2

Representation of Numbers and Characters

Overview

You saw in Chapter 1 that computer circuits are capable of processing only binary information. In this chapter, we show how numbers can be expressed in binary; this is called the **binary number system**. We also introduce a very compact way of representing binary information called the **hexadecimal number system.**

Conversions between binary, decimal, and hexadecimal numbers are covered in section 2.2. Section 2.3 treats addition and subtraction in these number systems.

Section 2.4 shows how negative numbers are represented and what effects the fixed physical size of a byte or word has on number representation.

We conclude the chapter by exploring how characters are encoded and used by the computer.

2.1 Number Systems

Before we look at how numbers are represented in binary, it is instructive to look at the familiar decimal system. It is an example of a *positional number system;* that is, each digit in the number is associated with a power of 10, according to its position in the number. For example, the decimal number 3932 represents 3 thousands, 9 hundreds, 3 tens, and 2 ones. In other words,

$$3,932 = 3 \times 10^3 + 9 \times 10^2 + 3 \times 10^1 + 2 \times 10^0$$

In a positional system, some number b is selected as the base and symbols are assigned to numbers between 0 and b – 1. For example, in the decimal system there are ten basic symbols (digits): 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The base ten is represented as 10.

Binary Number System

In the binary number system, the base is two and there are only two digits, 0 and 1. For example, the binary string 11010 represents the number

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 26$$

The base two is represented in binary as 10.

Hexadecimal Number System

Numbers written in binary tend to be long and difficult to express. For example, 16 bits are needed to represent the contents of a memory word in an 8086-based computer. But decimal numbers are difficult to convert into binary. When we write assembly language programs we tend to use both binary, decimal, and a third number system called *hexadecimal,* or *hex* for short. The advantage of using hex numbers is that the conversion between binary and hex is easy.

The hexadecimal (hex) system is a base sixteen system. The hex digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The hex letters A through F denote numbers ten to fifteen, respectively. After F comes the base sixteen, represented in hex by 10.

Because sixteen is 2 to the power of 4, each hex digit corresponds to a unique four-bit number, as shown in Table 2.1. This means that the contents of a byte—eight bits—may be expressed neatly as two hex digits, which makes hex numbers useful with byte-oriented computers.

Table 2.2 shows the relations among binary, decimal, and hexadecimal numbers. It is a good idea to take a few minutes and memorize the first

Table 2.1 Hex Digits and Binary Equivalent

Hex Digits	*Binary*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Table 2.2 Decimal, Binary, and Hexadecimal Numbers

Decimal	*Binary*	*Hexadecimal*
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	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
16	10000	10
17	10001	11
18	10010	12
19	10011	13
20	10100	14
21	10101	15
22	10110	16
23	10111	17
24	11000	18
25	11001	19
26	11010	1A
27	11011	1B
28	11100	1C
29	11101	1D
30	11110	1E
31	11111	1F
32	100000	20
⋮	⋮	⋮
256	100000000	100
⋮		⋮
1024		400
⋮		⋮
32767		7FFF
32768		8000
⋮		⋮
65535		FFFF

1 Kilobyte (1 KB) = 1024 = 400h
64 Kilobytes (64 KB) = 65536 = 10000h
1 Megabyte (1 MB) = 1,048,576 = 100000h

16 or so lines of the table, because you will often need to express small numbers in all three systems.

A problem in working with different number systems is the meaning of the symbols used. For example, as you have seen, 10 means ten in the decimal system, sixteen in hex, and two in binary. In this book, the following convention is used whenever confusion may arise: hex numbers are followed by the letter h; for example, 1A34h. Binary numbers are followed by the letter b; for example, 101b. Decimal numbers are followed by the letter d; for example, 79d.

2.2 Conversion Between Number Systems

In working with assembly language, it is often necessary to take a number expressed in one system and write it in a different system.

Converting Binary and Hex to Decimal

Consider the hex number 82AD. It can be written as

$$\begin{aligned} 8A2Dh &= 8 \times 16^3 + A \times 16^2 + 2 \times 16^1 + D \times 16^0 \\ &= 8 \times 16^3 + 10 \times 16^2 + 2 \times 16^1 + 13 \times 16^0 = 35373d \end{aligned}$$

Similarly, the binary number 11101 may be written as

$$11101b = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 29d$$

This gives one way to convert a binary or hex number to decimal, but an easier way is to use nested multiplication. For example,

$$\begin{aligned} 8A2D &= 8 \times 16^3 + A \times 16^2 + 2 \times 16^1 + D \times 16^0 \\ &= ((8 \times 16 + A) \times 16 + 2) \times 16 + D \\ &= ((8 \times 16 + 10) \times 16 + 2) \times 16 + 13 \\ &= 35373d \end{aligned}$$

This can be easily implemented with a calculator: Multiply the first hex digit by 16, and add the second hex digit. Multiply that result by 16, and add the third hex digit. Multiply the result by 16, add the next hex digit, and so on.

The same procedure converts binary to decimal. Just multiply each result by 2 instead of 16.

Example 2.1 Convert 11101 to decimal.

Solution: 1 1 1 0 1

$$= 1 \times 2 + 1 \rightarrow 3 \times 2 + 1 \rightarrow 7 \times 2 + 0 \rightarrow 14 \times 2 + 1 = 29d$$

Example 2.2 Convert 2BD4h to decimal.

Solution: 2 B D 4

$$= 2 \times 16 + 11 \rightarrow 43 \times 16 + 13 \rightarrow 701 \times 16 + 4 = 11220$$

where we have used the fact that Bh = 11 and Dh = 13.

Converting Decimal to Binary and Hex

Suppose we want to convert 11172 to hex. The answer 2BA4h may be obtained as follows. First, divide 11172 by 16. We get a quotient of 698 and a remainder of 4. Thus

$$11172 = 698 \times 16 + 4$$

The remainder 4 is the unit's digit in hex representation of 11172. Now divide 698 by 16. The quotient is 43, and the remainder is 10 = Ah. Thus

$$698 = 43 \times 16 + \text{Ah}$$

The remainder Ah is the sixteen's digit in the hex representation of 11172. We just continue this process, each time dividing the most recent quotient by 16, until we get a 0 quotient. The remainder each time is a digit in the hex representation of 11172. Here are the calculations:

$$\begin{aligned} 11172 &= 698 \times 16 + 4 \\ 698 &= 43 \times 16 + 10(\text{Ah}) \\ 43 &= 2 \times 16 + 11(\text{Bh}) \\ 2 &= 0 \times 16 + 2 \end{aligned}$$

Now just convert the remainders to hex and put them together in reverse order to get 2BA4h.

This same process may be used to convert decimal to binary. The only difference is that we repeatedly divide by 2.

Example 2.3 Convert 95 to binary.

Solution:

$$\begin{aligned} 95 &= 47 \times 2 + 1 \\ 47 &= 23 \times 2 + 1 \\ 23 &= 11 \times 2 + 1 \\ 11 &= 5 \times 2 + 1 \\ 5 &= 2 \times 2 + 1 \\ 2 &= 1 \times 2 + 0 \\ 1 &= 0 \times 2 + 1 \end{aligned}$$

Taking the remainders in reverse order, we get 95 = 1011111b.

Conversions Between Hex and Binary

To convert a hex number to binary, we need only express each hex digit in binary.

Example 2.4 Convert 2B3Ch to binary.

Solution:

```
  2    B    3    C
= 0010 1011 0011 1100
= 0010101100111100
```

To go from binary to hex, just reverse this process; that is, group the binary digits in fours starting from the right. Then convert each group to a hex digit.

Example 2.5 Convert 1110101010 to hex.

Solution: 1110101010 = 11 1010 1010 = 3AAh

2.3 Addition and Subtraction

Sometimes you will want to do binary or hex addition and subtraction. Because these operations are done by rote in decimal, let's review the process to see what is involved.

Addition

Consider the following decimal addition

$$\begin{array}{r} 2546 \\ +\,1872 \\ \hline 4418 \end{array}$$

To get the unit's digit in the sum, we just compute 6 + 2 = 8. To get the ten's digit, compute 4 + 7 = 11. We write down 1 and carry 1 to the hundred's column. In that column we compute 5 + 8 + 1 = 14. We write down 4 and carry 1 to the last column. In that column we compute 2 + 1 + 1 = 4 and write it down, and the sum is complete.

A reason that decimal addition is easy for us is that we memorized the addition table for small numbers a long time ago. Table 2.3A is an addition table for small hex numbers. To compute Bh + 9h, for example, just intersect the row containing B and the column containing 9, and read 14h.

By using the addition table, hex addition may be done in exactly the same way as decimal addition. Suppose we want to compute the following hex sum:

Table 2.3A Hexadecimal Addition Table

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Table 2.3B Binary Addition Table

	0	**1**
0	0	1
1	1	10

```
  5B39h
+ 7AF4h
  -----
  D62Dh
```

In the unit's column, we compute 9h + 4h = 13d = Dh. In the next column, we get 3h + Fh = 12h. Write down 2 and carry 1 to the next column. In that column, compute Bh + Ah + 1 = 16h. Write 6, and carry 1 to the last column. There we compute 5h + 7h + 1 = Dh, and we are done.

Binary addition is done the same way as decimal and hex addition, but is a good deal easier because the binary addition table is so small (Table 2.3B). To do the sum

```
  100101111
+    110110
  ---------
  101100101
```

Compute 1 + 0 = 1 in the unit's column. In the next column, add 1 + 1 = 10b. Write down 0 and carry 1 to the next column, where we get 1 + 1 + 1 = 11b. Write down 1, carry 1 to the next column, and so on.

Subtraction

Let's begin with the decimal subtraction

```
  bb
  9145
- 7283
  ----
  1862
```

In the unit's column, we compute 5 - 3 = 2. To do the ten's, we first borrow 1 from the hundred's column (to remember that we have done this, we may place a "b" above the hundred's column), and compute 14 - 8 = 6. In the hundred's column, we must again borrow 1 from the next column, and compute 11 - 2 - 1 (the previous borrow) = 8. In the last column, we get 9 - 7 - 1 = 1.

Hex subtraction may be done the same way as decimal subtraction. To compute the hex difference

```
  bb
  D26F
- BA94
  ----
  17DB
```

we start with Fh - 4h = Bh. To do the next (sixteen's) column, we must borrow 1 from the third column, and compute

16h - 9h = ?

The easy way to figure this is to go to row 9 in Table 2.3A, and notice that 16 appears in column D. This means that 9h + Dh = 16h, so 16h - 9h = Dh. In the third column, after borrowing, we must compute 12h - Ah - 1 = 11h - Ah. In row A, 11 appears in column 7 so 11h - Ah = 7h. Finally in the last column, we have Ch - Bh = 1.

Now let us look at binary subtraction, for example,

```
  bb
  1001
- 0111
  ----
  0010
```

The unit's column is easy, 1 - 1 = 0. We must borrow to do the two's column, getting 10 - 1 = 1. To do the four's column, we must again borrow, computing 10 - 1 - 1 (since we borrowed from this column) = 0. Finally in the last column, we have 0 - 0 = 0.

2.4 How Integers Are Represented in the Computer

The hardware of a computer necessarily restricts the size of numbers that can be stored in a register or memory location. In this section, we will see how integers can be stored in an 8-bit byte or a 16-bit word. In Chapter 18 we talk about how real numbers can be stored.

In the following, we'll need to refer to two particular bits in a byte or word: the **most significant bit**, or **msb**, is the leftmost bit. In a word, the msb is bit 15; in a byte, it is bit 7. Similarly, the **least significant bit**, or **lsb**, is the rightmost bit; that is, bit 0.

2.4.1 Unsigned Integers

An **unsigned integer** is an integer that represents a magnitude, so it is never negative. Unsigned integers are appropriate for representing quantities that can never be negative, such as addresses of memory locations, counters, and ASCII character codes (see later). Because unsigned integers are by definition nonnegative, none of the bits are needed to represent the sign, and so all 8 bits in a byte, or 16 bits in a word, are available to represent the number.

The largest unsigned integer that can be stored in a byte is 11111111 = FFh = 255. This is not a very big number, so we usually store integers in words. The biggest unsigned integer a 16-bit word can hold is 1111111111111111 = FFFFh = 65535. This is big enough for most purposes. If not, two or more words may be used.

Note that if the least significant bit of an integer is 1, the number is odd, and it's even if the lsb is 0.

2.4.2 Signed Integers

A **signed integer** can be positive or negative. The most significant bit is reserved for the sign: 1 means negative and 0 means positive. Negative integers are stored in the computer in a special way known as **two's complement**. To explain it, we first define **one's complement**, as follows.

One's Complement

The one's complement of an integer is obtained by complementing each bit; that is, replace each 0 by a 1 and each 1 by a 0. In the following, we assume numbers are 16 bits.

Example 2.6 Find the one's complement of 5 = 0000000000000101.

Solution:

```
                     5 = 0000000000000101
One's complement of 5 = 1111111111111010
```

Note that if we add 5 and its one's complement, we get 1111111111111111.

Two's Complement

To get the two's complement of an integer, just add 1 to its one's complement.

Example 2.7 Find the two's complement of 5.

Solution: From above,

```
  one's complement of 5 = 1111111111111010
                                       + 1
  -----------------------------------------
  two's complement of 5 = 1111111111111011 = FFFBh
```

Now look what happens when we add 5 and its two's complement:

```
                        5 = 0000000000000101
  + two's complement of 5 = 1111111111111011
                            -----------------
                           10000000000000000
```

We end up with a 17-bit number. Because a computer word circuit can only hold 16 bits, the 1 carried out from the most significant bit is lost, and the 16-bit result is 0. As 5 and its two's complement add up to 0, the two's complement of 5 must be a correct representation of -5.

It is easy to see why the two's complement of any integer *N* must represent -*N*: Adding *N* and its one's complement gives 16 ones; adding 1 to this produces 16 zeros with a 1 carried out and lost. The result stored is always 0000000000000000.

The following example shows what happens when a number is complemented two times.

Example 2.8 Find the two's complement of the two's complement of 5.

Solution: We would guess that after complementing 5 two times, the result should be 5. To verify this, from above,

```
                two's complement of 5 = 1111111111111011
  one's complement of 1111111111111011 = 0000000000000100
                                                      + 1
  --------------------------------------------------------
  two's complement of 1111111111111011 = 0000000000000101 = 5
```

Example 2.9 Show how the decimal integer -97 would be represented (a) in 8 bits, and (b) in 16 bits. Express the answers in hex.

Solution: A decimal-to-hex conversion using repeated division by 16 yields

$$97 = 6 \times 16 + 1$$
$$6 = 0 \times 16 + 6$$

Thus 97 = 61h. To represent -97, we need to express 61h in binary and take the two's complement.

a. In 8 bits, we get

```
                  61h = 0110 0001
     one's complement = 1001 1110
                             + 1
     ----------------------------
     two's complement = 1001 1111 = 9Fh
```

b. In 16 bits, we get

```
                  61h = 0000 0000 0110 0001
     one's complement = 1111 1111 1001 1110
                                       + 1
                        -------------------
                        1111 1111 1001 1111 = FF9Fh
```

Subtraction as Two's Complement Addition

The advantage of two's complement representation of negative integers in the computer is that subtraction can be done by bit complementation and addition, and circuits that add and complement bits are easy to design.

Example 2.10 Suppose AX contains 5ABCh and BX contains 21FCh. Find the difference of AX minus BX by using complementation and addition.

Solution:

```
                  AX contains 5ABCh = 0101 1010 1011 1100
                  BX contains 21FCh = 0010 0001 1111 1100
                              5ABCh = 0101 1010 1011 1100
     + one's complement of 21FCh    = 1101 1110 0000 0011
                                                      + 1
                                    ---------------------
                        Difference = 1 0011 1000 1100 0000 = 38C0h
```

A one is carried out of the most significant bit and is lost. The answer stored, 38C0h, is correct, as may be verified by hex subtraction.

2.4.3 Decimal Interpretation

In the last section, we saw how signed and unsigned decimal integers may be represented in the computer. The reverse problem is to interpret the contents of a byte or word as a signed or unsigned decimal integer.

- *Unsigned decimal interpretation:* Just do a binary-to-decimal conversion. It's usually easier to convert binary to hex first, and then convert hex to decimal.
- *Signed decimal interpretation:* If the most significant bit is 0, the number is positive, and the signed decimal is the same as the unsigned decimal. If the msb is 1, the number is negative, so call it $-N$. To find N, just take the twos' complement and then convert to decimal as before.

Example 2.11 Suppose AX contains FE0Ch. Give the unsigned and signed decimal interpretations.

Table 2.4A Signed and Unsigned Decimal Interpretations of 16-Bit Register/Memory Contents

Hex	*Unsigned decimal*	*Signed decimal*
0000	0	0
0001	1	1
0002	2	2
.	.	.
.	.	.
.	.	.
0009	9	9
000A	10	10
.	.	.
.	.	.
.	.	.
7FFE	32766	32766
7FFF	32767	32767
8000	32768	-32768
8001	32769	-32767
.	.	.
.	.	.
.	.	.
FFFE	65534	-2
FFFF	65535	-1

Solution: Conversion of FE0Ch to decimal yields 65036, which is the unsigned decimal interpretation.

For the signed interpretation, FE0Ch = 1111111000001100. Since the sign bit is 1, this is a negative number, call it –*N*. To find *N*, get the two's complement.

```
                 FE0Ch = 1111 1110 0000 1100
     one's complement  = 0000 0001 1111 0011
                                         + 1
                   N = 0000 0001 1111 0100 = 01F4h = 500
```

Thus, AX contains –500.

Tables 2.4A and 2.4B give 16-bit word and 8-bit byte hex values and their signed and unsigned decimal interpretations. Note the following:

1. Because the most significant bit of a positive signed integer is 0, the leading hex digit of a positive signed integer is 0 – 7; integers beginning with 8–Fh have 1 in the sign bit, so they are negative.
2. The largest 16-bit positive signed integer is 7FFFh = 32767; the smallest negative integer is 8000h = –32768. For a byte, the largest positive integer is 7Fh = 127 and the smallest is 80h = –128.
3. The following relationship holds between the unsigned and signed decimal interpretations of the contents of a 16-bit word:

Table 2.4B Signed and Unsigned Decimal Interpretations of a Byte

Hex	*Unsigned decimal*	*Signed decimal*
00	0	0
01	1	1
02	2	2
.	.	.
.	.	.
.	.	.
09	9	9
0A	10	10
.	.	.
.	.	.
.	.	.
7E	126	126
7F	127	127
80	128	-128
81	129	-127
.	.	.
.	.	.
.	.	.
FE	254	-2
FF	255	-1

For 0000h–7FFFh, signed decimal = unsigned decimal.
For 8000h–FFFFh, signed decimal = unsigned decimal – 65536.

There are similar relations for the contents of an eight-bit byte:

For 00h–7Fh, signed decimal = unsigned decimal.
For 80h–FFh, signed decimal = unsigned decimal – 256.

Example 2.12 Use observation 3, from the above, to rework example 2.11.

Solution: We saw that the unsigned decimal interpretation of FE0Ch is 65036. Because the leading hex digit is Fh, the content is negative in a signed sense. To interpret it, just subtract 65536 from the unsigned decimal. Thus

signed decimal interpretation = 65036 – 65536 = –500

2.5 Character Representation

ASCII Code

Not all data processed by the computer are treated as numbers. I/O devices such as the video monitor and printer are character oriented, and programs such as word processors deal with characters exclusively. Like all data, characters must be coded in binary in order to be processed by the computer. The most popular encoding scheme for characters is **ASCII (American Standard Code for Information Interchange) code.** Originally used in communications by teletype, ASCII code is used by all personal computers today.

The ASCII code system uses seven bits to code each character, so there are a total of $2^7 = 128$ ASCII codes. Table 2.5 gives the ASCII codes and the characters associated with them.

Notice that only 95 ASCII codes, from 32 to 126, are considered to be printable. The codes 0 to 31 and also 127 were used for communication control purposes and do not produce printable characters. Most microcomputers use only the printable characters and a few control characters such as LF, CR, BS, and Bell.

Because each ASCII character is coded by only seven bits, the code of a single character fits into a byte, with the most significant bit set to zero. The printable characters can be displayed on the video monitor or printed by the printer, while the control characters are used to control the operations of these devices. For example, to display the character A on the screen, a program sends the ASCII code 41h to the screen; and to move the cursor back to the beginning of the line, a program sends the ASCII code 0Dh, which is the CR character, to the screen.

A computer may assign special display characters to some of the non-printed ASCII codes. As you will see later, the screen controller for the IBM PC can actually display an extended set of 256 characters. Appendix A shows the 256 display characters of the IBM PC.

Example 2.[illegible] Show how the character string "RG 2z" is stored in memory, starting at address 0.

Solution: From Table 2.5, we have

Character	*ASCII Code (hex)*	*ASCII Code (binary)*
R	52	0101 0010
G	47	0100 0111
space	20	0010 0000
2	32	0011 0010
z	7A	0111 1010

So memory would look like this:

Address	*Contents*
0	01010010
1	01000111
2	00100000
3	00110010
4	01111010

The Keyboard

It's reasonable to guess that the keyboard identifies a key by generating an ASCII code when the key is pressed. This was true for a class of keyboards known as *ASCII keyboards* used by some early microcomputers.

Table 2.5 ASCII Code

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	<CC>	32	20	SP	64	40	@	96	60	‘
1	01	<CC>	33	21	!	65	41	A	97	61	a
2	02	<CC>	34	22	"	66	42	B	98	62	b
3	03	<CC>	35	23	#	67	43	C	99	63	c
4	04	<CC>	36	24	$	68	44	D	100	64	d
5	05	<CC>	37	25	%	69	45	E	101	65	e
6	06	<CC>	38	26	&	70	46	F	102	66	f
7	07	<CC>	39	27	'	71	47	G	103	67	g
8	08	<CC>	40	28	(	72	48	H	104	68	h
9	09	<CC>	41	29	)	73	49	I	105	69	i
10	0A	<CC>	42	2A	*	74	4A	J	106	6A	j
11	0B	<CC>	43	2B	+	75	4B	K	107	6B	k
12	0C	<CC>	44	2C	,	76	4C	L	108	6C	l
13	0D	<CC>	45	2D	-	77	4D	M	109	6D	m
14	0E	<CC>	46	2E	.	78	4E	N	110	6E	n
15	0F	<CC>	47	2F	/	79	4F	O	111	6F	o
16	10	<CC>	48	30	0	80	50	P	112	70	p
17	11	<CC>	49	31	1	81	51	Q	113	71	q
18	12	<CC>	50	32	2	82	52	R	114	72	r
19	13	<CC>	51	33	3	83	53	S	115	73	s
20	14	<CC>	52	34	4	84	54	T	116	74	t
21	15	<CC>	53	35	5	85	55	U	117	75	u
22	16	<CC>	54	36	6	86	56	V	118	76	v
23	17	<CC>	55	37	7	87	57	W	119	77	w
24	18	<CC>	56	38	8	88	58	X	120	78	x
25	19	<CC>	57	39	9	89	59	Y	121	79	y
26	1A	<CC>	58	3A	:	90	5A	Z	122	7A	z
27	1B	<CC>	59	3B	;	91	5B	[	123	7B	{
28	1C	<CC>	60	3C	<	92	5C	\	124	7C	|
29	1D	<CC>	61	3D	=	93	5D	]	125	7D	}
30	1E	<CC>	62	3E	>	94	5E	^	126	7E	~
31	1F	<CC>	63	3F	?	95	5F	_	127	7F	<CC>

<CC> denotes a control character
SP = blank space

Special Control Characters

Dec	Hex	Char	Meaning
7	07	BEL	bell
8	08	BS	backspace
9	09	HT	horizontal tab
10	0A	LF	line feed
12	0C	FF	form feed
13	0D	CR	carriage return

However, modern keyboards have many control and function keys in addition to ASCII character keys, so other encoding schemes are used. For the IBM PC, each key is assigned a unique number called a **scan code**; when a key is pressed, the keyboard sends the key's scan code to the computer. Scan codes are discussed in Chapter 12.

SUMMARY

- Numbers are represented in different ways, according to the basic symbols used. The binary system uses two symbols, 0 and 1. The decimal system uses 0–9. The hexadecimal system uses 0–9, A–F.
- Binary and hex numbers can be converted to decimal by a process of nested multiplication.
- A hex number can be converted to decimal by a process of repeated division by 16; similarly, a binary number can be converted to decimal by a process of repeated division by 2.
- Hex numbers can be converted to binary by converting each hex digit to binary; binary numbers are converted to hex by grouping the bits in fours, starting from the right, and converting each group to a hex digit.
- The process of adding and subtracting hex and binary numbers is the same as for decimal numbers, and can be done with the help of the appropriate addition table.
- Negative numbers are stored in two's complement form. To get the two's complement of a number, complement each bit and add 1 to the result.
- If A and B are stored integers, the processor computes A – B by adding the two's complement of B to A.
- The range of unsigned integers that can be stored in a byte is 0–255; in a 16-bit word, it is 0–65535.
- For signed numbers, the most significant bit is the sign bit; 0 means positive and 1 means negative. The range of signed numbers that can be stored in a byte is –128 to 127; in a word, it is –32768 to 32767.
- The unsigned decimal interpretation of a word is obtained by converting the binary value to decimal. If the sign bit is 0, this is also the signed decimal interpretation. If the sign bit is 1, the signed decimal interpretation may be obtained by subtracting 65536 from the unsigned decimal interpretation.
- The standard encoding scheme for characters is the ASCII code.
- A character requires seven bits to code, so it can be stored in a byte.
- The IBM screen controller can generate a character for each of the 256 possible numbers that can be stored in a byte.

Glossary

ASCII (American Standard Code for Information Interchange) codes	The encoding scheme for characters used on all personal computers
binary number system	Base two system in which the digits are 0 and 1
hexadecimal number system	Base sixteen system in which the digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F
least significant bit, lsb	The rightmost bit in a word or byte; that is, bit 0
most significant bit, msb	The leftmost bit in a word or byte; that is, bit 15 in a word or bit 7 in a byte
one's complement of a binary number	Obtained by replacing each 0 bit by 1 and each 1 bit by 0
scan code	A number used to identify a key on the keyboard
signed integer	An integer that can be positive or negative
two's complement of a binary number	Obtained by adding 1 to the one's complement
unsigned integer	An integer representing a magnitude; that is, always positive

Exercises

In many applications, it saves time to memorize the conversions among small binary, decimal, and hex numbers. Without referring to Table 2.2, fill in the blanks in the following table:

Binary	*Decimal*	*Hex*
1001	9	B
1010	10	C
[illegible]	11	D
1101	12	E
1110	13	
1111	14	8

2. Convert the following binary and hex numbers to decimal:
   a. 1110
   b. 100101011101
   c. 46Ah
   d. FAE2Ch
3. Convert the following decimal numbers:
   a. 97 to binary
   b. 627 to binary
   c. 921 to hex
   d. 6120 to hex
4. Convert the following numbers:

a. 1001011 to hex
b. 1001010110101110 to hex
c. A2Ch to binary
d. B34Dh to binary

5. Perform the following additions:
   a. 100101b + 10111b
   b. 100111101b + 10001111001b
   c. B23CDh + 17912h
   d. FEFFEh + FBCADh
6. Perform the following subtractions:
   a. 11011b - 10110b
   b. 10000101b - 111011b
   c. 5FC12h - 3ABD1h
   d. F001Eh - 1FF3Fh
7. Give the 16-bit representation of each of the following decimal integers. Write the answer in hex.
   a. 234
   b. -16
   c. 31634
   d. -32216
8. Do the following binary and hex subtractions by two's complement addition.
   a. 10110100 - 10010111
   b. 10001011 - 11110111
   c. FE0Fh - 12ABh
   d. 1ABCh - B3EAh
9. Give the unsigned and signed decimal interpretations of each of the following 16-bit or 8-bit numbers.
   a. 7FFEh
   b. 8543h
   c. FEh
   d. 7Fh
10. Show how the decimal integer -120 would be represented
    a. in 16 bits.
    b. in 8 bits.
11. For each of the following decimal numbers, tell whether it could be stored (a) as a 16-bit number (b) as an 8-bit number.
    a. 32767
    b. -40000
    c. 65536
    d. 257 ✓
    e. ✓ -128
12. For each of the following 16-bit signed numbers, tell whether it is positive or negative.
    a. 1010010010010100b
    b. 78E3h

c. CB33h

d. 807Fh

e. 9AC4h

13. If the character string "$12.75" is being stored in memory starting at address 0, give the hex contents of bytes 0–5.

14. Translate the following secret message, which has been encoded in ASCII as 41 74 74 61 63 6B 20 61 74 20 44 61 77 6E.

15. Suppose that a byte contains the ASCII code of an uppercase letter. What hex number should be added to it to convert it to lower case?

16. Suppose that a byte contains the ASCII code of a decimal digit; that is, "0". . . "9." What hex number should be subtracted from the byte to convert it to the numerical form of the characters?

17. It is not really necessary to refer to the hex addition table to do addition and subtraction of hex digits. To compute Eh + Ah, for example, first copy the hex digits:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
```

Now starting at Eh, move to the right Ah = 10 places. When you go off the right end of the line, continue on from the left end and attach a 1 to each number you pass:

```
10 11 12 13 14 15 16 17 18 9 A B C D E F
                     STOP ^       START ^
```

You get Eh + Ah = 18h. Subtraction can be done similarly. For example, to compute 15h – Ch, start at 15h and move left Ch = 12 places. When you go off the left end, continue on at the right:

```
10 11 12 13 14 15 6 7 8 9 A B C D E F
             ^ START   ^ STOP
```

You get 15h – Ch = 9h.
Rework exercises 5(c) and 6(c) by this method.