



Chapter03_2.Arithmetic for Computers

Floating Point

- Representing non-integer numbers
 - $\pi = 3.141592\dots$
 - $e = 2.71828\dots$
 - $0.000000001 = 1.0 \times 10^{-9}$
 - $3155760000 = 3.15576 \times 10^9$
- Scientific notation
 - Single digit to the left of the decimal point
 - Normalized number: 1.0×10^{-9}
 - No leading zero
 - 0.1×10^{-8} , 10.0×10^{-10} are not normalized
 - **Binary number in scientific notation: $1.0(2) \times 2^{-1}$**
- Floating point numbers

- Numbers in which binary point is not fixed(이진 소수점이 고정되지 않은 숫자)
 - No fixed number of digits before and after the point(포인트 전후의 고정된 자릿수 없음)

Floating Point Representation

- In binary
 - $\pm 1.xxxxxxx_2 \times 2^{yyyy}$
 - Fraction and exponent
- Fracton and exponent must fit into a word : 분과 지수 부분은 반드시 word 단위에 맞춰야 한다.
 - How many bits for fraction and exponent? (분수와 지수는 몇 비트일까?)
- FP number representation

31	30 - 23	23 - 0
s	exponent	fraction (mantissa)

8 bits 23 bits

- Range in decimal: $2.0 \times 10^{-38} \sim 2.0 \times 10^{38}$
- Overflow in FP arithmetic
 - 지수가 지수 필드에 비해 너무 큽니다.
- Underflow in FP arithmetic
 - 음수 지수가 너무 커서 지수 필드에 맞지 않은 경우
- Single precision : 32bits
- Double precision : 64bits

- 11 bits for exponent, 52 bits for fraction
 - Range (in decimal): $2.0 \times 10^{-308} \sim 2.0 \times 10^{308}$
 - benefit: Increased precision from larger fraction bits

- Double precision은 총 64비트다. LSB의 맨 앞 1비트는 부호를 나타내는 부분이다.
 - 11비트는 지수를 표현하는 공간이다. 남은 52비트는 분수를 나타내기 위함이다.
 - 장점 : 분수 비트의 공간이 더 많이 할당됐으므로 정밀도가 향상됐다.
 - IEEE 754 floating point standard
 - 지수 0000 0000, 분수 0 → 0
 - E : 1111 1111, F : 0 → 무한대
 - E : 1~254, F : anything → normal FP number
 - Consideration for sorting
 - MSB 는 부호 비트를 필요로 한다.
 - 지수는 분수 부분 이전에 오게 된다.

- Biased notation

- $\bullet \quad X = 1.0 \times 2^{-1}$

- $$\bullet \quad Y = 1.0 \times 2^{+1}$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	...

- d위와 같은 경우를 살펴보면, Y 가 X 보다 커야 하는데 오히려 그 반대인 것이 되어버렸다. 이럴 때 필요한 게 바로 Sorting 이다.
 - Rearranging the exponent value range

- 00000000 ~ 11111111
most negative ~ most positive

- 그래서 우리가 흔히 0은 양수 비트, 1은 음수 비트라고 알고 있지만, 여기선 반대로 1을 양수로 0을 음수로 간주한다.
- IEEE 는 단정밀도에 대해 127의 바이어스를 사용한다.(이중 P의 경우 1023)
- Sorting 방법은 되게 쉽다. 어떤 십진수 수에 127(십진수)를 더한 후, 이진수로 바꿔주면 된다.
- -1: $-1 + 127 = 126 = 0111\ 1110_2$
- 1: $1 + 127 = 128 = 1000\ 0000_2$

Precision Range
 $\pm 1.00000000000000000000000000000000 \times 2^{-126}$
 $\pm 1.111111111111111111111111111111 \times 2^{+127}$

Floating Point Bias

Exponent		Adjusted Exponent	
127	01111111	254	11111110
.	.	.	.
.	.	.	.
.	.	.	.
1	00000001	128	10000000
0	00000000	127	01111111
-1	11111111	126	01111110
.	.	.	.
.	.	.	.
-126	10000010	1	00000001
-127	10000001	0	00000000
-128	10000000	-1	11111111

- IEEE 754 Floating-Point Format

- Encoding of the single precision floating-point numbers
- $E = 0, F = 0 \rightarrow$ Originally 1 but convention is 0
- $E = 255, F = 0 \rightarrow$ infinity
- $E : 1\sim254, F : \text{anything} \rightarrow$ normal FP number
- $E : 255, F : \text{Nonzero} \rightarrow \text{NaN}$
 - NaN 은 잘못된 연산의 결과를 나타내는 기호이다.

Single precision		Double precision		Object represented
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	Nonzero	0	Nonzero	\pm denormalized number
1~254	Anything	1~2046	Anything	\pm floating-point number
255	0	2047	0	\pm infinity
255	Nonzero	2047	Nonzero	NaN (Not a Number)

- Floating Point Example

- Represent -0.75

- $= -3/4_{10}$ or $-3/2^2_{10}$
- $= -11_2/2^2_{10} = -0.11_2$
- $= -0.11_2 \times 2^0$
- $= -1.1_2 \times 2^{-1}$

- $(-1)^s \times (1 + \text{Fraction}) \times 2^{(\text{Exponent}-127)}$

- $(-1)^1 \times (1 + 0.100\dots000) \times 2^{126}$

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

1 bit 8 bits 23 bits

- What number does this represent?

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

$$\begin{aligned}
 (-1)^s \times (1 + \text{Fraction}) \times 2^{(\text{Exponent}-\text{Bias})} &= (-1)^1 \times (1 + 0.25) \times 2^{(129-127)} \\
 &= -1 \times 1.25 \times 2^2 \\
 &= -1.25 \times 4 \\
 &= -5.0
 \end{aligned}$$

Floating Point Addition

- Consider

- $9.999(10) \times 10^1 + 1.610(10) \times 10^{-1}$
 - 추정컨대 오로지 4비트만 저장 할 수 있다.
- Steps
 - Align decimal points - shift smaller exponent number
 - $9.999 \times 10^1 + 0.016 \times 10^1$, 여기서 보면, $10^{-1} \rightarrow 10^1$ 이 됐다. 이건 100을 곱해준건데, 소수점이 왼쪽으로 가면 10의 배수를 곱해준다. 여기선 두 번 옮겨갔으므로, 100을 곱해준다.
 - Add significands
 - $9.999 + 0.016 = 10.015$
 - result = 10.015×10^1
 - Normalize, check over/underflow
 - 1.0015×10^2
 - Round it to 4digits
 - 1.002×10^2 (4비트만 저장 가능해서 반올림 해 준 값이다.)

Binary FP Addition

- FP Addition 예제

- $0.5_{10} + (-0.4375)_{10}$
 - $0.5_{10} = 0.1_2 = 0.1 \times 2^0 = 1.000 \times 2^{-1}$
 - $0.4375_{10} = 7/16 = 7/2^4 = 111 \times 2^{-4} = 1.110 \times 2^{-2}$
- Align
 - $1.000 \times 2^{-1} - 1.110 \times 2^{-2} = 1.000 \times 2^{-1} - 0.110 \times 2^{-1}$
- Add significands
 - $1.000 \times 2^{-1} - 0.111 \times 2^{-1} = 0.001 \times 2^{-1}$
- Normalize, check over/underflow
 - 1.0×2^{-4}
- Round
 - No need

FP Adder Hardware

