

Floating-point Representation in Computer

소프트웨어 끝대 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	Complement, Radix, n-ary System, etc. 보수, 기수, 진법 등

Recap: Fixed-point Representation

Pros	Cons
Simple Implementation	Limited Range on Float Numbers
Consistence Precision	Rigid Bit Allocation
Predictable Memory Usage	Poor Flexibility in Precision vs. Range
High Speed	Unused Capacity (Memory)

Floating Point Representation

Floating
(둥둥 떠 다니는)

Point
(수소점)

In Korean,

부동(浮動) 소수점

→ 물에 뜨다, 떠다니다.
→ 움직이다, 옮기다.

또는,

떠돌이 소수점



Core Idea,

소수점 위치를 고정하지 말고, 움직일 수 있도록 하자!

표준: IEEE 754 (https://ko.wikipedia.org/wiki/IEEE_754)

Overview of Floating Point

Floating Point

소수점의 위치를 고정하지 않고

그 위치를 나타내는 수를 따로 표현(적는 것)

유효숫자를 나타내는 가수(Fraction/Mantissa, 假數)와

소수점의 위치를 풀이하는 지수(Exponent, 指數)로 나누어 표현



이렇게 하면

아주 큰 값을

표현할 수 있어요 ^^

고정 소수점 방식: $\boxed{1\ 2} . \boxed{3\ 4\ 5}$

정수 소수

부동 소수점 방식: $\boxed{1\ .\ 3\ 4\ 5} \times \boxed{10^2}$

가수 (mantissa) 지수 (exponent)

정규화(Normalization)
소수점 위치를 통일하는 과정

소수점 위치를 지수로 표현해서
"떠다니는 소수점" 구현

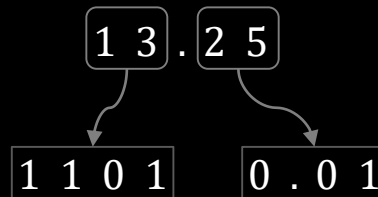
Example of Normalization

Normalization in Floating Point Representation

부동 소수점에서 가수(Mantissa)의
소수점 위치를 조정하여 항상 1.xxx 형태가 되도록 만드는 과정

[10진수 예시] 4 5 0 0 . 7 8 \longrightarrow 4 . 5 0 0 7 8 $\times 10^3$

[2진수 예시]



왜 이 형식이 중요할까?
(다음 슬라이드로...)

1 1 0 1 . 0 1 \longrightarrow 1 . 1 0 1 0 1 $\times 2^3$

왜 이 형식이 중요한가?

무한대에 가까운 수 표현 가능

고정된 비트 수 (예 32비트) 만으로도 큰/작은 수 표현 가능

3.4^{38} 같은 거대한 수 표현 가능

3.4^{-45} 같은 아주 작은 수 표현 가능

정규화 표현으로 일관성 유지 가능

모든 수를 $1.xxx \times 2^n$ 형식으로 통일

연산 결과의 정밀도 예측이 용이 → HW, SW 최적화 구현 가능

국제적 표준화 (IEEE 754) 가능

모든 컴퓨터에서 같은 방식으로 해석 가능

Cross Platform에서도 결과의 일관성 확보

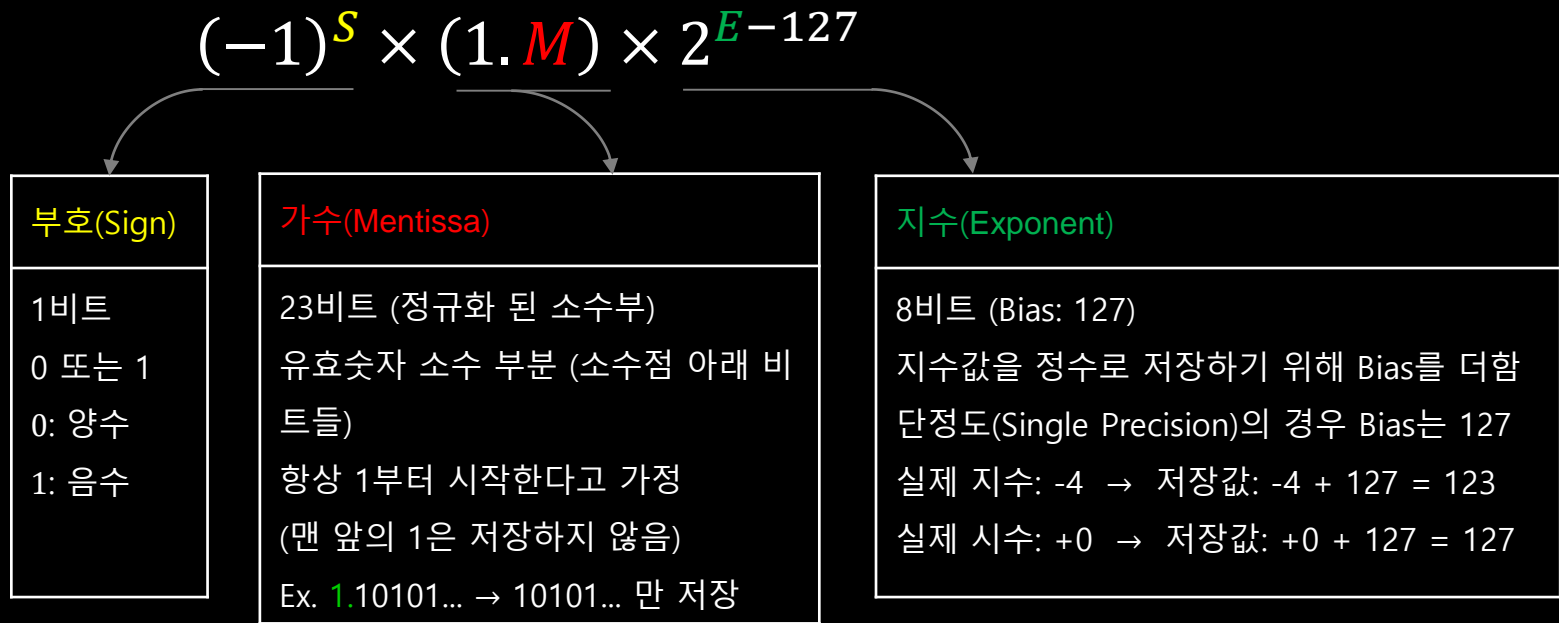
Structure of IEEE 754

IEEE 754란?

실수를 이진수로 저장하는 국제 표준.

컴퓨터마다 실수 계산이 다르면 큰일 나니까, 전 세계적으로 약속된 통일된 규칙

IEEE 기본 구조



Example of IEEE 754 (Positive Number)

10진수 5.75를 IEEE 754로 표현

Step 1. 10진수를 2진수로 변환

$$5 = 101$$

$$0.75 = 0.11$$

$$5.75 = 101.11 \longrightarrow 1.0111 \times 2^2$$

Step 3. 최종 비트 표현

0 10000001 01110000000000000000 000

$$(-1)^S \times (1.M) \times 2^{E-127}$$

$$= (-1)^0 \times (1.0111) \times 2^{129-127}$$

$$= 1.0111 \times 2^2$$

Step 2. 부호, 가수, 지수 값 만들기

부호(Sign)	가수(Mantissa)	지수(Exponent)
1비트 0: 양수 1: 음수	23비트 (정규화된 소수부) 유효숫자 소수 부분 (맨 앞의 1은 저장하지 않음) $1.0111 \rightarrow M = 0111$ (남은 비트는 0으로 채움)	8비트 (Bias: 127) 지수값을 정수로 저장하기 위해 Bias를 더함 단정도(Single Precision)의 경우 Bias는 127
0: 양수 0	01110000000000000000000	실제 지수: +2 $\rightarrow E = 2 + 127 = 129$ 10000001

Example of IEEE 754 (Negative Number)

10진수 -5.75를 IEEE 754로 표현

음수는 부호 비트만 1로 설정하고
나머지(지수, 가수)는 양수와 동일

Step 1. 10진수를 2진수로 변환

$$5 = 101$$

$$0.75 = 0.11$$

$$5.75 = -101.11 \longrightarrow -1.0111 \times 2^2$$

Step 3. 최종 비트 표현

1 10000001 011100000000000000000000

$$(-1)^S \times (1.M) \times 2^{E-127}$$

$$= (-1)^1 \times (1.0111) \times 2^{129-127}$$

$$= -1.0111 \times 2^2$$

Step 2. 부호, 가수, 지수 값 만들기

부호(Sign)	가수(Mantissa)	지수(Exponent)
1비트 0: 양수 1: 음수	23비트 (정규화된 소수부) 유효숫자 소수 부분 (맨 앞의 1은 저장하지 않음) $1.0111 \rightarrow M = 0111$ (남은 비트는 0으로 채움)	8비트 (Bias: 127) 지수값을 정수로 저장하기 위해 Bias를 더함 단정도(Single Precision)의 경우 Bias는 127
1: 음수 1	011100000000000000000000	실제 지수: +2 $\rightarrow E = 2 + 127 = 129$ 10000001

무한 소수의 근사 처리

10진수 5.7를 IEEE 754로 표현

Step 1. 10진수를 2진수로 변환

$$5 = 101$$

$$0.7 = 0.101100110 \dots_2 \text{ (무한반복)}$$

$$5.75 = 101.101100110 \dots_2$$

정규화

$$\rightarrow 1.01101100110 \dots \times 2^2$$

Step 3. 최종 비트 표현

$$0 \text{ } 10000001 \text{ } 01101100110011001100110$$

$$(-1)^S \times (1.M) \times 2^{E-127}$$

$$= (-1)^0 \times (1.0111 \dots) \times 2^{129-127}$$

$$\approx 5.699999809265137 \text{ (오차 존재!)}$$

Step 2. 부호, 가수, 지수 값 만들기

부호(Sign)	가수(Mantissa)	지수(Exponent)
1비트 0: 양수 1: 음수	23비트 (정규화된 소수부) 유효숫자 소수 부분 (맨 앞의 1은 저장하지 않음) $1.01101100110 \dots \rightarrow$	8비트 (Bias: 127) 지수값을 정수로 저장하기 위해 Bias를 더함 단정도(Single Precision)의 경우 Bias는 127
0: 양수 0	M = 23개만 취하고 나머지는 버림 01101100110011001100110	실제 지수: +2 $\rightarrow E = 2 + 127 = 129$ 10000001

Need more precise number... Double Precision!!!



항목	단정도 (Single, 32bit)	배정도 (Double, 64bit)
총 비트 수	32비트	64비트
부호 비트	1비트	1비트
지수 비트 (E)	8비트 (Bias = 127)	11비트 (Bias = 1023)
가수 비트 (M)	23비트	52비트
유효숫자 정밀도	약 7자리	약 15~17자리
표현 가능한 범위	$\pm 10^{\pm 38}$	$\pm 10^{\pm 308}$
사용 예시	그래픽, 센서 데이터 등	과학 계산, 통계, 머신러닝, 금융 등
π	저장 값 ≈ 3.1415927	저장 값 $\approx 3.141592653589793$

Comparison of Floating-Point Types

- 대부분의 현대 프로그래밍 언어는 64비트 배정도(double)를 기본 실수 타입으로 채택
- 단정도(float)는 명시적으로 사용할 때만 적용

언어	단정도 (32bit float)	배정도 (64bit double)	비고
C / C++	float	double	명시적으로 구분
Java	float (32bit, float f = 3.14f)	double (64bit, 기본 실수 타입)	double이 기본
Python	지원 안 함 (기본 double 사용)	모든 실수는 float (double precision를 가짐)	float는 64비트
JavaScript	없음 (모든 숫자 = Number)	Number는 내부적으로 IEEE 754 double	항상 64bit
R	없음	모든 숫자는 double	단정도 없음
MATLAB	없음	모든 숫자는 double	과학 계산용, 고정 double 사용

Pros & Cons in Floating-point Representation

구분	장점 	단점 
표현 범위	매우 큰 수와 작은 수까지 넓은 실수 표현 가능 ($\pm 10^{38}$ 수준)	지수 정렬 및 정규화 과정이 복잡하여 하드웨어 자원 소모가 큼
정밀도	정규화와 가변 소수부 사용으로 다양한 실수 근사 가능	무한소수 및 근사 표현에 따른 오차 발생 가능성 있음
연산 처리	과학/공학용 계산에 적합 (지수, 로그, 삼각함수 등)	고정 소수점보다 연산 속도 느림, 복잡한 연산 필요
표준화	IEEE 754 등 표준 존재 → 다양한 언어/하드웨어 간 호환성 우수	연산 순서에 따라 결과 달라질 수 있음 $(a + b) + c \neq a + (b + c)$
활용 사례	일반적 컴퓨터 연산, 머신러닝, 그래픽, 시뮬레이션 등 다양한 기반 분야에 활용	저전력 장치(MCU 등)에는 부적합, FPU (Floating Point Unit) 없는 환경에서 사용 어려움

- 고정 소수점: 연산 속도와 전력 효율, 예측 가능성이 중요한 상황에서 여전히 널리 사용
(임베디드 시스템, 디지털 신호처리, 배터리 기반 시스템, AI 모델 경량화 등)
- 부동 소수점: 정밀도와 표현 범위가 필요한 일반 컴퓨터나 과학 연산에 적합



수고하셨습니다 ..^^..