

Floating point

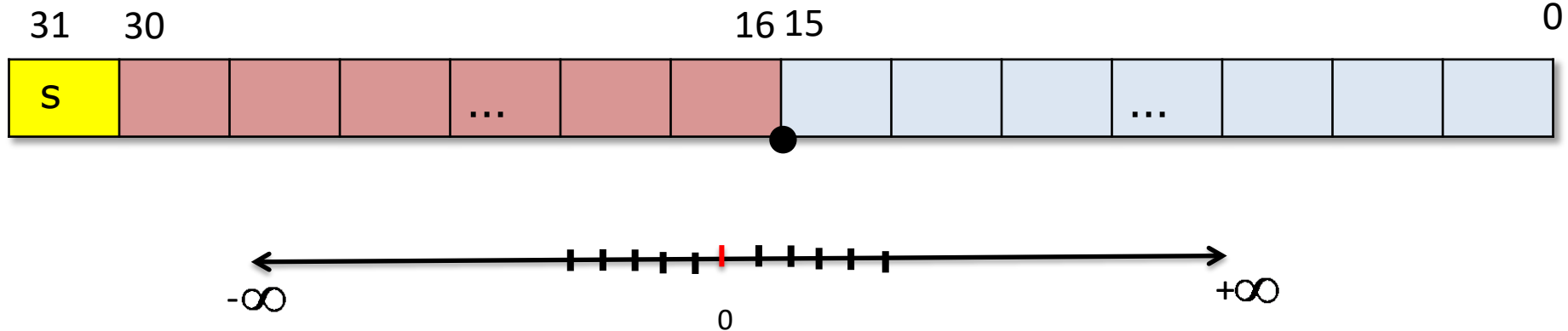
Jinyang Li

Floating Point (FP) lesson plan

- Normalized binary exponential notation
- Strawman 32-bit FP
- IEEE FP format
- Rounding

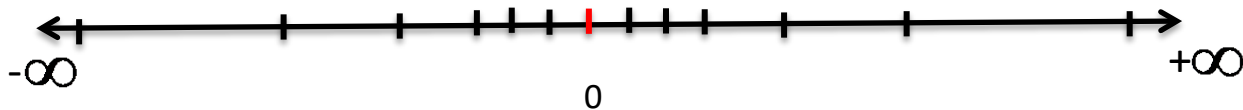
Floating point: key idea

Previously, fixed point



Fixed Point limitation: hard tradeoff between range and precision

- Idea: How about un-even spacing between numbers?



Floating Point: decimal

Based on exponential notation (aka normalized scientific notation)

$$r_{10} = \pm M * 10^E, \text{ where } 1 \leq M < 10$$

M: significant (mantissa), E: exponent

Floating Point: decimal

Example:

$$365.25 = 3.6525 * 10^2$$

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



Decimal point **floats** to the position immediately after the first nonzero digit.

Floating Point: binary

Binary exponential representation

$\pm M * 2^E$, where $1 \leq M < 2$

$$M = (1.b_1b_2b_3\dots b_n)_2$$

M: significant, E: exponent

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

Floating Point

Binary exponential representation

$$\pm M * 2^E, \text{ where } 1 \leq M < 2$$

$$M = (1.b_1b_2b_3...b_n)_2$$

M: significant, E: exponent



Also called normalized
representation

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



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



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

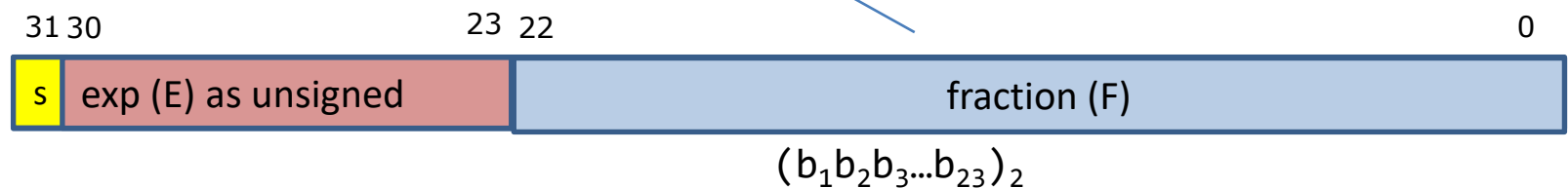
Answer: $(10.25)_{10} = (1010.01)_2 = (1.01001)_2 * 2^3$

How to represent a normalized number in a fixed-length format?

$\pm M * 2^E$, where $1 \leq M < 2$

$$M = (1.b_1b_2b_3...b_{23})_2$$

Strawman 32-bit FP representation:



Normalized representation

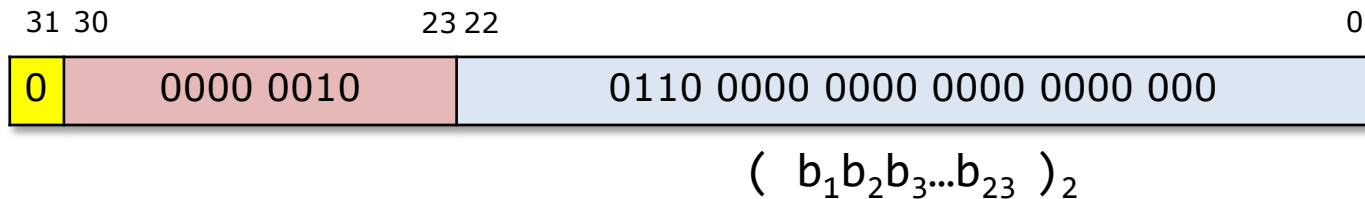
$$\pm M * 2^E, \text{ where } 1 \leq M < 2$$

↙ significant
↘ exponent

$$M = (1.b_1b_2b_3...b_{23})_2$$

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

Strawman 32-bit FP representation:



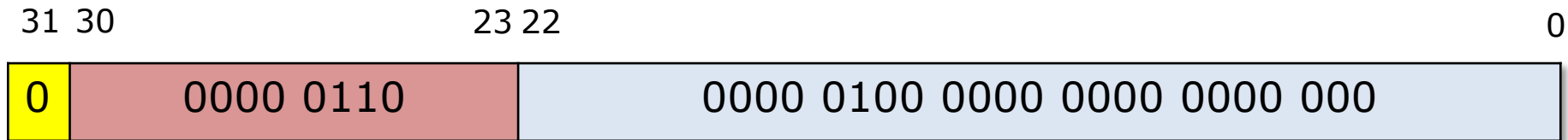
Exercise



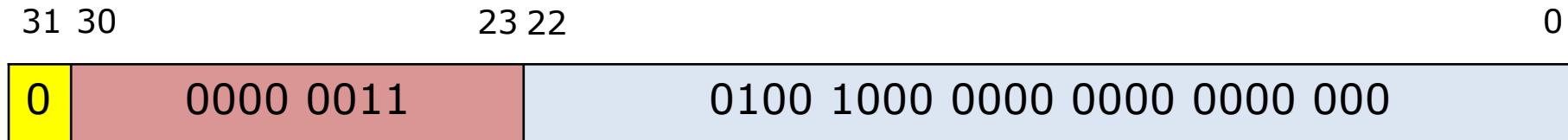
The strawman 32-bit FP representation of $(65)_{10}$?

Exercise

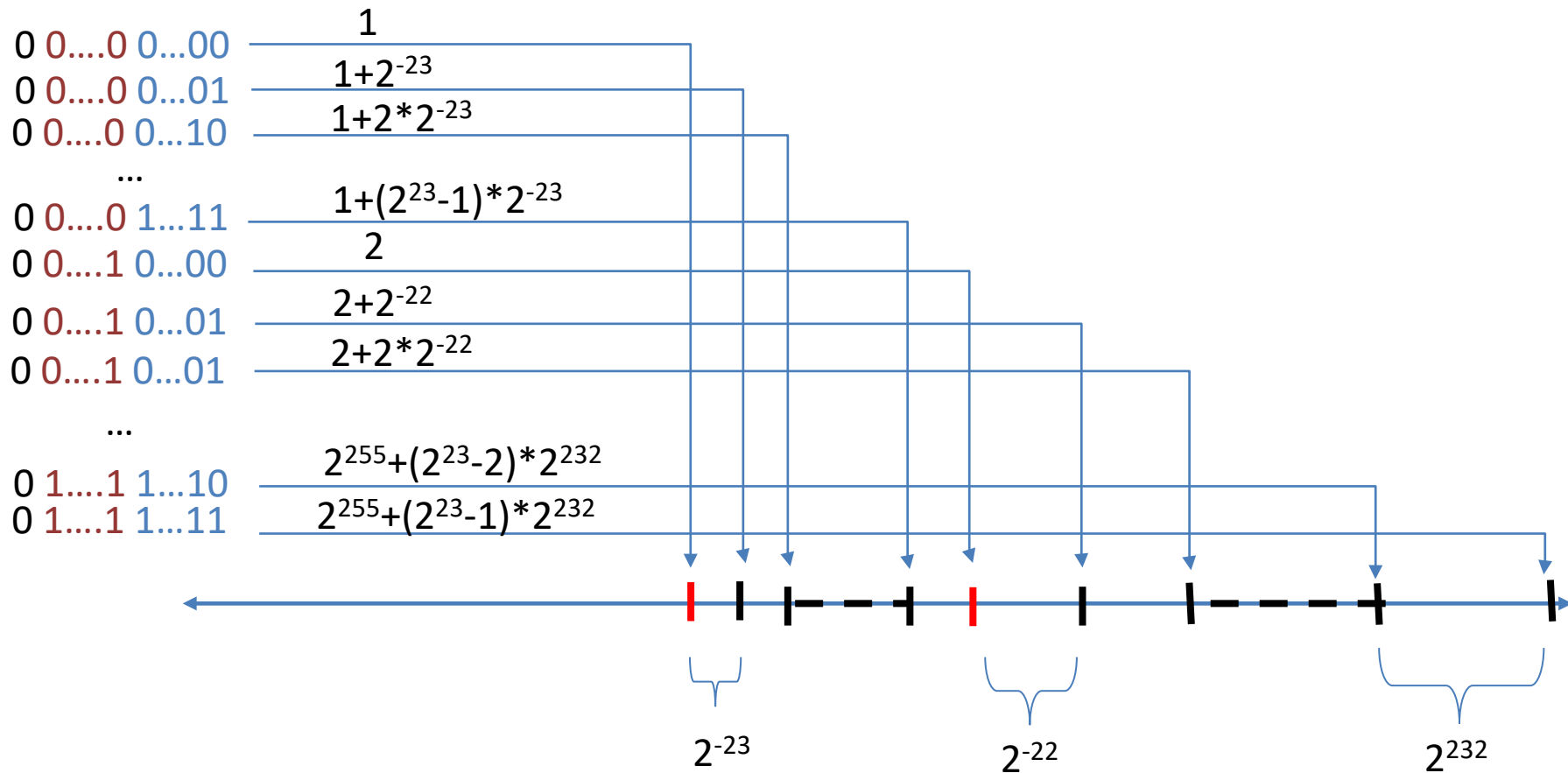
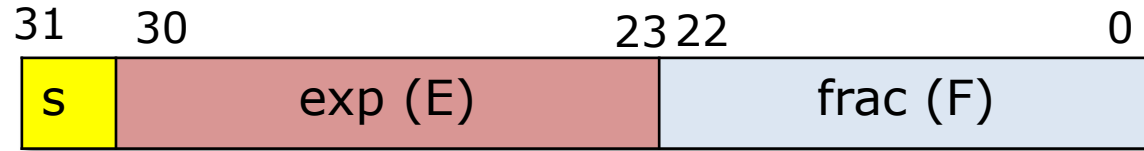
Example: $(65)_{10} = (1000001)_2 = (1.000001)_2 * 2^6$



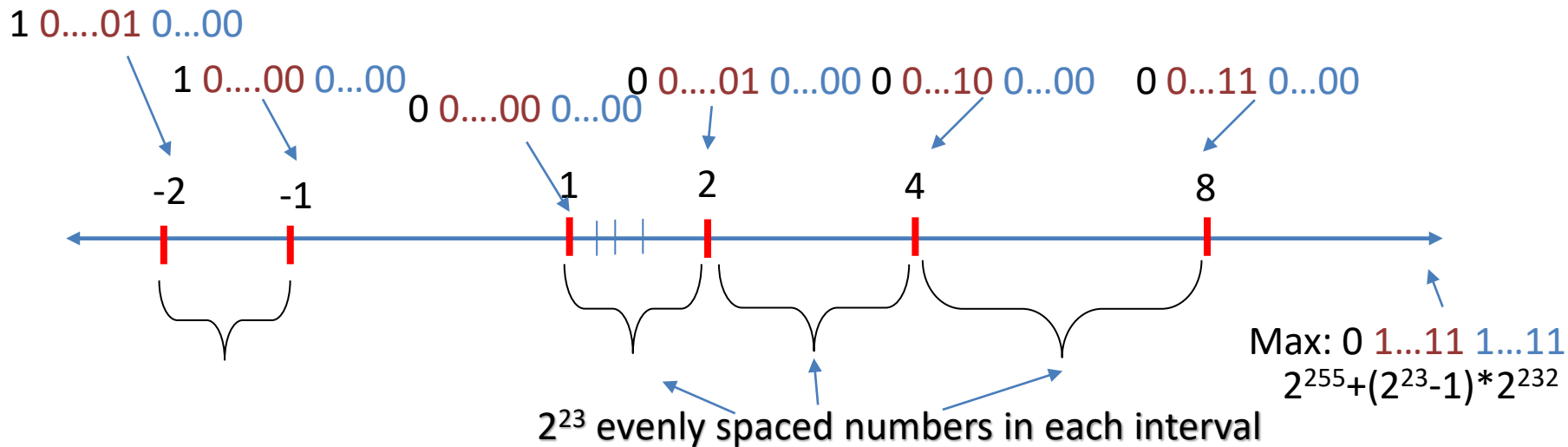
Another example: $(10.25)_{10} = (1010.01)_2 = (1.01001)_2 * 2^3$



Strawman 32-bit FP



Strawman 32-bit FP



- The good 👍
 - Large range $[1, 2^{255} + (2^{23}-1) \cdot 2^{232}]$, $[-2^{255} - (2^{23}-1) \cdot 2^{232}, -1]$
 - Allows easy comparison: compare FPs by bit patterns
- The bad 🙄
 - No 0!
 - No $[-1, 1]$
 - Max precision (2^{-23}) not high enough
 - No representation of special cases: ∞

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

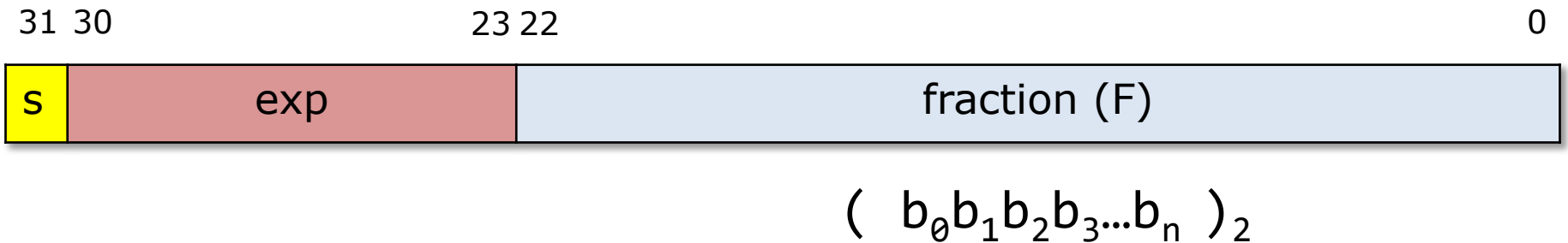
Courant Institute of Mathematical Sciences
New York University
New York, New York

Goals of IEEE Standard

- Consistent representation of floating point numbers
 - Address the limitation of our FP strawman
- Correctly rounded floating point operations, using several rounding modes.
- Consistent treatment of exceptional situations such as division by zero

IEEE FP: Carve out subsets of bit-patterns from normalized representation

$$\pm M \cdot 2^E \quad M = (1.b_0b_1b_2b_3...b_n)_2$$



For normalization representation,
exp can not be $(1111\ 1111)_2$ or $(0000\ 0000)_0$

$$\text{exp}_{\max} = ? \quad 254, \quad (1111 \ 1110)_2$$

$$\text{exp}_{\min} = ? \quad 1, (0000 \ 0001)_2$$

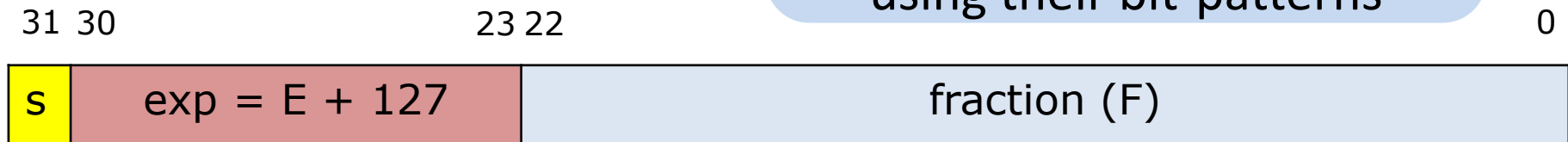
IEEE FP: Represent negative exponents using bias

$$\pm M * 2^E, \quad M = (1.b_0b_1b_2b_3...b_n)_2$$

To represent FPs in $(-1,1)$, we must allow negative exponent.

- How to represent negative E?
 - ~~2's complement~~
 - use bias

Why? Using bias instead of 2's complement allows simple comparison of FPs using their bit-patterns

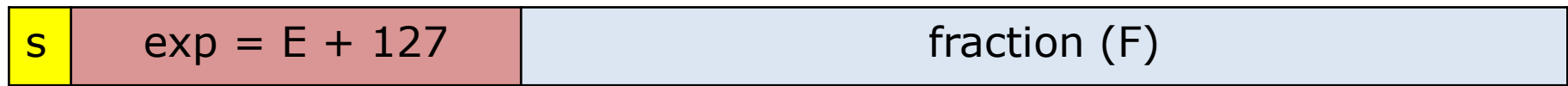


$$(b_0b_1b_2b_3...b_n)_2$$

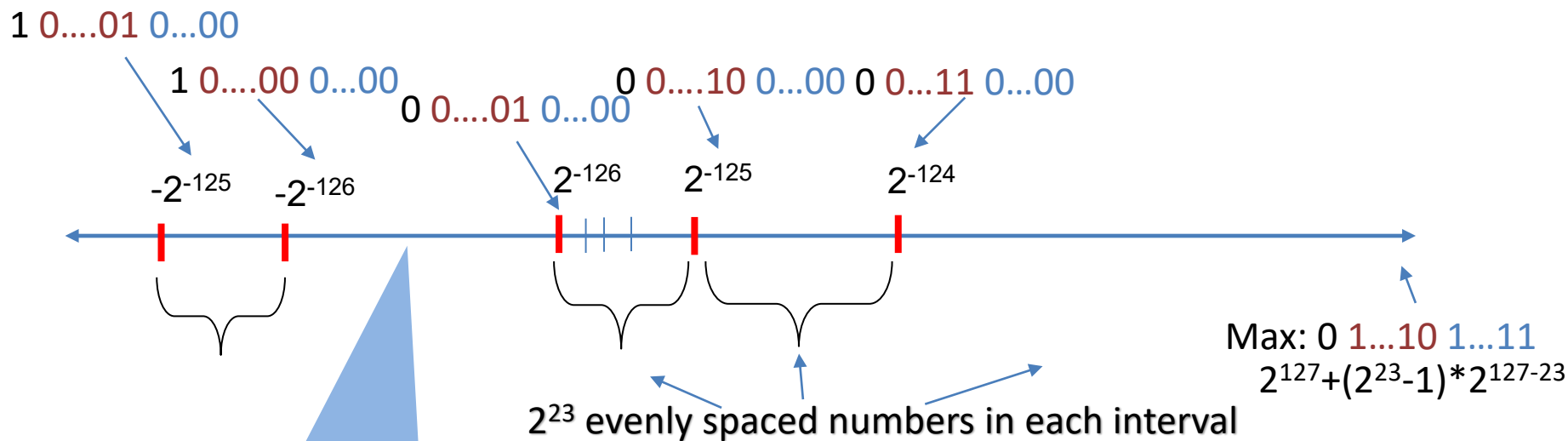
IEEE FP normalized representation

$$\pm M * 2^E, \quad M = (1.b_0b_1b_2b_3...b_n)_2$$

31 30
23 22
0



$$(b_0b_1b_2b_3...b_n)_2$$



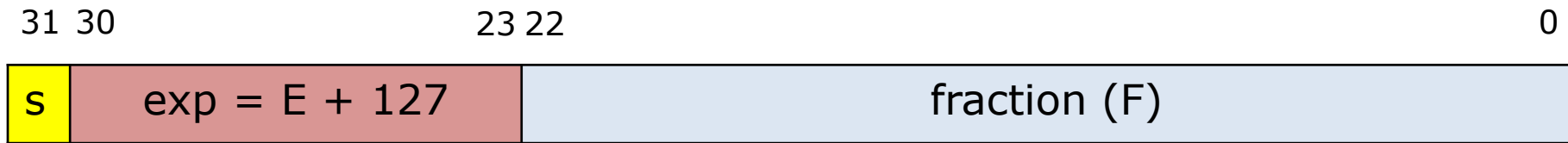
The gap $[-2^{-126}, 2^{-126}]$ is 2^{-125}

Represent values close and equal to 0

IEEE FP denormalized representation: represent values close and equal to 0

$$\pm M * 2^E$$

Normalized Encoding:



$$1 \leq M < 2, M = (1.F)_2$$

Denormalized Encoding:

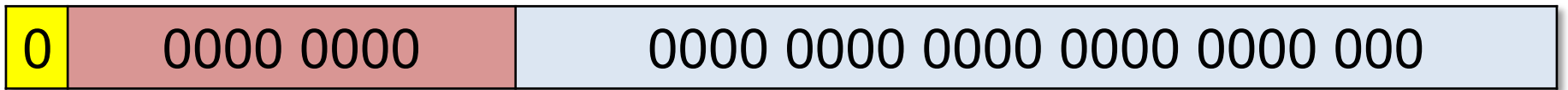


$$E = 1 - \text{Bias} = -126$$

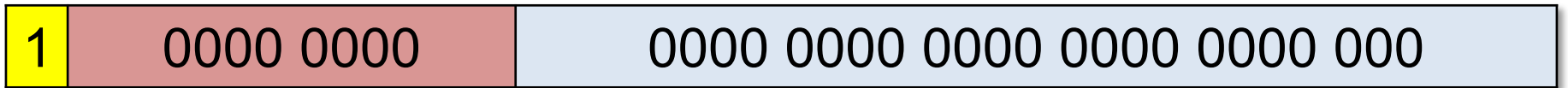
$$0 \leq M < 1, M = (0.F)_2$$

Zeros

+0.0



-0.0

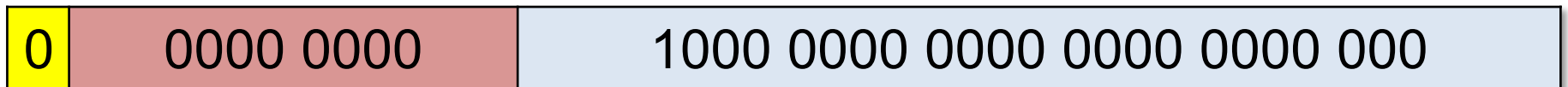


Denormalized FP example

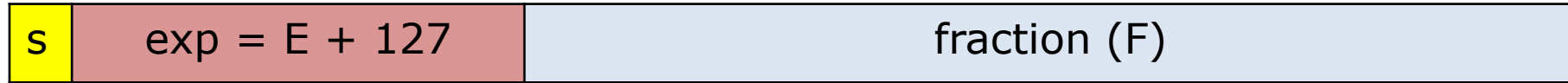
Smaller than the smallest E (-126)
of normalized encoding

What's the IEEE FP format of $(1.0)_2 * 2^{-127}$?

$$(1.0)_2 * 2^{-127} = (0.1)_2 * 2^{-126}$$



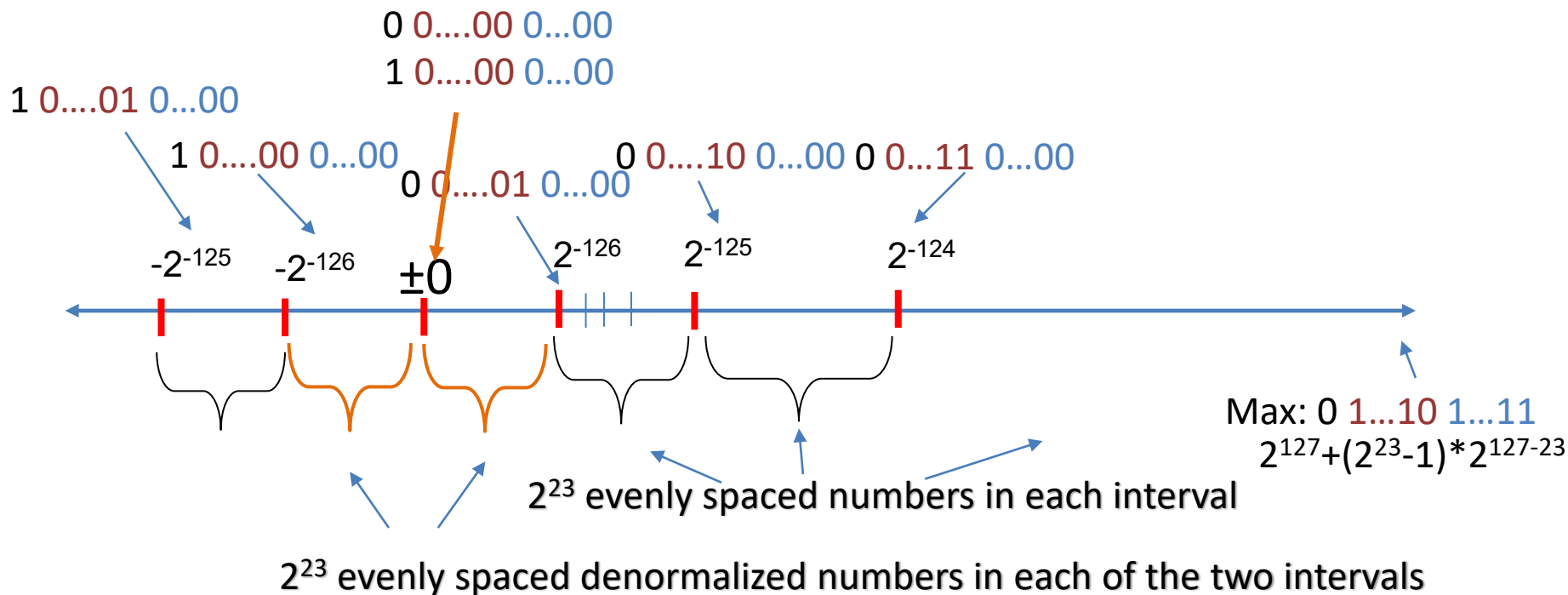
IEEE FP normalized + denormalized



$$1 \leq M < 2, M = (1.F)_2$$



$$0 \leq M < 1, M = (0.F)_2$$



IEEE FP: special values

Special Value's Encoding:



values	sign	frac
$+\infty$	0	all zeros
$-\infty$	1	all zeros
NaN	any	non-zero

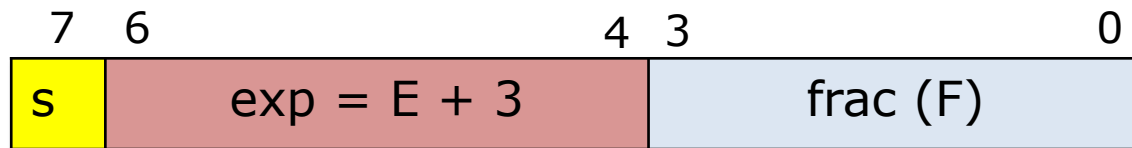


Breakout time!

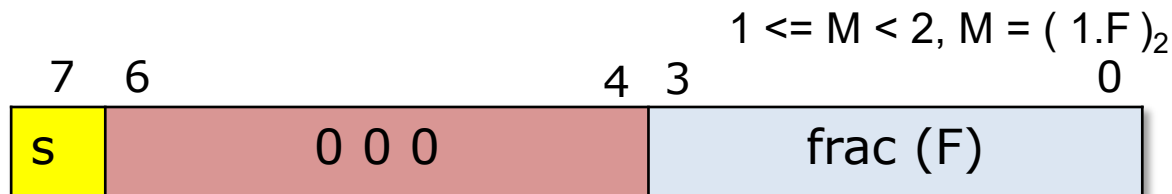
A toy 8-bit FP in the spirit of IEEE FP

$$\pm M * 2^E$$

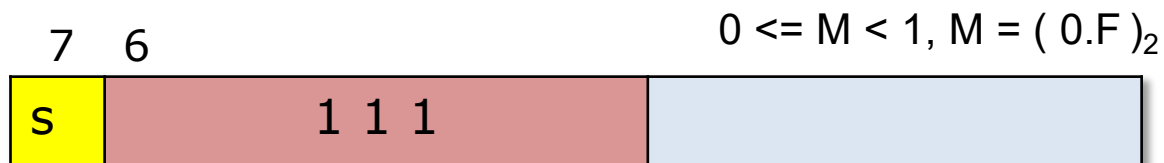
- exponent: 3 bits
- fraction: 4 bits
- **bias: 3**



Normalized encoding
exp \neq 000, 111



Denormalized encoding
exp = 000



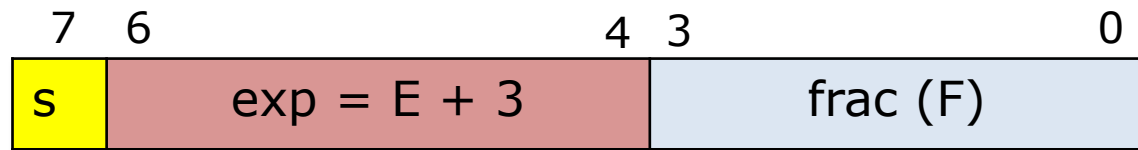
Special values encoding
exp = 111

- Smallest positive number?
- Range?
- How many distinct numbers?

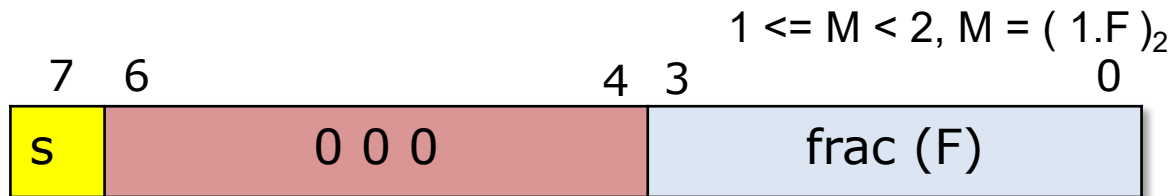
A toy 8-bit FP in the spirit of IEEE FP

$$\pm M * 2^E$$

- exponent: 3 bits
- fraction: 4 bits
- **bias: 3**

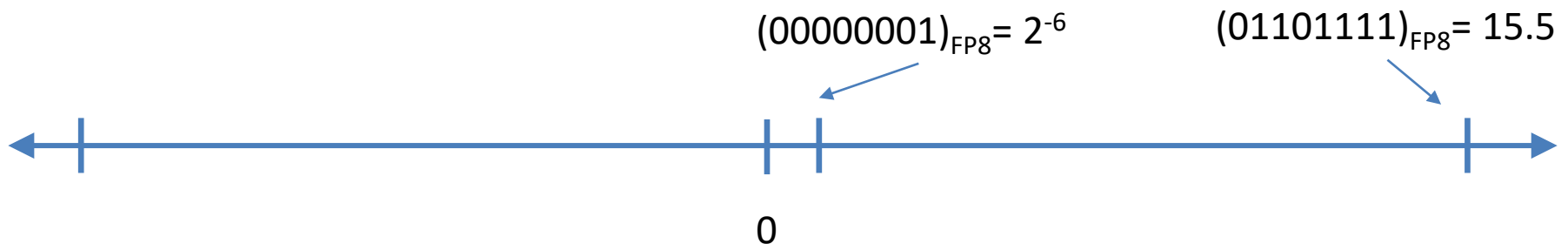


Normalized encoding
exp \neq 000, 111



Denormalized encoding
exp = 000 aka $E=1-3=-2$

$$0 \leq M < 1, M = (0.F)_2$$



$2^8 - 2^5 - 1$ distinct numbers: there are 2^8 total bit-patterns, 2^5 special values, 0 has 2 bit-patterns.

Floating Point (FP) lesson plan

- Normalized binary exponential notation
- Strawman 32-bit FP
- IEEE FP format
- Rounding

FP: Rounding



Values that are represented precisely

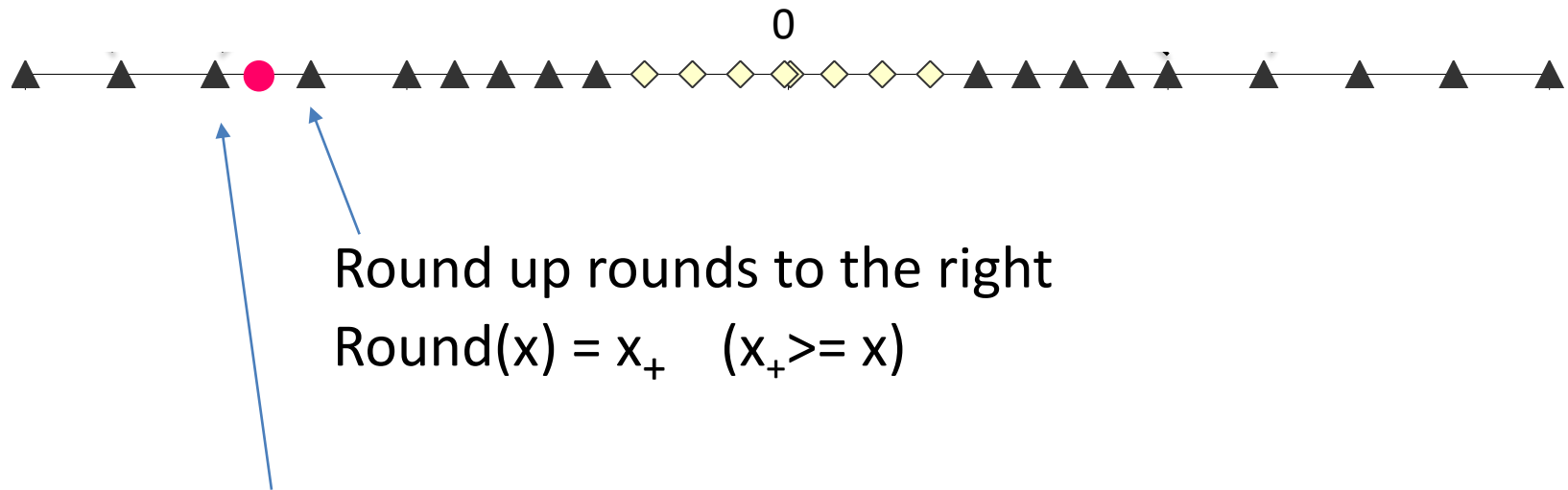
What if the result of computation is at ● ?

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)

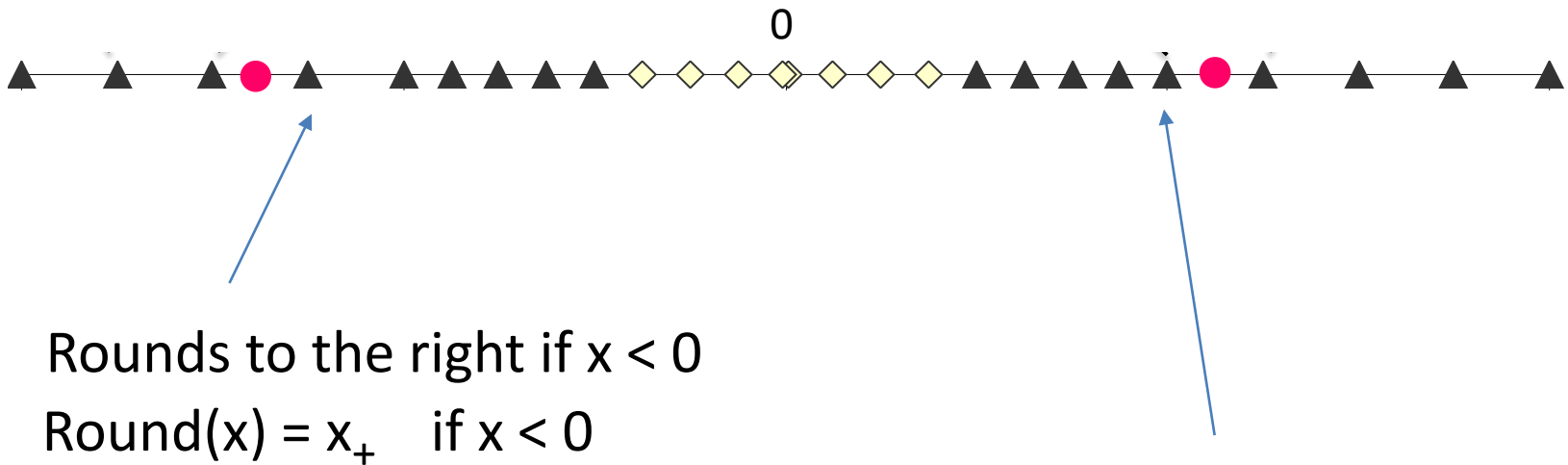
Round down vs. round up



Round up rounds to the right
 $\text{Round}(x) = x_+ \quad (x_+ \geq x)$

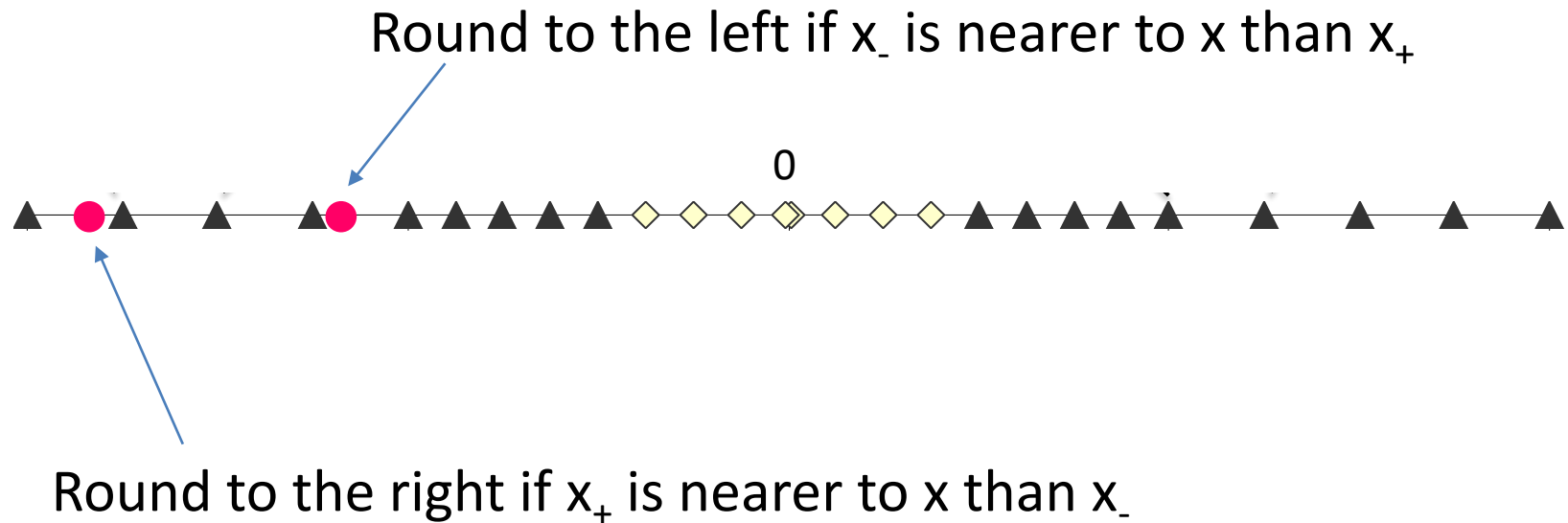
Round down rounds to the left
 $\text{Round}(x) = x_- \quad (x_- \leq x)$

Round towards zero



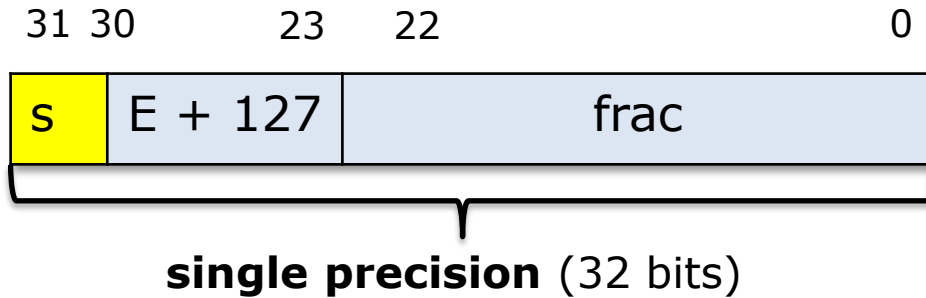
Rounds to the left if $x > 0$
 $\text{Round}(x) = x_-$ if $x > 0$

Round to nearest; ties to even

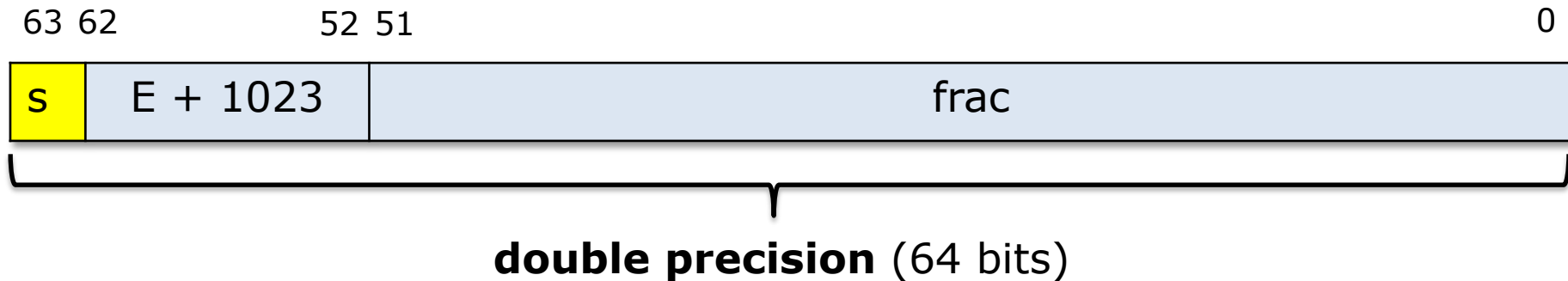


In case of a tie, the one with its least significant bit equal to zero is chosen.

IEEE FP: single vs. double precision



float f = 0.1
double d = 0.1



single/ double precision

	E_{\min}	E_{\max}	N_{\min}	N_{\max}
Float	-126	127	2^{-149}	$\approx 2^{128}$
Double	-1022	1023	2^{-1078}	$\approx 2^{1024}$

How does CPU know if it is floating point or integers ?

CPU has separate registers for floating point and integers.

CPU uses different instructions for floating points and integer operations.

FP summary

- FP representation is based on normalized binary exponential notation
- IEEE FP format
 - Normalized, denormalized, special values
- Precision gets higher as numbers approach 0
- Rounding

Floating point operations

- Addition, subtraction, multiplication, division etc.
- FP Caveats:
 - Invalid operation: $0/0$, $\text{sqrt}(-1)$, $\infty + \infty$
 - Divide by zero: $x/0 \rightarrow \infty$
 - Overflows: result too big to fit
 - Underflows: $0 < \text{result} < \text{smallest denormalized value}$
 - Inexact: round it!

Floating point addition

- Commutative? $x+y == y+x$?
- Associative? $(x+y)+z = x + (y+z)$?
 - Rounding:
$$(3.14+1e10) - 1e10 = 0$$
$$3.14 + (1e10 - 1e10) = 3.14$$
 - Overflow
- Every number has an additive inverse?
 - Yes except for ∞ and NaN

Floating point multiplication

- Commutative? $x * y == y * x$?
- Associative? $(x * y) * z = x * (y * z)$?
 - Overflow:
 $(1e20 * 1e20) * 1e-20 = \text{inf}, 1e20 * (1e20 * 1e-20) = 1e20$
 - Rounding
- $(x + y) * z = x * z + y * z$?
 - $1e20 * (1e20 - 1e20) = 0.0, 1e20 * 1e20 - 1e20 * 1e20 = \text{NaN}$

Floating point in real world

- Storing time in computer games as a FP?
- Precision diminishes as time gets bigger

FP value (decimal)	Time value	FP precision	Time precision
1	1 sec	1.19E-07	119 nanoseconds
100	~1.5 min	7.63E-06	7.63 microseconds
10 000	~3 hours	0.000977	.976 milliseconds
1000 000	~11 days	0.0625	62.5 milliseconds

Floating point in the real world

- Using floating point to measure distances

FP value	Length	FP precision	Precision size
1	1 meter	1.19E-07	Virus
100	100 meter	7.63E-06	red blood cell
10 000	10 km	0.000977	toenail thickness
1000 000	.16x earth radius	0.0625	credit card width

Table source: Random ASCII

Floating point trouble

- Comparing floats for equality is a bad idea!

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

Floating point trouble

- Never count using floating points

```
count = 0;
for (float f = 0.0; f < 1.0; f += 0.1) {
    count++;
}
```

Floating point summary

- FP format is based on normalized exponential notation
- Floating points are tricky
 - Precision diminishes as magnitude grows
 - overflow, rounding error
- Many real world disasters 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)

