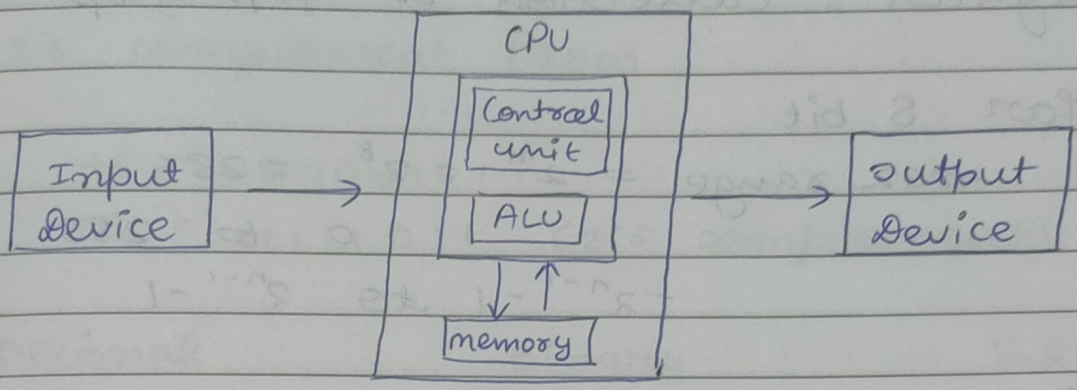
* Von - Neumann Architecture :-



→ The program and operand are at same place in memory - USP of Von-neumann.

→ Control unit :-

The control unit defines the order of execution of instruction.

- It also generates the number of timming signals (clock pulses) that are synchronized with the execution of every microoperation

Accumulator : Type of register for short-term intermediate storage of arithmetic and logic data in computer CPU.

* Signed Number representation :-

(i) Sign magnitude
(ii) 1's complement
(iii) 2's complement.

Signed → having negative sign.
unsigned → positive numbers.
MSB → 0 for +ve
1 for -ve.

M T W T F S S
Page No.:
Date:
YOUVA

(i) Sign magnitude :

FLIP-FLOP → Stores 1 bit.

Register : collection of flip flop.

for 8 bit

max range $= 2^n = 2^8 = 256$

0 to 255.

$-2^{n-1} - 1$ to $2^{n-1} - 1$

Decimal	Binary	Sign-magnitude
0	00000000	0
1	00000001	1
⋮	⋮	⋮
128	10000000	-0
129	10000001	-1
⋮	⋮	⋮
255	11111111	-127

range → -127 to 127.

and also includes -0.

(ii) 1's complement representation.

-5 → 11111010 +5 → 00000101

Decimal	Binary	1's complement
0	00000000	+0
1	00000001	+1
⋮	⋮	⋮
128	10000000	-127
⋮	⋮	⋮
255	11111111	-0

Range $= -2^{n-1} - 1$ to $2^{n-1} - 1$.

(iii) 2's complement form

+5 = 00000101

-5 = 11111011 (2's complement)

Decimal	Binary	2's complement
0	00000000	0
1	00000001	1
⋮	⋮	
127	01111111	127
128	10000000	-128
⋮	⋮	⋮
255	11111111	-1

Range $= -2^{n-1}$ to $2^{n-1} - 1$

☆. Signed number addition and subtraction (2's complement):

→ Binary addition:

(+5) + (+7)

(+5) + (-7)

(-5) + (+7)

(-5) + (-7)

Step-1) Determine the binary of the given numbers.

Step-2) Perform the addition if their is carry out then discard it.

Step-3) If the result is negative then determine the 2's complement of the result except the sign-bit. (MSB).

eg. (+5) + (+7)

101 + 111

= 00001100 = 12.

→ Binary subtraction:

Step-1) Determine the binary of given 2 numbers.

step-2) Take 2's complement of subtraend and add it to the minuend.

Now apply rules of binary addition.

eg. (+5) - (+7)

+7 = 00000111

-7 = 11111001

+5 = 00000101

11111110 → (-2)

↳ -(00000010)

→ Checking overflow condition:

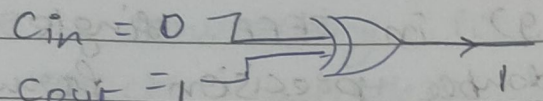
+6 → 110

+5 → 101

1011

↙ Cout Cin = 0.

(carry inside)

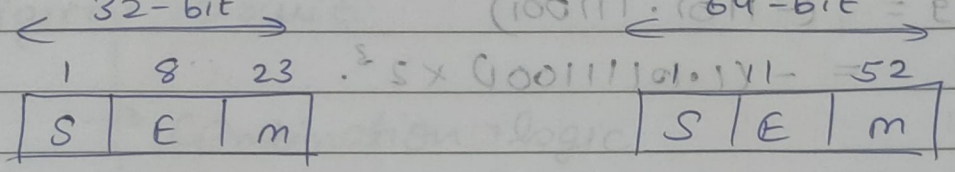


Hence it is condition of overflow.

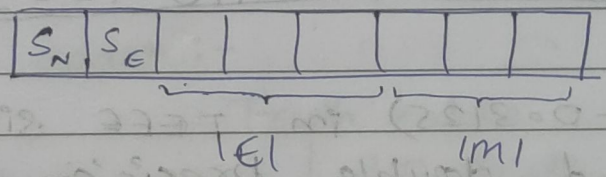
XOR gate is used to check this overflow condition.

★ Floating point operations :-

① IEEE 32-bit format (Single precision format)
② IEEE 64-bit format (Double precision format)



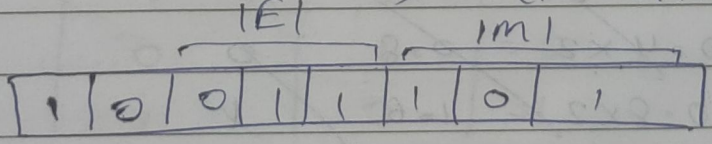
→ 8 bit floating point representation with unbiased exponent :-

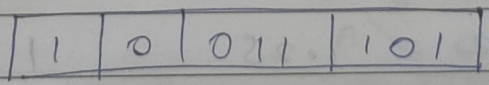


$(-13.9) = -(1101.1\overline{1001})$

→ Normalised Mantissa –

$\Rightarrow -(1.101111001) \times 2^3$





Note: From unbiased to biased exponent
i) in single precision – +127.
ii) in double precision – +1023.

→ By default we consider unbiased exponent

Q. Represent (-13.9) in IEEE single precision and double precision format.

→ $-13.9 = -(1101.111001)$

$= -(1.101111001) \times 2^3$

unbiased exp = 3

∴ biased exp = 3 + 127 = 130.

1	10000010	101.11---

→ $-13.9 = (-1101.111001)$

$= -(1.101111001) \times 2^3$.

∴ biased exp = 3 + 1023

= 1026.

I	10000000010	10111001---

Q. Represent (-0.3125) in IEEE single precision and double precision format.

0.3 × 2 = 0.6 — 0

0.6 × 2 = 1.2 — 1

0.2 × 2 = 0.4 — 0

0.4 × 2 = 0.8 — 0

0.8 × 2 = 1.6 — 1

0.3125 × 2 = 0.625 — 0

0.625 × 2 = 1.25 — 1

0.25 × 2 = 0.5 — 0

0.5 × 2 = 1.0 — 1

$(-)(0.0101) = -(1.010--) \times 10^{-2}$

biased range = -2 + 127 = 125

→ Before converting to the biased exponent first check whether unbiased exponent is in range or not.

1	01111101	0100000--

☆ Combinational logic :-

→ Digital circuit

- → Combination logic
- → Sequential logic.

(i) Combinational logic :-

a) Half adder :-

Truth table.

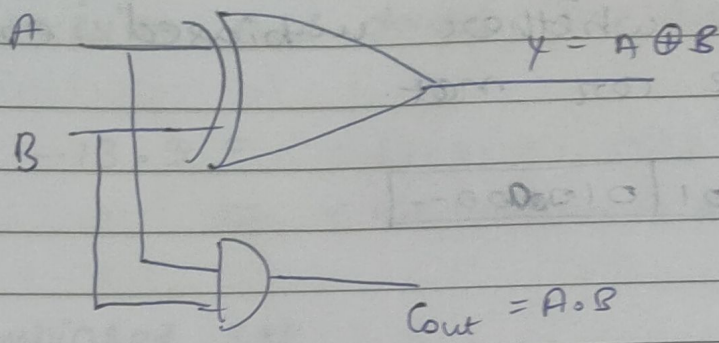
Inputs		Outputs	
A	B	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A \ B	0	1
0	0	1
1	1	0

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

A \ B	0	1
0	0	0
1	0	1

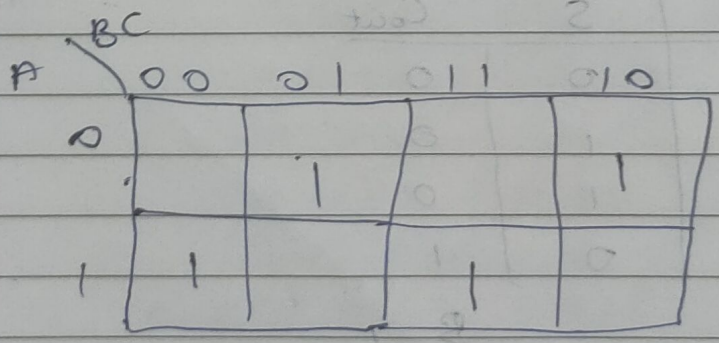
$C_{out} = AB$



b) Full adder :-

Truth table

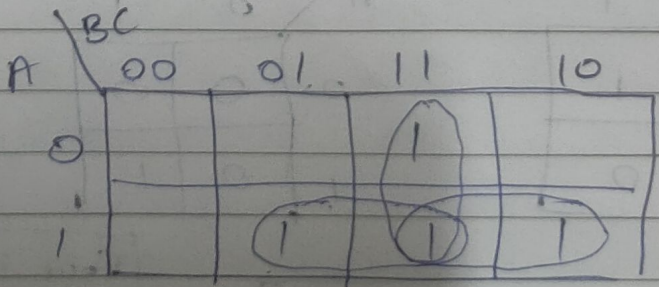
Inputs			Outputs	
A	B	C	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
1	0	0	1	0
0	1	1	0	1
1	1	0	0	1
1	1	1	1	1
1	0	1	0	1



Sum

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

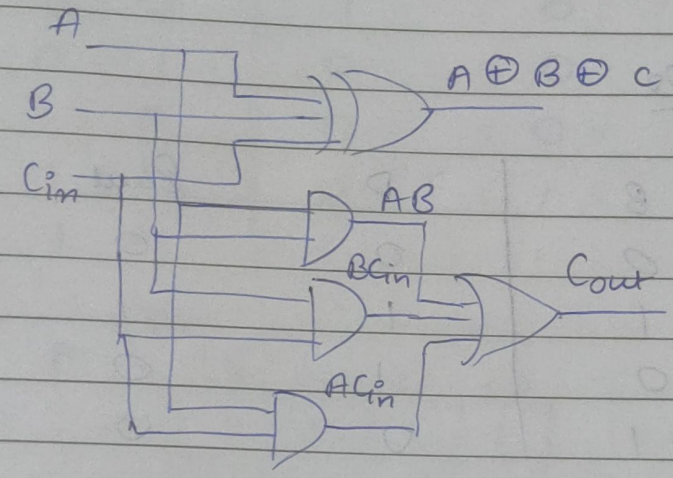
$= A \oplus B \oplus C.$



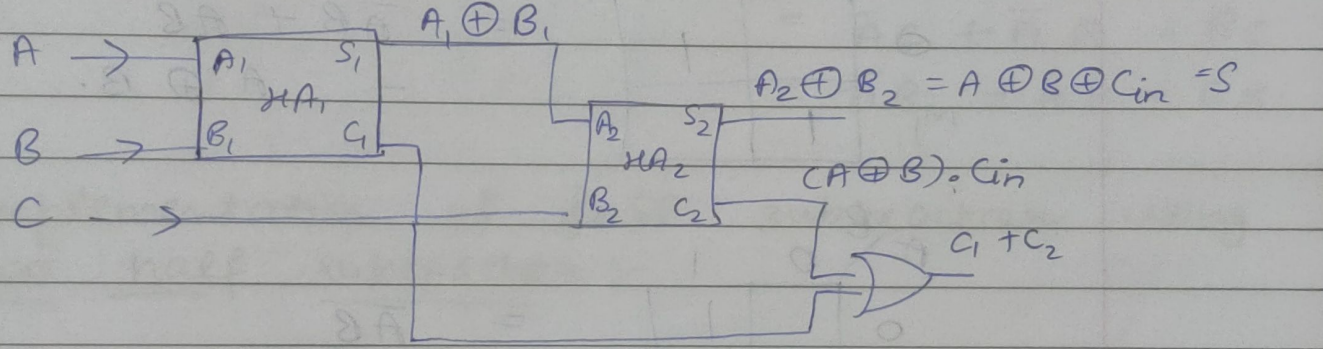
Cout.

$= BC + AC + AB$

- diagram :



- Implementation of full adder using two half adder :



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

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

$= AB + \bar{A}BC_{in} + A\bar{B}C_{in}$

$= A(B + \bar{B}C_{in}) + \bar{A}BC_{in}$

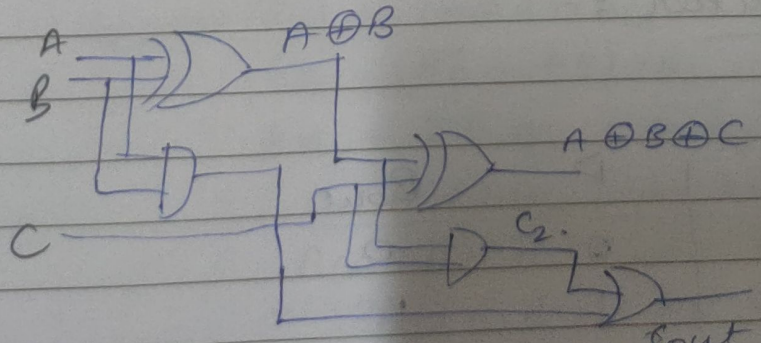
$= AB + AC_{in} + \bar{A}BC_{in}$

$= B(A + \bar{A}C_{in}) + AC_{in}$

$= B(A+\bar{A})(A+C_{in}) + AC_{in}$

$C_{out} = AB + BC_{in} + AC_{in}$

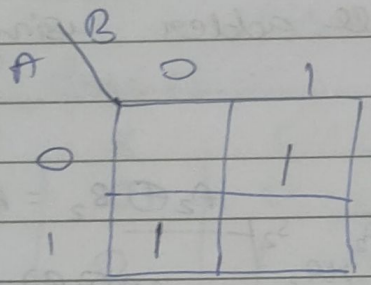
$A + \bar{A}C_{in} \rightarrow \underbrace{(A+\bar{A})}_{1}(A+C_{in})$



☆. Half Subtractor :-

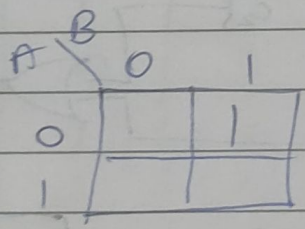
→ Truth table

A	B	D	Bout
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

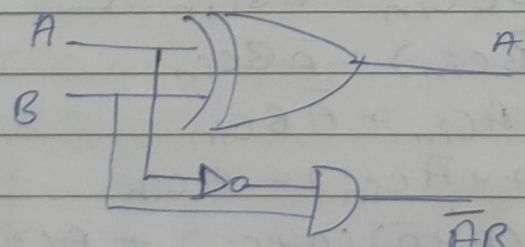


$= \bar{A}B + A\bar{B}$

$= A \oplus B.$



$= \bar{A}B$



☆. Full Subtractor :-

→ Truth table

A	B	C	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1

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

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

$= A \oplus B \oplus C$

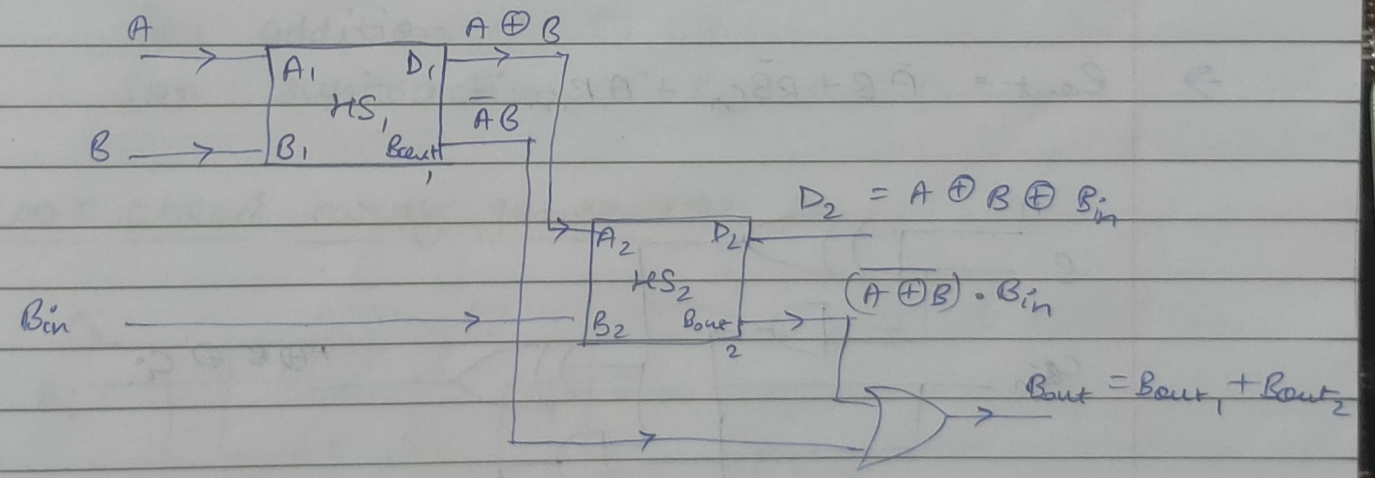
$(C = B_{in})$

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

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

$= \bar{A}B_{in} + \bar{A}B + BB_{in}$

→ Implementation of full subtractor using two half subtractor :-



$B_{out} = \bar{A}B + (\overline{A \oplus B}) \cdot B_{in}$

$= \bar{A}B + (\bar{A}\bar{B} + AB) \cdot B_{in}$

$= \bar{A}B + \bar{A}\bar{B} \cdot B_{in} + ABB_{in}$

$= B(\bar{A} + AB_{in}) + \bar{A}\bar{B} \cdot B_{in}$

$= B(A + \bar{A})(\bar{A} + B_{in}) + \bar{A}\bar{B} \cdot B_{in}$

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

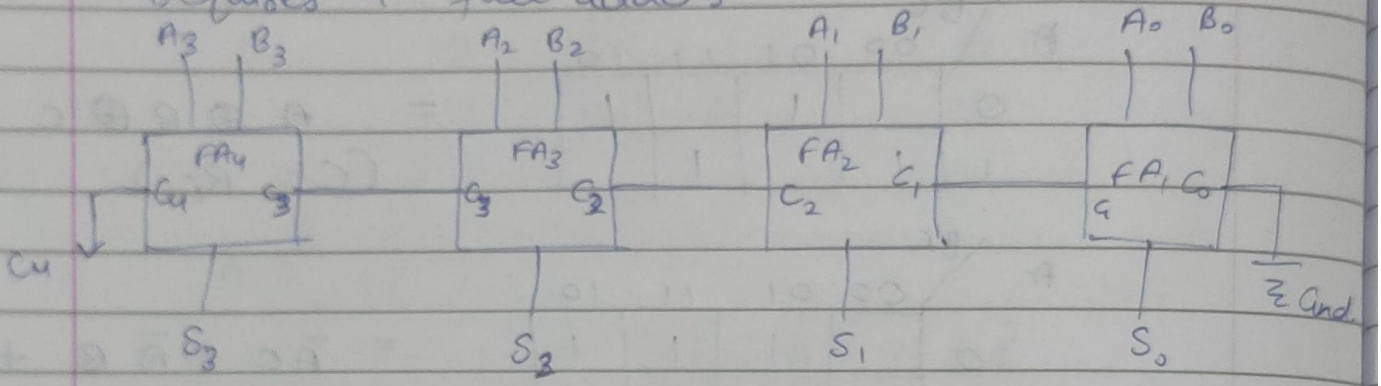
$= \bar{A}(B + \bar{B} \cdot B_{in}) + B \cdot B_{in} = \bar{A}(B + \bar{B})(B + B_{in}) + BB_{in}$

$B_{out} = \bar{A}B + \bar{A}B_{in} + BB_{in}$

☆. 4-bit parallel binary adder :-

		C_3	C_2	C_1	C_0
A	=	A_3	A_2	A_1	A_0
B	=	B_3	B_2	B_1	B_0
		S_3	S_2	S_1	S_0

requires 4- full adders.

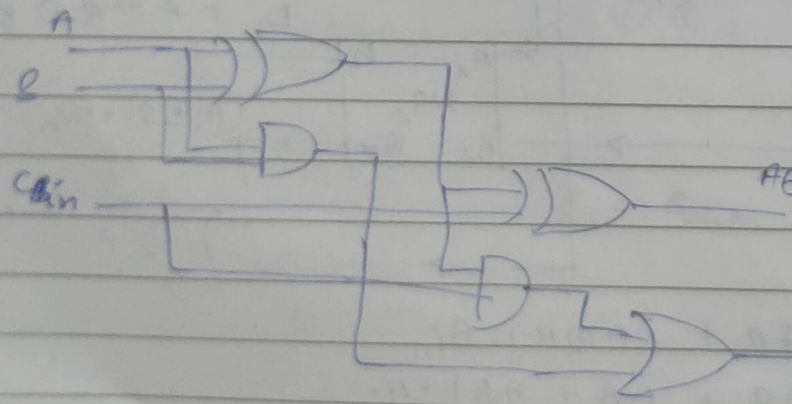


elapsed time = $n \times t_p$

↓

carry propagation time.

→ $B_{out} = \bar{A}B + B\bar{B}_{in} + AB_{in}$



→ '2n' = gate level delay

4 bit = 8

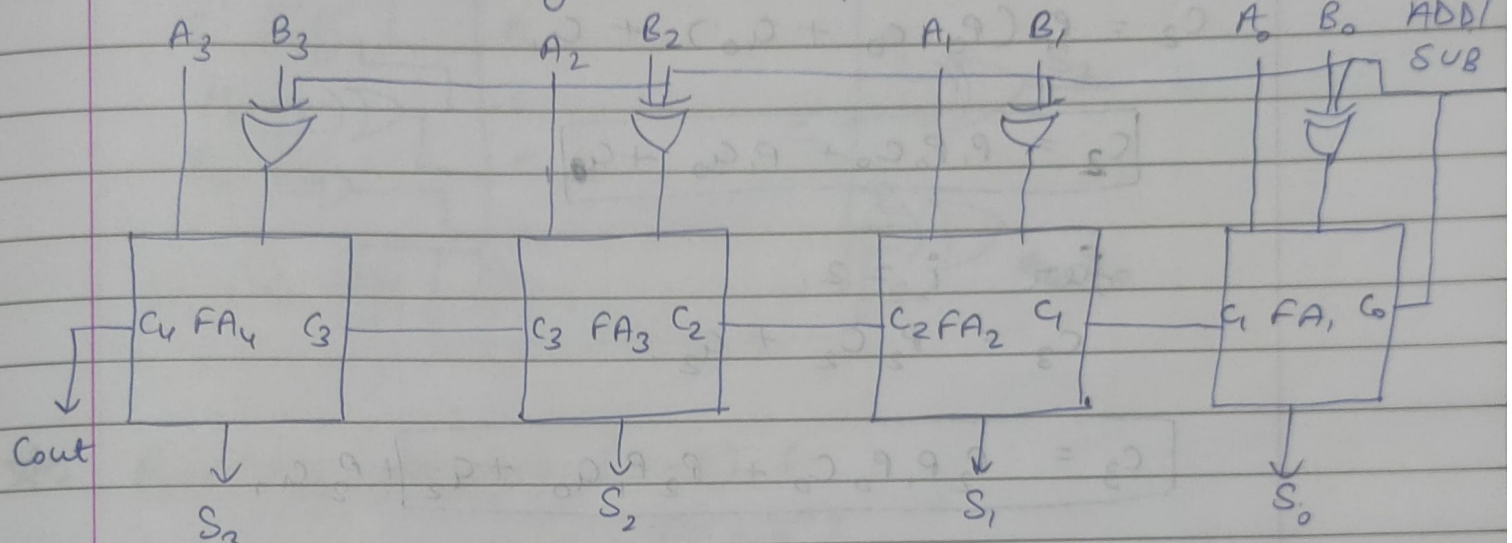
10 bit = 20 gate level delay

☆. 4- bit parallel binary adder and subtractor

$$\begin{array}{rcccc} & C_{+3} & C_2 & C_1 & C_0 \\ A = & A_3 & A_2 & A_1 & A_0 \\ +(-B) = & \bar{B}_3 & \bar{B}_2 & \bar{B}_1 & \bar{B}_0 \\ \hline & S_3 & S_2 & S_1 & S_0 \end{array}$$

$-B = \bar{B}_3\ \bar{B}_2\ \bar{B}_1\ \bar{B}_0 + 1$

Taking 2's complement.

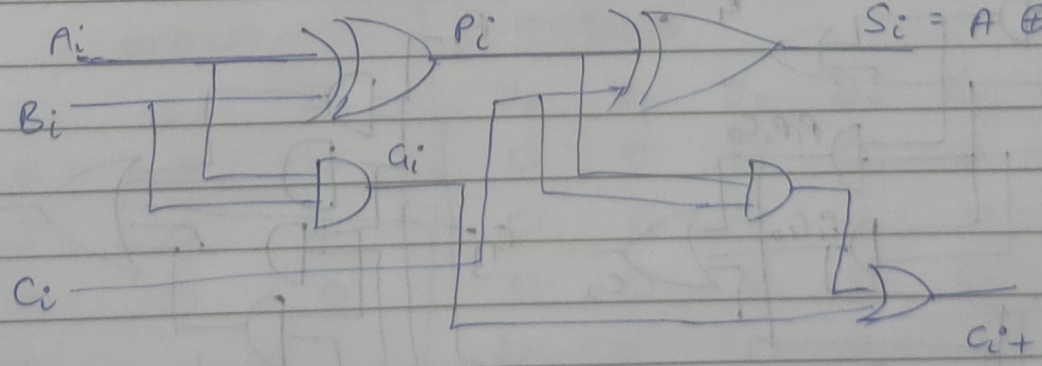


for addition → $C_0 = 0$

for subtraction → $C_0 = 1$.

☆. Look ahead carry generator :-

→



$P_i = A_i \oplus B_i$

$G_i = A_i \cdot B_i$

$S_i = P_i \oplus C_i$

$C_{i+1} = P_i \cdot C_1 + G_i$ — (4)

→ from eqⁿ ④

for $i = 0$.

$$\boxed{c_1 = p_0 \cdot c_0 + g_0}$$

for $i = 1$

$$\boxed{c_2 = p_1 \cdot c_1 + g_1}$$

$$c_2 = p_1 (p_0 \cdot c_0 + g_0) + g_1$$

$$\boxed{c_2 = p_1 p_0 c_0 + p_1 g_0 + g_1}$$

for $i = 2$.

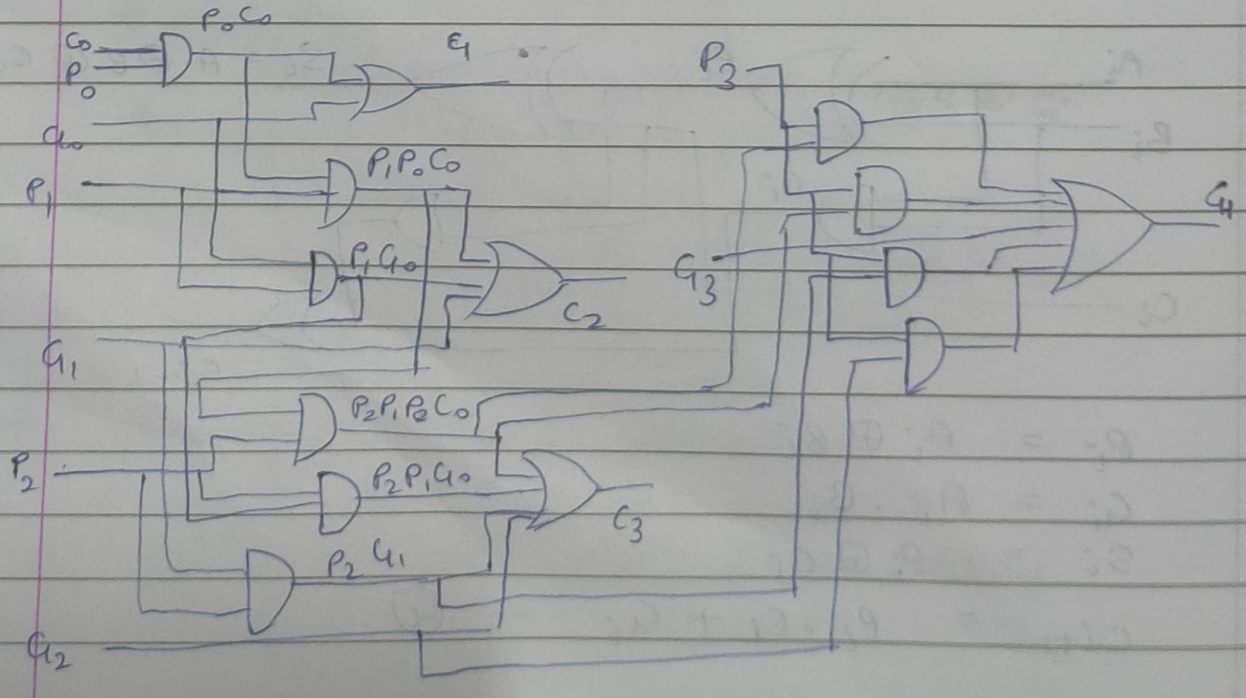
$$c_3 = p_2 c_2 + g_2$$

$$\boxed{c_3 = p_2 p_1 p_0 c_0 + p_2 p_1 g_0 + g_2 + p_2 g_1}$$

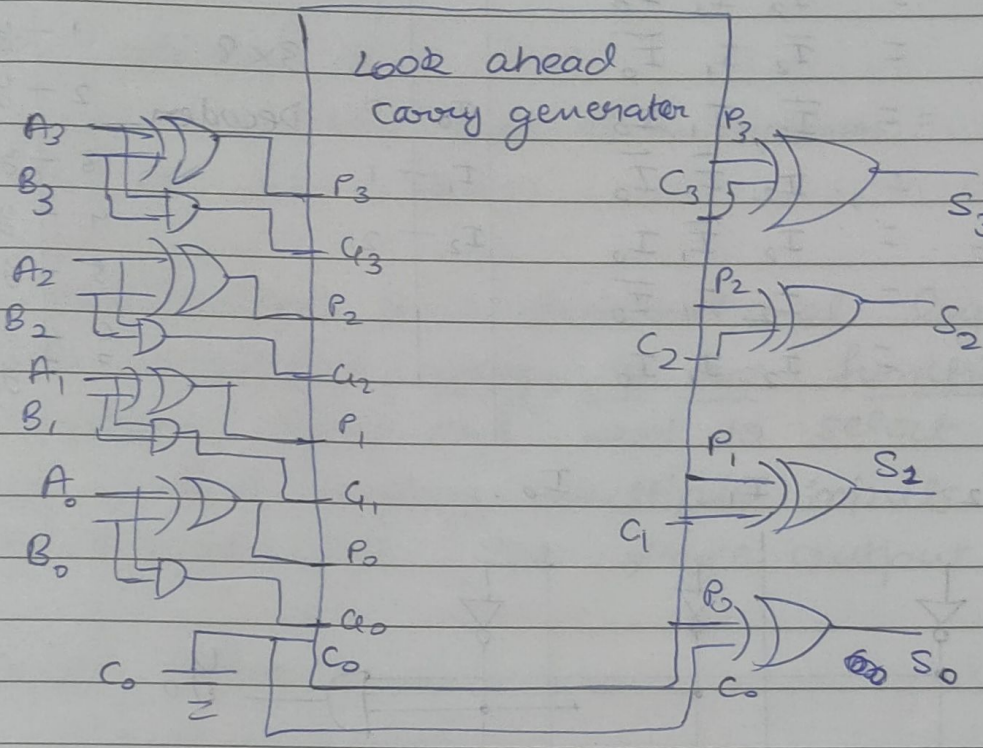
for $i = 3$

$$\boxed{c_4 = p_3 p_2 p_1 p_0 c_0 + p_3 p_2 p_1 g_0 + g_3} +$$

$$p_3 p_2 g_1 + p_3 g_2$$



☆. Look ahead carry binary adder :-



☆. Binary decoder (3×8) :-

$(n \times 2^n)$

⇒ It is a combinational logic circuit that converts binary input code to one of the possible outputs. (decoder is used create minterm)

⇒ Fouth table

Inputs			Outputs.							
I_2	I_1	I_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

$y_0 = \bar{I}_2 \bar{I}_1 \bar{I}_0$

$y_1 = \bar{I}_2 \bar{I}_1 I_0$

$y_2 = \bar{I}_2 I_1 \bar{I}_0$

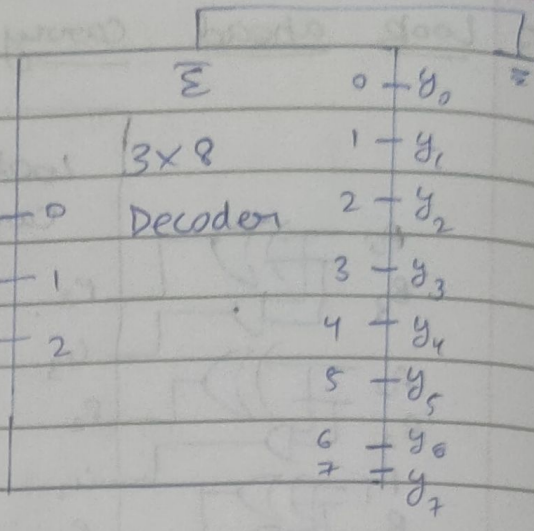
$y_3 = \bar{I}_2 I_1 I_0$

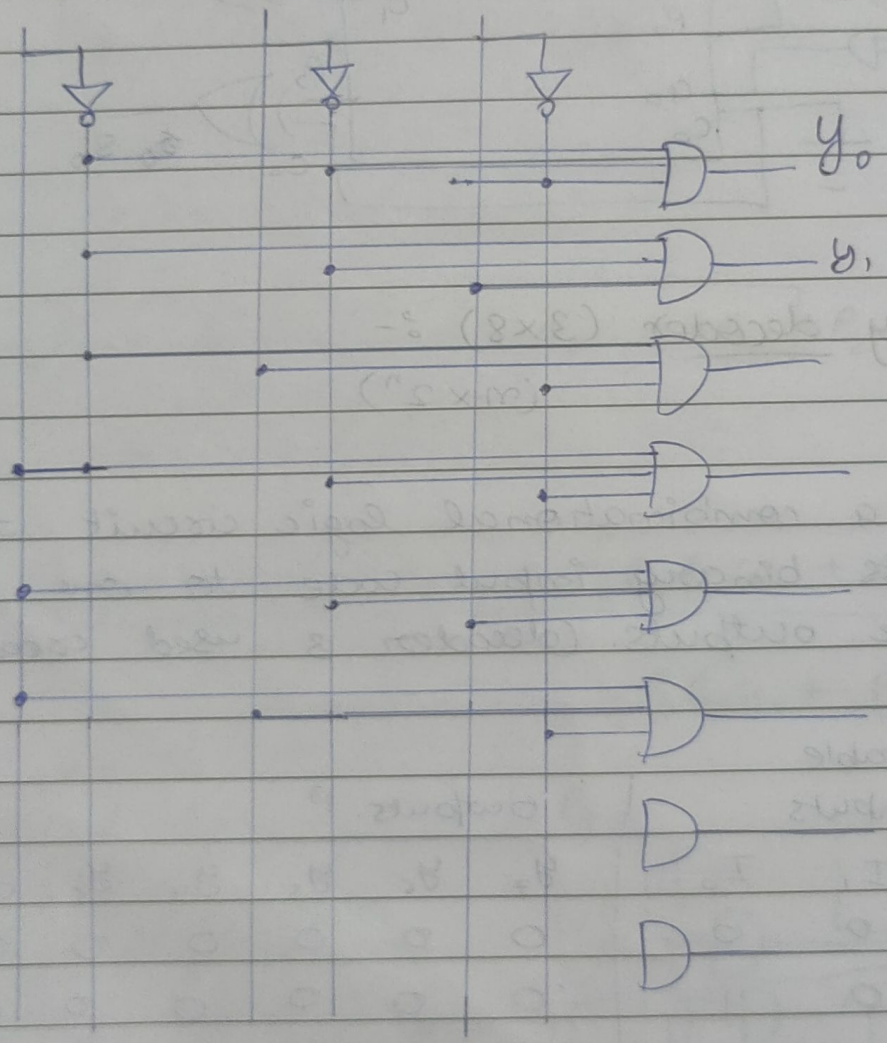
$y_4 = I_2 \bar{I}_1 \bar{I}_0$

$y_5 = I_2 \bar{I}_1 I_0$

$y_6 = I_2 I_1 \bar{I}_0$

$y_7 = I_2 I_1 I_0$





Ye galat hogya

☆. multiplexer :-

If selection lines 's'
then size of multiplexer = $2^s \times 1$.
eg. $s = 3$, (8×1)

→ A multiplexer is a combinational logic circuit that performs many to one function. The selection lines are used to select only one input among multiple inputs to be transferred to the single output line.

→ Truth table :-

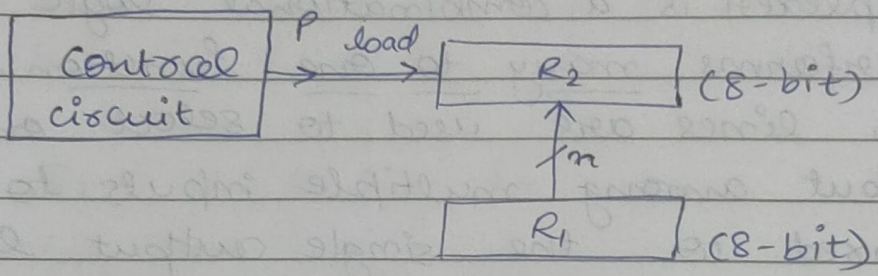
Selection lines			output.
S_2	S_1	S_0	y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

$$y = \bar{S}_2 \bar{S}_1 \bar{S}_0 D_0 + \bar{S}_2 \bar{S}_1 S_0 D_1 + \bar{S}_2 S_1 \bar{S}_0 D_2 + \bar{S}_2 S_1 S_0 D_3 + S_2 \bar{S}_1 \bar{S}_0 D_4 + S_2 \bar{S}_1 S_0 D_5 + S_2 S_1 \bar{S}_0 D_6 + S_2 S_1 S_0 D_7.$$

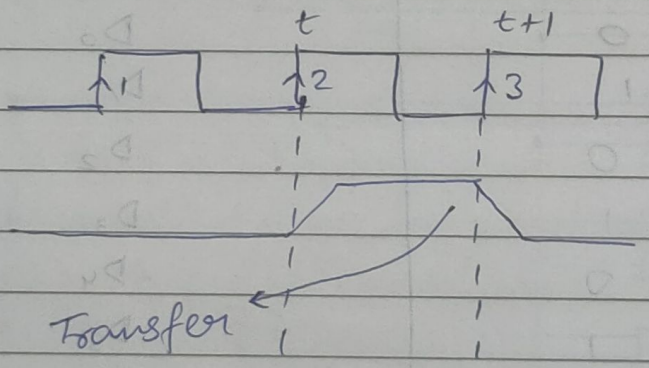
M T W T F S S
Page No.:
Date: 5/9/23
YOUVA

☆. RTL :- (Register Transfer language).

→ The symbolic notation used to describe the micro-operation transfer among register is called RLT.



Loading signal always connected to destination.
when P - high (R_2 takes signal) (loading)
P - low (R_2 not takes signal)



(takes time of one cycle for loading)

Representation :- $P : R_2 \leftarrow R_1$, $R_1 \leftarrow R_2$.

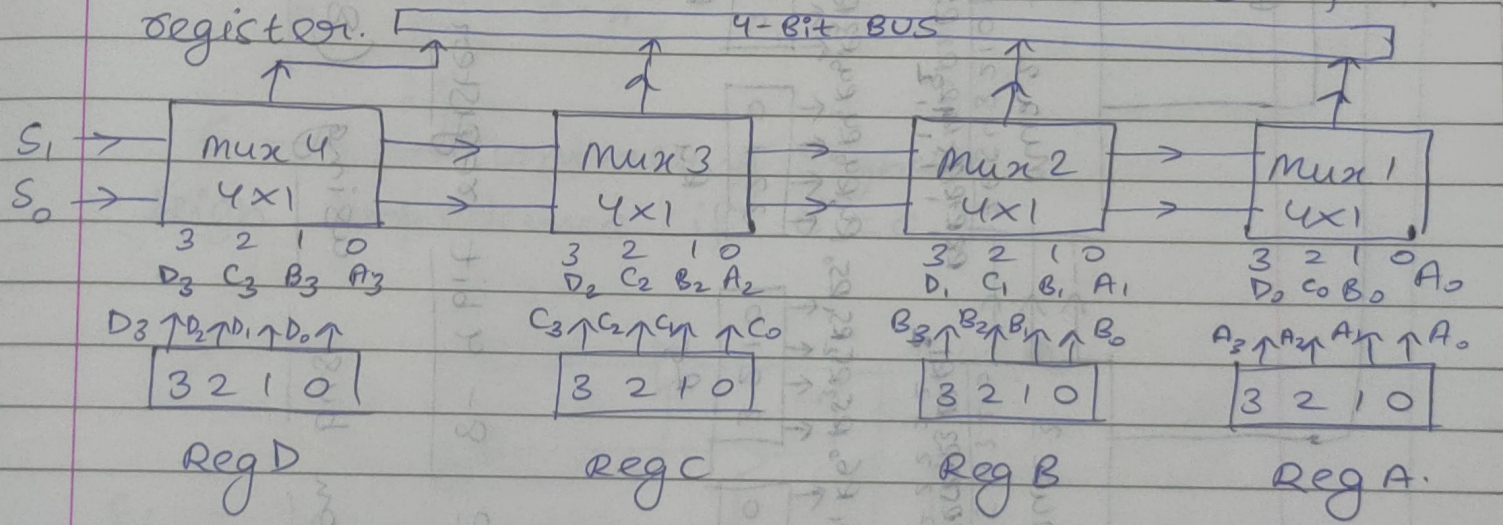
M T W T F S S
Page No.:
Date: 8/9/23
YOUVA

☆. Register transfer in a common bus configuration :-

→ We have two approaches for implementing register transfer in common bus configuration
① Multiplexer
② Tri-state Buffer.

① 1st approach (Multiplexer):

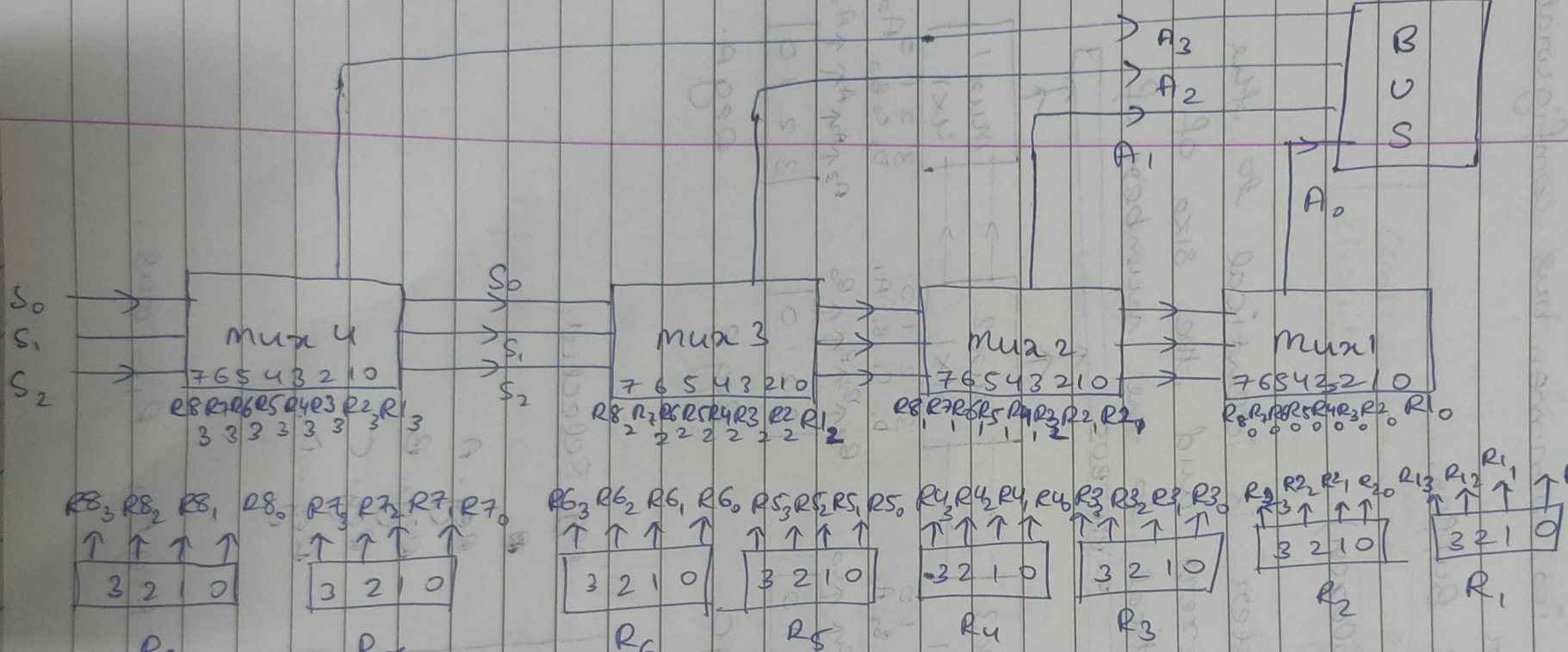
→ The length of register is identical to the number of multiplexers and the size of multiplexer is identical to the number of register.



Selection lines		Reg. Selected.
S_1	S_0	
0	0	Reg. A
0	1	Reg. B
1	0	Reg. C
1	1	Reg. D.

P: BUS ← Reg A, R_1 ← BUS.

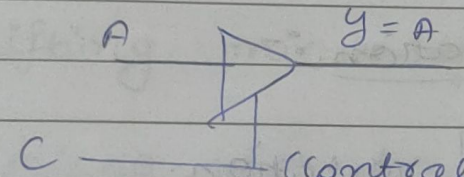
Q. Design a register transfer operation in a common bus config. Given 8 - 4 bit registers.



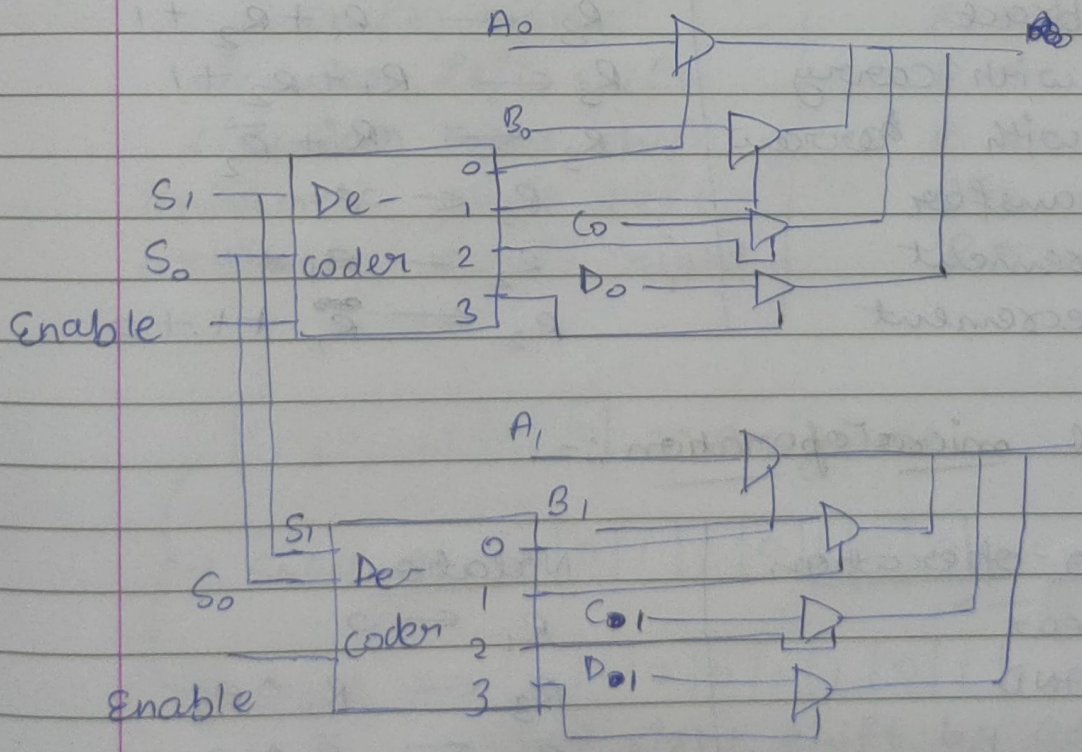
Designing of 8 - 4 bit register configuration on common bus using (8×1) MUX.

Selection lines			Reg. selected.
S_2	S_1	S_0	
0	0	0	R_1
0	0	1	R_2
0	1	0	R_3
1	0	0	R_4
0	1	1	R_5
1	1	0	R_6
1	0	1	R_7
1	1	1	R_8.

② 2^n approach :- (Using tri-state buffer)



High impedance if C = 0.



☆ Memory Transfer :-

Read operation :

Read : $R_1 \leftarrow M[AR]$
Write : $M[AR] \leftarrow R_1$

☆ Types of microoperation :-

1. Arithmatic operations.
2. Logical.
3. Shifting microoperation.

① Arithmetic microoperation :-

micro-operation	Natation.
Add	$R_3 \leftarrow R_1 + R_2$
Subtract	$R_3 \leftarrow R_1 + \bar{R}_2 + 1$
Add with carry	$R_3 \leftarrow R_1 + R_2 + 1$
Sub. with borrow	$R_3 \leftarrow R_1 + \bar{R}_2$
Transfer	$R_2 \leftarrow R_1$
increment	$R_2 \leftarrow R_1 + 1$
decrement	$R_2 \leftarrow \bar{R}_1 + 1 + 0$

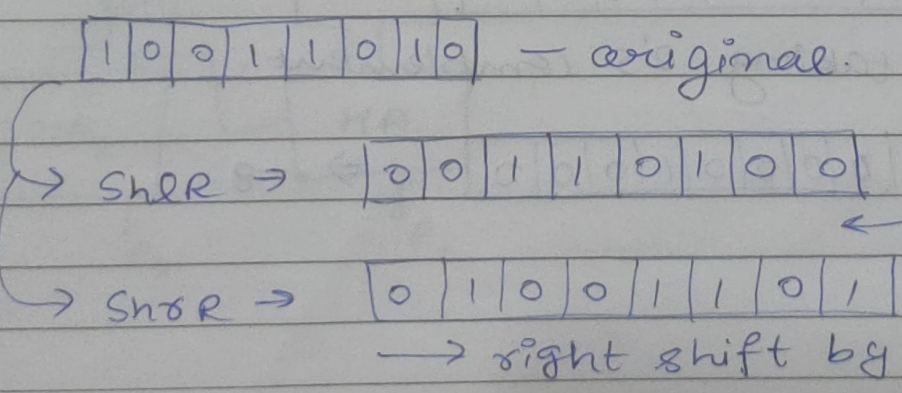
② Logical microoperation :-

micro-operation	Natation
clear	$R_1 \leftarrow 0$
AND	$R_3 \leftarrow R_1 \wedge R_2$
	$R_3 \leftarrow \bar{R}_1 \wedge R_2$
	$R_3 \leftarrow R_1 \wedge \bar{R}_2$

NAND	$R_3 \leftarrow \overline{R_1 \cap R_2}$
OR	$R_3 \leftarrow R_1 \cup R_2$ $R_3 \leftarrow \overline{R}_1 \cup R_2$ $R_3 \leftarrow R_1 \cup \overline{R}_2$
NOR	$R_3 \leftarrow \overline{R_1 \cup R_2}$
Ex-OR	$R_3 \leftarrow R_1 \oplus R_2$
Ex-NOR	$R_3 \leftarrow \overline{R_1 \oplus R_2}$
Complement	$R_2 \leftarrow \overline{R}_1$
Set to 1	$R_1 \leftarrow 1$

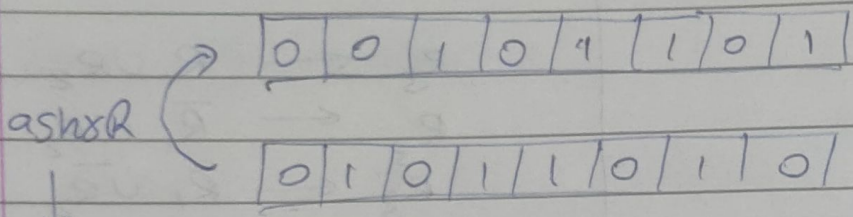
③ Shifting micro-operation :-

Micro-operation	Notation
shift left	R ← shlR
shift right	R ← shrR
circular left shift	R ← cilR
circular right shift	R ← cirR
arithmetic shift left	R ← ashlR
arithmetic shift right	R ← ashrR.

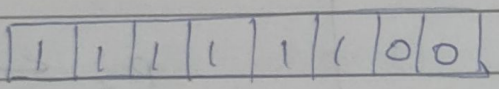


in cilR → MSB becomes LSB.
cirR → LSB becomes MSB.

ashrR → MSB never changes.

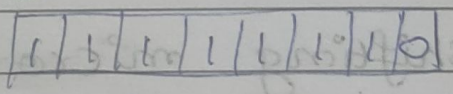


↓
divides the content by 2.

-4 ⇒ 

100 —1's→ 011 +1 ⇒ 100

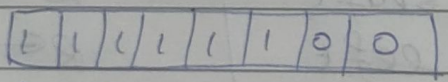
Applying AshrR.



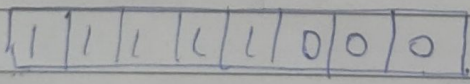
↓
taking 2's complement.

0 0 0 0 0 0 1 0 ⇒ -2.

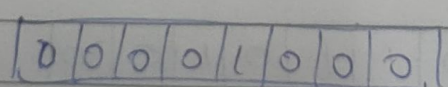
ashlR → identical to shlR microoperation (mul. by 2^n)

-4 ⇒ 

↓ applying asheR.



↳ taking 2's complement.



⇒ -8.

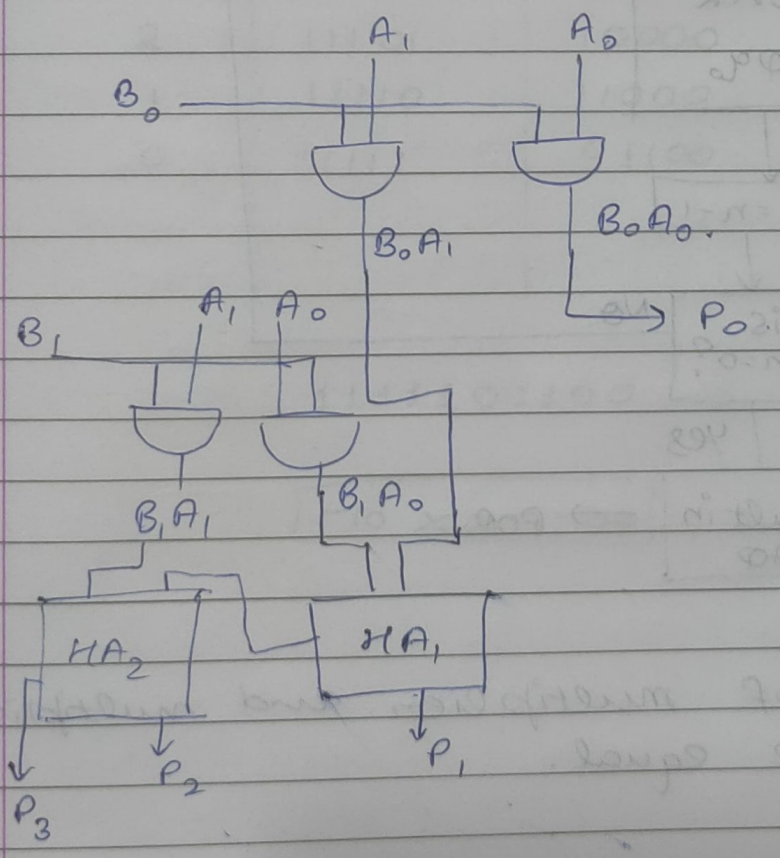
* Binary multiplication :-

M → multiplicant
× Q → multiplier.

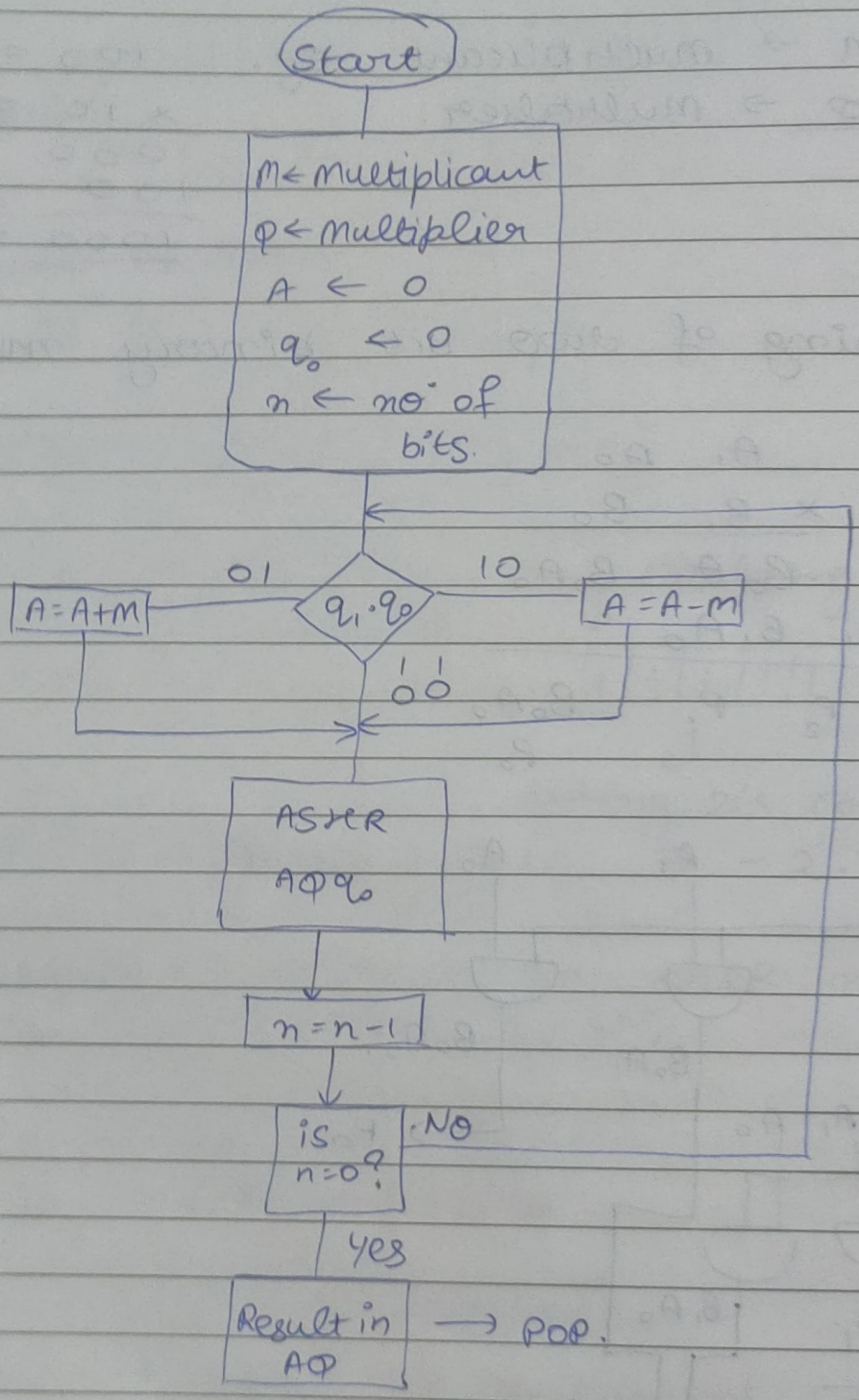
eg.
$$\begin{array}{r} 100 = 4 \\ \times\ 10 = 2 \\ \hline 000 \\ 100 \\ \hline 1000 = 8 \end{array}$$

→ Designing of two bit binary multiplier.

eg.
$$\begin{array}{r} A_1\ A_0 \\ \times\ B_1\ B_0 \\ \hline B_0A_1\ B_0A_0 \\ B_1A_1\ B_1A_0 \\ \hline P_3\ P_2\ P_1\ B_0A_0 \\ P_0 \end{array}$$



* Boothe Algorithm :-



→ No of bits of multiplier and multiplicant both must be equal.

Q. Perform the binary multiplication between -10 and 2 using boothe's algoritum.

→ $-10 \rightarrow (01010)' + 1 = 10101 + 1$

$(-10) = 10110$

$m = (-10) = 10110$

$\Phi = (+2) = 00010$

$A = 00000$, $q_0 = 0$ ↳ (LSB of 'Φ')

$n = 5$.

n	A	Φ	q_0	Action.
5	00000	00010	0	Initialisation
4	00000	00001	0	ASHR
	01010	00001	0	A = A - m.
3	00101	00000	1	ASHR.
	11011	00000	1	A = A + m
2	11101	10000	0	ASHR.
1	11110	11000	0	ASHR
0	11111	01100	0	ASHR.

11111 01100 → that is (-20)

$\boxed{-10 \times 2 = -20}$

Q2) Perform the binary multiplication of -16 with -5 with booth's algo.

$-16 = (010000)' + 1$

$(-16) = 110000$

$-5 = (000101)' + 1$

$(-5) = 111011$

$m = 110000$

$Q = 111011$

$n = 6$

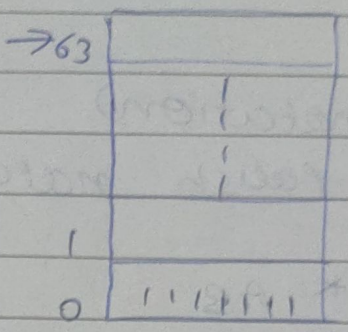
$A = 000000$, $q_0 = 0$.

n	A	Q	q_0	Action.
6	000000	111011	0	Initialization
	010000	111011	0	A = A-m
5	001000	011101	1	ASHR
4	000100	001110	1	ASHR
	110100	001110	1	A = A+m
3	111010	000111	0	ASHR
	001010	000111	0	A = A-m
2	000101	000011	1	ASHR
1	000010	100001	1	ASHR
0	0000001	010000	1	ASHR

$AQ = 0000010010000 = (80)$

$(-16) \times (-5) = 80$

* Stack organisation :-



Works on LIFO principle (Last in first out).

Stack Pointer (Address Register)

$2^6 = 64$

require 6-bits for address.

flag register

I	0
Empty	Full

→ **PUSH :**

SP → 0 ; Empty → 1 , full → 0.

SP ← SP + 1

M[SP] ← DR → Data Register.

o when SP → 63 [1111111]

SP+1 +1

1000000 (only 6-bits)

Hence returned to first position.

If (SP = 0) then empty → 0, full → 1.

→ **POP :**

DR ← M[SP]

SP ← SP - 1

If (SP = 0) empty → 1 , full → 0.

☆. Notations :-

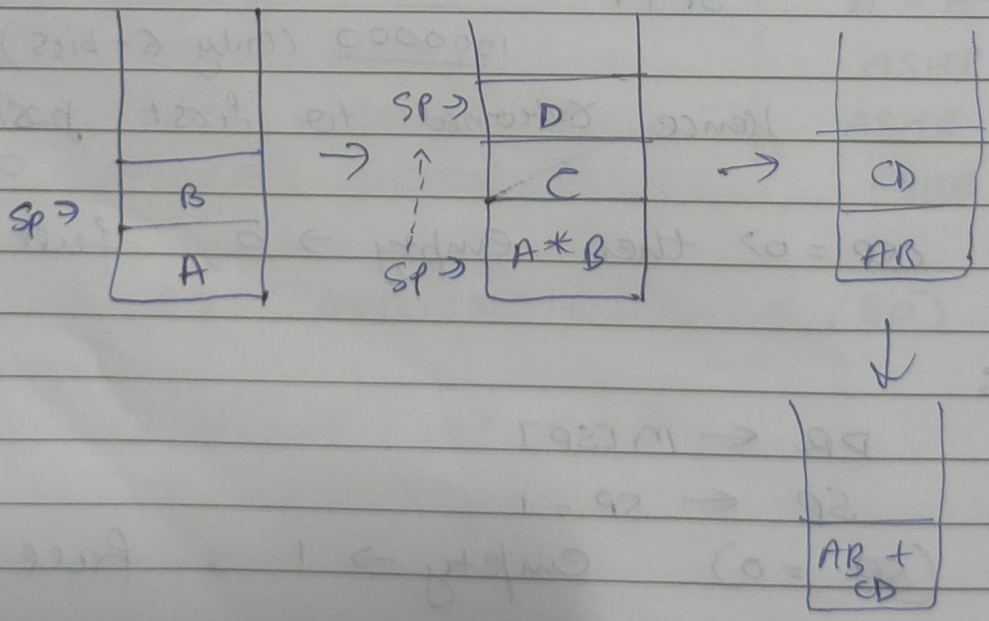
1) Infix
2) Prefix (Polish notation)
3) Postfix (reverse Polish notation)

A * B	* AB	AB *
(Infix)	(Prefix)	(Postfix)

(A * B) + (B * C)
+ (*AB)(*BC)
(AB*)(BC*)+

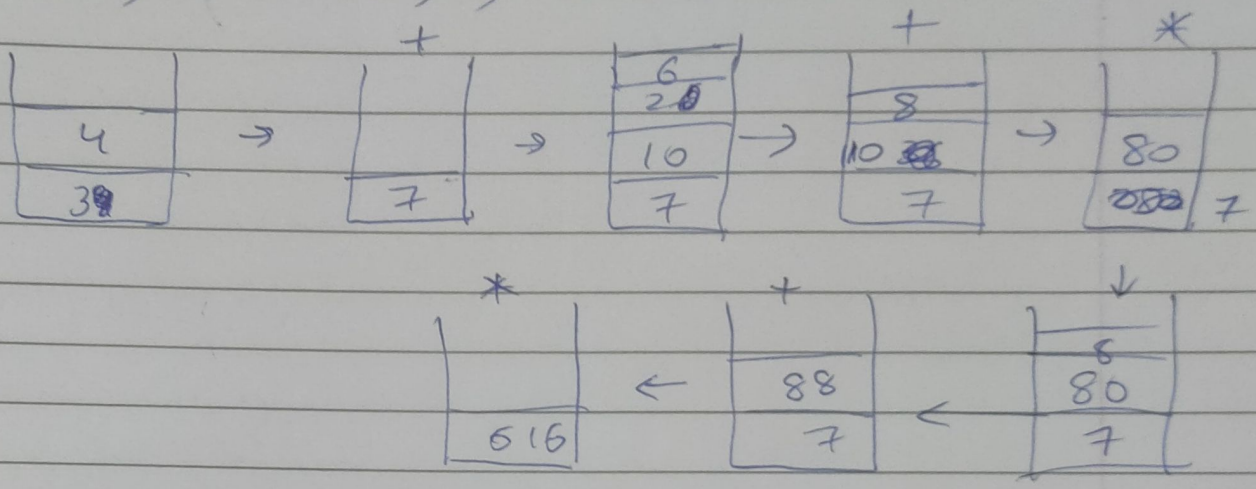
→ Evaluation of arithematic expression using stack by following postfix notation :-

① (A * B) + (C * D)
⇒ postfix
(AB*)(CD*)+



(2) (3+4) (10* (2+6) +8)

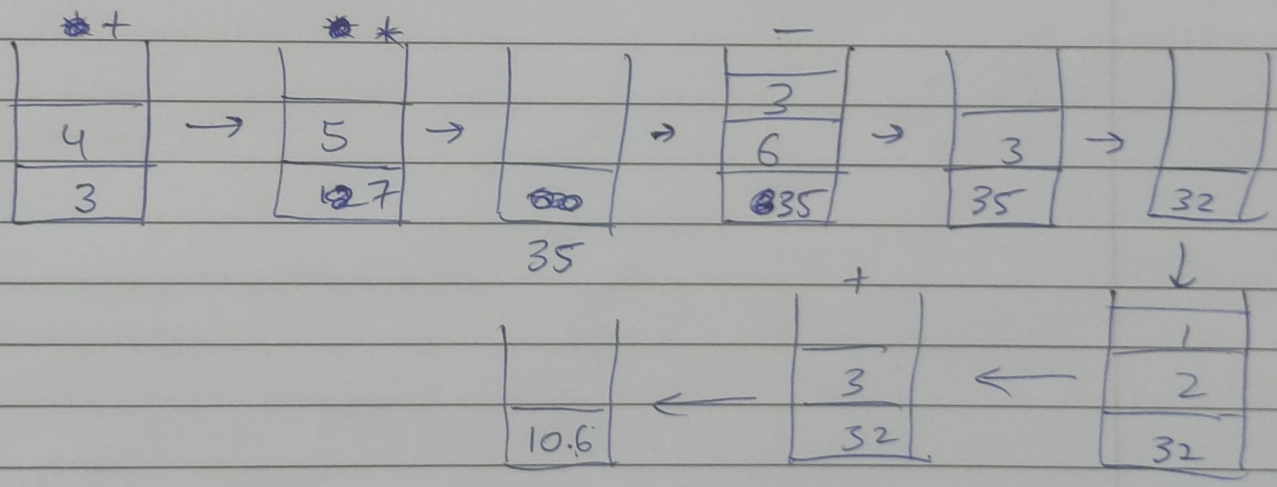
(34+) (10 (26+)*) 8+*



(3) (3+4)* 5 − (6−3)
(2+1)

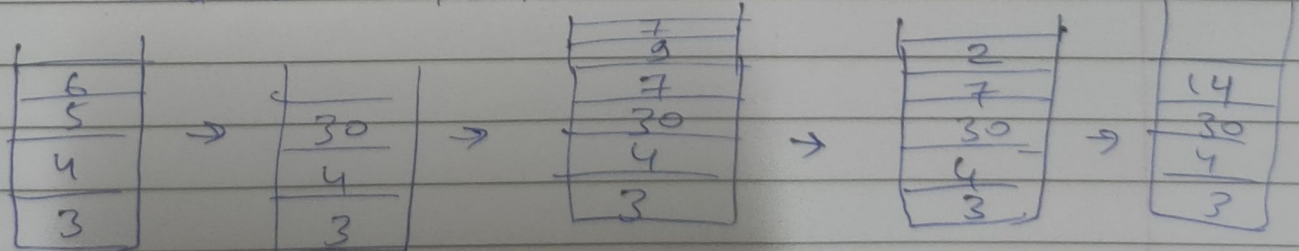
→

(34+)5 * (63−) − (21+) /



(4) (3) (4)((56*) * 7(97−) *+)*
3 4 5 6 * 7 9 7 − * + * +

→



44