[JO2122] 컴퓨터구조

2022년 1학기

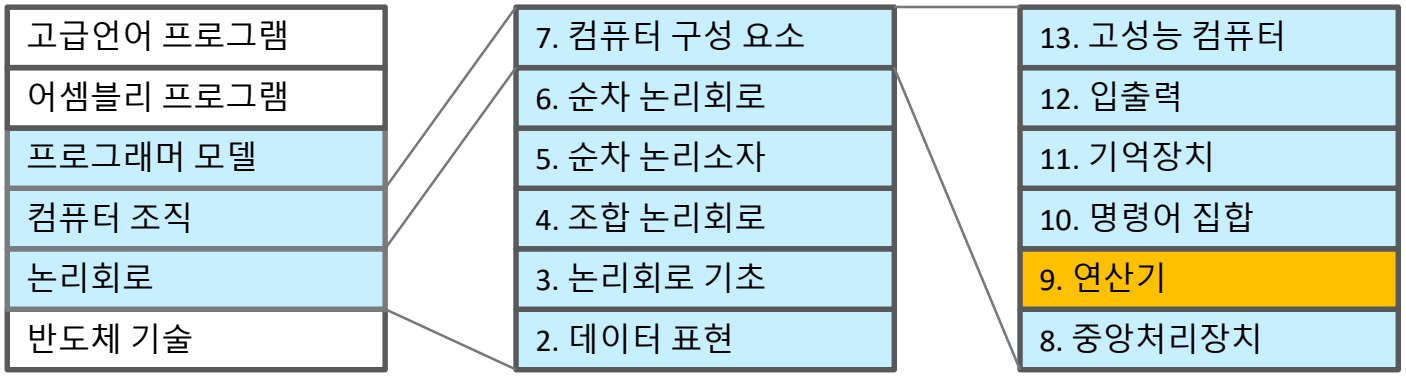
상명대학교 소프트웨어학과 박희민

9.1 연산기 개요
9.2 정수 표현
9.3 논리 연산
9.4 시프트 연산
9.5 산술 연산
9.6 실수
9.7 요약

2022-05-11

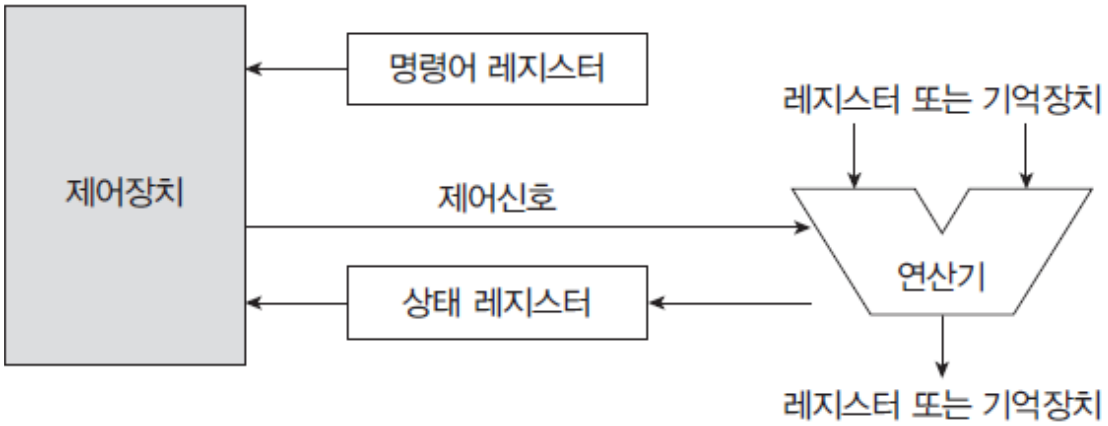
CHAP09 연산기

제9장 연산기



- 학습 목표
  - 정수와 실수를 컴퓨터 표현으로 변환할 수 있다.
  - 논리, 시프트, 산술 연산 실행 결과를 제시할 수 있다.
- 내용
  - 9.1 연산기 개요
  - 9.2 정수 표현
  - 9.3 논리 연산
  - 9.4 시프트 연산
  - 9.5 산술 연산
  - 9.6 실수
  - 9.7 요약

9.1 연산기 개요



[그림 9-1] 연산기 주변 회로

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

컴퓨터의 수

수의 종류	표현 방법	표현 범위	적용 연산
부호 없는 수	무게 수	$0 \sim 2^n-1$	사칙연산, 논리연산, 논리 시프트
정수	2의 보수	$-2^{n-1} \sim 2^{n-1}-1$	사칙연산, 산술 시프트
실수	IEEE754 형식	소수점이 있는 수	실수 사칙연산

9.2 정수 표현

- 소수점이 없는 수
  - 부호 없는 수(unsigned number): 2장
  - 정수(음수 포함): 9.2절

- 학습 목표
  - 정수를 컴퓨터 표현 방법으로 변환할 수 있다.
  - 수의 표현 범위를 제시할 수 있다.

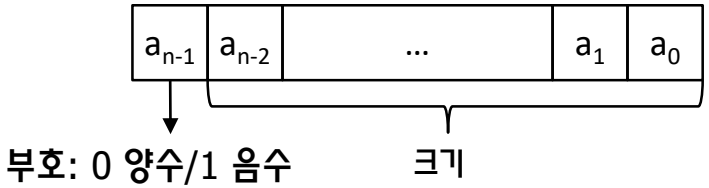
- 내용

9.2.1 부호화 크기 (signed magnitude)

9.2.2 보수 (complement)

9.2.3 2의 보수(2's complement) - 표준

9.2.1 부호화 크기

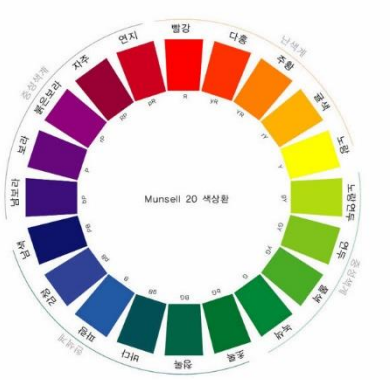


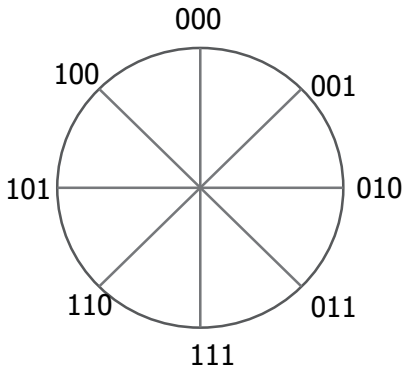
$$N = (-1)^{a_{n-1}} \times \sum_{i=0}^{n-2} a_i \times 2^i$$

- 부호 비트(sign bit): $S = a_{n-1}$
- [예제 9-1] 19과 -19에 대한 8비트 부호화 크기는?
  - +19: 0_001_0011: 0001_0011
  - -19: 1_001_0011: 1001_0011
- n비트 표현 범위: $-(2^{n-1}-1) \sim +(2^{n-1}-1)$
- 특징
  - 0이 두 개: 0000_0000, 1000_0000
  - 덧셈, 뺄셈할 때 부호를 별도로 고려해야 한다.

9.2.2 보수

- R 진법의 수 N에 대한 보수(complement)
  - (R-1)의 보수: $N + C_{R-1} = R^n - 1$
  - R의 보수: $N + C_R = R^n$
  - $C_R = C_{R-1} + 1$
  - $R^n = 10....0$ (0이 n개)





[보색과 보수]

- [예제 9-2] (R-1)의 보수와 R의 보수 구하기
  - 3자리 10진수 457
    - 9의 보수: (1000-1) – 457 = 542
    - 10의 보수: 9의 보수 + 1 = 543
  - 8비트 2진수 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)

```
  1111_1111
- 0011_1000
  1100_0111
```

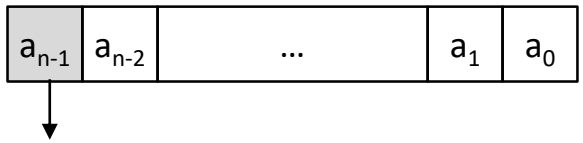
- 2의 보수: 1의 보수 + 1

- [예제 9-3] 8비트 2진수의 보수
  - 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
+ 00000001
-----------
 (1)00000000
```

제거 ←

9.2.3 2의 보수



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

- [예제 9-4] 10진수 +43과 –43에 대한 8비트 2의 보수 표현
  - $43_{10} = 101011_2$
  - +43: 0010_1011
  - –43: (1) 1_0000_0000 – 0010_1011 = 1101_0101
    (2) (+43에 대한 1의 보수) + 1 = 1101_0101

43 / 2 = 21 … 1
21 / 2 = 10 … 1
10 / 2 = 5 … 0
5 / 2 = 2 … 1
2 / 2 = 1 … 0
1 / 2 = 0 … 1

- [예제 9-5] 8비트 정수를 10진수로 변환하기

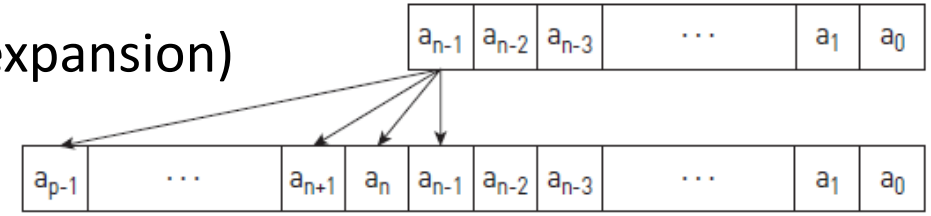
1) 0110_1010 64 + 32 + 8 + 2 = 106
2) 1001_0110 (1) –128 + (16 + 4 + 2) = -128 + 22 = –106
(2) 2의 보수: 0110_1010 = 106, ∴ –106

표현 범위와 부호 확장

- n비트 정수의 표현 범위: $-2^{n-1} \sim +(2^{n-1}-1)$

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

- 부호 확장 (sign expansion)



  - n비트 정수를 p비트로 확장 (n<p)
  - 늘어나는 비트를 부호 비트로 채워야 크기가 변하지 않는다.
- [예제 9-6] +106과 -106에 대한 16비트 표현
  - $+106_{10} = 0100\_1010_2 = 0000\_0000\_0100\_1010_2$
  - $-106_{10} = 1011\_0110_2 = 1111\_1111\_1011\_0110_2$

9.2 정수 표현 요약

- 부호화 크기 (signed magnitude)
  - 부호 + 크기: (부호=0: 양수, 부호=1: 음수)
  - 덧셈과 뺄셈을 구분해야 함.
- 2의 보수 (2's complement)
  - 음수 표현: 양수에 대한 2의 보수
  - 부호를 고려하지 않고 덧셈, 뺄셈을 할 수 있다.
  - 표준
- 부호 확장(sign expansion)
  - 비트를 확장할 때, 늘어나는 자리를 부호로 채운다.

9.3 논리 연산

- 논리 연산
  - 데이터를 부호 없는 수로 취급한다.
- 학습 목표
  - 레지스터의 데이터에 대한 논리 연산을 계산할 수 있다.
  - 논리 연산의 활용 방법을 이해한다.

- 내용

9.3.1 NOT 연산

9.3.2 AND 연산

9.3.3 OR 연산

9.3.4 XOR 연산

9.3.1 NOT 연산

- Not 연산
  - 오퍼랜드의 각 비트를 NOT
  - 명령어: NOT R // R ← R에 대한 1의 보수

- [예제 9-7] R0 = 0010_1011일 때, NOT R0 실행 결과는?
  - 명령어 NOT R0
  - 실행 결과 R0 = 1101_0100

9.3.2 AND 연산

- AND 연산
  - 비트 단위로 AND 연산을 수행한다.
  - 명령어 AND R1, R2 // R1 ← R1 AND R2
- 마스크(mask) 연산
  - 특정 비트를 0으로 만든다.
  - R2의 값 = 마스크 패턴

- [예제 9-8] AND R1, R2 실행 결과는?

(1)

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

(2)

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

9.3.3 OR 연산

- OR 연산
  - 비트 단위로 OR 연산을 수행한다.
  - 명령어 OR R1, R2 // R1 ← R1 OR R2
- 선택적 세트(selective set) 연산
  - 특정 비트를 1로 만든다.

- [예제 9-9] OR R1, R2 실행 결과는?

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

- [예제 9-10] 아스키 코드 대문자를 소문자로 바꾸기

R1 = 'A'

R1 = 'a'

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

9.3.4 XOR 연산

- XOR 연산
  - 비트 단위로 XOR 연산을 수행한다.
  - 명령어 XOR R1, R2 // R1 ← R1 XOR R2
- 선택적 보수 (selective complement) 연산
  - 특정 비트를 보수로 만든다.
- [예제 9-11] XOR R1, R2 실행 결과는?

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

- [예제 9-12] XOR R0, R0 실행 결과는?
  - 자신에 대한 XOR 연산 결과는 항상 0000_0000
  - 예)

```
      R0 = 1010_0110
XOR   R0 = 1010_0110
      R0 = 0000_0000
```

체크섬(checksum)

- 체크섬
  - 일련의 데이터에 대하여 XOR 연산으로 만들어진 값
  - 데이터 무결성 확인용
- [예제 9-13] [11, 23, 4A, 9B]에 대한 체크섬은?

(1) 초기값을 0으로 설정하고, 한 번에 하나씩 계산

0001_0001	0010_0011	0100_1010	1001_1011
XOR 0000_0000	XOR 0001_0001	XOR 0011_0010	XOR 0111_1000
0001_0001	0011_0010	0111_1000	1110_0011

(2) 한 번에 계산

```
  0001_0001
  0010_0011
  0100_1010
XOR 1001_1011
  1110_0011
```

9.3 논리 연산 요약

- 레지스터 데이터에 대한 비트 단위 논리 연산
- 논리 연산 종류
  - NOT 연산: 1의 보수
  - AND 연산: 특정 비트를 0으로 리셋. 마스크 연산
  - OR 연산: 특정 비트를 1로 세트. 선택적 세트 연산
  - XOR 연산: 특정 비트를 보수. 선택적 보수 연산. 체크섬 생성

9.4 시프트 연산

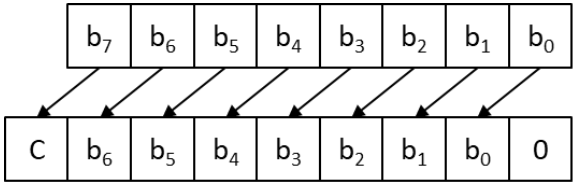
- 시프트 (shift) 연산
  - 왼쪽이나 오른쪽으로 한 비트씩 자리 이동
- 학습 목표
  - 데이터에 대한 시프트 연산을 계산할 수 있다.

- 내용

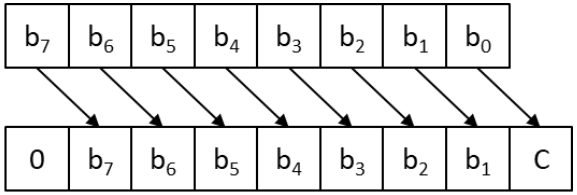
9.4.1 논리 시프트

9.4.2 산술 시프트

9.4.1 논리 시프트



(a) 왼쪽 논리 시프트



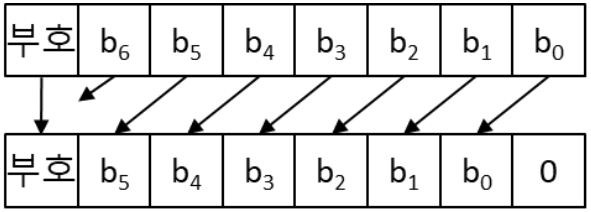
(b) 오른쪽 논리 시프트

- 논리 시프트(logical shift)
  - 왼쪽 논리 시프트: SHL R1 // 한 비트씩 왼쪽으로 자리이동
  - 오른쪽 논리 시프트: SHR R1 // 한 비트씩 오른쪽으로 자리이동
  - 새로 채워지는 비트의 값은 0
  - 제거되는 비트는 상태 레지스터의 Carry 플래그에 저장

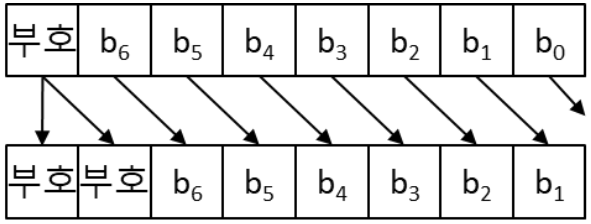
- [예제 9-14] 명령어 실행 결과는?

1) SHL R1 (R1=0010_0110) C=___ R1=_____
2) SHR R2 (R2=1110_0011) C=___ R2=_____

9.4.2 산술 시프트



(a) 왼쪽 산술 시프트



(b) 오른쪽 산술 시프트

- 산술 시프트(arithmetic shift)
  - 왼쪽 산술 시프트: ASL R1 // R1 ← R1×2. 왼쪽에 0을 채움
  - 오른쪽 산술 시프트: ASR R1 // R1 ← R1÷2. 부호 비트를 채움
  - 부호 비트는 고정
- [예제 9-15] 명령어 실행 결과는? 곱하기2, 나누기2 효과가 있는가?

1) ASL R1 (R1=0010_0110$_2$=38$_{10}$) R1=_____(2진수)=___(10진수)
2) ASR R2 (R2=1110_0010$_2$=-30$_{10}$) R2=_____(2진수)=___(10진수)

9.5 산술 연산

- 산술 연산
  - 양수, 음수를 구분하지 않는다.
  - 2의 보수로 해석한다.
- 학습 목표
  - 2진수에 대한 덧셈과 뺄셈을 계산할 수 있다.
  - 오버플로우가 발생하는 이유를 설명할 수 있다.

- 내용
  - 9.5.1 단항 연산
  - 9.5.2 덧셈과 뺄셈

9.5.1 단항 연산

- 단항 연산
  - NEG R // R ← -R (2의 보수)
  - INC R // R ← R+1 (증가)
  - DEC R // R ← R-1 (감소)

- [예제 9-16] 명령어 실행 결과는?

1) R1=0010_1100, NEG R1 R1 = _____
2) R2=0011_1111, INC R2 R2 = _____
3) R3=0000_0000, DEC R3 R3 = _____

9.5.2 덧셈과 뺄셈

- 덧셈: ADD R1, R2 // R1 ← R1+R2
- 뺄셈: SUB R1, R2 // R1 ← R1-R2 = R1 + (R2에 대한 2의 보수)

- [예제 9-17] 덧셈 결과 해석

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

(a) 2진수 덧셈 (b) 부호 없는 수 해석 (c) 정수 해석

- [예제 9-18] 뺄셈 계산

```
  0100_1110        0100_1110        78
- 0001_1000    + 1110_1000      - 24
  0011_0110      10011_0110       54
```

(a) 2진수 뺄셈 (b) 2의 보수 덧셈 (c) 정수 해석

오버플로우

- 오버플로우(Overflow)
  - 레지스터 크기 제한
  - 연산 결과가 수의 표현 범위를 초과하는 현상
  - Overflow = Cn ⊕ Cn-1
- [예제 9-19, 20] 오버플로우가 발생하는지 예측하라. 플래그 값은?

(1) 89+45 (2) 89–45 (3) –89+45 (4) –89–45

89+45=132>127

C_8C_7

01111 001
0101_1001
+0010_1101
1000_0110=-122

Sign flag= ___
Zero flag= ___
Carry flag= ___
Parity flag= ___
Overflow flag= ___

89-45=44

C_8C_7

11010 011
0101_1001
+1101_0011
0010_1100=44

Sign flag = ___
Zero flag = ___
Carry flag = ___
Parity flag = ___
Overflow flag = ___

-89+45 = -44

C_8C_7

00101 111
1010_0111
+0010_1101
1101_0100=-44

Sign flag = ___
Zero flag = ___
Carry flag = ___
Parity flag = ___
Overflow flag = ___

-89-45=-134<-128

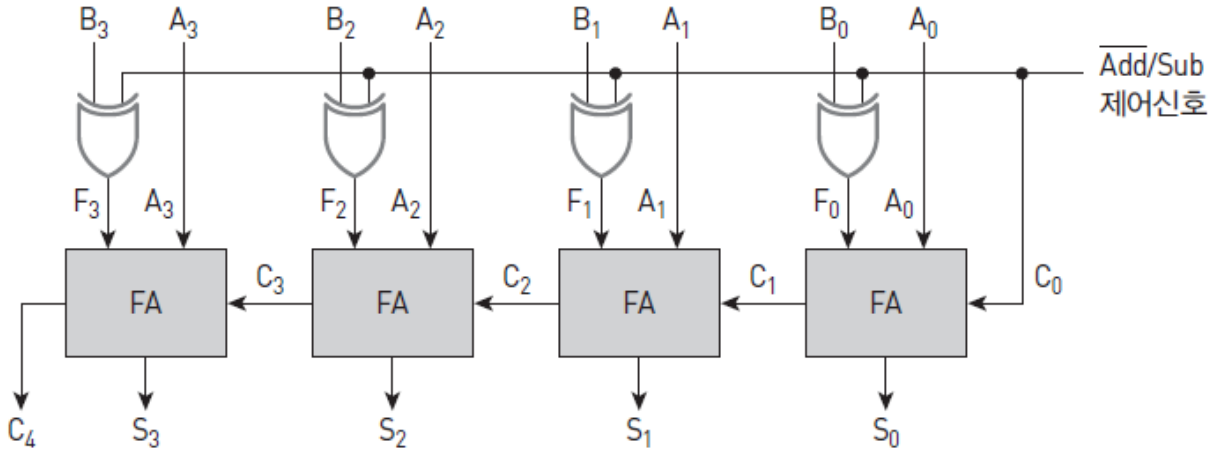
C_8C_7

10000 111
1010_0111
+1101_0011
0111_1010=122

Sign flag = ___
Zero flag = ___
Carry flag = ___
Parity flag = ___
Overflow flag = ___

병렬 가산기

- Add/Sub = 0일 때, 덧셈: $C_4\_S_3S_2S_1S_0 = A+B$
- Add/Sub = 1일 때, 뺄셈: $C_4S_3S_2S_1S_0 = A + /B + 1 = A - B$



[그림 9-12] 병렬 가산기

9.5 산술 연산 요약

- 부호를 고려하지 않고 연산
- 단항 산술 연산
  - 2의 보수(음수 만들기), 증가, 감소
- 덧셈과 뺄셈
  - 덧셈 = A + B
  - 뺄셈 = A + /B + 1
- 오버플로우
  - 계산 결과가 수의 표현 범위를 벗어나는 현상
  - MSB와 그 아래 단 자리오림수의 XOR 연산으로 검출

9.6 실수

- 학습 목표
  - 실수를 표현하는 원리를 이해한다.
  - 표준 규격에 따라 실수 값을 해석할 수 있다.

- 내용

9.6.1 부동소수점 표현

9.6.2 IEEE754 형식

9.6.1 부동 소수점 표현

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

- 10진수 과학 표기 예
  - $976,000,000,000 = 9.76 \times 10^{11}$ 부호: + 가수: 9.76 지수: 11
  - $-0.00000000000976 = -9.76 \times 10^{-12}$ 부호: – 가수: 9.76 지수: –12

- 2진수 예
  - $0.1101 \times 2^2 = 11.010 \times 2^0 = 110.10 \times 2^{-1}$
  - 정규화(normalize) 필요

가수 정규화(normalized mantissa)

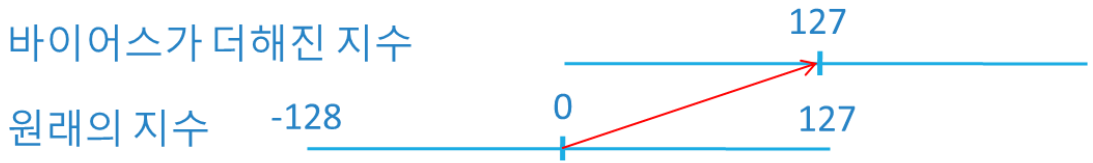
- 가수 정규화
  - (1.bbb…b x $2^{\pm E}$) 형식이 되도록 지수 조정
  - 1.은 표현에서 생략 (항상 1.0은 있는 것으로 취급)
  - 가수를 표현하는 비트 영역 최대 활용
  - 숫자 0은 모든 비트를 0으로 채움

- [예제 9-21] 2진수 101.1010_1111에 대한 가수 표현은?
  (단, 가수를 8비트로 표현)

  - 101.1010_1111 = 1.0110_1011_11 x 2^{+2}
  - 1.은 생략
  - 가수: 0110_1011만 표현
  - 하위 두 비트는 없어짐

바이어스 지수(biased exponent)

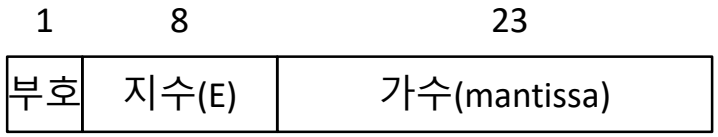
- 바이어스 지수
  - 표현 = 지수 + bias // 지수를 양수로 변환
  - 실제 지수 = 표현 – bias
  - 예) 지수 영역이 8비트일 때, bias = 127_{10} = 0111_1111



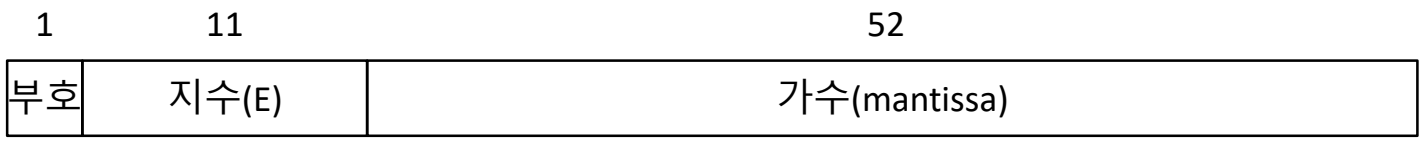
  - 이유? 덧셈 뺄셈을 할 때, 양수 영역에서 지수를 비교하기 위하여

- [예제 9-22] 101.101011에 대한 지수 표현 값은?
  (단, 지수에 8비트 할당, bias = 0111_1111)
  - 가수 정규화: 101.1010_11 = 1.0110_1011 x 2^2
  - 지수의 값 = 2_{10} = 0000_0010
  - 지수 표현 = 0000_0010 + 0111_1111 = 1000_0001.

9.6.2 IEEE 754 형식



(a) **단정도 형식** (single precision)



(b) **배정도 형식** (double precision)

단정도 형식			배정도 형식		
지수	가수	해석	지수	가수	해석
255	≠ 0	NaN	2047	≠ 0	NaN
255	0	$(-1)^S\infty$	2047	0	$(-1)^S\infty$
0<e<255	-	$(-1)^s 2^{e-127}(1.f)$	0<e<2047	-	$(-1)^s 2^{e-1023}(1.f)$
0	≠ 0	$(-1)^s 2^{e-126}(0.f)$	0	≠ 0	$(-1)^s 2^{e-1022}(0.f)$
0	0	$(-1)^S 0$	0	0	$(-1)^S 0$

IEEE 754 예

- [예제 9-23] 단정도 형식으로 표현된 실수의 값은?
  - 2진수 패턴: 1100_0001_0101_0000_0000_0000_0000_0000
  - 단정도 패턴: 1_10000010_10100000000000000000000
  - 해석: $(-1)^s 2^{e-127}(1.f)$
    부호: 1
    지수: e = 1000_0010 = 130_{10}
    가수: m = 1.101
  - 값: $N = (-1)^1 \times 1.101 \times 2^{130-127}$
    $= -1.101 \times 2^3$
    $= -1101$
    $= -13_{10}$.

IEEE 754 예

- [예제 9-24] $(12.5)_{10}$에 대한 단정도 실수 표현은?

  - 2진수 변환 — $12.5_{10} = (1100.1)_2$
  - 부호 — 0 (양수)
  - 가수 정규화 — $1100.1 = 1.1001 \times 2^3$
  - 가수 표현 (23비트) — 100_1000_0000_0000_0000_0000
  - 지수 정규화 — $3+127 = 130_{10}$
  - 지수 표현 (8비트) — 1000_0010

  - 2진수 패턴 = 부호_지수_가수
    = 0_1000_0010_100_1000_0000_0000_0000_0000
    = 0100_0001_0100_1000_0000_0000_0000_0000
    = $(4148\_0000)_{16}$

9.6 실수 표현 요약

- 과학 표기법
  - 부호: 수 전체의 부호
  - 가수 정규화: $(1.bbb...b \times 2^{\pm E})$. 1.을 생략하고 소수점 이하 부분만 표시
  - 지수: 바이어스를 더하여 표시
- IEEE754 표준
  - 단정도 형식 (32비트)
  - 배정도 형식 (64비트)

9.7 요약

9.1 연산기 개요
- 연산기 구조, 단항 연산자, 이항 연산자

9.2 정수
- 정수 표현 방법: 부호화 크기, 2의 보수(표준)

9.3 논리 연산
- NOT, AND(mask), OR(selective set), XOR(selective complement)

9.4 시프트 연산
- 논리 시프트, 산술 시프트

9.5 정수 산술 연산
- 단항 연산: 음수 만들기, 증가, 감소
- 덧셈과 뺄셈: 2의 보수에 의한 연산, 오버플로우

9.6 실수
- 부동소수점 표현: 정규화 가수, 바이어스 지수
- IEEE 754 형식