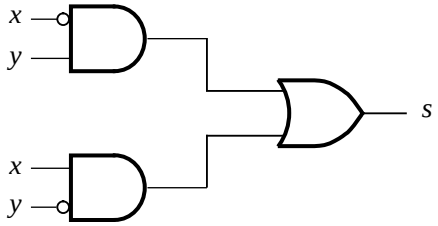
Chapter 9
Arithmetic

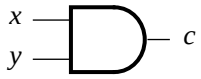
9.1. (a) The half adder is implemented as:

x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

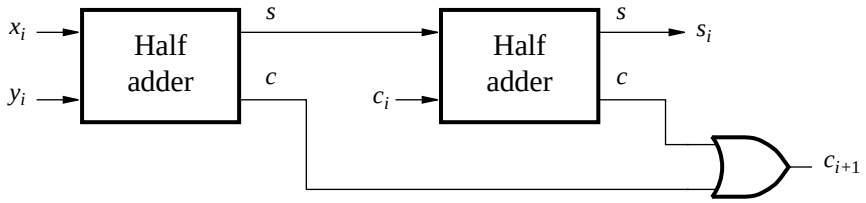


$s = x \oplus y$

$c = x\,y$



(b)



(c) The longest path through the circuit in part (b) is 4 gate delays (not including input inversions) in producing s_i; and the longest path through the circuit in Figure 9.2a is 2 gate delays (not including input inversions) in producing either c_i or s_i, assuming that s_i is implemented as a two-level AND-OR circuit.

9.2. The two ternary representations are given as follows:

Sign-and-magnitude	3's-complement
+11011	011011
−10222	212001
+2120	002120
−1212	221011
+10	000010
−201	222022

9.3. Ternary numbers with addition and subtraction operations:

Decimal Sign-and-magnitude	Ternary Sign-and-magnitude	Ternary 3's-complement
56	+2002	002002
−37	−1101	221122
122	11112	011112
−123	−11120	211110

Addition operations:

002002	002002	002002
+ 221122	+ 011112	+ 211110
000201	020121	220112
221122	221122	011112
+ 011112	+ 211110	+ 211110
010011	210002	222222

Subtraction operations:

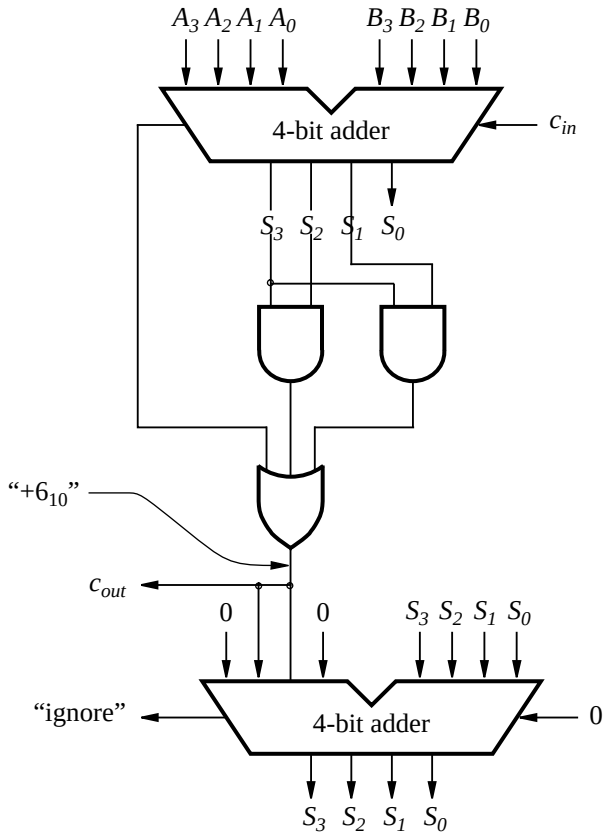
002002	002002
− 221122	+ 001101
	010110
002002	002002
− 011112	+ 211111
	220120
002002	002002
− 211110	+ 011120
	020122
221122	221122
− 011112	+ 211111
	210010
221122	221122
− 211110	+ 011120
	010012
011112	011112
− 211110	+ 011120
	100002
	overflow

9.4. (a) The output carry is 1 when $A + B \geq 10$. This is the condition that requires the further addition of 6_{10}.

(b)

(1)	0101		5
	+ 0110		+ 6
	1011	$> 10_{10}$	11
	+ 0110		
	0001	output carry = 1	
(2)	0011		3
	+ 0100		+ 4
	0111	$< 10_{10}$	7

(c)



9.5. Consider the truth table in Figure 9.1 for the case $i = n-1$, that is, for the sign bit position. Overflow occurs only when x_{n-1} and y_{n-1} are the same and s_{n-1} is different. This occurs in the second and seventh rows of the table; and c_n and c_{n-1} are different only in those rows. Therefore, $c_n \oplus c_{n-1}$ is a correct indicator of overflow.

9.6. Four 4-bit adders require $4 \times 31 = 124$ gates, and the carry-lookahead logic block requires 19 gates because it has the same structure as the lookahead block in Figure 9.4. Total gate count is thus 143. However, we should subtract $4 \times 5 = 20$ gates from this total corresponding to the logic for c_4, c_8, c_{12}, and c_{16}, that is in the 4-bit adders but which is replaced by the lookahead logic in Figure 9.5. Therefore, total gate count for the 16-bit adder is 143 − 20 = 123 gates.

9.7. (a) The additional logic is defined by the logic expressions:

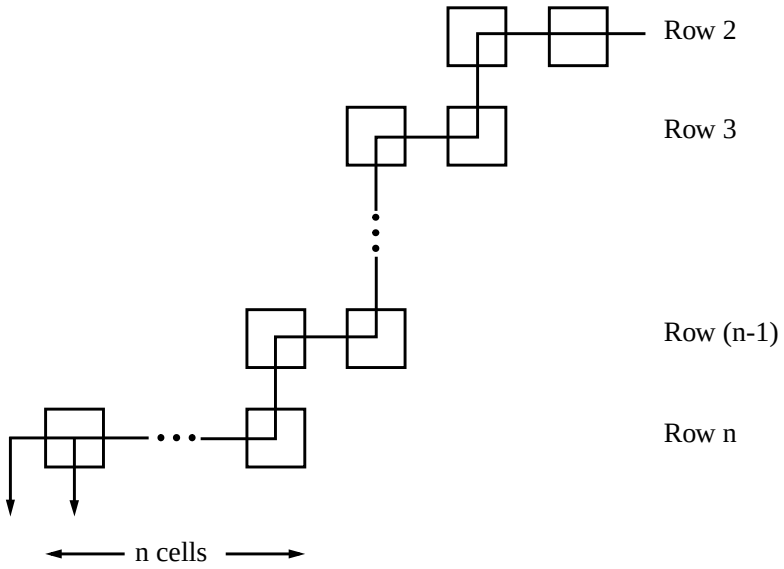
$$\begin{aligned}
c_{16} &= G_0^{II} + P_0^{II} c_0 \\
c_{32} &= G_1^{II} + P_1^{II} G_0^{II} + P_1^{II} P_0^{II} c_0 \\
c_{48} &= G_2^{II} + P_2^{II} G_1^{II} + P_2^{II} P_1^{II} G_0^{II} + P_2^{II} P_1^{II} P_0^{II} c_0 \\
c_{64} &= G_3^{II} + P_3^{II} G_2^{II} + P_3^{II} P_2^{II} G_1^{II} + P_3^{II} P_2^{II} P_1^{II} G_0^{II} + P_3^{II} P_2^{II} P_1^{II} P_0^{II} c_0
\end{aligned}$$

This additional logic is identical in form to the logic inside the lookahead circuit in Figure 9.5. (Note that the outputs c_{16}, c_{32}, c_{48}, and c_{64}, produced by the 16-bit adders are not needed because those outputs are produced by the additional logic.)

(b) The inputs G_i^{II} and P_i^{II} to the additional logic are produced after 5 gate delays, the same as the delay for c_{16} in Figure 9.5. Then all outputs from the additional logic, including c_{64}, are produced 2 gate delays later, for a total of 7 gate delays. The carry input c_{48} to the last 16-bit adder is produced after 7 gate delays. Then c_{60} into the last 4-bit adder is produced after 2 more gate delays, and c_{63} is produced after another 2 gate delays inside that 4-bit adder. Finally, after one more gate delay (an XOR gate), s_{63} is produced with a total of 7 + 2 + 2 + 1 = 12 gate delays.

(c) The variables s_{31} and c_{32} are produced after 12 and 7 gate delays, respectively, in the 64-bit adder. These two variables are produced after 10 and 7 gate delays in the 32-bit adder, as shown in Section 9.2.1.

9.8. As noted in Section 9.3.1, no full adders are needed in the first row of the array because the incoming partial product PP0 is zero. The worst case delay path is shown in the following figure:



Each of the two FA blocks in rows 2 through $n-1$ introduces 2 gate delays, for a total of $4(n-2)$ gate delays. Row n introduces $2n$ gate delays. Adding in the initial AND gate delay for row 1 and all other cells, total delay is:

$$4(n-2)+2n+1 = 6n-8+1 = 6(n-1)-1$$

9.9. Part (d) below is the answer to part (c) of the problem. Ignore part (c) below. It is used in Example 9.2 in Chapter 9.

(a)

$$\begin{array}{r} 010111 \\ \times\, 110110 \\ \hline \end{array} \qquad \begin{array}{r} +23 \\ \times\ -10 \\ \hline -230 \end{array}$$

$$\begin{array}{rcccccccccccc} & & & & & & & 0 & 1 & 0 & 1 & 1 & 1 \\ & & & & & & \times & 0 & -1 & +1 & 0 & -1 & 0 \\ \hline & & & & & & & & & & & & 0 \\ & & & & & & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ & & & & & & & & & & & 0 & \\ \text{sign} & & & & & 0 & 0 & 0 & 1 & 1 & 1 & & \\ \text{extension} & & & & 1_1 & 1_1 & 1_1 & 0_2 & 1_1 & 0_1 & 0_1 & 1 & \\ \hline & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}$$

(b)

$$\begin{array}{r} 110011 \\ \times\, 101100 \\ \hline \end{array} \qquad \begin{array}{r} -13 \\ \times\ -20 \\ \hline 260 \end{array}$$

$$\begin{array}{rcccccccccccc} & & & & & & & 1 & 1 & 0 & 0 & 1 & 1 \\ & & & & & & \times & -1 & +1 & 0 & -1 & 0 & 0 \\ \hline & & & & & & & & & & & & 0 \\ & & & & & & & & & & & 0 & \\ & & & & & 0 & 0 & 0 & 0 & 0 & 1 & & \\ \text{sign} & & & & & & & & & & 0 & & \\ \text{extension} & & & & 1 & 1 & 1 & 1 & 0 & 0 & 1 & & \\ & & & 0_1 & 0_1 & 0_1 & 1_1 & 1_1 & 0_2 & 1_1 & & & \\ \hline & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}$$

(c)

$$\begin{array}{r} 110101 \\ \times\, 011011 \\ \hline \end{array} \qquad \begin{array}{r} -11 \\ \times\ \ 27 \\ \hline -297 \end{array}$$

$$\begin{array}{rcccccccccccc} & & & & & & & 1 & 1 & 0 & 1 & 0 & 1 \\ & & & & & & \times & +1 & 0 & -1 & +1 & 0 & -1 \\ \hline & & & & & & & & & & & & 0 \\ & & & & & & & & & & & 0 & \\ & & & & & & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ \text{sign} & & & & & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{1} & 1 & 1 \\ \text{extension} & & & & & & & & & & 0 & & \\ & & & & 1_1 & 1_1 & 1_1 & 0_1 & 1_1 & 0_1 & 1_1 & 1_1 & \\ \hline & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}$$

(d)

$$\begin{array}{r} 001111 \\ \times\, 001111 \\ \hline \end{array} \qquad \begin{array}{r} 15 \\ \times\ 15 \\ \hline 225 \end{array}$$

$$\begin{array}{rcccccccccccc} & & & & & & & 0 & \mathbf{0} & 1 & 1 & 1 & 1 \\ & & & & & & \times & 0 & +1 & 0 & 0 & 0 & -1 \\ \hline & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & & & & \\ \hline & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}$$

9.10. Part (d) below is the answer to part (c) of the problem. Ignore part (c) below. It is used in Example 9.2 in Chapter 9.

(a)

$$\begin{array}{r} 010111 \\ \times\ 110110 \\ \hline \end{array}$$

$$\begin{array}{cccccc|cccccc} & & & & & & 0 & 1 & 0 & 1 & 1 & 1 \\ & & & & & & & -1 & & +2 & & -2 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & & & \\ 1_1 & 1_1 & 1_1 & 0_2 & 1_1 & 0_1 & 0_1 & 1 & & & & & \\ \hline 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}$$

(b)

$$\begin{array}{r} 110011 \\ \times\ 101100 \\ \hline \end{array}$$

$$\begin{array}{cccccc|cccccc} & & & & & & 1 & 1 & 0 & 0 & 1 & 1 \\ & & & & & & & -1 & & -1 & & 0 \\ \hline & & & & & & & & & & & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & & \\ 0 & 0 & 0 & 0_1 & 1_1 & 1_1 & 0_1 & 1 & & & & & \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}$$

(c)

$$\begin{array}{r} 110101 \\ \times\ 011011 \\ \hline \end{array}$$

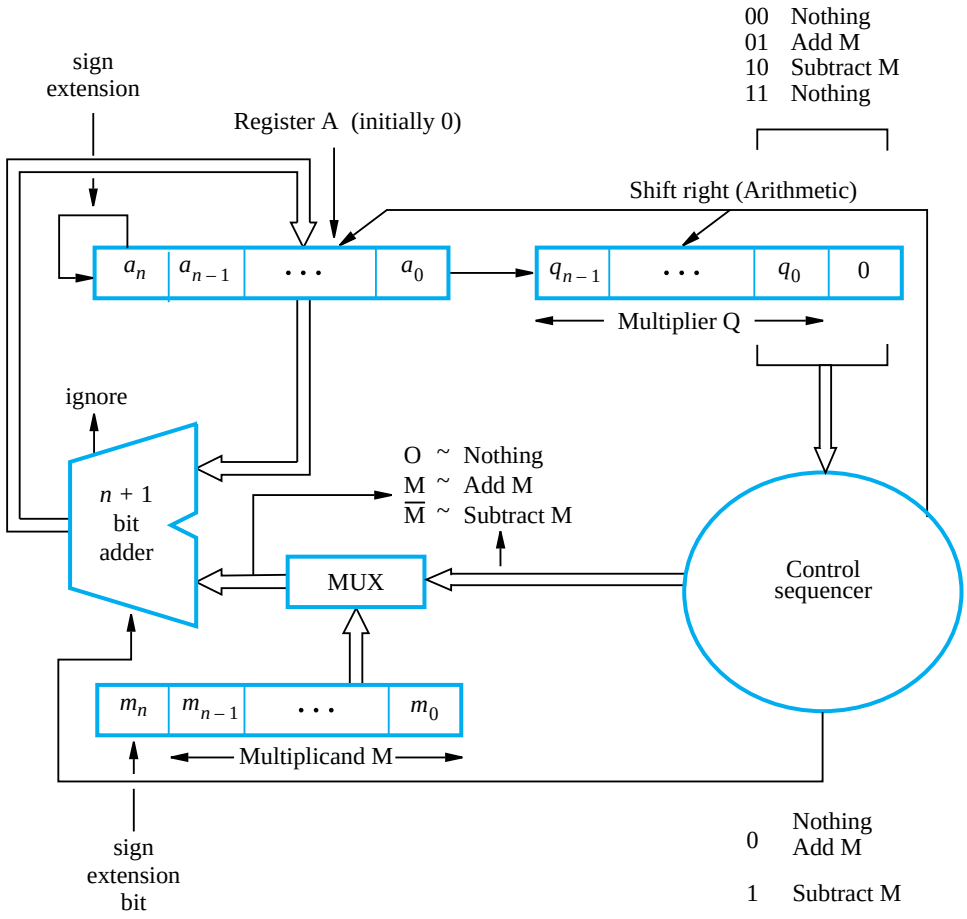
$$\begin{array}{cccccc|cccccc} & & & & & & 1 & 1 & 0 & 1 & 0 & 1 \\ & & & & & & & +2 & & -1 & & -1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & & \\ 1 & 1 & 1 & 0 & 1 & 0_1 & 1 & & 1 & & & & \\ \hline 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}$$

(d)

$$\begin{array}{r} 001111 \\ \times\ 001111 \\ \hline \end{array}$$

$$\begin{array}{cccccc|cccccc} & & & & & & 0 & 0 & 1 & 1 & 1 & 1 \\ & & & & & & & +1 & & & & -1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & & & & \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}$$

9.11. Both the A and M registers are augmented by one bit to the left to hold a sign extension bit. The adder is changed to an $n + 1$-bit adder. A bit is added to the right end of the Q register to implement the Booth multiplier recoding operation. It is initially set to zero. The control logic decodes the two bits at the right end of the Q register according to the Booth algorithm. The right shift is an arithmetic right shift as indicated by the repetition of the extended sign bit at the left end of the A register.



Multiplier 3-bit recoding			Multiplier bit on the right	Multiplicand selected at position
$i+2$	$i+1$	i	$i-1$	i
0	0	0	0	$+0 \times \mathrm{M}$
0	0	0	1	$+1 \times \mathrm{M}$
0	0	1	0	$+1 \times \mathrm{M}$
0	0	1	1	$+2 \times \mathrm{M}$
0	1	0	0	$+2 \times \mathrm{M}$
0	1	0	1	$+3 \times \mathrm{M}$
0	1	1	0	$+3 \times \mathrm{M}$
0	1	1	1	$+4 \times \mathrm{M}$
1	0	0	0	$-4 \times \mathrm{M}$
1	0	0	1	$-3 \times \mathrm{M}$
1	0	1	0	$-3 \times \mathrm{M}$
1	0	1	1	$-2 \times \mathrm{M}$
1	1	0	0	$-2 \times \mathrm{M}$
1	1	0	1	$-1 \times \mathrm{M}$
1	1	1	0	$-1 \times \mathrm{M}$
1	1	1	1	$+0 \times \mathrm{M}$

The versions $+3 \times \mathrm{M}$ and $-3 \times \mathrm{M}$ of the multiplicand cannot be generated by shifting and/or negating the multiplicand. These versions are required for four cases as shown in the table. This limits the practicality of recoding 3 bits of the multiplier at a time.

9.13 (a)

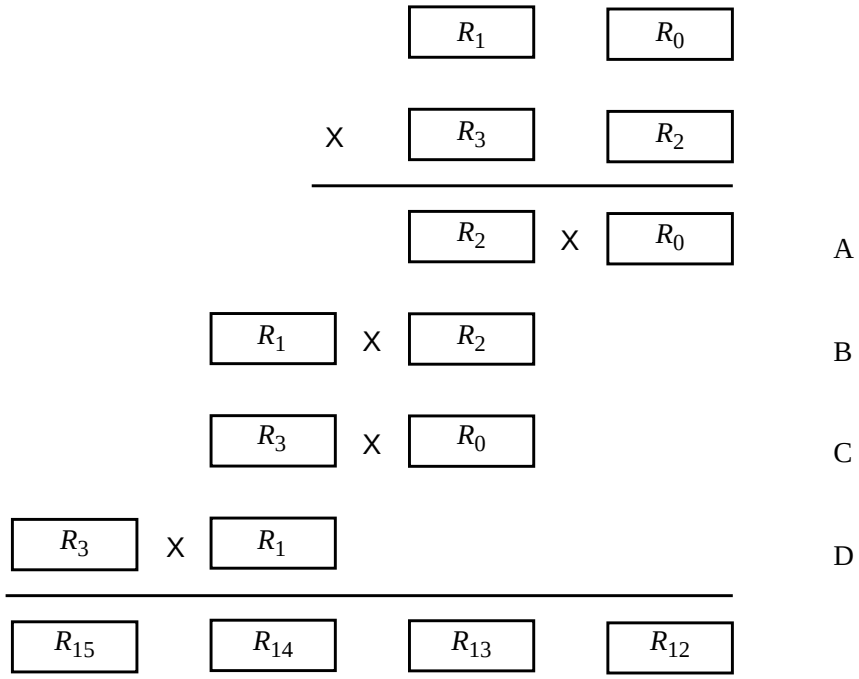
1110	-2
$\times$ 1101	$\times -3$
1110	6
0000	
1000	
0000	
0110	

(b)

0010	2
$\times$ 1110	$\times -2$
0000	-4
0100	
1000	
0000	
1100	

This technique works correctly for the same reason that modular addition can be used to implement signed-number addition in the 2's-complement representation, because multiplication can be interpreted as a sequence of additions of the multiplicand to shifted versions of itself.

9.14. The four 32-bit subproducts needed to generate the 64-bit product are labeled A, B, C, and D, and shown in their proper shifted positions in the following figure:



The 64-bit product is the sum of A, B, C, and D. Using register transfers and multiplication and addition operations executed by the arithmetic unit described, the 64-bit product is generated without using any extra registers by the following steps:

$$
\begin{aligned}
R_{12} &\leftarrow [R_0] \\
R_{13} &\leftarrow [R_2] \\
R_{14} &\leftarrow [R_1] \\
R_{15} &\leftarrow [R_3] \\
R_3 &\leftarrow [R_{14}] \\
R_1 &\leftarrow [R_{15}] \\
R_{13}, R_{12} &\leftarrow [R_{13}] \times [R_{12}] \\
R_{15}, R_{14} &\leftarrow [R_{15}] \times [R_{14}] \\
R_3, R_2 &\leftarrow [R_3] \times [R_2] \\
R_1, R_0 &\leftarrow [R_1] \times [R_0] \\
R_{13} &\leftarrow [R_2] \text{ Add } [R_{13}] \\
R_{14} &\leftarrow [R_3] \text{ Add with carry } [R_{14}] \\
R_{15} &\leftarrow 0 \text{ Add with carry } [R_{15}] \\
R_{13} &\leftarrow [R_0] \text{ Add } [R_{13}] \\
R_{14} &\leftarrow [R_1] \text{ Add with carry } [R_{14}] \\
R_{15} &\leftarrow 0 \text{ Add with carry } [R_{15}]
\end{aligned}
$$

This procedure destroys the original contents of the operand registers. Steps 5 and 6 result in swapping the contents of R_1 and R_3 so that subproducts B and C can be computed in adjacent register pairs. Steps 11, 12, and 13, add the subproduct B into the 64-bit product registers; and steps 14, 15, and 16, add the subproduct C into these registers.

9.15. (a) The worst case delay path in the ripple-carry array in Figure 9.16(a) is along the staircase pattern that includes the two FA blocks at the right end of each of the first two rows (a total of four FA block delays), followed by the four FA blocks in the third row. Total delay is therefore 8 FA block delays, ignoring the initial AND gate delay to develop all product bits.

In the carry-save array in Figure 9.16(b), the worst case delay path is vertically through the first two rows, followed by the four FA blocks in the third row, for a total of 6 FA block delays, ignoring the initial AND gate delay to develop all product bits.

(b) For each of the array forms in Figure 9.16, there are $n-2$ rows of FA blocks before the final row of n FA blocks. Therefore, total FA block delay in the ripple-carry array is

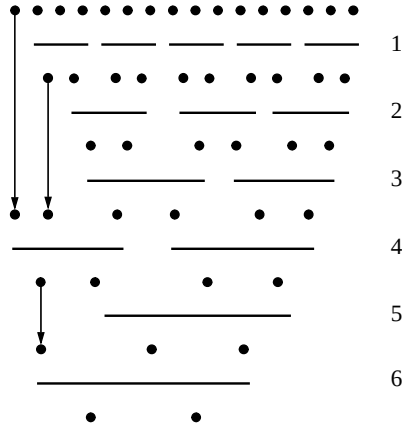
$$2(n-2)+n=3n-4$$

and in the carry-save array it is

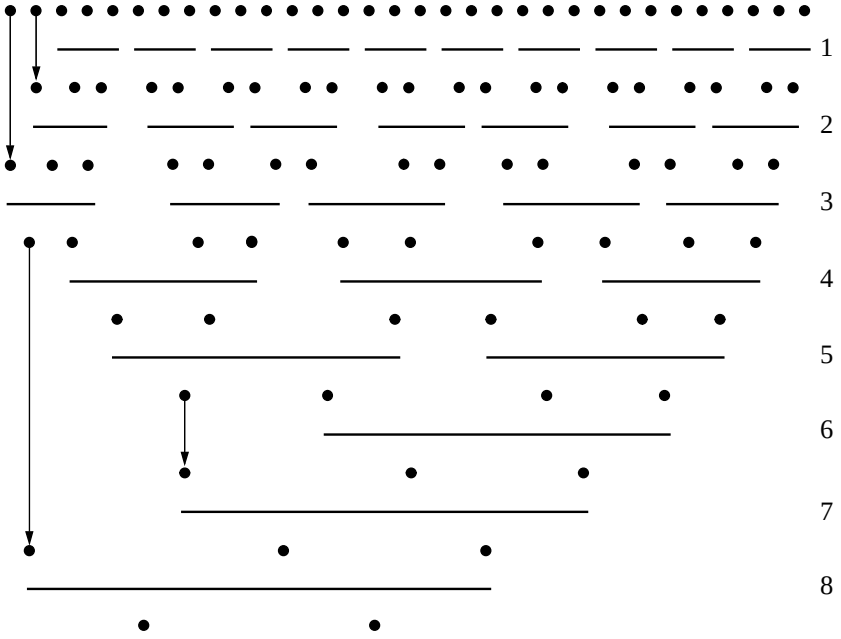
$$(n-2)+n=2n-2$$

For the 32×32 case, total FA block delay in the ripple-carry array is 92; and in the carry-save array it is 62.

9.16. (a) Six 3-2 reduction levels are needed:



(b) Eight 3-2 reduction levels are needed:



(c) The approximation gives 5.1 and 6.8 reduction levels, compared to 6 and 8 from parts (a) and (b).

9.17. (a) Using a pattern similar to that shown in Figure 9.19, it is seen that four 7-3 reduction levels are needed, as compared to seven 3-2 reduction levels.

(b) Let L be the number of levels needed to reduce k summands to 3. Following the method used in Example 9.3 in Section 9.10, we have

$$k(3/7)^L = 3$$

Taking logarithms to the base 2, we derive

$$log_2 k + L(log_2 3 - log_2 7) = log_2 3$$

$$log_2 k + L(1.59 - 2.81) = 1.59$$

$$L = (log_2 k - 1.59)/1.22 = 0.82 log_2 k - 1.3$$

For $k = 32$, $L = 2.8$. The discrepancy of this approximation from the actual value of 4 is due to the irregularities of the 7-3 reductions at the left side of the reduction tree, similar to the irregularities at the left side of the reduction tree in Figure 9.19.

9.18. A cascade connection of two 3-2 reducers can implement a 4-2 reducer. Using the input and output variable notation of Figure 9.20, the 4-2 reducer is implemented as follows:

Input variables w, x, and y, are applied as the three inputs to the first 3-2 reducer. Its carry-out is c_{out} for the 4-2 reducer. Its sum output is applied as an input to the second 3-2 reducer; and the other two inputs to that reducer are z and c_{in}. The carry-out and sum outputs from the second 3-2 reducer are the outputs c and s, respectively, for the 4-2 reducer.

9.19. The solutions, including decimal equivalent checks, are:

$$
\begin{array}{rcll}
\text{B} & = & 00101 & (5) \\
\times \text{A} & = & 10101 & \times (21) \\
\hline
 & & 00101 & (105) \\
 & & 0 & \\
 & 00101 & & \\
 & 001010 & & \\
\hline
 & 001101001 & & (105)
\end{array}
$$

$$
\begin{array}{rr}
100 & 4 \\
\hline
00101 \,\big)\overline{10101} & 5 \,\big)\overline{21} \\
\underline{101} & \underline{20} \\
00001 & 1
\end{array}
$$

9.20. The multiplication and division charts are:

$A \times B$:

C	A	Q	M	
			00101	
0	00000	10101		Initial configuration
0	00101	10101		1st cycle
0	00010	11010		
0	00010	11010		2nd cycle
0	00001	01101		
0	00110	01101		3rd cycle
0	00011	00110		
0	00011	00110		4th cycle
0	00001	10011		
0	00110	10011		5th cycle
0	00011	01001		
	product			

A / B :

	A	Q	M	
	000000	10101		
			000101	Initial configuration
shift	000001	0 1 0 1 □		
subtract	111011			1st cycle
	111100	0 1 0 1 0		
shift	111000	1 0 1 0 □		
add	000101			2nd cycle
	111101	1 0 1 0 0		
shift	111011	0 1 0 0 □		
add	000101			3rd cycle
	000000	0 1 0 0 1		
shift	000000	1 0 0 1 □		
subtract	111011			4th cycle
	111011	1 0 0 1 0		
shift	110111	0 0 1 0 □		
add	000101			5th cycle
	111100	0 0 1 0 0		
add	000101	quotient		
	000001			
	remainder			

9.21. (a)

+1.7	0	01111	101101
-0.012	1	01000	100010
+19	0	10011	001100
1/8	0	01100	000000

"Rounding" has been used as the truncation method in these answers.

(b) Other than exact 0 and $\pm$infinity, the smallest numbers are $\pm 1.000000 \times 2^{-14}$ and the largest numbers are $\pm 1.111111 \times 2^{15}$.

(c) Assuming sign-and-magnitude format, the smallest and largest integers (other than 0) are ± 1 and $\pm(2^{11} - 1)$; and the smallest and largest fractions (other than 0) are $\pm 2^{-11}$ and approximately ± 1.

(d)

$$\begin{aligned} A + B &= 0\ 10000\ 000000 \\ A - B &= 0\ 10000\ 110110 \\ A \times B &= 1\ 10000\ 001011 \\ A/B &= 1\ 10000\ 101110 \end{aligned}$$

"Rounding" has been used as the truncation method in these answers.

9.22. (a) Shift the mantissa of B right two positions, and tentatively set the exponent of the sum to 100001. Add mantissas:

$$\begin{array}{ll} (A) & 1.11111111000 \\ (B) & 0.01001010101 \\ \hline & 10.01001001101 \end{array}$$

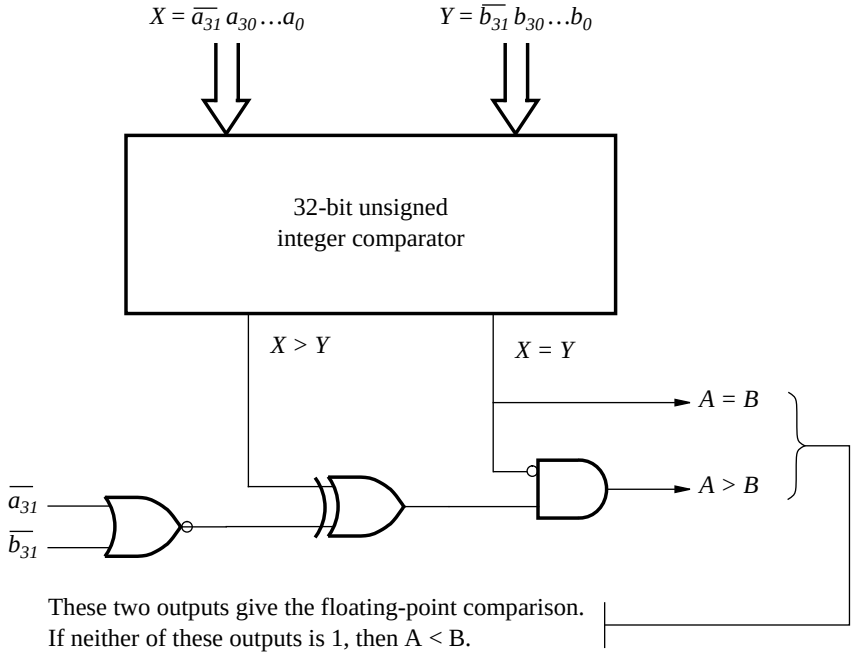
Shift right one position to put in normalized form: 1.001001001101 and increase exponent of sum to 100010. Truncate the mantissa to the right of the binary point to 9 bits by rounding to obtain 001001010. The answer is 0 100010 001001010.

(b)

$$\begin{aligned} \text{Largest} &\approx 2 \times 2^{31} \\ \text{Smallest} &\approx 1 \times 2^{-30} \end{aligned}$$

This assumes that the two end values, 63 and 0 in the excess-31 exponent, are used to represent infinity and exact 0, respectively.

9.23. Let A and B be two floating-point numbers. First, assume that $S_A = S_B = 0$. If $E_A^{'} > E_B^{'}$, considered as unsigned 8-bit numbers, then $A > B$. If $E_A^{'} = E_B^{'}$, then $A > B$ if $M_A > M_B$. This means that $A > B$ if the 31 bits after the sign in the representation for A is greater than the 31 bits representing B, when both are considered as integers. In the logic circuit shown below, all possibilities for the sum bit are also taken into account. In the circuit, let $A = a_{31}a_{30}\ldots a_0$ and $B = b_{31}b_{30}\ldots b_0$ be the two floating-point numbers to be compared.



9.24. Convert the given decimal mantissa into a binary floating-point number by using the integer facilities in the computer to implement the conversion algorithms in Section 9.8. This will yield a floating-point number f_i. Then, using the computer's floating-point facilities, compute $f_i \times t_i$, as required.

9.25. Consider $A - B$, where A and B are 7-bit (normalized) mantissas of floating-point numbers. Because of differences in exponents, B must be shifted 7 positions before subtraction.

$$\begin{aligned} A &= \quad 1.000000 \\ B &= \quad 1.000001 \end{aligned}$$

After shifting, we have:

$$\begin{aligned} A &= \quad 1.000000\ 000 \\ -B &= \quad 0.000000\ 101 \quad \longleftarrow \text{sticky bit} \\ &\quad\ \overline{0.111111\ 011} \\ \text{normalize} &\quad 1.111110\ 110 \\ \text{round} &\quad 1.111111 \quad \longleftarrow \text{correct answer (rounded)} \end{aligned}$$

With only 2 guard bits, we would have had:

$$\begin{aligned} A &= \quad 1.000000\ 00 \\ -B &= \quad 0.000000\ 11 \\ &\quad\ \overline{0.111111\ 01} \\ \text{normalize} &\quad 1.111110\ 10 \\ \text{round} &\quad 1.111110 \end{aligned}$$

9.26. The relevant truth table and logic equations are:

ADD(0) / SUBTRACT(1) (AS)	S_A	S_B	sign from 8-bit subtractor (8_s)	sign from 25-bit adder/ subtractor (25_s)	ADD/ SUB	S_R
0	0	0	0	0	0	0
				1		d
			1	0		0
				1		d
0	0	1	0	0	1	0
				1		1
			1	0		1
				1		d
0	1	0	0	0	1	1
				1		0
			1	0		0
				1		d
0	1	1	0	0	0	1
				1		d
			1	0		1
				1		d
1	0	0	0	0	1	0
				1		1
			1	0		1
				1		d
1	0	1	0	0	0	0
				1		d
			1	0		0
				1		d
1	1	0	0	0	0	1
				1		d
			1	0		1
				1		d
1	1	1	0	0	1	1
				1		0
			1	0		0
				1		d

these variables determine ADD/SUB

ADD(0)/ SUBTRACT(1) (AS)	$S_A S_B$ = 00	01	11	10
0	0	1	0	1
1	1	0	1	0

$\text{ADD/SUB} = AS \oplus S_A \oplus S_B$

$S_B\ 8_s$ \ $AS\ S_A$	00	01	11	10
00	0	1	1	0
01	0	0	1	1
11	1	1	0	0
10	0	1	1	0

$25_s = 0$

$S_B\ 8_s$ \ $AS\ S_A$	00	01	11	10
00	d	0	d	1
01	d	d	d	d
11	d	d	d	d
10	1	d	0	d

$25_s = 1$

$$S_R = 25_s \overline{S_A} + \overline{25_s} S_A \overline{8_s} + AS\ \overline{S_B} 8_s + \overline{AS}\ S_B 8_s$$

9.27. The largest that n can be is 253 for normal values. The mantissas, including the leading bit of 1, are 24 bits long. Therefore, the output of the SHIFTER can be non-zero for $n \leq 23$ only, ignoring guard bit considerations. Let $n = n_7 n_6 \ldots n_0$, and define an enable signal, EN, as EN = $\overline{n}_7 \overline{n}_6 \overline{n}_5$. This variable must be 1 for any output of the SHIFTER to be non-zero. Let $m = m_{23} m_{22} \ldots m_0$ and $s_{23} s_{22} \ldots s_0$ be the SHIFTER inputs and outputs, respectively. The largest network is required for output s_0, because any of the 24 input bits could be shifted into this output position. Define an intermediate binary vector $i = i_{23} i_{22} \ldots i_0$. We will first shift m into i based on EN and $n_4 n_3$. (Then we will shift i into s, based on $n_2 n_1 n_0$.) Only the part of i needed to generate s_0 will be specified.

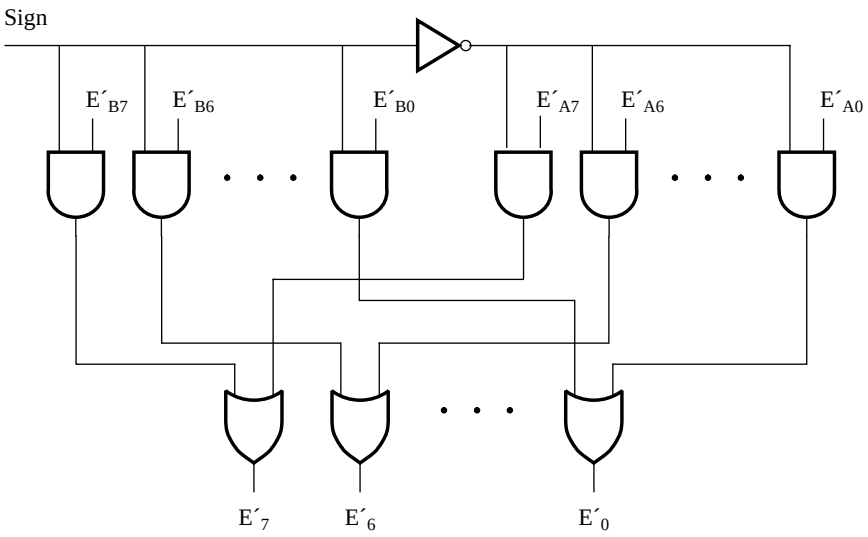
$$
\begin{aligned}
i_7 &= \text{EN} n_4 \overline{n}_3 m_{23} + \text{EN} \overline{n}_4 n_3 m_{15} + \text{EN} \overline{n}_4 \overline{n}_3 m_7 \\
i_6 &= (\ldots) m_{22} + (\ldots) m_{14} + (\ldots) m_6 \\
i_5 &= (\ldots) m_{21} + (\ldots) m_{13} + (\ldots) m_5 \\
&\vdots \\
i_0 &= (\ldots) m_{16} + (\ldots) m_8 + (\ldots) m_0
\end{aligned}
$$

Gates with fan-in up to only 4 are needed to generate these 8 signals. Note that all bits of m are involved, as claimed. We now generate s_0 from these signals and $n_2 n_1 n_0$ as follows:

$$
\begin{aligned}
s_0 = \; & n_2 n_1 n_0 i_7 + n_2 n_1 \overline{n}_0 i_6 + n_2 \overline{n}_1 n_0 i_5 + n_2 \overline{n}_1 \overline{n}_0 i_4 \\
& + \overline{n}_2 n_1 n_0 i_3 + \overline{n}_2 n_1 \overline{n}_0 i_2 + \overline{n}_2 \overline{n}_1 n_0 i_1 + \overline{n}_2 \overline{n}_1 \overline{n}_0 i_0
\end{aligned}
$$

Note that this requires a fan-in of 8 in the OR gate, so that 3 gates will be needed. Other s_i positions can be generated in a similar way.

9.28. The network is:



9.29. The SWAP network is a pair of multiplexers, each one similar to the solution to problem 9.28.

9.30. Let $m = m_{24}m_{23}\ldots m_0$ be the output of the adder/subtractor. The leftmost bit, m_{24}, is the overflow bit that could result from addition. (We ignore the handling of guard bits.) Derive a series of variables, z_i, as follows:

$$\begin{aligned} z_{-1} &= m_{24} \\ z_0 &= \overline{m}_{24}m_{23} \\ z_1 &= \overline{m}_{24}\overline{m}_{23}m_{22} \\ &\vdots \\ z_{23} &= \overline{m}_{24}\overline{m}_{23}\ldots m_0 \\ z_{24} &= \overline{m}_{24}\overline{m}_{23}\ldots \overline{m}_0 \end{aligned}$$

Note that exactly one of the z_i variables is equal to 1 for any particular m vector. Then encode these z_i variables, for $-1 \leq i \leq 23$, into a 6-bit signal representation for X, so that if $z_i = 1$, then $X = i$. The variable z_{24} signifies whether or not the resulting mantissa is zero.

9.31. Augment the 24-bit operand magnitudes entering the adder/subtractor by adding a sign bit position at the left end. Subtraction is then achieved by complementing the bottom operand and performing addition. Group corresponding bit-pairs from the two, signed, 25-bit operands into six groups of four bit-pairs each, plus one bit-pair at the left end, for purposes of deriving P_i and G_i functions. Label these functions P_6, G_6, ..., P_0, G_0, from left-to-right, following the pattern developed in Section 6.2.

The lookahead logic must generate the group input carries c_0, c_4, c_8, ..., c_{24}, accounting properly for the "end-around carry". The key fact is that a carry c_i may have the value 1 because of a generate condition (i.e., some $G_i = 1$) in a higher-order group as well as in a lower-order group. This observation leads to the following logic expressions for the carries:

$$\begin{aligned} c_0 &= G_6 + P_6G_5 + \ldots + P_6P_5P_4P_3P_2P_1G_0 \\ c_4 &= G_0 + P_0G_6 + P_0P_6G_5 + \ldots + P_0P_6P_5P_4P_3P_2G_1 \\ &\vdots \end{aligned}$$

Since the output of this adder is in 1's-complement form, the sign bit determines whether or not to complement the remaining bits in order to send the magnitude M on to the "Normalize and Round" operation. Addition of positive numbers leading to overflow is a valid result, as discussed in Section 6.7.4, and must be distinguished from a negative result that may occur when subtraction is performed. Some logic at the left-end sign position solves this problem.

9.32. (a) In the following answers, rounding has been used as the truncation method (see Section 9.7.2) when the answer cannot be represented exactly in the signed 6-bit format.

0.5:	010000	all cases
-0.123:	100100	Sign-and-magnitude
	111011	1's-complement
	111100	2's-complement
-0.75:	111000	Sign-and-magnitude
	100111	1's-complement
	101000	2's-complement
-0.1:	100011	Sign-and-magnitude
	111100	1's-complement
	111101	2's-complement

(b)

$e = 2^{-6}$ (assuming rounding, as in (a))

$e = 2^{-5}$ (assuming chopping or von Neumann rounding)

(c) Assuming that the given representation error bounds are in decimal notation:

(a)	3
(b)	6
(c)	9

9.33. The binary versions of the decimal fractions -0.123 and -0.1 are not exact. Using 3 guard bits, with the last bit being the sticky bit, the fractions 0.123 and 0.1 are represented as:

$$\begin{aligned} 0.123 &= 0.00011\,111 \\ 0.1 &= 0.00011\,001 \end{aligned}$$

The three representations for both fractions using each of the three truncation methods are:

		Chop	von Neumann	Round
-0.123:	Sign-and-magnitude	1.00011	1.00011	1.00100
	1's-complement	1.11100	1.11100	1.11011
	2's-complement	1.11101	1.11101	1.11100
-0.1:	Sign-and-magnitude	1.00011	1.00011	1.00011
	1's-complement	1.11100	1.11100	1.11100
	2's-complement	1.11101	1.11101	1.11101