

William Stallings  
Computer Organization  
and Architecture  
8<sup>th</sup> Edition

---

Chapter 9  
Computer Arithmetic

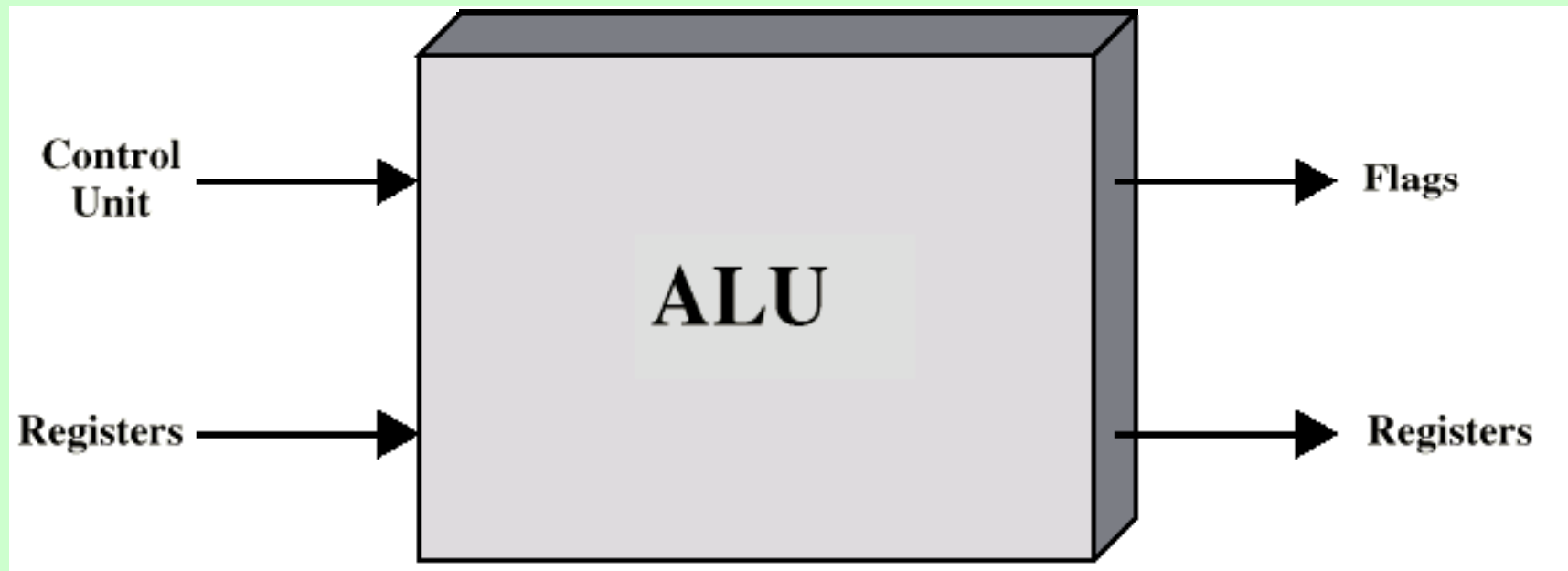
# Arithmetic & Logic Unit

---

- Does the calculations
- Everything else in the computer is there to service this unit
- Handles integers
- May handle floating point (real) numbers
- May be separate FPU (maths co-processor)
- May be on chip separate FPU (486DX +)

# ALU Inputs and Outputs

---



# Integer Representation

---

- Only have 0 & 1 to represent everything
- Positive numbers stored in binary
  - e.g.  $41 = 00101001$
- No minus sign
- No period
- Sign-Magnitude
- Two's compliment

# Sign-Magnitude

---

- Left most bit is sign bit
- 0 means positive
- 1 means negative
- $+18 = 00010010$
- $-18 = 10010010$
- Problems
  - Need to consider both sign and magnitude in arithmetic
  - Two representations of zero (+0 and -0)

# Two's Complement

---

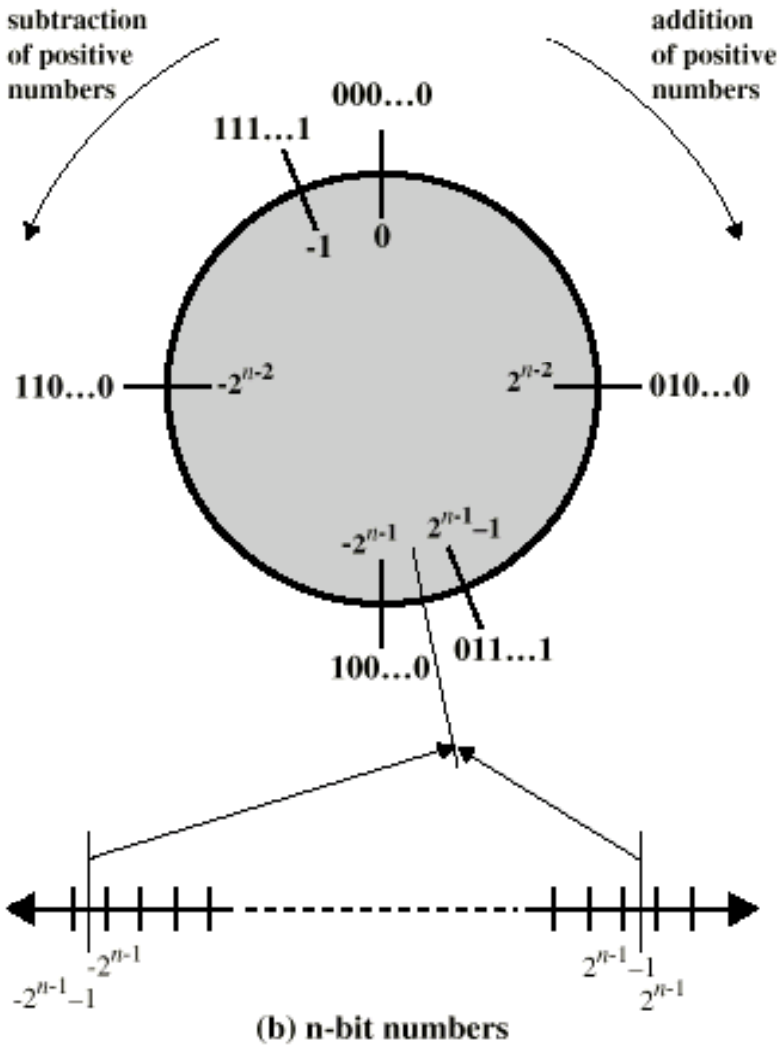
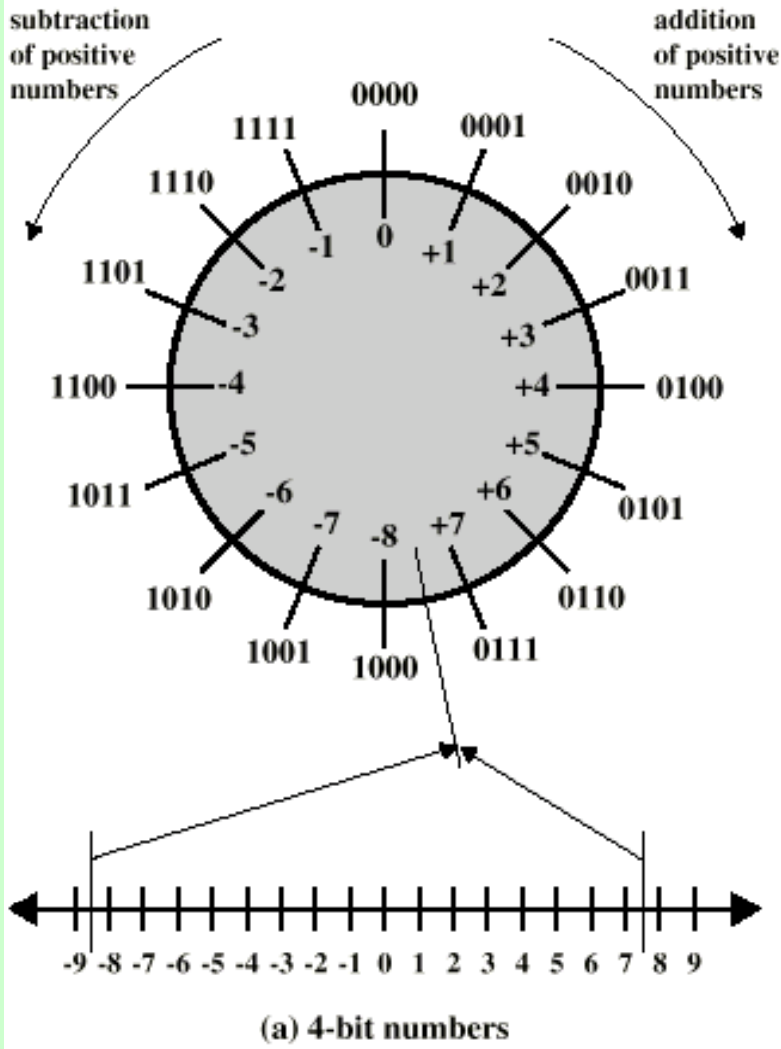
- $+3 = 00000011$
- $+2 = 00000010$
- $+1 = 00000001$
- $+0 = 00000000$
- $-1 = 11111111$
- $-2 = 11111110$
- $-3 = 11111101$

## Benefits

---

- One representation of zero
- Arithmetic works easily (see later)
- Negating is fairly easy
  - $-3 = 00000011$
  - Boolean complement gives  $11111100$
  - Add 1 to LSB  $11111101$

# Geometric Depiction of Twos Complement Integers





## Negation Special Case 1

---

- 0 = 00000000
- Bitwise not 11111111
- Add 1 to LSB +1
- Result 1 00000000
- Overflow is ignored, so:
- - 0 = 0 ✓

## Negation Special Case 2

---

- $-128 = 10000000$
- bitwise not  $01111111$
- Add 1 to LSB  $+1$
- Result  $10000000$
- So:
- $-(-128) = -128 \quad X$
- Monitor MSB (sign bit)
- It should change during negation

## Range of Numbers

---

- 8 bit 2s compliment

— +127 = 01111111 =  $2^7 - 1$

— -128 = 10000000 =  $-2^7$

- 16 bit 2s compliment

— +32767 = 01111111 11111111 =  $2^{15} - 1$

— -32768 = 10000000 00000000 =  $-2^{15}$

## Conversion Between Lengths

---

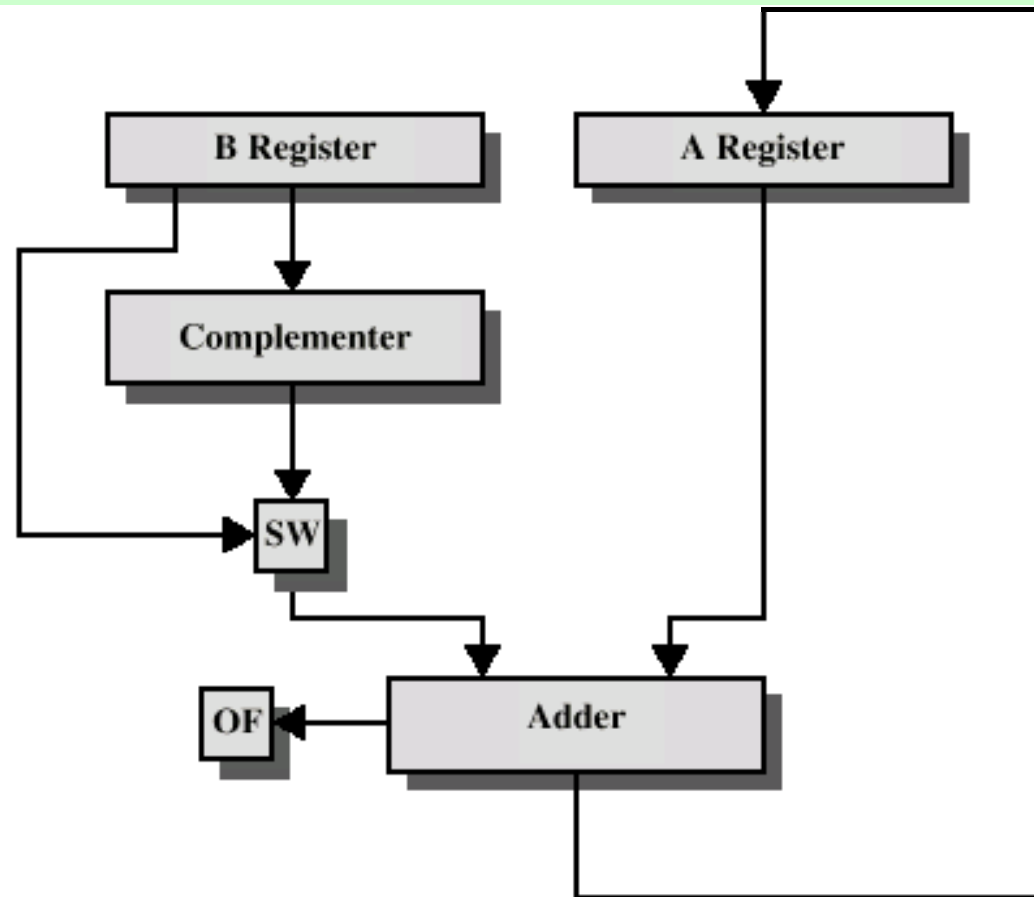
- Positive number pack with leading zeros
- $+18 = \quad\quad\quad 00010010$
- $+18 = 00000000\ 00010010$
- Negative numbers pack with leading ones
- $-18 = \quad\quad\quad 10010010$
- $-18 = 11111111\ 10010010$
- i.e. pack with MSB (sign bit)

## Addition and Subtraction

---

- Normal binary addition
- Monitor sign bit for overflow
- Take two's complement of subtrahend and add to minuend
  - i.e.  $a - b = a + (-b)$
- So we only need addition and complement circuits

# Hardware for Addition and Subtraction



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

# Multiplication

---

- Complex
- Work out partial product for each digit
- Take care with place value (column)
- Add partial products

## Multiplication Example

---

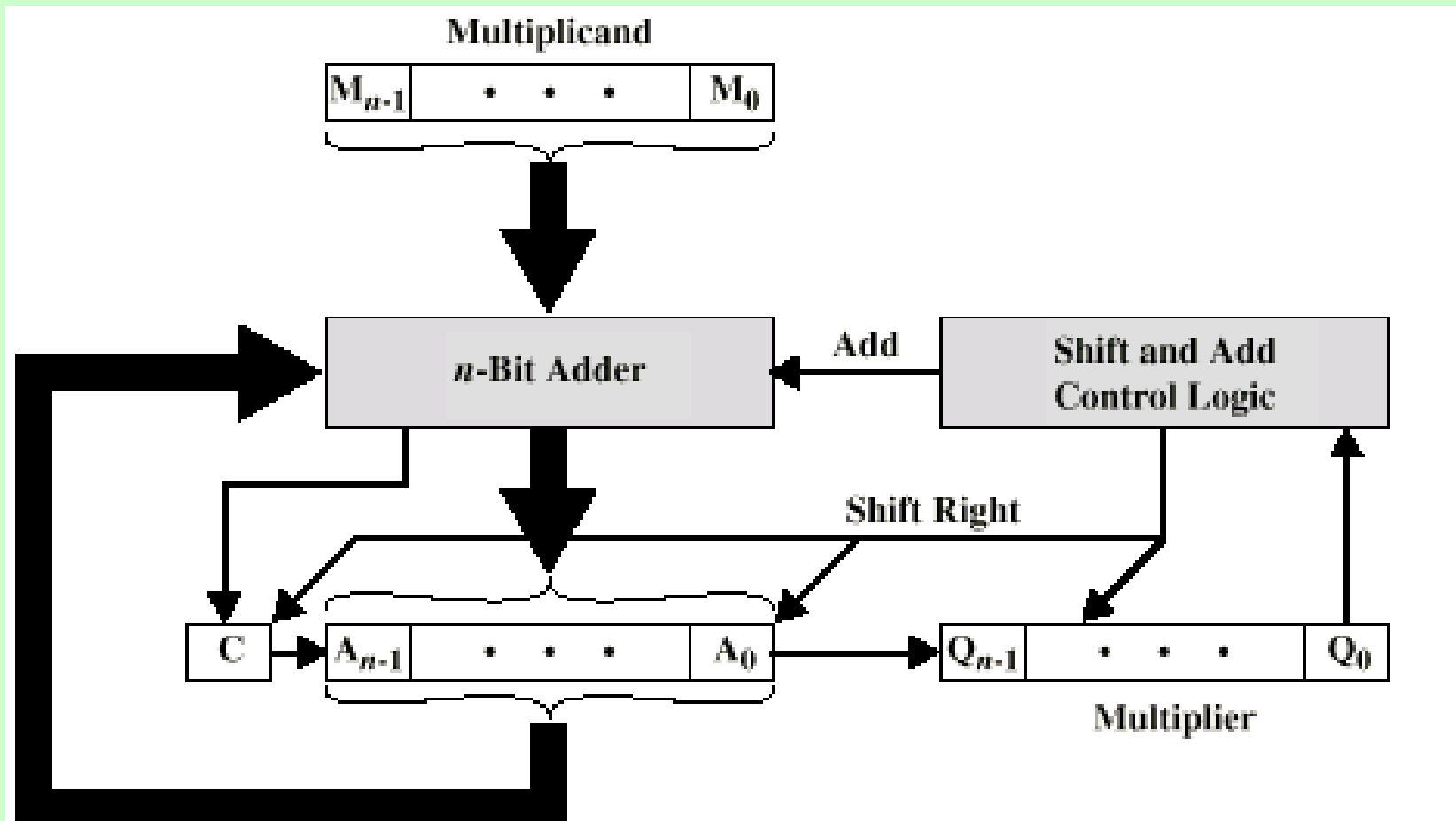
- 1011 Multiplicand (11 dec)
- x 1101 Multiplier (13 dec)
- 1011 Partial products
- 0000 Note: if multiplier bit is 1 copy  
multiplicand (place value)
- 1011 otherwise zero
- 10001111 Product (143 dec)
- Note: need double length result

---

---



# Unsigned Binary Multiplication



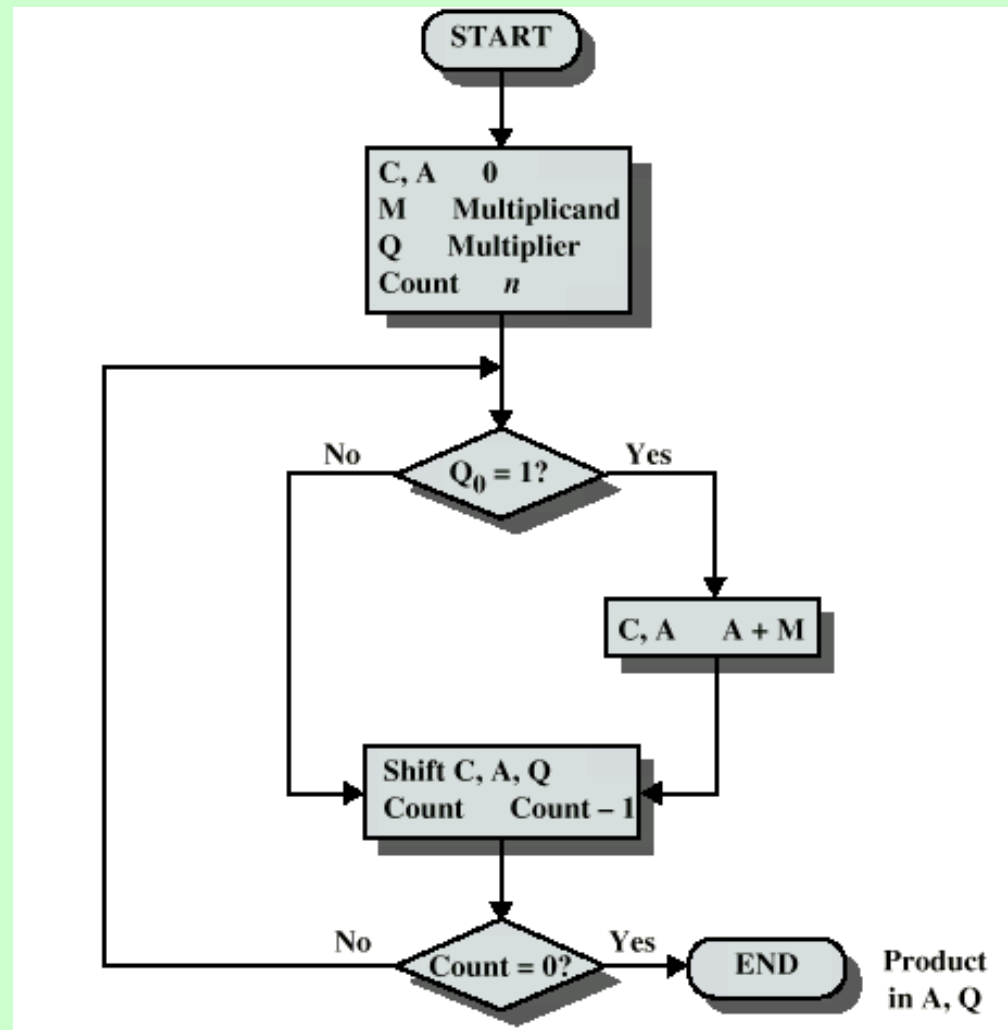
(a) Block Diagram

# Execution of Example

---

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	

# Flowchart for Unsigned Binary Multiplication

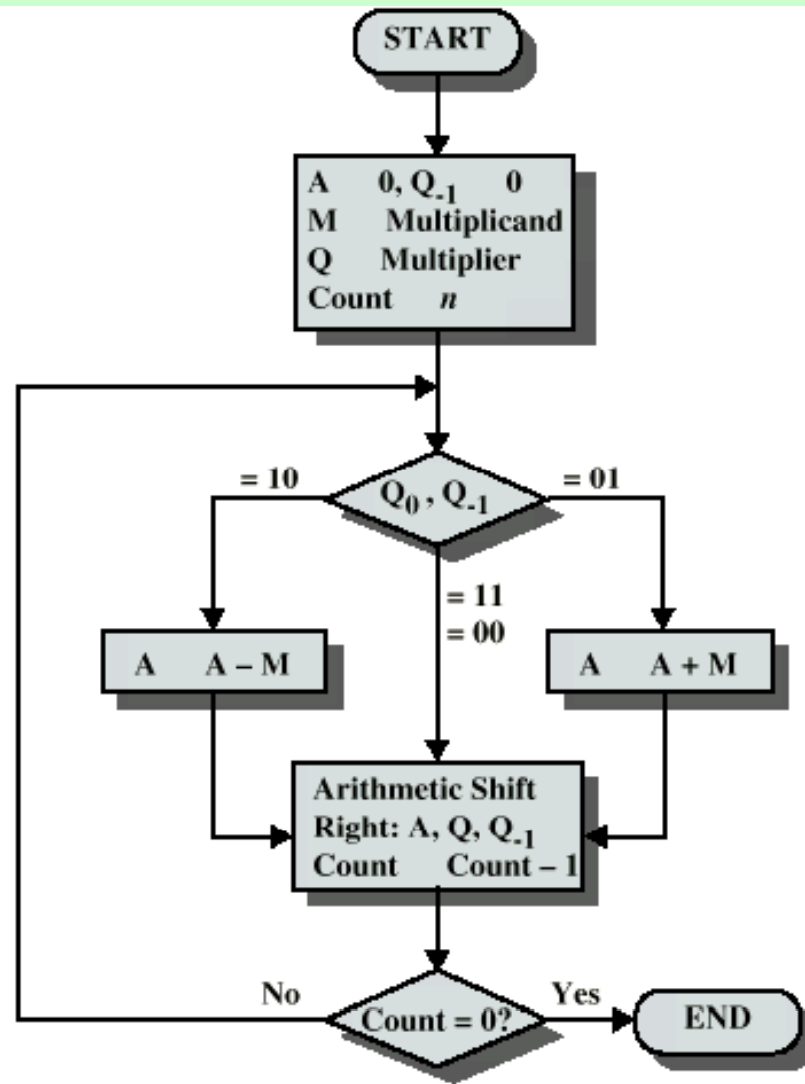


# Multiplying Negative Numbers

---

- This does not work!
- Solution 1
  - Convert to positive if required
  - Multiply as above
  - If signs were different, negate answer
- Solution 2
  - Booth's algorithm

# Booth's Algorithm



# Example of Booth's Algorithm

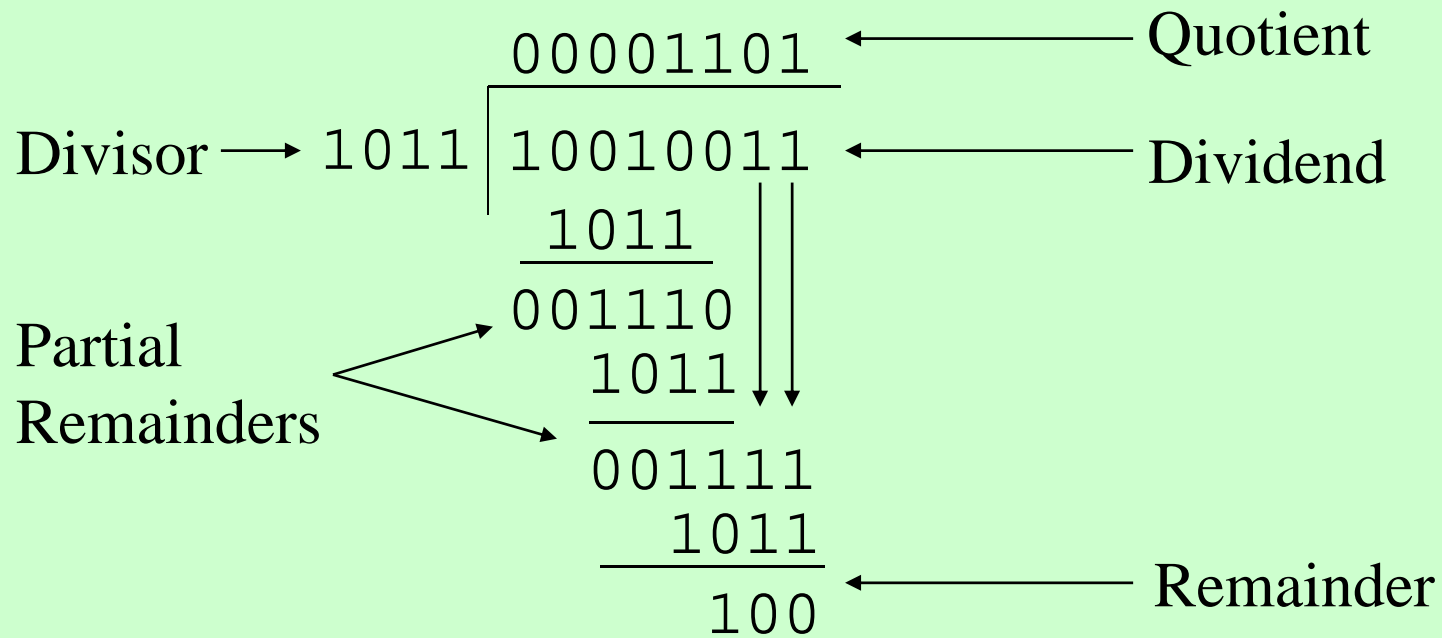
A	Q	Q <sub>-1</sub>	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A	} First Cycle
1100	1001	1	0111	A - M Shift	
1110	0100	1	0111	Shift	} Second Cycle
0101	0100	1	0111	A	
0010	1010	0	0111	A + M Shift	} Third Cycle
0001	0101	0	0111	Shift	
					} Fourth Cycle

# Division

---

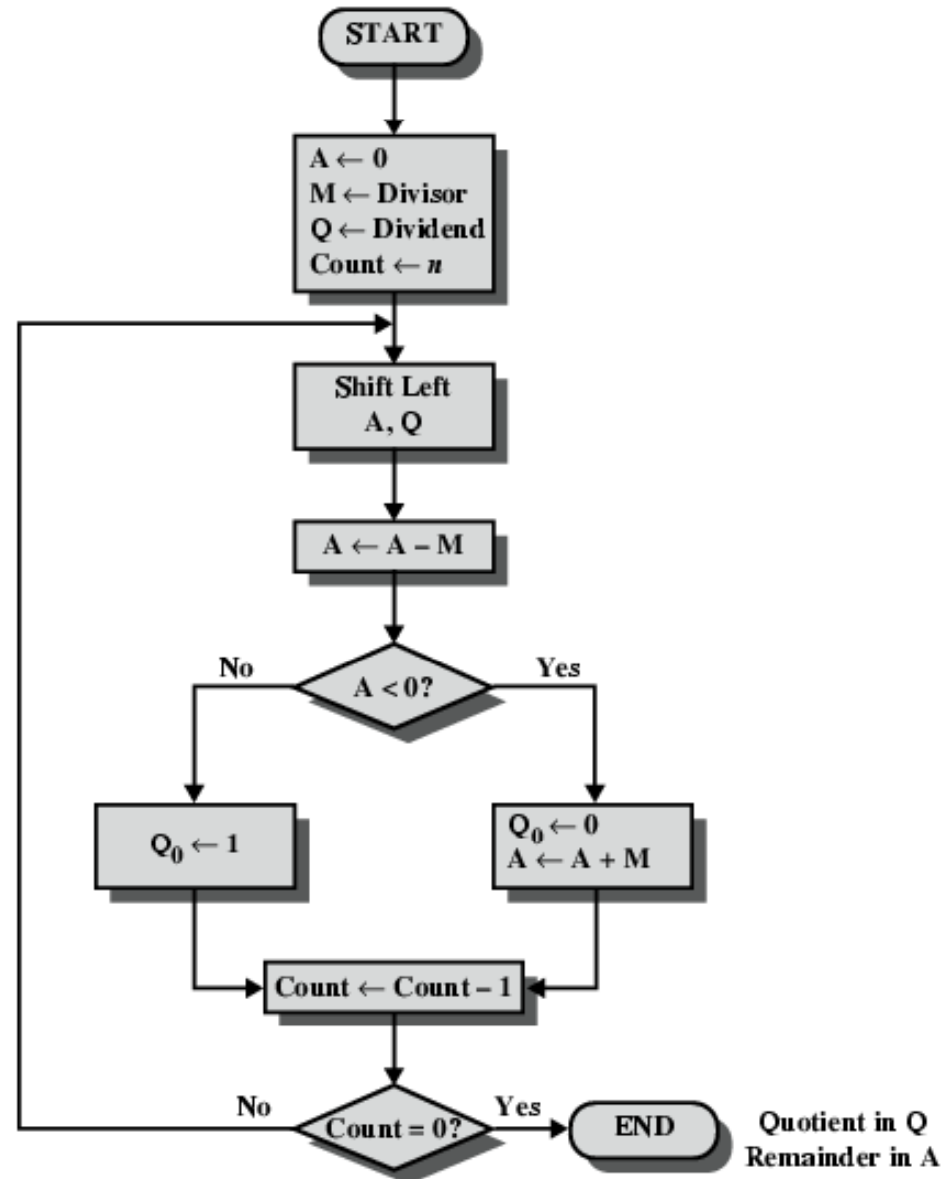
- More complex than multiplication
- Negative numbers are really bad!
- Based on long division

# Division of Unsigned Binary Integers





# Flowchart for Unsigned Binary Division

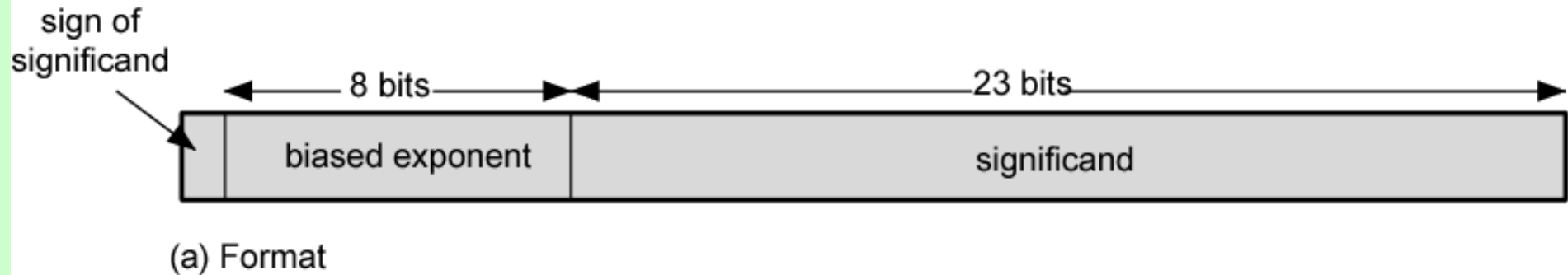


# Real Numbers

---

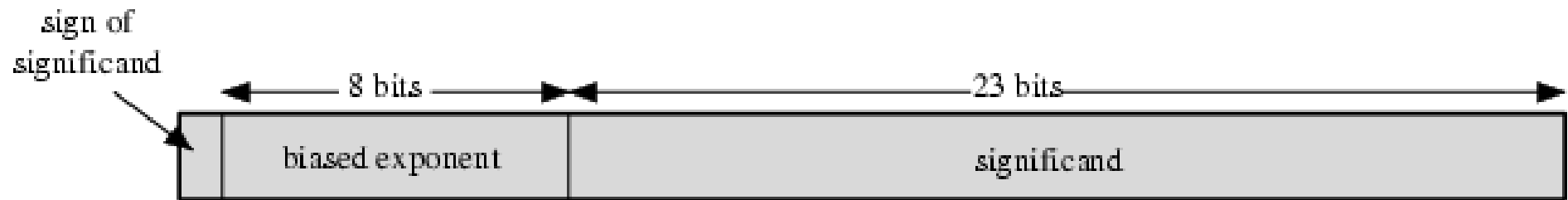
- Numbers with fractions
- Could be done in pure binary
  - $1001.1010 = 2^4 + 2^0 + 2^{-1} + 2^{-3} = 9.625$
- Where is the binary point?
- Fixed?
  - Very limited
- Moving?
  - How do you show where it is?

# Floating Point



- $+/- \text{ .significand} \times 2^{\text{exponent}}$
- Misnomer
- Point is actually fixed between sign bit and body of mantissa
- Exponent indicates place value (point position)

# Floating Point Examples



(a) Format

$$\begin{aligned}
 1.1010001 \times 2^{10100} &= 0\ 10010011\ 101000100000000000000000 &= 1.638125 \times 2^{20} \\
 -1.1010001 \times 2^{10100} &= 1\ 10010011\ 101000100000000000000000 &= -1.638125 \times 2^{20} \\
 1.1010001 \times 2^{-10100} &= 0\ 01101011\ 101000100000000000000000 &= 1.638125 \times 2^{-20} \\
 -1.1010001 \times 2^{-10100} &= 1\ 01101011\ 101000100000000000000000 &= -1.638125 \times 2^{-20}
 \end{aligned}$$

(b) Examples

## Signs for Floating Point

---

- Mantissa is stored in 2s compliment
- Exponent is in excess or biased notation
  - e.g. Excess (bias) 128 means
  - 8 bit exponent field
  - Pure value range 0-255
  - Subtract 128 to get correct value
  - Range -128 to +127

# Normalization

---

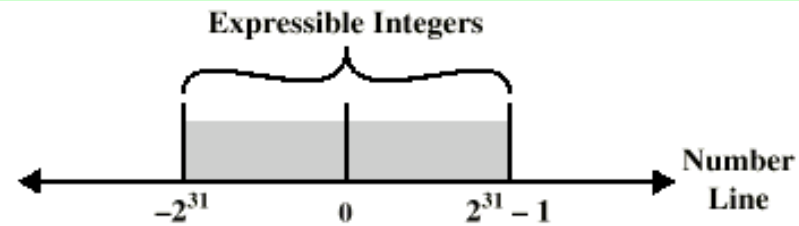
- FP numbers are usually normalized
- i.e. exponent is adjusted so that leading bit (MSB) of mantissa is 1
- Since it is always 1 there is no need to store it
- (c.f. Scientific notation where numbers are normalized to give a single digit before the decimal point
- e.g.  $3.123 \times 10^3$ )

## FP Ranges

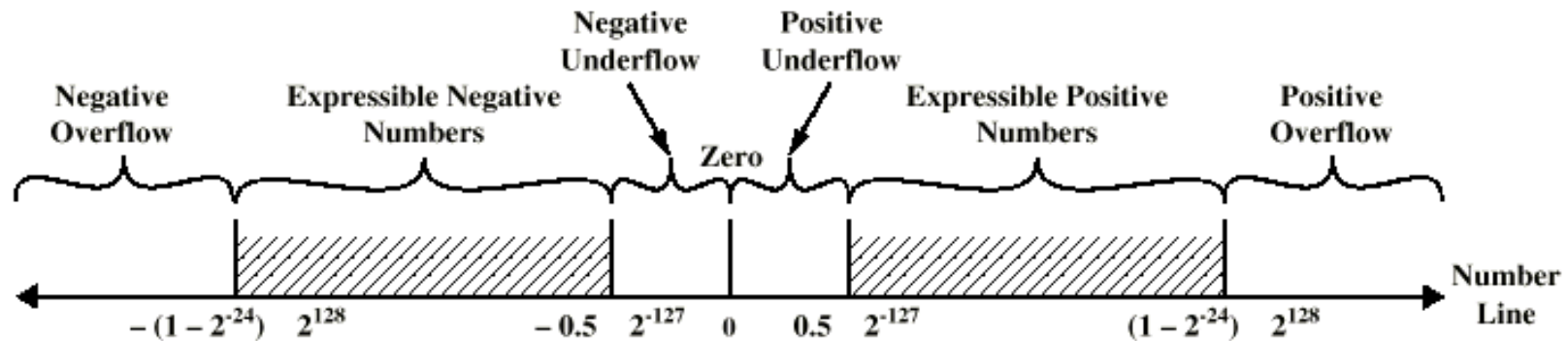
---

- For a 32 bit number
  - 8 bit exponent
  - +/-  $2^{256} \approx 1.5 \times 10^{77}$
- Accuracy
  - The effect of changing lsb of mantissa
  - 23 bit mantissa  $2^{-23} \approx 1.2 \times 10^{-7}$
  - About 6 decimal places

# Expressible Numbers



(a) Two's Complement Integers

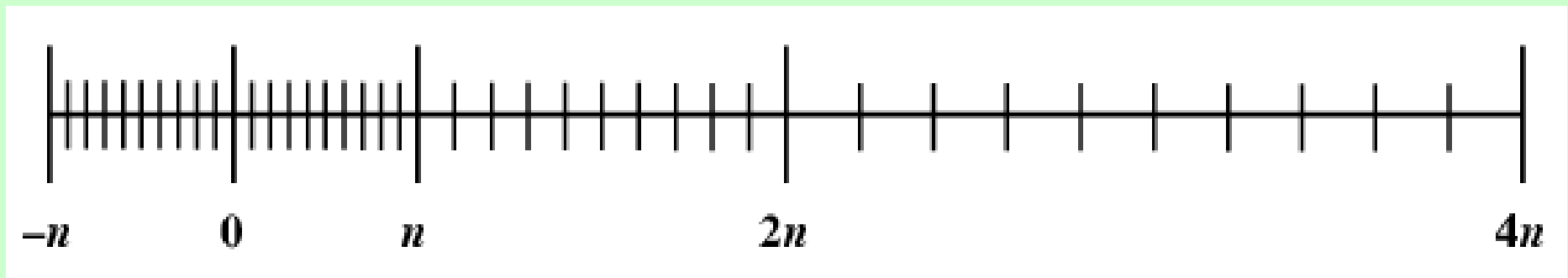


(b) Floating-Point Numbers



# Density of Floating Point Numbers

---

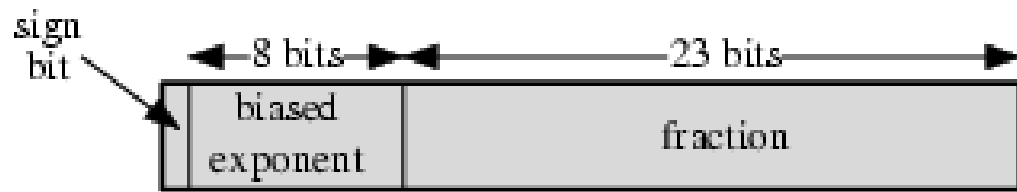


## IEEE 754

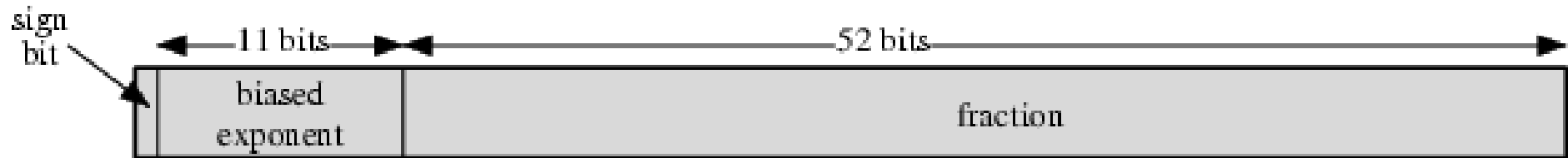
---

- Standard for floating point storage
- 32 and 64 bit standards
- 8 and 11 bit exponent respectively
- Extended formats (both mantissa and exponent) for intermediate results

# IEEE 754 Formats



(a) Single format



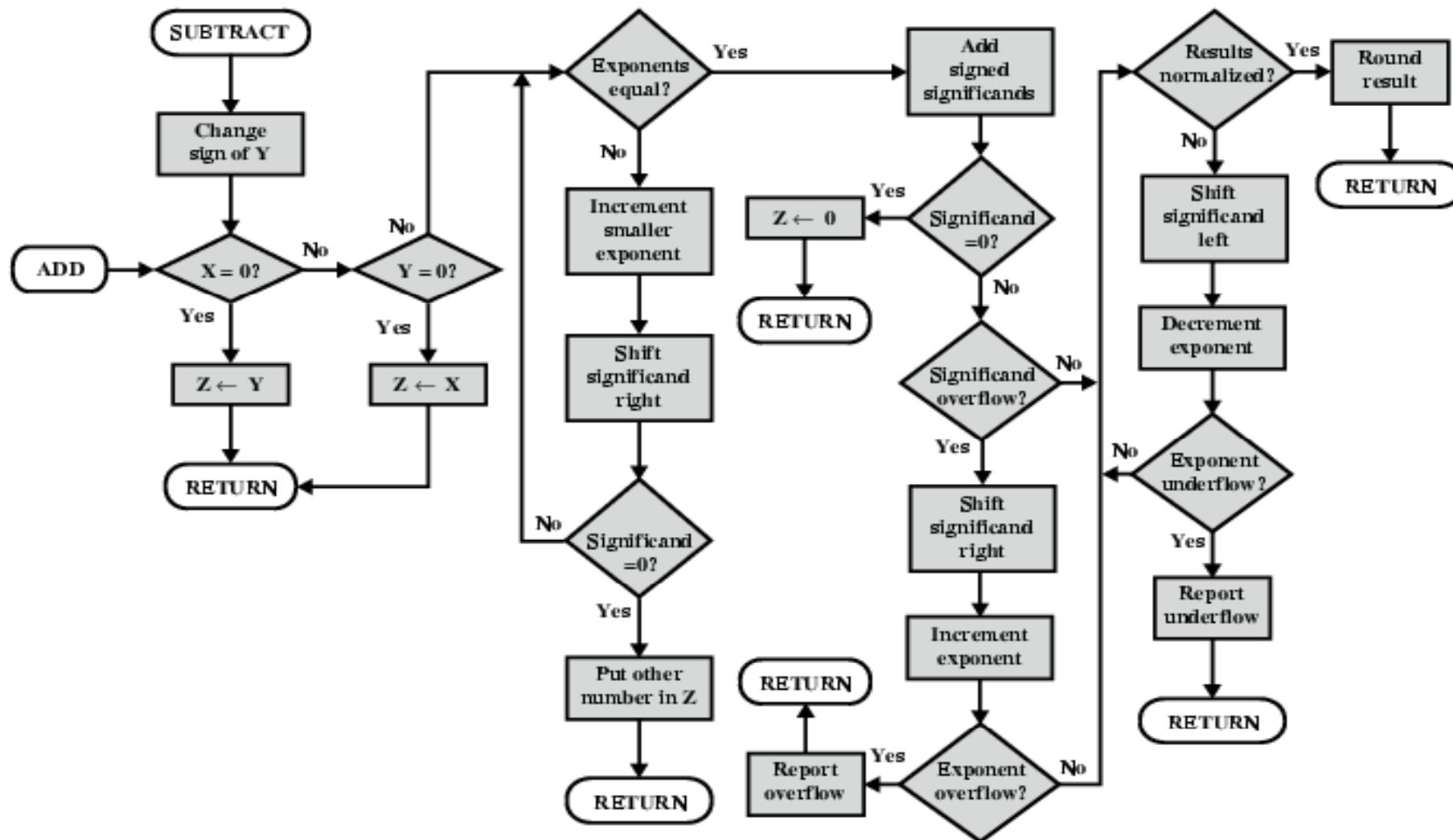
(b) Double format

## FP Arithmetic +/-

---

- Check for zeros
- Align significands (adjusting exponents)
- Add or subtract significands
- Normalize result

# FP Addition & Subtraction Flowchart

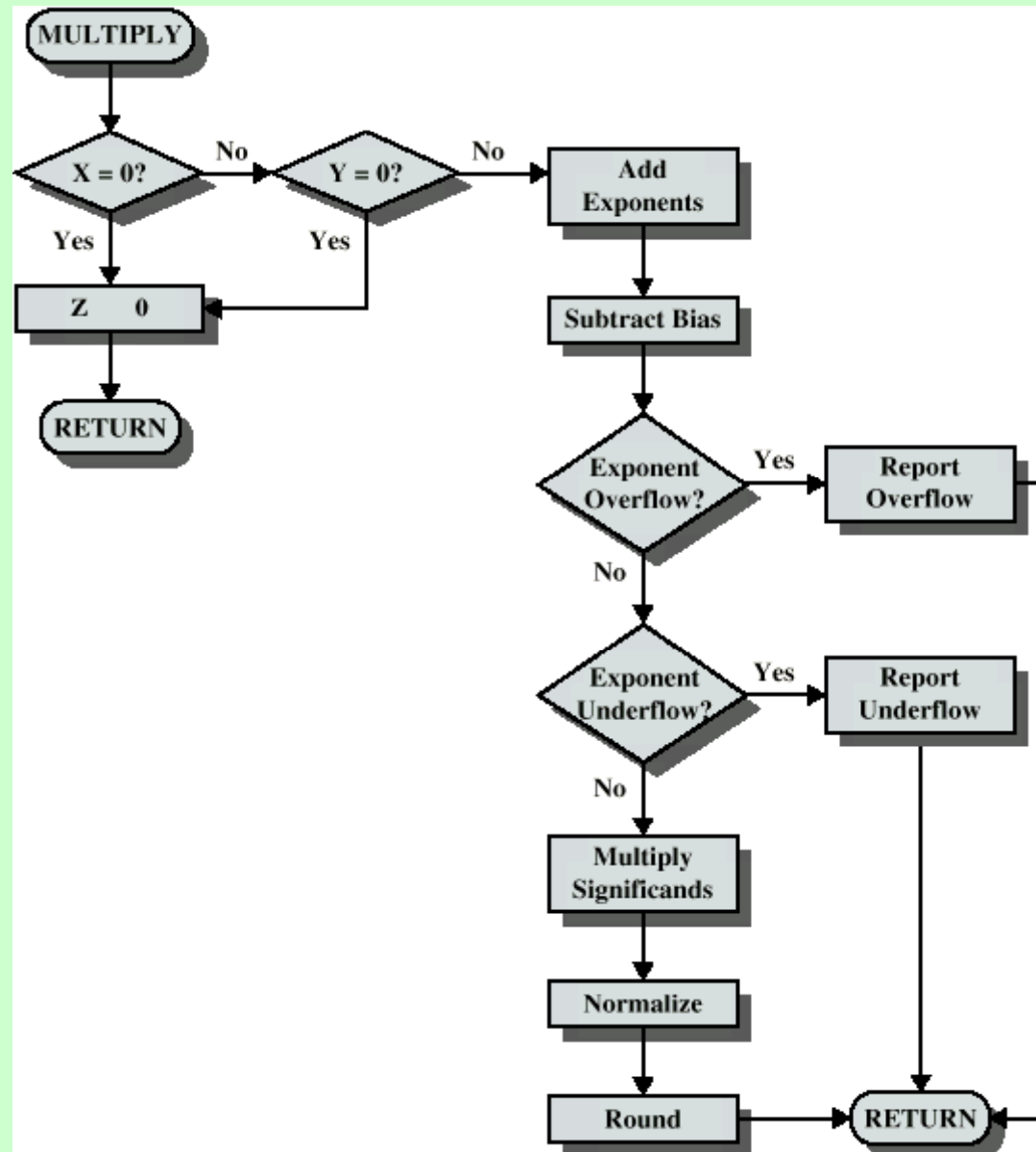


## FP Arithmetic $\times/\div$

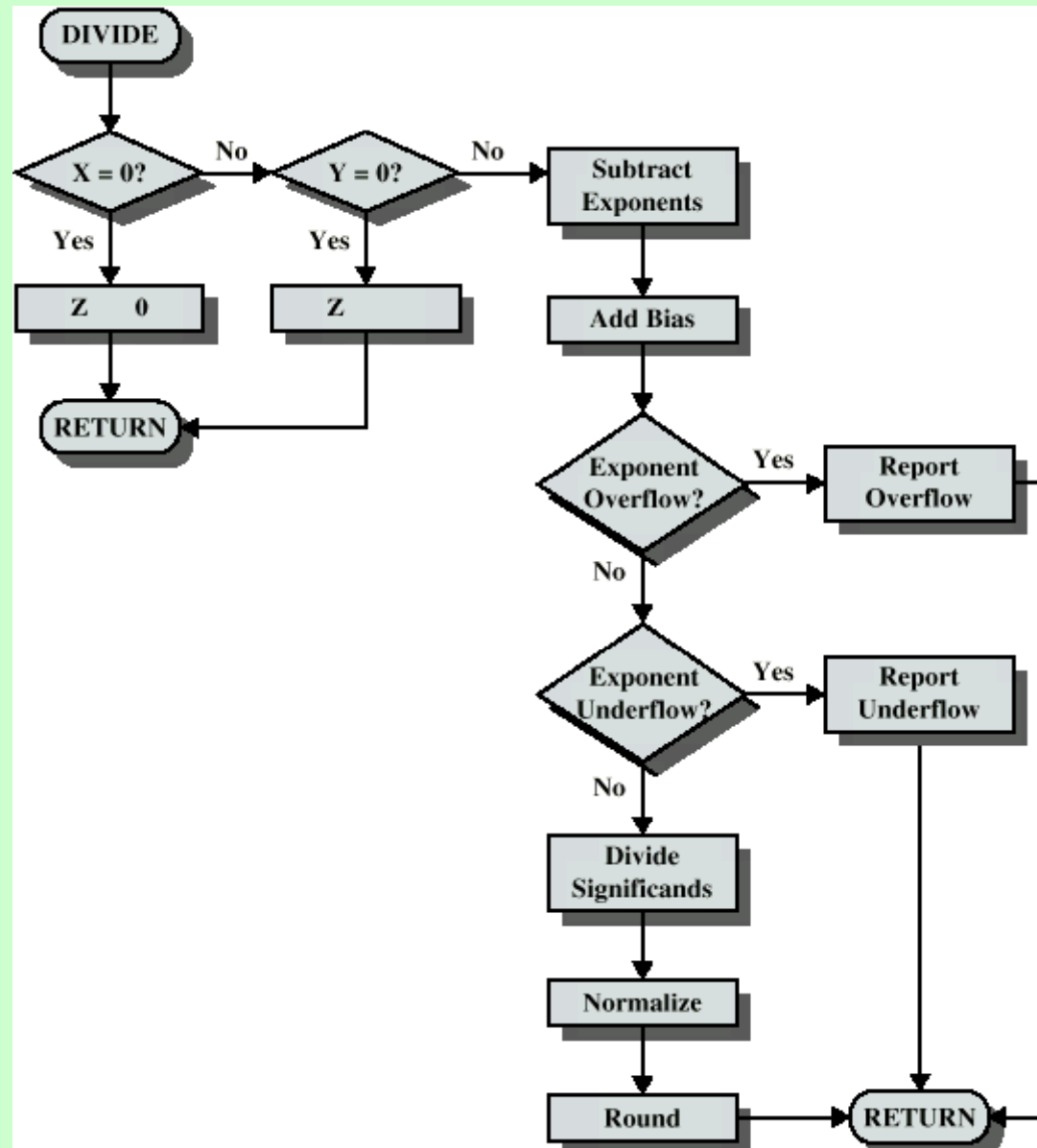
---

- Check for zero
- Add/subtract exponents
- Multiply/divide significands (watch sign)
- Normalize
- Round
- All intermediate results should be in double length storage

# Floating Point Multiplication



# Floating Point Division





## Required Reading

---

- Stallings Chapter 9
- IEEE 754 on IEEE Web site