Chapter-3

DATA REPRESENTATION

➢ Introduction

• In Digital Computer, data and instructions are stored in computer memory using binary code (or machine code) represented by Binary digit’s 1 and 0 called BIT’s.
• The data may contain digits, alphabets or special character, which are converted to bits, understandable by the computer.
• 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.
• This number system cannot be used effectively to perform arithmetic operations.

➢ 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 \( 512.45 \) which can be represented in equivalent value as:

\[
5 \times 10^2 + 1 \times 10^1 + 2 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2}
\]

**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.
- These digits can be used to represent any numeric value.
- Example: \( 123, 456, 7890 \).

Consider a decimal number \( 542.76 \) which can be represented in equivalent value as:

\[
5 \times 10^2 + 4 \times 10^1 + 2 \times 10^0 + 7 \times 10^{-1} + 6 \times 10^{-2}
\]

<table>
<thead>
<tr>
<th>Weights</th>
<th>Hundreds</th>
<th>Tens</th>
<th>Units</th>
<th>One-tenth</th>
<th>One-hundredth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td>10^2</td>
<td>10^1</td>
<td>10^0</td>
<td>10^{-1}</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>Values</td>
<td>500</td>
<td>40</td>
<td>2</td>
<td>0.7</td>
<td>0.06</td>
</tr>
</tbody>
</table>

**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: \( 1011, 111, 100001 \).

Consider a binary number \( 11011.10 \) which can be represented in equivalent value as:

\[
1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2}
\]

<table>
<thead>
<tr>
<th>Weights</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
<th>2^{-1}</th>
<th>2^{-2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Values</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0.5</td>
<td>0.25</td>
</tr>
</tbody>
</table>

**Note:** In the binary number \( 11010 \),

- 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:

\[
2 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 + 5 \times 8^{-1} + 6 \times 8^{-2}
\]

<table>
<thead>
<tr>
<th>Weights</th>
<th>8(^2)</th>
<th>8(^1)</th>
<th>8(^0)</th>
<th>8(^{-1})</th>
<th>8(^{-2})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td>2 (\times)</td>
<td>3 (\times)</td>
<td>4 (\times)</td>
<td>5 (\times)</td>
<td>6 (\times)</td>
</tr>
<tr>
<td>Values</td>
<td>64</td>
<td>8</td>
<td>1</td>
<td>0.125</td>
<td>0.015625</td>
</tr>
</tbody>
</table>

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.
- 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:

\[
5 \times 16^2 + A \times 16^1 + F \times 16^0 + D \times 16^{-1}
\]

<table>
<thead>
<tr>
<th>Weights</th>
<th>16(^2)</th>
<th>16(^1)</th>
<th>16(^0)</th>
<th>16(^{-1})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td>5 (\times)</td>
<td>A (\times)</td>
<td>F (\times)</td>
<td>D (\times)</td>
</tr>
<tr>
<td>Values</td>
<td>256</td>
<td>16</td>
<td>1</td>
<td>0.0625</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number System</th>
<th>Base</th>
<th>Symbol used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>2</td>
<td>0, 1</td>
</tr>
<tr>
<td>Octal</td>
<td>8</td>
<td>0,1,2,3,4,5,6,7</td>
</tr>
<tr>
<td>Decimal</td>
<td>10</td>
<td>0,1,2,3,4,5,6,7,8,9</td>
</tr>
</tbody>
</table>
| Hexadecimal   | 16   | 0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F  
where A=10; B=11; C=12; D=13; E=14; F=15 |
Number System Conversions

Conversion from Decimal to Binary:

1. Steps to convert decimal number to binary number:
   - Step 1: Divide the given decimal number by 2.
   - Step 2: Take the remainder and record it on the right side.
   - Step 3: Repeat the Step 1 and Step 2 until the decimal number cannot be divided further.
   - Step 4: The first remainder will be the LSB and the last remainder is the MSB. The equivalent binary number is then written from left to right i.e. from MSB to LSB.

Example: To convert the decimal number 87\(_{10}\) to binary.
   - So 87 decimal is written as 1010111 in binary.
   - It can be written as 87\(_{10}\) = 1010111\(_2\)
2. Steps to convert decimal fraction number to binary number:

- **Step 1**: Multiply the given decimal fraction number by 2.
- **Step 2**: Note the carry and the product.
- **Step 3**: Repeat the Step 1 and Step 2 until the decimal number cannot be divided further.
- **Step 4**: The first carry will be the MSB and the last carry is the LSB. The equivalent binary fraction number is written from MSB to LSB.

**Example 1**: To convert the decimal number 0.3125\(_{10}\) to binary.

<table>
<thead>
<tr>
<th>Multiply by 2</th>
<th>Carry</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3125 x 2</td>
<td>0 (MSB)</td>
<td>0.625</td>
</tr>
<tr>
<td>0.625 x 2</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>0.25 x 2</td>
<td>0</td>
<td>0.50</td>
</tr>
<tr>
<td>0.50 x 2</td>
<td>1 (LSB)</td>
<td>0.00</td>
</tr>
</tbody>
</table>

- Therefore, 0.3125\(_{10}\) = 0.0101\(_{2}\).  

**OR**

```
0.3125 x 2 = 0.6250
0.6250 x 2 = 1.2500
0.2500 x 2 = 0.5000
0.5000 x 2 = 1.0000
Hence 0.3125_{10} = 0.0101_{2}
Stop this process since Fraction part is zero
```
Example 2: To convert the decimal number $152.671875_{(10)}$ to binary.

![Binary Conversion Diagram]

3. Steps to convert binary number to decimal number

- **Step 1:** Start at the rightmost bit.
- **Step 2:** Take that bit and multiply by $2^n$, when $n$ is the current position beginning at 0 and increasing by 1 each time. This represents a power of two.
- **Step 3:** Then, add all the products.
- **Step 4:** After addition, the resultant is equal to the decimal value of the binary number.

Example 1: To convert the binary number $1010111_{(2)}$ to decimal.

\[
1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 64 + 16 + 4 + 2 + 1 = 87
\]

Therefore, $1010111_{(2)} = 87_{(10)}$
Example 2: To convert the binary number 11011.101_2 to decimal.

\[1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}\]

\[= 1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 + 1 \times 0.5 + 0 \times 0.25 + 1 \times 0.125\]

\[= 16 + 8 + 2 + 1 + 0.5 + 0.125\]

\[= 27.625_{(10)}\]

OR

<table>
<thead>
<tr>
<th>Weights</th>
<th>$2^4$</th>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
<th>$2^{-1}$</th>
<th>$2^{-2}$</th>
<th>$2^{-3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Values</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0.5</td>
<td>0.25</td>
<td>0.125</td>
</tr>
</tbody>
</table>

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

Conversion from Decimal to Octal

1. Steps to convert decimal number to octal number

   - Step 1: Divide the given decimal number by 8.
   - Step 2: Take the remainder and record it on the side.
   - Step 3: Repeat the Step 1 and Step 2 until the decimal number cannot be divided further.
   - Step 4: The first remainder will be the LSB and the last remainder is the MSB. The equivalent octal number is then written from left to right i.e. from MSB to LSB.

Example 1: To convert the decimal number 3034_{(10)} to octal number.

- So 3034 decimal is written as 5732 in octal.
- It can be written as $3034_{(10)} = 5732_{(8)}$

- Note: If the number is less than 8 the octal number is same as decimal number.

Example 2: To convert the decimal number 0.3125_{(10)} to octal number.

\[0.3125 \times 8 = 2.5000\]

\[0.5000 \times 8 = 4.0000\]

- Therefore, $0.3125_{(10)} = 0.24_{(8)}$
2. Steps to convert octal number to decimal number

- **Step 1:** Start at the rightmost bit.
- **Step 2:** Take that bit and multiply by $8^n$, when $n$ is the current position beginning at 0 and increasing by 1 each time. This represents the power of 8.
- **Step 3:** Then, add all the products.
- **Step 4:** After addition, the resultant is equal to the decimal value of the octal number.

**Example 1:** To convert the octal or base-8 number $5732_8$ to decimal

\[
\begin{align*}
5 & \times 8^3 + 7 & \times 8^2 + 3 & \times 8^1 + 2 & \times 8^0 \\
& = 5 \times 512 + 7 \times 64 + 3 \times 8 + 2 \times 1 \\
& = 2560 + 448 + 24 + 2 \\
& = 3034
\end{align*}
\]

Therefore, $5732_8 = 3034_{10}$

**Example 2:** To convert the octal number $234.56_8$ to decimal number.

\[
\begin{align*}
2 & \times 8^2 + 3 & \times 8^1 + 4 & \times 8^0 + 5 & \times 8^{-1} + 6 & \times 8^{-2} \\
& = 2 \times 64 + 3 \times 8 + 4 \times 1 + 5 \times 0.125 + 6 \times 0.015625 \\
& = 128 + 24 + 4 + 0.625 + 0.09375 \\
& = 156.71875_{10}
\end{align*}
\]

OR

<table>
<thead>
<tr>
<th>Weights</th>
<th>$8^2$</th>
<th>$8^1$</th>
<th>$8^0$</th>
<th>$8^{-1}$</th>
<th>$8^{-2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Values</td>
<td>64</td>
<td>8</td>
<td>1</td>
<td>0.125</td>
<td>0.015625</td>
</tr>
</tbody>
</table>

Therefore, $234.56_8 = 156.71875_{10}$

**Conversion from Decimal to Hexadecimal**

1. Steps to convert decimal number to hexadecimal number

- **Step 1:** Divide the decimal number by 16.
- **Step 2:** Take the remainder and record it on the side.
- **Step 3:** Repeat the Step 1 and Step 2 until the decimal number cannot be divided further.
Step 4: The first remainder will be the LSB and the last remainder is the MSB. The equivalent hexadecimal number is then written from left to right i.e. from MSB to LSB.

Example: To convert the decimal number 16242\(_{(10)}\) to hexadecimal

- So 16242 decimal is written as 3F72 in hexadecimal.
- It can be written as 16242\(_{(10)}\) = 3F72\(_{(16)}\)
- Note: If the number is less than 16 the hexadecimal number is same as decimal number.

2. Steps to convert hexadecimal number to decimal number

- Step 1: Start at the rightmost bit.
- Step 2: Take that bit and multiply by 16\(^n\), where \(n\) is the current position beginning at 0 and increasing by 1 each time. This represents a power of 16.
- Step 3: Then, add all the products.
- Step 4: After addition, the resultant is equal to the decimal value of the hexadecimal number.

Example 1: To convert the Hexadecimal or base-16 number 3F72 to a decimal number.

\[
\begin{align*}
3 \times 16^3 + F \times 16^2 + 7 \times 16^1 + 2 \times 16^0 \\
3 \times 4096 + 15 \times 256 + 7 \times 16 + 2 \\
12288 + 3840 + 112 + 2 \\
16242
\end{align*}
\]

Therefore, 3F72\(_{(16)}\) = 16242\(_{(10)}\)

Example 2: To convert the hexadecimal number 5AF.D\(_{(16)}\) to decimal number.

\[
\begin{align*}
5 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 + 13 \times 16^{-1} \\
5 \times 256 + 10 \times 16 + 15 \times 1 + 13 \times 0.0625 \\
1280 + 160 + 15 + 0.8125 \\
1455.8125
\end{align*}
\]

OR

<table>
<thead>
<tr>
<th>Weights</th>
<th>16(^2)</th>
<th>16(^1)</th>
<th>16(^0)</th>
<th>16(^{-1})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td>5</td>
<td>A</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>Values</td>
<td>256</td>
<td>16</td>
<td>1</td>
<td>0.0625</td>
</tr>
</tbody>
</table>

Therefore, 5AF.D\(_{(16)}\) = 1455.8125\(_{(10)}\)
Conversion from Binary to Octal

Steps to convert Binary to octal
- Take a binary number in groups of 3 and use the appropriate octal digit in its place.
- Begin at the rightmost 3 bits. If we are not able to form a group of three, insert 0s to the left until we get all groups of 3 bits each.
- Write the octal equivalent of each group. Repeat the steps until all groups have been converted.

Example 1: Consider the binary number 1010111₂
```
1 010 111
1 2 7
```
Therefore, 1010111₂ = 127₈

Example 2: Consider the binary number 0.110111₂
```
0 110 111
0 6 7
```
Therefore, 0.110111₂ = 0.67₈

Example 3: Consider the binary number 1101.10111₂
```
001 101 101 110
1 5 5 6
```
Therefore, 1101.10111₂ = 15.56₈

Note: To make group of 3 bits, for whole numbers, it may be necessary to add a 0’s to the left of MSB and when representing fractions, it may be necessary to add a 0’s to right of LSB.

Conversion from Octal to Binary

Steps to convert octal to binary
- Step 1: Take the each digit from octal number
- Step 2: Convert each digit to 3-bit binary number. (Each octal digit is represented by a three-bit binary number as shown in Numbering System Table)
### Chapter 3 - Data Representation

<table>
<thead>
<tr>
<th>Octal digit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Equivalent</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
</tr>
</tbody>
</table>

**Example 1:** Consider the octal number $456_{(8)}$ into binary

- $4 \rightarrow 100$
- $5 \rightarrow 101$
- $6 \rightarrow 110$

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

**Example 2:** Consider the octal number $73.16_{(8)}$ into binary

- $7 \rightarrow 100$
- $3 \rightarrow 101$
- $1 \rightarrow 001$
- $6 \rightarrow 110$

Therefore, $73.16_{(8)} = 100101.001110_{(2)}$

**Conversion from Binary to Hexadecimal**

**Steps to convert Binary to Hexadecimal**

- Take a binary number in groups of 4 and use the appropriate hexadecimal digit in its place.
- Begin at the rightmost 4 bits. If we are not able to form a group of four, insert 0s to the left until we get all groups of 4 bits each.
- Write the hexadecimal equivalent of each group. Repeat the steps until all groups have been converted.

**Example 1:** Consider the binary number $1011001_{(2)}$

- $0101$
- $1001$

5 9  Therefore, $1011001_{(2)} = 59_{(16)}$

**Example 2:** Consider the binary number $0.11010111_{(2)}$

- $0$
- $1101$
- $0111$

0 D 7  Therefore, $0.110111_{(2)} = 0.D7_{(16)}$
### Conversion from Hexadecimal to Binary

**Steps to convert hexadecimal to binary**

- **Step 1:** Take the each digit from hexadecimal number
- **Step 2:** Convert each digit to 4-bit binary number. (Each hexadecimal digit is represented by a four-bit binary number as shown in Numbering System Table)

**Example:** Consider the hexadecimal number CEBA\(_{16}\)

\[
\begin{array}{c|c|c}
C & \rightarrow & 12 \rightarrow 1100 \\
E & \rightarrow & 14 \rightarrow 1110 \\
B & \rightarrow & 11 \rightarrow 1011 \\
A & \rightarrow & 10 \rightarrow 1010
\end{array}
\]

Therefore, CEBA\(_{16}\) = 1100 1110 1011 1010\(_{2}\)

### Conversion from Octal to Hexadecimal

**Steps to convert Octal to Hexadecimal**

Using Binary system, we can easily convert octal numbers to hexadecimal numbers and vice-versa

- **Step 1:** write the binary equivalent of each octal digit.
- **Step 2:** Regroup them into 4 bits from the right side with zeros added, if necessary.
- **Step 3:** Convert each group into its equivalent hexadecimal digit.

**Example:** Consider the octal number 274\(_{8}\)

\[
\begin{array}{c|c|c|c}
2 & \rightarrow & 010 \\
7 & \rightarrow & 111 \\
4 & \rightarrow & 100
\end{array}
\]

Therefore, 274\(_8\) = 010 111 100\(_2\)

Group the bits into group of 4 bits as 0 1011 1100

\[
\begin{array}{c|c|c|c}
| & 1011 & | & 1100 \\
| & 0 & | & B \\
| & 0 & | & C
\end{array}
\]

Therefore, 274\(_8\) = BC\(_8\)
Conversion from Hexadecimal to Octal

Steps to convert Hexadecimal to Octal

- **Step 1**: write the binary equivalent of each hexadecimal digit.
- **Step 2**: Regroup them into 3 bits from the right side with zeros added, if necessary.
- **Step 3**: Convert each group into its equivalent octal digit.

**Example**: Consider the hexadecimal number FADE\(_{(16)}\)

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>1111</td>
</tr>
<tr>
<td>A</td>
<td>1010</td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
</tr>
</tbody>
</table>

Therefore, FADE\(_{(16)}\) = 1111 1010 1101 1110\(_{(2)}\)

Group the bits into group of 3 bits from LSB as 001 111 101 011 011 110

Therefore, FADE\(_{(16)}\) = 175336\(_{(8)}\)

**Binary Arithmetic**

- It involves addition, subtraction, multiplication and division operations.
- Binary arithmetic is much simpler to learn because system deals with only two digit 0’s and 1’s.
- When binary arithmetic operations are performed on binary numbers, the results are also 0’s and 1’s.
Binary Addition

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

<table>
<thead>
<tr>
<th>Addend1</th>
<th>Addend2</th>
<th>Sum</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- It adds only two bits and gives sum and carry. If a carry is generated, it should be carried over to the addition of next two bits.
- **Example 1**: Add the following number
  a) 4 and 3
  b) 9 and 5

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>101</td>
</tr>
<tr>
<td>+ 3</td>
<td>+ 011</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>+ 5</td>
<td>+ 101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
</tr>
</tbody>
</table>

- **Example 2**: Add 75 and 18 in binary number.

  Convert this decimal number into binary i.e.

  \[
  75 = 64 + 8 + 2 + 1 = 1001011 \\
  18 = 16 + 2 = 10010
  \]

<table>
<thead>
<tr>
<th>Carry</th>
<th>Append 1</th>
<th>Append 2</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1001011</td>
<td>10010</td>
<td>1011101</td>
</tr>
</tbody>
</table>

- **Example 3**: Add binary number 1011.011 and 1101.111

<table>
<thead>
<tr>
<th>Carry</th>
<th>Append 1</th>
<th>Append 2</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1011.011</td>
<td>1101.111</td>
<td>11001.010</td>
</tr>
</tbody>
</table>

**Exercise:**
- 1) Add 10101 and 11011
- 2) Add 1011101 and 1100111
### Binary Subtraction

- This operation is same as the one performed in the decimal system.
- This operation consists of two steps:
  - Determine whether it is necessary for us to borrow. If the subtrahend (the lower digit) is larger than the minuend (the upper digit), it is necessary to borrow from the column to the left. In binary two is borrowed.
  - Subtract the lower value from the upper value
- The basic rules of binary subtraction are:

<table>
<thead>
<tr>
<th>Minuend</th>
<th>Subtrahend</th>
<th>Difference</th>
<th>Borrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- When we subtract 1 from 0, it is necessary to borrow 1 from the next left column i.e. from the next higher order position.

#### Example 1: Subtract the following number

<table>
<thead>
<tr>
<th>a) 10 from 14</th>
<th>b) 9 from 29</th>
<th>c) 3 from 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>Binary</td>
<td>Decimal</td>
</tr>
<tr>
<td>14</td>
<td>1 1 1 0</td>
<td>29</td>
</tr>
<tr>
<td>- 10</td>
<td>- 1 0 1 0</td>
<td>- 0 9</td>
</tr>
<tr>
<td>4</td>
<td>0 1 0 0</td>
<td>20</td>
</tr>
</tbody>
</table>

#### Example 2: Add 75 and -18 in binary number.

\[75 = 64 + 8 + 2 + 1 = 1001011\]
\[18 = 16 + 2 = 10010\]

- Borrow \(\rightarrow\) 
- Minuend \(\rightarrow\) 1 0 0 1 0 1 1 \(\Rightarrow\) 75
- Subtrahend \(\rightarrow\) 1 0 0 1 0 \(\Rightarrow\) -18
- Difference \(\rightarrow\) 0 1 1 1 0 0 1 \(\Rightarrow\) 57

#### Example 3: Subtract binary number 0110100 from 1011000

**Binary**

\[
\begin{align*}
\text{Binary} & \quad \text{Decimal} \\
0 & \rightarrow 5 \quad 1010_2 \\
1 & \rightarrow 8 \quad 1000_2 \\
1 & \rightarrow 8 \quad 1000_2 \\
0 & \rightarrow 4 \quad 100_2 \\
1 & \rightarrow 4 \quad 100_2 \\
1 & \rightarrow 4 \quad 100_2 \\
0 & \rightarrow 2 \quad 10_2 \\
0 & \rightarrow 2 \quad 10_2 \\
\end{align*}
\]

**Exercise:**

1) Subtract 01110 from 10101 
2) Subtract 25 from 35
Representation of signed Integers

- The digital computer handles both positive and negative integers.
- It means, it is required for representing the sign of the number (- or +), but cannot use the sign (-) to denote the negative number or sign (+) to denote the positive number.
- So, this is done by adding the leftmost bit to the number called **sign bit**.
- A positive number has a sign bit 0, while the negative has a sign bit 1.
- It is also called as **fixed point representation**.
- A negative signed integer can be represented in one of the following:
  1) Sign and magnitude method
  2) One’s complement method
  3) Two’s complement method

Sign and magnitude method

- An integer containing a sign bit followed by magnitude bits are called **sign-magnitude integer**.
- In this method, first bit (MSB) is considered as a sign bit and the remaining bits stand for magnitude.
- Here positive number starts with 0 and negative number starts with 1.
- **Example**: Consider a decimal number $25_{(10)}$

<table>
<thead>
<tr>
<th>2</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>$\rightarrow 1$</td>
</tr>
</tbody>
</table>

- So the binary number is $11001_{(2)}$. If we take the size of the word is 1 byte (8 bits), then the number 25 will be represented as

```
0 0 0 1 1 0 0 1
```

- Suppose, if the number is -25, and then it will be represented as:

```
1 0 0 1 1 0 0 1
```

**Sign Bit**  Magnitude = 25
1’s Complement representation

- This is the simplest method of representing negative binary number.
- The 1’s complement of a binary number is obtained by changing each 0 to 1 and each 1 to 0.
- In other words, change each bit in the number to its complement.

**Example 1:** Find the 1’s complement of 101000.

Original binary number \(\rightarrow\) 1 0 1 0 0 0
Find 1’s Complement \(\rightarrow\) 0 1 0 1 1 1
Thus 1’s complement of 101000 is 010111.

**Example 2:** Find the 1’s complement of 1010111.

Original binary number \(\rightarrow\) 1 0 1 0 1 1 1
Find 1’s Complement \(\rightarrow\) 0 1 0 1 0 0 0
Thus 1’s complement of 101000 is 010111.

2’s Complement representation

- The 2’s complement of a binary number is obtained by taking 1’s complement of the number and adding 1 to the Least Significant Bit (LSB) position.
- The general procedure to find 2’s complement is given by:

\[
2's\ Complement = 1's\ Complement + 1
\]

**Example 1:** Find the 2’s complement of 101000.

Original binary number \(\rightarrow\) 1 0 1 0 0 0
Find 1’s Complement \(\rightarrow\) 0 1 0 1 1 1
Add 1 to LSB \(+\) 1
Hence 2’s Complement of 101000 is \(\rightarrow\) 0 1 1 0 0 0

**Example 2:** Find the 1’s and 2’s complement of 101101.

Original binary number \(\rightarrow\) 1 0 1 1 1 0 1
1’s Complement \(\rightarrow\) 0 1 0 0 0 1 0
\(+\) 1
Hence 2’s Complement is \(\rightarrow\) 0 1 0 0 0 1 1
Note:

Negation: It is the operation of converting a positive number to its negative equivalent or a negative number to its positive equivalent. Negation is performed by performing 2’s complement system.

- **Example 1**: Consider the number +12. Its binary representation is \(01100_2\). Find the 2’s complement of +12.

  | Original binary number | \(\rightarrow\) | 0 1 1 0 0 |
  | Find 1’s Complement     | \(\rightarrow\) | 1 0 0 1 1 |
  | Add 1 to LSB            | +           | 1          |
  | 2’s Complement           | \(\rightarrow\) | 1 0 1 0 0 |

  Clearly, this is a negative number since the sign bit is 1. Actually, \(10100\) represent \(-12_{(10)}\), which is the negative equivalent of the number \(12_{(10)}\).

- **Example 2**: Consider the number \(-12\). Its binary representation is \(10100_2\). Find the 2’s complement of \(-12\).

  | Original binary number | \(\rightarrow\) | 1 0 1 0 0 |
  | Find 1’s Complement     | \(\rightarrow\) | 0 1 0 1 1 |
  | Add 1 to LSB            | +           | 1          |
  | 2’s Complement           | \(\rightarrow\) | 0 1 1 0 0 |

  Clearly, this is a positive number since the sign bit is 0. Actually, \(01100\) represent \(12_{(10)}\), which is the negative equivalent of the number \(-12_{(10)}\).

**Subtraction of Binary Number using Complement**

- Most of the computers perform subtraction using complemented number. This is less expensive because the same addition circuit is used for subtraction with slight changes in the circuit.
- In the binary number system we can perform subtraction operation using two methods of complements:
  - Subtraction using 1’s Complement
  - Subtraction using 2’s Complement
Subtraction using 1’s Complement

- **Case 1**: Subtracting a smaller number from a larger number (Minuend is greater than Subtrahend)
  - **Step 1**: Find the 1’s complement of the subtrahend.
  - **Step 2**: Add this to the minuend.
  - **Step 3**: Carry is generated, this carry is called as the end around carry
  - **Step 4**: Add the end around carry back to the LSB to get the final difference.

- **Example 1**: Subtract 15 from 23 using 1’s complement.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minuend</td>
<td>23</td>
</tr>
<tr>
<td>Subtrahend</td>
<td>-15</td>
</tr>
</tbody>
</table>

1’s complement of subtrahend is 10000

<table>
<thead>
<tr>
<th>Minuend</th>
<th>1 0 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1’s Complement of subtrahend</td>
<td>+ 1 0 0 0 0</td>
</tr>
<tr>
<td>End around carry</td>
<td>1 1 0 1 1</td>
</tr>
<tr>
<td>Add end around carry</td>
<td>1</td>
</tr>
<tr>
<td>Difference is</td>
<td>1 0 0 0</td>
</tr>
</tbody>
</table>

- **Case 2**: Subtracting a larger number from a smaller number (Minuend is less than Subtrahend)
  - **Step 1**: Find the 1’s complement of the subtrahend.
  - **Step 2**: Add this to the minuend.
  - **Step 3**: There will be no carry, Re complement the answer to get the difference

- **Example 1**: Subtract 52 from 25 using 1’s complement.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minuend</td>
<td>25</td>
</tr>
<tr>
<td>Subtrahend</td>
<td>-52</td>
</tr>
</tbody>
</table>

1’s complement of subtrahend is 001011

<table>
<thead>
<tr>
<th>Minuend</th>
<th>0 1 1 0 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1’s Complement of subtrahend</td>
<td>+ 0 0 1 0 1 1</td>
</tr>
<tr>
<td>Since there is no carry</td>
<td>1 0 0 1 0 0</td>
</tr>
<tr>
<td>take 1’s complement and attach a negative sign</td>
<td>0 1 1 0 1 1</td>
</tr>
</tbody>
</table>

Hence, the result = - 011011 i.e. - 27
**Subtraction using 2’s Complement**

- **Case 1:** Subtracting a smaller number from a larger number (Minuend is greater than Subtrahend)
  - **Step 1:** Find the 2’s complement of the subtrahend.
  - **Step 2:** Add this to the minuend.
  - **Step 3:** Carry is generated, Discard the carry and the remaining bits give the difference.

- **Example 1:** Subtract 09 from 17 using 2’s complement.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minuend</td>
<td>17</td>
</tr>
<tr>
<td>Subtrahend</td>
<td>-09</td>
</tr>
</tbody>
</table>

1’s complement of subtrahend (9) is \(1 0 1 1 0\)  
Add 1 to LSB  
2’s complement of 9 is  
Minuend  
2’s Complement of subtrahend  
End around carry  
Discard the carry  

Difference is  
Hence, the result = 1000 i.e. 8

- **Case 2:** Subtracting a larger number from a smaller number (Minuend is less than Subtrahend)
  - **Step 1:** Find the 2’s complement of the subtrahend.
  - **Step 2:** Add this to the minuend.
  - **Step 3:** There will be no carry, hence take the 2’s complement of the answer and place a negative sign in front.

- **Example 1:** Subtract 47 from 26 using 2’s complement.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minuend</td>
<td>26</td>
</tr>
<tr>
<td>Subtrahend</td>
<td>-47</td>
</tr>
</tbody>
</table>

1’s complement of subtrahend (47) is \(0 1 0 0 0 0\)  
Add 1 to LSB  
2’s complement of 9 is  

Chapter 3 - Data Representation

Minuend → 0 1 1 0 1 0

2’s Complement of subtrahend → +0 1 0 0 0 1
                                  1 0 1 0 1 1

STEP 2

1’s Complement of answer → 0 1 0 1 0 0

2’s Complement of answer → 0 1 0 1 0 0
+ 1
0 1 0 1 0 1

Hence, the result = 10101 i.e. 21

➤ Computer Codes

- Computer code helps us to represent characters in a coded form in the memory of the computer.
- These codes represent specific formats which are used to record data.
- Some of the commonly used computer codes are:
  - Binary Coded Decimal (BCD)
  - Extended Binary Coded Decimal Interchange Code (EBCDIC)
  - American Standard Code for Information Interchange (ASCII)
  - Excess-3 Code.

➤ BCD code (or Weighted BCD Code or 8421 Code)

- BCD stands for Binary Coded Decimal.
- It is one of the early computer codes.
- In this coding system, the bits are given from left to right, the weights 8,4,2,1 respectively.
- The BCD equivalent of each decimal digit is shown in table.

<table>
<thead>
<tr>
<th>Decimal Digit</th>
<th>BCD Equivalent</th>
<th>Decimal Digit</th>
<th>BCD Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>9</td>
<td>1001</td>
</tr>
</tbody>
</table>

- Example: Convert the decimal number 537 into BCD.

0101 0110 0111

Hence, the BCD of 537 is 010101100111
• In 4-bit BCD only $2^4 = 16$ configurations are possible which is insufficient to represent the various characters.
• Hence 6-bit BCD code was developed by adding two zone positions with which it is possible to represent $2^6 = 64$ characters.

➤ **Excess-3 BCD code (or XS-3 Code)**

• The Excess-3 BCD code is a non-weighted code used to express decimal number.
• The name Excess-3 code derived from the fact that each binary code is the corresponding BCD code plus $0011_2$ (i.e. Decimal 3).
• This code is used in some old computers.
• The following table gives the Excess-3 code equivalent of decimal (0-9).

<table>
<thead>
<tr>
<th>Decimal Number</th>
<th>BCD Number</th>
<th>Adding 3 to BCD</th>
<th>Excess-3 Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>11</td>
<td>0011</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>11</td>
<td>0100</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>11</td>
<td>0101</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>11</td>
<td>0110</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>11</td>
<td>0111</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>11</td>
<td>1000</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>11</td>
<td>1001</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>11</td>
<td>1010</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>11</td>
<td>1100</td>
</tr>
</tbody>
</table>

• **Example**: Find Excess-3(XS3) representation of decimal number 537.

```
5 3 7  \rightarrow  Decimal Digit
0101 0110 0111  \rightarrow  8421 BCD Code
0011 0011 0011  \rightarrow  Add 3
1000 1010 0110  \rightarrow  Excess-3 Code
```

➤ **EBCDIC**

• It stand for Extended Binary Coded Decimal Interchange Code.
• This was developed by IBM.
• It uses an 8-bit code and hence possible to represent 256 different characters or bit combinations.
• EBCDIC is used on most computers and computer equipment today.
• It is a coding method generally used by larger computers (mainframes) to present letters, numbers or other symbols in a binary language the computer can understand.
• EBCDIC is an 8-bit code; therefore, it is divided into two 4-bit groups, where each 4-bit can be represented as 1 hexadecimal digit.

➢ ASCII

• It stands for the American Standard Code for Information Interchange.
• It is a 7-bit code, which is possible to represent $2^7 = 128$ characters.
• It is used in most microcomputers and minicomputers and in mainframes.
• The ASCII code (Pronounced ask-ee) is of two types – ASCII-7 and ASCII-8.
• ASCII-7 is 7-bit code for representing English characters as numbers, with each letter assigned a number from 0 to 127.
• Example: The ASCII code for uppercase M is 77.