UNIT-III

Computer Arithmetic

→ There are 3 ways of representing negative fixed pt binary no's. 1) Signed magnitude
2) " 1's complement
3) " 2's "

→ Most computers use the signed 2's comp. representation when performing arithmetic operations with integers.

→ For floating pt operation, most computers use the signed-mag. representation for the mantissa.

Addition & subtraction with signed-mag. data:-

Addition & subtraction] algorithm:-

→ When the signs of A and B are identical (Different) add the two magnitudes and attach the sign of A to the result.

→ When the signs of A and B are different (identical), Compare the magnitudes and subtract the smaller number from the larger.

→ If the two choose the sign of the result to be same as A if A>B or the complement of the sign of A if A<B.

If the two mag. are Equal, subtract B from A and make the sign of the result positive.

→ Two compare algorithms are similar except for the sign comparis

Hardware implementation:-

→ To implement the two arithmetic operations with hardware, the two nos are stored in regs. Let A and B be two regs that hold the mag. of the no's.

→ A_s & B_s be two FF's that hold corresponding signs. The result of the operation is transferred to a 3rd reg, A, & A_s. A & A_s reg called as Accumulator.

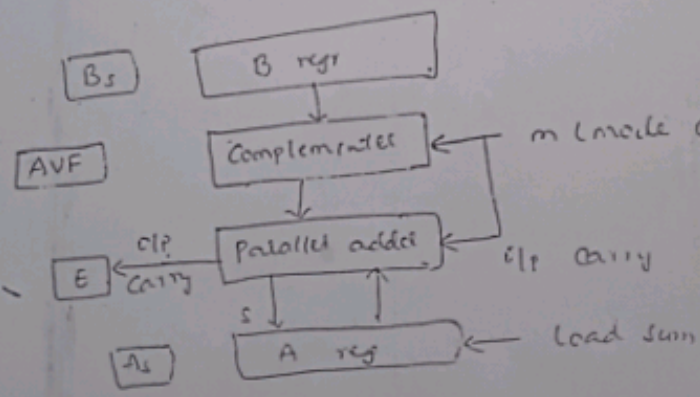
→ First a parallel adder is needed to perform micro operation A+B. Second comparator ckt is needed to establish if A>B, A=B, A<B.

→ Third, a two parallel subtractor ckts are needed to perform the operations A−B & B−A. The sign relationship can be determined from an X-OR gate with A_s & B_s as i/p's.

→ This procedure requires a magnitude comparator, an adder and two subtractors.

→ Subtraction can be accomplished by means of complement and add.

→ Second, the result of comparation can be determined from the end carry after the subtraction.



Hardware for Signed-magnitude addition & Subtraction.

→ It consists of reg's A & B and sign FF's A_s & B_s. Subtraction is done by adding A to the 2's comp. of B. The o/p Carry is transferred to FF 'E'. The add-overflow FF (VF) holds

the overflow bit when A and B are added.

The A-reg → The addition of A & B is done through the parallel adder. The sum (S) o/p of the adder is applied to the i/p of the A reg.

→ The complementer provides an o/p of B or the complement of 'B' depending on the state control of the mode control 'm'.

→ The complementer consists of x-OR gates and the parallel adder consists of full adder.

→ The m signal is also applied to the i/p carry of the adder. When m=0, the o/p of 'B' is transferred to the adder, the i/p carry is 0, and the o/p of the adder is equal to the sum A+B.

→ When m=1, the 1's comp. of 'B' is applied to the adder, the i/p carry 1, and o/p S = A + $\bar{B}$ + 1. This is equal to A plus the 2's comp. of B. which is equivalent to the subtraction A-B.

Hardware Algorithm

→ The two signs A_s & B_s are compared by an x-OR gate. If the o/p of the gate is '0', the signs are identical.

- If it is '1', the signs are different.

→ For an add operation, identical signs indicate that the magnitudes are added.

for sub. operation, different signs indicate that the mag. are added. The mag. are added with a micro operation EA ← A+B. EA is a reg that combines E and A.

→ The carry in 'E' after the addition constitute an overflow if it is equal to 1. It is transfered into add-overflow FF (AVF)

→ The two neg.. are subtracted, by adding A to the 2's comp.. of B. NO OF can occur if the no's are subtracted. so AVF = 0.

→ A '1' in E indicates that A ≥ B. → (continue)
A '0' " " " A < B. For this case it is necessary to take 2's comp.. of the value in A. This operation can be done with one micro operation

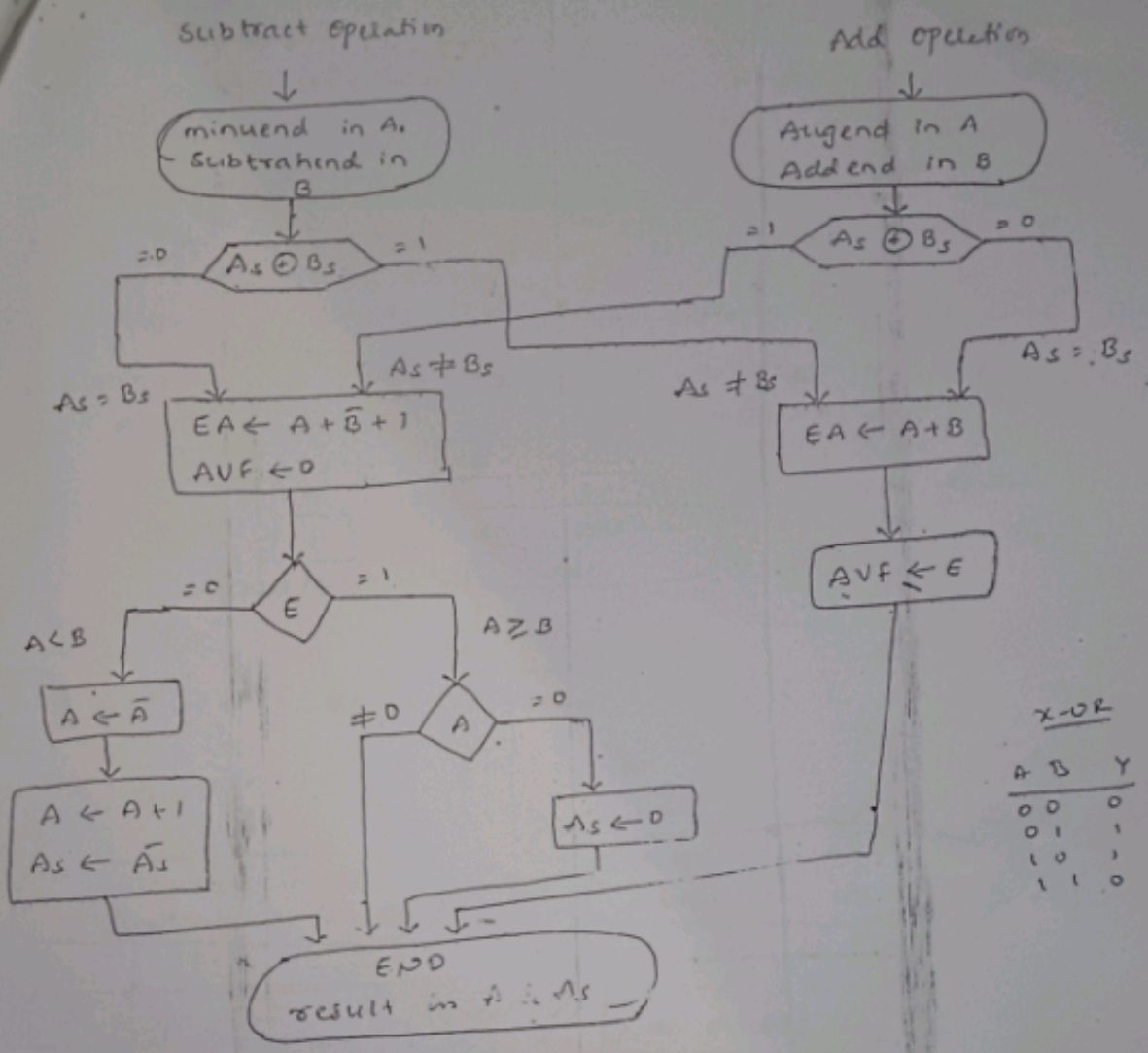
$A \leftarrow \bar{A} + 1$

→ and the no' in A is the correct result. If 'A' is 0, the sign As must be made positive.

→ A reg has micro operations complement & increment. so the 2's comp.. is obtained from these two micro operations.

→ when A < B, the sign of the result is complement of the original sign of A.

→ The final result is found in reg 'A' & its sign in As.



flow chart for Add & sub.. operations.

Addition & sub - of Signed - mag.. no's

Operation	Add mag..	Subtract mag.. when A>B	A<B	A = B
(+A) + (+B)	+ (A+B)			
(+A) + (-B)		+ (A-B)	-(B-A)	+ (A-B)
(-A) + (+B)		-(A-B)	+ (B-A)	+ (A-B)
(-A) + (-B)	-(A+B)			
(+A) - (+B)		+ (A-B)	-(B-A)	+ (A-B)
(+A) - (-B)	+ (A+B)			
(-A) - (+B)	-(A+B)			

$(-A)-(-B)$ $-$ $-(A-B)$ $+(B-A)$ $+(A-B)$

Addition & Subtraction with Signed 2's Comp.. data:

→ The left most bit of a binary no' represents the sign bit. '0' for +ve & '1' for -ve.

→ The addition of two no's in signed 2's comp.. form consists of adding the no's with sign bits.

→ The subtraction consists of first taking the 2's comp.. of the subtrahend and then adding +1 to the minuend.

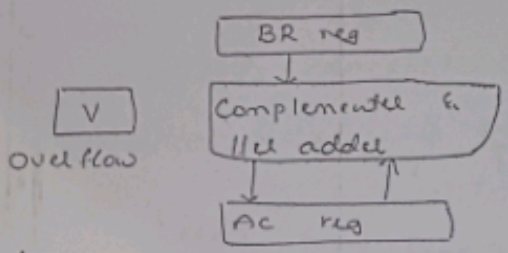
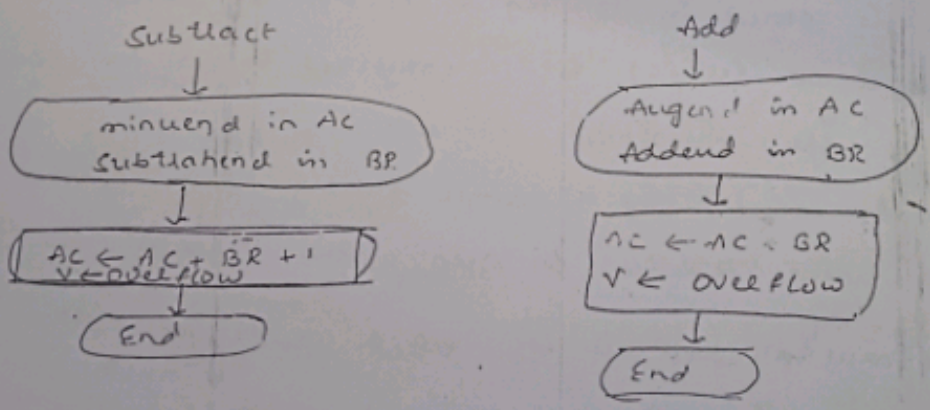


Fig: H/w for signed 2's comp.. addition & subtraction



→ Comparing above algorithm with signed mag.. representation in signed 2's comp.. representation it is much simpler to add & subtract no's if the no's are -ve.

→ For this reason most computers use signed 2's comp.. method rather than signed mag.. representation.

multiplication Algorithm:-

→ multiplication of two fixed pt binary no's in signed mag. representation is done with paper & pencil by a process of successive shift & add operations

```
23        10111   multiplicand
x 19    x 10011   multiplier
        ---------
          10111
         10111
        00000
       00000
      10111
      ---------
437   1.101101    product
```

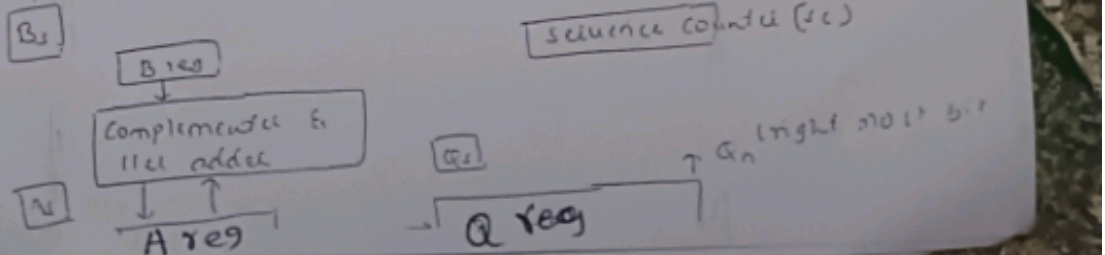
H/w implementation for Signed-mag. Data:

→ The multiplier is stored in 'Q' reg & its sign is stored in Q_s reg. The sequence counter (SC) is initially set to a no' equal to no. of bits in the multiplier.

→ The counter is decremented by 1 after forming each partial product when the counter value reaches to zero, the process stops.

→ Initially, the multiplicand is in reg 'B' and multiplier in 'Q'. The sum of A & B forms a partial product which is transferred to EA Reg.

→ Both partial product and multiplier are shifted to right. The LSB of A is shifted into MSB position of Q, the bit from 'E' is shifted into the most significant position of 'A', and '0' is shifted into E.



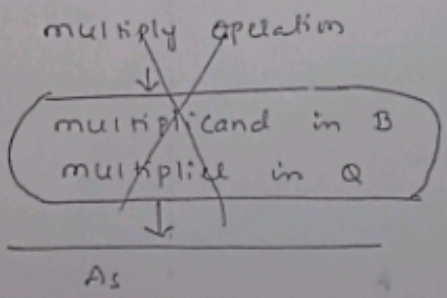
Hardware Algorithm

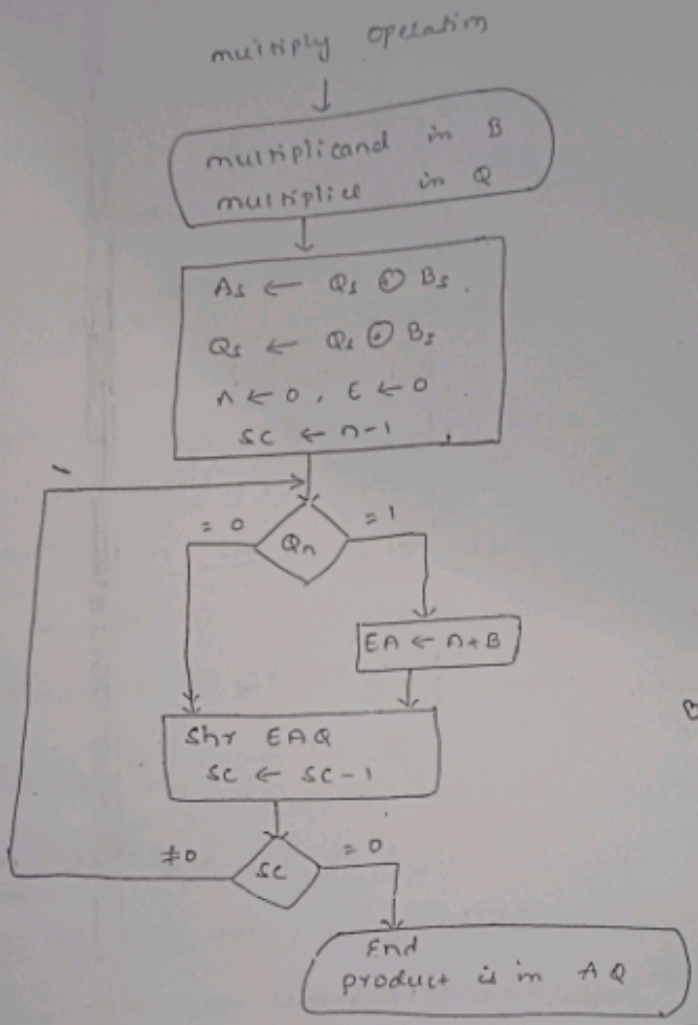
→ Initially multiplicand is in 'B' reg & multiplier in 'Q' reg. Signs are stored in B_s & Q_s reg. Reg A & E are cleared. SC is set to a no' Equal to the no. of bits of the multiplier.

→ Assume that operands are transfered to reg's from a memory unit, that has words of n bits. Operand must be stored with its sign. In the word one of the bit is sign bit & magnitude consists of n-1 bits.

→ After the initialization, the LSB of the multiplier in Q_n is tested. If it is '1', the multiplicand in 'B' is added to the present partial product in A. If it is '0', nothing is done.

→ EAQ reg's all shifted towards right by one bit position. It forms new partial product. The SC is ↓ by 1 and its new value checked. If it is not Equal to zero, the process is repeated until SC = 0. Final product is available in both A & Q. 'A' hold MSB's & 'Q' holds the least significant bits.





B → 10111 → 23
Q = 10011 → 19

23
×19
437

-multiplicand B = 10111	E	A	Q	SC
multiplier in Q	0	00000	10011	101
$Q_n = 1$, add B	0	10111		
first partial product	0	01011	11001	100
shift right EAQ				
$Q_n = 1$, add B		10111		
second partial product	1	00010		011
shift right EAQ	0	10001	01100	010
$Q_n = 0$, SHR EAQ	0	01000	10110	001
$Q_n = 0$ SHR EAQ	0	00100	01011	000
$Q_n = 1$, add B		10111		
fifth partial product	1	01011	01011	
SHR EAQ	0	00101	10101	

Booth multiplication Algorithm

→ with the help of B.M.A, multiplying binary integers in signed 2's comp.. form.

→ It operates that the strings of 0's in the multiplier reduce no addition, but only shifting.

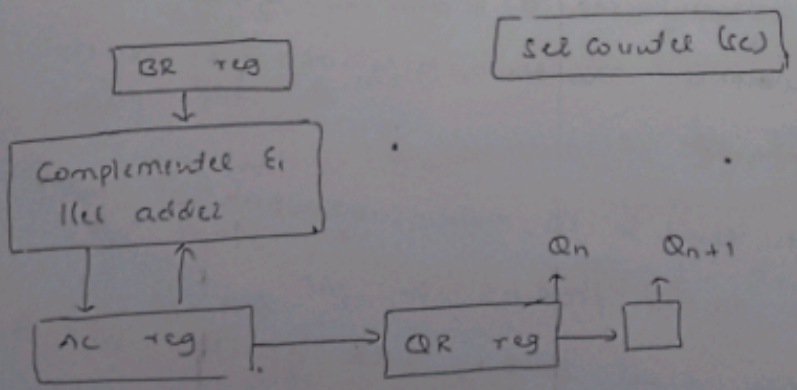
→ string of 1's in the multiplier from bit weight 2^k to 2^m can be treated as $2^{k+1} - 2^m$.

→ For Ex: the binary no' 0001110 (+14) has a string of 1's from 2^3 to 2^1 ($k=3, m=1$). The no' can be represented as $2^{k+1} - 2^m = 2^4 - 2^1 = 16 - 2 = 14$.

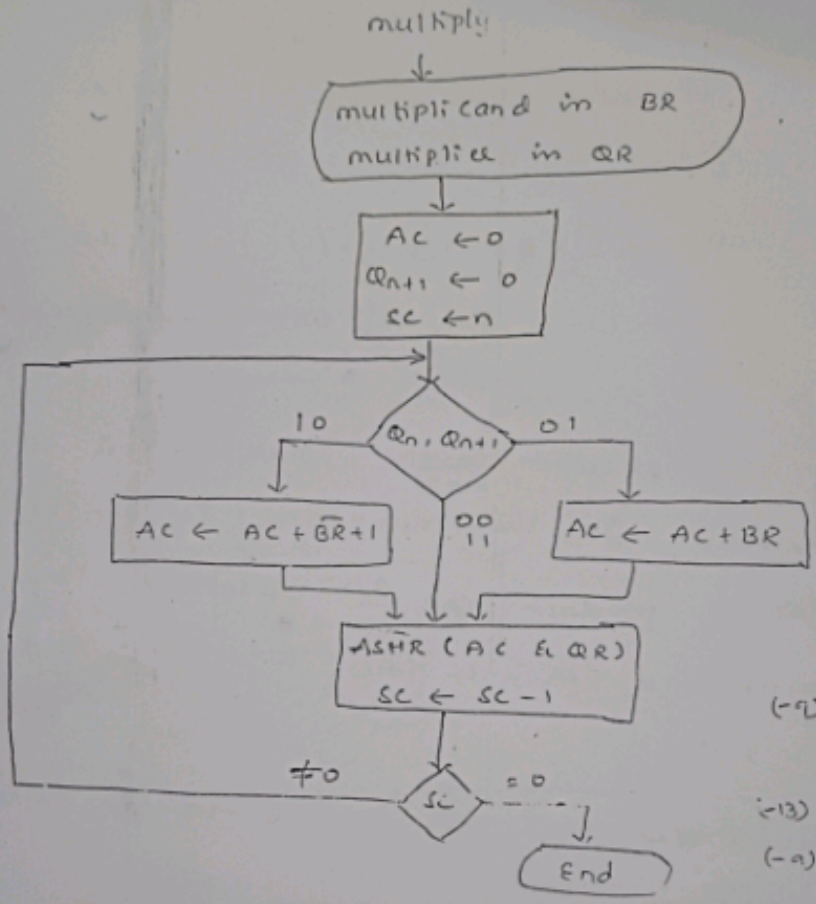
→ Therefore, the multiplication $m \times 14$, where 'm' is the multiplicand and 14 is the multiplier. It can be done as $m \times 2^4 - m \times 2^1$. This product can be obtained by shifting the binary multiplicand 'm' 4 times to left and subtracting 'm' shifted to 1 time.

→ This algorithm works for both the +ve & -ve multipliers in 2's comp.. form.

H/W for Booth Algorithm



→ Q_n stores LSB of the multiplier in reg QR. An extra FF Q_{n+1} is appended to QR to facilitate a double bit inspection of the multiplier.



9 → 0 1001
(−9) 2's comp → 1 0 1 1 1
13 → 0 1 1 0 1
(−13) 2's comp → 1 0 0 1 1
(−9)·(−13) = +117

Fig: Booth Algorithm for multiplication of Signed 2's comp.. no's.

→ AC & Q_{n+1} bits are initially set to '0' and SC is set to a no of bits in the multiplier. The two bits of the multiplier in Q_n & Q_{n+1} are compared.

→ If the two bits are equal to 10, a subtraction of the multiplicand from the partial product in AC.

→ If the two bits are equal to 01, the addition of the multiplicand to the partial product in AC.

→ When two bits are Equal, the partial product doesn't change

→ Next step is Arithmetic shift right of (7)
reg's AC, QR, Q_{n+1}. SC ⊖ is ↓ by 1. Loop is repeated until SC = 0.

Q_n	Q_{n+1}	BR = 10111, BR+1 = 01001	AC	QR	Q_{n+1}	SC
1	0	Initial	00000	10011	0	101
		Subtract BR	01001			
			01001			
		ASHR	00100	11001	1	100
1	1	ASHR	00010	01100	1	011
0	1	Add BR	10111			
			11001			
		ASHR	11100	10110	0	010
0	0	ASHR	11110	01011	0	001
1	0	Sub BR	01001			
			1 00111			
		ASHR	00011	10101	1	000

Division Algorithm

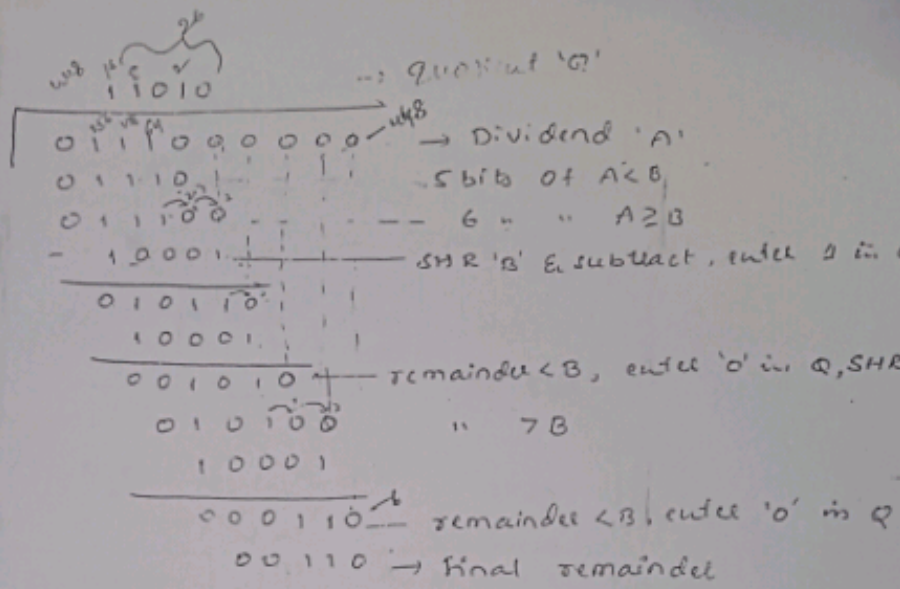
→ The divisor 'B' consists of 5-bits, and Dividend has 10-bits. The 5 msBits of the dividend are compared with the divisor. The 5-bit no' is smaller than B, we try again by taking the 6 ms bits of A & compare this no' with B.

→ The 6-bit no' is greater than B, we place 1 for the quotient bit. The Divisor is then shifted once to the right and subtracted from the Dividend. The Difference is called Partial remainder.

→ If the partial remainder is smaller than the divisor, the quotient bit is 0, and no

H/w implementation for signed-magnitude Data

→ with the help of paper & pencil binary Division is given by



→ Binary Division with the help of Digital H/w

→ EAQ reg's are now shifted towards left, '0' is inserted into Qn, and the previous value of 'E' is lost.

→ The Divisor is stored in the B reg & the Double length dividend is stored in reg's A & Q. The Dividend is shifted to the left, and the divisor is subtracted by adding its 2's comp. value.

→ If E=1, it signifies that A≥B. quotient bit 1 is inserted into Qn, & the partial remainder is shifted to the left to repeat the process.

→ If E=0, it signifies that A<B, so quotient in Qn remains '0'. The value of 'B' is added to 'A'. The partial remainder is shifted to the left, and the process is repeated until all 5 quotient bits are formed.

→ The quotient is stored in 'Q' & remainder in 'A'

Divisor B = 10001 $\quad \bar{B}+1 = 01111$

	E	A	Q	SC
Dividend		01110	00000	5
SHL EAQ	0	11100	00000	
Add $\bar{B}+1$	1	01111		
E = 1, set $Q_n = 1$	1	01011	00001	4
SHL EAQ	0	10110	00010	
Add $\bar{B}+1$	1	01111		
E = 1, set $Q_n = 1$	1	00101	00011	3
SHL EAQ	0	01010	00110	
Add $\bar{B}+1$		01111		
		11001		
E = 0, leave $Q_n = 0$	0	11001	00110	2
Add B		10001		
Restore remainder	1	01010		
SHL EAQ	0	10100	01100	
Add $\bar{B}+1$		01111		
		00011		
Set $Q_n = 1$	1	00011	01101	1
SHL EAQ	0	00110	11010	
Add $\bar{B}+1$		01111		
	0	10101		
E = 0, leave $Q_n = 0$	0	10101	11010	
Add B		10001		
	1	00110		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A		00110		
Quotient in Q			11010	

Divide overflow:- The quotient will consists of six bits if the 5 MSB bits of the dividend constitute a no' greater than the divisor.

→ The Quotient is to be stored in a standard 5-bit reg.

→ In some older computers, the occurance of a devide overflow stopped the computer & is referred to as a Divide stop.

→ Floating pt data can be used to ~~stop~~ avoid DVF.

1) If the bit shifted into E is '1', then EA ≥ B, becz EA consists of a '1' followed by n-1 bits. while 'B' consists of only (n-1) bits

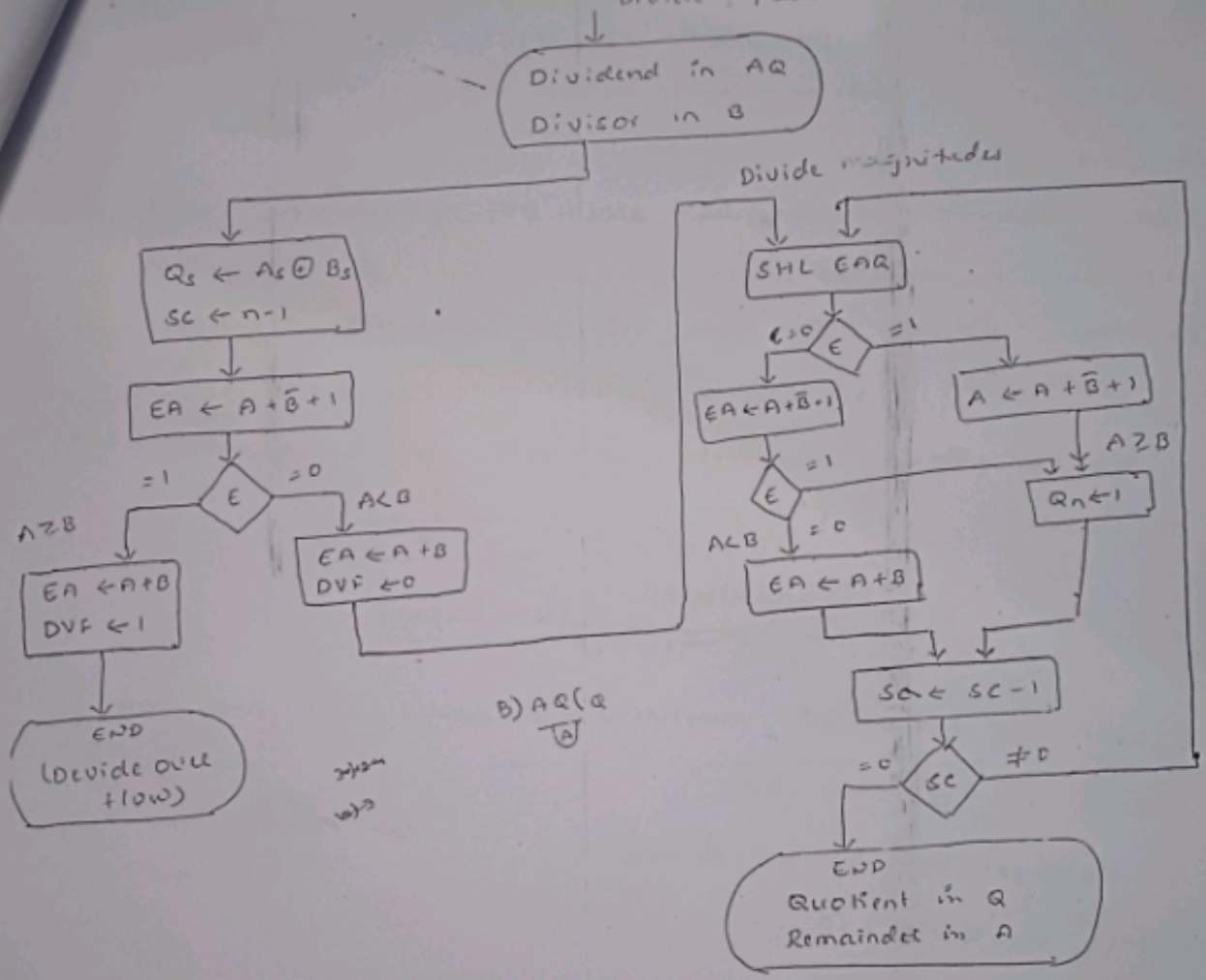
→ In this case, 'B' must be subtracted from EA & '1' inserted into Q_n.

→ The carry from this addition is not transfered to 'E' if we want E to remain a 1.

2) If the bit shifted in E is '0', the divisor is subtracted by adding 2's comp.. value & the carry is transferred into E.

If E = 1, then $Q_n = 1$

Flowchart for Divide Operation



B) AQ(Q
A

Floating point Arithmetic Operations:

→ The floating pt no's are specified by an real declaration statement

→ Fixed pt no's are " " Integer "

statement.

→ Arithmetic operations with floating pt no's are more complicated than fixed pt no's

→ Adding (or subtracting) two no's requires first an alignment of the radix pt. Since the exponent parts must be made equal before adding (or subtracting) the mantissas.

→ The alignment is done by shifting one mantissa while its exponent is adjusted until it is equal to the other exponent. Consider the sum of the following floating pt no's

$$.5372400 \times 10^{2} + .1580000 \times 10^{-1}$$

→ The alignment when the exponents are equal, mantissas are added.

$$\begin{array}{l} .5372400 \times 10^{2} \\ .0001580 \times 10^{2} \\ \hline .5373980 \times 10^{2} \end{array}$$

→ When two normalized mantissas are added, the sum may contain an overflow digit.

→ A floating pt no' has a '0' in the most significant position of the mantissa is said to be underflow.

→ Floating pt multiplication & Division don't require an alignment of the mantissas. The product can be formed by multiplying the two mantissas and adding the exponents

→ Division is accomplished by Dividing the mantissas and Subtracting the exponents.

→ There are 4 ways to represent the biased exponents.

1) signed - magnitude
2) " 1's comp.
3) " 2's "
4) Biased exponent

→ The bias is a fixed no that is added to each exponent as the floating pt no is formed.

Ex:- Exponent range from -50 to 49 (e+50)
0 to 99

* Register Configuration:-

→ There are 3 reg. BR, AC, QR. Each reg is subdivided into 2 parts

→ The mantissa part is represented with the help of upper case letter symbol.

→ Exponent is represented with the " " lower case letter symbol.

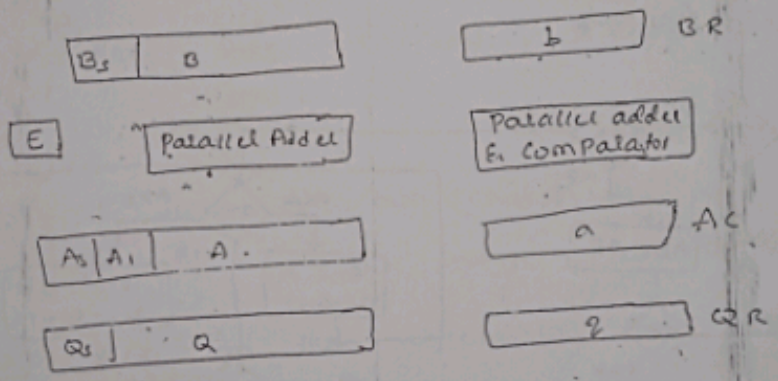


Fig: reg for floating pt. arithmetic operation

→ Floating pt no has a mantissa in signed magnitude representation & a biased exponent.

→ AC has a mantissa whose sign is in A_s, mag is in A. Exponent is stored in a reg represented by lower case letter 'a'.

Addition & Subtraction :

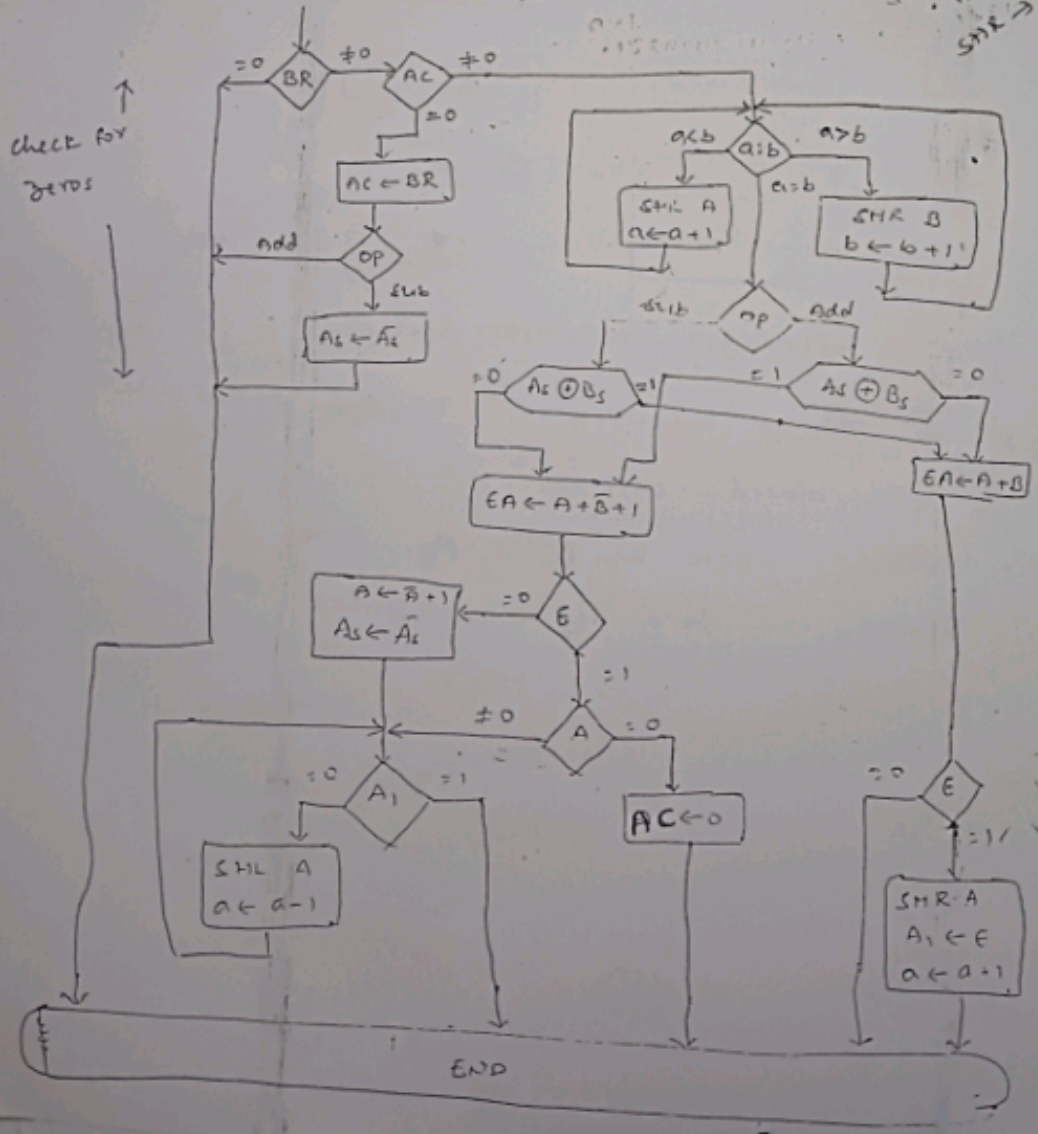
→ During addition (or) subtraction, the two floating pt. operands are in AC & BR. The sum (or) difference is formed in the AC. The algorithm is divided into 4 parts.

1) check for zeros
2) Align the mantissas
3) Add (or) subtract the mantissas
4) Normalize the result

If the MSB of floating point number is non-zero, it is called normalization

→ A floating pt no' that is zero can't be normalized.

Add (or) subtract



check for zeros

Align mantissas

mantissa addition (or) subtraction

Normalization

→ If BR is equal to zero, the operation is terminated, the value in the AC is the result. If AC is equal to zero, we transfer the contents of BR into AC and also complement its sign if the nos are to be subtracted. If neither no' is equal to zero, we proceed to align the mantissas.

→ mag.. Comparator compares the exponents. If two exponent are equal, perform arithmetic operation.

→ If exponents are not equal, the mantissas having the smaller exponent is shifted to the right and its exponent incremented. This process is repeated until the two exponents are equal.

→ The addition & sub.. of two mantissas are identical to the fixed pt .. .. The mag.. part is added or subtracted depending on the operation and the signs of the two mantissas.

→ If an OF occurs when the mag.. are added, it is transferred into E FF. If E = 1, the bit is transferred into A1, and all other bits of A are shifted right.

→ The exponent must be incremented to maintain the correct no. No OF can occur bcoz the original mantissa that was not shifted during the alignment was already in a normalized form.

→ If the mag.. were subtracted, the result may be zero, or may have an underflow. If the mantissa is zero,

the entire floating pt no' in the AC is made zero. otherwise the mantissa must have at least one bit that is filled to 1.

→ The mantissa has an underflow if the most significant bit in position A_1 is 0. In that case, the mantissa is shifted left and the exponent decremented.

→ The bit in A_1 is checked again and the process is repeated until it is filled to 1. When $A_1 = 1$, the mantissa is normalized and the operation is completed.

multiplication :-

→ The mul. of two floating pt no's requires that we multiply the mantissas & add the exponents. There is no comparision of exponents & alignment of mantissas.

→ The multiplication algorithm can be subdevided into 4 parts.

1) check for zeros
2) Add the exponents
3) multiply the mantissas
4) Normalize the product

→ The two operands are checked to determine if they contain a zero. If either operand is filled to zero, the product in the AC is set to zero and the operation is terminated. If neither of the operand is filled to zero, the process continues with the exponent addition

→ The exponent of the multiplier is in 'q' and the addn is b/n exponents a and b.

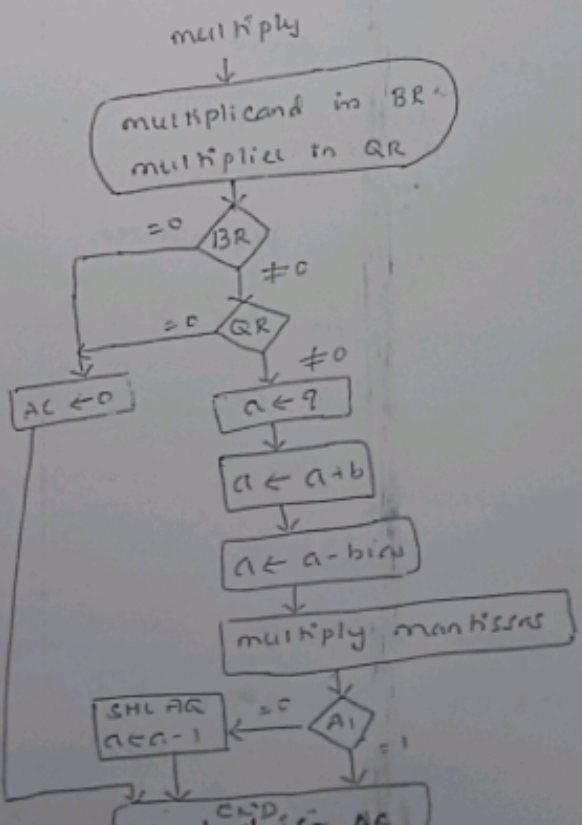
It is necessary to transfer the exponents from q to a. Add the two exponents, and transfer the sum into a.

→ Since both exponents are biased by the addition of a constant, the exponent sum will have double this bias.

→ The correct biased exponent for the product is obtained by subtracting the bias no from the sum.

→ multiplication of mantissas which is same as that of fixed pt no's. Result is stored in A & Q reg. overflow can't occur during multiplication, so there is no need to check for it.

→ The product may have an underflow, the msb in A is checked. If it is 1, the product is already normalized. If it is 0, the mantissa in AQ is shifted left and the exponent decremented.



Division

→ Floating pt division requires that the exponents be subtracted and the mantissas devided. The mantissa division is done as in that of fixed pt no's.

→ If dividend is greater than divisor devide overflow occurs. To avoid this devidend is shifted to right & its exponent is ↑ by 1. It is referred to as devidend alignment.

→ The devision algorithm is subdevided into 5 parts.

1) check for zeros
2) Initialize reg's & evaluate the sign
3) Align the dividend.
4) Subtract the exponents.
5) Devide the mantissas.

→ The two operands are checked for zero. If the divisor is zero, it indicates that devide by zero. which is an illegal operation operation is terminated.

→ If the dividend in Ac is zero, the quotient in QR is zero, and operation terminated.

→ If the operands are not zero, determine the sign of the quotient & stored in Qs.

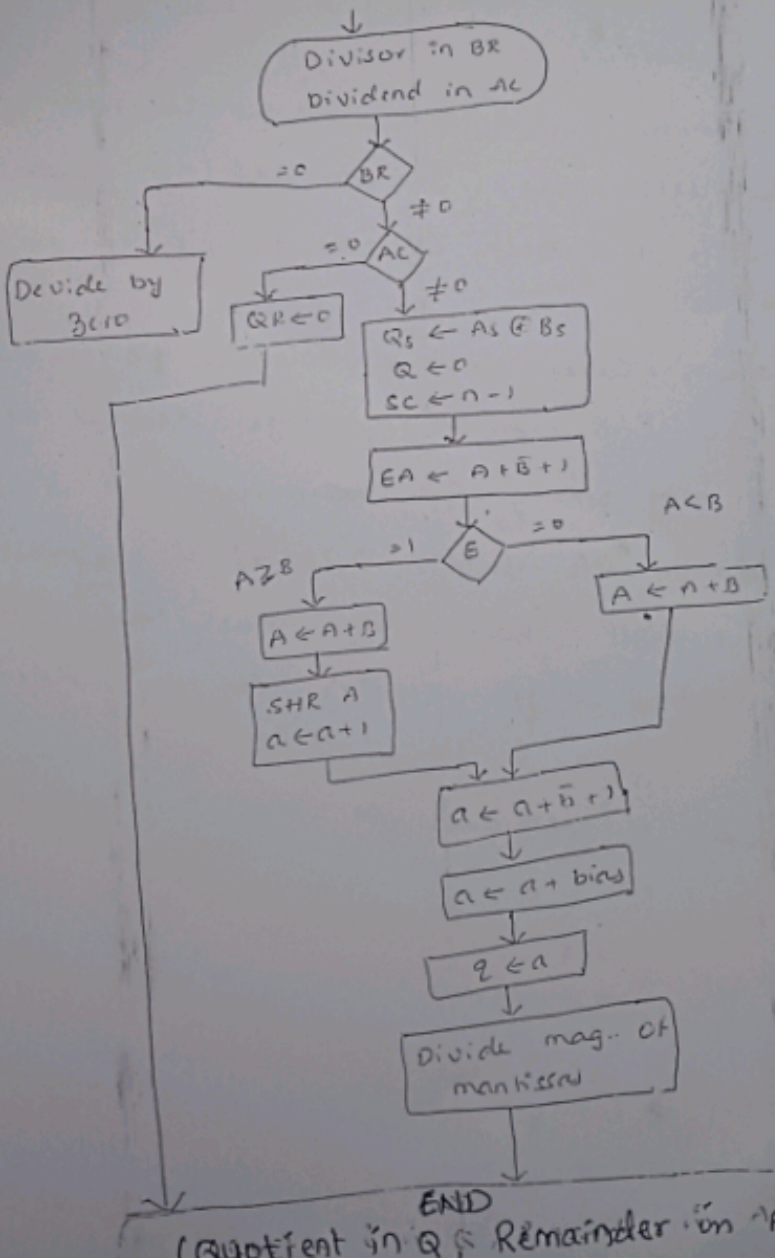
Division operation simply nothing but subtraction & check the value of E. If E=0, the divided fraction is restored to its original value by adding the divisor.

→ If A≥B, it is necessary to shift A once to right and increment the dividend exponent.

→ Divisor exponent is subtracted from the dividend exponent. Both exponents were originally biased, the subtraction operation gives the difference without the bias. The bias is then added and the result is then transferred into 'q'.

→ Devide the mantissas which is same as that of fixed pt no's.

→ Quotient is stored in .Q & Remainder in A.



Decimal Arithmetic unit

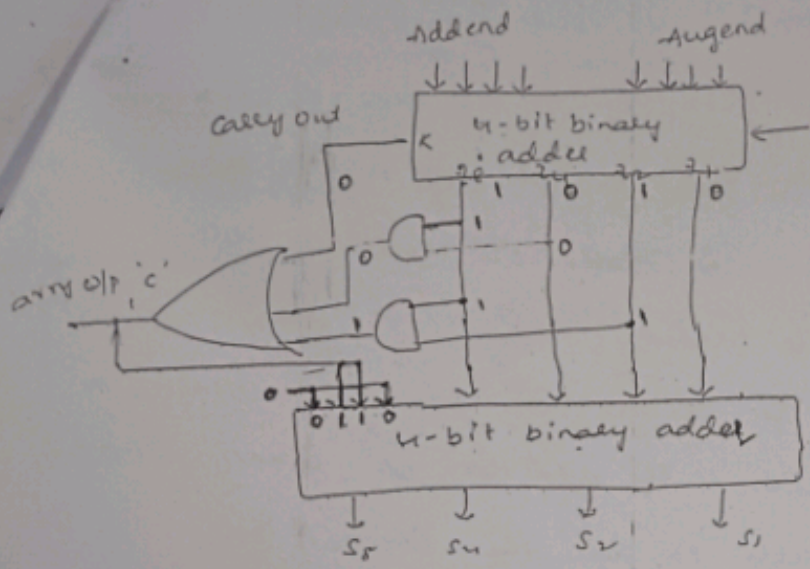
→ The use of a computer prepares data with decimal no's & received results in decimal form. A CPU with an ALU can perform arithmetic operations with binary data. → To perform arithmetic operations with decimal data, it is necessary to convert the i/p decimal no's to binary, to perform all calculations with binary no's, and convert the results into decimal.

→ A decimal arithmetic unit is a digital function that performs decimal operations. It can add (or subtract decimal no's with the help of 9's or 10's comp.. of the subtrahend.

→ → The unit accepts coded decimal no's & generates results in binary form. A single stage decimal arithmetic unit consists of nine binary i/p variables and 5 binary o/p variables. A min.. of 4 bits required to represent each coded decimal digit. Each stage must have 8 i/p's for the augend digit, 4 i/p's for the addend digit, & 1 i/p carry.

→ The o/p's include 4 terminals for the sum digit, and 1 for the o/p carry.

BCD Adder



Binary sum

K	Z_8	Z_4	Z_2	Z_1
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1

BCD sum

C	S_8	S_4	S_2	S_1	Decimal
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
0	1	0	0	1	9
1	0	0	0	0	10
1	0	0	0	1	11
1	0	0	1	0	12
1	0	0	1	1	13
1	0	1	0	0	14
1	0	1	0	1	15
1	0	1	1	0	16
1	0	1	1	1	17
1	1	0	0	0	18
1	1	0	0	1	19

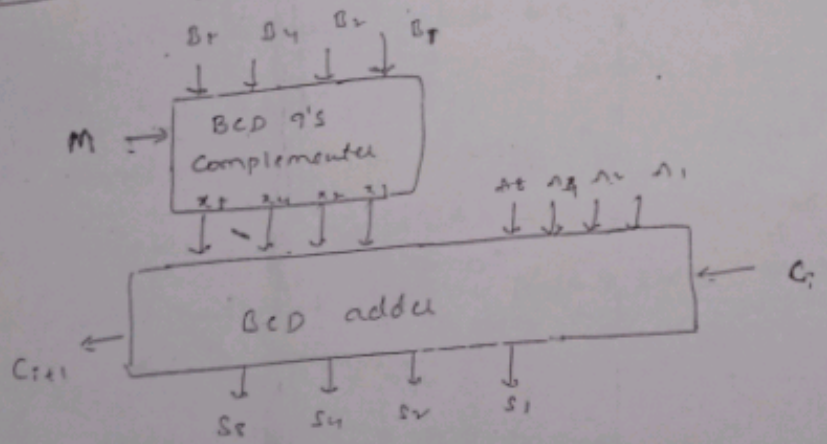
→ Apply two BCD digits to a 4-bit Binary adder. The adder will form the sum in binary & produce a result in the range of 0 to 19.

→ If the binary sum is equal to or less than 1001, the corresponding BCD no' is identical to the binary sum. So no' conversion is needed.

→ When the binary sum is greater than 1001, we obtain a non valid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an o/p carry.

→ $C = K + Z_8 Z_4 + Z_8 Z_2$

BCD Subtraction



→ Above CKT can perform both addition & subtraction of two BCD no's.

→ The mode 'm' controls the operation of the unit. With m=0, the 5 o/p's form the sum of A & B.

→ With m=1, the 5 o/p's form the sum of A plus the 9's complement of B. Apply '1' to the C_i of the first stage

→ The o/p is A + 10's comp.. of B. which is equivalent to the subtraction operation, if the carry out of the last stage is discarded.

→ Ex:- method I
9's comp.. of BCD 0111, is computed by first complementing each bit to obtain 1000
Adding binary 1010 & discarding carry, we obtain 0010

method II
add 0110 to 0111

```
  0110
+ 0111
------
  1101
```

Complement each bit 0010

Decimal Arithmetic operations

Decimal Arithmetic operation symbols:

Symbolic Designation	Description
$A \leftarrow A + B$	Add decimal no's & result transfered to A:
$A \leftarrow A + \bar{B} + 1$	9's comp. of B. Contents of A + 10's comp.. B & result transfered to A.
$Q_L \leftarrow Q_L + 1$	Incremend BCD no' in Q_L.
dshr A	Decimal shift right reg A
dshL A	" " left " "

→ for Ex:- decimal 7660 in BCD

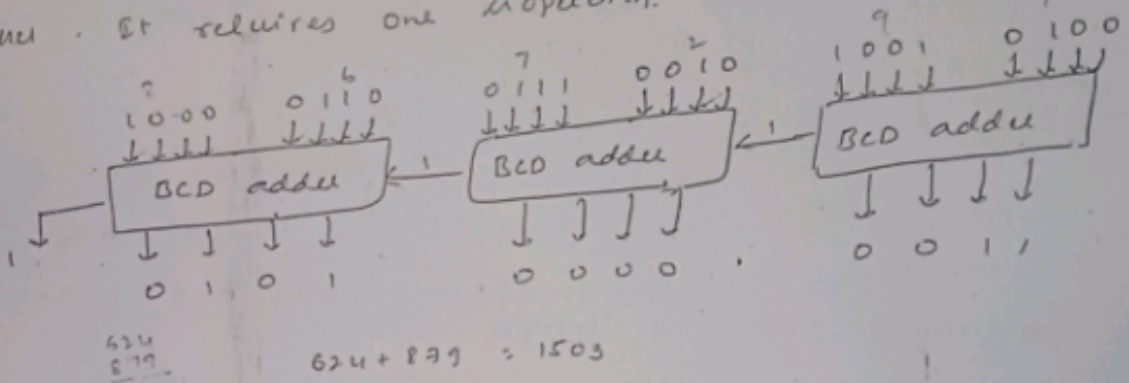
0111 1000 0110 0000 (7660)

dshr A 0000 0111 1000 0110 (0766)
0766

Addition & subtraction

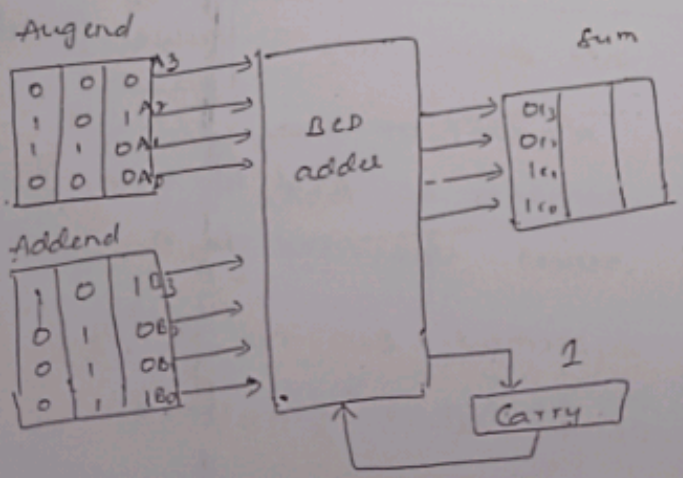
→ Decimal data can be added in 3 diff.. ways

1) The parallel method uses a decimal arithmetic unit composed of many BCD adders. The sum is formed in parallel. It requires one operation.



624 + 879 = 1503

2) Digit-serial bit parallel method, the digits are applied to a single BCD adder serially, & coded bits all transfered in parallel.



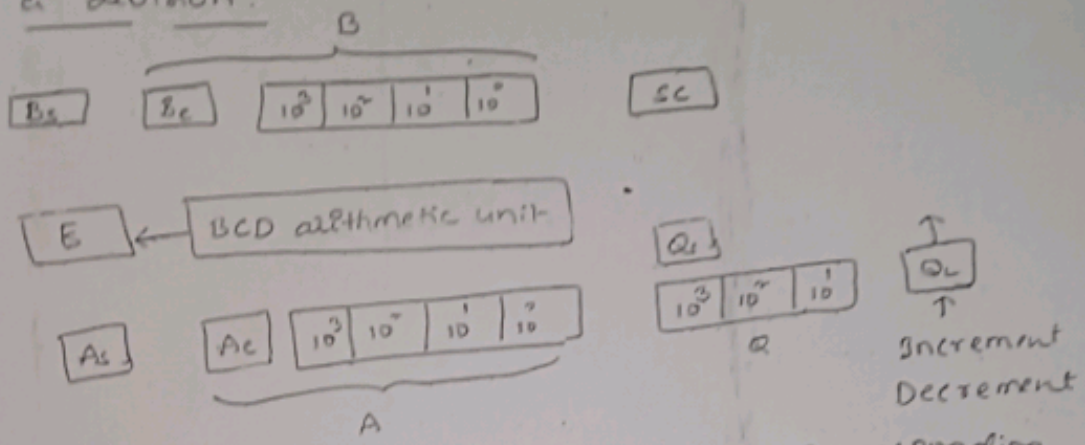
0	1	0	0
1	0	0	1
1	1	0	1

result is 79; to add '6'

1101
0110
0011
C3 C2 C1 C0

→ The sum is formed by shifting the decimal no's through the BCD adder one at a time. The binary sum formed after 4 shifts must be corrected into a valid BCD digit. If the result is 79 add 6 to the binary sum and generating a carry for the next pair of digits.

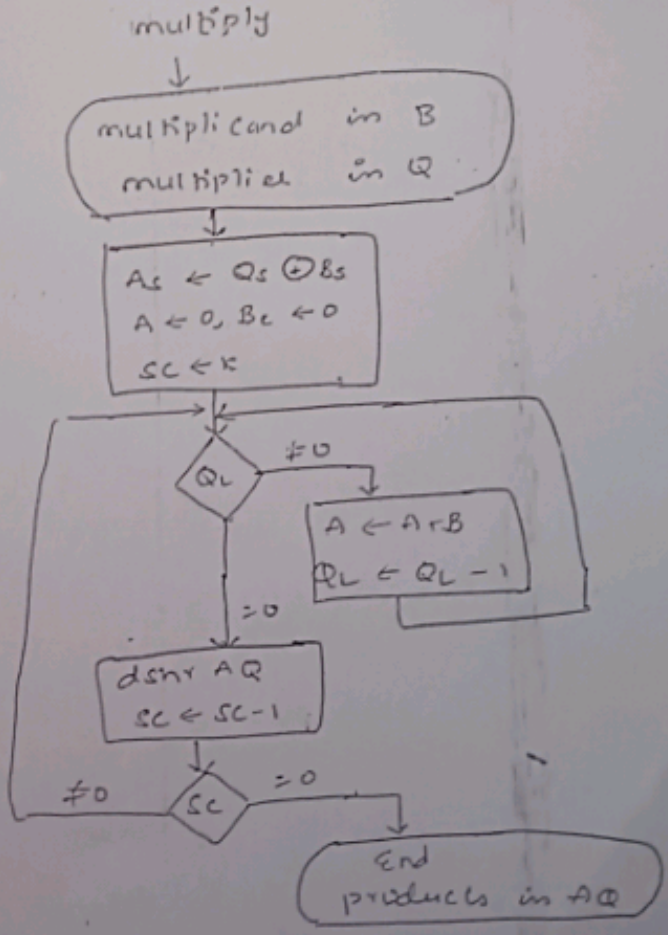
Regr's for decimal Arithmetic multiplication & division:



→ There are 3 reg's. A, B, Q each having a corresponding sign FF As, Bs, Qs.

→ Reg's A & B have four more bits designated by Ae, Be, The purpose of digit Ae is to detect

Binary multiplication:



→ Initially A & Bc are cleared & Sc is set to a no' k equal to the no. of digits in the multiplier.

• The low-order digit of the multiplier in Qr is checked. If it is not equal to 0, the multiplicand in B is added to the partial product in A and Qc is decremented. Qc is checked again and the process is repeated until it is equal to 0.

• The multiplicand in B is added to the partial product a no. of times equal to the multiplier digit.

• The partial product & the multiplier are shifted once to the right. This places zero in Ac and transfers the next multiplier quotient into Qc. The process is repeated k times to form a double length product in AQ.