Chapter 9 Computer Arithmetic

Integer Representation

(1) Sign-Magnitude Method

+18 = [0]0 0 1 0 0 1 0
-18 = [1]0 0 1 0 0 1 0

Drawbacks –

(2) Two's Complement Method

Ex. (-3) in 8-bit number system

+3	0 0 0 0 0 0 1 1
complement	1 1 1 1 1 1 0 0
add 1	1 1 1 1 1 1 0 1 (= -3)

Ex. 5 – 3 = 5 + (-3)

```
 5      0 0 0 0 0 1 0 1
-3      1 1 1 1 1 1 0 1  +
        -----------------
   [1]  0 0 0 0 0 0 1 0   (= 2)
ignore
```

Note. No subtractor is necessary.

Note: If two numbers are added and they are both positive or both negative, then overflow occurs iff the result has the opposite sign.

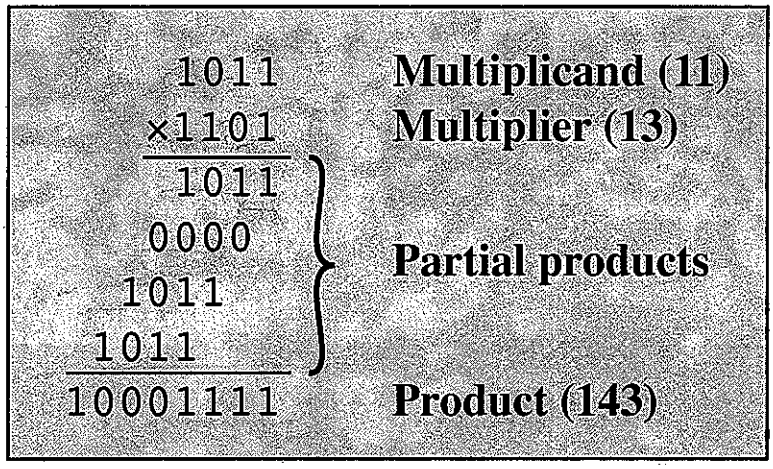
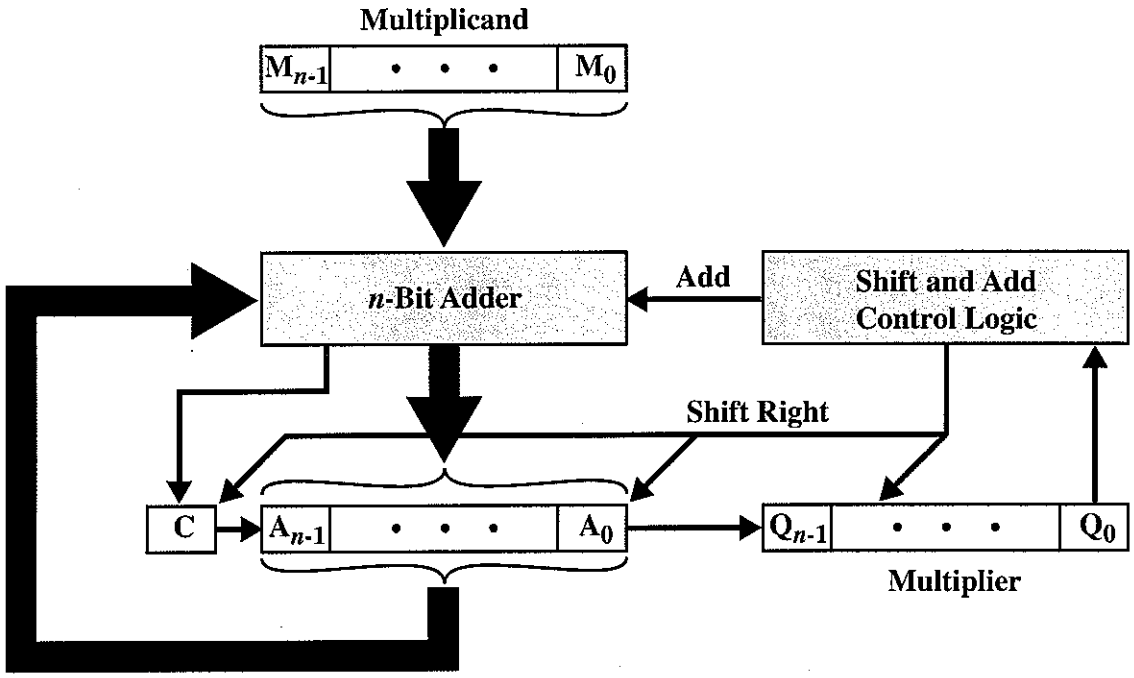


Figure 9.7 Multiplication of Unsigned Binary Integers



(a) Block Diagram

C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

(b) Example from Figure 9.7 (product in A, Q)

Figure 9.8 Hardware Implementation of Unsigned Binary Multiplication

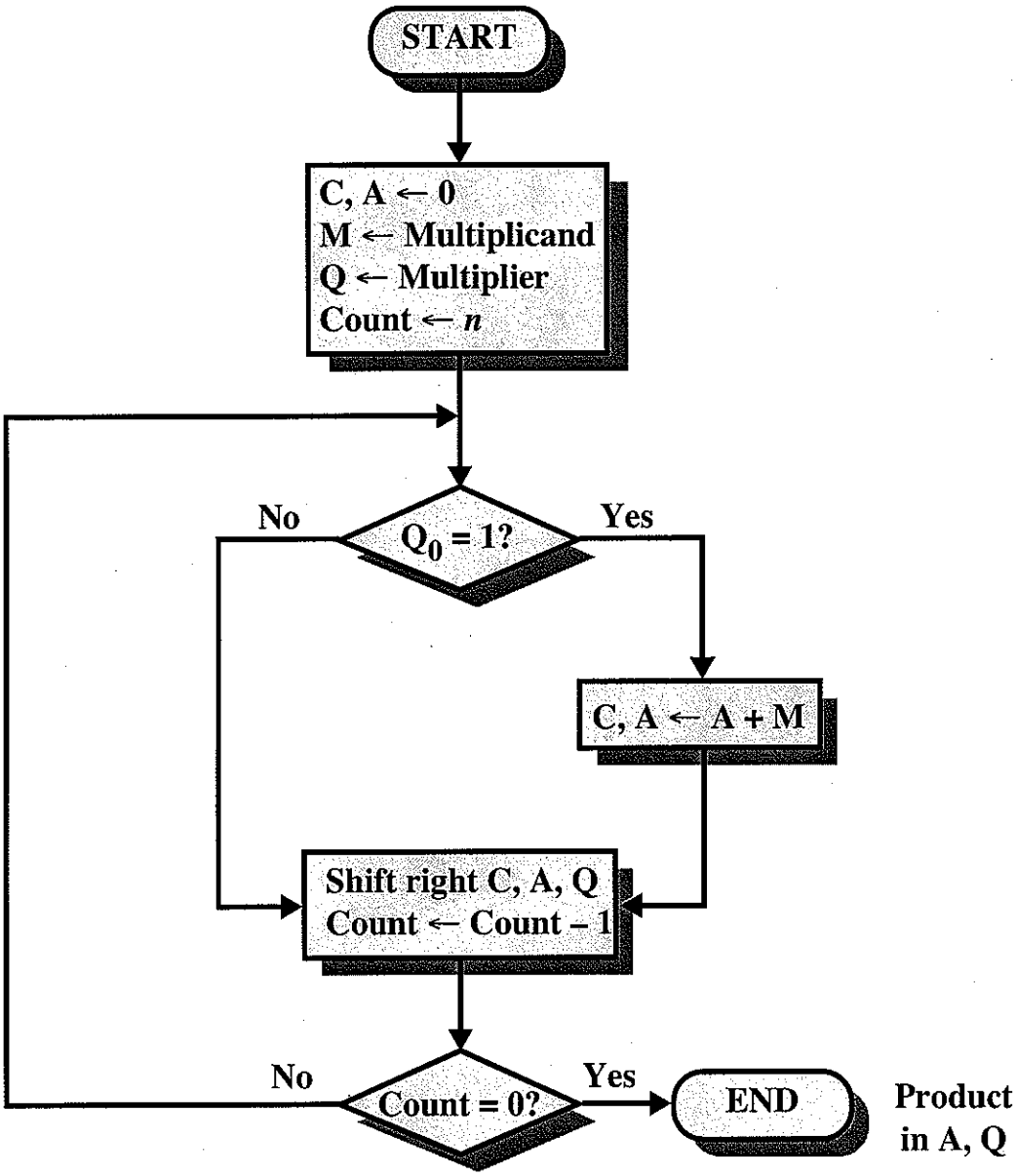


Figure 9.9 Flowchart for Unsigned Binary Multiplication

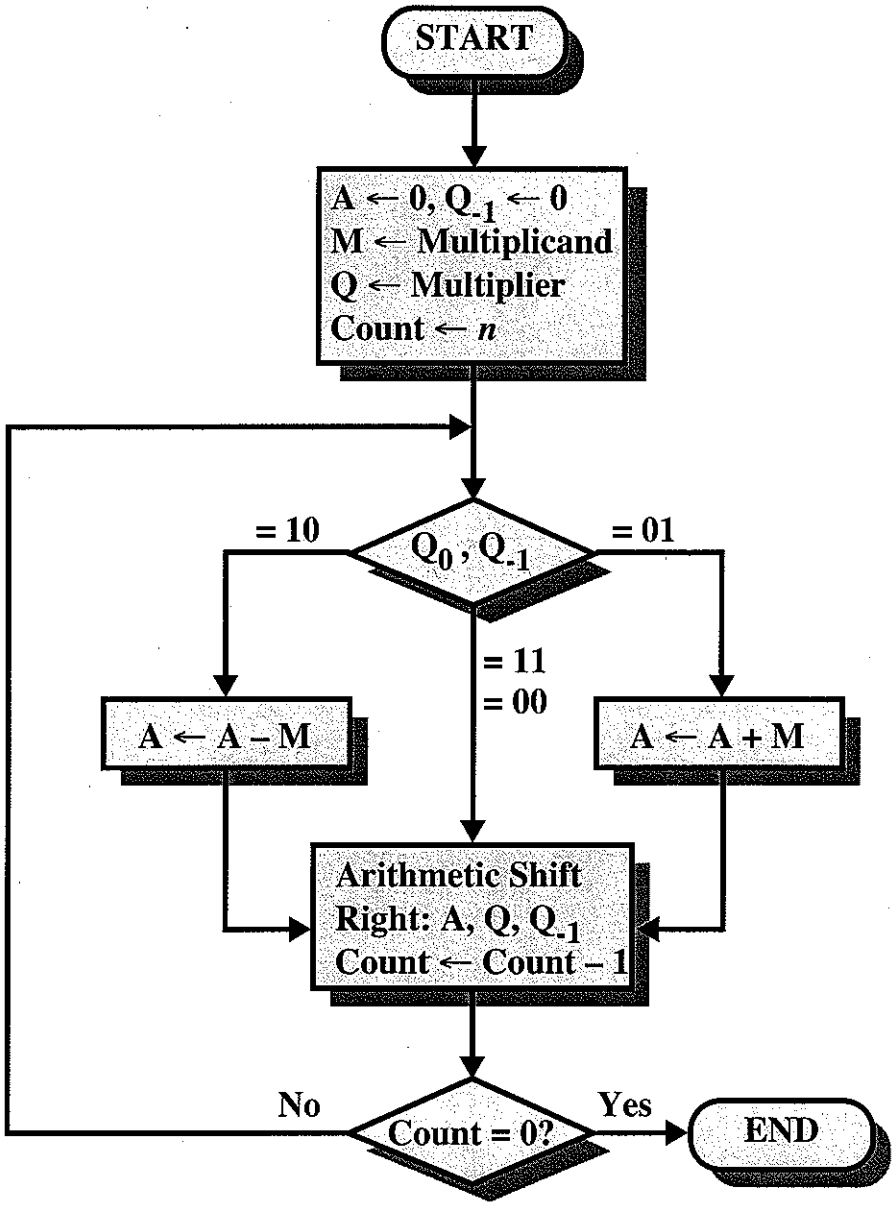


Figure 9.12 Booth's Algorithm for Twos Complement Multiplication

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A ← A − M	First Cycle
1100	1001	1	0111	Shift	First Cycle
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A ← A + M	Third Cycle
0010	1010	0	0111	Shift	Third Cycle
0001	0101	0	0111	Shift	Fourth Cycle

Figure 9.13 Example of Booth's Algorithm (7 × 3)

Booth's algorithm - continued

Ex. $0\ 0\ \boxed{0\ (1}\ 1\ 1\ \boxed{1)\ 0}$ (end of run; beginning of run)

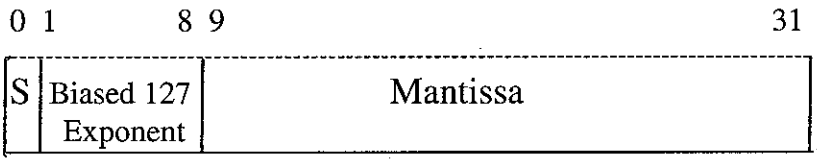
Note. $2^4 + 2^3 + 2^2 + 2^1 \equiv 2^5 - 2^1$

Ex. (7) x (-3)

A	Q	Q_{-1}		
0 0 0 0	1 1 0 1	0	**Initial**	
1 0 0 1	1 1 0 1	0	**A ← A − M**	First cycle
1 1 0 0	1 1 1 0	1	**shift**	
0 0 1 1	1 1 1 0	1	**A ← A + M**	Second cycle
0 0 0 1	1 1 1 1	0	**shift**	
1 0 1 0	1 1 1 1	0	**A ← A − M**	Third cycle
1 1 0 1	0 1 1 1	1	**shift**	
1 1 1 0	1 0 1 1	1	**shift**	Fourth cycle

Floating-Point Numbers

$$\pm S \times B^{\pm E}$$



-127 ~ +128

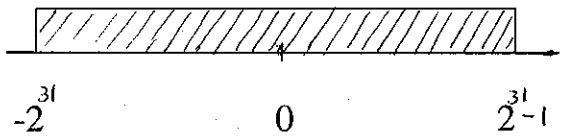
Ex. $1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010\ldots$

$1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010\ldots$

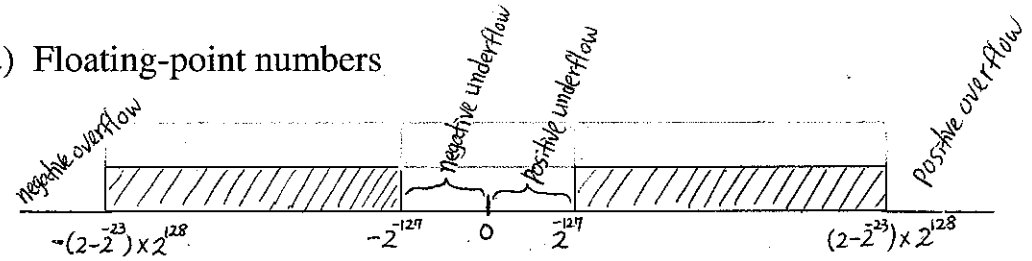
Note. Normalized: $+ 1.bbb..b * 2^{\pm E}$
Base: Either 2 or 16 (IBM S/370)

Expressible numbers in 32-bit word

(1) 2's complement integers



(2) Floating-point numbers

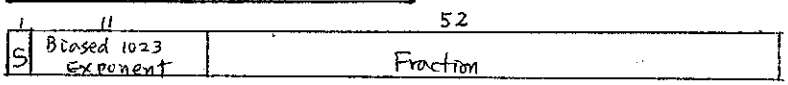


IEEE Standard

Single form

$10^{-38} - 10^{38}$

Double form



$10^{-308} - 10^{308}$

Floating-point Arithmetic

Addition and Subtraction

1. **Check for zero**
2. **Align the significands** // smaller number is shifted right
3. **Add or subtract the significands**
4. **Normalize the result** // shift left until most-significant digit is nonzero.

Example. (decimal, 3 significands)

$123 + 4.56$

Normalized: $1.23 \times 10^2 + 4.56 \times 10^0$

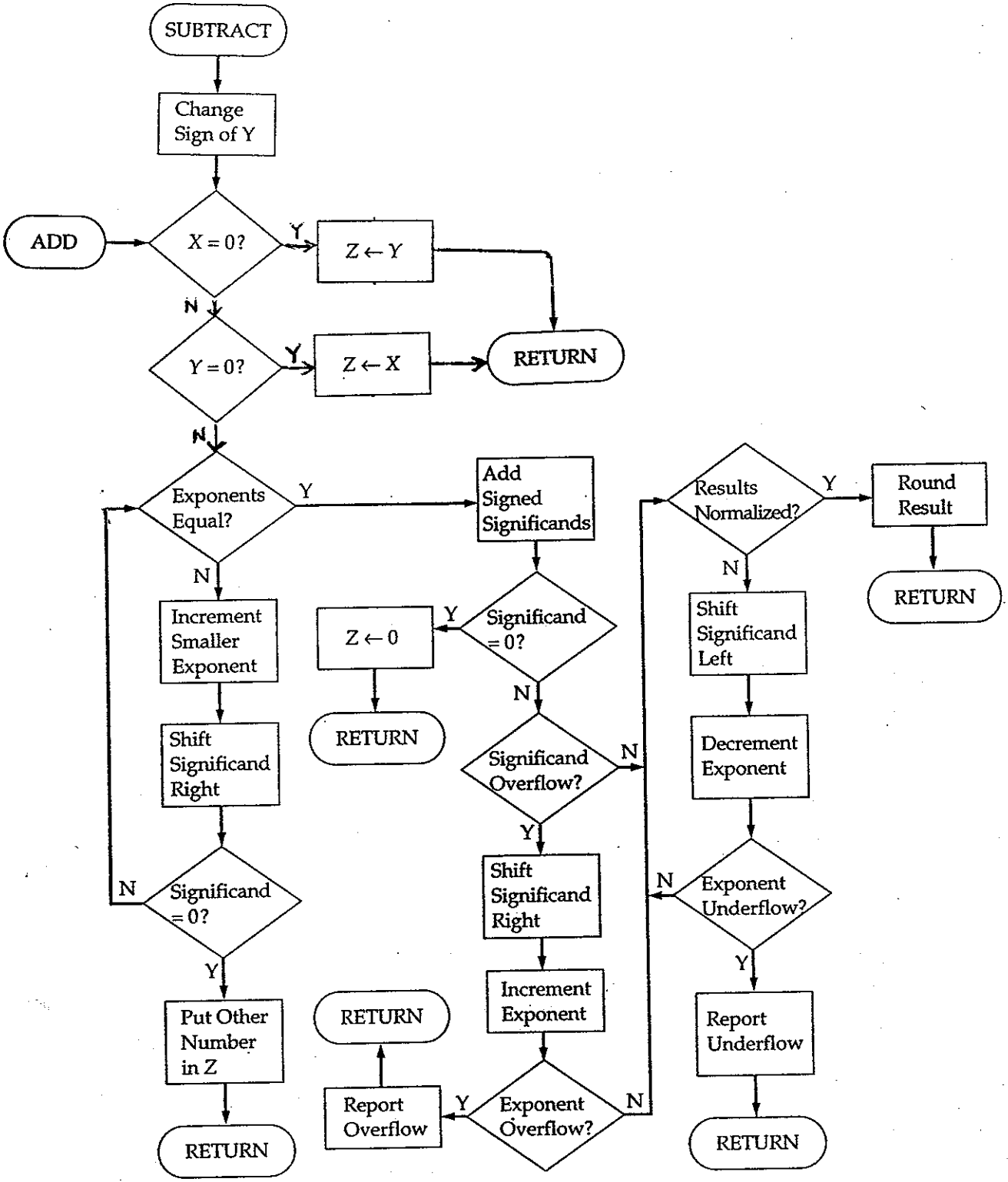
Align

$$\begin{array}{rl} & 1.23 \times 10^2 \\ +) & 0.0456 \times 10^2 \\ \hline & 1.2756 \times 10^2 \end{array}$$

truncate: 1.27×10^2

round: 1.28×10^2

Floating-point addition and subtraction ($z \leftarrow x \pm y$)



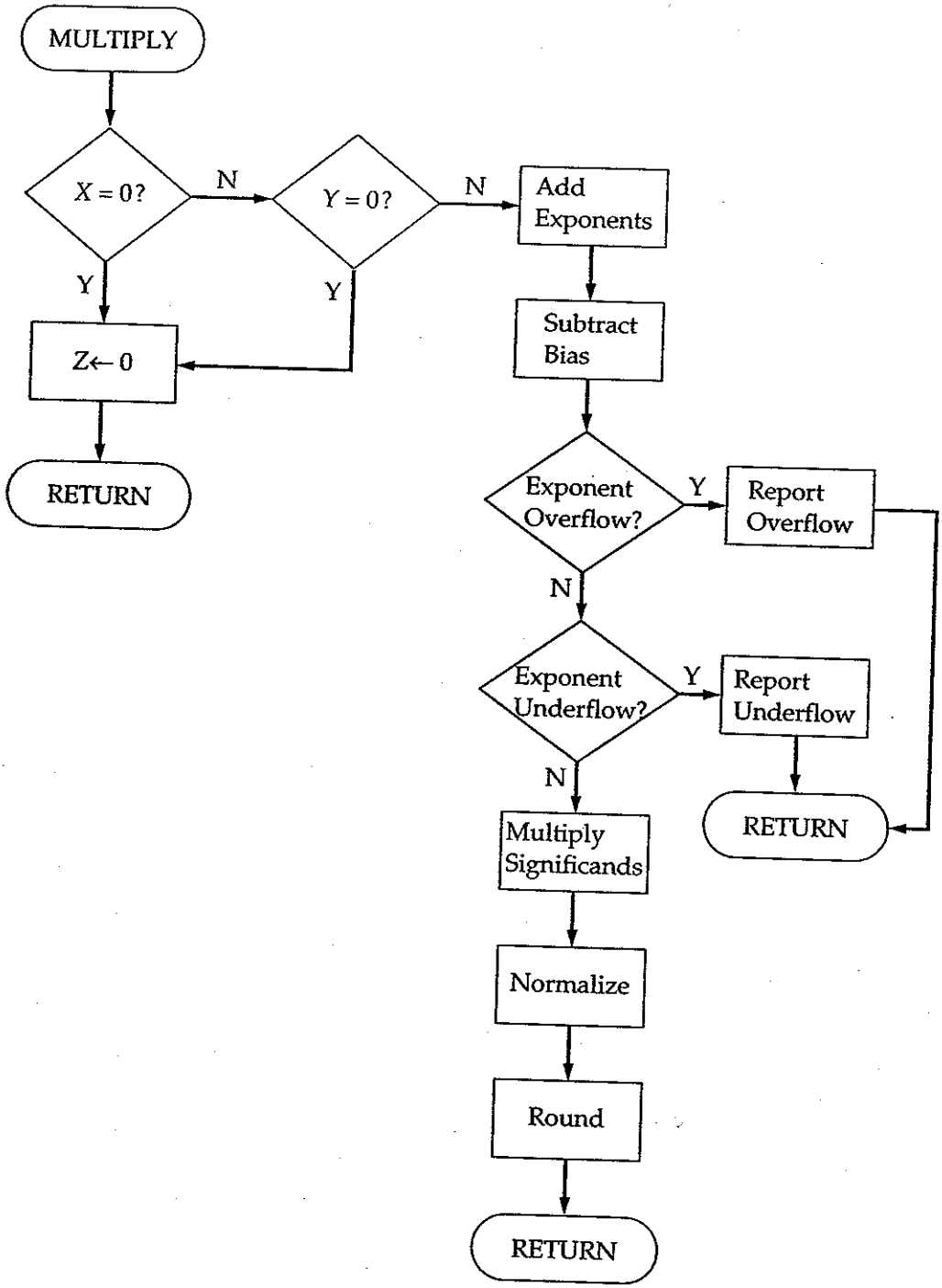


FIGURE 9.23. Floating-point multiplication ($z \leftarrow x \times y$)

Precision Consideration

1. Guard Bits

Motivation.

$X = 1.00...00 \times 2$
$Y = 1.11...11 \times 2$

For X – Y, Y should be shifted one bit to the right (to align).
➔ Y will lose one bit of significance.

$$\begin{aligned} x &= 1.000.....00 \times 2^{1} \\ -y &= 0.111.....11 \times 2^{1} \\ z &= 0.000.....01 \times 2^{1} \\ &= 1.000.....00 \times 2^{-22} \end{aligned}$$

(a) Binary example, without guard bits

$$\begin{aligned} x &= .100000 \times 16^{1} \\ -y &= .0FFFFF \times 16^{1} \\ z &= .000001 \times 16^{1} \\ &= .100000 \times 16^{-4} \end{aligned}$$

(c) Hexadecimal example, without guard bits

$$\begin{aligned} x &= 1.000.....00\ 0000 \times 2^{1} \\ -y &= 0.111.....11\ 1000 \times 2^{1} \\ z &= 0.000.....00\ 1000 \times 2^{1} \\ &= 1.000.....00\ 0000 \times 2^{-23} \end{aligned}$$

(b) Binary example, with guard bits

$$\begin{aligned} x &= .100000\ 00 \times 16^{1} \\ -y &= .0FFFFF\ F0 \times 16^{1} \\ z &= .000000\ 10 \times 16^{1} \\ &= .100000\ 00 \times 16^{-5} \end{aligned}$$

(d) Hexadecimal example, with guard bits

2. Rounding Policy

- Round to nearest number (default)

Example.

(1) ....xxx | 1 0 0 0 1 0 ... roundup
(2) ....xxx | 0 1 1 0 1 0 ... truncate

(23)

Case of tie: .. xxx | 1 0 0 0 0 ...

Solution.

..xx1 | 1 0 0 0 0 ... roundup
..xx0 | 1 0 0 0 0 ... truncate