Ecuaciones de un sumador completo (*full adder*)

$s_i = a_i b_i' c_i' + a_i' b_i c_i' + a_i' b_i' c_i + a_i b_i c_i$

$c_{i+1} = a_i b_i + a_i c_i + b_i c_i$

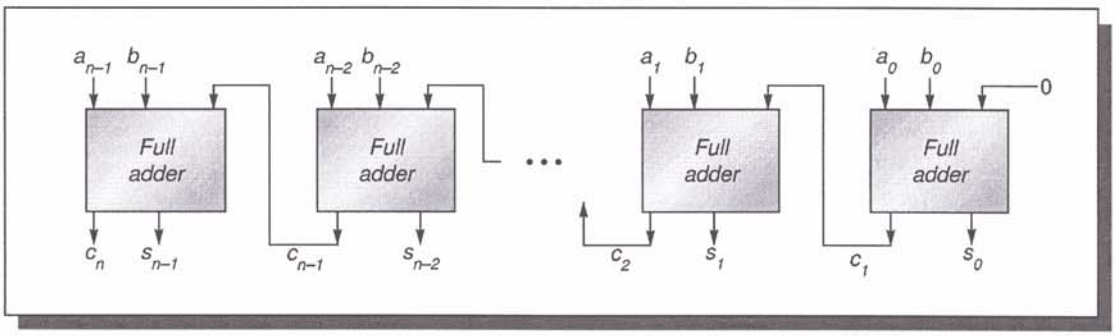

FIGURE A.1 Ripple-carry adder, consisting of *n* full adders.

$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i$$

$$\mathbf{c_{i+1} = g_i + p_i c_i}$$

$$g_i = a_i b_i$$

$$p_i = a_i + b_i$$

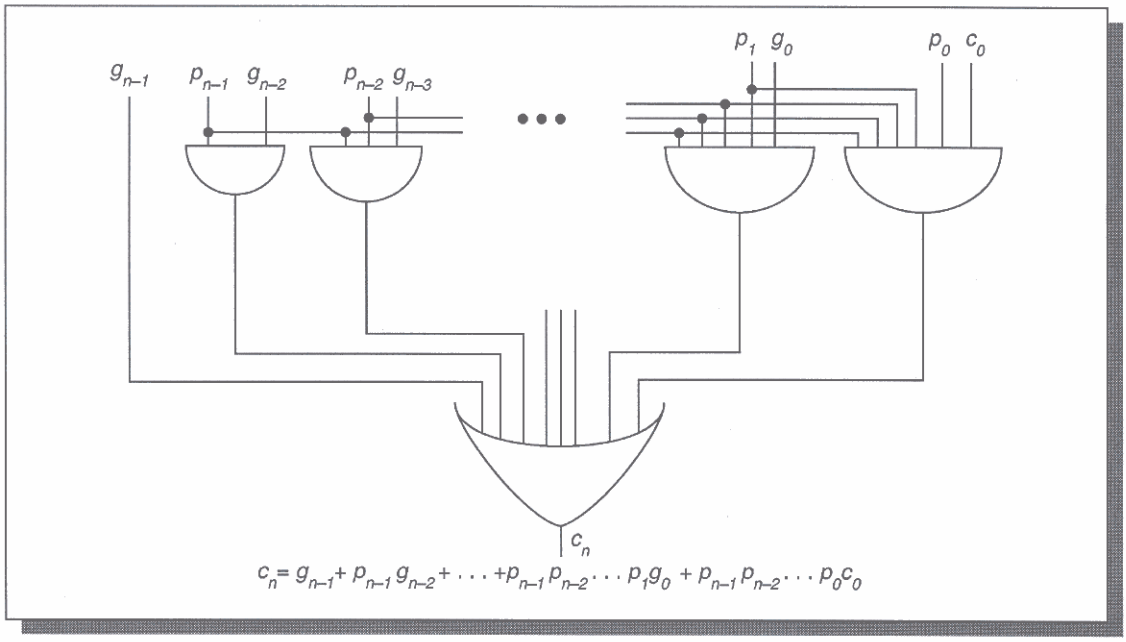

FIGURE A.14 Pure carry-lookahead circuit for computing the carry-out c_n of an *n*-bit adder.

$$c_{j+1} = G_{ij} + P_{ij}C_i$$

$$G_{ik} = G_{j+1,k} + P_{j+1,k}G_{ij}$$

$$P_{ik} = P_{ij}P_{j+1,k}$$

$$P_{ii} = p_i;\ G_{ii} = g_i$$

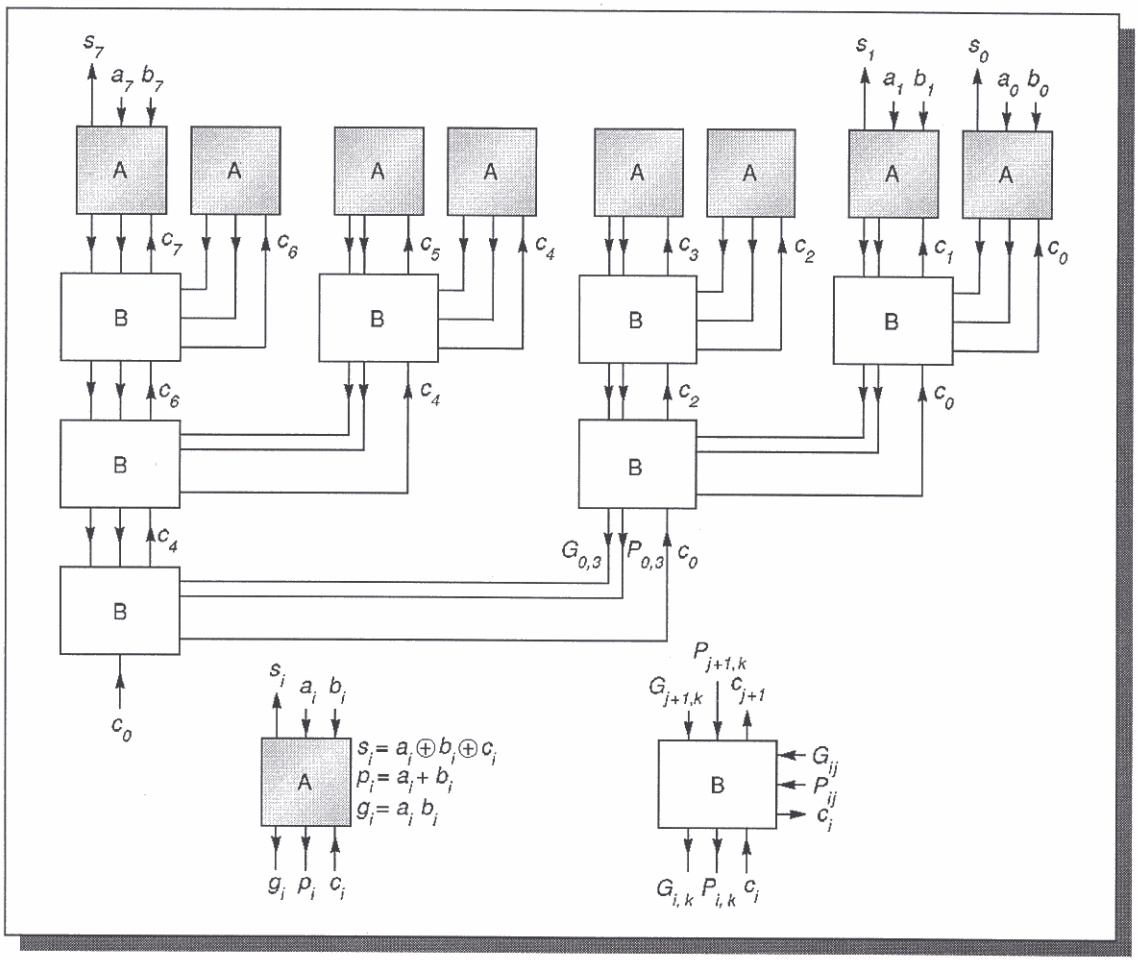

FIGURE A.17 Complete carry-lookahead tree adder.

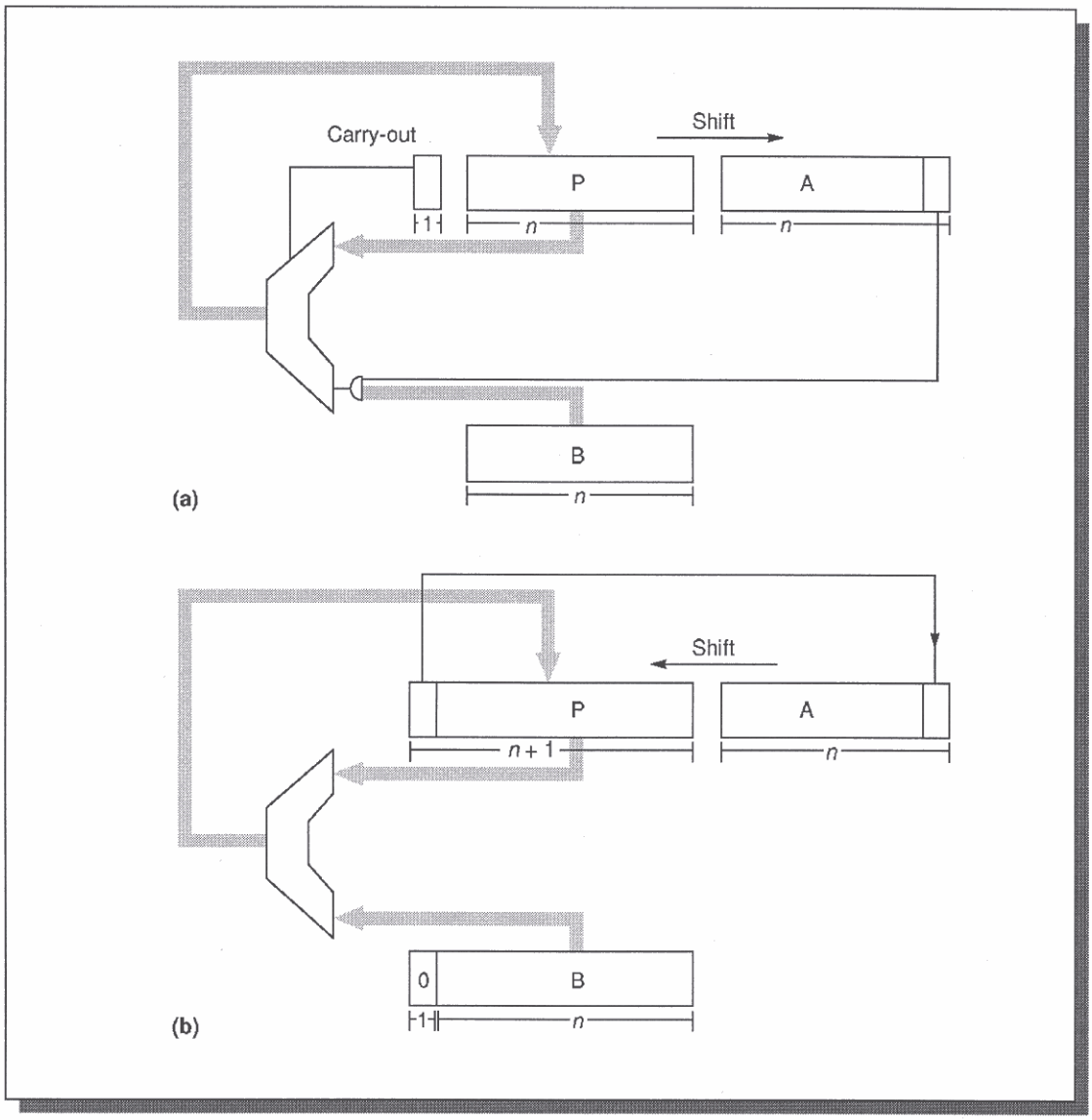

FIGURE A.2 Block diagram of (a) multiplier and (b) divider for *n*-bit unsigned integers.

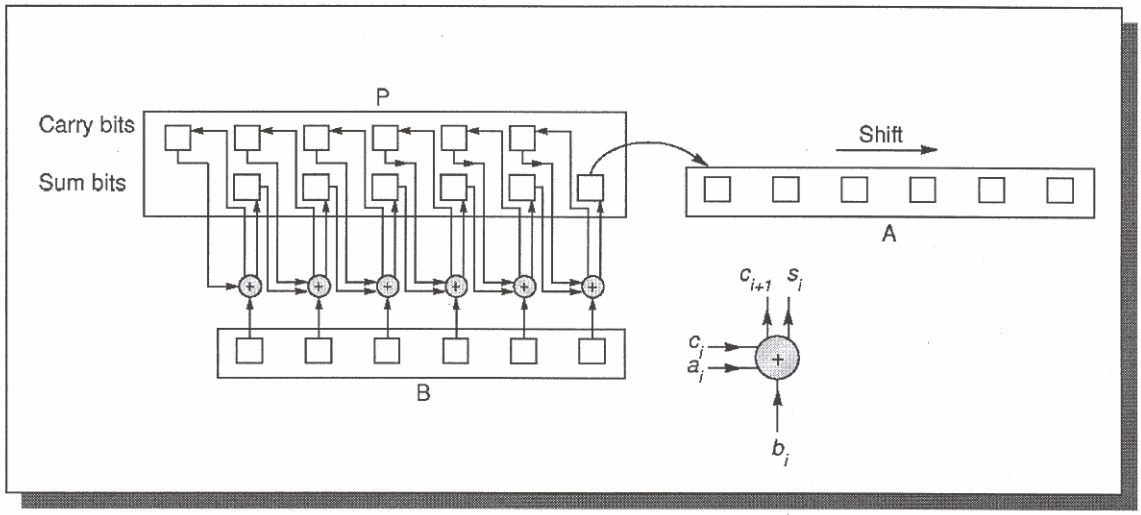

FIGURE A.24 Carry-save multiplier.

Low-order bits of A		Last bit shifted out	
$2i+1$	$2i$	$2i-1$	**Multiple**
0	0	0	0
0	0	1	$+b$
0	1	0	$+b$
0	1	1	$+2b$
1	0	0	$-2b$
1	0	1	$-b$
1	1	0	$-b$
1	1	1	0

FIGURE A.25 Multiples of *b* to use for radix-4 Booth recoding.

```
   P      A   L
 00000  1001        Multiply –7 = 1001 times –5 = 1011. B contains 1011.
+ 11011             Low order bits of A are 0, 1; L=0, so add B.
 11011  1001
 11110  1110  0     Shift right by two bits, shifting in 1s on the left.
+ 01010             Low order bits of A are 1, 0; L=0, so add –2b.
 01000  1110  0
 00010  0011  1     Shift right by two bits.
                    Product is 35 = 0100011.
```

FIGURE A.26 Multiplication of –7 times –5 using radix-4 Booth recoding.

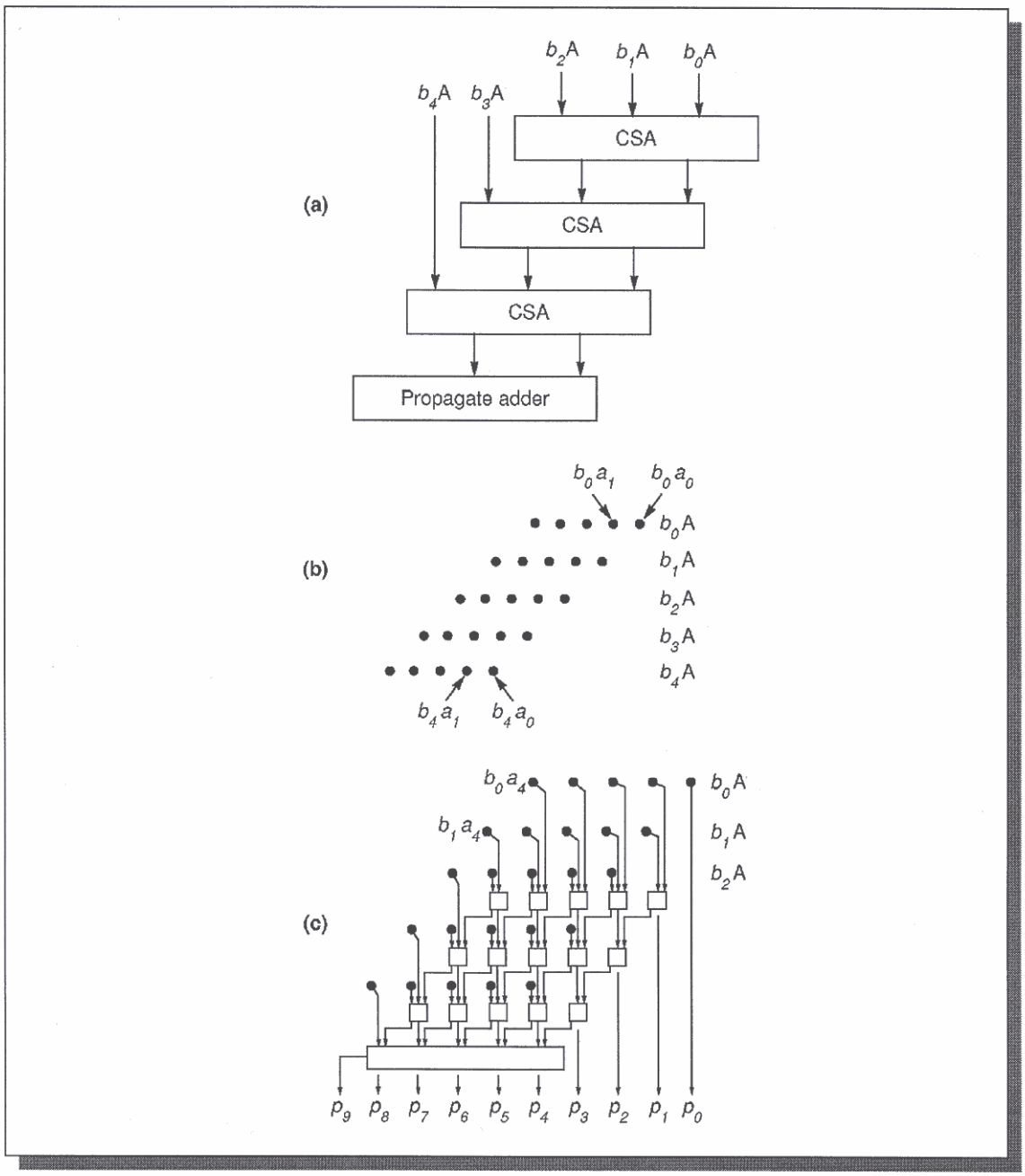

FIGURE A.27 An array multiplier.

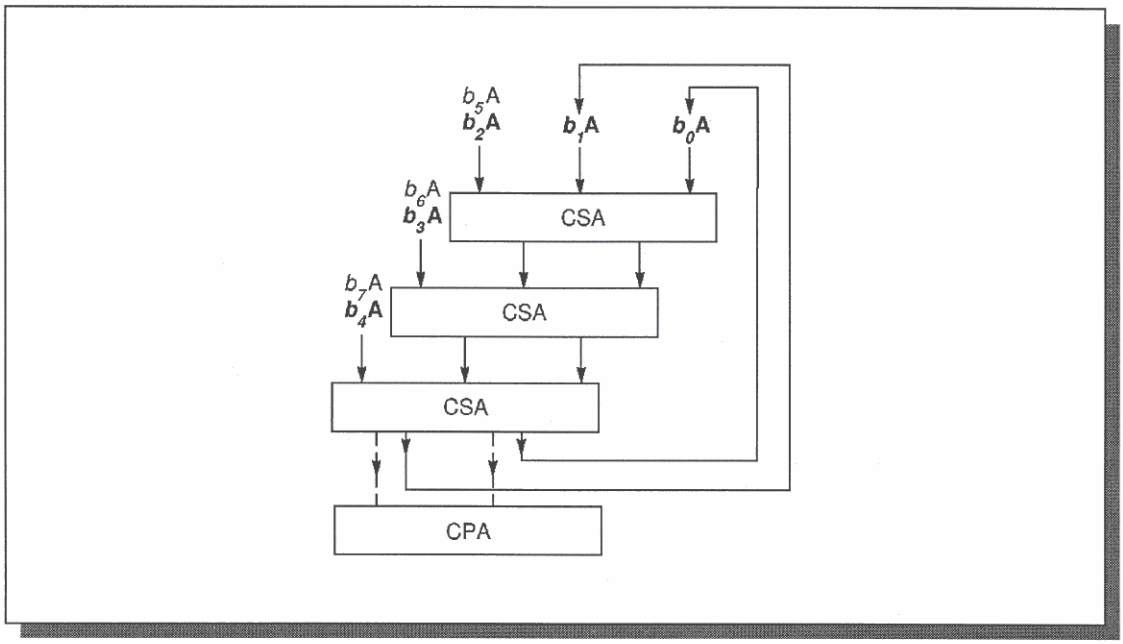

FIGURE A.28 Multipass array multiplier.

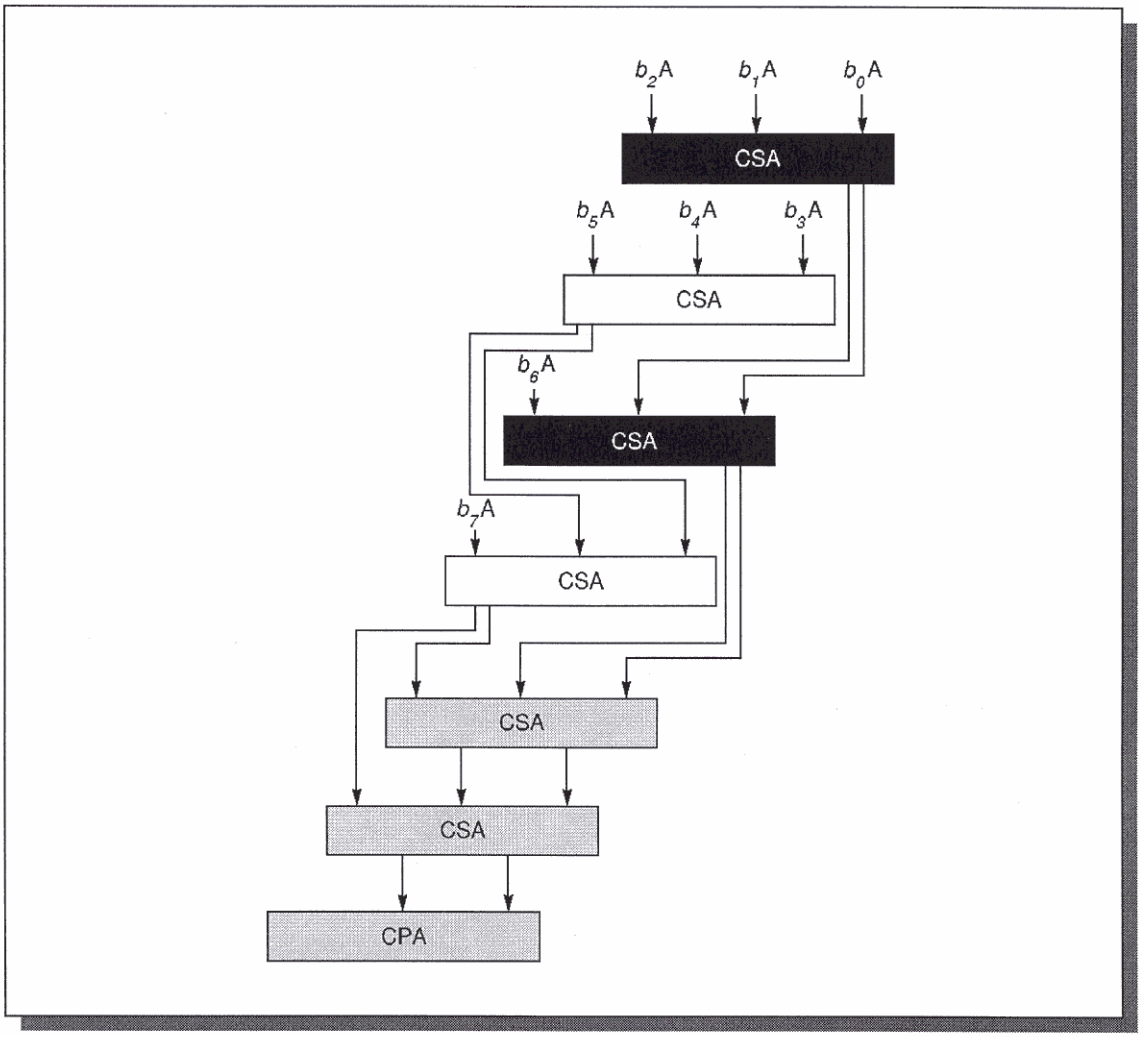

FIGURE A.29 Even/odd array.

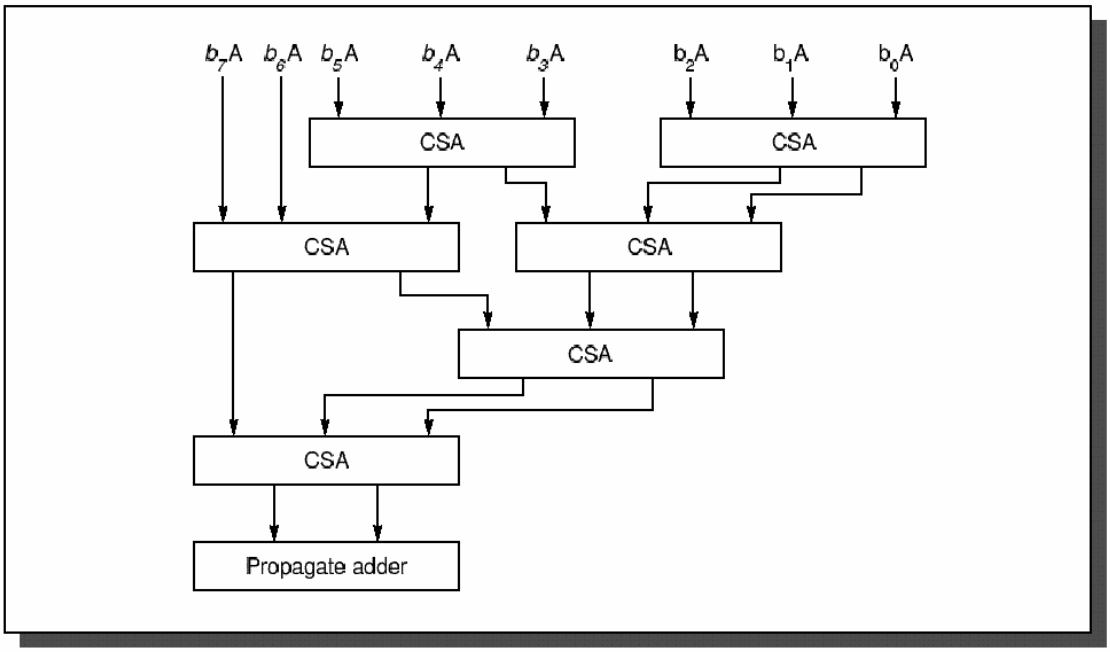

FIGURE A.30 Wallace tree multiplier.

$\begin{array}{r} 1 \\ +1 \\ \hline 1\,0 \end{array}$	$\begin{array}{r} 1 \\ +\bar{1} \\ \hline 0\,0 \end{array}$	$\begin{array}{r} \bar{1} \\ +\bar{1} \\ \hline \bar{1}\,0 \end{array}$	$\begin{array}{r} 0 \\ +0 \\ \hline 0\,0 \end{array}$	$\begin{array}{r} 1\ x \\ +0\ y \\ \hline 1\,\bar{1} \\ 0\,1 \end{array}$	if $x \geq 0$ and $y \geq 0$ otherwise
				$\begin{array}{r} \bar{1}\ x \\ +0\ y \\ \hline 0\,\bar{1} \\ \bar{1}\,1 \end{array}$	if $x \geq 0$ and $y \geq 0$ otherwise

FIGURE A.31 Signed-digit addition table.

DIVISIÓN SRT

/* División de a entre b (ambos de n bits). Ruta de datos: ver transparencia *Mult/div enteras-1* */

A := a; B := b; P := 0;

if "B, expresado con n bits, tiene k ceros a la izda." **then** "desplazar B y (P&A) k bits a la izquierda"; /* A1 */

for i:=0 **to** n-1 do

{ a: **if** "los tres bits a la izda. de P son iguales" **then** /* A2 */
{"desplazar (P&A) un bit a la izquierda"; q_i :=0} /* A3 */
else

b: **if** P<0 **then** /* A4 */
{"desplazar (P&A) un bit a la izquierda"; q_i := -1; "sumar B"};
/* A5 */
else /* A6 */
c: {"desplazar (P&A) un bit a la izquierda"; q_i := 1; "restar B"};
/* A7 */

};

if P<0 **then** {"sumar B"; restar 1 al cociente q};

"desplazar el resto (P), k bits a la derecha"

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 A.23 SRT division of $1000_2/0011_2$.

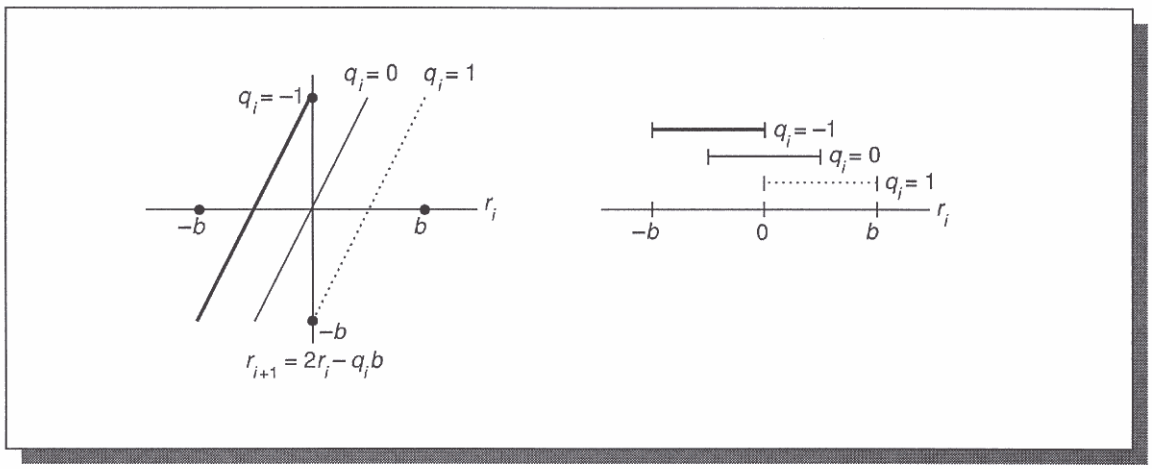

FIGURE A.32 Quotient selection for radix-2 division.

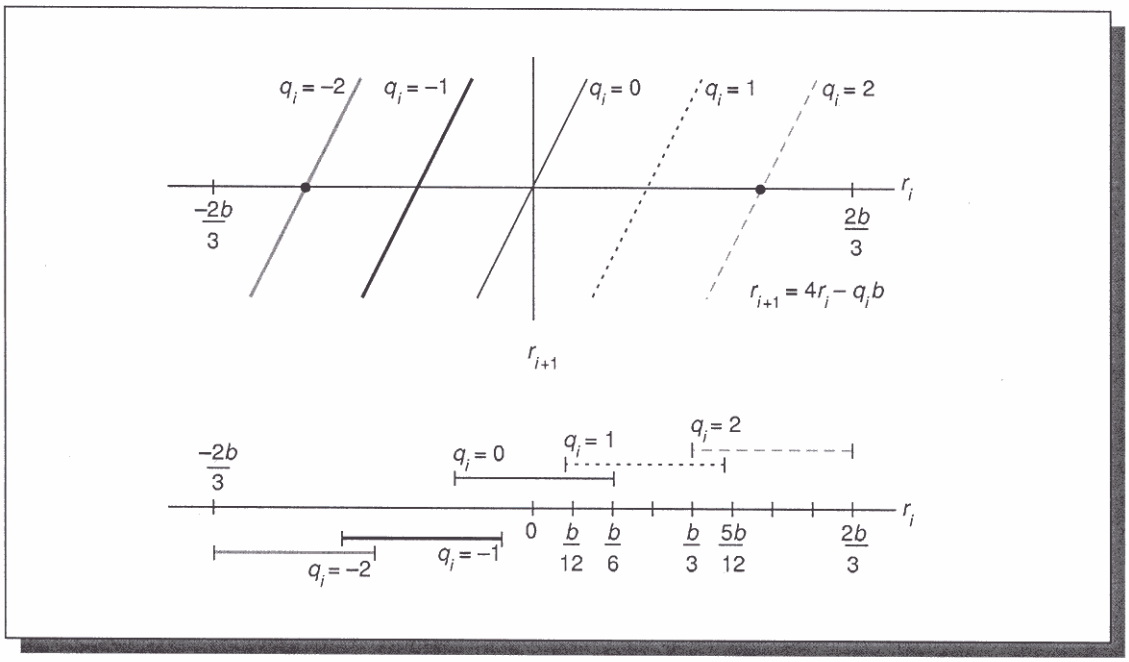

FIGURE A.33 Quotient selection for radix-4 division with quotient digits –2, –1, 0, 1, 2.

b	Range of P		q	b	Range of P		q
8	−12	−7	−2	12	−18	−10	−2
8	−6	−3	−1	12	−10	−4	−1
8	−2	1	0	12	−4	3	0
8	2	5	1	12	3	9	1
8	6	11	2	12	9	17	2
9	−14	−8	−2	13	−19	−11	−2
9	−7	−3	−1	13	−10	−4	−1
9	−3	2	0	13	−4	3	0
9	2	6	1	13	3	9	1
9	7	13	2	13	10	18	2
10	−15	−9	−2	14	−20	−11	−2
10	−8	−3	−1	14	−11	−4	−1
10	−3	2	0	14	−4	3	0
10	2	7	1	14	3	10	1
10	8	14	2	14	10	19	2
11	−16	−9	−2	15	−22	−12	−2
11	−9	−3	−1	15	−12	−4	−1
11	−3	2	0	15	−5	4	0
11	2	8	1	15	3	11	1
11	8	15	2	15	11	21	2

FIGURE A.34 Quotient digits for radix-4 SRT division with a propagate adder.

	Single	Single extended	Double	Double extended
p (bits of precision)	24	≥ 32	53	≥ 64
E_{max}	127	≥ 1023	1023	≥ 16383
E_{min}	−126	≤ −1022	−1022	≤ −16382
Exponent bias	127		1023	

FIGURE A.7 Format parameters for the IEEE 754 floating-point standard.

Exponent	Fraction	Represents
$e = E_{min} - 1$	$f = 0$	±0
$e = E_{min} - 1$	$f \neq 0$	$0.f \times 2^{E_{min}}$
$E_{min} \leq e \leq E_{max}$	—	$1.f \times 2^{e}$
$e = E_{max} + 1$	$f = 0$	$\pm\infty$
$e = E_{max} + 1$	$f \neq 0$	NaN

FIGURE A.8 Representation of special values.

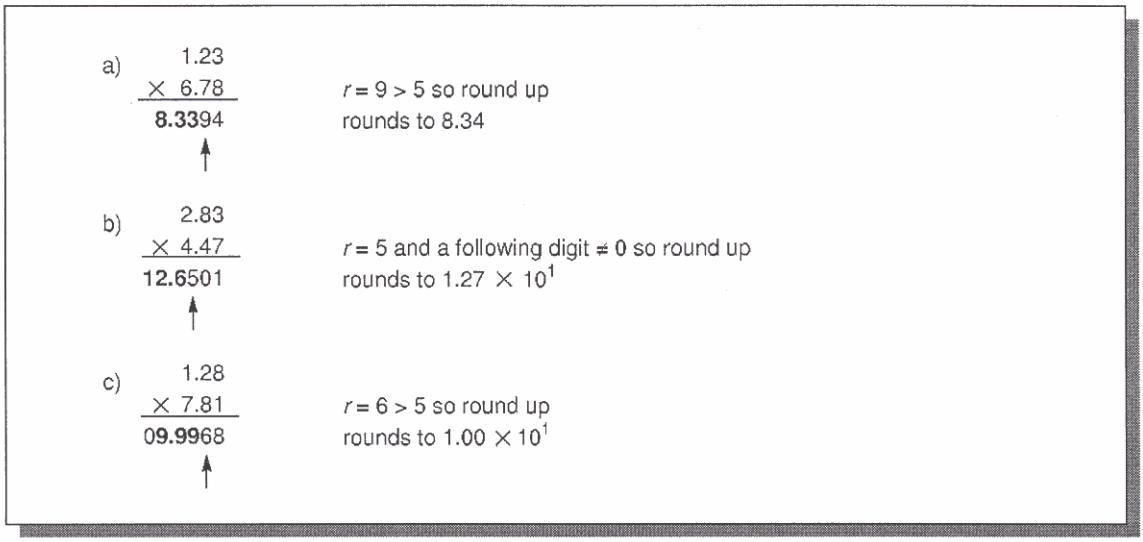

FIGURE A.9 Examples of rounding a multiplication.

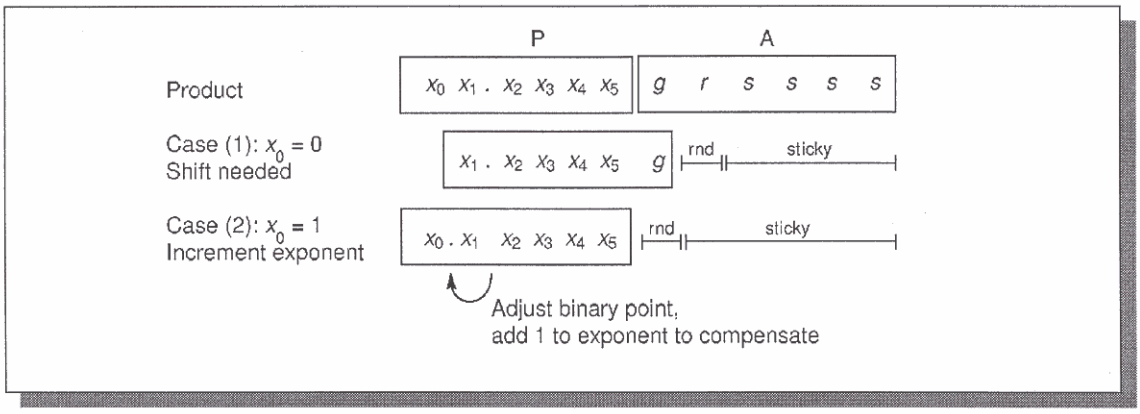

FIGURE A.10 The two cases of the floating-point multiply algorithm.

Rounding mode	Sign of result ≥ 0	Sign of result < 0
$-\infty$		+1 if $r \vee s$
$+\infty$	+1 if $r \vee s$	
0		
Nearest	+1 if $r \wedge p_0$ or $r \wedge s$	+1 if $r \wedge p_0$ or $r \wedge s$

FIGURE A.11 Rules for implementing the IEEE rounding modes.

Ejemplos de sumas/restas en punto flotante con 6 bits de precisión (mantisas en base 2).

1º) $1{,}10011 + 1{,}10001 \times 2^{-5}$

$$\begin{array}{r} 1{,}10011 \\ +\ {,}00001\mathbf{1}0001 \end{array}$$

Con un sumador de 6 bits

$$\begin{array}{r} 1{,}10011 \\ +\ {,}00001 \\ \hline 1{,}10100 \end{array}$$

Bit de redondeo (en negrita) = 1. Bit retenedor s= 1 ($0 \vee 0 \vee 0 \vee 1$).

Redondeo (sumar 1) => mantisa final = 1,10101

2º) $1{,}11011 + 1{,}01001 \times 2^{-2}$

$$\begin{array}{r} 1{,}11011 \\ +\ {,}0101001 \end{array}$$

Con un sumador de 6 bits

$$\begin{array}{r} 1{,}11011 \\ +\ {,}01010 \\ \hline 10{,}0010\mathbf{1} \end{array}$$

Bit de redondeo (en negrita) = 1. Bit retenedor s= 1 ($0 \vee 1$).

Redondeo (sumar 1) => mantisa final = 10,0011

3º) $1{,}00000 - 1{,}01111 \times 2^{-6}$

$$\begin{array}{r} 1{,}00000 \\ -\ {,}00000101111 \end{array}$$

Usamos el complemento a 2 del sustraendo para convertir la resta en suma

$$\begin{array}{r} 1{,}00000 \\ +\ 1{,}11111\mathbf{01}0001 \end{array}$$

Con un sumador de 6 bits

$$\begin{array}{r} 1{,}00000 \\ +\ 1{,}11111 \\ \hline 0{,}11111 \end{array}$$

Bit de guarda (negrita) = 0 => Resultado sin redondeo = 0,111110.

Bit de redondeo (negrita) = 1. Bit retenedor s= 1 ($0 \vee 0 \vee 0 \vee 1$).

Redondeo (sumar 1) => mantisa final = 0,111111.

ALGORITMO DE SUMA EN PUNTO FLOTANTE

/* a_1 y a_2: números por sumar.
e_i: exponente de a_i. s_i: mantisa desempaquetada de a_i */

1: **if** e1 < e2 **then** {"intercambiar a_1 y a_2"};
d := e1 – e2/* d ≥ 0 */; e := e1 /* exponente inicial del resultado */

2: **if** sign(a1) ≠ sign(a2) **then** s2 := Ca2(s2) /* Ca2: complemento a 2 */

3: /* PS2: registro de p bits */ PS2 := s2;
"desplazar PS2, d posiciones a la derecha
(entrando unos por la izda. si s2 se complementó en el paso 2) y,
de los bits que han salido por la derecha de PS2, llamar **g** al más
significativo, **r** al siguiente y **s** a la disyunción lógica de los demás"

4: S := s1 + PS2;
if sign(a1) ≠ sign(a2) ∧ (Bit más significativo de S) = 1 ∧
∧ no hay acarreo de salida
then /* S<0 (d=0) */ S := Ca2(S)

5: **if** sign(a1) = sign(a2) ∧ hay acarreo de salida
then "desplazar S una posición a la dcha. e introducir un 1 (el acarreo) por la izda."
else "desplazar S hacia la izda hasta que quede normalizado (en el primer desplazamiento introducir por la derecha el bit g, en los siguientes introducir ceros)"**;**
"Ajustar el exponente del resultado, e"

6: **if** "S fue desplazado a la derecha" **then**
{ s := g ∨ r ∨ s; r := "bit menos significativo de S antes de desplazar"**};**
if "S no fue desplazado" **then {** s := r ∨ s; r := g**};**
if "S fue desplazado 2 ó más posiciones a la izquierda"
then {r := 0; s:= 0**}**

7: "redondear S"; **if** "se produce acarreo de salida" **then**
"desplazar S un bit a la dcha., introducir un 1 por la izda. y ajustar el exponente, e"
/* **S** es la mantisa y **e** el exponente del resultado */

8: /* Calcular el signo del resultado, sr */
if sign(a1) ≠ sign(a2) ∧ e1=e2 **then**
if "se complementó S en el paso 4" /* $|PS2| > s1$ */
then sr := sign(a2) **else** sr := sign(a1)
else sr := sign(a1) /* e1 > e2 ∨ sign(a1) = sign(a2) */

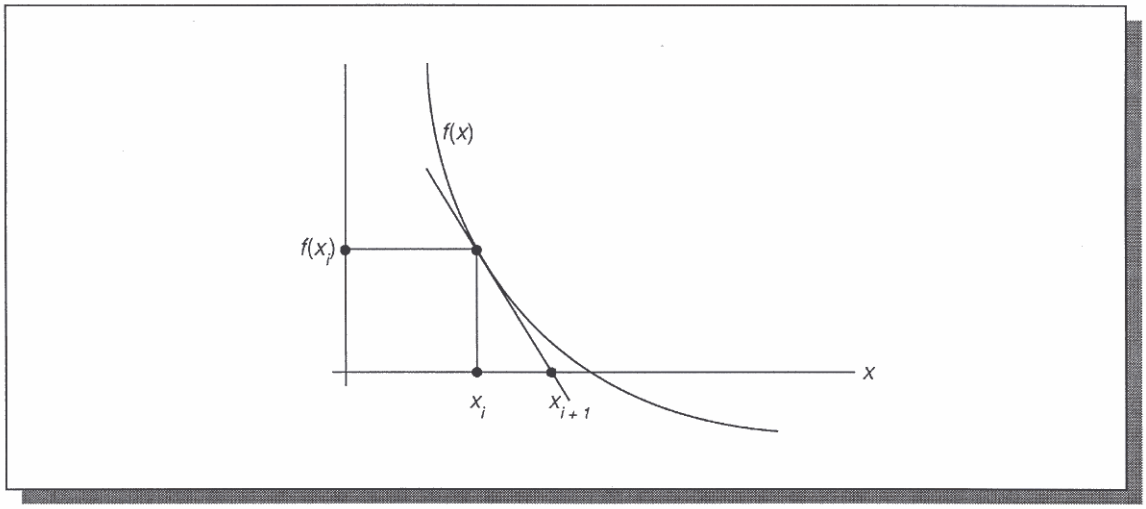

FIGURE A.13 Newton's iteration for zero finding.

Ecuación de la tangente a la curva:

$$y - f(x_i) = f'(x_i)\,(x - x_i)$$

Corte con el eje de abscisas (y = 0):

$$-f(x_i) = f'(x_i)\,(x - x_i)$$

$$x_{i+1} = x = x_i - f(x_i)/f'(x_i)$$

Aplicación a la función

$$f(x) = x^{-1} - b$$

Se tiene

$$f'(x) = -x^{-2}$$

y, por tanto,

$$x_{i+1} = x_i - f(x_i)/f'(x_i) = x_i + x_i - b\,x_i^2$$

En definitiva

$$\mathbf{x_{i+1} = 2\,x_i - b\,x_i^2 = x_i\,(2 - b\,x_i)}$$