UNIT 3:

Data Representation: Data types, Complements, Fixed Point Representation, Floating Point Representation.

Computer Arithmetic: Addition and subtraction, multiplication Algorithms, Division Algorithms, Floating–point Arithmetic operations. Decimal Arithmetic unit, Decimal Arithmetic operations.

BASIC COMPUTER DATA TYPES:

• Binary information in digital computers is stored in memory or processor registers.

• 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.

• A number system of base or radix r is a system of that uses distinct symbols for r digits

• The decimal number system in everyday use employs radix 10 system. The 10 symbols are 0, 1, 2, 3, 4, 5, 6,7, 8, and 9; highest number being r-1

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

• The string of digits 101101 is interpreted to represent 1 x 25 + 0 x 24 + 1 x 23 + 1 x 22 + 0 x 21 + 1 x 20 = 45

• Besides the decimal and binary number systems,

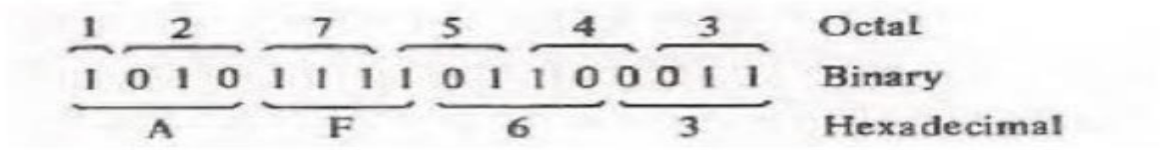
• Octal (radix 8)- 0,1,2,3,4,5,6,7 and Hexadecimal (radix 16) – 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F • Octal can be converted to decimal as follows (736.4)8 = 7 X 82 + 3 X 81 + 6 X 80 + 4X8-1 = 7X64 + 3X8 + 6X1 + 4/8 = (478.5) 10

• **Basic computer data types**

• Hexadeciamal can be converted to decimal

$$(F3)_{16} = F \times 16 + 3$$
$$= 15 \times 16 + 3$$
$$= (243)_{10}$$

Octal and Hex can be obtained from Binary as shown below:



Complements:

• A binary code is a group of n bits that assume upto 2^n distinct combinations of 0s and 1s.

• A BCD code is a binary coded decimal i.e. binary coding decimal numbers.

• ASCII (American Standard Code for Information Interchange),

• which uses seven bits to code 128 characters is standard alphanumeric character code.

• Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. There are two types of complements

• For each base r system: the r's complement and the (r - 1)'s Complement

For binary base 2 system: the 2's complement and the 1's complement • For decimal base 10 system: the 10's complement and the 9's complement • The 9's complement of 546700 is 999999 - 546700 = 453299 • The 9's complement of 12389 is 99999 - 12389 = 87610 • The 1's complement of a binary number is formed by Changing 1's into 0's and 0's into 1's. • For example, the 1's complement of 1011001 is 0100110 and the 1's complement of 0001111 is 1110000.

The 10's complement of the decimal 2389 is 7610 + 1 = 7611 and is obtained by adding 1 to the 9's complement value.

• The 2's complement of binary 101100 is 010011 + 1 = 010100 and is obtained by adding 1 to the 1's complement value.

• For example, the 1's complement of 1011001 is 0100110 and the 1's complement of 0001111 is 1110000.

Solve: Find the 9's and 10's complement of 246700. Find the 1's and 2's complement of 1101100.

Fixed-Point Representation

• In computer binary systems it is customary to represent the sign with a bit placed in the leftmost position of the number.

• sign bit is equal to 0 for positive and to 1 for negative.

• a number may have a binary (or decimal) point.

• There are two ways of specifying the position of the binary point: by giving it a fixed position or by employing a floatingpoint representation.

• The fixed-point method assumes that the binary point is always. fixed in one position.

• The two positions most widely used are (1) a binary point in the extreme left of the register to make the stored number a fraction, and (2) a binary point in the extreme right of the register to make the stored number an integer. In either case, the binary point is not actually present.

Integer Representation for signed numbers

• signed-magnitude representation 1 0001110

• signed-1's complement representation 1 1110001

• signed-2's complement representation 1 1110010

Floating Point Representation:

The floating-point representation uses a second register to store a number that designates the position of the decimal point in the first register. • The floating-point representation of a number has two parts. The first part represents a signed, fixed-point number called the mantissa. The second part designates the position of the decimal (or binary) point and is called the exponent The fixedpoint mantissa may be a fraction or an 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**

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 00010

Sign-Magnitude

used in every day arithmetic calculations .

Left most bit is sign bit- **0 – positive 1 – negative**

• +18 = **00010010**

• -18 = **10010010**

• Problems

—**Need to consider both sign and magnitude in**

arithmetic

—**Two representations of zero (+0 and -0)**

Addition and Subtraction:

Normal binary addition

• Monitor sign bit for overflow

• So we only need addition and complement

Circuits

Assume:-

-magnitudes of two numbers as A and B.

when those sign numbers are added we have eight different conditions depending on the sign bits and operations performed.

SIGNED MAGNITUDE ADDITION AND SUBTRACTION

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

Case 1: addition

- **If signs are same:**
- 3+5 both numbers positive
- (-3) + (-5) both numbers negative -(3+5)
- **If signs differ:**
- (+3) + (-5) numbers differ in sign so, 3-5 **(results in subtraction operation)**
- (-3) + (+5) entin signs so, -(3-5)
- **(results in subtraction operation)**

Case 2: Subtraction

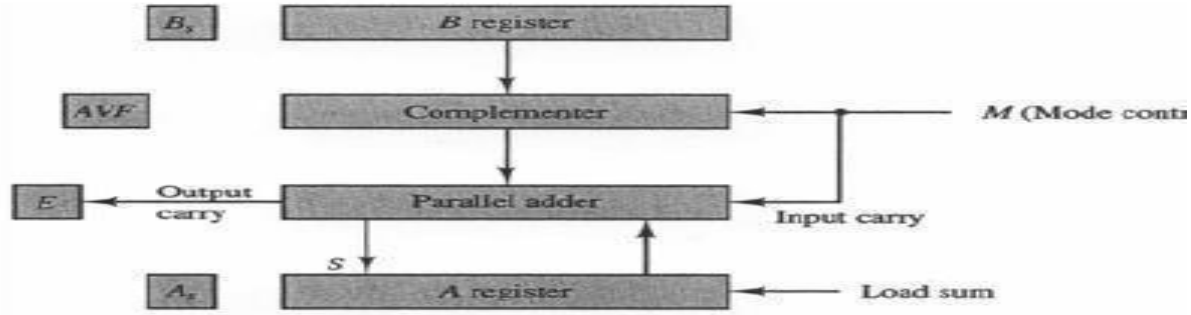
- **If signs are same:**
- 3-5 both numbers positive
- (-3) - (-5) both numbers negative -(3-5)
- **If signs differ:**
- (+3) - (-5) numbers differ in sign so, 3+5 **(results in addition operation)**
- (-3) -(+5) entin signs so, -(3+5)
- **(results in addition operation)**

Add / Subtract Signed-Magnitude

Operation	Add Magnitudes	Subtract Magnitudes		
		When $A > B$	When $A < B$	When $A = B$
$(+A) + (+B)$	$+(A + B)$			
$(+A) + (-B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(-A) + (+B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$
$(-A) + (-B)$	$-(A + B)$			
$(+A) - (+B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(+A) - (-B)$	$+(A + B)$			
$(-A) - (+B)$	$-(A + B)$			
$(-A) - (-B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$

Forces zero to be positive

Hardware



- A_S Sign of A
- B_S Sign of B
- A_S & A Accumulator
- AVF Overflow bit for $A + B$
- E Output carry for parallel adder

ALGORITHM:

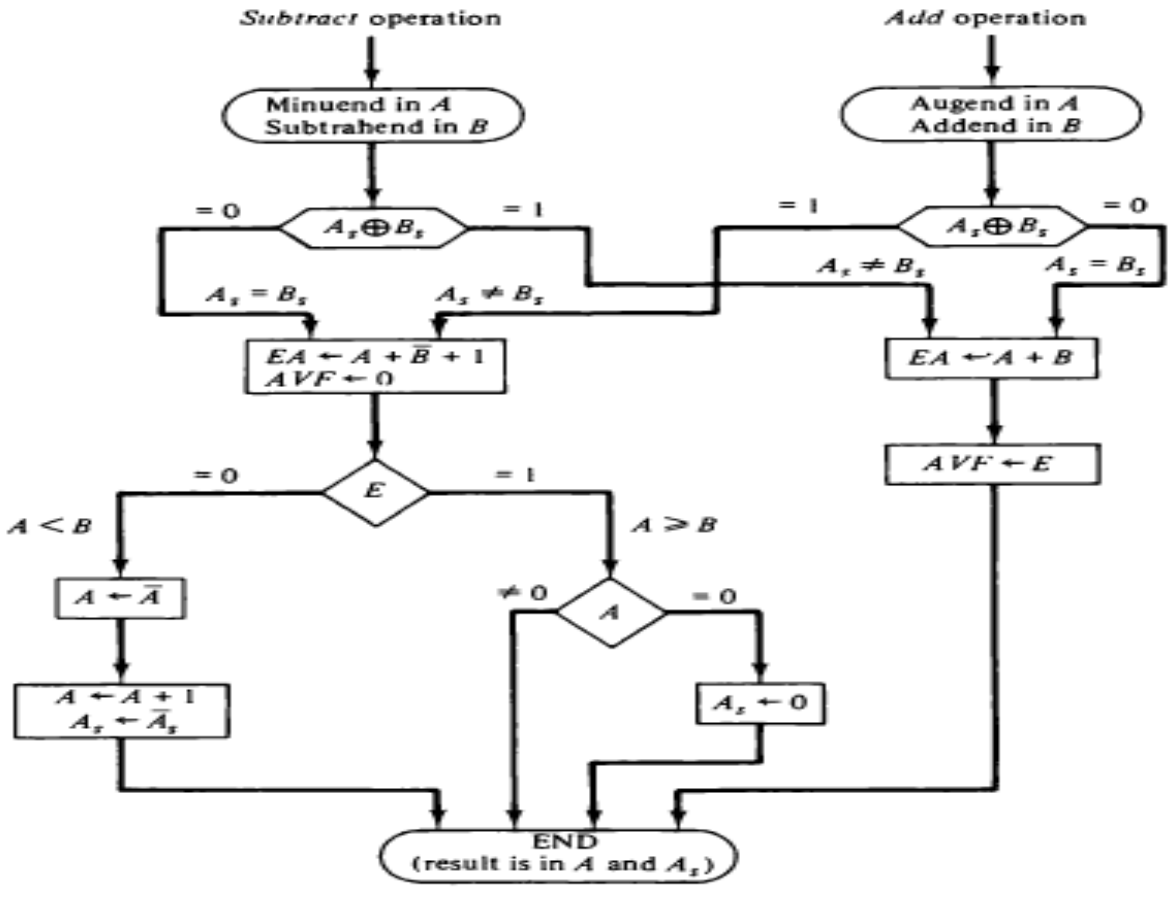
The two signs As and Bs are compared by EX-OR them. If result is 0 then As = Bs and if result is 1 the As ≠ Bs.

o For add operations if have same sign bits the magnitude must be added. For subtract operations different sign bits means magnitudes be added as well.

o E bit is carry bit after addition and moves to AVE overflow bit only at this state.

o If sign bits are different in add operations or the same in subtract operations the two magnitudes will be subtracted A – B. No overflow can occur here.

o After subtract if E=1 this means A>B and if E=0 then A<B. then here it is necessary to get 2's complement of A (by invert A then add 1) and sign of A is inverted only in this case.



SIGNED 2'S COMPLEMENT ADDITION AND SUBTRACTION:

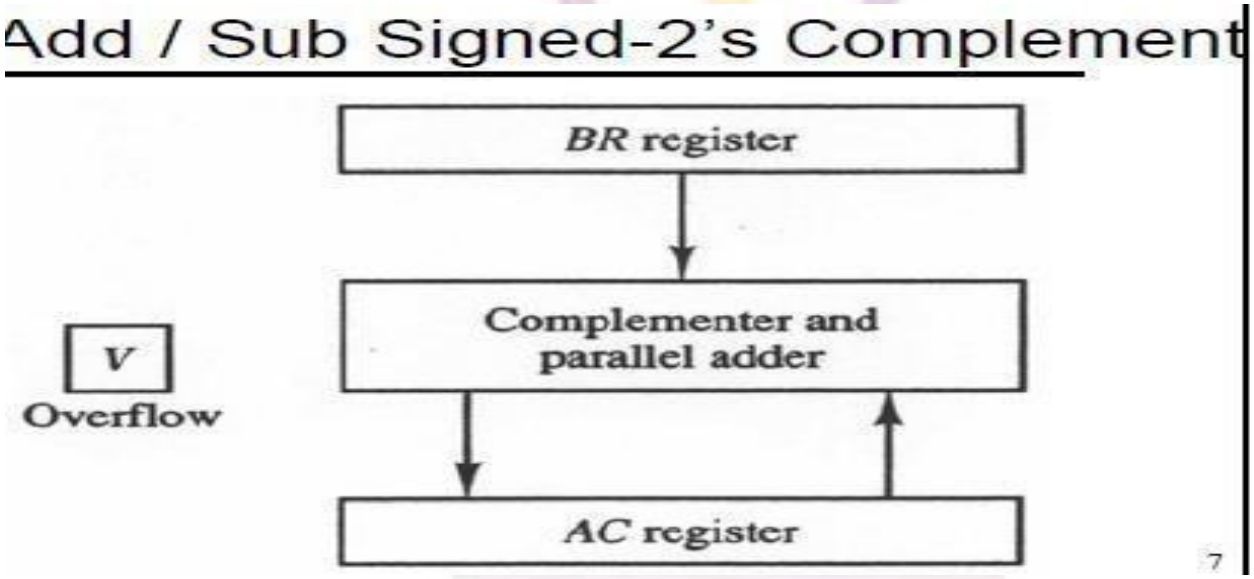
The left most bit in 2's complement represented binary number is the sign bit. If 0 the number is positive and if 1 then number is negative. If sign bit is 1 the entire number is represented in 2's complement.

The addition of two numbers represented in 2's complement is carried out by normal binary addition with carry discarded.

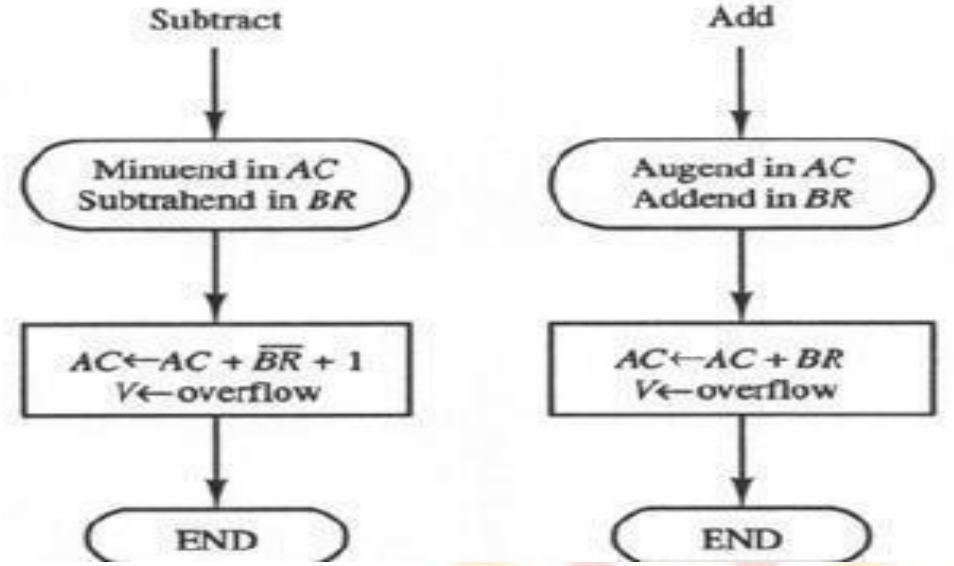
The subtraction is carried out by taking 2's complement (B) of subtrahend and adding it to minuend (A).

Overflow can be detected by inspecting last 2 carries out of addition by EX-OR them. If different then overflow is detected.

For addition simply implement add then see overflow. For subtract add 2's complement of B to A and watch overflow since the A and –B could be of same sign.



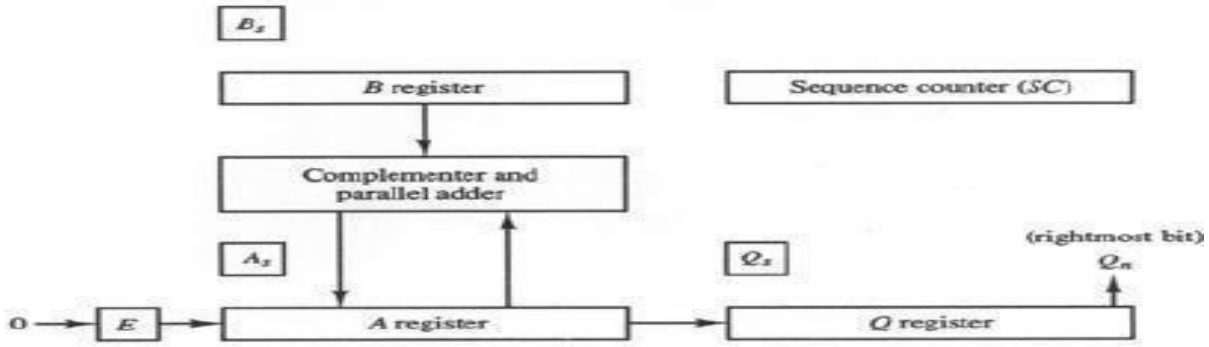
Algorithm



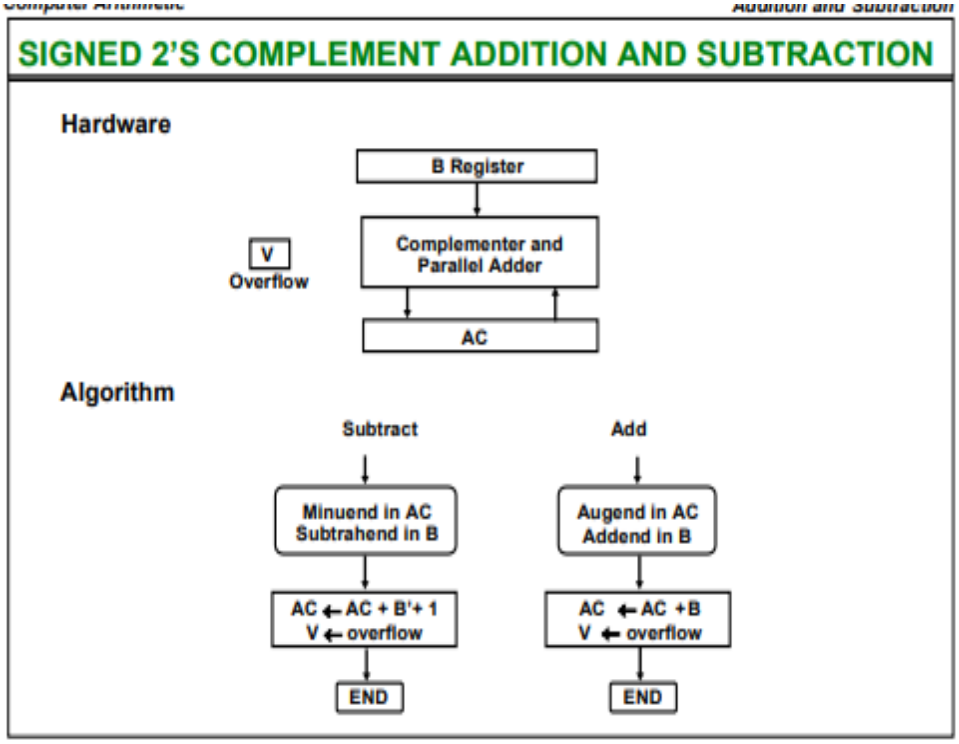
Multiply Signed-Magnitude

- Series of successive shift and add operations
- 23 10111 Multiplicand
  19 x 10011 Multiplier
  10111
  10111
  00000
  00000
  10111 +
  437 110110101 Product

Hardware



- Q multiplier
- B multiplicand
- A 0
- SC number of bits in multiplier
- E overflow bit for A
- Do SC times
  - If low-order bit of Q is 1
    - $A \leftarrow A + B$
  - Shift right EAQ
- Product is in AQ


SIGNED 2'S COMPLEMENT ADDITION AND SUBTRACTION
Hardware
B Register
Complementer and Parallel Adder
V
Overflow
AC
Algorithm
Subtract
Minuend in AC
Subtrahend in B
AC ← AC + B'+ 1
V ← overflow
END
Add
Augend in AC
Addend in B
AC ← AC +B
V ← overflow
END

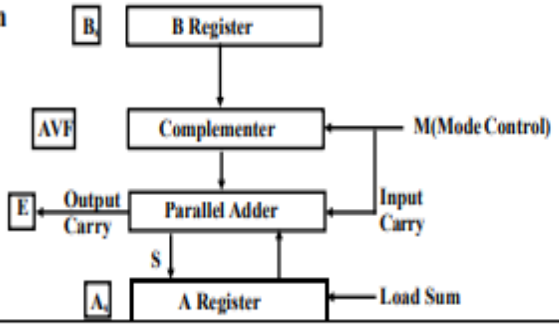
SIGNED MAGNITUDEADDITION AND SUBTRACTION

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)

Hardware Implementation



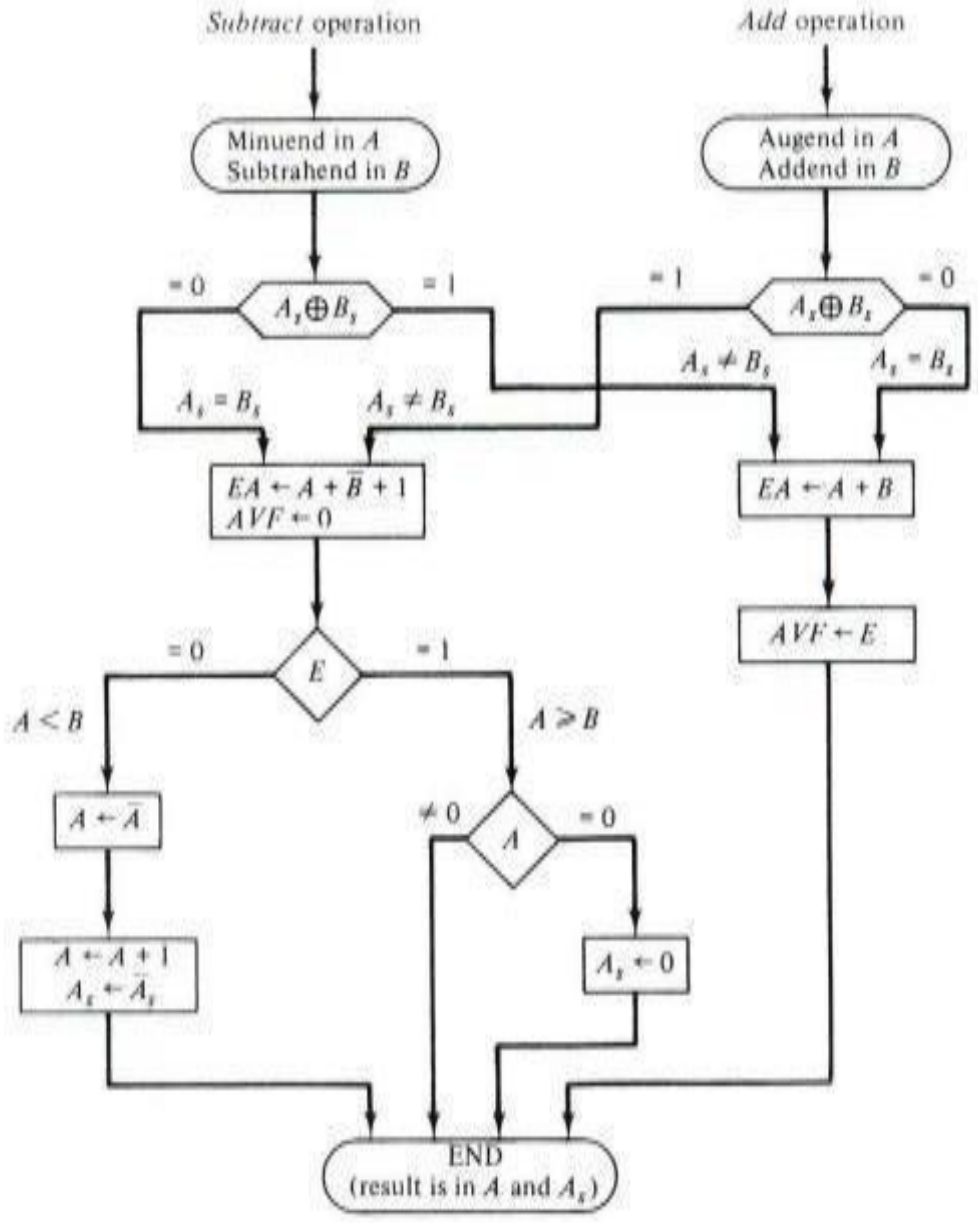
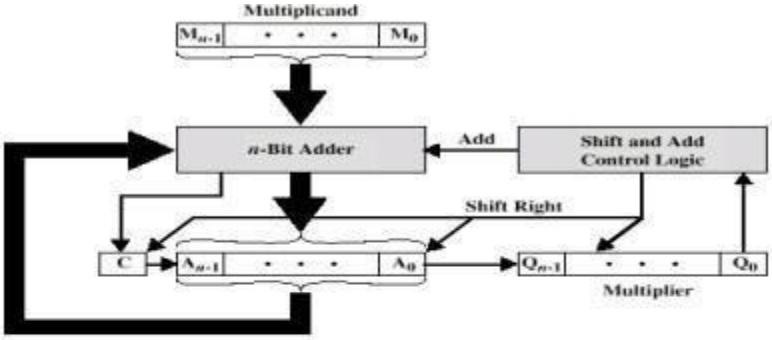


Figure 10-2 Flowchart for add and subtract operations.

Multiplication Algorithm: In the beginning, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in Bs and Qs 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. Since an operand must be stored with its sign, one bit of the word will be occupied by the sign and the magnitude will consist of n-1 bits. Now, the low order bit of the multiplier in Qn is

tested. If it is 1, the multiplicand (B) is added to present partial product (A), 0 otherwise. 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.



C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

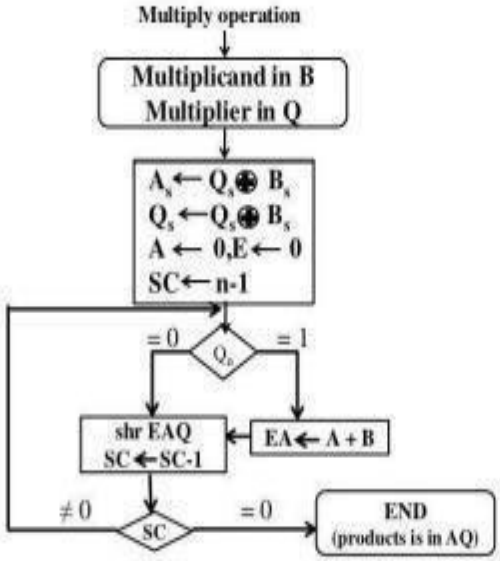


Figure: Flowchart for multiply operation.

Booth's algorithm :

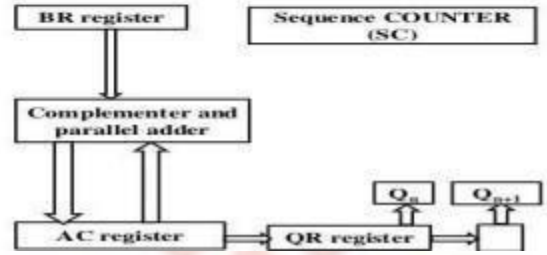
- Booth algorithm gives a procedure for multiplying binary integers in signed- 2's complement representation.

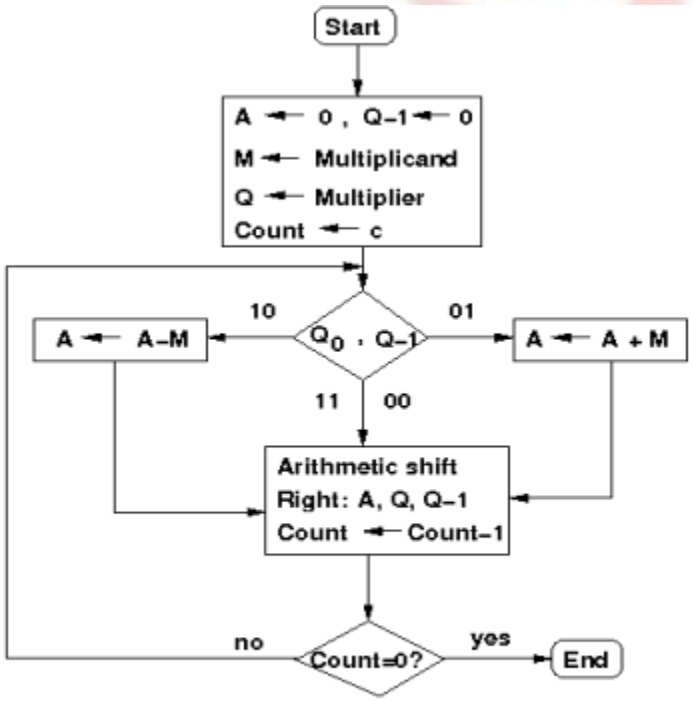
shifting, and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as $2^{k+1} - 2^m$.

- For example, the binary number 001110 (+14) has a string 1's from 2^3 to 2^1 (k=3, m=1). The number can be represented as $2^{k+1} - 2^m$. = $2^4 - 2^1 = 16 - 2 = 14$. Therefore, the multiplication M X 14, where M is the multiplicand and 14 the multiplier, can be done as M X 2^4 – M X 2^1.
- Thus the product can be obtained by shifting the binary multiplicand M four times to the left and subtracting M shifted left once.

Hardware for Booth Algorithm

- Sign bits are not separated from the rest of the registers
- rename registers A,B, and Q as AC,BR and QR respectively
- Q_n designates the least significant bit of the multiplier in register QR
- Flip-flop Qn+1 is appended to QR to facilitate a double

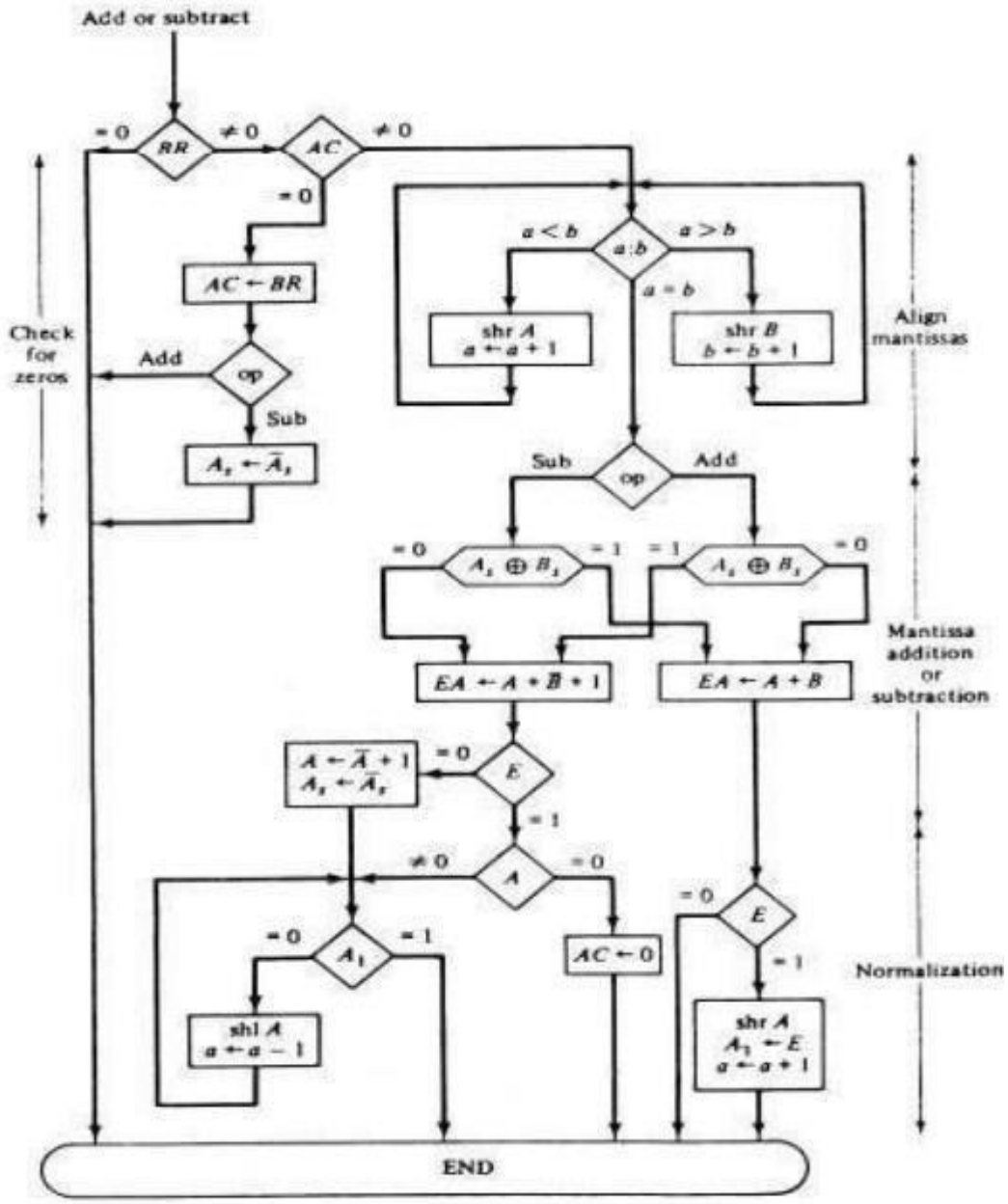




- As in all multiplication schemes, booth algorithm requires examination of the multiplier bits and shifting of partial product.

Division of two fixed-point binary numbers in signed magnitude representation is performed with paper and pencil by a process of successive compare, shift and subtract operations. Binary division is much simpler than decimal division because here the quotient digits are either 0 or 1 and there is no need to estimate how many times the dividend or partial remainder fits into the divisor. The division process is described in Figure.

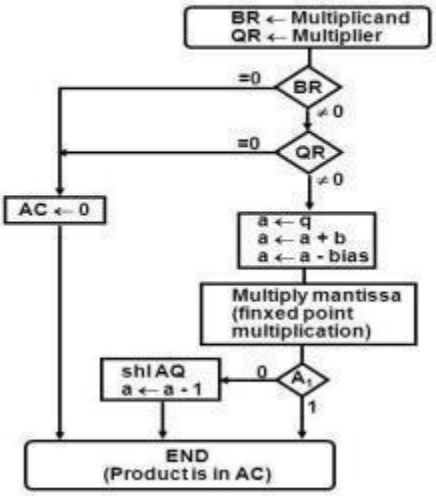
The devisor is compared with the five most significant bits of the dividend. Since the 5-bit number is smaller than B, we again repeat the same process. Now the 6-bit number is greater than B, so we place a 1 for the quotient bit in the sixth position above the dividend. Now we shift the divisor once to the right and subtract it from the dividend. The difference is known as a partial remainder because the division could have stopped here to obtain a quotient of 1 and a remainder equal to the partial remainder. Comparing a partial remainder with the divisor continues the process. If the partial remainder is greater than or equal to the divisor, the quotient bit is equal to 1. The divisor is then shifted right and subtracted from the partial remainder. If the partial remainder is smaller than the divisor, the quotient bit is 0 and no subtraction is needed. The divisor is shifted once to the right in any case. Obviously the result gives both a quotient and a remainder.


Add or subtract
BR
AC
= 0
≠ 0
AC ← BR
Check for zeros
Add
Sub
op
A_s ← Ā_s
a < b
a : b
a > b
a = b
shr A
a ← a + 1
shr B
b ← b + 1
Align mantissas
A_i ⊕ B_i
EA ← A + B̄ + 1
EA ← A + B
A ← Ā + 1
E
A
A_1
AC ← 0
shl A
a ← a − 1
shr A
A_1 ← E
a ← a + 1
Mantissa addition or subtraction
Normalization
END

Multiplication:

Computer Arithmetic 16 Floating Point Arithmetic

FLOATING POINT MULTIPLICATION



Computer Organization Prof. H. Yoon

Computer Arithmetic 17 Floating Point Arithmetic

FLOATING POINT DIVISION