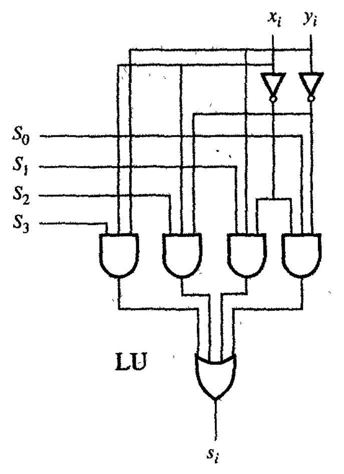
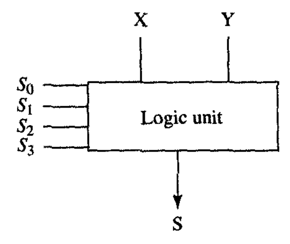


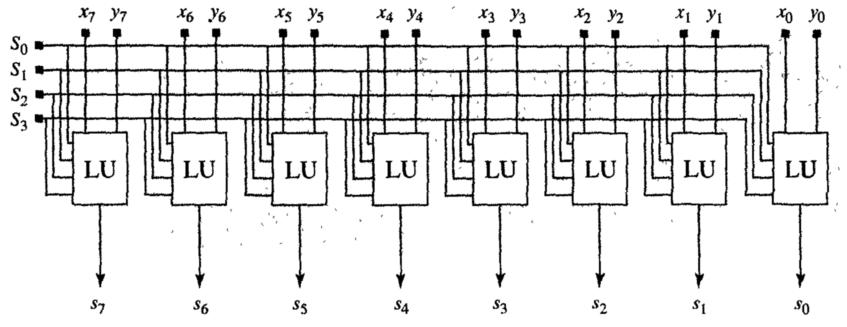
(b) Logic-unit implementation



(c) Graphic symbol

	x_i	y_i	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	
m_0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	S_0
m_1	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	S_1
m_2	1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	S_2
m_3	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	S_3

(a) Boolean functions of two variables



(d) 8-bit logic unit

FIGURE 5.5
Logic unit.

5.5 ARITHMETIC-LOGIC UNIT

An **arithmetic-logic unit** (ALU) performs the basic arithmetic and logic operations in a microprocessor. Its arithmetic operations include, for example, addition, subtraction, increment, and decrement, and its logic operations include AND, OR, identity, and complement operations. Since all the arithmetic operations are based on addition, we can design an ALU simply by modifying the inputs of a ripple-carry or CLA adder. The modifying logic used for arithmetic operations is sometimes called an **arithmetic extender** (AE), and the modifying logic used for logic operations is called a **logic extender** (LE). Either one or both of these extenders is connected to the input of the adder, as indicated by the dashed lines in Figure 5.6. We now show how to design these extenders one at a time.

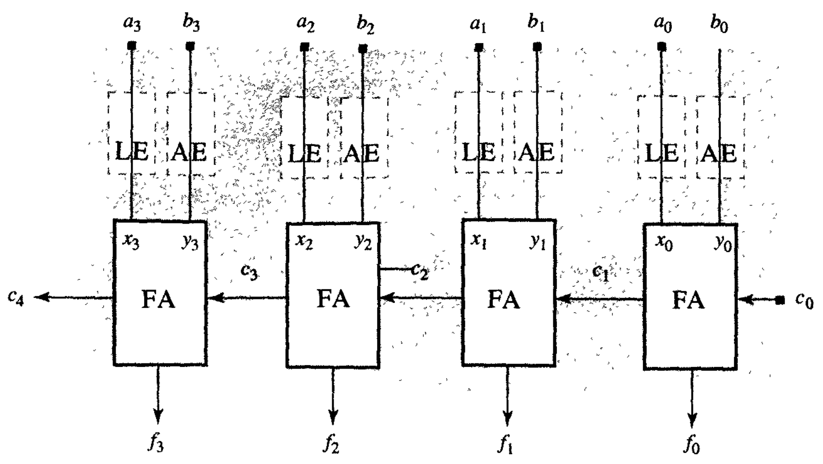


FIGURE 5.6
4-bit adder with arithmetic (AE) and logic (LE) extenders.

Since our ALU is to perform four arithmetic and four logic operations, we need to introduce a mode control variable M, which will select either arithmetic or logic operations in such a way that whenever $M = 1$, the ALU will perform arithmetic operations, and whenever $M = 0$, it will perform logic operations. We also need to use two select variables, S_1 and S_0, which will enable us to select any one of the four arithmetic or four logic operations. The values assigned to S_1 and S_0 for each arithmetic operation are summarized in the functional table in Figure 5.7(a).

As you can see, this table also shows the value of the ALU output F, as well as the values for the adder inputs X, Y, and c_0 which are required to achieve that value of F. Note that according to this table, the X input of the adder always requires the value of A, while the Y input can require all 1's, B, B', or all 0's. These values for the Y input will be generated by the AE, the truth table for which has been shown in Figure 5.7(b). This table was obtained from the functional table in Figure 5.7(a) simply by expanding column Y into columns b_i and y_i. In Figure 5.7(c) we show the map representation of the AE, from which we can see that its Boolean expression would be

$$y_i = MS_1'b_i + MS_0'b_i'$$

Finally, the logic schematic of this AE is shown in Figure 5.7(d).

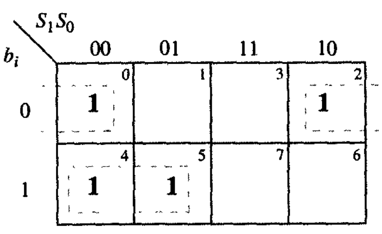
Having shown the procedure for designing an AE, we can now turn to the design of a LE, which starts with the **functional table** describing its operations, as shown in Figure 5.8(a). From this table you can see that

M	S_1	S_0	FUNCTION NAME	F	X	Y	c_0
1	0	0	Decrement	$A-1$	A	all 1's	0
1	0	1	Add	$A+B$	A	B	0
1	1	0	Subtract	$A+B'+1$	A	B'	1
1	1	1	Increment	$A+1$	A	all 0's	1

(a) Functional table

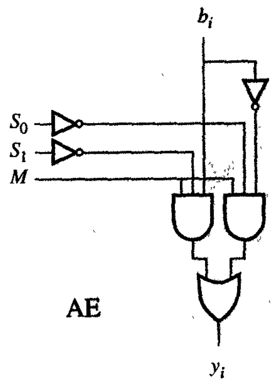
M	S_1	S_0	b_i	y_i
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

(b) Truth table



$y_i = MS_1'b_i + MS_0'b_i'$

(c) Map representation



(d) Logic schematic

FIGURE 5.7
Arithmetic extender.

the Y and c_0 inputs always require a value of 0 for any one of the logic operations, whereas the X input requires a different Boolean expression for each of these operations. On the basis of this functional table, we are able to develop a truth table for the LE, which is shown in Figure 5.8(b), and its map representation, which is presented in Figure 5.8(c). From this map representation we are able to derive the following Boolean expression to describe the LE:

$$x_i = M'S_1'S_0'a_i' + M'S_1S_0b_i + S_0a_ib_i + S_1a_i + Ma_i$$

Having obtained this expression, we then proceed to construct the logic schematic for the LE, shown in Figure 5.8(d).

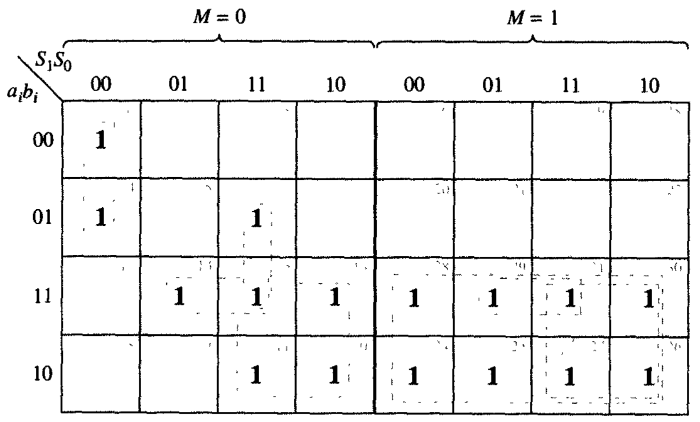
At this point we have obtained logic schematics for both the AE and the LE, and our next task is to connect them with an adder, thus forming a complete arithmetic-logic unit like the 4-bit ALU shown in Figure 5.9(a). Note that in the ALU, the logic operations are performed in the logic extender, and the FAs are used to pass the LE's results without change. In other words, the FAs are used as connections that have a fixed delay.

M	S_1	S_0	FUNCTION NAME	F	X	Y	c_0
0	0	0	Complement	A'	A'	0	0
0	0	1	AND	A AND B	A AND B	0	0
0	1	0	Identity	A	A	0	0
0	1	1	OR	A OR B	A OR B	0	0

(a) Functional table

M	S_1	S_0	x_i
0	0	0	a_i'
0	0	1	a_ib_i
0	1	0	a_i
0	1	1	a_i+b_i
1	X	X	a_i

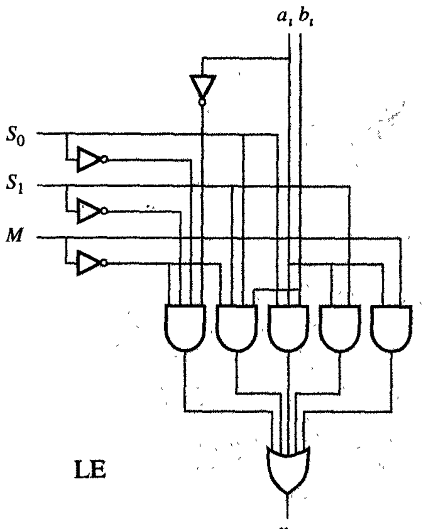
(b) Truth table



	$M=0$				$M=1$			
a_ib_i \ S_1S_0	00	01	11	10	00	01	11	10
00	1							
01	1		1					
11		1	1	1	1	1	1	1
10			1	1	1	1	1	1

$$x_i = M'S_1'S_0'a_i' + M'S_1S_0b_i + S_0a_ib_i + S_1a_i + Ma_i$$

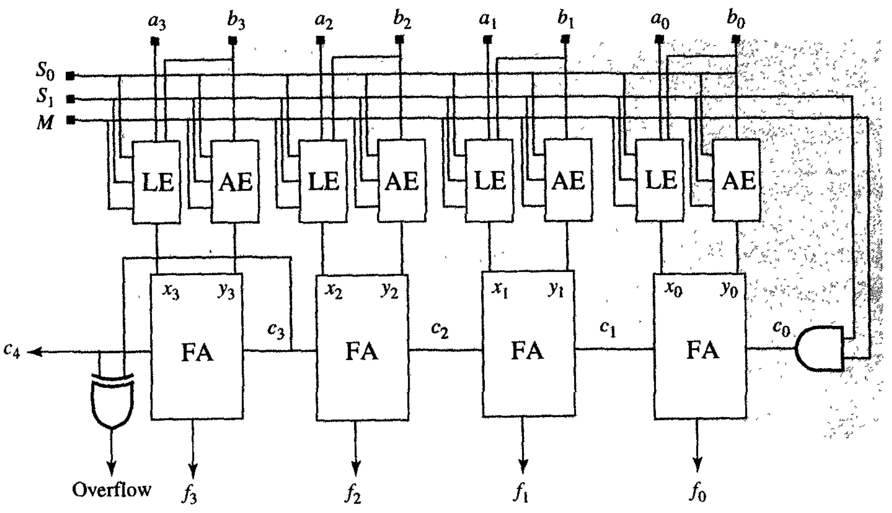
(c) Map representation



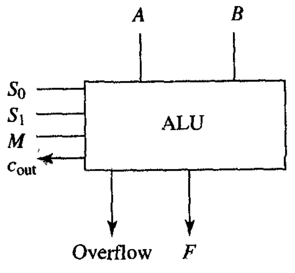
(d) Logic schematic

FIGURE 5.8
Logic extender.

Note also that the carry-out of the most significant bit represents an overflow in the case of unsigned arithmetic and that the EX-OR of the carry-outs of the two most significant bits represents the overflow in the case of 2's-complement arithmetic. If necessary, the 4-bit ALU shown in Figure 5.9(a) can be extended into an n-bit ALU, by using an n-bit adder in conjunction with n AEs and n LEs. The graphic symbol for such an ALU has been shown in Figure 5.9(b). In the industry, most ALUs used in real products are constructed in this fashion, except that they can differ in the type and number of their arithmetic and logic operations and in the implementation of their carry chains.



(a) 4-bit ALU



(b) Graphic symbol

FIGURE 5.9
Final ALU design.

5.6 DECODERS

Decoders (sometimes also called **demultiplexers**) are frequently incorporated into larger units for use whenever we need to activate or enable only one of n subcomponents. In such a case, each subcomponent can be assigned an index between 0 and $n-1$ which is represented by a binary address A. To activate a particular subcomponent at any given time, this address A is decoded into n enable lines, out of which only one line is equal to 1. In general, an m-to-n decoder has $m = \log_2 n$ input lines, $A_{m-1}, \ldots, A_0$, and n output lines,