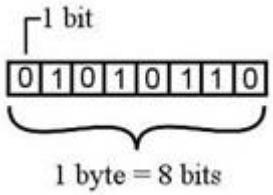
COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- DATA REPRESENTATION-BASICS, DATA TYPES

Basics

Bit: The most basic unit of information in a digital computer is called a bit.
A bit is the smallest data unit that a computer uses in computation.
All the computation tasks done by the computer systems are based on bits.A bit represents a binary digit in terms of 0 or 1. The computer usually uses bits in groups. It's the basic unit of information storage and communication in digital computing.

Byte: In 1964, the designers of the IBM System/360 main frame computer established a convention of using groups of 8 bits as the basic unit of addressable computer storage. They called this collection of 8 bits a byte. Half of a byte(4-bits) is called a nibble.

A byte is a fundamental addressable unit of computer memory and storage. Bytes are used to determine file sizes, storage capacity, and available memory space



Byte Value	Bit Value
1 Byte	8 Bits
1024 Bytes	1 Kilobyte
1024 Kilobytes	1 Megabyte
1024 Megabytes	1 Gigabyte
1024 Gigabytes	1 Terabyte
1024 Terabytes	1 Petabyte
1024 Petabytes	1 Exabyte
1024 Exabytes	1 Zettabyte
1024 Zettabytes	1 Yottabyte
1024 Yottabytes	1 Brontobyte
1024 Brontobytes	1 Geopbytes

Conversion of Bits and Bytes

• **Word:** Computer words consist of two or more adjacent bytes that are sometimes addressed and almost always are manipulated collectively.
• The word size represents the data size that is handled most efficiently by a particular architecture. Words can be 16 bits, 32 bits, 64 bits.

Data types

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. and

(3) Other discrete symbols used for specific purposes.

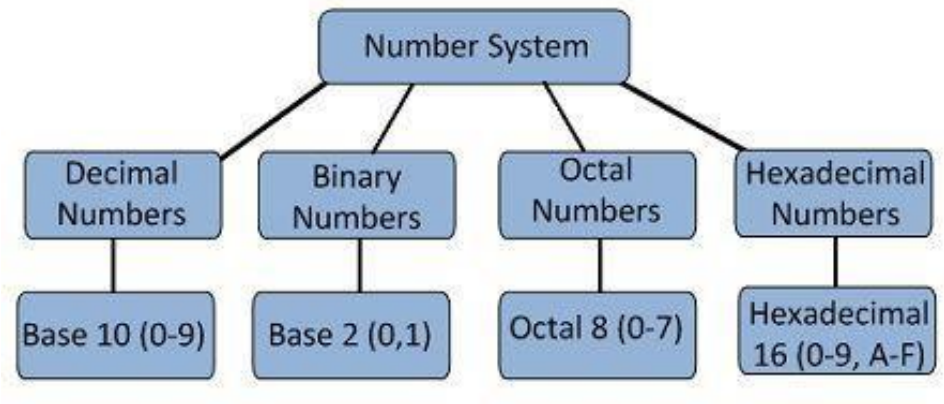
All types of data, except binary numbers, are represented in computer registers in binary coded form. This is because registers are made up of flip-flops and flip-flops are two-state devises that can store only l's and 0's. The binary number system is the most natural system to use in a digital computer. But sometimes it is convenient to employ different number systems, especially the decimal number system, since it is used by people to perform arithmetic computations.

Number System

Number systems are the technique to represent numbers in the computer system architecture.

Computer architecture supports following number systems.

- **Binary number system**
- **Octal number system**
- **Decimal number system**
- **Hexadecimal(hex) number system**



1.Binary Number System

A Binary number system has only two digits **0 and 1**. Every number (value) represents with 0 and 1. The base of binary number system is 2, because it has only two digits.

2.Octal number system

Octal number system has only eight (8) digits from **0 to 7**. Every number (value)represents with0,1,2,3,4,5,6 and 7.The base of octal number system is 8, because it has only 8 digits.

3. Decimal number system

Decimal number system has only ten (10) digits from **0 to 9**. Every number (value) represents with 0,1,2,3,4,5,6,7,8 and 9.The base of decimal number system is 10, because it has only 10 digits.

4.Hexa decimal number system

A Hexadecimal number system has sixteen (16) alphanumeric values from**0 to 9** and **A to F**. Every number (value) represents with 0,1,2,3,4,5,6, 7,8 ,9,A,B,C,D,E and F. The base of hexadecimal number system is 16, because it has 16 alphanumeric values. Here **A is 10**, **B is11**, **C is 12**, **D is 13**, **E is 14** and **F is 15**.

Table of the Numbers Systems

Number system	Base	Used digits	Example
Binary	2	0,1	$(11110000)_2$
Octal	8	0,1,2,3,4,5,6,7	$(360)_8$
Decimal	10	0,1,2,3,4,5,6,7,8,9	$(240)_{10}$
Hexadecimal	16	0,1,2,3,4,5,6,7,8, 9, A, B, C, D, E, F	$(F0)_{16}$

Number Conversions

To convert Number system from **Decimal Number System** to **Any Other Base** is done with just two steps:

a) Divide the Number (Decimal Number) by the base of target base system in which you want to convert the number: Binary (2), octal (8) and Hexadecimal (16)).

b) Write the remainder from step 1 as a Least Signification Bit (LSB) to Step last as a Most Significant Bit (MSB).

Decimal to Binary conversion

Steps:

1. Divide the decimal number by 2 and store remainders in array.
2.Divide the quotient by 2.
3.Repeat step 2 until we get the quotient equal to zero.

4. Equivalent binary number would be reverse of all remainders of step 1

For example:

a.$(149)_{10}$

2	**149**	1
2	**74**	0
2	**37**	1
2	**18**	0
2	**9**	1
2	**4**	0
2	**2**	0
2	**1**	1
	0	

Therefore,$(149)_{10}=(10010101)_2$

To Convert fractional Part to binary

1. Multiply the fractional decimal number by 2.
2.Integral part of resultant decimal number will be first digit of fraction binary number.
3.Repeat step 1 using only fractional part of decimal number and then step 2.

- Repeat this procedure unless and until the fractional part happens to be 0.
- If the fractional part doesn't terminate to 0, then one needs to find the result of the fraction up to as many places as required.

Example:

a.Here,$(0.55)_{10}$

0.55x2= 1.1 1

0.1x2=0.2 0

0.2x2=0.4 0

0.4x2=0.8 0

0.8x2=1.6 1

0.6x2=1.2 1

Therefore,$(0.55)_{10}=(0.100011)_2$

Decimal to Octal Conversion

Result

Decimal Number is:$(12345)_{10}$

8	12345	1	LSB
8	1543	7	
8	192	0	
8	24	0	
	3	3	MSB

Octal Number is $(30071)_8$

Decimal to Hexa decimal Conversion

Example: Convert 1228_{10} into hex.

Divide by 16	Quotient	Remainder	Hex Value
1228 ÷ 16	76	12	C
76 ÷ 16	4	12	C
4 ÷ 16	0	4	4

Therefore, $1228_{10} = 4CC_{16}$

Binary to Decimal Conversion

- Multiply each bit by 2^n,where n is the "weight" of the bit
- The weight is the position of the bit, starting from 0 on the right
- Add the results.

Example1:(101011)2=(?)10=(43)10

$(101011)_2=(1x2^5)+(0x2^4)+(1x2^3)+(0x2^2)+(1x2^1)+(1x2^0)$

$=32+0+8+0+2+1$

$=(43)_{10}$

Example2:Convert(111.101)2

$(111.101)_2=(1x2^2)+(1x2^1)+(1x2^0)+(1x2^{-1})+(0x2^{-2})+(1x2^{-3})$

$=4+2+1+0.5+0+ 0.125$

$=(7.625)_{10}$

Binary to Octal conversion

Example 1: Convert 1010101_2 to octal

Solution:

Given binary number is 1010101_2

First, we convert given binary to decimal

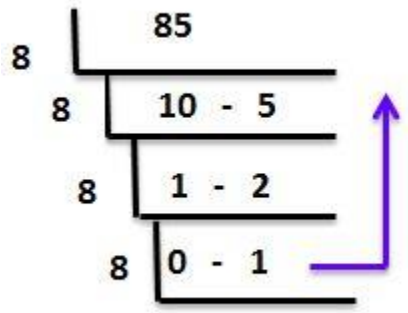
$1010101_2 = (1 * 2^6) + (0 * 2^5) + (1 * 2^4) + (0 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2^0)$

$= 64 + 0 + 16 + 0 + 4 + 0 + 1$

$= 64 + 21$

$010101_2 = 85$ (Decimal form)

Now we will convert this decimal to octal form



Therefore, the equivalent octal number is 125_8.

Binary to Hex Converter

To convert binary number to hexadecimal is an easy method. We have to group the given binary number in pair of 4 and then find the equivalent hexadecimal number from the below table.

Binary	Hex
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6

0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Example: Convert 1001001_2 into a hexadecimal number.

Solution: 1001001_2

= 0100 1001

= 49_{16}

COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT –III

TOPIC- DATA REPRESENTATION-BINARY ARITHMETIC

Binary Arithmetic

Binary arithmetic is an essential part of various digital systems. You can add, subtract, multiply, and divide binary numbers

Binary Addition

Adding two binary numbers will give us a binary number itself. It is the simplest method.

Binary Numbers		Addition
0	0	0
0	1	1
1	0	1
1	1	0;Carry→1

Example:

(111101)2+(10111)2=(1010100)2

For example: Add1101_2 and 1001_2.

```
  1101
+1001
------
 10110
------
```

Binary subtraction

Binary subtraction is one of the four binary operations, where we perform the subtraction method for two binary numbers (comprising only two digits, 0 and 1). This operation is similar to the basic arithmetic subtraction performed on decimal numbers in Maths. Hence, when we subtract 1 from 0, we need to borrow 1 from the next higher order digit, to reduce the digit by 1 and the remainder left here is also 1

When you subtract several columns of binary digits, you must take into account the borrowing. When 1 is to be subtracted from 0, the result is 1 where 1 is borrowed from the next highest order bit or digit.

Binary Number	Subtraction Value
0 – 0	0
1 – 0	1
0 – 1	1 (Borrow 1 from the next high order digit)
1 – 1	0

Note:

The addition of two binary numbers 1 and 1 is 10, where we consider 0 and carry forward 1 to the next high order. But in the case of subtraction of 1 and 1, the answer is equal to 0, and nothing is carried forward.

In the case of decimal subtraction, when 1 is subtracted from 0, then we borrow 1 from the next preceding number and make it 10, and after subtraction, it results in 9, i.e. 10 – 1 = 9. But for binary subtraction, it results in 1 only.

Example1:. Subtract 1 10 from 1 1 1 0

Step 1: You subtract the numbers in the one's column and not down the result. In this case, the value of 0 - 1 = 0. We borrow 1 from the number in the ten's place and continue with the subtraction.

```
1 1 1 0
  1 1 0 (-)
_________
      0
_________
```

Step 2: Now you subtract the values in the 10's place. Apply the aforementioned binary subtraction rules.

1 1 1 0
1 1 0 (-)
_____
0 0
_____

Step 3: Subtract the value that is present in the hundreds place value.

1 1 1 0
1 1 0 (-)
_____
0 0 0
_____

Step 4: Since we don't have anything in the thousand's place, we retain it as it is.

1 1 1 0
1 1 0 (-)
_____
1 0 0 0
_____

Example 2: 1 0 1 1 0 1 0 - 0 0 1 0 1 0

1 0 1 1 0 1 0
0 0 1 0 1 0 (-)
_____
1 0 1 0 0 0 0
_____

Binary Multiplication

- Binary multiplication is similar to decimal multiplication.
- However, as there are only bits,0 and 1,It is much simpler than decimal multiplication because there are only two possible results of multiplying two bits.

There are four rules of binary multiplication.

1. **0x0=0**
2. **0x1=0**
3. **1x0=0**
4. **1x1=1**

Example:(10111)2x(1010)2=(0110110)2

			1	0	1	1	1
x				1	0	1	0
			0	0	0	0	0
		1	0	1	1	1	
	0	0	0	0	0		
1	0	1	1	1			
1	1	1	0	0	1	1	0

Binary Division

The division is probably one of the most challenging operations of the basic arithmetic operations. There are different ways to solve division problems using binary operations. Long division is one of them and the easiest and the most efficient way.

The main rules of the binary division include:

- 1÷1 = 1
- 1÷0 = Meaningless
- 0÷1 = 0
- 0÷0 = Meaningless

Similar to the decimal number system, the binary division is similar, which follows the four-step process:

- Divide
- Multiply
- Subtract
- Bring down

Example 1.

Question: Solve 01111100 ÷ 0010

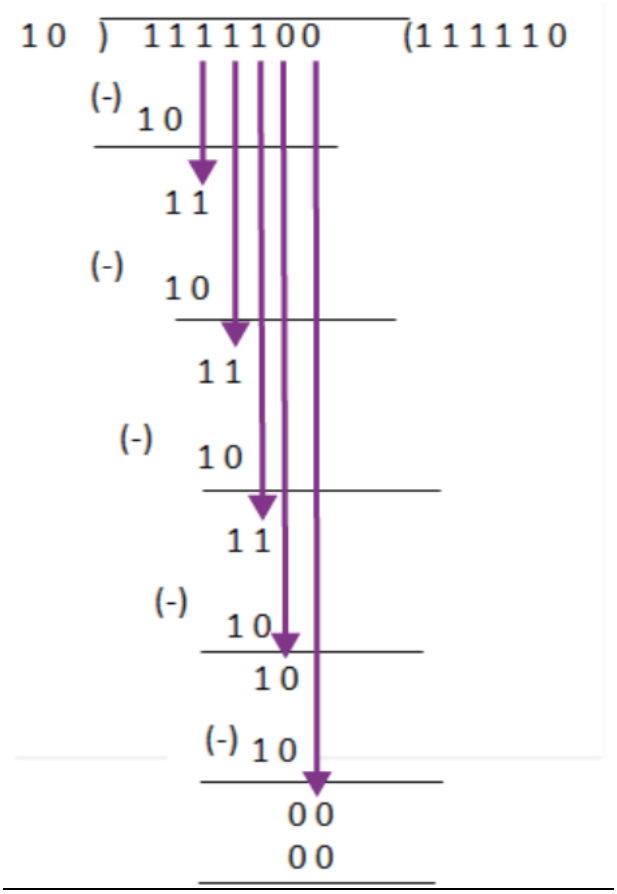
Solution: Given 01111100 ÷ 0010

Here the dividend is 01111100, and the divisor is 0010

Remove the zero's in the **Most Significant Bit** in both the dividend and divisor, that doesn't change the value of the number.

So the dividend becomes 1111100, and the divisor becomes 10.

Now, use the long division method.


10) 1111100 (111110
(-) 10
11
(-) 10
11
(-) 10
11
(-) 10
10
(-) 10
00
00

COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- DATA REPRESENTATION-COMPLEMENTS

Complements

In binary arithmetic, "complements" refer to a method of representing negative numbers by manipulating the bits of their positive binary equivalent, primarily using "1's complement" (flipping all bits) and "2's complement" (flipping all bits and then adding 1), which allows for subtraction to be performed using addition operations within a computer system.

1's Complement

- The 1's complement of a binary number is obtained by change each 0 to 1 and each 1 to 0 or subtract each bit from Binary "1".

Example for 7-digit binary numbers:

- 1's complement is $(r^n–1)–N=(2^7–1)–N= 1111111– N$.

Example1:

Find the1's complement of $(1011000)_2$

$$\begin{array}{r} 1111111 \\ 1011000 \\ \hline 0100111 \end{array} -$$

1's complement of 1011000 is 0100111

2's Complement

- Obtain1's complement of given number and then add 1 to the least significant bit (L.S.B)
- Toggle all bits and add '1'fromtheright

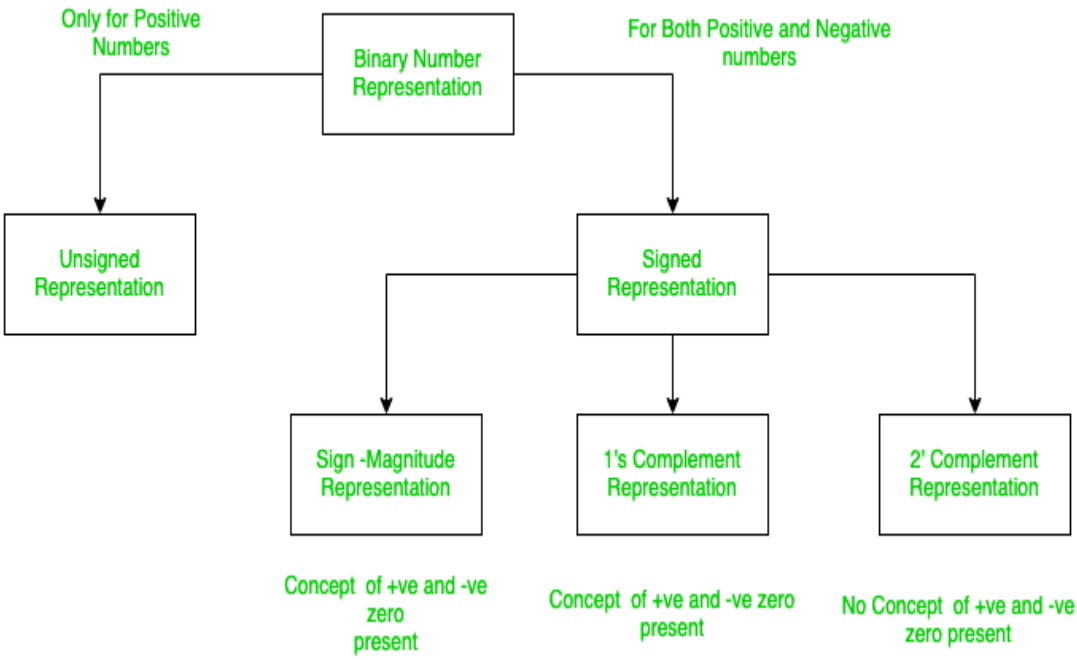
Example1:

The 2's complement of 10110000 is 01010000

Number:1 0 1 1 0 0 0 0
1'sComplement:01001111
2'sComplement:01010000

$$\begin{array}{r} 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0 \\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1 \\ + \qquad\qquad\qquad 1 \\ \hline 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0 \end{array}$$

Fixed-Point Representation



In Binary Number Representation, Positive integers and zero can be represented by unsigned representation. Negative numbers must be represented by signed representation since + and – signs are not available, only 1's and 0's are used.

• Signed numbers have MSB as 0 for positive and 1 for negative – MSB is the sign bit

• Two ways to designate binary point position in a register:

1. Fixed point representation
2. Floating-point representation

• When an integer is positive, the MSB, or sign bit, is 0 and the remaining bits represent the magnitude.

• When an integer is negative, the MSB, or sign bit, is 1, but the rest of the number can be represented in one of three ways:

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

• Consider an 8-bit register and the number +14 which is represented as 00001110

• Consider an 8-bit register and the number –14 which can be represented as:

o Signed magnitude: 1 0001110

o Signed 1's complement: 1 1110001

o Signed 2's complement: 1 1110010

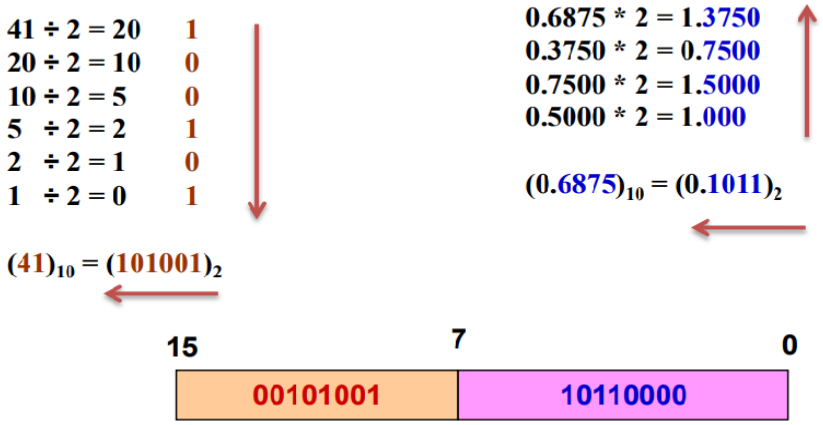
Example 1

Representation of-9 in Sign-Magnitude, One's Complement and Two's Complement form.

Signed-magnitude representation:	10001001
Signed-1's-complement representation:	11110110
Signed-2's-complement representation:	11110111

Example 2:

represent the real number 41.6875 by using fixed point method.



41.6875 = 101001.1011

Signed Fixed point Representation

Eg: Represent $(-7.5)_{10}$ using 8-bit binary representation with 4 digits integer & 4 fractie bits

Sol: $(7.5)_{10} = (111.1)_2 \Rightarrow$ Binary form.

$= (0111.1000)_2 \Rightarrow$ 4 int & 4 fraction digits

$(7.5)_{10} =$ 2's compl = 1's complement + 1

1's complement = 1000.0111

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1} \\ \hline 1000.1000 \end{array} \quad +$$

2's complement = 1000.1000

In fixed point representation, the position of binary point is fixed.

COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- DATA REPRESENTATION-FLOATING POINT REPRESENTATION

Floating-Point Representation

Using 32 bits to represent a number, positive or negative, the range of possible values is large but there are circumstances when bigger number representations are needed. The way to do this is to use floating point numbers. The reason for using floating point representation is that the range of possible values is much greater.

The floating-point representation of a number has two parts:
- The first part represents a signed, fixed-point number – the mantissa
- The second part designates the position of the binary point – the exponent
- The mantissa may be a fraction or an integer

Example: decimal number +6132.789 is
Mantissa: +6123789
Exponent: +04
Equivalent to $+0.6132789 \times 10^{+4}$

A floating-point number is always interpreted to represent $\boldsymbol{m \times r^e}$
Example: binary number+1001.11 (with 8-bit fraction and 6-bit exponent)
Mantissa: 01001110
Exponent: 000100
Equivalent to $+(.1001110)_2 \times 2^{+4}$

Normalisation of Floating point numbers

Normalisation, in floating-point number systems, is a process of standardizing these numbers into a consistent format. This procedure is crucial in computing for ensuring uniformity, accuracy, and efficiency in arithmetic operations and data storage.

Normalised form for floating point binary numbers is the equivalent of **standard form** for denary numbers. For example, the denary number 9257 can be represented as:

- 9257×10^0
- 925.7×10^1
- 92.57×10^2
- 9.257×10^3

Only the last example is represented in standard form. In standard form, the mantissa must be between 1 and 10 (>=1 and <10).

In the normalized form, there is only a single non-zero digit before the radix point.

Similarly, suppose you want to represent +12.510 as a floating point binary number. Here are some examples of the way that the number can be represented:

- 01100.1×2^0
- 0110.01×2^1
- 011.001×2^2
- 01.1001×2^3
- 0.11001×2^4

All of these examples represent the same value (+12.510), but only the final example is normalized.

For example, decimal 1234.567 is normalized as 1.234567×10^3

- by moving the decimal point so that only one digit appears before the decimal.
- The exponent expresses the number of positions the decimal point was moved left (positive exponent) or moved right (negative exponent).

To normalise a floating point number, you need to first determine whether the number is positive or negative by checking the most significant bit of the mantissa

Adjusting the mantissa

If the number is **positive**, you need to move through the mantissa from left to right until you find the first bit whose value is 1. The binary point will be positioned immediately in front of this bit.

A normalised positive number always starts as 0.1

For example, the floating point number 0.1101×2^3 **is normalised**, whilst the floating point number 0.01101×2^4 **is not**.

If the number is **negative**, you need to move through the mantissa from left to right until you find the first bit whose value is 0. The binary point will be positioned immediately in front of this bit.

A normalised negative number always starts as 1.0

For example, the floating point number 1.0011×2^3 **is normalised**, whilst the floating point number 100.11×2^1 **is not**.

Adjusting the exponent

If you move the binary point in the mantissa, you must adjust the exponent so that the value of the number remains the same.

- Every time you move the binary point **one place to the right** you must **reduce the value of the exponent by 1**.
- Every time you move the binary point **one place to the left** you must **increase the value of the exponent by 1**.

Ex 1:

- the positive floating point number 0.00011×2^5 is not normalised since it does not start with 0.1.
- To normalise this number, move the binary point 3 places to the right and reduce the value of the exponent by 3 to give 0.11×2^2.

Ex 2:

- the positive floating point number 010.01×2^{-3} is not normalised since it does not start with 0.1.
- To normalise this number, move the binary point 2 places to the left and increase the exponent by 2 to give 0.1001×2^{-5}.

COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT –III

TOPIC- ADDITION AND SUBTRACTION OF SIGNED MAGNITUDE DATA

PART-1

Addition and Subtraction of Signed Magnitude Data

We designate the magnitude of the two numbers by A and B. Where the signed numbers are added or subtracted, we find that there are eight different conditions to consider, depending on the sign of the numbers and the operation performed. T listed in the table below: when two equal numbers are subtracted, the result should be +0 not -0.

The algorithms for addition and subtraction are derived from the table below:

Addition: A + B ; A: Augend; B: Addend
Subtraction: A - B: A: Minuend; B: Subtrahend

Operation	Add Magnitude	Subtract Magnitude		
		When A>B	When A<B	When 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)

- When the signs of A and B are same, add the two magnitudes and attach the sign of result is that of A.
- When the signs of A and B are not same, 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 or 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 will be positive.

Examples of Signed magnitude addition and subtraction

Add

+35 = A
+40 = B
+(A+B)

+35 = 0 0 1 0 0 0 1 1
+40 = 0 0 1 0 1 0 0 0 +
0 1 0 0 1 0 1 1 → +75

+35 = A
−40 = B

A < B = −(B−A)

B − A = B + (−A)
= B + (2's complement of A)

+35 = 0 0 1 0 0 0 1 1
1 1 0 1 1 1 0 0 +
1
1 1 0 1 1 1 0 1 → −A
0 0 1 0 1 0 0 0 → B
1 1 1 1
1 0 0 0 0 0 1 0 1 → −5
o/p Carry

Subtract

+35 = A
+40 = B

A < B
−(B−A)
= B + (2's complement of A)

A = 0 0 1 0 0 0 1 1
1 1 0 1 1 1 0 0
1
1 1 0 1 1 1 0 1 → −A
40 → 0 0 1 0 1 0 0 0 + → B
1 1 1 1
1 0 0 0 0 0 1 0 1 → −5
o/p Carry

+35 = A
−40 = B. +(A+B)

A = 0 0 1 0 0 0 1 1
B = 0 0 1 0 1 0 0 0 +
1
0 1 0 0 1 0 1 1 → +75

COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- ADDITION AND SUBTRACTION OF SIGNED MAGNITUDE DATA
PART-2

Addition and Subtraction of Signed Magnitude Data

Hardware Implementation

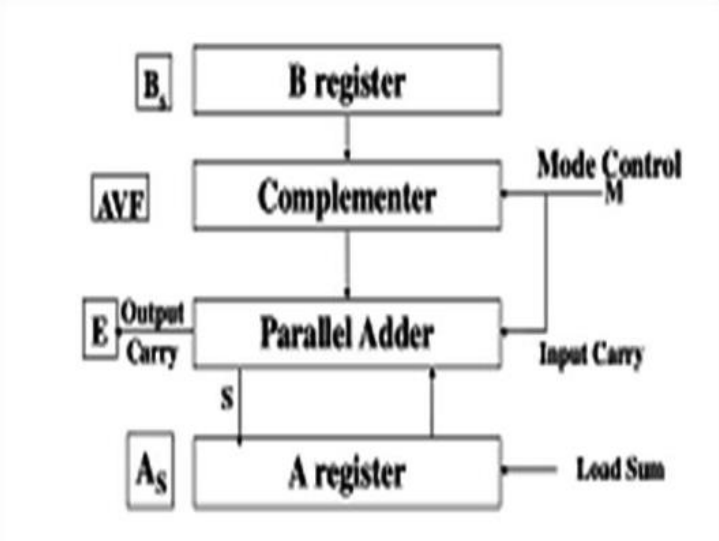


Figure: Hardware Implementation of signed magnitude addition and subtraction.

- Let A and B be the two registers that holds the magnitudes of the numbers and A_s and B_s be two flip flops that holds the corresponding sign.
- The result of the operation may be transferred to the third register or the result is transferred to A and A_s.
- First parallel adder is needed to perform micro operation: A+B. Second comparator circuit needed to establish if A<B,A>B or A==B.
- Third subtractor circuit is needed to perform the microoperation A-B and B-A.
- The hardware implementation of addition and subtraction in the above figure consist of registers A and B and the sign flip flops A_s and B_s.
- Subtraction is done by adding A to the 2's complement of B.
- The o/p carry is transferred to E.
- The add overflow flip flop(AVF) holds the overflow bit when A and B are added.
- The addition A+B is done through the parallel adder and the sum is transferred to A register.
- When the Mode bit M=0 the output of B register is transferred to the adder, the

input carry is 0 and the output of the adder is equal to sum A+B.

- When M=1, the 1's complement of B is applied to adder, the input carry is 1 and the output is equal to A+ B'+1.

Flow chart for Signed magnitude addition and subtraction

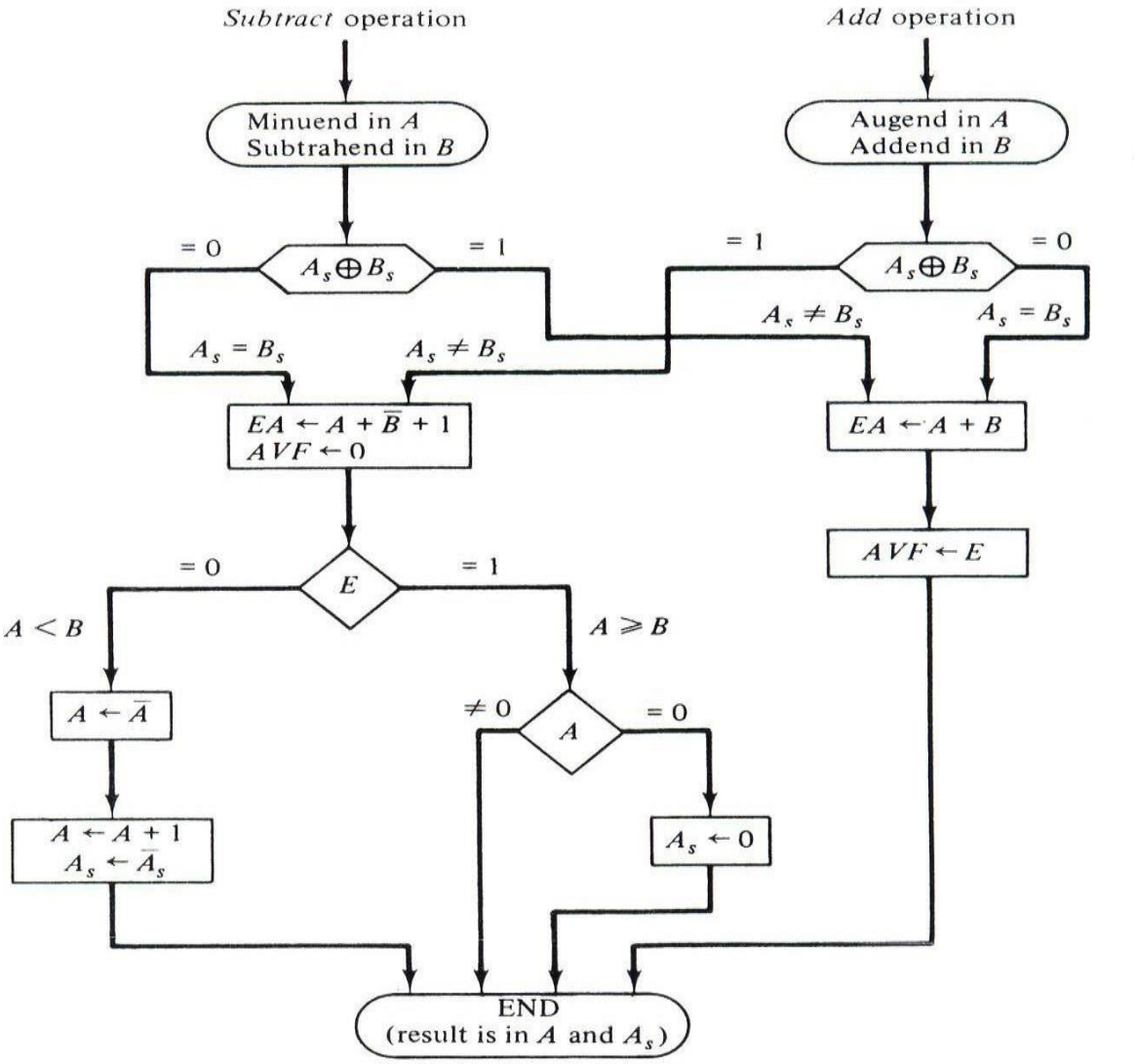


Figure 10-2 Flowchart for add and subtract operations.

- The two signs A, and B, are compared by an exclusive-OR gate.
- If the output of the exclusive-OR gate is 0 the signs are identical; If it is 1, the signs are different.
- For an add operation, identical signs dictate that the magnitudes be added.
- For a subtract operation, different signs dictate that the magnitudes be added.

- The magnitudes are added with a micro operation: 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 flip-flop AVF.

- The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation.

- The magnitudes are subtracted by adding A to the 2's complemented B. No overflow can occur if the numbers are subtracted so AVF is cleared to 0.

- If E=1, then the condition is A>=B and the number in A is the correct result.

- If E=0 then the condition is A<B and the number in A is taken 2’s complement which is the correct result.

- If the sign of the result is same as the sign of A, So no change in A_s is required.
- When A<B the sign of the result is the complement of the original sign of A.
- The final result is found in register A and its sign in A_s.

2 input XOR gate

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

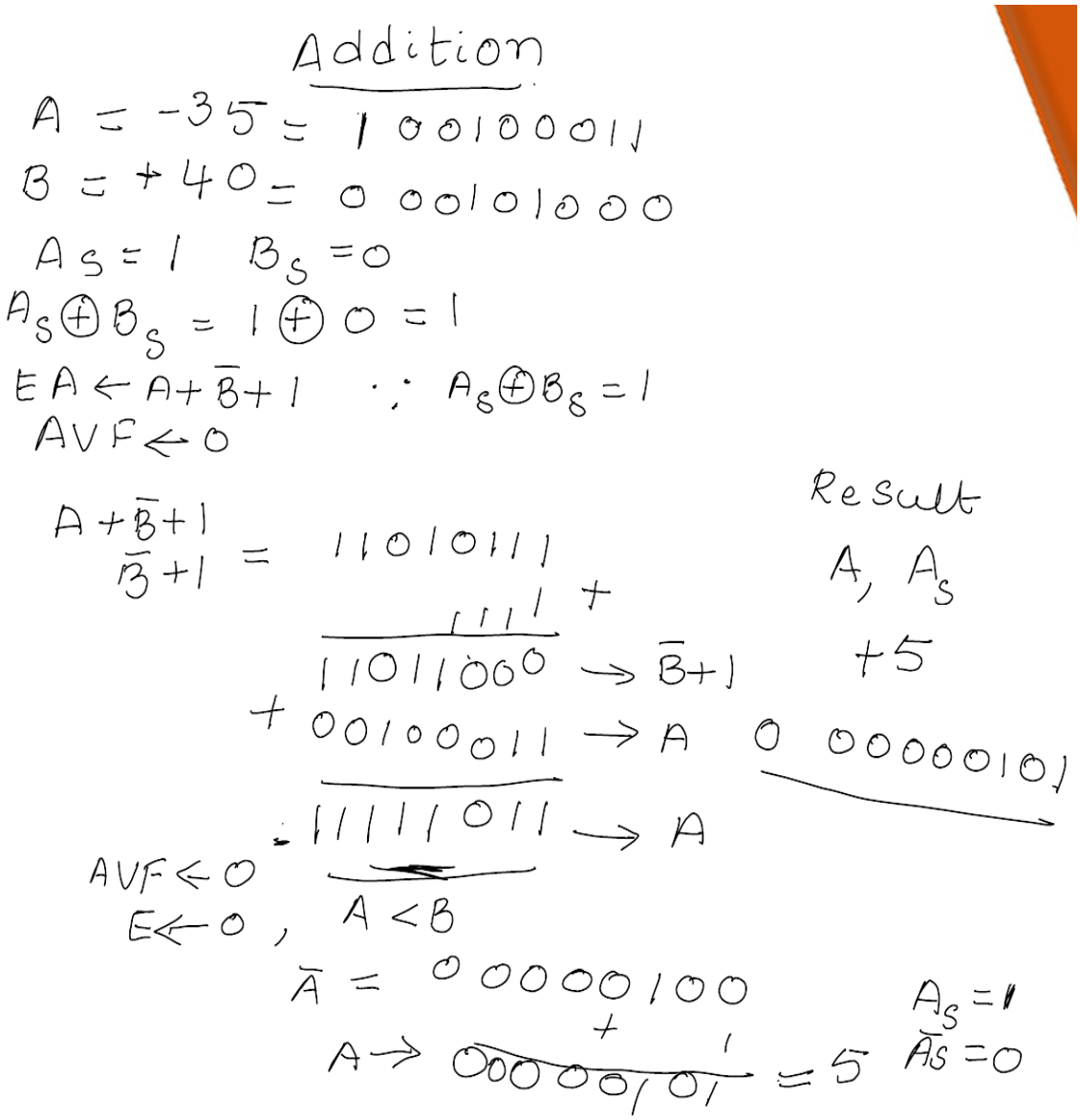
COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT –III

TOPIC- ADDITION AND SUBTRACTION OF SIGNED MAGNITUDE DATA PART-3

Addition and Subtraction of Signed Magnitude Data

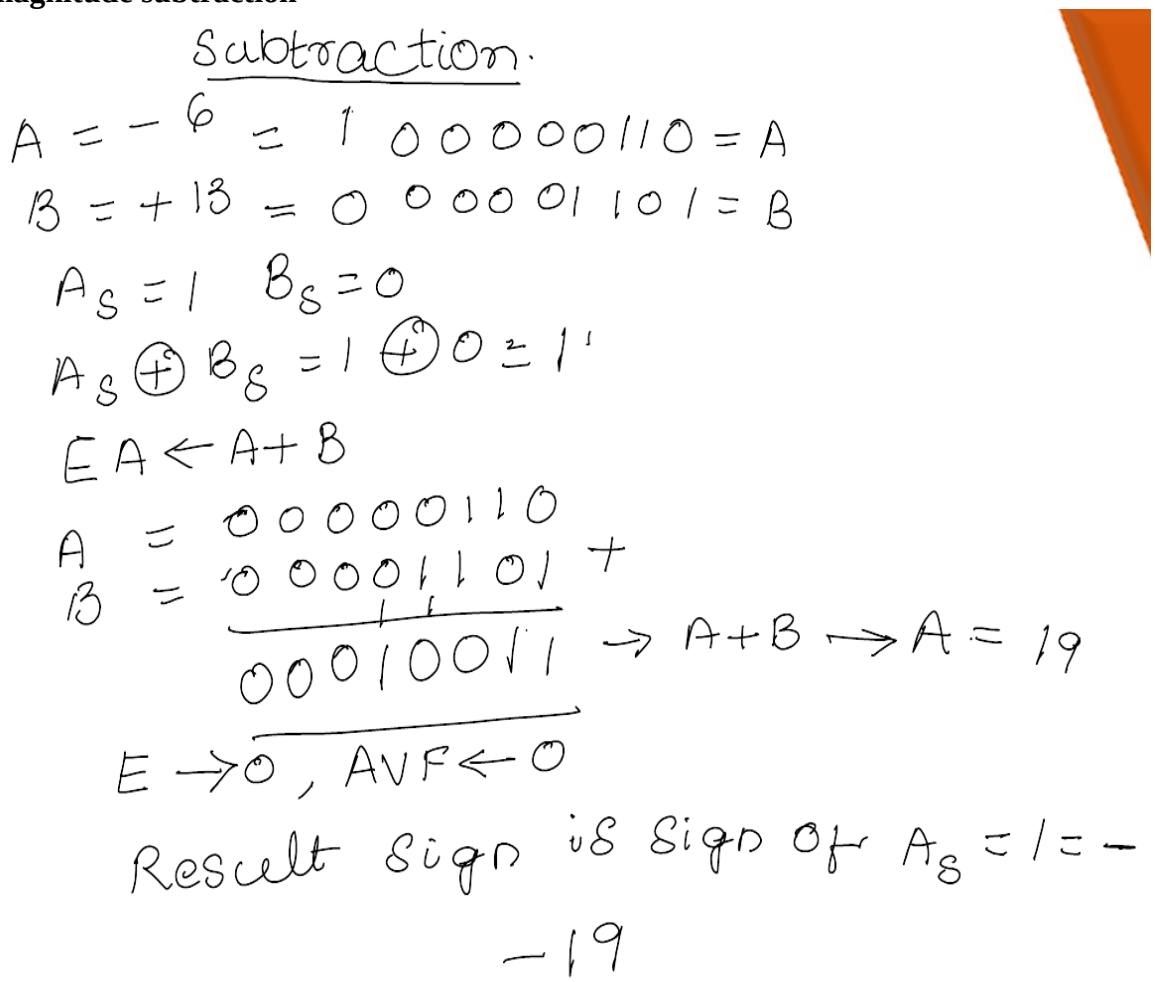
Example problem 1: Perform an addition of -35 and +40 using signed magnitude addition.



Try it yourself:

1. **35 + (- 40)**
2. **35 + 40**
3. **(-35) + (- 40)**

Example problem 2: Perform a subtraction of -6 and +13 using signed magnitude subtraction



Try it yourself:

1. **6 – (-13)**
2. **6 – 13**
3. **(-6) – (-13)**

COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- MULTIPLICATION OF SIGNED MAGNITUDE DATA PART-1

Introduction

Multiplicand Multiplier Product

Example of Binary Multiplication: $(10111)_2$ x $(1010)_2=(0110110)_2$

Four rules of binary multiplication: **0x0=0**
0x1=0
1x0=0
1x1=1

			1	0	1	1	1
x				1	0	1	0
			0	0	0	0	0
		1	0	1	1	1	
	0	0	0	0	0		
1	0	1	1	1			
1	1	1	0	0	1	1	0

- If the multiplier bit is a 1,the multiplicand is copied down; otherwise zero are copied down.

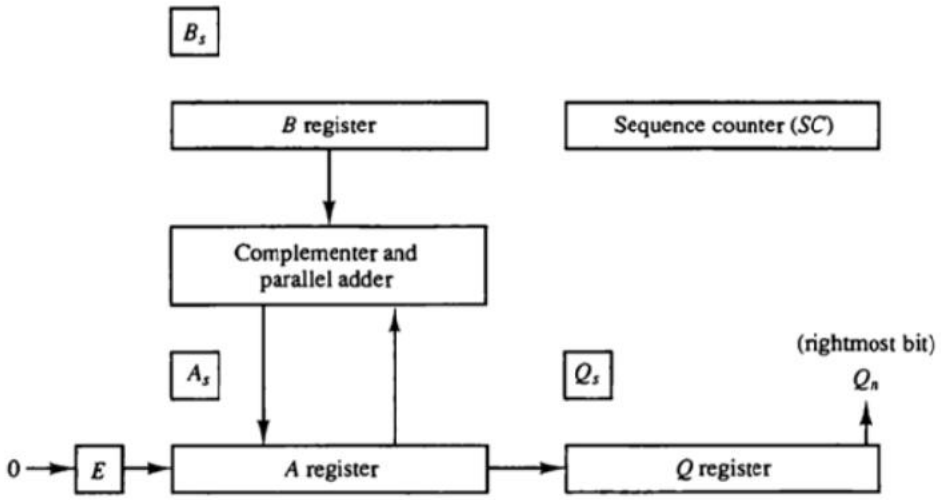
- Finally all the partial products are added to get the desired product.

- The sign of the product is determined from the sign of the multiplicand and multiplier.

- If they are same sign then product is positive and if they are of different sign, the sign of the product is negative.

Multiplication Algorithm of signed magnitude Data

Hardware Implementation for Signed-Magnitude data



- The multiplier stored in the Q register and its sign in Q_s
- 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. When the counter reaches to zero, the product is formed and process stops.
- Initially the multiplicand is in B register and multiplier in Q register.
- The sum of A and B forms a partial product which is transferred to the EA register .
- The shift will be denoted by the statement shr EAQ to designate the right shift.
- The least significant bit of A is shifted into the most significant position of Q.

Flowchart of signed magnitude Multiplication

- The multiplicand is in B and the multiplier in Q. Their corresponding signs are in B_s and Q_s respectively.
- We compare the signs of both A and Q and set to corresponding sign of the product since a double-length product will be stored in registers A and Q.
- Registers A and E are cleared and the sequence counter SC is set to the number of bits of the multiplier.
- Now, the low order bit of the multiplier in Q_n is tested. If it is 1, the multiplicand (B) is added to present partial product(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.
- When SC = 0 we stops the process.
- The final product is available in both A and Q, with A holding the most significant bits

and Q holding the least significant bits and the sign of the product stored in A_s and Q_s.

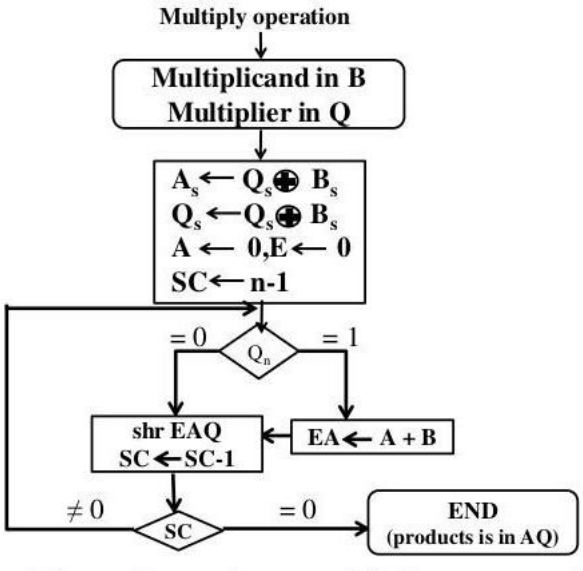


Figure: Flowchart of signed magnitude multiplication

COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT –III

TOPIC- MULTIPLICATION OF SIGNED MAGNITUDE DATA PART-2

Signed magnitude multiplication

Flowchart of signed magnitude Multiplication

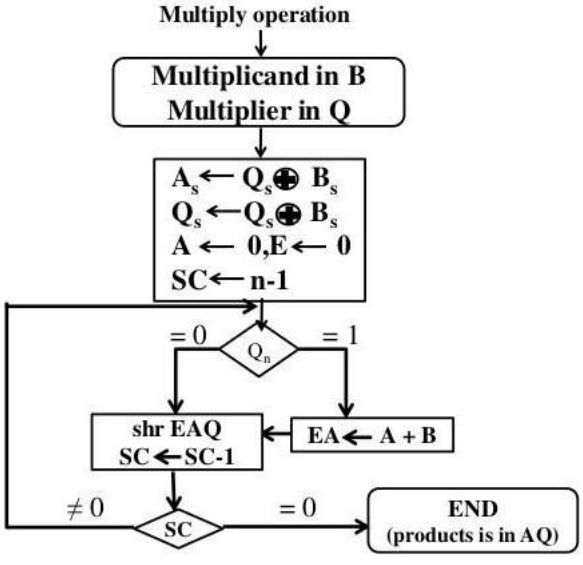


Figure: Flowchart of signed magnitude multiplication

EXAMPLE 1:

Example 23 × 19.

0 10111 → Multiplicand

0 10011 → Multiplier

```
  10111
× 10011
-------
  10111
 10111
00000
00000
-------
0 110110101  ⇒ +437.
-------
```

product.

Example
Multiply 23 X 19 = 437

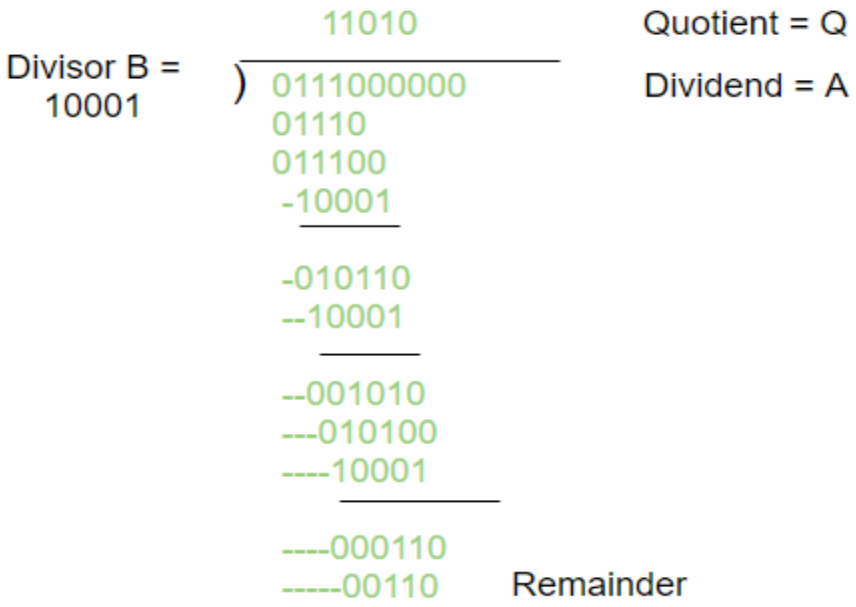
TABLE 10-2 Numerical Example for Binary Multiplier

Multiplicand B = 10111	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
Q_n = 1; add B		10111		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
Q_n = 1; add B		10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
Q_n = 0; shift right EAQ	0	01000	10110	010
Q_n = 0; shift right EAQ	0	00100	01011	001
Q_n = 1; add B		10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final product in AQ = 0110110101				

COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- DIVISION OF SIGNED MAGNITUDE DATA PART-1

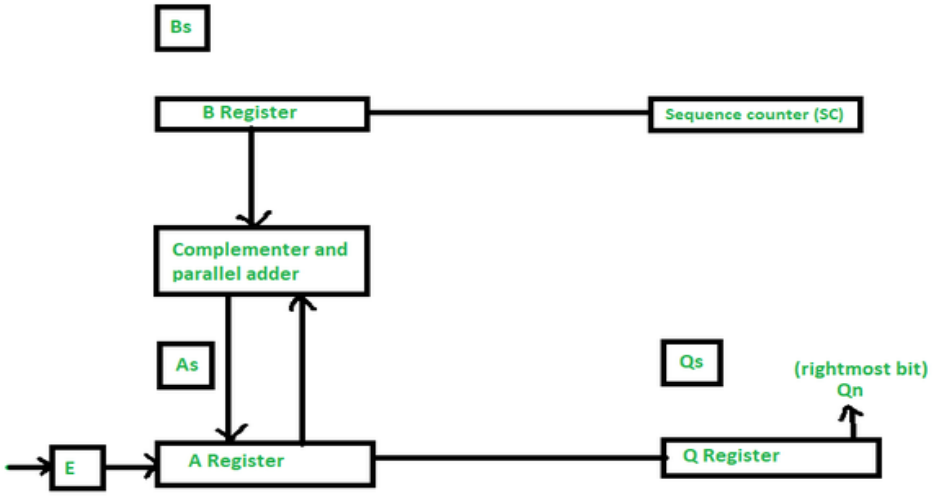
Signed magnitude Division

The Division of two fixed-point binary numbers in the signed-magnitude representation is done by the cycle of successive compare, shift, and subtract operations.



- First the bits of dividend are examined from left to right, until the set of bits examined represent the number greater than or equal to divisor.
- Until this event occurs 0's are placed in quotient from left to right.
- When the event occurs 1 is placed in the quotient and divisor is subtracted from the partial dividend. The result is referred as partial remainder.
- At each cycle of this process, an additional bits from the dividend are appended to the partial remainder, until the result is greater than or equal to the divisor.
- This process continues until all the bits of the dividend are exhausted.

Hardware Implementation:



- The hardware implementation of division is identical to multiplication.
- Here, the divisor is stored in the B register and the double-length dividend is stored in registers A and Q.
- Register EAQ is now shifted to the left with 0 inserted into Q_n and the previous value of E is lost. The dividend is shifted to the left and the divisor is subtracted by adding its 2's complement value.
- End carry gives the information about the relative magnitudes.

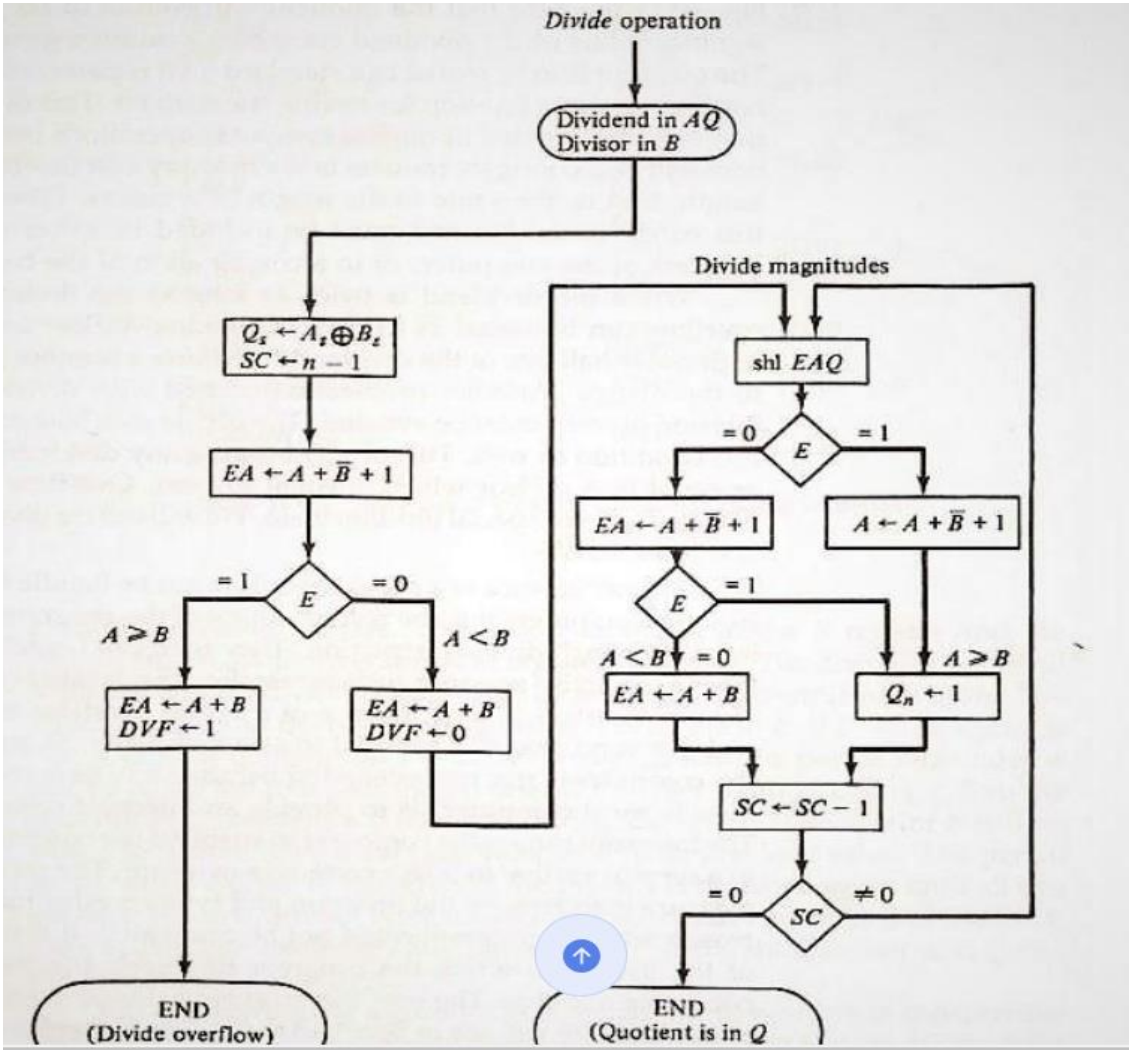
Divide overflow

- The division operation may result in a quotient with an overflow.
- The quotient is to be stored in a standard register, so the overflow bit will require one more flip flop for storing the sixth bit.
- The divide overflow condition must be avoided in normal computer operations because the entire quotient will be too long for transfer into a memory unit that has words of standard length.
- Provision to ensure that this condition is detected must be included in either the hardware or the software.
- When the dividend is twice as long as the divisor, the condition for overflow can be stated as follows: a divide overflow condition occurs if the high order half bits of the dividend constitute a number greater than or equal to the divisor.
- Another problem associated with division is the fact that a division by zero must be avoided.
- Overflow condition is usually detected when a special flip flop is set.(DVF)

COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- DIVISION OF SIGNED MAGNITUDE DATA PART-2

Signed magnitude Division

Flowchart:



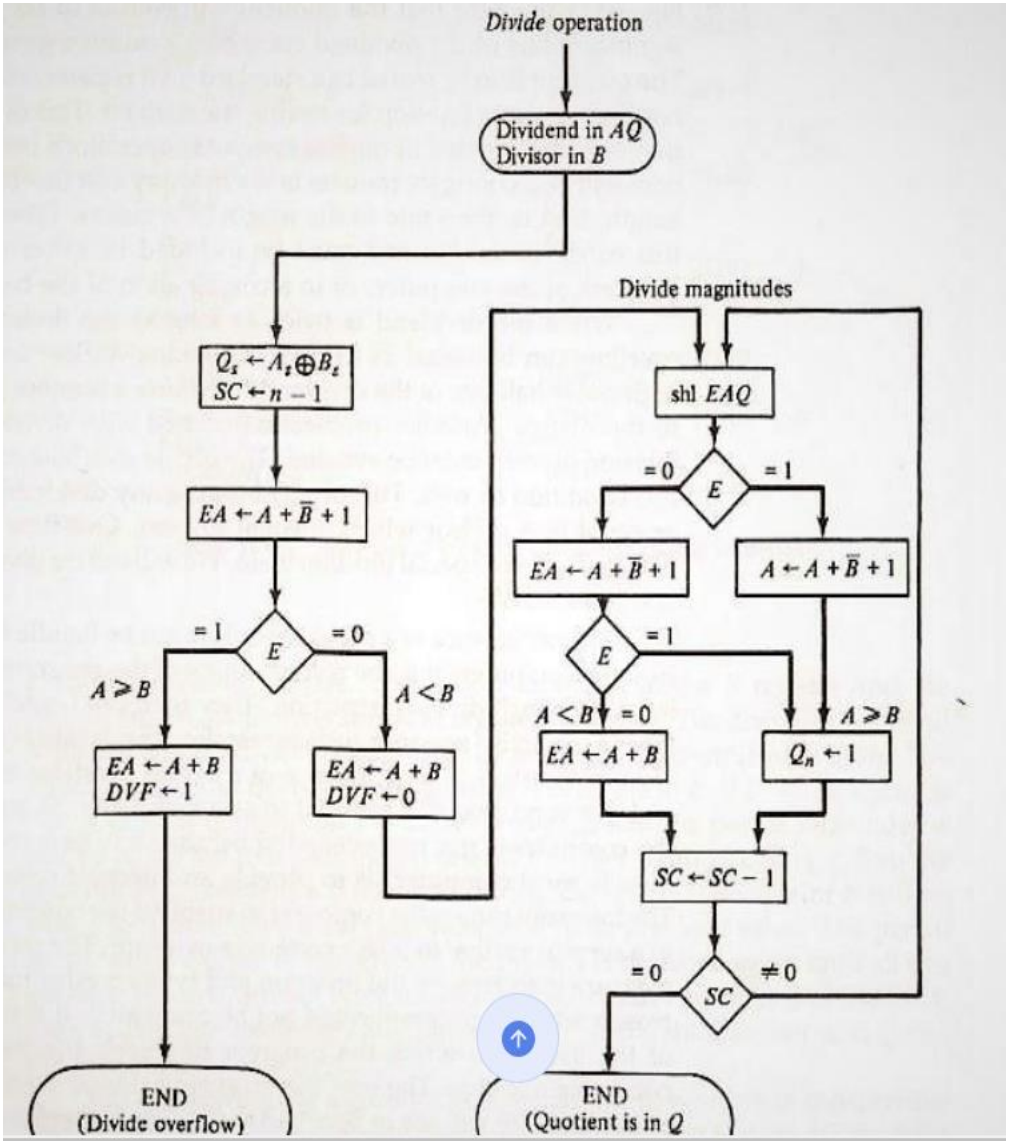
- Initially, the dividend is in A & Q and the divisor is in B and their sign bits in A_s and Q_s and B_s respectively.
- After the division is performed, the quotient is stored in Q and its sign in Q_s and remainder is stored in A.
- The sign of the result is transferred into $Q_s \leftarrow A_s$ xor B_s, to be part of the quotient.
- SC is set to specify the number of bits in the quotient.

- The condition of divide-overflow is checked by subtracting the divisor in B from the half of the bits of the dividend stored in A.
- If $A \geq B$, DVF is set and the operation is terminated before time.
- If $A < B$, no overflow condition occurs and so the value of the dividend is reinstated by adding B to A.
- The division of the magnitudes starts with the dividend in AQ to left in the high-order bit shifted into E.
- If shifted a bit into E is equal to 1, and we know that $EA > B$.
- In this case, B must be subtracted from EA, and 1 should insert into Q_n, for the quotient bit.
- If the shift-left operation inserts a 0 into E, the divisor is subtracted by adding its 2's complement value and the carry is moved into E.
- If $E = 1$, it means that $A \geq B$; thus, Q_n, is set to 1.
- If $E = 0$, it means that $A < B$, and the original number is reimposed by adding B into A.
- Now, this process is repeated with register A containing the partial remainder.

COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- DIVISION OF SIGNED MAGNITUDE DATA PART-3

Signed magnitude Division

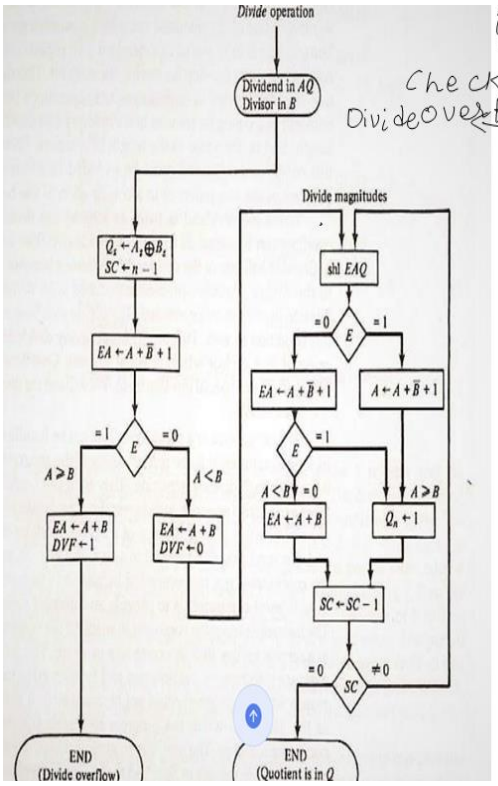
Flowchart:



Example 1:

Divisor B = 10001, Dividend A = 0111000000

Check for Divide overflow:



Ex: Divisor B = 10001 $B_S = 0$

Check for Divide overflow

Divident = 01110 00000

A Q

$Q_S \leftarrow A_S \oplus B_S = 0 \oplus 0 = 0$ $A_S = 0$ $Q_S = 0$

$SC \leftarrow n - 1 = 5$

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

A = 01110

$\bar{B} + 1$ = 01111

E → 0 A → 11101

if E = 0, A < B, DVF → 0.

$EA \leftarrow A + B$

A = 11101

B = 10001 +

E ← 1 | 01110 → A

Divide Magnitudes:

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

Final Remainder: 00110
Final Quotient: 11010

Try it yourself

1. Show the contents of registers E, A, Q, and SC during the process of division of
(a) 10100011 by 1011;
(b) 00001111 by 0011. (Use a dividend of eight bits.)

COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT –III

TOPIC- ADDITION & SUBTRACTION OF SIGNED 2'S COMPLEMENT DATA

• When an integer is positive, the MSB, or sign bit, is 0 and the remaining bits represent the magnitude.
• When an integer is negative, the MSB, or sign bit, is 1, but the rest of the number can be represented in one of three ways:

- Signed-magnitude representation
- Signed-1's complement representation
- Signed-2's complement representation

Addition and subtraction of Signed 2's complement data

Addition using 2's complement data

- The sign bit is represented by the leftmost bit of a binary number: 0 for positive and 1 for negative.
- The full number is displayed in 2's complement form if the sign bit is 1.
- As a result, +33 is represented as 00100001, and -33 is represented as 11011111.
- The 2's complement of 00100001 is 11011111.
- When two numbers are added in signed -2's complement form, the sign bits of the numbers are regarded the same as the other bits of the number.
- The sign-bit position's carry-out is disregarded.

Ex1: Use the 2's complement approach to add two decimal integers of +7 and +4.

Solution: The 2's complement representations of +4 and +7 with 5 bits each are shown below.

$+7_{10} = 00111_2$
$+4_{10} = 00100_2$
The addition of these two numbers is
$(+7_{10}) + (+4_{10}) = 00111_2 + 00100_2 = 01011_2$

- The resultant sum is 5 bits long.
- As a result, there is no carryover from the sign bit.
- The final total is positive, as shown by the sign bit '0'.
- In the decimal number system, the magnitude of the sum is 11.
- Therefore, the addition of two positive numbers will give another positive number.

Ex 2:use the 2's complement approach to subtract two decimal integers of +7 and +4.

Solution: The subtraction of these two numbers is

$(+4_{10}) - (+7_{10}) = (+4_{10}) + (-7_{10})$
The 2's complement representations of +4 and -7 with 5 bits each are shown below:

$+4_{10} = 00100_2$
$-7_{10} = 11001_2$
$(+4_{10}) + (-7_{10}) = 00100_2 + 11001_2 = 11101_2$

- Here, carry is not obtained from the sign bit.
- The final total is negative, as shown by the sign bit '1'.
- As a result, we may determine the magnitude of the resultant sum as 3 in the decimal number system by taking 2's complement of it.
- Therefore, the subtraction of two decimal numbers, +4 and +7, is -3.

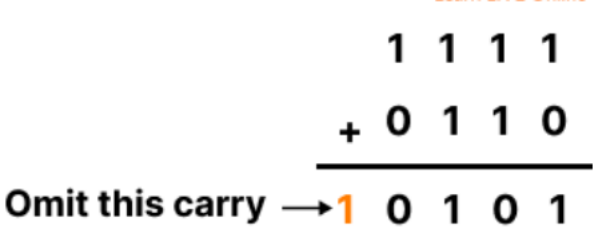
Subtraction Using 2's Complement data

The following steps are to be followed:

- **Step 1:** Determine the 2's complement of the smaller number
- **Step 2:** Add this to the larger number.
- **Step 3:** Omit the carry. Note that there is always a carry in this case.

Example: Subtract $(1010)_2$ from $(1111)_2$ using 2's complement method.

- **Step 1:** 2's complement of $(1010)_2$ is $(0110)_2$.
- **Step 2:** Add $(0110)_2$ to $(1111)_2$.

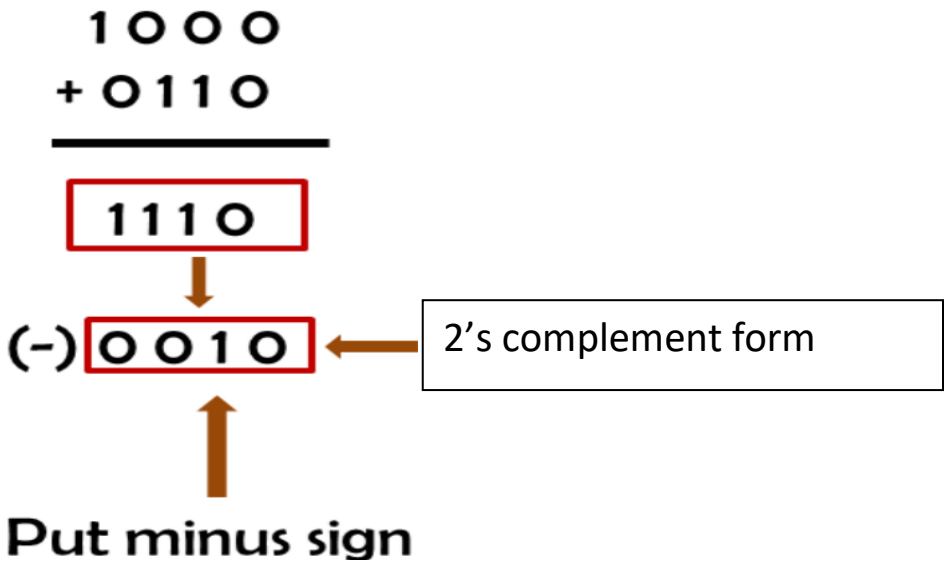


To subtract a larger number from a smaller number using 2's complement subtraction, the following steps are to be followed:

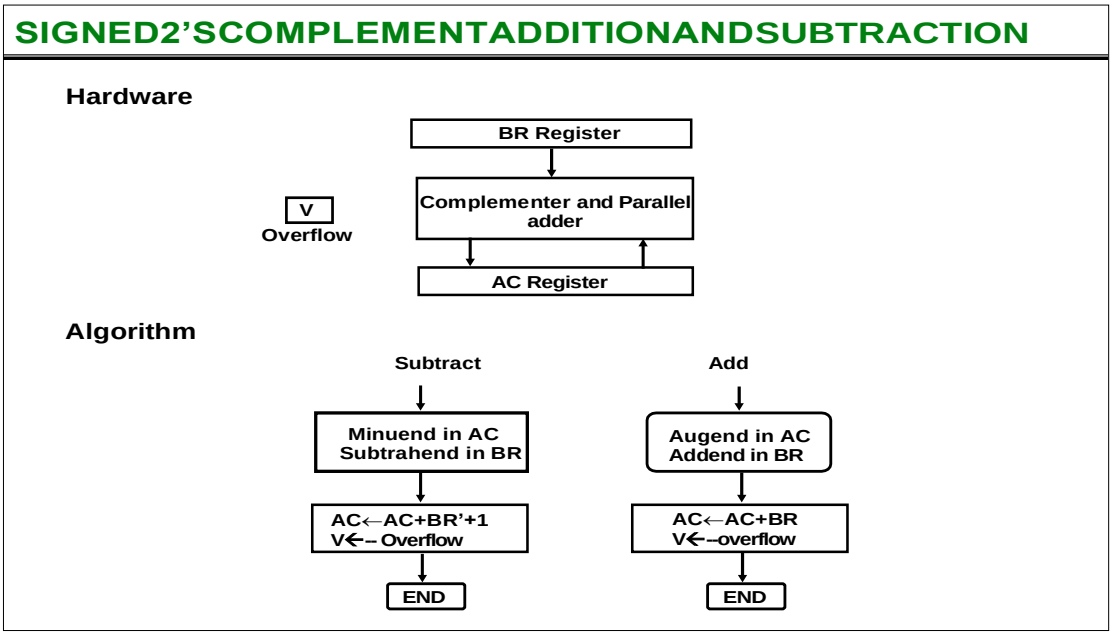
- **Step 1:** Determine the 2's complement of the smaller number.
- **Step 2:** Add this to the larger number.
- **Step 3:** There is no carry in this case. The result is in 2's complement form and is negative.
- **Step 4:** To get an answer in true form, take 2's complement and change its sign.

Example: Subtract $(1010)_2$ from $(1000)_2$ using 2's complement.

- **Step 1:** Find the 2's complement of $(1010)_2 = (0110)_2$.
- **Step 2:** Add $(0110)_2$ to $(1000)_2$.



Hardware Implementation



- This is same as signed magnitude addition & subtraction except that the sign bits are not separated from the rest of the registers.
- 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.
- The two sign bits are added or 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.

Algorithm

- The sum is obtained by adding the contents of AC and BR (including their sign bits).
- 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.
- An overflow must be checked during this operation because the two numbers added could have the same sign.
- The programmer must realize that if an overflow occurs, there will be an erroneous result in the AC register.
- Comparing this algorithm with signed magnitude, it is simpler to add and subtract numbers if negative numbers are in signed 2’s complement form.
- So the most of the computers adopt this representation over signed magnitude form.

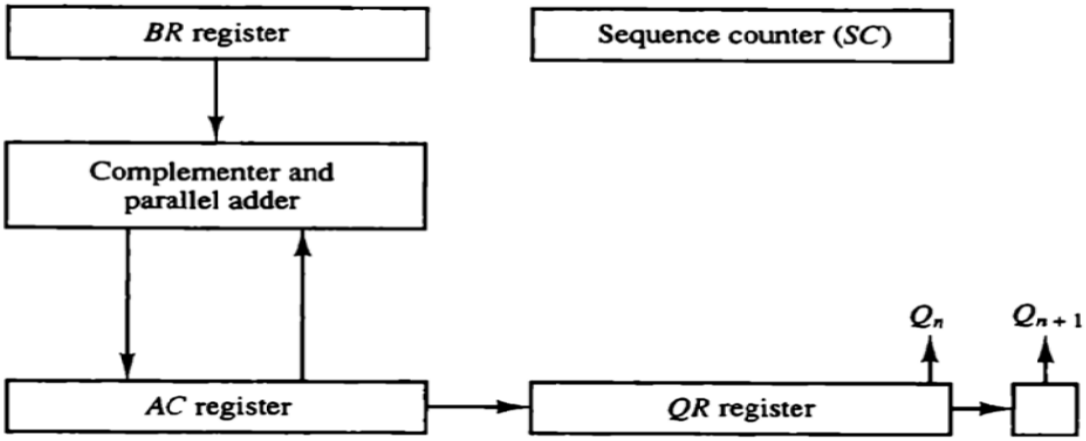
COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- BOOTH'S MULTIPLICATION ALGORITHM PART-1

Introduction

The Booth Algorithm is a widely used technique for multiplying binary integers particularly in signed 2's complement representation.
It was proposed by Andrew D in 1951.
This algorithm is an efficient way to perform multiplication, minimizing the number of additions and subtractions.

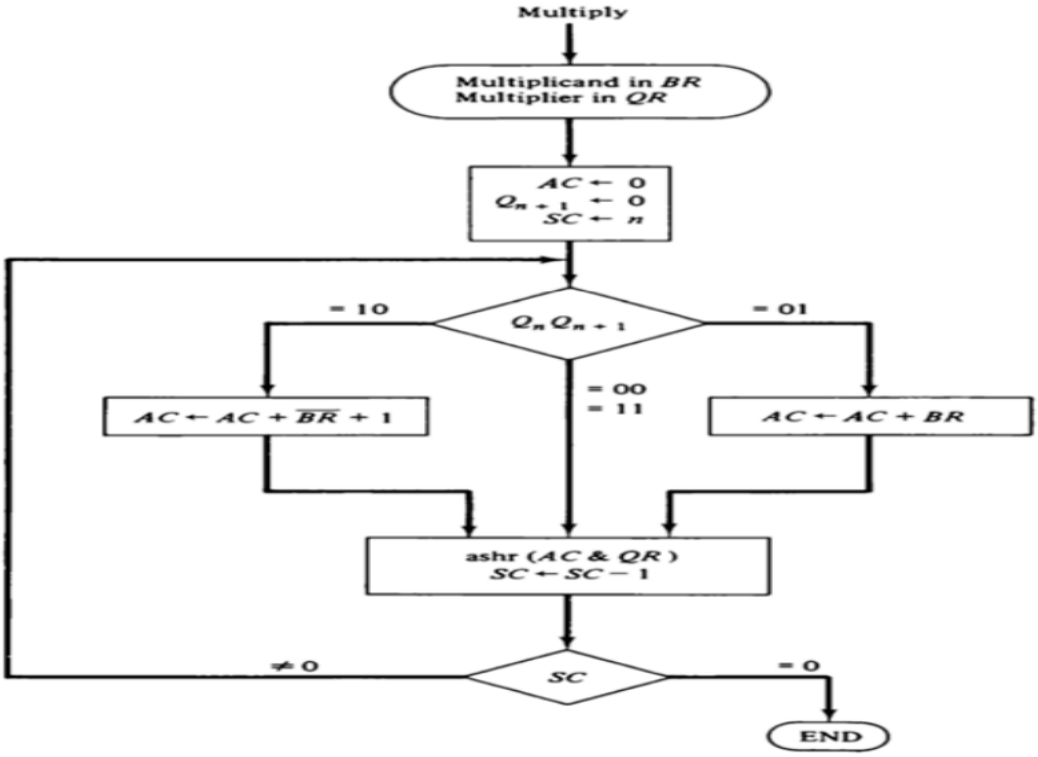
Hardware Implementation

Hardware for Booth algorithm.



• The multiplicand is stored in BR register and the multiplier is stored in QR register.
• AC is a temporary register initialized to 0.
• The complementer has a mode bit which can be 0 or 1.
• If the mode bit is 0, complementer transfers the contents of BR register to the parallel adder.
• If the mode bit is 1, complementer transfers the complement of BR register to the parallel adder.
• Parallel adder performs addition of BR and AC registers and register is transferred to AC register.
• An arithmetic shift right is performed on AC and QR registers by checking the values of Q_n and Q_{n+1} bits.
• Q_n is LSB of the multiplier.
• Q_{n+1} is an extra bit or flip flop initialized to 0 in the algorithm.

Flowchart



Flowchart of Booth's multiplication

1. Initially, we set the AC and Q_{n+1} registers value to 0.
2. SC represents the number of Multiplier bits (Q), and it is a sequence counter that is continuously decremented till equal to the number of bits (n) or reached to 0.
3. A Q_n represents the last bit of the Q, and the Q_{n+1} shows the incremented bit of Q_n by 1.
4. On each cycle of the booth algorithm, Q_n and Q_{n+1} bits will be checked on the following parameters as follows:

    - When two bits Q_n and Q_{n+1} are 00 or 11, we simply perform the arithmetic shift right operation (ashr) of AC and QR by 1 and SC is decremented by 1.
    - If the bits of Q_n and Q_{n+1} is shows to 01, the multiplicand bits (M) will be added to the AC (Accumulator register). Then, we perform the right shift operation to the AC and QR bits by 1 and SC is decremented by 1.
    - If the bits of Q_n and Q_{n+1} is shows to 10, the multiplicand bits (M) will be subtracted from the AC (Accumulator register). Then, we perform the right shift operation to the AC and QR bits by 1 and SC is decremented by 1.

5. The operation continues until SC reaches 0.
6. Results of the multiplication (product) will be stored in the AC and QR registers with 2's complement representation for negative numbers.

Advantages of Booth's Algorithm

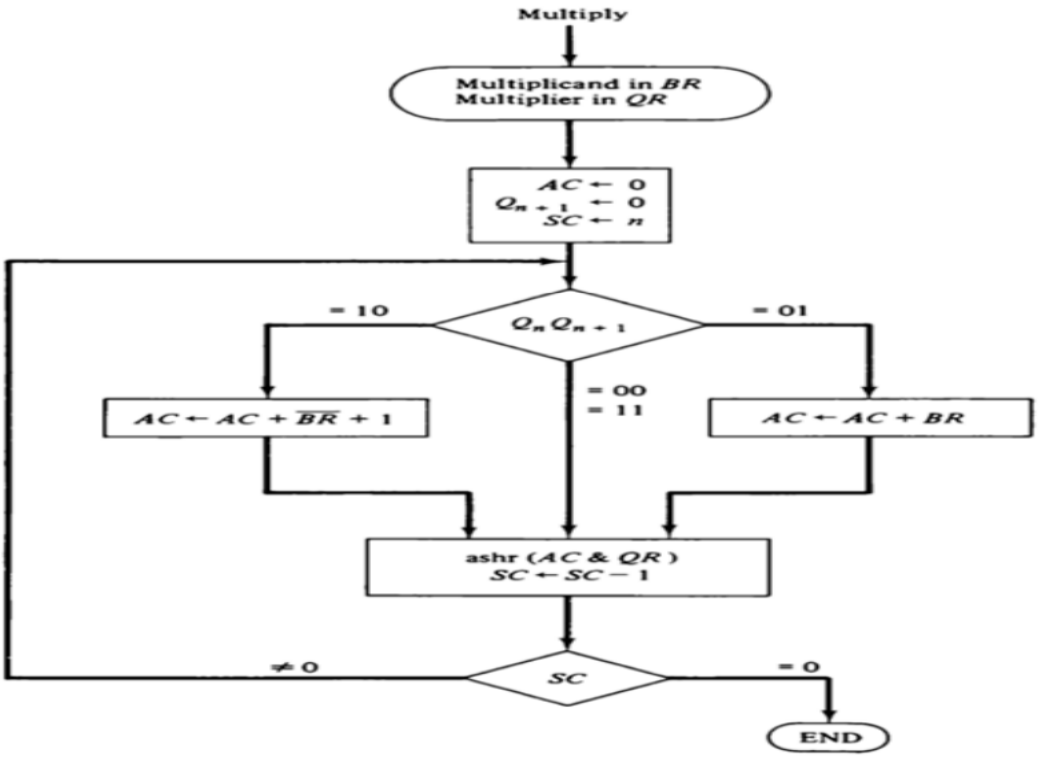
- This algorithm is used to multiply signed binary numbers, handling both positive and negative numbers effectively without requiring separate logic for negative numbers.
- The algorithm minimizes the number of additions and subtractions required during multiplication, leading to a faster computation compared to traditional methods.
- Booth's algorithm is well-suited for hardware implementation because it reduces the complexity of arithmetic operations, which is crucial for devices with limited resources like embedded systems.

Disadvantages of Booth's Algorithm

- It can be more difficult to understand and implement compared to traditional multiplication methods. It requires careful handling of sign bits.
- It is primarily designed for signed binary numbers. To use it for unsigned numbers, modifications are required, making it less versatile for all types of binary multiplication.
- The algorithm involves multiple iterations of addition, subtraction, and shifting, which may lead to higher latency compared to simpler multiplication methods.
- Although it reduces the number of operations, the algorithm may still consume more power due to the multiple cycles involved in shifting and adding or subtracting the multiplicand.

COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- BOOTH'S MULTIPLICATION ALGORITHM PART-2

Flowchart of Booth's algorithm



Flowchart of Booth's multiplication

EXAMPLE 1:

Example

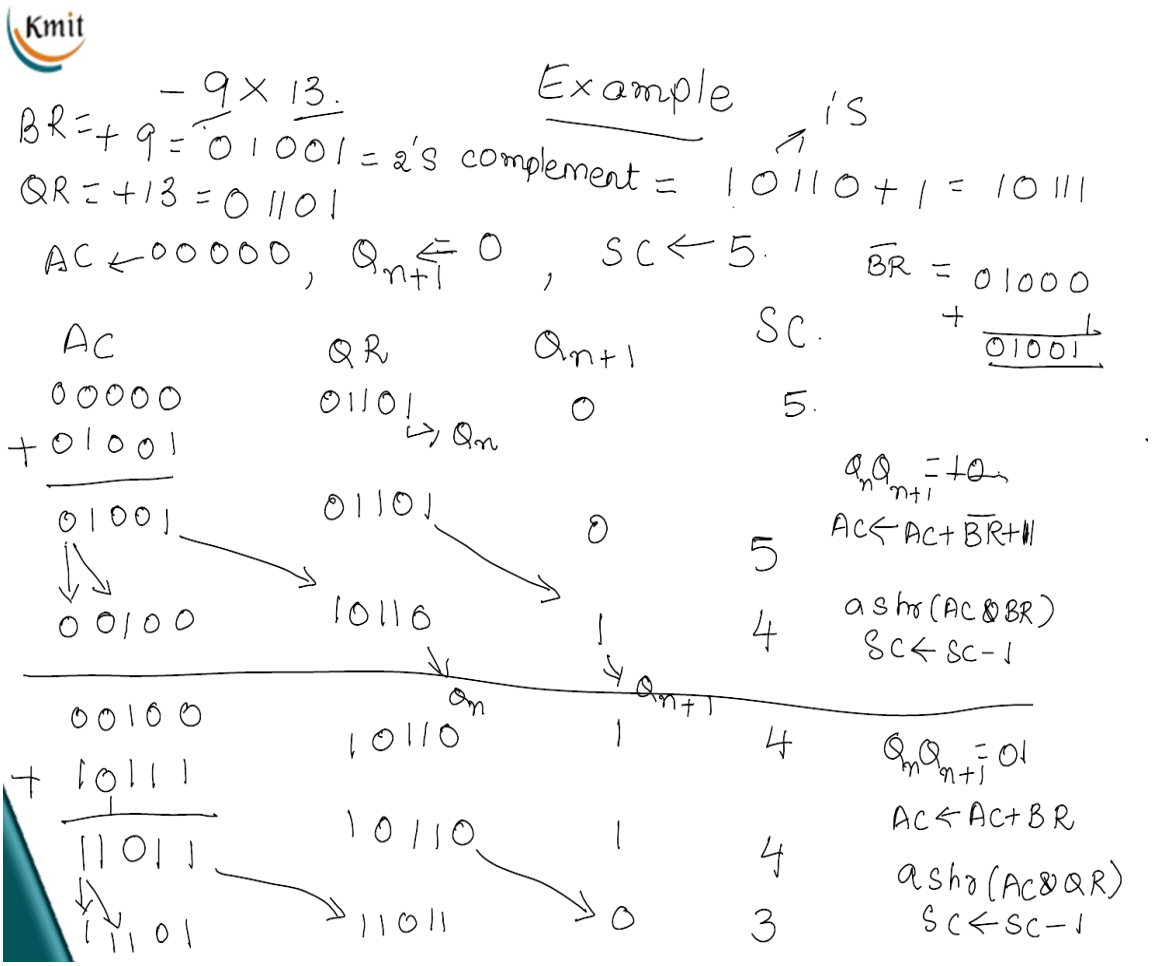
-9 X -13 = + 117

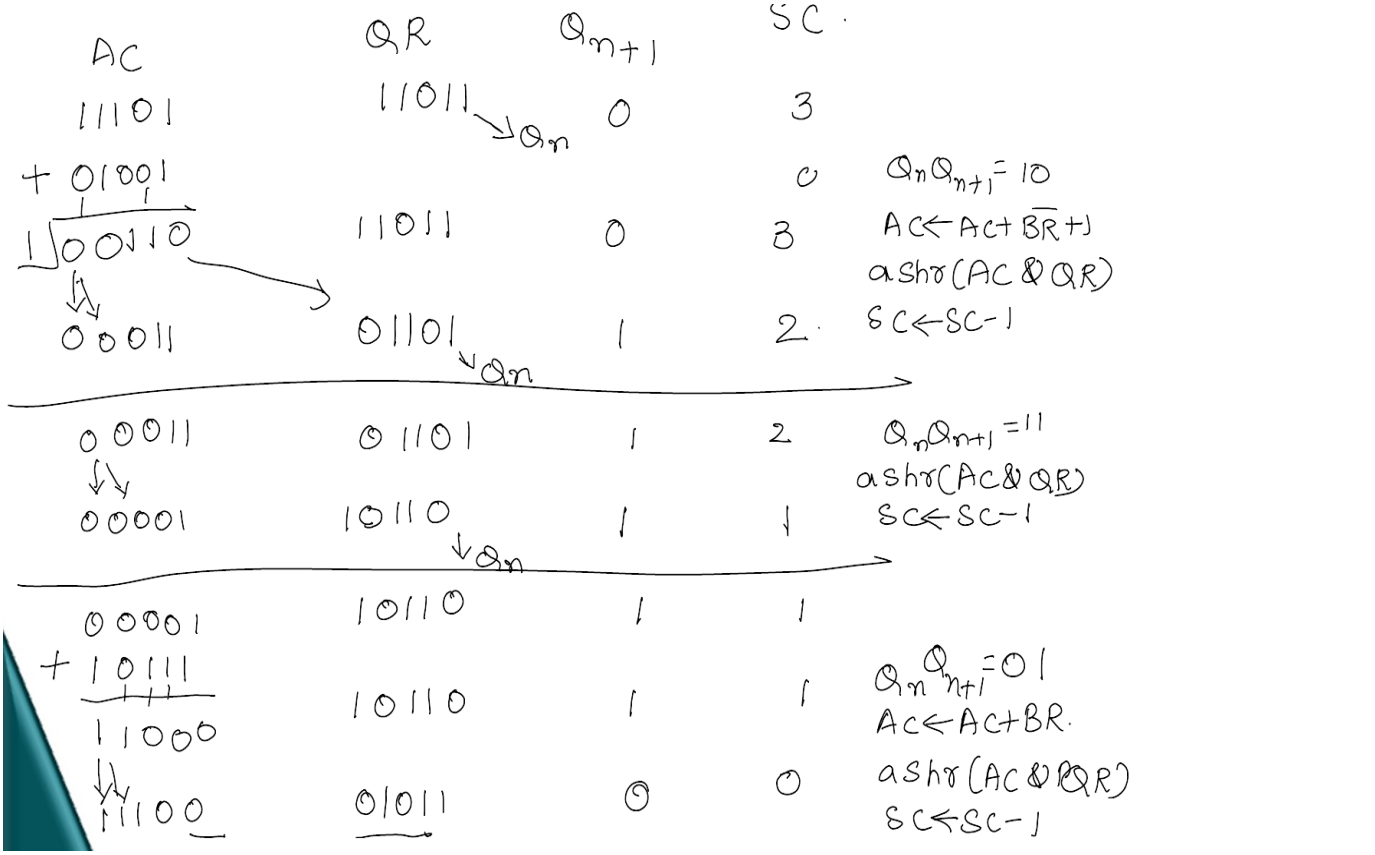
$Q_n\ Q_{n+1}$	BR = 10111 $\overline{BR}$ + 1 = 01001	AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
1 0	Subtract BR	01001			
		01001			
	ashr	00100	11001	1	100
1 1	ashr	00010	01100	1	011
0 1	Add BR	10111			
		11001			
	ashr	11100	10110	0	010
0 0	ashr	11110	01011	0	001
1 0	Subtract BR	01001			
		00111	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

Product= AC & QR= 0001110101=117

Since the msb bit of product is 0 , the sign of the product is positive= +117

EXAMPLE 2:


Example
-9 x 13.
BR = +9 = 01001 = 2's complement = 10110 + 1 = 10111
QR = +13 = 01101
AC ← 00000, Q_{n+1} ← 0, SC ← 5.
$\overline{BR}$ = 01000 + 1 = 01001
AC
QR
Q_{n+1}
SC.
00000
+01001
01101
Q_n
0
5.
Q_nQ_{n+1} = +0
AC ← AC + $\overline{BR}$ + 1
01001
01101
0
5
ashr (AC & BR)
SC ← SC - 1
00100
10110
1
4
Q_n
Q_{n+1}
00100
+10111
10110
1
4
Q_nQ_{n+1} = 01
AC ← AC + BR
11011
10110
1
4
ashr (AC & QR)
SC ← SC - 1
11101
11011
0
3



AC = 11100 QR = 01011 Q_{n+1} = 0 SC = 0

SC = 0 end.

→ msb = 1, is a negative no.

product = 1110001011

= 0001110100 ↓ +

0001110101 ⇒ −117.

Since the MSB bit of the product is 1, it indicates that the result is not the actual product. Perform the 2's complement of the product stored in AQ & QR to get the actual product.

COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- FLOATING POINT ADDITION & SUBTRACTION PART-1

Floating Point Addition & Subtraction

Introduction

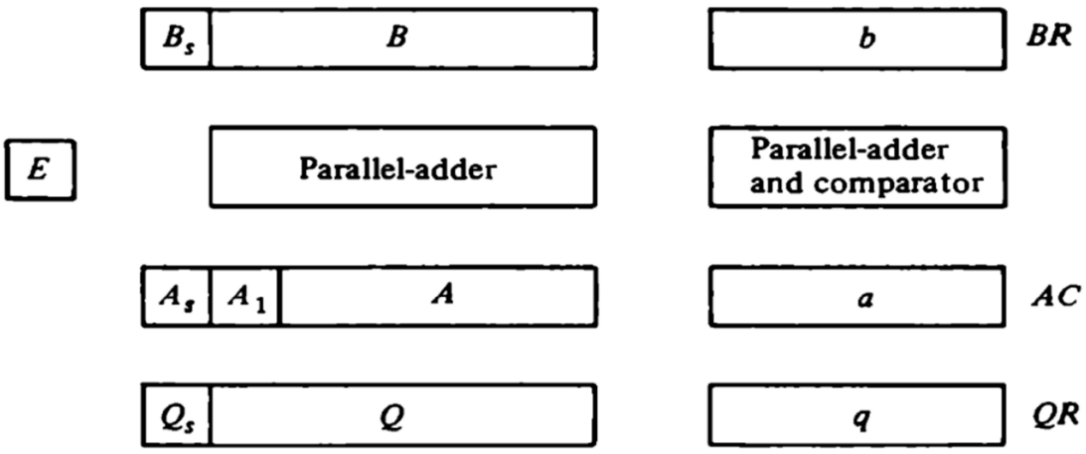
Floating point number in computer register consists of two parts: a mantissa m and exponent e which will be represented as m X r ^{e}

F = m x r^{e}
where m: Mantissa
r: Radix
e: Exponent

Number	Mantissa	Base	Exponent
9x10^8	9	10	8
110x2^7	110	2	7
4364.784	4364784	10	-3

Hardware Implementation

Figure 10-14 Registers for floating-point arithmetic operations.



- The register configuration for floating point operation is quite similar to the layout for fixed point operation.

- The same register and adder used for fixed point arithmetic are used for processing the mantissas.
- The difference lies in the way the exponents are handled.
- There are three registers BR, AC and QR.
- Each register is subdivided into two parts.
- The mantissa part has same upper case letters, the exponent part uses the corresponding lower case letters.
- A parallel adder adds the two mantissas and transfers the sum into A and the carry into E.
- A separate parallel adder is used for the exponents. Since the exponents are biased

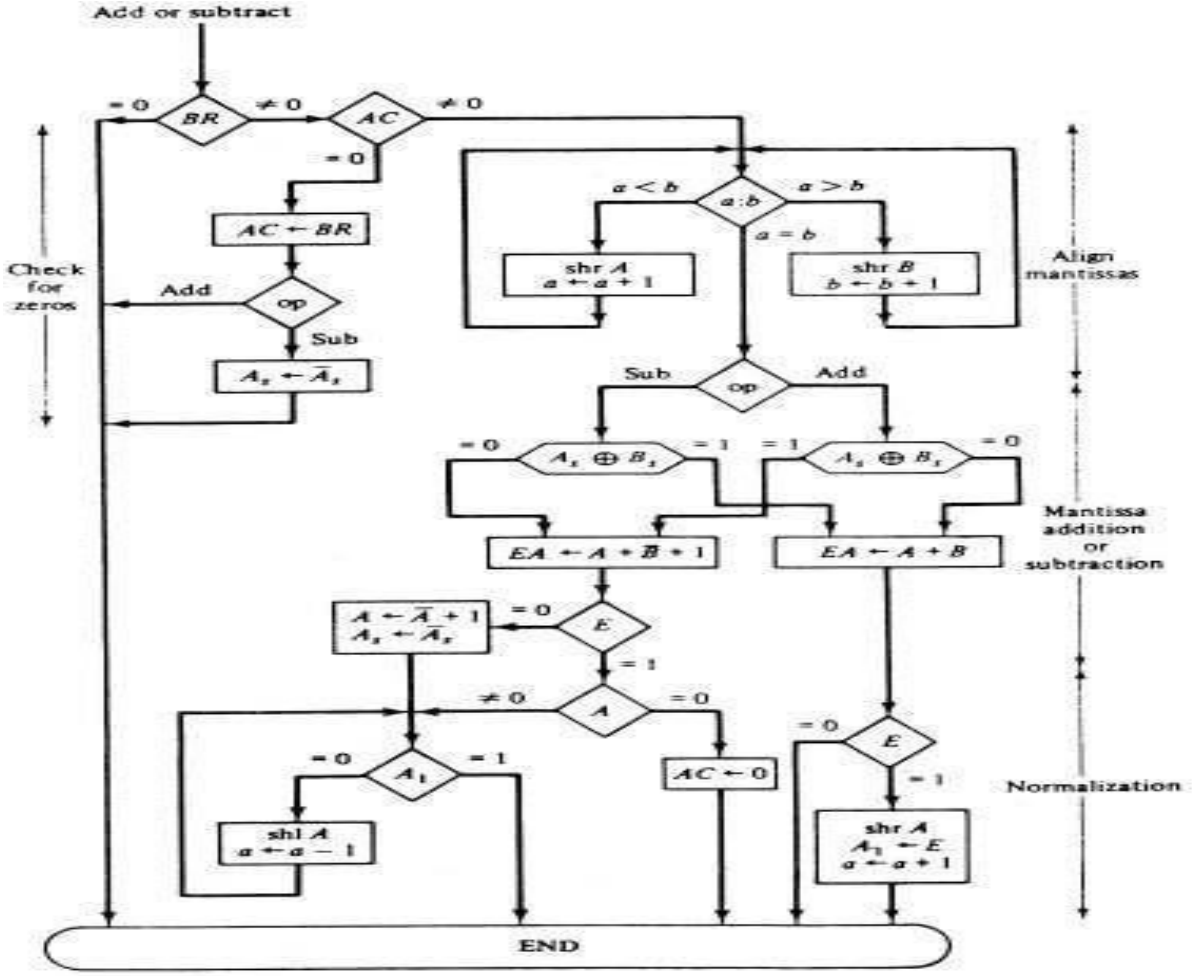
COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- FLOATING POINT ADDITION & SUBTRACTION PART-2

Floating point Addition & Subtraction
Flowchart

- During addition and subtraction, the two floating point operands are in AC and BR. The sum or difference is formed in the AC.

- The algorithm can be divided into four consecutive parts :

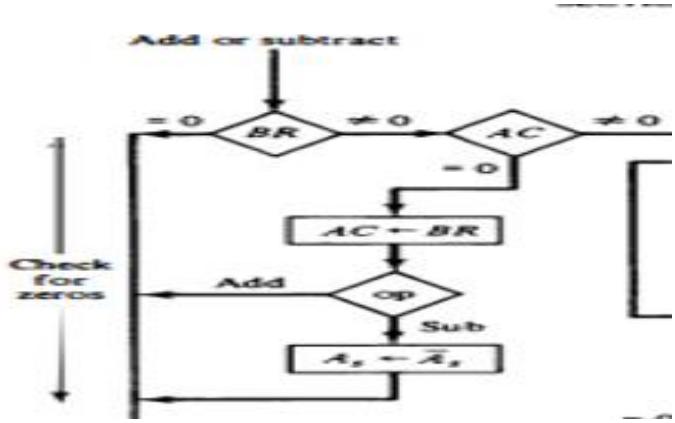
1. Check for zeros.
2. Align the mantissa.
3. Add or subtract the mantissa.
4. Normalize the result.



1. **Check for zeros.**

AC=0	AC=0.00	AC=0.00
BR=0	BR=0.125	BR=-125
AC=0	AC←BR	AC←-BR

(Reversing sign bit)



2**. Align the mantissa.**

AC=0.538123X10^3
BR=0.123000X10^-1

Align mantissa i.e.,exponent values must be equal.

Difference between 3 & -1 is 4
[3-(-1)]=4

Approach 1: perform **shift left** operation(making first number exponent as 10^-1

AC=0.538123X10^3
AC=0.230000X10^-1
BR=0.123000X10^-1

Data will be lost(MSB)

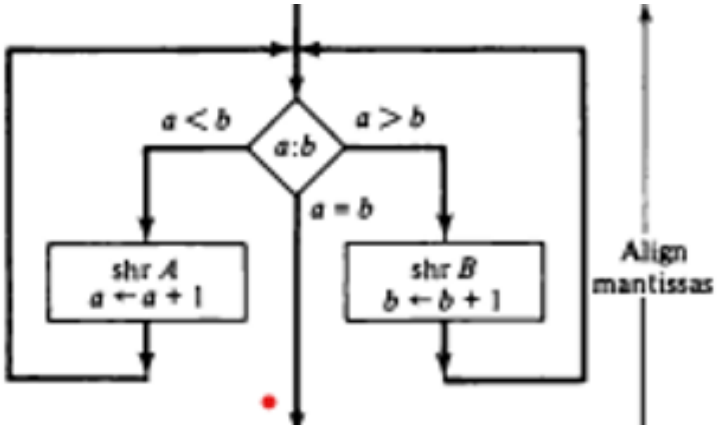
Decrement exponent by 4

Approach 2: perform **shift right** operation(making 2nd number exponent as 10^3

AC=0.538123X10^3

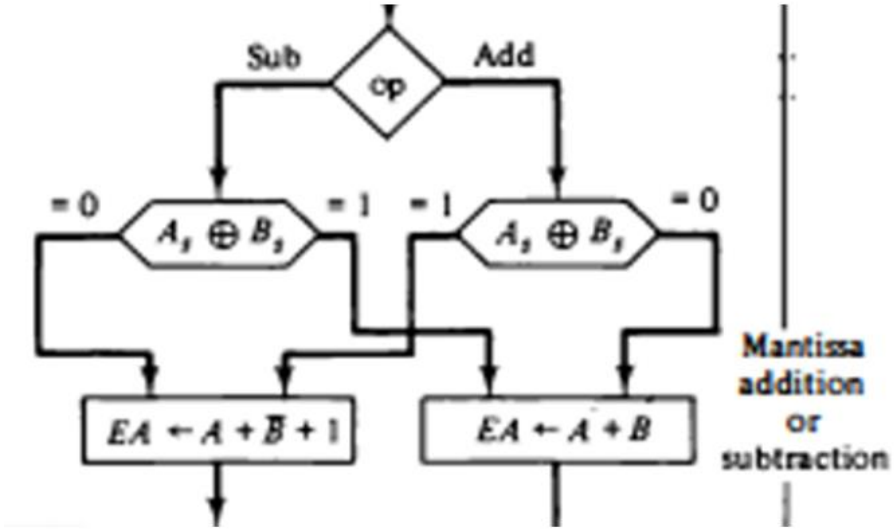
BR=0.123000X10^-1

BR=0.000012X10^3



2. Add or subtract the mantissa.

0.532x10^3	0.532x10^3
+ 10.712x10^3	- 0.521x10^3
1.244x10^3	0.011x10^3



4. Normalize the result.

• A floating point number that has a 0 in the most significant position of the mantissa is said to have an UNDERFLOW.

• To normalize a number that contains an underflow, it is necessary to shift the mantissa to the left and decrement the exponent until a nonzero digit appears in the first position

0.532x10^3
+ 10.712x10^3
1.244x10^3

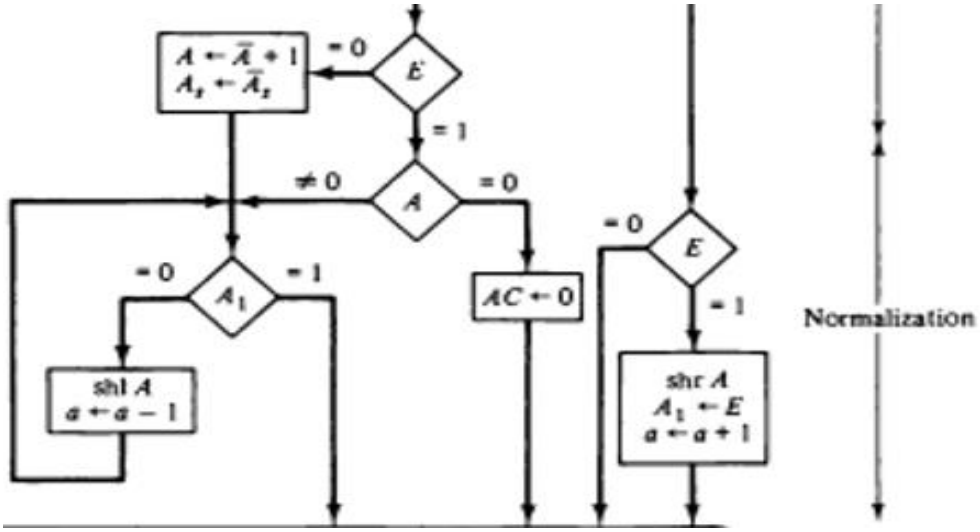
overflow

shift right
1.244x10^3
0.124x10^4(increment exponent)

0.532x10^3
- 0.521x10^3
0.011x10^3

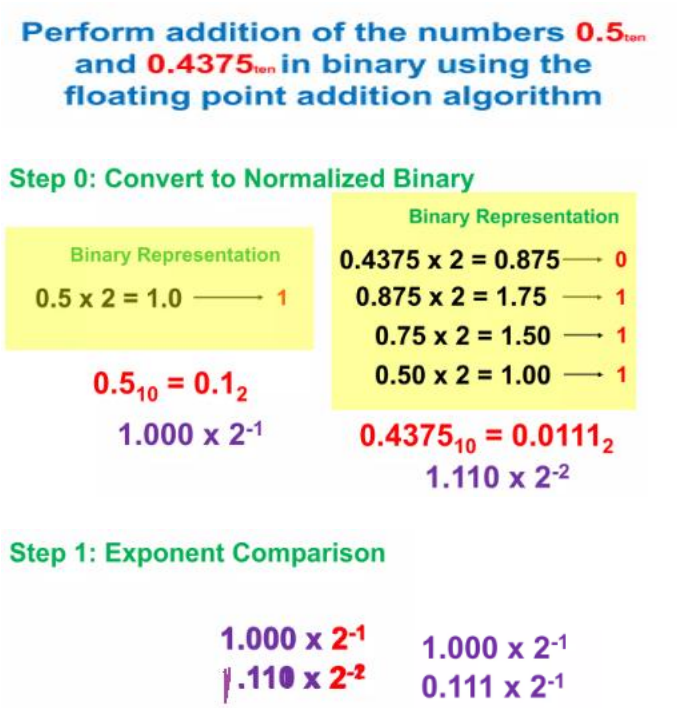
underflow

shift left
0.011x10^3
0.110x10^2(decrement exponent)



- 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 are to be subtracted.
- If neither number is equal to zero, we proceed to align the mantissas.
- The magnitude comparator attached to exponents a and b provides three outputs that indicate their relative magnitude. If the two exponents are equal, perform the arithmetic operation.
- If the exponents are not equal, the mantissa 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 and subtraction of the two mantissas is identical to the fixed-point addition and subtraction algorithm.
- The magnitude part is added or 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 A1 and all other bits of A are shifted right. The exponent must be incremented to maintain the correct number.
- If the magnitudes were subtracted, the result may be zero or may have an underflow.
- If the mantissa (A) is zero, the entire floating-point number in the AC is made zero. Otherwise, the mantissa must have at least one bit that is equal 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 A1 is checked again and the process is repeated until it is equal to1. When A1 = 1, the mantissa is normalized and the operation is completed.

Example of floating point addition



Step 2: Addition

$$\begin{array}{l} 1.000 \\ 0.111 \quad (+) \\ \hline 1.111 \end{array} \qquad 1.111 \times 2^{-1}$$

Step 3: Normalization

$$1.111 \times 2^{-1}$$

There is an overflow out of the result.

To normalize the result, perform a shift right operation and increment the exponent.

Final Answer

$$1.111 \times 2^{-1} = 0.1111$$

$$0.9375_{10}$$

COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- FLOATING POINT MULTIPLICATION PART-1

Floating Point Multiplication

Introduction

Floating point multiplication is comparatively easy than the floating point addition algorithm but off course consumes more hardware than fixed point multiplier circuit.

The multiplication of two floating point numbers requires that we multiply the mantissas and add the exponents. No comparison of exponents or alignment of mantissa is necessary.

Flowchart

The multiplication algorithm can be subdivided into four parts :-

1. Check for zeros.

2. Add the exponents.

3. Multiply the mantissa.

4. Normalize the product.

1. Check for zeros.

$BR=0.5812x10^5$

$QR=0.234x10^7$

If BR or QR = 0

Result i.e., $AC \leftarrow 0$

2. **Add the exponents.**

In multiplication there is no need to align mantissa, we need to add the exponents which will become the exponent of the result.

$a^m \times a^n = a^{m+n}$

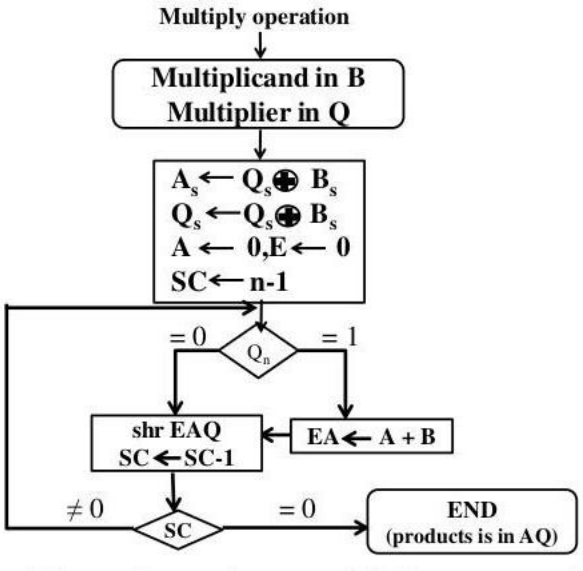
BR=0.5812x10⁵

QR=0.234x10⁷

$10^5 \times 10^7 = 10^{5+7} = 10^{12}$

3. Multiply the mantissa.

Use signed magnitude multiplication to perform multiplication of mantissa.



4. Normalize the product.

let the result is

1.0101101×10^{12}

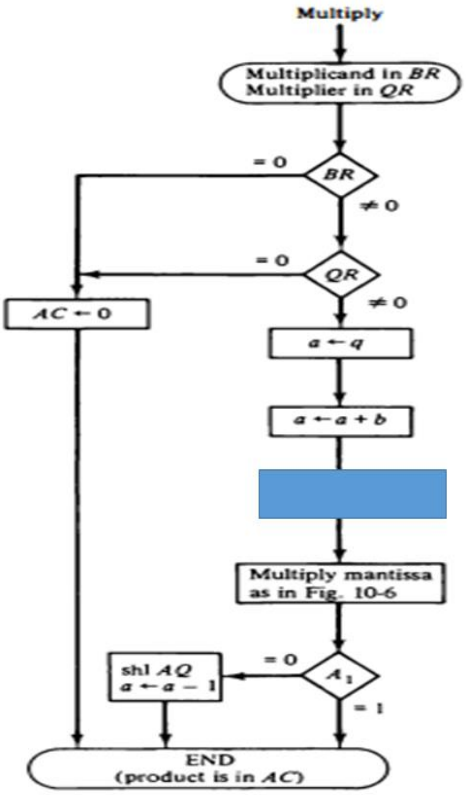
Msb of mantissa=0 underflow

0.101101×10^{11}

NORMALIZED

To overcome underflow, perform a shift left and exponent is decreased.

Figure 10-16 Multiplication of floating-point numbers.



b-multiplicand exponent
q-multiplier exponent

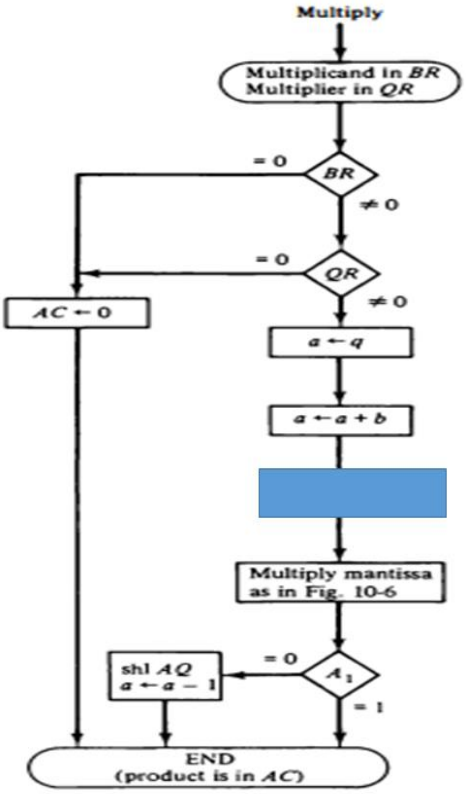
- The two operands are checked to determine if they contain a zero.If either operand is equal to zero, the product in the AC is set to zero and the operation is terminated. If neither of the operands is equal to zero, the process continues with the exponent addition.
- The exponent of the multiplier is in q and the adder is between exponents a and b. It is necessary to transfer the exponent from q to a, add the two exponents, and transfer the sum into a.
- The multiplication of the mantissas is done as in the fixed-point case with the product residing in A and Q. Overflow cannot occur during multiplication, so there is no need to check for it.
- The product may have an underflow, so the most significant bit in A is checked. If it is a 1, the product is already normalized. If it is a 0, the mantissa in AQ is shifted left and the exponent decremented.
- Although the low-order half of the mantissa is in Q, we do not use it for the floating-point product. Only the value in the AC is taken as the product.

COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT –III
TOPIC- FLOATING POINT MULTIPLICATION PART-2

Floating Point Multiplication

Example: 3.5×10^2
5.25×10^4

Figure 10-16 Multiplication of floating-point numbers.



b-multiplicand exponent
q-multiplier exponent

Ex : 3.5×10^2
5.25×10^4

① check for zero

$BR = 3.5 \times 10^2 \neq 0 = 011.1 \times 10^2 = BR$

$QR = 5.25 \times 10^4 \neq 0 = 101.01 \times 10^4 = QR$.

go to step ②

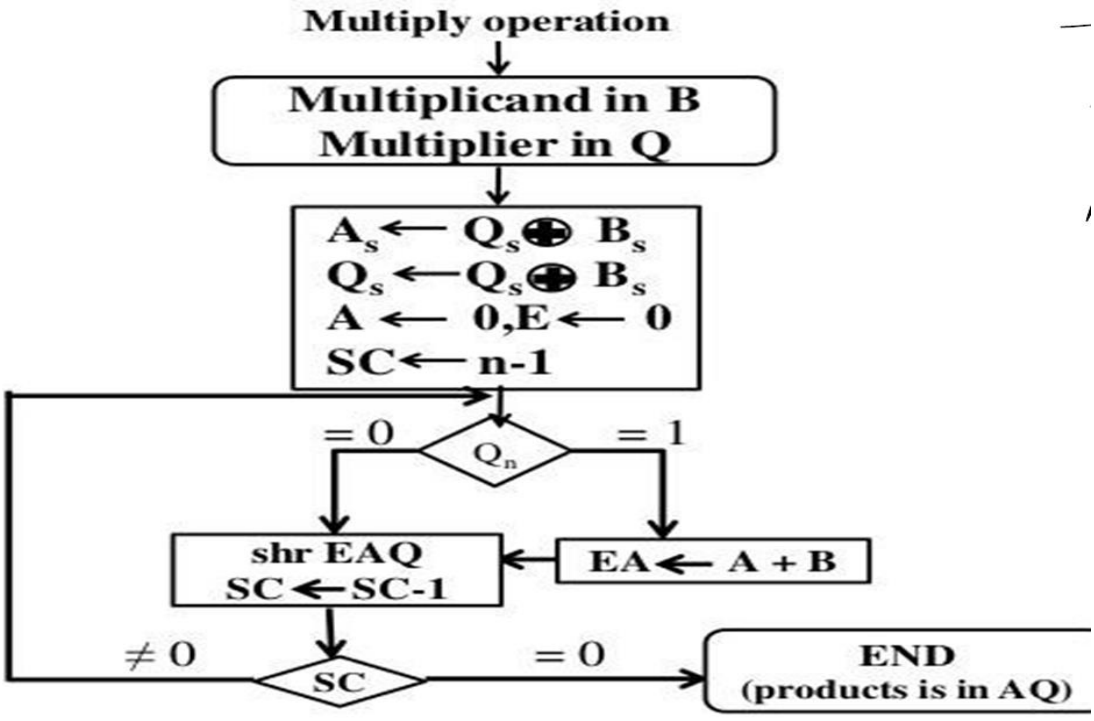
② Add the exponents

$a \leftarrow q \quad b = 10$ (2)

$a \leftarrow 10$ (4) $\quad a \leftarrow a + b = 4 + 2 = 6$.

③ Multiply the mantissa.

Sign magnitude multiplication flowchart



Signed Magnitude Multiplication flowchart.

$A \rightarrow 0,\ E \rightarrow 0,\ SC \leftarrow 5$

$BR = 011 \cdot \underline{1} = 00111$

$QR = 101 \cdot \underline{01} = 10101$

E	A	Q	SC	
	00000	10101 $\rightarrow Q_n$	5	$Q_n = 1$
0	+ 00111			$EA \leftarrow A + B$
	00111	10101	5	
0	00011	11010	4	Shr EAQ, $SC \leftarrow SC - 1$
0				

$SC \neq 0$

0	00011	11010 $\rightarrow Q_n$	4	
0	00001	11101	3	Shr EAQ, $SC \leftarrow SC - 1$

$SC \neq 0$

0	00001	11101 $\rightarrow Q_n$	3	$Q_n = 1$
	+ 00111			$EA \leftarrow A + B$
	01000	11101	3	
0	00100	01110	2	Shr EAQ, $SC \leftarrow SC - 1$

E	A	Q	SC	
0	00100	01110 → Q_n	2	$Q_n=0$
0	00010	00111	1	Shr EAQ SC ← SC-1

SC ≠ 0

0	00010	00111 → Q_n	1	$Q_n=1$ EA ← A+B
	+ 00111			
	01001			
0	01001	00111	1	Shr EAQ SC ← SC-1
0	00100	10011	0	

Since SC = 0, End

$AQ = 00100\,10.011 \times 10^6$

(4) Normalize the result.

$00100\,10.011 \times 10^6$

↳ A_1

underflow

$01001\,00.11 \times 10^5 \Rightarrow$ Normalized result

$\Downarrow$

$AC = 01001\,00.11 \times 10^5$

COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT –III

TOPIC- FLOATING POINT DIVISION PART-1

Floating Point Division

Introduction

• Floating Point division requires that the exponents be subtracted and the mantissa divided.
• The mantissa division is done as in fixed point division.

• The division algorithm can be divided into five parts:

1. Check for zeros.

2. Initialize the registers and evaluate the sign.

3. Align the dividend

4. Subtract the exponents.

5. Divide the mantissa.

Flowchart:

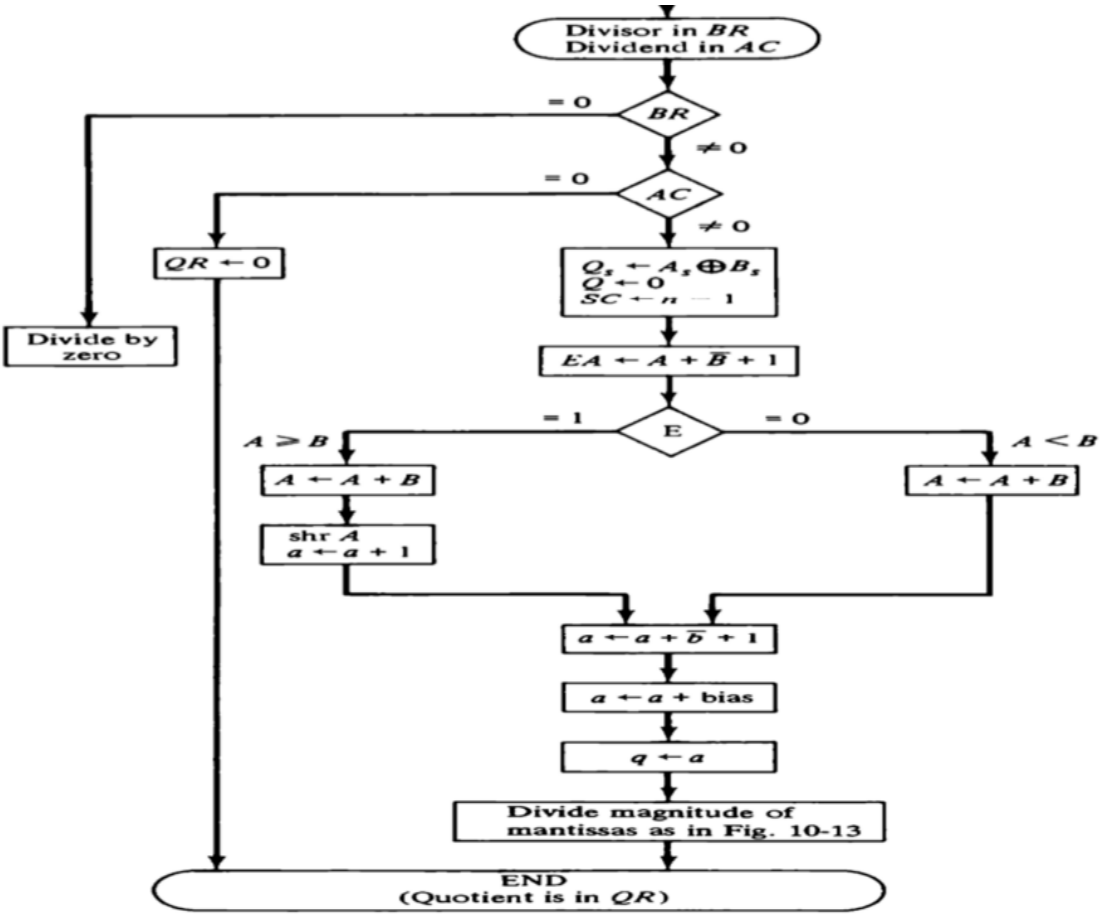


Figure 10-17 Division of floating-point numbers.

1. Check for zeros

If BR = 0, Divide by zero
If AC = 0, Result QR ← 0
If AC=BR <>(Not equal to) 0, goto step 2

- The two operands are checked for zero.
- If the divisor (BR) is zero, it indicates an attempt to divide by zero, which is an illegal operation.
- The operation is terminated with an error message.
- If the dividend (AC) is zero, the quotient QR becomes zero.

2. Initialize the registers and evaluate the sign.

- Determine the sign of the quotient and store it in Q_s .
- The XOR of the sign of the dividend in A_S and the sign of the divisor in Q_s becomes the sign of the quotient.

$$Qs \leftarrow A_s + B_s$$

- The sign of the dividend in A_S, is left unchanged to be the sign of the remainder.
- The Q register is cleared to 0 and the sequence counter sc is set to a number equal to the number of bits in the quotient.

$$Q \leftarrow 0, SC \leftarrow n-1$$

3. Align the dividend (or) check for divide overflow

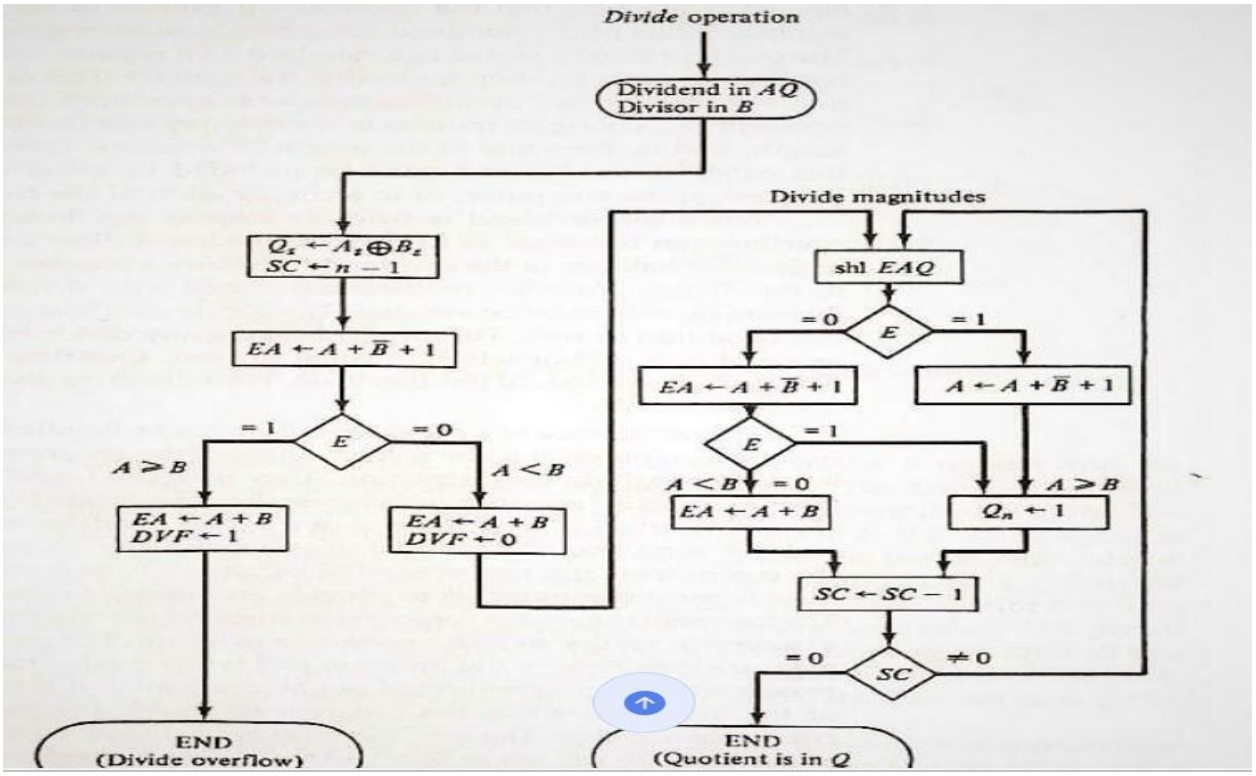
- The dividend alignment is similar to the divide-overflow check in the fixed-point division operation.
- Check if there is an overflow or not.
- To check overflow, subtract the divisor from the dividend.
- If carry =0, no overflow (A< B), restore the dividend.
- If carry =1, overflow (A>= B), perform a shift right on A & increment the exponent and then restore the dividend.

4. Subtract the exponents.

- Next, the divisor exponent is subtracted from the dividend exponent.
- The result transferred into q because the quotient is formed in QR.

5. Divide the mantissa.

- The magnitudes of the mantissas are divided as in the fixed-point division.
- After the operation, the mantissa quotient resides in Q and the remainder in A.(Sign Magnitude Division)
- The floating-point quotient is already normalized and resides in QR.



Flowchart of Signed Magnitude Division

COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT –III

TOPIC- FLOATING POINT DIVISION PART-2

Floating Point Division

Flowchart:

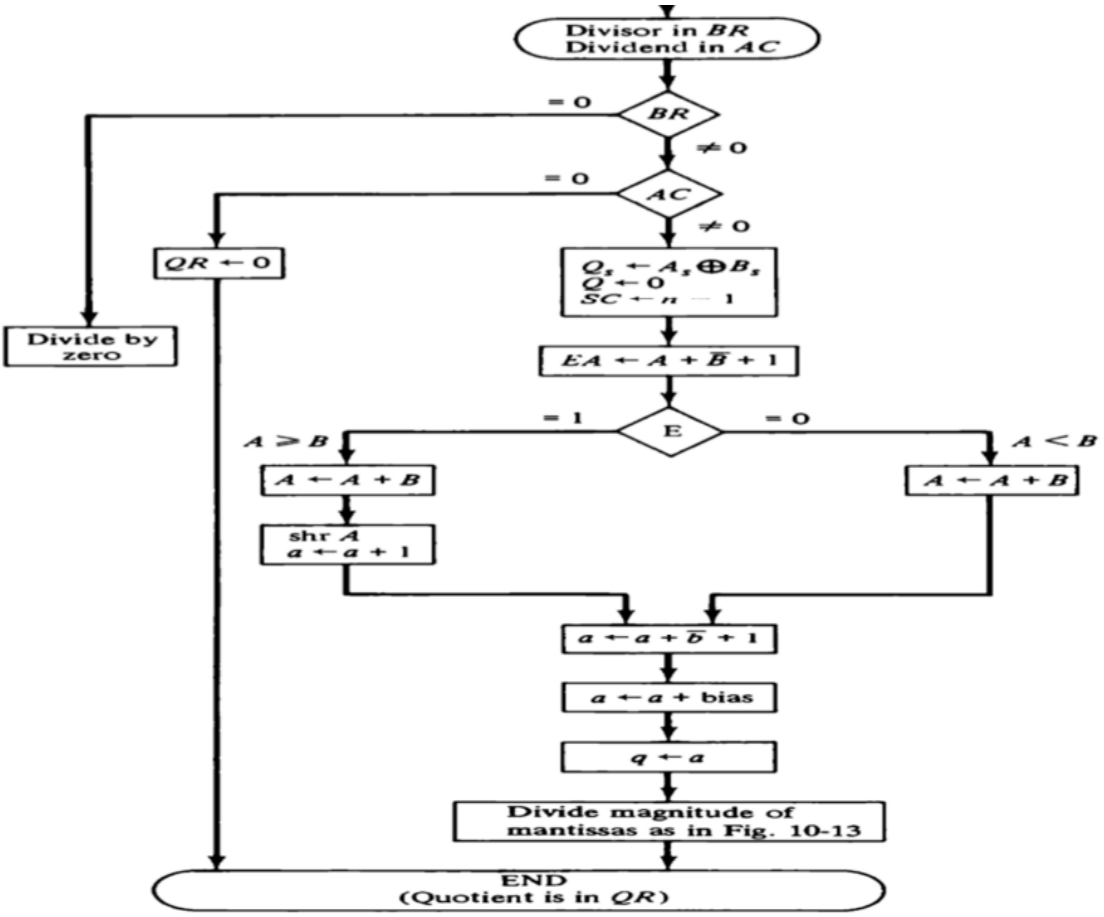


Figure 10-17 Division of floating-point numbers.

Example: $\frac{78.75 \times 10^{5}}{5.25 \times 10^{2}}$

Example

$BR = 5.25 \times 10^2$

$AC = 78.75 \times 10^5$

$$\frac{78.75 \times 10^5}{5.25 \times 10^2}$$

(1) check for zeros.

$AC = 78.75 \times 10^5 \neq 0 = 1001110.11$

$BR = 5.25 \times 10^2 \neq 0 = 101.01$

(2) Initialize the registers & evaluate the sign.

$Q_S \leftarrow A_S \oplus B_S = 0 \oplus 0 = 0.$

$Q \leftarrow 00000, \; SC \leftarrow 5.$

(3) Align dividend (or) check for divide overflow

$AQ = 0100111011$ (A | Q)

$A = 01001$ $\quad B = 10101$

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

$\bar{B} = 01010$ $+$ 1

$\bar{B}+1 = 01011$

$A = 01001$

$\bar{B}+1 = 01011$ $+$

$A \leftarrow 10100$

$E \leftarrow 0$

$A = 10100$

$B = 10101$ $+$

$E \rightarrow 1\ 01000$

No carry; No divide overflow. $\rightarrow A < B$

(4) Subtract the exponents.

$a = 5, \; b = 2.$

$a \leftarrow a + \bar{b} + 1$

$a = 101$

$\bar{b}+1 = 110$ $+$

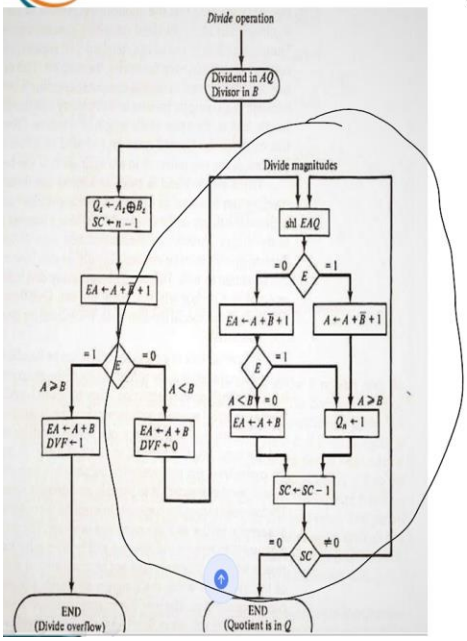
$1\,011 \rightarrow a = 3 \Rightarrow q$

$a = 101$

$b = 010$

$\bar{b} = 101$ $+$ 1

$\bar{b}+1 = 110$



(5) Divide the mantissa

E	A	Q	SC
1	01001	11011	5

$A = 00000$

$Q = 01111 = +15$