1.1 LEARNING OBJECTIVES

After going through this unit, you will be able to :

- define and describe number system
- identify how data is represented in computers
- convert a number from one number system to another
- describe how computers perform binary arithmatic

1.2 INTRODUCTION

Number system is a fundamental concept used in micro computer system. They are of different types and can represent by some digit symbols. The knowledge of binary, octal and hexadecimal number system is essential to understand the operation of a computer. This unit deals with all this system. In this unit we will discuss about all the representation of this number system and their conversion from one number system to its equivalent other number system. The arithmetic operations of all the system is very much important to know how a system can operate our data inside. This is also discussed in this unit. In addition you we will get the internal data representation method which is called computer codes . Computer codes like BCD code i

1.3 NUMBER SYSTEM

We are familier with the decimal number system which is used in our day-to-day work. Ten digits are used to four decimal number. To represent these decimal digits, ten separate symbols 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 are used. But a digital computer stores, understands and manipulates information composed of only zeros and ones. So, each decimal digits, letters, symbols etc. written by the programmer (an user) are converted to binary codes in the form of 0's and 1's within the computer. The number system is divided into different categories according to the base (or radix) of the system as binary, octal and hexadecimal. If a number system of base r is a system, then the system have r distinct symbols for r digits. The knowledge of the number system is essential to understand the operation of a computer.

1.3.1 Decimal Number System

Decimal Number System uses 10 digits from 0 to 9 to represents its system.

Decimal no. system have ten digits represented by 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. So, the base or radix of such system is 10.

In this system the successive position to the left of the decimal point represent units, tens, hundreds, thousands etc. For example, if we consider a dicimal number 1257, then the digit representations are :

$$\begin{tikzcd}
1 \arrow[d] & 2 \arrow[d] & 5 \arrow[d] & 7 \arrow[d] \\
\text{thousands positions} & \text{hundreds position} & \text{tens position} & \text{units position}
\end{tikzcd}$$

The weight of each digit of a number depends on its relative position within the number.

Example 1.1 :

The weight of each digit of the decimal no. 6472

$6472 = 6000 + 400 + 70 + 2 = 6 \times 10^3 + 4 \times 10^2 + 7 \times 10^1 + 2 \times 10^0$

The weight of digits from right hand side are :
Weight of 1st digit = 2×10^0

Weight of 2nd digit = 7×10^1

Weight of 3rd digit = 4×10^2

Weight of 4th digit = 6×10^3

The above expressions can be written in general forms as the weight of n^{th} digit of the number from the right hand side :

= n^{th} digit × 10^{n-1}

= n^{th} digit × $(\text{base})^{n-1}$

The no. system in which the weight of each digit depends on its relative position within the number is called positional number system. The above form of general expression is true only for positional number system.

A ***Binary Number System*** uses only digit 0 and 1

1.3.2 Binary Number System

Only two digits 0 and 1 are used to represent the binary number system. So the base or radix is two (2). The digits 0 and 1 are called bits (Binary Digits). In this number system the value of the digit will be two times greater than its predecessor. Thus the value of the places are :

$$\leftarrow \leftarrow 32 \leftarrow 16 \leftarrow 8 \leftarrow 4 \leftarrow 2 \leftarrow 1$$

The weight of each binary bit depends on its relative position within the number. It is explained by the following example--

Example 1.2 :

The weight of bits of the binary number 10110 is :

$= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

$= 16 + 0 + 4 + 2 + 0 = 22$ (decimal number)

The weight of each bit of a binary no. depends on its relative pointer within the no. and explained from right hand side as :

Weight of 1st bit = 1st bit X 2^0

Weight of 2nd bit = 2nd bit X 2^1

....

....

and so on.

The weight of the n^{th} bit of the number from right hand side

= n^{th} bit × 2^{n-1}

= n^{th} bit × $(\text{Base})^{n-1}$

It is seen that this rule for a binary number is same as that for a decimal number system. The above rule holds good for any other positioned number system. The weight of a digit in any positioned number system depends on its relative positon within the number and the base of the number system.

Table 1.1 shows the binary equivalent numbers for decimal digits.

Table 1.1 : Binary equivalent of decimal numbers

Decimal Number	Equivalent Binary Number
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

Binary Fractions : A binary fractions can be represented by a series of 1s and 0s to the right of a binary point. The weight of digit positions to the right of the binary point are given by 2^{-1}, 2^{-2}, 2^{-3} and so on.

Example 1.3 : Show the representation of binary fraction 0.1101.

Solution : The binary representation of 0.1101 is :

$$0.1101 = 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$$
$$= 1 \times 0.5 + 1 \times 0.25 + 0 \times 0.125 + 1 \times 0.0625$$
$$= 0.8125$$

So, $(0.1101)_2 = (0.8125)_{10}$

1.3.3 Octal Number System

Octal Number System uses 8 digits from 0 to 7.

A commonly used positional number system is the Octal Number System. This system has eight (8) digit representation as 0,1,2,3,4,5,6 and 7. The base or radix of this system is 8. The values increase from left to right as 1,8,64,512, 4096 etc. The decimal value 8 is represented in octal as 10,9 as 11,10 as 12 and so on. As $8=2^3$, an octal number is represented by a group of three binary bits. For example 3 is represented as 011, 4 as 100 etc.

Table 1.2 The octal number and their binary representations.

Decimal Number	Octal Number	Binary Coded Octal No.
0	0	000
1	1	001
.	.	.
.	.	.
7	7	111
8	10	100 000
15	17	001 111

Hexadecimal System groups numbers by 16 and power of 16.

1.3.4 Hexadecimal Number System

The hexadecimal number system is now extensively used in computer industry. Its base (or radix) is 16, ie. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. The hexadecimal numbers are used to represent binary numbers because of case of conversion and compactness.

As $16 = 2^4$, hexadecimal number is represented by a group of four binary bits. For example, 5 is represented by 0101. Table 2.3 shows the binary equivalent of a decimal number and its hexadecimal representation.

Table 1.3 : Hexadecimal number and their Binary representation

Decimal No.	Hexadecimal No.	Binary coded Hex. No
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101

6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

1.4 NUMBER SYSTEM CONVERSION

As the computer uses different number systems, there is a process of converting generally used decimal number systems to other number systems and vice-versa.

1.4.1 Binary to Decimal Conversion

To convert a binary number to its decimal equivalent we use the following expression.The weight of the n^{th} bit of the number from right hand side

$= n^{th}$ bit $\times 2^{n-1}$

First we mark the bit position and then we give the weight of each bit of the number depending on its position. The sum of the weight of all bits gives the equivalent number.

Example 1.4 : Convert binary $(100101)_2$ to its decimal equivalent.

Solution : $(100101)_2 = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^0$

$= 32 + 0 + 0 + 4 + 0 + 1$

$= 37$

So, $(100101)_2 = (37)_{10}$

Mixed number contain both integer and fractional parts and can convert to its decimal equivalent is as follows :

Example 1.5 : Converting $(11011.101)_2$ to its equivalent decimal no.

Solutaion :

$$(11011.101)_2 = (1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) + (1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3})$$

$$= (16 + 8 + 0 + 1) + (0.5 + 0 + 0.125) = 27.625$$

So, $(11011.101)_2 = (27.625)_{10}$

1.4.2 Decimal to Binary Conversion

There are different methods used to convert decimal number to binary numbers. The most common method is, repeatedly divide the decimal number to binary number by 2, then the remainders 0's and 1's obtained after division is read in reverse order to obtain the binary equivalent of the decimal number. This method is called **double-double method**.

Example 1.6 : Convert $(75)_{10}$ to its binary equivalent.

Solution :

$2\,\vert\underline{75}$		Remainder	
$2\,\vert\underline{37}$	LSB	1	
$2\,\vert\underline{18}$		1	
$2\,\vert\underline{9}$		0	Read in
$2\,\vert\underline{4}$		1	reverse order
$2\,\vert\underline{2}$		0	
1	MSB	0	

So, $(75)_{10} = (001011)_2$

Example 1.7 : Convert decimal fraction $(25.625)_{10}$ to its equivalent binary no.

Solution :

$2\vert\underline{25}$	Remainder	MSB	0.625
$2\,\vert\underline{12}$	1		$\underline{\times 2}$
$2\,\vert\underline{6}$	0		1.250
$2\,\vert\underline{3}$	0		$\underline{\times 2}$
1	1		0.500
$(25)_{10} = (11001)_2$			$\underline{\times 2}$
			1.000

$(0.625)_{10} = (0.101)_2$

So, $(25.625)_{10} = (11001.101)_2$

1.4.3 Octal to Decimal Conversion

The method of converting octal numbers to decimal numbers is simple. The decimal equivalent of an octal number is the sum of the numbers multiplied by their corresponding weights.

Example 1.8 : Find decimal equivalent of octal number $(153)_8$

Solution : $1 \times 8^2 + 1 \times 8^1 + 1 \times 8^0 = 64 + 40 + 3 = 107$

So, $(153)_8 = (107)_{10}$

The fractional part can be converted by multiplying it by the negative powers of 8 as shown in the following example.

Example 1.9 : Find decimal equivalent of octal number $(123.21)_8$

Solution : $(1X8^2 + 2X8^1 + 3X8^0) + (2 \times 8^{-1} + 1 \times 8^{-2})$

$= (64 + 16 + 3) + (0.25 + 0.0156) = 83.2656$

So, $(123.21)_8 = (83.2656)_{10}$

1.4.4 Decimal to Octal Conversion

The procedure for conversion of decimal numbers to octal numbers is exactly similar to the conversion of decimal number to binary numbers except replacing 2 by 8.

Example 1.10 : Find the octal equivalent of decimal $(3229)_{10}$

Solution : Remainders

```
8 |4121
 8 |515    1   ↑  read from MSB
  8 |64    3   |  to LSB
   8 |8    0   |
      1    0   |
```

So, $(4121)_{10} = (10031)_8$

The fractional part is multiplied by 8 to get a carry and a fraction as shown in the following example.

Example 1.11 : Find the octal equivalent of $(.123)_{10}$

Solution : Octal equivalent of fractional part of a decimal number as follows :

$$\begin{array}{lll}
8 \times 0.123 = 0.984 & 0 & \\
8 \times 0.984 = 7.872 & 7 & \big\downarrow \text{ read from LSB} \\
8 \times 0.872 = 6.976 & 6 & \big\downarrow \text{ to MSB} \\
8 \times 0.976 = 7.808 & 7 &
\end{array}$$

Read the integer to the left of the decimal point.

The calculation can be terminated after a few steps if the fractional part does not become zero.

The octal equivalent of $(0.123)_{10} = (0.0767)_8$

NOTE : The octal to binary and binary to octal conversion is very easy. Since, 8 is the third power of 2, we can convert each octal digit into its three-bit binary form and vice versa.

Example 1.12 : Convert $(567)_8$ to its binary form.

Solution :

$$\begin{tikzcd}
5 \arrow[d] & 6 \arrow[d] & 7 \arrow[d] \\
101 & 110 & 111
\end{tikzcd}$$

So, $(567)_8 = (101\ 110\ 111)_2$

Conversion from binary to octal is just opposit of the above example.

1.4.5 Hexadecimal to Decimal Conversion

The method of converting Hexadecimal numbers to decimal number is simple. The decimal equivalent of an Hexadecimal number is the sum of the numbers multiplied by their corresponding weights.

Example 1.13 : Find the decimal equivalent of $(4A8C)_{16}$

Solution :

$$\begin{aligned}
(4A8C)_{16} &= (4 \times 16^3) + (10 \times 16^2) + (8 \times 19^{1)}) + (12 \times 16^0) \\
&= 16384 + 2560 + 128 + 12 \\
&= (19084)_{10}
\end{aligned}$$

$(4A83)_{16} = (19084)_{10}$

Example 1.14 : Find the decimal equivalent of $(53A.0B4)_{16}$

Solution :

$$\begin{aligned}
(53A.0B4)16 &= (5 \times 16^2) + (3 \times 16^1) + (10 \times 16^0) + (0 \times 16^{-1}) \\
&\quad + (11 \times 16^{-2}) + (4 \times 16^{-3})
\end{aligned}$$

$= 1280 + 48 + 10 + 0 + 0.04927 + 0.0009765$

$= (1338.0439)_{10}$

$(53A.0B4)_{16} = (1338.0439)_{10}$

1.4.6 Decimal to Hexadecimal Conversion

The procedure for conversion from decimal no. and decimal fraction no. to hexadecimal equivalent is exactly similar to the conversion of decimal to binary no. except replacing 2 by 16.

Example 1.15 : Convert decimal $(1234.675)_{10}$ to hexadecimal.

Solution : 1st consider $(1234)_{10}$

	Remainder	
	Decimal	Hexadecimal
16 |1234	2	2
16 |77	13	D
16 |4	4	4

$(1234)_{10} = (4D2)_{16}$

Conversion of $(0.675)_{10}$:

	Decimal	Hexadecimal
$0.675 \times 16 = 10.8$	10	A
$0.800 \times 16 = 12.8$	12	C
$0.800 \times 16 = 12.8$	12	C
$0.800 \times 16 = 12.8$	12	C

$(0.675)_{10} = (0.ACC)_{16}$

Hence $(1234.675)_{10} = (4D2.ACC)_{16}$

If the decimal number is very large, it is tedious to convert the number to binary directly. So it is always advisable to convert the number into hex first, and then convert the hex to binary.

CHECK YOUR PROGRESS

Q.1. What is the largest numbar that can be represented using 8 bits?

....

Q.2. What is the weight of 1 in $(10000)_2$.

....

Q.3. Convert the following:

a) $(565.25)_{10}$ to its equivalent binary number.

b) $(256.24)_8$ to decimal equivalent.

c) $(A3B.BB)_{16}$ to decimal equivalent.

d) $(10010.110)_2$ to decimal equivalent.

e) $(3964.63)_{10}$ to octal equivalent.

1.5 COMPLEMENT OF NUMBERS

Complements are used in digital computers for simplifying the substraction operation and for logical manipulation.The complementof a binary number is obtained by inverting itsall the bits.

For example, the complement of 10011 is 01100 and 00101 is 11010 etc. The complement again depends on the base of the number.

There are two types of complements for a number of base r. These are :

- r's complement and
- (r-1)'s complement,

For example, for decimal numbers the base is 10. Therefore, complements will be 10's complement and (10–1)=9's complement. For binary numbers, the complement are 2's complement and 1's complement since base is 2.

1.5.1 (r–1)’s Complement

Giiven a number N in base r having n digits, the (r–1)’s complement of N is defined as $(r^n–1) – N$.

1’s Complement : For binary numbers, r = 2 and (r–1) = 1, so the 1’s complement of N is $(2^n–1) – N$. Again, 2^n is represented by a binary number that consists of a 1 followed by n 0’s. 2^n-1 is a binary number represented by n 1’s. For example,with n = 4, we have $2^4 = (10000)_2$ and $2^4–1 = (1111)_2$. Thus the 1’s complement of a binary number is obtained by subtracting each digit from 1.

However, the subtraction of a binary digit from 1causes the bit to change from 0 to 1 or from 1 to 0. Therefore, the 1’s complement of a binary number is formed by changing 1’s into 0 and 0’s into 1’s. For example, 1’s complement of 1010111 is 0101000.

The 7’s and 15’s complement of a number is found by subtracting each digit of the number from 7 and 15 respectively.

1.5.2 r’s Complement

The r’s complement of a 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.

Like 8’s complement and 16’s complement of a number is found by adding 1 to the LSB of the 7’s and 15’s complement of an octal and hexadecimal number respectively.

2's Complement : It is obtained by adding 1 in the 1's complement form of the binary numbers.

i.e. 2's complement of binary number = 1's complement of that number + 1

For example, 2's complement of 1010111 is 0101000 + 1 = 0101001

CHECK YOUR PROGRESS

Q.4. Find 9's complement 10's complement of decimal numbers 44 and 182.

....

....

Q.5. Find 1's complement and 2's complement of binary numbers 1101001 and 0000.

....

....

1.6 DATA REPRESENTATION

Data are usually represented by using the alphabets A to Z, numbers 0 to 9 and various other symbols. This form of representation is used to formulate problem and fed to the Computer. The processed output is required in the same form. This form of representation is called *external data representation*. However, the computer can understand data only in the form of 0's and 1's. The method of data representaion in a form suitable for storing in the memory and for processing by the CPU is called the internal data representation on digital computer.

Data, in general, are of two types : Numeric and non-numeric (Character data). The numeric data deals only with numbers and arithmatic operations and non numeric data deals with characters, names addresses etc. and non-arithmatic operations.

1.6.1 Fixed Point Representaion

A fixed point numbers in binary system uses a sign bit. A positive number has a sign bit 0 while the negative number has a sign bit 1. A negative number can be represented in one of the following ways.

– Signed magnitude representaion

– Signed 1’s complement representaion

– Signed 2’s complement representaion

Assume that the size of the register is 7 bit and the 8th position bit in used for error checking and correction or other purposes.

a) Signed magnitude representation

	+6	–6	
0	000110	1	000110
↑		↑	No change in the
Sign bit		Sign bit	magnitude, only the
			sign bit changes

b) Signed 1’s complement representation

	+6	–6	
0	000110	1	111001

Here 0 and 1 are sign bits. 1’s complement is getting for the –ve integer is by taking complement of all the bits of +ve no. including sign bit.

c) Signed 2’s complement representation

	+6	–6	
0	000101	1	111011
↑		↑	2’s complement of the
Sign bit		Sign bit	positive number
			including sing bit

The signed magnitude system is easier to interpret but computer arithmatic with this is not efficient. The circuits for handling numbers are simplified if 1’s or 2’s complement systems are used and as a result one of these is almost always adopted.

Note 1 : In 1’s and 2’s complements, all positive integers are represented as sign magnitued system.

Note 2 : When all the bits of the computer word are used to represent the number and no bit is used for signed representation, it is called unsigned representation of the number.

1.6.2 Floating Point Representation

A number which has both an integer part as well as a fractional part is called real number or floating point number. A floating point number is either positive or negative. Examples of real decimal numbers are 156.65, 0.893, –235.75, –0.253 etc. Examples of binary real numbers are 101.101, 0.11101, –1011.101, –0.1010 etc.

The first part of the number is a fixed point number which is called mantissa. It can be an integer or a fraction.

The second part specifies the decimal or binary point position and is termed exponent. It is not a physical point. Therefore, whenever we are representing a point and is termed as an exponent. It is only the assumed position. For example, for decimal o. +15.37, the typical floating point notaion is :

$51.47 = 0.5147 \times 10^2$ or 5147×10^{-2}

Now, The floating point representation of 0.5147×10^3 is :

Sign	Sign
↓	↓
|0 .5147|	|0 02|
Mantissa (fraction)	Exponent

The floating point representation of 5147×10^{-2} is

Sign	Sign
↓	↓
|0 5147|	|0 02|
Mantissa (Integer)	Exponent

Similarly for example a floating point binary number 1011.1010 can be represnted as : $1011.1010 = 0.10111010 \times 2^4$

This can be represented in a 16 bit register as follows

```
Sign                    Sign
 ↓                       ↓
|0    .10111010|        |0    000100|
Mantissa (fraction)     Exponent
```

The mantissa occupies 9 bits (1 bit for sign and 8 bits for value) and the exponent 7 bits (1 bit for sign and 6 bits for value). The binary point (.) is not physically indicated in the register, but it is only assumed (position) to be there.

In general form, the floating point numbers is expressed as :

$$N = M \text{ X } R^e$$

Where, M – Mantissa

R – Radix (or base)

e – Exponent

The mantissa M and exponent e are physically present in register. But the radix R and the point (decimal or binary point) are not indicated in the register. There are only assumed for computation and manipulation.

Normalized Floating point Number : Floating point numbers are often represented in normalized forms. A floating point number where mantissa does not contain zero as the most significant digit of the number is considered to be in normaliged form. For example, 0.00038695×10^5 and 0.0589×10^{-4} are not normaliged numbers. But 0.38695×10^2 and 0.589×10^{-5} are normaliged numbers. Similarly, for binary number also, 0.0011001×2^8 and 0.0001011×2^{-5} are not non-normaliged binary numbers. But 0.11001×2^6 and 0.1011×2^{-8} are normaliged binary numbers.

A zero cannot be normaliged as all the digits in the mantissa is zero.

Arithmatic operations involved with floating point numbers are more complex. It takes larger time for execution and requires complex hardware. But floating point representaion is frequently used in scientific calculations.

Overflow and Underflow : When the result is too small to be presented by the computer, an overflow or underflow condition exists. When two floating-point numbers of the same sign are added, a carry may be generated out of high-order bit position. This is known as mantissa overflow. In case of addition or subtraction floating point numbers are aligned. The mantissa is shifted right for the alignment of a floating point number. Sometimes, the low order bits are lost in the process of alignment. This is referred as mantissa underflow. To perform the multiplication of two floating point numbers, the exponents are added. In certain cases the sum of the exponents maybe too large and it may exceed the storing capacity of the exponent field. This is called exponent overflow. In case of division the exponent of the divisior is subtracted from the exponent of the dividend. The result of subtraction may be too small to be represented. This is called exponent underflow.

Overflow or underflow resulting from a mantissa operation can be corrected by shifting the mantissa of the result and adjusting the exponent. But the exponent overflow or underflow can not be corrected and hence, an error indication has to be displayed on the computer screen.

CHECK YOUR PROGRESS

Q.6. Represent 10 by the following fixed point representation method.

a) Signed magnitude representation.

b) Signed 1's complement representation.

c) Signed 2's complement representation.

Q.7. Represent $(1010.1010)_2$ with floating point representation method.

....

1.7 BINARY ARITHMATIC

1.7.1 Addition

Binary addition is performed in the same manner as decimal addition. Since, in binary system only two digit 0s and 1s are used, the addition will be like–

0 + 0 = 0

0 + 1 = 1 = 1+0

1 + 1 = 0, Carry 1 to the next left column

1 + 1 +1 = 1, Carry 1 to the next column.

Carry overs are performed in the same manner as in decimal arithmetic.

Example 1.16 : Add the binary numbers

(i) 1011 and 1001

(ii) 10.011 and 1.001

Solution :

	Binary no.		**Equivalent decimal no.**
(i)	11	carry	
	1011		11
	+1001		9
	10100		20
(ii)	10.011		2.375
	1.001		1.125
	11.100		3.500

Since the circuit in all digital systems actually can handle two numbers to performs addition, it is not necessary to consider the addition of more than two binary numbers. When more than two numbers to be added, the first two are added first and then their sum is added to the third and so on.

The complexity may rise when to add combination of positive and negative binary number. In this case the arithmatic addition is dependent on the representation of

a) Signed magnitude

b) Signed 1’s complement

c) Signed 2’s complement

This will be more clear if we discuss through the following example:

Example 1.17 : Add 25 and -30 in binary using 7 bit register in signed magnitude representation

a) Signed 1’s complement representaion

b) Signed 2’s complement representaion

Solution : Here, 25 is + 25 = 0011001 in binary system

–30 = 1011110 in binary system

To do the arithmatic addition with one negative number we have to check the magnitude of the numbers. The number having smaller magnitude is there subtracted from the bigger number and the sign of bigger number is selected. To implement such a scheme in digits, hardware will require a long sequence of control decisions as well as circuits that will add, compare and subtract numbers. The better attentative of arithmatic with one negative number is signed 2’s complement.

In signed 2’s complement representation :

We get that	+30 is	0	011110
	–30 is	1	011110

Now, 2’s complement of	–30 (including sign bit)	1	100010
	+25 is	0	011001

Addition

+25	0	011	001
<u>–30</u>	<u>1</u>	<u>100</u>	<u>010</u>
–05	1	111	011 (Just add the numbers)

The result for negative number will store n signed 2’s complement form. So the above result in signed 2’s complement form. So the above result in signed 2’s complemnt form including sign bit is

1 000 100 +1 = 1 000 101

Which is -05 in decimal system.

From the above example it is noticed that, signed 2’s complement representation is simpler than signed magnitude representaion. This procedure requires only one central decision and only one circuit for adding the two numbers. But it puts additional condition that the negative numbers should be stored in signed 2’s complment form in the register. This can be achieved by complementing the positive number bit by bit then incrementing the resultant by 1 to get signed 2’s complement.

In signed 1’s complement representation : This method is also simple. The rule is taht, add the two numbers including the sign bit. If carry of the most significant bit or sign bit is one, then increment the result by 1 and discard the carry over.

Addition :

+25	=	0	011 001	
$\underline{-30}$	=	$\underline{1}$	$\underline{100\ 001}$	(1’s complement of -30)
–5	=	1	111 010	

The result will store in 1’s complement format. So, 1111 010 in 1’s complement format including the sign bit is 1 000 101 which is the required result.

Example 1.18 : Add -25 and +30 using 7-bit register.

Solution :

–25	1	100	110	(1’s complement of 25)
$\underline{+30}$	$\underline{0}$	$\underline{011}$	$\underline{110}$	
+5	1 0	000	100	
	↑			

Carry bit, so add 1 to the sum and discard the carry.

This sum is now = 0 000 101 which is +5

Example 1.19 : Add –25 and –30 using 7-bit register.

Solution :

–25	1	100	110	(1’s complement of 25)
$\underline{-30}$	$\underline{1}$	$\underline{100}$	$\underline{001}$	(1’s complement of 30)
–55	1 1	000	111	
	↑			

Carry bit, so add 1 to sum and discard the carry.

Now the sum is = 1 001 000, which is –55

Since, +55 is 0 110 111

So, –55 is in 1's complement 1 001 000

The interesting feature about these representation is the representation of 0 in signed magnitude and 1's complement. There are two representation for zero are :

	+0		–0	
Signed magnitude	0	000000	1	000000
Signed 1's complement	0	000000	1	111111

But in signed 2's complement, there is just one zero and there are no positive or negative zero.

+0 000000

–0 in 2's complement is +0 = 1 111111

1

1 0 000000

$\uparrow$

discard this carry

Thus, both +0 and –0 are same in 2's complement notaton. This is an *added advantage* in favour of 2's complement notation. The maximum number which can be accomodated in registers also depends on the type of representation. In general, in a 8 bit register 1 bit is used as sign. Therefore, the rest of 7 bit are used for representing the value. The value of maximum and minumum number which can be represented are :

For signed magnitude representation 2^7 –1 to –(2^7–1)

= 128 –1 to – (128–1)

= 127 to – 127,

which is for signed 1's complement representation. For signed 2's complement representation is from + 127 to –128. The –128 is represented in signed 2's complement notation as 10000000.

1.7.2 Subtraction

Though there are other method of performing subtraction, we will consider the method of subtraction known as complementary subtracton. This is a more efficient method of subtraction while using electronics circuits.The following three steps have to follow to subtract binary numbers.

In 1's complement method :

1. Find the 1's complement of the number which is subtracting.
2. Add the number which is subtracting from with the complement value obtained from step 1.
3. If there is a carry of 1, add the carry with the result of addition. Else, take complement again of the result and attach a negative sign with the result.

Example 1.20 : Subtract 5 – 6 by 1's complement method.

Solution :	5	Binary equivalent is	101
	6	Binary equivalent is	110

Step 1 : 1's complement of 6 is 001

Step 2 : Adding 001 with 101 give the result as

$$\begin{array}{r} 001 \\ \underline{+\ 101} \\ 110 \end{array}$$

Step 3 : Since there is no carry in step 2, we take the complement again which will be 001 and after attaching negative sign the required result will be –001 which is -1.

In 2's complement method : It is same as 1's complement method except the step 3. The steps are :

Step 1 : Find the 2's complement of the number which is subtracting.

Step 2 : Add the number which is subtracting from with the comple ment value obtained from step 1.

Step 3 : If there is a carry of 1, Ignore it. Else, take 2's complement of the result again and attach a negative sign with the result.

Example 1.21 : Subtract 5 – 7 by 2's complement method.

Solution :	5	Binary equivalent is	101
	7	Binary equivalent is	111

Step 1 : The 2's complement of 7 is 000 + 1 = 001

Step 2 : Adding 001 with 101 will give result as :

$$\begin{array}{l} 001 \\ \underline{101} \\ 110 \quad \text{(No carry)} \end{array}$$

Step 3 : Since no carry, the 2's complement of 110 is 001 + 1 = 010 and attaching a negative sign the required result is –10.

Overflow : An overflow is said to have occured when the sum of two n digits number occupies (n+1) digits. This definition is valid for both binary as well as decimal digits. But what is the significance of overflow for binary numbers since it is not a problem for the cases when we add two numbers? Well, the answer lies in the limits or representation of numbers. Every computer employs a limit for representation number eg. in our examples we are using 8 bit registers of calculating the sum. But what will happen if the sum of the two numbers can be accommodated in 9 bits? Where are we going to store the 9th bit? The problem will be more clear by the following example. In case of a +ve no. added to a –ve no., the sum of result will always be smaller than the two numbers. An overflow always occurs when the added nos. are both +ve or both -ve.

Example 1.22 : Add the numbers 65 and 75 in 8 bit register in signed 2's complement notation.

Solution :	65	0	1000001
	75	0	1001011
	140	1	0001100

This is a -ve number and the 2's complement of teh result is equal to -115 which obvious is a wrong result. This has occured because of overflow.

Detection of Overflow : Overflow can be detected as :

If the carry out of the MSBs of number (or, carry into the sign bit) is equal to the carry out of the sign bit then overflow must have occured. For example

–65	1	0111111	–65	1	0111111
–15	1	1110001	–75	1	0110101
–80	1 1	0110000	–140	1 0	1110100

Carry	= 1	Carry	= 10
Carry from MSB	= 1	Carry from MSB	= 0
Carry from sign bit	= 1	Carry from sign bit	= 1
Sign bit is	= 1	Sign bit is	= 0
No overflow	Therefore, overflow		

Thus, overflow has occured, i.e. the arithmatic results so calculated have exceeded the capacity of the representation. This overflow also implies that the calculated results might be erronous.

1.8 COMPUTER CODES

A **code** is a symbol or group of symbols that represents discrete elements. Coding of characters has been standardised to enable transfer of data between computers. *Numeric data* is not the only form of data handled by a computer. We often require to process *alphanumeric data* also. An alphanumeric data is a string of symbols, where a symbol may be one of the letters A, B, C, ..., Z, or one of the digits 0, 1, 2, ..., 9, or a special character, such as + – */, . () = (space for blank)etc. However, the bits 0 and 1 must represent any data internally. Hence, computers use binary coding schemes to represent data internally. In binary coding, a group of bits represent every symbol that appears in the data. The group of bits used to represent a symbol is called a *byte*. To indicate the numbers of bits in a group, sometimes a byte is referred to as “n-bit byte”, where the group contains n bits. However, the term “byte” commonly means an 8-bit byte because most modern computers use 8 bits to represent a symbol.

1.8.1 BCD

In computing and electronic systems, **Binary-Coded Decimal** (**BCD**) is a way to express each of the decimal digits with a binary code. Its main virtue is that it allows easy conversion to decimal digits for printing or display and faster decimal calculations. Decimal numbers with their BCD equivalent are given in the table 2.4 :

Table 2.4 : BCD Code

Decimal	BCD
0	0000
1	0001
2	0010

3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Unlike binary encoded numbers, BCD encoded numbers can easily be displayed by mapping each of the nibbles(4-bits) to a different character. Conversion of decimal to BCD and BCD to decimal are shown below:

Example 1.24 : Convert the decimal numbers 47 and 180 to BCD.

Solution : 4 = 0100 and 7 = 111

47 = 0100111

Similarly, 180=0001 1000 0000

Example 1.25 : Convert each of the BCD code 1000111 to decimal.

Solution : First we have to divide the whole BCD code by a set of 4-bits from right to left. Then from left to right we can put the corresponding decimal numbers.

0100	0111
4	7

Thus, 1000111(in BCD) = 47 (in Decimal)

BCD addition : BCD is a numeric code and can be used in arithmetic operations. Addition is the most important operation because the other three operations (subtraction, multiplication, and division) can be accomplished by the use of addition. Here is how to add two BCD numbers:

Step1 : Add the two BCD numbers, using the rules for binary addition.

Step 2 : If a 4-bit sum is equal to or less than 9, it is a valid BCD number.

Step 3 : If a 4-bit sum is greater than 9, or if a carry out of the 4-bit group is generated, it is an invalid result. Add 6(0110) to the 4-bit sum in order to skip the six invalid states and return the code to 8421. If a carry results when 6 is added, simply add the carry to the next 4-bit group.

Example 1.26 : Add the following BCD numbers :

a) 0001 + 0100

b) 10000111 + 01010011

Solution : The decimal number addition are shown for comparison.

$$\text{(a)}\quad \begin{array}{r r} 0001 & 1 \\ \underline{+0100} & \underline{+4} \\ 0101 & 5 \end{array}$$

$\therefore$ 0001 + 0100 = 0101 Which is valid BCD no. (Value < 9)

$$\text{(b)}\quad \begin{array}{r r r l r} & 1000 & 0111 & & 87 \\ & \underline{+0101} & \underline{0011} & & \underline{+\ 53} \\ & 1101 & 1010 & \text{Both groups are invalid (>9)} & +140 \\ & \underline{+0110} & \underline{+0110} & \text{Add 6 (i.e.,0110) to both groups} & \\ 0001 & 0100 & 0000 & \text{Valid BCD number which is 140} & \\ & & & \text{in decimal.} & \end{array}$$