2/11/23

Date.....

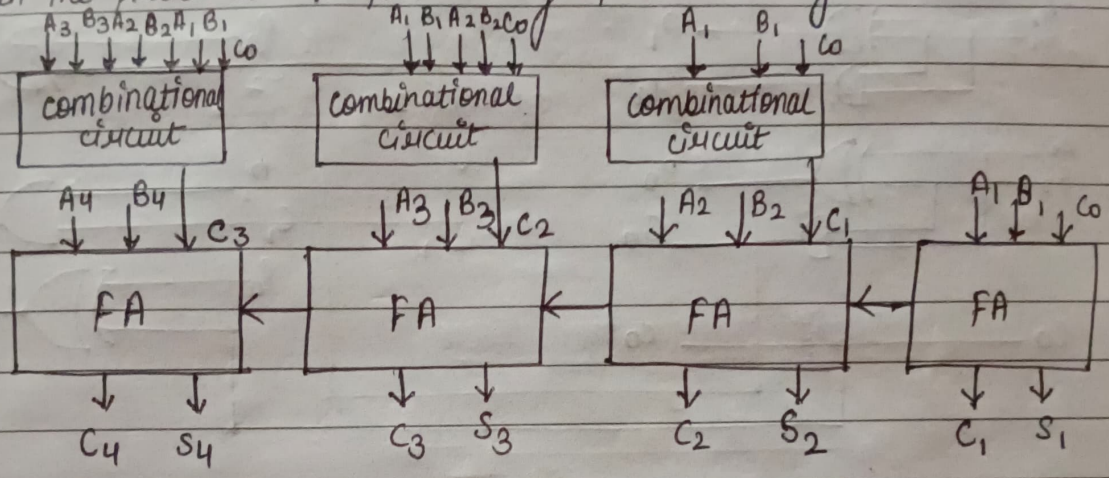
Computer Organization and Architecture

Unit-02

Arithmetic Logic shift unit → Sub part of processor.
① Responsible for execution.
② It contains arithmetic, logic & shift circuit.

Design of 4-bit ahead adder :-

→ It is used to speed up the delay in the processor for adding of two binary numbers.

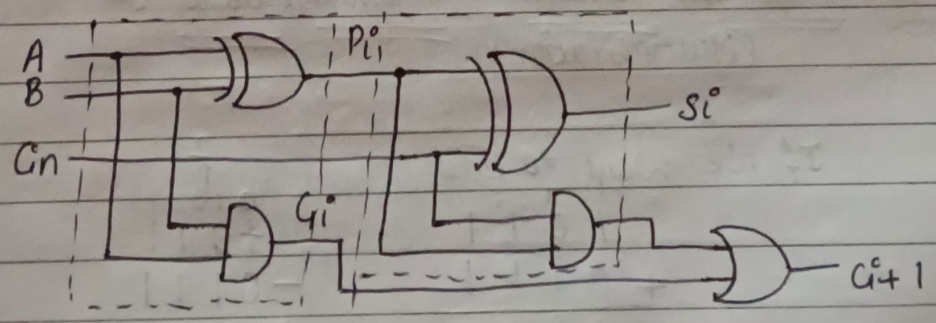


→ Full adder can be implemented using 2-Half adder and a OR-gate.

→ Idea is to look at the lower order bits to see if a higher order carry is to be generated, it uses two functions.

→ Carry generate G_i and carry propogate P_i.

$P_i = A_i \oplus B_i$, $G_i = A_i \cdot B_i$, $S_i = P_i \oplus C_i$, $C_{i+1} = G_i + P_i C_i$



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

$i=0$, $C_1 = G_0 + P_0 C_0$, $i=1 \Rightarrow C_2 = G_1 + P_1 C_1$ ($C_1 = G_0 + P_0 C_0$)

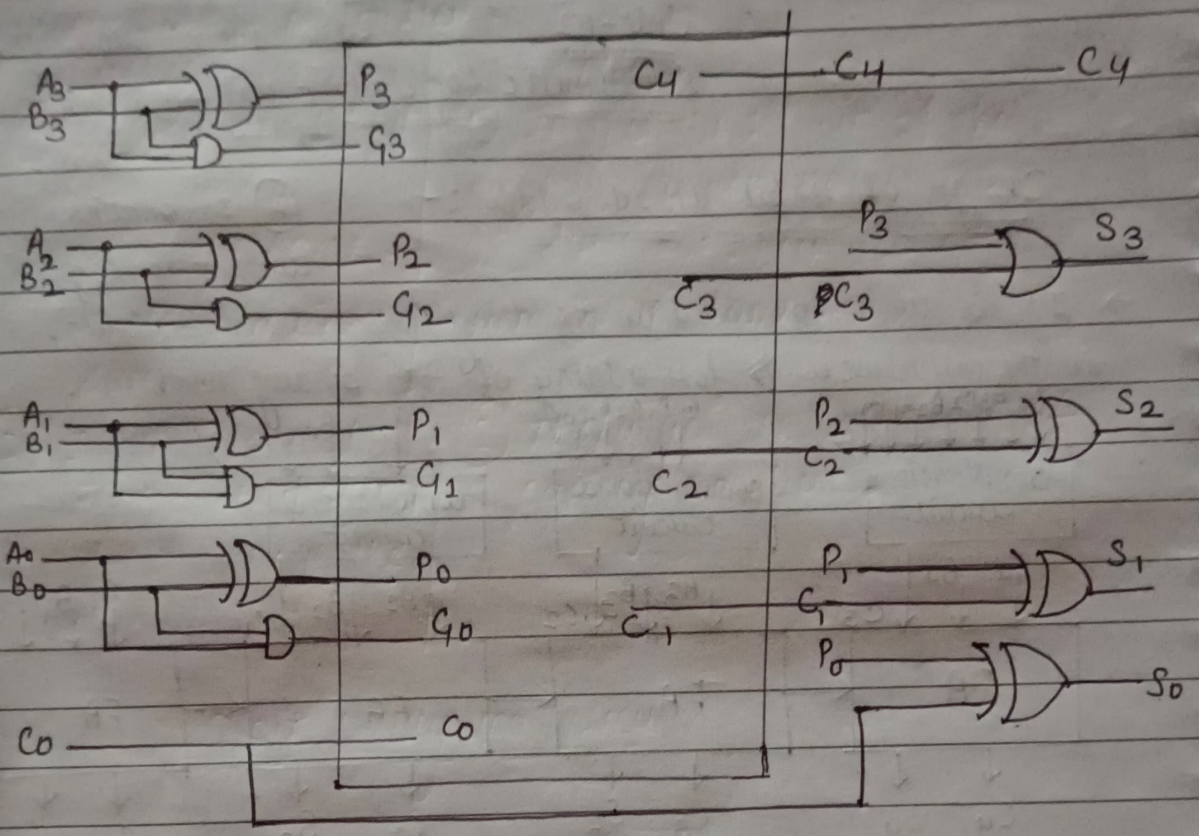
$C_3 = G_2 + P_2 C_2 \Rightarrow C_3 = G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_0)$

Spiral

{ P = Propogation
  G = Generation }

Date.....

→ Carry propogation problem has been resolved through carry look ahead adder.



Block Diagram

Disadvantage : Hardware is required to decrease the propogation delay.

* For signed complement - magnitude data :-
(Multiplication)

If we want to add

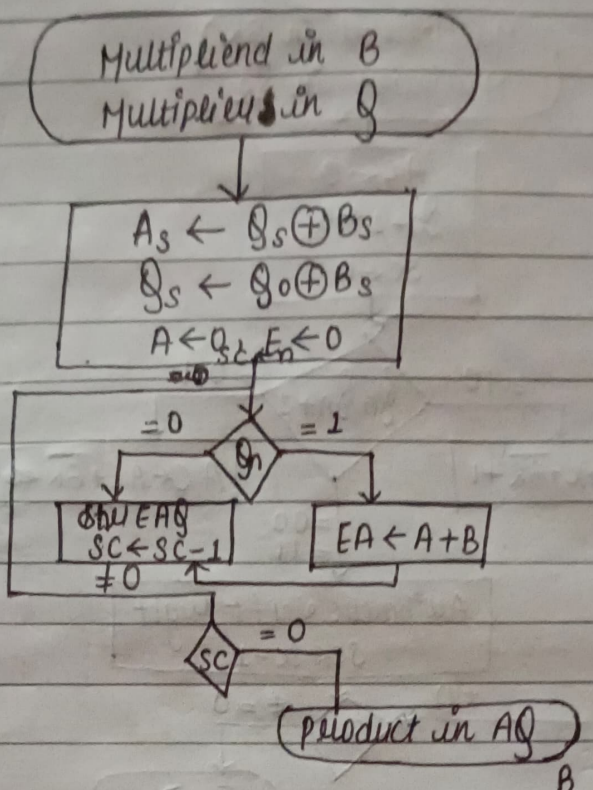
```
  1 0 1 1
+ 0 1 1 1
-------
1 0 0 1 0
↓
Carry
```

```
    2
    ↑
0 1 0
+ 0 1 1
- 0 1 1 1
---------
  0 1 0 0
```

Date.....

Algorithm :-

Multiply operation



11 × 9 = 99 11 - Multiplicand = 1011, 9 - Multiplier = 1001

	E	A	Q	SC
	0	0000	1001	4 = 100
Q_n=1, add B		+1011		
		1011		
shr EAQ	0	0101	1100	3 = 011
Q_n=0, shr EAQ	0	0010	1110	2 = 010
Q_n=0	0	0001	0111	1 = 001
shr EAQ		+1011		
Q_n=1	0	1100		
add B				
shr EAQ	0	0110	0011	0 = 000

Product = AQ = 01100011 → 1's complement

= 64+32+2+1 = 10011101

= 99 dw = 7 6 5 4 3 2 1 0 = 16+8+4+1

Spiral

The hardware for multiplication consist of register A & B. The multiplier is stored in the Q register & its sign in Qs. The sequence counter SC is initially set to a number equal to the no. of bits in the multiplier, the counter is decremented by 1 after forming each partial product. When the content of the counter reaches 0, the product is formed and process stops. Initially, the multiplicand is in register B and multiplier in Q. The sum of A & B forms a partial product which is transferred to EA register. Both partial product & multiplier are shifted to the right. This shift will be denoted by the statement. "shrEAQ" to designate the right shift. The least significant bit (LSB) of A is shifted into the MSB of Q. The bit from E is shifted into MSB of A and 0 is shifted into E. After the shift, 1 bit of partial product is shifted into Q, Pushing the multiplier bits 1 position to the right, in this manner the right most flip flop in register Q designated by Qn will hold the bit of multiplier which must be inspected next.

Spiral

Date.....

Hardware algorithm :-

Initially, the multiplicand in B and multiplier in Q, their corresponding signs are in Bs & Qs respectively. The size are compared, & both A & Q are set to correspond to the sign of the product since a double length product will be stored in register A & Q.

Register A & E are cleared and the sequence counter SC is set to an number equal to the no. of bits of multiplier.

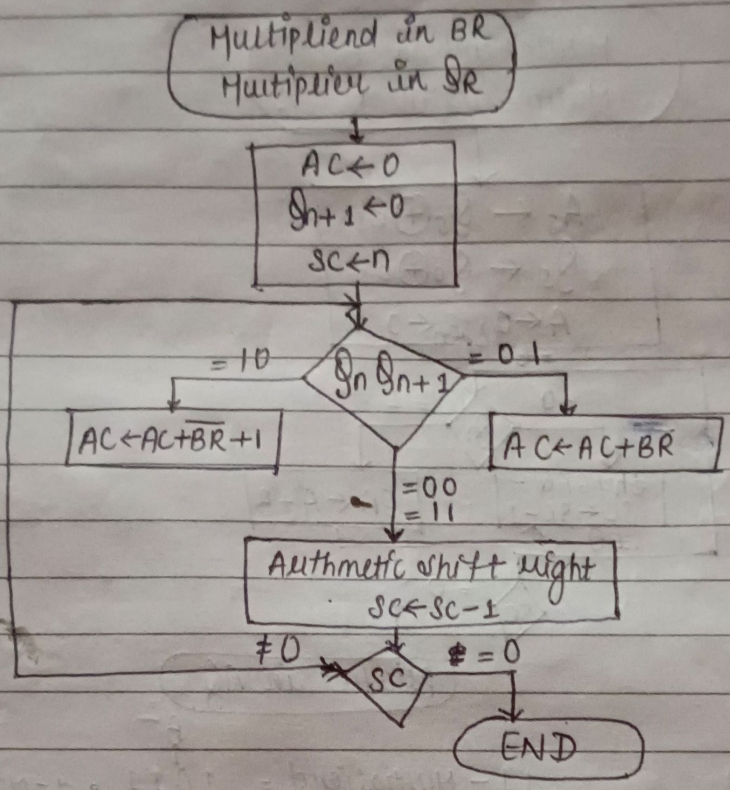
→ After the initialization, the lower product bit of the multiplier in Q_n is tested, if it is 1, the multiplicand in B is added to the present partial product A, if it is 0, nothing is done, register EAQ is then shifted once to the right to form the new partial product. The SC is decremented by 1, and its new value checked. If it is not 0 then process is repeated and new partial product is formed.

→ The process stops when SC = 0, not that the partial product formed in A is shifted into Q, 1 bit at a time & eventually replaces the multiplier, the product is available in both A & Q, A holding the MSB's & Q holding LSB's.

Date.....

Booth multiplication signed 2's complement :-

Multiplication



Ques $12 \times (-6) \Rightarrow 12 = \text{Multiplicand} = BR = 01100$ 2's compleme

$\overline{BR} + 1 = 10100$

$-6 = \text{Multiplier} = QR \Rightarrow 00110 \rightarrow 11010 = QR$

2's complement of Q

$Q_n Q_{n+1}$	AC	QR	Q_{n+1}	SC
	00000	11010	0	5 = 101
0 0	00000	01101	0	4 = 100
ashr				
1 0	00000			
sub BR	+10100			
	10100			
ashr	11010	00110	1	3 = 011
0 1	+01100			
add BR	(1) 00110			
	↓ neglect			
ashr	00011	00011	0	2 = 010
1 0	+10100			
Sub BR	10111			
ashr	11011	10001	1	1 = 001
1 1	11101	11000	1	0 = 000
ashr				

Product = 1110111000 → 2's complement

= 0001001000

9 8 7 6 5 4 3 2 1 0

= 64 + 8 = **72** dus

→ Booth algorithm give a procesoon for multiplying binary integers in signed 2's complement representation, as in all multiplication scheme booth algorithm requires examination of multiplier bits & shifting of the partial product prior to the shifting the multiplicand may be added to the partial product, subtract from the partial product or left unchanged according to the following rules :-

1. The multiplicand is sub. from the partial product upon encountering the first least significant one in strings of one's in the multiplier.
2. The multiplicand is added to the partial product upon encountering the first 0 in the string of zero's in the multiplier.
3. The partial product does not change when the multiplier bits are identical ~~and~~ to the previous multiplier bits.

Algorithm:-

The 1-H/W representation of booth algorithm require the register configuration similar to

Date.....

multiplication of except that the signed bits are not seperated from the rest of the registers. On designated the LSB of the ~~DR~~ Multiplier in the register Q_R. An extra ~~flip flop~~ flop Q_{n+1} is appended to the Q_R to facilitate a double bit inspection, of the ~~expression~~ multiplier Register AC and appended bit Q_{n+1} are initially cleared to 0. & SC is set to a no. n= to the no. of bits in the multiplier. The 2 bits of the multiplier Q_n & Q_{n+1} are inserted.

If the 2 bits are equal to 10, it means that the 1st 1 in a a string of 1's has been encountered this requires a subtracting of multiplicand from the partial product of AC if the 2 bit are equal to 01, it means that the 1st 0 in a string of 0's has been encountered, this required the add of ~~of~~ multiplicand to the partial product of AC, where the 2 bit ~~of~~ are equal, the partial product has not changed and over flow cannot occur because the addition & subtraction of the multiplicand follows each other. The next step is to shift right the partial product & the multiplier (including the bit to n+1) this is a ashr (arithmetic shift right) operation which shifts AC & BR to the right and leave the signed bit in AC unchanged. The SC is implemented & the computational rule is repeated n times.

Spiral

Array multiplier :- Less complex and faster then booth and signed multiplier.

Arithmetic shift right :- 10110

110110 → rejected

Arithment shift left :- 10110

1 ← 01100 ← inserted

discarded

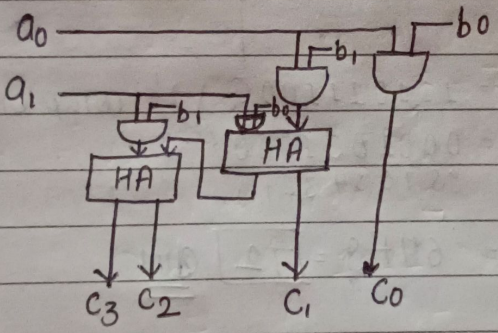
Spiral

Date.....

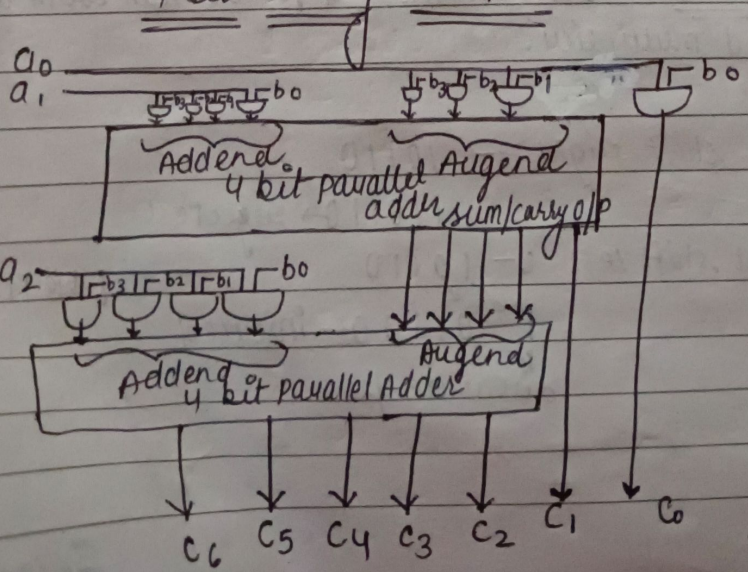
→ More shift operations are used in Booth multiplication & signed multiplication. For better hardware management we use array multiplier.

ashr	shr
(1) In ashr the 1st bit of the number remains the same. i.e. MSB remains same.	(1) In this the 1st bit of the number does not remains the same i.e. MSB does n't remains the same.
(2) In this carry is neglected and doesn't change the value of E.	(2) It changes the value of E if carry appears.

2 bit array multiplier



4 bit array multiplier



Spiral

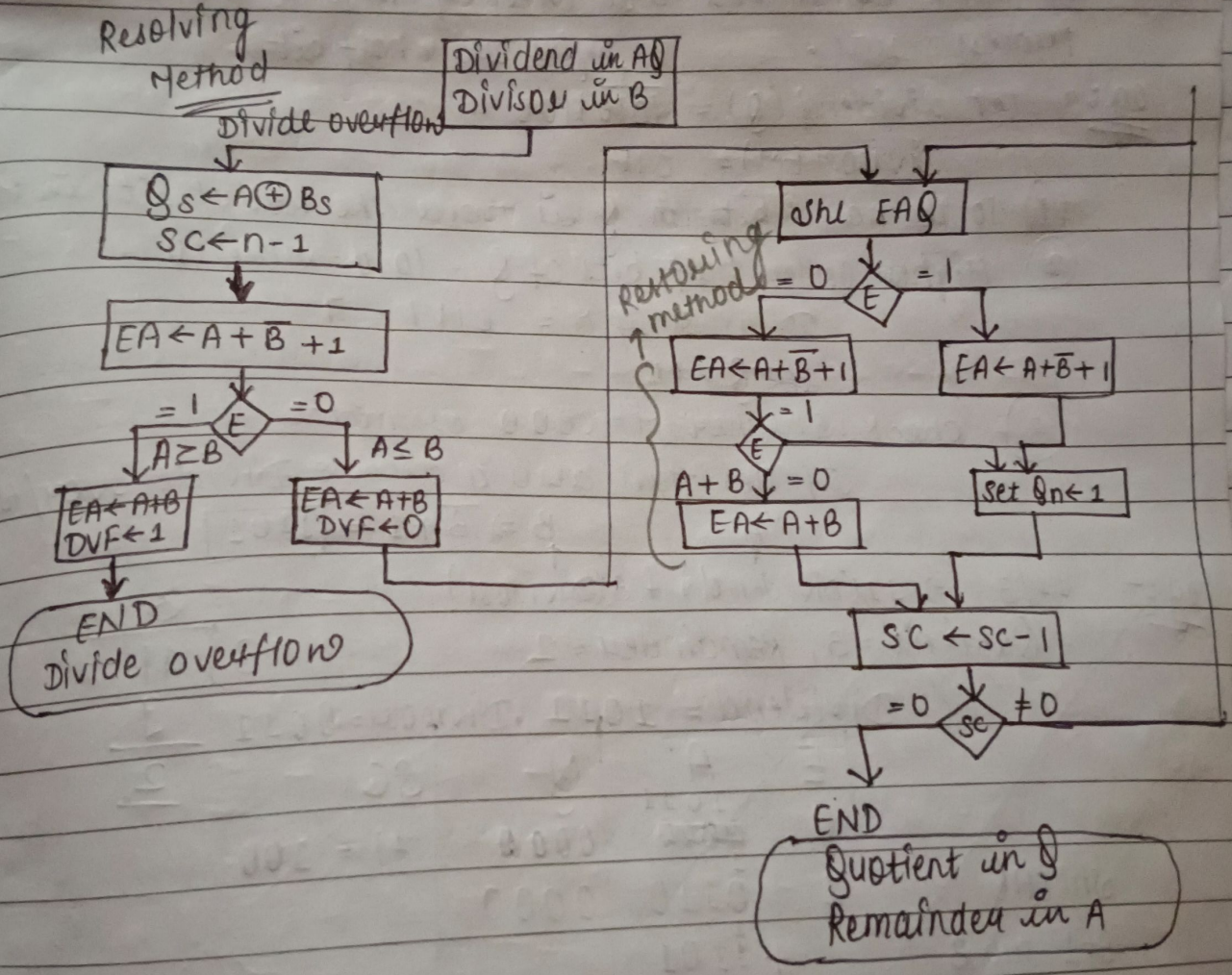
Date.....

2 by 2 bit

a_1 a_0 → Multiplicand
b_1 b_0 → Multiplier

a_1b_0 a_0b_0
$+ a_1b_1$ a_0b_1 ⟶ shift 1 bit
C_2 C_1 C_0

Division algorithm



Spiral

divisor ⟌ dividend

Date.....

Divide overflow condition occurs when :-

(i) The no. of bits in dividend is twice as the no. of divisor.

(ii) The higher order half bits of the dividend has a value more than or equal to the value of the total bit of the divisor.

(iii) The divide overflow also check that whether the divisor is zero or not.

(iv) All these conditions are checked before the division process starts.

eg: Let dividend (Q) = 10101100 (higher order half bit)

Divisor (M) = 0111

(1) So the no. of bits in Q is twice the no. of bits in M.

(2) Higher order half bits of Q = 1010 = value 10

Divisor of M = 0111 = 7

So Q > M

(3) check whether M = 0000 or not

If yes then also indicate Divide overflow

A ↑ B ⇒ $\overline{B}+1$ = 1101 ↑

eg:- 11/3 ⇒ 11 (Dividend) & 3 (Divisor)

Quotient = 3, Remainder = 2 = 0010 (=0011)

Dividend = 1011, Divisor = 0011

```
    3
3 ⟌ 11
    9
    2
```

Restoring Method

	E	A	Q	SC
	0	0000 (1011)	0000	4 = 100
shl EAQ	1	0110	0000	
E=1, sub B		+ 1101		
	1	0011		
E=1, set Q0 ← 1	1	0011	0001	3 = 011
shl EAQ	0	0110	0010	
E=0, sub B		+ 1101		
	1	0011		
E=1, set Q0 ← 1	1	0011	0011	2 = 010

Spiral

Date.....

Qq: 11/3 = Dividend = 11 & Divisor = 3

Quotient = 3, Remainder = 2

Dividend = A = 1011, Divisor = B = 0011

$\overline{B} + 1 = 1101$

	E	A	Q	SC
	0	0000	1011	4 = 100
ShlEAQ	0	0001	0110	
SubB		+ 1101		
		1110		
E=0	0			
E=0, addB		+ 0011		
	1	0001	0110	3 = 011
ShlEAQ	0	0010	1100	
E=0, SubB		+ 1101		
	0	1111		
		+ 0011		
E=0, addB	1	0010	1100	2 = 010
ShlEAQ	0	0101	1000	
E=0, ~~add~~ sub B		+ 1101		
	1	0010		
E=1, set Qn←1	1	0010	1001	1 = 001
ShlEAQ	0	0101	0010	
E=0, subB		+ 1101		
	1	0010		
E=1, set Qn←1	1	0010	0011	0 = 000

Quotient in Q = 0011 = 3

Remainder in A = 0010 = 2

Date.....

The divident is in AQ and divisor is in B. The sign of the divident is transferred into Qs to be the part of the quotient the constant is set into the sequence counter sc to specify the number of bits in the quotient. As in multiplication we assume that operands are transferred to registers from memory word until that has words of n bits. and not since an operand must be stored with its sign. 1 bit of the word will be occupied by the sign and magnitude will be consists of n-1 bits. If divide overflow condition is tested by substracting the divisor in B from half of the bits of the dividend stored in A. If A ≥ B, the divide overflow flop (DVF) is set and the operation is terminated prematurally. If A < B no overflow divide occurs so the value of the dividend is restored by adding B to A. The division of the magnitude starts by shifting the divdend in AQ to the left with the high order bit shifted into E. If the bit shifted into E is 1 we know that EA ≥ B bcoz EA consists of a 1 followed by n-1 bits while B consists of only n-1 bits. In this case, B must be subtracted from EA and 1 inserted into Qn for the Quotient bit. since register A is missing the high order bit of the divident which is in E() its value is $EA - 2^{n-1}$ adding to this value the 2's complement of B result in $(EA - 2^{n-1}) + (2^{n-1} - B) = EA - B$. The carry from this addition is not transferred to E. If we want to E to remain a 1 the shift left operation inserts a 0 into E the divisor is subtracted by adding its 2's complement value and carry is transferred into E. If E=1 it signifies that

Date

A<B and original number is restored by adding B to A in the latter case we leave 0 in Q_n. This process is repeated again with register A holding the partial remainder after n-1 time. The Quotient magnitude is form in register Q & remainder is found in register the Quotient sign is in Q_n & the sign of num is in A_n, is the same as the original sign of the dividend.

Types of division :-

(1) Restoring Method :- In this method the partial remainder is restored by adding the divisor to the -ve differences.

(2) comparison Method :- In this method A & B are compared prior to the subtraction operation. If A≥B, B is subtracted from A so if A<B nothing is done. The partial remainder is shifted left and the numbers are compared again. The comparison is determined prior to the subtraction by inspecting the END carry out of the parallel adder prior to its transfer to register E.

(3) Non restoring Method :- In this method B is not added if the difference is -ve but instead the -ve difference is shifted left and then B is added. We note that the operation perform a A-B+B i.e. B subtracted and then added to restore A. The next time around the loop this number is shifted left (or multiplied by 2). The next time around the loop we subtract the value again this gives 2(A-B+B)-B = 2A-B. This result is obtained in the non-restoring method by leaving A-B as is. The next time around the loop the number is shifted left & B added to give A-B+B i.e. equal to A this

Spiral

Date.....

is same as before. Thus in the non-restoring method B is subtracted if the previous value of Qn was 1 but B is added in previous value of Qn was 0 and no restoring of the partial remainder is required. This process save the steps to adding the divisor if A<B but it requires special control logic to remember the previous result. The first time the divident is shifted, B must be subtracted. Also if the last bit of the quotient is 0, the partial remainder must be restored to obtain the correct final remainder.

⇒ Divide overflow condition occurs if the high order half bits of the dividend consists a number greater than or equal to the divisor. Another problem associated with division is the fact that a divide by 0 must be avoided. This occurs because any divident will be greater than or equal to an divisor which is equal to an 0.

Date.....

* Floating point Arithmetic operations :-

(1) Addition / subtraction of floating point numbers :-

Algo :-

(1) Check for zero's, if either no. is zero, the result will be other number with appropriate sign.

Let, $X = X_A * 2^{E_A}$ (X_A: mantissa, E_A: exponent)

$Y = X_B * 2^{E_B}$ $\Big]$ $Z = ?$

During addition :- If $Y = 0$, $Z = X = X_A * 2^{E_A}$

If $X = 0$, $Z = Y = X_B * 2^{E_B}$

During subtraction :- If $Y = 0$, $Z = X = X_A * 2^{E_A}$

If $X = 0$, $Z = -Y = -X_B * 2^{E_B}$

(2) Align the mantissa :- If exponent are equal, perform the arithmetic operations ; else shift the mantissa to the right with smaller exponent until its exponent becomes equals to the larger exponent.

Let $X = 1.10 * 2^{+3}$ (larger exponent)

$Y = 1.01 * 2^{+1}$ (smaller exponent)

$Z = X + Y$

Shift the mantissa of Y to the right, until its exponent is equal to the exponent of X.

$Y = 0.101 * 2^{+2}$

$Y = 0.0101 * 2^{+3}$

$X = 1.10 * 2^{+3}$, $Y = 0.0101 * 2^{+3}$

(3) Depending upon the operation & sign of the two mantissa perform addition / subtraction.

* During addition, if two significant are equal with opposite sign, the result will be zero.

Date.....

* There is also a possibility of significant (mantissa) overflow by 1 digit, then the mantissa of the result is shifted right and exponent is incremented.

Significant overflow :- Occurs, when the addition of two significant (mantissa) of same sign may result in a carry at MSB of mantissa as we increment exponent, there is also a possibility of exponent overflow. The result will show "error".

① If $X = 1.01 * 2+3$

$Y = -1.01 * 2+3$

then $Z = X+Y$

$0 * 2+3 = 0$

② If $X = 1.01 * 2+3$

$Y = 1.10 * 2+3$

$\overline{\textcircled{1}0.11 * 2+3}$

↓ carry in MSB

correction = $\boxed{1.011 * 2+4}$

④ Normalize the result :- If result is not normalized, then we shift mantissa left and decrement the exponent until value" left of binary point is 1". As we are decreasing the exponent, so exponent can also be underflow then again error is reported.

If $X = 1.01 * 2+3$

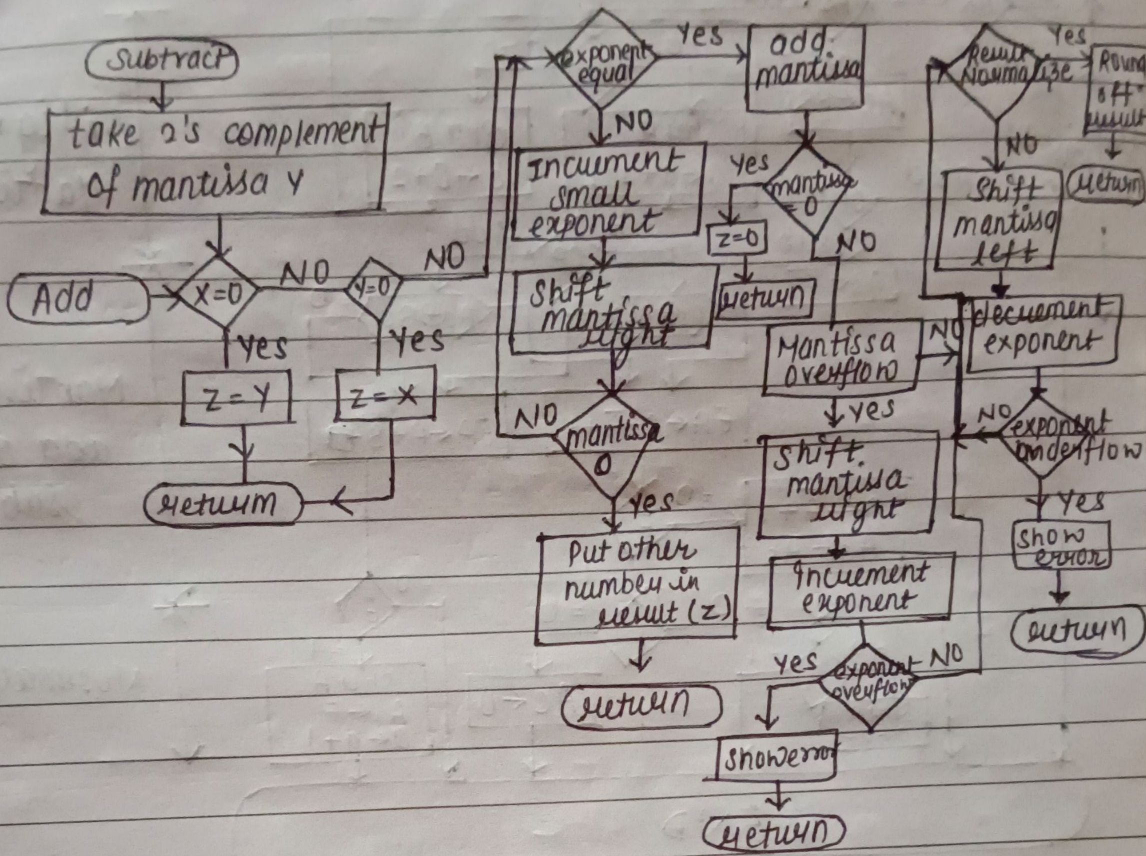
$-1.00 * 2+3$

$\overline{0.01 * 2+3}$ (Not Normalized)

$1.0 * 2+1$ (Normalized)

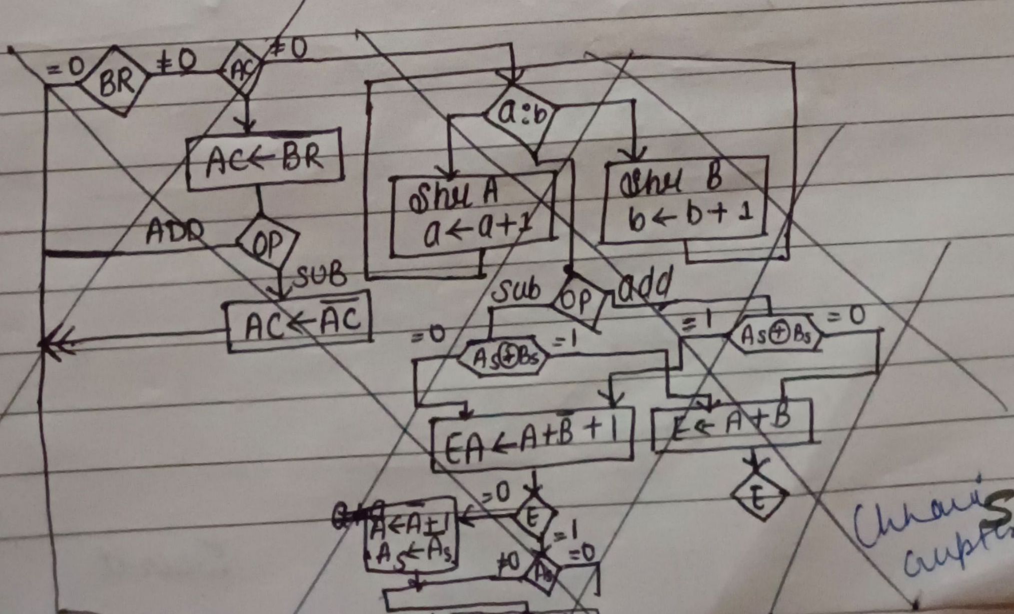
Date....................

Flowchart



① Significant overflow :- When two similar sign significant (mantissa) are added, the sum may result in a carry out of the MSB. This is called significant overflow.

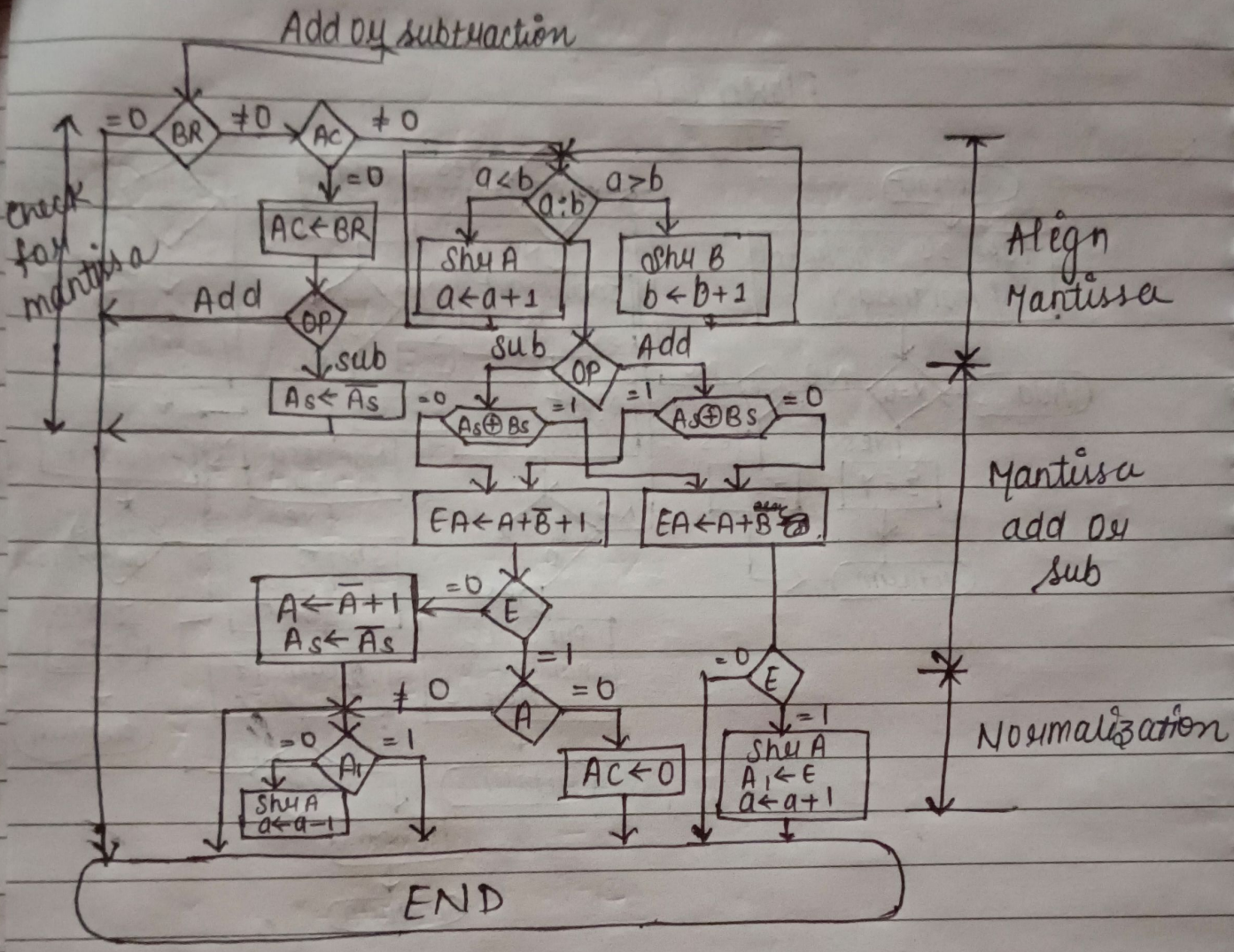
② significant underflow :- If the resulting value has 0 in the MSB.



Chhavi Gupta

Spiral

Date.....



Add or sub for floating point numbers

Date.....

Floating point multiplication

1. Check for zeros.
2. Add the exponents.
3. Multiply the mantissa.
4. Normalize the product.

$X = X_A * 2^{E_A+e}$ → biased exponent

$Y = X_B * 2^{E_B+e}$

$X * Y = (X_A * 2^{E_A+e})(X_B * 2^{E_B+e})$

$Z = (X_A * X_B)(2^{E_A+E_B+2e})$

$Z = (X_A * X_B)(2^{E_A+E_B+2e-e}$ (as exponent is added twice)

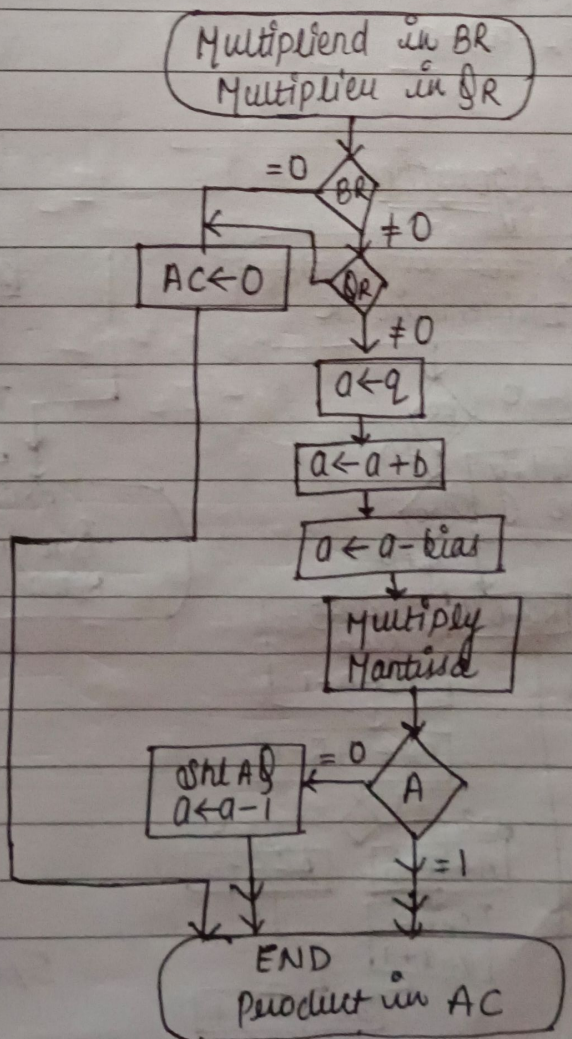
$Z = (X_A * X_B)(2^{E_A+E_B+e})$

$Z = X * Y$

$Z = 0$ ($\neq 0$) ($= 0$)

$Z = 0$

Algorithm



Spiral

Date.....

Floating point four Division

1. Check for zeros.
2. Initialize registers & evaluate sign.
3. Align the dividend.
4. Subtract the exponent
5. Divide the mantissa.

$$X = X_A * 2^{E_A + e}$$

$$Y = X_B * 2^{E_B + e}$$

$$\frac{X}{Y} = \frac{X_A}{X_B} * 2^{(E_A + e - E_B - e)}$$

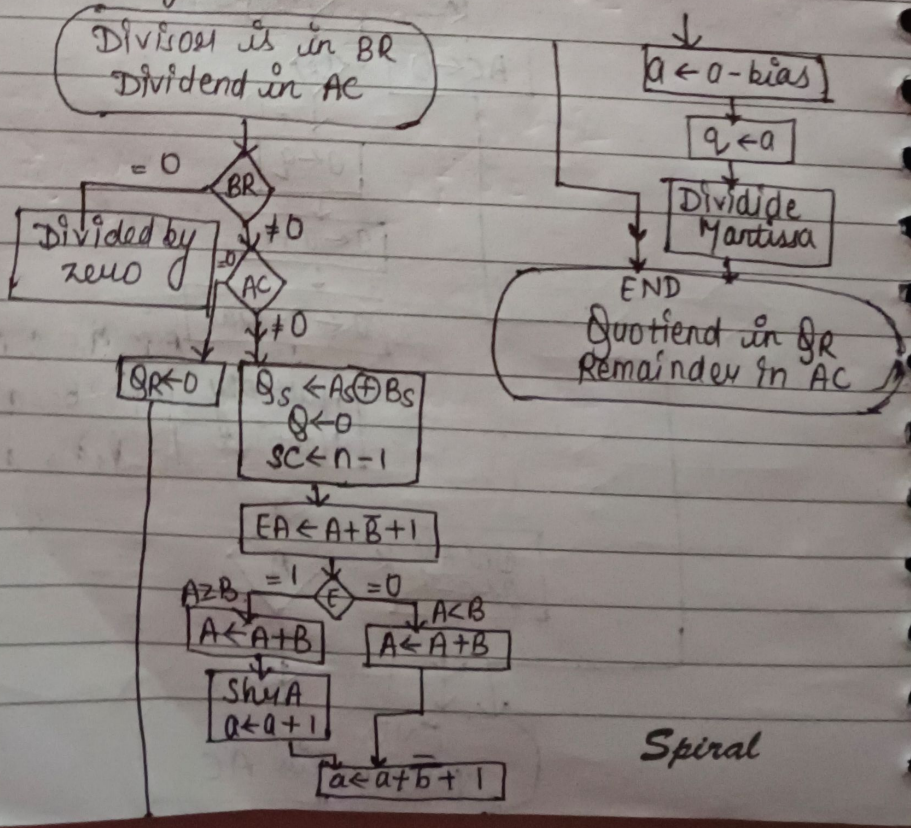
$$\frac{X}{Y} = \frac{X_A}{X_B} * 2^{(E_A - E_B)}$$

$$\Downarrow$$

biasing exponent

$$\frac{X}{Y} = \frac{X_A}{X_B} * 2^{(E_A - E_B) + e}$$

Algorithm



Spiral

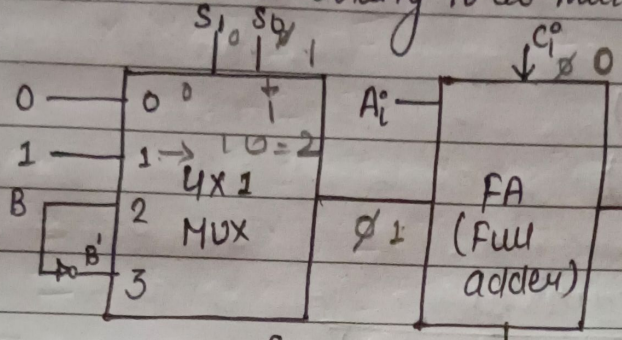
Date.....

Arithmetic and Logic unit Design

ALU = Arithmetic unit + Logic unit + shift unit

Arithmetic Circuit

→ Multiplexer has the ability to do multiple operations.

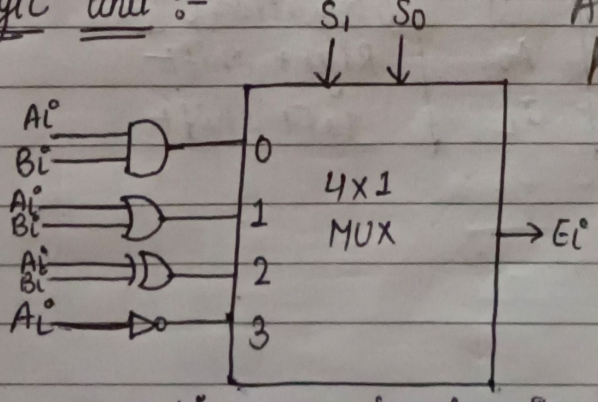


$A_3 A_2 A_1 A_0$
$- 0\ 0\ 0\ 1$
2's
Ai , Ai+1, Ai-1 + (1 1 1 1)
Every bit is subtracted to 1

→ Selection Lines decides which operⁿ will perform.

	S_i	S_0	C_i	Operations
0	0	0	0	Ai (Transfer)
1	0	0	1	Ai+1 (Increment)
2	0	1	0	Ai-1 (decrement)
3	0	1	1	Ai (Transfer)
4	1	0	0	Ai+Bi (Add)
5	1	0	1	Ai+Bi+1 (Add with carry)
6	1	1	0	Ai+ $\overline{Bi}$ (subtract with Borrow)
7	1	1	1	Ai + $\overline{Bi}$ +1 (subtract)

Logic unit :-



At a time one operⁿ has been performed i.e. we use multiplexer.

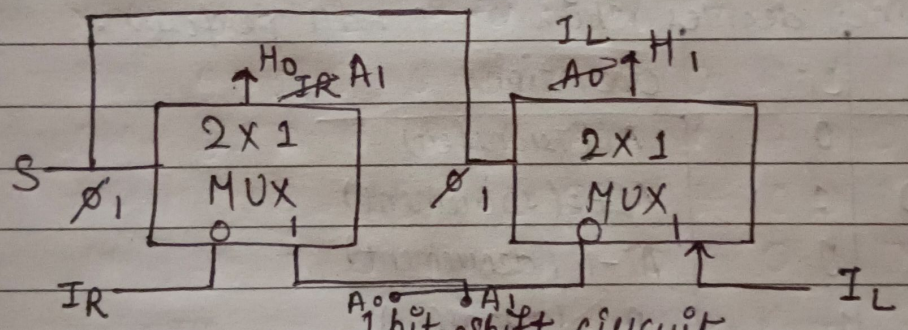
Date.....

S_1	S_0	Operation (E_i)
0	0	AND ($A_i \cdot B_i$)
0	1	($A_i + B_i$) OR
1	0	($A_i \oplus B_i$) XOR
1	1	A_i' (NOT)

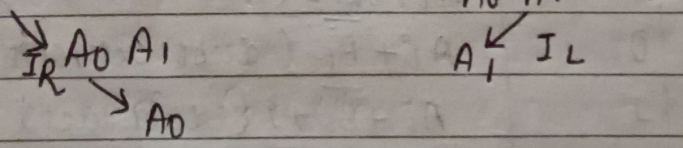
Shift unit :-

1. Right shift (shr)
2. Left shift (shl)

S	Operations
0	shr (right shift)
1	shl (left shift)



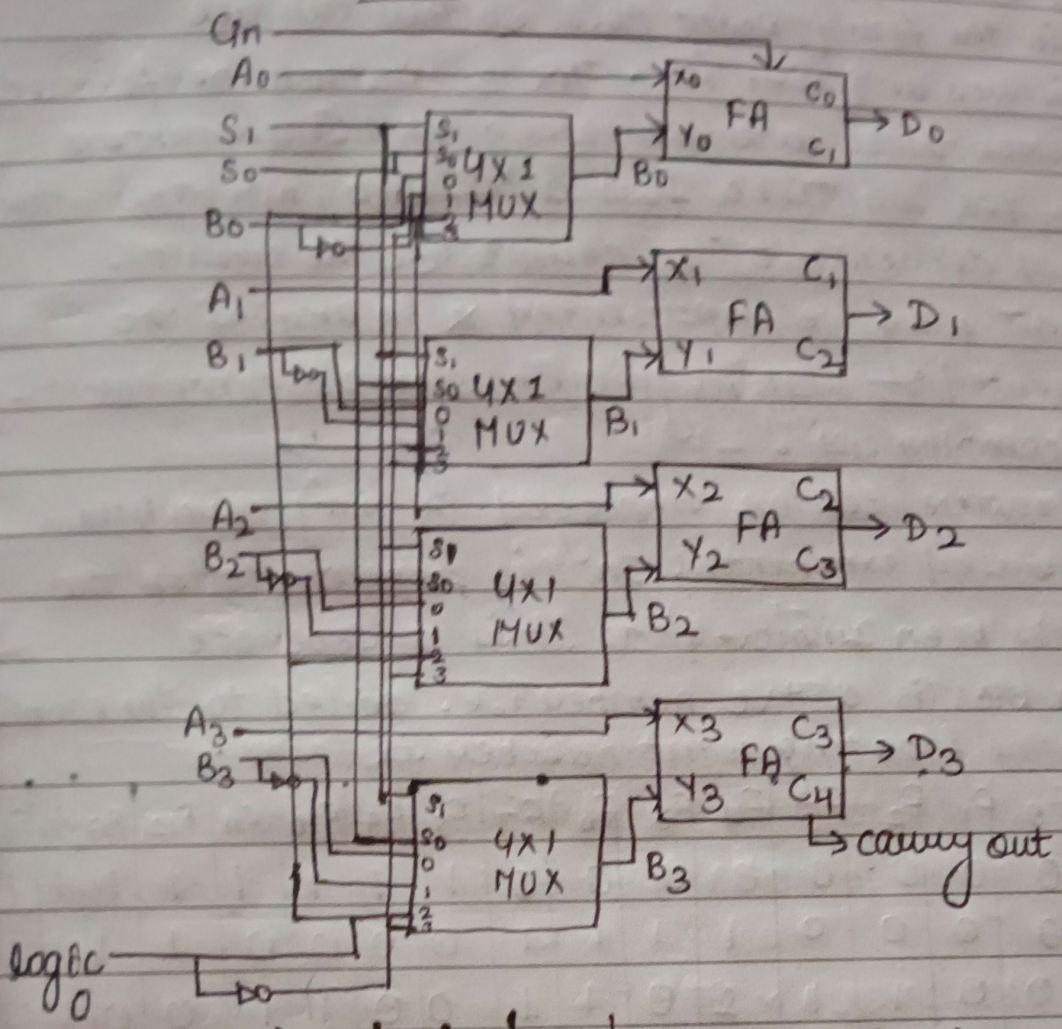
1 bit shift circuit



S	H_0	H_1	Operation
0	I_R	A_0	Right shift
1	A_1	I_L	Left shift

Date.....

ALU



S_3	S_2	S_1	S_0	C_i	G_i (operation)
0	0	0	0	0	A_i (Transfer)
0	0	0	0	1	A_i+1 (Increment)
0	0	0	1	0	A_i-1 (decrement)
0	0	0	1	1	A_i (Transfer)
0	0	1	0	0	A_i+B_i (Add)
0	0	1	0	1	A_i+B_i+1 (Add with carry)
0	0	1	1	0	$A_i+\overline{B_i}$ (Sub with Borrow)
0	0	1	1	1	$A_i+\overline{B_i}+1$ (subtract)
0	1	0	0	X	AND (logic)
0	1	0	1	X	OR (logic)
0	1	1	0	X	XOR (logic)
0	1	1	1	X	NOT (logic)
1	0	X	X	X	Shift right
1	1	X	X	X	Shift left

Spiral

Date.....

→ The operation performed with the mantissa's are same as in the fixed point no.s so they can use same registers and circuits. The operation performed with the exponents are compared and incremented to align the mantissa for add, sub, multiplication & divide and decrement (to normalise the result). The exponent may be represented in any of the three represented ~~is~~ sign magnitude, sign 2's complement & sign 1's complement.

→ Fourth representation employed in many computers known as biased exponents. In this representation the sign bit has been removed as it is a seperate entity. The bias is a +ve number i.e. added to the each exponent as the floating point number is form.

X	Y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

eg: consider an exponent that ranges from -50 to +49. Internally it is represented by +ve number (without a sign).

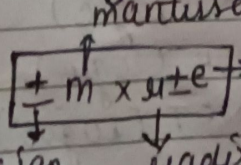
IEEE standard for floating point numbers

215 = fixed point number ⇒ $21.5 \times 10^{+1} \Rightarrow 2.15 \times 10^{+2}$

Floating point numbers : contains two parts :

(i) Mantissa : contains signed fixed point number

(ii) Exponent : specify the position of decimal binary point.



Date.....

examples :-

$+215.37 = +0.21537 \times 10^{+3}$ → exponent

sign Mantissa Base

2	25
2	12 1
2	6 0
2	3 1
	1

$1000.110 = +0.1000110 \times 2^{+4}$ → exponent

sign Mantissa Base (Radix)

0.1101

$-101011 = -0.101011 \times 10^{+6}$ → exponent

= sign Mantissa

These numbers are called floating point numbers because of the decimal (binary) point is fluctuating (floating).

Normalization :-

A floating point number is said to be normalized, when we force the integer part of its ~~magnitude~~ mantissa to be 1, and allow its fraction part to anything.

eg :- $13.25 = 1101.01$

$= 1.10101 \times 2^{+3}$ (Normalized)

integer part → fraction part

$= 1.ffff \times 2^{exponent}$

0 1

0.25 × 2 = 0.50

0.50 × 2 = 1.0

0.25 × 2 = 0.5

0.5 × 2 = 1.0

IEEE standard :-

① single precision floating point standard :- / 32 bit floating point representation

1 bit	8 bits	
sign bit	biased exponent bits e+127	Mantissa (normalized)

Mantissa 32 bits

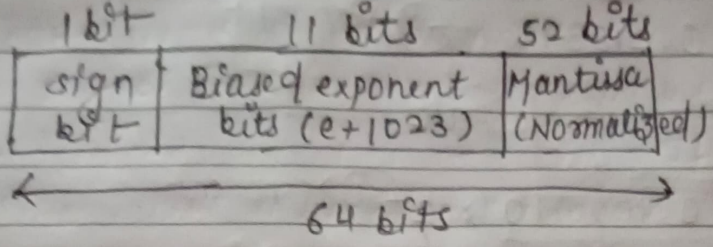
positive → 0

negative → 1

Spiral

Date.....

(2) Double precision floating point standard :-



Biased exponent for 32 bits

$-(2^{8-1})$ to $+(2^{8-1}-1)$

-127 to $+127$

for 64 bits

$-(2^{11-1}-1)$ to $(2^{11-1}-1)$

Ques Represent 85.125 in IEEE 754 single precision floating point representation.

Ans

85.125 (decimal)

1010101.001 (binary)

So,

$1.010101001 \times 2^{+6}$

exponent = e = 6, sign bit = 0

Biased exponent = 6 + 127 = 133 = 10000101

Mantissa (Normalized) = 01010101001

1 bit	8 bit	23 bits
0	10000101	01010100100000000000000

Spiral