Floating point

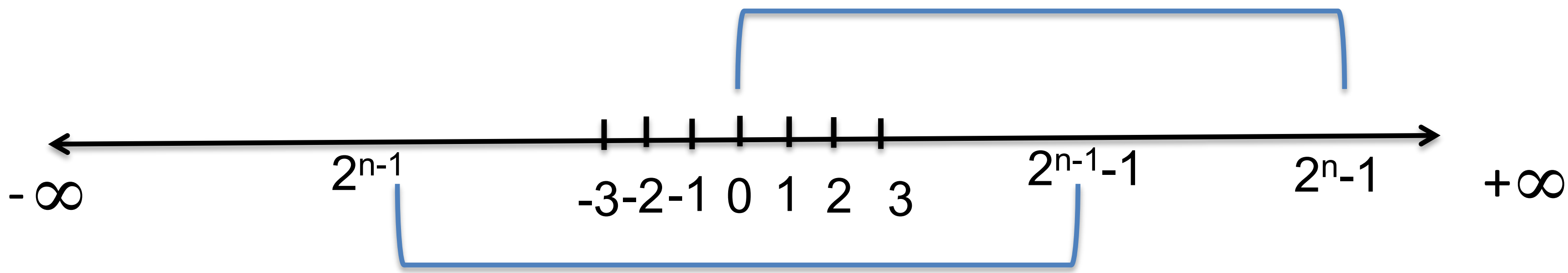
Shuai Mu

based on Tiger Wang’s and Jinyang Li’ slides

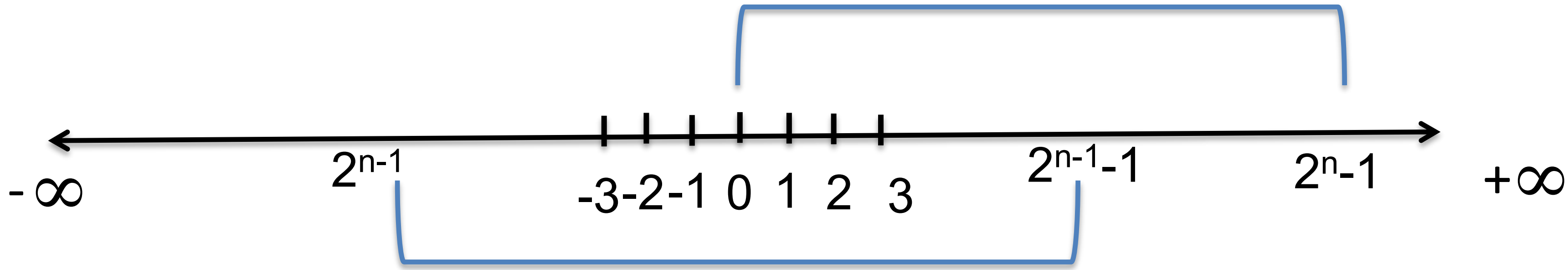
Representing Real Numbers using bits

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	$(0.3333333...)_{10}$
Square root (2)	$(1.414...)_{10}$

Decimal Representation

Real Numbers	Decimal Representation (Expansion)
11 / 2	$(5.5)_{10}$
1 / 3	$(0.3333333...)_{10}$
Square root (2)	$(1.414...)_{10}$

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

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

$(1.414...)_{10} = 1 * 10^0 + 4 * 10^{-1} + 1 * 10^{-2} + 4 * 10^{-3} + ...$

Decimal Representation

Real Numbers	Decimal Representation (Expansion)
11 / 2	$(5.5)_{10}$
1 / 3	$(0.3333333...)_{10}$
Square root (2)	$(1.4142...)_{10}$

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

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

$(1.414...)_{10} = 1 * 10^0 + 4 * 10^{-1} + 1 * 10^{-2} + 4 * 10^{-3} + ...$

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

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

Binary Representation

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

$= 1 * 2^2 + 1 * 2^0 + 1 * 2^{-1}$

Binary Representation

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

$= 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1}$

Binary Representation

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

$= 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1}$

$= (101.1)_2$

Binary Representation

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

$= 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1}$

$= (101.1)_2$

$(0.333333...)_{10} = 1 / 4 + 1 / 16 + 1 / 64 + ...$

$= (0.01010101...)_2$

Binary Representation

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

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

$$\overset{2^p}{b_p}\,\overset{2^{p-1}}{b_{p-1}} \cdots \overset{2}{b_1}\,\overset{1}{b_0} . \underset{1/2}{b_{-1}}\,\underset{1/4}{b_{-2}} \cdots \underset{1/2^q}{b_{-q}} = \sum_{i=-q}^{p} 2^i \times b_i$$

Exercise

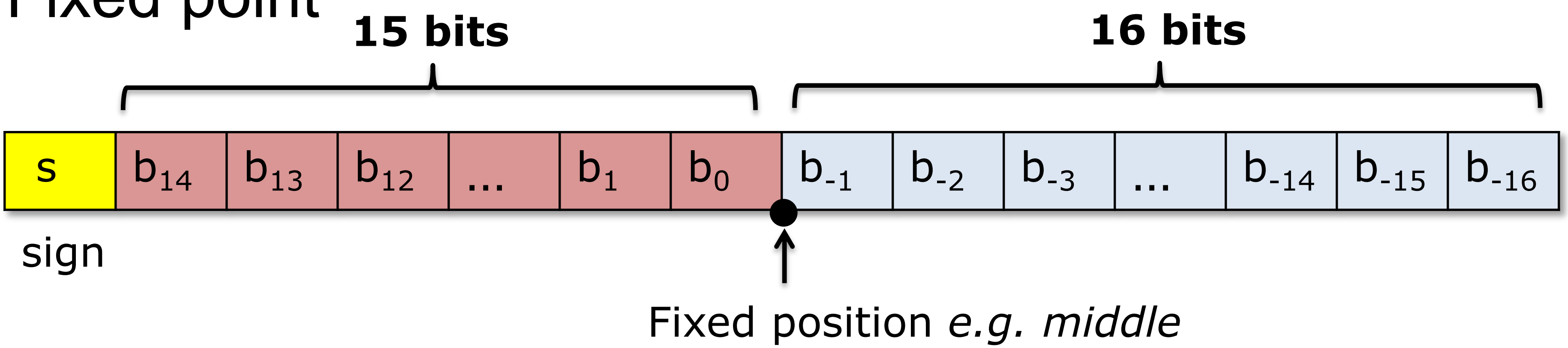
Binary Expansion	Formula	Decimal
10.011_2		
	$2^{-3} + 2^{-4} + 2^{-6}$	
	$2^{-1} + 2^{-2} + 2^{-3} + 2^{-4}$	

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}

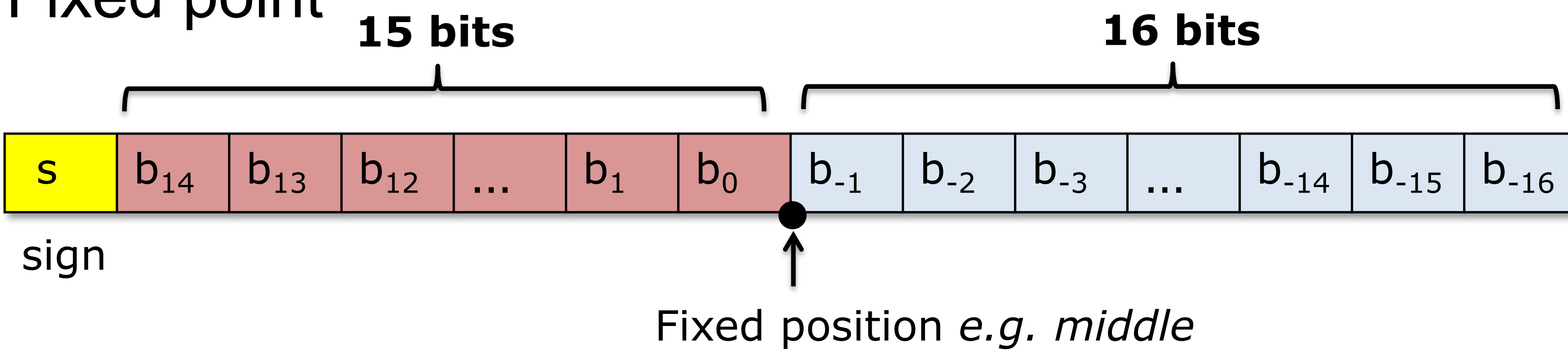
Intuitive Idea

Fixed point

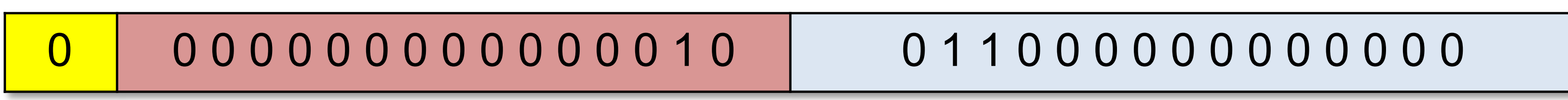


Intuitive Idea

Fixed point



$(\ 10.011\ )_2$



Problems of Fixed Point

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

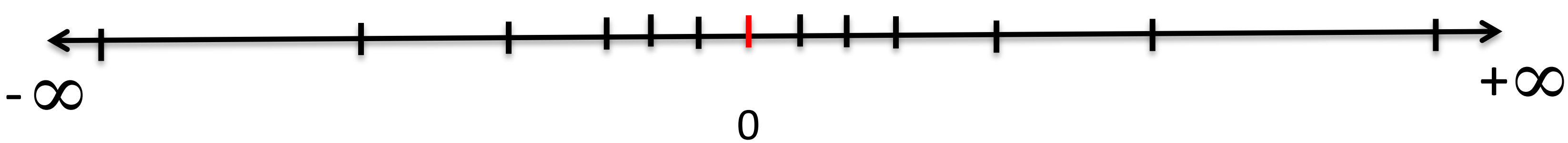
- Largest number: 2^{15} $(011...111)_2$
- Highest precision: 2^{-16}

→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)

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

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

$r_{10} = \underline{+}M * 2^E$, where $1 <= M < 2$

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

M: significant, E: exponent

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

Floating Point

Binary exponential representation

$$\left.\begin{array}{l} r_{10} = \underline{+}M * 2^E, \text{ where } 1 <= M < 2 \\ M = (\ 1.b_1b_2b_3...b_n\ )_2 \end{array}\right\}$$ 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

$$\left.\begin{array}{l} r_{10} = \pm M * 2^{E}, \text{ where } 1 <= M < 2 \\ M = (\,1.b_1b_2b_3\ldots b_n\,)_2 \end{array}\right\}$$ 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?

Normalized representation

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

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

M: significant, E: exponent

$(\ 1.b_1b_2b_3\ldots b_n\ )_2$

Normalized representation in computer

$r_{10} = \underline{+}M * 2^E$, where $1 <= M < 2$

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

M: significant, E: exponent

$(b_1b_2b_3 \ldots b_{23})_2$

Normalized representation

$r_{10} = \underline{+}M * 2^E$, where $1 <= M < 2$

$M = (1.b_1 b_2 b_3 \ldots b_{23})_2$

M: significant, E: exponent

31	30 … 23	22 … 0
0	0000 0010	0110 0000 0000 0000 0000 000
		$(b_1 b_2 b_3 \ldots b_{23})_2$

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

Exercise

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

Exercise

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

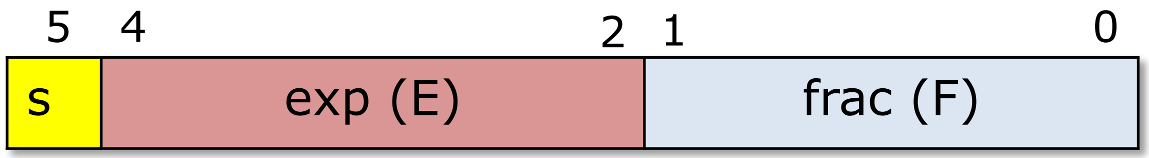
$(10.25)_{10} = (1010.01)_2 = (1.01001)_2 * 2^3$

31	30 … 23	22 … 0
0	0000 0011	0100 1000 0000 0000 0000 000

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

31	30 … 23	22 … 0
0	0000 0110	0001 1100 0000 0000 0000 000

Toy Number System

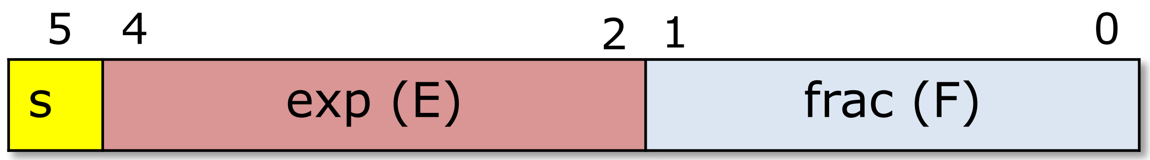


6-bit floating point representation

- exponent: 3 bits
- fraction: 2 bits

Largest positive number ?

Toy Number System



6-bit floating point representation

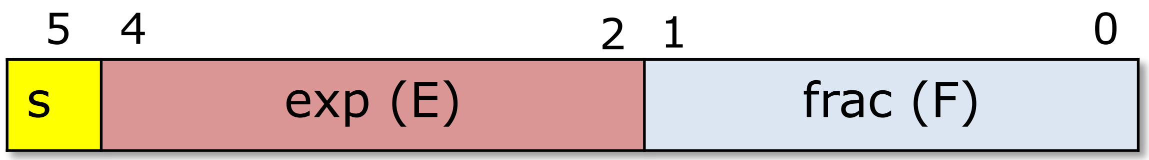
– exponent: 3 bits

– 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

5	4 2	1 0
s	exp (E)	frac (F)

6-bit floating point representation

- exponent: 3 bits
- fraction: 2 bits

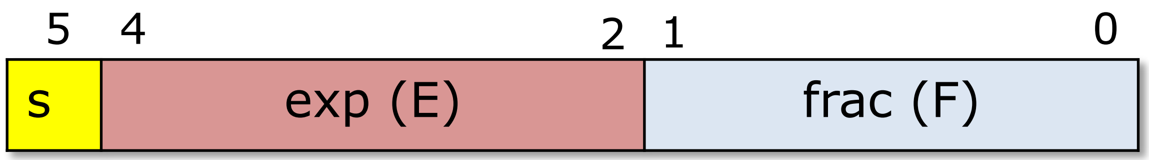
Largest positive number: 224

Smallest positive number: 1

0	0 0 0	0 0

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

Toy Number System

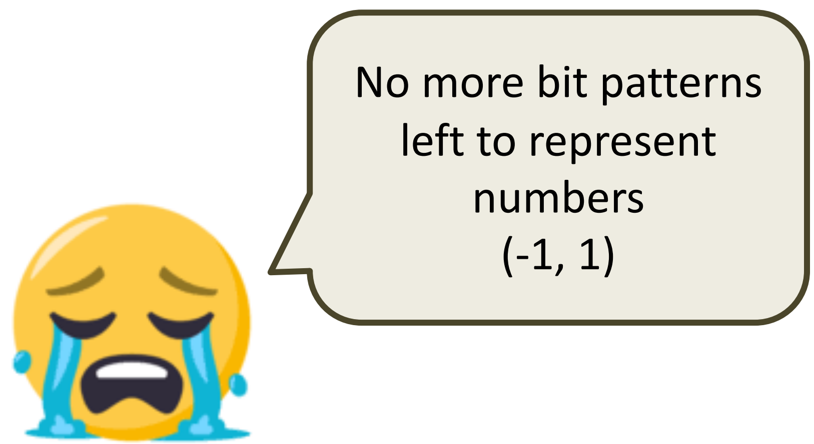


6-bit floating point representation

– exponent: 3 bits
– fraction: 2 bits

Positive number: 1 to 224

Negative number: -224 to -1



Questions

How to represent

1. number close or equal to 0?
2. larger numbers, even ∞ ?
3. the result of dividing by 0 ?

Questions

How to represent

1. number close or equal to 0?
2. larger numbers, even ∞ ?
3. the result of dividing by 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)

National Semiconductor
The Sight & Sound of Information

intel®

hp

digital
EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS

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 by all machines adopting the standard

Correctly rounded floating point operations, using several rounding modes, since exact answers often cannot be represented exactly on the computer

Consistent treatment of exceptional situations such as division by zero

Restrictions for Normalized Representation

$r_{10} = \underline{+}M * 2^E$, where $1 <= M < 2$

$M = (1.b_0b_1b_2b_3...b_n)_2$

M: significant, E: exponent

31	30 … 23	22 … 0
s	exp (E)	fraction (F)
		$(b_0b_1b_2b_3...b_n)_2$

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

Restrictions for Normalized Representation

$r_{10} = \underline{+}M * 2^E$, where $1 <= M < 2$

$M = (\ 1.b_0b_1b_2b_3\ldots b_n\ )_2$

M: significant, E: exponent

$(\ b_0b_1b_2b_3\ldots b_n\ )_2$

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

$E_{max} = ?$

$E_{min} = ?$

Restrictions for Normalized Representation

$r_{10} = \underline{+}M * 2^E$, where $1 <= M < 2$

$M = (\ 1.b_0b_1b_2b_3\ldots b_n\ )_2$

M: significant, E: exponent

$(\ b_0b_1b_2b_3\ldots b_n\ )_2$

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

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

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

represent values between -1 and 1 in normalized representation

Exponential Bias

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

$M = (1.b_0b_1b_2b_3 \ldots b_n)_2$

M: significant, E: exponent

$(b_0b_1b_2b_3 \ldots b_n)_2$

Bias: 127

Exponential Bias

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

$M = (1.b_0b_1b_2b_3\ldots b_n)_2$

M: significant, E: exponent

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

$(b_0b_1b_2b_3\ldots b_n)_2$

Bias: 127

$E_{max} = ?$

$E_{min} = ?$

Exponential Bias

$r_{10} = \underline{+}M * 2^E$, where $1 <= M < 2$

$M = (1.b_0b_1b_2b_3...b_n)_2$

M: significant, E: exponent

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

$(b_0b_1b_2b_3...b_n)_2$

Bias: 127

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

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

Exponential Bias

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

$M = (1.b_0b_1b_2b_3...b_n)_2$

M: significant, E: exponent

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

$(b_0b_1b_2b_3...b_n)_2$

Bias: 127

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

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

Smallest positive number: ?

Negative number with smallest absolute value: ?

Exponential Bias

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

$M = (1.b_0b_1b_2b_3 \ldots b_n)_2$

M: significant, E: exponent

31	30 … 23	22 … 0
s	0000 0001	fraction (F)

$(b_0b_1b_2b_3 \ldots b_n)_2$

Bias: 127

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

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

Smallest positive number: 2^{-126}

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

Questions from Munachiso

Q1. Why does it need **bias**?

Q2. Why is the **bias** 127?

Questions from Munachiso

Q1. Why does it need **bias**?

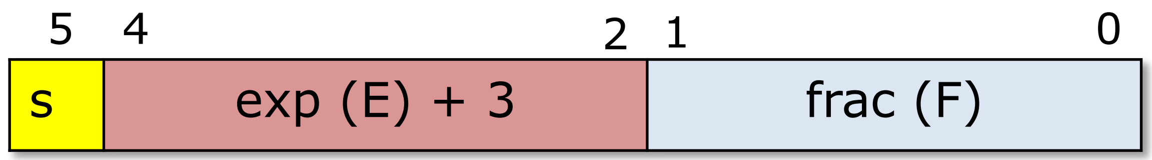
A1. Use unsigned number to represent negative numbers (-1 ~ -126)

Questions from Munachiso

Q2. Why is 127?

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

Toy Number System



6-bit floating point representation

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

Smallest positive number ?

Toy Number System

5	4 – 2	1 – 0
s	exp (E) + 3	frac (F)

6-bit floating point representation

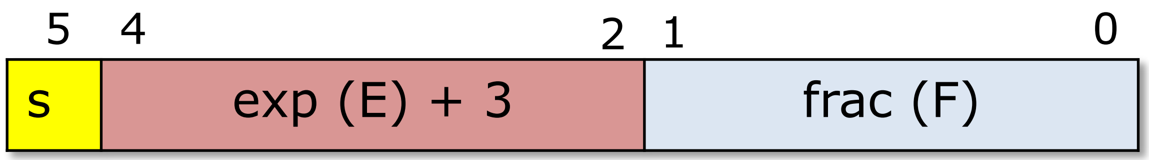
- exponent: 3 bits
- fraction: 2 bits
- **bias: 3**

Smallest positive number: 0.25

0	0 0 1	0 0

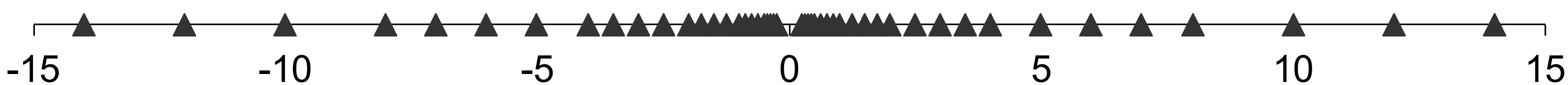
$(1.00)_2 * 2^{-2} = 0.25$

Toy Number System

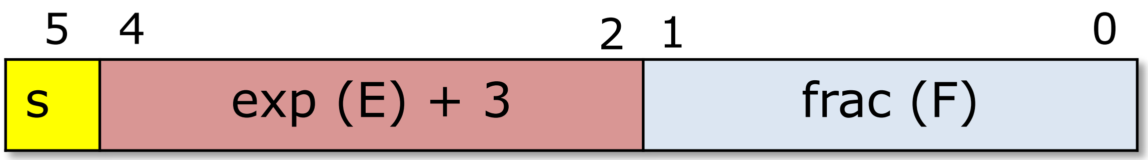


6-bit floating point representation

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

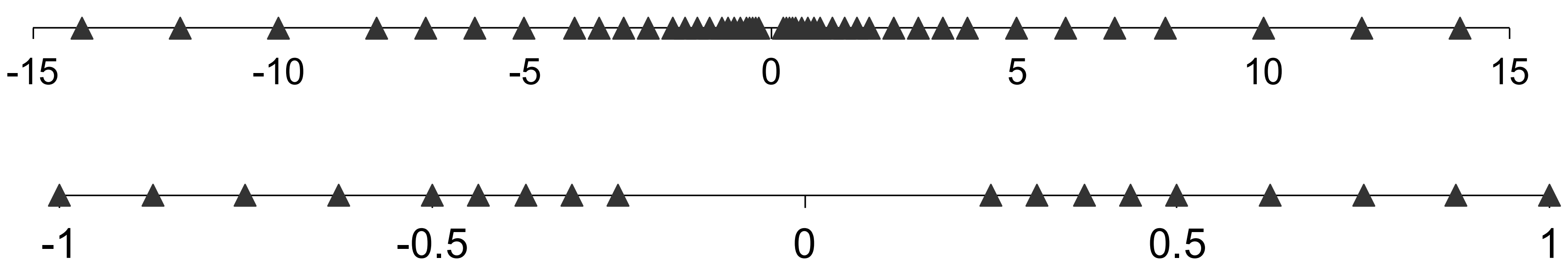


Toy Number System

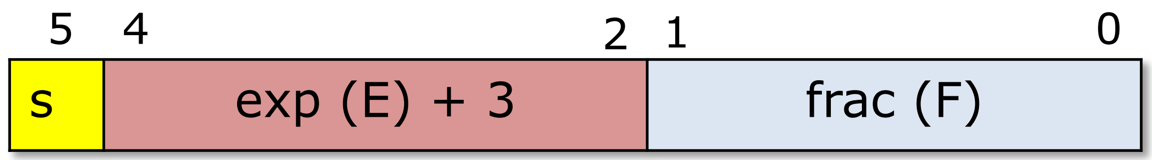


6-bit floating point representation

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

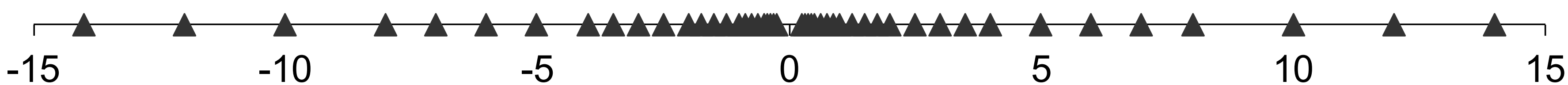


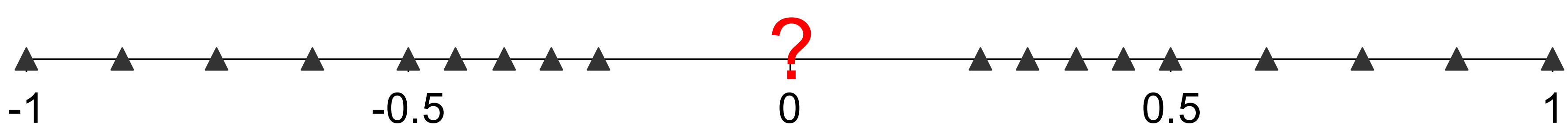
Toy Number System



6-bit floating point representation

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





represent values which are close and equal to 0

Denormalization

$r_{10} = \pm M * 2^E$, M: significant, E: exponent

Normalized Encoding:

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

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

Denormalization

$r_{10} = \underline{+}M * 2^{E}$, M: significant, E: exponent

Normalized Encoding:

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

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

Denormalized Encoding:

31	30 … 23	22 … 0
s	0000 0000	fraction (F)

$E = 1 – Bias = -126$

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

Zeros

+0.0

0	0000 0000	0000 0000 0000 0000 0000 000

-0.0

1	0000 0000	0000 0000 0000 0000 0000 000

Examples

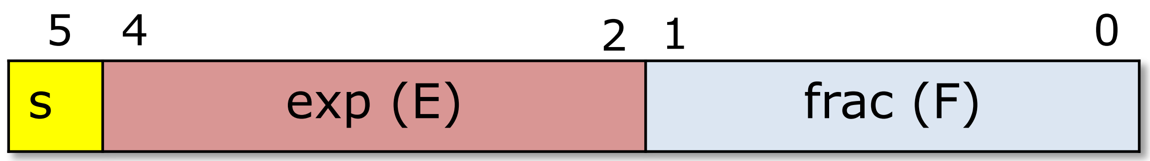
$(0.1)_2 * 2^{-126}$

0	0000 0000	1000 0000 0000 0000 0000 000

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

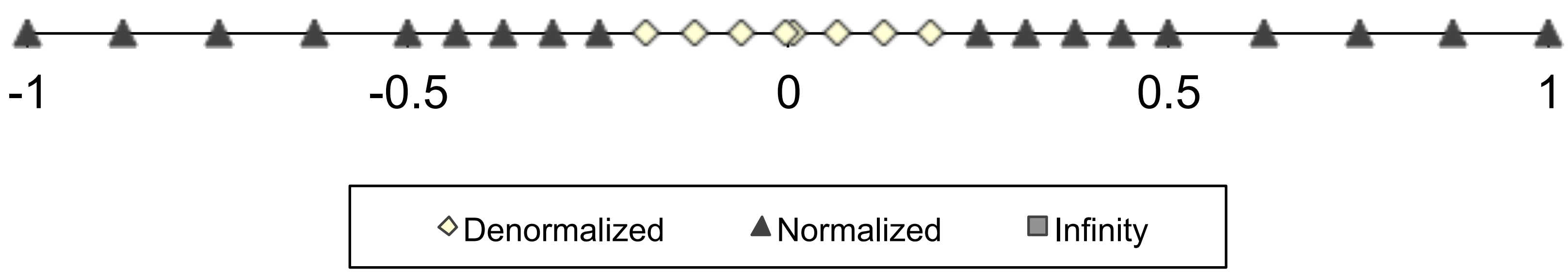
1	0000 0000	0101 0100 0000 0000 0000 000

Toy Number System



6-bit floating point representation

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



Special Values

Special Value's Encoding:

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

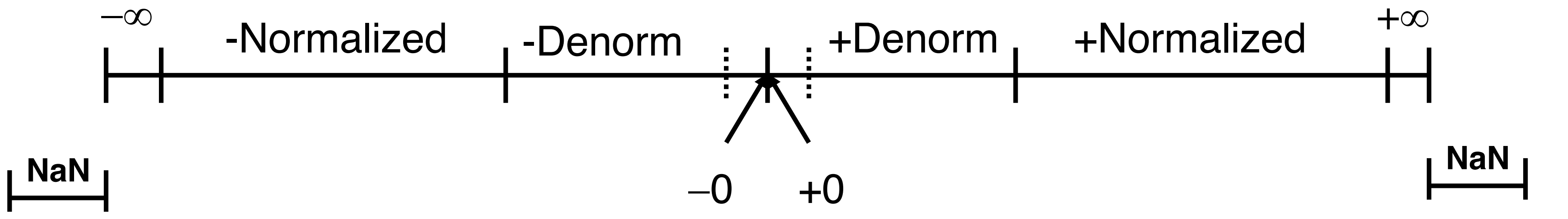
Exercises

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			
			$1.5 * 2^{-127}$

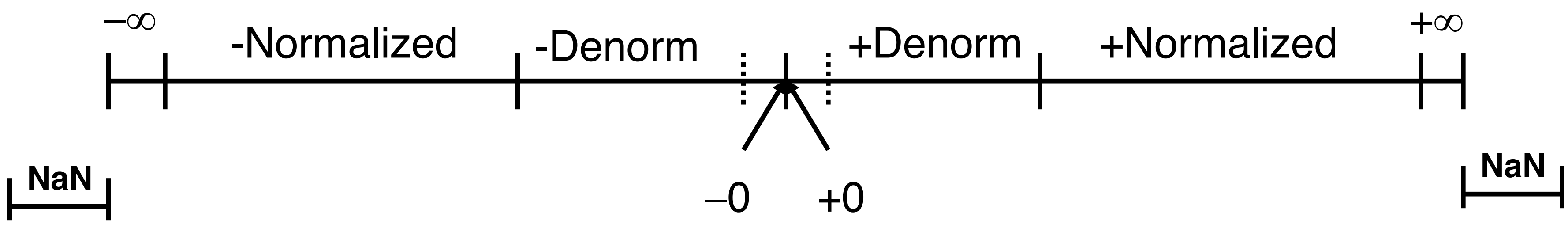
Exercises

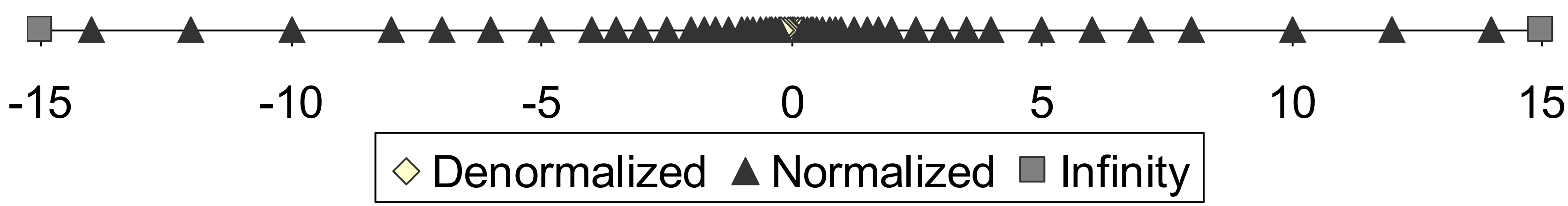
representation	**E**	**M**	**V**
0100 1001 0101 0000 0000 0000 0000 0000	146 – 127 = 19	$(1.101)_2$ = 1.625	1.625 * 2^{19}
0000 0000 1010 0000 0000 0000 0000 0000	1 – 127 = -126	$(1.01)_2$ = 1.25	2.5 * 2^{-127} = $(1.01)_2$ * 2^{-126}
1000 1000 0010 0000 0000 0000 0000 0000	16 – 127 = -111	$(1.01)_2$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

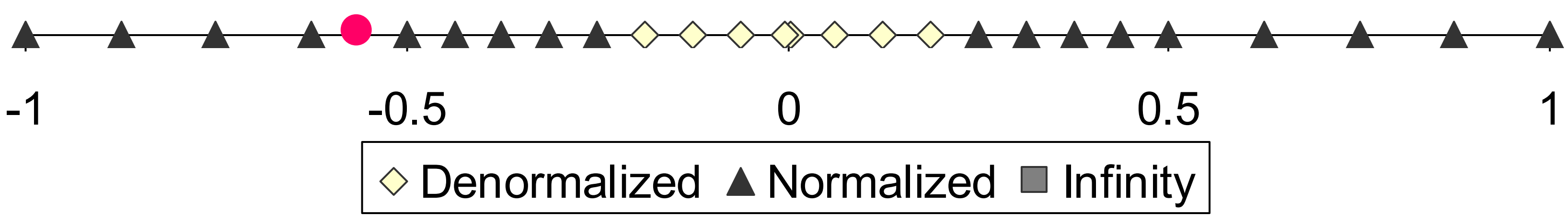
Distribution of Representable Values



Distribution of Representable Values







How to represent the point ● in the format ?

Rounding

Goal

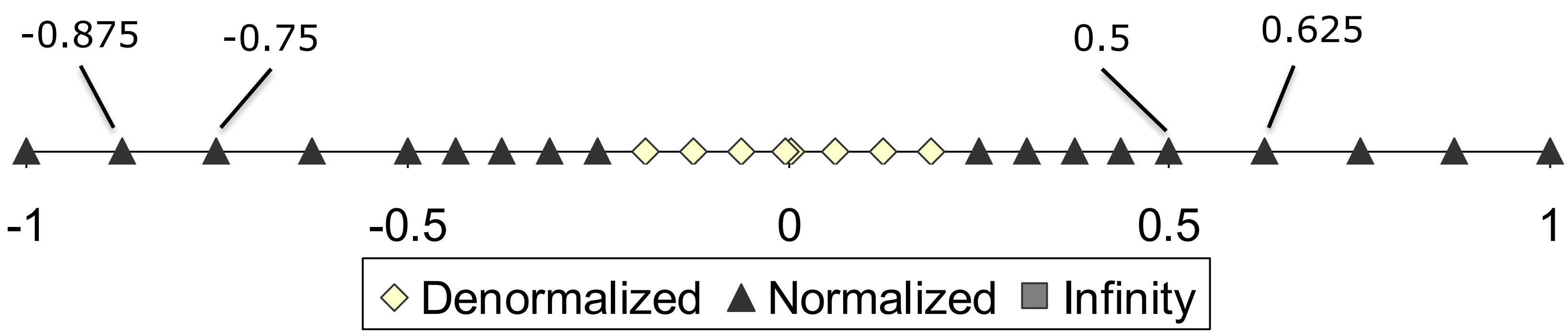
- Given a value x, finding the "closest" matching representable value x'.

Round modes

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

Round down

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

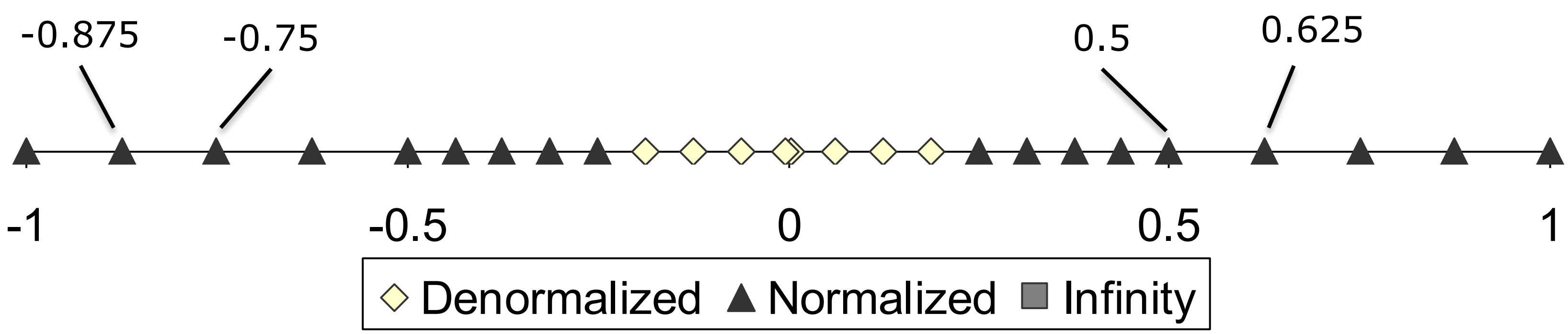


Round(-0.86) = ?

Round(0.55) = ?

Round down

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



Round(-0.86) = -0.875

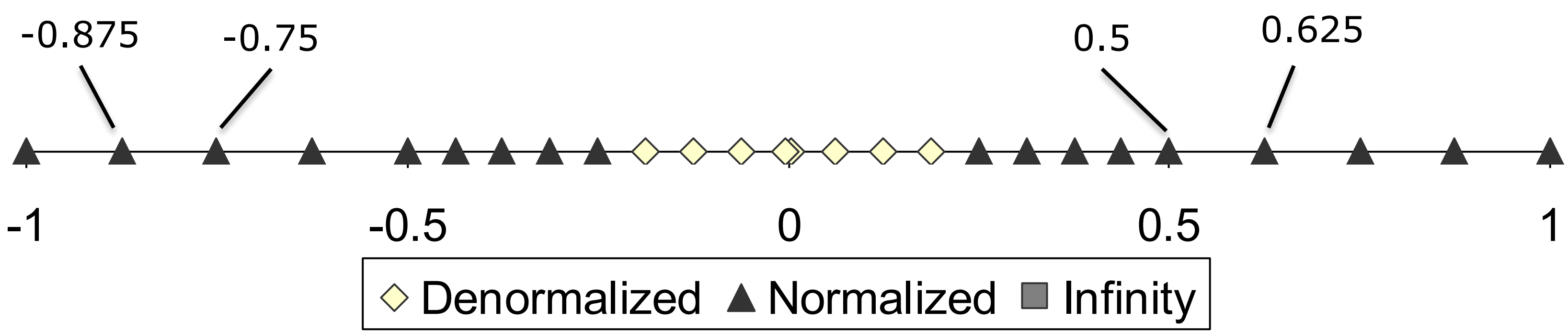
Round(0.55) = 0.5

Round up

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

Round up

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

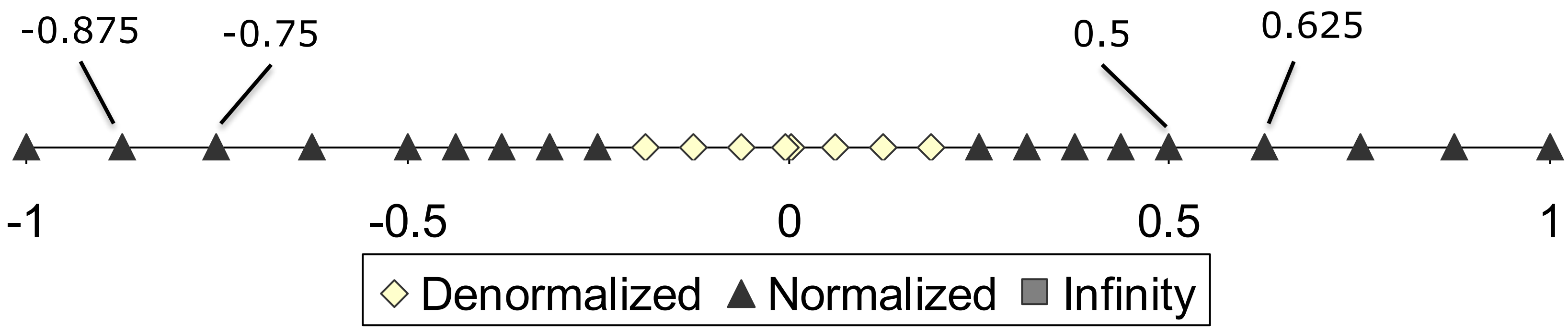


Round(-0.86) = ?

Round(0.55) = ?

Round up

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



Round(-0.86) = -0.75

Round(0.55) = 0.625

Round towards zero

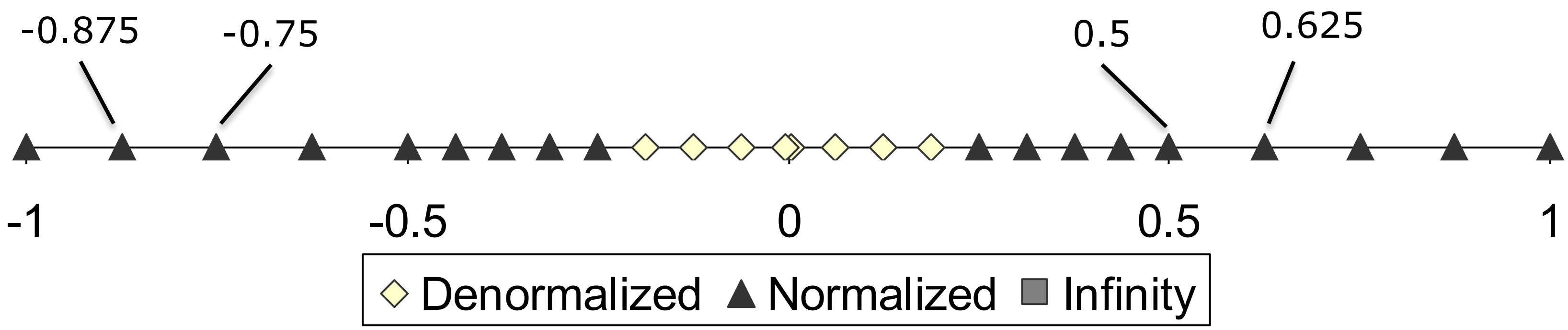
Round(x) = x_+ if x < 0

Round(x) = x_- if x > 0

Round towards zero

Round(x) = x_+ if x < 0

Round(x) = x_- if x > 0



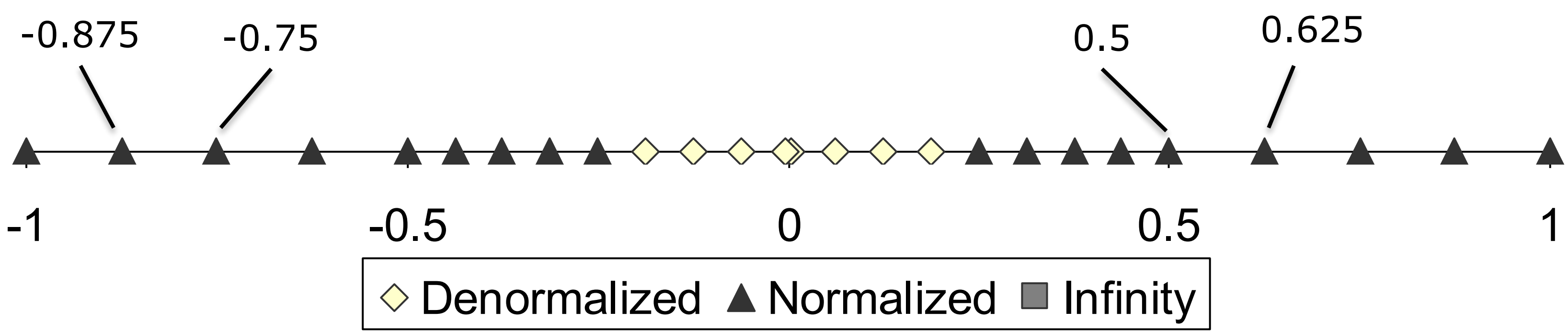
Round(-0.86) = ?

Round(0.55) = ?

Round towards zero

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

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



Round(-0.86) = -0.75

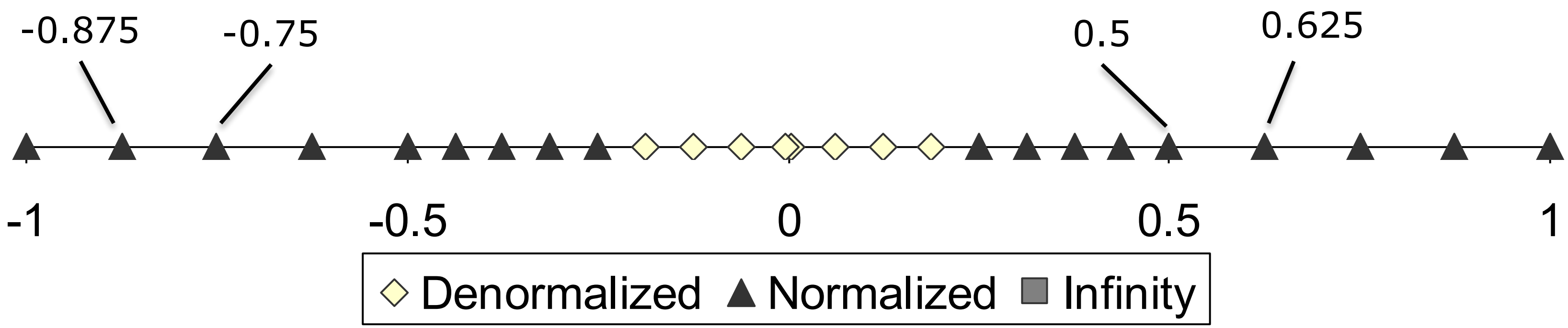
Round(0.55) = 0.5

Round to nearest

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

Round to nearest

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

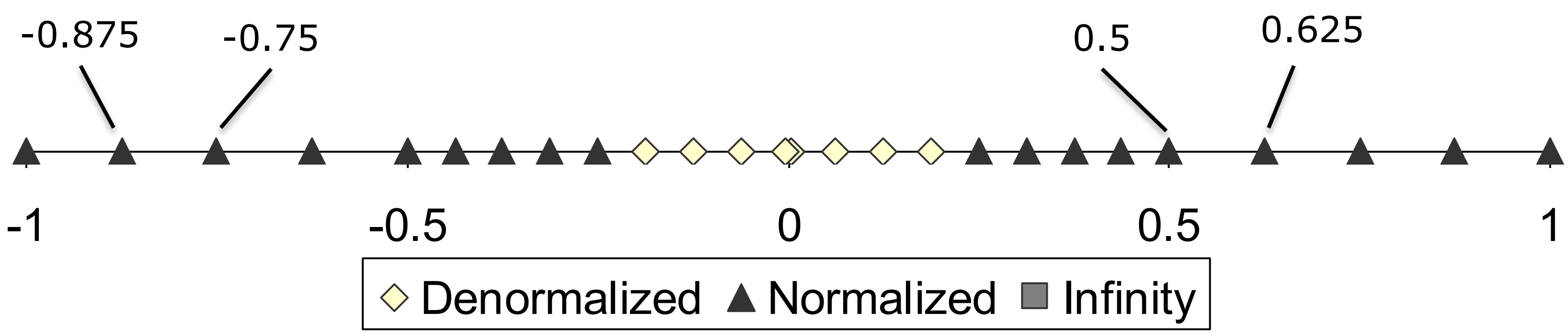


Round(-0.86) = ?

Round(0.55) = ?

Round to nearest

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

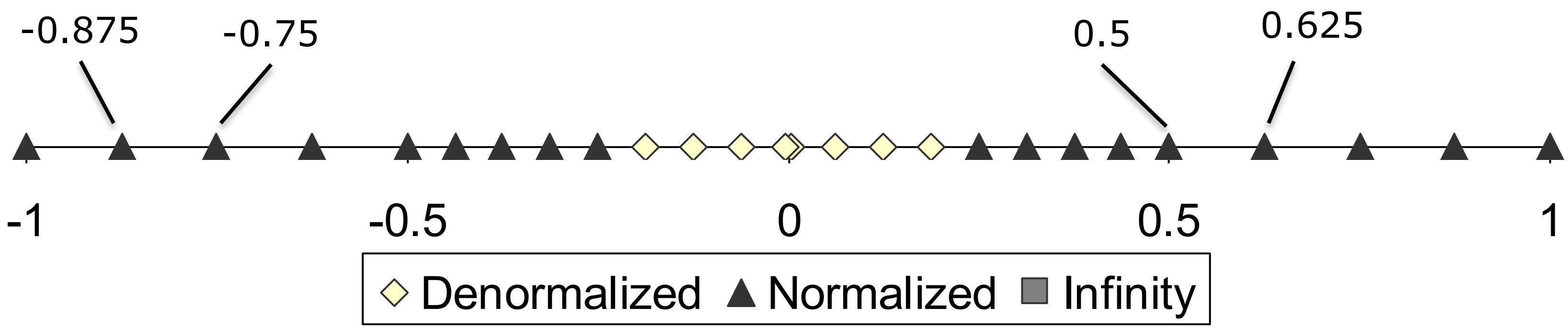


Round(-0.86) = -0.875

Round(0.55) = 0.5

Round to nearest

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

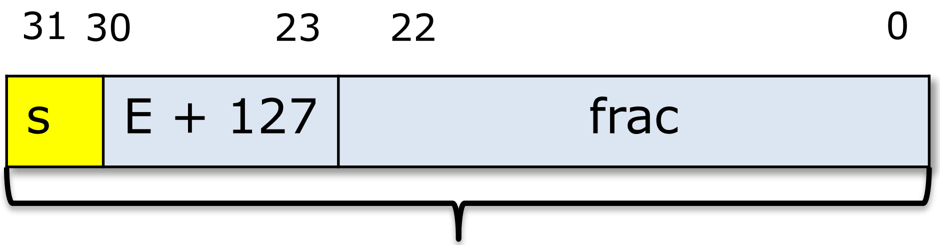


Round(-0.86) = -0.875

Round(0.55) = 0.5

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

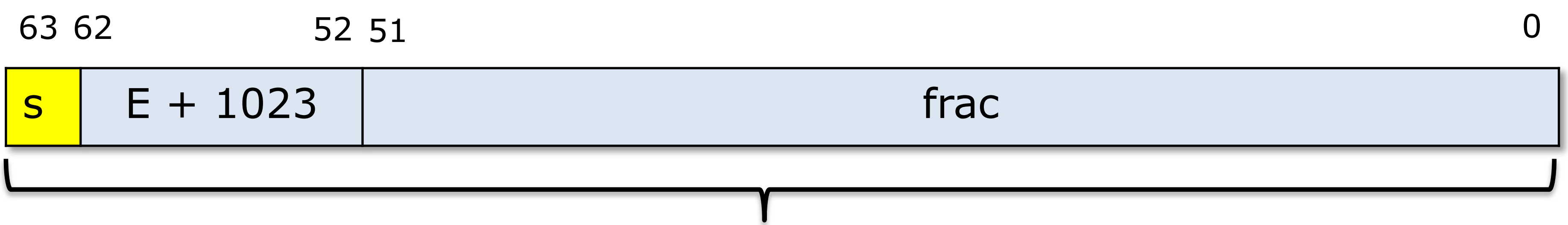
single/ double precision



single precision (32 bits)

float f = 0.1

double d = 0.1



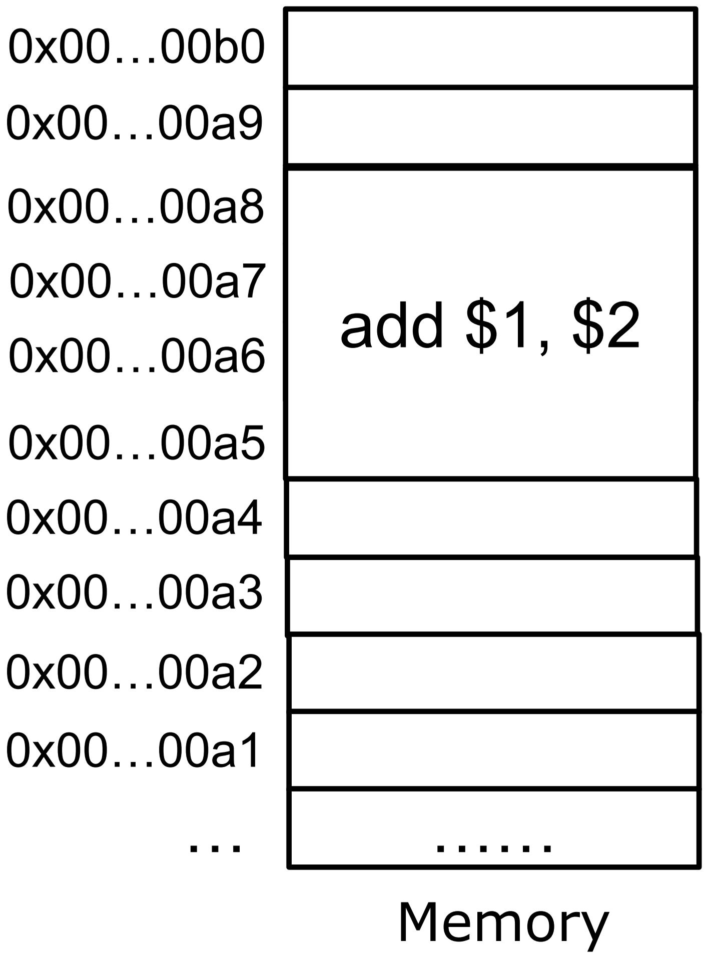
double precision (64 bits)

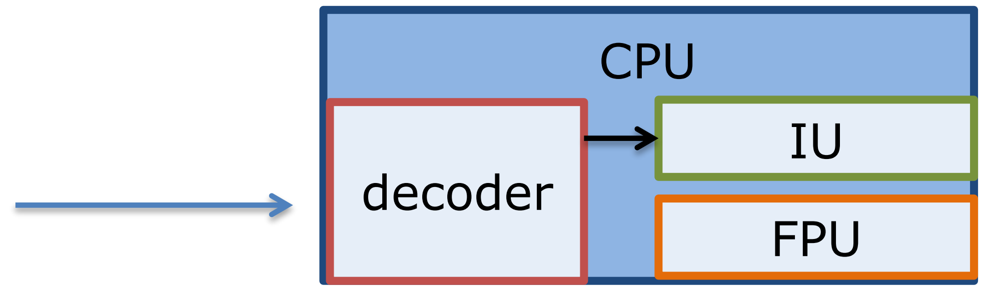
single/ double precision

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

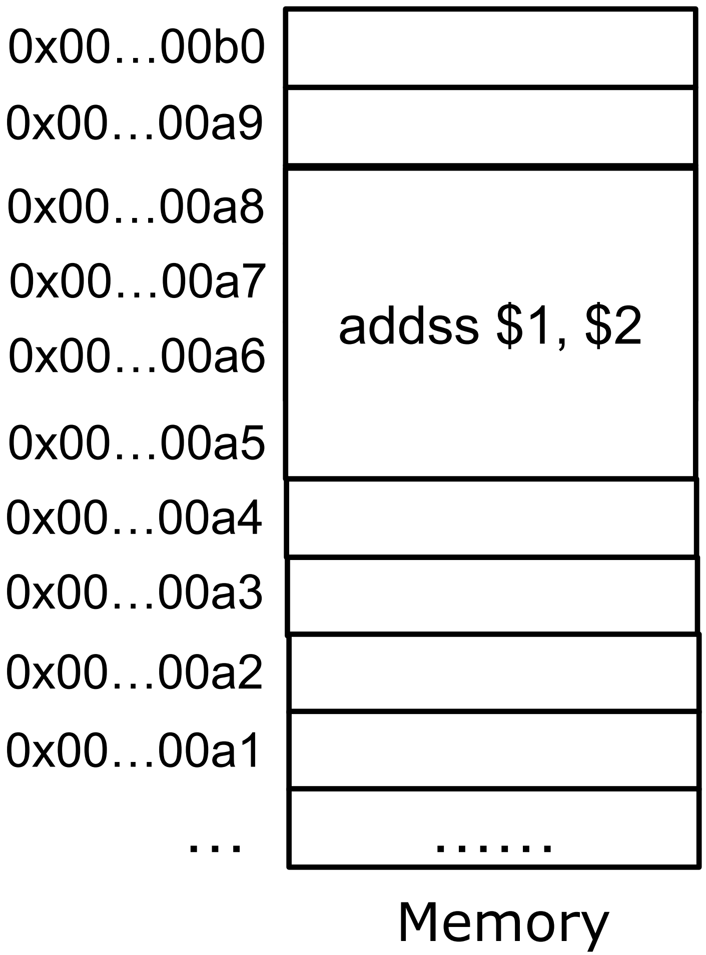
How does CPU know if it is floating point or integers ?

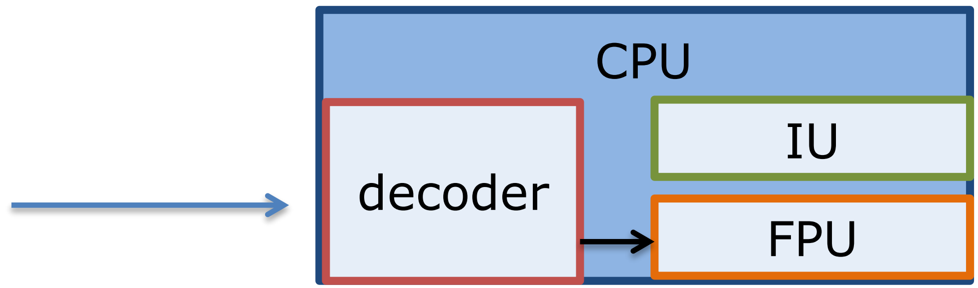
By having specific instruction for floating points operation.


0x00…00b0
0x00…00a9
0x00…00a8
0x00…00a7
0x00…00a6
0x00…00a5
0x00…00a4
0x00…00a3
0x00…00a2
0x00…00a1
…
add $1, $2
……
Memory


CPU
decoder
IU
FPU

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


0x00…00b0
0x00…00a9
0x00…00a8
0x00…00a7
0x00…00a6
0x00…00a5
0x00…00a4
0x00…00a3
0x00…00a2
0x00…00a1
…
addss $1, $2
……
Memory


CPU
decoder
IU
FPU

```
float f = 0.1 + 0.2
```

First lab is out

http://mpaxos.com/teaching/cso18spring/labs