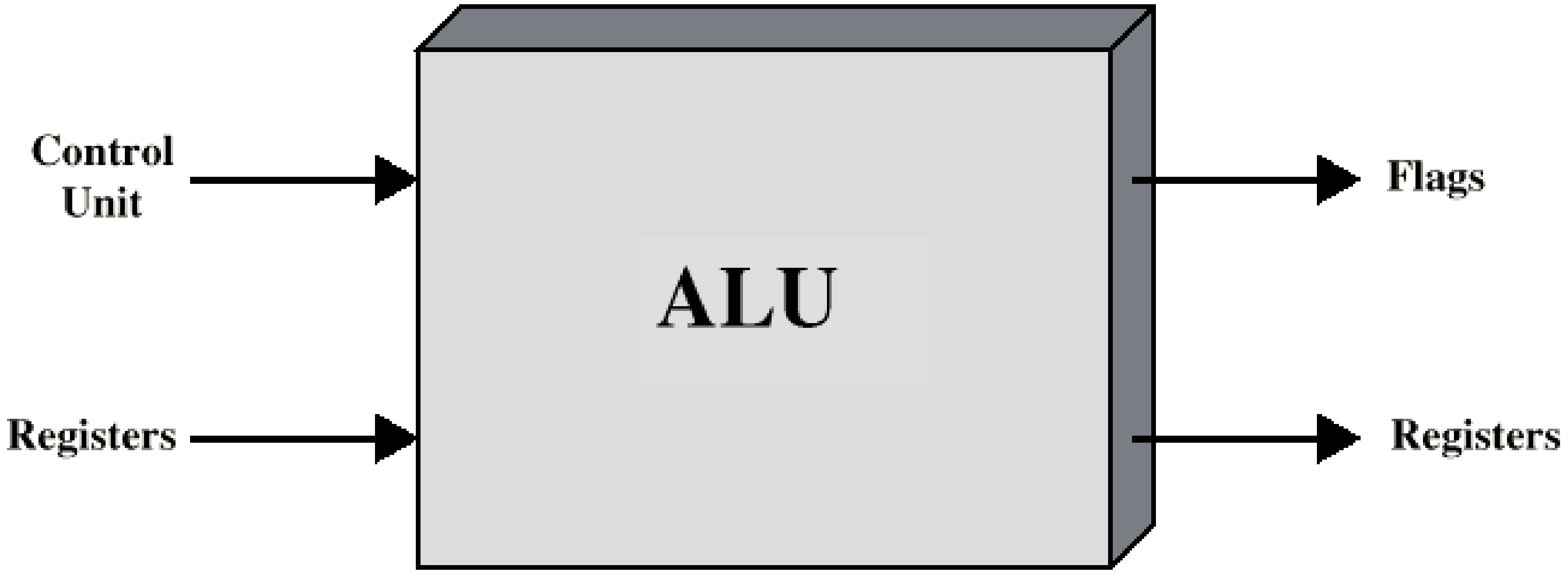
William Stallings
Computer Organization
and Architecture
7th Edition

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 Compliment

- +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

Table 9.1 Characteristics of Twos Complement Representation and Arithmetic

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 9.2 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	—

–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

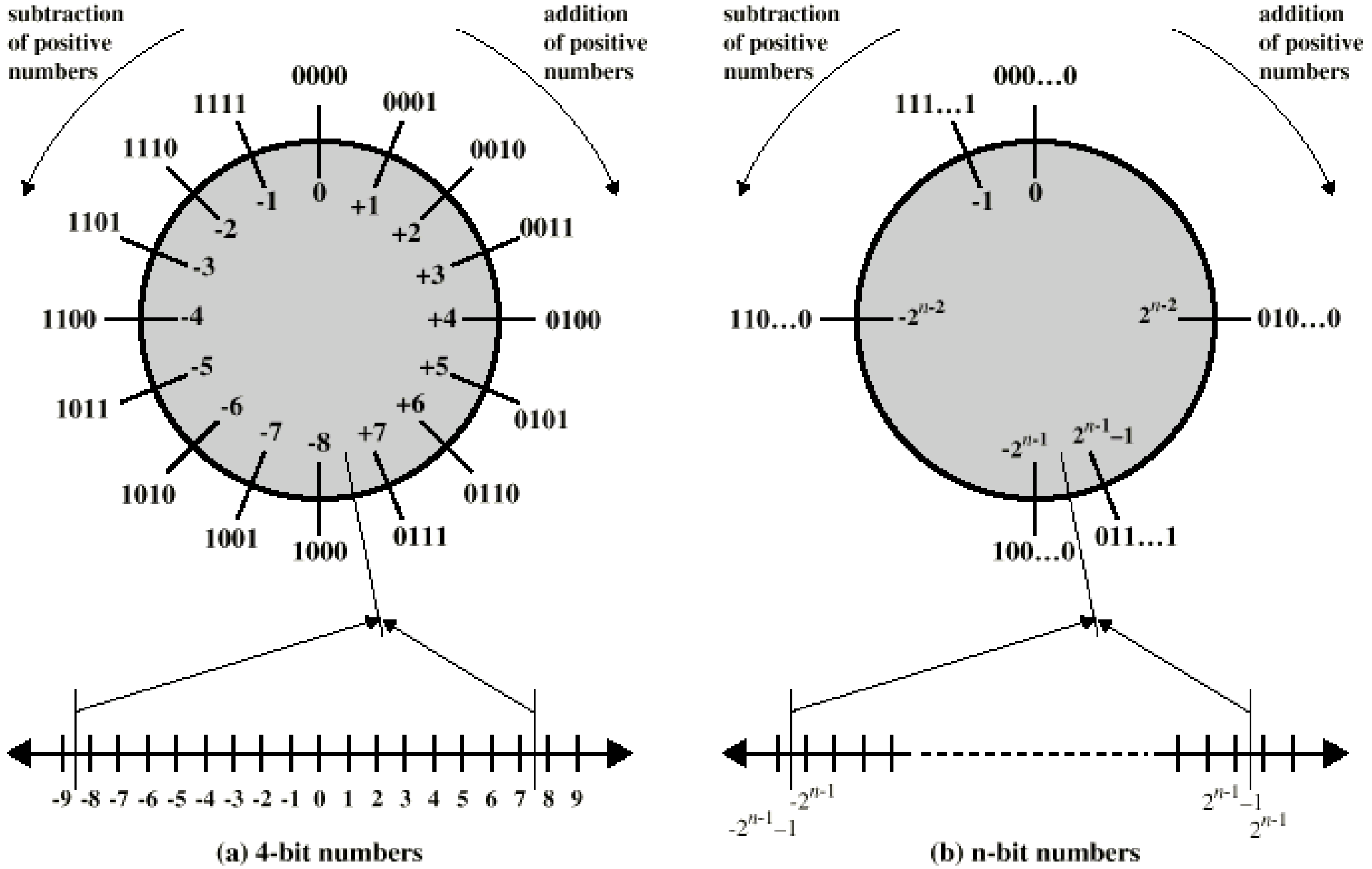
–128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

–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

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 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

- 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)

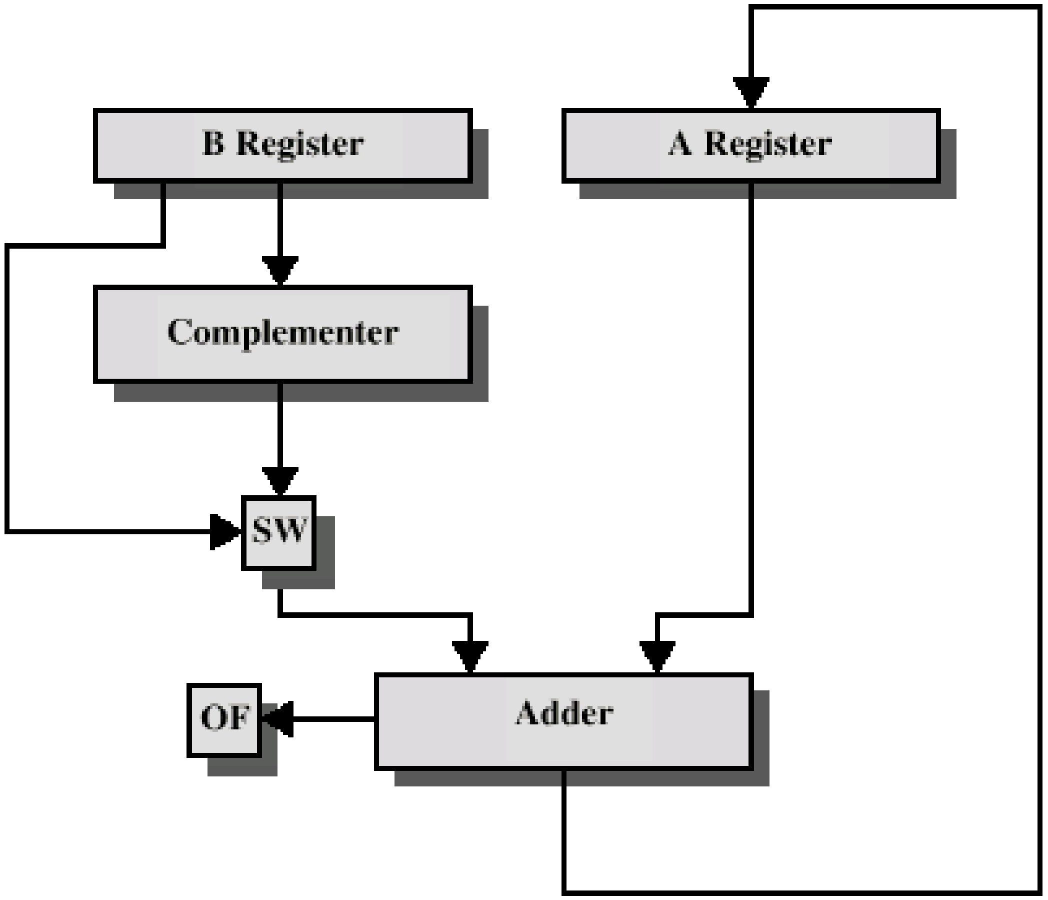
Addition and Subtraction

- Normal binary addition
- 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

Hardware for Addition and Subtraction



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

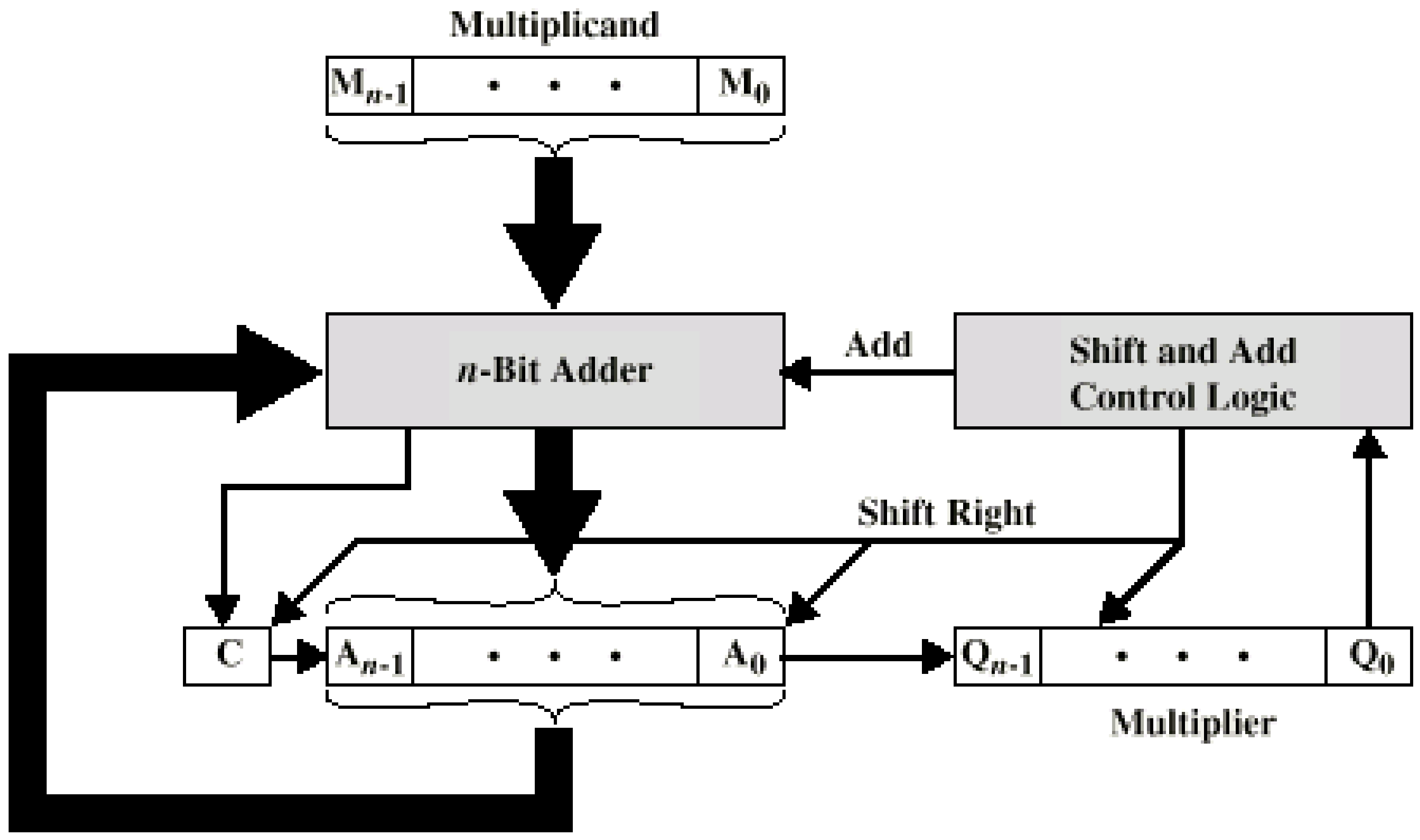
Multiplication

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

Multiplication Example

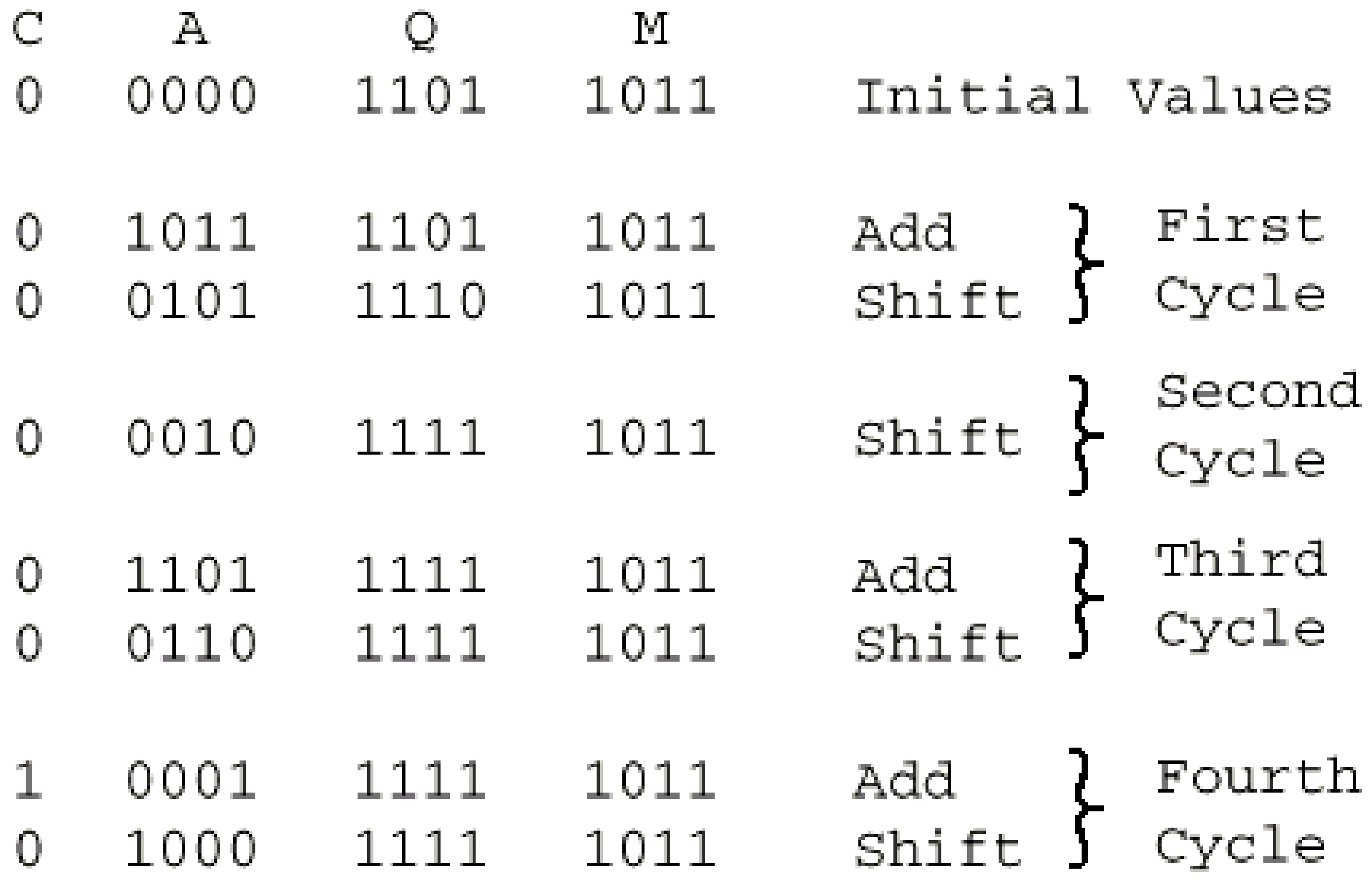
- 1011 Multiplicand (11 dec)
- x 1101 Multiplier (13 dec)
- 1011 Partial products
- 0000 Note: if multiplier bit is 1 copy
- 1011 multiplicand (place value)
- 1011 otherwise zero
- 10001111 Product (143 dec)
- Note: need double length result

Unsigned Binary Multiplication



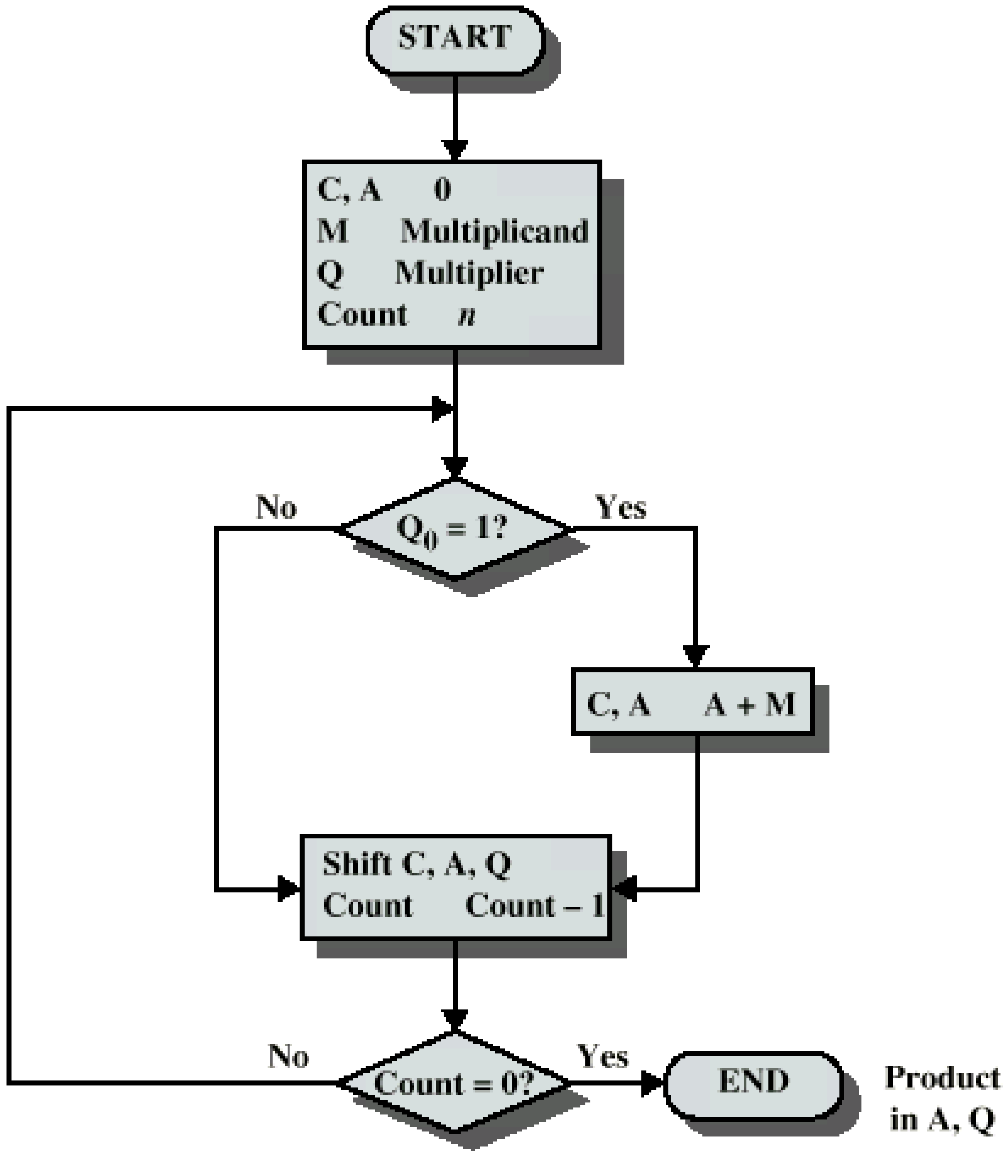
(a) Block Diagram

Execution of Example



C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First 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	

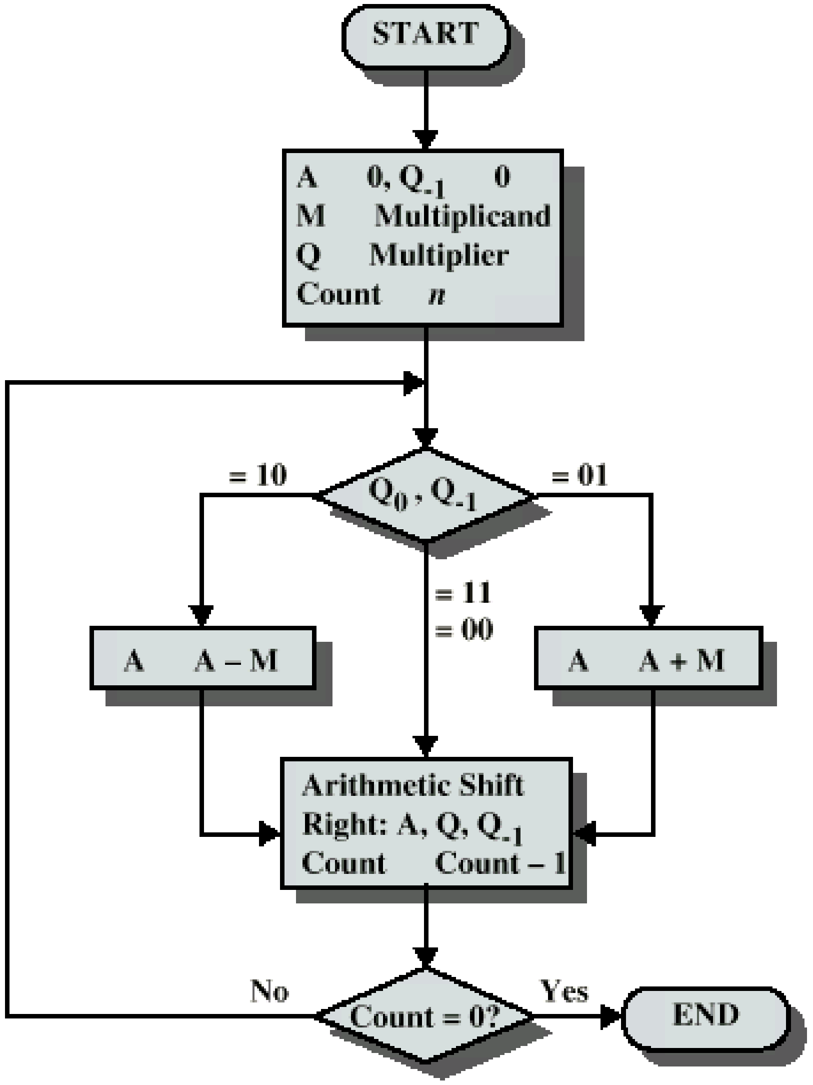
Flowchart for Unsigned Binary Multiplication



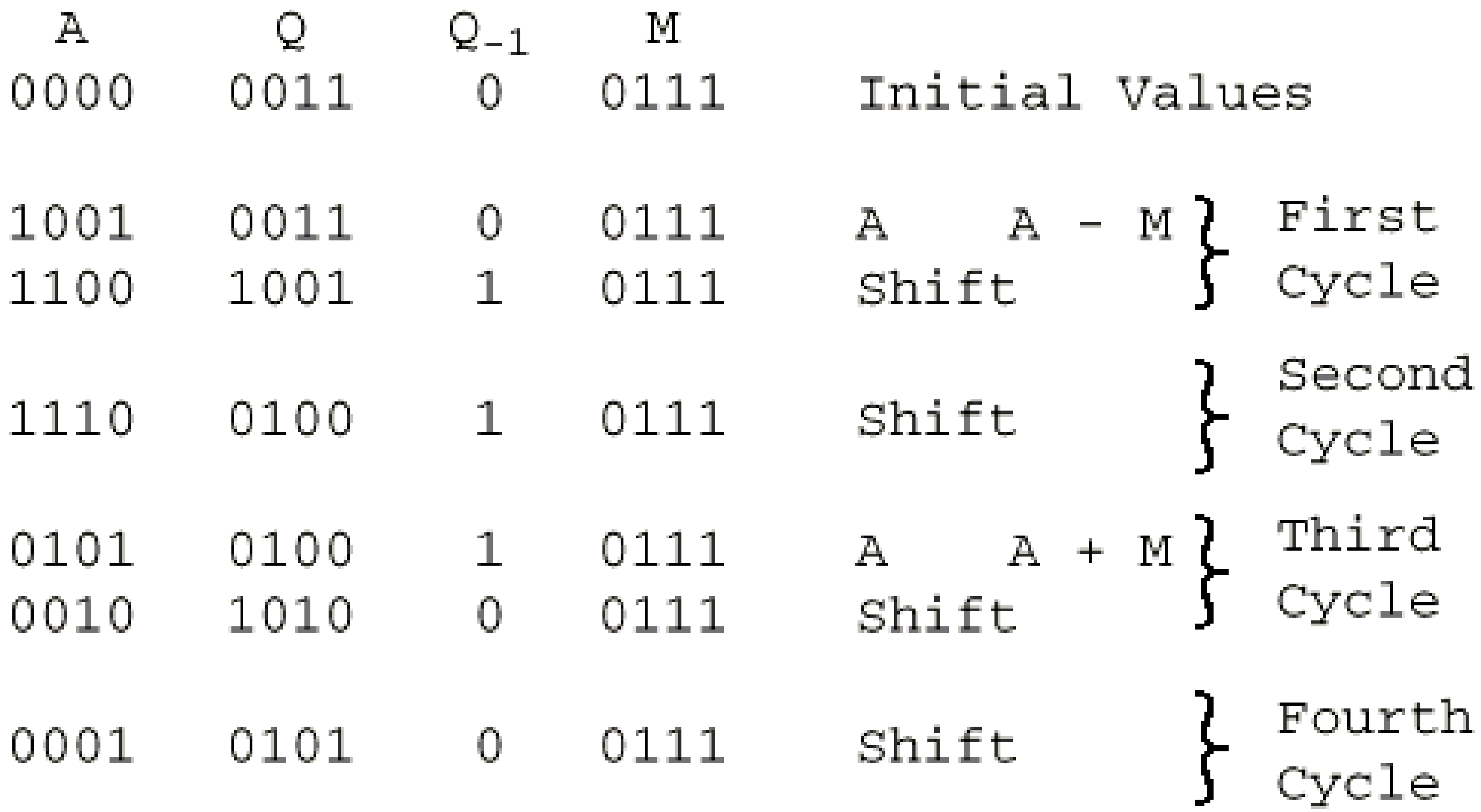
Multiplying Negative Numbers

- 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

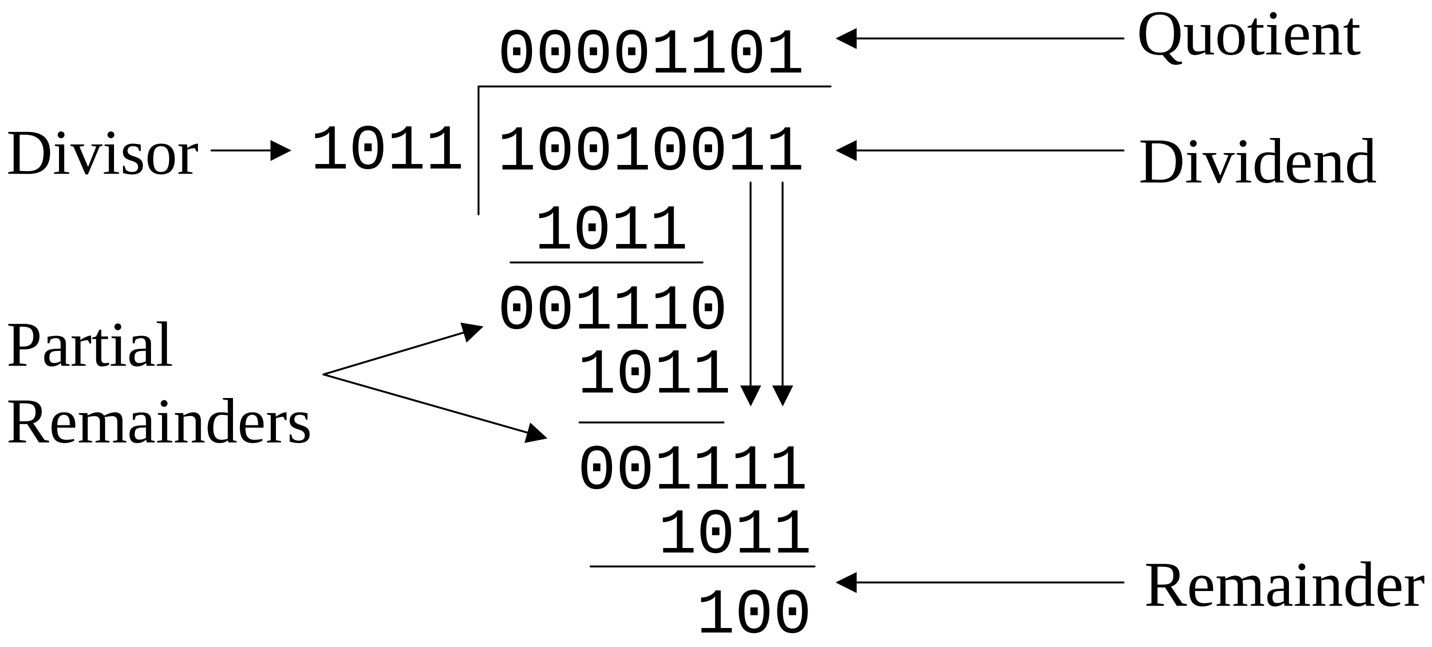


A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A A − M	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A A + M	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

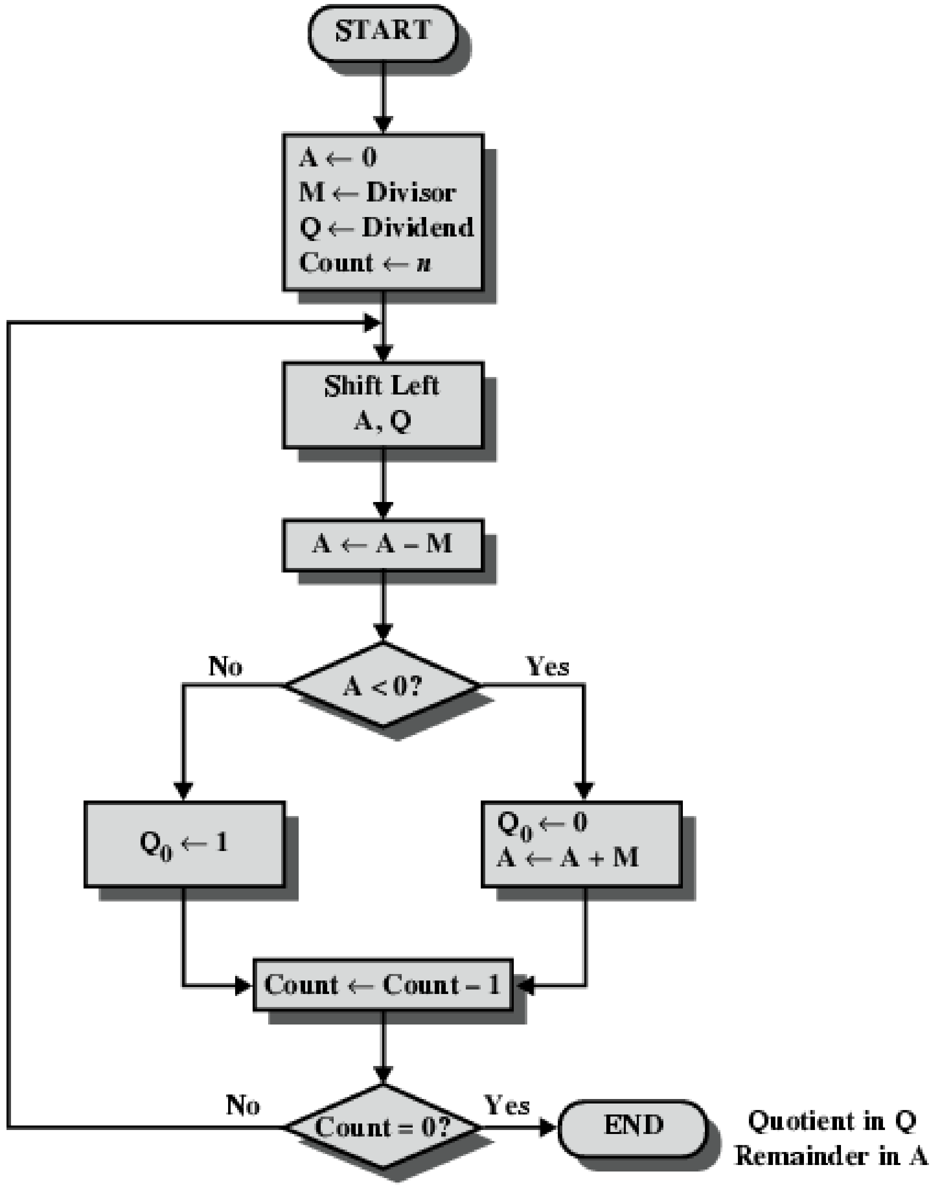
Division

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

Division of Unsigned Binary Integers



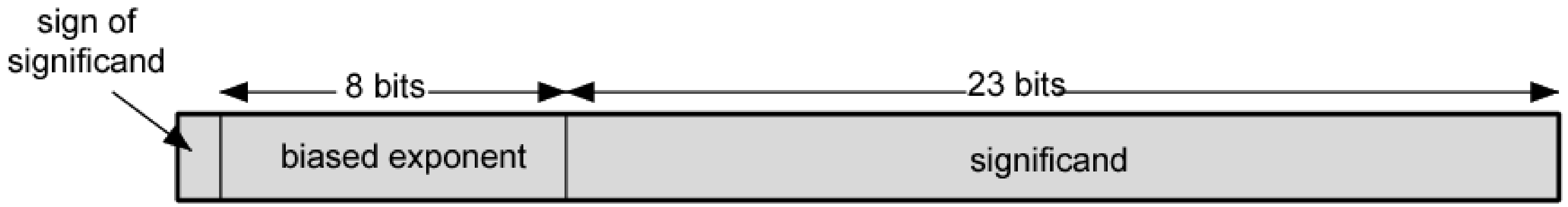
Flowchart for Unsigned Binary Division



Real Numbers

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

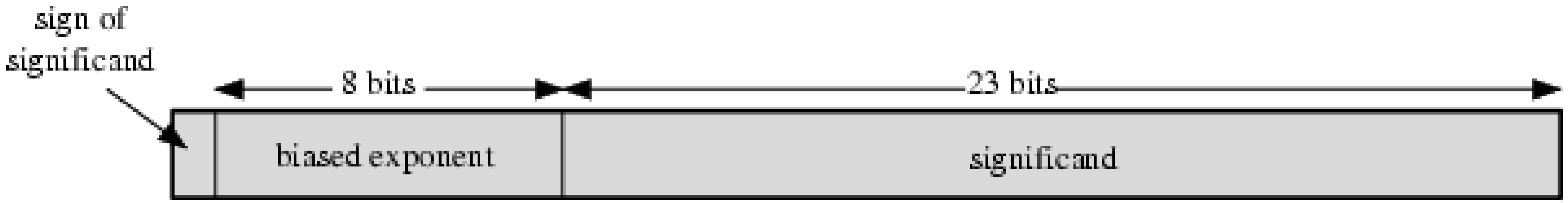
Floating Point



(a) Format

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

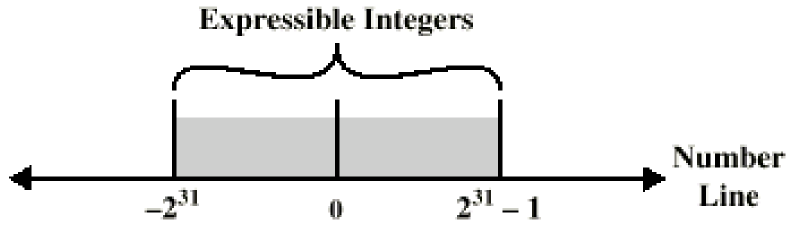
Normalization

- 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)

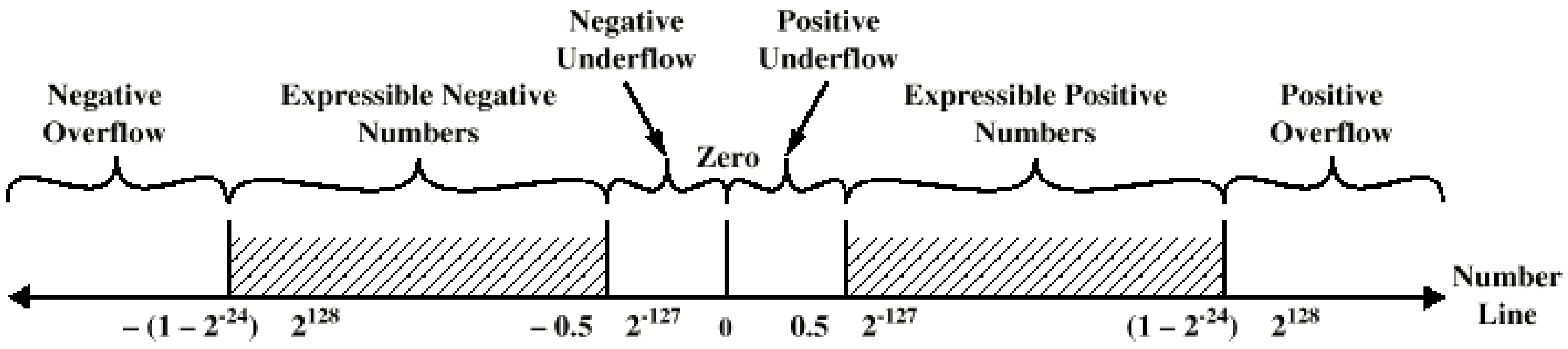
FP Ranges

- 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

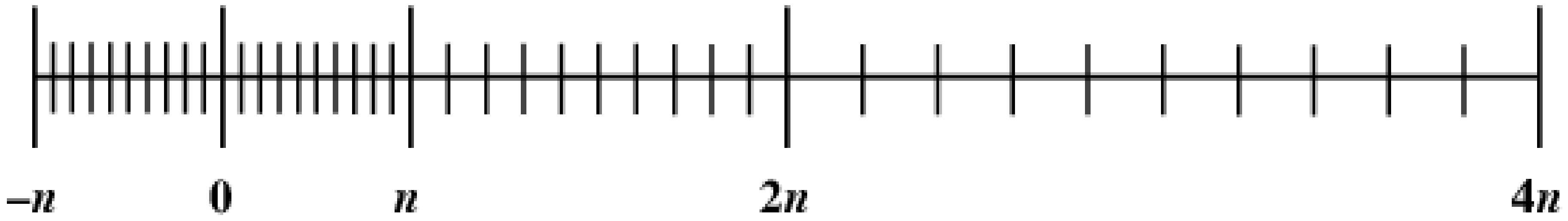


(a) Twos Complement Integers



(b) Floating-Point Numbers

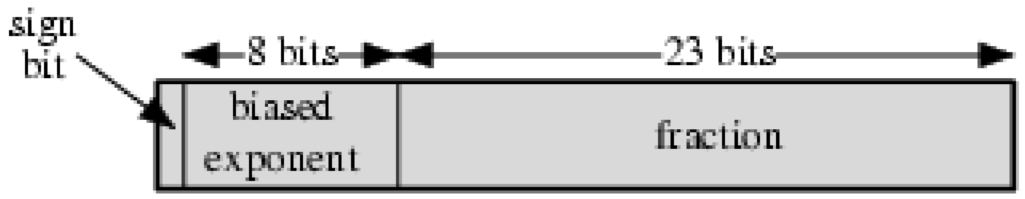
Density of Floating Point Numbers



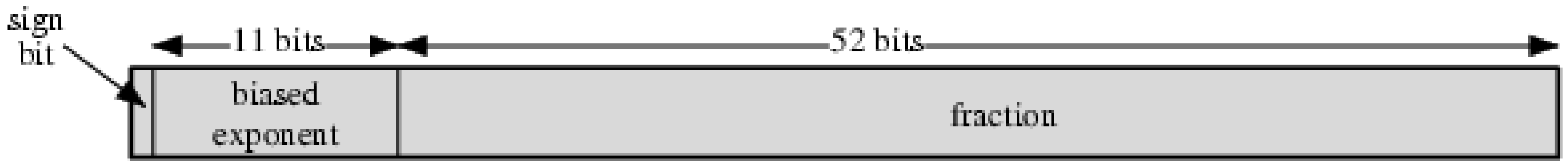
IEEE 754

- 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

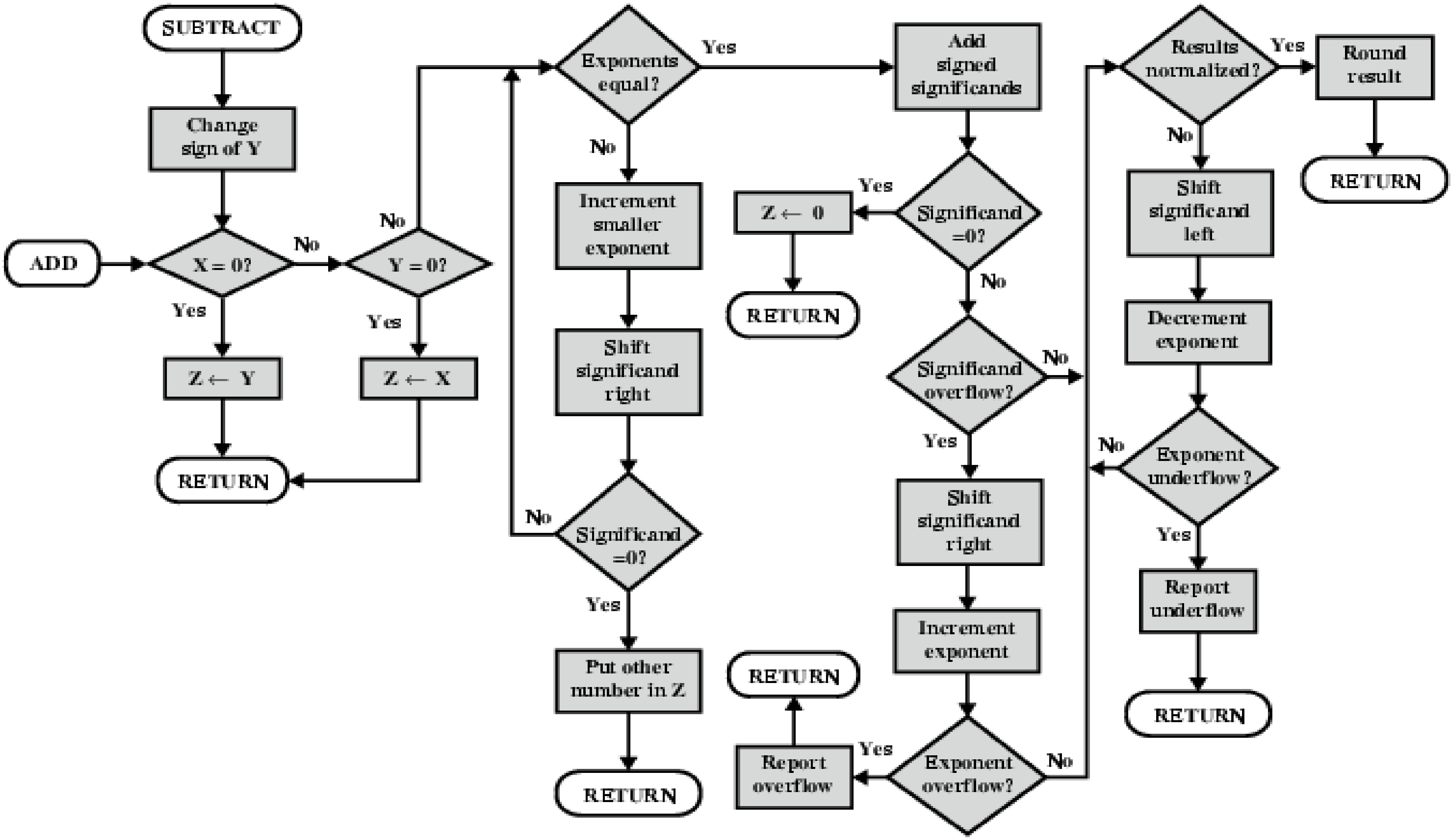


(b) Double format

FP Arithmetic +/-

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

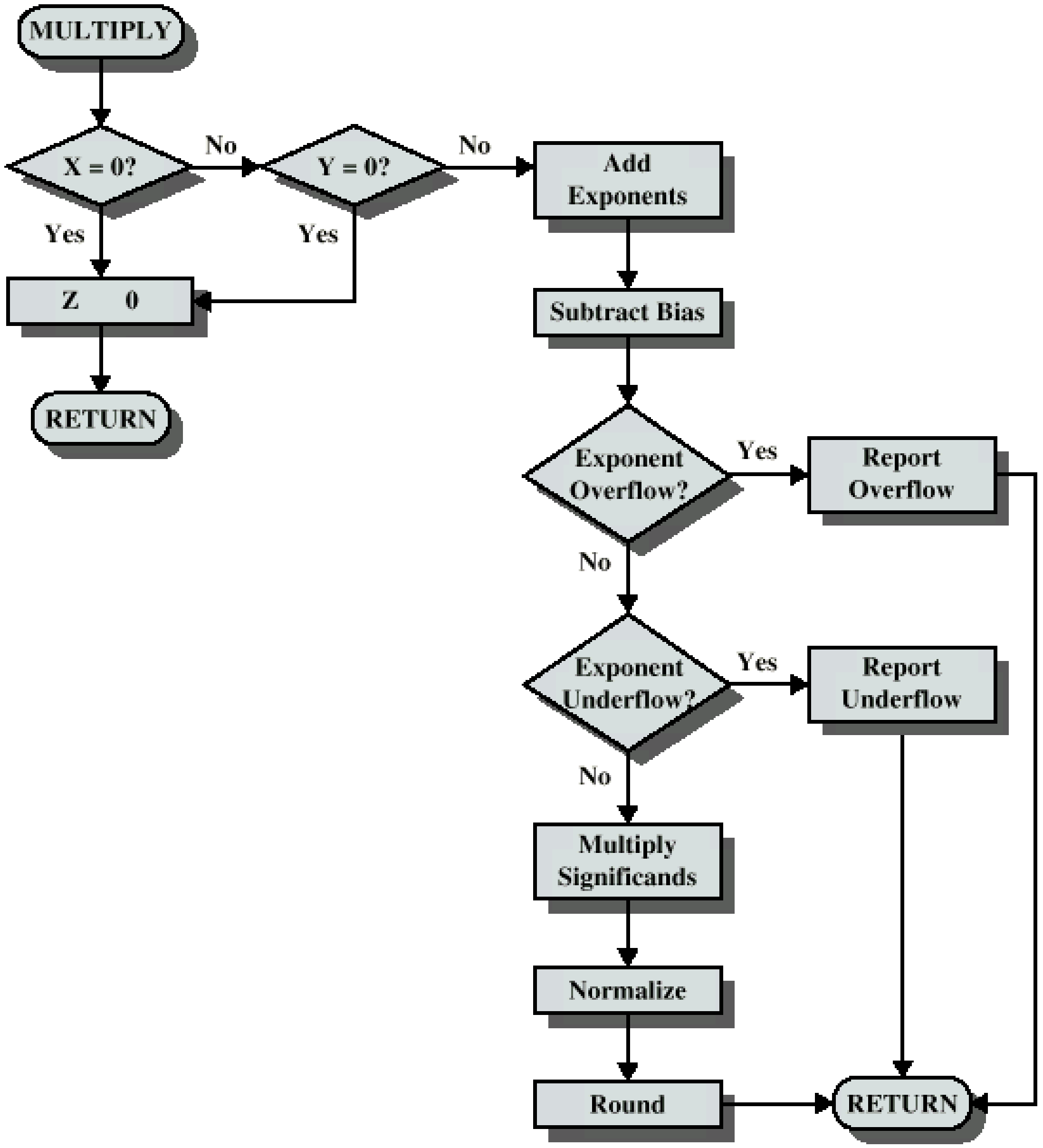
FP Addition & Subtraction Flowchart



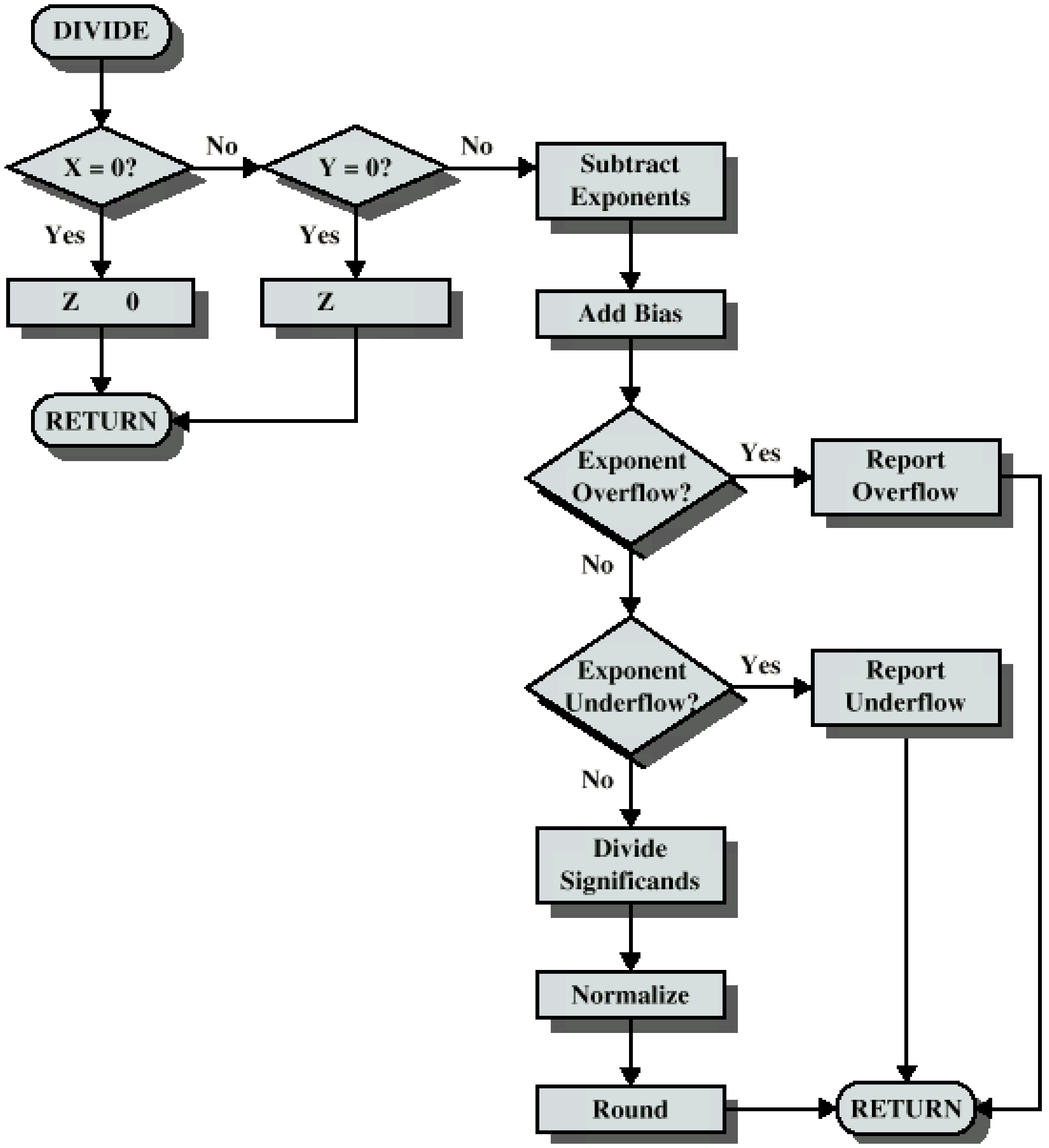
FP Arithmetic x/÷

- 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



Required Reading

- Stallings Chapter 9
- IEEE 754 on IEEE Web site