# CHAPTER 2

# NUMBER SYSTEMS AND CODES

# OUTLINE

- 2-1 Binary-to-Decimal Conversions
- 2-2 Decimal-to-Binary Conversions
- 2-3 Hexadecimal Number System
- 2-4 BCD Code
- 2-5 The Gray Code

- 2-6 Putting It All Together
- 2-7 The Byte, Nibble, and Word
- 2-8 Alphanumeric Codes
- 2-9 Parity Method for Error Detection
- 2-10 Applications

# **OBJECTIVES**

Upon completion of this chapter, you will be able to:

- Convert a number from one number system (decimal, binary, hexadecimal) to its equivalent in one of the other number systems.
- Cite the advantages of the hexadecimal number system.
- Count in hexadecimal.
- Represent decimal numbers using the BCD code; cite the pros and cons of using BCD.
- Understand the difference between BCD and straight binary.
- Understand the purpose of alphanumeric codes such as the ASCII code.
- Explain the parity method for error detection.
- Determine the parity bit to be attached to a digital data string.

# INTRODUCTION

The binary number system is the most important one in digital systems, but several others are also important. The decimal system is important because it is universally used to represent quantities outside a digital system. This means that there will be situations where decimal values must be converted to binary values before they are entered into the digital system. For example, when you punch a decimal number into your hand calculator (or computer), the circuitry inside the machine converts the decimal number to a binary value.

Likewise, there will be situations where the binary values at the outputs of a digital system must be converted to decimal values for presentation to the outside world. For example, your calculator (or computer) uses binary numbers to calculate answers to a problem and then converts the answers to decimal digits before displaying them.

As you will see, it is not easy to simply look at a large binary number and convert it to its equivalent decimal value. It is very tedious to enter a long sequence of 1s and 0s on a keypad, or to write large binary numbers on a piece of paper. It is especially difficult to try to convey a binary quantity while speaking to someone. The hexadecimal (base-16) number system has become a very standard way of communicating numeric values in digital systems. The great advantage is that hexadecimal numbers can be converted easily to and from binary.

Other methods of representing decimal quantities with binary-encoded digits have been devised that are not truly number systems but offer the ease of conversion between the binary code and the decimal number system. This is referred to as binary-coded decimal. Quantities and patterns of bits might be represented by any of these methods in any given system and throughout the written material that supports the system, so it is very important that you are able to interpret values in any system and convert between any of these numeric representations. Other codes that use 1s and 0s to represent things such as alphanumeric characters will be covered because they are so common in digital systems.

# 2-1 BINARY-TO-DECIMAL CONVERSIONS

As explained in Chapter 1, the binary number system is a positional system where each binary digit (bit) carries a certain weight based on its position relative to the LSB. Any binary number can be converted to its decimal equivalent simply by summing together the weights of the various positions in the binary number that contain a 1. To illustrate, let's change  $11011_2$  to its decimal equivalent.

Let's try another example with a greater number of bits:

Note that the procedure is to find the weights (i.e., powers of 2) for each bit position that contains a 1, and then to add them up. Also note that the MSB has a weight of  $2^7$  even though it is the eighth bit; this is because the LSB is the first bit and has a weight of  $2^0$ .

#### **REVIEW QUESTIONS**

1. Convert 100011011011<sub>2</sub> to its decimal equivalent.

2. What is the weight of the MSB of a 16-bit number?

# 2-2 DECIMAL-TO-BINARY CONVERSIONS

There are two ways to convert a decimal *whole* number to its equivalent binary-system representation. The first method is the reverse of the process described in Section 2-1. The decimal number is simply expressed as a sum of powers of 2, and then 1s and 0s are written in the appropriate bit positions. To illustrate:

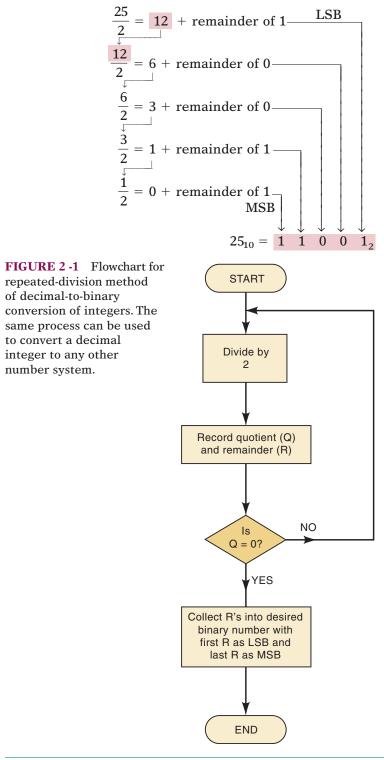
$$45_{10} = 32 + 8 + 4 + 1 = 2^{5} + 0 + 2^{3} + 2^{2} + 0 + 2^{0}$$
  
= 1 0 1 1 0 1<sub>2</sub>

Note that a 0 is placed in the  $2^1$  and  $2^4$  positions, since all positions must be accounted for. Another example is the following:

$$76_{10} = 64 + 8 + 4 = 2^{6} + 0 + 0 + 2^{3} + 2^{2} + 0 + 0$$
  
= 1 0 0 1 1 0 0<sub>2</sub>

# **Repeated Division**

Another method for converting decimal integers uses repeated division by 2. The conversion, illustrated below for  $25_{10}$ , requires repeatedly dividing the decimal number by 2 and writing down the remainder after each division until a quotient of 0 is obtained. Note that the binary result is obtained by writing the first remainder as the LSB and the last remainder as the MSB. This process, diagrammed in the flowchart of Figure 2-1, can also be used to convert from decimal to any other number system, as we shall see.



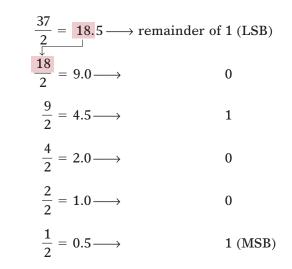
# **CALCULATOR HINT:**

If you use a calculator to perform the divisions by 2, you can tell whether the remainder is 0 or 1 by whether or not the result has a fractional part. For instance, 25/2 would produce 12.5. Since there is a fractional part (the .5), the remainder is a 1. If there were no fractional part, such as 12/2 = 6, then the remainder would be 0. The following example illustrates this.

EXAMPLE 2-1

Convert 37<sub>10</sub> to binary. Try to do it on your own before you look at the solution.

#### Solution



Thus,  $37^{10} = 100101_2$ .

#### **Counting Range**

Recall that using N bits, we can count through  $2^N$  different decimal numbers ranging from 0 to  $2^N - 1$ . For example, for N = 4, we can count from  $0000_2$  to  $1111_2$ , which is  $0_{10}$  to  $15_{10}$ , for a total of 16 different numbers. Here, the largest decimal value is  $2^4 - 1 = 15$ , and there are  $2^4$  different numbers. In general, then, we can state:

Using *N* bits, we can represent decimal numbers ranging from 0 to  $2^N - 1$ , a total of  $2^N$  different numbers.

#### EXAMPLE 2-2

- (a) What is the total range of decimal values that can be represented in eight bits?
- (b) How many bits are needed to represent decimal values ranging from 0 to 12,500?

#### Solution

(a) Here we have N = 8. Thus, we can represent decimal numbers from 0 to  $2^8 - 1 = 255$ . We can verify this by checking to see that  $1111111_2$  converts to  $255_{10}$ .

(b) With 13 bits, we can count from decimal 0 to  $2^{13} - 1 = 8191$ . With 14 bits, we can count from 0 to  $2^{14} - 1 = 16,383$ . Clearly, 13 bits aren't enough, but 14 bits will get us up beyond 12,500. Thus, the required number of bits is 14.

#### **REVIEW QUESTIONS**

- 1. Convert  $83_{10}$  to binary using both methods.
- 2. Convert  $729_{10}$  to binary using both methods. Check your answer by converting back to decimal.
- 3. How many bits are required to count up to decimal 1 million?

# 2-3 HEXADECIMAL NUMBER SYSTEM

The **hexadecimal number system** uses base 16. Thus, it has 16 possible digit symbols. It uses the digits 0 through 9 plus the letters A, B, C, D, E, and F as the 16 digit symbols. The digit positions are weighted as powers of 16 as shown below, rather than as powers of 10 as in the decimal system.

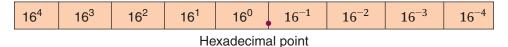


Table 2-1 shows the relationships among hexadecimal, decimal, and binary. Note that each hexadecimal digit represents a group of four binary digits. It is important to remember that hex (abbreviation for "hexadecimal") digits A through F are equivalent to the decimal values 10 through 15.

#### TABLE 2-1



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

### **Hex-to-Decimal Conversion**

A hex number can be converted to its decimal equivalent by using the fact that each hex digit position has a weight that is a power of 16. The LSD has a weight of  $16^0 = 1$ ; the next higher digit position has a weight of  $16^1 = 16$ ; the next has a weight of  $16^2 = 256$ ; and so on. The conversion process is demonstrated in the examples below.

# **CALCULATOR HINT:**

You can use the  $y^x$  calculator function to evaluate the powers of 16.

$$356_{16} = 3 \times 16^{2} + 5 \times 16^{1} + 6 \times 16^{0}$$
  
= 768 + 80 + 6  
= 854\_{10}  
$$2AF_{16} = 2 \times 16^{2} + 10 \times 16^{1} + 15 \times 16^{0}$$
  
= 512 + 160 + 15  
= 687\_{10}

Note that in the second example, the value 10 was substituted for A and the value 15 for F in the conversion to decimal.

For practice, verify that  $1BC2_{16}$  is equal to  $7106_{10}$ .

### **Decimal-to-Hex Conversion**

Recall that we did decimal-to-binary conversion using repeated division by 2. Likewise, decimal-to-hex conversion can be done using repeated division by 16 (Figure 2-1). The following example contains two illustrations of this conversion.

#### **EXAMPLE 2-3**

(a) Convert  $423_{10}$  to hex.

Solution

$$\frac{423}{16} = 26 + \text{remainder of 7}$$

$$\frac{26}{16} = 1 + \text{remainder of 10}$$

$$\frac{1}{16} = 0 + \text{remainder of 1}$$

$$423_{10} = 1A7_{16}$$

(b) Convert  $214_{10}$  to hex.

#### Solution

$$\frac{214}{16} = 13 + \text{remainder of } 6 - \frac{13}{16} = 0 + \text{remainder of } 13 - \frac{13}{16} = 0 + \frac{13}{16} = 0 + \frac{13}{16} - \frac{13}{16} + \frac{13}{16} = 0 + \frac{13}{16} - \frac{13}{16} + \frac{13}{16} - \frac{13}{16} + \frac{13}{16} - \frac{13}{16} + \frac{13}{16} + \frac{13}{16} - \frac{13}{16} + \frac{13}{16} + \frac{13}{16} - \frac{13}{16} + \frac{13}{16} +$$

Again note that the remainders of the division processes form the digits of the hex number. Also note that any remainders that are greater than 9 are represented by the letters A through F.

# **CALCULATOR HINT:**

If a calculator is used to perform the divisions in the conversion process, the results will include a decimal fraction instead of a remainder. The remainder can be obtained by multiplying the fraction by 16. To illustrate, in Example 2-3(b), the calculator would have produced

$$\frac{214}{16} = 13.375$$

The remainder becomes  $(0.375) \times 16 = 6$ .

### **Hex-to-Binary Conversion**

The hexadecimal number system is used primarily as a "shorthand" method for representing binary numbers. It is a relatively simple matter to convert a hex number to binary. *Each* hex digit is converted to its four-bit binary equivalent (Table 2-1). This is illustrated below for 9F2<sub>16</sub>.

For practice, verify that  $BA6_{16} = 101110100110_2$ .

#### **Binary-to-Hex Conversion**

Conversion from binary to hex is just the reverse of the process above. The binary number is grouped into groups of *four* bits, and each group is converted to its equivalent hex digit. Zeros (shown shaded) are added, as needed, to complete a four-bit group.

$$1 1 1 0 1 0 0 1 1 0_{2} = \underbrace{\begin{array}{c} 0 & 0 & 1 & 1 \\ 3 & & A \\ \end{array}}_{3} \underbrace{\begin{array}{c} 1 & 0 & 1 & 0 \\ 4 & & A \\ \end{array}}_{6} \underbrace{\begin{array}{c} 0 & 1 & 1 & 0 \\ 6 & & 6 \\ \end{array}}_{6}$$

To perform these conversions between hex and binary, it is necessary to know the four-bit binary numbers (0000 through 1111) and their equivalent hex digits. Once these are mastered, the conversions can be performed quickly without the need for any calculations. This is why hex is so useful in representing large binary numbers.

For practice, verify that  $101011111_2 = 15F_{16}$ .

#### **Counting in Hexadecimal**

When counting in hex, each digit position can be incremented (increased by 1) from 0 to F. Once a digit position reaches the value F, it is reset to 0, and the

next digit position is incremented. This is illustrated in the following hex counting sequences:

- (a) 38, 39, 3A, 3B, 3C, 3D, 3E, 3F, 40, 41, 42
- (b) 6F8, 6F9, 6FA, 6FB, 6FC, 6FD, 6FE, 6FF, 700

Note that when there is a 9 in a digit position, it becomes an A when it is incremented.

With *N* hex digit positions, we can count from decimal 0 to  $16^N - 1$ , for a total of  $16^N$  different values. For example, with three hex digits, we can count from  $000_{16}$  to FFF<sub>16</sub>, which is  $0_{10}$  to  $4095_{10}$ , for a total of  $4096 = 16^3$  different values.

### **Usefulness of Hex**

Hex is often used in a digital system as sort of a "shorthand" way to represent strings of bits. In computer work, strings as long as 64 bits are not uncommon. These binary strings do not always represent a numerical value, but—as you will find out—can be some type of code that conveys nonnumerical information. When dealing with a large number of bits, it is more convenient and less error-prone to write the binary numbers in hex and, as we have seen, it is relatively easy to convert back and forth between binary and hex. To illustrate the advantage of hex representation of a binary string, suppose you had in front of you a printout of the contents of 50 memory locations, each of which was a 16-bit number, and you were checking it against a list. Would you rather check 50 numbers like this one: 0110111001100111, or 50 numbers like this one: 6E67? And which one would you be more apt to read incorrectly? It is important to keep in mind, though, that digital circuits all work in binary. Hex is simply used as a convenience for the humans involved. You should memorize the 4-bit binary pattern for each hexadecimal digit. Only then will you realize the usefulness of this tool in digital systems.

#### **EXAMPLE 2-4**

Convert decimal 378 to a 16-bit binary number by first converting to hexadecimal.

#### Solution

$$\frac{378}{16} = 23 + \text{remainder of } 10_{10} = A_{16}$$

$$\frac{23}{16} = 1 + \text{remainder of } 7$$

$$\frac{1}{16} = 0 + \text{remainder of } 1$$

Thus,  $378_{10} = 17A_{16}$ . This hex value can be converted easily to binary 000101111010. Finally, we can express  $378_{10}$  as a 16-bit number by adding four leading 0s:

$$378_{10} = 0000 \quad 0001 \quad 0111 \quad 1010_2$$

EXAMPLE 2-5

Convert B2F<sub>16</sub> to decimal.

#### Solution

$$\begin{split} B2F_{16} &= B \times 16^2 + 2 \times 16^1 + F \times 16^0 \\ &= 11 \times 256 + 2 \times 16 + 15 \\ &= 2863_{10} \end{split}$$

#### **Summary of Conversions**

Right now, your head is probably spinning as you try to keep straight all of these different conversions from one number system to another. You probably realize that many of these conversions can be done *automatically* on your calculator just by pressing a key, but it is important for you to master these conversions so that you understand the process. Besides, what happens if your calculator battery dies at a crucial time and you have no handy replacement? The following summary should help you, but nothing beats practice, practice, practice!

- 1. When converting from binary [or hex] to decimal, use the method of taking the weighted sum of each digit position.
- 2. When converting from decimal to binary [or hex], use the method of repeatedly dividing by 2 [or 16] and collecting remainders (Figure 2-1).
- 3. When converting from binary to hex, group the bits in groups of four, and convert each group into the correct hex digit.
- 4. When converting from hex to binary, convert each digit into its four-bit equivalent.

# **REVIEW QUESTIONS** 1. Convert 24CE<sub>16</sub> to decimal.

- 2. Convert 3117<sub>10</sub> to hex, then from hex to binary.
- 3. Convert 1001011110110101<sub>2</sub> to hex.
- 4. Write the next four numbers in this hex counting sequence: E9A, E9B, E9C, E9D, \_\_\_\_\_, \_\_\_\_.
- 5. Convert 3527 to binary<sub>16</sub>.
- 6. What range of decimal values can be represented by a four-digit hex number?

# 2-4 BCD CODE

When numbers, letters, or words are represented by a special group of symbols, we say that they are being encoded, and the group of symbols is called a *code*. Probably one of the most familiar codes is the Morse code, where a series of dots and dashes represents letters of the alphabet.

We have seen that any decimal number can be represented by an equivalent binary number. The group of 0s and 1s in the binary number can be thought of as a code representing the decimal number. When a decimal number is represented by its equivalent binary number, we call it **straight binary coding**. Digital systems all use some form of binary numbers for their internal operation, but the external world is decimal in nature. This means that conversions between the decimal and binary systems are being performed often. We have seen that the conversions between decimal and binary can become long and complicated for large numbers. For this reason, a means of encoding decimal numbers that combines some features of both the decimal and the binary systems is used in certain situations.

#### **Binary-Coded-Decimal Code**

If *each* digit of a decimal number is represented by its binary equivalent, the result is a code called **binary-coded-decimal** (hereafter abbreviated BCD). Since a decimal digit can be as large as 9, four bits are required to code each digit (the binary code for 9 is 1001).

To illustrate the BCD code, take a decimal number such as 874. Each *digit* is changed to its binary equivalent as follows:

8	7	4	(decimal)
$\downarrow$	$\downarrow$	$\downarrow$	
1000	0111	0100	(BCD)

As another example, let us change 943 to its BCD-code representation:

9	4	3	(decimal)
$\downarrow$	$\downarrow$	$\downarrow$	
1001	0100	0011	(BCD)

Once again, each decimal digit is changed to its straight binary equivalent. Note that four bits are *always* used for each digit.

The BCD code, then, represents each digit of the decimal number by a four-bit binary number. Clearly only the four-bit binary numbers from 0000 through 1001 are used. The BCD code does not use the numbers 1010, 1011, 1100, 1101, 1110, and 1111. In other words, only 10 of the 16 possible four-bit binary code groups are used. If any of the "forbidden" four-bit numbers ever occurs in a machine using the BCD code, it is usually an indication that an error has occurred.

# **EXAMPLE 2-6** Convert 0110100000111001 (BCD) to its decimal equivalent. Solution Divide the BCD number into four-bit groups and convert each to decimal. 0110 1000 0011 1001 6 8 3 9 **EXAMPLE 2-7** Convert the BCD number 011111000001 to its decimal equivalent. Solution 0111 1100 0001 7 1 The forbidden code group indicates an error in the BCD number

### **Comparison of BCD and Binary**

It is important to realize that BCD is not another number system like binary, decimal, and hexadecimal. In fact, it is the decimal system with each digit encoded in its binary equivalent. It is also important to understand that a BCD number is *not* the same as a straight binary number. A straight binary number takes the *complete* decimal number and represents it in binary; the BCD code converts *each* decimal *digit* to binary individually. To illustrate, take the number 137 and compare its straight binary and BCD codes:

 $\begin{array}{ll} 137_{10} = 10001001_2 & (binary) \\ 137_{10} = 0001 \ 0011 \ 0111 & (BCD) \end{array}$ 

The BCD code requires 12 bits, while the straight binary code requires only eight bits to represent 137. BCD requires more bits than straight binary to represent decimal numbers of more than one digit because BCD does not use all possible four-bit groups, as pointed out earlier, and is therefore somewhat inefficient.

The main advantage of the BCD code is the relative ease of converting to and from decimal. Only the four-bit code groups for the decimal digits 0 through 9 need to be remembered. This ease of conversion is especially important from a hardware standpoint because in a digital system, it is the logic circuits that perform the conversions to and from decimal.

REVIEW QUESTIONS	<ol> <li>Represent the decimal value 178 by its straight binary equivalent. Then encode the same decimal number using BCD.</li> <li>How more bits are provided to provide the size of the straight decimal number.</li> </ol>
	<ol> <li>How many bits are required to represent an eight-digit decimal number in BCD?</li> <li>What is an advantage of encoding a decimal number in BCD rather than</li> </ol>
	in straight binary? What is a disadvantage?

# 2-5 THE GRAY CODE

Digital systems operate at very fast speeds and respond to changes that occur in the digital inputs. Just as in life, when multiple input conditions are changing at the same time, the situation can be misinterpreted and cause an erroneous reaction. When you look at the bits in a binary count sequence, it is clear that there are often several bits that must change states at the same time. For example, consider when the three-bit binary number for 3 changes to 4: all three bits must change state.

In order to reduce the likelihood of a digital circuit misinterpreting a changing input, the **Gray code** has been developed as a way to represent a sequence of numbers. The unique aspect of the Gray code is that only one bit ever changes between two successive numbers in the sequence. Table 2-2 shows the translation between three-bit binary and Gray code values. To convert binary to Gray, simply start on the most significant bit and use it as the Gray MSB as shown in Figure 2-2(a). Now compare the MSB binary with the next binary bit (B1). If they are the same, then G1 = 0. If they are different, then G1 = 1. G0 can be found by comparing B1 with B0.

B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	G <sub>2</sub>	G <sub>1</sub>	G <sub>0</sub>
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

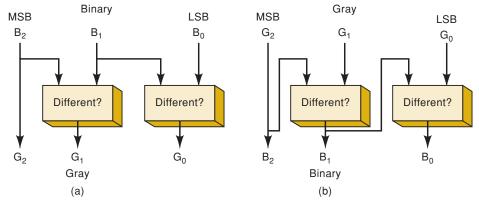
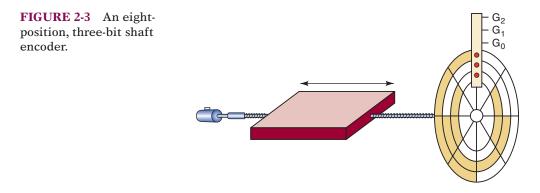


FIGURE 2-2 Converting (a) binary to Gray and (b) Gray to binary.

Conversion from Gray code back into binary is shown in Figure 2-2(b). Note that the MSB in Gray is always the same as the MSB in binary. The next binary bit is found by comparing the *binary* bit to the left with the *corresponding Gray code bit*. Similar bits produce a 0 and differing bits produce a 1. The most common application of the Gray code is in shaft position encoders as shown in Figure 2-3. These devices produce a binary value that represents the position of a rotating mechanical shaft. A practical shaft encoder would use many more bits than just three and divide the rotation into many more segments than eight, so that it could detect much smaller increments of rotation.



36

**TABLE 2-2**Three-bitbinary and Gray codeequivalents.

**REVIEW QUESTIONS** 

- 1. Convert the number 0101 (binary) to its Gray code equivalent.
- 2. Convert 0101 (Gray code) to its binary number equivalent.

# 2-6 PUTTING IT ALL TOGETHER

Table 2-3 gives the representation of the decimal numbers 1 through 15 in the binary and hex number systems and also in the BCD and Gray codes. Examine it carefully and make sure you understand how it was obtained. Especially note how the BCD representation always uses four bits for each decimal digit.

Decimal	Binary	Hexadecimal	BCD	GRAY
0	0	0	0000	0000
1	1	1	0001	0001
2	10	2	0010	0011
3	11	3	0011	0010
4	100	4	0100	0110
5	101	5	0101	0111
6	110	6	0110	0101
7	111	7	0111	0100
8	1000	8	1000	1100
9	1001	9	1001	1101
10	1010	А	0001 0000	1111
11	1011	В	0001 0001	1110
12	1100	С	0001 0010	1010
13	1101	D	0001 0011	1011
14	1110	E	0001 0100	1001
15	1111	F	0001 0101	1000

#### TABLE 2-3

# 2-7 THE BYTE, NIBBLE, AND WORD

#### **Bytes**

Most microcomputers handle and store binary data and information in groups of eight bits, so a special name is given to a string of eight bits: it is called a **byte**. A byte always consists of eight bits, and it can represent any of numerous types of data or information. The following examples will illustrate.

### EXAMPLE 2-8

How many bytes are in a 32-bit string (a string of 32 bits)?

#### Solution

32/8 = 4, so there are **four** bytes in a 32-bit string.

EXAMPLE 2-9	What is the largest decimal value that can be represented in binary usi two bytes?
	Solution Two bytes is 16 bits, so the largest binary value will be equivalent to decime $2^{16} - 1 = 65,535$ .
EXAMPLE 2-10	How many bytes are needed to represent the decimal value 846,569 in BC
	Solution
	Each decimal digit converts to a four-bit BCD code. Thus, a six-digit decim number requires 24 bits. These 24 bits are equal to <b>three</b> bytes. This is di

 $\underbrace{\begin{array}{c} 8 & 4 & 6 & 5 & 6 & 9 \\ 1000 & 0100 & 0110 & 0101 \\ 0110 & 0101 & 0110 & 1001 \\ 0110 & 0101 & 0110 & 1001 \\ 0110 & 1001 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0101 & 0101 \\ 0110 & 0100 & 0100 \\ 0110 & 0100 & 0100 \\ 0110 & 0100 & 0100 \\ 0110 & 0100 & 0100 \\ 0110 & 0100 & 000 \\ 0110 & 0100 & 000 \\ 0110 & 0100 & 000 \\ 0110 & 0100 & 000 \\ 0110 & 0100 & 000 \\ 0100 & 000 & 000 \\ 000 & 0$ 

#### **Nibbles**

grammed below.

Binary numbers are often broken down into groups of four bits, as we have seen with BCD codes and hexadecimal number conversions. In the early days of digital systems, a term caught on to describe a group of four bits. Because it is half as big as a byte, it was named a **nibble**. The following examples illustrate the use of this term.

EXAMPLE 2-11	How many nibbles are in a byte?
	Solution 2
EXAMPLE 2-12	What is the hex value of the least significant nibble of the binary number 1001 0101?
	Solution

1001 0101

The least significant nibble is 0101 = 5.

#### Words

Bits, nibbles, and bytes are terms that represent a fixed number of binary digits. As systems have grown over the years, their capacity (appetite?) for

handling binary data has also grown. A **word** is a group of bits that represents a certain unit of information. The size of the word depends on the size of the data pathway in the system that uses the information. The **word size** can be defined as the number of bits in the binary word that a digital system operates on. For example, the computer in your microwave oven can probably handle only one byte at a time. It has a word size of eight bits. On the other hand, the personal computer on your desk can handle eight bytes at a time, so it has a word size of 64 bits.

# **REVIEW QUESTIONS** 1. How many bytes are needed to represent 235<sub>10</sub> in binary?

- 2. What is the largest decimal value that can be represented in BCD using two bytes?
- 3. How many hex digits can a nibble represent?
- 4. How many nibbles are in one BCD digit?

# 2-8 ALPHANUMERIC CODES

In addition to numerical data, a computer must be able to handle nonnumerical information. In other words, a computer should recognize codes that represent letters of the alphabet, punctuation marks, and other special characters as well as numbers. These codes are called **alphanumeric codes**. A complete alphanumeric code would include the 26 lowercase letters, 26 uppercase letters, 10 numeric digits, 7 punctuation marks, and anywhere from 20 to 40 other characters, such as +, /, #, %, \*, and so on. We can say that an alphanumeric code represents all of the various characters and functions that are found on a computer keyboard.

# ASCII Code

The most widely used alphanumeric code is the **American Standard Code** for Information Interchange (ASCII). The ASCII (pronounced "askee") code is a seven-bit code, and so it has  $2^7 = 128$  possible code groups. This is more than enough to represent all of the standard keyboard characters as well as the control functions such as the (RETURN) and (LINEFEED) functions. Table 2-4 shows a listing of the standard seven-bit ASCII code. The table gives the hexadecimal and decimal equivalents. The seven-bit binary code for each character can be obtained by converting the hex value to binary.

#### **EXAMPLE 2-13**

Use Table 2-4 to find the seven-bit ASCII code for the backslash character (\).

#### Solution

The hex value given in Table 2-4 is 5C. Translating each hex digit into fourbit binary produces 0101 1100. The lower seven bits represent the ASCII code for \, or 1011100. 40

Character	HEX	Decimal	Character	HEX	Decimal	Character	HEX	Decimal	Character	HEX	Decima
NUL (null)	0	0	Space	20	32	@	40	64		60	96
Start Heading	1	1	!	21	33	Α	41	65	а	61	97
Start Text	2	2	"	22	34	В	42	66	b	62	98
End Text	3	3	#	23	35	С	43	67	с	63	99
End Transmit.	4	4	\$	24	36	D	44	68	d	64	100
Enquiry	5	5	%	25	37	E	45	69	е	65	101
Acknowlege	6	6	&	26	38	F	46	70	f	66	102
Bell	7	7	`	27	39	G	47	71	g	67	103
Backspace	8	8	(	28	40	н	48	72	h	68	104
Horiz. Tab	9	9	)	29	41	I	49	73	i	69	105
Line Feed	А	10	*	2A	42	J	4A	74	j	6A	106
Vert. Tab	В	11	+	2B	43	К	4B	75	k	6B	107
Form Feed	С	12	3	2C	44	L	4C	76	I	6C	108
Carriage Return	D	13	-	2D	45	М	4D	77	m	6D	109
Shift Out	Е	14		2E	46	N	4E	78	n	6E	110
Shift In	F	15	/	2F	47	0	4F	79	о	6F	111
Data Link Esc	10	16	0	30	48	Р	50	80	р	70	112
Direct Control 1	11	17	1	31	49	Q	51	81	q	71	113
Direct Control 2	12	18	2	32	50	R	52	82	r	72	114
Direct Control 3	13	19	3	33	51	S	53	83	S	73	115
Direct Control 4	14	20	4	34	52	Т	54	84	t	74	116
Negative ACK	15	21	5	35	53	U	55	85	u	75	117
Synch Idle	16	22	6	36	54	V	56	86	v	76	118
End Trans Block	17	23	7	37	55	W	57	87	w	77	119
Cancel	18	24	8	38	56	Х	58	88	х	78	120
End of Medium	19	25	9	39	57	Y	59	89	У	79	121
Substitue	1A	26	:	ЗA	58	Z	5A	90	Z	7A	122
Escape	1B	27	,	3B	59	[	5B	91	{	7B	123
Form separator	1C	28	<	ЗC	60	١	5C	92		7C	124
Group separator	1D	29	=	3D	61	]	5D	93	}	7D	125
Record Separator	1E	30	>	3E	62	^	5E	94	~	7E	126
Unit Separator	1F	31	?	3F	63	_	5F	95	Delete	7F	127

TABLE 2-4 Standard ASCII codes.

The ASCII code is used for the transfer of alphanumeric information between a computer and the external devices such as a printer or another computer. A computer also uses ASCII internally to store the information that an operator types in at the computer's keyboard. The following example illustrates this.

# EXAMPLE 2-14

An operator is typing in a C language program at the keyboard of a certain microcomputer. The computer converts each keystroke into its ASCII code and stores the code as a byte in memory. Determine the binary strings that will be entered into memory when the operator types in the following C statement:

if (x > 3)

#### Solution

Locate each character (including the space) in Table 2-4 and record its ASCII code.

i	69	0110	1001
$\mathbf{f}$	66	0110	0110
space	20	0010	0000
(	28	0010	1000
х	78	0111	1000
>	3E	0011	1110
3	33	0011	0011
)	29	0010	1001

Note that a 0 was added to the leftmost bit of each ASCII code because the codes must be stored as bytes (eight bits). This adding of an extra bit is called *padding with 0s.* 

#### **REVIEW QUESTIONS**

- 1. Encode the following message in ASCII code using the hex representation: "COST = \$72."
- 2. The following padded ASCII-coded message is stored in successive memory locations in a computer:

01010011 01010100 01001111 01010000

What is the message?

# 2-9 PARITY METHOD FOR ERROR DETECTION

The movement of binary data and codes from one location to another is the most frequent operation performed in digital systems. Here are just a few examples:

- The transmission of digitized voice over a microwave link
- The storage of data in and retrieval of data from external memory devices such as magnetic and optical disk
- The transmission of digital data from a computer to a remote computer over telephone lines (i.e., using a modem). This is one of the major ways of sending and receiving information on the Internet.

Whenever information is transmitted from one device (the transmitter) to another device (the receiver), there is a possibility that errors can occur such that the receiver does not receive the identical information that was sent by the transmitter. The major cause of any transmission errors is *electrical noise*, which consists of spurious fluctuations in voltage or current that are present in all electronic systems to varying degrees. Figure 2-4 is a simple illustration of a type of transmission error.

The transmitter sends a relatively noise-free serial digital signal over a signal line to a receiver. However, by the time the signal reaches the receiver,

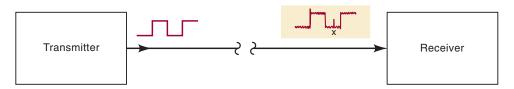


FIGURE 2-4 Example of noise causing an error in the transmission of digital data.

it contains a certain degree of noise superimposed on the original signal. Occasionally, the noise is large enough in amplitude that it will alter the logic level of the signal, as it does at point x. When this occurs, the receiver may incorrectly interpret that bit as a logic 1, which is not what the transmitter has sent.

Most modern digital equipment is designed to be relatively error-free, and the probability of errors such as the one shown in Figure 2-4 is very low. However, we must realize that digital systems often transmit thousands, even millions, of bits per second, so that even a very low rate of occurrence of errors can produce an occasional error that might prove to be bothersome, if not disastrous. For this reason, many digital systems employ some method for detection (and sometimes correction) of errors. One of the simplest and most widely used schemes for error detection is the **parity method**.

#### **Parity Bit**

A **parity bit** is an extra bit that is attached to a code group that is being transferred from one location to another. The parity bit is made either 0 or 1, depending on the number of 1s that are contained in the code group. Two different methods are used.

In the *even-parity* method, the value of the parity bit is chosen so that the total number of 1s in the code group (including the parity bit) is an *even* number. For example, suppose that the group is 1000011. This is the ASCII character "C." The code group has *three* 1s. Therefore, we will add a parity bit of 1 to make the total number of 1s an even number. The *new* code group, *including the parity bit*, thus becomes

1 1 0 0 0 0 1 1 ↑ \_\_\_\_\_\_ added parity bit\*

If the code group contains an even number of 1s to begin with, the parity bit is given a value of 0. For example, if the code group were 1000001 (the ASCII code for "A"), the assigned parity bit would be 0, so that the new code, *including the parity bit*, would be 01000001.

The *odd-parity* method is used in exactly the same way except that the parity bit is chosen so the total number of 1s (including the parity bit) is an *odd* number. For example, for the code group 1000001, the assigned parity bit would be a 1. For the code group 1000011, the parity bit would be a 0.

Regardless of whether even parity or odd parity is used, the parity bit becomes an actual part of the code word. For example, adding a parity bit to

<sup>\*</sup>The parity bit can be placed at either end of the code group, but it is usually placed to the left of the MSB.

the seven-bit ASCII code produces an eight-bit code. Thus, the parity bit is treated just like any other bit in the code.

The parity bit is issued to detect any *single-bit* errors that occur during the transmission of a code from one location to another. For example, suppose that the character "A" is being transmitted and *odd* parity is being used. The transmitted code would be

When the receiver circuit receives this code, it will check that the code contains an odd number of 1s (including the parity bit). If so, the receiver will assume that the code has been correctly received. Now, suppose that because of some noise or malfunction the receiver actually receives the following code:

1 1 0 0 0 0 0 0

The receiver will find that this code has an *even* number of 1s. This tells the receiver that there must be an error in the code because presumably the transmitter and receiver have agreed to use odd parity. There is no way, however, that the receiver can tell which bit is in error because it does not know what the code is supposed to be.

It should be apparent that this parity method would not work if *two* bits were in error, because two errors would not change the "oddness" or "evenness" of the number of 1s in the code. In practice, the parity method is used only in situations where the probability of a single error is very low and the probability of double errors is essentially zero.

When the parity method is being used, the transmitter and the receiver must have agreement, in advance, as to whether odd or even parity is being used. There is no advantage of one over the other, although even parity seems to be used more often. The transmitter must attach an appropriate parity bit to each unit of information that it transmits. For example, if the transmitter is sending ASCII-coded data, it will attach a parity bit to each seven-bit ASCII code group. When the receiver examines the data that it has received from the transmitter, it checks each code group to see that the total number of 1s (including the parity bit) is consistent with the agreedupon type of parity. This is often called *checking the parity* of the data. In the event that it detects an error, the receiver may send a message back to the transmitter asking it to retransmit the last set of data. The exact procedure that is followed when an error is detected depends on the particular system.

#### **EXAMPLE 2-15**

Computers often communicate with other remote computers over telephone lines. For example, this is how dial-up communication over the internet takes place. When one computer is transmitting a message to another, the information is usually encoded in ASCII. What actual bit strings would a computer transmit to send the message HELLO, using ASCII with even parity?

#### Solution

First, look up the ASCII codes for each character in the message. Then for each code, count the number of 1s. If it is an even number, attach a 0 as the

MSB. If it is an odd number, attach a 1. Thus, the resulting eight-bit codes (bytes) will all have an even number of 1s (including parity).

#### **REVIEW QUESTIONS**

- 1. Attach an odd-parity bit to the ASCII code for the \$ symbol, and express the result in hexadecimal.
- 2. Attach an even-parity bit to the BCD code for decimal 69.
- 3. Why can't the parity method detect a double error in transmitted data?

# 2-10 APPLICATIONS

Here are several applications that will serve as a review of some of the concepts covered in this chapter. These applications should give a sense of how the various number systems and codes are used in the digital world. More applications are presented in the end-of-chapter problems.

#### APPLICATION 2-1

A typical CD-ROM can store 650 megabytes of digital data. Since mega =  $2^{20}$ , how many bits of data can a CD-ROM hold?

#### Solution

Remember that a byte is eight bits. Therefore, 650 megabytes is  $650 \times 2^{20} \times 8 = 5,452,595,200$  bits.

#### **APPLICATION 2-2**

In order to program many microcontrollers, the binary instructions are stored in a file on a personal computer in a special way known as Intel Hex Format. The hexadecimal information is encoded into ASCII characters so it can be displayed easily on the PC screen, printed, and easily transmitted one character at a time over a standard PC's serial COM port. One line of an Intel Hex Format file is shown below:

#### : 10002000 F7 CFFF CF1 FEF2 FEF2 A95 F1 F71 A95 D9 F7 EA

The first character sent is the ASCII code for a colon, followed by a 1. Each has an even parity bit appended as the most significant bit. A test instrument captures the binary bit pattern as it goes across the cable to the microcontroller.

(a) What should the binary bit pattern (including parity) look like? (MSB-LSB)

- (b) The value 10, following the colon, represents the total hexadecimal number of bytes that are to be loaded into the micro's memory. What is the decimal number of bytes being loaded?
- (c) The number 0020 is a four-digit hex value representing the address where the first byte is to be stored. What is the biggest address possible? How many bits would it take to represent this address?
- (d) The value of the first data byte is F7. What is the value (in binary) of the least significant nibble of this byte?

#### Solution

- (a) ASCII codes are 3A (for :) and 31 (for 1) 00111010 10110001 even parity bit \_\_\_\_\_\_
- (b)  $10 \text{ hex} = 1 \times 16 + 0 \times 1 = 16$  decimal bytes
- (c) FFFF is the biggest possible value. Each hex digit is 4 bits, so we need 16 bits.
- (d) The least significant nibble (4 bits) is represented by hex 7. In binary this would be 0111.

#### **APPLICATION 2-3**

A small process-control computer uses hexadecimal codes to represent its 16-bit memory addresses.

- (a) How many hex digits are required?
- (b) What is the range of addresses in hex?
- (c) How many memory locations are there?

#### Solution

- (a) Since 4 bits convert to a single hex digit, 16/4 = 4 hex digits are needed.
- (b) The binary range is  $00000000000000_2$  to  $1111111111111111_2$ . In hex, this becomes  $0000_{16}$  to FFFF<sub>16</sub>.
- (c) With 4 hex digits, the total number of addresses is  $16^4 = 65,536$ .

#### **APPLICATION 2-4**

Numbers are entered into a microcontroller-based system in BCD, but stored in straight binary. As a programmer, you must decide whether you need a one-byte or two-byte storage location.

- (a) How many bytes do you need if the system takes a two-digit decimal entry?
- (b) What if you needed to be able to enter three digits?

#### Solution

- (a) With two digits you can enter values up to 99 (1001  $1001_{BCD}$ ). In binary this value is 01100011, which will fit into an eight-bit memory location. Thus you can use a single byte.
- (b) Three digits can represent up to 999 (1001 1001 1001). In binary this value is 1111100111 (10 bits). Thus you cannot use a single byte; you need two bytes.

#### **APPLICATION 2-5**

When ASCII characters must be transmitted between two independent systems (such as between a computer and a modem), there must be a way of telling the receiver when a new character is coming in. There is often a need to detect errors in the transmission as well. The method of transfer is called asynchronous data communication. The normal resting state of the transmission line is logic 1. When the transmitter sends an ASCII character, it must be "framed" so the receiver knows where the data begins and ends. The first bit must always be a start bit (logic 0). Next the ASCII code is sent LSB first and MSB last. After the MSB, a parity bit is appended to check for transmission errors. Finally, the transmission is ended by sending a stop bit (logic 1). A typical asynchronous transmission of a seven-bit ASCII code for the pound sign # (23 Hex) with even parity is shown in Figure 2-5.

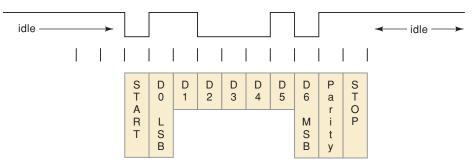


FIGURE 2-5 Asynchronous serial data with even parity.

# **SUMMARY**

- 1. The hexadecimal number system is used in digital systems and computers as an efficient way of representing binary quantities.
- 2. In conversions between hex and binary, each hex digit corresponds to four bits.
- 3. The repeated-division method is used to convert decimal numbers to binary or hexadecimal.
- 4. Using an N-bit binary number, we can represent decimal values from 0 to  $2^N 1$ .
- 5. The BCD code for a decimal number is formed by converting each digit of the decimal number to its four-bit binary equivalent.
- 6. The Gray code defines a sequence of bit patterns in which only one bit changes between successive patterns in the sequence.
- 7. A byte is a string of eight bits. A nibble is four bits. The word size depends on the system.
- 8. An alphanumeric code is one that uses groups of bits to represent all of the various characters and functions that are part of a typical computer's keyboard. The ASCII code is the most widely used alphanumeric code.
- 9. The parity method for error detection attaches a special parity bit to each transmitted group of bits.

# **IMPORTANT TERMS**

hexadecimal number	Gray code	American Standard
system	byte	Code for
straight binary	nibble	Information
coding	word	Interchange
binary-coded-decimal	word size	(ASCII)
(BCD) code	alphanumeric code	parity method
		parity bit

# PROBLEMS

#### SECTIONS 2-1 AND 2-2

2-1. Convert these binary numbers to decimal.

(a)*10110	(d) 01101011	(g)*1111010111		
(b) 10010101	(e)*11111111	(h) 11011111		
(c)*100100001001	(f) 01101111			
Construct the following of a size of a size of the big and				

2-2. Convert the following decimal values to binary.

(a)*37	(d) 1000	(g)*205
(b) 13	(e)*77	(h) 2133
(c) <b>*</b> 189	(f) 390	(i)* 511

2-3. What is the largest decimal value that can be represented by (a)\* an eight-bit binary number? (b) A 16-bit number?

#### **SECTION 2-4**

2-4. Convert each hex number to its decimal equivalent.

(a)*743	(d) 2000	(g)*7FF
(b) 36	(e) <b>*</b> 165	(h) 1204
(c)*37FD	(f) ABCD	

2-5. Convert each of the following decimal numbers to hex.

(a) <b>*</b> 59	(d) 1024	(g) <b>*</b> 65,536
(b) 372	(e)*771	(h) 255
(c)*919	(f) 2313	

- 2-6. Convert each of the hex values from Problem 2-4 to binary.
- 2-7. Convert the binary numbers in Problem 2-1 to hex.
- 2-8. List the hex numbers in sequence from  $195_{16}$  to  $280_{16}$ .
- 2-9. When a large decimal number is to be converted to binary, it is sometimes easier to convert it first to hex, and then from hex to binary. Try this procedure for  $2133_{10}$  and compare it with the procedure used in Problem 2-2(h).
- 2-10. How many hex digits are required to represent decimal numbers up to 20,000?
- 2-11. Convert these hex values to decimal.

(a)*92	(d) ABCD	(g)*2C0
(b) 1A6	(e)*000F	(h) 7FF
(c)*37FD	(f) 55	

<sup>\*</sup>Answers to problems marked with an asterisk can be found in the back of the text.

#### 2-12. Convert these decimal values to hex.

(a)*75	(d) 24	(g)*25,619
(b) 314	(e) <b>*</b> 7245	(h) 4095
(c)*2048	(f) 498	

2-13. Take each four-bit binary number in the order they are written and write the equivalent hex digit without performing a calculation by hand or by calculator.

(a) 1001	(e) 1111	(i) 1011	(m) 0001
(b) 1101	(f) 0010	(j) 1100	(n) 0101
(c) 1000	(g) 1010	(k) 0011	(o) 0111
(d) 0000	(h) 1001	(1) 0100	(p) 0110

2-14. Take each hex digit and write its four-bit binary value without performing any calculations by hand or by calculator.

(a) 6	(e) 4	(i) 9	(m) 0
(b) 7	(f) 3	(j) A	(n) 8
(c) 5	(g) C	(k) 2	(o) D
(d) 1	(h) B	(l) F	(p) 9

- 2-15.\* Convert the binary numbers in Problem 2-1 to hexadecimal.
- 2-16.\* Convert the hex values in Problem 2-11 to binary.
- 2-17.\* List the hex numbers in sequence from 280 to 2A0.
- 2-18. How many hex digits are required to represent decimal numbers up to 1 million?

#### **SECTION 2-5**

2-19. Encode these decimal numbers in BCD.

(a)*47	(d) 6727	(g <b>*</b> 89,627
(b) 962	(e)*13	(h) 1024
(c) <b>*</b> 187	(f) 529	

- 2-20. How many bits are required to represent the decimal numbers in the range from 0 to 999 using (a) straight binary code? (b) Using BCD code?
- 2-21. The following numbers are in BCD. Convert them to decimal.

(a)*1001011101010010	(d) 0111011101110101
(b) 000110000100	(e)*010010010010
(c)*011010010101	(f) 010101010101

#### **SECTION 2-7**

- 2-22.\*(a) How many bits are contained in eight bytes?
  - (b) What is the largest hex number that can be represented in four bytes?
  - (c) What is the largest BCD-encoded decimal value that can be represented in three bytes?
- 2-23. (a) Refer to Table 2-4. What is the most significant nibble of the ASCII code for the letter X?
  - (b) How many nibbles can be stored in a 16-bit word?
  - (c) How many bytes does it take to make up a 24-bit word?

#### SECTIONS 2-8 AND 2-9

- 2-24. Represent the statement "X =  $3 \times Y$ " in ASCII code. Attach an odd-parity bit.
- 2-25.\*Attach an *even*-parity bit to each of the ASCII codes for Problem 2-24, and give the results in hex.
- 2-26. The following bytes (shown in hex) represent a person's name as it would be stored in a computer's memory. Each byte is a padded ASCII code. Determine the name of each person.

(a)\*42 45 4E 20 53 4D 49 54 48

- (b) 4A 6F 65 20 47 72 65 65 6E
- 2-27. Convert the following decimal numbers to BCD code and then attach an *odd* parity bit.

(a) <b>*</b> 74	(c) <b>*</b> 8884	(e)*165
(b) 38	(d) 275	(f) 9201

- 2-28\*In a certain digital system, the decimal numbers from 000 through 999 are represented in BCD code. An *odd*-parity bit is also included at the end of each code group. Examine each of the code groups below, and assume that each one has just been transferred from one location to another. Some of the groups contain errors. Assume that *no more than* two errors have occurred for each group. Determine which of the code groups have a single error and which of them *definitely* have a double error. (*Hint:* Remember that this is a BCD code.)
  - (a) 1001010110000
    - \_\_\_\_\_ parity bit
  - (b) 0100011101100
  - (c) 0111110000011
  - (d) 1000011000101
- 2-29. Suppose that the receiver received the following data from the transmitter of Example 2-16:

010010	0 0
110001	01
110011	0 0
110010	0 0
110011	0 0

What errors can the receiver determine in these received data?

#### **DRILL QUESTIONS**

- 2-30.\*Perform each of the following conversions. For some of them, you may want to try several methods to see which one works best for you. For example, a binary-to-decimal conversion may be done directly, or it may be done as a binary-to-hex conversion followed by a hex-todecimal conversion.
  - (a)  $1417_{10} = \__2$
  - (b)  $255_{10} = \__2$
  - (c)  $11010001_2 = \__{10}$
  - (d)  $1110101000100111_2 = \____10$

- (e)  $2497_{10} = \__{16}$
- (f)  $511_{10} = \_$  (BCD)
- (g)  $235_{16} = \__{10}$
- (h)  $4316_{10} = \__{16}$
- (i)  $7A9_{16} = \__{10}$
- (j)  $3E1C_{16} = \__{10}$
- (k)  $1600_{10} = \__{16}$
- (1)  $38,187_{10} = \__{16}$
- (m)  $865_{10} = \_$  (BCD)
- (n) 100101000111 (BCD) = \_\_\_\_\_ 10
- (o)  $465_{16} = \__2$
- (p)  $B34_{16} = \__2$
- (q)  $01110100 (BCD) = \____2$
- (r)  $111010_2 = \_$  (BCD)
- 2-31.\*Represent the decimal value 37 in each of the following ways.
  - (a) straight binary
  - (b) BCD
  - (c) hex
  - (d) ASCII (i.e., treat each digit as a character)
- 2-32.\*Fill in the blanks with the correct word or words.
  - (a) Conversion from decimal to \_\_\_\_\_ requires repeated division by 16.
  - (b) Conversion from decimal to binary requires repeated division by
  - (c) In the BCD code, each \_\_\_\_\_ is converted to its four-bit binary equivalent.
  - (d) The \_\_\_\_\_ code has the characteristic that only one bit changes in going from one step to the next.
  - (e) A transmitter attaches a \_\_\_\_\_ to a code group to allow the receiver to detect \_\_\_\_\_.
  - (f) The \_\_\_\_\_ code is the most common alphanumeric code used in computer systems.
  - (g) \_\_\_\_\_ is often used as a convenient way to represent large binary numbers.
  - (h) A string of eight bits is called a \_\_\_\_\_.
- 2-33. Write the binary number that results when each of the following numbers is incremented by one.
  - (a)\*0111 (b) 010011 (c) 1011
- 2-34. Decrement each binary number.

(a)\*1110 (b) 101000 (c) 1110

2-35. Write the number that results when each of the following is incremented.

(a)*7779 <sub>16</sub>	$(c)*OFFF_{16}$	(e)*9FF <sub>16</sub>
(b) 9999 <sub>16</sub>	(d) 2000 <sub>16</sub>	(f) 100A <sub>16</sub>

2-36.\*Repeat Problem 2-35 for the decrement operation.

#### **CHALLENGING EXERCISES**

- 2-37.\*In a microcomputer, the *addresses* of memory locations are binary numbers that identify each memory circuit where a byte is stored. The number of bits that make up an address depends on how many memory locations there are. Since the number of bits can be very large, the addresses are often specified in hex instead of binary.
  - (a) If a microcomputer uses a 20-bit address, how many different memory locations are there?
  - (b) How many hex digits are needed to represent the address of a memory location?
  - (c) What is the hex address of the 256th memory location? (*Note:* The first address is always 0.)
- 2-38. In an audio CD, the audio voltage signal is typically sampled about 44,000 times per second, and the value of each sample is recorded on the CD surface as a binary number. In other words, each recorded binary number represents a single voltage point on the audio signal waveform.
  - (a) If the binary numbers are six bits in length, how many different voltage values can be represented by a single binary number? Repeat for eight bits and ten bits.
  - (b) If ten-bit numbers are used, how many bits will be recorded on the CD in 1 second?
  - (c) If a CD can typically store 5 billion bits, how many seconds of audio can be recorded when ten-bit numbers are used?
- 2-39.\*A black-and-white digital camera lays a fine grid over an image and then measures and records a binary number representing the level of gray it sees in each cell of the grid. For example, if four-bit numbers are used, the value of black is set to 0000 and the value of white to 1111, and any level of gray is somewhere between 0000 and 1111. If six-bit numbers are used, black is 000000, white is 111111, and all grays are between the two.

Suppose we wanted to distinguish among 254 different levels of gray within each cell of the grid. How many bits would we need to use to represent these levels?

- 2-40. A 3-Megapixel digital camera stores an eight-bit number for the brightness of each of the primary colors (red, green, blue) found in each picture element (pixel). If every bit is stored (no data compression), how many pictures can be stored on a 128-Megabyte memory card? (Note: In digital systems, Mega means 2<sup>20</sup>.)
- 2-41. Construct a table showing the binary, hex, and BCD representations of all decimal numbers from 0 to 15. Compare your table with Table 2-3.

# ANSWERS TO SECTION REVIEW QUESTIONS

#### SECTION 2-1

1. 2267 2. 32768

#### SECTION 2-2

1. 1010011 2. 1011011001 3. 20 bits

#### **SECTION 2-3**

1. 94222. C2D; 1100001011013. 97B54. E9E, E9F, EA0, EA15. 110101001001116. 0 to 65,535

#### **SECTION 2-4**

1. 10110010<sub>2</sub>; 000101111000 (BCD) 2. 32 3. Advantage: ease of conversion. Disadvantage: BCD requires more bits.

#### **SECTION 2-5**

1.0111 2.0110

#### **SECTION 2-7**

1. One 2. 9999 3. One 4. One

#### **SECTION 2-8**

1. 43, 4F, 53, 54, 20, 3D, 20, 24, 37, 32 2. STOP

#### **SECTION 2-9**

1. A4 2. 001101001 3. Two errors in the data would not change the oddness or evenness of the number of 1s in the data.