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 (four digits) or in hexadecimal as FFF (three 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. When the human communicates with the machine (through console switches or indicator lights or by means of programs written in *machine language*), the conversion from octal or hexadecimal to binary and vice versa is done by inspection by the human user.

1-5 COMPLEMENTS

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. 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 is defined as $(r^n - 1) - N$. For decimal numbers, $r = 10$ and $r - 1 = 9$, so the 9's complement of N is $(10^n - 1) - N$. Now, 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. Some numerical examples follow.

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 causes the bit to change from 0 to 1 or from 1 to 0. 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 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^n 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.

The 10's complement of 012398 is 987602.

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.

The 2's complement of 1101100 is 0010100.

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.

In the previous definitions, it was assumed that the numbers do not have a radix point. If the original number N contains a radix point, the point should be removed

temporarily in order to form the r's or $(r-1)$'s complement. The radix point is then restored to the complemented number in the same relative position. It is also worth mentioning that the complement of the complement restores the number to its original value. The r's complement of N is $r^n - N$. The complement of the complement is $r^n - (r^n - N) = N$, giving back the original number.

Subtraction with Complements

The direct method of subtraction taught in elementary schools uses the borrow concept. In this method, we borrow a 1 from a higher significant position when the minuend digit is smaller than the subtrahend digit. This seems to be easiest when people perform subtraction with paper and pencil. When subtraction is implemented with digital hardware, this method is found to be less efficient than the method that uses complements.

The subtraction of two n-digit unsigned numbers $M - N$ in base r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. This performs $M + (r^n - N) = M - N + r^n$.
2. If $M \geq N$, the sum will produce an end carry, r^n, which is discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

The following examples illustrate the procedure.

Example 1-5 Using 10's complement, subtract 72532 − 3250.

M =	72532
10's complement of N =	+ 96750
Sum =	169282
Discard end carry 10^5 =	−100000
Answer =	69282

■

Note that M has 5 digits and N has only 4 digits. Both numbers must have the same number of digits; so we can write N as 03250. Taking the 10's complement of N produces a 9 in the most significant position. The occurrence of the end carry signifies that $M \geq N$ and the result is positive.

Example 1-6 Using 10's complement, subtract 3250 − 72532.

M =	03250
10's complement of N =	+ 27468
Sum =	30718

There is no end carry.

Answer: −(10's complement of 30718) = −69282 ■

Note that since $3250 < 72532$, the result is negative. Since we are dealing with unsigned numbers, there is really no way to get an unsigned result for this case. When subtracting with complements, the negative answer is recognized from the absence of the end carry and the complemented result. When working with paper and pencil, we can change the answer to a signed negative number in order to put it in a familiar form.

Subtraction with complements is done with binary numbers in a similar manner using the same procedure outlined before.

Example 1-7 Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$ and (b) $Y - X$ using 2's complements.

(a)

X =	1010100
2's complement of Y =	+ 0111101
Sum =	10010001
Discard end carry 2^7 =	−10000000
Answer: $X - Y$ =	0010001

(b)

Y =	1000011
2's complement of X =	+ 0101100
Sum =	1101111

There is no end carry.

Answer: $Y - X$ = −(2's complement of 1101111) = −0010001 ■

Subtraction of unsigned numbers can be done also by means of the $(r-1)$'s complement. Remember that the $(r-1)$'s complement is one less than the r's complement. Because of this, the result of adding the minuend to the complement of the subtrahend produces a sum that is 1 less than the correct difference when an end carry occurs. Removing the end carry and adding 1 to the sum is referred to as an *end-around carry*.

Example 1-8 Repeat Example 1-7 using 1's complement.

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

$X =$	1010100
1's complement of $Y =$	+ 0111100
Sum =	10010000
End-around carry	→ + 1
Answer: $X - Y =$	0010001

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

$Y =$	1000011
1's complement of $X =$	+ 0101011
Sum =	1101110

There is no end carry.

Answer: $Y - X = -$(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 for 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, commonly referred to as *bits*. 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 a +9 (signed binary) because the leftmost bit is 0. The string of bits 11001 represent the binary equivalent of 25 when considered as an unsigned number or as − 9 when considered as a signed number because of the 1 in the leftmost position, which designates negative, and the other four bits, which represent binary 9. Usually, there is no confusion in identifying the bits if the 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 for representing negative numbers, referred to as the *signed-complement* system. 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 to give 00001001. Note that all eight bits must have a value and, 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:

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

In signed-magnitude, −9 is obtained from +9 by changing 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.

The signed-magnitude system is used in ordinary arithmetic, but is awkward when employed in computer arithmetic. Therefore, the signed-complement is normally used. The 1's complement imposes some difficulties and is seldom used for arithmetic operations except in some older computers. The 1's complement 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 following discussion of signed binary arithmetic deals exclusively with the signed-2's-complement representation of negative numbers. The same procedures can be applied to the signed-1's-complement system by including the end-around carry as 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 result the sign of the larger magnitude. For example, $(+25) + (-37) = -(37 - 25) = -12$ and is done by subtracting the smaller magnitude 25 from the larger magnitude 37 and using the sign of 37 for the sign of the result. This is a process that requires the comparison of the signs and the 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. Note that negative numbers must be initially in 2's complement and that the sum obtained after the addition if negative is in 2's-complement form.

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

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 since we are not limited by the width of the page. We just add another 0 to a positive number and another 1 to a negative number in the most-significant position to extend them 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 when in signed-2's complement, it is necessary to convert it 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 very 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 occurs because a subtraction operation can be changed to an addition operation if the sign of the subtrahend is changed. This 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 its 2's complement. The reverse is also true because the complement of a negative number in complement form produces the equivalent positive number. Consider the subtraction of $(-6) - (-13) = +7$. In binary with eight bits, this is written as (11111010 − 11110011). The subtraction is changed to addition by taking the 2's complement of the subtrahend (−13) to give (+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. 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

Electronic 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 a 0 or a 1. Digital systems represent and manipulate not only binary numbers, but also many other discrete elements of information. Any discrete element of information distinct among a group of quantities can be represented by a binary code. Binary codes play an important role in digital computers. The codes must be in binary because computers can only hold 1's and 0's. 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.

A *bit*, by definition, is a binary digit. When used in conjunction with a binary code, it is better to think of it as denoting a binary quantity equal to 0 or 1. To represent a

group of 2^n distinct elements in a binary code requires a minimum of n bits. This is because it is possible to arrange n bits in 2^n distinct ways. For example, a group of four distinct quantities can be represented by a two-bit code, with each quantity assigned one of the following bit combinations: 00, 01, 10, 11. A group of eight elements requires a three-bit code, with each element assigned to one and only one of the following: 000, 001, 010, 011, 100, 101, 110, 111. The examples show that the distinct bit combinations of an n-bit code can be found by counting in binary from 0 to $(2^n - 1)$. Some bit combinations are unassigned when the number of elements of the group to be coded is not a multiple of the power of 2. The ten decimal digits 0, 1, 2, . . . , 9 are an example of such a group. A binary code that distinguishes among ten elements must contain at least four bits; three bits can distinguish a maximum of eight elements. Four bits can form 16 distinct combinations, but since only ten digits are coded, the remaining six combinations are unassigned and not used.

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 ten decimal digits can be coded with ten bits, and each decimal digit 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.

Decimal Codes

Binary codes for decimal digits require a minimum of four bits. Numerous different codes can be obtained by arranging four or more bits in ten distinct possible combinations. A few possibilities are shown in Table 1-2.

TABLE 1-2
Binary codes for the decimal digits

Decimal digit	(BCD) 8421	Excess-3	84-2-1	2421	(Biquinary) 5043210
0	0000	0011	0000	0000	0100001
1	0001	0100	0111	0001	0100010
2	0010	0101	0110	0010	0100100
3	0011	0110	0101	0011	0101000
4	0100	0111	0100	0100	0110000
5	0101	1000	1011	1011	1000001
6	0110	1001	1010	1100	1000010
7	0111	1010	1001	1101	1000100
8	1000	1011	1000	1110	1001000
9	1001	1100	1111	1111	1010000

The BCD (binary-code decimal) is a straight assignment of the binary equivalent. It is possible to assign weights to the binary bits according to their positions. The weights in the BCD code are 8, 4, 2, 1. The bit assignment 0110, for example, can be interpreted by the weights to represent the decimal digit 6 because $0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 = 6$. It is also possible to assign negative weights to a decimal code, as shown by the 8, 4, −2, −1 code. In this case, the bit combination 0110 is interpreted as the decimal digit 2, as obtained from $0 \times 8 + 1 \times 4 + 1 \times (-2) + 0 \times (-1) = 2$. Two other weighted codes shown in the table are the 2421 and the 5043210. A decimal code that has been used in some old computers is the excess-3 code. This is an unweighted code; its code assignment is obtained from the corresponding value of BCD after the addition of 3.

Numbers are represented in digital computers either in binary or in decimal through a binary code. When specifying data, the user likes to give the data in decimal form. The input decimal numbers are stored internally in the computer by means of a decimal code. Each decimal digit requires at least four binary storage elements. The decimal numbers are converted to binary when arithmetic operations are done internally with numbers represented in binary. It is also possible to perform the arithmetic operations directly in decimal with all numbers left in a coded form throughout. For example, the decimal number 395, when converted to binary, is equal to 110001011 and consists of nine binary digits. The same number, when represented internally in the BCD code, occupies four bits for each decimal digit, for a total of 12 bits: 001110010101. The first four bits represent a 3, the next four a 9, and the last four a 5.

It is very important to understand the difference between *conversion* of a decimal number to binary and the binary *coding* of a decimal number. In each case, the final result is a series of bits. The bits obtained from conversion are binary digits. Bits obtained from coding are combinations of 1's and 0's arranged according to the rules of the code used. Therefore, it is extremely important to realize that a series of 1's and 0's in a digital system may sometimes represent a binary number and at other times represent some other discrete quantity of information as specified by a given binary code. The BCD code, for example, has been chosen to be both a code and a direct binary conversion, as long as the decimal numbers are integers from 0 to 9. For numbers greater than 9, the conversion and the coding are completely different. This concept is so important that it is worth repeating with another example. The binary conversion of decimal 13 is 1101; the coding of decimal 13 with BCD is 00010011.

From the five binary codes listed in Table 1-2, the BCD seems the most natural to use and is indeed the one most commonly encountered. The other four-bit codes listed have one characteristic in common that is not found in BCD. The excess-3, the 2, 4, 2, 1, and the 8, 4, −2, −1 are self-complementing codes, that is, the 9's complement of the decimal number is easily obtained by changing 1's to 0's and 0's to 1's. For example, the decimal 395 is represented in the 2, 4, 2, 1 code by 001111111011. Its 9's complement 604 is represented by 110000000100, which is easily obtained from the replacement of 1's by 0's and 0's by 1's. This property is useful when arithmetic operations are internally done with decimal numbers (in a binary code) and subtraction is calculated by means of 9's complement.

The biquinary code shown in Table 1-2 is an example of a seven-bit code with error-detection properties. Each decimal digit consists of five 0's and two 1's placed in the corresponding weighted columns. The error-detection property of this code may be understood if one realizes that digital systems represent binary 1 by one distinct signal

and binary 0 by a second distinct signal. During transmission of signals from one location to another, an error may occur. One or more bits may change value. A circuit in the receiving side can detect the presence of more (or less) than two 1's and if the received combination of bits does not agree with the allowable combination, an error is detected.

Error-Detection Code

Binary information can be transmitted from one location to another by electric wires or other communication medium. Any external noise introduced into the physical communication medium may change some of the bits from 0 to 1 or vice versa. The purpose of an error-detection code is to detect such bit-reversal errors. One of the most common ways to achieve error detection is by means of a *parity bit*. A parity bit is an extra bit included with a message to make the total number of 1's transmitted either odd or even. A message of four bits and a parity bit P are shown in Table 1-3. If an odd parity is adopted, the P bit is chosen such that the total number of 1's is odd in the five bits that constitute the message and P. If an even parity is adopted, the P bit is chosen so that the total number of 1's in the five bits is even. In a particular situation, 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 is done in the following manner. An even parity bit is generated in the sending end for each message transmission. The message, together with the parity bit, is transmitted to its destination. The parity of the received data is

TABLE 1-3
Parity bit

Odd parity		Even parity	
Message	P	Message	P
0000	1	0000	0
0001	0	0001	1
0010	0	0010	1
0011	1	0011	0
0100	0	0100	1
0101	1	0101	0
0110	1	0110	0
0111	0	0111	1
1000	0	1000	1
1001	1	1001	0
1010	1	1010	0
1011	0	1011	1
1100	1	1100	0
1101	0	1101	1
1110	0	1110	1
1111	1	1111	0

checked in the receiving end. If the parity of the received information is not even, it means that at least one bit has changed value during the transmission. This method detects one, three, or any odd combination of errors in each message that is transmitted. An even combination of errors is undetected. Additional error-detection schemes may be needed to take care of an even combination of errors.

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 a negative acknowledge message. If no error is detected, the receiver sends back an acknowledge message. The sending end will respond to a previous error 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 human operator to check for malfunctions in the transmission path.

Gray Code

Digital systems can be designed to process data in discrete form only. Many physical systems supply continuous output data. 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 1-4 to represent the digital data when it is converted from analog data. The advantage of the Gray code over binary numbers is that only one bit in the code group changes when 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

TABLE 1-4
Four-bit 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

first bit from the left changes from 0 to 1; the other three bits remain the same. When comparing this 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 where the normal sequence of binary numbers may produce an error or ambiguity during the transition from one number to the next. If binary numbers are used, a change from 0111 to 1000 may produce an intermediate erroneous number 1001 if the rightmost bit takes more time to change than the other three bits. The Gray code eliminates this problem since only one bit changes in value during any transition between two numbers.

A typical application of the Gray code occurs when analog data are represented by continuous change of a shaft position. 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 when detection is sensed in the line that separates any two segments.

ASCII Character Code

Many applications of digital computers require the handling of data not only of numbers, but also of letters. For instance, an insurance company with thousands of policy holders will use a computer to process its files. 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 ASCII (American Standard Code for Information Interchange). It uses seven bits to code 128 characters, as shown in Table 1-5. The seven bits of the code are designated by b_1 through b_7, with b_7 being the most-significant bit. The letter *A*, for example, is represented in ASCII as 1000001 (column 100, row 0001). The ASCII code 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 $.

The 34 control characters are designated in the ASCII table with abbreviated names. They are listed in the table with their full 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 typewriter controls such as backspace (BS), horizontal tabulation (HT), and carriage return (CR). Information separators are used to