The majority of digital components adhere to power-of-two magnitude definitions. However, some industries break from these conventions largely for reasons of product promotion. A key example is the hard disk drive industry, which specifies prefixes in decimal terms (e.g., 1 MB = 1,000,000 bytes). The advantage of doing this is to inflate the apparent capacity of the disk drive: a drive that provides 10,000,000,000 bytes of storage can be labeled as "10 GB" in decimal terms, but it would have to be labeled as only 9.31 GB in binary terms ($10^{10} \div 2^{30} = 9.31$).

1.5 BINARY ADDITION

Despite the fact that most engineers use hex data representation, it has already been shown that logic gates operate on strings of bits that compose each unit of data. Binary arithmetic is performed according to the same rules as decimal arithmetic. When adding two numbers, each column of digits is added in sequence from right to left and, if the sum of any column is greater than the value of the highest digit, a carry is added to the next column. In binary, the largest digit is 1, so any sum greater than 1 will result in a carry. The addition of 111_2 and 011_2 (7 + 3 = 10) is illustrated below.

	1	*1*	*1*	*0*	*carry bits*
		1	1	1	
+		0	1	1	
	1	0	1	0	

In the first column, the sum of two ones is 2_{10}, or 10_2, resulting in a carry to the second column. The sum of the second column is 3_{10}, or 11_2, resulting in both a carry to the next column and a one in the sum. When all three columns are completed, a carry remains, having been pushed into a new fourth column. The carry is, in effect, added to leading 0s and descends to the sum line as a 1.

The logic to perform binary addition is actually not very complicated. At the heart of a 1-bit adder is the XOR gate, whose result is the sum of two bits without the associated carry bit. An XOR gate generates a 1 when either input is 1, but not both. On its own, the XOR gate properly adds 0 + 0, 0 + 1, and 1 + 0. The fourth possibility, 1 + 1 = 2, requires a carry bit, because $2_{10} = 10_2$. Given that a carry is generated only when both inputs are 1, an AND gate can be used to produce the carry. A so-called *half-adder* is represented as follows:

$$\text{sum} = A \oplus B$$

$$\text{carry} = AB$$

This logic is called a *half-adder* because it does only part of the job when multiple bits must be added together. Summing multibit data values requires a carry to ripple across the bit positions starting from the LSB. The half-adder has no provision for a carry input from the preceding bit position. A *full-adder* incorporates a carry input and can therefore be used to implement a complete summation circuit for an arbitrarily large pair of numbers. Table 1.9 lists the complete full-adder input/output relationship with a carry input (C_{IN}) from the previous bit position and a carry output (C_{OUT}) to the next bit position. Note that all possible sums from zero to three are properly accounted for by combining C_{OUT} and sum. When $C_{IN} = 0$, the circuit behaves exactly like the half-adder.

TABLE 1.9 1-Bit Full-Adder Truth Table

C_{IN}	A	B	C_{OUT}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full-adder logic can be expressed in a variety of ways. It may be recognized that full-adder logic can be implemented by connecting two half-adders in sequence as shown in Fig. 1.9. This full-adder directly generates a sum by computing the XOR of all three inputs. The carry is obtained by combining the carry from each addition stage. A logical OR is sufficient for C_{OUT}, because there can never be a case in which both half-adders generate a carry at the same time. If the A + B half-adder generates a carry, the partial sum will be 0, making a carry from the second half-adder impossible. The associated logic equations are as follows:

$$\text{sum} = A \oplus B \oplus C_{IN}$$

$$C_{OUT} = AB + [(A \oplus B)C_{IN}]$$

Equivalent logic, although in different form, would be obtained using a K-map, because XOR/XNOR functions are not direct results of K-map AND/OR solutions.

1.6 SUBTRACTION AND NEGATIVE NUMBERS

Binary subtraction is closely related to addition. As with many operations, subtraction can be implemented in a variety of ways. It is possible to derive a Boolean equation that directly subtracts two numbers. However, an efficient solution is to add the negative of the subtrahend to the minuend

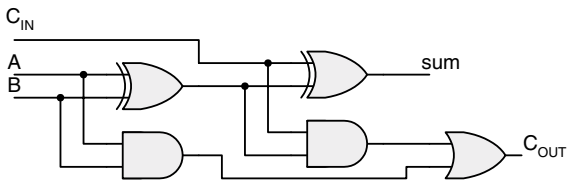

FIGURE 1.9 Full-adder logic diagram.