systems is also straightforward. Because it is easier to work with fewer digits than with a large number of bits, digital system users prefer to work with octal or hexadecimal systems when understanding or verifying the results produced by the system or communicating data between users or between users and the machine.

1.4 Arithmetic

Arithmetic in all other number systems follows the same general rules as in decimal. Binary arithmetic is simpler than decimal arithmetic since only two digits (0 and 1) are involved. Arithmetic in octal and hexadecimal systems requires some practice because of the general unfamiliarity with those systems. In this section, we will describe binary arithmetic in detail, followed by a brief discussion of octal and hexadecimal arithmetic. For simplicity, integers will be used in all the examples in this section. Nonetheless, the procedures are valid for fractions and numbers with both integer and fraction portions.

In the so-called *fixed-point representation* of binary numbers in digital systems, the radix point is assumed to be either at the right end or the left end of the field in which the number is represented. In the first case the number is an integer, and in the second it is a fraction. Fixed-point representation is the most common type of representation. In scientific computing applications, in which a large range of numbers must be represented, *floating-point representation* is used. Floating-point representation of numbers will be discussed in Section 1.6.

1.4.1 Binary Arithmetic

Table 1.3 illustrates the rules for binary addition, subtraction, and multiplication.

Table 1.3 Binary Arithmetic

(a) Addition

$A + B$		A	
		0	1
B	0	0	1
	1	1	10

Carry Sum

(b) Subtraction

$A - B$		A	
		0	1
B	0	0	1
	1	11	0

Borrow Difference

(c) Multiplication

$A \times B$		A	
		0	1
B	0	0	0
	1	0	1

Addition In Table 1.3(a), note that $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$, and $1 + 1 = 10$. Thus, the addition of two 1s results in a SUM of 0 and a CARRY of 1.

When two binary numbers are added, the carry from any position is included in the addition of bits in the next most significant position, as in decimal arithmetic. Example 1.16 illustrates this.

Example 1.16

$$\begin{array}{rccccl}
 & \boxed{4} & \boxed{3} & \boxed{2} & \boxed{1} & \boxed{0} \quad \text{bit position} \\
 & 0 & 1 & 1 & 0 & \quad \text{carry} \\
 & 1 & 0 & 1 & 1 & 0 \quad \text{augend} \\
+ & 0 & 0 & 1 & 1 & 1 \quad \text{addend} \\
\hline
 & 1 & 1 & 1 & 0 & 1 \quad \text{sum}
\end{array}$$

Note: The carry in the LSB position is 0.

Here, bits in the LSB position (i.e., position 0) are first added, resulting in a sum bit of 1 and a carry of 0. The carry is included in the addition of bits at position 1. The three bits in position 1 are added using two steps ($0 + 1 = 1$, $1 + 1 = 10$), resulting in a sum bit of 0 and a carry bit of 1 to the next most significant position (position 2). This process is continued through the most significant bit (MSB).

In general, the addition of two n-bit numbers results in a number that is $n + 1$ bits long. If the number representation is to be confined to n bits, the operands of the addition should be kept small enough so that their sum does not exceed n bits.

Subtraction From Table 1.3(b), we can see that $0 - 0 = 0$, $1 - 0 = 1$, $1 - 1 = 0$, and $0 - 1 = 1$ with a BORROW of 1. That is, subtracting a 1 from a 0 results in a 1 with a borrow from the next most significant position, as in decimal arithmetic. Subtraction of two binary numbers is performed stage by stage as in decimal arithmetic, starting from the LSB to the MSB. Some examples follow.

Example 1.17

$$\begin{array}{rccccccl}
 & \boxed{5} & \boxed{4} & \boxed{3} & \boxed{2} & \boxed{1} & \boxed{0} & \text{bit position} \\
 & 0 & & & 0 & & & \\
 & \cancel{1} & 0 & 1 & \cancel{1} & 0 & 1 & \text{minuend} \\
- & 0 & 1 & 1 & 0 & 1 & 0 & \text{subtrahend} \\
\hline
 & 0 & 1 & 0 & 0 & 1 & 1 & \text{difference}
\end{array}$$