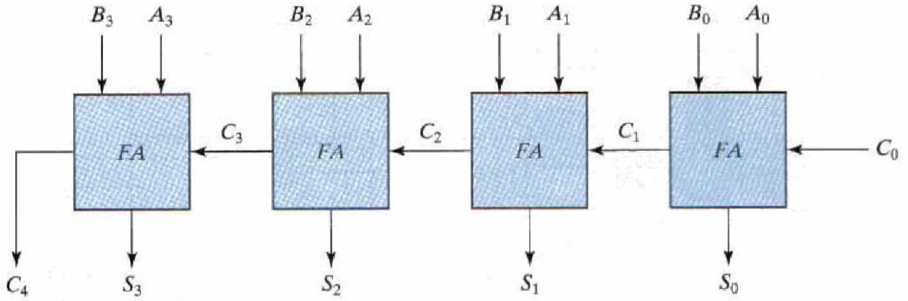

FIGURE 4.9
Four-bit adder

the circuit. By using an iterative method of cascading a standard function, it is possible to obtain a simple and straightforward implementation.

Carry Propagation

The addition of two binary numbers in parallel implies that all the bits of the augend and addend are available for computation at the same time. As in any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals. The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit. The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders. Since each bit of the sum output depends on the value of the input carry, the value of S_i at any given stage in the adder will be in its steady-state final value only after the input carry to that stage has been propagated. In this regard, consider output S_3 in Fig. 4.9. Inputs A_3 and B_3 are available as soon as input signals are applied to the adder. However, input carry C_3 does not settle to its final value until C_2 is available from the previous stage. Similarly, C_2 has to wait for C_1 and so on down to C_0. Thus, only after the carry propagates and ripples through all stages will the last output S_3 and carry C_4 settle to their final correct value.

The number of gate levels for the carry propagation can be found from the circuit of the full adder. The circuit is redrawn with different labels in Fig. 4.10 for convenience. The input and

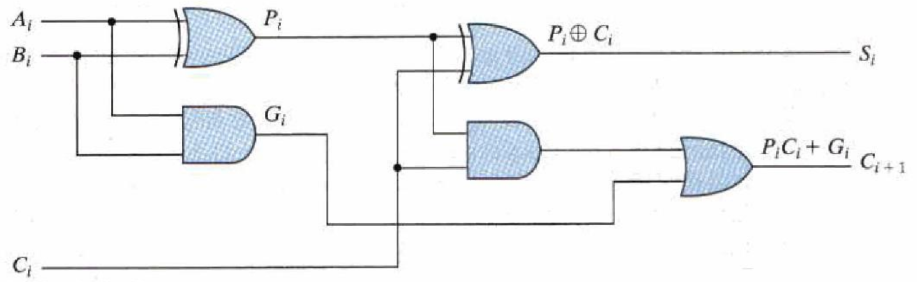

FIGURE 4.10
Full adder with *P* and *G* shown

output variables use the subscript i to denote a typical stage of the adder. The signals at P_i and G_i settle to their steady-state values after they propagate through their respective gates. These two signals are common to all full adders and depend only on the input augend and addend bits. The signal from the input carry C_i to the output carry C_{i+1} propagates through an AND gate and an OR gate, which constitute two gate levels. If there are four full adders in the adder, the output carry C_4 would have $2 \times 4 = 8$ gate levels from C_0 to C_4. For an n-bit adder, there are $2n$ gate levels for the carry to propagate from input to output.

The carry propagation time is an important attribute of the adder because it limits the speed with which two numbers are added. Although the adder—or, for that matter, any combinational circuit—will always have some value at its output terminals, the outputs will not be correct unless the signals are given enough time to propagate through the gates connected from the inputs to the outputs. Since all other arithmetic operations are implemented by successive additions, the time consumed during the addition process is critical. An obvious solution for reducing the carry propagation delay time is to employ faster gates with reduced delays. However, physical circuits have a limit to their capability. Another solution is to increase the complexity of the equipment in such a way that the carry delay time is reduced. There are several techniques for reducing the carry propagation time in a parallel adder. The most widely used technique employs the principle of *carry lookahead logic*.

Consider the circuit of the full adder shown in Fig. 4.10. If we define two new binary variables

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

the output sum and carry can respectively be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

G_i is called a *carry generate*, and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i. P_i is called a *carry propagate*, because it determines whether a carry into stage i will propagate into stage $i + 1$ (i.e., whether an assertion of C_i will propagate to an assertion of C_{i+1}).

We now write the Boolean functions for the carry outputs of each stage and substitute the value of each C_i from the previous equations:

$$\begin{aligned}
C_0 &= \text{input carry} \\
C_1 &= G_0 + P_0 C_0 \\
C_2 &= G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0 \\
C_3 &= G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 = P_2 P_1 P_0 C_0
\end{aligned}$$

Since the Boolean function for each output carry is expressed in sum-of-products form, each function can be implemented with one level of AND gates followed by an OR gate (or by a two-level NAND). The three Boolean functions for C_1, C_2, and C_3 are implemented in the carry lookahead generator shown in Fig. 4.11. Note that this circuit can add in less time because C_3 does not have to wait for C_2 and C_1 to propagate; in fact, C_3 is propagated at the same time as C_1 and C_2. This gain in speed of operation is achieved at the expense of additional complexity (hardware).

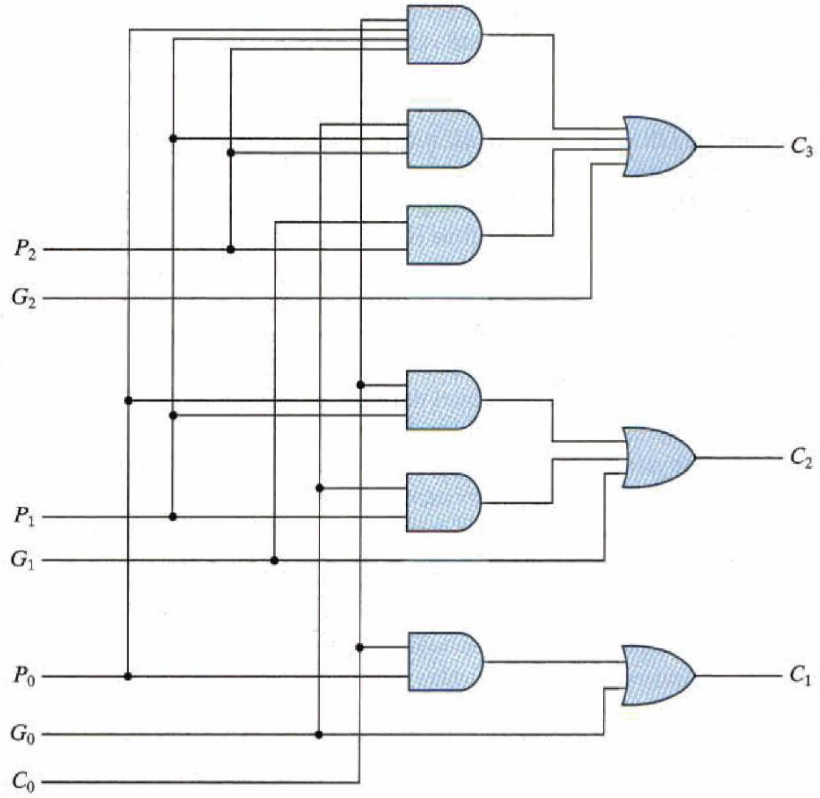

FIGURE 4.11
Logic diagram of carry lookahead generator

The construction of a four-bit adder with a carry lookahead scheme is shown in Fig. 4.12. Each sum output requires two exclusive-OR gates. The output of the first exclusive-OR gate generates the P_i variable, and the AND gate generates the G_i variable. The carries are propagated through the carry lookahead generator (similar to that in Fig. 4.11) and applied as inputs to the second exclusive-OR gate. All output carries are generated after a delay through two levels of gates. Thus, outputs S_1 through S_3 have equal propagation delay times. The two-level circuit for the output carry C_4 is not shown. This circuit can easily be derived by the equation-substitution method.

Binary Subtractor

The subtraction of unsigned binary numbers can be done most conveniently by means of complements, as discussed in Section 1.5. Remember that the subtraction $A - B$ can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.

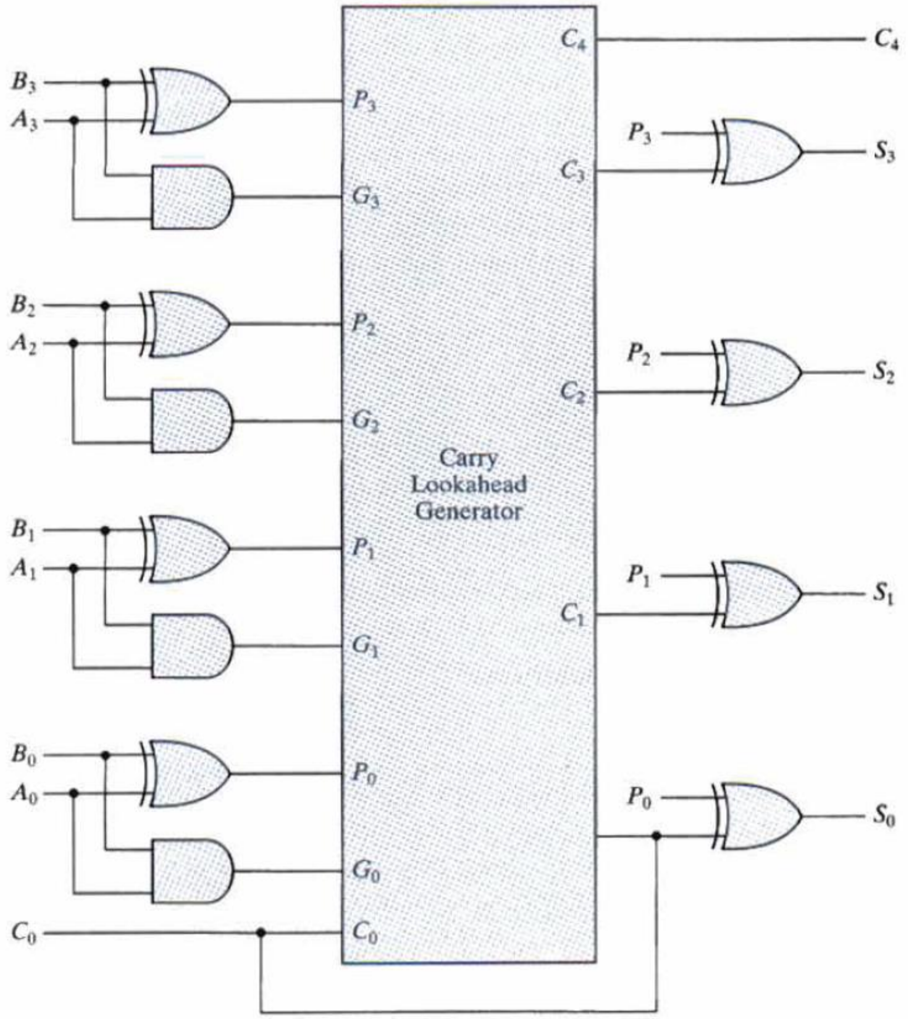

FIGURE 4.12
Four-bit adder with carry lookahead

The circuit for subtracting $A - B$ consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The input carry C_0 must be equal to 1 when subtraction is performed. The operation thus performed becomes A, plus the 1's complement of B, plus 1. This is equal to A plus the 2's complement of B. For unsigned numbers, that gives $A - B$ if $A \geq B$ or the 2's complement of $(B - A)$ if $A < B$. For signed numbers, the result is $A - B$, provided that there is no overflow. (See Section 1.6.)

The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full adder. A four-bit adder–subtractor circuit is shown in Fig. 4.13. The mode input M controls the operation. When $M = 0$, the circuit is an adder, and when $M = 1$, the circuit becomes a subtractor. Each exclusive-OR gate receives input M and one of the inputs of B. When $M = 0$, we have $B \oplus 0 = B$. The full adders receive the value of B, the input carry is 0, and the circuit performs A plus B. When $M = 1$,

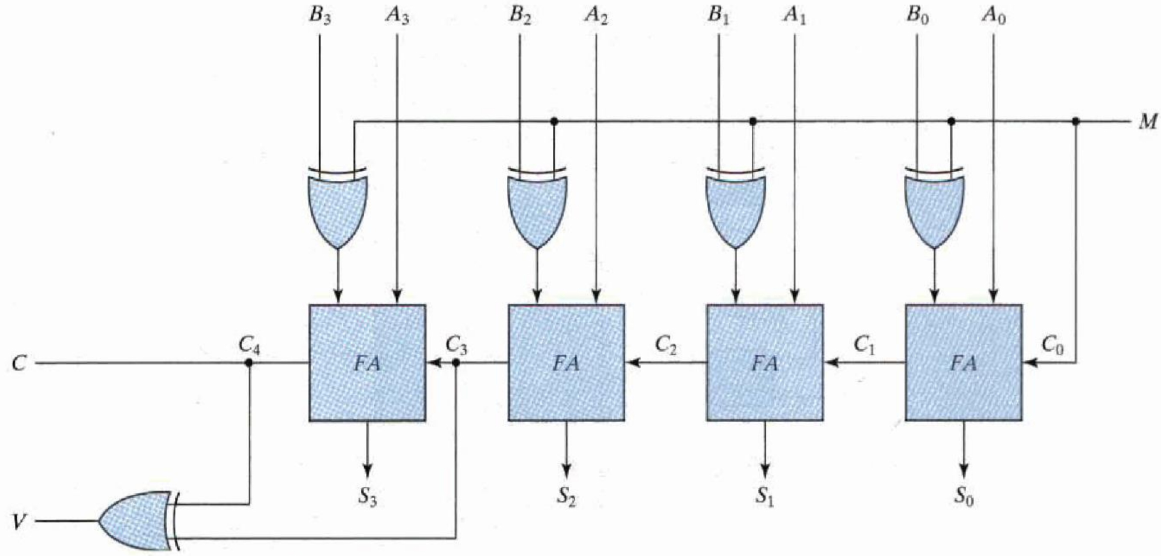

FIGURE 4.13
Four-bit adder–subtractor

we have $B \oplus 1 = B'$ and $C_0 = 1$. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B. (The exclusive-OR with output V is for detecting an overflow.)

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 are 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.

Overflow

When two numbers with n digits each are added and the sum is a number occupying $n + 1$ digits, we say that an overflow occurred. This is true for binary or decimal numbers, signed or unsigned. When the addition is performed with paper and pencil, an overflow is not a problem, since there is no limit by the width of the page to write down the sum. Overflow is a problem in digital computers because the number of bits that hold the number is finite and a result that contains $n + 1$ bits cannot be accommodated by an n-bit word. For this reason, many computers detect the occurrence of an overflow, and when it occurs, a corresponding flip-flop is set that can then be checked by the user.

The detection of an overflow after the addition of two binary numbers depends on whether the numbers are considered to be signed or unsigned. When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant position. In the case of signed numbers, two details are important: the leftmost bit always represents the sign, and negative numbers are in 2's-complement form. When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.

An overflow cannot occur after an addition if one number is positive and the other is negative, since adding a positive number to a negative number produces a result whose magnitude is smaller than the larger of the two original numbers. An overflow may occur if the two numbers added are both positive or both negative. To see how this can happen, consider the following example: Two signed binary numbers, +70 and +80, are stored in two eight-bit registers. The range of numbers that each register can accommodate is from binary +127 to binary −128. Since the sum of the two numbers is +150, it exceeds the capacity of an eight-bit register. This is also true for −70 and −80. The two additions in binary are shown next, together with the last two carries:

carries:	0 1	carries:	1 0
+70	0 1000110	−70	1 0111010
+80	0 1010000	−80	1 0110000
+150	1 0010110	−150	0 1101010

Note that the eight-bit result that should have been positive has a negative sign bit (i.e., the 8-th bit) and the eight-bit result that should have been negative has a positive sign bit. If, however, the carry out of the sign bit position is taken as the sign bit of the result, then the nine-bit answer so obtained will be correct. But since the answer cannot be accommodated within eight bits, we say that an overflow has occurred.

An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position. If these two carries are not equal, an overflow has occurred. This is indicated in the examples in which the two carries are explicitly shown. If the two carries are applied to an exclusive-OR gate, an overflow is detected when the output of the gate is equal to 1. For this method to work correctly, the 2's complement of a negative number must be computed by taking the 1's complement and adding 1. This takes care of the condition when the maximum negative number is complemented.

The binary adder–subtractor circuit with outputs C and V is shown in Fig. 4.13. If the two binary numbers are considered to be unsigned, then the C bit detects a carry after addition or a borrow after subtraction. If the numbers are considered to be signed, then the V bit detects an overflow. If $V = 0$ after an addition or subtraction, then no overflow occurred and the n-bit result is correct. If $V = 1$, then the result of the operation contains $n + 1$ bits, but only the rightmost n bits of the number fit in the space available, so an overflow has occurred. The $(n + 1)$th bit is the actual sign and has been shifted out of position.