COA - UNIT- 3

Data Representation & Computer Arithmetic

II-I

Data types :-

Binary information in digital computers is stored in memory or processor registers. Registers contain either data or control information.

→ The data types found in the registers of digital computers may be classified as being one of the following categories.

(1) numbers used in arithmetic computations
(2) letters of the alphabet used in data processing
(3) other discrete symbols used for specific purposes

→ The binary number system is the most natural system to use in a digital computer. But sometimes it is convenient to employ different number systems especially the decimal number system, it is used by people to perform arithmetic computations.

Number System :-

Most common number systems are binary, decimal, octal, hexa decimal. & BCD

→ Base & radix r system uses distinct symbols for r digits.

→ positional-value (weight) system : $r^2 r^1 r^0 . r^{-1} r^{-2} r^{-3}$

multiply each digit by an integer power of r and then form the sum of all weighted digits.

→ Decimal number system :-

The decimal number system in everyday use employs the radix 10 system. The 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

Eg :- The string of digits 724.5 is interpreted to represent the quantity

$7 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 + 5 \times 10^{-1}$, that is 7 hundreds plus 2 tens, plus 4 units, plus 5 tenths.

Binary number system :-

The binary number system uses the radix 2. The 2 digit symbols used are 0 and 1.

Eg:- The string of digits 101101 is interpreted to represent the quantity.

$$1\times 2^5+0\times 2^4+1\times 2^3+1\times 2^2+0\times 2^1+1\times 2^0 = 45$$

The distinguish between different radix numbers, the digits will be enclosed in parantheses and the radix of the number inserted as a subscript.

→ For example to show the quantity between decimal and binary forty-five we will write $(101101)_2 = (45)_{10}$.

Octal and hexadecimal System

Besides the decimal and binary number systems, the octal (radix 8) and hexadecimal (radix 16) are important in digital computer work. The eight symbols of the octal system are 0, 1, 2, 3, 4, 5, 6 and 7. The 16 symbols of the hexadecimal system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. The last 6 ~~digits~~ symbols are identical to the letters of the alphabet and can cause confusion sometimes.

when used to represent hexadecimal digits, the symbols A, B, C, D, E, F correspond to the decimal numbers 10, 11, 12, 13, 14, 15 respectively.

→ A number in radix r can be converted to the familiar decimal system by forming the sum of the weighted digits. For example octal 736.4 is converted to decimal as follows

$$(736.4)_8 = 7\times8^2 + 3\times8^1 + 6\times8^0 + 4\times8^{-1}$$
$$= 7\times64 + 3\times8 + 6\times1 + 4/8 = (478.5)_{10}$$

→ The equivalent decimal number of hexadecimal F3 is obtained from the following calculation

$$(F3)_{16} = F\times16^1 + 3\times16^0 = 15\times16 + 3\times1 = (243)_{10}$$

→ Conversion from decimal to its equivalent representation in the radix r system is carried out by separating the number into its integer and fraction parts and converting each part separately.

Conversions :-

Eg :- The conversion of decimal 41.6875 into binary is done by first seperating the number into its integer part 41 and fraction part 0.6875

→ The integer part is converted by dividing 41 by r=2 to give an integer quotient of 20 and a remainder of 1. The quotient is again divided by 2 to give a new quotient and remainder. This process is repeated until the integer quotient becomes 0.

→ The fraction part is converted by multiplying it by .r=2 to give an integer and a fraction. The new fraction (without the integer) is multiplied again by 2 to give a new integer and a new fraction. This process is repeated until the fraction part becomes zero & until the number of digits obtained gives the required accuracy.

Integer = 41

41	
20	1
10	0
5	0
2	1
1	0
0	1

$(41)_{10} = (101001)_2$

Fraction = 0.6875

$$
\begin{array}{r}
0.6875 \\
\times\ 2 \\
\hline
1.3750 \\
\times\ 2 \\
\hline
0.7500 \\
\times\ 2 \\
\hline
1.5000 \\
\times\ 2 \\
\hline
1.0000
\end{array}
$$

$(0.6875)_{10} = (0.1011)_2$

$(41.6875)_{10} = (101001.1011)_2$

Conversion of decimal 41.6875 into binary

BCD (Binary - Coded - Decimal code)

→ each digit of a decimal number is represented by its binary equivalent

8 7 4 (Decimal)

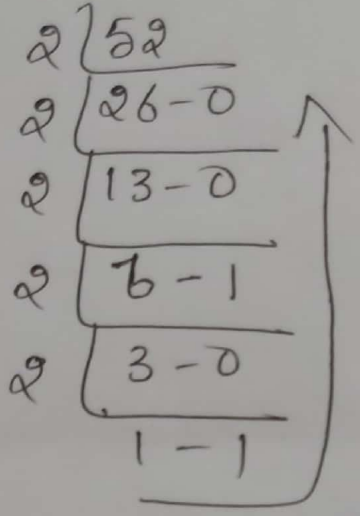
1000 0111 0100 (BCD)

Only the four bit binary numbers from 0000 through 1001 are used.

Conversion of Decimal, octal and hexadecimal numbers into binary numbers

Decimal to binary conversion

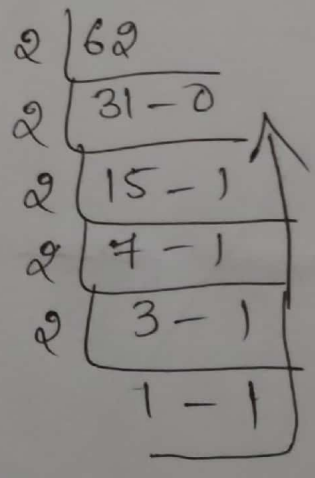
Eg:- $(52)_{10} \Rightarrow (\quad)_2$?



$(52)_{10} \Rightarrow (110100)_2$

octal to binary conversion

Eg:- $(62)_8 \rightarrow (\quad)_2$?



$\therefore (111110)_2$

$(62)_8 = (111110)_2$

Hexadecimal to binary conversion

eg:- $(F3)_{16} \rightarrow (\ )_2$?

$(F3)_{16}$

$\downarrow$

15

2	153
2	76 – 1
2	38 – 0
2	19 – 0
2	9 – 1
2	4 – 1
2	2 – 0
2	1 – 0

$(F3)_{16} \rightarrow (10011001)_2$

Conversion of binary to decimal :-

Eg:- $(1101)_2 \rightarrow (\ )_{10}$?

$1\times2^3+1\times2^2+0\times2^1+1\times2^0 = 8+4+0+1 = 13.$

$(1101)_2 = (13)_{10}.$

Conversion of binary to octal & Hexadecimal

Eg:- 1 0 1 0 1 1 1 1 0 1 1 0 0 0 1 1

→ The conversion from and to binary, octal, and hexadecimal representation plays an important part in digital computers. Since $2^3=8$ and $2^4=16$, each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits.

→ The conversion from binary to octal is easily accomplished by partitioning the binary number into three groups of three bits each.

for example

1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 Binary

1 2 7 5 4 3 octal.

→ Conversion from binary to hexadecimal is similar except that the bits are divided into groups of four.

For example

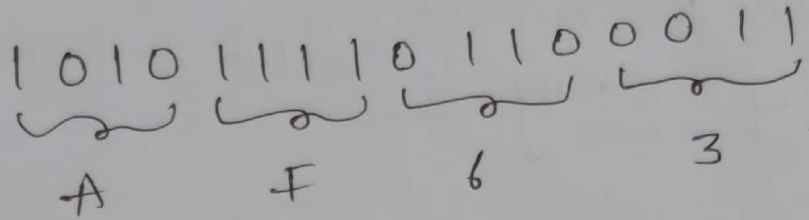


Table Binary-Coded octal numbers

Octal number	Binary-coded octal	Decimal equivalent
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7
10	001 000	8
11	001 001	9
12	001 010	10
24	010 100	20
62	110 010	50
143	001 100 011	99
370	011 111 000	248

code to one octal digit

Binary-Coded Hexadecimal Numbers

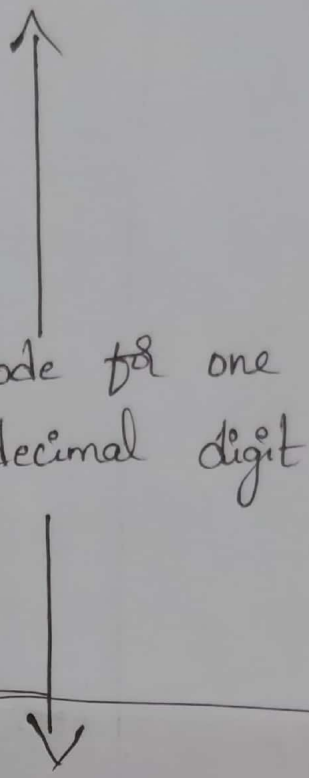
Hexadecimal number	Binary coded Hexadecimal	Decimal equivalent
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Code for one hexadecimal digit

14	0001 0100	20
32	0011 0010	50
63	01100011	99
F8	1111 1000	248

Binary-coded Decimal numbers

Decimal Number	Binary-Coded decimal (BCD) Number
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000
20	0010 0000
50	0101 0000
99	1001 1001
248	0010 0100 1000



Alphanumeric Representation :-

Many applications of digital computers require the handling of data that consist not only of numbers, but also of the letters of the alphabet and certain special characters.

→ An alphanumeric character set is a set of elements that includes the 10 decimal digits, the 26 letters of the alphabet and a number of special characters, such as $., +, and =.

→ The standard alphanumeric binary code is the ASCII (American Standard Code for Information Interchange), which uses seven bits to code 128 characters.

→ The binary code for the uppercase letters, the decimal digits, and a few special characters is listed in table.

→ Note that the decimal digits in ASCII can be converted to BCD by removing the three high-order bits, 011.

Table :- American Standard Code for Information Interchange (ASCII)

Character	Binary code
A	1000001
B	100 0010
C	100 0011
D	100 0100
E	100 0101
F	100 0110
G	100 0111
H	100 1000
I	100 1001
J	100 1010
K	100 1011
L	100 1100
M	100 1101
N	100 1110
O	100 1111
P	101 0000
Q	101 0001
R	101 0010
S	101 0011
T	101 0100
U	101 0101
V	101 0110
W	101 0111
X	101 1000
Y	101 1001
Z	101 1010

Character	Binary code
0	011 0000
1	011 0001
2	011 0010
3	011 0011
4	011 0100
5	011 0101
6	011 0110
7	011 0111
8	011 1000
9	011 1001
Space	010 0000
.	010 1110
(	010 1000
+	010 1011
$	010 0100
*	010 1010
)	010 1001
-	010 1101
/	010 1111
,	010 1100
=	011 1101

Complements :-

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation.

→ There are two types of complements for base r system.

a) r's Complement

b) (r-1)'s Complement

For Binary number : 2's or 1's complement

For Decimal number : 9's & 10's complement.

(r-1)'s Complement

(r-1)'s complement of N = (r-1) - N

* 9's complement of N = 546700

Substitute in formula

(10-1) - 546700 = (1000000-1) - 546700

⇒ 999999 - 546700 = 453299

* 1's complement of N = 101101

$(2^6 - 1) - 101101 = (1000000 - 1) - 101101$

⇒ 111111 - 101101

= 010010.

r's Complement :-

→ r's complement of $N = r^n - N$ for $N \neq 0$

* 10's complement of 2389 is

step 1 :- Find the 9's complement of 2389

step 2 :- Add 1 to the result.

```
  9999
- 2389
------
  7610
```

```
then add    +  1
  +1        ------
            7611
```

The 10's complement of 2389 is 7611

* 2's Complement of 1101100 =

step 1 :- Find the 1's complement of 1101100

step 2 :- Add 1 to the result.

```
          1101100
            ↓
1's       0010011
complement
          +     1
Then add  -------
+1        0010100
```

The 2's complement of 1101100 is 0010100

Subtraction of unsigned Numbers

The subtraction of two n-digit unsigned numbers $M-N (N \neq 0)$ in base r can be done as follows.

1. Add the minuend M to the r's complement of the subtrahend N. This performs $M+(r^n-N) = M-N+r^n$
2. If $M \geq N$, the sum will produce an end carry r^n which is discarded, and what is left is the result M-N.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n-(N-M)$, which is the r's complement of (N-M). To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Consider, for example, the subtraction. 72532 − 13250 = 59282. The 10's complement of 13250 is 86750.

M = 72532

10's complement of N = +86750.

add M & N

$$\begin{array}{r} 72532 \\ +\ 86750 \\ \hline (1)\,59282 \end{array}$$

Discard end carry 10^5 = −100000

= 59282

Now consider an example with M<N. The subtraction 13250 - 72532 produces negative 59282. using the procedure with complements, we have

M = 13250

10's complement of N = +27468

Sum = 40718

There is no end carry

Answer is negative 59282 = 10's complement of 40718

→ Subtraction with complements is done with binary numbers in a similar manner using the same procedure outlined above. Using the two binary numbers X = 1010100 and Y = 1000011, we perform the subtraction X - Y and Y - X using 2's complements.

X = 1010100

2's complement of Y = +0111101

Sum = 10001001

Discard end carry 2^7 = −10000000

Answer X - Y = 0001001

Y = 1000011

2's complement of X = 0101100

Sum = 1101111

There is no end carry

Answer is negative 0010001 = 2's complement of 1101111

Fixed - Point Representation :-

→ Positive integers, including zero, can be represented as unsigned numbers.

→ To represent negative integers, we need a notation for negative values.

→ In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign. Because of hardware limitations computers must represent everything with 1's and 0's. including the sign of number.

→ The convention is to make the sign bit equal to 0 for positive and 1 for negative.

→ In addition to the sign, a number may have a binary (& decimal) point.

→ The position of the binary point is needed to represent fractions, integers, & mixed integer-fraction number.

→ There are two ways of specifying the position of the binary point in a register.

→ By giving it a fixed position Representation

→ By employing a floating point. representation.

Integer Representation :-

When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number. when the number is negative, the sign is represented by 1 but the rest of the number may be represented in one of three possible ways.

1. Signed - magnitude representation
2. Signed 1's complement representation
3. Signed - 0's complement representation.

1. Signed-magnitude representation :-

The Signed-magnitude representation of a negative number consists of the magnitude and a negative sign.

In Signed magnitude representation +14 will be represented as 0.0001110. −14 will ber represented as 1.0001110.

Signed 1's and 2's complement.

The negative number is represented in signed representation and 1's, 2's complement representations.

For example, -14 will be represented in 1's and 2's complement. Consider the signed number 14 stored in 8-bit register. +14 is represented by a sign bit of 0 in the leftmost position.

14 : 00001110.

→ There is only one way to represent +14, there are three different ways to represent -14 with eight bits.

In signed-magnitude representation 1 0001110

In Signed -1's complement representation 1 1110001

In Signed -2's complement representation 1 1110010

Arithmetic Addition :-

The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic. If the signs are same, we add the two magnitude and give the sum the common sign.

If the signs are different, we subtract the smaller magnitude from the larger and give the result the sign of the larger magnitude.

For example

+6	00000110	−6	11111010
+13	00001101	+13	00001101
+19	00010011	+7	00000111

+6	00000110	−6	11111010
−13	11110011	−13	11110011
−7	11111001	−19	11101101

→ For adding numbers in the signed 2's complement system does not require a comparision or subtraction, only addition and complementation. The procedure is very simple.

Step 1 :- add the two numbers, including their sign bits, and discard any carry out of the sign (leftmost) bit position.

To determine the value of a negative number when in signed -2's complement, it is necessary to convert it to a positive number to place it in a more familiar form. For example the signed binary number 11111001 is negative because the leftmost bit is 1. It's 2's complement is 00000111 which is the binary equivalent of +7. We therefore recognize the original negative number to be equal to -7.

Arithmetic Subtraction :-

Subtraction of two signed binary numbers when negative numbers are in 2's complement form is very simple and can be stated follows. Take the 2's complement of the subtrahend. and add it to the minuend. A carry out of the sign bit position is discarded.

This procedure stems from the fact that a subtraction operation can be changed to an addition operation if the sign of the subtrahend is changed.

This is demonstrated by the following relationship.

$$(\pm A) - (+B) = (\pm A) + (-B)$$

$$(\pm A) - (-B) = (\pm A) + (+B)$$

For example

Consider the subtraction of $(-6) - (-13) = +7$.

In binary with eight bits this is written as

11111010 − 11110011. The subtraction is changed to addition by taking the 2's complement of the subtrahend (−13) to give (+13). In binary this is

11111010 + 00001101 = 100000111. Removing the end carry, we obtain the correct answer

00000111 (+7).

Decimal fixed point representation :-

The representation of decimal numbers in registers is a portion of the binary code used to represent a decimal digit. A 4-bit decimal code requires four flip-flops for each decimal digit. The representation of 4385 in BCD requires 16-flip-flops, four flip-flops for each digit.

The number will be represented in a register with 16 flip-flops as follows.

4385 : 0100 0011 1000 0101

Floating-Point Representation :-

The floating-point representation of a number has two parts. The first part represents a signed, fixed-point number called the mantissa.

→ The second ~~point~~ part designates the position of the decimal (or binary) point and is called the exponent.

→ The fixed-point mantissa may be a fraction & can integer.

→ For example, the decimal number +6132.789 is represented in floating-point with a fraction and an exponent as follows:

Fraction	Exponent
+0.6132789	+04

→ The value of the exponent indicates that the actual position of the decimal point is four positions to the right of the indicated decimal point in the fraction.

→ This representation is equivalent to the scientific notation $+0.6132789 \times 10^{+4}$

→ floating point is always interpreted to represent a number in the following form.

$$m \times r^e$$

→ Only the mantissa m and the exponent e are physically represented in the register (including their signs)

→ A floating-point binary number is represented in a similar manner except that it uses base 2 for the exponent.

→ For example, the binary number $+1001.11$ is represented with an 8-bit fraction and 6-bit exponent as follows.

Fraction	Exponent
01001110	000100

→ The fraction has a 0 in the leftmost position to denote positive. The exponent has the equivalent binary number +4. The floating-point number is equivalent to.

$$m \times r^e = +(.1001110)_2 \times 2^{+4}$$

Normalization :-

A floating-point number is said to be normalized if the most significant digit of the mantissa is non zero. For example, the decimal number 350 is normalized but 00035 is not.

→ A zero cannot be normalized because it does not have a non-zero digit.

→ Arithmetic operations with floating-point numbers are more complicated than arithmetic operations with fixed point numbers and their execution takes longer and requires more complex hardware.

→ However floating-point representation is a must for scientific computations because of the scaling problems involved with fixed point computations.

Computer Arithmetic

Addition & Subtraction Algorithms

Introduction :-

Arithmetic instructions in digital computers manipulate data to produce results necessary for the solution of computational problems. The four basic arithmetic operations are addition, subtraction, multiplication and division.

→ We consider addition, subtraction, multiplication and division for the following types of data

1. Fixed-point binary data in signed-magnitude representation.
2. Fixed-point binary data in signed-2's complement representation.
3. Floating-point binary data
4. Binary-coded decimal (BCD) data.

Addition and Subtraction

There are three ways of representing negative fixed-point binary numbers.

→ Signed magnitude, Signed – 1's Complement. or Signed-2's Complement.

→ most computers use the Signed-2's complement representation when performing arithmetic operations with integers.

→ We develop the addition and subtraction algorithms for data represented in Signed-magnitude and again for data represented in Signed-2's Complement.

Addition and subtraction with Signed-magnitude Data

→ The magnitude of the two numbers by A and B when the signed numbers are added & subtracted, we find that there are eight different conditions to consider., depending on the sign of the numbers and the operations performed.

→ the algorithms for addition and subtraction are derived from the table and can be stated as follows.

Addition (Subtraction) algorithms :-

Addition and Subtraction of Signed-Magnitude numbers

Operation	Add magnitudes	Subtraction magnitudes: 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)

For addition operation :- when the signs of A & B ~~are~~ ~~magnitudes~~ equal, add the two magnitudes and attach the sign of A to the result.

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

→ Choose the sign of the result to be same as A if A > B & the complement of the sign of A if A < B.

→ If the two magnitudes are equal, subtract B from A and make the sign of the result positive.

For subtraction operation :-

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

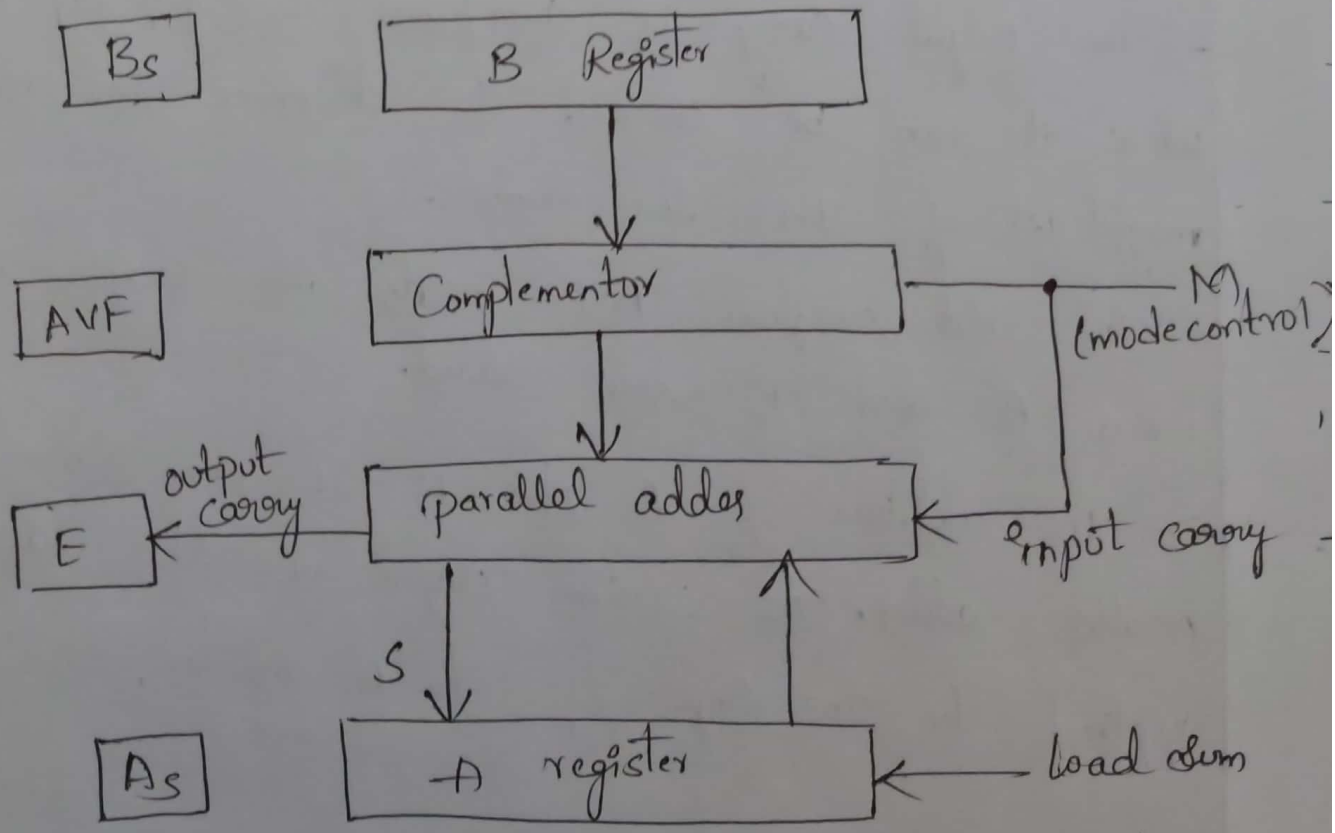
→ When the signs of A and B are different, compare the magnitudes and subtract the smaller number from the larger.

→ Choose the sign of the result to be the same as A if A>B & the complement of the sign of A if A<B.

→ If the two magnitudes were equal, subtract B from A and make the sign of the result positive.

→ The two algorithms were similar except for the sign comparison. The procedure to be followed for identical signs. ~~The addition~~

Hardware implementation :-



Hardware for signed-magnitude addition and subtraction

To implement the two arithmetic operations with hardware, it is first necessary that the two numbers be stored in registers.

→ Let A and B be two registers that hold the magnitudes of the numbers, and As and Bs be two flipflops that hold the corresponding signs.

→ Hardware implementation can be shown in block diagram i.e addition and subtraction operations.

→ It consists of registers A and B and sign flip-flops As and Bs.

→ Subtraction is done by adding A to the 2's complement of B.

→ The output carry is transferred to flip-flop E, where it can be checked to determine the relative magnitudes of the two numbers.

→ The add overflow AVF holds the overflow bit when A and B are added.

→ The addition of A plus B is done through the parallel adder. The S (sum) output of the adder is applied to the input of the A register.

→ The complementer provides an output of B or the complement of B depending on the state of the mode control. M.

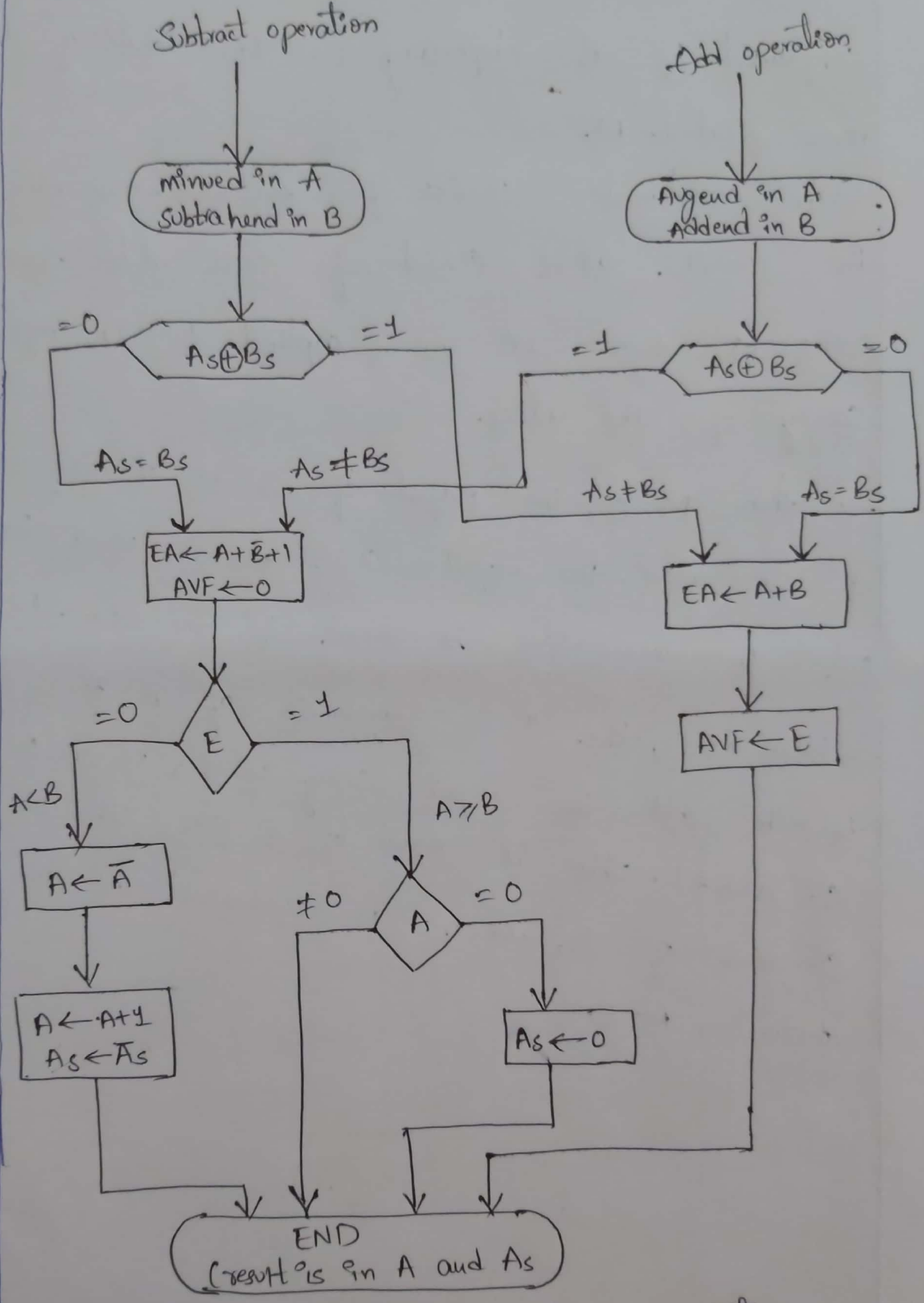
→ The complementer consist of exclusive-OR gates and the parallel adder consist of full-adder circuits.

→ The M signal is also applied to the input carry of the adder.

→ When M=0, the output of B is transferred to the adder, the input carry is 0, and the output of the adder is equal to the sum. A+B.

→ When M=1, the 1's complement of B is applied to the adder, the input carry is 1, and output $S = A + \bar{B} + 1$. This is equal to A plus the 2's complement of B, which is equivalent to the subtraction A−B.

Hardware Algorithm :-



Flowchart for add and subtract operations

→ The flowchart for the hardware algorithm is shown in flowchart. The two signs As and Bs are compared by an exclusive-OR gate. If the output of the gate is 0, the signs are identical; if it is 1, the signs are different.

→ For an add operation, different signs dictate that the magnitudes be added.

→ The magnitudes are added with a microoperation EA ← A+B, where EA is a register that combines E and A.

→ The carry in E after the addition constitutes an overflow if it is equal to 1. The value of E is transferred into the add-overflow flipflop AVF.

→ If the signs are opposite, subtraction is initiated and stored in A.

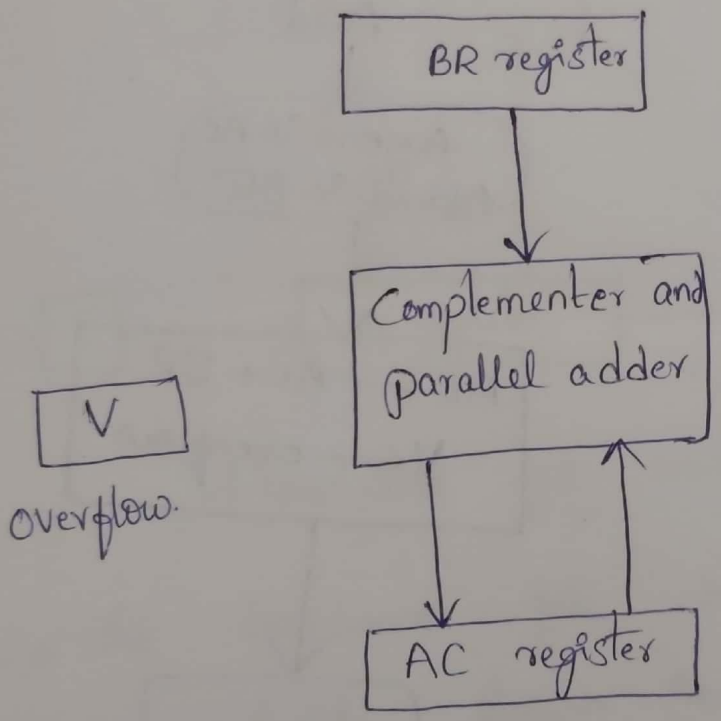
→ No overflow can occur with subtraction so the AVF is cleared.

→ If E = 1, the A ≥ B, However, if A = 0, then A = B and the sign is made positive

→ If E = 0, then A < B and sign for A is complemented.

Addition and Subtraction with Signed-2's Complement Data

The register configuration for the hardware implementation is shown in below. We name the A register AC (accumulator) and the B register BR. The leftmost bit in AC and BR represent the sign bits of the numbers.

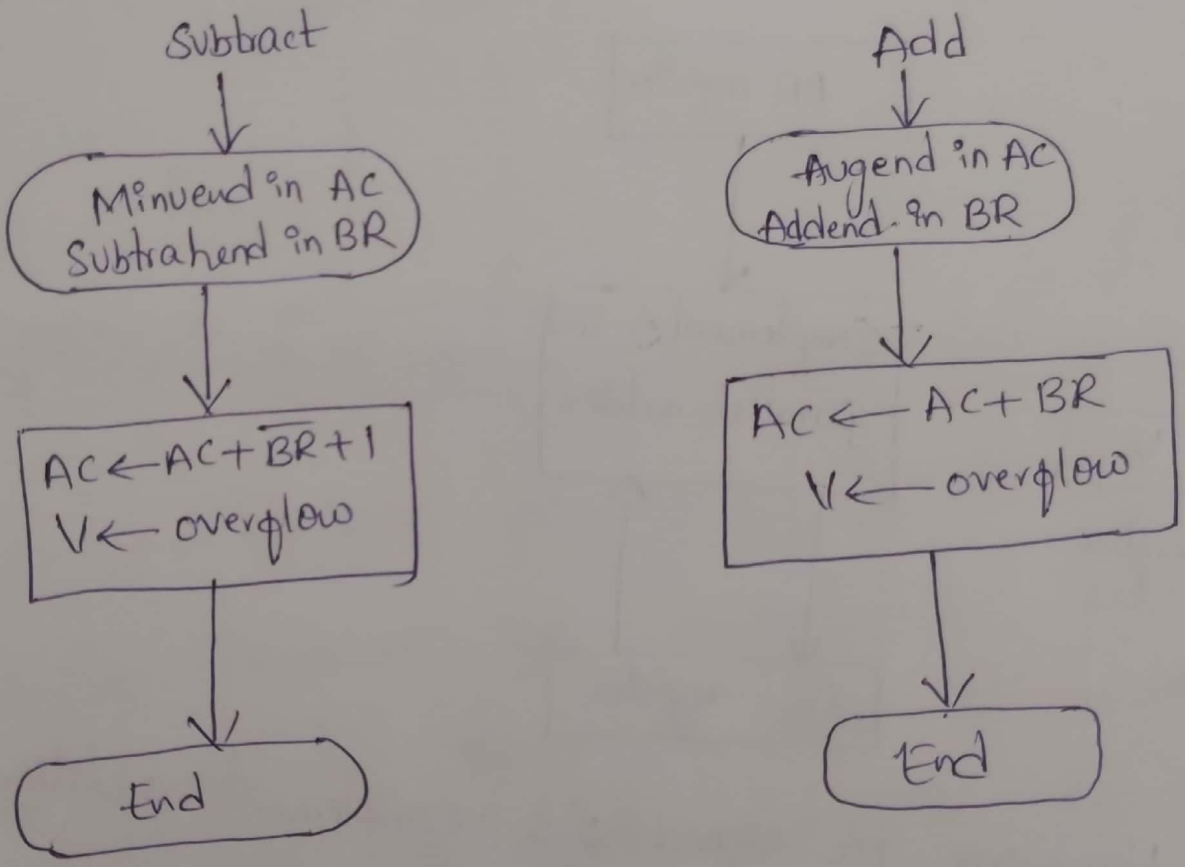


Hardware for signed-2's complement addition and subtraction.

→ The two sign bits are added & subtracted together with the other bits in the complementer and parallel adder.

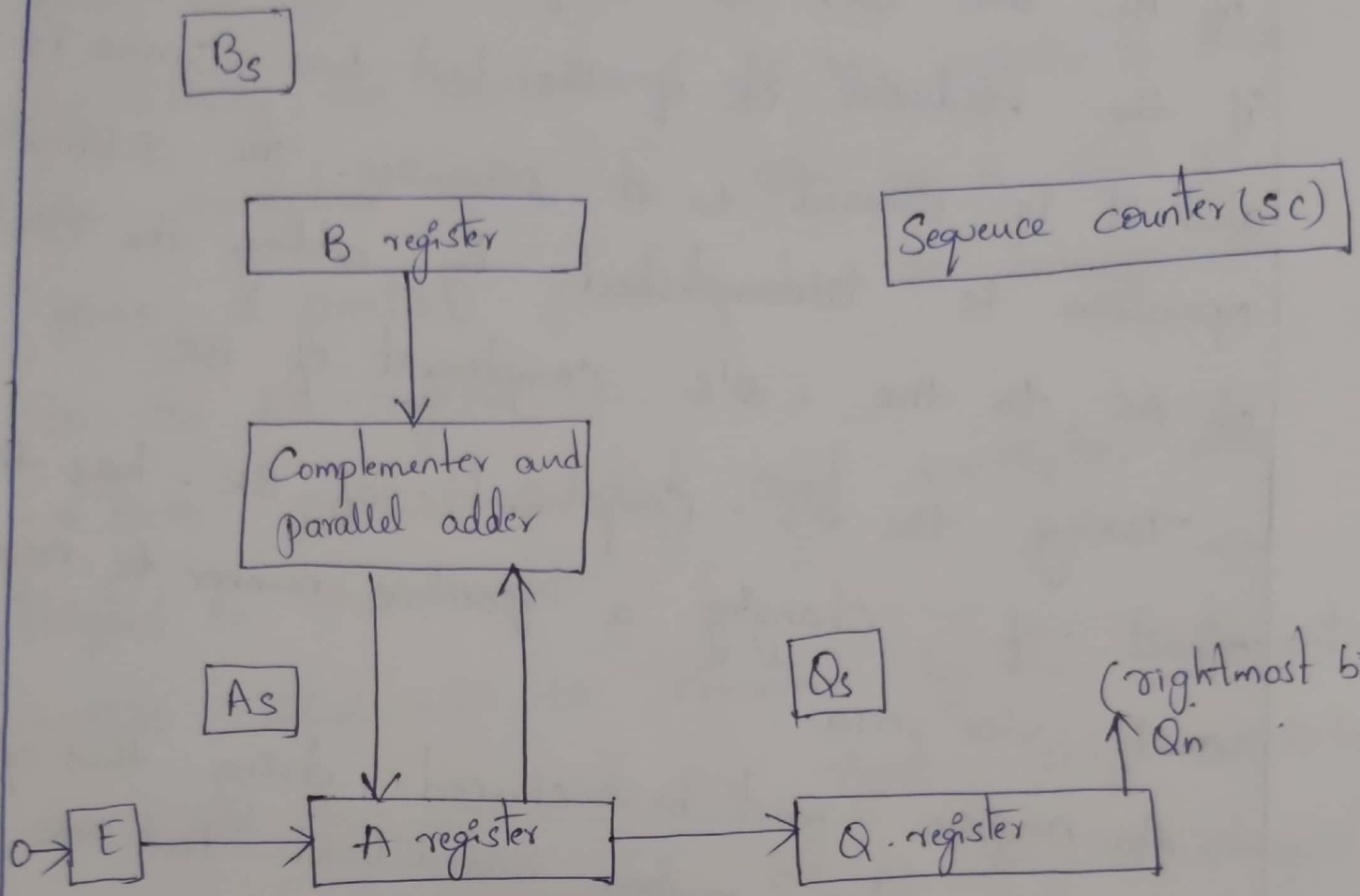
→ The overflow flip-flop V is set to 1 if there is an overflow. The output carry in this case is discarded.

→ The algorithm for adding and subtracting two binary numbers in signed-2's complement representation is shown in flowchart.



Algorithm for adding and subtracting numbers in signed-2's complement representation.

Hardware implementation for signed-Magnetude data



Hardware for multiply operation

→ The hardware for multiplication consist of the equipment shown in above. The multiplier is stored in the Q register and its sign in Qs.

→ The Sequence counter SC is initially set to a number equal to the number of bits in the multiplier.

→ The counter is decremented by 1 after forming each partial product.

→ The sum is obtained by adding the contents of AC and BR. The overflow bit V is set to 1 if the exclusive-OR of the last two carries is 1, and it is cleared to 0 otherwise. The subtraction operation is accomplished by adding the content of AC to the 2's complement of BR.

→ Taking the 2's complement of BR has the effect of changing a positive number to negative, and vice versa.

→ An overflow must be checked during this operation because the two numbers added could have the same sign.

Multiplication Algorithms :-

Multiplication of two fixed-point binary numbers in signed-magnitude representation is

```
                10111    Multiplicand
  23
  19          x10011    Multiplier
  ---------------------
              10111
             10111 +
            00000
           00000
          10111
  ---------------------
  437   110110101   product
```

→ when the content of the counter reaches zero, the product is formed and the process stops.

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

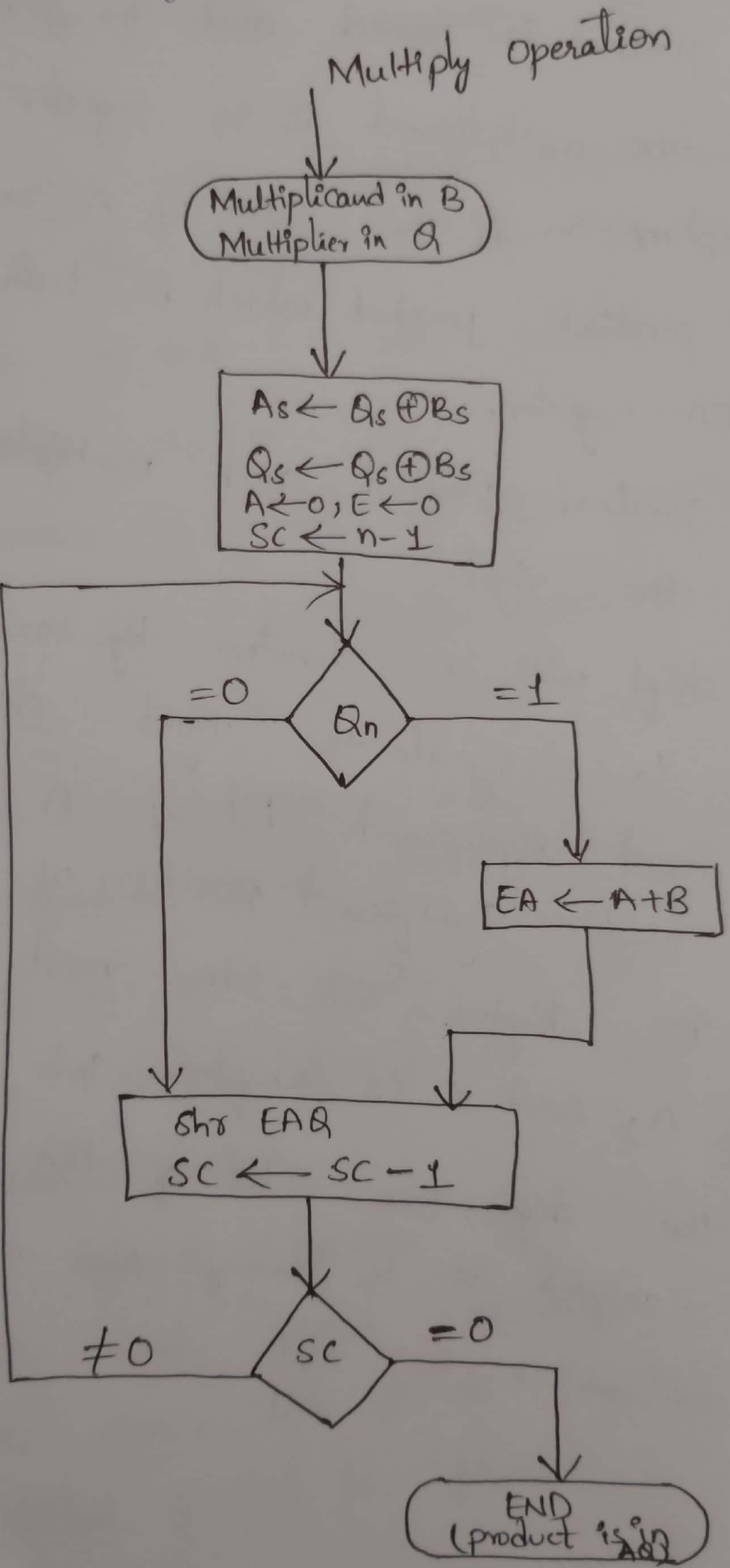
→ Both partial product and multiplier are shifted to the right.

→ This shift will be denoted by the statement shr EAQ to designate the right shift depicted.

→ The least significant bit of A is shifted into the most significant position of Q, the bit from E is shifted into the most significant position of A, and 0 is shifted into E.

→ After the shift, one bit of the partial product is shifted into Q, pushing the multiplier bits one position to the right.

Hardware Algorithm :-



→ Initially, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in Bs and Qs.

→ The signs are compared, and both A and Q are set to correspond to the sign of the product

→ Registers A and E are cleared and the sequence counter SC is set to a number equal to the number of bit of the multiplier.

→ After the initialization, the low-order bit of the multiplier in Qn is tested.

→ If it is a 1, the multiplicand in B is added to the present partial product in A.

→ If it is a 0, nothing is done. Register EAQ is then shifted once to the right to form the new partial product.

→ the sequence counter is decremented by 1 and its new value checked. If it is not equal to zero, the process is repeated and a new partial product is formed.

Numerical Example for Binary number Multiplier

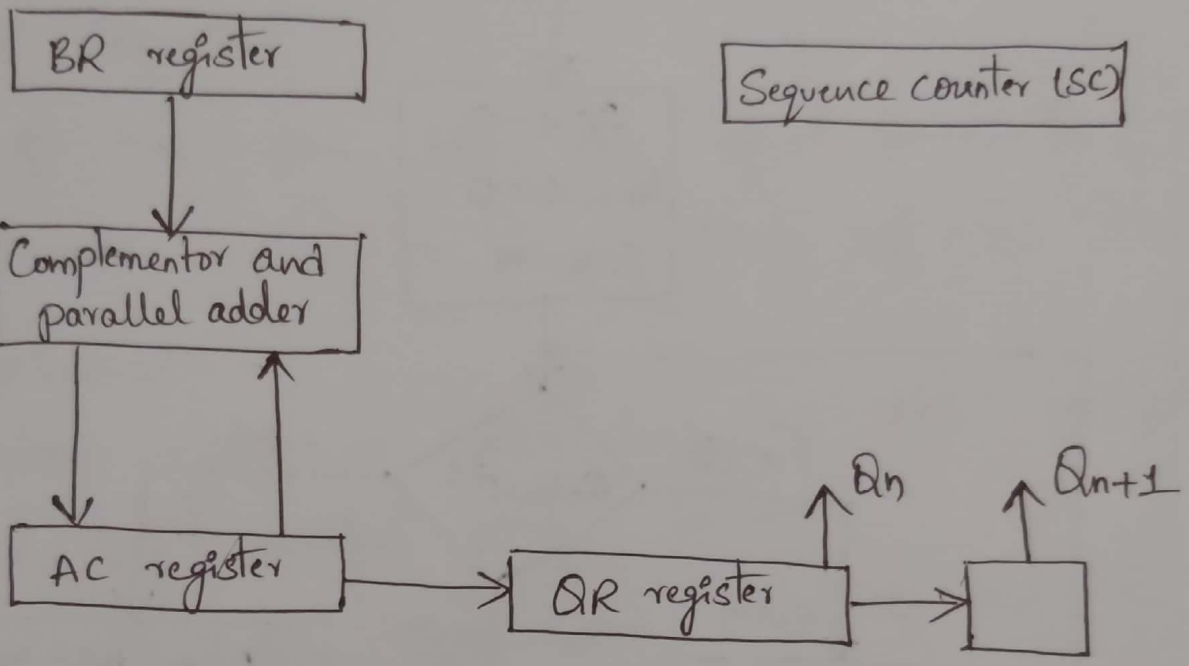
Multiplicand B = 10111

	E	A	Q	SC
Multiplier in Q				
$Q_n = 1$; add B	0	00000	10010	5
		+ 10111		
First partial product	0	10111		
shift right EAQ	0	01011	11001	4
$Q_n = 1$; add B		+ 10111		
Second partial product	1	00010		
shift right EAQ	0	10001	01100	3
$Q_n = 0$; shift right EAQ	0	01000	10110	2
$Q_n = 0$; shift right EAQ	0	00100	01011	1
$Q_n = 1$: add B		+ 10111		
Fifth partial product	0	11011		
shift right EAQ	0	01101	10101	0

Final product in AQ = 01101 10101

Booth Multiplication algorithm :-

→ Booth algorithm gives a procedure for multi-plying binary integers in signed 2's complement representation.

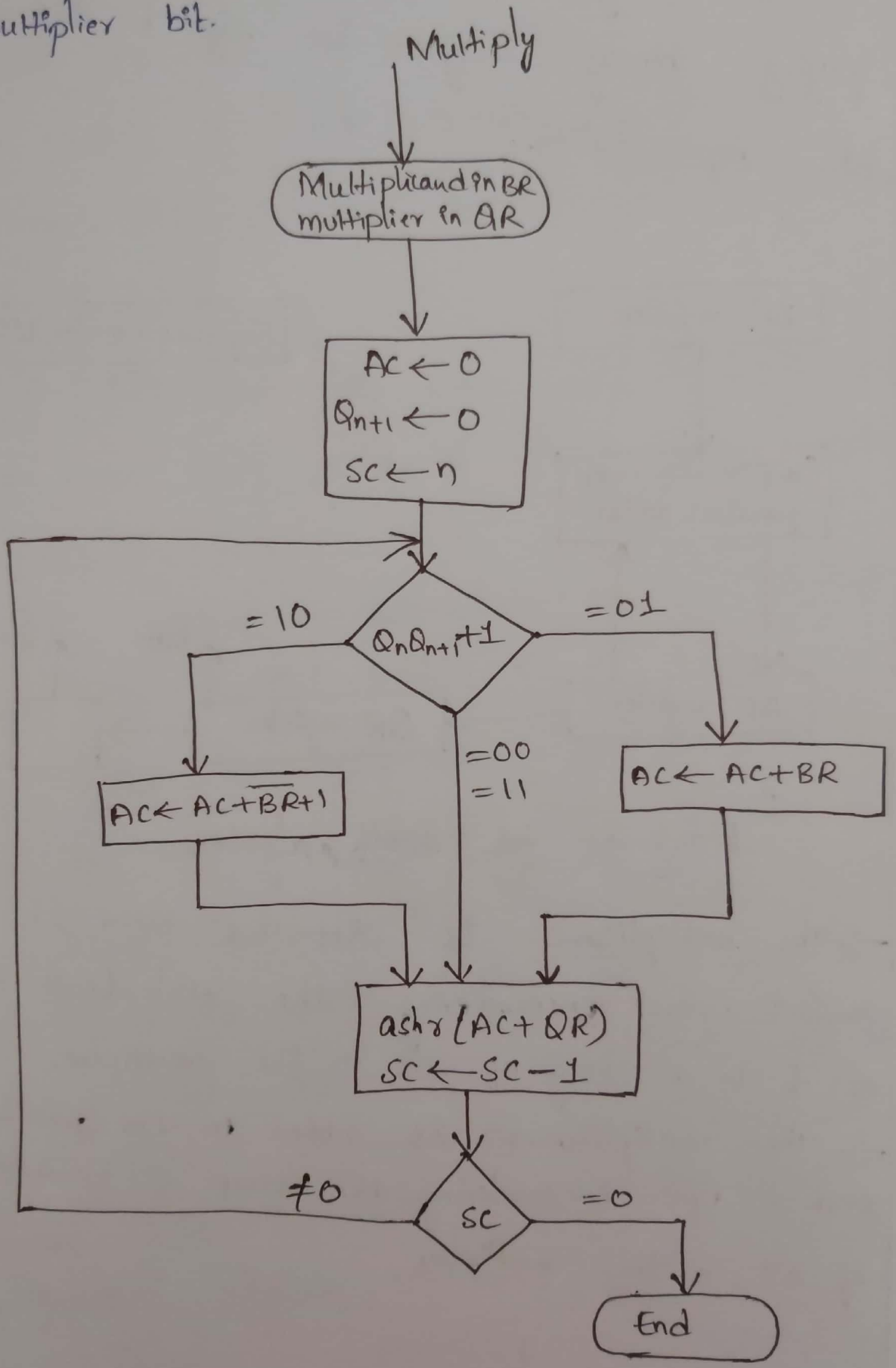


Hardware for Booth algorithm

→ The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier.

→ The multiplicand is added to the partial product upon encountering the first 0 in a string of 0's in the multiplier.

→ The partial product does not change when the multiplier bit is identical to the previous multiplier bit.



→ AC and the appended bit Q_{n+1} are initially cleared to 0 and the sequence counter SC is set to a number n equal to the number of bits in the multiplier.

→ Two bits of the multiplier in Q_n and Q_{n+1} are inspected.

→ If the two bits are equal to 10, it means that the first 1 in a string of 1's has been encountered.

→ This requires a subtraction of the multiplicand from the partial product in AC.

→ If the two bits are equal to 01, it means that the first 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the partial product in AC.

→ When two bits are equal, the partial product doesnot change. An overflow cannot occur because the addition and subtraction of the multiplicand follow each other.

→ The next step is to shift right the partial product and the multiplier (Qn+1 bit)
→ This is an arithmetic right shift (ashr) operation shifts AC and QR to the right. and leaves the sign bit in AC unchanged.
→ The sequence counter is decremented and the computational loop is repeated n times.

Example of Multiplication with Booth Algorithm

BR = 10111

$\overline{BR}+1$ = 01001

Qn	Qn+1		AC	QR	Qn+1	SC
		Initial	00000	10011	0	101
1	0	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	Subtract BR (->)	01001			
			00111			
		ashr	00011	10101	1	000

→ A numerical example of Booth algorithm is shown in above table.

→ for n = 5.

→ It shows the step-by-step multiplication of (-9) × (-13) = +117.

→ -9 will be stored in BR.

-9
↓
step 1 :- find 9 binary value

step 2 :- do 2's complement of 9

	16	8	4	2	1
9 -	0	1	0	0	1
↓					
1's comple :	1	0	1	1	0
add +1 :-					+ 1
-9 =	1	0	1	1	1

-9 = 10111 (BR)

→ In the same way find out $\overline{BR}+1$

BR = 10111
↓
$\overline{BR}$ = 01000
+ 1
$\overline{BR}+1$ = 01001

$\overline{BR}+1$ = 01001.

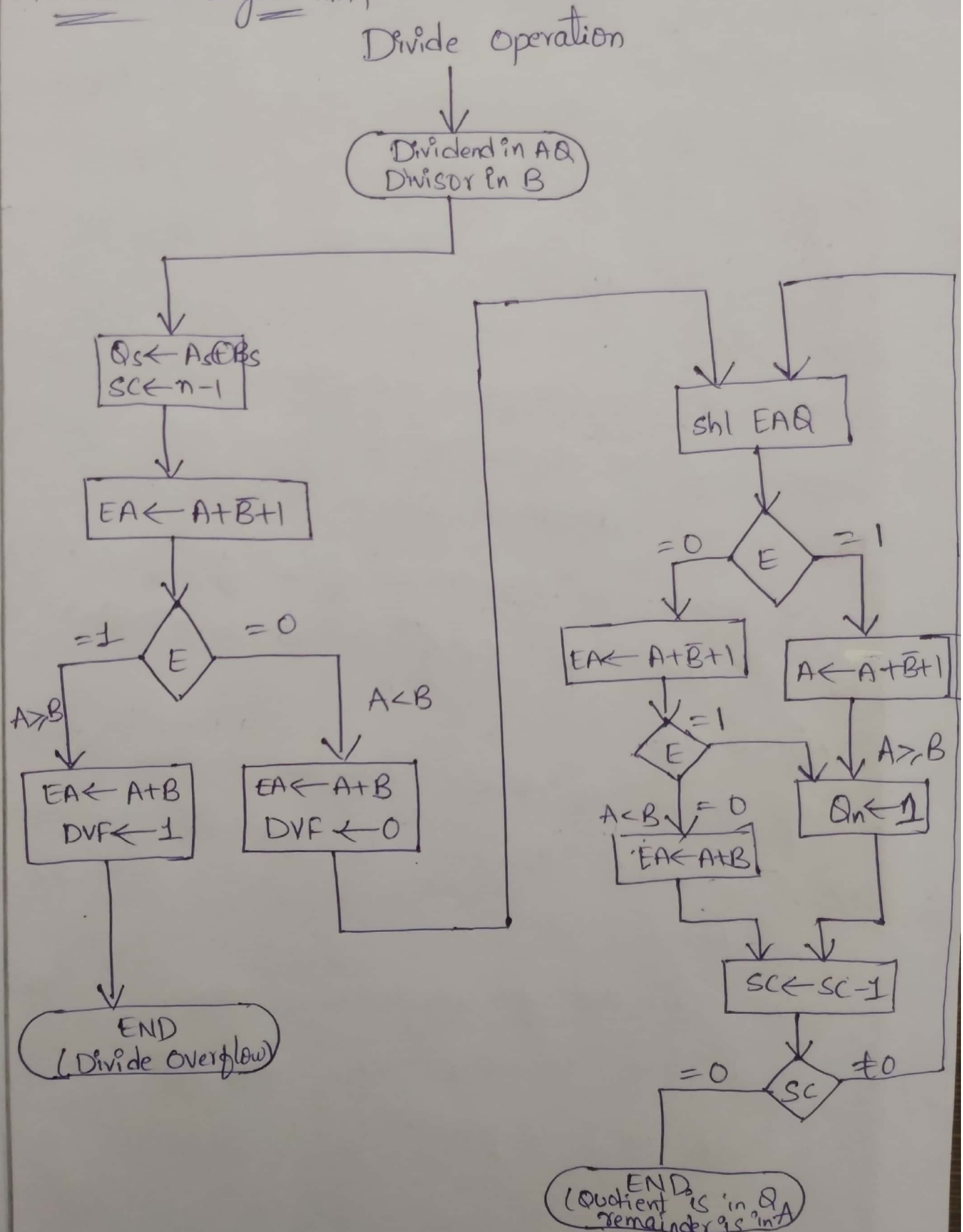
→ Now -13
step 1 :- find 13 value in binary

	16	8	4	2	1
13 =	0	1	1	0	1
do 2's	1	0	0	1	0
complement				+	1

⇒ -13 = 10011

Division algorithm for signed-Magnitude data

Hardware algorithm



→ The hardware divide algorithm is shown in flowchart. The dividend is in A and Q and the divisor in B.

→ The sign of the result transferred into Qs.

→ A constant is set into the sequence counter SC to specify the number of bits in the quotient.

→ A divide-overflow condition is tested by subtracting the divisor in B from half of the bits of the dividend stored in A.

→ If A ≥ B, the devide overflow flipflop DVF is set and the operation is terminated prematurely.

→ If A < B, no divide overflow occurs so the value of the dividend is restored by adding B to A.

→ The division of magnitudes starts by shifting the dividend 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 because 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 Q_n for the quotient bit.

→ If the shift-left operation inserts a 0 into E, the divisor is subtracted by adding 2's complement value and the carry is transferred into E.

→ If E = 1, it signifies that A ≥ B; therefore Q_n is set to 1.

→ If E = 0, it signifies that A < B and the original number is restored by adding B to A.

→ In latter case, we leave a 0 in Q_n.

→ This process is repeated again with register A holding the partial remainder.

→ This division algorithm is also known as "Restoring algorithm". (or) "restoring method". The reason for this name is that the partial remainder is restored by adding the divisor to the negative difference.

Non-restoring method:- In the non-restoring method, B is not added if the difference is negative but instead, the negative difference is shifted left and then B added.

Example of binary division:-

```
                    .11010
Divisor          √.0111000000      Quotient := Q
B = 10001          01110      Dividend in A
                 . 011100     5 bits of A<B, quo has 5 bits
                 - 10001      6 bits of A ≥ B. ; shr B
                              and subtract ; enter 1 in Q
               = 010110 -> 7 bits of remainder ≥ B in Q
                  10001 -> shr B, and subtract ; 1 in Q
                . 001010 -> Rem < B; enter 0 in Q; shr B
                  010100 -> Rem ≥ B
                   10001 -> shr B and sub ; enter 1 in Q
                  000110 -> rem < B; enter 0 in Q
                   00110
                     ↓
              Final remainder.
```

Example of binary division with digital hardware

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

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

Neglect E

Remainder in A: 00110

quotient in Q: 11010.

→ when the division is implemented in a digital computer, it is convenient to change the process slightly.

→ Register EAQ is now shifted to the left with 0 insested into Qn and the previous value of E lost.

→ The divisor is stored in the B register and the double-length dividend is stored in registers A and Q.

→ The dividend is shifted to the left and the divisor is subtracted by adding its 2's complement value.

→ The information about the relative magnitude is available in E.

→ If E = 1, it signifies that A >, B.

→ A quotient bit 1 is insested into Qn and the partial remainder is shifted to the left to repeat the process.

→ If E = 0, it signifies that A < B so the quotient in Qn remains a 0.

→ The value of B is then added to restore the partial remainder in A to its previous value.

→ The partial remainder is shifted to the left and the process is repeated again until all five quotient bits are formed.

→ The quotient is in Q and the final remainder is in A.

Floating point arithmetic Operations

A floating point number in computer registers consist of two parts: a mantissa m and an exponent e

→ The two parts represent a number obtained from multiplying m times a radix r raised to the value of e; thus

$$m \times r^e$$

→ The mantissa may be a fraction or an integer. For example, assume a fraction representation and a radix 10. The decimal number 537.25 is

$$.53725 \times 10^3$$

represented in a register with $m = 53725$ and $e = 3$ and is interpreted to represent the following floating point number.

Normalization :- A floating point number is said to be normalized if the most significant digit of the mantissa is non zero.

Eg: 350 → normalized.

00035 → not normalized.

(It can be normalized by shifting it 3 positions to the left and discarding the leading zeros.

→ Operations on fp numbers require complex hardware and takes more time

Addition & Subtraction requires :-

→ Alignment of radix point (exponents must be equal)

→ Done by shifting mantissa and adjust exponent accordingly

→ Eg: $0.5372400 * 10^2 + 0.1580000 * 10^{-1}$

either shift the 1st no 3 positions to the left & shift the 2nd by 3 positions to the right.

So,

$.5372400 * 10^2 +$

$.0001580 * 10^2$

$.5373980 * 10^2$

→ Normalized addition will cause an overflow and can be corrected by shifting the sum once to the right and incrementing the exponent.

→ Underflow :- Floating point number has a 0 in the MSB of the mantissa.

→ Shift the mantissa to the left and decrement

→ Multiplication and division doesnot require alignment

→ Result (multiply mantissa and add the exponent for multiplication).

→ Divide the mantissa and subtract the exponent for division.

→ the operation performed with exponent are:

- Compare & increment (align mantissa)
- Add & subtract (multiplication and division)
- Decrement (normalization)

Advantages:-

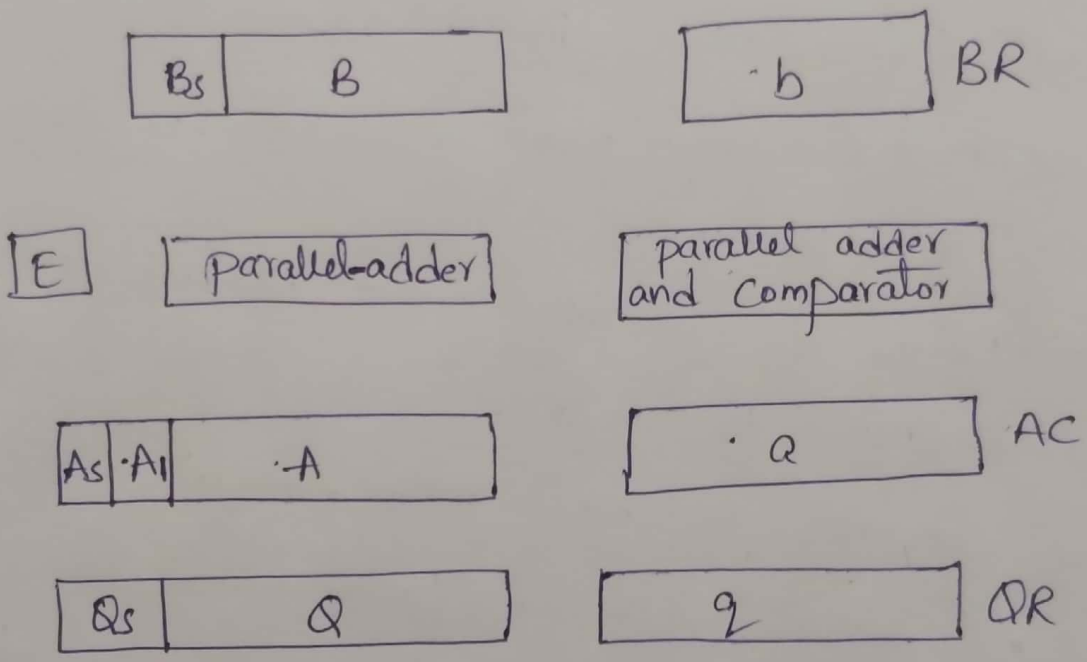
→ They contain only positive numbers (easy to compare)

→ The smallest possible biased exponent contains all zeros.

Register Configuration :-

→ The same registers & adders were in the case of fixed point arithmetic were used for processing mantissa.

→ differs the way in which exponents were handled.



Registers for floating-point arithmetic operations.

→ BR, AC, QR. (divided into 2 parts).

→ Mantissa is stored in B, A, Q registers

→ Exponent in b, a, q registers.

- Mantissa in signed magnitude in A and sign in As and MSB in A1
- Biased exponent in a
- A1 → 1 if the no. has to be normalized
- Similarly for other registers.

- 2 parallel adders
  - 1, adds two mantissas and sum stores in A, carry to E.
  - 2nd adds the exponents (don't have distinct sign bit, but taken as +ve)
- Exponent overflow is neglected.
- Exponents are connected to comparator that provides 3 binary outputs to indicate their relative magnitude.
- The number in the mantissa is taken as a fraction, so binary point resides to the left of the magnitude part.
- Numbers are normalized both during initial and after the operation.

→ A floating point operation may produce overflow, underflow, significand underflow, and significand overflow.

Addition and Subtraction

- Check for Zeros
- Align the mantissas
- Add & Subtract the mantissas
- Normalize the result.

→ The flowchart for adding & subtracting two floating-point binary numbers is shown in figure.

→ If BR is equal to zero, the operation is terminated with the value in the AC being the result.

→ If AC is equal to zero, we transfer the content of BR into AC and also complement its sign if the numbers were to be subtracted.

→ If neither number is equal to Zero, we proceed to align the mantissas.

→ If the two exponents were equal, ~~the mantissa having the smaller exponent~~ we go to perform the arithmetic operation. If the exponents were not equal, the mantissa having the smaller component is shifted to the right and its exponent incremented.

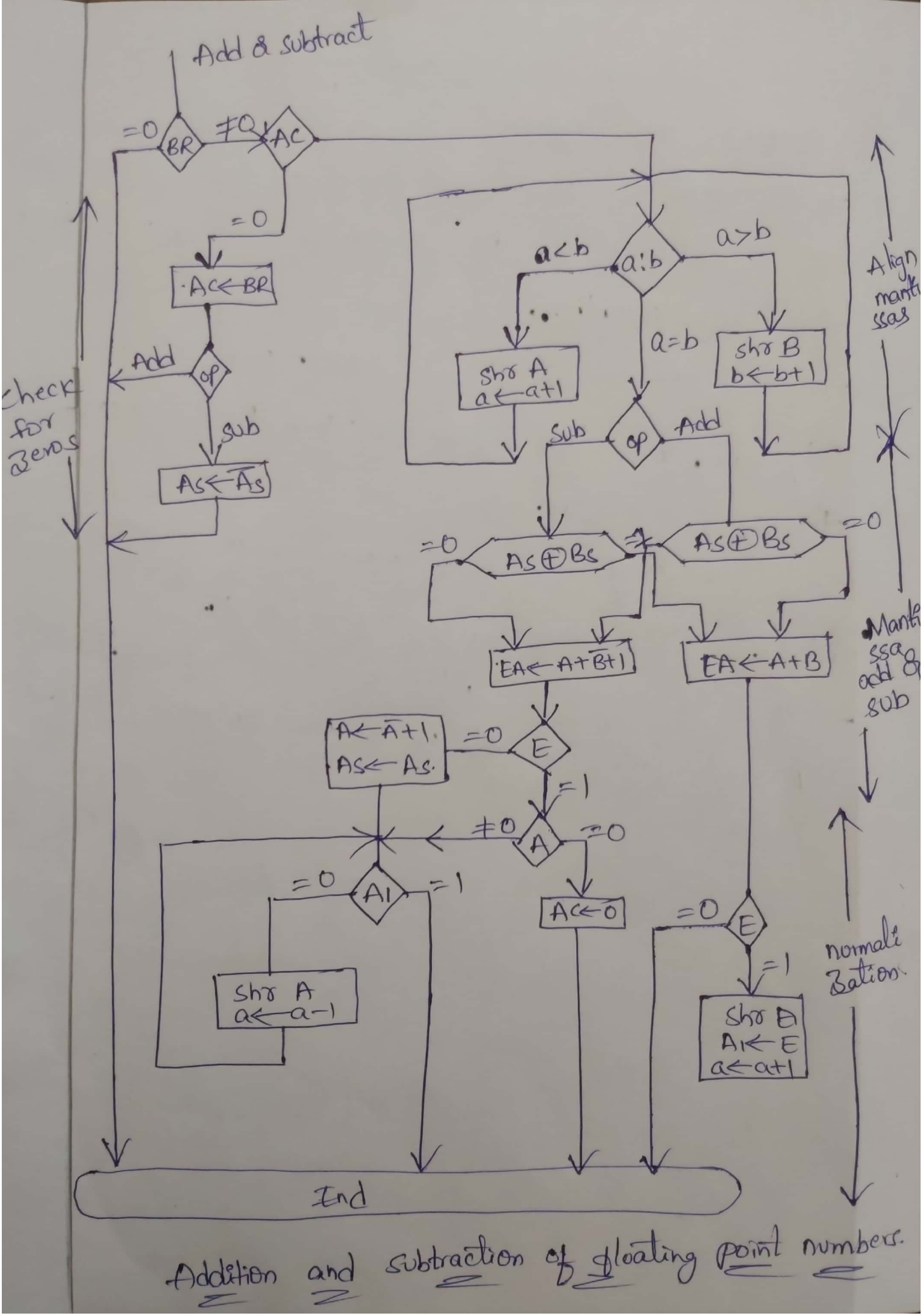
→ The processes is repeated until the two exponents are equal.

→ The addition and subtraction of the two mantissas is identical to the fixed-point addition and subtraction algorithm.

→ The magnitude part is added & subtracted depending on the operation and the signs of the two mantissas.

→ If an overflow occurs when the magnitudes are added, it is transferred into flip-flop E.

→ If E is equal to 1, the bit is transferred into A_1 and all other bits of A are shifted right.



Addition and subtraction of floating point numbers.

Multiplication of floating point numbers

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

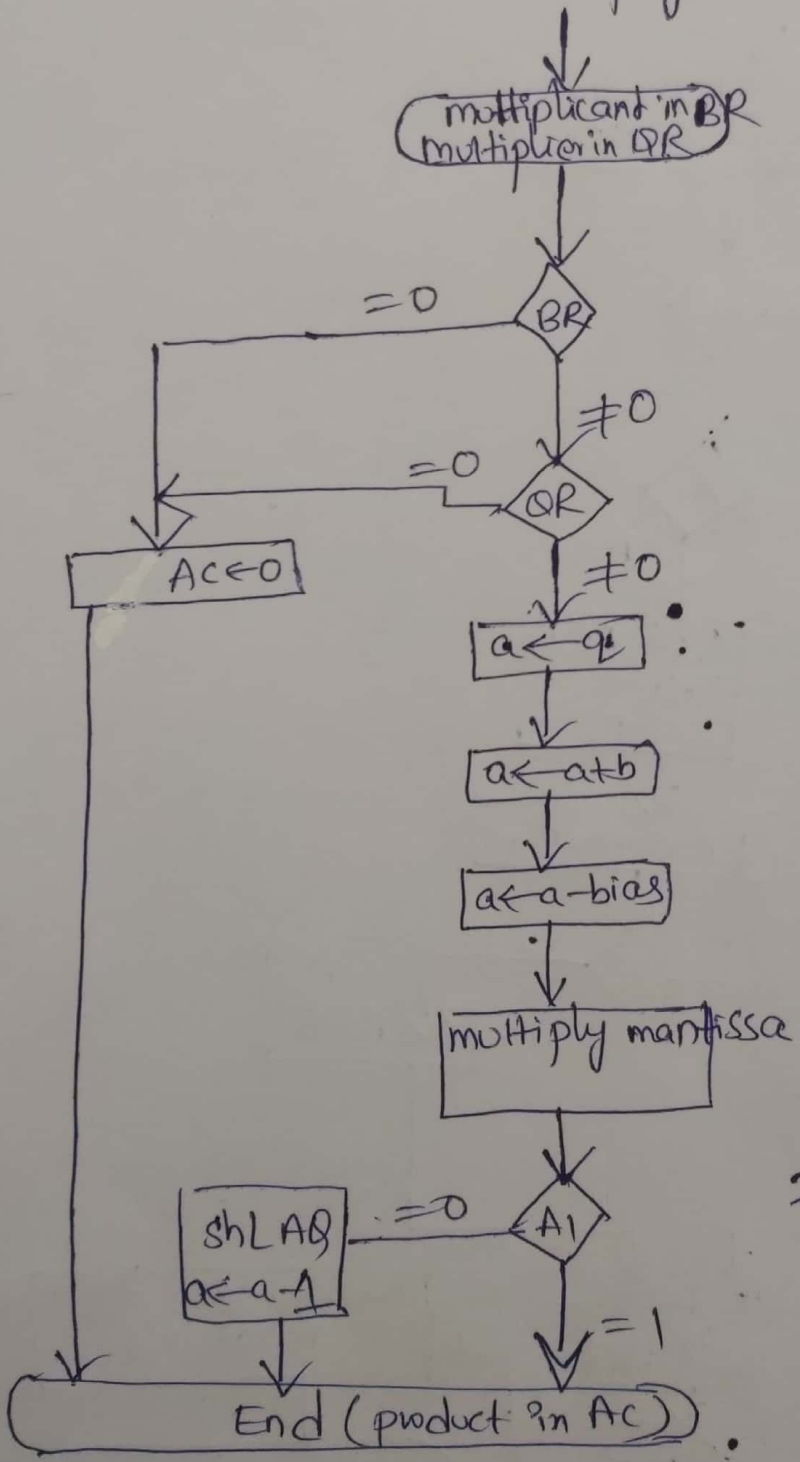
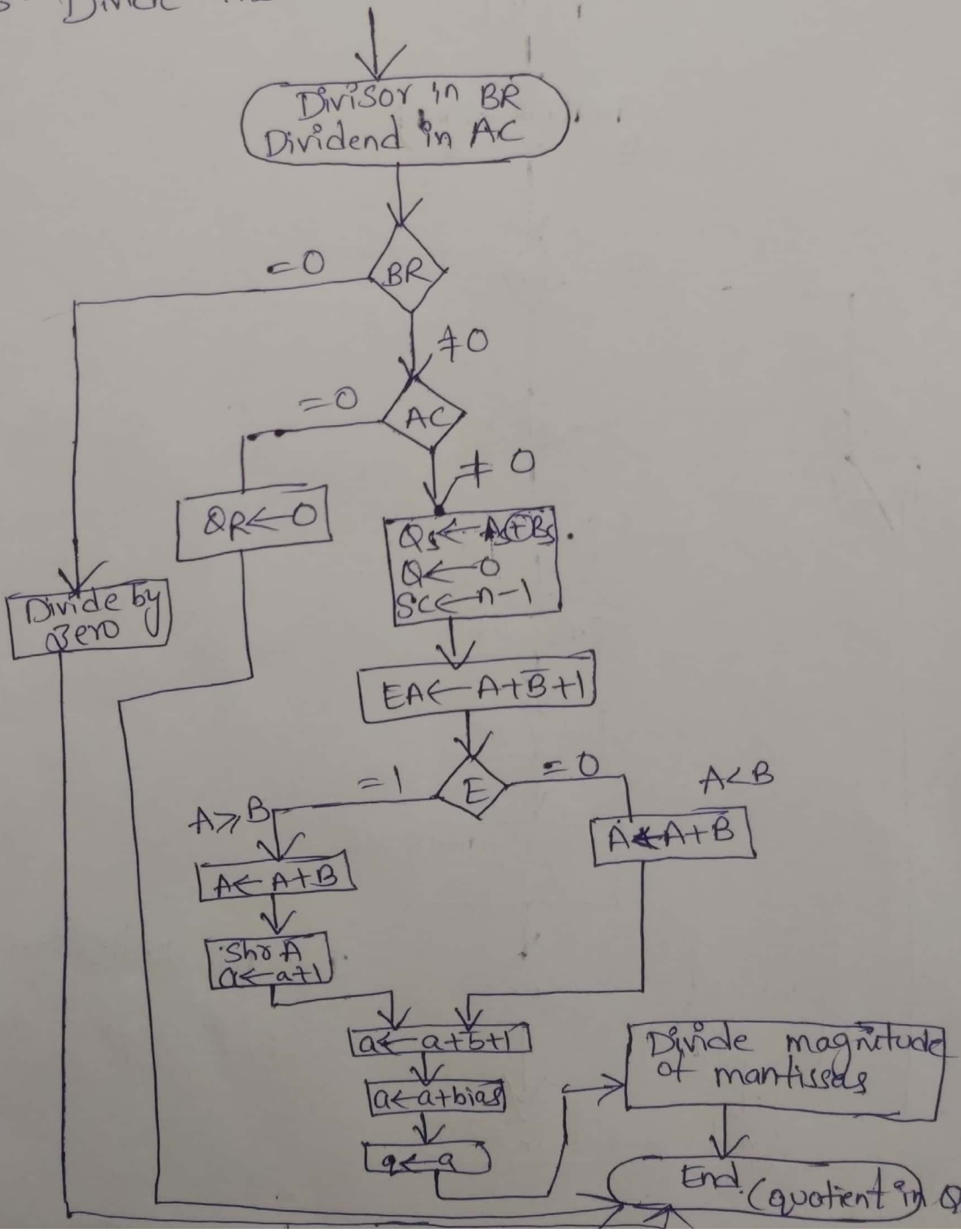


Fig: Multiplication of floating point numbers.

Division of floating point numbers

1. Check for Zeros
2. Initialize registers and evaluate the sign
3. Align the dividend.
4. Subtract the exponents
5. Divide the mantissas.



Decimal Arithmetic Unit :-

→ A decimal arithmetic unit is a digital function that performs decimal microoperations.

→ It can be add & subtract decimal numbers, usually by forming the 9's & 10's complement of the subtrahend.

→ The unit accepts coded decimal numbers and generates results in the same adopted binary code.

→ A single stage decimal arithmetic unit consists of nine binary input variables and five binary output variables, since a minimum of four bits is required to represent each coded decimal digit.

BCD adder :-

→ Consider the arithmetic addition of two decimal digits in BCD, together with a possible carry from a previous stage.

→ Since each input digit does not exceed 9, the output sum cannot be greater than 9+9+1 = 19.

→ The 1 in the sum being an input carry.

→ Suppose that we apply two BCD digits to a 4-bit binary adder. The adder will form the sum in binary and produce a result that may range from 0-19.

Derivation of BCD adder

	Binary sum				BCD sum					
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	Decimal
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

→ The above table represents the binary numbers and are labeled by symbols K, Z_8, Z_4, Z_2 and Z_1.

→ K is the carry and the subscripts under the letter z represents the weights. 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code

→ The first column in the table lists the binary sums as they appear in the outputs of a 4-bit binary adder.

→ The output sum of two decimal numbers must be represented in BCD and should appear in the form listed in the second column of the table.

→ When the binary sum is equal to or less than 1001, the corresponding BCD number is identical and therefore no conversion is needed.

→ When the binary sum is greater than 1001, we obtain a nonvalid BCD representation.

→ The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry.

→ It is obvious that a correction is needed when the binary sum has an output carry K=1

→ The other six combinations from 1010 to 1111 that need a correction have a 1 in position Z_8, we specify further that either Z_4 & Z_2 must have a 1.

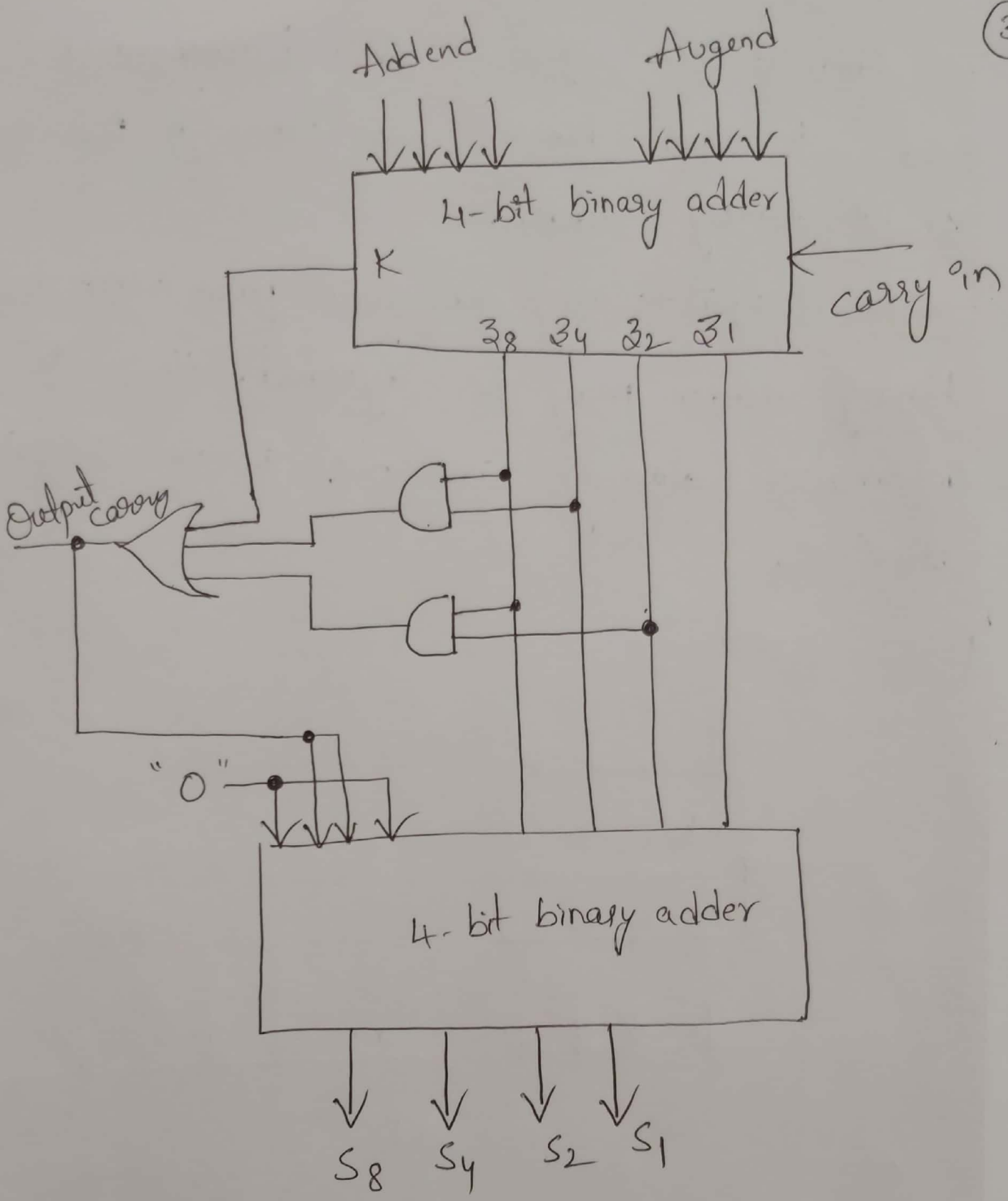
→ The condition for a correction and an output-carry can be expressed by the boolean function.

$$C = K + Z_8 Z_4 + Z_8 Z_2$$

→ When C=1, it is necessary to add 0110 to the binary sum and provide an output-carry for the next stage.

BCD Addition :- A BCD adder is a circuit that adds two BCD digits in parallel and produces a sum digit also in BCD.

→ A BCD adder must include the correction logic in its internal construction.



Block diagram of BCD adder

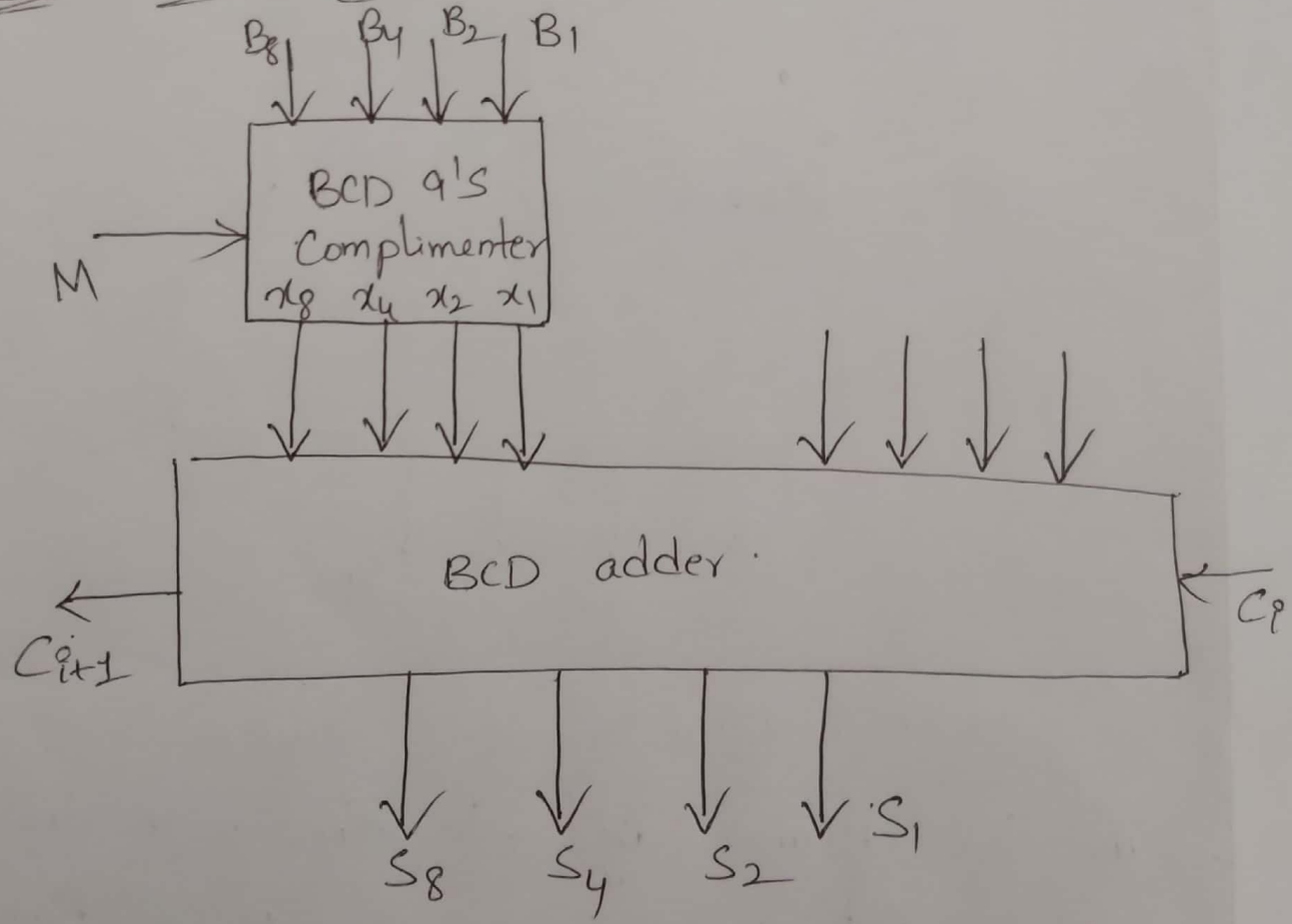
→ The two decimal digits, together with the Output carry, are first added in the top 4-bit binary adder to produce the binary sum.

→ When the Output carry is equal to 0, nothing is added to the binary sum.

→ When it is equal to 1, binary 0110 is added to the binary sum through the bottom 4-bit binary adder.

→ The output-carry generated from the bottom binary adder may be ignored, since it supplies information already available at the output-carry terminal.

BCD subtraction



One stage of a decimal arithmetic unit

→ A straight subtraction of two decimal numbers will require a subtractor circuit that will be somewhat different from a BCD adder.

→ It is more economical to perform the subtraction by taking the 9's or 10's complement of the subtrahend and adding it to the minuend.

→ The boolean functions for the 9's complementer circuit are

$$x_1 = B_1M + B_1'M$$

$$x_2 = B_2$$

$$x_4 = B_4M' + (B_4'B_2 + B_4'B_2')M$$

$$x_8 = B_8M' + B_8'B_4B_2'M$$

From these equations we see that $x = B$ when $M = 0$ when $M = 1$ the x outputs produce the 9's complement of B.

→ One stage of a decimal arithmetic unit that can add or subtract two BCD digits.

→ It consist of a BCD adder and a 9's complementer. The mode M controls the operation of the unit.

→ With M=0, the S outputs form the sum of A and B.

→ With M=1, the S outputs form the sum of A plus the 9's complement of B.

→ For numbers with n decimal digits we need n such stages.

→ The output carry C_{i+1} from one stage must be connected to the input carry C_i of the next higher order stage.

→ The best way to subtract the two decimal numbers is to let M=1 and apply a 1 to the input carry C_1 of the first stage.

Decimal arithmetic Operations

→ The algorithms for arithmetic operations with decimal data are similar to the algorithms for the corresponding operations with binary data.

Decimal Arithmetic Microoperation Symbols.

Symbolic designation	Description.
$A \leftarrow A+B$	Add decimal numbers and transfer sum into A.
$\overline{B}$	9's complement of B.
$A \leftarrow A+\overline{B}+1$	Content of A plus 10's complement of B into A
$Q_L \leftarrow Q_L+1$	Increment BCD number in Q_L
dshr A	Decimal shift right register A
dshl A	Decimal shift-left register A.

Decimal arithmetic Operations

→ Addition and subtraction

→ Multiplication

→ Division.

Addition and subtraction :-

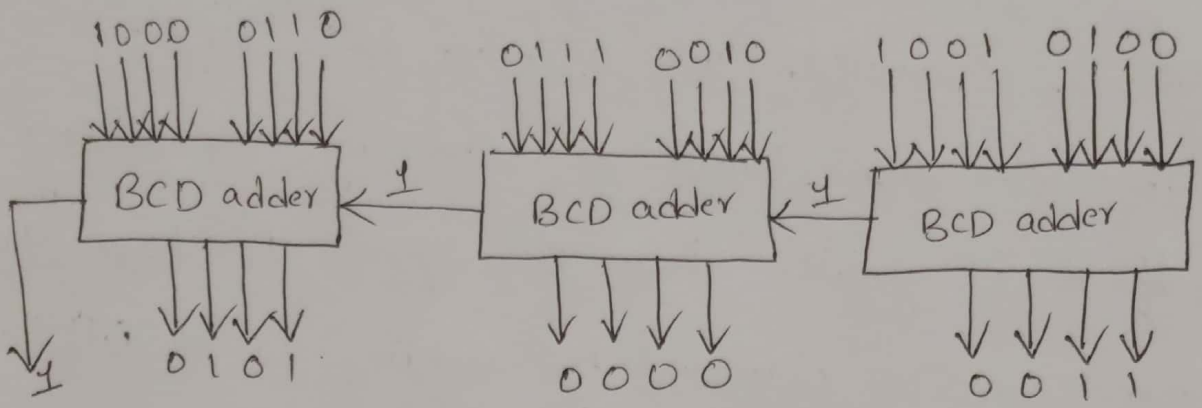
→ The algorithm for addition and subtraction of binary signed-magnitude numbers applies also to decimal signed magnitude numbers.

→ the decimal data can be added in three different ways.

- Parallel method
- Digit-Serial bit-parallel method
- All serial adder method.

→ In parallel method uses a decimal arithmetic unit composed of as many BCD adders as these are digits in the number.

→ the sum is formed in parallel and requires only one micro operation.

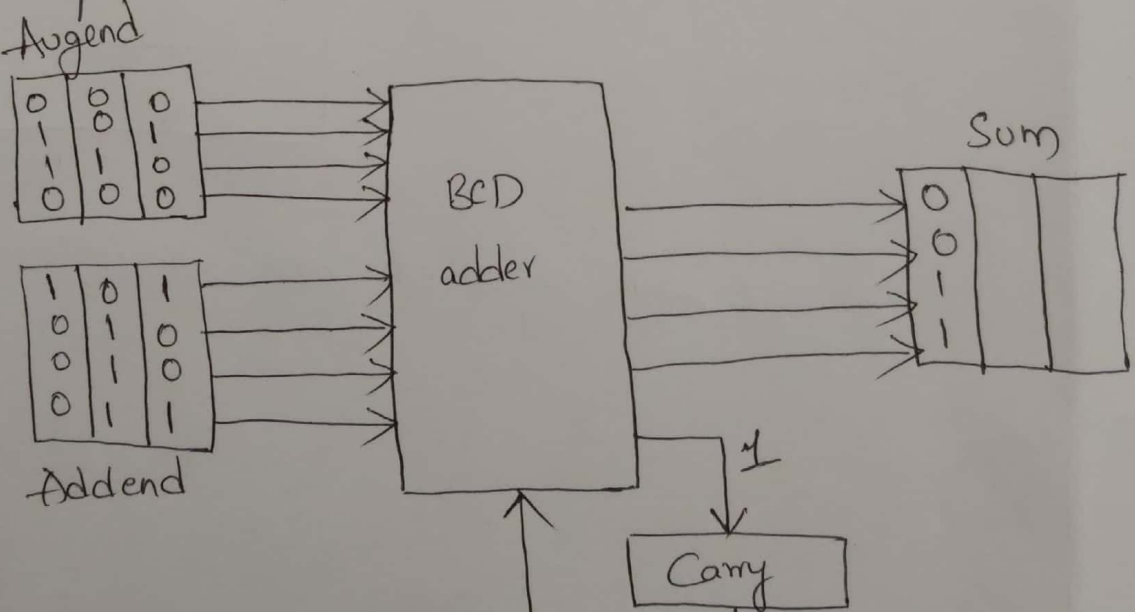


(a) Parallel decimal addition

→ In the digit-serial bit-parallel method, the digits are transferred in parallel.

→ The sum is formed by shifting the decimal numbers through the BCD adder one at a time.

→ for K decimal digits, this configuration requires K · microoperations, one for each decimal shift.

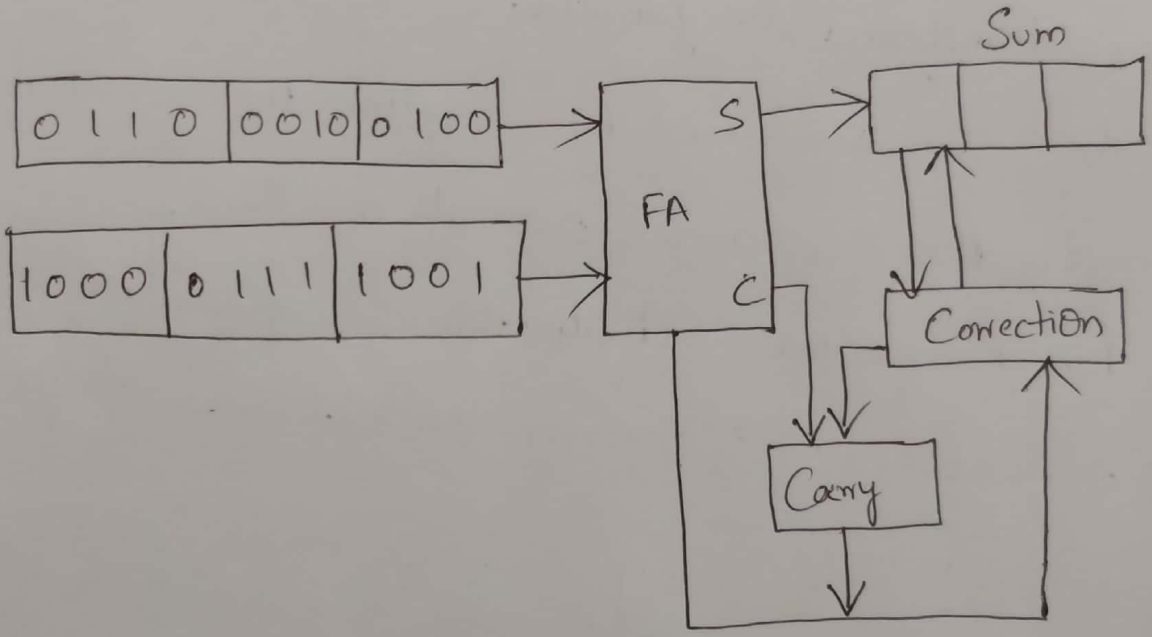


(b) Digit-Serial, bit-parallel decimal addition

→ In the all serial adder, the bits are shifted must be corrected into a valid BCD digit.

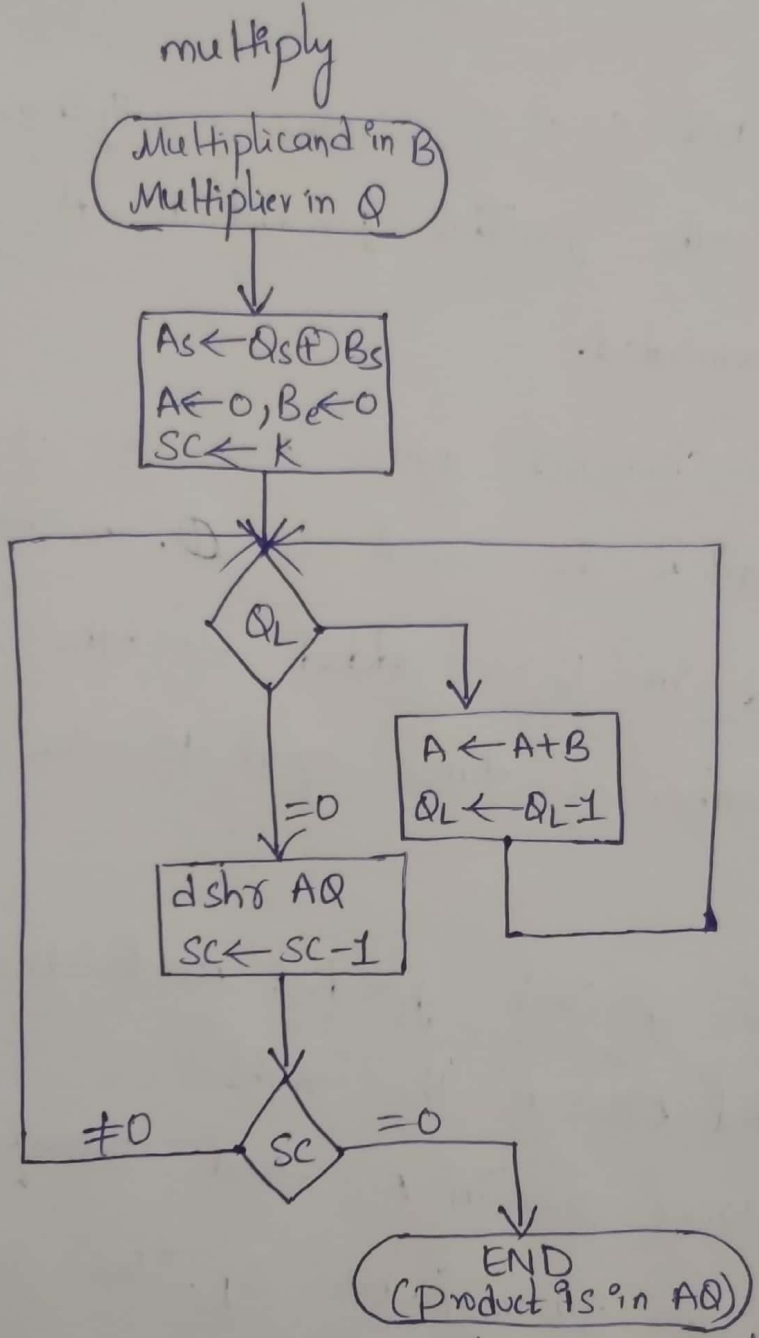
→ This correction consist of checking the binary sum.

→ If it is greater than or equal to 1010, the binary sum is corrected by adding to it 0110. and generating a carry for the next pair of digits.



(C) All serial decimal addition

Multiplication :-



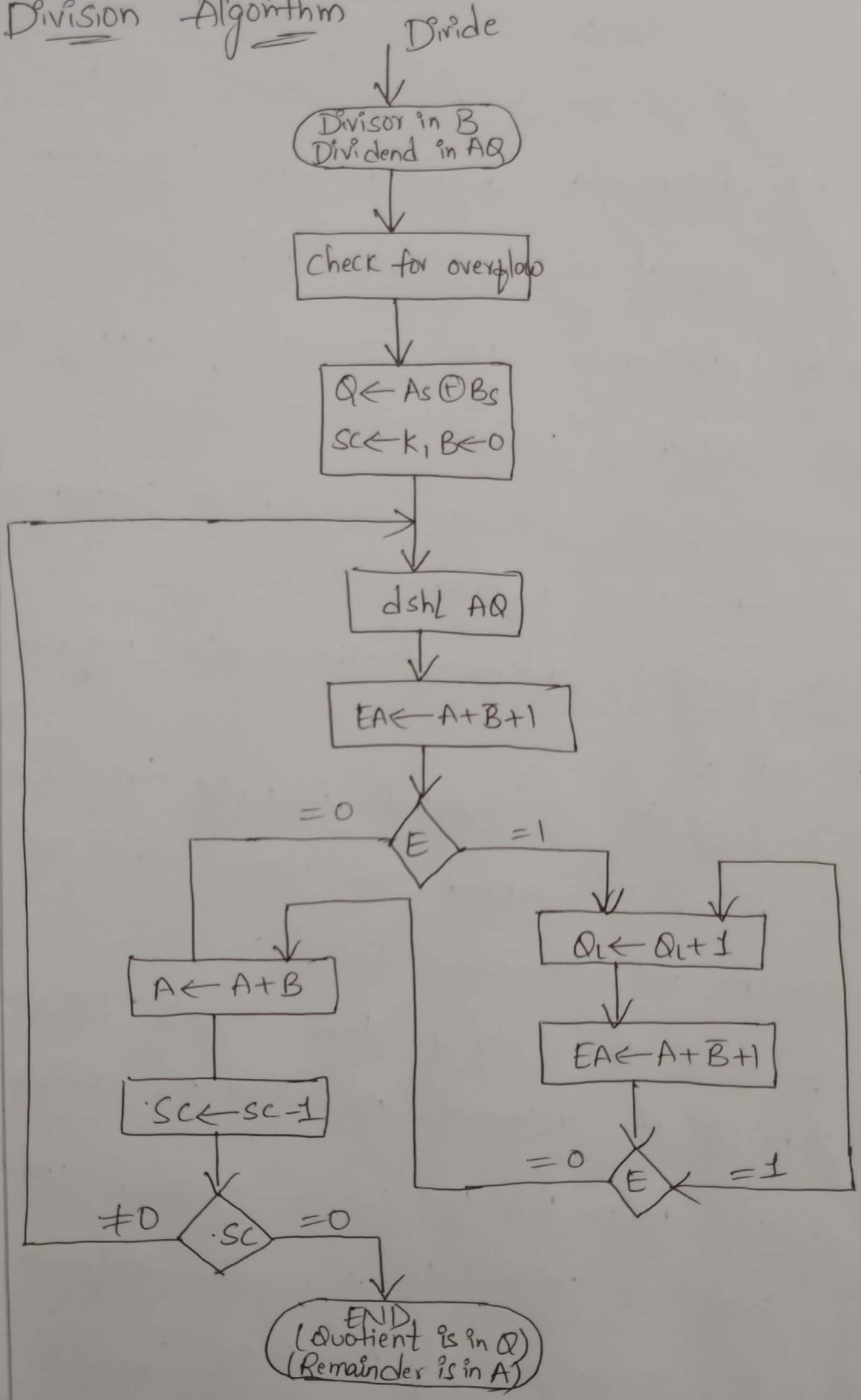
Flow chart for decimal multiplication

→ Multiplication of fixed point decimal numbers is similar to binary except for the way partial products formed.

→ Initially the entire A register and Be are cleared and the sequence counter SC is set to a number K equal to the number of digits in the multiplier.

→ The low-order digit of the multiplier in Q_L is checked.
→ If it is not equal to 0, the multiplicand in B is added to the partial product in A once and Q_L is decremented.
→ Q_L is checked again and the process is repeated until it is equal to 0. In this way the multiplicand in B is added to the partial product number of times equal to the multiplier digit.
→ Next, the partial product and the multiplier are shifted once to the right.
→ This places zero in Ae and transfers the next multiplier quotient into Q_L. The process is then repeated K times to form a double-length product in AQ.

Division Algorithm



Flowchart for decimal division

→ It is similar to the algorithm with binary data except for the way the quotient bits are formed

→ The dividend is shifted to the left, with its most significant digit placed in Ae.

→ The divisor is then subtracted by adding its 10's complement value.

→ Since Be is initially cleared, its complement value is 9 as required.

→ The Carry E determines the relative magnitude of A and B.

→ If E = 0, it signifies that A < B, In this case the divisor is added to restore the partial remainder and Q_L stays at 0.

→ If E = 1 it signifies that A ≥ B. The quotient digit in Q_L is incremented once and the divisor subtracted again. This process is repeated until the subtraction results in a negative difference which is recognized by E being 0.

→ The partial remainder and the quotient bits are shifted once to the left and the process is repeated K times to form K quotient digits.

→ The remainder is then found in register A and the quotient is in register Q. The value of E is neglected.

* unit - III completed *