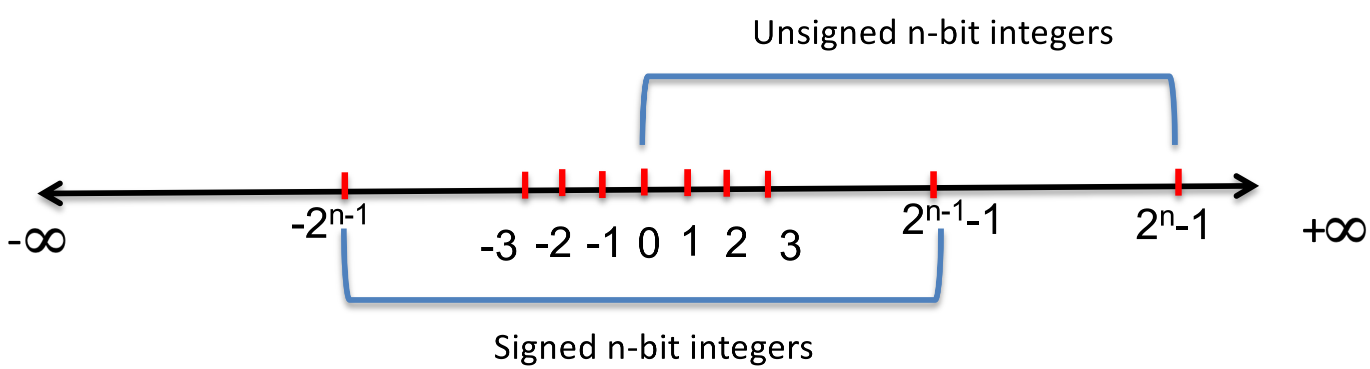
Floating point

Jinyang Li

Floating Point lesson plan

- Binary Scientific notation
- FP8 example
- IEEE FP standard
- Rounding
- FP operations caveats

Previously…



What about real numbers?

Represent real numbers: the decimal way

Real Number	Decimal Representation
11 / 2	$(5.5)_{10}$
1 / 3	$(0.3333333...)_{10}$
√2	$(1.4128...)_{10}$

$$(1.4128...)_{10} = 1 * 10^{0} + 4 * 10^{-1} + 1 * 10^{-2} + 2 * 10^{-3} + ...$$

Binary Representation

$$(5.5)_{10} = 4 + 1 + 1/2 = 2^2 + 2^0 + 2^{-1}$$

$$= (101.1)_2$$

2^p

2^{p-1}

2

1

$$b_p b_{p-1} \cdots b_1 b_0 . b_{-1} b_{-2} \cdots b_{-q} = \sum_{i=-q}^{p} 2^i \times b_i$$

$1/2$

$1/4$

$1/2^q$

Binary Representation

$$(0.1)_{10} = 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + \ldots$$

$$= (0.0001100110011\ldots)_2$$

2^p

2^{p-1}

2

1

$$b_p b_{p-1} \cdots b_1 b_0 \bullet b_{-1} b_{-2} \cdots b_{-q} = \sum_{i=-q}^{p} 2^i \times b_i$$

$1/2$

$1/4$

$1/2^q$

Binary representation

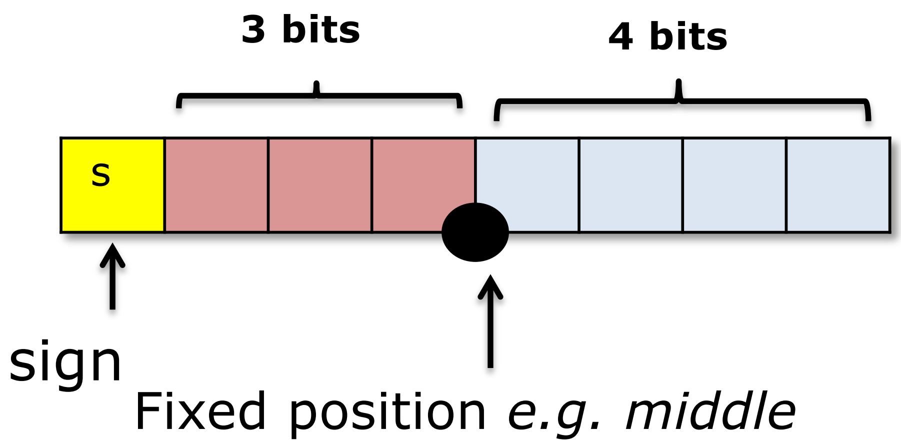
What's the decimal value of $(10.01)_2$

Binary representation

What's the decimal value of $(10.01)_2$

Answer: 2.25

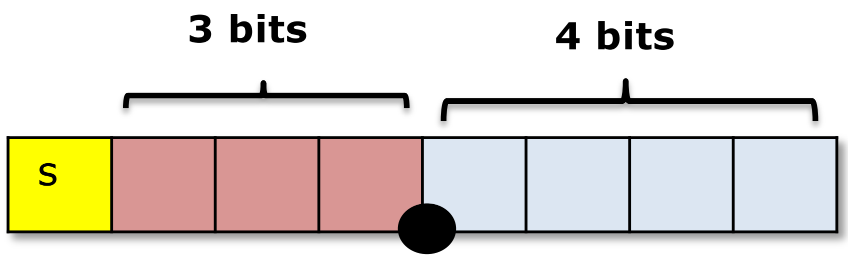
Making the representation fixed width
Strawman: fixed point



Example: $(10.011)_2$

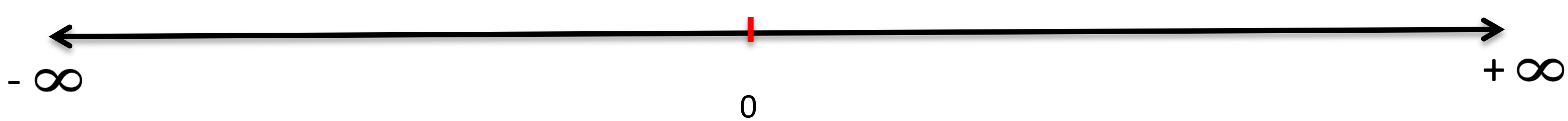
0	0	1	0	0	1	1	0

Problems of Fixed Point

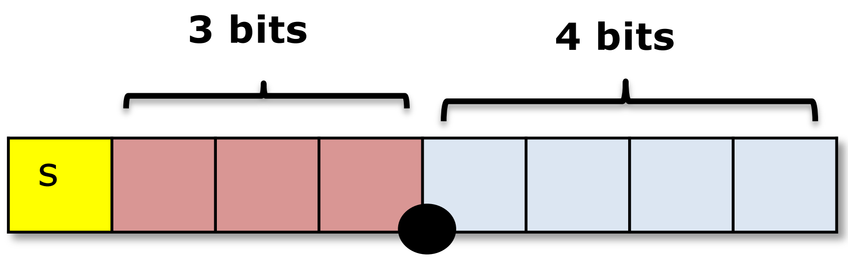


Range?

Precision?



Problems of Fixed Point



- Limited range $[-2^3+2^{-4}, 2^3-2^{-4}]$
- Limited precision 2^{-4}
- Equivalent to integer with a scaling factor

Floating Point is based on scientific notation

Scientific notation in decimal:

`365.25 = 3.6525 *` 10^2

`0.0123 = 1.23 *` 10^{-2}

Normalize mantissa

$\pm M * 10^E$, where $1 <= M < 10$

E: exponent

M: mantissa

Floating Point: binary scientific notation

Binary scientific notation

$\underline{+}M * 2^E$, where $1 <= M < 2$, aka $M = (1.b_1b_2b_3\ldots b_n)_2$

$(5.5)_{10} = (101.1)_2 = (1.011)_2 * 2^2$

(Binary) normalized representation of $(10.25)_{10}$?

(Binary) normalized representation of $(10.25)_{10}$?

Answer: $(10.25)_{10} = (1010.01)_2 = (1.01001)_2 * 2^3$

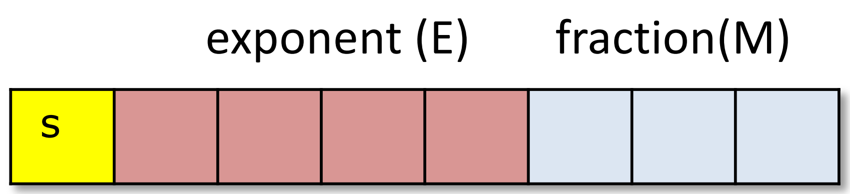
Floating Point lesson plan

- Scientific notation
- FP8 example
- IEEE FP standard
- Rounding
- FP operations caveats

An example FP8 representation: E4M3

$\underline{+}M * 2^{E}$, where $1 <= M < 2$, aka $M = (1.b_1 b_2 b_3)_2$

Example: $(0.1875)_{10} = (0.0011)_2 = (1.1)_2 * 2^{-3}$

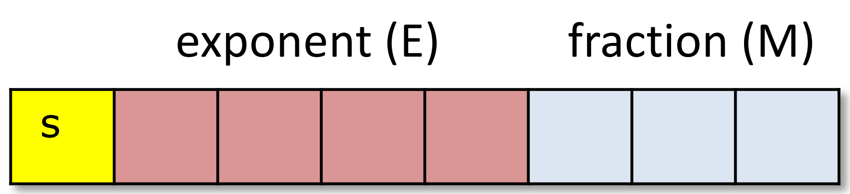


An example FP8 representation: E4M3

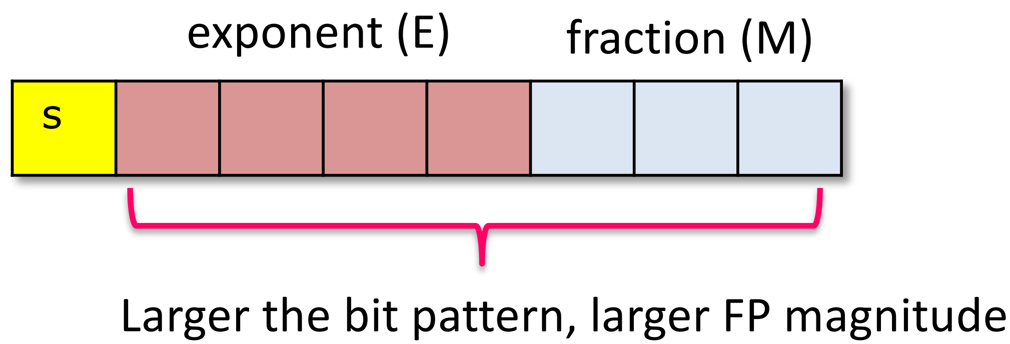
$\underline{+}M * 2^E$, where 1 <= M < 2, aka $M = (1.b_1b_2b_3)_2$

Example: $(0.1875)_{10} = (0.0011)_2 = (1.1)_2 * 2^{-3}$

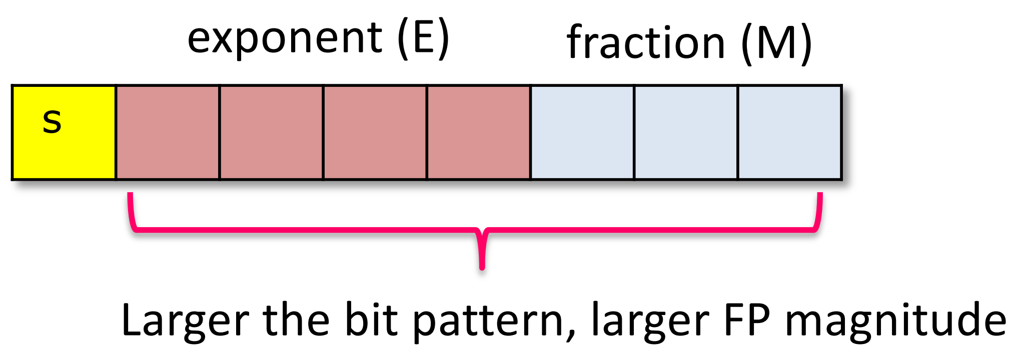
exponent = E + bias, bias= $2^{(e-1)} - 1 = 7$

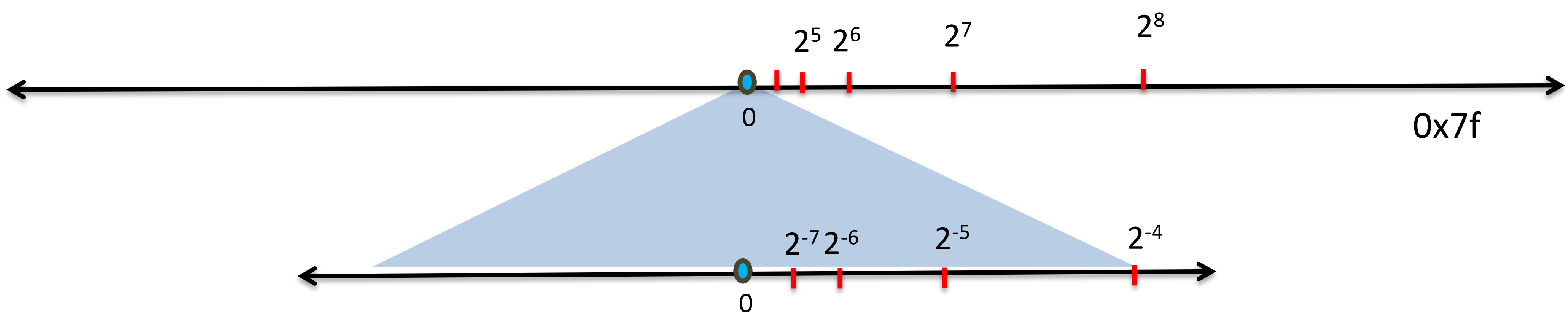


FP8 (E4M3) on the number line



FP8 (E4M3) on the number line





Having normalized representation only leaves a disproportionally large gap around zero.

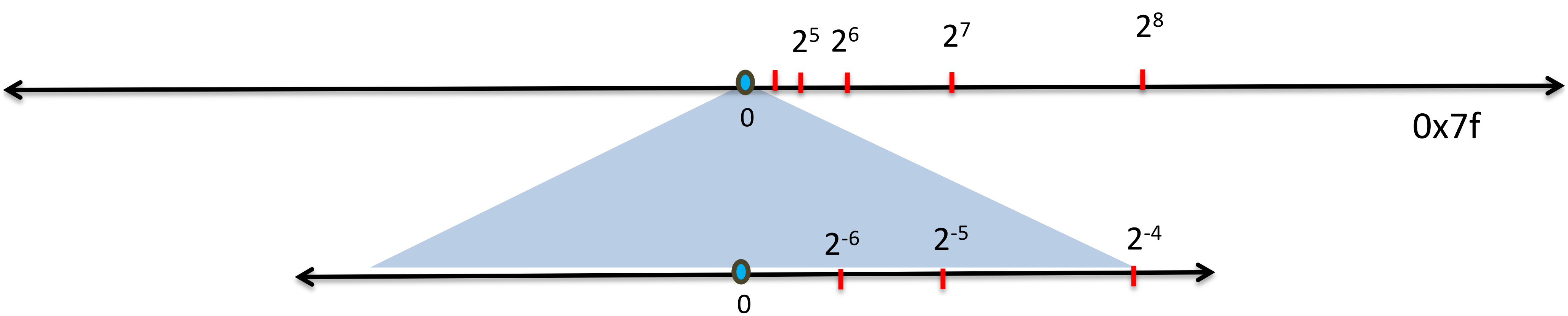
FP8 (E4M3): denormalized (subnormal) number

s	exponent (E)				fraction (M)		
s	0	0	0	0	b_1	b_2	b_3

$\pm M * 2^{1-bias}$, where $M = (0.b_1b_2b_3)_2$

Denormalize $(1.01)_2 \times 2^{-7}$?

$= (0.101)_2 \times 2^{-6}$?



IEEE Floating Point Standard

- Lots of FP implementations in 60s/70s
  - Code was not portable across processors
- IEEE formed a committee (IEEE.754) to standardize FP format and specification.
  - IEEE FP standard published in 1985
  - Led by William Kahan

Prof. William Kahan
University of California at Berkeley
Turing Award (1989)

IEEE Floating Point Standard

- This class only covers basic FP materials
- A deep understanding of FP is crucial for numerical/scientific computing
  - More FP is covered in undergrad/grad classes on numerical methods

Numerical Computing with IEEE Floating Point Arithmetic

Including One Theorem, One Rule of Thumb, and One Hundred and One Exercises

Michael L. Overton
Courant Institute of Mathematical Sciences
New York University
New York, New York

Goals of IEEE Standard

- Consistent representation of floating point numbers at various widths

- Correctly rounded floating point operations, using several rounding modes.

- Consistent treatment of exceptional situations such as division by zero

IEEE FP32 normalized + denormalized

31	30 … 23	22 … 0
s	exp = E + 127	fraction (F)

If (exp!=0 && exp!=255) $n = (1.F)_2 * 2^{exp-127}$ (normalized)

0	1000 0010	0100 1000 0000 0000 0000 000

$n = (1.01001)_2 * 2^{130-127}$

---

s	0000 0000	fraction (F)

If (exp == 0) $n = (0.F)_2 * 2^{-126}$ (denormalized)

0	0000 0000	0100 1000 0000 0000 0000 000

$n = (0.01001)_2 * 2^{-126}$

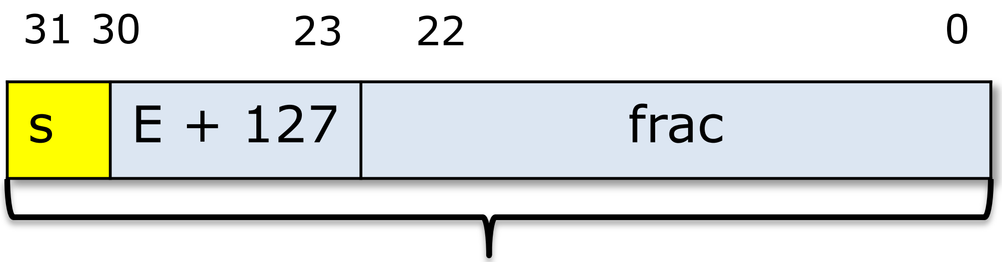
IEEE FP32: special values

Special Value's Encoding:

31	30 … 23	22 … 0
s	1111 1111	fraction (F)

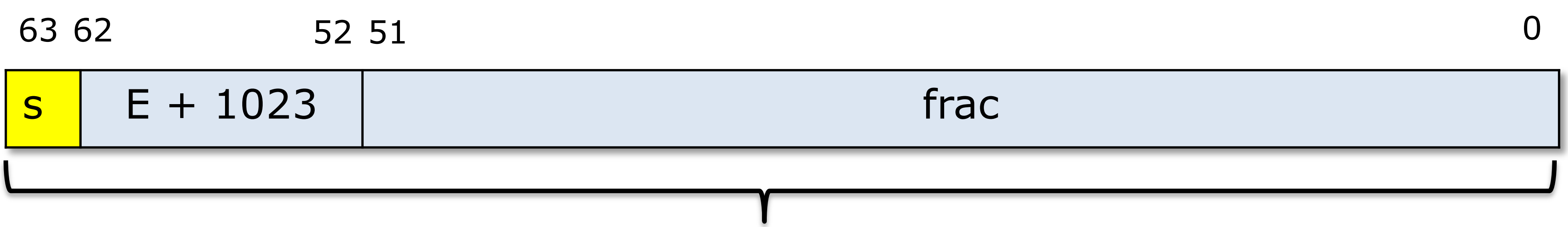
values	sign	frac
+∞	0	all zeros
- ∞	1	all zeros
NaN	any	non-zero

IEEE FP32: single vs. double precision



single precision (32 bits)

float f = 0.1;
double d = 0.1;

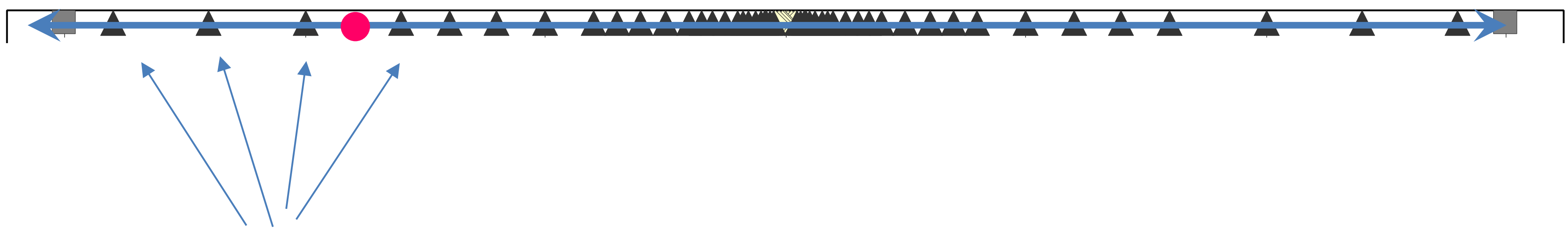


double precision (64 bits)

Floating Point lesson plan

- Scientific notation
- FP8 example
- IEEE FP standard
- Rounding
- FP operations caveats

FP: Rounding



Values that are represented precisely

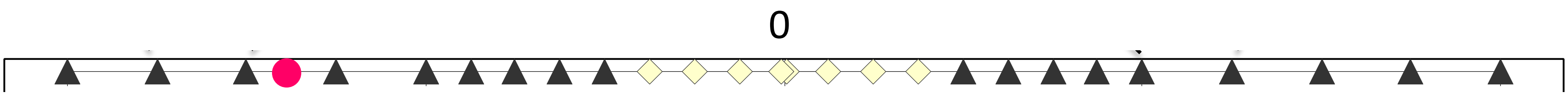
What if the result of computation is at ● ?

Rounding: Use the "closest" representable value *x'* for x.

4 modes:

- Round-down
- Round-up
- Round-toward-zero
- Round-to-nearest (Round-to-even in text book)

Round up vs. round down



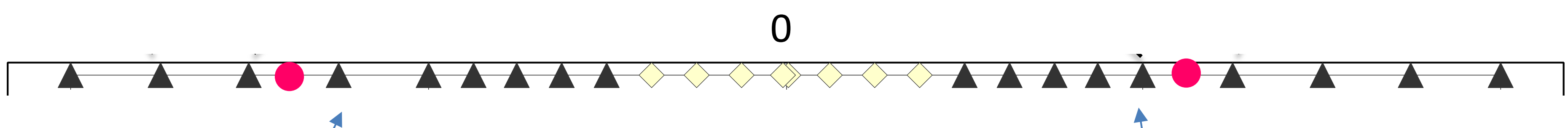
Round up rounds to the right

Round(x) = x_+ $(x_+ >= x)$

Round down rounds to the left

Round(x) = x_- $(x_- <= x)$

Round towards zero



Rounds to the right if $x < 0$

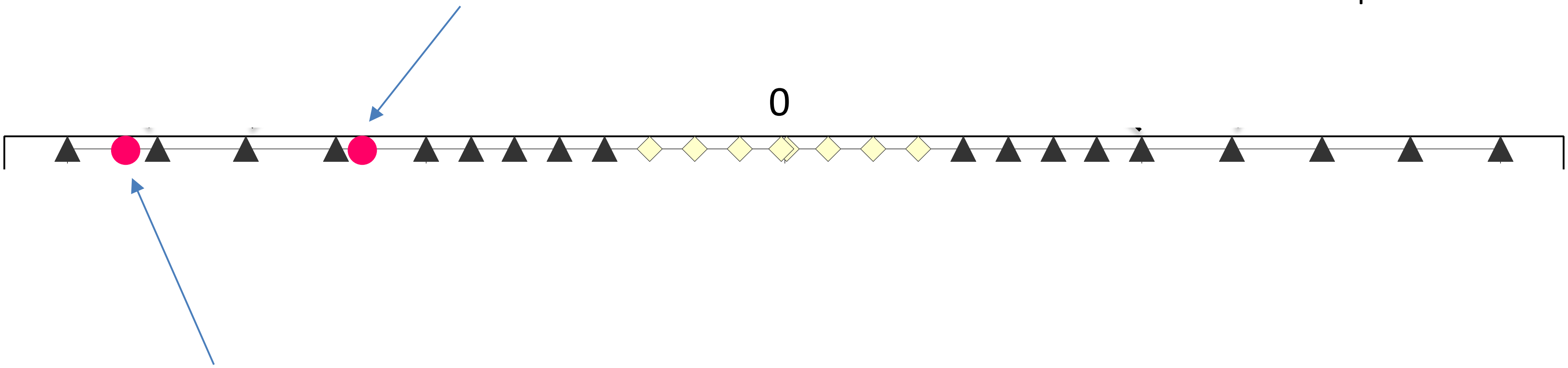
$\mathrm{Round}(x) = x_+ \quad \text{if } x < 0$

Rounds to the left if $x > 0$

$\mathrm{Round}(x) = x_- \text{ if } x < 0$

Round to nearest; ties to even

Round to the left if x_- is nearer to x than x_+



Round to the right if x_+ is nearer to x than x_-

In case of a tie, the one with its least significant bit equal to zero is chosen.

How does CPU know if some 4-byte value should be interpreted as IEEE FP or integers?

CPU uses separate registers for floating point and ints.

CPU uses different instructions for floating points and int operations.

Floating Point lesson plan

- Scientific notation
- FP8 example
- IEEE FP standard
- Rounding
- FP operations caveats

Floating point operations

- FP Caveats:
  - Invalid operation: 0/0, sqrt(-1), $\infty+\infty$
  - Divide by zero: $x/0 \rightarrow \infty$
  - Overflows: result too big to fit
  - Underflows: 0 < result < smallest denormalized value
  - Inexact: round it!
- FP addition: commutative but not always associative
- FP multiplication: commutative but not always associative and distributive

Floating point trouble

- Comparing floats for equality is a bad idea!

```
float f = 0.1;
while (f != 1.0) {
        f += 0.1;
}
```

```
f=0.2000000030
f=0.3000000119
f=0.4000000060
f=0.5000000000
f=0.6000000238
f=0.7000000477
f=0.8000000715
f=0.9000000954
f=1.0000001192
f=1.1000001431
f=1.2000001669
f=1.3000001907
f=1.4000002146
f=1.5000002384
f=1.6000002623
```

Floating point trouble

- FP is not associative: the order of operations affects results
  - $(a + b) + c \neq a + (b + c)$

```
0.1+1e20 - 1e20
>>> 0
0.1 + (1e20-1e20)
>>> 0.1
```

```
import random

vals = [1e-10, 1e-5, 1e-2, 1]
vals = vals + [-v for v in vals]

results = []
random.seed(42)
for _ in range(10000):
    random.shuffle(vals)
    results.append(sum(vals))

results = sorted(set(results))
print(f"There are {len(results)} unique results: {results}")

# Output:
# There are 102 unique results: [-8.326672684688674e-17, -7.45931094670027e-17, ..
```

Horace He, “Defeating Nondeterminism in LLM Inference

FP point trouble

- Many real world disasters are due to FP trickiness
  - Patriot Missile failed to intercept due to rounding error (1991)
  - Ariane 5 explosion due to overflow in converting from double to int (1996)

Floating point summary

- FP format is based on binary scientific notation
- IEEE FP format
  - Normalized, denormalized, special values
- Floating points are tricky
  - Precision diminishes as magnitude grows
  - overflow, rounding error