Floating Point Numbers
(and use in Modern AI)

Instructor: Marcelo Siero
CSE12

Floating Point Numbers

Consider: $A \times 10^B$, where A is one decimal digit, B, a signed number, allows this one number to express extremely large numbers, or extremely small numbers based on the value of the exponent B.

A	B	$A \times 10^B$
0	any	0
1 .. 9	0	1 .. 9
1 .. 9	1	10 .. 90
1 .. 9	2	100 .. 900
1 .. 9	-1	0.1 .. 0.9
1 .. 9	-2	0.01 .. 0.09

Using exponential powers you can express a very large dynamic range, i.e. huge numbers and very small numbers.

International System of Units (SI)

Prefix		Base 10	Decimal	Adoption [nb 1]
Name	Symbol			
quetta	Q	10^{30}	1 000 000 000 000 000 000 000 000 000 000	2022[3]
ronna	R	10^{27}	1 000 000 000 000 000 000 000 000 000	2022
yotta	Y	10^{24}	1 000 000 000 000 000 000 000 000	1991
zetta	Z	10^{21}	1 000 000 000 000 000 000 000	1991
exa	E	10^{18}	1 000 000 000 000 000 000	1975
peta	P	10^{15}	1 000 000 000 000 000	1975
tera	T	10^{12}	1 000 000 000 000	1960
giga	G	10^{9}	1 000 000 000	1960
mega	M	10^{6}	1 000 000	1873
kilo	k	10^{3}	1 000	1795
hecto	h	10^{2}	100	1795
deca	da	10^{1}	10	1795
		10^{0}	1	—

For **integer** type numbers,
i.e. >= 1.0

A **metric prefix** is a unit prefix that precedes a basic unit of measure to indicate a multiple or sub-multiple of the unit. These metric prefixes used today are **decadic**.

SI prefixes are metric prefixes that were standardised for use in the International System of Units (SI) by the International Bureau of Weights and Measures (BIPM) in resolutions dating from 1960 to 2022

Source: Wikipedia "Metric_prefix"

International System of Units (SI)

		10^{0}	1	—
deci	d	10^{-1}	0.1	1795
centi	c	10^{-2}	0.01	1795
milli	m	10^{-3}	0.001	1795
micro	μ	10^{-6}	0.000 001	1873
nano	n	10^{-9}	0.000 000 001	1960
pico	p	10^{-12}	0.000 000 000 001	1960
femto	f	10^{-15}	0.000 000 000 000 001	1964
atto	a	10^{-18}	0.000 000 000 000 000 001	1964
zepto	z	10^{-21}	0.000 000 000 000 000 000 001	1991
yocto	y	10^{-24}	0.000 000 000 000 000 000 000 001	1991
ronto	r	10^{-27}	0.000 000 000 000 000 000 000 000 001	2022
quecto	q	10^{-30}	0.000 000 000 000 000 000 000 000 000 001	2022

For **fractional** type numbers i.e. < 1.0

Source: Wikipedia “Metric_prefix”

Real numbers as Decimal Floating Point

- Our decimal system handles non-integer *real* numbers by adding an extra symbol, the decimal point (.) for **fixed point** notation:
  - e.g. $3{,}456.78 = 3*10^3 + 4*10^2 + 5*10^1 + 6*10^0 + 7*10^{-1} + 8.10^{-2}$
- The **floating point**, or **scientific**, notation further allows us to represent very large and very small numbers (integer or real), with as much or as little precision as needed, e.g.:
  - Unit of electric charge e = $1.602\ 176\ 462 * 10^{-19}$ Coul.
  - Volume of universe = $1 * 10^{85}$ cm^3
    - the two components of these numbers are called the mantissa (provides precision) and the
    - exponent (provides dynamic range)

How to do scientific notation in binary?
Standard: IEEE 754 Floating-Point

Real numbers in Binary Floating Point

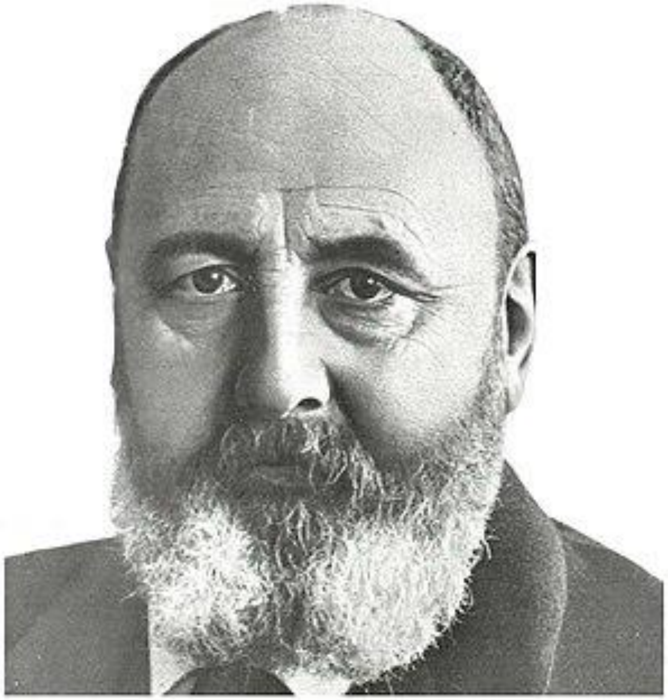
- We mimic the decimal floating point notation to create a “hybrid” binary floating point number:
  - We first use a **“binary point”** to separate whole numbers from fractional numbers to make a fixed point notation:
    - e.g. 00011001.110 = $1*2^4 + 1*2^3 + 1*2^0 + 1*2^{-1} + 1*2^{-2}$ => 25.75
      ($2^{-1} = 0.5$ and $2^{-2} = 0.25$, etc.)
  - We then **“float” the binary point**:
    - **00011001.110 => 1.1001110 x 2^4**
      mantissa = 1.1001110, exponent = 4

Note: The 1st 1 coresponds to the most significant bit of the binary number.

  - Now we have to express this without the extra symbols (x, 2, .)
    - by convention, we divide the available bits into three fields:

sign, mantissa, exponent

INNOVATORS OF FLOATING POINT



In 1914, Leonardo Torres y Quevedo proposed a form of floating point in the course of discussing his design for a special-purpose electromechanical calculator

In 1938, Konrad Zuse of Berlin completed the Z1, the first binary, programmable mechanical computer;[9] it uses a 24-bit binary floating-point number representation with a 7-bit signed exponent, a 17-bit significand (including one implicit bit), and a sign bit

Source: Wikipedia

STANDARDIZATION OF FLOATING POINT

Floating-point **compatibility across multiple computing systems** was in desperate need of standardization by the early **1980s**, leading to the creation of the **IEEE 754 standard** once the **32-bit (or 64-bit)** word had become commonplace. This standard was significantly based on a proposal from Intel, which was designing the i8087 numerical co-processor; Motorola, which was designing the 68000 around the same time, gave significant input as well.

In 1989, a Canadian mathematician and computer scientist **William Kahan** was honored with the Turing Award for being the **primary architect** behind this proposal; he was aided by his student Jerome Coonen and a visiting professor, Harold Stone.[13]

Kahan is now emeritus professor of mathematics and of electrical engineering and computer sciences (EECS) at the University of California, Berkeley.

Source: Wikipedia

IEEE-754 fp Numbers Single Precision

32 bits:	1	8 bits	23 bits (+1)
	s	biased exp.	fraction

$$N = (-1)^{s} \times 1.\mathbf{fraction} \times 2^{(\mathbf{biased\ exp.} - 127)}$$

- **Sign: 1 bit (like in sign-magnitude)**
- **Mantissa: 23 bits (like in signed-magnitude)**
  - We **"normalize"** the **mantissa** (by adjusting the exponent) and **DROP** the **leading 1** thus **recording only the fractional binary part**
- **Exponent: 8 bits**
  - To handle both positive and negative exponents , we **add 127 to** the **actual exponent** creating a "**biased exponent**":
    - 2^{-127} => biased exponent = 0000 0000 (= 0)
    - 2^{0} => biased exponent = 0111 1111 (= 127)
    - 2^{+127} => biased exponent = 1111 1110 (= 254)

Range in decimal: ~1.18e-38 to ~3.40e38 with 6–9 significant digits of precision

IEEE-754 fp numbers

- Example:
  - $25.75 \Rightarrow 00011001.110 \Rightarrow 1.1001110 \times 2^4$
  - Sign bit = 0 (indicates positive)
  - Normalized mantissa (fraction) =
  - (1.)100 1110 0000 0000 0000 0000
  - The first 1 is assumed, i.e. not explicitly given.
  - Biased exponent = 4 + 127 = 131 => 1000 0011
  - So 25.75 => 0100 0001 1100 1110 0000 0000 0000 0000
  - => 0x41CE0000 as a 32-bit number in hex

How to convert 64.2 into IEEE SP

- 1) Get integer and fractional binary form for 64.2
  - Binary left of radix/decimal point (integer): 1000000
  - Binary of right of radix/decimal (fractional):
    - Successively multiply value by 2 and compare to 1
      - 0.2 x 2 = 0.4 less than 1 so… **0**
      - 0.4 x 2 = 0.8 less than 1 so… **0**
      - 0.8 x 2 = 1.6 g.t. 1 so… **1**
      - 0.6 x 2 = 1.2 g.t. 1 so… **1**
      - 0.2 x 2 = 0.4 **0**
      - 0.4 x 2 = 0.8 **0**
      - 0.8 x 2 = 1.6 **1**
      - 0.6 x 2 = 1.2 **1**
      - **… (repeats)**

(continued)

- So the integer for 64: 1000000
- Binary for .2: .0011 0011 0011 0011 ...
- 64.2 is: 1000000.0011 0011 0011 0011 …

- 2) Normalized in binary form
  - Produces: $1.00000000110011... \text{ X } 2^6$

(continued)

- 3. Turn positive exponent into biased with 127
  - E = 6 + 127 = 133 = 10000101
- 4. Put it together:
  - 23-bit F is: (1.)0000000011 0011 0011 0011 0
- S E F is: (Sign, Exponent, Fraction)
  - S = 0
  - E = 10000101
  - F = 00000000110011001100110
- In hex:
  - 0x42806666

  0100 0010 1000 0000 0110 0110 0110 0110

Convert IEEE SP to Decimal Real

- What is the decimal value for this SP FP number 0xC228 0000?
  - Convert to binary

    1100 0010 0010 1000 0000 0000 0000 0000
  - Break into S, E, F:
  - E is 10000100 = 132 decimal: 132 – 127 = 5
  - F is (1.)0101000…
  - Move decimal over 5: 101010.000…
  - Accounting for S E F results in -42!

Convert IEEE SP to Real: Example

- 0x3F800000

Convert IEEE SP to Real

- 0x3F800000

0011 1111 1000 0000 0000 0000 0000 0000

Convert IEEE SP to Real

- 0x3F800000

0011 1111 1000 0000 0000 0000 0000 0000

S = 0

E = 0111 1111 = 127 - 127

F = 1.0

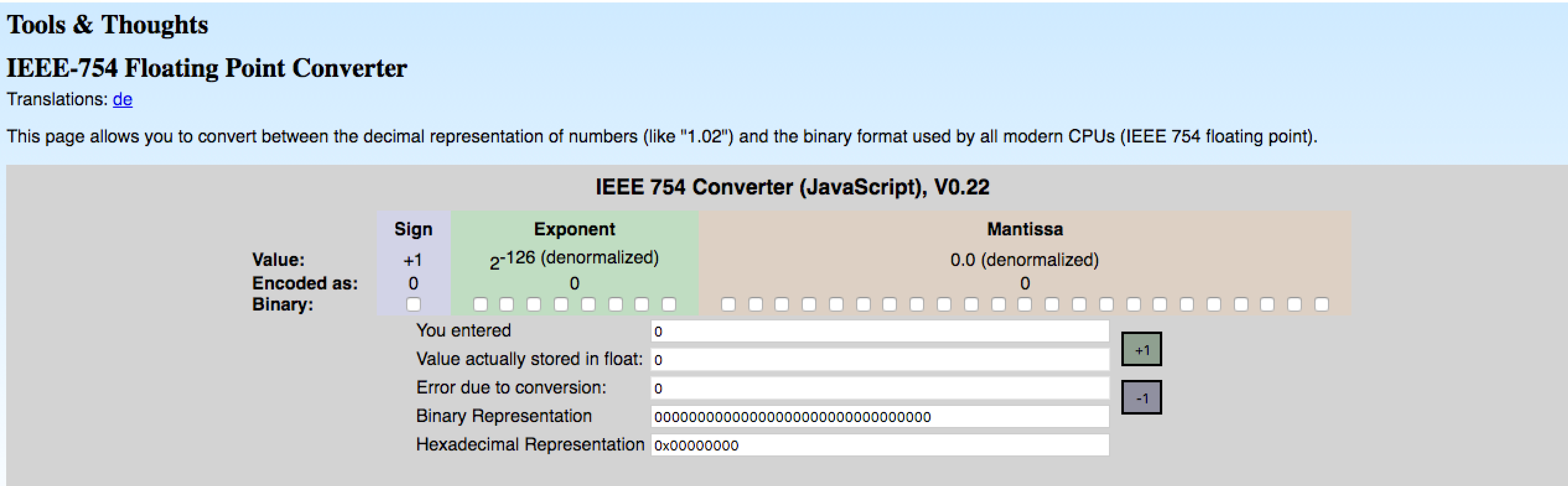
0x3F8 = 1 in single precision floating point

Take Home Practice

- What is 47.625_{10} in SP FP format?
- What is 0x44ed8000 as real number?

Check your Practice

- http://www.h-schmidt.net/FloatConverter/IEEE754.html



IEEE-754 Double Precision

- Double precision (64 bit) floating point

64 bits:	1	11 bits	52 bits (+1)
	s	**biased exp.**	**fraction**

$$N = (-1)^{\mathbf{s}} \times 1.\mathbf{fraction} \times 2^{(\mathbf{biased\ exp.} - 1023)}$$

Range: ~2.23e-308 … ~1.80e308 with full 15–17 decimal digits precision

IEEE-754 Half Precision, FP16: For AI Applications

- Half-precision (16-bit) floating point
- Trend in AI towards FP16, because lower precision calculations seem not to be critical. It often speeds up calculations, without harming the results.
- Smaller range and precision than FP32

16 bits:	1	5 bits	10 bits (+1)
	s	**biased exp.**	**fraction**

$$N = (-1)^{s} \times 1.\mathbf{fraction} \times 2^{(\mathbf{biased\ exp.} - 15)}$$

Range: (6.10e−5 to 6.1e+4) with 3.3 significant decimal digits precision
Supported by modern NVIDIA GPUs and others.

Source: https://moocaholic.medium.com/fp64-fp32-fp16-bfloat16-tf32-and-other-members-of-the-zoo-a1ca7897d407
Source: https://blogs.nvidia.com/blog/tensorfloat-32-precision-format/

BFP16: Brain Floating Point, AI Applications

- Half-precision (16-bit) floating point
- Trend in AI towards FP16, because lower precision calculations seem not to be critical. It often speeds up calculations, without harming the results.
- Same dynamic range but less precision than FP32
- Better for training avoiding underflow and overflow.
- Faster and smaller than FP32
- Google TPU, Intel AMX, NVIDIA Hopper,

16 bits:

1	8 bits	7 bits (+1)
s	**biased exp.**	**fraction**

$$N = (-1)^{s} \times 1.\mathbf{fraction} \times 2^{(\mathbf{biased\ exp.} - 127)}$$

Range: (+/- 3.4e−38 to 3.4e+38) with ~2.4 significant decimal digits precision

Source: https://moocaholic.medium.com/fp64-fp32-fp16-bfloat16-tf32-and-other-members-of-the-zoo-a1ca7897d407
Source: https://blogs.nvidia.com/blog/tensorfloat-32-precision-format/
Source: https://nhigham.com/2020/06/02/what-is-bfloat16-arithmetic/

Processors with BFloat16 Support

Vendor	Product / Family	Type	BF16 Support	Architecture / Notes
Intel	Xeon Scalable (4th Gen)	CPU	Yes	AMX, Sapphire Rapids
Intel	Xeon Scalable (3rd Gen)	CPU	Yes	AVX-512 + BF16
Intel	Habana Gaudi / Gaudi2	AI Accelerator	Yes	Native BF16
AMD	EPYC 9004 (Zen 4)	CPU	Yes	AVX-512 + BF16
AMD	Ryzen 7000 (Zen 4)	CPU	Yes	AVX-512 BF16
AMD	Instinct MI300 Series	GPU	Yes	BF16, FP8, CDNA 3
Google	TPU v2, v3, v4	AI Accelerator	Yes	Native BF16 only
NVIDIA	A100 (Ampere)	GPU	Yes	Tensor Cores: FP16, BF16, FP32
NVIDIA	H100 (Hopper)	GPU	Yes	BF16, FP8, Tensor memory format
ARM	Neoverse V2 / V3	CPU	Partial	SVE2 + BF16 optional

BFP16: Brain Floating Point Notes

- **AVX-512 BF16** is key enabling instruction set for x86 CPUs (Intel & AMD).
- **AMX** (Intel) provides matrix-tile-based H/W blocks accelerate BF16 DNN ops
- **TPUs** use BF16 as their primary numeric format — no FP32 inside.
- **NVIDIA** uses BF16 in newer generations (Ampere and Hopper)
- **BF16 is not IEEE 754**, but most compilers/hardware treat it as a standard due to its practical benefits
- **BF16 Supports:** Zero, NaN, +/- Infinity, and sub-normal?? (see up-coming slides)

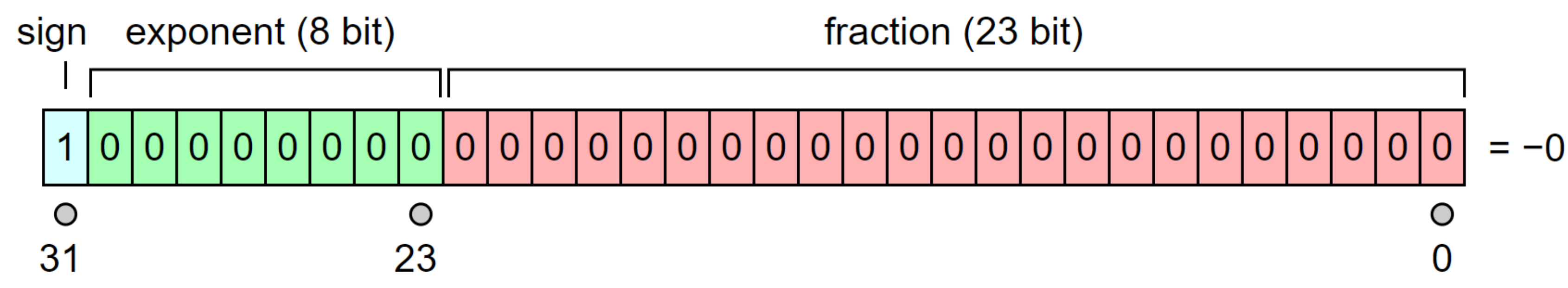
Great Reference: https://github.com/riscv/riscv-isa-manual/blob/main/src/bfloat16.adoc

Floating-Point Format Comparison

Format	Bits	Exponent Bits	Mantissa Bits	Exponent Bias	Precision (≈ digits)	Dynamic Range	IEEE 754?	Notes
FP64	64	11	52	1023	~15.7	±1.8e-308 to 1.8e+308	Yes	Double precision
FP32	32	8	23	127	~7.2	±1.4e-45 to 3.4e+38	Yes	Standard float
FP16	16	5	10	15	~3.3	±6.1e-5 to 6.5e+4	Yes	Half precision
BFloat16	16	8	7	127	~2.4	±3.4e-38 to 3.4e+38	No	Same range as FP32
TF32	19	8	10	127	~3.3	±1.4e-45 to 3.4e+38	No	NVIDIA-only format
FP8 (E4M3)	8	4	3	7	~1.0	±6e-3 to 2.4e+1	No	Used for ML inference
FP8 (E5M2)	8	5	2	15	~0.75	±6e-5 to 6.5e+4	No	Wider range, less precision

Expressing Zero in IEEE 754

In IEEE 754 binary floating-point formats, zero values are represented by the convention of having the biased exponent and significand both being zero.



Unlike Integers, Floating Point has both positive and negative zeros.

Note when exponent = 0 the fraction is de-normalized (i.e. no hidden 1 is now implied)

Source: By Octahedron80 - Modified from Image:IEEE 754 Single Floating Point Format.svg, original by Codekaizen, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=3697716

Representing NaN, +/- Infinity

Like for 0, these values are represented based on a convention:

- **Infinity (+ and -)**:
- exponent = 255 (1111 1111)
- and fraction = 0
- **NaN (Not a Number):**
- exponent = 255 and fraction ≠ 0
- e.g. 0.0 divided by 0.0
- is arithmetically undefined or a NaN.

Subnormal Numbers

- A normal number is defined as a floating point number with a 1 at the start of the significand.
  Thus, the smallest normal number in double precision is 1.000...x2-1022.
  The smallest representable normal number is called the underflow level, or UFL.
  Can go even smaller by removing restriction that first number significand myst be a 1.
  These numbers are known as subnormal, and are **stored with all zeros in the exponent**.
  Technically, zero is also a subnormal number.
  Subnormal numbers do not have as many significant digits as normal numbers.
  The use of subnormal numbers allows for more gradual underflow to zero.

32-bit (SP) subnormal number:

1	8 bits	23 bits
s	**00000000**	**010 0000 0000 0000 0000**

$$N = (-1)^s \times 0.\mathbf{fraction} \times 2^{(-126)}$$

Source: https://courses.physics.illinois.edu/cs357/sp2020/notes/ref-4-fp.html

Example FP8: For AI Applications

- There are 2 8-bit floating points formats, used for AI.
- FP8, lower precision calculations always critical.
  It can speeds up calculations and takes less space.
- Better dynamic range than int8
- e.g.

8 bits: **1** **4 or 5 bits** **3 or 2 bits (+1)**

s	biased exp.	fraction

$$N = (-1)^s \times 1.\mathbf{fraction} \times 2^{(\mathbf{biased\ exp.} - (7 \text{ or } 15))}$$

Supported by very modern NVIDIA GPUs

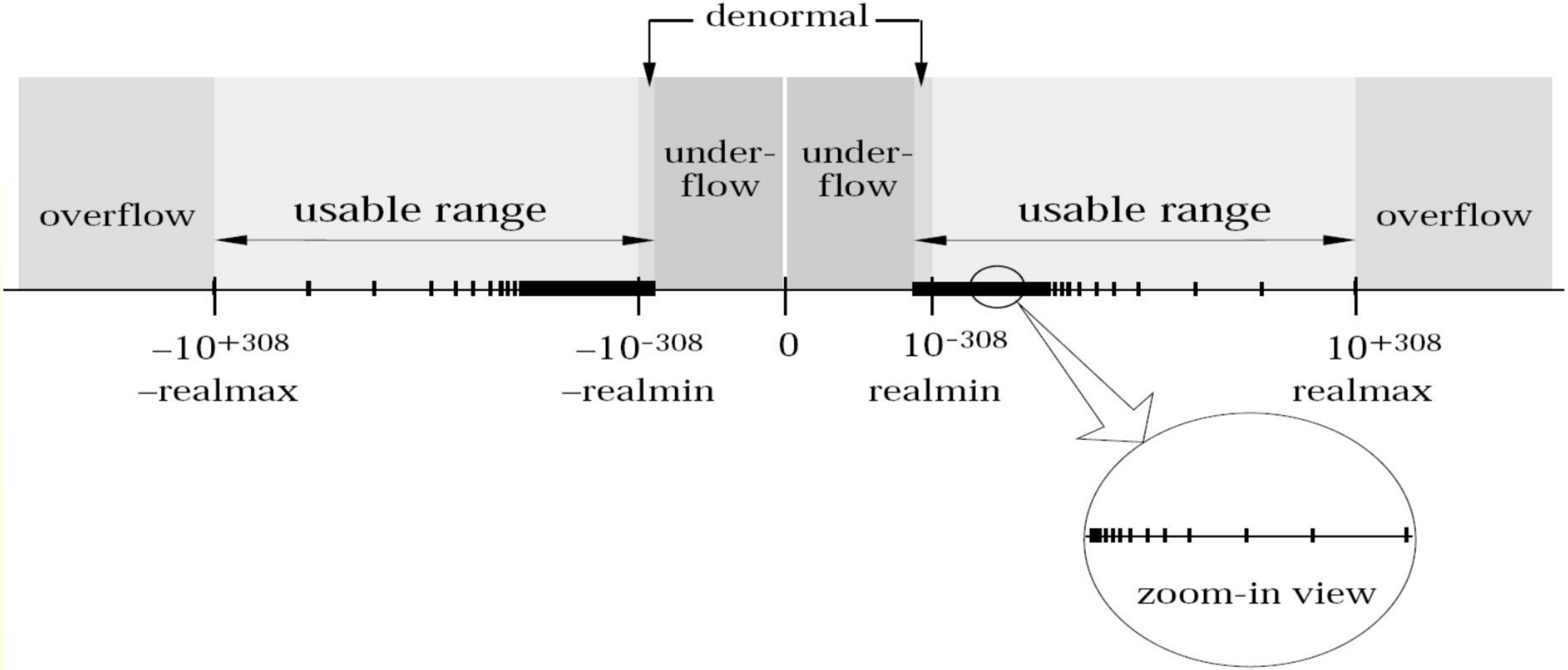
Source: https://moocaholic.medium.com/fp64-fp32-fp16-bfloat16-tf32-and-other-members-of-the-zoo-a1ca7897d407

Details of FP8 Binary Formats:

	E4M3	E5M2
Exponent Bias	7	15
Infinities	N/A	S11111.00
NaN	S1111.111	S11111.[01,10,11]
Zeros	S0000.000	S00000.00
Max Normal	S0000.111 = $1.75*2^{-6}$=448	S11110.11 = $1.75*2^{-15}$=57,344
Min Normal	S0001.000 = 2^{-6}	S00001.00 = 2^{-14}
Max subnorm	S0000.111 = $0.875*2^{-6}$	S00000.11 = $0.75*2^{-14}$
Min subnorm	S0000.001 = 2^{-9}	S00000.01 = 2^{-16}

Source: https://ar5iv.labs.arxiv.org/html/2209.05433
FP8 Formats for Deep Learning

IEEE-754 Floating Point Number Line (SP=32-bits)



Source: https://courses.physics.illinois.edu/cs357/sp2020/notes/ref-4-fp.html