Course Contents

Unit-05: Fixed point Computer Arithmetic

Addition and Subtraction, Multiplication, Division Algorithm

Unit 5	Fixed point Computer Arithmetic	5 Hours
5.1	Signed number Representation: Signed Magnitude, 1's Complement and 2's Complement Form	0.5 Hour
5.2	Addition and Subtraction(with Numerical Example), Addition and Subtraction with Signed Magnitude Data, Hardware implementation, Hardware Algorithm, Addition and Subtraction with Signed 2's Complemented Data	2.5 Hours
5.3	Multiplication Algorithm: Hardware Implementation for Signed Magnitude Data, Booth Multiplication Algorithm (with Numerical Example)	1 Hour
5.4	Division Algorithm: Hardware Implementation for Signed Magnitude Data, Hardware Algorithm (Restoring Only)	1 Hour

Course Contents

Unit-05: Fixed point Computer Arithmetic

5.1 Signed number Representation: Signed Magnitude, 1's Complement and 2's Complement Form

Course Contents

Unit-05: Fixed point Computer Arithmetic

- Arithmetic instructions in digital computers manipulate data to produce results necessary for the solution of computational problems.
- These instructions perform arithmetic calculations and are responsible for the bulk of activity involved in processing data in a computer.
- The four basic arithmetic operations are addition, subtraction, multiplication and division.

Course Contents

Unit-05: Fixed point Computer Arithmetic

- From these four basic operations, it is possible to formulate other arithmetic functions and solve scientific problems by means of numerical analysis methods.
- Here, we will discuss these 4 operations only on fixed-point binary data (there are others types too, viz, floating point binary data, binary-coded decimal data) and hence the unit is named "**Fixed point Computer Arithmetic**"

Course Contents

Unit-05: Fixed point Computer Arithmetic

1. Signed number Representation:

A signed integer is an integer with a positive '+' or negative sign '-' associated with it. Since the computer only understands binary, it is necessary to represent these signed integers in binary form. In binary, signed Integer can be represented in three ways:

1. **Signed Magnitude,**
2. **1's Complement and**
3. **2's Complement Form**

Let us see why 2's complement is considered to be the best method.

Course Contents

Unit-05: Fixed point Computer Arithmetic

1. Signed bit Representation:

In the signed integer representation method, the following rules are followed:

- The MSB (Most Significant Bit) represents the sign of the Integer.
- Magnitude is represented by other bits other than MSB i.e. (n-1) bits where n is the no. of bits.
- If the number is positive, MSB is 0 else 1.
- The range of signed integer representation of an n-bit number is given as $-(2^{n-1}-1)$ to $(2^{n-1}-1)$.

Course Contents

Unit-05: Fixed point Computer Arithmetic

1. Signed bit Representation:

Example:

Let n = 4

- Range:
  $-(2^{4-1}-1)$ to $2^{4-1}-1$
  $= -(2^3-1$ to $2^3-1)$
  $= -(7)$ to $+7$
- For 4 bit representation, minimum value=-7 and maximum value=+7

Course Contents

Unit-05: Fixed point Computer Arithmetic

5.2 Addition and Subtraction(with Numerical Example), Addition and Subtraction with Signed Magnitude Data, Hardware implementation, Hardware Algorithm, Addition and Subtraction with Signed 2's Complemented Data

Unit-05: Fixed point Computer Arithmetic

Addition and Subtraction:

- There are three ways of representing negative fixed-point binary numbers: signed-magnitude, signed-1's complement, or signed-2's complement. Most computers use the signed-2's complement representation when performing arithmetic operations with integers.
- In our course, we study the addition and subtraction algorithms and hardware implementation for data represented in signed-magnitude and signed-2's complement.
- It is important to realize that the adopted representation for negative numbers refers to the representation of numbers in the registers before and after the execution of the arithmetic operation. It does not mean that complement arithmetic may not be used in an intermediate step.

Addition and Subtraction with Signed Magnitude Data:

- When the signed numbers (A and B) 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.
- 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 prevent a negative zero (i.e. when two equal numbers are subtracted, the result should be +0 not -0).

TABLE 10-1 Addition and Subtraction of Signed-Magnitude Numbers

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

The algorithms for addition and subtraction are:

Note: Bracket ( ) for Subtraction

- When the signs of A and B are **identical** (**different**), add the two magnitudes and attach the sign of A to the result.
- When the signs of A and B are **different** (**identical**), compare the magnitudes and subtract the smaller number from the larger. 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 positive.

The two algorithms are similar except for the sign comparison. The procedure to be followed for identical signs in the addition algorithm is the same as for different signs in the subtraction algorithm, and vice versa.

Unit-05: Fixed point Computer Arithmetic

Addition and Subtraction with Signed Magnitude Data:

TABLE 10-1 Addition and Subtraction of Signed-Magnitude Numbers

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

The algorithms for addition and subtraction are:
Note: Bracket ( ) for Subtraction

- When the signs of A and B are **identical** (**different**), add the two magnitudes and attach the sign of A to the result.
- When the signs of A and B are **different** (**identical**), compare the magnitudes and subtract the smaller number from the larger. 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 positive.

The two algorithms are similar except for the sign comparison. The procedure to be followed for identical signs in the addition algorithm is the same as for different signs in the subtraction algorithm, and vice versa.

Unit-05: Fixed point Computer Arithmetic

Addition and Subtraction with Signed Magnitude Data:

TABLE 10-1 Addition and Subtraction of Signed-Magnitude Numbers

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

Unit-05: Fixed point Computer Arithmetic

Addition and Subtraction with Signed Magnitude Data:

TABLE 10-1 Addition and Subtraction of Signed-Magnitude Numbers

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

Exercises:

a. (+35) + (+40)
b. (-35) + (-40)
c. (-35) + (+40)
d. (+35) + (-40)
e. (-35) - (+40)
f. (-35) - (+40)
g. (-35) - (-40)
h. (+40) - (+35)

A +4=0100
B - 3=1101
---
1 0001

A +3=0011
B - 4=1100
---
0 1111

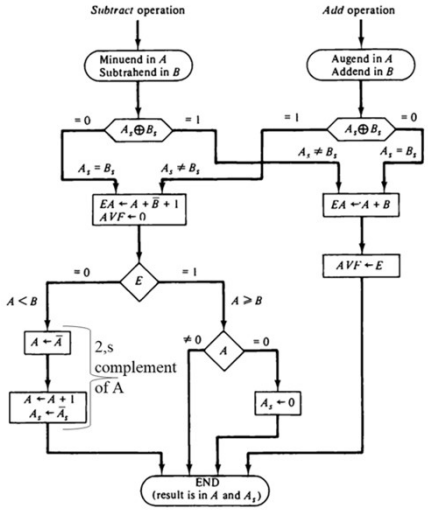


Figure 10-2 Flowchart for add and subtract operations.

Unit-05: Fixed point Computer Arithmetic

Addition and subtraction Algorithm:

- A_s and B_s are compared by an exclusive-OR gate. If output=0, signs are identical, if 1 signs are different.
- For add operation identical signs dictate addition of magnitudes and for subtraction, different magnitudes dictate magnitudes be **added**. Magnitudes are added with a microoperation EA ←A+B (EA is a register that combines A and E). If E=1, overflow occurs and is transferred to AVF (add-overflow flip-flop).
- Two magnitudes are subtracted if signs are different for add operation and identical for subtract operation. Magnitudes are subtracted with a microoperation EA ← A+B'+1. No overflow occurs if the numbers are subtracted so AVF is cleared to 0. E=1 indicates A>=B and number (this number is checked again for 0 to make positive 0 [As=0]) in A is correct result. E=0 indicates A<B, so we take 2's complement of A.

A +4=0100
B -3=1101
---
1 0001

A +3=0011
B -4=1100
---
0 1111

Whether A>B or A<B; subtract B from A and if E=0 means, A<B so the result negative so take 2's comp of A

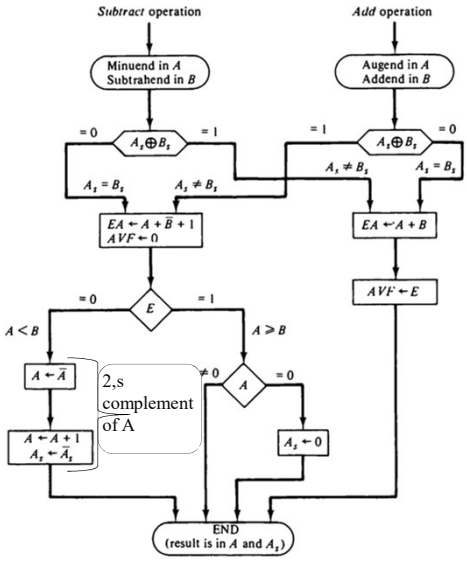


Figure 10-2 Flowchart for add and subtract operations.

Unit-05: Fixed point Computer Arithmetic

Hardware implementation:

- A and B be two registers that hold the magnitudes of the numbers, and A_s and B_s be two flip-flops that hold the corresponding signs. The result of the operation is transferred into A and A_s. Thus A and A_s together form an accumulator register.

Consider now the hardware implementation of the algorithms above.

- First, a parallel-adder is needed to perform the microoperation A + B.
- Second, a comparator circuit is needed to establish if A > B, A = B, or A < B.
- Third, two parallel-subtractor circuits are needed to perform the microoperations A - B and B - A.
- The sign relationship can be determined from an exclusiveOR gate with A_s and B_s as inputs.

Unit-05: Fixed point Computer Arithmetic

Hardware implementation:

- First, we know that subtraction can be accomplished by means of complement and add.
- Second, the result of a comparison can be determined from the end carry after the subtraction. So, the use of 2's complement for subtraction and comparison is an efficient procedure that requires only an adder and a complementer.
- Figure shows a block diagram of the hardware for implementing the addition and subtraction operations.

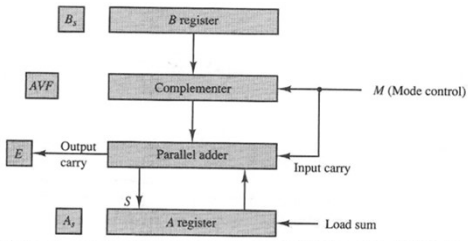


Fig: hardware for signed-magnitude addition and subtraction

Unit-05: Fixed point Computer Arithmetic

Hardware implementation: Block Diagram Description

Hardware consists of registers A and B and sign flip-flops A_s and B_s. Subtraction is done by adding A to the 2' s complement of B. The output carry is transferred to flip-flop E, where it can be checked to determine the relative magnitudes of the two numbers. The add-overflow 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. – details flow chart below slide…

The addition of A plus B is done through the parallel adder. The S (sum) output of the adder is applied to the input of the A register. The complementer provides an output of B or the complement of B depending on the state of the mode control M. The complementer consists of exclusive-OR gates and the parallel adder consists of full-adder circuits.

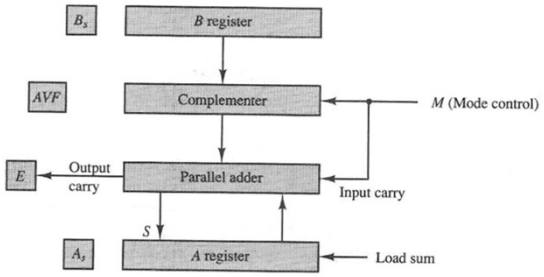


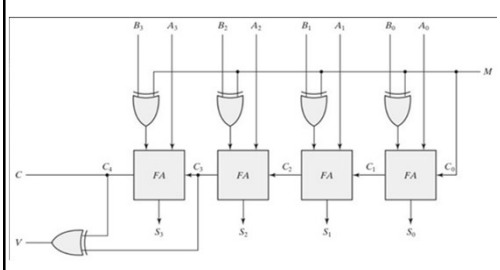
Fig: hardware for signed-magnitude addition and subtraction

Unit-05: Fixed point Computer Arithmetic

Hardware implementation: Block Diagram Description

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

- When M = 0, the output of B is transferred to the adder, the input carry is 0, and the output of the adder is equal to the sum A + B.
- When M=1, the 1's complement of B is applied to the adder, the input carry is 1, and output S = A + B' + 1. This is equal to A plus the 2's complement of B, which is equivalent to the subtraction A-B.



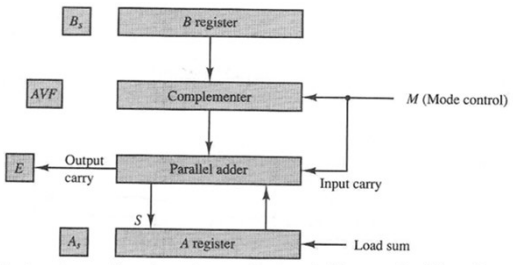
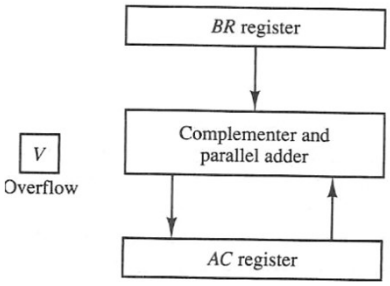


Fig: hardware for signed-magnitude addition and subtraction

Unit-05: Fixed point Computer Arithmetic

Addition and Subtraction with Signed 2's Complemented Data:

- The left most bit of binary number represents the sign bit; 0 for positive and 1 for negative. If the sign bit is 1, the entire number is represented in 2's complement form.
- The addition of two numbers in signed-2's complement form consists of adding the number with the sign bits treated the same as the other bits of the number. A carry out of the sign bit position is discarded.
- The subtraction consists of first taking the 2's complement of the subtrahend and then adding it to the minuend.
- When two numbers of n digits each are added and sum occupies n+1 digits, we say that an overflow occurred.
- When the two carries are applied to an exclusive-OR gate, the overflow is detected when the output of the gate is equal to 1.

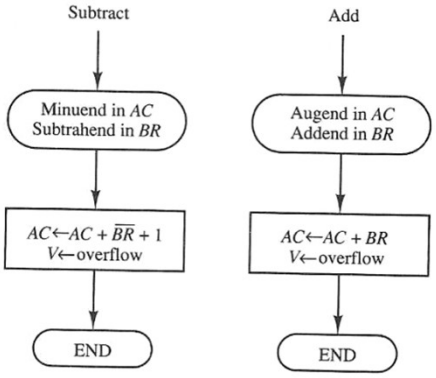


Unit-05: Fixed point Computer Arithmetic

Addition and Subtraction with Signed 2's Complemented Data: Algorithm:

Example: 33 + (-35)
AC = 33 = 00100001
BR = -35 = 2's complement of 35 = 11011101
AC + BR = 11111110 = -2 which is the result

Comparing this algorithm with its signed-magnitude counterpart, it is much easier to add and subtract numbers. For this reason most computers adopt this representation over the more familiar signed-magnitude.

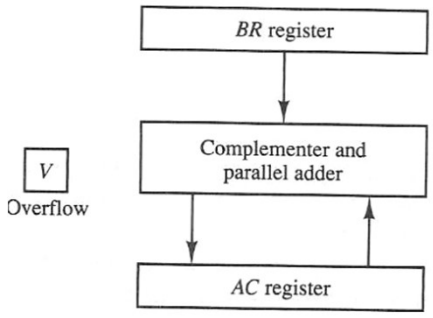


Unit-05: Fixed point Computer Arithmetic

Addition and Subtraction with Signed 2's Complemented Data: Hardware

→Register configuration is same as signed-magnitude representation except sign bits are not separated. The leftmost bits in AC and BR represent sign bits.

→Significant difference: sign bits are added are subtracted together with the other bits in complementer and parallel adder. The overflow flip-flop V is set to 1 if there is an overflow. Output carry in this case is discarded.

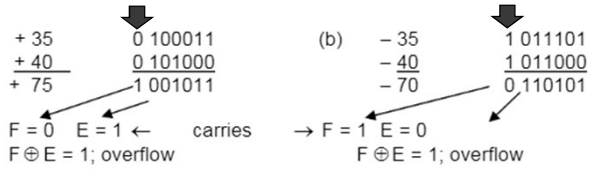


Unit-05: Fixed point Computer Arithmetic

Problem:

Perform the arithmetic operations below with binary numbers and with negative numbers in signed-2's complement. Use seven bits to accommodate each number together with its sign. In each case, determine if there is an overflow by checking the carries into and out of the sign bit position .

a. (+35) + (+40)
b. (-35) + (-40)
c. (-35) - (+40)



E = ? , Overflow (AVF) ?

35 = 0 100011

40 = 0 101000

Carries into sign bit position -> (E)
Carries out of the sign bit position ->(F)

Unit-05: Fixed point Computer Arithmetic

Problem:

10-2. Mark each individual path in the flowchart of Fig. 10-2 by a number and then indicate the overall path that the algorithm takes when the following signed-magnitude numbers are computed. In each case give the value of *AVF*. The leftmost bit in the following numbers represents the sign bit.

a. 0 101101 + 0 011111
b. 1 011111 + 1 101101
c. 0 101101 − 0 011111
d. 0 101101 − 0 101101
e. 1 011111 − 0 101101

a. 0 101101 + 0 011111
b. 1 011111 + 1 101101
c. 0 101101 - 0 011111
d. 0 101101 + 0 101101
e. 1 011111 + 0 101101

Unit-05: Fixed point Computer Arithmetic

Problem:

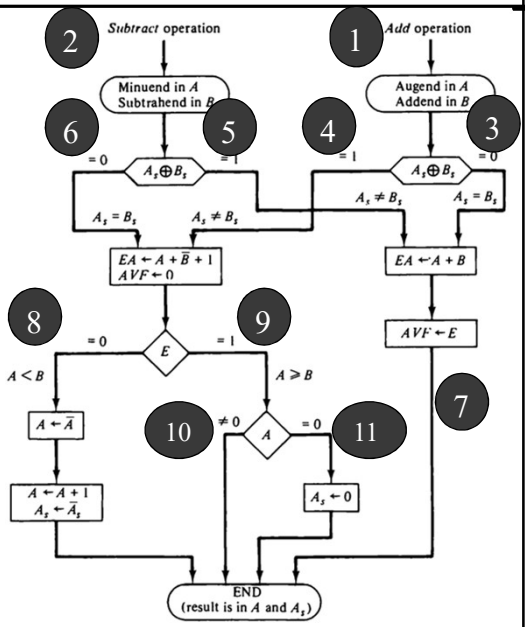


Figure 10-2 Flowchart for add and subtract operations.

Unit-05: Fixed point Computer Arithmetic

Unit-05: Fixed point Computer Arithmetic

5.3 Multiplication Algorithm: Hardware Implementation for Signed Magnitude Data, Booth Multiplication Algorithm (with Numerical Example)

Unit-05: Fixed point Computer Arithmetic

Multiplication Algorithm: signed-magnitude representation

Multiplication of two fixed-point binary numbers in signed-magnitude representation is done by a process of successive shift and add operations.

Signed-magnitude representation

For this representation, multiplication is done by a process of successive shift and adds operations. As an example:

```
23      10111    Multiplicand
19    × 10011    Multiplier
        10111
       10111
      00000   +
     00000
    10111
437 110110101    Product
```

Process consists of looking successive bits of the multiplier, least significant bits first. If the multiplier bit is 1, the multiplicand is copied down; otherwise, zeros are copied down. Numbers copied down in successive lines are shifted one position Shifted left one position. Finally, numbers are added to form a product.

The sign of the product is determined from the signs of the multiplicand and multiplier. If they are alike, the sign of the product is positive. If they are unlike, the sign of the product is negative.

Unit-05: Fixed point Computer Arithmetic

Multiplication Algorithm: signed-magnitude representation

When multiplication is implemented in a digital computer, it is convenient to change the process slightly.

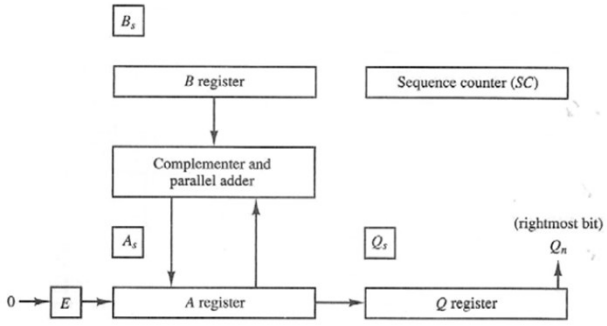
- First, instead of providing registers to store and add simultaneously as many binary numbers as there are bits in the multiplier, it is convenient to provide an adder for the summation of only two binary numbers and successively accumulate the partial products in a register.
- Second, instead of shifting the multiplicand to the left, the partial product is shifted to the right, which results in leaving the partial product and the multiplicand in the required relative positions.
- Third, when the corresponding bit of the multiplier is 0, there is no need to add all zeros to the partial product since it will not alter its value.

Unit-05: Fixed point Computer Arithmetic

Multiplication Algorithm: signed-magnitude representation

Hardware implementation for signed-magnitude data

It needs same hardware as that of addition and subtraction of signed-magnitude. In addition it needs two more registers Q and SC.



- Successively accumulate partial products and shift it right.
- Q ← multiplier and Qs ← sign.
- SC ← no. of bits in multiplier (magnitude only).
- SC is decremented after forming each partial product. When SC is 0, process halts and final product is formed.
- B ← multiplicand, B_s ← sign
- Sum of A and B forms a partial product

Fig: Hardware for multiply operation

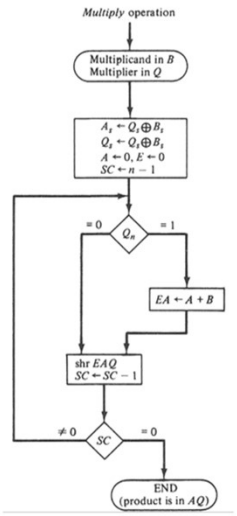
Unit-05: Fixed point Computer Arithmetic

Multiplication Algorithm: Signed-magnitude representation

Note: The sign of the product is determined from the signs of the multiplicand and multiplier. If they are alike, the sign of the product is positive. If they are unlike, the sign of the product is negative.

Hardware Algorithm

Flowchart below shows a hardware multiply algorithm.



SC=n-1 (total no of bits including sign bit minus 1)

B= 23

Q= 19

Example: B = 10111 (**Multiplicand**)
Q = 10011 (**Multiplier**)

Operation	E	A	Q	SC
Initial conf.	0	00000	10011	101
Iteration 1 (Q_n = 1) EA← A+B		00000 + 10111		
PP1-->	0	10111	10011	
shr EAQ, SC←SC-1	0	01011	11001	100
Iteration 2 (Q_n = 1) EA← A+B		01011 + 10111		
PP2-->	1	00010	11001	
shr EAQ, SC←SC-1	0	10001	01100	011
Iteration 3 (Q_n = 0) shr EAQ, SC←SC-1	0	01000	10110	010
Iteration 4 (Q_n = 0) shr EAQ, SC←SC-1	0	00100	01011	001
Iteration 5 (Q_n = 1) EA← A+B		00100 + 10111		
PP3-->	0	11011	01011	
shr EAQ, SC←SC-1	0	01101	10101	000
Final Product in AQ	0110110101			

Unit-05: Fixed point Computer Arithmetic

Booth Multiplication Algorithm: signed-2's complement representation.

Booth algorithm requires examination of the multiplier bits and shifting of the partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, 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. (Q_n, Q_{n+1} = 1 0)
2. The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous 1) in a string of 0's in the multiplier. (Q_n, Q_{n+1} = 0 1)
3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit. (Q_n, Q_{n+1} = 0 0 , 11)

Unit-05: Fixed point Computer Arithmetic

Booth Multiplication Algorithm: signed-2's complement representation.

Multiplier (QR= -13(01101); 2's comp=10011 ; Multiplier
Multiplicand (BR= -9(01001); 2's comp=10111
and Result are stored in AC and QR. (Note: write both QR and BR initially in 2's complement form)

TABLE 10-3 Example of Multiplication with Booth Algorithm

$Q_n Q_{n+1}$	$BR = 10111$ $\overline{BR} + 1 = 01001$	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

Note: An arithmetic shift right (ashr) operation shifts AC and QR to the right and leaves the sign bit in AC unchanged.

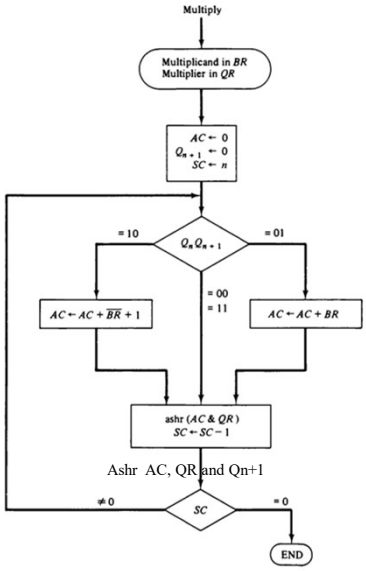


Figure 10-8 Booth algorithm for multiplication of signed-2's complement

Example of Booth Multiplication Algorithm: signed-2's complement representation

Example: Multiply the two numbers 23 and -9 by using the Booth's multiplication algorithm.

Here, M = 23 = (010111) and Q = -9 = (110111)

Q_{n+1}	Q_n	M = 0 1 0 1 1 1 M' + 1 = 1 0 1 0 0 1	AC	Q	Q_{n+1}	SC
		Initially	000000	110111	0	6
1	0	Subtract M	101001			
			101001			
		Perform Arithmetic right shift operation	110100	111011	1	5
1	1	Perform Arithmetic right shift operation	111010	011101	1	4
1	1	Perform Arithmetic right shift operation	111101	001110	1	3
0	1	Addition (A + M)	010111			
			010100			
		Perform Arithmetic right shift operation	001010	000111	0	2
1	0	Subtract M	101001			
			110011			
		Perform Arithmetic right shift operation	111001	100011	1	1
1	1	Perform Arithmetic right shift operation	**111100**	**110001**	**1**	0

Last bit of Q is 1, it means the output is negative.
Hence, 23 * -9 = 2's complement of 111100110001 => **(00001100111)**

Unit-05: Fixed point Computer Arithmetic

Booth Multiplication Algorithm: signed-2's complement representation.

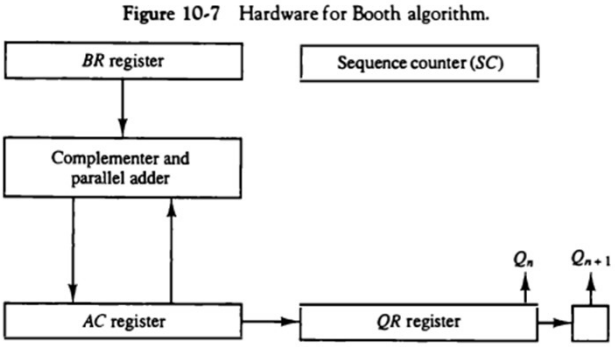


Figure 10-7 Hardware for Booth algorithm.

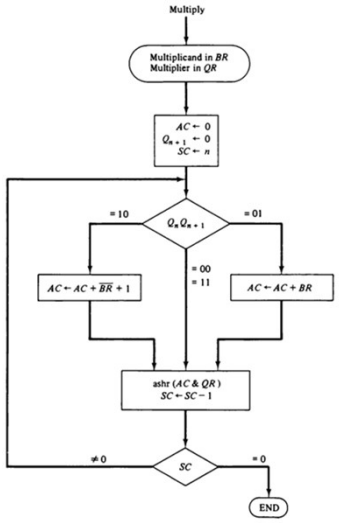


Figure 10-8 Booth algorithm for multiplication of signed-2's complement

Unit-05: Fixed point Computer Arithmetic

Booth Multiplication Algorithm: signed-2's complement representation.

The hardware implementation of Booth algorithm requires the register configuration shown in Fig. 10-7. This is similar to Fig. 10-5 except that the sign bits are not separated from the rest of the registers.

To show this difference, we 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. An extra flip-flop Q_{n+1} is appended to QR to facilitate a double bit inspection of the multiplier.

- The flowchart for Booth algorithm is shown in Fig. 10-8. AC and the appended bit Q_{n+1} are initially cleared to 0 and the sequence counter SC is set to a number n equal to the number of bits in the multiplier.
- The next step is to shift right the partial product and the multiplier (including bit Q_{n+1}). This is an arithmetic shift right (ashr) operation which shifts AC and QR to the right and leaves the sign bit in AC unchanged (see Sec. 4-6). The sequence counter is decremented and the computational loop is repeated n times.

Unit-05: Fixed point Computer Arithmetic

Booth Multiplication Algorithm: signed-2's complement representation.

- The two bits of the multiplier in Q_n and Q_{n+1} are inspected.
  a. 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.
  b. 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.
  c. When the two bits are equal, the partial product does not change.

Booth's Algorithm Work out Example (-12 x -11 =132)

Comments	SC	Multiplicand M	Product P	Product Q	Q_e
Initial (-12 x -11)	0	10100	00000	10101	0
Q_0Q_e = 10; P←P-M	0	10100	01100	10101	0
Ar.sh.r . PQ Q_e	0	10100	00110	01010	1
SC ← SC+1	1	10100	00110	01010	1
Q_0Q_e = 01; P←P+M	1	10100	11010	01010	1
Ar.sh.r . PQ Q_e	1	10100	11101	00101	0
SC ← SC+1	2	10100	11101	00101	0
Q_0Q_e = 10; P←P-M	2	10100	01001	00101	0
Ar.sh.r . PQ Q_e	2	10100	00100	10010	1
SC ← SC+1	3	10100	00100	10010	1
Q_0Q_e = 01; P←P+M	3	10100	11000	10010	1
Ar.sh.r . PQ Q_e	3	10100	11100	01001	0
SC ← SC+1	4	10100	11100	01001	0
Q_0Q_e = 10; P←P-M	4	10100	01000	01001	0
Ar.sh.r . PQ Q_e	4	10100	00100	00100	1
SC ← SC+1	5	10100	**00100**	**00100**	1

M= -12 (01100); 2's comp =10100
Q= -11 (01011); 2's comp = 10101

-12 x -11 = 132

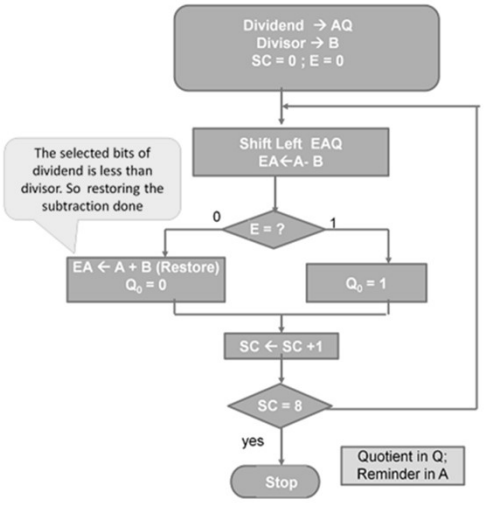
Course Contents

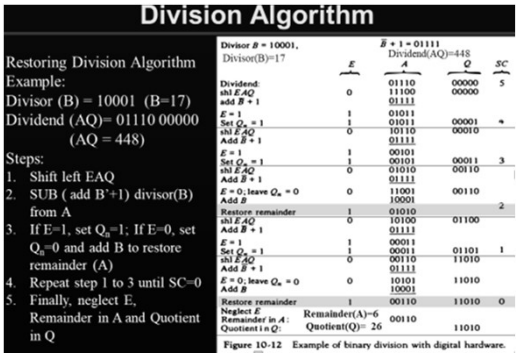
Unit-05: Fixed point Computer Arithmetic

5.4 Division Algorithm: Hardware Implementation for Signed Magnitude Data, Hardware Algorithm (Restoring Only)

Division Algorithm

Division Algorithm:(Restoring Only)





Division Algorithm:(Restoring Only)

The division process are in below table:

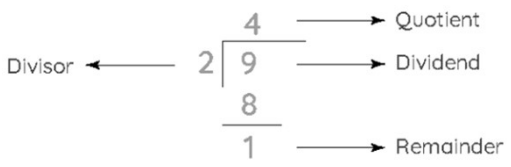
- The divisor is stored in the B register and the double-length dividend is stored in registers A and Q.
- The dividend is shifted to the left and the divisor is subtracted by adding its 2' s complement value.
- The information about the relative magnitude is available in E. If E = 1, it signifies that A >= B. A quotient bit 1 is inserted into Q, and the partial remainder is shifted to the left to repeat the process.
- If E = 0, it signifies that A < B so the quotient in Q, remains a 0 (inserted during the shift). The value of B is then added to restore the partial remainder in A to its previous value. The partial remainder is shifted to the left and the process is repeated again until all five quotient bits are formed.

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

Division Algorithm

Note that while the partial remainder is shifted left, the quotient bits are shifted also and after five shifts, the quotient is in Q and the final remainder is in A.

We have to consider the sign of the result and a possible overflow condition. The sign of the quotient is determined from the signs of the dividend and the divisor. If the two signs are alike, the sign of the quotient is plus. If they are unalike, the sign is minus. The sign of the remainder is the same as the sign of the dividend.



Division Algorithm

Restoring Division Algorithm
Example:
Divisor (B) = 10001 (B=17)
Dividend (AQ)= 01110 00000
(AQ = 448)

Steps:

1. Shift left EAQ
2. SUB (add B'+1) divisor(B) from A
3. If E=1, set Q_n=1; If E=0, set Q_n=0 and add B to restore remainder (A)
4. Repeat step 1 to 3 until SC=0
5. Finally, neglect E, Remainder in A and Quotient in Q

Divisor B = 10001, $\bar{B}$ + 1 = 01111
Divisor(B)=17 Dividend(AQ)=448

	E	A	Q	SC
Dividend:		01110	00000	5
shl EAQ	0	11100	00000	
add $\bar{B}$ + 1		01111		
E = 1	1	01011		
Set Q_n = 1	1	01011	00001	4
shl EAQ	0	10110	00010	
Add $\bar{B}$ + 1		01111		
E = 1	1	00101		
Set Q_n = 1	1	00101	00011	3
shl EAQ	0	01010	00110	
Add $\bar{B}$ + 1		01111		
E = 0; leave Q_n = 0	0	11001	00110	
Add B		10001		
Restore remainder	1	01010		2
shl EAQ	0	10100	01100	
Add $\bar{B}$ + 1		01111		
E = 1	1	00011		
Set Q_n = 1	1	00011	01101	1
shl EAQ	0	00110	11010	
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:	Remainder(A)=6	00110		
Quotient in Q:	Quotient(Q)= 26		11010	

Figure 10-12 Example of binary division with digital hardware.

Division Algorithm

Restoring Division Algorithm
Example:
Divisor (B) = 10001 (B=17)
Dividend (AQ)= 01110 00000
(AQ = 448)

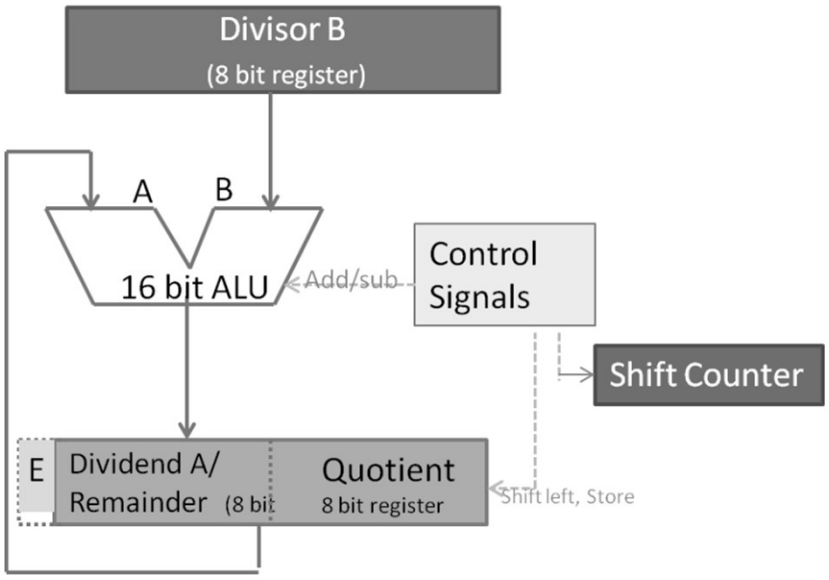
Steps:

1. Shift left EAQ
2. SUB (add B'+1) divisor(B) from A
3. If E=1, set Q_n=1; If E=0, set Q_n=0 and add B to restore remainder (A)
4. Repeat step 1 to 3 until SC=0
5. Finally, neglect E, Remainder in A and Quotient in Q

Restoring Division Algorithm

Initial	Operation	E	A	Q	SC
1	a. Shift b. SUB c. Set d. Restore/No Restore				SC-5
2	a. Shift b. SUB c. Set d. Restore/No Restore				SC-4
3	a. Shift b. SUB c. Set d. Restore/No Restore				SC-3
4	…				SC-2
5					Until 0

Division Algorithm



Division Algorithm

Restoring Division Algorithm

Restoring Division Workout

Given : 194 ÷ 10 194 = 1100 0010 10 = 0000 1010

2 's complement of 10 = 1111 0110

Initial Values: B = 0000 1010 , A =0000 0000, Q = 1100 0010 , SC = 8, E = 0

Step Details	SC	E	A Q
Shift left EAQ		0	0000 0001 1000 0100
Subtract EA← A – B		0	1111 0111 1000 0100
E = 0; Therefore (Restore) A←A+B; Q_0 = 0		0	0000 0001 1000 0100
SC ← SC + 1	1		
Shift left EAQ		0	0000 0011 0000 1000
Subtract EA← A – B		0	1111 1001 0000 1000
E = 0; Therefore (Restore) A←A+B; Q_0 = 0		0	0000 0011 0000 1000
SC ← SC + 1	2		
Shift left EAQ		0	0000 0110 0001 0000
Subtract EA← A – B		0	1111 1100 0001 0000
E = 0; Therefore (Restore) A←A+B; Q_0 = 0		0	0000 0110 0001 0000
SC ← SC + 1	3		
Shift left EAQ		0	0000 1100 0010 0000
Subtract EA← A – B		1	0000 0010 0010 0000
E = 1; Therefore Q_0 = 1		1	0000 0010 0010 0001
SC ← SC + 1	4		
Shift left EAQ		0	0000 0100 0100 0010
Subtract EA← A – B		0	1111 1110 0100 0010
E = 0; Therefore(Restore) A←A+B; Q_0 = 0		0	0000 0100 0100 0010
SC ← SC + 1	5		
Shift left EAQ		0	0000 1000 1000 0100
Subtract EA← A – B		0	1111 1110 1000 0100
E = 0; Therefore (Restore) A←A+B; Q_0 = 0		0	0000 1000 1000 0100
SC ← SC + 1	6		
Shift left EAQ		0	0001 0001 0000 1000
Subtract EA← A – B		1	0000 0111 0000 1000
E = 1; Therefore Q_0 = 1		1	0000 0111 0000 1001
SC ← SC + 1	7		
Shift left EAQ		0	0000 1110 0001 0010
Subtract EA← A – B		1	0000 0100 0001 0010
E = 1; Therefore Q_0 = 1		1	0000 0100 0001 0011
SC ← SC + 1	8		R=4 Q=19

Division Algorithm

Problem-01: Show the contents of registers E, A, Q, and SC (as in Table 10-2: Numerical Example for Binary Multiplier -Signed-Magnitude Data) during the process of multiplication of two binary numbers, 11111 (multiplicand) and 10101 (multiplier). The signs are not included.

Problem-02: Show the step-by-step multiplication process using Booth algorithm (as in Table 10-3: Example of Multiplication with Booth Algorithm) when the following binary numbers are multiplied. Assume 5-bit registers that hold signed numbers. The multiplicand in both cases is + 15.

a. (+15) x (+13)
b. (+15) x (-13)

Problem-03: Show the contents of registers E, A, Q, and SC (as in Fig. 10-12: Example of binary division with digital hardware.) during the process of division of (a) 10100011 by 1011; (b) 00001111 by 0011. (Use a dividend of eight bits.)

Problem-04: Show that adding B after the operation A + B' + 1 restores the original value of A. What should be done with the end carry?