UNIT-IV

COMPUTER ARITHMETIC

Introduction:

Data is manipulated by using the arithmetic instructions in digital computers. Data is manipulated to produce results necessary to give solution for the computation problems. The Addition, subtraction, multiplication and division are the four basic arithmetic operations. If we want then we can derive other operations by using these four operations.

To execute arithmetic operations there is a separate section called arithmetic processing unit in central processing unit. The arithmetic instructions are performed generally on binary or decimal data. Fixed-point numbers are used to represent integers or fractions. We can have signed or unsigned negative numbers. Fixed-point addition is the simplest arithmetic operation.

If we want to solve a problem then we use a sequence of well-defined steps. These steps are collectively called algorithm. To solve various problems we give algorithms.

In order to solve the computational problems, arithmetic instructions are used in digital computers that manipulate data. These instructions perform arithmetic calculations.

And these instructions perform a great activity in processing data in a digital computer. As we already stated that with the four basic arithmetic operations addition, subtraction, multiplication and division, it is possible to derive other arithmetic operations and solve scientific problems by means of numerical analysis methods.

A processor has an arithmetic processor(as a sub part of it) that executes arithmetic operations. The data type, assumed to reside in processor, registers during the execution of an arithmetic instruction. Negative numbers may be in a signed magnitude or signed complement representation. 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 magnitude representation for the mantissa.

Addition and Subtraction :

Addition and Subtraction with 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. These conditions are listed in the first column of Table 4.1. The other columns in the table show the actual operation to be performed with the magnitude of the numbers. The last column is needed to present a negative zero. In other words, when two equal numbers are subtracted, the result should be +0 not -0.

The algorithms for addition and subtraction are derived from the table and can be stated as follows (the words parentheses should be used for the subtraction algorithm)

Addition and Subtraction of Signed-Magnitude Numbers

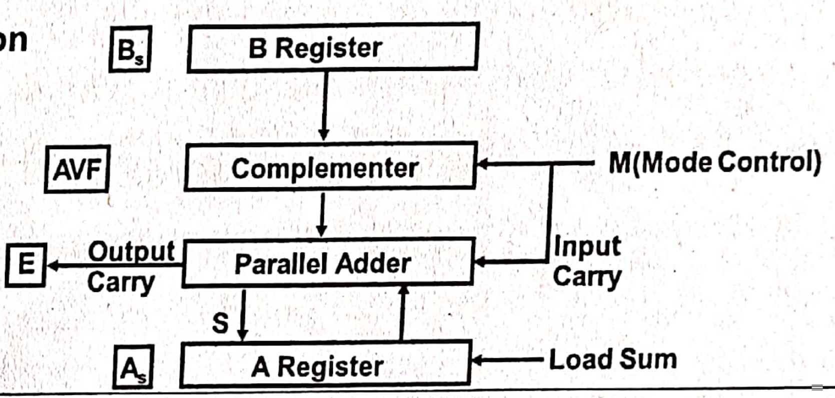
SIGNED MAGNITUDEADDITION AND SUBTRACTION

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)

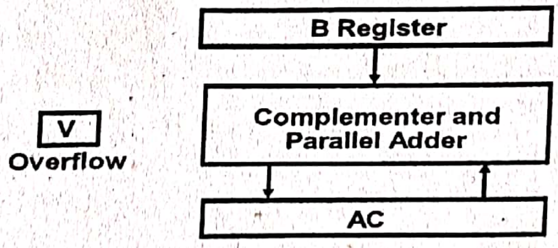
Hardware Implementation

Add overflow flip flop

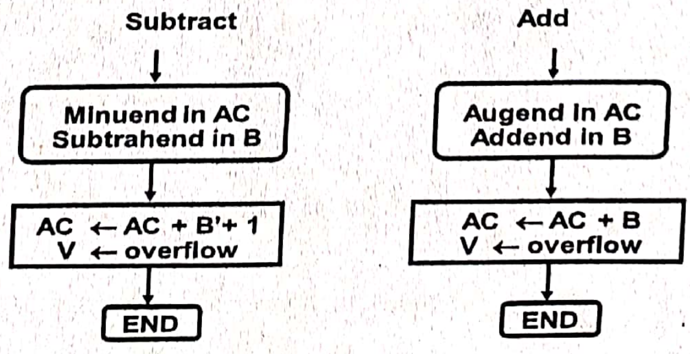


SIGNED 2'S COMPLEMENT ADDITION AND SUBTRACTION

Hardware



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

Algorithm:

□ The flowchart is shown in Figure 7.1. The two signs A, and B, 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, 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 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 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.

□ 1 in E indicates that A >= B and the number in A is the correct result. If this numbs is zero, the sign A must be made positive to avoid a negative zero.

□ 0 in E indicates that A < B. For this case it is necessary to take the 2's complement of the value in A. The operation can be done with one microoperation A□ A' +1.

□ However, we assume that the A register has circuits for microoperations complement and increment, so the 2's complement is obtained from these two microoperations.

□ In other paths of the flowchart, the sign of the result is the same as the sign of A. so no change in A is required. However, when A < B, the sign of the result is the complement of the original sign of A. It is then necessary to complement A, to obtain the correct sign.

□ The final result is found in register A and its sign in As. The value in AVF provides an overflow indication. The final value of E is immaterial.

□ Figure 7.2 shows a block diagram of the hardware for implementing the 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 two numbers.

□ The add-overflow flip-flop *AVF* holds the overflow bit when A and B are added.

□ The A register provides other microoperations that may be needed when we specify the sequence of steps in the algorithm.

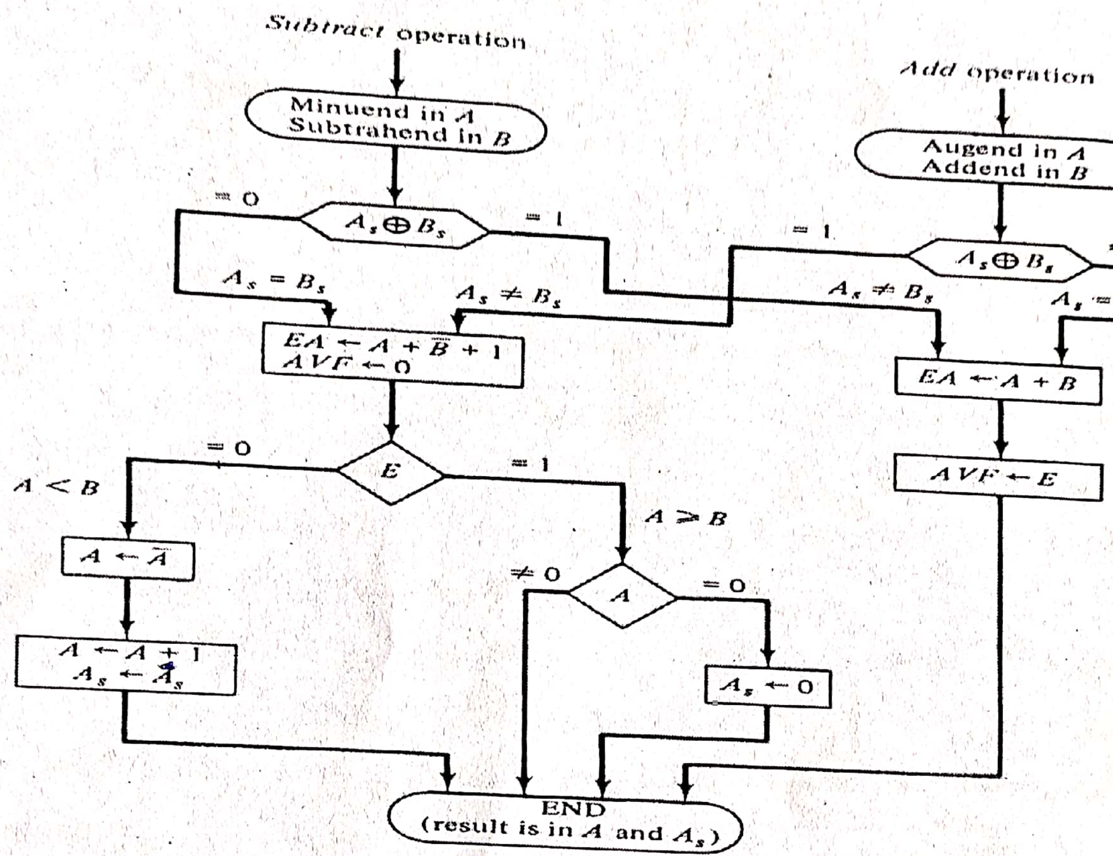


Figure 10-2 Flowchart for add and subtract operations.

XOR

0 0 → 0
0 1 → 1
1 0 → 1
1 1 → 0

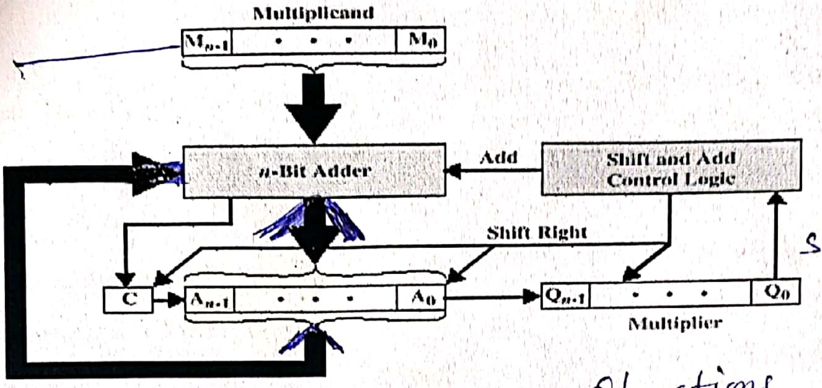
Multiplication Algorithm:

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.

11 X 13 = 143

M Q AQ



shift last bit

Operations

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

13 13

right CAQ

$Q_0 = 0$ lost

Add M with A A=A+M

If $Q_0 = 1$, we have to perform addition

– Since $Q_0 = 1$ ∴ A+M

⇒ 0010
1011
1101

Carry

– $Q_0 = 1$

∴ 0110+1011=①0001

1 lost

128 15 =143

Multiplication using add shift method

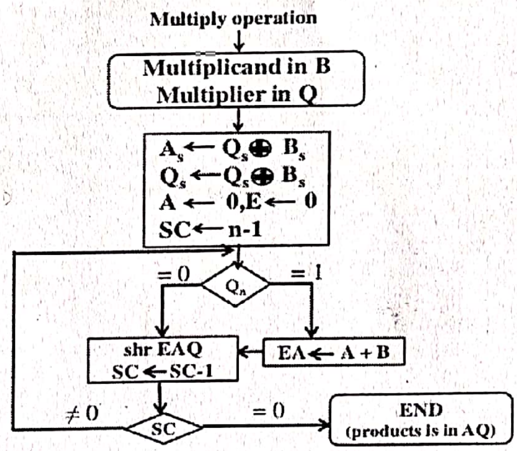


Figure: Flowchart for multiply operation.

Booth's algorithm :

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

□ It operates on the fact that strings of 0's in the multiplier require no addition but just 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.

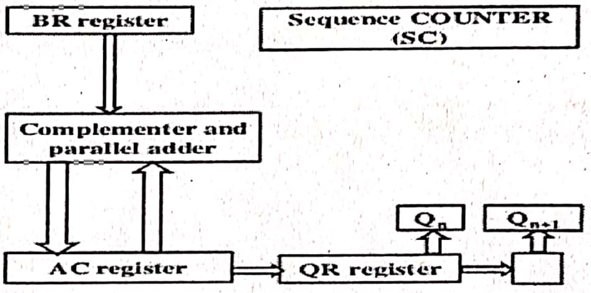
Hardware for Booth Algorithm

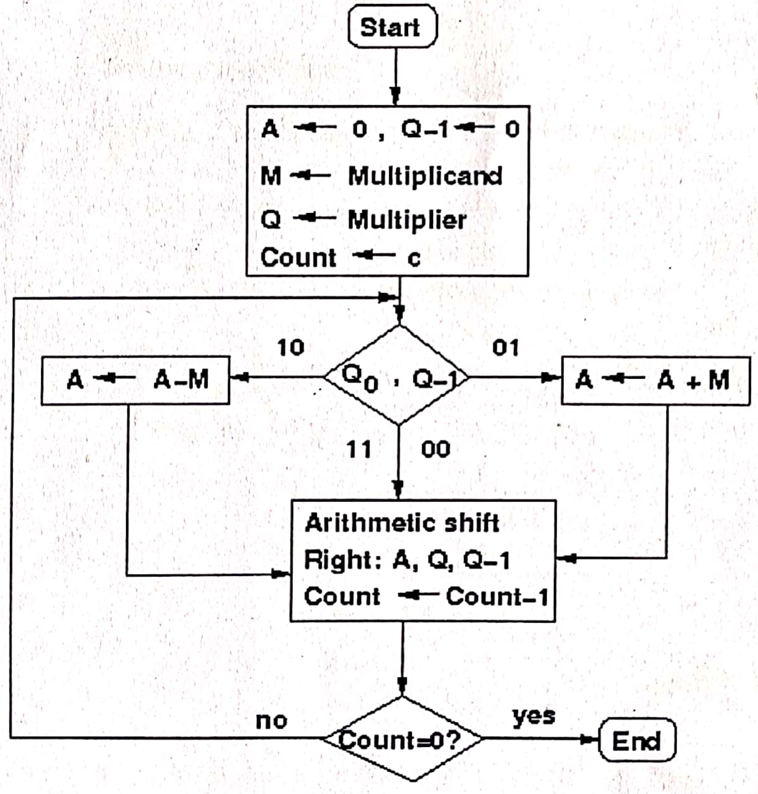
➢ Sign bits are not separated from the rest of the registers

➢ 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 bit inspection of the multiplier





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

□ Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial, or left unchanged according to the following rules:

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

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

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

□ The algorithm works for positive or negative multipliers in 2's complement representation.

□ This is because a negative multiplier ends with a string of 1's and the last operation will be a subtraction of the appropriate weight.

□ The two bits of the multiplier in Qn and Qn+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 the two bits are equal, the partial product does not change.