Computer Organization & Architecture

Unit-2 Notes

By

Dr. Lalit Saraswat

• Carry Look Ahead Adder

What is carry look ahead adder?

Definition:-

"A carry-look ahead adder (CLA) is a type of fast parallel adder used in digital logic to calculate the carry signals in advance from the input signals."

They reduce carry propagation time and implement addition of binary numbers.

It is also known as:

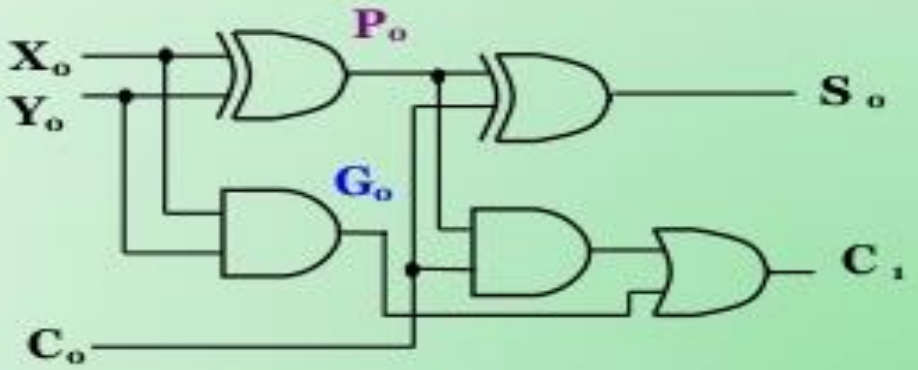
Carry look ahead generator or fast adder or

Carry predictor.

It is an improvement over **'Ripple carry adder'** circuit.

Carry Look Ahead Adder

- Consider the full adder



where intermediate signals are labelled as P_o, G_o, and defined as:

$$P_o = X_o \oplus Y_o$$

$$G_o = X_o \cdot Y_o$$

- The outputs, C_1,S_0, in terms of P_0 ,G_0 ,C_0 , are:

$$S_0 = P_0 \oplus C_0 \quad \ldots(1)$$

$$C_1 = G_0 + P_0.C_0 \quad \ldots(2)$$

- If you look at equation (2),

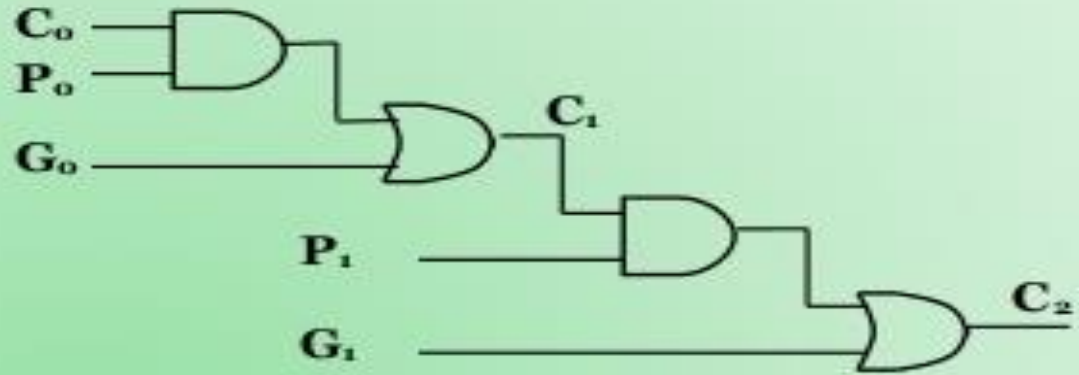
$G_0 = X_0.Y_0$ is a *carry generate* signal

$P_0 = X_0 \oplus Y_0$ is a *carry propagate* signal

Carry Look Ahead Adder

- For 4-bit ripple-carry adder, the equations to obtain four carry signals are:

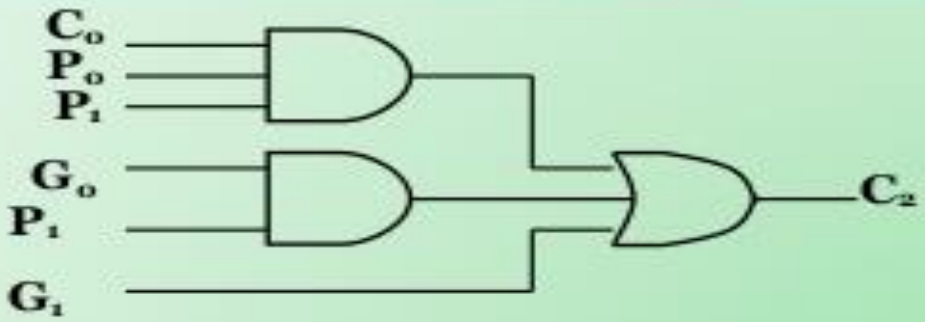
$$C_1 = G_0 + P_0.C_0$$
$$C_2 = G_1 + P_1.C_1$$
$$C_3 = G_2 + P_2.C_2$$
$$C_4 = G_3 + P_3.C_3$$

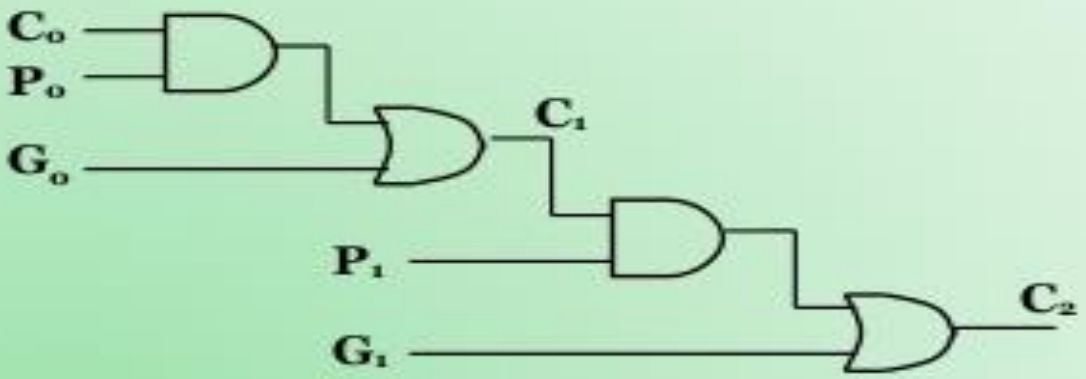


- 4-level circuit for $C_2 = G_1 + P_1.C_1$
- These formula are deeply nested, as shown here for C_2:
- Nested formula/gates cause ripple-carry propagation delay.
- Can reduce delay by expanding and flattening the formula for carries. For example, $C_2 = G_1 + P_1.C_1 = G_1 + P_1.(G_0 + P_0C_0)$

$$= G_1 + P_1.G_0 + P_1.P_0.C_0$$

Carry Look Ahead Adder



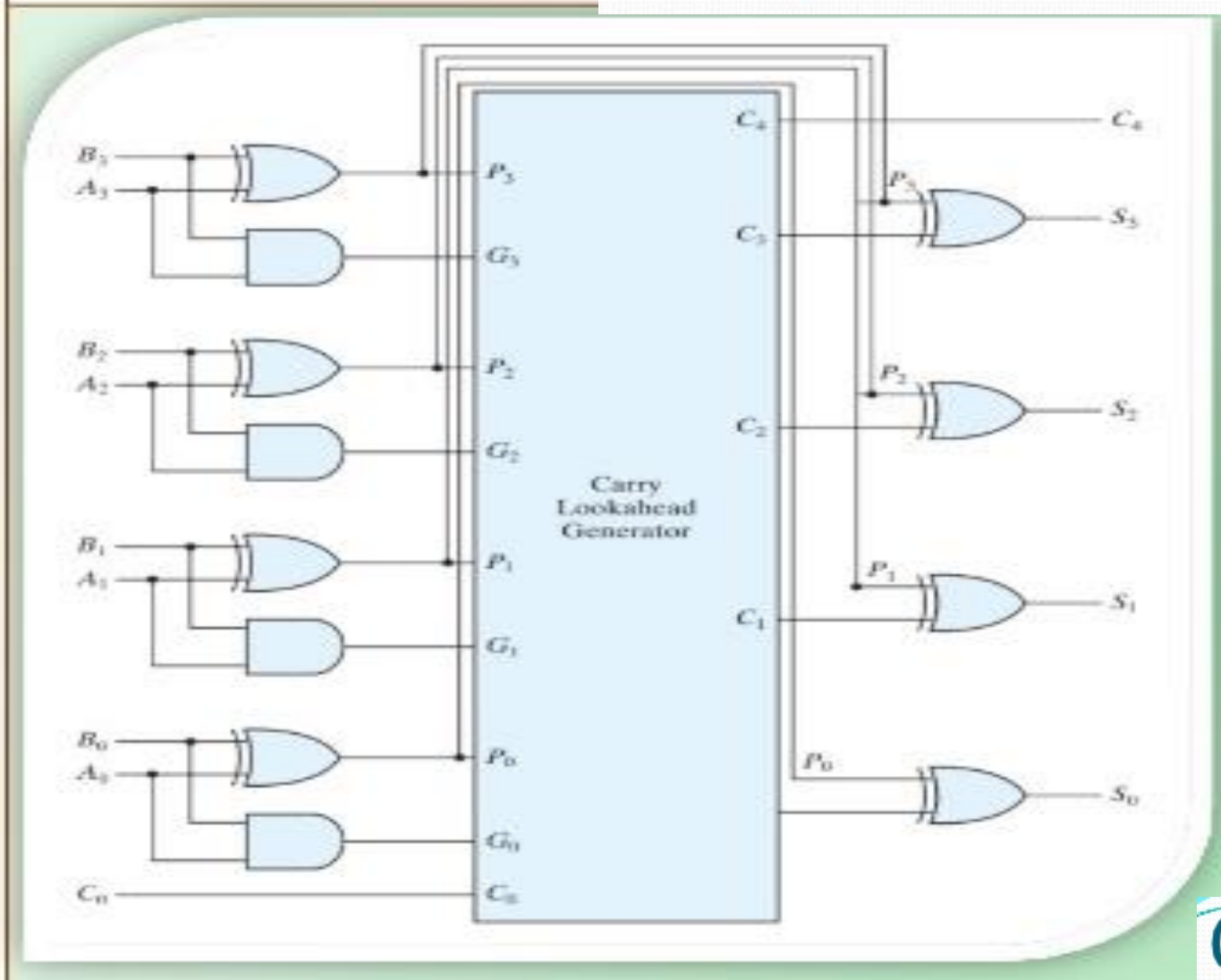


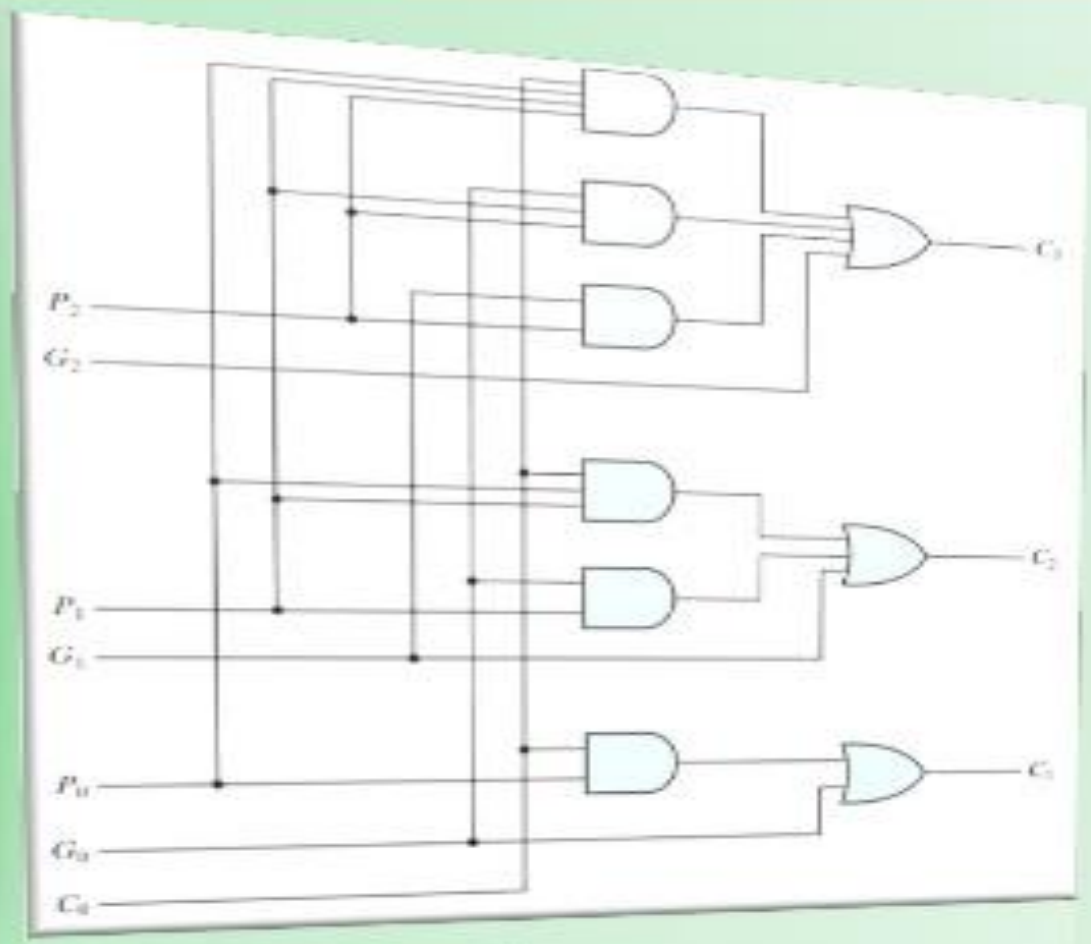
- Other carry signals can also be similarly flattened.

$$C_3 = G_2 + P_2C_2 = G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0)$$
$$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

$$C_4 = G_3 + P_3C_3$$
$$= G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0)$$
$$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

Carry Look Ahead Adder





Carry Look Ahead generator

Advantages of carry look ahead adder

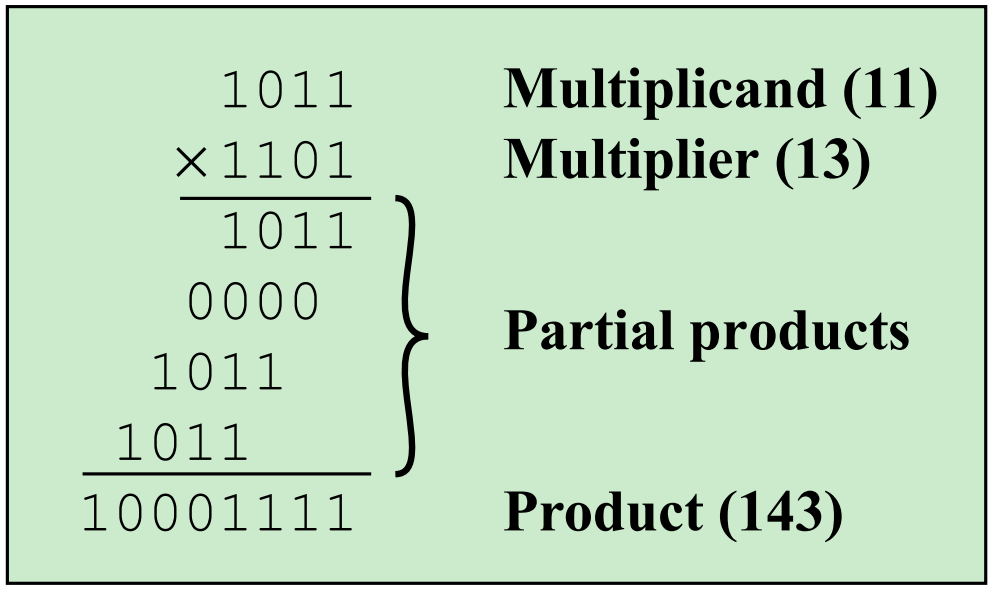
1. Like ripple carry adder we need not to wait for the propagation of carries to get the sum.
2. A carry-look ahead adder improves speed by reducing the amount of time required to determine carry bits.
3. The carry-look ahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger value bits.

Disadvantages of carry look ahead adder

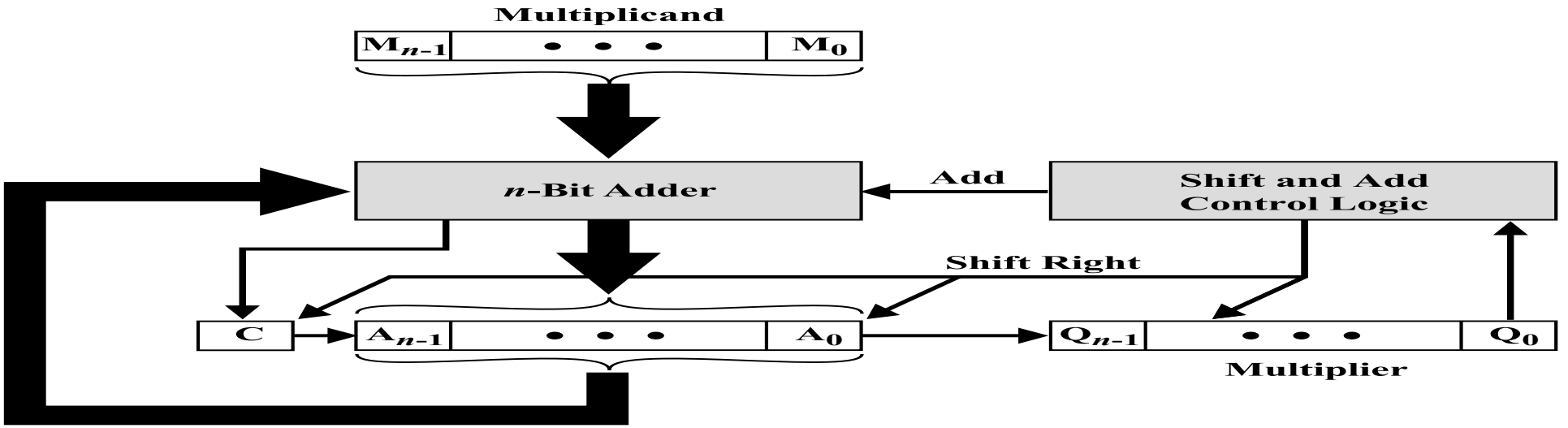
The disadvantage of CLA is that the carry logic block gets very complicated for more than 4-bits.

For that reason, CLAs are usually implemented as 4-bit modules and are used in a hierarchical structure to realize adders that have multiples of 4-bits.

Multiplication



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 (product in A, Q)

Hardware Implementation of Unsigned Binary Multiplication

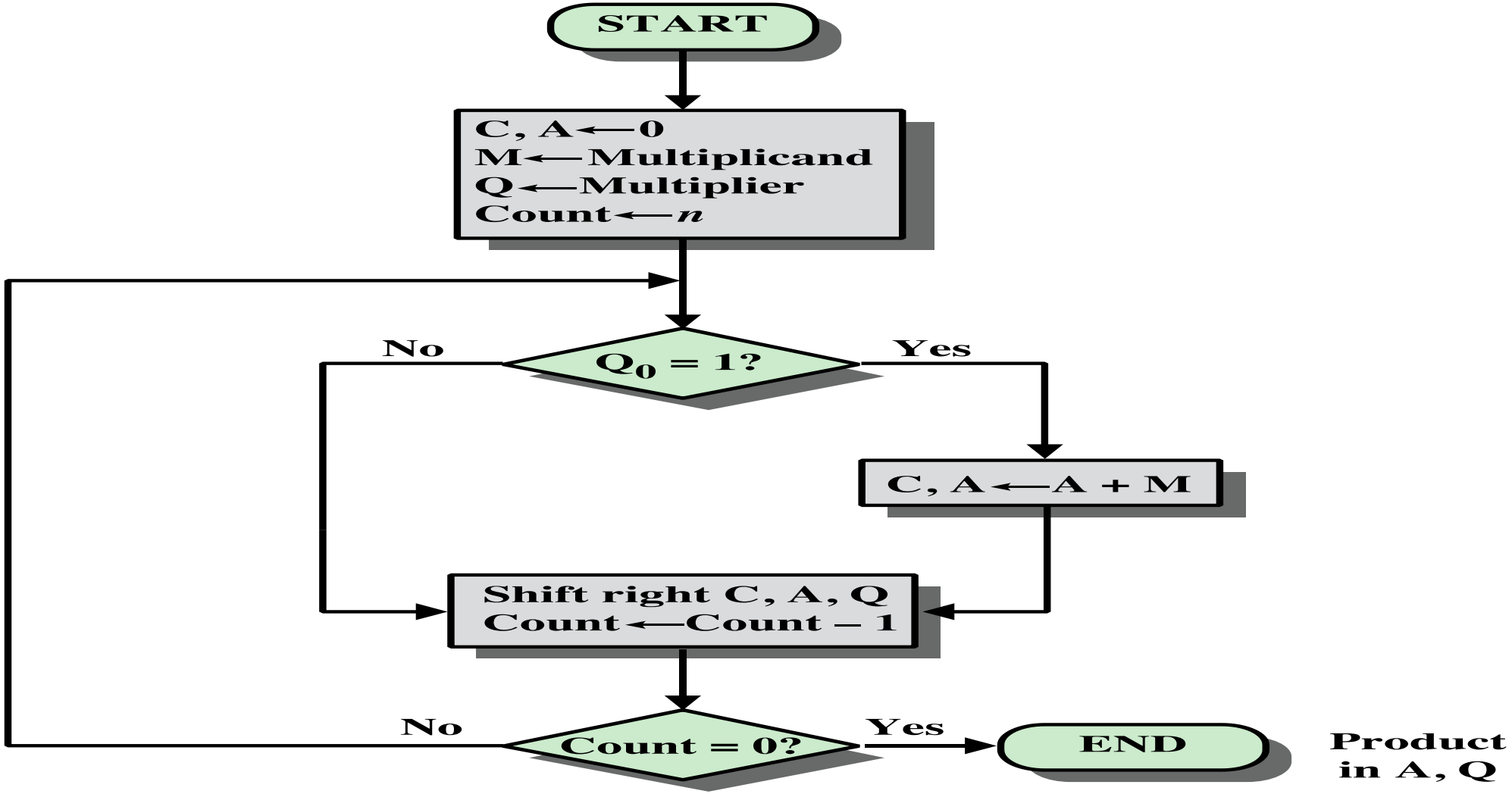


Figure Flowchart for Unsigned Binary Multiplication

BOOTH'S MULTIPLICATION ALGORITHM

What is booth's algorithm?

- Booth's multiplication algorithm is an algorithm which multiplies 2 signed or unsigned integers in 2's complement.
- This approach uses fewer additions and subtractions than more straightforward algorithms.

Booth's Algorithm

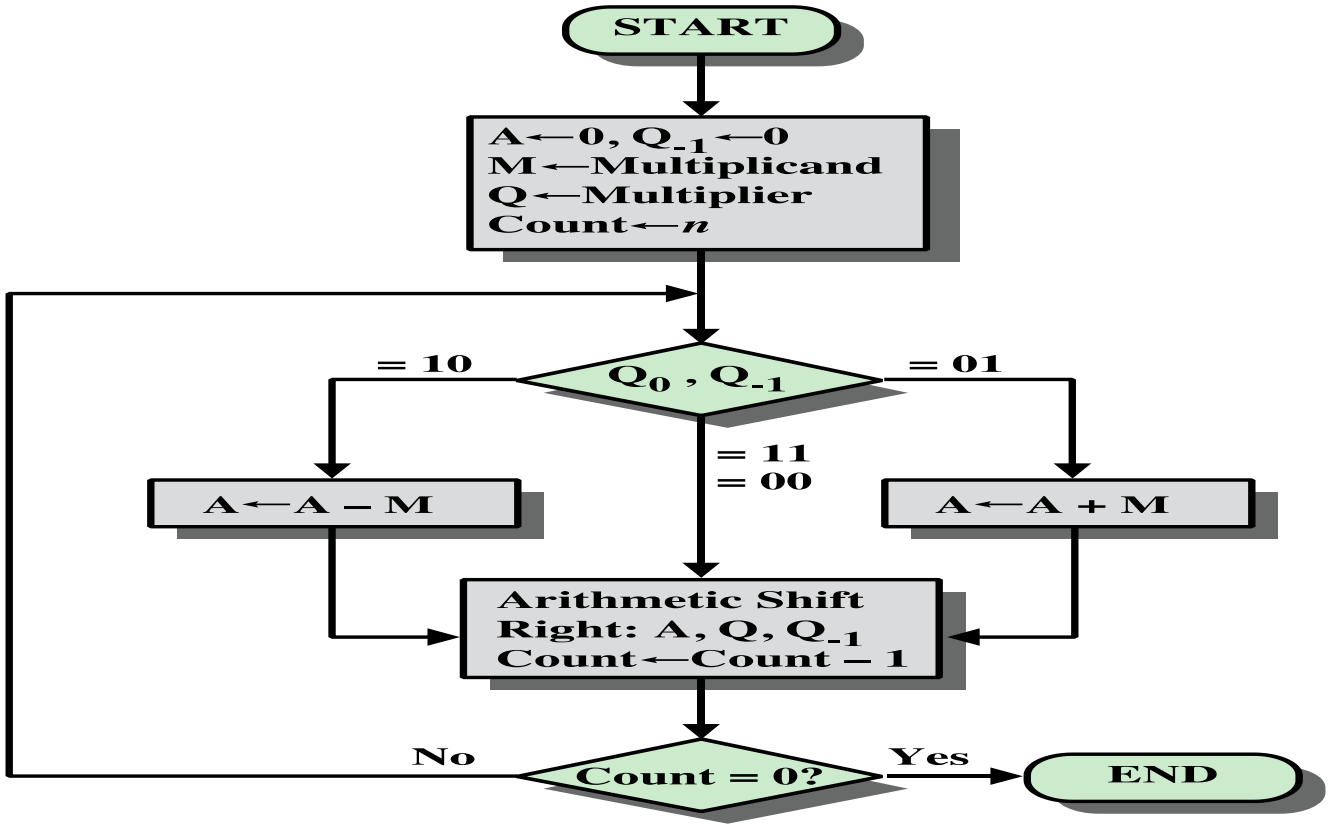


Figure **Booth's Algorithm for Twos Complement Multiplication**

Example of Booth's Algorithm

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

Figure 1 Example of Booth's Algorithm (7× 3)

CASE 2 – Negative Multiplier: $(5)_{10} X (-4)_{10}$

Multiplicand (B) = 0101(5) , Multiplier (Q) =1100(- 4)

Steps	A	Q	Q-1	Operation
	0000	1100	0	Initial
Step 1 :	0000	0110	0	Right Shift
Step 2 :	0000	0011	0	Right Shift
Step 3 :	1011	0011	0	A ← A - B
	1101	1001	1	Right Shift
Step 4:	1110	1100	1	Right Shift
	1110	**1100**		

11101100 = -20 (2's complement of 20)

Examples Using Booth's Algorithm

```
    0111
  ×0011     (0)
11111001    1–0
0000000     1–1
000111      0–1
00010101    (21)
```

(a) (7) × (3) = (21)

```
    0111
  ×1101     (0)
11111001    1–0
0000111     0–1
111001      1–0
11101011    (–21)
```

(b) (7) × (–3) = (–21)

```
    1001
  ×0011     (0)
00000111    1–0
0000000     1–1
111001      0–1
11101011    (–21)
```

(c) (–7) × (3) = (–21)

```
    1001
  ×1101     (0)
00000111    1–0
1111001     0–1
000111      1–0
00010101    (21)
```

(d) (–7) × (–3) = (21)

Figure Examples Using Booth's Algorithm

ARRAY MULTIPLIER

- An array multiplier is a digital combinational circuit that is used for the multiplication of two binary numbers by employing an array of full adders and half adders.
- Array multiplier is well known due to its regular structure.

A 2-Bit by 2-Bit Binary Multiplier

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$

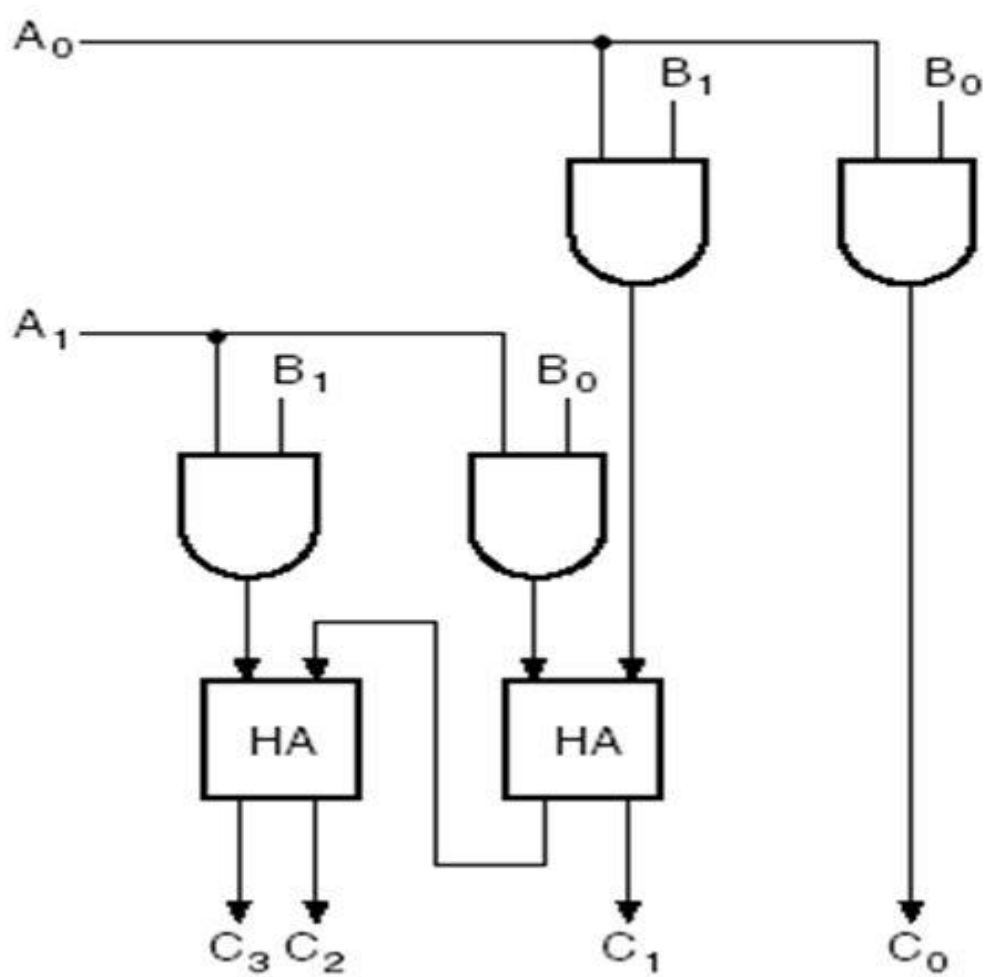


Fig. A 2-Bit by 2-Bit Binary Multiplier

Array Multiplier

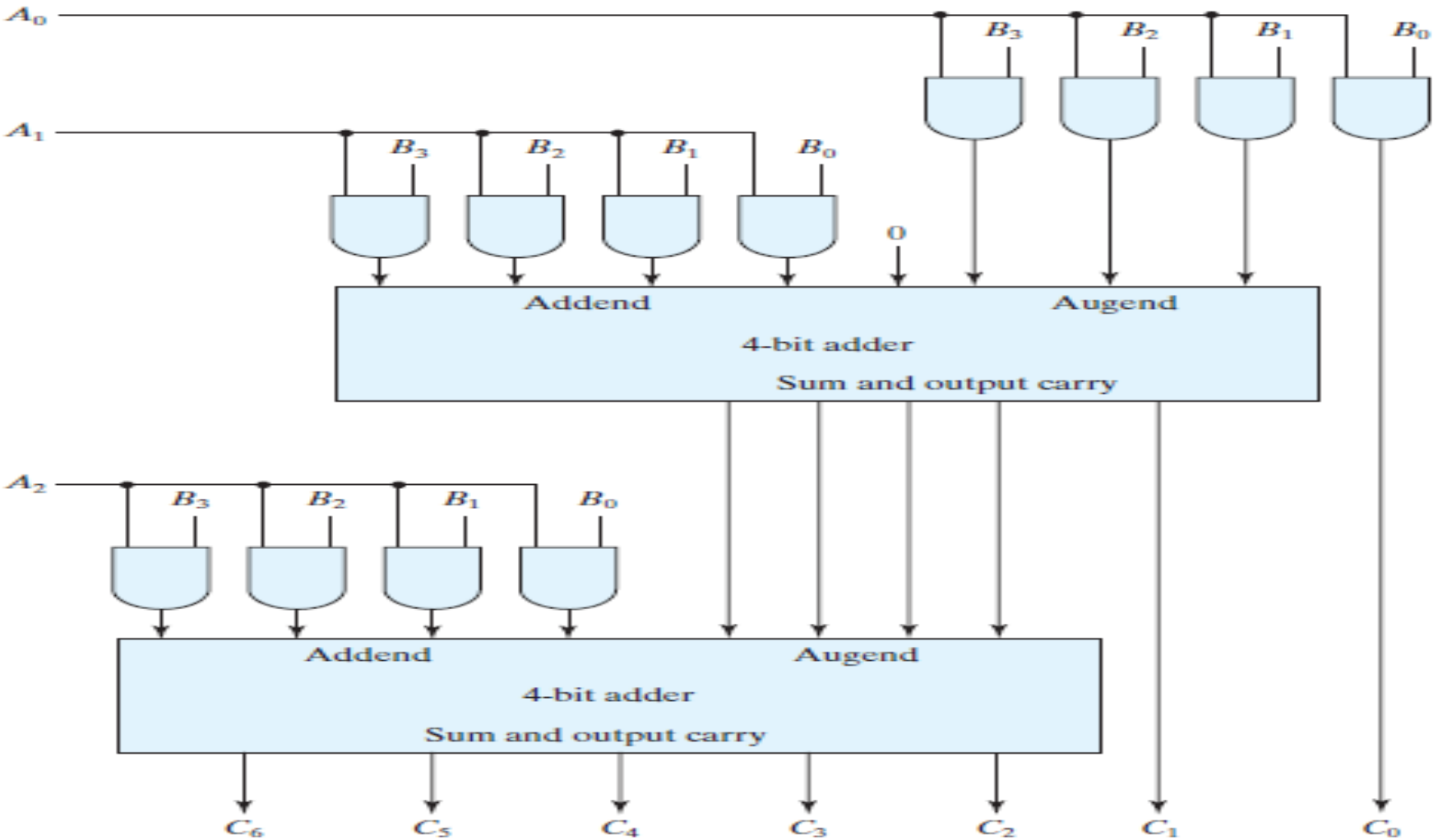
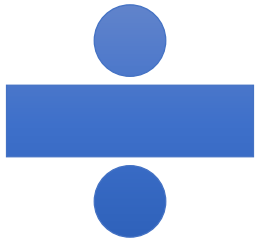


FIGURE
Four-bit by three-bit binary multiplier



Division

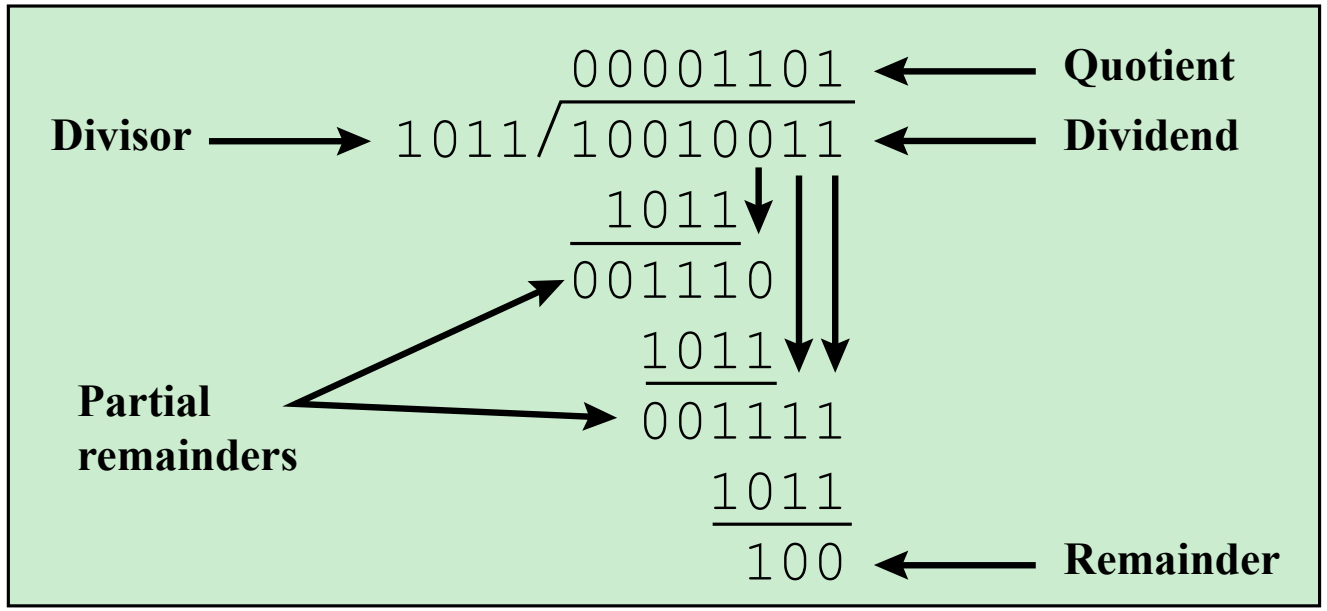


Figure Example of Division of Unsigned Binary Integers

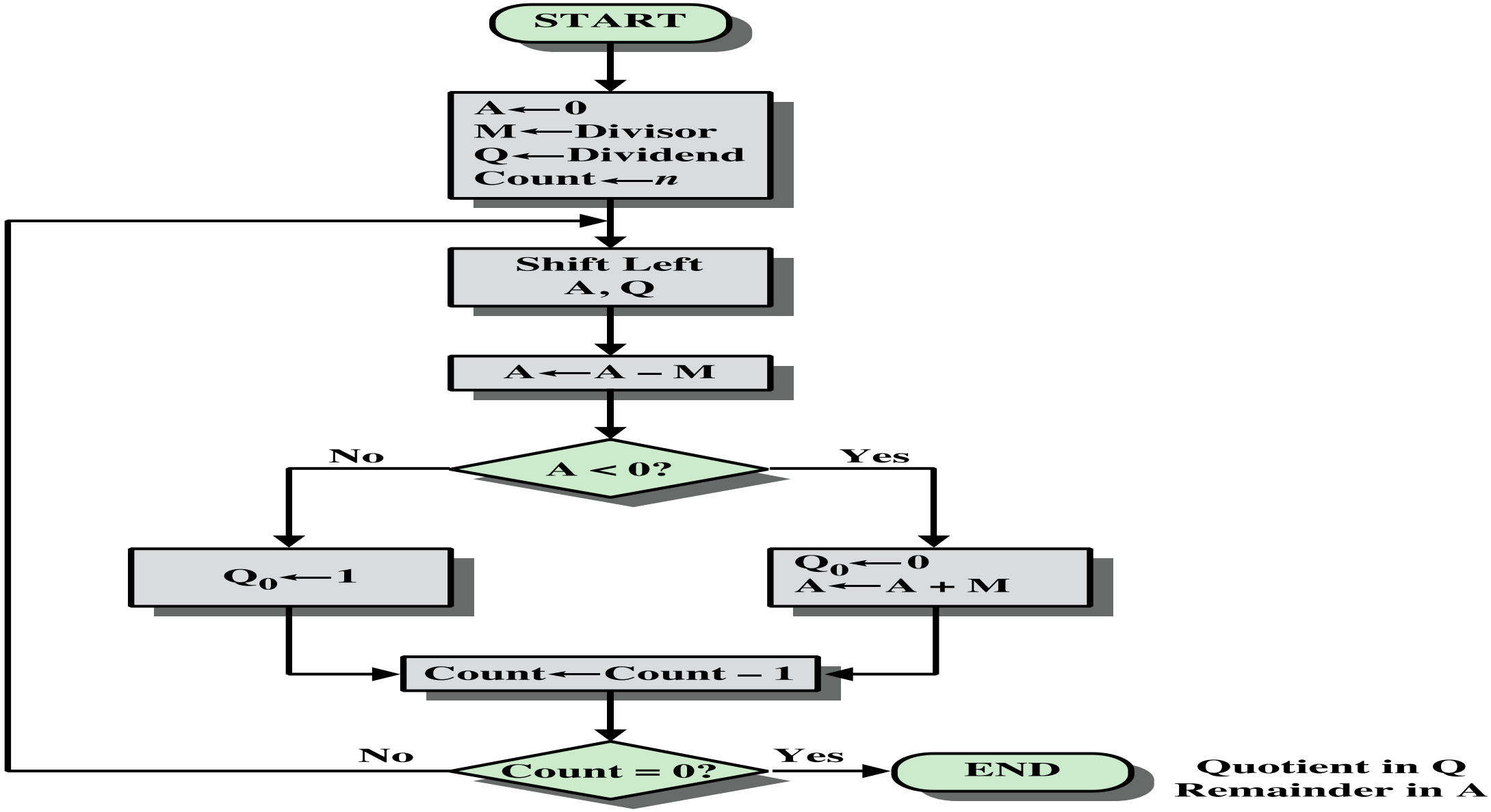


Figure **Flowchart for Unsigned Binary Division**

A	Q	
0000	0111	Initial value
0000 1101 1101 0000	1110 1110	Shift Use twos complement of 0011 for subtraction Subtract Restore, set $Q_0 = 0$
0001 1101 1110 0001	1100 1100	Shift Subtract Restore, set $Q_0 = 0$
0011 1101 0000	1000 1001	Shift Subtract, set $Q_0 = 1$
0001 1101 1110 0001	0010 0010	Shift Subtract Restore, set $Q_0 = 0$

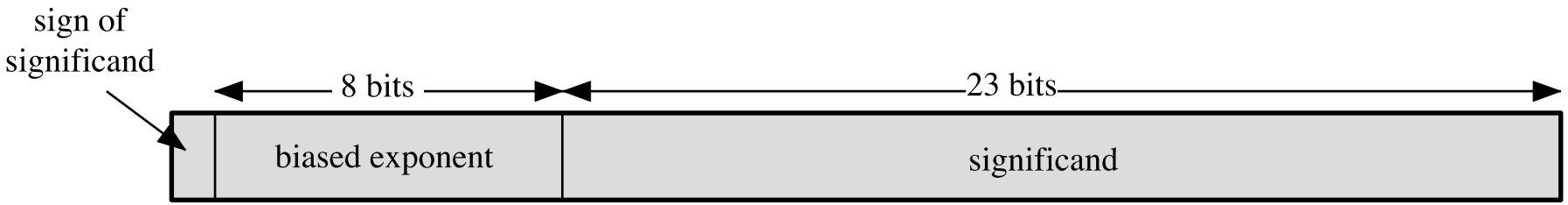
Figure Example of Restoring Twos Complement Division (7/3)

Floating-Point Representation

Principles

- With a fixed-point notation it is possible to represent a range of positive and negative integers centered on or near 0
- By assuming a fixed binary or radix point, this format allows the representation of numbers with a fractional component as well
- Limitations:
  - Very large numbers cannot be represented nor can very small fractions
  - The fractional part of the quotient in a division of two large numbers could be lost

Typical 32-Bit Floating-Point Format



(a) Format

$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.6328125 \times 2^{20}$

$-1.1010001 \times 2^{10100} = 1\ 10010011\ 10100010000000000000000 = -1.6328125 \times 2^{20}$

$1.1010001 \times 2^{-10100} = 0\ 01101011\ 10100010000000000000000 = 1.6328125 \times 2^{-20}$

$-1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010000000000000000 = -1.6328125 \times 2^{-20}$

(b) Examples

Figure Typical 32-Bit Floating-Point Format

Floating-Point

Significand

- The final portion of the word
- Any floating-point number can be expressed in many ways

The following are equivalent, where the significand is expressed in binary form:

$0.110 * 2^5$

$110 * 2^2$

$0.0110 * 2^6$

- *Normal number*
  - The most significant digit of the significand is nonzero

IEEE 754 Formats

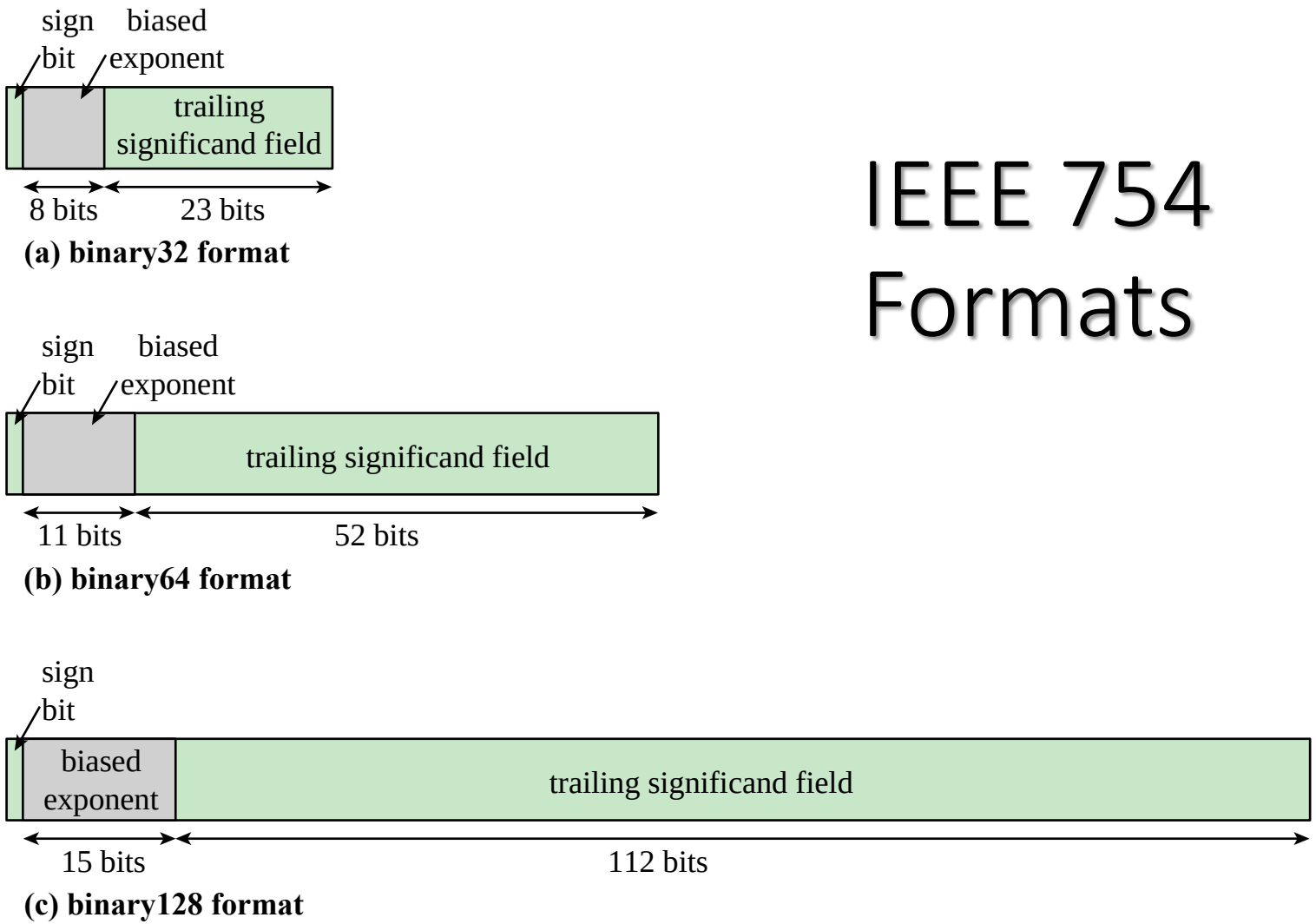


Figure IEEE 754 Formats

Table 10.6 Floating-Point Numbers and Arithmetic Operations

Floating Point Numbers	Arithmetic Operations
$X = X_S \times B^{X_E}$ $Y = Y_S \times B^{Y_E}$	$X + Y = \left(X_S \times B^{X_E - Y_E} + Y_S\right) \times B^{Y_E}$ $X - Y = \left(X_S \times B^{X_E - Y_E} - Y_S\right) \times B^{Y_E}$ $X_E \le Y_E$ $X \times Y = \left(X_S \times Y_S\right) \times B^{X_E + Y_E}$ $\frac{X}{Y} = \left(\frac{X_S}{Y_S}\right) \times B^{X_E - Y_E}$

Examples:

$X = 0.3 \times 10^2 = 30$
$Y = 0.2 \times 10^3 = 200$

$X + Y = (0.3 \times 10^{2-3} + 0.2) \times 10^3 = 0.23 \times 10^3 = 230$
$X - Y = (0.3 \times 10^{2-3} - 0.2) \times 10^3 = (-0.17) \times 10^3 = -170$
$X \times Y = (0.3 \times 0.2) \times 10^{2+3} = 0.06 \times 10^5 = 6000$
$X / Y = (0.3 / 0.2) \times 10^{2-3} = 1.5 \times 10^{-1} = 0.15$

Floating-Point Addition and Subtraction

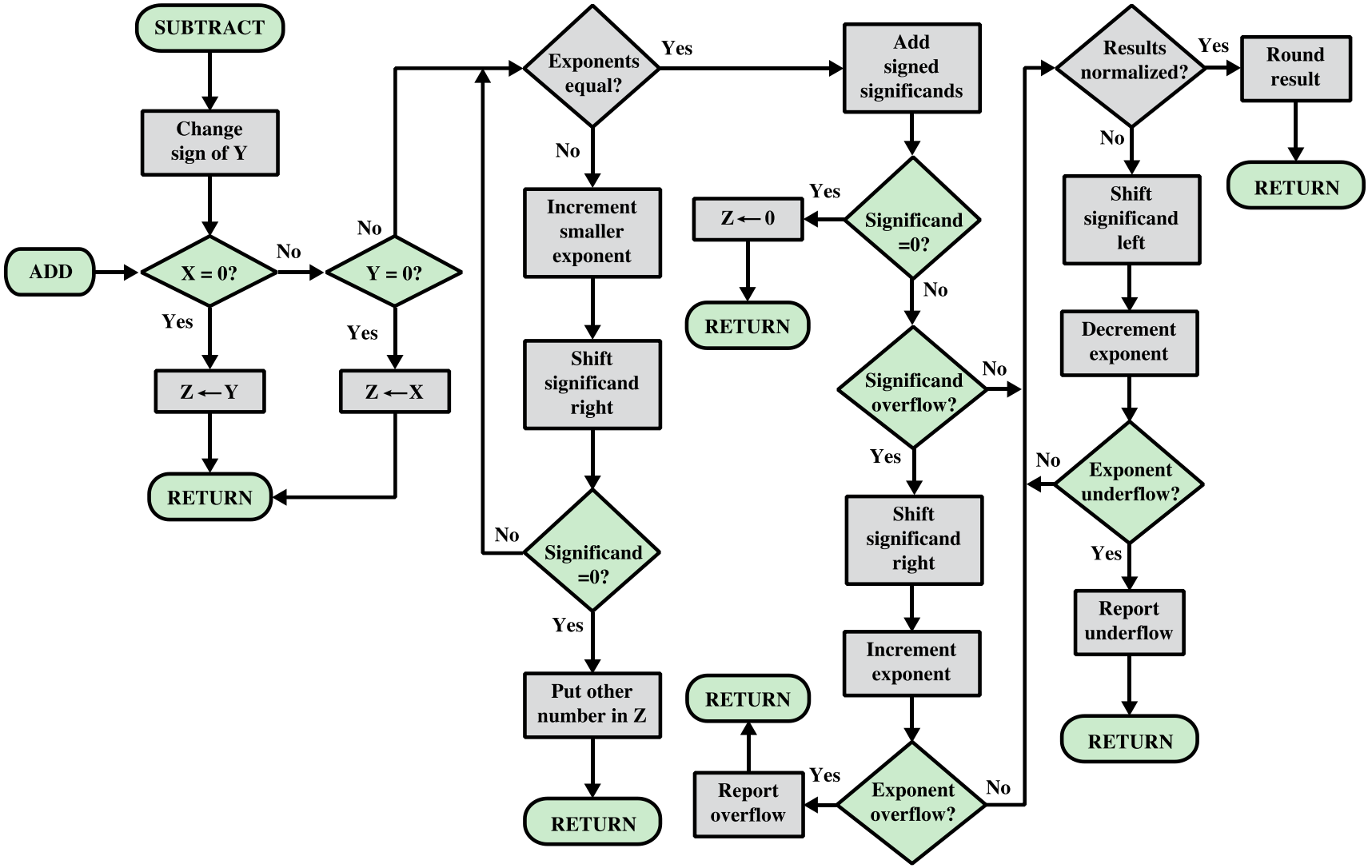


Figure **Floating-Point Addition and Subtraction (Z← X ± Y)**

Floating-Point Multiplication

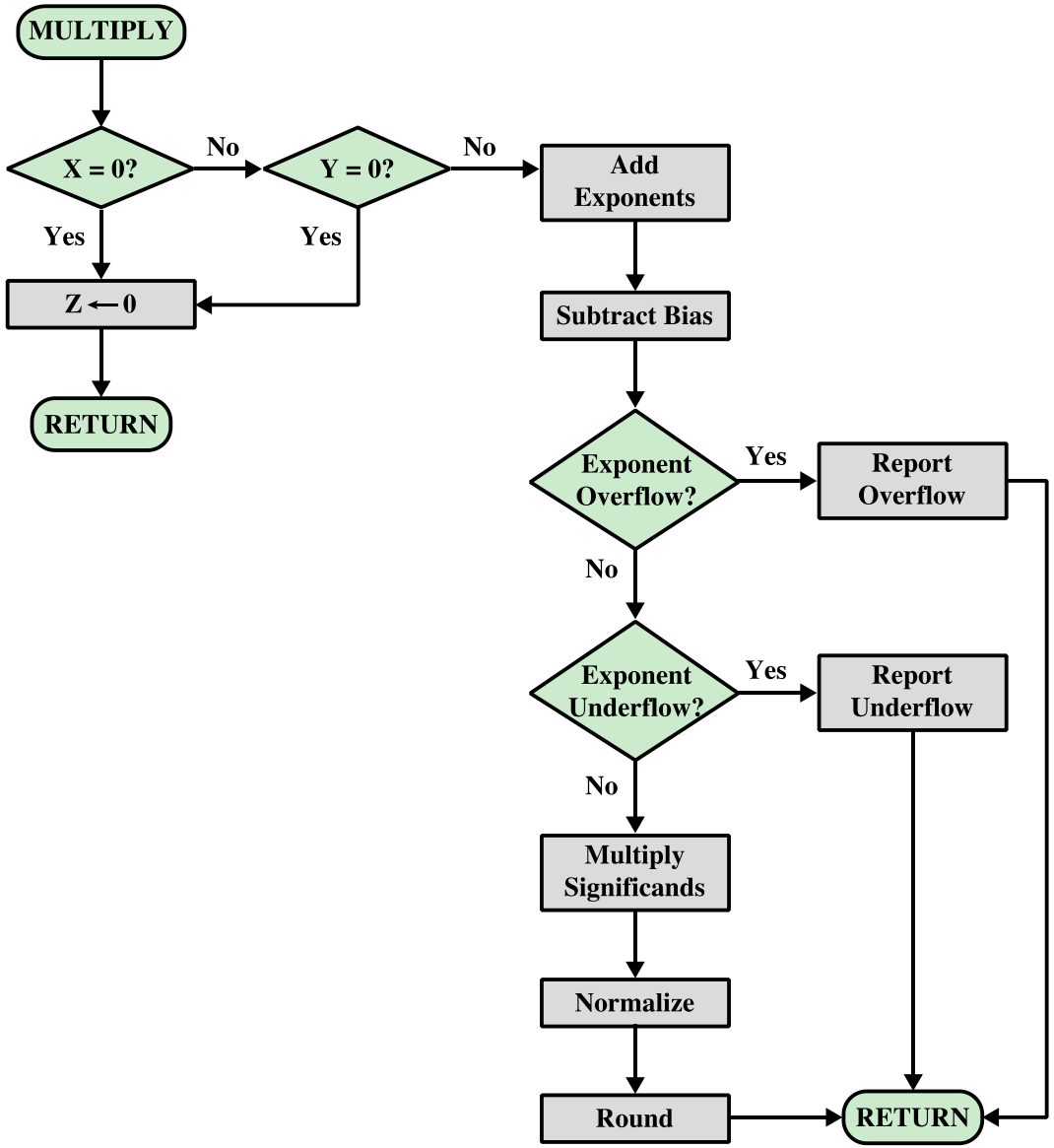


Figure **Floating-Point Multiplication (Z← X× Y)**

Floating-Point Division

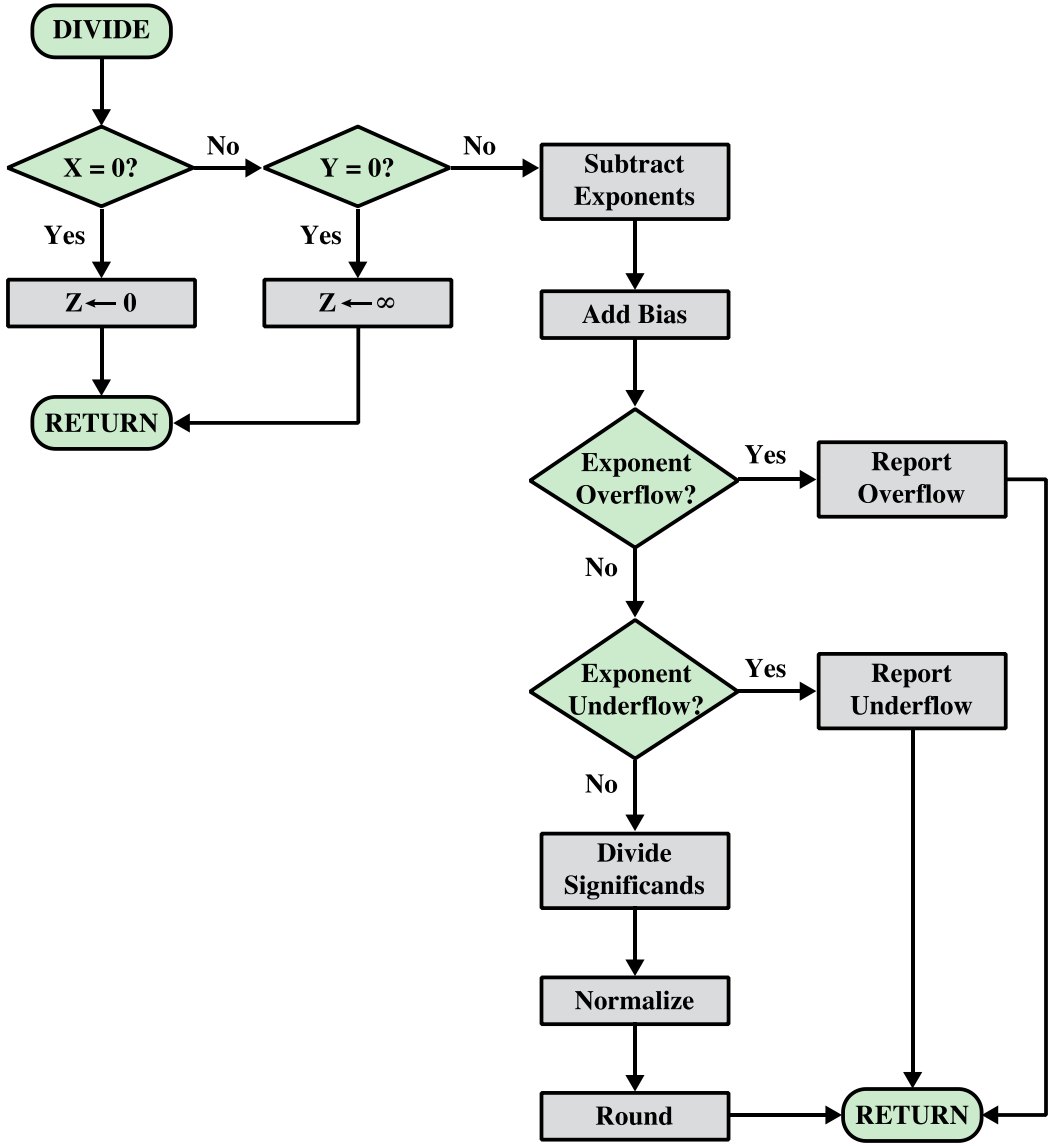


Figure **Floating-Point Division (Z← X/Y)**