## Unit # 3

**Data Representation**

**Dr. Rajesh Tiwari Professor ( CSE – AIML) CMREC, Hyderabad, Telangana**

# Complements

* 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.
* When the value of the base r is substituted in the name, the two types are referred to as the 2's and 1's complement for binary numbers and the 10's and 9's complement for decimal numbers.

# ( r - 1 )'s Complement

* Given a number N in base r having n digits, the (r - 1)'s complement of N is defined as (r n - 1) - N.
* For decimal numbers r = 10 and r - 1 = 9, so the 9's complement of N is (10 n - 1) - N.
* 10 n represents a number that consists of a single 1 followed by n 0's.
* 10 n - 1 is a number represented by n 9's.
* Example,

– With n = 4 we have 104 = 10000 and 104 - 1 = 9999.

* + It follows that the 9' s complement of a decimal number is obtained by subtracting each digit from 9.
  + For example, the 9's complement of 546700 is 999999 - 546700 =

453299 and the 9's complement of I2389 is 99999 - 12389 = 87610.

* The (r - 1)'s complement of octal or hexadecimal numbers are obtained by subtracting each digit from 7 or F (decimal 15) respectively.

# (r's) Complement

* The r's complement of an n-digit number N in base r is defined as r n - N for N ≠ 0 and 0 for N = 0.
* Comparing with the (r - 1)'s complement, we note that the r's complement is obtained by adding 1 to the (r - 1)'s complement since r n - N = [(r n - 1) - N] + 1.
* Thus the 10's complement of the decimal 2389 is 7610 +

1 = 761 1 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.

**Fixed-Point Representation**

* Generally, Negative number is indicated by a minus sign and a positive number by a plus sign.
* As a consequence, it is customary to represent the sign with a

bit placed in the leftmost position of the number.

* The convention is to make the sign bit equal to 0 for positive and to 1 for negative.
* A number may have a binary (or decimal) point.
* The position of the binary point is needed to represent fractions, integers, or mixed integer-fraction numbers.
* The representation of the binary point in a register is complicated by the fact that it is characterized by a position in the register.
* There are two ways of specifying the position of the binary point in a register: by giving it a fixed position or by employing a floating-point 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.

## Integer Representation

* When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number.
* When the number is negative, the sign is represented by 1 but the rest of the number may be represented in one of three possible ways:
  + Signed-magnitude representation
  + Signed-1' s complement representation
  + Signed 2' s complement representation
* The signed-magnitude representation of a negative number consists of the magnitude and a negative sign.
* In the other two representations, the negative number is represented in either the 1's or 2's complement of its positive value.

## Integer Representation

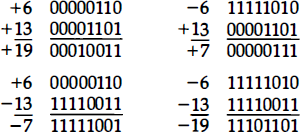
* As an example, consider the signed number 18 stored in an 8-bit register. + 18 is represented by a sign bit of 0 in the leftmost position followed by the binary equivalent of 18: 00010010.
* Note that each of the eight bits of the register must have a value and therefore 0' s must be inserted in the most significant positions following the sign bit.
* Although there is only one way to represent + 14, there are three different ways to represent - 14 with eight bits.
  + In signed-magnitude representation 1 0010010
  + In signed-1's complement representation 1 11 01101
  + In signed-2's complement representation 1 11 01110
* The signed-magnitude representation of - 18 is obtained from + 18

by complementing only the sign bit.

* The signed-1's complement representation of – 18 is obtained by complementing all the bits of + 18, including the sign bit.
* The signed-2' s complement representation is obtained by taking the

2' s complement of the positive number, including its sign bit.

## Arithmetic Addition

* The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic.
* If the signs are the same, we add the two magnitudes and give the sum the common sign.
* If the signs are different, we subtract the smaller magnitude from the larger and give the result the sign of the larger magnitude.

## Arithmetic Subtraction

* Subtraction of two signed binary numbers when negative numbers are in 2' s complement form is very simple and can be stated as follows: Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign bit).
* A carry out of the sign bit position is discarded.
* Solve - 6 – (-13).
* Solve – 10 – (-15)

# Addition and Subtraction

* Addition and Subtraction with Signed-Magnitude Data
  + We designate the magnitude of the two numbers by A and B.
  + When 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 3.1.
  + 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 prevent a negative zero.
  + In other words, when two equal numbers are subtracted, the

result should be +0 not -0.

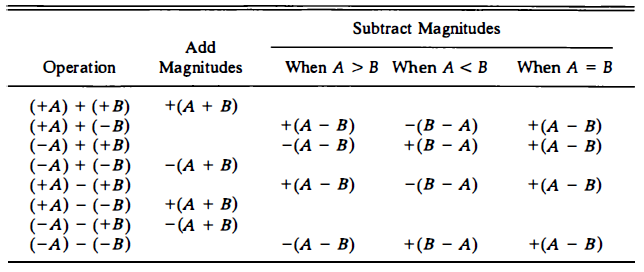
## Addition (subtraction) Algorithm

* 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.

## Addition (subtraction) Algorithm

**Table 3.1: Addition and Subtraction of Signed·Magnitude Numbers**



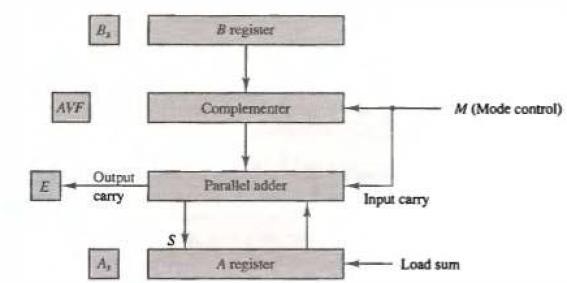


Fig. 3.1 : Hardware for sign-magnitude addition and subtraction.

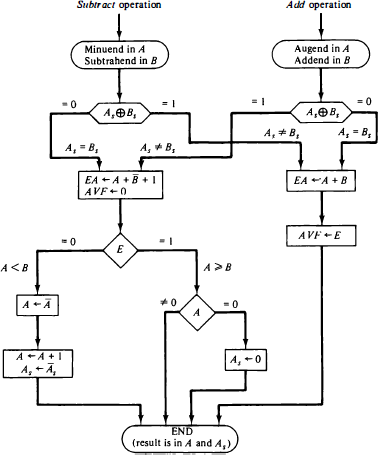


Fig. 3.2: Flowchan for add and subtract operations

Multiplication Algorithms

* The hardware for multiplication

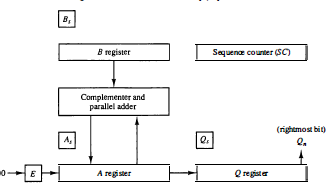
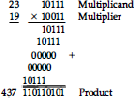


Fig. 3. : Hardware for multiply operation.



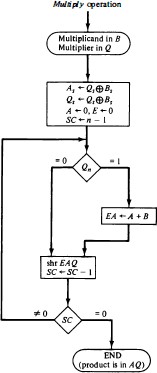
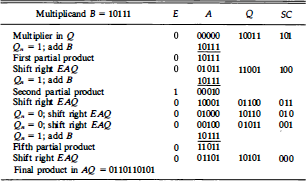


Fig. 3. : Flowchart for multiply operatio

TABLE 3. 2: Numerical Example for Binary Multiplier



# Booth Multiplication 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 2k to weight 2m can be treated as 2k+1 - 2m.
* For example, the binary number 001 110 ( + 14) has a string of

1's from 23 to 21 (k = 3, m = 1).

* The number can be represented as 2 k+ l - 2m = 24 - 21 = 16 -2 = 14.
* Therefore, the multiplication M x 14, where M is the multiplicand and 14 the multiplier, can be done as M x 24 - M X 21.
* Thus the product can be obtained by shifting the binary multiplicand M four times to the left and subtracting M shifted left once.
* As in all multiplication schemes, 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.
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 O's in the multiplier.
3. The partial product does not change when the multiplier bit is

identical to the previous multiplier bit.

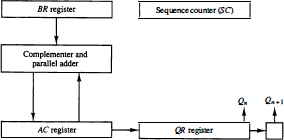


Figure : Hardware for Booth algorithm.

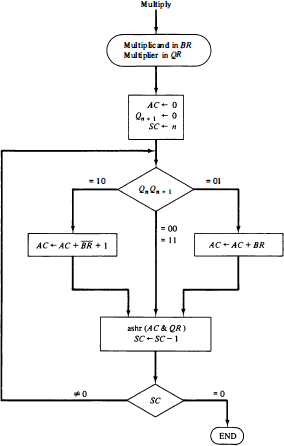


Figure : Booth algorithm for multiplication o f signed- 2's complement numbers.

TABLE : Example of Multiplication with Booth Algorithm

