Chapter 1

Digital Representation of Information

This chapter is focused on the ways computers represent information in digital format. In particular, it discusses the mechanisms used to represent various quantities in a digital computer. The digital electronic circuits that are commonly used in a digital computer can assume only one of two possible values, which implies that different quantities have to be represented in a format compatible with this restriction.

We will start by describing how integers are represented, both in the decimal number system, which is familiar to everybody and in other number systems more adequated to be manipulated by computers. In this chapter, this study is limited to non-negative integers and non-negative fractional numbers.

The chapter begins with Section 1.1, where we study binary, octal and hexadecimal number systems. Section 1.2 focuses on studying the foundations of binary arithmetic. Section 1.3 deals with the use of codes, both numeric (with emphasis on decimal codes) as well as alphanumeric (to represent other types of information). Section 1.4 concludes the chapter with some basic concepts on the organisation of binary representation of information.

1.1 Number Systems

This chapter deals with the representation of non-signed integer and fractional numbers. Later, in Chapter 5, this subject will be reconsidered to address the representation of signed integer and real numbers. The representation of numbers in digital systems has to be undertaken considering that they use devices which can represent only two possible values.

Given that the common representation of numbers is based on the utilisation of a decimal *number system*, using base-10, it is natural to consider that the representation of numbers in digital systems may be made using the binary system, using base-2. The *base* is the number of digits used to represent a number under a given number system.

The general case of representation using a generic base-b will be studied first, followed by the study of the base-2 case.

1.1.1 *Representation of Integers in Base-b*

The representation of a non-signed integer in base-10 is made using a sequence of digits. The number 435, for example, is represented by the sequence of digits 4, 3 and 5. The interpretation of the representation of a number results, firstly, from the digits used and, secondly, from their position within the sequence. As is evident, $435 \neq 354$, even though the digits used are the same.

The position of the digits indicates the *weight* for each digit. In the previous example, because the digit 4 is in the third position from the right, this means, in fact, four hundreds. The digit 3 represents three tens, and 5 represents five units. This system of representation of numbers is referred as *positional*.

This analysis can be stated more formally as follows:

$$\begin{aligned} 435 &= 400 + 30 + 5 \\ &= 4 \times 100 + 3 \times 10 + 5 \end{aligned} \tag{1.1}$$

or, expressing the powers of 10 involved,

$$435 = 4 \times 10^2 + 3 \times 10^1 + 5 \times 10^0 \tag{1.2}$$

which is a more general way of representation, emphasising the powers of the base.

The number 435 is said to be represented in *base-10* since it results from the sum of consecutive powers of 10, each multiplied by the value of the corresponding digit as shown in Equation (1.2). To explicitly indicate that the number is represented in base-10, the following notation is used: 435_{10}. To represent a number in base-10, the weights of each power of 10 are indicated using digits from 0 to 9, using a total of 10 distinct digits.

There is nothing to prevent the use of another base to represent a number. Consider, for example, the sequence of digits 1161 in base-7, which is usually indicated by 1161_7. In this case, this representation has the following meaning:

$$\begin{aligned} 1161_7 &= 1 \times 7^3 + 1 \times 7^2 + 6 \times 7^1 + 1 \times 7^0 \\ &= 1 \times 343 + 1 \times 49 + 6 \times 7 + 1 \\ &= 435_{10}. \end{aligned} \tag{1.3}$$

Therefore, 1161_7 is another way of representing the number 435_{10}.

Generally speaking, any integer N can be represented in any base-b with $b \geq 2$

$$N = p_{n-1} \times b^{n-1} + p_{n-2} \times b^{n-2} + \cdots + p_1 \times b^1 + p_0 \times b^0 \tag{1.4}$$

or

$$N = \sum_{j=0}^{n-1} p_j \times b^j, \tag{1.5}$$

where p_j is the digit which represents the weight of the jth power of the base. The number of digits necessary is b and it is usual that the digits are the integers between 0 and $b - 1$

$$p_j \in \{0, 1, \ldots, b-1\}. \tag{1.6}$$

Thus, to represent numbers in base-b, digits of a value equal to or greater than b cannot be used. For example, the representation of a number in base-7 cannot use the digit 7 nor any other digit greater than 7. The sequence of digits 1742_7 is therefore not a valid representation of a number.

The *conversion* of the representation of a number in base-b to a representation in base-10 is not difficult, as illustrated by Equation (1.3). The reverse, converting a number represented in base-10 to its representation in base-b requires a little more work, but it is also simple. One of the most common methods is the *method of successive divisions*. As an example, consider a number N represented in base-b, as shown in Equation (1.4). If the number is divided by b, this results in

$$\begin{aligned}\frac{N}{b} &= (p_{n-1} \times b^{n-2} + p_{n-2} \times b^{n-3} + \cdots + p_1 \times b^0) + \frac{p_0 \times b^0}{b} \\ &= (p_{n-1} \times b^{n-2} + p_{n-2} \times b^{n-3} + \cdots + p_1 \times b^0) + \frac{p_0}{b},\end{aligned} \tag{1.7}$$

where p_0 is the rest of the division of N by b (remember that $p_0 < b$). In this way, the digit p_0 can be identified.

Repeating the previous procedure for the number $\frac{N}{b}$ will enable us to derive p_1 and through the successive application of the procedure, all the digits that represent the number.

As an example, consider obtaining the representation of the number 273_{10} in base-5

$$\frac{273}{5} = 54 + \frac{3}{5}. \tag{1.8}$$

In the same way, the following can be obtained:

$$\begin{aligned}\frac{54}{5} &= 10 + \frac{4}{5},\\ \frac{10}{5} &= 2 + \frac{0}{5},\\ \frac{2}{5} &= 0 + \frac{2}{5}.\end{aligned} \tag{1.9}$$

From Equations (1.8) and (1.9) it is easy to obtain $p_0 = 3$, $p_1 = 4$, $p_2 = 0$ e $p_3 = 2$. From which,

$$273_{10} = 2043_5. \tag{1.10}$$

1.1.2 *Representation of Non-signed Integers in Base-2*

The representation of numbers in *base-2* is important because, in computers and other digital systems, the representation of numbers has to be based on a set of two different values for some physical quantity. In digital computers, that physical quantity is usually the voltage between two points of an electronic circuit.

To represent an integer in base-2, two digits are required, usually designated by 0 and 1. Just as with other bases, an integer is, therefore, represented by a sequence of digits, in this case, *binary digits* or *bits*. Table 1.1 shows the integers from 0 to 15 represented in base-2.

For example, 110101_2 is a number represented in base-2 or, as it is also said, represented in *binary*.

The previous section presented a method to take a number in binary (or in any other base) and obtain the representation of the same number in base-10, the base we normally use.

Table 1.1 Representation of integers from 0 to 15 in base-2.

Base-10	Base-2	Base-10	Base-2
0	0	8	1000
1	1	9	1001
2	10	10	1010
3	11	11	1011
4	100	12	1100
5	101	13	1101
6	110	14	1110
7	111	15	1111

The technique used consisted in expressing the representation of the number in terms of weighted sums of the powers of the base and calculating the value of the number in base-10. This is a general technique, which can therefore also be applied for numbers represented in base-2. In the case of the number 110101_2 mentioned above, given that this is a 6 digit number, then

$$\begin{aligned} 110101_2 &= 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 32 + 16 + 4 + 1 \\ &= 53_{10}. \end{aligned} \tag{1.11}$$

The inverse problem, the determination of the representation of a number in base-2 (or in any other base-b), given its representation in base-10 was also dealt with, by presenting the method of successive divisions. Now, using the number 23_{10} as an example, note that this number can be rewritten as

$$23 = 11 \times 2 + 1, \tag{1.12}$$

which makes explicit the quotient and the remainder of the division of the number by 2.

Now, the number 11 can be represented as $11 = 5 \times 2 + 1$, so that substituting this in Equation (1.12), the following is obtained:

$$\begin{aligned} 23 &= (5 \times 2 + 1) \times 2 + 1 \\ &= 5 \times 2^2 + 1 \times 2 + 1. \end{aligned} \tag{1.13}$$

Now, given that $5 = 2 \times 2 + 1$, it is the case that

$$\begin{aligned} 23 &= (2 \times 2 + 1) \times 2^2 + 1 \times 2 + 1 \\ &= 2 \times 2^3 + 1 \times 2^2 + 1 \times 2 + 1. \end{aligned} \tag{1.14}$$

And, as $2 = 1 \times 2 + 0$,

$$\begin{aligned} 23 &= (1 \times 2 + 0) \times 2^3 + 1 \times 2^2 + 1 \times 2 + 1 \\ &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2 + 1. \end{aligned} \tag{1.15}$$

Finally, writing out all the powers of 2, we obtain

$$23 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0. \tag{1.16}$$

It is now easy to see that the representation of number 23_{10} in base-2 is 10111_2.

The binary digits of the number are, as seen above, the successive remainders of the division by 2 of the initial number and the successive quotients. The most common (and fastest) way of carrying out the calculation, however, is the successive application of the usual algorithm for performing division and collecting the various remainders at the end.

2	23	Remainders	
2	11	**1**	Least significant bit
2	5	**1**	
2	2	**1**	
2	1	**0**	
	0	**1**	Most significant bit

The digit with the greatest weight corresponds to the remainder of the last division and so on until the digit with the least weight, which is the remainder of the first division.

1.1.3 *Representation of Fractional Numbers in Base-2*

The *representation of fractional numbers* in base-2 (or any other), does not present any problems, using the same method used for integers. A fractional number may have an integer part and a decimal part, that is, with a value less than 1. That decimal part, with n digits, is representable by the following expression:

$$N = p_{-1} \times b^{-1} + p_{-2} \times b^{-2} + \cdots + p_{-n} \times b^{-n}, \tag{1.17}$$

or

$$N = \sum_{i=-1}^{-n} p_i \times b^i. \tag{1.18}$$

Consider the number 0.1011010_2 as an example. The representation, as in the case of integer numbers, gives a direct method to convert the binary fractional number to decimal

$$\begin{aligned} 0.1011010_2 &= 1 \times 2^{-1} + 1 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-6} \\ &= 0.5 + 0.125 + 0.0625 + 0.015625 \\ &= 0.703125_{10}. \end{aligned} \tag{1.19}$$

The conversion of a fractional number between base-b and base-10 is made, therefore, using the same algorithm used for integers, taking care not add more accuracy than was present in the original number through the conversion process, since this would have no meaning. Note that any non-integer number can have an infinitely long representation in a different base.

In fact, the number 0.1011010_2 is represented by 7 binary digits after the decimal point. This means that it is represented with a *precision* of 1 in 2^7 since, with these 7 digits, it is possible to represent 2^7 different numbers. The number obtained through base conversion represents the same measurable quantity and, thus, the number of digits must be such that the accuracy does not exceed the original representation. We will, therefore, have to choose a number p of digits, such that p is the largest integer verifying the equation

$$10^p \leq 2^7 \tag{1.20}$$

and, therefore,

$$p = \lfloor \log_{10} 2^7 \rfloor. \tag{1.21}$$

From Expression (1.21),[1] it can be seen that $p = 2$. Therefore, the correct representation in base-10 of the number 0.1011010_2 is 0.70_{10}, which is obtained by rounding the result of Expression (1.19).

The inverse problem of converting a fractional number represented in base-10 to any base, particularly base-2, uses, just as in the case of integers, a more sophisticated algorithm. The procedure used is shown below, using as an example the number 0.627_{10}. This number, when represented in base-2, will be

$$0.627_{10} = 0.p_{-1}p_{-2}p_{-3}p_{-4} \cdots p_{-n2}. \tag{1.22}$$

The objective is to obtain the value of the digits $p_{-1}, p_{-2}, p_{-3}, p_{-4} \cdots p_{-n}$. Multiplying both sides of Equation (1.22) by 2 gives

$$1.254_{10} = p_{-1}.p_{-2}p_{-3}p_{-4} \cdots p_{-n2}. \tag{1.23}$$

In fact, multiplying by 2 in base-2 results in moving all the digits one position to the left, like the multiplication by 10 in base-10. Analysing

[1]The function $\lfloor x \rfloor$ (floor) returns, for the real number x, the largest integer less or equal to x.

Equation (1.23) and noting that the integer parts of the two sides of the equation must be the same, as must the fractional parts, we obtain

$$p_{-1} = 1 \tag{1.24}$$

and

$$0.254_{10} = 0.p_{-2}p_{-3}p_{-4} \ldots p_{-n2}. \tag{1.25}$$

Equation (1.24) allows the identification of the digit p_{-1}. For the second digit, p_{-2}, the same algorithm is now applied to the resulting fractional number, represented in Expression (1.25). The successive use of multiplication by 2 gives the sequence of digits for the fractional part. It is clear that, just as in the inverse conversion, we should use the maximum number of digits, in the new base which does not increase the accuracy of the number compared to its original representation. In the above example, to not exceed the initial accuracy (1 in 10^3), 9 digits will be used to represent this in base-2 ($2^9 = 512$ and $2^{10} = 1\,024$). Completing the example

$$0.627_{10} = 0.101000001_2. \tag{1.26}$$

When the aim is to convert the representation of numbers with an integer and a fractional part between two bases, the conversion of the integer part and the fractional part is performed using the respective algorithms and the results added together at the end.

1.1.4 *Representation of Numbers in Bases Powers of* 2

The representation of numbers in base-2 is the one used by digital systems to represent numbers internally, but it has the major disadvantage of using relatively long sequences of digits. For example, 153.845_{10} is represented in base-2 by 10011001.110110001_2. These representations become difficult to remember and manipulate. Representation in base-10 does not have these drawbacks, but it cannot be used as the internal representation in digital systems. The solution to this dilemma is the use of condensed forms of binary representation, which become possible by representing numbers in *bases which are a power of* 2, that is, $4, 8, 16, \ldots$ Base-8 is commonly used or, even more frequently, base-16. As will be seen, these representations provide an abbreviated and readily convertible representation of binary numbers with many digits.

Base-8 uses digits from 0 to 7, and the conversion of numbers between base-8 and base-10, naturally, uses the procedures described above.

A number represented in base-8 is also said to be represented in *octal*. The representation of numbers in base-16, or in *hexadecimal*, as this type of representation is commonly designated, is similar to any other base, but it is necessary to take into account that there are 16 digits, from 0 to 15. For the digits which represent $10, 11, 12, 13, 14$ and 15, uppercase letters A to F are normally used. The 16 digits are listed in Table 1.2.

The number $4\text{A}6\text{F}_{16}$ is therefore represented in decimal as follows:

$$\begin{aligned}4\text{A}6\text{F}_{16} &= 4 \times 16^3 + 10 \times 16^2 + 6 \times 16 + 15\\ &= 4 \times 4096 + 10 \times 256 + 6 \times 16 + 15\\ &= 19\,055_{10}.\end{aligned} \tag{1.27}$$

Similarly, in base-8 the number 3605_8 is

$$\begin{aligned}3605_8 &= 3 \times 8^3 + 6 \times 8^2 + 5\\ &= 3 \times 512 + 6 \times 64 + 5\\ &= 1\,925_{10}.\end{aligned} \tag{1.28}$$

Conversion between bases in which one is a power of the other is carried out in an extremely easy way. Consider, in some detail, the conversion of the number 101101110101_2 to base-16. The first step is to represent the number, expressing it in terms of the powers of the base, similarly to what was carried out before

$$\begin{aligned}101101110101_2 = 1 \times 2^{11} &+ 0 \times 2^{10} + 1 \times 2^9 + 1 \times 2^8 + 0 \times 2^7\\ &+ 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2\\ &+ 0 \times 2^1 + 1 \times 2^0.\end{aligned} \tag{1.29}$$

Table 1.2 Digits in base-16.

Value	Digit	Value	Digit
0	0	8	8
1	1	9	9
2	2	10	A
3	3	11	B
4	4	12	C
5	5	13	D
6	6	14	E
7	7	15	F

Then, the terms are put into groups of 4 starting from the least significant

$$\begin{aligned}101101110101_2 &= (1\times 2^{11} + 0\times 2^{10} + 1\times 2^{9} + 1\times 2^{8})\\ &\quad + (0\times 2^{7} + 1\times 2^{6} + 1\times 2^{5} + 1\times 2^{4})\\ &\quad + (0\times 2^{3} + 1\times 2^{2} + 0\times 2^{1} + 1\times 2^{0}). \qquad (1.30)\end{aligned}$$

In each group, we factor out the powers of 2 necessary to have each group represented in terms of the weighted sum of the powers of order $0, 1, 2$ and 3.

$$\begin{aligned}101101110101_2 &= (1\times 2^{3} + 0\times 2^{2} + 1\times 2^{1} + 1\times 2^{0})\times 2^{8}\\ &\quad + (0\times 2^{3} + 1\times 2^{2} + 1\times 2^{1} + 1\times 2^{0})\times 2^{4}\\ &\quad + (0\times 2^{3} + 1\times 2^{2} + 0\times 2^{1} + 1\times 2^{0})\times 2^{0}. \qquad (1.31)\end{aligned}$$

However, $2^8 = 16^2, 2^4 = 16^1$ and $2^0 = 16^0$ are powers of 16. This can, therefore, be written,

$$101101110101_2 = 11\times 16^2 + 7\times 16^1 + 5\times 16^0 \qquad (1.32)$$

and so, it is now easy to conclude that

$$101101110101_2 = \mathrm{B}75_{16}. \qquad (1.33)$$

Note that each digit in base-16 was defined by the 4 binary digits which were initially grouped. So, if the equivalence between the digits of base-16 and its correspondence to a binary four-digit number is known, it is possible to directly determine the number in base-16 from the number in base-2. This correspondence can be easily obtained and is shown in Table 1.3.

Table 1.3 Correspondence between digits of base-16 and their representation in binary using 4 digits.

Binary	Digit	Binary	Digit
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

If the groups of four binary digits are isolated in the number $1011\ 0111\ 0101_2$, it can, therefore, be directly converted to base-16

$$\begin{array}{ccc} 1011 & \Leftrightarrow & \mathrm{B} \\ 0111 & \Leftrightarrow & 7 \\ 0101 & \Leftrightarrow & 5, \end{array}$$

obtaining the number $\mathrm{B75}_{16}$. It should be noted that the four-digit groups are formed from the least significant digit, i.e., from the right. This is a consequence of the way the algorithm was defined.

For a binary number with a number of bits that is not a multiple of 4, the method is applied in the same way. Take the number 1101011011_2. Separating the digits into groups of 4 from the least significant gives

$$\begin{aligned} 1101011011_2 &= 11\ 0101\ 1011_2 \\ &= 3\ 5\ \mathrm{B}_{16} \\ &= 35\mathrm{B}_{16}. \end{aligned} \tag{1.34}$$

For another base which is a power of 2, such as base-8, the method is similar, with a variation only in the number of digits to be grouped together. For base-8, the digits are grouped into groups of 3 ($2^3 = 8$). As an example

$$\begin{aligned} 10010110101001_2 &= 10\ 010\ 110\ 101\ 001_2 \\ &= 2\ 2\ 6\ 5\ 1_8 \\ &= 22651_8. \end{aligned} \tag{1.35}$$

In the same way, going from a base which is a power of 2 to base-2 is carried out using the inverse process

$$\begin{aligned} 7\mathrm{DA3F}_{16} &= 7\ \mathrm{D}\ \mathrm{A}\ 3\ \mathrm{F}_{16} \\ &= 0111\ 1101\ 1010\ 0011\ 1111_2 \\ &= 01111101101000111111_2. \end{aligned} \tag{1.36}$$

A similar example may be considered for base-8

$$\begin{aligned} 3461_8 &= 3\ 4\ 6\ 1_8 \\ &= 011\ 100\ 110\ 001_2 \\ &= 11100110001_2. \end{aligned} \tag{1.37}$$

An alternative way, which is often used, of signalling that a number is represented in base-16 consists in ending the number with the letter *h*.

Therefore, writing 4871_{16} is the same as writing 4871h, for example. In the same way, it is usual to use the letters b, o and d to indicate that the number is represented in binary, octal or decimal, respectively. For example, 1110101_2 may be represented by 1110101b. This form of representation will, from now on, be the preferred way when it is necessary to indicate the base in which a number is represented.

Representing numbers in bases which are powers of 2, particularly base-8 and base-16, thus offers a compact representation of binary numbers with many digits. In addition, as has been seen, there is a very simple way of converting numbers between base-2 and those bases, which is practically independent in complexity, of the number of digits of the representation of the numbers. Finally, this allows for partial conversions of sections of the number which may be of interest. It is easy to discover, for example, the least or most significant binary digits of a number represented in these bases without the need for its complete conversion. For example, in the number A23Bh, it is easy to see that the five most significant binary digits are 10100.

The conversion of numbers with a fractional part between base-2 and bases that are powers of 2 is carried out in the same manner, taking care to group the binary digits starting at the decimal point. Take, for example, the number 1001010.101100111111b

$$\begin{aligned} 1001010.101100111111\text{b} &= 100\ 1010\,.\,1011\ 0011\ 1111\text{b} \\ &= 4\ \text{A}\,.\,\text{B}\ 3\ \text{Fh} \\ &= 4\text{A.B3Fh}. \end{aligned} \tag{1.38}$$

In the same way, the conversion from a base which is a power of 2 to base-2 does not present any difficulty for numbers with a fractional part

$$\begin{aligned} 5271.3527\text{o} &= 5\ 2\ 7\ 1\,.\,3\ 5\ 2\ 7\text{o} \\ &= 101\ 010\ 111\ 001\,.\,011\ 101\ 010\ 111\text{b} \\ &= 101010111001.011101010111\text{b}. \end{aligned} \tag{1.39}$$

1.2 Arithmetic Operations in Base-2, Base-8 and Base-16

This section briefly deals with simple *arithmetic operations* in base-2, base-8 and base-16. The scope is limited to the sums and products of positive integers. The study of arithmetic involving negative integers will be dealt with later, in Chapter 5, where subtraction will also be considered.

Table 1.4 Table of addition in base-2.

$X+Y$		Y	
		0	1
X	0	0	1
	1	1	10

1.2.1 *Sums in Base-2*

The *sum in base-2*, as with any other base, is not fundamentally different from a sum in base-10. The procedure adopted is based on the existence of a table of addition and a method of adding numbers digit by digit. The sum is carried out by adding, for each digit starting with the least significant digit, the digits of the numbers to be added to the *carry* of the previous digit. The sum should also create the carry to the next digit. The table of addition in base-2 (Table 1.4) is particularly simple.

It should be noted that, in the sum $1+1$, the result cannot be represented by a single digit, and it is necessary to use two digits. This means that, in the sum algorithm, there will be in this case a carry of 1. In the other cases of the table, the carry is always 0.

As an example, take the sum of 10001111_2 and 1011010_2. In the least significant digits, in the rightmost column, the sum does not have to consider the carry from the previous column. The sum of 1 and 0, according to Table 1.4, is 1, and the carry, 0 (Figure 1.1(a)).

In the second column, the sum of 1 and 1 is 10 (that is, 2_{10}). As the carry from the previous column is 0, the sum does not change. However, since 10 is a two-digit number, in this column, the result of the sum is 0, and the carry is 1 (Figure 1.1(b)).

In the next column, adding 1 and 0 gives 1. As the carry is 1, it is also necessary to add this 1, which gives 10. Therefore, the sum is 0, and the carry is 1 (Figure 1.1(c)).

$$\begin{array}{rl} 10001111 & \\ +1011010 & \\ \hline 1 & \\ 0 & \text{carry} \end{array} \qquad \begin{array}{rl} 10001111 & \\ +1011010 & \\ \hline 01 & \\ 10 & \text{carry} \end{array} \qquad \begin{array}{rl} 10001111 & \\ +1011010 & \\ \hline 001 & \\ 110 & \text{carry} \end{array}$$

(a) (b) (c)

Fig. 1.1 Process for executing a sum.

$$\begin{array}{rl}
10001111 & \\
+1011010 & \\
\hline
1001 & \\
1110 & \text{carry}
\end{array}
\qquad\qquad
\begin{array}{rl}
10001111 & \\
+1011010 & \\
\hline
11101001 & \\
00011110 & \text{carry}
\end{array}$$

(a) (b)

Fig. 1.2 Process for executing a sum (continuation of Figure 1.1).

Table 1.5 Multiplication table in base-2.

$X \times Y$		Y	
		0	1
X	0	0	0
	1	0	1

In the fourth column from the right, there is the sum of 1 and 1, which gives 10, and also a carry of 1, which gives 11. In this case, the sum is 1, and the carry is also 1 (Figure 1.2(a)).

Continuing this reasoning, the final result is shown in Figure 1.2(b).

It is, of course, possible to sum more than two numbers The only aspect to be taken into consideration is that the carry may not be just one digit. Take the sum of the 4 numbers given below.

In the first column, the sum is 100_2. Here, the sum is, of course, 0, and the carry, 10. Taking this aspect into consideration, the final result will be:

$$\begin{array}{r}
10011101 \\
101011 \\
11001 \\
+\ 1011011 \\
\hline
100111100
\end{array}$$

In digital systems, sums are normally made between two numbers, with the sums of more than two numbers made by successive sums of the various terms to be summed.

1.2.2 *Multiplications in Base-2*

Just as with addition, *multiplication in base-2* follows the same methods for multiplication in base-10. The multiplication table is shown in Table 1.5.

Each digit of the multiplier is multiplied by the multiplicand to generate a partial product. The sum of all the partial products is the product of the two numbers. In fact, the product $M \times N$ may be expressed by representing the multiplier N as the sum of the powers of the base weighed by the correspondent digits (the same can be done for any base) using Equation (1.40).

$$\begin{aligned} M \times N &= M \times (p_{n-1} \times 2^{n-1} + p_{n-2} \times 2^{n-2} + \cdots + p_1 \times 2^1 + p_0 \times 2^0) \\ &= M \times p_{n-1} \times 2^{n-1} + M \times p_{n-2} \times 2^{n-2} + \cdots + M \times p_1 \times 2^1 \\ &\quad + M \times p_0 \times 2^0. \end{aligned} \tag{1.40}$$

For example, when multiplying $M = 10001111$b by $N = 1010$b, the product is obtained by adding the four partial products

$$\begin{array}{l} 10001111 \times 1000 \\ 10001111 \times 0 \\ 10001111 \times 10 \\ 10001111 \times 0, \end{array}$$

corresponding to the four digits of the multiplier.

In the usual algorithm, it is, therefore, essential to place the various partial products aligned with the respective digit of the multiplier.

$$\begin{array}{rl} 10001111 & \\ \times\ 1010 & \\ \hline 00000000 & (= 10001111 \times 0) \\ 10001111 & (= 10001111 \times 10) \\ 00000000 & (= 10001111 \times 0) \\ 10001111 & (= 10001111 \times 1000). \\ \hline 10110010110 & \end{array}$$

Naturally, just as in the case of multiplication in base-10, the results of the product of the multiplicand by digits of the multiplier that are 0 can be omitted

$$\begin{array}{r} 10001111 \\ \times\ 1010 \\ \hline 10001111 \\ 10001111 \\ \hline 10110010110 \end{array} .$$

It should be noted that, in base-2, there are only two hypotheses for the partial products: either the multiplier digit is 0, and the partial product is 0, or that digit is 1, and the partial product is equal to the multiplicand with the displacement corresponding to the weighting of the digit. It will be seen later that this characteristic is important when digital systems carry out this operation.

1.2.3 *Arithmetic Operations in Other Bases*

Performing arithmetic operations between numbers represented on other bases do not raise any problems besides knowing the tables of operations for those bases. Using as an example the *sum in base-16*, the table of sums can be constructed (Table 1.6).

This table can be constructed in various ways. One possibility is to consider the value of the digits for each sum, sum them in decimal and transfer the result to hexadecimal. As an example, the sum 5h + Dh, is the sum of 5d and 13d (the decimal value of digit Dh). The result of this sum is 18d or, in hexadecimal, 12h, which is the value to be placed in the table.

As an example, take the sum of 1F3A5h and A542h:

$$\begin{array}{rl} 1F3A5 & \\ +A542 & \\ \hline 298E7 & \\ 01000 & \text{carry.} \end{array}$$

Multiplication is also easy to carry out. This will be shown using the *multiplication table in base-8* (Table 1.7) and the product of two numbers in that base.

Table 1.6 Table of the sum in base-16.

$X+Y$		Y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
	2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
	3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
	4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
	5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
	6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
X	7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
	8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
	9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
	A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
	B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
	C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
	D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
	E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
	F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Table 1.7 Table of multiplication in base-8.

$X \times Y$		Y							
		0	1	2	3	4	5	6	7
X	0	0	0	0	0	0	0	0	0
	1	0	1	2	3	4	5	6	7
	2	0	2	4	6	10	12	14	16
	3	0	3	6	11	14	17	22	25
	4	0	4	10	14	20	24	30	34
	5	0	5	12	17	24	31	36	43
	6	0	6	14	22	30	36	44	52
	7	0	7	16	25	34	43	52	61

The product of 1537o and 314o is

$$\begin{array}{r}
1537 \\
\times\ 314 \\
\hline
6574 \\
1537 \\
5035 \\
\hline
527664
\end{array} .$$

The multiplication operation in base-b, although not conceptually different from the operation in base-10, requires a good knowledge of the respective multiplication table, which, in general, is unavailable. In these circumstances, it is clear that carrying out multiplications directly in these bases is a difficult task and is not frequent. The sum, however, is very useful, as will be seen later, in the context of computer architecture and programming.

1.3 Codes

Representation in base-2 allows for the representation of numbers in digital systems. However, not all information is numeric. There are many other types of data that need to be stored and processed. Text is an obvious example, but it is far from being the only one. This section will analyse the methods to represent information using codes. Particular emphasis will be given to binary codes, BCD and alphanumeric codes. The representation of other types of information is outside the scope of this book.

1.3.1 *Coding*

The *representation of information in digital systems* is based on the fact that systems of this type are supported in quantities that can only assume two values: 0 and 1. As is the case with binary digits, an entity which can take on two values is called a bit. If there is an entity which generates information and that information is created in sequences of symbols with several possible values, the solution for the representation of those values in a digital system is to *encode* them. This means that each possible value is made to correspond to a specified combination of bits which then represent that value. Take as an example an elevator in a building with six floors: two basements, the ground floor and another three floors. If the goal is to register the floor where the elevator is in a digital system, or where it is heading if moving, there is the need to codify that information, i.e., make each floor correspond to a particular bit pattern. The number of bits to be used will be at least those necessary to produce six combinations. Three bits are enough for that code. A coding example is shown in Table 1.8.

The correspondence between the entities to be represented and their coding is called *code*. In this way, the previous table sets out a code. Each of the configurations is referred as a *code word*. If the number of bits for the code words is equal for all configurations, it is known as the *code length*. This code is, therefore, a code of length 3. Obviously, this is not the only possible code for this application. Table 1.9 shows another possibility.

The only restriction for a valid code is that it does not have repeated encodings, which means that two different entities may not be encoded with the same code word.

Table 1.8 Sample code.

Floor	Coding
2nd basement	000
1st basement	001
Ground floor	010
1st floor	011
2nd floor	100
3rd floor	101

Table 1.9 Alternative sample code.

Floor	Coding
2nd basement	110
1st basement	111
Main floor	000
1st floor	001
2nd floor	010
3rd floor	011

Table 1.10 Sample code with restrictions.

Floor	Coding
2nd basement	0011
1st basement	0101
Main floor	1001
1st floor	0110
2nd floor	1010
3rd floor	1100

Either of this codings is acceptable. The two codes shown have no feature which significantly distinguishes them from one another. However, codes can be designed with some particular characteristics. For example, using words of 4 bits to represent the various floors, it is possible to design a code that represents each floor as a sequence of 2 bits with the value 0 and 2 bits with value 1, as shown in Table 1.10.

A possible use for codes of this type is to *detect errors* which could happen in the coding process or as a consequence of a malfunction of a system. If at any given time, the system has registered that the elevator is on a floor represented by the word 0010, it is immediately clear there was an error, since the word 0010 does not belong to the code.

1.3.2 *Numeric Codes*

Although numbers are generally stored and processed using their representation in base-2 (and, in a way, this also corresponds to a code), it is often necessary to use codes to represent numbers, particularly integers. These codes are known as *numeric codes*. The easiest way to represent integers by a code is to have each number represented by a code word which is its binary representation. Table 1.11 shows a code of this type with words of constant length.

It should be noted that all the code words have the same length. For example, the number 3 is represented by 00011 and not by 11, which would be its normal representation in base-2. This type of code in which each number is coded by its representation in base-2 with a fixed number of bits, is known as *natural binary code*. Obviously, there are natural binary codes for any number of bits.

A very common situation of a different kind is the need to encode decimal digits. It is sometimes convenient to represent a number not by its natural binary code, but rather by the binary coding of each of its digits in base-10. An obvious situation is that of the display of a calculator which represents a number in decimal for easy reading by the user, but which, internally, is

Table 1.11 Natural binary code with 5 bits.

Number	Coding	Number	Coding
0	00000	16	10000
1	00001	17	10001
2	00010	18	10010
3	00011	19	10011
4	00100	20	10100
5	00101	21	10101
6	00110	22	10110
7	00111	23	10111
8	01000	24	11000
9	01001	25	11001
10	01010	26	11010
11	01011	27	11011
12	01100	28	11100
13	01101	29	11101
14	01110	30	11110
15	01111	31	11111

Table 1.12 BCD code.

Digit	Coding
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

represented in binary. The classic solution is to represent each decimal digit of the number using its binary representation and a fixed length, as shown in Table 1.12.

This code is called *BCD code*. BCD stands for *Binary Coded Decimal*. It should be noted that the BCD code for each digit has the same representation as the natural binary code representation with 4 bits. However, not all the configurations have a meaning. The indication that a number is represented in BCD is made by writing BCD in subscript. For example, 7_{10} is represented by 0111_{BCD} in BCD.

The representation of a number in BCD code is made by representing each decimal digit by its binary representation, as shown in the representation of the number 2 719 in BCD

$$2\,719_{10} = 0010011100011001_{\mathrm{BCD}}, \tag{1.41}$$

in which 0010 corresponds to the digit 2_{10}; 0111, to 7_{10}; 0001, to 1_{10}; and 1001, to 9_{10}.

It should be noted that not every sequence of bits may correspond to a coding using BCD code. For example, the sequence 10110101 is not a number encoded in BCD, since the 4 bits on the left (1011) do not correspond to any digit, as can be seen in Table 1.12.

In the same way, it is important to mention that a representation of a number in BCD code is not a representation of this number in binary. Returning to the previous example,

$$0010011100011001_{\text{BCD}} = 2\,719_{10}, \tag{1.42}$$

although the same sequence of bits interpreted as a number in base-2 has another meaning

$$0010011100011001_{2} = 10\,009_{10}, \tag{1.43}$$

which, clearly, does not correspond to the same number.

1.3.3 *Reflected Codes*

A particularly important type of numeric codes are the *reflected codes* (sometimes referred to as *Gray code*). The fundamental characteristic of these codes is that two words which code consecutive numbers differ in only one bit. This feature is important, as will become clear in Chapter 2.

Table 1.13 shows a three-bit reflected code. It should be noted that with 3 bits, it is possible to encode 8 different numbers.

As can be easily seen, in this code each number is not, except in some cases, represented by its binary representation. It is possible to verify that between each two successive numbers only one bit of its representation is

Table 1.13 Three-bit reflected code.

Number	Coding
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Table 1.14 Construction of 4 bit reflected code. (a) Reflection and (b) additional bit.

000
001
011
010
110
111
101
100
100
101
111
110
010
011
001
000

(a)

Number	Coding
0	**0**000
1	**0**001
2	**0**011
3	**0**010
4	**0**110
5	**0**111
6	**0**101
7	**0**100
8	**1**100
9	**1**101
10	**1**111
11	**1**110
12	**1**010
13	**1**011
14	**1**001
15	**1**000

(b)

altered. This is also true with the representations of 0 and 7, the two extremes represented.

The most direct way to build an n bit reflected code is to start from a $n-1$ bit code. Start with 2^{n-1} configurations of the $n-1$ bit code and repeat them in an inverse manner, i.e., as if reflected in an imaginary mirror (hence the name reflected code), as shown in Table 1.14(a).

The fourth bit of the coding is then added, making it equal to 0, in the initial positions, and equal to 1, in the "reflected" positions, as shown in Table 1.14(b), where the code is represented.

A suggested exercise would be to obtain the reflected codes of 2 and 3 bits, from the one-bit code. Clearly, the latter does not differ from 1 bit natural binary code.

1.3.4 *Alphanumeric Codes*

One type of information that it is important to represent in digital systems is text. Therefore, it is essential to use a code to represent all the possible characters in a text. These codes are called *alphanumeric codes*. Throughout the evolution of digital systems, various codes with this purpose have been utilised. The code most commonly used is the *ASCII code* (American Standard Code for Information Interchange), which originated in

Table 1.15 ASCII code.

$b_6b_5b_4$	000	001	010	011	100	101	110	111
$b_3b_2b_1b_0$								
0000	NUL	DLE	SP	0	@	P	‘	p
0001	STH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	ˆ	n	˜
1111	SI	US	/	?	O	_	o	DEL

the United States, quickly became a universally accepted code. A code of this type has to encode the 26 letters of the alphabet, both uppercase and lowercase, the 10 digits, all punctuation signs and some special characters. The code has words of 7 bits, the number of bits necessary to encode all the characters. It is usual to represent the code word with the bits ordered from 1 to 7 in the following way: $b_6b_5b_4b_3b_2b_1b_0$. The code is shown in Table 1.15.

The character ≪SP≫ is the space.

The characters in the first two columns and the ≪DEL≫ in the last position of the table are *control characters*. The meaning of the control characters is mostly connected to the applications of the code in transmitting information between devices. In many cases, these characters are not used anymore due to the evolution of communication protocols.

The representation of a text in ASCII is done by indicating the succession of codes of the text characters. For example, the text ≪Digital Systems≫ is encoded in ASCII as 1000100b 1101001b 1100111b 1101001b 1110100b 1100001b 1101100b 0100000b 1010011b 1111001b 1110011b 1110100b 1100101b 1101101b 1110011b .

It is more convenient to represent the ASCII code configurations in hexadecimal. In this case, the text above would, therefore, be codified as: 44h 69h 67h 69h 74h 61h 6Ch 20h 53h 79h 73h 74h 65h 6Dh 73h.

The ASCII sequence 44h 69h 67h 69h 74h 61h 6Ch 20h 73h 79h 73h 74h 65h 6Dh73h 20h 75h 73h 65h 20h 41h 53h 43h 49h 49h 20h 74h 6Fh 20h

72h 65h 70h 72h 65h 73h 65h 6Eh 74h 20h 63h 68h 61h 72h 61h 63h 74h 65h 72h 73h 2Eh codifies the following phrase: ≪Digital systems use ASCII to represent characters.≫

Sometimes the designation of the characters in ASCII code is carried out in decimal instead of hexadecimal. For example, the letter ≪A≫, which has the hexadecimal code 41h, has the decimal code 65d.

ASCII code enables representing a text, but with some limitations. In one hand, it does not allow the special characters of a specific language to be represented, for example, ≪ç≫ in Portuguese, ≪ø≫ in Swedish or ≪ß≫ in German. Furthermore, it does not code accentuated characters nor can it represent other alphabets. As such, various extensions have been developed to represent specific characters and accentuated characters. The extension is done by changing the code to an eight-bit code, while keeping it compatible with the basic ASCII. Unfortunately, there are currently several mutually incompatible codes. In all of them, however, the representation of the 00h to 7Fh values are the same as in classic ASCII, to maintain compatibility between these extensions and ASCII. For example, the *ISO-8859-1 code*, the ISO (International Standard Organisation) standardised extension for Western European alphabets, has the correspondence for characters coded from 80h to FFh shown in Table 1.16.

Table 1.16 Code ISO-8859-1 (with $b_7 = 1$).

$b_7b_6b_5b_4$	1000	1001	1010	1011	1100	1101	1110	1111
$b_3b_2b_1b_0$								
0000			nbsp	∘	À	Đ	à	ð
0001			¡	±	Á	Ñ	á	ñ
0010			¢	^{2}	Â	Ò	â	ò
0011			£	^{3}	Ã	Ó	ã	ó
0100			€	´	Ä	Ô	ä	ô
0101			¥	μ	Å	Õ	å	õ
0110			|	¶	Æ	Ö	œ	ö
0111			§	·		×		÷
1000			¨	¸	È	Ø	è	ø
1001			©	^{1}	É	Ù	é	ù
1010					Ê	Ú	ê	ú
1011			≪	≫	Ë	Û	ë	û
1100			¬	$^1/_4$	Ì	Ü	ì	ü
1101			-	$^1/_2$	Í	Ý	í	ý
1110			®	$^3/_4$	Î	Þ	î	þ
1111			¯	¿	Ï	ß	ï	ÿ

The ≪nbsp≫ character is a special character preventing the breaking up of a line of text by a word processor or a browser.

The ISO-8859 standard contains other substandards which define codes for characters for other areas or linguistic groups.

UNICODE has since appeared, which seeks to code all the characters of every language. Each code word has 16 bits and, if necessary, uses a 20-bit extension. Normal ASCII and the ISO-8859-1 extension shown above are compatible with UNICODE. This code is recommended as a support for future Internet protocols.

1.4 Units of Information

From an abstract point of view, digital information can be represented using two values, usually designated by 0 and 1. As has already been mentioned, an entity of this type is called a bit. As is evident, a bit — which can only have one of two values — cannot, by itself, support the type of information mentioned above. To represent information it is usual to arrange the bits into larger capacity entities.

If we consider, for example, alphanumeric information represented using the ISO-8859-1 code (Table 1.16), each character, which is the minimum useful information unit, occupies 8 bits. Those 8 bits are a primary unit of information, if processing text, and should be processed, transferred or stored together. A set of 8 bits is called a *byte*, or *octet*.

A byte has this designation, independently of the meaning attributed to the information it carries. It is immaterial whether the byte represents a character, two digits codified in BCD, the colour of a pixel from the image on a computer monitor, or any other type of information. The concept of a byte is linked to the set of 8 bits interpreted as a coherent entity.

In the same way, a *nibble* is defined as a set of 4 bits, which is interpreted as a coherent unit. A BCD coded digit, for example, is represented by a nibble. The basic unit of information in an Intel 4004, one of the first microprocessors, which processed sets of 4 bits, was also a nibble.

Clearly, a set of two nibbles, if regarded as one unit, is a byte.

Another entity to consider at this stage is the *word*. A word is the smallest unit processed or stored in a system. For example, the Intel 4004 mentioned above is a processor with words of 4 bits. The Intel 8080 or the Motorola 6800, which process bytes, have words of 8 bits. The Intel 8086 or the Motorola

68000 or the P3 processor, presented later in this book, process words of 16 bits. Current computers use words of larger sizes. Unlike the concepts of byte or nibble, the concept of word is not linked to a fixed size. The number of bits of a word depends on the context being considered. However, it is common to consider a size of 16 bits when referring to a word, unless another size is explicitly specified.

No non-trivial amount of text, image, film or sound can be represented by a byte or a word. Useful digital systems applications, in the overwhelming majority of cases, need to store large quantities of information in sets of bytes or words.

When considering large amounts of objects or abstract entities, it is usual to refer to hundreds, thousands or millions of units. That reference assumes that the reference basis of the numbers utilised is base-10, where concepts such as thousand and million are natural since they are powers of the base. In digital systems, however, the normal base to represent numbers is base-2. In base-2, the sizes used are also usually powers of the base.

In base-10, as used in units of measurement systems, the representation of large numbers can be achieved using multiples of the units. For example, instead of speaking of 1 000 m, it is natural to talk about 1 km. The «k» here represents $1\,000\times$.

In base-2, 1 000d= 1111101000b. The philosophy at the basis of representation of multiples in base-10 may be applied to any base, but with different values. These values must be, just as in base-10, powers of the base. But it is convenient to have "useful" multiples in base-2, that is, multiples not too different from those which are useful in base-10 to maintain the intuition associated with the use of multiples.

The power of 2 closest to 1 000d is

$$2^{10} = 10000000000\text{b} = 1\,024\text{d}. \tag{1.44}$$

Therefore, it is conventional that in digital systems of all types (including computers), $1\text{K} = 2^{10} = 1\,024\text{d}$. Thus, 1 Kbyte (or KB) means exactly 1 024 bytes,[2] but, if you are willing to admit a small error, it may be considered as 1 thousand bytes.

The *multiples* normally used in digital systems are described in Table 1.17.

[2]Note that in this context the letters used are uppercase.

Table 1.17 Multiples in base-2.

Multiple	Power	Relation with lower multiple	Representation in base-10	Denomination
1 K	2^{10}		1 024d	Kilo
1 M	2^{20}	$= 2^{10}$ K	1 048 576d	Mega
1 G	2^{30}	$= 2^{10}$ M	1 073 741 824d	Giga
1 T	2^{40}	$= 2^{10}$ G	1 099 511 627 776d	Tera
1 P	2^{50}	$= 2^{10}$ T	1 125 899 906 842 624d	Peta

1.5 Summary

This chapter presents various ways of representing information in digital systems.

The first aspect dealt with is the binary representation of non-negative integers and fractional numbers, describing the conversion processes between the representation in binary and the representation in decimal. To facilitate the representation and manipulation of binary numbers with a large number of digits, octal and hexadecimal representations were also studied.

The chapter also described the use of the normal arithmetic rules for performing operations directly in binary, octal or hexadecimal.

The representation of non-numeric information content is also considered, with the introduction of the use of codes. Besides numeric and alphanumeric codes, reflected codes are also introduced. These codes are of great importance in the study of methods to simplify logic functions presented in the next chapter.

The chapter ends with some definitions related to basic aspects of information representation.

Exercises

1.1 Represent the following numbers in base-2:

(a) 33_{10}
(b) 162_{10}.

1.2 Consider the following numbers and represent them in base-2, taking care not to exceed the precision represented in the original numbers:

(a) 28.54_{10}
(b) 28.540_{10}
(c) 143.7_{10}
(d) 143.70_{10}.

1.3 Represent the following numbers in base-8, base-10 and base-16:

(a) 10101101011_2
(b) 110000001.1101_2
(c) 11111000.0011_2.

1.4 Represent the following numbers in base-2 and base-8:

(a) A57D_{16}
(b) CAFE_{16}
(c) 17D.A8_{16}.

1.5 Consider the number 35_x. It is known that this number is less than 64_{10}. Which values can x assume?

1.6 Numbers are usually represented in base-n, with n being positive. However, that is not strictly necessary. Take, for example, base-(-2). Nothing prevents the representation of numbers in that base. Represent the first 16 integers in base-(-2). Note that the number of bits necessary may be greater than four, the minimum number of bits necessary to represent 16 configurations.

1.7 Carry out the following operations in the bases indicated:

(a) $1001.11011_2 + 0.10101_2$
(b) $10010.11_2 \times 10.01_2$
(c) $2314.21_5 + 12.413_5$
(d) $24.1_5 \times 3.2_5$.

1.8 Suggest the algorithm for the subtraction in base-2.

1.9 Construct the addition and multiplication tables in base-16. Use them to carry out the following operations:

(a) $\text{A57D}_{16} + \text{CAFE}_{16}$
(b) $\text{A57D}_{16} \times 37_{16}$
(c) $\text{35A.7E}_{16} + \text{2D.1A6}_{16}$
(d) $\text{35A.7E}_{16} \times \text{7.F}_{16}$.

1.10 The conversion of numbers from one base-b to base-10, is, as has been seen, easy to carry out, by describing the number as a weighted sum (by its digits) of powers of the base and carrying out the operations thus made explicit. The operations are carried out in base-10. In the opposite way, conversion from base-10 to base-b, a different algorithm was presented. However, the situation is perfectly symmetrical, and the number in base-10 may be described as a weighted sum of powers of 10, with all the intervening numbers expressed in base-b. Carrying out the operations in base-b the representation of the number in that base can be obtained. Try using this method to represent in base-2 and base-5 the numbers 35_{10} and 572_{10} and confirm these by using the usual algorithms. Does the method also work with fractional numbers?

1.11 Starting from the usual algorithms for decimal division, obtain the algorithm for division in binary, carry it out and validate it (using the inverse operation), for the division of 1010110_2 by 101_2.

1.12 Consider the following sequence of bits: 100001110011

(a) If the sequence is a number in binary, what is the number represented?
(b) If the sequence is the representation of a number in BCD, what is the number represented?
(c) If the sequence is a number in base-3, what is the number represented?

1.13 Consider the numbers in Problem 1.1.

(a) Represent them in BCD code.
(b) Represent them in ASCII code.
(c) Represent them in ISO-Latin code (ISO-8859-1).

1.14 What is the phrase represented by the sequence of bytes in the ISO-8859-1 code represented below?
45h, 6Eh, 67h, 6Ch, 69h, 73h, 68h, 20h, 6Ch, 61h, 6Eh, 67h, 75h, 61h, 67h, 65h, 20h, 64h, 6Fh, 65h, 73h, 20h, 6Eh, 6Fh, 74h, 20h, 68h, 61h, 76h, 65h, 20h, 63h, 68h, 61h, 72h, 61h, 63h, 74h, 65h, 72h, 73h, 20h, 61h, 73h, 20h, C3h, A7h, 2Ch, 20h, C3h, A5h, 2Ch, 20h, C3h, 98h, 20h, 61h, 6Eh, 64h, 20h, C2h, A4h, 2Eh.

1.15 The names of airports are encoded by sequences of three capital letters. For example, the Changi Airport in Singapore has the code SIN, the El Prat Airport in Barcelona has the code BCN, and the John Kennedy Airport in New York has the code JFK.

(a) How many airports can be encoded in this way?
(b) How many bits will be necessary in ASCII code to code the airport codes in binary? And can a more efficient code be used to make it possible to encode only uppercase letters?
(c) How many bits would be necessary if a single binary code was attributed to each airport with a minimum number of bits?

1.16 A weighted code is a numerical code in which each bit of the code has a weight, and the encoded number is obtained by adding the weights multiplied by the value of the bit in each respective position. Natural binary code and BCD code are weighted codes with weights 8, 4, 2 and 1. Represent the ten decimal digits in the following weighted codes:

(a) 2, 4, 2, 1
(b) 5, 4, 2, 1
(c) 8, −4, 2, 1.