

Floating point

Jinyang Li

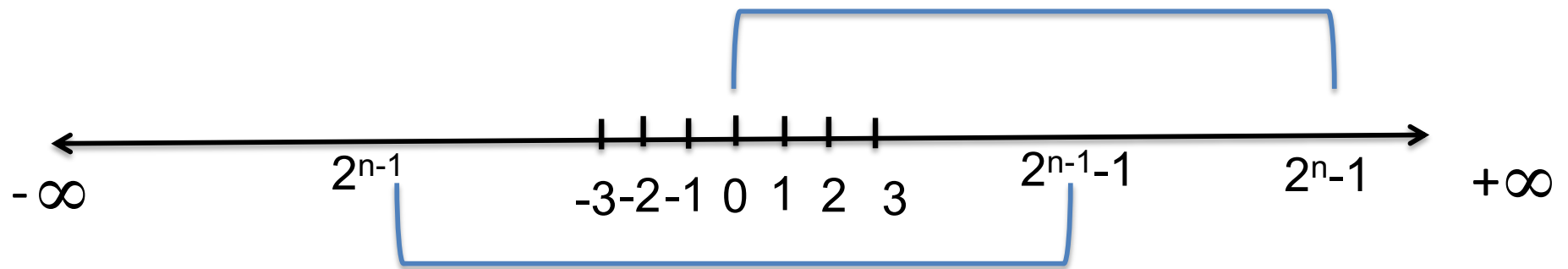
based on Tiger Wang's slides

Representing Real Numbers using bits

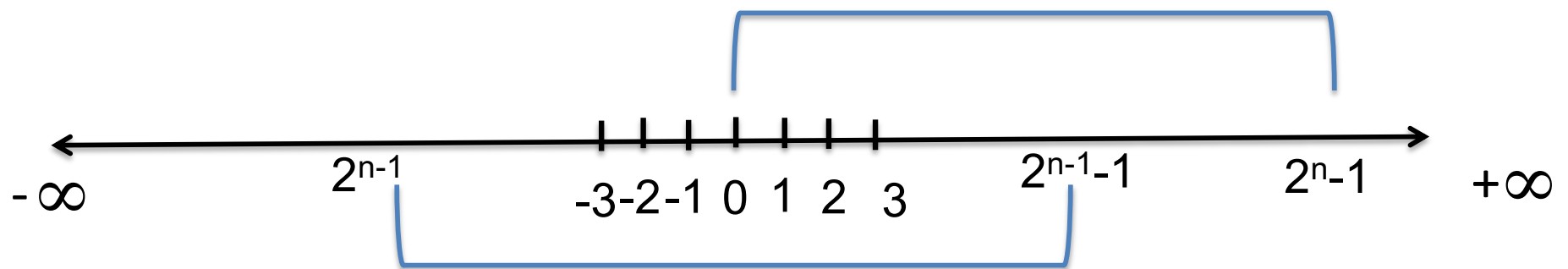


Representing Real Numbers using bits

What we have studied



Representing Real Numbers using bits



Today: How to represent fractional numbers?

Decimal Representation

Real Numbers	Decimal Representation (Expansion)
$11 / 2$	$(5.5)_{10}$
$1 / 3$	$(0.3333333...)_{10}$
$\sqrt{2}$	$(1.4128...)_{10}$

Decimal Representation

Real Numbers	Decimal Representation (Expansion)
--------------	------------------------------------

$11 / 2$	$(5.5)_{10}$
----------	--------------

$1 / 3$	$(0.3333333...)_{10}$
---------	-----------------------

$\sqrt{2}$	$(1.4128...)_{10}$
------------	--------------------

$$(5.5)_{10} = 5 * 10^0 + 5 * 10^{-1}$$

$$(0.333333...)_{10} = 3 * 10^{-1} + 3 * 10^{-2} + 3 * 10^{-3} + \dots$$

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

Decimal Representation

Real Numbers	Decimal Representation (Expansion)
--------------	------------------------------------

$11 / 2$	$(5.5)_{10}$
----------	--------------

$1 / 3$	$(0.3333333...)_{10}$
---------	-----------------------

$\sqrt{2}$	$(1.4128...)_{10}$
------------	--------------------

$$(5.5)_{10} = 5 * 10^0 + 5 * 10^{-1}$$

$$(0.333333...)_{10} = 3 * 10^{-1} + 3 * 10^{-2} + 3 * 10^{-3} + \dots$$

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

$$r_{10} = (d_m d_{m-1} \dots d_1 d_0 \cdot d_{-1} d_{-2} \dots d_{-n})_{10}$$

$$= \sum_{i=-n}^m 10^i \times d_i$$

Binary Representation

$$\begin{aligned}(5.5)_{10} &= 4 + 1 + 1 / 2 \\ &= 1 * 2^2 + 1 * 2^0 + 1 * 2^{-1}\end{aligned}$$

Binary Representation

$$\begin{aligned}(5.5)_{10} &= 4 + 1 + 1 / 2 \\ &= 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1}\end{aligned}$$

Binary Representation

$$\begin{aligned}(5.5)_{10} &= 4 + 1 + 1 / 2 \\ &= 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1} \\ &= (101.1)_2\end{aligned}$$

Binary Representation

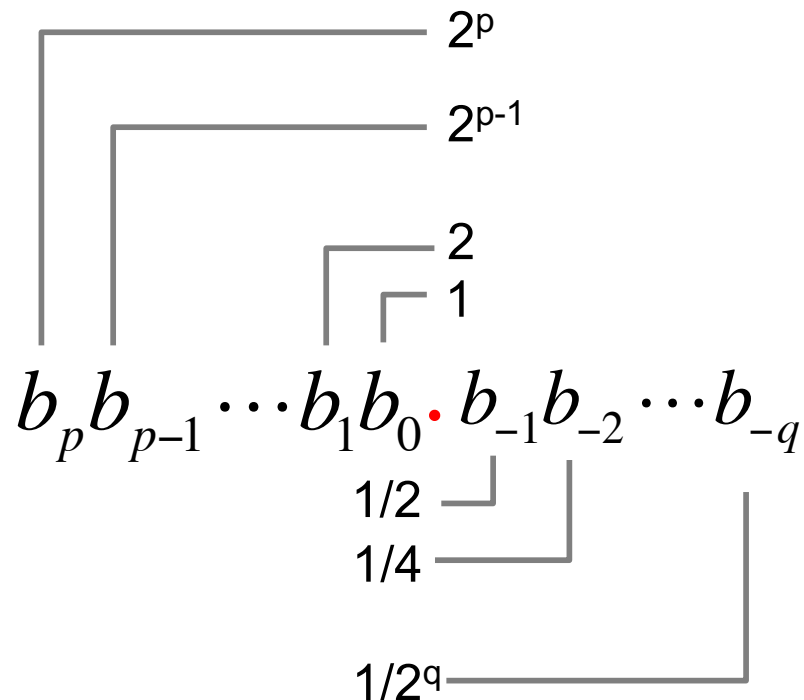
$$\begin{aligned}(5.5)_{10} &= 4 + 1 + 1 / 2 \\ &= 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1} \\ &= (101.1)_2\end{aligned}$$

$$\begin{aligned}(0.333333...)_{10} &= 1 / 4 + 1 / 16 + 1 / 64 + ... \\ &= (0.01010101...)_2\end{aligned}$$

Binary Representation

$$r_{10} = (d_m d_{m-1} d_1 d_0 \cdot d_{-1} d_{-2} \dots d_{-n})_{10}$$

$$= (b_p b_{p-1} b_1 b_0 \cdot b_{-1} b_{-2} \dots b_{-q})_2$$



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

Exercise

Binary
Expansion

10.011_2

Formula

Decimal

$$2^{-3} + 2^{-4} + 2^{-6}$$

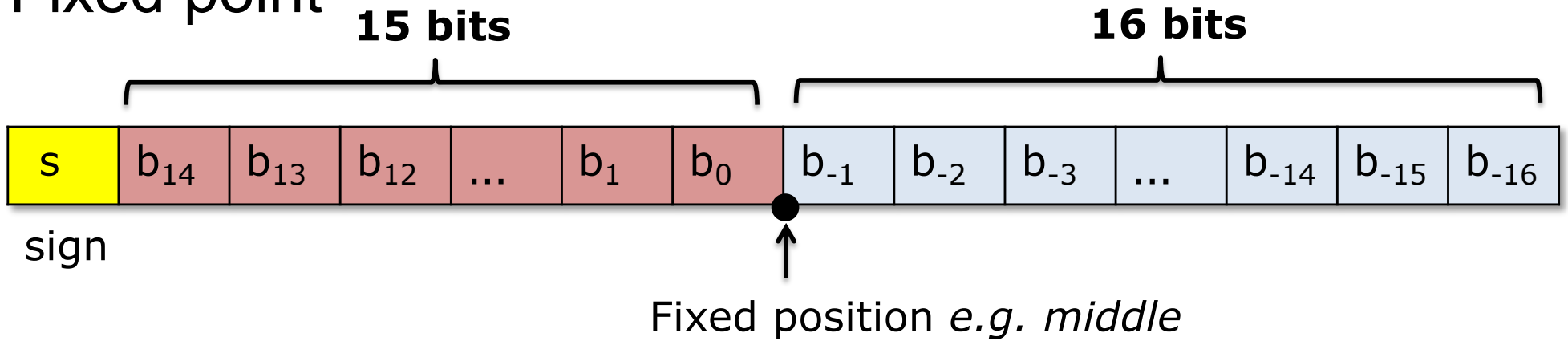
$$2^{-1} + 2^{-2} + 2^{-3} + 2^{-4}$$

Exercise

Binary Expansion	Formula	Decimal
10.011_2	$2^1 + 2^{-2} + 2^{-3}$	2.375_{10}
0.001101_2	$2^{-3} + 2^{-4} + 2^{-6}$	0.203125_{10}
0.1111_2	$2^{-1} + 2^{-2} + 2^{-3} + 2^{-4}$	0.9375_{10}

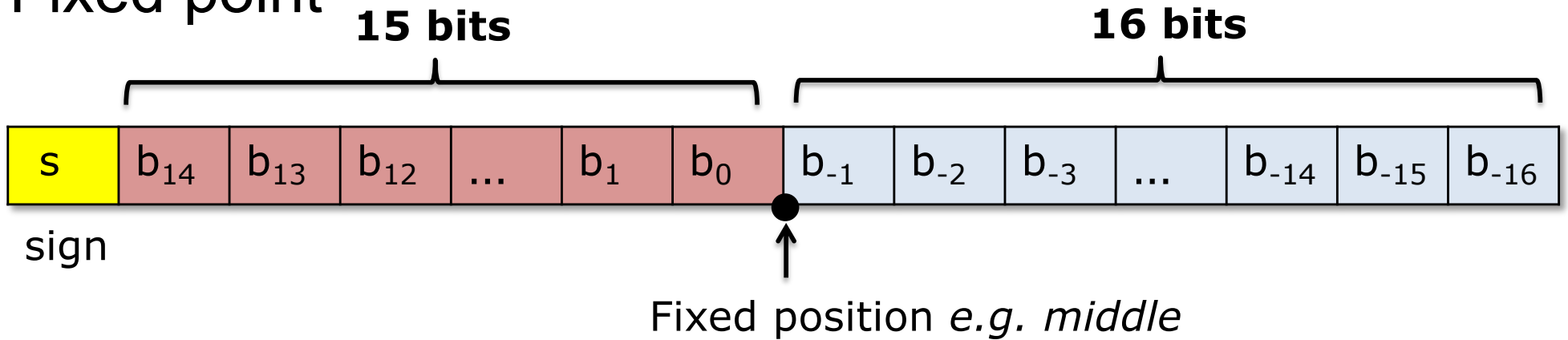
Intuitive Idea

Fixed point

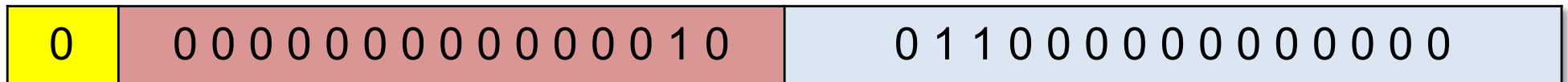


Intuitive Idea

Fixed point



$(10.011)_2$



Problems of Fixed Point

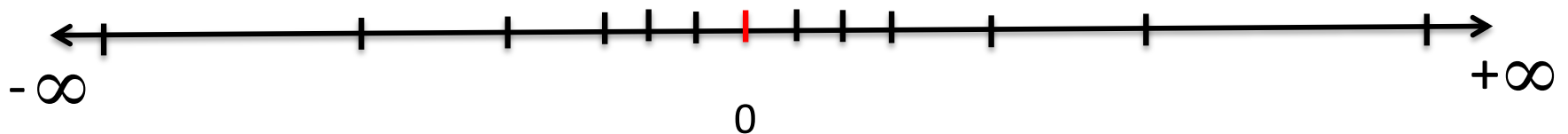
Limited range and precision: e.g., 32 bits

- Largest number: $2^{15} (011...111)_2$
- Highest precision: 2^{-16}

→ Rarely used (No built-in hardware support)

The idea

- Limitation of fixed point notation:
 - Represents evenly spaced fractional numbers
 - ➔ hard tradeoff between high precision and high magnitude
- 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

$$r_{10} = \pm M * 2^E, \text{ where } 1 \leq M < 2$$

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

M: significant, E: exponent

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

Floating Point

Binary exponential representation

$$r_{10} = \pm M * 2^E, \text{ where } 1 \leq M < 2$$
$$M = (1.b_1b_2b_3...b_n)_2$$

} Normalized representation of r

M: significant, E: exponent

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

Normalization: give a number r, obtain its normalized representation

Exercises

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

Exercises

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

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

Floating Point

Binary exponential representation

$$r_{10} = \pm M * 2^E, \text{ where } 1 \leq M < 2$$
$$M = (1.b_1b_2b_3...b_n)_2$$

} Normalized representation of r

M: significant, E: exponent

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

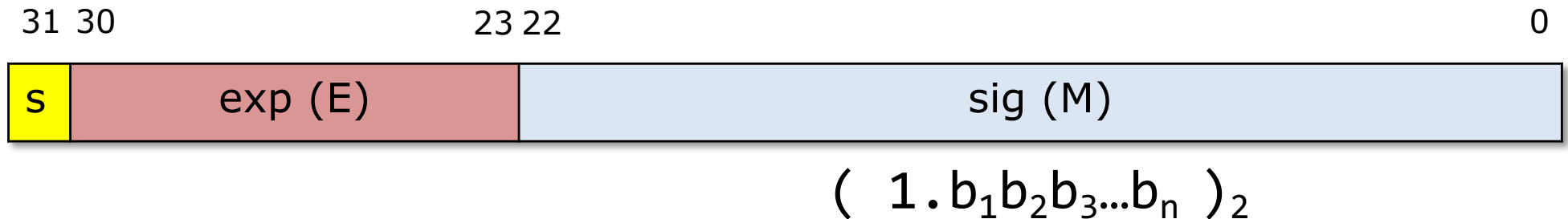
How to represent a normalized number in a fixed-length format?

Normalized representation

$$r_{10} = \pm M * 2^E, \text{ where } 1 \leq M < 2$$

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

M: significant, E: exponent



Normalized representation in computer

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

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

M: significant, E: exponent

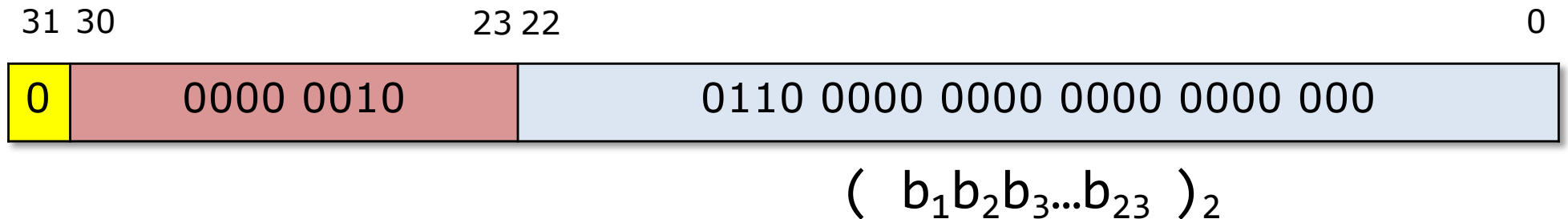


Normalized representation

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

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

M: significant, E: exponent



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

Exercise

Given the normalized representation of $(71)_{10}$ and $(10.25)_{10}$

Exercise

Given the normalized representation of $(71)_{10}$ and $(10.25)_{10}$

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

31 30

23 22

0

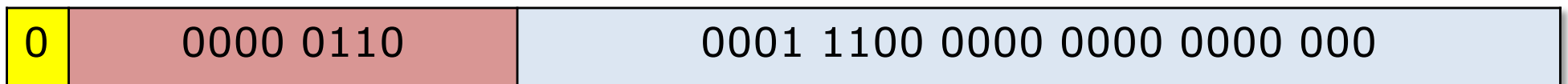


$$(71)_{10} = (1000111)_2 = (1.000111)_2 * 2^6$$

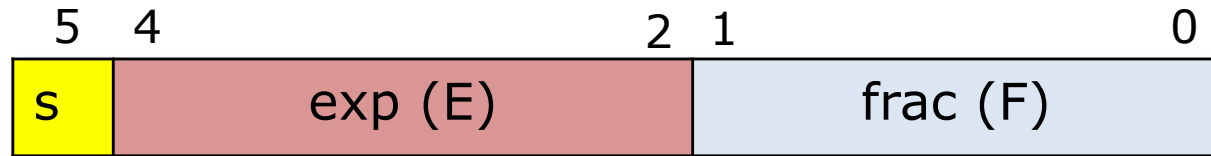
31 30

23 22

0



Toy Number System

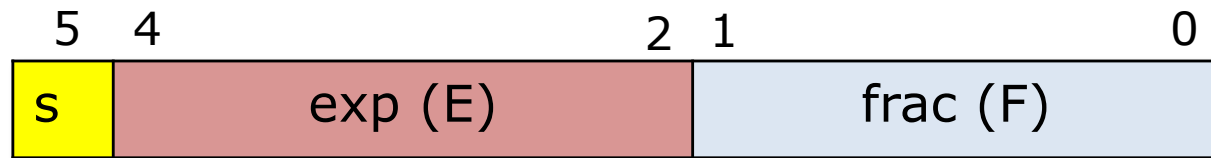


6-bit floating point representation

- exponent: 3 bits
- fraction: 2 bits

Largest positive number ?

Toy Number System



6-bit floating point representation

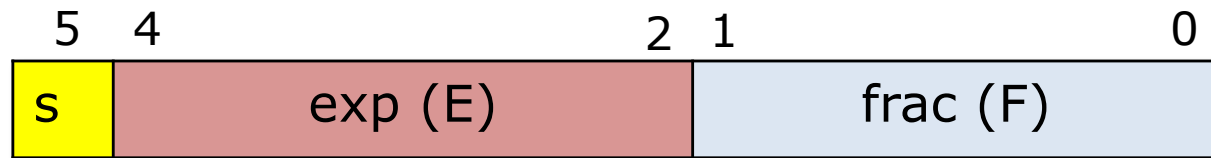
- exponent: 3 bits
- fraction: 2 bits

Largest positive number ?



$$(1.11)_2 * 2^7 = 224$$

Toy Number System



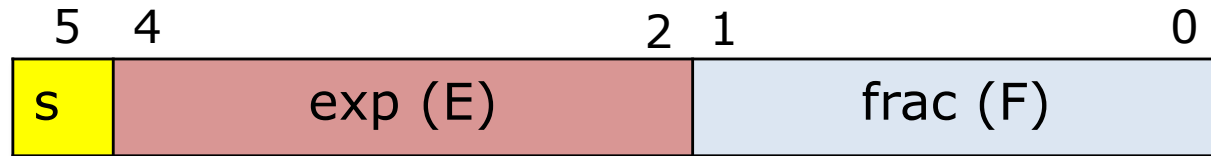
6-bit floating point representation

- exponent: 3 bits
- fraction: 2 bits

Largest positive number: 224

Smallest positive number ?

Toy Number System



6-bit floating point representation

- exponent: 3 bits
- fraction: 2 bits

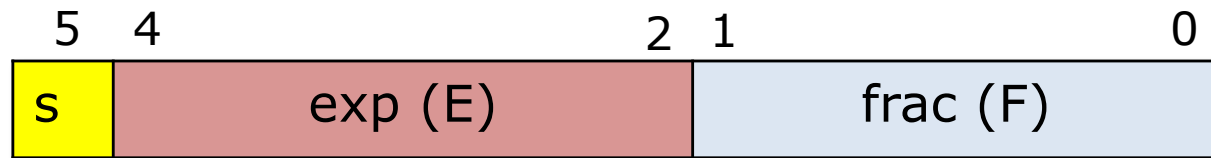
Largest positive number: 224

Smallest positive number: 1



$$(1.00)_2 * 2^0 = 1$$

Toy Number System



6-bit floating point representation

- exponent: 3 bits
- fraction: 2 bits

Positive number: 1 to 224

Negative number: -224 to -1



No more bit patterns
left to represent 0!

Questions

How to represent

1. numbers close or equal to 0?
2. special cases:
 - the result of dividing by 0, e.g. $1/0$?
 -

$$\infty * 0$$

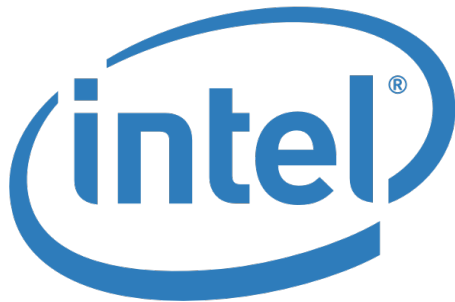
Lots of different implementations around 1950s!

IEEE Floating Point Standard



IEEE p754
A standard for binary
floating point representation

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



The Only Book Focuses On IEEE Floating Point Standard



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

hardware. This degree of altruism was so astonishing that MATLAB's creator Cleve Moler used to advise foreign visitors not to miss the country's two most awesome spectacles: the Grand Canyon, and meetings of IEEE p754."

<https://cs.nyu.edu/overton/NumericalComputing/protected/NumericalComputingSIAM.pdf>

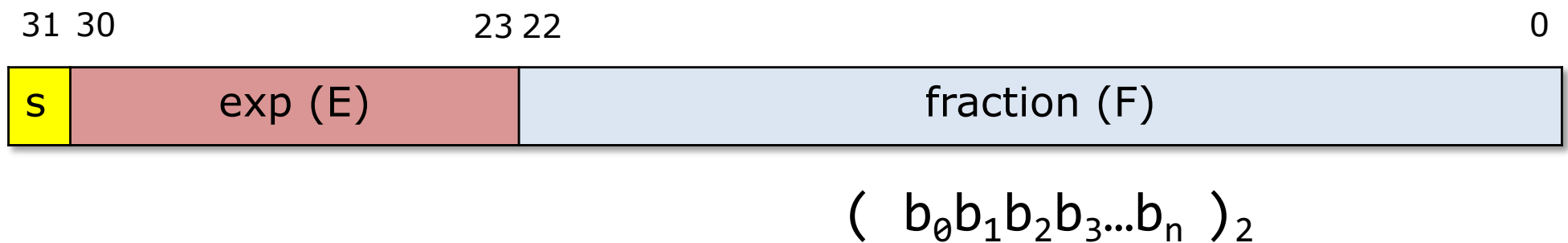
With you nyu netid/password. You can also search the pdf with google.

Goals of IEEE Standard

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

Restrictions on Normalized Representation

$$r_{1\theta} = \pm M * 2^E \quad M = (1.b_0b_1b_2b_3...b_n)_2$$



E can not be $(1111\ 1111)_2$ or $(0000\ 0000)_0$

$$E_{\max} = ? \quad 254, (1111\ 1110)_2$$

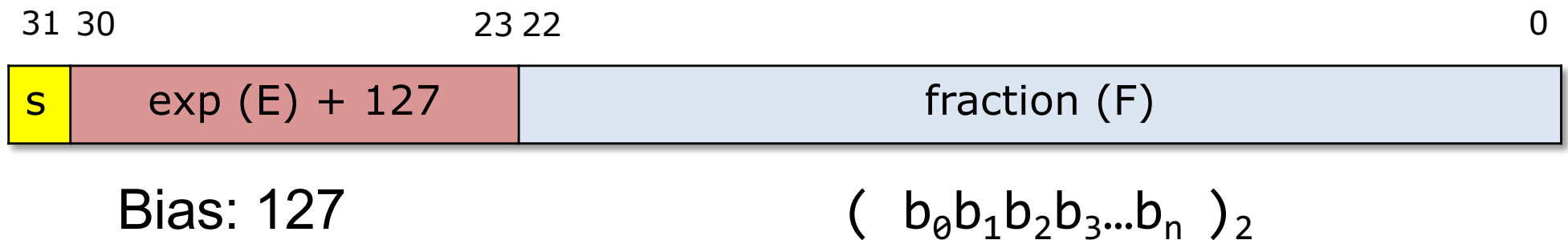
$$E_{\min} = ? \quad 1, (0000\ 0001)_2$$

Exponential Bias

$$r_{10} = \pm M * 2^E, \quad M = (1.b_0b_1b_2b_3...b_n)_2$$

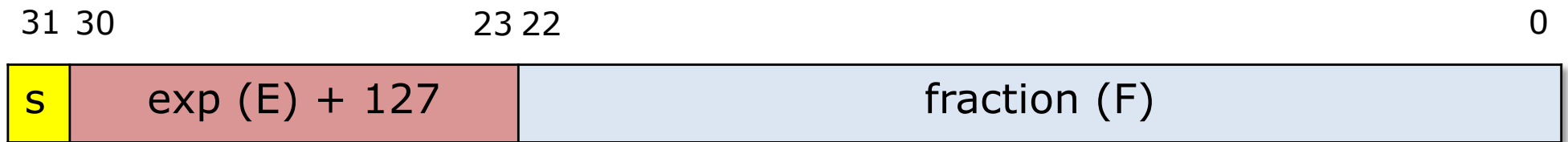
To represent (-1,1),
we must allow
negative exponent.

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



IEEE normalized representation

$$r_{10} = \pm M * 2^E, \quad M = (1.b_0b_1b_2b_3...b_n)_2$$



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

Bias: 127

$$E_{\max} = 254 - 127 = 127$$

Smallest positive number 2^{-126}

$$E_{\min} = 1 - 127 = -126$$

Negative number with smallest absolute value: -2^{-126}



Questions

Q1. Why using **bias**?

Q2. Why is **bias** 127?



Questions

Q1. Why using **bias** instead of 2's complement?

Answer: easier circuitry for comparison.

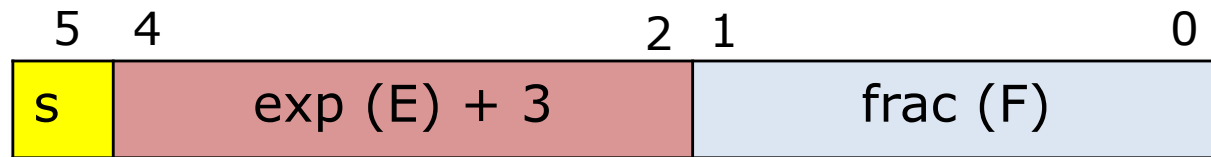


Questions

Q2. Why is bias 127?

A2. Balance positive numbers
(magnitude) and negative numbers
(precision)

Example Toy Number System

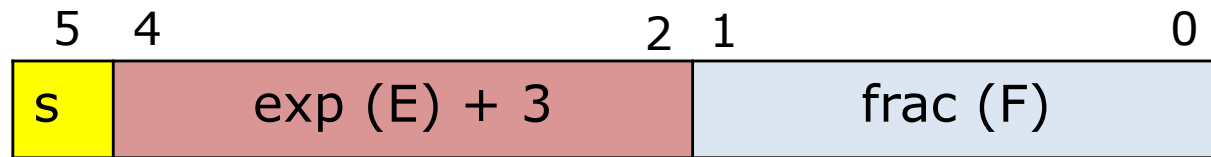


6-bit floating point representation

- exponent: 3 bits
- fraction: 2 bits
- **bias: 3**

Smallest positive number ?

Toy Number System



6-bit floating point representation

- exponent: 3 bits
- fraction: 2 bits
- **bias: 3**

Smallest positive number: 0.25

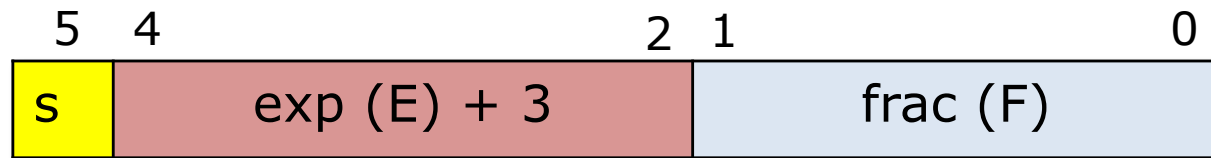
Smallest number >0.25?



$$(1.00)_2 * 2^{-2} = 0.25$$

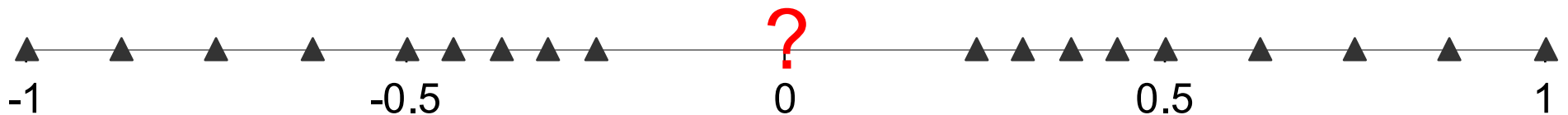
$$(1.01)_2 * 2^{-2} = 0.25 + 0.0625$$

Toy Number System



6-bit floating point representation

- exponent: 3 bits
- fraction: 2 bits
- **bias: 3**

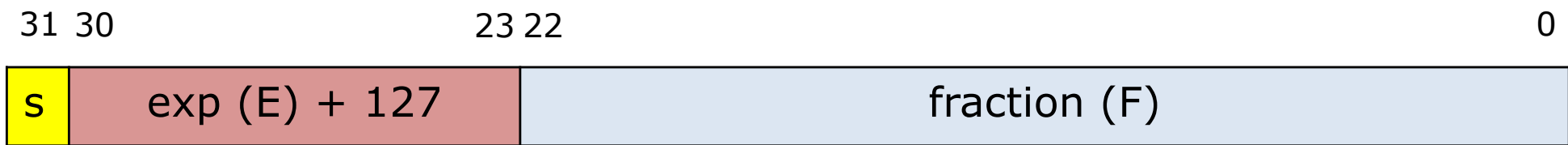


**represent values which are
close and equal to 0**

IEEE denormalized representation

$$r_{10} = \pm M * 2^E$$

Normalized Encoding:



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

Denormalized Encoding:

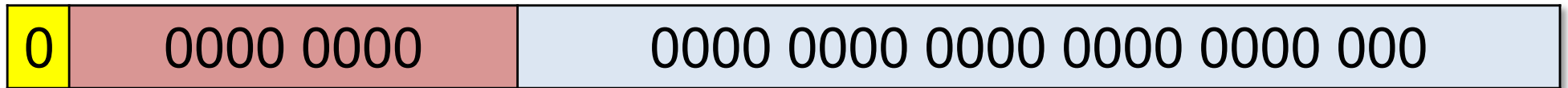


$$E = 1 - \text{Bias} = -126$$

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

Zeros

+0.0

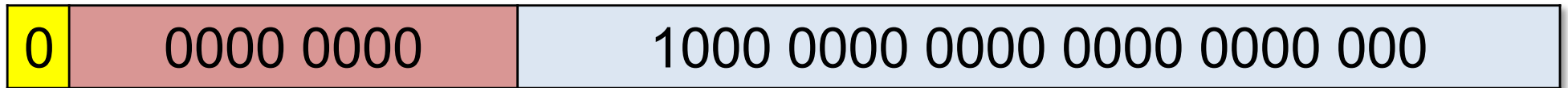


-0.0

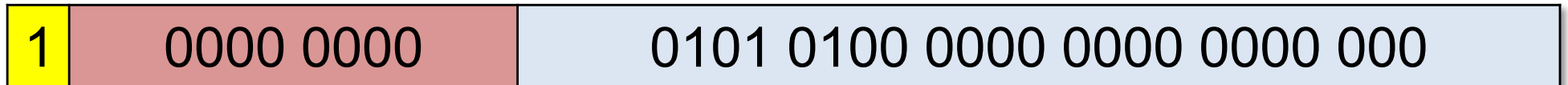


Examples

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



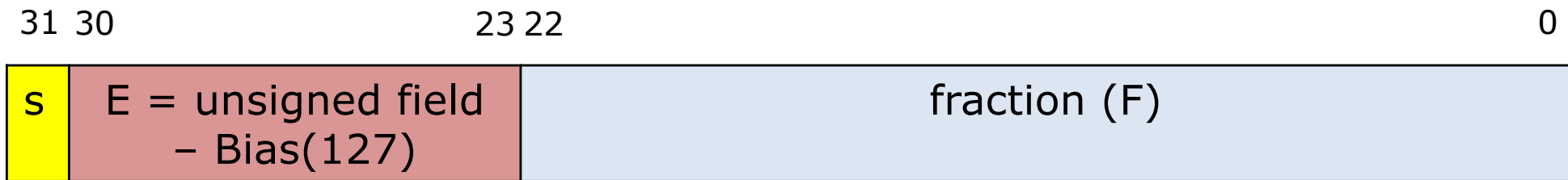
$$-(0.010101)_2 * 2^{-126}$$



What we've learnt last time

$$r_{10} = \pm M * 2^E$$

Normalized Encoding:



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

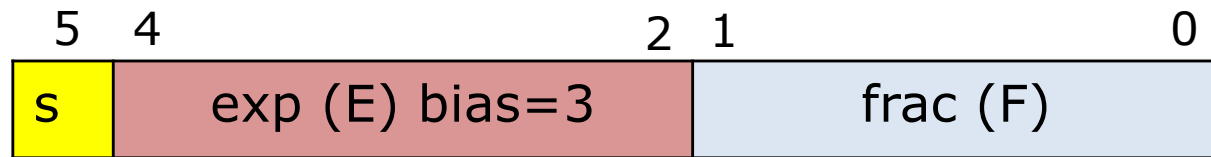
Denormalized Encoding:



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

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

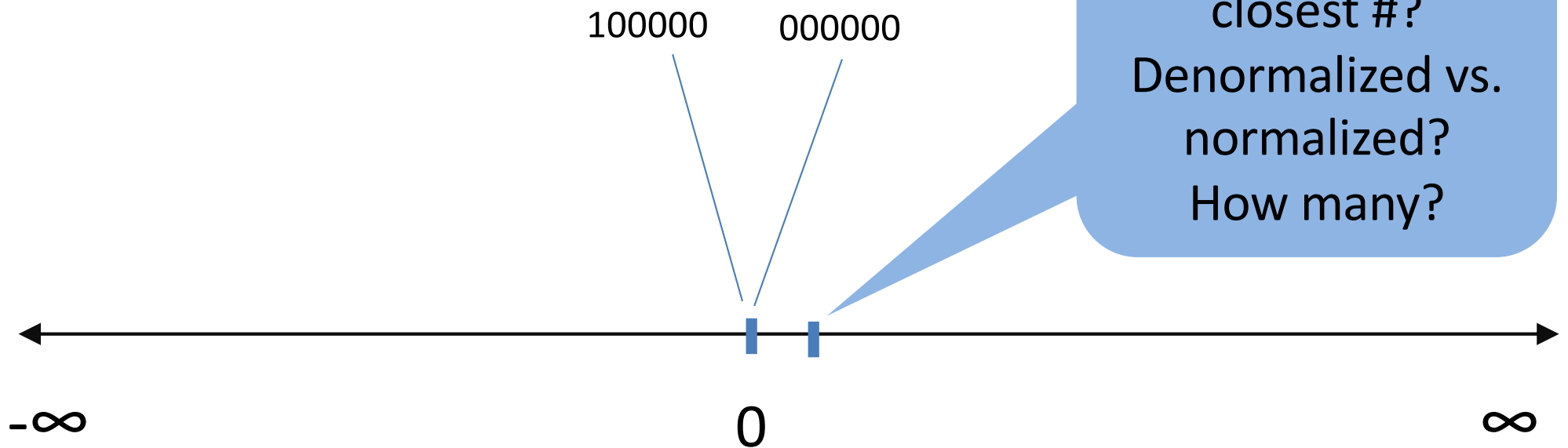
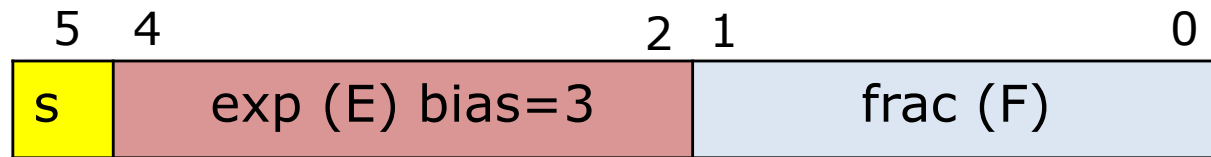
Toy Number System



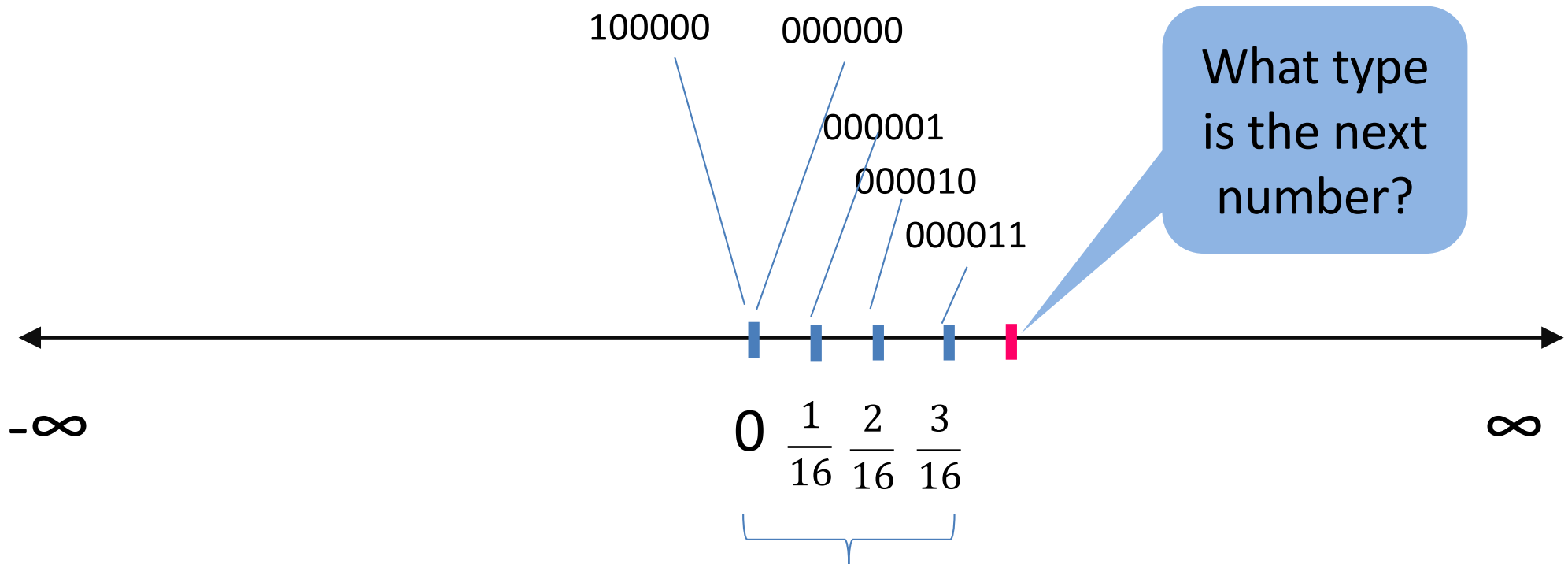
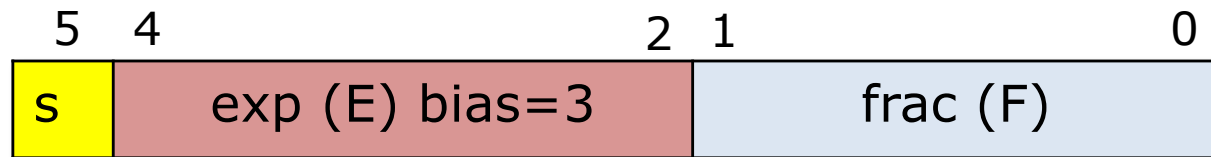
6-bit floating point representation

- exponent: 3 bits
- fraction: 2 bits
- bias: 3

Toy Number System

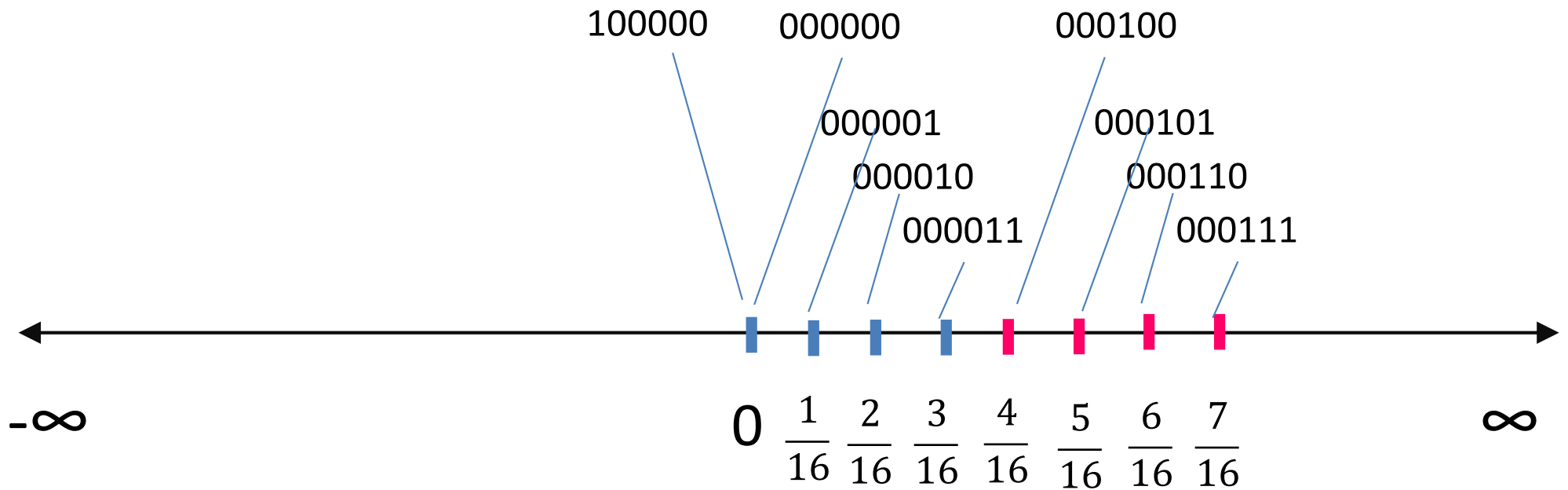
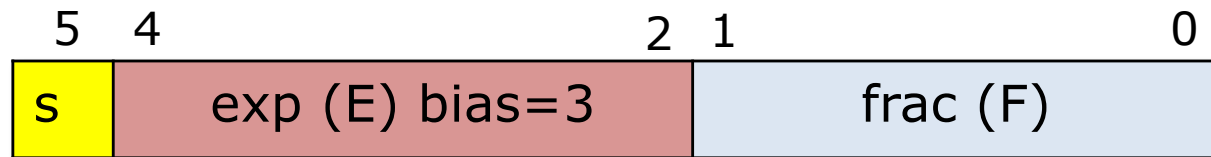


Toy Number System

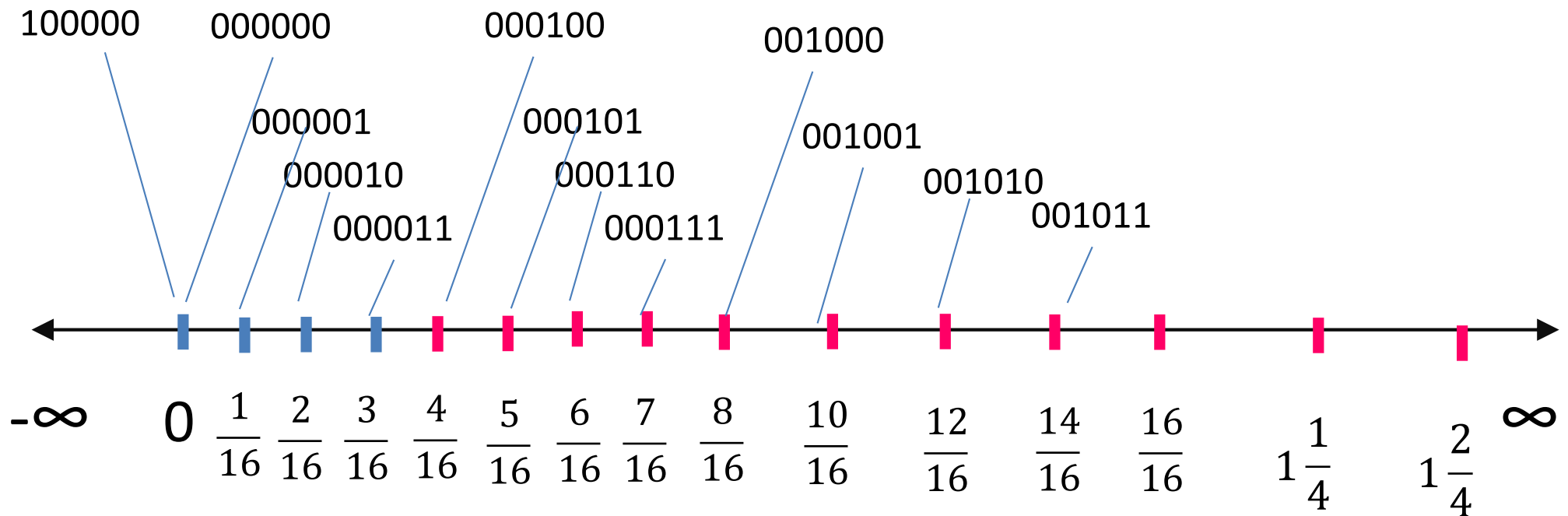
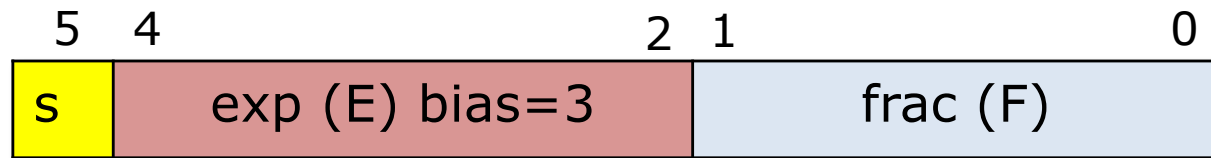


All denormalized # have equal spacing

Toy Number System



Toy Number System



Special Values

Special Value's Encoding:



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

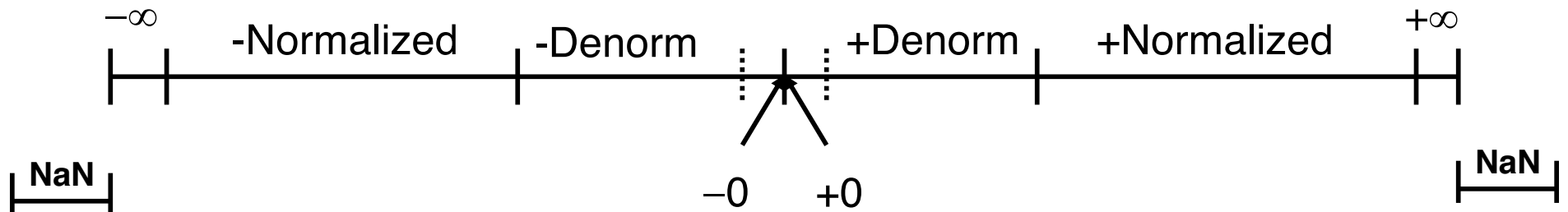
Exercises

representation	E	M	V
0100 1001 0101 0000 0000 0000 0000 0000			
			$2.5 * 2^{-127}$
			$-1.25 * 2^{-111}$
1111 1111 1111 1111 0000 0000 0000 0000			
1111 1111 1000 0000 0000 0000 0000 0000			
			$1.5 * 2^{-127}$

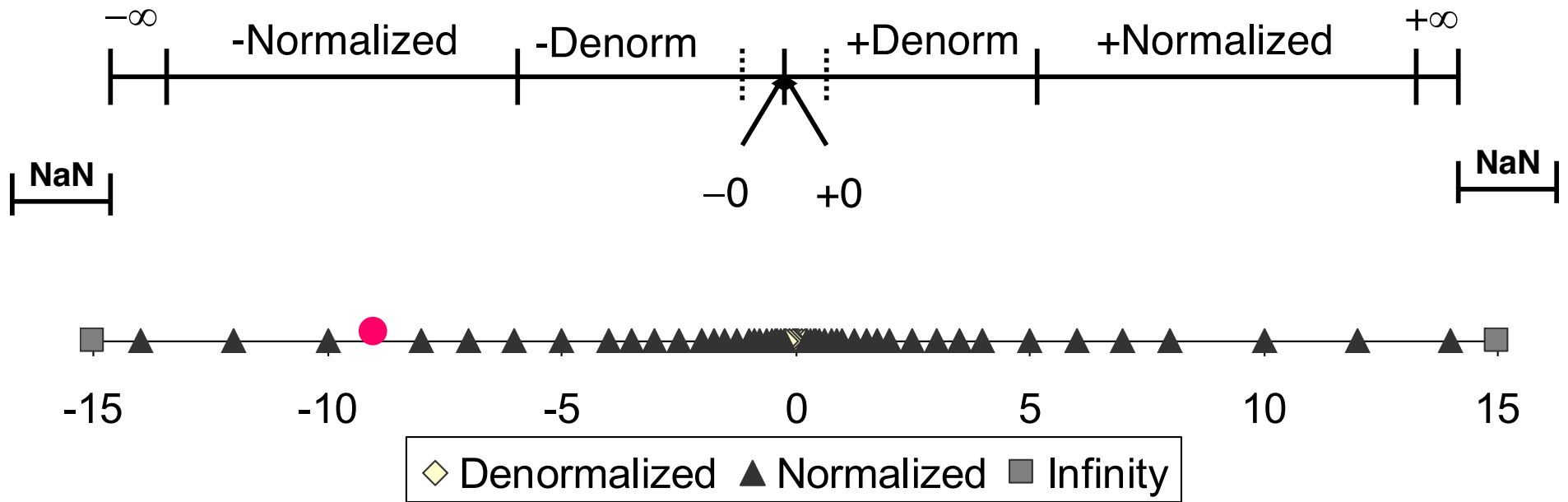
Exercises

representation	E	M	V
0100 1001 0101 0000 0000 0000 0000 0000	$146 - 127 = 19$	$(1.101)_2$ $= 1.625$	$1.625 * 2^{19}$
0000 0000 1010 0000 0000 0000 0000 0000	$1 - 127 = -126$	$(1.01)_2$ $= 1.25$	$2.5 * 2^{-127}$ $= (1.01)_2 * 2^{-126}$
1000 1000 0010 0000 0000 0000 0000 0000	$16 - 127 = -111$	$(1.01)_2$ $= 1.125$	$-1.25 * 2^{-111}$
1111 1111 1111 1111 0000 0000 0000 0000	-	-	Nan
1111 1111 1000 0000 0000 0000 0000 0000	-	-	$-\infty$
0000 0000 0110 0000 0000 0000 0000 0000	-126	$(0.11)_2$	$(0.11)_2 * 2^{-126}$ $= 1.5 * 2^{-127}$

Distribution of Representable Values



Distribution of Representable Values



What if the result of computation is at ● ?

Rounding

Goal

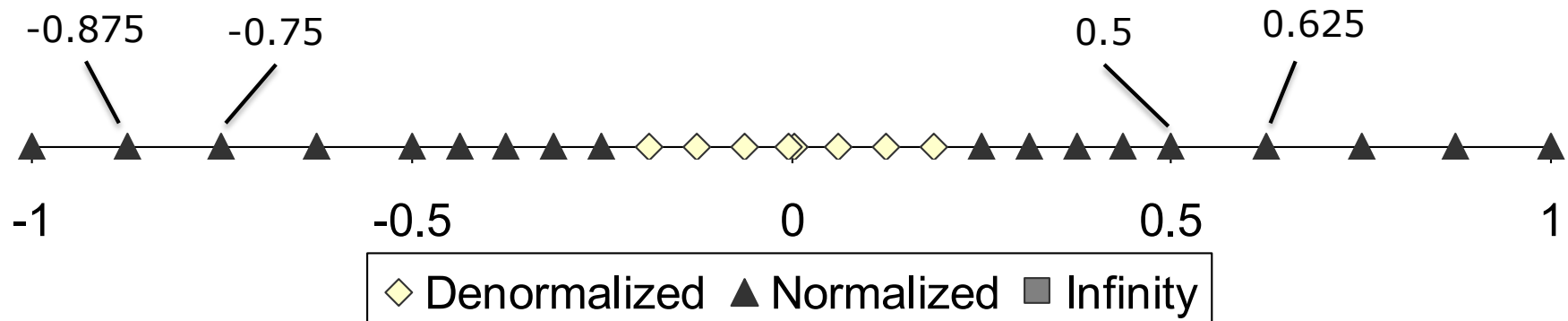
- Use the “closest” representable value x' to represent x .

Round modes

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

Round down

$$\text{Round}(x) = x_- \quad (x_- \leq x)$$

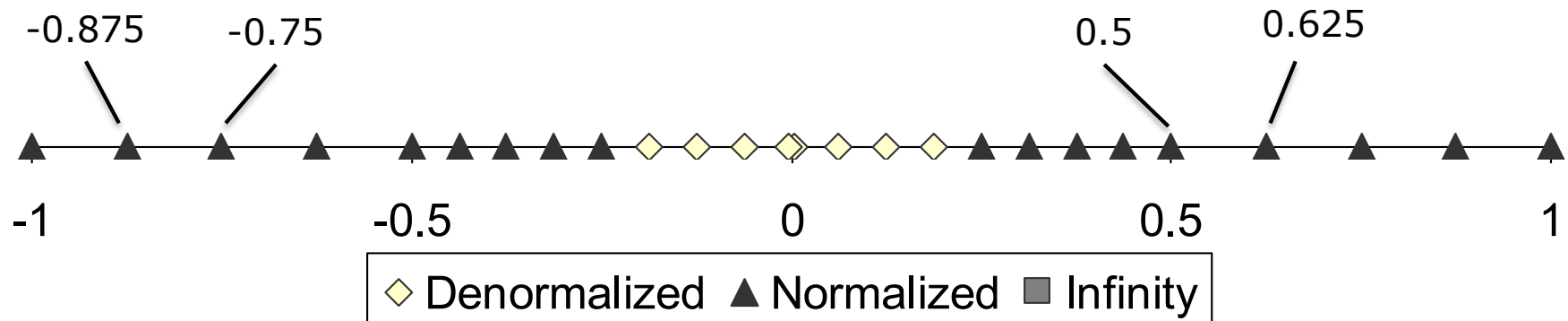


Round(-0.86) = ?

Round(0.55) = ?

Round down

$$\text{Round}(x) = x_- \quad (x_- \leq x)$$



$$\text{Round}(-0.86) = -0.875$$

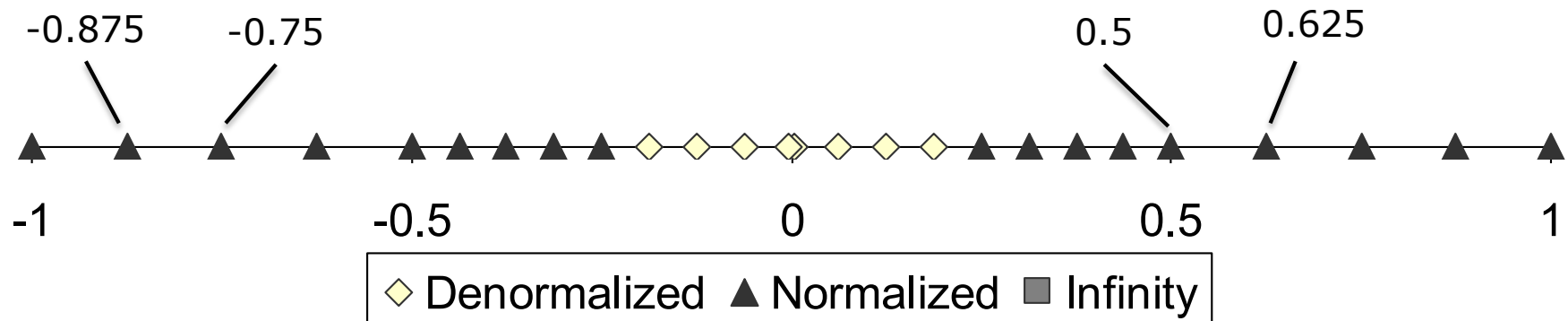
$$\text{Round}(0.55) = 0.5$$

Round up

$$\text{Round}(x) = x_+ \quad (x_+ \geq x)$$

Round up

$$\text{Round}(x) = x_+ \quad (x_+ \geq x)$$

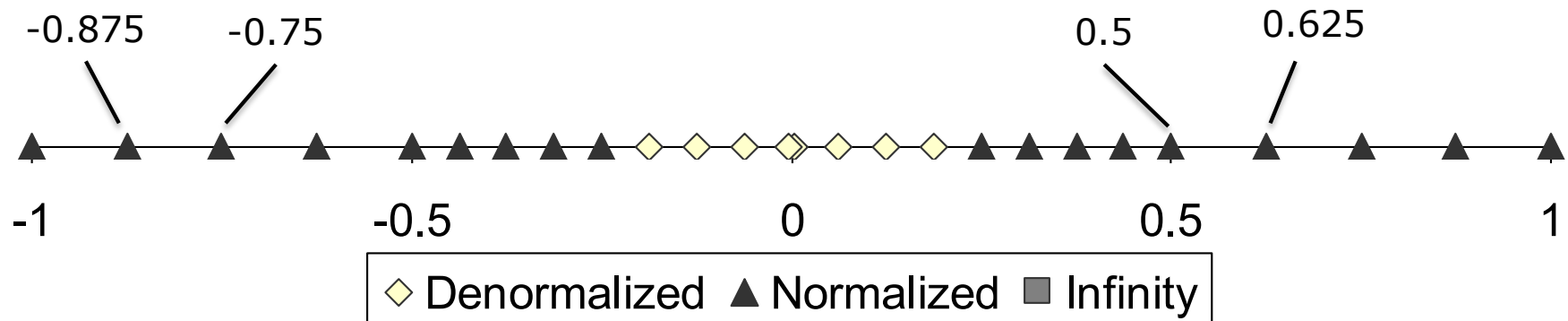


Round(-0.86) = ?

Round(0.55) = ?

Round up

$$\text{Round}(x) = x_+ \quad (x_+ \geq x)$$



$$\text{Round}(-0.86) = -0.75$$

$$\text{Round}(0.55) = 0.625$$

Round towards zero

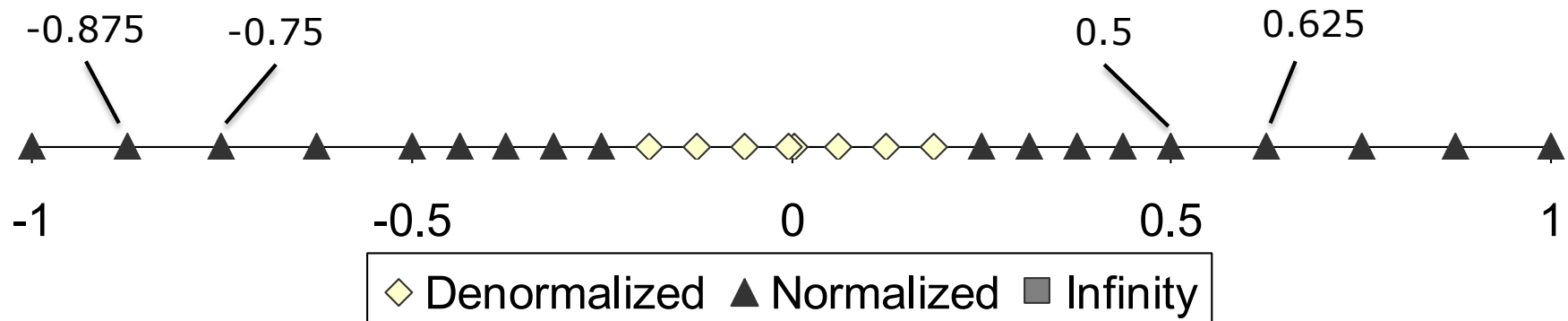
$$\text{Round}(x) = x_+ \text{ if } x < 0$$

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

Round towards zero

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

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



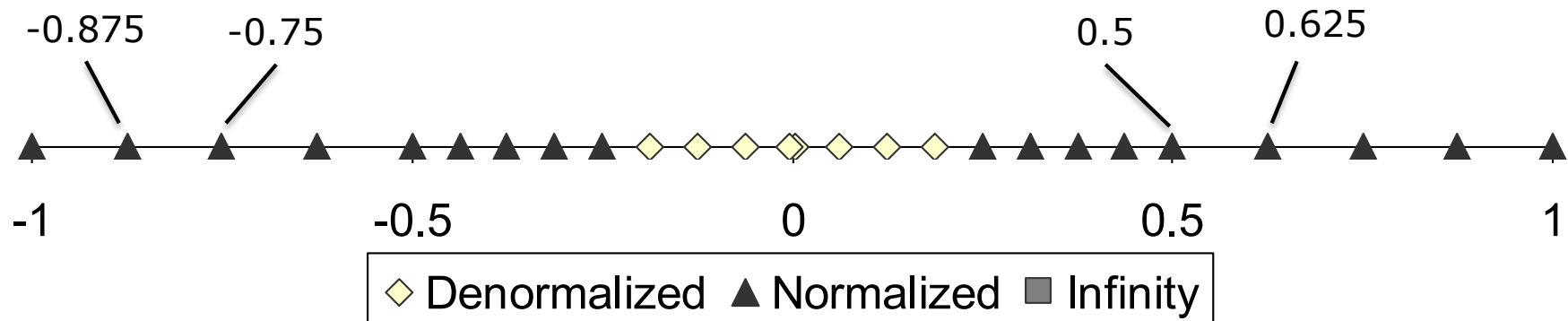
$\text{Round}(-0.86) = ?$

$\text{Round}(0.55) = ?$

Round towards zero

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

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



$\text{Round}(-0.86) = -0.75$

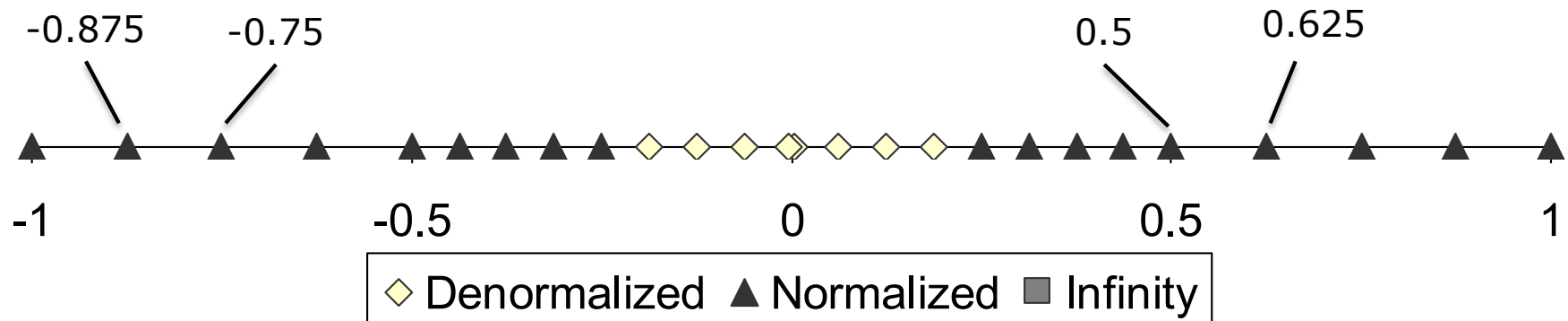
$\text{Round}(0.55) = 0.5$

Round to nearest

Round(x) either x_+ or x_- , whichever is nearer to x .

Round to nearest

$\text{Round}(x)$ either x_+ or x_- , whichever is nearer to x .

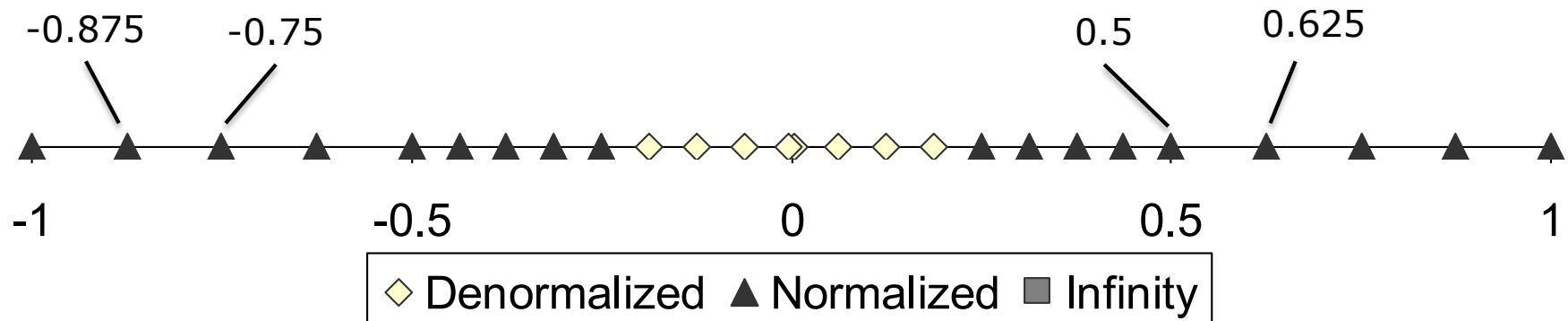


$\text{Round}(-0.86) = ?$

$\text{Round}(0.55) = ?$

Round to nearest

$\text{Round}(x)$ either x_+ or x_- , whichever is nearer to x .

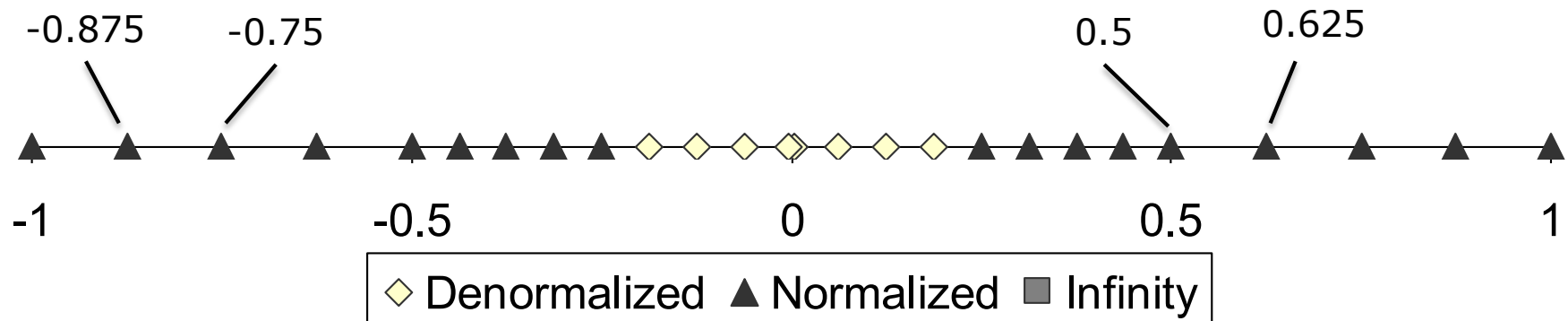


$$\text{Round}(-0.86) = -0.875$$

$$\text{Round}(0.55) = 0.5$$

Round to nearest; ties to even

Round(x) either x_+ or x_- , whichever is nearer to x .

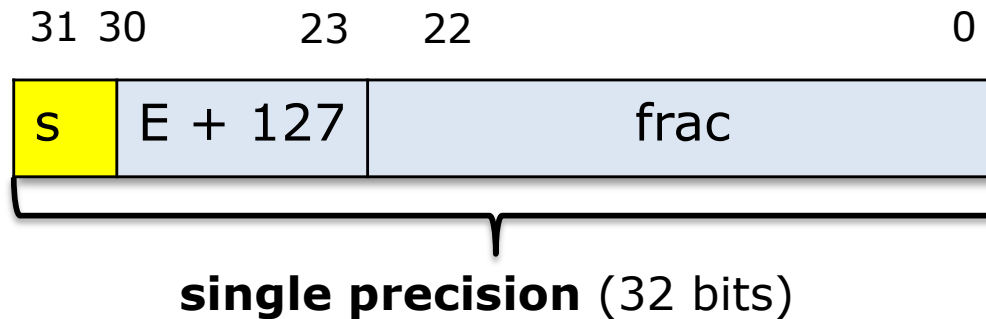


$$\text{Round}(-0.86) = -0.875$$

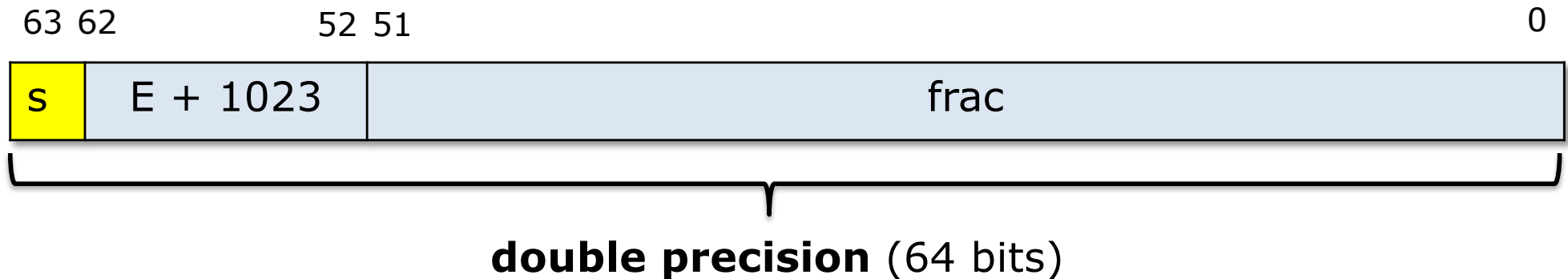
$$\text{Round}(0.55) = 0.5$$

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

single/ double precision



float $f = 0.1$
double $d = 0.1$

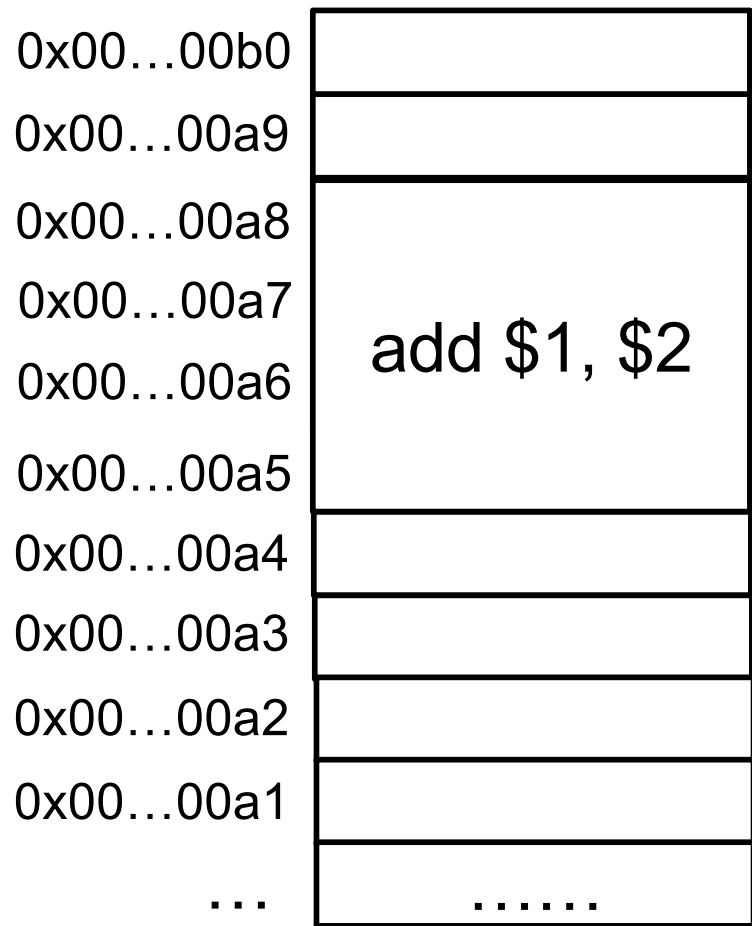


single/ double precision

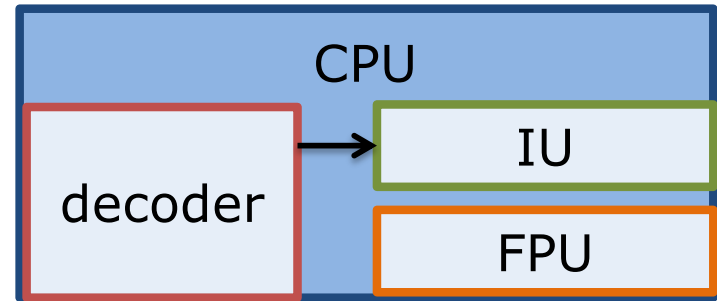
	E_{\min}	E_{\max}	N_{\min}	N_{\max}
Float	-126	127	$\approx 2^{-126}$	$\approx 2^{128}$
Double	-1022	1023	$\approx 2^{-1022}$	$\approx 2^{1024}$

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

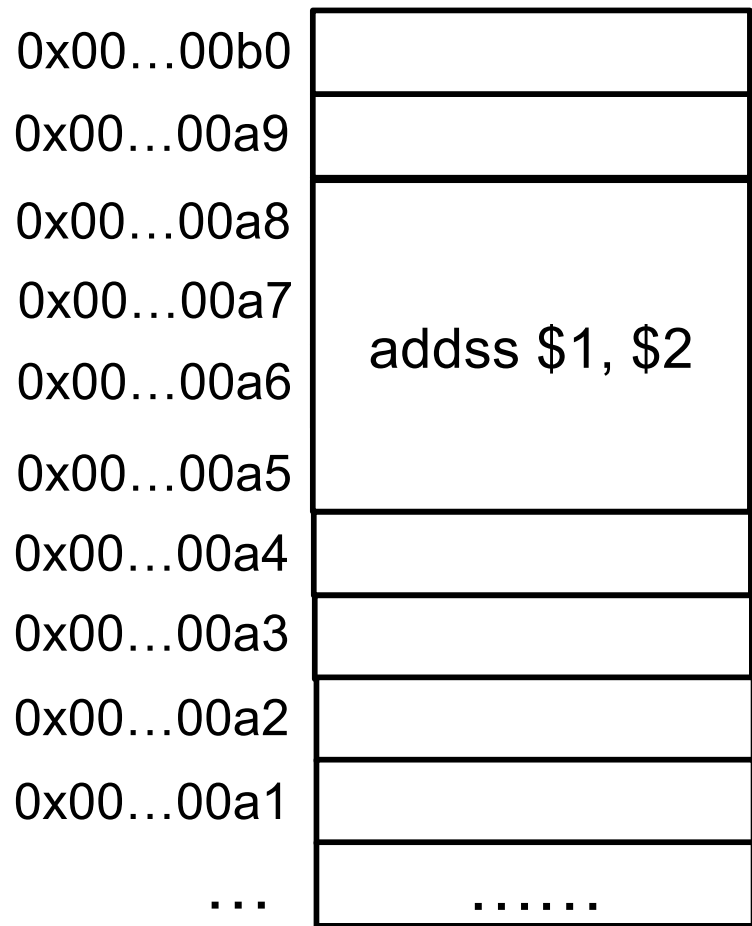
By having specific instruction for floating points operation.



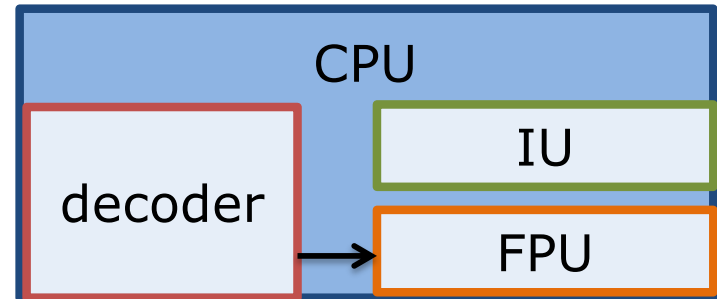
Memory



```
int d = 1 + 2
```

Memory



float f = 0.1 + 0.2

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!

Why divide by zero = ∞ ?

- Allow a calculation to continue and produce a valid result
- Example:



A circuit diagram showing a voltage source V connected in series with two parallel resistors, R_1 and R_2 . The voltage source is represented by a battery symbol with a '+' sign on the right and a '-' sign on the left. The resistors are represented by zigzag lines.

$$\text{Parallel resistance} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$$

If R_1 or R_2 is 0, overall resistance should be 0

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	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

- 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)

