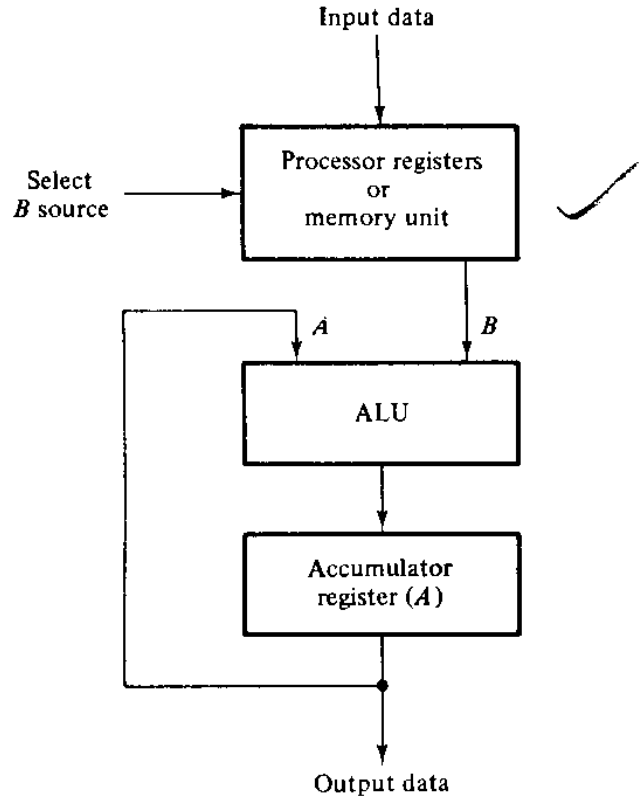



**Figure 9-4** Processor with an accumulator register

to add them in the $A$ register using the following sequence of microoperations:

| | | |
|---|---|---|
| $T_1$: | $A \leftarrow 0$ | clear $A$ |
| $T_2$: | $A \leftarrow A + R1$ | transfer $R1$ to $A$ |
| $T_3$: | $A \leftarrow A + R2$ | add $R2$ to $A$ |

Register $A$ is first cleared. The first number in $R1$ is transferred into the $A$ register by adding it to the present zero content of $A$. The second number in $R2$ is then added to the present value of $A$. The sum formed in $A$ may be used for other computations or may be transferred to a required destination.

## 9-3 ARITHMETIC LOGIC UNIT

An arithmetic logic unit (ALU) is a multioperation, combinational-logic digital function. It can perform a set of basic arithmetic operations and a set of logic operations. The ALU has a number of selection lines to select a particular operation in the unit. The selection lines are decoded within the ALU so that $k$ selection variables can specify up to $2^k$ distinct operations.

Figure 9-5 shows the block diagram of a 4-bit ALU. The four data inputs from $A$ are combined with the four inputs from $B$ to generate an operation at the $F$

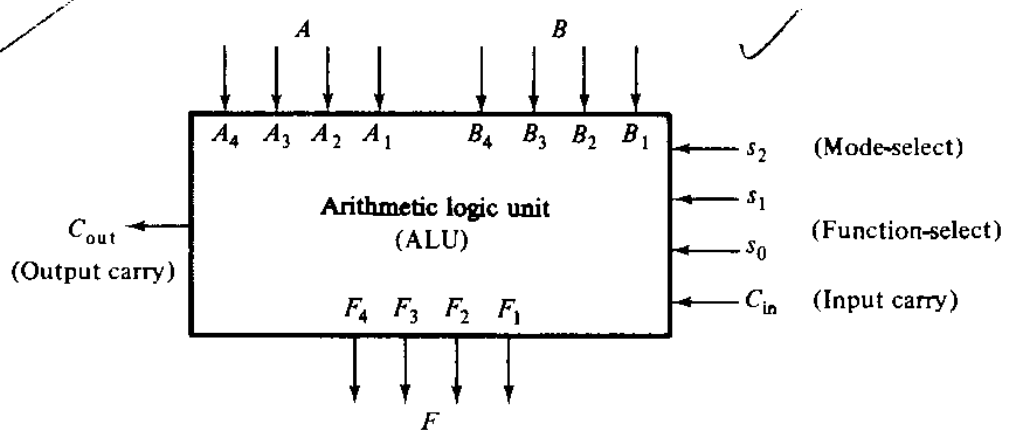



**Figure 9-5** Block diagram of a 4-bit ALU

outputs. The mode-select input $s_2$ distinguishes between arithmetic and logic operations. The two function-select inputs $s_1$ and $s_0$ specify the particular arithmetic or logic operation to be generated. With three selection variables, it is possible to specify four arithmetic operations (with $s_2$ in one state) and four logic operations (with $s_2$ in the other state). The input and output carries have meaning only during an arithmetic operation.

The input carry in the least significant position of an ALU is quite often used as a fourth selection variable that can double the number of arithmetic operations. In this way, it is possible to generate four more operations, for a total of eight arithmetic operations.

The design of a typical ALU will be carried out in three stages. First, the design of the arithmetic section will be undertaken. Second, the design of the logic section will be considered. Finally, the arithmetic section will be modified so that it can perform both arithmetic and logic operations.

## 9-4 DESIGN OF ARITHMETIC CIRCUIT

The basic component of the arithmetic section of an ALU is a parallel adder. A parallel adder is constructed with a number of full-adder circuits connected in cascade (see Section 5-2). By controlling the data inputs to the parallel adder, it is possible to obtain different types of arithmetic operations. Figure 9-6 demonstrates the arithmetic operations obtained when one set of inputs to a parallel adder is controlled externally. The number of bits in the parallel adder may be of any value. The input carry $C_{in}$ goes to the full-adder circuit in the least significant bit position. The output carry $C_{out}$ comes from the full-adder circuit in the most significant bit position.

The arithmetic addition is achieved when one set of inputs receives a binary number $A$, the other set of inputs receives a binary number $B$, and the input carry is maintained at 0. This is shown in Fig. 9-6(a). By making $C_{in} = 1$ as in Fig.

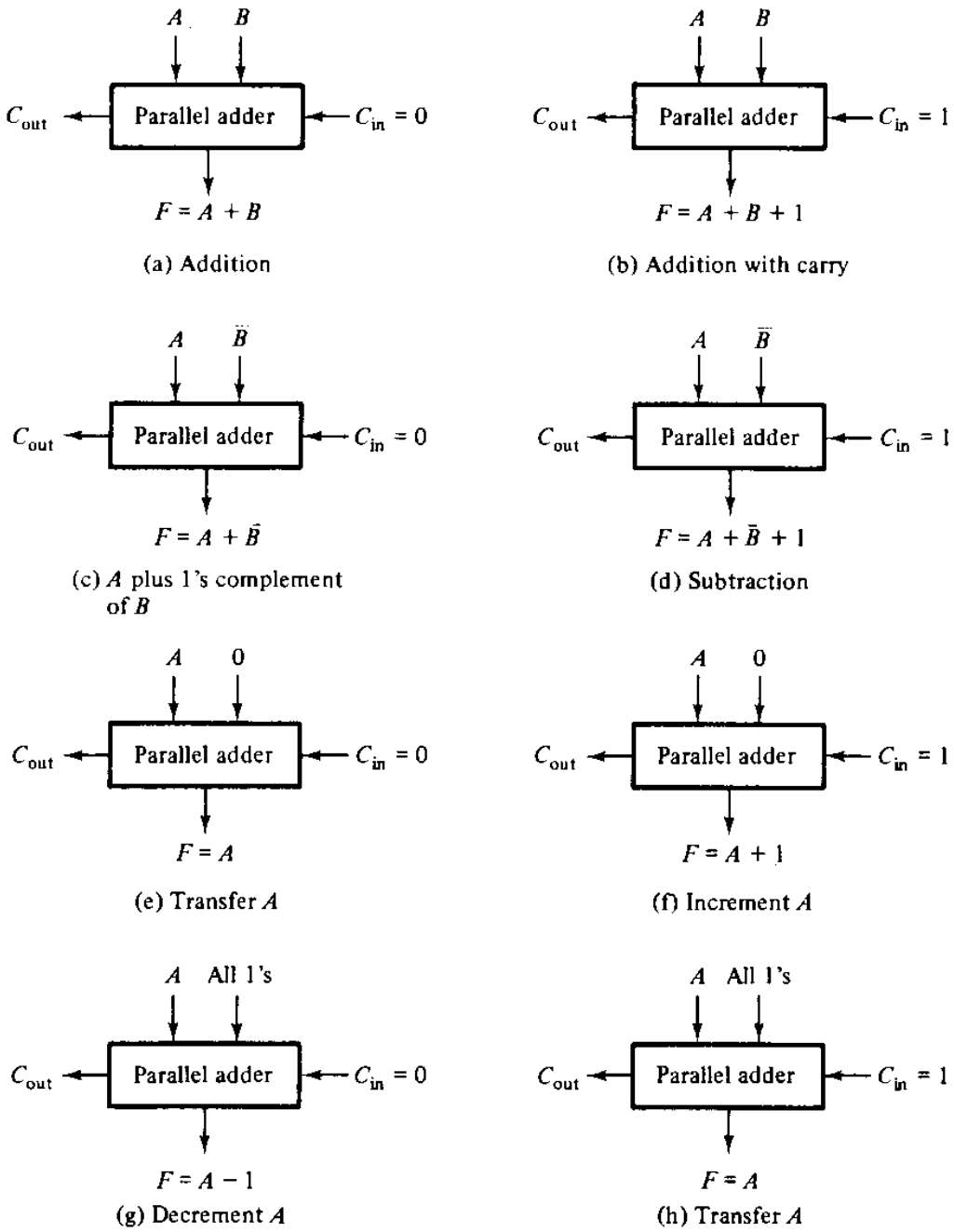



**Figure 9-6** Operations obtained by controlling one set of inputs to a parallel adder

9-6(b), it is possible to add 1 to the sum in $F$. Now consider the effect of complementing all the bits of input $B$. With $C_{in} = 0$, the output produces $F = A + \overline{B}$, which is the sum of $A$ plus the 1's complement of $B$. Adding 1 to this sum by making $C_{in} = 1$, we obtain $F = A + \overline{B} + 1$, which produces the sum of $A$ plus the 2's complement of $B$. This operation is similar to a subtraction operation if the output carry is discarded. If we force all 0's into the $B$ terminals, we obtain

$F = A + 0 = A$, which transfers input $A$ into output $F$. Adding 1 through $C_{in}$ as in Fig. 9-6(f), we obtain $F = A + 1$, which is the increment operation.

The condition illustrated in Fig. 9-6(g) inserts all 1's into the $B$ terminals. This produces the decrement operation $F = A - 1$. To show that this condition is indeed a decrement operation, consider a parallel adder with $n$ full-adder circuits. When the output carry is 1, it represents the number $2^n$ because $2^n$ in binary consists of a 1 followed by $n$ 0's. Subtracting 1 from $2^n$, we obtain $2^n - 1$, which in binary is a number of $n$ 1's. Adding $2^n - 1$ to $A$, we obtain $F = A + 2^n - 1 = 2^n + A - 1$. If the output carry $2^n$ is removed, we obtain $F = A - 1$.

To demonstrate with a numerical example, let $n = 8$ and $A = 9$. Then:

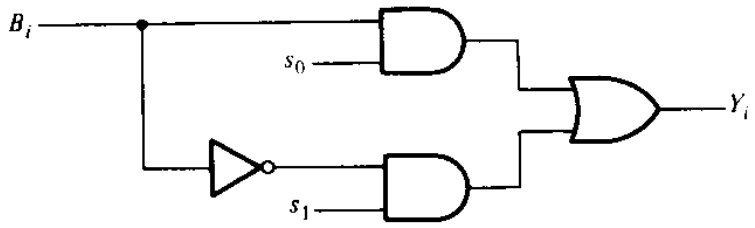
$$\begin{aligned} A &= \quad 0000\ 1001 = (9)_{10} \\ 2^n &= 1\ 0000\ 0000 = (256)_{10} \\ 2^n - 1 &= \quad 1111\ 1111 = (255)_{10} \\ A + 2^n - 1 &= 1\ 0000\ 1000 = (256 + 8)_{10} \end{aligned}$$

Removing the output carry $2^n = 256$, we obtain $8 = 9 - 1$. Thus, we have decremented $A$ by 1 by adding to it a binary number with all 1's.

The circuit that controls input $B$ to provide the functions illustrated in Fig. 9-6 is called a *true/complement, one/zero* element. This circuit is illustrated in Fig. 9-7. The two selection lines $s_1$ and $s_0$ control the input of each $B$ terminal. The diagram shows one typical input designated by $B_i$ and an output designated by $Y_i$. In a typical application, there are $n$ such circuits for $i = 1, 2, \ldots, n$. As shown in the table of Fig. 9-7, when both $s_1$ and $s_0$ are equal to 0, the output $Y_i = 0$, regardless of the value of $B_i$. When $s_1s_0 = 01$, the top AND gate generates the value of $B_i$ while the bottom gate output is 0; so $Y_i = B_i$. With $s_1s_0 = 10$, the bottom AND gate generates the complement of $B_i$ to give $Y_i = B_i'$. When $s_1s_0 = 11$, both gates are active and $Y_i = B_i + B_i' = 1$.

A 4-bit arithmetic circuit that performs eight arithmetic operations is shown in Fig. 9-8. The four full-adder (FA) circuits constitute the parallel adder. The carry into the first stage is the input carry. The carry out of the fourth stage is the output carry. All other carries are connected internally from one stage to the next. The selection variables are $s_1$, $s_0$, and $C_{in}$. Variables $s_1$ and $s_0$ control all of the $B$ inputs to the full-adder circuits as in Fig. 9-7. The $A$ inputs go directly to the other inputs of the full adders.

The arithmetic operations implemented in the arithmetic circuit are listed in Table 9-1. The values of the $Y$ inputs to the full-adder circuits are a function of




| $s_1$ | $s_0$ | $Y_i$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | $B_i$ |
| 1 | 0 | $B_i'$ |
| 1 | 1 | 1 |

**Figure 9-7** True/complement, one/zero circuit

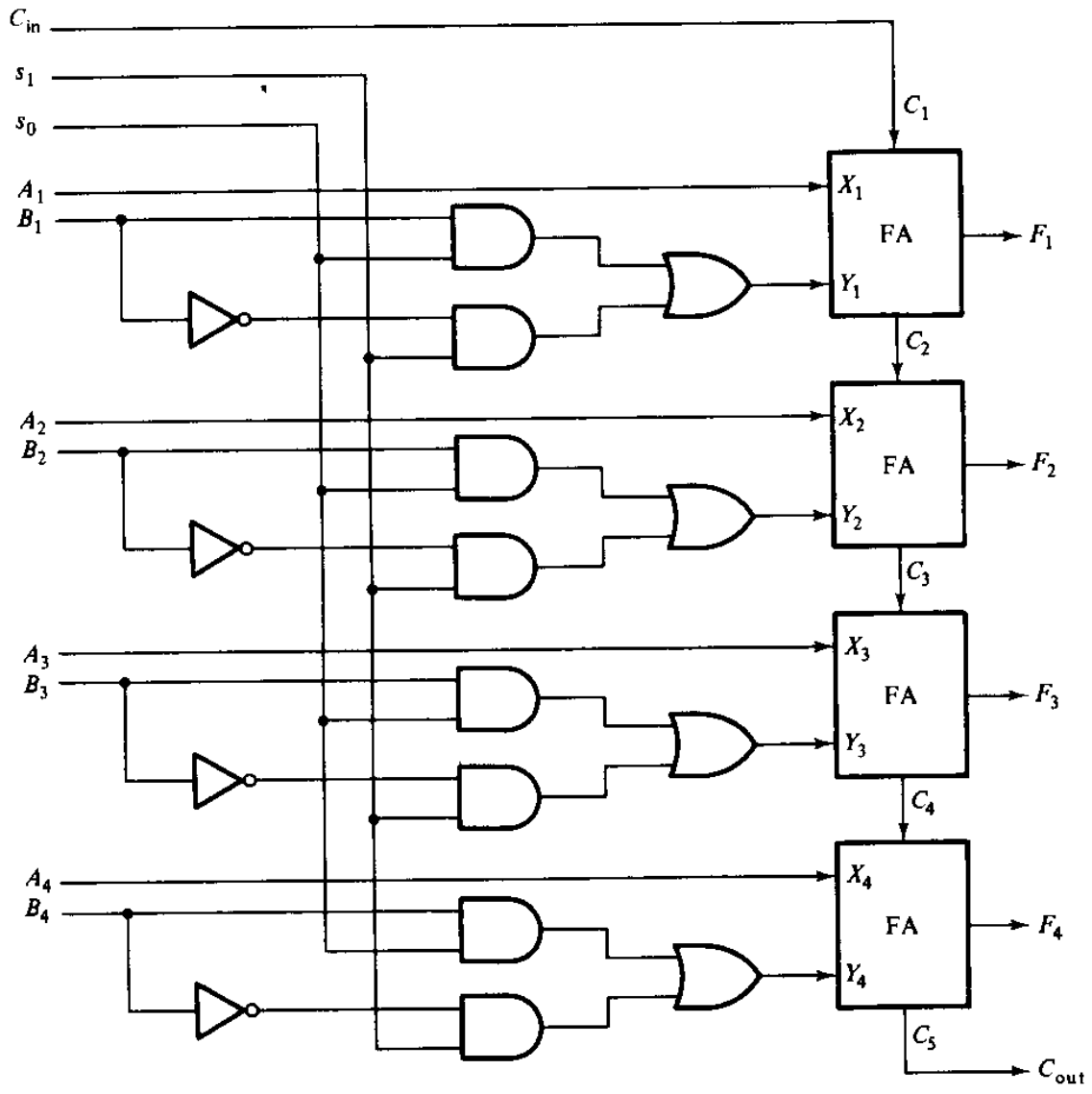



**Figure 9-8** Logic diagram of arithmetic circuit

**TABLE 9-1** Function table for the arithmetic circuit of Fig. 9-8

| Function select | | | Y equals | Output equals | Function |
|---|---|---|---|---|---|
| $s_1$ | $s_0$ | $C_{in}$ | | | |
| 0 | 0 | 0 | 0 | $F = A$ | Transfer $A$ |
| 0 | 0 | 1 | 0 | $F = A + 1$ | Increment $A$ |
| 0 | 1 | 0 | $B$ | $F = A + B$ | Add $B$ to $A$ |
| 0 | 1 | 1 | $B$ | $F = A + B + 1$ | Add $B$ to $A$ plus 1 |
| 1 | 0 | 0 | $\overline{B}$ | $F = A + \overline{B}$ | Add 1's complement of $B$ to $A$ |
| 1 | 0 | 1 | $\overline{B}$ | $F = A + \overline{B} + 1$ | Add 2's complement of $B$ to $A$ |
| 1 | 1 | 0 | All 1's | $F = A - 1$ | Decrement $A$ |
| 1 | 1 | 1 | All 1's | $F = A$ | Transfer $A$ |

selection variables $s_1$ and $s_0$. Adding the value of $Y$ in each case to the value of $A$ plus the $C_{in}$ value gives the arithmetic operation in each entry. The eight operations listed in the table follow directly from the function diagrams illustrated in Fig. 9-6.

This example demonstrates the feasibility of constructing an arithmetic circuit by means of a parallel adder. The combinational circuit that must be inserted in each stage between the external inputs $A_i$ and $B_i$ and the inputs of the parallel adder $X_i$ and $Y_i$ is a function of the arithmetic operations that are to be implemented. The arithmetic circuit of Fig. 9-8 needs a combinational circuit in each stage specified by the Boolean functions:

$$X_i = A_i$$
$$Y_i = B_i s_0 + B_i' s_1 \qquad i = 1, 2, \ldots, n$$

where $n$ is the number of bits in the arithmetic circuit. In each stage $i$, we use the same common selection variables $s_1$ and $s_0$. The combinational circuit will be different if the circuit generates different arithmetic operations.

## Effect of Output Carry

The output carry of an arithmetic circuit or ALU has special significance, especially after a subtraction operation. To investigate the effect of the output carry, we expand the arithmetic circuit of Fig. 9-8 to $n$ bits so that $C_{out} = 1$ when the output of the circuit is equal to or greater than $2^n$. Table 9-2 lists the conditions for having an output carry in the circuit. The function $F = A$ will always have the output carry equal to 0. The same applies to the increment operation $F = A + 1$, except when it goes from an all-1's condition to an all-0's condition, at which time

**TABLE 9-2** Effect of output carry in the arithmetic circuit of Fig. 9-8

| Function select | | | Arithmetic function | $C_{out} = 1$ if | Comments |
|---|---|---|---|---|---|
| $s_1$ | $s_0$ | $C_{in}$ | | | |
| 0 | 0 | 0 | $F = A$ | | $C_{out}$ is always 0 |
| 0 | 0 | 1 | $F = A + 1$ | $A = 2^n - 1$ | $C_{out} = 1$ and $F = 0$ if $A = 2^n - 1$ |
| 0 | 1 | 0 | $F = A + B$ | $(A + B) > 2^n$ | Overflow occurs if $C_{out} = 1$ |
| 0 | 1 | 1 | $F = A + B + 1$ | $(A + B) > (2^n - 1)$ | Overflow occurs if $C_{out} = 1$ |
| 1 | 0 | 0 | $F = A - B - 1$ | $A > B$ | If $C_{out} = 0$, then $A < B$ and $F$ = 1's complement of $(B - A)$ |
| 1 | 0 | 1 | $F = A - B$ | $A \geq B$ | If $C_{out} = 0$, then $A < B$ and $F$ = 2's complement of $(B - A)$ |
| 1 | 1 | 0 | $F = A - 1$ | $A \neq 0$ | $C_{out} = 1$, except when $A = 0$ |
| 1 | 1 | 1 | $F = A$ | | $C_{out}$ is always 1 |

selection variables $s_1$ and $s_0$. Adding the value of $Y$ in each case to the value of $A$ plus the $C_{in}$ value gives the arithmetic operation in each entry. The eight operations listed in the table follow directly from the function diagrams illustrated in Fig. 9-6.

This example demonstrates the feasibility of constructing an arithmetic circuit by means of a parallel adder. The combinational circuit that must be inserted in each stage between the external inputs $A_i$ and $B_i$ and the inputs of the parallel adder $X_i$ and $Y_i$ is a function of the arithmetic operations that are to be implemented. The arithmetic circuit of Fig. 9-8 needs a combinational circuit in each stage specified by the Boolean functions:

$$X_i = A_i$$
$$Y_i = B_i s_0 + B_i' s_1 \qquad i = 1, 2, \ldots, n$$

where $n$ is the number of bits in the arithmetic circuit. In each stage $i$, we use the same common selection variables $s_1$ and $s_0$. The combinational circuit will be different if the circuit generates different arithmetic operations.

## Effect of Output Carry

The output carry of an arithmetic circuit or ALU has special significance, especially after a subtraction operation. To investigate the effect of the output carry, we expand the arithmetic circuit of Fig. 9-8 to $n$ bits so that $C_{out} = 1$ when the output of the circuit is equal to or greater than $2^n$. Table 9-2 lists the conditions for having an output carry in the circuit. The function $F = A$ will always have the output carry equal to 0. The same applies to the increment operation $F = A + 1$, except when it goes from an all-1's condition to an all-0's condition, at which time

**TABLE 9-2** Effect of output carry in the arithmetic circuit of Fig. 9-8

| Function select | | | Arithmetic function | $C_{out} = 1$ if | Comments |
|---|---|---|---|---|---|
| $s_1$ | $s_0$ | $C_{in}$ | | | |
| 0 | 0 | 0 | $F = A$ | | $C_{out}$ is always 0 |
| 0 | 0 | 1 | $F = A + 1$ | $A = 2^n - 1$ | $C_{out} = 1$ and $F = 0$ if $A = 2^n - 1$ |
| 0 | 1 | 0 | $F = A + B$ | $(A + B) \geq 2^n$ | Overflow occurs if $C_{out} = 1$ |
| 0 | 1 | 1 | $F = A + B + 1$ | $(A + B) \geq (2^n - 1)$ | Overflow occurs if $C_{out} = 1$ |
| 1 | 0 | 0 | $F = A - B - 1$ | $A > B$ | If $C_{out} = 0$, then $A \leq B$ and $F$ = 1's complement of $(B - A)$ |
| 1 | 0 | 1 | $F = A - B$ | $A \geq B$ | If $C_{out} = 0$, then $A < B$ and $F$ = 2's complement of $(B - A)$ |
| 1 | 1 | 0 | $F = A - 1$ | $A \neq 0$ | $C_{out} = 1$, except when $A = 0$ |
| 1 | 1 | 1 | $F = A$ | | $C_{out}$ is always 1 |

it produces an output carry of 1. An output carry of 1 after an addition operation denotes an overflow condition. It indicates that the sum is greater than or equal to $2^n$ and that the sum consists of $n + 1$ bits.

The operation $F = A + \bar{B}$ adds the 1's complement of $B$ to $A$. Remember from Section 1-5 that the complement of $B$ can be expressed arithmetically as $2^n - 1 - B$. The arithmetic result in the output will be:

$$F = A + 2^n - 1 - B = 2^n + A - B - 1$$

If $A > B$, then $(A - B) > 0$ and $F > (2^n - 1)$, so that $C_{out} = 1$. Removing the output carry $2^n$ from this result gives:

$$F = A - B - 1$$

which is a subtraction with borrow. Note that if $A \leqslant B$, then $(A - B) \leqslant 0$ and $F \leqslant (2^n - 1)$, so that $C_{out} = 0$. For this condition it is more convenient to express the arithmetic result as:

$$F = (2^n - 1) - (B - A)$$

which is the 1's complement of $B - A$.

The condition for output carry when $F = A + \bar{B} + 1$ can be derived in a similar manner. $\bar{B} + 1$ is the symbol for the 2's complement of $B$. Arithmetically, this is an operation that produces a number equal to $2^n - B$. The result of the operation can be expressed as:

$$F = A + 2^n - B = 2^n + A - B$$

If $A \geqslant B$, then $(A - B) \geqslant 0$ and $F \geqslant 2^n$, so that $C_{out} = 1$. Removing the output carry $2^n$, we obtain:

$$F = A - B$$

which is a subtraction operation. If, however, $A < B$, then $(A - B) < 0$ and $F < 2^n$, so that $C_{out} = 0$. The arithmetic result for this condition can be expressed as:

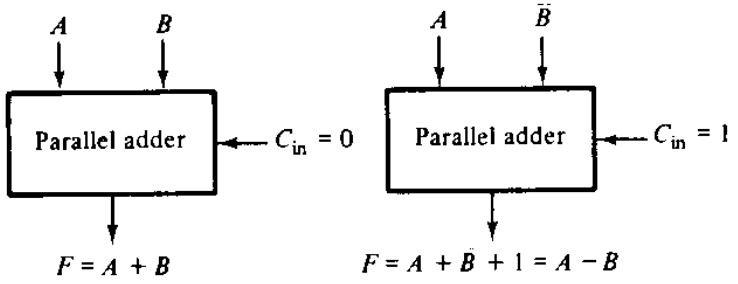
$$F = 2^n - (B - A)$$

which is the 2's complement of $B - A$. Thus, the output of the arithmetic subtraction is correct as long as $A \geqslant B$. The output should be $B - A$ if $B > A$, but the circuit generates the 2's complement of this number.

The decrement operation is obtained from $F = A + (2^n - 1) = 2^n + A - 1$. The output carry is always 1 except when $A = 0$. Subtracting 1 from 0 gives $-1$, and $-1$ in 2's complement is $2^n - 1$, which is a number with all 1's. The last entry in Table 9-2 generates $F = (2^n - 1) + A + 1 = 2^n + A$. This operation transfers $A$ into $F$ and gives an output carry of 1.

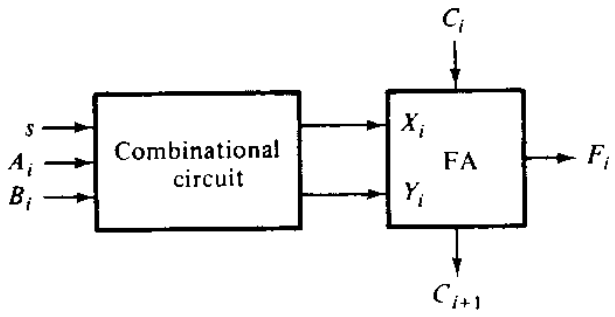
## Design of Other Arithmetic Circuits

The design of any arithmetic circuit that generates a set of basic operations can be undertaken by following the procedure outlined in the previous example. Assuming that all operations in the set can be generated through a parallel adder, we start by obtaining a function diagram as in Fig. 9-6. From the function diagram, we obtain a function table that relates the inputs of the full-adder circuit to the external inputs. From the function table, we obtain the combinational gates that must be added to each full-adder stage. This procedure is demonstrated in the following example.




(a) Function specification

| $s$ | $X_i$ | $Y_i$ | $C_{in}$ |
|---|---|---|---|
| 0 | $A_i$ | $B_i$ | 0 |
| 1 | $A_i$ | $B_i'$ | 1 |




(b) Specifying combinational circuit

| $s$ | $A_i$ | $B_i$ | $X_i$ | $Y_i$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

$X_i = A_i$

$Y_i = B_i \oplus s$

$C_{in} = s$

(c) Truth table and simplified equations

**Figure 9-9** Derivation of an adder/subtractor circuit

***EXAMPLE 9-1:*** Design an adder/subtractor circuit with one selection variable $s$ and two inputs $A$ and $B$. When $s = 0$ the circuit performs $A + B$. When $s = 1$ the circuit performs $A - B$ by taking the 2's complement of $B$.

The derivation of the arithmetic circuit is illustrated in Fig. 9-9. The function diagram is shown in Fig. 9-9(a). For the addition part, we need $C_{in} = 0$. For the subtraction part, we need the complement of $B$ and $C_{in} = 1$. The function table is listed in Fig. 9-9(b). When $s = 0$, $X_i$ and $Y_i$ of each full adder must be equal to the external inputs $A_i$ and $B_i$, respectively. When $s = 1$, we must have $X_i = A_i$ and $Y_i = B_i'$. The input carry must be equal to the value of $s$. The diagram in (b) shows the position of the combinational circuit in one typical stage of the arithmetic circuit. The truth table in (c) is obtained by listing the eight values of the binary input variables. Output $X_i$ is made to be equal to input $A_i$ in all eight entries. Output $Y_i$ is equal to $B_i$ for the four entries when $s = 0$. It is equal to the

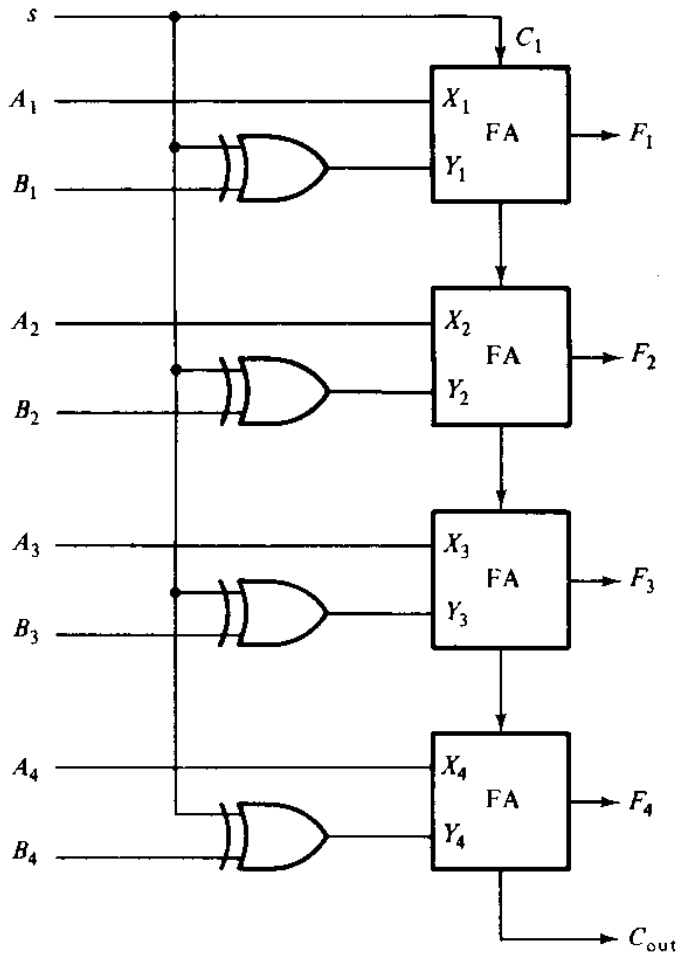



**Figure 9-10** 4-bit adder/subtractor circuit

complement of $B_i$ for the last four entries where $s = 1$. The simplified output functions for the combinational circuit are:

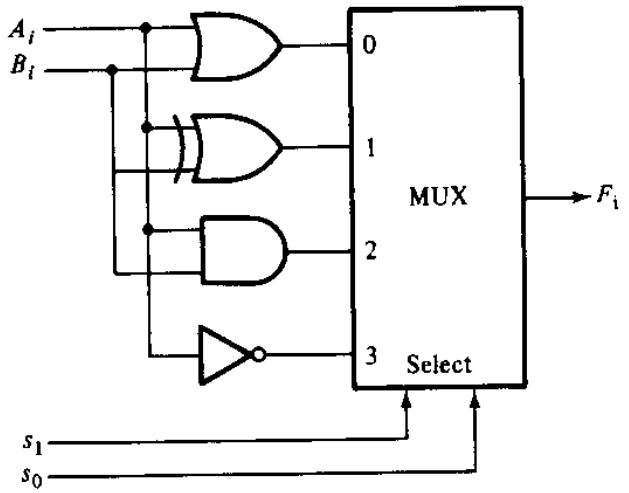
$$X_i = A_i$$
$$Y_i = B_i \oplus s$$

The diagram of the 4-bit adder/subtractor circuit is shown in Fig. 9-10. Each input $B_i$ requires an exclusive-OR gate. The selection variable $s$ goes to one input of each gate and also to the input carry of the parallel adder. The 4-bit adder/subtractor can be constructed with two ICs. One IC is the 4-bit parallel adder and the other is a quadruple exclusive-OR gates.

## 9-5 DESIGN OF LOGIC CIRCUIT

The logic microoperations manipulate the bits of the operands separately and treat each bit as a binary variable. Table 2-6 listed 16 logic operations that can be performed with two binary variables. The 16 logic operations can be generated in one circuit and selected by means of four selection lines. Since all logic operations can be obtained by means of AND, OR, and NOT (complement) operations, it may be more convenient to employ a logic circuit with just these operations. For three operations, we need two selection variables. But two selection lines can select among four logic operations, so we choose also the exclusive-OR (XOR) function for the logic circuit to be designed in this and the next section.

The simplest and most straightforward way to design a logic circuit is shown in Fig. 9-11. The diagram shows one typical stage designated by subscript $i$. The circuit must be repeated $n$ times for an $n$-bit logic circuit. The four gates generate




(a) Logic diagram

| $s_1$ | $s_0$ | Output | Operation |
|---|---|---|---|
| 0 | 0 | $F_i = A_i + B_i$ | OR |
| 0 | 1 | $F_i = A_i \oplus B_i$ | XOR |
| 1 | 0 | $F_i = A_i B_i$ | AND |
| 1 | 1 | $F_i = A_i'$ | NOT |

(b) Function table

**Figure 9-11** One stage of logic circuit

the four logic operations OR, XOR, AND, and NOT. The two selection variables in the multiplexer select one of the gates for the output. The function table lists the output logic generated as a function of the two selection variables.

The logic circuit can be combined with the arithmetic circuit to produce one arithmetic logic unit. Selection variables $s_1$ and $s_0$ can be made common to both sections provided we use a third selection variable, $s_2$, to differentiate between the two. This configuration is illustrated in Fig. 9-12. The outputs of the logic and arithmetic circuits in each stage go through a multiplexer with selection variable $s_2$. When $s_2 = 0$, the arithmetic output is selected, but when $s_2 = 1$, the logic output is selected. Although the two circuits can be combined in this manner, this is not the best way to design an ALU.

A more efficient ALU can be obtained if we investigate the possibility of generating logic operations in an already available arithmetic circuit. This can be done by inhibiting all input carries into the full-adder circuits of the parallel adder. Consider the Boolean function that generates the output sum in a full-adder circuit:

$$F_i = X_i \oplus Y_i \oplus C_i$$

The input carry $C_i$ in each stage can be made to be equal to 0 when a selection variable $s_2$ is equal to 1. The result would be:

$$F_i = X_i \oplus Y_i$$

This expression is valid because of the property of the exclusive-OR operation $x \oplus 0 = x$. Thus, with the input carry to *each* stage equal to 0, the full-adder circuits generate the exclusive-OR operation.

Now consider the arithmetic circuit of Fig. 9-8. The value of $Y_i$ can be selected by means of the two selection variables to be equal to either 0, $B_i$, $B_i'$, or 1.

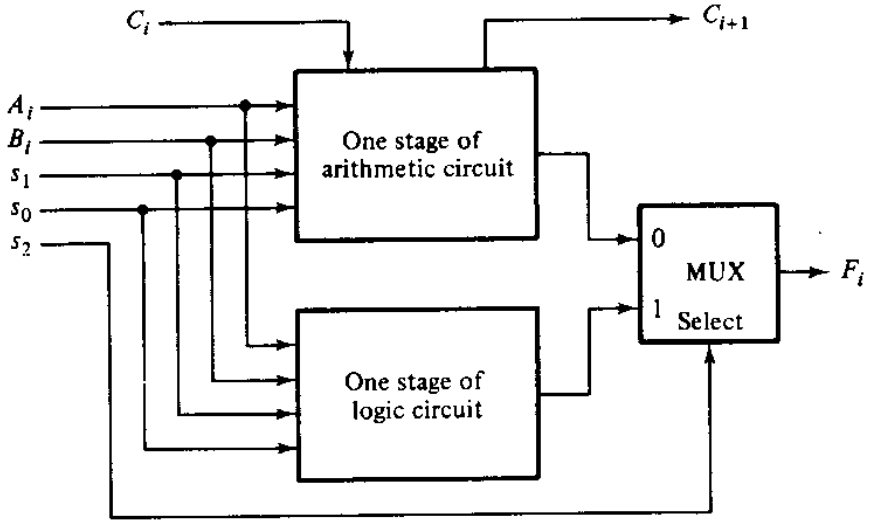



**Figure 9-12** Combining logic and arithmetic circuits

TABLE 9-3 Logic operations in one stage of arithmetic circuit

| $s_2$ | $s_1$ | $s_0$ | $X_i$ | $Y_i$ | $C_i$ | $F_i = X_i \oplus Y_i$ | Operation | Required operation |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | $A_i$ | 0 | 0 | $F_i = A_i$ | Transfer $A$ | OR |
| 1 | 0 | 1 | $A_i$ | $B_i$ | 0 | $F_i = A_i \oplus B_i$ | XOR | XOR |
| 1 | 1 | 0 | $A_i$ | $B_i'$ | 0 | $F_i = A_i \odot B_i$ | Equivalence | AND |
| 1 | 1 | 1 | $A_i$ | 1 | 0 | $F_i = A_i'$ | NOT | NOT |

The value of $X_i$ is always equal to input $A_i$. Table 9-3 shows the four logic operations obtained when a third selection variable $s_2 = 1$. This selection variable forces $C_i$ to be equal to 0 while $s_1$ and $s_0$ choose a particular value for $Y_i$. The four logic operations obtained by this configuration are transfer, exclusive-OR, equivalence, and complement. The third entry is the equivalence operation because:

$$A_i \oplus B_i' = A_i B_i + A_i' B_i' = A_i \odot B_i$$

The last entry in the table is the NOT or complement operation because:

$$A_i \oplus 1 = A_i'$$

The table has one more column which lists the four logic operations we want to include in the ALU. Two of these operations, XOR and NOT, are already available. The question that must be answered is whether it is possible to modify the arithmetic circuit further so that it will generate the logic functions OR and AND instead of the transfer and equivalence functions. This problem is investigated in the next section.

## 9-6 DESIGN OF ARITHMETIC LOGIC UNIT

In this section, we design an ALU with eight arithmetic operations and four logic operations. Three selection variables $s_2$, $s_1$, and $s_0$ select eight different operations, and the input carry $C_{in}$ is used to select four additional arithmetic operations. With $s_2 = 0$, selection variables $s_1$ and $s_0$ together with $C_{in}$ will select the eight arithmetic operations listed in Table 9-1. With $s_2 = 1$, variables $s_1$ and $s_0$ will select the four logic operations OR, XOR, AND, and NOT.

The design of an ALU is a combinational-logic problem. Because the unit has a regular pattern, it can be broken into identical stages connected in cascade through the carries. We can design one stage of the ALU and then duplicate it for the number of stages required. There are six inputs to each stage: $A_i$, $B_i$, $C_i$, $s_2$, $s_1$, and $s_0$. There are two outputs in each stage: output $F_i$ and the carry out $C_{i+1}$. One can formulate a truth table with 64 entries and simplify the two output functions.

Here we choose to employ an alternate procedure that uses the availability of a parallel adder.

The steps involved in the design of an ALU are as follows:

1. Design the arithmetic section independent of the logic section.
2. Determine the logic operations obtained from the arithmetic circuit in step 1, assuming that the input carries to all stages are 0.
3. Modify the arithmetic circuit to obtain the required logic operations.

The third step in the design is not a straightforward procedure and requires a certain amount of ingenuity on the part of the designer. There is no guarantee that a solution can be found or that the solution uses the minimum number of gates. The example presented here demonstrates the type of logical thinking sometimes required in the design of digital systems.

It must be realized that various ALUs are available in IC packages. In a practical situation, all that one must do is search for a suitable ALU or processor unit among the ICs that are available commercially. Yet, the internal logic of the IC selected must have been designed by a person familiar with logic design techniques.

The solution to the first design step is shown in Fig. 9-8. The solution to the second design step is presented in Table 9-3. The solution of the third step is carried out below.

From Table 9-3, we see that when $s_2 = 1$, the input carry $C_i$ in each stage must be 0. With $s_1s_0 = 00$, each stage as it stands generates the function $F_i = A_i$. To change the output to an OR operation, we must change the input to each full-adder circuit from $A_i$ to $A_i + B_i$. This can be accomplished by ORing $B_i$ and $A_i$ when $s_2s_1s_0 = 100$.

The other selection variables that give an undesirable output occur when $s_2s_1s_0 = 110$. The unit as it stands generates an output $F_i = A_i \odot B_i$, but we want to generate the AND operation $F_i = A_iB_i$. Let us investigate the possibility of ORing each input $A_i$ with some Boolean function $K_i$. The function so obtained is then used for $X_i$ when $s_2s_1s_0 = 110$:

$$F_i = X_i \oplus Y_i = (A_i + K_i) \oplus B_i' = A_iB_i + K_iB_i + A_i'K_i'B_i'$$

Careful inspection of the result reveals that if the variable $K_i = B_i'$, we obtain an output:

$$F_i = A_iB_i + B_i'B_i + A_iB_iB_i' = A_iB_i$$

Two terms are equal to 0 because $B_iB_i' = 0$. The result obtained is the AND operation as required. The conclusion is that, if $A_i$ is ORed with $B_i'$ when $s_2s_1s_0 = 110$, the output will generate the AND operation.

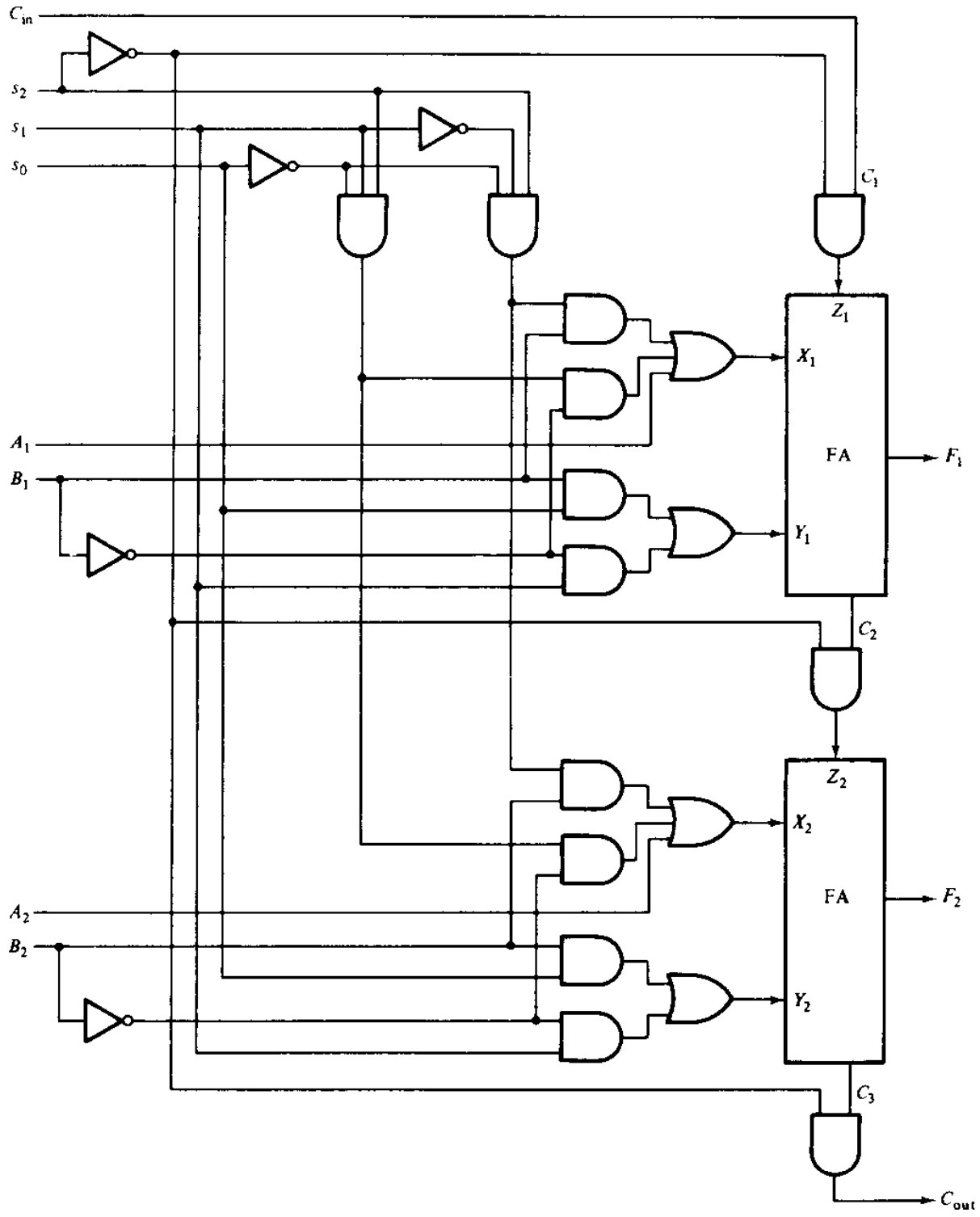



**Figure 9-13** Logic diagram of arithmetic logic unit (ALU)

The final ALU is shown in Fig. 9-13. Only the first two stages are drawn, but the diagram can be easily extended to more stages. The inputs to each full-adder circuit are specified by the Boolean functions:

$$X_i = A_i + s_2 s_1' s_0' B_i + s_2 s_1 s_0' B_i'$$
$$Y_i = s_0 B_i + s_1 B_i'$$
$$Z_i = s_2' C_i$$

When $s_2 = 0$, the three functions reduce to:

$$X_i = A_i$$
$$Y_i = s_0 B_i + s_1 B_i'$$
$$Z_i = C_i$$

which are the functions for the arithmetic circuit of Fig. 9-8. The logic operations are generated when $s_2 = 1$. For $s_2 s_1 s_0 = 101$ or 111, the functions reduce to:

$$X_i = A_i$$
$$Y_i = s_0 B_i + s_1 B_i'$$
$$C_i = 0$$

Output $F_i$ is then equal to $X_i \oplus Y_i$ and produces the exclusive-OR and complement operations as specified in Table 9-3. When $s_2 s_1 s_0 = 100$, each $A_i$ is ORed with $B_i$ to provide the OR operation as discussed above. When $s_2 s_1 s_0 = 110$, each $A_i$ is ORed with $B_i'$ to provide the AND operation as explained previously.

The 12 operations generated in the ALU are summarized in Table 9-4. The particular function is selected through $s_2$, $s_1$, $s_0$, and $C_{in}$. The arithmetic operations

**TABLE 9-4** Function table for the ALU of Fig. 9-13

| Selection | | | | Output | Function |
|---|---|---|---|---|---|
| $s_2$ | $s_1$ | $s_0$ | $C_{in}$ | | |
| 0 | 0 | 0 | 0 | $F = A$ | Transfer $A$ |
| 0 | 0 | 0 | 1 | $F = A + 1$ | Increment $A$ |
| 0 | 0 | 1 | 0 | $F = A + B$ | Addition |
| 0 | 0 | 1 | 1 | $F = A + B + 1$ | Add with carry |
| 0 | 1 | 0 | 0 | $F = A - B - 1$ | Subtract with borrow |
| 0 | 1 | 0 | 1 | $F = A - B$ | Subtraction |
| 0 | 1 | 1 | 0 | $F = A - 1$ | Decrement $A$ |
| 0 | 1 | 1 | 1 | $F = A$ | Transfer $A$ |
| 1 | 0 | 0 | $X$ | $F = A \vee B$ | OR |
| 1 | 0 | 1 | $X$ | $F = A \oplus B$ | XOR |
| 1 | 1 | 0 | $X$ | $F = A \wedge B$ | AND |
| 1 | 1 | 1 | $X$ | $F = \overline{A}$ | Complement $A$ |

are identical to the ones listed for the arithmetic circuit. The value of $C_{in}$ for the four logic functions has no effect on the operation of the unit and those entries are marked with don't-care $X$'s.

## 9-7 STATUS REGISTER

The relative magnitudes of two numbers may be determined by subtracting one number from the other and then checking certain bit conditions in the resultant difference. If the two numbers are unsigned, the bit conditions of interest are the output carry and a possible zero result. If the two numbers include a sign bit in the highest-order position, the bit conditions of interest are the sign of the result, a zero indication, and an overflow condition. It is sometimes convenient to supplement the ALU with a status register where these status-bit conditions are stored for further analysis. Status-bit conditions are sometimes called *condition-code* bits or *flag* bits.

Figure 9-14 shows the block diagram of an 8-bit ALU with a 4-bit status register. The four status bits are symbolized by $C$, $S$, $Z$, and $V$. The bits are set or cleared as a result of an operation performed in the ALU.

1. Bit $C$ is set if the output carry of the ALU is 1. It is cleared if the output carry is 0.
2. Bit $S$ is set if the highest-order bit of the result in the output of the ALU (the sign bit) is 1. It is cleared if the highest-order bit is 0.
3. Bit $Z$ is set if the output of the ALU contains all 0's, and cleared otherwise. $Z = 1$ if the result is zero, and $Z = 0$ if the result is nonzero.
4. Bit $V$ is set if the exclusive-OR of carries $C_8$ and $C_9$ is 1, and cleared otherwise. This is the condition for overflow when the numbers are in sign-2's-complement representation (see Section 8-6). For the 8-bit ALU, $V$ is set if the result is greater than 127 or less than $-128$.

The status bits can be checked after an ALU operation to determine certain relationships that exist between the values of $A$ and $B$. If bit $V$ is set after the addition of two signed numbers, it indicates an overflow condition. If $Z$ is set after an exclusive-OR operation, it indicates that $A = B$. This is so because $x \oplus x = 0$, and the exclusive-OR of two equal operands gives an all-0's result which sets the $Z$ bit. A single bit in $A$ can be checked to determine if it is 0 or 1 by masking all bits except the bit in question and then checking the $Z$ status bit. For example, let $A = 101x1100$, where $x$ is the bit to be checked. The AND operation of $A$ with $B = 00010000$ produces a result $000x0000$. If $x = 0$, the $Z$ status bit is set, but if $x = 1$, the $Z$ bit is cleared since the result is not zero.

The *compare* operation is a subtraction of $B$ from $A$, except that the result of the operation is not transferred into a destination register, but the status bits are affected. The status register then provides the information about the relative

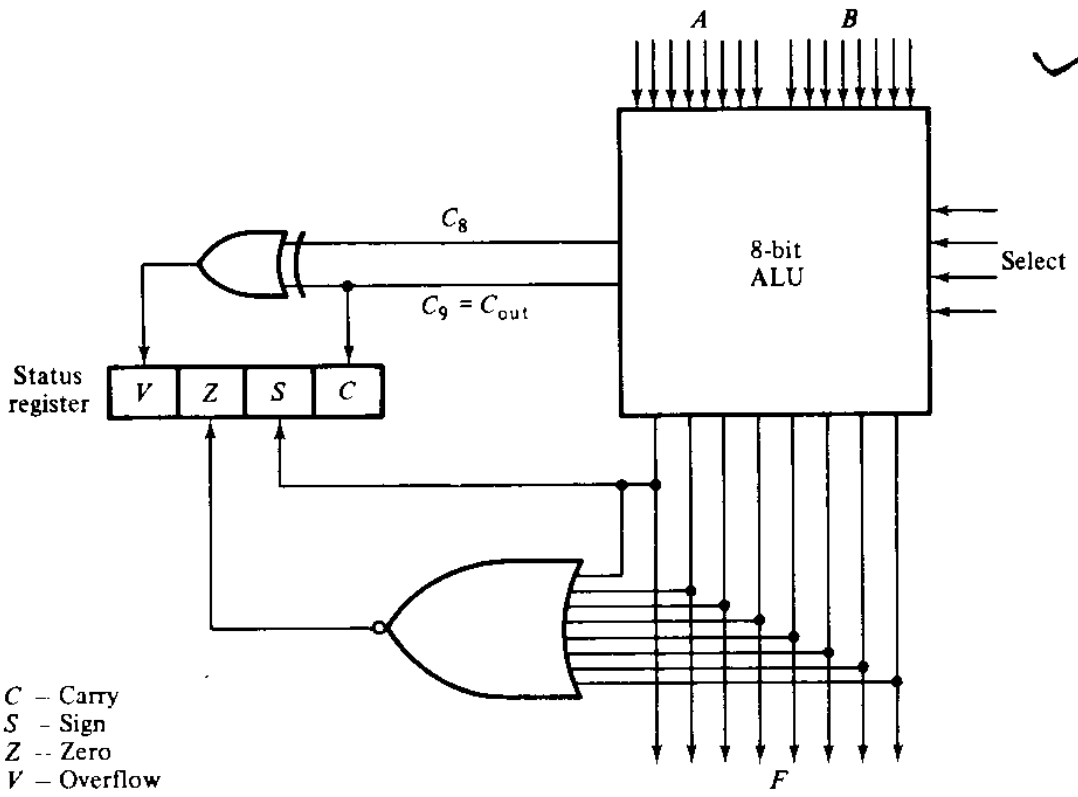



**Figure 9-14** Setting bits in a status register

magnitudes of $A$ and $B$. The status bits to consider depend on whether we take the two numbers to be unsigned or signed and in 2's-complement representation.

Consider the operation $A - B$ done with two *unsigned* binary numbers. The relative magnitudes of $A$ and $B$ can be determined from the values transferred to the $C$ and $Z$ status bits. If $Z = 1$, then we know that $A = B$, since $A - B = 0$. If $Z = 0$, then we know that $A \neq B$. From Table 9-2, we have that $C = 1$ if $A \geq B$ and $C = 0$ if $A < B$. These conditions are listed in Table 9-5. The table lists two other conditions. For $A$ to be greater than but not equal to $B$ ($A > B$), we must have $C = 1$ and $Z = 0$. Since $C$ is set when the result is 0, we must check $Z$ to

**TABLE 9-5** Status bits after the subtraction of unsigned numbers ($A - B$)

| Relation | Condition of status bits | Boolean function |
|---|---|---|
| $A > B$ | $C = 1$ and $Z = 0$ | $CZ'$ |
| $A \geq B$ | $C = 1$ | $C$ |
| $A < B$ | $C = 0$ | $C'$ |
| $A \leq B$ | $C = 0$ or $Z = 1$ | $C' + Z$ |
| $A = B$ | $Z = 1$ | $Z$ |
| $A \neq B$ | $Z = 0$ | $Z'$ |

ensure that the result is not 0. For $A$ to be less than or equal to $B$ ($A \leqslant B$), the $C$ bit must be 0 (for $A < B$) or the $Z$ bit must be 1 (for $A = B$). Table 9-5 also lists the Boolean functions that must be satisfied for each of the six relationships.

Some computers consider the $C$ bit to be a borrow bit after a subtraction operation $A - B$. An end borrow does not occur if $A > B$, but an extra bit must be borrowed when $A < B$. The condition for a borrow is the complement of the output carry obtained when the subtraction is done by taking the 2's complement of $B$. For this reason, a processor that considers the $C$ bit to be a borrow after a subtraction will complement the $C$ bit after a subtraction or compare operation and denote this bit as a borrow.

Now consider the operation $A - B$ done with two *signed* binary numbers when negative numbers are in 2's-complement form. The relative magnitudes of $A$ and $B$ can be determined from the values transferred to the $Z$, $S$, and $V$ status bits. If $Z = 1$, then we know that $A = B$; when $Z = 0$, we have that $A \neq B$. If $S = 0$, the sign of the result is positive, so $A$ must be greater than $B$. This is true if there was no overflow and $V = 0$. If the result overflows, we obtain an erroneous result. It was shown in Section 8-5 that an overflow condition changes the sign of the result. Therefore, if $S = 1$ and $V = 1$, it indicates that the result should have been positive and therefore $A$ must be greater than $B$.

Table 9-6 lists the six possible relationships that can exist between $A$ and $B$ and the corresponding values of $Z$, $S$, and $V$ in each case. For $A - B$ to be greater than but not equal to zero ($A > B$), the result must be positive and nonzero. Since a zero result gives a positive sign, we must ensure that the $Z$ bit is 0 to exclude the possibility of $A = B$. For $A \geqslant B$, it is sufficient to check for a positive sign when no overflow occurs or a negative sign when an overflow occurs. For $A < B$, the result must be negative. If the result is negative or zero, we have that $A \leqslant B$. The Boolean functions listed in the table express the status-bit conditions in algebraic form.

**TABLE 9-6** Status bits after the subtraction of sign-2's complement numbers ($A - B$)

| Relation | Condition of status bits | Boolean function |
|---|---|---|
| $A > B$ | $Z = 0$ and ($S = 0, V = 0$ or $S = 1, V = 1$) | $Z'(S \odot V)$ |
| $A \geqslant B$ | $S = 0, V = 0$ or $S = 1, V = 1$ | $S \odot V$ |
| $A < B$ | $S = 1, V = 0$ or $S = 0, V = 1$ | $S \oplus V$ |
| $A \leqslant B$ | $S = 1, V = 0$ or $S = 0, V = 1$ or $Z = 1$ | $(S \oplus V) + Z$ |
| $A = B$ | $Z = 1$ | $Z$ |
| $A \neq B$ | $Z = 0$ | $Z'$ |

## 9-8 DESIGN OF SHIFTER

The shift unit attached to a processor transfers the output of the ALU onto the output bus. The shifter may transfer the information directly without a shift, or it may shift the information to the right or left. Provision is sometimes made for no