컴퓨터 산술(9,10장)

성결대학교 컴퓨터공학부
최정열 교수
(passjay@sungkyul.ac.kr)

게으른 자여 네가 어느 때까지 누워 있겠느냐 네가 어느 때에 잠이 깨어 일어나겠느냐 좀더 자자, 좀더 졸자, 손을 모으고 좀더 누워있자 하면 네 빈궁이 강도 같이 오며 네 곤핍이 군사 같이 이르리라(잠언6:9-11)

수업 목표

- 산술논리연산장치의 구성요소와 그 기능을 이해한다
- 10진수, 2진수, 16진수들 간의 변환을 할 수 있다
- 2의 보수 표현을 설명할 수 있다
- 정수 산술 연산(가감승제)을 할 수 있다
- 부동 소수점 표현을 설명할 수 있다
- 부동 소수점 산술 연산이 어떻게 수행되는지 설명할 수 있다

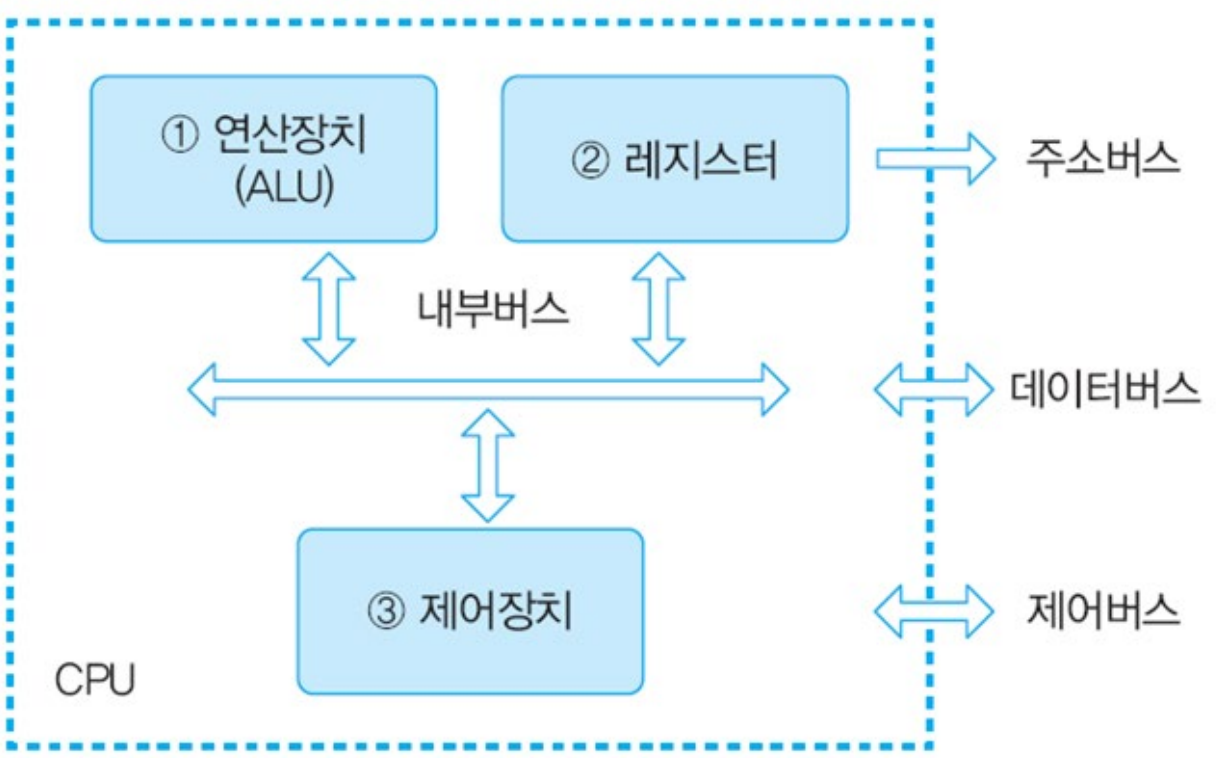
목차

- 산술논리연산장치
- 정수 표현
- 정수의 산술 연산
- 부동소수점 표현
- 부동소수점의 산술 연산
- 논리 연산

1. 산술논리연산장치(The Arithmetic and Logic Unit, ALU)

❑ 중앙처리장치(CPU)의 구성

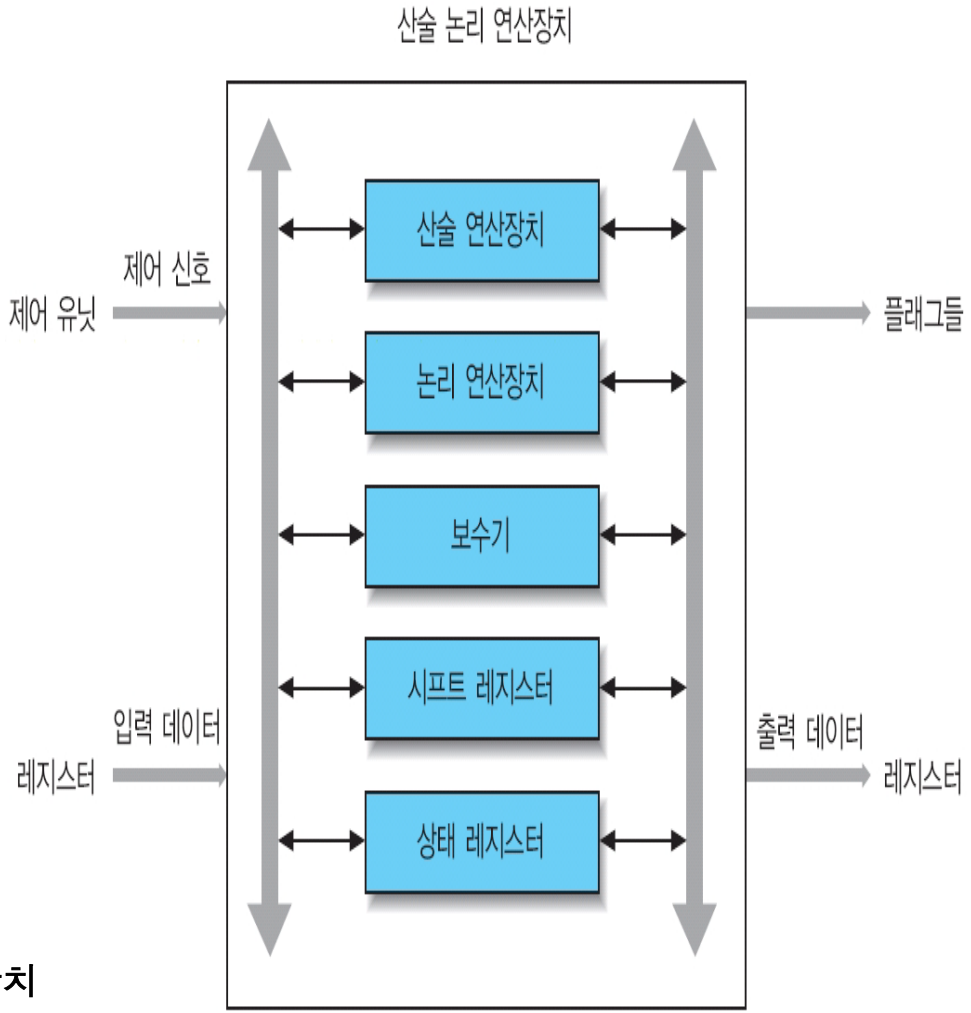
- _____
  - (Arithmetic & Logic Unit : ALU)
  - 산술 및 논리 연산을 수행
- _____
- _____



CPU의 구성 요소

ALU 내부 구성 요소

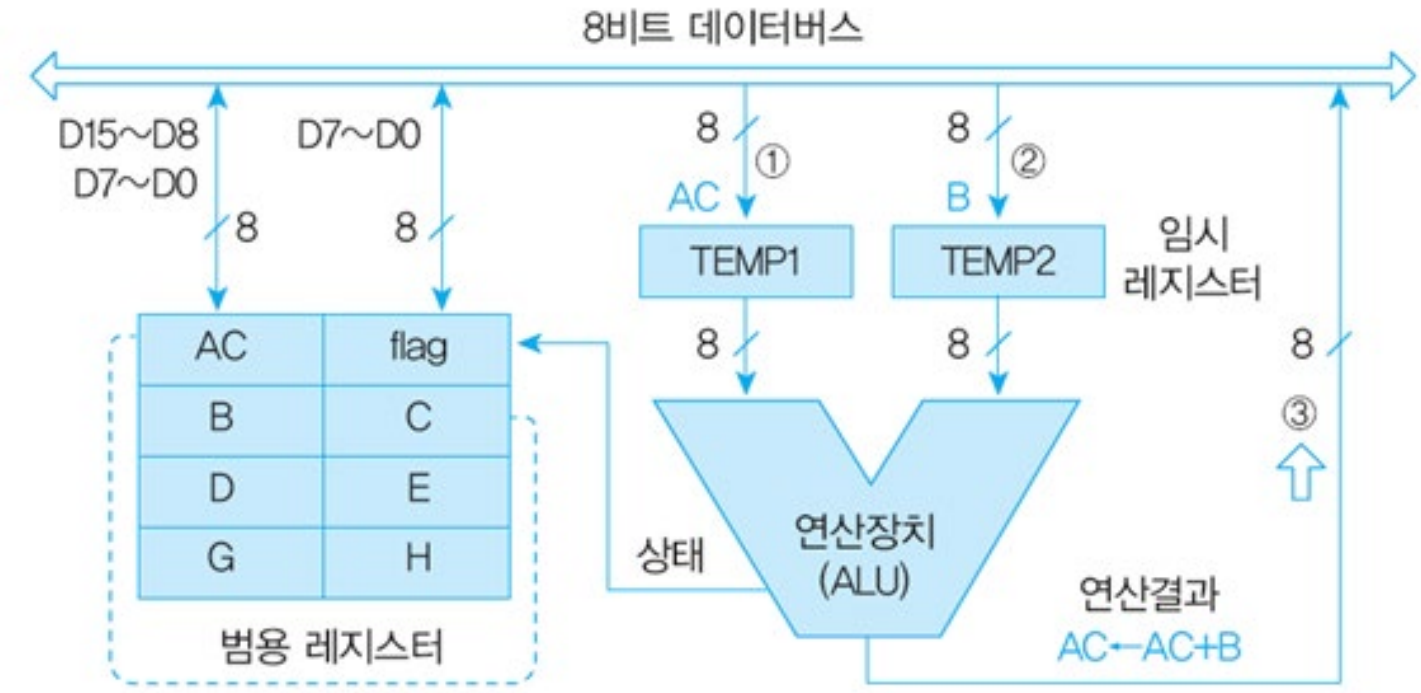
- **산술 연산장치**
  - +, -, ×, ÷등을 수행
- **논리 연산장치**
  - AND, OR, XOR, NOT 등을 수행
- **시프트 레지스터**
  - shift register
  - 비트들을 좌우측으로 이동
- **보수기(complementer)**
  - 2진 데이터를 2의 보수로 변환
  - 음수를 만드는 역할
- **상태 레지스터(status register)**
  - 연산 결과의 상태를 나타내는 플래그(flag)들을 저장



ALU의 구성 장치

ALU의 연산 동작 예

- **ADD AC, B ; AC ← AC + B**
  - t_1 : TEMP1 ← AC
  - t_2 : TEMP2 ← B
  - t_3 : AC ← TEMP1 + TEMP2



ALU의 내부 구조

2. 정수 표현

❑ 10진수의 개념

- 0 ~ 9의 10가지의 기호를 이용하여 수를 표현
- 10진수 $(724)_{10}$의 분석 : 700+20+4
- 10의 승수(10^N)로 표현
  - 10진수의 표시: $(724)_{10}$ = _____

❑ 2진수의 개념

- 0과 1만을 가지고 수를 표현
- 10진수의 관계는_____로 표현
  - 예) $(101101)_2$ =_____= $(45)_{10}$

❑ 16진수의 개념

- _____나누어 16진수로 표현
- 0~9, A, B, C, D, E, F의 기호를 사용
- 16진수와 10진수의 관계는 16진수를 16의 승수(16^N)로 표현
  - 예) $(F3)_{16}$ = _____ = $(243)_{10}$

진법 변환

❑ 10진수를 2진수로의 변환

- 연속적으로 2로 나눗셈을 수행하면서 얻어지는 나머지에 의해서 만들어진다.
- 예) $(41)_{10}$의 2진수로 변환
  - 41을 2로 연속해서 나눗셈
  - 생성된 나머지를 정렬
  - $(41)_{10} = (101001)_2$

❑ 2진수를 10진수로의 변환

- 2진수를 2^N로 표현
- 예) $(101001)_2 = 1\times2^5+0\times2^4+1\times2^3+0\times2^2+0\times2^1+1\times2^0 = 32+8+1 = (41)_{10}$

음수의 표현

- 음수를 표현하기 위해서 단어의 가장 맨 좌측 비트를 부호 비트로 사용
- 부호 비트를 사용한 음수의 표현 방법들
  - 부호 크기 표현(signed-magnitude representation)
  - 보수 표현(complement representation)

부호 크기 표현

- 부호가 있는 n비트의 2진수에서, _____이고 나머지 n-1개의 비트들은 _____를 나타낸다
  - 예) +9 = 0 0001001 -9 = 1 0001001

- 부호를 표현할 수 있는 가장 간단한 방법
- 덧셈과 뺄셈 연산을 수행하기 위해서는 부호 비트와 크기 부분을 별도로 처리해야 한다
- 0(zero)의 표현이 2개 존재하므로 표현할 수 있는 _____
  - 0 00000000 = +0 1 00000000 = -0
  - 즉, n비트 단어로 표현할 수 있는 수가 2^n이 아니라 ____임

보수 표현

- 2진수에서 1의 보수 표현: 모든 비트들을 반전시킨다
- 2진수에서 2의 보수 표현: 모든 비트들을 _____

- 2의 보수로 표현된 2진수(양수)를 십진수로 변환(부호 비트 $a_{n-1} = 0$)

$$A = a_{n-2} \times 2^{n-2} + a_{n-3} \times 2^{n-3} + \ldots + a_1 \times 2^1 + a_0 \times 2^0$$

- 2의 보수로 표현된 2진수(음수)를 십진수로 변환(부호 비트 $a_{n-1} = 1$)

$$A = -2^{n-1} + (a_{n-2} \times 2^{n-2} + a_{n-3} \times 2^{n-3} + \ldots + a_1 \times 2^1 + a_0 \times 2^0)$$

  - 또는, 먼저 _____붙인다

수의 표현 범위

- **2의 보수로 표현된 n비트 데이터의 표현 할 수 있는 수의 범위**

- **비트에 따른 수의 범위와 최대값과 최소값의 표현**
  - 8비트 2의 보수: -128 ≤ N ≤ +127

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

    -2^7 = 10000000 = -128

  - 16비트 2의 보수: -32768 ≤ N ≤ +32767

    $2^{15}-1$ = 011111111 11111111 = +32767

    -2^{15} = 100000000 00000000 = - 32768

예) 4비트 정수에 대한 다른 표현들

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	—

비트 확장(bit extension)

❑ 부호가 있는 데이터의 비트 수를 늘리는 연산

❑ 부호화-크기 표현

- 부호 비트를 맨 좌측으로 이동시키고, 그 외의 비트들은 0으로 채운다

+18 = 00010010 (8비트)
+18 = 0000000000010010 (16비트)
-18 = 00010010 (8비트)
-18 = 1000000000010010 (16비트)

❑ 2의 보수 표현

- 확장된 상위 비트들을 부호 비트와 _____채운다.

+18 = 00010010 (8비트)
+18 = 0000000000010010 (16비트)
-18 = 11101110 (8비트)
-18 = 1111111111101110 (16비트)

3. 정수의 산술 연산

- 음수화
- 덧셈 C = A + B
- 뺄셈 C = A - B
- 곱셈 C = A × B
- 나눗셈 C = A / B

음수화(negation)

부호-크기 표현

- 부호 비트를 반전시킨다

2의 보수를 사용

- 예) 2의 보수를 이용하여 -19을 이진수로 표현

$$
\begin{aligned}
+19: &\quad 00010011 \\
1\text{의 보수}: &\quad 11101100 \\
&+ \qquad\quad 1 \\
\hline
-19: &\quad 11101101
\end{aligned}
$$

2의 보수로 표현된 수들의 덧셈

- **두 수가 부호 없는 수인 것으로 간주하고 진행**
  - 연산의 결과가 양수이면 정상적인 2진수로 표현된 양수를 얻음
  - 연산의 결과가 음수이면 2의 보수 형태의 음수를 얻음

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)

결과값의 최상위 비트를 넘어가는 올림수 비트가 발생되면 무시한다

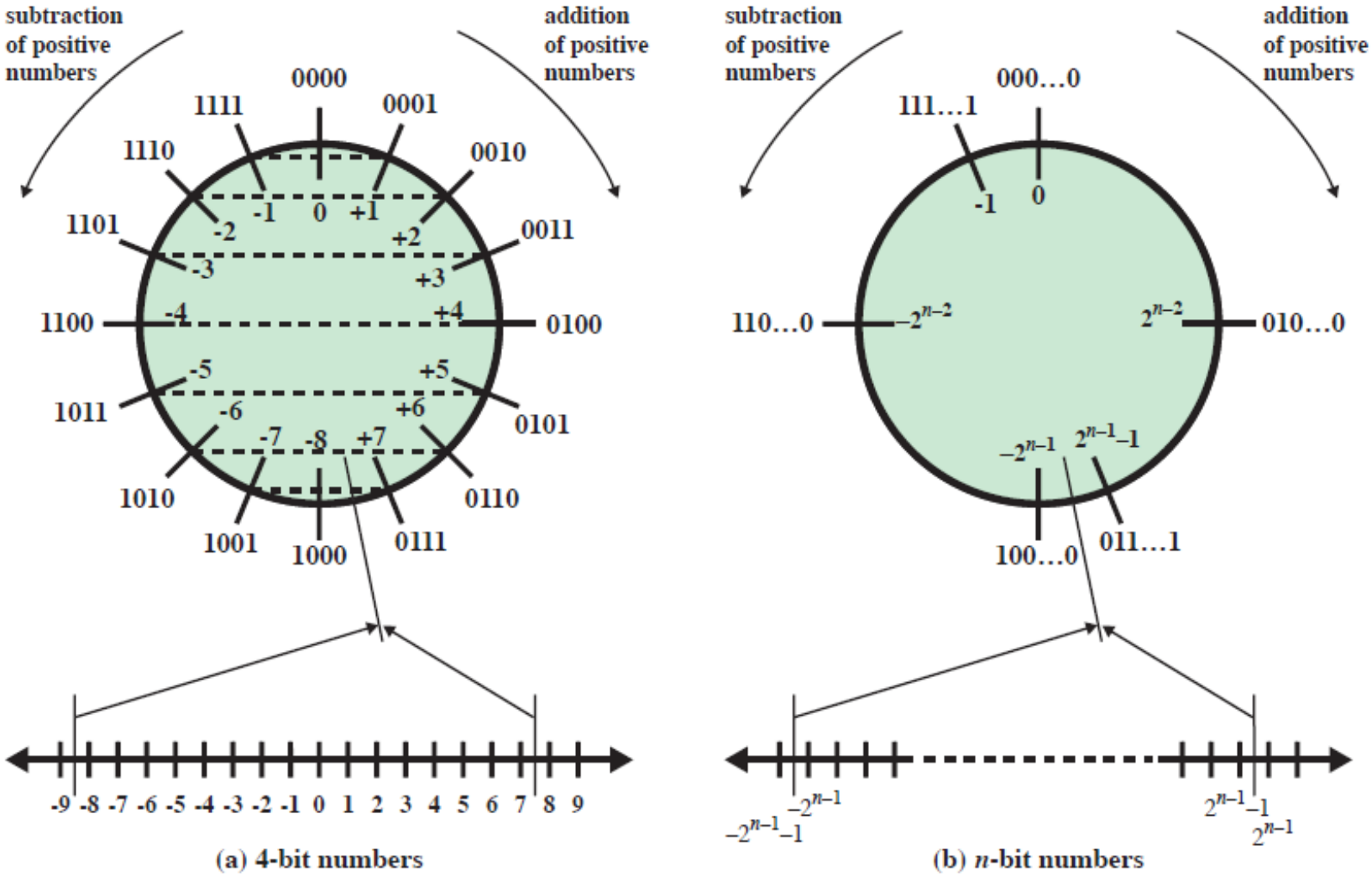
오버플로우: 어떤 덧셈에서 결과 값이 한 단어로 표현할 수 있는 범위를 초과

2의 보수로 표현된 수들의 뺄셈

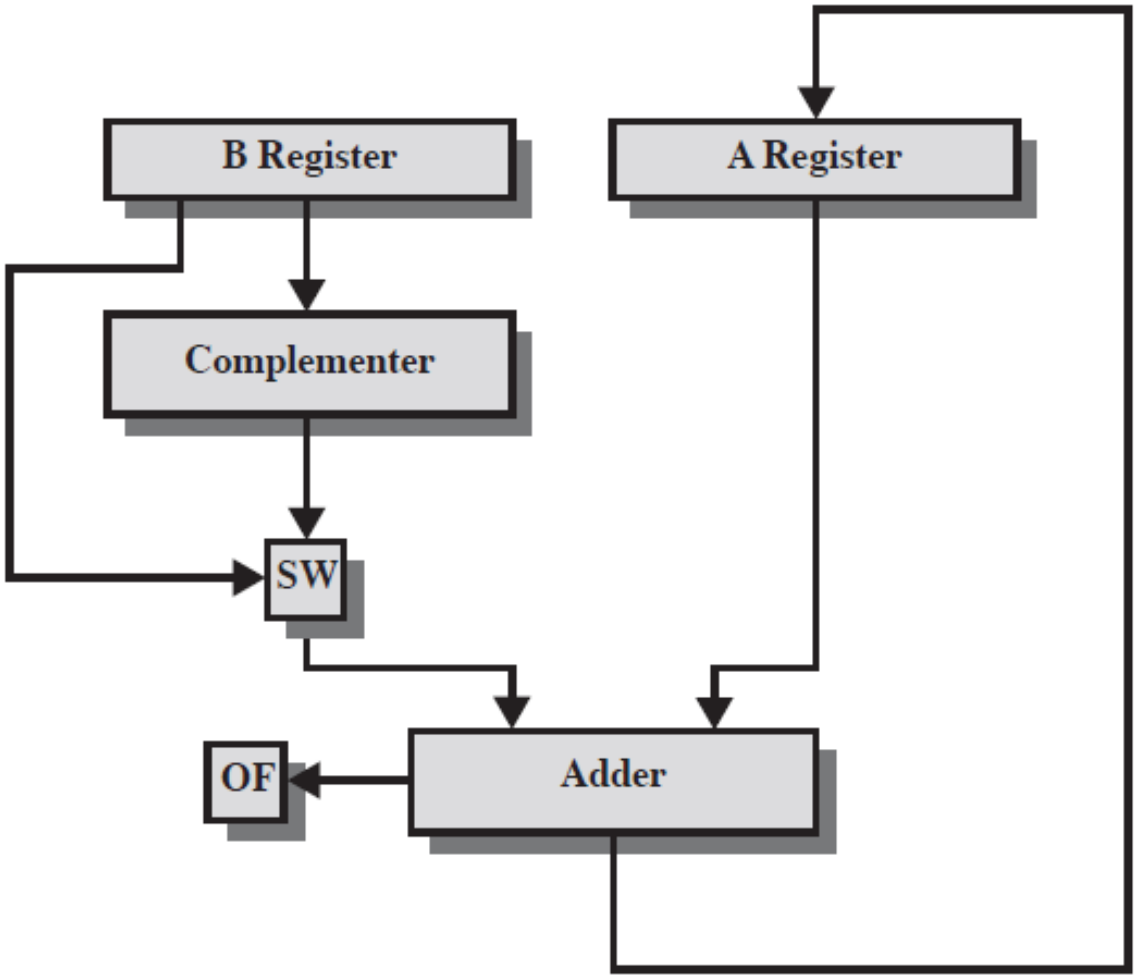
- 어떤 수(피감수: minuend)에서 다른 수(감수: subtrahend)를 빼기 위해서는 감수의 보수를 취한 뒤 그것을 피감수와 더한다

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

2의 보수 정수들의 기하학적 표현



덧셈/뺄셈 하드웨어 블록 다이어그램

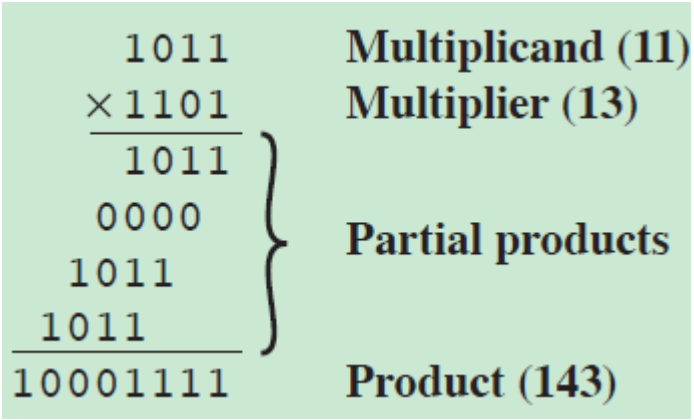


OF = overflow bit
SW = Switch (select addition or subtraction)

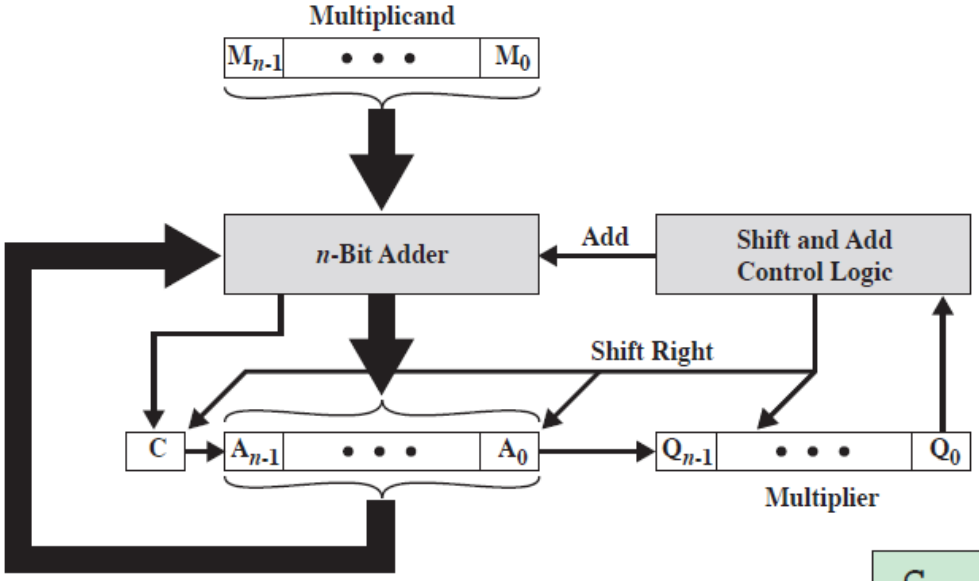
곱셈

- 곱하여 지는 수를 피승수(multiplicand, M), 곱하는 수를 승수(multiplier, Q)

- 부호가 없는 경우의 곱셈의 예



  - 4비트의 두 수가 서로 곱셈을 수행하면, 2배인 8비트의 길이의 결과를 출력



부호 없는 2진 곱셈을 위한 하드웨어 구현

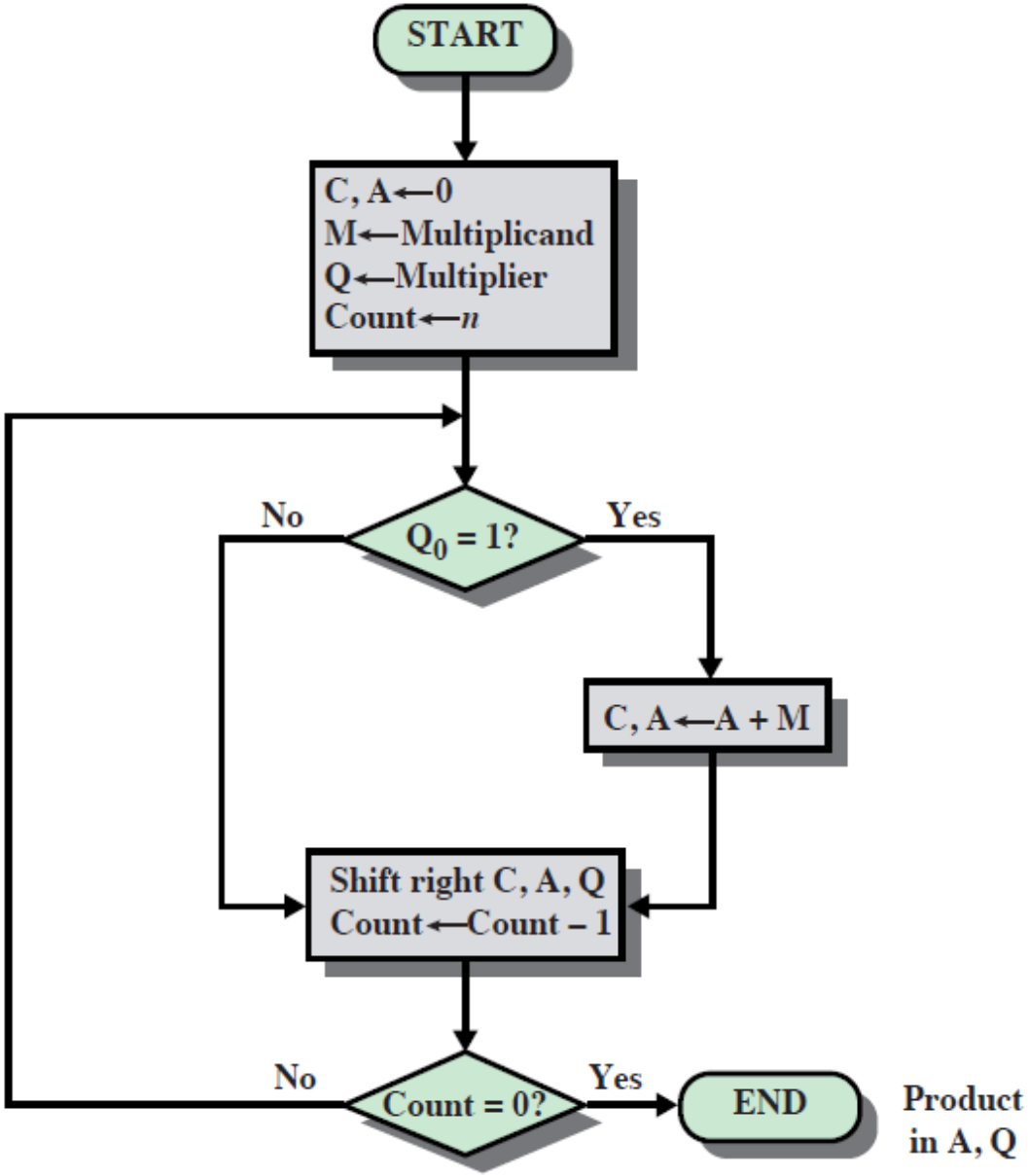
```
   1011      Multiplicand (11)
 × 1101      Multiplier (13)
   1011
  0000       Partial products
 1011
1011
10001111     Product (143)
```

M
Q

계산 예

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	

부호없는 2진 곱셈의 흐름도



부호있는 두 수의 곱셈: Booth 알고리즘

- 승수와 피승수를 Q와 M 레지스터에 저장
- Q 레지스터의 LSB인 Q_0 오른쪽에 Q_{-1} 라는 한 비트 레지스터 추가
- Q_0 와 Q_{-1} 가 같으면: A, Q, Q_{-1} 레지스터의 모든 비트를 우측으로 한 비트씩 산술시프트
- Q_0 와 Q_{-1} 가 0 1 이면: 피승수를 A에 더한다 & 우측 산술시프트
- Q_0 와 Q_{-1} 가 1 0 이면: A로부터 피승수를 뺀다 & 우측 산술시프트

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

결과

```
    0111
   ×0011      (0)
11111001      1−0
0000000       1−1
000111        0−1
00010101      (21)
```

(a) (7) × (3) = (21)

```
    0111
   ×1101      (0)
11111001      1−0
0000111       0−1
111001        1−0
11101011      (−21)
```

(b) (7) × (−3) = (−21)

```
    1001
   ×0011      (0)
00000111      1−0
0000000       1−1
111001        0−1
11101011      (−21)
```

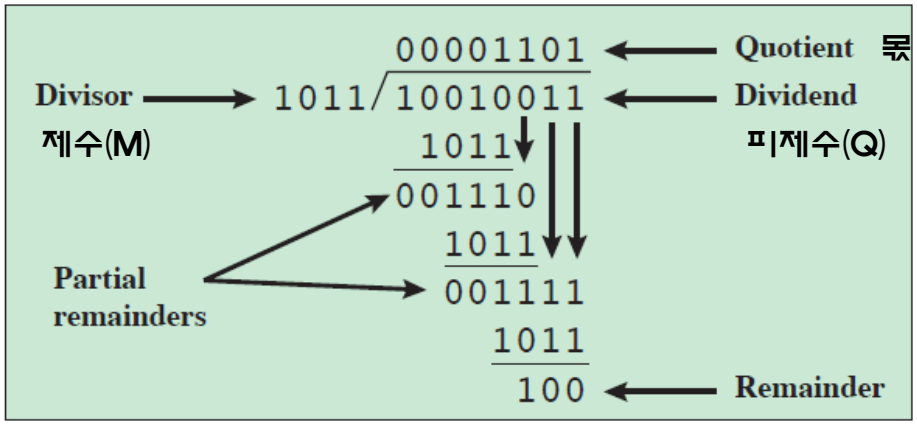
(c) (−7) × (3) = (−21)

```
    1001
   ×1101      (0)
00000111      1−0
1111001       0−1
000111        1−0
00010101      (21)
```

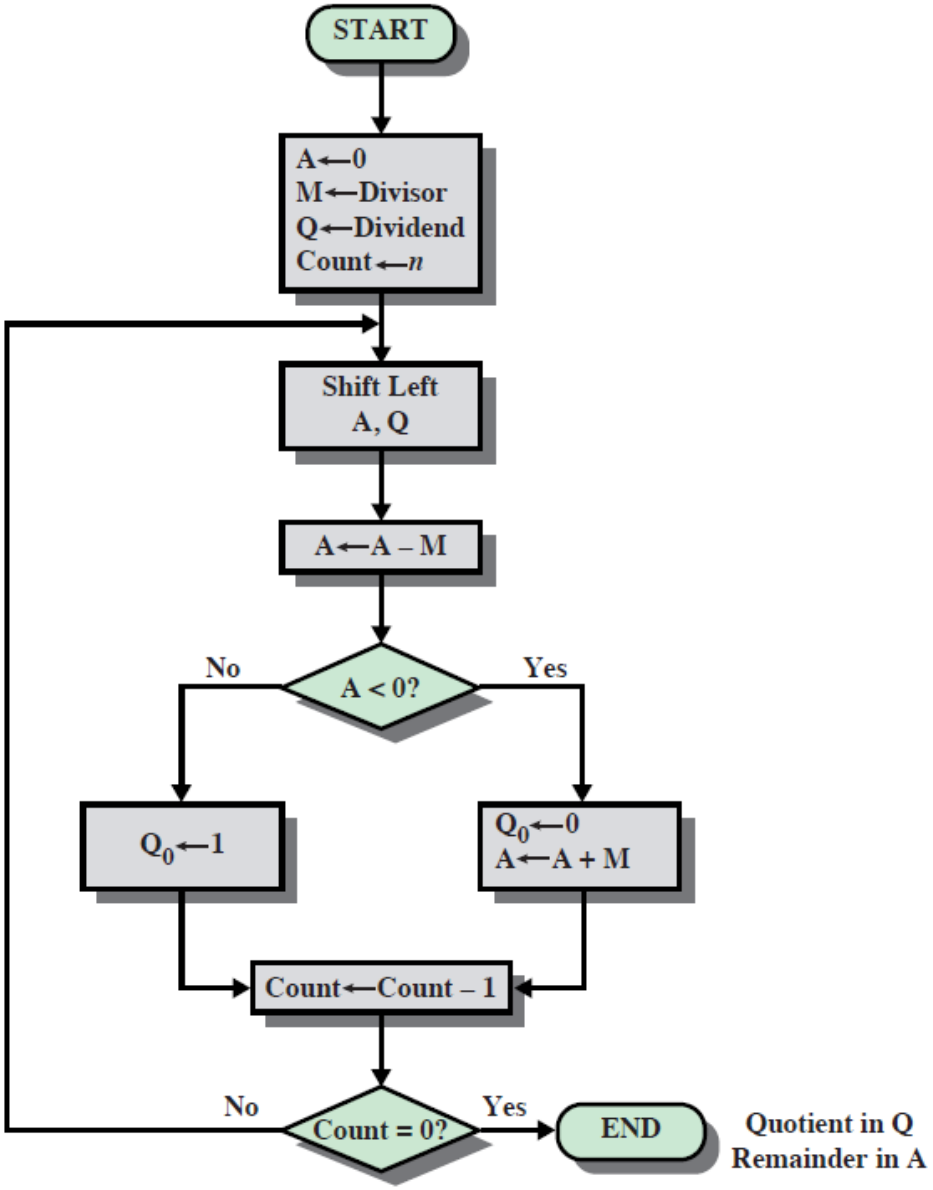
(d) (−7) × (−3) = (21)

Booth 알고리즘을 사용한 예

나눗셈



부호 없는 2진 정수의 나눗셈 예



부호 없는 2진 나눗셈을 위한 흐름도

A	Q	
0000	0111	Initial value
0000 1101 1101 0000	1110 1110	Shift Use twos complement of 0011 for subtraction Subtract Restore, set Q_0 = 0
0001 1101 1110 0001	1100 1100	Shift Subtract Restore, set Q_0 = 0
0011 1101 0000	1000 1001	Shift Subtract, set Q_0 = 1
0001 1101 1110 0001	0010 0010	Shift Subtract Restore, set Q_0 = 0

나머지 몫

2의 보수 나눗셈 예(7/3)

4. 부동소수점 표현

❑ 고정소수점(fixed-point) 표현 방식

$$1010.1010 = 2^3 + 2^1 + 2^{-1} + 2^{-3} = 10.625$$

- 매우 큰 수 및 매우 작은 수의 표현이 불가능

❑ 부동소수점(floating-point) 표현

- 과학적 표기의 지수(exponent)를 사용하여 소수점의 위치를 이동시킬 수 있는 표현 방법
- 표현의 범위가 확대
- 예) 십진수에 대한 부동소수점 표현

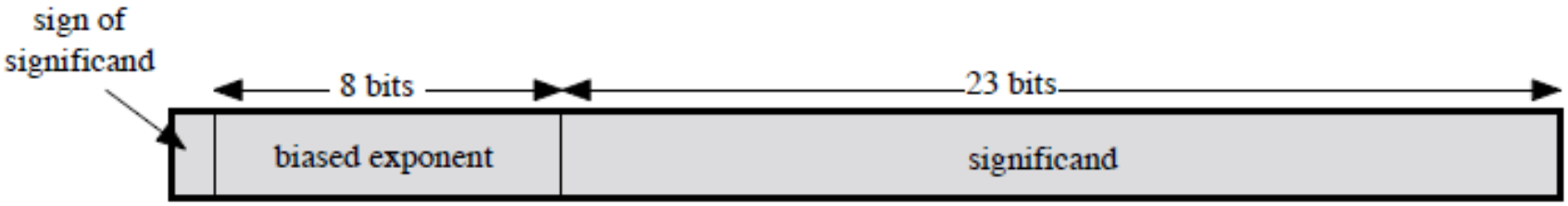
$176{,}000 = 1.76 \times 10^5$ $\quad$ $176{,}000 = 17.6 \times 10^4$

$0.000176 = 1.76 \times 10^{-4}$ $\quad$ $0.000176 = 17.6 \times 10^{-5}$

부동소수점 수의 표현법

$$\pm S \times B^{\pm E}$$

- 부호(sign), S: 가수(significand), B: 기수(base), E: 지수(exponent)



전형적인 32비트 부동소수점 형식

정규화된 표현

- 지수를 이용하는 부동소수점의 수는 지수의 값에 따라 동일한 수에 대한 부동소수점 표현이 여러 가지가 존재

0.1001×2^5 100.1×2^2 0.01001×2^6

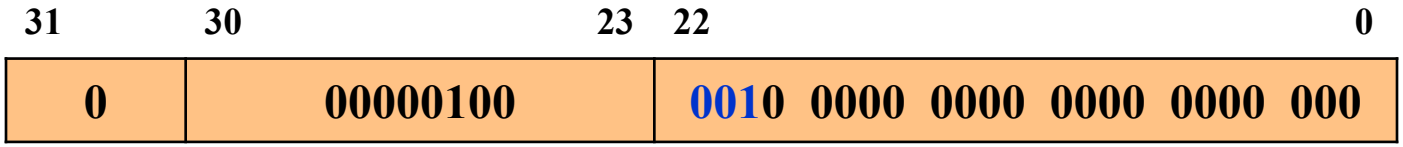
- 부동소수점의 수를 통일되게 표현하는 방법

$\pm 1.bbb...b \times 2^E$

- 예) 정규화된 표현의 단일 정밀도 부동소수점의 수 배열

1.001×2^4

  - 부호 비트 = 0(양수)
  - 지수(E) = 00000100
  - 가수(S) = 0010 0000 0000 0000 0000 000



❑ 가수

- 정규화된 표현에서 가수의 맨 좌측 비트는 항상 1로 정해져 있으므로 가수 필드에 저장할 필요가 없다
- 23비트를 이용하여 [1,2) 사이의 값을 가지는 24-비트 가수를 저장

❑ 기수

- 기수는 묵시적이며 모든 수에 동일하므로 저장할 필요가 없다

❑ 지수(바이어스된 표현)

- 지수는 부호를 가지므로 이에 대한 표현이 필요하다
- 바이어스된 표현(2^K-1)을 사용하여 음수가 아닌 부동소수점수의 크기를 비교하기 위해서 정수로 취급될 수 있다
- 바이어스된 값은 원래의 지수 비트 값에서 바이어스 값을 더해준다

- 예) 지수값: $(4)_{10} = (00000100)_2$
  - 127로 바이어스된 지수값: 00000100 + 01111111 = 10000011

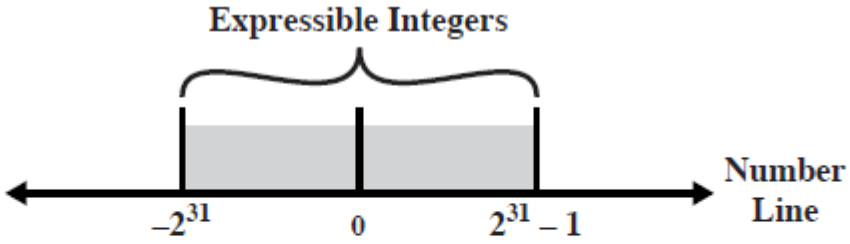
- 예) 10진수 -13.625를 32비트 부동소수점 형식으로 표현하면?

- 예) 32비트 부동소수점으로 표현된 아래 수에 대한 10진수는?

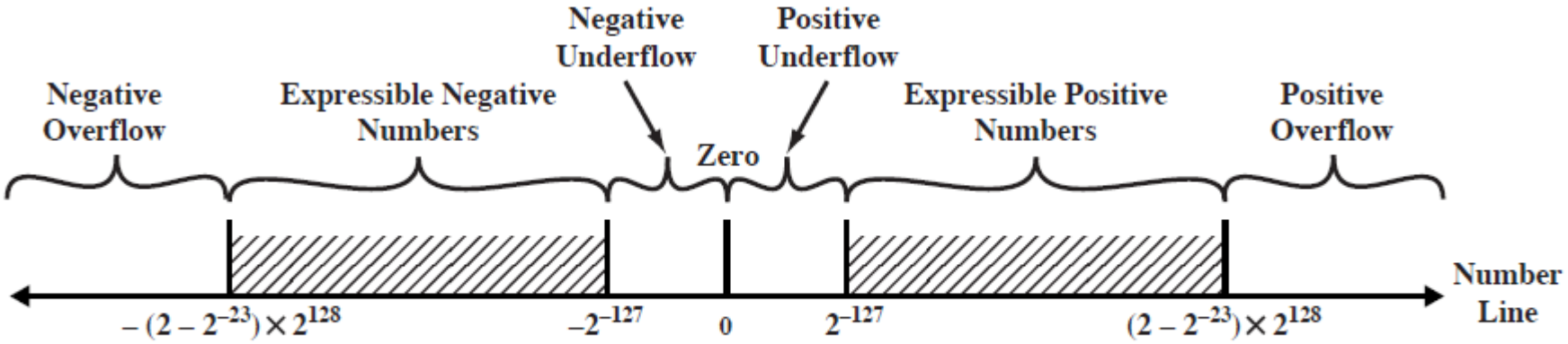
0 10000011 101 0000 0000 0000 0000 0000

전형적인 32-비트 형식으로 표현 가능한 수의 범위

- 정수: -2^{31}~$2^{31}-1$ (2^{32}가지의 수를 표현)
- 부동소수점수: $-(2-2^{-23})\times 2^{128}$~$-2^{-127}$, 2^{-127}~$(2-2^{-23})\times 2^{128}$

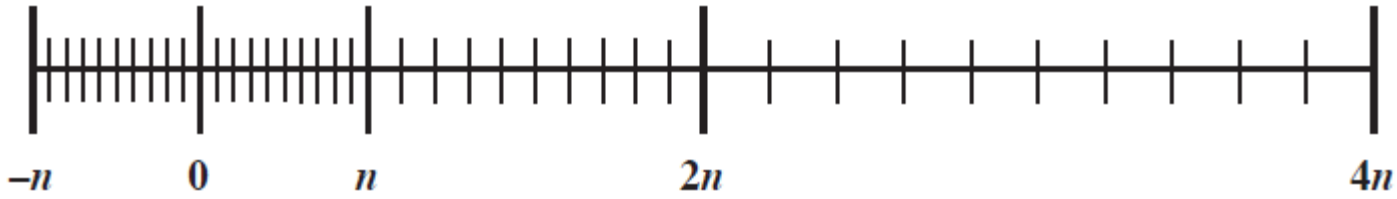


(a) Twos Complement Integers



(b) Floating-Point Numbers

부동소수점의 밀도

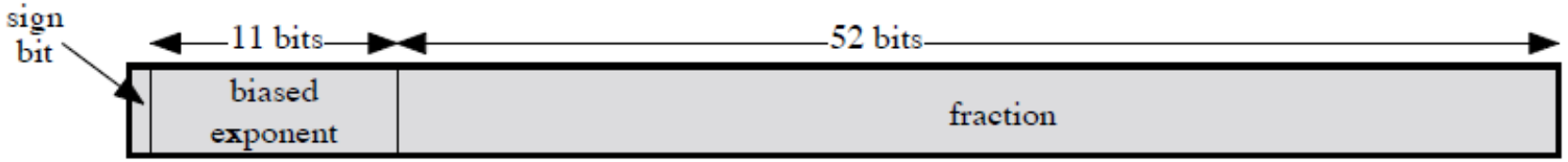


❑ 필드에 비트할당 문제

- 표현하는 수의 범위와 정밀도를 결정
- 지수(E) 필드의 비트 수가 늘어나면, 소수점을 이동시키는 범위가 커져서 표현 가능한 수의 범위가 확장
- 가수(S) 필드의 비트 수가 늘어나면, 이진수로 표현할 수 있는 수가 많아져서 정밀도(precision)가 증가

IEEE 754 formats

(a) Single format



(b) Double format

- 지수부(2^k-1 바이어스된 지수로 저장)
  - 지수부가 0 & 부호비트가 0이면: 0
  - 지수부가 0 & , 부호비트가 1이면: -0
    - 표현할 수 있는 가장 작은 수보다 더 작은 0이 아닌 수
  - 지수부가 제일 큰 값 & 부호비트가 0/1 = ∞/-∞
    - 표현할 수 있는 수의 범위를 벗어남

5. 부동소수점의 산술 연산

- **가수와 지수의 연산을 분리해서 수행**
- **덧셈과 뺄셈**
  - 지수를 같은 값으로 조정한 후, 가수들에 대하여 덧셈과 뺄셈을 수행
- **곱셈과 나눗셈**
  - 가수끼리는 곱셈과 나눗셈을 수행
  - 지수의 연산에서는 곱셈의 경우는 덧셈을 수행하고 나눗셈의 경우에는 뺄셈을 한다

부동소수점 수들의 연산 예

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

Examples:

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

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

부동소수점 수의 덧셈과 뺄셈

1. 0인지 검사한다
2. 두 수의 지수들을 같아지도록 가수의 자리수를 조정한다
3. 가수들 간에 덧셈/뺄셈을 수행
   - 가수들의 부호를 고려해 더해진다
4. 결과를 정규화한다(normalization)
   - 가장 왼쪽 비트가 0이 아닐 때까지 좌측으로 쉬프트시킨다

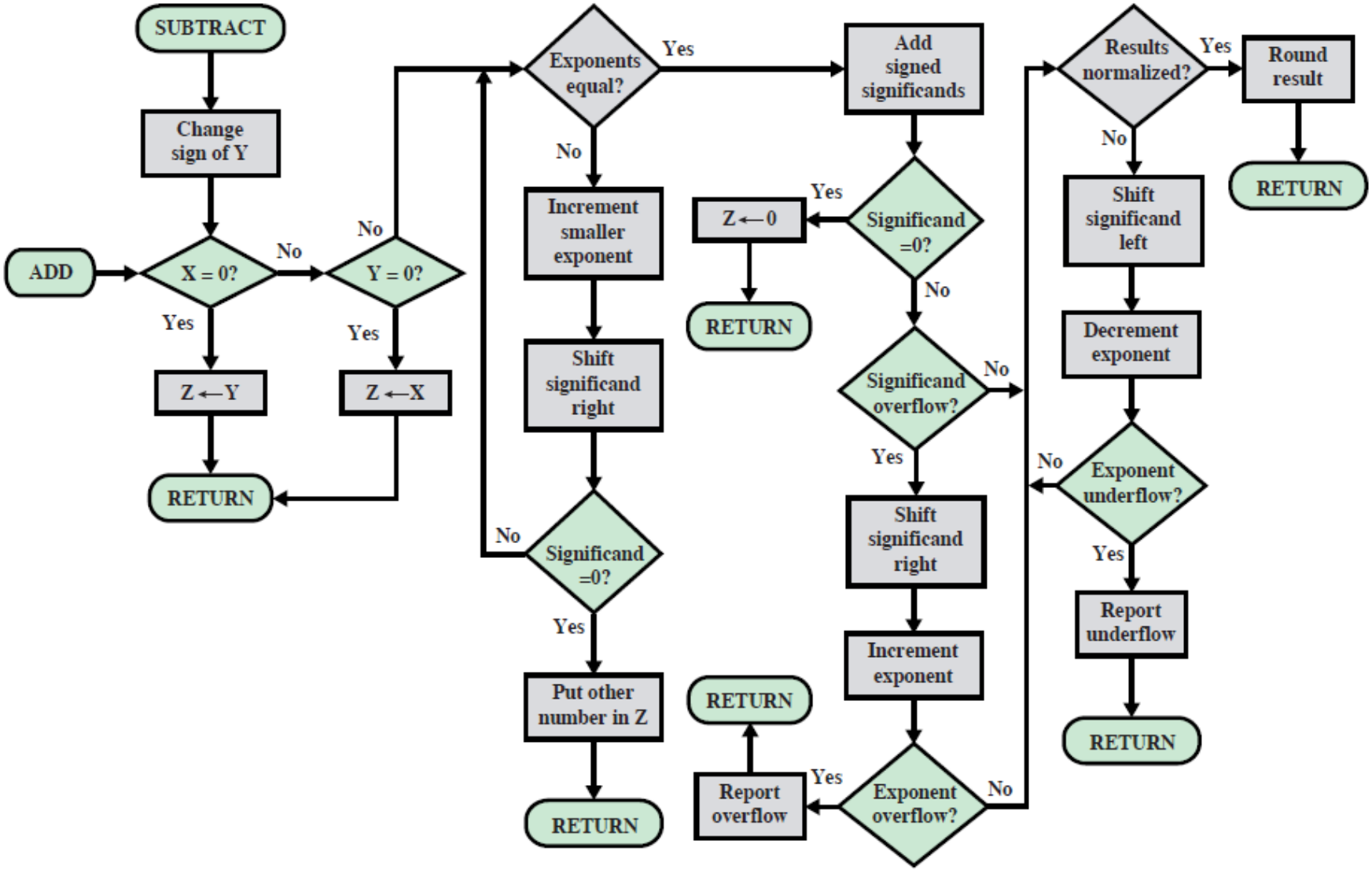
❑ 예) 이진수의 부동소수점 수의 덧셈

지수조정

$1.110010 \times 2^2 \Rightarrow 0.111001 \times 2^3$

$+1.111011 \times 2^3 \quad +1.111011 \times 2^3$

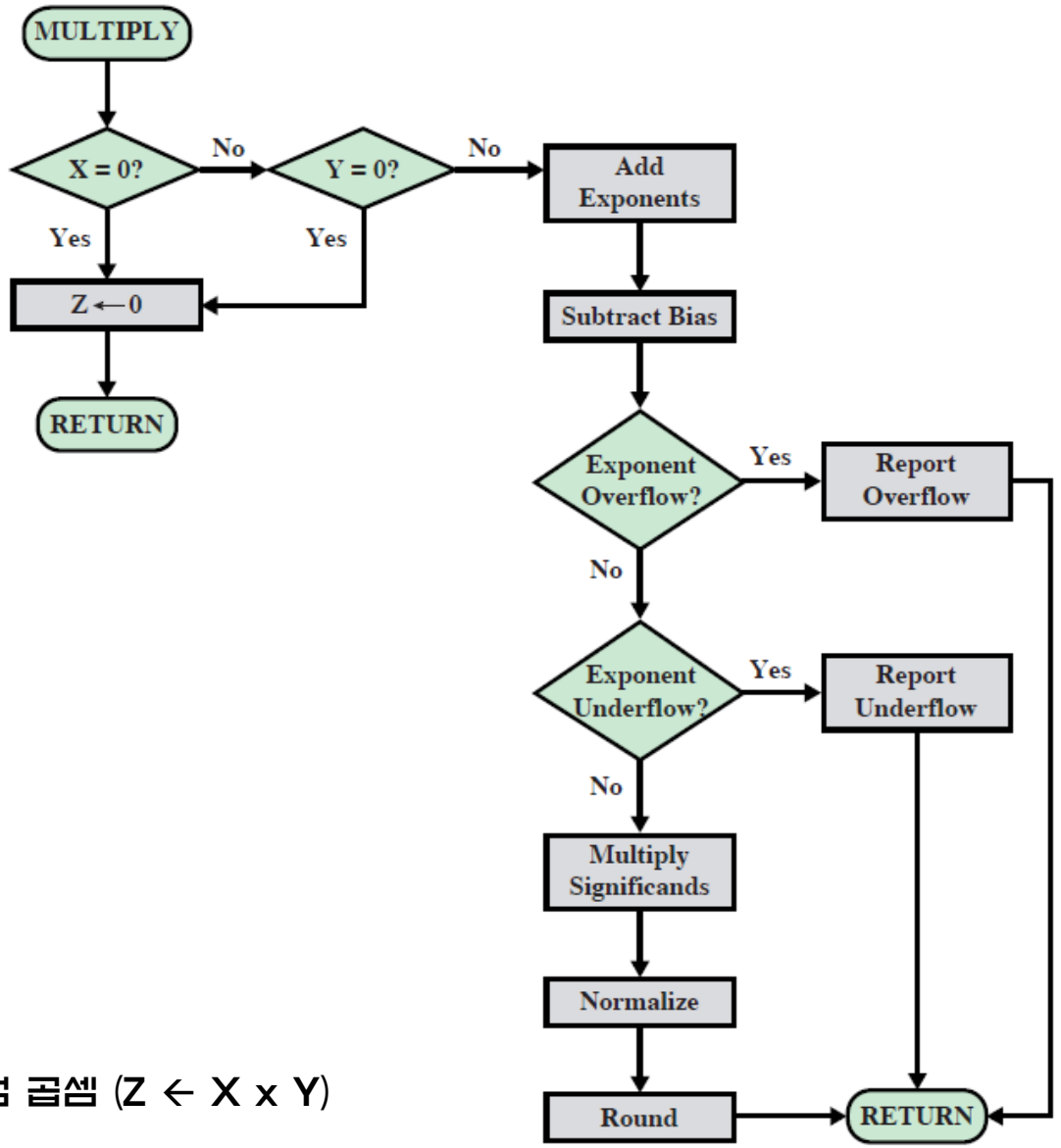
10.010100×2^3 정규화 $\Rightarrow 1.00101 \times 2^4$



부동소수점 덧셈과 뺄셈 연산 흐름도(Z = X ± Y)

부동소수점 수의 곱셈

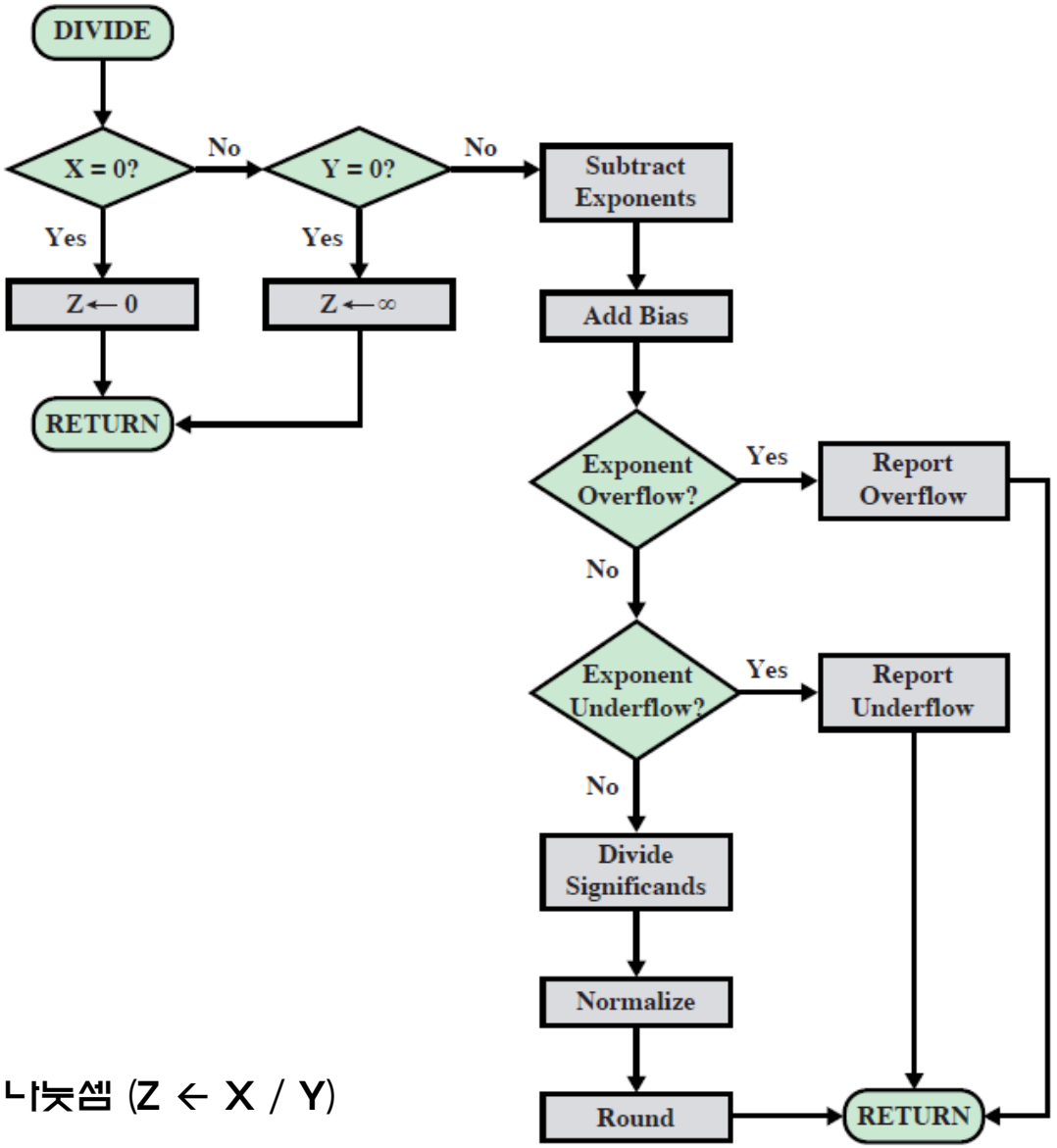
- 가수끼리는 곱셈 연산을 수행하고 지수끼리는 덧셈을 수행



부동소수점 곱셈 (Z ← X x Y)

부동소수점 수의 나눗셈

- 가수부분은 나눗셈 연산을 수행하고, 지수부분은 뺄셈 연산 수행



부동소수점 나눗셈 (Z ← X / Y)

6. 논리 연산

❑ 마스크(mask) 연산

- 원하는 비트들을 선택적으로 clear하는 데 사용하는 연산
- A 레지스터의 상위 4비트를 0으로 clear하는 경우의 예

A = 1 0 1 1 0 1 0 1	**(연산 전)**
B = 0 0 0 0 1 1 1 1	**마스크 (AND)연산**
A = 0 0 0 0 0 1 0 1	**(연산 후)**

비교(compare) 연산

- 두 데이터를 비교하는 연산으로 대응되는 비트들의 값이 같으면 해당 비트를 0으로 세트하고, 서로 다르면 해당 비트를 1로 세트
- 모든 비트들이 같은 경우, Z 플래그(zero 플래그)를 1로 세트

A = 1 1 1 0 0 0 0 1	(연산 전)
B = 0 1 0 1 1 0 0 1	비교(XOR)연산
A = 1 0 1 1 1 0 0 0	(연산 후)

산술적 쉬프트(arithmetic shift)

- 쉬프트 과정에서 부호 비트는 유지하고, 수의 크기를 나타내는 비트들만 쉬프트한다.
- 산술적 좌측-쉬프트
  - D4 (불변), D3 ← D2, D2 ← D1, D1 ← 0
- 산술적 우측-쉬프트
  - D4 (불변), D4 → D3 , D3 → D2, D2 → D1
- 예)

  A = 1 0 1 0 1 1 1 0 ; 초기 상태

  1 1 0 1 1 1 0 0 ; A의 산술적 좌측-쉬프트 결과

  1 1 0 1 0 1 1 1 ; A의 산술적 우측-쉬프트 결과

수고하고 무거운 짐 진 자들아
다 내게로 오라 내가 너희를
쉬게 하리라(마태복음11:28)