Chapter 1

Digital Systems and Binary Numbers

1.1 DIGITAL SYSTEMS

Digital systems have such a prominent role in everyday life that we refer to the present technological period as the *digital age*. Digital systems are used in communication, business transactions, traffic control, spacecraft guidance, medical treatment, weather monitoring, the Internet, and many other commercial, industrial, and scientific enterprises. We have digital telephones, digital televisions, digital versatile discs, digital cameras, handheld devices, and, of course, digital computers. We enjoy music downloaded to our portable media player (e.g., iPod Touch™) and other handheld devices having high-resolution displays. These devices have graphical user interfaces (GUIs), which enable them to execute commands that appear to the user to be simple, but which, in fact, involve precise execution of a sequence of complex internal instructions. Most, if not all, of these devices have a special-purpose digital computer embedded within them. The most striking property of the digital computer is its generality. It can follow a sequence of instructions, called a program, that operates on given data. The user can specify and change the program or the data according to the specific need. Because of this flexibility, general-purpose digital computers can perform a variety of information-processing tasks that range over a wide spectrum of applications.

One characteristic of digital systems is their ability to represent and manipulate discrete elements of information. Any set that is restricted to a finite number of elements contains discrete information. Examples of discrete sets are the 10 decimal digits, the 26 letters of the alphabet, the 52 playing cards, and the 64 squares of a chessboard. Early digital computers were used for numeric computations. In this case, the discrete elements were the digits. From this application, the term *digital* computer emerged. Discrete elements of information are represented in a digital system by physical quantities from the decimal system. The letters A, B, C, D, E, and F are used for the digits 10, 11, 12, 13, 14, and 15, respectively. An example of a hexadecimal number is

$$(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46{,}687)_{10}$$

The hexadecimal system is used commonly by designers to represent long strings of bits in the addresses, instructions, and data in digital systems. For example, B65F is used to represent 1011011001010000.

As noted before, the digits in a binary number are called *bits*. When a bit is equal to 0, it does not contribute to the sum during the conversion. Therefore, the conversion from binary to decimal can be obtained by adding only the numbers with powers of two corresponding to the bits that are equal to 1. For example,

$$(110101)_2 = 32 + 16 + 4 + 1 = (53)_{10}$$

There are four 1's in the binary number. The corresponding decimal number is the sum of the four powers of two. Zero and the first 24 numbers obtained from 2 to the power of n are listed in Table 1.1. In computer work, 2^{10} is referred to as K (kilo), 2^{20} as M (mega), 2^{30} as G (giga), and 2^{40} as T (tera). Thus, $4K = 2^{12} = 4{,}096$ and $16M = 2^{24} = 16{,}777{,}216$. Computer capacity is usually given in bytes. A *byte* is equal to eight bits and can accommodate (i.e., represent the code of) one keyboard character. A computer hard disk with four gigabytes of storage has a capacity of $4G = 2^{32}$ bytes (approximately 4 billion bytes). A terabyte is 1024 gigabytes, approximately 1 trillion bytes.

Arithmetic operations with numbers in base r follow the same rules as for decimal numbers. When a base other than the familiar base 10 is used, one must be careful to use only the r-allowable digits. Examples of addition, subtraction, and multiplication of two binary numbers are as follows:

augend:	101101	minuend:	101101	multiplicand:	1011
addend:	+100111	subtrahend:	−100111	multiplier:	× 101
sum:	1010100	difference:	000110		1011
					0000
				partial product:	1011
				product:	110111

Table 1.1
Powers of Two

n	2^n	n	2^n	n	2^n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024 (1K)	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096 (4K)	20	1,048,576 (1M)
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

The sum of two binary numbers is calculated by the same rules as in decimal, except that the digits of the sum in any significant position can be only 0 or 1. Any carry obtained in a given significant position is used by the pair of digits one significant position higher. Subtraction is slightly more complicated. The rules are still the same as in decimal, except that the borrow in a given significant position adds 2 to a minuend digit. (A borrow in the decimal system adds 10 to a minuend digit.) Multiplication is simple: The multiplier digits are always 1 or 0; therefore, the partial products are equal either to a shifted (left) copy of the multiplicand or to 0.

1.3 NUMBER-BASE CONVERSIONS

Representations of a number in a different radix are said to be equivalent if they have the same decimal representation. For example, $(0011)_8$ and $(1001)_2$ are equivalent—both have decimal value 9. The conversion of a number in base r to decimal is done by expanding the number in a power series and adding all the terms as shown previously. We now present a general procedure for the reverse operation of converting a decimal number to a number in base r. If the number includes a radix point, it is necessary to separate the number into an integer part and a fraction part, since each part must be converted differently. The conversion of a decimal integer to a number in base r is done by dividing the number and all successive quotients by r and accumulating the remainders. This procedure is best illustrated by example.

EXAMPLE 1.1

Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. Then the quotient is again divided by 2 to give a new quotient and remainder. The process is continued until the integer quotient becomes 0. The *coefficients* of the desired binary number are obtained from the *remainders* as follows:

	Integer Quotient		Remainder	Coefficient
$41/2 =$	20	+	$\frac{1}{2}$	$a_0 = 1$
$20/2 =$	10	+	0	$a_1 = 0$
$10/2 =$	5	+	0	$a_2 = 0$
$5/2 =$	2	+	$\frac{1}{2}$	$a_3 = 1$
$2/2 =$	1	+	0	$a_4 = 0$
$1/2 =$	0	+	$\frac{1}{2}$	$a_5 = 1$

Therefore, the answer is $(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$.

The arithmetic process can be manipula

Integer	Rema
41	
20	
10	
5	
2	
1	
0	

Conversion from decimal integers to any b
that division is done by r instead of 2.

EXAMPLE 1.2

Convert decimal 153 to octal. The require
an integer quotient of 19 and a remainder
quotient of 2 and a remainder of 3. Finall
a remainder of 2. This process can be con

153	
19	
2	
0	

The conversion of a decimal *fraction* t
to that used for integers. However, multipl
instead of remainders are accumulated. A

EXAMPLE 1.3

Convert $(0.6875)_{10}$ to binary. First, 0.6875 is
Then the new fraction is multiplied by 2 to g
is continued until the fraction becomes
accuracy. The coefficients of the binary nur

	Integer
$0.6875 \times 2 =$	1
$0.3750 \times 2 =$	0
$0.7500 \times 2 =$	1
$0.5000 \times 2 =$	1

in decimal, except
0 or 1. Any carry
ne significant posi-
still the same as in
to a minuend digit.
plication is simple:
ts are equal either

valent if they have
e equivalent—both
ecimal is done by
shown previously.
nverting a decimal
t, it is necessary to
each part must be
er in base r is done
lating the remain-

ger quotient of 20
new quotient and
s 0. The *coefficients*
lows:

oefficient
$a_0 = 1$
$a_1 = 0$
$a_2 = 0$
$a_3 = 1$
$a_4 = 0$
$a_5 = 1$

The arithmetic process can be manipulated more conveniently as follows:

Integer	**Remainder**
41	
20	1
10	0
5	0
2	1
1	0
0	1 101001 = answer

Conversion from decimal integers to any base-r system is similar to this example, except that division is done by r instead of 2.

EXAMPLE 1.2

Convert decimal 153 to octal. The required base r is 8. First, 153 is divided by 8 to give an integer quotient of 19 and a remainder of 1. Then 19 is divided by 8 to give an integer quotient of 2 and a remainder of 3. Finally, 2 is divided by 8 to give a quotient of 0 and a remainder of 2. This process can be conveniently manipulated as follows:

153	
19	1
2	3
0	2 = $(231)_8$

The conversion of a decimal *fraction* to binary is accomplished by a method similar to that used for integers. However, multiplication is used instead of division, and integers instead of remainders are accumulated. Again, the method is best explained by example.

EXAMPLE 1.3

Convert $(0.6875)_{10}$ to binary. First, 0.6875 is multiplied by 2 to give an integer and a fraction. Then the new fraction is multiplied by 2 to give a new integer and a new fraction. The process is continued until the fraction becomes 0 or until the number of digits has sufficient accuracy. The coefficients of the binary number are obtained from the integers as follows:

	Integer		**Fraction**	**Coefficient**
$0.6875 \times 2 =$	1	+	0.3750	$a_{-1} = 1$
$0.3750 \times 2 =$	0	+	0.7500	$a_{-2} = 0$
$0.7500 \times 2 =$	1	+	0.5000	$a_{-3} = 1$
$0.5000 \times 2 =$	1	+	0.0000	$a_{-4} = 1$

The sum of two binary numbers is calculated by the same rules as in decimal, except that the digits of the sum in any significant position can be only 0 or 1. Any carry obtained in a given significant position is used by the pair of digits one significant position higher. Subtraction is slightly more complicated. The rules are still the same as in decimal, except that the borrow in a given significant position adds 2 to a minuend digit. (A borrow in the decimal system adds 10 to a minuend digit.) Multiplication is simple: The multiplier digits are always 1 or 0; therefore, the partial products are equal either to a shifted (left) copy of the multiplicand or to 0.

1.3 NUMBER-BASE CONVERSIONS

Representations of a number in a different radix are said to be equivalent if they have the same decimal representation. For example, $(0011)_8$ and $(1001)_2$ are equivalent—both have decimal value 9. The conversion of a number in base r to decimal is done by expanding the number in a power series and adding all the terms as shown previously. We now present a general procedure for the reverse operation of converting a decimal number to a number in base r. If the number includes a radix point, it is necessary to separate the number into an integer part and a fraction part, since each part must be converted differently. The conversion of a decimal integer to a number in base r is done by dividing the number and all successive quotients by r and accumulating the remainders. This procedure is best illustrated by example.

EXAMPLE 1.1

Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. Then the quotient is again divided by 2 to give a new quotient and remainder. The process is continued until the integer quotient becomes 0. The *coefficients* of the desired binary number are obtained from the *remainders* as follows:

	Integer Quotient		Remainder	Coefficient
$41/2 =$	20	+	$\frac{1}{2}$	$a_0 = 1$
$20/2 =$	10	+	0	$a_1 = 0$
$10/2 =$	5	+	0	$a_2 = 0$
$5/2 =$	2	+	$\frac{1}{2}$	$a_3 = 1$
$2/2 =$	1	+	0	$a_4 = 0$
$1/2 =$	0	+	$\frac{1}{2}$	$a_5 = 1$

Therefore, the answer is $(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$.

in decimal, except
0 or 1. Any carry
ıe significant posi-
still the same as in
to a minuend digit.
plication is simple:
ts are equal either

valent if they have
equivalent—both
ecimal is done by
shown previously.
nverting a decimal
, it is necessary to
each part must be
er in base r is done
ılating the remain-

ger quotient of 20
new quotient and
s 0. The *coefficients*
ows:

oefficient

$a_0 = 1$
$a_1 = 0$
$a_2 = 0$
$a_3 = 1$
$a_4 = 0$
$a_5 = 1$

The arithmetic process can be manipulated more conveniently as follows:

Integer	Remainder
41	
20	1
10	0
5	0
2	1
1	0
0	1 101001 = answer

Conversion from decimal integers to any base-r system is similar to this example, except that division is done by r instead of 2.

EXAMPLE 1.2

Convert decimal 153 to octal. The required base r is 8. First, 153 is divided by 8 to give an integer quotient of 19 and a remainder of 1. Then 19 is divided by 8 to give an integer quotient of 2 and a remainder of 3. Finally, 2 is divided by 8 to give a quotient of 0 and a remainder of 2. This process can be conveniently manipulated as follows:

153	
19	1
2	3
0	2 = $(231)_8$

The conversion of a decimal *fraction* to binary is accomplished by a method similar to that used for integers. However, multiplication is used instead of division, and integers instead of remainders are accumulated. Again, the method is best explained by example.

EXAMPLE 1.3

Convert $(0.6875)_{10}$ to binary. First, 0.6875 is multiplied by 2 to give an integer and a fraction. Then the new fraction is multiplied by 2 to give a new integer and a new fraction. The process is continued until the fraction becomes 0 or until the number of digits has sufficient accuracy. The coefficients of the binary number are obtained from the integers as follows:

	Integer		Fraction	Coefficient
$0.6875 \times 2 =$	1	+	0.3750	$a_{-1} = 1$
$0.3750 \times 2 =$	0	+	0.7500	$a_{-2} = 0$
$0.7500 \times 2 =$	1	+	0.5000	$a_{-3} = 1$
$0.5000 \times 2 =$	1	+	0.0000	$a_{-4} = 1$

Therefore, the answer is $(0.6875)_{10} = (0.\,a_{-1}\,a_{-2}\,a_{-3}\,a_{-4})_2 = (0.1011)_2$.

To convert a decimal fraction to a number expressed in base r, a similar procedure is used. However, multiplication is by r instead of 2, and the coefficients found from the integers may range in value from 0 to $r - 1$ instead of 0 and 1.

■

EXAMPLE 1.4

Convert $(0.513)_{10}$ to octal.

$$
\begin{aligned}
0.513 \times 8 &= 4.104 \\
0.104 \times 8 &= 0.832 \\
0.832 \times 8 &= 6.656 \\
0.656 \times 8 &= 5.248 \\
0.248 \times 8 &= 1.984 \\
0.984 \times 8 &= 7.872
\end{aligned}
$$

The answer, to seven significant figures, is obtained from the integer part of the products:

$$(0.513)_{10} = (0.406517\ldots)_8$$

The conversion of decimal numbers with both integer and fraction parts is done by converting the integer and the fraction separately and then combining the two answers. Using the results of Examples 1.1 and 1.3, we obtain

$$(41.6875)_{10} = (101001.1011)_2$$

From Examples 1.2 and 1.4, we have

$$(153.513)_{10} = (231.406517)_8$$

■

1.4 OCTAL AND HEXADECIMAL NUMBERS

The conversion from and to binary, octal, and hexadecimal plays an important role in digital computers, because shorter patterns of hex characters are easier to recognize than long patterns of 1's and 0's. Since $2^3 = 8$ and $2^4 = 16$, each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits. The first 16 numbers in the decimal, binary, octal, and hexadecimal number systems are listed in Table 1.2.

The conversion from binary to octal is easily accomplished by partitioning the binary number into groups of three digits each, starting from the binary point and proceeding to the left and to the right. The corresponding octal digit is then assigned to each group. The following example illustrates the procedure:

$$
\begin{array}{cccccccccl}
(10 & 110 & 001 & 101 & 011 & \cdot & 111 & 100 & 000 & 110)_2 = (26153.7406)_8 \\
2 & 6 & 1 & 5 & 3 & & 7 & 4 & 0 & 6
\end{array}
$$

0.6875)$_{10}$ = $(0.\ a_{-1}\ a_{-2}\ a_{-3}\ a_{-4})_2 = (0.1011)_2$.

action to a number expressed in base r, a similar procedure is
ation is by r instead of 2, and the coefficients found from the
ie from 0 to $r - 1$ instead of 0 and 1.

$$0.513 \times 8 = 4.104$$
$$0.104 \times 8 = 0.832$$
$$0.832 \times 8 = 6.656$$
$$0.656 \times 8 = 5.248$$
$$0.248 \times 8 = 1.984$$
$$0.984 \times 8 = 7.872$$

icant figures, is obtained from the integer part of the products:

$$(0.513)_{10} = (0.406517\ldots)_8$$

nal numbers with both integer and fraction parts is done by
the fraction separately and then combining the two answers.
oles 1.1 and 1.3, we obtain

$$(41.6875)_{10} = (101001.1011)_2$$

1.4, we have

$$(153.513)_{10} = (231.406517)_8$$

ADECIMAL NUMBERS

binary, octal, and hexadecimal plays an important role in digi-
ter patterns of hex characters are easier to recognize than long
e $2^3 = 8$ and $2^4 = 16$, each octal digit corresponds to three
ecimal digit corresponds to four binary digits. The first 16 num-
octal, and hexadecimal number systems are listed in Table 1.2.
ary to octal is easily accomplished by partitioning the binary
e digits each, starting from the binary point and proceeding
The corresponding octal digit is then assigned to each group.
strates the procedure:

$$\begin{array}{cccccccc} 011 & \cdot & 111 & 100 & 000 & 110)_2 = (26153.7406)_8 \\ 3 & & 7 & 4 & 0 & 6 \end{array}$$

Table 1.2
Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Conversion from binary to hexadecimal is similar, except that the binary number is divided into groups of *four* digits:

$$\begin{array}{ccccccc} (10 & 1100 & 0110 & 1011 & \cdot & 1111 & 0010)_2 = (2C6B.F2)_{16} \\ 2 & C & 6 & B & & F & 2 \end{array}$$

The corresponding hexadecimal (or octal) digit for each group of binary digits is easily remembered from the values listed in Table 1.2.

Conversion from octal or hexadecimal to binary is done by reversing the preceding procedure. Each octal digit is converted to its three-digit binary equivalent. Similarly, each hexadecimal digit is converted to its four-digit binary equivalent. The procedure is illustrated in the following examples:

$$\begin{array}{ccccccc} (673.124)_8 = (110 & 111 & 011 & \cdot & 001 & 010 & 100)_2 \\ 6 & 7 & 3 & & 1 & 2 & 4 \end{array}$$

and

$$\begin{array}{ccccc} (306.D)_{16} = (0011 & 0000 & 0110 & \cdot & 1101)_2 \\ 3 & 0 & 6 & & D \end{array}$$

Binary numbers are difficult to work with because they require three or four times as many digits as their decimal equivalents. For example, the binary number 111111111111 is equivalent to decimal 4095. However, digital computers use binary numbers, and it is sometimes necessary for the human operator or user to communicate directly with the machine by means of such numbers. One scheme that retains the binary system in the computer, but reduces the number of digits the human must consider, utilizes the relationship between the binary number system and the octal or hexadecimal system. By this method, the human thinks in terms of octal or hexadecimal numbers and performs the required conversion by inspection when direct communication with the machine is necessary. Thus, the binary number 111111111111 has 12 digits and is expressed in octal as 7777 (4 digits) or in hexadecimal as FFF (3 digits). During communication between people (about binary numbers in the computer), the octal or hexadecimal representation is more desirable because it can be expressed more compactly with a third or a quarter of the number of digits required for the equivalent binary number. Thus, **most computer manuals use either octal or hexadecimal numbers to specify binary quantities.** The choice between them is arbitrary, although hexadecimal tends to win out, since it can represent a byte with two digits.

1.5 COMPLEMENTS OF NUMBERS

Complements are used in digital computers to **simplify the subtraction operation** and for logical manipulation. Simplifying operations leads to simpler, less expensive circuits to implement the operations. There are two types of complements for each base-r system: the radix complement and the diminished radix complement. The first is referred to as the r's complement and the second as the $(r - 1)$'s complement. When the value of the base r is substituted in the name, the two types are referred to as the 2's complement and 1's complement for binary numbers and the 10's complement and 9's complement for decimal numbers.

Diminished Radix Complement

Given a number N in base r having n digits, the $(r - 1)$'s complement of N, i.e., its diminished radix complement, is defined as $(r^n - 1) - N$. For decimal numbers, $r = 10$ and $r - 1 = 9$, so the 9's complement of N is $(10^n - 1) - N$. In this case, 10^n represents a number that consists of a single 1 followed by n 0's. $10^n - 1$ is a number represented by n 9's. For example, if $n = 4$, we have $10^4 = 10{,}000$ and $10^4 - 1 = 9999$. It follows that the 9's complement of a decimal number is obtained by subtracting each digit from 9. Here are some numerical examples:

The 9's complement of 546700 is $999999 - 546700 = 453299$.

The 9's complement of 012398 is $999999 - 012398 = 987601$.

For binary numbers, $r = 2$ and $r - 1 = 1$, so the 1's complement of N is $(2^n - 1) - N$. Again, 2^n is represented by a binary number that consists of a 1 followed by n 0's. $2^n - 1$ is a binary number represented by n 1's. For example, if $n = 4$, we have $2^4 = (10000)_2$ and $2^4 - 1 = (1111)_2$. Thus, the 1's complement of a binary number is obtained by subtracting each digit from 1. However, when subtracting binary digits from 1, we can

have either $1 - 0 = 1$ or $1 - 1 = 0$, which cau
1 to 0, respectively. Therefore, **the 1's comple**
changing 1's to 0's and 0's to 1's. The following

The 1's complement of 10

The 1's complement of 01

The $(r - 1)$'s complement of octal or hexadec
each digit from 7 or F (decimal 15), respectively.

Radix Complement

The r's complement of an n-digit number N in ba
as 0 for $N = 0$. Comparing with the $(r - 1)$'s com
is obtained by adding 1 to the $(r - 1)$'s complen
Thus, the 10's complement of decimal 2389 is 761
1 to the 9's complement value. The 2's complement o
and is obtained by adding 1 to the 1's-complemen
Since 10 is a number represented by a 1 follo
complement of N, can be formed also by leavi
subtracting the first nonzero least significant di
significant digits from 9. Thus,

the 10's complement of 0

and

the 10's complement of 2

The 10's complement of the first number is obtain
significant position and subtracting all other digi
second number is obtained by leaving the two le
ing 7 from 10, and subtracting the other three di
Similarly, the 2's complement can be formed
the first 1 unchanged and replacing 1's with 0's
nificant digits. For example,

the 2's complement of 110

and

the 2's complement of 011

The 2's complement of the first number is obtain
0's and the first 1 unchanged and then replacing 1
four most significant digits. The 2's complement
leaving the least significant 1 unchanged and com

have either $1 - 0 = 1$ or $1 - 1 = 0$, which causes the bit to change from 0 to 1 or from 1 to 0, respectively. Therefore, **the 1's complement of a binary number is formed by changing 1's to 0's and 0's to 1's.** The following are some numerical examples:

The 1's complement of 1011000 is 0100111.

The 1's complement of 0101101 is 1010010.

The $(r - 1)$'s complement of octal or hexadecimal numbers is obtained by subtracting each digit from 7 or F (decimal 15), respectively.

Radix Complement

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N = 0$. Comparing with the $(r - 1)$'s complement, we note that the r's complement is obtained by adding 1 to the $(r - 1)$'s complement, since $r^n - N = [(r^n - 1) - N] + 1$. Thus, the 10's complement of decimal 2389 is $7610 + 1 = 7611$ and is obtained by adding 1 to the 9's complement value. The 2's complement of binary 101100 is $010011 + 1 = 010100$ and is obtained by adding 1 to the 1's-complement value.

Since 10 is a number represented by a 1 followed by n 0's, $10^n - N$, which is the 10's complement of N, can be formed also by leaving all least significant 0's unchanged, subtracting the first nonzero least significant digit from 10, and subtracting all higher significant digits from 9. Thus,

the 10's complement of 012398 is 987602

and

the 10's complement of 246700 is 753300

The 10's complement of the first number is obtained by subtracting 8 from 10 in the least significant position and subtracting all other digits from 9. The 10's complement of the second number is obtained by leaving the two least significant 0's unchanged, subtracting 7 from 10, and subtracting the other three digits from 9.

Similarly, the 2's complement can be formed by leaving all least significant 0's and the first 1 unchanged and replacing 1's with 0's and 0's with 1's in all other higher significant digits. For example,

the 2's complement of 1101100 is 0010100

and

the 2's complement of 0110111 is 1001001

The 2's complement of the first number is obtained by leaving the two least significant 0's and the first 1 unchanged and then replacing 1's with 0's and 0's with 1's in the other four most significant digits. The 2's complement of the second number is obtained by leaving the least significant 1 unchanged and complementing all other digits.

EXAMPLE 1.8

Repeat Example 1.7, but this time using 1's complement.

(a) $X - Y = 1010100 - 1000011$

$$
\begin{aligned}
X &= 1010100\\
\text{1's complement of } Y &= +\ \underline{0111100}\\
\text{Sum} &= 10010000\\
\text{End-around carry} &= +\ \underline{1}\\
\textit{Answer: } X - Y &= 0010001
\end{aligned}
$$

(b) $Y - X = 1000011 - 1010100$

$$
\begin{aligned}
Y &= 1000011\\
\text{1's complement of } X &= +\ \underline{0101011}\\
\text{Sum} &= 1101110
\end{aligned}
$$

There is no end carry. Therefore, the answer is $Y - X = -$(1's complement of 1101110) $= -0010001$.

Note that the negative result is obtained by taking the 1's complement of the sum, since this is the type of complement used. The procedure with end-around carry is also applicable to subtracting unsigned decimal numbers with 9's complement.

1.6 SIGNED BINARY NUMBERS

Positive integers (including zero) can be represented as unsigned numbers. However, to represent negative integers, we need a notation for negative values. In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign. Because of hardware limitations, computers must represent everything with binary digits. It is customary to represent the sign with a bit placed in the leftmost position of the number. The convention is to make the sign bit 0 for positive and 1 for negative.

It is important to realize that both signed and unsigned binary numbers consist of a string of bits when represented in a computer. The user determines whether the number is signed or unsigned. If the binary number is signed, then the leftmost bit represents the sign and the rest of the bits represent the number. If the binary number is assumed to be unsigned, then the leftmost bit is the most significant bit of the number. For example, the string of bits 01001 can be considered as 9 (unsigned binary) or as +9 (signed binary) because the leftmost bit is 0. The string of bits 11001 represents the binary equivalent of 25 when considered as an unsigned number and the binary equivalent of −9 when considered as a signed number. This is because the 1 that is in the leftmost position designates a negative and the other four bits represent binary 9. Usually, there is no confusion in interpreting the bits if the type of representation for the number is known in advance.

The representation of the signed numbe
signed-magnitude convention. In this notatio
a symbol (+ or −) or a bit (0 or 1) indicating
numbers used in ordinary arithmetic. When
a computer, it is more convenient to use a
complement system, for representing negativ
ber is indicated by its complement. Where
number by changing its sign, the signed-com
its complement. Since positive numbers alw
tion, the complement will always start wit
signed-complement system can use either
complement is the most common.

As an example, consider the number 9, r
represented with a sign bit of 0 in the leftmc
lent of 9, which gives 00001001. Note that al
are inserted following the sign bit up to the
represent +9, there are three different ways

signed-magnitude represent

signed-1's-complement repr

signed-2's-complement repr

In signed-magnitude, −9 is obtained from +9
position from 0 to 1. In signed-1's-compleme
bits of +9, including the sign bit. The signe
obtained by taking the 2's complement of th

Table 1.3 lists all possible four-bit signed b
The equivalent decimal number is also show
bers in all three representations are identic
signed-2's-complement system has only one
tive. The other two systems have either a
encountered in ordinary arithmetic. Note
leftmost bit position; that is the way we dis
With four bits, we can represent 16 binary
1's-complement representations, there are
numbers, including two zeros. In the 2's-co
positive numbers, including one zero, and ei

The signed-magnitude system is used in
employed in computer arithmetic because o
magnitude. Therefore, the signed-complem
plement imposes some difficulties and is se
useful as a logical operation, since the cha
logical complement operation, as will be sh
signed binary arithmetic that follows deals e

ıis time using 1's complement.

- 1000011

$$
\begin{aligned}
X &= 1010100\\
\text{complement of } Y &= +\ \underline{0111100}\\
\text{Sum} &= 10010000\\
\text{End-around carry} &= +\ \underline{1}\\
\textit{Answer: } X - Y &= 0010001
\end{aligned}
$$

- 1010100

$$
\begin{aligned}
Y &= 1000011\\
\text{s complement of } X &= +\ \underline{0101011}\\
\text{Sum} &= 1101110
\end{aligned}
$$

efore, the answer is $Y - X = -$(1's complement of 1101110) =

ılt is obtained by taking the 1's complement of the sum, since ıent used. The procedure with end-around carry is also appliıed decimal numbers with 9's complement.

NUMBERS

g zero) can be represented as unsigned numbers. However, to s, we need a notation for negative values. In ordinary arithis indicated by a minus sign and a positive number by a plus limitations, computers must represent everything with binary present the sign with a bit placed in the leftmost position of on is to make the sign bit 0 for positive and 1 for negative.

e that both signed and unsigned binary numbers consist of a ıted in a computer. The user determines whether the number binary number is signed, then the leftmost bit represents the s represent the number. If the binary number is assumed to ıost bit is the most significant bit of the number. For example, be considered as 9 (unsigned binary) or as +9 (signed binary)). The string of bits 11001 represents the binary equivalent of ınsigned number and the binary equivalent of −9 when con- This is because the 1 that is in the leftmost position designates our bits represent binary 9. Usually, there is no confusion in type of representation for the number is known in advance.

The representation of the signed numbers in the last example is referred to as the *signed-magnitude* convention. In this notation, the number consists of a magnitude and a symbol (+ or −) or a bit (0 or 1) indicating the sign. This is the representation of signed numbers used in ordinary arithmetic. When arithmetic operations are implemented in a computer, it is more convenient to use a different system, referred to as the *signed-complement* system, for representing negative numbers. In this system, a negative number is indicated by its complement. Whereas the signed-magnitude system negates a number by changing its sign, the signed-complement system negates a number by taking its complement. Since positive numbers always start with 0 (plus) in the leftmost position, the complement will always start with a 1, indicating a negative number. The signed-complement system can use either the 1's or the 2's complement, but the 2's complement is the most common.

As an example, consider the number 9, represented in binary with eight bits. +9 is represented with a sign bit of 0 in the leftmost position, followed by the binary equivalent of 9, which gives 00001001. Note that all eight bits must have a value; therefore, 0's are inserted following the sign bit up to the first 1. Although there is only one way to represent +9, there are three different ways to represent −9 with eight bits:

signed-magnitude representation:	10001001
signed-1's-complement representation:	11110110
signed-2's-complement representation:	11110111

In signed-magnitude, −9 is obtained from +9 by changing only the sign bit in the leftmost position from 0 to 1. In signed-1's-complement, −9 is obtained by complementing all the bits of +9, including the sign bit. The signed-2's-complement representation of −9 is obtained by taking the 2's complement of the positive number, including the sign bit.

Table 1.3 lists all possible four-bit signed binary numbers in the three representations. The equivalent decimal number is also shown for reference. Note that the positive numbers in all three representations are identical and have 0 in the leftmost position. The signed-2's-complement system has only one representation for 0, which is always positive. The other two systems have either a positive 0 or a negative 0, something not encountered in ordinary arithmetic. Note that all negative numbers have a 1 in the leftmost bit position; that is the way we distinguish them from the positive numbers. With four bits, we can represent 16 binary numbers. In the signed-magnitude and the 1's-complement representations, there are eight positive numbers and eight negative numbers, including two zeros. In the 2's-complement representation, there are eight positive numbers, including one zero, and eight negative numbers.

The signed-magnitude system is used in ordinary arithmetic, but is awkward when employed in computer arithmetic because of the separate handling of the sign and the magnitude. Therefore, the signed-complement system is normally used. The 1's complement imposes some difficulties and is seldom used for arithmetic operations. It is useful as a logical operation, since the change of 1 to 0 or 0 to 1 is equivalent to a logical complement operation, as will be shown in the next chapter. The discussion of signed binary arithmetic that follows deals exclusively with the signed-2's-complement

Table 1.3
Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

representation of negative numbers. The same procedures can be applied to the signed-1's-complement system by including the end-around carry as is done with unsigned numbers.

Arithmetic Addition

The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic. If the signs are the same, we add the two magnitudes and give the sum the common sign. If the signs are different, we subtract the smaller magnitude from the larger and give the difference the sign of the larger magnitude. For example, $(+25) + (-37) = -(37 - 25) = -12$ is done by subtracting the smaller magnitude, 25, from the larger magnitude, 37, and appending the sign of 37 to the result. This is a process that requires a comparison of the signs and magnitudes and then performing either addition or subtraction. The same procedure applies to binary numbers in signed-magnitude representation. In contrast, the rule for adding numbers in the signed-complement system does not require a comparison or subtraction, but only addition. The procedure is very simple and can be stated as follows for binary numbers:

The addition of two signed binary numbers with negative numbers represented in signed-2's-complement form is obtained from the addition of the two numbers, including their sign bits. A carry out of the sign-bit position is discarded.

Numerical examples for addition follow:

$$\begin{array}{rl} +\ 6 & 00000110 \\ \underline{+13} & \underline{00001101} \\ +19 & 00010011 \end{array}$$

$$\begin{array}{rl} +\ 6 & 00000110 \\ \underline{-13} & \underline{11110011} \\ -\ 7 & 11111001 \end{array}$$

Note that negative numbers must be initially i
obtained after the addition is negative, it is in
represented as 11111001, which is the 2s com
In each of the four cases, the operation
included. Any carry out of the sign-bit positi
automatically in 2's-complement form.
In order to obtain a correct answer, we m
number of bits to accommodate the sum. If we
occupies $n + 1$ bits, we say that an overflow oc
paper and pencil, an overflow is not a problen
of the page. We just add another 0 to a positive
in the most significant position to extend the n
addition. Overflow is a problem in computer
number is finite, and a result that exceeds the f
The complement form of representing neg
to the signed-magnitude system. To determine t
complement, it is necessary to convert the nu
more familiar form. For example, the signed bi
the leftmost bit is 1. Its 2's complement is 00
+7. We therefore recognize the original nega

Arithmetic Subtraction

Subtraction of two signed binary numbers whe
form is simple and can be stated as follows:

Take the 2's complement of the subtrahend
minuend (including the sign bit). A carry o

This procedure is adopted because a subtract
tion operation if the sign of the subtrahen
following relationship:

$$(\pm A) - (+B) = ($$
$$(\pm A) - (-B) = ($$

But changing a positive number to a negativ
complement of the positive number. The reve

Numerical examples for addition follow:

$+6$	00000110		-6	11111010
$+13$	00001101		$+13$	00001101
$+19$	00010011		$+7$	00000111
$+6$	00000110		-6	11111010
-13	11110011		-13	11110011
-7	11111001		-19	11101101

Note that negative numbers must be initially in 2's-complement form and that if the sum obtained after the addition is negative, it is in 2's-complement form. For example, -7 is represented as 11111001, which is the 2s complement of $+7$.

In each of the four cases, the operation performed is addition with the sign bit included. Any carry out of the sign-bit position is discarded, and negative results are automatically in 2's-complement form.

In order to obtain a correct answer, we must ensure that the result has a sufficient number of bits to accommodate the sum. If we start with two n-bit numbers and the sum occupies $n + 1$ bits, we say that an overflow occurs. When one performs the addition with paper and pencil, an overflow is not a problem, because we are not limited by the width of the page. We just add another 0 to a positive number or another 1 to a negative number in the most significant position to extend the number to $n + 1$ bits and then perform the addition. Overflow is a problem in computers because the number of bits that hold a number is finite, and a result that exceeds the finite value by 1 cannot be accommodated.

The complement form of representing negative numbers is unfamiliar to those used to the signed-magnitude system. To determine the value of a negative number in signed-2's complement, it is necessary to convert the number to a positive number to place it in a more familiar form. For example, the signed binary number 11111001 is negative because the leftmost bit is 1. Its 2's complement is 00000111, which is the binary equivalent of $+7$. We therefore recognize the original negative number to be equal to -7.

Arithmetic Subtraction

Subtraction of two signed binary numbers when negative numbers are in 2's-complement form is simple and can be stated as follows:

> Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign bit). A carry out of the sign-bit position is discarded.

This procedure is adopted because a subtraction operation can be changed to an addition operation if the sign of the subtrahend is changed, as is demonstrated by the following relationship:

$$(\pm A) - (+B) = (\pm A) + (-B);$$
$$(\pm A) - (-B) = (\pm A) + (+B).$$

But changing a positive number to a negative number is easily done by taking the 2's complement of the positive number. The reverse is also true, because the complement of a negative number in complement form produces the equivalent positive number. To see this, consider the subtraction $(-6) - (-13) = +7$. In binary with eight bits, this operation is written as $(11111010 - 11110011)$. The subtraction is changed to addition by taking the 2's complement of the subtrahend (−13), giving (+13). In binary, this is $11111010 + 00001101 = 100000111$. Removing the end carry, we obtain the correct answer: 00000111 (+7).

It is worth noting that binary numbers in the signed-complement system are added and subtracted by the same basic addition and subtraction rules as unsigned numbers. Therefore, **computers need only one common hardware circuit to handle both types of arithmetic.** This consideration has resulted in the signed-complement system being used in virtually all arithmetic units of computer systems. The user or programmer must interpret the results of such addition or subtraction differently, depending on whether it is assumed that the numbers are signed or unsigned.

1.7 BINARY CODES

Digital systems use signals that have two distinct values and circuit elements that have two stable states. There is a direct analogy among binary signals, binary circuit elements, and binary digits. A binary number of n digits, for example, may be represented by n binary circuit elements, each having an output signal equivalent to 0 or 1. Digital systems represent and manipulate not only binary numbers, but also many other discrete elements of information. Any discrete element of information that is distinct among a group of quantities can be represented with a binary code (i.e., a pattern of 0's and 1's). The codes must be in binary because, in today's technology, only circuits that represent and manipulate patterns of 0's and 1's can be manufactured economically for use in computers. However, it must be realized that binary codes merely change the symbols, not the meaning of the elements of information that they represent. If we inspect the bits of a computer at random, we will find that most of the time they represent some type of coded information rather than binary numbers.

An n-bit binary code is a group of n bits that assumes up to 2^n distinct combinations of 1's and 0's, with each combination representing one element of the set that is being coded. A set of four elements can be coded with two bits, with each element assigned one of the following bit combinations: 00, 01, 10, 11. A set of eight elements requires a three-bit code and a set of 16 elements requires a four-bit code. The bit combination of an n-bit code is determined from the count in binary from 0 to $2^n - 1$. Each element must be assigned a unique binary bit combination, and no two elements can have the same value; otherwise, the code assignment will be ambiguous.

Although the *minimum* number of bits required to code 2^n distinct quantities is n, there is no *maximum* number of bits that may be used for a binary code. For example, the 10 decimal digits can be coded with 10 bits, and each decimal digit can be assigned a bit combination of nine 0's and a 1. In this particular binary code, the digit 6 is assigned the bit combination 0001000000.

mplement form produces the equivalent positive number. To traction $(-6) - (-13) = +7$. In binary with eight bits, this 111010 − 11110011). The subtraction is changed to addition nent of the subtrahend (−13), giving (+13). In binary, this is 100000111. Removing the end carry, we obtain the correct

binary numbers in the signed-complement system are added ie basic addition and subtraction rules as unsigned numbers. **d only one common hardware circuit to handle both types of** ion has resulted in the signed-complement system being used units of computer systems. The user or programmer must ch addition or subtraction differently, depending on whether ibers are signed or unsigned.

---

als that have two distinct values and circuit elements that iere is a direct analogy among binary signals, binary circuit ts. A binary number of n digits, for example, may be repre- elements, each having an output signal equivalent to 0 or 1. t and manipulate not only binary numbers, but also many of information. Any discrete element of information that is of quantities can be represented with a binary code (i.e., a he codes must be in binary because, in today's technology, nt and manipulate patterns of 0's and 1's can be manufac- ise in computers. However, it must be realized that binary symbols, not the meaning of the elements of information inspect the bits of a computer at random, we will find that present some type of coded information rather than binary

a group of n bits that assumes up to 2^n distinct combinations ombination representing one element of the set that is being ients can be coded with two bits, with each element assigned ombinations: 00, 01, 10, 11. A set of eight elements requires a f 16 elements requires a four-bit code. The bit combination of ed from the count in binary from 0 to $2^n - 1$. Each element e binary bit combination, and no two elements can have the e code assignment will be ambiguous.

n number of bits required to code 2^n distinct quantities is n, iber of bits that may be used for a binary code. For example, be coded with 10 bits, and each decimal digit can be assigned 0's and a 1. In this particular binary code, the digit 6 is assigned 000000.

Binary-Coded Decimal Code

Although the binary number system is the most natural system for a computer because it is readily represented in today's electronic technology, most people are more accustomed to the decimal system. One way to resolve this difference is to convert decimal numbers to binary, perform all arithmetic calculations in binary, and then convert the binary results back to decimal. This method requires that we store decimal numbers in the computer so that they can be converted to binary. Since the computer can accept only binary values, we must represent the decimal digits by means of a code that contains 1's and 0's. It is also possible to perform the arithmetic operations directly on decimal numbers when they are stored in the computer in coded form.

A binary code will have some unassigned bit combinations if the number of elements in the set is not a multiple power of 2. The 10 decimal digits form such a set. A binary code that distinguishes among 10 elements must contain at least four bits, but 6 out of the 16 possible combinations remain unassigned. Different binary codes can be obtained by arranging four bits into 10 distinct combinations. The code most commonly used for the decimal digits is the straight binary assignment listed in Table 1.4. This scheme is called *binary-coded decimal* and is commonly referred to as BCD. Other decimal codes are possible and a few of them are presented later in this section.

Table 1.4 gives the four-bit code for one decimal digit. A number with k decimal digits will require $4k$ bits in BCD. Decimal 396 is represented in BCD with 12 bits as 0011 1001 0110, with **each group of 4 bits representing one decimal digit.** A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 and 9. A BCD number greater than 10 looks different from its equivalent binary number, even though both contain 1's and 0's. Moreover, **the binary combinations 1010 through 1111 are not used and have no meaning in BCD.** Consider decimal 185 and its corresponding value in BCD and binary:

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

Table 1.4
Binary-Coded Decimal (BCD)

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

The BCD value has 12 bits to encode the characters of the decimal value, but the equivalent binary number needs only 8 bits. It is obvious that the representation of a BCD number needs more bits than its equivalent binary value. However, there is an advantage in the use of decimal numbers, because computer input and output data are generated by people who use the decimal system.

It is important to realize that BCD numbers are decimal numbers and not binary numbers, although they use bits in their representation. The only difference between a decimal number and BCD is that decimals are written with the symbols $0, 1, 2, \ldots, 9$ and BCD numbers use the binary code $0000, 0001, 0010, \ldots, 1001$. The decimal value is exactly the same. Decimal 10 is represented in BCD with eight bits as 0001 0000 and decimal 15 as 0001 0101. The corresponding binary values are 1010 and 1111 and have only four bits.

BCD Addition

Consider the addition of two decimal digits in BCD, together with a possible carry from a previous less significant pair of digits. Since each digit does not exceed 9, the sum cannot be greater than $9 + 9 + 1 = 19$, with the 1 being a previous carry. Suppose we add the BCD digits as if they were binary numbers. Then the binary sum will produce a result in the range from 0 to 19. In binary, this range will be from 0000 to 10011, but in BCD, it is from 0000 to 1 1001, with the first (i.e., leftmost) 1 being a carry and the next four bits being the BCD sum. When the binary sum is equal to or less than 1001 (without a carry), the corresponding BCD digit is correct. However, when the binary sum is greater than or equal to 1010, the result is an invalid BCD digit. The addition of $6 = (0110)_2$ to the binary sum converts it to the correct digit and also produces a carry as required. This is because a carry in the most significant bit position of the binary sum and a decimal carry differ by $16 - 10 = 6$. Consider the following three BCD additions:

$$
\begin{array}{rr@{\qquad}rr@{\qquad}rr}
4 & 0100 & 4 & 0100 & 8 & 1000 \\
\underline{+5} & \underline{+0101} & \underline{+8} & \underline{+1000} & \underline{+9} & \underline{1001} \\
9 & 1001 & 12 & 1100 & 17 & 10001 \\
 & & & \underline{+0110} & & \underline{+0110} \\
 & & & 10010 & & 10111
\end{array}
$$

In each case, the two BCD digits are added as if they were two binary numbers. If the binary sum is greater than or equal to 1010, we add 0110 to obtain the correct BCD sum and a carry. In the first example, the sum is equal to 9 and is the correct BCD sum. In the second example, the binary sum produces an invalid BCD digit. The addition of 0110 produces the correct BCD sum, 0010 (i.e., the number 2), and a carry. In the third example, the binary sum produces a carry. This condition occurs when the sum is greater than or equal to 16. Although the other four bits are less than 1001, the binary sum requires a correction because of the carry. Adding 0110, we obtain the required BCD sum 0111 (i.e., the number 7) and a BCD carry.

The addition of two n-digit unsigned BCD numbers follows the same procedure. Consider the addition of $184 + 576 = 760$ in BCD:

BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6		0110	0110	
BCD sum	0111	0110	0000	760

The first, least significant pair of BCD digits produces a BCD digit sum of 0000 and a carry for the next pair of digits. The second pair of BCD digits plus a previous carry produces a digit sum of 0110 and a carry for the next pair of digits. The third pair of digits plus a carry produces a binary sum of 0111 and does not require a correction.

Decimal Arithmetic

The representation of signed decimal numbers in BCD is similar to the representation of signed numbers in binary. We can use either the familiar signed-magnitude system or the signed-complement system. The sign of a decimal number is usually represented with four bits to conform to the four-bit code of the decimal digits. It is customary to designate a plus with four 0's and a minus with the BCD equivalent of 9, which is 1001.

The signed-magnitude system is seldom used in computers. The signed-complement system can be either the 9's or the 10's complement, but the 10's complement is the one most often used. To obtain the 10's complement of a BCD number, we first take the 9's complement and then add 1 to the least significant digit. The 9's complement is calculated from the subtraction of each digit from 9.

The procedures developed for the signed-2's-complement system in the previous section also apply to the signed-10's-complement system for decimal numbers. Addition is done by summing all digits, including the sign digit, and discarding the end carry. This operation assumes that all negative numbers are in 10's-complement form. Consider the addition $(+375) + (-240) = +135$, done in the signed-complement system:

0	375
+9	760
0	135

The 9 in the leftmost position of the second number represents a minus, and 9760 is the 10's complement of 0240. The two numbers are added and the end carry is discarded to obtain +135. Of course, the decimal numbers inside the computer, including the sign digits, must be in BCD. The addition is done with BCD digits as described previously.

The subtraction of decimal numbers, either unsigned or in the signed-10's-complement system, is the same as in the binary case: Take the 10's complement of the subtrahend and add it to the minuend. Many computers have special hardware to perform arithmetic calculations directly with decimal numbers in BCD. The user of the computer can specify programmed instructions to perform the arithmetic operation with decimal numbers directly, without having to convert them to binary.

Other Decimal Codes

Binary codes for decimal digits require a minimum of four bits per digit. Many different codes can be formulated by arranging four bits into 10 distinct combinations. BCD and three other representative codes are shown in Table 1.5. Each code uses only 10 out of a possible 16 bit combinations that can be arranged with four bits. The other six unused combinations have no meaning and should be avoided.

BCD and the 2421 code are examples of weighted codes. In a weighted code, each bit position is assigned a weighting factor in such a way that each digit can be evaluated by adding the weights of all the 1's in the coded combination. The BCD code has weights of 8, 4, 2, and 1, which correspond to the power-of-two values of each bit. The bit assignment 0110, for example, is interpreted by the weights to represent decimal 6 because $8 \times 0 + 4 \times 1 + 2 \times 1 + 1 \times 0 = 6$. The bit combination 1101, when weighted by the respective digits 2421, gives the decimal equivalent of $2 \times 1 + 4 \times 1 + 2 \times 0 + 1 \times 1 = 7$. Note that some digits can be coded in two possible ways in the 2421 code. For instance, decimal 4 can be assigned to bit combination 0100 or 1010, since both combinations add up to a total weight of 4.

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
Unused bit combi-nations	1010	0101	0000	0001
	1011	0110	0001	0010
	1100	0111	0010	0011
	1101	1000	1101	1100
	1110	1001	1110	1101
	1111	1010	1111	1110

BCD adders add BCD values directly, digit by d
to binary. However, it is necessary to add 6 to th
adders require significantly more hardware and r
conventional binary adders [5].

The 2421 and the excess-3 codes are examples
codes have the property that the 9's complemen
directly by changing 1's to 0's and 0's to 1's (i.e., by
tern). For example, decimal 395 is represented in
The 9's complement of 604 is represented as 1001
by complementing each bit of the code (as with the
The excess-3 code has been used in some ol
complementing property. **Excess-3 is an unweigh**
bination is obtained from the corresponding bin
code is not self-complementing.

The 8, 4, −2, −1 code is an example of assignin
to a decimal code. In this case, the bit combination
is calculated from $8 \times 0 + 4 \times 1 + (-2) \times 1 +$

Gray Code

The output data of many physical systems are q
data must be converted into digital form before
Continuous or analog information is converted in
log-to-digital converter. It is sometimes conveni
Table 1.6 to represent digital data that have bee
advantage of the Gray code over the straight bi
one bit in the code group changes in going from o
in going from 7 to 8, the Gray code changes fro
changes, from 0 to 1; the other three bits remain
numbers the change from 7 to 8 will be from 011
to change values.

The Gray code is used in applications in which th
generated by the hardware may produce an error or
one number to the next. If binary numbers are used
1000 may produce an intermediate erroneous num
bit takes longer to change than do the values of the
ous consequences for the machine using the inform
problem, since only one bit changes its value during

A typical application of the Gray code is the rep
tinuous change in the angular position of a shaft. T
and each segment is assigned a number. If adjacen
with the Gray-code sequence, ambiguity is elimin
and the value encoded by the sensor.

BCD adders add BCD values directly, digit by digit, without converting the numbers to binary. However, it is necessary to add 6 to the result if it is greater than 9. BCD adders require significantly more hardware and no longer have a speed advantage of conventional binary adders [5].

The 2421 and the excess-3 codes are examples of self-complementing codes. Such codes have the property that the 9's complement of a decimal number is obtained directly by changing 1's to 0's and 0's to 1's (i.e., by complementing each bit in the pattern). For example, decimal 395 is represented in the excess-3 code as 0110 1100 1000. The 9's complement of 604 is represented as 1001 0011 0111, which is obtained simply by complementing each bit of the code (as with the 1's complement of binary numbers).

The excess-3 code has been used in some older computers because of its self-complementing property. **Excess-3 is an unweighted code in which each coded combination is obtained from the corresponding binary value plus 3.** Note that the BCD code is not self-complementing.

The 8, 4, −2, −1 code is an example of assigning both positive and negative weights to a decimal code. In this case, the bit combination 0110 is interpreted as decimal 2 and is calculated from $8 \times 0 + 4 \times 1 + (-2) \times 1 + (-1) \times 0 = 2$.

Gray Code

The output data of many physical systems are quantities that are continuous. These data must be converted into digital form before they are applied to a digital system. Continuous or analog information is converted into digital form by means of an analog-to-digital converter. It is sometimes convenient to use the Gray code shown in Table 1.6 to represent digital data that have been converted from analog data. The advantage of the Gray code over the straight binary number sequence is that only one bit in the code group changes in going from one number to the next. For example, in going from 7 to 8, the Gray code changes from 0100 to 1100. Only the first bit changes, from 0 to 1; the other three bits remain the same. By contrast, with binary numbers the change from 7 to 8 will be from 0111 to 1000, which causes all four bits to change values.

The Gray code is used in applications in which the normal sequence of binary numbers generated by the hardware may produce an error or ambiguity during the transition from one number to the next. If binary numbers are used, a change, for example, from 0111 to 1000 may produce an intermediate erroneous number 1001 if the value of the rightmost bit takes longer to change than do the values of the other three bits. This could have serious consequences for the machine using the information. The Gray code eliminates this problem, since only one bit changes its value during any transition between two numbers.

A typical application of the Gray code is the representation of analog data by a continuous change in the angular position of a shaft. The shaft is partitioned into segments, and each segment is assigned a number. If adjacent segments are made to correspond with the Gray-code sequence, ambiguity is eliminated between the angle of the shaft and the value encoded by the sensor.

Table 1.6
Gray Code

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

ASCII Character Code

Many applications of digital computers require the handling not only of numbers, but also of other characters or symbols, such as the letters of the alphabet. For instance, consider a high-tech company with thousands of employees. To represent the names and other pertinent information, it is necessary to formulate a binary code for the letters of the alphabet. In addition, the same binary code must represent numerals and special characters (such as $). An alphanumeric character set is a set of elements that includes the 10 decimal digits, the 26 letters of the alphabet, and a number of special characters. Such a set contains between 36 and 64 elements if only capital letters are included, or between 64 and 128 elements if both uppercase and lowercase letters are included. In the first case, we need a binary code of six bits, and in the second, we need a binary code of seven bits.

The standard binary code for the alphanumeric characters is the American Standard Code for Information Interchange (ASCII), which uses seven bits to code 128 characters, as shown in Table 1.7. The seven bits of the code are designated by b_1 through b_7, with b_7 the most significant bit. The letter A, for example, is represented in ASCII as 1000001 (column 100, row 0001). The ASCII code also contains 94 graphic characters that can be printed and 34 nonprinting characters used for various control functions. The graphic characters consist of the 26 uppercase letters (A through Z), the 26 lowercase letters (a through z), the 10 numerals (0 through 9), and 32 special printable characters, such as %, *, and $.

Table 1.7
American Standard Code for Information Inte

$b_4b_3b_2b_1$	000	001	010	01
0000	NUL	DLE	SP	0
0001	SOH	DC1	!	1
0010	STX	DC2	"	2
0011	ETX	DC3	#	3
0100	EOT	DC4	$	4
0101	ENQ	NAK	%	5
0110	ACK	SYN	&	6
0111	BEL	ETB	'	7
1000	BS	CAN	(	8
1001	HT	EM	)	9
1010	LF	SUB	*	:
1011	VT	ESC	+	;
1100	FF	FS	,	<
1101	CR	GS	–	=
1110	SO	RS	.	>
1111	SI	US	/	?

Control Ch

NUL	Null	DLE
SOH	Start of heading	DC1
STX	Start of text	DC2
ETX	End of text	DC3
EOT	End of transmission	DC4
ENQ	Enquiry	NA
ACK	Acknowledge	SYN
BEL	Bell	ET
BS	Backspace	CA
HT	Horizontal tab	EM
LF	Line feed	SU
VT	Vertical tab	ES
FF	Form feed	FS
CR	Carriage return	GS
SO	Shift out	RS
SI	Shift in	US
SP	Space	DE

The 34 control characters are designated in th
are listed again below the table with their funct
for routing data and arranging the printed text i
of control characters: format effectors, informa

Table 1.6
Gray Code

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

ital computers require the handling not only of numbers, but or symbols, such as the letters of the alphabet. For instance, ıpany with thousands of employees. To represent the names mation, it is necessary to formulate a binary code for the let-ddition, the same binary code must represent numerals and ıs $). An alphanumeric character set is a set of elements that ligits, the 26 letters of the alphabet, and a number of special ıtains between 36 and 64 elements if only capital letters are ınd 128 elements if both uppercase and lowercase letters are , we need a binary code of six bits, and in the second, we need ts.

ıde for the alphanumeric characters is the American Standard erchange (ASCII), which uses seven bits to code 128 charac-. The seven bits of the code are designated by b_1 through b_7, ınt bit. The letter *A*, for example, is represented in ASCII as w 0001). The ASCII code also contains 94 graphic characters 34 nonprinting characters used for various control functions. ınsist of the 26 uppercase letters (A through Z), the 26 lower- the 10 numerals (0 through 9), and 32 special printable char-s.

Table 1.7
American Standard Code for Information Interchange (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Control Characters

NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End-of-transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

The 34 control characters are designated in the ASCII table with abbreviated names. They are listed again below the table with their functional names. The control characters are used for routing data and arranging the printed text into a prescribed format. There are three types of control characters: format effectors, information separators, and communication-control characters. Format effectors are characters that control the layout of printing. They include the familiar word processor and typewriter controls such as backspace (BS), horizontal tabulation (HT), and carriage return (CR). Information separators are used to separate the data into divisions such as paragraphs and pages. They include characters such as record separator (RS) and file separator (FS). The communication-control characters are useful during the transmission of text between remote devices so that it can be distinguished from other messages using the same communication channel before it and after it. Examples of communication-control characters are STX (start of text) and ETX (end of text), which are used to frame a text message transmitted through a communication channel.

ASCII is a seven-bit code, but most computers manipulate an eight-bit quantity as a single unit called a *byte*. Therefore, ASCII characters most often are stored one per byte. The extra bit is sometimes used for other purposes, depending on the application. For example, some printers recognize eight-bit ASCII characters with the most significant bit set to 0. An additional 128 eight-bit characters with the most significant bit set to 1 are used for other symbols, such as the Greek alphabet or italic type font.

Error-Detecting Code

To detect errors in data communication and processing, an eighth bit is sometimes added to the ASCII character to indicate its parity. A *parity bit* is an extra bit included with a message to make the total number of 1's either even or odd. Consider the following two characters and their even and odd parity:

	With even parity	**With odd parity**
ASCII A = 1000001	01000001	11000001
ASCII T = 1010100	11010100	01010100

In each case, we insert an extra bit in the leftmost position of the code to produce an even number of 1's in the character for even parity or an odd number of 1's in the character for odd parity. In general, one or the other parity is adopted, with even parity being more common.

The parity bit is helpful in detecting errors during the transmission of information from one location to another. This function is handled by generating an even parity bit at the sending end for each character. The eight-bit characters that include parity bits are transmitted to their destination. The parity of each character is then checked at the receiving end. If the parity of the received character is not even, then at least one bit has changed value during the transmission. This method detects one, three, or any odd combination of errors in each character that is transmitted. An even combination of errors, however, goes undetected, and additional error detection codes may be needed to take care of that possibility.

What is done after an error is detected depends on the particular application. One possibility is to request retransmission of the message on the assumption that the error was random and will not occur again. Thus, if the receiver detects a parity error, it sends

back the ASCII NAK (negative acknowledge
parity eight bits 10010101. If no error is de
(acknowledge) control character, namely, 000
NAK by transmitting the message again unti
number of attempts, the transmission is still ir
ator to check for malfunctions in the transmis

1.8 BINARY STORAGE AND REG

The binary information in a digital compute
medium for storing individual bits. A *binary*
states and is capable of storing one bit (0 or
receives excitation signals that set it to one c
a physical quantity that distinguishes betwee
in a cell is 1 when the cell is in one stable stat
state.

Registers

A *register* is a group of binary cells. A register
of information that contains *n* bits. The state o
each bit designating the state of one cell in t
function of the interpretation given to the info
a 16-bit register with the following binary con

1100001111

A register with 16 cells can be in one of 2^{16} p
tent of the register represents a binary integ
number from 0 to $2^{16} - 1$. For the particular e
is the binary equivalent of the decimal numb
register stores alphanumeric characters of a
register is any two meaningful characters. For
in the eighth most significant bit position, the
leftmost eight bits) and I (the rightmost eight
tent of the register to be four decimal digits
content of the register is a four-digit decimal
holds the decimal number 9,096. The conte
because the bit combination 1100 is not assig
ple, it is clear that a register can store discrete
bit configuration may be interpreted differe
on the application.

rs are characters that control the layout of printing. They include r and typewriter controls such as backspace (BS), horizontal tabu- eturn (CR). Information separators are used to separate the data graphs and pages. They include characters such as record separator FS). The communication-control characters are useful during etween remote devices so that it can be distinguished from other communication channel before it and after it. Examples of aracters are STX (start of text) and ETX (end of text), which are ge transmitted through a communication channel.

code, but most computers manipulate an eight-bit quantity byte. Therefore, ASCII characters most often are stored one sometimes used for other purposes, depending on the appli- ne printers recognize eight-bit ASCII characters with the to 0. An additional 128 eight-bit characters with the most used for other symbols, such as the Greek alphabet or italic

mmunication and processing, an eighth bit is sometimes added indicate its parity. A *parity bit* is an extra bit included with a l number of 1's either even or odd. Consider the following two and odd parity:

	With even parity	With odd parity
1000001	01000001	11000001
1010100	11010100	01010100

extra bit in the leftmost position of the code to produce an character for even parity or an odd number of 1's in the char- eral, one or the other parity is adopted, with even parity being

l in detecting errors during the transmission of information her. This function is handled by generating an even parity bit ch character. The eight-bit characters that include parity bits estination. The parity of each character is then checked at the of the received character is not even, then at least one bit has transmission. This method detects one, three, or any odd com- character that is transmitted. An even combination of errors, l, and additional error detection codes may be needed to take

error is detected depends on the particular application. One transmission of the message on the assumption that the error ccur again. Thus, if the receiver detects a parity error, it sends back the ASCII NAK (negative acknowledge) control character consisting of an even-parity eight bits 10010101. If no error is detected, the receiver sends back an ACK (acknowledge) control character, namely, 00000110. The sending end will respond to an NAK by transmitting the message again until the correct parity is received. If, after a number of attempts, the transmission is still in error, a message can be sent to the operator to check for malfunctions in the transmission path.

1.8 BINARY STORAGE AND REGISTERS

The binary information in a digital computer must have a physical existence in some medium for storing individual bits. A *binary cell* is a device that possesses two stable states and is capable of storing one bit (0 or 1) of information. The input to the cell receives excitation signals that set it to one of the two states. The output of the cell is a physical quantity that distinguishes between the two states. The information stored in a cell is 1 when the cell is in one stable state and 0 when the cell is in the other stable state.

Registers

A *register* is a group of binary cells. A register with n cells can store any discrete quantity of information that contains n bits. The state of a register is an n-tuple of 1's and 0's, with each bit designating the state of one cell in the register. The content of a register is a function of the interpretation given to the information stored in it. Consider, for example, a 16-bit register with the following binary content:

$$1100001111001001$$

A register with 16 cells can be in one of 2^{16} possible states. If one assumes that the content of the register represents a binary integer, then the register can store any binary number from 0 to $2^{16} - 1$. For the particular example shown, the content of the register is the binary equivalent of the decimal number 50,121. If one assumes instead that the register stores alphanumeric characters of an eight-bit code, then the content of the register is any two meaningful characters. For the ASCII code with an even parity placed in the eighth most significant bit position, the register contains the two characters C (the leftmost eight bits) and I (the rightmost eight bits). If, however, one interprets the content of the register to be four decimal digits represented by a four-bit code, then the content of the register is a four-digit decimal number. In the excess-3 code, the register holds the decimal number 9,096. The content of the register is meaningless in BCD, because the bit combination 1100 is not assigned to any decimal digit. From this example, it is clear that a register can store discrete elements of information and that the same bit configuration may be interpreted differently for different types of data depending on the application.