

# 제3장 컴퓨터 산술과 논리 연산

3.1 ALU의 구성 요소

3.2 정수의 표현

3.3 논리 연산

3.4 시프트 연산

3.5 정수의 산술 연산

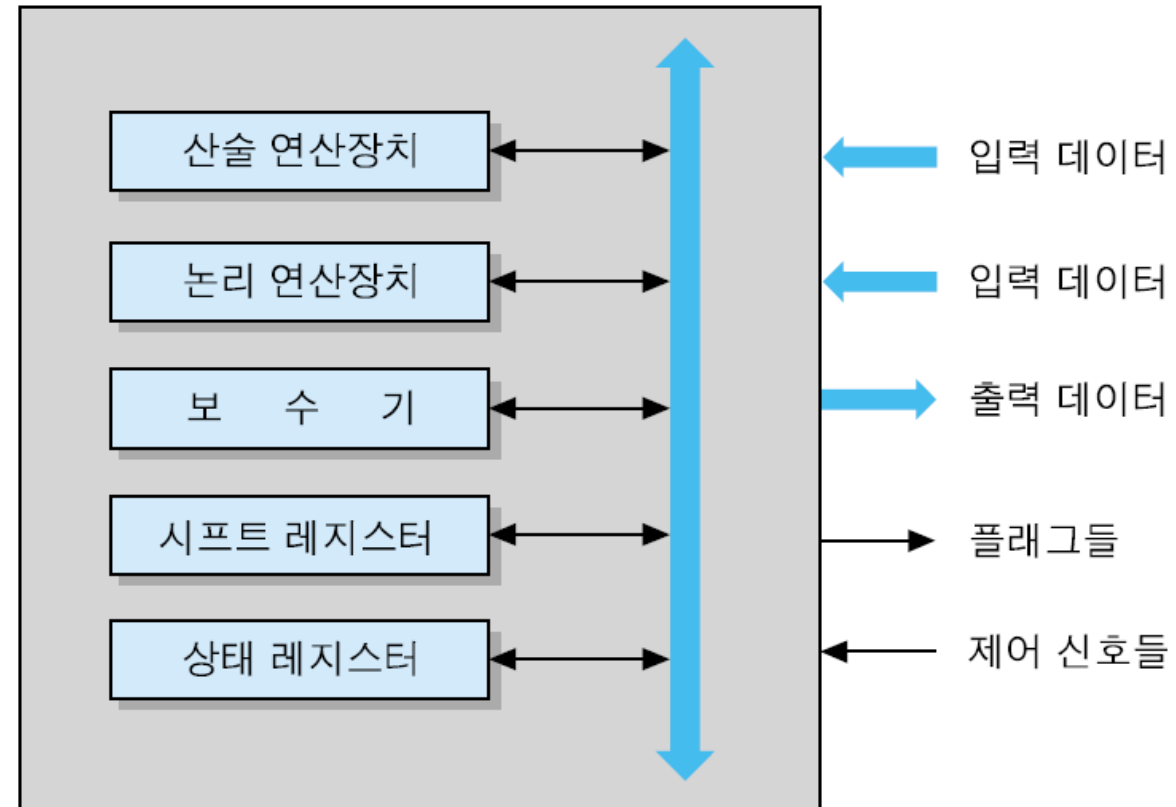
3.6 부동소수점 수의 표현

3.7 부동소수점 산술 연산

## 3.1 ALU의 구성 요소

- 산술 연산장치 : 산술 연산들(+, -, ×, ÷)을 수행
- 논리 연산장치 : 논리 연산들(AND, OR, XOR, NOT 등)을 수행
- 시프트 레지스터(shift register) : 비트들을 좌측 혹은 우측으로 이동시키는 기능을 가진 레지스터
- 보수기(complementer) : 2진 데이터를 2의 보수로 변환(음수화)
- 상태 레지스터(status register) : 연산 결과의 상태를 나타내는 플래그(flag)들을 저장하는 레지스터

## ALU의 내부 구성 요소들



## 3.2 정수의 표현

- 2진수 체계 : 0, 1, 부호 및 소수점으로 수를 표현

[예]  $-13.625_{10} = -1101.101_2$

- 부호 없는 정수 표현의 예

$$00111001 = 57$$

$$00000000 = 0$$

$$00000001 = 1$$

$$10000000 = 128$$

$$11111111 = 255$$

- $n$ -비트 2진수를 부호 없는 정수  $A$ 로 변환하는 방법 :

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

## 소수와 음수의 표현

□ 최상위 비트인  $a_{n-1}$ 의 좌측에 소수점이 있는 소수의 10진수 변환방법

$$\blacksquare A = a_{n-1} \times 2^{-1} + a_{n-2} \times 2^{-2} + \dots + a_1 \times 2^{-(n-1)} + a_0 \times 2^{-n}$$

[예]  $2^3 \quad 2^2 \quad 2^1 \quad 2^0 \quad 2^{-1} \quad 2^{-2} \quad 2^{-3}$  : 자리수(weight)

1    1    0    1    .    1    0    1

$$= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

$$= 8 + 4 + 1 + 0.5 + 0.125 = 13.625$$

□ 음수 표현 방법

- 부호화-크기 표현(signed-magnitude representation)
- 1의 보수 표현(1's complement representation)
- 2의 보수 표현(2's complement representation)

## 3.2.1 부호화-크기 표현

- 맨좌측 비트는 부호 비트, 나머지  $(n-1)$ 개의 비트들은 수의 크기 (magnitude)를 나타내는 표현 방식

$$\text{[예]} \quad +9 = 0\ 0001001 \qquad +35 = 0\ 0100011$$

$$\qquad -9 = 1\ 0001001 \qquad -35 = 1\ 0100011$$

- 부호화-크기로 표현된 2진수  $(a_{n-1} a_{n-2} \dots a_1 a_0)$ 를 10진수로 변환

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

$$\begin{aligned} \text{[예]} \quad 0\ 0100011 &= (-1)^0 (0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) \\ &= (32 + 2 + 1) = 35 \end{aligned}$$

$$\begin{aligned} 1\ 0001001 &= (-1)^1 (0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0) \\ &= -(8 + 1) = -9 \end{aligned}$$

## 부호화-크기 표현 (계속)

### □ 결점

- 덧셈과 뺄셈을 수행하기 위해서는 부호 비트와 크기를 비교하여 처리하는 복잡한 과정 필요

- 0에 대한 표현이 두 개 존재

$$0 \ 0000000 = +0$$

$$1 \ 0000000 = -0$$

➔  $n$ -비트 단어로 표현할 수 있는 수들이  $2^n$  개가 아닌,  
 $(2^n - 1)$ 개로 감소

## 3.2.2 보수 표현

### □ 1의 보수(1's complement) 표현

- 모든 비트들을 반전 ( $0 \rightarrow 1, 1 \rightarrow 0$ )

### □ 2의 보수(2's complement) 표현

- 모든 비트들을 반전하고, 결과값에 1을 더한다

[예]

$$+ 9 = 0\ 0001001$$

$$- 9 = 1\ 1110110 \text{ (1의 보수)}$$

$$- 9 = 1\ 1110111 \text{ (2의 보수)}$$

$$+ 35 = 0\ 0100011$$

$$- 35 = 1\ 1011100 \text{ (1의 보수)}$$

$$- 35 = 1\ 1011101 \text{ (2의 보수)}$$



□ 8-비트 2진수로 표현할 수 있는 10진수의 범위

- 1의 보수 :  $-(2^7 - 1) \sim +(2^7 - 1)$
- 2의 보수 :  $-2^7 \sim +(2^7 - 1)$

10진수	1의 보수	2의 보수
127	01111111	01111111
126	01111110	01111110
⋮	⋮	⋮
2	00000010	00000010
1	00000001	00000001
+0	00000000	00000000
-0	11111111	—
-1	11111110	11111111
-2	11111101	11111110
⋮	⋮	⋮
-126	10000001	10000010
-127	10000000	10000001
-128	—	10000000

## 2의 보수 → 10진수 변환

- 2의 보수로 표현된 양수( $a_{n-1} = 0$ )를 10진수로 변환하는 방법
  - $A = a_{n-2} \times 2^{n-2} + a_{n-3} \times 2^{n-3} + \dots a_1 \times 2^1 + a_0 \times 2^0$
- 2의 보수로 표현된 음수( $a_{n-1} = 1$ )를 10진수로 변환하는 방법
  - $A = -2^{n-1} + (a_{n-2} \times 2^{n-2} + a_{n-3} \times 2^{n-3} + \dots a_1 \times 2^1 + a_0 \times 2^0)$

$$\begin{aligned} \text{[예]} \quad 10101110 &= -128 + (1 \times 2^5 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1) \\ &= -128 + (32 + 8 + 4 + 2) = -82 \end{aligned}$$

[다른 방법] 10101110 → 01010010 으로 먼저 변환한 후, 음수 표시

$$\begin{aligned} 01010010 &= -(1 \times 2^6 + 1 \times 2^4 + 1 \times 2^1) \\ &= -(64 + 16 + 2) = -82 \end{aligned}$$

### 3.2.3 비트 확장 (Bit Extension)

#### □ 데이터의 길이(비트 수)를 늘리는 방법

- 필요성: 데이터를 더 많은 비트의 레지스터에 저장하거나, 더 긴 데이터와 연산을 수행하기 위해 필요

##### 예제 3-7

10진수 '21'과 '-21'에 대한 8-비트 길이의 부호화-크기 표현을 16-비트 길이로 확장하라.

##### 풀이

+21 =	00010101	(8-비트 부호화-크기 표현)
+21 =	0000000000010101	(16-비트 부호화-크기 표현)
-21 =	10010101	(8-비트 부호화-크기 표현)
-21 =	1000000000010101	(16-비트 부호화-크기 표현)

## 비트 확장 (계속)

- 2의 보수 표현의 경우 : 확장되는 상위 비트들을 부호 비트와 같은 값으로 세트
- 부호-비트 확장(sign-bit extension)이라 함

### 예제 3-8

10진수 '21'과 '-21'에 대한 8-비트 길이의 2의 보수 표현을 16-비트 길이로 확장하라.

#### 풀이

+21 = 00010101 (8-비트 2의 보수)

+21 = 0000000000010101 (16-비트 2의 보수)

-21 = 11101011 (8-비트 2의 보수)

-21 = 1111111111101011 (16-비트 2의 보수)

## 3.3 논리 연산

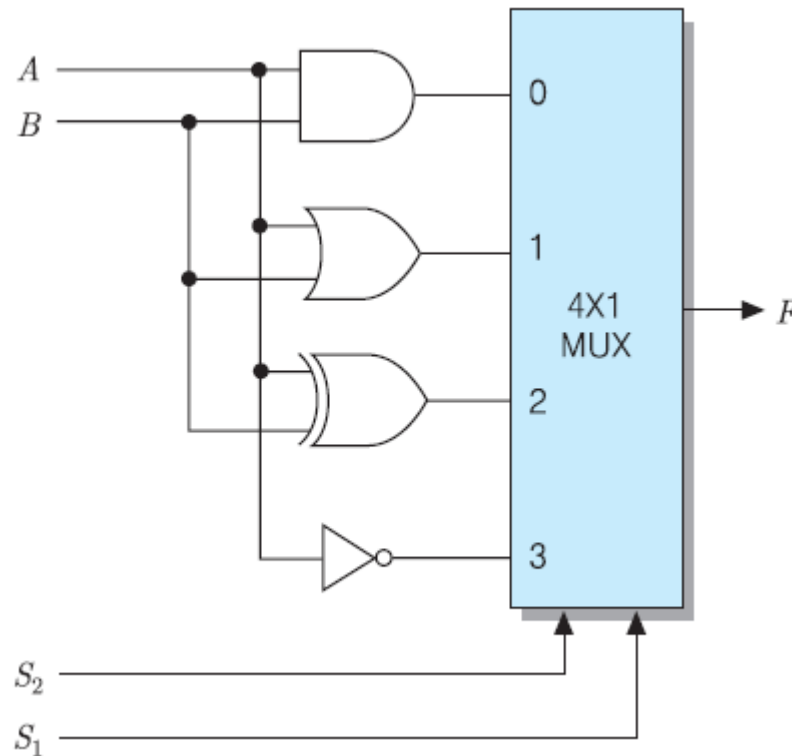
### □ 기본적인 논리 연산들

$A$	$B$	NOT $A$	NOT $B$	$A$ AND $B$	$A$ OR $B$	$A$ XOR $B$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	1	0

# 논리 연산을 위한 하드웨어 모듈

## □ 하드웨어의 구성

- 입력 비트들은 모든 논리 게이트들을 통과
- 선택 신호들에 의하여 멀티플렉서의 네 입력들 중의 하나를 출력



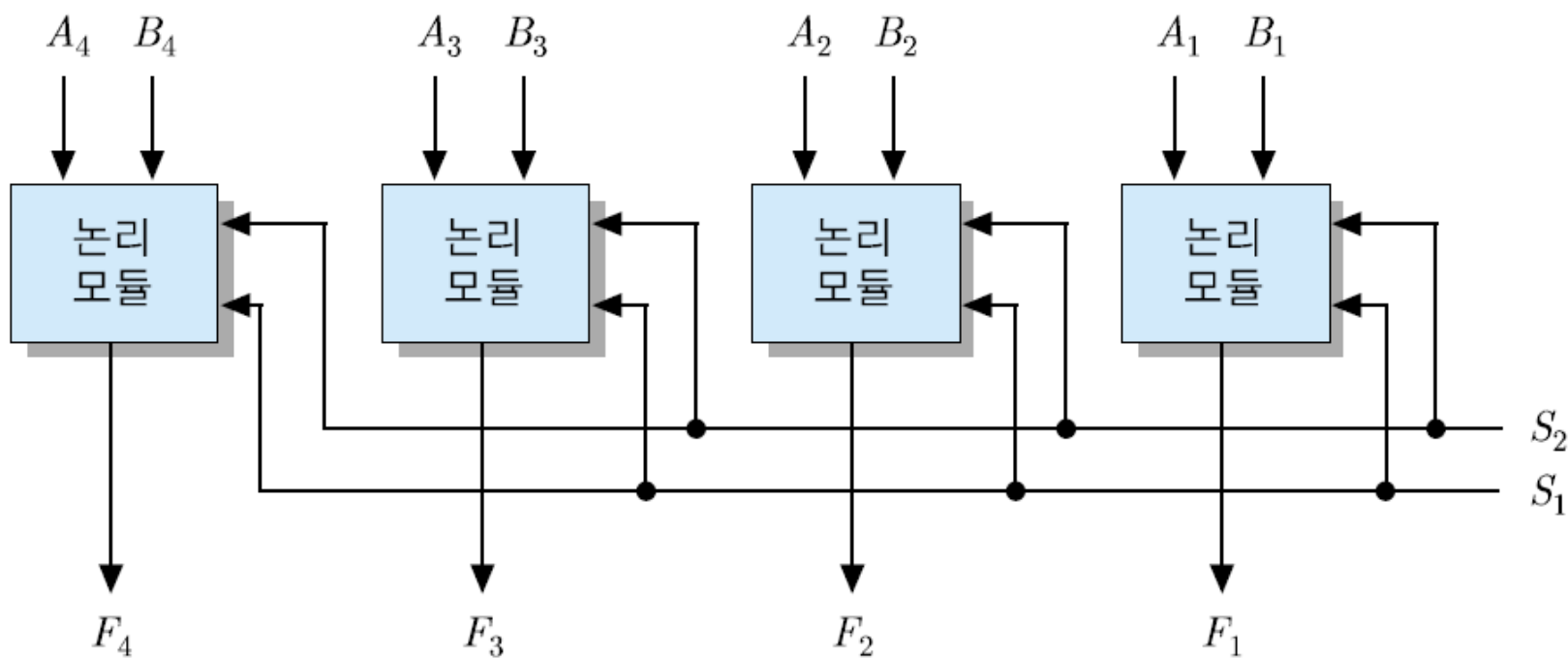
S <sub>2</sub>	S <sub>1</sub>	출력	연산
0	0	$F = A \wedge B$	AND
0	1	$F = A \vee B$	OR
1	0	$F = A \oplus B$	XOR
1	1	$F = \bar{A}$	NOT

## $N$ -비트 논리 연산장치

□  $N$ -비트 데이터들을 위한 논리 연산장치

- 기본 논리 모듈들을 병렬로 접속

[예] 4-비트 논리 연산장치



## AND 연산 / OR 연산

- **AND 연산** : 두 데이터 단어들의 대응되는 비트들 간에 AND 연산을 수행

A = 1 0 1 1 0 1 0 1

B = 0 0 1 1 1 0 1 1

-----

0 0 1 1 0 0 0 1 (연산 결과)

- **OR 연산** : 두 데이터 단어들의 대응되는 비트들 간에 OR 연산 수행

A = 1 0 0 1 0 1 0 1

B = 0 0 1 1 1 0 1 1

-----

1 0 1 1 1 1 1 1 (연산 결과)



## XOR 연산 / NOT 연산

- **XOR 연산** : 두 데이터 단어들의 대응되는 비트들 간에 exclusive-OR 연산을 수행

A = 1 0 0 1 0 1 0 1

B = 0 0 1 1 1 0 1 1

-----

1 0 1 0 1 1 1 0 (연산 결과)

- **NOT 연산** : 데이터 단어의 모든 비트들을 반전(invert)

[예]

A = 1 0 0 1 0 1 0 1 (연산 전)

-----

0 1 1 0 1 0 1 0 (연산 결과)

## 선택적-세트 연산 / 선택적-보수 연산

- 선택적-세트(selective-set) 연산 : B 레지스터의 비트들 중에서 1로 세트 된 비트들과 같은 위치에 있는 A 레지스터의 비트들을 1로 세트 <OR 연산 이용>

A = 1 0 0 1 0 0 1 0 (연산 전)

B = 0 0 0 0 1 1 1 1

-----

A = 1 0 0 1 1 1 1 1 (연산 결과)

- 선택적-보수(selective-complement) 연산 : B 레지스터의 비트들 중에서 1로 세트 된 비트들에 대응되는 A 레지스터의 비트들을 보수로 변환 <XOR 연산 이용>

A = 1 0 0 1 0 1 0 1 (연산 전)

B = 0 0 0 0 1 1 1 1

-----

A = 1 0 0 1 1 0 1 0 (연산 결과)

## 마스크 연산

- **마스크(mask) 연산** : B 레지스터의 비트들 중에서 값이 0인 비트들과 같은 위치에 있는 A 레지스터의 비트들을 0으로 바꾸는(clear하는) 연산 <AND 연산 이용>
- 용도 : 단어 내의 원하는 비트들을 선택적으로 clear하는 데 사용

[예] A = 1 1 0 1 0 1 0 1 (연산 전)  
      B = 0 0 0 0 1 1 1 1  
-----  
      A = 0 0 0 0 0 1 0 1 (연산 결과)

## 삽입 연산

□ **삽입(insert) 연산** : 새로운 비트 값들을 데이터 단어내의 특정 위치에 삽입

- 방법 : ① 삽입할 비트 위치들에 대하여 **마스크(AND) 연산** 수행  
② 새로이 삽입할 비트들과 **OR 연산**을 수행

[예]

A = 1 0 0 1 0 1 0 1

B = 0 0 0 0 1 1 1 1    마스크 (AND 연산)

-----

A = 0 0 0 0 0 1 0 1    첫 단계 결과

B = 1 1 1 0 0 0 0 0    삽입 (OR 연산)

-----

A = 1 1 1 0 0 1 0 1    최종(삽입) 결과

## 비교 연산

### □ 비교(compare) 연산

- A와 B 레지스터의 내용을 비교 → XOR 연산
  - 만약 대응되는 비트들의 값이 같으면, A 레지스터의 해당 비트를 '0'으로 세트
  - 만약 서로 다르면, A 레지스터의 해당 비트를 '1'로 세트
  - 모든 비트들이 같으면(A = 00000000), Z 플래그를 1로 세트

[예]

A = 1 1 0 1 0 1 0 1

B = 1 0 0 1 0 1 1 0

-----

A = 0 1 0 0 0 0 1 1 (연산 결과)

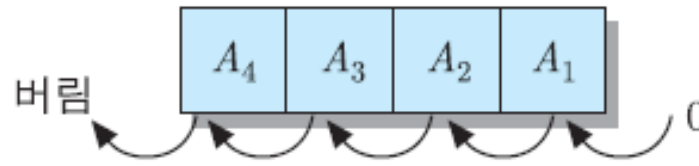
### 3.4 시프트(shift) 연산

□ 논리적 시프트 (logical shift) : 레지스터 내의 데이터 비트들을 왼쪽 혹은 오른쪽으로 한 칸씩 이동

■ 좌측 시프트(left shift)

- 모든 비트들을 좌측으로 한 칸씩 이동
- 최하위 비트( $A_0$ )로는 '0'이 들어오고, 최상위 비트( $A_4$ )는 버림

$$A_4 \leftarrow A_3, A_3 \leftarrow A_2, A_2 \leftarrow A_1, A_1 \leftarrow 0$$

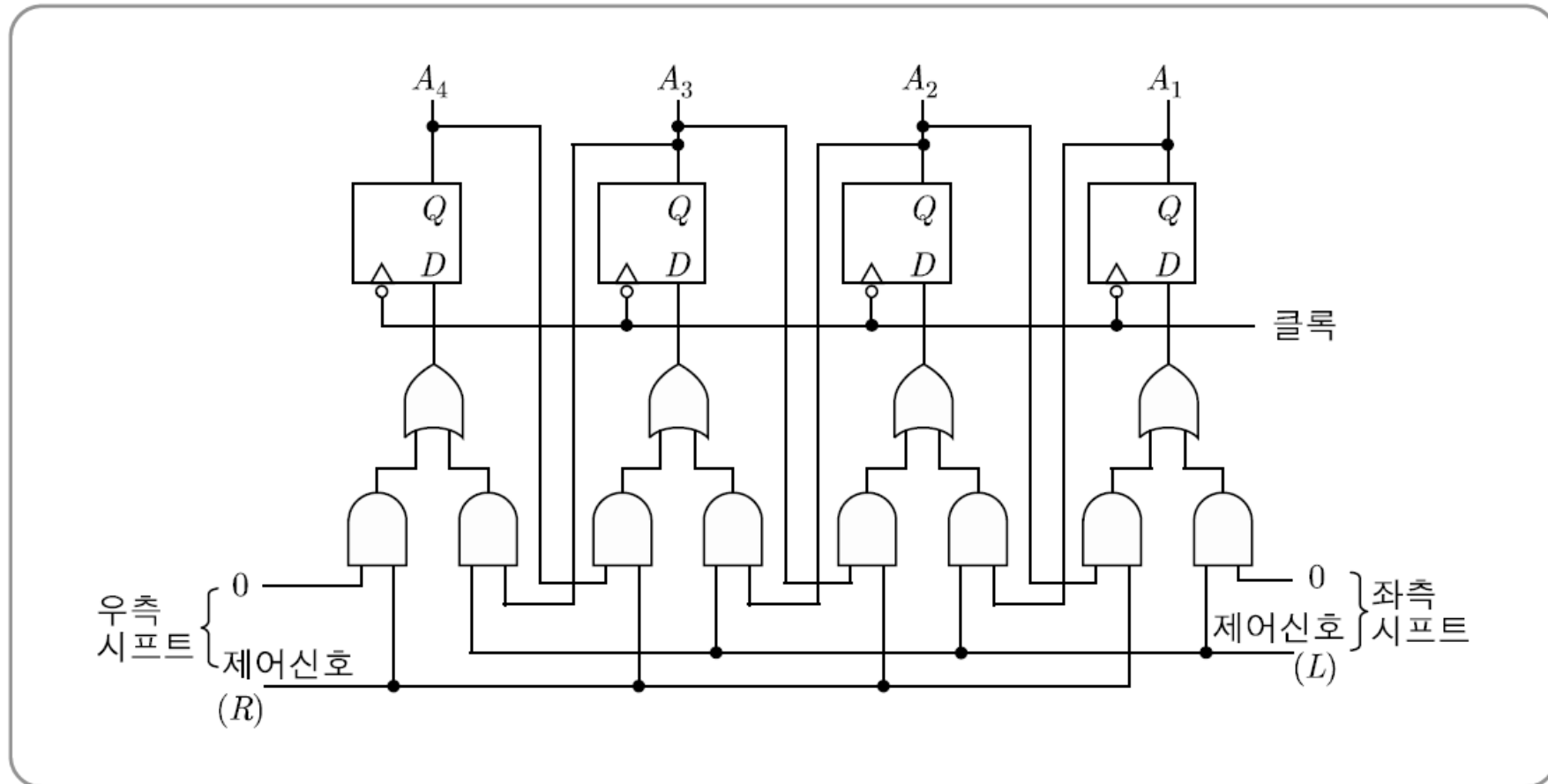


■ 우측 시프트(right shift)

- 모든 비트들이 우측으로 한 칸씩 이동
- 최상위 비트( $A_4$ )로 '0'이 들어오고, 최하위 비트( $A_0$ )는 버림

## 시프트 레지스터(shift register)

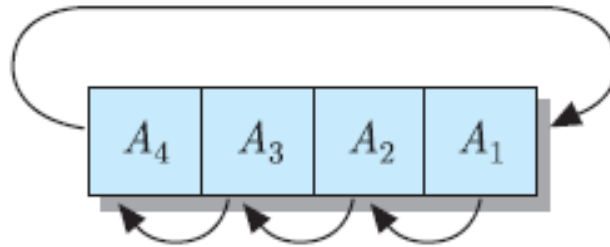
- 시프트 기능을 가진 레지스터의 내부 회로



□ 순환 시프트(circular shift) : 회전(rotate)이라고도 부르며, 최상위 혹은 최하위에 있는 비트를 버리지 않고 반대편 끝에 있는 비트 위치로 이동

- 순환 좌측-시프트(circular shift-left) : 최상위 비트인  $A_4$ 가 최하위 비트 위치인  $A_1$ 으로 이동

$$(A_4 \leftarrow A_3, A_3 \leftarrow A_2, A_2 \leftarrow A_1, A_1 \leftarrow A_4)$$

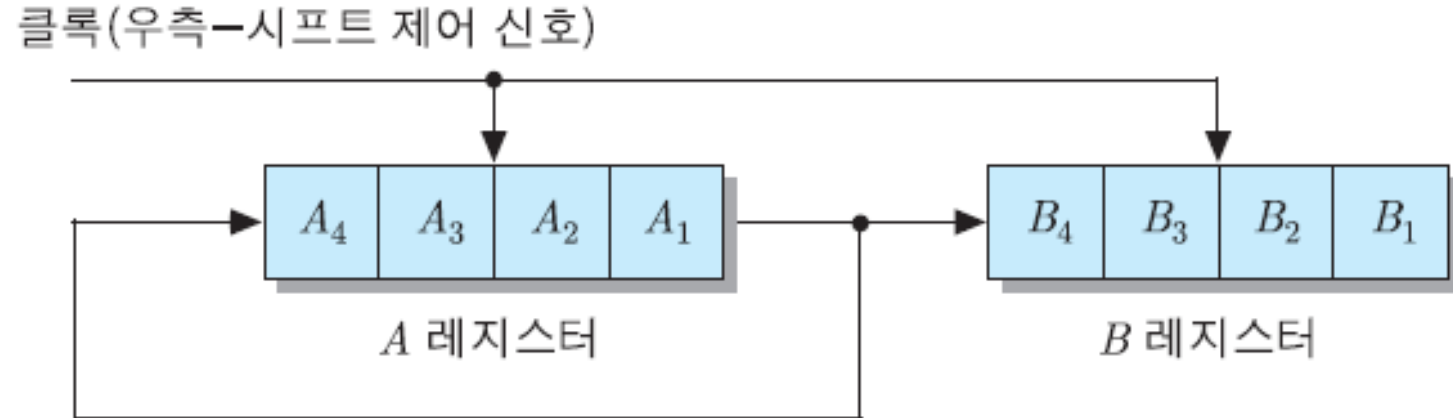


- 순환 우측-시프트(circular shift-right)
  - $A_4 \rightarrow A_3, A_3 \rightarrow A_2, A_2 \rightarrow A_1, A_1 \rightarrow A_4$



## 직렬 데이터 전송 (serial data transfer)

- 시프트 연산을 데이터 비트 수만큼 연속적으로 수행함으로써 두 레지스터들 사이에 한 개의 선을 통하여 전체 데이터를 이동하는 동작



### 예제 3-18

그림 3-7과 같이 두 개의 4-비트 레지스터들이 직렬로 접속된 상태에서 A 레지스터에 저장되어 있는 데이터인 '1011'을 네 클럭 주기 후에 모두 B 레지스터로 전송하고, A 레지스터는 원래의 데이터를 그대로 유지하게 되는 과정을 표시하라.

풀이

	$A_4$	$A_3$	$A_2$	$A_1$	$B_4$	$B_3$	$B_2$	$B_1$
초기상태	1	0	1	1	0	0	0	0
$t_1$	1	1	0	1	1	0	0	0
$t_2$	1	1	1	0	1	1	0	0
$t_3$	0	1	1	1	0	1	1	0
$t_4$	1	0	1	1	1	0	1	1

□ 산술적 시프트(arithmetic shift) : 수(number)를 나타내는 데이터에 대한 시프트

- 방법: 시프트 과정에서 부호 비트는 그대로 유지시키고, 수의 크기를 나타내는 비트들만 시프트

(1) 산술적 좌측-시프트(arithmetic shift-left)

$$A4 \text{ (불변)}, A3 \leftarrow A2, A2 \leftarrow A1, A1 \leftarrow 0$$

(2) 산술적 우측-시프트(arithmetic shift-right)

$$A4 \text{ (불변)}, A4 \rightarrow A3, A3 \rightarrow A2, A2 \rightarrow A1$$

### 예제 3-19

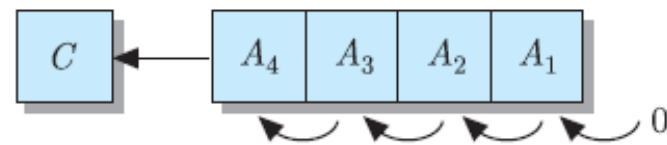
10진수 -2에 대한 4-비트 2진수에 대하여 산술적 좌측-시프트를 수행한 후, 그 결과 값에 대하여 두 번의 산술적 우측-시프트를 수행하라.

풀이

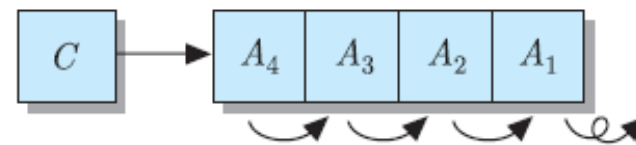
$A = 1\ 1\ 1\ 0$	(-2)	(초기 상태)
$1\ 1\ 0\ 0$	(-4)	(산술적 좌측-시프트 결과)
$1\ 1\ 1\ 0$	(-2)	(첫 번째 산술적 우측-시프트 결과)
$1\ 1\ 1\ 1$	(-1)	(두 번째 산술적 우측-시프트 결과)

## □ C 플래그를 포함한 시프트 연산

- C 플래그를 포함한 좌측-시프트(**SHLC** : shift left with carry)
- C 플래그를 포함한 우측-시프트(**SHRC** : shift right with carry)

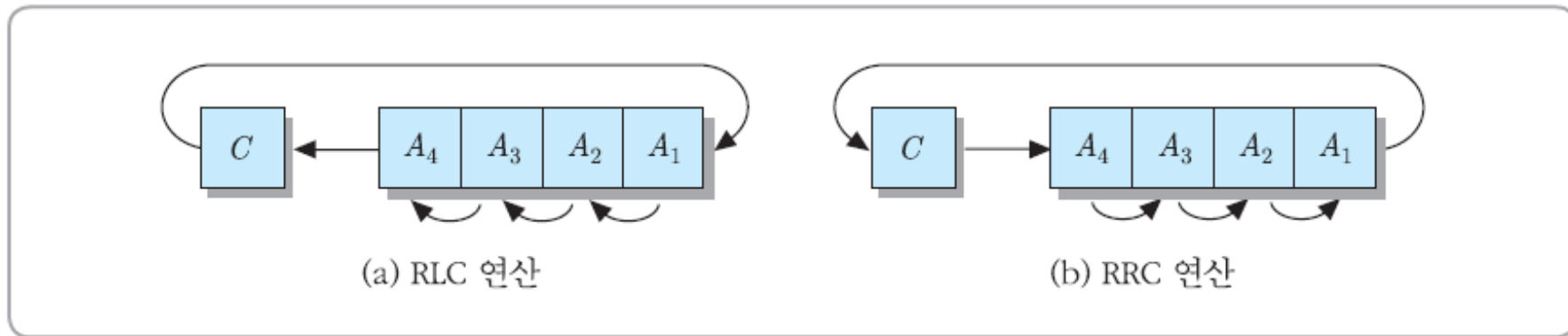


(a) SHLC 연산



(b) SHRC 연산

- ❑ RLC(rotate left with carry):  $C$  플래그를 포함하는 좌측 순환 시프트(회전) 연산
- ❑ RRC(rotate right with carry):  $C$  플래그를 포함하는 우측 순환 시프트(회전) 연산



## 3.5 정수의 산술 연산

### □ 기본적인 산술 연산들

---

$A \leftarrow \overline{A} + 1$  ; 보수화(2의 보수 변환)

$A \leftarrow A + B$  ; 덧셈

$A \leftarrow A - B$  ; 뺄셈

$A \leftarrow A \times B$  ; 곱셈

$A \leftarrow A \div B$  ; 나눗셈

$A \leftarrow A + 1$  ; 증가(increment)

$A \leftarrow A - 1$  ; 감소(decrement)

---

## 3.5.1 덧셈

### □ 2의 보수로 표현된 수들의 덧셈 방법

- 두 수를 더하고, 만약 올림수가 발생하면 버림

#### 예제 3-20

아래와 같은 2진수 덧셈들을 수행하고, 10진수 덧셈의 결과와 비교하라.

풀이

(a)  $(+3) + (+4) = +7$

$$\begin{array}{r} 0011 \\ + 0100 \\ \hline 0111 = +7 \end{array}$$

(c)  $(-6) + (+2) = -4$

$$\begin{array}{r} 1010 \\ + 0010 \\ \hline 1100 = -4 \end{array}$$

(b)  $(-3) + (+3) = 0$

$$\begin{array}{r} 1101 \\ + 0011 \\ \hline \text{버림 } \textcircled{1} 0000 = 0 \end{array}$$

(d)  $(-4) + (-1) = -5$

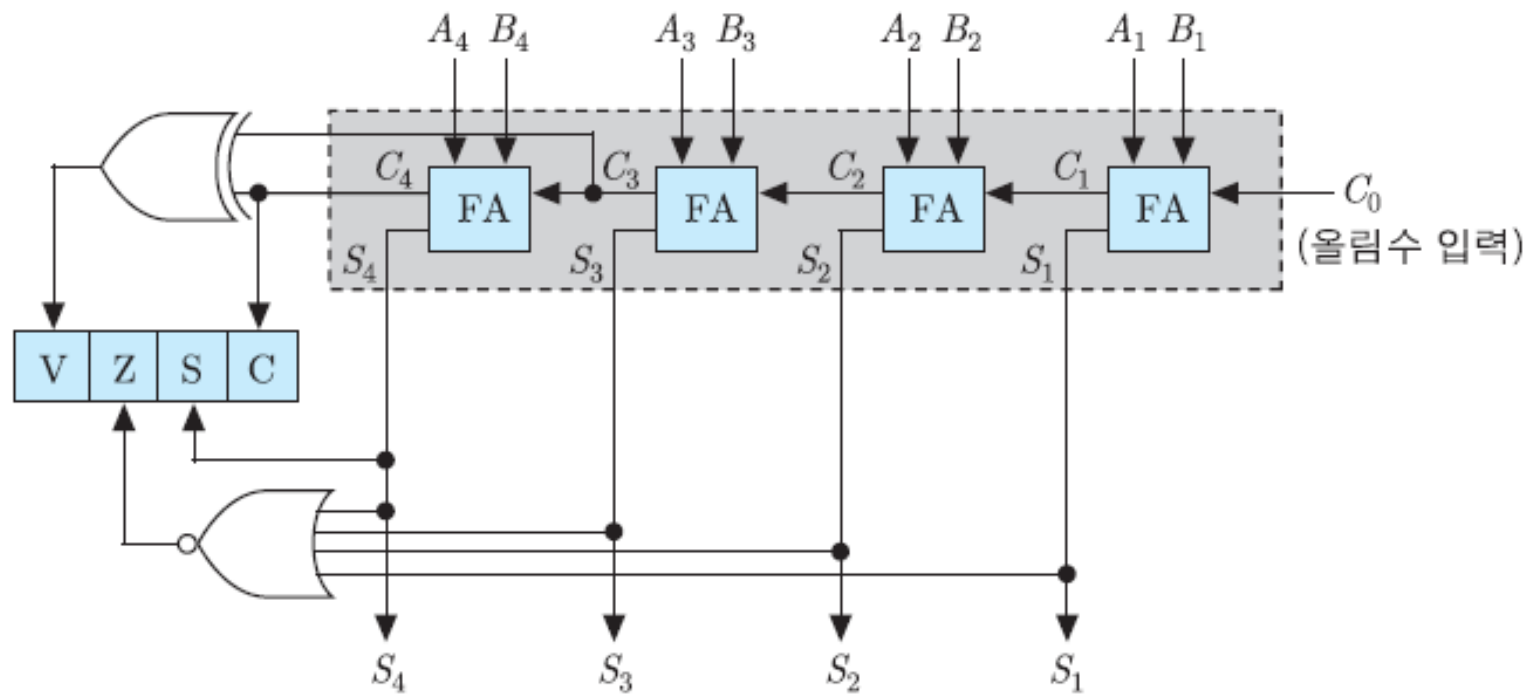
$$\begin{array}{r} 11111100 \\ + 11111111 \\ \hline \text{버림 } \textcircled{1} 11111011 = -5 \end{array}$$



## 병렬 가산기(parallel adder)

- 덧셈을 수행하는 하드웨어 모듈
- 비트 수만큼의 전가산기(full-adder)들로 구성
- 덧셈 연산 결과에 따라 해당 조건 플래그들(condition flags)을 세트
  - *C* 플래그 : 올림수(carry)
  - *S* 플래그 : 부호(sign)
  - *Z* 플래그 : 0(zero)
  - *V* 플래그 : 오버플로우(overflow)

## 4-비트 병렬 가산기와 상태 비트 제어회로



## 덧셈 오버플로우

- ❑ 덧셈 결과가 그 범위를 초과하여 결과값이 틀리게 되는 상태
- ❑ 검출 방법 : 두 올림수(carry)들 간의 exclusive-OR를 이용

$$V = C_4 \oplus C_3$$

### 예제 3-21

아래의 덧셈에서 오버플로우가 발생하는지 확인하라.

풀이

$$(a) (+6) + (+3) = +9$$

$$\begin{array}{r} 0110 \\ + 0011 \\ \hline 1001 = -7 \text{ (오버플로우)} \end{array}$$

$$(b) (-7) + (-6) = -13$$

$$\begin{array}{r} 1001 \\ + 1010 \\ \hline \textcircled{1} 0011 = +3 \text{ (오버플로우)} \end{array}$$

## 3.5.2 뺄셈

□ 뺄셈을 이용하여 수행

(A : 피감수(minuend), B : 감수(subtrahend))

$$A - (+B) = A + (-B)$$

$$A - (-B) = A + (+B)$$

### 예제 3-22

2의 보수를 이용하여 아래와 같은 2진수 뺄셈들을 수행하라.

풀이

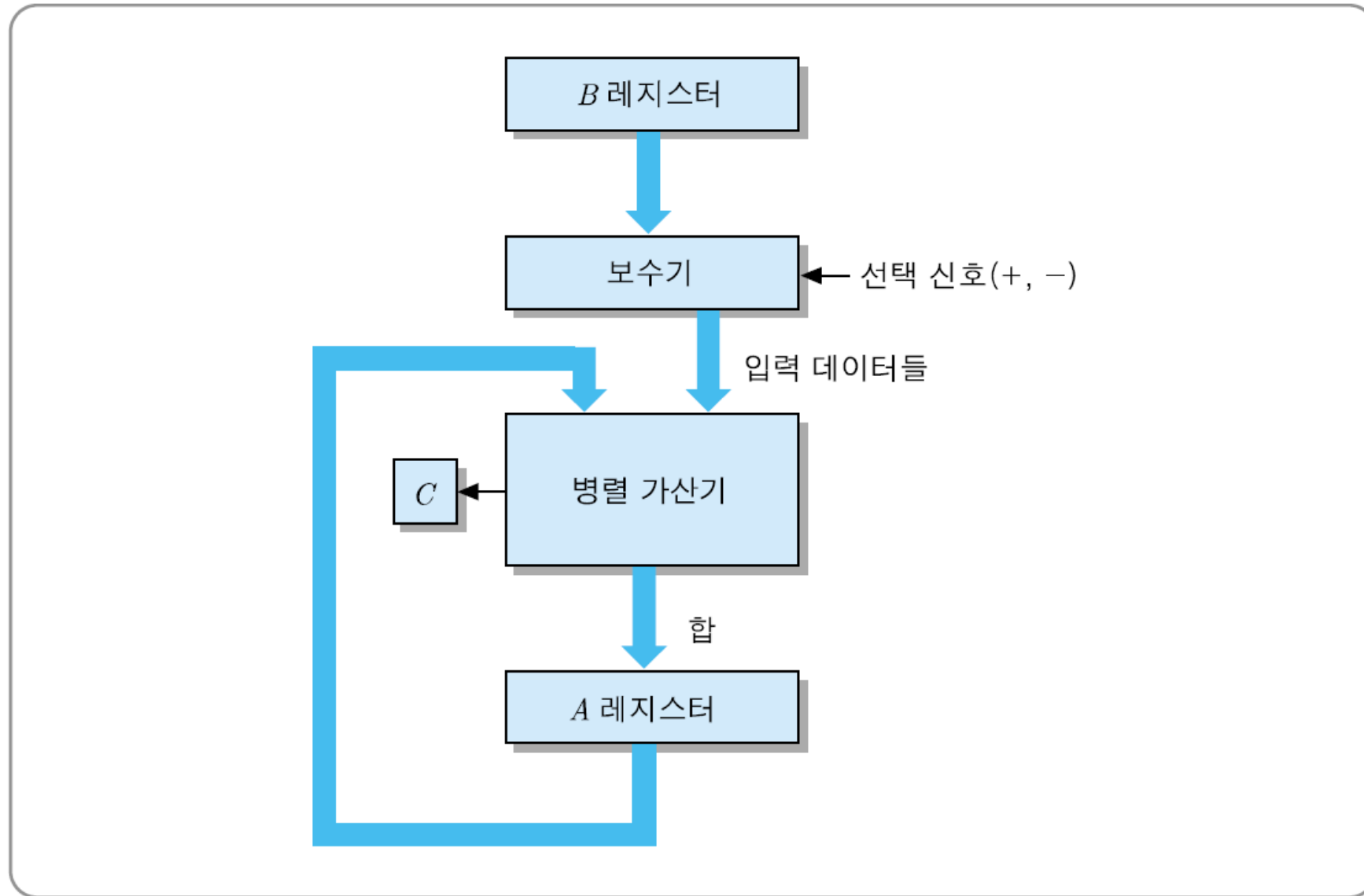
$$(a) (+2) - (+6) = (+2) + (-6) = -4$$

$$\begin{array}{r} 0010 \\ + 1010 \\ \hline 1100 = -4 \end{array}$$

$$(b) (+7) - (+4) = (+7) + (-4) = +3$$

$$\begin{array}{r} 00000111 \\ + 11111100 \\ \hline ①00000011 = +3 \end{array}$$

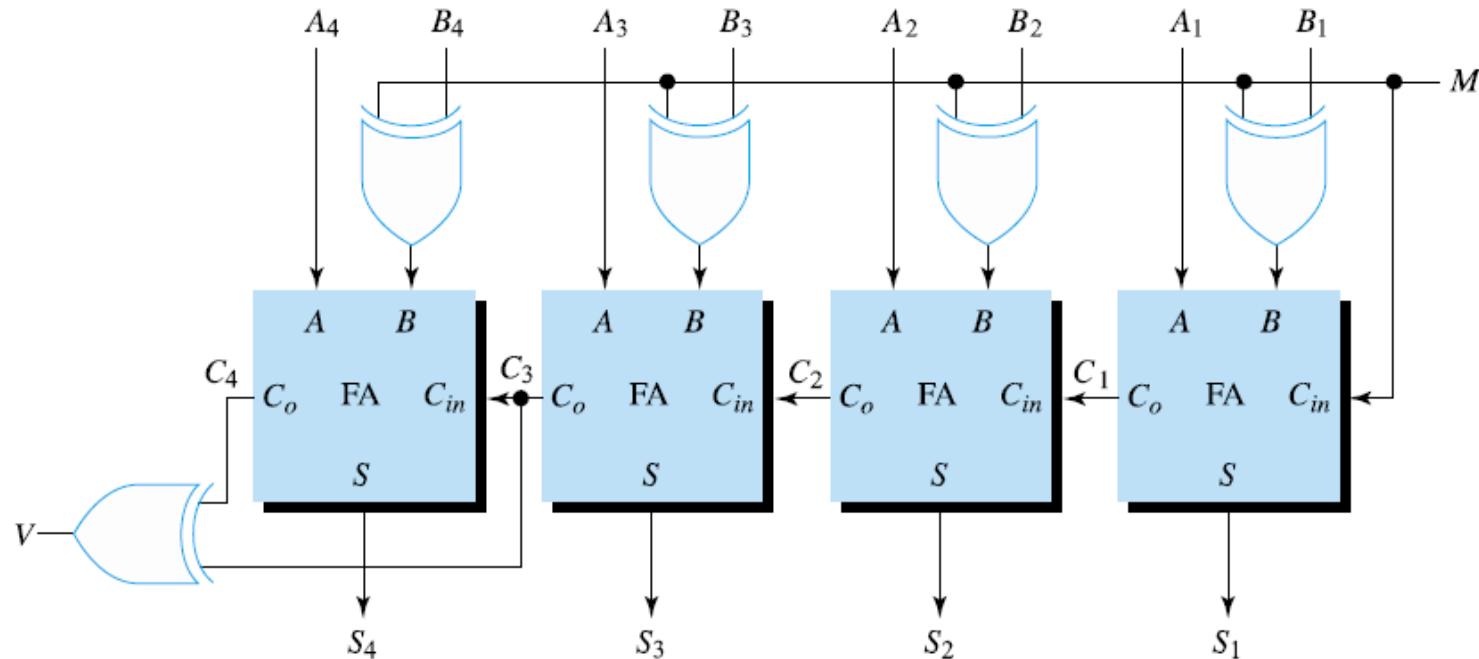
## 덧셈과 뺄셈 겸용 하드웨어의 블록 구성도



## 4-비트 병렬 가감산기 (4-bit parallel adder-subtractor)

<논리회로설계 제6장 참조>

- 4-비트 데이터들 간의 덧셈( $A+B$ ) 및 뺄셈( $A-B$ )을 모두 수행하는 조합회로
- 제어신호  $M = 0$  : 덧셈,  $M = 1$  : 뺄셈(입력  $B$ 의 비트들을 반전하고, 최하위 올림수로서  $M^\oplus$ 을 입력)



## 뺄셈 오버플로우

- ❑ 뺄셈 결과가 그 범위를 초과하여 결과값이 틀리게 되는 상태
- ❑ 검출 방법 : 덧셈과 동일 ( $V = C_4 \oplus C_3$ )

### 예제 3-23

2의 보수를 이용하여 수행하는 아래의 뺄셈들에서 오버플로우가 발생하는지 검사하라.

풀이

$$(a) (+6) - (-4) = (+6) + (+4) = +10$$

$$\begin{array}{r} 0110 \\ + 0100 \\ \hline \end{array}$$

$$1010 = -6 \quad (C_4 \oplus C_3 = 0 \oplus 1 = 1 : \text{오버플로우 발생})$$

$$(b) (-7) - (+6) = (-7) + (-6) = -13$$

$$\begin{array}{r} 1001 \\ + 1010 \\ \hline \end{array}$$

$$\textcircled{1} 0011 = +3 \quad (C_4 \oplus C_3 = 1 \oplus 0 = 1 : \text{오버플로우 발생})$$

### 3.5.3 부호 없는 정수의 곱셈

#### □ 방법

- 각 비트에 대하여 **부분 적(partial product)** 계산
- 부분 적들을 모두 더하여 최종 결과를 얻음

#### 예제 3-24

2진수 곱셈( $1101 \times 1011$ )을 수행하고, 결과값이 10진수 곱셈의 결과( $13 \times 11 = 143$ )와 일치하는지 확인하라.

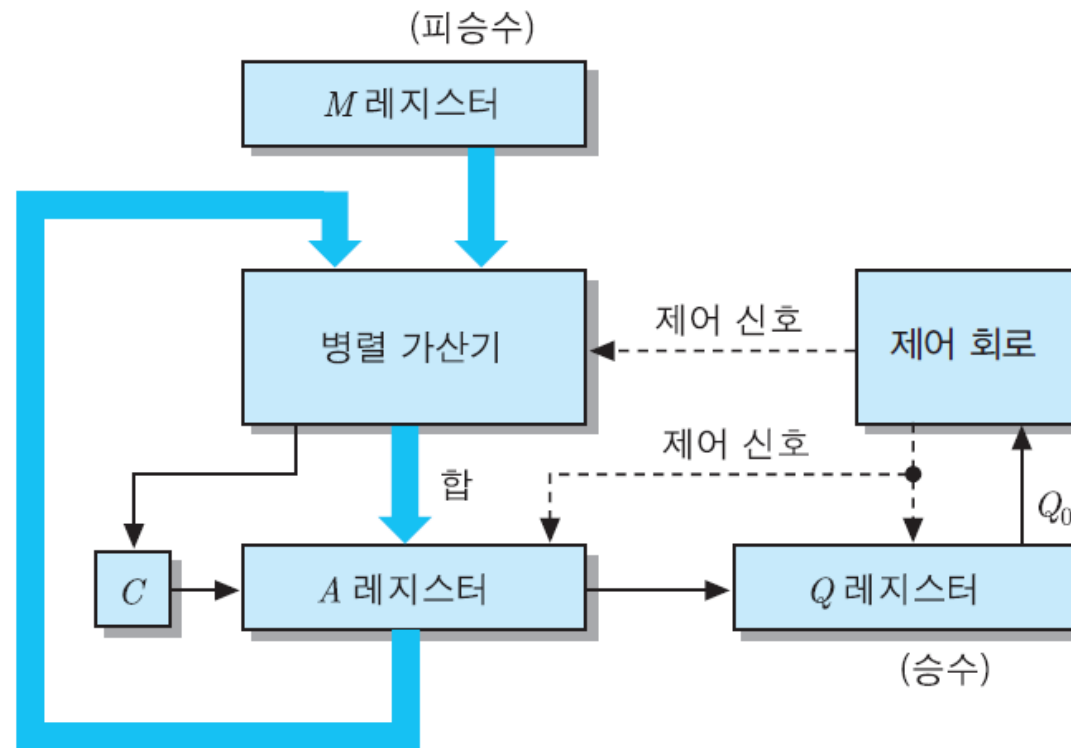
풀이

1101	(피승수 = 13)
$\times 1011$	(승수 = 11)
<hr/>	
1101	} 부분 적들(partial products)
1101	
0000	
1101	
<hr/>	
10001111	(최종 결과 = 143)



## 부호 없는 정수 승산기의 하드웨어 구성도

- $M$  레지스터 : 피승수(multiplicand) 저장
- $Q$  레지스터 : 승수(multiplier) 저장
- 두 배 길이의 결과값은  $A$  레지스터와  $Q$  레지스터에 저장



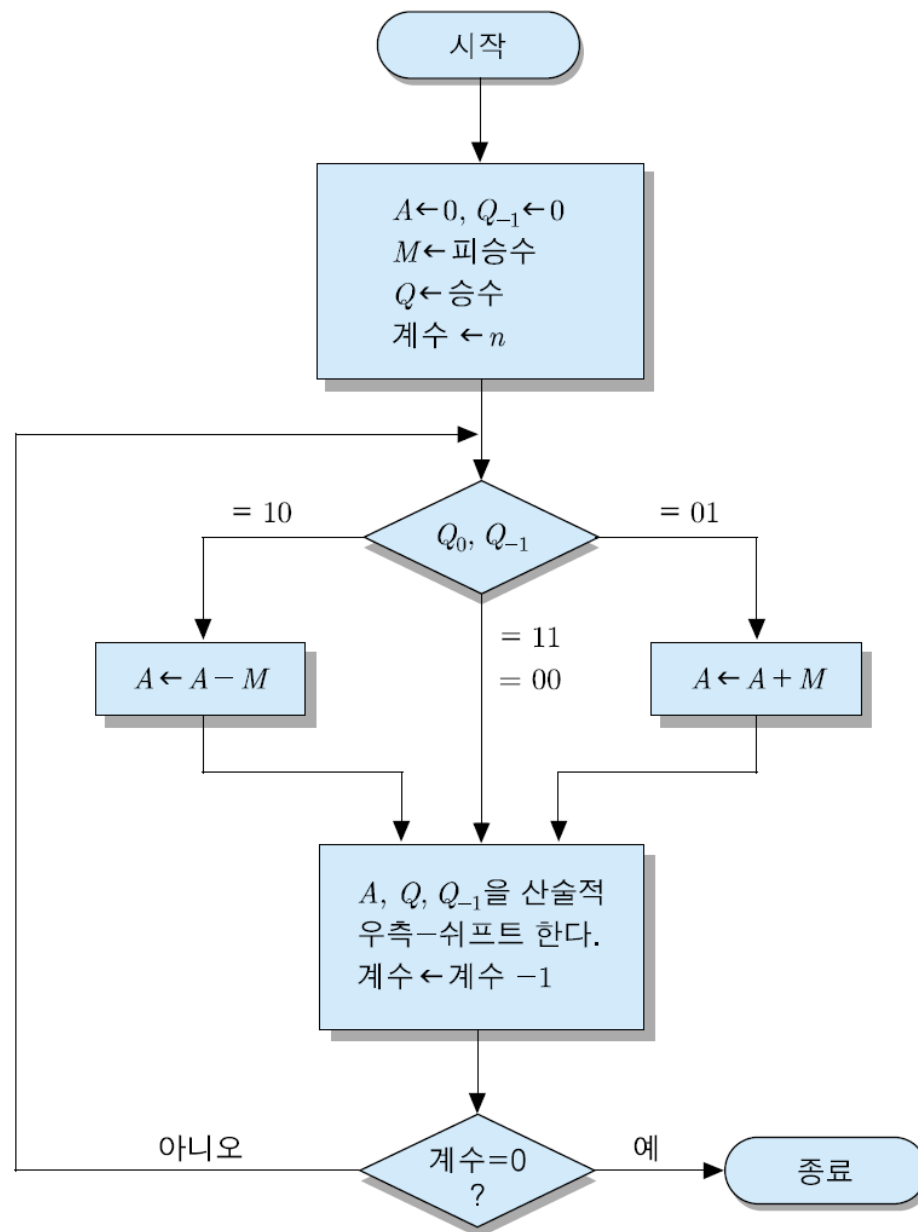
## 곱셈이 수행되는 과정에서의 레지스터 내용들

	<u>C</u>	<u>A</u>	<u>Q</u>	
[초기 상태]	0	0000	1011	
[사이클 1]	0	1101	1011	; $Q_0=1$ 이므로, $A \leftarrow A+M$
	0	0110	1101	; 우측 시프트( $C-A-Q$ )
[사이클 2]	1	0011	1101	; $Q_0=1$ 이므로, $A \leftarrow A+M$
	0	1001	1110	; 우측 시프트( $C-A-Q$ )
[사이클 3]	0	0100	1111	; $Q_0=0$ 이므로, 우측 시프트( $C-A-Q$ )만 수행
[사이클 4]	1	0001	1111	; $Q_0=1$ 이므로, $A \leftarrow A+M$
	0	1000	1111	; 우측 시프트( $C-A-Q$ )

## 2의 보수들 간의 곱셈

- Booth 알고리즘(Booth's algorithm) 사용
- 하드웨어 구성
  - 부호 없는 정수 승산기의 하드웨어에 다음 부분을 추가
    - $M$  레지스터와 병렬 가산기 사이에 보수기(complementer) 추가
    - $Q$  레지스터의 우측에  $Q_{-1}$  이라고 부르는 1-비트 레지스터를 추가하고, 출력을  $Q_0$ 와 함께 제어 회로로 입력

## Booth 알고리즘의 흐름도



## Booth 알고리즘을 이용한 곱셈의 예 (-7x3)

(M)	1001				초기값 : A = 0000, $Q_{-1} = 0$ , 계수 = 4
(Q)	× 0011	$Q_{-1}$	계수		
(AQ)	0111 0011	0			; ( $Q_0 Q_{-1}$ ) = (1 0)이므로, A로부터 피승수(1001)을 뺀다 (실제로는 그 보수인 0111을 더한다).
	0011 1001	1	3		; $AQQ_{-1}$ 을 산술적 우측-쉬프트 하고, 계수에서 1을 뺀다.
	0001 1100	1	2		; ( $Q_0 Q_{-1}$ ) = (1 1)이므로, $AQQ_{-1}$ 에 대한 산술적 우측-쉬프트만 하고, 계수에서 1을 뺀다.
	1010 1100	1			; ( $Q_0 Q_{-1}$ ) = (0 1)이므로, A에 피승수(1001)를 더하고, 계수에서 1을 뺀다.
	1101 0110	0	1		; $AQQ_{-1}$ 을 산술적 우측-쉬프트 하고, 계수에서 1을 뺀다.
	1110 1011	0	0		; ( $Q_0 Q_{-1}$ ) = (0 0)이므로, $AQQ_{-1}$ 을 산술적 우측-쉬프트한다. 계수에서 1을 빼면 0이므로 계산이 종료되었다.
					→ -21 (곱셈 결과)

### 3.5.4 나눗셈

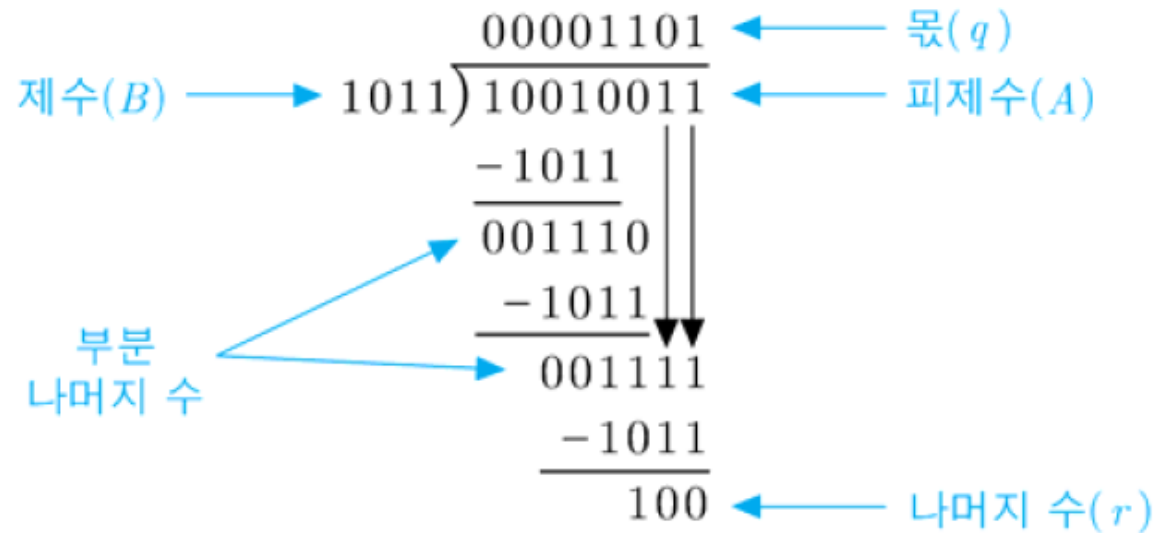
#### □ 나눗셈의 수식 표현

$$A \div B = q \cdots r$$

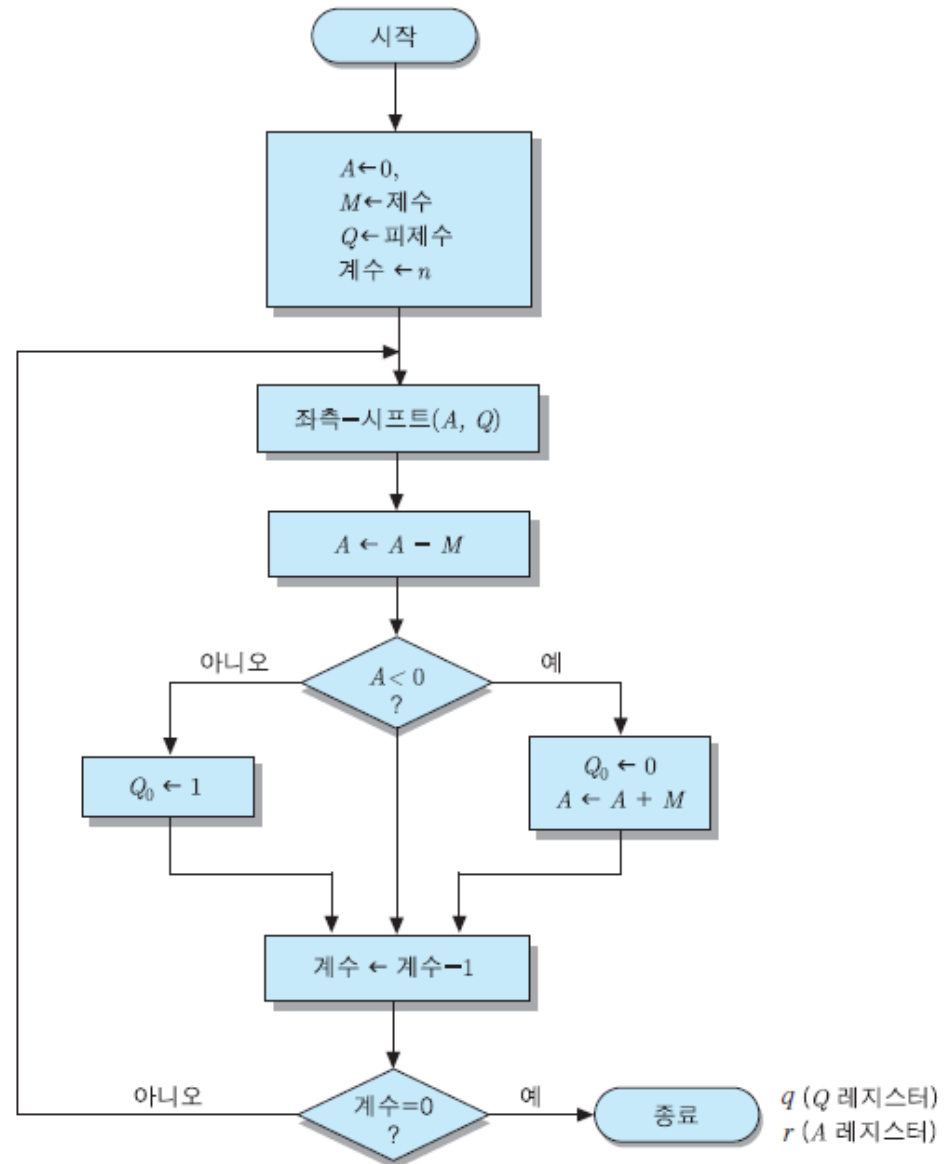
단,  $A$ : 피제수(dividend),  $B$ : 제수(divisor)

$q$ : 몫(quotient)  $r$ : 나머지 수(remainder)

#### □ 부호 없는 2진 나눗셈



## 부호 없는 2진 나눗셈 알고리즘의 흐름도



## 2의 보수 나눗셈 방법

- [초기 상태] 제수는  $M$  레지스터에, 피제수는  $A$ 와  $Q$  레지스터에 저장한다. 각 레지스터가  $n$ 비트일 때, 피제수는  $2n$ 비트 길이의 2의 보수로 표현한다.
- [사이클 1]  $A$ 와  $Q$  레지스터를 좌측으로 한 비트씩 시프트 한다.
- [사이클 2] 만약  $M$ 과  $A$ 의 부호가 같다면  $A \leftarrow A - M$ 을 수행하고, 다르다면  $A \leftarrow A + M$ 을 수행한다.
- [사이클 3] 연산 전과 후에서  $A$ 의 부호가 같다면, 위의 연산은 성공이다.  
(a) 연산이 성공이거나  $A = 0$ 이면,  $Q_0 \leftarrow 1$ 로 세트한다.  
(b) 산이 실패이고  $A \neq 0$ 이면,  $Q_0 \leftarrow 0$ 으로 세트하고,  $A$ 를 이전의 값으로 복구한다.
- [사이클 4]  $Q$  레지스터에 비트가 남아있다면, 단계 2에서 4까지를 반복한다.
- [사이클 5] 나머지 수는  $A$ 에 남는다. 만약 제수와 피제수의 부호가 같다면 몫은  $Q$  레지스터의 값이 되고, 그렇지 않다면  $Q$  레지스터의 내용에 대한 2의 보수가 몫이 된다.



## 3.6 부동소수점 수의 표현

- 부동소수점 표현(floating-point representation) : 소수점의 위치를 이동시킬 수 있는 수 표현 방법

➔ 수 표현 범위 확대

- 부동소수점 수(floating-point number)의 일반적인 형태

$$N = (-1)^S M \times B^E$$

단,  $S$ : 부호(sign),  $M$ : 가수(mantissa),  $B$ : 기수(base),  
 $E$ : 지수(exponent)

## 부동소수점 표현 (계속)

### □ 10진 부동소수점 수(decimal floating-point number)

[예]  $274,000,000,000,000 \rightarrow 2.74 \times 10^{14}$

$0.000000000000274 \rightarrow 2.74 \times 10^{-12}$

### □ 2진 부동소수점 수(binary floating-point number)

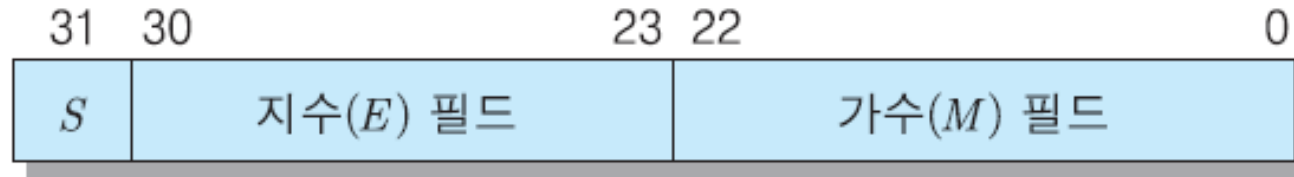
- 기수  $B = 2$

[예]  $11.101 \rightarrow 0.11101 \times 2^2$ ,  $0.00001101 \rightarrow 0.1101 \times 2^{-4}$

- 단일-정밀도(single-precision) 부동소수점 수 : 32 비트
- 복수-정밀도(double-precision) 부동소수점 수 : 64 비트

## 단일-정밀도 부동소수점 수 형식의 예

□  $S$ : 1 비트,     $E$ : 8 비트,     $M$ : 23 비트



□ 지수( $E$ ) 필드의 비트 수 증가 → 표현 가능한 수의 범위 확장

□ 가수( $M$ ) 필드의 비트 수 증가 → 정밀도(precision) 증가

□ 표현가능한 수 크기의 범위 :  $(0.5 \times 2^{-128} \sim 0.5 \times 2^{127})$   
 $\approx 1.47 \times 10^{-39} \sim 1.7 \times 10^{38}$

[비교] 32-비트 고정소수점(fixed-point) 표현 방식의 경우 :

$$1.0 \times 2^{-31} \sim 1.0 \times 2^{31} \approx 2.0 \times 10^{-9} \sim 2.0 \times 10^9$$

## 같은 수에 대한 부동소수점 표현

- 같은 수에 대한 부동소수점 표현이 여러 가지가 존재

$$0.1101 \times 2^5$$

$$11.01 \times 2^3$$

$$0.001101 \times 2^7$$

- 정규화된 표현(Normalized representation)

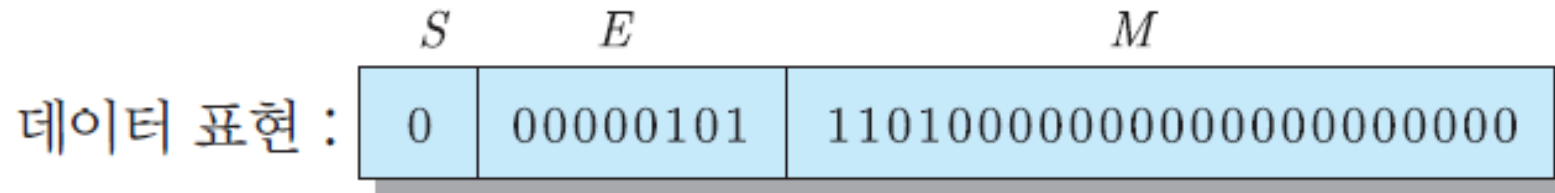
- 수에 대한 표현을 한 가지로 통일하기 위한 방법

$$\pm 0.1bbb...b \times 2^E$$

- 위의 예에서 정규화된 표현은  $0.1101 \times 2^5$

## 부동소수점 표현의 예 ( $0.1101 \times 2^5$ )

- 부호(S) 비트 = 0
- 지수(E) = 00000101
- 가수(M) = 1101 0000 0000 0000 0000 000



- ❖ 소수점 아래 첫 번째 비트는 항상 1이므로, 반드시 저장할 필요는 없음 → 가수 23 비트를 이용하여 소수점 아래 24 자리 수까지 표현 가능

## 바이어스된 지수(biased exponent)

- 지수를 바이어스된 수(biased number)로 표현
- 사용 목적
  - 0(혹은 0에 매우 가까운 값)에 대한 표현에서 모든 비트들이 0이 되도록 함으로써,  
0-검사(zero-test)가 용이하게 하기 위함  
➔ 0-검사가 정수 경우와 같은 방법으로 가능해짐

## 8-비트 바이어스된 지수값들(8-bit biased exponents)

지수 비트 패턴	절대값	실제 지수값	
		바이어스 = 127	바이어스 = 128
11111111	255	+128	+127
11111110	254	+127	+126
⋮	⋮	⋮	⋮
10000001	129	+2	+1
10000000	128	+1	0
01111111	127	0	-1
01111110	126	-1	-2
⋮	⋮	⋮	⋮
00000001	1	-126	-127
00000000	0	-127	-128

### 예제 3-27

10진수  $N = -13.625$ 에 대한 32-비트 부동소수점 표현을 구하라.

풀이

$$13.625_{10} = 1101.101_2 = 0.1101101 \times 2^4$$

부호( $S$ ) 비트 = 1 (−)

지수( $E$ ) = 00000100 + 10000000 = 10000100 (바이어스 128을 더한다)

가수( $M$ ) = 101101000000000000000000 (소수점 우측의 첫 번째 '1'은 제외)

<결과>

$S$	$E$	$M$
1	10000100	101101000000000000000000



## 부동소수점 수의 표현 범위

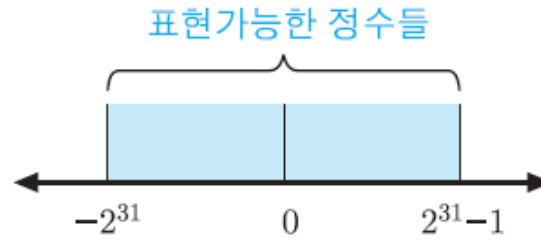
### □ 부동소수점 수의 표현 범위

- $0.5 \times 2^{-128} \sim (1 - 2^{-24}) \times 2^{127}$  사이의 양수들  
(대략  $1.47 \times 10^{-39} \sim 1.7 \times 10^{38}$ )
- $-(1 - 2^{-24}) \times 2^{127} \sim -(0.5 \times 2^{-128})$  사이의 음수들

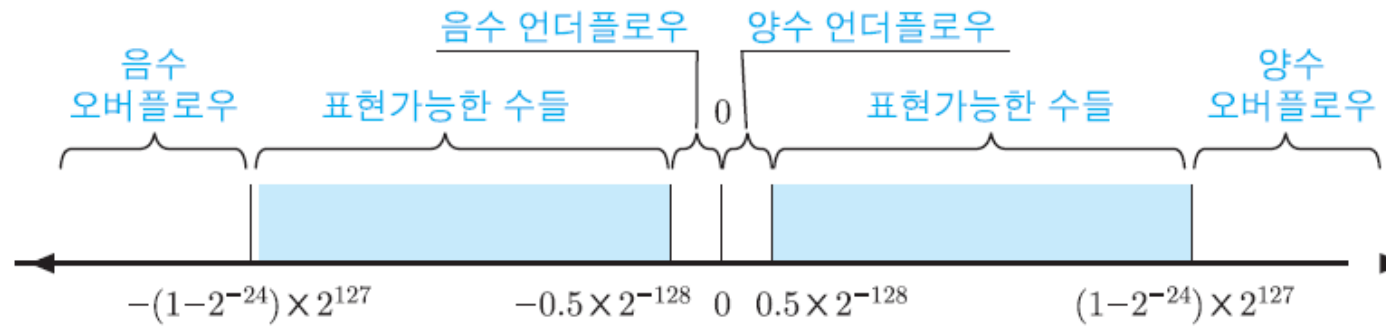
### □ 제외되는 범위

- $(1 - 2^{-24}) \times 2^{127}$ 보다 작은 음수 → 음수 오버플로우(negative overflow)
- $0.5 \times 2^{-128}$ 보다 큰 음수 → 음수 언더플로우(negative underflow)
- 0
- $0.5 \times 2^{-128}$ 보다 작은 양수 → 양수 언더플로우(positive underflow)
- $(1 - 2^{-24}) \times 2^{127}$ 보다 큰 양수 → 양수 오버플로우(positive overflow)

## 32-비트 데이터 형식의 표현 가능한 수의 범위



(a) 2의 보수 정수의 표현 범위



(b) 부동소수점 수의 표현 범위

## IEEE 754 표준 부동소수점 수의 형식

- 부동소수점 수의 표현 방식의 통일을 위하여 미국전기전자공학회(IEEE)에서 정의한 표준

- 32-비트 단일-정밀도 부동소수점 수의 표현

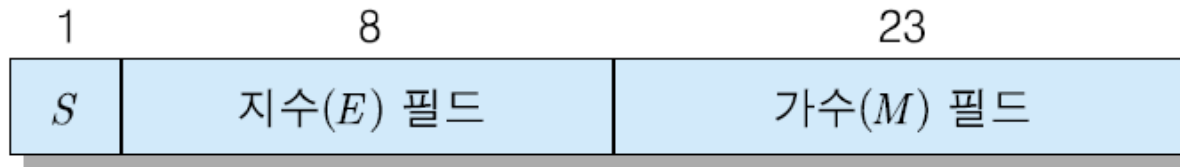
$$N = (-1)^S 2^{E-127} (1.M)$$

- 가수 : 부호화-크기 표현 사용
- 지수 필드 : 바이어스 127 사용
- $1.M \times 2^E$ 의 형태를 가지며, 소수점 아래의 M 부분만 가수 필드에 저장 (소수점 왼쪽의 저장되지 않는 1을 *hidden bit* 라고 부름)

- 64-비트 복수-정밀도 부동소수점 수의 표현

$$N = (-1)^S 2^{E-1023} (1.M)$$

## IEEE 754 표준 부동소수점 수의 형식 (계속)



(a) 단일-정밀도 형식(single-precision format)



(b) 복수-정밀도 형식(double-precision format)

### 예제 3-28

10진수  $-13.625$ 를 IEEE 754 단일-정밀도 표준 형식으로 표현하라.

풀이

$$13.625_{10} = 1101.101_2 = 1.101101 \times 2^3$$

부호( $S$ ) 비트 = 1 (−)

지수  $E = 00000011 + 01111111 = 10000010$  (바이어스 127을 더한다)

가수  $M = 101101000000000000000000$  (소수점 좌측의 1은 제외한다)

<결과>

$S$	$E$	$M$
1	10000010	101101000000000000000000

## 예외(exception) 경우를 포함한 IEEE 754 표준

### □ 예외 경우를 포함한 정의 (32-비트 형식)

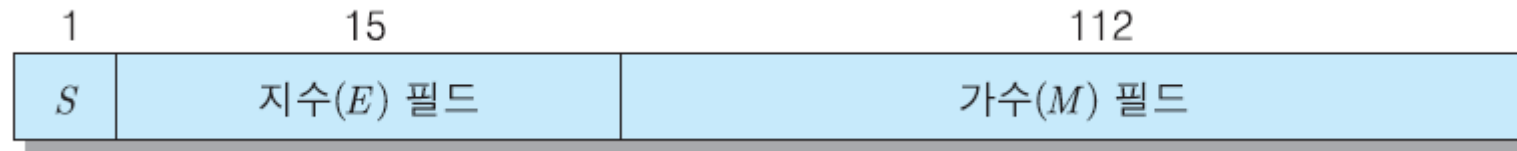
- 만약  $E = 255$ 이고  $M \neq 00$ 이면,  $N = \text{NaN}$
- 만약  $E = 255$ 이고  $M = 00$ 이면,  $N = (-1)^S \infty$
- 만약  $0 < E < 255$  이면,  $N = (-1)^S 2^{E-127} (1.M)$
- 만약  $E = 0$ 이고  $M \neq 00$ 이면,  $N = (-1)^S 2^{-126} (0.M)$
- 만약  $E = 0$ 이고  $M = 00$ 이면,  $N = (-1)^S 0$

### □ 예외 경우를 포함한 정의 (64-비트 형식)

- 만약  $E = 2047$ 이고  $M \neq 00$ 이면,  $N = \text{NaN}$
- 만약  $E = 2047$ 이고  $M = 00$ 이면,  $N = (-1)^S \infty$
- 만약  $0 < E < 2047$  이면,  $N = (-1)^S 2^{E-1023} (1.M)$
- 만약  $E = 0$ 이고  $M \neq 00$ 이면,  $N = (-1)^S 2^{-1022} (0.M)$
- 만약  $E = 0$ 이고  $M = 00$ 이면,  $N = (-1)^S 0$

## IEEE 754-2008에서 추가된 정의

### □ 4배수-정밀도(quadruple-precision: 128-비트) 부동소수점 수의 표현



$$N = (-1)^S 2^{E-16383} (1.M)$$

- 가수 : 부호화-크기 표현 사용
- 지수 필드 : 바이어스 16383 사용 (범위:  $-16383 \sim +16383$ )

표 3-5 IEEE 754 형식의 주요 파라미터들

파라미터	부동소수점 형식		
	단일 정밀도	복수 정밀도	4배수 정밀도
비트 수	32	64	128
지수 필드 폭(비트)	8	11	15
가수 필드 폭(비트)	23	52	112
지수 바이어스	127	1023	16383
최대 지수값	127	1023	16383
최소 지수값	-126	-1023	-16382
정수의 대략적 표현 범위 (최소/최대)	$\sim \pm 10^{-38} / \pm 10^{+38}$	$\sim \pm 10^{-308} / \pm 10^{+308}$	$\sim \pm 10^{-4932} / \pm 10^{+4932}$



## 3.7 부동소수점 산술 연산

### 3.7.1 덧셈과 뺄셈

- 지수들이 일치되도록 조정 (alignment)
- 가수들 간의 연산(더하기 혹은 빼기) 수행
- 결과를 정규화 (normalization)

[참고] 10진 부동소수점 산술의 예

$$(135 \times 10^{-5}) + (246 \times 10^{-3}) \quad \rightarrow \quad \begin{array}{r} 1.35 \times 10^{-3} \\ + 246 \times 10^{-3} \\ \hline 247.35 \times 10^{-3} \end{array}$$

### 예제 3-29

부동소수점 수들 간의 덧셈 ( $0.110100 \times 2^3 + 0.111100 \times 2^5$ )을 수행하라.

**풀이**

이 덧셈은 아래와 같은 세 단계를 통하여 수행될 수 있다.

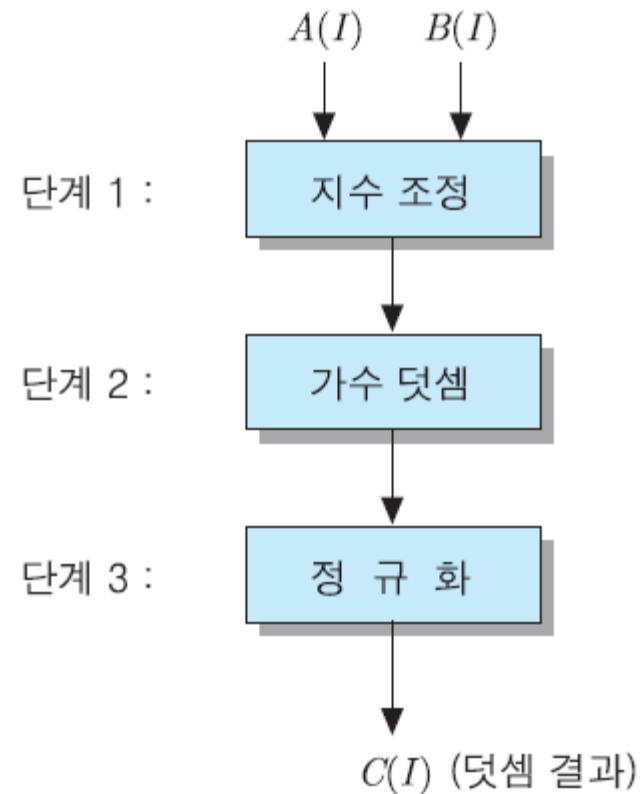
$$\begin{array}{rcl}
 \begin{array}{l} 0.110100 \times 2^3 \\ + 0.111100 \times 2^5 \\ \hline \end{array} & \xrightarrow{\text{(1) 지수 조정}} & \begin{array}{l} 0.001101 \times 2^5 \\ + 0.111100 \times 2^5 \\ \hline 1.001001 \times 2^5 \end{array} \\
 & \nearrow \text{(2) 더하기} & \xrightarrow{\text{(3) 정규화}} 0.1001001 \times 2^6 \\
 & & \text{(최종 결과)}
 \end{array}$$

## 부동소수점 산술의 파이프라이닝

- 연산 과정을 독립적 단계들로 분리 가능
- 단계 수만큼의 속도 향상
- 대규모의 부동소수점 계산을 처리하는 거의 모든 슈퍼컴퓨터들에서 채택

[예] 수 배열(number array)들 간의 덧셈

$$C(I) = A(I) + B(I)$$



## 3.7.2 부동소수점 곱셈 / 나눗셈

### □ 2진수 부동소수점 곱셈 과정

- ① 가수들을 곱한다
- ② 지수들을 더한다
- ③ 결과값을 정규화

### □ 2진수 부동소수점 나눗셈 과정

- ① 가수들을 나눈다
- ② 피제수의 지수에서 제수의 지수를 뺀다
- ③ 결과값을 정규화

#### [부동소수점 곱셈의 예]

$$(0.1011 \times 2^3) \times (0.1001 \times 2^5)$$

#### <가수 곱하기>

$$1011 \times 1001 = 01100011$$

#### <지수 더하기>

$$3 + 5 = 8$$

#### <정규화>

$$\begin{aligned} &0.01100011 \times 2^8 \\ &= 0.1100011 \times 2^7 \quad (\text{결과값}) \end{aligned}$$

## 부동소수점 연산 과정에서 발생 가능한 문제점

### □ 지수 오버플로우(exponent overflow)

- 양의 지수값이 최대 지수값을 초과

➔ 수가 너무 커서 표현될 수 없는 상태이므로,  $+\infty$  또는  $-\infty$ 로 세트

### □ 지수 언더플로우(exponent underflow)

- 음의 지수값이 최대 지수값을 초과

➔ 수가 너무 작아서 표현될 수 없는 상태이므로, 0으로 세트

## 부동소수점 연산 과정에서 발생 가능한 문제점 (계속)

### □ 가수 언더플로우(mantissa underflow)

- 가수의 소수점 위치 조정 과정에서 비트들이 가수의 우측 편으로 넘치는 상태 → 반올림 (rounding) 적용

### □ 가수 오버플로우(mantissa overflow)

- 같은 부호를 가진 두 가수들을 덧셈하였을 때 올림수가 발생  
→ 재조정(realignment) 과정을 통하여 정규화