2.1 ARITHMETIC

2.1.1 INTRODUCTION

Data is manipulated by using the arithmetic instructions in digital computers to give solution for the computation problems. The addition, subtraction, multiplication and division are the four basic arithmetic operations. **Arithmetic processing unit** is responsible for executing these operations and it is located in central processing unit.

The arithmetic instructions are performed on binary or decimal data**. Fixed-point numbers** are used to represent integers or fractions. These numbers can be signed or unsigned negative numbers. A wide range of arithmetic operations can be derived from the basic operations.

2.1.2 Signed and Unsigned Numbers:

Signed numbers:

These numbers require an arithmetic sign. The most significant bit of a binary number is used to represent the sign bit. If the sign bit is equal to zero, the signed binary number is positive; otherwise, it is negative. The remaining bits represent the actual number. The negative numbers may be represented either 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
- Signed 2's complement

Unsigned binary numbers:

These are positive numbers and thus do not require an arithmetic sign. An m-bit unsigned number represents all numbers in the range 0 to $2^m - 1$. For example, the range of 16-bit unsigned binary numbers is from 0 to 65,53510 in decimal and from 0000 to FFFF16 in

hexadecimal.

Signed Magnitude Representation:

The most significant bit (MSB) represents the *sign*. A 1 in the MSB bit position denotes a negative number and 0 denotes a positive number. The remaining *n* •1 bits are preserved and represent the magnitude of the number.

Examples:

Number	Signed Magnitude Representation
+ 3	0011
-3	1011
0	0000
-0	1011
5	0101
-5	1101

One's Complement Representation:

In one's complement, positive numbers remain unchanged as before with the sign- magnitude numbers. Negative numbers are represented by taking the one's complement (inversion, negation) of the unsigned positive number. Since positive numbers always start with a 0, the complement will always start with a 1 to indicate a negative number.

The one's complement of a negative binary number is the complement of its positive counterpart, so to take the one┼s complement of a binary number.

Number	One's complement Representation
00001000 (+8)	11110111
10001000(-8)	01110111
00001100(+12)	11110011
10001100(-12)	01110011

Two's Complement Representation:

In two's complement, the positive numbers are exactly the same as before for unsigned binary numbers. A negative number, is represented by a binary number, which when added to its corresponding positive equivalent results in zero.

In two's complement form, a negative number is the 2's complement of its positive number with the subtraction of two numbers being A – B = A + (2's c complement of B) using much the same process as before as basically, two's complement is adding 1 to one's complement of the number.

The main difference between 12 s complement and 22 s complement is that 12 s complement has two representations of 0 (+0): 00000000, and (-0): 11111111. In 22 s complement, there is only one representation for zero: 00000000 (0).

+0: 00000000

2's complement of -0:

-0: 00000000 (Signed magnitude representation)

11111111 (1's complement representation)

11111111 + 1= 00000000 (2'scomplement representation)

These shows in 2'scomplement representation both +0 and -0 takes same value. This solves the double-zero problem, which existed in the 1's complement.

Example 2.1: Convert 210 and -210 to 32 bit binary numbers.

+2= 0000 0000 0000 0010 (16 bits)

= 0000 0000 0000 0000 0000 0000 0000 0010 (32 bits)

It is converted to a 32-bit number by making 16 copies of the value in the most significant bit

(0) and placing that in the left-

hand half of the word. 2=0000

0000 0000 0010

-2=1's complement of 2 +1

1111 1111 1111 1101 (1's complement of 2) + 1

= 1111 1111 1111 1110 (16 bits)

= 1111 1111 1111 1111 1111 1111 1111 1110 (32 bits)

To convert to 32 bit number copy the digit in the MSB of the 16 bit number for 16 times and fill the left half.

2.1.3 FIXED POINT ARITHMETIC

A fixed-point number representation is a real data type for a number that has a fixed number of digits after the radix point or decimal point.

This is a common method of integer representation is sign and magnitude representation. One bit is used for denoting the sign and the remaining bits denote the magnitude. With 7 bits reserved for the magnitude, the largest and smallest numbers represented are +127 and –127. Fixed-point numbers are useful for representing fractional values, usually in base 2 or base 10, when the executing processor has no floating point unit (FPU) or if fixed-point provides improved performance or accuracy for the application at hand. Most low-cost embedded microprocessors and microcontrollers do not have an FPU.

A value of a fixed-point data type is essentially an integer that is scaled by a specific factor. The scaling factor is usually a power of 10

(for human convenience) or a power of 2 (for computational efficiency). However, other scaling factors may be used occasionally, e.g. a time value in hours may be represented as a fixed-point type with a scale factor of 1/3600 to obtain values with one-second accuracy. The maximum value of a fixed-point type is the largest value that can be represented in the underlying integer type, multiplied by the scaling factor; and similarly for the minimum value.

Example:

The value 1.23 can be represented as 1230 in a fixed-point data type with scaling factor of 1/1000.

Precision loss and overflow

- The fixed point operations can produce results that have more bits than the operands there is possibility for information loss.
- In order to fit the result into the same number of bits as the operands, the answer must be rounded or truncated.
- Fractional bits lost below this value represent a precision loss which is common in fractional multiplication.
- If any integer bits are lost, however, the value will be radically inaccurate.
- Some operations, like divide, often have built-in result limiting so that any positive overflow results in the largest possible number that can be represented by the current format.
- Likewise, negative overflow results in the largest negative number represented by the current format. This built in limiting is often referred to as **saturation.**
- Some processors support a hardware overflow flag that can generate an exception on the occurrence of an overflow, but it is usually too late to salvage the proper result at this point.