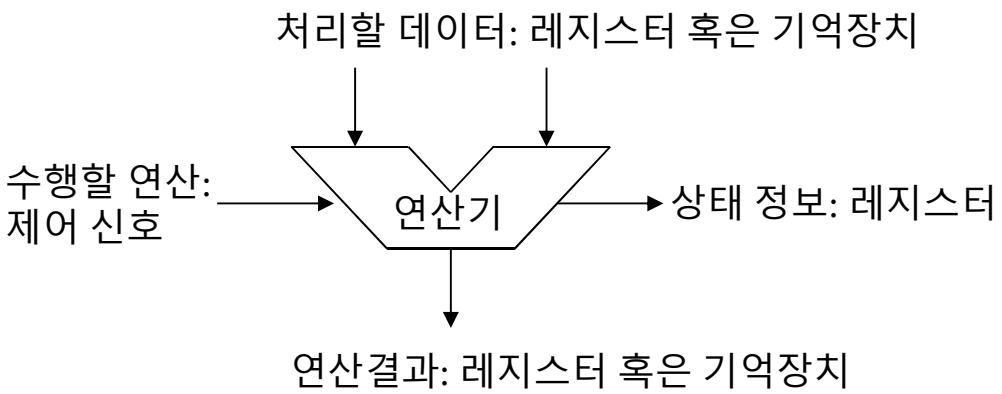
# 제5장 연산기

- 연산기 개요
- 정수
- 논리 연산
- 쉬프트 연산
- 정수 산술 연산
- IEEE 754 형식
- 실수 연산

# 연산기 개요

- 연산기 구조




- 연산의 종류
  - 단항 연산자 (unary operator)
    - -(음수 만들기), 1의 보수(NOT), 왼쪽/오른쪽 쉬프트, 증가, 감소 등
  - 이항 연산자 (binary operator)
    - 사칙 연산(+, -, x , /), 논리 연산(AND, OR, XOR), 비교(compare, test) 등

# 수의 형식과 연산 종류

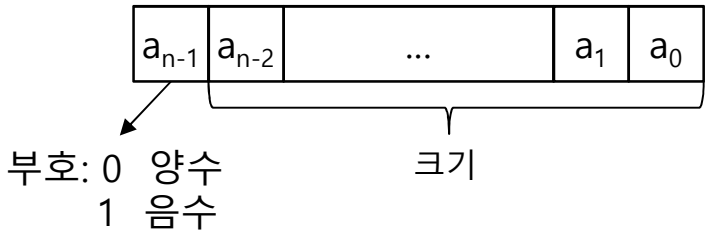
- 수의 표현과 연산 방법

<table>
<tr><th colspan="2">수의 형식</th><th>연산 방법</th></tr>
<tr><td rowspan="2">정수</td><td>부호 없는 수</td><td rowspan="2">논리 연산<br>쉬프트 연산<br>산술 연산</td></tr>
<tr><td>부호 있는 수</td></tr>
<tr><td colspan="2">실수</td><td>산술 연산</td></tr>
</table>

# 정수

- 정수
  - 부호 없는 수
    - 0을 포함한 양의 정수만을 표현
  - 부호 있는 수
    - 0, 양수, 음수
- 부호 있는 정수 표현 방법
  - signed magnitude (부호화 크기)
  - 1's complement (1의 보수)
  - 2's complement (2의 보수)

# 부호화 크기




$$N = (-1)^S \cdot \sum_{i=0}^{n-2} 2^i \cdot a_i$$

+19: 0001_0011 -19: 1001_0011

- 해석 방법: S = $a_{n-1}$: 부호 비트(sign bit)
- n 비트 표현 범위: $-(2^{n-1}-1)$ ~ $+(2^{n-1}-1)$
- 특징
  - 0이 두 개: 0000_0000, 1000_0000
  - 덧셈, 뺄셈시 부호를 별도로 고려해야 한다.

# 보수

- R 진법의 수 N에 대한 보수(complement)
  - (R-1)의 보수: $N + C_{R-1} = R^n - 1$
  - R의 보수 $N + C_R = R^n$
  - $C_R = C_{R-1} + 1$

- 예
  - 10 진수 457
    - 9의 보수: (1000-1) – 457 = 542
    - 10의 보수: 9의 보수 + 1 = 543
  - 2 진수 0011_1000
    - 1의 보수: (1_0000_0000 – 1) – 0011_1000 = 1100_0111
    - 2의 보수: 1의 보수 + 1 = 1100_1000

# 2진수의 보수

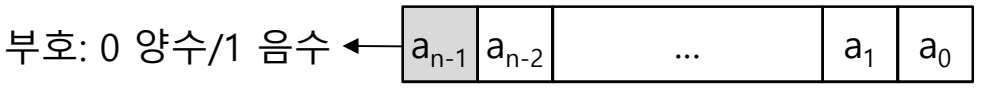
- 1의 보수: 0 ↔ 1 (NOT gate)
- 2의 보수: 1의 보수 + 1

- 예)
  - 0101_1010  1의 보수: 1010_0101
    2의 보수: 1010_0110

  - 0000_0000  1의 보수: 1111_1111 (0의 표현이 2 개)
    2의 보수: 0000_0000 (0의 표현이 1 개)

```
          11111111
        + 00000000
        ----------
제거 ←  (1)00000000
```

# 2의 보수 방식으로 부호있는 수 표현



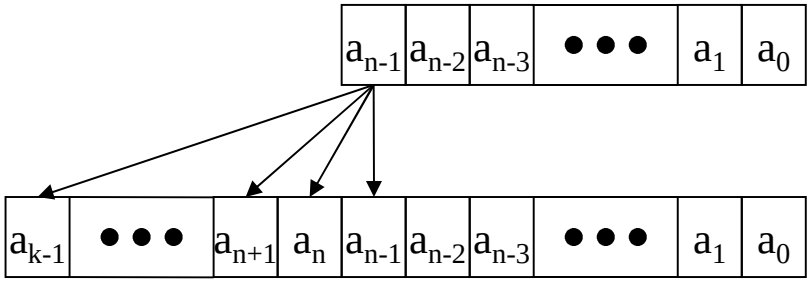

$$N = -2^{n-1} \cdot a_{n-1} + \sum_{i=0}^{n-2} 2^i \cdot a_i$$

- 수의 표현 범위: $-2^{n-1}$ ~ $+(2^{n-1}-1)$

| 비트 수 | 8 비트 | 16 비트 | 32 비트 |
|---|---|---|---|
| 표현 범위 | -128 ~ 127 | -32,768 ~ 32,767 | -2,146,483,648 ~ 2,146,483,647 |

- 예
  - 0110_1010  64 + 32 + 8 + 2 = 106
  - 1001_0110  -128 + 16 + 4+ 2 = -106

# 부호 확장 (sign expansion)




- 예: 8비트 → 16 비트
  - (122) 0111_1010 → 0000_0000_0111_1010
  - (-83) 1010_1101 → 1111_1111_1010_1101

# 논리 연산

- 논리 연산: 부호 없는 수로 취급
  - 단항 논리 연산: NOT
  - 이항 논리 연산: AND, OR, XOR

# NOT 연산

- 오퍼런드의 각 비트를 NOT
- 1의 보수를 구한 결과와 동일

- 예) R0 = 0010_1000일 때
  - R0 ← NOT R0 = 1101_0111

# AND 연산

- 마스크(mask) 연산
  - 특정 비트만 0으로 만든다.

R1 ← R1 AND R2, R2: mask pattern

```
   R1 = 1010_0110
AND R2 = 1110_0011
   R1 = 1010_0010
```

```
   R1 = 1010_0110
AND R2 = 0000_1111
   R1 = 0000_0110
```

# OR 연산

- 선택적 세트(selective set) 연산
  - 특정 비트만 1로 만든다.

R1 ← R1 OR R2

```
    R1 = 1010_0110
OR  R2 = 1110_0011
    R1 = 1110_0111
```

R1 = 'A' = 0100_0001

```
    R1 = 0100_0001
OR  R2 = 0010_0000
    R1 = 0110_0001
```

R1 = 'a'

# XOR 연산

- 선택적 보수 (selective complement) 연산
  - 특정 비트만 보수로 만든다.

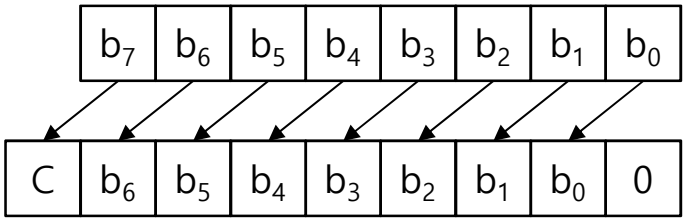
R1 ← R1 XOR R2

```
    R1 = 1010_0110
XOR R2 = 0000_1111
    R1 = 1010_1001
```

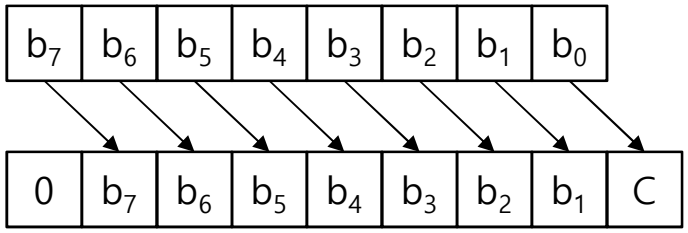
R1 ← R1 XOR R1

```
    R1 = 1010_0110
XOR R1 = 1010_0110
    R1 = 0000_0000
```

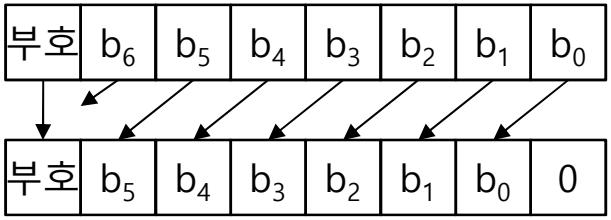
# 쉬프트 연산



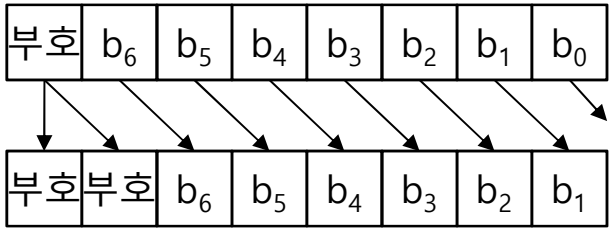

(a) 왼쪽 논리 쉬프트



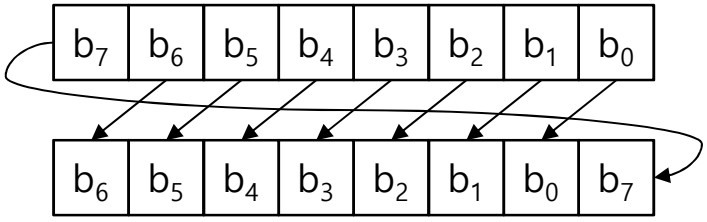

(b) 오른쪽 논리 쉬프트



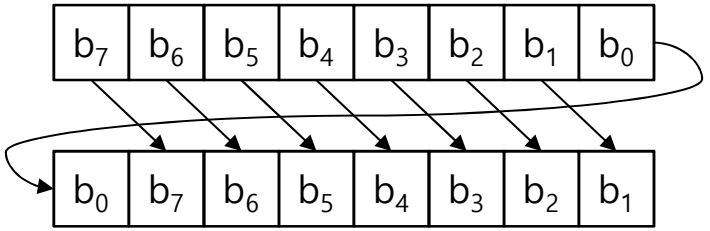

(a) 왼쪽 산술 쉬프트




(b) 오른쪽 산술 쉬프트




(a) 왼쪽 회전

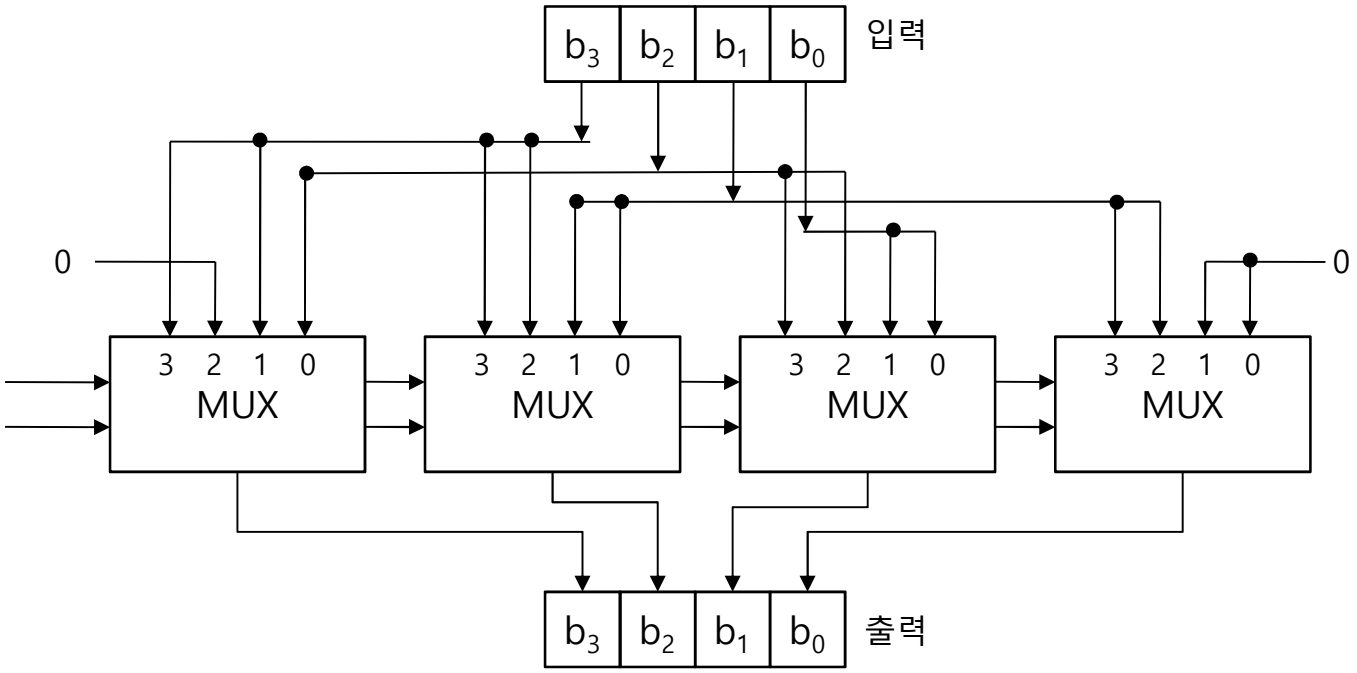



(b) 오른쪽 회전

# 쉬프트 회로

- 멀티플렉서를 사용한 쉬프트 회로

제어신호
00: 왼쪽 논리 쉬프트
01: 왼쪽 산술 쉬프트
10: 오른쪽 논리 쉬프트
11: 오른쪽 산술 쉬프트



# 정수 산술 연산

- 단항 연산
- 덧셈과 뺄셈
- 곱셈
- 나눗셈

# 단항 연산

- NEG : R ← -R
- INC : R ← R+1
- DEC : R ← R-1

# 덧셈과 뺄셈

- 덧셈과 뺄셈
  - 덧셈은 그대로 더하고, 뺄셈은 2의 보수를 더한다.
  - 결과를 부호 없는 수 혹은 정수로 해석할 수 있다.

```
  1000_1101        141        -115
+ 0110_0101      + 101      + 101
-----------      -----      -----
  1111_0010        242        -  14
```

(a) 이진수 덧셈 (b) 부호 없는 수 해석 (c) 정수 해석

```
  0100_1110        0100_1110      78
- 0001_1000      + 1110 1000    - 24
-----------      -----------    ----
                  10011_0110      54
```

(a) 이진수 뺄셈 (b) 2의 보수 더하기 (c) 정수 해석

# 오버플로우

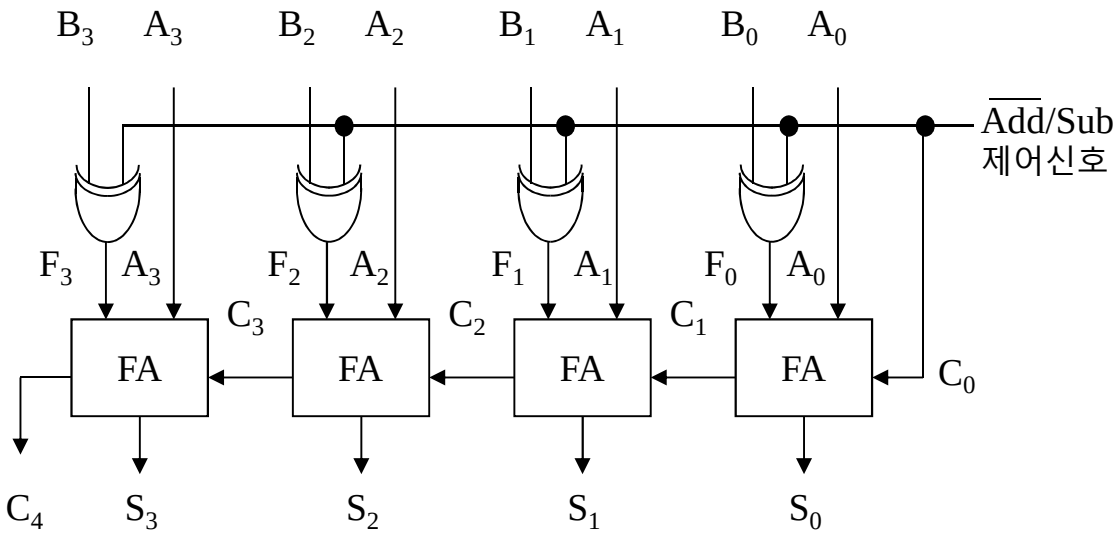
- 레지스터 크기 제한
- 연산 결과가 수의 표현 범위를 초과하는 현상
- Overflow = $C_n \oplus C_{n-1}$

```
C8C7                          C8C7
 01111 001                     11010 011
  0101_1001                     0101_1001
 +0010_1101                    +1101_0011
1000_0110 = -122                0010_1100 = 44

89+45 = 132 > 127               89-45 = 44
  overflow
```

```
C8C7                          C8C7
00101 111                     10000 111 ← 자리올림
 1010_0111                     1010_0111
+0010_1101                    +1101_0011
 1101_0100 = -44                0111_1010 = 122

-89+45 = -44                  -89-45 = -134 < -128
                                  overflow
```

# 병렬 덧셈기

- 4비트 덧셈기

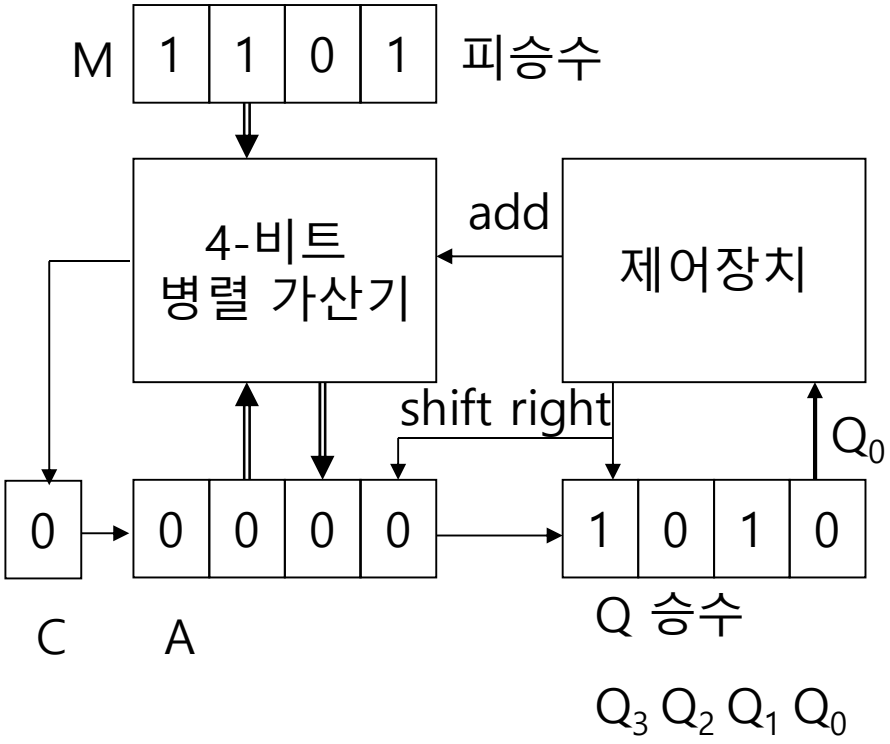


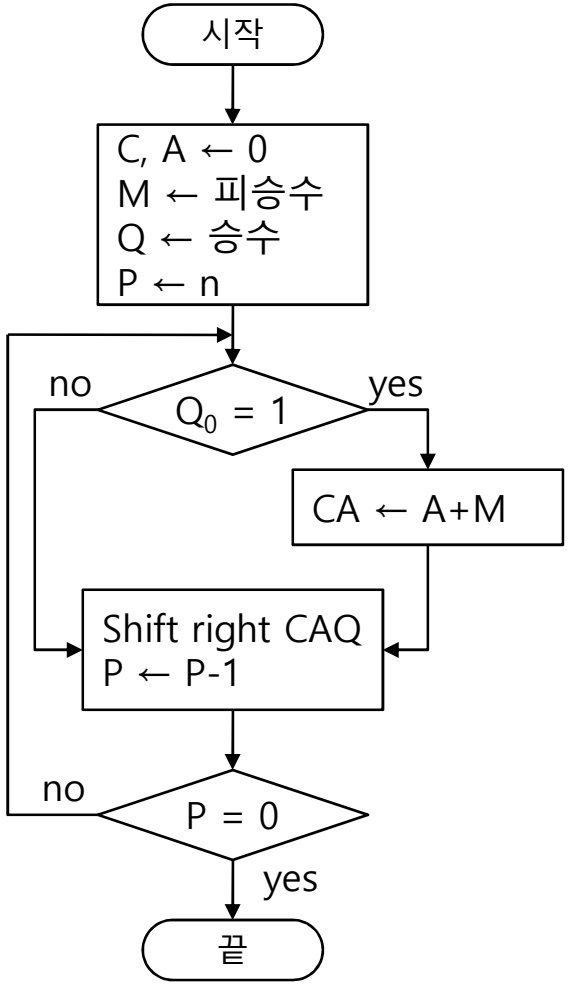
# 곱셈

- n-비트 곱셈
  - a x b = 곱 (product, 2n 비트)
  - a: 피승수(multiplicand), b: 승수(multiplier)

```
    1101    피승수(13)
  × 1010    승수(10)
  ------
    0000
   1101     부분 곱
  0000
 +1101
 --------
 10000010   곱(130)
```

# 부호 없는 수 곱셈 알고리즘



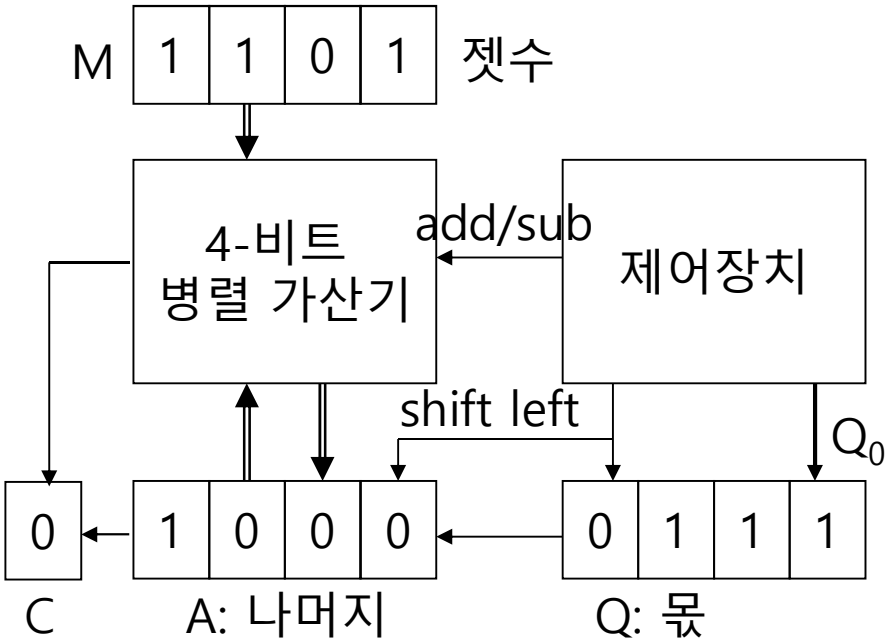



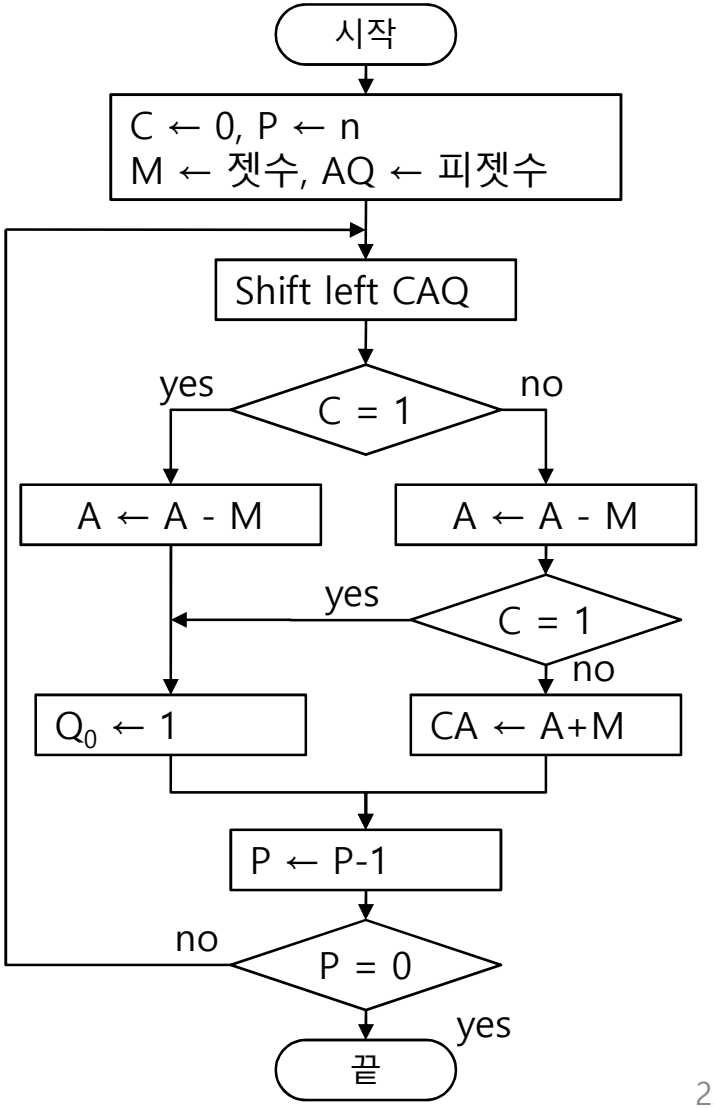
# 나눗셈

- 2n 비트 / n 비트 나눗셈 = n 비트 몫 … n 비트 나머지
  - a/b = 몫 (quotient) … 나머지(remainder)
  - a: 피젯수(dividend), b: 젯수(divisor)

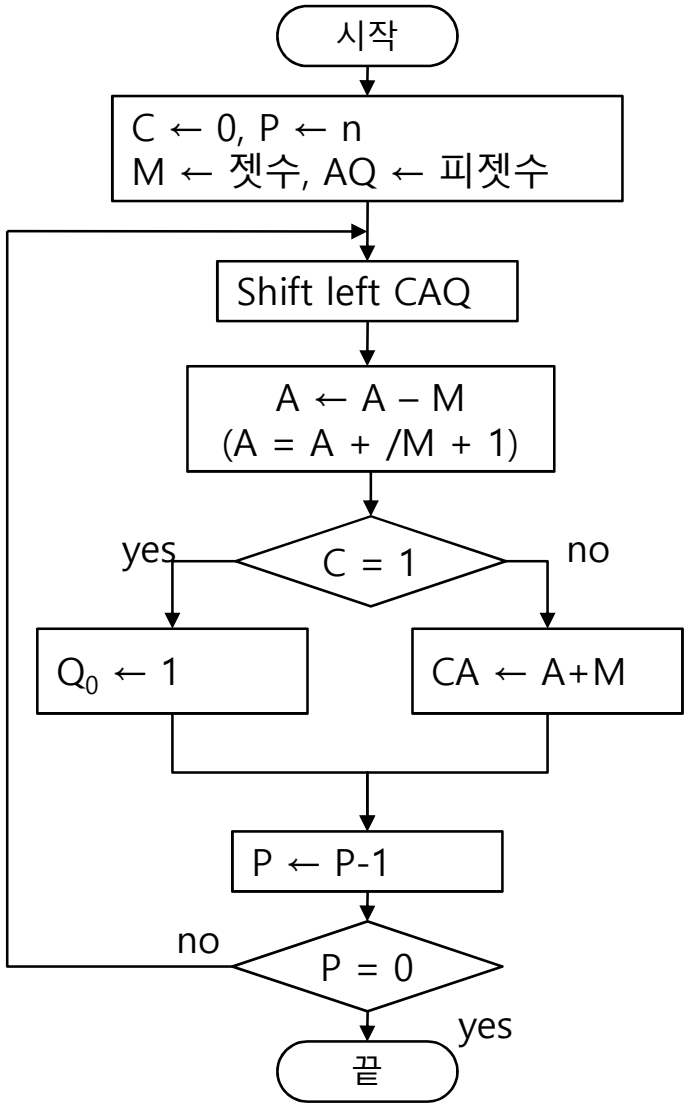
```
                      1010   몫(10)
젯수(13) 1101 ) 10000111   피젯수(135)
               - 1101
               ------
부분 나머지      00111
                - 0000
                ------
                 01111
                - 1101
                ------
                  00101
                - 0000
                ------
                  0101   나머지(5)
```

# 부호 없는 수 나눗셈 알고리즘



# 부호 없는 수 나눗셈 알고리즘




| 레지스터 | M | C | A | Q | P | |
|---|---|---|---|---|---|---|
| [초기 상태] | 1101 | 0 | 1000 | 0111 | 4 | |
| Shift CAQ | | 1 | 0000 | 1110 | 4 | |
| A = A-M | | 1 | 0011 | | | |
| C=1: $Q_0$ ←1 | | 1 | 0011 | 1111 | 3 | // P = P-1 |
| Shift CAQ | | 0 | 0111 | 1110 | 3 | |
| A=A-M | | 0 | 1010 | | | |
| C=0: A=A+M | | 1 | 0111 | 1110 | 2 | // P = P-1 |
| Shift CAQ | | 0 | 1111 | 1100 | 2 | |
| A=A-M | | 1 | 0010 | | | |
| C=1: $Q_0$ ←1 | | 1 | 0010 | 1101 | 1 | // P = P-1 |
| Shift CAQ | | 0 | 0101 | 1010 | 1 | |
| A=A-M | | 0 | 1000 | 1010 | 1 | |
| C=0: A=A+M | | 1 | 0101 | 1010 | 0 | // P = P-1 |
| | | | 나머지 몫 | | | |

# 실수

- 컴퓨터에서 실수를 표현 하는 방법
  - 과학 표기(scientific notation)
  - Floating-point number

- 부동소수점 표현
- IEEE 754 형식

# 부동 소수점 표현

- 과학적 표기(scientific notation)
  - ±significant x $Base^{exponent}$ (부호·가수 x 기수$^{지수}$)
  - significant = mantissa = fraction

- 10 진수 예
  - 976,000,000,000 = 9.76 x $10^{11}$
  - -0.00000000000976 = -9.76 x $10^{-12}$

- 2 진수 예→ 정규화(normalize) 필요
  - 0.1101 x $2^{2}$
  - 11.010 x $2^{0}$
  - 1.1010 x $2^{-1}$

# 가수 정규화(normalized mantissa)

- 가수 정규화
  - (1.bbb…b x $2^{\pm E}$) 형식이 되도록 지수 조정
  - 1.은 표현에서 생략 (항상 1.0은 존재)
  - 가수를 표현하는 비트 영역 최대 활용

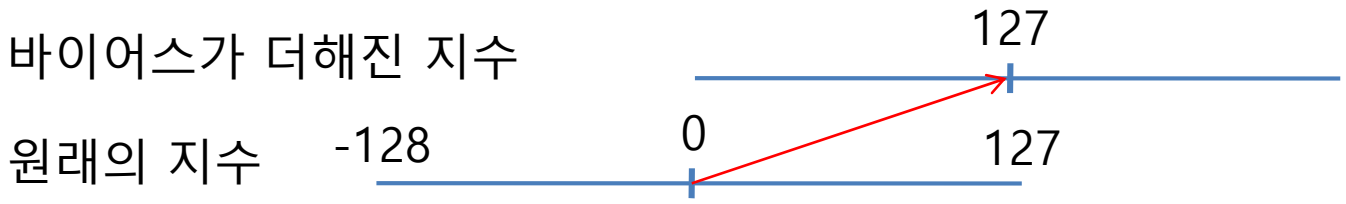
- 예) 가수를 8 비트를 사용할 때

  101.1010_1111 = 1.0110_1011_11 x $2^{+2}$

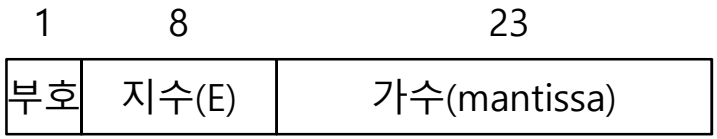
# 바이어스 지수(biased exponent)

- 바이어스 지수
  - 표현 = 지수 + bias (실제 지수 = 표현 – bias)
  - 이유? 간단하게 지수를 비교하기 위하여

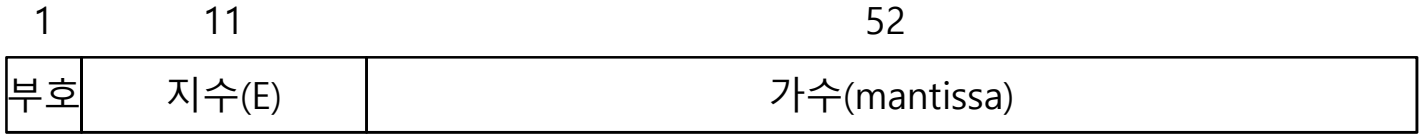
- 예) 101.101011
  - 정규화: 101.1010_11 = 1.0110_1011 x $2^2$
  - 지수의 값 = 2 = 0000_0010
  - 지수 표현 = 0000_0010 + 0111_1111 = 1000_0001.



# IEEE 754 형식




(a) 단정도 형식




(b) 배정도 형식

| Exponent | Significant | Value |
|---|---|---|
| 255 | ≠ 0 | NaN |
| 255 | 0 | $(-1)^s\infty$ |
| 0<e<255 | - | $(-1)^s 2^{e-127}(1.f)$ |
| 0 | ≠ 0 | $(-1)^s 2^{e-126}(0.f)$ |
| 0 | 0 | $(-1)^s 0$ |

Single precision

| Exponent | Significant | Value |
|---|---|---|
| 2047 | ≠ 0 | NaN |
| 2047 | 0 | $(-1)^s\infty$ |
| 0<e<2047 | - | $(-1)^s 2^{e-1023}(1.f)$ |
| 0 | ≠ 0 | $(-1)^s 2^{e-1022}(0.f)$ |
| 0 | 0 | $(-1)^s 0$ |

Double precision

# 수 범위

*Floating Point Range*

| | Denormalized | Normalized | Approximate Decimal |
|---|---|---|---|
| **Single Precision** | $\pm 2^{-149}$ to $(1-2^{-23})\times 2^{-126}$ | $\pm 2^{-126}$ to $(2-2^{-23})\times 2^{127}$ | $\pm \approx 10^{-44.85}$ to $\approx 10^{38.53}$ |
| **Double Precision** | $\pm 2^{-1074}$ to $(1-2^{-52})\times 2^{-1022}$ | $\pm 2^{-1022}$ to $(2-2^{-52})\times 2^{1023}$ | $\pm \approx 10^{-323.3}$ to $\approx 10^{308.3}$ |

# IEEE 754 예

- 다음 single precision 형식이 나타내는 값은?

1100_0001_0101_0000_0000_0000_0000_0000

(부호) 1
(지수) 10000010
(가수) 101_0000_0000_0000_0000_0000

부호: 1
지수: e = 1000_0010 = $130_{10}$
가수: M = 1.101

$N = (-1)^1 \times 1.101 \times 2^{130-127}$
$= -1.101 \times 2^3$
$= -1101 = -13_{10}$

# IEEE 754 가장 작은 수와 큰 수

- 가장 작은 수 0000_0000_0000_0000_0000_0000_0000_0001

  (부호) 0

  (지수) 0

  (가수) 000_0000_0000_0000_0000_0001

  크기: $(-1)^s \times 2^{e-126} \times 0.f = (-1)^0 \times 2^{-126} \times 0.000\_0000\_0000\_0000\_0000\_0001$

  $= (-1)^0 \times 2^{-126} \times 2^{-23}$

  $= 1 \times 2^{-149}$

- 가장 큰 수 0111_1111_0111_1111_1111_1111_1111_111

  (부호) 0

  (지수) 254

  (가수) 111_1111_1111_1111_1111_1111

  크기: $(-1)^s \times 2^{254-127} \times 1.f = (-1)^0 \times 2^{127} \times 1.111\_1111\_1111\_1111\_1111\_1111$

  $= (-1)^0 \times 2^{127} \times ( 2 - 2^{-23} )$

  $= ( 2 - 2^{-23} ) \times 2^{127} = ( 1 - 2^{-24} ) \times 2^{128}$

# 실수 연산

- 실수 연산
  - 덧셈과 뺄셈
  - 곱셈과 나눗셈

- 실수 연산 예외
  - 지수 오버플로우(exponent overflow)
    - 양의 지수값이 최대 지수값을 초과한 경우
    - 연산 결과는 $+\infty$ 또는 $-\infty$로 표시된다.
  - 지수 언더플로우(exponent underflow)
    - 음의 지수값이 최소 지수값보다 적은 경우
    - 연산 결과는 0으로 표시된다.
  - 가수 오버플로우(significant overflow)
    - 최상위 비트에서 자리올림수가 발생한 경우
    - 재정렬((realignment): 가수를 왼쪽으로 쉬프트하고 지수를 조정
  - 가수 언더플로우(significant underflow)
    - 가수를 조정할 때 가수가 오른편으로 밀려 잘려나가는 경우
    - 연산 결과의 최하위 비트를 반올림한다.

# 덧셈과 뺄셈

- 다음 두 수를 더하는 경우에

  $x = 1.1111 \times 2^{-1}$

  $y = 1.1111 \times 2^{1}$

$$
\begin{array}{rl}
x = & 0.0111\boxed{11} \times 2^{1} \\
y = & \boxed{1.1111} \times 2^{1} \\
\hline
x+y = & 10.0110 \times 2^{1} = 100.011
\end{array}
$$

(a) 작은 수를 큰 수에 맞춤

$$
\begin{array}{rl}
x = & \boxed{1.1111} \times 2^{-1} \\
y = & 11\boxed{1.1100} \times 2^{-1} \\
\hline
x+y = & 10.1011 \times 2^{-1} = 1.0101
\end{array}
$$

(b) 큰 수를 작은 수에 맞춤

(a) 정확성에는 문제가 있지만 심각하지 않음

(b) y 의 상위 비트가 없어져서 문제가 심각함.

# 덧셈과 뺄셈

- $(x = S_x * 2^{Ex}) \pm (y = S_y * 2^{Ey})$
  - 제로 검사
    - x와 y가 0인지 검사한다
    - 만일 둘 중 하나가 0이면, 덧셈과 뺄셈을 수행할 필요가 없다.
  - 정렬
    - 지수를 비교하여 작은 수의 지수를 큰 수의 지수와 같아지도록 조정
    - 작은 수의 지수를 증가시키고, 가수를 오른쪽으로 쉬프트 한다.
  - 가수 덧셈/뺄셈
    - 가수를 서로 더하거나 뺀다.
  - 정규화
    - 결과를 정규화 한다.

# 곱셈과 나눗셈

$$x = S_x * 2^{Ex}, \quad y = S_y * 2^{Ey}$$

$$x * y = (S_x * S_y) * 2^{Ex+Ey} \qquad x / y = (S_x / S_y) * 2^{Ex-Ey}$$

## 곱셈

- 제로 검사
  - 만일 x와 y 둘 중 하나가 0이면, 곱은 0이다.
- 지수 조정
  - 지수 $E_x$와 $E_y$를 더하고 바이어스를 뺀다.
- 가수 곱셈
  - 부호를 고려하여 $S_x$와 $S_y$를 곱한다.
- 정규화
  - 곱셈 결과를 정규화 한다.

## 나눗셈

- 제로 검사
  - 만일 x가 0이면, 결과는 0이고, y가 0이면 결과는 NaN으로 설정하고 계산을 종료한다.
- 지수 조정
  - 지수 $E_x$에서 $E_y$를 빼고 바이어스를 더한다.
- 가수 나눗셈
  - 부호를 고려하여 $S_x$에서 $S_y$를 나눈다.
- 정규화
  - 나눗셈 결과를 정규화 한다.