Appendix J

Authors: John Hennessy & David Patterson

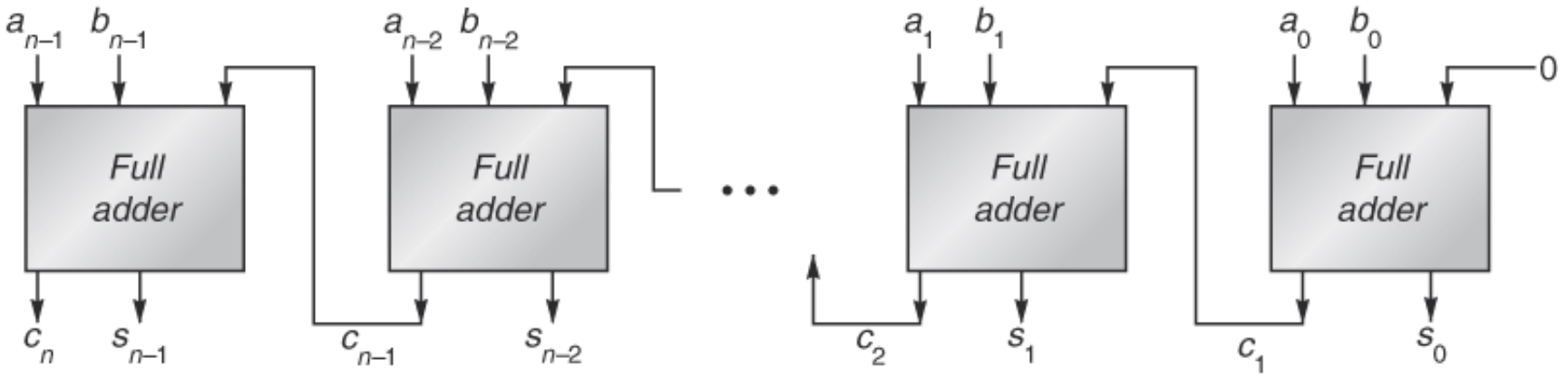


Figure J.1 Ripple-carry adder, consisting of *n* full adders. The carry-out of one full adder is connected to the carry-in of the adder for the next most-significant bit. The carries ripple from the least-significant bit (on the right) to the most-significant bit (on the left).

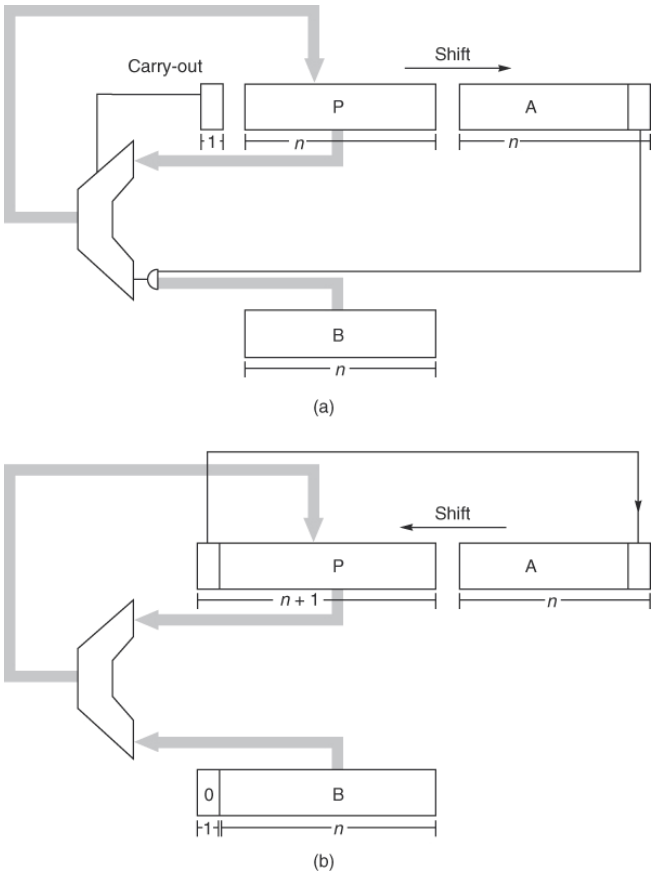


Figure J.2 Block diagram of (a) multiplier and (b) divider for n-bit unsigned integers. Each multiplication step consists of adding the contents of P to either B or 0 (depending on the low-order bit of A), replacing P with the sum, and then shifting both P and A one bit right. Each division step involves first shifting P and A one bit left, subtracting B from P, and, if the difference is nonnegative, putting it into P. If the difference is nonnegative, the low-order bit of A is set to 1.

P	A	
00000	1110	Divide 14 = 1110_2 by 3 = 11_2. B always contains 0011_2.
00001	110	step 1(i): shift.
−00011		step 1(ii): subtract.
−00010	1100	step 1(iii): result is negative, set quotient bit to 0.
00001	1100	step 1(iv): restore.
00011	100	step 2(i): shift.
−00011		step 2(ii): subtract.
00000	1001	step 2(iii): result is nonnegative, set quotient bit to 1.
00001	001	step 3(i): shift.
−00011		step 3(ii): subtract.
−00010	0010	step 3(iii): result is negative, set quotient bit to 0.
00001	0010	step 3(iv): restore.
00010	010	step 4(i): shift.
−00011		step 4(ii): subtract.
−00001	0100	step 4(iii): result is negative, set quotient bit to 0.
00010	0100	step 4(iv): restore. The quotient is 0100_2 and the remainder is 00010_2.

(a)

00000	1110	Divide 14 = 1110_2 by 3 = 11_2. B always contains 0011_2.
00001	110	step 1(i-b): shift.
+11101		step 1(ii-b): subtract b (add two's complement).
11110	1100	step 1(iii): P is negative, so set quotient bit to 0.
11101	100	step 2(i-a): shift.
+00011		step 2(ii-a): add b.
00000	1001	step 2(iii): P is nonnegative, so set quotient bit to 1.
00001	001	step 3(i-b): shift.
+11101		step 3(ii-b): subtract b.
11110	0010	step 3(iii): P is negative, so set quotient bit to 0.
11100	010	step 4(i-a): shift.
+00011		step 4(ii-a): add b.
11111	0100	step 4(iii): P is negative, so set quotient bit to 0.
+00011		Remainder is negative, so do final restore step.
00010		The quotient is 0100_2 and the remainder is 00010_2.

(b)

Figure J.3 Numerical example of (a) restoring division and (b) nonrestoring division.

P	A	
0000	1010	Put $-6 = 1010_2$ into A, $-5 = 1011_2$ into B.
0000	1010	step 1(i): $a_0 = a_{-1} = 0$, so from rule I add 0.
0000	0101	step 1(ii): shift.
+ 0101		step 2(i): $a_1 = 1$, $a_0 = 0$. Rule III says subtract b (or add $-b = -1011_2 = 0101_2$).
0101	0101	
0010	1010	step 2(ii): shift.
+ 1011		step 3(i): $a_2 = 0$, $a_1 = 1$. Rule II says add b (1011).
1101	1010	
1110	1101	step 3(ii): shift. (Arithmetic shift—load 1 into leftmost bit.)
+ 0101		step 4(i): $a_3 = 1$, $a_2 = 0$. Rule III says subtract b.
0011	1101	
0001	1110	step 4(ii): shift. Final result is $00011110_2 = 30$.

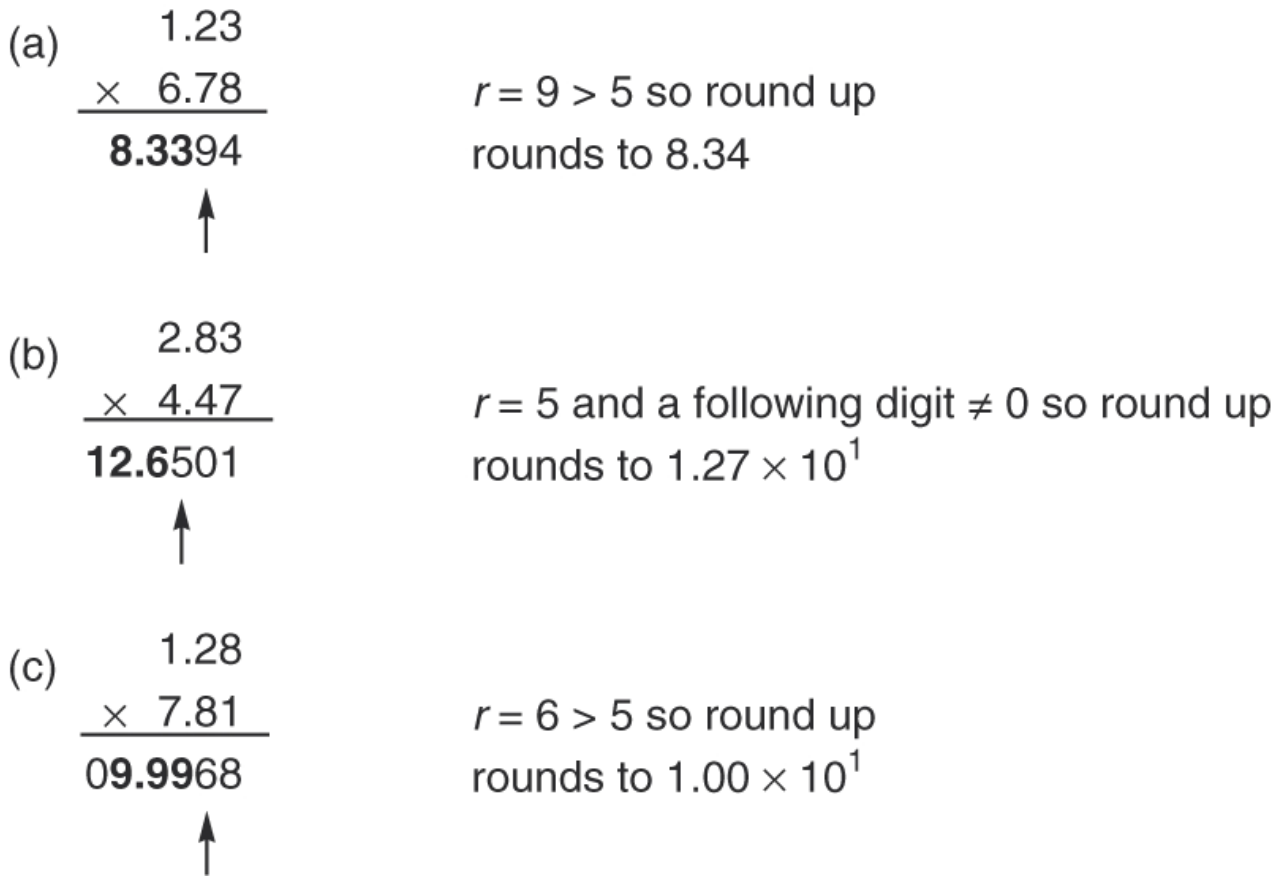


Figure J.9 Examples of rounding a multiplication. Using base 10 and $p = 3$, parts (a) and (b) illustrate that the result of a multiplication can have either $2p - 1$ or $2p$ digits; hence, the position where a 1 is added when rounding up (just left of the arrow) can vary. Part (c) shows that rounding up can cause a carry-out.

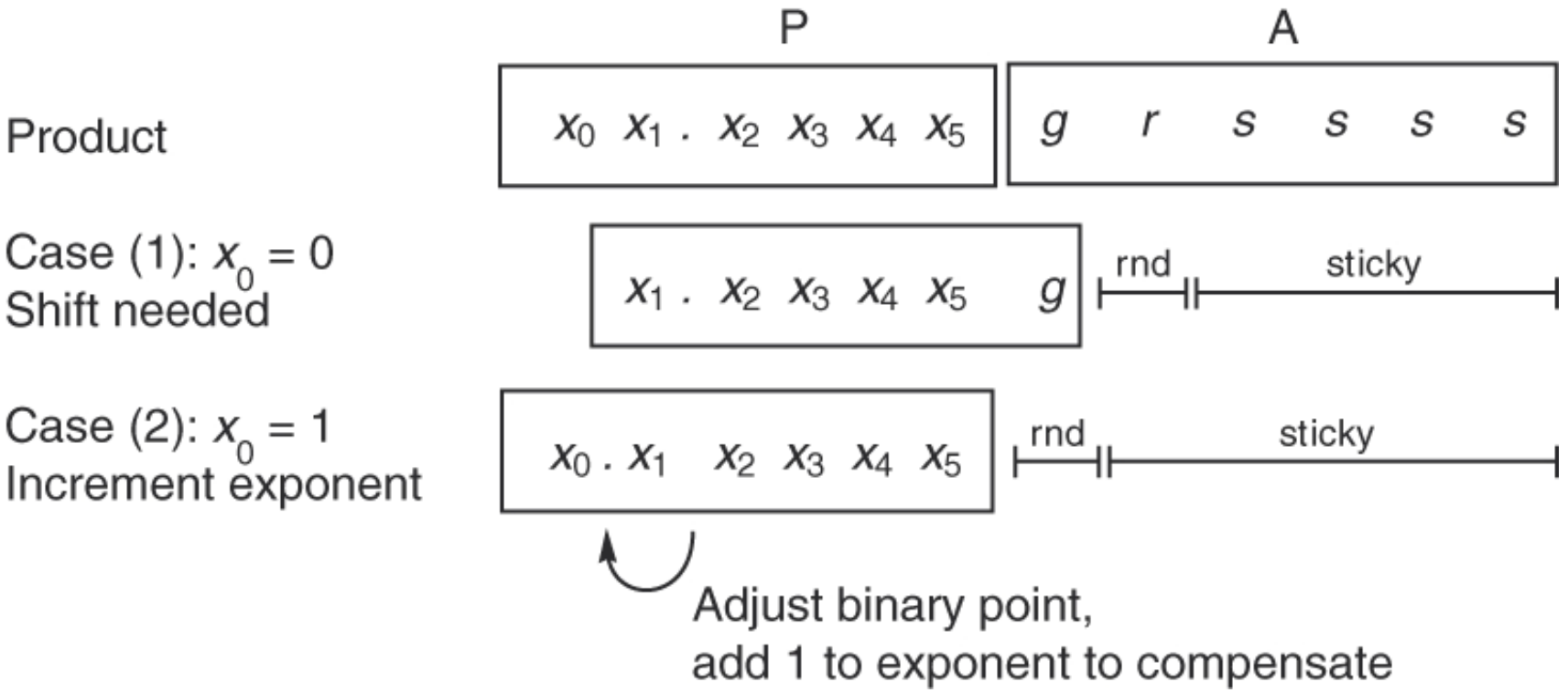


Figure J.10 The two cases of the floating-point multiply algorithm. The top line shows the contents of the P and A registers after multiplying the significands, with $p = 6$. In case (1), the leading bit is 0, and so the P register must be shifted. In case (2), the leading bit is 1, no shift is required, but both the exponent and the round and sticky bits must be adjusted. The sticky bit is the logical or of the bits marked *s*.

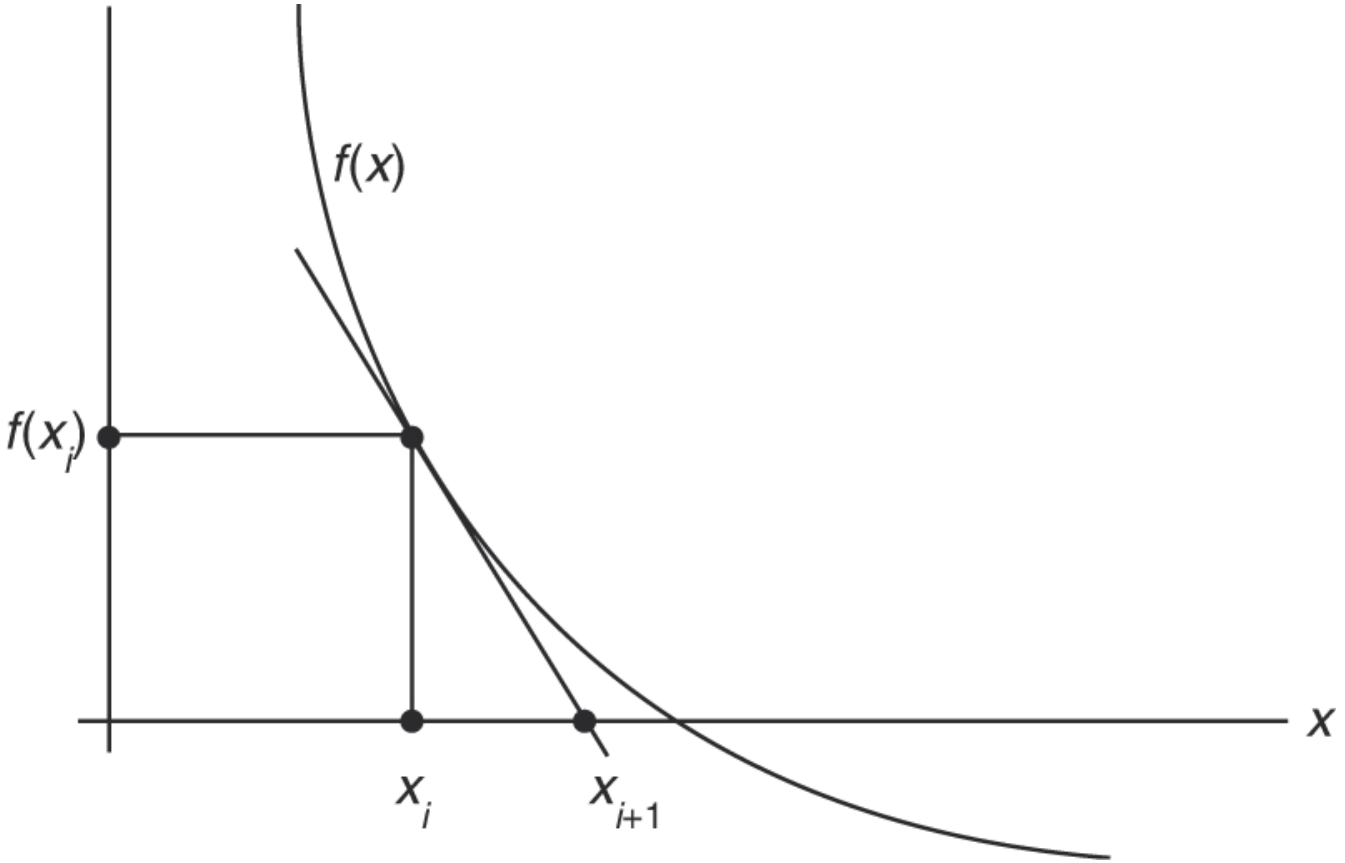


Figure J.13 Newton's iteration for zero finding. If x_i is an estimate for a zero of f, then x_i+1 is a better estimate. To compute x_i+1, find the intersection of the x-axis with the tangent line to f at $f(x_i)$.

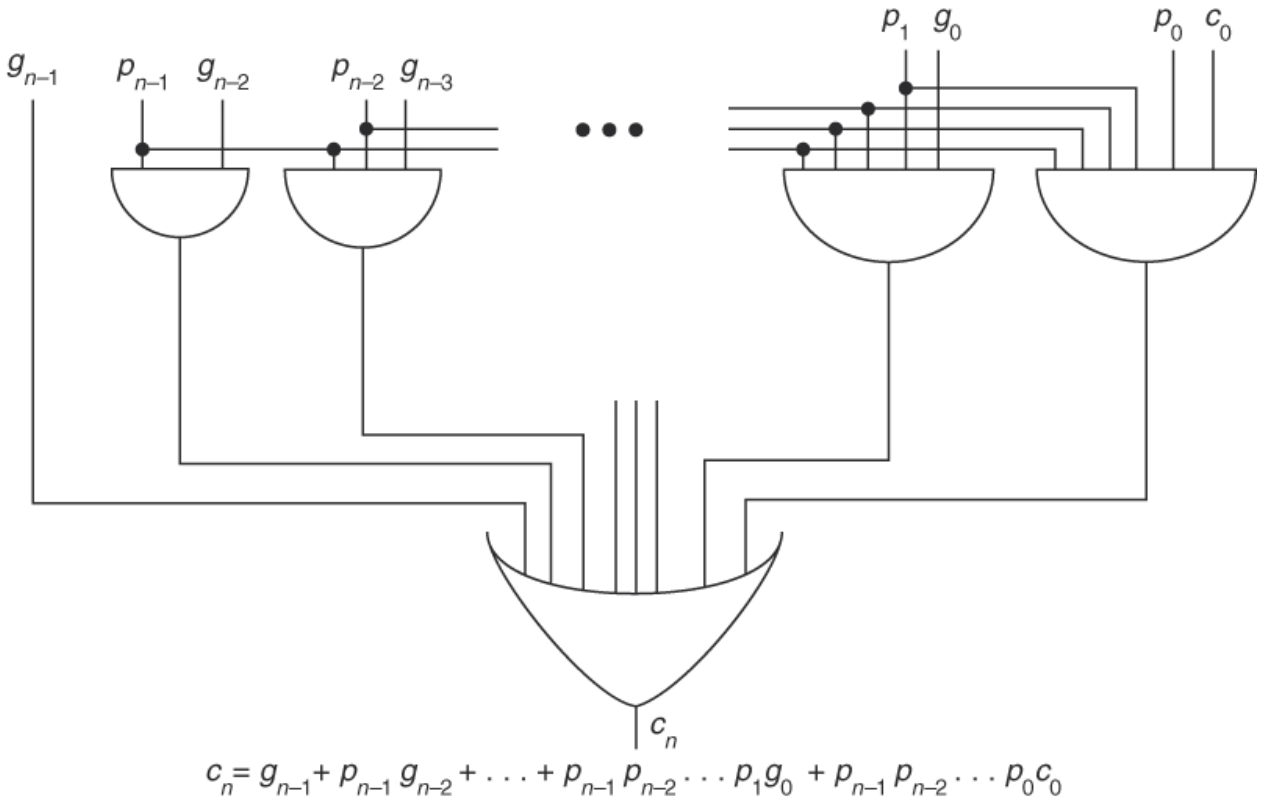


Figure J.14 Pure carry-lookahead circuit for computing the carry-out *c*n of an *n*-bit adder.

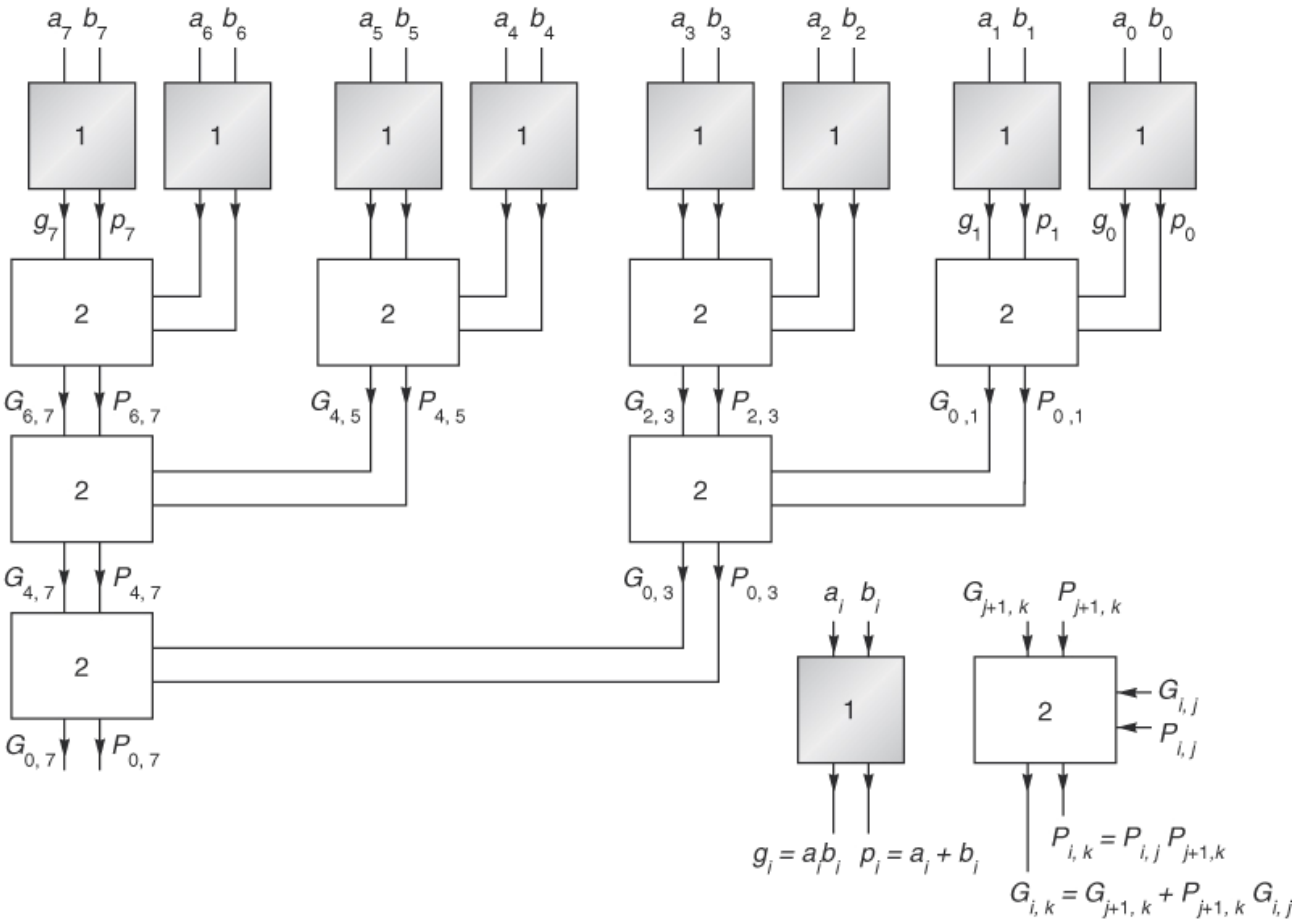


Figure J.15 First part of carry-lookahead tree. As signals flow from the top to the bottom, various values of *P* and *G* are computed.

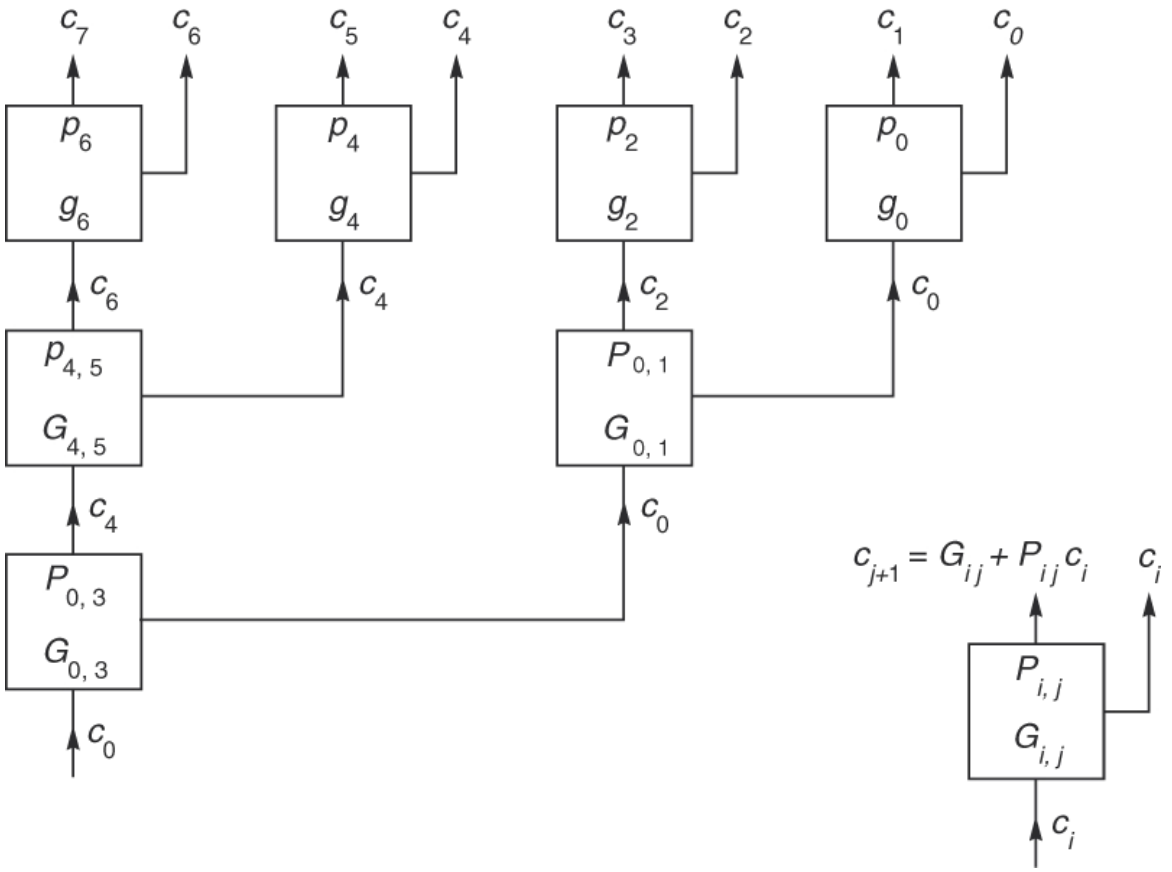


Figure J.16 Second part of carry-lookahead tree. Signals flow from the bottom to the top, combining with *P* and *G* to form the carries.

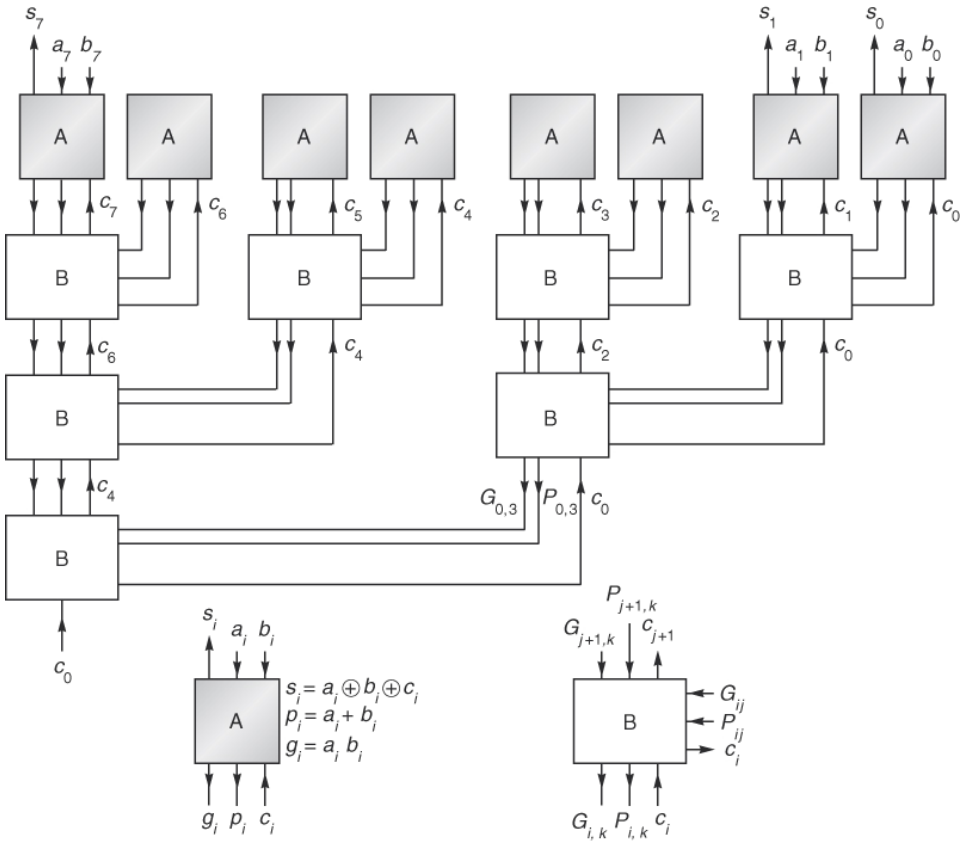


Figure J.17 Complete carry-lookahead tree adder. This is the combination of Figures J.15 and J.16. The numbers to be added enter at the top, flow to the bottom to combine with *c*0, and then flow back up to compute the sum bits.

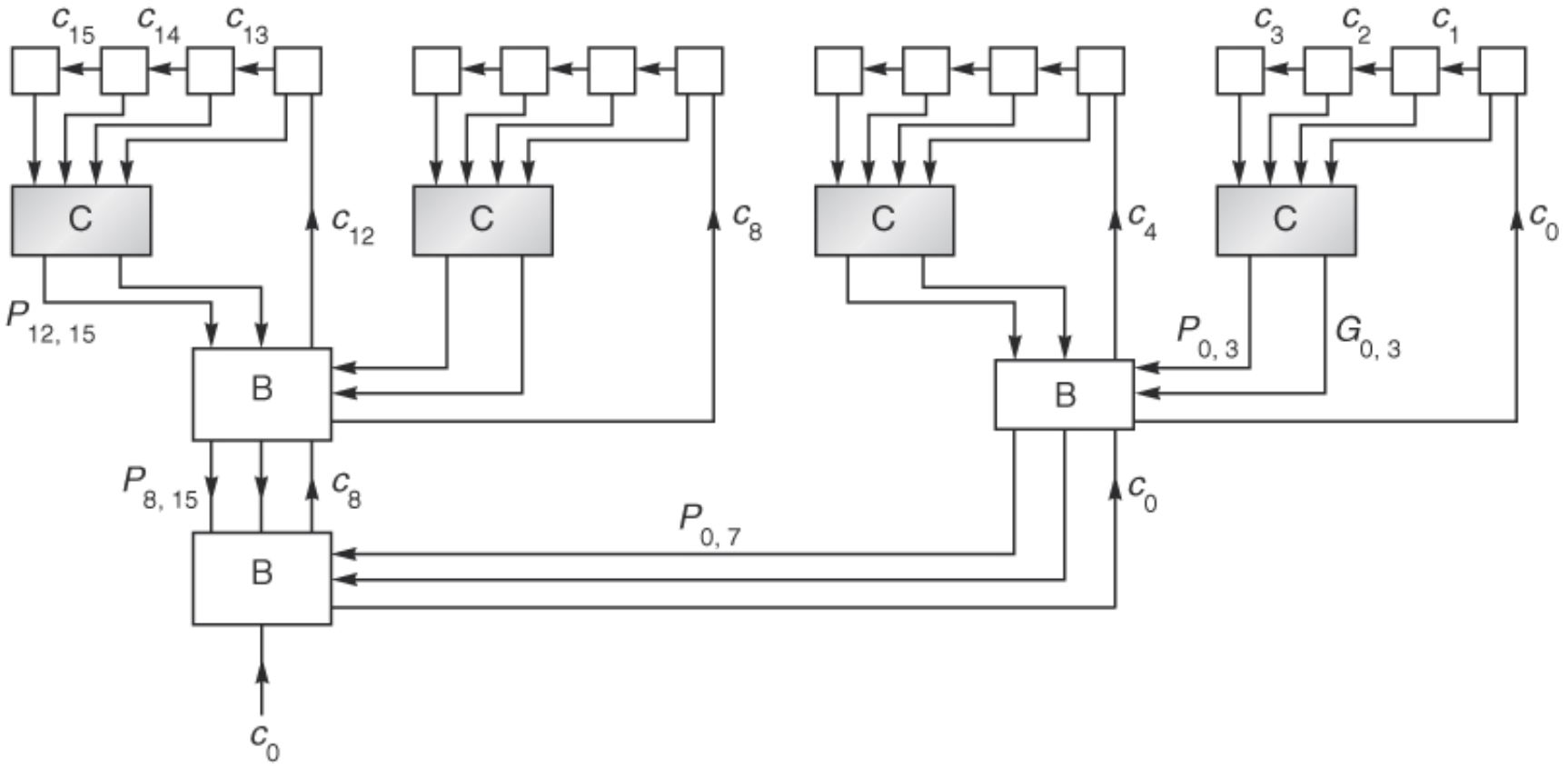


Figure J.18 Carry-skip adder. This is a 20-bit carry-skip adder (n = 20) with each block 4-bits wide (k = 4).

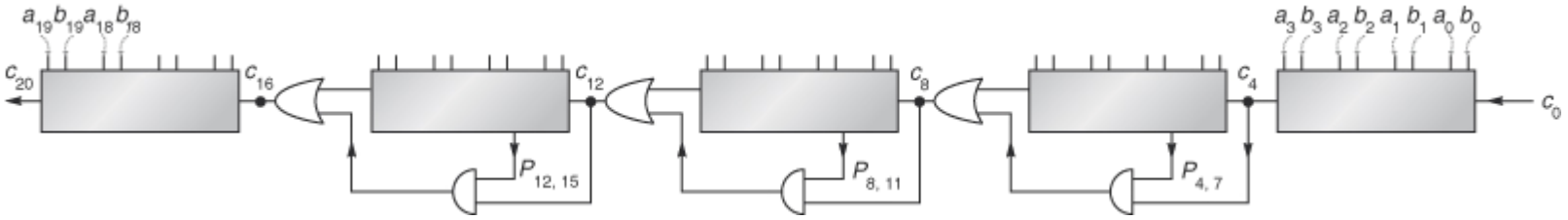


Figure J.19 Combination of CLA and ripple-carry adder. In the top row, carries ripple within each group of four boxes.

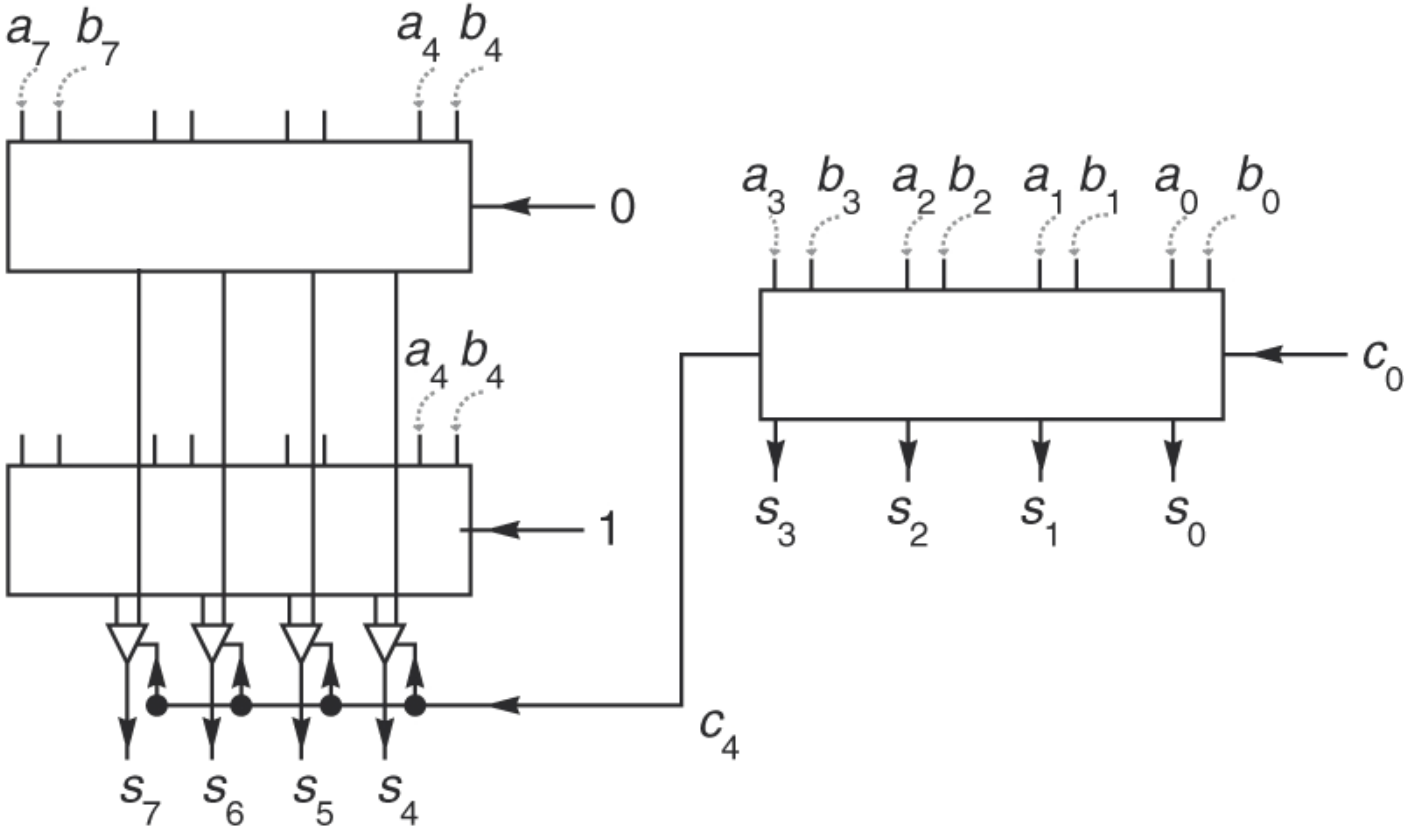


Figure J.20 Simple carry-select adder. At the same time that the sum of the low-order 4 bits is being computed, the high-order bits are being computed twice in parallel: once assuming that $c_4 = 0$ and once assuming $c_4 = 1$.

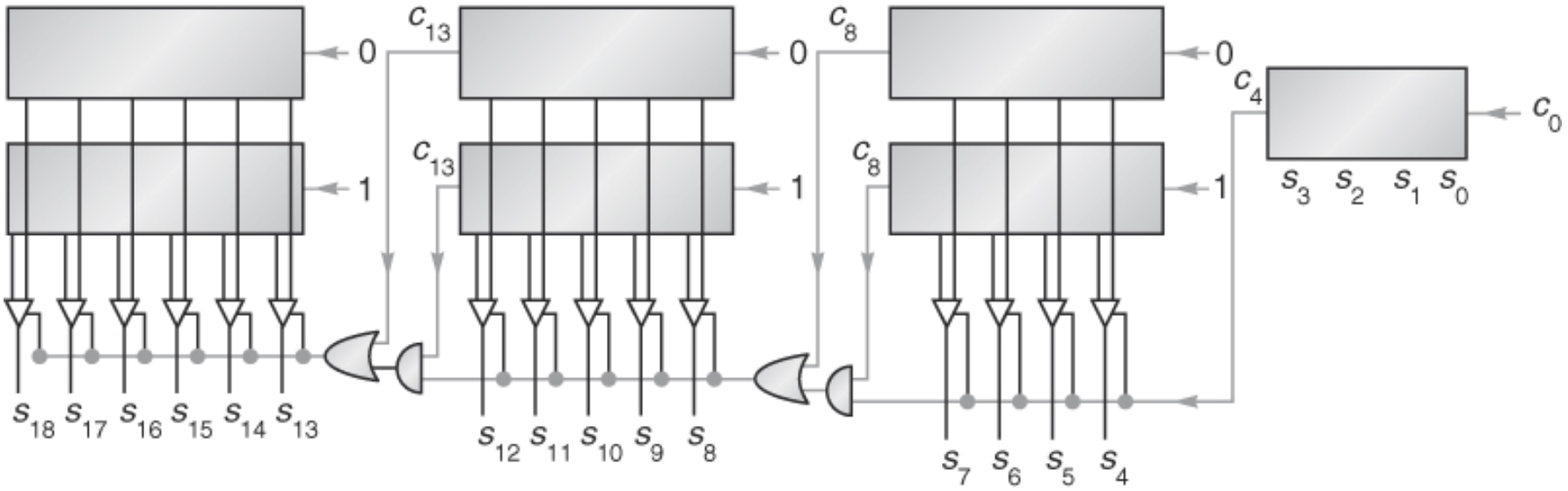


Figure J.21 Carry-select adder. As soon as the carry-out of the rightmost block is known, it is used to select the other sum bits.

P	A	
00000	1000	Divide 8 = 1000 by 3 = 0011. B contains 0011.
00010	0000	Step 1: B had two leading 0s, so shift left by 2. B now contains 1100.
		Step 2.1: Top three bits are equal. This is case (a), so
00100	0000	set $q_0 = 0$ and shift.
		Step 2.2: Top three bits not equal and P ≥ 0 is case (c), so
01000	0001	set $q_1 = 1$ and shift.
+ 10100		Subtract B.
11100	0001	Step 2.3: Top bits equal is case (a), so
11000	0010	set $q_2 = 0$ and shift.
		Step 2.4: Top three bits unequal is case (b), so
10000	0101	set $q_3 = -1$ and shift.
+ 01100		Add B.
11100		Step 3. Remainder is negative so restore it and subtract 1 from *q*.
+ 01100		
01000		Must undo the shift in step 1, so right-shift by 2 to get true remainder.
		Remainder = 10, quotient = $010\bar{1} - 1 = 0010$.

Figure J.23 SRT division of $1000_2/0011_2$. The quotient bits are shown in bold, using the notation $\bar{1}$ for –1.

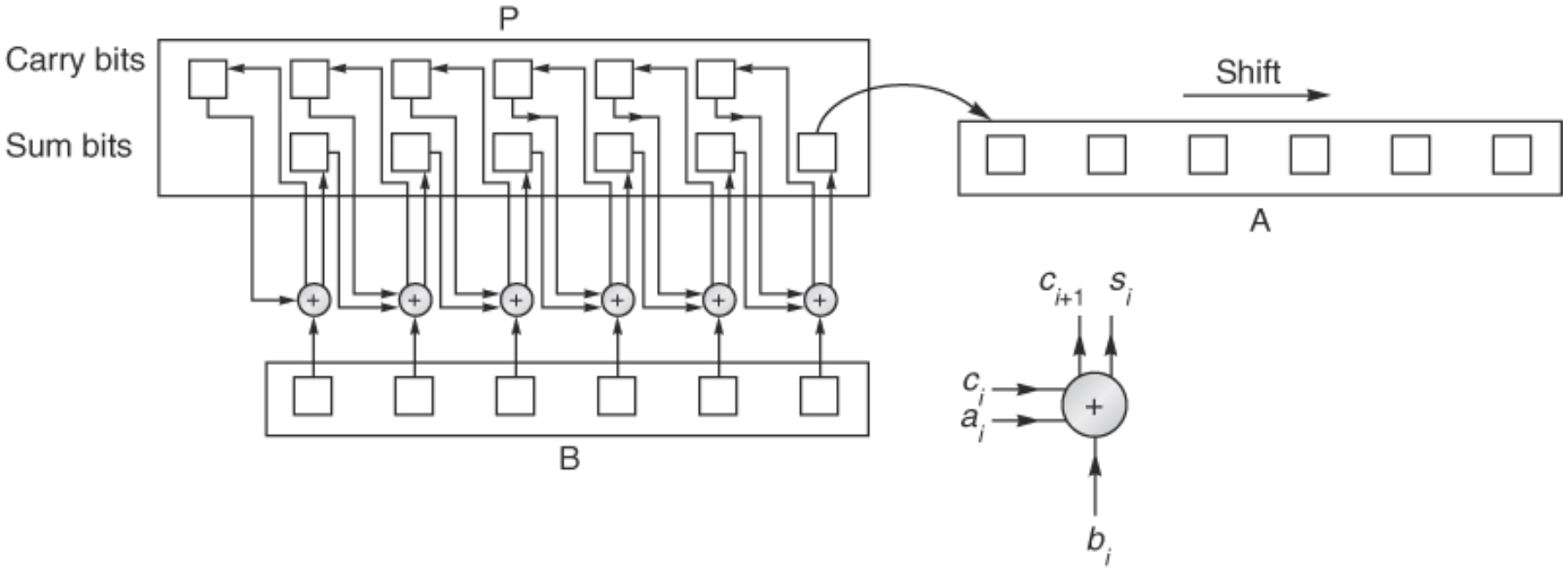


Figure J.24 Carry-save multiplier. Each circle represents a (3,2) adder working independently. At each step, the only bit of P that needs to be shifted is the low-order sum bit.

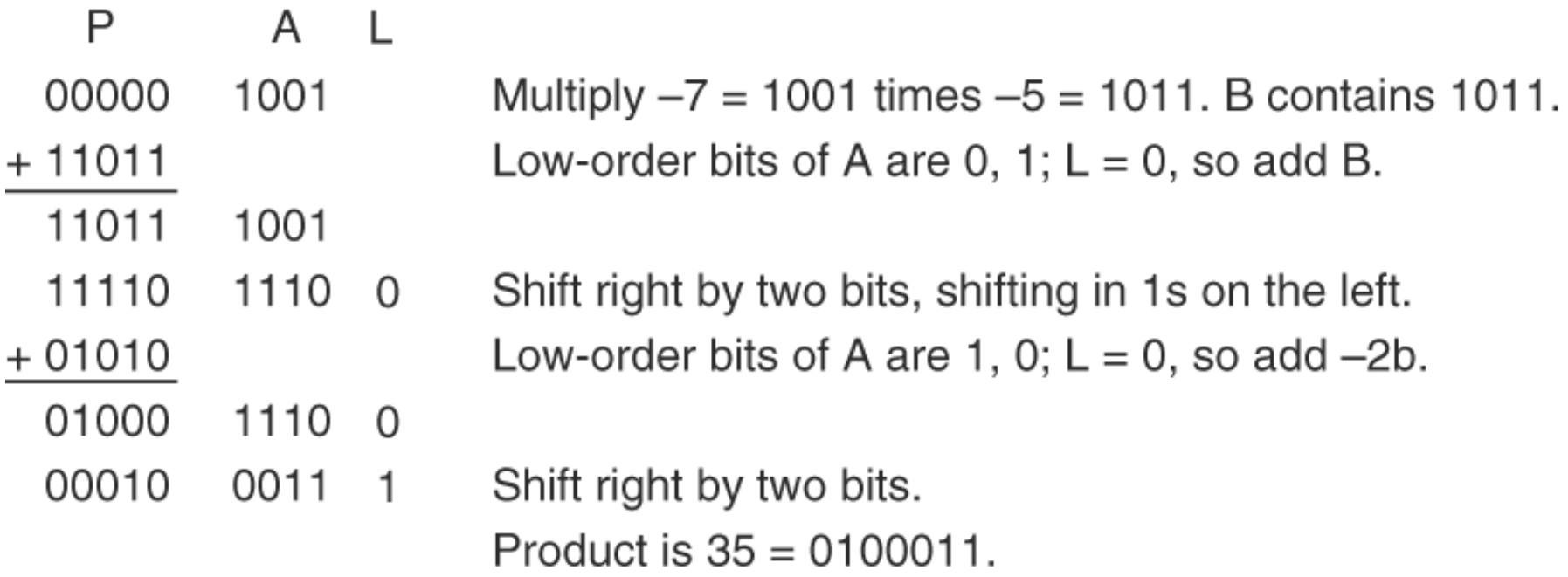


Figure J.26 Multiplication of –7 times –5 using radix-4 Booth recoding. The column labeled L contains the last bit shifted out the right end of A.

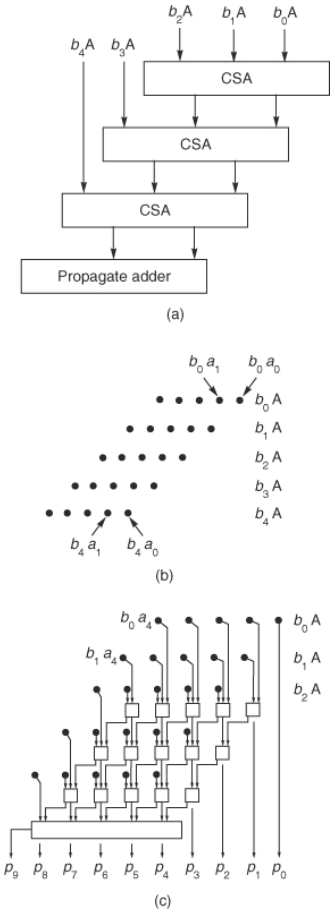


Figure J.27 An array multiplier. The 5-bit number in A is multiplied by $b_4b_3b_2b_1b_0$. Part (a) shows the block diagram, (b) shows the inputs to the array, and (c) expands the array to show all the adders.

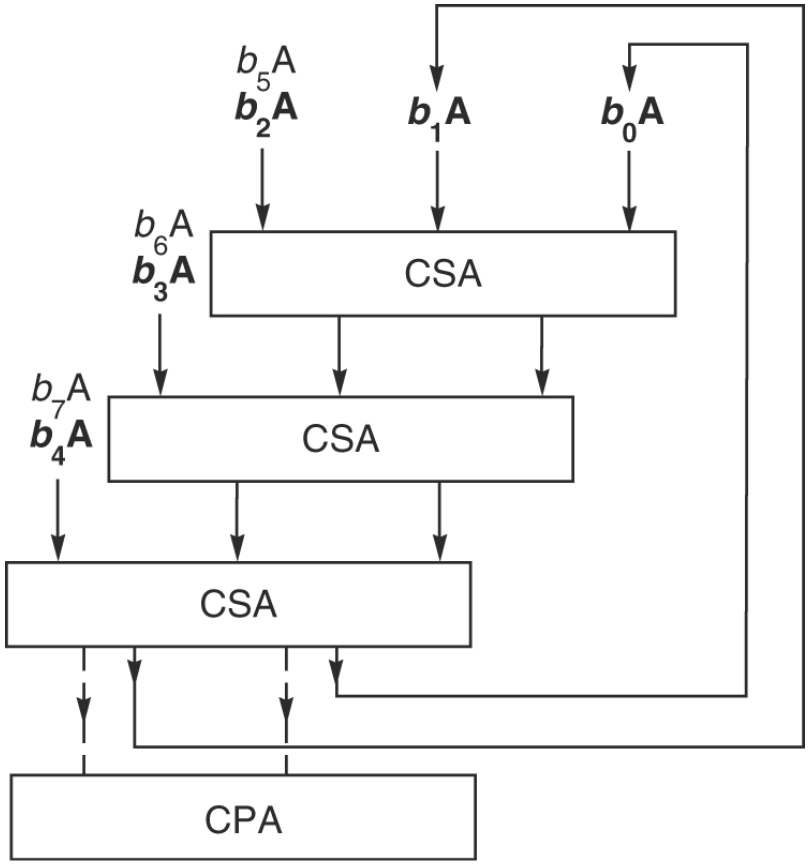


Figure J.28 Multipass array multiplier. Multiplies two 8-bit numbers with about half the hardware that would be used in a one-pass design like that of Figure J.27. At the end of the second pass, the bits flow into the CPA. The inputs used in the first pass are marked in bold.

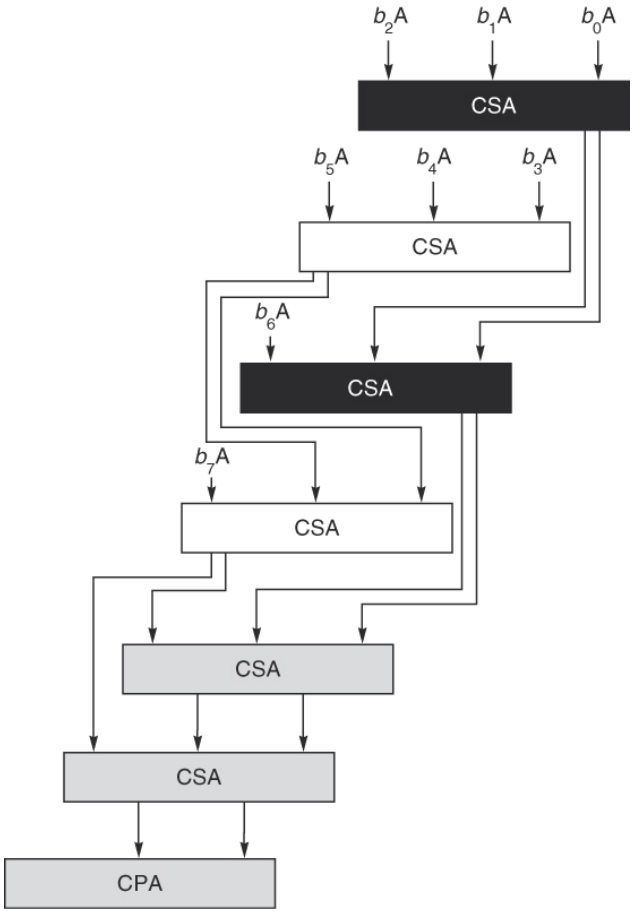


Figure J.29 Even/odd array. The first two adders work in parallel. Their results are fed into the third and fourth adders, which also work in parallel, and so on.

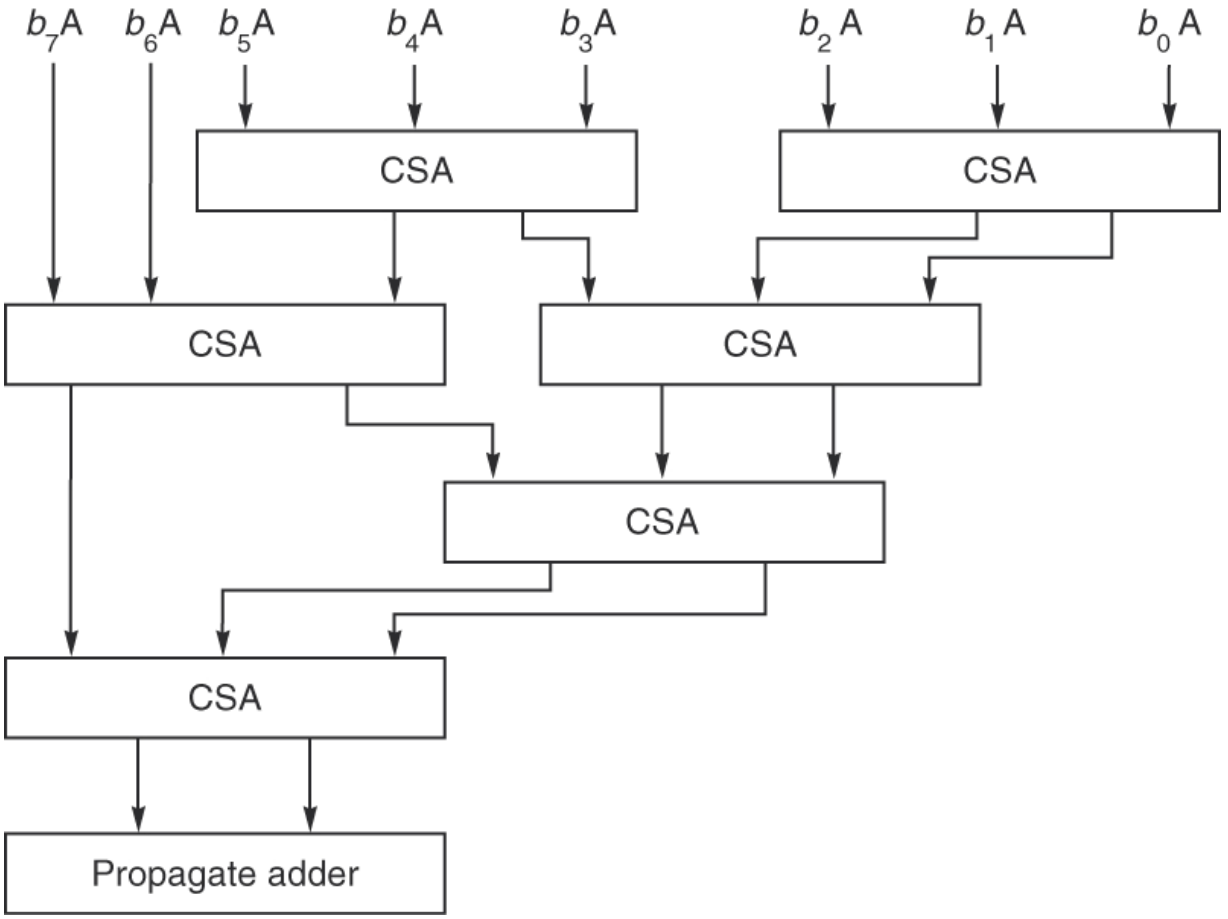


Figure J.30 Wallace tree multiplier. An example of a multiply tree that computes a product in 0(log *n*) steps.

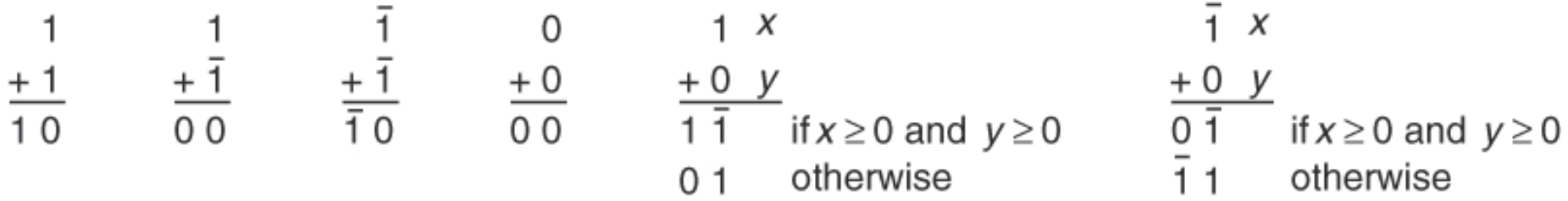


Figure J.31 Signed-digit addition table. The leftmost sum shows that when computing 1 + 1, the sum bit is 0 and the carry bit is 1.

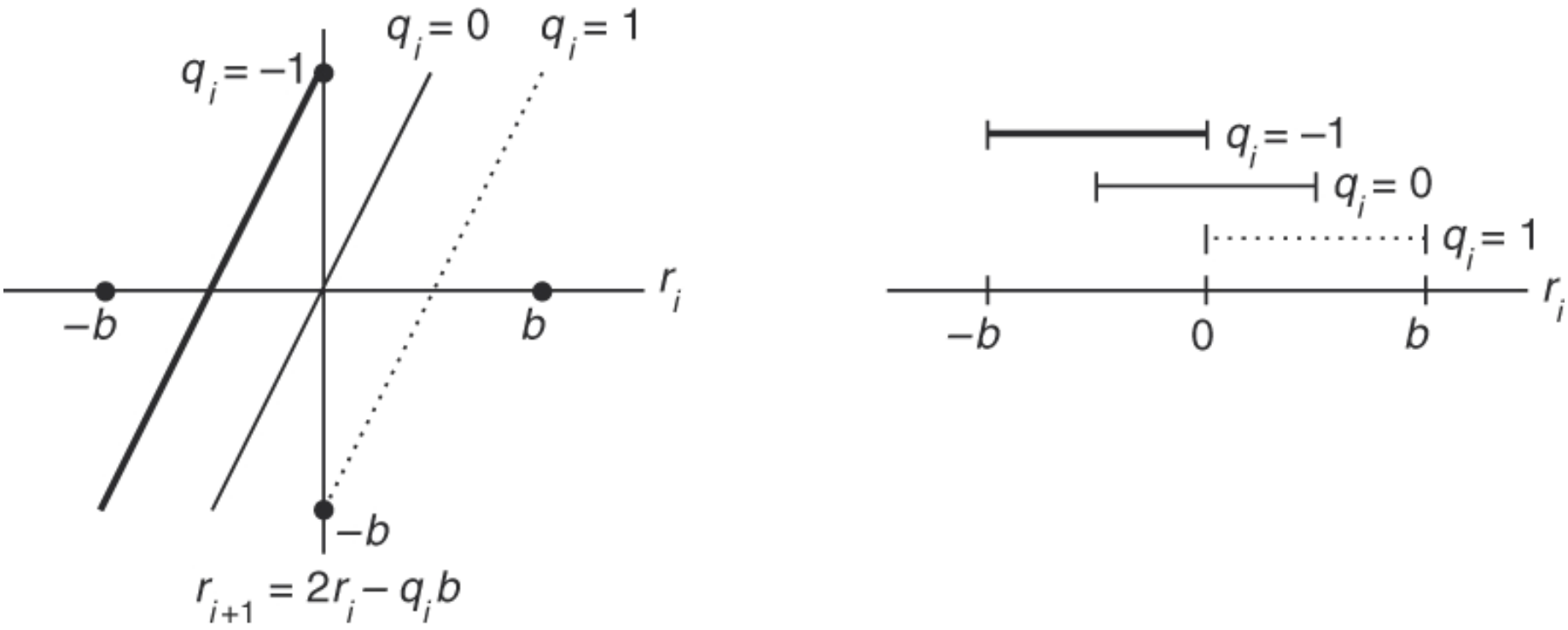


Figure J.32 Quotient selection for radix-2 division. The *x*-axis represents the *I*th remainder, which is the quantity in the (P,A) register pair. The *y*-axis shows the value of the remainder after one additional divide step. Each bar on the right-hand graph gives the range of r_i values for which it is permissible to select the associated value of q_i.

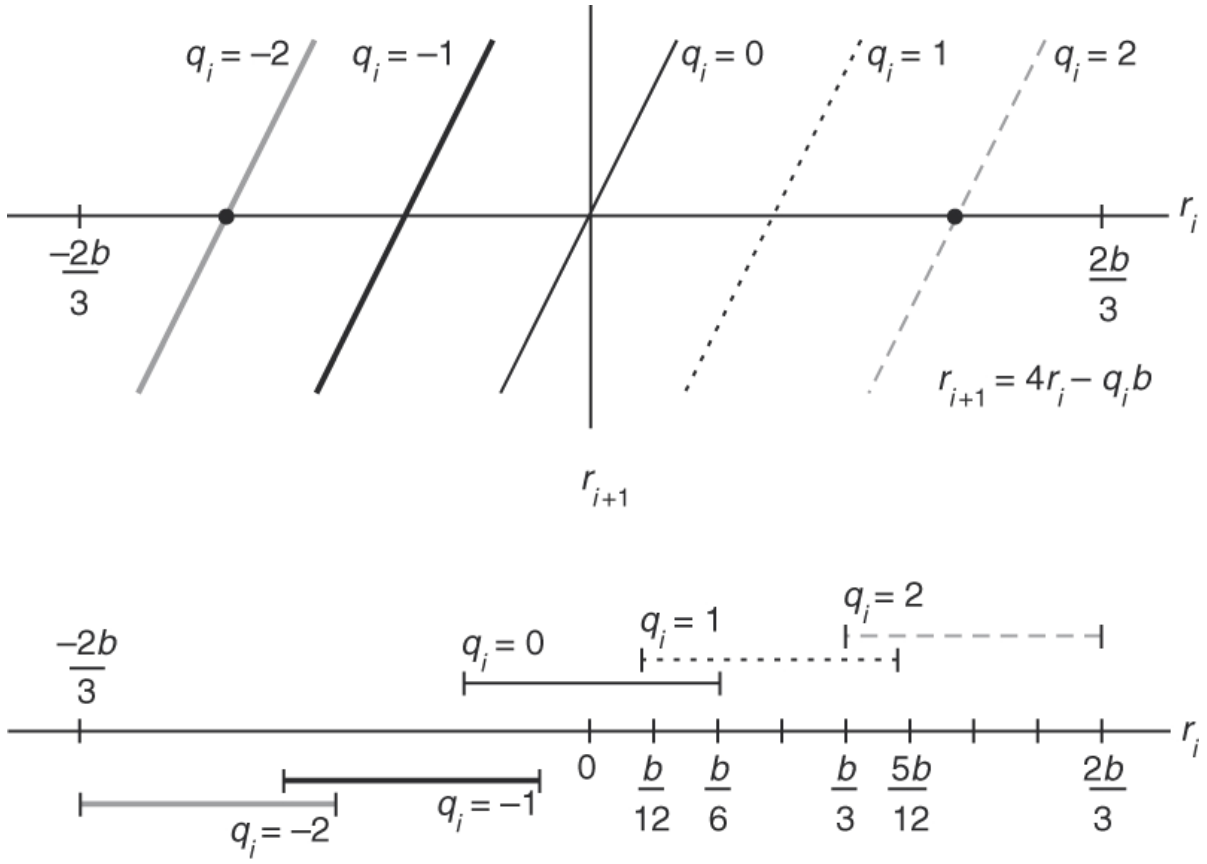


Figure J.33 Quotient selection for radix-4 division with quotient digits –2, –1, 0, 1, 2.

P	A		
000000000	10010101	Divide 149 by 5. B contains 00000101.	
000010010	10100000	Step 1:	B had 5 leading 0s, so shift left by 5. B now contains 10100000, so use $b = 10$ section of table.
		Step 2.1:	Top 6 bits of P are 2, so shift left by 2. From table, can pick q to be
001001010	1000000		0 or 1. Choose $q_0 = 0$.
		Step 2.2:	Top 6 bits of P are 9, so
100101010	000002		shift left 2. $q_1 = 2$.
+ 011000000			Subtract $2b$.
111101010	000002	Step 2.3:	Top bits = –3, so
110101000	00020		shift left 2. Can pick 0 or –1 for q, pick $q_2 = 0$.
		Step 2.4:	Top bits = –11, so
010100000	0202		shift left 2. $q_3 = -2$.
+ 101000000			Add $2b$.
111100000		Step 3:	Remainder is negative, so restore
+ 010100000			by adding b and subtract 1 from q.
010000000		Answer:	$q = 020\bar{2} - 1 = 29$.
			To get remainder, undo shift in step 1 so remainder = 010000000 >> 5 = 4.

Figure J.35 Example of radix-4 SRT division. Division of 149 by 5.

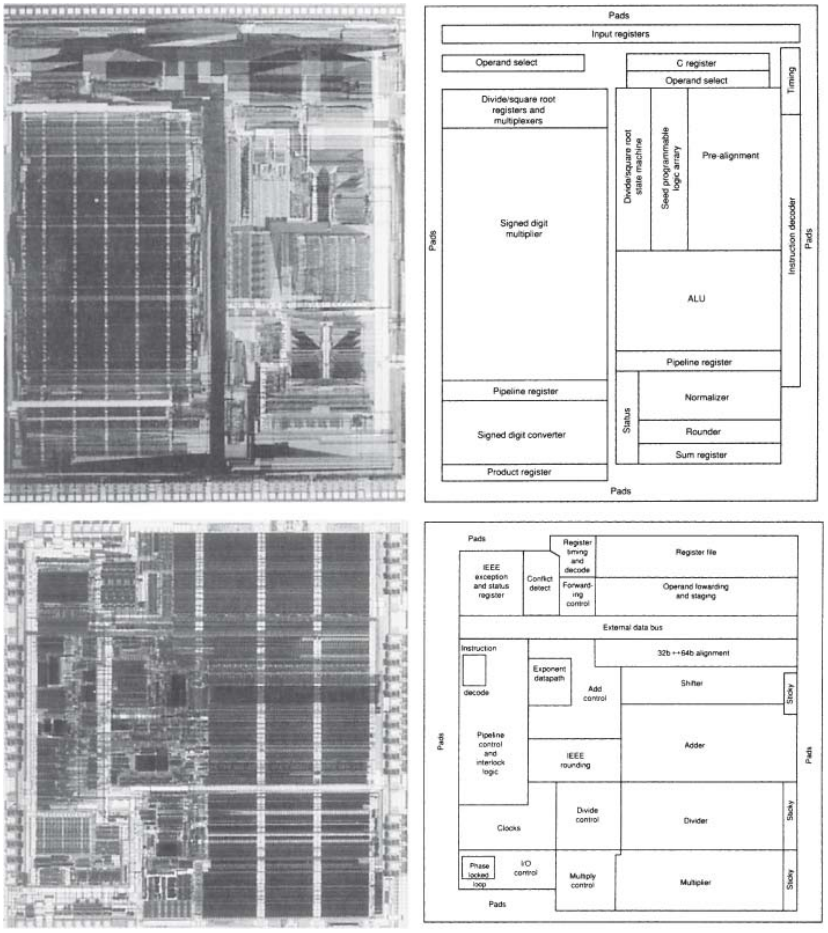
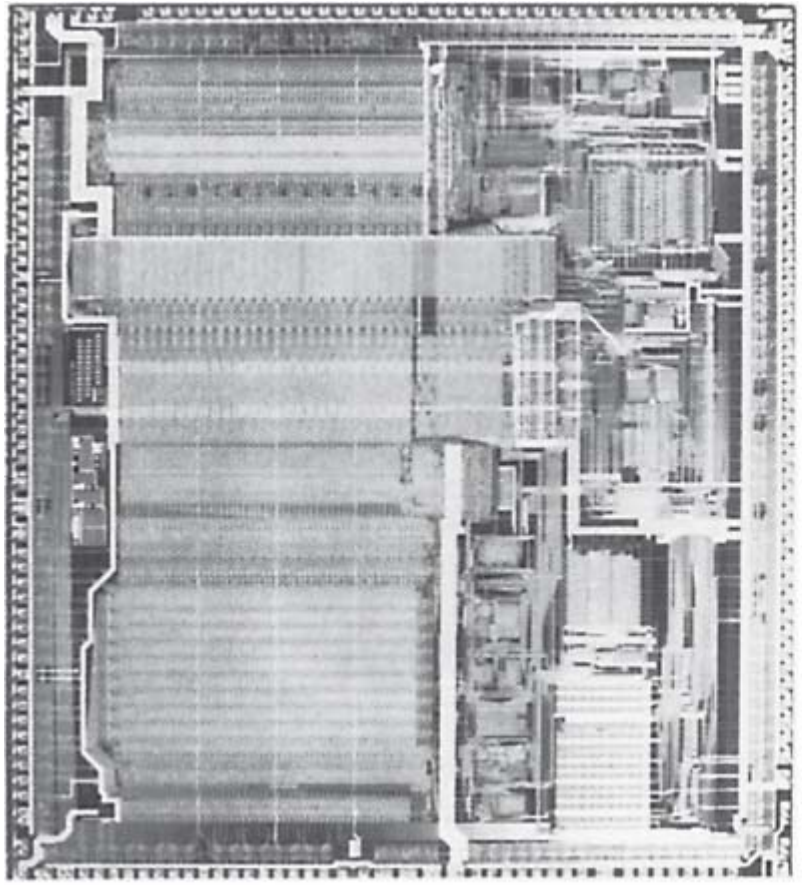


Figure J.37A Chip layout for the TI 8847, MIPS R3010, and Weitek 3364. In the left-hand columns are the photomicrographs; the right-hand columns show the corresponding floor plans.

B



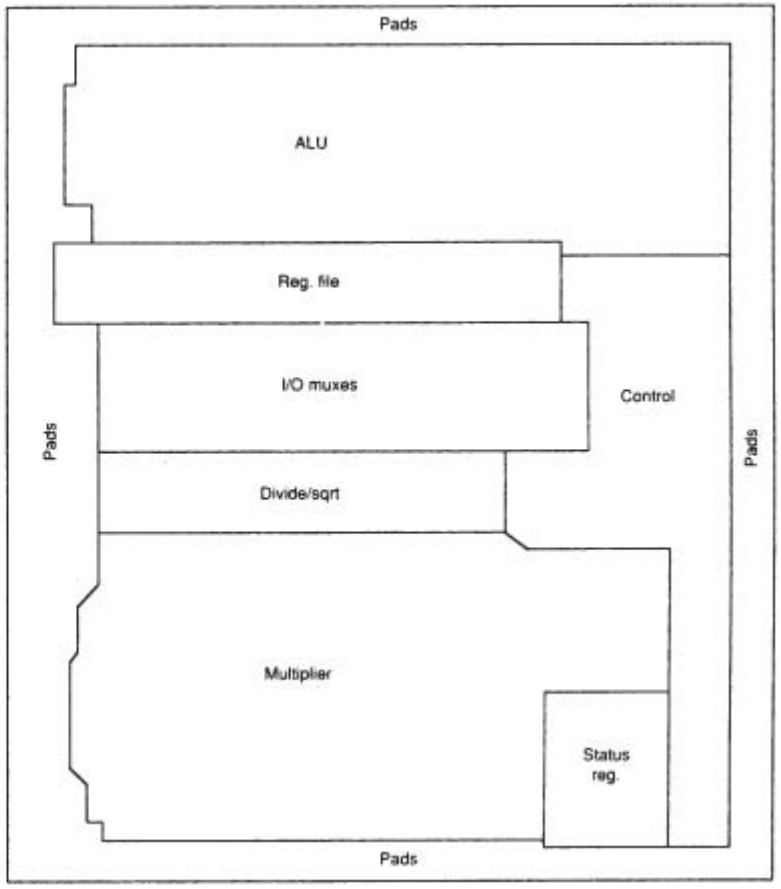


Figure J.37B Chip layout for the TI 8847, MIPS R3010, and Weitek 3364. In the left-hand columns are the photomicrographs; the right-hand columns show the corresponding floor plans.