Chapter 9
Computer Arithmetic

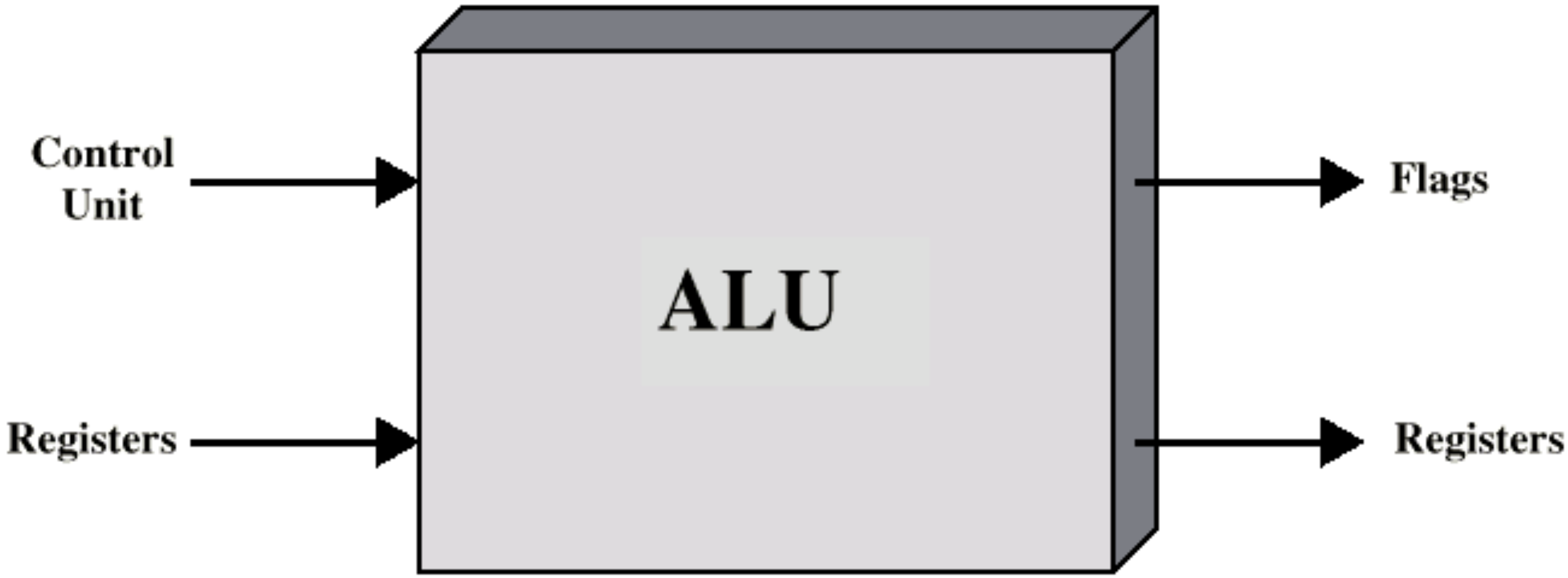
2020.5
Howon Kim

- 정보보호 및 지능형 IoT연구실 - http://infosec.pusan.ac.kr
- 부산대 지능형융합보안대학원 - http://aisec.pusan.ac.kr

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 complement

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

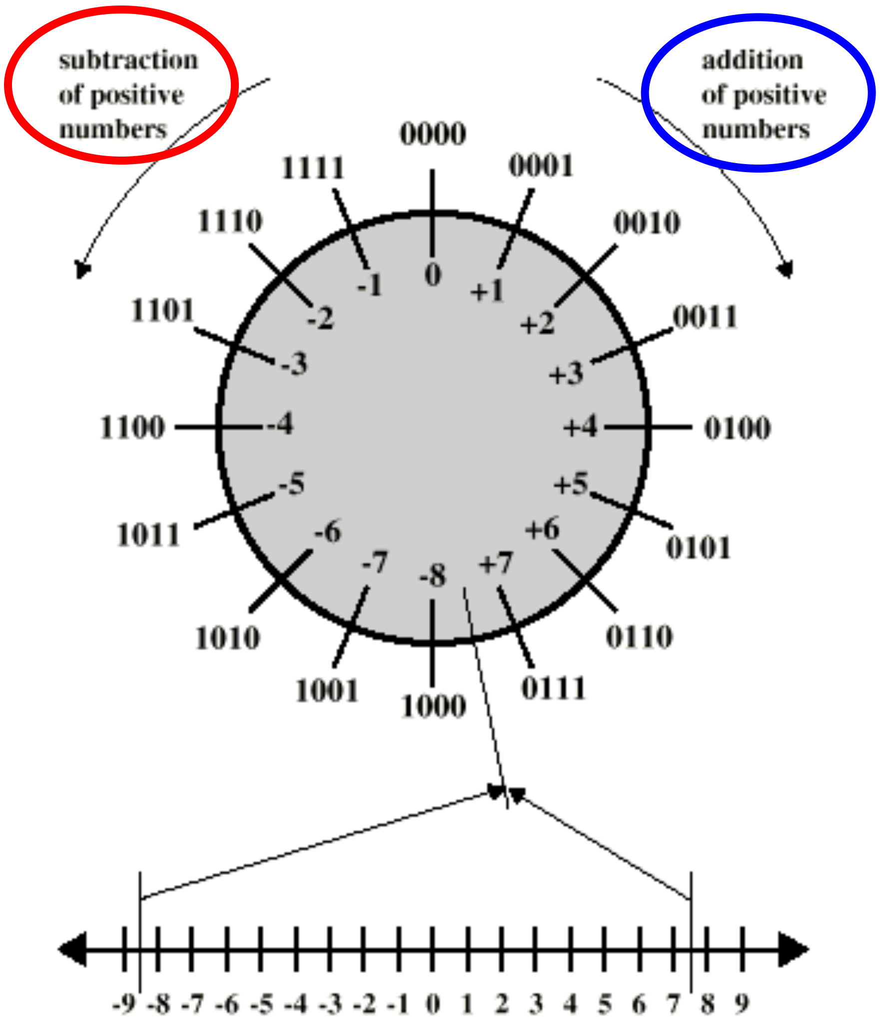
- +3 = 00000011
- +2 = 00000010
- +1 = 00000001
- +0 = 00000000
- -1 = 11111111
- -2 = 11111110
- -3 = 11111101

Benefits

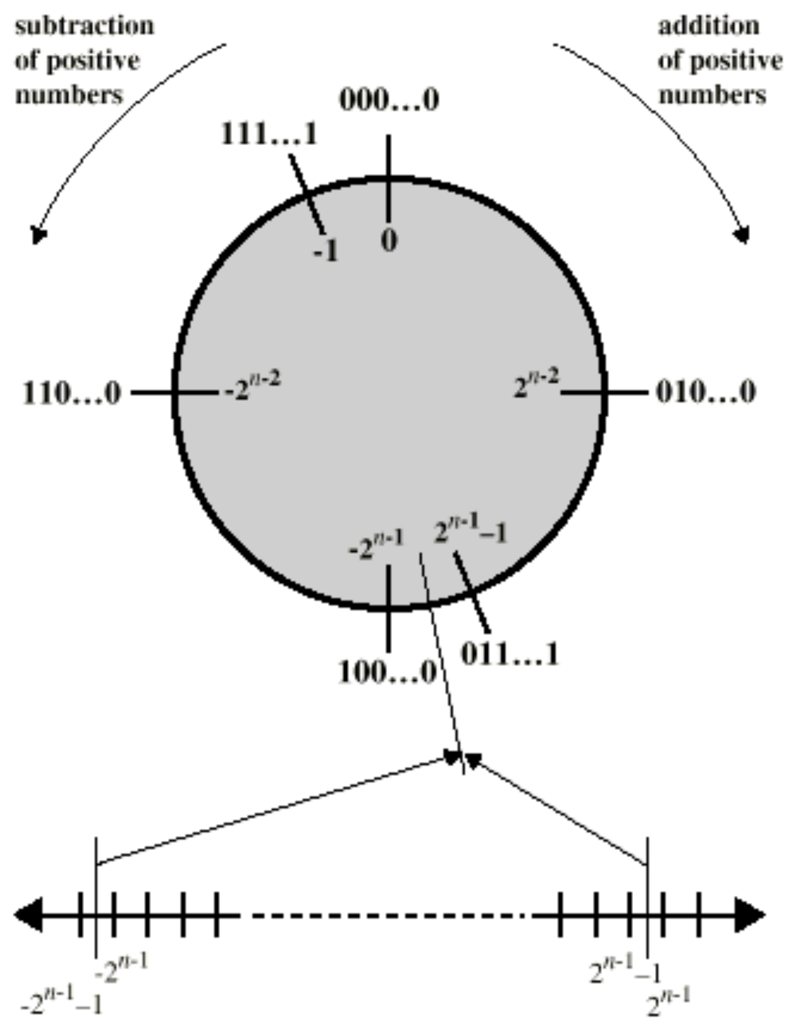
- One representation of zero
- Arithmetic works easily (see later)
- Negating is fairly easy
  - 3 = 00000011
  - Boolean complement gives 11111100
  - Add 1 to LSB 11111101

-3

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 최대표현양수:127 !
- Monitor MSB (sign bit)
- It should change during negation

Range of Numbers

- 8 bit 2s complement

  $+127 = 01111111 = 2^7 - 1$

  $-128 = 10000000 = -2^7$

- 16 bit 2s complement

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

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

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)

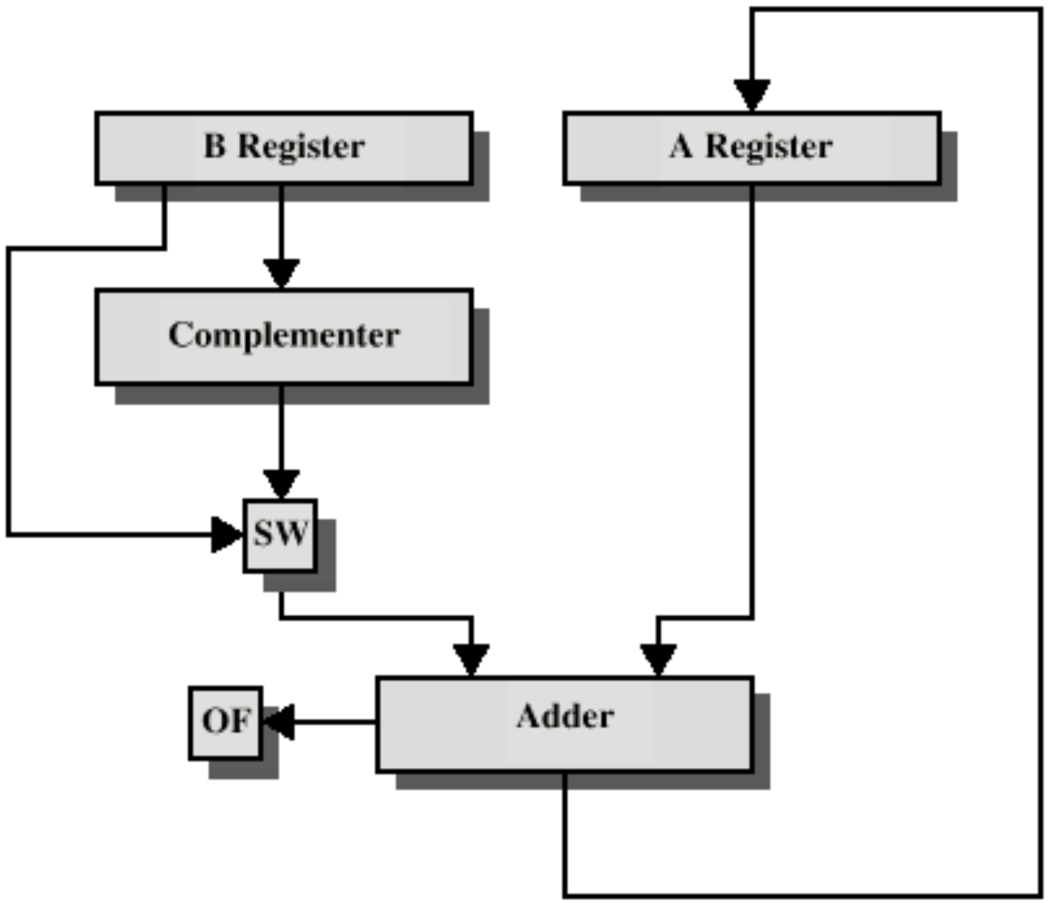
Addition and Subtraction

- Normal binary addition
- Monitor sign bit for overflow

- Take two's complement of subtrahend (b) and add to minuend (a)
  - 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

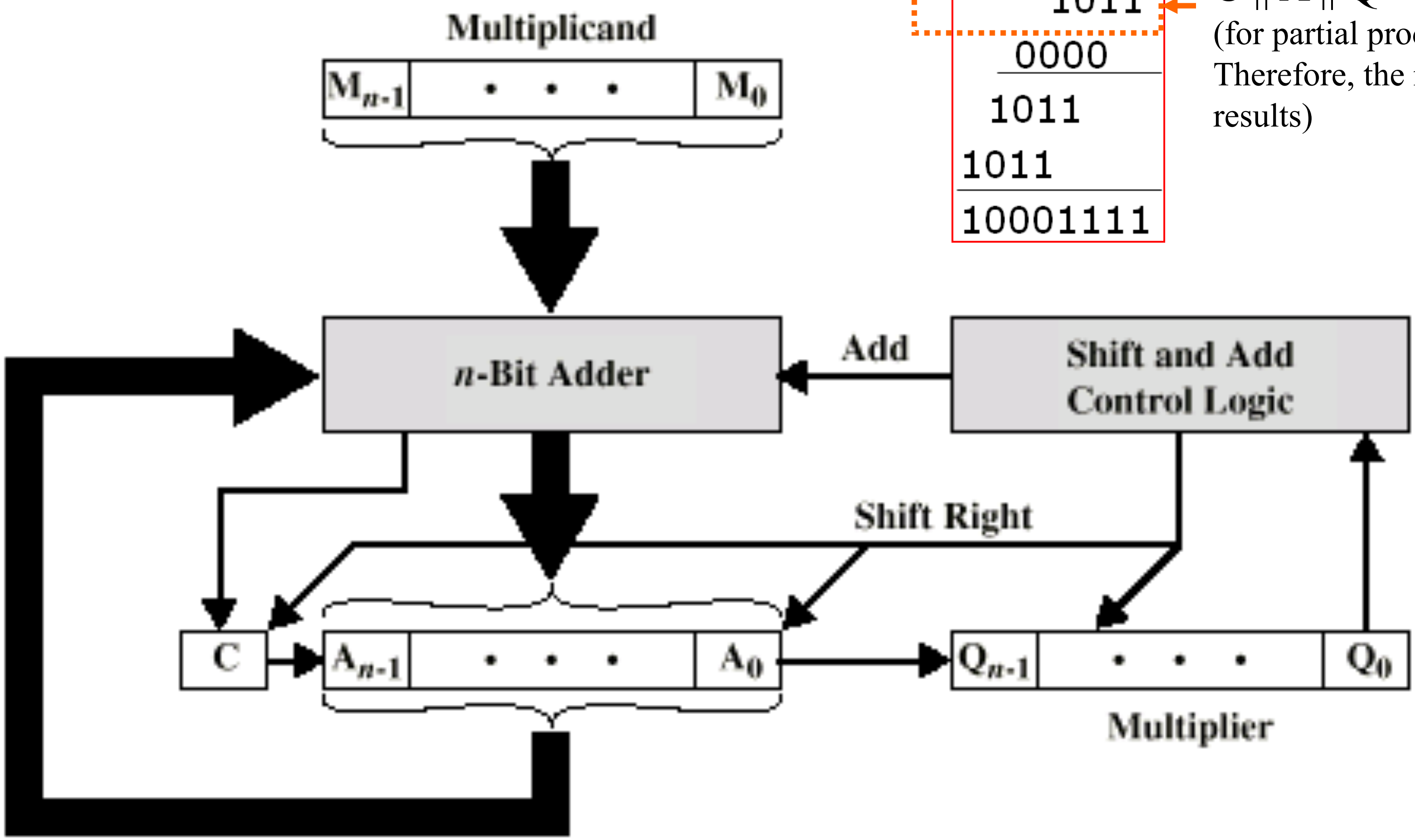
Unsigned Binary Multiplication

M (Multiplicand)

Q (Multiplier)

```
    1011
  x 1101
    1011
   0000
  1011
 1011
10001111
```

C || A || Q
(for partial product. Therefore, the multiplication results)



(a) Block Diagram

Execution of Example

M(Multiplicand)

Q (Multiplier)

```
    1011
  x 1101
    1011
   0000
  1011
 1011
10001111
```

If Q0 ==1 then Add and Shift
else only Shift

	Partial product	multiplier	multiplicand		
C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	A ← A + M Add	First Cycle
0	0101	1110	1011	Shift {C||A||Q} >> 1	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	A ← A + M Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	A ← A + M Add	Fourth Cycle
0	1000	1111	1011	Shift	

A=A+M

```
  0000
+ 1011
  1011
```

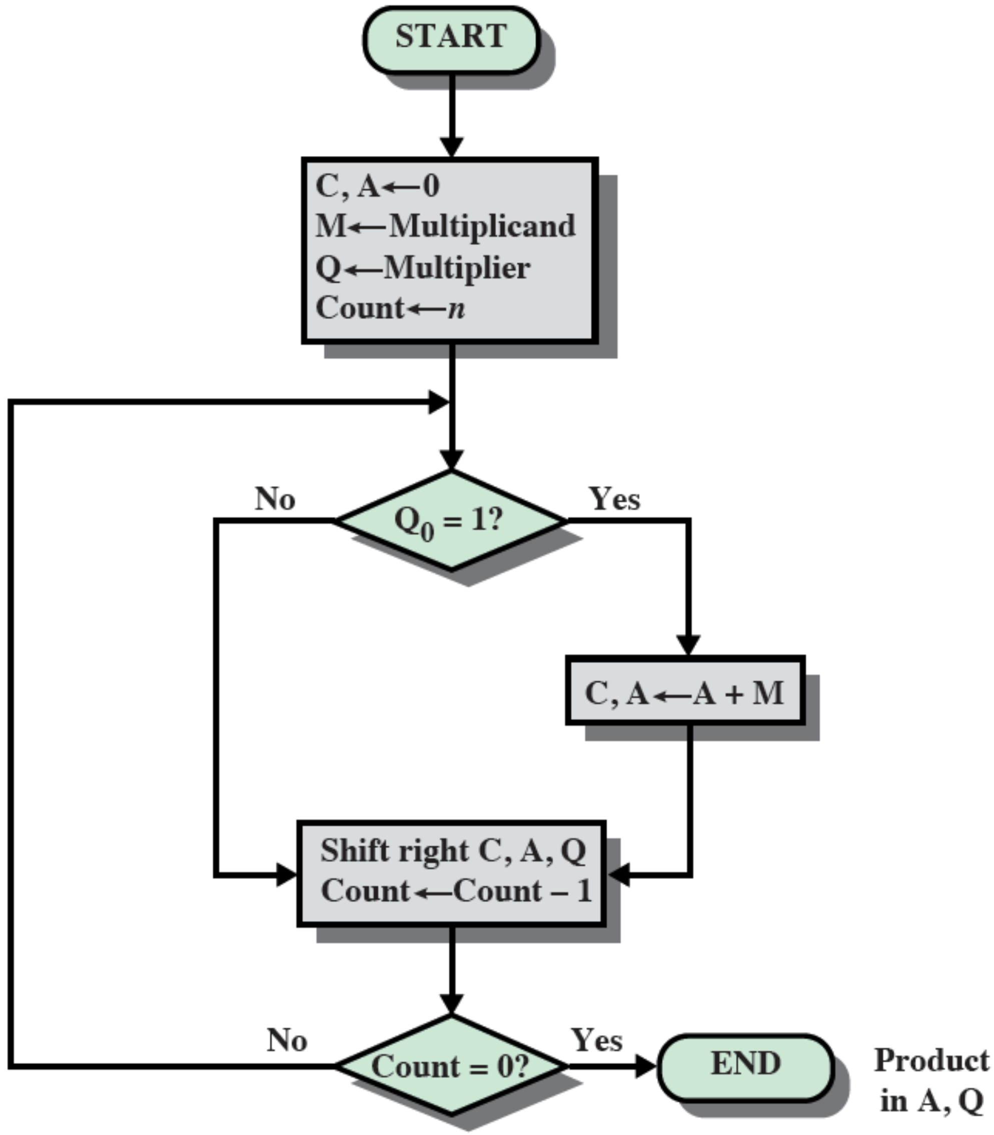
A=A+M

```
  0010
+ 1011
  1101
```

```
  0110
+ 1011
 10001
```

A=A+M

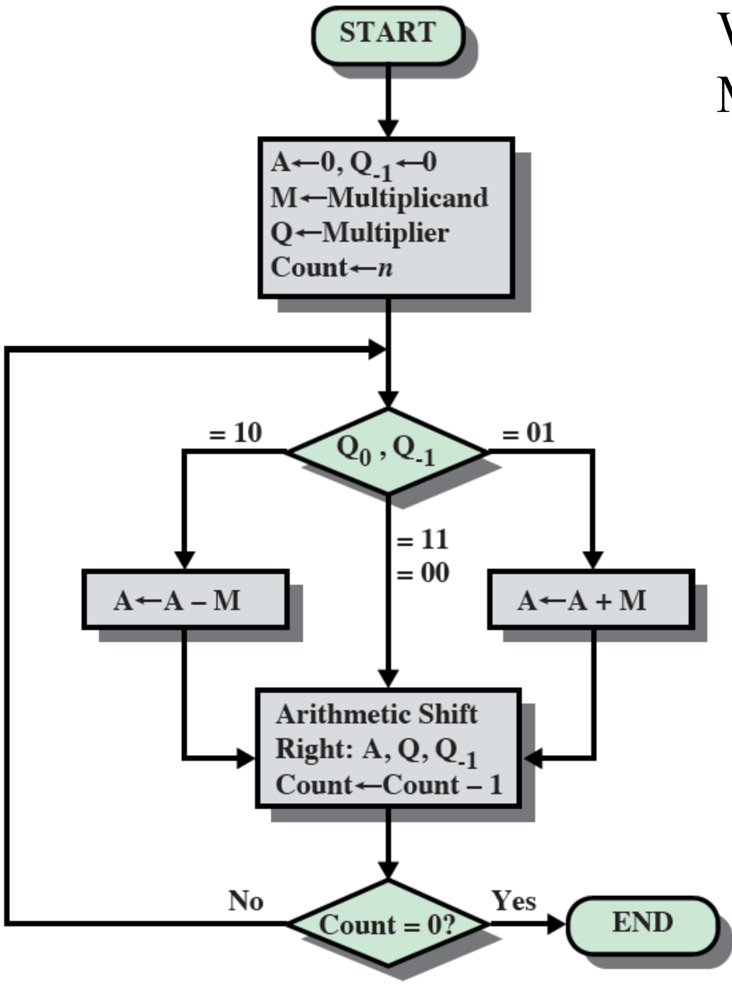
Flowchart for Unsigned Binary Multiplication



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



Booth's Algorithm for Twos Complement Multiplication

We should check every two consecutive bits in Multiplier at a time:

Q_0	Q_{-1}	Q_{-1} - Q_0	operations
0	0	0	No operations
1	0	-1	Subtract M from partial product
1	1	0	No operations
0	1	1	Add M to partial product

Example of Booth's Algorithm (7 X 3)

$$\begin{array}{r} M\,(\text{Multipicand})=7 \\ \times\ Q(\text{Multiplier})=3 \\ \hline A\,||\,Q = 21 \end{array}$$

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

1 0 : A = A-M, and then shift;
*** 0000 + 1001 (2's) = 1001**

1 1 : just shift;

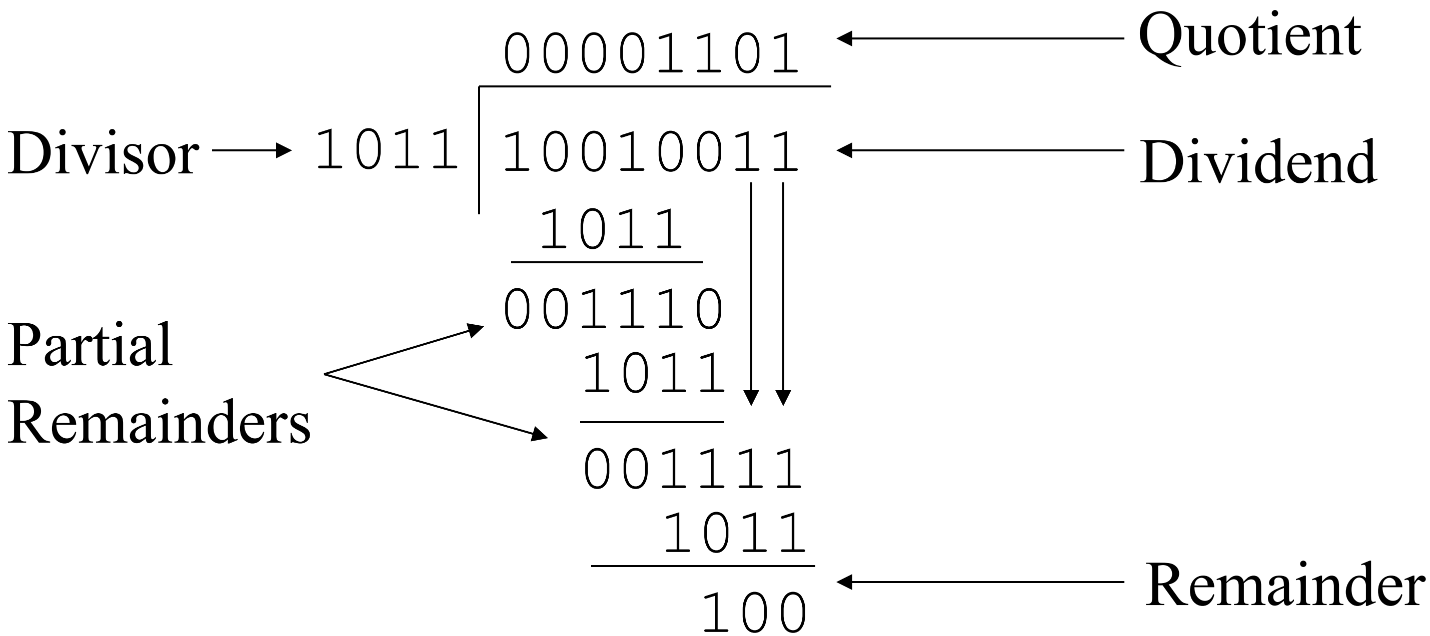
0 1: A = A+M, and then shift;
*** 1110 + 0111 = 0101**

0 0: just shift;

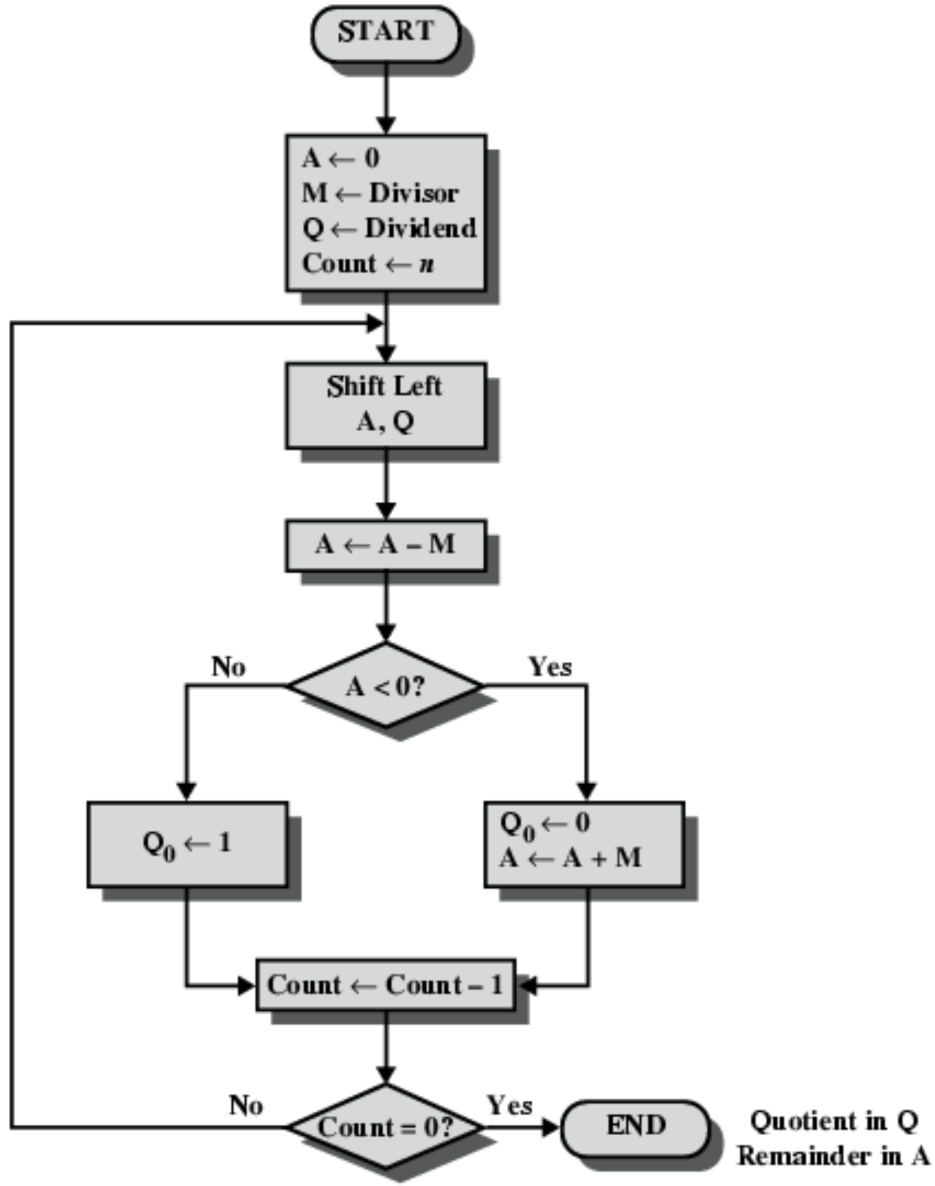
Division

- 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? (vs. decimal point)
- Fixed?
  - Very limited
- Moving?
  - How do you show where it is?

Floating Point

(a) Format

Significand = mantissa = coefficient

- +/- .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

Excess 127로 표현함

```
 1.1010001 × 2^10100   = 0 10010011 10100010000000000000000 =  1.638125 × 2^20
-1.1010001 × 2^10100   = 1 10010011 10100010000000000000000 = -1.638125 × 2^20
 1.1010001 × 2^-10100  = 0 01101011 10100010000000000000000 =  1.638125 × 2^-20
-1.1010001 × 2^-10100  = 1 01101011 10100010000000000000000 = -1.638125 × 2^-20
```

(-20)에 127을 더한 값
(Excess 127)

(b) Examples

$$\begin{array}{rr} 147 & 10010011 \\ 127 & -01111111 \\ \hline & 00010100 \end{array}$$

20

bias value $-(127)$

No two's complement

$$\begin{array}{rr} 01101011 & 107 \\ -01111111 & \\ \hline 11101100 & \end{array}$$

bias value $-(127)$

-20

Two's complement notation of -20?

$11101100 \rightarrow 00010011 + 1 = 00010100$

20

Signs for Floating Point

- Negative Mantissa is not expressed as 2s complement
- 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 (excess 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

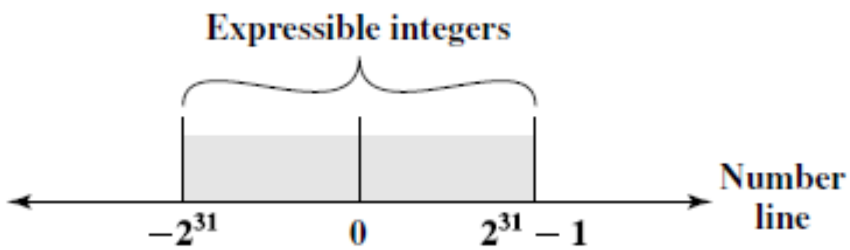
Expressible Numbers (in typical 32-bit format)

- Negative numbers between $-(2 - 2^{-23}) \times 2^{128}$ and -2^{-127}
- Positive numbers between 2^{-127} and $(2 - 2^{-23}) \times 2^{128}$

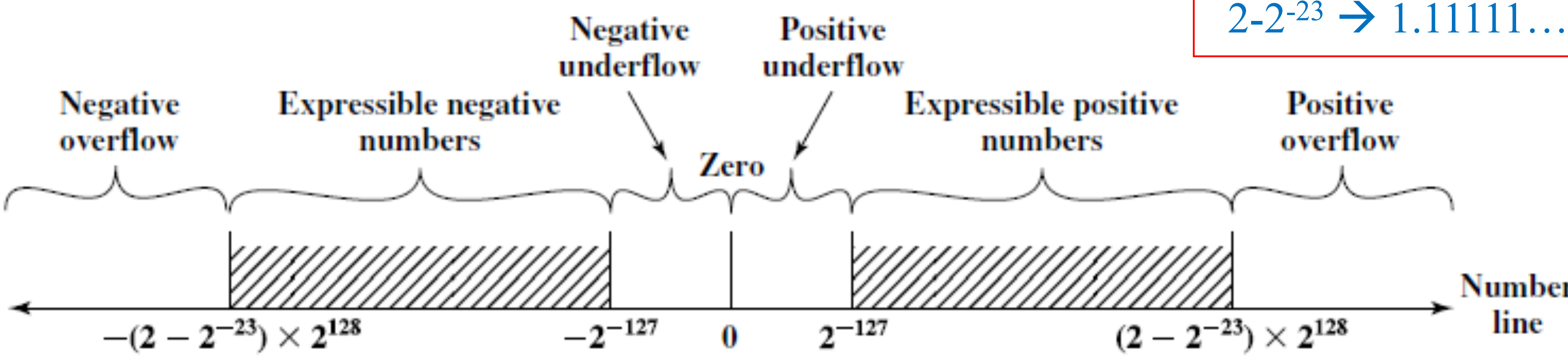
8 bit exponent $-2^{-127} \sim 2^{128}$
23 bit mantissa 2^{-23}
$0.1 \rightarrow 2^{-1}$
$0.01 \rightarrow 2^{-2}$
$0.001 \rightarrow 2^{-3}$

$2-2^{-1} \rightarrow 1.1$
$2-2^{-2} \rightarrow 1.11$
…
$2-2^{-23} \rightarrow 1.11111....1$

소수점
23번째자리



(a) Twos complement integers

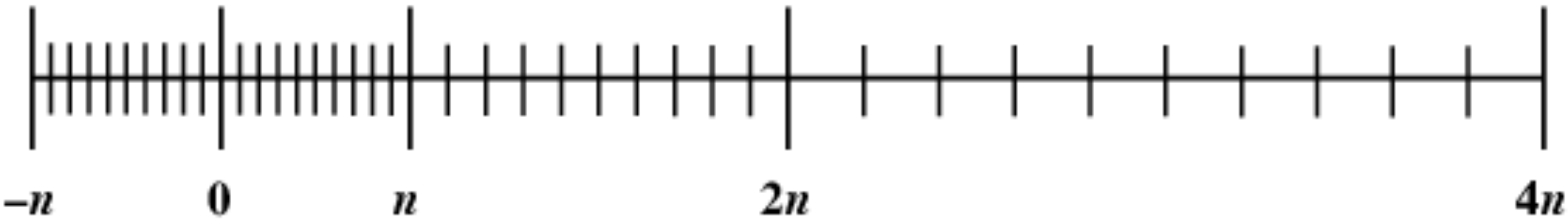


(b) Floating-point numbers

Figure 9.19 Expressible Numbers in Typical 32-Bit Formats

Density of Floating Point Numbers

- The numbers represented in floating-point notation are not spaced evenly along the number line, as are fixed-point numbers
- The possible values get closer together near the origin and farther apart as you move away.

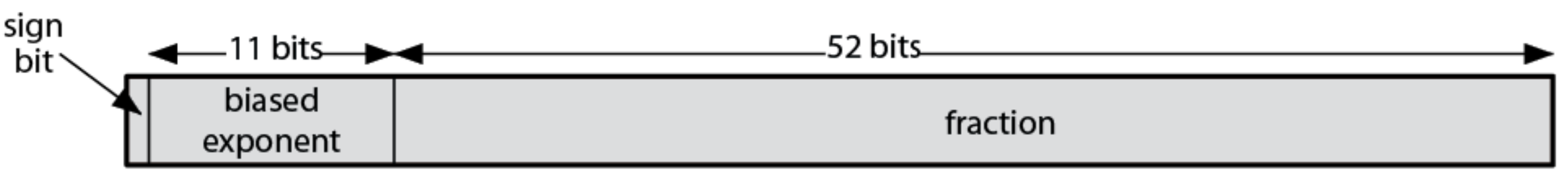


IEEE 754

- 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

IEEE 754 Formats

Table 9.4 Interpretation of IEEE 754 Floating-Point Numbers

	Single Precision (32 bits)			
	Sign	**Biased exponent**	**Fraction**	**Value**
positive zero	0	0	0	0
negative zero	1	0	0	–0
plus infinity	0	255 (all 1s)	0	∞
minus infinity	1	255 (all 1s)	0	$-\infty$
quiet NaN	0 or 1	255 (all 1s)	$\neq 0$	NaN
signaling NaN	0 or 1	255 (all 1s)	$\neq 0$	NaN
positive normalized nonzero	0	$0 < e < 255$	f	$2^{e-127}(1.f)$
negative normalized nonzero	1	$0 < e < 255$	f	$-2^{e-127}(1.f)$
positive denormalized	0	0	$f \neq 0$	$2^{e-126}(0.f)$
negative denormalized	1	0	$f \neq 0$	$-2^{e-126}(0.f)$

FP Arithmetic

- **Table 9.5 Floating-Point Numbers and Arithmetic Operations**

Floating Point Numbers	Arithmetic Operations
$X = X_S \times B^{X_E}$ $Y = Y_S \times B^{Y_E}$	$\left.\begin{aligned} X + Y &= \left(X_S \times B^{X_E - Y_E} + Y_S\right) \times B^{Y_E} \\ X - Y &= \left(X_S \times B^{X_E - Y_E} - Y_S\right) \times B^{Y_E} \end{aligned}\right\} X_E \le Y_E$ $X \times Y = (X_S \times Y_S) \times B^{X_E + Y_E}$ $\dfrac{X}{Y} = \left(\dfrac{X_S}{Y_S}\right) \times B^{X_E - Y_E}$

Examples:

$X = 0.3 \times 10^2 = 30$
$Y = 0.2 \times 10^3 = 200$

$X + Y = (0.3 \times 10^{2-3} + 0.2) \times 10^3 = 0.23 \times 10^3 = 230$
$X - Y = (0.3 \times 10^{2-3} - 0.2) \times 10^3 = (-0.17) \times 10^3 = -170$
$X \times Y = (0.3 \times 0.2) \times 10^{2+3} = 0.06 \times 10^5 = 6000$
$X \div Y = (0.3 \div 0.2) \times 10^{2-3} = 1.5 \times 10^{-1} = 0.15$

FP Arithmetic +/-

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

FP Addition & Subtraction Flowchart