

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 computationally 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.

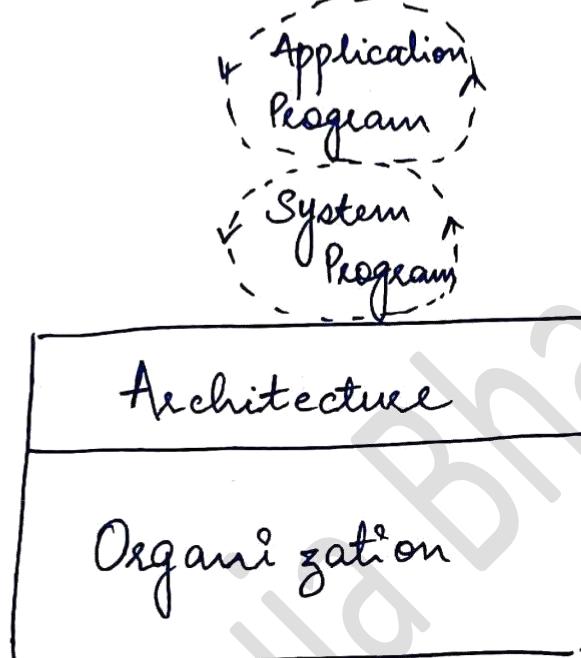
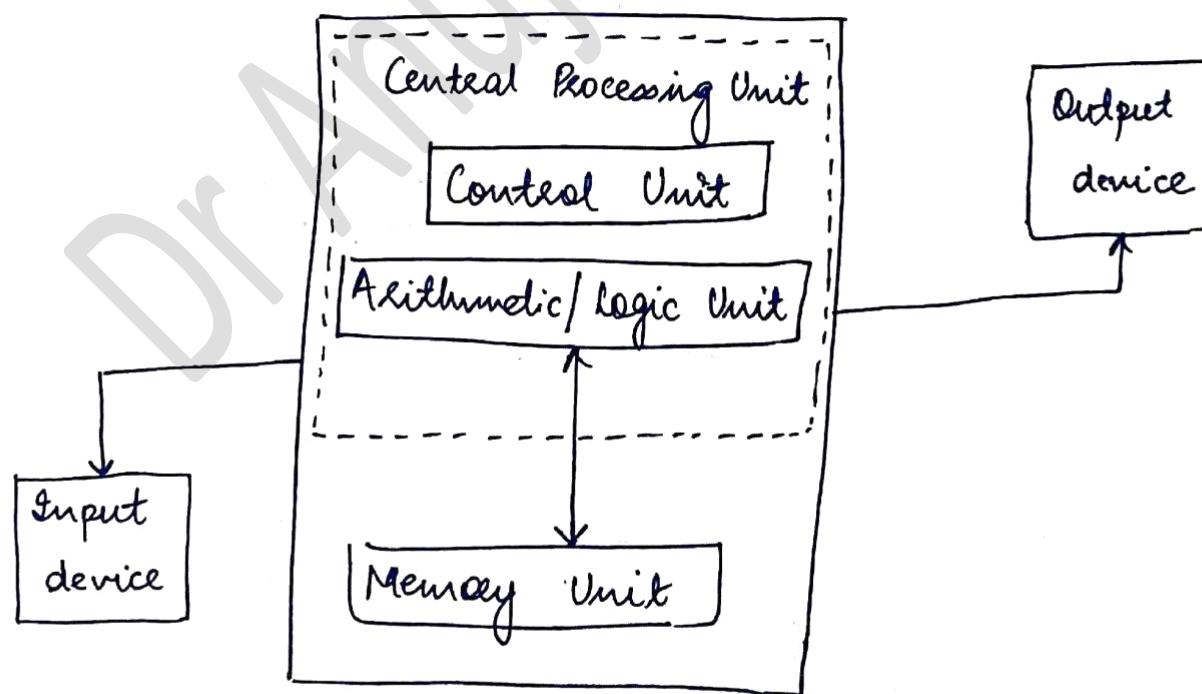


fig: Organization implements the architecture

- 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.

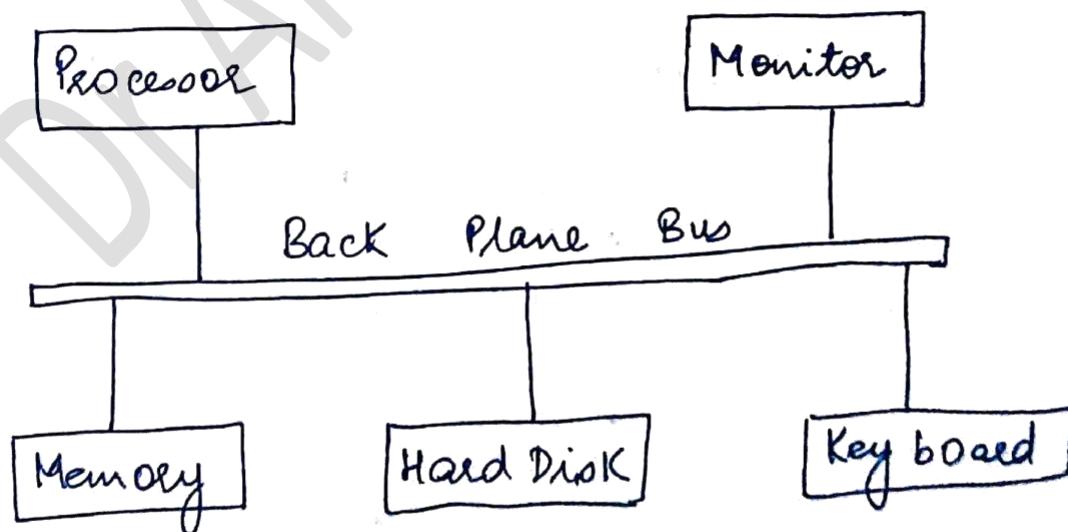
Von Neumann Architecture :-

- 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
- Van 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 contains 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 numbers 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

④ Decimal to Binary

eg 17

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

Binary Number

10001

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

$$\begin{array}{rcl}
 0.32 \times 2 & = & 0.64 & 0 \\
 0.64 \times 2 & = & 1.28 & 1 \\
 0.28 \times 2 & = & 0.56 & 0 \\
 0.56 \times 2 & = & 1.12 & 1 \\
 1.12 \times 2 & = & 0.24 & 0
 \end{array}$$

Binary Number $(110111.01010)_2$

(B) Binary to Decimal

eg 101100101

$$\begin{aligned}
 & 1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 \\
 & + 0 \times 2^1 + 1 \times 2^0 \\
 = & 256 + 64 + 32 + 4 + 1 \\
 = & (357)_{10}
 \end{aligned}$$

eg 1101.101

$$\begin{aligned}
 & 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\
 = & 8 + 4 + 1 & = 0.5 + 0.125 \\
 = & 13 & = 0.625
 \end{aligned}$$

Decimal Number $(13.625)_{10}$

(C) Decimal to Octal

eg 33

$$\begin{array}{r} 8 \mid 33 \mid 1 \\ \hline 4 \end{array}$$

$$= (41)_8$$

eg 45.26

$$\begin{array}{r} 8 \mid 45 \mid 5 \\ \hline 5 \mid 5 \end{array}$$

$$\begin{aligned} 0.26 \times 8 &= 2.08 & 2 \\ 0.08 \times 8 &= 0.48 & 0 \\ 0.48 \times 8 &= 3.84 & 3 \end{aligned}$$

(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

$$\begin{array}{r} 16 \mid 455 \\ \hline 16 \mid 28 \\ \hline 1 \end{array}$$

7

12 (C)

$$(1C7)_{16}$$

F) Hexadecimal to Decimal

eg 54. D2

$$\begin{aligned} &= 5 \times 16^1 + 4 \times 16^0 + 13 \times 16^{-1} + 2 \times 16^{-2} \\ &= 80 + 4 + 0.8125 + 0.0078125 \\ &= 84.8203125 \end{aligned}$$

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.

$$\begin{array}{ccc} . & 7 & 4 & 2 \\ & (111 & 100 & 010)_2 \end{array}$$

eg Convert $(10100110)_2$ to Octal.

$$\begin{array}{ccc} 010 & 100 & 110 \\ (2 & 4 & 6)_8 \end{array}$$

#	<u>Binary</u>	<u>Coded Hexadecimal</u>	
	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	A	A
	1011	B	B
	1100	C	C
	1101	D	D
	1110	E	E
	1111	F	F

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

3 D 9

0011 1101 1001

$$= (1111011001)_2$$

eg Convert $(110100110)_2$ to hexadecimal.

0001 1010 0110

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

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

$$\begin{array}{cccc} E & 8 & A \\ \downarrow & & \\ 1110 & 1000 & 1010 \\ \downarrow & & \\ 111 & 010 & 001 & 010 \\ \downarrow & & & \\ (7 \ 2 \ 1 \ 2)_8 \end{array}$$

eg Convert $(3756)_8$ to Hexadecimal.

$$\begin{array}{cccc} 3 & 7 & 5 & 6 \\ \downarrow & & & \\ 011 & 111 & 101 & 110 \\ \downarrow & & & \\ 0111 & 1110 & 1110 \end{array}$$

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

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 & system.
 - (i) n 's complement
 - (ii) $(n-1)$'s complement

→ When $n=10$, 10's and 9's complement.

→ When $n=2$, 2's and 1's complement.

• Generally,

$$\begin{array}{r} 110011 \\ - 11000 \\ \hline \end{array} \xrightarrow{\text{1's comp.}} \begin{array}{r} 001100 \\ - 00111 \\ \hline \end{array} \xrightarrow{\text{2's comp.}} 001101$$
$$\begin{array}{r} 11000 \\ - 1100 \\ \hline \end{array} \xrightarrow{\text{1's comp.}} \begin{array}{r} 00111 \\ - 00110 \\ \hline \end{array} \xrightarrow{\text{2's comp.}} 01000$$

→ 1's Complement Subtraction

• Subtraction of binary numbers using the 1's complement Method allows 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

$$\begin{array}{r} 1111 \\ - 1010 \\ \hline 0101 \end{array}$$

1's complement Method

$$\begin{array}{r} 1111 \\ 1's \text{ complement} \rightarrow 0101 \\ \hline 10100 \\ \text{Carey} \rightarrow \underline{1} \\ \text{Add Carey} \rightarrow \underline{0101} \end{array}$$

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

Soln

Direct subtraction

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

1's complement Method

$$\begin{array}{r} 1000 \\ 0101 \\ \hline 1101 \end{array}$$

$\downarrow 1's \text{ comp}$

$$-0010$$

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

- Determine the 2's complement of smaller number.
- Add this to the larger number.
- 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

$$\begin{array}{r} -1 \ 1 \ 1 \ 1 \\ 0 \ 1 \ 1 \ 0 \\ \hline \text{Carry} \rightarrow 1 \ 0 \ 1 \ 0 \ 1 \end{array}$$

∴ 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.

$$\begin{array}{r} 1000 \\ 0110 \\ \hline \text{No carry} \rightarrow 1110 \end{array}$$

∴ 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,

$$+33 \rightarrow 00100001$$

Sign	Magnitude
------	-----------

$-33 = 1's \text{ complement of } (+33)$

$= 1's \text{ compl. of } (00100001)_2$

$= \text{ (11011110)}_2$

$\text{2}'$'s complement Representation

e.g. $(-33) = ?$

$$33 \rightarrow (100001)_2$$

$$8\text{ bit} \rightarrow (00010001)_2$$

$-33 = \text{2}'\text{s complement of } (+33)$

$= \text{2}'\text{s complement of } (00100001)_2$

$$= (11011111)_2$$

Exercise

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

④ + 26	Sign Magnitude	1's <small>represⁿ</small>	2's <small>represⁿ</small>
+26	0 0011010	0 0011010	0 0011010

⑤ - 37	1 0100101	0 1011010	0 1011011
--------	-----------	-----------	-----------

⑥ - 101	1 1100101	0 0011010	0 0011011
---------	-----------	-----------	-----------

Addition in 2's Complement System

(Case I.) Two Positive Numbers

$$\begin{array}{r}
 +29 \rightarrow 0 \quad 0011101 \\
 +19 \rightarrow 0 \quad 0010011 \\
 \hline
 +48 \quad 0 \quad 0110000
 \end{array}$$

Sign Magnitude.

$\text{Sign} \rightarrow 0$: When sum is positive they have the same number of bits.

(Case 2.) Positive and Negative Numbers

$$(-22) \rightarrow ?$$

$$+22 \rightarrow 0 \quad 0010110$$

$$\begin{array}{r}
 -22 \rightarrow \text{2}'\text{s Complement of } (+22) \\
 = 1 \quad 1101010
 \end{array}$$

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

$$-22 \rightarrow 1 \quad 1101010$$

$$\begin{array}{r}
 +17 \quad 1 \quad 0 \quad 0010001
 \end{array}$$

Carry Sign Magnitude

∴ Carry is omitted.

∴ Answer is $(0001\ 0001)_2$

Case 3 Negative and Positive Number

$$\begin{array}{r}
 -47 \rightarrow 11010001 \\
 +29 \rightarrow 00011101 \\
 \hline
 -18 \quad 11101110
 \end{array}$$

Sign

\therefore Sign bit $\rightarrow 1$ i.e. Negative Number

$$\begin{aligned}
 \text{True Sum} &= 2^{\text{'s complement}} \text{ of Magnitude} \\
 &= 2^{\text{'s complement}} (1101110) \\
 &= 0010010
 \end{aligned}$$

Result is $(\underline{10010010})_2$

Case 4 Two Negative Number

$$\begin{array}{r}
 -32 \rightarrow 11100000 \\
 -44 \rightarrow 1010100 \\
 \hline
 -76 \quad 110110100
 \end{array}$$

Carry Sign

$$\begin{array}{r}
 +32 \rightarrow 00100000 \\
 -82 \rightarrow 11000000 \\
 \hline
 +44 \rightarrow 00101100
 \end{array}$$

$$\begin{array}{r}
 -44 \rightarrow 11010100
 \end{array}$$

\therefore Carry is discarded. \therefore True sum = $2^{\text{'s comp. of Mag.}}$

\therefore Result is $(\cancel{10010010100})_2 = 2^{\text{'s comp}} (00101100)$

Note :- $2^{\text{'s complement}}$ addition is within the range

-128 to +127. Otherwise, we get an overflow.

Subtraction in 2's Complement

10

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. $28 - 19$ ($\because +19 \rightarrow 2's\ comp$)

$$\begin{array}{r} +28 \rightarrow 0 \quad 0011100 \\ -19 \rightarrow 1 \quad 1101101 \\ \hline +9 \quad 10001001 \end{array}$$

$$[9]$$

Carry sign

\therefore Omit the Carry \therefore Result is $(00001001)_2$

Case 2

Positive Number and Smaller Negative Number

$$39 - (-21) = +60$$

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

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

$$2's\ comp.\ of (00010101)$$

$$11101011$$

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

$$+21 \rightarrow 00010101$$

It then adds +39 & +21

$$\begin{array}{r} +39 \rightarrow 00100111 \\ +21 \rightarrow 00010101 \\ \hline +60 \quad 00111100 \end{array}$$

60

(Case 3) Positive Number and Larger Negative Number

In the 2's complement system

$$19 - (-43) = 62$$

$$+19 \rightarrow 0\ 0010011$$

$$-43 \rightarrow \text{2's comp of } (+43)$$

$$\text{2's comp. of } (00101011)$$

$$\rightarrow 11010101$$

The computer sends 2's complement of -43

$$\text{i.e. } +43 \rightarrow 0\ 0101011$$

$$+19 \rightarrow 0\ 0010011$$

$$+43 \rightarrow 0\ 0101011$$

$$\begin{array}{r} +19 \rightarrow 0\ 0010011 \\ +43 \rightarrow 0\ 0101011 \\ \hline +62 \quad 0\ 0111110 \end{array}$$

62

Case 4 Both numbers are negative

In 2's complement system

$$-57 \rightarrow 11000111$$

$$-33 \rightarrow 11011111$$

$$\therefore -57 + (-33)$$

\therefore Take 2's complement of -33

$$+33 \rightarrow 00100001$$

Then add +33 to -57

$$-57 = 11000111$$

$$+33 = 00100001$$

$$\begin{array}{r} \cancel{1} \cancel{1} \cancel{0} \cancel{0} \\ \underline{1} \underline{1} \underline{0} \underline{1} \underline{0} \underline{0} \end{array}$$

Sign magnitude

$\downarrow 2^7$

$$\text{Ans} (10011000)_2$$

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

$$\begin{array}{r}
 + 120 \quad \rightarrow \quad 0\ 111\ 1000 \\
 + 65 \quad \rightarrow \quad 0\ 100\ 0001 \\
 \hline
 185
 \end{array}$$

$$1011\ 1001$$

↑

Negative, ans. is not correct.

(Case 2) Two Negative Numbers

$$\begin{array}{r}
 - 77 \quad \rightarrow \quad 1011\ 0011 \\
 - 122 \quad \rightarrow \quad 1000\ 0110 \\
 \hline
 - 199
 \end{array}$$

$$10011\ 1001$$

↑

Carry positive ↓ 2's ans. is not correct

~~10000110~~

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. Therefore, a programmer must check the overflow after each addition or subtraction.

Extra dues

Addition

$$1. (+23) + (+19) \quad 00101010$$

$$2. (+23) + (-19) \quad 1) 00000100.$$

$$3. (-23) + (+19) \quad 10000100$$

$$4. (-23) + (-19) \quad 10101010$$

Subtraction

$$1. (+23) - (+19) \quad 1) 00000100$$

$$2. (+23) - (-19) \quad 00101010$$

$$3. (-23) - (+19) \quad 10101010$$

$$4. (-23) - (-19) \quad 10000100$$

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

e.g. 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

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

Mantissa

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

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

$$\begin{array}{ccc} \text{Fraction} & \text{Exponent} & \xrightarrow{\quad} \text{Scientific Notation} \\ +0.6132789 & +04 & +0.6132789 \times 10^{+4} \end{array}$$

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

$$01001110 \qquad \qquad 000100 \rightarrow .1001110 \times 2^4$$

e.g. 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

$$2^{\text{size}-1} - 1 = 127.$$
 If size of exponent field is 11-bits then bias value

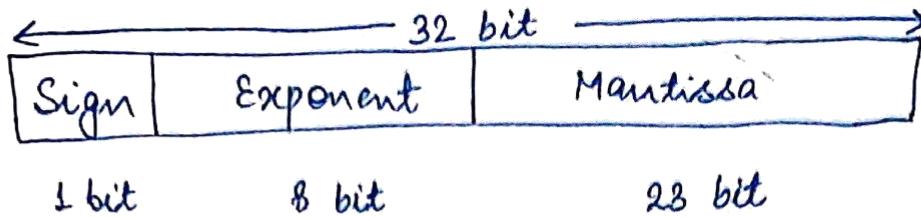
$$= 1023.$$

Normalized Mantissa

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

1. -----

Single Precision (IEEE 754 - 32 bit)



$\frac{85}{125}$ in IEEE - 32 bit format.

$$\text{Binary of } 85.125 = 1010101.001$$

$$= \underbrace{1.010101001}_{\text{Mantissa}} \times 2^{+6} \rightarrow \text{Exponent}$$

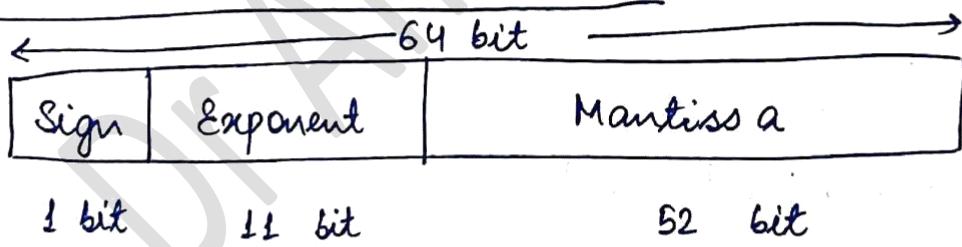
Sign = 0 ($^{\circ\circ}$ +ve)

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

~~Rhoadsides~~ =

0	10000101	010101001000000000000000
---	----------	--------------------------

Double Precision (64 bit)



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

Represent 100.125 in IEEE 754 single and double¹⁴ precision.

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

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

Mantissa = 100100001

$$\text{Exponent} = +6$$

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

Single Precision 32 bit

0	10000101	100100001000000000000000
1 bit	8 bit	23 bit

Double Precision

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

The diagram illustrates the 64-bit floating-point number format. It consists of three main fields: a 1-bit sign field, a 11-bit exponent field, and a 52-bit fraction field. The fraction field is explicitly labeled as "52 bit".

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

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

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

$$\text{Mantissa} = 00111001$$

$$\text{Exponent} = +5$$

for 32-bit ,

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

1	10000100	001110010000000000000000
---	----------	--------------------------

1 bit ← 8 bit → ← 23 bit →

for 64-bit,

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

1 bit ← 11 bit → ← 52 bit →

Extra que

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

$$\text{Solve: } +36 \rightarrow 00100100$$

$$+23 \rightarrow 000\ 10\ 111$$

-36 → 11011100

$$-(+23) \rightarrow 11101001$$



 Carry Sign Magnitude

$$1000 \text{ } 101 \xrightarrow{\text{2's}} 0111011$$

\therefore 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.

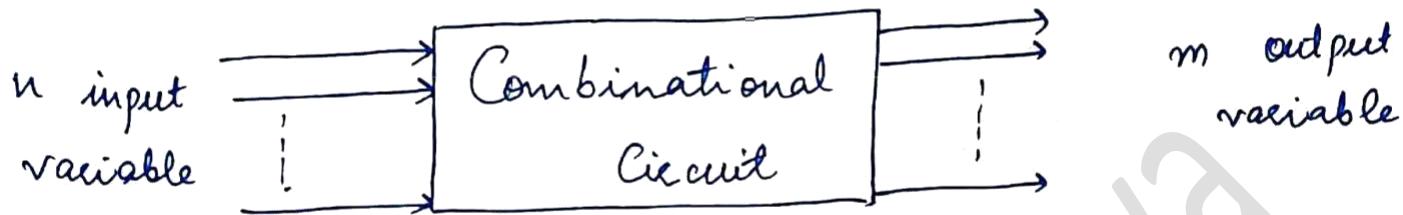


Fig :- Basic Block Diagram of Combinational Circuit

• 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.

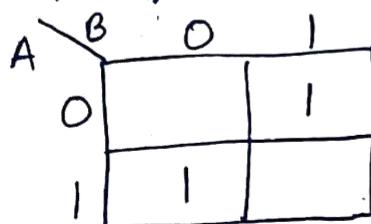
→ Half Adder :-

- 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

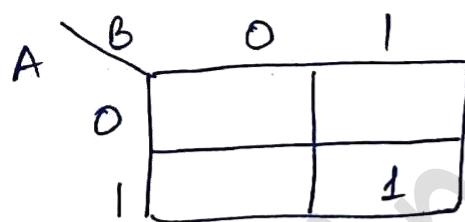
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

K-Map for Sum



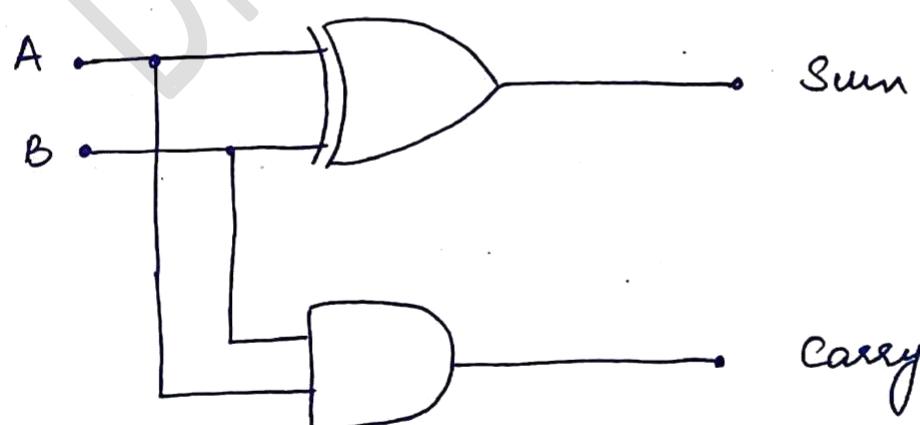
$$\text{Sum} = \overline{A}\overline{B} + A\overline{B}$$

K-Map for Carry



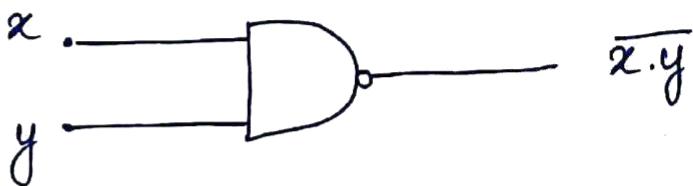
$$\text{Carry} = A \cdot B$$

Logic Diagram

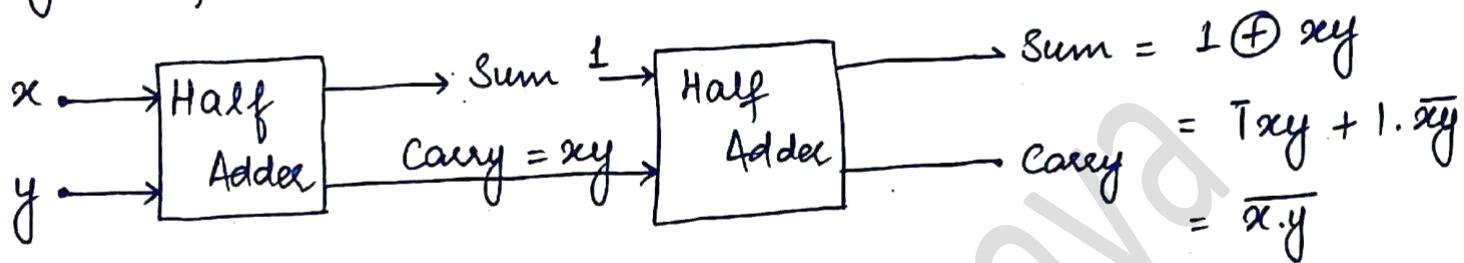


Q Construct universal gates (NAND & NOR) using Half Adder⁶

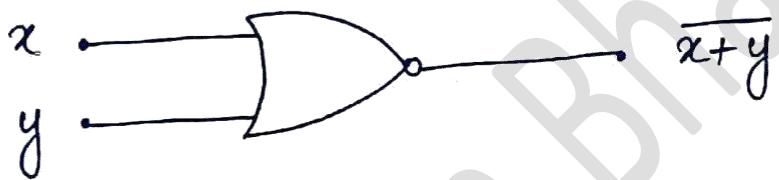
for NAND Gate,



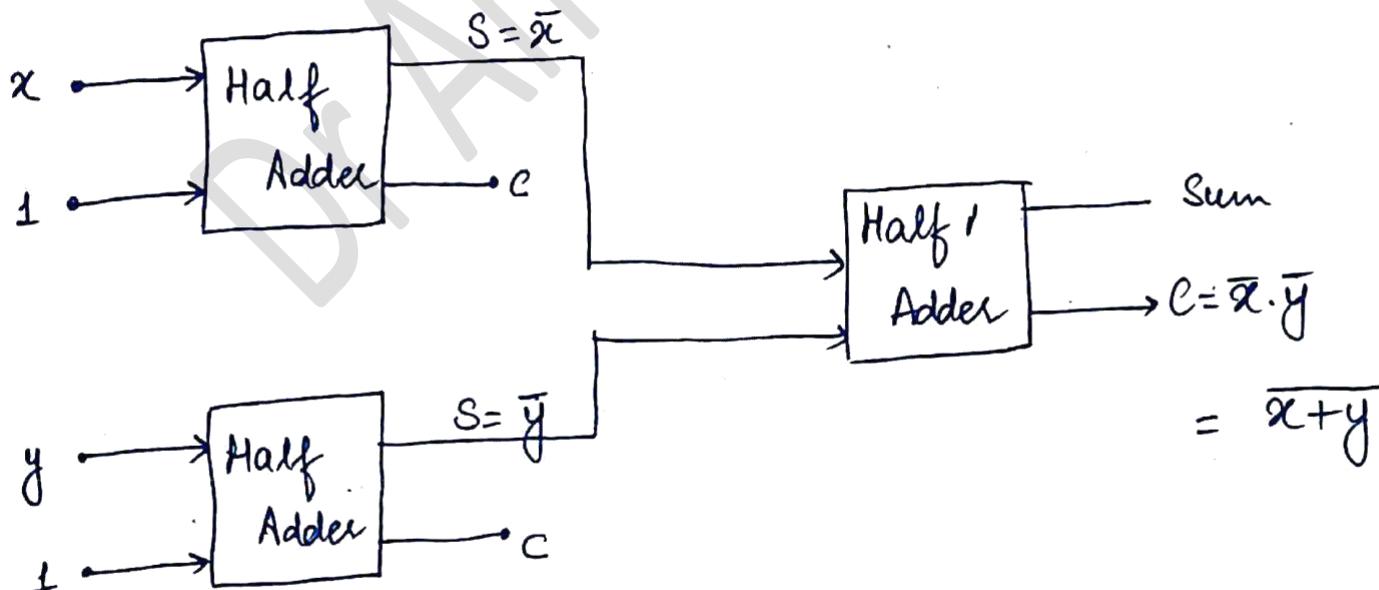
Using Half Adder,



for NOR Gate,



Using Half Adder,



→ full Adder

- Addition of three binary digits.
- Truth Table

<u>Input</u>			<u>Output</u>	
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

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

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

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

$\downarrow z$ $\downarrow z$

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

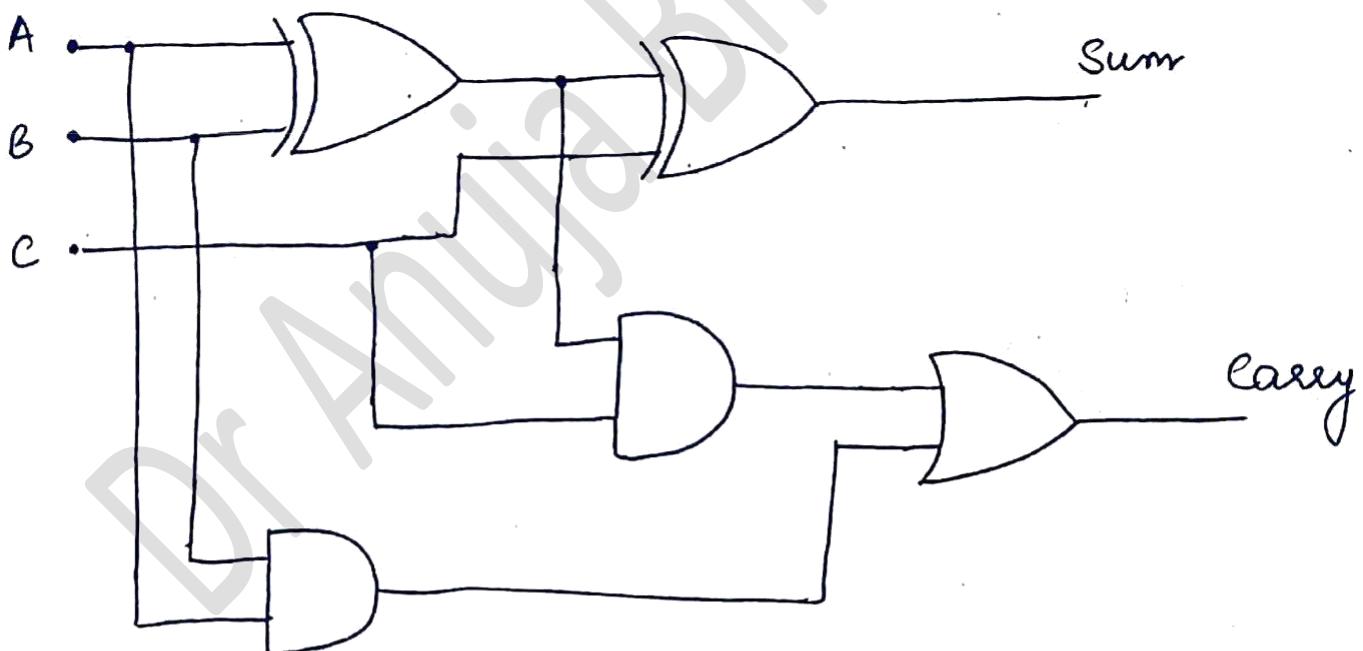
K-Map for Carry

-17

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

$$\begin{aligned}
 \text{Carry} &= \bar{A}BC + A\bar{B}C + ABC + AB\bar{C} \\
 &= (\bar{A}B + A\bar{B}) C + AB(C + \bar{C}) \\
 &= (A \oplus B) C + AB
 \end{aligned}$$

Logic Diagram



Binary Adder

- The digital circuit that generates the arithmetic sum of two binary numbers 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.

~~of~~

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

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

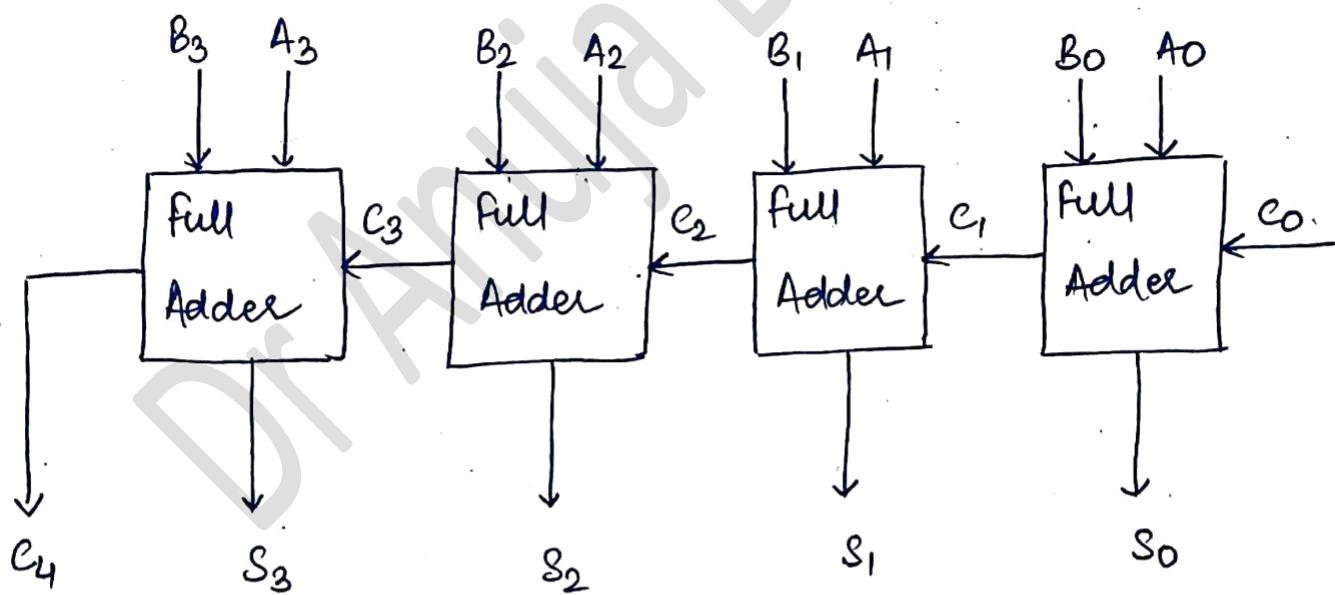


Fig 8 - 4 bit binary adder

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.

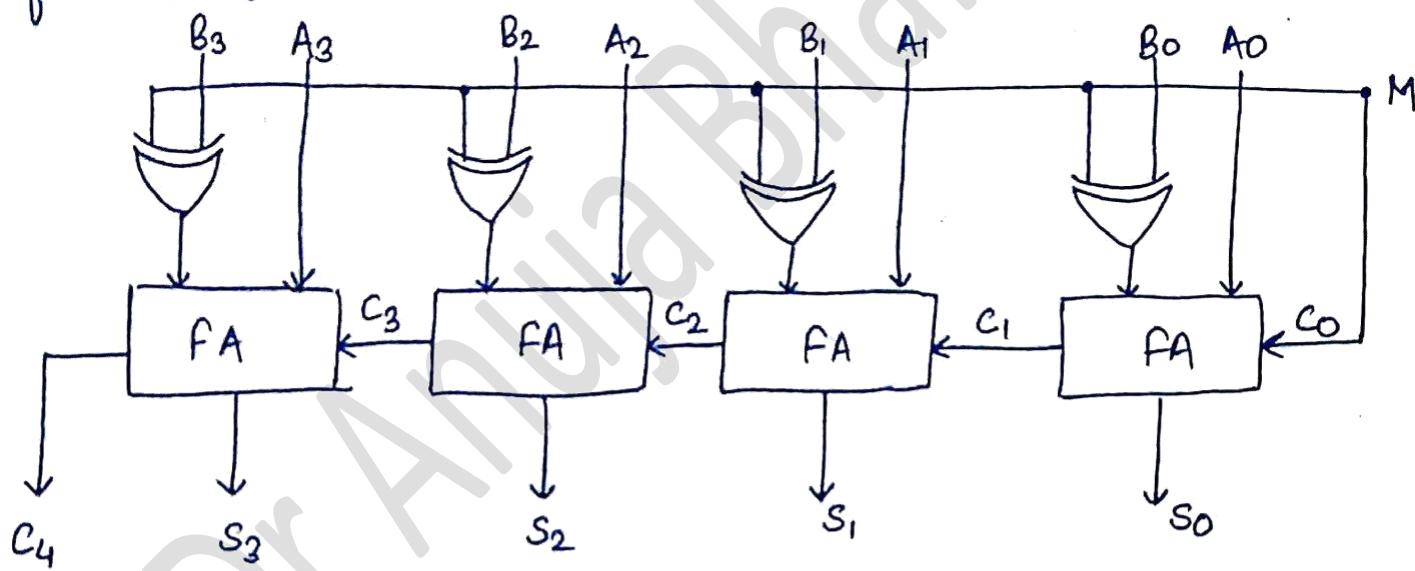


Fig :- 4 Bit Adder / Subtractor

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

$$\rightarrow \text{When } M=0, \quad B \oplus 0 = B$$

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

$$\rightarrow \text{When } M=1, \quad B \oplus 1 = \bar{B} \quad \& \quad C_0 = 1$$

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 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 propagate.
- Consider Truth Table of Full Adder

A	B	C_{in}	S	C_{out}	
0	0	0	0	0	No carry generation i.e $C_{out} = 0$
0	0	1	1	0	
0	1	0	1	0	
0	1	1	0	1	Carry Propagation i.e $C_{out} = C_{in}$
1	0	0	1	0	
1	0	1	0	1	
1	1	0	0	1	Carry generation i.e $C_{out} = 1$
1	1	1	1	1	

• From above table,

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

Therefore,

$$\begin{aligned} G_i &= A_i B_i C_{in} + A_i B_i \bar{C}_{in} \\ &= A_i B_i [C_{in} + \bar{C}_{in}] \\ &= A_i B_i \end{aligned}$$

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

$$\begin{aligned} P_i &= A_i \bar{B}_i + \bar{A}_i B_i \\ &= A_i \oplus B_i \end{aligned}$$

→ 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.

★★ 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,

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

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

$$\text{for } i=1, C_1 = G_1 + P_1 C_0$$

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

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

$$\text{where } G_1 = A_1 B_1, P_1 = A_1 \oplus B_1$$

$$\text{for } i=2, C_2 = G_2 + P_2 C_1$$

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

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

$$\text{where } G_2 = A_2 B_2, P_2 = A_2 \oplus B_2$$

$$\text{for } i=3, C_3 = G_3 + P_3 C_2$$

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

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

$$\text{where } G_3 = A_3 B_3, P_3 = A_3 \oplus B_3$$

- 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

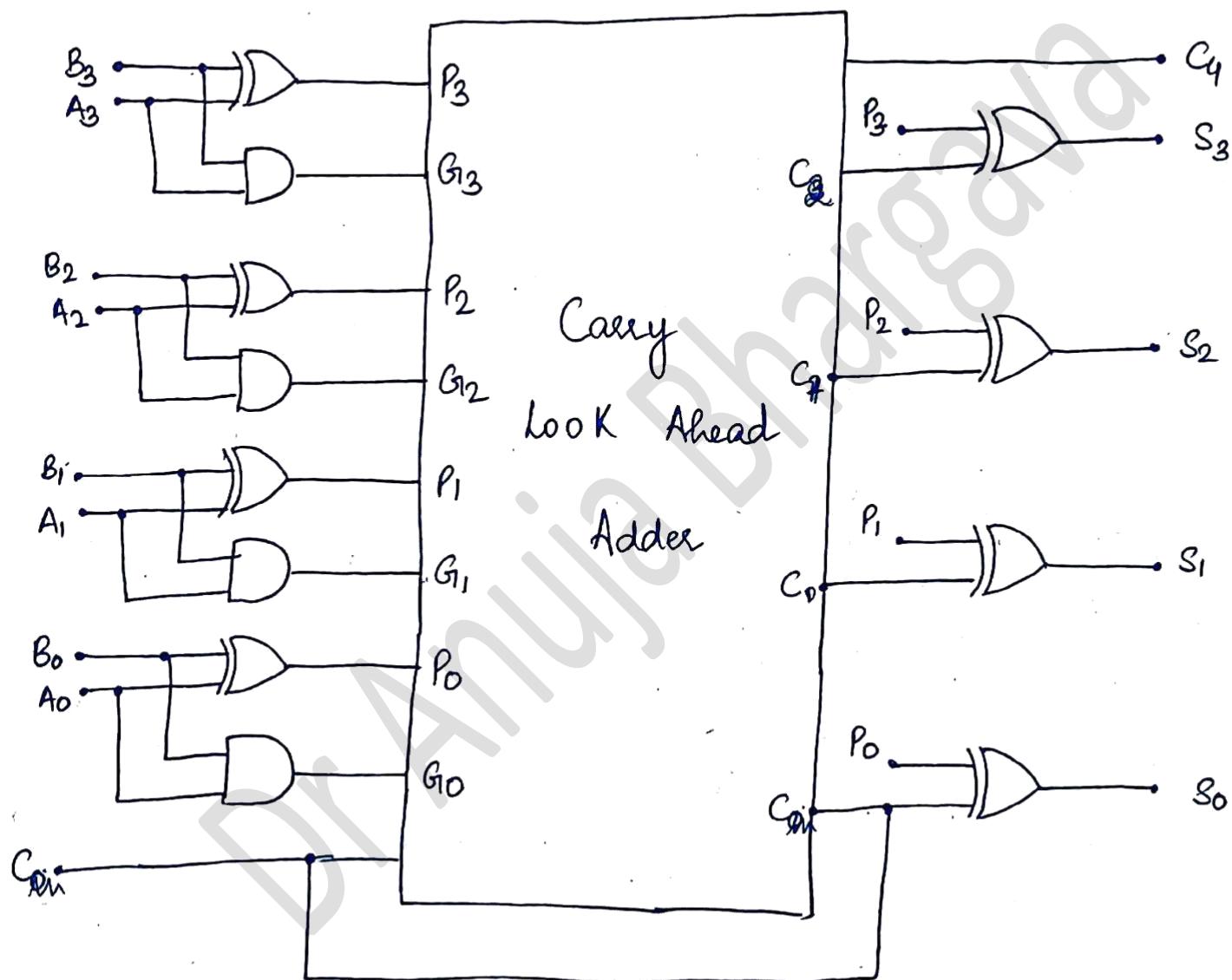
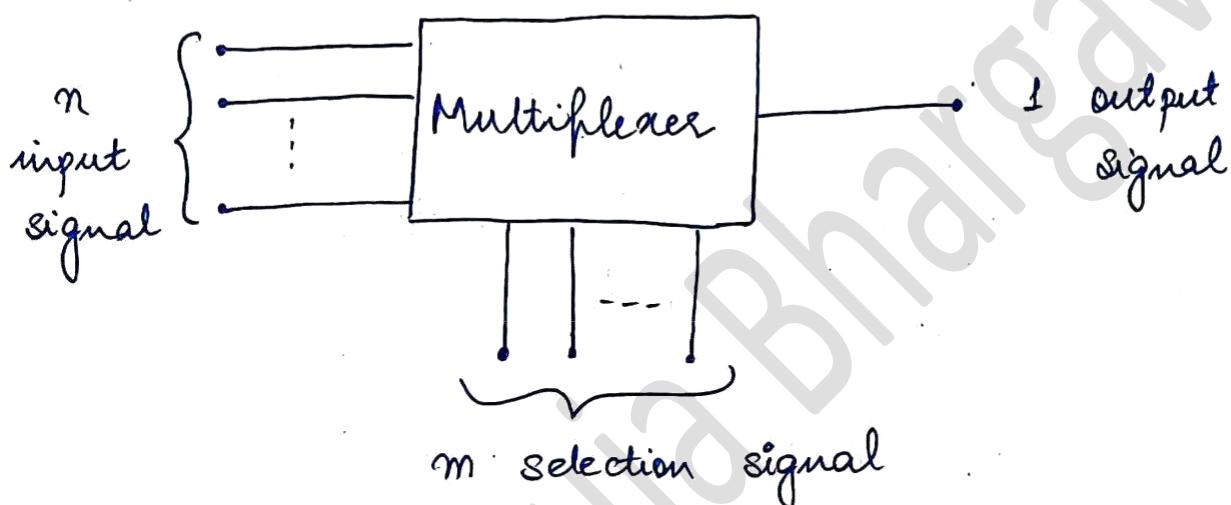


Fig :- 4 Bit Adder with Carry Look Ahead

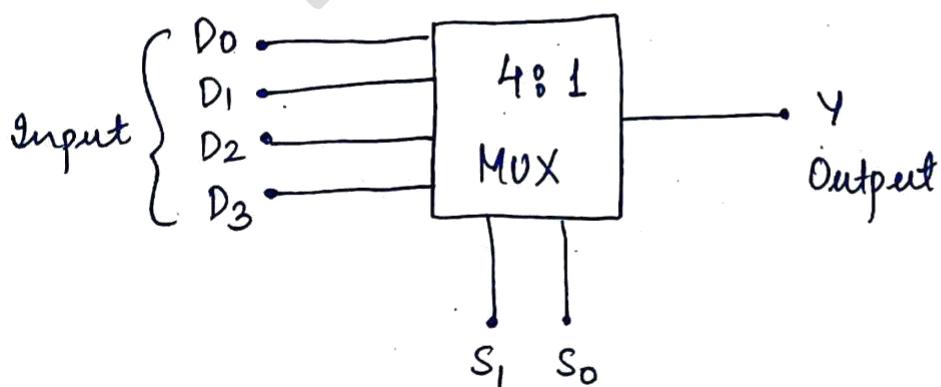
Multiplexers :-

- 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.

e.g 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

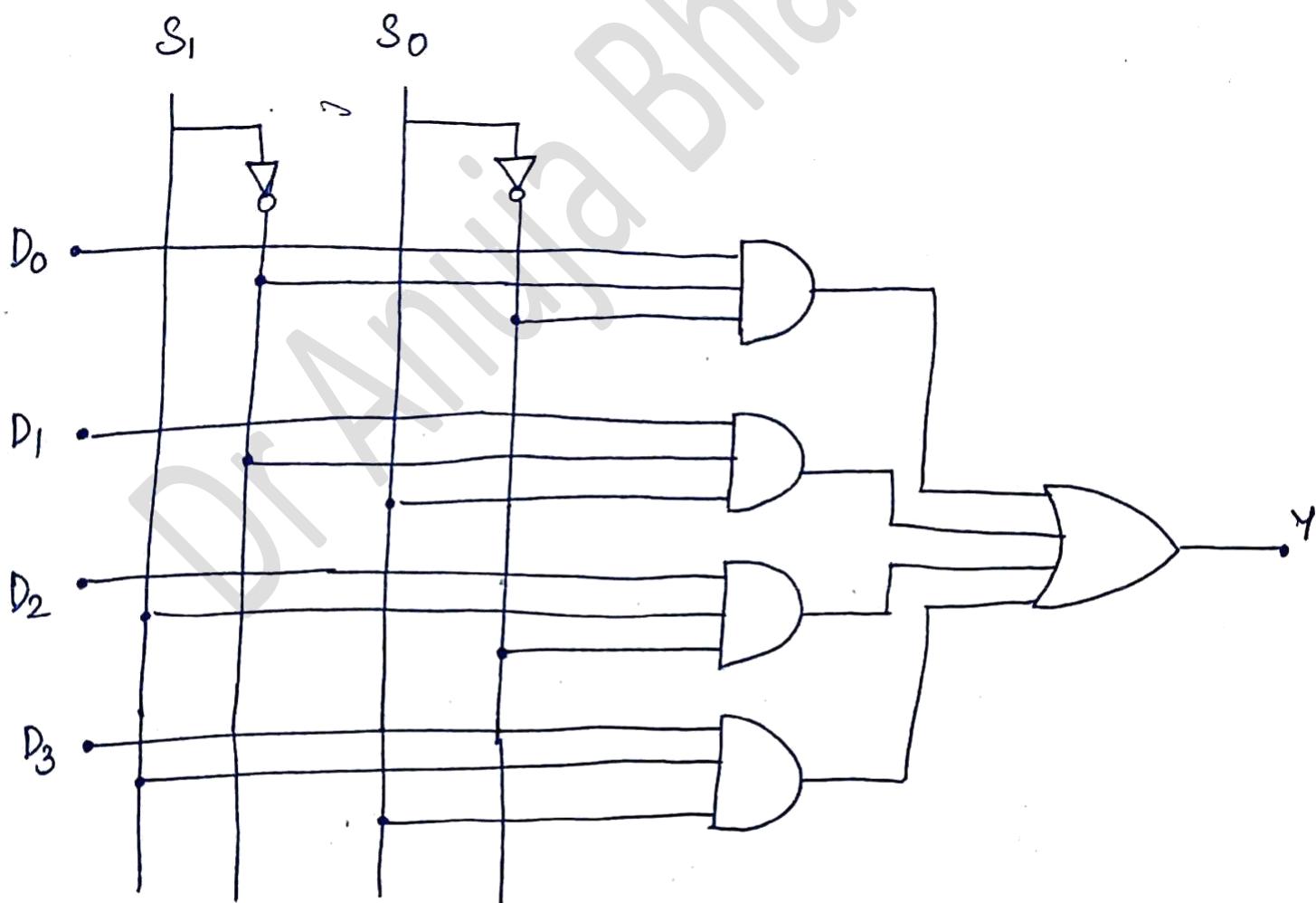
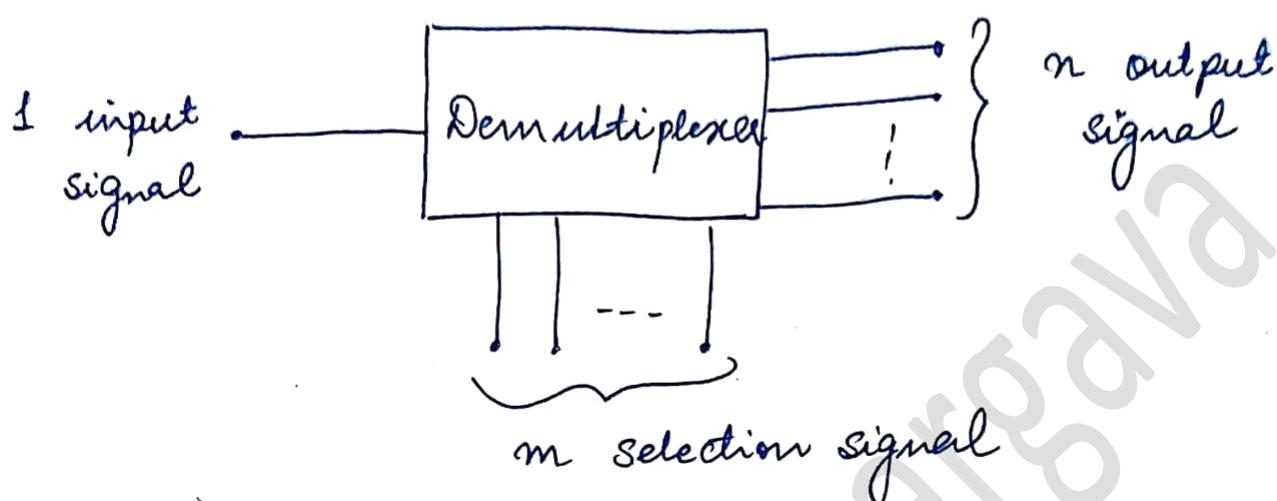


Fig :- 4:1 Multiplexer

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 D	Selection Signal		Output			
	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

$$y_0 = D \bar{S}_1 \bar{S}_0$$

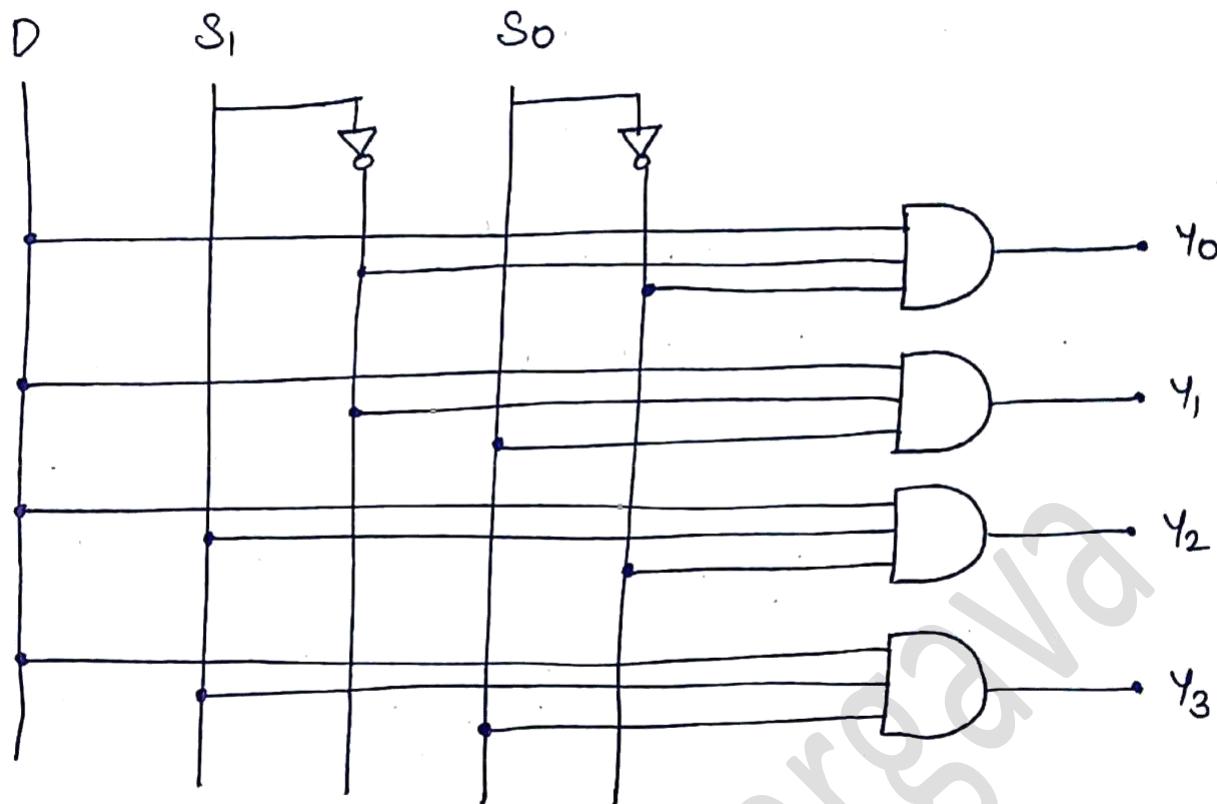
$$y_1 = D \bar{S}_1 S_0$$

$$y_2 = D S_1 \bar{S}_0$$

$$y_3 = D S_1 S_0$$

Logic Diagram

22



Difference between Multiplexer & Demultiplexer

Multiplexer

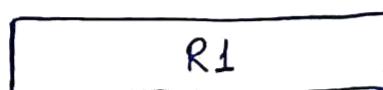
De-Multiplexer

- 1) It processes the digital information from various source into a single source. It receives digital information from single source and convert it into several sources.
- 2) It is known a Data Selector. It is known a Data Distributor.
- 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

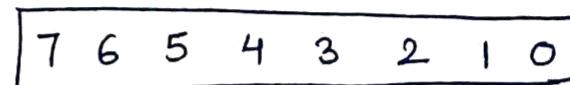
Register Transfer Language :-

- 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
 - RL - Processor Register

- Figure shows the representation of registers in block diagram form

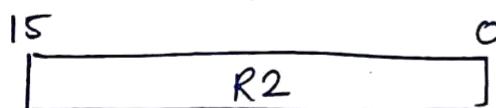


Register R

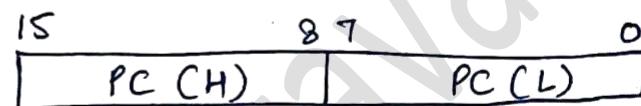


Showing individual bits

- 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.

- Bits 0-7 low byte \rightarrow PC(0-7) / PC(L)
- Bits 8-15 high byte \rightarrow 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 occurs using pre-determined condition
 $\text{if } (P=1) \text{ then } (R2 \leftarrow R1) \text{ or } P: R2 \leftarrow R1$

where P is control signal generated in control section.

Bus & Memory Transfer

- A more efficient scheme for transferring information between registers in a multiple register configuration is a common bus system.
- A bus structure consist af set of common lines, one for each bit of a register, through which binary information is transferred 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.

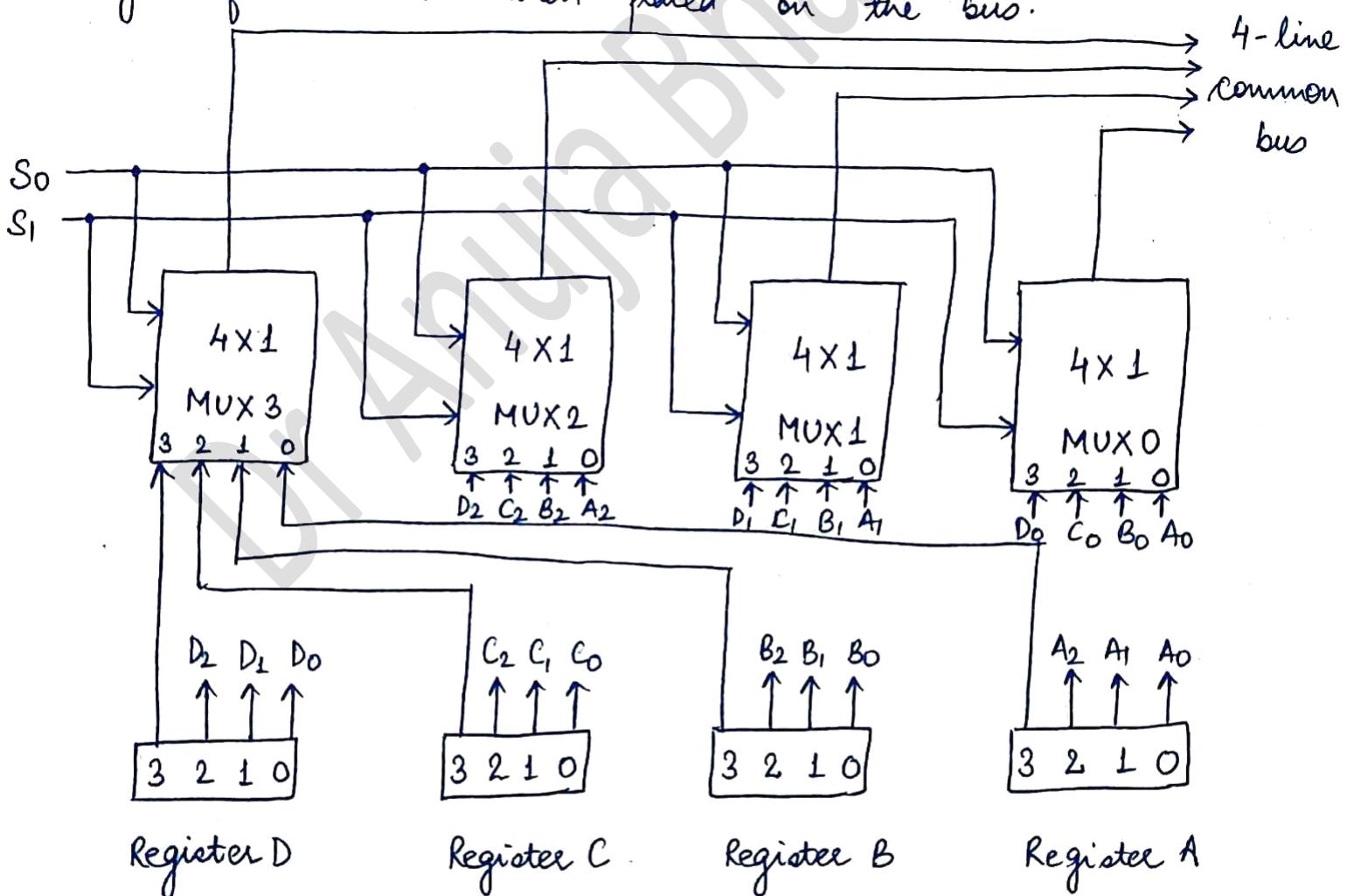


Fig :- Bus system for four register

- Above figure shows construction of a bus system for 24 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 receives the address from register is called address register (AR). The data is transferred to another register called data register (DR).

Read : $DR \leftarrow M[AR]$

Write : $M[AR] \leftarrow RI$

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?

Soln:- 16 registers = 2^4 registers

\therefore 4 selection lines are there to select 16 registers

b) What size of multiplexer are needed?

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

c) How many multiplexers are there in a bus?

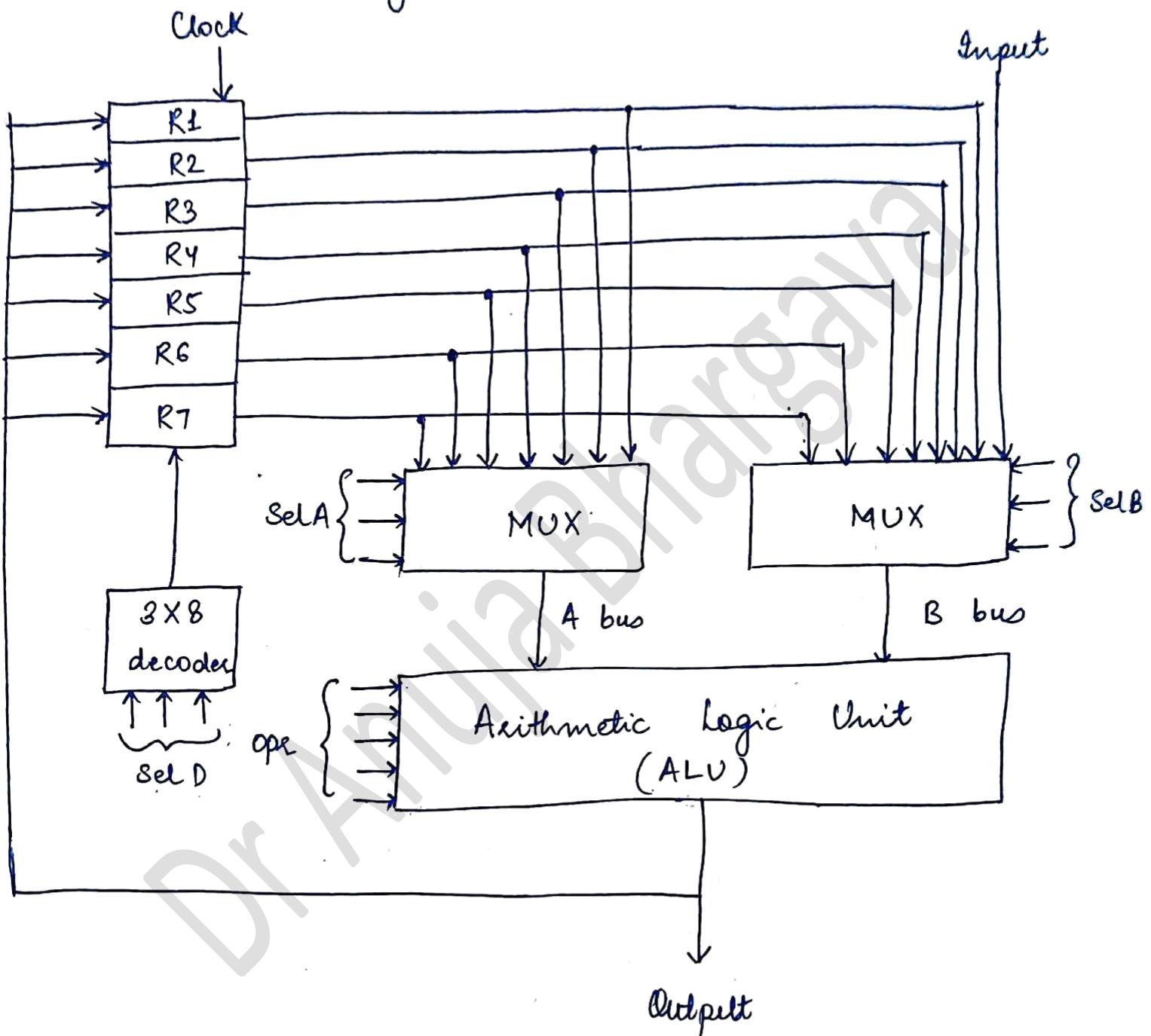
Soln:- Number of multiplexer = Number of bits in each register

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

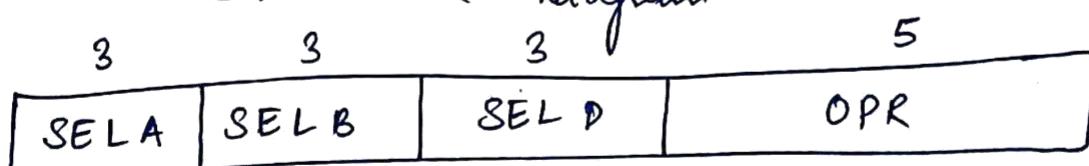
General Register Organization :-

- 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 efficient to store immediate values in processor registers.
- When a large number of registers are included in the CPU, it is more efficient to connect them through a common bus system.
- The registers communicate with each other not only for direct data transfer but also for performing microoperations.
- Therefore, it is necessary to provide a common unit that can perform all the arithmetic, logic and shift microoperations 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 microoperation to be performed.

- The result of microoperation is available for output data & also goes into the input of all the registers.
- The register that receives the information from the output bus is selected by decoder.



(a) Block Diagram



(b) Control Word

Fig :- Register Set with common ALU

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 (SEL D) : 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
0 0 0 0 0	Transfer A	TSFA
0 0 0 0 1	Increment A	INC A
0 0 0 1 0	Add A + B	ADD
0 0 1 0 1	Subtract A - B	SUB
0 0 1 1 0	Decrement A	DECA
0 1 0 0 0	AND A & B	AND
0 1 0 1 0	OR A & B	OR
0 1 1 0 0	XOR A & B	XOR
0 1 1 1 0	Complement A	COM A
1 0 0 0 0	Shift right A	SHR A
1 1 0 0 0	Shift left A	SHL A

eg

Microoperation	SEL A	SEL B	SEL D	Control Word
$R1 \leftarrow R2 - R3$	R2	R3	R1	0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 1
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	1 0 0 1 0 1 1 0 1 0 1 0 0 1 0 1
$R7 \leftarrow R1$	R1	-	R7	0 0 1 0 0 0 1 1 1 0 0 0 0 0 0 0
Output $\leftarrow R2$	R2	-	-	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1

Stack Organization :-

27

- 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 ~~pop~~ ~~removal~~
~~executing~~ 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.

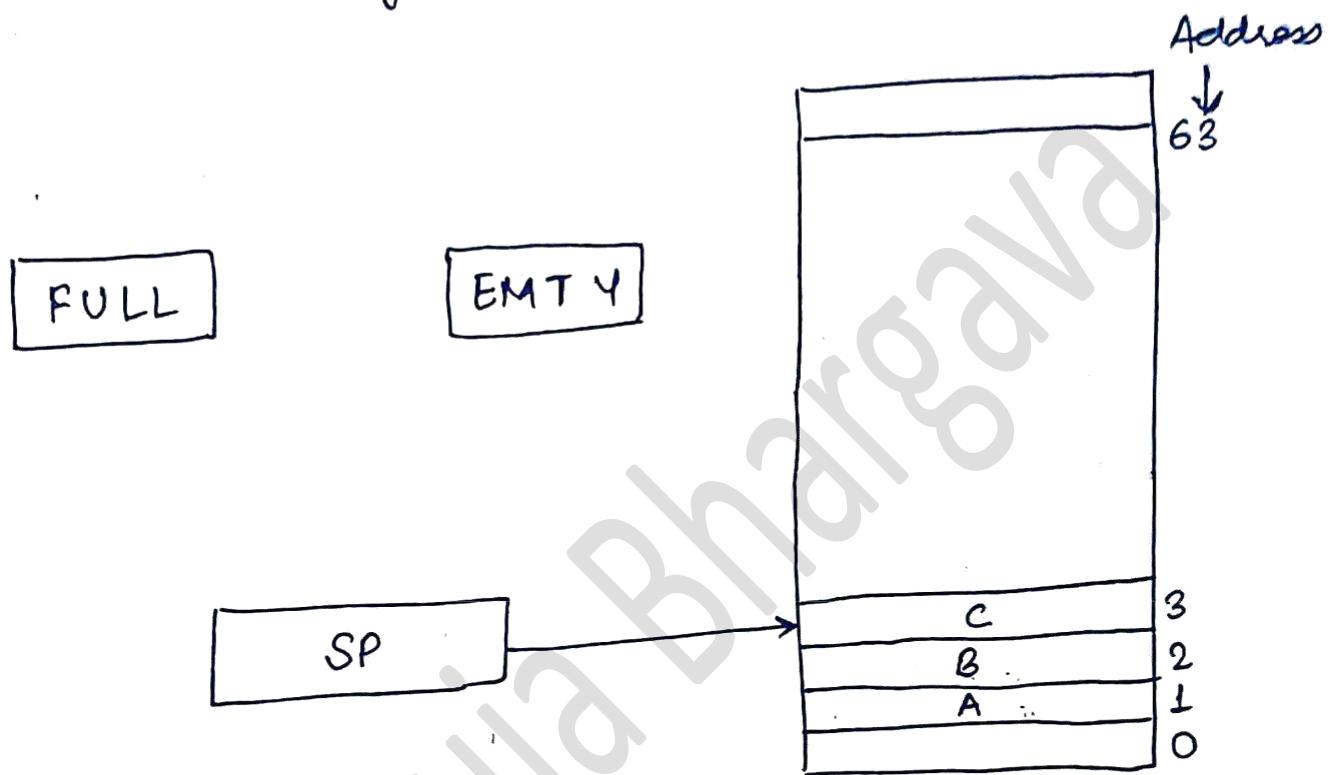


Fig :- Block Diagram of 64-word stack

- 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

$SP \leftarrow SP + 1$

Increment stack pointer

 $M[SP] \leftarrow DR$

Write item on top of stack

If $(SP=0)$ then $(FULL \leftarrow 1)$

Check if stack is full

 $EMTY \leftarrow 0$

Mark the stack not empty

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

 $DR \leftarrow M[SP]$ Read item from top of stack

 $SP \leftarrow SP - 1$ Decrement stack pointer

If $(SP=0)$ then $(EMTY \leftarrow 1)$ Check if stack is empty

 $FULL \leftarrow 0$ 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 referred 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 operators.

Q The expression $A * B} + C * D$

a) in Reverse Polish Notation

$$AB * CD * +$$

** $(A * B) CD * +$

$$(A * B) (C * D) +$$

$$(A * B) + (C * D)$$

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

$$AB + DE + C * F + *$$

Evaluation of Arithmetic Expressions

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

Q The arithmetic expression

$$(3 * 4) + (5 * 6)$$

In Reverse Polish Notation

$$34 * 56 * +$$

Stack Operation

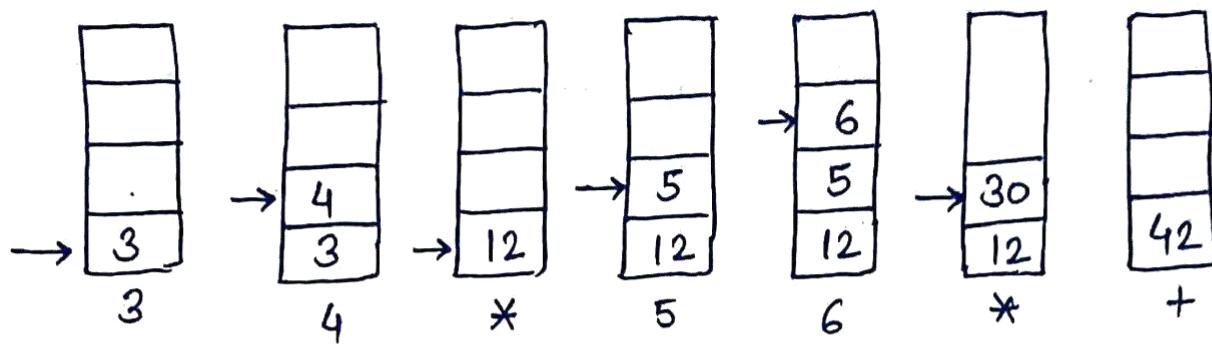


Fig 8:- Stack Operator to evaluate $3*4 + 5*6$

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

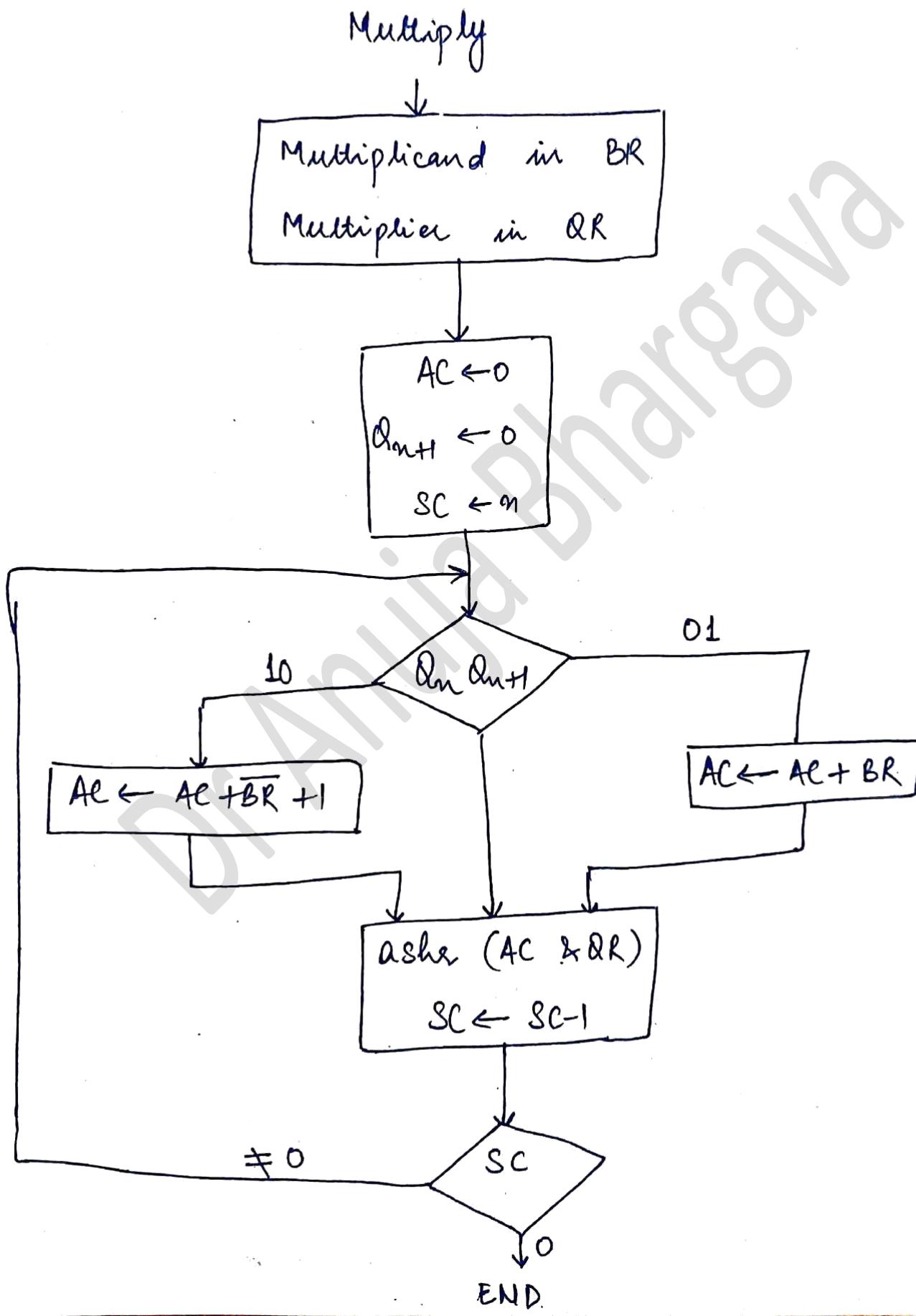
e) $(A * B) + (C * D) + (E * F)$

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

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

Booth Algorithm :-

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



$B \rightarrow$ Multiplicand

$A \rightarrow$ Partial Product, Initially zero

$Q \rightarrow$ Multiplier

$Q_{out} \rightarrow$ Initially zero

$Q_m \rightarrow$ LSB of Q

Q_m Q_{out}

0 0 ash

0 1 add + ash

1 0 sub + ash

1 1 ash

ADD $\rightarrow A + B$

Sub $\rightarrow A + \bar{B} + 1$

eg $(7) \times (-5)$

$$B = 7 \quad + 7 \rightarrow 0111 \text{ (B)} \quad + 5 \rightarrow 0101$$

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

Q_m	Q_{out}	Operation	A	Q	Q_{out}	SC
-	-	-	0000	1011	0	100(4)
1	0	sub	$\begin{array}{r} + 1001 \\ \hline 1001 \end{array}$	1011	0	-
		ash	1100	1101	1	011(3)
1	1	ash	1110	0110	1	010(2)
0	1	add	$\begin{array}{r} 0111 \\ + 1011 \\ \hline 0110 \end{array}$	0110	1	
		(Carry omit)				
		ash	0010	1011	0	001(1)
1	0	sub	$\begin{array}{r} 1001 \\ - 1011 \\ \hline 1011 \end{array}$	1011	0	
		ash	1101	1101	1	000(0)

31

Answer = A Q

$$= \begin{array}{r} 11011101 \\ \uparrow \quad \underbrace{\hspace{1cm}}_{\text{-ve Mag.}} \\ \downarrow \quad 2's \end{array}$$

$$\left(\begin{array}{r} 0100011 \\ -35 \end{array} \right)$$

eg $(-15) \times (-8)$

$$B + 15 \rightarrow 01111 (\bar{B}+1) \quad Q + 8 \rightarrow 01000$$

$$-15 \rightarrow 10001 \quad B \quad -8 \rightarrow 11000 (Q)$$

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	$\frac{01111}{01111}$	00011	0	
		ashr	00111	10001	1	001
1	1	ashr	00011	11000	1	000

$$\text{Ans} = A\bar{Q} = 0001111000 = +120$$

$$\text{eg } (-9) \times (-13)$$

$$+9 \rightarrow 01001 (\bar{B}+1) + 13 \rightarrow 01101$$

$$-9 \rightarrow 10111 (\bar{B}) - 13 \rightarrow 10011 (\bar{Q})$$

Q_m	Q_{m+1}	Operation	A	Q	Q_{m+1}	SC
			00000	10011	0	101
-	0	sub	$\begin{array}{r} 01001 \\ - 01001 \\ \hline 0 \end{array}$	10011	0	-
-	1	ashe	00100	11001	1	100
-	1	ashe	00010	01100	1	011
0	1	add	$\begin{array}{r} 10111 \\ + 11001 \\ \hline 11000 \end{array}$	01100	1	.
		ashe	11100	10110	0	010
0	0	ashe	11110	01011	0	001
-	0	sub	$\begin{array}{r} 01001 \\ - 10011 \\ \hline 01011 \end{array}$	01011	0	.
		Carry				.
		ashe	00011	10101	1	000

$$\text{Answer} = AQ$$

$$= 0001110101$$