산술 논리 연산

컴퓨터구조론 3장

연세대학교

내 용

연세대학교

3.1 ALU의 구성 요소

연세대학교

- ALU (Arithmetic and Logical Unit)
  - 컴퓨터의 기본이 되는 산술, 논리 연산을 수행하는 장치
- ALU의 구성 요소
  - 산술 연산장치 : 산술 연산들(+, -, ×, ÷)을 수행
  - 논리 연산장치 : 논리 연산들(AND, OR, XOR, NOT 등)을 수행
  - 상태 레지스터 : 연산 결과의 상태를 나타내는 플래그(flag)들을 저장하는 레지스터 (C, V, N, Z 플래그)
- ALU와 제어신호
  - 제어장치에서 제공되는 제어신호에 따라서 ALU가 수행하는 연산이 정해짐

3.2 정수의 표현

연세대학교

- 2진수 체계 (binary number system)
  - 0, 1, 부호 및 소수점으로 수를 표현 (실수 포함)

    [예] $-13.625_{10} = -1101.1012_2$
- 2진수를 10진수로 변환하는 방법 :

  2진수 $a_{n-1}\ a_{n-2} \ldots a_1\ a_0 \cdot a_{-1}\ a_{-2} \ldots a_{-(n-1)}\ a_{-n}$ ➔ 10진수 A.B
  - 정수부분의 변환

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

  - 소수부분의 변환

$$B = a_{-1} \times 2^{-1} + a_{-2} \times 2^{-2} + \ldots + a_{-(n-1)} \times 2^{-(n-1)} + a_{-n} \times 2^{-n} = \sum_{i=-1}^{-n} a_i 2^i$$

연세대학교

[예] 1101.101 = 정수부분변환 + 소수부분변환

2^3 2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3} : 자리수(weight)

1 1 0 1 . 1 0 1

$= 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$

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

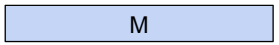
$= 8 + 4 + 1 + 0.5 + 0.125 = 13.625$

- 정수(integer)의 표현 : 정수를 n개의 2진 비트로 표현
  - 부호 없는 정수(unsigned integer)의 표현
    - 0과 양수만 표현
  - 부호 있는 정수(signed integer)의 표현
    - 0, 양수, 음수를 모두 표현

부호 없는 정수의 표현

연세대학교

- 부호 없는 정수 표현



  - M : 정수 크기의 n비트 2진수 표현
  - 표현범위: 0 ~ 2^n-1 (예) 8비트는 0 ~ 255

- [예]
  - 00000000_2 = 0 (가장 작은 부호 없는 정수)
  - 00000001_2 = 1
  - 00111001_2 = 57
  - 10000000_2 = 128
  - 11111111_2 = 255 (가장 큰 8비트 부호 없는 정수)

부호 있는 정수의 표현

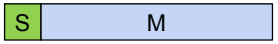
연세대학교

- 부호 있는 정수의 표현 – 여러 가지 방법이 있음
  - 부호화-크기 표현(signed-magnitude representation)
  - 1의 보수 표현(1's complement representation)
  - 2의 보수 표현(2's complement representation)
  - 바이어스(bias) 표현

- 현재의 대부분의 컴퓨터에서 부호 있는 정수의 표현에 2의 보수 표현 방법을 사용
  - 부호 없는 정수와 부호 있는 정수의 덧셈, 뺄셈 연산 장치가 같아서 하드웨어가 간단함

부호화-크기 표현

연세대학교



- 최상위 비트(S): 부호(sign) 비트 → 양수 0, 음수 1
  나머지 n-1 개의 비트(M): 크기(magnitude)의 2진수 표기

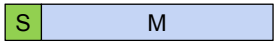
  [예] + 9 = 0 0001001 + 35 = 0 0100011

  - 9 = 1 0001001 - 35 = 1 0100011
- 부호화-크기로 표현된 2진수(a_{n-1} a_{n-2} ... a_1 a_0)를 10진수로 변환

  [예] 0 0100011 = + $(1\times2^5+1\times2^1+1\times2^0)$ = (32 + 2 + 1) = 35

  1 0001001 = - $(1\times2^3+1\times2^0)$ = -(8 + 1) = -9
- 단점
  - 덧셈, 뺄셈 시에 부호비트와 크기부분을 별도로 처리해야 함
  - 두 개의 0 표현 존재: 0 0000000 (+0), 1 0000000 (-0)
- 표현 범위 : $-2^{n-1}-1 \sim 2^{n-1}-1$ (예) 8비트는 -127 ~ 127

보수 표현



- 1의 보수(1's complement) 표현
  - 양수: S=0, M=크기의 2진수표현
  - 음수: S=1, M=크기의 1의 보수 (모든 비트들을 반전: 0→1, 1→ 0)
- 2의 보수(2's complement) 표현
  - 양수: S=0, M=크기의 2진수표현
  - 음수: S=1, M=크기의 2의 보수 (1의 보수 + 1)

[예] + 9 = 0 0001001 + 35 = 0 0100011

- 9 = 1 1110110 (1의 보수) - 35 = 1 1011100 (1의 보수)
- 9 = 1 1110111 (2의 보수) - 35 = 1 1011101 (2의 보수)

8비트 보수 표현 정수

10진수	1의 보수	2의 보수
127	01111111	01111111
126	01111110	01111110
:	:	:
1	00000001	00000001
+0	00000000	00000000
−0	11111111	–
−1	11111110	11111111
−2	11111101	11111110
:	:	:
−126	10000001	10000010
−127	10000000	10000001
−128	–	10000000

두 개의 0표기

- 표현 범위 :
  - 1의 보수 표현: $-2^{n-1}-1 \sim 2^{n-1}-1$ (2개의 0)
  - 2의 보수 표현: : $-2^{n-1} \sim 2^{n-1}-1$ (음수가 1개 더 많음)

[예] 8비트의 표현 범위

- 1의 보수 : $-(2^7-1) \sim +(2^7-1)$ → -127 ~ 127
- 2의 보수 : $-2^7 \sim +(2^7-1)$ → -128 ~ 127

- 같은 2진 표현에 대한 부호 없는 정수 A와 2의 보수 표현 부호 있는 정수 B의 관계
  - 부호비트=1: $B = -(2^n-A) = A - 2^n$
  - 부호비트=0: B = A
- 음수 –A의 2의 보수 표현은 부호 없는 정수 2^n-A의 2진수 표현과 같음

2의 보수 표현 정수 → 10진수 변환

- 방법 1 : $A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$
  - 양수(a_{n-1} = 0)를 10진수로 변환하는 방법: $A = \sum_{i=0}^{n-2} a_i 2^i$
  - 음수(a_{n-1} = 1)를 10진수로 변환하는 방법: $A = -2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$

[예] $10101110 = -128 + (1 \times 2^5 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1)$
$= -128 + (32 + 8 + 4 + 2) = -82$

- 방법 2 : 음수인 경우 2의 보수로 변환하여 크기를 구한 후 - 부호 표시

[예] 10101110

2의 보수 = 01010010

-크기 = $-01010010 = -(2^6 + 2^4 + 2^1)$
$= -(64 + 16 + 2) = -82$

비트 확장 (Bit Extension)

연세대학교

- 데이터의 길이(비트 수)를 확장
  - 데이터의 표현을 같은 값을 갖는 더 긴 비트로 표현하는 것
  - 목적: 데이터를 더 긴 비트의 레지스터에 저장하거나
    더 긴 데이터와의 연산 수행하기 위함
- 비트 수 확장 방법
  - 부호 없는 정수: zero 확장

8비트	16비트
0100 0010 (66)	0000 0000 0100 0010 (66)
1000 0010 (130)	0000 0000 1000 0010 (130)

  - 부호 있는 정수: 표현 방법에 따라서 확장 방법이 다름
    - 보수 표현: 부호 확장
    - 부호화-크기 표현:
      - 부호비트를 확장된 최상위 비트로 이동
      - 나머지 확장 비트들과 기존 부호비트은 zero 확장

부호화-크기 표현의 비트 확장

연세대학교

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)
```


보수 표현의 비트 확장

연세대학교

- 부호 확장(sign-bit extension)
  - 확장되는 상위 비트들을 부호 비트와 같은 값으로 설정

8비트	16비트
0100 0010 (66)	0000 0000 0100 0010 (66)
1000 0010 (-126)	1111 1111 1000 0010 (-126)

예제 3-8

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

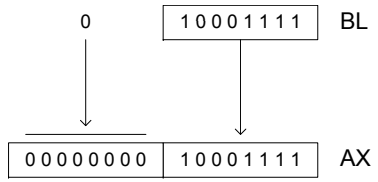
[풀이]

```
+21 =          00010101  (8-비트 2의 보수)
+21 = 0000000000010101  (16-비트 2의 보수)
−21 =          11101011  (8-비트 2의 보수)
−21 = 1111111111101011  (16-비트 2의 보수)
```

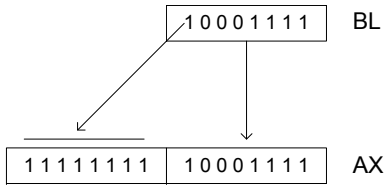

(참고) x86 프로세서의 비트확장 명령어

연세대학교

- MOVZX : zero 확장 명령어
  - MOVZX AX, BL



- MOVSX : sign 확장 명령어
  - MOVSX AX, BL



3.3 논리 연산

연세대학교

- 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

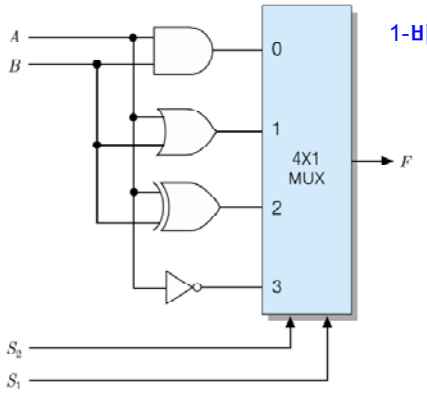
  - 비트들을 조작하는 데 사용 가능

논리 연산을 위한 하드웨어 모듈

연세대학교

- 하드웨어의 구성
  - 게이트들을 사용하여 여러 논리 연산 수행
  - 선택 신호(S_2S_1)에 의하여 멀티플렉서의 4 연산결과 중 하나를 출력

1-비트 연산장치



S_2	S_1	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	$F = \overline{A}$	NOT

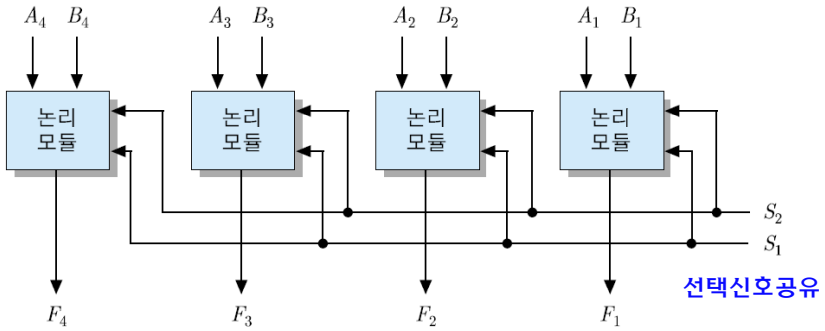
N-비트 논리 연산장치

연세대학교

- N-비트 데이터들을 위한 논리 연산장치
  - N개의 기본 논리 모듈들을 병렬로 접속

비트단위 연산:
대응되는 비트들 간에 연산 수행

[예] 4-비트 논리 연산장치



선택신호공유

논리 연산의 활용

연세대학교

- 선택적-set 연산: 특정비트를 1로 설정 → **OR** 이용

  A = 1 0 0 1 0 0 1 0 (연산 전)
  B = 0 0 0 0 1 1 1 1 **(OR**
  A = 1 0 0 1 1 1 1 1 (연산 결과)

- 선택적-clear 연산 : 특정비트를 0으로 설정 → **AND** 이용

  A = 1 1 0 1 0 1 0 1 (연산 전)
  B = 0 0 0 0 1 1 1 1 **(AND**
  A = 0 0 0 0 0 1 0 1 (연산 결과)

  mask 연산이라고도 함

- 선택적-보수 연산 : 특정비트를 보수화 → **XOR** 이용

  A = 1 0 0 1 0 1 0 1 (연산 전)
  B = 0 0 0 0 1 1 1 1 **(XOR**
  A = 1 0 0 1 1 0 1 0 (연산 결과)

삽입 연산

연세대학교

- 삽입(insert) 연산 :
  - 데이터의 특정 위치의 값을 새로운 비트 값으로 변경(삽입)
  - 방법 : ① 값을 변경할 비트 위치들에 대해 AND 연산(마스크) 수행
    ② 새로이 삽입할 비트 값들과 OR 연산을 수행

```
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

연세대학교

- 비교(compare) 연산
  - A와 B 레지스터의 내용을 비교
  - 방법: XOR 연산
    - 각 비트에 대해서
      - 대응되는 비트들의 값이 같으면, 결과 = 0
      - 서로 다르면, 결과 = 1
    - 모든 비트들이 같으면: 결과 = 00000000 (Z 플래그=1)

```
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) → 같지 않음
```


3.4 시프트(shift) 연산

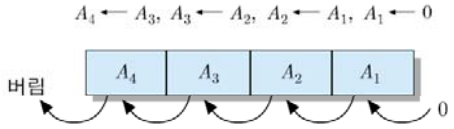
연세대학교

- 시프트(shift) 연산
  - 데이터 비트들을 왼쪽 또는 오른쪽으로 이동시키는 연산
- 시프트 연산의 종류
  - 논리적 시프트 (logical shift)
  - 산술적 시프트 (arithmetic shift)
  - 순환 시프트 (circular shift) 또는 회전(rotate)

3.4 시프트(shift) 연산

연세대학교

- 논리적 시프트 (logical shift) :
  - 좌측 시프트(left shift) - 모든 비트들을 좌측으로 이동
    - 최하위 비트(A1) ← 0



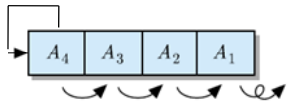
  - 우측 시프트(right shift) - 모든 비트들을 우측으로 이동
    - 최상위 비트(A4) ← 0

      0 → A4, A4→A3, A3→A2, A2→A1
  - 부호 없는 정수의 시프트에 사용

산술적 시프트

연세대학교

- 산술적 시프트(arithmetic shift) :
  - 산술적 우측 시프트(arithmetic shift-right) - 부호 비트 불변



    - A4 (불변), A4 → A3, A3 → A2, A2 → A1
  - 산술적 좌측-시프트(arithmetic shift-left)
    - A4(불변), A3 ← A2, A2 ← A1, A1 ← 0 , 또는
    - A4←A3, A3 ← A2, A2 ← A1, A1 ← 0 (주로 사용됨)
      - 논리적 우측시프트와 같음
      - 부호비트(A4)가 바뀌면 Overflow 플래그가 1로 설정됨
  - 부호 있는 정수에 대한 시프트

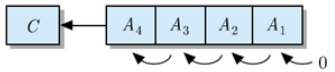
시프트와 곱셈/나눗셈

연세대학교

- 시프트와 곱셈/나눗셈 연산
  - 좌측 시프트: 1비트 좌측시프트 → x 2
    k비트 좌측시프트 → x 2^k
  - 우측 시프트: 1비트 우측시프트 → ÷ 2
    k비트 우측시프트 → ÷ 2^k

0100 (4) ← shift left **0010** (2) → shift right **0010** (1)

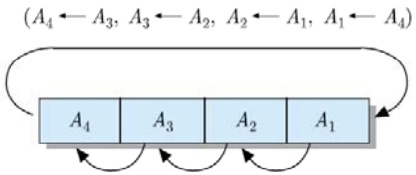
  - 부호없는 정수에 대한 곱셈/나눗셈은 논리적 시프트를 사용
  - 부호있는 정수에 대한 곱셈/나눗셈은 산술적 시프트를 사용
- 버려지는 비트(좌측시프트에서는 A4, 우측시프트에서는 A1)는 대개 Carry 플래그로 저장됨



회전

연세대학교

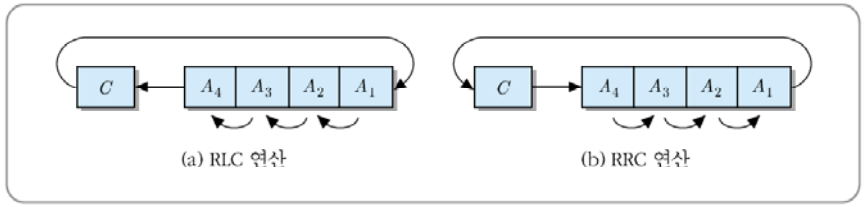
- 회전(rotate) 또는 순환 시프트(circular shift)
  - 최상위 혹은 최하위 비트(버려지는 비트)를 반대편 끝에 있는 비트 위치로 이동
  - 좌측 회전 (rotate-left) :



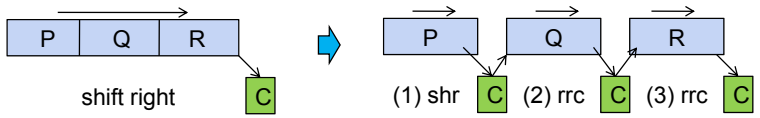
  - 우측 회전(rotate-right)
    - A4 → A3, A3 → A2, A2 → A1, A1 → A4

연세대학교

- C 플래그를 포함한 회전 연산
  - RLC(rotate left with carry) / RRC(rotate right with carry)



  - 여러 워드에 대한 시프트에 이용됨



(참고) x86 프로세서의 시프트 명령어

연세대학교

Instructions	동작	구분
SHL dst, count	shift left	logical shift
SHR dst, count	shift right	
SAL dst, count	shift arithmetic left (SHL과 같음)	arithmetic shift
SAR dst, count	shift arithmetic right	
ROL dst, count	rotate left	rotate
ROR dst, count	rotate right	
RCL dst, count	rotate carry left	rotate with carry
RCR dst, count	rotate carry right	

- SHL AX, 1 (1비트 왼쪽 논리 시프트)
- SAL AX, CL (CL=3이면 3비트 오른쪽 산술 시프트)

3.5 정수의 산술 연산

연세대학교

- 기본적인 산술 연산들

$$
\begin{aligned}
A &\leftarrow \overline{A} + 1 \quad ; \text{보수화(2의 보수 변환)} \quad \text{(-A)} \\
A &\leftarrow A + B \quad ; \text{덧셈} \\
A &\leftarrow A - B \quad ; \text{뺄셈} \\
A &\leftarrow A \times B \quad ; \text{곱셈} \\
A &\leftarrow A \div B \quad ; \text{나눗셈} \\
A &\leftarrow A + 1 \quad ; \text{증가(increment)} \\
A &\leftarrow A - 1 \quad ; \text{감소(decrement)}
\end{aligned}
$$

덧셈

연세대학교

- 부호 있는 정수(2의 보수 표현법 사용)들의 덧셈 방법
  - 두 수의 2진 표현을 그대로 더하고, 올림수는 무시함

예제 3-20

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

[풀이]

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

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

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

$$\begin{array}{r} 1101 \\ +\ 0011 \\ \hline (1)\ 0000 = 0 \end{array}$$

버림

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

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

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

$$\begin{array}{r} 1100 \\ +\ 1111 \\ \hline (1)\ 1011 = -5 \end{array}$$

덧셈 오버플로우

연세대학교

- 덧셈 결과가 부호 있는 정수의 표현범위를 초과하여 결과값이 틀리게 되는 상태
- 같은 부호의 덧셈에서 결과의 부호비트가 반대로 될 때 Overflow가 발생함

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

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

병렬 가산기(parallel adder)

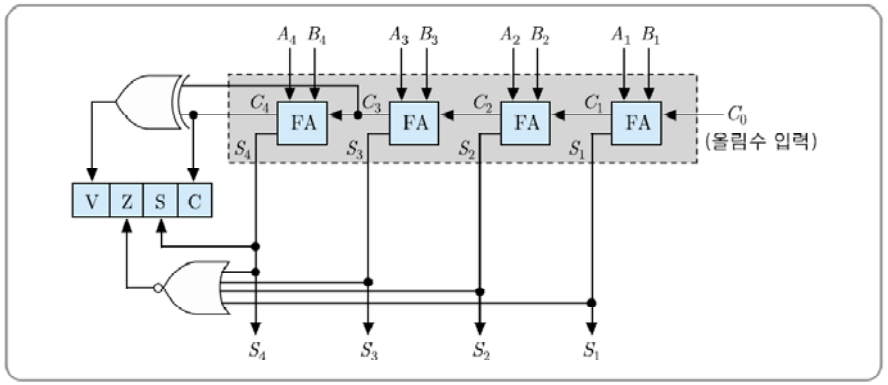
연세대학교

- 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) – unsigned overflow
  - S 플래그 : 부호(sign)
  - Z 플래그 : 0(zero)
  - V 플래그 : 오버플로우(overflow) – signed overflow

4-비트 병렬가산기와 상태비트제어회로

연세대학교



Carry와 Overflow 플래그 설정

연세대학교

- Carry 플래그
  - 검출방법: C_4 = 1일 때 올림수 발생 ➔ C 플래그 = 1
  - C 플래그가 1이면 덧셈 결과가 unsigned 정수의 표현 범위를 벗어났음을 의미함.
- Overflow 플래그
  - 검출방법:
    - 같은 부호의 덧셈에서 결과의 부호비트가 반대로 될 때 V=1
      ➔ $V = A_4' B_4' C_4 \vee A_4 B_4 C_4'$
    - (다른 방법) C_3와 C_4가 다를 때 V=1
      ➔ $V = C_4 \oplus C_3$ (앞 쪽의 회로)
  - V 플래그가 1이면 덧셈 결과가 signed 정수의 표현 범위를 벗어났음을 의미함

뺄셈

연세대학교

- 덧셈을 이용하여 수행 (A : 피감수(minuend), B : 감수(subtrahend))
  A - (B) = A + (-B) = A + (B의 2의 보수)

(a) (+2) − (+6) = (+2) + (−6) = −4

```
  0010      <-- 2의 보수
+ 1010
------
  1100 = −4
```

(b) (+5) − (+2) = (+5) + (−2) = +3

```
  0101
+ 1110
------
(1) 0011 = +3
```

- 부호 있는 정수와 부호 없는 정수에 대해서 모두 적용됨

뺄셈 오버플로우

연세대학교

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

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

$$\begin{array}{r} 0110 \\ +\ 0100 \\ \hline 1010 = -6 \end{array} \quad (C_4 \oplus C_3 = 0 \oplus 1 = 1 \ : \ \text{오버플로우 발생})$$

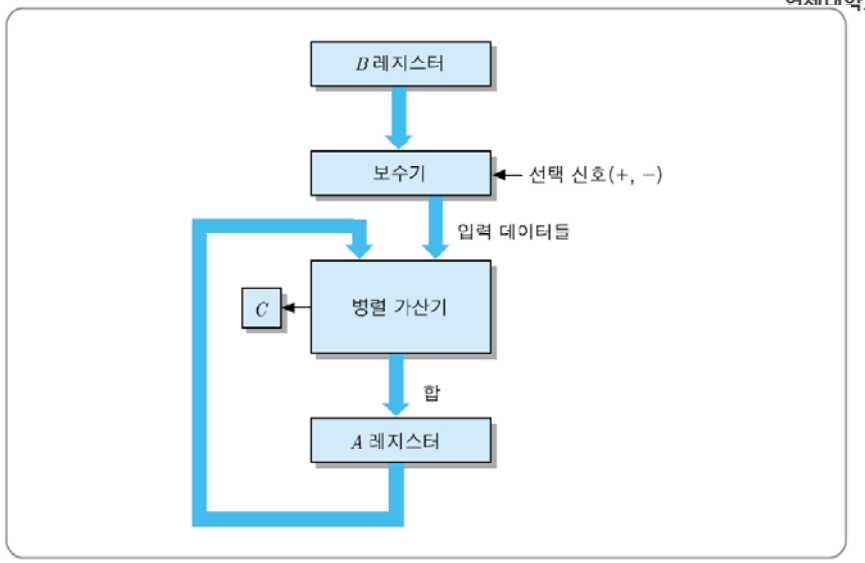
(b) $(-7) - (+6) = (-7) + (-6) = -13$

$$\begin{array}{r} 1001 \\ +\ 1010 \\ \hline (1)\ 0011 = +3 \end{array} \quad (C_4 \oplus C_3 = 1 \oplus 0 = 1 \ : \ \text{오버플로우 발생})$$

- 빌림수 검출 : C_4 = 0일 때 빌림수(unsigned overflow) ➔ C 플래그 = 1

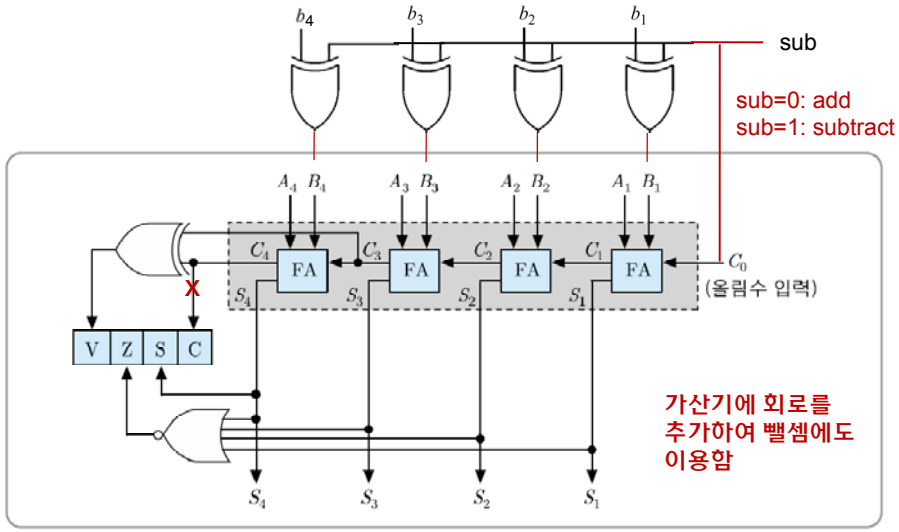
덧셈과 뺄셈 겸용 하드웨어 블록구성도

연세대학교



가감산기

연세대학교



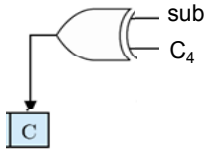
가감산기의 설계

연세대학교

- 2의 보수 회로
  - b의 2의 보수 = (b의 1의 보수) + 1
    - 1의 보수는 XOR 게이트를 이용하여 구현
      - b ⊕ 0 = b (덧셈) , b ⊕ 1 = b' (뺄셈) → B = b ⊕ sub
    - 1을 추가로 더하는 것은 Carry 입력 (C_0)를 이용하여 구현
      - C_0 = 0 (덧셈), C_0 = 1 (뺄셈) → C_0 = sub
- Carry 플래그 회로 – 수정해야 함
  - 덧셈: C_4 = 1일 때 올림수 발생 ➔ C 플래그 = 1
  - 뺄셈: C_4 = 0일 때 빌림수 발생 ➔ C 플래그 = 1

  ➔ C = $C_4 \oplus$ sub
- Overflow, Zero, Negative 플래그 회로
  - 가산기와 같음

수정 회로



3.6 부동소수점 수의 표현

연세대학교

- 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

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

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

부동소수점 표현 (계속)

연세대학교

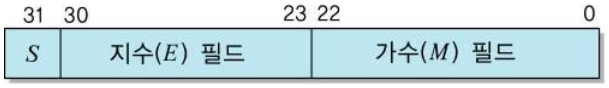
- 10진 부동소수점 수(decimal floating-point number)
  - [예] 274,000,000,000,000 → 2.74×10^{14}
    0.00000000000274 → 2.74×10^{-12}

- 2진 부동소수점 수(binary floating-point number)
  - 기수 B = 2
  - 단일-정밀도(single-precision) 부동소수점 수 : 32 비트
  - 복수-정밀도(double-precision) 부동소수점 수 : 64 비트

단일-정밀도 부동소수점 수 형식

연세대학교

- 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 ≈ $1.47 \times 10^{-39} \sim 1.7 \times 10^{38}$
  - [비교] 32-비트 고정소수점 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

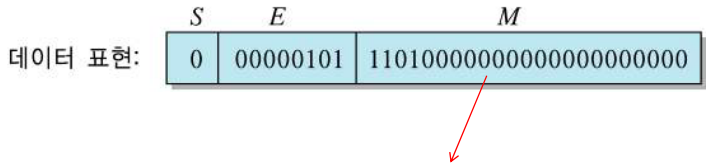
연세대학교

- 같은 수에 대한 부동소수점 표현이 여러 가지가 존재
  0.1101×2^5
  11.01×2^3
  0.001101×2^7
- 정규화된 표현(Normalized representation)
  - 수에 대한 표현을 한 가지로 통일하기 위한 방법
    $\pm 0.1bbb...b \times 2^E$
  - 위의 예에서 정규화된 표현은 0.1101×2^5

부동소수점 표현의 예

연세대학교

- 예: 0.1101×2^5
  - 부호(S) = 0
  - 지수(E) = 00000101
  - 가수(M) = 1101 0000 0000 0000 0000 000
- 소수점 아래 첫 번째 비트는 항상 1이므로, 저장할 필요가 없음 → 가수 23 비트를 이용하여 소수점 아래 24 자리 수까지 표현 가능



1010 ... 000로 표기하면 소수점이하 24자리까지 표현가능

바이어스된 지수(biased exponent)

연세대학교

- 지수를 바이어스된 수(biased number)로 표현
  - 지수 + 바이어스 수 의 2진수 표현으로 나타냄
- 사용 목적
  - 지수의 크기 비교가 용이함
  - 0에 대한 표현이 모든 비트들이 0이 되어서, 0-검사(zero-test)가 용이함

8-비트 바이어스된 지수값들

연세대학교

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

바이어스된 지수를 사용한 표현

연세대학교

- 바이어스값 = 128일 때, N = -13.625에 대한 부동소수점 표현

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

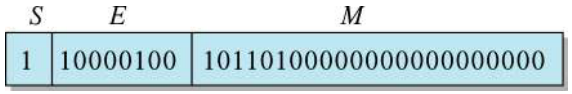
  부호(S) 비트 = 1 (-)

  지수(E) = 00000100 + 10000000 = 10000100

  (바이어스 128을 더한다)

  가수(M) = 10110100000000000000000

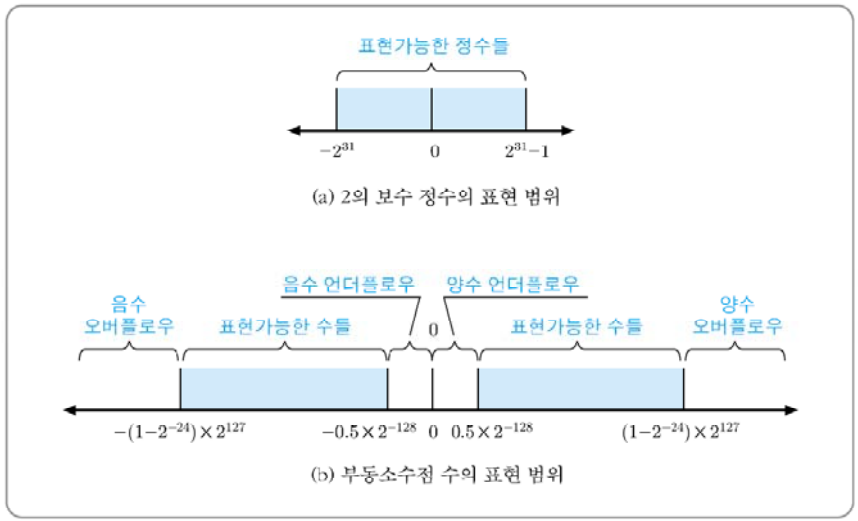
  (소수점 우측의 첫 번째 1은 제외)



바이어스-128

32-비트 데이터 형식의 표현 가능한 수의 범위

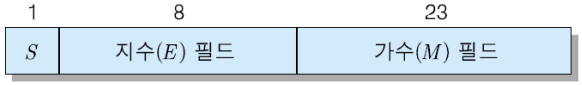
연세대학교



IEEE-754 표준 부동소수점 수의 형식

연세대학교

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



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

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

연세대학교

- 표현 방법
  - 가수(M) : 부호화-크기 표현 사용
    (최상위 비트 1은 생략: hidden bit 라고 부름)
  - 지수 필드 : 바이어스 127 (바이어스 1023) 사용
    단일정밀도: $N = (-1)^S \times 1.M \times 2^{E-127}$
    복수정밀도: $N = (-1)^S \times 1.M \times 2^{E-1023}$

IEEE 754 형식 표현 예

연세대학교

- 예: $13.625_{10} = 1101.101_2 = 1.101101 \times 2^3$
  - 부호(S) 비트 = 1 (-)
  - 지수 E = 00000011 + 01111111 = 10000010
    (바이어스 127을 더한다)
  - 가수 M = 10110100000000000000000
    (소수점 좌측의 1은 비트 표현에서 제외)