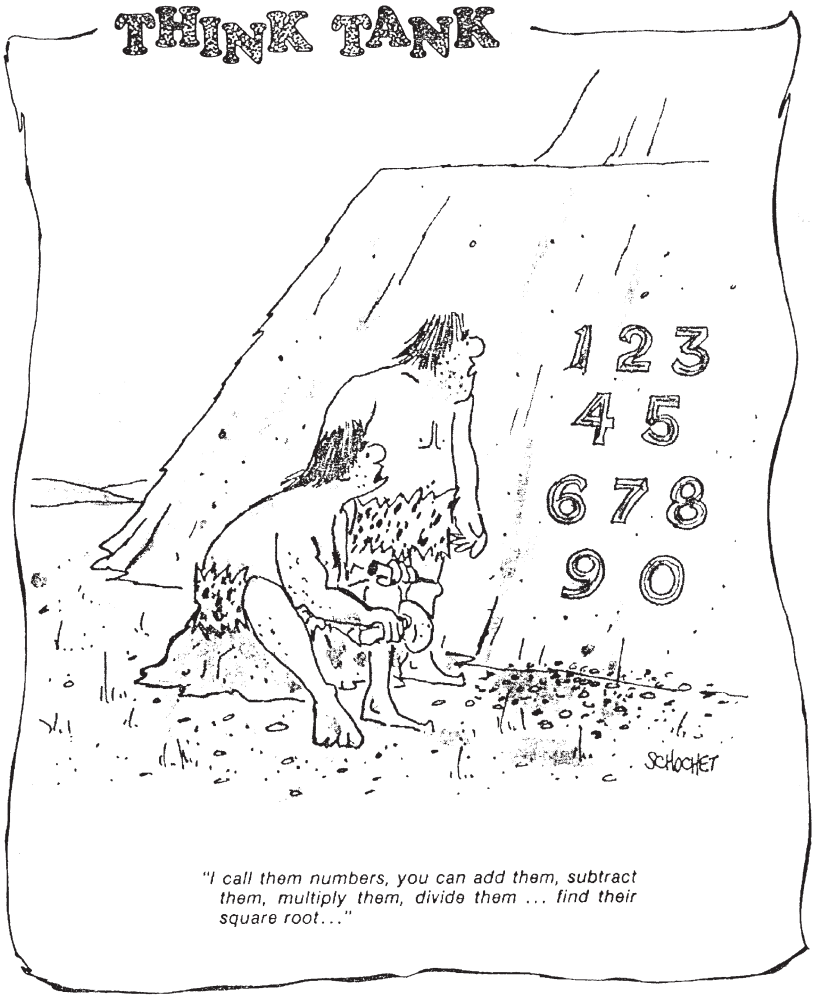
CHAPTER 3

NUMBER SYSTEMS



Courtesy of David Ahl, Creative Computing

3.0 INTRODUCTION

As humans, we generally count and perform arithmetic using the decimal, or base 10, number system. The **base** of a number system is simply the number of different digits, including zero, that exist in the number system. In any particular set of circumstances, a particular base might be chosen for convenience, efficiency, technological, or any other reasons. Historically, it seems that the main reason that we use base 10 is that humans have ten fingers, which is as good a reason as any.

Any number can be represented equivalently in any base, and it is always possible to convert a number from one base to another without changing its meaning.

Computers perform all of their operations using the **binary**, or base 2, **number** system. All program code and data are stored and manipulated in binary form. Calculations are performed using **binary arithmetic**. Each digit in a binary number is known as a **bit** (for binary digit) and can have only one of two values, **0** or **1**. Bits are commonly stored and manipulated in groups of 8 (known as a byte), 16 (usually known as a halfword), 32 (a word), or 64 bits (a doubleword). Sometimes other groupings are used.

The number of bits used in calculations affects the accuracy and size limitations of numbers manipulated by the computer. And, in fact, in some programming languages, the number of bits used can actually be specified by the programmer in declaration statements. In the programming language Java, for example, the programmer can declare a signed integer variable to be *short* (16 bits), *int* (32 bits), or *long* (64 bits) depending on the anticipated size of the number being used and the required accuracy in calculations.

The knowledge of the size limits for calculations in a particular language is sometimes extremely important, since some calculations can cause a numerical result that falls outside the range provided for the number of bits used. In some cases this will produce erroneous results, without warning to the end user of the program.

It is useful to understand how the binary number system is used within the computer. Often, it is necessary to read numbers in the computer in their binary or equivalent hexadecimal form. For example, colors in Visual Basic can be specified as a six-digit hexadecimal number, which represents a 24-bit binary number.

This chapter looks informally at number systems in general and explores the relationship between our commonplace decimal number system and number systems of other bases. Our emphasis, of course, is upon base 2, the binary number system. The discussion is kept more general, however, since it is also possible, and in fact common, to represent computer numbers in base 8 (**octal**) or base 16 (**hexadecimal).** Occasionally we even consider numbers in other bases, just for fun, and also, perhaps, to emphasize the idea that these techniques are completely general.

3.1 NUMBERS AS A PHYSICAL REPRESENTATION

As we embark upon our investigation of number systems, it is important to note that numbers usually represent some physical meaning, for example, the number of dollars in our paycheck or the number of stars in the universe. The different number systems that we use are equivalent. The physical objects can be represented equivalently in any of them. Of course, it is possible to convert between them.

In Figure 3.1, for example, there are a number of oranges, a number that you recognize as 5. In ancient cultures, the number might have been represented as

I I I I I

or, when in Rome,

V

Similarly, in base 2, the number of oranges in Figure 3.1 is represented as

101_2

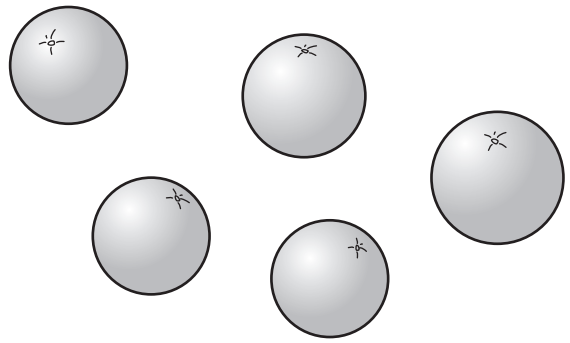
And in base 3, the representation looks like this:

12_3

The point we are making is that each of the foregoing examples is simply a different way of *representing* the same number of oranges. You probably already have experience at converting between the standard decimal number system and Roman numerals. (Maybe you even wrote a program to do so!) Once you understand the methods, it is just about as easy to convert between base 10 and the other number bases that we shall use.

FIGURE 3.1

A Number of Oranges



3.2 COUNTING IN DIFFERENT BASES

Let's consider how we count in base 10, and what each digit means. We begin with single digits,

0
1
2
3
.
.
.
9

When we reach 9, we have exhausted all possible single digits in the decimal number system; to proceed further, we extend the numbers to the 10's place:

```
10
11
12
.
.
.
```

It is productive to consider what "the 10's place" really means.

The 10's place simply represents a count of the number of times that we have cycled through the entire group of 10 possible digits. Thus, continuing to count, we have

```
1 group of 10 + 0 more
1 group of 10 + 1 more
1 group of 10 + 2
.
.
.
1 group of 10 + 9
2 groups of 10 + 0
.
.
.
9 groups of 10 + 9
```

At this point, we have used all combinations of two digits, and we need to move left another digit. Before we do so, however, we should note that each group shown here represents a count of 10, since there are 10 digits in the group. Thus, the number

$$43$$

really refers to

$$4 \times 10 + 3$$

As we move leftward to the next digit, that is, the hundreds place, we are now counting cycles of the rightmost two digits or, in other words, groups of 10×10, or 10^2, or hundreds. Thus, the number

$$527$$

really represents

```
five groups of (10 × 10)+
two groups of 10 + 7
```

FIGURE 3.2

Counting in Base 2

NUMBER	EQUIVALENT	DECIMAL EQUIVALENT
0	0×2^0	0
1	1×2^0	1
10	$1 \times 2^1 + 0 \times 2^0$	2
11	$1 \times 2^1 + 1 \times 2^0$	3
100	1×2^2	4
101	$1 \times 2^2 + 1 \times 2^0$	5
110	$1 \times 2^2 + 1 \times 2^1$	6
111	$1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$	7
1000	1×2^3	8
1001	$1 \times 2^3 + 1 \times 2^0$	9
1010	$1 \times 2^3 + 1 \times 2^1$	10

This is also represented as

$$5 \times 10^2 + 2 \times 10^1 + 7 \times 10^0$$

This method can, of course, be extended indefinitely.

The same method, exactly, applies to any number base. The only change is the size of each grouping. For example, in base 8, there are only eight different digits available (0, 1, 2, 3, 4, 5, 6, 7). Thus, each move left represents eight of the next rightmost grouping. The number

$$624_8$$

corresponds to

$$6 \times 8^2 + 2 \times 8^1 + 4 \times 8^0$$

Since $8^2 = 64_{10}$, $8^1 = 8_{10}$, and $8^0 = 1$,

$$624_8 = 6 \times 64 + 2 \times 8 + 4 = 404_{10}$$

Each digit in a number has a *weight*, or importance, relative to its neighbors left and right. The weight of a particular digit in a number is the multiplication factor used to determine the overall value of the particular digit. For example, the weights of the digits in base 8, reading from right to left are 1, 8, 64, 256, . . . , or, if your prefer, 8^0, 8^1, 8^2, 8^3, ... Just as you would expect, the weight of a digit in any base n is n times as large as the digit to its right and $(1/n)$th as large as the digit to its left.

Figure 3.2 shows the corresponding method of counting in base 2. Note that each digit has twice the weight of its next rightmost neighbor, just as in base 10 each digit had ten times the weight of its right neighbor. This is what you would expect if you consider that there are only two different values for digits in the binary cycle. You should spend enough time studying this table until you understand every detail thoroughly.

Note, too, that the steps that we have followed do not really depend on the number base that we are using. We simply go through a complete cycle, exhausting all possible different digits in the base set, and then move to the left one place and count the cycles. We repeat this process as necessary to represent the entire number.

In general, for any number base B, each digit position represents B to a power, where the power is numbered from the rightmost digit, starting with B^0. B^0, of course, is one (known as the units place) for any number base.

Thus, a simple way to determine the decimal equivalent for a number in any number base is to multiply each digit by the weight in the given base that corresponds to the position of the digit for that number.

EXAMPLES

As an example,

$$142305_6 =$$
$$1 \times 6^5 + 4 \times 6^4 + 2 \times 6^3 + 3 \times 6^2 + 0 \times 6 + 5 =$$
$$7776 + 5184 + 432 + 108 + 0 + 5 = 13505_{10}$$

■ ■ ■

Similarly,

$$110010100_2 =$$
$$1 \times 2^8 + 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 +$$
$$0 \times 2 + 0 =$$
$$256 + 128 + 16 + 4 = 404_{10}$$

You should probably work out these two examples and check your results against ours.

Often it is useful to be able to estimate quickly the value of a binary number. Since the weight of each place in a binary number doubles as we move to the left, we can generate a rough order-of-magnitude by considering only the weight for the leftmost bit or two. Starting from 1, and doubling for each bit in the number to get the weight, you can see that the most significant bit in the previous example has a value of 256. We can improve the estimate by adding half that again for the next most significant bit, which gives the value of the number in the neighborhood of 384, plus a little more for the additional bits. With a little practice, it is easy to estimate the magnitudes of binary numbers almost instantly. This technique is often sufficient for checking the results of calculations when debugging programs. (You might also want to consider it as a way of doing quick checks on your solutions to exam problems!)

We will discuss number conversion between different bases more carefully later in the chapter.

From the preceding discussion, it is fairly easy to determine the total range of possible numbers—or, equivalently, the smallest and largest integer—for a given number of digits in a particular number base. Since the weight of each digit is one larger than the largest value that can be represented by all the digits to its right, then the range of possible values for n digits is simply the weight of the nth digit, which is represented by the value

$$\mathrm{range} = \mathrm{base}^n$$

Thus, if we want to know how many different numbers can be represented by two decimal digits, the answer is 10^2. We can represent one hundred different numbers (0 . . . 99) with two decimal digits.

It's obviously easier to simply memorize the formula; if you are told that you are working with four digit numbers in base 8, you know from the formula that you can represent 8^4, or 4096 different numbers, ranging from 0 . . . 7777_8, or the decimal equivalent (0 . . . 4095).

Just as a pocket calculator stores, manipulates, and displays numbers as a group of digits, so computers store and manipulate numbers as groups of bits. Most computers work with numbers 16 bits, 32 bits, or 64 bits at a time. Applying the preceding formula to a "16-bit" PC, you can represent $2^{16} = 65{,}536$ different number values in each 16-bit

FIGURE 3.3

Decimal Range for Selected Bit Widths

BITS	DIGITS	RANGE
1	0+	2 (0 and 1)
4	1+	16 (0 to 15)
8	2+	256
10	3	1,024
16	4+	65,536 (64K)
20	6	1,048,576 (1M)
32	9+	4,294,967,296 (4G)
64	19+	approx. 1.6×10^{19}
128	38+	approx. 2.6×10^{38}

location. If you wish to extend this range, it is necessary to use some technique for increasing the number of bits used to hold your numbers, such as using two 16-bit storage locations together to hold 32 bits. There are other methods used, which are discussed in Chapter 5, but note that, regardless of the technique used, there is *no* way to store more than 65,536 different number values using 16 bits.

A table of base 10 equivalent ranges for several common computer "word lengths" is shown in Figure 3.3. There is a simple way to calculate the approximate range for a given number of bits, since 2^{10} is approximately 1000. To do so, we break up the total number of bits into a sum that consists of values where the range is easily figured out. The overall range is equal to the product of the subranges for each value. This method is best seen with examples.

For example, if you need to know the range for 18 bits, you would break up the number 18 into the sum of 10 and 8, then multiply the range for 10 bits to that for 8 bits. Since the range for 10 bits is approximately 1 K (1024, actually) and 8 bits is 256, the range for 18 bits is approximately 256 K. Similarly, the range for 32 bits would be (10-bit range) × (10-bit range) × (10-bit range) × (2-bit range) = 1 K × 1 K × 1 K × 4 = 4 gigabytes. This technique becomes easy with a bit of practice.

Notice that it takes 18 bits to represent a little more than five decimal digits. In general, approximately 3.3 bits are required for each equivalent decimal digit. This is true because $2^{3.3}$ is approximately equal to 10.

3.3 PERFORMING ARITHMETIC IN DIFFERENT NUMBER BASES

Next, we consider simple arithmetic operations in various number bases. Let us begin by looking at the simple base 10 addition table shown in Figure 3.4.

FIGURE 3.4

The Base 10 Addition Table

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13

etc.

We add two numbers by finding one in the row and the other in the column. The table entry at the intersection is the result. For example, we have used the table to demonstrate that the sum of 3 and 6 is 9. Note that the extra digit sometimes required becomes a carry that gets added into the next left column during the addition process.

More fundamentally, we are interested in how the addition table is actually created. Each column (or row) represents an increase of 1 from the previous column (or row), which is equivalent to counting. Thus, starting

FIGURE 3.5

The Base 8 Addition Table

+	0	1	2	3	4	5	6	7	
0	0	1	2	3	4	5	6	7	
1	1	2	3	4	5	6	7	10	
2	2	3	4	5	6	7	10	11	
3	3	4	5	6	7	10	11	12	**(no 8 or 9, of course)**
4	4	5	6	7	10	11	12	13	
5	5	6	7	10	11	12	13	14	
6	6	7	10	11	12	13	14	15	
7	7	10	11	12	13	14	15	16	

from the leftmost column in the table, it is only necessary to count up 1 to find the next value. Since $3 + 6 = 9$, the next column will have to carry to the next place, or 10, just as occurred when we demonstrated counting in base 10, earlier. This knowledge should make it easy for you to create a base 8 addition table. Try to create your own table before looking at the one in Figure 3.5.

Of special interest is the base 2 addition table:

+	0	1
0	0	1
1	1	10

Clearly, addition in base 2 is going to be easy!

Addition in base 2 (or any other base, for that matter) then follows the usual methods of addition that you are familiar with, including the handling of carries that you already know. The *only* difference is the particular addition table being used. There are practice problems representing multidigit binary arithmetic and column arithmetic (Exercise 3.8) at the end of the chapter.

EXAMPLE

Add 11100001_2 and 101011_2 (superscripts are carried amounts).

$$\begin{array}{r} {}^{1}1\ \ {}^{1}1\ 1\ 0\ 0\ 0^{1}\ 0^{1}\ 1 \\ +\quad\quad 1\ 0\ 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0 \end{array}$$

Let's use the estimation technique to see if our result is approximately correct. 11100001 is approximately $128 + 64 + 32$, or 224. 101011 is approximately 32. Thus, the sum should be about 256; 100001100 is indeed approximately 256, so at least we know that our calculation is in the ballpark.

As an aside, it may be of interest to some readers to consider how this addition table can be implemented in the computer using only Boolean logic, without performing any

FIGURE 3.6

The Base 10 Multiplication Table

×	0	1	2	3	4	5	6	7	8	9
0					← 0 →					
1		1	2	3	4	5	6	7	8	9
2		2	4	6	8	10	12	14	16	18
3		3	6	9	12	15	18	21	24	27
4		4	8	12	16	20	24	28	32	36
5	0	5	10	15	20	25	30	35	40	45
6		6	12	18	24	30	36	42	48	54
7		7	14	21	28	35	42	49	56	63
8		8	16	24	32	40	48	56	64	72
9		9	18	27	36	45	54	63	72	81

actual arithmetic: the result bit (the bit in the column that corresponds to the inputs) can be represented by the EXCLUSIVE-OR function of the two input bits. The EXCLUSIVE-OR function has a "1" as output only if either input, but not both inputs, is a "1." Similarly, the carry bit is represented as an AND function on the two input bits. ("1" as output if and only if both inputs are a "1.") This approach is discussed in more detail in Supplementary Chapter 1.

The process of multiplication can be reduced conceptually to multiple addition, so it should not surprise you that multiplication tables in different number bases are also reasonably straightforward. The major difference in appearance results from the fact that the carry occurs at different places.

The easiest way to create a multiplication table is to treat multiplication as multiple addition: each column (or row) represents the addition of the value in the row (or column) being created. Thus, in the following table, you can see that 5×8 is equivalent to $5 \times 7 + 5 = 40$. The familiar decimal multiplication table appears in Figure 3.6, with the example just given indicated.

FIGURE 3.7

The Base 8 Multiplication Table

×	0	1	2	3	4	5	6	7
0				← 0 →				
1		1	2	3	4	5	6	7
2		2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4		4	10	14	20	24	30	34
5		5	12	17	24	31	36	43
6		6	14	22	30	36	44	52
7		7	16	25	34	43	52	61

The same technique can be applied to the base 8 multiplication table (Figure 3.7).

Note in the foregoing table that $3 \times 3 = 3 \times 2 + 3$. Note, though, that counting up 3 from 6 (or adding 3 to 6) results in a carry after 7 is reached: $6 \rightarrow 7 \rightarrow 10 \rightarrow 11$.

The base 2 multiplication table is almost trivial, since 0 times anything is 0 and 1 times 1 is itself:

×	0	1
0	0	0
1	0	1

Because the binary multiplication table is so simple, it turns out that multiplication can be implemented in a computer fairly easily. There are only two possible results: if the multiplier is 0, the answer is 0, even if the multiplicand is a nonzero multidigit number. If the multiplier is 1, the multiplicand is brought down as the result. You might recognize the multiplication table as a Boolean AND function.

If you recall that decimal multidigit multiplication is performed by multiplying the multiplicand by each digit of the multiplier, shifting the result of each multiplication to line up with the multiplier, and adding up the results, then you realize that multidigit binary multiplication can be performed by simply shifting the multiplicand into whatever positions in the multiplier are "1" bits and adding to the result. This is easily illustrated with an example:

EXAMPLE

Multiply

```
      1101101
    × 100110
    -------
      1101101   bits shifted to line up with 2's place of multiplier
     1101101    4's place
    1101101     32's place
    ----------
    1000000101110  result (note the 0 at the end, since the 1's place is
                   not brought down)
```

We note in passing that shifting a binary number one position to the left has the effect of doubling its value. This is a result you would expect, since the shift is equivalent to multiplying the value by a 1 in the 2's place of the multiplier. This result is consistent with the fact that shifting a decimal number to the left by one position will multiply its value by 10. In general, **shifting** a number in any base **left** one digit multiplies its value by the base, and, conversely, **shifting** a number **right** one digit divides its value by the base.

Although we have not mentioned subtraction or division, the methods are similar to those that we have already discussed. In fact, the addition and multiplication tables can be directly used for subtraction and division, respectively.

3.4 NUMERIC CONVERSION BETWEEN NUMBER BASES

Conversions between whole numbers in decimal (base 10) and any other number base are relatively straightforward. With the exception of one special case discussed in Section 3.6, it is impractical to convert directly between two nondecimal number bases. Instead, base 10 would be used as an intermediary conversion base.

The easiest intuitive way to convert between base 10 and another number base is to recognize the weight of each digit in the alternative number base and to multiply that

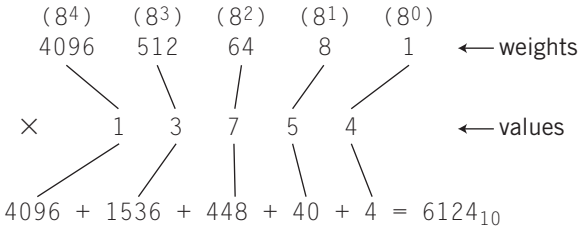
weight by the value of the digit in that position. The sum taken over all digits represents the base 10 value of the number. This is easily seen in an example:

EXAMPLE

Convert the number

$$13754_8$$

to base 10.
From the following diagram we can see the result easily:



We can use the same method in reverse to convert from base 10 to another base, although the technique is not quite as simple. In this case, it is just a question of finding the value corresponding to the weight of each digit such that the total will add up to the base 10 number that we are trying to convert.

Note that the value for each digit must be the largest value that will not exceed the number being converted. If this were not true, then there would be more than a full grouping of the next less significant digit. This idea is best clarified by example:

EXAMPLE

Suppose that we are reverse converting the preceding example, and we assume that there are six groups of 64 instead of seven. In this case, the 8's place and 1's place combined must add up to more than 64, and we've already seen that is impossible.

This provides a simple methodology for the conversion. Start with the digit whose weight is the largest possible without exceeding the number to be converted. Determine the largest value for that weight that does not exceed the number to be converted. Then, do the same for each successive digit, working from left to right.

EXAMPLE

As an example, let us convert 6124_{10} to base 5. The weights of each digit in base 5 are as follows:

$$15625 \quad 3125 \quad 625 \quad 125 \quad 25 \quad 5 \quad 1$$

Clearly the 15625 digit is too large, so the result will be a six-digit base 5 number. The number 3125 fits into 6124 only once; thus, the first digit is a 1, and the remainder to be converted is 2999. Proceeding to the next digit, 625 goes into 2999 four times with a remainder of 499, 125 into 499 three times with a remainder of 124, 25 into 124 four

times, and so on. We get a final result of

$$143444_5$$

It would be useful for you to confirm the answer by converting the result back to base 10.

This method is particularly simple if you are converting from decimal to binary, since the value that corresponds to a particular bit either fits (1) or it doesn't (0). Consider the following example:

EXAMPLE

Convert 3193_{10} to binary. The weights in binary are 4096, 2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, and 1.

Proceeding as before, the largest bit value in this conversion is the 2048 weight. Subtracting 2048 from 3193 leaves 1145 yet to be converted; thus, there is also a 1 in the 1024 place. Now the remainder is $1145 - 1024 = 121$. This means that there are 0's in the 512, 256, and 128 places. Continuing, you should confirm that the final result is

$$110001111001_2$$

Note that, in general, as the base gets smaller, the representation of a value requires more digits, and looks bigger.

An Alternative Conversion Method

Although the preceding methods are easy to understand, they are computationally difficult and prone to mistakes. In this section we will consider methods that are usually simpler to compute but are less intuitive. It is helpful to understand the reasons that these methods work, since the reasoning adds insight to the entire concept of number manipulation.

BASE 10 TO ANOTHER BASE Suppose we divide the number to be converted successively by the base, *B*, that we are converting to, and look at the remainders of each division. We will do this until there is nothing left to divide. Each successive remainder represents the value of a digit in the new base, reading the new value from right to left. Again, let us convert 6124_{10} to base 5:

EXAMPLE

```
5 ) 6124 ( 4        least significant digit
5 ) 1224 ( 4              ↑
5 )  244 ( 4              |
5 )   48 ( 3              |
5 )    9 ( 4              |
5 )    1 ( 1        most significant digit
      0
```

The answer is 143444_5, which agrees with our earlier result.

The first time that we perform the division, we are, in effect, determining how many groups of 5 (or, in the general case, *B*) fit into the original number. The remainder is the number of single units left over, which is, in other words, the units place of the converted number.

The original number has now been divided by 5, so the second division by 5 determines how many groups of 5^2, or 25, fit into the number. The remainder in this case is the number of 5-groups that are left over, which is the second digit from the right.

Each time we divide by the base, we are increasing the power of the group being tested by one, and we do this until there is no group left. Since the remainders correspond to the part of the number that does not exactly fit the group, we can read the converted number easily by reading the remainders from the bottom up.

Here's another example:

EXAMPLE

Convert 8151_{10} to base 16, also known as hexadecimal:

```
16 )  8151  ( 7
16 )   509  (13    in base 16, this is represented by the letter "D"
16 )    31  (15    in base 16, this is represented by the letter "F"
         1
```

The answer is $1FD7_{16}$. We suggest that you verify this answer by using the technique of digit weight multiplication to convert this answer back to decimal form.

ANOTHER NUMBER BASE TO BASE 10 An alternative method can be used to convert from other number bases to base 10. The technique is also computationally simple: starting from the most significant digit, we multiply by the base, *B*, and add the next digit to the right. We repeat this process until the least significant digit has been added.

EXAMPLE

Convert 13754_8 to base 10:

```
    1
  × 8
  8 + 3 = 11
        × 8
       88 + 7 = 95
             × 8
            760 + 5 = 765
                  × 8
                 6120 + 4 = 6124₁₀
```

If you count the number of times that each digit in the example is multiplied by the base number, in this case 8, you discover that the leftmost digit is multiplied by 8 four times, or 8^4, and that each successive digit is multiplied by 8 one less time, until you arrive at the rightmost digit, which is not multiplied by the base number at all. Thus, each digit is multiplied by its proper weight, and the result is what we would expect. In the next chapter, you will see that this method is also useful for converting a sequence of digits in alphanumeric form to an actual number.

You have now been introduced to two different methods for performing conversions in each direction. You should practice all four methods; then you can use whichever two methods are easiest for you to remember.

3.5 HEXADECIMAL NUMBERS AND ARITHMETIC

The hexadecimal, or base 16, number representation system is important because it is commonly used as a shorthand notation for binary numbers. The conversion technique between hexadecimal and binary notations is particularly simple because there is a direct relationship between the two. Each hexadecimal number represents exactly 4 binary bits. Most computers store and manipulate instructions and data using word sizes that are multiples of 4 bits. Therefore, the hexadecimal notation is a convenient way to represent computer words. Of course, it is also much easier to read and write than binary notation. The technique for converting between binary and hexadecimal is shown later in this chapter.

Although hexadecimal numbers are represented and manipulated in the same way as those of other bases, we must first provide symbols to represent the additional digits beyond 9 that we require to represent sixteen different quantities with a single integer.

By convention, we use the digits 0–9, followed by the first six alphabetical characters A–F. Thus, the digits 0–9 have their familiar meaning; the letters A–F correspond to what in a decimal base would be quantities of 10–15, respectively. To count in hexadecimal we count from 0 to 9, then A to F, and then move left to the next digit. Since there are sixteen digits, each place represents a power of 16. Thus, the number

$$2A4F_{16}$$

is equivalent to

$$2 \times 16^3 + 10 \times 16^2 + 4 \times 16 + 15, \quad \text{or}$$
$$10831_{10}$$

Addition and multiplication tables can be created for the hexadecimal number system. These tables each have sixteen rows and sixteen columns, as you would expect. The addition table is shown in Figure 3.8. Before you look at the figure, you should try to work the hexadecimal addition and multiplication tables out for yourself (see Exercise 3.7).

3.6 A SPECIAL CONVERSION CASE—NUMBER BASES THAT ARE RELATED

A special possibility for conversion exists when one number base is an integer power of another. In this case, a direct conversion can easily be made. In fact, with a bit of practice, the conversion can be done mentally and the answer written down directly. These conversions

FIGURE 3.8

Hexadecimal Addition Table

+	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
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

work because a grouping of several digits in the smaller number base corresponds, or maps, exactly to a single digit in the larger number base.

Two particularly useful examples for computer work are the cases of conversion between base 2 and base 8 and conversion between base 2 and base 16. Since $8 = 2^3$, we can represent binary numbers directly in base 8 using one octal digit to correspond to each three binary digits. Similarly, it takes one hexadecimal digit to exactly represent 4 bits.

The advantage of representing binary numbers in hexadecimal or octal is obvious: it is clearly much easier to read and manipulate four-digit hexadecimal numbers than 16-bit binary numbers. Since the conversion between binary and octal and hexadecimal is so simple, it is common to use hexadecimal or octal representation as a shorthand notation for binary. (Note that base 8 and base 16 are not directly related to each other by power, but conversion could be performed easily by using base 2 as an intermediary.)

Since the correspondence of binary and octal or hexadecimal is exact, the conversion process simply consists of breaking the binary number into groups of three or four, starting from the least significant bit (the unit bit), and converting each group independently. It

may be necessary to mentally add 0s to the left end of the number to convert the most significant digit. This is most easily illustrated with an example:

EXAMPLE

Let us convert

$$11010111011000$$

to hexadecimal.

Grouping the binary number by fours, we have

$$0011\ 0101\ 1101\ 1000$$

or

$$35D8_{16}$$

Note that we added two zeros at the left end of the binary number to create groups of four.

The conversion in the other direction works identically. Thus,

$$275331_8$$

becomes

$$010\ 111\ 101\ 011\ 011\ 001_2$$

For practice, now convert this value to hexadecimal.

Most computer manufacturers today prefer to use hexadecimal, since a 16-bit or 32-bit number can be represented exactly by a four- or eight-digit hexadecimal number. (How many octal digits would be required?) A few manufacturers still use octal representation for some applications.

You might ask why it is necessary to represent data in binary form at all. After all, the binary form is used within the computer, where it is usually invisible to the user. There are many occasions, however, where the ability to read the binary data is very useful. Remember that the computer stores both instructions and data in binary form. When debugging a program, it may be desirable to be able to read the program's instructions and to determine intermediate data steps that the computer is using. Older computers used to provide binary dumps for this purpose. Binary dumps were complete octal listings of everything stored in memory at the time the dump was requested. Even today it is sometimes important, for example, to be able to read the binary data from a floppy disk to recover a lost or damaged file. Modern computer operating systems and networks present a variety of troubleshooting data in hexadecimal form.

Conversions between binary and hexadecimal notation are used frequently. We strongly recommend that you practice to become proficient at working with hexadecimal notation.

3.7 FRACTIONS

Up to this point we have limited our discussion to whole numbers, or, if you prefer, integers. The representation and conversion of fractional numbers are somewhat more difficult because there is not necessarily an exact relationship between fractional numbers

in different number bases. More specifically, fractional numbers that can be represented exactly in one number base may be impossible to represent exactly in another. Thus, exact conversion may be impossible. A couple of simple examples will suffice:

EXAMPLE

The decimal fraction

$$0.1_{10} \text{ or } 1/10$$

cannot be represented exactly in binary form. There is no combination of bits that will add up exactly to this fraction. The binary equivalent begins

$$0.0001100110011_2\ldots$$

This binary fraction repeats endlessly with a repeat cycle of four. Similarly, the fraction

$$1/3$$

is not representable as a decimal value in base 10. In fact, we represent this fraction decimally as

$$0.3333333\ldots$$

As you will realize shortly, this fraction can be represented exactly in base 3 as

$$0.1_3$$

Recall that the value of each digit to the left of a **decimal point** in base 10 has a weight ten times that of its next right neighbor. This is obvious to you, since you already know that each digit represents a group of ten objects in the next right neighbor. As you have already seen, the same basic relationship holds for any number base: the weight of each digit is *B* times the weight of its right neighbor. This fact has two important implications:

1. If we move the number point one place to the right in a number, the value of the number will be multiplied by the base. A specific example will make this obvious:

   $$1390_{10} \text{ is ten times as large as } 139.0_{10}$$

   $$139_{\times}0.$$

   Moving the point right one space, therefore, multiplies the number by ten. Only a bit less obvious (pun intended),

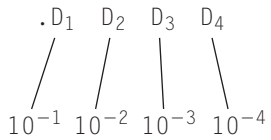
   $$100_2 \text{ is twice as big as } 10_2$$

   (Note: We have used the phrase "number point" because the word "decimal" specifically implies base 10. More generally, the number point is known by the name of its base, for example, **binary point** or *hexadecimal point*. It is sometimes also called a **radix point.**)

2. The opposite is also true: if we move the number point to the left one place, the value is divided by the base. Thus, each digit has strength $1/B$ of its left neighbor. This is true on both sides of the number point.

Moving the point to the left one space divides the value by ten.

Thus, for numbers to the right of the number point, successive digits have values $1/B$, $1/B^2$, $1/B^3$, and so on. In base 10, the digits then have value



which is equivalent to

$$1/10\ 1/100\ 1/1000\ 1/10,000$$

This should come as no surprise to you, since $1/10 = 0.1$, $1/100 = 0.01$, and so forth. (Remember from algebra that $B^{-k} = 1/B^k$.)

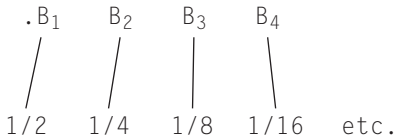
Then, a decimal number such as

$$0.2589$$

has value

$$2 \times (1/10) + 5 \times (1/100) + 8 \times (1/1000) + 9 \times (1/10,000)$$

Similarly in base 2, each place to the right of the binary point is 1/2 the weight of its left-hand neighbor. Thus, we have



As an example,

$$0.101011$$

is equivalent to

$$1/2 + 1/8 + 1/32 + 1/64$$

which has decimal value

$$0.5 + 0.125 + 0.03125 + 0.015625 = 0.671875_{10}$$

Since there is no general relationship between fractions of types $1/10^k$ and $1/2^k$, there is no reason to assume that a number that is representable in base 10 will also be representable

in base 2. Commonly, it isn't so. (The converse is not the case; since all fractions of the form $1/2^k$ can be represented in base 10, and since each bit represents a fraction of this form, fractions in base 2 can always be converted exactly to fractions in base 10.) As we have already shown with the value 0.1_{10}, many base 10 fractions result in endless base 2 fractions.

Incidentally, as review, consider the hexadecimal representation of the binary fraction representing 0.1_{10}. Starting from the numeric point, which is the common element of all number bases ($B^0 = 1$ in all bases), you group the bits into groups of four:

$$0.0001\ 1001\ 1001\ 1001 = 0.19999_{16}$$

In this particular case, the repeat cycle of four happens to be the same as the hexadecimal grouping of four, so the digit "9" repeats forever.

When fractional conversions from one base to another are performed, they are simply stopped when the desired accuracy is attained (unless, of course, a rational solution exists).

Fractional Conversion Methods

The intuitive conversion methods previously discussed can be used with fractional numbers. The computational methods have to be modified somewhat to work with fractional numbers.

Consider the intuitive methods first. The easiest way to convert a fractional number from some base B to base 10 is to determine the appropriate weights for each digit, multiply each digit by its weight, and add the values. You will note that this is identical to the method that we introduced previously for integer conversion.

EXAMPLE

Convert 0.12201_3 to base 10.

The weights for base 3 fractions (we remind you that the rules work the same for *any* number base!) are:

$$\frac{1}{3} \qquad \frac{1}{9} \qquad \frac{1}{27} \qquad \frac{1}{81} \qquad \frac{1}{243}$$

Then, the result is

$$1 \times 1/3 + 2 \times 1/9 + 2 \times 1/27 + 1 \times 1/243$$

Two different approaches could be taken at this point. Either we can convert each value to decimal base, multiply, and add,

$$\mathtt{value} = 0.33333 + 0.22222 + 0.07407 + 0.00412 = 0.63374_{10}$$

or, more easily, we can find a common denominator, convert each fraction to the common denominator, add, and then divide by the common denominator. Most easily, we can pick the denominator of the least significant digit, in this case 243:

$$\mathtt{value} = \frac{81 + 2 \times 27 + 2 \times 9 + 1}{243} = \frac{154}{243} = 0.63374$$

If you look at the numerator of the last equation carefully, you might notice that the numerator consists of weighted digits, where the digits correspond to the weights of the

fraction as if the ternary point had been shifted five places right to make the fraction into a whole number. (The base 3 number point is called a *ternary* point.) A shift five places to the right multiplies the number by $3 \rightarrow 9 \rightarrow 27 \rightarrow 81 \rightarrow 243$; therefore, we have to divide by 243 to restore the original fraction.

Repeating this exercise with another, perhaps more practical, example should help to solidify this method for you:

EXAMPLE

Convert 0.110011_2 to base 10.

Shifting the binary point six places to the right and converting, we have

$$\texttt{numerator value} = 32 + 16 + 2 + 1 = 51$$

Shifting the binary back is equivalent to dividing by 2^6, or 64. Dividing the numerator 51 by 64 yields

$$\texttt{value} = 0.796875$$

The intuitive method for converting numbers from base 10 to another base can also be used. This is the method shown earlier where you fit the largest product of weights for each digit without exceeding the original number. In the case of fractions, however, you are working with fractional decimal numbers, and the actual calculation may be time consuming and difficult except in simple cases.

EXAMPLE

Convert the number 0.1_{10} to binary representation. The weights for binary fractions are

$$\frac{1}{2} \qquad \frac{1}{4} \qquad \frac{1}{8} \qquad \frac{1}{16} \qquad \frac{1}{32} \qquad \texttt{etc.}$$

These are easier to use when converted into decimal form: 0.5, 0.25, 0.125, 0.0625, and 0.03125, respectively. The largest value that fits into 0.1_{10} is 0.0625, which corresponds to a value of 0.0001_2. The remainder to be converted is $0.1 - 0.0625 = 0.0375$. Since 0.03125 fits into this remainder, the next bit is also a 1: 0.00011_2, and so on. As an exercise, you may want to carry this conversion out a few more places.

To convert fractional numbers from base 10 to another base, it is usually easier to use a variation on the division method shown earlier. Recall that for an integer, this involved dividing the number repeatedly by the base value and retaining the remainders. Effectively, this method works by shifting the radix point to the left one place each time we divide by the base value and noticing what drops over the radix point, which is the remainder. The number point is initially assumed to be to the right of the number.

When the value being converted is to the right of the number point, the procedure must work exactly the opposite. We *multiply* the fraction by the base value repeatedly, and record, then drop, the values that move to the left of the radix point. We repeat this procedure until the desired number of digits of accuracy is attained or until the value being multiplied is zero. Each time we multiply, we effectively expose the next digit.

For example, if the value in base 10 is 0.5, multiplying that by 2 would yield 1.0, which says that in base 2 there would have been a 1 in the 1/2-bit location. Similarly, 0.25 would be multiplied by 2, twice, to reach a value of 1.0, indicating a 1 in the 1/4-bit location. An example of the procedure should clarify this explanation:

EXAMPLE

Convert 0.828125_{10} to base 2. Multiplying by 2, we get

```
  .828125
×        2
1.656250   The 1 is saved as result,
×        2 then dropped, and the
1.312500   process repeated
×        2
0.625000
×        2
1.250000
×        2
0.500000
×        2
1.000000
```

The final result, reading the overflow values downward, is 0.110101_2. This is an example of a conversion that reaches closure. You will recall that we stated earlier that 0.1_{10} is an example of a number that does not convert exactly into base 2. The procedure for that case follows.

```
  .100000
×        2
0.200000
×        2
0.400000
×        2
0.800000
×        2
1.600000
×        2
1.200000
×        2
0.400000
```

The repeating nature of this conversion is clear at this point.

Finally, we note that conversion between bases where one base is an integer power of the other can be performed for fractions by grouping the digits in the smaller base as

before. For fractions, the grouping must be done from left to right; the method is otherwise identical.

EXAMPLE

To convert 0.1011_2 to base 8, group the digits by threes (since $2^3 = 8$) and convert each group as usual. Note that it is necessary to supplement the second group with 0's. As you would expect, fractional zeros are appended to the right of the fraction.

Therefore,

$$0.101\_100_2 = 0.54_8$$

3.8 MIXED NUMBER CONVERSIONS

The usual arithmetic rules apply to fractional and mixed numbers. When adding and subtracting these numbers, the radix points must line up. During multiplication and division, the radix point is determined in exactly the same way as it would be in base 10. For multiplication in base 8, for example, you would add the number of digits to the right of the radix in the multiplier and the multiplicand; the total would be the number of digits to the right of the radix point in the result.

Extra caution is required when performing base conversions on numbers that contain both integer and fractional parts. The two parts must be converted separately.

The radix point is the fixed reference in a conversion. It does not move, since the digit to its left is a unit digit in every base; that is, B^0 is always 1, regardless of B.

It is possible to shift a mixed number in order to make it an integer. Unfortunately, there is a tendency to forget that the shift takes place in a particular base. A number shifted in base 2, say, cannot be converted and then shifted back in base 10 because the factor used in the shift is 2^k, which obviously has a different value than 10^k. Of course, it is possible to perform the shift and then divide the converted number by the original shift value, but this is usually more trouble than it's worth.

Instead, it's usually easier to remember that each part is converted separately, with the radix point remaining fixed at its original location.

SUMMARY AND REVIEW

Counting in bases other than 10 is essentially similar to the familiar way of counting. Each digit place represents a count of a group of digits from the next less significant digit place. The group is of size B, where B is the base of the number system being used. The least significant digit, of course, represents single units. Addition, subtraction, multiplication, and division for any number base work similarly to base 10, although the arithmetic tables look different.

There are several different methods that can be used to convert whole numbers from base B to base 10. The informal method is to recognize the base 10 values for each digit place and simply to add the weighted values for each digit together. A more formal method converts from base B to base 10 using successive multiplication by the present base and addition of the next digit. The final total represents the base 10 solution to the conversion. Similar methods exist for converting from base 10 to a different number base.

The conversion of number bases in which one base is an integer power of the other may be performed by recognizing that multiple digit places in the smaller base represent a single-digit place in the larger. Conversion is then done by grouping and converting each multiple set of digits individually.

Fractional and mixed numbers must be handled more carefully. The integer and fractional parts must be treated independently of each other. Although the conversion method is the same, the choice of the multiplication or division operation is reversed for the fractional part. Again, directly related bases can be converted by grouping digits in one base and converting each group independently.

FOR FURTHER READING

Working in different number bases was part of a trend in the teaching of mathematics in the 1960s and 1970s known as "the new math". The material is still taught in many elementary schools.

Many libraries carry texts with such titles as "Elementary Math". A good, brief review of arithmetic as it applies to the computer can be found in the Schaum outline series book *Essential Computer Mathematics* [LIPS82]. A funny introduction to "new math" can be found on the recording "That Was the Year That Was" by Tom Lehrer [LEHR65]. In addition, most books on computer arithmetic contain substantial discussions of the topics covered in this chapter. Typical computer arithmetic books include those by Spaniol [SPAN81] and Kulisch and Maranker [KULI81]. A clear and thorough discussion of this material can be found in the computer architecture book by Hennessy and Patterson [HENN06].

KEY CONCEPTS AND TERMS

base
binary arithmetic
binary number
binary point
binary-decimal conversion
binary-hexadecimal conversion
binary-octal conversion
bit
decimal point
decimal-binary conversion
fractional conversion
hexadecimal-binary conversion
hexadecimal number
left shift
mixed number conversion
octal number
radix point
right shift

READING REVIEW QUESTIONS

3.1 In the book we show that 527_{10} represents $5 \times 10^2 + 2 \times 10^1 + 7 \times 10^0$. What is the representation for 527_8? What would its equivalent base 10 value be?

3.2 How many different digits would you expect to find in base 6? What is the largest digit in base 6? Let z represent that largest digit. What is the next value after 21z if you're counting up by 1's? What is the next value after 4zz if you're counting up by 1's?

3.3 Use the table in Figure 3.5 to add 21_8 and 33_8. Use the table in Figure 3.5 to add 46_8 and 43_8.

3.4 Use the base 2 addition table to add 10101_2 and 1110_2. Use the base 2 multiplication table to multiply 10101_2 and 1110_2.

3.5 What are the first six weights in base 2? Using these weights, convert 100101_2 to base 10.

3.6 What are the first three weights in base 16? Using these weights, convert 359_{16} to base 10. (Notice that the same technique works for *any* base, even if the base is larger than 10.)

3.7 Using the weights in base 8, convert 212_{10} into base 8. Convert 3212_{10} into base 8.

3.8 Using the weights in base 16, convert 117_{10} into base 16. Convert 1170_{10} into base 16.

3.9 Use the division conversion method to convert 3212_{10} into base 8. Confirm that your answer is the same as that in question 7, above.

3.10 Use the division method to convert 1170_{10} to base 16. Confirm that your answer is the same as that in question 8, above.

3.11 Use the division method to convert 12345_{10} to base 16. Verify your answer by using the weights method to convert your answer back to base 10.

3.12 Use the division method to convert 12345_{10} to base 2. Verify your answer by using the weights method to convert your answer back to base 10.

3.13 Use the multiplication method to convert 1011_2 to base 10. Verify your answer by using the wieghts method to convert the number back to base 2.

3.14 Use the multiplication method to convert 1357_{16} to base 10. Verify your answer by using the division method to convert your answer back to base 16.

3.15 What number in base 10 is equivalent to *D* in base 16? What number in base 16 is equivalent to the number 10 in base 10? Use the weights method to convert the number $5D_{16}$ to base 10. Use the division method to convert your answer back to base 16.

3.16 Convert the number 101000101100_2 *directly* from binary to hexadecimal. Without looking at the original number, convert your answer directly back to binary and compare your final answer with the original number.

3.17 Convert the number 1111001101100_2 *directly* from binary to hexadecimal. Without looking at the original number, convert your answer directly back to binary and compare your final answer with the original number.

EXERCISES

3.1 **a.** Determine the power of each digit for five-digit numbers in base 6.

b. Use your results from part (a) to convert the base 6 number 24531_6 to decimal.

3.2 Determine the power of each digit for four-digit numbers in base 16. Which place digits in base 2 have the same power?

3.3 Convert the following hexadecimal numbers to decimal:

a. 4E

b. 3D7

c. 3D70

3.4 Some older computers used an 18-bit word to store numbers. What is the decimal range for this word size?

3.5 How many bits will it take to represent the decimal number 3,175,000? How many bytes will it take to store this number?

3.6 **a.** Create addition and multiplication tables for base 12 arithmetic. Use alphabetic characters to represent digits 10 and larger.

b. Using your tables from part (a), perform the following addition:

$$\begin{array}{r} 25A84_{12} \\ +\ 70396_{12} \\ \hline \end{array}$$

c. Multiply the following numbers together:

$$\begin{array}{r} 2A6_{12} \\ \times\ B1_{12} \\ \hline \end{array}$$

3.7 **a.** Create the hexadecimal multiplication table.

b. Use the hexadecimal table in Figure 3.8 to perform the following addition:

$$\begin{array}{r} 2AB3 \\ +\ 35DC \\ \hline \end{array}$$

c. Add the following numbers:

$$\begin{array}{r} 1\,FF9 \\ +\ F7 \\ \hline \end{array}$$

d. Multiply the following numbers:

$$\begin{array}{r} 2E26 \\ \times\ 4A \\ \hline \end{array}$$

3.8 Add the following binary numbers:

a.

$$\begin{array}{r} 101101101 \\ +\ 10011011 \\ \hline \end{array}$$

b.

$$\begin{array}{r} 110111111 \\ +\ 110111111 \\ \hline \end{array}$$

c.

$$\begin{array}{r} 11010011 \\ +\ 10001010 \\ \hline \end{array}$$

d.

$$\begin{array}{r} 1101 \\ 1010 \\ 111 \\ +\ 101 \\ \hline \end{array}$$

e. Repeat the previous additions by converting each number to hexadecimal, adding, and converting the result back to binary.

3.9 Multiply the following binary numbers together:

a.

$$\begin{array}{r} 1101 \\ \times\ 101 \\ \hline \end{array}$$

b.

$$\begin{array}{r} 11011 \\ \times\ 1011 \\ \hline \end{array}$$

3.10 Perform the following binary divisions:

a.

$$110\overline{)1010001001}$$

b.

$$1011\overline{)11000000000}$$

3.11 Using the powers of each digit in base 8, convert the decimal number 6026 to octal.

3.12 Using the powers of each digit in hexadecimal, convert the decimal number 6026 to hexadecimal.

3.13 Using the division method, convert the following decimal numbers:

a. 13750 to base 12

b. 6026 to hexadecimal

c. 3175 to base 5

3.14 Using the division method, convert the following decimal numbers to binary:

a. 4098

b. 71269

c. 37

In each case, check your work by using the power of each digit to convert back to decimal.

3.15 Using the multiplication method, convert the following numbers to decimal:

a. 1100010100100001_2

b. $C521_{16}$

c. $3ADF_{16}$

d. 24556_7

3.16 Convert the following binary numbers directly to hexadecimal:

a. 101101110111010

b. 1111111111110001

c. 1111111101111

d. 110001100011001

3.17 Convert the following hexadecimal numbers to binary:

a. 4F6A

b. 9902

c. A3AB

d. 1000

3.18 Select a number base that would be suitable for direct conversion from base 3, and convert the number 22011210_3 to that base.

3.19 **a.** Convert the base 4 number 13023031_4 directly to hexadecimal. Check your result by converting both the original number and your answer to decimal.

b. Convert the hexadecimal number $9B62_{16}$ directly to base 4; then convert both the original number and your answer to binary to check your result.

3.20 Convert the base 3 number 210102_3 to octal. What process did you use to do this conversion?

3.21 Convert the octal number 27745_8 to hexadecimal. Do *not* use decimal as an intermediary for your conversion. Why does a direct conversion not work in this case?

3.22 Using whatever programming language is appropriate for you, write a program that converts a whole number input by the user from base 8 to base 10. Your program should flag as an error any input that contains the digits 8 or 9.

3.23 Using whatever programming language is appropriate for you, write a program that converts a whole number input from decimal to hexadecimal.

3.24 Using whatever programming language is appropriate for you, write a program that converts whole numbers in either direction between binary and hexadecimal.

3.25 Convert the following numbers from decimal to hexadecimal. If the answer is irrational, stop at four hexadecimal digits:

a. 0.6640625

b. 0.3333

c. 69/256

3.26 Convert the following numbers from their given base to decimal:

a. 0.1001001_2

b. $0.3A2_{16}$

c. $0.2A1_{12}$

3.27 Convert the following numbers from decimal to binary and then to hexadecimal:

a. 27.625

b. 4192.37761

3.28 What is the decimal value of the following binary numbers?

a. 1100101.1

b. 1110010.11

c. 11100101.1

3.29 Draw a flow diagram that shows step by step the process for converting a mixed number in a base other than 10 to decimal.

3.30 Write a computer program in a language appropriate for you that converts mixed numbers between decimal and binary in both directions.