Date _/_/_

Basic Functional Unit of a Computer

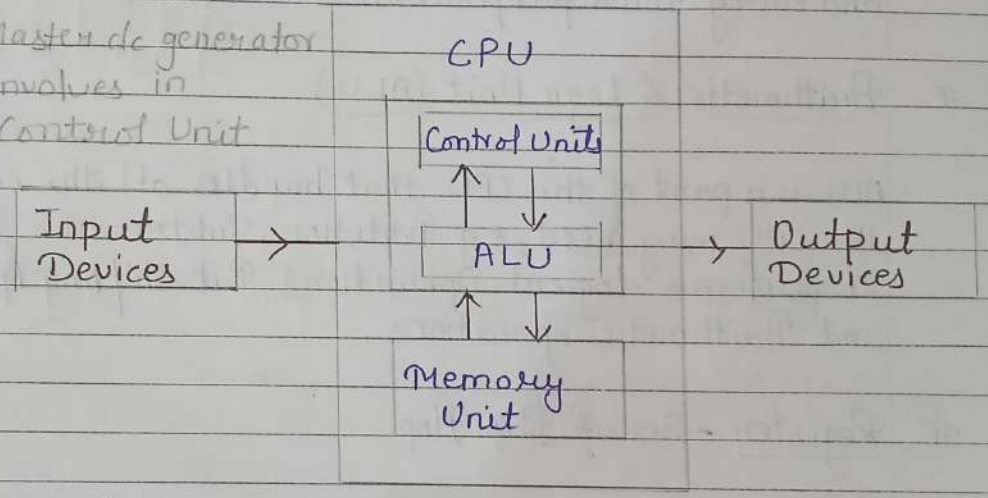
Von Neumann Architecture

In this stored-program concept, programs and data are stored in a separate storage unit called memories and are treated the same.

OR

Defines the programs & operands in same location.

Master dc generator involves in Control Unit



It is also known as ISA (Instruction set architecture) computer and is having three basic units:

1. The Central Processing Unit (CPU)
2. The Main Memory Unit
3. The Input/Output Device.

1. Central Processing Unit-

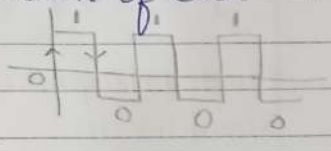
CPU is defined as the it is an electric circuit used for the executing the instruction of computer program.

Date _/_/_

It has following major components:
1. Control Unit (CU)
2. Arithmetic and Logic Unit (ALU)
3. Variety of Register.

Control Unit — 11 bits generates 2^{11} Location/Address

1. The CU defines the order of the execution of instruction.
2. It also generates the no. of timing signals (clock waveform) that are synchronized with execution of each /pulses. and every microproperation.



Arithmetic & Logic Unit (ALU)

ALU is a part of the CPU that handles all the calculations the CPU may need, e.g. Addition, Subtraction, Comparisons, It performs Logical Operations, Bit Shifting Operations and Arithmetic operations.

Register - Set of flip flop.

Refer to high-speed storage areas in the CPU. The data processed by the CPU are fetched from the registers. There are diff. types of registers used in architecture.

IR
PC (Program Counter)
INPR
MAR (Memory Address Register)
MDR (Memory Data Register)
OUTR (

Date _/_/_

1 Mark

Signed Number Representation

- Sign-Magnitude
- 1's Complement
- 2's complement

+ve no. are unsigned
-ve no. are signed

+ve → '0'
-ve → '1'

⇒ **Sign-Magnitude**

Binary of :

-5 → 1 0000101
MSD — Binary of 5

Sign Magnitude

-15 → 1 0001111
S — M

8-bit format → $2^8 = 256$ Range → -127 to 127 for 8-bit

Decimal	Binary	Sign-Magnitude interpretation
0	00000000	+0
1	00000001	+1
2		
3		
⋮		
127	01111111	+127
128	10000000	-0
129	10000001	-1
⋮		
256	11111111	-127

Generalised Range of Signed Magnitude

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

Date _/_/_

⇒ 1's Complement Representation

-5

+5 → 00000101

1's Comp. ↓

Binary of -5 → 11111010

1's Complement Interpretation

Decimal	Binary	1's Complement Representation
0	00000000	+0
1	00000001	+1
2	00000010	+2
3		+3
⋮		⋮
127	01111111	+127
128	10000000	-127
129	1's 01111111 → +127	-126
254		⋮
255	11111111	-0

Range :- $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$

⇒ 2's Complement Representation

-5 +5 → 00000101 Take 1's comp.

1's ↓ +1

-5 → 11111010

\+ 1

Binary of -5 in 2's Comp. → 11111011

Date _/_/_

Decimal	Binary	2's Comp. Representation
0	00000000	+0
1	00000001	+1
2		+2
3		+3
⋮		⋮
127	01111111	+127
128	10000000	-128
129	is ↳ 01111111	-127
⋮	+ 1	⋮
254	10000000 → +128	⋮
255	11111111	-1

Range :- -128 to $+127$

Generalised Range :- -2^{n-1} to $+2^{n-1}-1$

Signed number Addition & Subtraction in (2's Complement) form (Default)

Binary Addition

1. $(+5)+(+7)$
2. $(+5)+(-7)$
3. $(-5)+(+7)$
4. $(-5)+(-7)$

Rules :-

1. Determine the binary of the given no.
2. Perform the addition, if there is a carry out then discard it.
3. If the result is -ve then determine the 2's Complement of the result except the sign bit.

Date _/_/_

Ex-1 $(+5)+(+7)$

Sol.

$+5 \rightarrow 00000101$

$+7 \rightarrow 00000111$

00001100 Ans

$0+0=0$

$0+1=1$

$1+0=1$

$1+1=0$ Carry $=1$

$\{1+1+1=1$ Carry $=1\}$

Ex-2 $(+5)+(-7)$

$+5 \rightarrow 00000101$

$-7 \rightarrow 11111001$

11111110

(-2)

$+7 \rightarrow 00000111$

$\downarrow$

$-7 \rightarrow 11111000$

$+\quad 1$

11111001

$\because$ Result is (-ve), take the 2's Comp.

$\rightarrow -(00000010)$ Ans

-2

1-bit 7-bit

Ex-3 $(-5)+(+7)$

$-5 \rightarrow 11111011$

$+7 \rightarrow 00000111$

$1\ 00000010$ Ans

$+5 \rightarrow 00000101$

$\downarrow$

$-5 \rightarrow 11111010$

$+\quad 1$

11111011

Ex-4 $(-5)+(-7)$

$-5 \rightarrow 11111011$

$-7 \rightarrow 11111001$

$\downarrow 11110100$

$\Rightarrow -(00001100)$

-12 Ans

11110100

00001011

$+\quad 1$

00001100

PASSION

Date _/_/_

Binary Subtraction

A → Minuend
-B → Subtrahend

Rules :-

1. Determine the binary of the given two no.s.
2. Take 2's complement of Subtrahend & add it to the Minuend.
3. Now, Apply the Rules of Binary Addition.

Ex-1 (+5) - (+7)

① +5 → 00000101
+7 → 00000111 ... +7 -

② Taking 2's complement of subtrahend.

00000111 —1's→ 11111000
+ 1
11111001

③ Add it to the minuend

00000101
+ 11111001
11111110 → -2

⇒ -(0000010) Ans

1's ↓ 1111110
0000001
+ 1
0000010

Ex-2 (+5) + (-7)

+5 → 00000101
-7 → 11111001

Taking 2's complement of subtrahend

11111001 —1's→ 00000110
+ 1
00000111

Add it to the minuend

Date _/_/_

```
   1 1 1
   00000101
 + 00000111
 ----------
   00001100
```
Ans

Ex-3 (-5) - (+7)

-5 → 11111011

+7 → 00000111

2's comp. of Subtrahend

00000111 $\xrightarrow{1's}$ 11111000 $\xrightarrow{+1}$ 11111001

Add it to the minuend.

```
   00000100
 + 11111001
 ----------
 1 00000101
```

```
   1 1 1 1   1 1
   11111011
 + 11111001
 ----------
 1 1 1 0 1 0 0
```

-(00001100)

↳ -12

00000101 $\xrightarrow{1's}$ 11111010 $\xrightarrow{+1}$ 11111011

11110100 $\xrightarrow{1's}$ 00001011 $\xrightarrow{+1}$ 00001100

Ex-4 (-5) - (-7)

-5 → 11111011

-7 → 11111001

2's comp. of subtrahend

11111001 $\xrightarrow{1's}$ 00000110 $\xrightarrow{+1}$ 00000111

Add it to the minuend

Date _/_/_

```
 1 1 1 1 1 1
  1 1 1 1 1 0 1 1
+ 0 0 0 0 0 1 1 1
-------------------
1 0 0 0 0 0 0 1 0
```

```
0 0 0 0 0 0 0 1 0
    1's
1 1 1 1 1 1 0 1
        +     1
---------------
1 1 1 1 1 1 1 0
```

Overflow Condition

IC XOR - 7486

① (+6) + (+5)

Cout, Cin

+6 → 1 1 0

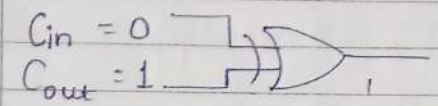
+5 → 1 0 1

(1) 0 1 1

↳ this 1 shows overflow cond^n

Cin = 0

Cout = 1



O = Cin ⊕ Cout

② (-6) + (-5) (+6) + (+7)

Double

-6 → 0 1 0 +6 → (1) 1 1 0 → 3-bit

-5 → +7 → 1 1 1

1 1 0 1

```
1 1 0
↓ 1
0 0 1
  + 1
-----
0 1 0
```

Cin = 1

Cout = 1

In this case, overflow cond^n doesn't occur.

(1) 1 → 4-bit

③ +6 → 0 1 1 0

+7 → 0 1 1 1

1 1 0 1

Cin = 1 } overflow cond^n

Cout = 0 }

PASSION

4 $-6 + (-7)$

$-6 \rightarrow 0\,1\,0$

$-7 \rightarrow 0\,0\,1$

$0\,1\,1$

$C_{in} = 0$, $C_{out} = 0$ } not overflow.

$+6 \rightarrow 110$

1's ↓ 001

$+1$

$-6 \rightarrow 010$

$+7 \rightarrow 111$

1's ↓ 000

$+1$

001

5 $(+2) + (127)$

$+2 \rightarrow 00000010$

$+127 \rightarrow 01111111$

10000001

$C_{in} = 1$, $C_{out} = 0$ } overflow cond"

6.

Q1. $(-2) + (-127)$

Q2 $(+2) + (+127)$

Q3 $(-60) + (-80)$

$-60 \rightarrow 11000100$

$-80 \rightarrow 10110000$

101110100

$+60 \rightarrow 00111100$

1's ↓ 11000011

$+1$

$-60 \rightarrow 11000100$

$+80 \rightarrow 01010000$

1's ↓ 10101111

$+1$

10110000

Soln (1) $-60 \rightarrow 11000100$

$-80 \rightarrow 10110000$

101110100

11000100

$+10110000$

101110100

Carry out exist, so discard it.

The Result is, $-(0001100)$

$C_{in} = 1$.

$C_{out} = 0$

∵ Both the bits are diff. or C_{in} & C_{out} are diff.

So, the cond" of overflow exist.

Rough

$+60 \rightarrow 00111100$

1's ↓ 11000011

$+1$

$-60 \rightarrow 11000100$

$+80 \rightarrow 01010000$

1's ↓ 10101111

$+1$

$-80 \rightarrow 10110000$

11101100

1's ↓ 00010011

$+1$

00010100

-60

-80

-140 in 2's comp. ↓

10001100

Date _/_/_

Practice Sheet 1

1. Considering an 8-bit processor, perform the following arithmetic operations & state whether the overflow cond" exist or not.

a) (-64) + (-79) Overflow

-64 → 11000000
-79 → 10110001

11000000
+ 10110001
① 01110001

Carrout exist, so discard it,
The result is -(0001111)

Cin = 1,
Cout = 0
∵ Cin & Cout are diff.
So, Overflow cond" exist.

64 → 01000000
1's ↓ 10111111
+ 1
11000000

79 → 01001111
1's ↓ 10110000
+ 1
10110001

-64 → 11000000
1's ↓ 00111111
+ 1
00111111

b) 125 + 2 No overflow

125 → 01111101
2 → 00000010

01111101
+ 00000010
01111111

∵ Cin & Cout are 0
∴ Overflow cond" doesn't exist.

Date _/_/_

c) 111 - 126 No overflow 10001111

111 → 01101111
126 → 01111110

Taking 2's complement of subtrahend
01111110 1's ↓ 10000001
+ 1
10000010

Add it to the minuend
01101111
+ 10000010
11110001

∵ Cin = 0 } No overflow
Cout = 0

00001110
1's ↓ 11110001
+ 1
11110010

d) (-100) - (-100) Overflow

-100 → 10011100
-100 → 10011100

Takes 2's comp. of subtrahend,
10011100 → 01100011
+ 1
01100100

⇒ 10011100
01100100
① 00000000

Cin = 1 } Overflow cond"
Cout = 0

100 → 01100100
1's ↓ 10011011
+ 1
10011100

e) 100 + 100 → overflow 11001000

100 → 01100100

100 → 01100100

11001000

∵ Cin =

Cout =

f) 75 − (−55) No overflow 01101010

75 → 01001011

−55 → 11001001

Take 2's comp. of subtrahend

→ 01001011

00110111

10000010

55 → 00110111

1's → 11001000

+ 1

11001001

1's → 00110110

+ 1

00110111

∵ Cin =

Cout =

g) (−56) + 65 No overflow 00001001

−56 → 11001000

65 → 01000001

10001001

56 → 00111000

1's → 11000111

+ 1

11001000

∵ Cin =

Cout =

h) 127 + (−128) No overflow 10000001

127 → 01111111

−128 → 10000000

Take 2's comp. of subtrahend

10000000 → 1's → 01111111

+ 1

10000000

01111111

+ 10000000

11111111

128 → 10000000

1's → 01111111

+ 1

10000000

∵ Cin =

Cout =

i) (−60) + (−80) Overflow 01110100

−60 → 11000100

−80 → 10110000

101110100

60 → 00111100

1's → 11000011

+ 1

11000100

80 → 01010000

1's → 10101111

+ 1

10110000

Cin =

Cout =

Date = 1/10

2. Considering a 4-bit processor, perform the following arithmetic operations & state whether the overflow condition exists or not. (-8 to +7)

a) 6+7 Overf. 1101

6 → 0110

7 → 0111

1101

Cin =

Cout =

b) (-7) - (-1) No overf, 1110

-7 → 1001

-1 → 1111

Two's comp. of -1

1111 1's → 0000 + 1 = 0001

1001

0001

1010

7 → 0111

1's → 1000 + 1 = 1001

1 → 0001

1's → 1110 + 1 = 1111

Cin =

Cout =

c) 4 - (-5) overf; 1001

4 → 0100

-5 → 1011

Cin = 2's of -5

Cout = 0100 → 0101 → 1011

5 → 0101

1's → 1010 + 1 = 1011

PASSION

Overf, 0111

d) (-1) + (-8)

-1 → 1111

-8 → 1000

①0111

Cin =

Cout =

1 → 0001

1's → 1110 + 1 = 1111

8 → 1000

1's → 0111 + 1 = 1000

e) 3 - 4 → No overf, 1001

3 → 0011

4 → 0100

Two's of 4

0100 1's → 1011 + 1 = 1100

= 0011

1100

1111

Cin =

Cout =

Date _/_/_

Floating Point Operation

1.1×10^3 → Scientific Notation

1.1E3 → Programming Notation

Floating Point ~~Operation~~ Representation → $\pm M \times B^E$

M → Mantassa

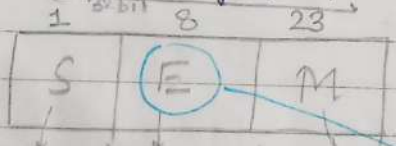
B → Base

E → Exponent

① IEEE 32-bit format :-
(Single Precision format)

② IEEE 64-bit format :-
(Double Precision format)

① IEEE 32-bit format :-

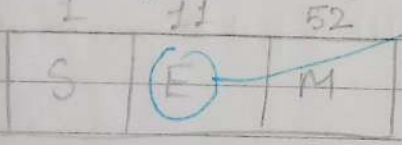


Sign bit — Exponential — Mantassa

Sys. stored in this format →

Biased Exponents
exponents which contains sign with it.

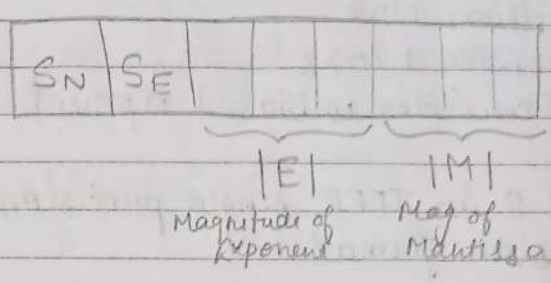
⑤ IEEE 64-bit format



PASSION

Date / /

8-bit floating Point Representation with unbiased exponents :-



Ex- $-13.9 \rightarrow (1101.1110001)_2$

$13 \rightarrow 1101$

$0.9 \times 2 \rightarrow 1.8$ 1

$0.8 \times 2 \rightarrow 1.6$ 1

$0.6 \times 2 \rightarrow 1.2$ 1

$0.2 \times 2 \rightarrow 0.4$ 0

$0.4 \times 2 \rightarrow 0.8$ 0

$0.8 \times 2 \rightarrow 1.6$ 1

Normalised Mantissa

$-(1.101111001 \times 2^3)$

Unbiased

1	0	011	101

4 $\Rightarrow$ $0 \rightarrow 2^4 - 1$

$0 \rightarrow 15$ Biased $\frac{15}{2} = 7.5$

$-7 \quad -7$ $-7 \downarrow \quad \uparrow +7$

-7 to $+8$ unbiased

floor of $(7.5) = 7$

$0 - 2^6 - 1$

$0 - 255$, $\frac{255}{2} = 127.5$

Biased

$-127 \downarrow \quad \uparrow +127$

To get biased

1	1010	101

Range of Unbiased :- -127 to +127

⇒ Conversion from unbiased to biased exponents
In single precision +127
In double precision +1023
* By default we consider unbiased exponent

Q. Represent -13.9 in IEEE single precision & double precision format :-

-13.9

1	8	23
S_N	E	M

$-13.9 \rightarrow -(1101.11001)$

$-(1.101111001 \times 2^3)$

Unbiased exponent = 3
∴ Biased Exponent = 3 + 127 = 130

2 | 130
2 | 65 — 0
2 | 32 — 1

1	10000010	10111001

In Double Precision

1	11	32
1	10000000010	10111001

∴ Biased Exp. = 3 + 1023 = 1026

* Before converting in the biased exponent, first check whether the unbiased exponent is in range or not.

Q. -0.3125
$-0.3125 \rightarrow -(0.0101)_2$
$= (0.01.01 \times 2^{-2})$

Unbiased exponent = -2 ~~+127 = 125~~
∴ Biased exponent = -2 + 127 = 125

2 | 125
2 | 62 — 1
2 | 31 — 0
2 | 15 — 1
2 | 7 — 1
2 | 3 — 1
1 — 1

1	1111101	01
1	8	23

In Double Precision.
Biased exponent = -2 + 1023 = 1021

2 | 1021
2 | 510 — 1

Combinational Logic

digital circuits
- Combinational logic
- Sequential logic

→ Half-Adder

S-1 Design Truth-Table

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

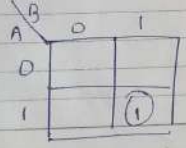
S-2 Computation of Boolean func^n with the help of K-map

Note: The no. of O/P = no. of Kmap

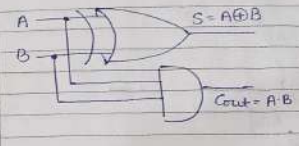
No. of I/P given no. of variable

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

$S = A \oplus B$



$Cout = A \cdot B$

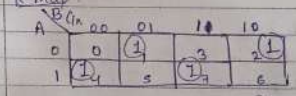


→ Full-Adder

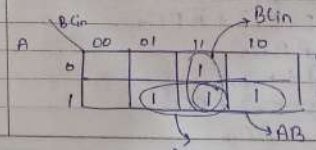
Truth Table

Input			Output	
A	B	Cin	S	Cout
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

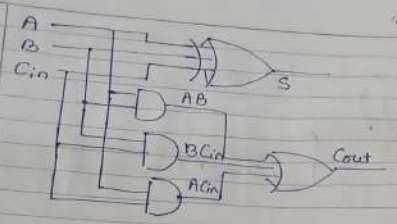
K-map



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

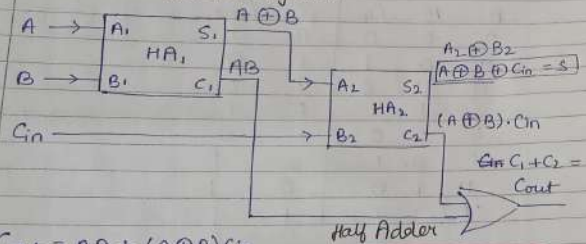


$Cout = AB + BC_{in} + AC_{in}$



Implementation of full adder using two half adder.

Block Diagram



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

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

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

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

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

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

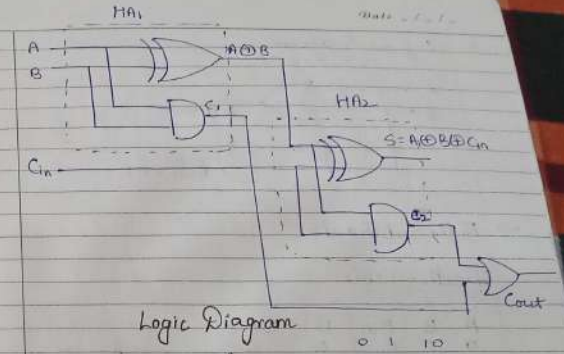
$C_{out} = AB + A\bar{C}_{in} + \bar{A}BC_{in}$

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

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

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

$C_{out} = AB + BC_{in} + AC_{in}$ → To derive



Logic Diagram

⇒ Half-Subtractor { C_{02} H.s. can't subtract

0 1 10

1 0 0 {4}

0 0 1 {1}

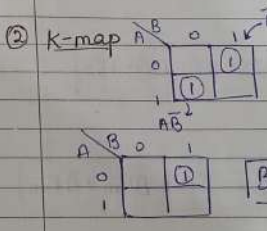
0 1 1 {3}

borrow that's why it is called H.S

① Truth-table.

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

② K-map

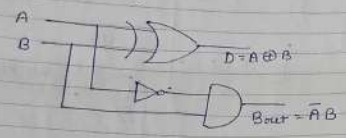


$D = \bar{A}B + A\bar{B}$

$D = A \oplus B$

$B_{out} = \bar{A}B$

③ Logical Diagram



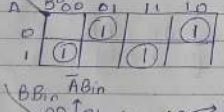
⇝ Full Subtractor

Truth - Table

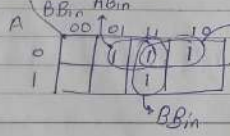
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Bin	D
A	0
B	1

K-map

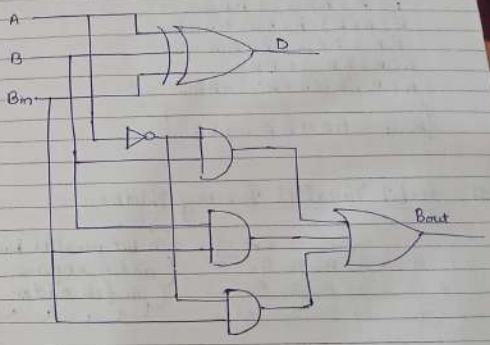


D = A ⊕ B ⊕ Bin

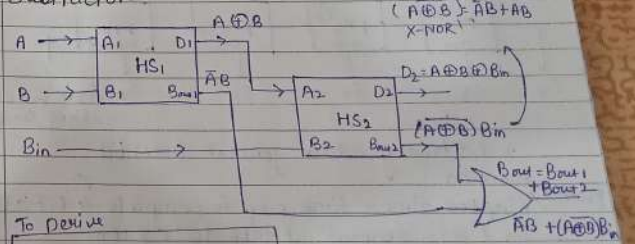


Bout = Ā B + B Bin + Ā Bin

Logical Diagram



Designing of a full subtractor using two half subtractor



To derive

Bout = Ā B + B Bin + A Bin

Date _/_/_

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

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

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

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

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

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

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

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

4-bit Parallel Binary Adder

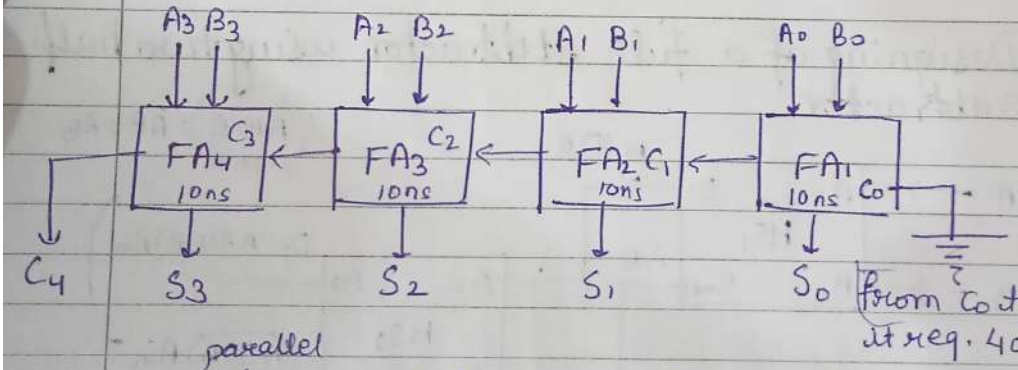
$C_3\ C_2\ C_1\ C_0$

$A = A_3\ A_2\ A_1\ A_0$

$+\ B = B_3\ B_2\ B_1\ B_0$

$S_4\ \ S_3\ S_2\ S_1\ S_0$

(n-bit parallel binary adder require n full adder)



n-bit parallel binary adder generate (n+1) bit.

propagation delay / elapsed time — time req. to complete 4-bit adder.

→ for 1 outp. generat^n it take 10^{-9} s (10ns)

n-bit binary adder req. n x 10ns

Date - / - / -

* for n-bit parallel binary adder

⇒ Elapse time

$n \cdot t_p$

$t_p = 10ns$

↳ t_p → Propogation delay.
↳ $t_{p_{Carry}}$

→ Gates responsible for Carry Propogation delay (10ns)
↳ 2 Gates

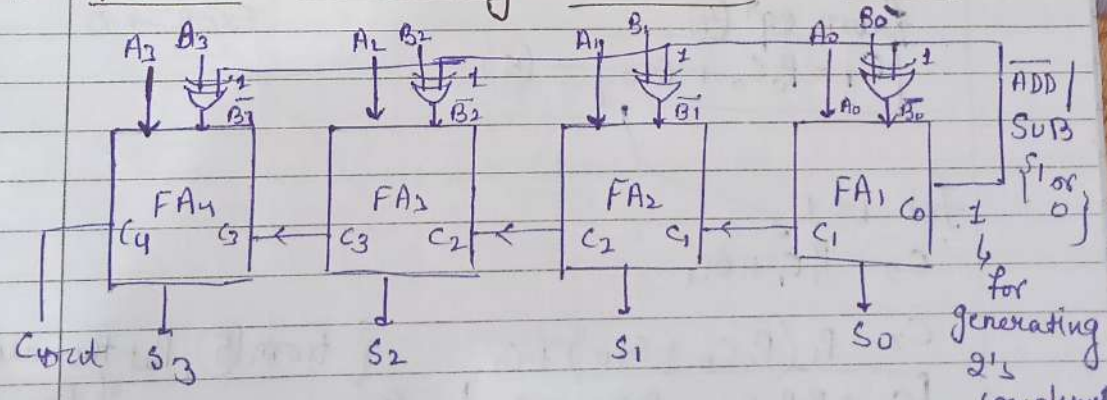
2n gate (Least OR gate & Second least AND gate)

for n-bit parallel binary adder 2n gate are responsible for propogation delay.

Drawback of 4-bit parallel binary adder.

→ Carry propogation delay.

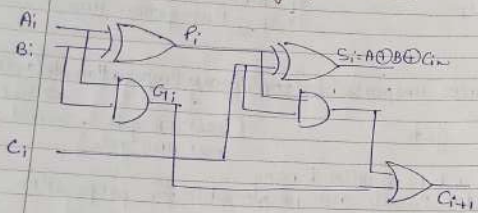
4-bit Parallel binary Adder & Subtractor



$C_0 \rightarrow 0$ {Add^n}

$C_0 \rightarrow 1$ {Sub^n}

Look Ahead Carry Generator



$P_i = A_i \oplus B_i$ — (1)

$G_i = A_i \cdot B_i$ — (2)

$S_i = P_i \oplus C_i$ — (3)

$C_{i+1} = P_i C_i + G_i$ — (4)

Extra, Interview Q.] If we have 100 bits except the last bit is not one then the EXOR will be If no. of ones are odd, EXOR → 1 If no. of ones even EXOR → 0

for i = 0

from eqn (4)

$C_1 = P_0 C_0 + G_{10}$ — (5)

for i = 1

$C_2 = P_1 C_1 + G_{11}$

$C_2 = P_1 (P_0 C_0 + G_{10}) + G_{11}$ { from (5), Put the value of C_1 }

$C_2 = P_1 P_0 C_0 + P_1 G_{10} + G_{11}$ — (6)

for i = 2

$C_3 = P_2 C_2 + G_{12}$ { from (6) }

$C_3 = P_2 (P_1 P_0 C_0 + P_1 G_{10} + G_{11}) + G_{12}$

$C_3 = P_2 P_1 P_0 C_0 + P_2 P_1 G_{10} + G_{11} P_2 + G_{12}$ — (7)

for i = 3

$C_4 = P_3 C_3 + G_{13}$

$C_4 = P_3 \{ P_2 P_1 P_0 C_0 + P_2 P_1 G_{10} + G_{11} P_2 + G_{12} \} + G_{13}$

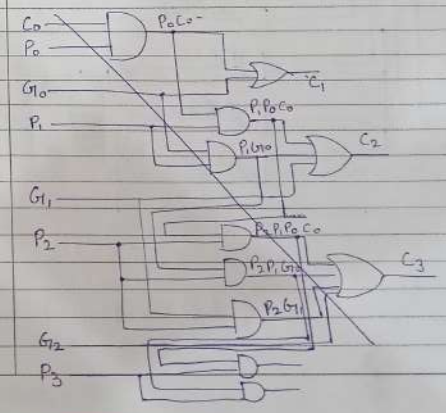
$C_4 = P_3 P_2 P_1 P_0 C_0 + P_3 P_2 P_1 G_{10} + P_3 P_2 G_{11} + P_3 G_{12} + G_{13}$ — (8)

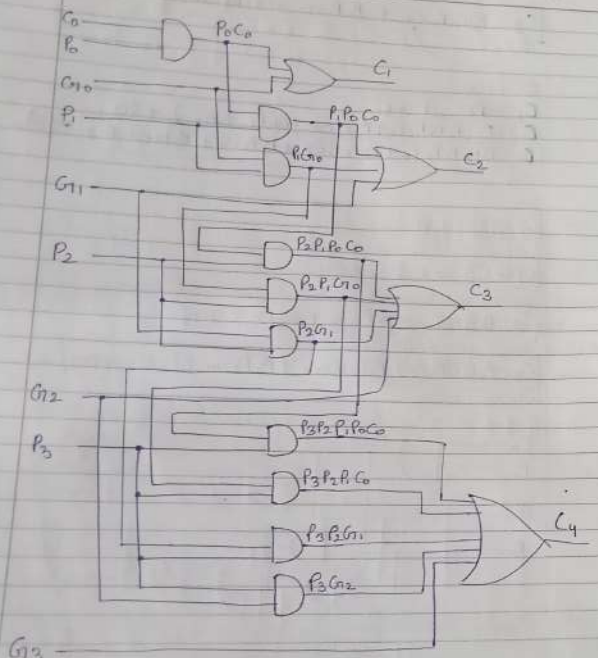
→ $C_1 = P_0 C_0 + G_{10}$

$C_2 = P_1 P_0 C_0 + P_1 G_{10} + G_{11}$

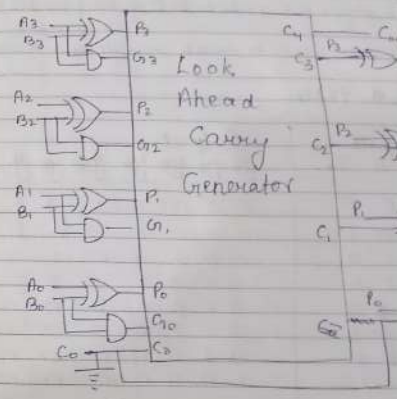
$C_3 = P_2 P_1 P_0 C_0 + P_2 P_1 G_{10} + G_{11} P_2 + G_{12}$

$C_4 = P_3 P_2 P_1 P_0 C_0 + P_3 P_2 P_1 G_{10} + P_3 P_2 G_{11} + P_3 G_{12} + G_{13}$





Look Ahead Carry binary Adder



Binary Decoder (3 x 8)

32 KB
32×2^{10} B
$2^5 \times 2^{10} \times 8$ Bits
$2^{15} \times 8$ bits
15×2^{15}
3×2^3

Used to convert their binary to correspon

In General, $n \times 2^n$

Date - / - / -

3×8 Decoder

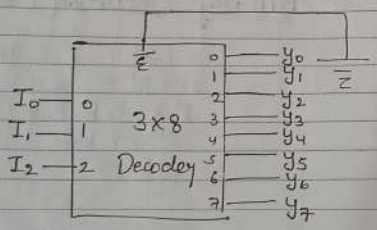
→ It is the combinational logic circuit that converts a binary input code to one of the possible output. → includes all variables.

→ Decoder is used to create min. term.

Truth table

Inputs I_2 I_1 I_0	Outputs 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

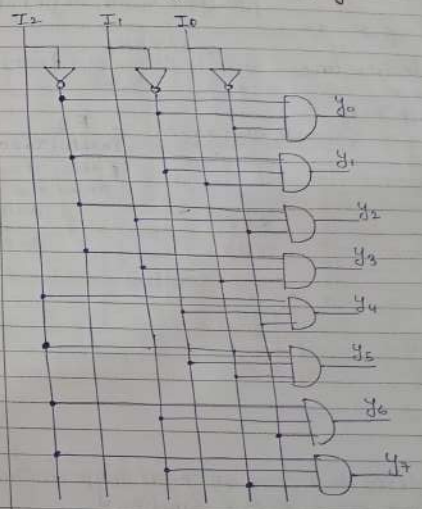
$y_0 = \overline{I_2}\,\overline{I_1}\,\overline{I_0}$
$y_1 = \overline{I_2}\,\overline{I_1}\,I_0$
$y_2 = \overline{I_2}\,I_1\,\overline{I_0}$
$y_3 = \overline{I_2}\,I_1\,I_0$
$y_4 = I_2\,\overline{I_1}\,\overline{I_0}$
$y_5 = I_2\,\overline{I_1}\,I_0$
$y_6 = I_2\,I_1\,\overline{I_0}$
$y_7 = I_2\,I_1\,I_0$



Block Diagram

Date - / - / -

Combinational Logic Circuit diagram



Multiplexer (8×1)

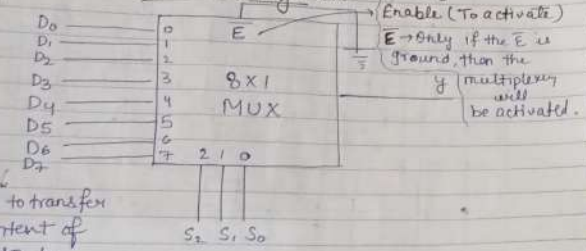
{Multiple inputs, Single Outputs}

* Many to One.

Selection lines (3) (S) ← variable {Used to select inputs}

Size of multiplexer → $(2^S \times 1)$

Block Diagram



Inputs to transfer the content of data to the output.

→ 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 transfer to the single output line.

Truth-table

Selection lines S_2 S_1 S_0	Output 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

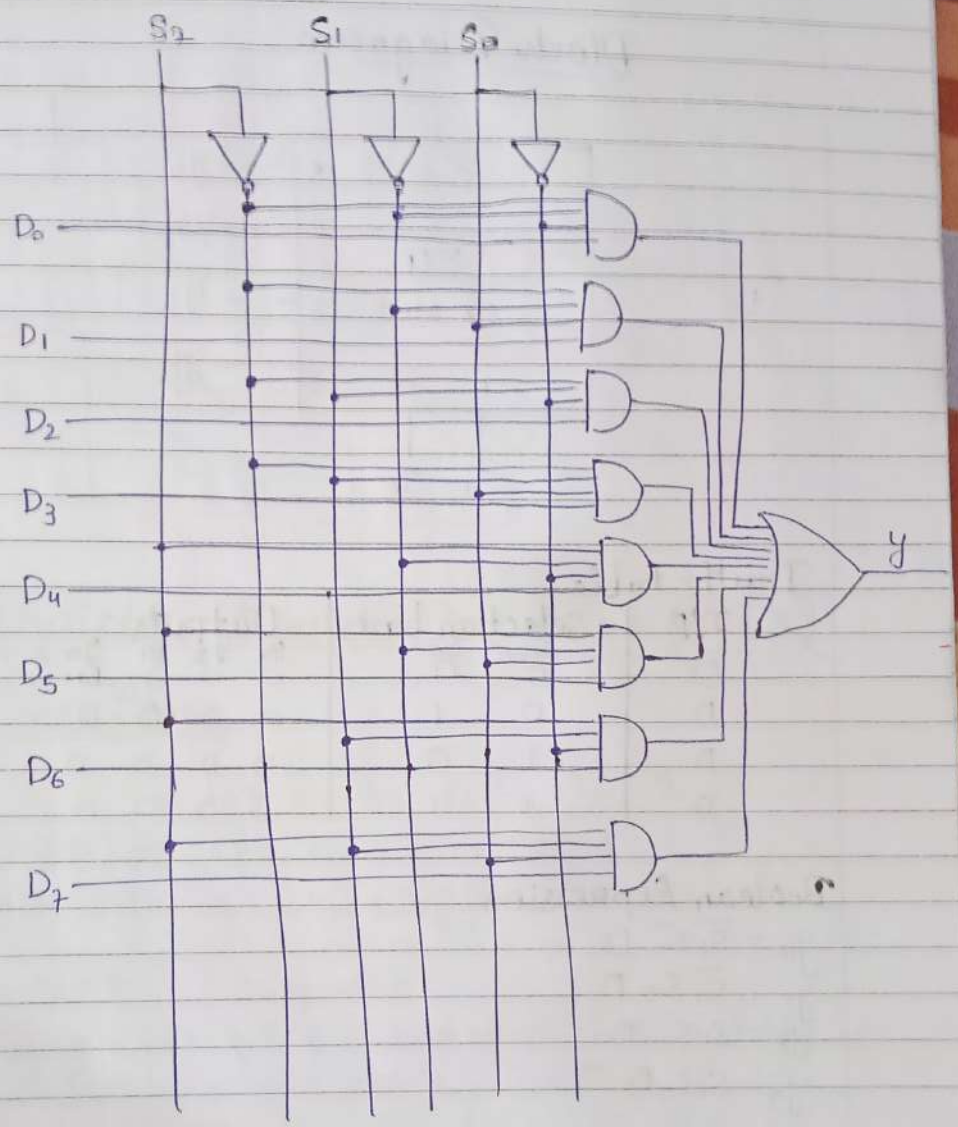
If Quest^n comes in Exam
→ Defn
→ Block diag.
→ Truth table
→ Boolean logic func^n

Boolean function

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

Date : / / .

$y = \bar{S_2}\bar{S_1}\bar{S_0}D_0 + \bar{S_2}\bar{S_1}S_0D_1 + \bar{S_2}S_1\bar{S_0}D_2 + \bar{S_2}S_1S_0D_3 + S_2\bar{S_1}\bar{S_0}D_4 + S_2\bar{S_1}S_0D_5 + S_2S_1\bar{S_0}D_6 + S_2S_1S_0D_7$



Demultiplexer (De-MUX) (1x4)

$2^3 = N$

Block Diagram

Truth-table

I/P	Selection lines S_1	S_0	Outputs y_3	y_2	y_1	y_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

Boolean Expression

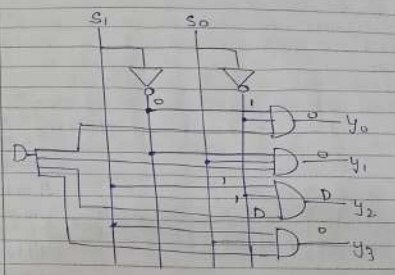
$y_0 = \bar{S_1}\bar{S_0}D$

$y_1 = \bar{S_1}S_0D$

$y_2 = S_1\bar{S_0}D$

$y_3 = S_1S_0D$

Logical Diagram



Register Transfer Language (RTL)

Overflow

	128	64	32	16	8	4	2	1
(+70) →	0	1	0	0	0	1	1	0
(+80) →	0	1	0	1	0	0	0	0
150	①	0	0	1	0	1	1	0

Carry in bit → ① ← Carry out bit

The carry in bit is changing to carry out bit i.e. 0 to 1. Then it is overflow cond".

i.e. Cin is diff from Cout bit.

No overflow when we use two diff. sign bit

∴ It is an Overflow

2.

	128	64	32	16	8	4	2	1	R
+127	0	1	1	1	1	1	1	1	for 2's comp. from Right side uptide the 1st as it is like here it's 1 & take comp. from before Right
-127	1	0	0	0	0	0	0	1	
+2	0	0	0	0	0	0	1	0	
-2	1	1	1	1	1	1	1	0	

Jb tak Phla 1 ni dikhta hai tab tak as it is likhenge then take 1's comp from Right side to left side.

C_{in}

-127 : 1 0 0 0 0 0 0 1
-2 : 1 1 1 1 1 1 1 0
-129 : 1 1 1 1 1 1 1 1

C_{out} discard it.

$\therefore C_{in} = 1$, $C_{out} = 0$ → Overflow

Register Transfer Language (RTL)

Register - A storage device which comprises of set of flip flops. Flip flops stores 1 bit at a time.

RTL is a description of a computer process.

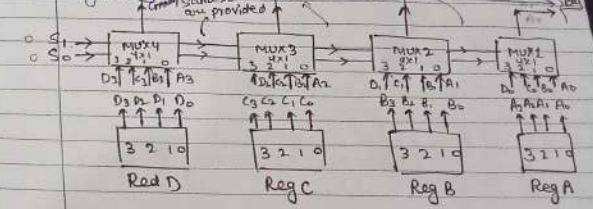
Register transfer in a Common Bus Configuration

There are two mechanism/approach to implement the reg. trans. in a common bus config :—

1) MUX 2) Tristate Buffers

1) With the help of Multiplexer (MUX) :—

- The length of register is identical to the no. of multiplexers.
- The size of multiplexer is identical to the no. of registers.

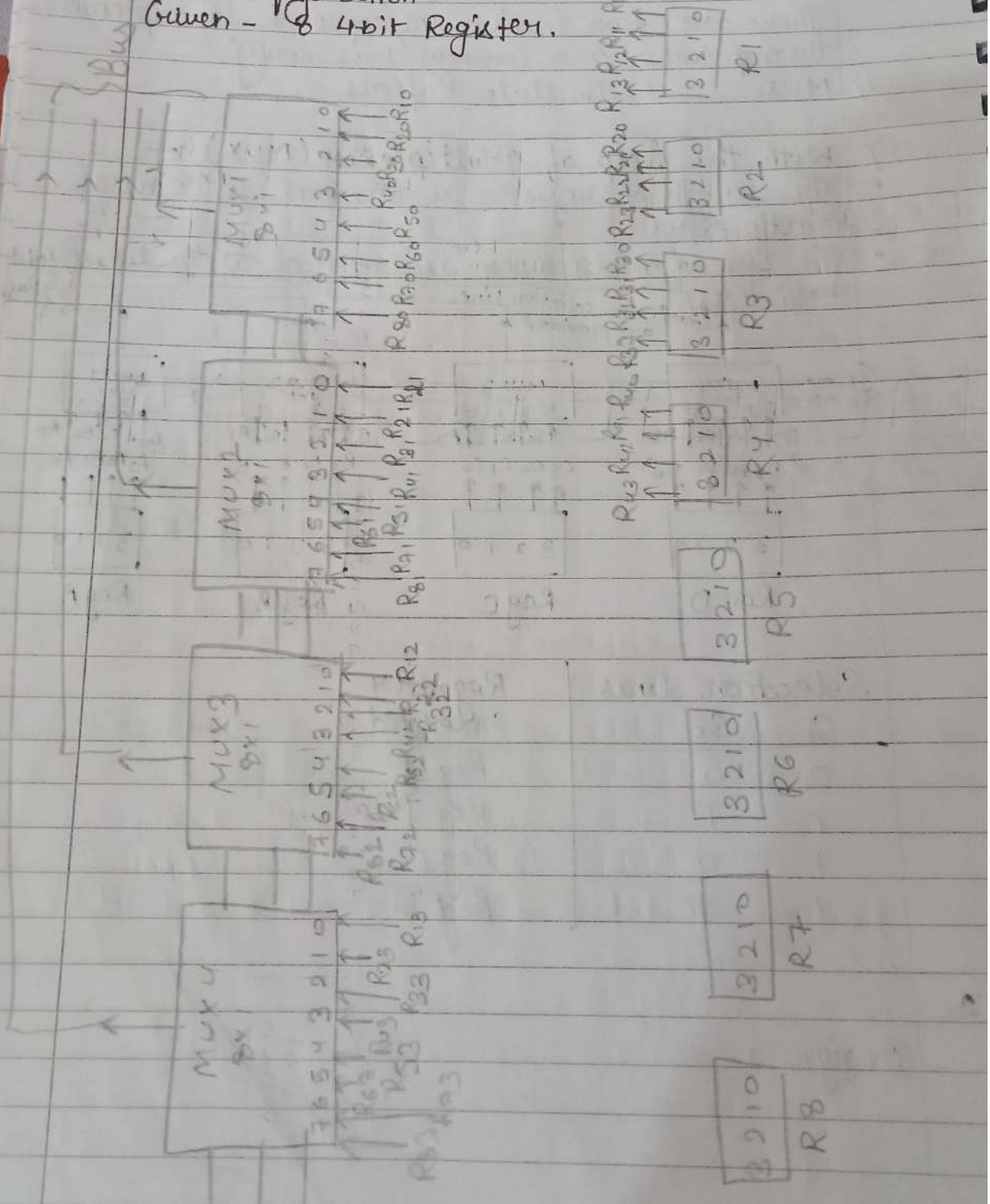


Selection lines		Register selected
S_1	S_0	
0	0	Reg A
0	1	Reg B
1	0	Reg C
1	1	Reg D

2) Design a

Date _/_/_

Q. Design a reg. transfer operation in a common bus configuration.
Given - 8 4bit Register.

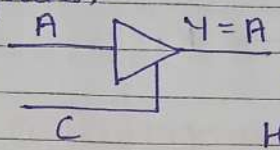


Date _/_/_

2) With the help of Tri state Buffer :-

State 1 → '0'
state 2 → '1'
state 3 → 'High Impedence'

Buffer is a logic gate which provide the exact replica of inputs.



Y = A, if C = 1
High Imped. if C = 0
↳ I/P is disconnected with the O/P.
(No effect)

{For normal functioning, C = 1}

Truth table

Inputs of decoders | Reg. select

S_1	S_0	
0	0	A
0	1	B
1	0	C
1	1	D

Memory Transfer

Date _/_/_

Read Operation

Read : $R_1 \leftarrow M[AR]$

(Controlling Singnal; M → Memory; AR → Address)

Write Operation

Write : $M[AR] \leftarrow R_1$

Types of Microoperation

→ Arithmetic microoperation
→ Logical microoperation
→ Shift ,,

1. Arithmetic microoperation

Microoperation	Notation
Add	$R_3 \leftarrow R_1 + R_2$
Subtract	$R_3 \leftarrow R_1 + \overline{R_2} + 1$ (2's complement)
Add with carry	$R_3 \leftarrow R_1 + R_2 + 1$
Subtract with borrow	$R_3 \leftarrow R_1 + \overline{R_2}$ $\{R_1 + \overline{R_2} + 1 - 1\}$
Transfer	$R_2 \leftarrow R_1$
Increment	$R_2 \leftarrow R_1 + 1$
Decrement	$R_2 \leftarrow R_1 - 1$ (2's complement) $(R_1 + \overline{1} + 1 = R_1 + 0 + 1)$

Date _/_/_

2. Logical microoperation

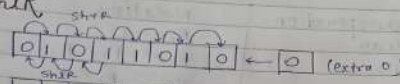
Microoperation	Notation
Clear	$R_1 \leftarrow 0$
AND	$R_3 \leftarrow R_1 \wedge R_2$
	$R_3 \leftarrow \overline{R_1} \wedge R_2$
	$R_3 \leftarrow R_1 \wedge \overline{R_2}$
NAND	$R_3 \leftarrow \overline{R_1 \wedge R_2}$
OR	$R_3 \leftarrow R_1 \vee R_2$
	$R_3 \leftarrow \overline{R_1} \vee R_2$
	$R_3 \leftarrow R_1 \vee \overline{R_2}$
NOR	$R_3 \leftarrow \overline{R_1 \vee R_2}$
EXOR	$R_3 \leftarrow R_1 \oplus R_2$
EXNOR	$R_3 \leftarrow \overline{R_1 \oplus R_2}$
Complement	$R_2 \leftarrow \overline{R_1}$
Set to 1	$R_3 \leftarrow 1$

3. Shifting microoperation

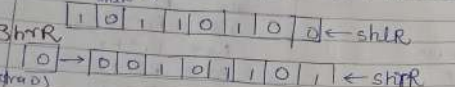
Microoperation	Notation
Shift left	R ← shlR
Shift right	R ← shrR
Circular left shift	R ← cilR
Circular right shift	R ← cirR
Arithmetic shift left	R ← ashlR
Arithmetic shift right	R ← ashrR

8-bit Register

1) shlR

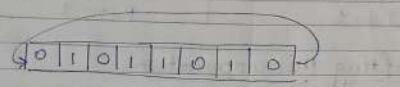


2) shrR

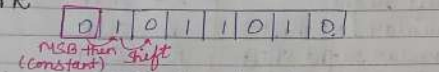


3) cilR

4) cirR



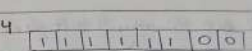
5) ashrR (divide by 2^n)



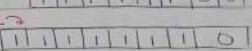
In this case, MSB doesn't change.
Now, shift all the bits starting from MSB.

{without changing MSB, just apply shrR}

Ex- -4



$\frac{-4}{2} = -2$



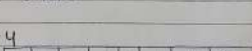
$\frac{-2}{2} \rightarrow -1$

$\frac{-4}{2} = -2$

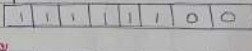
6) ashlR (Multiply by 2^n)

This microoperation is identical to shlR.

-4



$-4 \times 2 = -8$

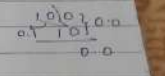


$-4 \times 2 = -8$

Binary Multiplication

M (Multiplicant) → 100 (4)
X 8 (Multiplier) → X 10 (2)

00 0-0
100
1000



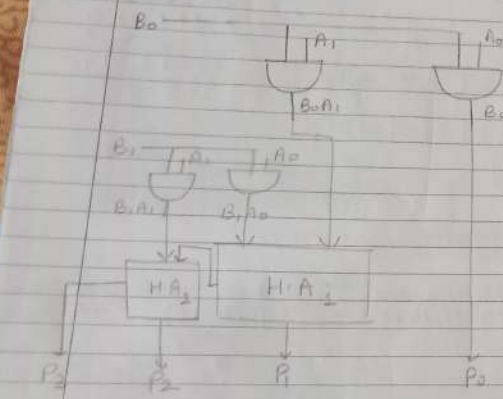
Designing of 2-Bit Binary Multiplier

A (Multiplicant) $A_1\ A_0$ (2)
× B (Multiplier) $B_1\ B_0$ (2) 2+2 = 4

Partial product: $B_0.A_1\ \ B_0.A_0$; $2^4 = 16$, 0–15 max.
$B_1.A_1\ \ B_1.A_0$

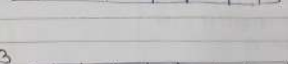
Extra carry – $P_3\ \ P_2\ \ P_1\ \ P_0$ (~~$B_0.A_0$~~)

final product



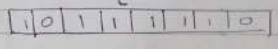
Q. -65

Q. -65 in 8-bit
+65 → 01000001
-65 → 10111111



Overflow occurs.

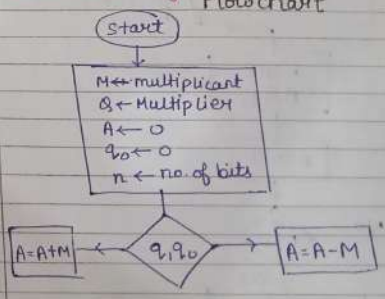
Q. -33



-33×2 = -66

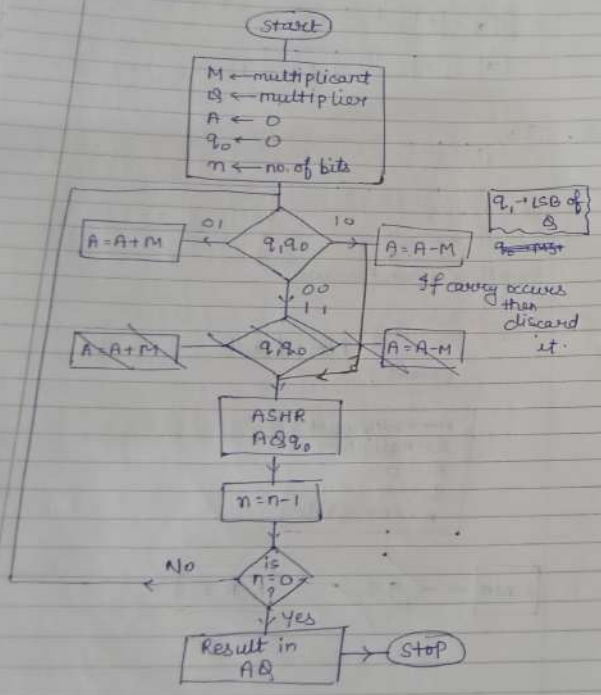
Overflow doesn't occur

Booth's Algo Flowchart



Booth's Algo

Flowchart



Q. Perform binary multiplication : m = -10 and 2.

Take an extra bit in binary representation

M = -10 ⟶ 10110
Q = +2 ⟶ 00010
A ⟶ 00000
q_0 ⟶ 0

n	A	Q	q_0	Action
5	00000	00010	0	Initialisation
4	00000	00001	0	ASHR {AQq_0 applied}
	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

00101
10110
11011

Result :- 1111101100

Q. Perform the binary multiplication of $-16 \times (-5)$ by using booth algo.

$-16 \rightarrow 10000$

$-5 \rightarrow 11011$

A = A - M

A - A - M

010000

000000
- 11 0000
00 1111
01 0000
00 0000
01 0000

$M = -16 \rightarrow 110000$

$Q = -5 \rightarrow 11101$

$A \rightarrow 000000$

$q_0 \rightarrow 0$

n	A	Q	q_0	Action
6	000000	11101	0	Initialisation
5	010000	11101	0	A = A - M
	001000	01110	1	ASHR
4	000100	00111	0	ASHR
	110100	00111	0	A = A + M
3	111010	00011	1	ASHR
	001010	00011	1	A = A - M
2	000101	00001	1	ASHR
1	000010	10000	1	ASHR
0	000001	01000	0	ASHR

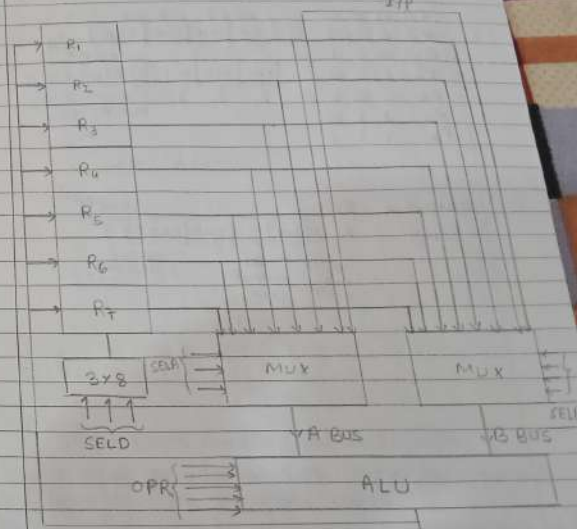
Result :- 000001010000

-16×-5

$= 80$

General Register Organization (GRO)

for 7 Reg. Min. Sel. line = 3



14 bits

3	3	3	5
SELA	SELB	SELD	OPR

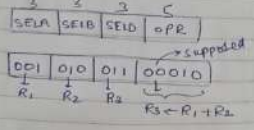
Control Word :-

max. no. of micro-operations performed = 2^5

Control word generated from control unit to perform the operation $R_3 \leftarrow R_1 + R_2$

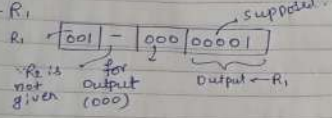
Ex- $R_3 \leftarrow R_1 + R_2$

	0 0 0
R_1	0 0 1
R_2	0 1 0
R_3	0 1 1
R_4	1 0 0
R_5	1 0 1
R_6	1 1 0
R_7	1 1 1



Ex Output ← R_1

Same



Stack Organization

→ Useful Property in CPU.

→ Based on the property of LIFO (Last In First Out)

32 MB

$32 \times 2^{20} \times 8$ bits

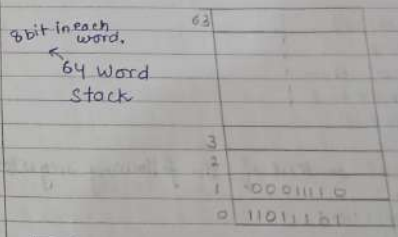
$2^5 \wedge 2^{20}$

2^{25}

Locations / Address

8 bit in each word.

64 Word Stack



SP → Stack Pointer (Address Register)

it holds the address of the top of the element.

Here 1 is the top of the stack.

If there is 63 then only 6 bit is required

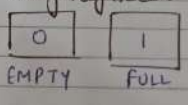
000001

Flag Register It is always of 1-bit.

Shows the availability or unavailability of the content of register.

1 → availability (Stack full)

0 → unavailability. (Stack empty)



PUSH OPERATION

SP←0, EMPTY ← 1, FULL ← 0
SP ← SP + 1
M[SP] ← DR
If (SP=0) then
EMPTY ← 0
FULL ← 1

{DR → Data Register}

POP OPERATION

DR ← M[SP]
SP ← SP − 1
If (SP=0) then
EMPTY ← 1
FULL ← 0

The initial content of the following registers is given by :
$R_1 = 10111011$
$R_2 = 00111101$
$R_3 = 10101100$
Perform the following micro-operation and write the final content of each register i.e R_1, R_2 & R_3

Q.
(i) $R_2 \leftarrow \overline{R_1} + 1$
(ii) $R_3 \leftarrow R_2 - 1$
(iii) $R_1 \leftarrow \overline{R_1} \vee R_2$
(iv) $R_3 \leftarrow ashR_3$

$R_2 \leftarrow \overline{R_1} + 1$

i) $R_1 = 10111011$
$\overline{R_1} = 01000100$
$+ \quad 1$
01000101

$R_2 \leftarrow 01000101$

ii) $R_3 \leftarrow R_2 - 1$
$R_2 \rightarrow 01000101$
$1 \rightarrow 00000001$
$-1 \rightarrow 11111110$
$+ \quad 1$
11111111
$R_3 \rightarrow 01000101$
$+ \quad 11111111$
$\textcircled{1}\,01000100$

$R_3 \leftarrow 01000100$

iii) $R_1 \leftarrow \overline{R_1} \vee R_2$
$R_1 \rightarrow 10111011$
$\overline{R_1} \rightarrow 01000100$
$R_2 \rightarrow 01000101$
$R_1 \leftarrow 01000101$

iv) $R_3 \leftarrow ashrR_3$
$R_3 \leftarrow 01000100$
ashr $R_3 \leftarrow 00100010$

Notations for Writing Arithmetic Operation

1. Infix – A*B + C*D
2. Prefix (Polish Notation) – +*AB*CD
3. Postfix (Reverse Polished Notation) – AB*CD*+

Evaluation of Arithmetic Expression using Stack by following Reverse Polish Notation or Post fix notation :

① A * B + C * D

Sol. Postfix
AB* CD* +

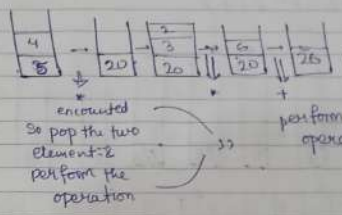
S-1 Just consider the variable sized stack.

S-2 Whenever the operand is encountered just push the stack.

S-3 Now Pop out the operand whether operator is encountered & the stack will perform the operation.

Ex- 5*4 + 3*2
20 + 6 = 26

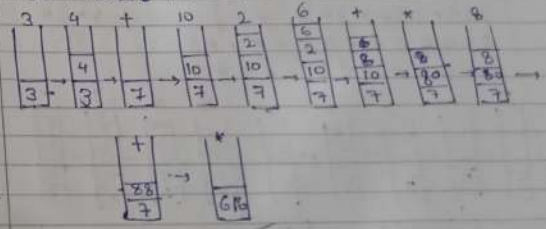
Postfix – AB* CD* +
54* 32* +



Q2. (3+4)[10*(2+6)+8]

Apply Postfix (BODMAS)
(34+)[1026+*+8)
(34+)[1026+*8+]
34+1026+*8+*

Rough –
7[10*8+8]
→[80+8]
88×7
88/616
616



Date _ / _ / _

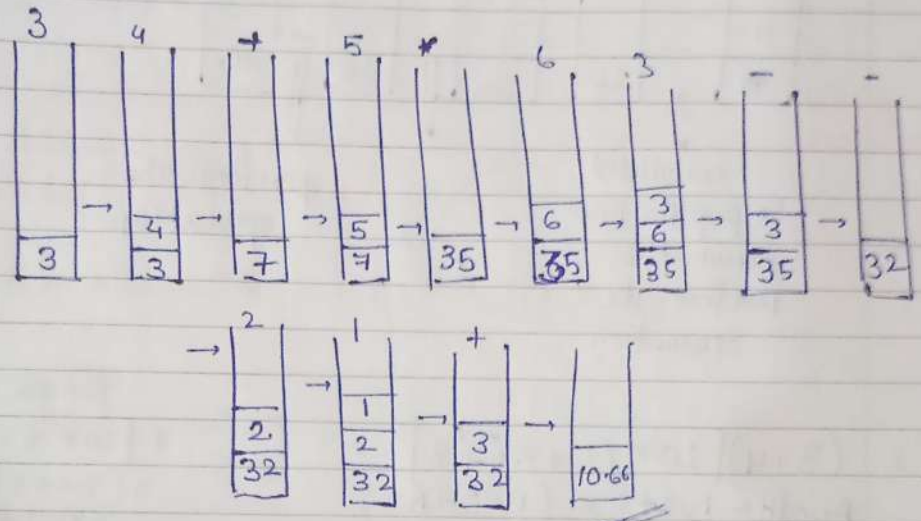
Q3. $\frac{(3+4)*5-(6-3)}{(2+1)}$

$\frac{7*5-(3)}{3}$

$\frac{35-3}{3} = \frac{32}{3} = 10.66$

~~(3 4+) 5*~~ $\frac{(34+5*)-(63-)}{21+}$

34+5*63--21+/



Q4. 3+4*[5*6+7*(9-7)]

3+4*[56*+79 7-*]

3+4*[56*79 7-*+]

3+456*79 7-*+*

3456*797-*+*+

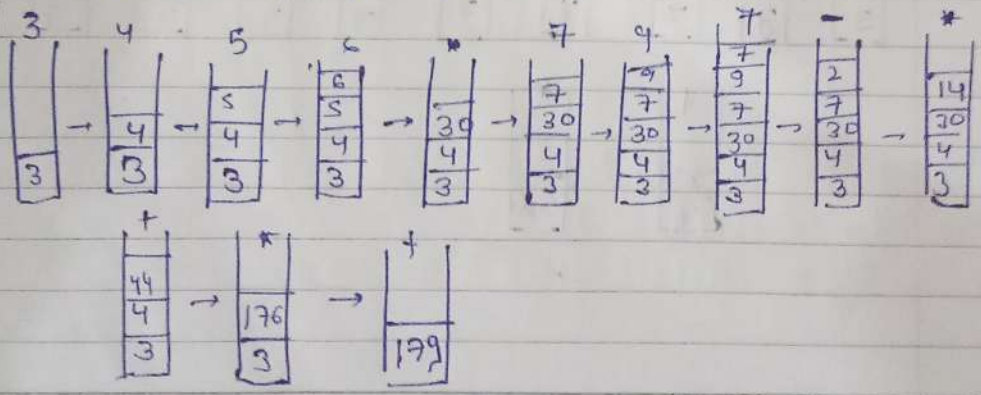
7*[30+7*2]

7*[30+14]

7*44

308

44 / 308 (2)



44 / 176 (1)