Handling Negative Numbers

소프트웨어 꼰대 강의

노기섭 교수

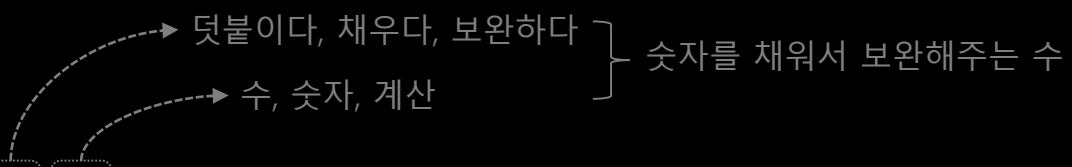
(kafa46@gmail.com)

Course Overview

Topic	Contents
01. Orientation 오리엔테이션	Course introduction, motivations, final objectives 과정 소개, 동기부여, 최종 목표
02. Converting floating point 실수 변환	How to convert float from decimal to binary 어떻게 십진수를 이진수로 변환하는가?
03. Fixed-point Representation 고정 소수점 방식	How to represent float in fixed representation 어떻게 고정 소수점 방식으로 실수를 표현하는가?
04. Floating-point Representation 부동 소수점 방식	How to represent float in floating representation 어떻게 부동 소수점 방식으로 실수를 표현하는가?
05. Handling Negative Numbers 음수 처리 (보너스 강의)	Complement, Radix, n-ary System, etc. 보수, 기수, 진법 등

컴퓨터가 음수를 표현하는 방법
(Understanding Complement System)

직관적으로 이해하는 보수 (Complement)



보수(補數, Complement)는 어떤 기준값이 있을 때,

그 기준에서 특정 수를 얼마만큼 더해야 되는지를 나타내는 값

즉, "합이 기준이 되는 값이 되도록 하는 수"

기준값: $10^{\#digits}$

$Complement\ of\ 6$	기준값: 10^1	$6+4$	$=10$	$Complement$: 4
$Complement\ of\ 57$	기준값: 10^2	$57+43$	$=100$	$Complement$: 43
$Complement\ of\ 723$	기준값: 10^3	$723+297$	$=1{,}000$	$Complement$: 297

How computers understand negative numbers?

What about negative float?

보수 (complement)를 이용해 처리할 수 있습니다!

- Complement는 음수를 표현하는 방식
- Complement는 모든 연산을 덧셈으로 처리하는 훌륭한 방법 ^^

보수를 이해하려면 기수 (radix) 라는 것을 먼저 알아야 해요

- 수(number)를 표현할 때 필요한 기본 숫자들의 개수, 일반적으로 r 로 표기
- 진법의 밑 (base)를 의미함 ➔ 한 자리에 표현할 수 있는 숫자의 개수

진법 이름	사용 가능한 숫자	기수 (radix)
이진법 (binary)	0, 1	2
팔진법 (octal)	0 ~ 7	8
십진법 (decimal)	0 ~ 9	10
16진법 (hex)	0 ~ 9, A ~ F	16

각 자리의 값은 r 의 거듭제곱으로 가중

Base-n numeral system & Complement

n 진법?

Base-n numeral system (일반적 표현)

Radix-n system (더 엄밀한 수학적 표현)

n-ary number system (컴공, 이론 수학 등)

다 같은 말...

Complement in radix - r system?

Notation	Definition
$(r-1)$'s complement	Subtract each digit from $r-1$
r's complement	$(r-1)$'s complement plus 1

Examples in Complement in radix - r system

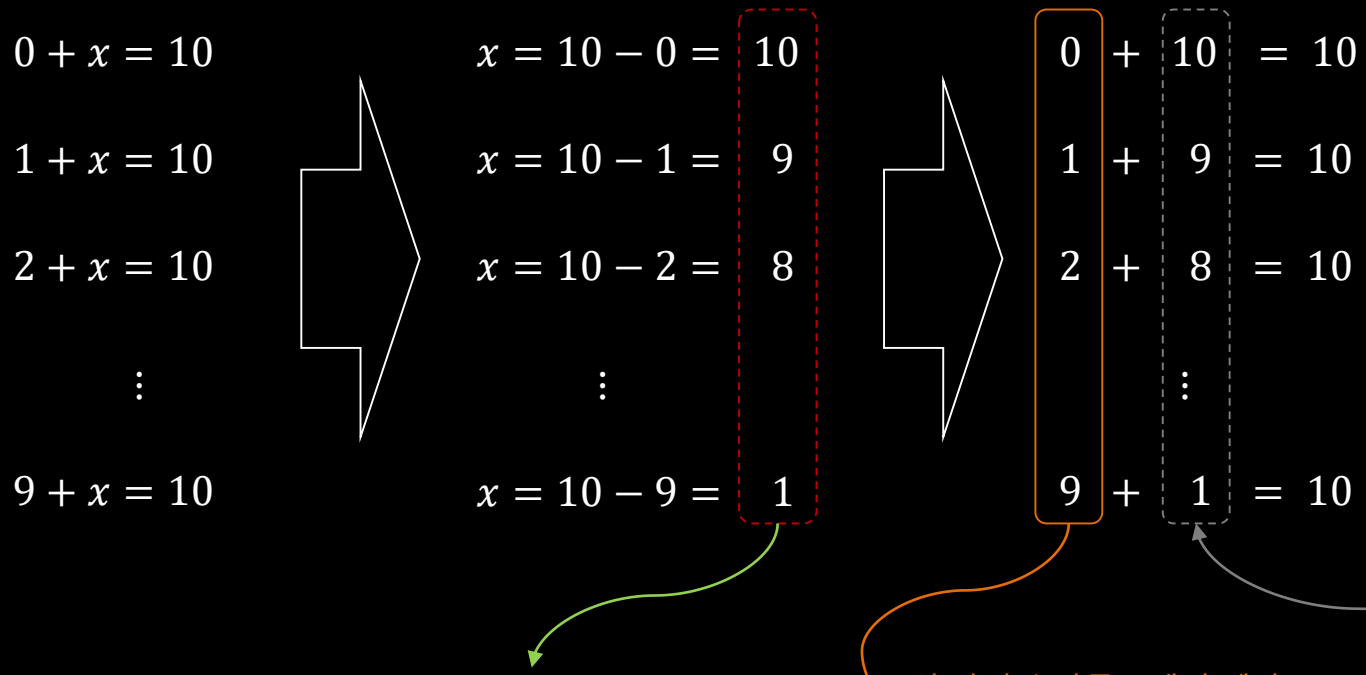
Complement in radix - r system?

Notation	Definition
$(r-1)$'s complement	Subtract each digit from $r-1$
r's complement	$(r-1)$'s complement plus 1

진법 이름	문자 (Alphabet)	기수 (radix)	보수 (complement)
이진법 (binary)	0, 1	2	1의 보수, 2의 보수
팔진법 (octal)	0 ~ 7	8	7의 보수, 8의 보수
십진법 (decimal)	0 ~ 9	10	9의 보수, 10의 보수
16진법 (hex)	0 ~ 9, A ~ F	16	F의 보수, 16의 보수

One step more in complement

10진법에서 어떤 숫자를 조작해서 radix-10을 만드는 방법??



Subtraction Without Using Negative Numbers

음수 없이도 뺄셈할 수 있어요~~

도대체 어떻게 가능함?

!!!!!!!!!!!!!!

만약 자리 올림(overflow) 있으면? → 버린다.

$5 - 3 = 2$ ⇨ $5 + 7 = \square\,2$ ⇨ $5 - 7 = 2$

9의 보수를 구한다

각 자리 숫자를 9에서 뺀다.
(결과는 9 – 3 = 6)

10의 보수를 구한다

9의 보수에 1을 더한다.
(결과는 6 + 1 = 7)

음수 대신에 3에 대한
10의 보수를 더해준다.

보수를 이용한 덧셈 결과를
뺄셈 결과로 처리한다.

계산은 컴퓨터가 합니다.

너무 긴장할 필요 없어요 ^^.

원리만 이해하면 충분합니다!

One More Example in radix - 10 system

$4\ 3\ 2\ 1 - 1\ 2\ 3\ 4 = 3,0\ 8\ 7$

보수를 이용한 덧셈 결과를
뺄셈 결과로 처리한다.

9 $(r - 1 = 10 - 1 = 9)$의 보수를 구한다

각 자리 숫자를 9에서 뺀다.

사실은 기준값에서
빼주는 것임
$10^{\#digits} - 1$
$= 10^4 - 1$
$= 9999$

$$\begin{array}{r} 9\ 9\ 9\ 9 \\ -\ 1\ 2\ 3\ 4 \\ \hline 8\ 7\ 6\ 5 \end{array}$$

$4\ 3\ 2\ 1 + 8\ 7\ 6\ 6 = \square\ 3,0\ 8\ 7$

음수 대신에 3에 대한
10의 보수를 더해준다.

만약 자리 올림(overflow) 있으면?
→ 버린다.

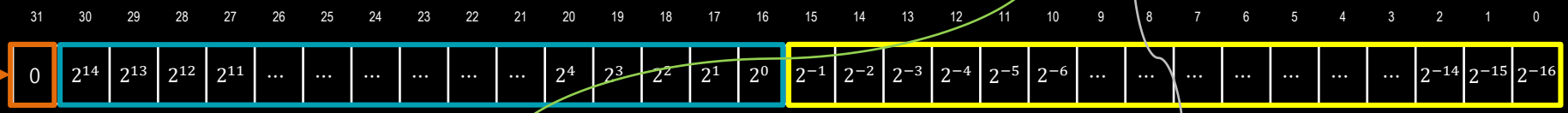
10$(r = 10)$의 보수를 구한다
9의 보수에 1을 더한다.
(결과는 $8765 + 1 = 8766$)

이제는 어떤 뺄셈도
보수를 이용하면
음수 없이 더하기로
처리할 수 있다!

Move onto radix - 2 system

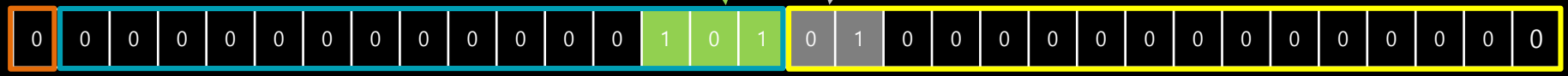
0: Positive
1: Negative

도전: 고정 소수점 방식에서 음수 표현: -5.25



정수부: $5 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

소수부: $0.25 = 0 \cdot 2^{-1} + 1 \cdot 2^{-2}$

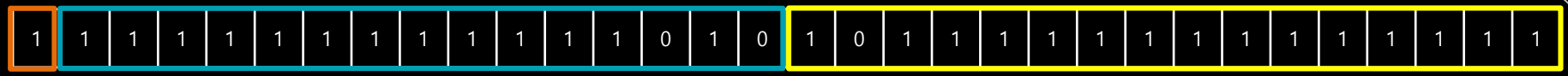


$1\ (r - 1 = 2 - 1)$의 보수를 구한다

기준값 $2^{15} - 1$에서 뺀다 (각 자리 숫자를 1에서 뺀다.)

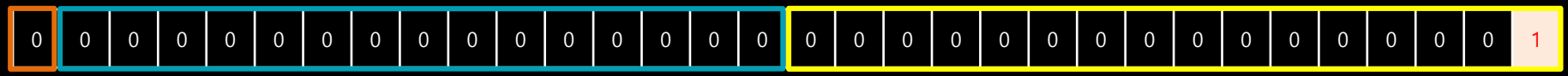
부호 비트를 Flip하는 것과 같은 효과

(1 → 0으로, 0 → 1 으로 변경)

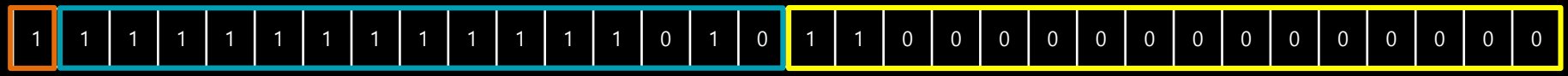


2의 보수를 구한다 (1의 보수에 1을 더한다.)

만약 자리 올림(overflow) 있으면?
→ 버린다.



최종 결과



Minimum negative number in 32-bit Fixed-point System

0: Positive
1: Negative

소수점을 고려하지 않은 최솟값 구해보기

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	2^{14}	2^{13}	2^{12}	2^{11}	…	…	…	…	…	…	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	…	…	…	…	…	…	…	…	2^{-14}	2^{-15}	2^{-16}

음의 최댓값 $= -1 \times Max\ Positive\ Integer = -1 \times 32{,}767$

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1의 보수를 구한다

각 자리 숫자를 1에서 뺀다.

모든 비트를 Flip하는 것과 같은 효과

(1 → 0으로, 0 → 1 으로 변경)

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

2의 보수를 구한다 (1의 보수에 1을 더한다.)

만약 자리 올림(overflow) 있으면?
→ 버린다.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

최종 결과

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$-1 \times Max\ Positive\ Integer = -1 \times 32{,}767$

Minimum negative number in 32-bit Fixed-point System

0: Positive
1: Negative

소수점까지 고려한 최솟값 구해보기

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	2^{14}	2^{13}	2^{12}	2^{11}	…	…	…	…	…	…	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	…	…	…	…	…	…	…	…	2^{-14}	2^{-15}	2^{-16}

음의 최댓값 $= -1 \times Max\ Positive\ Number = -1 \times 32{,}767.999985$

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

1의 보수를 구한다

각 자리 숫자를 1에서 뺀다.

모든 비트를 Flip하는 것과 같은 효과

(1 → 0으로, 0 → 1 으로 변경)

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2의 보수를 구한다 (1의 보수에 1을 더한다.)

만약 자리 올림(overflow) 있으면?
→ 버린다.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

최종 결과

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

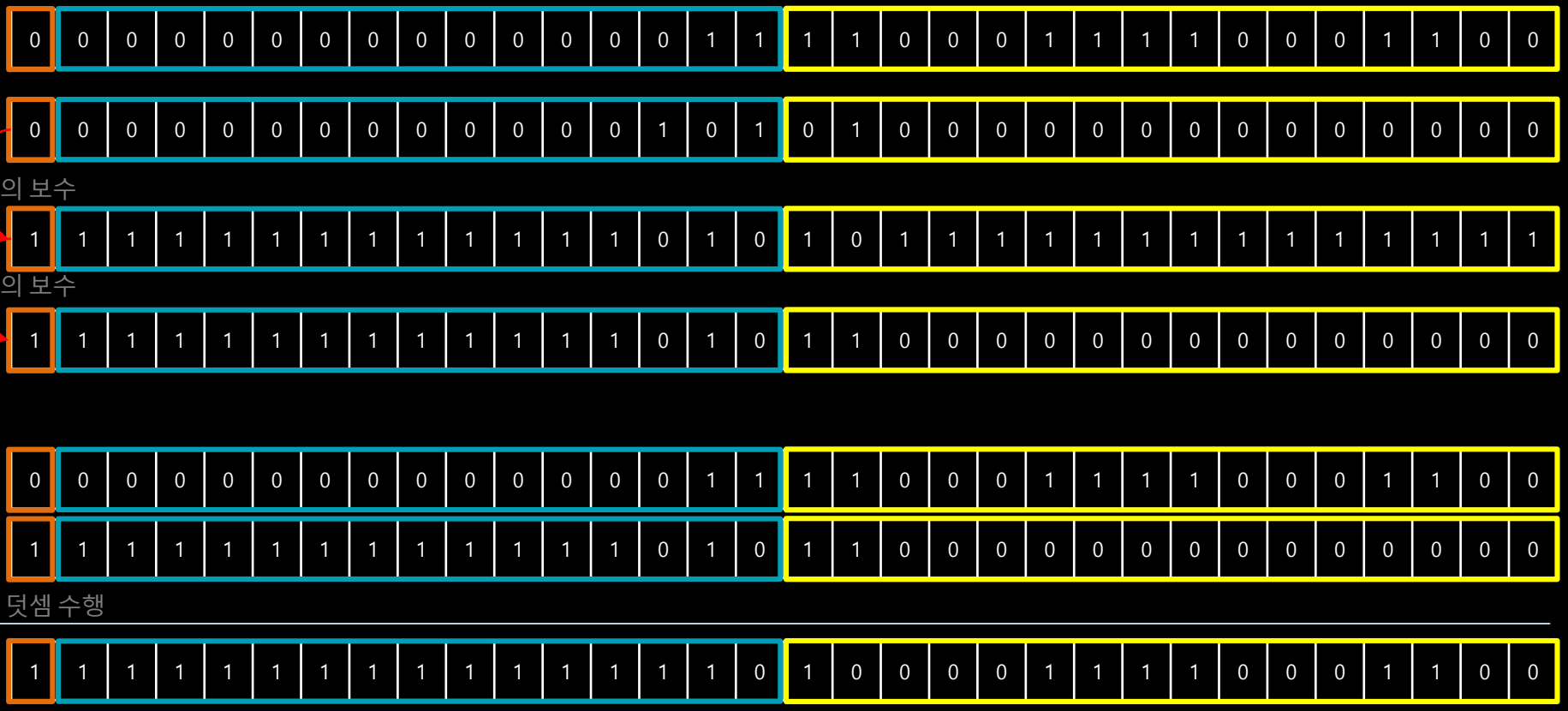
Maximum negative number in 32-bit Fixed-point System

두 실수를 보수를 이용해 처리하기

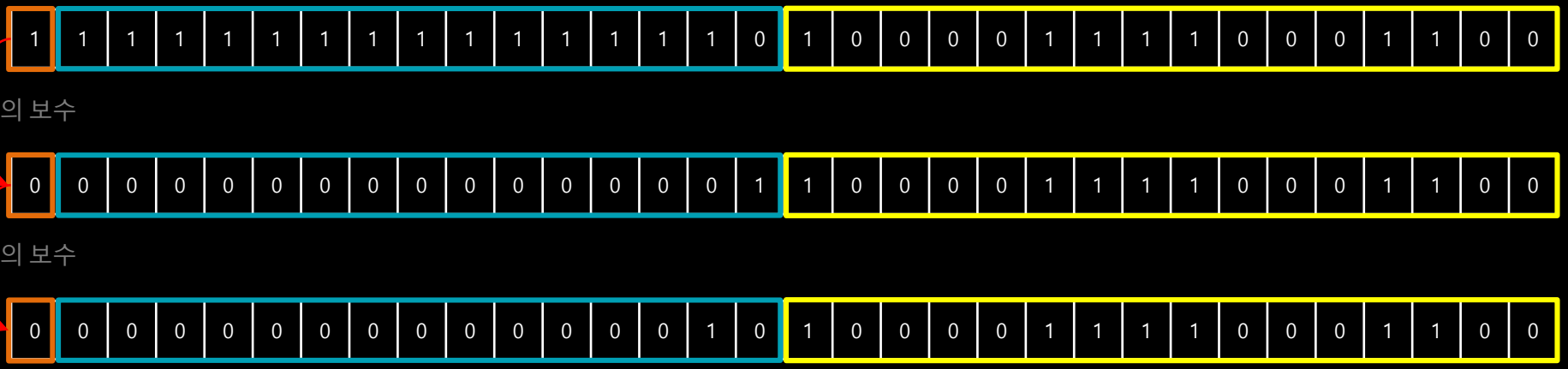
$3.78 - 5.25 = 3.78 + (-5.25)$

11 0.1100011111000000… 101 0.01

무한소수



2의 보수인 상태 → 정수부의 10진수 값을 확인하기 위해 다시 보수를 취하여 십진수 확인



정수부: −2

소수부 ≈ 0.5
+ 0.0078125
+ 0.00390625
+ 0.001953125
+ 0.0009765625
+ 0.00048828125
+ 0.000061035156125
+ 0.000030517578125
≈ 0.5283203125 (오차 있음)

-2 + 0.5283203125 = -1.4716796875

3.78 − 5.25 ≈ −1.4716796875 (오차 있음)

실수 연산 오차 ≈ 0.0016796875

Do we need complement in Floating-point System?

항목	고정 소수점 (Fixed-point)	부동 소수점 (Floating-point)
음수 표현 방식	2의 보수 사용	부호 비트(Sign Bit) 사용
보수가 필요한 이유	하드웨어에서 뺄셈 = 덧셈 + 보수 로 구현	절댓값 연산 기반 → 보수 불필요
보수 사용 위치	레지스터 수준에서 직접 사용됨	사용하지 않음
예시	-5 → 11111011 (8비트 2의 보수)	-5.75 → 부호 비트 1, 나머지는 +5.75와 동일
장점	덧셈 회로만으로 뺄셈도 처리 가능	정규화 구조로 넓은 범위 실수 표현 가능
단점	범위 한정, 오버플로우, 연산 유연성 낮음	연산 복잡, 정밀도 오차 가능성 여전히 있음

고정 소수점에서 2의 보수 사용하는 이유

1. 덧셈 회로만으로 뺄셈 가능	음수를 2의 보수로 표현하면, A - B를 A + (-B)로 계산할 수 있어 별도의 뺄셈 회로 없이 **덧셈 회로 하나로 구현** 가능
2. 하드웨어 단순화	가산기(adders)만으로 산술 연산 처리가 가능 **연산 장치 간소화**
3. 효율적인 레지스터 사용	양수와 음수를 같은 방식으로 표현할 수 있어 **레지스터 구조가 단순화 가능**
4. 오버 플로우 판별 쉬움	부호 비트와 캐리 비트를 통해 **오버 플로우 검출 편리**

적용 사례:

- 스마트 워치 센서 계산,
- 딥러닝 부동 소수점 연산,
- 오디오 코덱,
- 게임 엔진 등

수고하셨습니다 ..^^..