UNIT-2 Contents

1. Arithmatic Addition, Substraction of signed numbers.
2. Addition or substraction logic unit.
3. Design of fast adders
4. Carry look ahead addition
5. Multiplication of positive numbers
6. Signed operand multiplication
7. Booth algorithm
8. Fast multiplication
9. Bitpair recording multipliers
10. Carry save addition of summands
11. Integer division
12. Floating point number and operation -
    i) I-EEE Standard for floating point numbers
    ii) Arithmatic operations on floating point numbers
    iii) Guard bits and truncketion
    iv) Implementing floating point operations

Addition or Subtraction Logic Unit:
(Arithmatic Logic Unit - ALU)

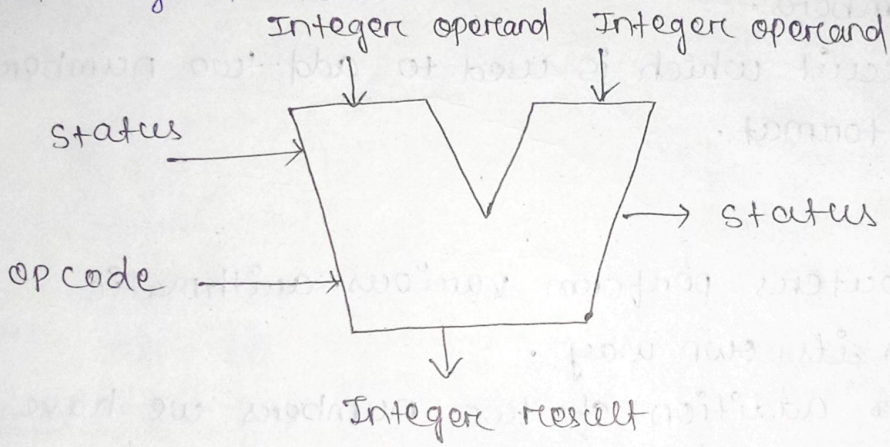
→ Computer's processing unit contains an arithmatic and logic unit (ALU) which is capable of performing logical operations (e.g→ AND, OR, NOT) etc. In addition to the arithmatic operations e.g→ addition, substraction etc.

→ The control unit supplies the data required by the ALU from memory or from input devices and directs ALU to perform a specific operation based on the ~~in~~ the instructions fetched from the memory.

→ ALU can be said as the calculator of the computer.

→ It is a major component of the CPU.

→ It performs all operations related to arithmatic and logical needs.



(Diagram of Arithmatic logic unit)

→ When the operations become more complex then the ALU will also become more and more expensive & also requires more and more space in the CPU.

→ It generates more heat.

→ Hence it is constructed with powerful and fast architecture.

→ It consumes more energy.
→ It is costly as per the complexity nature.
→ Operations performed by ALU.
  i) Logical Operations:- AND, OR, XOR, NAND, NOR
  ii) Bit shifting operations :- At the time of multiplication and division operations to be performed the bit shifting requirements occur either to right shift or left shift by a certain number of place.
  iii) Arithmatic operations :- Arithmatic operations like addition, substraction, multiplication etc. are performed.

Binary Adder :-

→ Adder is a kind of calculator used to add two binary numbers.
→ It is a circuit which is used to add two numbers in binary format.

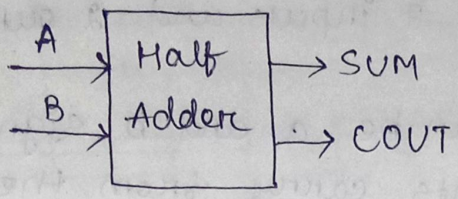
Half Adder -

→ Digital computers perform various arithmatic operations in its own way.
→ Normally for addition of two numbers we have 4 possible elementary operations.

	SUM	carry
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1

→ The logic circuit which performs addition of two binary digits and produces a sum and a carry output is called as a half adder.

→ A half adder inputs two bits only A and B and produces sum value SUM and carry value COUT.

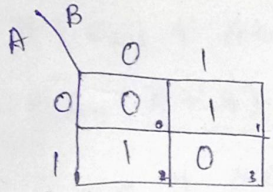


(Functional diagram for half adder)

Truth Table :

A	B	SUM	COUT
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

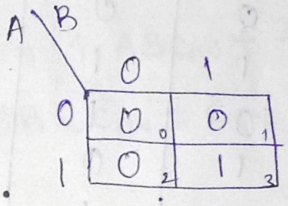
K-map for SUM



01 + 10

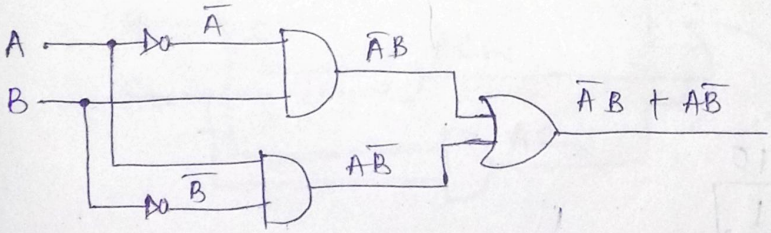
SUM = $\bar{A}B + A\bar{B}$

K-map for COUT

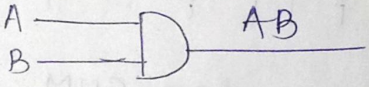


COUT = 11 = AB

Circuit Diagram



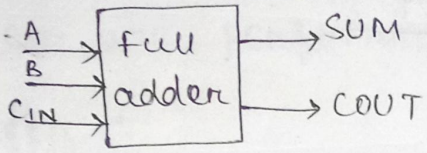
Circuit Diagram



Full Adder -

- → A full adder is a combinational circuit that takes a carrey received from previous bit position.
- → It perforems addition of 3 bits two significant bit and a previous carry.
- → Full adder consists of 3 inputs and 2 outputs. (SUM and COUT)
- → 2 input variables can be a and b significant bits and C_{IN} represents carry from the previous lower significant bit position.

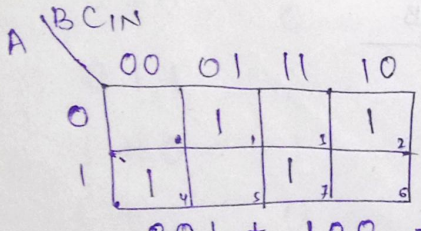
Functional Diagram



Truth Table :-

A	B	C_{IN}	SUM	COUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

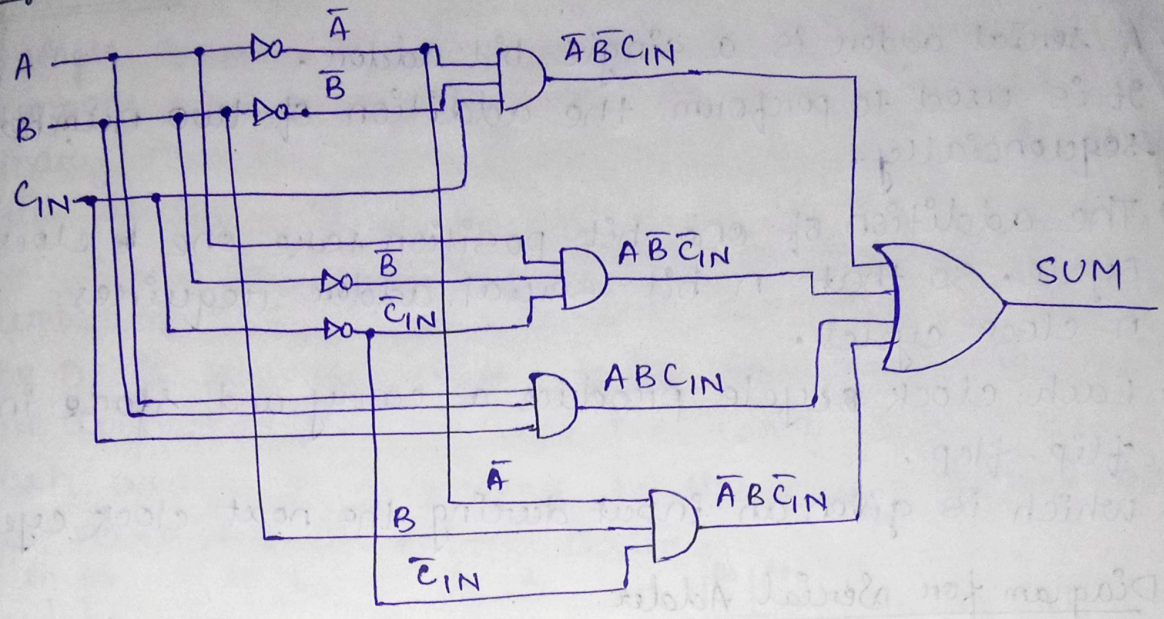
K-map for SUM



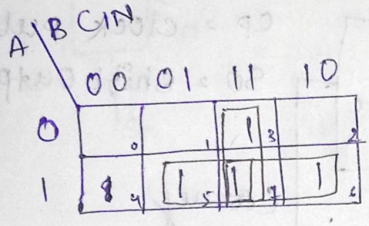
001 + 100 + 111 + 010

$SUM = \bar{A}\bar{B}C_{IN} + A\bar{B}\bar{C}_{IN} + ABC_{IN} + \bar{A}B\bar{C}_{IN}$

Logic gate:



K-map for COUT :-



pair 1 = 011 + 111 = $\bar{A}BC_{IN} + ABC_{IN}$

pair 2 = 101 + 111 = $A\bar{B}C_{IN} + ABC_{IN}$

pair 3 = 111 + 110 = $ABC_{IN} + AB\bar{C}_{IN}$

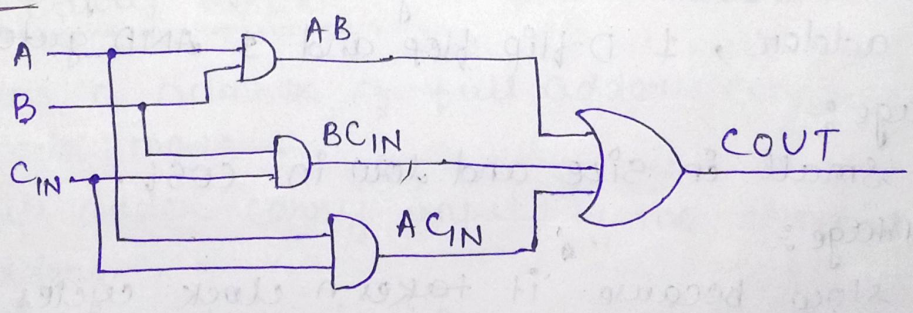
$\bar{A}BC_{IN} + ABC_{IN} + A\bar{B}C_{IN} + ABC_{IN} + ABC_{IN} + AB\bar{C}_{IN}$

$= BC_{IN}(\bar{A} + A) + AC_{IN}(\bar{B} + B) + AB(\bar{C}_{IN} + C_{IN})$

$= BC_{IN} + AC_{IN} + AB$

$= AB + BC_{IN} + AC_{IN}$

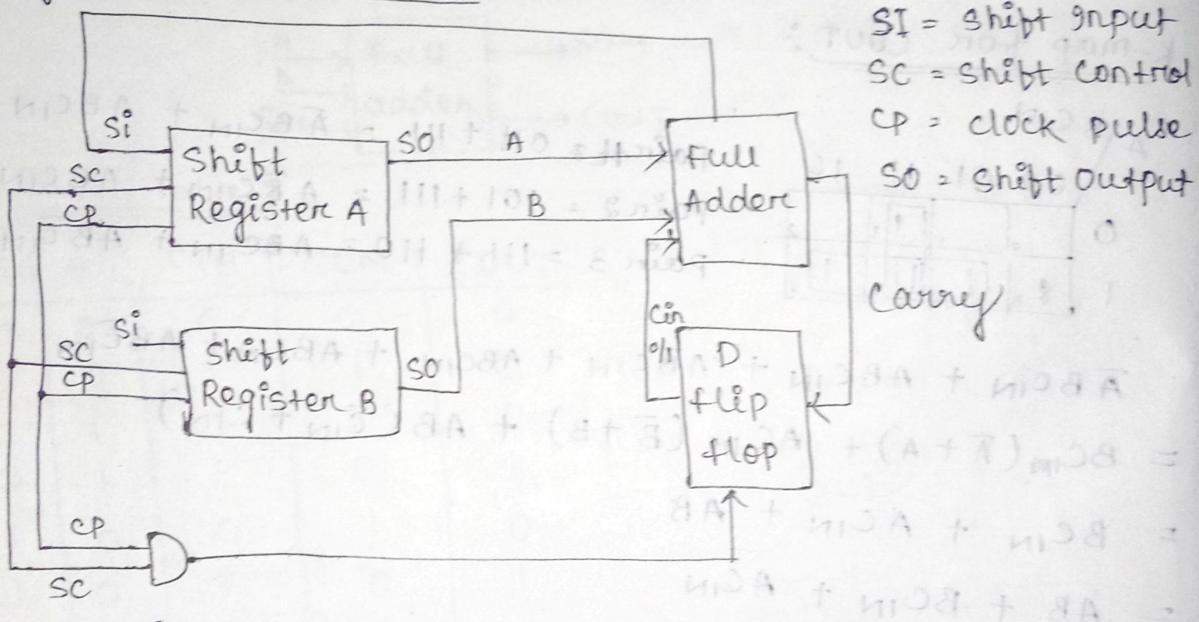
Logic gate:



Serial Adder :

→ A serial adder is a single bit adder.
→ It is used to perform the addition of two numbers sequencially.
→ The addition of one bit position take one b clock cycle. So that n bit serial adder requires n-clock cycles.
→ Each clock cycle produce a carry and store in a flip-flop.
→ which is given an input during the next clock cycle.

Diagram for Serial Adder



SI = Shift input
SC = Shift control
CP = clock pulse
SO = Shift output

→ The serial adder uses a right shift register, 1 full adder, 1 D-flip-flop and 1 AND gate.

Advantage :-
It is small in size and low in cost.

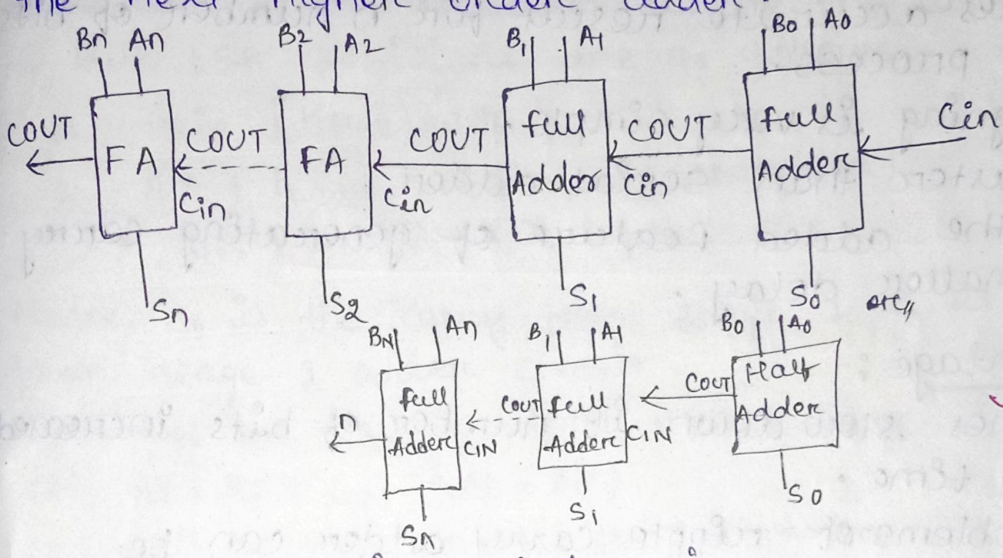
Disadvantage :
It is slow because it takes n clock cycles for completion of addition of n bit numbers. Because it takes n-clock cycle.

Parallel Adder :-

→ A single full adder is capable of adding two 1 bit numbers and an input carry. In order to add binary numbers with more than 1 bit additional full adders must be used.

→ A n bit parallel adder can be constructed using number of full adder circuits connected in parallel.

→ The n bit parallel adder using full adder circuits are connected in cascade i.e. carry output of each adder is connected to the carry input to the next higher order adder.



Parallel adder is of 2 types :

i) Ripple carry adder

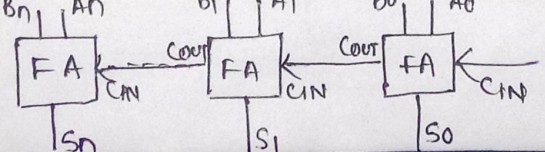
ii) carry Look Ahead adder

i) Ripple Carry Adder :

→ There are n number of full adders connected into a cascade mode.

→ Each full adder carry input to the ~~circuit~~ next step adders.

Block diagram of n bit ripple carry adder :-



→ From the above diagram it can be clear that each stage adder is a combinational circuit in which the current output depends on the current inputs and generates 2 relevant outputs i.e. Sum and carry.
→ Ripple carry adder is a simple form of parallel adder where the carry out of each full adder is connected to the carry (in) of the next full adder.
→ Hence the total delay time of the adder is the time it would take for a carrey to ripple through all bit pair full adder.

Advantage :-

→ It provides accurate result for n number of bits addition process.
→ Its designing is very simple.
→ It is faster than serial adder.
→ It has the added feature of generating carrey with smaller delay.

Disadvantage :-

→ It becomes slow down if number of bits increased at run time.
→ The problems of ripple carry adder can be solved using carry look ahead adder with extra circuitry.

Cascading ripple carry adder :-

Using cascading ripple carrey adder with any bit-length, construct n bit adder using two number of n bit adders of ripple time and then both the modules can simoulteniously perform addition process and allow the carry propogation module to the next module.

Carry Look Ahead Adder :-

→ A carry look ahead adder is a high speed adder which allows a special strategy of quick carry generation form.

→ At this generation stage without waiting for the carries of the previous pages of the circuit the adder involves early determination of carry input stage without using carry output from the previous stage, it directly determines the carry signals.

→ consider an n-bit adder where the carry output of two LSB positions are as follows -

$C_0 = A_0 B_0$ [Half Adder]

$C_1 = A_1 B_1 + B_1 C_{IN} + A_1 C_{IN}$ [Full Adder]

$= A_1 B_1 + C_{IN} (A_1 + B_1)$

Hence C_0 is the carry from stage 1 C_1 is the carry from stage 2 adder circuit.

Similarly the carry output from stage i will be

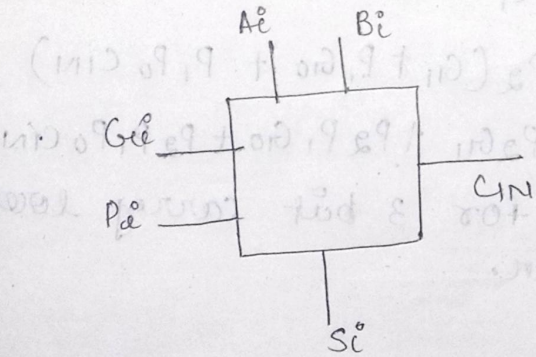
$C_i = A_i + B_i + C_{IN} (A_i + B_i)$

→ C_{i-1} will be the carry output from stage $i-1$.

Hence the role of generator G and propagator P can be identified in the following manner.

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

The symbolic diagram :-

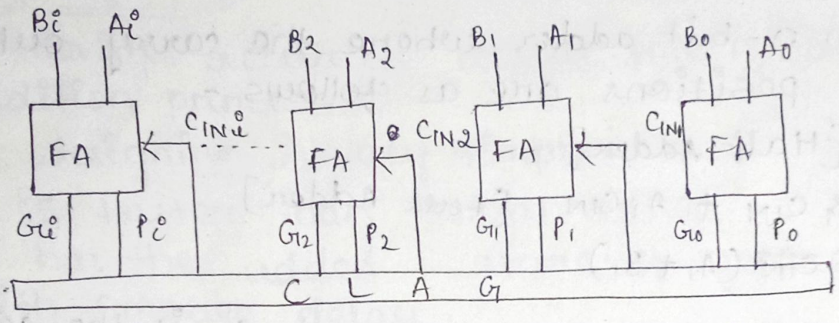


→ When expanding is ~~stage~~ statement for all the Carries it is confirmed that all the carries in Parallel adder can be determined directly from the input data with the given pattern without actually waiting for the carry of the previous bit position through the parallel adder.

→ It means that every stage can perform independent of other stage.

→ This procedure increases the speed of addition.

→ But it has several gates with multiple inputs.



(Carry Look Ahead Generator)

Equations :-

$$C_0 = G_{10} + P_0 C_{IN}$$

At stage 1 = $C_1 = G_{11} + P_1 C_0$

$= G_1 + P_1 (G_{10} + P_0 C_{IN})$

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

↳ This is the expression for 2 bit carry look ahead adder.

At stage 2 = $C_2 = G_{12} P_2 C_1$

$= G_{12} + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_{IN})$

$= G_{12} + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{IN}$

↳ Expression for 3 bit carry look ahead adder.

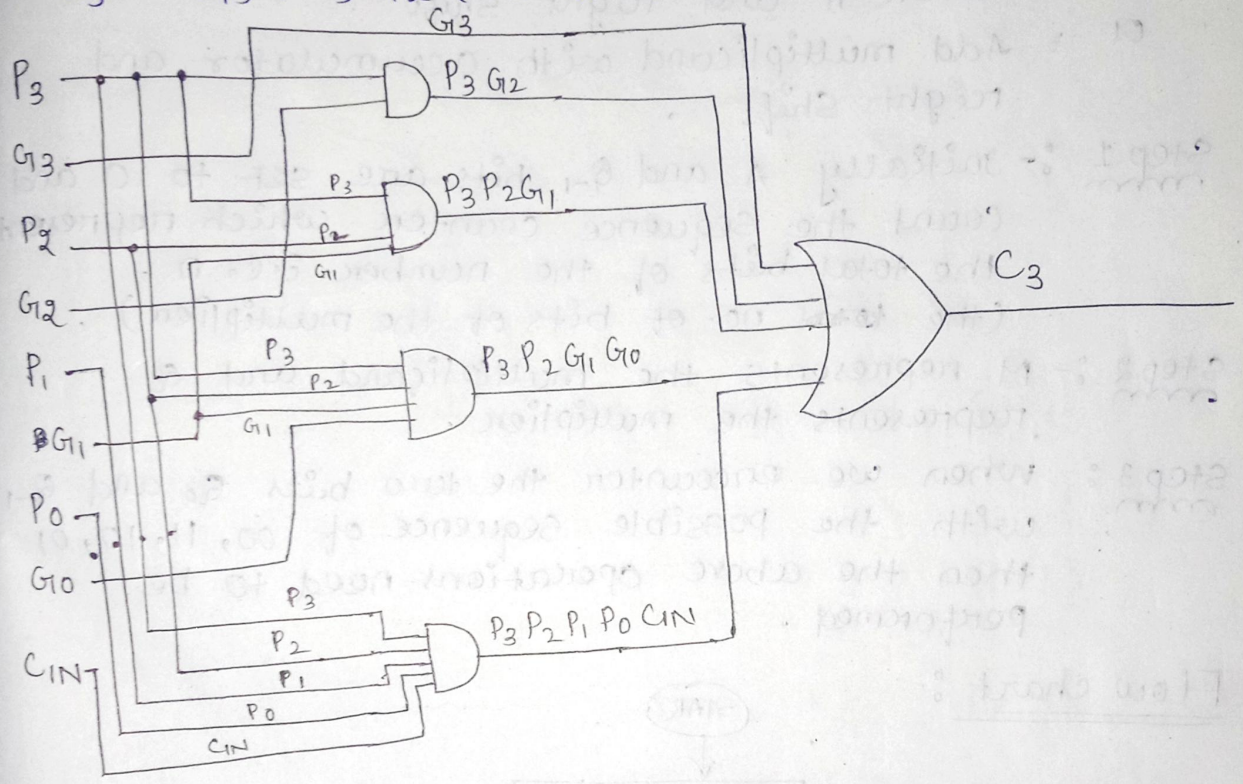
At stage $n = C_n = G_n + P_n G_{n-1} + P_n P_{n-1} G_{n-2} + \cdots + P_n P_{n-1} P_{n-2} C_{IN}$

[n bit carry look ahead adder]

construct the circuit diagram for 4 bit carry look ahead adder :-

The expression for the carry look ahead adder at stage 3 -

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 G_1 G_0 + P_3 P_2 P_1 P_0 C_{IN}$$



Booth's Multiplication :-

Steps involved in Booth's multiplication :-

→ Use the LSB (current LSB) and previous LSB to determine the algorithmic operation.

→ The possible algorithmic operations can be

00, 11 } where no change only right shift is required.

10 : then subtract multiplicand from the accumulator A and right shift.

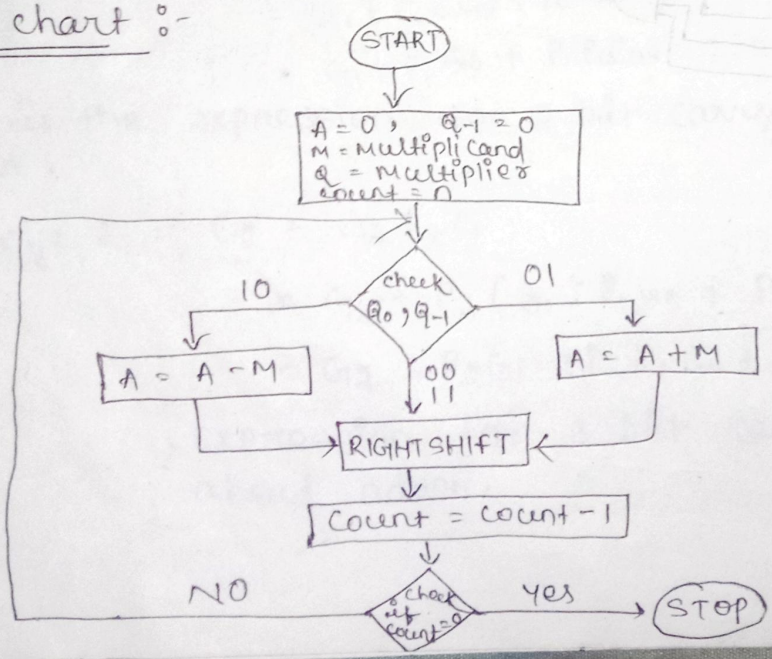
01 : Add multiplicand with accumulator and right shift.

Step 1 :- Initially A and Q_{-1} bits are set to 0 and count the sequence counter which represents the total bits of the number i.e. n. (the ~~total~~ no. of bits of the multiplier).

Step 2 :- M represents the multiplicand and Q represents the multiplier.

Step 3 :- When we encounter the two bits Q_0 and Q_{-1} with the possible sequence of 00, 11, 10, 01 then the above operations need to be performed.

Flow chart :-



Rules for MSB :-

- For the positive number multiplicant at MSB of A put LSB of Q of the previous cycle.
- For the negative number multiplicant at MSB of A put MSB of A of previous cycle.

1) Multiply -9×6

$M = -9 = 10111$

$Q = 6 = 00110$

$M = 10111$

$-M = 01000 + 1$

$01001 \ (-M)$

M	A	Q	Q-1	operation	count	
10111	00000	00110	0	Initialisation		00
	00000	00011	0	Right shift	5	10
	00000 + 01001 = 01001			A = A − M		A = A + (−M)
	00100	10001	0	Right shift	4	
	00010	01000	1	Right shift	3	11
	00010 + 10111 = 11001			A = A + M		01
	11100	10100	0	Right shift	2	
	11110	01010	0	Right shift	1	00
	11110	01010			0	

Check :- $-9 \times 6 = -54$

$54 = 0000110110$

$-54 = 1111001001 + 1$

1111001010 //

$2 \mid 54 \ — 0$
$2 \mid 27 \ — 1$
$2 \mid 13 \ — 1$
$2 \mid 6 \ — 0$
$2 \mid 3 \ — 1$
$2 \mid 1 \ — 1$
0

3) Multiply -5×4

Multiplicand = M = -5 = 11011

Multiplier = Q = 4 = 00100

M = 11011

-M = 00100 + 1 = 00101

M	A	Q	Q-1	operations	count	
11011	00000	00100	0	Initialisation	5	00
	00000	00010	0	Right shift		00
	00000	00001	0	Right shift	4	10
	00000 + 00101 = 00101	00001	1	A = A - M	3	A = A + (-M)
	00010	10000	1	Right shift		01
	00010 + 11011 = 11101	10000	0	A = A + M	2	
	11101	11000	0	Right shift		00
	11111	01100	0	Right shift	1	
	11111	01100			0	

Check $-5 \times 4 = -20$

20 : 00000 10100

-20 : 11111 01011 + 1

11111 01100 = -20

2 | 20 - 0
2 | 10 - 0
2 | 5 - 01
2 | 2 - 0
2 | 1 - 1
0

3) Multiply -13×6

Multiplicand = M = −13 = 10011

Multiplier = Q = 6 = 00110

M = 10011

−M = 01100 + 1 = 01101

M	A	Q	Q-1	operation	count	
10011	00000	00110	0	Initialisation	5	00
	00000	00011	0	Right shift		10
	00000 + 01101 = 01101			A = A − M	4	11 A = A+(−M)
	00110	10001	1	Right shift		
	00011	01000	1	Right shift	3	01
	00011 + 10011 = 10110			A = A+M	2	A = A+M
	11011	00100	0	Right shift		00
	11101	10010	0	Right shift	1	
	11101	10010		Q	0	

Check: $-13 \times 6 = -78$

78 = 0001001110

−78 = 1110110001 + 1 = 1110110010 = −78

2 | 78 — 0
2 | 39 — 1
2 | 19 — 1
2 | 9 — 1
2 | 4 — 0
2 | 2 — 0
2 | 1 — 1
0

4) 14 x -5

Multiplicand = M = 14 = 01110 $\quad -M = \begin{array}{r} 10001 \\ \quad\quad 1 \\ \hline 10010 \end{array}$

Multiplier = Q = -5 = 11011

M	A	Q	Q-1	Operation	count	
01110	00000	11011	0	Initialisation	5	10
A	00000			A = A - M		A + (-M)
-M	10010			Right Shift		
	10010	11011				
	11001	01101	1			11
	11100	10110	1	R.S	4	01
A	11100			A = A + M	3	A = A + M
M	01110					
discard	01010	10110		R.S		
	00101	01011	0	A = A - M	2	10
A	00101					A + (-M)
-M	10010			R.S		
	10111	01011				11
	11011	10101	1			
	11100	11010	1	R.S	1	
	11100	11010			0	

check :-

14 x -5 = -70

70 : 0001000110

-70 : 1110111001

\+ 1

1110111010

2 | 70 — 0

2 | 35 — 1

2 | 17 — 1

2 | 8 — 0

2 | 4 — 0

2 | 2 — 0

2 | 1 — 1

0

5) 5 x -4

M = 5 = 00101

Q = -4 = 11100

M	A	Q	Q-1	Operation	count	
00101	00000	11100	0	Initialisation	5	00
	00000	01110	0	Right shift		00
	00000	00111	0	Right shift	4	10
A	00000			A - M	3	A+(-M)
-M	11011					
	11011	00111		Right shift		
	11101	10011	1			11
	11110	11001	1	Right shift	2	11
	11111	01100	1	Right shift	1	
	11111	01100			0	

Check :-

5 x -4 = -20

20 = 0000010100

-20 = 1111101011 + 1

1111101100

2|20 - 0

2|10 - 0

2|5 - 1

2|2 - 0

2|1 - 1

0

for MSB

6) -13×-9

$M = -13 = 10011$ $\quad -M = 01100 + 1 = 01101$

$Q = -9 = 10111$

M	A	Q	Q-1	operation	count	
10011	00000	10111	0	Initialisation	5	10
	A 00000			A = A - M		A+(-M)
	-M 01101			R.S		
	01101	10111				
	→00110	11011	1	Right shift	4	11
	→00011	01101	1			11
	→00001	10110	1	Right shift	3	01
	A 00001			A = A + M	2	A+M
	M 10011					
	10100	10110		R.S		
	→11010	01011	0	A = A - M	1	10
	A 11010					A+(-M)
	-M 01101			Right shift		
discard	100111	01011				
	→00011	10101	0		0	
	00011	10101				

Check

$-13 \times -9 = 117$

$117 = 0001110101$

2 | 117 — 1
2 | 58 — 0
2 | 29 — 1
2 | 14 — 0
2 | 7 — 1
2 | 3 — 1
2 | 1 — 1
0

7) -5 × -4

M = -5 = 11011   -M = 00100 + 1 = 00101

Q = -4 = 11100

M	A	Q	Q_{-1}	operation	count	
11011	00000	11100	0	Initialisation	5	00
	00000	01110	0	Right shift		00
	00000	00111	0	Right shift	4	10
A	00000			A = A - M	3	A+(-M)
-M	00101					
	00101	00111		Right shift		
	00010	10011	1	Right shift	2	11
	00001	01001	1			11
	00000	10100	1	Right shift	1	
	00000	10100			0	

check: -5 × -4 = 20 = 0000010100

8) -14 × -5

M = -14 = 10010   -M = 01101 + 1 = 01110

Q = -5 = 11011

M	A	Q	Q_{-1}	operation	count	
10010	00000	11011	0	Initialisation	5	10
A	00000			A = A - M		A+(-M)
-M	01110			Right shift		
	01110	11011				
	00111	01101	1	R.S	4	11
	00011	10110	1			01
A	00011			A = A + M	3	A+M
M	10010			R.S		
	10101	10110				
	11010	11011	0	A = A - M	2	10
A	11010					A+(-M)
-M	01110			Right shift		
discard 1	01000	11011				
	00100	01101	1	Right shift	1	11
	00010	00110	1			
	00010	00110			0	

for M... Multiplicand...

Check: $-14 \times -15 = 70 = 0001000110$

$2|70 - 0$
$2|35 - 1$
$2|17 - 1$
$2|8 - 0$
$2|4 - 0$
$2|2 - 0$
$2|1 - 1$
0

9) 5×4

$M = 5 = 00101$ $\quad -M = \overline{11010} + 1 = 11011$

$Q = 4 = 00100$

M	A	Q	Q-1	operation	count	
00101	00000	00100	0	Initialisation	5	00
	00000	00010	0	Right shift		00
	00000	00001	0	Right Shift	4	10
A	00000			A = A - M	3	A+(-M)
-M	11011					
	11011	00001		Right Shift		
	11101	10000	1	A = A + M	2	01
A	11101					
M	00101					
discard	00010	10000		Right Shift		
	00001	01000	0	Right shift	1	00
	00000	10100	0			
	00000	10100			0	

Check

$5 \times 4 = 20 = 0000010100$

$2|20 - 0$
$2|10 - 0$
$2|5 - 1$
$2|2 - 0$
$2|1 - 1$
0

$0000010100 = 20$

Restoring Division Algorithm or Booth's Division Algorithm :-

(Dividend)
(diviser) 10 | 11110 | 110 (Quotient)
101

101
101

00
0
---
0 (remainder)

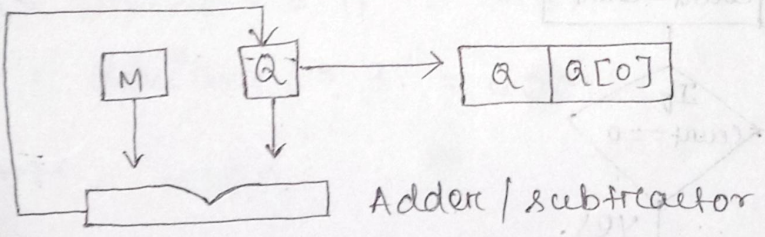
check :- $11110 = 2^4 + 2^3 + 2^2 + 2^1$
$= 16 + 8 + 4 + 2$
$= 30$

$101 = 5$
$110 = 6$

5 | 30 | 6
30

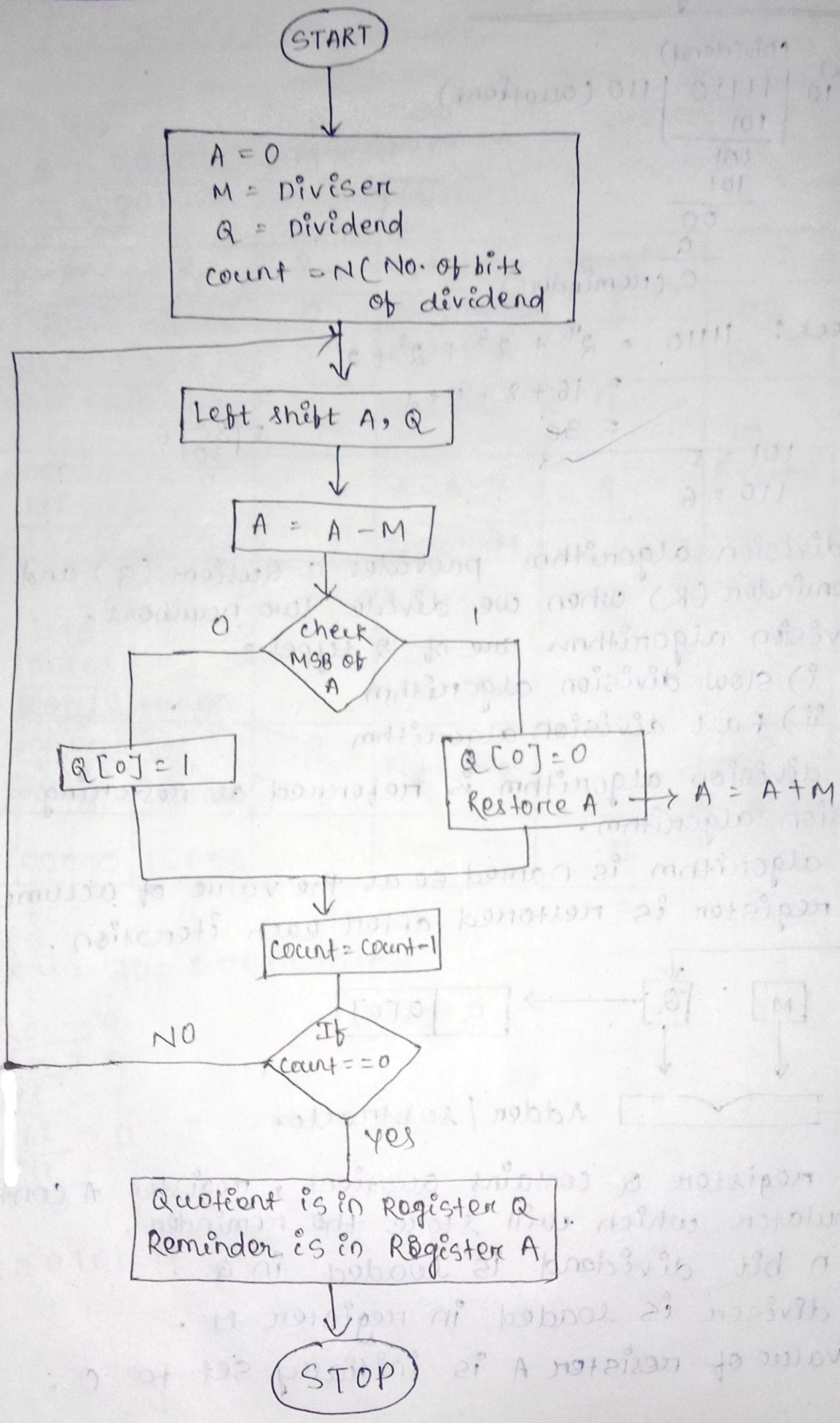
0

→ A division algorithm provides a Qutient (Q) and a remainder (R) when we divide two numbers.
→ Division algorithms are of 2 type :-
  i) slow division algorithm
  ii) Fast division algorithm
→ slow division algorithm is referred as restoring division algorithm.
→ The algorithm is named so as the value of accumulator (A) register is restored after each iteration.



→ Here register Q contains Quotient, register A contains Accumulator which will store the remainder.
→ The n bit dividend is loaded in Q.
→ The diviser is loaded in register M.
→ The value of resister A is initially set to 0.

Algorithm :-


START
A = 0
M = Divisor
Q = Dividend
count = N (No. of bits of dividend
Left shift A, Q
A = A - M
check MSB of A
0
1
Q[0] = 1
Q[0] = 0
Restore A
A = A + M
Count = Count - 1
If Count == 0
NO
yes
Quotient is in Register Q
Reminder is in Register A
STOP

Steps involved in restoring Division:-

step 1: The values will be initialized as A=0, M=diviser, Q= divident, count = no. of bits available in Q.

step 2: Register A and Q will be treated as single unit and shifted left.

step 3: Here value of register M will be subtracted from A and restored in A (A = A - M)

step 4: Now check the most significant Bit (MSB) of Register A, if this bit of A is 0 then the LSB of register Q will be set to 1.
If MSB of A is 1 then LSB of Q will be 0 and restore the value of A (A = A+M) befor subtraction with M.

step 5: After that the value of count will be decreamented to 1.

step 6: If value of count is 0 we will break the loop otherwise go to step-2

step 7: Here we will found the cotient will be available in register Q and reminder will be available in register A.

step 8: End or exit

e.g →

Divide 11 by 3.

Q = Dividend = 11 = 01011

M = diviser = 3 = 00011

$-M = 11100$

11101

$3)\overset{11}{9}(3$ / 11

$\overline{2}$ / 10

M	A	Q	operation	count
00011	00000	01011	Initialisation	
	00000	10111	Left shift	5
A	00000		A = A − M	
−M	11101			
	11101	10110	Q[0] = 0, so restore A	
A	11101		A = A + M	
M	00011			
discard (1)	00000	10110		
	00001	0110	Left shift	
A	00001		A = A − M	
−M	11101			
	11110	01100	Q[0] = 0, so represent A	4
A	11110		A = A + M	
M	00011			
discard (1)	00001	01100		
	00010	1100	Left shift	
A	00010		A = A − M	3
−M	11101			
	11111	11000	Q[0] = 0, so represent A	
A	11111		A = A + M	
M	00011			
discard (1)	00010	11000		
	00101	1000	Left shift	
A	00101		A = A − M	2
−M	11101			
discard (1)	00010	10001	Q[0] = 1	
	00101	0001	Left shift	
A	00101		A = A − M	1
−M	11101			
discard (1)	00010	00011	Q[0] = 1	
	00010	00011		0

remainder coffient

check : 3 | 11 | 3 = 111
9
2 = 10

1) Divide 13 by 6

Q = Dividend = 13 = 01101

M = Divisor = 6 = 00110

$-M = \overline{00110} + 1 = 11001 + 1 = 11010$

M	A	Q	operation	count
00110	00000	01101	Initialisation	
	00000	1101	Left shift	5
A	00000		A = A - M	
-M	11010		Q[0] = 0, so	
	11010	11010	restore A	
A	11010		A = A + M	
M	00110			
discard (1)	00000	11010		
	00001	1010	Left Shift	
A	00001		A = A - M	4
-M	11010		Q[0] = 0, So	
	11011	10100	restore A	
A	11011		A = A + M	
M	00110			
discard (1)	00001	10100		
	00011	0100	Left shift	
A	00011		A = A - M	3
-M	11010		Q[0] = 0, So	
	11101	01000	restore A	
A	11101		A = A + M	
M	00110			
discard (1)	00011	01000		
	00110	1000	Left Shift	
A	00110		A = A - M	2
-M	11010			
discard (1)	00000	10001	Q[0] = 1,	
	00001	0001	Left shift	
A	00001		A = A - M	1
-M	11010		Q[0] = 0, So	
	11011	00010	restore A	
A	11011		A = A + M	
M	00110			
discard (1)	00001	00010		
	00001 (remainder)	00010 (quotient)		0

check:

$6 \overline{)13} (2 = 10$
$\underline{12}$
$1 = 01$

2) Divide 17 by 4 :-

Q = dividend = 17 = 10001

M = Diviser = 4 = 00100

$-M = 11011_{1}$

11100

M	A	Q	operation	Count
00100	00000	10001	Initialisation	5
	00001	0001	Left shift	
A	00001		A = A − M	
−M	11100			
	11101	00010	Q[0] = 0, so restore A	
A	11101		A = A + M	
M	00100			
discard ①	00001	00010		
	00010	0010	Left shift	4
A	00010		A = A − M	
− M	11100			
	11110	00100	Q[0] = 0, so restore A	
A	11110		A = A + M	
+ M	00100			
discard ①	00010	00100		
	00100	0100	Left shift	3
A	00100		A = A − M	
−M	11100			
discard ①	00000	01001	Q[0] = 1	
	00000	1001	Left shift	2
A	00000		A = A − M	
−M	11100			
	11100	10010	Q[0] = 0, so restore A	
A	11100		A = A + M	
M	00100			
discard ①	00000	10010		
	00001	0010	Left shift	1
A	00001		A = A − M	
−M	11100			
	11101	00100	Q[0] = 0, so restore A	
A	11101			
M	00100			
discard ①	00001	00100		
	00001	00100		0

remainder quotient

check =

4) 17 (4 11 100
16
11101

3) Divide 12 by 3

Q = 12 = 01100 ∴ -M = 11100 / 11101

M = 3 = 00011

M	A	Q	operation	count
00011	00000	01100	Initialisation	
	00000	1100	left shift	5
A	00000		A = A - M	
-M	11101			
	11101	11000	Q[0] = 0, so restore A	
A	11101		A = A + M	
M	00011			
discard ①	00000	11000		
	00001	1000	Left shift	4
A	00001		A = A - M	
-M	11101			
	11110	10000	Q[0] = 0, so restore A	
A	11110		A = A + M	
M	00011			
discard ①	00001	10000		
	00011	0000	Left shift	3
A	00011		A = A - M	
-M	11101		Q[0] = 1	
discard ①	00000	00001		
	00000	0001	Left shift	2
A	00000		A = A - M	
-M	11101		Q[0] = 0, so restore A	
	11101	00010		
A	11101		A = A + M	
M	00011			
discard ①	00000	00010		
	00000	0010	Left shift	1
A	00000		A = A - M	
-M	11101		Q[0] = 0, so restore A	
	11101	00100		
A	11101		A + M	
M	00011			
discard ①	00000	00100		
	00000	00100		0
	rem.	Q		

Check

3) 12 (4 // 100

12

0 // 0

Floating Point Representation:-

→ In digital systems floating point numbers containing integer part and the fractional part. Which reserved a certain number of bits in the name Mantissa Part and the exponent part.

→ Hence the floating point representation consists of two parts,

1st part represents signed fixed point Mantissa & 2nd part designates the position of the decimal or binary digits called as exponent.

The format for this is $M \times r^e \longrightarrow$ exponent

Mantissa, radix

→ Consider the floating point numbers A and B expressed as

$$A = Ma \times r^{ea}$$

$$B = Mb \times r^{eb}$$

sign	Mantissa	Exponent
1 bit	8 bit	23 bits

where Ma and Mb are Mantissa, r is the redix & ea - eb are exponents values of A and B respectively.

→ Before performing any arithmatic operation between A and B the operands must be restructured, so that both A and B have same exponent values. This is done by shifting redix point of the operands

e.g→ $2 = 2 \times 10^0$

$300 = 3 \times 10^2$

$4321.768 = 4.321768 \times 10^3$

$6720000000 = 6.72 \times 10^9$

$67.2 = 6.72 \times 10^1$

$0.2 = 2 \times 10^{-1}$

$0.25 = 2.5 \times 10^{-1}$

$0.0025 = 2.5 \times 10^{-3}$

Floating point Addition Algorithm :-

Consider the addition of A and B floating point numbers.

Step-1 :- check whether A or B is 0. if so then the result is the other operand.

Step-2 :- Calculate difference between the exponents.

Suppose $A = Ma \times r^{ea}$

$B = Mb \times r^{eb}$

difference $= ed = eb - eb / eb - ea$

Step-3 :- Select the number A or B who has smaller exponent, then right shift the mantissa of the selected number by number of time = difference ed

Step-4 :- Perform addition of two mantissa i.e. Ma + Mb

Step-5 :- Represent the result as $Sum = (Ma + Mb) \times r^e$

Step-6 :- Normalise the result. This is done by shifting left the Mantissa till the MSB is a non-zero number. For each shift, the exponent of