There are 10 kinds of people in the world—those who understand binary and those who don't.

—Anonymous

CHAPTER 2 Data Representation in Computer Systems

2.1 INTRODUCTION

The organization of any computer depends considerably on how it represents numbers, characters, and control information. The converse is also true: Standards and conventions established over the years have determined certain aspects of computer organization. This chapter describes the various ways in which computers can store and manipulate numbers and characters. The ideas presented in the following sections form the basis for understanding the organization and function of all types of digital systems.

The most basic unit of information in a digital computer is called a **bit**, which is a contraction of **binary digit**. In the concrete sense, a bit is nothing more than a state of "on" or "off" (or "high" and "low") within a computer circuit. In 1964, the designers of the IBM System/360 mainframe computer established a convention of using groups of 8 bits as the basic unit of addressable computer storage. They called this collection of 8 bits a **byte**.

Computer **words** consist of two or more adjacent bytes that are sometimes addressed and almost always are manipulated collectively. The **word size** represents the data size that is handled most efficiently by a particular architecture. Words can be 16 bits, 32 bits, 64 bits, or any other size that makes sense within the context of a computer's organization (including sizes that are not multiples of eight). Eight-bit bytes can be divided into two 4-bit halves called **nibbles** (or **nybbles**). Because each bit of a byte has a value within a positional numbering system, the nibble containing the least-valued binary digit is called the low-order nibble, and the other half the high-order nibble.

2.2 POSITIONAL NUMBERING SYSTEMS

At some point during the middle of the sixteenth century, Europe embraced the decimal (or base 10) numbering system that the Arabs and Hindus had been using for nearly a millennium. Today, we take for granted that the number 243 means two hundreds, plus four tens, plus three units. Notwithstanding the fact that zero means "nothing," virtually everyone knows that there is a substantial difference between having 1 of something and having 10 of something.

The general idea behind positional numbering systems is that a numeric value is represented through increasing powers of a **radix** (or base). This is often referred to as a **weighted numbering system** because each position is weighted by a power of the radix.

The set of valid numerals for a positional numbering system is equal in size to the radix of that system. For example, there are 10 digits in the decimal system, 0 through 9, and 3 digits for the ternary (base 3) system, 0, 1, and 2. The largest valid number in a radix system is one smaller than the radix, so 8 is not a valid numeral in any radix system smaller than 9. To distinguish among numbers in different radices, we use the radix as a subscript, such as in 33_{10} to represent the decimal number 33. (In this book, numbers written without a subscript should be assumed to be decimal.) Any decimal integer can be expressed exactly in any other integral base system (see Example 2.1).

EXAMPLE 2.1 Three numbers represented as powers of a radix.

$$243.51_{10} = 2 \times 10^2 + 4 \times 10^1 + 3 \times 10^0 + 5 \times 10^{-1} + 1 \times 10^{-2}$$
$$212_3 = 2 \times 3^2 + 1 \times 3^1 + 2 \times 3^0 = 23_{10}$$
$$10110_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 22_{10}$$

The two most important radices in computer science are binary (base two), and hexadecimal (base 16). Another radix of interest is octal (base 8). The binary system uses only the digits 0 and 1; the octal system, 0 through 7. The hexadecimal system allows the digits 0 through 9 with A, B, C, D, E, and F being used to represent the numbers 10 through 15. Table 2.1 shows some of the radices.

2.3 DECIMAL TO BINARY CONVERSIONS

Gottfried Leibniz (1646–1716) was the first to generalize the idea of the (positional) decimal system to other bases. Being a deeply spiritual person, Leibniz attributed divine qualities to the binary system. He correlated the fact that any integer could be represented by a series of ones and zeros with the idea that God (1) created the universe out of nothing (0). Until the first binary digital computers were built in the late 1940s, this system remained nothing more than a mathemat-

Powers of 2
$2^{-2} = \frac{1}{4} = 0.25$
$2^{-1} = \frac{1}{2} = 0.5$
$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1{,}024$
$2^{15} = 32{,}768$
$2^{16} = 65{,}536$

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

TABLE 2.1 Some Numbers to Remember

ical curiosity. Today, it lies at the heart of virtually every electronic device that relies on digital controls.

Because of its simplicity, the binary numbering system translates easily into electronic circuitry. It is also easy for humans to understand. Experienced computer professionals can recognize smaller binary numbers (such as those shown in Table 2.1) at a glance. Converting larger values and fractions, however, usually requires a calculator or pencil and paper. Fortunately, the conversion techniques are easy to master with a little practice. We show a few of the simpler techniques in the sections that follow.

2.3.1 Converting Unsigned Whole Numbers

We begin with the base conversion of unsigned numbers. Conversion of signed numbers (numbers that can be positive or negative) is more complex, and it is important that you first understand the basic technique for conversion before continuing with signed numbers.

Conversion between base systems can be done by using either repeated subtraction or a division-remainder method. The subtraction method is cumbersome and requires a familiarity with the powers of the radix being used. Because it is the more intuitive of the two methods, however, we will explain it first.

As an example, let's say that we want to convert 104_{10} to base 3. We know that $3^4 = 81$ is the highest power of 3 that is less than 104, so our base 3 number will be 5 digits wide (one for each power of the radix: 0 through 4). We make note that 81 goes once into 104 and subtract, leaving a difference of 23. We know that the next power of 3, $3^3 = 27$, is too large to subtract, so we note the zero

"placeholder" and look for how many times $3^2 = 9$ divides 23. We see that it goes twice and subtract 18. We are left with 5 from which we subtract $3^1 = 3$, leaving 2, which is 2×3^0. These steps are shown in Example 2.2.

EXAMPLE 2.2 Convert 104_{10} to base 3 using subtraction.

$$\begin{array}{r} 104 \\ -81 \\ \hline 23 \end{array} = 3^4 \times 1$$

$$\begin{array}{r} -0 \\ \hline 23 \end{array} = 3^3 \times 0$$

$$\begin{array}{r} -18 \\ \hline 5 \end{array} = 3^2 \times 2$$

$$\begin{array}{r} -3 \\ \hline 2 \end{array} = 3^1 \times 1$$

$$\begin{array}{r} -2 \\ \hline 0 \end{array} = 3^0 \times 2$$

$$104_{10} = 10212_3$$

The division-remainder method is faster and easier than the repeated subtraction method. It employs the idea that successive divisions by the base are in fact successive subtractions by powers of the base. The remainders that we get when we sequentially divide by the base end up being the digits of the result, which are read from bottom to top. This method is illustrated in Example 2.3.

EXAMPLE 2.3 Convert 104_{10} to base 3 using the division-remainder method.

$3 \mid \underline{104}$	2	3 divides 104 34 times with a remainder of 2
$3 \mid \underline{34}$	1	3 divides 34 11 times with a remainder of 1
$3 \mid \underline{11}$	2	3 divides 11 3 times with a remainder of 2
$3 \mid \underline{3}$	0	3 divides 3 1 time with a remainder of 0
$3 \mid \underline{1}$	1	3 divides 1 0 times with a remainder of 1
0		

Reading the remainders from bottom to top, we have: $104_{10} = 10212_3$.

This method works with any base, and because of the simplicity of the calculations, it is particularly useful in converting from decimal to binary. Example 2.4 shows such a conversion.

EXAMPLE 2.4 Convert 147_{10} to binary.

$2\lfloor\underline{147}$	1	2 divides 147 73 times with a remainder of 1
$2\lfloor\underline{73}$	1	2 divides 73 36 times with a remainder of 1
$2\lfloor\underline{36}$	0	2 divides 36 18 times with a remainder of 0
$2\lfloor\underline{18}$	0	2 divides 18 9 times with a remainder of 0
$2\lfloor\underline{9}$	1	2 divides 9 4 times with a remainder of 1
$2\lfloor\underline{4}$	0	2 divides 4 2 times with a remainder of 0
$2\lfloor\underline{2}$	0	2 divides 2 1 time with a remainder of 0
$2\lfloor\underline{1}$	1	2 divides 1 0 times with a remainder of 1
0		

Reading the remainders from bottom to top, we have: $147_{10} = 10010011_2$.

A binary number with N bits can represent unsigned integers from 0 to $2^N - 1$. For example, 4 bits can represent the decimal values 0 through 15, while 8 bits can represent the values 0 through 255. The range of values that can be represented by a given number of bits is extremely important when doing arithmetic operations on binary numbers. Consider a situation in which binary numbers are 4 bits in length, and we wish to add 1111_2 (15_{10}) to 1111_2. We know that 15 plus 15 is 30, but 30 cannot be represented using only 4 bits. This is an example of a condition known as **overflow**, which occurs in unsigned binary representation when the result of an arithmetic operation is outside the range of allowable precision for the given number of bits. We address overflow in more detail when discussing signed numbers in Section 2.4.

2.3.2 Converting Fractions

Fractions in any base system can be approximated in any other base system using negative powers of a radix. **Radix points** separate the integer part of a number from its fractional part. In the decimal system, the radix point is called a decimal point. Binary fractions have a binary point.

Fractions that contain repeating strings of digits to the right of the radix point in one base may not necessarily have a repeating sequence of digits in another base. For instance, ⅔ is a repeating decimal fraction, but in the ternary system it terminates as 0.2_3 ($2 \times 3^{-1} = 2 \times \frac{1}{3}$).

We can convert fractions between different bases using methods analogous to the repeated subtraction and division-remainder methods for converting integers. Example 2.5 shows how we can use repeated subtraction to convert a number from decimal to base 5.

EXAMPLE 2.5 Convert 0.4304_{10} to base 5.

$$
\begin{array}{rl}
0.4304 & \\
\underline{-\ 0.4000} & = 5^{-1} \times 2 \\
0.0304 & \\
\underline{-\ 0.0000} & = 5^{-2} \times 0 \qquad \text{(A placeholder)} \\
0.0304 & \\
\underline{-\ 0.0240} & = 5^{-3} \times 3 \\
0.0064 & \\
\underline{-\ 0.0064} & = 5^{-4} \times 4 \\
0.0000 &
\end{array}
$$

Reading from top to bottom, we find $0.4304_{10} = 0.2034_5$.

Because the remainder method works with positive powers of the radix for conversion of integers, it stands to reason that we would use multiplication to convert fractions, because they are expressed in negative powers of the radix. However, instead of looking for remainders, as we did above, we use only the integer part of the product after multiplication by the radix. The answer is read from top to bottom instead of bottom to top. Example 2.6 illustrates the process.

EXAMPLE 2.6 Convert 0.4304_{10} to base 5.

$$
\begin{array}{rl}
.4304 & \\
\underline{\times\quad 5} & \\
2.1520 & \text{The integer part is 2. Omit from subsequent multiplication.} \\
.1520 & \\
\underline{\times\quad 5} & \\
0.7600 & \text{The integer part is 0. We'll need it as a placeholder.} \\
.7600 & \\
\underline{\times\quad 5} & \\
3.8000 & \text{The integer part is 3. Omit from subsequent multiplication.} \\
.8000 & \\
\underline{\times\quad 5} & \\
4.0000 & \text{The fractional part is now zero, so we are done.}
\end{array}
$$

Reading from top to bottom, we have $0.4304_{10} = 0.2034_5$.

This example was contrived so that the process would stop after a few steps. Often things don't work out quite so evenly, and we end up with repeating fractions. Most computer systems implement specialized rounding algorithms to pro-

vide a predictable degree of accuracy. For the sake of clarity, however, we will simply discard (or truncate) our answer when the desired accuracy has been achieved, as shown in Example 2.7.

EXAMPLE 2.7 Convert 0.34375_{10} to binary with 4 bits to the right of the binary point.

```
  .34375
×     2
-------
0.68750   (Another placeholder)
  .68750
×     2
-------
1.37500
  .37500
×     2
-------
0.75000
  .75000
×     2
-------
1.50000   (This is our fourth bit. We will stop here.)
```

Reading from top to bottom, $0.34375_{10} = 0.0101_2$ to four binary places.

The methods just described can be used to directly convert any number in any base to any other base, say from base 4 to base 3 (as in Example 2.8). However, in most cases, it is faster and more accurate to first convert to base 10 and then to the desired base. One exception to this rule is when you are working between bases that are powers of two, as you'll see in the next section.

EXAMPLE 2.8 Convert 3121_4 to base 3.

First, convert to decimal:

$$3121_4 = 3 \times 4^3 + 1 \times 4^2 + 2 \times 4^1 + 1 \times 4^0$$
$$= 3 \times 64 + 1 \times 16 + 2 \times 4 + 1 = 217_{10}$$

Then convert to base 3:

```
3 | 217    1
3 |  72    0
3 |  24    0
3 |   8    2
3 |   2    2
      0    We have 3121₄ = 22001₃.
```
