + yz}$ ⇒ by combining each two terms

$\Downarrow$

Irredundant or irreducible = (can't be simplified further)



Date.....

⇒ Irreducible exp. does not implies that it is minimal (as done in method 1).

⇒ There could be many minimal forms for any expression.

Irredundant or Irreducible expression:-

→ An sop expression from which no term or literal can be deleted w/o altering its logical value.

(3) K-Map introduction:-

→ So, to avoid hectic algebraic computation, K-map is used for simplification.

→ The map method a systmetic method for combining terms & deriving minimal exp.

→ A K-map is modified form of a truth table in which the arrangement of comb^n is particulary convinient for minimization.

→ Every 'n' var. map consists '2^n' cells representing all possible comb^n of variable (ie. minterms)

→ The func^n value associated with a particular comb^n is entered in the corresponding cell.

⇒ Cycle code is used in the comb^n as column & row heading. (ie. only one bit changes b/w two adjacent cells)

⇓

Cyclic codes

Date.....

→ Because of these codes, two cells which have a common side correspond to combinations that differ by the value of just a single variable.

→ These two cells play a major rule in simplification process. Because they can be combined by means of rule $AA + AA' = A$

A 4 variable K-map:-

yz \ wx	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

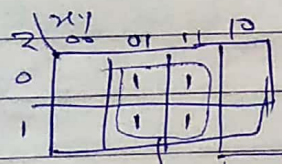
$f(w, x, y, z)$

(4) K-Map Simplifications:-

Simplifications & minimization of functions using K-Map:-

(1) A collection of 2^m (always power of 2) cells, each adjacent to 'm' cells of collection is called a subcube, & the subcube is said to cover these cells.

ex:-



subcube (collection of 2^2 cells) & each cell is adjacent to '2' cells

(2) Each subcube can be expressed by a product containing 'n-m' literals, where 'n' is no. of var. on which func^n depends.

Date.....

→ By doing this (ie. forming the subcubes) the no. of literal is reduced by 'm'.

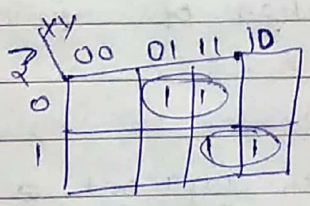
⇒ *** Any cell can be included in as many subcubes as desired.

→ A func^n 'f' can be expressed as sum of those product terms, which corresponds to subcubes nece necessary to cover all its '1' cells.

*** → The no. of product terms in the Expression, for 'f' is equal to the no. of subcubes, while the no. of literals in each term is determined by size of corresponding subcubes.

→ Therefore to obtain a minimal expression, we must cover all '1' cells with min^m no. of subcubes such that each subcubes are as large as possible.

Ex:-

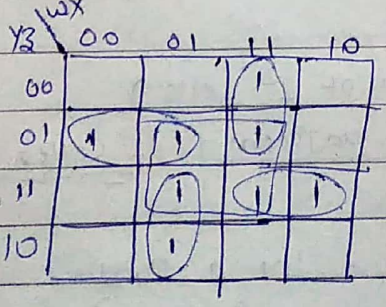


Largest subcube ⇒ size 2

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

(5) Examples on K-map:-

$f(w,x,y,z) = \Sigma(1,5,6,7,11,12,13,15)$



→ All the blank cells are 0's & are counted for ~~sum prod~~ pos form

$\bar{w}\bar{y}\bar{z} + w\bar{x}\bar{y} + y\bar{w}x + wxz + xz$

xz is not req.

(6) Covering functions:-

Date.....

~~Implicants:~~

→ A switching func^n $f(x_1, x_2 \dots x_n)$ is said to cover $g(x_1, x_2 \dots x_n)$ denoted by "f is superset of g" if 'f' assumes true value whenever 'g' does.

ex:-

A B	g	f
0 0	0	1
0 1	1	1
1 0	0	0
1 1	0	0

f g

$\{0,1\} \supseteq \{1\}$

(1's at 01)

→ If "f' covers 'g'" & simultaneously "g covers f" then f & g are equiv.

ex:- How many func^{n}s are possible which covers 'g'

ex:-

A B	g	f
0 0	0	0,1 → 2
0 1	1	1 → X
1 0	0	0,1 → 2
1 1	0	0,1 → 2

Two possibility for each except at 1

⇒ Total = 8 func^{n}s are possible which can cover g.

→ If 'g' has 'x' min terms & 'g' is a func^n of 'n' variables Then no. of functions covering 'g' is $\boxed{2^{(2^n - x)}}$

ex:- $f(w, x, y, z) = wx + yz$ then how many functions are covering f.

Date 11/02/19

⇒ No. of minters possible for $f = 2^4 = 16$

$f(w,x,y,z) = 11(--) + --11$

$= 1100 + 1101 + 1101 + 1111 + 0011 + 0111 + 1011 + 1111$ (repeated)

⇒ So, total 7 minterms

'f' is containing max-terms $= 16 - 7 = 9$

So, total func^{n}s covering $f = 2^9$ (No. of powerset)

(7) Implicants & prime implicants :-

→ If "f covers g" then 'g' is said to imply 'f'.

$(g \to f)$ (Not $f \to g$) (or 'g' is an implicant of 'f')

ex:- (Whenever 'g' is true 'f' is also true)

ex:- $f(a,b,c) = ab + c$

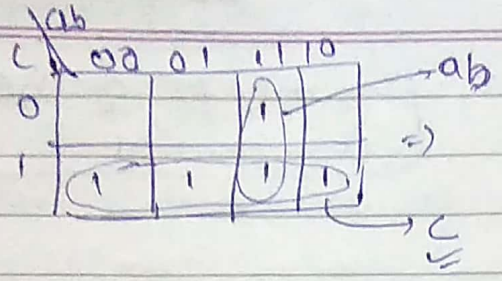
$f_1 \quad f_2$

a	b	c	f_1 = ab	f_2 = c	f_3 = ab+c
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

⇒ $f \supseteq (f_1 \& f_2)$

So, f_1 & f_2 are implicants of f

Date.....



$f = ab + c$

So, every subcube is an implicant.

prime implicant :-

→ An implicant 'p' of a func^n 'f' is said to be prime implicant if

(i) 'p' is a product term (ie. subcube)

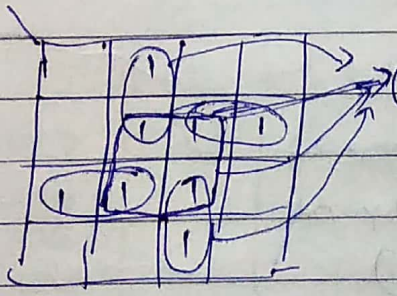
(ii) Deletion of any literal from 'p' results in a new product which is not covered by 'f' (ie. a subcube cannot be part of any other subcube).

→ A subcube is called essential prime implicant if it is not a part of any other prime implicant.

(8) Essential prime implicant :-

→ A prime implicant 'p' of a func^n 'f' is said to be an essential prime implicant, if it covers atleast one min term of 'f' which is not covered by any other prime implicand.

ex :-



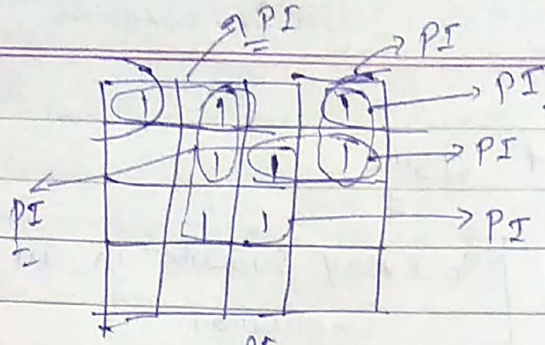
PI (1 of size 4) & (4 of size 2)

4 of size '2' are EPI

for minimizing only use EPI.

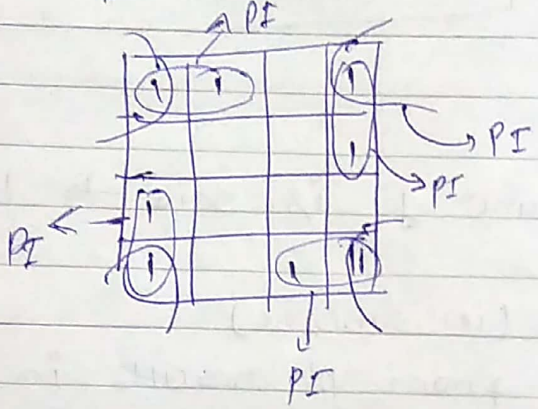
Date.....

ex:-



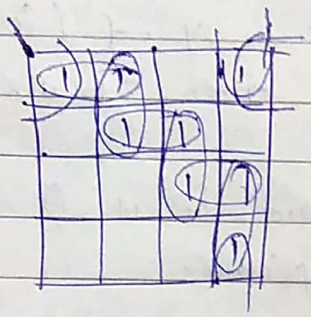
So, only PI of size '4' is essential.

ex:-



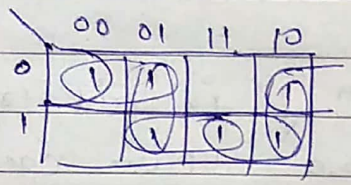
(All 4 PI of size '2' are EPI)

***
Ex:-



→ (All are PI but no EPI)

ex:-



(All are PI but no EPI)

***
→ last two examples are ~~called~~ examples of cyclic K-maps.

⇓

(If all PI are of same size ~~(atleast 2)~~ & each PI ~~are~~ are covered by at least '2' PI & no EPI)

Date... 12-02-2019

(9) procedure for obtaining minimal sop:-

(i) Determine all EPI & include them in the minimal sop

(ii) Remove from list of PI all those which are covered by EPI

(iii) If the set determined in step 2 covers all the minterms of 'f' then it is unique minimal expression. Otherwise select the additional prime implicants, so that the func^n 'f' is covered completely & the total no. & size of PI added are minimal.

→ Sometimes when the question is complex directly finding out ~~essential PI & min~~ minimal sop is bit difficult so, we'll use PI-table for that. (It will also give us no. of minimal sop.)

Ex:-

YZ \ WX	00	01	11	10
00			1	
01	1	1	1	
11		1	1	1
10		1		

$f = \Sigma(1, 5, 6, 7, 11, 12, 13, 15)$

PI-table:-

The minterms which are covered by corresponding PI

All the minterms

	1	5	6	7	11	12	13	15
$\bar{W}\bar{Y}Z$	(1)	1						
$WX\bar{Y}$						1	1	
WYZ					1			1
$\bar{W}XY$			1	1				
XY		1	1	1			1	1

All the Prime implicants

XY is not EPI & all the minterms are covered by EPI

So, $f = \bar{W}\bar{Y}Z + WX\bar{Y} + WYZ + \bar{W}XY$ (No + XY)

Spiral

Date.....

→ After filling out the values in PI table go to each column (ie. each minterm) & if, a PI is present in only one row, then that PI is essential & the minterms which are part of that PI (which is essential need not be checked)

→ Once all the minterms are covered then stop the procedure & the remaining PI are not essential, & those PI which are found to be essential are the only EPI.

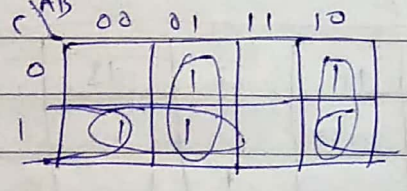
→ Even after including all EPI if some minterms are left then we need to take the PI to cover those minterms (in this case we can get more than one minimal SOP, depending on what we're choosing)

(Note:- Only use this if very neccessary)

(10) Minimal SOP example:-

(Q) Which of the following PI are essential

(a) $B'C$, $A'B$, (b) $A'C$, $A'B$ (c) $A'B$, AB' (d) $A'B$, AB', $B'C$



⇒ total 4 PI

⇒ $\bar{A}B$ & $A\bar{B}$ are two EPI. (only these two cover the minterms which are not covered by any other PI).

Date.....

(ii) How many minimal SOPs are possible? {in previous ex}

⇒

	1	2	3	4	5
$\bar{A}B$		11	1		
$A\bar{B}$				1	1
$\bar{A}C$	1	•	1	•	
$\bar{B}C$	1	•			1

{even though for this question PI-table is not req. & but we are doing it}

→ ~~Mintams ⇒ 2,3,4,5 are covered only by~~

⇒ Only mintams 2,3,4,5 are covered by EPI & not 1

⇒ So, 1 can be covered by $\bar{A}C$ or $\bar{B}C$, ~~so to~~

=

Thus, total 2 minimal sop are possible

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

(II) Minimal POS:-

→ for finding minimal pos make subcubes of 0's.

→ While writing the expression for ~~pos~~ PI of pos. (0 for 0's. ~~use~~ use true form & for 1's use complemented form).

ex:- $f(w,x,y,z) = \Sigma(5,6,9,10)$ find minimal sop

yz \ wx	00	01	11	10
00	0	0	0	0
01	0	1	0	1
11	0	0	0	0
10	0	1	0	1

all PI are essential

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

Unique pos

$sop = wx\bar{y}\bar{z} + \bar{w}xy\bar{z} + w\bar{x}yz + \bar{w}\bar{x}\bar{y}z$

Spiral

Date.....

Complement of a func^n:- Replace 0's with 1's & vice versa in K-map to get the complement of a func^n.

(or)

→ Convert Min-terms to max terms

(Q) for previous example Minimal sop & pos are:-

$F' = \Pi(0,5,6,9,10)$

$sop = \bar{y}\bar{z} + \bar{w}\bar{x} + wx + yz$

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

→ for complementation, for pos complement → sop

for sop complement → pos

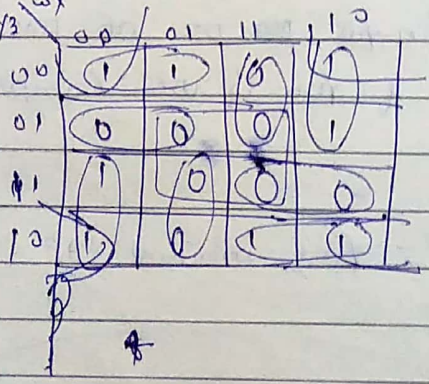
(Changing + to ·)

(12) Examples on minimal pos:-

ex:- find the no. of literals in min. pos & sop for

$f(w,x,y,z) = \Pi(1,5,6,7,11,12,13,15)$

=>



Minimal pos =>

4 EPI. (out of 5)

$(w+y+\bar{z})(\bar{w}+\bar{x}+y)(w+\bar{x}+\bar{y})(\bar{w}+\bar{y}+\bar{z})$

no. of literals = 12

Minimal sop:

→ Only 4 EPI (of size 2) | the subcube of size 4 is not a EPI)

Date.....

Minimal SOP :- $\bar{w}\bar{y}\bar{z} + w\bar{x}\bar{y} + \bar{w}\bar{x}y + wy\bar{z}$

no. of literals = 12

(13) Introduction to Don't Cares :-

→ for converting BCD to Excess 3, we need only numbers 0 to 9 (i/p) & o/p (3 to 12) in Excess 3.

(Because the i/p is in Decimal no. system)

→ for BCD 10 to 15 (since 4 bits are used for BCD) we DON'T CARE. (Don't cares are used in K-map)

ex:- $w = \Sigma(5,6,7,8,9) + \phi(10,11,12,13,14,15)$

Don't care combinations :-

→ A func^n is said to be completely specified if it is given 0 (or) 1 for every combination of i/p variables. There exists some functions which are not completely specified.

→ Combinations for which the value of a func^n is not specified are called Don't Care combinations

→ The value of a func^n for such comb^n is donated by 'ϕ' or 'd'.

Date.....

→ Since each don't care comb^n represents two values {0,1}, an incompletely specified func^n containing k-don't care comb^{n}s corresponds to a class of 2^k distinct functions. (don't cares can be 0's or 1's depending on situation)

ex:-

a	b	f
0	0	1
0	1	1
1	0	∅
1	1	∅

⇒ The single func^n 'f' can represent a class of $2^2 = 4$ functions

f_1	f_2	f_3	f_4
1	1	1	1
1	1	1	1
0	0	∅	1
0	1	0	1

⇒ (Representation of f_4 is easy)

→ Our task is to chose a func^n having minimal representation out of these 2^k functions

→ We could assign a '0' or '1' to a don't care comb^n in such a way to increase or decrease size of a subcube.

→ Don't cares are changed to '1' to ↑ the size of subcube (only if it is possible) in case of minterms but it does not form a larger subcube then leave don't cares (∅) as it is.

ex:-

1	1
∅	1

only make this '∅' as '1'

∅ → don't make this ∅ as '1'

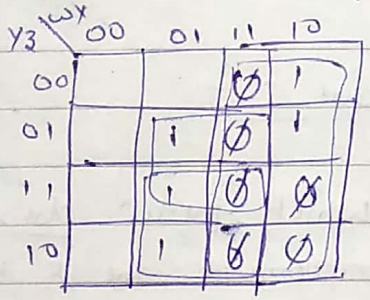
Spiral

Date.....

*** → There should not be any subcube which only consists of don't cares (∅). (because don't care need not be implemented)

(14) Examples on don't care - set 1 :-

ex:- $w = \Sigma(5,6,7,8,9) + \emptyset(10,11,12,13,14,15)$

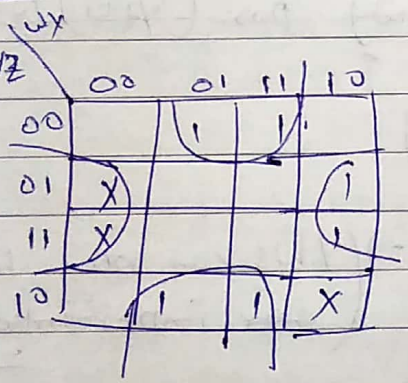


⇒ (2^6 class of functions are possible.)

$W = (w + xz + xy)$

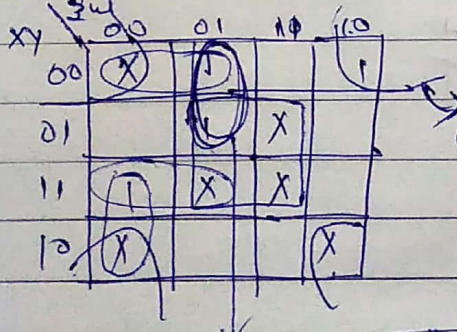
(16) Examples on don't care ⇒ Set 2.2 :-

ex:-



$f = x\bar{z} + \bar{x}z$

*** (17) Examples on don't care :- set 2.3 :-



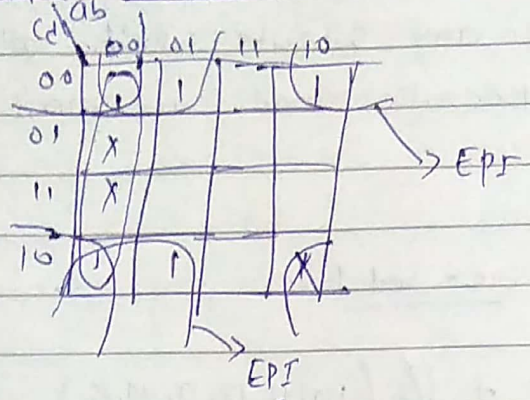
$f = \bar{w}\bar{y} + \bar{z}w\bar{x} + \bar{z}\bar{w}x$

Spiral *** This one will be used because it requires 3 literals while if we 4-sized subcube then to cover remaing 1 we'll need 3 more literals (total 5)

Date.....

(18) Examples on don't care set 2 :-

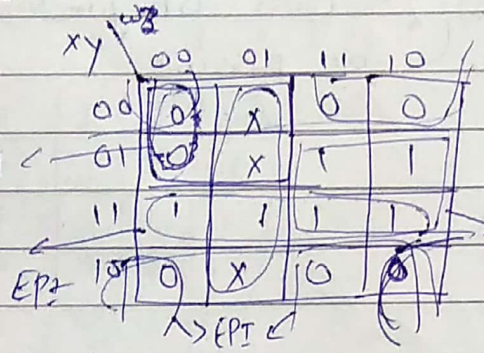
ex:-



=> EPIs are able to cover all the 1's (minterms)

$f = \bar{a}\bar{d} + \bar{b}\bar{d}$

(20) Examples on don't care set 3 :-



(Q) Which func^ doesn't implement the K-map

(a) $(w+x)y$ (b) $xy+yw$

(c) $(w+x)(\bar{w}+y)$ $(\bar{x}+y)$ (d) none

=> SOP = $xy + yw \Rightarrow (w+x)y$

~~pos = $(y+\bar{w})(\bar{x}+y)(x+w)$~~ pos = $(y+\bar{w})(\bar{x}+y)(x+w)$

pos = $y(w+x)$ (minimal)

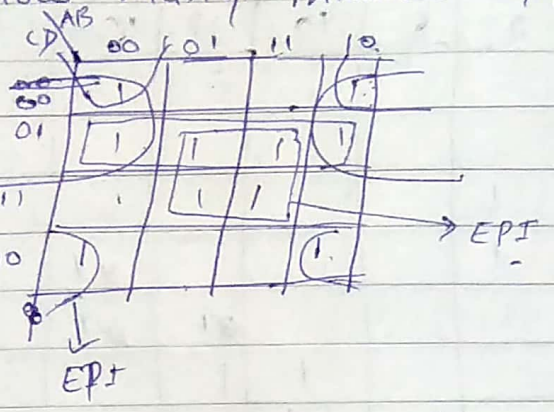
pos = $(w+x)(\bar{w}+y)(\bar{x}+y)$ => (This can also be ~~done~~ implemented)

-> They are not asking about minimal, its just whether it can be implemented or not.

Date.....

(21) Finding minimal Expressions:-

(Q) How many minimal exp. are possible?



Draw PI chart:-

	0	1	2	5	7	8	9	10	13	15
EPI → $B\bar{D}$	1	1	1		1	1	1	1		
$B\bar{C}$	1	1				1	1			
$\bar{C}D$		1		1	1		1		1	1
BD ← EPI				1	1				1	1

(only 1 & 9 are not covered by EPI)

$$f = B\bar{D} + BD + B\bar{C}$$

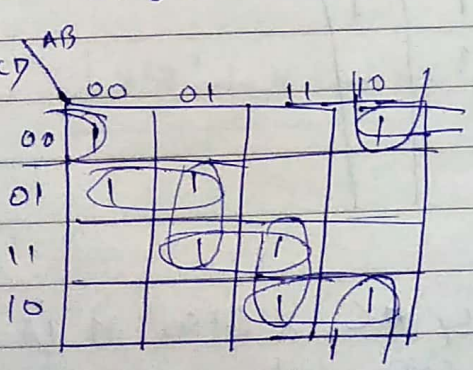
or

$$f = B\bar{D} + BD + \bar{C}D$$

} 2 minimal exp. are possible

(22) Branching technique for minimizing cyclic functions:-

ex:-



⇒ It is cyclic function

⇒ (all prime implicants are of same size & each min-term is covered by two PI)

→ There is no EPI.

Date.....

PI chart :-

	0	1	5	7	8	10	14	15
A = w'x'y'	1	1						
B = w'y'z'		1	1					
C = w'xz			1	1				
D = xyz				1				1
E = wxy							1	1
f = wyz'						1	1	
G = wx'z'					1	1		
H = x'y'z'	1				1			

→ Since each column contains two entries & no EPI, so chose any one column & include in minimal exp. & Reduce the table (by Removing that Row & Corresponding column)

Step-1 :- Include 'A' in minimal sop & ~~delete the~~ Reduce the table

	5	7	8	10	14	15
B = w'x'z'	1					
C = w'xz	1	1				
D = xyz		1				1
E = wxy					1	1
f = wyz'				1	1	
G = wx'z'			1	1		
H = x'y'z'			1			

MSOP = A + ...

NOW, B is completely covered by C & also H is covered by 'G' i.e. (B → C & H → G), so B & H can be removed.

smaller → H, larger → G

→ B & H are removed because a smaller implicant which is a part of bigger PI & thus the smaller implicant can't be a part of PI.

Spiral

Date.....

Step-2:-

	5 ✓	7 ✓	8 ✓	10 ✓	14	15
C = w'xz	1	1				
D = xyz		1				1
E = wxy					1	1
F = wyz'				1	1	
G = wxz'			1	1		

NOW, for this table 'C' & G are ~~Prime~~ EPI.

So,

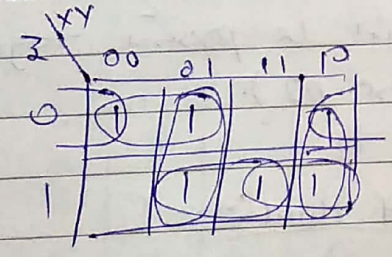
MSOP = A+C+G -

NOW, Minterms 14 & 15 are covered by 'E' completely (They are also covered by D & F but they will not be included because for that two PI will & needed thus ↑ no. of literals)

So, I find MSOP = A+C+G+E

$$\Rightarrow \boxed{w'x'y' + w'xz' + wx'z' + wxy} \quad **$$

ex:-



⇒ cyclic func^n
(do it by your own while revising)

⇒ Ans = $\boxed{x'z' + yz + xy}$ { by first removing x'z' PI }

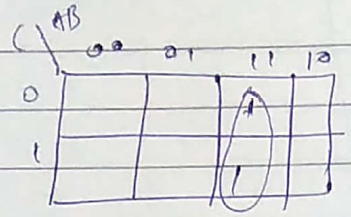
Date.....

(23) Implicant & PI diff:-

ex:- Which of the following can be a PI of a func^n $f(A,B,C)$

(a) $ab+c$ (b) $bc+a$ (c) $ac+b$ (d) ab

=>



(all other are comb^n of two PI)

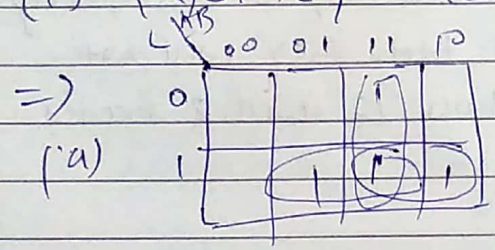
=

ex:- which of the following cannot represent PI set of $f(A,B,C)$.

(a) $\{ab, bc, ca\}$ (b) $\{bc'a, ba, bc\}$

(c) $\{a, b, ac\}$ (d) $\{a'b, ab\}$

=

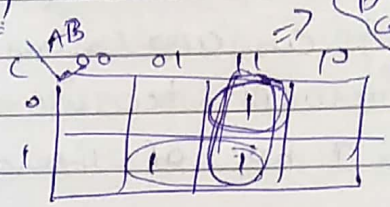


(a)

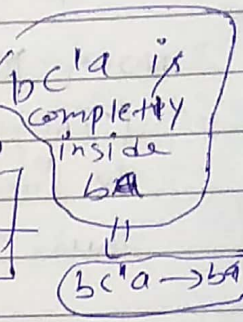
=>

(b)

X



=>



(Subcube inside a subcube can't be PI)

xxx

=> A' bigger set & its subset can't be present together in PI set. (ie. both can't be PI).

(29) Converting a function into self dual:-

(Q) what min terms have to be added to make the following func^n a self dual?

$f(A,B,C,D) = A'BC + (AC+B)D'$

=> Mutually exclusive min terms :- (0,15), (1,14), (2,13), (3,12), (4,11), (5,10), (6,9), (7,8).

$\Rightarrow A'BC(D+D') + A(B+B')CD + (A+A')B(C+C')D'$

Spiral $A'BCD + A'BCD' + ABCD + AB'CD + A'BC'D + ABC'D'$

7 6 15 11 5 13

Date..13-02-19

→ Any one of (1,14) or (3,12) can be added to make it self dual.

(A'B'C'D' or ABCD') & (A'B'CD or ABC'D')

Any comb^n out of 4 comb^n possible

(25) Combining functions having don't cares :-

(Q) How many functions does $f_1 \cdot f_2$ & $f_1 + f_2$ represent?

$f_1(a,b,c) = \Sigma(0,2,4) + \Sigma\phi(3,5,7)$

$f_2(a,b,c) = \Sigma(2,3) + \Sigma\phi(1,6,7)$

=)

The funcn's are of 3-var. so total possible comb^{n}s are 8

	f_1	f_2	$f_1 \cdot f_2$	$f_1 + f_2$
0	1	0	0	1
1	0	ϕ	0	ϕ
2	1	1	1	1
3	ϕ	1	ϕ	1
4	1	0	0	1
5	ϕ	0	0	ϕ
6	0	ϕ	0	ϕ
7	ϕ	ϕ	ϕ	ϕ

2^2 functions

2^4 functions

$1+\phi = 1$	$0+\phi = \phi$
$\phi \cdot 1 = \phi$	$0 \cdot \phi = 0$

Date.....

ex:- How many functions does $f_1 \cdot f_2$ & $f_1 + f_2$ represent?

$f_1(w,x,y,z) = \Sigma(1,3,9,5,9,10,11) + \Sigma_\phi(6,8)$

$f_2(w,x,y,z) = \Sigma(0,4,9,7,8,15) + \Sigma_\phi(9,12)$

	f_1	f_2	$f_1 \cdot f_2$	$f_1 + f_2$
0	0	1	0	1
1	1	0	0	1
2	0	1	0	1
3	1	0	0	1
4	1	1	1	1
5	1	0	0	1
6	ϕ	0	0	ϕ
7	0	1	0	1
8	ϕ	1	ϕ	1
9	1	ϕ	ϕ	1
10	1	0	0	1
11	1	0	0	1
12	0	ϕ	0	ϕ
13	0	0	0	0
14	0	0	0	0
15	0	1	0	1

2^1 functions — 2^2 functions

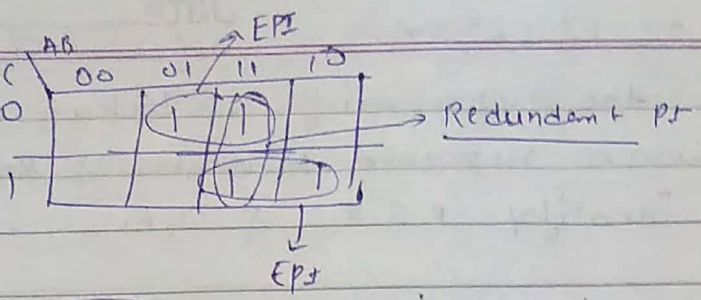
2^2 functions

(26) Prime implicants & Don't cares:-

(Q) The no. of PI, EPI & redundant PI for the func^n $f(A,B,C) = \Sigma(2,5,6,7)$

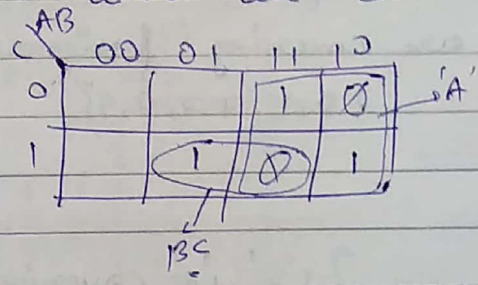
Redundant PI :- The PI which are covered by other EPI

Date.....



PI → 3
Red. PI → 1
EPI => 2

ex:- A function $f(A,B,C) = \Sigma(3,5,6)$ is minized to $(A+BC)$ then what are don't cares.



=> Don't cares are :- {4, 7}

(27) Number of minimal expressions :-

(Q) In a K-map it was found out that EPI are covering all terms except 2 min terms. Those two min terms are in turn covered by 3 non-essential PI each. What is the no. of minimal SOP expressions.

=) For each min term 3 choices are available.

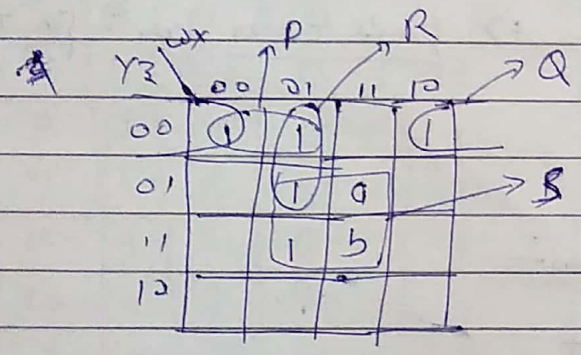
So, no. of minimal sop exp = 3x3 => 9

(28) Beautiful question on PI - chart :- Date.....

(Q) In a PI chart representing a boolean exp. $f(w,x,y,z)$, column represent Min-terms & Row represent PI, identify P, Q, R, S & a, b.

	0	4	5	7	8	a	b
P	✓	✓					
Q	✓				✓		
R		✓	✓				
S			✓	✓		✓	✓

⇒ Q & S are EPI which are covering {Min-terms {0, 4, 7, 8, a, b}}



'S' is covering 4 min-terms

→ Min-terms a & b could be (1,3) or (13,15)

$P = w'y'z'$
$Q = x'y'z'$
$R = w'xy'$
$S = xz$
&
$a = 13, b = 15$

→ Now, if a & b are (1,3) then P would never be a PI. So, (1,3) is not possible.

⇒ (13,15) is the value of 'a' & 'b'.

Date.....

(29) Variable entrant Maps (VEM)

→ It is similar to K-map but ~~last~~ apart from 0's & 1's, variables could also be present.

Adv. :

→ using VEM smaller ~~K-map~~ we can represent more variable than same sized K-map.

ex:- $f(A,B,C) = \Sigma(1,2,3,5,6)$

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

⟹

A	B	f
0	0	C
0	1	1
1	0	C
1	1	C'

(o/p written in terms of variable 'C'.)

⇓

Now, since this is a two var. func^n we need only two var. K-map.

B \ A	0	1
0	C	C
1	1	C'

} using smaller K-map more var. can be represented.

Date.....

(8a) Minimization using VEM :-

Steps:-

(i) Set all the variables in cell as '0' & obtain the SOP.

(ii) (a) Make One var. in the cell as '1' & obtain SOP by making earlier minterms (is) as don't cares

(b) Multiply the above SOP with the Concerned var.

(iii) Repeat the step '2' until all the var. in the cell are Covered.

(iv) SOP of VEM is obtained by ORing the previous SOP expressions.

ex:-

C \ AB	00	01	11	10
0	D	1	D'	D'
1	D	1	0	Ø

Consider D & D' as seperate variables

Step 1:-

C \ AB	00	01	11	10
0	0	1	D'	D'
1	0	1	0	Ø

SOP = $\bar{A}B$

Step 2:-

C \ AB	00	01	11	10
0	1	Ø	0	0
1	1	Ø	0	Ø

$\bar{A}$ for 'D'

C \ AB	00	01	11	10
0	0	Ø	1	1
1	0	Ø	0	Ø

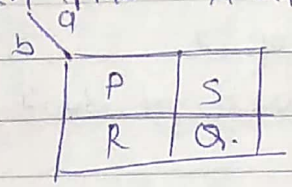
$A\bar{C}$ for 'D'

Step 3:- SOP = $\bar{A}B + \bar{A}D + A\bar{C}\bar{D}$

Date.....

(31) Example on VEM :-

(Q) $f(A,B,C) = \Sigma(3,5,6,7)$ is realised by following VEM, then find P, Q, R, S.



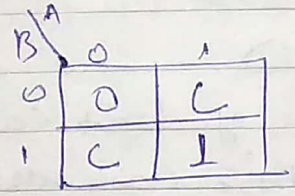
(a) P = 0, R = S = C, Q = 1

(b) P = Q = 0, R = C, S = 1

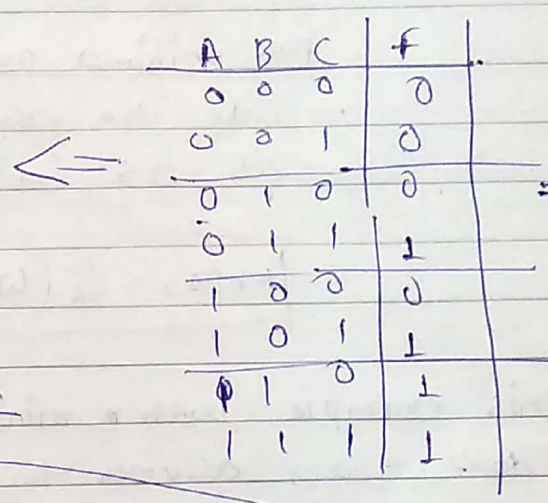
(c) P = 0, Q = R = C, S = 1

(d) None

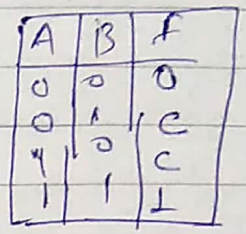
→



VEM for f



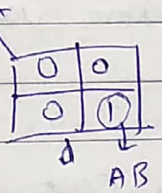
A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



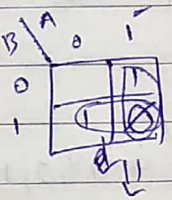
A	B	f
0	0	0
0	1	C
1	0	C
1	1	1

Minimisation

Step 1 :-



Step 2 :-



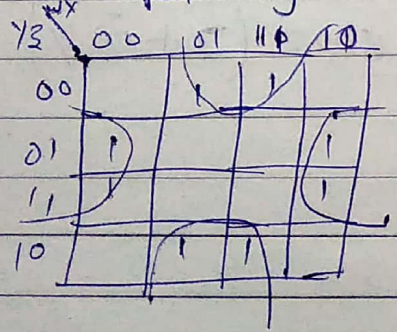
for f'

SOP = AB + AC + BC

Not the part of soln

(32) finding the free variables :-

(Q) How many variables are free in the expression denoted by the following K-map.



$f = X\bar{Z} + \bar{Z}\bar{X}$

two free variables (Y & W)

Date.....

(33) Relationship b/w minimal pos & sop in case of don't cares :→

ex:-

yz \ wx	00	01	11	10
00	0	∅	∅	0
01	0	0	1	1
11	1	1	1	1
10	0	0	0	0

for minimal sop ⇒

→ Take the values of don't cares as 0's.

$m\,sop = (w\bar{z} + y\bar{z})$

For minimal pos ⇒

→ Take the values of don't cares as 0's in this case also.

$m\,pos = \bar{z}(w+y)$

***

⟹ So, in this example both m minimal pos & sop in case of don't cares assumes the same value (ie. 0) of don't cares.

(34) Relationship b/w minimal pos & sop in case of don't cares 2 :-

ex:- let $f(w,x,y,z) = \Sigma(1,2,3,5,13) + \Sigma\emptyset(6,7,8,9,11,15)$ & let $f(w,x,y,z)$ be minimal sop & $g(w,x,y,z)$ be the minimal pos. Are 'f' & 'g' identical?

⇒

yz \ wx	00	01	11	10
00				∅
01	1	1	1	∅
11	1	∅	∅	∅
10	1	∅		1

Minimal sop = $(\bar{z} + \bar{w}y)$

$f = \Sigma(1,2,3,5,6,7,9,11,13,15)$

Spiral

⇒ Both the minimal pos & sop will be independent of same number of variables if the function does not have any don't cares.

↓ in case of don't cares func^n is not unique

Date.....

YZ \ WX	00	01	11	10
00	0	0	0	Ø
01	1	1	1	Ø
11	1	Ø	Ø	Ø
10	1	Ø	0	0

$g = \Sigma(1,2,3,5,6,7,9,11,13,15)$ → same as f

but if PI chosen are diff. (i.e.)

Y3 \ WX	00	01	11	10
00	0	0	0	Ø
01	1	1	1	Ø
11	1	Ø	Ø	Ø
10	1	Ø	0	0

$g = \Sigma(1,2,3,5,13,9)$ → (f & g not identical)

So, 'f' & 'g' are not identical..

→ In case of don't cares it's not always true that Minimal pos & sop have same functim

(35) Minimal expression represented by the map is free from

(a) 1-var. (b) 2-variables (c) 3 variables (d) dependent on all

CD \ AB	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	1	1	1	1
10	1	1	1	1

⇒ # for minimal sop:-

(All the three PI are EPI)

$M\,SOP = (C + \bar{A}D + \bar{A}\bar{B})$

dependent on all vars.

for minimal POS:-

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

dependent on all vars.

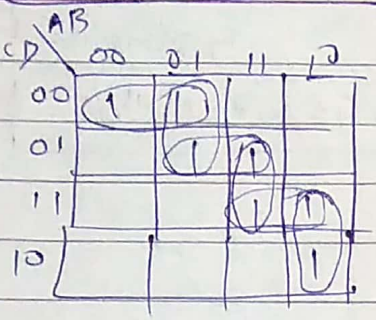
Spiral

****

Date.....

(36) No. of irredundant & minimal exp:-

ex:-



(i) How many PI?
(ii) " " EPI?
(iii) " " Redundant PI?
(iv) Minimal sop
(v) How many minimal sop?

(i) 6
(ii) 2
(iii) 4 (6-2)

~~(iv)~~

PI chart:-

	0	4	5	13	15	14	10
$P = \bar{A}C\bar{D}$	✓	✓					
$Q = \bar{A}B\bar{C}$		✓	✓				
$R = B\bar{C}D$			✓	✓			
$S = ABD$				✓	✓		
$T = ACD$					✓	✓	
$U = AB\bar{C}$						✓	✓

P & U are EPI, so (0, 4, 11 & 10) are covered by EPI.

Reduced PI Chart:-

(Q implies R)

	5	13	15
$Q = AB\bar{C}$	✓		
$R = B\bar{C}D$	✓	✓	
$S = ABD$		✓	✓
$T = ACD$			✓

⇒ Q is dominated by R (i.e. Q→R)
T " " " S (i.e. T→S)
(So Q & T can be deleted)

Date.....

PI Chart Reduced (once again) :-

	5	13	15
R = BCD	✓	✓	
S = ABD		✓	✓

⇒ For this Reduced PI chart both R & S are EPI

(iv) minimal sop = P + V + R + S

(v) Total minimal ~~pos~~ sop :-

→ Every minimal ~~core~~ exp. are irredundant exp but every irredundant, are not minimal.

→ Apart from the minterms which are covered by EPI the minterms 5, 13 & 15 are not covered by EPI.

Minterms →	5	13	15
(covered by ~~minterms~~ PI) ⇒	(Q, R) (0, 1)	(R, S) (0, 1)	(S, T) (0, 1)

⇒ total 8 combinations are :-

⇒ (Q, R, S), (Q, R, T), (Q, S, S), (Q, S, T), (R, R, S), (R, R, T)

(R, S, S), (R, S, T)

(repeating)

⇒ QRS + QRT + QS + QST + RS + RT + RST

⇒ QS + RT + QRT + QST + RS

⇒ QS + RT + RS

→ either (Q & S) or (R & T) or (R & S) PI can be used as pair to find minimal sop.

i.e. (P + V + Q + S) or (P + V + R + T) or (P + V + R + S)

3 minimal sop.

Date.....

(37) Don't cares are never included in PI chart:-

⇒ Never include Don't cares in PI chart.

ex:- $f(v,w,x,y,z) = \Sigma(13,15,17,18,19,20,21,23,25,27,29,31) + \Sigma\phi(1,2,12,29)$

	13	15	17	18	19	20	21	23	25	27	29	31
A = vz			✓		✓	✓	✓	✓	✓	✓	✓	✓
B = wx	✓	✓									✓	✓
C = vwx'y'									✓			
D = vw'xy'						✓	✓					
E = vw'x'y				✓	✓							
F = vwxy'	✓											
G = w'x'yz'				✓								
H = w'x'y'z			✓									

⇒ It can be observed from the PI chart that A, B & D are EPI. All minterms covered by them are:- (13, 15, 17, 19, 20, 21, 23, 25, 27, 29, 31) only minterm which is not covered by any EPI is '18'

Reduced PI chart:-

	18
C = vwx'y'	
E = vw'x'y	✓
F = v'wxy'	
G = w'x'yz'	✓
H = w'x'y'z	

two minimal SOPs possible

Minimal SOP = (A + B + D + E) or ~~AB~~(A + B + D + G)

Date.....

→ No. of ticks (✓) in Rows of C, F, G, H is only one but the size of ~~minterm~~ PI is reduced, because Don't care is also present in the K-map & not in PI chart. (missing ticks (✓) are don't cares)

(38) Functions involving functions example 1 :-

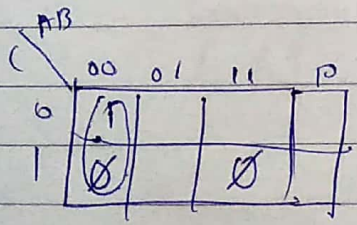
(Q) Consider 3 variable functions
$f(A,B,C) = \Sigma(0,7) + \Sigma\phi(1,2,5)$ & $f_2(A,B,C) = \Sigma(0,1,3) + \Sigma\phi(4,7)$

find $f_1 \cdot f_2$ & $f_1 + f_2$ when minimized.

=> method 1

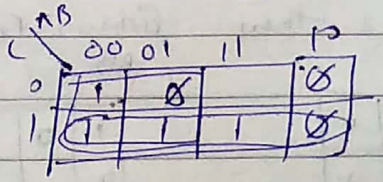
	f_1	f_2	$f_1 \cdot f_2$	$f_1 + f_2$
0	1	1	1	1
1	Ø	1	Ø	1
2	Ø	0	0	Ø
3	0	1	0	1
4	0	Ø	0	Ø
5	Ø	0	0	Ø
6	0	0	0	0
7	1	Ø	Ø	1

for $f_1 \cdot f_2$



$f_1 \cdot f_2 = \bar{A}\bar{B}$

for $f_1 + f_2$



$f_1 + f_2 = (C + \bar{A})$

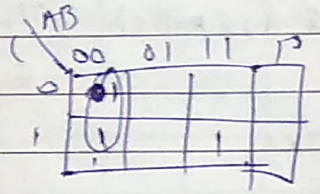
Date.....

Method - 2:-

⇒ $f_1 \cdot f_2$ will be equal to '1' if both f_1 & f_2 are '1'.
ie: intersection of minterms of f_1 & f_2

→ $f_1 \cdot f_2$ will be don't cares (ϕ) when ($f_1 = 1$ & $f_2 = \phi$) or ($f_2 = 1$ & $f_1 = \phi$)

$$\therefore f_1 \cdot f_2 = \Sigma(0) + \Sigma_\phi(1,7)$$

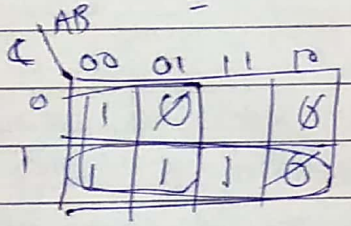


$\bar{A}\bar{B}$

⇒ $f_1 + f_2$ will be 1 when any one of f_1 or f_2 is '1'

→ $f_1 + f_2$ will be ϕ when either ($f_1 = 0$ & $f_2 = \phi$) or ($f_2 = 0$ & $f_1 = \phi$)

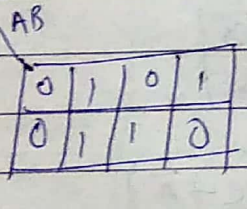
$$\therefore f_1 + f_2 = \Sigma(0,1,3,7) + \Sigma_\phi(4,2,5)$$



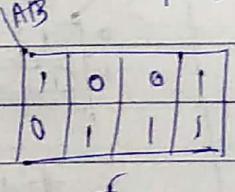
⇒ $\bar{C} + A$

(39) Functions involving functions 2:-

(Q) Consider a new boolean oper^n '$' defined as $A\$B = A' + B$, then find $f_1 \$ f_2$.

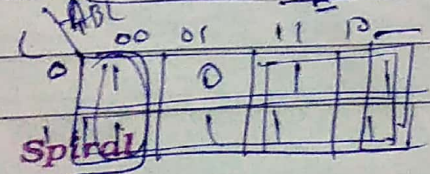


f_1



f_2

⇒ $f_1 \$ f_2 = f_1' + f_2$ (all 3 PI are EPI)



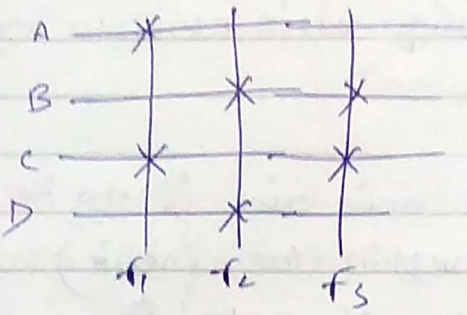
$f_1' + f_2$

$\Rightarrow f_1' + f_2 = (\bar{C} + A\bar{B} + A)$

Date.....

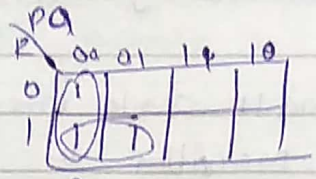
(9a) Functions involving functions example 3:-

(Q) consider 3-var. functions
$f_1(P,Q,R) = \Sigma(0,1,3)$, $f_2(P,Q,R) = \Sigma(3,5,7)$ & $f_3(P,Q,R) = \Sigma(1,3,7)$. These functions are sharing 4 prime implicants A,B,C,D (all of them are pairs) as shown. find A,B,C,D.



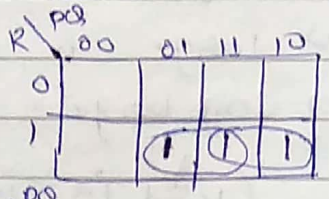
{A,B,C,D are 4 Prime implicants}

=> f_1 :-



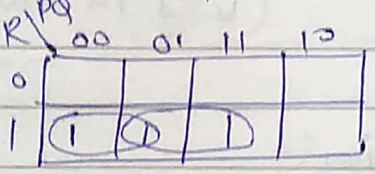
$f_1 = \bar{P}\bar{Q} + \bar{P}R$

f_2 :-



$f_2 = (QR + PR)$

f_3 :-



$f_3 = (\bar{P}R + QR)$

=> So, $C = \bar{P}R$, $B = QR$, $A = \bar{P}\bar{Q}$, $D = PR$

C: Common in f_1 & f_2

B: Common in f_2 & f_3

3. Design & Synthesis of Combinational Circuits

Date.....

(1) Introduction to logic design :-

→ The main application of switching theory is in the design of digital circuits. It is called logic design.

→ These circuits are designed using basic elements called gates.

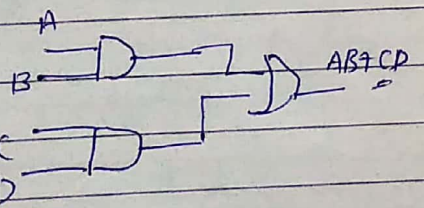
→ While designing the circuit the main aim is to :-
(i) ↑ the speed (by ↓ the levels in the circuit)
(ii) ↓ the cost (by ↓ the no. of gates)

→ If no. of gates < 10 ⇒ small scale integeration (SSI)
→ " " " " (10-100) ⇒ Medium " " (MSI)
→ " " " " (100-1,000) ⇒ large " " (LSI)
→ " " " " > 1000 ⇒ very large " " (VLSI)
(used in current systems)

(2) AND-OR, OR-AND realisation :-

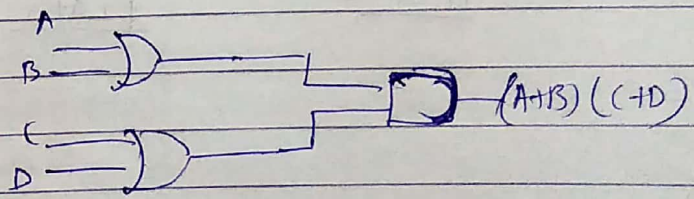
→ (AND-OR) ⇒ SOP

ex:- AB+CD



→ (OR-AND) ⇒ POS

ex:- (A+B)(C+D)



→ No. of i/p to a gate = fan in

→ So, no. of levels in a circuit depends on fan in & no. of gates used in the circuit.

Spiral

→ Realisation means implementation of a func^n using gates

Date.....

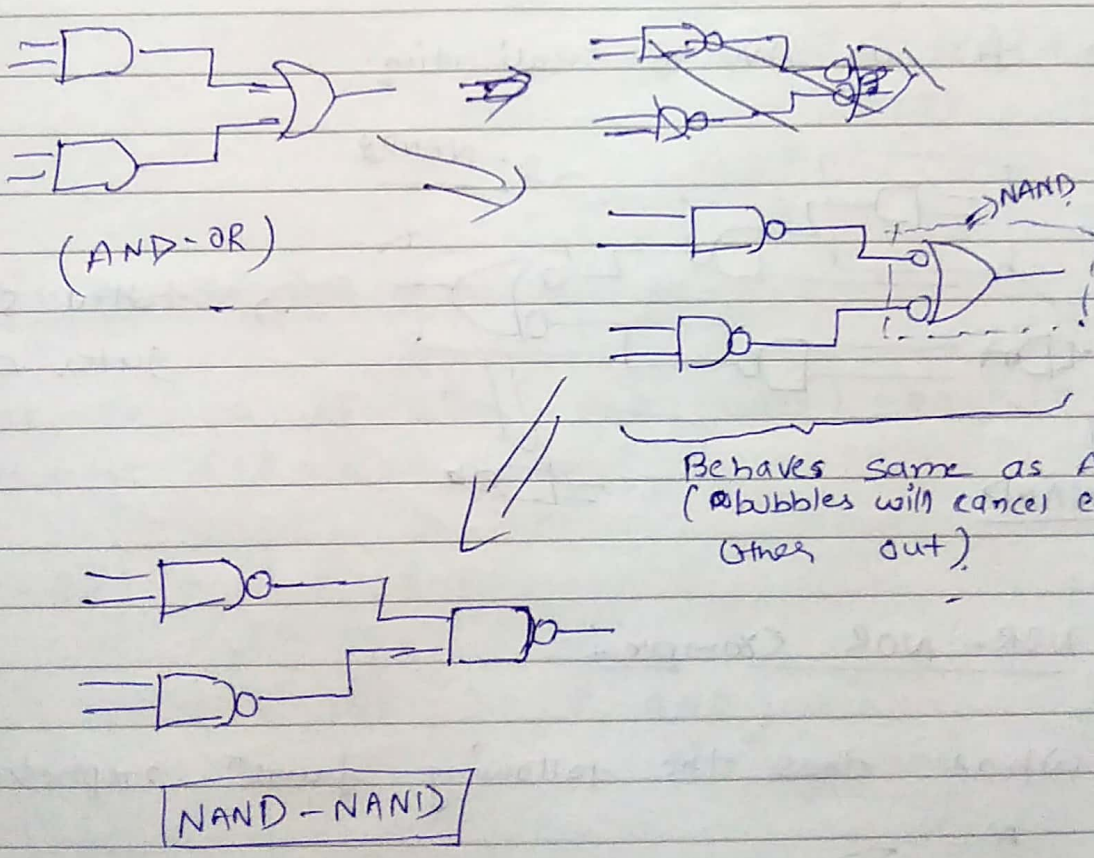
→ If no. of levels ↑ in a circuit then speed ↓.
(due to propagation time of signals.)

***
→ The cost of gates with lower ~~fan-in~~ fan-in(s) are lesser compared to gates with higher fan-in.
ex: ~~A gate of~~ (AND-gate with '3' fan-in is costly compared to AND-gate with '2' fan-in.)

***
→ If gates with higher fan-in are used in the circuit then no. of levels ↓ (thus speed ↑) but ~~cost~~ ↑

***
AND-OR ~~circuit~~ can be realised using NAND-NAND

ex:-



**
→ Using only NAND gates ~~or~~ any function can be realised. (because NAND is Comb^n of (AND & NOT) which is functionally complete)

Spiral

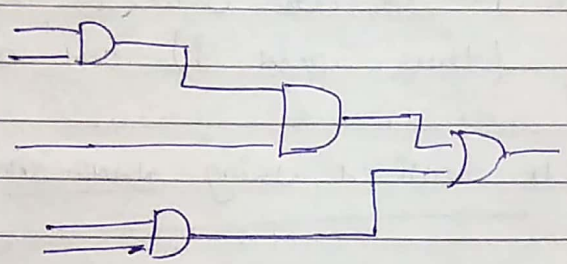
Date..15/02/19..

→ Similarly, OR-AND can be realised with NOR-NOR

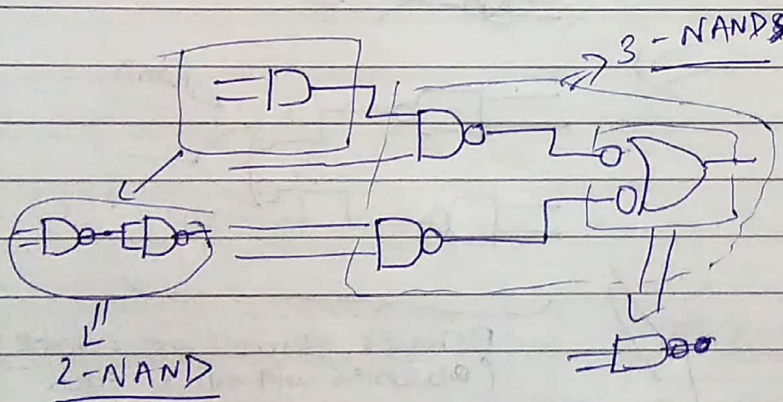
→ NOR & NAND are universal gates.

3) Minm no. of NAND gates example :-

(Q) Identify min. no. of NAND gates req. to represent the following.



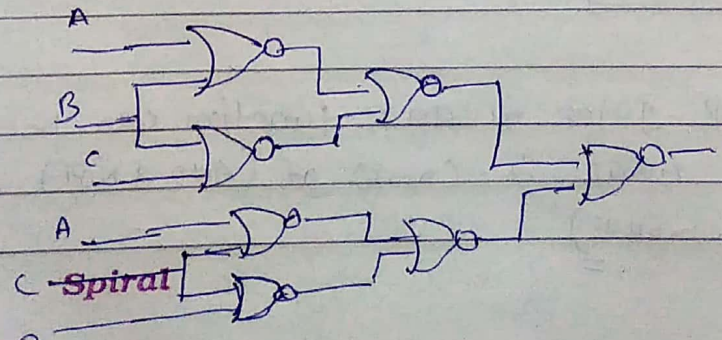
⇒ It's an AND-OR realisation.



→ total 5 NAND gates are req.

(4) NOR-NOR example :-

(Q) What does the following func^n represent?

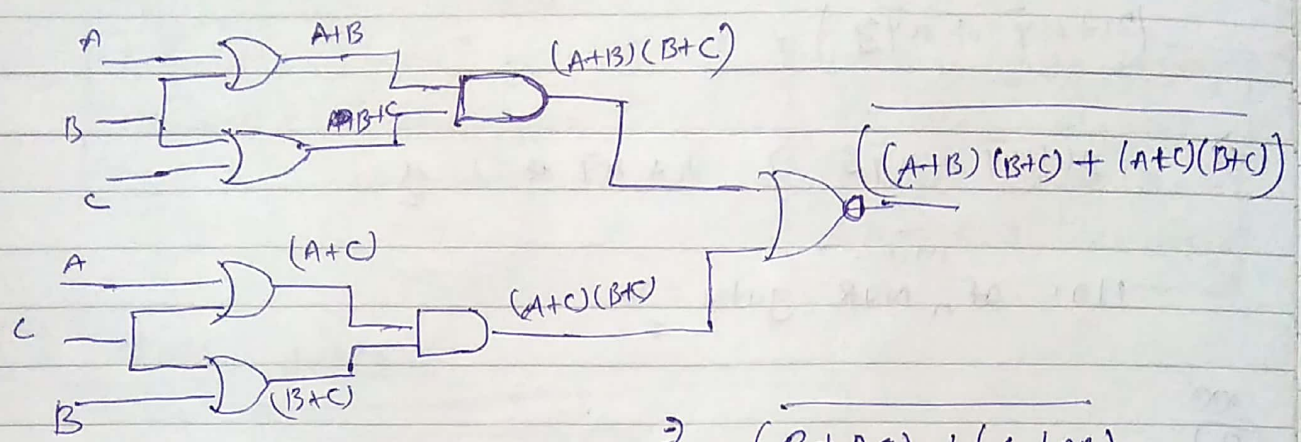


Spirat

→ ANY funch ~~must~~ have atleast two levels of gates.

Date.....

⇒ NOR-NOR ~~to~~ to OR-AND



⇒ $\overline{(B+AC)+(C+AB)}$

→ $\overline{(B+AC)}\,\overline{(C+AB)}$

→ $(\bar{B}\cdot\overline{AC})(\bar{C}\cdot\overline{AB})$

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

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

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

→ $\bar{B}\bar{C}(1)$

⇒ $\bar{B}\bar{C}$

=

(5) Min. no. of NOR gates example:-

ex:- find the no. of 2 i/p NOR gates req. to represent $f(P,Q,R) = P+QR$.

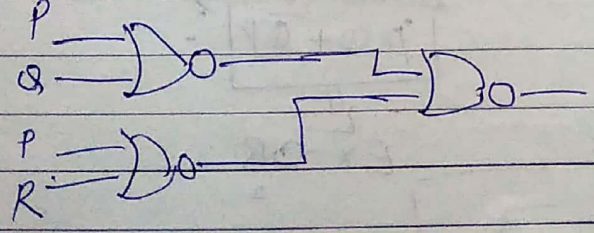
⇒ $(P+QR) = (P+Q)(P+R)$

↓↓

POS (ie. OR-AND)

↓↓

(NOR-NOR)



3-NOR gates are req.

Spiral

→ While finding min. no. of gates req. convert the exp. in minimal form first.

Date.....

(6) Min. no. of NOR-gates:-

(Q) Min. no. of NOR gate required to implement:

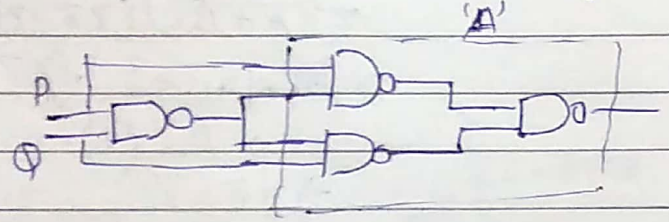
$(x + x\bar{y} + x\bar{y}z)$?

⇒ $x + x\bar{y} + x\bar{y}z \Rightarrow x + x\bar{y} \Rightarrow x$

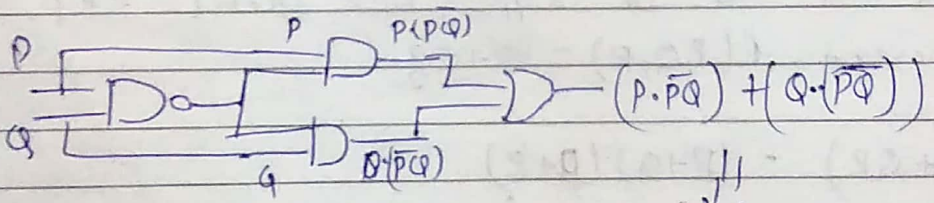
Min. no. of NOR gate = 0

(7) EX-OR & EX-NOR implementation with NOR & NAND:-

ex:- what does the following represent?



⇒ Part 'A' can be converted to AND-OR



$\Rightarrow \overline{PQ}(P+Q)$

$\Rightarrow (\bar{P}+\bar{Q})(P+Q)$

$\bar{P}Q + \bar{Q}P$

EX-OR

EX-OR $\Rightarrow (NAND)_{min.} = 4$

EX-NOR $\Rightarrow (NOR)_{min.} = 4$

Spiral

Date.....

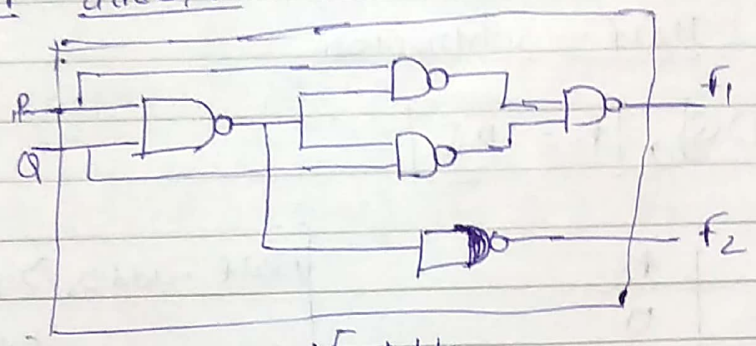
→ (To get EX-NOR, replace NAND with NOR.)

EX-OR ⇒ $(NOR)_{min} = 5$

EX-NOR ⇒ $(NAND)_{min} = 5$

= 1 more gate is req. to negate the o/p to get EX-OR from EX-NOR & vice versa

(8) Half adder :-



Half-Adder

$f_1 = P \oplus Q$ & $f_2 = PQ$

P	Q	f_1	f_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half adder ⇒ It adds only two bits

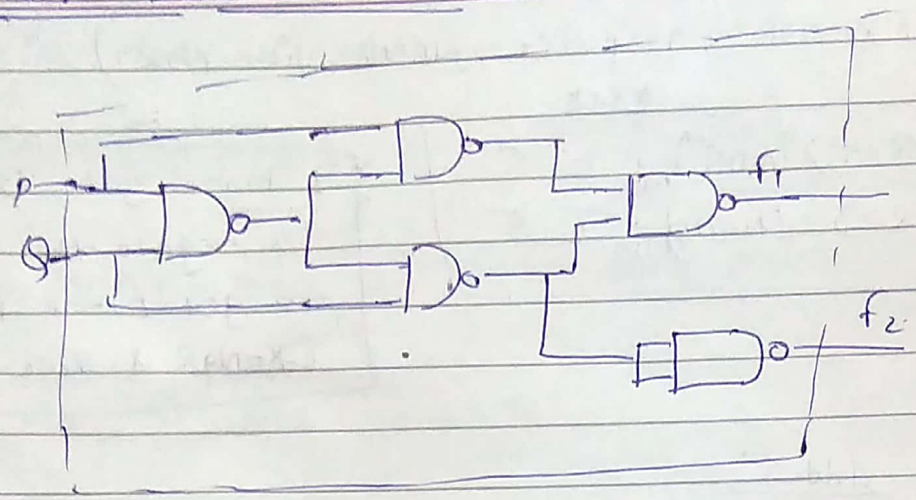
full adder ⇒ It adds three bits

1+1 = 10 → sum, carry

so, f_1 = sum & f_2 = carry

Date.....

(9) Half subtractor :-



Half - Subtractor.

$f_1 = P \oplus Q$, $f_2 = \bar{P}Q$

P	Q	f_1	f_2
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Half-adder ⇒ TWO bit Subtraction.

borrow

0 - 1 = 1

(1)0 → 2

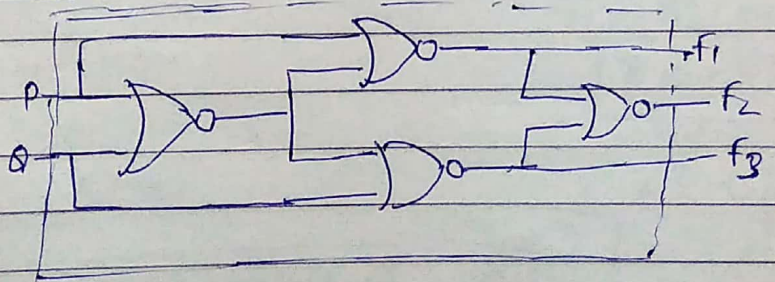
-(1) - 1

1

f_1 ⇒ Subtraction,
f_2 ⇒ Borrow

When '1' is borrowed then no. becomes '10' i.e. '2' & '1' is subtracted, so sub = 1

(10) Comparator :-



e. TWO - bit Comparator.

Spiral

Date.....

$f_2 = \bar{P}\bar{Q} + PQ = P \odot Q$, $f_1 = \bar{P}Q$, $f_2 = P\bar{Q}$

P	Q	f_2	f_1	f_3
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

All f_1, f_2 & f_3 are Mutually excl. ie. if any func^n is 1 then other is '0'.

$(P=Q)$ $(P<Q)$ $(P>Q)$

$f_2 \Rightarrow P=Q$ $f_1 \Rightarrow P<Q$ $f_3 \Rightarrow P>Q$

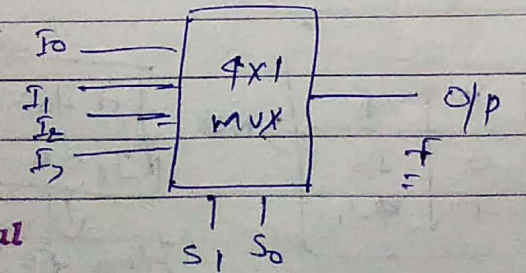
(11.) Introduction to MUX :-

→ A Multiplexer is an electric switch that connects one out of 'n' i/p to o/p. (always 1 o/p)

→ It cannot change the logical level of the i/p, it only provides the connection b/w i/p & o/p. (It only takes the i/p & gives the same o/p acc. to the i/p).

→ It is functionally complete, i.e. all boolean func^n can be realised using only MUX w/o any other gates.

ex:-



Block-diag.

→ Characteristic table

S_1	S_0	o/p
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Date.....

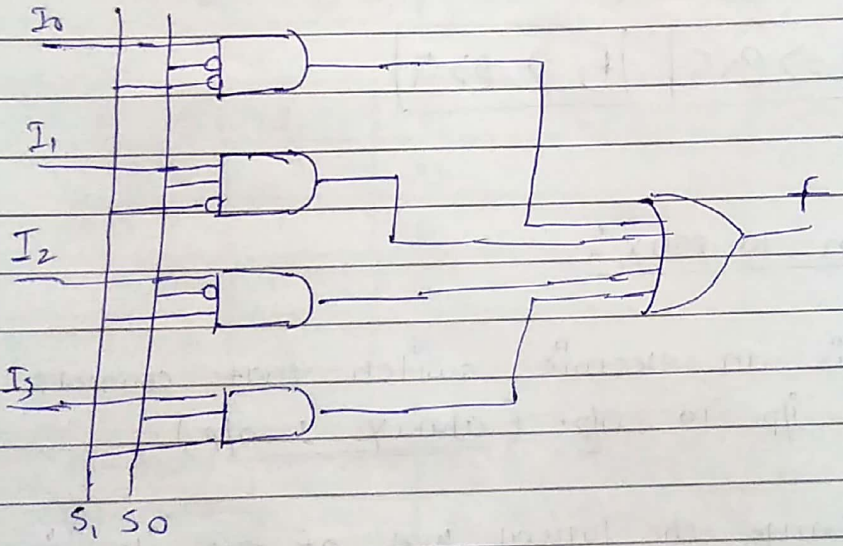
→ $2^n \times 1$ Mux have 'n' select lines.

Characteristics table:- How the o/p behaves on the basis of select lines:

$$f = \bar{S}_1 \bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1 \bar{S}_0 I_2 + S_1 S_0 I_3$$

o/p

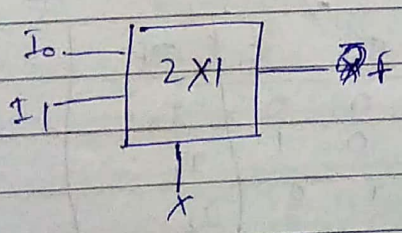
AND-OR realisation of 4×1 Mux:-



(12) Proving Mux is functionally complete:-

→ To prove it is functionally comp. we need to desire (+, '-') or (·, -).

(i) realising 'NOT':-



$$f = \bar{x} I_0 + x I_1$$

x	f
0	I_0
1	I_1

→ NOT

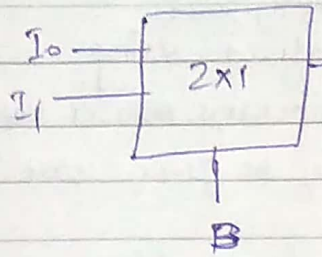
if $I_0 = 1$ & $I_1 = 0$

then $f = \bar{x}$ = NOT is realised

Spiral

Date.....

(ii) Realising 'AND' with MUX:-



$f = \bar{B} I_0 + B I_1$

A & B are i/p

⇓

char. table

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

⇒ Truth table of AND

~~A B ⇒ AB~~

B	f
0	I_0
1	I_1

⇒ ~~$f = I_0 B +$~~

$f = \bar{B} I_0 + I_1 B$

⇓

(for $I_1 = A$ & $I_0 = 0$ then $f = AB$)

i.e. AND is realised.

(iii) Realising 'OR' using MUX:-

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

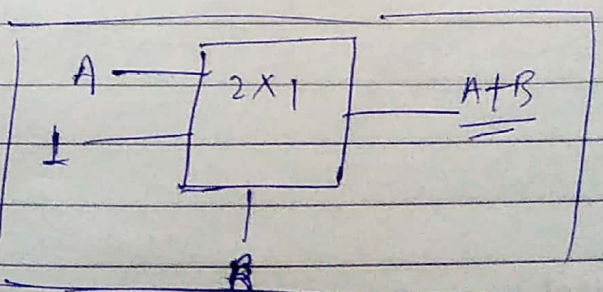
⇒ $A+B = \bar{A}B + A\bar{B} + AB$

⇒ $B(\bar{A}+A) + \bar{B}(A)$

Comparing with char. eqn of 2x1 MUX

⇒ Use 'B' as select line & $I_0 = A$ & $I_1 = 1$ in 2x1 MUX to get 'OR'

⇓



Spiral → So, MUX is functionally complete

→ The order of select lines & var. assignment to select lines are very imp. (Order is $S_n S_{n-1} \cdots S_0$)

Date.....

(13) Implementing functions with MUX

Example 1:-

→ Any 'n' var. func^n can be easily realised (w/o using gates) using $2^n \times 1$ MUX. (since, no. of possible minterms for 'n' var. func^n is 2^n, & each i/p of MUX can be given one value)

→ If along with MUX, gates are also used (not always), then bigger var. func^n can also be realised using smaller mux.

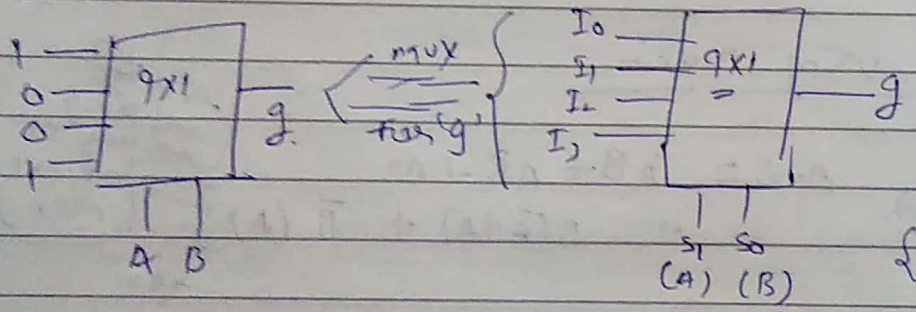
ex:- 3 var. using 9x1 MUX.

ex:-

A	B	g
0	0	1
0	1	0
1	0	0
1	1	1

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

{2 var. func^n with $2^2 \times 1$ MUX & extra gate is needed?}



{$S_1 = A$ & $S_0 = B$}

$g = \bar{A}\bar{B}(1) + \bar{A}B(0) + A\bar{B}(0) + AB(1)$

So, $I_0 = 1$, $I_1 = 0$, $I_2 = 0$; $I_3 = 1$

Spiral

Date.....

ex:- Realise 'g' using a 4x1 MUX

A	B	C	g
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

⇒ Since 'g' is a 3-var. Func^n & we can only assign 2 var. to select lines.

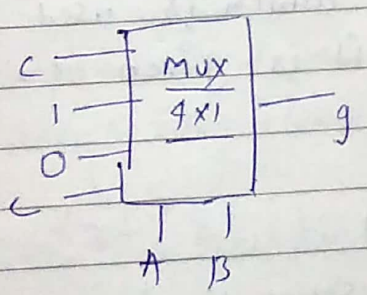
⇓

Let's take $S_1 = A$ & $S_0 = B$

$g = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}C + ABC$

$\Rightarrow \bar{A}\bar{B}(C) + \bar{A}B(\bar{C}+C) + A\bar{B}(0) + AB(C)$

S_0, $I_0 = C$, $I_1 = 1$, $I_2 = 0$, $I_3 = C$



{ In this case we have implemented ~~the~~ a 3 var. func^n with 4x1 MUX, but we didn't used any gate, in some cases we might require gates ~~when~~.

Date.....

(14) Implementing functions with MUX

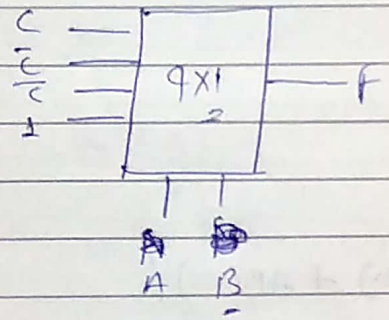
Example - 2 :-

ex:- Implement $f(A,B,C) = \Sigma(1,2,4,6,7)$ using 4x1 MUX

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

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

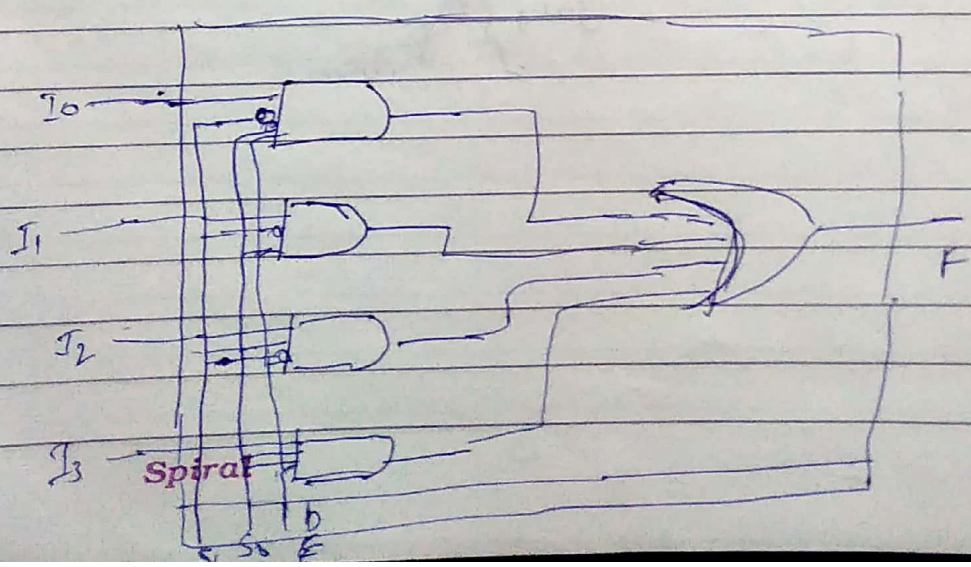
$I_0 = C,\ I_1 = \bar{C},\ I_2 = \bar{C},\ I_3 = 1$



(15) Multiplexer with enable i/p :-

→ Enable i/p works as a switch which is used to turn on or off the MUX & it is given to all the gates of the MUX (i.e. AND)

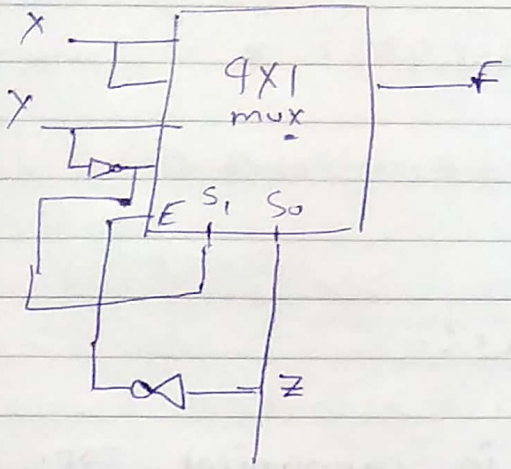
→ when E = 0 (o/p is always 0)
& when E = 1 (O/p is acc. to select line)



Date.....

$$f = E(\bar{S}_1\bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1\bar{S}_0 I_2 + S_1 S_0 I_3)$$

ex:- Minimize the func^n represented by following MUX:



$S_1 = \bar{Y}$, $S_0 = \bar{Z}$.

$E = \bar{Z}$

$f = E(\bar{S}_1\bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1\bar{S}_0 I_2 + S_1 S_0 I_3)$

~~$f = \bar{Z}(S_1 S_0$~~

$f = \bar{Z}(Y\bar{Z}_0(X) + Y\bar{Z}(X) + \bar{Y}\bar{Z}(Y) + \bar{Y}\bar{Z}(\bar{Y}))$

$\Rightarrow \bar{Z}(XY + Z\bar{Y})$

$f \Rightarrow XY\bar{Z}$

(16) Relationships b/w select lines i/p of a MUX:-

ex:- Implement f using 4x1 mux ~~such that~~ where

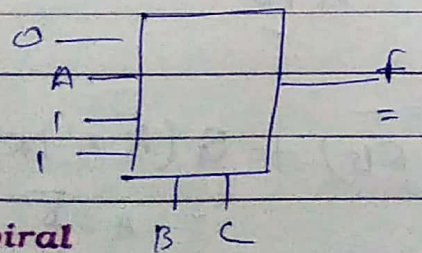
(a) $S_1 = B$, $S_0 = C$ (b) $S_1 = C$, $S_0 = B$.

$f(A,B,C) = \Sigma(2,3,5,6,7)$

$\Rightarrow$ (a) $f = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

$\Rightarrow A\bar{B}\bar{C}(0) + \bar{B}C(A) + B\bar{C}(\bar{A}+A) + BC(\bar{A}+A)$

$\Rightarrow I_0 = 0$, $I_1 = A$, $I_2 = 1$, $I_3 = 1$

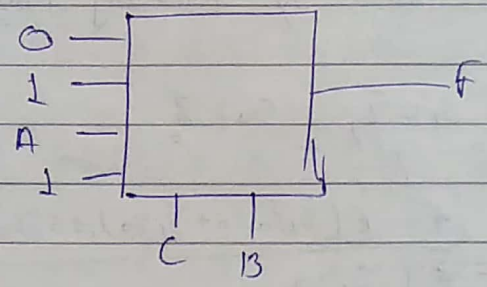


Spiral

Date.....

(b) $f = \bar{C}\bar{B}(0) + \bar{C}B(A+\bar{A}) + C\bar{B}(A) + CB(\bar{A}+A)$

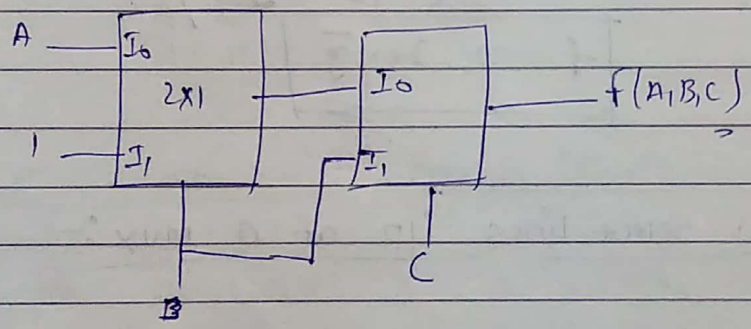
$I_0 = 0,\ I_1 = A+\bar{A},\ I_2 = A,\ I_3 = 1$



(17)

Cascading multiplexers – ex 1 :-

ex:- what is the functional in canonical sop.



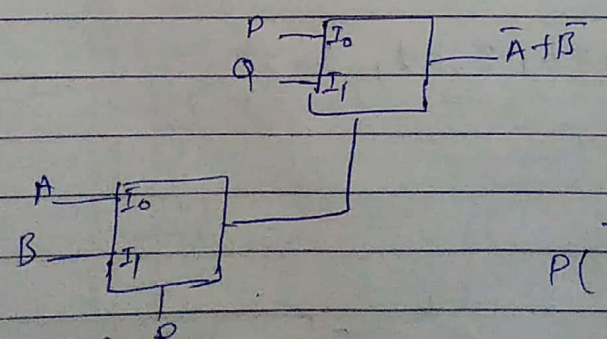
$\Rightarrow (\bar{B}A + B)\bar{C} + BC = f$

$\Rightarrow A\bar{B}\bar{C} + B\bar{C} + BC = f$

$A\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C} + \bar{A}BC + ABC = f$ — Canonical sop

(19) Cascading mux – ex 3:-

ex:-



what are the value of (P, Q, R) to get o/p $\bar{A}+\bar{B}$

$\overline{P(\bar{R}A + RB) + Q(\bar{R}A + RB)}$

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

Spiral R

Date.....

⇒ ~~$\overline{R+A}=1$~~

$\left((R+\bar{A})\cdot(\bar{R}+\bar{B})\right)P + \overline{(RA)}\cdot 1$

$\bar{R}AQ + RBQ$

⇒ $\left(R\bar{B} + \bar{A}\bar{R} + \bar{A}\bar{B}\right)P + \bar{R}AQ + RBQ$

⇒ $R\bar{B}(RP) + \bar{A}(RP) + \bar{A}\bar{B}\cdot + A\bar{R}Q + BRQ$

⇒ ~~$R=0, \bar{R}=1$~~ (or) ~~$R=1$, then $\bar{R}=0$~~

~~& $P=1$ & $Q=1$~~

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

Now, match it with option (option is given)

(20) Expansion of multiplexers :-

→ Building larger order mux using smaller order mux.

~~XXX~~

Ex:- No. of 2X1 mux needed to build a 8X1 mux.

⇒ 1 mux $\xrightarrow{\text{Covers}}$ 2 line

then 1 line is covered by → $\frac{1}{2}$ mux

Now, (8X1) mux has 8 lines

$8 \longrightarrow 8\times\frac{1}{2} \Rightarrow 4$ (mux) (level-1)

$4 \longrightarrow (4\times\frac{1}{2}) \Rightarrow 2$ (mux) (level-2)

$2 \longrightarrow (2\times\frac{1}{2}) \Rightarrow 1$ (mux) (level-3)

So ~~7~~ (Mux) needed of 2X1 in 3-levels to construct 8X1 mux.

Spiral

Date.....

Ex:- No. of 9×1 mux needed to build a 32×1 mux.

1 Mux Covers ⟶ 9 lines

So 1 line is covered by ⟶ $\frac{1}{9}$ mux

Now, (32×1) mux has 32 lines

lines as i/p at each level

- (32), $\left(32 \times \frac{1}{9}\right) = 8$ (mux) ⇒ (level-1)
- (8), $\left(8 \times \frac{1}{9}\right) = 2$ (mux) ⇒ (level-2)
- (2), $\left(2 \times \frac{1}{9}\right) = \frac{1}{2}$ (mux) = 1 (mux) ⇒ (level-3)

Total = 11 (MUX) & 3 levels

½ mux can't be designed one entire 9×1 mux will be needed, but only two i/p lines will be used.

→ Using all 9×1 mux at 3-levels at its full capacity a 69×1 mux can be build.

Building a (m×1) mux using (N×1) mux (m>N)

'N' lines are covered by ⟶ 1 mux

1 " is " " ⟶ $\frac{1}{N}$ mux

Now, for a m×1 mux having 'm' lines.

- (m) ⇒ $\left(\frac{m}{N} \times 1\right)$ ⟶ $\frac{m}{N}$ (mux) ⟶ level 1
- $\left(\frac{m}{N}\right)$ $\left(\frac{m}{N} \times \frac{1}{N}\right)$ ⟶ $\frac{m}{N^2}$ (mux) ⟶ level 2
- ⟶ $\frac{m}{N^k}$ (mux) ⟶ level-k

Spiral

Date.....

where, $\left(\frac{M}{N^K} \leq 1\right)$ $\Rightarrow$ $N^K \geq M$

$K \geq \log_N^M$ where K = no. of levels

or

$K = \lceil \log_N^M \rceil$

total no. of NX1 MUX req.

$\Downarrow$

$\frac{M}{N} + \frac{M}{N^2} + \frac{M}{N^3} + \cdots \frac{M}{N^K} \Rightarrow M\left(\frac{1}{N} + \frac{1}{N^2} + \cdots \frac{1}{N^K}\right)$

$= M \sum_{K=1}^{\lceil \log_N^M \rceil} \frac{1}{N^K}$

Total no. of NX1 MUX req. to realise one MX1 MUX $= M \sum_{K=1}^{\lceil \log_N^M \rceil} \frac{1}{N^K}$

ex:- 64X1 using 2X1

$\Rightarrow$ no. of levels $= \lceil \log_2 64 \rceil = 6$

no. of 2X1 MUX $= 64 \sum_{K=1}^{6}$

$64\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64}\right)$

$\Rightarrow 32 + 16 + 8 + 4 + 2 + 1$

$\Rightarrow \boxed{63}$

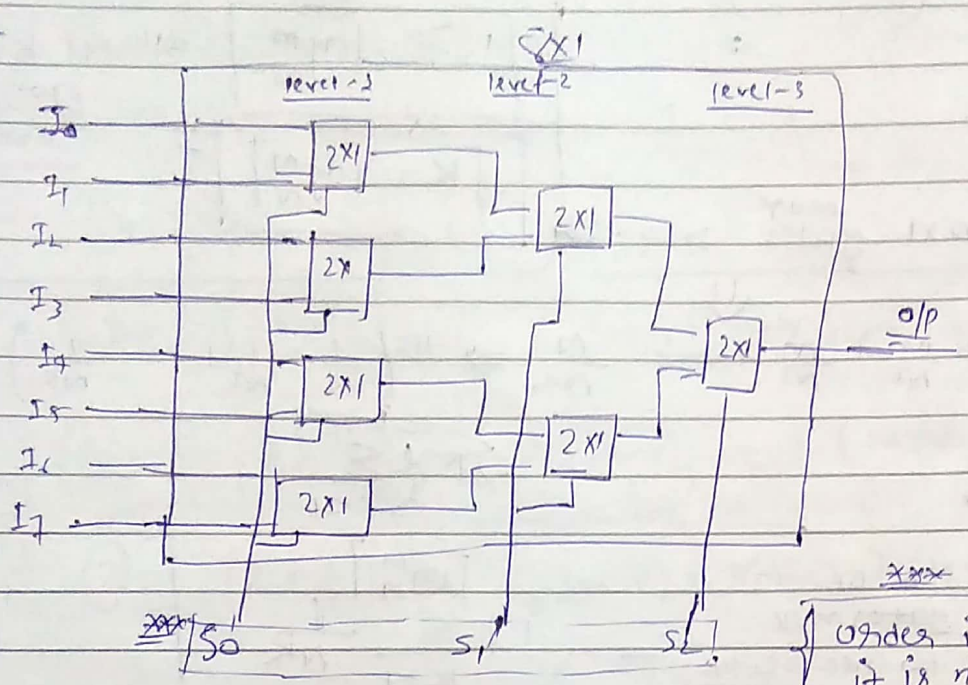
Spiral

Date.....

(21) Assigning select lines ~~using MUX~~ :-

while expanding the mux :-

ex :-



order is very imp
it is not S_2, S_1, S_0

~~ex :- for $S_0 S_1 S_2 = 001$~~

ex :- $S_2 S_1 S_0 = 001$ (ie. I_1 of 8X1 MUX should be selected)

(check it using the above order of select lines.

→ From level-1 lines selected will be ⇒ I_1, I_3, I_5, I_7 ($S_0 = 1$)

→ " level-2 " " " " ⇒ I_1, I_5 ($S_1 = 0$)

→ " level-3 " " " " ⇒ I_1 ($S_2 = 0$)

we got I_1 as o/p

→ Always check the order of select lines by assigning some values to select lines. ~~(This is only~~

Date.....

(22) Introduction to demultiplexer:-

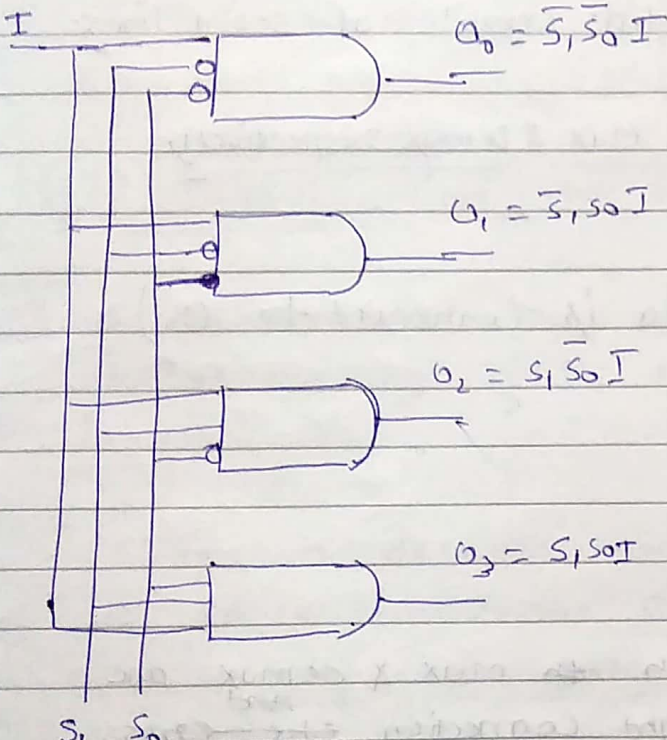
→ It performs opposite operⁿ of MUX.

→ It has 1 i/p & 2^n o/p where 'n' is no. of select lines

***
→ It is derived from MUX by joining all the i/p together & removing 'OR' gate.

***
→ It is mainly used in the constrⁿ of switches.

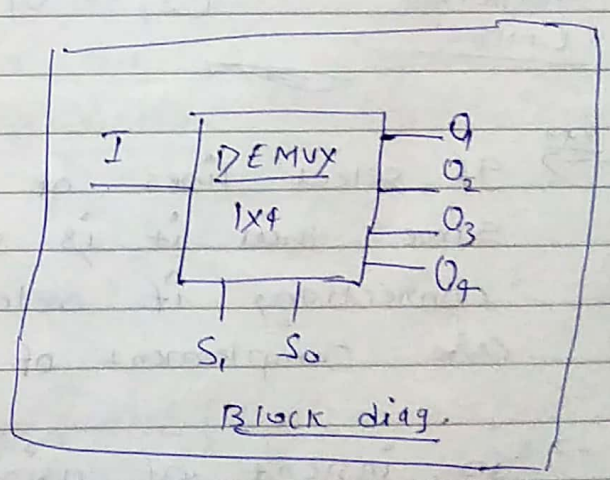
***
→ DEMUX is used at reciever end & MUX is used at transmitting end.



(realisation using AND gates)

S_1	S_0	O_0	O_1	O_2	O_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

Char - table



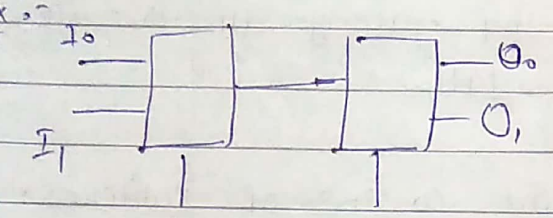
Block diag.

Date.....

Multiplexer & Demux as a switch:-

→ Comb^n of MUX & DEMUX acts as a switch.

ex:-



{ I_0 can be connected to (O_0 or O_1). Similarly I_1 can be connected to (O_0 or O_1) }.

⇒ In switching one of the i/p (from many i/ps) is connected to one of the o/p (from many o/p). & dest^n add. is selected by giving diff. comb^{n}s of select lines.

→ A & B are select lines of MUX & Demux respectively

	A	B	O/P
Straight Connec^n ⇐	0	0	($I_0 - O_0$) { I_0 is connected to O_0 }
Cross Connec^n ⇐	0	1	($I_0 - O_1$)
	1	0	($I_1 - O_0$)
Straight Connec^n ⇐	1	1	($I_1 - O_1$)

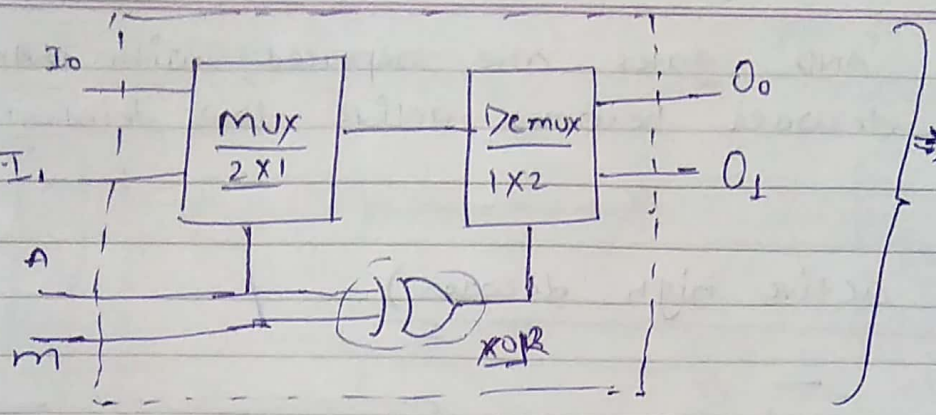
⇒ If select lines of both MUX & demux are same then it is straight connection And cross connection, if select line of both mux & demux are complement of each other.

→ So, instead of using NOT gate to complement both the MUX & DEMUX

Spiral

→ Multiplexer is universal, because it a is a two level AND-OR realisation, but Demux & decoder are only AND realisations so they OR gates to implement any function

Date..21/02/19.



⇒ (MUX & Demux as a switch)

→ When m=0 ; → Straight connection.
→ When m=1 ; → Cross Connection.

→ ~~Instead of us~~ { m=0 ⇒ o/p of XOR = A
m=1 ⇒ " " " = $\bar{A}$ }

(23) Introduction to decoder:-

***

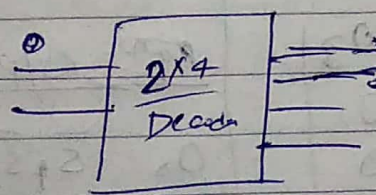
→ A Demux can be converted to a decoder by always setting I=1 & making select lines as inputs.

→ Since, a decoder provides all the minterms, we could implement any function in cannonical sop using 'OR' gate. (ie. universal)

⇓

(OR gate is additional)

ex:-

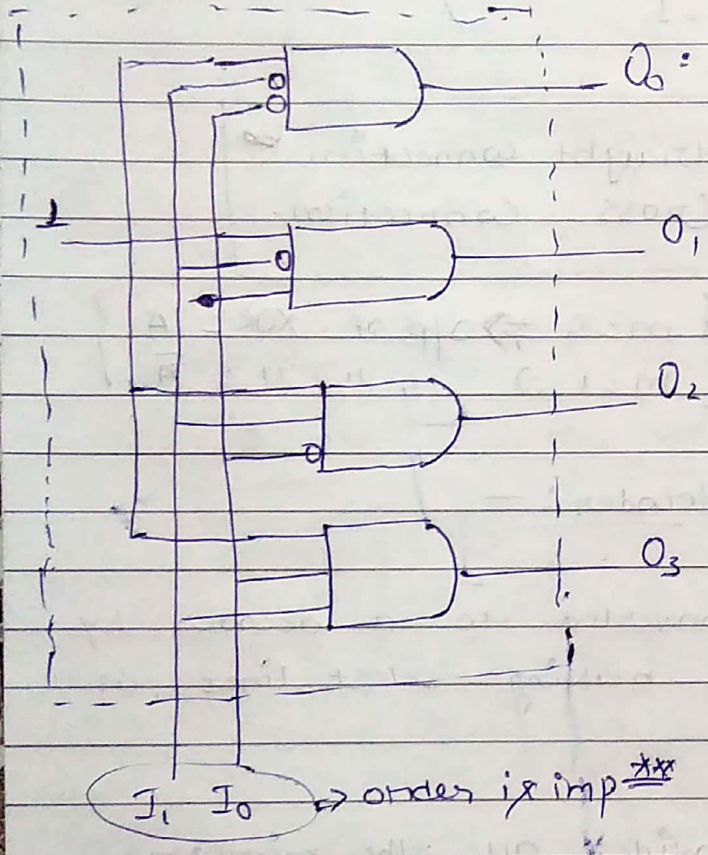


⇒ $f = \Sigma(0,1)$

Date.....

-> If all the 'AND' gates are replaced with 'NAND' gates the decoder becomes active low decoder.

Decoder:- (or active high decoder)



Same as Demux only diff. is 'I' is replaced with const. '1'.

Select lines act as i/p lines.

Decoder is of the form $n \times 2^n$

i/p: n o/p: 2^n

(realisation of Decoder)
(2X4)

Truth table:-

I_0	I_1	O_0	O_1	O_2	O_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

o/p's:-

$O_0 = \overline{S_1} \overline{S_0}$

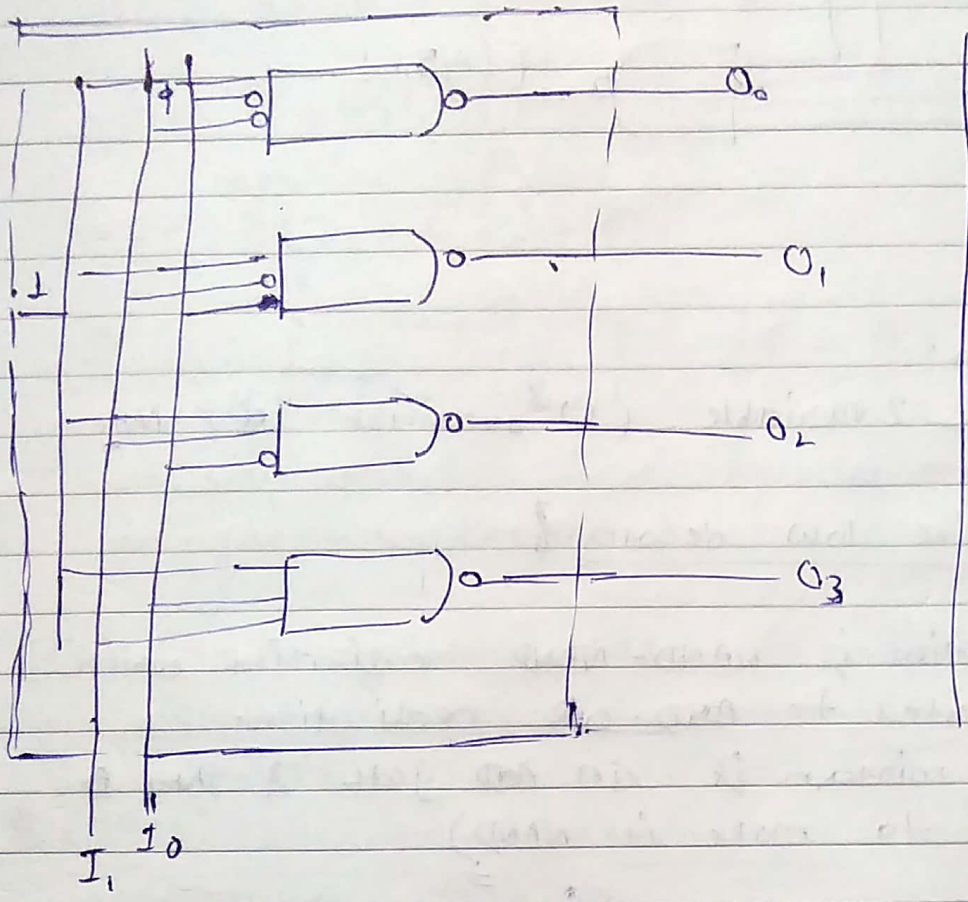
$O_1 = \overline{S_1} S_0$

$O_2 = S_1 \overline{S_0}$

$O_3 = S_1 S_0$

Date..22/02/19

Active low decoder:-

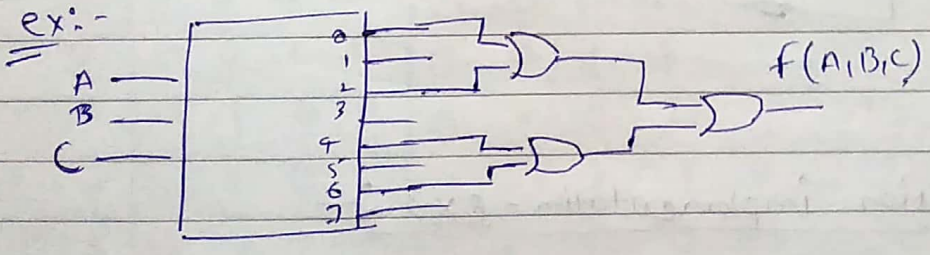


I_0	I_1	O_0	O_1	O_2	O_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

~~Active lo~~

(29) Implementing functions with decoder ex1:-

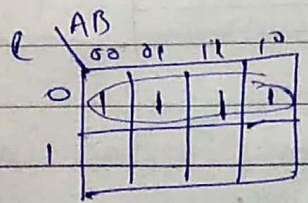
ex:-



f is ~~free from~~

(a) 1 variable (b) 2 variable (c) 3 variable (d) None.

$\Rightarrow f = \Sigma(0,2,4,6)$

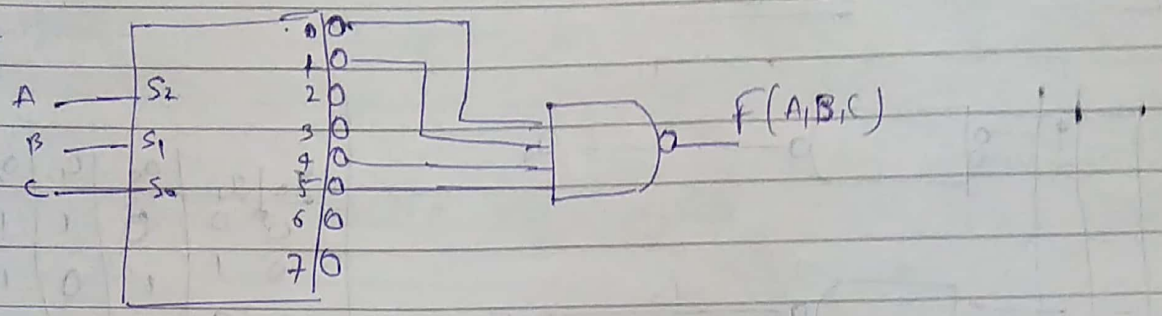


$f = \bar{C}$

Date.....

(25) Implementing functions with decoder – ex?:-

ex:-



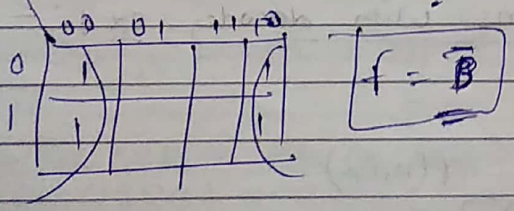
f is free from.

(a) 1 variable (b) 2 variable (c) 3 variable (d) None

⇒ This is active low decoder.

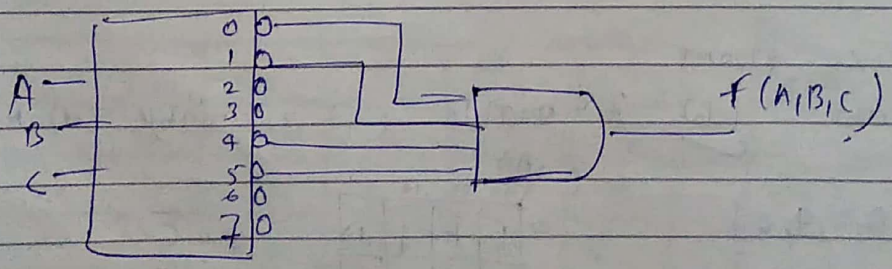
→ The realisation is NAND-NAND realisation which can be converted to AND-OR realisation.
(because each minterm is an AND gate & then complemented to make it NAND)

So, $f = \Sigma(0,1,4,5)$



(26) Decoders for function implementation – Ex-3 :-

ex:-



f is free from.

(a) 1-variable (b) 2-variable (c) 3-variable (d) None

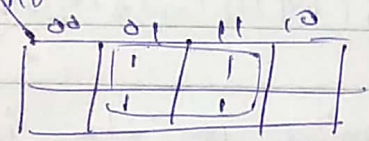
Date.....

→ It is ANDing of max terms (complement of min terms) ie. cannonical ~~collection of~~ POS.

$$f = \Pi(0,1,4,5)$$

or

$$f = \Sigma(2,3,6,7)$$



$F = \bar{B}$

→ It can also be done by complementing the function in previous example.

(27) Converting one code to other code using decoder :-

8421 (→BCD) code to 2421 code :- (∵ BCD to 2421)

	A B C D (8 4 2 1)	W X Y Z (2 4 2 1)
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	1 0 1 1
6	0 1 1 0	1 1 0 0
7	0 1 1 1	1 1 0 1
8	1 0 0 0	1 1 1 0
9	1 0 0 1	1 1 1 1

9's complement of '5' ie. 9-5 = 4 (take no. (4) & 1's comp it)

→ Such ~~function~~ code are called self-complementing code.

Date.....

$W = \Sigma(5,6,7,8,9) + \Sigma\phi(10,11,12,13,14,15)$

$X = \Sigma(4,6,7,8,9) + \Sigma\phi(10,11,12,13,14,15)$

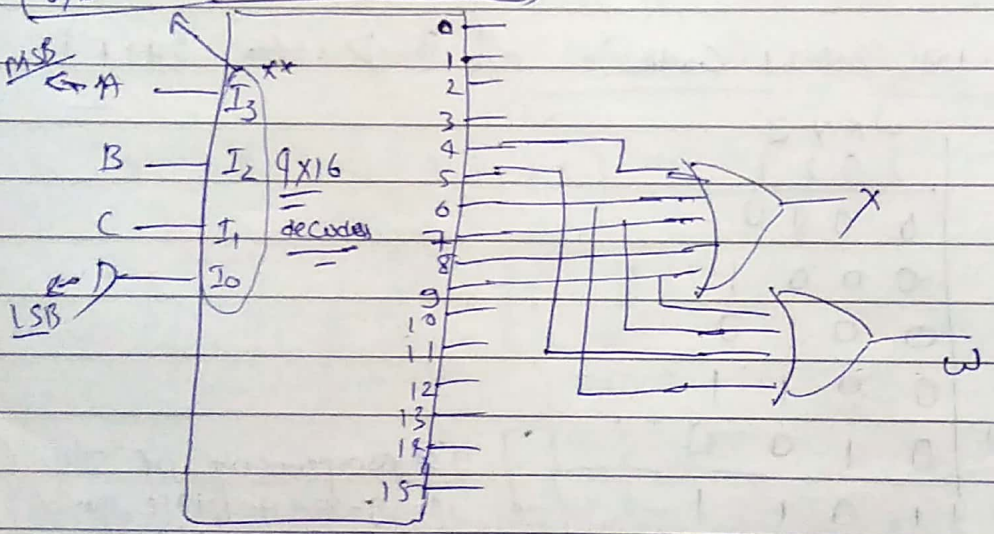
$Y = \Sigma(2,3,5,8,9) + \Sigma\phi(10,11,12,13,14,15)$

$Z = \Sigma(1,3,5,7,9) + \Sigma\phi(10,11,12,13,14,15)$

→ We don't need to minimize it to implement using decoder (because each o/p line of decoder represents Min-terms itself). Minimize only if implementation is done using gates.

→ So, atleast 4X16 decoder is needed (because 4 var)

(Order of ABCD, should be correct)



{similarly Y & Z}

→ Even though each func^n W, X, Y, Z contains don't cares they'll not be used because don't cares are used in minimization but not in implementation.

→ Use NAND gates outside for active-low decoder. (i.e. NAND-NAND realisation)

Spiral

→ Active-high ⇒ AND-OR Realisation.

Date.....

(29) ROM implementation using decoder:-

→ ROM can be used to realise combinational functions by storing appropriate values at appropriate locations.

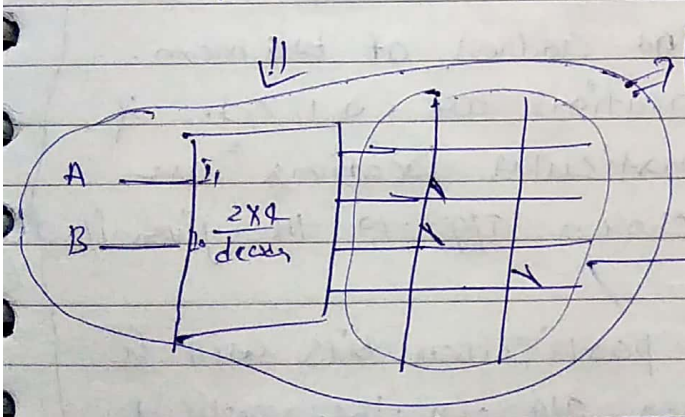
→ Every ROM is expressed in terms of ROM matrix & decoder.

→ ROM matrix contains the set of links & collections. The lines entering the matrix (horizontal lines) & leaving it (vertical lines) are called connections & the intersecn of rows & columns are called links.

ex:- Implementation of Half adder using ROM:-

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

sum = $\Sigma(1,2)$
carry = $\Sigma(3)$



(collectively it is called ROM)

→ Rom Matrix

It has 6 connection & 8 links

→ no. of vertical lines = no. of functions to be implemented.

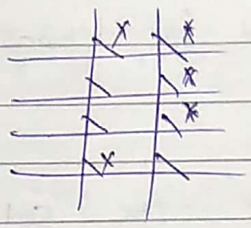
→ The vertical & horizontal lines are not overlapped, they are parallel to each other.

Spiral

Date.....

→ Initially when ROM is manufactured all the links are connected & then those ~~connections~~ links are burned which are not req. (ie. they get broken)

ex:- initially :-

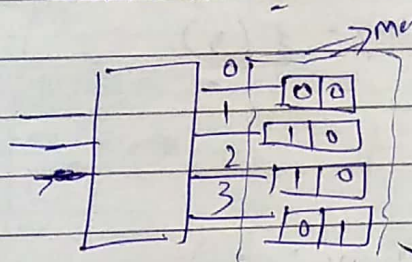


{ all the crossed ~~connections~~ links are burned to impl implement half-adder }

*** → This is why ROMs are not Reprogrammable because once they are burned they can't be joined again { so, it only Read only }

*** → It is called memory because it is remembering the connections.

ie.



→ Add address of ~~loc~~ mem. locations are 0, 1, 2, 3. & particular locations are chosen a/q to the given i/p.

→ And particular bits will be chosen a/q to the vertical lines.

Spiral

Date..23/02/19

(29) Implementing functions using only decoder:-

→ For implementing a function of n-variables the size of decoder needed is $n \times 2^n$

→ If 'm' functions are needed to be implemented then the no. of connections = $(2^n + m)$ & no. of links = $(2^n \times m)$

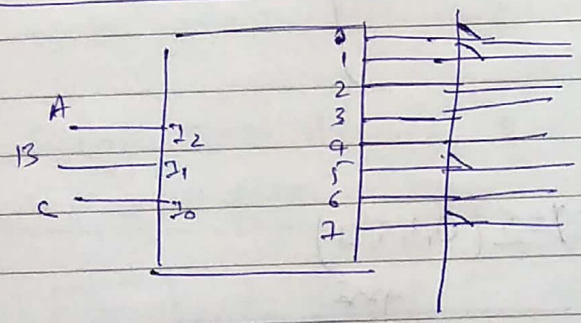
*** → The cost of the ROM depends on the no. of connections (ie. no. of horizontal & vertical lines) it can be reduced by ↓ the no. of connecn.

(30) Implementing functions using Decoder + mux ex-1:-

→ Implementing a 3-var func^n using Decoder & ROM matrix. (ie. ROM):-

ex:-

$f(A,B,C) = \Sigma(0,1,5,7)$



⇒ No. of connections = 8+1 = 9

& no. of links = 8 × 1 = 8

→ NOW, to decrease the cost of the ROM, the no. of connections can be reduced using a mux.

*** → No. of links can't be reduced (because no. of links represents no. of min terms & they'll be constant.)

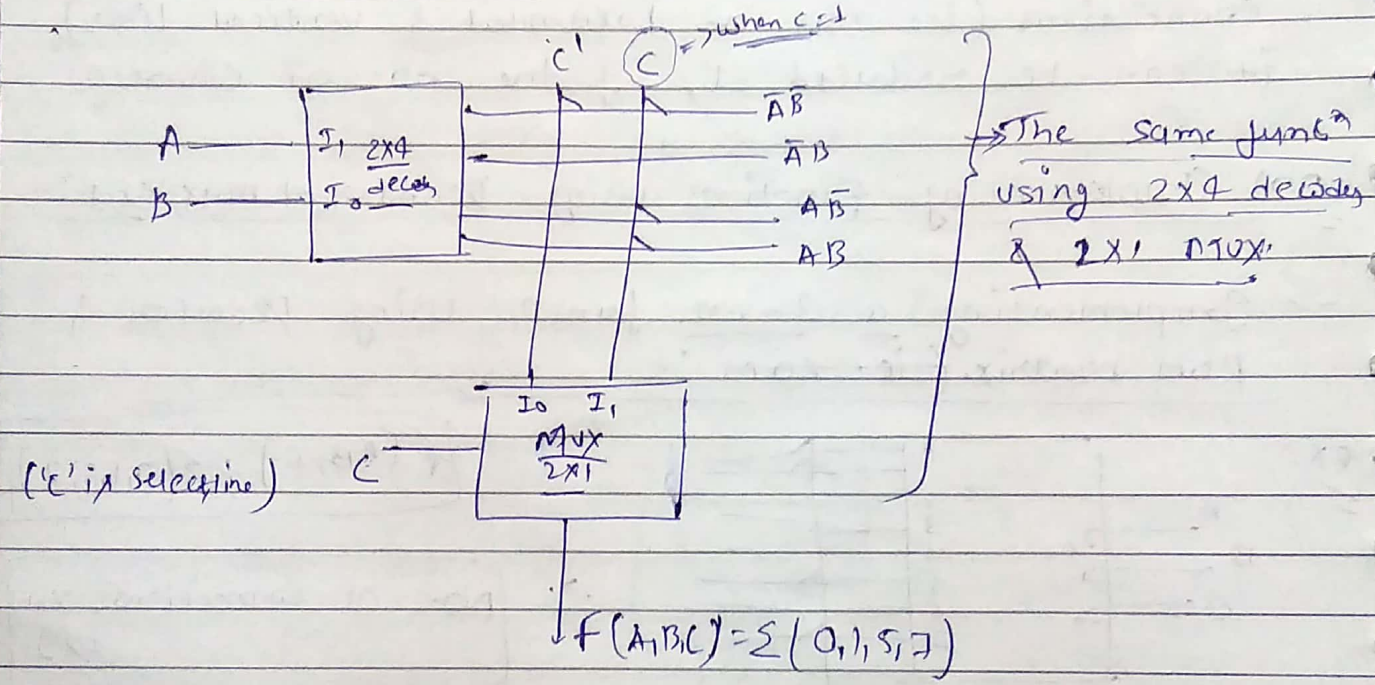
Date.....

$f = \Sigma(0,1,5,7)$

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

$\therefore$ This func^n can be implemented using 2x4 decoder & a 2x1 MUX.

ROM design using MUX & decoder to ↓ the no. of connections :-



$\Rightarrow$ Then, No. of connection = 9+2 = 6

NO. of links = 9x2 = 8

$\Rightarrow$ So, NO. of connections are reduced from 9 to 6 & no. of links remains same.

$\Rightarrow$ The reduc^n of connec^n will be huge when no. of var. in func^n is more.

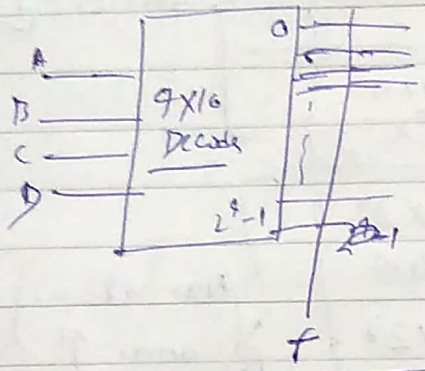
→ var. to MUX are always given as select lines.

Date.....

(31) Implementing functions using decoder & MUX (ex:-2):-

ex:- A 4-var func^n. f:-

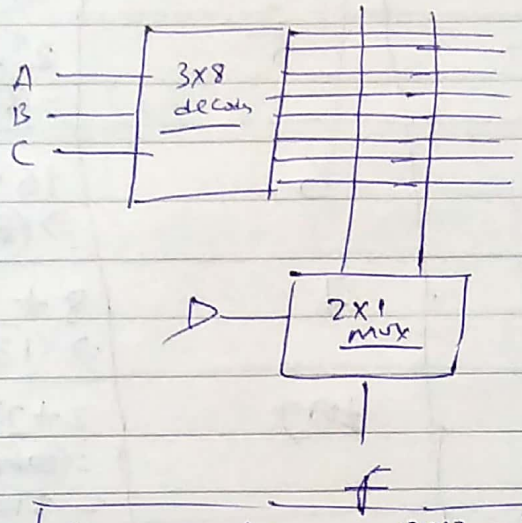
(i) using 4×2^4 decoder:-



no. of links = 16X1 = 16

no. of connec^n = 16+1 = 17

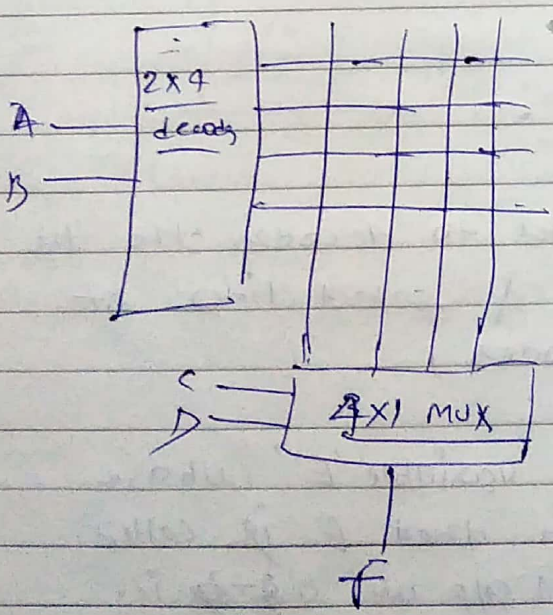
(ii) using 3X8 decoder & 2X1 MUX:-



→ NO. of links = 8X2 = 16

→ NO. of Connections = (8+2) = 10

(iii) Using 2X4 decoder & 4X1 MUX:-



→ NO. of links = 4X4 = 16

→ NO. of Connections = 4+4 = 8

XXX

→ Even though it seems as more var. are given to MUX (or) size of MUX ↑ & size of decoder ↓ then the no. of Connec^n ↓, it is not always true

Date.....

ex:- One func^n of 10 variables :-

	No. of var. in decoder	no. of var. in mux (as select lines)	No. of connections	No. of links	
(1)	10	0	1024+1 = 1025	1024×1	No. of Connec^n ↓
(2)	5	5	2^5+2^5 = 64	1024 ($2^5 \times 2^5$)	
(3)	4	6	16+64 => 80	$2^4 \times 2^6$ = 1024	No. of Conn. ↑
(4)	3	7	8+128 => 136	$2^3 \times 2^7$ = 1024	
(5)	1	~~8~~ 9	2+~~256~~ 512 = ~~258~~ 514	$2^1 \times 2^9$ = 1024	

⇒ So, No. of Connec^n does not ↑ always when no. of select lines in Mux is increased.

⇒ To implement more than 'm' ~~one~~ func^{n}s use 'm' Mux.

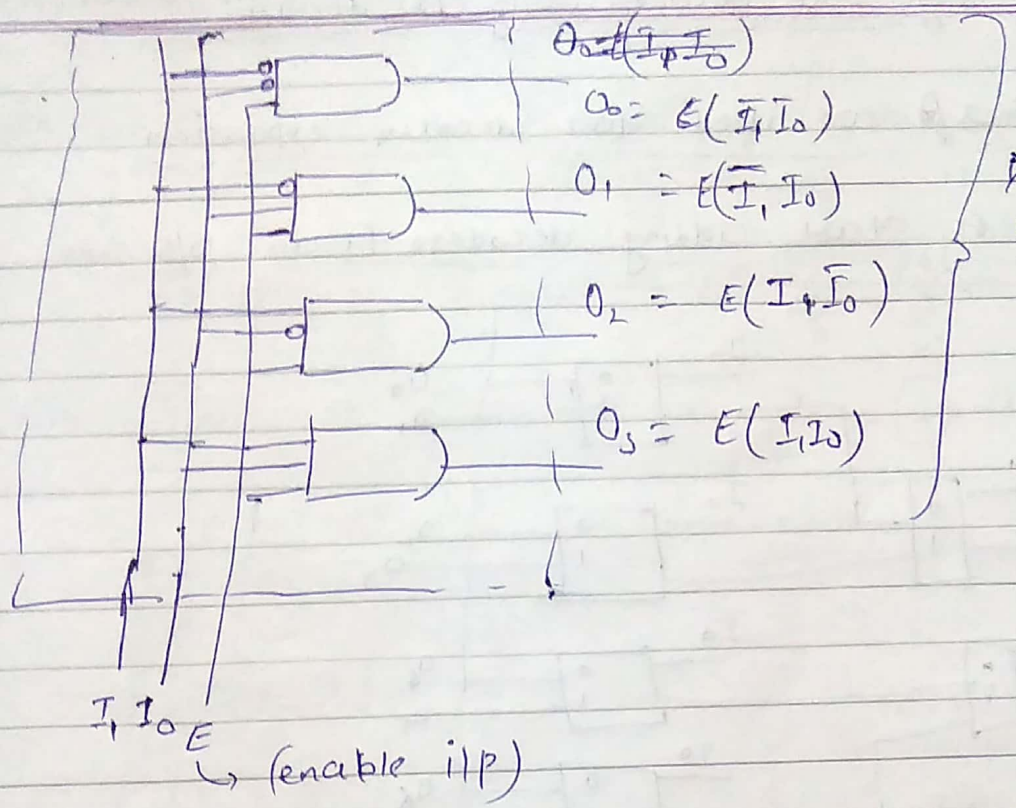
(3.2) Decoder with enable input :-

→ When the Demux uses converted to decoder the the i/p I is converted to '1' & select lines of Demux acts as i/p in decoder,

→ But if that '1' will be a variable 'E' whose value can be '0 or 1' then that E is called enable i/p. & (if E = 0 all o/p are '0' ie. Decoder is disabled)

Spiral

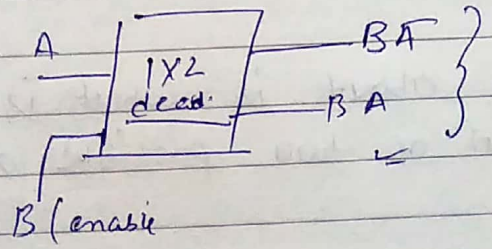
Date.....



⇒ enable i/p can be used to expand the decoder (ie expansion)

→ enable i/p can also act as i/p to the decoder. but it will not give all the min terms. (but can be used to implement some func^n)

ex:



only two min-terms out of 4 possible

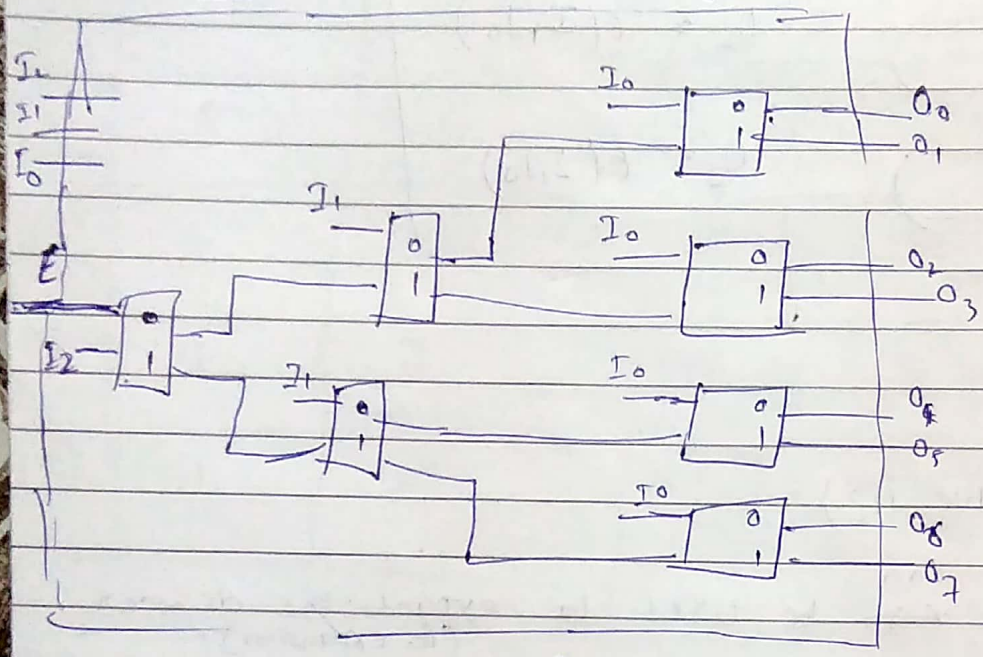
Spiral

Date.....

(33) Constructing 3X8 decoder using 1X2 decoders

→ Enable lines are used for decoder expansion.

→ ~~Start~~ Start adding decoders from o/p side



3X8 using 1X2

→ Enable lines ~~are~~ o/p ~~of~~ of next level are o/p of previous level.

→ The only thing to take care about is what is order of I_2, I_1 & I_0 - { out of two possible orders ie. I_2, I_1, I_0 or I_0, I_1, I_2 }

→ test for 001 $\{I_2, I_1, I_0\}$ => (O_1 should be selected)

Date.....

35) Constructing 6X64 decoder using 3X8 decoder's:-

⇒ 8 lines can be covered by 1 decoder

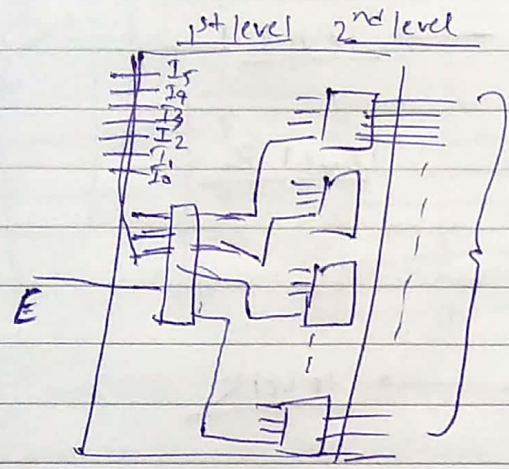
So, 1 line " " " " $\frac{1}{8}$ ".

Hence

64 " " " " " $\frac{64}{8} = (8)$ ".

↳ 8 lines " " " " $\frac{8}{8} = (1)$ "

Total = 9 decoders



→ Assign i/p $I_5 - I_0$ in order, $I_5 I_4 I_3$ at 1st level & $I_2 I_1 I_0$ at 2nd level.

6X64 decoder

ex:- No. of decoders & levels for constructing 6X64 using 2X4 decoder.

⇒ 4 lines can be covered by 1 decoder

1 line " " " " $\frac{1}{4}$ "

64 " " " " " $\frac{64}{4} = (16)$ " → level 3

↳ 16 " " " " " $\frac{16}{4} = (4)$ " → level 2

↳ 4 " " " " " $\frac{4}{4} = (1)$ → level 1

Total = (21) decoders

Date.....

(36) Expansion of decoder in general:-

→ Construct $m \times 2^m$ decoder using $n \times 2^n$ decoders

or

" $(\log_n m \times m)$ " " $(\log n \times n)$ ".

⇒ n lines → 1 decoder

1 " → $\frac{1}{n}$ "

m " → $\frac{m}{n}$ " → level 1

$\frac{m}{n}$ " → $\frac{m}{n^2}$ " → level 2

$\frac{m}{n^{k-1}}$ → $\frac{m}{n^k}$ " → level k

So,

$\frac{m}{n^k} \leq 1 \Rightarrow k \geq \lceil \log_n m \rceil$ levels

*** take ceil

$$\text{No. of decoders} = \sum_{k=1}^{\log_n m} \left(\frac{m}{n^k}\right)$$

ex: 6×64 using 4×16

⇒ levels: $\log_{16}^{64} \Rightarrow 3/2 = 2$ levels

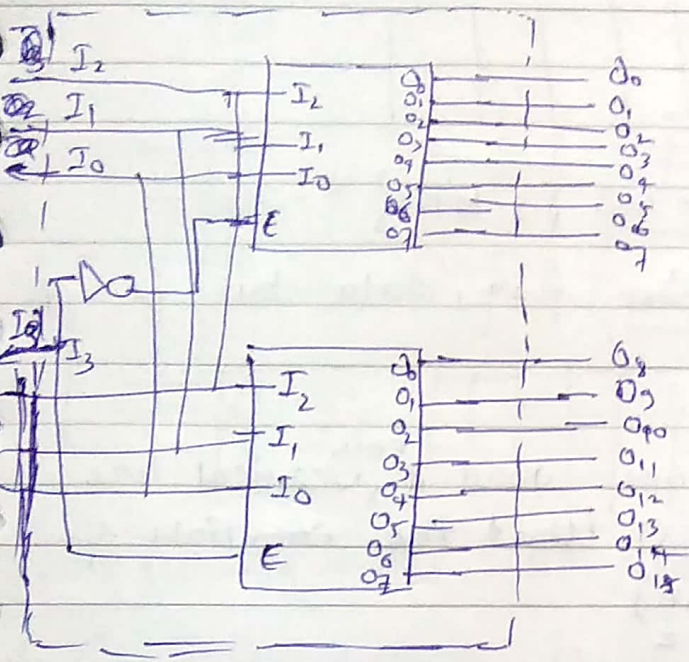
⇒ total decoders: $\frac{64}{16} + \frac{64}{16^2} \Rightarrow \left(4 + \frac{1}{4}\right) = 5$

*** Last level decoder is not fully utilised

(38) Expansion of decoder in another way:-

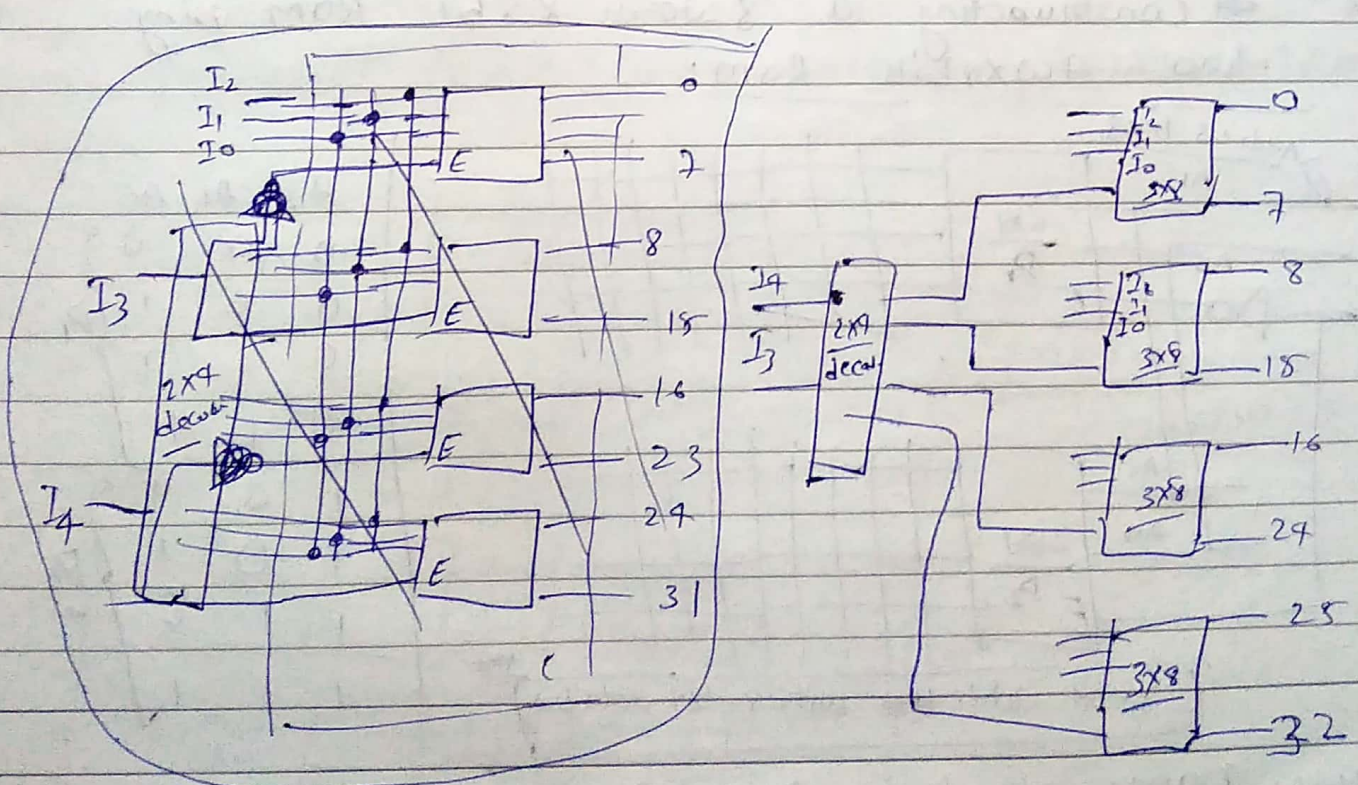
Date.....

ex:- Construct 4x16 decoder using 2->(3x8) decoder.



ex:- I_3 I_2 I_1 I_0 : 1 0 0 0 { D_8 will be be selected

ex:- Construct 4->(3x8) decoder to construct a(5x32) decoder:-



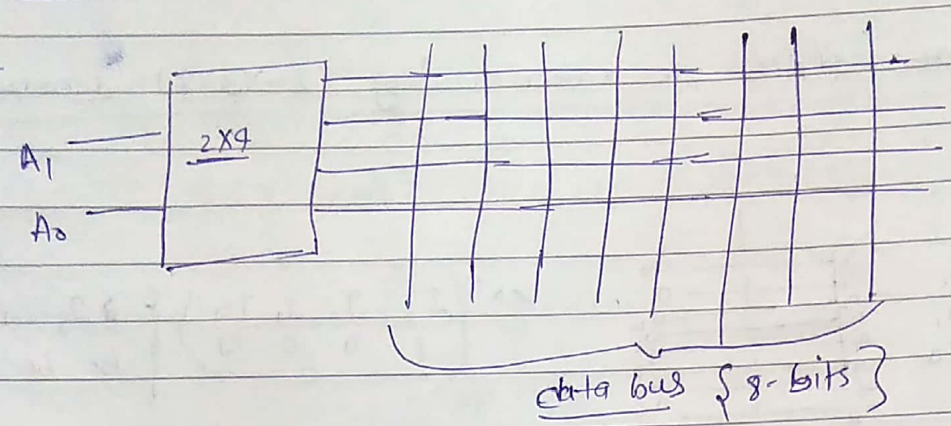
4->(3x8) & 1 (2x4) decoder

Spiral

Date.....

(39) Address expansion of ROM:-

ex:-

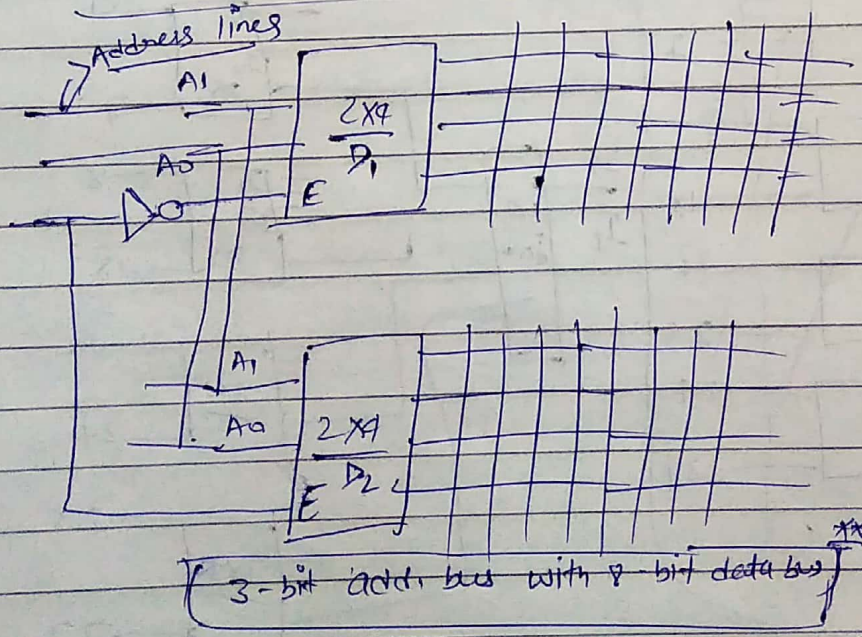


→ 4 words each of size 8 bits {so, data bus is of size 8-bits}

→ horizontal lines represent each word & vertical line each one bit. (to store '0' burn the links & to store '1' don't burn)

Address expansion:- increasing no. of memory locations w/o increasing the size of each location

ex:- Construacting a 8 words X 8 bit ROM using two 4wx8bits Rom.



A_2	A_1	A_0	
0	0	0	D_1
0	0	1	
0	1	0	
0	1	1	
1	0	0	D_2
1	0	1	
1	1	0	
1	1	1	

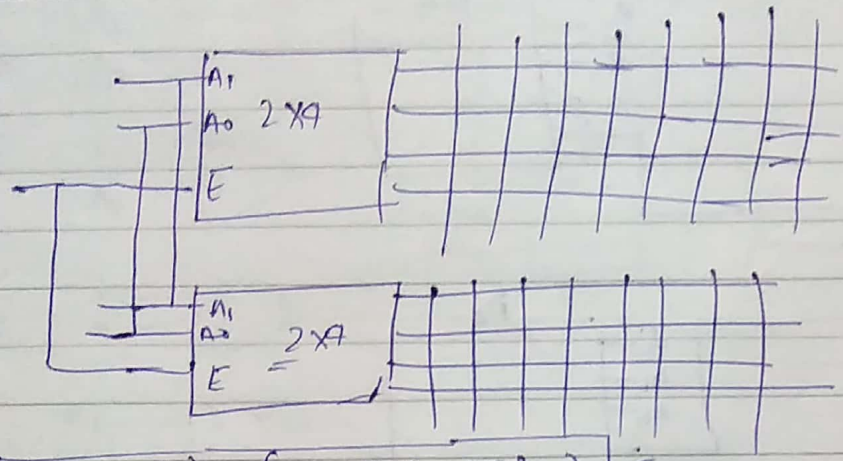
Address range of device D_1 : 0-3 } total 8 words each of size 8 bits
" " " D_2 : 4-7 }

→ Any device can be expanded using enable lines ***

Date.....

(4c) Word expansion of ROM :-

Word expansion :- Increasing the size of each memory location w/o increasing the no. of memory locations

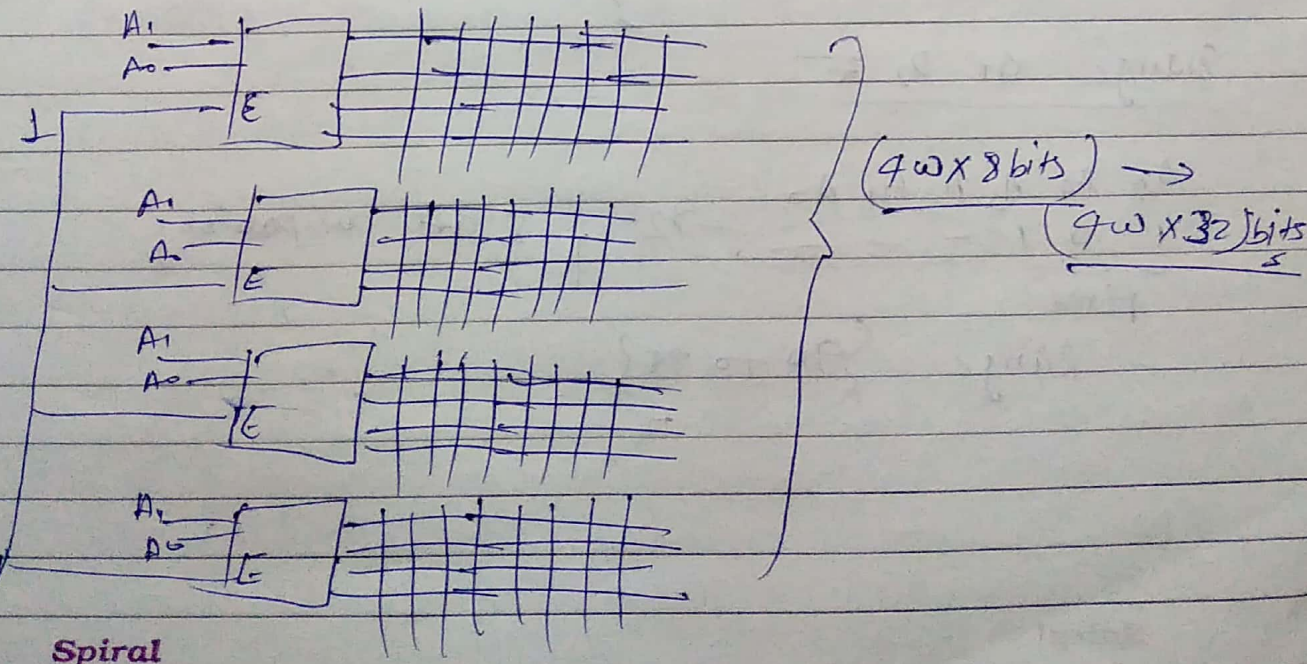


→ Not is removed from from Enable lines

(2w x 8 bits) → (4w x 16 Bits)

→ When, enable is '1' both devices will select 8 bits thus resulting in 16-bit data bus. for 2-bit Add. bus

Ex:- Construct (4w x 32) bits ROM using (4w x 8 bits) ROM.



Spiral

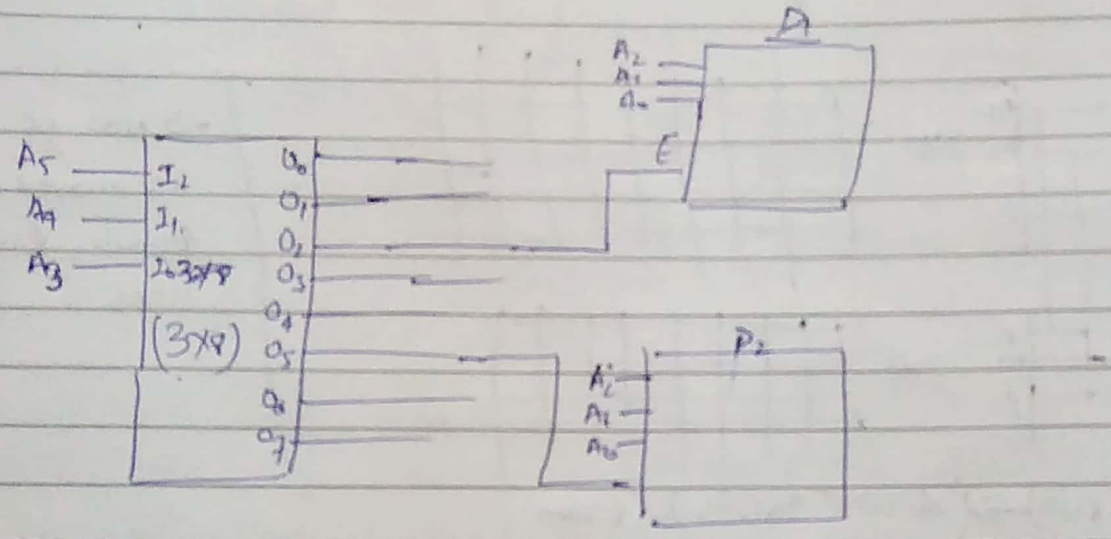
→ No. of vertical lines = size of data bus

→ No. of i/p given to decoder = size of add. bus

Date.....

(41) Finding the address ranges of devices

ex:- If $A_5 A_4 A_3 A_2 A_1 A_0$ are addresses, then what range does D_1 & D_2 get?



⇒ range of D_1:-

A_5 A_4 A_3 A_2 A_1 A_0

0 1 0 — — — → 2^3 { 8 address are possible }

fixed

range = { 16, to 24 }

range of D_2 :-

A_5 A_4 A_3 A_2 A_1 A_0

1 0 1 — — — → 2^3 { 8 add. are possible }

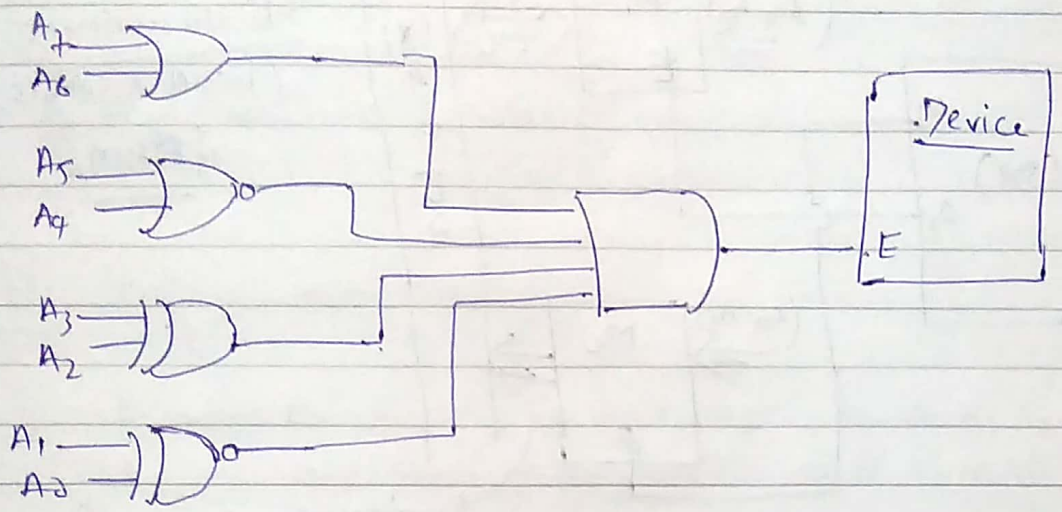
fixed

Range = { 40 to 48 }

Spiral

(92) Example on enabling a device:- Date...24/02/19.

ex:- Consider a device that is enabled using 8 add. bits as shown below. for how many addresses, the device is enabled.



(a) 1 (b) 8 (c) 12 (d) none.

=> All the i/p to AND gate should be '1' simultaneously.

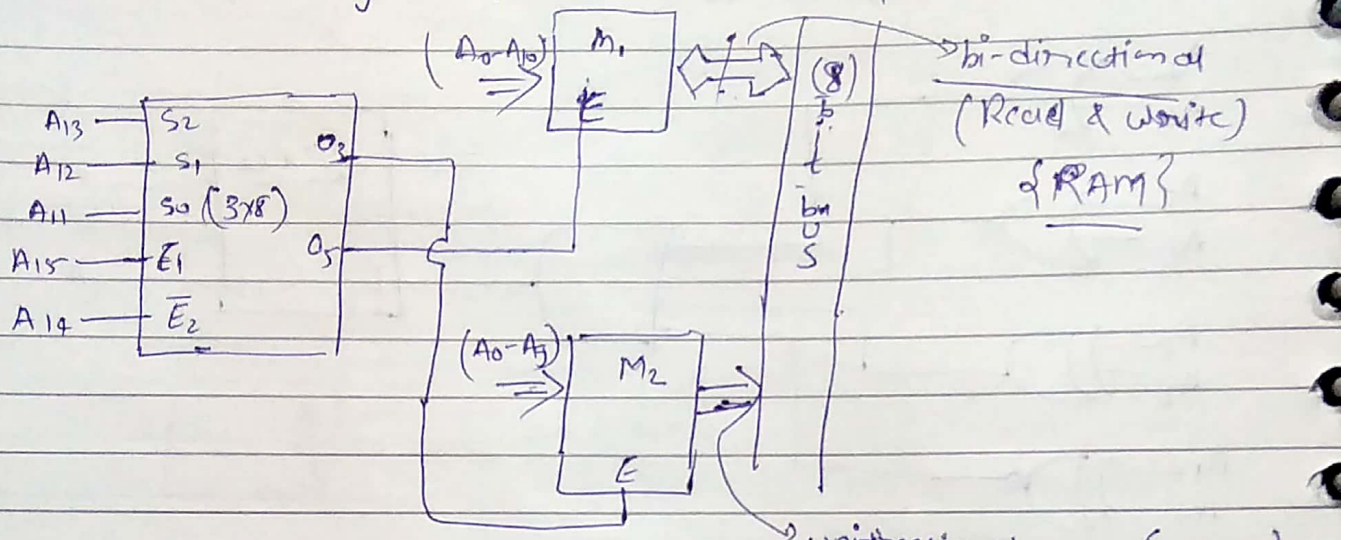
=> OR gate is 1 for 3 comb^n.
NOR " " " " 1 "
EX-OR " " " " 2 "
EX-NOR " " " " 2 ".

total = 3X2X2X1 = 12

Date.....

(43) Finding the address ranges of memory device:-

ex:- Consider the following interface of two memory devices with 3x8 decoder. Compute the nature & address range of each memory device.



for M_1:-

total Add = 2^{11}

& Size = 2^{11} x 8 bits ⇒ 2^{11} bytes

for M_2:-

total add. = 2^{10}

size = 2^{10} bytes {each add. is 8 bit long}

Add. ranges of M_1:-

Address lines: A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0

1 0 1 0 1 - - - - - - - - - - -

fixed

*** two enables
(one Active high & other is active low)

Date.....

So, range is :-

A_{15} A_{14} A_{13} A_{12} A_{11} A_{10} A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0

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

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

A800 H → AFFF H

Hexadecimal

Add. range for M_2 :-

A_{15} A_{14} A_{13} A_{12} A_{11} A_{10} A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0

fixed ← 1 0 0 1 1 Ø 0 0 0 0 0 0 0 0 0 0

1 0 0 1 1 Ø 1 1 1 1 1 1 1 1 1 1

→ don't care because A_{10} is not present

***

For Min. put Ø = 0 & For max put Ø = 1

9800 H — 9FFF H

***

→ Even though Ø is included in the range the no. of address possible are 2^{10} only, but each address (location), can be assigned in two ways depending on the value of Ø {0 or 1}

ex:- 10011 0 0000000000 } Same add. location
10011 1 0000000000 } can be addressed in two ways

Date.....

(49) Introduction to Encoders:-

→ They convert one code to another. {bigger No. syst to smaller no. syst.}

→ They perform lossless compression {encoding is nothing but a type of compression}

→ They are of types:-

(a) Non-priority (doesn't support simultaneous i/p activation). {only one i/p at a time is activated}

(b) priority (supports simultaneous i/p activation & used for interrupt servicing).

→ Encoder is used a sender side & decoder at receiver side {thus infom transmitted will be reduced}

→ Due to static priorities (in priority encoder), the lower priority i/p is exposed to starvation.

Encoders = $2^n \times n$

i/p, o/p.

ex:- Char. table for 4x2 encoder (Non-priority):-

I_0	I_1	I_2	I_3	B_1	B_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

$B_1 = \bar{I}_0 \bar{I}_1 (I_2 \oplus I_3)$ & $B_2 = B_0 = \bar{I}_0 \bar{I}_2 (I_1 \oplus I_3)$

→ Encoder is not a complete func^n because with 4 i/ps total 16 comb^{n}s are possible, but it uses only '4' of them.

Spiral

Date.....

(95) priority Encoders:-

→ Highest suffix i/p is given highest priority.

$I_3 > I_2 > I_1 > I_0$

Characteristics table:-

I_0	I_1	I_2	I_3	B_1	B_0
∅	∅	∅	1	1	1
∅	∅	1	0	1	0
∅	1	0	0	0	1
1	0	0	0	0	0

$B_1 = (I_3 + \bar{I}_3 I_2)$ ***

$\Rightarrow I_3 + I_2 = B_1$

$B_0 = I_3 + I_1 \bar{I}_2 \bar{I}_3$ ***

$\Rightarrow I_3 + \bar{I}_2 I_1 = B_0$

⇒ priority encoder is useful when more than one devices wants the access simultaneously, but only one should be given the service.

(96) Introduction to Hazards:-

Hazards:- problems in electronic devices.

Hazards

Temporary

→ when the problem persists for small amount of time.

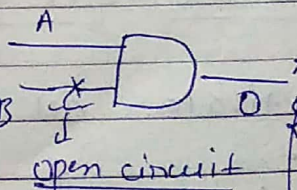
ex:- A, A, $\bar{A}$ → 1 {always 1}

because of propagation delay of NOT gate there could be O/P '0'. (instead of 1)

Permanent

→ It mainly occurs due to short circuiting or open circuiting

ex:-



open circuit

{always not matter what is A & B}

Date.....

→ Temporary hazard like in previous ex. can be taken care by adding a delay device to other wire. (whose delay is equal to NOT gate's delay)

→ permanent hazards can be removed by path sensitization. (ie. testing the doubtful path by disabling all the other paths)

(47) Hazards & test vectors:-

Hazards in digital circuit:-

→ The malfunction of a digital circuit is called hazard.
→ They can be permanent or temporary.
- permanent: due to open circuit or short circuit of connecting leads
- temporary: due to uneven delays of i/p

→ The hazards can be stuck at '0' (s-a-0) or stuck at 1 (s-a-1).

→ To check (s-a-0) or (s-a-1) single fault analysis is made using "path sensitization technique". This technique provides test vectors. {same as test cases of s/w}

Procedure for finding test vector:-

→ Inactivate all the other paths except the tested one.
→ for 'AND' or 'NAND' apply the logic '1' for inactivation.

Spiral

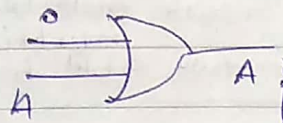
Date.....

→ for 'OR' or 'NOR' apply logic '0' for inactivation.

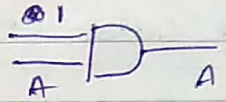
→ Apply the opposite logic value at the tested place.

→ The i/p combinations satisfying the above req. form the test vectors & test vectors is the subset of i/p comb's

ex:-



o/p depends on A when '0' is used as other i/p.

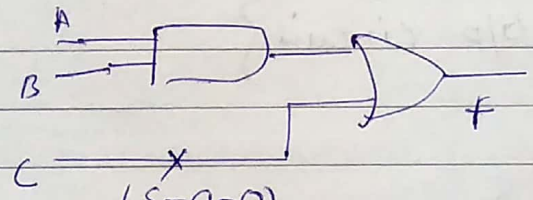


o/p depends on 'A' when 1 is used as other i/p

(48) Examples on test vectors:-

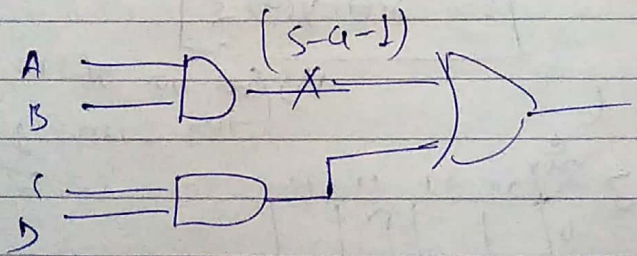
ex 1:- ~~Test vectors are~~

Test hazard (s-a-0):-



disconnect all the other i/p & check the link 'c' by giving 1, if o/p of OR gate is '1' then it is fine else not fine.

ex: (Test hazard s-a-1)



→ Inactivate all the links give '0' to the (s-a-1) link & check the o/p if '0' then fine else it is (s-a-1)

***

→ Make the o/p depend on only the link which is being tested.

Spiral

Date.....

(49) Half adder-2 :-

→ Half-adder → (for adding two bits)
- used to construct → full adder → (for adding three bits)
- used to construct → Binary adder → for adding two ~~multi bit~~ multi bit binary nos. ex:- two 4 bit binary no.

Half-adder :- {two bit addition} {'2' i/p & '2' o/p circuit}

X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

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

$C = XY$

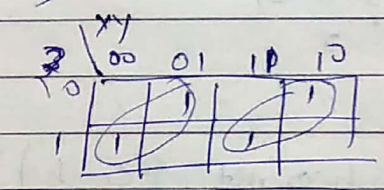
2 i/p

(50) full-adder :- { 3 i/p, 2 o/p circuit }

→ 3-bit adder

X	Y	Z	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

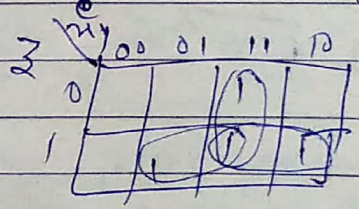
S



$S = x \oplus y \oplus z$

(i) (odd no. of 1's then sum = 1)

C



$C = xy + yz + xz$

(ii) (carry will be generated if any two i/ps are 1.) ***

→ for adding LSB of two Binary no. Half adder is used
~~& after that full adder~~

ex:- 0 1 1 1
0 1 0 1 → half adder.

Date.....

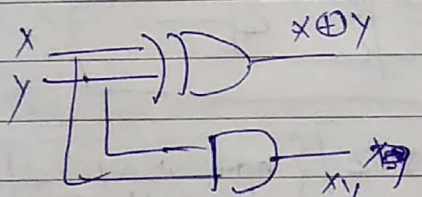
Carry can also be expressed as:-

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

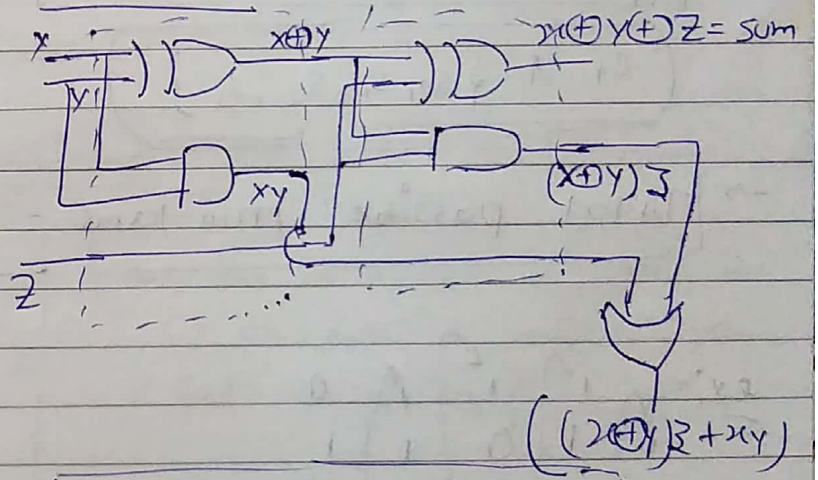
(if $z = 1$ & any one of x & y are 1) or (if $xy = 1$)

→ This form is useful to design a full adder using two half adders.

Half adder:-



full adder:-



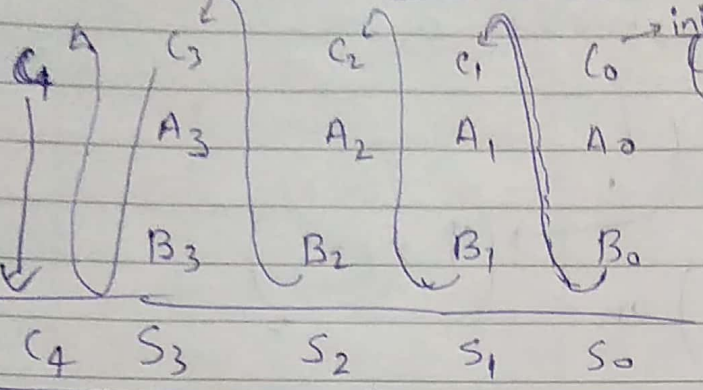
→ 2- HA & ~~a~~ OR gate can be used to design a full adder

(5.1) Ripple Carry ~~Adder~~:-

→ This is why 'full' adder is called full because it is designed using two half adders.

Date.....

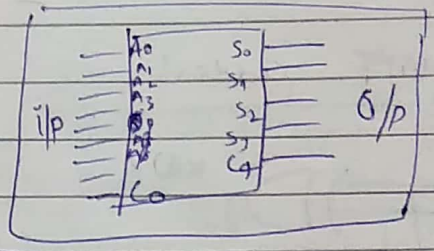
(51) Ripple carry adder :-



{ C_{i+1} is carry generated from addition of A_i & B_i }

→ So, i/ps to implement this adder are '9'.
{ ie. (A_0, A_1, A_2, A_3) & (B_0, B_1, B_2, B_3) & C_0 }

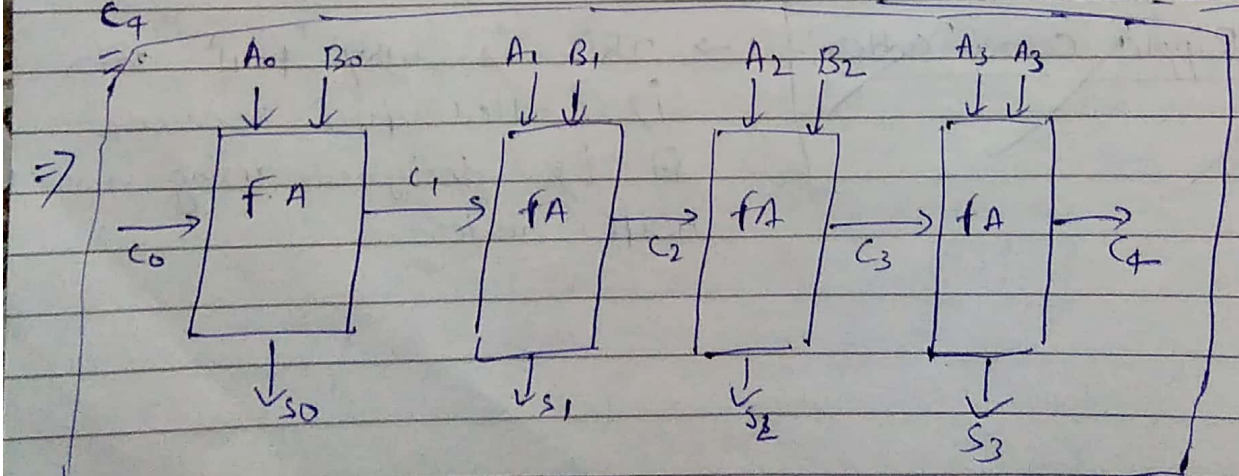
→ O/Ps obtained are '5'.
(C_4 & $\{S_0, S_1, S_2, S_3\}$)



→ Total possible min-terms = $2^9 = 512$ { not very difficult to implement }

ex:- 1 1 0 1 1 + 1 1 0 1 0 = 1 1 0 0 0

So, FA are used to implement, this as building block (only 3-bits are added at any phase at max)



Spiral
FA is 3 i/p & 2 O/p device

(Full adders to perform sum)
(called Ripple carry adder)

Date.....

→ The problem with ~~this phe~~ Ripple carry adder is ~~that~~ that the carry passes slowly through all the phases. (And next phase has to wait for that, so it takes more time).

→ Time taken to add to n-bit nos. = $n \times T_{FA}$

T_{FA} = time taken to pass 1 FA or Propagn time of 1 FA

→ To resolve this problem of waiting for carry ~~of~~ previous stage, some circuit is used to generate carry prior, that adder is called carry lookahead adder.

(52) Carry lookahead adder:-

$C_1 = C_0(A_0 \oplus B_0) + A_0B_0$ { any one of A_0 or B_0 is 1 along with C_0 or A_0B_0 is '1' to get get carry C_1 }

$C_2 = C_1(A_1 \oplus B_1) + A_1B_1$

$C_3 = C_2(A_2 \oplus B_2) + A_2B_2$

$C_4 = C_3(A_3 \oplus B_3) + A_3B_3$ ⇒ Try to convert everything in the form of C_0 & since C_0 is directly available initially we don't need to wait for carry of previous stage.

let ~~$G_i = A_i \oplus B_i$ & $P_i =$~~

let $G_i = A_iB_i$ & $P_i = A_i \oplus B_i$.

Generator { generated by stage i i.e. A_iB_i }

propagation { carry has propagated from the last stage i }

Date.....

Substituting the values of G_i & P_i in the equations

$C_1 = C_0P_0 + G_0$

$C_2 = C_1P_1 + G_1$

$C_3 = C_2P_2 + G_2$

$C_4 = C_3P_3 + G_3$

=

⇒ NOW, $C_1 = C_0P_0 + G_0$

$C_2 = (C_0P_0 + G_0)P_1 + G_1$

$\Rightarrow C_0P_0P_1 + G_0P_1 + G_1$

to get carry in stage '2' carry of stage '0' could have propagated through stage 0 & 1

or

carry would have been generated at stage 0 & got propagated through stage 1

or

carry is generated in stage '1'

$C_3 = C_0P_0P_1P_2 + G_0P_1P_2 + G_1P_2 + G_2$

$C_4 = C_0P_0P_1P_2P_3 + G_0P_1P_2P_3 + G_1P_2P_3 + G_2P_3 + G_3$

⇒ Every carry C_1, C_2, C_3, C_4 are dependent on only C_0 & (G, P). {NOT on previous stage carry}

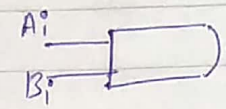
⇒ In Gate, questions can be asked on these equations to write any part of the equation, so don't remember it try to understand it.

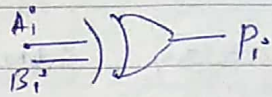
=

Spiral

Date.....

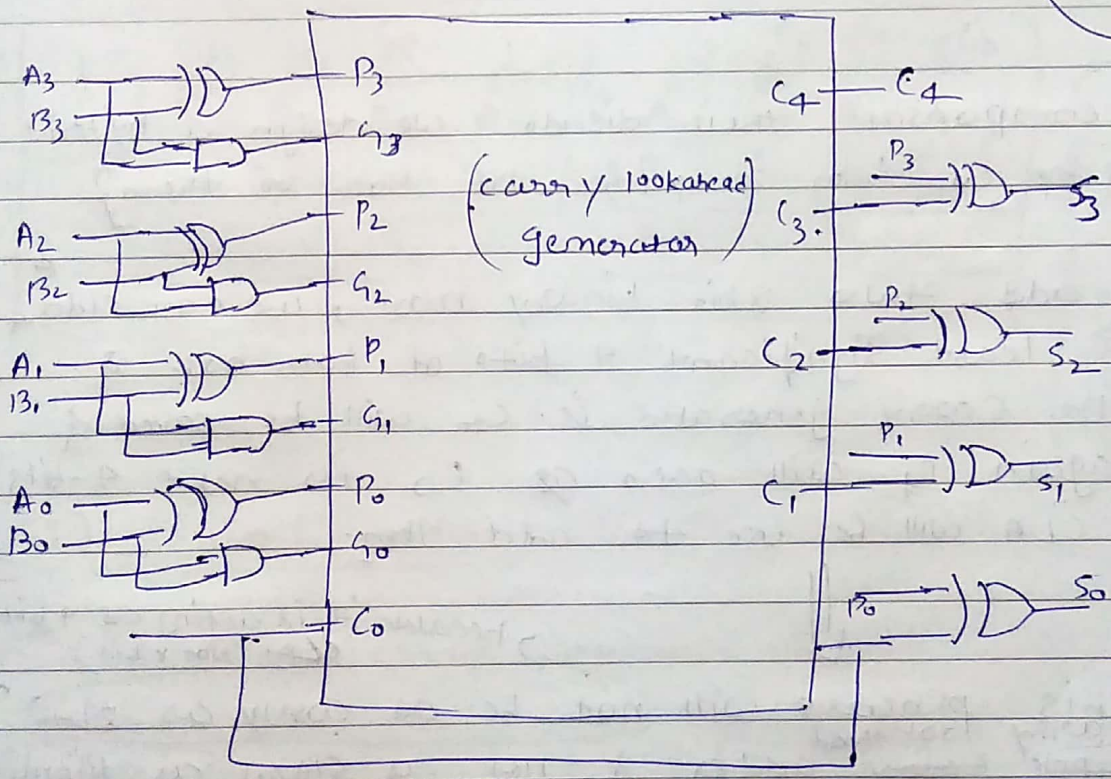
(53) Carry lookahead implementation:-

→ G_i => 

P_i => 

→ C_1, C_2 & C_3 can be generated using AND-OR realisation. & they will be available to use at each stage instantly. & C_4 need not be computed it will not be required in any steage for addition)

$S_i = A_i \oplus B_i \oplus C_i$
$\Rightarrow (P_i \oplus C_i)$



Carry lookahead adder {for '4' bit nos.}

→ Carry lookahead generator:- It is the circuit inside CLA which is used to generate (C_1, C_2, C_3) using ~~g_i & P_0 &~~ (g_i, P_i & C_0).

(5q) Hybrid adder :-

Date.....

Ripple carry adder :-

(i) Adv :-	(ii) disadv :-
(a) Simple to design & cheaper	→ slower.

Carry lookahead adder :-

(i) Adv :-	(ii) disadv :-
(a) faster	(a) complex to design & costly.

→ So, to componsate their disadv. we design a hybrid adder. {by combining properties of both of them}

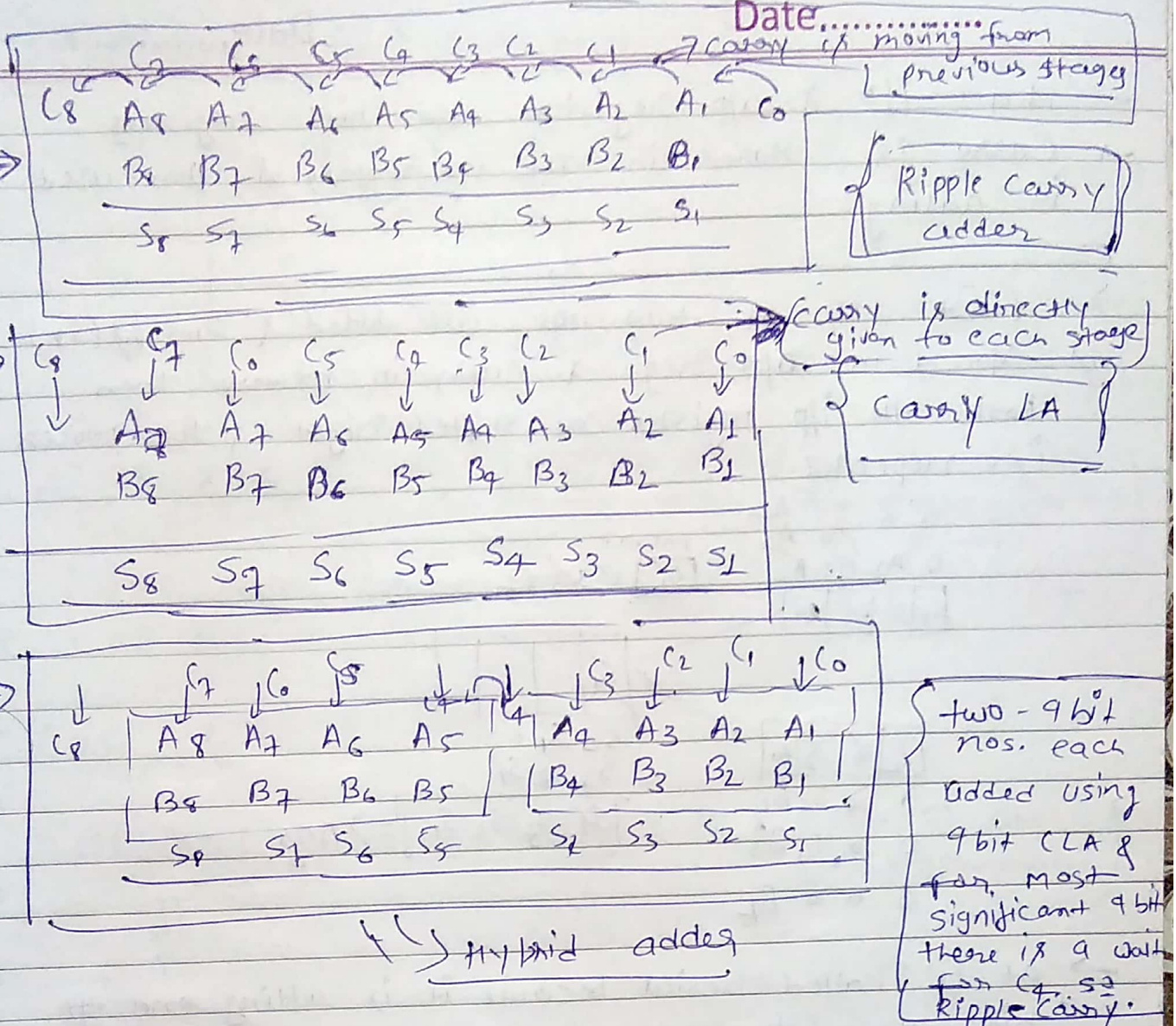
ex :- To add two 8 bit binary nos., we can add (using CLA) least significant 4 bits of both nos. & then the carry generated i.e. C_4 will be saved & then again C_4 will act as C_0 to next 4-bits & then CLA will be use to add them.

⇓

{ This process will (not be as costly as 8 bit Carry lookahead Ripple Carry adder & not as slow as Ripple Carry Adder. } ← because it is acting as 4 bit CLA (Not 8 bit)

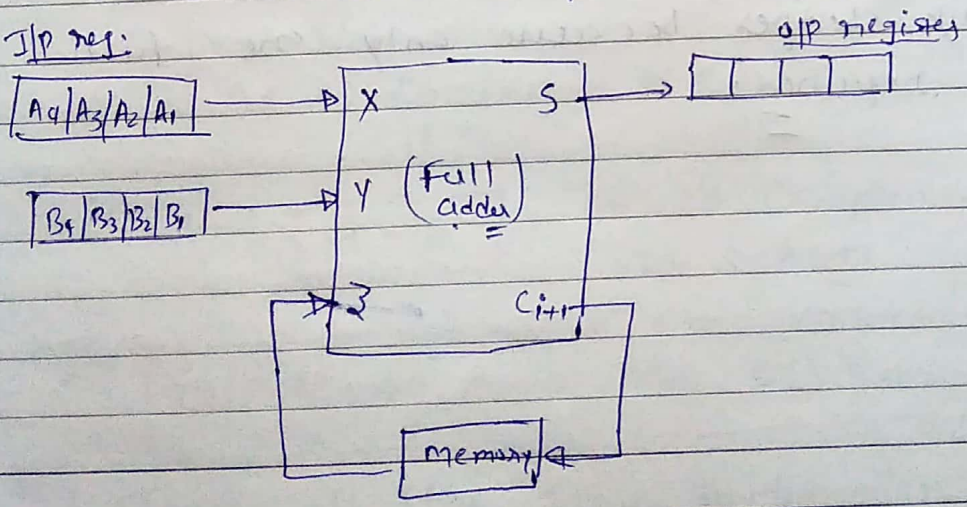
→ It is acting as RCA because for Most significant 4-bits we are waiting for the carry. (C_4).

Date.....



(55) Serial adder:-

→ It comes under sequential circuit.



→ both binary nos. are stored in i/p registers

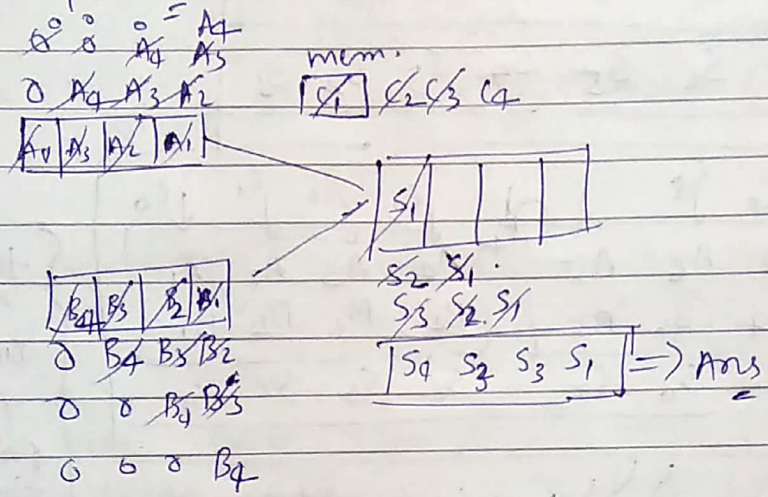
Spiral

Date.....

→ Shift i/p & o/p registers are shift registers.

→ Carry is stored in the memory & then used in adding

→ first LSB of two nos. are added & sum (S1) is stored in o/p reg. & carry in memory then both the i/p register are shifted right & the process is repeated.



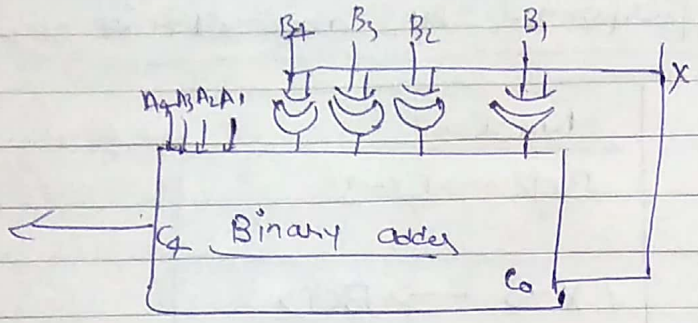
→ It is called serial because it is adding one bit after the other.

→ It is very slow (slower than any other adder)

→ But, It is cheaper because only one full adder is required.

Date.....

(5C) Binary adder / subtractor :-



Case - 1

⇒ If $X = 0$ ⇒ then $B_i \oplus X = B_i$ (i.e. the o/p of ex-OR gates are the i/ps B_i itself)

(then adder perform sum of two nos.)

Case - 2

→ If $X = 1$ ⇒ then $B_i \oplus X = \overline{B_i}$ (i/p B_i are complemented)

All the bits are complemented ie. 1's Complement of 'B'

→ A + (1's complement of B) + 1 (because C_0 is also 1)

(A + 2's Complement of B)

$A - B$ ⇒ in 2's Complement no. system.

***

So, binary adder acts as subtractor as well

***

Thus all the basic mathematical opern can be performed using a single circuit.

Spiral

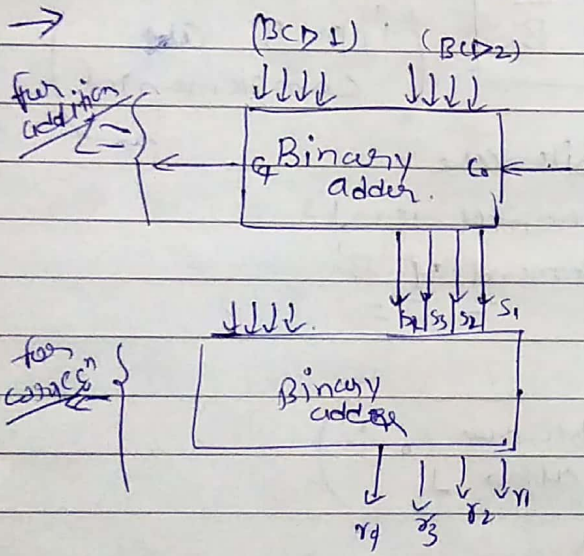
Date... 25-02-19

Other operations performed by same ckt.:-

A	B	X	function
(a) BCD	0011 (3)	0	BCD → EX-3
(b) EX-3	0011 (3)	1	EX-3 → BCD
(c) 1001 (9)	BCD	1	9's complement of BCD.
(d) 1010 (10)	BCD	1	10's complement of BCD.

(57) BCD adder:-

→



→ In case of addition of two BCD nos. the o/p is invalid when:-

(i) carry is generated (i.e. $C_4 = 1$)
or
(ii) sum is > 9.

→ For these invalid o/ps correction is done, by giving a i/p of 6' (0110) to 2nd binary adder (i.e. 6' is added)

→ Else (0000) will be given as i/p to 2nd binary adder.

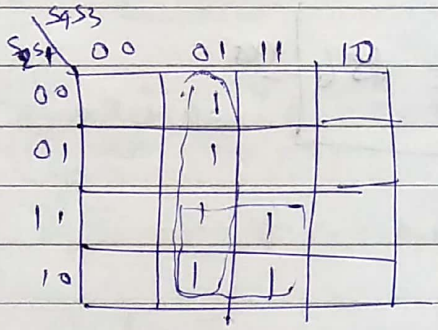
Cases when correction is required:- Date.....

(i) When $S_4S_3S_2S_1 > 9$

	S_4	S_3	S_2	S_1	f
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

→ f should ~~always~~ give value 1 when sum > 9 else '0' {1 represent that sum > 9}

K-map for 'f':-



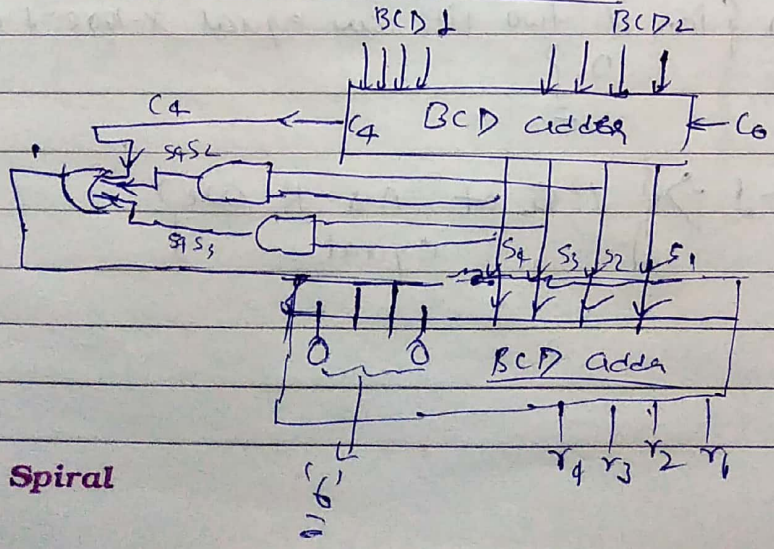
⇒ $f = (S_4S_3 + S_4S_2)$

(i.e. either S_4, S_3 should be '1' or S_4, S_2 should be '1' for sum to be > 9)

(ii) when $C_4 = 1$

F = 1

So, BCD adder will become:-

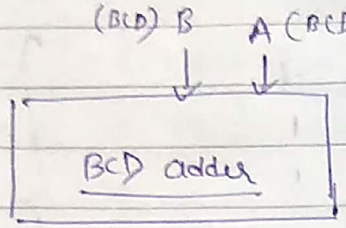


Spiral

Date.....

(58) Invalid Combinations for BCD Adder:-

→ Invalid combinations possible at i/p of BCD adder.



{where A & B are 4 bit nos}

(Invalid combinations of 'A' & B) = {Total combinations − valid comb^n}

⇒ $16 \times 16 - (10 \times 10) \Rightarrow 156$

(0-9 for A) ((0-9) for B)

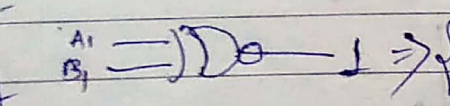
59 Invalid comb^n of two BCD nos. A & B = 156

(59) 2-bit Comparator:-

→ Instead of using convention method (or Brute force) to compare two nos. of two bits by writing all possible I/P comb^n, we'll use some other method.

→ In this method both the nos. are compared bit-by-bit. using X-NOR gates {ie. if two bits are equal X-NOR = 1 else 0}.

ex:-



⇒ {LSB of A & B are equal.}

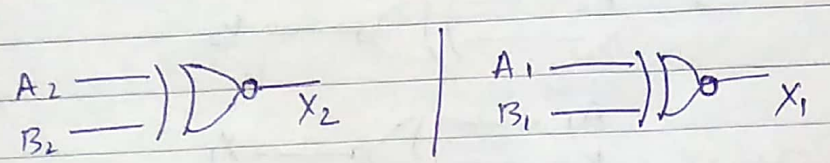
$A = A_2 A_1$

$B = B_2 B_1$

Date.....

1-bit Comparator :-

Let $A = A_2 A_1$ & $B = B_2 B_1$



Case 1 :- $(A = B)$

if $X_2 = 1$ & $X_1 = 1$

i.e. $X_1 \cdot X_2 = 1$ ⇒ $(A_2 \odot B_2) \cdot (A_1 \odot B_1) = 1$ **

Case 2 :- $A > B$

if $(A_2 > B_2)$ or $(A_2 = B_2$ & $A_1 > B_1)$
(1) (0) (1) (0)

i.e. $= (A_2 \bar{B}_2 + X_2 \cdot A_1 \bar{B}_1)$

⇒ $A_2 \bar{B}_2 + (A_2 \odot B_2) \cdot A_1 \bar{B}_1$ **

Case 3 :- $A < B$

if $(A_2 < B_2)$ or $(A_2 = B_2$ & $A_1 < B_1)$
(0) (1) (0) (1)

i.e. $\bar{A}_2 B_2 + (X_2 \cdot \bar{A}_1 B_1)$ ⇒ $\bar{A}_2 B_2 + (A_2 \odot B_2) \bar{A}_1 B_1$ ***

Date.....

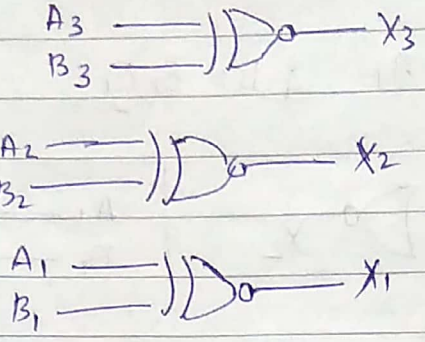
(6) 3, 4 bit Comparator :-

(i) 3-bit

$A = A_3 A_2 A_1$

$B = B_3 B_2 B_1$

~~$C = C_3 C_2 C_1$~~



Case 1 :- $(A = B)$

$$X_3 \cdot X_2 \cdot X_1 = 1$$

Case -2 :- $(A > B)$

$$\left(A_3 \bar{B}_3 + X_3 \cdot A_2 \bar{B}_2 + X_3 X_2 A_1 \bar{B}_1 \right) = 1$$

Case-3 :- $(A < B)$

$$\left(\bar{A}_3 B_3 + X_3 \bar{A}_2 B_2 + X_3 X_2 \bar{A}_1 B_1 \right) = 1$$

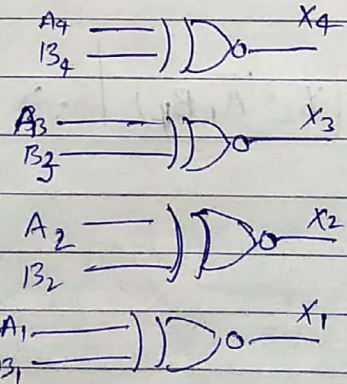
(ii) 4-bit Comparator :-

$A = A_4 A_3 A_2 A_1$

$B = B_4 B_3 B_2 B_1$

Case 1 :- $(A = B)$

$$X_4 \cdot X_3 \cdot X_2 \cdot X_1 = 1$$



→ EX-NOR gate is used for comparison on designing comparator

Date.....

Case - 2 $(A > B)$

$$\left(A_4 \bar{B}_4 + x_4 \cdot A_3 \bar{B}_3 + x_4 \cdot x_3 \cdot A_2 \bar{B}_2 + x_4 \cdot x_3 \cdot x_2 \, A_1 \bar{B}_1 \right) = 1$$

Case-3 $(A < B)$

$$\left(\bar{A}_4 B_4 + x_4 \bar{A}_3 B_3 + x_4 x_3 \bar{A}_2 B_2 + x_4 x_3 x_2 \bar{A}_1 B_1 \right) = 1$$

→ This method is better than tabular method in which for 4-bit comparison the no. of entries will be 256.

(61) Analysing the cases of comparators :-

→ for two binary nos. $(A \& B)$ of 2-bit each.

(i) $A = B \Rightarrow$ 4 cases
(ii) $A < B \Rightarrow$ 6 cases
(iii) $A > B \Rightarrow$ 6 cases

total combⁿs possible $= 2^{2n} \Rightarrow (2^4 = 16)$

→ for two binary nos. of n-bit each :-

(i) No. of cases for which $A = B = 2^n$
(ii) " " " " " $A < B = \dfrac{(2^{2n} - 2^n)}{2}$
(iii) " " " " " $A > B = \dfrac{(2^{2n} - 2^n)}{2}$ ***

Spiral

Date.....

(63) Time complexity of Ripple carry adder:-

Time complexity = $\Theta(n \times T_{FA})$

T_{FA} → time taken by '1' full Adder

n → (no. of stages or no. of bits in nos.)

⇒ **Time complexity = $\Theta(n)$** ⇒ assuming '$T_{FA} = K$'

(64) Time complexity of carry-lookahead adder:-

→ In carry lookahead adder the most time consuming operation is finding 'C_4'. While the value of C_4 is calculated the value of C_3, C_2, C_1 will already be calculated because they require less computation.

$$C_4 = C_0P_0P_1P_2P_3 + G_0P_1P_2P_3 + G_1P_2P_3 + G_2P_3 + G_3$$

(↑) for 4-bit nos.

→ And in C_4 also the 1st term will decide the time Complexity of C_4 & since C_4 decides the time Complexity of CLA the entire time complexity of CLA depends on the 1st term.

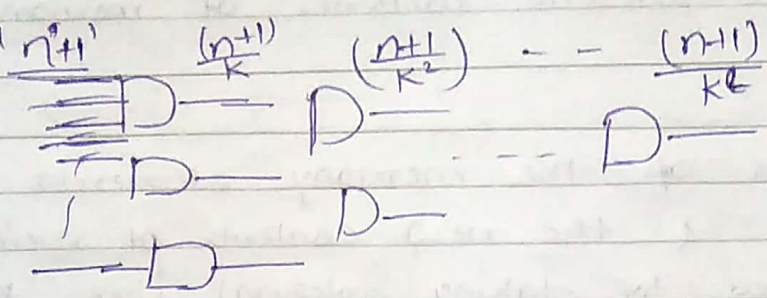
→ The no. of variable in 1st term of C_4 while adding two 'n' bit nos. will be '$n+1$'.

→ The time depends on type of 'AND' gate used for 1st term ie. How many fan-ins are there in AND gate. (ex. AND gate with 2 fan-in is slower than 3-fan-ins), because lower fan-ins ↑ the no. of levels.

Spiral

Date.....

→ The time complexity depends on no. of levels.



{→ AND gates with fan-in = k & no. of bits in no. that are added = n

→ No. of var. in 1st term of $v_n = n+1$

$$\frac{n+1}{k^L} = 1$$

$$\Rightarrow k^L = n+1 \Rightarrow \boxed{L = \log_k^{n+1}}$$

where L = no. of levels

Total time = $\log_k^{(n+1)} \times T_{and}$ → (time for each AND gate)

$$\boxed{\text{Time complexity} = O\left(\log_k^{(n+1)}\right)}$$ { assuming T_{and} as const. 'k' }

k = fan-in of AND gate

n = no. of bits in the numbers

→ As k (ie fan-in) ↑ then time complexity ↓.

→ This is obtained assuming the OR gate has 'n+1' fan-in. (for n+1 terms in c_n)

→ If fan-in of AND gate is 'b' & OR-gate is 'a' then,

$$\text{time complexity} = O\left(\left(\log_b^{n+1}\right) + \left(\log_a^{n+1}\right)\right)$$

Spiral

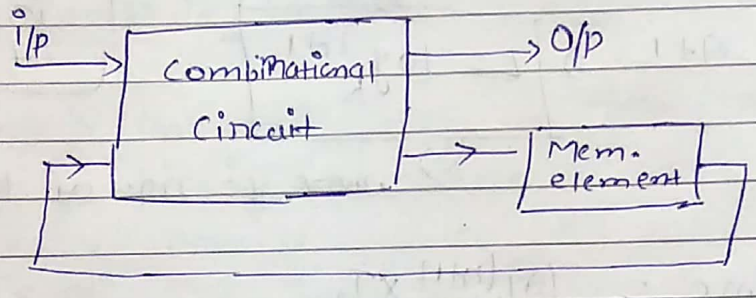
(4) Sequential Circuits

Date..05/03/19...

(1) Introduction to sequential ckts:-

→ The external o/p of seq. circuit depends on external i/p's & on present contents of "memory elements"

→ The present contents of the memory elements are called present state & the new contents of memory elements are obtained by taking external i/p's & present state; this is called next state.



(seq. circuit)

(2) Latch & flip-flop:-

→ 1-bit memory cell:-

A memory element is some medium in which one bit of infor^n can be stored or retained until necessary, & can be replace there bytes its contents can be replaced by new values

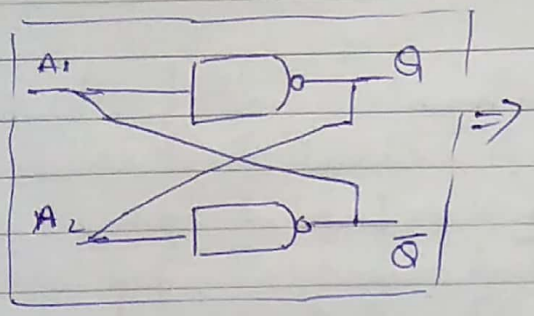
→ The basic binary or digital memory ckt. is known as "flip-flops". Millions & billions of ff are used to build a memory device like RAM)

→ It has 2 stable states which are known as '1' or '0'. So it is also called "bistable Multi-vibrator".

→ To build ROM Encoder & ROM matrix can be used & only read can be performed, but RAM are made up of flip-flops, so read & write can be performed.

Date.....

Latch (or Lock) :-

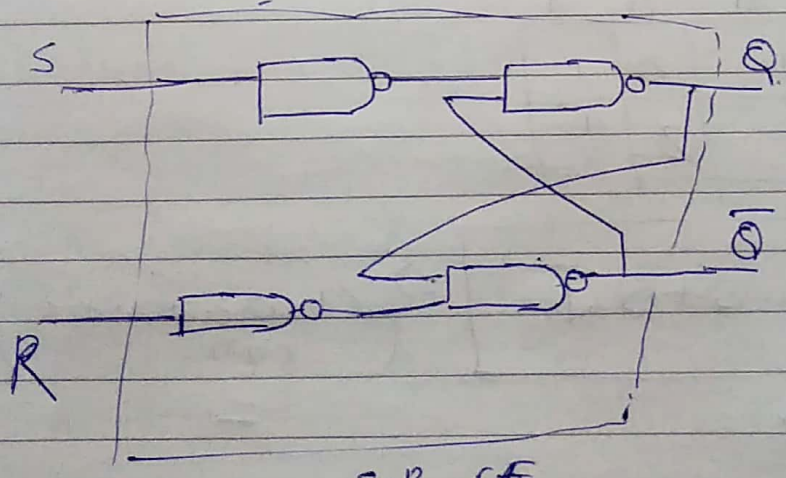


⇒ There is no i/p lines / o/p lines for this ckt, i.e. it can either lock '0' (or latch '0') or lock '1'. (Q/q to value of Q 'Q' if Q=1, 1 is locked)

→ latch does not have any i/ps or o/p. ie. values can not be changed by the user.

(3) S-R flip-flop :-

→ To a latch if i/ps can be given by a user & o/p is recieved accordingly then it becomes a flip-flop.



S-R ff

⇒ o/p 'Q' of next state ie. Q_{n+1} depends on i/ps S, R & o/p of present state Q_n.

ie $Q_{n+1} = f(S, R, Q_n)$

Spiral

Date.....

~~Truth table~~ Characteristic table:-

S	R	Q_n	Q_{n+1}	
0	0	0	0	Latch mode
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	∅	invalid {because values of Q_{n+1} & $\bar{Q}_{n+1}$ are always '1' irr. of Q_n when S & R = 1, which is invalid}
1	1	1	∅	

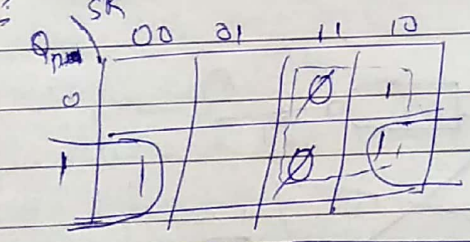
=

***⇒ So, S-R ff can be used only in 3 - modes ie. Latch, Reset & Set.

(S=R=0) (S=0, R=1) (S=1, R=0)

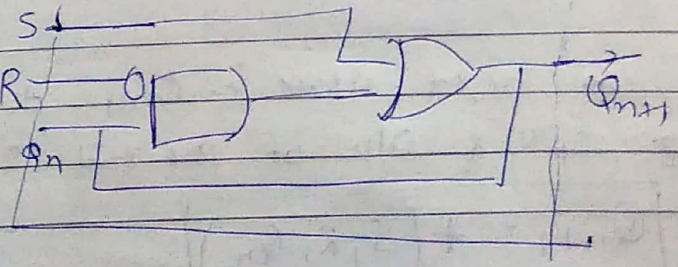
***⇒ S-R ~~ff~~ f.f. is invalid when S = R = 1, (disadv. of S-R ff.)

=



$$Q_{n+1} = S + Q_n\bar{R}$$ ⇒ (Characteristic eqn)

=



Spiral

Date..07-03-19

Excitation table:- {what is i/p should be given so, that the o/p behaves the given way}.

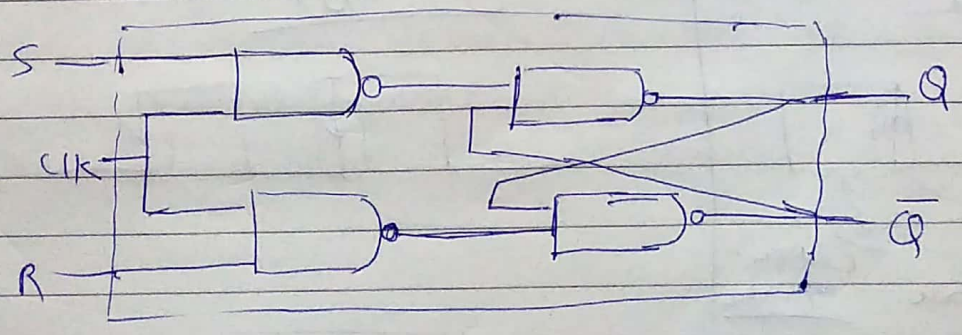
Q_n	Q_{n+1}	S	R
0	0	0	Ø
0	1	1	0
1	0	0	1
1	1	Ø	0

{i/ps are also called the excitation}

function table:- {Minimized characteristic table of next state, i.e. o/p is the func^n of present state o/p.}

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	Ø

(4) **Clocked flip-flop:-**



⇒ when clock i/p '0' S-R f.f works in latch mode.

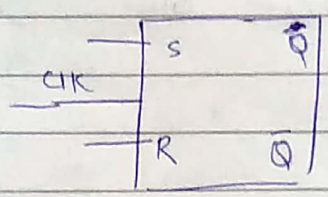
→ Clocked flip-flop is used to avoid noise or interference which is encounted during ~~dat~~ signal transmission, Instead of activating ff all the time during transmission we activate it for small time, so that Prob. of noise or interferance is less & that is where clock is needed.

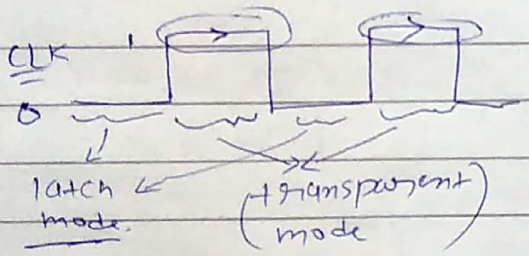
Date.....

⇒ O/p of a clocked f.f. will depend only when the clock is triggered (or transparent mode) else O/p will be constant (latch mode), ie. O/p will not depend on i/p.

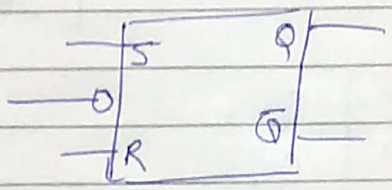
Types of clock-triggering:-

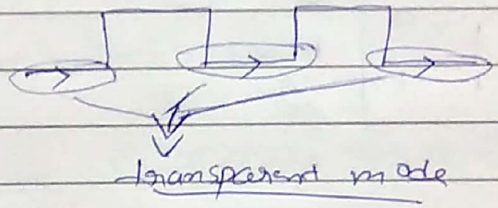
(i) +ve - level



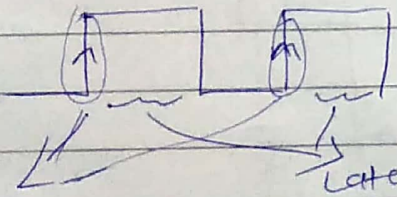


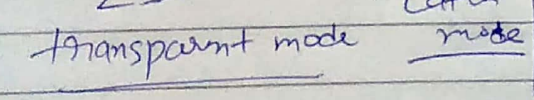
(ii) -ve level

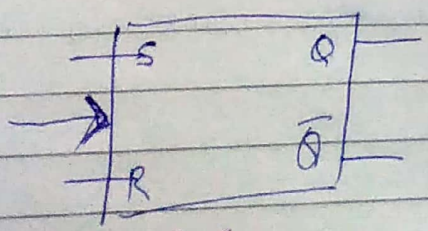




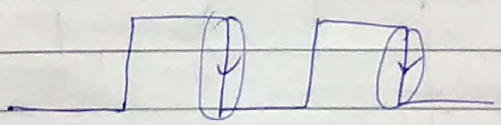
(iii) +ve-edge

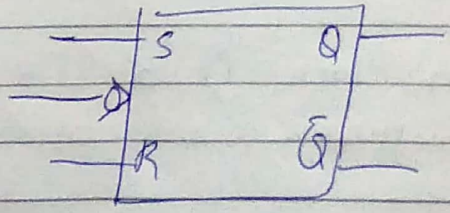






-ve edge



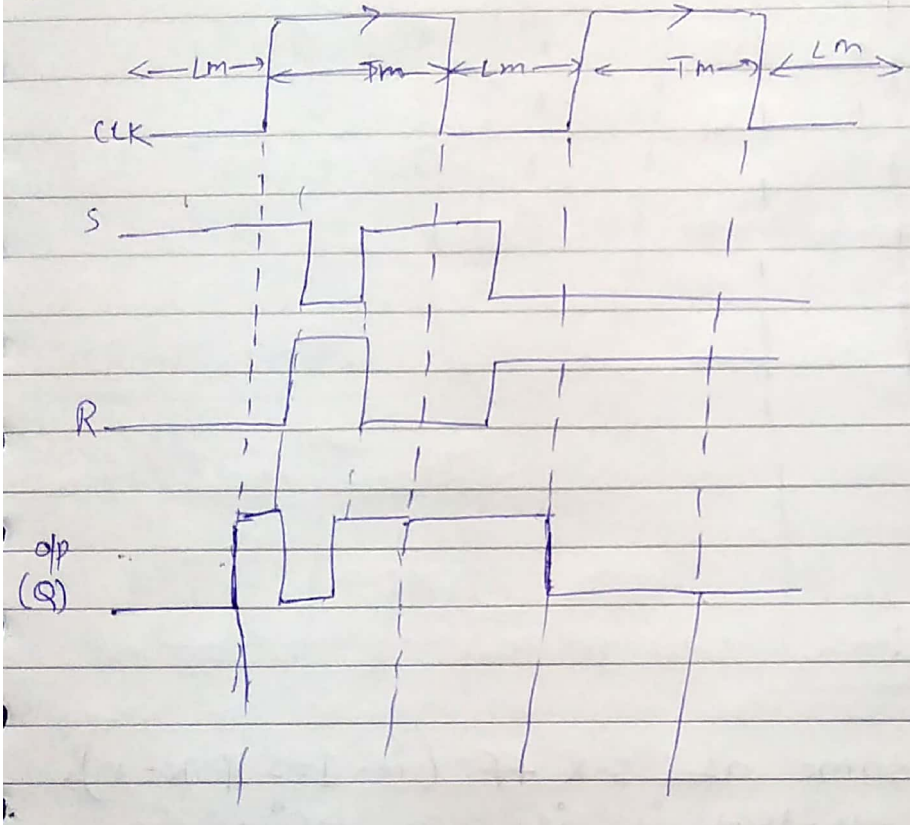


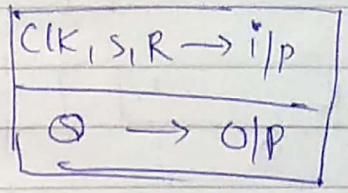
Spiral

Date.....

** ⇒ Edge-triggering are better than level triggring because the ff are in transparent mode for very less amount of time thus chances of getting noise or interforance is very low.

(5) positive level triggered :-



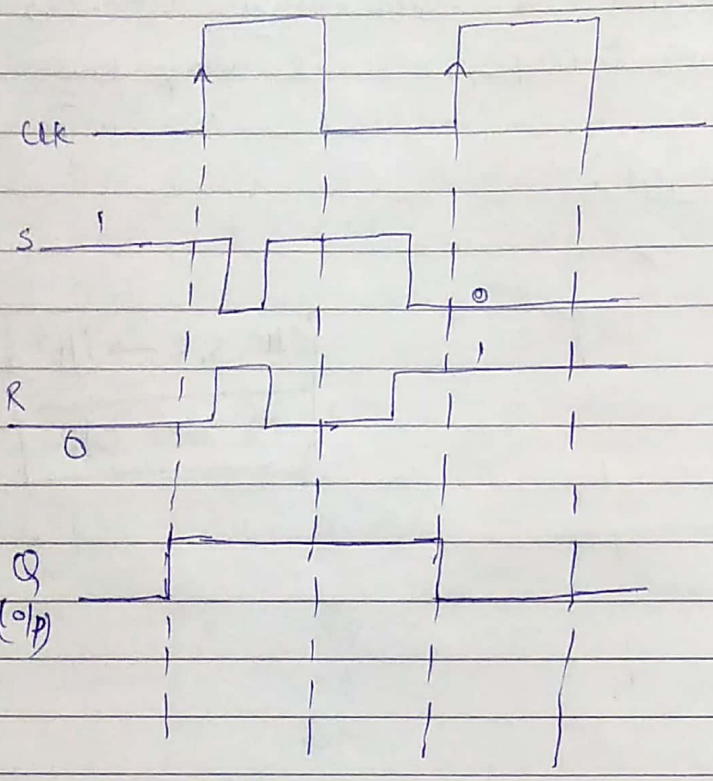


→ One more benifit of clock trigerring is it takes care of uneven delays in the ~~of~~ i/p present is transparent mode.

Date.....

(6) Edge triggered :-

+ve - edge triggered :-



Clk, S & R → i/p

Q → o/p

(7) J-K flip-flop :-

→ J-K ff is same as S-R ff (ie. J=S & K=R). Only diff. is that it is also defined for J=K=1 (which is invalid in S-R). ~~& the ff. performs~~

→ The J-K ff performs complementation for J=K=1

function table :-

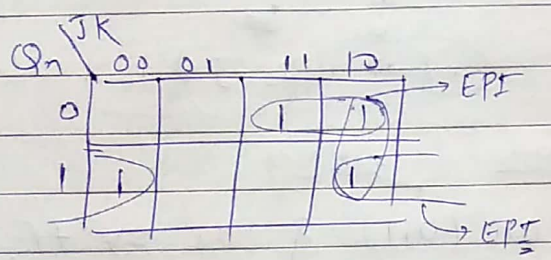
J	K	Q_{n+1}	
0	0	Q_n	→ latch
0	1	0	→ Reset
1	0	1	→ Set
1	1	$\overline{Q_n}$	→ Complement.

Date.....

Characteristic table :-

J	K	Q_n	Q_{n+1}	
0	0	0	0	Latch
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Complement
1	1	1	0	

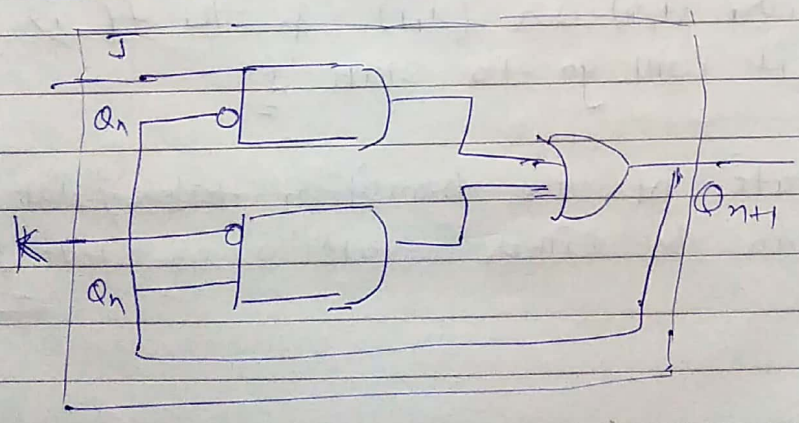
Char. eqn :-



So, $Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$

↳ Char. eqn

Realisation :-



Spiral

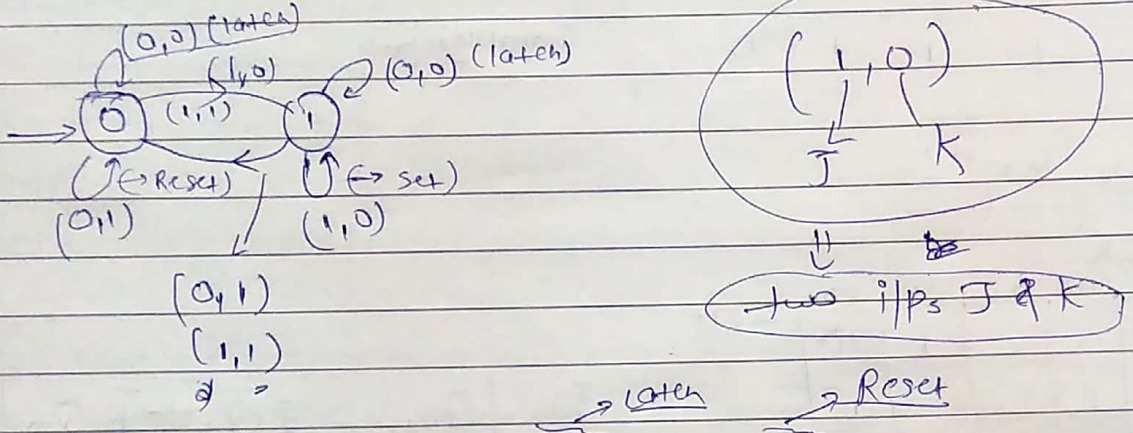
→ No. of states in state diag. of ff is always '2' ie. '0' & '1'

Date.....

Excitation table :-

Q_n	Q_{n+1}	J & K
0	0	0 Ø
0	1	1 Ø
1	0	Ø 1
1	1	Ø 0

State diagram :-



→ Latch → Reset

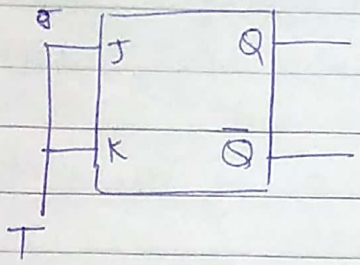
→ when J & K are (0,0) or (0,1) & the ~~sy~~ ff is in state '0' the it will remain in state '0'

→ Set → Complement

→ when J & K are (1,0) or (1,1) & the ff is in state '0' it will go to state '1'.

→ The initial state of the ~~transition table~~/state state diag. can be either state '0' or state '1'

(8) T-flip flop:- (Toggle ff) Date.....



{ both J & K are made a single i/p 'T' }

#func^n table:-

T	Q_{n+1}
0	Q_n → latch
1	$\bar{Q}_n$ → Complement

#Chara. table

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

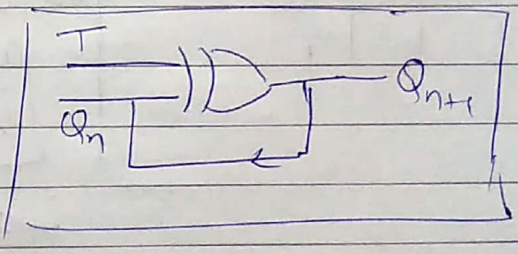
#Excitation table:-

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

#Chara. eq^n:-

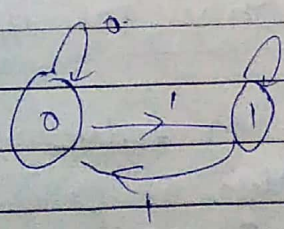
$$Q_{n+1} = \bar{T}Q_n + T\bar{Q}_n = T \oplus Q_n$$

Realisation



→ There are only two operations latching & Complementing that is why it is called toggle ff.

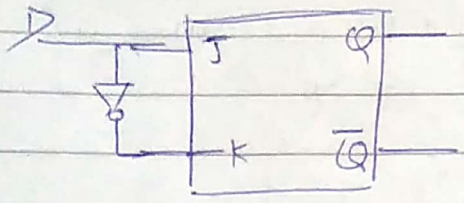
State diag:-



{ only one i/p i.e. T }

Date.....

(9) ▷ - flip flop :-



function table :-

▷	Q_{n+1}
0	0
1	1

characteristic table :-

▷	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Excitation table :-

Q_n	Q_{n+1}	▷
0	0	0
0	1	1
1	0	0
1	1	1

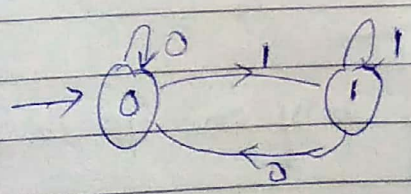
Char. eqn :- $Q_{n+1} = ▷\bar{Q}_n + ▷Q_n$

$$Q_{n+1} = ▷$$

⇒ (***) It's giving the same o/p (ie. '▷') as the i/p, after some delay; this is why it is called ▷ - ff.

→ ▷ - ff has got lot of appln like acting as delay device for uneven propagn delays.

State diag :-



Only one i/p '▷'

→ when value of ff. is '0' & value of ▷ is '1' then value of ff. will be '1'

Date.....

(10) Example on flip-flop ↓:-

(Q) If char. eqn of a flip-flop is $Q_n = \bar{X}_1\bar{Q} + \bar{X}_2 Q$ define the behaviour of this.

⇒ Make func^n table using the given eqn:-

X_1	X_2	Q_n	
0	0	1	→ set
0	1	$\bar{Q}$	→ Toggle
1	0	Q	→ Latch
1	1	0	→ Reset.

Ans

Char. table:- (using func^n table)

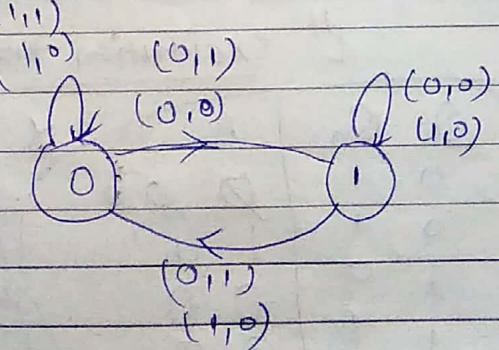
X_1	X_2	Q	Q_n
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

⇒

Excitation table:-

Q	Q_n	X_1	X_2
0	0	1	∅
0	1	0	∅
1	0	∅	1
1	1	∅	0

State diag:-



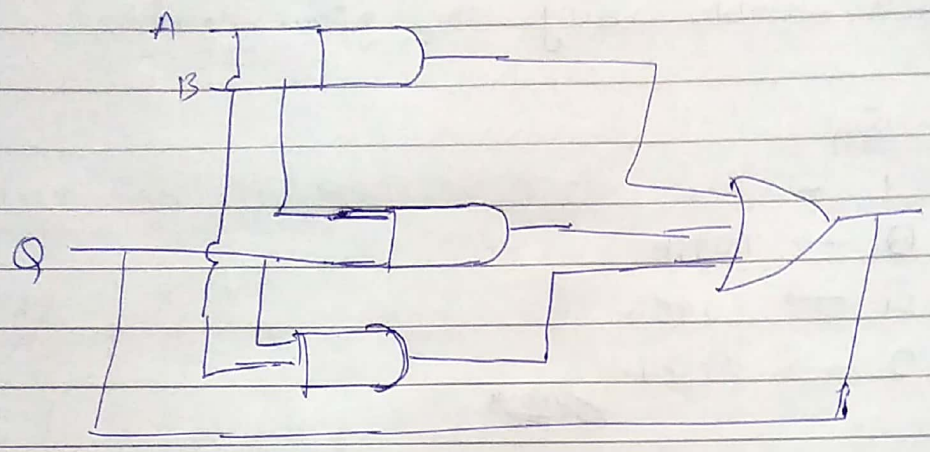
→ ***

State diag. is easy to draw with the help of char. table

Date.....

(11) Example on flip-flop (2):-

(Q) Consider the following realisation & construct excitation table.



Char. eqn = $AB + BQ + AQ = Q_n$

Func^n table:-

A	B	Q_n	
0	0	0	→ set
0	1	Q	latch
1	0	Q	
1	1	1	→ Reset

Char. table:-

A	B	Q	Q_n
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	Ø 1
1	1	1	1

excitation table:

Q	Q_n	A	B
0	0	0	Ø
0	1	1	1
1	0	0	0
1	1	1	Ø
0	0	1	0
1	1	0	1

⇐ excitation table

Q	Q_n	A	B	X
0	0	Ø	Ø	
0	1	1	1	
1	0	0	0	
1	1	Ø	Ø	

⇒ both A & 'B' can't be Ø (don't cares)

→ Conversion of one ff to other is done because only some type of ffs are manufactured.

Date.....

(12) Introduction to ff inter Conversion :-

Conversion of given ff to another ff :-

(i) Get the char. table of the target ff.
(ii) Replace the next state using excitation of the given ff. (Replace Q_n by excitation of given ff.)
(iii) Obtain the expressions for the i/p of given ff & realise them.

ex:- Convert J-K ff to T.

→ char. table for T :-

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

⇒ Excitation of J K

T	Q_n	Q_{n+1}	J	K
0	0	0	0	Ø
0	1	1	Ø	0
1	0	1	1	Ø
1	1	0	Ø	1

ex:-

Q_n	Q_{n+1}	→ (J, K)
0	0	(0 0)
		(0 1)

→ Now, realise 'J' & 'K' in terms of T & Q_n.

for J :-

Q_n \ T	0	1
0	Ø	1
1	Ø	Ø

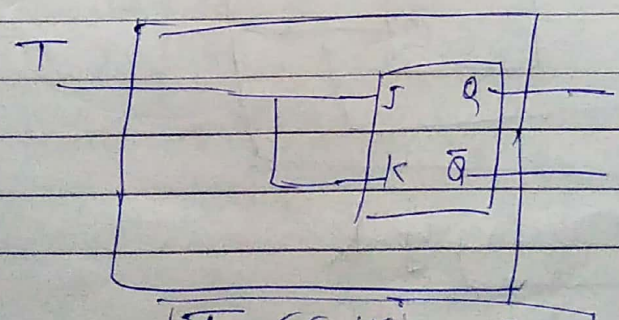
⇒ $J = T$

for K :-

Q_n \ T	0	1
0	Ø	1
1	Ø	Ø

$K = T$

So,



T ff using J-K

Date.....

Ex:- Convert T ff to JK :-

=> char. table for JK :-

T	Q_n	Q_{n+1}
0	0	0
		1
1	0	1
	1	0

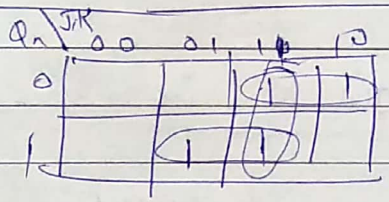
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

=>

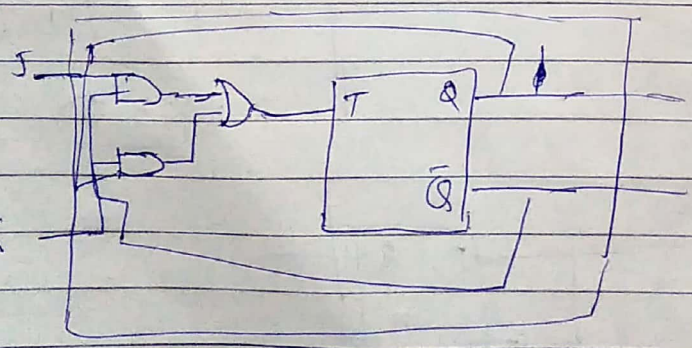
excitation for T

J	K	Q_n	Q_{n+1}	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1

=> NOW, characteristic eqn for 'T' in terms of J, K & Q_n.



$$T = Q(J\bar{Q}_n + KQ_n)$$

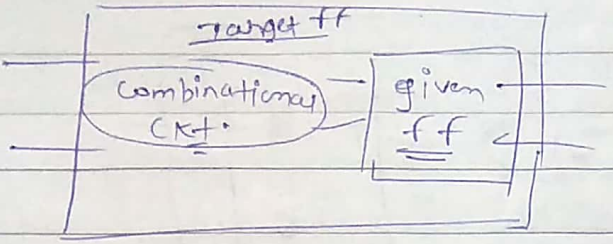


T using J-K

Spiral

Date..08/03/19...

=>



(13) Inter Conversions of ff example 1 :-

(Q) A New ff, $X_1 X_2$ has char. eqn $Q_n = \bar{X}_1 \bar{Q} + \bar{X}_2 Q$. Realise it using T-ff.

=> char. table for T :-

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

=>

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

=> Char. eqn of $X_1 X_2$ (target ff) :-

X_1	X_2	Q_n	Q_{n+1}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Excitation Of T

=>

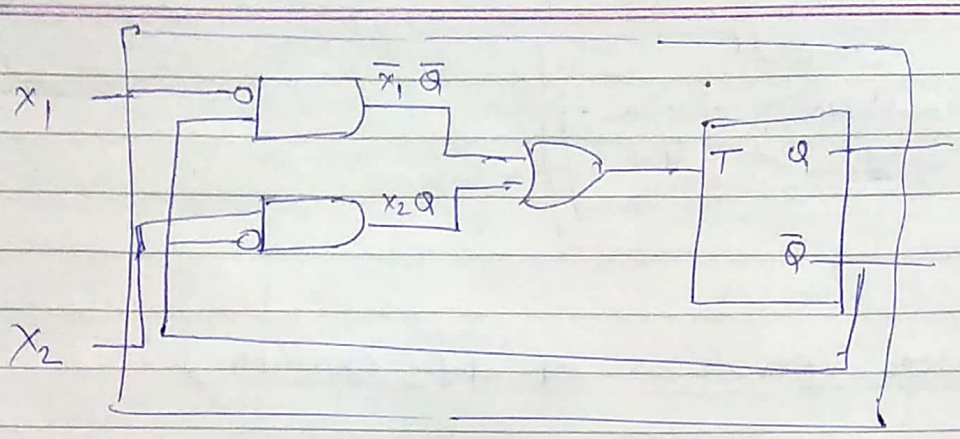
Q	Q_n	T
0	1	1
1	1	0
0	1	1
1	0	1
0	0	0
1	1	0
0	0	0
1	0	1

Q \ X_1X_2	00	01	11	10
0	1	1		
1		1	1	

$T = (\bar{X}_1 \bar{Q} + X_2 Q)$

=

Date..09/03/19.....



X_1X_2 ff using T ff.

(19) Inter Conversions of ff example 2:-

ex:- given ff X_1X_2 & char. eqn $Q_n = \bar{X}_1\bar{Q} + \bar{X}_2 Q$
Now realise T-ff using X_1X_2.

⇒ ~~F~~ char. table of ~~to~~ T:-

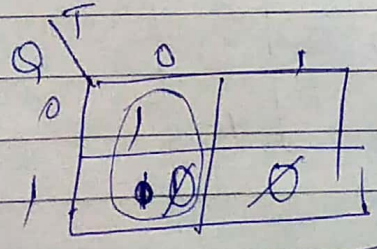
T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

excitation of X_1X_2

Q_n	Q_{n+1}	X_1	X_2
0	0	1	Ø
0	1	Ø	0
1	1	0	Ø
1	0	Ø	1

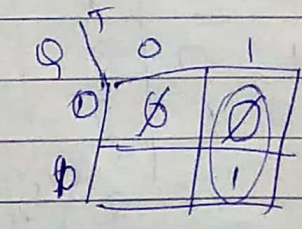
(Use char. eqn for this)

Now, Realising T ff



(for X_1)

$X_1 = \bar{T}$

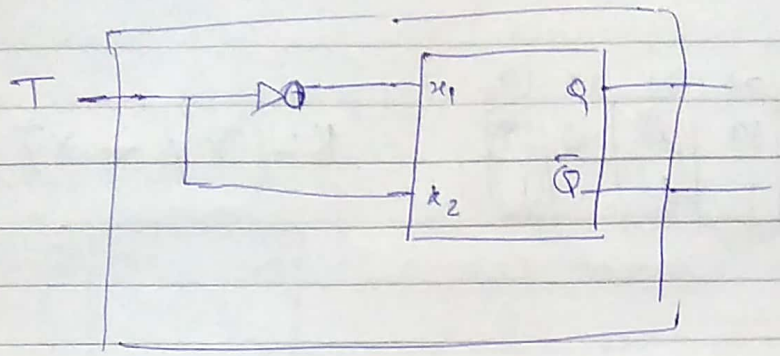


(for X_2)

$X_2 = T$

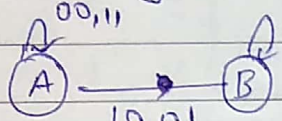
Spiral

Date.....



Inter Conversions of ff example 3:-

(ex:-) An X, Y. ff is represented by the following state diagram



{ NO arrow on the edge connecting A & B. ~~Even~~ }

To realise it using JK-ff. find J & K.

⇒

X	Y	Q_n	Q_{n+1}
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

(using state diag.)

excitation of JK ⇒

Q_n	Q_{n+1}	J	K
0	0	0	Ø
0	1	1	Ø
1	0	Ø	1
1	1	Ø	0

Q_n \ XY	00	01	11	10
0		1		1
1	Ø	Ø	Ø	Ø

for J

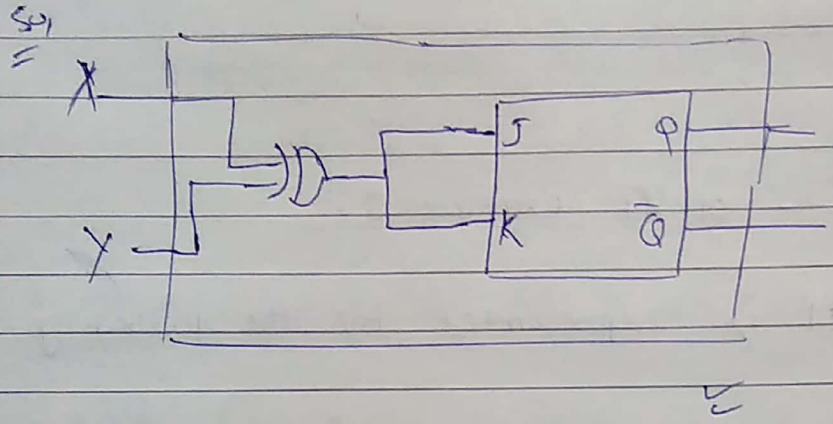
$J = \bar{X}Y + X\bar{Y}$

Date.....

for K

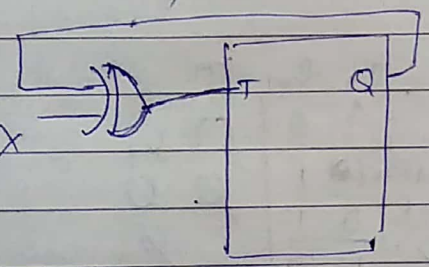
Q \ xy	00	01	11	10
0	Ø	Ø	Ø	Ø
1		1		1

$K = (\bar{x}y + x\bar{y})$



(16) Inter Conversion of flip flops eg:-

(Q) what is behaviour of following one-input ff 'x'



(a) ▷-ff (b) T-ff.
(c) Inverted D-ff (d) Inverted T-ff.

⇒ It is implementation of T-ff using X-ff.

Char. table for X:-

X	Q_n	Q_{n+1}	$T(X \oplus Q_n)$
0	0	0	0
0	1	0	1
1	0	1	1
1	1	0	0

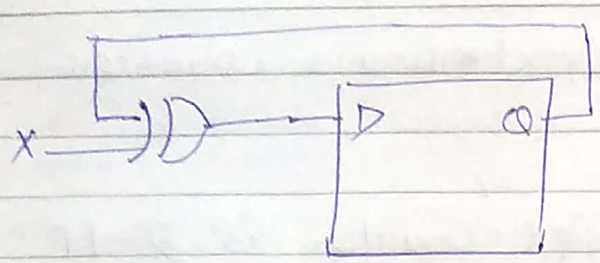
if since ff is depending on T ff so, find T & then using T & Q_n find 'Q_{n+1}'

▷-ff

Date.....

(17) Inter conversion of flip-flops exs:-

(Q) Same as prev. question



(a) D-ff (b) T-ff
(c) inverted D-ff (d) inverted T-ff.

=>

X	Qn	Qn+1	D (X⊕Qn)
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

for X=0 ⇒ latch
& X=1 ⇒ Toggle
||
T-ff

||
T-ff

(18) Introduction to counters:-

→ Counters are used to provide accurate timing & control signals

→ These are of two types
(a) Synchronous (b) Asynchronous.

→ In synchronous counters all the ff responds to the same clock instances.

→ In Asynchronous counters all the ff drives the clock of another ff.

→ Synchronous counters are faster than asynchronous counters.

→ Generally Asynchronous Counters are implemented using T-ff & Synchronous counters are implemented using D-ff.

Date.....

→ Due to simplification in design Asynchronous counters are used in IC ~~design~~ fabrication.

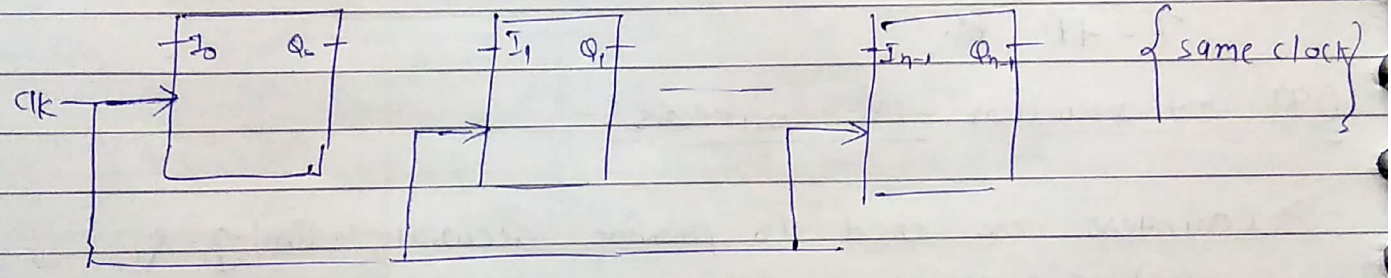
→ The simplified version of synchronous counters is called "shift Counters"

→ The basic element in shift-Counter is D-ff

→ The ring counter & Johnson counter are further simplified versions of shift Counter.

(19) Asynchronous & synchronous counters:-

Synchronous Counter:-



→ The i/ps I_0 to I_{n-1} can be func^n of O/p of ~~previous~~ present state & ~~they~~ there might be combinational ckts which are used to connect all the i/ps. (I_0 to I_{n-1})

→ propagn delay in Synchronous counter is:-

$$T_{pd\,syn} = T_{ff} + T_{comb^n}$$

(T_{ff}: propagn delay of one ff.)

(T_{comb^n}: propagn delay of combinational ckt. used for i/ps)

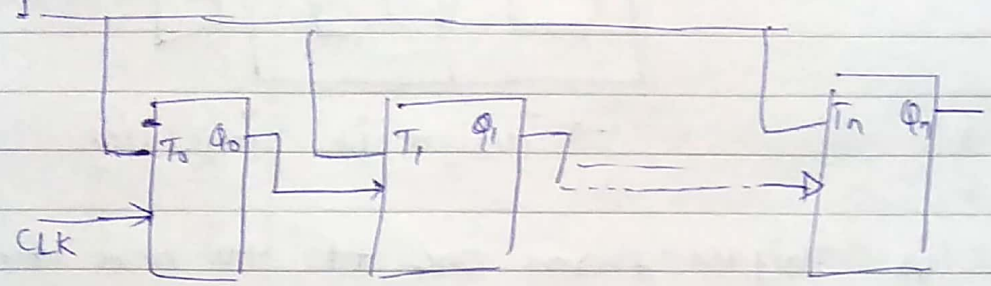
Spiral

Date.....

$T_{clk} \geq T_{pdsyn}$ ⇒ propagn delay should always be less than time period of the clock. ie. Time T

Asynchronous Counter:-

→ o/p of previous ff will be clock to next ff.



propagn delay in Asynchronous Counter (T_{pdasyn}) = $N \times T_{ff} + T_{comb^n}$

$T_{clk} \geq T_{pdasyn}$ ⇒ (ie. After applying the 1st signal atleast wait for "T_{pdasyn}" time before applying the next signal.)

(20) Shift Counters:-

Gn synchronous Counter:-

$I_i = f(Q_0, Q_1, -- Q_{n-1})$ where $i = 0, 1, -- N-1$.

→ It's design are more complex, because each i/ps are func^n of all the o/p's.

Date.....

→ So, Simplifications are done as:-

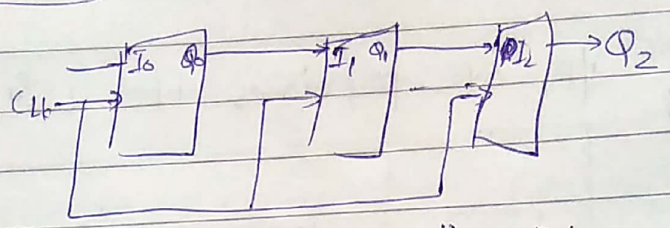
$I_i = Q_{i-1}$ (***) ⇒ each I/p depends only one O/p ie. O/p of previous ff.

↳ where $i = 0$ to $N-1$

ex:- $I_1 = Q_0$

$I_2 = Q_1$

ex:-

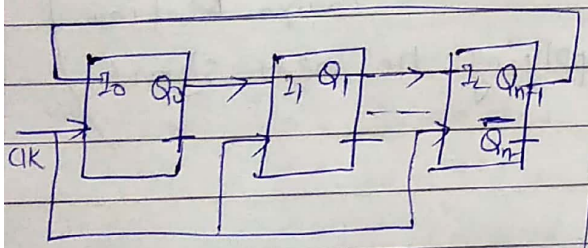


→ design is simplified.

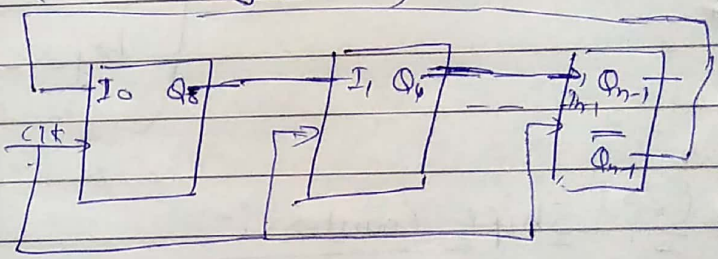
→ Here data is shifted from one ff to next ff. so this is called shift counter.

NOW, $I_0 = f(Q_{N-1})$ {ie. I_0 is function of O/P of last ff}

if $(I_0 = Q_{N-1})$
(Ring counter)



if $I_0 = (\overline{Q_{N-1}})$
(Johnson counter)
or
(Twisted ring counter)



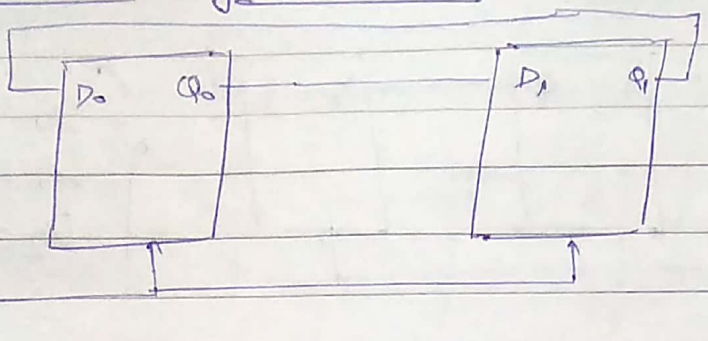
⇒ D-ffs are used for all the shift counters.

→ Ring & Johnson are also types of shift-counters.

Spiral

Date.....

(21) Mod 2 ring counters :-



Present state		Next State	
MSB Q_1	LSB Q_0	Q_{1N}	Q_{0N}
0	0		
0	1		
1	0		
1	1		

⇒ Now to get Q_{1N} & Q_{0N} we need to know D_0 & D_1 but D_0 depends on Q_1 & D_1 depends on Q_0.

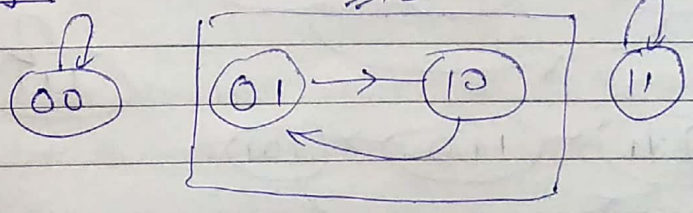
⇓

same as Q_1 ← same as Q_0

Q_1	Q_0	Q_{1N}	Q_{0N}	D_0	D_1
0	0	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	0
1	1	1	1	1	1

State diag :-

⇒ Mod 2 Counter



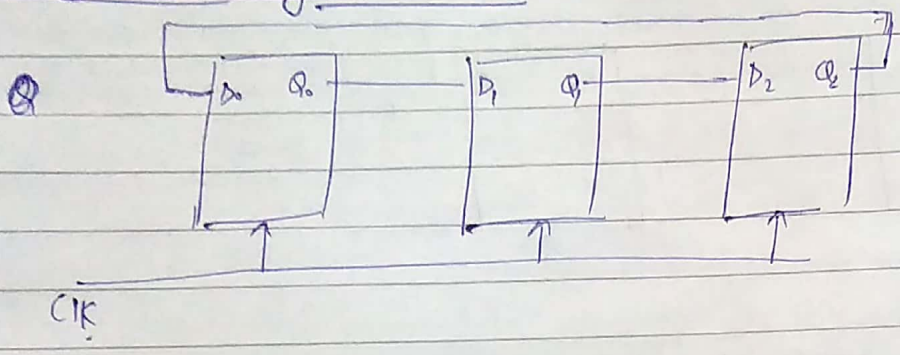
⇒ Even though there are 4-states in the state diag. the ring counter can only ~~count~~ do Mod 2 Counting. This is because of applying simplification to the shift counter ie. by making i/p depends on the o/p of prev. state)

Spiral → Ideally a N-bit counter should do Mod 2^n Counting

A N-bit Ring Counter can only do Mod N Counting

Date.....

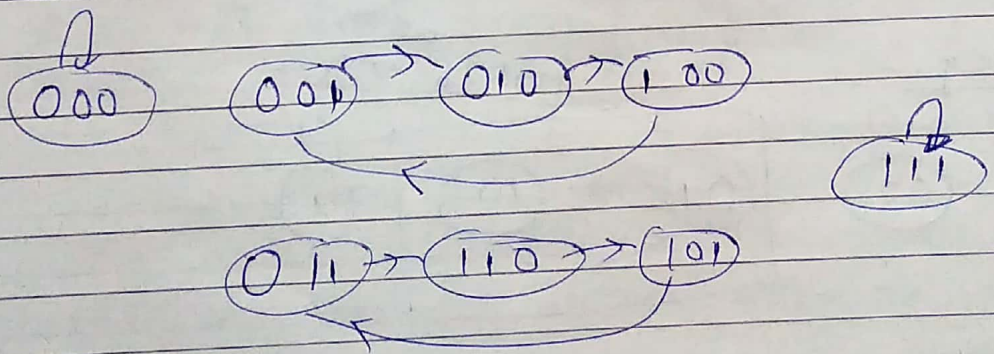
(22) Mod 3 Ring Counter :-



$D_0 = Q_2$
$D_1 = Q_0$
$D_2 = Q_1$

Present State			Next State					
Q_2	Q_1	Q_0	Q_{2N}	Q_{1N}	Q_{0N}	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1	0
0	1	0	1	0	0	1	0	0
0	1	1	1	1	0	1	1	0
1	0	0	0	0	1	0	0	1
1	0	1	0	1	1	0	1	1
1	1	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	1

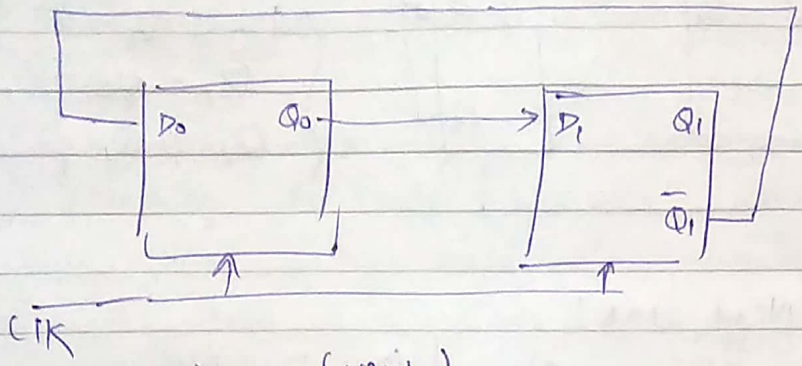
o/p of D-ff is same as D



→ So, Mod 3 Ring Counter can only have 3 (out of 8) Counting states ie. either top '3' or bottom '3' (000 & (111) are not Counting states.

Date.....

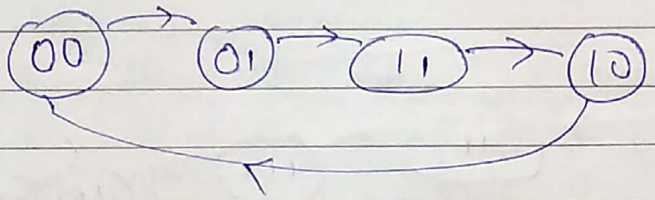
(23) Mod 4 Johnson Counter:-



⇒ $D_0 = \bar{Q}_1$
$D_1 = Q_0$

(Present)		(Next)			
Q_1	Q_0	Q_{1N}	Q_{0N}	D_1	D_0
0	0	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	1	1	0	1	0

State diag:-



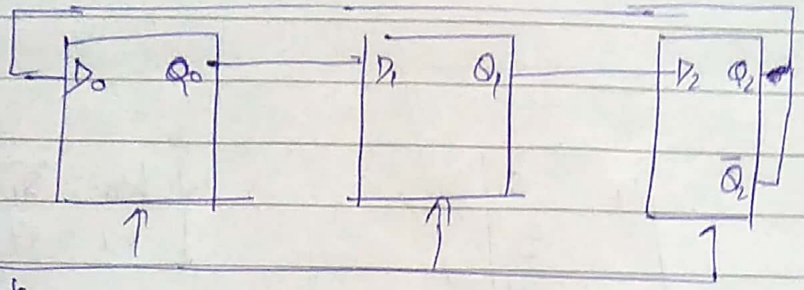
{ All the states are involved in Counting }

for N-bit twisted ring counter or johnson counter, the counting possible is mod(2N).

→ In johnson counter out of 2^N states possible for N-bit " " the no. of counting states are '2N'.

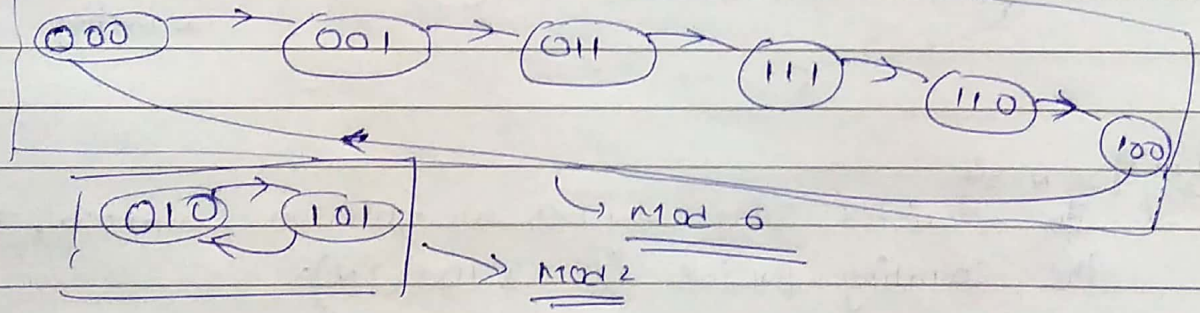
→ for 2-bit johnson counter $2^2 = 2 \times 2$ i.e. all the states are counting states & 2 bit johnonson Counter does mod 4 Counting.

(29) Mod-6 Johnson Counter:- Date.....



$$\begin{cases} D_0 = \bar{Q}_2 \\ D_1 = Q_0 \\ D_2 = Q_1 \end{cases}$$

(Present state)			(Next state)					
Q_2	Q_1	Q_0	Q_{2N}	Q_{1N}	Q_{0N}	D_2	D_1	D_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	1
0	1	0	1	0	1	1	0	1
0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	1	0
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	1	1	0



⇒ Counting Mod 2^N is NOT possible with N-bith shift counter because of limitations on i/p.

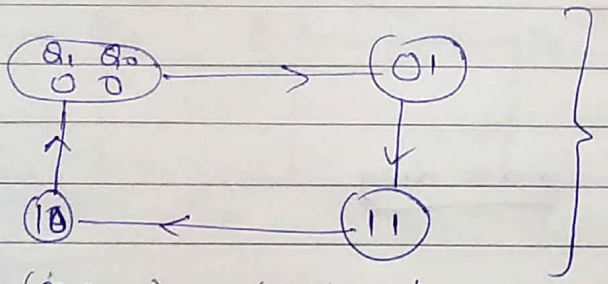
(25) Mod 4 Gray counter using T-ff :- Date.....

Designing a counter using specifications given:-

Steps:-

1.) Get the state table from state diag.
2.) Identify ff to be used & replace the concerned next state using excitation table of associated ff.
3.) Get the expressions for i/ps & realise them.

Ex:- Design a synchronous counter for the following using T-ff.



*** since there are 4 states, 2-ffs are needed.

(present) (Next state)

Q_1	Q_0	Q_{1N}	Q_{0N}	i/ps (excitation T_1	T_0)
0	0	0	1	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	1	1	0	0	1

→ Use (Q_1 & Q_{1N} to get T_1) & (Q_0 & Q_{1N} to get T_0)

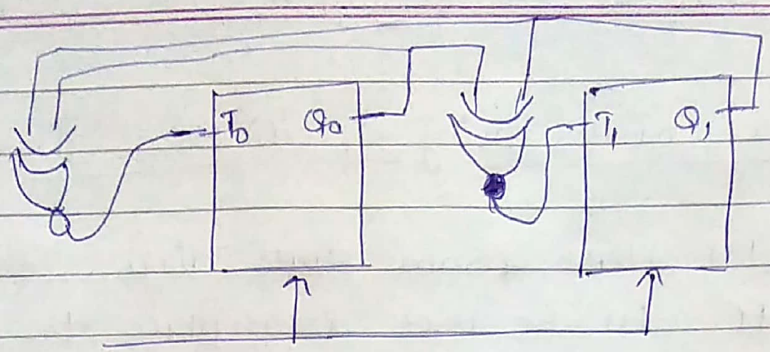
Realise

Now, T_1 & T_2 in terms of Q_1 & Q_0.

$$T_1 = \bar{Q}_1 Q_0 + Q_1 \bar{Q}_0 \quad , \quad T_2 = \bar{Q}_1 \bar{Q}_0 + Q_1 Q_0$$

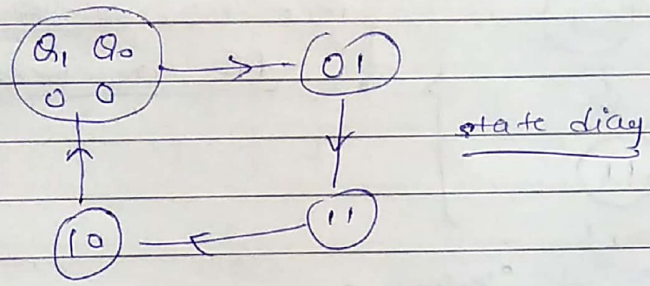
$$T_1 = Q_1 \oplus Q_0 \qquad T_2 = \overline{Q_1 \oplus Q_0}$$

Date..10-03-19



→ In the counter given in each state only 1-bit is changing so, it is called gray code counter. Or Mod 4 gray counter.

(26) Mod 4 Gray Counter using d-ff :-



state diag

→ State table :-

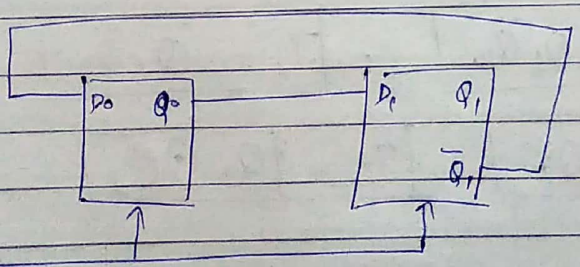
Present		Next			
Q_1	Q_0	Q_{1N}	Q_{0N}	D_1	D_0
0	0	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	1	1	0	1	0

$D_1 = \bar{Q}_1 Q_0 + Q_1 \bar{Q}_0$

$\Rightarrow Q_0$

$D_0 = \bar{Q}_1 \bar{Q}_0 + \bar{Q}_1 \bar{Q}_0$

$\Rightarrow \bar{Q}_1$



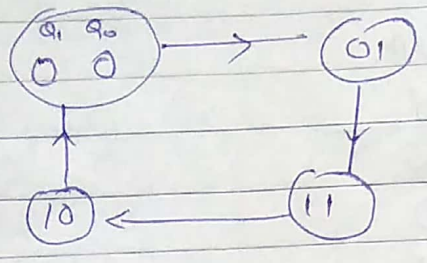
→ since this implementation of Mod 4 Gray Counter using D-ff req. ~~less~~ NO comb^n ckt compared to T-ff implementation, so this is popularly used

Spiral

→ generally Synchronous

Date.....

(27) Mod 4 Gray Counter using D & T ff:-



⇒

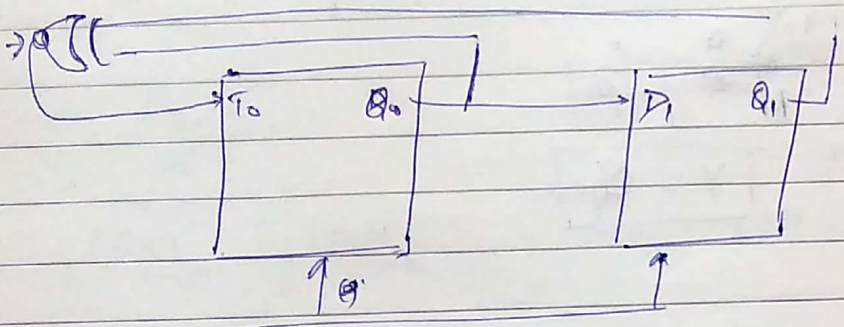
Q_1	Q_0	Q_{1N}	Q_{0N}	D_1	T_0
0	0	0	1	0	1
0	1	1	1	1	0
1	0	0	0	0	0
1	1	1	0	1	1

⇒ $D_1 = \bar{Q}_1 Q_0 + Q_1 Q_0$

⇒ Q_0

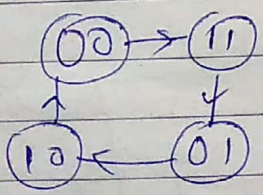
$T_0 = \bar{Q}_1 \bar{Q}_0 + Q_1 Q_0$

⇒ $Q_0 \odot Q_1$



(28) Counter using two diff. ffs:-

(Q) Consider the following state diagram which is to be designed using T-ff for MSB & xy for LSB, the behaviour of xy is given below.



x	y	Q_n
0	0	0
0	1	Q
1	0	$\bar{Q}$
1	1	1

Date.....

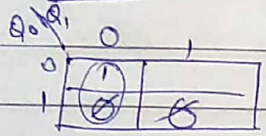
=> state table :-

Q_1	Q_0	Q_{1N}	Q_{0N}	T_1	x (MSB)	y (LSB)
0	0	1	1	1	1	Ø
0	1	1	0	1	Ø	0
1	0	0	0	1	0	Ø
1	1	0	1	1	Ø	1

=> from given function table of x y

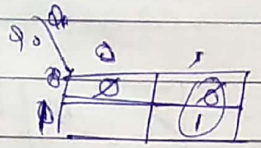
$T = 1$

for x :-

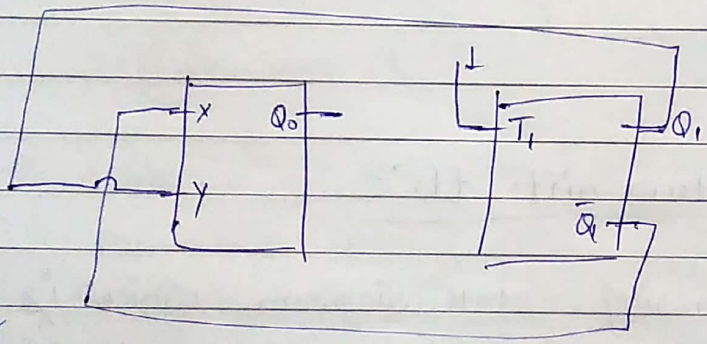


$X = \bar{Q}_1$

for y :-

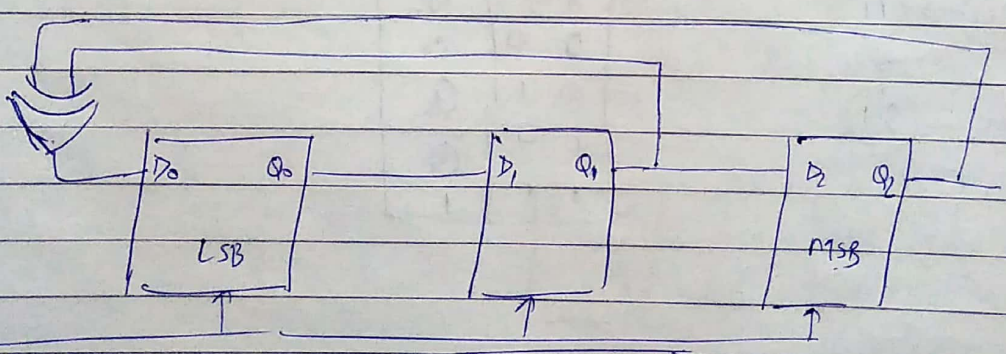


$Y = Q_1$



**** (29) Model on analysis of counting states & sequence generations :-

ex :-



CLK

$Q_{0N} = D_0 = Q_1 \oplus Q_2$
$Q_{1N} = D_1 = Q_0$
$Q_{2N} = D_2 = Q_1$

***

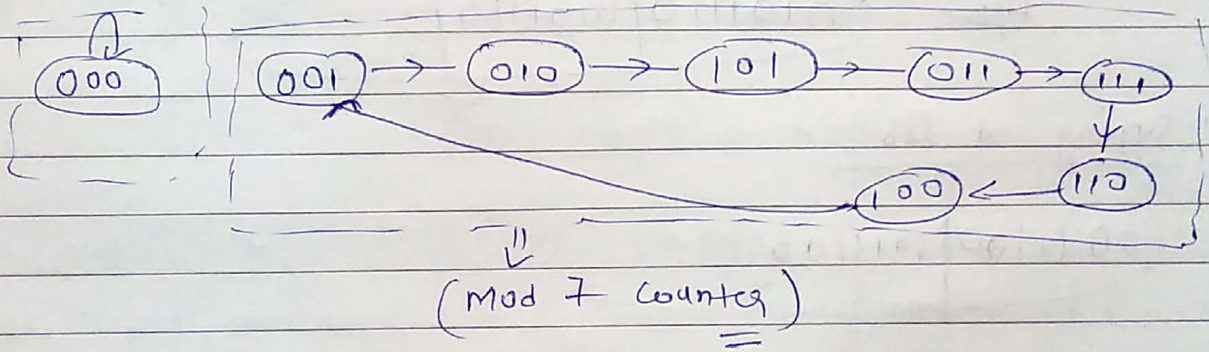
Spiral

Date.....

⇒ State table :-

Q_2	Q_1	Q_0	Q_{2N}	Q_{1N}	Q_{0N}
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	0

State diag :-



(Mod 7 counter)

(Q1) How many states are there in modular counting?

⇒ 7 {mod 7}

(Q2) given that 001 is initial state what is the state after 4-clock cycles & 10 clock cycles

⇒ 111 → 011

4-clocks 10-clock cycles {7 + 3} → 3 clocks

ex:- after 25 clocks :-

(21+4) ⇒ 4 cycles = 111

Date.....

(Q9) If initial state is 001 & o/p is tapped at Q_2 then what is the o/p?

⇒ Since tapping is done at Q_2 all the MSBs will be selected then

o/p = 0 0 1 0 1 1 1 ... → { MSB of each state }

this will keep on repeating (we'll get a seq.)

⇒ Tapping the o/p at Q_1 :-

o/p = 0 1 0 1 1 1 0 | 0 1 0 1 1 1 0 ...

⇒ Tapping at Q_0 :-

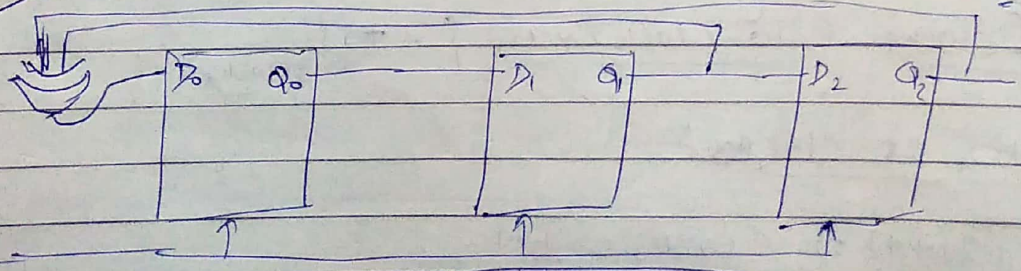
1 0 1 1 1 0 0 | 1 0 1 1 1 0 0 ...

→ Since seq. are generated thus counters can also be act as sequence generators.

(30) Deriving the clock frequency :-

(Q) Which of the following clock freq. ensures proper counting.

(a) 90 MHz (b) 60 MHz (c) 90 MHz (d) 300 MHz.



T_{ff} = 15ns, T_{comb} = 5ns → given

→ Counter can't perform as Counting the as Modular Counting.

Date.....

⇒ T_{ff} = propagation delay of ff.
T_{comb^n} = " " " Combinational ckt.

⇒ For proper Counting :-

$T_{clock} \geq T_{Pd\,syn}$

⇒ (propagn delay of syn. Counter)

$T_{Pd\,syn} = T_{ff} + T_{comb^n}$

⇒ 15 ns + 5 ns

⇒ 20 ns

$T_{clock} \geq 20$ ns ⇒ freq. of clock = $\frac{1}{20\,ns}$

⇒ $\frac{1000}{20} \times 10^6$ Hz

⇒ 50 MHz

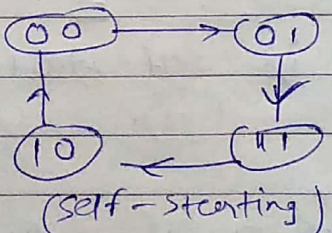
***
{ Check the option which is less than 50 MHz }

(31) Self-starting & free running :-

On the basis of state diag. Counters are of two types :-

(i) Self-starting Counter :- A Counter is said to be self-starting if it is possible to enter counting loops irrespective of the initial state.

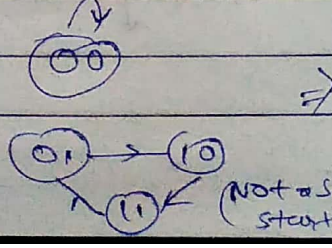
ex :-



(self-starting)

⇒ { No matter which is the initial state, it will always be in counting loops }

ex :-



(Not a self-starting)

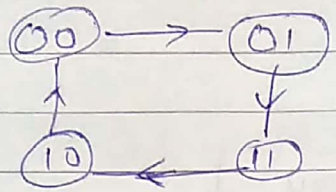
⇒ { If initial state is 00 then it will never enter the Counting loop }

Date.....

(ii) Free running counter :-

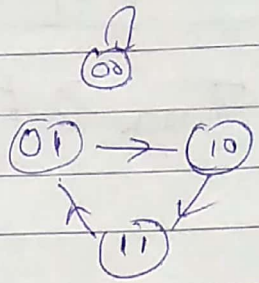
=> A counter is said to be free running if it contains all possible states in the counting loop.

ex:- (a)



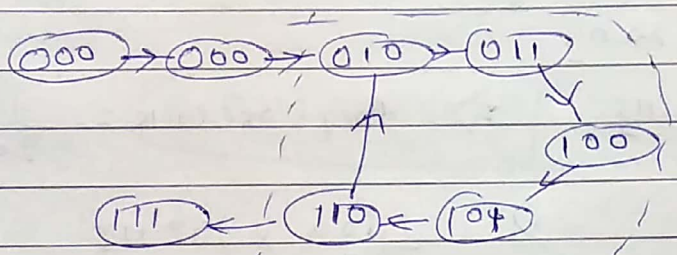
(free running as well as self starting)

ex:- (b)



Neither free running nor self-starting

ex. c:-



(Counting loop)

-> Not free running because all the states are not in counting loop.

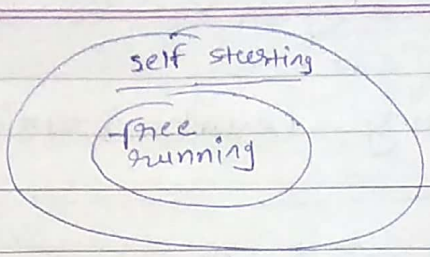
-> Self-starting, because after few clock cycles the counter will enter the counting loop.
{In worst case counter will enter counting loop after two clock cycles}

Every free-running counter is self-starting counter but inverse is not true.

If a counter is not self-starting then it also not free-running.

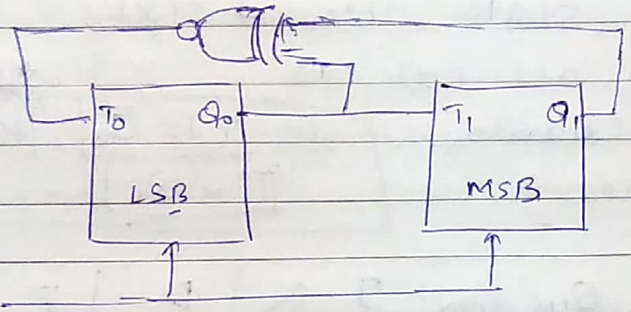
Date.....

→



(32) Example on self-starting & free-running :-

(Q) Identify the following counter is self-starting or free running.



~~$Q_{0N} = T_1$~~

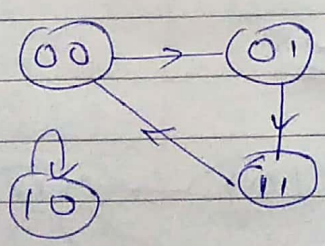
~~$Q_{1N} = 1$~~

$T_1 = Q_0$

$T_0 = Q_1 \odot Q_0$

State table :-

Present		Next			
Q_1	Q_0	Q_{1N}	Q_{0N}	T_1	T_0
0	0	0	1	0	1
0	1	1	1	1	0
1	0	1	0	0	0
1	1	0	0	1	1

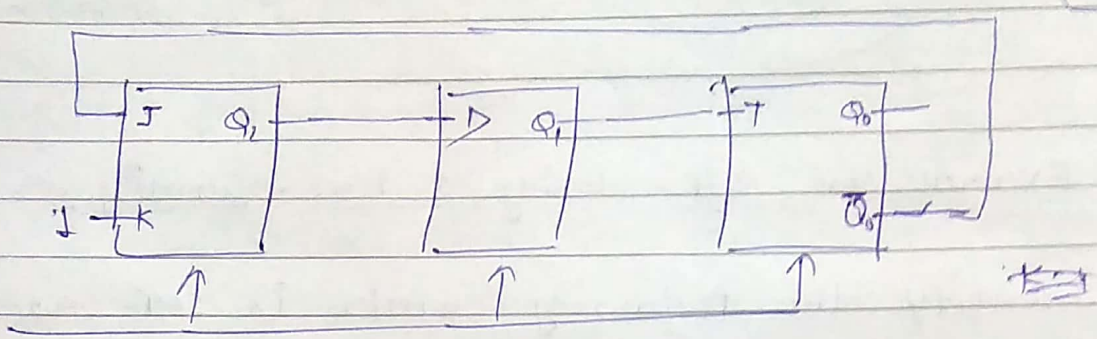


⇒ It is neither self-starting nor free-running.

Date.....

(33) Counter using 3-diff ffs :-

(Q) Consider the following counter, initially $Q_2Q_1Q_0 = 000$



(i) What will be the state after 4 clocks

(ii) (a) 000 (b) 010 (c) 011 (d) 101

(ii) Modulus of the counter,

(a) 4 (b) 5 (c) 6 (d) 7

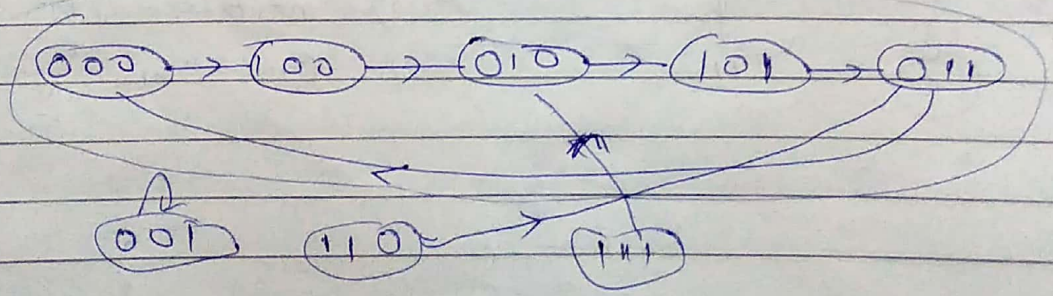
$K=1,\ J=\bar{Q}_0,\ D=Q_2,\ T=Q_1$

⇒

Q_2	Q_1	Q_0	Q_{2N}	Q_{1N}	Q_{0N}	J	K	D	T
0	0	0	1	0	0	1	1	0	0
0	0	1	0	0	1	0	1	0	0
0	1	0	1	0	1	1	1	0	1
0	1	1	0	0	0	0	1	0	1
1	0	0	0	1	0	1	1	1	0
1	0	1	0	1	1	0	1	1	0
1	1	0	0	1	1	1	1	1	1
1	1	1	0	1	0	0	1	1	1

$K=1$, $J=$

State diagr ⇒ Mod 5

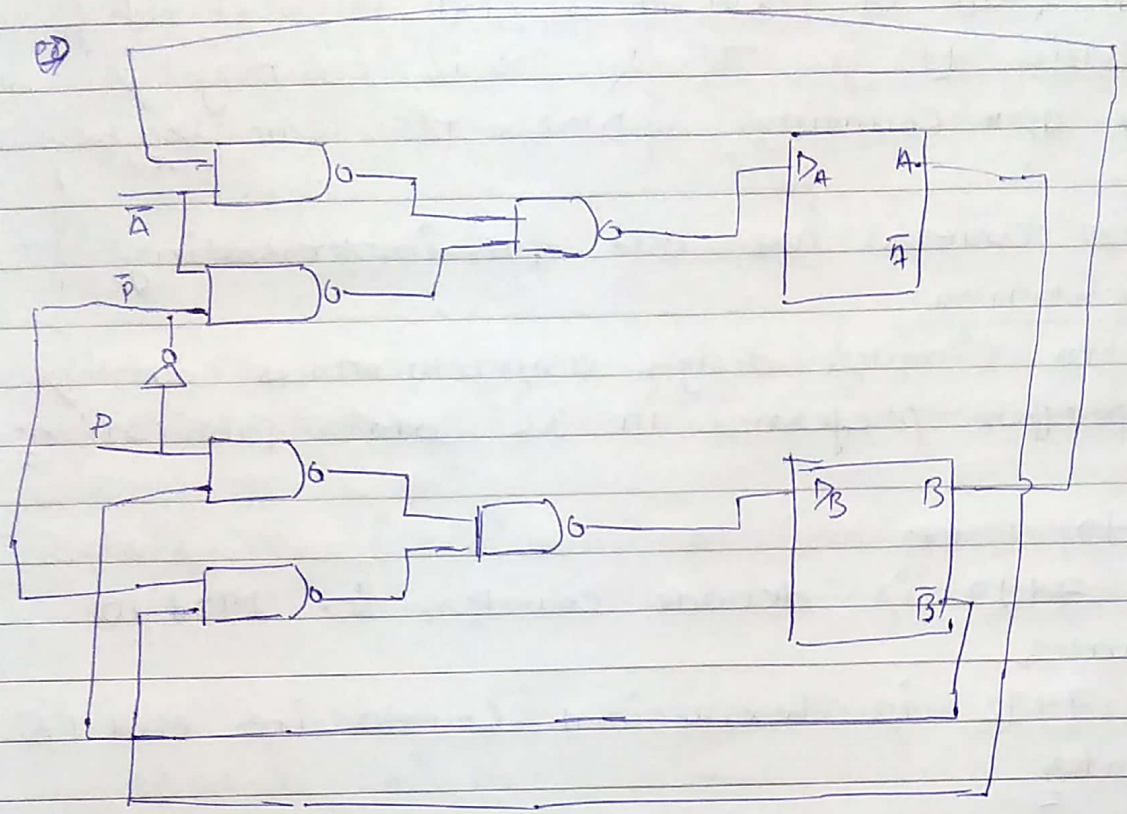


Spiral Not free-running & Not self-starting

Date.....

(39) Example on combinational ckt. & ff :-

(Q) Consider the following Counter, if P=0, the Counting seq. of AB is.



⇒ A is MSB & B is LSB

$$D_A = (\bar{P}\bar{A} + \bar{A}B)$$ & $$D_B = P\bar{B} + \bar{P}A$$

$\Rightarrow D_B = A$ {∵ P=0}

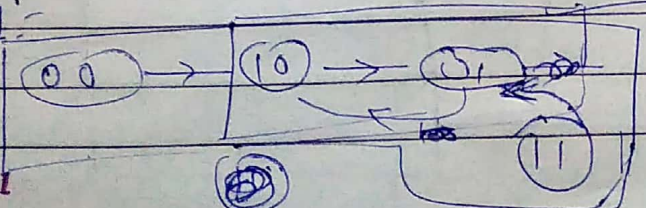
$D_A = \bar{A}$ {∵ P=0}

(NAND - NAND or AND - OR)

A	B	A_n	B_n	D_A	D_B
0	0	1	0	1	0
0	1	1	0	1	0
1	0	0	1	0	1
1	1	0	1	0	1

→ Mod 2 Counter

Statediagram:-



⇒ {It is self-starting but not free running}

Date.....

(35) Introduction to Asynchronous counter:-

→ Asynchronous counter are also called as ripple counter. {i.e. O/p of prev. ff is clock to next ff.}

→ The basic ff is T & basic counting is binary up counting. {other ffs. can also be used}

→ The up counters are used for implementing incrementation.

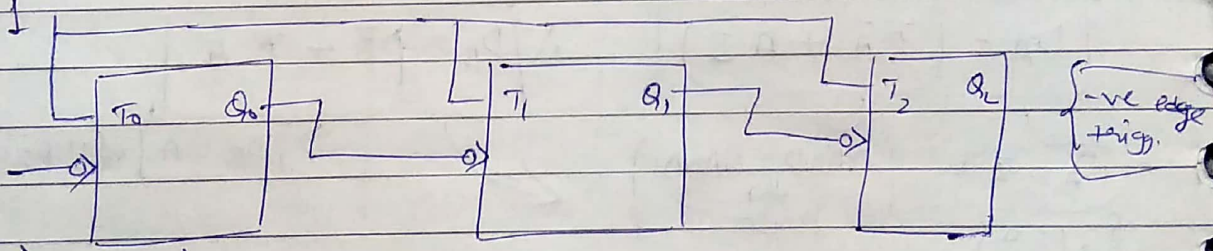
→ Due to simpler design asynchronous counters are preferred in IC counter fabrication.

→ ~~O/p to prev.~~

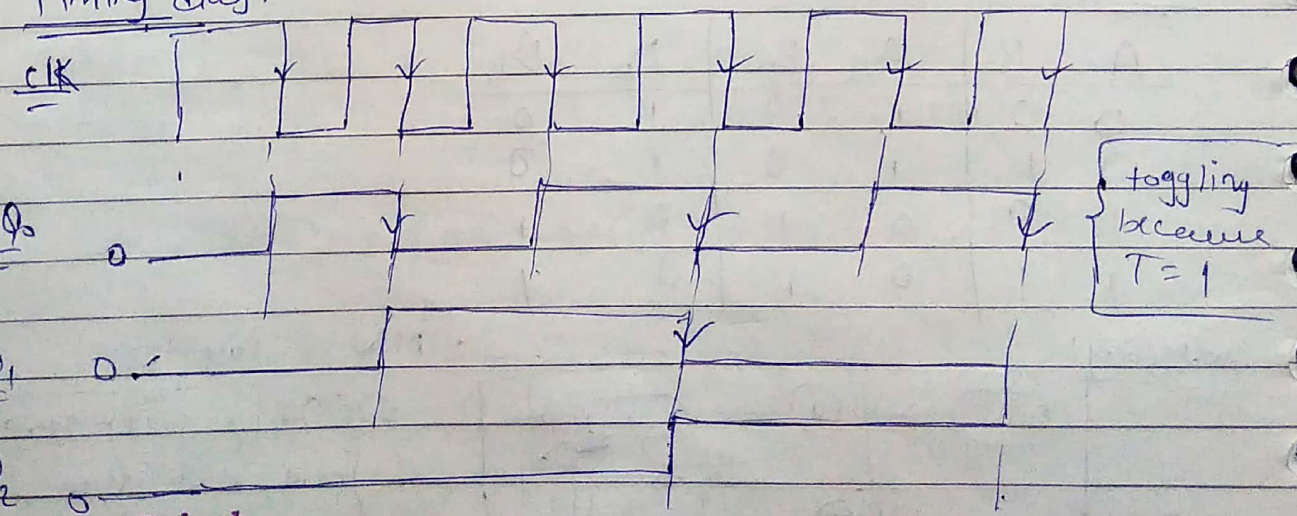
→ IC 7490 is decade counter i.e. Mod 10 counter.

→ IC 7492 is hexadecimal counter i.e. Mod 16 counter.

(36) Mod 8 up counter:-



Timing diag:-



Spiral

Date.....

Time period $(Q_0) = 2 * T_{CLK}$

Time period $Q_0 = 2 * T_{CLK}$

freq. $Q_0 = \frac{1}{2} * f_{CLK}$

Time. period of $Q_1 = 4 * T_{CLK}$ & freq. of $Q_1 = \frac{1}{4} * f_{clock}$

Time period of $Q_2 = 8 * T_{CLK}$ & freq. of $Q_2 = \frac{1}{8} * f_{CLK}$

⇒ Each ff. indivisually acts as a Mod 2 counter ie. each counter counting 2 clock cycles a '1'

→ AND combining 3 Mod-2 counters we'll get Mod 8 counter.

→ Each T-ff is going to toggle depending one o/p of prev. ff.

ie:

$Q_{0N} = \bar{Q}_0$, for every clock

$Q_{1N} = \bar{Q}_1$, when $Q_0 : 1 \to 0$ (high to low -ve edge trigg)

$Q_{2N} = \bar{Q}_2$, when $Q_1 : 1 \to 0$

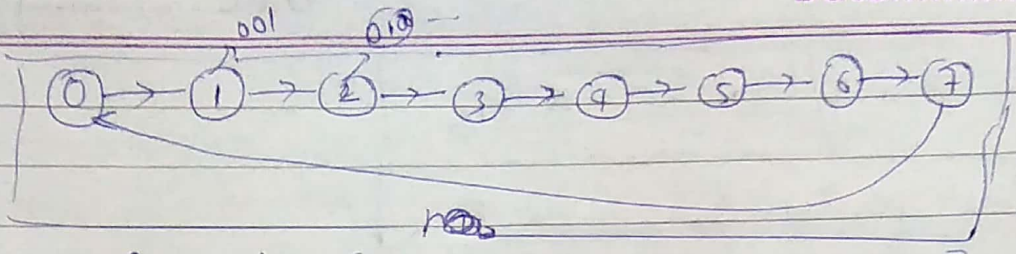
⇒ (Q_{0N} toggle at each clock)

Q_2	Q_1	Q_0	Q_{1N}	Q_{1N}	Q_{0N}
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

(Q_{1N} is toggling when Q_{0N} goes from 1 to 0 else later mode)

→ Q_{2N} is toggling when Q_{1N} goes from 1 to 0

Date.....

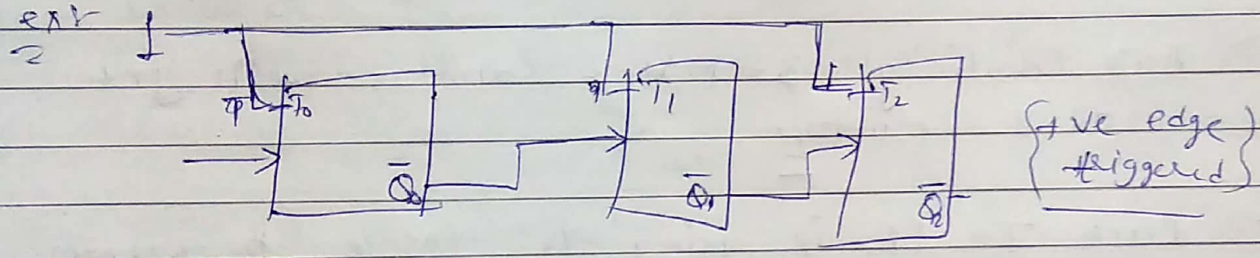


→ {It is free-running as well as self-starting.}

=

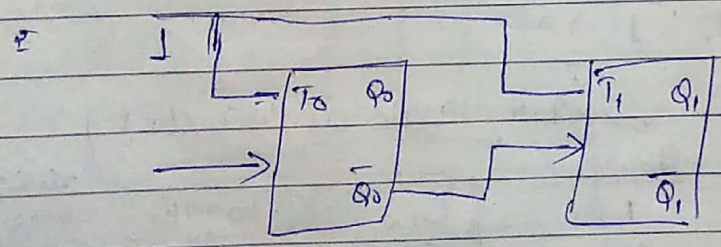
→ Mod 8 counting. {or Mod 8 up counting}

→ In the Mod 8 up Counter shown, if clock is changed to +ve triggered & $\bar{Q}$ of clock is made the clock it will be same as "Mod 8 up counter"



(37) Mod 4 up counter:-

[Q] What kind of counting does the following counter is performing? Is it self-starting or free running.?



⇒ $(T_0 \& T_1 = 1)$, ~~$Q_{0N} = \bar{Q}_0$ for c~~

=

$Q_{0N} = \bar{Q}_0$, for every clk

$Q_{1N} = \bar{Q}_1$, when $Q_0 : (0 \to 1)$ or ~~$\bar{Q}_0$~~ $\equiv Q_0 : (1 \to 0)$

Spiral

Date.....

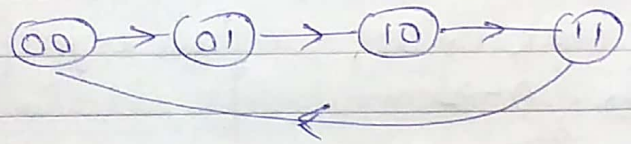
state table :-

Q_1	Q_0	Q_{1N}	Q_{0N}
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Q_{0N} is $\bar{Q}_0$ for every clock

Q_{1N} is toggling when Q_0 changes from 1 to 0

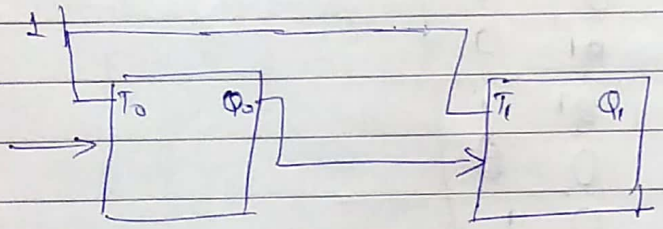
state diag :-



ckt is free running & self-starting

(Mod 4 up Counter)

(38) Mod 4 down counter :-



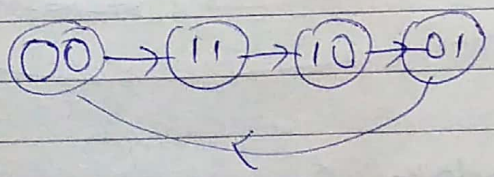
$\Rightarrow Q_{0N} = \bar{Q}_0$, for every clk

$Q_{1N} = \bar{Q}_1$; when Q_0 : 0 → 1

because it is +ve edge trigg

=>

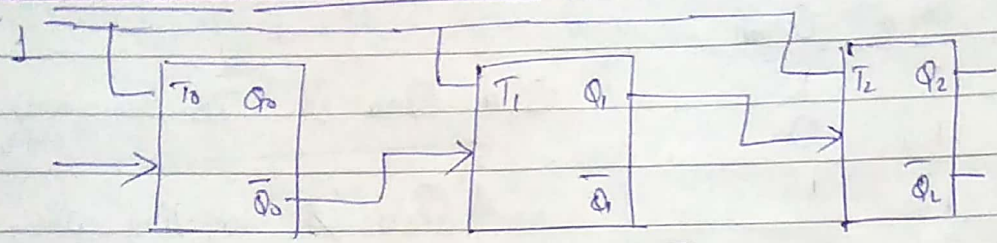
Q_1	Q_0	Q_{1N}	Q_{0N}
0	0	1	1
0	1	0	0
1	0	0	1
1	1	1	0



Mod 4 down Counter

Date.....

(39) Mod 8 random counter:-

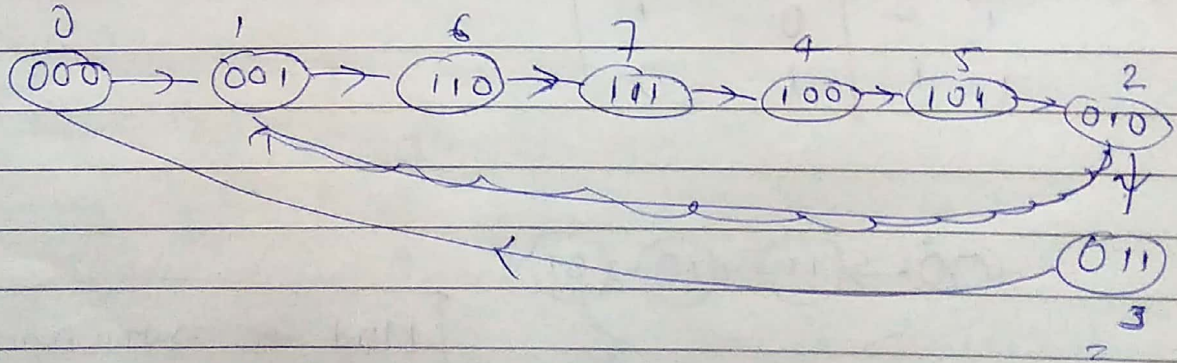


→ If the initial state $Q_2 Q_1 Q_0 = 101$ what will be the state after 4 clock cycles?

⇒ $Q_{0N} = \bar{Q}_0$; for all clock cycles { +ve edge }
$Q_{1N} = \bar{Q}_1$; when $\bar{Q}_0 : 0 \to 1$
or
$Q_0 : 1 \to 0$
$Q_{2N} = \bar{Q}_2$; when $\bar{Q}_1 : 0 \to 1$

→

Q_2	Q_1	Q_0	Q_{2N}	Q_{1N}	Q_{0N}
0	0	0	0	0	1
0	0	1	1	0	0
0	1	0	0	0	1
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	1	0	0



Mod 8 random Counter

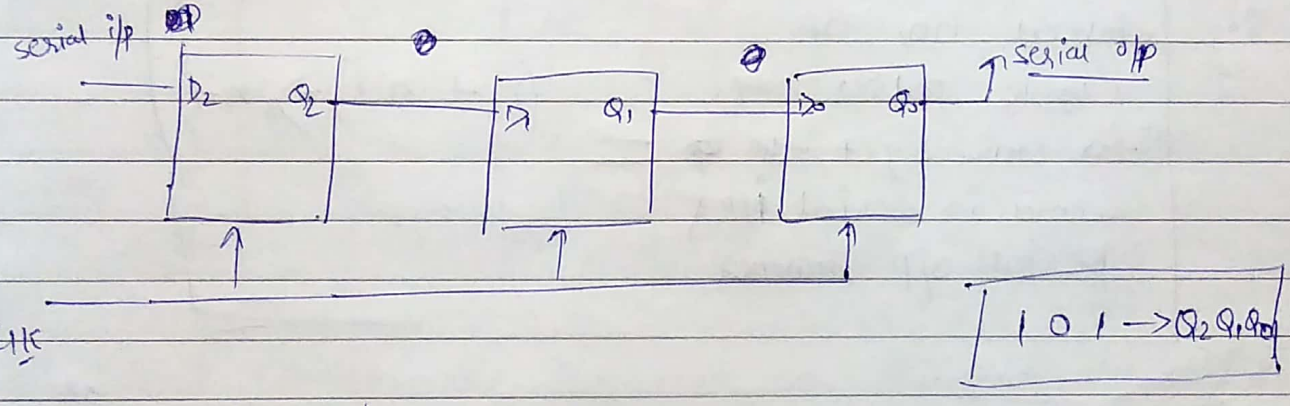
(40) Applications of ff :- Date.....

1) Shift registers :- used as "sequential memory"
ex :- Accumulator in µp is made up of ffs.

2) Counters :- (i) used to count no. of pulses
(ii) used as frequency divider (frequency of clocks gets divided in asynchronous counters)

(41) 3-bit shift registers :-

→ 3-bit shift reg. requires 3 D-ff.



CLK	S/IP	Q_2	Q_1	Q_0
0	—	0	0	0
1	1 →	1	0	0
2	0 →	0	1	0
3	1 →	1	0	1

} → initial condⁿ when i/p is applied

→ This will be serial o/p on shift reg.

→ No. of clocks req. to put 3-bit data on serial i/p shift reg. is '3'

→ i/p are given serially ~~xxx~~ serial i/p on shift reg.
for N-bit data, to put on shift reg. no. of clocks req. = O(N)

Spiral

Date.....

→ TO- get the o/p in serial manner:-

CLK	Q_2	Q_1	Q_0	o/p
0	1	0	1	1
1	0	1	0	0
2	0	0	1	1

} → initially LSB is already present at o/p

⇒ {we got the o/p in 2 clock cycle.}

→ So, NO. of clock cycles req. to get o/p serially from a shift reg = O(n-1)

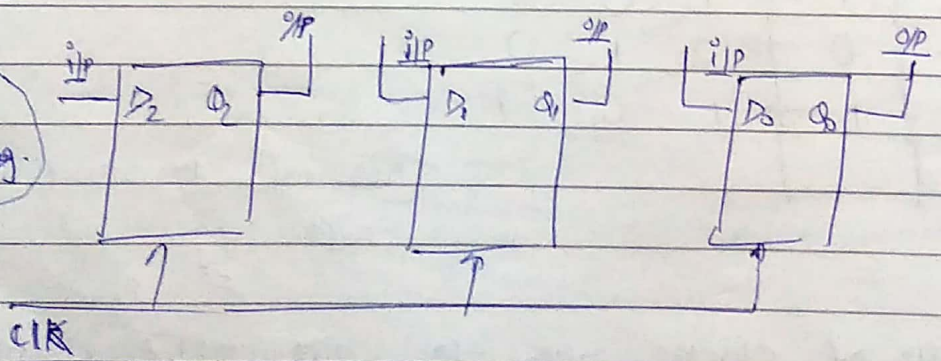
for n-bit shift reg.

***

So, total no. of clock pulses req. to store & get the o/p word in serial i/p & serial o/p manner = (n + n-1) ⇒ 2n-1

For parallel i/p & parallel o/p the no. of clock pulses req. to store & get the data = 1

but it does not form a shift reg.



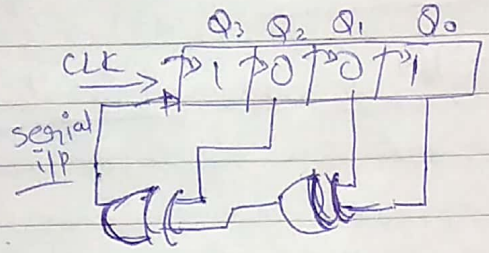
(give the i/p at each i/p port & get the o/p in 1 clock cycle)

Spiral

Date.....

(42) Example 1 on the shift right register :-

(Q) In the following right shift register, determine the no. of clocks req. to bring it to the final initial state of '1001'.



{ Serial i/p will be O/p of exor ie. $Q_2 \oplus Q_1 \oplus Q_0$ }

⇒

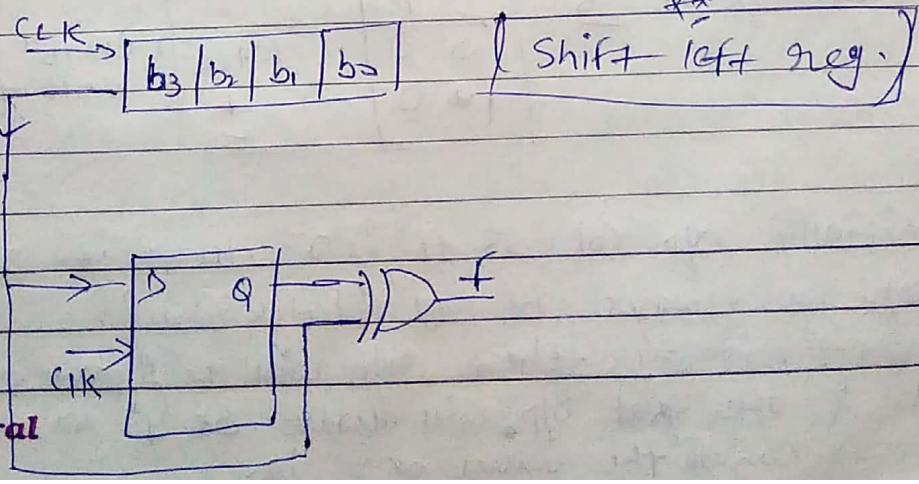
CLK	serial i/p	Q_3 Q_2 Q_1 Q_0
0	—	1 0 0 1
1	1 (O/p of EX-OR)	1 1 0 0
2	1	1 1 1 0
3	0	0 1 1 1
4	1	1 0 1 1
5	0	0 1 0 1
6	0	0 0 1 0
7	1	1 0 0 1

⇒ 7 clocks

→ So, 7 clocks are required to bring it to initial state 1001

(a) Binary to gray convertor :-

(Q) Determine the function of following ckt.



assuming initially o/p of ff = 0

Date.....

→ The 1st o/p will be = $0 \oplus b_3 = b_3$ ≈ 1st clock

→ " 2nd " " " = $b_3 \oplus b_2$ 2nd "

→ " 3rd " " " = $b_2 \oplus b_1$ 3rd "

→ " " " " " = $b_1 \oplus b_0$ 4th "

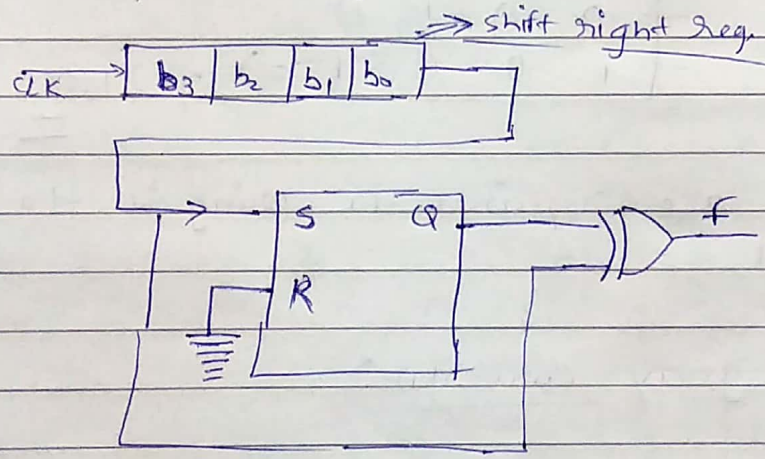
→ All the other o/ps are 0.

→ So, if $|b_3|b_2|b_1|b_0|$ is binary no. then $|b_3|b_3 \oplus b_2|b_2 \oplus b_1|b_1 \oplus b_0|$ represents equiv. gray code.

So, the ckt is binary to gray converter.

(Q5) finding 2's complement:-

ex:- what is the func^n of following ckt.?



⇒ R=0 (always) ∴ So

S	R	
0	0	→ latch
1	0	→ set

⇒ Comb^n of S-R ff.

⇒ If initially o/p of SR ff is '0' then, if S=0 the o/p will always be '0' {latch mode}

→ Once '1' is S=1 then o/p will be '1' (set state) & after that o/p will always be '1' no matter what the value of 'S' is.

Date.....

~~After first~~

After 0th clock => $b_0 \oplus 0 = b_0$.

After 1st " => ~~$b_0 \oplus 0 = b_0$~~

ex:- let $b_3\ b_2\ b_1\ b_0 = 0100$

then o/p will be => 1100 => 2's complement

-> Once the o/p of FF is '1' it will remain '1' & bits that leads to complementation of every other ~~FF~~

ie. $1 \oplus X = \bar{X}$

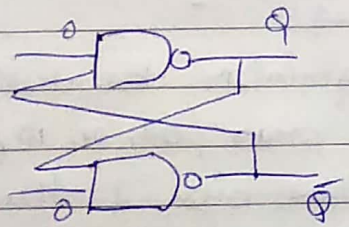
(47) Gate 2004 on S-R latch:-

(Q) SR latch made by cross coupling two NAND gates. If S=R=0 then it will result in.

(a) $Q=0, \bar{Q}=1$ (b) $Q=1, \bar{Q}=0$

(c) $Q=1, \bar{Q}=1$ (d) Indeterminate state.

=>



$Q=1, \bar{Q}=1$

(48) Gate 2019 on Counter:-

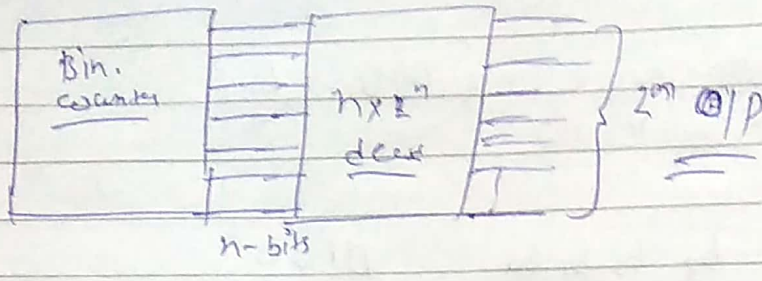
(Q) Let $K=2^n$, a ckt. is built by giving the o/p of an n bit binary counter as i/p to a n to 2^n bit decoder. This ckt. is equiv. to.

(a) k-bit binary up counter (c) k-bit ring counter

(b) " " " down " (d) " Johnson "

Date.....

2)



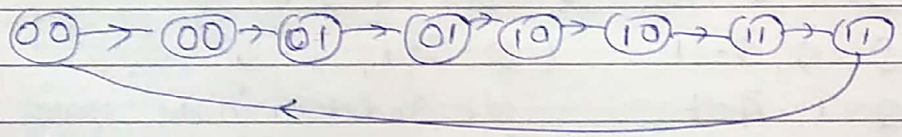
→ depending on O/p of binary counter only one bit will be selected as 1 in o/p of decoder & works same as 2^n bit ring counter.

(99) Gate 2015 on sequence generation:-

(Q) The Min^m no. of JK-ff req. to construct a synchronous counter with the count seq. (0, 0, 1, 1, 2, 2, 3, 3, 0, 0 ---) is _____

(A) 0 (B) 1 (C) 2 (d) 3.

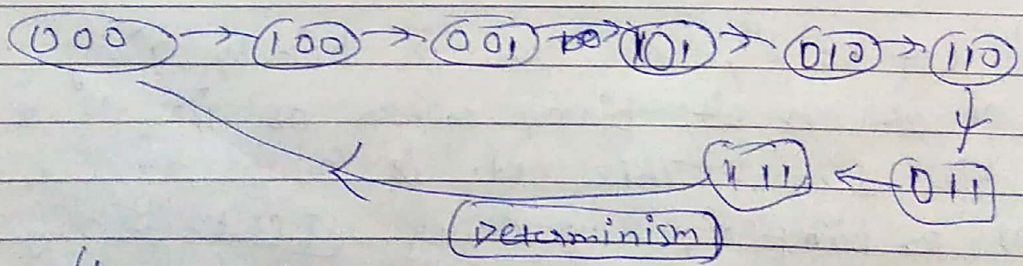
⇒



{Non-determinism}

~~for multiple 3.~~

→ Because of Non-determinism ~~the~~ two-bits are not enough to represent states (00, 01, 10 & 11) so, we need more than two bits & thus more than two ffs., so ~~a~~ three ff are req. & thus 8-diff. states are possible



(by adding 0 & 1 resp. to MSB of each Consecutive states)

Spiral

Date.....

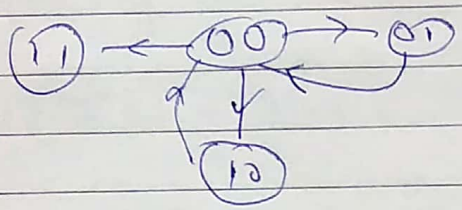
=> So, the seq. given is obtained by tapping only at two O/Ps out of 3 n O/ps of the counter.

(51) Gate 2016 question on seq. generation:-

(Q) We want to design a synch. counter that counts the seq. 0-1-0-2-0-3 & then repeats. The min^m no. of J-K ff. req. to implement this counter is ____

(a) 1 (b) 2 (c) 4 (d) 5

=> Since, states are 0 to 3 lets try with 2 ff.



{ always take the state with max^m no. outgoing edges.

Here Non-determinism in state '00'

=> Instead of 1 outgoing edge there are 3 outgoing edge, & so for those 2 extra outgoing edges we need extra 2 ffs.

So, 4 ffs are req.

no. of extra ff = $\lceil \log_2 3 \rceil$ = 2

=> Shortcut

-> previous question can also be solved using this shortcut.

5. Number System

Date...11/03/19

(1) Intro. to no. syst.:-

→ Every digit of any no. system should be only one symbol.

ex:- Base-12 :- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B (12 unique digits)

(2) Conversion to base 10:-

Conversion from base b_1 → base b_2 :-

(i) convert from b_1 → decimal. (10) → (to base 10)
(ii) " " Decimal → b_2 (from base 10)

ex:- $(abc)_x = (?)_{10}$

→ $(ax^2 + bx + c)_{10}$

ex:- $(123)_7 \rightarrow (\ )_{10}$

→ $(7^0 \times 3 + 7^1 \times 2 + 7^2 \times 1) \Rightarrow (56)_{10}$

Shortcut :-

ex:- 101 = 5 then (1010) ⇒ 10
5 × 2 + 0
Base

ex:- (10111) ⇒ 23
1 1 × 2 + 1

(ex) $(125)_{10}$ ⇒ (12) × 10 + 5 ⇒ 125
(1 × 10) + 2 = 12

Date.....

ex:- $(1212)_3 = (?)_{10}$

⇒ $1 \times 3 + 2 = 5 \Rightarrow 5 \times 3 + 1 \Rightarrow 16$

$16 \times 3 + 2 \Rightarrow 50$

(3) Conversion from base 10 :-

ex:- $(11)_{10} \rightarrow (?)_2$

⇒

2 | 11		
2 | 5	1	
2 | 2	1	↑
2 | 1	0	
1 | 0	1	

$(1011)_2$

→ The no. of steps are $O(\log_2 X)$ where X = no. in decimal

This will be very high when X is very big.

Shortcut :-

→ first try to convert the no. in base 2^m & then from there convert it to binary by pairing m-bits together.

ex:- $(32)_{10} \rightarrow (?)_2$

⇒

16 | 32		
16 | 2	0	
0	2	

$(20)_{16} \Rightarrow (0010\,0000)_2$ (2, 0)

$\Downarrow$

32

(doing this way will reduce the no. of steps.)

Spiral

Date.....

***

While converting ~~power base~~ binary to 2^m always take the nos. from LSB.

ex:- $(100111011111)_2 \longrightarrow (?)_{16}$

(~~from LSB~~ Not MSB)

$\Downarrow$

$(4EF)_{16}$

***

While converting 2^m to binary take either from LSB or MSB:-

ex:- $(273)_8 \longrightarrow (?)_2$

$\Downarrow$

$(010 \mid 111 \mid 011)_2$

2 7 3

ex:- $(68)_{10} \longrightarrow (?)_2$

16	68	
16	4	4
	0	4

$(99)_{16} \longrightarrow (?)_2$

$\Rightarrow$

$(01001\,0100)_2$

ex:- $(127)_{10} \longrightarrow (?)_2$

32	127	
32	3	31
	0	3

$(3\,31)_{32} \longrightarrow (?)_2$

~~00011 11111~~

10

$(0011 \mid 11110)_2$

Date.....

(9) Minm no. of bits req. for inter conversion:-

(Q) Minm no. of bits req. to represent $(6728)_{10}$ in binary?

⇒ Maxm value that can be represented in a n-digit, base 'x' no. system $= x^n - 1$

So, $2^n - 1 \geq 6728$

$\Rightarrow 2^n \geq 6729 \Rightarrow n \geq \log_2 6729$

$\Rightarrow n = \lceil \log_2 6729 \rceil$

$\Rightarrow \boxed{n = 13}$

ex:- If no. is $(516)_7$ then no. of digits req. to represent in base 4?

⇒ 1st Convert it to base-10.

$5 \times 7^2 + 1 \times 7 + 6 \times 7^0 \leq 4^n - 1$

$\Rightarrow 245 + 13 \leq 4^n - 1$

$n = \lceil \log_4 259 \rceil$

(5) Minm no. of bits req. for inter conversion:-

(Q) What is the minm no. of digits req. to represent a 32 digit decimal no. in binary.

⇒ $10^{32} - 1$

largest no. in base 10, $\Rightarrow 10^{32} - 1$

$10^{32} - 1 \leq 2^n - 1 \Rightarrow 2^n \geq 10^{32}$

$n \geq \log_2 10^{32} \Rightarrow n \geq 32 \times \log_2 10$

Date.....

$n \geq 32 \quad \boxed{n = \lceil 32 \times \log_2^{10} \rceil}$

ex:- 10 digits in base 8 to base 2 then min. no. of digits req. in base 2 ?

⇒ $8^{10} - 1 \leq 2^n - 1$

⇒ $2^n \geq 8^{10} \quad n \geq \lceil \log_2 8^{10} \rceil$

⇒ $n = 30$

ie. each 3-bits in binary is 1 digit in base-8

6) Example 3:-

(Q) If $(11)_2 + (22)_3 + (33)_4 + (99)_5 = (abc)_6$ then find a, b, c.

⇒ $(11)_2 \rightarrow (\ ?\ )_{10}$

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

$(22)_3 \rightarrow (8)_{10}$

$(33)_4 = (15)_{10}$

$(99)_5 \rightarrow (29)_{10}$

$3 + 8 + 15 + 29 = (50)_{10}$

6	50	
6	8	2
6	1	2
	0	1

$\Rightarrow \boxed{(122)_6}$

Spiral

Date.....

(7) Example 4:-

(Q) From the following relation, determine the possible values of 'x'.

$(123)_5 = (x8)_y$

$\Rightarrow (1\times5^2 + 2\times5 + 3) = (x\times y^1 + 8)_{10}$

$38 = 8 + xy$

$xy = 30$

x	1	2	3	5	6
y	30	15	10	6	5

(y > 8) (x) (because x ≤ y)

$(x8)_y \Rightarrow$ (i) $y > 8$ (ii) $x < y$

{ x → 10 | 15 | 30 ; y → 3 | 2 | 1 }

(Not possible because x is a single digit)

x	1	2	3
y	30	15	10

So, possible values of $x = 1, 2, 3$

(8) Example 5:-

(Q) How many values of x & y are possible for $(42)_9 = (x3)_y$.

$\Rightarrow (4\times9 + 2)_{10} = (xy + 3)_{10} \Rightarrow xy = 35$

$(x3)_y \Rightarrow$ (i) $y > x$ & (ii) $y > 3$

Now, $xy = 35$

x	1	5	7	35
y	35	7	5	1

→ x (because y < x)

Spiral

→ Base of a no. system can't be -ve

Date.....

So, possible values of x & y are (1,35) & (5,7)

(9) Example 6 :-

ex:- $(123)_x = (12x)_3$ then $x =$ ___ ?

(a) 3 (b) -3,4 (c) 3, -4 (d) None

$(123)_x \Rightarrow$ (i) $x > 3$

$(12x)_3 \Rightarrow$ (ii) $x < 3$

} Conditions (11)

→ No values of 'x' are possible

(10) Example - 7 :-

(a) Determine the base of the no. systems from the following relations.

(a) $38 + 43 = 80$

(b) Roots of $x^2 - 11x + 22 = 0$ are 3 & 6

(c) $\frac{66}{6} = 11$

⇒ (a) let's assume base is b i.e.

$(38)_b + (43)_b = (80)_b$

$\Rightarrow (3 \times b + 8) + (4b + 3) = (8b + 0)$ } ⇒ in base - 10

$\Rightarrow 7b + 11 = 8b$

$b = 11$

Spiral

Date.....

(b) Roots of $x^2 - 11x + 22 = 0$

(b) $\Rightarrow 9 - 33 + 22 = 36 - 66 + 22 = 0$ { by putting '3' & '6' we'll be equal }

$\Rightarrow$ let's assume base is 'b'

$\Rightarrow (9)_b - (33)_b + (22)_b = (36)_b - (66)_b + (22)_b = 0$

$\Rightarrow 9 - (3b+3) + (2b+2) = (3b+6) - (6b+6) + (2b+2) = 0$

$\Rightarrow 8 - b = -b + 2 = 0$

$\boxed{b = 8 \text{ & } b = 2}$, but 2 is rejected because b>

Don't do like this

(a) $\frac{66}{6} = 11$

$\Rightarrow$ lets base is 'b'

$\frac{6b+6}{6} \rightarrow b+1 \Rightarrow 6b+6 = 6b+6$

(b) $x^2 - 11x + 22 = 0$

Product of roots $= \frac{c}{a}$

$\Rightarrow (3)_b * (6)_b = \frac{(22)_b}{(1)_b}$ { assuming base is b }

$(3 \times 6) = \frac{(2b+2)}{1}$

$\Rightarrow 18 = 2b+2 \Rightarrow \boxed{b = 8}$

Date.....

(c) $\frac{66}{6} = 11$

lets assume base is 'b'

$\frac{(66)_b}{(6)_b} = (11)_b \Rightarrow \frac{6b+6}{6} = b+1$

$\Rightarrow \boxed{b+1 = b+1}$

$\Rightarrow$ for all values of $\boxed{b > 6}$ {$\because$ 6 is the no.}

(11) Example 8 :—

(a) In a no. syst., the range of ~~no.~~ numbers are -3 to 3, represented as C, B, A, 0, 1, 2, 3. Express $(102)_{10}$ in this no. system.

$\Rightarrow$ Since base there are 7 digits thus base is 7

A = -1, B = -2 & C = -3 (since it is -3 to 3)

$\rightarrow$ remainder should be b/w -3 to 3

7	102	
7	15	-3 (because rem. 4 is not possible)
7	2	1
	0	2

$(21(-3)) \Rightarrow \boxed{(21C)}$

Ex:- for $(26)_{10}$ $\Rightarrow$

7	26	
7	4	-2
7	1	-3
	0	1

$\Rightarrow$ (1CB)

Spiral

→ Base of a no. syst. is called the Radix of that no. syst.

Date.....

(12) Complementary no. system:-

⇒ The main reason to introduce complementary no. syst. is to perform subtraction using the same ckt. as for addition.

→ for a Base-b no. system:- (Types of Complements)

(b-1)'s Complement (Diminished radix comp.)	(b's Complement) (radix compliment)
for no. 'x' :- 9's Diminished radix Compl. = $(b^n - 1) - x$	→ 9's radix Complement is :- $(b^n - x)$
where n = no. of digits in 'x'	i.e. Diminished Radix + 1 = Radix compl.
→ NOTE:- $b^n - 1$ is the max no. possible with n-digits with base 'b'	ex:- 1's compl + 1 = 2's Compl.

ex:- base-2 → 1's compl. $((2^n - 1) - x)$ → 2's compl. $(2^n - x)$

base 3 → 2's compl. $((3^n - 1) - x)$ → 3's compl. $(3^n - x)$

ex:- Base-10 & x = 685, find its 9's compl.

⇒ $(10^{(3)} - 1) - 685$ (3 → no. of digits in x)

$999 - 685 = 314$ { Subtracting each digit from 9 }

& 10's compl. = 314 + 1 = 315

Date

ex:- $(1011)_2$ → 1's compl. → 2's compl.

1's compl.: $(2^4 - 1) - x$

$\Rightarrow (15 - x)$

$\Rightarrow 1111 - 1011 \Rightarrow 0100$

2's compl.:

$0100 + 1 = 0101$

2's Compl.:-

⇒ start from LSB & keep every things as same till find the 1st one (including that '1') & after that complement each bit.

ex:- 1011 → (2's Com) → 0101

ex:- 1001 → 0111

ex:- 011100 → 100100

ex:- $(1239)_{16}$ → 15's Compl. / 16's Compl.

⇒ 15's Compl. ⇒ $(16^4 - 1) - x$

16^4 → max^m value of a no. in Hexadecimal with 4 digits

⇒ ffff − 1234 ⇒ (EDCB) {15 − 4 = 11(B)}

16's Compl. = EDCB + 1 = EDCC

Date.....

(13) Why we use Complements:-

(1) In case of diminished Radix Compl. no. system:- (ie. (b-1)'s Compl.)

$$X + \bar{X} = X + ((b^n - 1) - X)$$

$$\Rightarrow \boxed{b^n - 1} \Rightarrow$$ (***) it is considered as '0'

for Base-2:-

$$\Rightarrow \boxed{2^n - 1 = \underbrace{111\cdots1}_{n\text{ times}}} \Rightarrow$$ (***) it is equivalent to '0' in diminished Radix Compl. no. syst.

And also all 3eroes ie. $\underbrace{000\cdots0}_{n\text{-times}}$ is also '0'

$$\Rightarrow$$ So, in diminished (b-1)'s Compl. no. syst. '0' is either all 1's or all 0's. (***)

for Base-3:

2's Compl. $\Rightarrow$ 222 — n times, 0000 — n times } both are '0'

$$\Rightarrow$$ This is the disadv. of (b-1)'s Compl. no. system, ie. more than one representation for '0'. (***)

(2) In Case of Radix Compl. no. system (ie. b's Compl):-

$$X + \bar{X} = X + b^n - X$$

$$\Rightarrow \boxed{\underbrace{b^n}_{(n+1\text{ digits long})}} \Rightarrow$$ (***) b^n is considered as '0'

$b^n = 1\underbrace{0000\cdots0}_{n\text{ times}}$

So by subtracting two nos. the no. of digits should not ↑ thus 1 is considered to be not present & thus the result is all 3eroes.

Adv. of b's compl. no. syst. :— Date.....

→ So, after discarding 'c', we get 'n' zeroes which is a single representation of '0' in b's complement no. syst.

→ Because of this adv. the b's compl. no. system is widely used not (b-1)'s compl.

(14) Subtraction in diminished radix compl. :— (b-1)'s compl

for Base 10 no. :—

$x - y \Rightarrow x + \bar{y}$

$\Rightarrow x + (10^2 - 1 - y)$ { assuming x & y are two digit nos }

$(b^{n-1} - y)$

$\Rightarrow (x-y) + 99$ { if carry is generated then subtract 100 }

$\Rightarrow (x - y - 1)$

$x - y - 1 \; (+1)$

$\Rightarrow \boxed{x - y}$

i.e. $\boxed{x + (9\text{'s comp. of } y)}$ => And if there is carry then wrap around & Add.

Ex:- 97 - 23.

$\Rightarrow$ 9's Comp. of 23 = 99 - 23 = 76

$97 + 76 \Rightarrow 173 \Rightarrow 73 + 1 = 74$ (carry)

Date.....

ex:- ⇒ $X-Y \Rightarrow X+\bar{Y}$

$\Rightarrow X+(b^n-1-Y) \Rightarrow \boxed{b^n-1+(x-Y)}$

$\Rightarrow (b^n-1)+(x-Y)$ (if $X-Y \leq 0$)

$\Rightarrow (b^n-1)-\left((b^n-1)+(X-Y)\right)$

$\Rightarrow \boxed{-(X-Y)}$ ***

Summary ****

$\boxed{b^n-1+(x-Y)}$

if $x-Y \geq 1$ → then carry is generated & wrap around is needed

if $X-Y \leq 0$ → NO, carry then take Complement of the final ans. again.

***

ex:- 23 - 97

⇒ 9's compl. of 97 ⇒ 99 − 97 = 2

⇒ 23 + 2 = 25 {NO carry}

So, again take the 9's compl. of 25

99 − 25 ⇒ 74

$-(x-Y) = 74 \Rightarrow \boxed{(x-Y) = -74}$ **

Summary:-

Date..... 12/03/18

(15) Examples on diminished radix complement :-

Base 10 :-

ex:- 87 - 6 { first make sure both nos. have same no. of digits } ***

=> 9's compl. of '06'. => ~~99-6~~ 99-06 = 93

87 87+93 => 180 => 80 +1 = 81 { Carry 1 is generated }

ex:- 6 - 87 { first make sure both nos. have same no. of digits }

=> 9's Compl. of 87 => 99-87 => 12

~~99~~ (06+12) = 18 { No carry }

Now, Complement of 18 => 99-18 => 81

-(X-Y) = 81 => X-Y = -81

ex:- ~~45~~ 456 - 213

=> 9's compl. of 213 => 999-213 => 786

Now, 456+786 = 1242 { carry is generated }

=> 242 +1 = 243

Date.....

ex:- 213 - 456

⇒ 9's compl. of 456 ⇒ 999 - 456 ⇒ 543

⇒ Now, 213 + 543 ⇒ 756

Since there is no carry find of 9's compl. of 756

⇒ 999 - 756 ⇒ 243 = -(X-Y)

⇒ X-Y = -243

Base - 2 :-

→ Diminished Radix compl. in base-2 no. syst. is 1's compl.

ex:- 1011 - 0101

⇒ 1's compl. of 0101 ⇒ 1010

Now,

```
  1011
  1010
 10101
```

⇒ Carry is generated { wrap around the carry }

⇒ 0101 + 1 = 0110 { 11 - 5 = 6 }

ex:- 0101 - 1011

⇒ 1's compl. of 1011 ⇒ 0100

⇒ 0101 + 0100 = 1001

{ No carry ⇒ 1001 —1's compl.→ 0110 }

-(X-Y) = 0110

X-Y = -0110

Date.....

ex:- 10100 − 111 { first make no. of digits same }

=> 1's compl. of 00111 => 11000

```
  10100
+ 11000
-------
 101100
```

carry

=> 01100 + 1 = 01101 { 20 − 7 = 13 }

ex:- 111 − 10100 { Make no. of digits same }

=> 00111

1's compl. => (01011)

```
  00111
+ 01011
-------
  10010
```

{ No carry. }

So, 1's compl. of 10010 => 01101

∴ (X−Y) = 01101 => X − Y = − 01101

Base 3:-

-> 2's compl. is diminished radix compl. in base 3

ex:- 212 − 121 = ?
(23) − (16)

=> 2's compl. of 121 => 222 − 121 => 101

```
  212
+ 101
-------
 1020 {carry}
```

020 + 1 = (021) -> 2×3+1 => 7

Date.....

ex:- 121 − 212 = ?

⇒ 222 − 212 ⇒ 010

Now,

```
  121
+ 010
-----
  201
```

{ No carry so ⇒ 222 − 201

⇒ 021

−(X − Y) = 021.

X − Y = − 021

7

(16) Examples On subtraction in radix compl:- (b's compl)

Base 10:-

→ b's compl. in base-10 is 10's complement.

ex:- 97 − 23 = ?

⇒ find 10's compl. of '23'.

99 − 23 = 76 + 1 = 77

Now,

```
  97
+ 77
-----
 174
```

⇒ If carry then ignore it

So, 74

Date.....

ex:- 23-97 =?

→ find 10's compl. of 97 ⇒ 99-97 = 02 + 1 ⇒ 03

→
$$\begin{array}{r} 23 \\ +\,03 \\ \hline 26 \end{array}$$
{since there is no carry find 10's compl.}

⇒
$$\begin{array}{r} 99 \\ -\,26 \\ \hline 73 \\ +\quad 1 \\ \hline 74 \end{array}$$
→ $-(X-Y) = 74 \Rightarrow X-Y = 79$

for b's complement subtraction:-

case 1:-
(when $X-Y \geq 0$)
→ carry is generated

(i) first find the b's compl. of the no. which is being subtracted.
(ii) Add the b's compl. to the other no. if there is carry then ignore it

$X + (b^n - Y) \Rightarrow b^n + (X-Y)$
(b^n: b's compl.; b^n: ignore)

Case-2:-
(when $X-Y \leq 0$)
→ No carry is generated.

(i) first find the b's compl. of the no. which is being subtracted.
(ii) Add this to the other no.
(iii) Again find the b's compl. of result
(iv) This will be − (Answer)

$\Rightarrow X + (b^n - Y) \rightarrow X +$ b's compl. of Y
$\Rightarrow b^n + (X-Y)$
$\Rightarrow b^n - (b^n + (X-Y)) \rightarrow$ b's compl. of result
$\Rightarrow \boxed{-(X-Y)}$

Spiral

Date.....

ex:- 456 - 132 = ?

⇒ 999 - 132 ⇒ 867 + 1 ⇒ 868

```
  456
+ 868
-------
x(1)324   ⇒ 324 Ans
```

ex:- 132 - 456 = ?

⇒ 999 - 456 = 543 + 1 ⇒ 544

132 + 544 ⇒ 676

⇒ 999 - 676 ⇒ 323 + 1 = 324 [10's compl. of 676]

-(X-Y) = 324 ⇒ (X-Y) = -324

Base - 2:-

→ Radix compl. in Base - 2 is 2's compl.

ex:- 1011 - 0101 = ?

⇒ 2's compl. of 0101 ⇒ 1011

```
  1011
+ 1011
-------
(1)0110   ⇒ delete the carry
```

⇒ 0110

Date.....

ex:- 0101 - 1011 = ?

⇒ 2's Compl. of 1011 ⇒ 0101

```
  0101
+ 0101
------
  1010   } ⇒ No carry
```

So, take 2's compl. of 1010 ⇒ 0110

$-(X-Y) = 0110 \Rightarrow (X-Y) = -0110$

→ Here we can see in b's compl. subtraction there is no wrap-around carry. so it's better than (b-1)'s Compl. subtraction.

#Base 3:-

→ In base-3 b's compl. is 3's compl.

ex:- $(212)_3 - (121)_3 = ?$

⇒ 3's compl. of 121 ⇒ 222 - 121 ⇒ 101 + 1

⇒ 102

```
  2 1 2
- 1 0 1
-------
  0 2 1
```

```
  2 1 2
+ 1 0 2
-------
(1)0 2 1
```

carry

$(9)_{10} = (11)_3$

$(3)_{10} = 10$

So, 021 → Answer.

Date.....

ex:- 121 - 212 = ?

⇒ 222 - 212 ⇒ 010 + 1 ⇒ 011

```
  121
+ 011
-----
  202   {No carry}
```

So, 222 - 202 ⇒ 020 + 1 ⇒ 021

021 ⇒ -(X-Y) ⇒ (X-Y) = -021

{-7}

***

→ All the subtraction done so far i.e. Radix compl. subtraction & diminished radix compl. are done for unsigned nos.

(17) Summary of subtraction using complements in case of unsigned nos:-

(1) (b-1)'s Complement:-

→ The subtraction of two n-digit unsigned nos. M-N (N≠0) can be done as follows:-

(i) Add M to (b-1)'s compl. of N. This performs:-
$M+(b^n-1-N) \Rightarrow (b^n-1)+(M-N)$

(ii) If $(M-N) \geq 1$ then sum will produce an "end carry" b^n, which will be discarded & 1 will be added (End around carry) which gives M-N.

(iii) ~~If $(M-N) < 1$, then sum does not p~~

Date... 15/03/19

(iii) If $(M-N) < 1$, then sum does not produce an end carry, to obtain the answer in the familiar form, take $(b-1)$'s compl. of the sum & place a negative sign in front.

(2) b's complement :-

→ The subtraction of two n-digit unsigned nos. $M-N$ (where $N \neq 0$) in base 'b' can be done as follows.

(i) Add M to b's compl. of N. This performs
$M + (b^n - N) \Rightarrow b^n + (M-N)$.

(ii) If $M-N \geq 0$ then sum will produce an end carry b^n, which will be discarded which gives $(M-N)$

(iii) If $M-N < 0$ the sum does not produce an end carry, To obtain the answer in the familiar form, take (b's) compl. of the sum & place a -ve sign in front.

(18) Signed no. representation :-

Binary nos.

- Unsigned nos. (only +ve nos.)
- Signed nos. (both +ve & -ve)
  - → signed Magnitude form
  - → " 1's compl.
  - " 2's " } (This is how we store nos in Computer)

Spiral

~~0.13~~

. 0; 13

↓

for base-10 the point is called decimal point & generally Radix point for any base

Date.....

→

Numbers (any general nos. not only binary)

Integers	Fraction	Integer with fraction
(ex:- 10)	(ex:- 0.14)	(ex:- 10.14) =
→ assume the point is at extreme right of the no.	→ point is at the extreme left of the no.	→ Point is not fixed it can be anywhere.
↓	↓	↓
fixed point nos.	fixed point nos.	floating point nos.

~~***~~

→ In our study of signed nos. we assume the binary nos. are integers in its decimal form.

(19) Example on signed no. representations:-

=

→ for signed nos:- if MSB is 0 → +ve no.
" " " 1 → -ve no.

	Unsigned	signed magnitude	1's compl.	2's compl.
0 0 0	0	0	0	0
+ve 0 0 1	1	1	1	1
0 1 0	2	2	2	2
0 1 1	3	3	3	3
1 0 0	4	-0	-3	-4 ***
-ve 1 0 1	5	-1	-2	-3
1 1 0	6	-2	-1	-2
1 1 1	7	-3	-0	-1

Unsigned nos. | Signed nos.

~~→~~

Date.....

(i) Signed magnitude:- MSB is for sign & rest bits are the general bits

disadv:-

→ '0' is represented twice ie. 000 & 100 (so one combⁿ is wasted)

→ for Any operation (like addition or subtsacⁿ) first signed bit need to be compared, so seperate h/w is required for this & Addⁿ & subtsⁿ are done using two diff. circuits.

→ Not used in any computers

(ii) 1's compl.:-

→ The -ve no. in 1's compl. no. syst. is represented using 1's compl. of it's positive counterpart.

ex:- 001 ⇒ 1 ⇒ 110 ⇒ -1

→ All 1's in a no. is '0' in 1's compl. syst.

Disadv:-

→ '0' has got Combⁿ. (all 1's & all 0's)

→ End around carry is generated.

Adv:-

→ We don't need diff. h/w circuits for Addⁿ & Subtraction.

→ It was used in earlier computers, but not now.

Date.....

(iii) 2's Compl. :- The -ve no. is 2's compl. no. system is represented by 2's complement of it's +ve counterpart.

ex:- 001 ⇒ 1 ⇒ 111 ⇒ -1

Adv:-

(i) Only one Combⁿ for Zero.

(ii) Range is more compared to signed magnitude & 1's Compl. no. sysd.

(iii) same h/w for Addⁿ & Subtrⁿ

(iv) No End around carry.

Weighted Nos:- {weight is assigned to each bit}

(i) Unsigned nos. are weighted;- 2^2 2^1 2^0

(ii) signed magnitude:- S 2^1 2^0

$(-1)^S \times (\quad)$

(iii) 2's Compl.:-

-2^2	2^1	2^0

ex:- 100 ⇒ -4

101 ⇒ -3 $\{-2^2 + 01 = -3\}$

110 ⇒ -2

(iv) 1's Compl. :- It is not weighted

Spiral

Date 15-03-2019

(20) Ranges of signed no. representation:-

no. of bits	signed magnitude	1's compl.	2's compl.
n	$-(2^{n-1}-1)$ to $(2^{n-1}-1)$	$-(2^{n-1}-1)$ to $(2^{n-1}-1)$	$-(2^{n-1})$ to $(2^{n-1}-1)$

ex:- Minm no. of bits req. to represent '15' in 2's compl. no. syst.

$\Rightarrow$ $15 \leq (2^{n-1}-1)$ $\Rightarrow$ $2^{n-1} \geq 16$

(check the inequality)

$\Rightarrow n \geq 5 \Rightarrow n_{min} = 5$

ex:- Minm no. of bits req. to represent '-30' in 2's compl. no. syst.

$\Rightarrow$ $-30 \geq -2^{n-1}$ $\Rightarrow$ $2^{n-1} \geq 30$

$n-1 \geq \log_2 30$

$n \geq \lceil 5.5 \rceil \Rightarrow n = 6 \Rightarrow$ Minm

Check the inequality in case of the -ve nos.

(21) Examples on Ranges:-

ex:- find min. no. of bits to represent $(+32)_{10}$ in 2's compl.

$\Rightarrow$ $32 \leq (2^{n-1}-1)$ $\Rightarrow$ $2^{n-1} \geq 33$

$n-1 \geq \log_2 33$

$n \geq \lceil 6.-- \rceil \Rightarrow n = 7$

ex2:- find min. no. of bits to represent $(-32)_{10}$ in 2's compl.

$\Rightarrow$ $-32 \geq -(2^{n-1})$ $\Rightarrow$ $2^{n-1} \geq 32$

$n \geq 6 \Rightarrow n_{min} = 6$

Spiral

Date.....

ex:- find maxm +ve no. that can be represented by 10 bits in 2's compl.

→ $2^{n-1}-1$ ⇒ Maxm +ve no. with n-bits in 2's compl.

⇒ $2^{10-1}-1$ ⇒ 511

(22) sign bit extension:-

→ When data is copied from lower sized registers (or memspace) to higher sized registers.

ex:- 0110 = +6 (4-bit reg)

0000 0110 = +6 (8-bit reg.)

Unsigned nos.

for unsigned nos. just append 0's to MSBs.

for signed nos:-

(i) Signed magnitudee nos:-

ex:- -6 = 1110 ⇒ 4 bit reg.

but, 0000 1110 ⇒ 8-bit reg.

⇒ 14

it is not -6, thus value is changed when data is copied from lower reg. to higher reg.

Extension of signed magnitude nos:

→ place the MSB of actual no. in MSB of new higher bit no. & fill remaining with '0'

ie. 1000 0110 ⇒ -6.

Date.....

(ii) 1's complement no. system :- (sign bit extension)

~~ex:- +001 → 6 { 4-bit reg.}~~

***

→ In this case find the binary representation for +ve of the given -ve no. then take its 1's compl.

ex:- 1001 => -6 { 4-bit register }

⇓

00000110 => 6 { 8-bit }

↓ 1's compl.

11111001 = -6

***

ie: Take the signed bit (ie. 1) & copy it in the MSB & rest remains the same.

ex:-

=> -6 = 1001 → 4-bit

=> -6 =

1111	1001

(iii) 2's complement no. system :- (signed bit extension).

***

Interesting property of 2's complement no. system:-

ex:

=> 1101 = -3

=> 1101 => -3

=> 101 => -3

ex:- 111101 = -7

=> 111001 = -7

=> 11001 = -7

=> 1001 = -7

Spiral

Date.....

→ The -ve no. in 2's complement remains same by deleting all 1's except the one in MSB except one '1'.

ie: ex:- if there are 9 1's in MSB then after deleting 3 1's the no. will remain same.

111 10000101 = -123

⇒ 10000101 = -123

Sign bit extension in 2's compl.:-

⇒ 1101 = -3 {4-bit}

1111 1101 = -3 {8-bit}

→ when the no. is +ve ie. '0' in the MSB simply add 0's in all the MSB for signed as well as unsigned nos.

ex:- 0101 ⇒ 5 (4-bit)

0000 0101 ⇒ 5 (8-bit)

(Q) find decimal equiv. of fff3 (in 2's compl.)?

⇒ 1111 1111 1111 0011 ⇒ $-2^4 + 2^1 + 2^0$

⇒ -13

Spiral

Date..18-03-2019

23) Example on signed bit extension:-

(Q) A 2's complement no. $N = P_3P_2P_1P_0$ is transformed as $P_3P_3P_3P_2P_1P_0$ which of the following operⁿ is performed on this no.

⇒ sign bit extension; so $P_3P_3P_3P_2P_1P_0$ (1)

P_3P_3 → discard

(1) → appending 1 is Multiplying the no. by 2 & adding '1'

↓

operation

(24) Overflow:-

⇒ Result can't fit in the space given, leads to overflow.

(i) Overflow in unsigned no.:-

→ carry out of MSB leads to overflow.

ex:-

1000 → 8
1100 → 12
(1)0100 → 20

{ range for 4-digit ⇒ 0 to 15 }

overflow.

(ii) Overflow in signed no.:-

(a) for 2's compl. no. syst.:-

⇒ In case of 2's compl. no. system a carry out of MSB doesn't necessarily mean there is overflow.

ex:-

1010 (-6)
0110 (6)
(1)0000 ⇒ (0)

{ carry is ignored & we got the right result so, no overflow. }

Spiral ~~No overflow~~ NO overflow (with carry)

Date.....

⇒ So, in 2's compl. no. system overflow will occur ~~only~~ when two -ve nos. are added & result is +ve. (If both nos. are +ve or one -ve one +ve then there will be no carry).

ex:- 1 0 0 0 (-8)
1 1 0 0 (-9)
1 0 1 0 0 ⇒ overflow (due to carry)

↳ The result in 4-bit represents +ve no. thus there is overflow because adding two -ve nos. will not result in +ve no.

⇒ In 2's compl. no. syst. overflow can occur when two +ve nos. are added & result is -ve. ~~then~~ at (Because addn of two +ve nos. will give +ve no.)

ex:- 0 1 1 1 (7)
+ 0 0 0 1 (1)
1 0 0 0 (-8) ⇒ overflow (w/o carry)

↳ Even though there is no carry ie. result fits in 4-bit, but it is overflow.

Rules to detect overflow:-

(1) Unsigned nos:- carry from MSB will indicate of

(2) Signed nos:- (i) Two +ve nos. added & result is -ve
(ii) " -ve " " " " " +ve

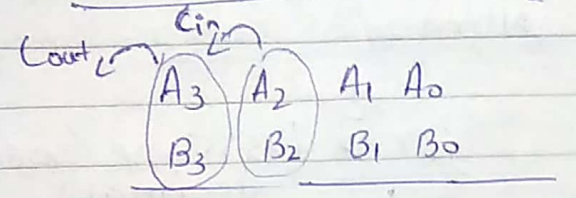
Overflow in signed nos.

Spiral

Date.....

Carry doesn't indicate 'Of' in case of signed nos.

Shortcut to decide whether there is Of or not :-



Cin = carry into the MSB
Cout = " out of " ".

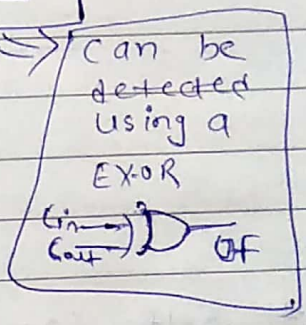
Case (i) :- If Cout ≠ Cin then Of

Case (ii) :- If Cout = Cin " NO Of

ex:- 1000
1100
10100

Cin = 0 & Cout = 1
0 ≠ 1 ⇒ Of

⇒ Can be detected using a EX-OR



ex:- 1010
0110
10000

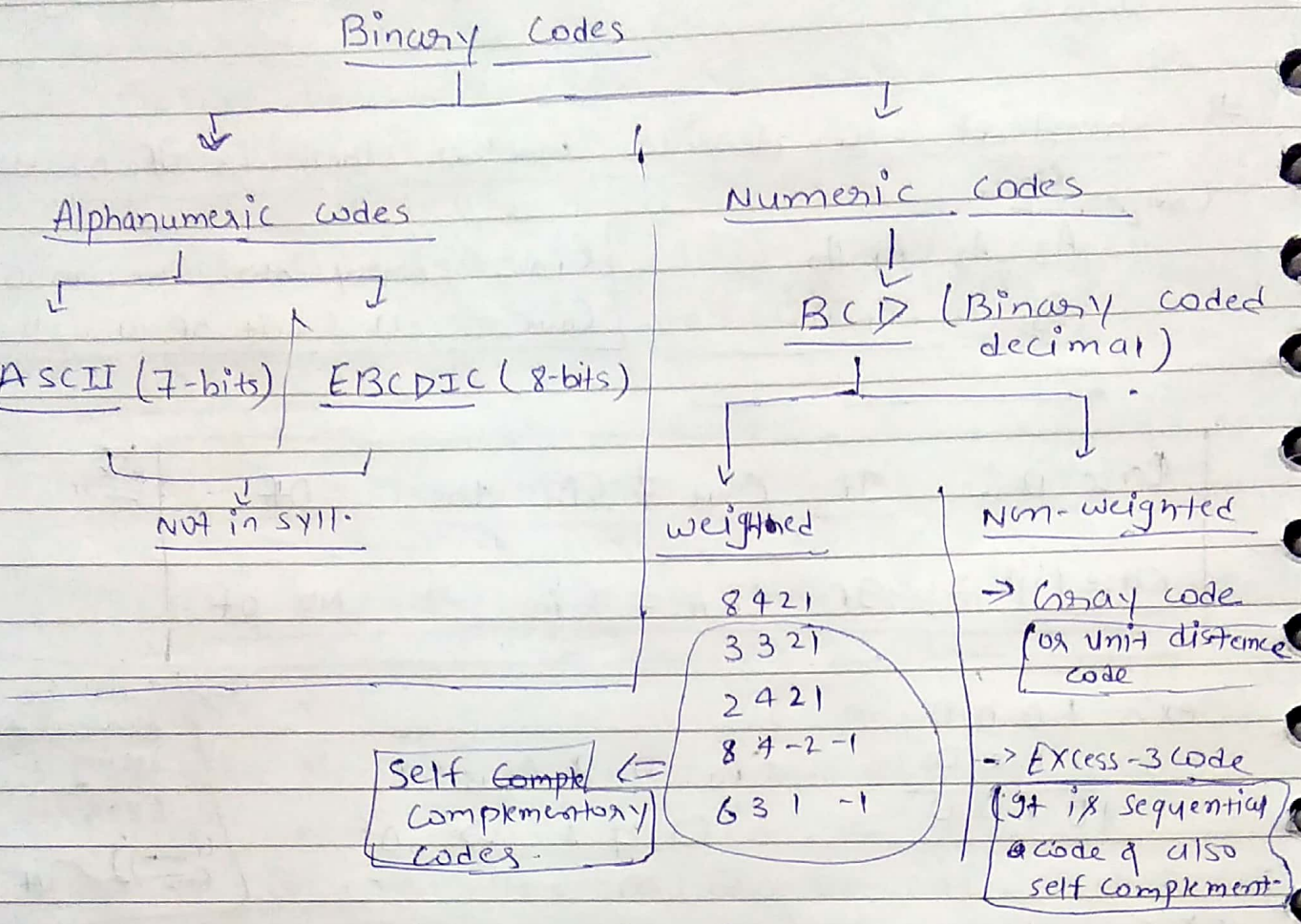
Cin = 1, Cout = 1
so, Cin = Cout ⇒ NO Of

ex:- 0111
0001
1000

Cin = 1, Cout = 0
Cin ≠ Cout ⇒ Of

Date.....

25) Classification of binary codes:-



Self complementary code:- In self complementary code, the code of a digit and the code of 9's compl. of digit are 1's compl. to each other.

ex:- 6 3 1 -1 (code)

0 1 0 1 (2) ⟷ 9's compl.

1's compl ⟷ 1 0 1 0 (7)

Unit distance code:- There is only one bit difference b/w two nos.

ex:- 0 0 0 0 → unit dist.
0 0 1 0

ex:- 0 0 1 0 → Not unit dist.
0 0 0 1 (two bits are changed)

Date.....

(26) 8421, Excess-3, 3321 codes :-

(also called BCD) → 8421

Excess-3

3321 → self complementary

Decimal digit	8421	Excess-3	3321
0	0000	0011	0000
1	0001	0100	0001
2	0010	0101	0010
3	0011	0110	0011
4	0100	0111	0101
5	0101	1000	1010
6	0110	1001	1100 → 1's compl. of 3
7	0111	1010	1101
8	1000	1011	1110
9	1001	1100	1111

sequential code

In case of 3321 more than one combⁿ can be possible for a single no; to avoid this some ~~Combⁿ~~ convention are ~~not~~ followed :-

***

⇒ for nos. ≤ 4 always go with the smallest binary no. & for nos. > 4 take 9's compl. of no. & take 1's compl

ex: for '3' possible representations are :- (in 3321)

0011, 1000, 0100

(this will be chosen)

ex: for '4' possible representations are :-

3321

1001

0101 → This will be chosen.

Date.....

ex:- '5' in 3321.

⇒ 5 > 4 so, 9's compl. of 5 ⇒ 9-5 = 4

~~4 ir take~~ NOW, 4 = 0101

↓ 1's compl.

1010

Generally when sum of codes of a weighted code is '9' that code is self complementary.

ex:- 3321 ⇒ 3+3+2+1 = 9 (self-compl.)

ex:- 2421 ⇒ 9 (")

ex:- 8421 ⇒ 15 (Not self-compl.)

→ 9 is not always true if sum is 9 the weighted code will be self-compl. but if for a weighted code is sum of their code is not '9' then that is definately not self. compl.

→ neccessary Condⁿ.

→ Excess-3 is also self-complementary

Sequential Code:- Add 1 to previous ~~code~~ no. to get the next no.

ex:- 8421, excess-3

⇒ since Excess-3 is self complementary as well as sequential; this is why it is papularily used.

→ *** unlike binary nos. on BCD or 3321 or EX-3 each digit of a decimal no. is converted indivisually

Date.....

27. Examples on Codes :-

(Q) which of the following is Invalid 8421 Code?
(a) 0010 (b) 1100 (c) 1001 (d) 0111

8421 also called as BCD

⇒ The no. should not be > '9'

(Q) which of the following is invalid Excess-3 Code?
(a) 0011 (b) 1100 (c) 1101 (d) 1001

⇒ The no. should be in [3, 12]

*** (Q) Represent $(863)_{10}$ in
(a) BCD, (b) 2291, (c) 3321 (d) EX-3 (e) Binary

⇒ (a) BCD :-

8 6 3
1000 0110 0011 ⇒ 863
BCD

(e) Binary :-

16 | 863
16 | 53 (15) = f
16 | 3 5
0 3

$(3\ 5\ f)_{16}$ →

$(0011\ 0101\ 1111)_2$

(b) 2291 :-

$(8\ 6\ 3)_{10}$
1110 1010 0101 ⇒ 2291
(1's compl. of 1) (1's compl of 3)

(c) 3321 :-

8 6 3
(1110) (1100) (0011)

⇒ (1110 1100 0011)$_{3321}$

(d) EX-3 :-

$(8\ 6\ 3)_{10}$
1011 1001 0110
(1011 1001 0110)$_{EX-3}$

Date.....

(2e) BCD addition:-

ex:- 3 — 0011
+ 6 — 0110
9 — 1001 (9) ✓

ex:- 3 → 0011
+ 7 — 0111
10 — 1010

9+ is 10 in binary no in BCD this is invalid comb^n for BCD.

10 (in BCD) ⇒ (0001 0000)

So, in BCD addition when we get invalid comb^n we add '6' (0110) to that no. to get a BCD no. ('6' to bypass '10' to '15' invalid comb^{n}s)

ex:- 3 → 0011
+ 7 — 0111
1010 (invalid)
+ 0110 (6)
10000
⇒ 0001 0000
⇒ 10 in BCD

ex:- 9 → (1001)
+ 9 → (1001)
10010 { 9+ is '12' in BCD which is wrong }
1 2

So, 10010
+ 0110
10000 ⇒ ('18' in BCD)
1 8

Date.....

ex:- 123 + 789 = 912

0001 0010 0011
0111 1000 1001

1100 → invalid
+0110 (6)
1 0010

1011
+0110 (6) → invalid
1 0001

valid ← 1001

So the result is:- 1001 0001 0010
9 1 2

(29) Excess-3 Addition:-

Rules:-

(i) If carry occurs then "add 3"
(ii) If No " " " "add -3"

⇒ derived from BCD addⁿ '+3' to Ex-3 is equiv. to '+6' in BCD & when no carry '-3'

→ I/p & O/p should have same no. of digit.

ex: 7 → 1010
+2 → 0101
9 1111 {No carry so subtract 3}
-0011
1100 =
↳ 9

ex:- 07 + 03 = 10 → make 7 & 3 two digit nos. by adding '0'

'0' in x's-3 → 7 in x's-3

0000 0010
0011 0110
0111 1000
carry ← 0000
+0011
No carry ← 0111 0011
-0011
0100

So, result is:-
0100 0011
1 0
↓
in x's 3

→ unit distance property of gray code is useful for counting in counters, because clock pulse is changed only once.

Date.....

(30) Gray code :-

properties of Gray code :-

(i) Reflexive :- ~~for a n-bit no. 'n' bit comb~~

(ii) unit distance :- → Every successive no. differs by 1-bit.

(iii) cyclic → Distance b/w last no. & 1st no. is also 'one'

(iv) Non-weighted code :-

→ gt's not like other codes (like X's-3, 8421 etc.) in which each digit of a no. is ~~represented by code~~ coded. gt is like binary nos.

ex:- '12' in ~~8421~~ 8421 ⇒ 0001 0010
1 2

but in binary 12 → 1100

Gray codes with diff. no. of bits :-

1-bit	2-bit		3-bit	
0 → 0	0 → 0	0	0 → 0	0 0
1 → 1	1 → 0	1	1 → 0	0 1
	2 → 1	1	2 → 0	1 1
	3 → 1	0	3 → 0	1 0
			4 → 1	1 0
			5 → 1	1 1
			6 → 1	0 1
			7 → 1	0 0

mirror images (thus reflexive)

mirror images

similarly for 4-bits

→ Distance b/w two nos. is ~~how~~ no. many no. of bits changed to get from one no. to other.

Reflection of all the comb^n possible by 'n-1' bits for 'n-bit' no.

thus called Reflexive

Spiral

Date.....

(31.) Binary to gray & vice versa:-

→ Since gray is non-weighted directly finding gray code of a no is difficult, so 1st convert no. to binary & then from binary to gray

Binary to gray:-

ex:- ⊕ ⊕ ⊕ → ex-or

12 → 1 1 0 0

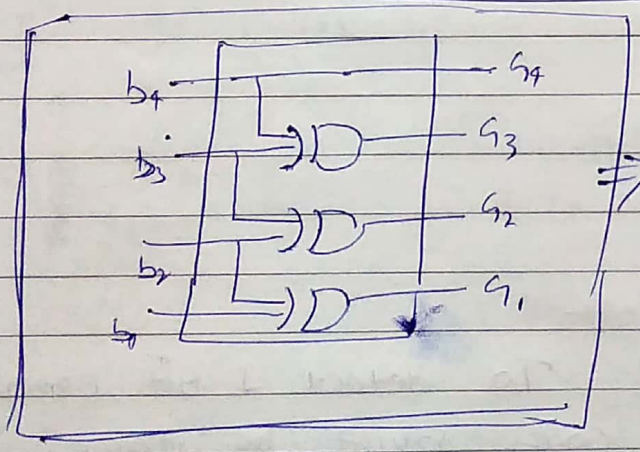
1 0 1 0 ⇒ 12 in Gray code

so, binary no:- $b_4\ b_3\ b_2\ b_1$ & Gray code:- $g_4\ g_3\ g_2\ g_1$

then binary to gray:-

$g_4 = b_4$, $g_3 = b_4 \oplus b_3$, $g_2 = b_3 \oplus b_2$

$g_1 = b_2 \oplus b_1$



⇒ Binary to gray converter

ex:- 200 in binary ⇒ 1 1 0 0 1 0 0 0

1 0 1 0 1 1 0 0 ⇒ Gray

Date.....

Gray to binary :-

ex:- (12) 1 0 1 0 → Gray

(12) 1 1 0 0 → Binary

$b_4 = g_4$, $b_3 = b_4 \oplus g_3$, $b_2 = b_3 \oplus g_2$, $b_1 = b_2 \oplus g_1$

$b_3 = g_4 \oplus g_3$ $b_2 = g_4 \oplus g_3 \oplus g_2$ $b_1 = g_4 \oplus g_3 \oplus g_2 \oplus g_1$

ex:- 1000 = 15 (Gray)

= 15 (binary)

(32) Error detection :-

Error Handling :-

→ Error detection

→ " Correction.

(i) Error detection :-

(a) 1-bit error detection :- To detect 1-bit error the distance b/w each codes must be two.

→ If an error is changing a given valid no. to other valid no. then Error detection is impossible.

ex:- 0000 (data) —error→ 0001 → 1-bit error

BCD nos.

Spiral

↓

it is also valid no. in BCD, so error can't be detected.

Date.....

ex:- 0000 (original data) ⟶ 1100 (error) { BCD nos. }

↓↓

[invalid no. so reciever will detect the error]

ex:- If patterns used in a code are as follows :-

0000
0001
1001
1100

⇒ Can this code be used for 1-bit error detection?

⇒ for this code to detect 1-bit error, ~~it has to~~ Every no. should differ each other by a distance of '2'.

∴ but, 0000 } unid dist. { so, it can't be used fo single bit error detecting
0001

***

ex:- If codes given are :-

0000
0011
1010
1001

Can this be used for 1-bit error detection?

⇒ only valid codes

⇒ Dist. b/w each code is two. { so it can be used for 1-bit Error detec.

Now, suppose ~~0000~~ changes to:

0000 ⟶ 0001

↓↓

This is not a part of our valid code. so it's an error, thus error is detected.

Date..19-03-2019

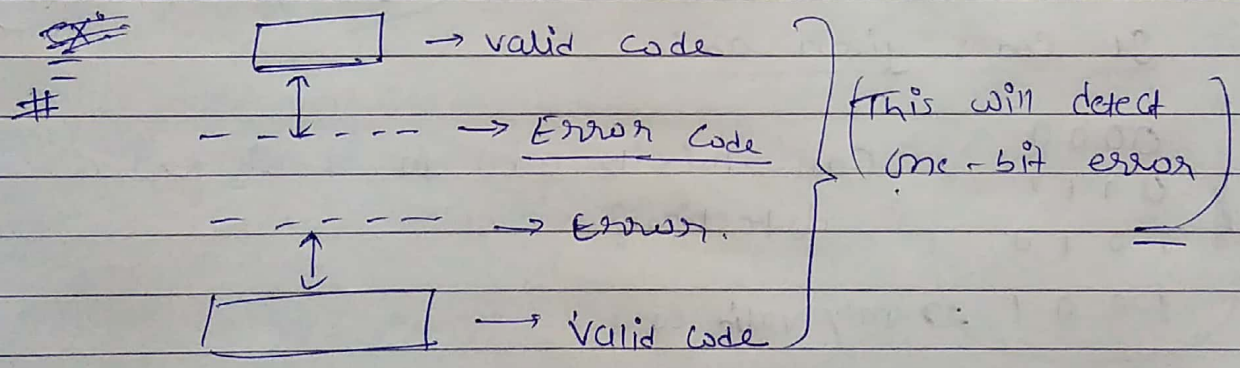
→ Similarly, for n-bit error the min. dist. b/w every two valid codes should be atleast 'n+1'

(i) The min. dist. b/w two valid codes is also called hamming dist.

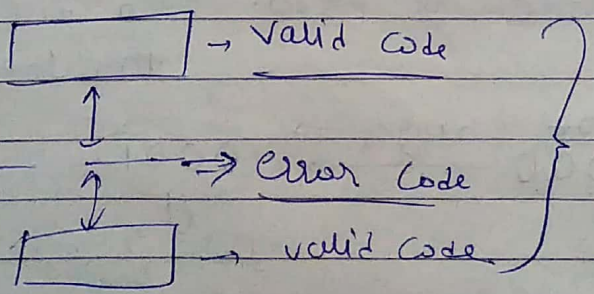
So, hamming dist ≥ n+1.

(33.) Error Correction:-

→ In error detection we needed to check whether the error no. belongs to valid codes or not, but in error Correction, after detecting the error, we need to figure out, which valid code to use for error correction.



but



Ambiguous because which valid code to chose when error occurs (so, this can't be used for 1-bit error)

Spiral

Date.....

***
→ So, for 1-bit Error correction a distance of min. '3' is required b/w any two valid codes.

→ Similarly, for n-bit error correction a distance (hamming distance) of min "2n+1" is req. b/w any two valid codes.

****

So, ⇒ For n-bit error detection ⇒ hamming dist. ≥ n+1

⇒ " " " " correction ⇒ " " ≥ 2n+1

***
⇒ But, by choosing the larger hamming distance we might run out of possible combinations to assign to valid codes.

ex:- If there are 4-bits then possible comb's are '16', so to get 10 valid codes from them the hamming dist. should be atmost '1'.

∴ if hamming dist. is '2' we might need to ↑ the no. of combinations by ↑ no. of bits to get 10 valid codes.

**
⇒ Since gray code have hamming dist. '1', so to increase the hamming dist. in gray code a parity bit (even or odd) can be added to make hamming dist. '2'.

→ Even parity :- No. of 1's are even.

→ odd " :- " " " " odd.

Date.....

ex:- Increasing the hamming dist. of gray code from 1 to 2:-

3-bit gray code:-

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

← adding a bit to make it even parity

↳ rejection

***

⇒ Now the hamming dist. of this code is made '2' so, it can now be used for 1-bit error detection.

(34) Hamming Code:- (1-bit Error-Correction technique.)

⇒ In digital we focus on 1-bit Error correction because it is most frequent error.

⇒ Message size (M) ⇒ actual data.
Parity bits added (P) ⇒ Redundant).

{ Parity used is even parity }

Total data = (M+P) bits

****

→ Total no. of possibilities, for 1-bit error = (M+P) + 1
(or cases)

~~Total no. of cases~~

'M' possibilities for error at any bit of actual data

P - possibilities for error at any bit of parity bits added

Or No error

Spiral

Date.....

→ Now, this error possibilities should be dealt by p-bit (which is added for error-correction)

So) $\boxed{2^P \geq (m+p)+1}$ ***

ex:- for 4-bit message m = 0101 what should be min no. parity bits are added for 1-bit error correction?

⇒ Total cases for 1-bit error = (4+p)+1

& $2^P \geq 4+p+1$.

for p=0 ⇒ 1 ≥ 5 X

for p=1 = 2 ≥ 6 X

for p=2 ⇒ 4 ≥ 7 X

for p=3 ⇒ 8 ≥ 8 ✓ { Min. no. of parity bits req = 3 }

Now, after finding no. of parity bits req., & where they should be place in the message for error correction

m = 4 & p = 3 { total size = m+p = 7 }

Now, assign parity bits at the location 2^K {ie. $2^0, 2^1, 2^2, \ldots$} & the bits of 'm' at the remaining locations.

{ assuming m = 0101 }

ie.	1	2	3	4	5	6	7
	P_1	P_2	0	P_3	1	0	1

→ Now, we need to find the values of P_1, P_2 & P_3.

Date.....

⇒ At the recievers end check bits C_1, C_2, C_3 will be used to check error in all the possible bit locations of data (ie. 1 to 7) & 0 0 0 ~~$C_3 C_2 C_1$~~ ($C_3 C_2 C_1$) will mean no error.

P_3 P_2 P_1

	C_3	C_2	C_1	
no error ←	0	0	0	→ 0
error at 1 ←	0	0	1	→ 1
error at 2 ←	0	1	0	→ 2
	0	1	1	→ 3
	1	0	0	→ 4
	1	0	1	→ 5
	1	1	0	→ 6
	1	1	1	→ 7

⇒

1	2	3	4	5	6	7
P_1	P_2	0	P_3	1	0	1

⇒ data

$P_1 = 0$ { because, at 3, 5, & 7 even no. of 1's }

$P_2 = 1$ { because at 3, 6, & 7 odd no. of 1's }

$P_3 = 0$ { because at 5, 6, 7 even no. of 1's }

⇒

So, the data will be sent is :-

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

⇓

At Reciever ~~this will be sent~~

→ Now if data sent at reciever is 0100001 (ie. error at bit position 5)

⇒

***

$P_1 = (1, 3, 5, 7)$

$P_2 = (2, 3, 6, 7)$

$P_3 = (4, 5, 6, 7)$

⇒ ~~if P_1 is 1 ie. there might be an error at (1, 3, 5, 7) similarly for P_2 & P_3~~

P_1 takes care of Error at bit position (3, 5, 7). & Similarly for P_2 & P_3

$P_1 = P_3 \oplus 5 \oplus 7$ ⇒ take one leave one

$P_2 = 2 \oplus 3 \oplus 6 \oplus 7$ ⇒ take two leave two start at two

$P_3 = P_3 \oplus 5 \oplus 6 \oplus 7$ ⇒ take 4 leave 4 start at 4

Date.....

→ data at reciever is 0100001 :-

Now, it will ~~Calculate~~ find the error position & rectify the error.

C_3 C_2 C_1

1 0 1

(because odd no. of 1's at (1, 3, 5, 7))

(because odd no. of 1's at 4, 5, 6, 7)

(because ~~odd~~ even no. of 1's at (2, 3, 6, 7))

⇒ Now, C_3 C_2 C_1

1 0 1 ⇒ error at bit location '5'

⇒ Now, change the bit at location '5'

0100101 ⇒ error connected.

(35) Examples on hamming code :-

Ex :- If Message (m) = 11011, find the final data Sent to recievers including parity bit.

⇒ Let p is no. of parity bits

then,

$2^p \geq 5 + p + 1 \Rightarrow 2^p \geq 6 + p$.

so, $P_{min.} = 4$

1	2	3	4	5	6	7	8	9
P_1	P_2	1	P_3	1	0	1	P_4	1

~~$P_1 = P_1 \oplus P_3 \oplus P_5 \oplus P_7$~~ $P_1 = 1 \oplus 3 \oplus 5 \oplus 7 \oplus 9$

$\Rightarrow P_1 = 0$

Spiral

Date.....

⇒ ~~start~~ take 2, skip 2, start from 2

take 4, skip 4, start from 4

$P_2 = 2 \oplus 3 \oplus 6 \oplus 7$

$\Rightarrow P_2 \oplus 1 \oplus 0 \oplus 1$

$P_2 = 0$

$P_3 = 4 \oplus 5 \oplus 6 \oplus 7$

$\Rightarrow P_3 \oplus 1 \oplus 0 \oplus 1$

$P_3 = 0$

$P_4 = 8 \oplus 9$

$P_4 = P_4 \oplus 1 \Rightarrow 1$

So, data send to reciever (if no error) = 0010101

if data recieved is : 0010001 {ie. error at 5th locn.}

$C_1 = 1 \oplus 3 \oplus 5 \oplus 7 \oplus 9$

$0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1$

$C_2 = 2 \oplus 3 \oplus 6 \oplus 7$

$\Rightarrow 0 \oplus 1 \oplus 0 \oplus 1 \Rightarrow 0$

$C_3 = 4 \oplus 5 \oplus 6 \oplus 7$

$\Rightarrow 0 \oplus 0 \oplus 0 \oplus 1 = 1$

$C_4 = 8 \oplus 9$

$\Rightarrow 1 \oplus 1 = 0$

So, 0101 $\Rightarrow$ So error at bit locn 5

Date.....

(36) Floating point Conversions:-

(i) Binary to decimal:-

ex: 1101.01 (2^{-1}, 2^{-2})

⇒ (13.25)

(ii) decimal to binary:-

ex:- 13.625

⇒ 0.625 × 2 = 1.250

⇒ 1.250 × 2 ⇒ 1

⇒ 0.50 × 2 ⇒ 0

⇒ 1.00 ⇒ 1

1101.101

⇒ But for some decimal no. with Radix point it might not be possible to give exact binary representation.

ex:- 0.623

⇒ 0.10011... (Not a fixed representation)

0.623 × 2 = 1.246

0.246 × 2 = 0.492

0.492 × 2 = 0.984

0.984 × 2 = 1.968

0.968 × 2 = 1.936

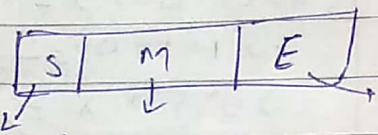
→ Real nos. can't be saved in a memory (fixed length) because they don't have fixed length. ex:- (total real nos. b/w 0 & 1 = ∞)

⇒ Thus close approximation is stored in the memory.

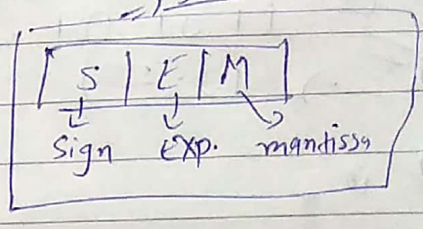
Date.....

(37) floating point representation-1 :-

→ ~~For~~ A floating point no. is represented by using 3 fields :-



or



ex:- Represent 100.1 ~~in~~ using floating point representation.

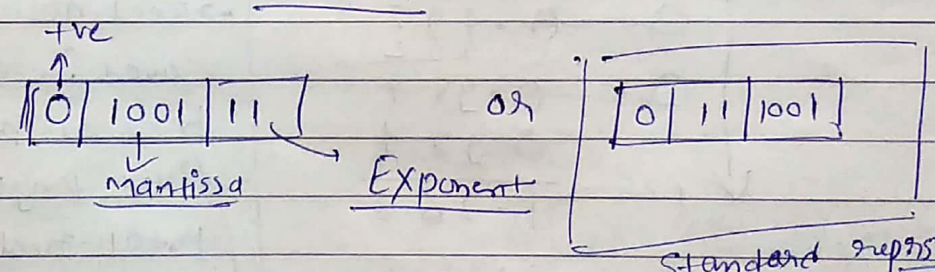
⇒ Step1 → Move the Radix point to the ~~left of~~ most significant 1. { also called ~~explicit / implicit~~ explicit normalisation }

Step2 :- ~~Use~~ Then multiply the no. by 2^k. where k = no. of bits Radix point comes to the left.

Step3 :- Now, fill the fields accordingly.

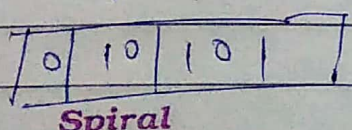
⇒ 100.1 ⇒ 0.1001×2^{3} → Exponent.

mantissa



ex:- .00101

⇒ $.101 \times 2^{-2}$ { 2^{-2} is multiplied because no. is reduced by moving Radix point to the right }



→ -ve represent Exponent in 2's Compl. form.

Date.....

→ There is one small problem ie. the exponent is -ve & it becomes difficult to compare -ve nos. so, a biasing is used by adding 2^{n-1} (if Exponent field is n-bit) to make the Exponent no. non-negetive.

→ 2^{n-1} is added because largest -ve no. with n-bits in 2's compl. is possible is (-2^{n-1}) & thus by adding 2^{n-1} the smallest no. will be '0' & largest $(2^n - 1)$

$= (2^{n-1} - 1 + 2^{n-1})$

→ Now to convert the no. represented to binary :-

$$(-1)^{\text{sign bit}} * 0.(M) * 2^{(E - \text{bias})}$$

(38) floating point representation :-

ex:- no. given is 100.111

⇒ $0.100111 * 2^3$ { Explicit normalisation ie. moving radix point just left of most significant '1' }

(Min. bits req. to save mantissa part is 6-bits)

⇒ $1.00111 * 2^2$ { Implicit normalisation. ie. moving radix point just Right to most significant '1' }

-> Normalisation (implicit or explicit) will help to represent a no. uniquely. ex: ~~10.001 = .10001 * 2^2 (using normalisation)~~
~~0010.001 -> .0010001 * 2^4~~

So, using Implicit normalisation some space (ie. 1 bit) in Mantissa part can be saved

S E (4) (only 4 bit is assigned given)

0	1010	00111
		M (5)

only 5 bit assigned for 'M' field

$2+(2^{4-1}) = 10 = E$

$M = 00111$

Converting it to binary => $(-1)^S * 1.M * 2^{(E-bias)}$ ***

not '0' — for implicit Normalisation

*** -> Normalisation (implicit or explicit) helps to represent a no. uniquely. (otherwise for a no. many representations can be possible)

ex:- $010.001 \Rightarrow .10001 \times 2^2$ {using explicit normⁿ}

but

$10.001 = 0010.001$ {adding 0's to MSB will not change the no.}

w/o. normalisation

$.0010001 \times 2^4$

Same no. diff. ~~representation~~ representation

*** -> biasing is done to eliminate the chances of Exponent field to be -ve.

→ By default implicit normalisation is used ***

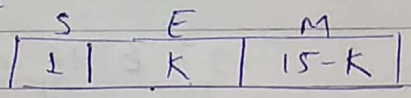
Date.....

(39) floating point representation example :-

(a) Consider a 16-bit reg. of the following format is used to store a floating point no. Mantissa(M) as normalised signed magnitude fraction, Exponent(E) is expressed in Excess-64 form, Base of the system is '2'

(i) How many bits are allocated for fractional mantissa.

=>



Excess-64 => ie. biasing is 64 ie. 2^6

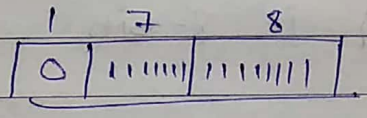
So, E-field is 7 bits

$K = 7$

So, $\boxed{M = 8}$

(ii) What is the value of largest no. that can be represented in base 10.

=>



$\Rightarrow (-1)^S * 1.M * 2^{E-64}$

=> for largest no: S=0 & (M & E) has to as large as possible

$1.11111111 * 2^{127-63}$

$\Rightarrow \left(1 + \underbrace{0.11111111}_{\downarrow}\right) * 2^{63}$

$2^{-1} + 2^{-2} + 2^{-3} \dots 2^{-8} \Rightarrow \left(\frac{a(1-r^n)}{1-r}\right)$ = GP

$\Rightarrow \frac{1}{2}\left(1-\frac{1}{2^8}\right)$

$\left(1-\frac{1}{2}\right)$

$\Rightarrow \left(1-\frac{1}{2^8}\right)$

$\left(1+\left(1-\frac{1}{2^8}\right)\right) * 2^{63}$

$\Rightarrow \left(2^{64} - 2^{55}\right)$ => largest decimal nos.

Spiral

Date.....

*** ⇒ By increasing no. of bits in exponent field the range can be increased (ie. larger no. can be represented)

*** ⇒ By increasing no. of bits in Mantissa field the precision can be increased.

(iii) What is the 16-bit pattern for $(-7.5)_{10}$

⇒ -ve so S = 1

S	E(7)	M(8)
1	1000010	11100000

→ 5 0's are padded

~~7 = 00000111~~

$(7.5)_{10} = (111.1)_2$

$(1.111 * 10^2)$ {implicit normalisation}

$E = 2 + 64 = 66 \Rightarrow$ 1000010

M = 11100000 {5 0's are padded to make it 8-bits}

(40) Example - 2 :-

(8) Consider a 16-bit registes of the following format is used to store a floating point no. Mantissa (M) is denoted as normalised signed magnitude fraction, Exponent (E) is expressed in Excess-64 form. base of the system is '2'.

(i) what is the diff. b/w 1st smallest +ve no. & 2nd smallest +ve no.

Date.....

⇒ All 0's will represent smallest +ve no.

S	E	M
0	0000 — 000 — 0	

⇒ Even All 0's in Mantissa part will not represent 0 in case of Implicit normalisation

$(-1)^S * (1.0M) * 2^{E-64}$ ⇒ $1.0 \times 2^{-64} = 2^{-64}$ ⇒ Smallest +ve no.

2nd smallest +ve no.:- put 1 at LSB of Mantissa.

$(-1)^0 * (1.00000001) * 2^{-64}$ ⇒

⇒ difference = $(0.00000001) * 2^{-64}$ ($0.00000001 \to 2^{-8}$)

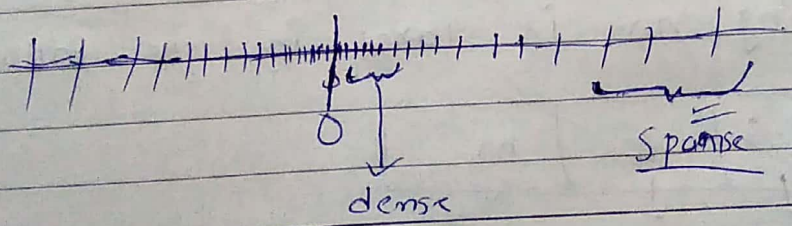
⇒ 2^{-72}

And, Diff. b/w 1st highest & 2nd highest no. is:-

$(1.11111111 * 2^{63}) - (1.11111110 * 2^{63})$

⇒ 2^{55}

⇒ So, it can be said that the representation is dense close to '0' & sparse as we move away ~~to~~ ~~from~~ '0'



Date.....

(ii) What is the pattern which denotes the value $(-16.125)_{10}$

⇒

1	1000 0000	0000 0010
5	7	8

→ padded

$-(1000.001)_2 \Rightarrow 1.0000001 * 2^4$

$E = 64 + 4 = 68 = 1000100$

$M = 00000010$

(42) Intro. to Floating point representation:-

→ The floating point nos. are stored in mantissa (M) & Exponent (E) form.

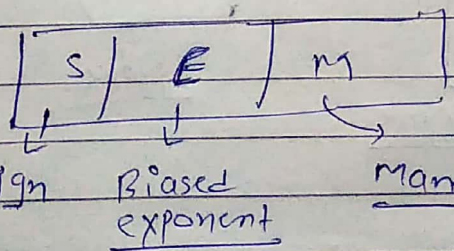
→ Most of the notations represent mantissa as normalised sign magnitude fraction.

→ The normalisation can be explicit or Implicit. (degree #)

→ The exponent is denoted in biased form.
(The biased exponent is an unsigned no., which can represent signed exponent of original no.)

True exponent = Biased exponent − Bias

→ The floating no. is stored in the following way,



Date.....

→ If E field is K-bits then Bias is $2^{(K-1)}$

& biased exponent will be { 0 to $2^{(K)}-1$ }
Unsigned

(43.) Introduction to IEEE standards :-

In 1985 :- Single, double precision } → in gate

In 2008 :- Half, Quadruple precision

1985 standard :-

→ The base of the system is 2.

→ Two kinds of precision. { single → 32-bits, double → 64-bits }

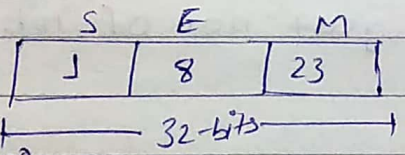
→ The floating point no. can be represented as.

(a) Implicit normalised form ⇒ (1.mantissa)

(b) fractional form ⇒ { .011, 0.110 - - }

→ Certain Mantissa & exponent Comb^n does not represent any no. ((Not a no. → NAN)

(44) single - precision :-



{ Excess - 127 } (Not 128 because all 1's are used for representing ±∞)

some special nos :-

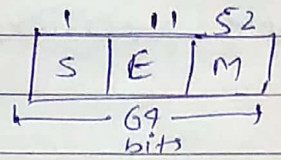
	S(1)	E(8)	M(23)	Value
(1)	(0 or 1)	000 - - 0 (E=0)	00 - - 0 (M=0)	±0
(2)	0 (1)	1111 - - 1 (E=255)	(000-0) M=0	±∞
(3)	(0 or 1)	(1 ≤ E ≤ 254)	M = xx - - - x	Implicit normalised form
(4)	0 or 1	E = 0	M ≠ 0	fractional form → $(-1)^S(0.M)*2^{-126}$
(5)	0 or 1	E = 255	M ≠ 0	NAN (Not A No.)

Date.....

When E=0, it's either ~~±0 or~~ represents ±0 or a fractional form (depending on mantissa)

When E=255, it's either represents ±∞ or a NAN (ie. Not a no.) {depending on Mantissa}

(45) Double precision:-



EXcess = ~~2047~~ 1023

S(1)	E(11)	M (52)	value.
(0 or 1)	E=0 {00—0}	M=0 {00—0}	±0
(0 or 1)	E=~~2047~~ {1111—1}	M=0 {00—0}	±∞
(0 or 1)	1≤E≤2046	M=xxx—x	Implicit normalised ~~form~~. $(-1)^S \times (1.m) \times 2^{E-1023}$
(0 or 1)	E=0 {00—0}	M≠0	fractional form $(-1)^S (0.M) \times 2^{-1022}$
(0 or 1)	E=2047	M≠0	NAN

-> same as single precision, just no. of bits ↑ to 64.

(46) -> Memorise / derive the table & 1st figure out the cases before solving the question.

Date.....

(46) Example 1 on single precision:-

(Q) Consider the 32-bit reg. which stores floating point nos. in IEEE single precision format.

(i) What is the value of the no., if 32-bits are given below.

1	(8)	(23)
0	1000 0011	1100 — 0

⇒ It's not in fractional form because $E \neq 0$, it is in Implicit normalised form.

⇒ $(-1)^0 * (1.1100--0) * 2^{(131-127)}$

⇒ $(1.1100--0) * 2^4$ ⇒ $(11100.0--)_2$

⇒ $(28.0)_{10}$

(ii) What is the value of the no., if 32-bits are 0 1111 1111 1100 0--01

⇒ $S = 0$ or 1; $E = 255$ {all 1's}; $M \neq 0$ ⇒ Not a No. (NAN)

(47) Example 1 on double precision:-

(Q) Consider a 64-bit reg. which stores floating point nos. in IEEE double precision format.

(i) what is the value of no., if 64 bits are:-

1	10000000011	1101100 — 0

Date.....

$\Rightarrow$ $S = 1$, $E = 1027$ $\wedge$ $M \neq 0$ {so, it's implicit normalised form}

$(-1)^1 * (1.110110--0) * 2^{1027-1023}$

$\Rightarrow$ $-(1.110110--0) * 2^4 \Rightarrow -(11101.10-)_2$

$\Downarrow$

$-(29.5)_{10}$

=

(50) Example on single & doubl

(i) what is the value of no. if 64 bits are.

0111 1111 1111 0000 0000 ---

$\Rightarrow$ $S = 0$,

$E =$ 2047 (all 1's) & $M = 0$ {all 0's}

$\Downarrow$

(not is $+\infty$)

=

(50) Example on single & double precision:-

(Q) Represent the no. $(117)_{10}$ in IEEE 754 singl & double precision format.

$\Rightarrow$ Single precision:-

$S = 0$, $(1110101)_2 = 117$

$(1.1101001) * 2^6$

=

so, $E = 7 + 127 \Rightarrow 134$ & $M = 11010010---0$

Spiral

Date.....

S (1)	E (8)	M (23)
0	10000100	10101000---0

Double precision:-

S (1)	E (11)	M (52)
0	10000000101	1101010---0

$E = 6 + 1023$

$\Rightarrow 1029$

(51) gate 2016 on no. of integer:-

(Q) let x be the no. of distinct 16-bit integers in 2's compl. representation. let y be the no. of distinct 16-bit integers represented in sign-magnitude representation. Then x-y is —.

$\Rightarrow x = -2^{15}$ to $(2^{15}-1)$ $\Rightarrow$ one no. more

$y = -2^{15}-1$ to $2^{15}-1$

So, $x-y = 1$

(58) EXample on fractional form:-

(Q) What is the range of floating point nos. represented using the fractional form of IEEE 754 single precision ~~method~~ format?

1	8	23
S	E	M

$\Rightarrow$ for fractional form:-

(Condn:- $E=0$ & $M \neq 0$)

fractional form $= (-1)^S * (0.M) * 2^{-126}$

Date.....

⇒ mean't be 0'

→ Only 'm' can be changed: every thing else is fixed & S = 0 or 1

Smallest +ve fractional ~~form~~ no:-

$(0.000-1) * 2^{-126}$

$\Rightarrow \boxed{2^{-149}}$

Largest +ve fractional no:-

$(0.111-1) * 2^{-126}$

$\left(1 - \frac{1}{2^{23}}\right) * 2^{-126} \Rightarrow 2^{-126} - 2^{-149}$

+ve range is $\Rightarrow \left(2^{-149}\right)$ to $\left(2^{-126} - 2^{-149}\right)$

-ve range is $\Rightarrow \left(-2^{-149} \text{ to } -(2^{-126} - 2^{-149})\right)$

(59) Example on fractional & implicit form:-

(Q) What is the value of A-B? where A is smallest number representing implicit normalised form, B is largest no. representable using fractional form. (A & B +ve nos. & use single precision format)

⇒ $A = (-1)^S * (1.00-0) * 2^{1-127}$

$\Rightarrow 2^{-126}$

S	E	M
0	000-1	0000

(min.) {Implicit}

$B = (-1)^S * (0.111-1) * 2^{-126}$

$\Rightarrow \left(1 - \frac{1}{2^{23}}\right) * 2^{-126}$

0	00-0	1111-1

(max.) {fractional}

$\therefore$ Q' $A - B = 2^{-126}\left(1 - 1 + \frac{1}{2^{23}}\right) \Rightarrow \boxed{2^{-149}}$

Spiral

Date.....

(52) Binary multiplication (partial sum method)

↳ same as done on pen and paper

ex:- (5) 0 1 0 1 (multiplicand)

(6) 0 1 1 0 (multiplier)

```
  0 0 0 0
0 1 0 1
0 1 0 1
0 0 0 0
-------
0 0 1 1 1 1 0
```

⇒ (partial sum method)

(30) 0 0 1 1 1 1 0

⇒ In worst case no. of bit req. to store the result of two n-bit multiplication is (2n).

→ In this method the no. of addn req = no. of 1's in multiplier

& No. of shifts = no. of bits in multiplier.

⇒ This can be reduced by using booth's algo.

(53) Idea behind Booth's algo:-

→ The main idea of booth's algo is that the diff. b/w value of a no. containing run of 1's is the diff. of two nos. (which are power of '2').

ex: $2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$0\ 1\ 1\ 1\ 1\ 0 = (2^5 - 2^1)$

$\Rightarrow 30$

ex: $0\ 1\ 1\ 0 \Rightarrow 2^3 - 2^1 \Rightarrow 6$

Spiral

Date.....

NOW, Suppose a Multiplicand (M) is multiplied by a no. 011110 (multiplier) then

M * 011110 => By partial sum method
no. of shift = 6 & Add^n = 4

But, using booth's algo:-

M * 011110 => M x ($2^5 - 2^1$)
=> $2^5 * M + 2^1 * (-M)$
(2^5 left shifts) + 1 left shift
(1 add^n)

So, 6 shifts & only 1 add^n are required.

***
(59) Booth's algo example :-

ex:- 01011 → M
* 01110 → Q

By using partial sum => 5 shifts & 3 add^n

$Q = (2^4 - 2^1)$
So, $2^4(M) + 2^1(-M)$ { -M = 10101 {2's compl.} }

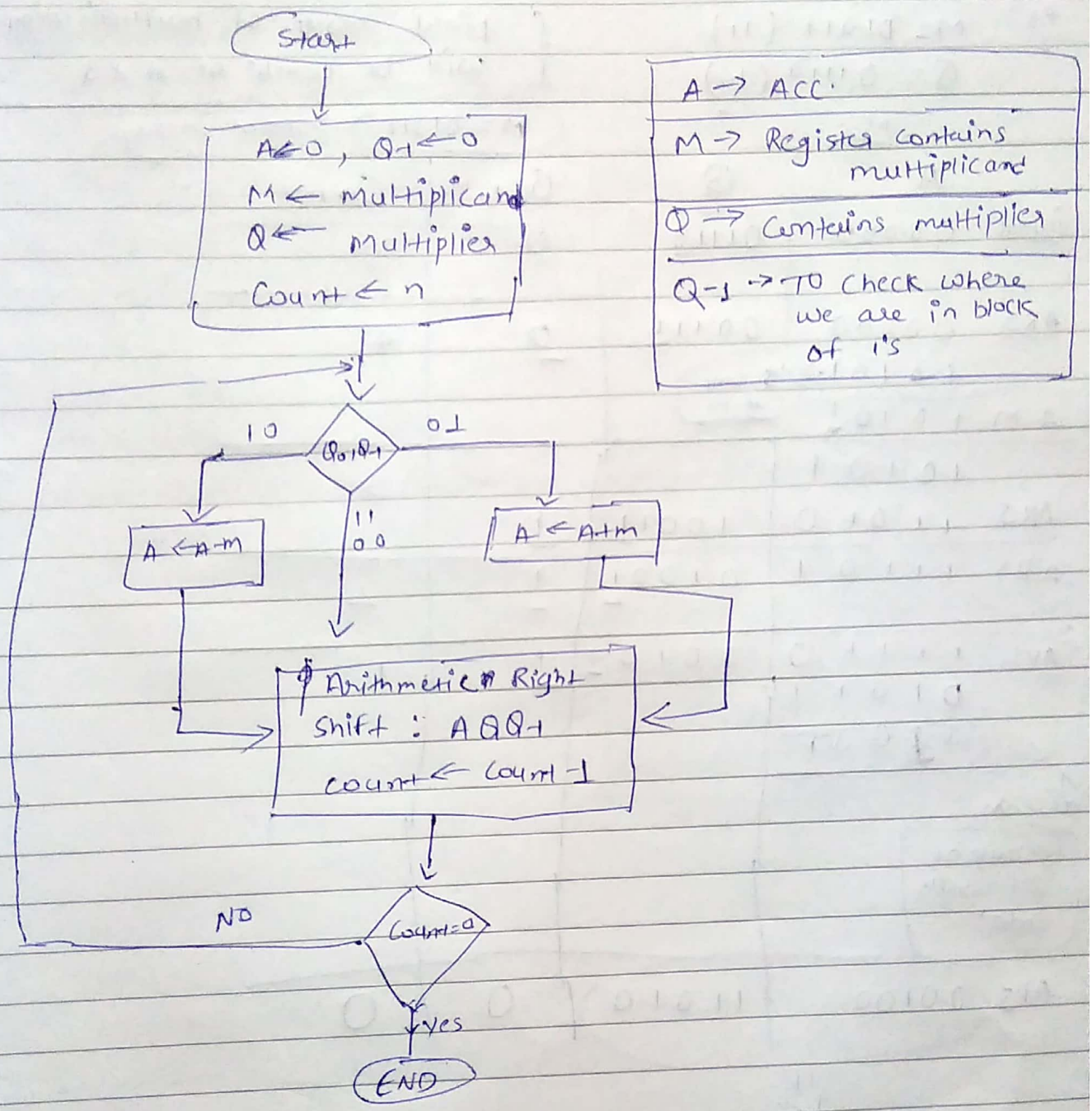
01011 00000
(4 - left shifts for m)

01010 {Left shift 1 time}
{MSB is 1, because no is -ve}

NOW
0101100000
+ 1111101010
Sign bit extension
10100110 10
Spiral because discard it in 2's compl. addition
154

{ total 5 shifts & 1 add^n & 1 subtn {for (-M)} }

(56) Booth's algorithm operation flow:- Date.....



Date.....

(55) Booth's algorithm:-

ex:- M = 01011 (11)
Q = 01110 (14)

{ final result of multiplication will be comb^n of A & Q

M = 01011 => stored in reg.

	A (Acc.)	Q	Q-1	Count
initial	00000	01110	0	5
ARS	00000	00111	0	4
	10101 => 2's comp. of m			
A-M	10101			
	10101			3
ARS	11010	10011	1	
ARS	11101	01001	1	2
ARS	11110	10100	1	1
	01011			
	(1)01001			
	← discard because of 2's compl. adder			
ARS	00100	11010	0	0

Result

When :-

Q_0 Q_{-1}	
0 0	→ ARS
1 1	→ ARS
1 0	→ A − M
0 1	→ A + M

***

Date.....

(57.) Advantages & disadv. of booth's algo:-

Adv:-

(i) In general, no. of operations required will be reduced.

(ii) It can do signed multiplications.

(iii) If -ve nos. are involved it gives better performance.

ex: 1111111111111011 = -5

Disadv:-

→ When the blocks contains only single ones.

ex:- 0101010101010101

every block has one '1' & for each block it requires 1 add^n & 1 subtr^n Compared to only one add^n in partial sum method.