\+

Integer Representation

- In the binary number system arbitrary numbers can be represented with:
  - The digits zero and one
  - The minus sign (for negative numbers)
  - The period, or ***radix point*** (for numbers with a fractional component)
- For purposes of computer storage and processing we do not have the benefit of special symbols for the minus sign and radix point
- Only binary digits (0,1) may be used to represent numbers

Sign-Magnitude Representation

There are several alternative conventions used to represent negative as well as positive integers	•All of these alternatives involve treating the most significant (leftmost) bit in the word as a sign bit •If the sign bit is 0 the number is positive •If the sign bit is 1 the number is negative
Sign-magnitude representation is the simplest form that employs a sign bit	
Drawbacks:	•Addition and subtraction require a consideration of both the signs of the numbers and their relative magnitudes to carry out the required operation •There are two representations of 0
Because of these drawbacks, sign-magnitude representation is rarely used in implementing the integer portion of the ALU	

+

Twos Complement Representation

- Uses the most significant bit as a sign bit
- Differs from sign-magnitude representation in the way that the other bits are interpreted

Range	-2_{n-1} through $2_{n-1} - 1$
Number of Representations of Zero	One
Negation	Take the Boolean complement of each bit of the corresponding positive number, then add 1 to the resulting bit pattern viewed as an unsigned integer.
Expansion of Bit Length	Add additional bit positions to the left and fill in with the value of the original sign bit.
Overflow Rule	If two numbers with the same sign (both positive or both negative) are added, then overflow occurs if and only if the result has the opposite sign.
Subtraction Rule	To subtract B from A, take the twos complement of B and add it to A.

Table 10.1 Characteristics of Twos Complement Representation and Arithmetic

Table 10.2
Alternative Representations for 4-Bit Integers

Decimal Representation	Sign-Magnitude Representation	Twos Complement Representation	Biased Representation
+8	—	—	1111
+7	0111	0111	1110
+6	0110	0110	1101
+5	0101	0101	1100
+4	0100	0100	1011
+3	0011	0011	1010
+2	0010	0010	1001
+1	0001	0001	1000
+0	0000	0000	0111
–0	1000	—	—
–1	1001	1111	0110
–2	1010	1110	0101
–3	1011	1101	0100
–4	1100	1100	0011
–5	1101	1011	0010
–6	1110	1010	0001
–7	1111	1001	0000
–8	—	1000	—

Range Extension

- Range of numbers that can be expressed is extended by increasing the bit length
- In sign-magnitude notation this is accomplished by moving the sign bit to the new leftmost position and fill in with zeros
- This procedure will not work for twos complement negative integers
  - Rule is to move the sign bit to the new leftmost position and fill in with copies of the sign bit
  - For positive numbers, fill in with zeros, and for negative numbers, fill in with ones
  - This is called *sign extension*

Fixed-Point Representation

The radix point (binary point) is fixed and assumed to be to the right of the rightmost digit

Programmer can use the same representation for binary fractions by scaling the numbers so that the binary point is implicitly positioned at some other location

Negation

- Twos complement operation
  - Take the Boolean complement of each bit of the integer (including the sign bit)
  - Treating the result as an unsigned binary integer, add 1

+18 = 00010010 (twos complement)
bitwise complement = 11101101
+ 1
11101110 = -18

- The negative of the negative of that number is itself:

-18 = 11101110 (twos complement)
bitwise complement = 00010001
+ 1
00010010 = +18

Negation Special Case 1

0 = 00000000 (twos complement)

Bitwise complement = 11111111

Add 1 to LSB + 1

Result 100000000

Overflow is ignored, so:

- 0 = 0

Negation Special Case 2

-128 = 10000000 (twos complement)

Bitwise complement = 01111111

Add 1 to LSB + 1

Result 10000000

So:

-(-128) = -128 X

Monitor MSB (sign bit)

It should change during negation

Addition

1001 = –7 +0101 = 5 1110 = –2	1100 = –4 +0100 = 4 10000 = 0
(a) (–7) + (+5)	(b) (–4) + (+4)
0011 = 3 +0100 = 4 0111 = 7	1100 = –4 +1111 = –1 11011 = –5
(c) (+3) + (+4)	(d) (–4) + (–1)
0101 = 5 +0100 = 4 1001 = Overflow	1001 = –7 +1010 = –6 10011 = Overflow
(e) (+5) + (+4)	(f) (–7) + (–6)

Figure 10.3 Addition of Numbers in Twos Complement Representation

OVERFLOW RULE:

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

Overflow

Rule

Subtraction

Rule

SUBTRACTION RULE:

To subtract one number (subtrahend) from another (minuend), take the twos complement (negation) of the subtrahend and add it to the minuend.

Subtraction

0010 = 2 +1001 = −7 1011 = −5	0101 = 5 +1110 = −2 10011 = 3
(a) M = 2 = 0010 S = 7 = 0111 −S = 1001	(b) M = 5 = 0101 S = 2 = 0010 −S = 1110
1011 = −5 +1110 = −2 11001 = −7	0101 = 5 +0010 = 2 0111 = 7
(c) M = −5 = 1011 S = 2 = 0010 −S = 1110	(d) M = 5 = 0101 S = −2 = 1110 −S = 0010
0111 = 7 +0111 = 7 1110 = Overflow	1010 = −6 +1100 = −4 10110 = Overflow
(e) M = 7 = 0111 S = −7 = 1001 −S = 0111	(f) M = −6 = 1010 S = 4 = 0100 −S = 1100

Figure 10.4 Subtraction of Numbers in Twos Complement Representation (M – S)

Geometric Depiction of Twos Complement Integers

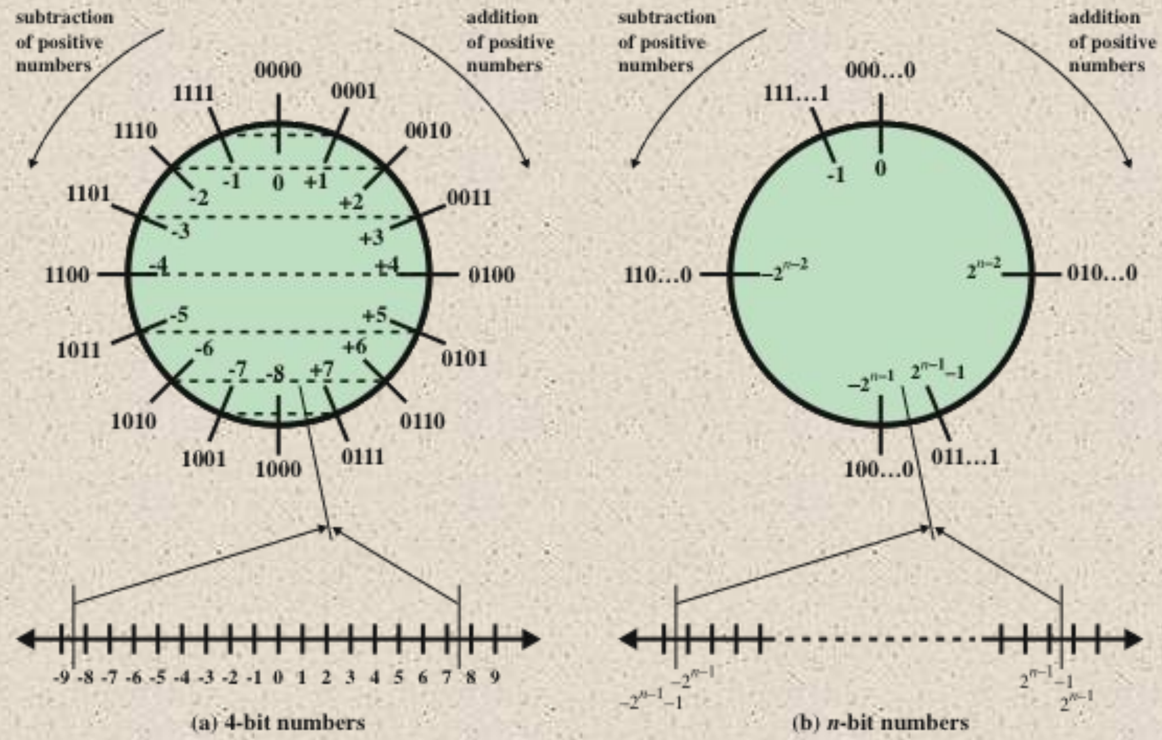


Figure 10.5 Geometric Depiction of Twos Complement Integers

+

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 10.18 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

Expressible Numbers

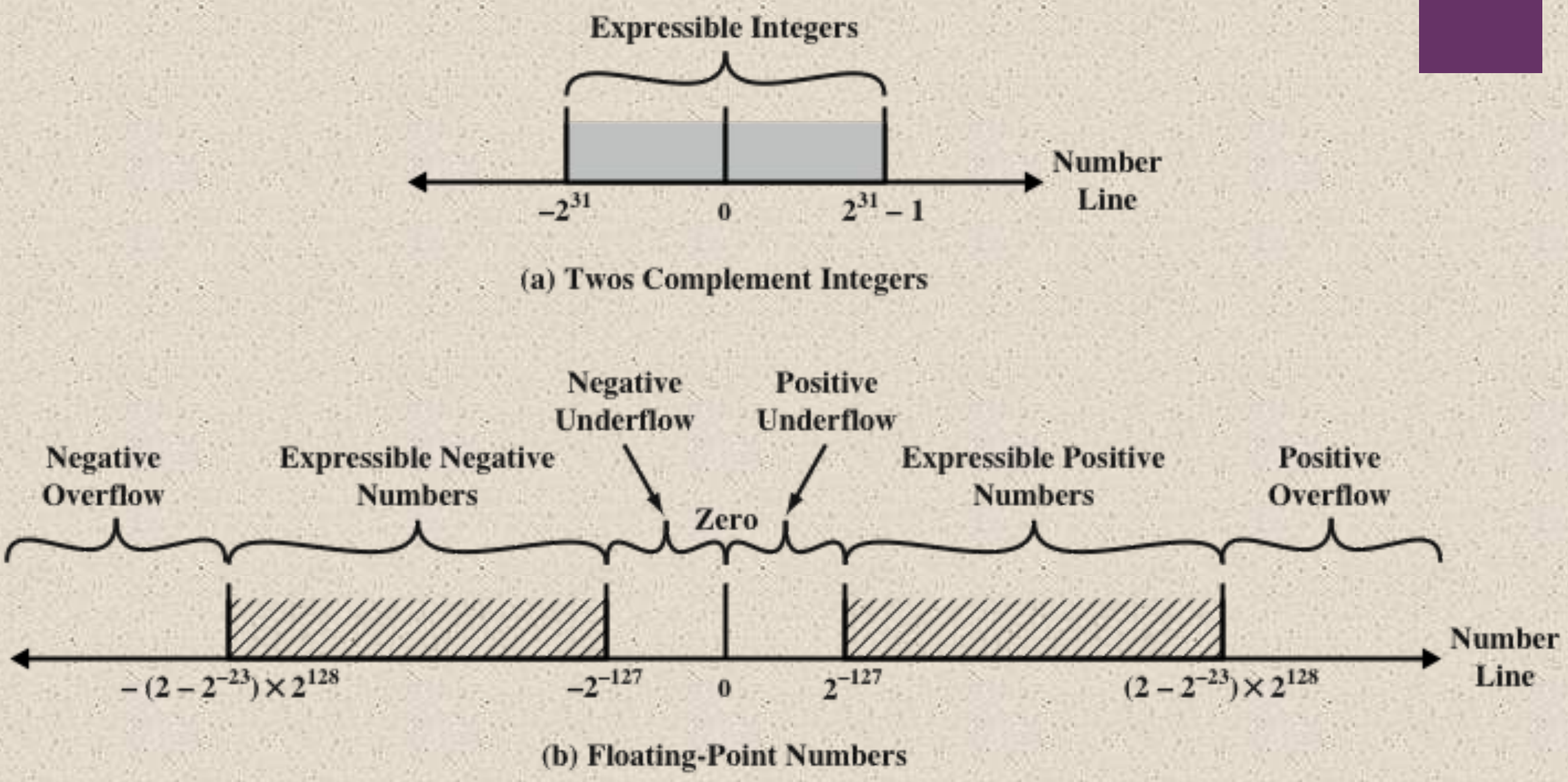


Figure 10.19 Expressible Numbers in Typical 32-Bit Formats

+

Density of Floating-Point Numbers

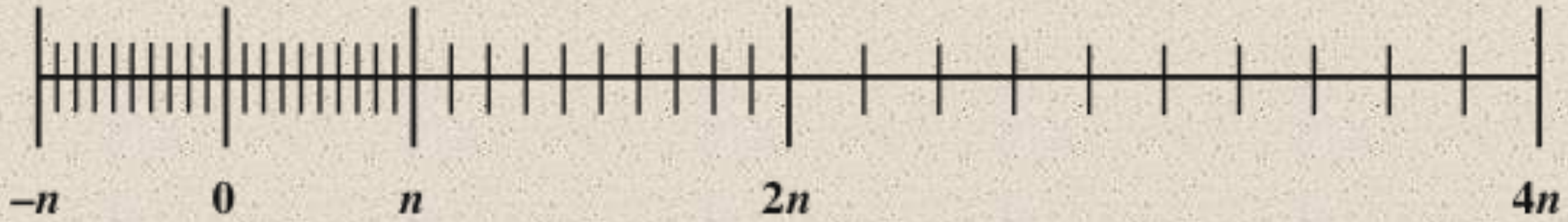


Figure 10.20 Density of Floating-Point Numbers

IEEE Standard 754

Most important floating-point representation is defined

Standard was developed to facilitate the portability of programs from one processor to another and to encourage the development of sophisticated, numerically oriented programs

Standard has been widely adopted and is used on virtually all contemporary processors and arithmetic coprocessors

IEEE 754-2008 covers both binary and decimal floating-point representations

IEEE 754-2008

- Defines the following different types of floating-point formats:

- Arithmetic format
  - All the mandatory operations defined by the standard are supported by the format. The format may be used to represent floating-point operands or results for the operations described in the standard.

- Basic format
  - This format covers five floating-point representations, three binary and two decimal, whose encodings are specified by the standard, and which can be used for arithmetic. At least one of the basic formats is implemented in any conforming implementation.

- Interchange format
  - A fully specified, fixed-length binary encoding that allows data interchange between different platforms and that can be used for storage.

IEEE 754 Formats

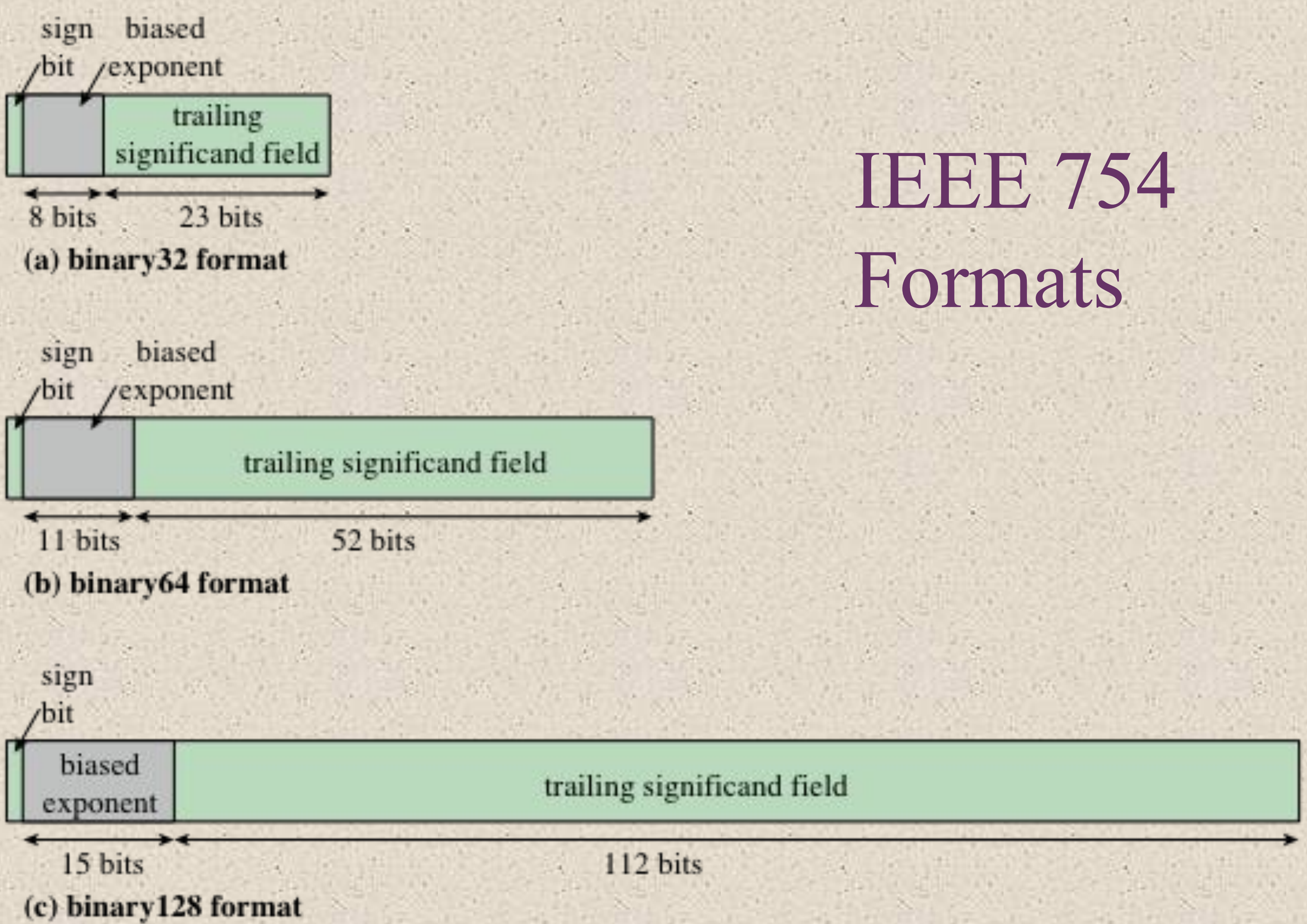


Figure 10.21 IEEE 754 Formats

Parameter	Format		
	binary32	binary64	binary128
Storage width (bits)	32	64	128
Exponent width (bits)	8	11	15
Exponent bias	127	1023	16383
Maximum exponent	127	1023	16383
Minimum exponent	–126	–1022	–16382
Approx normal number range (base 10)	$10_{-38}, 10_{+38}$	$10_{-308}, 10_{+308}$	$10_{-4932}, 10_{+4932}$
Trailing significand width (bits)*	23	52	112
Number of exponents	254	2046	32766
Number of fractions	2_{23}	2_{52}	2_{112}
Number of values	$1.98 \times 2_{31}$	$1.99 \times 2_{63}$	$1.99 \times 2_{128}$
Smallest positive normal number	2_{-126}	2_{-1022}	2_{-16362}
Largest positive normal number	$2_{128} - 2_{104}$	$2_{1024} - 2_{971}$	$2_{16384} - 2_{16271}$
Smallest subnormal magnitude	2_{-149}	2_{-1074}	2_{-16494}

* not including implied bit and not including sign bit

Table 10.3

IEEE 754

Format Parameters

Additional Formats

Extended Precision Formats

- Provide additional bits in the exponent (extended range) and in the significand (extended precision)
- Lessens the chance of a final result that has been contaminated by excessive roundoff error
- Lessens the chance of an intermediate overflow aborting a computation whose final result would have been representable in a basic format
- Affords some of the benefits of a larger basic format without incurring the time penalty usually associated with higher precision

Extendable Precision Format

- Precision and range are defined under user control
- May be used for intermediate calculations but the standard places no constraint or format or length

nterpretation of IEEE 754 Floating-Point Numbers

(a) binary 32 format

	Sign	Biased exponent	Fraction	Value
positive zero	0	0	0	0
negative zero	1	0	0	–0
plus infinity	0	all 1s	0	∞
Minus infinity	1	all 1s	0	–∞
quiet NaN	0 or 1	all 1s	≠ 0; first bit = 1	qNaN
signaling NaN	0 or 1	all 1s	≠ 0; first bit = 0	sNaN
positive normal nonzero	0	$0 < e < 255$	f	$2_{e-127}(1.f)$
negative normal nonzero	1	$0 < e < 255$	f	$-2_{e-127}(1.f)$
positive subnormal	0	0	$f \neq 0$	$2_{e-126}(0.f)$
negative subnormal	1	0	$f \neq 0$	$-2_{e-126}(0.f)$

Table 10.5 Interpretation of IEEE 754 Floating-Point Numbers (page 1 of 3)

nterpretation of IEEE 754 Floating-Point Numbers

(b) binary 64 format

	Sign	Biased exponent	Fraction	Value
positive zero	0	0	0	0
negative zero	1	0	0	–0
plus infinity	0	all 1s	0	∞
Minus infinity	1	all 1s	0	$-\infty$
quiet NaN	0 or 1	all 1s	$\neq 0$; first bit = 1	qNaN
signaling NaN	0 or 1	all 1s	$\neq 0$; first bit = 0	sNaN
positive normal nonzero	0	$0 < e < 2047$	f	$2_{e-1023}(1.f)$
negative normal nonzero	1	$0 < e < 2047$	f	$-2_{e-1023}(1.f)$
positive subnormal	0	0	$f \neq 0$	$2_{e-1022}(0.f)$
negative subnormal	1	0	$f \neq 0$	$-2_{e-1022}(0.f)$

Table 10.5 Interpretation of IEEE 754 Floating-Point Numbers (page 2 of 3)

nterpretation of IEEE 754 Floating-Point Numbers

(c) binary 128 format

	Sign	Biased exponent	Fraction	Value
positive zero	0	0	0	0
negative zero	1	0	0	−0
plus infinity	0	all 1s	0	∞
minus infinity	1	all 1s	0	$-\infty$
quiet NaN	0 or 1	all 1s	$\neq 0$; first bit = 1	qNaN
signaling NaN	0 or 1	all 1s	$\neq 0$; first bit = 0	sNaN
positive normal nonzero	0	all 1s	f	$2_{e-16383}(1.f)$
negative normal nonzero	1	all 1s	f	$-2_{e-16383}(1.f)$
positive subnormal	0	0	$f \neq 0$	$2_{e-16383}(0.f)$
negative subnormal	1	0	$f \neq 0$	$-2_{e-16383}(0.f)$

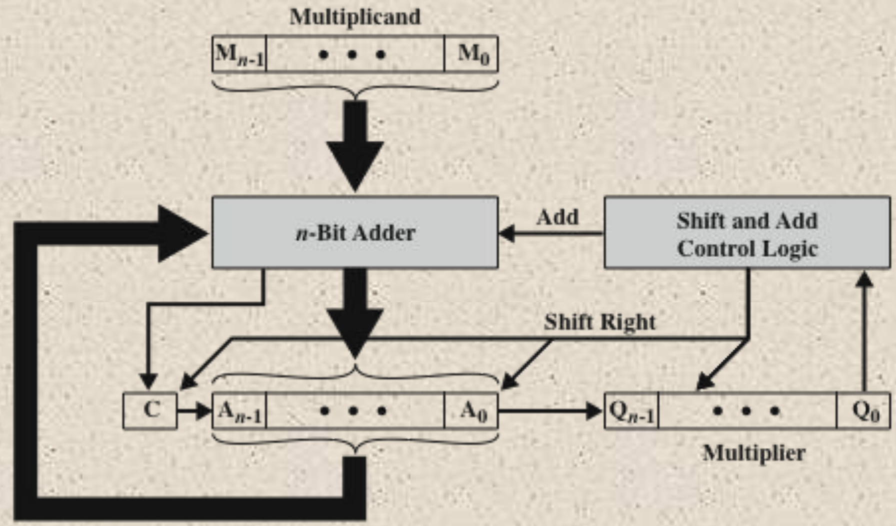
Table 10.5 Interpretation of IEEE 754 Floating-Point Numbers (page 3 of 3)

Multiplication

1011	Multiplicand (11)
×1101	Multiplier (13)
1011	Partial products
0000	
1011	
1011	
10001111	Product (143)

Figure 10.7 Multiplication of Unsigned Binary Integers

Hardware Implementation of Unsigned Binary Multiplication



(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 10.8 Hardware Implementation of Unsigned Binary Multiplication

Flowchart for Unsigned Binary Multiplication

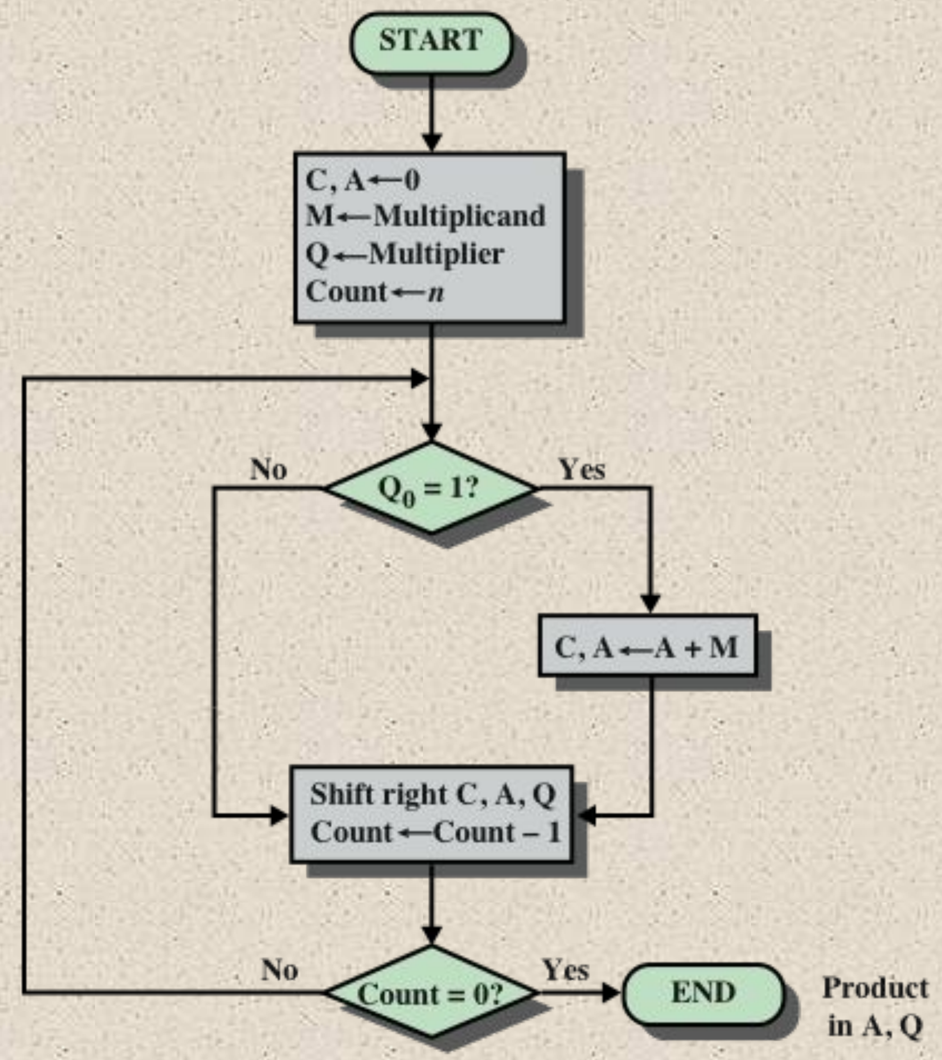


Figure 10.9 Flowchart for Unsigned Binary Multiplication

+

Twos Complement Multiplication

1011	
×1101	
00001011	$1011 \times 1 \times 2^0$
00000000	$1011 \times 0 \times 2^1$
00101100	$1011 \times 1 \times 2^2$
01011000	$1011 \times 1 \times 2^3$
10001111	

Figure 10.10 Multiplication of Two Unsigned 4-Bit Integers Yielding an 8-Bit Result

Comparison

(a) Unsigned integers		(b) Twos complement integers	
1001	(9)	1001	(−7)
×0011	(3)	×0011	(3)
00001001	1001×2^0	11111001	$(-7) \times 2^0 = (-7)$
00010010	1001×2^1	11110010	$(-7) \times 2^1 = (-14)$
00011011	(27)	11101011	(−21)

Figure 10.11 Comparison of Multiplication of Unsigned and Twos Complement Integers

Booth's Algorithm

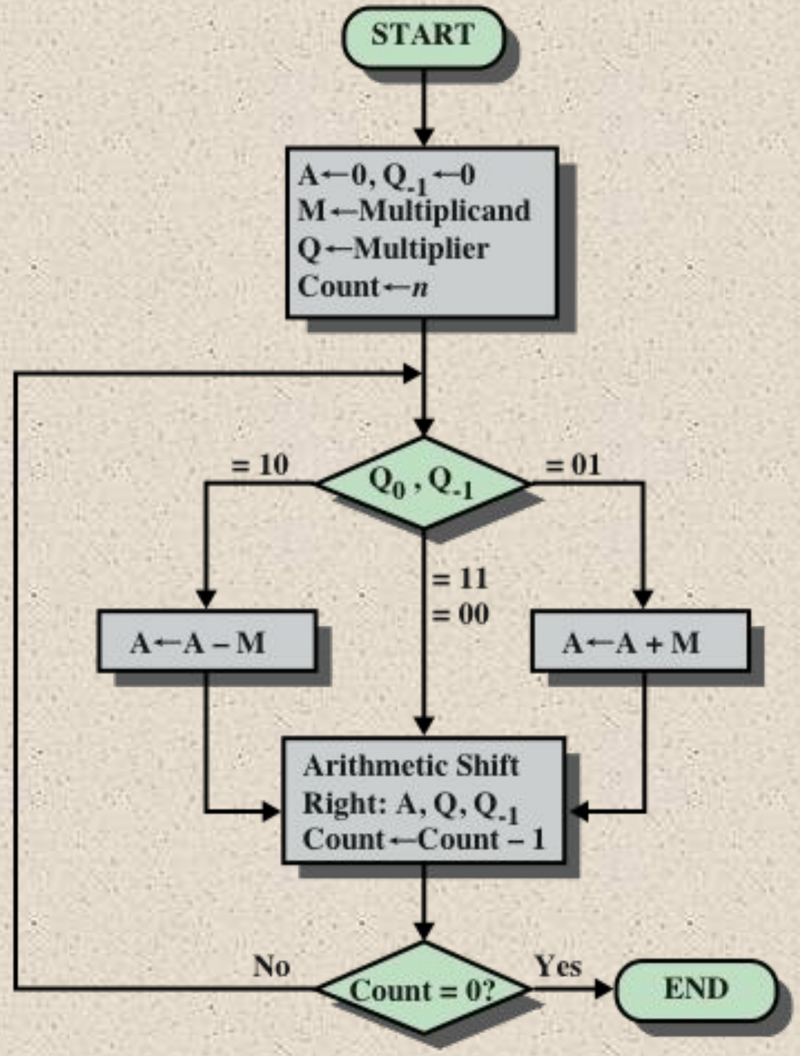


Figure 10.12 Booth's Algorithm for Twos Complement Multiplication

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 10.13 Example of Booth's Algorithm (7× 3)

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 10.14 Examples Using Booth's Algorithm

Division

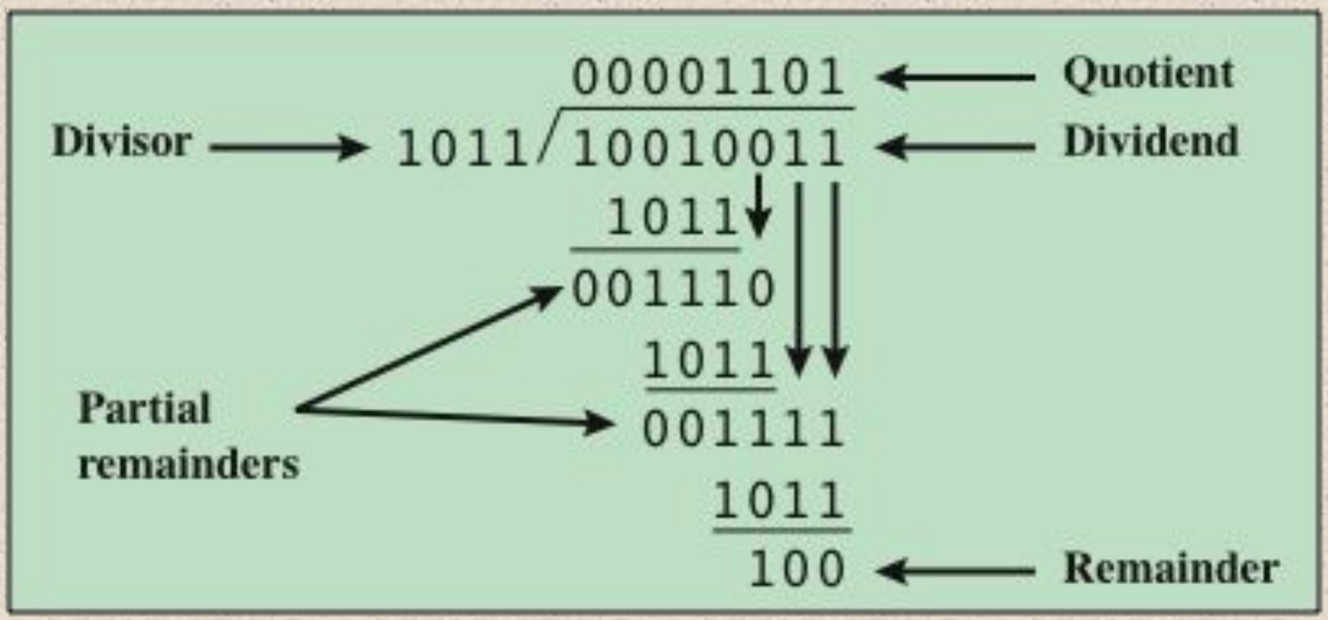


Figure 10.15 Example of Division of Unsigned Binary Integers

+

Flowchart for Unsigned Binary Division

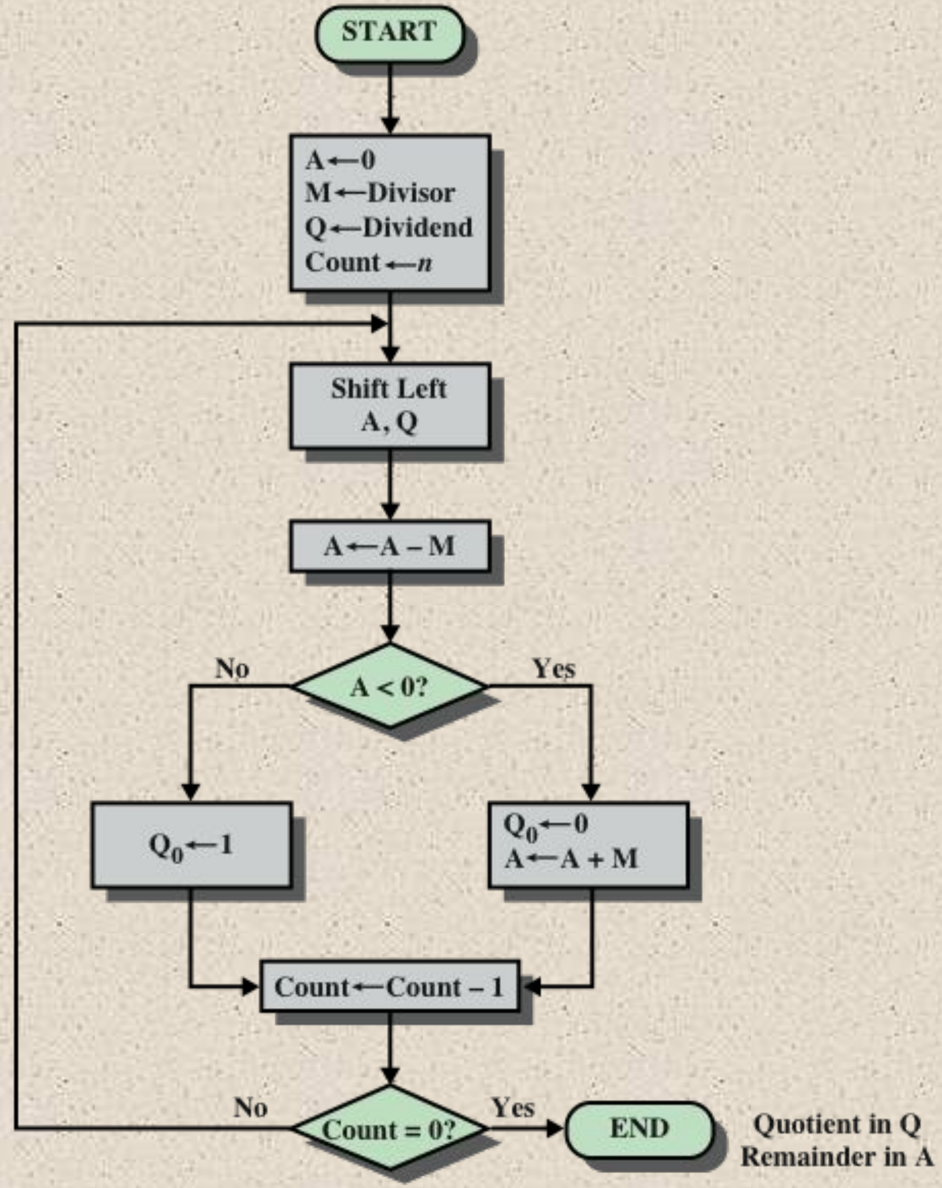


Figure 10.16 Flowchart for Unsigned Binary Division

Example of Restoring Twos Complement Division

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

Figure 10.17 Example of Restoring Twos Complement Division (7/3)

+

Flow chart of non restoring division algorithm

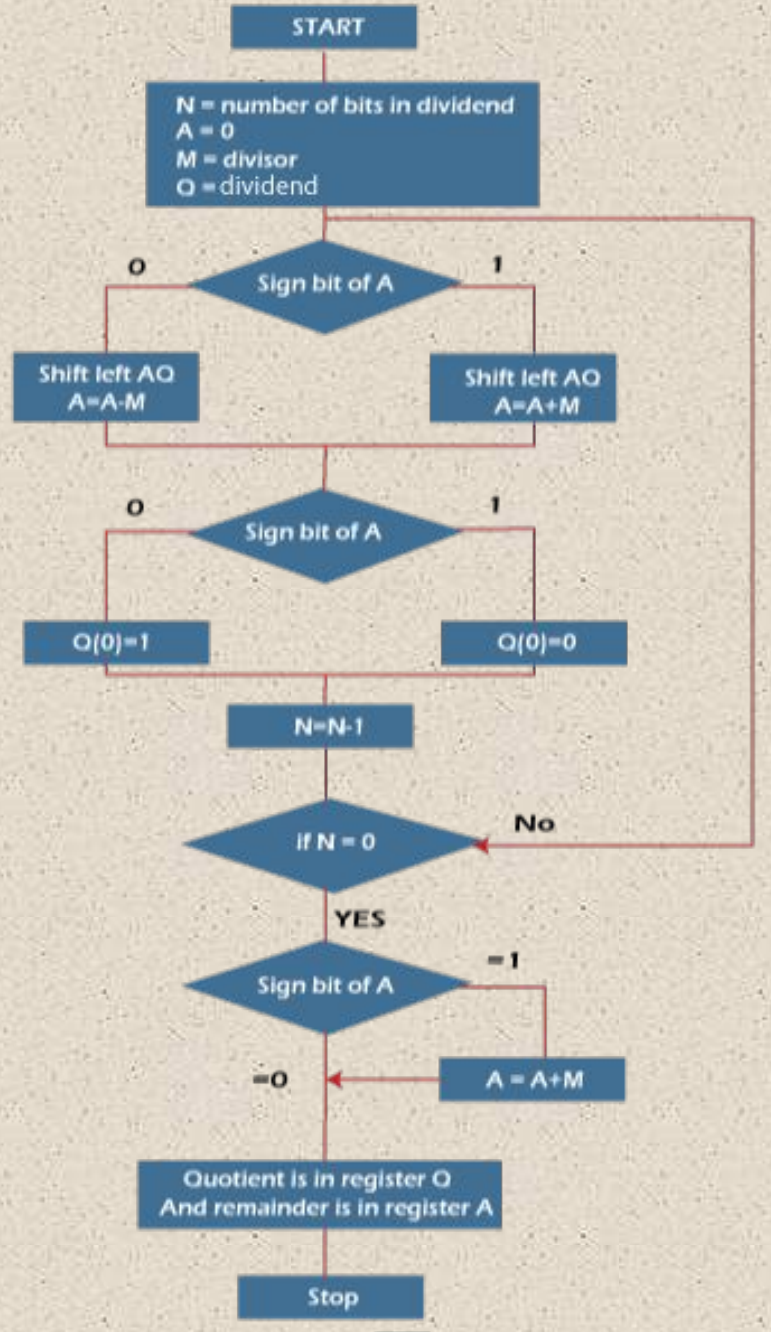


Table 10.6 Floating-Point Numbers and Arithmetic Operations

Floating Point Numbers	**Arithmetic Operations**
$X = X_S \times B^{X_E}$ $Y = Y_S \times B^{Y_E}$	$\left.\begin{array}{l} X + Y = \left(X_s \times B^{X_E - Y_E} + Y_s\right) \times B^{Y_E} \\ X - Y = \left(X_s \times B^{X_E - Y_E} - Y_s\right) \times B^{Y_E} \end{array}\right\} X_E \le Y_E$ $X \times Y = \left(X_s \times Y_s\right) \times B^{X_E + Y_E}$ $\dfrac{X}{Y} = \left(\dfrac{X_s}{Y_s}\right) \times B^{X_E - Y_E}$

Examples:

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

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

Floating-Point Addition and Subtraction

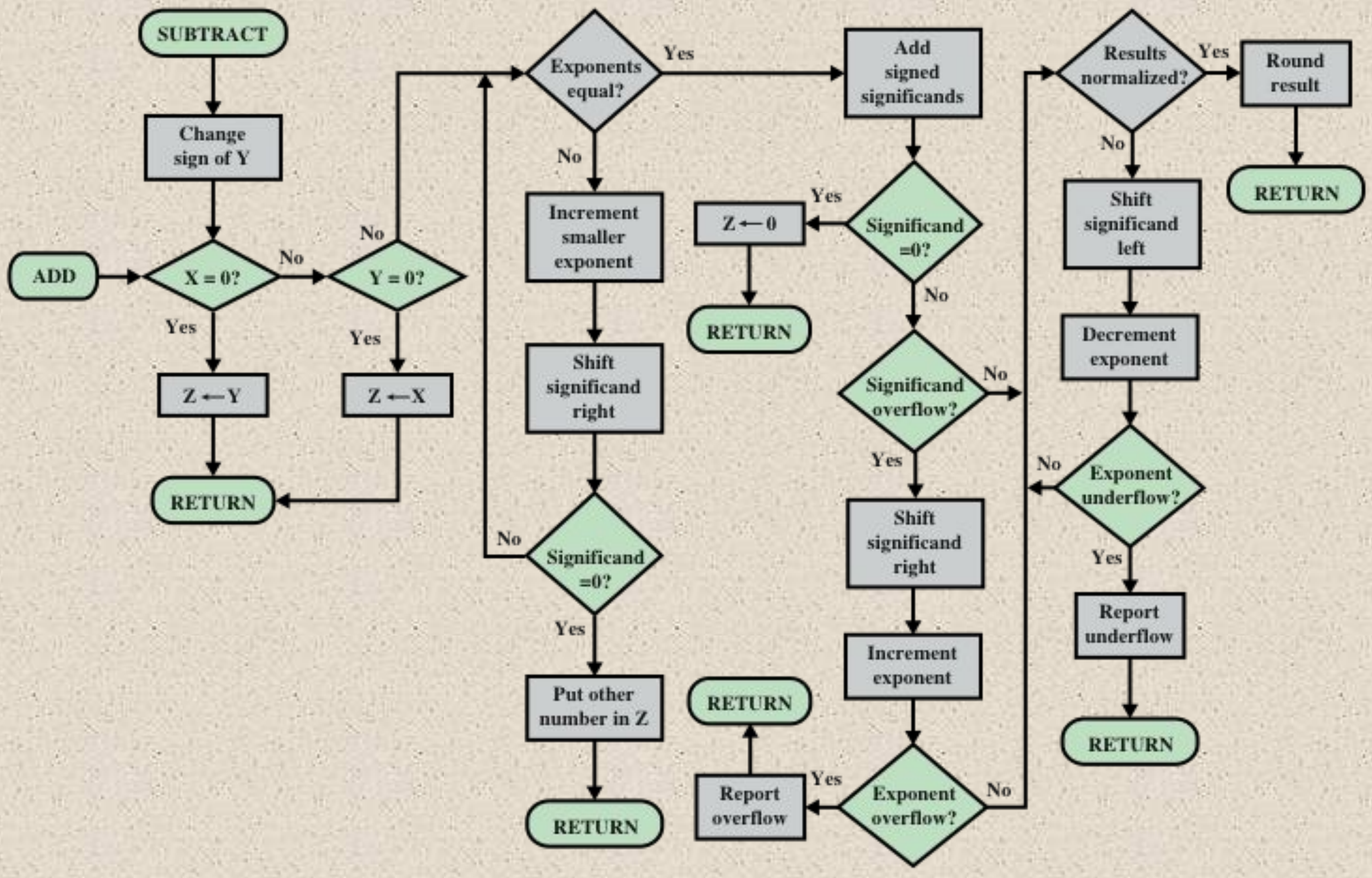


Figure 10.22 Floating-Point Addition and Subtraction ($Z \leftarrow X \pm Y$)

Floating-Point Multiplication

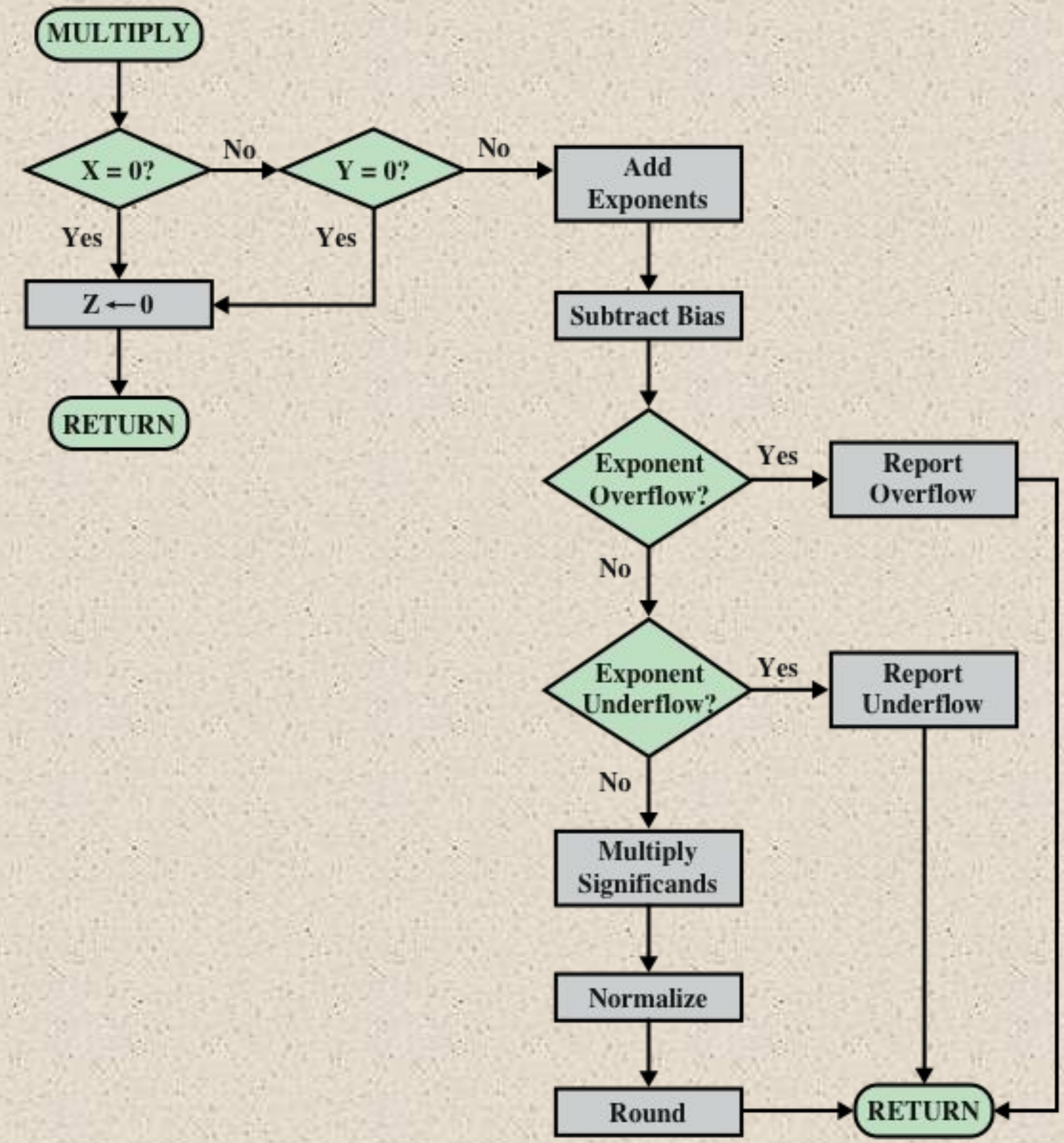


Figure 10.23 Floating-Point Multiplication (Z← X× Y)

+

Floating-Point Division

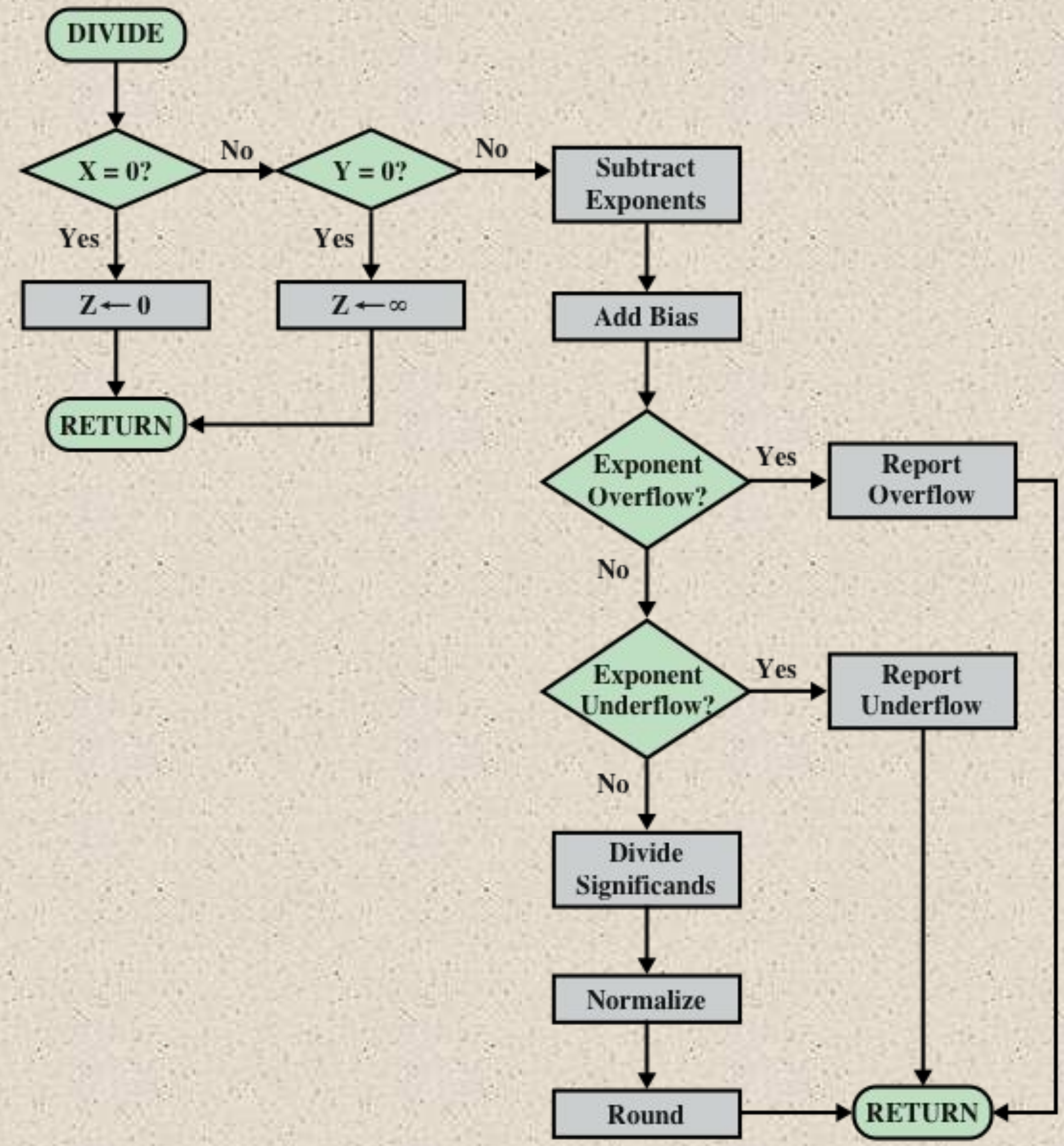


Figure 10.24 Floating-Point Division (Z← X/Y)