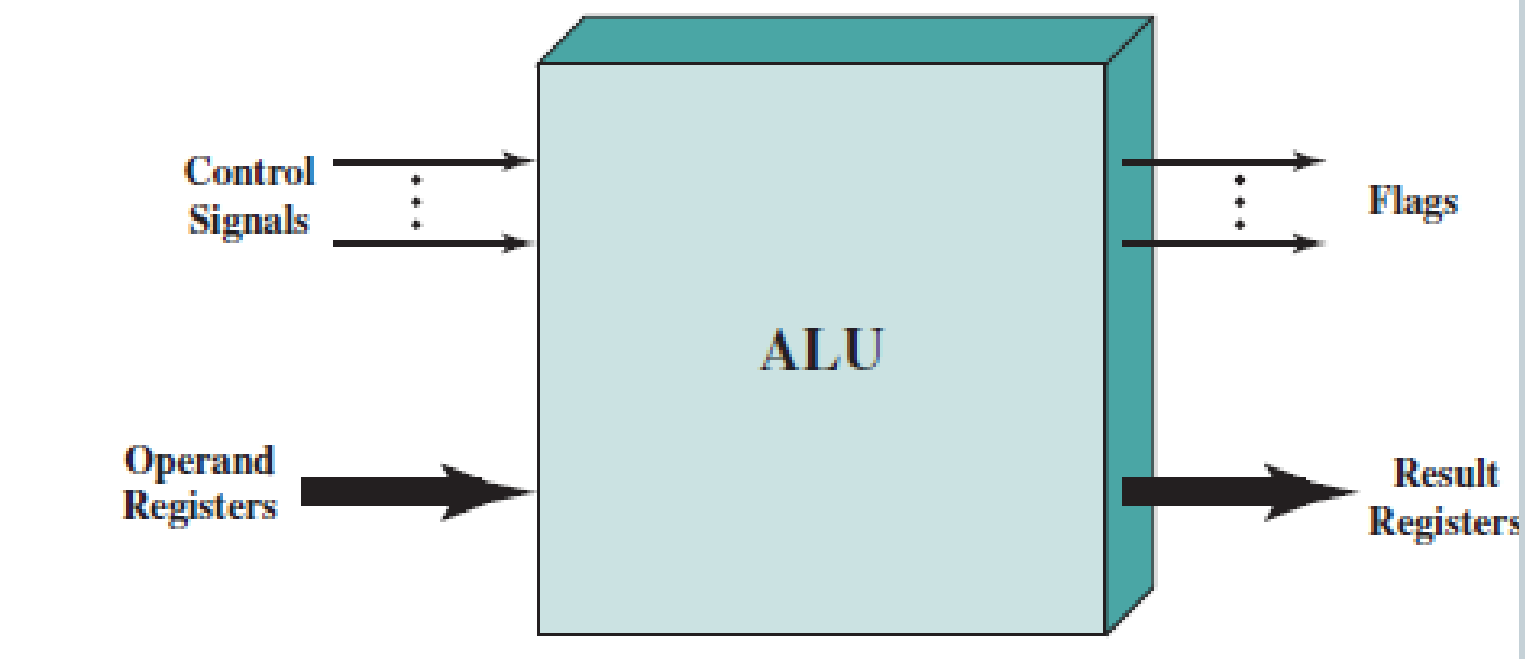
Chapter 2: Computer Arithmetic

COMPUTER ARCHITECTURE AND ORGANIZATION (CSEG3202)

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

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

$$-1101.0101_2 = -13.3125_{10}$$

Sign-Magnitude Representation

- 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)

Twos Complement Representation

- Like sign magnitude, twos complement representation uses the most significant bit as a sign bit,
- Easy to test whether an integer is positive or negative.
- Differs from the use of the sign-magnitude representation in the way that the other bits are interpreted
- +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

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

Main Features

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*.

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

- Positive number pack with leading zeros
- +18 = 00010010
- +18 = 00000000 00010010
- Negative numbers pack with leading ones
- -18 = 11101110
- -18 = 11111111 11101110
- i.e. pack with MSB (sign bit)

Addition and Subtraction

- Normal binary addition
- Monitor sign bit for overflow
- Take twos compliment of substahend and add to minuend
  - i.e. a - b = a + (-b)
- So we only need addition and complement circuits

Addition and Overflow rule

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)

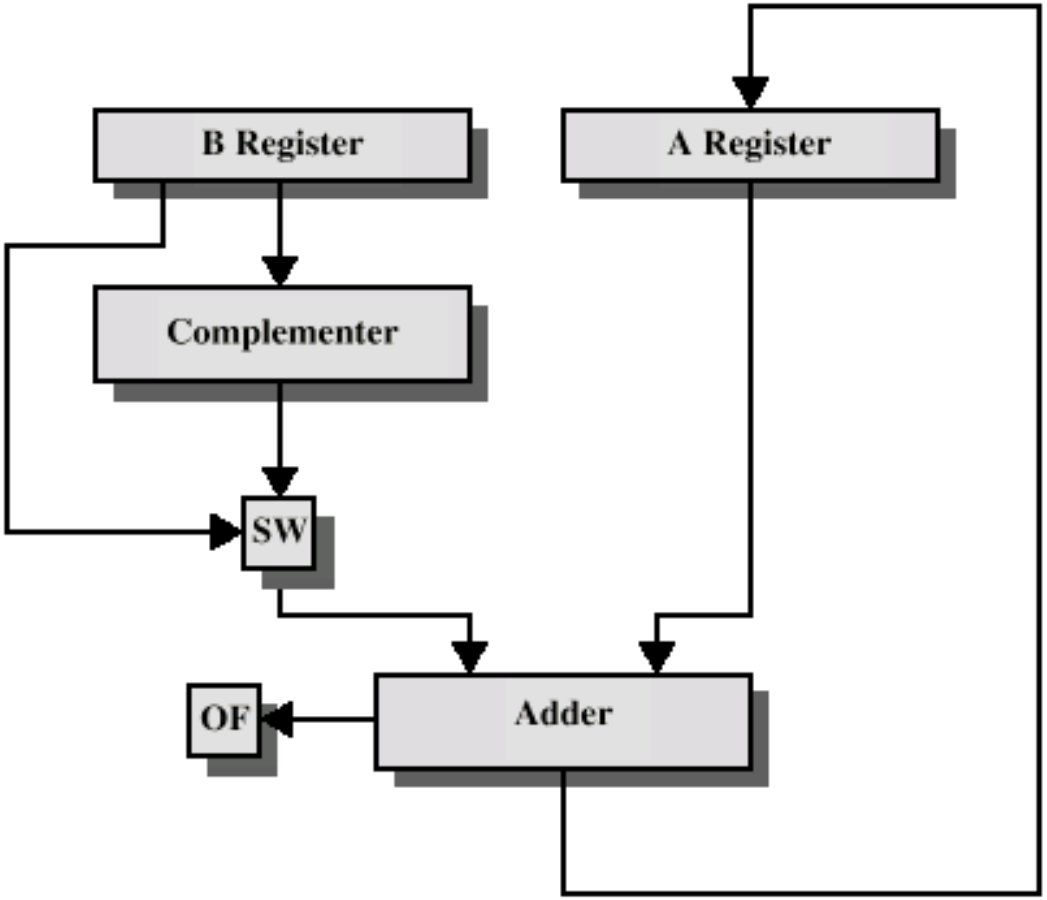
If two numbers are added, and they are both positive or both negative, then overflow occurs if and only if the result has the opposite sign.

Subtraction and overflow

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 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

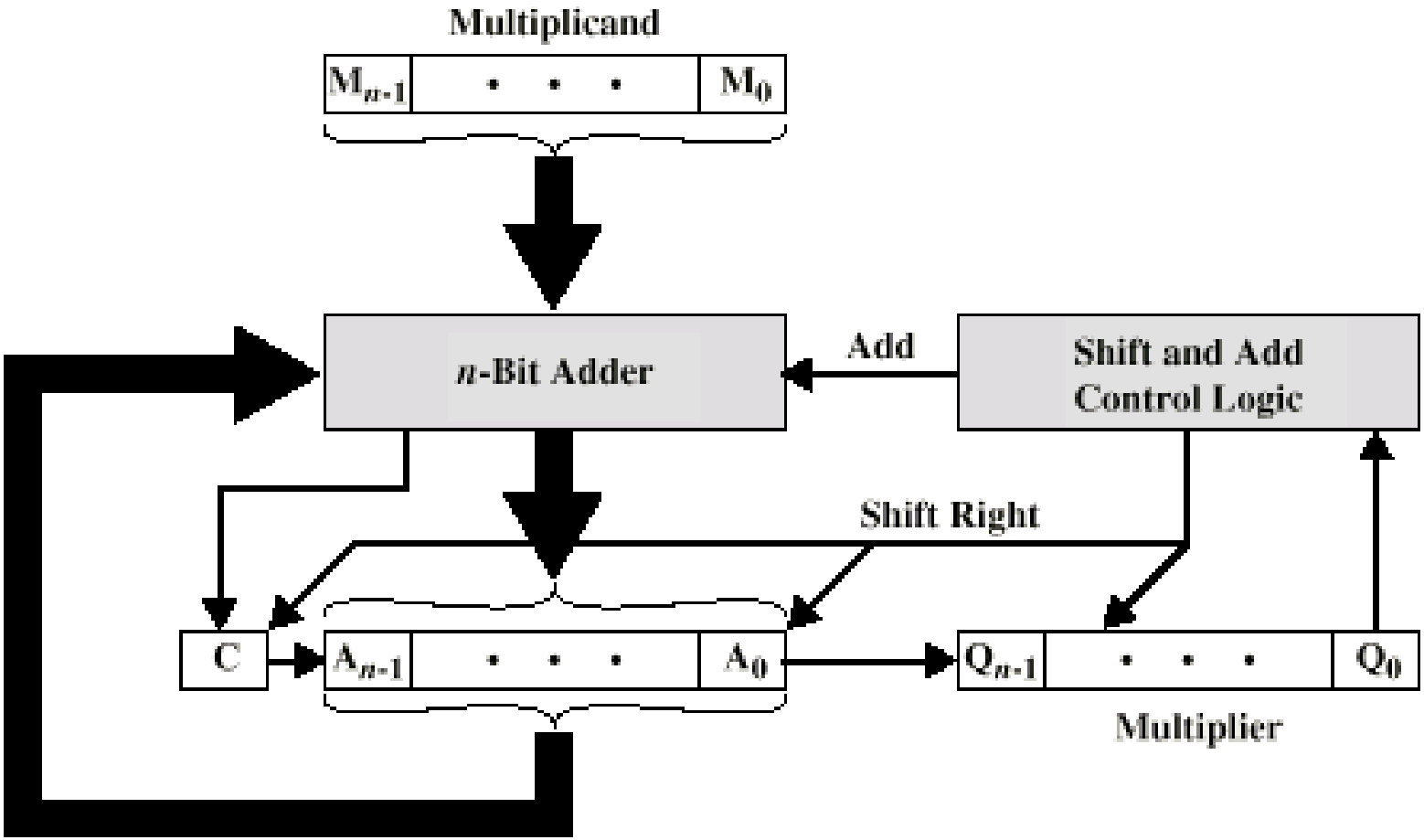
- Complex
- Work out partial product for each digit
- Take care with place value (column)
- Add partial products

Unsigned Multiplication 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	

M – multiplicand Q – multiplier C –Carry A – Partial Accumulator

Unsigned Binary Multiplication



(a) Block Diagram

Multiplying negative numbers

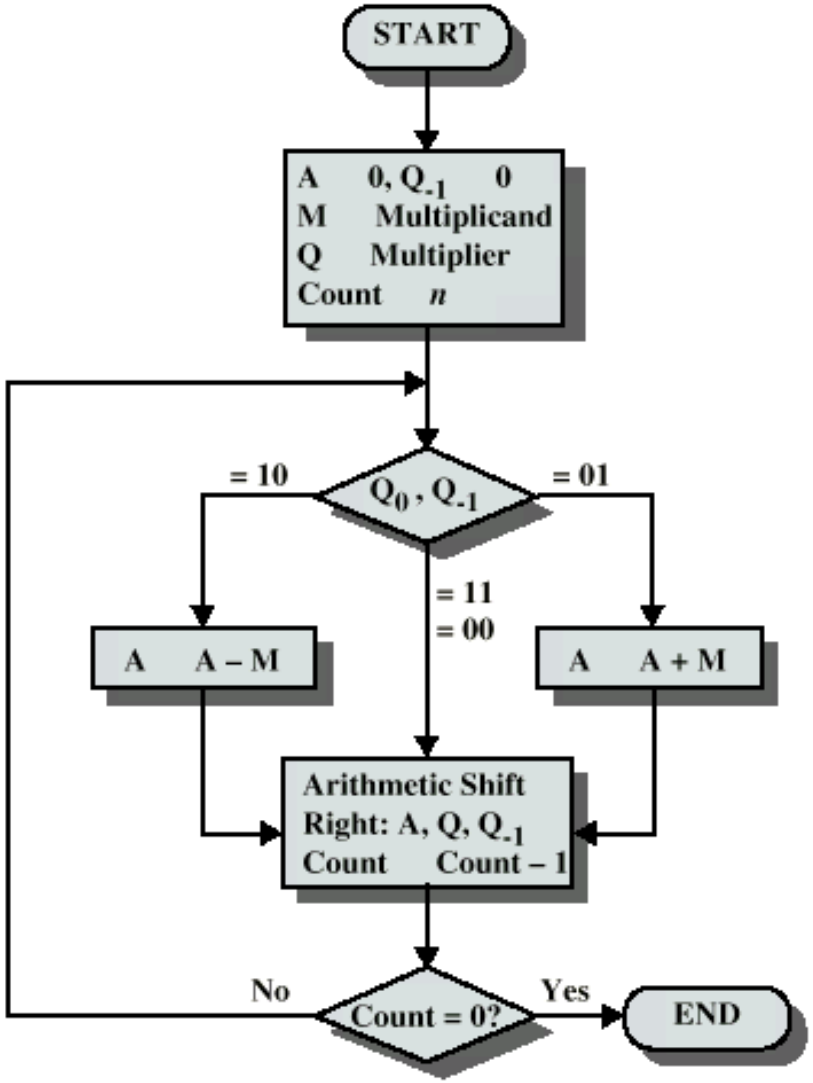
1001 (9) ×0011 (3) 00001001 1001×2^0 00010010 1001×2^1 00011011 (27)	1001 (−7) ×0011 (3) 11111001 $(-7) \times 2^0 = (-7)$ 11110010 $(-7) \times 2^1 = (-14)$ 11101011 (−21)
(a) Unsigned integers	(b) Twos complement integers

Figure 9.11 Comparison of Multiplication of Unsigned and Twos Complement Integers

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

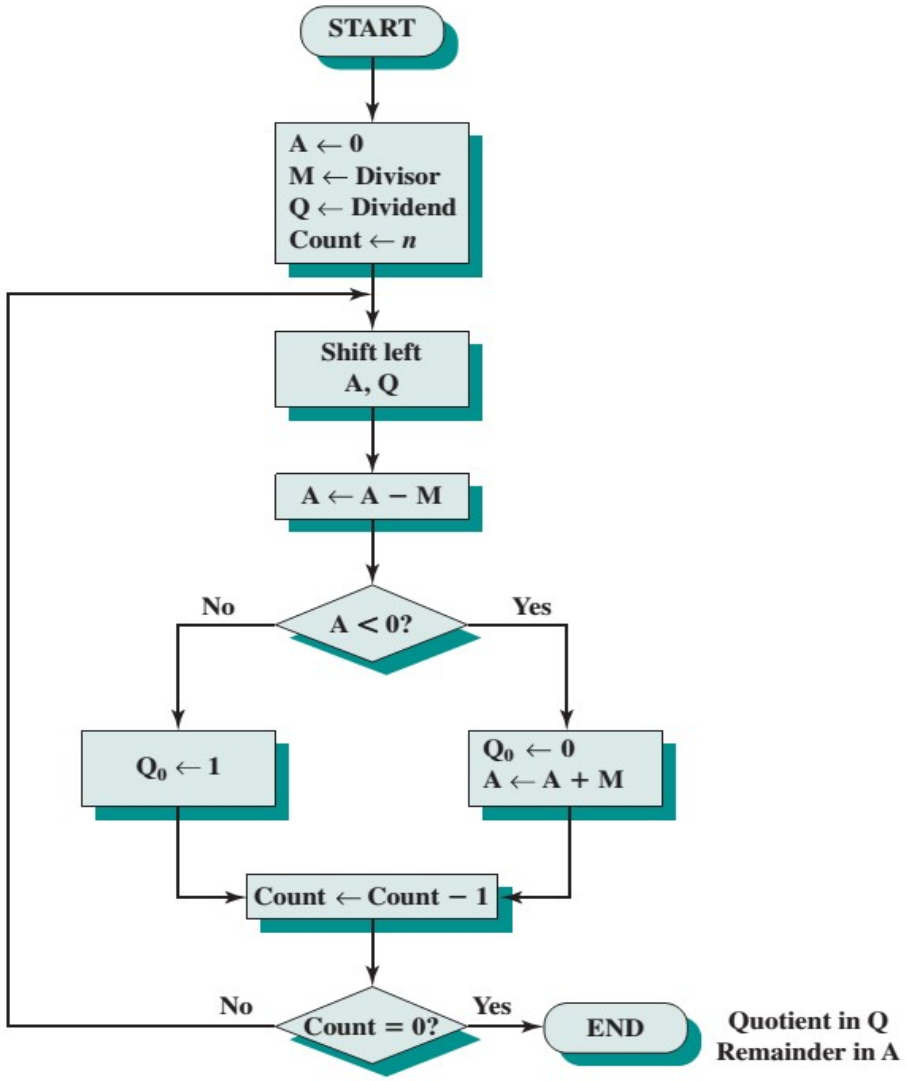
- More complex than multiplication
- Negative numbers are really bad!

Division of Unsigned Binary Integers

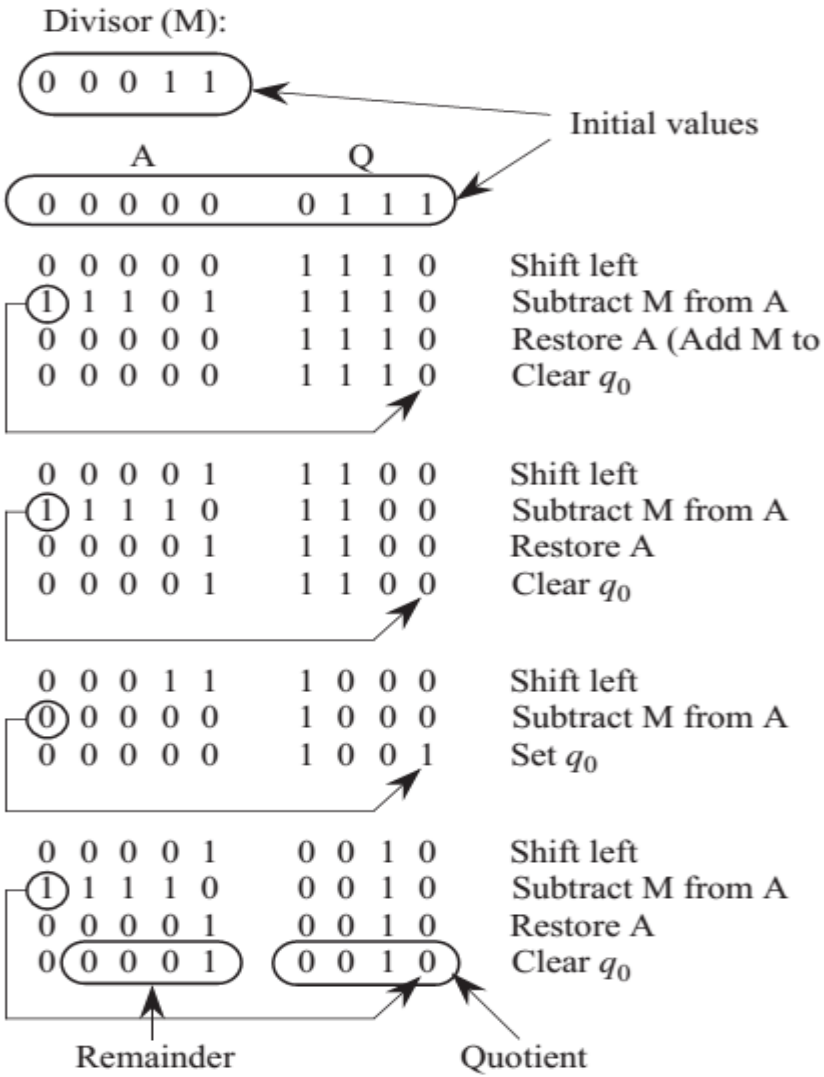

```
                 00001101   <- Quotient
Divisor -> 1011 | 10010011   <- Dividend
                  1011
                 001110
                   1011
                   001111
                     1011
                      100   <- Remainder
```

Partial Remainders

Flowchart for Unsigned Binary Division



Division


Divisor (M):
0 0 0 1 1
Initial values
A
Q
0 0 0 0 0 0 1 1 1
0 0 0 0 0 1 1 1 0 Shift left
1 1 1 0 1 1 1 1 0 Subtract M from A
0 0 0 0 0 1 1 1 0 Restore A (Add M to
0 0 0 0 0 1 1 1 0 Clear q_0
0 0 0 0 1 1 1 0 0 Shift left
1 1 1 1 0 1 1 0 0 Subtract M from A
0 0 0 0 1 1 1 0 0 Restore A
0 0 0 0 1 1 1 0 0 Clear q_0
0 0 0 1 1 1 0 0 0 Shift left
0 0 0 0 0 1 0 0 0 Subtract M from A
0 0 0 0 0 1 0 0 1 Set q_0
0 0 0 0 1 0 0 1 0 Shift left
1 1 1 1 0 0 0 1 0 Subtract M from A
0 0 0 0 1 0 0 1 0 Restore A
0 0 0 0 1 0 0 1 0 Clear q_0
Remainder
Quotient

Twos complement division

1. Load the divisor into the M register and the dividend into the A,Q registers.
2. Shift A,Q left one position.
3. If M and A have the same signs, perform A-M else perform A+M
4. The preceding operation is successful if the sign of A is the same as before, after the operation,
   * if the operation is successful or A = 0, then set $Q_0 = 1$
   * if the operation is unsuccessful and A is = not 0 then set $Q_0 = 0$ and restore the previous value of A.
5. Repeat steps 2 through 4 as many times as there are bit positions in Q.
6. The remainder is in A. If the signs of the divisor and dividend is the same, then the quotient is Q, else it is the twos complement of Q.

Is this correct?

D = Q X V + R -7 = -2 X 3 - 1

where

D = dividend - 1001 (-7)

Q = quotient – the twos complement of 1110(-2)

V = divisor – 0011 (3)

R = remainder – 1111 (-1)

D = 1110 X 0011+ 1111

= 1001

Signed Division

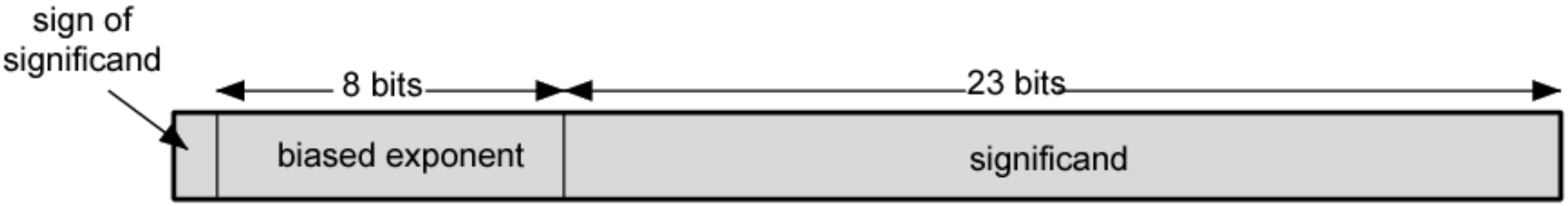
$$D = Q \times V + R$$

$$\begin{array}{llllll}
D = 7 & V = 3 & \Rightarrow & Q = 2 & R = 1 \\
D = 7 & V = -3 & \Rightarrow & Q = -2 & R = 1 \\
D = -7 & V = 3 & \Rightarrow & Q = -2 & R = -1 \\
D = -7 & V = -3 & \Rightarrow & Q = 2 & R = -1
\end{array}$$

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?

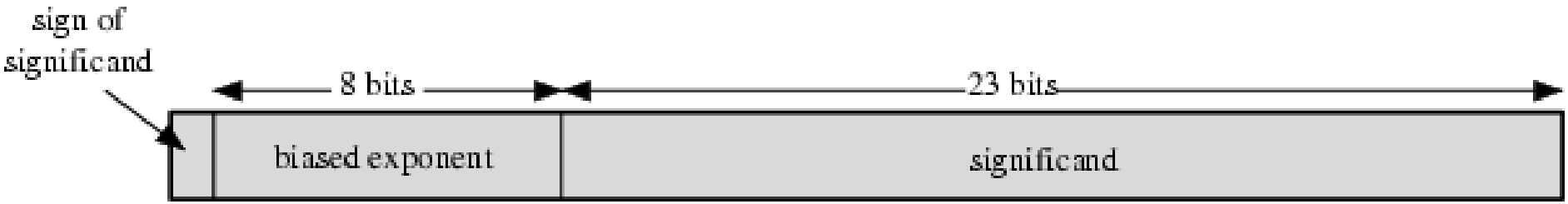
Floating Point



(a) Format

- +/- .significand x 2^{exponent}
- 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

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

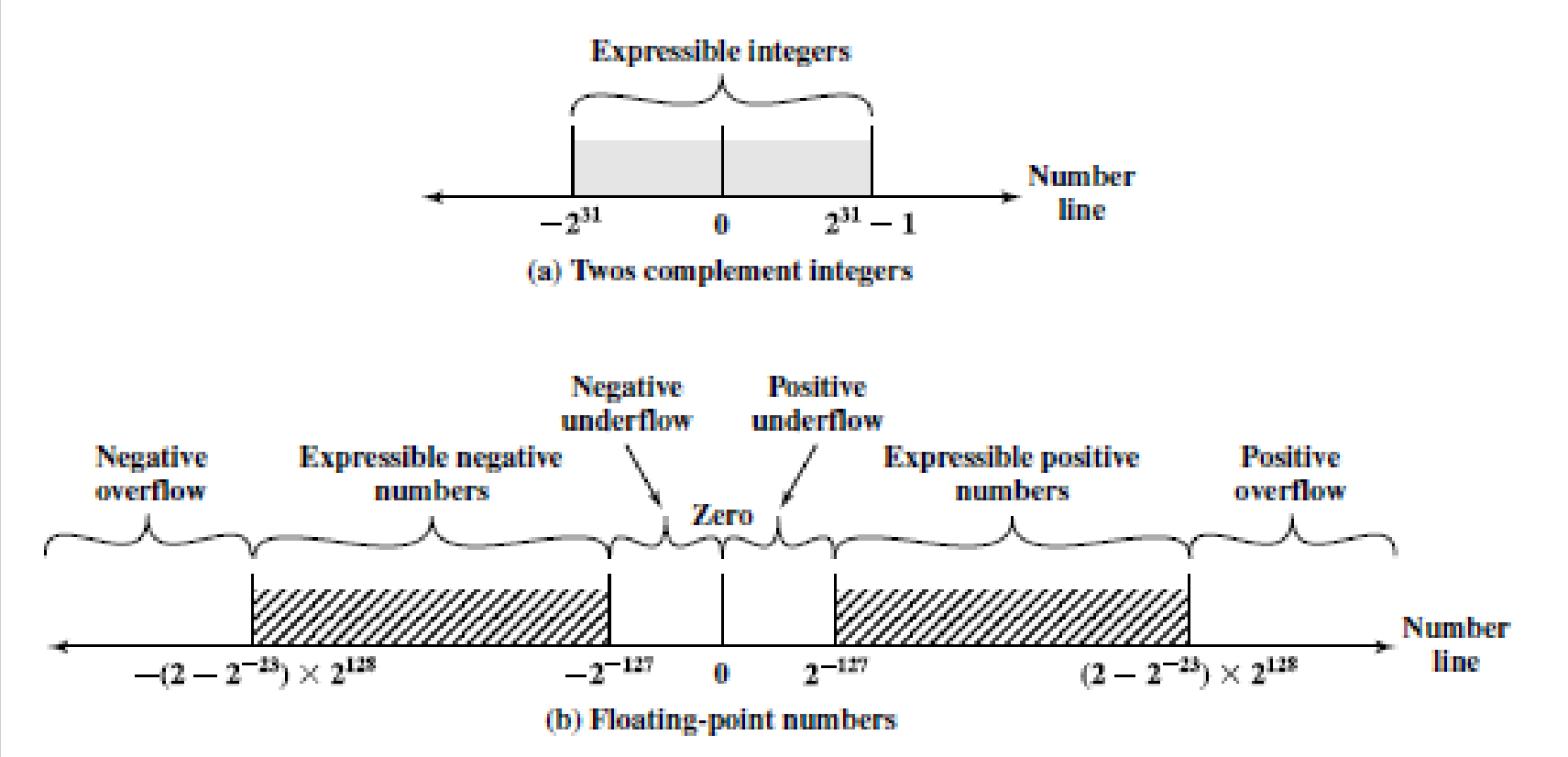
Normalization

- 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)

Accuracy

- Accuracy
  - The effect of changing lsb of mantissa
  - 23 bit mantissa 2^{-23} ≈ 1.2×10^{-7}
  - About 6 decimal places
- Maximum Value is determined by the exponent

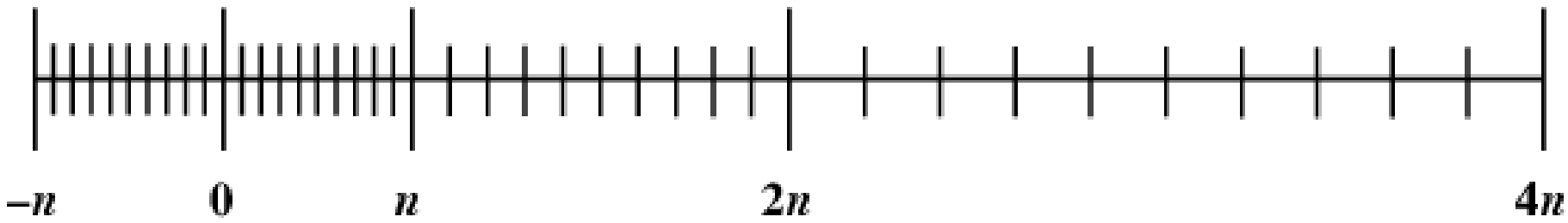
Expressible Numbers



Signs for Floating Point

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

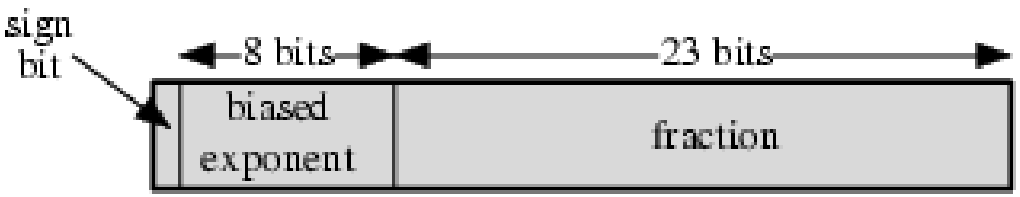
Density of Floating Point Numbers



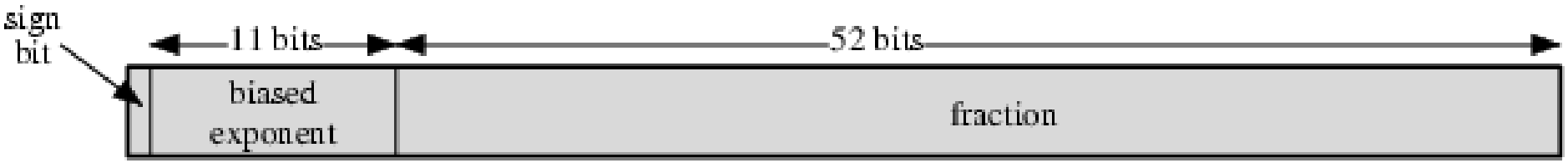
IEEE 754

- 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

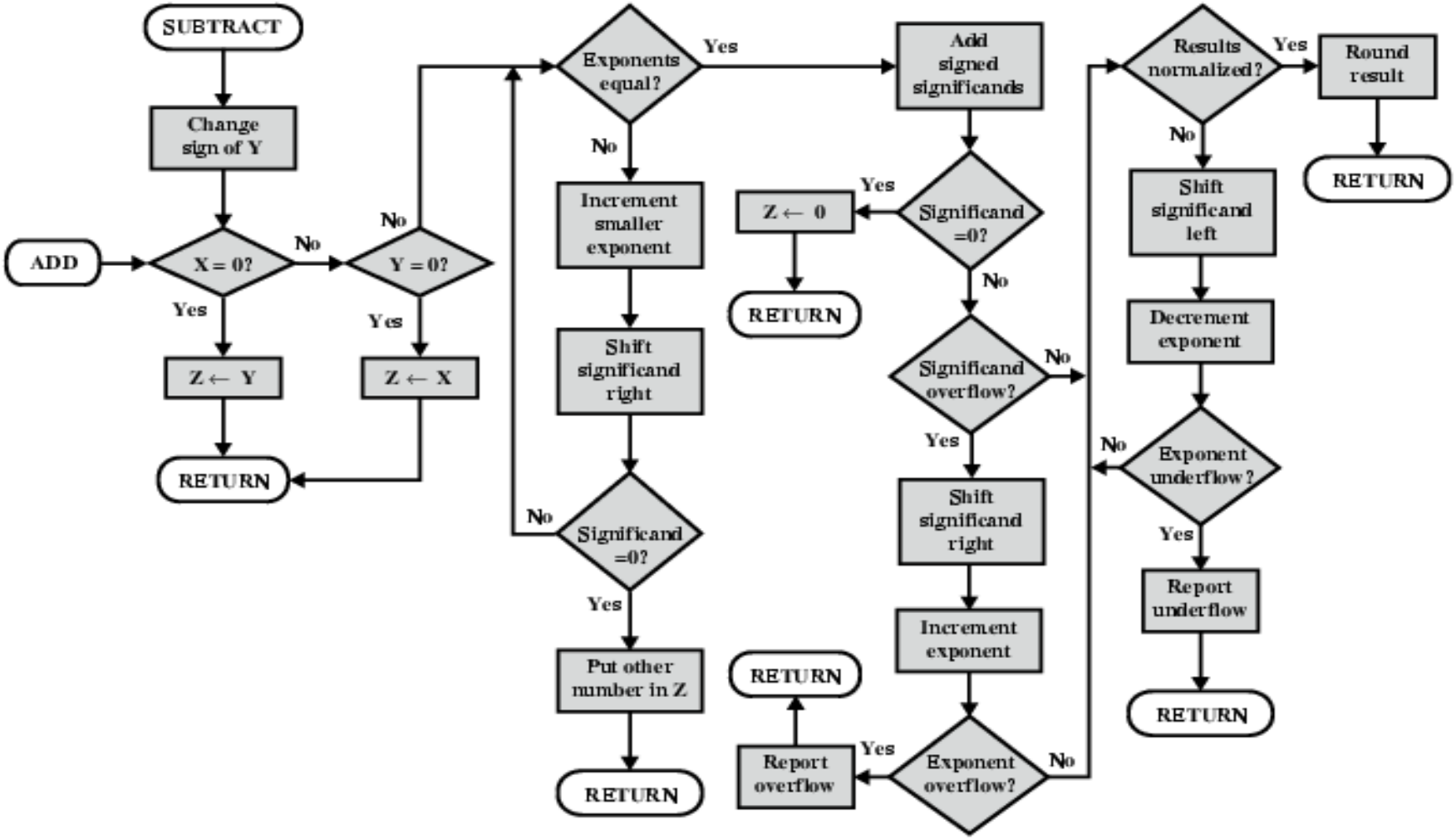


(b) Double format

FP Arithmetic +/-

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

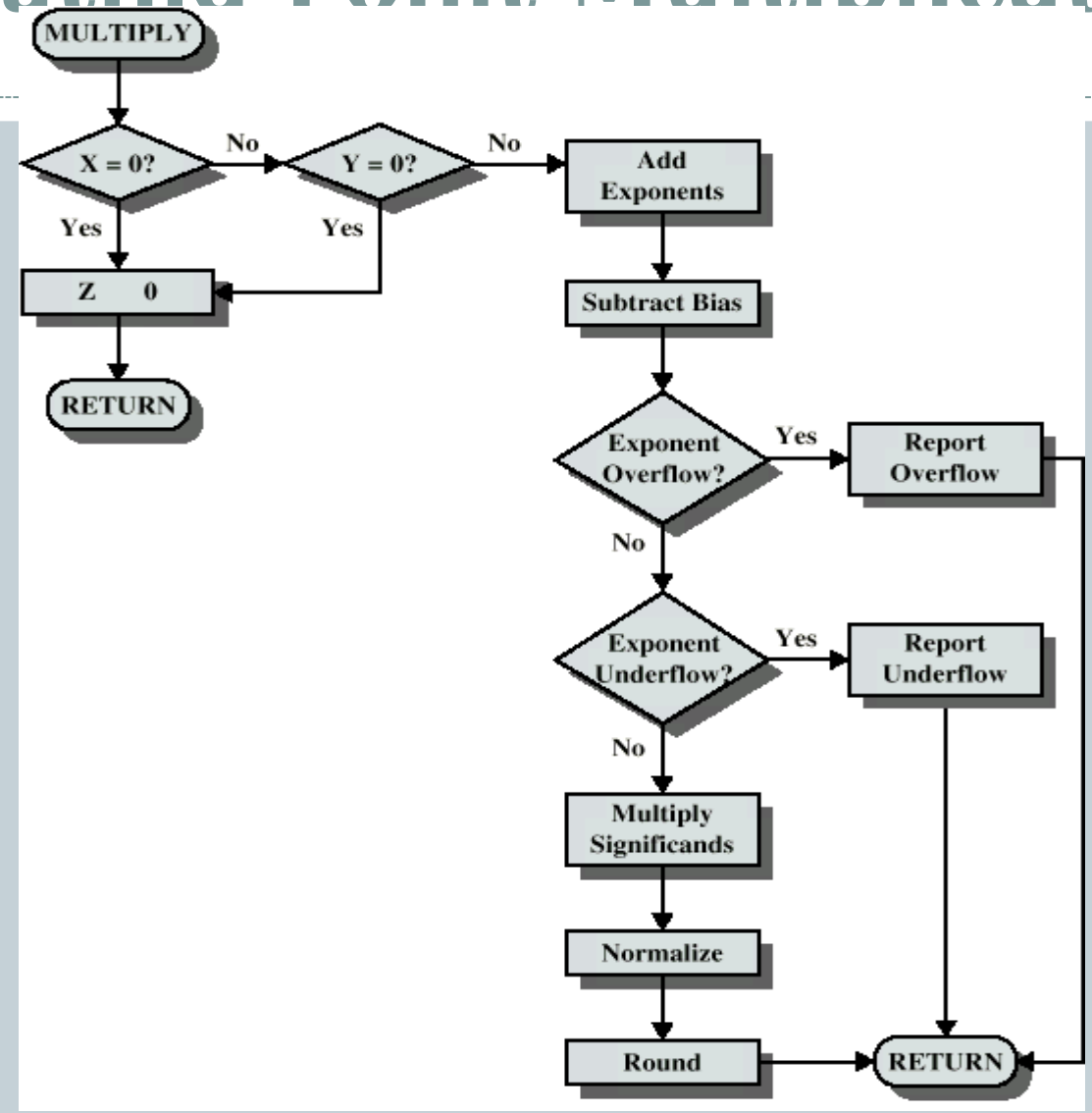
FP Addition & Subtraction Flowchart



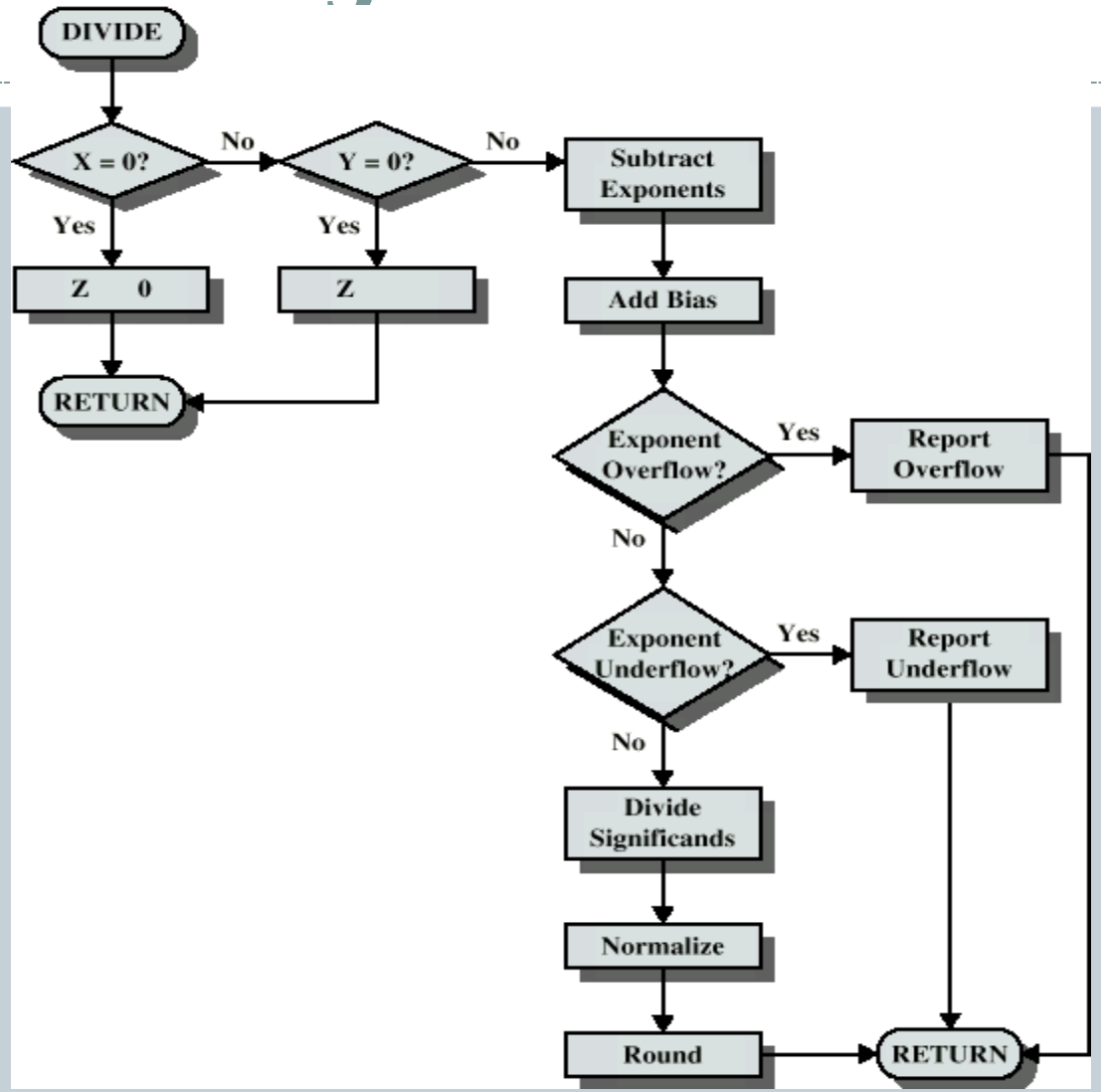
FP Arithmetic x/÷

- 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



End of Chapter 2

COMPUTER ARCHITECTURE AND ORGANIZATION (CSEG3202)