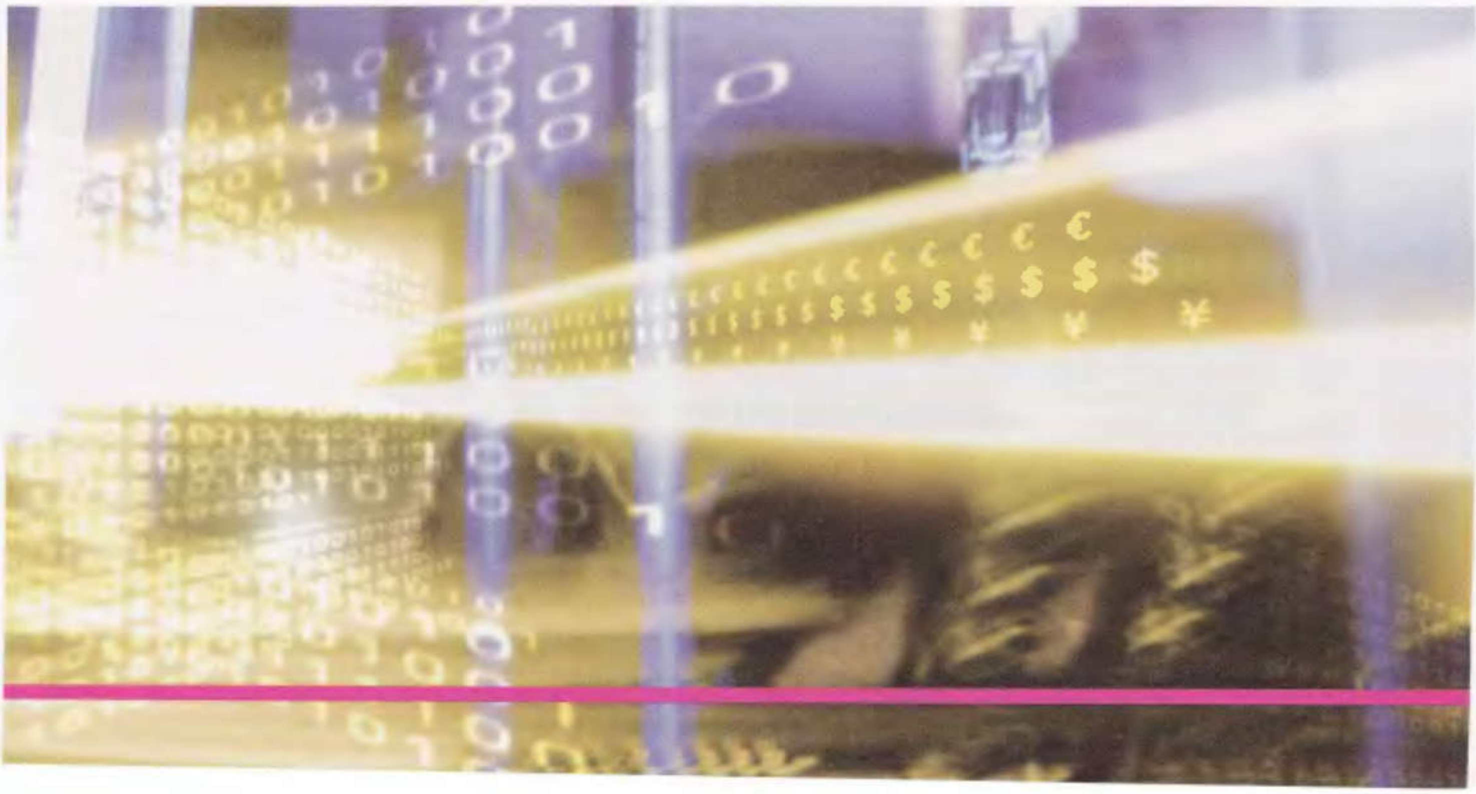
2

NUMBER SYSTEMS, OPERATIONS, AND CODES

CHAPTER OUTLINE



CHAPTER OBJECTIVES

- Review the decimal number system
- Count in the binary number system
- Convert from decimal to binary and from binary to decimal
- Apply arithmetic operations to binary numbers
- Determine the 1's and 2's complements of a binary number
- Express signed binary numbers in sign-magnitude, 1's complement, 2's complement, and floating-point format
- Carry out arithmetic operations with signed binary numbers
- Convert between the binary and hexadecimal number systems
- Add numbers in hexadecimal form
- Convert between the binary and octal number systems
- Express decimal numbers in binary coded decimal (BCD) form
- Add BCD numbers
- Convert between the binary system and the Gray code
- Interpret the American Standard Code for Information Interchange (ASCII)
- Explain how to detect and correct code errors

KEY TERMS

- LSB
- MSB
- Byte
- Floating-point number
- Hexadecimal
- Octal
- BCD
- Alphanumeric
- ASCII
- Parity
- Hamming code

INTRODUCTION

The binary number system and digital codes are fundamental to computers and to digital electronics in general. In this chapter, the binary number system and its relationship to other number systems such as decimal, hexadecimal, and octal is presented. Arithmetic operations with binary numbers are covered to provide a basis for understanding how computers and many other types of digital systems work. Also, digital codes such as binary coded decimal (BCD), the Gray code, and the ASCII are covered. The parity method for detecting errors in codes is introduced and a method for correcting errors is described. The tutorials on the use of the calculator in certain operations are based on the TI-86 graphics calculator and the TI-36X calculator. The procedures shown may vary on other types.

WWW. VISIT THE COMPANION WEBSITE

Study aids for this chapter are available at

http://www.prenhall.com/floyd

2–1 DECIMAL NUMBERS

You are familiar with the decimal number system because you use decimal numbers every day. Although decimal numbers are commonplace, their weighted structure is often not understood. In this section, the structure of decimal numbers is reviewed. This review will help you more easily understand the structure of the binary number system, which is important in computers and digital electronics.

After completing this section, you should be able to

- Explain why the decimal number system is a weighted system
- Explain how powers of ten are used in the decimal system
- Determine the weight of each digit in a decimal number

The decimal number system has ten digits.

In the **decimal** number system each of the ten digits, 0 through 9, represents a certain quantity. As you know, the ten symbols (**digits**) do not limit you to expressing only ten different quantities because you use the various digits in appropriate positions within a number to indicate the magnitude of the quantity. You can express quantities up through nine before running out of digits; if you wish to express a quantity greater than nine, you use two or more digits, and the position of each digit within the number tells you the magnitude it represents. If, for example, you wish to express the quantity twenty-three, you use (by their respective positions in the number) the digit 2 to represent the quantity twenty and the digit 3 to represent the quantity three, as illustrated below.

The digit 2 has a weight of 10 in this position.

The digit 3 has a weight of 1 in this position.

$$23$$

$$2 \times 10 + 3 \times 1$$

$$20 + 3$$

$$23$$

The decimal number system has a base of 10.

The position of each digit in a decimal number indicates the magnitude of the quantity represented and can be assigned a **weight.** The weights for whole numbers are positive powers of ten that increase from right to left, beginning with $10^0 = 1$.

$$\ldots 10^5\ 10^4\ 10^3\ 10^2\ 10^1\ 10^0$$

For fractional numbers, the weights are negative powers of ten that decrease from left to right beginning with 10^{-1}.

The value of a digit is determined by its position in the number.

$$10^2\ 10^1\ 10^0.10^{-1}\ 10^{-2}\ 10^{-3}\ldots$$

Decimal point

The value of a decimal number is the sum of the digits after each digit has been multiplied by its weight, as Examples 2–1 and 2–2 illustrate.

EXAMPLE 2–1

Express the decimal number 47 as a sum of the values of each digit.

Solution The digit 4 has a weight of 10, which is 10^1, as indicated by its position. The digit 7 has a weight of 1, which is 10^0, as indicated by its position.

$$47 = (4 \times 10^1) + (7 \times 10^0)$$
$$= (4 \times 10) + (7 \times 1) = \mathbf{40 + 7}$$

*Related Problem** Determine the value of each digit in 939.

*Answers are at the end of the chapter.

EXAMPLE 2–2

Express the decimal number 568.23 as a sum of the values of each digit.

Solution The whole number digit 5 has a weight of 100, which is 10^2, the digit 6 has a weight of 10, which is 10^1, the digit 8 has a weight of 1, which is 10^0, the fractional digit 2 has a weight of 0.1, which is 10^{-1}, and the fractional digit 3 has a weight of 0.01, which is 10^{-2}.

$$568.23 = (5 \times 10^2) + (6 \times 10^1) + (8 \times 10^0) + (2 \times 10^{-1}) + (3 \times 10^{-2})$$
$$= (5 \times 100) + (6 \times 10) + (8 \times 1) + (2 \times 0.1) + (3 \times 0.01)$$
$$= \mathbf{500} + \mathbf{60} + \mathbf{8} + \mathbf{0.2} + \mathbf{0.03}$$

Related Problem Determine the value of each digit in 67.924.

CALCULATOR TUTORIAL

Powers of Ten

Example Find the value of 10^3.

TI-86 **Step 1.** 2nd LOG (10^x)

Step 2. 3

Step 3. ENTER

10 ^ 3
1000

TI-36X **Step 1.** 1 0 y^x

Step 2. 3 =

1000

SECTION 2–1 REVIEW

Answers are at the end of the chapter.

1. What weight does the digit 7 have in each of the following numbers?
   (a) 1370 **(b)** 6725 **(c)** 7051 **(d)** 58.72
2. Express each of the following decimal numbers as a sum of the products obtained by multiplying each digit by its appropriate weight:
   (a) 51 **(b)** 137 **(c)** 1492 **(d)** 106.58

2–2 BINARY NUMBERS

The binary number system is another way to represent quantities. It is less complicated than the decimal system because it has only two digits. The decimal system with its ten digits is a base-ten system; the binary system with its two digits is a base-two system. The two binary digits (bits) are 1 and 0. The position of a 1 or 0 in a binary number indicates its weight, or value within the number, just as the position of a decimal digit determines the value of that digit. The weights in a binary number are based on powers of two.

After completing this section, you should be able to

■ Count in binary ■ Determine the largest decimal number that can be represented by a given number of bits ■ Convert a binary number to a decimal number

Counting in Binary

The binary number system has two digits (bits).

To learn to count in the binary system, first look at how you count in the decimal system. You start at zero and count up to nine before you run out of digits. You then start another digit position (to the left) and continue counting 10 through 99. At this point you have exhausted all two-digit combinations, so a third digit position is needed to count from 100 through 999.

A comparable situation occurs when you count in binary, except that you have only two digits, called *bits*. Begin counting: 0, 1. At this point you have used both digits, so include another digit position and continue: 10, 11. You have now exhausted all combinations of two digits, so a third position is required. With three digit positions you can continue to count: 100, 101, 110, and 111. Now you need a fourth digit position to continue, and so on. A binary count of zero through fifteen is shown in Table 2–1. Notice the patterns with which the 1s and 0s alternate in each column.

The binary number system has a base of 2.

► TABLE 2–1

DECIMAL NUMBER	BINARY NUMBER			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

As you have seen in Table 2–1, four bits are required to count from zero to 15. In general, with n bits you can count up to a number equal to $2^n - 1$.

The value of a bit is determined by its position in the number.

$$\text{Largest decimal number} = 2^n - 1$$

For example, with five bits ($n = 5$) you can count from zero to thirty-one.

$$2^5 - 1 = 32 - 1 = 31$$

With six bits ($n = 6$) you can count from zero to sixty-three.

$$2^6 - 1 = 64 - 1 = 63$$

A table of powers of two is given in Appendix A.

CALCULATOR TUTORIAL

Powers of Two

Example Find the value of 2^5.

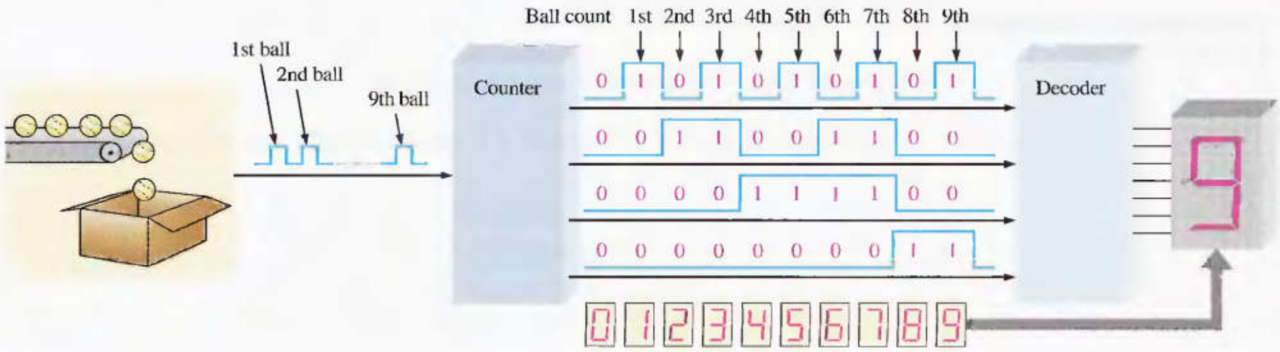
TI-86 **Step 1.** 2 ^
Step 2. 5 ENTER
Display: 2^5 32

TI-36X **Step 1.** 2 y^x
Step 2. 5 =
Display: 32

An Application

Learning to count in binary will help you to basically understand how digital circuits can be used to count events. This can be anything from counting items on an assembly line to counting operations in a computer. Let's take a simple example of counting tennis balls going into a box from a conveyor belt. Assume that nine balls are to go into each box.

The counter in Figure 2–1 counts the pulses from a sensor that detects the passing of a ball and produces a sequence of logic levels (digital waveforms) on each of its four parallel outputs. Each set of logic levels represents a 4-bit binary number (HIGH = 1 and LOW = 0), as indicated. As the decoder receives these waveforms, it decodes each set of four bits and converts it to the corresponding decimal number in the 7-segment display. When the counter gets to the binary state of 1001, it has counted nine tennis balls, the display shows decimal 9, and a new box is moved under the conveyor. Then the counter goes back to its zero state (0000), and the process starts over. (The number 9 was used only in the interest of single-digit simplicity.)



▲ **FIGURE 2–1**

Illustration of a simple binary counting application.

The Weighting Structure of Binary Numbers

The weight or value of a bit increases from right to left in a binary number.

A binary number is a weighted number. The right-most bit is the **LSB** (least significant bit) in a binary whole number and has a weight of $2^0 = 1$. The weights increase from right to left by a power of two for each bit. The left-most bit is the **MSB** (most significant bit); its weight depends on the size of the binary number.

Fractional numbers can also be represented in binary by placing bits to the right of the binary point, just as fractional decimal digits are placed to the right of the decimal point. The left-most bit is the MSB in a binary fractional number and has a weight of $2^{-1} = 0.5$. The fractional weights decrease from left to right by a negative power of two for each bit.

The weight structure of a binary number is

$$2^{n-1} \ldots 2^3\, 2^2\, 2^1\, 2^0 \,.\, 2^{-1}\, 2^{-2} \ldots 2^{-n}$$

Binary point

COMPUTER NOTE

Computers use binary numbers to select memory locations. Each location is assigned a unique number called an *address*. Some Pentium microprocessors, for example, have 32 address lines which can select 2^{32} (4,294,967,296) unique locations.

where n is the number of bits from the binary point. Thus, all the bits to the left of the binary point have weights that are positive powers of two, as previously discussed for whole numbers. All bits to the right of the binary point have weights that are negative powers of two, or fractional weights.

The powers of two and their equivalent decimal weights for an 8-bit binary whole number and a 6-bit binary fractional number are shown in Table 2–2. Notice that the weight doubles for each positive power of two and that the weight is halved for each negative power of two. You can easily extend the table by doubling the weight of the most significant positive power of two and halving the weight of the least significant negative power of two; for example, $2^9 = 512$ and $2^{-7} = 0.0078125$.

TABLE 2–2

Binary weights.

POSITIVE POWERS OF TWO (WHOLE NUMBERS)									NEGATIVE POWERS OF TWO (FRACTIONAL NUMBER)					
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
256	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64
									0.5	0.25	0.125	0.0625	0.03125	0.015625

Binary-to-Decimal Conversion

Add the weights of all 1s in a binary number to get the decimal value.

The decimal value of any binary number can be found by adding the weights of all bits that are 1 and discarding the weights of all bits that are 0.

EXAMPLE 2–3

Convert the binary whole number 1101101 to decimal.

Solution Determine the weight of each bit that is a 1, and then find the sum of the weights to get the decimal number.

Weight: $2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

Binary number: 1 1 0 1 1 0 1

$$\begin{aligned} 1101101 &= 2^6 + 2^5 + 2^3 + 2^2 + 2^0 \\ &= 64 + 32 + 8 + 4 + 1 = \mathbf{109} \end{aligned}$$

Related Problem Convert the binary number 10010001 to decimal.

EXAMPLE 2–4

Convert the fractional binary number 0.1011 to decimal.

Solution Determine the weight of each bit that is a 1, and then sum the weights to get the decimal fraction.

Weight:	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Binary number:	0 . 1	0	1	1

$$0.1011 = 2^{-1} + 2^{-3} + 2^{-4}$$
$$= 0.5 + 0.125 + 0.0625 = \mathbf{0.6875}$$

Related Problem Convert the binary number 10.111 to decimal.

SECTION 2–2 REVIEW

1. What is the largest decimal number that can be represented in binary with eight bits?
2. Determine the weight of the 1 in the binary number 10000.
3. Convert the binary number 10111101.011 to decimal.

2–3 DECIMAL-TO-BINARY CONVERSION

In Section 2–2 you learned how to convert a binary number to the equivalent decimal number. Now you will learn two ways of converting from a decimal number to a binary number.

After completing this section, you should be able to

■ Convert a decimal number to binary using the sum-of-weights method ■ Convert a decimal whole number to binary using the repeated division-by-2 method ■ Convert a decimal fraction to binary using the repeated multiplication-by-2 method

Sum-of-Weights Method

One way to find the binary number that is equivalent to a given decimal number is to determine the set of binary weights whose sum is equal to the decimal number. An easy way to remember binary weights is that the lowest is 1, which is 2^0, and that by doubling any weight, you get the next higher weight; thus, a list of seven binary weights would be 64, 32, 16, 8, 4, 2, 1 as you learned in the last section. The decimal number 9, for example, can be expressed as the sum of binary weights as follows:

To get the binary number for a given decimal number, find the binary weights that add up to the decimal number.

$$9 = 8 + 1 \quad \text{or} \quad 9 = 2^3 + 2^0$$

Placing 1s in the appropriate weight positions, 2^3 and 2^0, and 0s in the 2^2 and 2^1 positions determines the binary number for decimal 9.

2^3	2^2	2^1	2^0	
1	0	0	1	Binary number for decimal 9

EXAMPLE 2–5

Convert the following decimal numbers to binary:

(a) 12 **(b)** 25 **(c)** 58 **(d)** 82

Solution

(a) $12 = 8 + 4 = 2^3 + 2^2 \longrightarrow$ **1100**

(b) $25 = 16 + 8 + 1 = 2^4 + 2^3 + 2^0 \longrightarrow$ **11001**

(c) $58 = 32 + 16 + 8 + 2 = 2^5 + 2^4 + 2^3 + 2^1 \longrightarrow$ **111010**

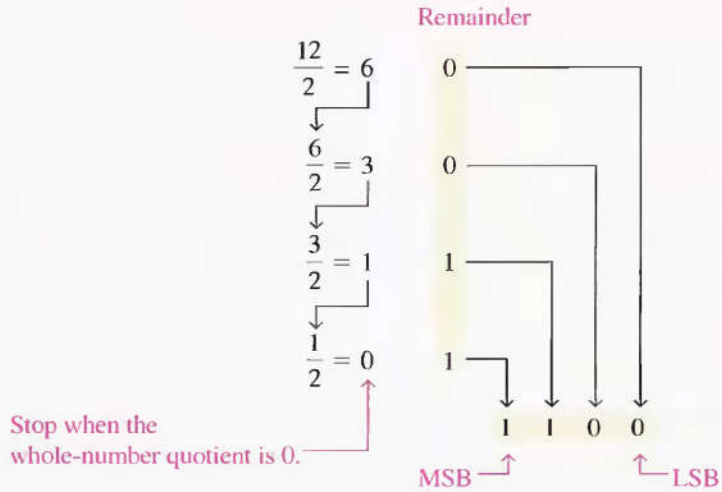
(d) $82 = 64 + 16 + 2 = 2^6 + 2^4 + 2^1 \longrightarrow$ **1010010**

Related Problem Convert the decimal number 125 to binary.

Repeated Division-by-2 Method

To get the binary number for a given decimal number, divide the decimal number by 2 until the quotient is 0. Remainders form the binary number.

A systematic method of converting whole numbers from decimal to binary is the *repeated division-by-2* process. For example, to convert the decimal number 12 to binary, begin by dividing 12 by 2. Then divide each resulting quotient by 2 until there is a 0 whole-number quotient. The **remainders** generated by each division form the binary number. The first remainder to be produced is the LSB (least significant bit) in the binary number, and the last remainder to be produced is the MSB (most significant bit). This procedure is shown in the following steps for converting the decimal number 12 to binary.



EXAMPLE 2–6

Convert the following decimal numbers to binary:

(a) 19 **(b)** 45

Solution

(a)

	Remainder
$\frac{19}{2} = 9$	1
$\frac{9}{2} = 4$	1
$\frac{4}{2} = 2$	0
$\frac{2}{2} = 1$	0
$\frac{1}{2} = 0$	1

1 0 0 1 1

MSB ↑ ↑ LSB

(b)

	Remainder
$\frac{45}{2} = 22$	1
$\frac{22}{2} = 11$	0
$\frac{11}{2} = 5$	1
$\frac{5}{2} = 2$	1
$\frac{2}{2} = 1$	0
$\frac{1}{2} = 0$	1

1 0 1 1 0 1

MSB ↑ ↑ LSB

Related Problem Convert decimal number 39 to binary.

CALCULATOR TUTORIAL

Conversion of a Decimal Number to a Binary Number

Example Convert decimal 57 to binary.

TI-86

Step 1. 2nd 1 F3 (BASE)

Step 2. 5 7

Step 3. F1

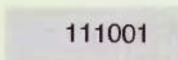
Step 4. ENTER

TI-36X

Step 1. 2nd EE (DEC)

Step 2. 5 7

Step 3. 2nd X (BIN)



Converting Decimal Fractions to Binary

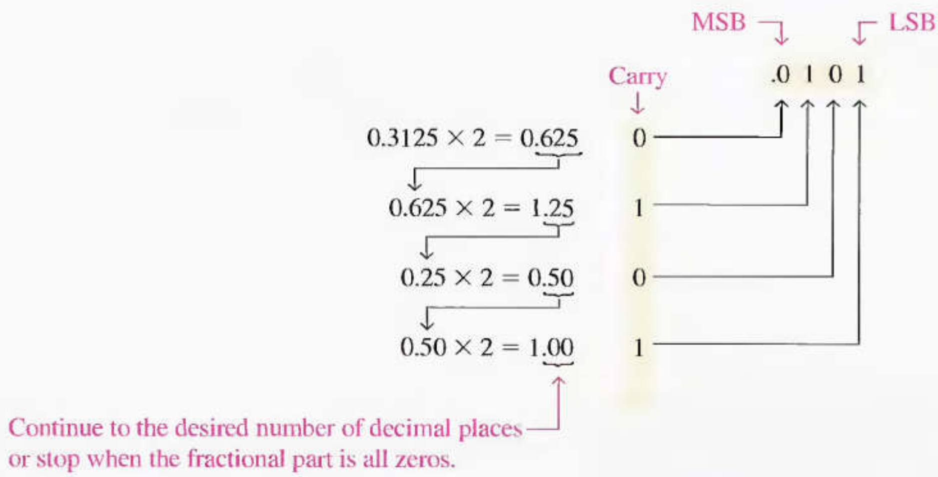
Examples 2–5 and 2–6 demonstrated whole-number conversions. Now let's look at fractional conversions. An easy way to remember fractional binary weights is that the most significant weight is 0.5, which is 2^{-1}, and that by halving any weight, you get the next lower weight; thus a list of four fractional binary weights would be 0.5, 0.25, 0.125, 0.0625.

Sum-of-Weights The sum-of-weights method can be applied to fractional decimal numbers, as shown in the following example:

$$0.625 = 0.5 + 0.125 = 2^{-1} + 2^{-3} = 0.101$$

There is a 1 in the 2^{-1} position, a 0 in the 2^{-2} position, and a 1 in the 2^{-3} position.

Repeated Multiplication by 2 As you have seen, decimal whole numbers can be converted to binary by repeated division by 2. Decimal fractions can be converted to binary by repeated multiplication by 2. For example, to convert the decimal fraction 0.3125 to binary, begin by multiplying 0.3125 by 2 and then multiplying each resulting fractional part of the product by 2 until the fractional product is zero or until the desired number of decimal places is reached. The carry digits, or **carries**, generated by the multiplications produce the binary number. The first carry produced is the MSB, and the last carry is the LSB. This procedure is illustrated as follows:



SECTION 2–3 REVIEW

1. Convert each decimal number to binary by using the sum-of-weights method:
   (a) 23 **(b)** 57 **(c)** 45.5
2. Convert each decimal number to binary by using the repeated division-by-2 method (repeated multiplication-by-2 for fractions):
   (a) 14 **(b)** 21 **(c)** 0.375

2–4 BINARY ARITHMETIC

Binary arithmetic is essential in all digital computers and in many other types of digital systems. To understand digital systems, you must know the basics of binary addition, subtraction, multiplication, and division. This section provides an introduction that will be expanded in later sections.

After completing this section, you should be able to

- Add binary numbers
- Subtract binary numbers
- Multiply binary numbers
- Divide binary numbers

Binary Addition

Remember, in binary 1 + 1 = 10, not 2.

The four basic rules for adding binary digits (bits) are as follows:

$0 + 0 = 0$	Sum of 0 with a carry of 0
$0 + 1 = 1$	Sum of 1 with a carry of 0
$1 + 0 = 1$	Sum of 1 with a carry of 0
$1 + 1 = 10$	Sum of 0 with a carry of 1

Notice that the first three rules result in a single bit and in the fourth rule the addition of two 1s yields a binary two (10). When binary numbers are added, the last condition creates a sum of 0 in a given column and a carry of 1 over to the next column to the left, as illustrated in the following addition of $11 + 1$:

$$\begin{array}{ccc} \text{Carry} & \text{Carry} & \\ 1 & 1 & \\ 0 & 1 & 1 \\ +0 & 0 & 1 \\ \hline 1 & 0 & 0 \end{array}$$

In the right column, $1 + 1 = 0$ with a carry of 1 to the next column to the left. In the middle column, $1 + 1 + 0 = 0$ with a carry of 1 to the next column to the left. In the left column, $1 + 0 + 0 = 1$.

When there is a carry of 1, you have a situation in which three bits are being added (a bit in each of the two numbers and a carry bit). This situation is illustrated as follows:

Carry bits

$1 + 0 + 0 = 01$	Sum of 1 with a carry of 0
$1 + 1 + 0 = 10$	Sum of 0 with a carry of 1
$1 + 0 + 1 = 10$	Sum of 0 with a carry of 1
$1 + 1 + 1 = 11$	Sum of 1 with a carry of 1

EXAMPLE 2–7

Add the following binary numbers:

(a) 11 + 11 **(b)** 100 + 10 **(c)** 111 + 11 **(d)** 110 + 100

Solution The equivalent decimal addition is also shown for reference.

$$\text{(a)}\ \begin{array}{r} 11 \\ +11 \\ \hline \mathbf{110} \end{array} \quad \begin{array}{r} 3 \\ +3 \\ \hline 6 \end{array} \qquad \text{(b)}\ \begin{array}{r} 100 \\ +10 \\ \hline \mathbf{110} \end{array} \quad \begin{array}{r} 4 \\ +2 \\ \hline 6 \end{array} \qquad \text{(c)}\ \begin{array}{r} 111 \\ +11 \\ \hline \mathbf{1010} \end{array} \quad \begin{array}{r} 7 \\ +3 \\ \hline 10 \end{array} \qquad \text{(d)}\ \begin{array}{r} 110 \\ +100 \\ \hline \mathbf{1010} \end{array} \quad \begin{array}{r} 6 \\ +4 \\ \hline 10 \end{array}$$

Related Problem Add 1111 and 1100.

Binary Subtraction

Remember in binary 10 − 1 = 1, not 9.

The four basic rules for subtracting bits are as follows:

$0 - 0 = 0$	
$1 - 1 = 0$	
$1 - 0 = 1$	
$10 - 1 = 1$	$0 - 1$ with a borrow of 1

When subtracting numbers, you sometimes have to borrow from the next column to the left. A borrow is required in binary only when you try to subtract a 1 from a 0. In this case, when a 1 is borrowed from the next column to the left, a 10 is created in the column being subtracted, and the last of the four basic rules just listed must be applied. Examples 2–8 and 2–9 illustrate binary subtraction; the equivalent decimal subtractions are also shown.

EXAMPLE 2–8

Perform the following binary subtractions:

(a) 11 − 01 (b) 11 − 10

Solution

$$\text{(a)}\quad \begin{array}{r} 11 \\ -\,01 \\ \hline \mathbf{10} \end{array} \qquad \begin{array}{r} 3 \\ -\,1 \\ \hline 2 \end{array} \qquad\qquad \text{(b)}\quad \begin{array}{r} 11 \\ -\,10 \\ \hline \mathbf{01} \end{array} \qquad \begin{array}{r} 3 \\ -\,2 \\ \hline 1 \end{array}$$

No borrows were required in this example. The binary number 01 is the same as 1.

Related Problem Subtract 100 from 111.

EXAMPLE 2–9

Subtract 011 from 101.

Solution

$$\begin{array}{r} 101 \\ -011 \\ \hline \mathbf{010} \end{array} \qquad \begin{array}{r} 5 \\ -3 \\ \hline 2 \end{array}$$

Let's examine exactly what was done to subtract the two binary numbers since a borrow is required. Begin with the right column.

Left column:
When a 1 is borrowed,
a 0 is left, so 0 − 0 = 0.

Middle column:
Borrow 1 from next column
to the left, making a 10 in
this column, then 10 − 1 = 1.

$$\begin{array}{r} \overset{0}{\cancel{1}}\,{}^{1}01 \\ -0\;11 \\ \hline 0\;10 \end{array}$$

Right column:
1 − 1 = 0

Related Problem Subtract 101 from 110.

Binary Multiplication

Binary multiplication of two bits is the same as multiplication of the decimal digits 0 and 1.

The four basic rules for multiplying bits are as follows:

$$\begin{aligned} 0 \times 0 &= 0 \\ 0 \times 1 &= 0 \\ 1 \times 0 &= 0 \\ 1 \times 1 &= 1 \end{aligned}$$

Multiplication is performed with binary numbers in the same manner as with decimal numbers. It involves forming partial products, shifting each successive partial product left one place, and then adding all the partial products. Example 2–10 illustrates the procedure; the equivalent decimal multiplications are shown for reference.

EXAMPLE 2–10

Perform the following binary multiplications:

(a) 11×11 **(b)** 101×111

Solution **(a)**

$$\begin{array}{lr} & 11 \\ & \times\, 11 \\ \hline \text{Partial} & 11 \\ \text{products} & +11 \\ \hline & \mathbf{1001} \end{array} \qquad \begin{array}{r} 3 \\ \times\, 3 \\ \hline 9 \end{array}$$

(b)

$$\begin{array}{lr} & 111 \\ & \times\, 101 \\ \hline \text{Partial} & 111 \\ \text{products} & 000 \\ & +111 \\ \hline & \mathbf{100011} \end{array} \qquad \begin{array}{r} 7 \\ \times\, 5 \\ \hline 35 \end{array}$$

Related Problem Multiply 1101×1010.

Binary Division

Division in binary follows the same procedure as division in decimal, as Example 2–11 illustrates. The equivalent decimal divisions are also given.

A calculator can be used to perform arithmetic operations with binary numbers as long as the capacity of the calculator is not exceeded.

EXAMPLE 2–11

Perform the following binary divisions:

(a) $110 \div 11$ **(b)** $110 \div 10$

Solution

(a)

$$\begin{array}{r} \mathbf{10} \\ 11\overline{)110} \\ \underline{11} \\ 000 \end{array} \qquad \begin{array}{r} 2 \\ 3\overline{)6} \\ \underline{6} \\ 0 \end{array}$$

(b)

$$\begin{array}{r} \mathbf{11} \\ 10\overline{)110} \\ \underline{10} \\ 10 \\ \underline{10} \\ 00 \end{array} \qquad \begin{array}{r} 3 \\ 2\overline{)6} \\ \underline{6} \\ 0 \end{array}$$

Related Problem Divide 1100 by 100.

SECTION 2–4 REVIEW

1. Perform the following binary additions:
   (a) 1101 + 1010 (b) 10111 + 01101
2. Perform the following binary subtractions:
   (a) 1101 − 0100 (b) 1001 − 0111
3. Perform the indicated binary operations:
   (a) 110×111 (b) $1100 \div 011$

2–5 1'S AND 2'S COMPLEMENTS OF BINARY NUMBERS

The 1's complement and the 2's complement of a binary number are important because they permit the representation of negative numbers. The method of 2's complement arithmetic is commonly used in computers to handle negative numbers.

After completing this section, you should be able to

- Convert a binary number to its 1's complement
- Convert a binary number to its 2's complement using either of two methods

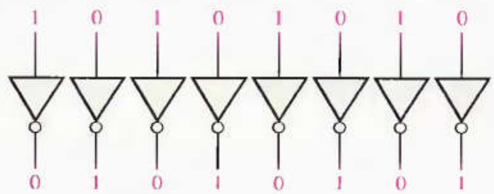
Finding the 1's Complement

Change each bit in a number to get the 1's complement.

The 1's **complement** of a binary number is found by changing all 1s to 0s and all 0s to 1s, as illustrated below:

1 0 1 1 0 0 1 0	Binary number
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
0 1 0 0 1 1 0 1	1's complement

The simplest way to obtain the 1's complement of a binary number with a digital circuit is to use parallel inverters (NOT circuits), as shown in Figure 2–2 for an 8-bit binary number.



► FIGURE 2–2

Example of inverters used to obtain the 1's complement of a binary number.

Finding the 2's Complement

Add 1 to the 1's complement to get the 2's complement.

The 2's complement of a binary number is found by adding 1 to the LSB of the 1's complement.

2's complement = (1's complement) + 1

EXAMPLE 2–12

Find the 2's complement of 10110010.

Solution

10110010	Binary number
01001101	1's complement
+ 1	Add 1
01001110	2's complement

Related Problem Determine the 2's complement of 11001011.

An alternative method of finding the 2's complement of a binary number is as follows:

Change all bits to the left of the least significant 1 to get 2's complement.

1. Start at the right with the LSB and write the bits as they are up to and including the first 1.
2. Take the 1's complements of the remaining bits.

EXAMPLE 2–13

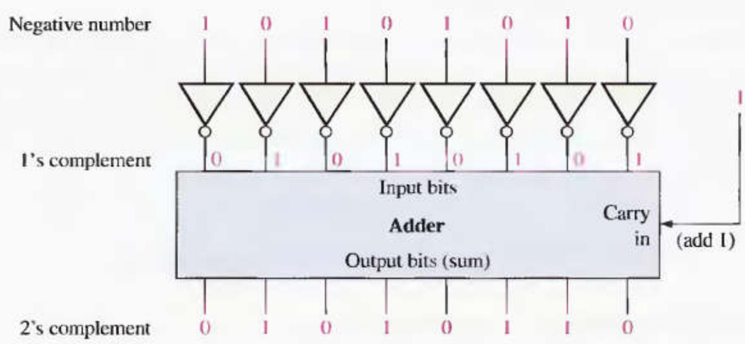
Find the 2's complement of 10111000 using the alternative method.

Solution

	10111000	Binary number
1's complements of original bits →	**01001000**	2's complement
	↑ These bits stay the same.	

Related Problem Find the 2's complement of 11000000.

The 2's complement of a negative binary number can be realized using inverters and an adder, as indicated in Figure 2–3. This illustrates how an 8-bit number can be converted to its 2's complement by first inverting each bit (taking the 1's complement) and then adding 1 to the 1's complement with an adder circuit.



◄ **FIGURE 2–3**

Example of obtaining the 2's complement of a negative binary number.

To convert from a 1's or 2's complement back to the true (uncomplemented) binary form, use the same two procedures described previously. To go from the 1's complement back to true binary, reverse all the bits. To go from the 2's complement form back to true binary, take the 1's complement of the 2's complement number and add 1 to the least significant bit.

SECTION 2–5 REVIEW

1. Determine the 1's complement of each binary number:
   (a) 00011010 (b) 11110111 (c) 10001101
2. Determine the 2's complement of each binary number:
   (a) 00010110 (b) 11111100 (c) 10010001

2–6 SIGNED NUMBERS

Digital systems, such as the computer, must be able to handle both positive and negative numbers. A signed binary number consists of both sign and magnitude information. The sign indicates whether a number is positive or negative, and the magnitude is the value of the number. There are three forms in which signed integer (whole) numbers can be represented in binary: sign-magnitude, 1's complement, and 2's complement. Of these, the 2's complement is the most important and the sign-magnitude is the least used. Noninteger and very large or small numbers can be expressed in floating-point format.

After completing this section, you should be able to

■ Express positive and negative numbers in sign-magnitude ■ Express positive and negative numbers in 1's complement ■ Express positive and negative numbers in 2's complement ■ Determine the decimal value of signed binary numbers ■ Express a binary number in floating-point format

The Sign Bit

The left-most bit in a signed binary number is the **sign bit,** which tells you whether the number is positive or negative.

A 0 sign bit indicates a positive number, and a 1 sign bit indicates a negative number.

Sign-Magnitude Form

When a signed binary number is represented in sign-magnitude, the left-most bit is the sign bit and the remaining bits are the magnitude bits. The magnitude bits are in true (uncomplemented) binary for both positive and negative numbers. For example, the decimal number +25 is expressed as an 8-bit signed binary number using the sign-magnitude form as

$$\underset{\text{Sign bit}}{0}\underbrace{0011001}_{\text{Magnitude bits}}$$

The decimal number −25 is expressed as

10011001

Notice that the only difference between +25 and −25 is the sign bit because the magnitude bits are in true binary for both positive and negative numbers.

In the sign-magnitude form, a negative number has the same magnitude bits as the corresponding positive number but the sign bit is a 1 rather than a zero.

COMPUTER NOTE

Computers use the 2's complement for negative integer numbers in all arithmetic operations. The reason is that subtraction of a number is the same as adding the 2's complement of the number. Computers form the 2's complement by inverting the bits and adding 1, using special instructions that produce the same result as the adder in Figure 2–3.

1's Complement Form

Positive numbers in 1's complement form are represented the same way as the positive sign-magnitude numbers. Negative numbers, however, are the 1's complements of the corresponding positive numbers. For example, using eight bits, the decimal number −25 is expressed as the 1's complement of +25 (00011001) as

11100110

In the 1's complement form, a negative number is the 1's complement of the corresponding positive number.

2's Complement Form

Positive numbers in 2's complement form are represented the same way as in the sign-magnitude and 1's complement forms. Negative numbers are the 2's complements of the corresponding positive numbers. Again, using eight bits, let's take decimal number −25 and express it as the 2's complement of +25 (00011001).

11100111

In the 2's complement form, a negative number is the 2's complement of the corresponding positive number.

EXAMPLE 2–14

Express the decimal number −39 as an 8-bit number in the sign-magnitude, 1's complement, and 2's complement forms.

Solution First, write the 8-bit number for +39.

00100111

In the *sign-magnitude form,* −39 is produced by changing the sign bit to a 1 and leaving the magnitude bits as they are. The number is

10100111

In the *1's complement form,* −39 is produced by taking the 1's complement of +39 (00100111).

11011000

In the *2's complement form,* −39 is produced by taking the 2's complement of +39 (00100111) as follows:

11011000	1's complement
+ 1	
11011001	2's complement

Related Problem Express +19 and −19 in sign-magnitude, 1's complement, and 2's complement.

The Decimal Value of Signed Numbers

Sign-magnitude Decimal values of positive and negative numbers in the sign-magnitude form are determined by summing the weights in all the magnitude bit positions where there are 1s and ignoring those positions where there are zeros. The sign is determined by examination of the sign bit.

EXAMPLE 2–15

Determine the decimal value of this signed binary number expressed in sign-magnitude: 10010101.

Solution The seven magnitude bits and their powers-of-two weights are as follows:

2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	0	1	0	1

Summing the weights where there are 1s,

$$16 + 4 + 1 = 21$$

The sign bit is 1; therefore, the decimal number is **−21.**

Related Problem Determine the decimal value of the sign-magnitude number 01110111.

1's Complement Decimal values of positive numbers in the 1's complement form are determined by summing the weights in all bit positions where there are 1s and ignoring those positions where there are zeros. Decimal values of negative numbers are determined by assigning a negative value to the weight of the sign bit, summing all the weights where there are 1s, and adding 1 to the result.

EXAMPLE 2–16

Determine the decimal values of the signed binary numbers expressed in 1's complement:

(a) 00010111 **(b)** 11101000

Solution **(a)** The bits and their powers-of-two weights for the positive number are as follows:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	1	0	1	1	1

Summing the weights where there are 1s,

$$16 + 4 + 2 + 1 = \mathbf{+23}$$

(b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of -2^7 or -128.

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	1	0	0	0

Summing the weights where there are 1s,

$$-128 + 64 + 32 + 8 = -24$$

Adding 1 to the result, the final decimal number is

$$-24 + 1 = \mathbf{-23}$$

Related Problem Determine the decimal value of the 1's complement number 11101011.

2's Complement Decimal values of positive and negative numbers in the 2's complement form are determined by summing the weights in all bit positions where there are 1s and ignoring those positions where there are zeros. The weight of the sign bit in a negative number is given a negative value.

EXAMPLE 2–17

Determine the decimal values of the signed binary numbers expressed in 2's complement:

(a) 01010110 **(b)** 10101010

Solution **(a)** The bits and their powers-of-two weights for the positive number are as follows:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	1	0	1	1	0

Summing the weights where there are 1s,

$$64 + 16 + 4 + 2 = \mathbf{+86}$$

(b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of $-2^7 = -128$.

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	0	1	0	1	0

Summing the weights where there are 1s,

$$-128 + 32 + 8 + 2 = \mathbf{-86}$$

Related Problem Determine the decimal value of the 2's complement number 11010111.

From these examples, you can see why the 2's complement form is preferred for representing signed integer numbers: To convert to decimal, it simply requires a summation of weights regardless of whether the number is positive or negative. The 1's complement system requires adding 1 to the summation of weights for negative numbers but not for positive numbers. Also, the 1's complement form is generally not used because two representations of zero (00000000 or 11111111) are possible.

Range of Signed Integer Numbers That Can Be Represented

The range of magnitude of a binary number depends on the number of bits (n).

We have used 8-bit numbers for illustration because the 8-bit grouping is common in most computers and has been given the special name **byte.** With one byte or eight bits, you can represent 256 different numbers. With two bytes or sixteen bits, you can represent 65,536 different numbers. With four bytes or 32 bits, you can represent 4.295×10^9 different numbers. The formula for finding the number of different combinations of n bits is

$$\text{Total combinations} = 2^n$$

For 2's complement signed numbers, the range of values for n-bit numbers is

$$\text{Range} = -(2^{n-1}) \text{ to } +(2^{n-1} - 1)$$

where in each case there is one sign bit and $n - 1$ magnitude bits. For example, with four bits you can represent numbers in 2's complement ranging from $-(2^3) = -8$ to $2^3 - 1 = +7$. Similarly, with eight bits you can go from -128 to $+127$, with sixteen bits you can go from $-32{,}768$ to $+32{,}767$, and so on.

Floating-Point Numbers

To represent very large **integer** (whole) numbers, many bits are required. There is also a problem when numbers with both integer and fractional parts, such as 23.5618, need to be represented. The floating-point number system, based on scientific notation, is capable of representing very large and very small numbers without an increase in the number of bits and also for representing numbers that have both integer and fractional components.

A **floating-point number** (also known as a *real number*) consists of two parts plus a sign. The **mantissa** is the part of a floating-point number that represents the magnitude of the number. The **exponent** is the part of a floating-point number that represents the number of places that the decimal point (or binary point) is to be moved.

A decimal example will be helpful in understanding the basic concept of floating-point numbers. Let's consider a decimal number which, in integer form, is 241,506,800. The mantissa is .2415068 and the exponent is 9. When the integer is expressed as a floating-point number, it is normalized by moving the decimal point to the left of all the digits so that the mantissa is a fractional number and the exponent is the power of ten. The floating-point number is written as

$$0.2415068 \times 10^9$$

For binary floating-point numbers, the format is defined by ANSI/IEEE Standard 754-1985 in three forms: *single-precision, double-precision,* and *extended-precision.* These all have the same basic formats except for the number of bits. Single-precision floating-point numbers have 32 bits, double-precision numbers have 64 bits, and extended-precision numbers have 80 bits. We will restrict our discussion to the single-precision floating-point format.

COMPUTER NOTE

In addition to the CPU (central processing unit), computers use *coprocessors* to perform complicated mathematical calculations using floating-point numbers. The purpose is to increase performance by freeing up the CPU for other tasks. The mathematical coprocessor is also known as the floating-point unit (FPU).

Single-Precision Floating-Point Binary Numbers In the standard format for a single-precision binary number, the sign bit (S) is the left-most bit, the exponent (E) includes the next eight bits, and the mantissa or fractional part (F) includes the remaining 23 bits, as shown next.

← 32 bits →		
S	Exponent (E)	Mantissa (fraction, F)
1 bit	8 bits	23 bits

In the mantissa or fractional part, the binary point is understood to be to the left of the 23 bits. Effectively, there are 24 bits in the mantissa because in any binary number the left-most (most significant) bit is always a 1. Therefore, this 1 is understood to be there although it does not occupy an actual bit position.

The eight bits in the exponent represent a *biased exponent,* which is obtained by adding 127 to the actual exponent. The purpose of the bias is to allow very large or very small numbers without requiring a separate sign bit for the exponents. The biased exponent allows a range of actual exponent values from -126 to $+128$.

To illustrate how a binary number is expressed in floating-point format, let's use 1011010010001 as an example. First, it can be expressed as 1 plus a fractional binary number by moving the binary point 12 places to the left and then multiplying by the appropriate power of two.

$$1011010010001 = 1.011010010001 \times 2^{12}$$

Assuming that this is a positive number, the sign bit (S) is 0. The exponent, 12, is expressed as a biased exponent by adding it to 127 ($12 + 127 = 139$). The biased exponent (E) is expressed as the binary number 10001011. The mantissa is the fractional part (F) of the binary number, .011010010001. Because there is always a 1 to the left of the binary point in the power-of-two expression, it is not included in the mantissa. The complete floating-point number is

S	E	F
0	10001011	01101001000100000000000

Next, let's see how to evaluate a binary number that is already in floating-point format. The general approach to determining the value of a floating-point number is expressed by the following formula:

$$\text{Number} = (-1)^S(1 + F)(2^{E-127})$$

To illustrate, let's consider the following floating-point binary number:

S	E	F
1	10010001	10001110001000000000000

The sign bit is 1. The biased exponent is 10010001 = 145. Applying the formula, we get

$$\begin{aligned}\text{Number} &= (-1)^1(1.10001110001)(2^{145-127})\\ &= (-1)(1.10001110001)(2^{18}) = -1100011100010000000\end{aligned}$$

This floating-point binary number is equivalent to −407,688 in decimal. Since the exponent can be any number between −126 and +128, extremely large and small numbers can be expressed. A 32-bit floating-point number can replace a binary integer number having 129 bits. Because the exponent determines the position of the binary point, numbers containing both integer and fractional parts can be represented.

There are two exceptions to the format for floating-point numbers: The number 0.0 is represented by all 0s, and infinity is represented by all 1s in the exponent and all 0s in the mantissa.

EXAMPLE 2–18

Convert the decimal number 3.248×10^4 to a single-precision floating-point binary number.

Solution Convert the decimal number to binary.

$$3.248 \times 10^4 = 32480 = 111111011100000_2 = 1.11111011100000 \times 2^{14}$$

The MSB will not occupy a bit position because it is always a 1. Therefore, the mantissa is the fractional 23-bit binary number 11111011100000000000000 and the biased exponent is

$$14 + 127 = 141 = 10001101_2$$

The complete floating-point number is

0	**10001101**	**11111011100000000000000**

Related Problem Determine the binary value of the following floating-point binary number:

0 10011000 10000100010100110000000

SECTION 2–6 REVIEW

1. Express the decimal number +9 as an 8-bit binary number in the sign-magnitude system.
2. Express the decimal number −33 as an 8-bit binary number in the 1's complement system.
3. Express the decimal number −46 as an 8-bit binary number in the 2's complement system.
4. List the three parts of a signed, floating-point number.

2–7 ARITHMETIC OPERATIONS WITH SIGNED NUMBERS

In the last section, you learned how signed numbers are represented in three different forms. In this section, you will learn how signed numbers are added, subtracted, multiplied, and divided. Because the 2's complement form for representing signed numbers is the most widely used in computers and microprocessor-based systems, the coverage in this section is limited to 2's complement arithmetic. The processes covered can be extended to the other forms if necessary.

After completing this section, you should be able to

■ Add signed binary numbers ■ Explain how computers add strings of numbers ■ Define *overflow* ■ Subtract signed binary numbers ■ Multiply signed binary numbers using the direct addition method ■ Multiply signed binary numbers using the partial products method ■ Divide signed binary numbers

Addition

The two numbers in an addition are the **addend** and the **augend.** The result is the **sum.** There are four cases that can occur when two signed binary numbers are added.

1. Both numbers positive
2. Positive number with magnitude larger than negative number
3. Negative number with magnitude larger than positive number
4. Both numbers negative

Let's take one case at a time using 8-bit signed numbers as examples. The equivalent decimal numbers are shown for reference.

Addition of two positive numbers yields a positive number.

Both numbers positive:

$$\begin{array}{r} 00000111 \\ +\,00000100 \\ \hline 00001011 \end{array} \qquad \begin{array}{r} 7 \\ +\;4 \\ \hline 11 \end{array}$$

The sum is positive and is therefore in true (uncomplemented) binary.

Addition of a positive number and a smaller negative number yields a positive number.

Positive number with magnitude larger than negative number:

$$\begin{array}{rr} & 00001111 \\ & +\,11111010 \\ \hline \text{Discard carry} \longrightarrow 1 & 00001001 \end{array} \qquad \begin{array}{r} 15 \\ +\,-6 \\ \hline 9 \end{array}$$

The final carry bit is discarded. The sum is positive and therefore in true (uncomplemented) binary.

Addition of a positive number and a larger negative number or two negative numbers yields a negative number in 2's complement.

Negative number with magnitude larger than positive number:

$$\begin{array}{r} 00010000 \\ +\,11101000 \\ \hline 11111000 \end{array} \qquad \begin{array}{r} 16 \\ +\;-24 \\ \hline -8 \end{array}$$

The sum is negative and therefore in 2's complement form.

Both numbers negative:

$$\begin{array}{rr} & 11111011 \\ & +\,11110111 \\ \hline \text{Discard carry} \longrightarrow 1 & 11110010 \end{array} \qquad \begin{array}{r} -5 \\ +\,-9 \\ \hline -14 \end{array}$$

The final carry bit is discarded. The sum is negative and therefore in 2's complement form.

In a computer, the negative numbers are stored in 2's complement form so, as you can see, the addition process is very simple: *Add the two numbers and discard any final carry bit.*

Overflow Condition When two numbers are added and the number of bits required to represent the sum exceeds the number of bits in the two numbers, an **overflow** results as indicated by an incorrect sign bit. An overflow can occur only when both numbers are positive or both numbers are negative. The following 8-bit example will illustrate this condition.

01111101	125
+ 00111010	+ 58
10110111	183

Sign incorrect
Magnitude incorrect

In this example the sum of 183 requires eight magnitude bits. Since there are seven magnitude bits in the numbers (one bit is the sign), there is a carry into the sign bit which produces the overflow indication.

Numbers Are Added Two at a Time Now let's look at the addition of a string of numbers, added two at a time. This can be accomplished by adding the first two numbers, then adding the third number to the sum of the first two, then adding the fourth number to this result, and so on. This is how computers add strings of numbers. The addition of numbers taken two at a time is illustrated in Example 2–19.

EXAMPLE 2–19

Add the signed numbers: 01000100, 00011011, 00001110, and 00010010.

Solution The equivalent decimal additions are given for reference.

68	01000100	
+ 27	+ 00011011	Add 1st two numbers
95	01011111	1st sum
+ 14	+ 00001110	Add 3rd number
109	01101101	2nd sum
+ 18	+ 00010010	Add 4th number
127	**01111111**	Final sum

Related Problem Add 00110011, 10111111, and 01100011. These are signed numbers.

Subtraction

Subtraction is addition with the sign of the subtrahend changed.

Subtraction is a special case of addition. For example, subtracting +6 (the **subtrahend**) from +9 (the **minuend**) is equivalent to adding −6 to +9. Basically, *the subtraction operation changes the sign of the subtrahend and adds it to the minuend.* The result of a subtraction is called the **difference.**

The sign of a positive or negative binary number is changed by taking its 2's complement.

For example, when you take the 2's complement of the positive number 00000100 (+4), you get 11111100, which is −4 as the following sum-of-weights evaluation shows:

$$-128 + 64 + 32 + 16 + 8 + 4 = -4$$

As another example, when you take the 2's complement of the negative number 11101101 (−19), you get 00010011, which is +19 as the following sum-of-weights evaluation shows:

$$16 + 2 + 1 = 19$$

Since subtraction is simply an addition with the sign of the subtrahend changed, the process is stated as follows:

To subtract two signed numbers, take the 2's complement of the subtrahend and add. Discard any final carry bit.

Example 2–20 illustrates the subtraction process.

EXAMPLE 2–20

Perform each of the following subtractions of the signed numbers:

(a) 00001000 − 00000011 (b) 00001100 − 11110111

(c) 11100111 − 00010011 (d) 10001000 − 11100010

Solution Like in other examples, the equivalent decimal subtractions are given for reference.

(a) In this case, $8 - 3 = 8 + (-3) = 5$.

	00001000	Minuend (+8)
	+ 11111101	2's complement of subtrahend (−3)
Discard carry →	**1 00000101**	Difference (+5)

(b) In this case, $12 - (-9) = 12 + 9 = 21$.

00001100	Minuend (+12)
+ 00001001	2's complement of subtrahend (+9)
00010101	Difference (+21)

(c) In this case, $-25 - (+19) = -25 + (-19) = -44$.

	11100111	Minuend (−25)
	+ 11101101	2's complement of subtrahend (−19)
Discard carry →	**1 11010100**	Difference (−44)

(d) In this case, $-120 - (-30) = -120 + 30 = -90$.

10001000	Minuend (−120)
+ 00011110	2's complement of subtrahend (+30)
10100110	Difference (−90)

Related Problem Subtract 01000111 from 01011000.

Multiplication

The numbers in a multiplication are the **multiplicand,** the **multiplier,** and the **product.** These are illustrated in the following decimal multiplication:

8	Multiplicand
× 3	Multiplier
24	Product

Multiplication is equivalent to adding a number to itself a number of times equal to the multiplier.

The multiplication operation in most computers is accomplished using addition. As you have already seen, subtraction is done with an adder; now let's see how multiplication is done.

Direct addition and *partial products* are two basic methods for performing multiplication using addition. In the direct addition method, you add the multiplicand a number of times equal to the multiplier. In the previous decimal example (3×8), three multiplicands are added: $8 + 8 + 8 = 24$. The disadvantage of this approach is that it becomes very lengthy if the multiplier is a large number. For example, to multiply 75×350, you must add 350 to itself 75 times. Incidentally, this is why the term *times* is used to mean multiply.

When two binary numbers are multiplied, both numbers must be in true (uncomplemented) form. The direct addition method is illustrated in Example 2–21 adding two binary numbers at a time.

EXAMPLE 2–21

Multiply the signed binary numbers: 01001101 (multiplicand) and 00000100 (multiplier) using the direct addition method.

Solution Since both numbers are positive, they are in true form, and the product will be positive. The decimal value of the multiplier is 4, so the multiplicand is added to itself four times as follows:

01001101	1st time
+ 01001101	2nd time
10011010	Partial sum
+ 01001101	3rd time
11100111	Partial sum
+ 01001101	4th time
100110100	Product

Since the sign bit of the multiplicand is 0, it has no effect on the outcome. All of the bits in the product are magnitude bits.

Related Problem Multiply 01100001 by 00000110 using the direct addition method.

The partial products method is perhaps the more common one because it reflects the way you multiply longhand. The multiplicand is multiplied by each multiplier digit beginning with the least significant digit. The result of the multiplication of the multiplicand by a multiplier digit is called a *partial product.* Each successive partial product is moved (shifted) one place to the left and when all the partial products have been produced, they are added to get the final product. Here is a decimal example.

239	Multiplicand
$\times$ 123	Multiplier
717	1st partial product (3×239)
478	2nd partial product (2×239)
+ 239	3rd partial product (1×239)
29,397	Final product

The sign of the product of a multiplication depends on the signs of the multiplicand and the multiplier according to the following two rules:

- **If the signs are the same, the product is positive.**
- **If the signs are different, the product is negative.**

The basic steps in the partial products method of binary multiplication are as follows:

Step 1. Determine if the signs of the multiplicand and multiplier are the same or different. This determines what the sign of the product will be.

Step 2. Change any negative number to true (uncomplemented) form. Because most computers store negative numbers in 2's complement, a 2's complement operation is required to get the negative number into true form.

Step 3. Starting with the least significant multiplier bit, generate the partial products. When the multiplier bit is 1, the partial product is the same as the multiplicand. When the multiplier bit is 0, the partial product is zero. Shift each successive partial product one bit to the left.

Step 4. Add each successive partial product to the sum of the previous partial products to get the final product.

Step 5. If the sign bit that was determined in step 1 is negative, take the 2's complement of the product. If positive, leave the product in true form. Attach the sign bit to the product.

EXAMPLE 2–22

Multiply the signed binary numbers: 01010011 (multiplicand) and 11000101 (multiplier).

Solution

Step 1: The sign bit of the multiplicand is 0 and the sign bit of the multiplier is 1. The sign bit of the product will be 1 (negative).

Step 2: Take the 2's complement of the multiplier to put it in true form.

$$11000101 \longrightarrow 00111011$$

Steps 3 and 4: The multiplication proceeds as follows. Notice that only the magnitude bits are used in these steps.

1010011	Multiplicand
× 0111011	Multiplier
1010011	1st partial product
+ 1010011	2nd partial product
11111001	Sum of 1st and 2nd
+ 0000000	3rd partial product
011111001	Sum
+ 1010011	4th partial product
1110010001	Sum
+ 1010011	5th partial product
100011000001	Sum
+ 1010011	6th partial product
1001100100001	Sum
+ 0000000	7th partial product
1001100100001	Final product

Step 5: Since the sign of the product is a 1 as determined in step 1, take the 2's complement of the product.

$$1001100100001 \longrightarrow 0110011011111$$

Attach the sign bit

$\longrightarrow$ **1 0110011011111**

Related Problem Verify the multiplication is correct by converting to decimal numbers and performing the multiplication.

Division

The numbers in a division are the **dividend,** the **divisor,** and the **quotient.** These are illustrated in the following standard division format.

$$\frac{\text{dividend}}{\text{divisor}} = \text{quotient}$$

The division operation in computers is accomplished using subtraction. Since subtraction is done with an adder, division can also be accomplished with an adder.

The result of a division is called the *quotient;* the quotient is the number of times that the divisor will go into the dividend. This means that the divisor can be subtracted from the dividend a number of times equal to the quotient, as illustrated by dividing 21 by 7.

21	Dividend
− 7	1st subtraction of divisor
14	1st partial remainder
− 7	2nd subtraction of divisor
7	2nd partial remainder
− 7	3rd subtraction of divisor
0	Zero remainder

In this simple example, the divisor was subtracted from the dividend three times before a remainder of zero was obtained. Therefore, the quotient is 3.

The sign of the quotient depends on the signs of the dividend and the divisor according to the following two rules:

- **If the signs are the same, the quotient is positive.**
- **If the signs are different, the quotient is negative.**

When two binary numbers are divided, both numbers must be in true (uncomplemented) form. The basic steps in a division process are as follows:

Step 1. Determine if the signs of the dividend and divisor are the same or different. This determines what the sign of the quotient will be. Initialize the quotient to zero.

Step 2. Subtract the divisor from the dividend using 2's complement addition to get the first partial remainder and add 1 to the quotient. If this partial remainder is positive, go to step 3. If the partial remainder is zero or negative, the division is complete.

Step 3. Subtract the divisor from the partial remainder and add 1 to the quotient. If the result is positive, repeat for the next partial remainder. If the result is zero or negative, the division is complete.

Continue to subtract the divisor from the dividend and the partial remainders until there is a zero or a negative result. Count the number of times that the divisor is subtracted and you have the quotient. Example 2–23 illustrates these steps using 8-bit signed binary numbers.

EXAMPLE 2–23

Divide 01100100 by 00011001.

Solution **Step 1:** The signs of both numbers are positive, so the quotient will be positive. The quotient is initially zero: 00000000.

Step 2: Subtract the divisor from the dividend using 2's complement addition (remember that final carries are discarded).

01100100	Dividend
+ 11100111	2's complement of divisor
01001011	Positive 1st partial remainder

Add 1 to quotient: 00000000 + 00000001 = 00000001.

Step 3: Subtract the divisor from the 1st partial remainder using 2's complement addition.

01001011	1st partial remainder
+ 11100111	2's complement of divisor
00110010	Positive 2nd partial remainder

Step 4: Subtract the divisor from the 2nd partial remainder using 2's complement addition.

00110010	2nd partial remainder
+ 11100111	2's complement of divisor
00011001	Positive 3rd partial remainder

Add 1 to quotient: 00000010 + 00000001 = 00000011.

Step 5: Subtract the divisor from the 3rd partial remainder using 2's complement addition.

00011001	3rd partial remainder
+ 11100111	2's complement of divisor
00000000	Zero remainder

Add 1 to quotient: 00000011 + 00000001 = **00000100** (final quotient). The process is complete.

Related Problem Verify that the process is correct by converting to decimal numbers and performing the division.

SECTION 2-7 REVIEW

1. List the four cases when numbers are added.
2. Add 00100001 and 10111100.
3. Subtract 00110010 from 01110111.
4. What is the sign of the product when two negative numbers are multiplied?
5. Multiply 01111111 by 00000101.
6. What is the sign of the quotient when a positive number is divided by a negative number?
7. Divide 00110000 by 00001100.

2–8 HEXADECIMAL NUMBERS

The hexadecimal number system has sixteen characters; it is used primarily as a compact way of displaying or writing binary numbers because it is very easy to convert between binary and hexadecimal. As you are probably aware, long binary numbers are difficult to read and write because it is easy to drop or transpose a bit. Since computers and microprocessors understand only 1s and 0s, it is necessary to use these digits when you program in "machine language." Imagine writing a sixteen bit instruction for a microprocessor system in 1s and 0s. It is much more efficient to use hexadecimal or octal; octal numbers are covered in Section 2–9. Hexadecimal is widely used in computer and microprocessor applications.

After completing this section, you should be able to

■ List the hexadecimal characters ■ Count in hexadecimal ■ Convert from binary to hexadecimal ■ Convert from hexadecimal to binary ■ Convert from hexadecimal to decimal ■ Convert from decimal to hexadecimal ■ Add hexadecimal numbers ■ Determine the 2's complement of a hexadecimal number ■ Subtract hexadecimal numbers

The **hexadecimal** number system has a base of sixteen; that is, it is composed of 16 **numeric** and alphabetic **characters.** Most digital systems process binary data in groups that are multiples of four bits, making the hexadecimal number very convenient because each hexadecimal digit represents a 4-bit binary number (as listed in Table 2–3).

The hexadecimal number system consists of digits 0–9 and letters A–F.

◄ TABLE 2–3

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

Ten numeric digits and six alphabetic characters make up the hexadecimal number system. The use of letters A, B, C, D, E, and F to represent numbers may seem strange at first, but keep in mind that any number system is only a set of sequential symbols. If you understand what quantities these symbols represent, then the form of the symbols themselves is less important once you get accustomed to using them. We will use the subscript 16 to designate hexadecimal numbers to avoid confusion with decimal numbers. Sometimes you may see an "h" following a hexadecimal number.

COMPUTER NOTE

With computer memories in the gigabyte (GB) range, specifying a memory address in binary is quite cumbersome. For example, it takes 32 bits to specify an address in a 4 GB memory. It is much easier to express a 32-bit code using 8 hexadecimal digits.

Counting in Hexadecimal

How do you count in hexadecimal once you get to F? Simply start over with another column and continue as follows:

10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 2A, 2B, 2C, 2D, 2E, 2F, 30, 31, . . .

With two hexadecimal digits, you can count up to FF_{16}, which is decimal 255. To count beyond this, three hexadecimal digits are needed. For instance, 100_{16} is decimal 256, 101_{16} is decimal 257, and so forth. The maximum 3-digit hexadecimal number is FFF_{16}, or decimal 4095. The maximum 4-digit hexadecimal number is $FFFF_{16}$, which is decimal 65,535.

Binary-to-Hexadecimal Conversion

Converting a binary number to hexadecimal is a straightforward procedure. Simply break the binary number into 4-bit groups, starting at the right-most bit and replace each 4-bit group with the equivalent hexadecimal symbol.

EXAMPLE 2–24

Convert the following binary numbers to hexadecimal:

(a) 1100101001010111 **(b)** 111111000101101001

Solution

(a) $\underbrace{1100}_{C}\,\underbrace{1010}_{A}\,\underbrace{0101}_{5}\,\underbrace{0111}_{7} = \mathbf{CA57_{16}}$

(b) $\underbrace{0011}_{3}\,\underbrace{1111}_{F}\,\underbrace{0001}_{1}\,\underbrace{0110}_{6}\,\underbrace{1001}_{9} = \mathbf{3F169_{16}}$

Two zeros have been added in part (b) to complete a 4-bit group at the left.

Related Problem Convert the binary number 1001111011110011100 to hexadecimal.

Hexadecimal-to-Binary Conversion

Hexadecimal is a convenient way to represent binary numbers.

To convert from a hexadecimal number to a binary number, reverse the process and replace each hexadecimal symbol with the appropriate four bits.

EXAMPLE 2–25

Determine the binary numbers for the following hexadecimal numbers:

(a) $10A4_{16}$ **(b)** $CF8E_{16}$ **(c)** 9742_{16}

Solution

(a) $\underbrace{1\ \ 0\ \ A\ \ 4}_{\mathbf{1000010100100}}$ **(b)** $\underbrace{C\ \ F\ \ 8\ \ E}_{\mathbf{1100111110001110}}$ **(c)** $\underbrace{9\ \ 7\ \ 4\ \ 2}_{\mathbf{1001011101000010}}$

In part (a), the MSB is understood to have three zeros preceding it, thus forming a 4-bit group.

Related Problem Convert the hexadecimal number 6BD3 to binary.

It should be clear that it is much easier to deal with a hexadecimal number than with the equivalent binary number. Since conversion is so easy, the hexadecimal system is widely used for representing binary numbers in programming, printouts, and displays.

Conversion between hexadecimal and binary is direct and easy.

Hexadecimal-to-Decimal Conversion

One way to find the decimal equivalent of a hexadecimal number is to first convert the hexadecimal number to binary and then convert from binary to decimal.

EXAMPLE 2–26

Convert the following hexadecimal numbers to decimal:

(a) $1C_{16}$ **(b)** $A85_{16}$

Solution Remember, convert the hexadecimal number to binary first, then to decimal.

(a)
$$\begin{array}{cc} 1 & C \\ \downarrow & \downarrow \\ 0001 & 1100 \end{array} = 2^4 + 2^3 + 2^2 = 16 + 8 + 4 = \mathbf{28_{10}}$$

(b)
$$\begin{array}{ccc} A & 8 & 5 \\ \downarrow & \downarrow & \downarrow \\ 1010 & 1000 & 0101 \end{array} = 2^{11} + 2^9 + 2^7 + 2^2 + 2^0 = 2048 + 512 + 128 + 4 + 1 = \mathbf{2693_{10}}$$

Related Problem Convert the hexadecimal number 6BD to decimal.

Another way to convert a hexadecimal number to its decimal equivalent is to multiply the decimal value of each hexadecimal digit by its weight and then take the sum of these products. The weights of a hexadecimal number are increasing powers of 16 (from right to left). For a 4-digit hexadecimal number, the weights are

16^3	16^2	16^1	16^0
4096	256	16	1

EXAMPLE 2–27

Convert the following hexadecimal numbers to decimal:

(a) $E5_{16}$ **(b)** $B2F8_{16}$

Solution Recall from Table 2–3 that letters A through F represent decimal numbers 10 through 15, respectively.

(a) $E5_{16} = (E \times 16) + (5 \times 1) = (14 \times 16) + (5 \times 1) = 224 + 5 = \mathbf{229_{10}}$

(b)
$$\begin{aligned} B2F8_{16} &= (B \times 4096) + (2 \times 256) + (F \times 16) + (8 \times 1) \\ &= (11 \times 4096) + (2 \times 256) + (15 \times 16) + (8 \times 1) \\ &= 45{,}056 + 512 + 240 + 8 = \mathbf{45{,}816_{10}} \end{aligned}$$

Related Problem Convert $60A_{16}$ to decimal.

CALCULATOR TUTORIAL

Powers of 16

Example Find the value of 16^4.

			Display
TI-86	**Step 1.**	1 6 ^	16 ^ 4
	Step 2.	4 ENTER	65536
TI-36X	**Step 1.**	1 6 y^x	
	Step 2.	4 =	65536

Decimal-to-Hexadecimal Conversion

Repeated division of a decimal number by 16 will produce the equivalent hexadecimal number, formed by the remainders of the divisions. The first remainder produced is the least significant digit (LSD). Each successive division by 16 yields a remainder that becomes a digit in the equivalent hexadecimal number. This procedure is similar to repeated division by 2 for decimal-to-binary conversion that was covered in Section 2–3. Example 2–28 illustrates the procedure. Note that when a quotient has a fractional part, the fractional part is multiplied by the divisor to get the remainder.

EXAMPLE 2–28

Convert the decimal number 650 to hexadecimal by repeated division by 16.

Solution

Hexadecimal remainder

$$\frac{650}{16} = 40.625 \rightarrow 0.625 \times 16 = 10 = \quad \text{A}$$

$$\frac{40}{16} = 2.5 \longrightarrow 0.5 \times 16 = 8 = \quad 8$$

$$\frac{2}{16} = 0.125 \longrightarrow 0.125 \times 16 = 2 = \quad 2$$

Stop when whole number quotient is zero.

2 8 A Hexadecimal number

MSD LSD

Related Problem Convert decimal 2591 to hexadecimal.

Hexadecimal Addition

Addition can be done directly with hexadecimal numbers by remembering that the hexadecimal digits 0 through 9 are equivalent to decimal digits 0 through 9 and that hexadecimal digits A through F are equivalent to decimal numbers 10 through 15. When adding two hexadecimal numbers, use the following rules. (Decimal numbers are indicated by a subscript 10.)

1. In any given column of an addition problem, think of the two hexadecimal digits in terms of their decimal values. For instance, $5_{16} = 5_{10}$ and $C_{16} = 12_{10}$.
2. If the sum of these two digits is 15_{10} or less, bring down the corresponding hexadecimal digit.
3. If the sum of these two digits is greater than 15_{10}, bring down the amount of the sum that exceeds 16_{10} and carry a 1 to the next column.

A calculator can be used to perform arithmetic operations with hexadecimal numbers.

CALCULATOR TUTORIAL

Conversion of a Decimal Number to a Hexadecimal Number

Example Convert decimal 650 to hexadecimal.

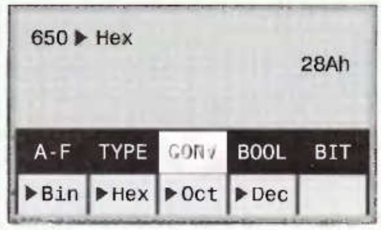
TI-86

Step 1. 2nd ↑ F3 (BASE)

Step 2. 6 5 0

Step 3. F2

Step 4. ENTER



TI-36X

Step 1. 3rd EE (DEC)

Step 2. 6 6 6

Step 3. 3rd ((HEX)

28A

EXAMPLE 2–29

Add the following hexadecimal numbers:

(a) $23_{16} + 16_{16}$ **(b)** $58_{16} + 22_{16}$ **(c)** $2B_{16} + 84_{16}$ **(d)** $DF_{16} + AC_{16}$

Solution

(a) $\begin{array}{r} 23_{16} \\ +16_{16} \\ \hline \mathbf{39_{16}} \end{array}$ right column: $3_{16} + 6_{16} = 3_{10} + 6_{10} = 9_{10} = 9_{16}$
left column: $2_{16} + 1_{16} = 2_{10} + 1_{10} = 3_{10} = 3_{16}$

(b) $\begin{array}{r} 58_{16} \\ +22_{16} \\ \hline \mathbf{7A_{16}} \end{array}$ right column: $8_{16} + 2_{16} = 8_{10} + 2_{10} = 10_{10} = A_{16}$
left column: $5_{16} + 2_{16} = 5_{10} + 2_{10} = 7_{10} = 7_{16}$

(c) $\begin{array}{r} 2B_{16} \\ +84_{16} \\ \hline \mathbf{AF_{16}} \end{array}$ right column: $B_{16} + 4_{16} = 11_{10} + 4_{10} = 15_{10} = F_{16}$
left column: $2_{16} + 8_{16} = 2_{10} + 8_{10} = 10_{10} = A_{16}$

(d) $\begin{array}{r} DF_{16} \\ +AC_{16} \\ \hline \mathbf{18B_{16}} \end{array}$ right column: $F_{16} + C_{16} = 15_{10} + 12_{10} = 27_{10}$
$27_{10} - 16_{10} = 11_{10} = B_{16}$ with a 1 carry
left column: $D_{16} + A_{16} + 1_{16} = 13_{10} + 10_{10} + 1_{10} = 24_{10}$
$24_{10} - 16_{10} = 8_{10} = 8_{16}$ with a 1 carry

Related Problem Add $4C_{16}$ and $3A_{16}$.

Hexadecimal Subtraction

As you have learned, the 2's complement allows you to subtract by adding binary numbers. Since a hexadecimal number can be used to represent a binary number, it can also be used to represent the 2's complement of a binary number.

There are three ways to get the 2's complement of a hexadecimal number. Method 1 is the most common and easiest to use. Methods 2 and 3 are alternate methods.

Method 1. Convert the hexadecimal number to binary. Take the 2's complement of the binary number. Convert the result to hexadecimal. This is illustrated in Figure 2–4.

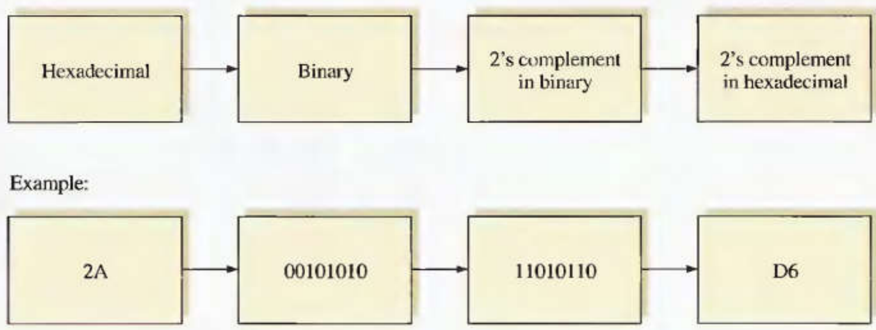


FIGURE 2–4

Getting the 2's complement of a hexadecimal number, Method 1.

Method 2. Subtract the hexadecimal number from the maximum hexadecimal number and add 1. This is illustrated in Figure 2–5.

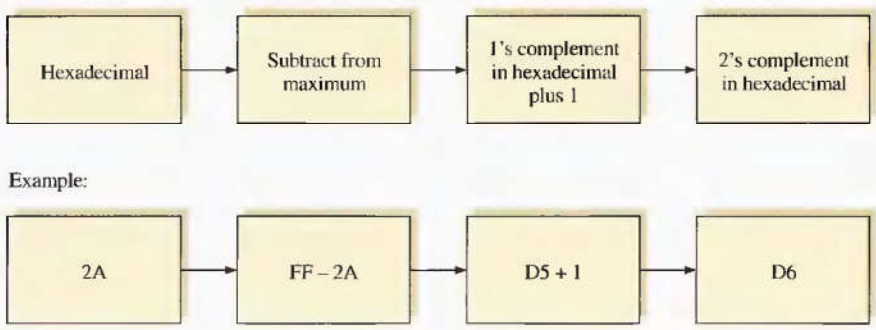


FIGURE 2–5

Getting the 2's complement of a hexadecimal number, Method 2.

Method 3. Write the sequence of single hexadecimal digits. Write the sequence in reverse below the forward sequence. The 1's complement of each hex digit is the digit directly below it. Add 1 to the resulting number to get the 2's complement. This is illustrated in Figure 2–6.

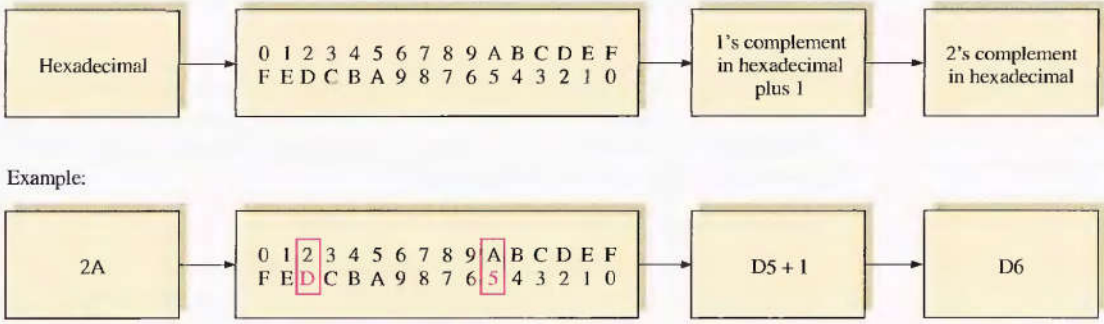


FIGURE 2–6

Getting the 2's complement of a hexadecimal number, Method 3.

EXAMPLE 2–30

Subtract the following hexadecimal numbers:

(a) $84_{16} - 2A_{16}$ **(b)** $C3_{16} - 0B_{16}$

Solution **(a)** $2A_{16} = 00101010$

2's complement of $2A_{16} = 11010110 = D6_{16}$ (using Method 1)

$$\begin{array}{r l} 84_{16} & \\ +\ D6_{16} & \text{Add} \\ \hline \cancel{1}5A_{16} & \text{Drop carry, as in 2's complement addition} \end{array}$$

The difference is **$5A_{16}$**.

(b) $0B_{16} = 00001011$

2's complement of $0B_{16} = 11110101 = F5_{16}$ (using Method 1)

$$\begin{array}{r l} C3_{16} & \\ +\ F5_{16} & \text{Add} \\ \hline \cancel{1}B8_{16} & \text{Drop carry} \end{array}$$

The difference is **$B8_{16}$**.

Related Problem Subtract 173_{16} from BCD_{16}.

SECTION 2–8 REVIEW

1. Convert the following binary numbers to hexadecimal:
   (a) 10110011 (b) 110011101000
2. Convert the following hexadecimal numbers to binary:
   (a) 57_{16} (b) $3A5_{16}$ (c) $F80B_{16}$
3. Convert $9B30_{16}$ to decimal.
4. Convert the decimal number 573 to hexadecimal.
5. Add the following hexadecimal numbers directly:
   (a) $18_{16} + 34_{16}$ (b) $3F_{16} + 2A_{16}$
6. Subtract the following hexadecimal numbers:
   (a) $75_{16} - 21_{16}$ (b) $94_{16} - 5C_{16}$

2–9 OCTAL NUMBERS

Like the hexadecimal number system, the octal number system provides a convenient way to express binary numbers and codes. However, it is used less frequently than hexadecimal in conjunction with computers and microprocessors to express binary quantities for input and output purposes.

After completing this section, you should be able to

- Write the digits of the octal number system
- Convert from octal to decimal
- Convert from decimal to octal
- Convert from octal to binary
- Convert from binary to octal

The **octal** number system is composed of eight digits, which are

0, 1, 2, 3, 4, 5, 6, 7

To count above 7, begin another column and start over:

10, 11, 12, 13, 14, 15, 16, 17, 20, 21, . . .

The octal number system has a base of 8.

Counting in octal is similar to counting in decimal, except that the digits 8 and 9 are not used. To distinguish octal numbers from decimal numbers or hexadecimal numbers, we will use the subscript 8 to indicate an octal number. For instance, 15_8 in octal is equivalent to 13_{10} in decimal and D in hexadecimal. Sometimes you may see an "o" or a "Q" following an octal number.

Octal-to-Decimal Conversion

Since the octal number system has a base of eight, each successive digit position is an increasing power of eight, beginning in the right-most column with 8^0. The evaluation of an octal number in terms of its decimal equivalent is accomplished by multiplying each digit by its weight and summing the products, as illustrated here for 2374_8.

$$\begin{aligned}
&\text{Weight:} \quad 8^3\ 8^2\ 8^1\ 8^0 \\
&\text{Octal number:} \quad 2\ \ 3\ \ 7\ \ 4 \\
2374_8 &= (2 \times 8^3) + (3 \times 8^2) + (7 \times 8^1) + (4 \times 8^0) \\
&= (2 \times 512) + (3 \times 64) + (7 \times 8) + (4 \times 1) \\
&= 1024 + 192 + 56 + 4 = 1276_{10}
\end{aligned}$$

Decimal-to-Octal Conversion

A method of converting a decimal number to an octal number is the repeated division-by-8 method, which is similar to the method used in the conversion of decimal numbers to binary or to hexadecimal. To show how it works, let's convert the decimal number 359 to octal. Each successive division by 8 yields a remainder that becomes a digit in the equivalent octal number. The first remainder generated is the least significant digit (LSD).

Remainder

$$\frac{359}{8} = 44.875 \rightarrow 0.875 \times 8 = 7$$

$$\frac{44}{8} = 5.5 \longrightarrow 0.5 \times 8 = 4$$

$$\frac{5}{8} = 0.625 \longrightarrow 0.625 \times 8 = 5$$

Stop when whole number quotient is zero

5 4 7 Octal number

MSD LSD

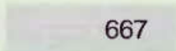
CALCULATOR TUTORIAL

Conversion of a Decimal Number to an Octal Number

Example Convert decimal 439 to octal.

TI-86 **Step 1.** 2nd 1 F3 (BASE)
Step 2. 4 3 3
Step 3. F3
Step 4. ENTER

TI-36X **Step 1.** 3rd EE (DEC)
Step 2. 4 0 0
Step 3. 3rd) (OCT)



Octal-to-Binary Conversion

Because each octal digit can be represented by a 3-bit binary number, it is very easy to convert from octal to binary. Each octal digit is represented by three bits as shown in Table 2–4.

Octal is a convenient way to represent binary numbers, but it is not as commonly used as hexadecimal.

TABLE 2–4

Octal/binary conversion.

OCTAL DIGIT	0	1	2	3	4	5	6	7
BINARY	000	001	010	011	100	101	110	111

To convert an octal number to a binary number, simply replace each octal digit with the appropriate three bits.

EXAMPLE 2–31

Convert each of the following octal numbers to binary:

(a) 13_8 **(b)** 25_8 **(c)** 140_8 **(d)** 7526_8

Solution

(a) 1 3 → $\overline{001}\,\overline{011}$

(b) 2 5 → $\overline{010}\,\overline{101}$

(c) 1 4 0 → $\overline{001}\,\overline{100}\,\overline{000}$

(d) 7 5 2 6 → $\overline{111}\,\overline{101}\,\overline{010}\,\overline{110}$

Related Problem Convert each of the binary numbers to decimal and verify that each value agrees with the decimal value of the corresponding octal number.

Binary-to-Octal Conversion

Conversion of a binary number to an octal number is the reverse of the octal-to-binary conversion. The procedure is as follows: Start with the right-most group of three bits and, moving from right to left, convert each 3-bit group to the equivalent octal digit. If there are not three bits available for the left-most group, add either one or two zeros to make a complete group. These leading zeros do not affect the value of the binary number.

EXAMPLE 2–32

Convert each of the following binary numbers to octal:

(a) 110101 **(b)** 101111001 **(c)** 100110011010 **(d)** 11010000100

Solution

(a) $\underbrace{110}_{6}\underbrace{101}_{5}$ = $\mathbf{65_8}$

(b) $\underbrace{101}_{5}\underbrace{111}_{7}\underbrace{001}_{1}$ = $\mathbf{571_8}$

(c) $\underbrace{100}_{4}\underbrace{110}_{6}\underbrace{011}_{3}\underbrace{010}_{2}$ = $\mathbf{4632_8}$

(d) $\underbrace{011}_{3}\underbrace{010}_{2}\underbrace{000}_{0}\underbrace{100}_{4}$ = $\mathbf{3204_8}$

Related Problem Convert the binary number 1010101000111110010 to octal.

SECTION 2–9 REVIEW

1. Convert the following octal numbers to decimal:
   (a) 73_8 (b) 125_8
2. Convert the following decimal numbers to octal:
   (a) 98_{10} (b) 163_{10}
3. Convert the following octal numbers to binary:
   (a) 46_8 (b) 723_8 (c) 5624_8
4. Convert the following binary numbers to octal:
   (a) 110101111 (b) 1001100010 (c) 10111111001

2–10 BINARY CODED DECIMAL (BCD)

Binary coded decimal (BCD) is a way to express each of the decimal digits with a binary code. There are only ten code groups in the BCD system, so it is very easy to convert between decimal and BCD. Because we like to read and write in decimal, the BCD code provides an excellent interface to binary systems. Examples of such interfaces are keypad inputs and digital readouts.

After completing this section, you should be able to

■ Convert each decimal digit to BCD ■ Express decimal numbers in BCD ■ Convert from BCD to decimal ■ Add BCD numbers

The 8421 Code

In BCD, 4 bits represent each decimal digit.

The 8421 code is a type of **BCD** (binary coded decimal) code. Binary coded decimal means that each decimal digit, 0 through 9, is represented by a binary code of four bits. The designation 8421 indicates the binary weights of the four bits (2^3, 2^2, 2^1, 2^0). The ease of conversion between 8421 code numbers and the familiar decimal numbers is the main advantage of this code. All you have to remember are the ten binary combinations that represent the ten decimal digits as shown in Table 2–5. The 8421 code is the predominant BCD code, and when we refer to BCD, we always mean the 8421 code unless otherwise stated.

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

◄ TABLE 2–5

Decimal/BCD conversion.

Invalid Codes You should realize that, with four bits, sixteen numbers (0000 through 1111) can be represented but that, in the 8421 code, only ten of these are used. The six code combinations that are not used—1010, 1011, 1100, 1101, 1110, and 1111—are invalid in the 8421 BCD code.

To express any decimal number in BCD, simply replace each decimal digit with the appropriate 4-bit code, as shown by Example 2–33.

EXAMPLE 2–33

Convert each of the following decimal numbers to BCD:

(a) 35 **(b)** 98 **(c)** 170 **(d)** 2469

Solution

(a) $\overbrace{0011}^{3}\overbrace{0101}^{5}$ **(b)** $\overbrace{1001}^{9}\overbrace{1000}^{8}$

(c) $\overbrace{0001}^{1}\overbrace{0111}^{7}\overbrace{0000}^{0}$ **(d)** $\overbrace{0010}^{2}\overbrace{0100}^{4}\overbrace{0110}^{6}\overbrace{1001}^{9}$

Related Problem Convert the decimal number 9673 to BCD.

It is equally easy to determine a decimal number from a BCD number. Start at the right-most bit and break the code into groups of four bits. Then write the decimal digit represented by each 4-bit group.

EXAMPLE 2–34

Convert each of the following BCD codes to decimal:

(a) 10000110 **(b)** 001101010001 **(c)** 1001010001110000

Solution

(a) $\underbrace{1000}_{8}\underbrace{0110}_{6}$ **(b)** $\underbrace{0011}_{3}\underbrace{0101}_{5}\underbrace{0001}_{1}$ **(c)** $\underbrace{1001}_{9}\underbrace{0100}_{4}\underbrace{0111}_{7}\underbrace{0000}_{0}$

Related Problem Convert the BCD code 10000010001001110110 to decimal.

BCD Addition

BCD is a numerical code and can be used in arithmetic operations. Addition is the most important operation because the other three operations (subtraction, multiplication, and division) can be accomplished by the use of addition. Here is how to add two BCD numbers:

Step 1. Add the two BCD numbers, using the rules for binary addition in Section 2–4.

Step 2. If a 4-bit sum is equal to or less than 9, it is a valid BCD number.

Step 3. If a 4-bit sum is greater than 9, or if a carry out of the 4-bit group is generated, it is an invalid result. Add 6 (0110) to the 4-bit sum in order to skip the six invalid states and return the code to 8421. If a carry results when 6 is added, simply add the carry to the next 4-bit group.

Example 2–35 illustrates BCD additions in which the sum in each 4-bit column is equal to or less than 9, and the 4-bit sums are therefore valid BCD numbers. Example 2–36 illustrates the procedure in the case of invalid sums (greater than 9 or a carry).

EXAMPLE 2–35

Add the following BCD numbers:

(a) 0011 + 0100 **(b)** 00100011 + 00010101

(c) 10000110 + 00010011 **(d)** 010001010000 + 010000010111

Solution The decimal number additions are shown for comparison.

(a)

0011		3
+ 0100		+ 4
0111		7

(b)

0010	0011	23
+ 0001	0101	+ 15
0011	**1000**	38

(c)

1000	0110	86
+ 0001	0011	+ 13
1001	**1001**	99

(d)

0100	0101	0000	450
+ 0100	0001	0111	+ 417
1000	**0110**	**0111**	867

Note that in each case the sum in any 4-bit column does not exceed 9, and the results are valid BCD numbers.

Related Problem Add the BCD numbers: 1001000001000011 + 0000100100100101.

EXAMPLE 2–36

Add the following BCD numbers

(a) 1001 + 0100 **(b)** 1001 + 1001

(c) 00010110 + 00010101 **(d)** 01100111 + 01010011

Solution The decimal number additions are shown for comparison.

(a)

	1001		9
	+ 0100		+4
	1101	Invalid BCD number (>9)	13
	+ 0110	Add 6	
0001	**0011**	Valid BCD number	
↓	↓		
1	3		

(b)

	1001		9
	+ 1001		+ 9
1	0010	Invalid because of carry	18
	+ 0110	Add 6	
0001	**1000**	Valid BCD number	
↓	↓		
1	8		

(c)	0001	0110		16
	+ 0001	0101		+ 15
	0010	1011	Right group is invalid (>9), left group is valid.	31
		+ 0110	Add 6 to invalid code. Add carry, 0001, to next group.	
	0011	**0001**	Valid BCD number	
	↓	↓		
	3	1		

(d)		0110	0111		67
		+ 0101	0011		+ 53
		1011	1010	Both groups are invalid (>9)	120
		+ 0110	+ 0110	Add 6 to both groups	
	0001	**0010**	**0000**	Valid BCD number	
	↓	↓	↓		
	1	2	0		

Related Problem Add the BCD numbers: 01001000 + 00110100.

SECTION 2–10 REVIEW

1. What is the binary weight of each 1 in the following BCD numbers?
   (a) 0010 **(b)** 1000 **(c)** 0001 **(d)** 0100
2. Convert the following decimal numbers to BCD:
   (a) 6 **(b)** 15 **(c)** 273 **(d)** 849
3. What decimal numbers are represented by each BCD code?
   (a) 10001001 **(b)** 001001111000 **(c)** 000101010111
4. In BCD addition, when is a 4-bit sum invalid?

2–11 DIGITAL CODES

Many specialized codes are used in digital systems. You have just learned about the BCD code; now let's look at a few others. Some codes are strictly numeric, like BCD, and others are alphanumeric; that is, they are used to represent numbers, letters, symbols, and instructions. The codes introduced in this section are the Gray code and the ASCII code.

After completing this section, you should be able to

- Explain the advantage of the Gray code
- Convert between Gray code and binary
- Use the ASCII code

The Gray Code

The **Gray code** is unweighted and is not an arithmetic code; that is, there are no specific weights assigned to the bit positions. The important feature of the Gray code is that *it exhibits only a single bit change from one code word to the next in sequence.* This property is important in many applications, such as shaft position encoders, where error susceptibility increases with the number of bit changes between adjacent numbers in a sequence.

The single bit change characteristic of the Gray code minimizes the chance for error.

Table 2–6 is a listing of the 4-bit Gray code for decimal numbers 0 through 15. Binary numbers are shown in the table for reference. Like binary numbers, *the Gray code can have any number of bits.* Notice the single-bit change between successive Gray code words. For instance, in going from decimal 3 to decimal 4, the Gray code changes from 0010 to 0110, while the binary code changes from 0011 to 0100, a change of three bits. The only bit change is in the third bit from the right in the Gray code; the others remain the same.

TABLE 2–6

Four-bit Gray code.

DECIMAL	BINARY	GRAY CODE	DECIMAL	BINARY	GRAY CODE
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Binary-to-Gray Code Conversion Conversion between binary code and Gray code is sometimes useful. The following rules explain how to convert from a binary number to a Gray code word:

1. The most significant bit (left-most) in the Gray code is the same as the corresponding MSB in the binary number.
2. Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries.

For example, the conversion of the binary number 10110 to Gray code is as follows:

1 − + → 0 − + → 1 − + → 1 − + → 0 Binary
↓ ↓ ↓ ↓ ↓
1 1 1 0 1 Gray

The Gray code is 11101.

Gray-to-Binary Conversion To convert from Gray code to binary, use a similar method; however, there are some differences. The following rules apply:

1. The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.
2. Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.

For example, the conversion of the Gray code word 11011 to binary is as follows:

1 1 0 1 1 Gray
↓ + ↗ ↓ + ↗ ↓ + ↗ ↓ + ↗ ↓
1 0 0 1 0 Binary

The binary number is 10010.

EXAMPLE 2–37

(a) Convert the binary number 11000110 to Gray code.

(b) Convert the Gray code 10101111 to binary.

Solution

(a) Binary to Gray code:

$$1 \xrightarrow{+} 1 \xrightarrow{+} 0 \xrightarrow{+} 0 \xrightarrow{+} 0 \xrightarrow{+} 1 \xrightarrow{+} 1 \xrightarrow{+} 0$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

$$\mathbf{1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1}$$

(b) Gray code to binary:

$$1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1$$

$$\downarrow \; + \; \downarrow \; + \; \downarrow \; + \; \downarrow \; + \; \downarrow \; + \; \downarrow \; + \; \downarrow \; + \; \downarrow$$

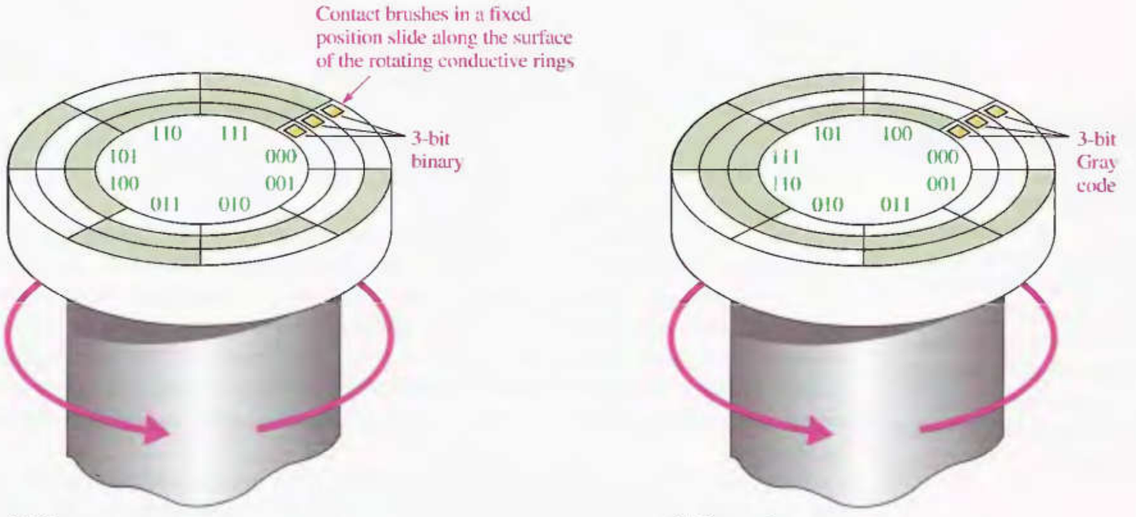
$$\mathbf{1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0}$$

Related Problem

(a) Convert binary 101101 to Gray code. **(b)** Convert Gray code 100111 to binary.

An Application

A simplified diagram of a 3-bit shaft position encoder mechanism is shown in Figure 2–7. Basically, there are three concentric conductive rings that are segmented into eight sectors. The more sectors there are, the more accurately the position can be represented, but we are using only eight for purposes of illustration. Each sector of each ring is fixed at either a high-level or a low-level voltage to represent 1s and 0s. A 1 is indicated by a color sector and a 0 by a white sector. As the rings rotate with the shaft, they make contact with a brush arrangement that is in a fixed position and to which output lines are connected. As the shaft rotates counterclockwise through 360°, the eight sectors move past the three brushes producing a 3-bit binary output that indicates the shaft position.



(a) Binary (b) Gray code

FIGURE 2–7

A simplified illustration of how the Gray code solves the error problem in shaft position encoders.

In Figure 2–7(a), the sectors are arranged in a straight binary pattern, so that the brushes go from 000 to 001 to 010 to 011, and so on. When the brushes are on color sectors, they output a 1 and when on white sectors, they output a 0. If one brush is slightly ahead of the others during the transition from one sector to the next, an erroneous output can occur. Consider what happens when the brushes are on the 111 sector and about to enter the 000 sector. If the MSB brush is slightly ahead, the position would be incorrectly indicated by a transitional 011 instead of a 111 or a 000. In this type of application, it is virtually impossible to maintain precise mechanical alignment of all the brushes; therefore, some error will always occur at many of the transitions between sectors.

The Gray code is used to eliminate the error problem which is inherent in the binary code. As shown in Figure 2–7(b), the Gray code assures that only one bit will change between adjacent sectors. This means that even though the brushes may not be in precise alignment, there will never be a transitional error. For example, let's again consider what happens when the brushes are on the 111 sector and about to move into the next sector, 101. The only two possible outputs during the transition are 111 and 101, no matter how the brushes are aligned. A similar situation occurs at the transitions between each of the other sectors.

Alphanumeric Codes

In order to communicate, you need not only numbers, but also letters and other symbols. In the strictest sense, **alphanumeric** codes are codes that represent numbers and alphabetic characters (letters). Most such codes, however, also represent other characters such as symbols and various instructions necessary for conveying information.

At a minimum, an alphanumeric code must represent 10 decimal digits and 26 letters of the alphabet, for a total of 36 items. This number requires six bits in each code combination because five bits are insufficient ($2^5 = 32$). There are 64 total combinations of six bits, so there are 28 unused code combinations. Obviously, in many applications, symbols other than just numbers and letters are necessary to communicate completely. You need spaces, periods, colons, semicolons, question marks, etc. You also need instructions to tell the receiving system what to do with the information. With codes that are six bits long, you can handle decimal numbers, the alphabet, and 28 other symbols. This should give you an idea of the requirements for a basic alphanumeric code. The ASCII is the most common alphanumeric code and is covered next.

ASCII

COMPUTER NOTE

A computer keyboard has a dedicated microprocessor that constantly scans keyboard circuits to detect when a key has been pressed and released. A unique scan code is produced by computer software representing that particular key. The scan code is then converted to an alphanumeric code (ASCII) for use by the computer.

ASCII is the abbreviation for American Standard Code for Information Interchange. Pronounced "askee," ASCII is a universally accepted alphanumeric code used in most computers and other electronic equipment. Most computer keyboards are standardized with the ASCII. When you enter a letter, a number, or control command, the corresponding ASCII code goes into the computer.

ASCII has 128 characters and symbols represented by a 7-bit binary code. Actually, ASCII can be considered an 8-bit code with the MSB always 0. This 8-bit code is 00 through 7F in hexadecimal. The first thirty-two ASCII characters are nongraphic commands that are never printed or displayed and are used only for control purposes. Examples of the control characters are "null," "line feed," "start of text," and "escape." The other characters are graphic symbols that can be printed or displayed and include the letters of the alphabet (lowercase and uppercase), the ten decimal digits, punctuation signs and other commonly used symbols.

Table 2–7 is a listing of the ASCII code showing the decimal, hexadecimal, and binary representations for each character and symbol. The left section of the table lists the names of the 32 control characters (00 through 1F hexadecimal). The graphic symbols are listed in the rest of the table (20 through 7F hexadecimal).

TABLE 2-7

American Standard Code for Information Interchange (ASCII).

CONTROL CHARACTERS				GRAPHIC SYMBOLS											
NAME	DEC	BINARY	HEX	SYMBOL	DEC	BINARY	HEX	SYMBOL	DEC	BINARY	HEX	SYMBOL	DEC	BINARY	HEX
NUL	0	0000000	00	space	32	0100000	20	@	64	1000000	40	`	96	1100000	60
SOH	1	0000001	01	!	33	0100001	21	A	65	1000001	41	a	97	1100001	61
STX	2	0000010	02	"	34	0100010	22	B	66	1000010	42	b	98	1100010	62
ETX	3	0000011	03	#	35	0100011	23	C	67	1000011	43	c	99	1100011	63
EOT	4	0000100	04	$	36	0100100	24	D	68	1000100	44	d	100	1100100	64
ENQ	5	0000101	05	%	37	0100101	25	E	69	1000101	45	e	101	1100101	65
ACK	6	0000110	06	&	38	0100110	26	F	70	1000110	46	f	102	1100110	66
BEL	7	0000111	07	'	39	0100111	27	G	71	1000111	47	g	103	1100111	67
BS	8	0001000	08	(	40	0101000	28	H	72	1001000	48	h	104	1101000	68
HT	9	0001001	09	)	41	0101001	29	I	73	1001001	49	i	105	1101001	69
LF	10	0001010	0A	*	42	0101010	2A	J	74	1001010	4A	j	106	1101010	6A
VT	11	0001011	0B	+	43	0101011	2B	K	75	1001011	4B	k	107	1101011	6B
FF	12	0001100	0C	,	44	0101100	2C	L	76	1001100	4C	l	108	1101100	6C
CR	13	0001101	0D	–	45	0101101	2D	M	77	1001101	4D	m	109	1101101	6D
SO	14	0001110	0E	.	46	0101110	2E	N	78	1001110	4E	n	110	1101110	6E
SI	15	0001111	0F	/	47	0101111	2F	O	79	1001111	4F	o	111	1101111	6F
DLE	16	0010000	10	0	48	0110000	30	P	80	1010000	50	p	112	1110000	70
DC1	17	0010001	11	1	49	0110001	31	Q	81	1010001	51	q	113	1110001	71
DC2	18	0010010	12	2	50	0110010	32	R	82	1010010	52	r	114	1110010	72
DC3	19	0010011	13	3	51	0110011	33	S	83	1010011	53	s	115	1110011	73
DC4	20	0010100	14	4	52	0110100	34	T	84	1010100	54	t	116	1110100	74
NAK	21	0010101	15	5	53	0110101	35	U	85	1010101	55	u	117	1110101	75
SYN	22	0010110	16	6	54	0110110	36	V	86	1010110	56	v	118	1110110	76
ETB	23	0010111	17	7	55	0110111	37	W	87	1010111	57	w	119	1110111	77
CAN	24	0011000	18	8	56	0111000	38	X	88	1011000	58	x	120	1111000	78
EM	25	0011001	19	9	57	0111001	39	Y	89	1011001	59	y	121	1111001	79
SUB	26	0011010	1A	:	58	0111010	3A	Z	90	1011010	5A	z	122	1111010	7A
ESC	27	0011011	1B	;	59	0111011	3B	[	91	1011011	5B	{	123	1111011	7B
FS	28	0011100	1C	<	60	0111100	3C	\	92	1011100	5C	|	124	1111100	7C
GS	29	0011101	1D	=	61	0111101	3D	]	93	1011101	5D	}	125	1111101	7D
RS	30	0011110	1E	>	62	0111110	3E	^	94	1011110	5E	~	126	1111110	7E
US	31	0011111	1F	?	63	0111111	3F	_	95	1011111	5F	Del	127	1111111	7F

EXAMPLE 2–38

Determine the binary ASCII codes that are entered from the computer's keyboard when the following BASIC program statement is typed in. Also express each code in hexadecimal.

20 PRINT "A=";X

Solution The ASCII code for each symbol is found in Table 2–7.

Symbol	Binary	Hexadecimal
2	0110010	32_{16}
0	0110000	30_{16}
Space	0100000	20_{16}
P	1010000	50_{16}
R	1010010	52_{16}
I	1001001	49_{16}
N	1001110	$4E_{16}$
T	1010100	54_{16}
Space	0100000	20_{16}
"	0100010	22_{16}
A	1000001	41_{16}
=	0111101	$3D_{16}$
"	0100010	22_{16}
;	0111011	$3B_{16}$
X	1011000	58_{16}

Related Problem Determine the sequence of ASCII codes required for the following program statement and express them in hexadecimal:

80 INPUT Y

The ASCII Control Characters The first thirty-two codes in the ASCII table (Table 2–7) represent the control characters. These are used to allow devices such as a computer and printer to communicate with each other when passing information and data. Table 2–8 lists the control characters and the control key function that allows them to be entered directly from an ASCII keyboard by pressing the control key (CTRL) and the corresponding symbol. A brief description of each control character is also given.

Extended ASCII Characters

In addition to the 128 standard ASCII characters, there are an additional 128 characters that were adopted by IBM for use in their PCs (personal computers). Because of the popularity of the PC, these particular extended ASCII characters are also used in applications other than PCs and have become essentially an unofficial standard.

The extended ASCII characters are represented by an 8-bit code series from hexadecimal 80 to hexadecimal FF.

◄ TABLE 2–8

ASCII control characters.

NAME	DECIMAL	HEX	KEY	DESCRIPTION
NUL	0	00	CTRL @	null character
SOH	1	01	CTRL A	start of header
STX	2	02	CTRL B	start of text
ETX	3	03	CTRL C	end of text
EOT	4	04	CTRL D	end of transmission
ENQ	5	05	CTRL E	enquire
ACK	6	06	CTRL F	acknowledge
BEL	7	07	CTRL G	bell
BS	8	08	CTRL H	backspace
HT	9	09	CTRL I	horizontal tab
LF	10	0A	CTRL J	line feed
VT	11	0B	CTRL K	vertical tab
FF	12	0C	CTRL L	form feed (new page)
CR	13	0D	CTRL M	carriage return
SO	14	0E	CTRL N	shift out
SI	15	0F	CTRL O	shift in
DLE	16	10	CTRL P	data link escape
DC1	17	11	CTRL Q	device control 1
DC2	18	12	CTRL R	device control 2
DC3	19	13	CTRL S	device control 3
DC4	20	14	CTRL T	device control 4
NAK	21	15	CTRL U	negative acknowledge
SYN	22	16	CTRL V	synchronize
ETB	23	17	CTRL W	end of transmission block
CAN	24	18	CTRL X	cancel
EM	25	19	CTRL Y	end of medium
SUB	26	1A	CTRL Z	substitute
ESC	27	1B	CTRL [	escape
FS	28	1C	CTRL /	file separator
GS	29	1D	CTRL]	group separator
RS	30	1E	CTRL ^	record separator
US	31	1F	CTRL _	unit separator

The extended ASCII contains characters in the following general categories:

1. Foreign (non-English) alphabetic characters
2. Foreign currency symbols
3. Greek letters
4. Mathematical symbols
5. Drawing characters
6. Bar graphing characters
7. Shading characters

Table 2–9 is a list of the extended ASCII character set with the decimal and hexadecimal representations.

TABLE 2–9

Extended ASCII characters.

SYMBOL	DEC	HEX	SYMBOL	DEC	HEX	SYMBOL	DEC	HEX	SYMBOL	DEC	HEX
Ç	128	80	á	160	A0	└	192	C0	α	224	E0
ü	129	81	í	161	A1	┴	193	C1	β	225	E1
é	130	82	ó	162	A2	┬	194	C2	Γ	226	E2
â	131	83	ú	163	A3	├	195	C3	π	227	E3
ä	132	84	ñ	164	A4	─	196	C4	Σ	228	E4
à	133	85	Ñ	165	A5	┼	197	C5	σ	229	E5
å	134	86	ª	166	A6	╞	198	C6	µ	230	E6
ç	135	87	º	167	A7	╟	199	C7	τ	231	E7
ê	136	88	¿	168	A8	╚	200	C8	Φ	232	E8
ë	137	89	⌐	169	A9	╔	201	C9	Θ	233	E9
è	138	8A	¬	170	AA	╩	202	CA	Ω	234	EA
ï	139	8B	½	171	AB	╦	203	CB	δ	235	EB
î	140	8C	¼	172	AC	╠	204	CC	∞	236	EC
ì	141	8D	¡	173	AD	═	205	CD	φ	237	ED
Ä	142	8E	«	174	AE	╬	206	CE	ε	238	EE
Å	143	8F	»	175	AF	╧	207	CF	∩	239	EF
É	144	90	░	176	B0	╨	208	D0	≡	240	F0
æ	145	91	▒	177	B1	╤	209	D1	±	241	F1
Æ	146	92	▓	178	B2	╥	210	D2	≥	242	F2
ô	147	93	│	179	B3	╙	211	D3	≤	243	F3
ö	148	94	┤	180	B4	╘	212	D4	⌠	244	F4
ò	149	95	╡	181	B5	╒	213	D5	⌡	245	F5
û	150	96	╢	182	B6	╓	214	D6	÷	246	F6
ù	151	97	╖	183	B7	╫	215	D7	≈	247	F7
ÿ	152	98	╕	184	B8	╪	216	D8	°	248	F8
Ö	153	99	╣	185	B9	┘	217	D9	•	249	F9
Ü	154	9A	║	186	BA	┌	218	DA	·	250	FA
¢	155	9B	╗	187	BB	█	219	DB	√	251	FB
£	156	9C	╝	188	BC	▄	220	DC	η	252	FC
¥	157	9D	╜	189	BD	▌	221	DD	²	253	FD
Pt	158	9E	╛	190	BE	▐	222	DE	■	254	FE
ƒ	159	9F	┐	191	BF	▀	223	DF	□	255	FF

SECTION 2–11 REVIEW

1. Convert the following binary numbers to the Gray code:
   (a) 1100 (b) 1010 (c) 11010
2. Convert the following Gray codes to binary:
   (a) 1000 (b) 1010 (c) 11101
3. What is the ASCII representation for each of the following characters? Express each as a bit pattern and in hexadecimal notation.
   (a) K (b) r (c) $ (d) +

2–12 ERROR DETECTION AND CORRECTION CODES

In this section, two methods for adding bits to codes to either detect a single-bit error or detect and correct a single-bit error are discussed. The parity method of error detection is introduced, and the Hamming method of single-error detection and correction is covered. When a bit in a given code word is found to be in error, it can be corrected by simply inverting it.

After completing this section, you should be able to

■ Determine if there is an error in a code based on the parity bit ■ Assign the proper parity bit to a code ■ Use the Hamming code for single-error detection and correction ■ Assign the proper parity bits for single-error correction

Parity Method for Error Detection

Many systems use a parity bit as a means for bit **error detection.** Any group of bits contain either an even or an odd number of 1s. A parity bit is attached to a group of bits to make the total number of 1s in a group always even or always odd. An even parity bit makes the total number of 1s even, and an odd parity bit makes the total odd.

A parity bit tells if the number of 1s is odd or even.

A given system operates with even or odd **parity,** but not both. For instance, if a system operates with even parity, a check is made on each group of bits received to make sure the total number of 1s in that group is even. If there is an odd number of 1s, an error has occurred.

As an illustration of how parity bits are attached to a code, Table 2–10 lists the parity bits for each BCD number for both even and odd parity. The parity bit for each BCD number is in the *P* column.

EVEN PARITY		ODD PARITY	
P	BCD	*P*	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

TABLE 2–10

The BCD code with parity bits.

The parity bit can be attached to the code at either the beginning or the end, depending on system design. Notice that the total number of 1s, including the parity bit, is always even for even parity and always odd for odd parity.

Detecting an Error A parity bit provides for the detection of a single bit error (or any odd number of errors, which is very unlikely) but cannot check for two errors in one group. For instance, let's assume that we wish to transmit the BCD code 0101. (Parity can be used with any number of bits; we are using four for illustration.) The total code transmitted, including the even parity bit, is

$$\overset{\text{Even parity bit}}{0}\underbrace{0101}_{\text{BCD code}}$$

Now let's assume that an error occurs in the third bit from the left (the 1 becomes a 0).

$$\overset{\text{Even parity bit}}{0}0\underbrace{0}_{\text{Bit error}}01$$

When this code is received, the parity check circuitry determines that there is only a single 1 (odd number), when there should be an even number of 1s. Because an even number of 1s does not appear in the code when it is received, an error is indicated.

An odd parity bit also provides in a similar manner for the detection of a single error in a given group of bits.

EXAMPLE 2–39

Assign the proper even parity bit to the following code groups:

(a) 1010 **(b)** 111000 **(c)** 101101

(d) 1000111001001 **(e)** 101101011111

Solution Make the parity bit either 1 or 0 as necessary to make the total number of 1s even. The parity bit will be the left-most bit (color).

(a) 01010 **(b)** 1111000 **(c)** 0101101

(d) 01000111001001 **(e)** 1101101011111

Related Problem Add an even parity bit to the 7-bit ASCII code for the letter K.

EXAMPLE 2–40

An odd parity system receives the following code groups: 10110, 11010, 110011, 110101110100, and 1100010101010. Determine which groups, if any, are in error.

Solution Since odd parity is required, any group with an even number of 1s is incorrect. The following groups are in error: **110011** and **1100010101010.**

Related Problem The following ASCII character is received by an odd parity system: 00110111. Is it correct?

The Hamming Error Correction Code

As you have seen, a single parity bit allows for the detection of single-bit errors in a code word. A single parity bit can indicate that there is an error in a certain group of bits. In order to correct a detected error, more information is required because the position of the bit in error must be identified before it can be corrected. More than one parity bit must be included in a group of bits to be able to correct a detected error. In a 7-bit code, there are seven possible single-bit errors. In this case, three parity bits can not only detect an error but can specify the position of the bit in error. The **Hamming code** provides for single-error correction. The following coverage illustrates the construction of a 7-bit Hamming code for single-error correction.

Number of Parity Bits If the number of data bits is designated d, then the number of parity bits, p, is determined by the following relationship:

$$2^p \geq d + p + 1$$

Equation 2–1

For example, if we have four data bits, then p is found by trial and error with Equation 2–1. Let $p = 2$. Then

$$2^p = 2^2 = 4$$

and

$$d + p + 1 = 4 + 2 + 1 = 7$$

Since 2^p must be equal to or greater than $d + p + 1$, the relationship in Equation 2–1 is *not* satisfied. We have to try again. Let $p = 3$. Then

$$2^p = 2^3 = 8$$

and

$$d + p + 1 = 4 + 3 + 1 = 8$$

This value of p satisfies the relationship of Equation 2–1, so three parity bits are required to provide single-error correction for four data bits. It should be noted here that error detection and correction are provided for *all* bits, both parity and data, in a code group; that is, the parity bits also check themselves.

Placement of the Parity Bits in the Code Now that we have found the number of parity bits required in our particular example, we must arrange the bits properly in the code. At this point you should realize that in this example the code is composed of the four data bits and the three parity bits. The left-most bit is designated *bit 1*, the next bit is *bit 2*, and so on as follows:

bit 1, bit 2, bit 3, bit 4, bit 5, bit 6, bit 7

he parity bits are located in the positions that are numbered corresponding to ascending powers of two (1, 2, 4, 8, . . .), as indicated:

$$P_1, \quad P_2, \quad D_1, \quad P_3, \quad D_2, \quad D_3, \quad D_4$$

he symbol P_n designates a particular parity bit, and D_n designates a particular data bit.

Assignment of Parity Bit Values Finally, we must properly assign a 1 or 0 value to each parity bit. Since each parity bit provides a check on certain other bits in the total code, we must know the value of these others in order to assign the parity bit value. To find the bit values, first number each bit position in binary, that is, write the binary number for each decimal position number, as shown in the second two rows of Table 2–11. Next, indicate the parity and data bit locations, as shown in the first row of Table 2–11. Notice that the binary position number of parity bit P_1 has a 1 for its right-most digit. *This parity bit checks all bit positions, including itself, that have 1s in the same location in the binary position numbers.* Therefore, parity bit P_1 checks bit positions 1, 3, 5, and 7.

TABLE 2–11

Bit position table for a 7-bit error correction code.

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4
BIT POSITION	1	2	3	4	5	6	7
BINARY POSITION NUMBER	001	010	011	100	101	110	111
Data bits (D_n)							
Parity bits (P_n)							

The binary position number for parity bit P_2 has a 1 for its middle bit. It checks all bit positions, including itself, that have 1s in this same position. Therefore, parity bit P_2 checks bit positions 2, 3, 6, and 7.

The binary position number for parity bit P_3 has a 1 for its left-most bit. It checks all bit positions, including itself, that have 1s in this same position. Therefore, parity bit P_3 checks bit positions 4, 5, 6, and 7.

In each case, the parity bit is assigned a value to make the quantity of 1s in the set of bits that it checks either odd or even, depending on which is specified. The following examples should make this procedure clear.

EXAMPLE 2–41

Determine the Hamming code for the BCD number 1001 (data bits), using even parity.

Solution

Step 1: Find the number of parity bits required. Let $p = 3$. Then

$$2^p = 2^3 = 8$$

$$d + p + 1 = 4 + 3 + 1 = 8$$

Three parity bits are sufficient.

$$\text{Total code bits} = 4 + 3 = 7$$

Step 2: Construct a bit position table, as shown in Table 2–12, and enter the data bits. Parity bits are determined in the following steps.

▼ **TABLE 2–12**

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4
BIT POSITION	1	2	3	4	5	6	7
BINARY POSITION NUMBER	001	010	011	100	101	110	111
Data bits			1		0	0	1
Parity bits	0	0		1			

Step 3: Determine the parity bits as follows:

Bit P_1 checks bit positions 1, 3, 5, and 7 and must be a 0 for there to be an even number of 1s (2) in this group.

Bit P_2 checks bit positions 2, 3, 6, and 7 and must be a 0 for there to be an even number of 1s (2) in this group.

Bit P_3 checks bit positions 4, 5, 6, and 7 and must be a 1 for there to be an even number of 1s (2) in this group.

Step 4: These parity bits are entered in Table 2–12, and the resulting combined code is 0011001.

Related Problem Determine the Hamming code for the BCD number 1000 using even parity.

EXAMPLE 2–42

Determine the Hamming code for the data bits 10110 using odd parity.

Solution

Step 1: Determine the number of parity bits required. In this case the number of data bits, d, is five. From the previous example we know that $p = 3$ will not work. Try $p = 4$:

$$2^p = 2^4 = 16$$
$$d + p + 1 = 5 + 4 + 1 = 10$$

Four parity bits are sufficient.

$$\text{Total code bits} = 5 + 4 = 9$$

Step 2: Construct a bit position table, Table 2–13, and enter the data bits. Parity bits are determined in the following steps. Notice that P_4 is in bit position 8.

▼ **TABLE 2–13**

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4	P_4	D_5
BIT POSITION	1	2	3	4	5	6	7	8	9
BINARY POSITION NUMBER	0001	0010	0011	0100	0101	0110	0111	1000	1001
Data bits			1		0	1	1		0
Parity bits	1	0		1				1	

Step 3: Determine the parity bits as follows:

Bit P_1 checks bit positions 1, 3, 5, 7, and 9 and must be a 1 for there to be an odd number of 1s (3) in this group.

Bit P_2 checks bit positions 2, 3, 6, and 7 and must be a 0 for there to be an odd number of 1s (3) in this group.

Bit P_3 checks bit positions 4, 5, 6, and 7 and must be a 1 for there to be an odd number of 1s (3) in this group.

Bit P_4 checks bit positions 8 and 9 and must be a 1 for there to be an odd number of 1s (1) in this group.

Step 4: These parity bits are entered in the Table 2–13, and the resulting combined code is 101101110.

Related Problem Determine the Hamming code for 11001 using odd parity.

Detecting and Correcting an Error with the Hamming Code

Now that the Hamming method for constructing an error-correction code has been covered, how do you use it to locate and correct an error? Each parity bit, along with its corresponding group of bits, must be checked for the proper parity. If there are three parity bits in a code word, then three parity checks are made. If there are four parity bits, four checks must be made, and so on. Each parity check will yield a good or a bad result. The total result of all the parity checks indicates the bit, if any, that is in error, as follows:

Step 1. Start with the group checked by P_1.

Step 2. Check the group for proper parity. A 0 represents a good parity check, and 1 represents a bad check.

Step 3. Repeat step 2 for each parity group.

Step 4. The binary number formed by the results of all the parity checks designates the position of the code bit that is in error. This is the *error position code.* The first parity check generates the least significant bit (LSB). If all checks are good, there is no error.

EXAMPLE 2–43

Assume that the code word in Example 2–41 (0011001) is transmitted and that 0010001 is received. The receiver does not "know" what was transmitted and must look for proper parities to determine if the code is correct. Designate any error that has occurred in transmission if even parity is used.

Solution First, make a bit position table, as indicated in Table 2–14.

TABLE 2–14

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4
BIT POSITION	1	2	3	4	5	6	7
BINARY POSITION NUMBER	001	010	011	100	101	110	111
Received code	0	0	1	0	0	0	1

First parity check:

Bit P_1 checks positions 1, 3, 5, and 7.
There are two 1s in this group.
Parity check is good. ⟶ 0 (LSB)

Second parity check:

Bit P_2 checks positions 2, 3, 6, and 7.
There are two 1s in this group.
Parity check is good. ⟶ 0

Third parity check:

Bit P_3 checks positions 4, 5, 6, and 7.
There is one 1 in this group.
Parity check is bad. ⟶ 1 (MSB)

Result:

The error position code is 100 (binary four). This says that the bit in position 4 is in error. It is a 0 and should be a 1. The corrected code is 0011001, which agrees with the transmitted code.

Related Problem Repeat the process illustrated in the example if the received code is 0111001.

EXAMPLE 2–44

The code 101101010 is received. Correct any errors. There are four parity bits, and odd parity is used.

Solution First, make a bit position table like Table 2–15.

TABLE 2–15

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4	P_4	D_5
BIT POSITION	1	2	3	4	5	6	7	8	9
BINARY POSITION NUMBER	0001	0010	0011	0100	0101	0110	0111	1000	1001
Received code	1	0	1	1	0	1	0	1	0

First parity check:
Bit P_1 checks positions 1, 3, 5, 7, and 9.
There are two 1s in this group.
Parity check is bad. ⟶ 1 (LSB)

Second parity check:
Bit P_2 checks positions 2, 3, 6, and 7.
There are two 1s in this group.
Parity check is bad. ⟶ 1

Third parity check:
Bit P_3 checks positions 4, 5, 6, and 7.
There are two 1s in this group.
Parity check is bad. ⟶ 1

Fourth parity check:
Bit P_4 checks positions 8 and 9.
There is one 1 in this group.
Parity check is good. ⟶ 0 (MSB)

Result:
The error position code is 0111 (binary seven). This says that the bit in position 7 is in error. The corrected code is therefore 101101110.

Related Problem The code 101111001 is received. Correct any error if odd parity is used.

SECTION 2–12 REVIEW

1. Which odd-parity code is in error?
   (a) 1011 (b) 1110 (c) 0101 (d) 1000
2. Which even-parity code is in error?
   (a) 11000110 (b) 00101000 (c) 10101010 (d) 11111011
3. Add an even parity bit to the end of each of the following codes.
   (a) 1010100 (b) 0100000 (c) 1110111 (d) 1000110
4. How many parity bits are required for data bits 11010 using the Hamming code?
5. Create the Hamming code for the data bits 0011 using even parity.

SUMMARY

- A binary number is a weighted number in which the weight of each whole number digit is a positive power of two and the weight of each fractional digit is a negative power of two. The whole number weights increase from right to left—from least significant digit to most significant.
- A binary number can be converted to a decimal number by summing the decimal values of the weights of all the 1s in the binary number.
- A decimal whole number can be converted to binary by using the sum-of-weights or the repeated division-by-2 method.
- A decimal fraction can be converted to binary by using the sum-of-weights or the repeated multiplication-by-2 method.
- The basic rules for binary addition are as follows:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

- The basic rules for binary subtraction are as follows:

$$0 - 0 = 0$$
$$1 - 1 = 0$$
$$1 - 0 = 1$$
$$10 - 1 = 1$$

- The 1's complement of a binary number is derived by changing 1s to 0s and 0s to 1s.
- The 2's complement of a binary number can be derived by adding 1 to the 1's complement.
- Binary subtraction can be accomplished with addition by using the 1's or 2's complement method.
- A positive binary number is represented by a 0 sign bit.
- A negative binary number is represented by a 1 sign bit.
- For arithmetic operations, negative binary numbers are represented in 1's complement or 2's complement form.
- In an addition operation, an overflow is possible when both numbers are positive or when both numbers are negative. An incorrect sign bit in the sum indicates the occurrence of an overflow.
- The hexadecimal number system consists of 16 digits and characters, 0 through 9 followed by A through F.
- One hexadecimal digit represents a 4-bit binary number, and its primary usefulness is in simplifying bit patterns and making them easier to read.
- A decimal number can be converted to hexadecimal by the repeated division-by-16 method.
- The octal number system consists of eight digits, 0 through 7.
- A decimal number can be converted to octal by using the repeated division-by-8 method.
- Octal-to-binary conversion is accomplished by simply replacing each octal digit with its 3-bit binary equivalent. The process is reversed for binary-to-octal conversion.
- A decimal number is converted to BCD by replacing each decimal digit with the appropriate 4-bit binary code.
- The ASCII is a 7-bit alphanumeric code that is widely used in computer systems for input and output of information.
- A parity bit is used to detect an error in a code.
- The Hamming code provides for single-error detection and correction.

KEY TERMS

Key terms and other bold terms in the chapter are defined in the end-of-book glossary.

Alphanumeric Consisting of numerals, letters, and other characters.

ASCII American Standard Code for Information Interchange; the most widely used alphanumeric code.

BCD Binary coded decimal; a digital code in which each of the decimal digits, 0 through 9, is represented by a group of four bits.

Byte A group of eight bits.

Floating-point number A number representation based on scientific notation in which the number consists of an exponent and a mantissa.

Hamming code A type of error correction code.

Hexadecimal Describes a number system with a base of 16.

LSB Least significant bit: the right-most bit in a binary whole number or code.

MSB Most significant bit; the left-most bit in a binary whole number or code.

Octal Describes a number system with a base of eight.

Parity In relation to binary codes, the condition of evenness or oddness of the number of 1s in a code group.

SELF-TEST

Answers are at the end of the chapter.

1. $2 \times 10^1 + 8 \times 10^0$ is equal to
   (a) 10 **(b)** 280 **(c)** 2.8 **(d)** 28
2. The binary number 1101 is equal to the decimal number
   (a) 13 **(b)** 49 **(c)** 11 **(d)** 3
3. The binary number 11011101 is equal to the decimal number
   (a) 121 **(b)** 221 **(c)** 441 **(d)** 256
4. The decimal number 17 is equal to the binary number
   (a) 10010 **(b)** 11000 **(c)** 10001 **(d)** 01001
5. The decimal number 175 is equal to the binary number
   (a) 11001111 **(b)** 10101110 **(c)** 10101111 **(d)** 11101111
6. The sum of 11010 + 01111 equals
   (a) 101001 **(b)** 101010 **(c)** 110101 **(d)** 101000
7. The difference of 110 − 010 equals
   (a) 001 **(b)** 010 **(c)** 101 **(d)** 100
8. The 1's complement of 10111001 is
   (a) 01000111 **(b)** 01000110 **(c)** 11000110 **(d)** 10101010
9. The 2's complement of 11001000 is
   (a) 00110111 **(b)** 00110001 **(c)** 01001000 **(d)** 00111000
10. The decimal number +122 is expressed in the 2's complement form as
    (a) 01111010 **(b)** 11111010 **(c)** 01000101 **(d)** 10000101
11. The decimal number −34 is expressed in the 2's complement form as
    (a) 01011110 **(b)** 10100010 **(c)** 11011110 **(d)** 01011101
12. A single-precision floating-point binary number has a total of
    (a) 8 bits **(b)** 16 bits **(c)** 24 bits **(d)** 32 bits
13. In the 2's complement form, the binary number 10010011 is equal to the decimal number
    (a) −19 **(b)** +109 **(c)** +91 **(d)** −109

14. The binary number 101100111001010100001 can be written in octal as
    (a) 5471230_8 (b) 5471241_8 (c) 2634521_8 (d) 23162501_8
15. The binary number 10001101010001101111 can be written in hexadecimal as
    (a) $AD467_{16}$ (b) $8C46F_{16}$ (c) $8D46F_{16}$ (d) $AE46F_{16}$
16. The binary number for $F7A9_{16}$ is
    (a) 1111011110101001 (b) 1110111110101001
    (c) 1111111010110001 (d) 1111011010101001
17. The BCD number for decimal 473 is
    (a) 111011010 (b) 110001110011 (c) 010001110011 (d) 010011110011
18. Refer to Table 2–7. The command STOP in ASCII is
    (a) 1010011101010010011111010000 (b) 1010010100110010011101010000
    (c) 1001010110110110011101010001 (d) 1010011101010010011101100100
19. The code that has an even-parity error is
    (a) 1010011 (b) 1101000 (c) 1001000 (d) 1110111

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

SECTION 2–1 Decimal Numbers

1. What is the weight of the digit 6 in each of the following decimal numbers?
   (a) 1386 **(b)** 54,692 **(c)** 671,920
2. Express each of the following decimal numbers as a power of ten:
   (a) 10 **(b)** 100 **(c)** 10,000 **(d)** 1,000,000
3. Give the value of each digit in the following decimal numbers:
   (a) 471 **(b)** 9356 **(c)** 125,000
4. How high can you count with four decimal digits?

SECTION 2–2 Binary Numbers

5. Convert the following binary numbers to decimal:
   (a) 11 **(b)** 100 **(c)** 111 **(d)** 1000
   (e) 1001 **(f)** 1100 **(g)** 1011 **(h)** 1111
6. Convert the following binary numbers to decimal:
   (a) 1110 **(b)** 1010 **(c)** 11100 **(d)** 10000
   (e) 10101 **(f)** 11101 **(g)** 10111 **(h)** 11111
7. Convert each binary number to decimal:
   (a) 110011.11 **(b)** 101010.01 **(c)** 1000001.111
   (d) 1111000.101 **(e)** 1011100.10101 **(f)** 1110001.0001
   (g) 1011010.1010 **(h)** 1111111.11111
8. What is the highest decimal number that can be represented by each of the following numbers of binary digits (bits)?
   (a) two **(b)** three **(c)** four **(d)** five **(e)** six
   (f) seven **(g)** eight **(h)** nine **(i)** ten **(j)** eleven
9. How many bits are required to represent the following decimal numbers?
   (a) 17 **(b)** 35 **(c)** 49 **(d)** 68
   (e) 81 **(f)** 114 **(g)** 132 **(h)** 205

10. Generate the binary sequence for each decimal sequence:
(a) 0 through 7 **(b)** 8 through 15 **(c)** 16 through 31
(d) 32 through 63 **(e)** 64 through 75

SECTION 2–3 Decimal-to-Binary Conversion

11. Convert each decimal number to binary by using the sum-of-weights method:
(a) 10 **(b)** 17 **(c)** 24 **(d)** 48
(e) 61 **(f)** 93 **(g)** 125 **(h)** 186

12. Convert each decimal fraction to binary using the sum-of-weights method:
(a) 0.32 **(b)** 0.246 **(c)** 0.0981

13. Convert each decimal number to binary using repeated division by 2:
(a) 15 **(b)** 21 **(c)** 28 **(d)** 34
(e) 40 **(f)** 59 **(g)** 65 **(h)** 73

14. Convert each decimal fraction to binary using repeated multiplication by 2:
(a) 0.98 **(b)** 0.347 **(c)** 0.9028

SECTION 2–4 Binary Arithmetic

15. Add the binary numbers:
(a) $11 + 01$ **(b)** $10 + 10$ **(c)** $101 + 11$
(d) $111 + 110$ **(e)** $1001 + 101$ **(f)** $1101 + 1011$

16. Use direct subtraction on the following binary numbers:
(a) $11 - 1$ **(b)** $101 - 100$ **(c)** $110 - 101$
(d) $1110 - 11$ **(e)** $1100 - 1001$ **(f)** $11010 - 10111$

17. Perform the following binary multiplications:
(a) 11×11 **(b)** 100×10 **(c)** 111×101
(d) 1001×110 **(e)** 1101×1101 **(f)** 1110×1101

18. Divide the binary numbers as indicated:
(a) $100 \div 10$ **(b)** $1001 \div 11$ **(c)** $1100 \div 100$

SECTION 2–5 1's and 2's Complements of Binary Numbers

19. Determine the 1's complement of each binary number:
(a) 101 **(b)** 110 **(c)** 1010
(d) 11010111 **(e)** 1110101 **(f)** 00001

20. Determine the 2's complement of each binary number using either method:
(a) 10 **(b)** 111 **(c)** 1001 **(d)** 1101
(e) 11100 **(f)** 10011 **(g)** 10110000 **(h)** 00111101

SECTION 2–6 Signed Numbers

21. Express each decimal number in binary as an 8-bit sign-magnitude number:
(a) $+29$ **(b)** -85 **(c)** $+100$ **(d)** -123

22. Express each decimal number as an 8-bit number in the 1's complement form:
(a) -34 **(b)** $+57$ **(c)** -99 **(d)** $+115$

23. Express each decimal number as an 8-bit number in the 2's complement form:
(a) $+12$ **(b)** -68 **(c)** $+101$ **(d)** -125

24. Determine the decimal value of each signed binary number in the sign-magnitude form:
(a) 10011001 **(b)** 01110100 **(c)** 10111111

25. Determine the decimal value of each signed binary number in the 1's complement form:
 (a) 10011001 **(b)** 01110100 **(c)** 10111111
26. Determine the decimal value of each signed binary number in the 2's complement form:
 (a) 10011001 **(b)** 01110100 **(c)** 10111111
27. Express each of the following sign-magnitude binary numbers in single-precision floating-point format:
 (a) 0111110000101011 **(b)** 100110000011000
28. Determine the values of the following single-precision floating-point numbers:
 (a) 1 10000001 01001001110001000000000
 (b) 0 11001100 10000111110100100000000

SECTION 2–7 Arithmetic Operations with Signed Numbers

29. Convert each pair of decimal numbers to binary and add using the 2's complement form:
 (a) 33 and 15 **(b)** 56 and −27 **(c)** −46 and 25 **(d)** −110 and −84
30. Perform each addition in the 2's complement form:
 (a) 00010110 + 00110011 **(b)** 01110000 + 10101111
31. Perform each addition in the 2's complement form:
 (a) 10001100 + 00111001 **(b)** 11011001 + 11100111
32. Perform each subtraction in the 2's complement form:
 (a) 00110011 − 00010000 **(b)** 01100101 − 11101000
33. Multiply 01101010 by 11110001 in the 2's complement form.
34. Divide 01000100 by 00011001 in the 2's complement form.

SECTION 2–8 Hexadecimal Numbers

35. Convert each hexadecimal number to binary:
 (a) 38_{16} **(b)** 59_{16} **(c)** $A14_{16}$ **(d)** $5C8_{16}$
 (e) 4100_{16} **(f)** $FB17_{16}$ **(g)** $8A9D_{16}$
36. Convert each binary number to hexadecimal:
 (a) 1110 **(b)** 10 **(c)** 10111
 (d) 10100110 **(e)** 1111110000 **(f)** 100110000010
37. Convert each hexadecimal number to decimal:
 (a) 23_{16} **(b)** 92_{16} **(c)** $1A_{16}$ **(d)** $8D_{16}$
 (e) $F3_{16}$ **(f)** EB_{16} **(g)** $5C2_{16}$ **(h)** 700_{16}
38. Convert each decimal number to hexadecimal:
 (a) 8 **(b)** 14 **(c)** 33 **(d)** 52
 (e) 284 **(f)** 2890 **(g)** 4019 **(h)** 6500
39. Perform the following additions:
 (a) $37_{16} + 29_{16}$ **(b)** $A0_{16} + 6B_{16}$ **(c)** $FF_{16} + BB_{16}$
40. Perform the following subtractions:
 (a) $51_{16} - 40_{16}$ **(b)** $C8_{16} - 3A_{16}$ **(c)** $FD_{16} - 88_{16}$

SECTION 2–9 Octal Numbers

41. Convert each octal number to decimal:
 (a) 12_8 **(b)** 27_8 **(c)** 56_8 **(d)** 64_8 **(e)** 103_8
 (f) 557_8 **(g)** 163_8 **(h)** 1024_8 **(i)** 7765_8
42. Convert each decimal number to octal by repeated division by 8:
 (a) 15 **(b)** 27 **(c)** 46 **(d)** 70
 (e) 100 **(f)** 142 **(g)** 219 **(h)** 435

43. Convert each octal number to binary:

(a) 13_8 **(b)** 57_8 **(c)** 101_8 **(d)** 321_8 **(e)** 540_8
(f) 4653_8 **(g)** 13271_8 **(h)** 45600_8 **(i)** 100213_8

44. Convert each binary number to octal:

(a) 111 **(b)** 10 **(c)** 110111
(d) 101010 **(e)** 1100 **(f)** 1011110
(g) 101100011001 **(h)** 10110000011 **(i)** 111111101111000

SECTION 2–10 Binary Coded Decimal (BCD)

45. Convert each of the following decimal numbers to 8421 BCD:

(a) 10 **(b)** 13 **(c)** 18 **(d)** 21 **(e)** 25 **(f)** 36
(g) 44 **(h)** 57 **(i)** 69 **(j)** 98 **(k)** 125 **(l)** 156

46. Convert each of the decimal numbers in Problem 45 to straight binary, and compare the number of bits required with that required for BCD.

47. Convert the following decimal numbers to BCD:

(a) 104 **(b)** 128 **(c)** 132 **(d)** 150 **(e)** 186
(f) 210 **(g)** 359 **(h)** 547 **(i)** 1051

48. Convert each of the BCD numbers to decimal:

(a) 0001 **(b)** 0110 **(c)** 1001
(d) 00011000 **(e)** 00011001 **(f)** 00110010
(g) 01000101 **(h)** 10011000 **(i)** 100001110000

49. Convert each of the BCD numbers to decimal:

(a) 10000000 **(b)** 001000110111
(c) 001101000110 **(d)** 010000100001
(e) 011101010100 **(f)** 100000000000
(g) 100101111000 **(h)** 0001011010000011
(i) 1001000000011000 **(j)** 0110011001100111

50. Add the following BCD numbers:

(a) 0010 + 0001 **(b)** 0101 + 0011
(c) 0111 + 0010 **(d)** 1000 + 0001
(e) 00011000 + 00010001 **(f)** 01100100 + 00110011
(g) 01000000 + 01000111 **(h)** 10000101 + 00010011

51. Add the following BCD numbers:

(a) 1000 + 0110 **(b)** 0111 + 0101
(c) 1001 + 1000 **(d)** 1001 + 0111
(e) 00100101 + 00100111 **(f)** 01010001 + 01011000
(g) 10011000 + 10010111 **(h)** 010101100001 + 011100001000

52. Convert each pair of decimal numbers to BCD, and add as indicated:

(a) 4 + 3 **(b)** 5 + 2 **(c)** 6 + 4 **(d)** 17 + 12
(e) 28 + 23 **(f)** 65 + 58 **(g)** 113 + 101 **(h)** 295 + 157

SECTION 2–11 Digital Codes

53. In a certain application a 4-bit binary sequence cycles from 1111 to 0000 periodically. There are four bit changes, and because of circuit delays, these changes may not occur at the same instant. For example, if the LSB changes first, the number will appear as 1110 during the transition from 1111 to 0000 and may be misinterpreted by the system. Illustrate how the Gray code avoids this problem.

54. Convert each binary number to Gray code:

(a) 11011 **(b)** 1001010 **(c)** 1111011101110

55. Convert each Gray code to binary:

(a) 1010 **(b)** 00010 **(c)** 11000010001

56. Convert each of the following decimal numbers to ASCII. Refer to Table 2–7.

(a) 1 **(b)** 3 **(c)** 6 **(d)** 10 **(e)** 18

(f) 29 **(g)** 56 **(h)** 75 **(i)** 107

57. Determine each ASCII character. Refer to Table 2–7.

(a) 0011000 **(b)** 1001010 **(c)** 0111101

(d) 0100011 **(e)** 0111110 **(f)** 1000010

58. Decode the following ASCII coded message:

1001000 1100101 1101100 1101100 1101111 0101110
0100000 1001000 1101111 1110111 0100000 1100001
1110010 1100101 0100000 1111001 1101111 1110101
0111111

59. Write the message in Problem 58 in hexadecimal.

60. Convert the following computer program statement to ASCII:

30 INPUT A, B

SECTION 2–12 Error Detection and Correction Codes

61. Determine which of the following even parity codes are in error:

(a) 100110010 **(b)** 011101010 **(c)** 10111111010001010

62. Determine which of the following odd parity codes are in error:

(a) 11110110 **(b)** 00110001 **(c)** 01010101010101010

63. Attach the proper even parity bit to each of the following bytes of data:

(a) 10100100 **(b)** 00001001 **(c)** 11111110

64. Determine the even-parity Hamming code for the data bits 1100.

65. Determine the odd-parity Hamming code for the data bits 11001.

66. Correct any error in each of the following Hamming codes with even parity.

(a) 1110100 **(b)** 1000111

67. Correct any error in each of the following Hamming codes with odd parity.

(a) 110100011 **(b)** 100001101