Unit 2: Number Systems, Codes and Logic Functions

Introduction

A digital computer manipulates discrete elements of data and that these elements are represented in the binary forms. Operands used for calculations can be expressed in the binary number system. Other discrete elements including the decimal digits, are represented in binary codes. Data processing is carried out by means of binary logic elements using binary signals. Quantities are stored in binary storage elements. The purpose of this unit is to introduce the various binary concepts as a frame of reference for further study in the succeeding units

Lesson 1 : Number Systems

1.1 Learning Objective

On completion of this lesson you will be able to :

- describe different number systems
- identify decimal, binary and other numbers
- explain different number systems with examples.

1.2 Number Systems

Any positive integer b>1 can be chosen as the base for a positional number system similar to the decimal system (b=10) or the binary system (b=2). Such a system uses b symbols for the integers

$$0, 1, 2, 3, ..., b-1$$

These symbols are called the digits of the system.

Number Systems

Any integer N is represented in the system by a sequence of base-b digits :

$$N = a_n a_{n-1} ... a_1 a_0$$

Then b^k is the place value of a_k, and

$$N = a_n \times b^n + a_{n-1} \times b^{n-1} + ... + a_2 \times b^2 + a_1 \times b^1 + a_0 \times b^0$$

1.3 Decimal System

Any positive integer N, represented in the decimal system as a string of decimal digits, may also be expressed as a sum of powers of 10, with each power weighted by a digit. For example, N = 8253 can be expressed as follows :

$$8253 = 8\times10^3 + 2\times10^2 + 5\times10^1 + 3\times10^0$$
$$= 8\times1000 + 2\times100 + 5\times10 + 3\times1$$
$$= 8000 + 200 + 50 + 3$$

Decimal System

The powers of ten,

$10^0 = 1$ $10^1 = 10$ $10^2 = 100$ $10^3 = 1000$ ...

which correspond respectively to the digits in a decimal integer as read from right to left, are called the place values of the digits.

Any fractional value M, represented in the decimal system by a string of decimal digits together with an embedded decimal point, may also be expressed in expanded notation by using negative powers of 10. Specifically, the place values of the digits in M to the decimal point are respectively

$10^{-1} = 0.1$ $10^{-2} = 0.01$ $10^{-3} = 0.001$ ...

For example, M = 837.526 is expressed in expanded notation as follows:

$$837.526 = 8\times10^2 + 3\times10^1 + 7\times10^0 + 5\times10^{-1} + 2\times10^{-2} + 6\times10^{-3}$$
$$= 800 + 30 + 7 + 0.5 + 0.02 + 0.006$$

This decimal fraction is said to have three decimal places, the number of digits to the right of the decimal point.

The arithmetic of decimal fractions is not very complicated; one has to keep track of the decimal points.

1.4 Binary System

Any binary number is therefore a sequence of bits, possibly with an embedded binary point. Those numbers that have no fractional part, i.e., are without an embedded binary point, are called binary integers.

Binary System

The place values in the binary system are the powers of the base b=2, just as the place values in the decimal systems are the powers of ten.

Specifically, the place values of the integral part of a binary number are the nonnegative powers of two.

2^0 2^1 2^2 2^3 ...

and the place values of the fractional part of a binary number are the negative powers of two,

2^{-1} 2^{-2} 2^{-3} ...

Table 1.1.

Binary place values	Decimal values
2^{-4}	0.0625
2^{-3}	0.125
2^{-2}	0.25
2^{-1}	0.5
2^0	1
2^1	2
2^2	4
2^3	8

1.5 Octal System

Since $8 = 2^3$, each octal digit has a unique 3-bit binary representation, given in Table 1.2.

Octal System

Table 1.2

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

The place values in the octal system are powers of 8; some of the these powers appear in Table 1.3.

Table 1.3

Octal Place Values	Decimal Values
8^{-3}	1/512 = 0.001953125
8^{-2}	1/64 = 0.015625
8^{-1}	1/8 = 0.125
8^{0}	1
8^{1}	8
8^{2}	64
8^{3}	512
8^{4}	4096
8^{5}	32768

1.6 Hexadecimal System

Hexadecimal System

Since $16 = 2^4$, each hexadecimal digit has a unique 4-bit representation which is shown in Table 1.4. The place values in the hexadecimal system are the powers of 16, some of which are listed, along with their decimal values, in Table 1.5.

Table 1.4

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

Table 1.5

Hexadecimal Place Values	Decimal Values
16^{-3}	1/4096 = 0.000244140625
16^{-2}	1/256 = 0.00390625
16^{-1}	1/16 = 0.0625
16^{0}	1
16^{1}	16
16^{2}	256
16^{3}	4096
16^{4}	65536
16^{5}	1048576

1. 7 Exercise

1. Multiple choice questions

a. The place values in the decimal systems are the powers of

i) 2
ii) 8
iii) 10
iv) 16.

b. The place values of the fractional part of a binary number are the

i) nonnegative powers of 2
ii) negative powers of 2
iii) negative powers of 10
iv) negative powers of 8.

c. Each hexadecimal digit has a unique

i) 2-bit binary representation
ii) 3-bit binary representation
iii) 4-bit binary representation
iv) 5-bit binary representation.

2. Analytical questions

i) How 625.536, 0.326, 735, 1278 can be expressed in expanded notation in respect of decimal system?
ii) Identify the number of digit needed to express decimal, binary, octal & hexadecimal system.

3. Questions for short answers

i) What are the bases of binary, decimal, octal and hexadecimal numbers?
ii) Give binary values for hexadecimal number AF3.

Lesson 2 : Conversion of Numbers

2.1 Learning Objective

On completion of this lesson you will be able to :

- identify the differences between different number systems
- convert binary to decimal
- convert decimal to binary
- interconvert hexadecimal-decimal
- interconvert hexadecimal-binary.

2.2 Binary-to-Decimal Conversion

Any binary number can be written in expanded notation as the sum of each digit times that digit's place value. For example,

$110101 = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$101.110 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3}$

Since each power of two is weighted by either 0 or 1, the binary number is simply the sum of those place values in which the bit 1 appears. This sum at once gives us the decimal equivalent of the binary number.

Table 1.6 lists the binary representations of the integers from 0 to 25, with the place of the bits shown at the top of the table. Sometimes a subscript 2 is used to distinguish a binary number, e.g. one may write 101011_2 if it is not clear from the context that 101011 is a binary number rather than a decimal number. Also, for easier reading, one sometimes separates a binary number into 4-bit groups, to the left and right of the binary point; e.g.

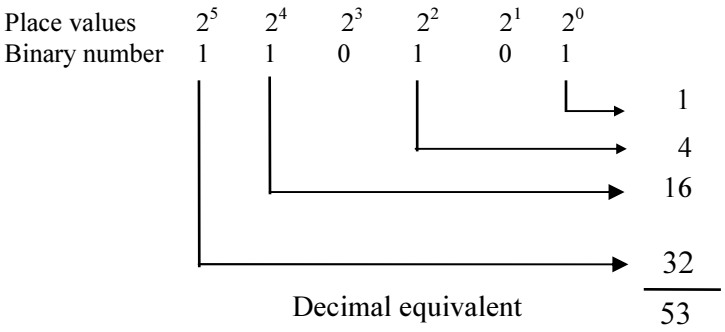
10110100.011010 might be written 1011 0100.0110 10

Table 1.6

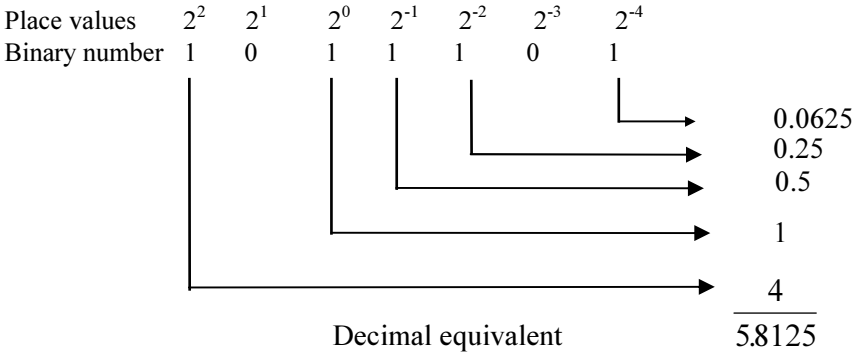
Decimal Number	Binary Number 16s 8s 4s 2s 1s
0	0
1	1
2	1 0
3	1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1
8	1 0 0 0
9	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1
16	1 0 0 0 0
17	1 0 0 0 1
18	1 0 0 1 0
19	1 0 0 1 1
20	1 0 1 0 0
21	1 0 1 0 1
22	1 0 1 1 0
23	1 0 1 1 1
24	1 1 0 0 0
25	1 1 0 0 1

Example 2.1:

(a) To convert 110101_2 to its decimal equivalent, write the appropriate place value over each bit and then add up those powers of two which are weighted by 1 :



(b) To convert 101.1101_2 to its decimal equivalent, use Table 1.1 for the decimal values of the negative powers of two



2.3 Decimal-to-Binary Conversion

It is possible to find binary representation of a decimal number N by converting its integral part (N_I), and its fractional part (N_F) separately. It is illustrated with the decimal number N = 109.78125.

Example 2.2

(a) To convert N_I = 109 to binary equivalent, divide N_I and each successive quotient by 2, noting the remainders, as follows :

Decimal-to-Binary Conversion

Divisions	Quotients	Remainders
109 ÷ 2	54	1
54 ÷ 2	27	0
27 ÷ 2	13	1
13 ÷ 2	6	1
6 ÷ 2	3	0
3 ÷ 2	1	1
1 ÷ 2	0	1

The zero quotient indicates the end of the calculations. The sequence of remainders from the bottom to up, as indicated by the arrow, yields the required binary equivalent. That is $N_I = 109 = 1101101_2$.

In practice, the above divisions may be condensed as follows :

	Remainders
2)109	
2)54	1
2)27	0
2)13	1
2)6	1
2)3	0
1	1

Here stop when the quotient, 1, is less than the divisor 2, since this last quotient will be next and last remainder. Again the arrow indicates the sequence of bits that gives the binary equivalent of the number.

(b) To convert $N_F = 0.78125$ to its binary equivalent, multiply N_F and each successive fractional part by 2, noting the integral part of the product, as follows :

Multiplications	Integral parts
$0.78125 \times 2 = 1.56250$	1
$0.5625 \times 2 = 1.1250$	1
$0.125 \times 2 = 0.250$	0
$0.25 \times 2 = 0.50$	0
$0.50 \times 2 = 1.00$	1

The zero fractional part indicates the end of the calculations. It is observed that the integral part of any product can only be 0 or 1, since it is required to double number which is less than one. The sequence of integral-part digits from the top to down, as indicated by the arrow, yields the required binary equivalent. That is, $N_F = 0.78125 = 0.11001_2$.

In practice, the above multiplications may be condensed as follows :

0.781 25
×2
1.562 50
×2
1.125 00
×2
0.250 00
× 2
0.500 00
×2
1.000 00

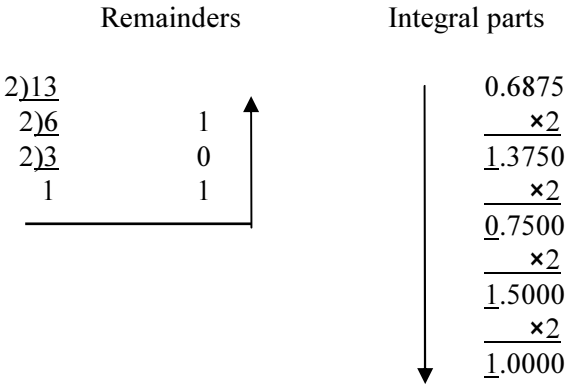
It is observed that the integral part of each product is underlined and does not figure in the next multiplication. Again the arrow indicates the sequence of integral or integral-part digits that give the required binary representation.

It is found that the binary equivalents of the integral and fractional parts of the decimal number N = 109.78125. The binary equivalent of N is simply the sum of these two equivalents:

$$N = N_I + N_F = 1101101.11001$$

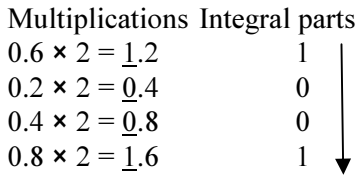
Example 2.3 :

Let, N = 13.6875. Convert the integral part, N_I = 13, and the fractional part, N_F = 0.6875, into binary forms :



Thus, N = 13.6875 = 1101.1011_2.

Remark: The binary equivalent of a terminating decimal fraction does not always terminate. For example, convert N = 0.6 as above:



At this point in the procedure, one can again multiply 0.6 by 2. This means the above four steps will be repeated again and again. That is,

$$N = 0.6 = 0.1001\ 1001\ 1001\ ..._2$$

(The number of bits which repeat is not always four; nor does the repeating block necessarily begin at the binary point, it depends on the given N.)

2.4 Hexadecimal-Decimal Interconversion

Hexadecimal-Decimal Inter-conversion

Conversion between the hexadecimal and decimal systems is accomplished via the algorithms of Section 2.3 with b = 16. There is an added difficulty in that one has to know how to handle the hexadecimal digits A, B, C, D, E and F. One can also convert from hexadecimal to decimal by decimal evaluation of the expanded hexadecimal form.

Example 2.4 :

(a) To convert $73D5_{16}$ to its decimal equivalent, express the number in expanded notation, change D to 13, and then calculate using decimal arithmetic.

$$73D5_{16} = 7\times16^3 + 3\times16^2 + 13\times16^1 + 5\times16^0$$
$$= 7\times4096 + 3\times256 + 13\times16 + 5\times1$$
$$= 28672 + 768 + 208 + 5 = 29653$$

Alternatively, one can apply the conversion algorithm as follows :

$$\begin{array}{r} 7 \\ \times 16 \\ \hline 112 \\ +3 \\ \hline 115 \\ \times 16 \\ \hline 1840 \\ +13 \\ \hline 1853 \\ \times 16 \\ \hline 29648 \\ +5 \\ \hline 29653 = 73D5_{16} \end{array}$$

(b) Convert $39.B8_{16}$ to its decimal equivalent as follows :

$$39.B8_{16} = 3 \times 16^1 + 9 \times 16^0 + 11 \times 16^{-1} + 8 \times 16^{-2}$$
$$= 3 \times 16 + 9 \times 1 + 11 \times 0.0625 + 8 \times 0.00390625$$
$$= 48 + 9 + 0.6875 + 0.03125 = 57.71875$$

(c) To convert the decimal number P = 9719 to its hexadecimal equivalent, divide P, and each successive quotient by the base b = 16, noting the remainders, as follows :

Divisions	Quotients	Remainders
9719 ÷ 16	607	7
607 ÷ 16	37	15
37 ÷ 16	2	5
2 ÷ 16	0	2

The sequence of remainders, which replaces the decimal remainder 15 by the hexadecimal digits F, in reverse order, gives the hexadecimal form for P; i.e. $P = 25F7_{16.}$

(d) To convert the decimal fraction Q = 0.78125 to its hexadecimal equivalent, apply the integral-part algorithm, with b = 16, as follows :

Multiplications	Integral parts
$0.78125 \times 16 = \underline{12}.50000$	12 ↓
$0.50000 \times 16 = \underline{8}.00000$	8

In this case a zero fractional part is reached. The sequence of integral parts, which replace the decimal 12 by the hexadecimal digit C, gives the required hexadecimal form for $Q = 0.C8_{16}$.

(e) To convert the decimal number N= 9719.78125 to its hexadecimal form, add the representations found in (c) and (d) :

$N = P + Q = 25F7.C8_{16}$

Hexadecimal-Binary Interconversion

This is accomplished exactly as octal-binary interconversion, except that 4-bit equivalents are now involved.

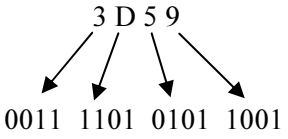
Hexadecimal-Binary Inter-conversion

Example 2.5 :

Convert to binary form (a) $3D59_{16}$, (b) $27.A3C_{16}$.

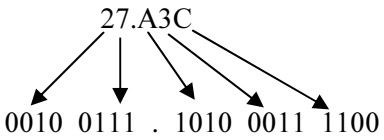
Replace each hexadecimal digit by its 4-bit representation (Table 1.4)

(a)



Hence, $3D59_{16} = 11110101011001_2$.

(b)



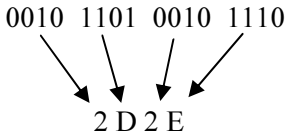
Hence, $27.A3C_{16} = 100111{\cdot}1010001111_2$.

Example 2.6 :

Convert to hexadecimal form (a) 10110100101110_2, (b) 11100.1011011011_2.

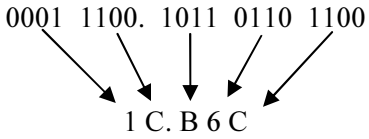
Partition each binary number into 4-bit blocks to the left and right of the binary point adding 0s if necessary. Then replace each 4-bit block by its equivalent hexadecimal digit (Table 1.4).

(a)



Hence, $2D2E_{16}$ is the required hexadecimal form.

(b)



Hence, $1C.B6C_{16}$ is the required hexadecimal form.

2.5 Exercise

1. Multiple choice questions

a. The decimal equivalent of 1110_2 is

i) 8
ii) 10
iii) 12
iv) 14.

b. The decimal equivalent of 10110_2 is

i) 15
ii) 18
iii) 22
iv) 24.

c. The binary equivalent of 109 is

i) 100110_2
ii) 1111001_2
iii) 1101101_2
iv) $1110101_{2.}$

d. The decimal equivalent of $25\ F7_{16}$ is

i) 1719
ii) 9610
iii) 9719
iv) 09919.

2. Analytical questions

a. Convert the following binary numbers to decimal equivalent.

i) 10101_2
ii) 100101_2
iii) 1011.101_2
iv) $101.1101_{2.}$

b. Convert the following decimal numbers to binary

i) 653.625
ii) 13.6875
iii) 367
iv) 235.07.

c. Convert the following hexadecimal numbers to decimal

i) $129A.B86_{16}$
ii) $73D5_{16}$
iii) 0.7825_{16}
iv) $39.C8_{16.}$

d. Convert the following hexadecimal numbers to binary

i) $129A.B86_{16}$
ii) $3D59_{16.}$

e. Convert the following binary numbers to hexadecimal

i) 10110100101110_2
ii) $101101101110.1000110_{2.}$

Lesson 3: Binary Arithmetic

3.1 Learning Objective

On completion of this lesson you will be able to :

- add two binary numbers
- multiply two or more binary numbers
- subtract one binary number from another
- do division of binary numbers.

3.2 Binary Addition

The execution of numerical calculations is essentially the same in all positional number systems. The addition of two binary numbers is accomplished according to the following three-step algorithm :

Binary Addition

Step 1. Add the first (rightmost) column.

Step 2. Record the unit digit of the column sum. If the sum exceeds one, carry the two's digit 1, to the next column.

Step 3. If there are additional columns or if there is a carry from Step 2, add the next column and repeat Step 2. Otherwise stop.

The addition table for the binary digits 0 and 1 appears as Table 1.7 the only additional facts needed for binary addition appear in Table 1.8.

+	0	1
0	0	1
1	1	10

Table 1.7 Binary Addition

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0, with a carry of 1 1 + 1 + 1 = 1, with a carry of 1

Table 1.8 Binary Addition Facts

Example 3.1 :

Evaluate the binary sum

```
  111     Addend
+101      Augend
```

by means of the three-step algorithm.

STEP 1: 1 + 1 = 0, with a carry of 1.
STEP 2:

1	Carries
111	Addend
+ 101	Augend
0	

STEP 3: 1+1 = 0, with a carry of 1.
STEP 2.

11	Carries
111	Addend
+101	Augend
00	

STEP 3: 1 + 1 + 1 = 1, with a carry of 1.
STEP 2.

111	Carries
111	Addend
+101	Augend
100	

STEP 3: 1 + 0 =1.
STEP 2.

111	Carries
111	Addend
+101	Augend
1100	Sum

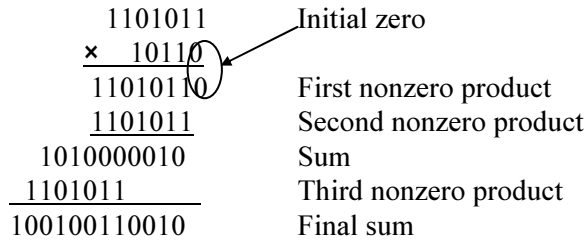
Step 3. Stop.

Example 3.4 :

To calculate the binary product 1101011 × 10110 multiply 1101011 by the digits 0, 1, 1, 0 and 1 as follows :

```
   1101011
×    10110
   0000000
  1101011
 1101011
0000000
1101011
```

Then add the five bottom rows of numbers. In actual practice, one does not write down any zero products. Finally bring down initial zero, if any and form a running total, adding one nonzero row after another :



The final sum is the required product. Here it is extremely important to line up the numbers in the correct columns.

3.3 Binary Subtraction

Subtraction in the binary system can be performed using the following two-step algorithm:

Binary Subtraction

Step 1. If the lower (subtrahend) digit is greater than the upper (minuend) digit, borrow from the next column to the left.

Step 2. Subtract the lower value from the upper value.

In Step 1 "borrowing" means appropriating, with no intention of paying back.

The only subtraction facts needed for binary subtraction are the four listed in Table 1.9.

The last entry comes from :

$$10 - 1 = 1$$

That is, the difference 0 - 1 requires borrowing, which then yields 10 -1 = 1.

Table 1.9 Binary subtraction facts

0 - 0 = 0
1 - 0 = 1
1 - 1 = 0
0 - 1 = 1, with a borrow of 1 from the next column

Example 3.5 :

To evaluate the binary difference 11101-1011, one could apply the subtraction facts in Table 1.5 and obtain.

$$\begin{array}{r} 11101 \\ -\ 1011 \\ \hline 10010 \end{array}$$

It is observed that 1 is borrowed from the third column because of the difference 0-1 in the second column.

As with decimal subtraction, binary subtraction becomes more complex when a borrow is needed from a digit which is 0. Again, a borrow is taken from the first nonzero digit to the left, but now each intervening 0 becomes 1 (as 10-1=1).

Example 3.6 :

Consider the difference

```
 11000
-10011
```

Obtain,

```
  011
 11000
- 10011
   101
```

Here a difference 0 - 1 occurs in the first column; hence it is required to borrow from the fourth column, where the first nonzero digit to the left appears, and the two intervening 0s become 1s.

Example 3.7 :

Calculate the difference 1100101001-110110110.

```
  00110  01
 1100101001
- 110110110
  101110011
```

3.4 Binary Division

Binary Division

Recall that the division of decimal numbers can be reduced to multiplying the divisor by individual digits of the dividend and subtraction.

Example 3.8 :

Calculate 42558 ÷ 123. Here 123 is the divisor. The algorithm for division yields :

```
          346
123)42558
    369
     565
     492
       738
       738
         0
```

That is, multiply 123 by 3 and subtract the product, from 425; then multiply 123 by 4 and subtract the product 492, from 565; lastly multiply 123 by 6 and subtract the product 738, from 738, to obtain a 0 remainder. [Because of the geometry of the scheme, what these steps actually accomplish is first to subtract 3×10^2 times the divisor from the dividend, then 4×10 times the divisor from what is left, and then 6 times the divisor from what is left. At that point the dividend is exhausted, showing that the dividend originally contained the divisor $3 \times 10^2 + 4 \times 10 + 6 = 346$ times].

The above algorithm also works for binary division. In fact, multiplying the divisor by the only nonzero digit, 1, does not change the number; hence the algorithm for division reduces to repeated subtraction of the divisor (times a power of 2).

Example 3.9 :

Evaluate 1010001 ÷ 11. This gives

```
        11011
11)1010001
  11
  100
   11
    100
     11
      11
      11
       0
```

Thus the quotient is 11011.

As in decimal division of integers, a remainder is possible when one binary is divided by another. Also, the division of binary fractions is handled the same way as the division of decimal fractions; that is, one converts the divisor to an integer by moving the binary point in both the divisor and the dividend the same the number of places.

Example 3.10 :

Evaluate 1110111 ÷ 1001. Applying the usual division algorithm,

```
        1101
1001)1110111
     1001
      1011
     1001
      1011
     1001
        10
```

The quotient is 1101, with a remainder 10.

3.5 Complements

Complements

Arithmetic complements require in two separate but related situations. First of all, complements come up in storing numbers in the computer. While human-beings use the signs **+** and - to denote positive and negative numbers, the computer can process data only in terms of bits. Although it is possible to reserve a bit to denote the sign of a number (say, 0 for + and 1 for -), many computers store negative numbers in the form of their arithmetic complements.

Complements also arise in the operation of subtraction. In fact, complements can be used to reduce subtraction to addition. This is especially useful as it avoids the possibility of repeated borrowing from one column to another.

There are two types of complements, the one's complement and two's complement, respectively.

A is a binary number, the one's complement of A is obtained by subtracting each digit of A from 1, and the two's complement of A is its one's complement plus 1.

Example 3.11 :

Binary number:	111100001111
One's complement:	000011110000
Two's complement:	000011110001

Observe that taking the ones complement simply inverts each digit, i.e. 0 is replaced by 1 and 1 is replaced by 0.

As in the decimal system, binary subtraction is performed by adding the radix-minus-one (one's) complement plus one or by adding the radix (two's) complement.

Example 3.12 :

Evaluate the difference Y = B - A, where A = 10001110 and B = 11110000.

(a) First by ordinary binary subtraction :

```
    011
 11110000     B
-10001110     A
 01100010     Y
```

Observe that the borrowing was propagated to the third digit to the left.

(b) The one's complement of A is 01110001. Add this to B and then add 1:

```
 11110000     B
+01110001     One's complement of A
①01100001
        1
 01100010
```

(This method is also given the name end-around carry).

(c) The two's complement of A is 01110010. Add this to B:

```
 11110000     B
+01110010     Two's complement of A
①01100010
```

Deleting the 1 (which would be an overflow in an 8-bit register) gives the difference Y.

3.6 Exercise

1. Multiple choice questions

a. Addition in the binary system can be performed using

i) 2 step algorithm
ii) 3 step algorithm
iii) 4 step algorithm
iv) 5 step algorithm.

b. Complement arise in the operation of

i) addition
ii) subtraction
iii) multiplication
iv) conversion.

2. Analytical questions

a. Describe the three step algorithm with example.

b. Describe the 2-step algorithm.

c. Add the following binary numbers

i) 10110_2 and 1100_2
ii)100101_2 and 10100_2.

d. Multiply the following numbers

i) 100011_2 and 101_2
ii)101_2 and 1011_2.

e. Evaluate the following

i) 10010 - 11011
ii) 10101 - 00110
iii)11101 - 1011.

f. Evaluate the following

i) 11010011 ÷ 11
ii) 11110111 ÷ 1001.

g. What do you understand by 1's complement and 2's complement method?

h. Find the 1's complement and 2's complement of the following numbers

i) 101101_2
ii) 111100001111_2
iii) 9090_{10}
iv) 1010101_2.

i. Perform following subtraction using 1's and 2's complement method

i)11011 - 10010
ii)10001110 - 11110000
iii) 10101 - 00110.

Lesson 4 : Data Representation and Codes

4.1 Learning Objective

On completion of this lesson you will be able to :

- describe different methods of coding
- know the definitions of data, information and code
- describe different methods of data representation in computers
- identify different representation of numbers in computers.

4.2 Data Information and Codes

'Data' are the names given to basic facts such as names and numbers. Examples are: unit price, quantity sold, times, dates, product, name, addresses, tax codes.

Data Information and Codes

Information is data which has been converted into a more useful form, i.e. processed facts. For example: total price = unit price × quantity sold. Here total price is information and unit price, quantity sold are data. Examples are: pay slips, receipts, reports.

'Codes' are used to reduce the volume of data. The recording of data can be made less laborious, less prone to error and the data will subsequently be more manageable and easier to manipulate if standard abbreviations or simplified representations are used. This technique is called data coding. Examples: Yes/No answers on forms can be represented by single Y's or N's. A person's sex may be indicated by M or F.

4.3 Data Representation

Here discussion will be on how numeric data are represented inside the computer using straight binary coding, which encodes an entire number as a whole. Straight binary coding requires that numbers be stored in computer locations as a fixed number of bits. A list of bits so treated as a unit is called a word, and the number of bits is called the length of the word. For definiteness, assume, unless otherwise stated, that computers have words of fixed length 32.

Integers Representation

Integers Representation

Integers or fixed-point numbers are numbers that have no decimal points. An integer J is represented in the memory of the computer by its binary form if J is positive, and by its 2's complement (i.e. the 2's complement of its absolute value) if J is negative.
Example 4.1 :

The computer stores $423 = 110100111_2$ in a 32-bit memory location by introducing sufficient 0s at the beginning of the binary form:

423

0	0	0	0	0	...	0	0	1	1	0	1	0	0	1	1	1

The computer stores -423 in a memory location by taking the 1's complement of the above representation for 423 and then adding 1:

-423

1	1	1	1	1	...	1	1	0	0	1	0	1	1	0	0	1

In the first display the dots represent omitted 0s; in the second omitted 1's.

The computer can tell whether an integer *J* in memory is positive or negative by looking at the left most bit. If the first bit is 0, then *J* is positive; if the left most bit is 1, then *J* is negative. Accordingly, the largest (positive) integer that can be stored in a 32-bit memory location

$$\underbrace{0\ 1\ 1\ 1\ 1\ \ldots\ 1\ 1\ 1\ 1\ 1}_{31\ \text{ones}}$$

Or 2^{31}-1, which is approximately 2 billion, Similarly, the smallest (negative) integer that can be stored in a 32-bit memory location is -2^{31}, or approximately -2 billion.

Binary Exponential Form

Binary numbers, like decimal numbers, can be written in exponential form, where powers of two are used instead of powers of ten. Thus, each nonzero binary number has a unique normalized exponential form in which the binary point appears before the first 1 bit. This unique form yields a unique mantissa M, and a unique integer *n* representing the exponent to two. Either of these numbers may be positive or negative, and the exponent *n* may also be zero.

Table 1.10 gives some binary numbers in normalized exponential form, each mantissa being written with exactly 5 bits.

Table 1.10

Binary Number	Normalized Exponential Form	Mantissa	Exponent
1010.1	0.10101×2^{4}	0.10101	4
0.001111	0.11110×2^{-2}	0.11110	-2
-111	-0.11100×2^{3}	-0.11100	3
0.1	0.10000×2^{0}	0.10000	0
- 0.01010101	-0.10101×2^{-1}	-0.10101	-1

Floating-Point Representation

Floating-Point Representation

Floating-point numbers (also called real number) have embedded decimal points. Such numbers are stored and processed in binary exponential forms. The memory location is divided into three fields, or blocks of bits. One field, the first bit, is reserved for sign of the number (usually 0 for + and 1 for -); a second field, for the exponent of the number; and last field, for the mantissa of the number. Figure 4.1 shows the usual fields of a 32-bit memory location. With a 24-bit mantissa field, the precision of the computer is 8 (significant decimal digits).

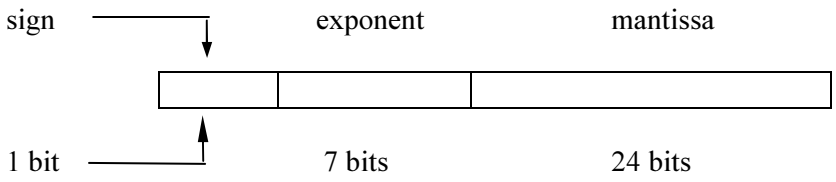


Figure 4.1

It remains to discuss the way the integer exponent, n, of a floating-point number is represented in its field. A few computers stores n as its binary form when n is positive or zero, and as its 2's complement when n is negative; i.e. the same way that fixed-point integers are stored in memory. However, most computers represent n by its characteristic, $n + 2^{t-1}$, where t is the number of bits in the exponent field. Table 1.11 shows the relationship between the true exponent n and its characteristic when t =7. Observe that a 7-bit exponent field can represent exponents from -64 to 63, which means that the computer can store floating-point numbers between 2^{-64} and 2^{63}.

Table 1.11

True Exponent	-64	-63	-62	-61	...	-1	0	1	...	63
Characteristic	0	1	2	3	...	63	64	65	...	127

Example 4.2 :

Given A = -419.8125. Converting A to binary form yields

$$A = -110100011.1101_2$$

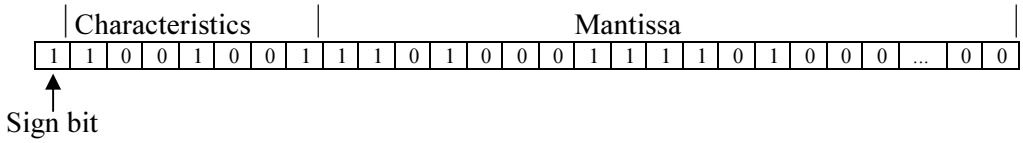
Hence the normalized exponential form of A is

$$A = -0.1101000111101 \times 2^9$$

The true exponent of A being 9, its 7-bit characteristic is

$$9 + 64 = 73 = 1001001_2$$

Thus A will be stored in the 32-bit memory location as follows.



Observe that (i) the first bit is 1, which indicates that A is negative; (ii) the first characteristic field is 1, which indicates that the exponent of A is nonnegative; and (iii) sufficiently many 0s are attached to the end of the mantissa of A to complete the 24-bit mantissa field.

4.4 BCD Code

BCD Code

There are many ways of representing numerical data in binary form. One way is simply to write the numbers to the base 2. This is called straight binary coding. Another way is to encode decimal digits. These code, which require 4 bits for each decimal digit, is called BCD (binary-coded decimal) code.

Table 1.12

Decimal Digits	BCD Code
	8-4-2-1
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

4-bit BCD words are shown in Table 1.12. The first one is a weighted code, in which the bits are given, from left to right, the weights 8, 4, 2,

and 1, respectively. Since these weights are just the place valves in the binary system, a decimal digit is encoded as its binary representation.

Example 4.3 :

The BCD representation of N = 469 is

4	6	9
0100	0110	1001

On the other hand, the straight binary representation of N is

$$N = 111010101_2$$

which involves 3 fewer bits.

4.5 Parity

For each character, the value of the check bit (0 or 1) is such as to make the sum of the bits, including the check bit, odd or even, according as the machine operates on odd or even parity.

Example 4.4 :

Parity

If the computer uses odd parity, the characters 7, 9 are stored as follows:

Character	6 bit code	Parity bit	Complete representation using odd parity
7	000111	0	0000111
9	001001	1	1001001

That is, the check bit for 7 is 0 because the sum of the odd bits in the 6-bit code for 7 is three, which is already odd. On the other hand, the check bit for 9 is 1 because the sum of the bits in the 6-bit code for 9 is two which is even.

The purpose of the check bit is to ensure that no bit is lost or gained when data are transmitted internally in a computer. After a character is transmitted, the computer adds up the bits in the character. If a single error occurs, the sum of the bits will not have the same parity as the parity of the computer. The computer would then retransmit the data. Clearly, the computer cannot use this type of checking to see if two errors occurred; but such an occurrence is very unlikely.

4.6 EBCDIC and ASCII

EBCDIC and ASCII

Modern data processing frequently requires more than the 28 special characters possible under any 6-bit BCD code. (Some data processing equipment may even want both lowercase and uppercase letters). Accordingly, various 8-bit codes have been developed. Each coded character, or byte, is normally divided into four zone bits and four 8-4-2-1 numeric bits, as shown;

	zone bits			numeric bits			
Z	Z	Z	Z	8	4	2	1

More generally, the word ‘byte’ is used to denote any group of eight bits. It is seen that a byte may be represented by two hexadecimal digits, the first corresponding to the zone bits and the second to the numeric bits. As with the 4- and 6-bit BCD codes, an extra, check bit is utilized in the computer.

There are two 8-bit BCD code predominant in the computer industry today. EBCDIC, pronounced ‘ebb-see-dick’ and is short for Extended Binary-Coded Decimal Interchange Code. This code developed by IBM, is used mainly by IBM-compatible computer systems (Table 1.13).

ASCII pronounced 'ass-key' and is short for American Standard Code for Information Interchange. This code was originally developed as a 7-bit standardization of various special codes, and was then extended to an 8-bit code. It is used mainly by non-IBM computer systems (Table 1.14).

Table 1.13 EBCDIC

Char.	Zone Numeric	Hex	Char.	Zone Numeric	Hex	Char.	Zone Numeric	Hex
			S	1110 0010	E2	black	0100 0000	40
A	1100 0001	C1	T	0011	E3	.	1011	4B
B	0010	C2	U	0100	E4	<	1100	4C
C	0011	C3	V	0101	E5	(	1101	4D
D	0100	C4	W	0110	E6	+	0100 1110	4E
E	0101	C5	X	0111	E7	&	0101 0000	50
F	0110	C6	Y	1000	E8	$	1011	5B
G	0111	C7	Z	1110 1001	E9	*	1100	5C
H	1000	C8				)	1101	5D
I	1100 1001	C9	Char.	Zone Numeric	Hex	;	0101 1110	5E
J	1101 0001	D1				-	0110 0000	60
K	0010	D2	0	1111 0000	F0	/	0001	61
L	0011	D3	1	0001	F1	,	1011	6B
M	0100	D4	2	0010	F2	%	1100	6C
N	0101	D5	3	0011	F3	>	1110	6E
O	0110	D6	4	0100	F4	?	0110 1111	6F
P	0111	D7	5	0101	F5	:	0111 1010	7A
Q	1000	D8	6	0110	F6	#	1011	7B
R	1101 1001	D9	7	0111	F7	@	1100	7C
			8	1000	F8	=	0111 1110	7E
			9	1111 1001	F9			

Table 1.14

Char.	Zone Numeric	Hex	Char.	Zone Numeric	Hex	Char.	Zone Numeric	Hex
0	0101 0000	50	A	1010 0001	A1	P	1011 0000	B0
1	0001	51	B	0010	A2	Q	0001	B1
2	0010	52	C	0011	A3	R	0010	B2
3	0011	53	D	0100	A4	S	0011	B3
4	0100	54	E	0101	A5	T	0100	B4
5	0101	55	F	0110	A6	U	0101	B5
6	0110	56	G	0111	A7	V	0110	B6
7	0111	57	H	1000	A8	W	0111	B7
8	1000	58	I	1001	A9	X	1000	B8
9	0101 1001	59	J	1010	AA	Y	1001	B9
			K	1011	AB	Z	1011 1011	BA
			L	1100	AC			
			M	1101	AD			
			N	1110	AE			
			O	1010 1111	AF			

In both systems a digit has its binary representation as the numeric portion of its code. For the zone portion, EBCDIC uses 1111 and ASCII uses 0101.

4.7 Exercise

1. Multiple choice questions

a. Codes are used to reduce the volume of

i) information ii) data
iii) numbers iv) files.

b. Integers numbers are numbers that have

i) decimal points ii) no decimal points
iii) embedded decimal points iv) no embedded decimal points.

c. BCD codes require

i) 4 bits for each decimal digit
ii) 7 bits for each decimal digit
iii) 8 bits for each decimal digit
iv) 10 bits for each decimal digit.

2. Analytical questions

a. What do you mean by data, information and codes?
b. Describe different representation of numbers in computer.
c. Write the BCD representation of the following numbers
a) 469 and b) 4793.
d. What do you understand by BCD, ASCII and EBCDIC codes?

Lesson 5 : Logic Functions

5.1 Learning Objective

On completion of this lesson you will be able to :

- know the elementary concept of Boolean algebra
- describe the primary logic gates like AND, OR, NOT
- draw truth table of output variable for a combination of input variables
- explain the operation of secondary gates like NAND, NOR and EXOR.

5.2 Introduction

Data and control instructions move inside a computer by means of pulses of electricity. Certain components of computers combine these pulses as if they were following a set of rules. The components are the logic elements. Computer logic is the combination of inputs and outputs produced by logic elements.

Pulses of electricity are called digital signals. A digital signal has two discrete levels or values. The two discrete signal levels HIGH and LOW can also be represented by binary digits 1 and 0 respectively. A binary digit (0 or 1) is referred to as a bit. Since a digital signal can have only one of the two possible levels 1 or 0, the binary number system can be used for the analysis and design of digital systems. The two levels (or states) can also be designated as on and off (or TRUE and FALSE). George Boole introduced the concept of binary number system in the studies of this mathematical theory of LOGIC in the Laws of Thought in 1854 and developed its algebra known as Boolean Algebra.

5.3 : Primary Logic Gates

Primary Logic Gates

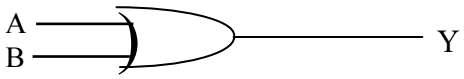
The common use of logic elements is to act as switches, although they have no moving parts. They open to pass on a pulse of electricity or close to shut it off. This is why they are known as gates. The primary gates are OR, AND, NOT.

OR Gate

OR Gate

An OR gate has an output 1 if any of its inputs are 1. The diagram and truth table for two input OR gate are shown in Fig 5.1. Ideal output, Y = A + B, where + denotes OR operation.

(a)



Truth Table

(b)

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Figure 5.1: Two-input OR gate (a) symbol (b) truth table.

Figure 5.2 Illustrates, the close relationship between 2-input OR gate and electrical switching circuits. Such a circuit normally contains some source of energy (a battery), an output device (a lamp), and one or more switches - all connected by wires. A switch is a two-state device that is either closed (on) or open (off). In Figure 5.2 switch A and B, are connected in parallel. The lamp will light if switch A is closed, or if switch B is closed, or if both switches are closed. But this is the property described by the truth table for the OR gate, where 1 denotes that the switches or lamp is on and 0 indicates that it is off.

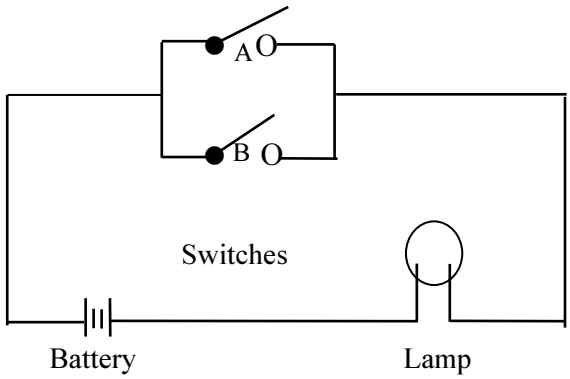


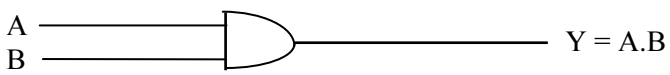
Figure 5.2: Parallel circuit.

AND Gate

AND Gate

An AND gate has an output 1 if all of its inputs are 1. The diagram and truth table for a two input AND gate are shown in Figure 5.3. Here output, Y= A.B, where '.' denotes AND operation.

(a)



(b) Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Figure 5.3: Two-input AND gate (a) symbol, (b) truth table.

Figure 5.4 is a circuit showing two switches, A and B, connected in series. The lamp will light only when both A and B are closed. This is exactly the property described by the truth table for the AND gate, here again 1 denotes that the circuit element is on and 0 denotes that it is off.

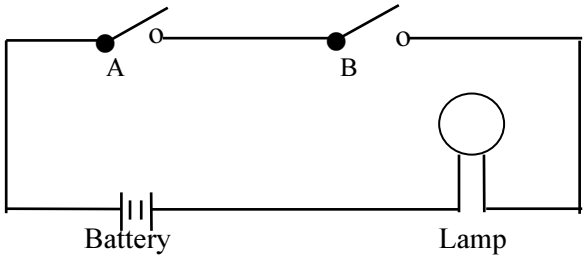


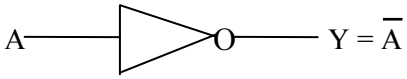
Figure 5.4: Series circuit

NOT Gate

NOT Gate

A NOT gate has one input and one output. It has the effect of reversing the input signal and is sometimes called an inverter. The diagram and truth table for a NOT gate are shown in Fig 5.5. Here output Y= $\overline{A}$ where '-' indicates NOT operation.

(a)



Truth Table

(b)

A	Y
0	1
1	0

Figure 5.5: NOT gate (a) symbol, (b) truth table

5.4 Secondary Logic Gates

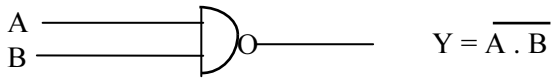
Secondary Logic Gates

Some secondary gates are NAND, NOR, EXOR etc. NAND and NOR gates are called universal gates because any one them can be used to realize of any logic expression.

NAND Gate

NAND Gate

A NAND gate has the same effect as an AND gate followed by a NOT gate. Hence the output will be opposite of the AND gate. The diagram and truth table for a two-input NAND gate are shown in Figure 5.6.

(a) 

(b) Truth Table

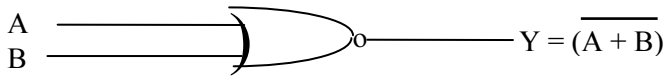
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Figure 5.6: NAND gate (a) symbol (b) truth table

NOR Gate

NOR Gate

A NOR gate has the same effect as OR gate followed by a NOT gate. Hence the output will be the opposite of OR gate. The diagram and truth table for a two-input NOR gate are shown in Figure 5.7.

(a) 

(b) Truth Table

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

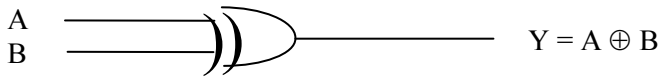
Figure 5.7: NOR gate (a) symbol, (b) truth table

EXOR Gate

EXOR Gate

It is widely used in digital circuits. EXOR is not a primary or basic gate. Diagram and truth table for a two input EXOR gate are shown in Figure 5.8. Here output $Y = A \oplus B$, where $\oplus$ denotes EXOR operation.

(a)



(b) Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Figure 5.8: EXOR gate (a) symbol, (b) truth table

Basic gates (AND, OR, NOT) and universal gates (NAND, NOR) can be used in combination to make up digital computer circuits.

5.5 Exercise

1. Multiple choice questions

a. Which of the following are the primary gates?

i) OR, AND, NAND ii) NOR, AND, NOT
iii) OR, AND, NOT iv) NOR, NAND, EXOR.

b. What is pulses of electricity?

i) analog signals ii) digital signals
iii) voltage iv) current.

c. An OR gate has an output 1 if

i) all of its inputs are 1 ii) any of its inputs are 1
iii) any of its inputs are 0 iv) all of its inputs are 0.

B. Analytical questions

a. What do you know about Boolean algebra?
b. Draw the diagram and truth table of two-input OR gate.
c. Explain the operation of the following gates
AND, NOT, NOR, XOR.
d. What do you understand by basic gate and universal gate?