Chapter 5

5.1. (a) 478
(b) 743
(c) 2025
(d) 41567
(e) 61680

5.2. (a) 478
(b) -280
(c) -1

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):	00110110	54	(b):	01110101	117	(c):	11011111	(-33)
	+01000101	+69		+11011110	$-$ 34		+10111000	+(-72)
	01111011	123		01010011	83		10010111	(-105)
(d):	00110110	54	(e):	01110101	(117)	(f):	11010011	(-45)
	$-$00101011	-43		$-$11010110	$-(-$ 42)		$-$11101100	$-(-20)$
	00001011	11		10011111	(159)		11100111	(-25)

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

5.6. The associativity of the XOR operation can be shown as follows:

$$\begin{aligned} x \oplus (y \oplus z) &= x \oplus (\overline{y}z + y\overline{z}) \\ &= \overline{x}(\overline{y}z + y\overline{z}) + x(\overline{y} \cdot \overline{z} + yz) \\ &= \overline{x} \cdot \overline{y}z + \overline{x}y\overline{z} + x\overline{y} \cdot \overline{z} + xyz \end{aligned}$$

$$\begin{aligned} (x \oplus y) \oplus z &= (\overline{x}y + x\overline{y}) \oplus z \\ &= (\overline{x} \cdot \overline{y} + xy)z + (\overline{x}y + x\overline{y})\overline{z} \\ &= \overline{x} \cdot \overline{y}z + xyz + \overline{x}y\overline{z} + x\overline{y} \cdot \overline{z} \end{aligned}$$

The two SOP expressions are the same.

5.7. In the circuit of Figure 5.5b, we have:

$$\begin{aligned} s_i &= (x_i \oplus y_i) \oplus c_i \\ &= x_i \oplus y_i \oplus c_i \end{aligned}$$

$$\begin{aligned} c_{i+1} &= (x_i \oplus y_i)c_i + x_i y_i \\ &= (\overline{x}_i y_i + x_i \overline{y}_i)c_i + x_i y_i \\ &= \overline{x}_i y_i c_i + x_i \overline{y}_i c_i + x_i y_i \\ &= y_i c_i + x_i c_i + x_i y_i \end{aligned}$$

The expressions for s_i and c_{i+1} are the same as those derived in Figure 5.4*b*.

5.8. We will give a descriptive proof for ease of understanding. The 2's complement of a given number can be found by adding 1 to the 1's complement of the number. Suppose that the number has k 0s in the least-significant bit positions, $b_{k-1} \ldots b_0$, and it has $b_k = 1$. When this number is converted to its 1's complement, each of these k bits has the value 1. Adding 1 to this string of 1s produces $b_k b_{k-1} b_{k-2} \ldots b_0 = 100 \ldots 0$. This result is equivalent to copying the k 0s and the first 1 (in bit position b_k) encountered when the number is scanned from right to left. Suppose that the most-significant $n - k$ bits, $b_{n-1} b_{n-2} \ldots b_k$, have some pattern of 0s and 1s, but $b_k = 1$. In the 1's complement this pattern will be complemented in each bit position, which will include $b_k = 0$. Now, adding 1 to the entire n-bit number will make $b_k = 1$, but no further carries will be generated; therefore, the complemented bits in positions $b_{n-1} b_{n-2} \ldots b_{k+1}$ will remain unchanged.

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

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

5.10. Since $s_k = x_k \oplus y_k \oplus c_k$, it follows that

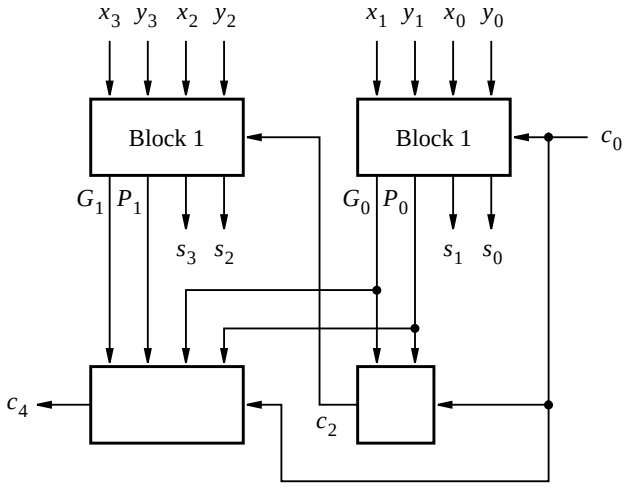
$$\begin{aligned} x_k \oplus y_k \oplus s_k &= (x_k \oplus y_k) \oplus (x_k \oplus y_k \oplus c_k) \\ &= (x_k \oplus y_k) \oplus (x_k \oplus y_k) \oplus c_k \\ &= 0 \oplus c_k \\ &= c_k \end{aligned}$$

5.11. Yes, it works. The NOT gate that produces c_i is not needed in stages where $i > 0$. The drawback is "poor" propagation of $\overline{c}_i = 1$ through the topmost NMOS transistor. The positive aspect is fewer transistors needed to produce $\overline{c}_{i+1}$.

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:

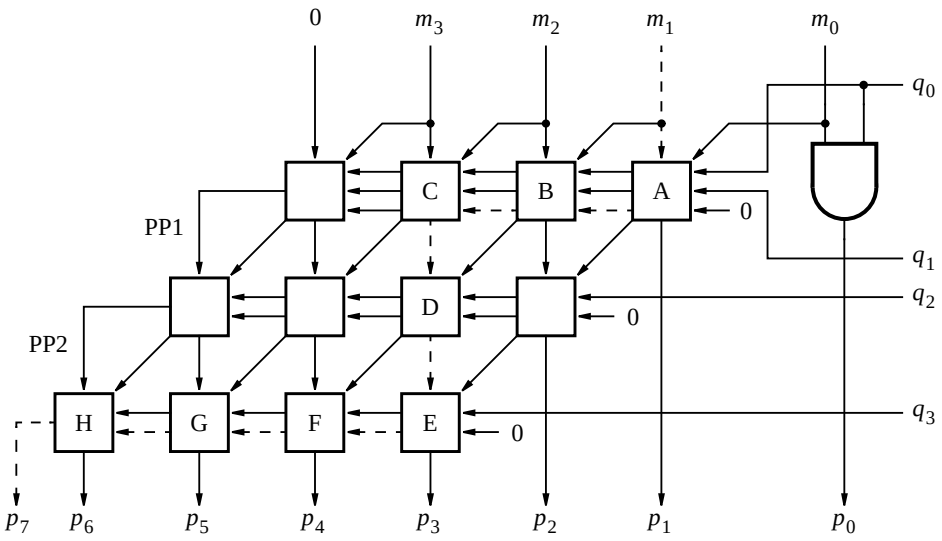


Blocks 0 and 1 have the structure similar to the circuit in Figure 5.16. The overall circuit is given by the expressions

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

$$
\begin{aligned}
P_1 &= p_3 p_2 \\
G_1 &= g_3 + p_3 g_2 \\
c_2 &= G_0 + P_0 c_0 \\
c_4 &= G_1 + P_1 G_0 + P_1 P_0 c_0
\end{aligned}
$$

5.15. The longest path, which causes the critical delay, is from the inputs m_0 and m_1 to the output p_7, indicated by the dashed path in the following copy of Figure 5.33a:



Propagation through the block A involves one gate delay in the AND gate shown in Figure 5.33b and two gate delays to generate the carry-out in the full-adder. Then, in each of the blocks B, C, D, E, F, G, and H, two more gate delays are needed to generate the carry-out signals in the circuits depicted by Figure 5.33c. Therefore, the total delay along the critical path is 17 gate delays.

5.16. The 4 × 4 multiplier in Figure 5.36 can be implemented as follows:

```
module  fig5_36 (M, Q, P);
   input  [3:0] M, Q;
   output  [7:0] P;
   wire  [3:1] Ctop, Csecond, Cbottom;
   wire  [5:2] PP1;
   wire  [6:3] PP2;

   assign  P[0] = M[0] & Q[0];
   fig5_36b  toprow_stage0  (M[1], M[0], Q[1], Q[0], 0, Ctop[1], P[1]);
   fig5_36b  toprow_stage1  (M[2], M[1], Q[1], Q[0], Ctop[1], Ctop[2], PP1[2]);
   fig5_36b  toprow_stage2  (M[3], M[2], Q[1], Q[0], Ctop[2], Ctop[3], PP1[3]);
   fig5_36b  toprow_stage3  (0, M[3], Q[1], Q[0], Ctop[3], PP1[5], PP1[4]);
   fig5_36c  secondrow_stage0  (PP1[2], M[0], Q[2], 0, Csecond[1], P[2]);
   fig5_36c  secondrow_stage1  (PP1[3], M[1], Q[2], Csecond[1], Csecond[2], PP2[3]);
   fig5_36c  secondrow_stage2  (PP1[4], M[2], Q[2], Csecond[2], Csecond[3], PP2[4]);
   fig5_36c  secondrow_stage3  (PP1[5], M[3], Q[2], Csecond[3], PP2[6], PP2[5]);
   fig5_36c  bottomrow_stage0  (PP2[3], M[0], Q[3], 0, Cbottom[1], P[3]);
   fig5_36c  bottomrow_stage1  (PP2[4], M[1], Q[3], Cbottom[1], Cbottom[2], P[4]);
   fig5_36c  bottomrow_stage2  (PP2[5], M[2], Q[3], Cbottom[2], Cbottom[3], P[5]);
   fig5_36c  bottomrow_stage3  (PP2[6], M[3], Q[3], Cbottom[3], P[7], P[6]);
endmodule

module  fig5_36b (m_k1, m_k, q1, q0, Cin, Cout, s);
   input  m_k1, m_k, q1, q0, Cin;
   output  Cout, s;
   wire x, y;

   assign x = m_k1 & q0;
   assign y = m_k & q1;
   fulladd FA (Cin, x, y, s, Cout);
endmodule

module  fig5_36c (ppi_k1, m_k, qj, Cin, Cout, s);
   input  ppi_k1, m_k, qj, Cin;
   output  Cout, s;
   wire y;

   assign y = m_k & qj;
   fulladd FA (Cin, ppi_k1, y, s, Cout);
endmodule

module  fulladd (Cin, x, y, s, Cout);
   input  Cin, x, y;
   output  s, Cout;
   reg s, Cout;

   always @(x or y or Cin)
      {Cout, s} = x + y + Cin;
endmodule
```

5.17. The code in Figure P5.2 represents a multiplier. It multiplies the lower two bits of *Input* by the upper two bits of *Input*, producing the four-bit *Output*. The style of code is poor, because it is not readily apparent what is being described.

5.18. Let $Y = y_3y_2y_1y_0$ be the 9's complement of the BCD digit $X = x_3x_2x_1x_0$. Then, Y is defined by the truth table

x_3	x_2	x_1	x_0	y_3	y_2	y_1	y_0
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0

This gives

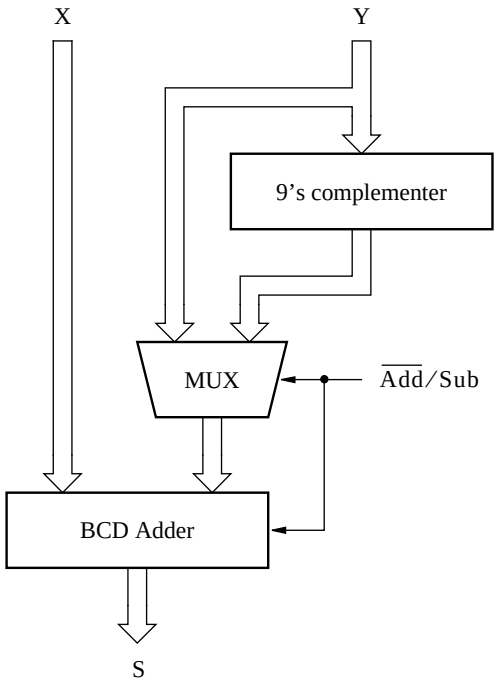
$$\begin{aligned} y_0 &= \overline{x}_0 \\ y_1 &= x_1 \\ y_2 &= \overline{x}_2 x_1 + x_2 \overline{x}_1 \\ y_3 &= \overline{x}_3 \overline{x}_2 \overline{x}_1 \end{aligned}$$

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:

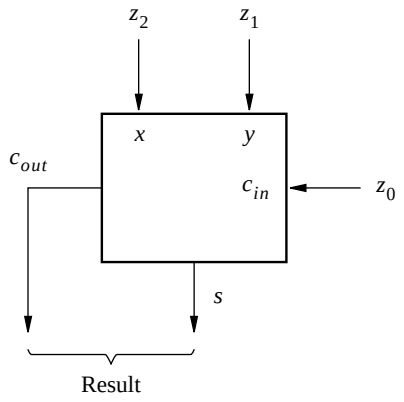


For the 9's complementer one can use the circuit designed in problem 5.18. The BCD adder is a circuit based on the approach illustrated in Figure 5.40.

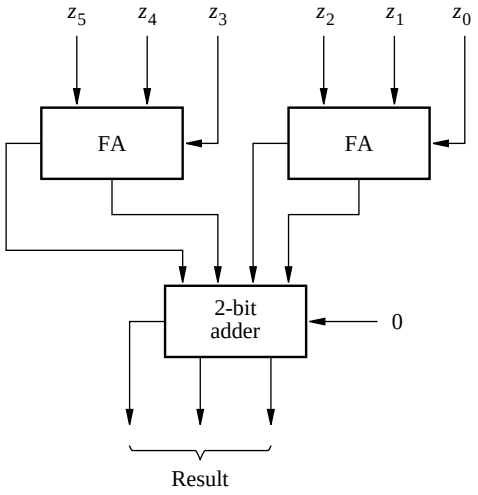
5.20. A possible Verilog code is

```
module bcdaddsubtract (A, B, D, Add_Sub, carryout);
   input [15:0] A, B;
   input Add_Sub;
   output [15:0] D;
   output carryout;
   reg [15:0] Bmux;
   wire [15:0] Bnot;
   wire [3:1] C;

   complement_digit dig0 (B[3:0], Bnot[3:0]);
   complement_digit dig1 (B[7:4], Bnot[7:4]);
   complement_digit dig2 (B[11:8], Bnot[11:8]);
   complement_digit dig3 (B[15:12], Bnot[15:12]);
   always @(B or Bnot or Add_Sub)
      if (Add_Sub == 0)  Bmux = B;
      else Bmux = Bnot;
   bcdadd stage0 (Add_Sub, A[3:0], Bmux[3:0], D[3:0], C[1]);
   bcdadd stage1 (C[1], A[7:4], Bmux[7:4], D[7:4], C[2]);
   bcdadd stage2 (C[2], A[11:8], Bmux[11:8], D[11:8], C[3]);
   bcdadd stage3 (C[3], A[15:12], Bmux[15:12], D[15:12], carryout);
endmodule

module complement_digit (W, Wnot);
   input [3:0] W;
   output [3:0] Wnot;

   assign Wnot[0] = ~W[0];
   assign Wnot[1] = W[1];
   assign Wnot[2] = (~W[2] & W[1]) | (W[2] & ~W[1]);
   assign Wnot[3] = ~W[3] & ~W[2] & ~W[1];
endmodule

module bcdadd (Cin, X, Y, S, Cout);
   input Cin;
   input [3:0] X, Y;
   output [3:0] S;
   output Cout;
   reg [3:0] S;
   reg Cout;
   reg [4:0] Z;

   always @(X or Y or Cin)
   begin
      Z = X + Y + Cin;
      if (Z < 10)  {Cout, S} = Z;
      else  {Cout, S} = Z + 6;
   end
endmodule
```

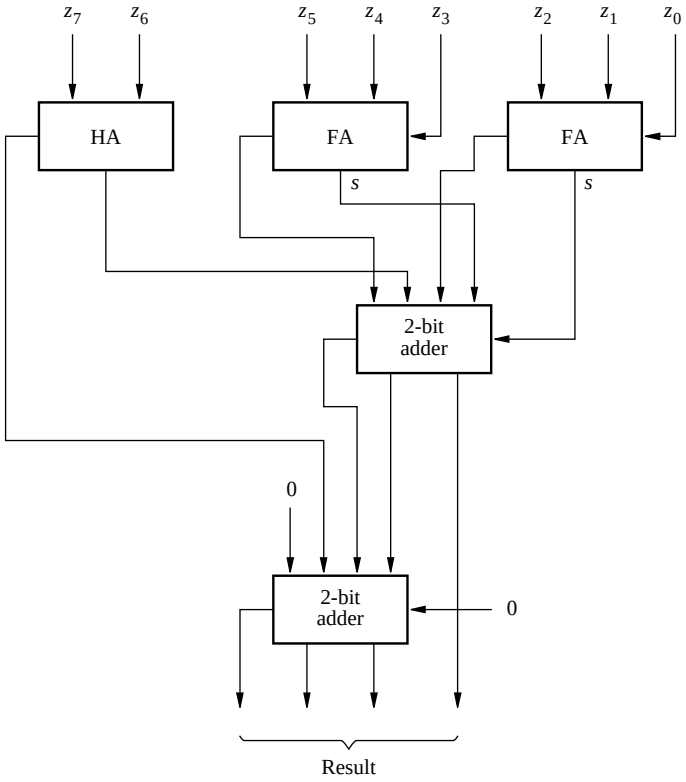
5.21. A full-adder circuit can be used, such that two of the bits of the number are connected as inputs x and y, while the third bit is connected as the carry-in. Then, the carry-out and sum bits will indicate how many input bits are equal to 1.



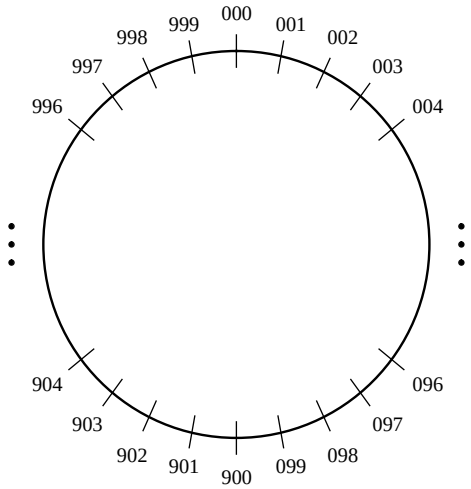
5.22. Using the approach explained in the solution to problem 5.21, the desired circuit can be built as follows:



5.23. Using the approach explained in the solutions to problems 5.21 and 5.22, the desired circuit can be built as follows:



5.24. The graphical representation is



For example, the addition $-3 + (+5) = 2$ involves starting at 997 (= -3) and going clockwise 5 numbers, which gives the result 002 (= +2). Similarly, the subtraction $4 - (+8) = -4$ involves starting at 004 (= +4) and going counterclockwise 8 numbers, which gives the result 996 (= -4).

5.25. The ternary half-adder in Figure P5.3 can be defined using binary-encoded signals as follows:

A		B		Carry	Sum	
a_1	a_0	b_1	b_0	c_{out}	s_1	s_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
1	0	0	0	0	1	0
1	0	0	1	1	0	0
1	0	1	0	1	0	1

The remaining 7 (out of 16) valuations, where either $a_1 = a_0 = 1$, or $b_1 = b_0 = 1$, can be treated as don't care conditions. Then, the minimum cost expressions are:

$$\begin{aligned} c_{out} &= a_0 b_1 + a_1 b_1 + a_1 b_0 \\ s_1 &= a_0 b_0 + \overline{a}_1 \overline{a}_0 b_1 + a_1 \overline{b}_1 \overline{b}_0 \\ s_0 &= a_1 b_1 + \overline{a}_1 \overline{a}_0 b_0 + a_0 \overline{b}_1 \overline{b}_0 \end{aligned}$$

5.26. Ternary full-adder is defined by the truth table:

c_{in}	A	B	c_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	0	2	0	2
0	1	0	0	1
0	1	1	0	2
0	1	2	1	0
0	2	0	0	2
0	2	1	1	0
0	2	2	1	1
1	0	0	0	1
1	0	1	0	2
1	0	2	1	0
1	1	0	0	2
1	1	1	1	0
1	1	2	1	1
1	2	0	1	0
1	2	1	1	1
1	2	2	1	2

Using binary-encoded signals for this full-adder gives the following truth table:

	A		B			Sum	
c_{in}	a_1	a_0	b_1	b_0	c_{out}	s_1	s_0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	1	0
0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	1
1	0	0	0	0	0	0	1
1	0	0	0	1	0	1	0
1	0	0	1	0	1	0	0
1	0	1	0	0	0	1	0
1	0	1	0	1	1	0	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	0	1	1	0

Treating the 14 (out of 32) valuations where either $a_1 = a_0 = 1$ or $b_1 = b_0 = 1$ as don't care conditions, leads to the minimum cost expressions

$$\begin{aligned} c_{out} &= a_0b_1 + a_1b_0 + a_1b_1 + a_1c_{in} + b_1c_{in} + a_0b_0c_{in} \\ s_1 &= a_0b_0\overline{c}_{in} + \overline{a}_1\overline{a}_0b_1\overline{c}_{in} + a_1\overline{b}_1\overline{b}_0\overline{c}_{in} + a_1b_1c_{in} + \overline{a}_1\overline{a}_0b_0c_{in} + a_0\overline{b}_1\overline{b}_0c_{in} \\ s_0 &= a_1b_1\overline{c}_{in} + \overline{a}_1\overline{a}_0b_0\overline{c}_{in} + a_0\overline{b}_1\overline{b}_0\overline{c}_{in} + a_1b_0c_{in} + a_0b_1c_{in} + \overline{a}_1\overline{a}_0\overline{b}_1\overline{b}_0c_{in} \end{aligned}$$

5.27. The subtractions $26 - 27 = 99$ and $18 - 34 = 84$ make sense if the two-digit numbers 00 to 99 are interpreted so that the numbers 00 to 49 are positive integers from 0 to +49, while the numbers 50 to 99 are negative integers from -50 to -1. This scheme can be illustrated graphically as follows: