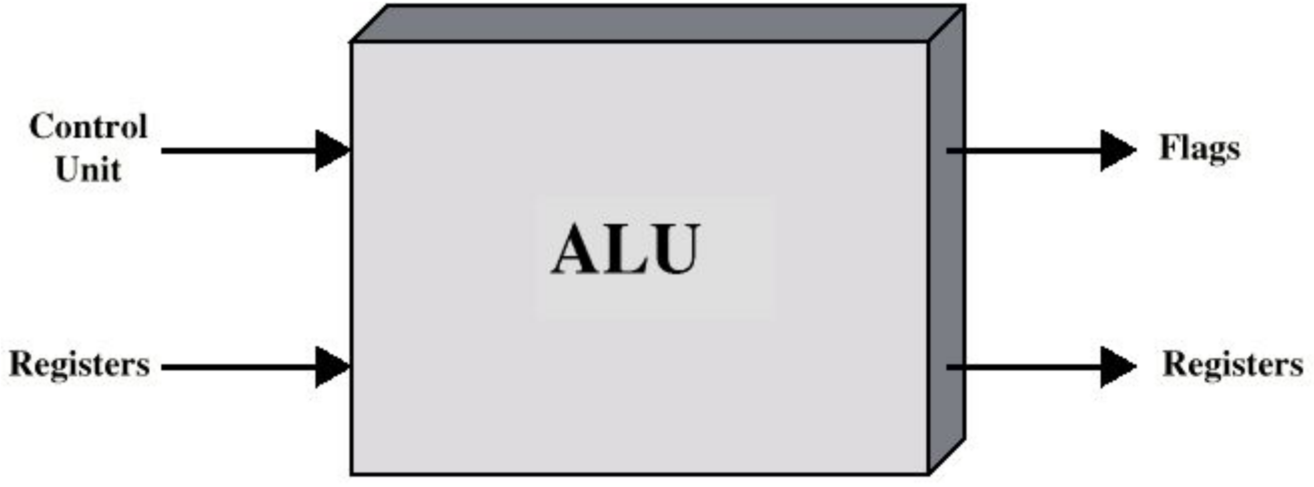
Computer Architecture

Lecture 8
Computer Arithmetic

Arithmetic & Logic Unit (ALU)

- ALU is a part of the computer that actually performs arithmetic and logical operation on the data.
- All of other elements of the computer system-control unit, registers, memory, I/O- are there mainly to bring data into ALU for it to process and then to take the results back out.
- Handles integers
- May handle floating point (real) numbers
- May be separate maths co-processor

ALU Inputs and Outputs



- ALU is interconnected with the processor
- Data are presented to the ALU in registers, and the results of an operation are stored in registers
- These registers are temporary storage locations within the processor
- The ALU may also set flags as the result of operation (e.g., an overflow flag is set to 1 if the result of computation exceeds the length of the register into which it is to be stored).
- The control unit provides signals that controls the operation of the ALU and the movement of the data into and out of the ALU

Integer Representation

- In the binary number system, arbitrary numbers can be represented with only 0 & 1, the minus sign, and the period (radix point)
- For purpose of computer storage and processing, we do not have benefit of minus sign and periods
- If we are limited to nonnegative integers, the representation is straightforward.
- Positive numbers stored in binary
  - e.g. 41=00101001

Sign-Magnitude

- **Left most bit is sign bit**
- **0 means positive**
- **1 means negative**
- +18 = 00010010
- -18 = 10010010 (sign magnitude)
- Problems to sign magnitude representation
  - Need to consider both sign and magnitude in arithmetic
  - Two representations of zero (+0 and -0)
  - +0 = 00000000, -0 = 10000000 (sign magnitude)

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

Two's Compliment

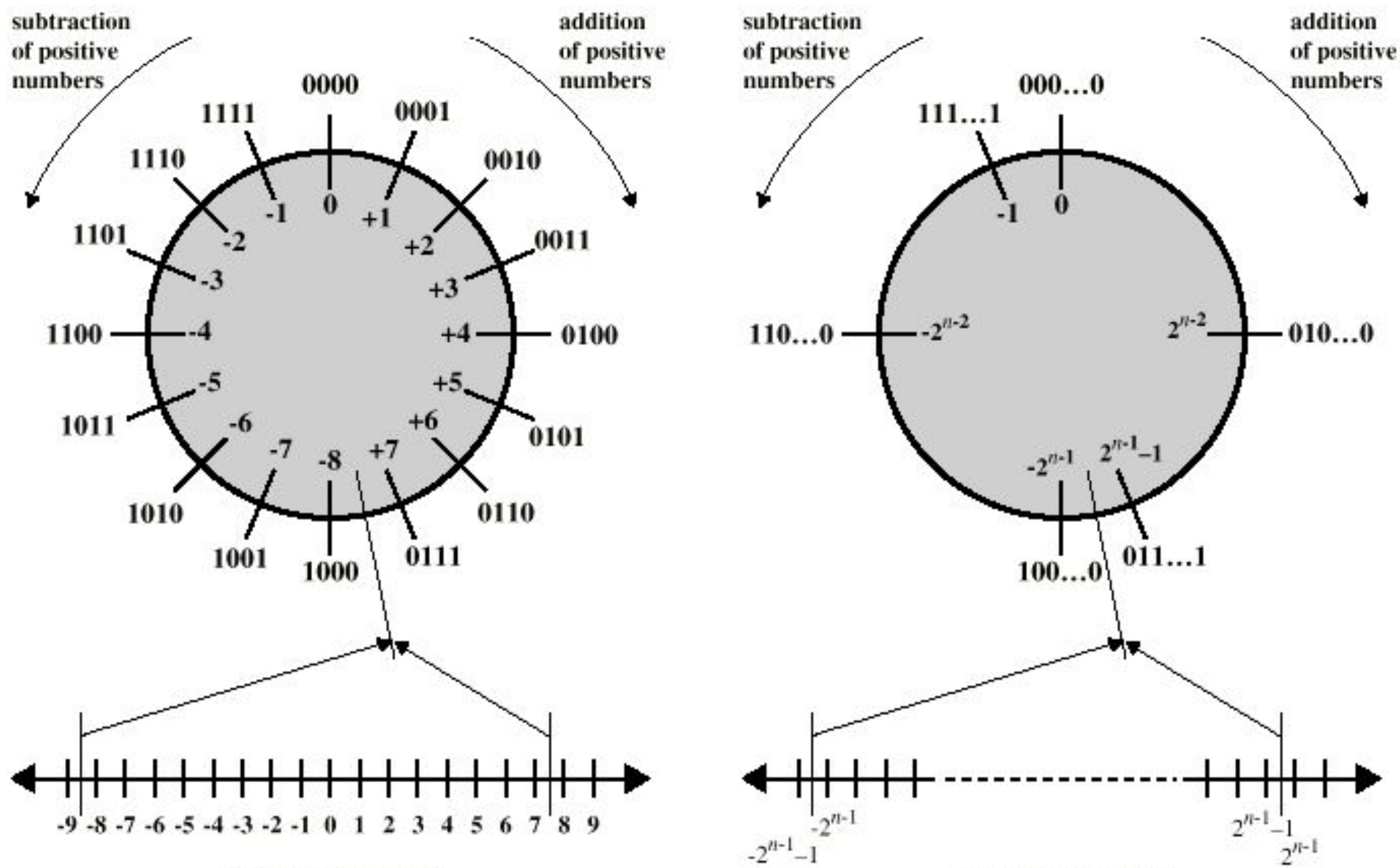
- Like sign magnitude, twos complement representation uses the most significant bit as a sign bit (whether the integer is positive or negative)
- But representation is different (Table 9.2, Page 280)
- +3 = 00000011
- +2 = 00000010
- +1 = 00000001
- +0 = 00000000
- -1 = 11111111
- -2 = 11111110
- -3 = 11111101

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

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



(a) 4-bit numbers

(b) n-bit numbers

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 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 = 011111111 11111111 = 2^{15} - 1
  - -32768 = 100000000 00000000 = -2^{15}

Conversion Between Lengths

–128	64	32	16	8	4	2	1

(a) An eight-position two's complement value box

–128	64	32	16	8	4	2	1	
1	0	0	0	0	0	1	1	
–128						+2	+1	= –125

(b) Convert binary 10000011 to decimal

	–128	64	32	16	8	4	2	1
	1	0	0	0	1	0	0	0
–120 =	–128				+8			

(c) Convert decimal –120 to binary

Figure 9.2 Use of a Value Box for Conversion Between Twos Complement Binary and Decimal

Conversion Between Lengths

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

Addition and Subtraction

- Normal binary addition
- Carry bit beyond the end of word (shading), which is ignored
- Monitor sign bit for overflow

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

Figure 9.3 Addition of Numbers in Twos Complement Representation

Addition and Subtraction

- Subtraction rule: Take twos compliment of subtrahend and add to minuend
  - i.e. a - b = a + (-b)

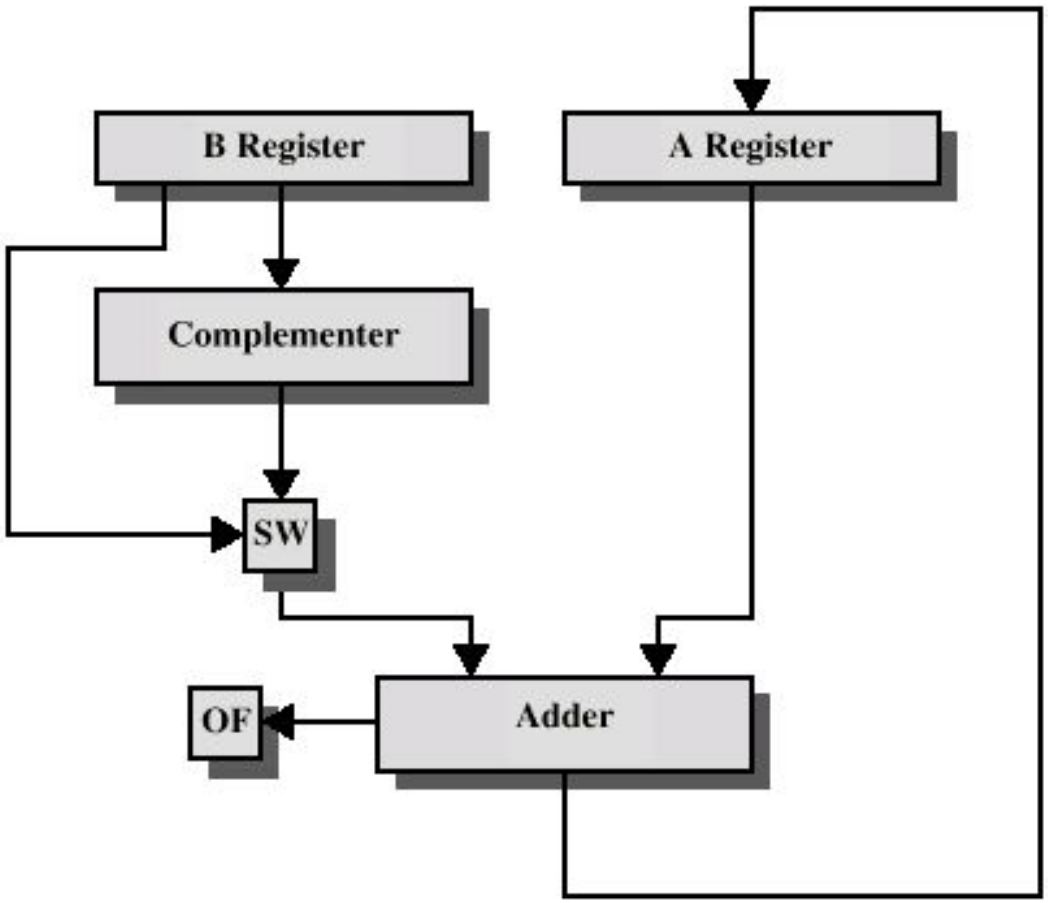
- So, we only need addition and complement circuits

Addition and Subtraction

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

Figure 9.4 Subtraction of Numbers in Twos Complement Representation (M – S)

Hardware for Addition and Subtraction



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

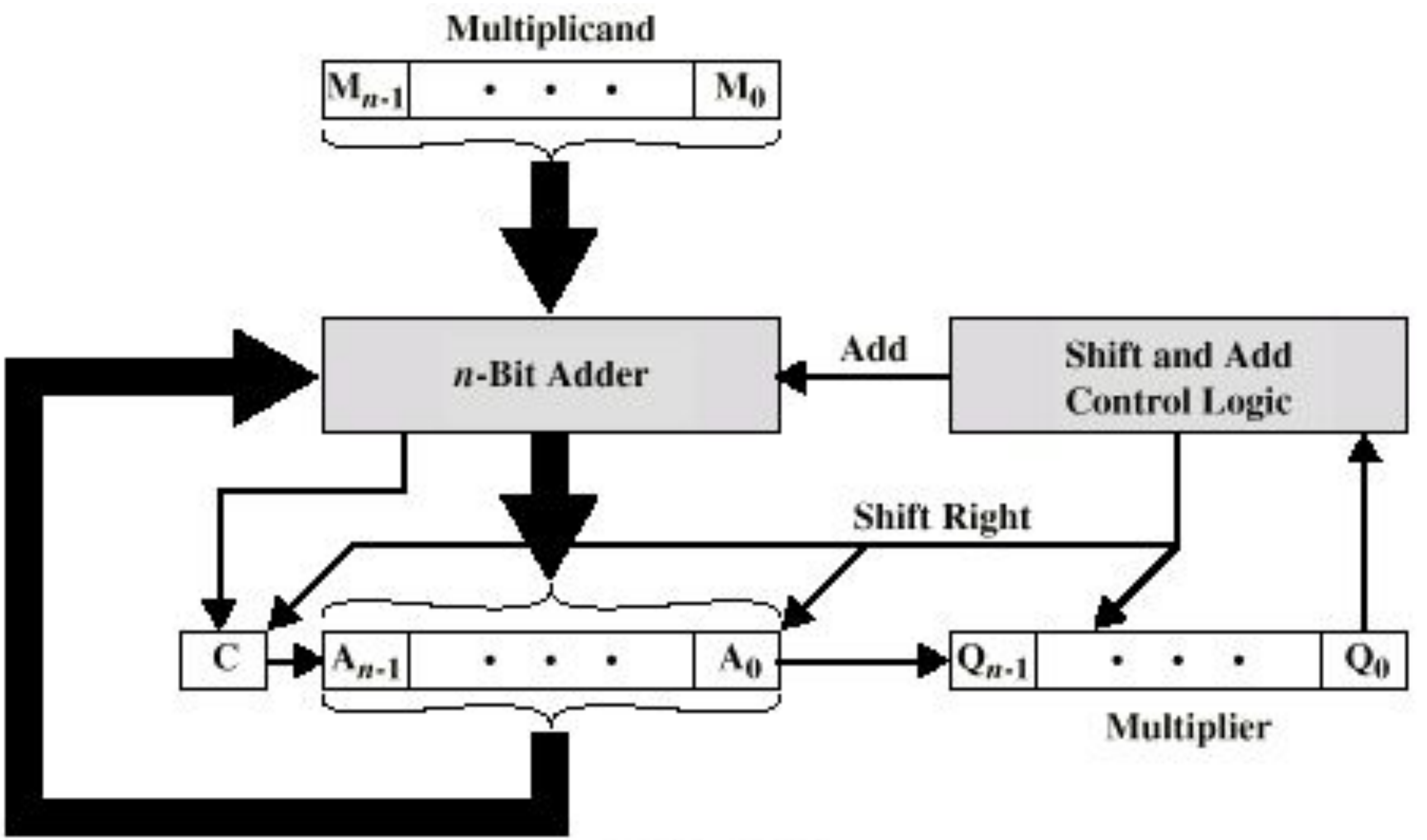
Multiplication

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

Multiplication Example

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

Unsigned Binary Multiplication

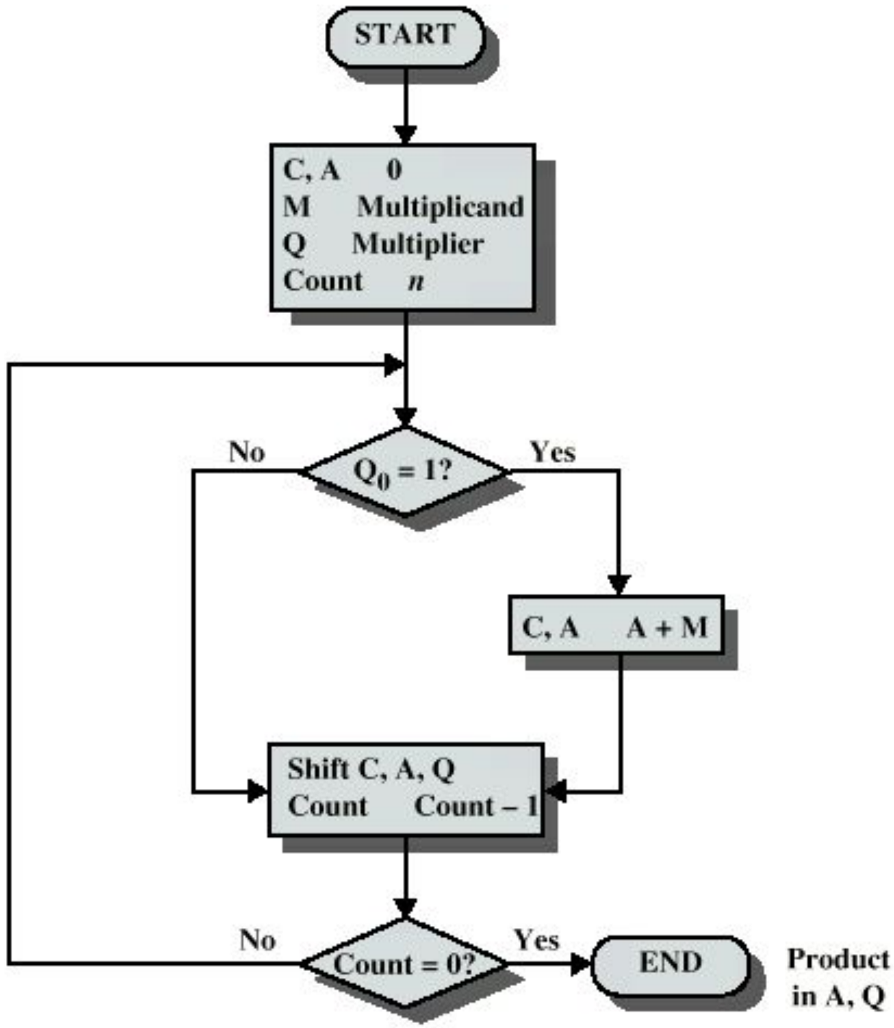


(a) Block Diagram

Execution of Example

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

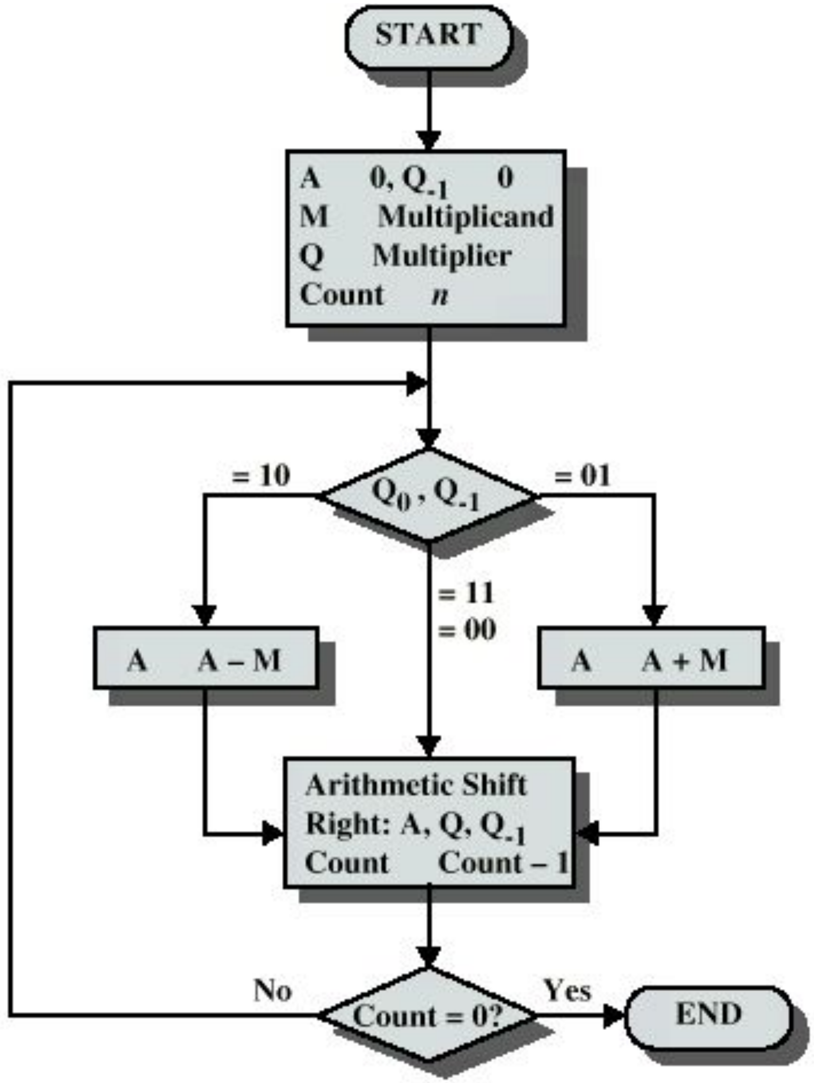
Flowchart for Unsigned Binary Multiplication



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



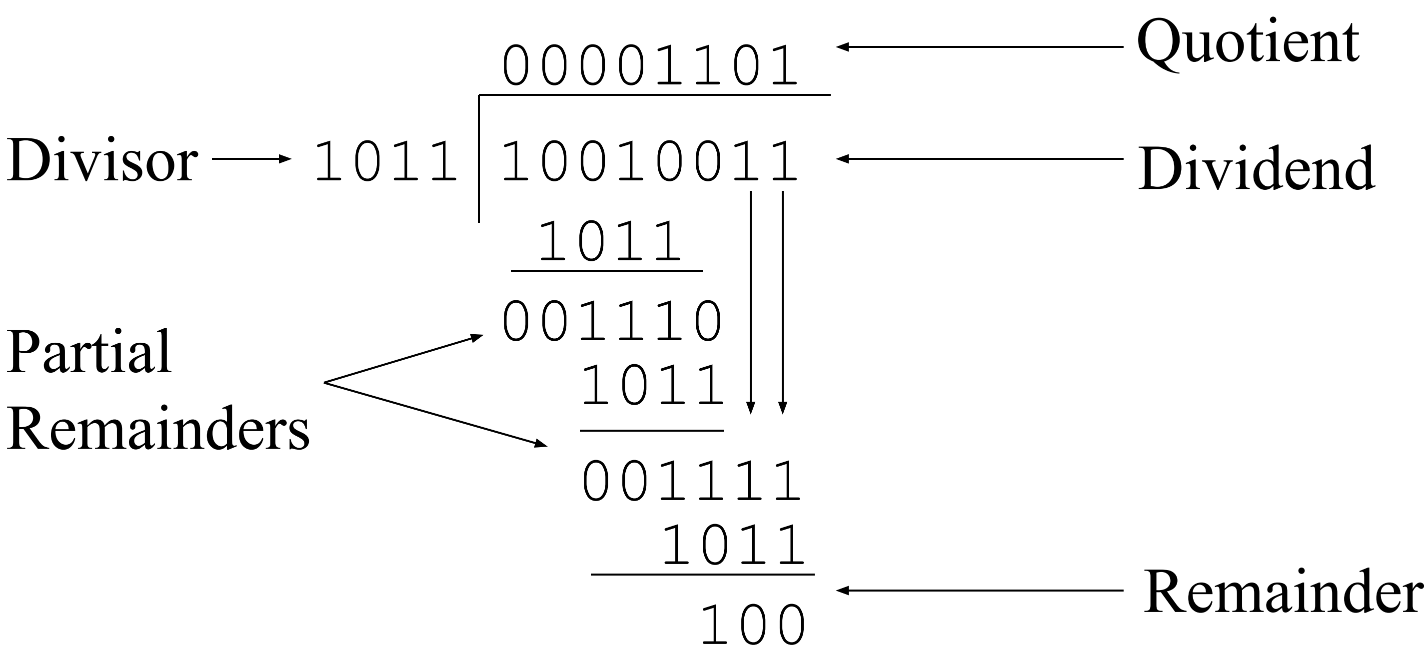
Example of Booth's Algorithm

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

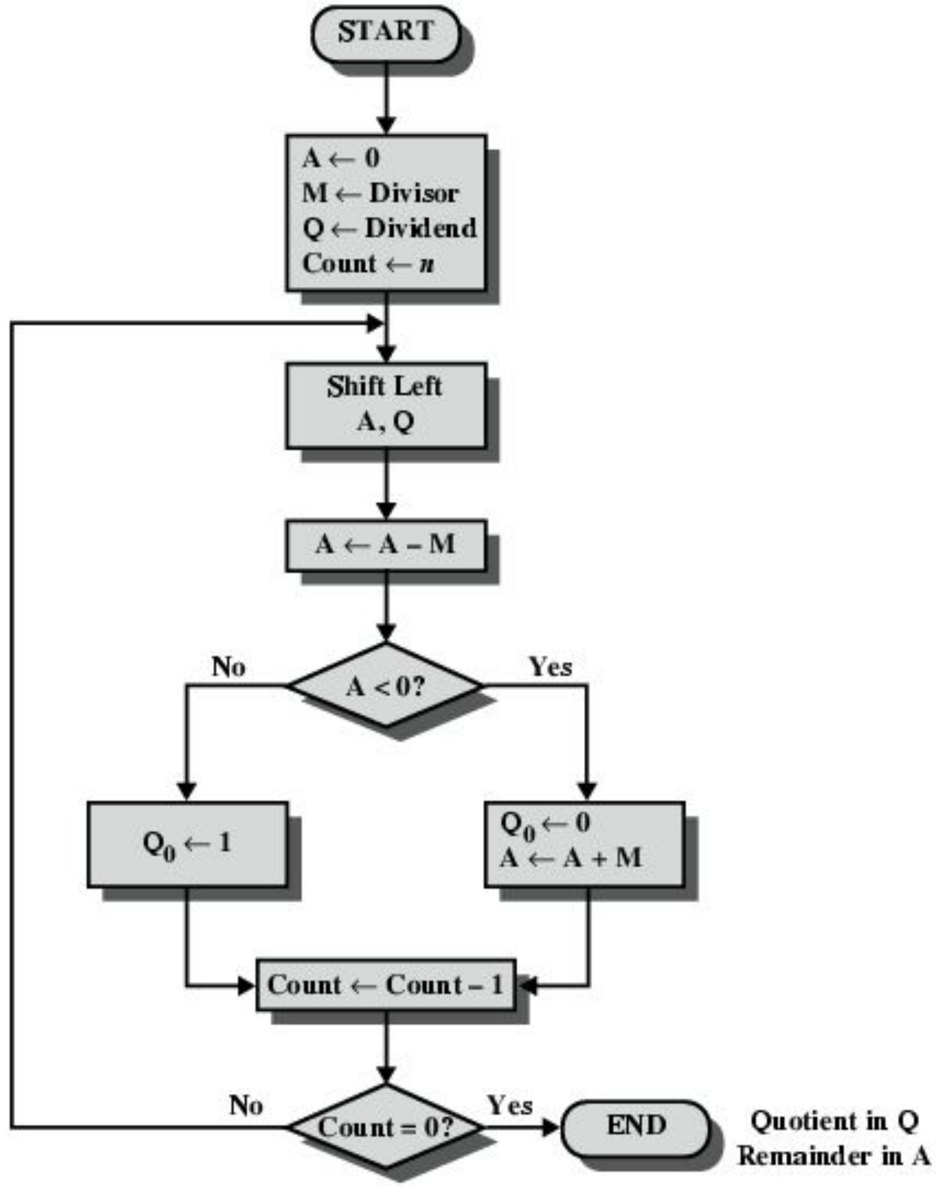
Division

- 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
  - $1001.1010 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9.625$
- Where is the binary point?
- Fixed?
  - Very limited
- Moving?
  - How do you show where it is?

Floating Point

(a) Format

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

Floating Point Examples

(a) Format

$$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.638125 \times 2^{20}$$
$$-1.1010001 \times 2^{10100} = 1\ 10010011\ 10100010000000000000000 = -1.638125 \times 2^{20}$$
$$1.1010001 \times 2^{-10100} = 0\ 01101011\ 10100010000000000000000 = 1.638125 \times 2^{-20}$$
$$-1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010000000000000000 = -1.638125 \times 2^{-20}$$

(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

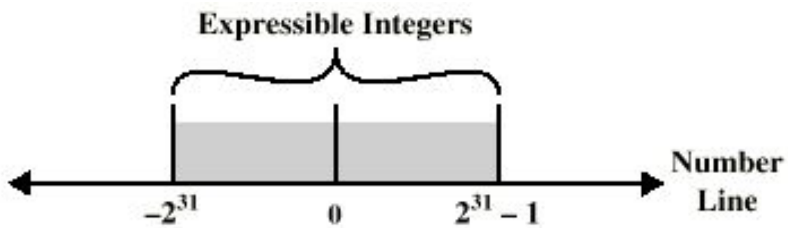
Normalization

- 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. 3.123×10^3)

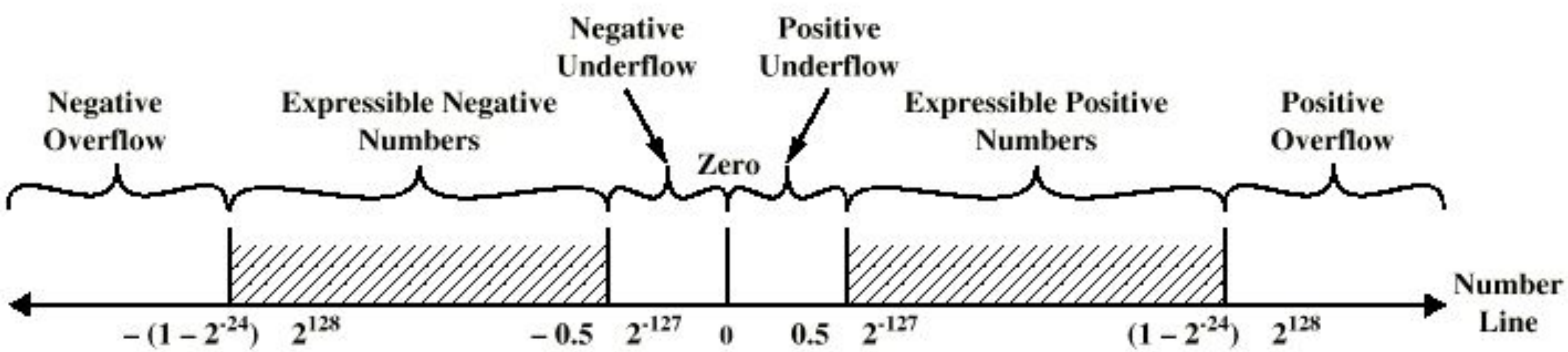
FP Ranges

- 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

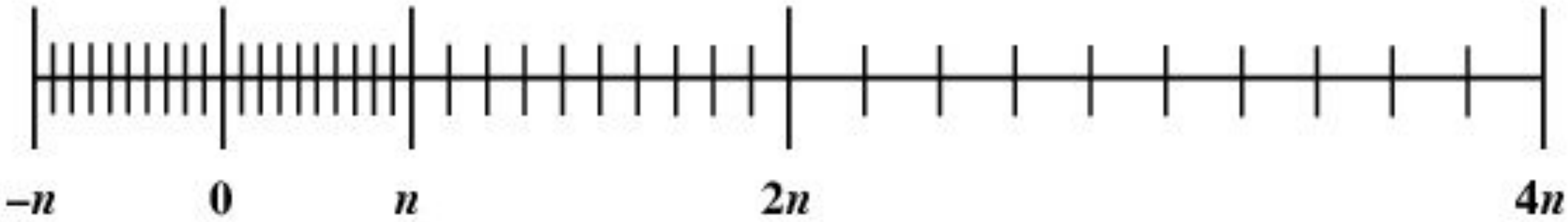


(a) Twos Complement Integers



(b) Floating-Point Numbers

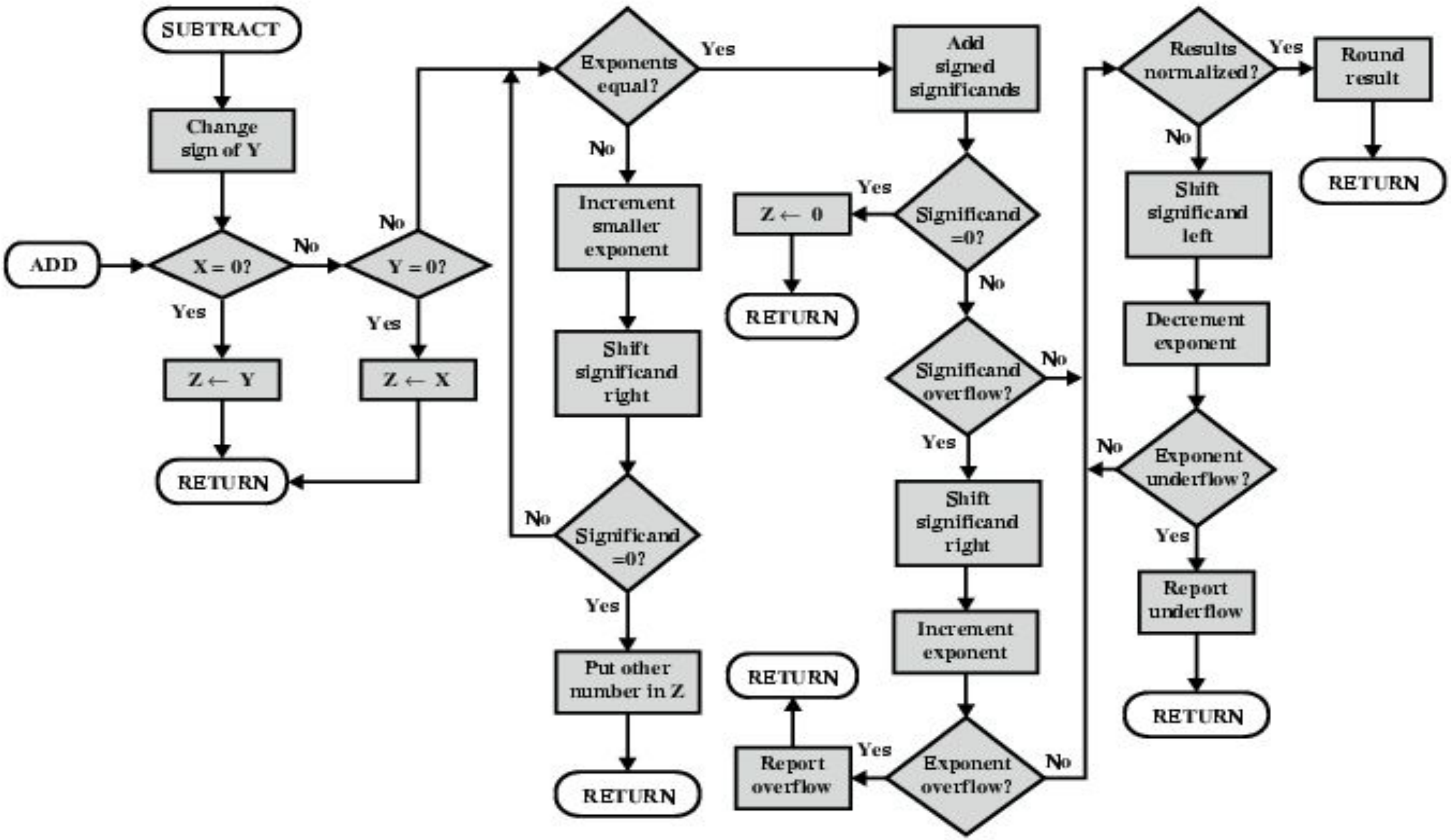
Density of Floating Point Numbers



FP Arithmetic +/-

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

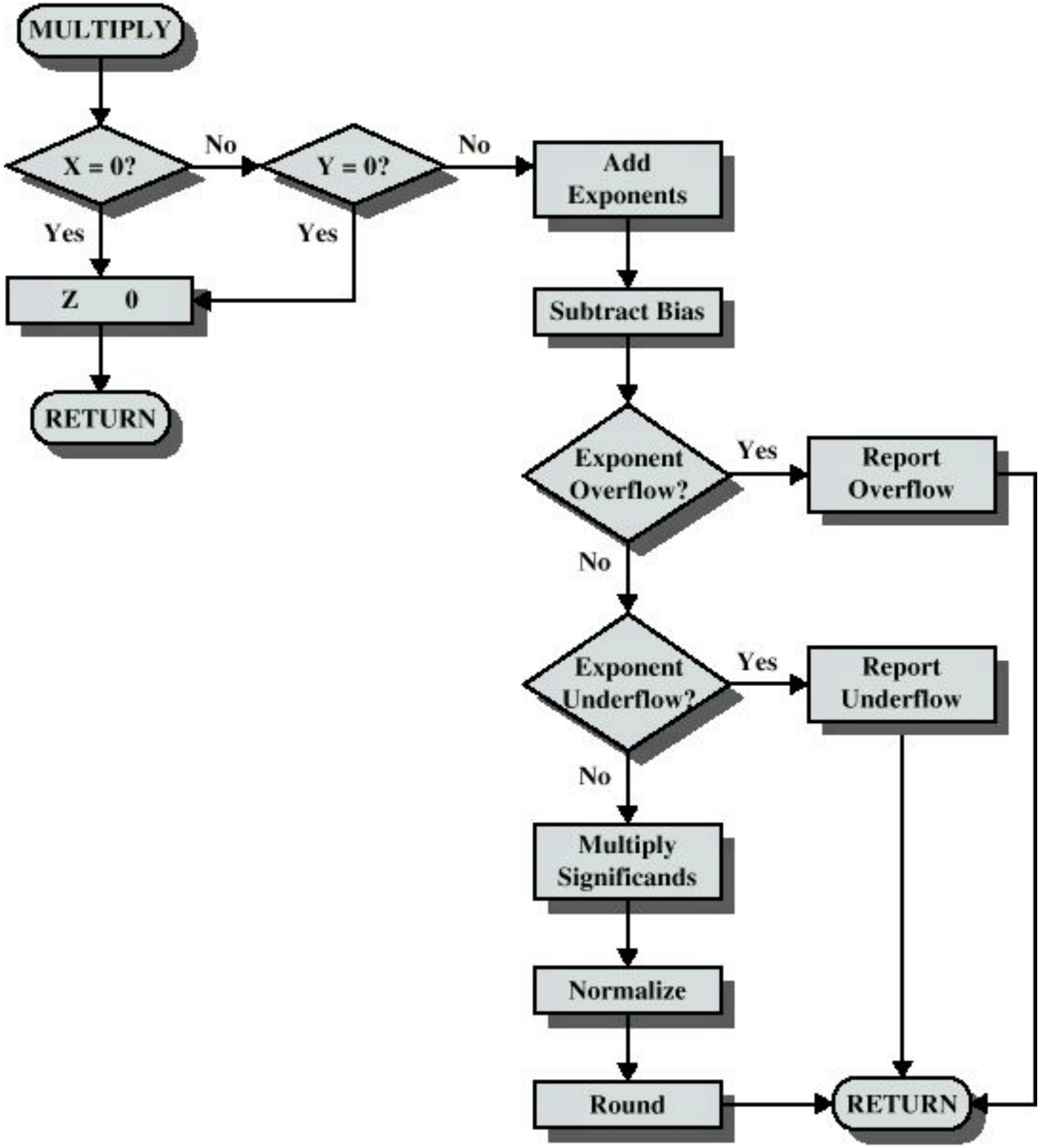
FP Addition & Subtraction Flowchart



FP Arithmetic x/÷

- 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