1

Number Systems

The study of *number systems* is important from the viewpoint of understanding how data are represented before they can be processed by any digital system including a digital computer. It is one of the most basic topics in digital electronics. In this chapter we will discuss different number systems commonly used to represent data. We will begin the discussion with the decimal number system. Although it is not important from the viewpoint of digital electronics, a brief outline of this will be given to explain some of the underlying concepts used in other number systems. This will then be followed by the more commonly used number systems such as the binary, octal and hexadecimal number systems.

1.1 Analogue Versus Digital

There are two basic ways of representing the numerical values of the various physical quantities with which we constantly deal in our day-to-day lives. One of the ways, referred to as *analogue*, is to express the numerical value of the quantity as a continuous range of values between the two expected extreme values. For example, the temperature of an oven settable anywhere from 0 to 100 °C may be measured to be 65 °C or 64.96 °C or 64.958 °C or even 64.9579 °C and so on, depending upon the accuracy of the measuring instrument. Similarly, voltage across a certain component in an electronic circuit may be measured as 6.5 V or 6.49 V or 6.487 V or 6.4869 V. The underlying concept in this mode of representation is that variation in the numerical value of the quantity is continuous and could have any of the infinite theoretically possible values between the two extremes.

The other possible way, referred to as *digital*, represents the numerical value of the quantity in steps of discrete values. The numerical values are mostly represented using binary numbers. For example, the temperature of the oven may be represented in steps of 1 °C as 64 °C, 65 °C, 66 °C and so on. To summarize, while an analogue representation gives a continuous output, a digital representation produces a discrete output. Analogue systems contain devices that process or work on various physical quantities represented in analogue form. Digital systems contain devices that process the physical quantities represented in digital form.

Digital Electronics: Principles, Devices and Applications Anil K. Maini

© 2007 John Wiley & Sons, Ltd. ISBN: 978-0-470-03214-5

Digital techniques and systems have the advantages of being relatively much easier to design and having higher accuracy, programmability, noise immunity, easier storage of data and ease of fabrication in integrated circuit form, leading to availability of more complex functions in a smaller size. The real world, however, is analogue. Most physical quantities – position, velocity, acceleration, force, pressure, temperature and flowrate, for example – are analogue in nature. That is why analogue variables representing these quantities need to be digitized or discretized at the input if we want to benefit from the features and facilities that come with the use of digital techniques. In a typical system dealing with analogue inputs and outputs, analogue variables are digitized at the input with the help of an analogue-to-digital converter block and reconverted back to analogue form at the output using a digital-to-analogue converter block. Analogue-to-digital and digital-to-analogue converter circuits are discussed at length in the latter part of the book. In the following sections we will discuss various number systems commonly used for digital representation of data.

1.2 Introduction to Number Systems

We will begin our discussion on various number systems by briefly describing the parameters that are common to all number systems. An understanding of these parameters and their relevance to number systems is fundamental to the understanding of how various systems operate. Different characteristics that define a number system include the number of independent digits used in the number system, the place values of the different digits constituting the number and the maximum numbers that can be written with the given number of digits. Among the three characteristic parameters, the most fundamental is the number of independent digits or symbols used in the number system. It is known as the *radix* or *base* of the number system. The decimal number system with which we are all so familiar can be said to have a radix of 10 as it has 10 independent digits, i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Similarly, the binary number system with only two independent digits, 0 and 1, is a radix-2 number system. The octal and hexadecimal number systems have a radix (or base) of 8 and 16 respectively. We will see in the following sections that the radix of the number system also determines the other two characteristics. The place values of different digits in the integer part of the number are given by r^0, r^1, r^2, r^3 and so on, starting with the digit adjacent to the radix point. For the fractional part, these are r^{-1}, r^{-2}, r^{-3} and so on, again starting with the digit next to the radix point. Here, r is the radix of the number system. Also, maximum numbers that can be written with n digits in a given number system are equal to r^n.

1.3 Decimal Number System

The decimal number system is a radix-10 number system and therefore has 10 different digits or symbols. These are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. All higher numbers after '9' are represented in terms of these 10 digits only. The process of writing higher-order numbers after '9' consists in writing the second digit (i.e. '1') first, followed by the other digits, one by one, to obtain the next 10 numbers from '10' to '19'. The next 10 numbers from '20' to '29' are obtained by writing the third digit (i.e. '2') first, followed by digits '0' to '9', one by one. The process continues until we have exhausted all possible two-digit combinations and reached '99'. Then we begin with three-digit combinations. The first three-digit number consists of the lowest two-digit number followed by '0' (i.e. 100), and the process goes on endlessly.

The place values of different digits in a mixed decimal number, starting from the decimal point, are 10^0, 10^1, 10^2 and so on (for the integer part) and 10^{-1}, 10^{-2}, 10^{-3} and so on (for the fractional part).

The value or magnitude of a given decimal number can be expressed as the sum of the various digits multiplied by their place values or weights.

As an illustration, in the case of the decimal number 3586.265, the integer part (i.e. 3586) can be expressed as

$$3586 = 6 \times 10^0 + 8 \times 10^1 + 5 \times 10^2 + 3 \times 10^3 = 6 + 80 + 500 + 3000 = 3586$$

and the fractional part can be expressed as

$$265 = 2 \times 10^{-1} + 6 \times 10^{-2} + 5 \times 10^{-3} = 0.2 + 0.06 + 0.005 = 0.265$$

We have seen that the place values are a function of the radix of the concerned number system and the position of the digits. We will also discover in subsequent sections that the concept of each digit having a place value depending upon the position of the digit and the radix of the number system is equally valid for the other more relevant number systems.

1.4 Binary Number System

The binary number system is a radix-2 number system with '0' and '1' as the two independent digits. All larger binary numbers are represented in terms of '0' and '1'. The procedure for writing higher-order binary numbers after '1' is similar to the one explained in the case of the decimal number system. For example, the first 16 numbers in the binary number system would be 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110 and 1111. The next number after 1111 is 10000, which is the lowest binary number with five digits. This also proves the point made earlier that a maximum of only 16 ($= 2^4$) numbers could be written with four digits. Starting from the binary point, the place values of different digits in a mixed binary number are 2^0, 2^1, 2^2 and so on (for the integer part) and 2^{-1}, 2^{-2}, 2^{-3} and so on (for the fractional part).

Example 1.1

Consider an arbitrary number system with the independent digits as 0, 1 and X. What is the radix of this number system? List the first 10 numbers in this number system.

Solution

- The radix of the proposed number system is 3.
- The first 10 numbers in this number system would be 0, 1, X, 10, 11, 1X, X0, X1, XX and 100.

1.4.1 Advantages

Logic operations are the backbone of any digital computer, although solving a problem on computer could involve an arithmetic operation too. The introduction of the mathematics of logic by George Boole laid the foundation for the modern digital computer. He reduced the mathematics of logic to a binary notation of '0' and '1'. As the mathematics of logic was well established and had proved itself to be quite useful in solving all kinds of logical problem, and also as the mathematics of logic (also known as Boolean algebra) had been reduced to a binary notation, the binary number system had a clear edge over other number systems for use in computer systems.

Yet another significant advantage of this number system was that all kinds of data could be conveniently represented in terms of 0s and 1s. Also, basic electronic devices used for hardware implementation could be conveniently and efficiently operated in two distinctly different modes. For example, a bipolar transistor could be operated either in cut-off or in saturation very efficiently.

Lastly, the circuits required for performing arithmetic operations such as addition, subtraction, multiplication, division, etc., become a simple affair when the data involved are represented in the form of 0s and 1s.

1.5 Octal Number System

The octal number system has a radix of 8 and therefore has eight distinct digits. All higher-order numbers are expressed as a combination of these on the same pattern as the one followed in the case of the binary and decimal number systems described in Sections 1.3 and 1.4. The independent digits are 0, 1, 2, 3, 4, 5, 6 and 7. The next 10 numbers that follow '7', for example, would be 10, 11, 12, 13, 14, 15, 16, 17, 20 and 21. In fact, if we omit all the numbers containing the digits 8 or 9, or both, from the decimal number system, we end up with an octal number system. The place values for the different digits in the octal number system are 8^0, 8^1, 8^2 and so on (for the integer part) and 8^{-1}, 8^{-2}, 8^{-3} and so on (for the fractional part).

1.6 Hexadecimal Number System

The hexadecimal number system is a radix-16 number system and its 16 basic digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. The place values or weights of different digits in a mixed hexadecimal number are 16^0, 16^1, 16^2 and so on (for the integer part) and 16^{-1}, 16^{-2}, 16^{-3} and so on (for the fractional part). The decimal equivalent of A, B, C, D, E and F are 10, 11, 12, 13, 14 and 15 respectively, for obvious reasons.

The hexadecimal number system provides a condensed way of representing large binary numbers stored and processed inside the computer. One such example is in representing addresses of different memory locations. Let us assume that a machine has 64K of memory. Such a memory has 64K ($= 2^{16} = 65\,536$) memory locations and needs 65 536 different addresses. These addresses can be designated as 0 to 65 535 in the decimal number system and 00000000 00000000 to 11111111 11111111 in the binary number system. The decimal number system is not used in computers and the binary notation here appears too cumbersome and inconvenient to handle. In the hexadecimal number system, 65 536 different addresses can be expressed with four digits from 0000 to FFFF. Similarly, the contents of the memory when represented in hexadecimal form are very convenient to handle.

1.7 Number Systems – Some Common Terms

In this section we will describe some commonly used terms with reference to different number systems.

1.7.1 Binary Number System

Bit is an abbreviation of the term 'binary digit' and is the smallest unit of information. It is either '0' or '1'. A *byte* is a string of eight bits. The byte is the basic unit of data operated upon as a single unit in computers. A *computer word* is again a string of bits whose size, called the 'word length' or 'word size', is fixed for a specified computer, although it may vary from computer to computer. The word length may equal one byte, two bytes, four bytes or be even larger.

The *1's complement* of a binary number is obtained by complementing all its bits, i.e. by replacing 0s with 1s and 1s with 0s. For example, the 1's complement of $(10010110)_2$ is $(01101001)_2$. The *2's complement* of a binary number is obtained by adding '1' to its 1's complement. The 2's complement of $(10010110)_2$ is $(01101010)_2$.

1.7.2 Decimal Number System

Corresponding to the 1's and 2's complements in the binary system, in the decimal number system we have the 9's and 10's complements. The *9's complement* of a given decimal number is obtained by subtracting each digit from 9. For example, the 9's complement of $(2496)_{10}$ would be $(7503)_{10}$. The *10's complement* is obtained by adding '1' to the 9's complement. The 10's complement of $(2496)_{10}$ is $(7504)_{10}$.

1.7.3 Octal Number System

In the octal number system, we have the 7's and 8's complements. The *7's complement* of a given octal number is obtained by subtracting each octal digit from 7. For example, the 7's complement of $(562)_8$ would be $(215)_8$. The *8's complement* is obtained by adding '1' to the 7's complement. The 8's complement of $(562)_8$ would be $(216)_8$.

1.7.4 Hexadecimal Number System

The 15's and 16's complements are defined with respect to the hexadecimal number system. The *15's complement* is obtained by subtracting each hex digit from 15. For example, the 15's complement of $(3BF)_{16}$ would be $(C40)_{16}$. The *16's complement* is obtained by adding '1' to the 15's complement. The 16's complement of $(2AE)_{16}$ would be $(D52)_{16}$.

1.8 Number Representation in Binary

Different formats used for binary representation of both positive and negative decimal numbers include the sign-bit magnitude method, the 1's complement method and the 2's complement method.

1.8.1 Sign-Bit Magnitude

In the sign-bit magnitude representation of positive and negative decimal numbers, the MSB represents the 'sign', with a '0' denoting a plus sign and a '1' denoting a minus sign. The remaining bits represent the magnitude. In eight-bit representation, while MSB represents the sign, the remaining seven bits represent the magnitude. For example, the eight-bit representation of $+9$ would be 00001001, and that for -9 would be 10001001. An $n-$bit binary representation can be used to represent decimal numbers in the range of $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$. That is, eight-bit representation can be used to represent decimal numbers in the range from -127 to $+127$ using the sign-bit magnitude format.

1.8.2 1's Complement

In the 1's complement format, the positive numbers remain unchanged. The negative numbers are obtained by taking the 1's complement of the positive counterparts. For example, +9 will be represented as 00001001 in eight-bit notation, and −9 will be represented as 11110110, which is the 1's complement of 00001001. Again, n-bit notation can be used to represent numbers in the range from $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$ using the 1's complement format. The eight-bit representation of the 1's complement format can be used to represent decimal numbers in the range from −127 to +127.

1.8.3 2's Complement

In the 2's complement representation of binary numbers, the MSB represents the sign, with a '0' used for a plus sign and a '1' used for a minus sign. The remaining bits are used for representing magnitude. Positive magnitudes are represented in the same way as in the case of sign-bit or 1's complement representation. Negative magnitudes are represented by the 2's complement of their positive counterparts. For example, +9 would be represented as 00001001, and −9 would be written as 11110111. Please note that, if the 2's complement of the magnitude of +9 gives a magnitude of −9, then the reverse process will also be true, i.e. the 2's complement of the magnitude of −9 will give a magnitude of +9. The n-bit notation of the 2's complement format can be used to represent all decimal numbers in the range from $+(2^{n-1}-1)$ to $-(2^{n-1})$. The 2's complement format is very popular as it is very easy to generate the 2's complement of a binary number and also because arithmetic operations are relatively easier to perform when the numbers are represented in the 2's complement format.

1.9 Finding the Decimal Equivalent

The decimal equivalent of a given number in another number system is given by the sum of all the digits multiplied by their respective place values. The integer and fractional parts of the given number should be treated separately. Binary-to-decimal, octal-to-decimal and hexadecimal-to-decimal conversions are illustrated below with the help of examples.

1.9.1 Binary-to-Decimal Conversion

The decimal equivalent of the binary number $(1001.0101)_2$ is determined as follows:

- The integer part = 1001
- The decimal equivalent $= 1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 = 1 + 0 + 0 + 8 = 9$
- The fractional part = .0101
- Therefore, the decimal equivalent $= 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 0 + 0.25 + 0$
  $+ 0.0625 = 0.3125$
- Therefore, the decimal equivalent of $(1001.0101)_2 = 9.3125$

1.9.2 Octal-to-Decimal Conversion

The decimal equivalent of the octal number $(137.21)_8$ is determined as follows:

- The integer part = 137
- The decimal equivalent $= 7 \times 8^0 + 3 \times 8^1 + 1 \times 8^2 = 7 + 24 + 64 = 95$

- The fractional part = .21
- The decimal equivalent = $2 \times 8^{-1} + 1 \times 8^{-2} = 0.265$
- Therefore, the decimal equivalent of $(137.21)_8 = (95.265)_{10}$

1.9.3 Hexadecimal-to-Decimal Conversion

The decimal equivalent of the hexadecimal number $(1E0.2A)_{16}$ is determined as follows:

- The integer part = 1E0
- The decimal equivalent = $0 \times 16^0 + 14 \times 16^1 + 1 \times 16^2 = 0 + 224 + 256 = 480$
- The fractional part = 2A
- The decimal equivalent = $2 \times 16^{-1} + 10 \times 16^{-2} = 0.164$
- Therefore, the decimal equivalent of $(1E0.2A)_{16} = (480.164)_{10}$

Example 1.2

Find the decimal equivalent of the following binary numbers expressed in the 2's complement format:

(a) 00001110;
(b) 10001110.

Solution

(a) The MSB bit is '0', which indicates a plus sign.
The magnitude bits are 0001110.

$$\begin{aligned}\text{The decimal equivalent} &= 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 0 \times 2^6 \\ &= 0 + 2 + 4 + 8 + 0 + 0 + 0 = 14\end{aligned}$$

Therefore, 00001110 represents +14

(b) The MSB bit is '1', which indicates a minus sign
The magnitude bits are therefore given by the 2's complement of 0001110, i.e. 1110010

$$\begin{aligned}\text{The decimal equivalent} &= 0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 \\ &\quad + 1 \times 2^6 \\ &= 0 + 2 + 0 + 0 + 16 + 32 + 64 = 114\end{aligned}$$

Therefore, 10001110 represents −114

1.10 Decimal-to-Binary Conversion

As outlined earlier, the integer and fractional parts are worked on separately. For the integer part, the binary equivalent can be found by successively dividing the integer part of the number by 2 and recording the remainders until the quotient becomes '0'. The remainders written in reverse order constitute the binary equivalent. For the fractional part, it is found by successively multiplying the fractional part of the decimal number by 2 and recording the carry until the result of multiplication is '0'. The carry sequence written in forward order constitutes the binary equivalent of the fractional part of the decimal number. If the result of multiplication does not seem to be heading towards zero in the case of the fractional part, the process may be continued only until the requisite number of equivalent bits has been obtained. This method of decimal–binary conversion is popularly known as the double-dabble method. The process can be best illustrated with the help of an example.

Example 1.3

We will find the binary equivalent of $(13.375)_{10}$.

Solution

- The integer part = 13

Divisor	Dividend	Remainder
2	13	—
2	6	1
2	3	0
2	1	1
—	0	1

- The binary equivalent of $(13)_{10}$ is therefore $(1101)_2$
- The fractional part = .375
- 0.375 × 2 = 0.75 with a carry of 0
- 0.75 × 2 = 0.5 with a carry of 1
- 0.5 × 2 = 0 with a carry of 1
- The binary equivalent of $(0.375)_{10} = (.011)_2$
- Therefore, the binary equivalent of $(13.375)_{10} = (1101.011)_2$

1.11 Decimal-to-Octal Conversion

The process of decimal-to-octal conversion is similar to that of decimal-to-binary conversion. The progressive division in the case of the integer part and the progressive multiplication while working on the fractional part here are by ‘8’ which is the radix of the octal number system. Again, the integer and fractional parts of the decimal number are treated separately. The process can be best illustrated with the help of an example.

Example 1.4

We will find the octal equivalent of $(73.75)_{10}$.

Solution

- The integer part = 73

Divisor	Dividend	Remainder
8	73	—
8	9	1
8	1	1
—	0	1

- The octal equivalent of $(73)_{10} = (111)_8$
- The fractional part = 0.75
- $0.75 \times 8 = 0$ with a carry of 6
- The octal equivalent of $(0.75)_{10} = (.6)_8$
- Therefore, the octal equivalent of $(73.75)_{10} = (111.6)_8$

1.12 Decimal-to-Hexadecimal Conversion

The process of decimal-to-hexadecimal conversion is also similar. Since the hexadecimal number system has a base of 16, the progressive division and multiplication factor in this case is 16. The process is illustrated further with the help of an example.

Example 1.5

Let us determine the hexadecimal equivalent of $(82.25)_{10}$.

Solution

- The integer part = 82

Divisor	Dividend	Remainder
16	82	—
16	5	2
—	0	5

- The hexadecimal equivalent of $(82)_{10} = (52)_{16}$
- The fractional part = 0.25
- $0.25 \times 16 = 0$ with a carry of 4
- Therefore, the hexadecimal equivalent of $(82.25)_{10} = (52.4)_{16}$

1.13 Binary–Octal and Octal–Binary Conversions

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. A binary number can be converted into an equivalent octal number by splitting the integer and fractional parts into groups of three bits, starting from the binary point on both sides. The 0s can be added to complete the outside groups if needed.

Example 1.6

Let us find the binary equivalent of $(374.26)_8$ and the octal equivalent of $(1110100.0100111)_2$.

Solution

- The given octal number = $(374.26)_8$
- The binary equivalent = $(011\ 111\ 100.010\ 110)_2 = (011111100.010110)_2$

- Any 0s on the extreme left of the integer part and extreme right of the fractional part of the equivalent binary number should be omitted. Therefore, $(011111100.010110)_2$= $(11111100.01011)_2$
- The given binary number = $(1110100.0100111)_2$
- $(1110100.0100111)_2$ = $(1\ 110\ 100.010\ 011\ 1)_2$
  = $(001\ 110\ 100.010\ 011\ 100)_2$ = $(164.234)_8$

1.14 Hex–Binary and Binary–Hex Conversions

A hexadecimal number can be converted into its binary equivalent by replacing each hex digit with its four-bit binary equivalent. We take the four-bit equivalent because the base of the hexadecimal number system is 16 and it is the fourth power of the base of the binary number system. All we have then to remember is the four-bit binary equivalents of the basic digits of the hexadecimal number system. A given binary number can be converted into an equivalent hexadecimal number by splitting the integer and fractional parts into groups of four bits, starting from the binary point on both sides. The 0s can be added to complete the outside groups if needed.

Example 1.7

Let us find the binary equivalent of $(17E.F6)_{16}$ and the hex equivalent of $(1011001110.011011101)_2$.

Solution

- The given hex number = $(17E.F6)_{16}$
- The binary equivalent = $(0001\ 0111\ 1110.1111\ 0110)_2$
  = $(000101111110.11110110)_2$
  = $(101111110.1111011)_2$
- The 0s on the extreme left of the integer part and on the extreme right of the fractional part have been omitted.
- The given binary number = $(1011001110.011011101)_2$
  = $(10\ 1100\ 1110.0110\ 1110\ 1)_2$
- The hex equivalent = $(0010\ 1100\ 1110.0110\ 1110\ 1000)_2$ = $(2CE.6E8)_{16}$

1.15 Hex–Octal and Octal–Hex Conversions

For hexadecimal–octal conversion, the given hex number is firstly converted into its binary equivalent which is further converted into its octal equivalent. An alternative approach is firstly to convert the given hexadecimal number into its decimal equivalent and then convert the decimal number into an equivalent octal number. The former method is definitely more convenient and straightforward. For octal–hexadecimal conversion, the octal number may first be converted into an equivalent binary number and then the binary number transformed into its hex equivalent. The other option is firstly to convert the given octal number into its decimal equivalent and then convert the decimal number into its hex equivalent. The former approach is definitely the preferred one. Two types of conversion are illustrated in the following example.

Example 1.8

Let us find the octal equivalent of $(2F.C4)_{16}$ and the hex equivalent of $(762.013)_8$.

Solution

- The given hex number = $(2F.C4)_{16}$.
- The binary equivalent = $(0010\ 1111.1100\ 0100)_2 = (00101111.11000100)_2$
  $= (101111.110001)_2 = (101\ 111.110\ 001)_2 = (57.61)_8$.
- The given octal number = $(762.013)_8$.
- The octal number = $(762.013)_8 = (111\ 110\ 010.000\ 001\ 011)_2$
  $= (111110010.000001011)_2$
  $= (0001\ 1111\ 0010.0000\ 0101\ 1000)_2 = (1F2.058)_{16}$.

1.16 The Four Axioms

Conversion of a given number in one number system to its equivalent in another system has been discussed at length in the preceding sections. The methodology has been illustrated with solved examples. The complete methodology can be summarized as four axioms or principles, which, if understood properly, would make it possible to solve any problem related to conversion of a given number in one number system to its equivalent in another number system. These principles are as follows:

1. Whenever it is desired to find the decimal equivalent of a given number in another number system, it is given by the sum of all the digits multiplied by their weights or place values. The integer and fractional parts should be handled separately. Starting from the radix point, the weights of different digits are r^0, r^1, r^2 for the integer part and r^{-1}, r^{-2}, r^{-3} for the fractional part, where r is the radix of the number system whose decimal equivalent needs to be determined.
2. To convert a given mixed decimal number into an equivalent in another number system, the integer part is progressively divided by r and the remainders noted until the result of division yields a zero quotient. The remainders written in reverse order constitute the equivalent. r is the radix of the transformed number system. The fractional part is progressively multiplied by r and the carry recorded until the result of multiplication yields a zero or when the desired number of bits has been obtained. The carrys written in forward order constitute the equivalent of the fractional part.
3. The octal–binary conversion and the reverse process are straightforward. For octal–binary conversion, replace each digit in the octal number with its three-bit binary equivalent. For hexadecimal–binary conversion, replace each hex digit with its four-bit binary equivalent. For binary–octal conversion, split the binary number into groups of three bits, starting from the binary point, and, if needed, complete the outside groups by adding 0s, and then write the octal equivalent of these three-bit groups. For binary–hex conversion, split the binary number into groups of four bits, starting from the binary point, and, if needed, complete the outside groups by adding 0s, and then write the hex equivalent of the four-bit groups.
4. For octal–hexadecimal conversion, we can go from the given octal number to its binary equivalent and then from the binary equivalent to its hex counterpart. For hexadecimal–octal conversion, we can go from the hex to its binary equivalent and then from the binary number to its octal equivalent.

Example 1.9

Assume an arbitrary number system having a radix of 5 and 0, 1, 2, L and M as its independent digits. Determine:

(a) the decimal equivalent of (12LM.L1);
(b) the total number of possible four-digit combinations in this arbitrary number system.

Solution

(a) The decimal equivalent of (12LM) is given by

$$\mathrm{M}\times 5^0+\mathrm{L}\times 5^1+2\times 5^2+1\times 5^3=4\times 5^0+3\times 5^1+2\times 5^2+1\times 5^3(\mathrm{L}=3,\mathrm{M}=4)$$
$$=4+15+50+125=194$$

The decimal equivalent of (L1) is given by

$$\mathrm{L}\times 5^{-1}+1\times 5^{-2}=3\times 5^{-1}+5^{-2}=0.64$$

Combining the results, $(12\mathrm{LM.L1})_5=(194.64)_{10}$.

(b) The total number of possible four-digit combinations $= 5^4 = 625$.

Example 1.10

The 7's complement of a certain octal number is 5264. Determine the binary and hexadecimal equivalents of that octal number.

Solution

- The 7's complement = 5264.
- Therefore, the octal number = $(2513)_8$.
- The binary equivalent = $(010\ 101\ 001\ 011)_2=(10101001011)_2$.
- Also, $(10101001011)_2=(101\ 0100\ 1011)_2=(0101\ 0100\ 1011)_2=(54\mathrm{B})_{16}$.
- Therefore, the hex equivalent of $(2513)_8=(54\mathrm{B})_{16}$ and the binary equivalent of $(2513)_8=(10101001011)_2$.

1.17 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

$$N=m\times b^e \tag{1.1}$$

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 ($\pm d.dddd\ldots dd$), 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 (1.1) in the case of decimal, hexadecimal and binary number systems will be written as follows:

Decimal system

$$N=m\times 10^e \tag{1.2}$$

Hexadecimal system

$$N = m \times 16^e \qquad (1.3)$$

Binary system

$$N = m \times 2^e \qquad (1.4)$$

For example, decimal numbers 0.0003754 and 3754 will be represented in floating-point notation as 3.754×10^{-4} and 3.754×10^3 respectively. A hex number 257.ABF will be represented as $2.57\text{ABF} \times 16^2$. 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 $.1101011 \times 2^3 = .1101011e+0011$. Here, .1101011 is the mantissa and $e+0011$ implies that the exponent is +3. As another example, $(0.000111)_2$ will be written as $.111e-0011$, with .111 being the mantissa and $e-0011$ implying an exponent of −3. Also, $(-0.00000101)_2$ may be written as $-.101 \times 2^{-5} = -.101e-0101$, where −.101 is the mantissa and $e-0101$ indicates an exponent of −5. If we wanted to represent the mantissas using eight bits, then .1101011 and .111 would be represented as .11010110 and .11100000.

1.17.1 Range of Numbers and Precision

The range of numbers that can be represented in any machine depends upon the number of bits in the exponent, while the fractional accuracy or precision is ultimately determined by the number of bits in the mantissa. The higher the number of bits in the exponent, the larger is the range of numbers that can be represented. For example, the range of numbers possible in a floating-point binary number format using six bits to represent the magnitude of the exponent would be from 2^{-64} to 2^{+64}, which is equivalent to a range of $10^{-19}$to 10^{+19}. The precision is determined by the number of bits used to represent the mantissa. It is usually represented as decimal digits of precision. The concept of precision as defined with respect to floating-point notation can be explained in simple terms as follows. If the mantissa is stored in n number of bits, it can represent a decimal number between 0 and $2^n - 1$ as the mantissa is stored as an unsigned integer. If M is the largest number such that $10^M - 1$ is less than or equal to $2^n - 1$, then M is the precision expressed as decimal digits of precision. For example, if the mantissa is expressed in 20 bits, then decimal digits of precision can be found to be about 6, as $2^{20} - 1$ equals 1 048 575, which is a little over $10^6 - 1$. We will briefly describe the commonly used formats for binary floating-point number representation.

1.17.2 Floating-Point Number Formats

The most commonly used format for representing floating-point numbers is the IEEE-754 standard. The full title of the standard is IEEE Standard for Binary Floating-point Arithmetic (ANSI/IEEE STD 754-1985). It is also known as Binary Floating-point Arithmetic for Microprocessor Systems, IEC

60559:1989. An ongoing revision to IEEE-754 is IEEE-754r. Another related standard IEEE 854-1987 generalizes IEEE-754 to cover both binary and decimal arithmetic. A brief description of salient features of the IEEE-754 standard, along with an introduction to other related standards, is given below.

ANSI/IEEE-754 Format

The IEEE-754 floating point is the most commonly used representation for real numbers on computers including Intel-based personal computers, Macintoshes 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. Table 1.1 lists characteristic parameters of the four formats contained in the IEEE-754 standard. 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.

Figure 1.1 shows the basic constituent parts of the single- and double-precision formats. 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. A ‘0’ denotes a positive number and a ‘1’ denotes a negative number. The n-bit exponent field needs to represent both positive and negative exponent values. To achieve this, a bias equal to $2^{n-1}-1$ is added to the actual exponent in order to obtain the stored exponent. This equals 127 for an eight-bit exponent of the single-precision format and 1023 for an 11-bit exponent of the double-precision format. The addition of bias allows the use of an exponent in the range from -127 to $+128$, corresponding to a range of 0–255 in the first case, and in the range from -1023 to $+1024$, corresponding to a range of 0–2047 in the second case. A negative exponent is always represented in 2’s complement form. The single-precision format offers a range from 2^{-127} to 2^{+127}, which is equivalent to 10^{-38} to 10^{+38}. The figures are 2^{-1023} to 2^{+1023}, which is equivalent to 10^{-308} to 10^{+308} in the case of the double-precision format.

The extreme exponent values are reserved for representing special values. For example, in the case of the single-precision format, for an exponent value of -127, the biased exponent value is zero, represented by an all 0s exponent field. In the case of a biased exponent of zero, if the mantissa is zero as well, the value of the floating-point number is exactly zero. If the mantissa is nonzero, it represents a denormalized number that does not have an assumed leading bit of ‘1’. A biased exponent of $+255$, corresponding to an actual exponent of $+128$, is represented by an all 1s exponent field. If the mantissa is zero, the number represents infinity. The sign bit is used to distinguish between positive and negative infinity. If the mantissa is nonzero, the number represents a ‘NaN’ (Not a Number). The value NaN is used to represent a value that does not represent a real number. This means that an eight-bit exponent can represent exponent values between -126 and $+127$. Referring to Fig. 1.1(a), the MSB of byte 1 indicates the sign of the mantissa. The remaining seven bits of byte 1 and the MSB of byte 2 represent an eight-bit exponent. The remaining seven bits of byte 2 and the 16 bits of byte 3 and byte 4 give a 23-bit mantissa. The mantissa m is normalized. The left-hand bit of the normalized mantissa is always

Table 1.1 Characteristic parameters of IEEE-754 formats.

Precision	Sign (bits)	Exponent (bits)	Mantissa (bits)	Total length (bits)	Decimal digits of precision
Single	1	8	23	32	>6
Single-extended	1	≥ 11	≥ 32	≥ 44	>9
Double	1	11	52	64	>15
Double-extended	1	≥ 15	≥ 64	≥ 80	>19

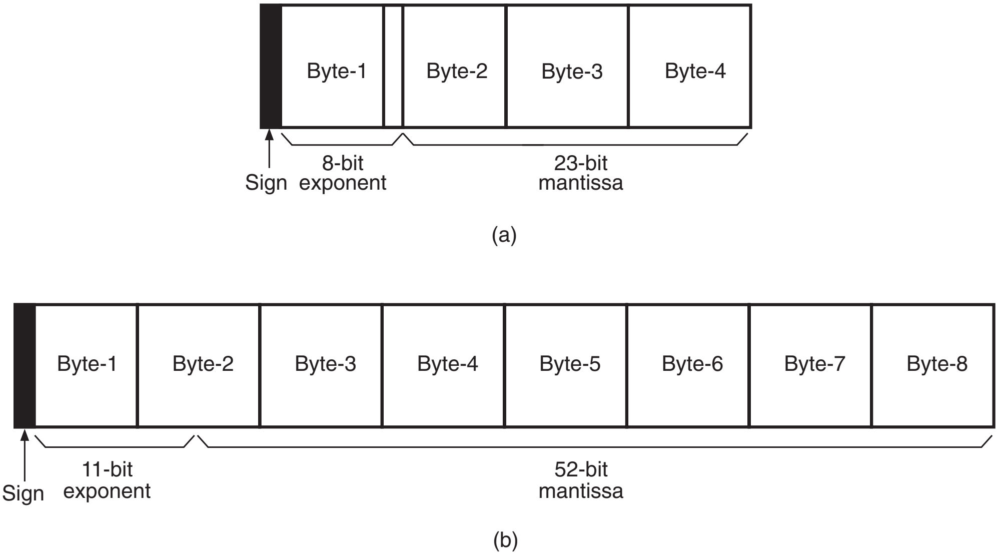

Figure 1.1 Single-precision and double-precision formats.

‘1’. This ‘1’ is not included but is always implied. A similar explanation can be given in the case of the double-precision format shown in Fig. 1.1(b).

Step-by-step transformation of $(23)_{10}$ into an equivalent floating-point number in single-precision IEEE format is as follows:

- $(23)_{10} = (10111)_2 = 1.0111e + 0100$.
- The mantissa = 0111000 00000000 00000000.
- The exponent = 00000100.
- The biased exponent = 00000100 + 01111111 = 10000011.
- The sign of the mantissa = 0.
- $(+23)_{10}$ = 01000001 10111000 00000000 00000000.
- Also, $(-23)_{10}$= 11000001 10111000 00000000 00000000.

IEEE-754r Format

As mentioned earlier, IEEE-754r is an ongoing revision to the IEEE-754 standard. The main objective of the revision is to extend the standard wherever it has become necessary, the most obvious enhancement to the standard being the addition of the 128-bit format and decimal format. Extension of the standard to include decimal floating-point representation has become necessary as most commercial data are held in decimal form and the binary floating point cannot represent decimal fractions exactly. If the binary floating point is used to represent decimal data, it is likely that the results will not be the same as those obtained by using decimal arithmetic.

In the revision process, many of the definitions have been rewritten for clarification and consistency. In terms of the addition of new formats, a new addition to the existing binary formats is the 128-bit ‘quad-precision’ format. Also, three new decimal formats, matching the lengths of binary formats, have been described. These include decimal formats with a seven-, 16- and 34-digit mantissa, which may be normalized or denormalized. In order to achieve maximum range (decided by the number of exponent bits) and precision (decided by the number of mantissa bits), the formats merge part of the exponent and mantissa into a combination field and compress the remainder of the mantissa using densely packed decimal encoding. Detailed description of the revision, however, is beyond the scope of this book.

IEEE-854 Standard

The main objective of the IEEE-854 standard was to define a standard for floating-point arithmetic without the radix and word length dependencies of the better-known IEEE-754 standard. That is why IEEE-854 is called the IEEE standard for radix-independent floating-point arithmetic. Although the standard specifies only the binary and decimal floating-point arithmetic, it provides sufficient guidelines for those contemplating the implementation of the floating point using any other radix value such as 16 of the hexadecimal number system. This standard, too, specifies four formats including single, single-extended, double and double-extended precision formats.

Example 1.11

Determine the floating-point representation of $(-142)_{10}$ using the IEEE single-precision format.

Solution

- As a first step, we will determine the binary equivalent of $(142)_{10}$. Following the procedure outlined in an earlier part of the chapter, the binary equivalent can be written as $(142)_{10} = (10001110)_2$.
- $(10001110)_2 = 1.000\ 1110 \times 2^7 = 1.0001110e + 0111$.
- The mantissa = 0001110 00000000 00000000.
- The exponent = 00000111.
- The biased exponent = 00000111 + 01111111 = 10000110.
- The sign of the mantissa = 1.
- Therefore, $(-142)_{10}$ = 11000011 00001110 00000000 00000000.

Example 1.12

Determine the equivalent decimal numbers for the following floating-point numbers:

(a) 00111111 01000000 00000000 00000000 (IEEE-754 single-precision format);
(b) 11000000 00101001 01100 . . . 45 0s (IEEE-754 double-precision format).

Solution

(a) From an examination of the given number:

The sign of the mantissa is positive, as indicated by the ‘0’ bit in the designated position.

The biased exponent = 01111110.

The unbiased exponent = 01111110 − 01111111 = 11111111.

It is clear from the eight bits of unbiased exponent that the exponent is negative, as the 2’s complement representation of a number gives ‘1’ in place of MSB.

The magnitude of the exponent is given by the 2’s complement of $(11111111)_2$, which is $(00000001)_2 = 1$.

Therefore, the exponent $= -1$.
The mantissa bits = 11000000 00000000 00000000 ('1' in MSB is implied).
The normalized mantissa = 1.1000000 00000000 00000000.
The magnitude of the mantissa can be determined by shifting the mantissa bits one position to the left.
That is, the mantissa $= (.11)_2 = (0.75)_{10}$.

(b) The sign of the mantissa is negative, indicated by the '1' bit in the designated position.
The biased exponent = 10000000010.
The unbiased exponent = 10000000010 − 01111111111 = 00000000011.
It is clear from the 11 bits of unbiased exponent that the exponent is positive owing to the '0' in place of MSB. The magnitude of the exponent is 3. Therefore, the exponent = +3.
The mantissa bits = 1100101100 . . . 45 0s ('1' in MSB is implied).
The normalized mantissa = 1.100101100 . . . 45 0s.
The magnitude of the mantissa can be determined by shifting the mantissa bits three positions to the right.
That is, the mantissa $= (1100.101)_2 = (12.625)_{10}$.
Therefore, the equivalent decimal number $= -12.625$.

Review Questions

1. What is meant by the radix or base of a number system? Briefly describe why hex representation is used for the addresses and the contents of the memory locations in the main memory of a computer.
2. What do you understand by the 1's and 2's complements of a binary number? What will be the range of decimal numbers that can be represented using a 16-bit 2's complement format?
3. Briefly describe the salient features of the IEEE-754 standard for representing floating-point numbers.
4. Why was it considered necessary to carry out a revision of the IEEE-754 standard? What are the main features of IEEE-754r (the notation for IEEE-754 under revision)?
5. In a number system, what decides (a) the place value or weight of a given digit and (b) the maximum numbers representable with a given number of digits?
6. In a floating-point representation, what represents (a) the range of representable numbers and (b) the precision with which a given number can be represented?
7. Why is there a need to have floating-point standards that can take care of decimal data and decimal arithmetic in addition to binary data and arithmetic?

Problems

1. Do the following conversions:

   (a) eight-bit 2's complement representation of $(-23)_{10}$;
   (b) The decimal equivalent of $(00010111)_2$ represented in 2's complement form.

   (a) 11101001; (b) +23

2. Two possible binary representations of $(-1)_{10}$ are $(10000001)_2$ and $(11111111)_2$. One of them belongs to the sign-bit magnitude format and the other to the 2's complement format. Identify.

   $(10000001)_2$ = sign-bit magnitude and $(11111111)_2$ = 2's complement form

3. Represent the following in the IEEE-754 floating-point standard using the single-precision format:

   (a) 32-bit binary number 11110000 11001100 10101010 00001111;
   (b) $(-118.625)_{10}$.

(a) 01001111 01110000 11001100 10101010;
(b) 11000010 11101101 01000000 00000000

4. Give the next three numbers in each of the following hex sequences:

   (a) 4A5, 4A6, 4A7, 4A8, . . . ;
   (b) B998, B999, . . .

(a) 4A9, 4AA, 4AB; (b) B99A, B99B, B99C

5. Show that:

   (a) $(13A7)_{16} = (5031)_{10}$;
   (b) $(3F2)_{16} = (1111110010)_2$.

6. Assume a radix-32 arbitrary number system with 0–9 and A–V as its basic digits. Express the mixed binary number $(110101.001)_2$ in this arbitrary number system.

1L.4

Further Reading

1. Tokheim, R. L. (1994) *Schaum's Outline Series of Digital Principles*, McGraw-Hill Companies Inc., USA.
2. Atiyah, S. K. (2005) *A Survey of Arithmetic*, Trafford Publishing, Victoria, BC, Canada.
3. Langholz, G., Mott, J. L. and Kandel, A. (1998) *Foundations of Digital Logic Design*, World Scientific Publ. Co. Inc., Singapore.
4. Cook, N. P. (2003) *Practical Digital Electronics*, Prentice-Hall, NJ, USA.
5. Lu, M. (2004) *Arithmetic and Logic in Computer Systems*, John Wiley & Sons, Inc., NJ, USA.