Lecture 04
부동소수점 표현과 연산

4190.414A
Multicore Computing
Fall 2017

과학적 표기법

과학적 표기법

- 크기가 아주 작거나 아주 큰 수를 나타내는 방법
- $m \times b^e$
  - m은 실수로서 가수(假數, mantissa) 또는 계수(係數, coefficient)
  - e는 정수로서 지수(指數, exponent)
  - b는 양의 정수로서 기수(基數, base)
- 소수점의 위치는 고정되어 있지 않음

$$\begin{aligned} -0.12345678 &= -0.12345678 \times 10^{0} \\ &= -1.2345678 \times 10^{-1} \\ &= -12.345678 \times 10^{-2} \\ &= -123.45678 \times 10^{-3} \\ &= -1234.5678 \times 10^{-4} \end{aligned}$$

정규화된 과학적 표기법

- Normalized scientific notation
- 하나의 수를 과학적 표기법으로 나타낼 수 있는 방법이 여러 가지가 있음
- 가수 m의 값을 $1 \leq m < b$으로 한정

전자 : 0.000000000000000000000000000000091093822 kg
지구 : 5973600000000000000000000 kg

전자 : $9.1093822 \times 10^{-31}$ kg
지구 : 5.9736×10^{24} kg

부동소수점 표현

부동소수점 표현

- 부동소수점(浮動小數點, floating-point) 표현은 유효숫자들의 위치를 고려하여 소수점의 위치를 상대적으로 정하는 방법
- 전부 8 개의 숫자를 사용하고 소수점 아래 자릿수가 2 개인 십진 고정소수점 표현에서 표현할 수 있는 수의 범위는 0.00 에서 999999.99까지
- 모두 8 개의 숫자를 사용하는 $m_0.m_1m_2m_3m_4m_5\times10^{e_1e_2}$ 형태의 부동소수점 표현에서 표현할 수 있는 수의 범위는 0.00000×10^{00} 에서 9.99999×10^{99}까지

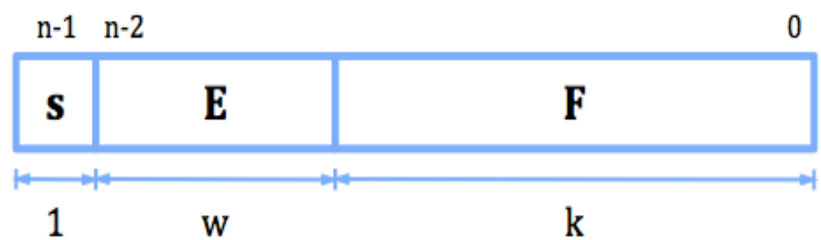
IEEE 754 부동소수점 표현

- 1985년에 IEEE(Institute of Electrical and Electronics Engineers)에서 제정한 부동소수점 표현 및 연산에 대한 표준
- IEEE 754 부동소수점 표준이 나오기 전에는 서로 다른 컴퓨터 제조사들이 서로 호환되지 않는 부동소수점 표현을 사용
- 현재는 거의 모든 제조사가 이 표준을 따름
- 원래의 IEEE 754 표준은 2008년에 IEEE754-2008 표준으로 대체됨
- 종류
  - 세 가지의 이진 부동소수점 기본 형식(32, 64, 128 비트)
  - 두 가지의 십진 부동소수점 기본 형식(64, 128 비트)
  - 확장 정도 형식(extended precision format)의 기준
    - x86 80-비트 확장 정도 표현

n-비트 IEEE 754 이진 부동소수점 표현

- n의 기본적인 값은 32, 64, 128
  - $n = 32$ 일 경우 단정도(單精度, single-precision)
  - $n = 64$ 일 경우 배정도(倍精度, double precision
  - $n = 128$ 일 경우 사배정도(四倍精度, quadruple precision)

경우	종류
$E = 00\cdots0_2$ 이고 $F = 0.00\cdots0_2$	±0
$E = 00\cdots0_2$ 이고 $F \neq 0.00\cdots0_2$	서브노멀 값
$E \neq 00\cdots0_2$ 이고 $E \neq 11\cdots1_2$	정규 값
$E = 11\cdots1_2$ 이고 $F = 0.00\cdots0_2$	$\pm\infty$
$E = 11\cdots1_2$ 이고 $F \neq 0.00\cdots0_2$	NaN



라운딩

- 어떤 실수는 그 값을 IEEE 754 부동소수점 표현으로 정확하게 표현할 수 없음
  - 그 수에 가장 가까우면서 부동소수점 표현으로 표현할 수 있는 값으로 어림잡아 표현
- 라운딩(rounding) : 더 적은 수의 자릿수를 가진 값으로 주어진 수를 어림잡는 작업
- 이를 라운딩 오차(rounding error) : 원래의 값과 라운딩한 값 간에 차이
  - 라운드-오프 오차(round-off error)라고도 부름

라운딩 규칙	13.3	13.5	13.7	14.5
가장 가까운 쪽으로 하되 비기면 짝수로 라운딩 (round to the nearest, ties to even)	13	14	14	14
가장 가까우면서 0에서 먼 수로 라운딩 (round to the nearest, ties away from zero)	13	14	14	15
$+\infty$를 향한 라운딩 (round towards $+\infty$, rounding up, ceiling)	14	14	14	15
$-\infty$를 향한 라운딩 (round towards $-\infty$, rounding down, floor)	13	13	13	14
0을 향한 라운딩 (round towards $-\infty$, truncation)	13	13	13	14

이진수의 라운딩

- 이진수의 경우도 십진수와 같은 규칙을 적용할 수 있음
- 가장 가까운 짝수로 라운딩
  - 표준에 내정된 라운딩 규칙
- 예) 소수점 아래 둘째 자리까지 구하는 경우
  - 100.00011_2
    - 유효숫자가 소수점 아래 둘째 자리까지 있는 수 중에서 100.00011_2에 가장 가까운 수는 100.00_2 하나밖에 없음
  - 100.0011_2
    - 100.01_2
  - 100.111_2
    - 가장 가까운 수가 101.00_2과 100.11_2로 두 개 존재하므로, LSB가 0인 101.00_2이 라운딩의 결과가 됨
  - 100.101_2
    - 100.10_2

0의 표현

- $E = 00 \cdots 0_2$ 이고 $F = 0.00 \cdots 0_2$ 일 경우

0	0000 0000	0000 0000 0000 0000 0000 000
1	0000 0000	0000 0000 0000 0000 0000 000

정규 값

- Normalized value 또는 normal value
- $E \neq 00 \cdots 0_2$ 이고 $E \neq 11 \cdots 1_2$ 일 때
- $v = (-1)^s \times m \times 2^e, \ 1 \leq m < 2$
- 소수부 F의 왼쪽에 암묵적인 1과 그 사이에 소수점이 존재함을 가정

0 1000 1100 1011 1111 1010 0100 0000 000

$$e = E - bias = 10001100_2 - 127_{10} = 140_{10} - 127_{10} = 13_{10}$$

$$m = 1 + f = 1 + 0.10111111101001_2 = 1.10111111101001_2$$

$$v = (-1)^0 \times 1.10111111101001_2 \times 2^{13} = 11011111110100.1_2 = 14324.5_{10}$$

정밀도	n	w	k	bias	e 의 범위	m 의 범위
단정도	32	8	23	127	$-126 \sim +127$	$1.0 \sim (2.0 - 2^{-23})$
배정도	64	11	52	1023	$-1022 \sim +1023$	$1.0 \sim (2.0 - 2^{-52})$
사배정도	128	15	112	16383	$-16382 \sim +16383$	$1.0 \sim (2.0 - 2^{-112})$

정규 값

- $103.625_{10} = 1100111.101_2$
- $(-1)^s \times m \times 2^e$의 꼴
  - $(-1)^0 \times 1.100111101 \times 2^6$
  - $E = e + bias, E = 6 + 127 = 133 = 10000101_2$

0 1000 0101 1001 1110 1000 0000 0000 000

정규 값

- $-3.141595\times10^{10} = (-1)^1\times$ $1.1101010000100010010101101011\times2^{34}$
- '가장 가까운 짝수로 라운딩' 적용 후
  - $(-1)^1\times1.11010100001000100101011\times2^{34}$
- $E = e + bias, E = 34 + 127 = 161 = 10100001_2$
- 하지만 부동소수점 표현이 나타내는 값을 다시 계산하여 보면 -3.14159493×10^{10}

1 1010 0001 1101 0100 0010 0010 0101 011

서브 노멀 값

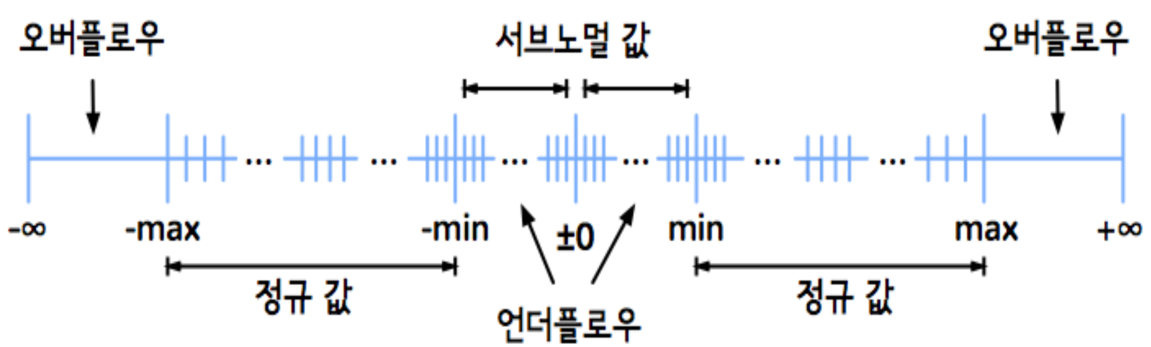
- Subnormal value 또는 비정규 값(denormalized value)
- $E = 00\cdots0_2$ 이고 $F \neq 0.00\cdots0_2$ 일 경우
- $v = (-1)^s \times m \times 2^{1-bias},\ 0 < m < 1$
- m의 값은 소수부 F의 값을 f라 할 때, $m = f$
  - 서브노멀 값은 소수부의 바로 왼쪽에 소수점을 가정하나 정규 값처럼 암묵적인 1을 가정하지 않음
- 2^{-149}

$$v = (-1)^0 \times 0.00000000000000000000001_2 \times 2^{1-127} = 2^{-23+(-126)} = 2^{-149}$$

0   0000 0000   0000 0000 0000 0000 0000 001

오버플로우와 언더플로우

- 값의 크기가 너무 커서 정규 값으로 나타낼 수 없을 경우 오버플로우가 일어나고 이 값은 +∞나 −∞로 표현됨
- 언더플로우의(underflow)의 경우는 값의 크기가 너무 작아서 나타낼 수 없을 때를 일컬음
  - 서브노멀 값이 언더플로우를 어느 정도 완충해 주는 작용을 함
  - 그 값에 가장 가까운 서브노멀 값으로 라운딩을 통해 어림잡음
  - 서브노멀 값으로도 표현할 수 없을 경우는 0.0으로 어림잡음



무한대

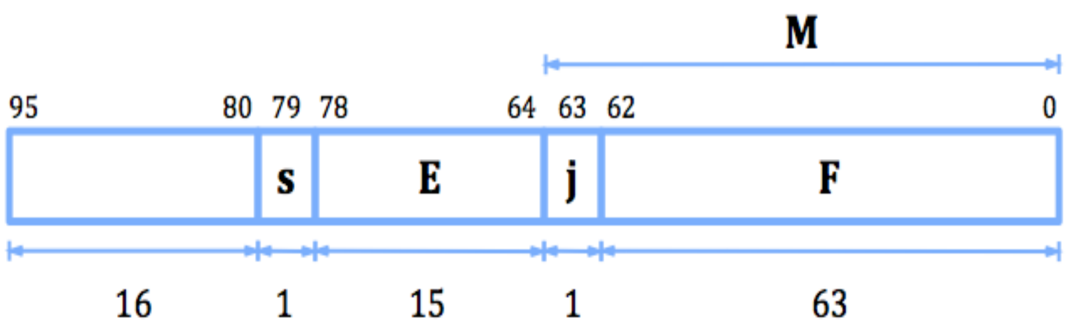
- $E = 11 \cdots 1_2$ 이고 $F = 0.00 \cdots 0_2$ 일 경우
- 사칙연산이 무한대를 피연산자로 가지는 경우, 그 연산의 결과값에 대한 규칙이 IEEE 754 표준에 정의되어 있음
  - $234.5 \div \infty = 0.0$
  - $-3.14 \times \infty = -\infty$
- 무한대와 정규 값 또는 서브노멀 값을 서로 비교하는 연산
  - $+\infty$ 는 다른 어떤 정규 값 또는 서브노멀 값보다 크고 $-\infty$ 는 작음

NaN

- $E = 11\cdots1_2$ 이고 $F \neq 0.00\cdots0_2$ 일 경우
- Not a Number
- 계산의 결과값을 정규화된 값, ± 0.0, 서브노멀 값 또는 $\pm\infty$로 나타낼 수 없는 경우에 사용
- NaN이 결과로 나올 수 있는 연산의 예
  - $3.15 + \mathrm{NaN}$, $0.0/0.0$, $\pm\infty/\pm\infty$, $0\times\pm\infty$, $+\infty + (-\infty)$, $\sqrt{-1.0}$, $\log(-1.0)$
- 무한대와 달리 NaN은 다른 수와 그 크기를 비교할 수 없음

X86 80-비트 확장 정도 표현

- X86 80-bit extended precision
- Intel의 80x87 계열의 부동소수점 코프로세서(coprocessor)를 위해 처음 제안되었고, 그 이후 x86 아키텍처의 부동소수점 연산장치에서 계속 사용함
- j와 F 사이에 암묵적인 소수점이 있는 것으로 간주
- 저장되는 메모리의 크기
  - 시스템에 따라 다른 얼라인먼트(alignment) 요구를 만족해야 함
  - 시스템에 따라 보통 96 비트나 128 비트가 할당됨
- 바이어스된 지수의 바이어스는 16383
  - 실제 지수 e는 $e = E - 16383$으로 계산됨



X86 80-비트 확장 정도 표현

경우	값
$0 < E < 11\cdots1_2, j = 1$	$(-1)^s \times (1+F) \times 2^{E-16383}$ (정규 값, normal values)
$0 < E < 11\cdots1_2, j = 0$	정의되지 않음
$E = 00\cdots0_2, j = 0, F \neq 0.00\cdots0_2$	$(-1)^s \times F \times 2^{E-16382}$ (서브노멀 값, subnormal values)
$E = 00\cdots0_2, j = 0, F = 0.00\cdots0_2$	$(-1)^s \times 0.0$ (± 0)
$E = 00\cdots0_2, j = 1$	$(-1)^s \times (1+F) \times 2^{E-16382}$ (수도 디노멀 값, pseudo-denormal values)
$s = 0, E = 11\cdots1_2, j = 1, F = 0.00\cdots0_2$	$+\infty$
$s = 1, E = 11\cdots1_2, j = 1, F = 0.00\cdots0_2$	$-\infty$
$E = 11\cdots1_2, j = 1, F = 0.1XX\cdots X_2$	QNaN (Quiet NaN), 익셉션을 발생시키지 않음
$E = 11\cdots1_2, j = 1, F = 0.0XX\cdots X_2,$ $F \neq 0.00\cdots0_2$	SNaN (Signaling NaN), 익셉션을 발생시킴

X86 80-비트 확장 정도 표현

- $-3.141595\times10^{10} = -1.11010100001000100101011010101011_2\times2^{34}$
- $(-1)^s\times M\times2^{E-16383}$의 형태
  - $(-1)^1\times(1.11010100001000100101011010101011_2)\times 2^{16417-16383}$
- $s = 1, j = 1, F = 1101010000100010010101101010101l1,$ $E = 16417_{10} = 100000000100001_2$

1	100 0000 0010 0001	1	1101 0100 0010 0010 0101 0110 1010 1100 0000 0000 0000 0000 0000 0000 0000 000

부동소수점 연산

부동소수점 연산

- IEEE 754 부동소수점 표준은 덧셈과 곱셈 같은 산술연산의 결과를 결정하는 간단한 규칙을 제공
- 실수의 덧셈은 교환법칙(commutativity)과 결합법칙(associativity)이 성립
  - 부동소수점 연산은 결합법칙이 성립하지 않음
  - $(1.125 +_r 10^{10}) +_r (-10^{10}) \neq 1.125 +_r (10^{10} +_r (-10^{10}))$
- 곱셈도 결합법칙이 성립하지 않음
  - $(10^{30} \times_r 10^{30}) \times_r 10^{-30} \neq 10^{30} \times_r (10^{30} \times_r 10^{-30})$

부동소수점 표현의 덧셈과 뺄셈

- 두 수 x와 y의 뺄셈 $x-y$는 덧셈 $x+(-y)$로 변환
- 정규화된 과학적 표기법이 모두 6 자리의 가수와 2 자리의 지수를 허용한다고 가정
  - $x = 9.96875\times10^{1}$
  - $y = 3.18750\times10^{-1}$
- 작은 지수를 가진 수의 소수점을 큰 지수를 가진 수의 소수점에 맞추어 두 수의 지수를 일치시킴
  - $y = 0.031875\times10^{1}$
- 두 수의 가수를 더함
- 결과는 10.000625×10^{1}
- 가수가 6 자리인 정규화된 과학적 표기법으로 나타냄
  - ’가장 가까운 짝수로 라운딩(round to the nearest, ties to even)’
  - 10.0006×10^{1}
- 결과는 $x+y = 1.00006\times10^{2}$

$$
\begin{array}{rcccccccc}
 & & 9 & . & 9 & 6 & 8 & 7 & 5\ 0 \\
+ & & 0 & . & 0 & 3 & 1 & 8 & 7\ 5 \\
 & 1 & 0 & . & 0 & 0 & 0 & 6 & 2\ 5
\end{array}
$$

부동소수점 표현의 덧셈과 뺄셈

- $x = 99.6875_{10}$와 $y = 0.31875_{10}$
  - $x = 1100011.1011_2 = (-1)^0 \times 1.1000111011_2 \times 2^6$
  - $y = 0.0101000110011001 1 \ldots_2 = (-1)^0 \times 1.01000110011001100 11 \ldots_2 \times 2^{-2}$
  - $E_x = e_x + 127_{10} = 6_{10} + 127_{10} = 133_{10} = 10000101_2$
  - $E_y = e_y + 127_{10} = -2_{10} + 127_{10} = 125_{10} = 01111101_2$
  - $F_x = m_x - 1 = 0.1000\ 1110\ \ 1100\ 0000\ 0000\ 000_2$
  - $F_y = m_y - 1 = 0.0100\ 0110\ 0110\ 0110\ 0110\ 011_2$

	s	E	$1.F$
x :	0	1000 0101	1.1000 1110 1100 0000 0000 000
y :	0	0111 1101	1.0100 0110 0110 0110 0110 011

부동소수점 표현의 덧셈과 뺄셈

- 두 지수의 차의 크기 $d = e_x - e_y = E_x - E_y = 8$을 구함
- x와 y 중 지수가 작은 수 y의 가수를 d 만큼 오른쪽으로 시프트하여 두 수의 지수를 일치시킴

x :	0	1000 0101	1.1000 1110 1100 0000 0000 000
y :	0	1000 0101	0.0000 0001 0100 0110 0110 011

부동소수점 표현의 덧셈과 뺄셈

- 자리를 맞춘 두 가수를 더하고 결과의 지수를 x와 y 중 큰 수의 지수에 맞춤
- 결과를 십진수로 표현하면 $100.0062484741210937 5_{10}$가 되는데, 원래 덧셈의 결과값인 100.00625_{10}와 차이가 남

x :	0	1000 0101	1.1000 1110 1100 0000 0000 000
y :	0	1000 0101	0.0000 0001 0100 0110 0110 011
$x+y$:	0	1000 0101	1.1001 0000 0000 0110 0110 011

라운딩 오차

- 실수
  - $x = (-1)^s \times m \times 2^e, \quad 1 \le m < 2$
- 단정도 IEEE 부동소수점 표현
- m_r과 m 간 절대오차(absolute error)의 상한(上限, upper bound)
  - $|m_r - m| \le \frac{1}{2} \times 2^{-23}$
- x_r와 x 간 절대오차의 상한
  - $|x_r - x| \le \frac{1}{2} \times 2^{-23} \times 2^e$
- x_r와 x 간의 상대오차(relative error)의 상한
  - $\frac{|x_r - x|}{|x|} \le \frac{\frac{1}{2} \times 2^{-23} \times 2^e}{|m| \times 2^e} = \frac{2^{-24}}{|m|}$
  - $1 \le m$이므로, $\frac{|x_r - x|}{|x|} \le 2^{-24}$

소수부가 k 비트인 IEEE 부동소수점 표현

- $\frac{|x_r - x|}{|x|} \leq \frac{1}{2} \times 2^{-k}$
- 상한 $\frac{1}{2} \times 2^{-k}$을 머신 엡실론(machine epsilon) 또는 유닛 라운드오프(unit roundoff)라고 부름
- 표현하는 수의 크기와 관계없이 이 수를 IEEE 754 이진 부동소수점 표현으로 나타내었을 때의 상대적인 라운딩 오차의 범위가 동일
  - 아래를 만족하는 어떤 실수 ε이 항상 존재
  - $x_r = x(1+\varepsilon), \quad |\varepsilon| \leq \frac{1}{2} \times 2^{-k}$

부동소수점 표현의 곱셈

- 두 실수 $x = 99.6875_{10}$와 $y = 0.31875_{10}$
  - 정규화된 과학적 표기법, 6 자리의 가수와 2 자리의 지수를 허용
    - $x = 9.96875 \times 10^{1}$
    - $y = 3.18750 \times 10^{-1}$
- $x \times y$를 계산하기 위해 먼저 두 수의 지수를 더하면 0이 됨
- 두 수의 가수를 부호를 고려하여 곱하면 $9.96875 \times 3.18750 = 31.775390625$가 됨
  - $x \times y = 31.775390625 \times 10^{0}$
- 정규화가 필요
  - '가장 가까운 짝수로 라운딩(round to the nearest, ties to even)'을 적용
  - $x \times y = 3.177539 \times 10^{1}$

부동소수점 표현의 곱셈

- 두 실수 $x = 99.6875_{10}$와 $y = 0.31875_{10}$
- 두 지수의 합 $e_{x \times y} = e_x + e_y$을 구함
  - E_x와 E_y가 바이어스되어 있으므로 바이어스(127)을 고려하여 두 지수의 합을 구함
  - $e_{x \times y} = e_x + e_y$ $e = e_x + e_y = (E_x - 127) + \left(E_y - 127\right) = (133 - 127) + (125 - 127) = 6 + (-2) = 4$
  - $E_{x \times y} = 4 + 127 = 131 = 10000011_2$

	s	E	$1.F$
x :	0	1000 0101	1.1000 1110 1100 0000 0000 000
y :	0	0111 1101	1.0100 0110 0110 0110 0110 011

x :	0	1000 0101	1.1000 1110 1100 0000 0000 000
y :	0	1000 0101	1.0100 0110 0110 0110 0110 011
$x \times y$:	0	1000 0011	

부동소수점 표현의 곱셈

- 두 수의 가수를 곱함

x :	0	1000 0101	1.1000 1110 1100 0000 0000 000
y :	0	1000 0101	1.0100 0110 0110 0110 0110 011
$x \times y$:	0	1000 0011	1.1111 1100 0110 0111 1111 1111 0110 0000 1

- 소수부의 자릿수를 정규화(23 자리)
  - 1.1111110001100111111111101100001
  - 라운딩을 적용하면 1.11111100011010000000000

x :	0	1000 0101	1.1000 1110 1100 0000 0000 000
y :	0	1000 0101	1.0100 0110 0110 0110 0110 011
$x \times y$:	0	1000 0011	1.1111 1100 0110 1000 0000 000

- 두 수 x와 y의 부호가 같으므로 결과의 부호($s_{x \times y}$)는 0이 됨

부동소수점 표현의 나눗셈

- 나눗셈도 곱셈과 비슷하게 지수끼리 빼고, 가수끼리 나누어서 수행
- $\frac{(-1)^{s_x} \times m_x \times 2^{e_x}}{(-1)^{s_y} \times m_y \times 2^{e_y}} = (-1)^{s_x+s_y} \times \frac{m_x}{m_y} \times 2^{e_x-e_y}$

부동소수점 연산을 위한 하드웨어

- 대부분의 컴퓨터가 부동소수점 연산을 빠르게 수행하기 위하여 FPU(Floating-Point Unit)이라 불리는 부동소수점 연산장치를 따로 가지고 있음
- 부동소수점 덧셈 및 뺄셈을 수행하는 기본적인 하드웨어
  - 지수를 비교하는 하드웨어
  - 지수의 차를 구하는 고정소수점 ALU
  - 가수의 소수점을 맞추는 시프터
  - 가수를 더하는 고정소수점 ALU
  - 결과를 정규화 하는 하드웨어
  - 라운딩을 수행하는 하드웨어
  - ± 0.0과 $\pm\infty$ 같은 특별한 값을 다루는 하드웨어
- 부동소수점 곱셈이나 나눗셈을 수행하는 하드웨어는 덧셈 및 뺄셈을 수행하는 하드웨어보다 상대적으로 간단

Fused Multiply-Add

- 부동소수점 곱셈을 수행하고 바로 다음에 부동소수점 덧셈을 수행하는 연산인데, $x \times y + z$ 형태의 계산을 한번에 수행하는 연산
- FMA 연산을 수행하는 특별한 연산장치
  - FMA 장치가 있는 프로세서는 보통 FMA 연산을 위한 머신 인스트럭션을 제공
  - 범용 CPU와 GPU를 포함하여 현재 대부분의 프로세서가 FMA 연산을 지원하는 머신 인스트럭션을 제공
  - FMA 인스트럭션을 이용하는 코드를 생성하려면 컴파일러 옵션 이용
- FMA 연산이 특히 유용한 경우
  - 내적(dot product)을 구하는 연산
  - 행렬을 곱하는 연산
  - 다항식의 계산
    - $c_n x^n + c_{n-1} x^{n-1} + \cdots + c_0 x^0 = \left(\cdots\left(\left(c_n x + c_{n-1}\right)x + c_{n-2}\right)x + \cdots + c_1\right)x + c_0$
- FMA 연산을 이용하여 계산 속도와 정확도를 높임
  - 한번의 라운딩만 적용