A simplified POS expression can be derived as

$$\begin{aligned} f &= ((x_1 + x_2) + x_3)((x_1 + x_2) + \bar{x}_3)(x_1 + (x_2 + \bar{x}_3))(\bar{x}_1 + (x_2 + \bar{x}_3)) \\ &= ((x_1 + x_2) + x_3\bar{x}_3)(x_1\bar{x}_1 + (x_2 + \bar{x}_3)) \\ &= (x_1 + x_2)(x_2 + \bar{x}_3) \end{aligned}$$

Note that by using the distributive property 12*b*, this expression leads to

$$f = x_2 + x_1\bar{x}_3$$

which is the same as the expression derived in Example 2.3.

Exa Suppose that a four-variable function is defined by

$$f(x_1, x_2, x_3, x_4) = \sum m(3, 7, 9, 12, 13, 14, 15)$$

The canonical SOP expression for this function is

$$f = \bar{x}_1\bar{x}_2x_3x_4 + \bar{x}_1x_2x_3x_4 + x_1\bar{x}_2\bar{x}_3x_4 + x_1x_2\bar{x}_3\bar{x}_4 + x_1x_2\bar{x}_3x_4 + x_1x_2x_3\bar{x}_4 + x_1x_2x_3x_4$$

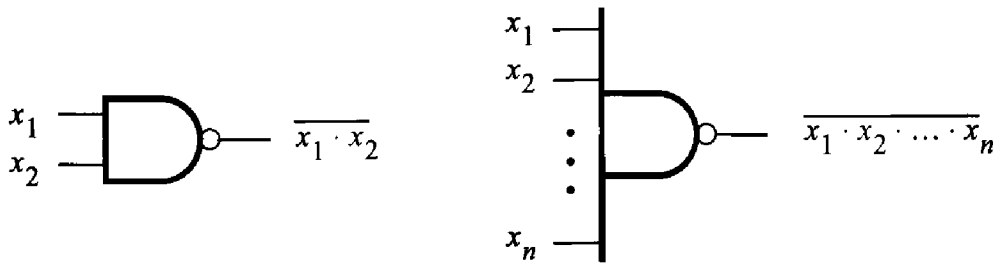
A simpler SOP expression can be obtained as follows

$$\begin{aligned} f &= \bar{x}_1(\bar{x}_2 + x_2)x_3x_4 + x_1(\bar{x}_2 + x_2)\bar{x}_3x_4 + x_1x_2\bar{x}_3(\bar{x}_4 + x_4) + x_1x_2x_3(\bar{x}_4 + x_4) \\ &= \bar{x}_1x_3x_4 + x_1\bar{x}_3x_4 + x_1x_2\bar{x}_3 + x_1x_2x_3 \\ &= \bar{x}_1x_3x_4 + x_1\bar{x}_3x_4 + x_1x_2(\bar{x}_3 + x_3) \\ &= \bar{x}_1x_3x_4 + x_1\bar{x}_3x_4 + x_1x_2 \end{aligned}$$

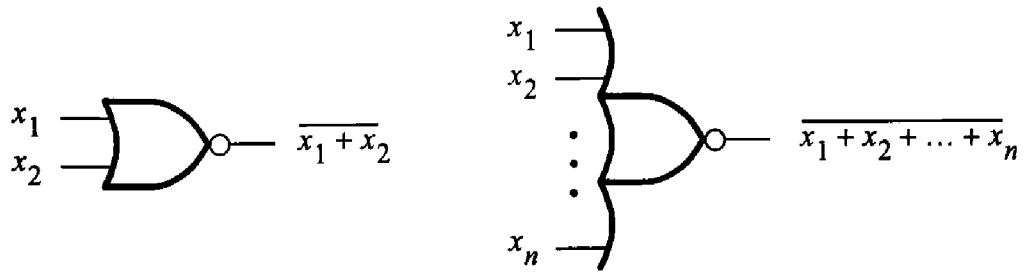
2.7 NAND AND NOR LOGIC NETWORKS

We have discussed the use of AND, OR, and NOT gates in the synthesis of logic circuits. There are other basic logic functions that are also used for this purpose. Particularly useful are the NAND and NOR functions which are obtained by complementing the output generated by AND and OR operations, respectively. These functions are attractive because they are implemented with simpler electronic circuits than the AND and OR functions, as we will see in Chapter 3. Figure 2.20 gives the graphical symbols for the NAND and NOR gates. A bubble is placed on the output side of the AND and OR gate symbols to represent the complemented output signal.

If NAND and NOR gates are realized with simpler circuits than AND and OR gates, then we should ask whether these gates can be used directly in the synthesis of logic circuits. In section 2.5 we introduced DeMorgan's theorem. Its logic gate interpretation is shown in Figure 2.21. Identity 15*a* is interpreted in part (*a*) of the figure. It specifies that a NAND of variables x_1 and x_2 is equivalent to first complementing each of the variables and then ORing them. Notice on the far-right side that we have indicated the NOT gates

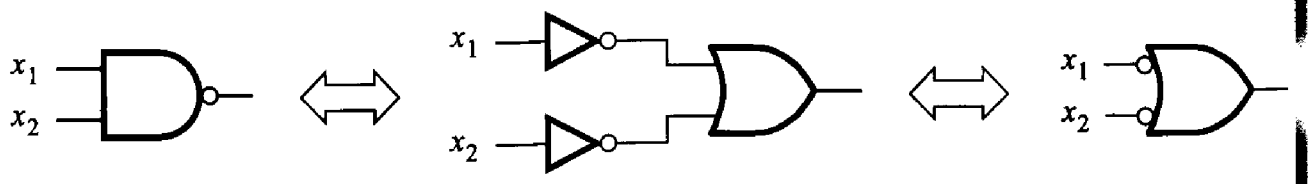


(a) NAND gates

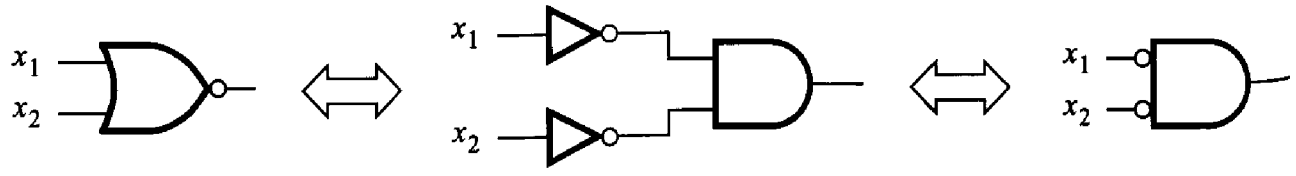


(b) NOR gates

Figure 2.20 NAND and NOR gates.



(a) $\overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2$



(b) $\overline{x_1 + x_2} = \bar{x}_1 \bar{x}_2$

Figure 2.21 DeMorgan's theorem in terms of logic gates.

simply as bubbles, which denote inversion of the logic value at that point. The other half of DeMorgan's theorem, identity 15*b*, appears in part (*b*) of the figure. It states that the NOR function is equivalent to first inverting the input variables and then ANDing them.

In section 2.6 we explained how any logic function can be implemented either in sum-of-products or product-of-sums form, which leads to logic networks that have either an AND-OR or an OR-AND structure, respectively. We will now show that such networks can be implemented using only NAND gates or only NOR gates.

Consider the network in Figure 2.22 as a representative of general AND-OR networks. This network can be transformed into a network of NAND gates as shown in the figure. First, each connection between the AND gate and an OR gate is replaced by a connection that includes two inversions of the signal: one inversion at the output of the AND gate and the other at the input of the OR gate. Such double inversion has no effect on the behavior of the network, as stated formally in theorem 9 in section 2.5. According to Figure 2.21*a*, the OR gate with inversions at its inputs is equivalent to a NAND gate. Thus we can redraw the network using only NAND gates, as shown in Figure 2.22. This example shows that any AND-OR network can be implemented as a NAND-NAND network having the same topology.

Figure 2.23 gives a similar construction for a product-of-sums network, which can be transformed into a circuit with only NOR gates. The procedure is exactly the same as the one described for Figure 2.22 except that now the identity in Figure 2.21*b* is applied. The conclusion is that any OR-AND network can be implemented as a NOR-NOR network having the same topology.

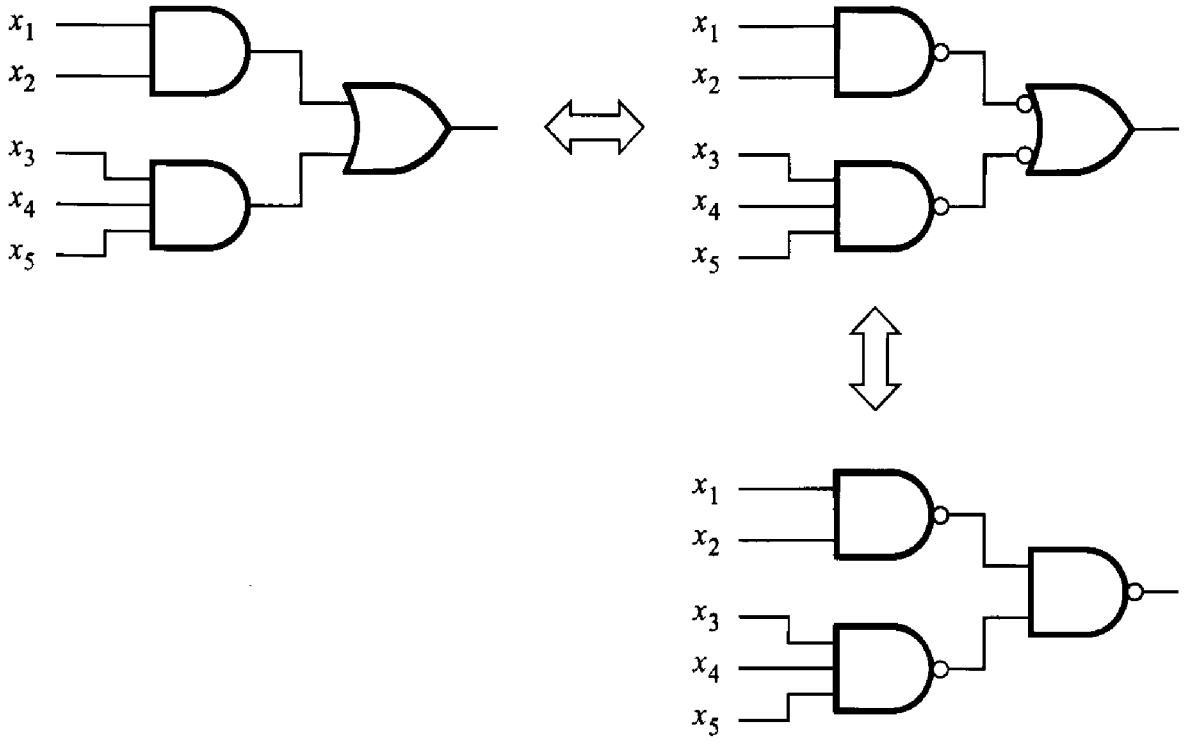


Figure 2.22 Using NAND gates to implement a sum-of-products.

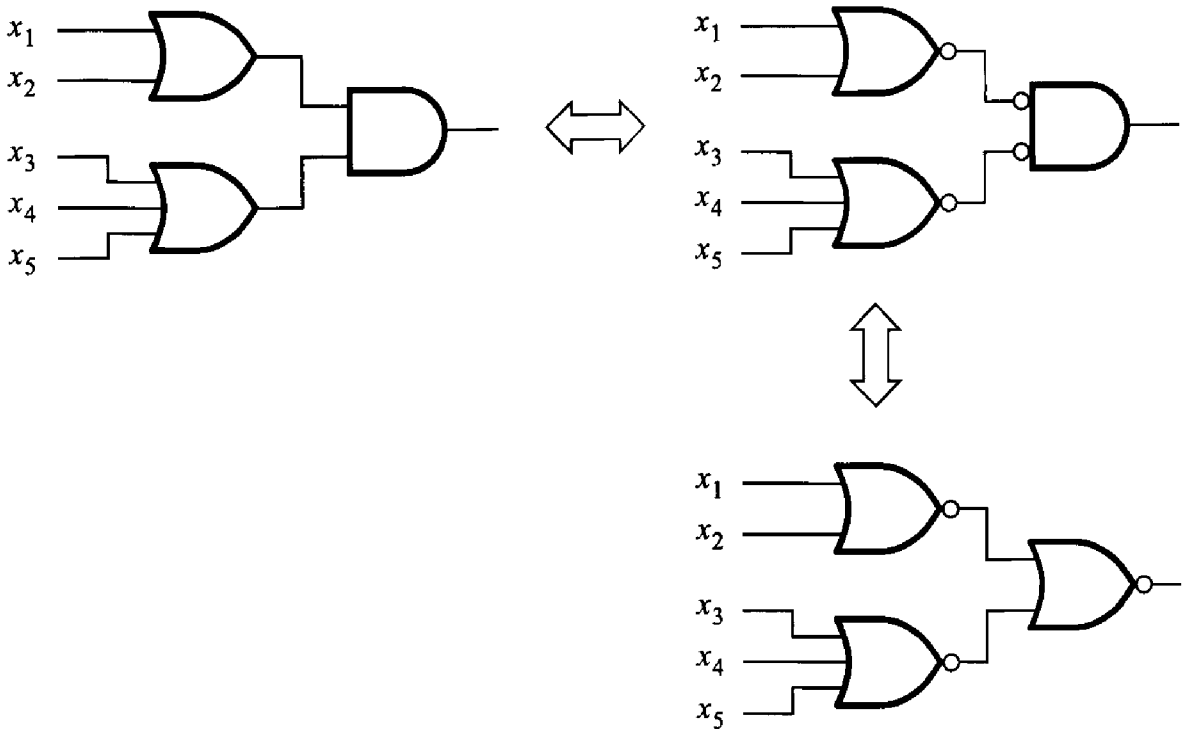


Figure 2.23 Using NOR gates to implement a product-of-sums.

ıle 2.6 Let us implement the function

$$f(x_1, x_2, x_3) = \sum m(2, 3, 4, 6, 7)$$

using NOR gates only. In Example 2.4 we showed that the function can be represented by the POS expression

$$f = (x_1 + x_2)(x_2 + \bar{x}_3)$$

An OR-AND circuit that corresponds to this expression is shown in Figure 2.24*a*. Using the same structure of the circuit, a NOR-gate version is given in Figure 2.24*b*. Note that x_3 is inverted by a NOR gate that has its inputs tied together.

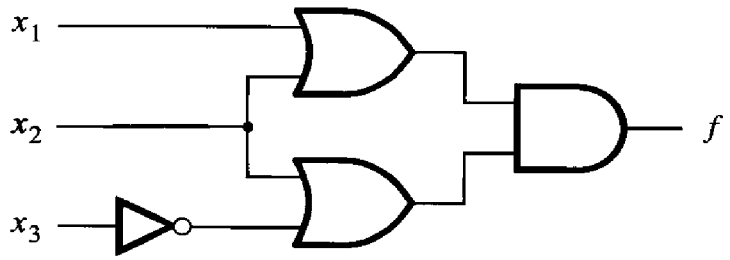
ıle 2.7 Let us now implement the function

$$f(x_1, x_2, x_3) = \sum m(2, 3, 4, 6, 7)$$

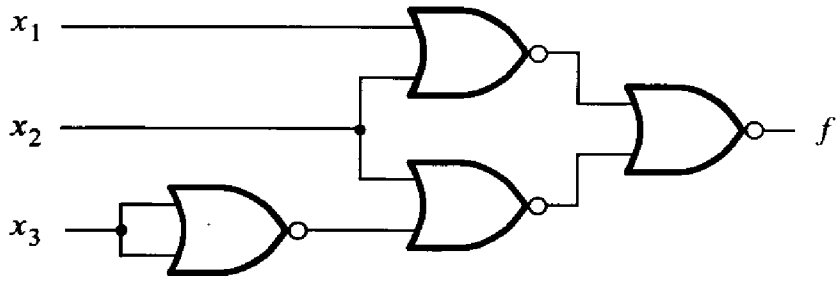
using NAND gates only. In Example 2.3 we derived the SOP expression

$$f = x_2 + x_1\bar{x}_3$$

which is realized using the circuit in Figure 2.25*a*. We can again use the same structure to obtain a circuit with NAND gates, but with one difference. The signal x_2 passes only through an OR gate, instead of passing through an AND gate and an OR gate. If we simply replace the OR gate with a NAND gate, this signal would be inverted which would result in a wrong output value. Since x_2 must either not be inverted, or it can be inverted twice,

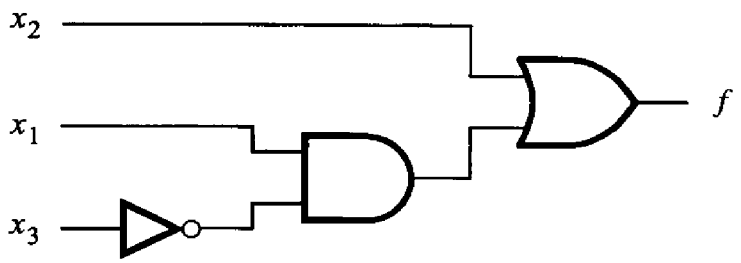


(a) POS implementation

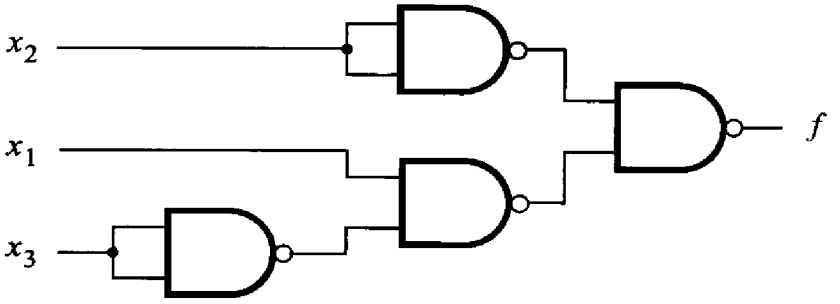


(b) NOR implementation

Figure 2.24 NOR-gate realization of the function in Example 2.4.



(a) SOP implementation



(b) NAND implementation

Figure 2.25 NAND-gate realization of the function in Example 2.3.

we can pass it through two NAND gates as depicted in Figure 2.25*b*. Observe that for this circuit the output f is

$$f = \overline{\overline{x}_2 \cdot \overline{x_1 \overline{x}_3}}$$

Applying DeMorgan's theorem, this expression becomes

$$f = x_2 + x_1 \overline{x}_3$$

2.8 DESIGN EXAMPLES

Logic circuits provide a solution to a problem. They implement functions that are needed to carry out specific tasks. Within the framework of a computer, logic circuits provide complete capability for execution of programs and processing of data. Such circuits are complex and difficult to design. But regardless of the complexity of a given circuit, a designer of logic circuits is always confronted with the same basic issues. First, it is necessary to specify the desired behavior of the circuit. Second, the circuit has to be synthesized and implemented. Finally, the implemented circuit has to be tested to verify that it meets the specifications. The desired behavior is often initially described in words, which then must be turned into a formal specification. In this section we give two simple examples of design.

2.8.1 THREE-WAY LIGHT CONTROL

Assume that a large room has three doors and that a switch near each door controls a light in the room. It has to be possible to turn the light on or off by changing the state of any one of the switches.

As a first step, let us turn this word statement into a formal specification using a truth table. Let x_1, x_2, and x_3 be the input variables that denote the state of each switch. Assume that the light is off if all switches are open. Closing any one of the switches will turn the light on. Then turning on a second switch will have to turn off the light. Thus the light will be on if exactly one switch is closed, and it will be off if two (or no) switches are closed. If the light is off when two switches are closed, then it must be possible to turn it on by closing the third switch. If $f(x_1, x_2, x_3)$ represents the state of the light, then the required functional behavior can be specified as shown in the truth table in Figure 2.26. The canonical sum-of-products expression for the specified function is

$$\begin{aligned} f &= m_1 + m_2 + m_4 + m_7 \\ &= \overline{x}_1 \overline{x}_2 x_3 + \overline{x}_1 x_2 \overline{x}_3 + x_1 \overline{x}_2 \overline{x}_3 + x_1 x_2 x_3 \end{aligned}$$

This expression cannot be simplified into a lower-cost sum-of-products expression. The resulting circuit is shown in Figure 2.27*a*.

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Figure 2.26 Truth table for the three-way light control.

An alternative realization for this function is in the product-of-sums form. The canonical expression of this type is

$$\begin{aligned} f &= M_0 \cdot M_3 \cdot M_5 \cdot M_6 \\ &= (x_1 + x_2 + x_3)(x_1 + \bar{x}_2 + \bar{x}_3)(\bar{x}_1 + x_2 + \bar{x}_3)(\bar{x}_1 + \bar{x}_2 + x_3) \end{aligned}$$

The resulting circuit is depicted in Figure 2.27*b*. It has the same cost as the circuit in part (*a*) of the figure.

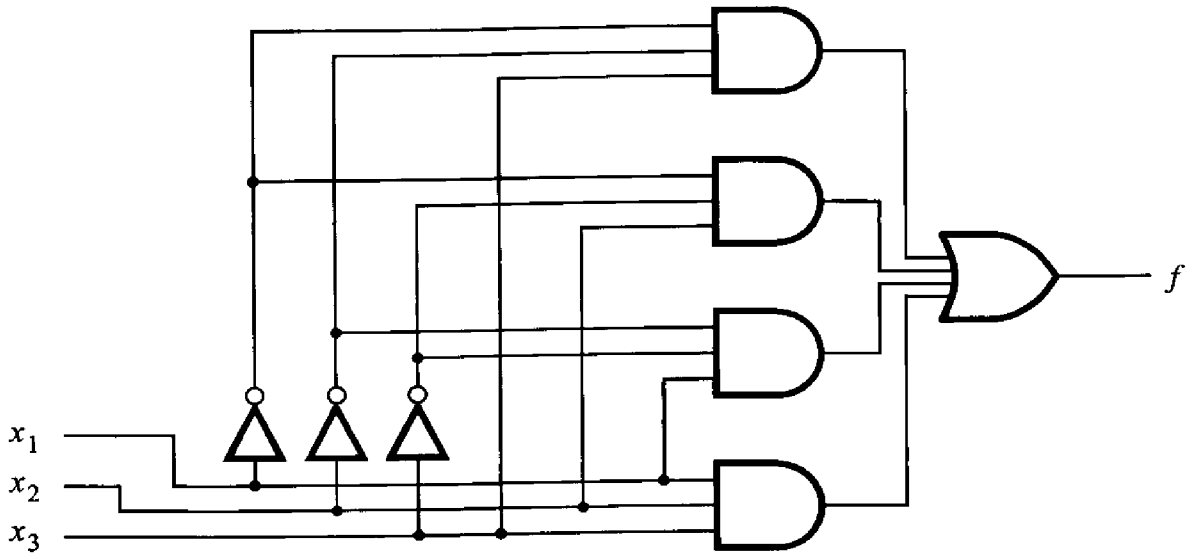
When the designed circuit is implemented, it can be tested by applying the various input valuations to the circuit and checking whether the output corresponds to the values specified in the truth table. A straightforward approach is to check that the correct output is produced for all eight possible input valuations.

2.8.2 MULTIPLEXER CIRCUIT

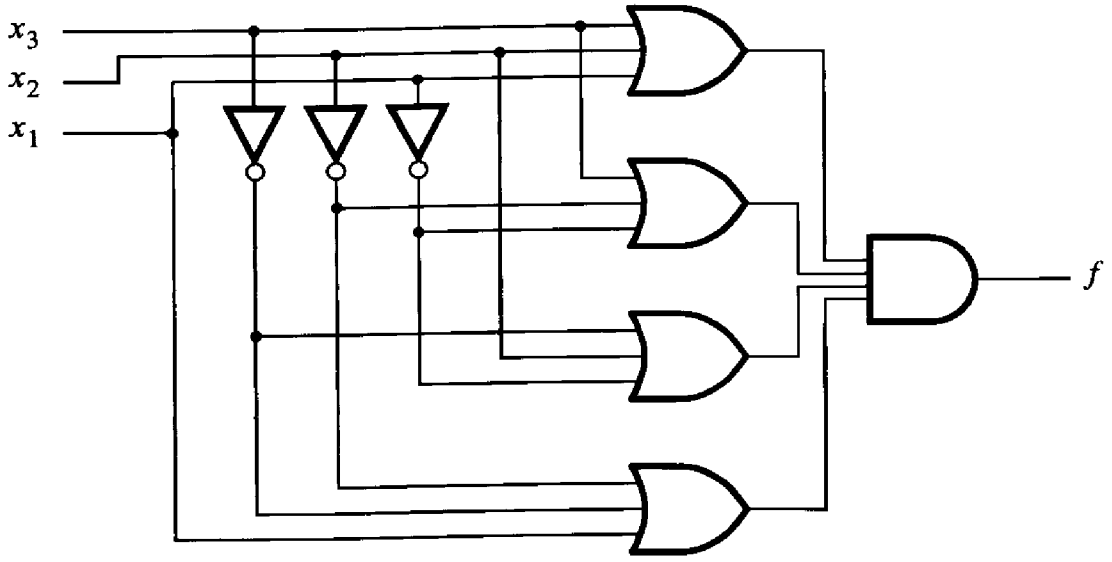
In computer systems it is often necessary to choose data from exactly one of a number of possible sources. Suppose that there are two sources of data, provided as input signals x_1 and x_2. The values of these signals change in time, perhaps at regular intervals. Thus sequences of 0s and 1s are applied on each of the inputs x_1 and x_2. We want to design a circuit that produces an output that has the same value as either x_1 or x_2, dependent on the value of a selection control signal s. Therefore, the circuit should have three inputs: x_1, x_2, and s. Assume that the output of the circuit will be the same as the value of input x_1 if $s = 0$, and it will be the same as x_2 if $s = 1$.

Based on these requirements, we can specify the desired circuit in the form of a truth table given in Figure 2.28*a*. From the truth table, we derive the canonical sum of products

$$f(s, x_1, x_2) = \bar{s}x_1\bar{x}_2 + \bar{s}x_1x_2 + s\bar{x}_1x_2 + sx_1x_2$$



(a) Sum-of-products realization



(b) Product-of-sums realization

Figure 2.27 Implementation of the function in Figure 2.26.

Using the distributive property, this expression can be written as

$$f = \bar{s}x_1(\bar{x}_2 + x_2) + s(\bar{x}_1 + x_1)x_2$$

Applying theorem 8*b* yields

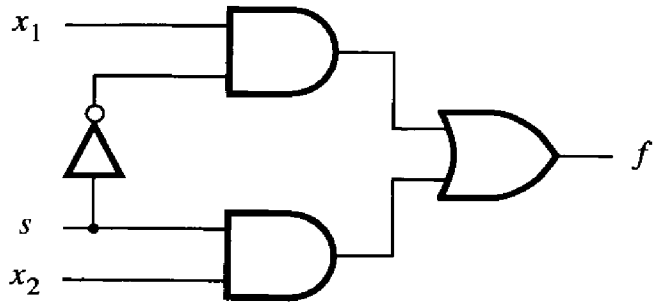
$$f = \bar{s}x_1 \cdot 1 + s \cdot 1 \cdot x_2$$

Finally, theorem 6*a* gives

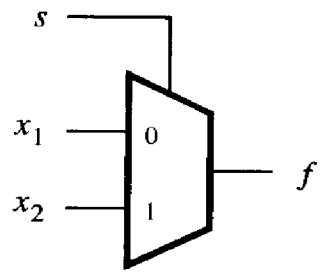
$$f = \bar{s}x_1 + sx_2$$

$s\ x_1\ x_2$	$f(s, x_1, x_2)$
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

(a) Truth table



(b) Circuit



(c) Graphical symbol

s	$f(s, x_1, x_2)$
0	x_1
1	x_2

(d) More compact truth-table representation

Figure 2.28 Implementation of a multiplexer.

A circuit that implements this function is shown in Figure 2.28*b*. Circuits of this type are used so extensively that they are given a special name. A circuit that generates an output that exactly reflects the state of one of a number of data inputs, based on the value of one or more selection control inputs, is called a *multiplexer*. We say that a multiplexer circuit "multiplexes" input signals onto a single output.