Data Representation

chapter 2

In This Chapter

2.1 Introduction

In science, technology, business, and, in fact, most other fields of endeavour, we are constantly dealing with quantities; so are computers. Quantities are measured, monitored, recorded, manipulated arithmetically, observed, or in some other way utilized in most physical systems. In digital systems like computers, the quantities are represented by symbols called digits. Many number systems are in use in digital technology that represent the digits in various forms. The most common are the decimal, binary, octal, and hexadecimal systems. This chapter discusses these number systems and the physical representation of digits in computers.

2.2 Digital Number Systems

In digital representation, various number systems are used. The most common number systems used are *decimal*, *binary*, *octal*, and *hexadecimal* systems. Let us discuss these number systems.

2.2.1 Decimal (Denary) Number System (Base 10)

The *decimal system* (also called **Denary number system**) is composed of 10 numerals or symbols (*Deca* means 10, that is why this system is called *decimal system*). These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ; using these symbols as *digits* of a number, we can express any quantity. The decimal system, also called the *base-10* system because it has 10 digits, has evolved naturally as a result of the fact that man has 10 fingers.

The decimal system is a *positional-value* system in which the value of a digit depends on its position. For example, consider the decimal number 729. We know that the digit 7 actually represents 7 *hundreds*, the 2 represents 2 *tens*, and the 9 represents 9 *units*. In essence, the 7 carries the most weight of three digits; it is referred to as the *most significant digit* (MSD). The 9 carries the least weight and is called the *least significant digit* (LSD).

Consider another example, 25.12. This number is actually equal to (2 *tens* plus 5 *units* plus 1 *tenths* plus 2 *hundredths*) *i.e.*, $2\times10+5\times1+1\times\frac{1}{10}+2\times\frac{1}{100}$. The decimal point is used to separate the integer and fractional parts of the number.

More rigorously, the various positions relative to the decimal point carry weights that can be expressed as powers of 10. This is illustrated in Fig. 2.1 where the number 2512.1971 is represented. The decimal point separates the positive powers of 10 from the negative powers.

The number 2512.1971 is thus equal to

$$2\times10^{3}+5\times10^{2}+1\times10^{1}+2\times10^{0}+1\times10^{-1}+9\times10^{-2}+7\times10^{-3}+1\times10^{-4}$$

NOTE

Base of a number system is also called **radix**.

In general, any number is simply the sum of the products of each digit value and its positional value.

The squence of decimal numbers goes as 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21....

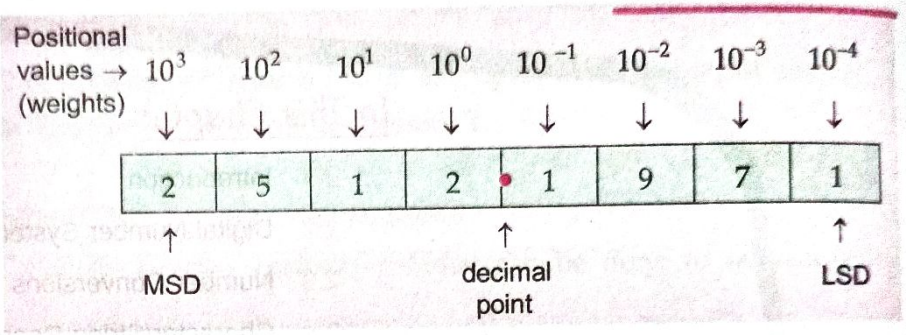


Figure 2.1 Positional values in decimal numbers.

2.2.2 Binary Number System (Base 2)

As electronic circuits operate with only two voltage levels, for this reason, almost every digital system uses the binary number system (base 2) as the basic number system of its operations, although other systems are often used in conjunction with binary.

In the *binary system* there are only two symbols or possible digit values, 0 and 1. Even so, this base-2 system can be used to represent any quantity that can be represented in decimal or other number systems.

The binary system is also a positional-value system, wherein each binary digit has its own value or weight expressed as a power of 2. This is illustrated in Fig. 2.2.

Here, places to the left of the binary point (counterpart of the decimal point) are positive powers of 2 and places to the right are negative powers of 2. The number 1010.0101 is shown represented in the figure.

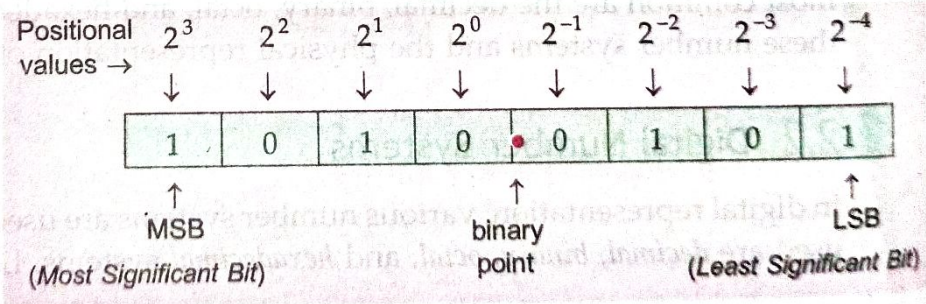


Figure 2.2 Positional values in binary numbers.

To find the decimal equivalent of above shown binary number, we simply take the sum of the products of each digit value (0 or 1) and its positional value :

$$1010.0101_{2}=(1\times2^{3})+(0\times2^{2})+(1\times2^{1})+(0\times2^{0})+(0\times2^{-1})+(1\times2^{-2})+(0\times2^{-3})+(1\times2^{-4})$$
$$=8+0+2+0+0+0.25+0+0.0625=10.3125_{10}$$

Notice in the preceding operation that subscripts (2 and 10) were used to indicate the base in which the particular number is expressed.

The leftmost bit carries the largest weight and hence, is called the **most significant bit (MSB).** The rightmost bit carries the smallest weight, and hence called **least significant bit (LSB).**

The sequence of binary numbers goes as 00, 01, 10, 11, 100, 101, 110, 111, 1000, - - - - - -.

2.2.3 Octal Number System (Base 8)

The octal number system is very important in digital computer work. The octal number system has a base of *eight*, meaning that it has eight unique symbols : 0, 1, 2, 3, 4, 5, 6, and 7. Thus, each digit of an octal number can have any value from 0 to 7.

The octal system is also a positional value system, wherein each octal digit has its own value or weight expressed as a power of 8 (see Fig. 2.3). The places to the left of the octal point (counter-part of decimal point and binary point) are positive powers of 8 and places to the right are negative powers of 8. The number 3721.2406 is shown represented in the figure.

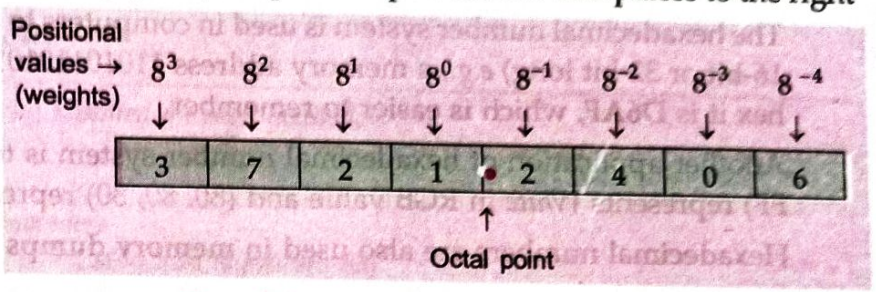


Figure 2.3 Positional values in octal numbers.

To find the decimal equivalent of above shown octal number, simply take the sum of products of each digit value and its positional value :

$$3721.2406_8 = (3\times 8^3)+(7\times 8^2)+(2\times 8^1)+(1\times 8^0)+(2\times 8^{-1})+(4\times 8^{-2})+(0\times 2^{-3})+(6\times 8^{-4})$$
$$= 3\times 512+7\times 64+2\times 8+1\times 1+2\times 0.125+4\times 0.015625+0+6\times 0.000244$$
$$= 1536+448+16+1+0.25+0.0625+0+0.001464 = 2001.313964_{10}$$

The sequence of octal numbers goes as 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22..... See each successive number after 7 is a combination of two or more unique symbols of octal system.

2.2.4 Hexadecimal Number System (Base 16)

The hexadecimal system uses base 16. Thus, it has 16 possible digit symbols. It uses the digits 0 through 9 plus the letters A, B, C, D, E, and F as the 16 digit symbols.

Just like above discussed systems, hexadecimal system is also a positional-value system, wherein each hexadecimal digit has its own value or weight expressed as a power of 16. (see Fig. 2.4). The digit positions in a hexadecimal number have weights as shown in Fig. 2.4.

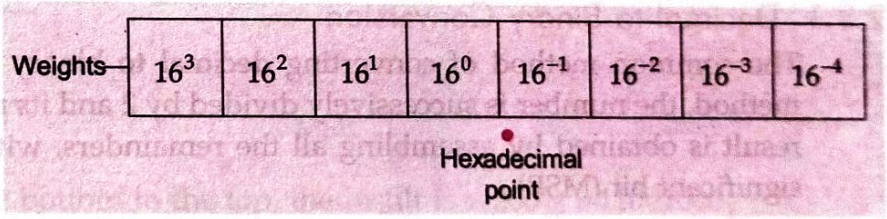


Figure 2.4 Positional values in hexadecimal numbers.

Following table 2.1 shows the relationships between hexadecimal, octal, decimal and binary.

Table 2.1 *Relationship between Various Number Systems*

Hexadecimal	Octal	Decimal	Binary
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	10	8	1000
9	11	9	1001
A	12	10	1010
B	13	11	1011
C	14	12	1100
D	15	13	1101
E	16	14	1110
F	17	15	1111

Note that each hexadecimal digit represents a group of four binary digits. It is important to remember that *hex* (abbreviation for hexadecimal) digits A through F are equivalent to the decimal values 10 through 15.

NOTE

Hexadecimal number systems are used on computers as they are better human-readable than binary numbers and they are more compact than binary, *e.g.*, 2 hex digits can represent range 0..255 whereas 8 binary digits will be required for the same.

Applications of Hexadecimal Number System

The hexadecimal number system is used in computers **to specify memory addresses** (which are 16-bit or 32-bit long) *e.g.*, a memory address 110101101010111 is a big binary address but with hex it is **D6AF**, which is easier to remember.

Another application of hexadecimal number system is to **represent colour codes** *e.g.*, (FF, FF, FF) represents *White* in RGB value and (80, 80, 80) represents *Grey* in RGB value.

Hexadecimal numbers are also used in **memory dumps** and **debugging codes.**

2.3 Number Conversions

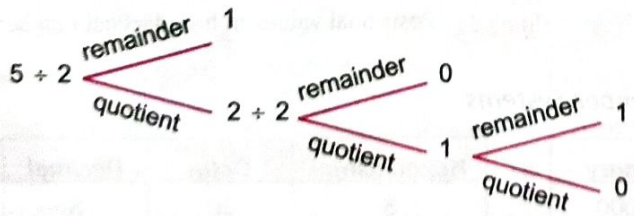
The binary number system is the most important one in digital systems as it is very easy to implement in circuitry. The decimal system is important because it is universally used to represent quantities outside a digital system.

In addition to binary and decimal, octal and hexadecimal number systems find widespread application in digital systems. These number systems (octal and hexadecimal) provide an efficient means for representing large binary numbers. As we shall see, both these number systems have the advantage that they can be easily converted to and from binary.

In a digital system, three or four of these number systems may be in use at the same time, so that an understanding of the system operation requires the ability to convert from one number system to another. This section discusses how to perform these conversions. So, let us discuss them one by one.

2.3.1 Decimal-to-Binary Conversion

The common method of converting decimal to binary is ***repeated-division*** method. In this method, the number is successively divided by 2 and its remainders recorded. The final binary result is obtained by assembling all the remainders, with the last remainder being the most significant bit (MSB).



Keep dividing quotient until you get 0.

It can be written as :

2	5	*Remainders*
2	2	1
2	1	0
	0	1

Then write the remainders in last-to-first order. It means 5 of decimal number system is equal to 101 in binary system $5_{10} = 101_2$. Consider another example

2	11	*Remainders*
2	5	1
2	2	1
2	1	0
	0	1

$\Rightarrow \quad 11_{10} = 1011_2$

Let us consider some examples now.

EXAMPLE 1 *Convert 43_{10} to binary using repeated division method.*

SOLUTION *Repeated Division*

2	43	*Remainders*
2	21	1 LSB (Least Significant Bit)
2	10	1
2	5	0 Write in this order
2	2	1
2	1	0
	0	1 MSB (Most Significant Bit)

Reading the remainders from the bottom to the top,

$43_{10} = 101011_2$

EXAMPLE 2 *Convert 200_{10} to binary using repeated division method.*

SOLUTION

2	200	*Remainders*
2	100	0 LSB
2	50	0
2	25	0
2	12	1
2	6	0
2	3	0
2	1	1
	0	1 MSB

Reading the remainders from the bottom to the top, the result is : $200_{10} = 11001000_2$

Converting Fraction from Decimal to Binary

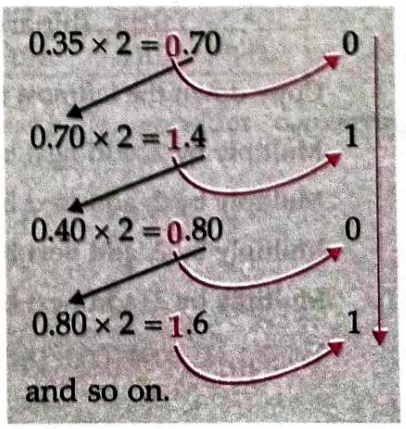
For a fractional decimal number, you can convert it to binary as :

abc. xyz (where *a, b, c, x, y, z* are digits)

abc as *repeated division* covered above and *.xyz* using *repeated multiplication* as illustrated below.

For instance, if you have 5.35, then convert **5** using *repeated division* as covered earlier and **0.35** as given in adjacent box.

Keep multiplying the fractional part with **2** and take out the non-fractional part : (*repeated multiplication*)



Consider the following example.

EXAMPLE 3 Convert 4.8125 decimal number into bin.

SOLUTION 4 – non fractional part using repeated division

0.8125 – fractional part using repeated multiplication.

		Remainders
2	4	
2	2	0
2	1	0
	0	1

$0.8125 \times 2 = 1.625$	1
$0.625 \times 2 = 1.250$	1
$0.250 \times 2 = 0.5$	0
$0.5 \times 2 = 1.0$	1

Thus $4.8125_{10} = 100.1101_2$

2.3.2 Binary-to-Decimal Conversion

The binary number system is a positional system where each binary digit (bit) carries a certain weight based on its position relative to the LSB. Any binary number can be converted to its decimal equivalent simply by summing together the weights of the various positions in the binary number which contain a 1. To illustrate, consider a binary number 11011_2

1	1	0	1	1	(*binary*)

Add positional weights for all 15 → $2^4 + 2^3 + 0 + 2^1 + 2^0 = 16 + 8 + 2 + 1 = 27_{10}$ (*decimal*)

Let's try another example with a greater number of bits *i.e.*, 10110101_2

1	0	1	1	0	1	0	1

Adding positional weights of all 1's → $2^7 + 0 + 2^5 + 2^4 + 0 + 2^2 + 0 + 2^0 = 181_{10}$

Note that the procedure is to find the weights (*i.e.*, powers of 2) for each bit position that contains a 1, and then to add them up. Also note that the *MSB has a weight of 2^7 even though it is the eighth bit; this is because the LSB is the first bit and has a weight of 2^0.*

The above method will always provide the correct decimal representation of a binary number. There is a second method, called the *dibble-dobble* method, that will also provide the solution. To use this method, start with the left-hand bit. Multiply this value by 2, and add the next bit to the right. Multiply the result value by 2, and add the next bit to the right. Stop when the binary point is reached. To illustrate to through the following two examples,

11011_2 (binary)

Copy down the leftmost bit :	1
Multiply by 2, add next bit	$(2 \times 1) + 1 = 3$
Multiply by 2, add next bit	$(2 \times 3) + 0 = 6$
Multiply by 2, add next bit	$(2 \times 6) + 1 = 13$
Multiply by 2, add next bit	$(2 \times 13) + 1 = 27$

$\therefore \quad 11011_2 = 27_{10}$

110100_2

The leftmost bit :	1
Multiply by 2, add next bit	$(2 \times 1) + 1 = 3$
Multiply by 2, add next bit	$(2 \times 3) + 0 = 6$
Multiply by 2, add next bit	$(2 \times 6) + 1 = 13$
Multiply by 2, add next bit	$(2 \times 13) + 0 = 26$
Multiply by 2, add next bit	$(2 \times 26) + 0 = 52$

$\therefore \quad 110100_2 = 52_{10}$

Converting Fractional Part

A fractional binary number say *pq.rst* (where *p, q, r, s, t* are bits) converted as :

$$p\times 2^1 + q\times 2^0 + r\times 2^{-1} + s\times 2^{-2} + t\times 2^{-3} \ .....$$

e.g., 100.1101 will be converted as

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

$$= 4 + 0 + 0 + \tfrac{1}{2} + \tfrac{1}{4} + 0 + \tfrac{1}{16} = 4 + 0.5 + 0.25 + 0.0625 = 4.8125$$

$\therefore \quad 100.1101_2 = 4.8125_{10}$

2.3.3 Decimal-to-Octal Conversion

A decimal integer can be converted to octal by using the same repeated-division method that we used in the decimal-to-binary conversion, but with a division factor of 8 instead of 2. An example is shown below :

8	266	*Remainders*
8	33	2 ↑
8	4	1
	0	4

Note that the first remainder becomes the least significant digit (LSD) of the octal number, and the last remainder becomes the most significant digit (MSD).

Reading up, $266_{10} = 412_8$

To convert to decimal system, every number system follows this rule :
Multiply each digit with base positional weight *i.e.*, for a number **ijk.lmn**
Binary to decimal $= i\times 2^2 + j\times 2^1 + k\times 2^0 + l\times 2^{-1} + m\times 2^{-2} + n\times 2^3$
Octal to decimal $= i\times 8^2 + j\times 8^1 + k\times 8^0 + l\times 8^{-1} + m\times 8^{-2} + n\times 8^{-3}$ .....
Hexadecimal to decimal
$= i\times 16^2 + j\times 16^1 + k\times 16^0 + l\times 16^{-1} + m\times 16^{-2} + m\times 16^{-3}$ .....

2.3.4 Octal-to-Decimal Conversion

An octal number, then, can be easily converted to its decimal equivalent by multiplying each octal digit by its positional weight.

For example, $372_8 = 3\times(8^2) + 7\times(8^1) + 2\times(8^0) = 3\times 64 + 7\times 8 + 2\times 1 = 250_{10}$

Another example : $24.6_8 = 2\times(8^1) + 4\times(8^0) + 6\times(8^{-1}) = 20.75_{10}$

2.3.5 Octal-to-Binary Conversion

The primary advantage of the octal number system is the ease with which conversion can be made between binary and octal numbers. The conversion from octal to binary is performed by converting *each* octal digit to its 3-bit binary equivalent. The eight possible digits are converted as indicated in Table 2.2.

Table 2.2 *Binary Equivalents of Octal Digits*

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

Using these conversions, any octal number is converted to binary by individually converting each digit. For example, we can convert 472_8 to binary using 3 bits for each octal digit as follows :

4	7	2
↓	↓	↓
100	111	010

Hence, octal 472 is equivalent to binary 100111010.

As another example, consider converting 5431 to binary :

5	4	3	1
↓	↓	↓	↓
101	100	011	001

Thus, $5431_8 = 101100011001_2$.

The same process applies equally on fractions. For example,

$$3.1_8 = \begin{matrix} 3 & . & 1 \\ \downarrow & & \downarrow \\ 011 & . & 001 \end{matrix}$$

$$3.1_8 = 011.001_2$$

NOTE

The octal to binary conversion takes place using 3-bit substitution for each octal character.

Because $2^3 = 8$, thus an octal number takes 3-bits to represent itself.

2.3.6 Binary-to-Octal Conversion

Converting from binary integers to octal integers is simply the reverse of the foregoing process. The bits of the binary integer are grouped into groups of *three* bits starting at the LSB. Then each group is converted to its octal equivalent (Table 2.2). To illustrate, consider the conversion of 100111010_2 to octal.

$$\underbrace{1\ 0\ 0}_{\downarrow\ 4}\quad \underbrace{1\ 1\ 1}_{\downarrow\ 7}\quad \underbrace{0\ 1\ 0}_{\downarrow\ 2_8}$$

Sometimes the binary number will not have even groups of 3 bits. For those cases, we can add one or two 0s to the left of MSB of the binary number to fill out the last group. This is illustrated below for the binary number 11010110.

0 added here to make it group of 3 bits

$$\underbrace{0\ 1\ 1}_{\downarrow\ 3}\quad \underbrace{0\ 1\ 0}_{\downarrow\ 2}\quad \underbrace{1\ 1\ 0}_{\downarrow\ 6_8}$$

Note that a 0 was placed to the left of the MSB in order to produce complete groups of 3.

The same process applies on fractions. But after the binary point, zeros are added to the right. For example

$$10110.0101_2 = \underbrace{0\ 1\ 0}_{2}\ \underbrace{1\ 1\ 0}_{6}\ .\ \underbrace{0\ 1\ 0}_{2}\ \underbrace{1\ 0\ 0}_{4}$$

2 zeros added here to make it a group of 3 bits

$$10110.0101_2 = 26.24_8$$

Note that, after the binary point, the groups of 3 bits are made starting from left-to-right. That is why, we added two zeros to make a group of three bits as the last group had only 1.

2.3.7 Decimal-to Hex Conversion

Recall that we did decimal-to-binary conversion using repeated division by 2, and decimal-to-octal using repeated division by 8. Likewise, decimal-to-hex conversion can be done using repeated division by 16. For example,

To convert 423_{10} to hex,

		Remainders
16	423	
16	26	7
16	1	A
	0	1

$(10_{10} = A_{16})$

Reading up, $423_{10} = 1A7_{16}$

Similarly, to convert 214_{10} to hex,

		Remainders
16	214	
16	13	6
	0	D

$(13_{10} = D_{16})$

Reading up, $214_{10} = D6_{16}$

Note that any remainders that are greater than 9 are represented by letters A through F.

2.3.8 Hex-to-Decimal Conversion

A hex number can be converted to its decimal equivalent by using the fact that each hex digit position has a weight that is a power of 16. The LSD has a weight of $16^0 = 1$, the next higher digit has a weight of $16^1 = 16$, the next higher digit has a weight of $16^2 = 256$, and so on. The conversion process is demonstrated in the examples below :

$$356_{16} = 3\times16^2 + 5\times16^1 + 6\times16^0 = 768 + 80 + 6 = 854_{10}$$

$$2AF_{16} = 2\times16^2 + 10\times16^1 + 15\times16^0 = 512 + 160 + 15 = 687_{10}$$

Note that in the second example the value 10 was substituted for A and the value 16 for F in the conversion to decimal. To convert a fractional number,

$$56.08_{16} = 5\times16^1 + 6\times16^0 + 0\times16^{-1} + 8\times16^{-2}$$
$$= 80 + 6 + 0 + 8/256 = 86 + 0.03125 = 86.03125_{10}$$

2.3.9 Binary-to-Hex Conversion

Binary numbers can be easily converted to hexadecimal by grouping in groups of four starting at the binary point.

EXAMPLE 4 *Convert 1010111010_2 to hexadecimal.*

SOLUTION

Group in fours 0010,1011,1010

Convert each number 2 B A

Thus, the solution is 2BA.

EXAMPLE 5 *Convert 10101110.010111_2 to hex.*

SOLUTION Groups in fours (inserting zeros before MSB or in the end, wherever required)

1010, 1110 . 0101, 1100

A E . 5 C

$10101110.010111_2 = AE.5C_{16}$

2.3.10 Hex to Binary Conversion

Like the octal number system, the hexadecimal number system is used primarily as a "shorthand" method for representing binary numbers. It is a relatively simple matter to convert a hex number to binary. *Each* hex digit is converted to its 4-bit binary equivalent (Table 2.1). This is illustrated below for $9F2_{16}$.

$9F2_{16}$ = 9 F 2

↓ ↓ ↓

1001 1111 0010

$= 100111110010_2$

Similarly, $3A6_{16}$ = 3 A 6

↓ ↓ ↓

0011 1010 0110

$= 001110100110_2$

Consider another example

$3BF.5C_{16}$ = 3 B F 5 C

↓ ↓ ↓ ↓ ↓

0011 1011 1111 0101 1100

$= 001110111111.01011100_2$

NOTE

Hex to binary conversion takes place using **4-bit substitution** for each hex character. Because $2^4 = 16$, thus a hexadecimal number takes 4 bits to represent itself.

Converting from Any Base to Any OTHER Base

As demonstrated in earlier examples and the table below, there is a direct correspondence between the number systems : with three binary digits corresponding to one octal digit ; four

binary digits corresponding to one hexadecimal digit, which you can use to convert from one number system to another.

NOTE

The hexadecimal and octal codes are used as shorthand means of expressing large binary numbers.

Table 2.3 *Correspondence of Binary and Octal Number Systems*

BIN	OCT	DEC
000	0	0
001	1	1
010	2	2
011	3	3
100	4	4
101	5	5
110	6	6
111	7	7

For conversion from ***base 2*** to ***base 8,*** we **use groups of three bits** and vice-versa.

Table 2.4 *Correspondence of Binary, Octal and Hexadecimal Number Systems*

BIN	HEX	OCT	DEC
0000	0	00	0
0001	1	01	1
0010	2	02	2
0011	3	03	3
0100	4	04	4
0101	5	05	5
0110	6	06	6
0111	7	07	7
1000	8	10	8
1001	9	11	9
1010	A	12	10
1011	B	13	11
1100	C	14	12
1101	D	15	13
1110	E	16	14
1111	F	17	15

For conversion from ***base 2*** to ***base 16,*** we **use groups of four bits** and vice-versa.

Let us consider some more examples regarding the same, *i.e.,* converting from any number system to another.

EXAMPLE 6 *Convert* $1948.B6_{16}$ *to Binary and Octal equivalents.*

SOLUTION

Hexadecimal	1	9	4	8	.	B	6
Binary	0001	1001	0100	1000	.	1011	0110

(By converting each individual Hex digit to equivalent 4 digit binary from above Table 2.4)

$\therefore \quad 1948.B6_{16} = 0001100101001000.10110110_2$

New Octal number can be generated from above Binary equivalent *i.e.,* as follows :

Binary	0	001	100	101	001	000	.	101	101	100
Octal		1	4	5	1	0	.	5	5	4

(By creating groups of 3 binary digits and converting them into equivalent octal no.)

$\therefore \quad 1948.B6_{16} = 14510.554_8.$

EXAMPLE 7 *Convert* 75643.5704_8 *to hexadecimal and binary numbers.*

SOLUTION We shall convert it in following way :

(*i*) From octal to binary — by representing each octal digit to 3 digit binary number.

(*ii*) From the complete binary number, we shall create groups of 4 binary digits around the decimal point.

(*iii*) Convert each 4-digit-binary group to equivalent hex digit.

Octal	7	5	6	4	3	.	5	7	0	4
Binary	111	101	110	100	011	.	101	111	000	100

$\therefore \quad 75643.5704_8 = 111101110100011.101111000100_2$

Binary	0111	1011	1010	0011	.	1011	1100	0100
Hexadecimal	7	B	A	3	.	B	C	4

$\therefore \quad 75643.5704_8 = 7BA3.BC4_{16}$

After learning about various digital number systems, let us talk about binary addition.

2.4 Character/String Representation

In addition to numerical data, a computer must be able to handle letters and other symbols too. To represent letters and symbols, a computer uses encoding schemes. An encoding scheme is a predetermined set of codes for each recognized latter, number and symbol.

Thus the role of an encoding scheme is very crucial in a computer system. Most popular encoding schemes are ASCII, Unicode, ISCII etc.

2.4.1 ASCII Code

The most widely used alphanumeric encoding scheme, the *American Standard Code for Information Interchange* (ASCII), was used in most microcomputers and minicomputers, and in many mainframes. The ASCII code (pronounced "askee") is a 7-bit code, and so it has $2^7 = 128$ possible code groups. ASCII was later converted to 8-bit Extended ASCII to represent more characters (256 characters).

Table 2.5 shows a partial listing of the ASCII code. In addition to the binary code group for each character, the table gives the octal and hexadecimal equivalents.

Check Point 2.1

1. What are the bases of decimal, octal, binary and hexadecimal systems ?
2. What is the common property of decimal, octal, binary and hexadecimal number systems ?
3. Complete the sequence of following binary numbers : 100, 101, 110, _____ , _____ , _____ .
4. Complete the sequence of following octal numbers : 525, 526, 527, _____ , _____ , _____ .
5. Complete the sequence of following hexadecimal numbers : 17, 18, 19, _____ , _____ , _____ .
6. Convert the following binary numbers to decimal and hexadecimal :
   (*a*) 1010 (*b*) 111010 (*c*) 101011111
   (*d*) 1100 (*e*) 10010101 (*f*) 11011100
7. Convert the following decimal numbers to binary and octal : (*a*) 23 (*b*) 100
   (*c*) 145 (*d*) 19 (*e*) 121 (*f*) 161
8. Convert the following hexadecimal numbers to binary : (*a*) A6 (*b*) A07
   (*c*) 7AB4 (*d*) BE (*e*) BC9 (*f*) 9BC8
9. Convert the following binary numbers to hexadecimal and octal :
   (*a*) 10011011101 (*b*) 1111011101011011
   (*c*) 11010111010111 (*d*) 1010110110111
   (*e*) 10110111011011
   (*f*) 1111101110101111

Table 2.5 *Partial Listing of ASCII Code*

Character	7-Bit ASCII	Decimal	Octal	Hex	Character	7-Bit ASCII	Decimal	Octal	Hex
A	100 0001	65	101	41	Y	101 1001	89	131	59
B	100 0010	66	102	42	Z	101 1010	90	132	5A
C	100 0011	67	103	43	0	011 0000	48	060	30
D	100 0100	68	104	44	1	011 0001	49	061	31
E	100 0101	69	105	45	2	011 0010	50	062	32
F	100 0110	70	106	46	3	011 0011	51	063	33
G	100 0111	71	107	47	4	011 0100	52	064	34
H	100 1000	72	110	48	5	011 0101	53	065	35
I	100 1001	73	111	49	6	011 0110	54	066	36
J	100 1010	74	112	4A	7	011 0111	55	067	37
K	100 1011	75	113	4B	8	011 1000	56	070	38
L	100 1100	76	114	4C	9	011 1001	57	071	39
M	100 1101	77	115	4D	blank	010 0000	32	040	20
N	100 1110	78	116	4E	.	010 1110	46	056	2E
O	100 1111	79	117	4F	(	110 1000	40	050	28
P	101 0000	80	120	50	+	010 1011	43	053	2B
Q	101 0001	81	121	51	$	010 0100	36	044	24
R	101 0010	82	122	52	*	010 1010	42	052	2A
					)	010 1001	41	051	29
S	101 0011	83	123	53	–	010 1101	45	055	2D
T	101 0100	84	124	54	/	010 1111	47	057	2F
U	101 0101	85	125	55	'	010 1100	39	054	2C
V	101 0110	86	126	56	=	011 1101	61	075	3D
W	101 0011	87	127	57	(RETURN)	000 1101	13	015	0D
X	101 1000	88	130	58	(LINEFEED)	000 1010	10	012	0A

EXAMPLE 8 *The following is a message encoded in ASCII code. What is the message ?*

1001000 1000101 1001100 1010000

SOLUTION Convert each 7-bit code to its hex equivalent. The results are : 48 45 4C 50

Now locate these hex values in Table 2.5 and determine the character represented by each. The results are : H E L P

The ASCII code is used for the transfer of alphanumeric information between a computer and input/output devices such as video terminals or printers. A computer also uses it internally to store the information that an operator types in at the computer's keyboard.

The following example illustrates this.

EXAMPLE 9 *An operator is typing in a BASIC program at the keyboard of a certain micro-computer. The computer converts each keystroke into its ASCII code and stores the code in memory. Determine the codes that will be entered into memory when the operator types in the following BASIC statement :* GOTO 25

SOLUTION Locate each character (including the space) in Table 2.5 and record its ASCII code.

G	1000111
O	1001111
T	1010100
O	1001111
(Space)	0100000
2	0110010
5	0110101

The main advantage of ASCII is its simplicity – it uses one byte to represent one character.

There is **extended ASCII** that uses 8 bits to represent various characters. It can represent 256 characters, as opposed to 128 characters of ASCII.

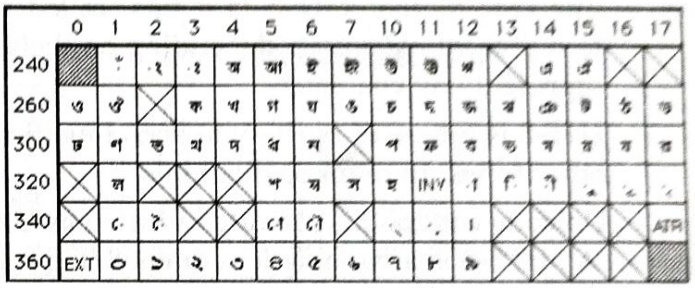
Apart from ASCII there are other systems that are also used to represent various symbols. In the following lines, we are talking about some of these — *ISCII* and *Unicode*.

2.4.2 ISCII Code

ISCII is an eight-bit code capable of coding 256 characters. ISCII code retains all ASCII characters and offers coding for Indian scripts also. Thus, it is also called ISCII (**Indian Scripts Code for Information Interchange**).

This standard does not only apply to the *Devanagari* script , but also to the *Gurmukhi, Gujarati, Oriya, Bengali, Assamese, Telugu, Kannada, Malayalam* and *Tamil script*. Because the structure of these scripts is so similar that a single coding can be applied to all of them, immediately providing *transliteration* between the scripts. Some of the ISCII versions for different scripts are being given below.

Devnagari ISCII script © ISCII



Bengali ISCII script © ISCII

2.4.3 Unicode

As world is becoming a global village thanks to modern technology, a need was being felt for an encoding scheme that could represent all the known languages characters through one encoding scheme. **Unicode** is the answer. Unicode is developed as a **universal character set** with an aim :

- to define all the characters needed for writing the majority of known languages in use on computers in one place.
- to be a superset of all other character sets that have been encoded.

Before Unicode was invented, there were hundreds of different encoding systems for assigning these numbers. These encoding systems also conflicted with one another. That is, two encodings could use the same number for two different characters, or use different numbers for the same character. For example, the character **code 0×FF** represents **Ÿ** in *west European code page ISO8859-1*, while the same code represented **Я** in Russian *code page 1251*.

Any given computer (especially servers) needs to support many different encodings; yet whenever data is passed between different encodings or platforms, that data always runs the risk of corruption.

Encoding Terminology

Before we proceed further, it is important to discuss *two* basic terms related to *encoding*.

Code Space. It refers to all the codes that an encoding scheme uses to represent characters. *e.g.*, ASCII encoding scheme has a code space from 0 to 127 (0 × 0 to 0 × 7F).

Code Point. The *code point* refers to a code (from a code space) that represents a single character from the character set represented by an encoding scheme, *e.g.*, 0×41 is one code point of ASCII that represents character 'A'.

ASCII has 128 code points while Unicode has 1,112,064 code points.

Check Point 2.2

1. Encode "INDIA" and "MARCHING" into ASCII codeform.
2. ASCII code is ____ bit code, extended ASCII is ____ bit code and Unicode is ____ bit code.
3. What is the use of ASCII code and Unicode ?
4. What is the full form of ASCII and Unicode ?
5. Name two Indian languages in Unicode along with their script names.

Unicode Encoding Schemes

Unicode defines multiple encoding systems to represent characters. These are UTF-8, UTF-16 and UTF-32. Let us discuss about two of these Unicode encoding schemes.

The character encoding scheme reflects the way the coded character set is actually **mapped to bytes** in form of binary code (machine intelligible code) for manipulation in a computer.

2.4.3A UTF-8 (Unicode Transformation Format)-8

NOTE

UTF-8 is another brilliant idea by Ken Thompson (one of the creators of UNIX).

UTF-8 is a variable-width encoding that can represent every character in Unicode character set. In other words, it can encode each of the 1,114,112 *code points* in the Unicode character set.

The *code unit* of UTF-8 is 8 bits, called an **octet**. UTF-8 can use 1 to maximum 6 octets to represent *code points* depending on their size, although till now it has used upto 4 octets to represent any character.

UTF-8 is a type of multi-byte encoding. Sometimes you only use 8 bits to store the character, other times, 16 or 24 or more bits.

Unicode codepoints are often written as U+<codepoint number> *e.g., U+0041* represents letter 'A'. UTF-8 is a variable length encoding scheme. That is, it uses different number of bytes or **octets** (set of 8 bits) to represent different characters. Following Table 2.6 gives an idea about how many **octets** will be used to represent a Unicode code point.

Table 2.6 *Unicode Code Points and No. of Octets used in UTF-8*

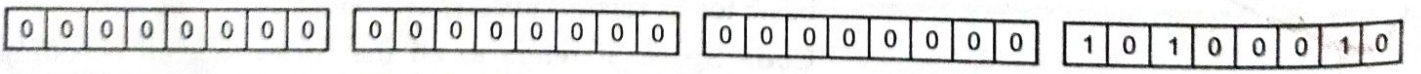
Unicode Code Points (in decimal)	Unicode Code Points (in Hexadecimal)	Number of Octets used
U-0 – U-127	(U+00 to U+07F)	1 octet (8 bits)
U-128 – U-2047	(U+80 to U+7FF)	2 octets (16 bits)
U-2048 – U-65535	(U+800 to U+FFFF)	3 octets (24 bits)
U-65536 – U-2097151	(U+10000 to U+1FFFFF)	4 octets (32 bits)

2.4.3B UTF-32

UTF-32 is a fixed length encoding scheme that uses exactly 4 bytes to represent all Unicode code points. That is, it directly stores the binary code of any Unicode code point in 4 bytes.

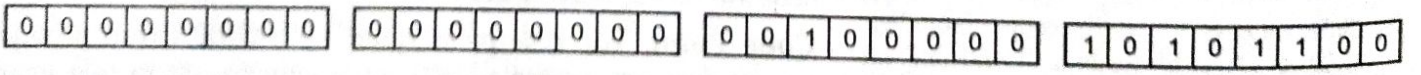
Consider these examples

- Symbols $ [Unicode code point : U + 0024, Binary code : 00100100]



- Symbol ¢ [Unicode code point : U + 00A2, Binary code : 10100010]

- Symbol € [Unicode code point : U + 20AC, Binary code : 10000010101100]

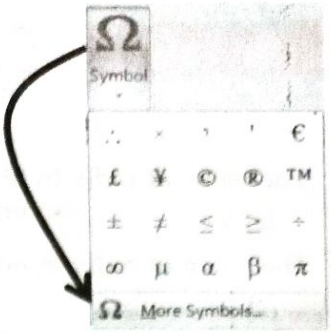


There is one more Unicode encoding scheme UTF-16, which is also a variable length encoding scheme that either uses 2 bytes or 4 bytes to represent Unicode code points. But we are not going into details of this encoding scheme as it is beyond the scope of the syllabus.

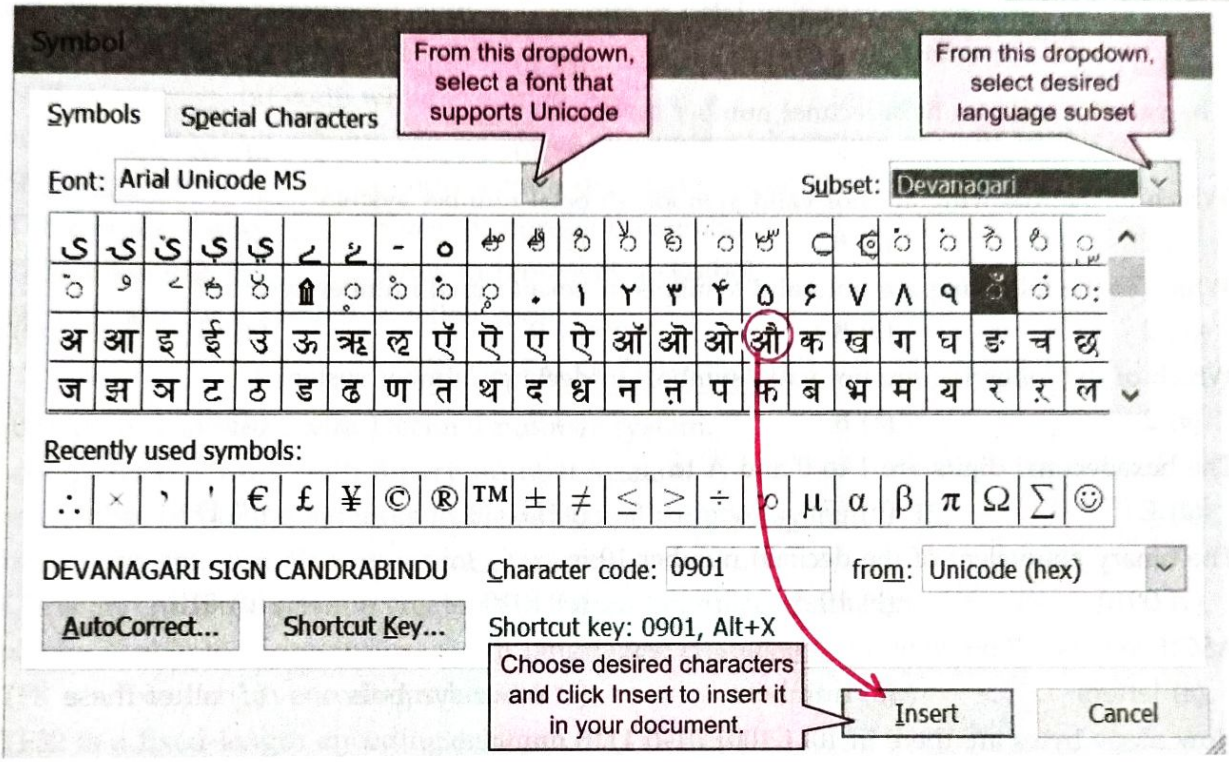
UTF-8 is the most popular encoding scheme with more than 90% websites using it.

Typing Multi-lingual Text in a Word Processor

You can easily type any multilingual text in a word processor. In the following lines you will learn how you can do this in Microsoft Word 2007 and later.



1. On the ***Insert** tab*, click **Symbol**, and from the drop down, click on **More Symbols** :
2. It will open the ***Symbols dialog***.



3. From the **Font** dropdown on the left, select a font that supports Unicode, *e.g.*, ***Arial Unicode MS***. From the **Subset** dropdown on the right, select desired language subset, *e.g.*, to type text in Hindi, we selected ***Devanagari***.
4. It will display the characters from the selected subset.
5. Click on the desired character to be inserted and click **Insert** button.

LET US REVISE

- *The decimal number system (base-10) is composed of 10 unique symbols : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.*
- *The binary number system (base-2) is composed of 2 unique symbols : 0 and 1.*
- *The octal number system (base-8) is composed of 8 unique symbols : 0, 1, 2, 3, 4, 5, 6, 7.*
- *The hexadecimal (hex) number system (base -16) is composed of 16 symbols : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.*
- *The decimal, binary, octal and hexadecimal systems are positional-value systems wherein each component digit carries certain weight depending upon its position.*
- *In a number of any number system, the rightmost digit carries the least weight, known as least significant digit, and the left most digit carries the largest weight, known as most significant digit.*
- *Characters are represented in memory by alpha-numeric codes. One such code is ASCII (American Standard Code for Information Interchange).*
- *ASCII and Unicode UTF-8 are popular character representation systems.*
- *UTF-8 is a variable length encoding scheme and UTF-32 is a fixed length encoding scheme.*