Number System and Boolean Algebra

Number System

There are many systems in which numbers can be expressed. The decimal number system is familiar to us. In this system the base is 10 and the digits are 0,1,2,3,4,5,6,7,8,9.

The following are other number systems which are more popular in use:

<table>
<thead>
<tr>
<th></th>
<th>Binary system</th>
<th>radix</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td>2^1</td>
</tr>
<tr>
<td>2</td>
<td>Octal system</td>
<td>8</td>
<td>2^3</td>
</tr>
<tr>
<td>3</td>
<td>Hexa-decimal system</td>
<td>16</td>
<td>2^4</td>
</tr>
</tbody>
</table>

Most of the digital computers use binary, octal and Hexa-decimal systems.

BINARY ARITHMETIC

In practice, we use both positive and negative numbers as operands. In the binary system, we represent the sign of number using an extra bit at the extreme left of the number. By convention the symbol ‘0’ is used to represent left the ‘+’ sign and ‘1’ to represent the ‘-‘ sign. For instance, +6 is represent by 0,110 and -7 is represented by 1,111. This method is known as sign magnitude representation.

When we have to subtract a number B from a number A, we look at the magnitude as well as the sign of these numbers. When A and B have opposite signs, in effect we add their magnitudes and determine the sign of the result. On the other hand, if A and B have the same sign, we always subtract the smaller magnitude from the large and once again, decide the sign independently. similarly, when we have to add, the procedures for which are different.

It would be more convenient if we could evolve another convention for representing positive and negative numbers which would allow us to use one basic procedure for both addition and subtraction. So we could use a single electronic circuit to implement both addition and subtraction, a convention for representing negative numbers which allows this ‘complement representation’ of numbers.

OCTAL SYSTEMS

Binary system is very convenient system for the present computers as they deal in terms of 0’s and 1’s. The representation of binary numbers to external world can be made compact with the help of octal and hexadecimal systems. A group of 3 bits will be able to represent $2^3 = 8$ different possibilities. We can use eight symbols say 0,1,2,3,4,5,6 & 7 to represent these sequences of 3 bits. Such a system of eight symbols is called octal system. Similarly, a group of 4 bits will be representing $2^4 = 16$ combinations. To represent 16 sequences of 4 bit each we can use 16 symbols say 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E & F. decimal system equivalent of A is 10, B is 11, c is 12, D is 13, E is 14 and F is 15. A number system with 16 symbols is known as hexadecimal are also positional number systems.
Octal to Decimal. To convert an octal number to a decimal number, we use the polynomial but this time, the radix will be 8. For example, 234 in octal is \(2 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 = 128 + 24 + 4 = 156\) in decimal. Octal number 15.24 is \(1 \times 8^1 + 5 \times 8^0 + 2 \times 8^{-1} + 4 \times 8^{-2} = 8 + 5 + 2/8 + 4/64 = 13.3125\) in decimal.

Binary to Octal. Since 3 bits are taken at a time, one octal digit, a given binary number can be directly converted in octal by making groups of three size bits each starting from octal point. For instance, 101011 in binary is 53 in octal, 1110001 in binary is 161 in octal (add two leading zeros to complete the last group of 3 bits), (1110100.0100111 in binary is 164.234 in octal.

Octal to Binary. An octal number can be converted into its binary equivalent by replacing each octal digit with its three-bit binary equivalent. We take the three-bit equivalent because the base of the octal number system is 8 and it is the third power of the base of the binary number system, i.e. 2. All we have then to remember is the three-bit binary equivalents of the basic digits of the octal number system. (i) Create a look up table with 8 entries and their corresponding 3 bit binary code (ii) separate each octal digit from the given number and using look up table find binary value and print it (iii) repeat step (ii) until all the digits are converted into binary.

HEXADECIMAL NUMBERS.

In principal, Hexadecimal numbers work in the exact same way as decimal numbers do. Nearly all the same rules apply. The main difference is that there are more symbols. In the case of Hexadecimal, Latin for 16, there are 16. The first ten are old faithful 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The six new ones are A, B, C, D, E, F. Just like we made a little counting table for Decimal, we can make one for Hexadecimal as well:

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexagonal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
</tbody>
</table>

Apart from the difference in the symbols and the different number of them, the Math rules in Hexadecimal, which we will from now on simply call Hex, are the same as the rules in Decimal. In Hexadecimal math, \(4 + 1 = 5\) and \(6 + 3 = 9\) and \(9 + 3 = C\). As in Decimal math, you add an MSD to the left in Hex when you run out of Symbols, example in Hex: \(C + 5 = 11\) and Hex \(11 + 3C = 4D\). Although this sounds simple and logical, our brain has trouble with it since we are trained in decimal!

Following table shows equivalences between hexagonal, binary and decimal digits.
Hexadecimal Arithmetic. Covert hexadecimal number to binary number and then apply arithmetic operations.

**RADIX**

The total number of digits applicable to any system is called its radix. It is always one more than the highest digit of the system. The digits of various types of number system are as under:

1. Binary system   0,1
2. Octal system   0, 1, 2, 3, 4, 5, 6, 7
3. Decimal system  0, 1, 2, 3, 4, 5, 6, 7, 8, 9
4. Hexa-Decimal system  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

**CONVERSION OF NUMBERS**

**Conversion of a number from any system to decimal system**

Let us study the formation of the number 1970 in decimal system.

\[(1970)_{10} = 1 \times 10^3 + 9 \times 10^2 + 7 \times 10^1 + 0 \times 10^0 \]
\[= 1000 + 900 + 70 + 0 = 1970 \]

(i) Convert binary number \((111)_2\) into decimal system.

\[(111)_2 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 4 + 2 + 1 = (7)_{10}\]

(ii) Convert octal number \((1654)_8\) into decimal system

\[(1654)_8 = 1 \times 8^3 + 6 \times 8^2 + 5 \times 8^1 + 4 \times 8^0 = 512 + 384 + 40 + 4 = (940)_{10}\]

(iii) Convert Hexa-decimal number \((123)_{16}\) to decimal system

\[(123)_{16} = 1 \times 16^2 + 2 \times 16^1 + 3 \times 16^0 = 256 + 32 + 3 = (291)_{10}\]
Conversion of a number from decimal system to other systems

The number of any system, may be converted to decimal system by the following two methods.

1. Reminder method
2. Power method

These methods will be discussed in the classified examples.

TWO’S COMPLEMENT REPRESENTATION OF NUMBERS

The 2’s complement of a binary number x, which has n bits, is given by \(2^n - x\). Consider the number +3 whose binary representation is 011. It has three bits. Hence the 2’s complement of +3 is \(2^3 - 3\), which is 5. The binary representation of 5 is 101. Thus, 101 is the 2’s complement of 011.

In a computer, all numbers are represented in a uniform fashion using a fixed number of bits. Thus, for an n-bit machine, the range of numbers it can handle is 0 to \(2^n - 1\). For simplicity, consider a 4 bit machine. Sixteen numbers (0 to 15) can normally be represented using these four bits. Now we devise a new scheme of representing negative numbers as follows. We use the first seven combinations of bits for representing positive numbers one to seven. We reserve seven of the remaining combinations for representing negative number -1 to -7. Thus, we have divided the total range into two parts, 0 and 8 being common to the two halves. Now we restrict ourselves to the use of numbers having a maximum magnitude of seven. In this way we can represent both positive and negative numbers as indicated in Fig. below. With reference to the figure we see that the code 1111, which normally represents 15, is assigned to -1. Similarly the binary equivalent of 14 is assigned to -2. Since \(15 = 2^4 - 1\), \(14 = 2^4 - 2\), etc., this is called the 2’s complement representation.

There is a simple procedure to obtain the 2’s complement of a binary number. We first complement each bit of the number (i.e. replace ‘1’ by ‘0’ and ‘0’ by ‘1’). Now we add a ‘1’ to the number. For example, consider the number 5 whose binary representation is 1011. Bit complementation yields 1010. Now adding a ‘1’ to this number gives 1011 which is the 2’s complement representation for -5.

Two’s complement representation of numbers
Yet another method of obtaining the 2’s complement of a binary number is to scan the number from right to left and complement all bits appearing after the first appearance of a ‘1’. For example the 2’s complement of 0010 is 1110.

**Example**

*Convert $(940)_{10}$ to binary system.*

**Solution**

**Reminder Method**

The radix of binary system is 2.

<table>
<thead>
<tr>
<th>2</th>
<th>940</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>470</td>
</tr>
<tr>
<td>2</td>
<td>235</td>
</tr>
<tr>
<td>2</td>
<td>117</td>
</tr>
<tr>
<td>2</td>
<td>58</td>
</tr>
<tr>
<td>2</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

i.e. $(940)_{10} = (1110101100)_{2}$

**Example**

*Convert $(2040)_{10}$ to octal system.*

**Solution**

**Reminder Method**

The radix of octal system is 8.

<table>
<thead>
<tr>
<th>8</th>
<th>2040</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>255</td>
</tr>
<tr>
<td>8</td>
<td>31</td>
</tr>
</tbody>
</table>

i.e. $(2040)_{10} = (3770)_{8}$.
Example

Convert $(2040)_{10}$ to hexa-decimal system.

Solution

Reminder Method

The radix of hexa-decimal system is 16.

\[
\begin{array}{c|c|c}
16 & 2040 \\
16 & 127 & 8 \\
& 7 & 15 \\
\end{array}
\]

i.e. $(2040)_{16} = (7E8)_{8}$.

Problem (AMIE Winter 2001)

Convert following decimal number to hexadecimal number:

$395$

Answer: $(18B)_{16}$

Example

Obtain the octal equivalent of $(3964)_{10}$.

Solution

Reminder Method

Radix of the octal system is 8.

\[
\begin{array}{c|c|c}
8 & 3964 \\
8 & 495 & 4 \\
8 & 61 & 7 \\
& 7 & 5 \\
\end{array}
\]

i.e. $(3964)_{10} = (7574)_{8}$.

Example

Convert $(300)_{10}$ to the binary system.

Solution

Power Method

Here radix of binary system is 2.
We note that 256 is the highest number which is less than 300, the given number

\[
\begin{array}{c}
300 \\
- 256 \\
44 \\
- 32 \quad 2^5 = 1 x 2^5 \\
12 \\
- 8 \quad 2^3 = 1 x 2^3 \\
4 \\
- 4 \quad 2^2 = 1 x 2^2 \\
0
\end{array}
\]

As powers of \(2^7, 2^6, 2^4, 2^1, 2^0\) are not represented, these values may be placed in their respective places after multiplying with zero i.e.

\[
1 \times 2^8, \quad 0 \times 2^7, \quad 0 \times 2^6, \quad 1 \times 2^5, \quad 0 \times 2^4, \quad 1 \times 2^3, \quad 1 \times 2^2, \quad 0 \times 2^1, \quad 0 \times 2^0
\]

i.e. \((300)_{10} = (100101100)\).

Example

Convert \((300)_{10}\) to octal system.

Solution

Power Method

The radix is 8

\[
\begin{array}{cccccccc}
8^5 & 8^4 & 8^3 & 8^2 & 8^1 & 8^0 \\
32768 & 4096 & 512 & 64 & 8 & 1
\end{array}
\]

We note that 256 is the highest number i.e., \(4 \times 8^2\).

\[
\begin{array}{c}
300 \\
- 256 \\
44 \\
- 40 \quad 5 \times 8^1 \\
4 \\
- 4 \quad 4 \times 8^0 \\
0
\end{array}
\]

i.e. \((300)_{10} = (454)_8\).
Example

Convert \((1654)_8\) into binary system.

Solution:

**Power Method**

First convert the given number to decimal system.

Then, convert the number from decimal system to binary system.

i.e. \((1654)_8 = 1 \times 8^3 + 6 \times 8^2 + 5 \times 8^1 + 4 \times 8^0\)

\[= 512 + 348 + 40 + 4\]

\[= (940)_{10}\]

Now, convert \((940)_{10}\) to binary system as in example 1.1.

i.e. \((1654)_{10} = (940)_{10} = (1110101100)_2\)

**Conversion of fractions of decimal system to other systems**

Example

Convert \((0.25)_{10}\) to the binary system.

Solution

\[
0.25 \times 2 = 0.50 \quad I_0 = 0 \\
0.50 \times 2 = 1.00 \quad I_1 = 1 \\
(0.25)_{10} = (0.01)_2
\]

Example

Convert \((634.640625)_{10}\) to octal equivalent.

Solution

Integer part is \((634)_{10}\)

\[
\begin{array}{c|c}
8 & 634 \\
8 & 79 \quad 2 \\
8 & 9 \quad 7 \\
& 1 \quad 1 \\
\end{array}
\]

\[\therefore \quad (634)_{10} = (1172)_8.\]

Fraction part is 0.640625

\[
0.640625 \times 8 = 5.125 \quad 5
\]
$0.125 \times 8 = 1.0$

i.e. Octal equivalent = $(1172.51)_8$

**Conversion of a fraction of any system with radix $R$ to the decimal system**

**Example**

Convert $(0.135)_8$ to the decimal system.

**Solution**

Radix is 8.

\[
(0.135)_8 = 1 \times 8^{-1} + 3 \times 8^{-2} + 5 \times 8^{-3}
\]

\[
= \frac{1}{8} + \frac{3}{64} + \frac{2}{512} = \frac{93}{512} = (0.1816406)_{10}
\]

**Example**

Convert $(3.75)_8$ to binary system.

**Solution**

The integer part $3 = (11)_2$.

The fraction part $(0.75)$ any first be converted to the decimal system.

i.e. \[
(0.75)_8 = 7 \times 8^{-1} + 5 \times 8^{-2}
\]

\[
= \frac{7}{8} + \frac{5}{64} = \frac{61}{64} = (0.953125)_{10}
\]

i.e. \[(0.75)_8 = (0.953125)_{10}.
\]

Now convert $(0.953125)_{10}$ to the binary system as under:

\[
\begin{align*}
0.953125 \times 2 &= 1.90620 & I_1 &= 1 \\
0.906250 \times 2 &= 1.81250 & I_2 &= 1 \\
0.81250 \times 2 &= 1.62500 & I_3 &= 1 \\
0.625 \times 2 &= 1.250 & I_4 &= 1 \\
0.25 \times 2 &= 0.50 & I_5 &= 0 \\
0.50 \times 2 &= 1.00 & I_6 &= 1
\end{align*}
\]

\[
\therefore (0.953125)_{10} = (0.111101)_2
\]

\[
\therefore (3.75)_8 = (11.111101)_2
\]
Problem (AMIE WINTER 98)

Convert

(i) 01011.101 binary to decimal
(ii) 22.75 decimal to binary

Answer: (i) 11.625  (ii) 1 0110.11

Direct method of conversion the number from one system to the other

For Converting the digits of decimal system into binary system, use Table 1.

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

From the above table, it is evident that a decimal digit may have at the most four bits.

For converting the digits of binary system into-decimal system, use Table 2

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexa-decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>

For converting the digits of binary system into octal system, use the Table 3
Table 3

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

Example

Convert \((1352)_8\) into binary system.

Solution

Here radix \(8 = 2^3\)

From pairs of each digit of the octal system with binary system.

\[\therefore (1352)_8 = (001, 011, 101, 010)_2\]

Example

Convert \((CD578)_{16}\) into binary system.

Solution

Here \(16 = 2^4\), power is 4

Form pairs of each digit of the hexa-decimal system into binary equivalents.

\[
\begin{align*}
C &= 12, \quad (12)_2 = 1100 \\
D &= 13, \quad (13)_2 = 1101 \\
&\quad (5)_2 = 0101 \\
&\quad (7)_2 = 0111 \\
&\quad (8)_2 = 1000 \\
\therefore & \text{ The required number is } (1100, 1101, 0101, 0111, 1000)_2.
\end{align*}
\]

Example

Convert the binary number \(1010011001.1011001\) into octal system.

Solution

Radix \(8 = 2^3\)
Example (AMIE Summer 2008, 8 marks)

Give an algorithm to convert an octal number into a binary number. Using an example, explain how the algorithm works.

Solution

Following steps are followed

(i) Create a look up table with 8 entries and their corresponding 3 bit binary code as given below:

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

(ii) Separate each octal digit from the given number and using look up table, find binary value and print it.

(iii) Repeat (ii) until all the digits are converted into binary.

Example: \((642.71)_8 = (110100010.111001)_2\)

Example (AMIE Summer 2014, 5 marks)

Write an algorithm to convert the decimal numbers into binary numbers. Explain your algorithm.

Solution

Following steps describe how to convert decimal to binary

Step 1: Divide the original decimal number by 2
Step 2: Divide the quotient by 2

Step 3: Repeat the step 2 until we get quotient equal to zero.

Equivalent binary number would be remainders of each step in the reverse order.

Decimal to binary conversion with example:
For example we want to convert decimal number 25 in the binary.

Step 1:  25 / 2  Remainder : 1 , Quotient : 12
Step 2:  12 / 2  Remainder : 0 , Quotient : 6
Step 3:   6 / 2  Remainder : 0 , Quotient : 3
Step 4:   3 / 2  Remainder : 1 , Quotient : 1
Step 5:   1 / 2  Remainder : 1 , Quotient : 0

So equivalent binary number is: 11001

That is (25)\text{10} = (11001)\text{2} 

Example (AMIE Summer 2014, 5 marks)

Write an algorithm to convert the binary numbers into octal numbers. Explain your algorithm.

Solution

Binary to octal conversion method:
Step1: Arrange the binary number in the group 3 from right side.
Step 2: Replace the each group with following values:

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

Binary to octal conversion examples:

For example we want to convert binary number 1011010101001101 to octal.

Step 1: 001 011 010 101 001 101
Step 2: \hspace{1cm} 1 \hspace{0.2cm} 3 \hspace{0.2cm} 2 \hspace{0.2cm} 5 \hspace{0.2cm} 1 \hspace{0.2cm} 5
So \((1011010101001101)_2 = (132515)_8

**Problem**

*Convert \((294.6875)_{10}\) into octal.*  
Answer: \((446.54)_{8}\)

**Problem**

*Convert \((642.71)_{8}\) into binary.*  
Answer: \((110100010.111001)_2\)

**Problem**

Convert the following numbers to their hexadecimal equivalent  
(i) \((49.5)_{10}\)  
(ii) \((972.625)_{10}\)

Answer: (i) \((13.8)_{16}\)  (ii) \((3CC.A)_{16}\)

**BINARY CALCULATIONS**

The addition, subtraction, multiplication and division of the binary numbers, may be done as in the case of decimal system.

**Addition**

While adding the binary numbers, the following rules are followed:

\[
\begin{align*}
0 + 0 &= 0 \\
0 + 1 &= 1 \\
1 + 0 &= 1 \\
1 + 1 &= 0 \text{ with 1 carry over}
\end{align*}
\]

**Example**

*Add \((110101)_2\) and \((101111)_2.*

**Solution**

\[
\begin{align*}
110101 &= 53 \\
+101111 &= 47 \\
1100100 &= 100
\end{align*}
\]

**Subtraction**

While subtracting the binary numbers, the following rules may be followed:
Example

Subtract \((100101100)_2\) from \((1110101010)_2\).

Solution

\[
\begin{array}{cccccc}
 & 1 & 1 & 1 & 0 & 1 & 0 \\
- & 1 & 0 & 0 & 1 & 1 & 1 \\
\hline
 & 1 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
\]

Example

Multiplying \((111)_2\) by \((101)_2\).

Solution

\[
\begin{array}{cccccc}
 1 & 1 & 1 & \times & 7 \\
\times & 1 & 0 & 1 & \times 5 \\
\hline
 & 1 & 1 & 1 & 35 \\
 & 0 & 0 & 0 \\
 & 1 & 1 & 1 \\
\hline
 & 1 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
\]

\[
(111)_2 \times (101)_2 = (100011)_2 \quad \text{Answer}
\]

\[
\begin{align*}
(100011)_2 &= 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
&= 32 + 0 + 0 + 2 + 1 = 35
\end{align*}
\]
Multiply 1101 by 1010
Answer: 1000010

Division
While dividing a binary number by another binary number, the rules applicable to the decimal system may be followed and of course obeying the rules of addition, subtraction and multiplication of the binary system.

Example

Dividing \((100011)_2\) by \((101)_2\)

Solution

\[
\begin{array}{c|c}
111 & 100011 \\
\hline
101 & \\
101 & \\
101 & \\
101 & \\
0 & \\
\end{array}
\]

i.e. \((100011)_2 + (101)_2 = (111)_2\)

Problem

Divide 1011011 by 111
Answer: 1101

Example (AMIE W2001)

Write following decimal number to BCD form

1782

Solution

To find BCD form of a decimal number convert each digit of the decimal number into its 4 bit binary form

\((1782)_{10} = (0001 \quad 0111 \quad 1000 \quad 0010)_{BCD}\)
Example (AMIE Summer 2001)

\[(F9A.BC3)_{16} = (?)_{10} = (?)_{2}\]

Solution

\[(F9A.BC3)_{16} = F \times 16^2 + 9 \times 16^1 + A \times 16^0 + B \times 16^{-1} + C \times 16^{-2} + 3 \times 16^{-3}\]

\[= (15 \times 56) + (9 \times 16) + (10 \times 1) + (11/16) + (12/256) + (3/4096)\]

\[= 3840 + 144 + 10 + (11/16) + (12/256) + (3/4096)\]

\[= (3994.7351074)_{10}\]

\[(F9A.BC3)_{2} = \begin{array}{cccccc}
F & 9 & A & . & B & C \\
1111 & 1001 & 1010 & 1011 & 1100 & 0011
\end{array}\]

FLOATING POINT NUMBERS

Floating-point notation can be used conveniently to represent both large as well as small fractional or mixed numbers. This makes the process of arithmetic operations on these numbers relatively much easier. Floating-point representation greatly increases the range of numbers, from the smallest to the largest, that can be represented using a given number of digits. Floating-point numbers are in general expressed in the form

\[N = m \times b^e\]  \hspace{1cm} (1)

where \(m\) is the fractional part, called the **significand** or **mantissa**, \(e\) is the integer part, called the exponent, and \(b\) is the base of the number system or numeration. Fractional part \(m\) is a \(p\)-digit number of the form \((\pm d.dddd......dd)\), with each digit \(d\) being an integer between 0 and \(b - 1\) inclusive. If the leading digit of \(m\) is nonzero, then the number is said to be normalized.

Equation (1) in the case of decimal, hexadecimal and binary number systems will be written as follows:

Decimal system

\[N = m \times 10^e\]

Hexadecimal system

\[N = m \times 16^e\]

Binary system

\[N = m \times 2^e\]

For example, decimal numbers 0.0003754 and 3754 will be represented in floating-point notation as \(3.754 \times 10^{-4}\) and \(3.754 \times 10^{3}\) respectively. A hex number 257.ABF will be represented as \(2.57ABF \times 16^2\). In the case of normalized binary numbers, the leading digit, which is the most significant bit, is always ‘1’ and thus does not need to be stored explicitly.
Also, while expressing a given mixed binary number as a floating-point number, the radix point is so shifted as to have the most significant bit immediately to the right of the radix point as a ‘1’. Both the mantissa and the exponent can have a positive or a negative value.

The mixed binary number \((110.1011)_2\) will be represented in floating-point notation as \(0.1101011 \times 2^3 = .1101011e + 0011\). Here, \(0.1101011\) is the mantissa and \(e+0011\) implies that the exponent is +3.

The IEEE-754 floating point is the most commonly used representation for real numbers on computers including Intel-based personal computers, Macintoshs and most of the UNIX platforms. It specifies four formats for representing floating-point numbers. These include single-precision, double-precision, single-extended precision and double-extended precision formats. Of the four formats mentioned, the single-precision and double-precision formats are the most commonly used ones. The single-extended and double-extended precision formats are not common.

Following figure shows the basic constituent parts of the single precision format. As shown in the figure, the floating-point numbers, as represented using these formats, have three basic components including the sign, the exponent and the mantissa.

![Single precision](image)

Single precision

For example, if we want to represent +10.25 in IEEE 754 format, the following steps will be used:

Step 1: Convert 10.25 in binary.

\[
(10.25)_{10} = 1010.01
\]

Step 2: Normalise by bringing decimal point in front of first non zero bit

\[
1010.01 = 1.01001\times 2^3
\]

Step 3: Now mantissa is 01001 and exponent is 3

Step 4: Add 127 to exponent to get exponent in excess 127 format

\[
127 + 3 = 30 = (10000010)_{2}
\]

Now the representation is
Example

What is a sign bit? If sign bit contains the binary digit 0 then the integer represents is either positive or negative. Give reason.

Answer

Sign bit is used to represent the sign (+ or -) of a binary number inside computer memory. It normally precedes the binary sequence of the number. e.g., 0, 1000000 represents +64 in decimal. e.g., 1,1100 can represent -12 in decimal when 0 represents positive sign, it is known a positive logic. It can also represent negative sign in negative logic. In 2’s complementation notation, it can be act both positive and negative inorder to remove the ambiguity of zero magnitude represent (when preceded by 0) in signed magnitude numbers.

Example

Why do suppose people adopted the decimal the decimal number system for everyday use? If you have to propose another number system so as to facilitate arithmetic computations, which radix would choose? Justify your answer. What radix would you recommend for a computer?

Answer

The decimal number system must have been adopted due to ten figures (including two thumbs) in two hands of humans. This led to convenience in counting also. It is sure that the decimal number system is the most appropriate owing to ease in arithmetic computation by people. However, one can use any of 2, 8 or 16 as radix to facilitate the arithmetic computations. Justification goes to these radices towards the inter-convertibility in a convenient way.

For a computer based switching circuits, the number system with radix 2 is the most suitable and is to be recommended. However, it depends upon the potential technology which might allow other radix systems.

Example

Explain briefly the following term

(i) Bit (ii) Byte
Bit : Short form of Binary digit. A binary digit is one of the two digits, represented by 0 and 1, that are used in the binary number system.

Byte : A group of eight bits, which forms the smallest portion of memory that an 8-bit CPU can recall from, or store in memory.

Example (AMIE SUMMER 2001)

Explain ASCII code? How many bits per character does ASCII code use?

Answer

The American standards Institution has evolved a standard code to represent characters to be stored and processed by computers. This code, called ASCII, uses 7 bits to represent each character. The ASCII code (American Standard Code for Information Interchange) defines codes for English letters (capital and small), decimal digits, 32 special characters, and codes for a number of symbols used to control the operation of a computer. The symbols used for control are non-printable.

Example (AMIE SUMMER 2001)

Explain EBCDIC?

Answer

In addition to the ASCII, another code known as Extended Binary Coded Decimal Interchange Code (EBCDIC) is used in computers manufactured by International Business machines Corporation (IBM). This code uses 8 bits per character and is thus capable of representing 256 characters. If data coded in ASCII is to be used in a computer which codes data in EBCDIC, it is necessary to transform ASCII code to EBCDIC code. A special ready made electronic circuit is available for carrying out this transformation.

Example

Write algorithm for converting binary to decimal.

Solution

Convert from binary to decimal algorithm:

For this we multiply each digit separately from right side by 1, 2, 4, 8, 16 ... respectively then add them.

Binary number to decimal conversion with example:

For example we want to convert binary number 101111 to decimal:
Computing and Informatics
Number System and Boolean Algebra

Step 1: $1 \times 1 = 1$
Step 2: $1 \times 2 = 2$
Step 3: $1 \times 4 = 4$
Step 4: $1 \times 8 = 8$
Step 5: $0 \times 16 = 0$
Step 6: $1 \times 32 = 32$

Its decimal value: $1 + 2 + 4 + 8 + 0 + 32 = 47$
That is $(101111)_2 = (47)_{10}$

**Boolean Algebra**

**Important Theorems and Expressions**

1. $A + 0 = A$
2. $A \cdot 1 = A$
3. $A + 1 = 1$
4. $A \cdot 0 = 0$
5. $A + A = A$
6. $A \cdot A = A$
7. $A + \bar{A} = 1$
8. $A \bar{A} = 0$
9. $A \cdot (B + C) = A \cdot B + A \cdot C$
10. $A + B \cdot C = (A + B) \cdot (A + C)$
11. $A + A \cdot B = A$
12. $A \cdot (A + B) = A$
13. $A + \bar{A} \cdot B = A + B$
14. $A \cdot (\bar{A} + B) = A \cdot B$
15. $A \cdot B + A \cdot \bar{B} = A$
16. $(A + B) \cdot (A + \bar{B}) = A$
17. $A \cdot B \cdot C = \bar{A} + \bar{B} + \bar{C} + \ldots \ldots$ (De Morgan’s Law)
18. \( A + B + C + \ldots = \overline{A} \overline{B} \overline{C} + \ldots \) (De Morgan’s Law)

19. \( A.B + \overline{A}.C = (A + C)(\overline{A} + B) \)

20. \( (A + B)(\overline{A} + C) = A.C + \overline{A}.B \)

21. \( A.B + \overline{A}.C + B.C = A.B + \overline{A}.C \)

22. \( (A + B)(\overline{A} + C).(B + C) = (A + B)(\overline{A} + C) \)

**Example**

Prove that

(i) \( A.(A + B) = A \)

(ii) \( A + \overline{A}.B = A + B \)

(iii) \( A.B + \overline{A}.C = (A + C)(\overline{A} + B) \)

**Solution**

(i) \( A.(A + B) = A.A + A.B = A + A.B = A.(1 + B) = A \) \([1 + B = 1]\)

(ii) \( A + \overline{A}.B = (A + \overline{A}).(A + B) \) (By theorem10) = \( A + B \)

(iii) \( A.B + \overline{A}.C = AB(C + \overline{C}) + \overline{A}C(B + \overline{B}) \)

\[= ABC + AB\overline{C} + \overline{A}BC + \overline{A}BC\]

\[= BC(A + \overline{A}) + AB\overline{C} + \overline{A}BC\]

\[= BC + AB\overline{C} + \overline{A}BC\]

\[= C(B + \overline{A}) + AB\overline{C}\]

\[= C(B + \overline{A}) + AB\overline{C} \quad [\text{Theorem 13}]\]

\[= CB + C\overline{A} + AB\overline{C}\]

\[= B(C + A\overline{C}) + \overline{A}C\]

\[= B(C + A) + \overline{A}C \quad [\text{Theorem 13}]\]

\[= BC + AB + \overline{A}C + A\overline{A}\]

\[= C(\overline{A} + B) + A(\overline{A} + B)\]

\[= (A + C)(\overline{A} + B)\]
Example

Prove that \((A + B)(\overline{A} + C)(B + C) = (A + B)(\overline{A} + C)\)

Solution

\((A + B)(\overline{A} + C)(B + C) = (B + A)(B + C)(\overline{A} + C)\) (rearranging)

\[= (B + AC)(\overline{A} + C)\] (Theorem 10)

\[= B(\overline{A} + C) + AC(\overline{A} + C)\]

\[= \overline{A}B + BC + AC\]

\[= \overline{A}B + BC + AC + A\overline{A}\]

\[= A(\overline{A} + C) + B(\overline{A} + C)\]

\[= (A + B)(\overline{A} + C)\]

Example

Factorize following Boolean equations

\(a\) \(Y = \overline{A}\overline{B} + AB\)

\(b\) \(Y = AB + AC + BD + CD\)

\(c\) \(Y = (B + CA)(C + \overline{A}B)\)

\(d\) \(Y = \overline{A}B\overline{C}D + \overline{A}B\overline{C}D + A\overline{B}\overline{C}D + A\overline{B}\overline{C}D\)

Solution

\(a\) \(Y = \overline{A}\overline{B} + AB = A(\overline{B} + B) = A.1 = A\)

\(b\) \(Y = AB + AC + BD + CD\)

\[= A(B + C) + D(B + C)\]

\[= (A + D)(B + C)\]

\(c\) \(Y = (B + CA)(C + \overline{A}B)\)

\[= B(C + \overline{A}B) + CA(C + \overline{A}B)\]

\[= BC + B\overline{A}B + CAC + CA\overline{A}B;\ A\overline{A} = 0, BB = B, CC = C\)
(d) \[ Y = \overline{A}BCD + \overline{A}B\overline{C}D + A\overline{B}C\overline{D} + AB\overline{C}D \]
\[ = \overline{B}C(\overline{A}D + \overline{A}D + AD) + \overline{A}(\overline{D} + D) + A(\overline{D} + D) \]
\[ = \overline{B}C[\overline{A}.1 + A.1] \]
\[ = \overline{B}C(\overline{A} + A) \]
\[ = \overline{B}C.1 = \overline{B}C \]

**Problem**

Prove the following identities using Boolean theorems.

(a) \[ A + \overline{A}B = A + B \]

(b) \[ (A + B)(A + C) = AC + \overline{A}B \]

(c) \[ (A + C)(A + D)(B + C)(B + D) = AB + CD \]

**SUM OF PRODUCTS/PRODUCTS OF SUM METHOD**

The switching function (Boolean function) implied by a truth table may be stated as sum of products (sop) or a product of sums (pos). For example for an EXOR truth table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
<th>Minterm</th>
<th>Maxterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\overline{A}\overline{B}</td>
<td>\overline{A} + B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>\overline{A}B</td>
<td>\overline{A} + B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>\overline{A}B</td>
<td>\overline{A} + B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>AB</td>
<td>\overline{A} + \overline{B}</td>
</tr>
</tbody>
</table>

Output is given by the sum of minterm corresponding to 1 output.
Y = \overline{AB} + A\overline{B} + AB; (sop form)

Output can be written as the product of those maxterms which correspond to 0 output.

Y = A + B; (pos form)

The equivalence of these expressions is established below:

\begin{align*}
\overline{AB} + A\overline{B} + AB &= \overline{AB} + A(\overline{B} + B) \\
&= \overline{AB} + A + \overline{B}C \\
&= (A + \overline{A})(A + B) \text{ Distributive} \\
&= (A + B)
\end{align*}

Example

Obtain sop and pos expressions for the truth table of the following table.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Decimal Eq.</th>
<th>Y</th>
<th>Minterm</th>
<th>Maxterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ABC</td>
<td>A + B + C</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>ABC</td>
<td>A + B + C</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>\overline{ABC}</td>
<td>A + B + C</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>\overline{ABC}</td>
<td>A + B + C</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>\overline{ABC}</td>
<td>A + B + C</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>\overline{ABC}</td>
<td>A + B + C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>\overline{ABC}</td>
<td>A + B + C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>ABC</td>
<td>A + B + C</td>
</tr>
</tbody>
</table>

From above table picking up the minterms corresponding to \(Y = 1\) in the truth table, we can write

\begin{align*}
Y &= \overline{ABC} + \overline{ABC} + A\overline{B}C + ABC; \text{ sop form}
\end{align*}

In term of maxterms which correspond to \(Y = 0\), we can write

\begin{align*}
Y &= (A + B + C)(A + \overline{B} + C)\overline{(A + B) + C}; \text{ pos form}
\end{align*}

We observed that

\begin{align*}
Y + \overline{Y} &= 1
\end{align*}

So while that minterms which correspond to 1 outputs constitute sop, the complimentary minterms which correspond to 0 outputs, their sop would then constitute \(\overline{Y}\) i.e.
similarly for maxterms the outputs corresponding to 1 outputs would constitute \( \overline{Y} \) i.e.

\[
\overline{Y} = (A + B + C)(A + \overline{B} + C)(\overline{A} + B + C)(\overline{A} + \overline{B} + C)
\]

Example

*Find the equivalent product-of-sums expression of the sum-of products (SOP) expression* 

\[A.B + \overline{A}\overline{B}\]

Solution

The dual of the given expression \(A.B + \overline{A}\overline{B}\)

\[\overline{(A + B)} = \overline{A} = A.B + \overline{B} = 0 + \overline{A}\overline{B} + A.B + 0 = A.B + \overline{A}\overline{B}
\]

The dual of \((A.B + \overline{A}\overline{B}) = (A + B)(\overline{A} + B)\)

Therefore \(A.B + \overline{A}\overline{B} = (A + B)(\overline{A} + B)\)

This is POS expression.

Example (AMIE Summer 2008, 5 marks)

*Draw the truth table for the boolean function \(\overline{ABC} + A\overline{B}C + AB\).*

Solution

Given boolean expression

\[
\overline{ABC} + A\overline{B}C + AB = \overline{ABC} + A\overline{B}C + AB(\overline{C} + \overline{C})
\]

\[
= \overline{ABC} + A\overline{B}C + ABC + AB\overline{C} \text{ ← SOP canonical form}
\]

Truth table can easily be obtained by making a 1 at the output column for each min term in above expression.

Minterms

\[
\begin{align*}
\overline{ABC} &\rightarrow 011 \\
A\overline{B}C &\rightarrow 101 \\
AB\overline{C} &\rightarrow 110 \\
ABC &\rightarrow 111
\end{align*}
\]
LOGIC GATES

A logic gate is an electronic circuit, which accepts a binary input and produces a binary output namely 0 and 1. The inverter (NOT) logic gate has one input and one output, but a logic gate in general accepts one or more inputs and produces one output. Apart from the NOT gate there are six other types of logic gates.

Input to a gate will be designated by binary variables A, B, C etc. and the output will be indicated by binary variable Y. As stated earlier, a binary variable can take on values 0 and 1 which are electronically represented by LOW and HIGH voltage levels. In terms of boolean algebra the function of a logic gate will be represented by a binary expression.

AND Gate

In this gate, output will be HIGH only if both the inputs are HIGH and so both diodes are OFF. Such a logic operation is called AND and is represented by the Boolean expression.

\[ Y = A \text{ AND } B = A \cdot B \]

wherein dot indicates ANDing.

(a) Symbol          (b) Truth table

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>Y</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The logic symbol for an AND gate is drawn in given figure (a) with its truth table given in figure (b).
A NOT gate is a one-input, one-output logic circuit whose output is always the complement of the input. That is, a LOW input produces a HIGH output, and vice versa. When interpreted for a positive logic system, a logic ‘0’ at the input produces a logic ‘1’ at the output, and vice versa. It is also known as a ‘complementing circuit’ or an ‘inverting circuit’. Following figure shows the circuit symbol and the truth table.

\[
\begin{array}{c|c}
X & Y = \overline{X} \\
\hline
0 & 1 \\
1 & 0 \\
\end{array}
\]

The NOT operation on a logic variable \( X \) is denoted as \( \overline{X} \) or \( X' \). That is, if \( X \) is the input to a NOT circuit, then its output \( Y \) is given by \( \overline{X} \) or \( X' \) and reads as \( Y \) equals NOT \( X \). Thus, if \( X = 0 \), \( Y = 1 \) and if \( X = 1 \), \( Y = 0 \).

**OR Gate**

The output will be HIGH, if \( A \) or \( B \) (or both) are HIGH. Such a logic operation is called OR and is expressed by the Boolean expression

\[
Y = A \text{ OR } B = A + B
\]

The symbolic representation of OR and its truth table are given in Figs. (a) and (b).

\[
\begin{array}{c|c|c}
A & B & Y \\
\hline
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

**NAND Gate**

It is an AND gate followed by a NOT gate and is represented by a symbol as shown in figure (a). Its truth table is given in figure (b). A small circle in its symbol shows NOT operation. Its Boolean expression is

\[
Y = \text{NOT}(A \text{ AND } B) = \overline{A}.B; \text{complement of AND}
\]
NAND gate may have more than two inputs.

**NOR Gate**

Logically a NOR gate is expressed as

\[ Y = \overline{A \lor B} = \overline{A} + \overline{B}; \text{ complement of Or} \]

Its symbol and truth table are shown in figures (a) and (b).

Universality of NAND/NOR Gates

NAND and NOR gates are called universal logic gates, because any logic operation can be realized by using these gates.

Consider the NAND gate of figure (a) with input A at both ports.

Then

\[ Y = \overline{A \cdot A} = \overline{A} = \text{NOT } A \]

Similarly for NOR gate

\[ Y = \overline{A + A} = \overline{A} = \text{NOT } A \]
OR Realization by NAND/NOR

Consider following circuit which shows OR realization by NAND gates.

![OR realization by NAND gates](image)

Its truth table is given by figure (b).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) Truth table

It follows that

\[ Y = \overline{A}.\overline{B} = A + B = A \text{ OR } B \]

OR realization by NOR gate is given in figure (c).

![NOR realization of OR](image)

(c) NOR realization of OR

Exclusive OR (EXOR) Gate

The output of this gate is high only when either of the inputs is high but not when both inputs are high i.e. either, but not both.

Symbol of XOR gate and its truth table are given in given figures (a) and (b).
Generally a symbol $\oplus$ is used in XOR equations. It immediately follows that

$$Y = A \text{ XOR } B = A \oplus B$$

NAND implementation of Exclusive OR is shown in figure below.

**Exclusive NOR Gate**

It is gate complementary of EXOR gate. Its symbol and truth table are given in figures in (a) and (b).

**FLIP FLOPS**

**Basic Memory Cell**

The basic memory cell is a circuit that stores one bit of information. This one bit memory element is called a *flip-flop* or *latch*, since it latches(or locks) data in it.

A flip flop has two outputs, $Q$ and $\bar{Q}$, that are always complements of each other. It can exist in two stable states; *set* and *reset*. In set state, $Q$ is HIGH(logic 1) and $\bar{Q}$ is LOW(logic 0). In reset state $\bar{Q}$ is HIGH(logic 1) and $Q$ is LOW(logic 0). For a flip flop to act as a memory device, it should retain the information stored in it. Thus, if it is in set state it should remain set and if it is in reset state, it should remain reset.
The circuit shown below is a basic NAND latch. The inputs are generally designated "S" and "R" for "Set" and "Reset" respectively. Because the NAND inputs must normally be logic 1 to avoid affecting the latching action, the inputs are considered to be inverted in this circuit.

![NAND Latch Circuit Diagram]

The outputs of any single-bit latch or memory are traditionally designated Q and Q'. In a commercial latch circuit, either or both of these may be available for use by other circuits. In any case, the circuit itself is:

For the NAND latch circuit, both inputs should normally be at a logic 1 level. Changing an input to a logic 0 level will force that output to a logic 1. The same logic 1 will also be applied to the second input of the other NAND gate, allowing that output to fall to a logic 0 level. This in turn feeds back to the second input of the original gate, forcing its output to remain at logic 1.

Applying another logic 0 input to the same gate will have no further effect on this circuit. However, applying a logic 0 to the other gate will cause the same reaction in the other direction, thus changing the state of the latch circuit the other way.

Note that it is forbidden to have both inputs at a logic 0 level at the same time. That state will force both outputs to a logic 1, overriding the feedback latching action. In this condition, whichever input goes to logic 1 first will lose control, while the other input (still at logic 0) controls the resulting state of the latch. If both inputs go to logic 1 simultaneously, the result is a "race" condition, and the final state of the latch cannot be determined ahead of time.

**SR Flip Flop (Latch)**

To serve the purpose of storing desired bits in flip flops, two input NAND or NOR gates are used as shown in following figure. This way we get an S-R flip flop.

![SR Flip Flop (Latch) Diagram]
Truth table is given below.

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>A₁</th>
<th>A₂</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>No change</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Reset</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Set</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Race</td>
</tr>
</tbody>
</table>

**Set state:** When S = 1 and R = 0, A₁ becomes 0, making Q = 1. So, \( Q = 1,\overline{Q} = 0 \); so it is called set state.

**Reset state:** When S = 0 and R = 1, A₂ becomes 0, making \( Q = 1 \) and \( Q = \overline{1},\overline{Q} = 0 \). This input condition resets the flip flop(\( Q = 0,\overline{Q} = 1 \)); so it is called reset state.

**No change:** When \( S = R = 0 \), \( Q = 1,\overline{Q} = Q \) and \( Q = \overline{1},\overline{Q} = \overline{Q} \). The flip flop remains in whatever state it is in.

**Race:** When \( S = R = 1 \), A₁ and A₂ both become 0. So \( Q = \overline{Q} = 1 \). This is an undesired output state because if S and R now change to 0, A₁ and A₂ will be 1, both Q and \( \overline{Q} \) will try to become 0. The actual state of the flip flop depends on the relative delays of the two gates. If \( N₂ \) is faster, \( \overline{Q} \) will become \( \overline{1},\overline{Q} = 0 \) first and will make \( Q = 1 \). Similarly if \( N₁ \) is faster, \( Q \) will become 0 first and will make \( Q = \overline{1} \). This is called **race condition**. Here the state of the flip flop is uncertain, so this condition is not allowed.

This circuit is called an SR flip flop. It is an asynchronous circuit because the output changes with changes as and when the S, R input change. If we want synchronous operation, a clock pulse can be introduced at the inputs as discussed below.

**Clocked SR Flip Flop (Latch)**

The SR flip flop modified to include clock(CK) pulses is drawn in following figure.
When \( CK = 1, A_1 = \overline{S} \) and \( A_2 = \overline{R} \), the circuit functions like the SR flip flop already discussed. When \( CK = 0, A_1 = A_2 = 1 \), the circuit reduces to that of a basic memory cell and outputs remain unchanged (latched).

Here \( N_1 \) and \( N_2 \) form the basic latch, whereas \( N_3 \) and \( N_4 \) control the state of the flip flop.

The truth table and logic symbol of SR flip flop are shown below.

**Truth table for clocked SR flip flop**

<table>
<thead>
<tr>
<th>CK</th>
<th>S</th>
<th>R</th>
<th>( Q_{n+1} )</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>( Q_n )</td>
<td>SR enabled</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>SR enabled</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>SR enabled</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>?</td>
<td>Not allowed</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>( Q_n )</td>
<td>SR disabled</td>
</tr>
</tbody>
</table>

Symbol clocked SR

**JK FLIP FLOP**

As discussed earlier, in SR flip flop, the input combination \( S = R = 1 \) is not allowed because its output is uncertain. This uncertainty may cause problems in a digital system. So, we need a one bit memory cell which has its outputs well defined for all possible input combinations.

One such device is JK flip flop with inputs \( J \) and \( K \). Also, let this flip flop have the same truth table as the SR flip flop except for the condition when \( J = K = 1 \). Since \( Q_{n+1} = Q_n, 1, 0 \) are already occurring in the truth table of the SR flip flop, let \( Q_{n+1} = \overline{Q_n} \) for the new condition \( (J = K = 1) \). Following table gives the combined truth table of SR/JK flip flops.

**Combined truth table of SR/JK flip flops**

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>( Q_n )</th>
<th>( Q_{n+1} )</th>
<th>S</th>
<th>R</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( 0/1 ) Inactive state</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>( Q_{n+1} = Q_n )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Reset state</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>( Q_{n+1} = 0 )</td>
</tr>
<tr>
<td>J</td>
<td>K</td>
<td>$Q_n$</td>
<td>$Q_{n+1}$</td>
<td>S</td>
<td>R</td>
<td>Comment</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---------</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Set state</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1/0</td>
<td>0</td>
<td>$Q_{n+1} = 1$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Toggle state</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$Q_{n+1} = Q_n$</td>
</tr>
</tbody>
</table>

The K-maps with S and R as outputs, and taking J, K and $Q_n$ as inputs are shown below.

$$S = J\bar{Q}_n$$

<table>
<thead>
<tr>
<th>JK → $Q_n$</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>x</td>
</tr>
</tbody>
</table>

$$R = K\bar{Q}_n$$

<table>
<thead>
<tr>
<th>JK → $Q_n$</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Thus we have a simple way of designing a JK flip flop from the SR flip flop. The circuit corresponding to K maps given earlier is drawn below.

This circuit can be simplified to following circuit.
Truth table of above JK flip flop is given below:

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>Q_{n+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q_n</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>\overline{Q_n}</td>
</tr>
</tbody>
</table>

**Race Around Condition**

The solution that we found in the JK flip flop, for the condition when both the inputs of the flip flop are high, is not perfect. For the level triggered JK flip flop of above figure, consider the case when J = K = 1, Q_n = 0 and a clock pulse occurs. After a propagation delay of \( \Delta t (< t_p, \text{ the pulse width}) \) equal to the delay time of two NAND gates, the output will toggle to \( Q_n = 1 \). Since this is feedback to the inputs, the output can toggle back to \( Q_n = 0 \) after another delay of \( \Delta t \), if the clock is still high.

Thus, as long as the clock pulse is present, the output will toggle at every \( \Delta t \) seconds. This is called race around condition.

**Removing race around condition**

- Making propagation delay \( \Delta t > t_p \)
- Passing outputs \( Q_n \) and \( \overline{Q_n} \) through delay lines before feeding them back to input
- Use a JK master slave flip flop

**JK MASTER SLAVE FLIP FLOP**

This flip flop is made of two SR flip flops. The output of the first, called the **master** flip flop, is given to the inputs of the second, called the **slave** flip flop. The output of the slave is feedback to the inputs of the master. The master is triggered by a positive clock pulse and the slave is triggered by a negative clock pulse, obtained by inverting the clock pulse applied to the master. The circuit of MS flip flop is shown below.
When the clock is HIGH (CK = 1, Pr = 1, Cr = 1), the master is enabled, while the slave is disabled as CK = 0. So, the master functions like a JK flip flop and its output appears at the input (S, R) of the slave. Since CK is LOW, the slave is inactive. Thus Q remains unchanged for the duration of the clock pulse t_p.

When the clock goes LOW, the slave functions like an SR flip flop and its output changes to Q_M, which also appears at the input of the master. Since the clock is LOW, the master is inactive so that Q_M (and so S and R) remains unchanged as long as the clock remains LOW.

Thus, when the clock is HIGH, Q_M changes according to JK flip flop logic and it is transferred to Q when the clock goes LOW (negative edge transition). This eliminates the race around condition as the output remains constant for the duration of one clock pulse. The truth table of this flip flop is shown below.

### Truth table of MS JK flip flop

<table>
<thead>
<tr>
<th>Pr</th>
<th>Cr</th>
<th>CK</th>
<th>J</th>
<th>K</th>
<th>Q_{n+1}</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>?</td>
<td>Not desired</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>Preset</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>Clear</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>Q_n</td>
<td>Inactive</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Q_n</td>
<td>Zero</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Q_n</td>
<td>One</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Q_n</td>
<td>Toggle</td>
</tr>
</tbody>
</table>

### D FLIP FLOP

In a D flip flop the output follows the input whenever the flip flop is triggered. This can be achieved by means of a JK/SR flip flop. A D flip using JK/SR flip flop along with its truth table is shown below.

### Truth table of D flip flop

<table>
<thead>
<tr>
<th>D</th>
<th>Q_{n+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Here the output follows the input, when the flip flop is triggered. If the flip flop is edge triggered the output is the same as the input, but is delayed by one clock pulse. Owing to this property, the D flip flop is used as a delay device. Its function is illustrated by following waveform.
A T flip flop can be realized by JK flip flop. T flip flop using JK flip flop along with truth table is given below.

\[
\begin{array}{cccc}
T & J & K & Q_{n+1} \\
0 & 0 & 0 & Q_n \\
1 & 1 & 1 & \overline{Q}_n \\
\end{array}
\]

T flip flops are widely used as toggle switches.

**SHIFT REGISTER**

Shift registers store as well as manipulate data. Following figure shows a 4-bit shift register.

Four JK(or SR) flip flops are converted into D flip flops by giving mutually complementary inputs. All the four flip flops are triggered by a common clock and can be cleared by giving a 0 at the CLR input. During normal operation, this input is held at 1.
Arithmetic Circuits

A logic gate is an electronic circuit, which accepts a binary input and produces a binary output namely 0 and 1. The inverter (NOT) logic gate has one input and one output, but a logic gate in general accepts one or more inputs and produces one output. Apart from the NOT gate there are six other types of logic gates.

Input to a gate will be designated by binary variables A, B, C etc. and the output will be indicated by binary variable Y. As stated earlier, a binary variable can take on values 0 and 1 which are electronically represented by LOW and HIGH voltage levels. In terms of boolean algebra the function of a logic gate will be represented by a binary expression.

HALF ADDER

This circuit adds two binary variables, yields a carry but does not accept carry from another circuit (adder). The truth table of half adder is given in below.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>S</th>
<th>C</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>

From this table

\[ S = \overline{A}B + AB = A \oplus B \]
\[ C = AB \]

Half adder logic circuit is shown in given figure.

Circuit using XOR

Half adder circuit using NOR gates is shown in given figure.

Half adder circuit using NOR gates
Here \[ S = \overline{A}B \overline{C}_i + \overline{A}B\overline{C}_i + \overline{A}BC_i + A BC_i \] (1)
and \[ C_0 = \overline{A}BC_i + A\overline{B}C_i + AB\overline{C}_i + ABC_i \] (2)

Recognizing that \( Y + Y + Y + \ldots + Y = Y \) and adding \( ABC_i \) twice to right hand side of eq. (2), we can write
\[ C_0 = (\overline{A}BC_i + A BC_i) + (A\overline{B}C_i + ABC_i) + (AB\overline{C}_i + ABC_i) \]

Observing that \( Y + \overline{Y} = 1 \), we get
\[ C_0 = BC_i + AC_i + AB \] (3)

The following is easily established
\[ S = [(\overline{A}C_i + BC_i + AB) + ABC_i](A + B + C_i) \]
\[ = [(\overline{A}C_i . BC_i . AB) + ABC_i](A + B + C_i) \]
\[ = (\overline{A}C_i . BC_i . AB)(A + B + C_i) + ABC_i \]
\[ = (\overline{A} + C_i)(\overline{B} + \overline{C}_i)(\overline{A} + \overline{B})(A + B + C_i) + ABC_i \]
\[ = \overline{A}BC_i + \overline{A}B\overline{C}_i + \overline{A}BC_i + ABC_i = S \]
Using AND OR INVERT(AOI) gates, equations (3) and (4) are implemented in figure given, which gives outputs $S$ and $C_0$.

**COMBINATIONS CIRCUITS VS. SEQUENTIAL CIRCUITS**

The main difference between sequential circuits and combinational circuits is that sequential circuits compute their output based on input and state, and that the state is updated based on a clock. Combinational logic circuits implement Boolean functions, so they are functions only of their inputs, and are not based on clocks. A sequential circuit uses flip flops. Unlike combinational logic, sequential circuits have state, which means basically, sequential circuits have memory. Here are some of the key things to notice:

- Like combinational logic circuits, a sequential logic circuit has inputs (labelled with $x$ with subscripts) and outputs (labelled with $z$ with subscripts).
- Unlike combinational logic circuits, a sequential logic circuit uses a **clock**.
- Also, there is a box inside the circuit called **State**.
- This box contains flip flops. Assume it has $k$ flip flops. The flip flops basically store a $k$-bit number representing the current state.
- The output $z$ is computed based on the inputs ($x$ with subscripts) and the state coming out of the state box ($q$ with subscripts).
- The state may be updated at each positive clock edge. When there's not a positive clock edge, the state remains unchanged.
- The information needed to update to the state (called the **next state**) comes from the current state (the current value of $q$) and the input, which is fed through combinational logic, and fed back into the state box, telling the state box how to update itself.

For example, suppose you are currently in state 00, and see an input of 1. This may produce an output of, say, 10, and then produce feedback that tells the state box to update to state 01 by the next clock edge.