**Computer Organization &Architecture**

**Data representation:** Signed number representation, fixed and floating point representations, character representation.

***Data types:***

Binary information in digital computers is stored in memory or processor registers. Registers contain either data or control information. Data are numbers and other binary coded information that are operated on to achieve required computational results.

The data types found in the registers of digital computers may be classified as the following categories:

* 1. Numbers used in arithmetic computations
  2. Letters of the alphabet used in data processing and
  3. Other discrete symbols used for specific purposes.

The **binary number system** is the most natural system to use in a digital computer. But sometimes it is convenient to employ different number systems (**Decimal Number System, Octal Number System, Hexadecimal Number System**), especially the decimal number system, since it is used by people to perform arithmetic computations.

A ***Number system*** of base, or radix r is a system that uses distinct symbols for r digits. Numbers are represented by a string of digit symbols.

* Base or Radix for Binary Number system is 2 (It has only 2 symbols 1 and 0 only).
* Base or Radix for Decimal Number system is 10 (It has 10 symbols from 0 to 9).
* Base or Radix for Octal Number system is 8 (It has 8 symbols from 0 to 7).
* Base or Radix for Hexadecimal Number system is 16(It has 16 symbols from 0 to 9 and also from A to F).

All types of data, except binary numbers, are represented in computer registers in binary-coded form, why because registers are made up of flip-flops and they can store only l's and 0's.

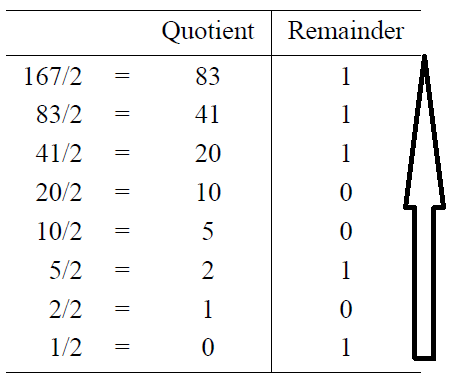
**Data Conversion:**

Converting the data from one number system to another is common in computers. Let us see how to convert data from one number system to another.

1. **Conversion from Decimal to Binary:**

1. *Divide the decimal number by the* ***base of the binary number system*** *and keep track of the quotient and remainder.*
2. *Repeatedly divide the successive quotients while keeping track of the* ***remainders*** *generated until the quotient is zero.*
3. *The remainders generated during the process, written in* ***reverse order*** *of generation from left to right, form the equivalent number in the* ***binary*** *system.*

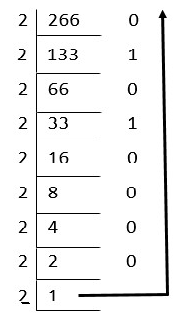
**Ex 1: Convert the decimal number 167 into its equivalent binary number**

****

The equivalent binary number for the decimal number 167 is : 10100111

i.e., (167)10 = (10100111)2

**Ex 2:** **Convert the decimal number 266 into its equivalent binary number.**



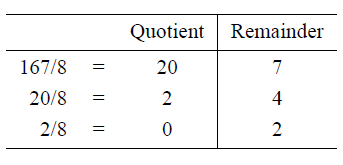
The equivalent binary number for the decimal number 266 is : 100001010

i.e., (266)10 = (100001010)2

**2. Conversion from Decimal to Octal:**

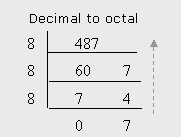
1. *Divide the decimal number by the* ***base of the octal number system*** *and keep track of the quotient and remainder.*
2. *Repeatedly divide the successive quotients while keeping track of the* ***remainders*** *generated until the quotient is zero.*
3. *The remainders generated during the process, written in* ***reverse order*** *of generation from left to right, form the equivalent number in the* ***octal system.***

**Ex 1: Convert the decimal number 167 into its equivalent octal number.**

****

The equivalent octal number is: 247 i,e., (167)10 = (247)8

**Ex 2:** **Convert the decimal number 487 into its equivalent octal number.**



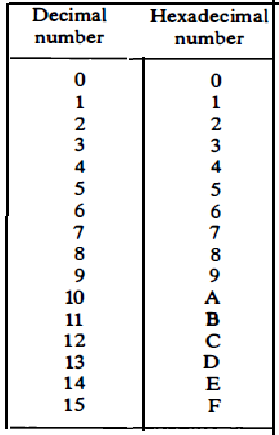
The equivalent octal number is: 747

i,e., (487)10 = (747)8

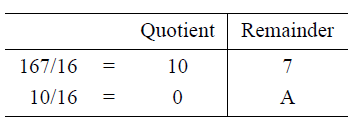
**3. Conversion from Decimal to Hexadecimal:**

1. *Divide the decimal number by the* ***base of the Hexadecimal number system*** *and keep track of the quotient and remainder.*
2. *Repeatedly divide the successive quotients while keeping track of the* ***remainders*** *generated until the quotient is zero.*
3. *The remainders generated during the process, written in* ***reverse order*** *of generation from left to right, form the equivalent number in the* ***Hexadecimal******system.***

**The below table shows decimal digits and its equivalent hexadecimal digits:**

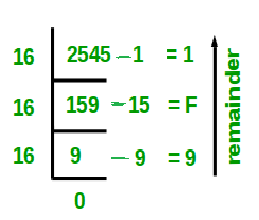


**Ex 1: Convert the decimal number 167 into its equivalent hexadecimal number.**



The equivalent Hexadecimal is: A7 i,e., (167)10 = (A7)16

**Ex 2:** **Convert the decimal number 2545 into its equivalent hexadecimal number.**



The equivalent Hexadecimal number is: 9F1 i,e., (2545)10 = (9F1)16

**4. Conversion from Binary to Decimal:**

To convert a number expressed in the binary system to the decimal system, we perform the arithmetic calculations of Equation given below, that is, multiply each digit by its weight, and add the results.

**d0b0 + d1b1 + \_ \_ \_ + dn-1bn-1 + dnbn**

where d0 represents the least significant digit (LSD) and where dn represents the most significant digit (MSD).

Here, b0, b1….. bn are binary powers and d0,d1……..dn are binary digits.

**Ex 1: Convert the binary number 10100111 into its equivalent in the decimal system.**

(10100111)2 = 1 x20 +1 x21 +1 x22 +0 x23 +0 x24 + 1 x 25 + 0 x26 + 1 x27

= 1+2+4+0+0+32+0+128

= (167)10

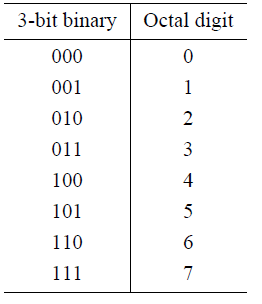
**5. Conversion from Binary to Octal:**

To convert a number in the binary system to the octal system,

1. *Form 3-bit groups starting from the right of given binary number.*
2. *Add extra 0s at the left-hand side of the binary number if the number of bits is not a multiple of 3.*
3. *Then replace each group of 3 bits by its equivalent octal digit.*

**Why three bit groups? Simply because 23 = 8**

The below table shows the 3-bit binary and its equivalent octal digits,



**Ex 1: Convert the binary number 10100111 into its equivalent in the octal system.**

(10100111)2 = 

= (247)8

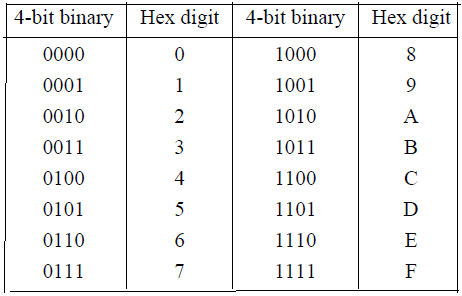
**6. Conversion from Binary to Hexadecimal:**

To convert a number in the binary system to the hexadecimal system,

1. *Form 4-bit groups starting from the right of given binary number.*
2. *Add extra 0s at the left-hand side of the binary number if the number of bits is not a multiple of 4.*
3. *Then replace each group of 4 bits by its equivalent hexadecimal digit.*

**Why four bit groups? Simply because 24 = 16**

The below table shows the 4-bit binary and its equivalent hexadecimal digits,



**Ex 1: Convert the binary number 1101011111 into its equivalent hexadecimal number.**

(1101011111)2 = 

= (35F)16

**7. Conversion from Octal to Binary:**

To convert a number in the octal system to the binary system,

* For each octal digit, write the 3-bit equivalent binary groups.
* We should write exactly 3 bits for each octal digit even if there are leading ‘0’s. For example, for octal digit 0, write the three bits 000.

**Ex 1: Convert (247)8 into its equivalent Binary Number**.

(247)8 = 

= (010100111)2

**Ex 2:** **Convert (105)8 into its equivalent Binary Number.**

(105)8 = 

= (001000101)2

**8. Conversion from Octal to Decimal:**

To convert a number expressed in the octal system to the decimal system, we perform the arithmetic calculations of Equation given below, that is, multiply each digit by its weight, and add the results.

**d0b0 + d1b1 + \_ \_ \_ + dn-1bn-1 + dnbn**

where d0 represents the least significant digit (LSD) and where dn represents the most significant digit (MSD).

Here, b0, b1….. bn are octal powers and d0,d1……..dn are octal digits.

**Ex 1: Convert the octal number 247 into its equivalent decimal number.**

(247)8 = 7X80 + 4X81 + 2X82

= 7+32+128

= 167

(247)8 = (167)10

**9. Conversion from Hexadecimal to Binary:**

To convert a number in the hexadecimal system to the binary system,

* For each hexadecimal digit, write the 4-bit equivalent binary groups.
* We should write exactly 4 bits for each hexadecimal digit even if there are leading ‘0’s. For example, for decimal hexadecimal digit 0, write the 4 bits 0000.

**Ex 1: Convert (247)16 into its equivalent Binary Number.**

(247)16 = 2 4 7

0010 0100 0111

= (001001000111)2

**10. Conversion from Hexadecimal to Decimal:**

To convert a number expressed in the hexadecimal system to the decimal system, we perform the arithmetic calculations of Equation given below, that is, multiply each digit by its weight, and add the results.

**d0b0 + d1b1 + \_ \_ \_ + dn-1bn-1 + dnbn**

where d0 represents the least significant digit (LSD) and where dn represents the most significant digit (MSD).

Here, b0, b1….. bn are hexadecimal powers, d0,d1……..dn are hexadecimal digits.

**Ex 1: convert the hexadecimal number A7 into its equivalent decimal number.**

(A7)16 = 7X160 + AX161

= 7 + 160

= 167

(A7)16 = (167)10

**Conversion from Octal to Hexadecimal and Vice-Versa:**

we don’t normally require conversion between hex and octal numbers. If we need to do this, use binary as the intermediate form, as shown below:

**Hex => Binary => Octal**

**Octal => Binary => Hex**

**11. Conversion from Octal to Hexadecimal:**

**Ex 1: Convert the octal number 247 into its equivalent hexadecimal number.**

i. First we need to convert the given octal number 247 into binary,

(247)8 = 

= (010100111)2

ii. Next we need to convert the resultant binary number 010100111 into hexadecimal ,

(010100111)2 = 0000 1010 0111

0 A 7

= (0A7)16

Here, we can ignore the leading hexadecimal zero’s at MSD positions, So answer will be (A7)16

**12. Conversion from Hexadecimal to Octal :**

**Ex 1: Convert the hexadecimal number A7 into its equivalent octal number.**

1. First we need to convert the given hexadecimal number A7 into binary,

(A7)16  = A 7

1010 0111

= (10100111)2

ii. Next we need to convert the resultant binary number 10100111 into octal ,

(10100111)2 = 010 100 111

2 4 7

= (247)8

***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:

1. r's complement
2. (r - 1)'s complement.

When the value of the **base r** is substituted in the name,

1. For binary numbers, **2's and 1's complement**
2. For decimal numbers, **10's and 9's complement.**
3. For octal numbers, **8's and 7's complement**
4. For hexadecimal numbers, **16's and F's complement**

**(r-1)’s Complement ( 9's complement):**

Given a number N in base r having n digits,

* **(r - 1)'s complement of N = (rn - 1) - N.**
* For decimal numbers r = 10 and r - 1 = 9, so the **9's complement** of N is (10n - 1) - N. Now, 10n represents a number that consists of a single 1 followed by n 0's. 10n - 1 is a number represented by n 9's.

If n = 4, we have 104 = 10000 and 104 - 1 = 9999. Then the 9' s complement of a decimal number is obtained by subtracting each digit from 9.

Ex:

* 9's complement of 546700 is 999999 - 546700 = 453299
* 9's complement of I2389 is 99999 - 12389 = 87610.

**(r-1)’s Complement (1's complement):**

Given a number N in base r having n digits,

* **(r - 1)'s complement of N = (rn - 1) - N.**
* For binary numbers r = 2 and r - 1 = 1, so the **1's complement** of N is ( 2n - 1) - N. Now, 2n represents a number that consists of a single 1 followed by n 0's. 2n - 1 is a number represented by n 1's.

If n = 4, we have 24 = 10000 and 24 - 1 = 1111. Then the 1' s complement of a binary number is obtained by subtracting each digit from 1.

* However, the subtraction of a binary digit from 1 causes 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's and 0's into 1's.

**Therefore, the 1's complement of a binary number is formed by changing 1's into 0's and 0's into 1's.**

Ex:

* 1's complement of 1011001 is 0100110.
* 1's complement of 0001111 is 1110000.

**r’s Complement:**

The r's complement of an n-digit number N in base r is defined as rn - N for N ≠ 0 and 0 for N = 0.

* Comparing with the (r - I)'s complement, the r's complement is obtained by adding 1 to the (r - 1)'s complement.
* so , **rn- N = [(rn- 1) - N] + 1**.

Ex:

* **10's complement** of 2389 is 7610 + 1 = 7611. It is obtained by adding 1 to the 9' s complement value of 2389.
* **2's complement** of binary 101100 is 010011 + 1 = 010100 and is obtained by adding 1 to the 1's complement value of 101100.

***Fixed-Point Representation:***

* Positive integers, including zero, can be represented as unsigned numbers. However, to represent negative integers, we need a notation for negative values.
* Computers must represent everything with 1's and 0's, including the sign of a number. the sign 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**.

**binary point**:

In addition to the sign, a number may have a binary (or decimal) point.

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:

1. **By giving it a fixed position (**or)
2. **By employing a floating-point representation.**

**Fixed-point representation:** Thismethod 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. In both cases, the binary point is not actually present, but its presence is assumed from the fact that the number stored in the register is treated as a fraction or as an integer.

**Floating-point representation:** uses a second register to store a number that designates the position of the decimal point in the first register.

**Integer Representation (Signed numbers):**

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:

1. Signed-magnitude representation

2. Signed-1' s complement representation

3. 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.

**Example:**  Representation of -14 with 8-bit register in 3-ways.

Signed-magnitude representation: 1 0001110

Signed-1' s complement representation: 1 1110001

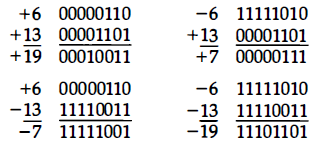
Signed 2' s complement representation: 1 1110010

* The signed-magnitude representation of - 14 is obtained from +14 by complementing only the sign bit.
* The signed-1's complement representation of –14 is obtained by complementing all the bits of +14, including the sign bit.
* The signed-2' s complement representation is obtained by taking the 2' s complement of +14, including its sign bit.

**Arithmetic Addition(2’s Complement Addition):**

* Add the two numbers, including their sign bits, and discard any carry out of the sign (leftmost) bit position.
* Note that negative numbers must initially be in 2' s complement and if the sum obtained after the addition is negative, it is in 2's complement form.

**Ex:**

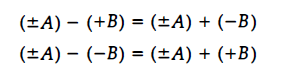


**Arithmetic Subtraction(2’s Complement Subtraction):**

* Subtraction of two signed binary numbers when negative numbers are in 2' s complement form is 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.***

* A subtraction operation can be changed to an addition operation if the sign of the subtrahend is changed. This is demonstrated by the following relationship:



**Example:**

Consider the subtraction of (-6) - (- 13) = +7.

In binary with eight bits this is written as 11111010 – 11110011.

The subtraction is changed to addition by taking the 2's complement of the subtrahend (- 13) to give (+13).

In binary this is 11111010 + 00001101 = 100000111 . Removing the end carry, we obtain the correct answer 00000111 ( + 7).

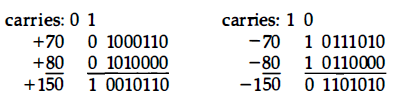
***Overflow***

* When two numbers of n digits each are added and the sum occupies n + 1 digits, we say that an **overflow** occurred. When the addition is performed with paper and pencil, an overflow is not a problem since there is no limit to the width of the page to write down the sum.
* An overflow is a problem in computers because the width of registers is finite. A result that contains n + 1 bits cannot be accommodated in a register with a length of n bits.

An overflow cannot occur after an addition if one number is positive and the other is negative,

An overflow may occur if the two numbers added are both positive or both negative.

To see how this can happen, consider the following example. Two signed binary numbers, + 70 and + 80, are stored in two 8-bit registers. The range of numbers that each register can accommodate is from binary + 127 to binary - I28. Since the sum of the two numbers is + I50, it exceeds the capacity of the 8-bit register.



An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position. If the two carries are applied to an **exclusive-OR** gate, an overflow will be detected when the output of the gate is equal to 1.

***Floating-Point Representation:***

**Floating-point representation:** Ituses a second register to store a number that designates the position of the decimal point in the first register.

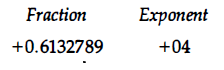
The floating-point representation of a number has two parts.

1. The first part represents a signed & fixed-point number called the **mantissa.**

2. The second part designates the position of the decimal (or binary) point and is called the **exponent**.

**Example:**

The decimal number + 6132.789 is represented in floating-point with a fraction and an exponent as follows:



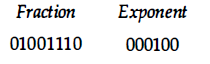
* This representation is equivalent to the scientific notation +0. 6132789 X 10+4.
* Floating-point is always interpreted to represent a number in the following form:

**m x re**

m- mantissa, r-radix or base of the given number and e-Exponent.

A floating-point binary number is represented in a similar manner except that it uses base 2 for the exponent.

**Ex: + 1001 . 11 is represented with a n 8-bit fraction and 6-bit exponent.**



m x re = + (.1001110) x 2+4

**Normalization:** A floating-point number is said to be normalized if the most significant digit of the mantissa is nonzero. For example, the decimal number 350 is normalized but 00035 is not.

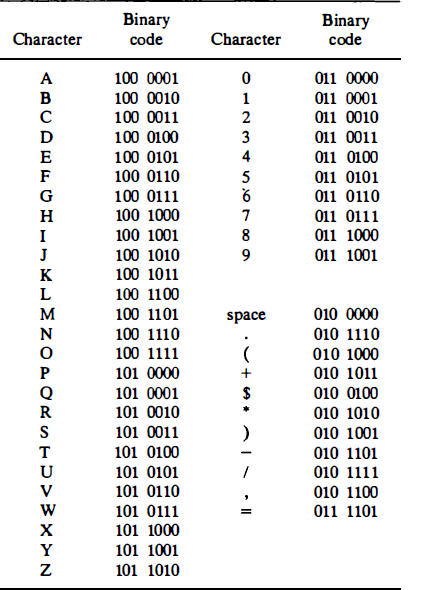
***Character Representation:***

Many applications of digital computers require the handling of data that consist not only of numbers, but also letters of the alphabet and certain special characters.

An ***alphanumeric character***set is a set of elements that includes the 10 decimal digits, the 26 letters of the alphabet and a number of special characters, such as $, + , and = .

The standard alphanumeric binary code is the **ASCII** **(American Standard Code for Information Interchange)**, which uses seven bits to code 128 characters. The binary code for the uppercase letters, the decimal digits, control characters, operators and a few special characters is listed in Table shown below.

Some of the ASCII codes are shown in the below table



* Another alphanumeric code is the **EBCDIC (Extended BCD Interchange Code).** It uses eight bits for each character (and a ninth bit for parity). EBCDIC has some more characters than ASCII and the same character symbols as ASCII but the bit assignment to characters is different.
* There is another new international information exchange code called **Unicode.** It uses 16-bits, it has the capacity to encode the majority of characters used in every language of the world.