Computer Architecture
CSCI 4350

Arithmetic for Computers

Kwangsung Oh

kwangsungoh@unomaha.edu

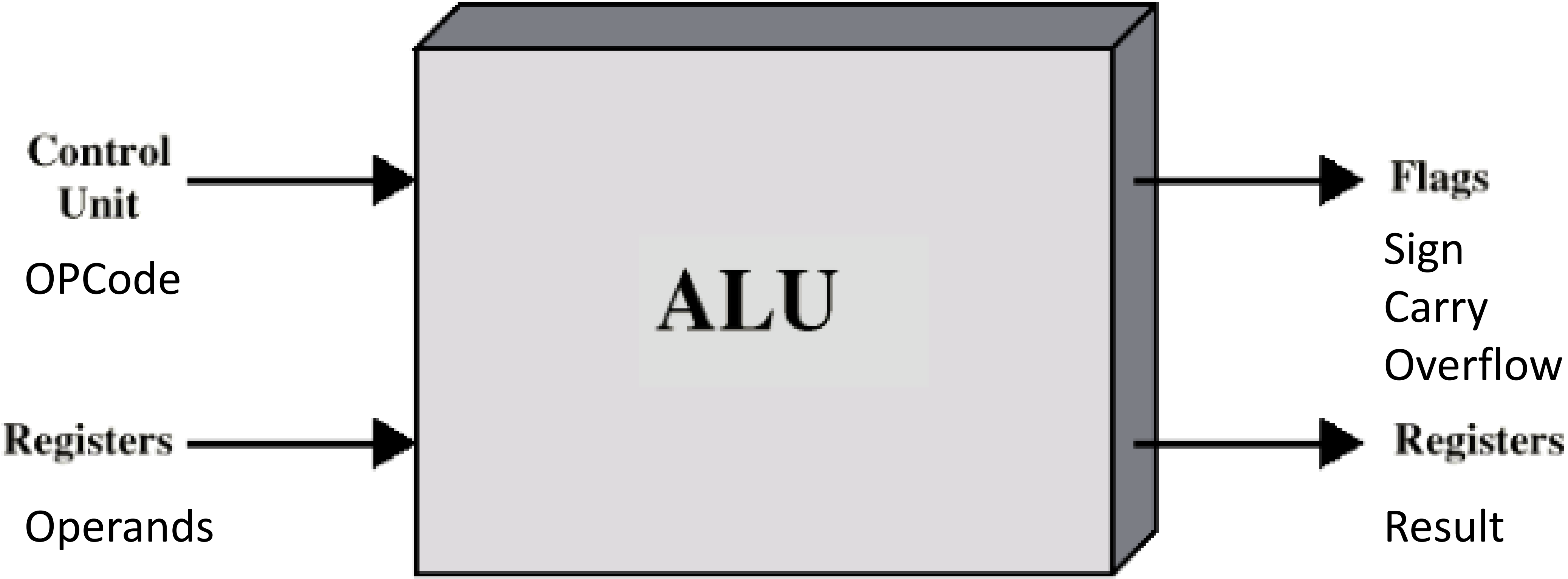
http://faculty.ist.unomaha.edu/kwangsungoh

University of Nebraska Omaha

Arithmetic Logic Unit (ALU)

- Roles of ALU
  - Performing addition, subtraction, and logical operations
  - Everything else in the computer is to service ALU
  - Handling integers
  - FPU (Floating Point Unit) – Unit for floating point (real) numbers
- Implementation
  - All microprocessors has integer ALUs
  - On-chip or off-chip FPU (co-processor)

ALU Inputs and Outputs



OPCode

Operands

Sign
Carry
Overflow

Result

Integer Representation

- Only 0 & 1
- Two representative representations
  - Sign-magnitude
  - Two’s complement
- Sign-magnitude
  - Sign bit (left most bit: 0 – positive, 1 – negative)
  - +18 = **0**0010010
  - -18 = **1**0010010
  - !! Two zeros = +0 (00000000), -0 (10000000)
  - !! Need to consider both sign bit and magnitude in arithmetic

2's Complement

- Given N, 2's complement of N with n-bits
  - $2^n - N = (\underline{2^n - 1) - N} + 1 = \underline{\text{bit complement of N}} + 1$
- 32-bit number
  - Positive number: 0 (x00000000) to $2^{31} - 1$ (x7FFFFFFF)
  - Negative number: -1 (xFFFFFFFF) to -2^{31} (x80000000)
- 3-bit Examples
  - +3 = 011 -4 = 100
  - +2 = 010 -3 = 101
  - +1 = 001 -2 = 110
  - 0 = 000 -1 = 111

Characteristics of 2's Complement

- A single zero
- Simple negation (bit complement of N + 1)
  - 4 = 00000100
  - Bit complement = 11111011
  - Add 1 to LSB = 11111100
- Overflow only when
  - The same sign bit of two numbers but an opposite sign bit of result

Operation	Sign of A	Sign of B	If sign of result
A + B	+	+	-
A + B	-	-	+
A – B	+	-	-
A – B	-	+	+

- Simple arithmetic
  - A – B = A + (-B) -> Adding 2's complement of B to A

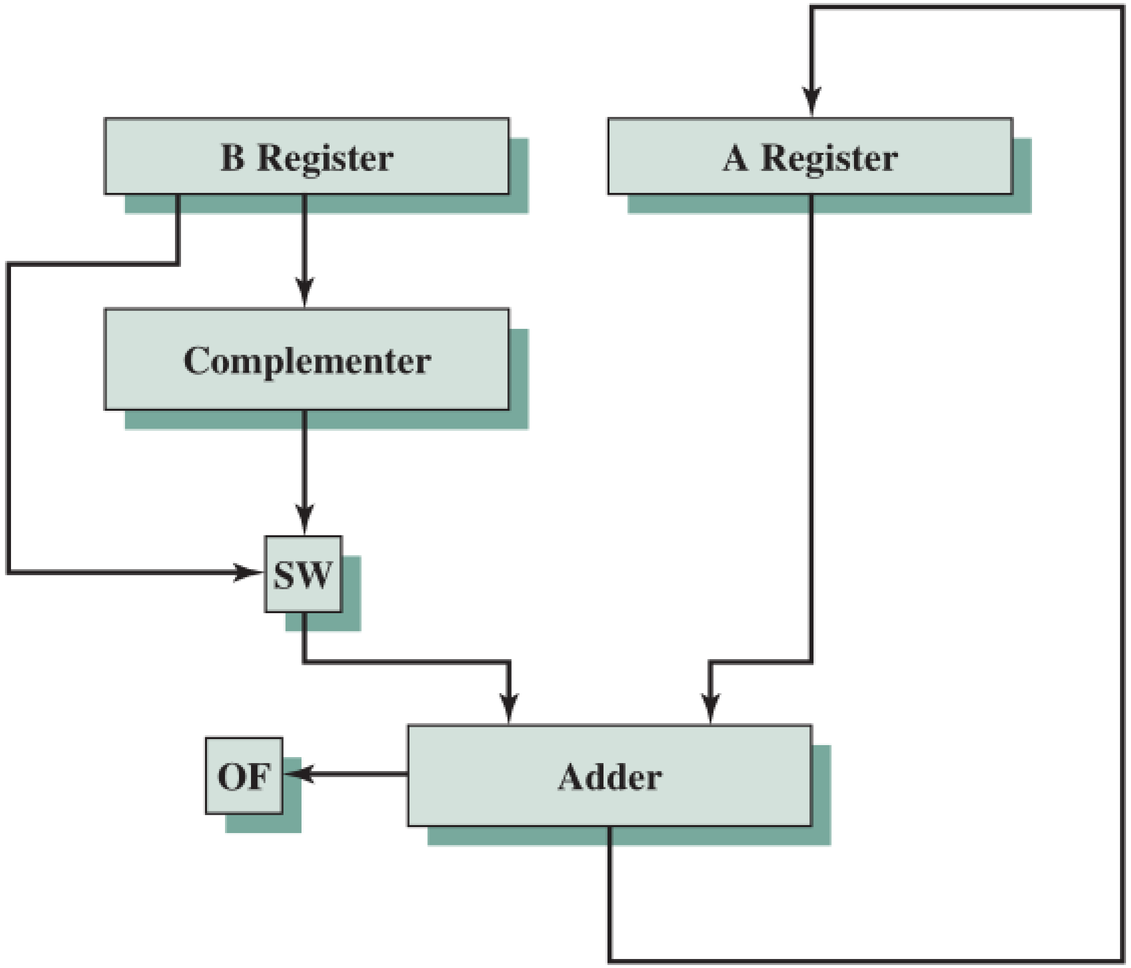
Range of Numbers

- 8-bit 2's complement
  - 127 = 0111 1111 = $2^7 – 1$
  - -128 = 1000 0000 = -2^7
- 16-bit 2's complement
  - 32767 = 0111 1111 1111 1111 = $2^{15} - 1$
  - -32768 = 1000 0000 0000 0000 = -2^{15}
- n-bit 2's complement
  - $-2^{n-1} \sim 2^{n-1} – 1$

Addition and Subtraction

- Addition
  - Normal binary addition
  - Monitor sign bit for overflow
- Subtraction
  - Take the two's complement of subtrahend and add to minuend
  - I.e., $a - b = a + (-b)$
  - Adder and complement circuits needed

Hardware for Addition and Subtraction



OF = Overflow bit
SW = Switch (select addition or subtraction)

Example of Multiplication

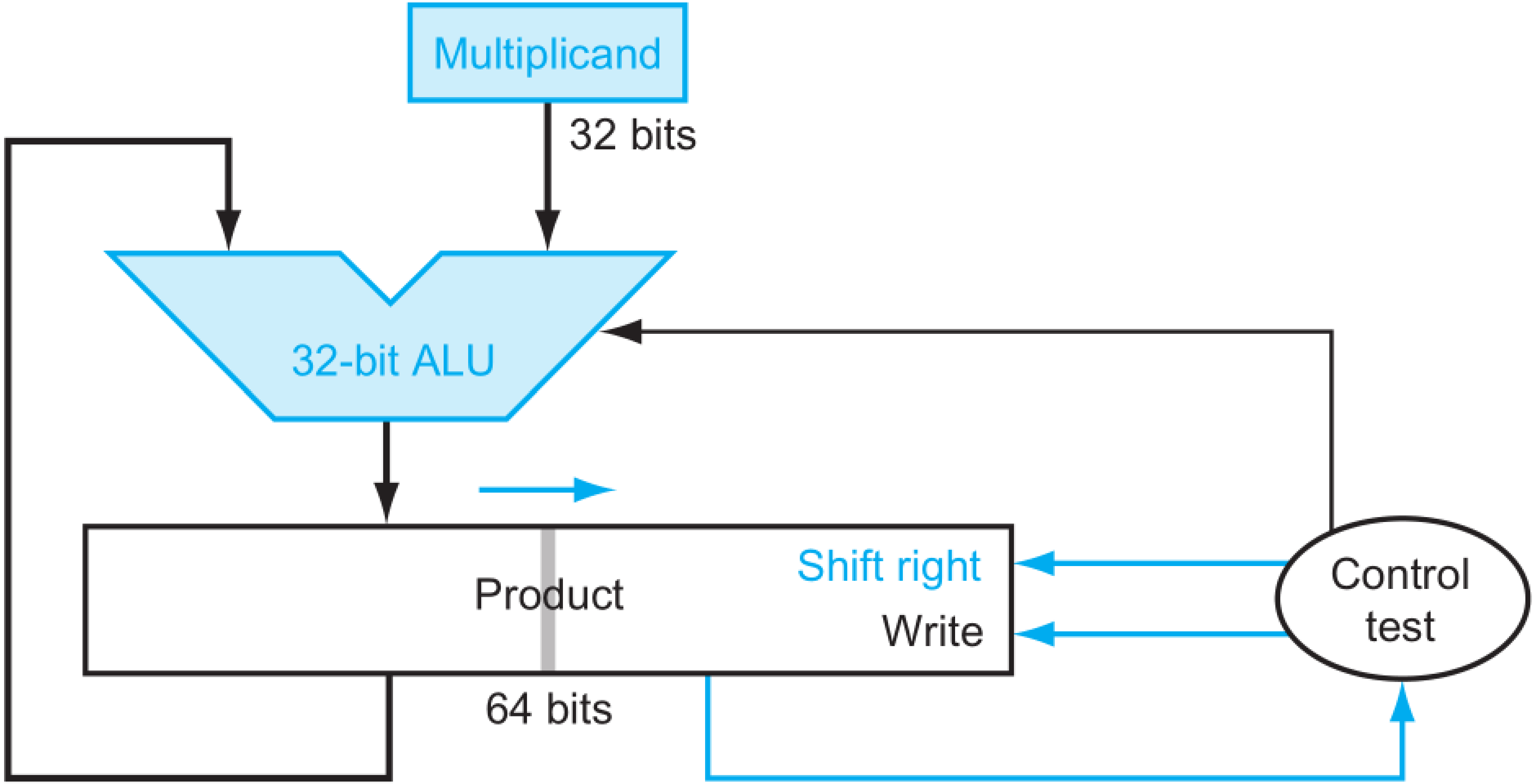
- 8 * 9 => 1000 * 1001

```
Multiplicand        1000  (8 decimal)
Multiplier    x     1001  (9 decimal)
                --------
                    1000        <-- Partial products
                   0000             if multiplier bit is 1, copy multiplicand
                  0000              else 0, put zero
                 1000
               ------------
Product         01001000 (72 decimal)
```

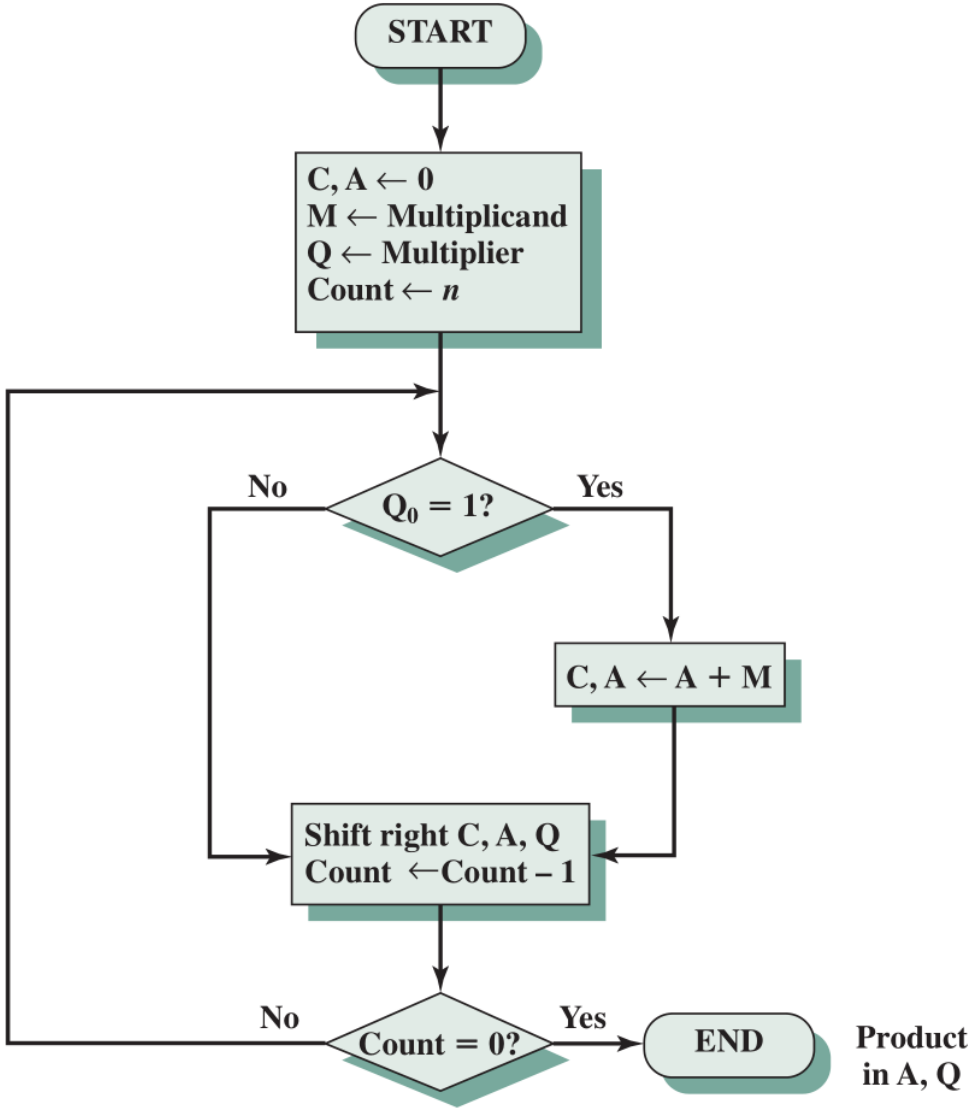
Multiplication

- Principles
  - Work out partial product for each digit
  - Shift each partial product
  - Add partial products
  - Note: need double length result

Multiplication Hardware



Unsigned Multiplication Algorithm



Example of Unsigned Binary Multiplication

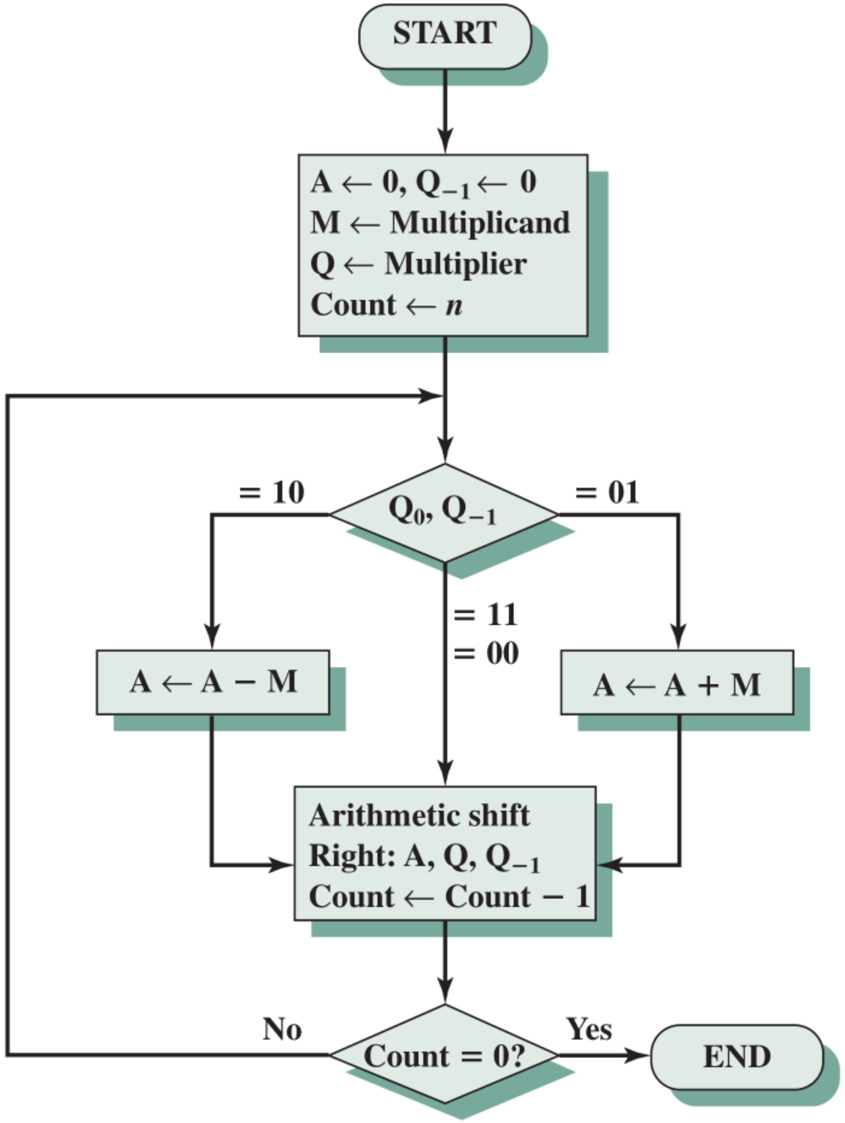
8 (1000) * 9 (1001)

C (Carry)	R1 (A)	R2 (Q) (Multiplier)	R3 (M) (Multiplicand)	
	Product			
0	0000	100**1**	1000	Initial values
0	1000	1001	1000	Add
0	0100	010**0**	1000	Shift
0	0010	001**0**	1000	Shift
0	0001	000**1**	1000	Shift
0	1001	0001	1000	Add
0	**0100**	**1000**	1000	Shift

Signed Multiplication

- Unsigned binary multiplication algorithm
  - Does not work for signed multiplication
- Solution 1
  - Converting to positive number
  - Multiplying as above
  - If sings were different, negate the answer
- Solution 2
  - Booth's algorithm

Booth’s Algorithm



Example of Booth's Algorithm

Production

8 (1000) * 9 (1001)

R1 (A)	R2 (Q) (Multiplier)	Q_{-1}	R3 (M) (Multiplicand)	
00000	01001	0	01000	Initial values
11000	01001	0	01000	Sub (11000 – 2's complement): A + (-M)
11100	00100	1	01000	Shift
00100	00100	1	01000	Add (01000): A + M
00010	00010	0	01000	Shift
00001	00001	0	01000	Shift
11001	00001	0	01000	Sub (11000 - 2's complement): A + (-M)
11100	10000	1	01000	Shift
00100	10000	1	01000	Add (01000 - 2's complement): A + (-M)
00010	**01000**	0	01000	Shift

Example of Booth's Algorithm

Production

11 (01011) * -13 (10011)

R1 (A)	R2 (Q) (Multiplier)	Q_{-1}	R3 (M) (Multiplicand)	
00000	10011	0	01011	Initial values
10101	10011	0	01011	Sub (10101, 2's complement): A + (-M)
11010	11001	1	01011	Shift
11101	01100	1	01011	Shift
01000	01100	1	01011	Add (01011): A + (-M)
00100	00110	0	01011	Shift
00010	00011	0	01011	Shift
10111	00011	0	01011	Sub (10101, 2's complement): A + (-M)
11011	**10001**	0	01011	Shift

Examples of More 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)

Division

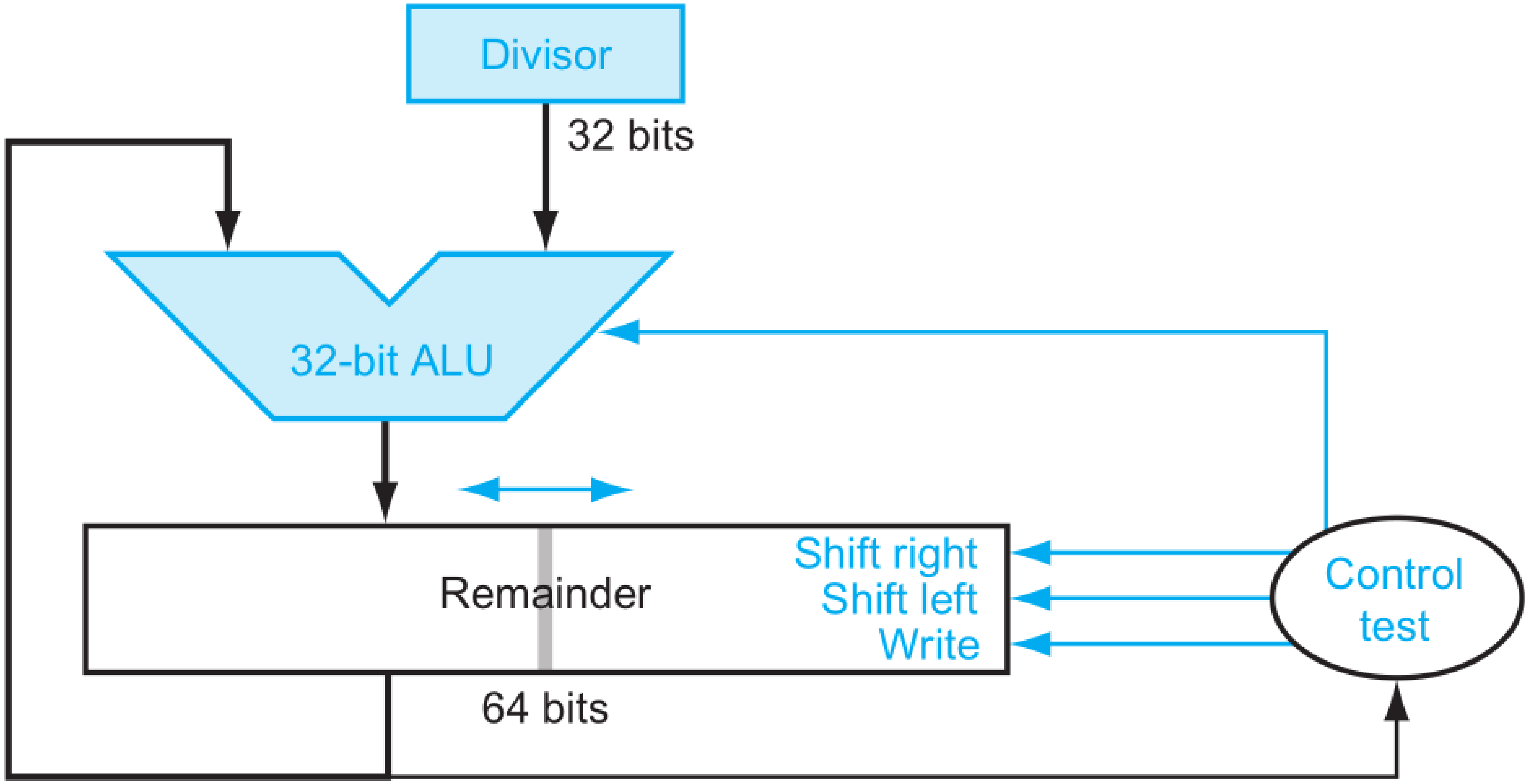
- Unsigned binary division
  - By shift and subtract
- Signed binary division
  - More complex than multiplication
  - The unsigned binary division can be extended to negative numbers

Example of Division

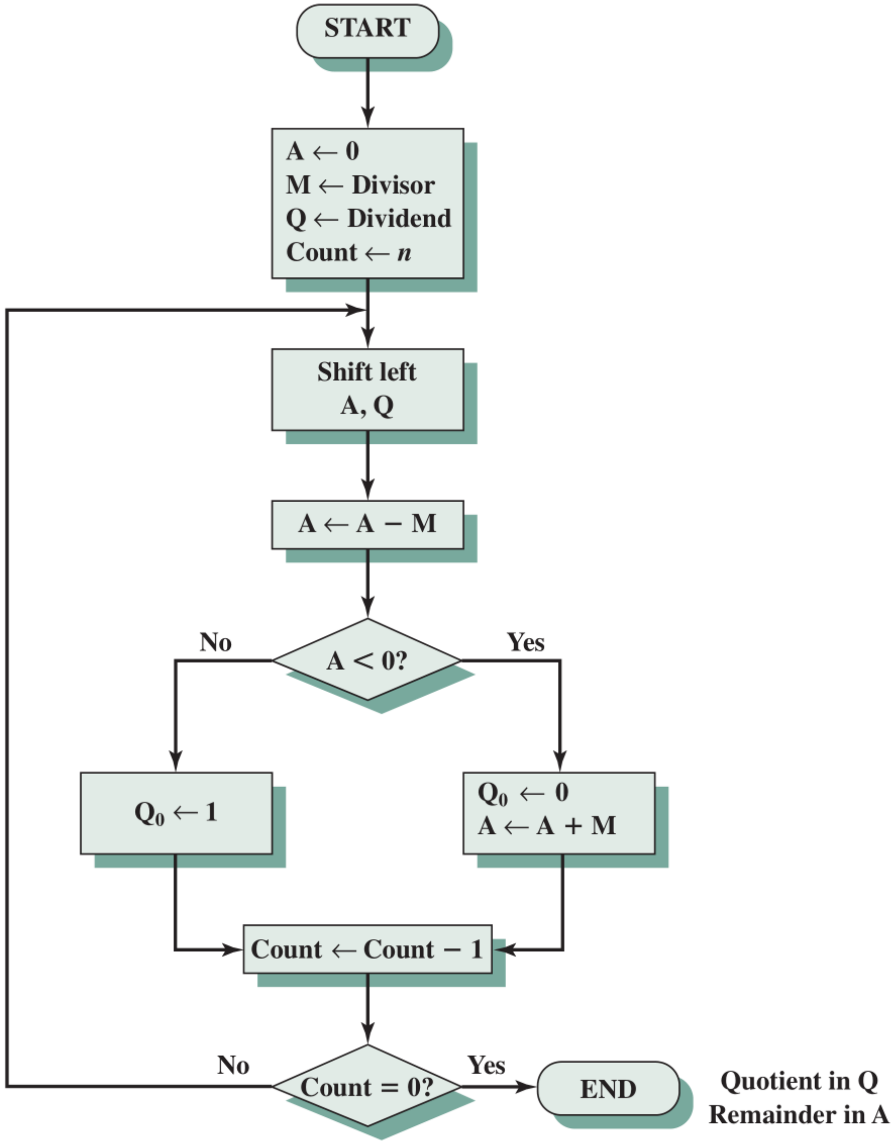
- 147 / 11 -> 10010011 / 1011

```
                     00001101   <---- Quotient
Divisor ----> 1011 | 10010011   <---- Dividend
                      1011
                      001110
Partial                 1011
Remainders ---->        001111
                          1011
                           100  <---- Reminder
```

Division Hardware



Unsigned Division Algorithm



Unsigned Binary Division Example

R1 (A) (Remainder)	Quotient R2 (Q) (Dividend)	R3 (M) (Divisor)	7 (0111) / 2 (0010)
0000	0111	0010	Initial values
0000	111_	0010	Shift Left
1110	111**0**	0010	Sub (1110, 2’s complement): A + (-M)
0000	1110	0010	Add (0010): A + M
0001	110_	0010	Shift Left
1111	110**0**	0010	Sub (1110, 2’s complement): A + (-M)
0001	1100	0010	Add (0010): A + M
0011	100_	0010	Shift Left
0001	100**1**	0010	Sub (1110, 2’s complement): A + (-M)
0011	001_	0010	Shift Left
0001	**0011**	0010	Sub (1110, 2’s complement): A + (-M)

Signed Division Algorithm

- Extending unsigned binary division
  1. 2n-bit 2’s complement number for a negative number
  2. Shift A, Q left by 1-bit position
  3. Checking A sign and M sign
  4. If same A - M, else A + M
  5. If sign of A is “the same as before” or (A = 0 and Q = 0), $Q_0 = 1$, else $Q_0 = 0$ and restore A value
  6. Repeat 2 ~ 4 n times
  7. If the signs of the divisor and dividend are the same, Q is quotient, else the quotient is the 2’s complement of Q

Examples of Signed Division

A	Q	M = 0011
0000	0111	Initial value
0000	1110	shift
1101		subtract
0000	1110	restore
0001	1100	shift
1110		subtract
0001	1100	restore
0011	1000	shift
0000		subtract
0000	1001	set $Q_0 = 1$
0001	0010	shift
1110		subtract
0001	0010	restore

(a) (7)/(3)

A	Q	M = 1101
0000	0111	Initial value
0000	1110	shift
1101		add
0000	1110	restore
0001	1100	shift
1110		add
0001	1100	restore
0011	1000	shift
0000		add
0000	1001	set $Q_0 = 1$
0001	0010	shift
1110		add
0001	0010	restore

(b) (7)/(–3)

A	Q	M = 0011
1111	1001	Initial value
1111	0010	shift
0010		add
1111	0010	restore
1110	0100	shift
0001		add
1110	0100	restore
1100	1000	shift
1111		add
1111	1001	set $Q_0 = 1$
1111	0010	shift
0010		add
1111	0010	restore

(c) (–7)/(3)

A	Q	M = 1101
1111	1001	Initial value
1111	0010	shift
0010		subtract
1111	0010	restore
1110	0100	shift
0001		subtract
1110	0100	restore
1100	1000	shift
1111		subtract
1111	1001	set $Q_0 = 1$
1111	0010	shift
0010		subtract
1111	0010	restore

(d) (–7)/(–3)

Real Numbers

- Numbers with fractions
  - 10.23, 0.999993, 358.323, and 3.14159…
- Pure binary: 1001.1010 = $2^3 + 2^0 + 2^{-1} + 2^{-3}$ = 9.625
- Where to put the binary point?
  - **Fixed-point**

    - Limited to present very large number and very small fraction
  - **Floating point**

    - Use the exponent to slide (place) the binary point

    - 976,000,000,000,000. = $9.76 * 10^{14}$

    - 0.0000000000000976 = $9.76 * 10^{-14}$

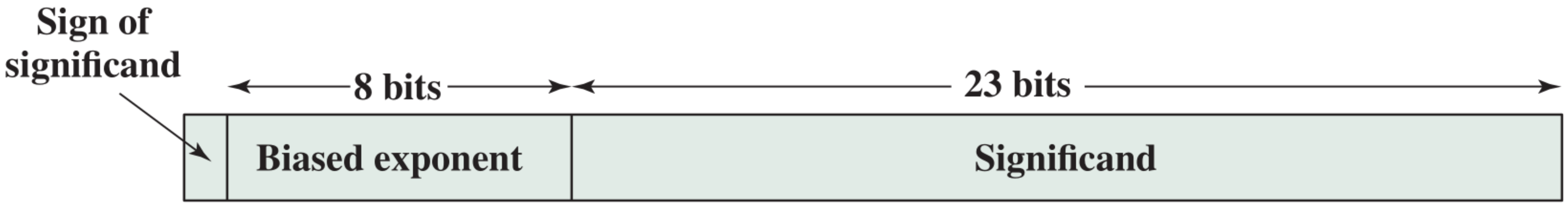
Floating Point Number

- Three components for FP number
  - **Sign** * **Significand** (Mantissa) * $\underline{2}^{\pm \mathbf{Exponent}}$ — Base 2 is omitted

Sign bit	Biased Exponent (E)	Significand or Mantissa (S)

  - Exponent **E** (Biased representation)
    - Fixed value (bias) for k-bit exponent: $2^{k-1} - 1$ (higher number representation)
    - E.g., for 8-bit exponent -127 ~ 128
    - Simple comparison for nonnegative FP (bigger exponent is bigger)
  - Significant **S** (Normalized representation)
    - $\pm\ \underline{\mathbf{1}.}\text{bbb} \ldots \text{bbb} * 2^{\pm E}$
    - The most significant digit is always **1** (omitted)
    - Thus, 23-bit significand can store 24-bit significand **[1, 2)**

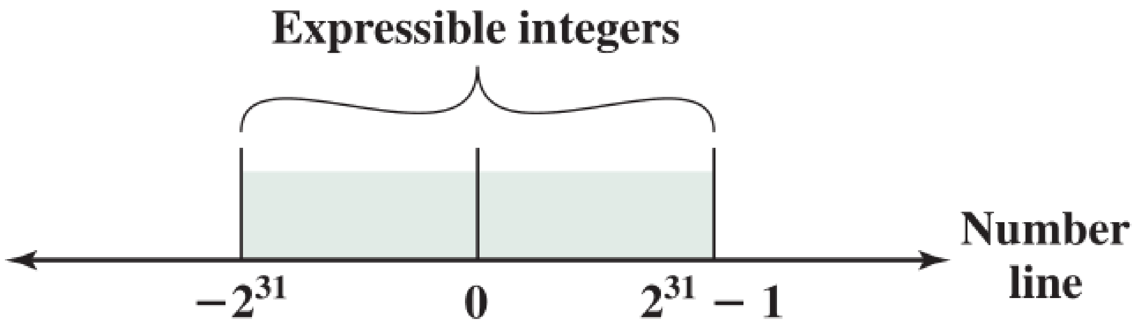
32-bit Floating Point Number Examples



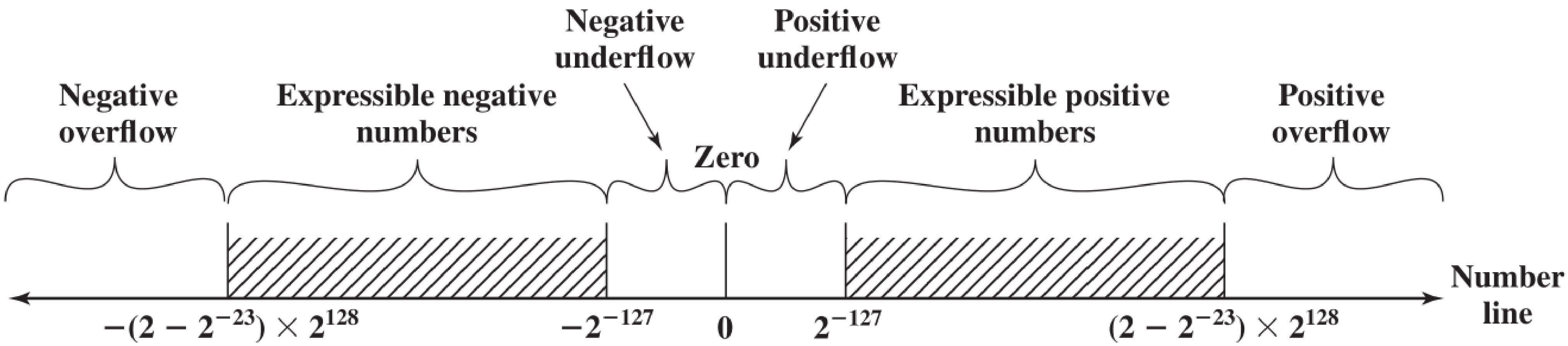
$$
\begin{array}{rcl}
1.1010001 \times 2^{10100} & = 0\ 10010011\ 10100010000000000000000 = & 1.6328125 \times 2^{20} \\
-1.1010001 \times 2^{10100} & = 1\ 10010011\ 10100010000000000000000 = & -1.6328125 \times 2^{20} \\
1.1010001 \times 2^{-10100} & = 0\ 01101011\ 10100010000000000000000 = & 1.6328125 \times 2^{-20} \\
-1.1010001 \times 2^{-10100} & = 1\ 01101011\ 10100010000000000000000 = & -1.6328125 \times 2^{-20}
\end{array}
$$

What is the smallest number in this representation?

Expressible Numbers in 32-bit Formats



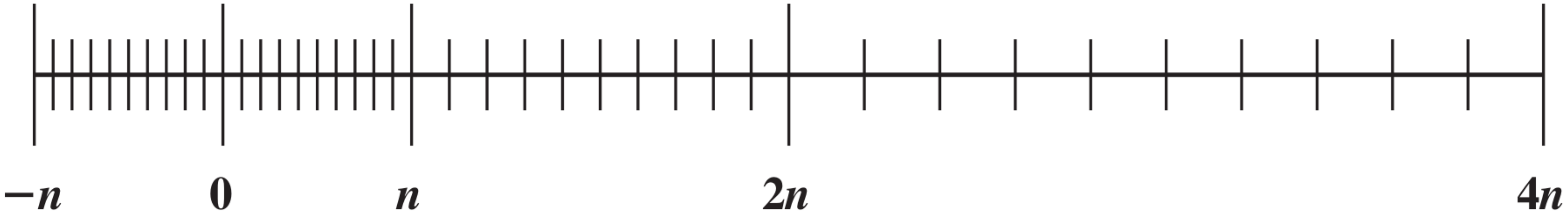
Twos complement integers



Floating-point numbers

Density of Floating-Point Numbers

- The maximum number of different values that can be represented with 32 bits is still 2^{32}
- FP numbers are not spaced evenly along the number line – Larger numbers are spaced more sparsely than smaller numbers



Standard for FP Number (IEEE 754)

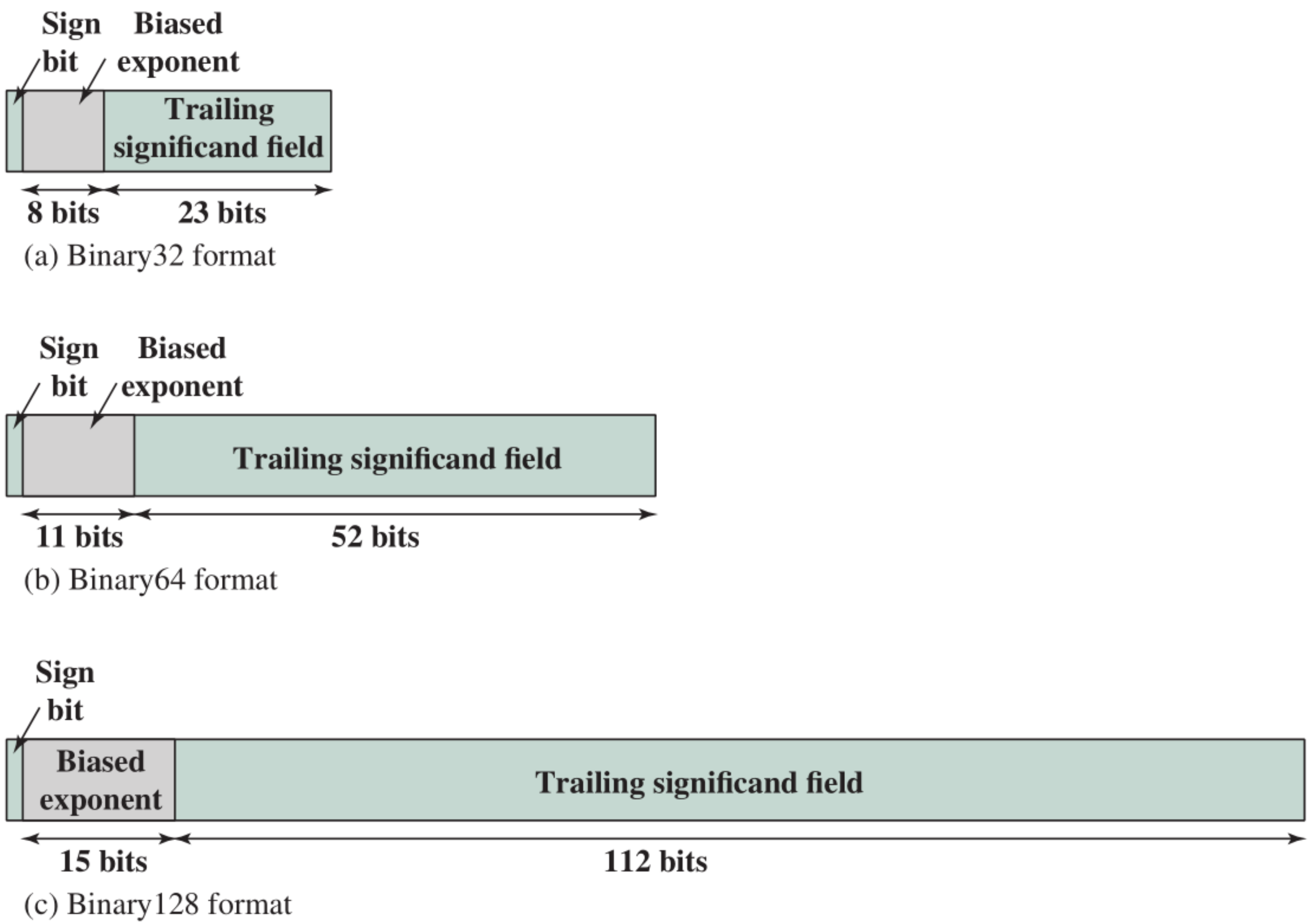
- Supported by virtually all commercial microprocessors
- Formats
  - 32-bit single precision – 8-bit exponent, 23-bit fraction
  - 64-bit double precision – 11-bit exponent, 52-bit fraction
- Characteristics
  - Range: -126 ~ 127 (single), -1022 ~ 1023 (double)

Number	**Exponent**	**Significand**
± 0	All 0	All 0
± ∞	All 1	All 0
Denormalized	All 0	Nonzero
NaN (Not a Number)	All 1	Nonzero

NaN (Not a Number)

- The following practices may cause NaNs
  - All mathematical operations with a NaN as at least one operand
  - The divisions 0/0, ∞/∞, $\infty/-\infty$, $-\infty/\infty$, and $-\infty/-\infty$
  - The multiplications $0\times\infty$ and $0\times-\infty$
  - The additions $\infty + (-\infty)$, $(-\infty) + \infty$ and equivalent subtractions
  - Applying a function to arguments outside its domain
    - Taking the square root of a negative number
    - Taking the logarithm of zero or a negative number
    - Taking the inverse sine or cosine of a number which is less than -1 or greater than +1

IEEE 754 Format



Encoding FP

- Example: 18.3 (32-bit)

18 = 10010

0.3 = 01001100110011001100110

18.3 = 10010.01001100110011001100110

= 1.**00100100110011001100110** x 2^4 (normalized)

IEEE 754 Format:

Sign bit: **0** (+)

Biased exponent: **10000011** = 4 + 127 (biased)

Significand = **00100100110011001100110**

Binary: 0 10000011 00100100110011001100110

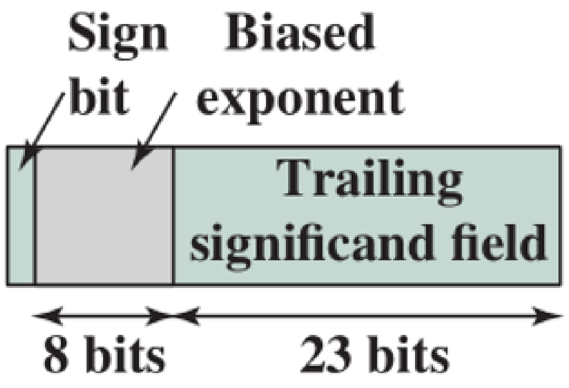
0100 0001 1001 0010 0110 0110 0110 0110

Hex: 0x41926666 = 18.3

0.3 x 2 = 0.6	0
0.6 x 2 = **1**.2	1
0.2 x 2 = 0.4	0
0.4 x 2 = 0.8	0
0.8 x 2 = **1**.6	1
0.6 x 2 = **1**.2	1

…

repeat (for 23 bit)

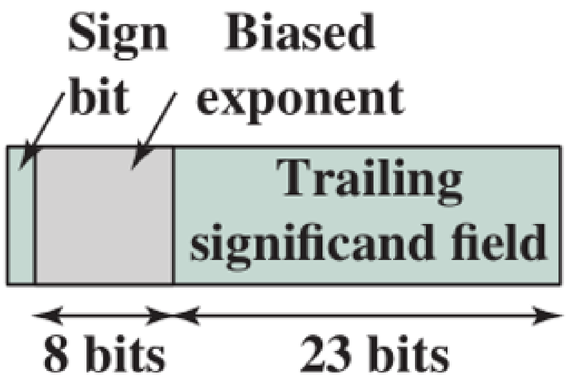


Decoding FP

- Example: 0xC34D4000 (hex)

C34D400 = 1100 0011 0100 1101 0100 0000 0000 0000

= 1 10000110 10011010100000000000000



IEEE 754 Format:

Sign bit: **1** (-)

Biased exponent: **10000110** = 134 - 127 (biased) = **7** (Exponent)

Significand = **100110101**

C34D400 = $-1.100110101 \times 2^{7}$ = -11001101.01 (Shift point to the right **7**)

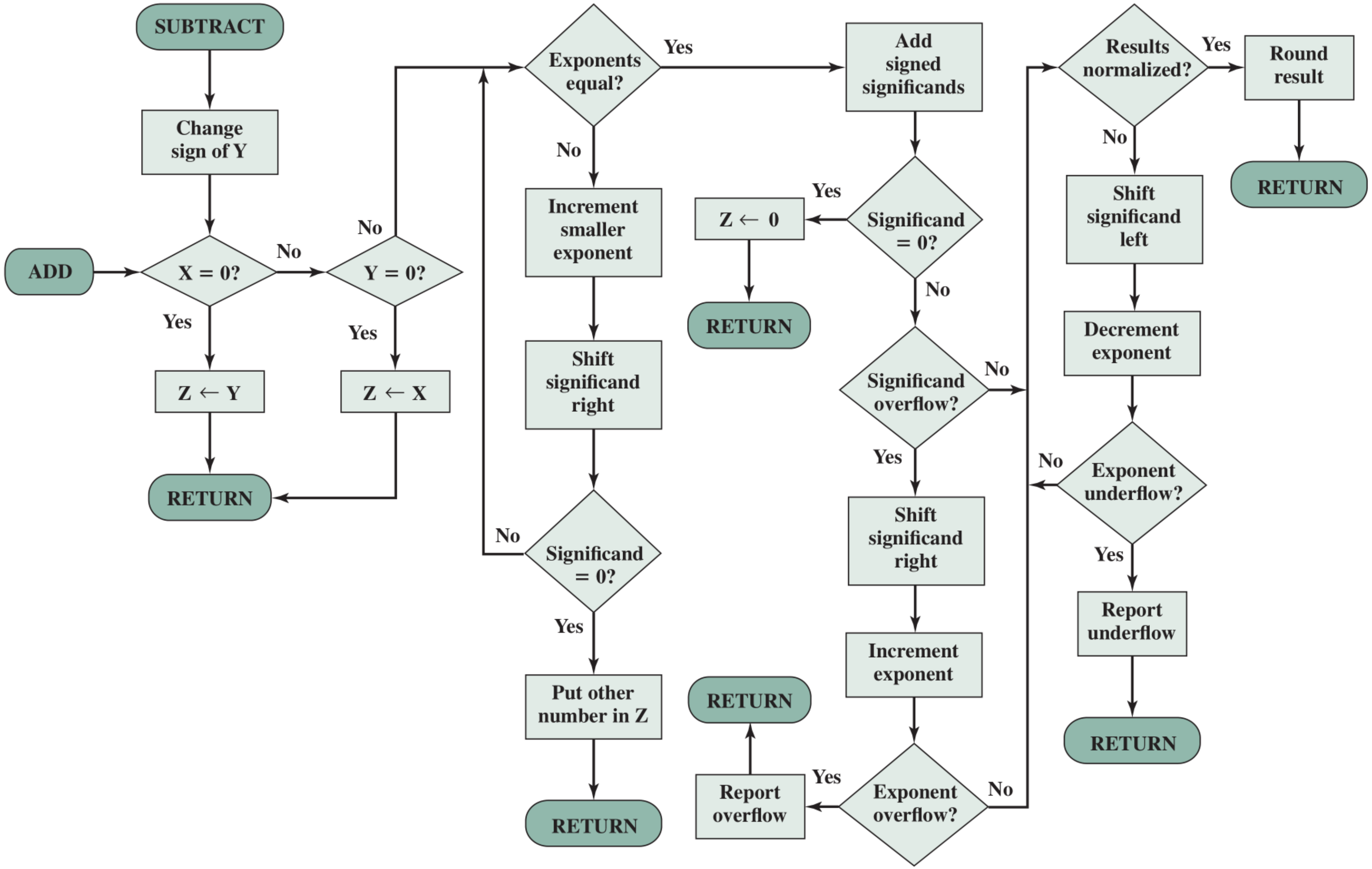
- 1	1	0	0	1	1	0	1.	0	1
128	64	32	16	8	4	2	1.	½	¼

= -205.25

FP Arithmetic +/-

- 4 Phases
  - Check for zeros
  - Align the significand of a smaller number (adjust the exponent)
  - Add or subtract the significands
  - Normalize the result

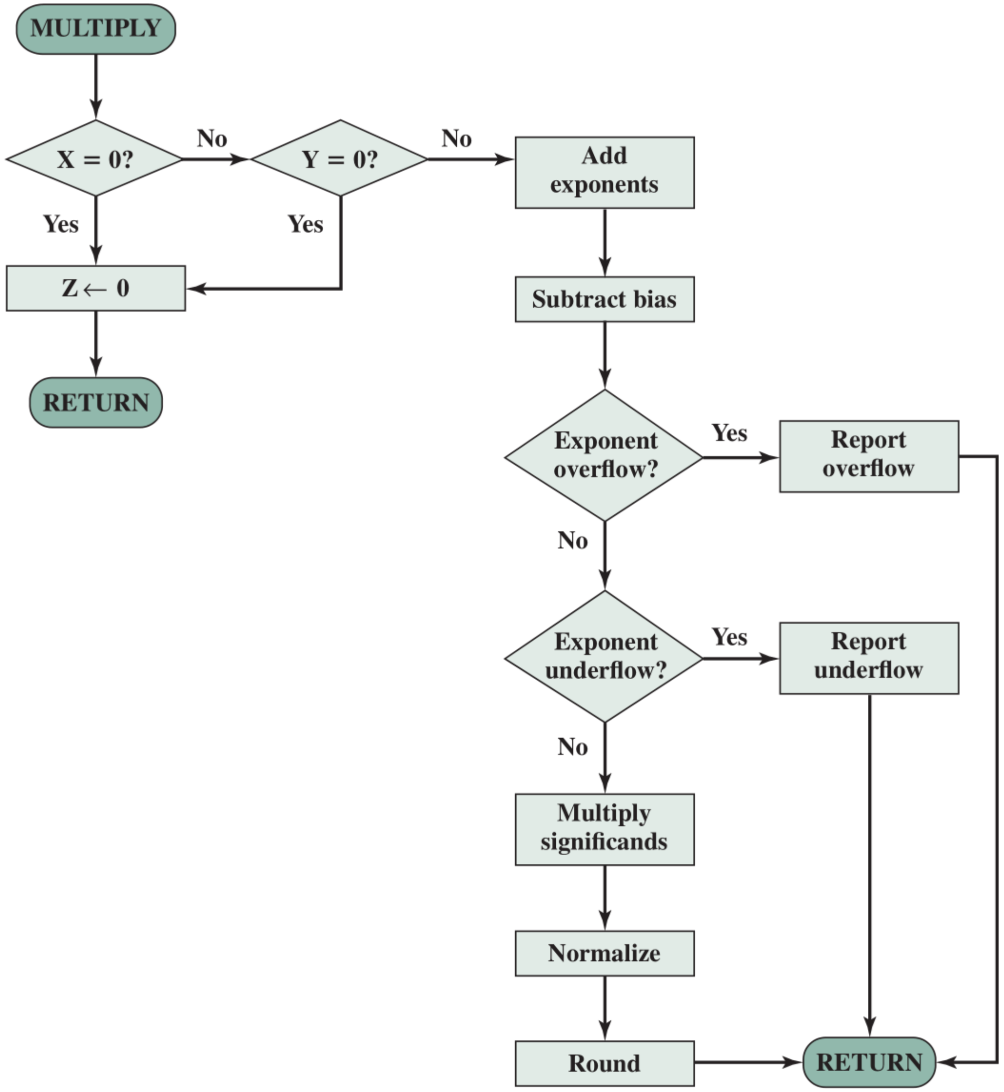
FP Arithmetic +/-



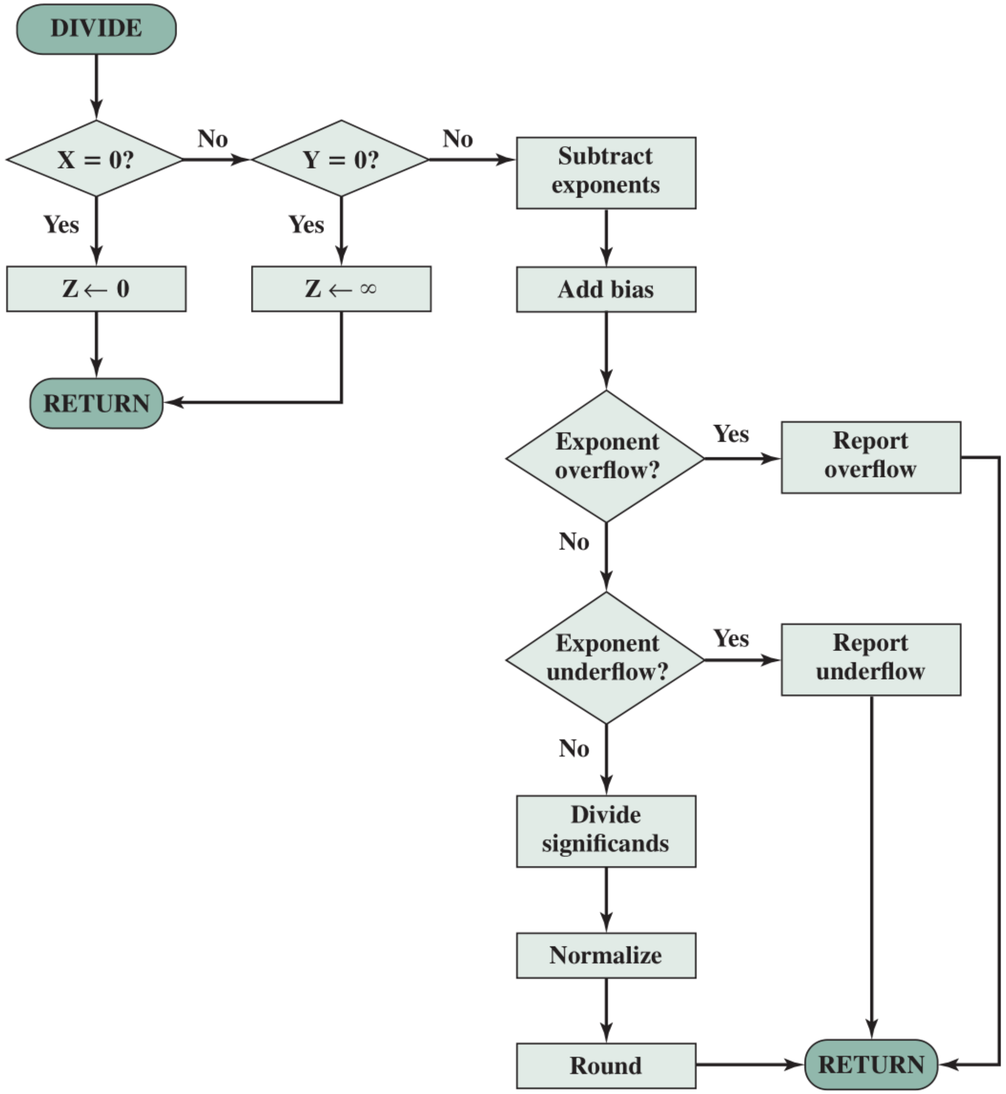
FP Arithmetic *x*/÷

- 4 Phases
  - Check for zeros
  - Add/subtract exponents
  - Multiply/divide significands (watch sign)
  - Normalize the result

FP Multiplication



FP Division



DIVIDE
X = 0?
No
Y = 0?
No
Subtract exponents
Yes
Yes
Z ← 0
Z ← ∞
Add bias
RETURN
Exponent overflow?
Yes
Report overflow
No
Exponent underflow?
Yes
Report underflow
No
Divide significands
Normalize
Round
RETURN