2.1. The proof is as follows:

$$\begin{aligned}(x+y)\cdot(x+z) &= xx + xz + xy + yz\\ &= x + xz + xy + yz\\ &= x(1+z+y) + yz\\ &= x\cdot 1 + yz\\ &= x + yz\end{aligned}$$

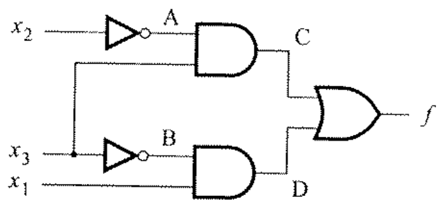
2.2. The proof is as follows:

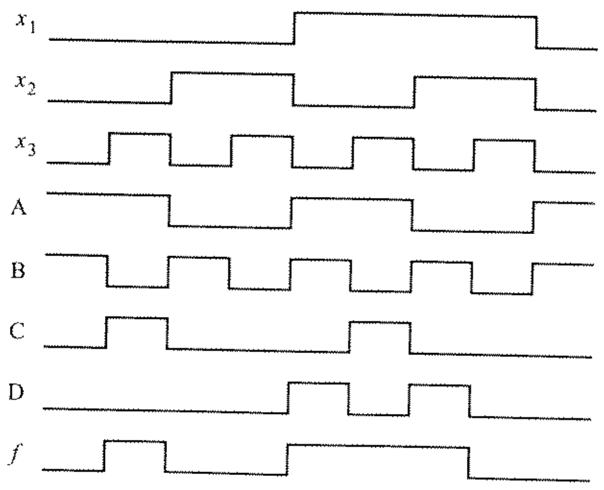
$$\begin{aligned}(x+y)\cdot(x+\overline{y}) &= xx + xy + x\overline{y} + y\overline{y}\\ &= x + xy + x\overline{y} + 0\\ &= x(1+y+\overline{y})\\ &= x\cdot 1\\ &= x\end{aligned}$$

2.6. A possible approach for determining whether or not the expressions are valid is to try to manipulate the left and right sides of an expression into the same form, using the theorems and properties presented in section 2.5. While this may seem simple, it is an awkward approach, because it is not obvious what target form one should try to reach. A much simpler approach is to construct a truth table for each side of an expression. If the truth tables are identical, then the expression is valid. Using this approach, we can show that the answers are:

(a) Yes
(b) Yes
(c) No

2.7. Timing diagram of the waveforms that can be observed on all wires of the circuit:





2.11. Derivation of the minimum sum-of-products expression:

$$\begin{aligned}
f &= x_1x_3 + x_1\overline{x}_2 + \overline{x}_1x_2x_3 + \overline{x}_1\overline{x}_2\overline{x}_3 \\
&= x_1(\overline{x}_2 + x_2)x_3 + x_1\overline{x}_2(\overline{x}_3 + x_3) + \overline{x}_1x_2x_3 + \overline{x}_1\overline{x}_2\overline{x}_3 \\
&= x_1\overline{x}_2x_3 + x_1x_2x_3 + x_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2x_3 + \overline{x}_1\overline{x}_2\overline{x}_3 \\
&= x_1x_3 + (x_1 + \overline{x}_1)x_2x_3 + (x_1 + \overline{x}_1)\overline{x}_2\overline{x}_3 \\
&= x_1x_3 + x_2x_3 + \overline{x}_2\overline{x}_3
\end{aligned}$$

2.13. The simplest POS expression is derived as

$$\begin{aligned}
f &= (x_1 + x_3 + x_4)(x_1 + \overline{x}_2 + x_3)(x_1 + \overline{x}_2 + \overline{x}_3 + x_4) \\
&= (x_1 + x_3 + x_4)(x_1 + \overline{x}_2 + x_3)(x_1 + \overline{x}_2 + x_3 + x_4)(x_1 + \overline{x}_2 + \overline{x}_3 + x_4) \\
&= (x_1 + x_3 + x_4)(x_1 + \overline{x}_2 + x_3)((x_1 + \overline{x}_2 + x_4)(x_3 + \overline{x}_3)) \\
&= (x_1 + x_3 + x_4)(x_1 + \overline{x}_2 + x_3)(x_1 + \overline{x}_2 + x_4) \cdot 1 \\
&= (x_1 + x_3 + x_4)(x_1 + \overline{x}_2 + x_3)(x_1 + \overline{x}_2 + x_4)
\end{aligned}$$

2.20. The simplest SOP implementation of the function is

$$\begin{aligned}
f &= \overline{x}_1\overline{x}_2x_3 + \overline{x}_1x_2x_3 + x_1\overline{x}_2\overline{x}_3 + x_1x_2\overline{x}_3 + x_1x_2x_3 \\
&= \overline{x}_1(\overline{x}_2 + x_2)x_3 + x_1(\overline{x}_2 + x_2)\overline{x}_3 + (\overline{x}_1 + x_1)x_2x_3 \\
&= \overline{x}_1x_3 + x_1\overline{x}_3 + x_2x_3
\end{aligned}$$

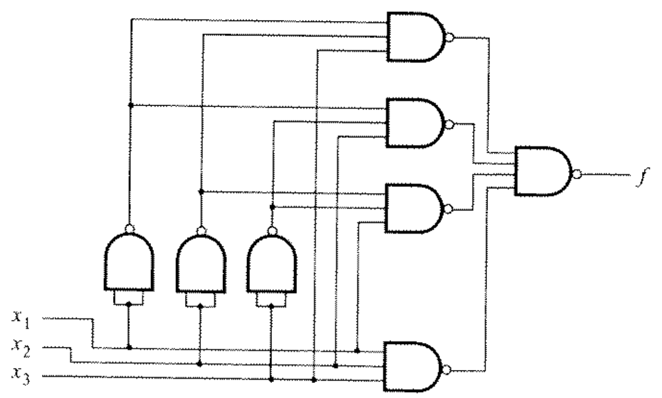
Another possibility is

$$f = \overline{x}_1x_3 + x_1\overline{x}_3 + x_1x_2$$

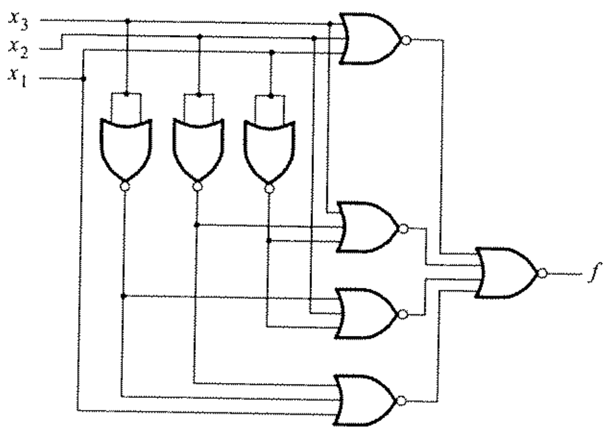
2.21. The simplest POS implementation of the function is

$$\begin{aligned}
f &= (x_1 + x_2 + x_3)(x_1 + \overline{x}_2 + x_3)(\overline{x}_1 + x_2 + \overline{x}_3) \\
&= ((x_1 + x_3) + x_2)((x_1 + x_3) + \overline{x}_2)(\overline{x}_1 + x_2 + \overline{x}_3) \\
&= (x_1 + x_3)(\overline{x}_1 + x_2 + \overline{x}_3)
\end{aligned}$$

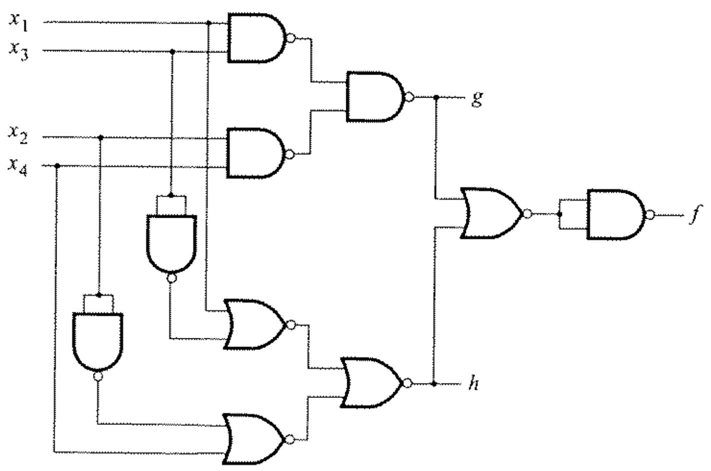
2.28. Using the ciruit in Figure 2.25a as a starting point, the function in Figure 2.24 can be implemented using NAND gates as follows:



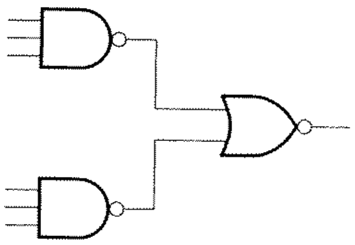
2.29. Using the ciruit in Figure 2.25*b* as a starting point, the function in Figure 2.24 can be implemented using NOR gates as follows:



2.30. The circuit in Figure 2.33 can be implemented using NAND and NOR gates as follows:

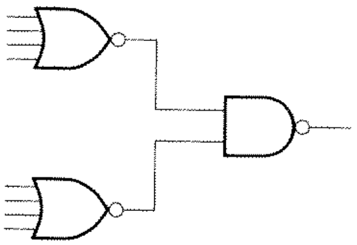


3.4. Using the circuit



The number of transistors needed is 16.

3.5. Using the circuit



The number of transistors needed is 20.

3.6. (*a*)

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

(*b*) The canonical SOP expression is

$$f = \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1\overline{x}_2x_3 + \overline{x}_1x_2\overline{x}_3 + \overline{x}_1x_2x_3 + x_1\overline{x}_2\overline{x}_3$$

The number of transistors required using only AND, OR, and NOT gates is

$$\begin{aligned}\#\text{transistors} &= \text{NOT\_gates} \times 2 + \text{AND\_gates} \times 8 + \text{OR\_gates} \times 12 \\ &= 3 \times 2 + 5 \times 8 + 1 \times 12 = 58\end{aligned}$$

3.7. (*a*)

x_1	x_2	x_3	x_4	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

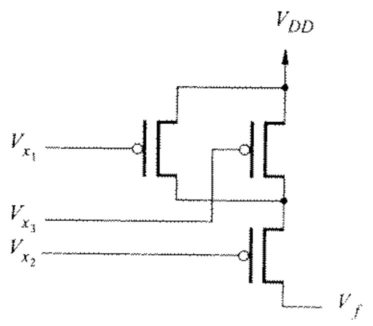
(*b*)

$$\begin{aligned}f &= \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3\overline{x}_4 + x_1\overline{x}_2\overline{x}_3\overline{x}_4 \\ &= \overline{x}_1\overline{x}_3\overline{x}_4 + \overline{x}_2\overline{x}_3\overline{x}_4\end{aligned}$$

The number of transistors required using only AND, OR, and NOT gates is

$$\begin{aligned}\#\text{transistors} &= \text{NOT\_gates} \times 2 + \text{AND\_gates} \times 8 + \text{OR\_gates} \times 4 \\ &= 4 \times 2 + 2 \times 8 + 1 \times 4 = 28\end{aligned}$$

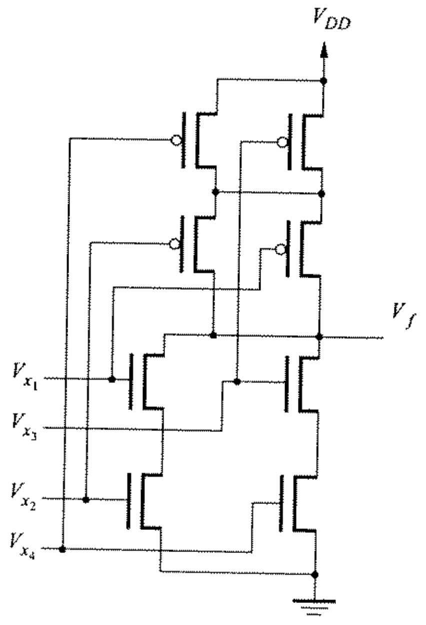
3.8.



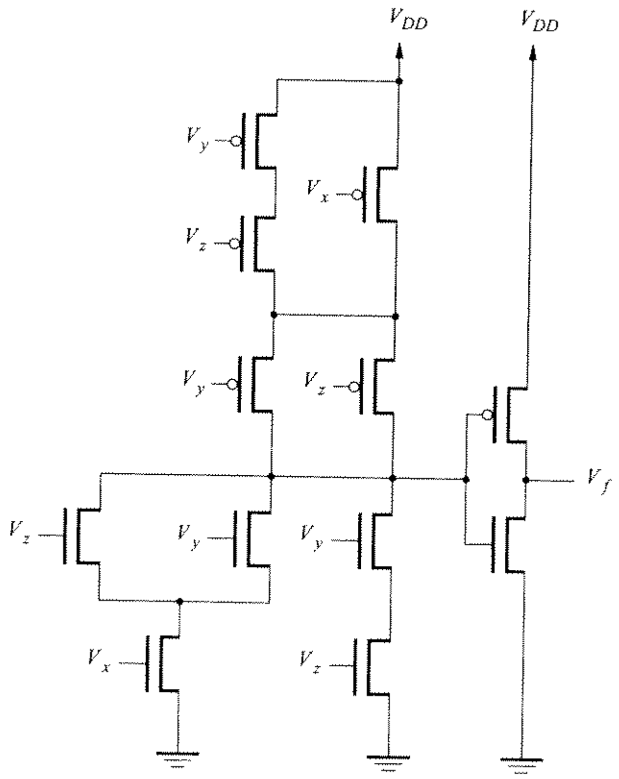
3.10. Minimum SOP expression for f is

$$\begin{aligned} f &= \overline{x}_2\overline{x}_3 + \overline{x}_1\overline{x}_3 + \overline{x}_2\overline{x}_4 + \overline{x}_1\overline{x}_4 \\ &= (\overline{x}_1 + \overline{x}_2)(\overline{x}_3 + \overline{x}_4) \end{aligned}$$

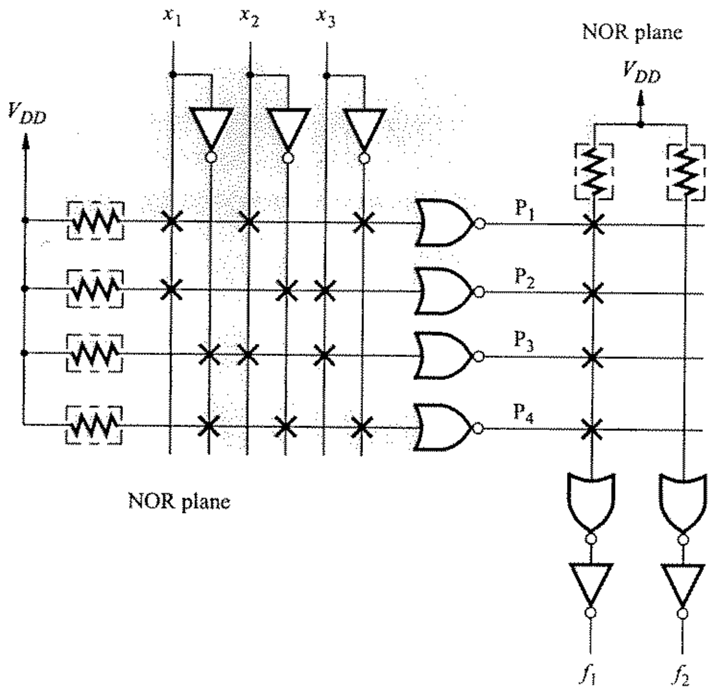
which leads to the circuit



3.13.



3.36.



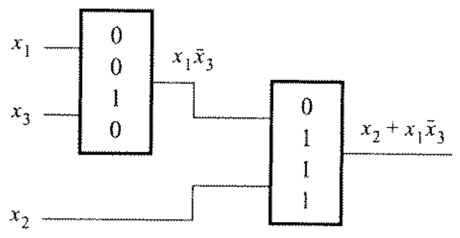
3.45. The canonical SOP for f is

$$f = \overline{x}_1 x_2 \overline{x}_3 + \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 \overline{x}_3 + x_1 x_2 \overline{x}_3 + x_1 x_2 x_3$$

This expression can be manipulated into

$$\begin{aligned} f &= \overline{x}_1 x_2 + x_1 \overline{x}_3 + x_1 x_2 \\ &= x_2 + x_1 \overline{x}_3 \end{aligned}$$

The circuit is



3.46. The canonical SOP for f is

$$f = x_1 x_2 x_4 + x_2 x_3 \overline{x}_4 + \overline{x}_1 \overline{x}_2 \overline{x}_3$$

This expression can be manipulated into

$$f = x_2 \cdot (x_1 x_4 + x_3 \overline{x}_4) + \overline{x}_2 \cdot (\overline{x}_1 \overline{x}_3)$$

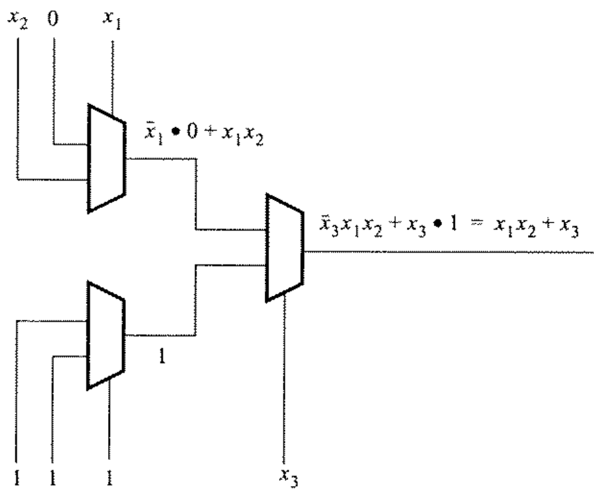
Using functional decomposition we have

$$f = x_2 f_1 + \overline{x}_2 f_2$$

where

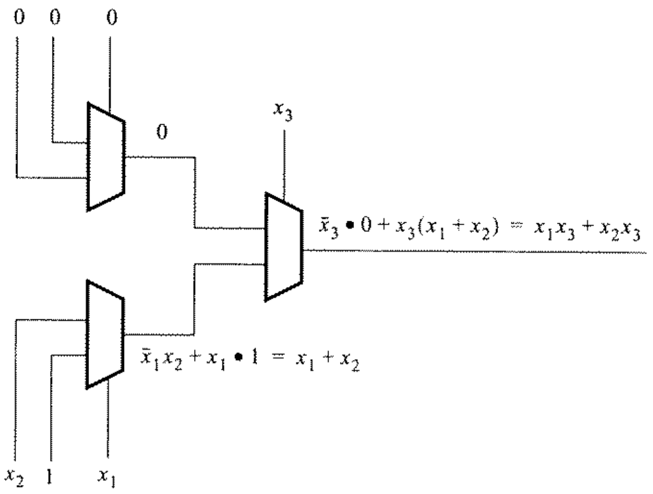
$$\begin{aligned} f_1 &= x_1 x_4 + x_3 \overline{x}_4 \\ f_2 &= \overline{x}_1 \overline{x}_3 \end{aligned}$$

3.49. (*a*)



x_2 0 x_1
$\bar{x}_1 \bullet 0 + x_1x_2$
$\bar{x}_3x_1x_2 + x_3 \bullet 1 = x_1x_2 + x_3$
1
1 1 1
x_3

(*b*)



0 0 0
0
x_3
$\bar{x}_3 \bullet 0 + x_3(x_1 + x_2) = x_1x_3 + x_2x_3$
$\bar{x}_1x_2 + x_1 \bullet 1 = x_1 + x_2$
x_2 1 x_1

4.1. SOP form: $f = \overline{x}_1x_2 + \overline{x}_2x_3$
POS form: $f = (\overline{x}_1 + \overline{x}_2)(x_2 + x_3)$

4.2. SOP form: $f = x_1\overline{x}_2 + x_1x_3 + \overline{x}_2x_3$
POS form: $f = (x_1 + x_3)(x_1 + \overline{x}_2)(\overline{x}_2 + x_3)$

4.3. SOP form: $f = \overline{x}_1x_2x_3\overline{x}_4 + x_1x_2\overline{x}_3x_4 + \overline{x}_2x_3x_4$
POS form: $f = (\overline{x}_1 + x_4)(x_2 + x_3)(\overline{x}_2 + \overline{x}_3 + \overline{x}_4)(x_2 + x_4)(x_1 + x_3)$

4.8. $f = \sum m(0, 7)$
$f = \sum m(1, 6)$
$f = \sum m(2, 5)$
$f = \sum m(0, 1, 6)$
$f = \sum m(0, 2, 5)$
etc.

4.11. The statement is false. As a counter example consider $f(x_1, x_2, x_3) = \sum m(0, 5, 7)$.
Then, the minimum-cost SOP form $f = x_1x_3 + \overline{x}_1\overline{x}_2\overline{x}_3$ is unique.
But, there are two minimum-cost POS forms:
$f = (x_1 + \overline{x}_3)(\overline{x}_1 + x_3)(x_1 + \overline{x}_2)$ and
$f = (x_1 + \overline{x}_3)(\overline{x}_1 + x_3)(\overline{x}_2 + x_3)$

4.13. If each circuit is implemented separately:

$f = \overline{x}_1 x_2 x_4 + x_2 x_4 x_5 + x_3 \overline{x}_4 \overline{x}_5 + \overline{x}_1 \overline{x}_2 \overline{x}_4 x_5$ Cost = 22

$g = \overline{x}_3 \overline{x}_5 + \overline{x}_4 \overline{x}_5 + \overline{x}_1 \overline{x}_2 \overline{x}_4 + \overline{x}_1 x_2 x_4 + x_2 x_4 x_5$ Cost = 24

In a combined circuit:

$f = \overline{x}_1 x_2 x_4 + x_2 x_4 x_5 + x_3 \overline{x}_4 \overline{x}_5 + \overline{x}_1 \overline{x}_2 \overline{x}_4 x_5$

$g = \overline{x}_1 x_2 x_4 + x_2 x_4 x_5 + x_3 \overline{x}_4 \overline{x}_5 + \overline{x}_1 \overline{x}_2 \overline{x}_4 x_5 + \overline{x}_3 \overline{x}_5$

The first 4 product terms are shared, hence the total cost is 31. Note that in this implementation $f \subseteq g$, thus g can be realized as $g = f + \overline{x}_3 \overline{x}_5$, in which case the total cost is lowered to 28.

5.1. (*a*) 478
(*b*) 743
(*c*) 2025
(*d*) 41567

5.3. (*a*) 478
(*b*) −281
(*c*) −2

5.4. The numbers are represented as follows:

Decimal	Sign and Magnitude	1's Complement	2's Complement
73	000001001001	000001001001	000001001001
1906	011101110010	011101110010	011101110010
−95	100001011111	111110100000	111110100001
−1630	111001011110	100110100001	100110100010

5.5. The results of the operations are:

$$(a):\ \begin{array}{r} 00110110 \\ \underline{+01000101} \\ 01111011 \end{array} \quad \begin{array}{r} 54 \\ \underline{+69} \\ 123 \end{array} \qquad (b):\ \begin{array}{r} 01110101 \\ \underline{+11011110} \\ 01010011 \end{array} \quad \begin{array}{r} 117 \\ \underline{-34} \\ 83 \end{array} \qquad (c):\ \begin{array}{r} 11011111 \\ \underline{+10111000} \\ 10010111 \end{array} \quad \begin{array}{r} (-33) \\ \underline{+(-72)} \\ (-105) \end{array}$$

$$(d):\ \begin{array}{r} 00110110 \\ \underline{-00101011} \\ 00001011 \end{array} \quad \begin{array}{r} 54 \\ \underline{-43} \\ 11 \end{array} \qquad (e):\ \begin{array}{r} 01110101 \\ \underline{-11010110} \\ 10011111 \end{array} \quad \begin{array}{r} (117) \\ \underline{-(-42)} \\ (159) \end{array} \qquad (f):\ \begin{array}{r} 11010011 \\ \underline{-11101100} \\ 11100111 \end{array} \quad \begin{array}{r} (-45) \\ \underline{-(-20)} \\ (-25) \end{array}$$

Arithmetic overflow occurs in example *e*; note that the pattern 10011111 represents −97 rather than +159.

5.9. Construct the truth table

x_{n-1}	y_{n-1}	c_{n-1}	c_n	s_{n-1} (sign bit)	Overflow
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

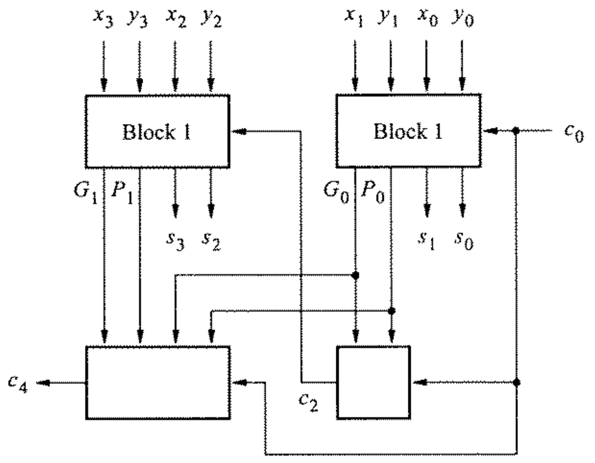
Note that overflow cannot occur when two numbers with opposite signs are added. From the truth table the overflow expression is

$$Overflow = \overline{c}_n c_{n-1} + c_n \overline{c}_{n-1} = c_n \oplus c_{n-1}$$

5.12. From Expression 5.4, each c_i requires i AND gates and one OR gate. Therefore, to determine all c_i signals we need $\sum_{i=1}^{n}(i+1) = (n^2+3n)/2$ gates. In addition to this, we need $3n$ gates to generate all g, p, and s functions. Therefore, a total of $(n^2+9n)/2$ gates are needed.

5.13. 84 gates.

5.14. The circuit for a 4-bit version of the adder based on the hierarchical structure in Figure 5.18 is constructed as follows:



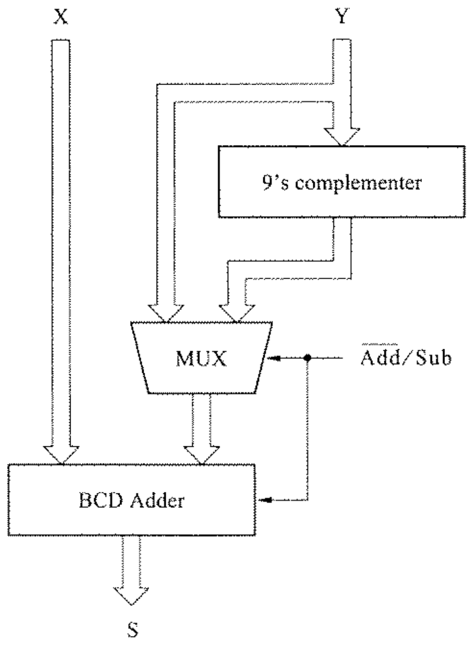
$$
\begin{aligned}
p_i &= x_i + y_i \\
g_i &= x_i y_i \\
P_0 &= p_1 p_0 \\
G_0 &= g_1 + p_1 g_0
\end{aligned}
$$

5.19. BCD subtraction can be performed using 10's complement representation, using an approach that is similar to 2's complement subtraction. Note that 10's and 2's complements are the radix complements in number systems where the radices are 10 and 2, respectively. Let X and Y be BCD numbers given in 10's complement representation, such that the sign (left-most) BCD digit is 0 for positive numbers and 9 for negative numbers. Then, the subtraction operation $S = X - Y$ is performed by finding the 10's complement of Y and adding it to X, ignoring any carry-out from the sign-digit position.

For example, let $X = 068$ and $Y = 043$. Then, the 10's complement of Y is 957, and $S' = 068 + 957 = 1025$. Dropping the carry-out of 1 from the sign-digit position gives $S = 025$.

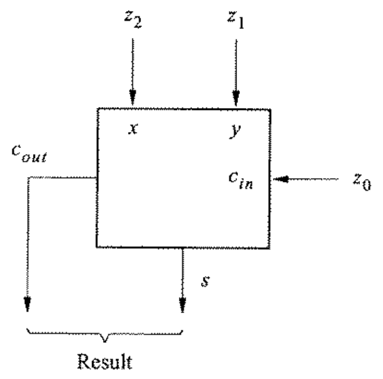
As another example, let $X = 032$ and $Y = 043$. Then, $S = 032 + 957 = 989$, which represents -11_{10}.

The 10's complement of Y can be formed by adding 1 to the 9's complement of Y. Therefore, a circuit that can add and subtract BCD operands can be designed as follows:

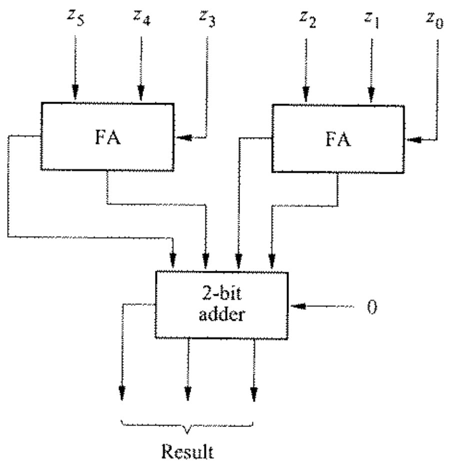


For the 9's complementer one can use the circuit designed in problem 5.18. The BCD adder is a circuit based on the approach illustrated in Figure 5.40.

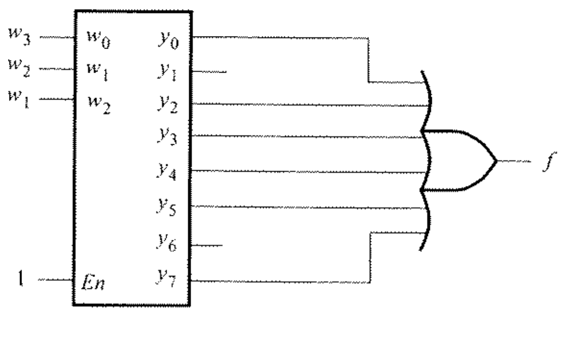
5.21. A full-adder circuit can be used, such that two of the bits of the number are connected as inputs x and y, while the third bit is connected as the carry-in. Then, the carry-out and sum bits will indicate how many input bits are equal to 1.



5.22. Using the approach explained in the solution to problem 5.21, the desired circuit can be built as follows:



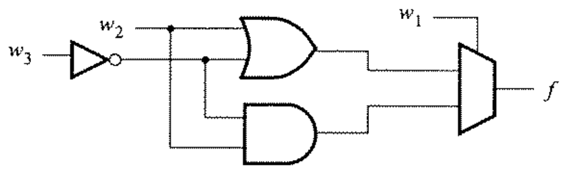
6.1.



6.3.

w_1	w_2	w_3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

w_1	f
0	$w_2 + \overline{w}_3$
1	$w_2\overline{w}_3$



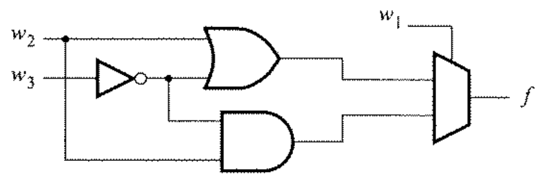
6.5. The function f can be expressed as

$$f = \overline{w}_1\overline{w}_2\overline{w}_3 + \overline{w}_1 w_2\overline{w}_3 + \overline{w}_1 w_2 w_3 + w_1 w_2\overline{w}_3$$

Expansion in terms of w_1 produces

$$f = \overline{w}_1(w_2 + \overline{w}_3) + w_1(w_2\overline{w}_3)$$

The corresponding circuit is



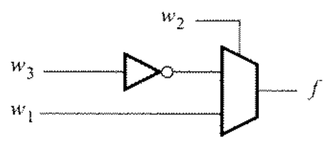
6.6. The function f can be expressed as

$$f = \overline{w}_1\overline{w}_2\overline{w}_3 + w_1\overline{w}_2\overline{w}_3 + w_1 w_2\overline{w}_3 + w_1 w_2 w_3$$

Expansion in terms of w_2 produces

$$f = \overline{w}_2(\overline{w}_3) + w_2(w_1)$$

The corresponding circuit is



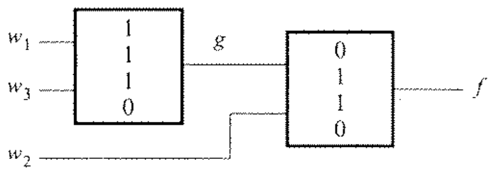
6.11. Expansion in terms of w_2 gives

$$f = \overline{w}_2(\overline{w}_1 + \overline{w}_3) + w_2(w_1 w_3)$$

Letting $g = \overline{w}_1 + \overline{w}_3$, we have

$$f = \overline{w}_2 g + w_2\overline{g}$$

The corresponding circuit is



6.13. Using Shannon's expansion in terms of w_2 we have

$$\begin{aligned} f &= \overline{w}_2(\overline{w}_3 + \overline{w}_1 w_4) + w_2(w_3\overline{w}_4 + w_1 w_3) \\ &= \overline{w}_2(\overline{w}_3 + \overline{w}_1 w_4) + w_2(w_3(w_1 + \overline{w}_4)) \end{aligned}$$

If we let $g = \overline{w}_3 + \overline{w}_1 w_4$, then

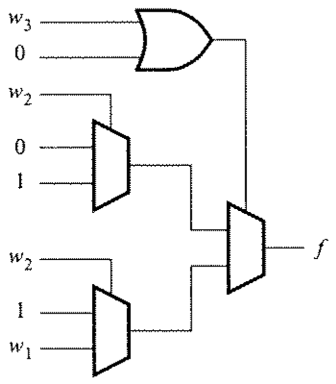
$$f = \overline{w}_2 g + w_2 \overline{g}$$

Thus, two 3-LUTs are needed to implement f.

6.16. Using Shannon's expansion in terms of w_3 we have

$$\begin{aligned} f &= \overline{w}_3(w_2) + w_3(w_1 + \overline{w}_2) \\ &= \overline{w}_3(w_2) + w_3(\overline{w}_2 + w_2 w_1) \end{aligned}$$

The corresponding circuit is



6.22. The code in Figure P6.3 will instantiate latches on the outputs of the decoder because the **if** statement does not specify all possibilities in a combinational circuit. It can be fixed by including the **else** clause

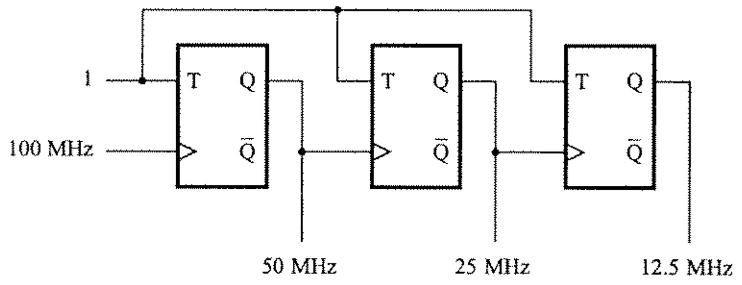
else Y[k] = 0;

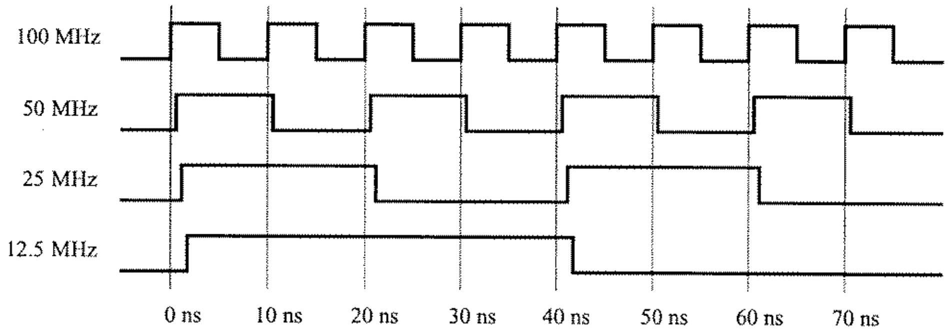
after the **if** clause.

6.24. An 8-to-3 priority encoder can be implemented using a **case** statement as follows:

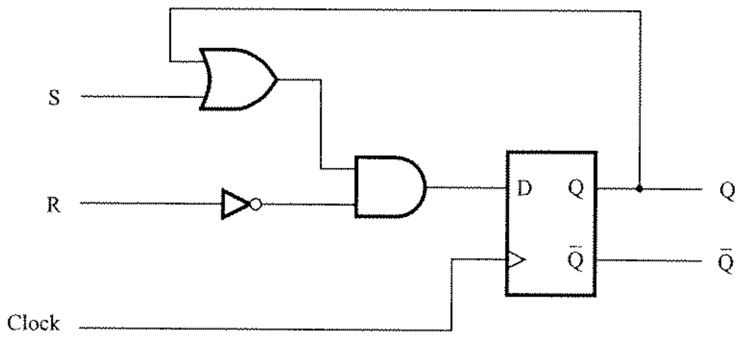
```
module prob6_24 (W, Y, z);
   input [7:0] W;
   output [2:0] Y;
   output z;
   reg [2:0] Y;
   reg z;

   always @(W)
   begin
      z = 1;
      case (W)
         8'b1xxxxxxx: Y = 7;
         8'b01xxxxxx: Y = 6;
         8'b001xxxxx: Y = 5;
         8'b0001xxxx: Y = 4;
         8'b00001xxx: Y = 3;
         8'b000001xx: Y = 2;
         8'b0000001x: Y = 1;
         8'b00000001: Y = 0;
         default:   begin
                       z = 0;
                       Y = 3'bx;
                    end
      endcase
endmodule
```

7.5.

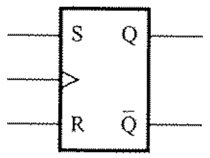




7.6.



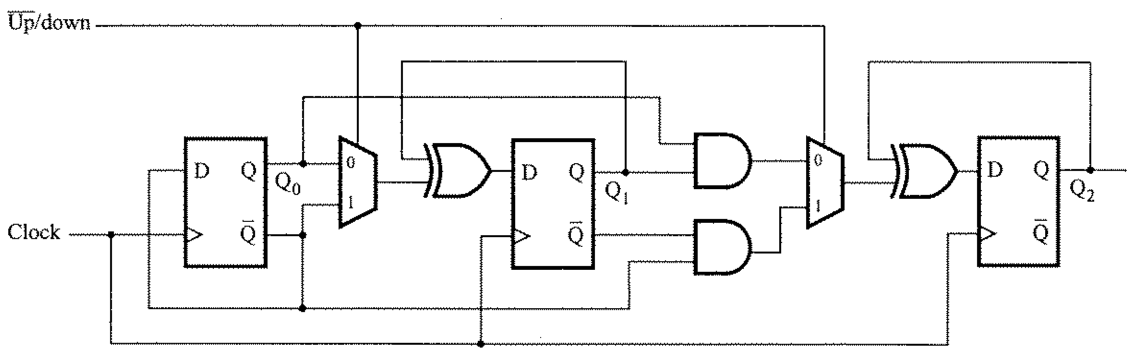
S	R	Q(t + 1)
0	0	Q(t)
0	1	0
1	0	1
1	1	0



7.13. Let $S = s_1s_0$ be a binary number that specifies the number of bit-positions by which to rotate. Also let L be a parallel-load input, and let $R = r_0r_1r_2r_3$ be parallel data. If the inputs to the flip-flops are $d_0 \ldots d_3$ and the outputs are $q_0 \ldots q_3$, then the barrel-shifter can be represented by the logic expressions

$$\begin{aligned}
d_0 &= L \cdot r_0 + \overline{L} \cdot (\bar{s}_1\bar{s}_0q_0 + \bar{s}_1s_0q_3 + s_1\bar{s}_0q_2 + s_1s_0q_1) \\
d_1 &= L \cdot r_1 + \overline{L} \cdot (\bar{s}_1\bar{s}_0q_1 + \bar{s}_1s_0q_0 + s_1\bar{s}_0q_3 + s_1s_0q_2) \\
d_2 &= L \cdot r_2 + \overline{L} \cdot (\bar{s}_1\bar{s}_0q_2 + \bar{s}_1s_0q_1 + s_1\bar{s}_0q_0 + s_1s_0q_3) \\
d_3 &= L \cdot r_3 + \overline{L} \cdot (\bar{s}_1\bar{s}_0q_3 + \bar{s}_1s_0q_2 + s_1\bar{s}_0q_1 + s_1s_0q_0)
\end{aligned}$$

7.17.



7.24.

```
// Ring counter with synchronous reset
module ripplen (Resetn, Clock, Q);
    parameter n = 8;
    input Resetn, Clock;
    output [n−1:0] Q;
    reg [n−1:0] Q;

    always @(posedge Clock)
        if (!Resetn)
        begin
            Q[7:1] <= 0;
            Q[0] <= 1;
        end
        else
            Q <= {{Q[6:0]}, {Q[7]}};
endmodule
```

7.34. With non-blocking assignments, the result of the assignment f <= A[1] & A[0] is not seen by the successive assignments inside the **for** loop. Thus, f has an uninitialized value when the **for** loop is entered. Similarly, each **for** loop interation sees the unitialized value of f. The result of the code is the sequential circuit specified by f = f | A[n-1] A[n-2].

8.1. The expressions for the inputs of the flip-flops are

$$D_2 = Y_2 = \overline{w}y_2 + \overline{y}_1\overline{y}_2$$
$$D_1 = Y_1 = w \oplus y_1 \oplus y_2$$

The output equation is

$$z = y_1 y_2$$

8.3. A possible state table is

Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	E	C	0	0
C	E	D	0	0
D	E	D	0	1
E	F	B	0	0
F	A	B	0	1

8.5. A minimal state table is

Present state	Next State $w=0$	Next State $w=1$	Output z
A	A	B	0
B	E	C	0
C	D	C	0
D	A	F	1
E	A	F	0
F	E	C	1

8.6. An initial attempt at deriving a state table may be

Present state	Next state $w=0$	Next state $w=1$	Output z $w=0$	Output z $w=1$
A	A	B	0	0
B	D	C	0	0
C	D	C	1	0
D	A	E	0	1
E	D	C	0	0

States B and E are equivalent; hence the minimal state table is

Present state	Next state $w=0$	Next state $w=1$	Output z $w=0$	Output z $w=1$
A	A	B	0	0
B	D	C	0	0
C	D	C	1	0
D	A	B	0	1

8.7. For Figure 8.51 have (using the straightforward state assignment):

	Present state $y_3y_2y_1$	Next state $w=0$ $Y_3Y_2Y_1$	Next state $w=1$ $Y_3Y_2Y_1$	Output z
A	000	001	010	1
B	001	011	101	1
C	010	101	100	0
D	011	001	110	1
E	100	101	010	0
F	101	100	011	0
G	110	101	110	0

This leads to

$$\begin{aligned}
Y_3 &= \overline{w}y_3 + \overline{y}_1y_2 + wy_1\overline{y}_3 \\
Y_2 &= wy_3 + w\overline{y}_1\overline{y}_2 + wy_1y_2 + \overline{w}y_1\overline{y}_2\overline{y}_3 \\
Y_1 &= \overline{y}_3\overline{w} + \overline{y}_1\overline{w} + wy_1\overline{y}_2 \\
z &= y_1\overline{y}_3 + \overline{y}_2\overline{y}_3
\end{aligned}$$

For Figure 8.52 have

	Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
A	0 0	0 1	1 0	1
B	0 1	0 0	1 1	1
C	1 0	1 1	1 0	0
F	1 1	1 0	0 0	0

This leads to

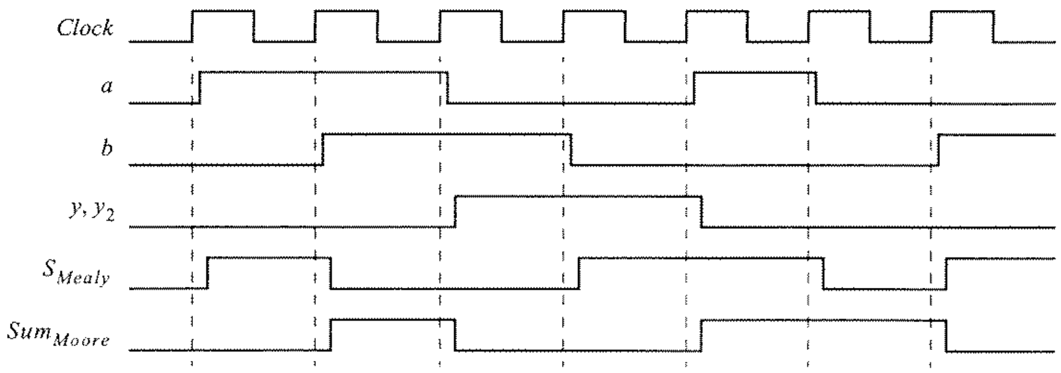
$$\begin{aligned} Y_2 &= \overline{w}y_2 + \overline{y}_1 y_2 + w\overline{y}_2 \\ Y_1 &= \overline{y}_1\overline{w} + wy_1\overline{y}_2 \\ z &= \overline{y}_2 \end{aligned}$$

Clearly, minimizing the number of states leads to a much simpler circuit.

8.12. A minimum state table is shown below. We assume that the 3-bit patterns do not overlap.

Present state	Next state $w=0$	Next state $w=1$	Output p
A	B	C	0
B	D	E	0
C	E	D	0
D	A	F	0
E	F	A	0
F	B	C	1

8.14. The timing diagram is



8.23. The state diagram is

Present state	Next state $w=0$	Next state $w=1$	Output $z_2z_1z_0$
A	A	B	000
B	B	C	001
C	C	D	010
D	D	E	011
E	E	F	100
F	F	A	101

The state-assigned table is

Present state $y_2y_1y_0$	Next state $w=0$ $Y_2Y_1Y_0$	Next state $w=1$ $Y_2Y_1Y_0$	Output $z_2z_1z_0$
000	000	001	000
001	001	010	001
010	010	011	010
011	011	100	011
100	100	101	100
101	101	000	101

The next-state expressions are

$$\begin{aligned} Y_2 &= \overline{y}_0y_2 + \overline{w}y_2 + wy_0y_1 \\ Y_1 &= \overline{y}_0y_1 + \overline{w}y_1 + wy_0\overline{y}_1\overline{y}_2 \\ Y_0 &= \overline{w}y_0 + w\overline{y}_0 \end{aligned}$$

The outputs are: $z_2 = y_2$, $z_1 = y_1$, and $z_0 = y_0$.

8.29. The next-state and output expressions are

$$\begin{aligned} D_1 &= Y_1 = w(y_1 + y_2) \\ D_2 &= Y_2 = w(\overline{y}_1 + \overline{y}_2) \\ z &= y_1\overline{y}_2 \end{aligned}$$

The corresponding state-assigned table is

Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
00	00	10	0
01	00	11	1
10	00	11	0
11	00	01	0

This leads to the state table

Present state	Next state $w=0$	Next state $w=1$	Output z
A	A	C	0
B	A	D	1
C	A	D	0
D	A	B	0

The circuit produces $z = 1$ whenever the input sequence on w comprises a 0 followed by an even number of 1s.