Data representation

In computer the data and instruction is stored in memory using binary code (or machine code) represented by **B**inary dig**IT**'s 1 and 0 called **BIT**'s.

The number system uses well defined symbols called digits. Number systems are basically classified into two types. They are:

o Non-positional number system
o Positional number system

➢ Non-Positional Number System

• In olden days people use of this type of number system for simple calculations like additions and subtractions.

• The non-positional number system consists of different symbols that are used to represent numbers.

• Roman number system is an example of the non-positional number system
i.e. I=1, V=5, X=10,L=50.

➢ Positional Number System

• This type of number system are:
o Decimal number system
o Binary number system
o Octal number system
o Hexadecimal number system

•The total number of digits present in any number system is called its **Base** or **Radix**.

• Every number is represented by a base (or radix) x, which represents x digits.

• The base is written after the number as subscript such as $512_{(10)}$.It is a Decimal number as its base is 10.

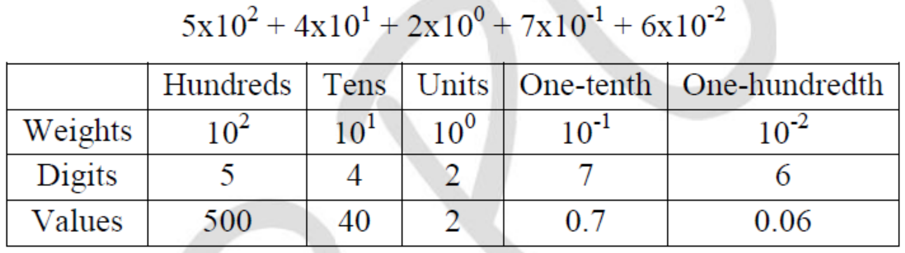
• To determine the quantity that the number represents, the number is multiplied by an integer power
of x depending on the position it is located and then finds the sum of the weighted digits.

• **Example**: Consider a decimal number $512.45_{(10)}$ which can be represented in equivalent value as:

$5x10^2 + 1x10^1 + 2x10^0 + 4x10^{-1} + 5x10^{-2}$

➢ Decimal Number System

• It is the most widely used number system.
• The decimal number system consists of 10 digits from 0 to 9.
• It has 10 digits and hence its base or radix is 10.
• Example: $123_{(10)}$, $456_{(10)}$, $7890_{(10)}$.
• Consider a decimal number **$542.76_{(10)}$** which can be represented in equivalent value as:



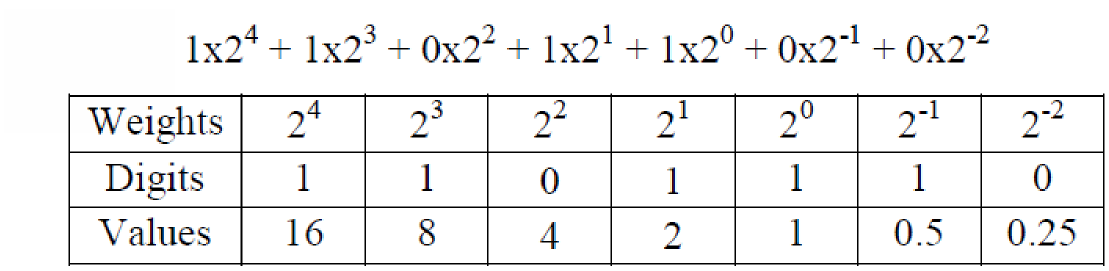
$$5x10^2 + 4x10^1 + 2x10^0 + 7x10^{-1} + 6x10^{-2}$$

	Hundreds	Tens	Units	One-tenth	One-hundredth
Weights	10^2	10^1	10^0	10^{-1}	10^{-2}
Digits	5	4	2	7	6
Values	500	40	2	0.7	0.06

➢ Binary Number System

- Digital computer represents all kinds of data and information in the binary system.
- Binary number system consists of two digits 0 (low voltage) and 1 (high voltage).
- Its base or radix is 2.
- Each digit or bit in binary number system can be 0 or 1.
- The positional values are expressed in power of 2.
- Example: $1011_{(2)}$, $111_{(2)}$, $100001_{(2)}$

Consider a binary number **$11011.10_{(2)}$**which can be represented in equivalent value as:



$$1x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 + 0x2^{-1} + 0x2^{-2}$$

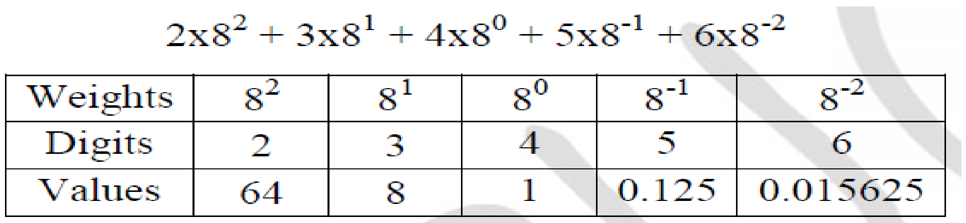
Weights	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}
Digits	1	1	0	1	1	1	0
Values	16	8	4	2	1	0.5	0.25

Note: In the binary number $11010_{(2)}$

- The left most bit 1 is the highest order bit. It is called as **Most Significant Bit (MSB)**.
- The right most bit 0 is the lower bit. It is called as **Least Significant Bit (LSB)**.

➢ Octal Number System

- The octal number system has digits starting from 0 to 7.
- The base or radix of this system is 8.
- The positional values are expressed in power of 8.
- Any digit in this system is always less than 8.
- Example: $123_{(8)}$, $236_{(8)}$, $564_{(8)}$
- The number 6418 is not a valid octal number because 8 is not a valid digit.
- Consider a Octal number **$234.56_{(8)}$** which can be represented in equivalent value as:

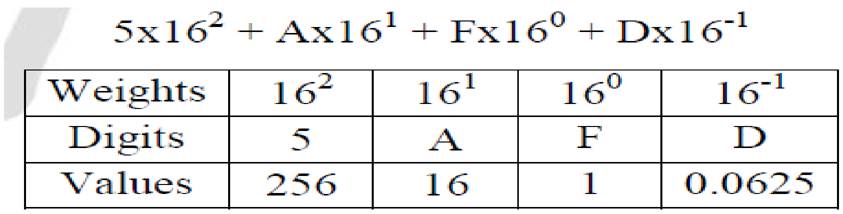


$$2x8^2 + 3x8^1 + 4x8^0 + 5x8^{-1} + 6x8^{-2}$$

Weights	8^2	8^1	8^0	8^{-1}	8^{-2}
Digits	2	3	4	5	6
Values	64	8	1	0.125	0.015625

➢ Hexadecimal Number System

- The hexadecimal number system consists of 16 digits from 0 to 9 and A to F.
- The letters A to F represent decimal numbers from 10 to 15.
- That is, ‘A’ represents 10, ‘B’ represents 11, ‘C’ represents 12, ‘D’ represents 13, ‘E’ represents 14 and ‘F’ represents 15.
- The base or radix of this number system is 16.
- Example: A4 $_{(16)}$, $1AB_{(16)}$, $934_{(16)}$, C $_{(16)}$
- Consider a Hexadecimal number **$5AF.D_{(16)}$**which can be represented in equivalent value as



$5x16^2 + Ax16^1 + Fx16^0 + Dx16^{-1}$

Weights	16^2	16^1	16^0	16^{-1}
Digits	5	A	F	D
Values	256	16	1	0.0625

Number system conversion: to convert decimal without fraction (whole number) to any other number system we have to undergo following sequence of steps

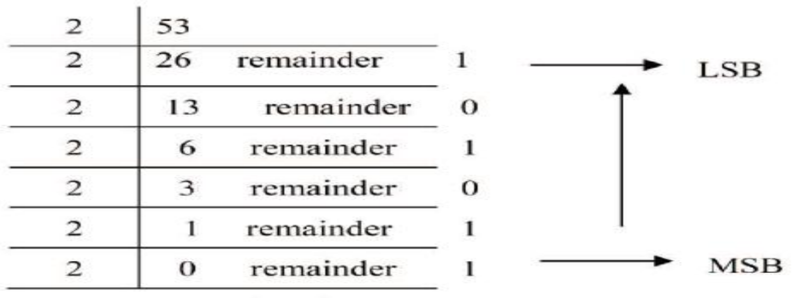
Step1: Divide the given decimal number by the base value to which you want to convert

Step2: Note down the quotient and reminder

Step3: repeat step 1 and 2 until quotient become zero or not possible to divide

step4: The first reminder is LSB and last reminder is the MSB. Write the answer from MSB and LSB

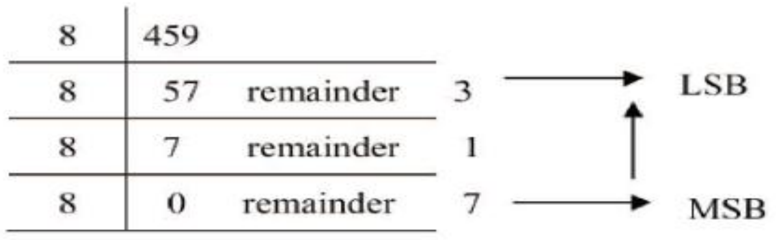
Consider the decimal number $53_{(10)}$ which can be represented in binary as:



2	53		
2	26	remainder	1
2	13	remainder	0
2	6	remainder	1
2	3	remainder	0
2	1	remainder	1
2	0	remainder	1

Therefore, $53_{(10)} = 110101_{(2)}$

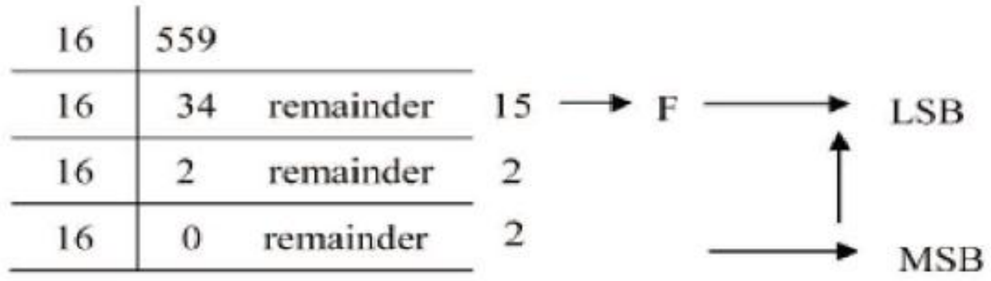
Convert the decimal number $459_{(10)}$ to octal



8	459		
8	57	remainder	3
8	7	remainder	1
8	0	remainder	7

Therefore, $459_{(10)} = 713_{(8)}$

Convert the decimal number $559_{(10}$ to hexadecimal



16	559		
16	34	remainder	15 → F → LSB
16	2	remainder	2
16	0	remainder	2 → MSB

Therefore, $559_{(10)} = 22F_{(16)}$

Decimal fraction to any other type conversion: We have to undergo following sequence of steps

Step1: Multiply the decimal fraction by the base value to which you want to convert and note down carry and product

Step2: Repeat step1 until the fractional product become zero

Step3: The first carry will be MSB and last carry will be LSB

Consider the decimal fraction $0.3125_{(10)}$

Multiply by 2	Carry	Product
0.3125 x 2	0 (MSB)	0.625
0.625 x2	1	0.25
0.25 x 2	0	0.50
0.50 x2	1 (LSB)	0.00
0.00		

Therefore, $0.3125_{(10)} = 0.0101_{(2)}$

Any other number system to decimal conversion: We have to undergo following sequence of steps

Step1: Multiply each bit of the number by its positional weight

Step2: Add all the product

Consider the binary number $11011.101_{(2)}$ which can be represented in decimal value as:

$$1x2^4+1x2^3+0x2^2+1x2^1+1x2^0+1x2^{-1}+0\times2^{-2}+1x2^{-3}$$

Therefore, $11011.101_{(2)} = 27.625_{(10)}$

Consider an octal number $234.56_{(8)}$ which can be represented in decimal value as:

$2x8^2 + 3x8^1 + 4x8^0 + 5x8^{-1} + 6x8^{-2}$ Therefore, $234.56_{(8)} = 156.71875_{(10)}$

Consider a hexadecimal number $5AF.D_{(16)}$ which can be represented in decimal value as:

$5x16^2 + A \text{ x } 16^1 + F \text{ x } 16^0 + D \text{ x } 16^{-1}$ Therefore, $5AF.D_{(16)} = 1455.8125_{(10)}$

Binary to octal conversion: The binary digits are grouped into group of three bits starting from binary point and convert each group into its equivalent octal number. For whole numbers it may be necessary to add a zero as the MSB. Similarly when representing fractions , it may be necessary to add a trialing zero in the LSB to complete grouping of three bits

Consider the binary number $1010111_{(2)}$

$$\underbrace{1}_{1} \quad \underbrace{010}_{2} \quad \underbrace{111}_{7}$$

Therefore, $1010111_{(2)} = 127_{(8)}$

Consider the binary number $0.110111_{(2)}$

$$\underbrace{110}_{6} \quad \underbrace{111}_{7}$$

Therefore, $0.110111_{(2)} = 0.67_{(8)}$

Octal to binary conversion: Each octal digit is represented by a three bit binary number as in table

Octal digit	Binary digit
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Consider the octal number $456_{(8)}$

$$4 \longrightarrow 100$$
$$5 \longrightarrow 101$$
$$6 \longrightarrow 110$$

Therefore, $456_{(8)} = 100101110_{(2)}$

Consider the octal number $73.16_{(8)}$

$$7 \longrightarrow 111$$
$$3 \longrightarrow 011$$
$$1 \longrightarrow 001$$
$$6 \longrightarrow 110$$

Therefore, $73.16_{(8)} = 111011.001110_{(2)}$

Binary to hexadecimal conversion: The binary digits are grouped into group of four bits starting from binary point and convert each group into its equivalent hexadecimal number. For whole numbers it may be necessary to add a zero as the MSB. Similarly when representing fractions , it may be necessary to add a trialing zero in the LSB to complete grouping of four bits

The following table shows hexadecimal number and its equal 4 bit binary value

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Consider a binary number $1011001_{(2)}$

$$\underbrace{0101}_{5} \qquad \underbrace{1001}_{9}$$

Therefore, $1011001_{(2)} = 59_{(16)}$

Consider a binary number $0.11010111_{(2)}$

$$\underbrace{1101}_{D} \qquad \underbrace{0111}_{7}$$

Therefore, $0.11010111_{(2)} = 0.D7_{(16)}$

Hexadecimal to binary conversion: Each digit of hexadecimal number is replaced by a 4 bit number.

Consider a hexadecimal number $CEBA_{(16)}$ $=1100111010111010_{(2)}$

C→12→1100

E→14→1110

B→11→1011

A→10→1010

Octal to hexadecimal conversion:

Step1: Write the binary equivalent of each octal digit

Step2: Regroup them into 4 bits from the right side

Step3: Convert each group into its equivalent hexadecimal digit

Example: Convert $274_{(8)}$ to hexadecimal

274→ 010111100→$BC_{(16)}$

Hexadecimal to octal conversion:

Step1: Write the binary equivalent of each hexadecimal digit

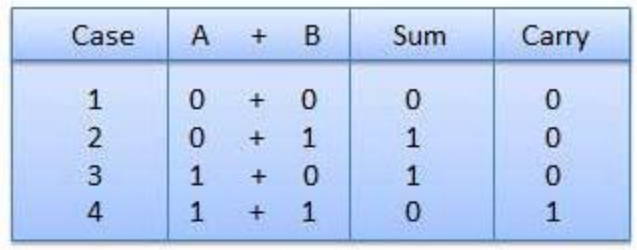
Step2: Regroup them into 3 bits from the right side

Step3: Convert each group into its equivalent octal digit

Example: Convert $FADE_{(16)}$ to octal

FADE→ 1111101011011110→$175336_{(2)}$

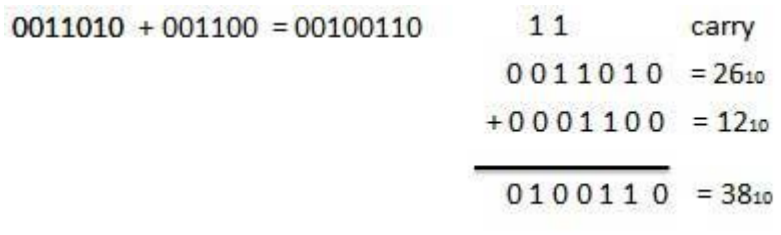
Binary addition: The addition of two binary numbers is performed in same manner as the addition of decimal number. The basic rules of binary addition are



Case	A + B	Sum	Carry
1	0 + 0	0	0
2	0 + 1	1	0
3	1 + 0	1	0
4	1 + 1	0	1

In fourth case, a binary addition is creating a sum of (1 + 1 = 10) i.e. 0 is written in the given column and a carry of 1 over to the next column.
Note: A binary addition of 1+1+1 is creating sum as 1 and carry is also 1

Example: Add $26_{(10)}$-$12_{(10)}$ in binary



0011010 + 001100 = 00100110

```
   1 1          carry
 0 0 1 1 0 1 0  = 26₁₀
+0 0 0 1 1 0 0  = 12₁₀
---------------
 0 1 0 0 1 1 0  = 38₁₀
```

Add binary number 1011.011 and 1101.111

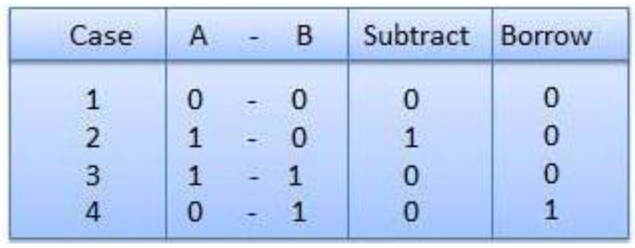
Carry 1 1 1 1 1 1

Append 1 1 0 1 1 . 0 1 1

Append 2 1 1 0 1 . 1 1 1

Sum **1 1 0 0 1 . 0 1 0**

Binary subtraction: The subtraction of two binary numbers is performed in same manner as the subtraction of decimal number. The basic rules of binary subtractions are



Case	A - B	Subtract	Borrow
1	0 - 0	0	0
2	1 - 0	1	0
3	1 - 1	0	0
4	0 - 1	0	1

Example: subtract $26_{(10)}$-$12_{(10)}$ in binary

0011010 - 001100 = 00001110

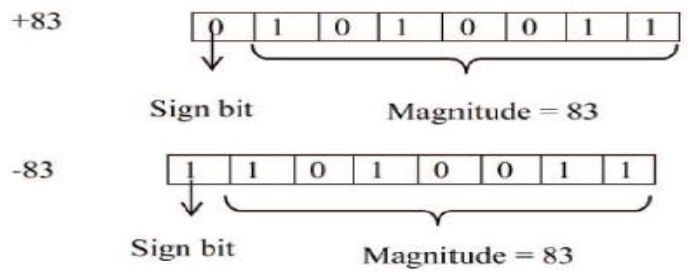
$$\begin{array}{r l} 1\ 1\quad\ \ & \text{borrow} \\ 0\,0\,\cancel{1}\,\cancel{1}\,0\,1\,0 & = 26_{10} \\ -0\,0\,0\,1\,1\,0\,0 & = 12_{10} \\ \hline 0\,0\,0\,1\,1\,1\,0 & = 14_{10} \end{array}$$

Representation of signed integers: The digital computer will handle both positive and negative integers. The sign of the number is represented by prefixing + or – sign. This is done by adding left most bit to the number called sign bit. If the number is positive then sign bit is 0 and the number is negative then the sign bit is 1. Hence it is also called fixed point representation. A negative signed integer can be represented

a) **sign and magnitude representation**: An integer containing a sign bit followed by magnitude bit is known as sign magnitude representation. The MSB is always sign bit and remaining is magnitude bit

Example:

If in a Computer of word size of 1 byte (8 bits), then an integer +83 and -83 is represented in 8-bit sign-magnitude representation as



Note: If computer word size is 7 bit then 2^7=128 so by sign and magnitude we can represent the number -128 to +127

b) **One's compliment form**: It is the simplest form, here we can convert to one's complemented value of binary number by converting each zero to one and one to zero .

Thus the one's complemented value of 101000 is 010111

c) Two's complement form: The two's complemented binary number can be obtained by adding one to the one's complimented binary number. Consider the binary 101000 its 2'complemented value is obtained as 1's complemented value of 101000+1 i.e. 010111+1=011000

Subtraction using 1’s complemented form: we have to undergo following steps

Step 1:convert given integer to binary

step 2: identify minuend and subtrahend

step 3: write down 1’s complement of the subtrahend.

step 4: Add 1’s complemented subtrahend with the minuend.

step 5: observe the result and do one of the following step

a) If the result of addition has a carry-over then it is dropped and an 1 is added to LSB

b) If there is no carry over, then write 1’s complement of the result of addition and put negative sign

Subtraction using 2’s complemented form: we have to undergo following steps

Step 1:convert given integer to binary

step 2: identify minuend and subtrahend

step 3: write down 2’s complement of the subtrahend.

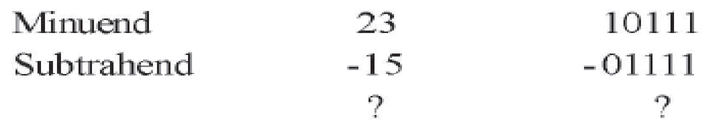
step 4: Add 2’s complemented subtrahend with the minuend.

step 5: observe the result and do one of the following step

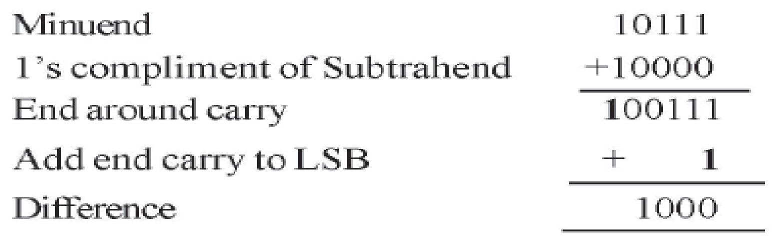
a) If the result of addition has a carry-over then it is discard he carry and the remaining bit is the difference

b) If there is no carry over, then write 2’s complement of the result of addition and put negative sign

Example: Subtract 15 from 23 using 1’s complemented method



Minuend	23	10111
Subtrahend	-15	-01111
	?	?



Minuend	10111
1’s compliment of Subtrahend	+10000
End around carry	100111
Add end carry to LSB	+ 1
Difference	1000

Example: subtract 52 from 25 using 1’s complement

```
Minuend              25          011001
Subtrahend          -52        - 110100
                   -----       --------
                     ?              ?
                   -----       --------


Minuend                              011001
1’s compliment of Subtrahend       + 001011
                                   --------
                                     100100
                                   --------
```

There is no carry. Therefore re-complement and insert a negative sign, we get - 011011

Example: subtract 9 from 17 using 2’s complement

```
Minuend              17       10001
Subtrahend          - 9      -01001
                   -----     -------
                     ?           ?
                   -----     -------
```

2’s complement of 01001 = 1’s complement of 01001 + 1

= 10110 + 1

= 10111

```
Minuend                             10001
2’s complement of Subtrahend       +10111
                                  -------
                                   101000
                                  -------
```

Carry is generated. Discarding the carry, we get 1000

Example: Subtract 47 from 26 using 2’s complement method

$$\begin{array}{lrr}
\text{Minuend} & 26 & 011010 \\
\text{Subtrahend} & -\,47 & -\,101111 \\
\hline
 & ? & ? \\
\hline
\end{array}$$

$$\begin{aligned}
\text{2's complement of } 101111 &= \text{1's complement of } 101111 + 1 \\
&= 010000 + 1 \\
&= 010001
\end{aligned}$$

$$\begin{array}{lr}
\text{Minuend} & 011010 \\
\text{2's complement of Subtrahend} & +\,010001 \\
\hline
 & 101011 \\
\hline
\text{2's complement of the answer} = & -10101
\end{array}$$

Note: In the above example there is no carry so answer should represent in 2’s complement form and put minus sign

Computer codes: The computer code is used to internal representation of data in computer. The most commonly used computer codes are

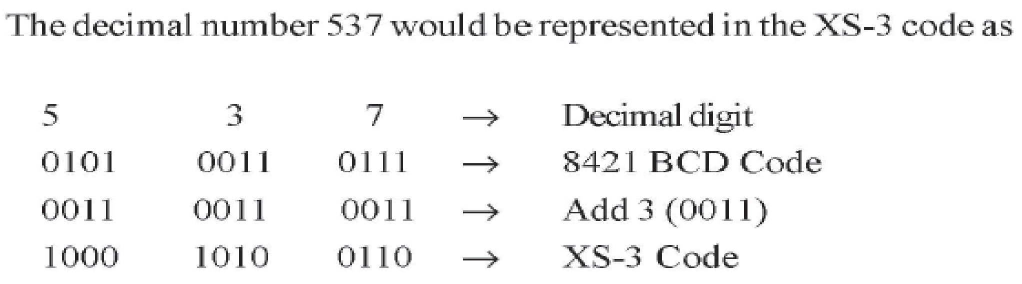
a) Binary coded decimal (BCD): In this code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code. In the BCD, with four bits we can represent sixteen numbers. It is insufficient to represent various character used in by the computer, hence 6 bit BCD code was developed by adding two zone position. With six bit it is possible to represent 64 character. For example

The decimal number 537 is represented in 4-bit 8421 BCD code as

$$\overbrace{0101}^{5}\ \overbrace{0011}^{3}\ \overbrace{0111}^{7}$$

b) Excess-3 BCD code or XS-3 code: It is a non weighted code and is obtained from

8421 BCD code by adding 3(0011)



The decimal number 537 would be represented in the XS-3 code as

5	3	7	→	Decimal digit
0101	0011	0111	→	8421 BCD Code
0011	0011	0011	→	Add 3 (0011)
1000	1010	0110	→	XS-3 Code

c) Extended Binary Coded Decimal Interchange Code(EBCDIC): This coding was developed by IBM. It is a 8 bit code so it has total 256 possible code group.

d) American standard Code for Information Interchange(ASCII): It is a 7 bit code so using this we can represent total 128 character. This code is used to represent alpha numeric and some special character. The ASCII value of A is 65