Why study Computer Organization and Architecture?

- Every programmer needs to have good knowledge of the subject of computer architecture and organization, which give them an insight of how computer executes program internally and can help them to write more effective programs.
- It also helps to understand why a program exhibits a poor performance or to implement a computionally efficient algorithm.
- For system programmers a good knowledge of computer organisation is essential because they need to program hardware without the support of operating system.

Computer Architecture

- The architecture of computer is concerned with the structure and behaviour of the computer as viewed by a user such as assembly or machine language programmer.

Computer Organization

- It is the way the hardware components operate and the way they are connected together to form computer system.
- A knowledge of computer organization can help to under-

Stand the internal operations that are carried out by a computer while a program is being executed.

- Computer organization deals with how the different components of a computer such as processor, memory or peripheral devices are interconnected and the roles that the different components play during program execution.

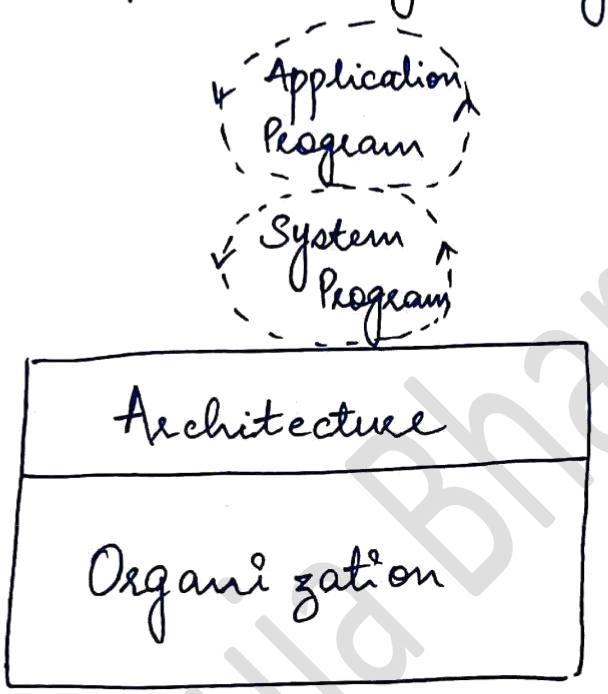
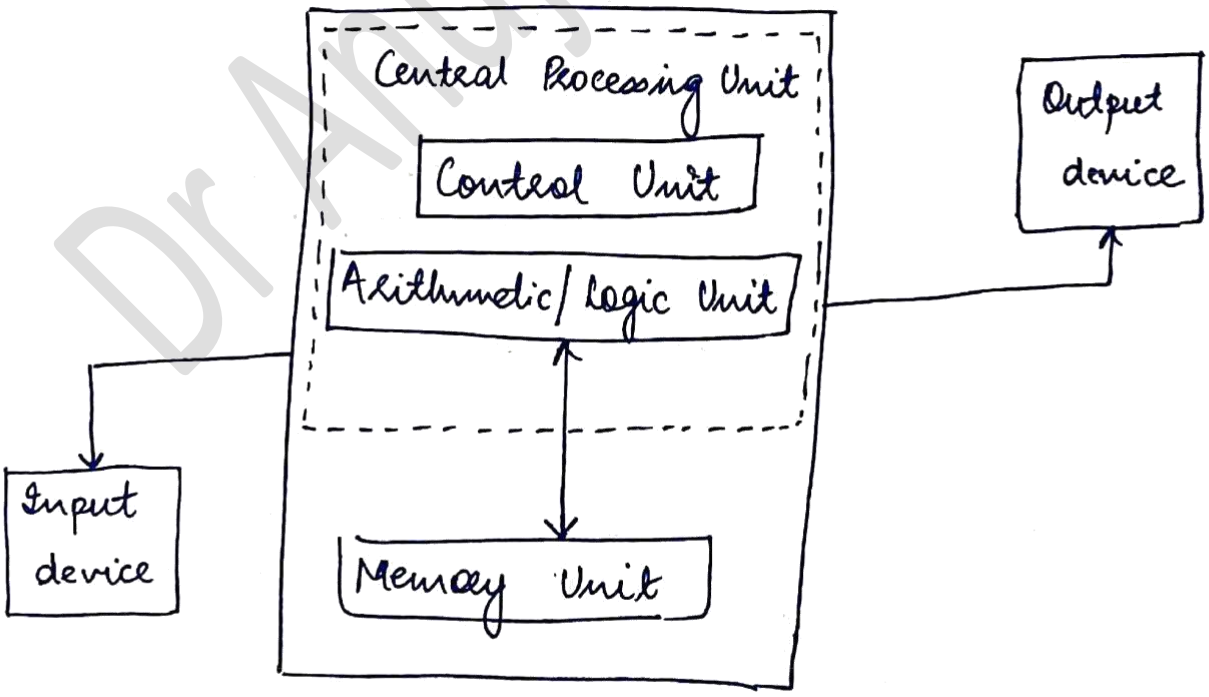


Fig: Organization implements the architecture

- The system program directly interact with computer hardware. therefore written for specific computer architecture.
- The application programs invoke the services offered by system programs.
- The computer architecture is defined as interface between hardware and software.

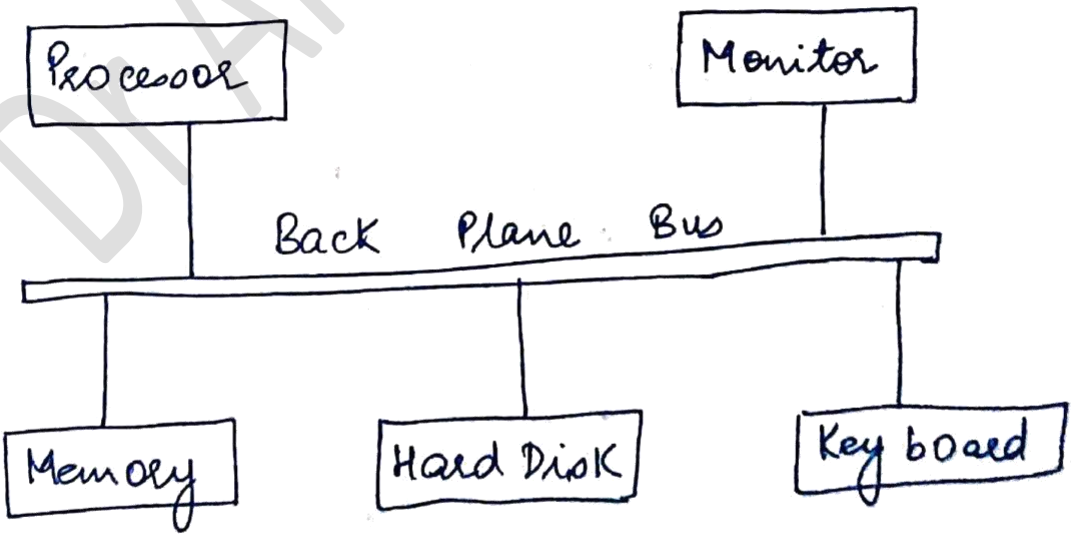
Von Neumann Architecture :-

- In the very early days of computing, one instruction at a time of a program was executed. The operator set up instructions by flipping some switches.
- Executing a program require a lot of effort by operator.
- At this point, the concept of stored programs computers was proposed by John Von Neumann in 1945.
- Von Neumann proposed that instructions can be encoded and stored in the memory like data.
- During the execution of program, the stored instructions can be fetched from memory and the fetched instruction can be decoded.



- The control unit is an electronic circuit that decodes each fetch instruction and generates necessary control signals
- The stored program concept with control unit, eliminate the need for an operator to enter each instruction
- Von Neumann computer organization not only made program execution much faster but also avoid the manual entry of instruction. This results in instruction cycle.
- In instruction cycle, an instruction is fetch from memory decode and executes.
- Drawback - a single connection exist between processor and memory. i.e at any time only one memory access can occur.

Basic Organization of Computer



Processor

→ The processor or CPU is responsible for fetching an instruction stored in the memory & executing it

→ It contain arithmetic & logic unit for manipulating data, number of registers for storing data and control unit.

Main Memory

→ It is also called random access memory because CPU can access any location in the memory at random.

Monitor

→ It is visual display unit and serves as primary output unit.

Keyboard

→ It is the primary input devices in the modern computers.

Peripheral devices

→ It can be attached to backplane bus.
Eg printer, scanner, speakers.

Backplane bus

→ It is group of wires as control & address wires.

→ Control wires carry control signals to different units.

→ The address wires carry the address of the specific data in memory and data wires are used to carry the data.

Number System → It is the technique to represent number in computer system architecture.

Decimal : Having base 10.

eg 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Binary : Having base 2.

eg 0, 1

Octal : Having base 8.

eg 0, 1, 2, 3, 4, 5, 6, 7

Hexadecimal : Having base 16.

eg 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

(A) Decimal to Binary

eg 17

2	17	1
2	8	0
2	4	0
2	2	0
	1	

Binary Number

10001

eg 55.32

2	55	
2	27	1
2	13	1
2	6	1
2	3	0
	1	1

$0.32 \times 2 = 0.64 \quad 0$

$0.64 \times 2 = 1.28 \quad 1$

$0.28 \times 2 = 0.56 \quad 0$

$0.56 \times 2 = 1.12 \quad 1$

$.12 \times 2 = 0.24 \quad 0$

Binary Number $(110111 \cdot 01010)_2$

(B) Binary to Decimal

eg 101100101

$1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 256 + 64 + 32 + 4 + 1$

$= (357)_{10}$

eg 1101.101

$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

$= 8 + 4 + 1$

$= 13$

$= 0.5 + 0.125$

$= 0.625$

Decimal Number $(13.625)_{10}$

(C) Decimal to Octal

eg 33

8	33	1
	4	

$= (41)_8$

eg 45.26

8	45	
	5	5

$0.26 \times 8 = 2.08 \quad 2$

$0.08 \times 8 = 0.48 \quad 0$

$0.48 \times 8 = 3.84 \quad 3$

(D) Octal to Decimal

eg $(253.64)_8$

$= 2 \times 8^2 + 5 \times 8^1 + 3 \times 8^0 + 6 \times 8^{-1} + 4 \times 8^{-2}$

$= 128 + 40 + 3 + 0.75 + 0.0625$

$= (171.8125)_{10}$

(E) Decimal to Hexadecimal

eg 455

16	455	
16	28	7
	1	12 (C)

$(1C7)_{16}$

(F) Hexadecimal to Decimal

eg 54. D2

$= 5 \times 16^{1} + 4 \times 16^{0} + 13. \times 16^{-1} + 2 \times 16^{-2}$

$= 80 + 4 + 0.8125 + 0.0078125$

$= 84.8203125$

few more examples

eg $(6AB.CD)_{16} = (?)_{8}$

Binary Coded Octal

- Three binary digits also known as binary coded octal can be coded to form one octal digit.

0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7

eg Convert $(742)_8$ to binary.

7 4 2

$(111\ 100\ 010)_2$

eg Convert $(10100110)_2$ to Octal.

010 100 110

$(2\ 4\ 6)_8$

Binary Coded Hexadecimal

Four bit Group	Decimal Digit	Hexadecimal digit
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

eg Convert $(3D9)_{16}$ to binary.

3 D 9

0011 1101 1001

$= (111101001)_2$

eg Convert $(110100110)_2$ to hexadecimal.

0001 1010 0110

$(1\ A\ 6)_{16}$

eg Convert $(E8A)_{16}$ to Octal

E 8 A

1110 1000 1010

111 010 001 010

$(7\ 2\ 1\ 2)_8$

eg Convert $(3756)_8$ to Hexadecimal.

3 7 5 6

011 111 101 110

0111 1110 1110

$(7\ E\ E)_{16}$

Complements :-

- Complements are used in digital computers for simplifying the subtraction process and for logical manipulation.
- There are two types of complements of each base r system.
  - (i) r's complement
  - (ii) (r-1)'s complement

→ When r = 10, 10's and 9's complement.

→ When r = 2, 2's and 1's complement.

- Generally,

$$11\,00\,11 \xrightarrow{1\text{'s comp.}} 00\,11\,00 \xrightarrow{2\text{'s comp.}} 00\,11\,01$$

$$11\,000 \xrightarrow{1\text{'s comp.}} 00\,111 \xrightarrow{2\text{'s comp.}} 01000$$

→ 1's Complement Subtraction

- Subtraction of binary numbers using the 1's complement method allow subtraction only by addition.
- Steps
  - (i) Determine the 1's complement of the smaller number
  - (ii) Add this to the larger number.
  - (iii) Remove the carry and add it to the result.

Note: This carry is called end around carry.

eg Subtract $(1010)_2$ from $(1111)_2$ using the 1's complement method. Also subtract using direct method.

Soln:

Direct Subtraction

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

1's Complement Method

```
    1 1 1 1
1's Complement → 0 1 0 1
-----------------------
Carry →   1 0 1 0 0
                  1
Add Carry →  -------
             0 1 0 1
             -------
```

eg Subtract $(1010)_2$ from $(1000)_2$ using 1's complement method

Soln

Direct Subtraction

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

1's Complement Method

```
  1000
  0101
-------
  1101
    ↓ 1's comp
- 0 0 1 0
```

Note :- Subtraction of larger number from smaller number then answer is the 1's complement of result and is opposite in sign.

** The 1's complement method is particularly useful in arithmetic logic circuits because subtraction can be done with the help of adder.

→ 2's Complement Subtraction :-

Steps

(i) Determine the 2's complement of smaller number

(ii) Add this to the larger number.

(iii) Omit the carry. (Always carry in this case)

eg Subtract $(1010)_2$ from $(1111)_2$ using 2's complement. Subtract by direct method also.

Soln

Direct Subtraction

$$\begin{array}{r} 1\ 1\ 1\ 1 \\ 1\ 0\ 1\ 0 \\ \hline 0\ 1\ 0\ 1 \end{array}$$

2's Complement

$$\begin{array}{r} -1\ 1\ 1\ 1 \\ \text{2's comp} \rightarrow\ 0\ 1\ 1\ 0 \\ \hline \text{Carry} \rightarrow\ 1\ 0\ 1\ 0\ 1 \end{array}$$

∴ Carry is discarded.

∴ Answer is $(0101)_2$.

eg Subtract $(1010)_2$ from $(1000)_2$ using 2's compl.

Soln

Direct Subtraction

$$\begin{array}{r} 1000 \\ -\ 1010 \\ \hline 0010 \end{array}$$

2's comp.

$$\begin{array}{r} 1\ 0\ 0\ 0 \\ \text{2's comp} \rightarrow\ 0\ 1\ 1\ 0 \\ \hline \text{No carry} \rightarrow\ 1\ 1\ 1\ 0 \end{array}$$

∴ No carry is obtained.

∴ Difference is negative and answer is 2's complement of (1110) i.e $(0010)_2$

Signed Binary Number Representation

- In the sign-magnitude representation, a number is represented in its binary form as
  - → The most significant bit (MSB) represents the sign.
  - → 1 in the MSB denotes a negative number.
    0 in the MSB denotes a positive number.
  - → The remaining bits represent magnitude of the number.

eg

8 bit binary Representation	Sign	Magnitude
+18	0	0010010
-18	1	0010010

1's complement Representation

eg $-33 = ?$

$33 \rightarrow (100001)_2$

for 8 bit notation,

$+33 \rightarrow 0\ 0100001$

Sign Magnitude

$-33 =$ 1's complement of $(+33)$

$=$ 1's compl. of $(00100001)_2$

$= (11011110)_2$

2's Complement Representation

eg (−33) = ?

33 → $(100001)_2$

8 bit → $(0\ 0\ 1\ 0\ 0\ 0\ 0\ 1)_2$

−33 = 2's complement of (+33)

= 2's complement of $(00100001)_2$

= $(11011111)_2$

Exercise

Q Convert the following numbers to sign magnitude, 1's and 2's representation.

		Sign Magnitude	1's repres^n	2's repres^n
(A)	+26			
	+26	0 0011010	0 0011010	0 0011010
(B)	−37	1 0100101	0 1011010	0 1011011
(C)	−101	1 1100101	0 0011010	0 0011011

Addition in 2's Complement System

Case I. Two Positive Numbers

+29	→	0	0011101
+19	→	0	0010011
+48		0	0110000
		Sign	Magnitude.

Sign → 0 : When sum is positive they have the same number of bits.

Case 2. Positive and Negative Numbers

(−22) → ?

+22 → 0 0010110

−22 → 2's Complement of (+22)

= 1 1101010

+39	→	0	0100111
−22	→	1	1101010
+17		1 (Carry)	0 0010001 (Sign, Magnitude)

∴ Carry is omitted.

∴ Answer is $(0001\ 0001)_2$

Case 3 Negative and Positive Number

$-47 \rightarrow$ 1 1 0 1 0 0 0 1

$+29 \rightarrow$ 0 0 0 1 1 1 0 1

-18 1 1 1 0 1 1 1 0

Sign

$+47 \rightarrow 0\ 0101111$

$-47 \downarrow$ 2's Comp

$1\ 1010001$

∴ Sign bit → 1 i.e Negative Number

True Sum = 2's complement of Magnitude

= 2's complement (1 1 0 1 1 1 0)

= 0 0 1 0 0 1 0

Result is $(1\ 0\ 0\ 1\ 0\ 0\ 1\ 0)_2$

Case 4 Two Negative Number

$-32 \rightarrow$ 1 1 1 0 0 0 0 0

$-44 \rightarrow$ 1 1 0 1 0 1 0 0

-76 1 1 0 1 1 0 1 0 0

Carry Sign

$+32 \rightarrow 0\ 0100000$

$\downarrow$ 2's

$-32 \rightarrow 1\ 1100000$

$+44 \rightarrow 0\ 0101100$

$\downarrow$ 2's

$-44 \rightarrow 1\ 1010100$

∴ Carry is discarded. ∴ True Sum = 2's Comp. of Mag.

= 2's Comp (0 1 1 0 1 0 0)

∴ Result is $(1\ 0\ 0\ 1\ 1\ 0\ 0)_2$ = 1 0 0 1 1 0 0

Note :- 2's complement addition is with in the range -128 to $+127$. Otherwise, we get an overflow.

Subtraction in 2's Complement

Case 1 Both numbers are positive $28-19=9$

$+28 \quad 0 \quad 0011100$

$+19 \quad 0 \quad 0010011$

To subtract +19 from +28,

i.e 28 - 19 (∵ +19 → 2's comp)

$+28 \rightarrow \quad 0 \quad 0011100$

$-19 \rightarrow \quad 1 \quad 1101101$

$+9 \quad 1 \; 0 \quad 0 \; 0 \; 0 \; 1 \; 0 \; 0 \; 1$ (9)

Carry Sign

∵ Omit the Carry ∴ Result is $(00001001)_2$

Case 2 Positive Number and Smaller Negative Number $39-(-21)=+60$

$+39 \rightarrow \quad 0 \quad 0100111$

$-21 \rightarrow$ 2's comp of (+21)

2's comp. of (00010101)

11101011

The computer sends -21 to a 2's complement circuit to produce

$+21 \rightarrow \quad 0001010\,1$

It then adds + 39 & + 21

+39 → 0010 0111

+21 → 0001 0101

+60 → 0011 1100 → 60

(Case 3) Positive Number and Larger Negative Number

In the 2's complement system 19 - (-43) = 62

+19 → 0 0010011

-43 → 2's comp of (+43)

2's comp. of (00101011)

→ 11010101

The computer sends 2's complement of -43

i.e +43 → 0010 1011

+19 → 0 0010011

+43 → 0 0101011

+62 → 0 0111110 → 62

Case 4 Both Numbers are Negative

In 2's Complement System

$-57 \rightarrow 1100\,0111$

$-33 \rightarrow 1101\,1111$

$\therefore -57 + (-33)$

$\therefore$ Take 2's complement of -33

$+33 \rightarrow 0010\,0001$

Then add $+33$ to -57

$-57 \rightarrow 1100\,0111$

$+33 \rightarrow 0010\,0001$

$-90 \quad 1\,1001000$

Sign magnitude

↓ 2's

So add $(1001\,1000)_2$

Arithmetic Overflow :-

- When number of bits in the sum exceeds the number of bits of the number added, overflow results.

Case 1 Two Positive Numbers

$$\begin{array}{rcl} +120 & \longrightarrow & 0\ 111\ 1000 \\ +\ 65 & \longrightarrow & 0\ 100\ 0001 \\ \hline 185 & & \hline 1011\ 1001 \end{array}$$

↑ Negative, ans. is not correct.

Case 2 Two Negative Numbers

$$\begin{array}{rcl} -77 & \longrightarrow & 1011\ 0011 \\ -122 & \longrightarrow & 1000\ 0110 \\ \hline -199 & & \hline 1\ 0011\ 1001 \end{array}$$

↑ Carry ↓ positive ↓ ans. is not correct

Note:- The range of the answer is -128 to $+127$.

- An overflow is a software problem and not a hardware problem. In digital computers, an overflow occurs when an operation results in quantity beyond the capacity of storage register. Therfore, a programmer must check the overflow after each addition or subtraction.

Extra Ques

Addition

1. (+23) + (+19) — 00 10 10 10

2. (+23) + (-19) — 1) 00000100 · 10000 100

3. (-23) + (+19) — 10 10 10 10

4. (-23) + (-19)

Subtraction

1. (+23) - (+19) — 1) 00000100 — 00 10 10 10 — 10 10 10 10 — 10000 100

2. (+23) - (-19)

3. (-23) - (+19)

4. (-23 - (-19)

Note :- N- bit two's complement number, one of the 2^N possible values. However, the values are split between positive and negative numbers.

eg 4 bit unsigned number represent 16 (2^4) values i.e 0 to 15.

4 bit two's complement number also represent 16 values i.e -8 to 7.

** In general the range of N- bit two's complement number is $(-2^{N-1}, 2^{N-1}-1)$

Floating Point Representation

- Floating point represent has two parts

$69.68 \rightarrow \underbrace{6.968}_{\text{Mantissa}} \times 10^{1} \rightarrow$ Exponent

- Mantissa - Signed fixed point number, either an integer or fractional number.
- Exponent - Designates the position of the decimal or binary point.

eg The decimal number +6132.789 is represented in floating point with fraction and exponent as follows

Fraction	Exponent	Scientific Notation
+0.6132789	+04	$+0.6132789 \times 10^{+4}$

eg The Binary number +1001.11 is represented in floating point representation with 8-bit fraction and 6-bit exponent as follows

0 1 0 0 1 1 1 0 000100 → $.1001110 \times 2^{4}$

eg Represent the number (+46.5) as floating point representation

$46.5 \rightarrow (101110.1)_2$

$= .1011101 \times 2^{6}$

IEEE 754 floating point representation

- It is having three basic components
  - → Sign (+ve / -ve)
  - → Biased Exponent
  - → Normalized Mantissa
- The exponent field needs to represent both the positive and negative exponent.
- The bias is added to actual exponent to get the stored exponent.
- If size of exponent field is 8-bits then bias value $2^{size-1} - 1 = 127$.

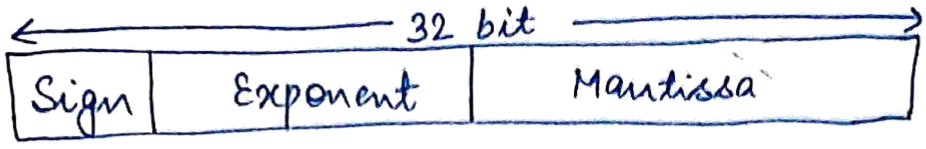
  If size of exponent field is 11-bits then bias value $= 1023$.

Normalized Mantissa

- Mantissa is normalized with only 1 to the left of the decimal.

  1. ----------

Single Precision (IEEE 754 - 32 bit)



1 bit 8 bit 23 bit

Eg 85.125 in IEEE - 32 bit format.

Binary of 85.125 = 1010101.001

$= 1.010101001 \times 2^{+6}$ → Exponent

Mantissa

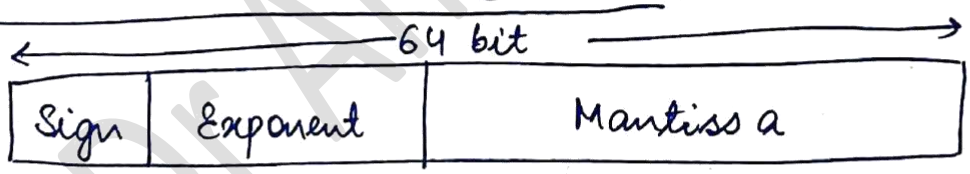
Sign = 0 (∵ +ve)

Biased = 127 + 6 = 133 $\xrightarrow{\text{Binary}}$ 10000101

~~Normalised~~ =

0	10000101	01010100100000000000000

Double Precision (64 bit)



1 bit 11 bit 52 bit

Biased = 1023 + 6 = 1029 $\xrightarrow{\text{Binary}}$ 10000000101

0	10000000101	0101010010000000000000000000000000000000000000000000

Eg Represent 100.125 in IEEE 754 single and double precision.

$100.125 \xrightarrow{\text{Binary}} 1100100.001$

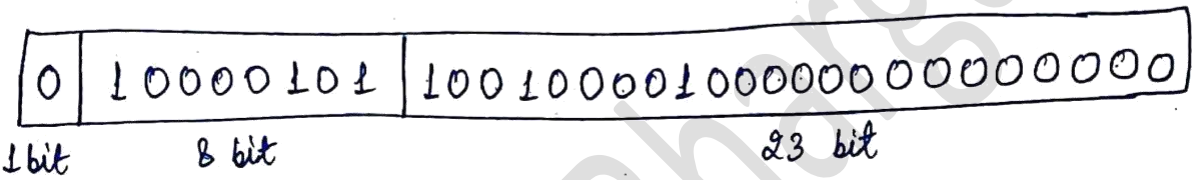
$= 1.100100001 \times 2^{+6}$

Mantissa = 100100001

Exponent = +6

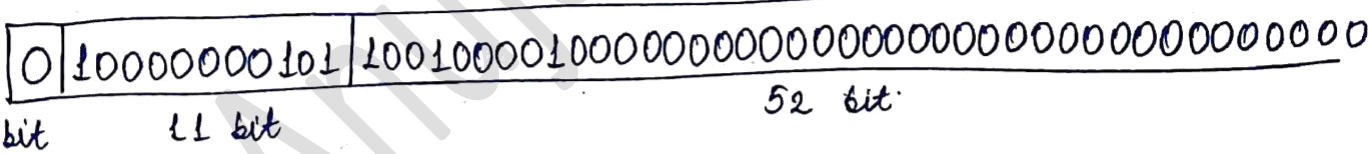
Biased Exponent = 127 + 6 = 133 $\xrightarrow{\text{Binary}}$ 10000101

Single Precision 32 bit



Double Precision

Biased Exponent = 1023 + 6 = 1029 $\xrightarrow{\text{Binary}}$ 10000000101



Eg $(-39.125)_{10}$ by using IEEE 32 bit and 64-bit floating point representation.

$-39.125 \xrightarrow{\text{Binary}} (-100111.001)_2$

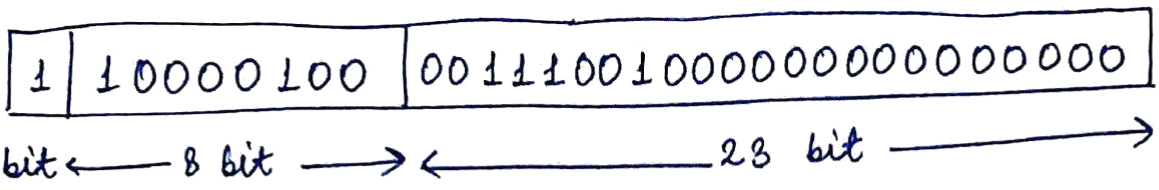
$= -1.00111001 \times 2^{+5}$

Mantissa = 00111001

Exponent = +5

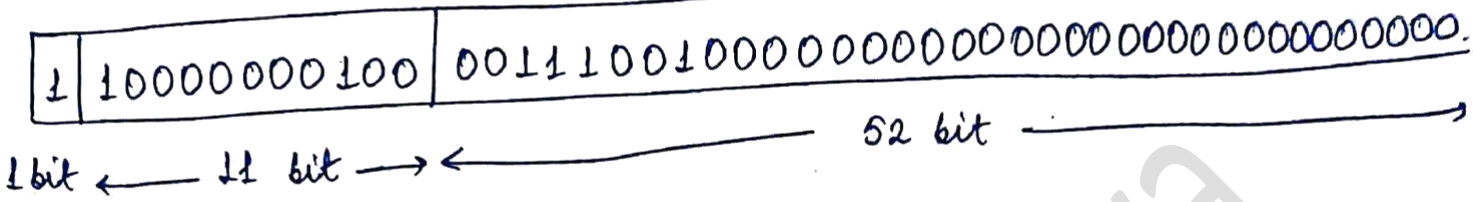
for 32,- bit,

Biased Exponent = 127 + 5 = 132 $\xrightarrow{\text{Binary}}$ 10000100



For 64-bit,

$E = 1023 + 5 = 1028 \xrightarrow{\text{Binary}} 10000000100$



Extra Que

eg Perform $(-36) - (+23)$ by using 8-bit signed 2's complement. Also, verify overflow exist or not.

Sol:- $+36 \longrightarrow 00100100$

$+23 \longrightarrow 00010111$

$-36 \longrightarrow 11011100$

$-(+23) \longrightarrow 11101001$

$11100010 1$

Carry ↑ | Sign | Magnitude

$1000101 \xrightarrow{2\text{'s}} 0111011$

∴ The value of carry in and carry out are same.

Therefore, no overflow exist.

Introduction to Combinational Circuit

- It is a connected arrangement of logic gates with sets of input and output.
- At any given time, binary values of output are function of binary combination of input.

Fig :- Basic Block Diagram of Combinational Circuit

- Applications
  1) In digital computers for generating binary control decisions
  2) for providing digital components required for data processing.
- For designing complex combinational circuit, simple arithmetic circuits are half adder & full adder.

→ Half Adder :-

- Half Adder adds only two input bits, has nothing to do with carry. So, if the input to half adder have a carry then it will add only the two bits. That means addition is not completed so called Half Adder.

Truth Table

Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

K-Map for Sum

A \ B	0	1
0		1
1	1	

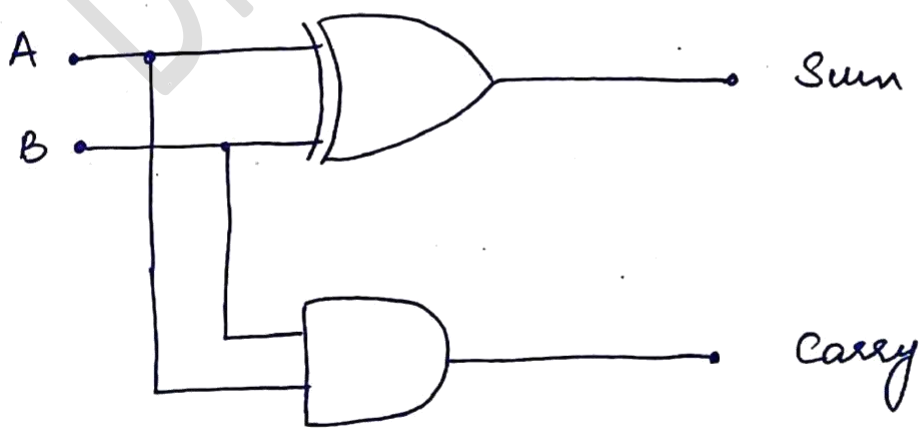
$$Sum = \bar{A}B + A\bar{B}$$

K-Map for Carry

A \ B	0	1
0		
1		1

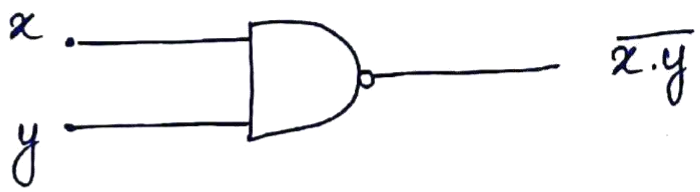
$$Carry = A \cdot B$$

Logic Diagram

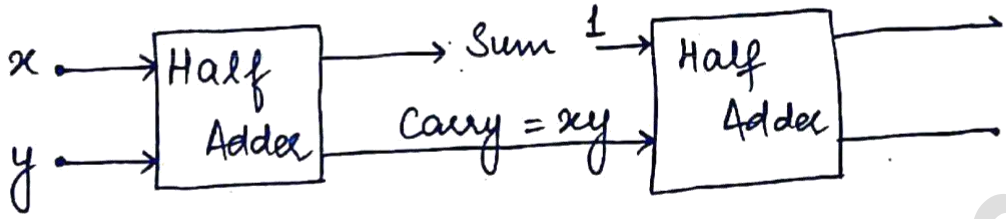


Q Construct universal gates (NAND & NOR) using Half Adder

for NAND Gate,



Using Half Adder,

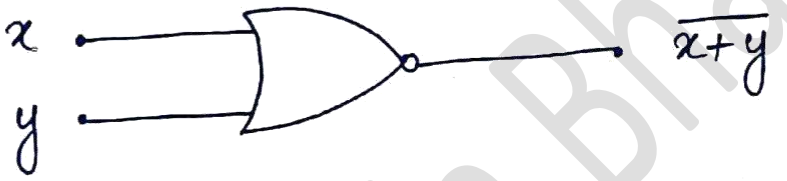


$\text{Sum} = 1 \oplus xy$

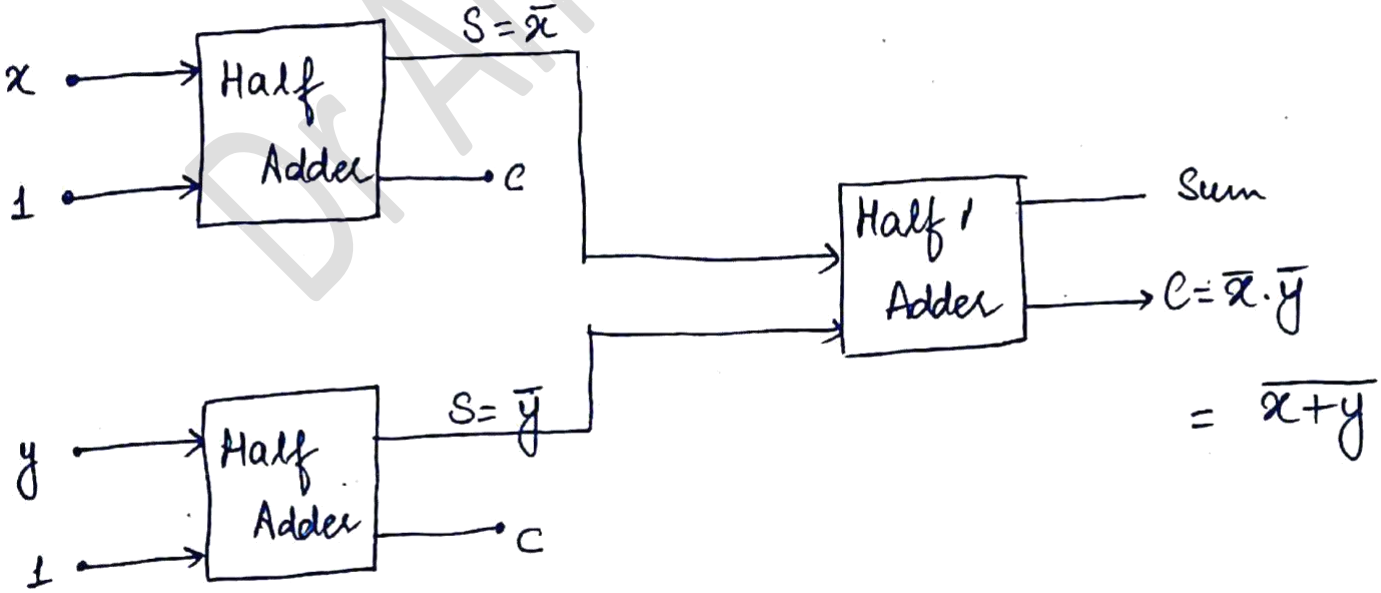
$= \bar{1}xy + 1.\overline{xy}$

$= \overline{x.y}$

for NOR Gate,



Using Half Adder,



$= \overline{x+y}$

→ Full Adder

- Addition of three binary digits.
- Truth Table

Input			Output	
A	B	C_{in}	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

K-Map for Sum

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

$$\text{Sum} = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

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

$$\downarrow \bar{Z} \qquad \downarrow Z$$

$$= A \oplus B \oplus C$$

K- Map for Carry

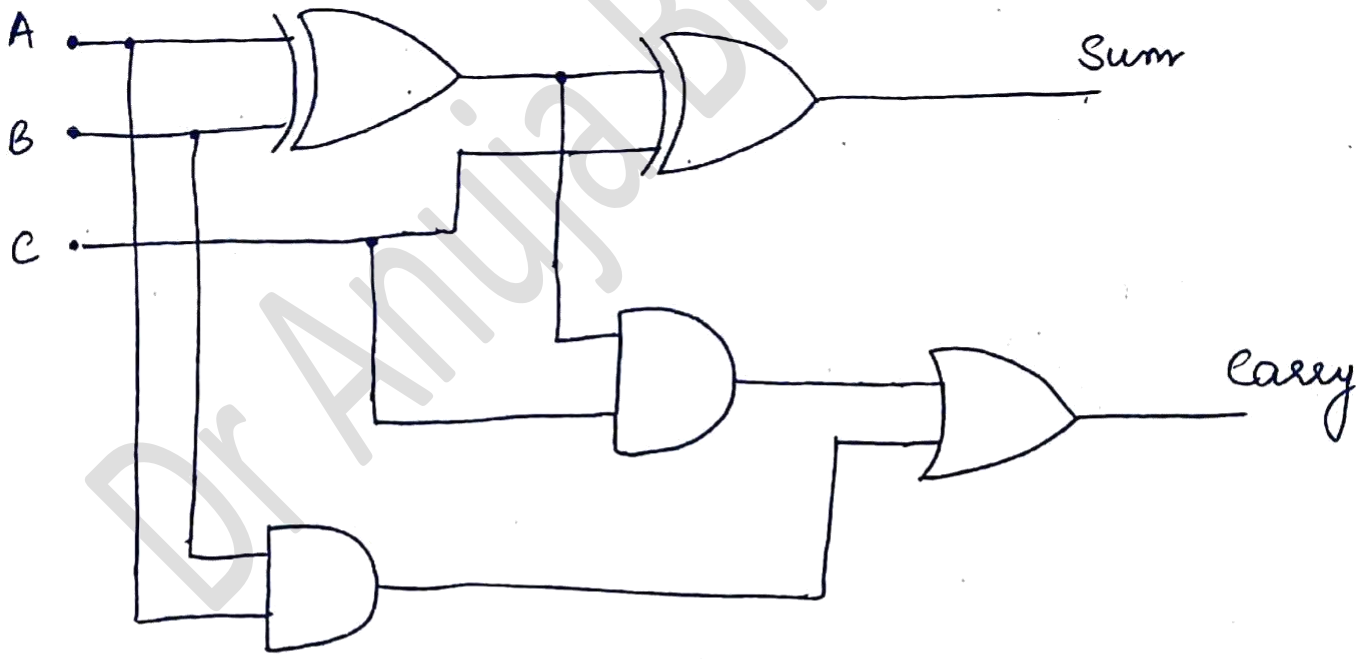
A \ BC	00	01	11	10
0			1	
1		1	1	1

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

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

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

Logic Diagram



Binary Adder

- The digital circuit that generates the arithmetic sum of two binary number of any length is called Binary Adder.

- The binary adder is constructed with full adder circuits connected in cascade with the output carry from one full adder connected to the input carry of the next full adder.

eg

$$\begin{array}{cccc} C_4 & C_3 & C_2 & C_1 \quad C_0 \\ & B_3 & B_2 & B_1 \quad B_0 \\ & A_3 & A_2 & A_1 \quad A_0 \\ \hline S_3 & S_2 & S_1 & S_0 \end{array}$$

- Fig. shows the interconnections of four full adder to provide a four bit binary adder.

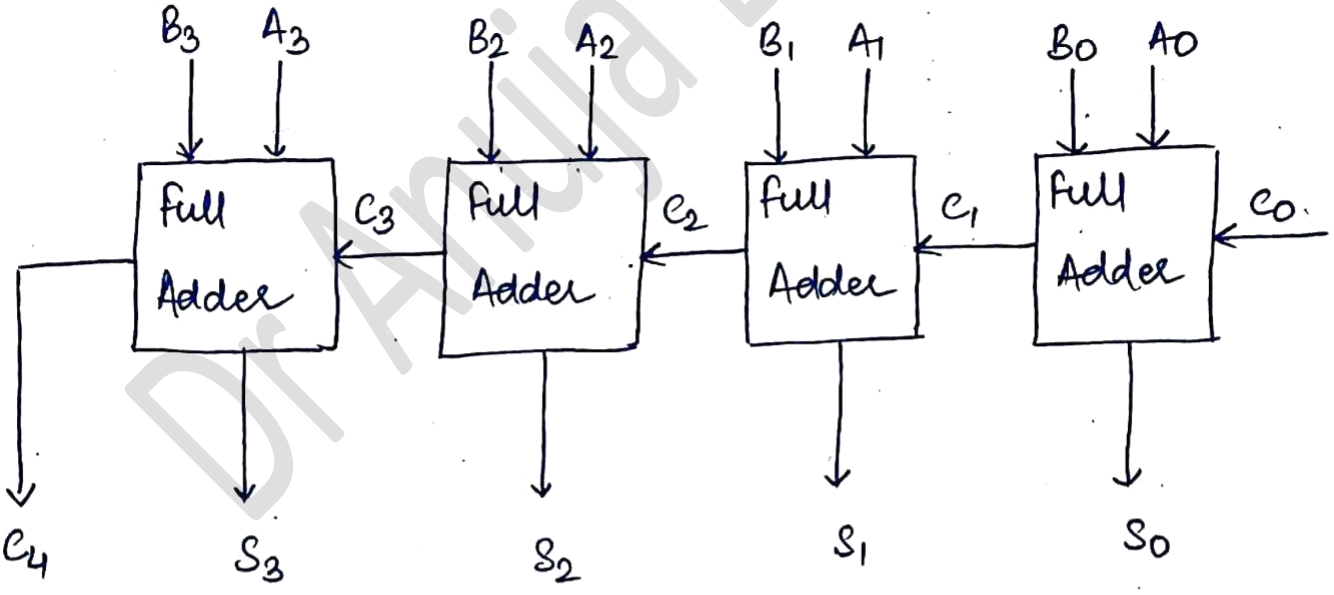


Fig :- 4 bit binary adder

Binary Adder / Subtractor :-

- The subtraction of binary numbers can be done most conveniently by means of complement.
- Since, subtraction A-B can be done by taking 2's complement of B and adding it to A.
  And, the two's complement can be obtained by taking 1's complement and adding one to the least bit.
- The addition and subtraction operations can be combined into one common circuit by including X-OR gate with full adder.

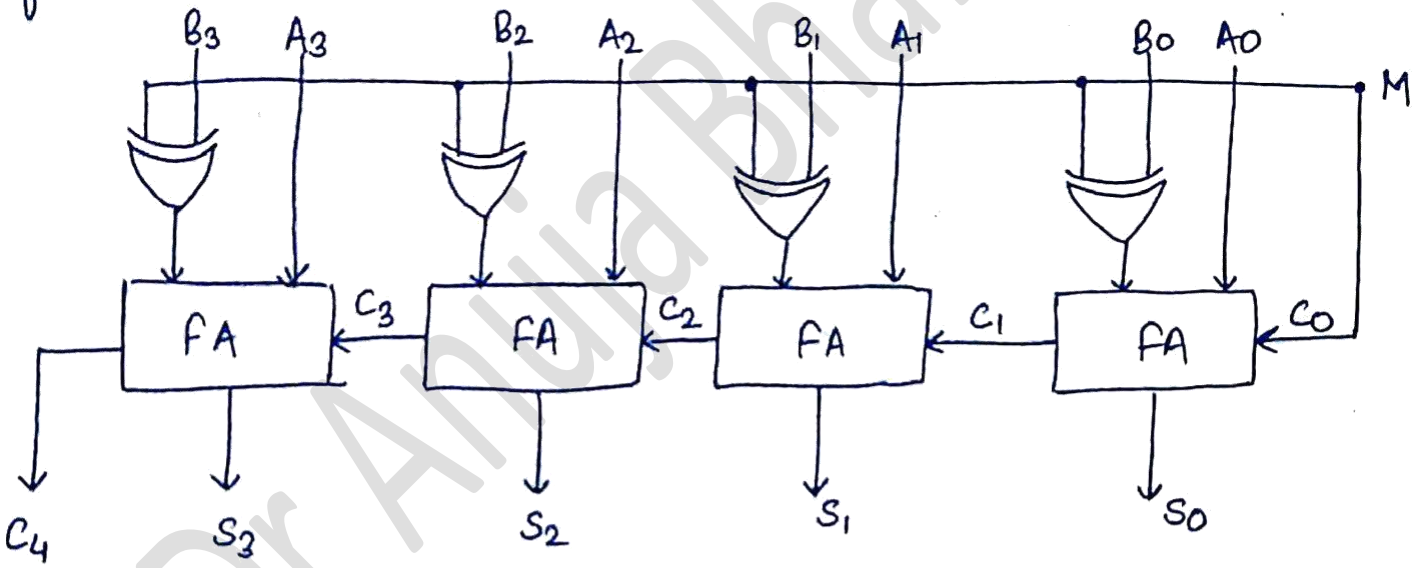


Fig :- 4 Bit Adder / Subtractor

- The input M controls the operation
  → When M = 0, the circuit is an adder.
  → When M = 1, the circuit becomes a subtractor.

→ When $M = 0$, $B \oplus 0 = B$

The full adder recieves the value of B, the input carry is 0, the circuit performs A + B.

→ When $M = 1$, $B \oplus 1 = \overline{B}$ & $C_0 = 1$

The B is complemented and 1 is added through the input carry.

Therefore, circuit performs A plus the 2's complement of B.

Carry Look Ahead Adder

- The carry look ahead adder is based on the principle of of looking at the lower bits of augend and addend if a high order carry is generated.
- This adder reduces the carry delay by reducing the number of gates through which a carry signal must propgate.
- Consider Truth Table of Full Adder

A	B	C_{in}	S	C_{out}	
0	0	0	0	0	} No Carry generation ie $C_{out} = 0$
0	0	1	1	0	
0	1	0	1	0	} Carry Propgation ie $C_{out} = C_{in}$
0	1	1	0	1	
1	0	0	1	0	
1	0	1	0	1	
1	1	0	0	1	} Carry generation ie $C_{out} = 1$
1	1	1	1	1	

• From above table,

→ Let G_i represent the unity carry ($C_{out} = 1$).

Therefore,

$$G_i = A_i B_i C_{in} + A_i B_i \overline{C}_{in}$$

$$= A_i B_i \left[C_{in} + \overline{C}_{in} \right]$$

$$= A_i B_i$$

→ Similarly, let P_i represent the carry propgate condition of the i^{th} stage of parllel adder.

$$P_i = A_i \overline{B}_i + \overline{A}_i B_i$$

$$= A_i \oplus B_i$$

→ Consider the addition of two 4 bit binary numbers,

$$A \ (A_3 \ A_2 \ A_1 \ A_0)$$

$$B \ (B_3 \ B_2 \ B_1 \ B_0)$$

→ The unity carry output of the i^{th} stage can be express as,

$$C_i \ (C_{out}) = G_i + P_i C_{i-1}$$

where C_{i-1} is unity carry of $(i-1)^{th}$ stage.

★★ $\text{Carry} = (A \oplus B) C + AB$

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

- In a 4-bit binary adder, four stage of addition is required i.e A_0+B_0, A_1+B_1, A_2+B_2, A_3+B_3.
- Therefore,

for $i=0$, $C_0 = G_0 + P_0 C_{in}$

where $G_0 = A_0 B_0$, $P_0 = A_0 \oplus B_0$, $C_{in} = 0$.

for $i=1$, $C_1 = G_1 + P_1 C_0$

$= G_1 + P_1 (G_0 + P_0 C_{in})$

$= G_1 + P_1 G_0 + P_1 P_0 C_{in}$

where $G_1 = A_1 B_1$, $P_1 = A_1 \oplus B_1$

for $i=2$, $C_2 = G_2 + P_2 C_1$

$= G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_{in})$

$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$

where $G_2 = A_2 B_2$, $P_2 = A_2 \oplus B_2$

for $i=3$, $C_3 = G_3 + P_3 C_2$

$= G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in})$

$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$

where $G_3 = A_3 B_3$, $P_3 = A_3 \oplus B_3$

- Similarly, the sum of A & B is given by

$S_0 = A_0 \oplus B_0 \oplus C_{in}$

$$S_1 = A_1 \oplus B_1 \oplus C_0$$

$$S_2 = A_2 \oplus B_2 \oplus C_1$$

$$S_3 = A_3 \oplus B_3 \oplus C_2$$

Using above equation, a 4-bit carry look ahead adder can be realized as

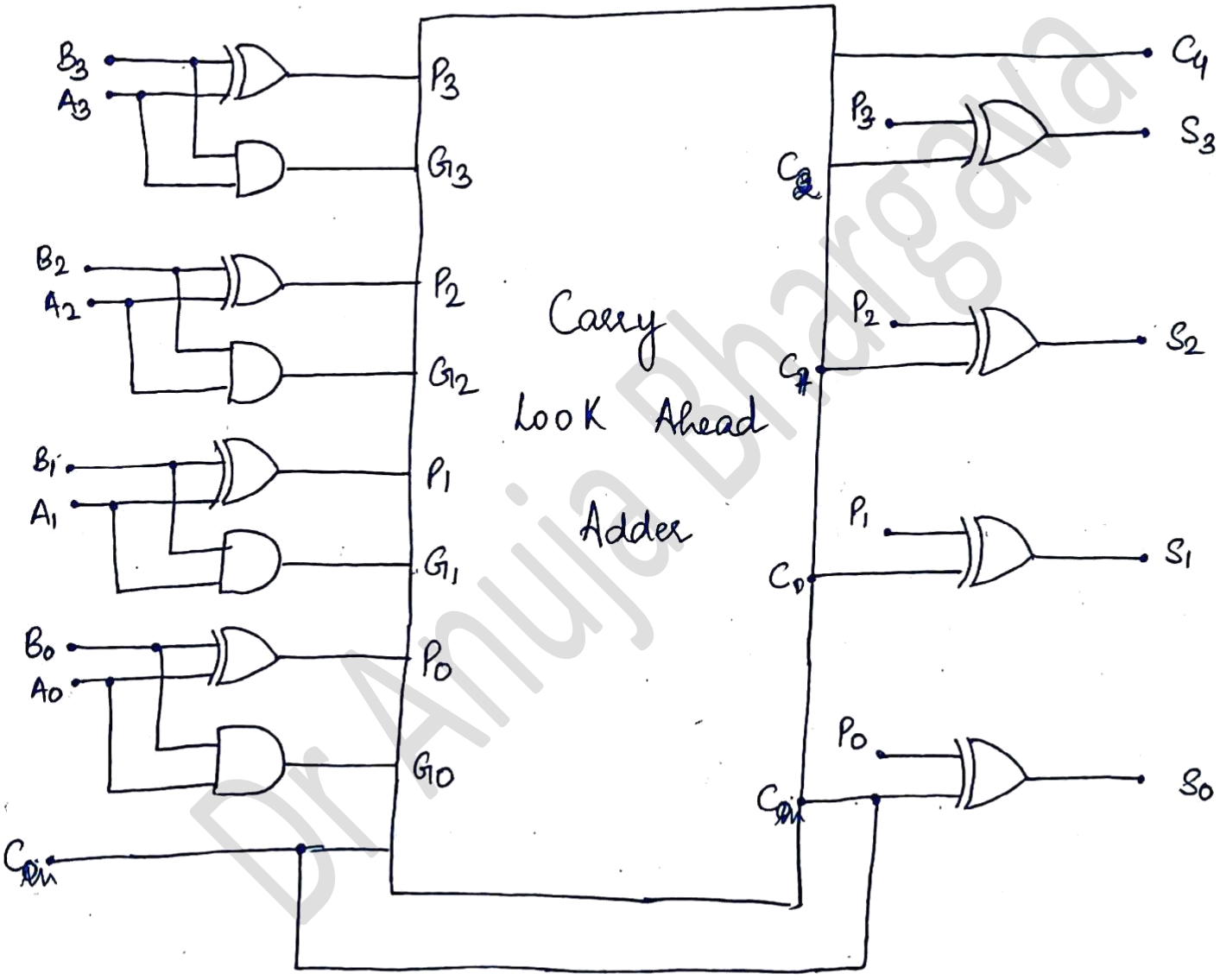
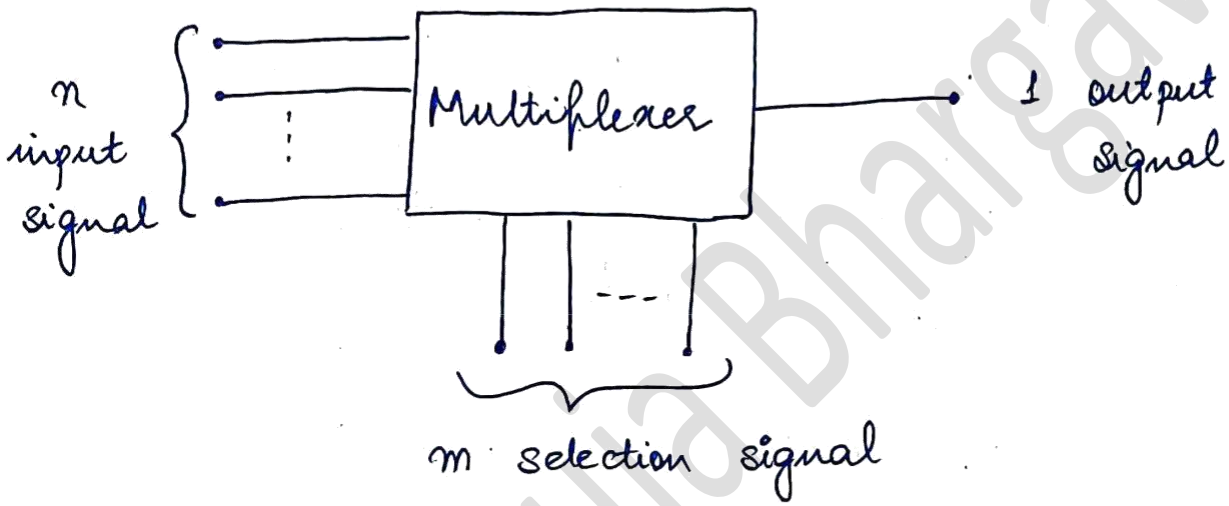


Fig :- 4 Bit Adder with Carry Look Ahead

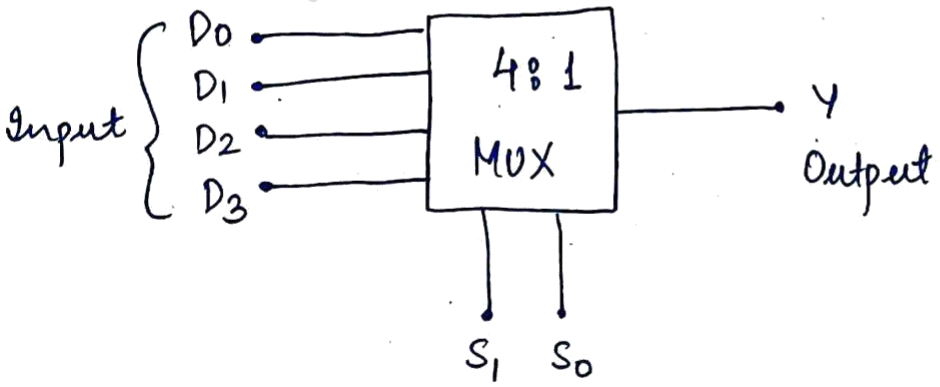
Multiplexers :-

- Multiplexing is the process of transmitting a large information over a single line.
- Multiplexer is a combinational circuit that selects one digital information from several sources and transmits the selected information on single output line.
- Block Diagram



- If n is input signal, then $2^m = n$.
- Application- act like digitally controlled multiposition switch.

eg 4:1 MUX



Truth Table

S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Boolean Expression

$$Y = D_0 \bar{S}_1 \bar{S}_0 + D_1 \bar{S}_1 S_0 + D_2 S_1 \bar{S}_0 + D_3 S_1 S_0$$

Logic Diagram

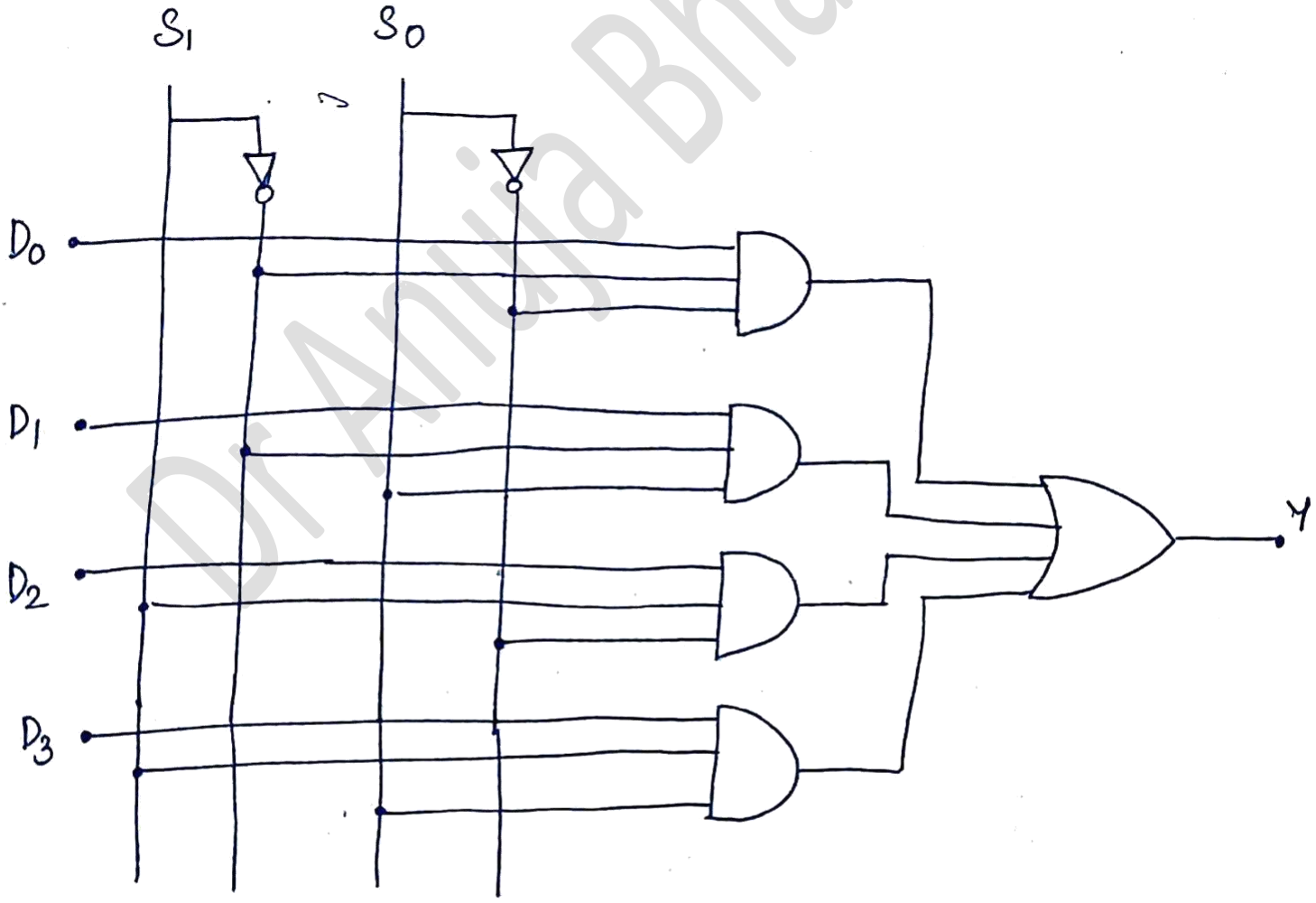
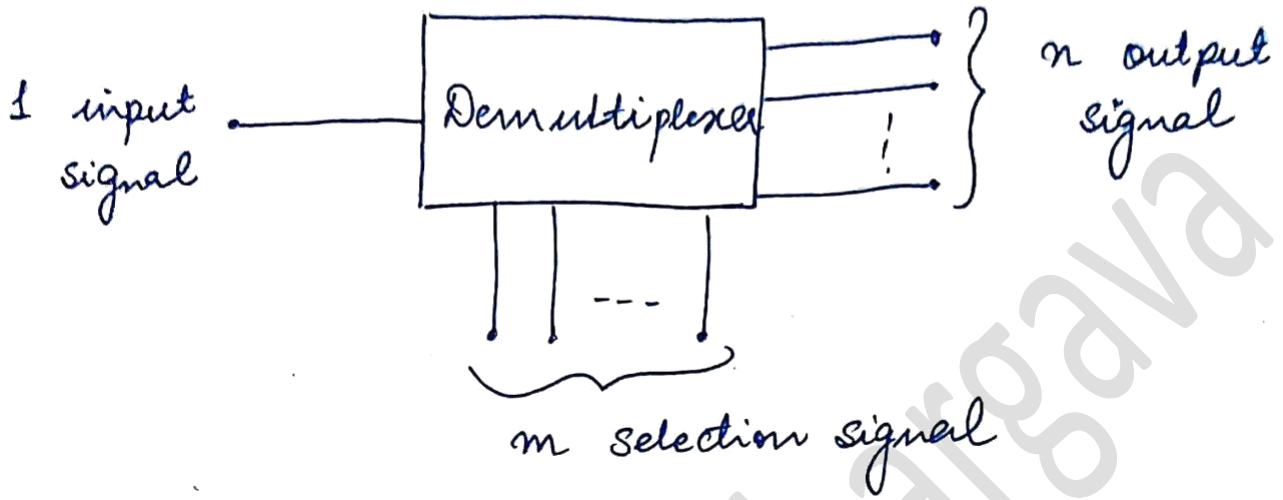


Fig :- 4 : 1 Multiplexer

Demultiplexer

- It is the process of taking information from one input and transmit the same over several output.
- Block Diagram



1 : 4 De-Mux

Truth Table

Input	Selection Signal		Output			
D	S_1	S_0	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

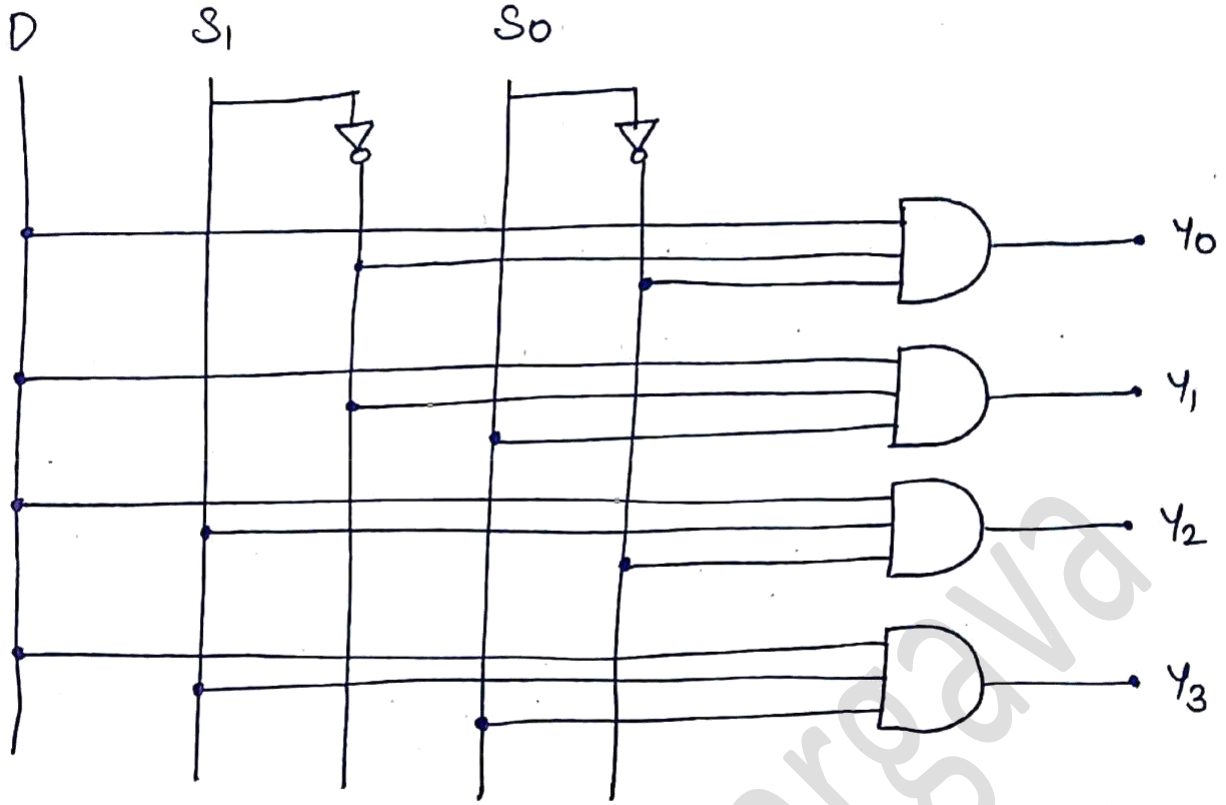
$Y_0 = D \bar{S}_1 \bar{S}_0$

$Y_1 = D \bar{S}_1 S_0$

$Y_2 = D S_1 \bar{S}_0$

$Y_3 = D S_1 S_0$

Logic Diagram



Difference between Multiplexer & Demultiplexer

Multiplexer	De - Multiplexer
1) It processes the digital information from various source into a single source.	It recieves digital information from single source and convert it into several sources
2) It is known a Data Selector.	It is known a Data Distributo
3) Multiplexer is a digital switch.	It is a digital circuit.
4) It has n data input.	It has single data input.
5) It has single data output.	It has n data output.
6) It is used at the transmitter end.	It is used at the receiver end.
7) Application - Telephone Network	

Register Transfer Language :-

- A digital system is an interconnection of digital hardware modules that accomplish a specific information processing task.
- Digital system design use a modular approach.
  - → The modules are constructed from such digital components as registers, decoders, arithmetic element & control logic.
  - → The various modules are interconnected with common data and control path to form a digital computer system.
- The operations executed on data stored in registers are called microoperations.
- The term register transfer implies the availability of hardware logic circuit that can perform a stated microoperation and transfer the result of operation to the same or another register.
- Common registers are designated by capital letters to denote the function of the register.

eg The register that holds an address for memory unit is called memory address register, indicated by MAR.

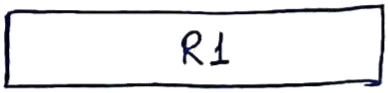
eg Other designations for registers are

PC - Program Counter (name of 16 bit register

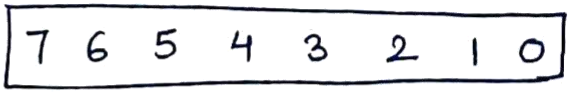
IR - Instruction Register

R1 - Processor Register

- Figure shows the representation of registers in block diagram form

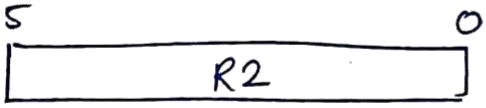


Register R

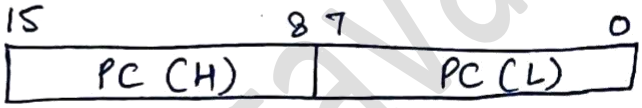


Showing individual bits

- Above is the most common way to represent a register



Numbering of bits



Divide into two parts

- In a 16-bit register, numbering of bits is done on top of the box. It is partitioned into two parts.

(i) Bits 0-7 low byte → PC (0-7) / PC (L)

(ii) Bits 8-15 high byte → PC (8-15) / PC (H)

- Information transfer from one register to another indicated by

$$R2 \leftarrow R1$$

Note:- 1) The content of register R1 transfer to Register R2.

2) The content of R1 (source register) does not change

- Generally, the transfer occur using predetermined condition

If (P = 1) then (R2 ← R1) or P: R2 ← R1

where P is control signal generated in control section

Bus & Memory Transfer

- A more efficient scheme for transfering information between registers in a multiple register configuration is a common bus system.
- A bus structure consist of set of common lines, one for each bit of a register, through which binary information is transfered one at a time.
- One way of constructing a common bus system is with multiplexers. The multiplexer select the source register whose binary information is then placed on the bus.

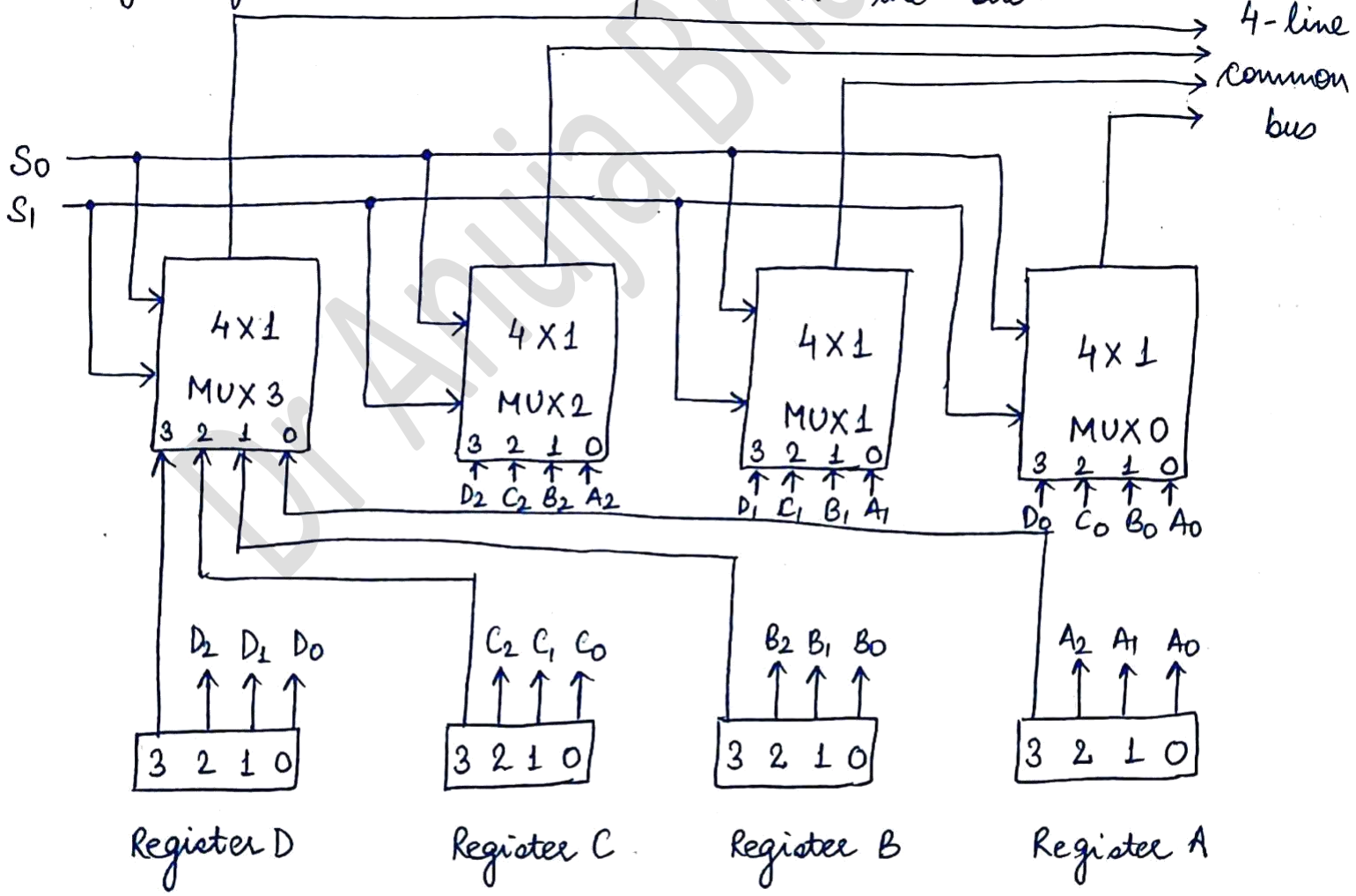


Fig :- Bus system for four register

- Above figure shows construction of a bus system for four registers.
- The selection lines choose the four bits of one register and transfer them into four line common bus.

Function Table of Bus

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

★★ Memory Transfer :-

- The transfer of information from a memory word to the outside environment is called a read operation.
- The transfer of new information to be stored in the memory is called a write operation.
- Consider a memory unit that recieves the address from register is called address register (AR). The data is transfered to another register called data register (DR).

Read : $DR \leftarrow M[AR]$

Write : $M[AR] \leftarrow R1$

Q A digital computer has a common bus system for 16 registers of 32 bit each.

The bus is constructed with multiplexers.

a) How many selection inputs are there in each multiplexer?

Sol:- 16 registers = 2^4 registers

∴ 4 selection lines are there to select 16 registers

b) What size of multiplexer are needed?

Sol:- Size of multiplexer is 16 × 1 multiplexer.

c) How many multiplexers are there in a bus?

Sol:- Number of multiplexer = Number of bits in each register

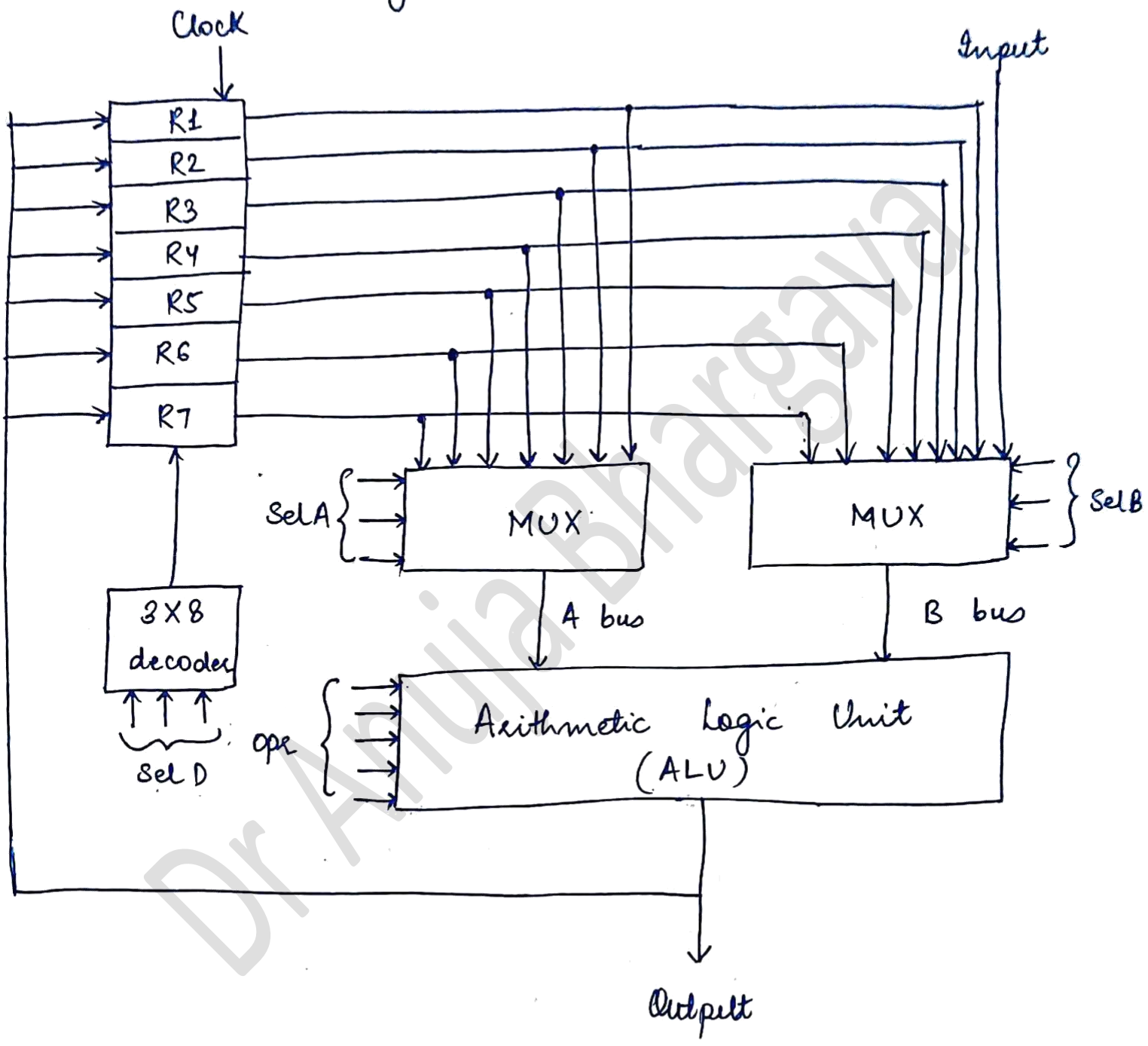
Therefore, 32 multiplexers, one for each bit of the register.

General Register Organization :-

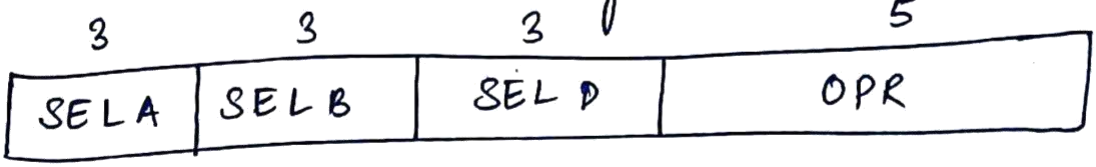
- Generally, memory location are needed for storing pointers, counters, return addresses.
- Having to refer to memory locations for any application is time consuming operation in a computer.
- It is more convenient and efficiant to store immediate values in processor registers.
- When a large number of registers are included in the CPU, it is more effecient to connect them through a common bus system.
- The registers communicate with each other not only for direct data transfer but also for performing micro operations.
- Therefore, it is necessary to provide a common unit that can perform all the arithmetic, logic and shift micro operations in the processor.
- A bus organization for seven CPU registers is shown.
  - → The output of each register is connected to two multiplexers to form two buses A & B.
  - → The selection lines selects one register.
  - → The A & B bus forms the input to arithmetic logic unit
  - → The operation selected in ALU determines the logic or arithmetic micro operation to be performed.

→ The result of microoperation is available for output data & also goes into the input of all the registers.

→ The register that recieves the information from the output bus is selected by decoder.



(a) Block Diagram



(b) Control Word

Fig :- Register Set with common ALU

Eg To perform the operation

$R1 \leftarrow R2 + R3$

1. MUX A Selector (Sel A): place the content of R2 into bus A.
2. MUX B Selector (Sel B): place the content of R3 into bus B.
3. ALU operation selector (OPR): provide A + B
4. Decoder destination selector (SELD): transfer the content of output bus into R1.

Control Word

- There are 14 binary selection input in the unit, and their combined value specify a control word.

Encoding of Register

Binary Code	SEL A	SEL B	SEL D
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

- Encoding of ALU operations

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INC A
00010	Add A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DEC A
01000	AND A & B	AND
01010	OR A & B	OR
01100	XOR A & B	XOR
01110	Complement A	COM A
10000	Shift right A	SHR A
11000	Shift left A	SHL A

eg

Micro operation	SEL A	SEL B	SEL D	Control Word
R1 ← R2 - R3	R2	R3	R1	010 011 001 00101
R4 ← R4 V R5	R4	R5	R4	100 101 101 01010
R7 ← R1	R1	-	R7	001 000 111 00000
Output ← R2	R2	-	-	010 000 000 000 xx

Stack Organization :-

- A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. (LIFO)
- The stack in digital computers is essentially a memory unit with an address register that can count only.
- The register that holds the address for the stack is called stack pointer (SP). because its value always point at the top item in the stack.
- The two operations of a stack are insertion and deletion of items.

→ The operation of insertion is called PUSH because it can be thought of as the result of pushing a new item on top.

→ The operation of deletion is called POP because it can be thought of as the result of removing one item so that the stack pops up.

Note :- However, nothing is pushed or popped in a stack. These operations are simulated by incrementing or decrementing the stack pointer register.

Register Stack

- A stack can be placed in a portion of a large memory or it can be organized as a collection of finite number of memory words or registers.
- Fig shows the organization of a 64-word register stack.

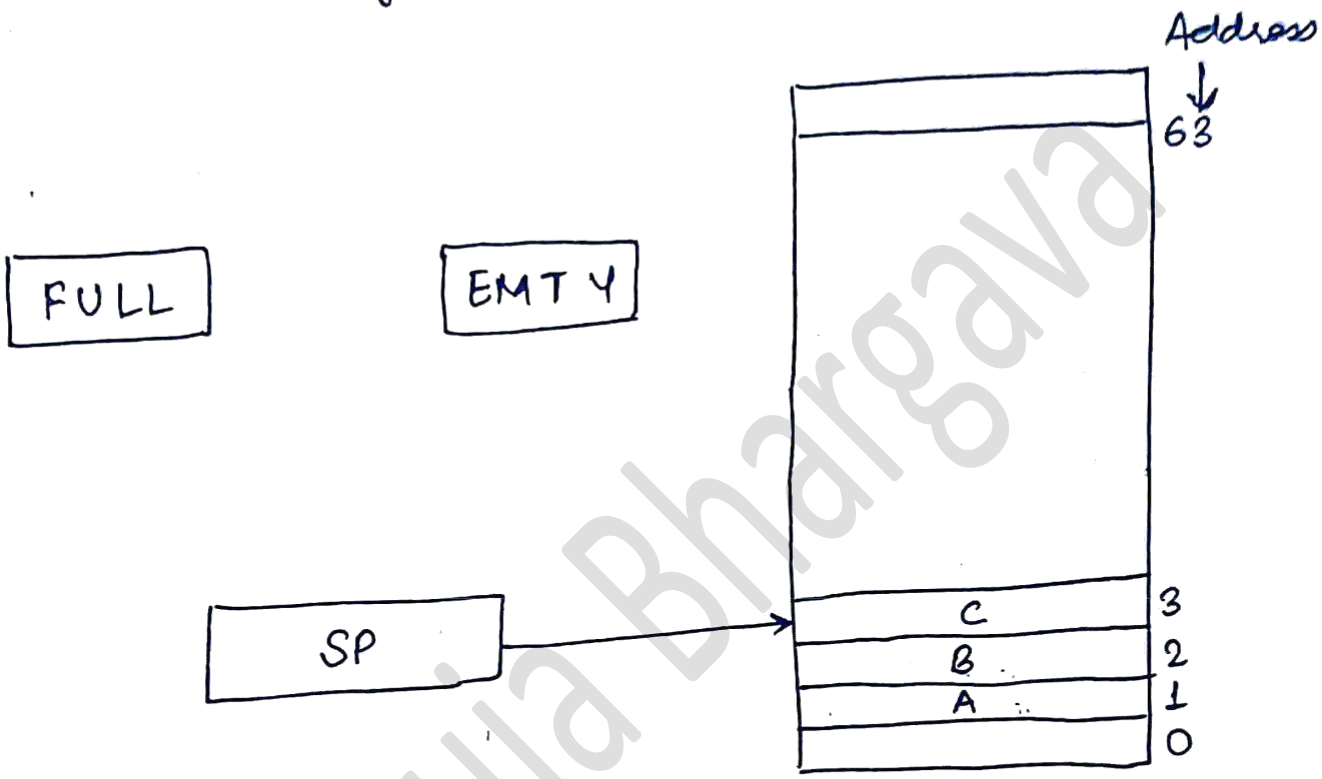


Fig :- Block Diagram of 64-word stack

→ The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on the top of the stack.

→ Initially, SP is cleared to 0, EMTY is set to 1 and FULL is cleared to 0, then push operation is implemented with the following sequence of microoperations

SP ← SP + 1	Increment stack pointer
M [SP] ← DR	Write item on top of stack
If (SP=0) then (FULL ← 1)	Check if stack is full
EMTY ← 0	Mark the stack not empty

→ A new item is deleted from stack if the stack is not empty (if EMTY=0). The POP operation consist of following micro operations:

DR ← M [SP]	Read item from top of stack
SP ← SP - 1	Decrement stack pointer
If (SP=0) then (EMTY ← 1)	Check if stack is empty
FULL ← 0	Mark the stack not full

- Reverse Polish Notation

**Note :- A stack organization is very effective for evaluating arithmetic expressions. The common arithmetic expression are written in infix notation with each operator written between the operands.

eg A * B + C * D

- The Polish mathematician Lukasiewicz showed that arithmetic expressions can be represented in prefix Notation or polish notation.
- The Postfix Notation refred as Reverse Polish Notation (RPN), places the operator after the operands.

→ examples

* A + B Infix Notation
* + AB Prefix or Polish Notation
* AB + Postfix or Reverse Polish Notation

- The reverse polish notation is in a form suitable for stack manipulation.

Rule: → Scan the expression from left to right.

→ When operator is reached, perform the operation with the two operands found on the left side of operator.

→ Remove the two operands and the operator and replace them by the number obtained from the result of the operation.

→ Continue the scan and repeat the procedure for every operator encountered until there are no more operator

Eg The expression A * B) + C * D

is in Reverse Polish Notation

AB * CD * +

** (A*B) CD * +

(A * B) (C * D) +

(A * B) + (C * D)

Eg (A + B) * [C * (D+E) + F]

AB + DE + C * F + *

Evaluation of Arithmetic Expressions

- Reverse Polish Notation with stack arrangement of registers is the most efficient way known for evaluating arithmetic expression

Eg The arithmetic expression

(3 * 4) + (5 * 6)

In Reverse Polish Notation

34 * 56 * +

Stack Operation

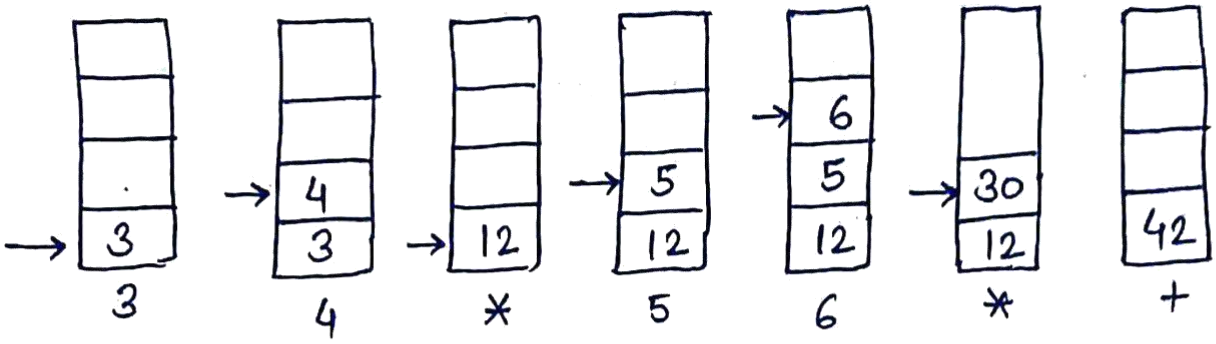


Fig:- Stack Operator to evaluate 3*4 + 5*6

Exercise

Q Convert the following arithmetic expressions into reverse polish notation and show the stack operations for evaluating the numerical result:

a) 8*3 + (4+8) + (4*7+2) + 2+3

b) 5 + 10*6 + (3+5) + 7*(6+2)

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

d) (A+B)*C - (D-E)*(F+G)

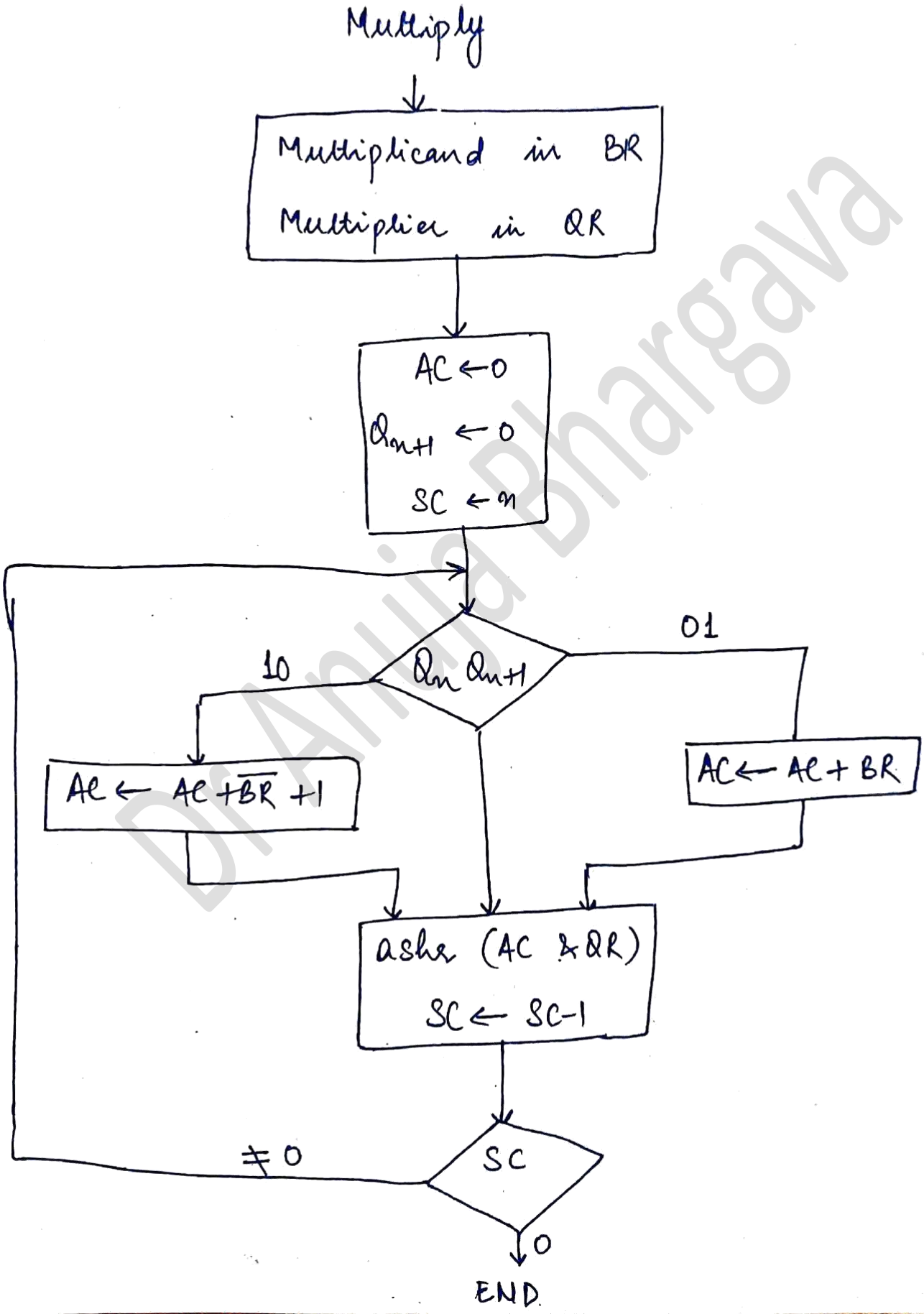
f) (A*B) + (C*D) + (E*F)

g) (A*B) + A*(B*D + C*E)

h) A+B*[C*D + E*(F+G)]

Booth Algorithm :-

- This algorithm gives a procedure for multiplying binary integers in signed 2's complement representation.



B → Multiplicand

A → Partial Product, Initially zero

Q → Multiplier

Q_{n+1} → Initially Zero

Q_n → LSB of Q

Q_n	Q_{n+1}	
0	0	ashr
0	1	add + ashr
1	0	sub + ashr
1	1	ashr

ADD → A + B

Sub → A + $\bar{B}$ + 1

eg (7) X (-5)

B = 7 +7 → 0111 (B) +5 → 0101

Q = -5 -7 → 1001 ($\bar{B}$+1) -5 → 1011

Q_n	Q_{n+1}	Operation	A	Q	Q_{n+1}	SC
-	-	-	0000	1011	0	100 (4)
1	0	subr	0000 + 1001 = 1001	1011	0	-
		ashr	1100	1101	1	011 (3)
1	1	ashr	1110	0110	1	010 (2)
0	1	add	0111 + 1010 = 10101	0110	1	
		(Carry omit)				
		ashr	0010	1011	0	001 (1)
1	0	sub	1001 + 1011 = 10100	1011	0	
		ashr	1101	1101	1	000 (0)

Answer = A Q

= 1 1 0 1 1 1 0 1

↑ (-ve) | Mag.

↓ 2's

(1 0 1 0 0 0 1 1)

(-35)

eg (-15) × (-8)

B +15 → 0 1 1 1 1 (B̄+1) Q + 8 → 0 1 0 0 0

-15 → 1 0 0 0 1 B -8 → 1 1 0 0 0 (Q)

Q_n	Q_{n+1}	Operation	A	Q	Q_{n+1}	SC
-	-	-	00000	11000	0	101
0	0	ashr	00000	01100	0	100
0	0	ashr	00000	00110	0	011
0	0	ashr	00000	00011	0	010
1	0	sub	01111 / 01111	00011	0	
		ashr	00111	10001	1	001
1	1	ashr	00011	11000	1	000

Ans = AQ = 000111111000 = +120

Q (−9) × (−13)

+9 → 01001 ($\bar{B}$+1) +13 → 01101

−9 → 10111 (B) −13 → 10011 (Q)

Q_n	Q_{n+1}	Operation	A	Q	Q_{n+1}	sc
			00000	10011	0	101
1	0	sub	01001			
			01001	10011	0	-
		ashr	00100	11001	1	100
1	1	ashr	00010	01100	1	011
0	1	add	10111			
			11001	01100	1	
		ashr	11100	10110	0	010
0	0	ashr	11110	01011	0	001
1	0	sub	01001			
		Carry →	(1) 00111	01011	0	
		ashr	00011	10101	1	000

Answer = AQ

= 00011 10101