

# **CSC 252: Computer Organization**

## **Spring 2025: Lecture 4**

Instructor: Yuhao Zhu

Department of Computer Science  
University of Rochester

# Announcement

- Programming Assignment 1 is out
  - Details: <https://cs.rochester.edu/courses/252/spring2025/labs/assignment1.html>
  - Due on Feb. 12, 11:59 PM
  - You have 3 slip days

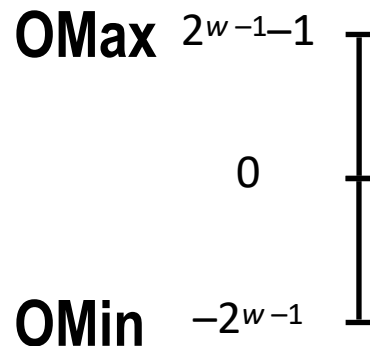
# Announcement

- Programming assignment 1 is in C language. Seek help from TAs.
- TAs are best positioned to answer your questions about programming assignments!!!
- Programming assignments do NOT repeat the lecture materials. They ask you to synthesize what you have learned from the lectures and work out something new.

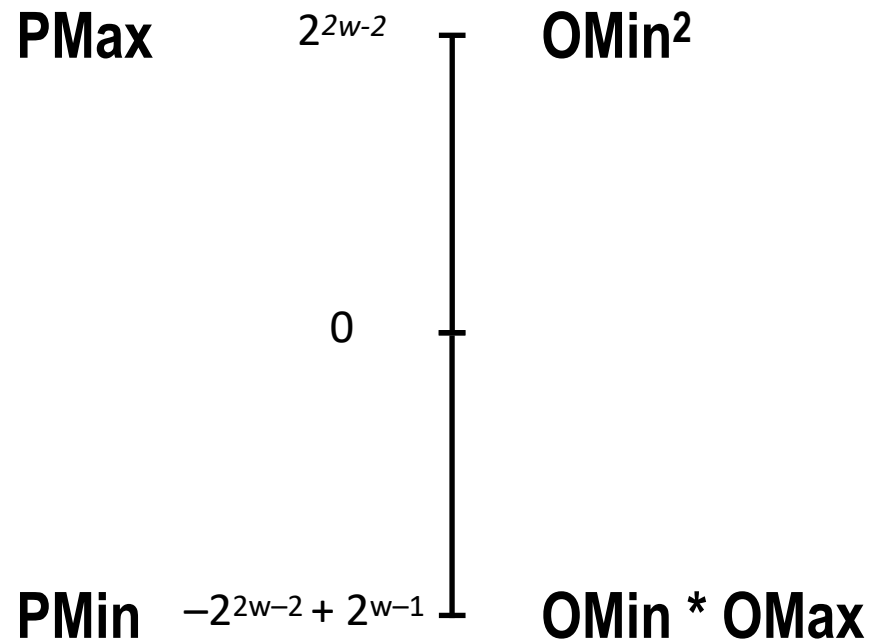
# Multiplication

- Goal: Computing Product of  $w$ -bit numbers  $x, y$
- Exact results can be bigger than  $w$  bits
  - Up to  $2w$  bits (both signed and unsigned)

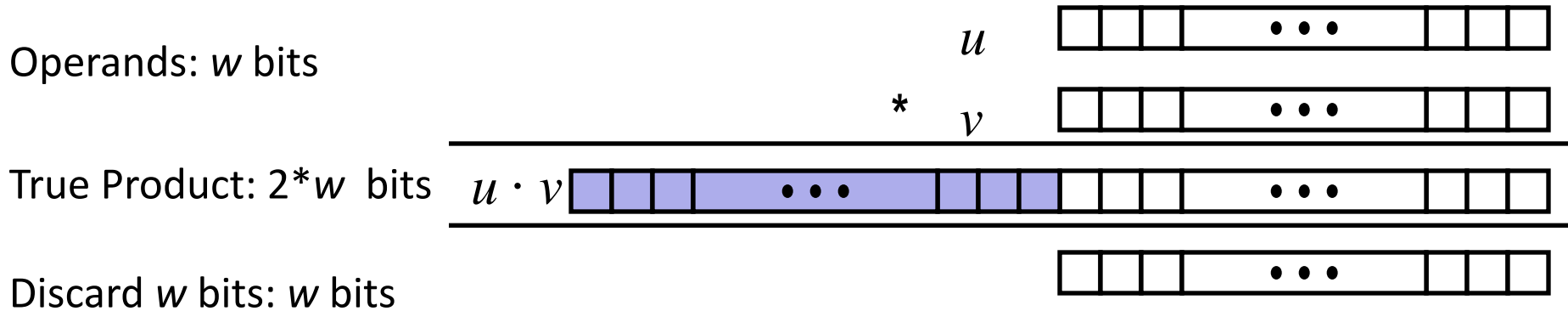
Original Number ( $w$  bits)



Product ( $2w$  bits)



# Unsigned Multiplication in C



- Standard Multiplication Function
  - Ignores high order  $w$  bits
- Effectively Implements the following:

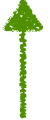
$$\text{UMult}_w(u, v) = u \cdot v \bmod 2^w$$


# Today: Floating Point

- Background: Fractional binary numbers and fixed-point
- Floating point representation
- IEEE 754 standard
- Rounding, addition, multiplication
- Floating point in C
- Summary

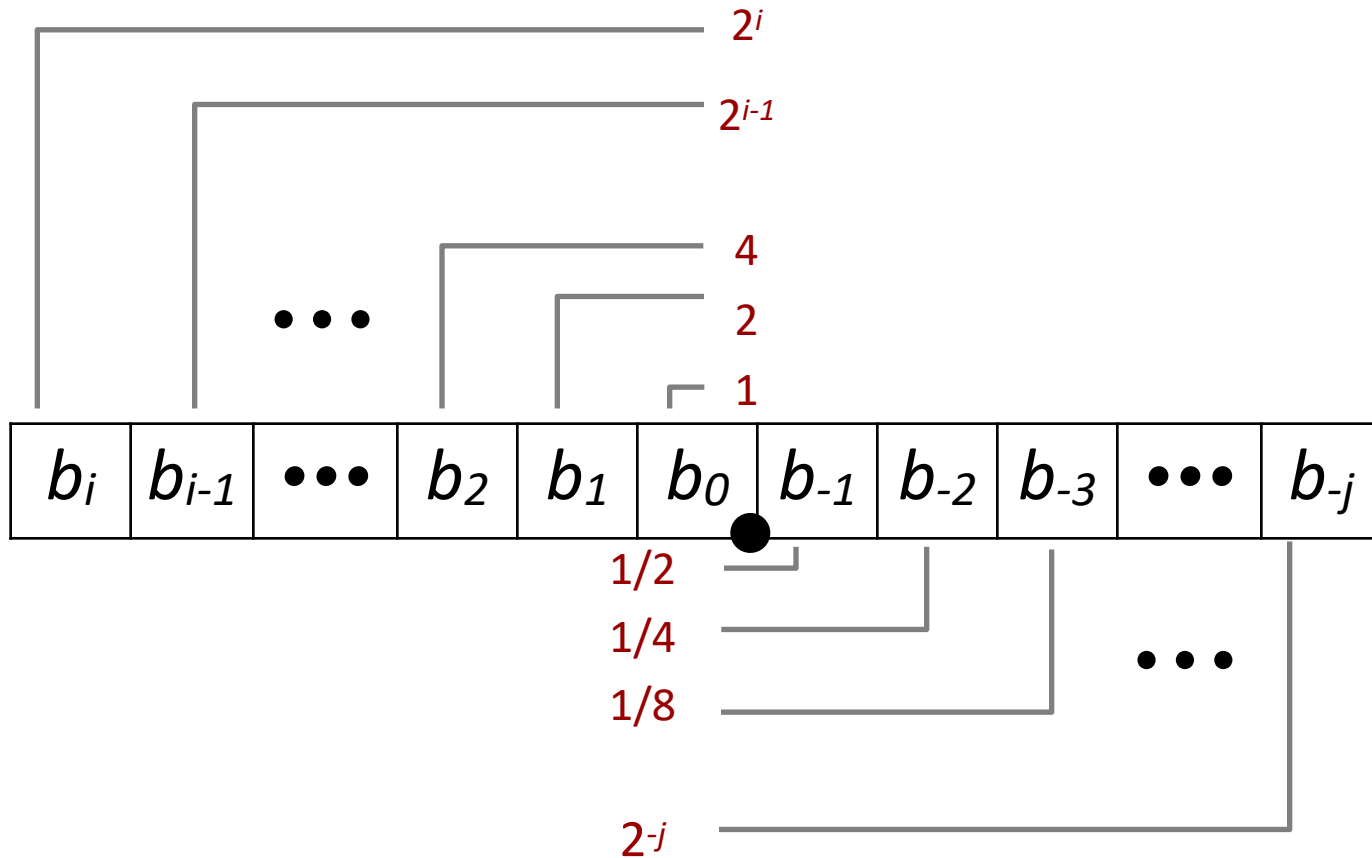
# Recall: Represent Fractions in Binary

- What does  $10.01_2$  mean?
  - C.f., Decimal

$$12.45 = \boxed{1 \cdot 10^1} + 2 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}$$


$$10.01_2 = \boxed{1 \cdot 2^1} + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}$$

$$= 2.25_{10}$$

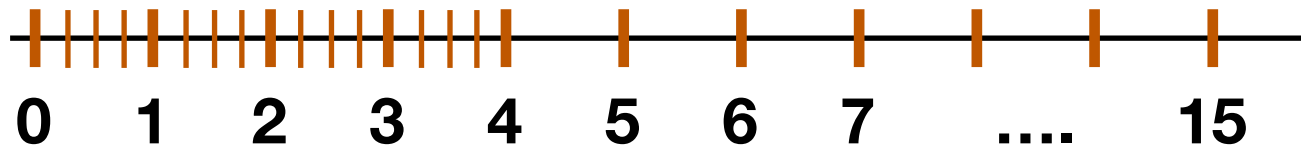
# Fractional Binary Numbers





# Fixed-Point Representation

- Binary point stays fixed
- Fixed interval between representable numbers
  - The interval in this example is  $0.25_{10}$



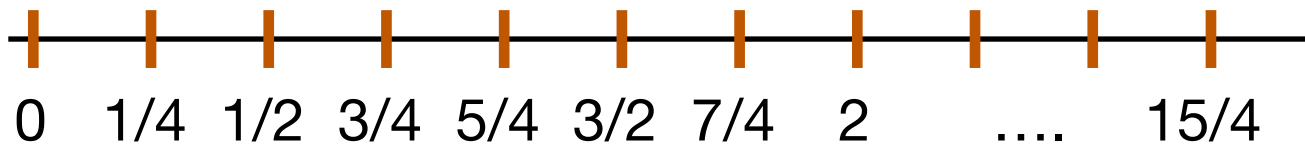
- Still need to remember the binary point, but just once for all numbers, which is implicit given the data type
- Usual arithmetics still work
  - No need to align (already aligned)

Decimal	Binary
0	00.00
0.25	00.01
0.5	00.10
0.75	00.11
1	01.00
1.25	01.01
1.5	01.10
1.75	01.11
2	10.00
2.25	10.01
2.5	10.10
2.75	10.11
3	11.00
3.25	11.01
3.5	11.10
3.75	11.11

# Limitations of Fixed-Point (#1)

- Can exactly represent numbers only of the form  $x/2^k$ 
  - Other rational numbers have repeating bit representations

Decimal Value	Binary Representation
1/3	0.0101010101[01]...
1/5	0.001100110011[0011]...
1/10	0.0001100110011[0011]...



$b_3b_2.b_1b_0$

# Limitations of Fixed-Point (#2)

- Can't represent very small and very large numbers at the same time
  - To represent very large numbers, the (fixed) interval needs to be large, making it hard to represent small numbers
  - To represent very small numbers, the (fixed) interval needs to be small, making it hard to represent large numbers

Unrepresentable  
small numbers

Unrepresentable  
large numbers



# Today: Floating Point

- Background: Fractional binary numbers and fixed-point
- Floating point representation
- IEEE 754 standard
- Rounding, addition, multiplication
- Floating point in C
- Summary

# Primer: (Normalized) Scientific Notation

- In decimal:  $M \times 10^E$ 
  - $E$  is an integer
  - Normalized form:  $1 \leq |M| < 10$

$$\begin{array}{c} \text{M} \times 10^E \quad \leftarrow \text{Exponent} \\ \uparrow \quad \uparrow \\ \text{Significand} \quad \text{Base} \end{array}$$

Decimal Value	Scientific Notation
2	$2 \times 10^0$
-4,321.768	$-4.321768 \times 10^3$
0.000 000 007 51	$7.51 \times 10^{-9}$

# Primer: (Normalized) Scientific Notation

- In binary:  $(-1)^s M 2^E$
- Normalized form:
  - $1 \leq M < 2$
  - $M = 1.b_0b_1b_2b_3\dots$   
Fraction

The diagram illustrates the components of the binary scientific notation formula  $(-1)^s M \times 2^E$ . It features four color-coded labels with arrows pointing to their respective parts in the formula: 'Sign' (orange) points to the superscript  $s$ ; 'Exponent' (green) points to the superscript  $E$ ; 'Significand' (red) points to the mantissa  $M$ ; and 'Base' (blue) points to the base  $2$ . The formula itself is written with  $(-1)$  in black,  $s$  in orange,  $M$  in red,  $\times$  in black,  $2$  in blue, and  $E$  in green.

Binary Value	Scientific Notation
1110110110110	$(-1)^0 1.110110110110 \times 2^{12}$
-101.11	$(-1)^1 1.0111 \times 2^2$
0.00101	$(-1)^0 1.01 \times 2^{-3}$

# Primer: (Normalized) Scientific Notation

- In binary:  $(-1)^s M 2^E$
- Normalized form:
  - $1 \leq M < 2$
  - $M = 1.b_0b_1b_2b_3\dots$   
Fraction

The diagram shows the formula  $(-1)^s M \times 2^E$  with four color-coded labels and arrows pointing to their respective parts: 'Sign' (orange) points to the exponent  $s$  of  $(-1)$ ; 'Exponent' (green) points to the exponent  $E$  of  $2$ ; 'Significand' (red) points to  $M$ ; and 'Base' (blue) points to the base  $2$ .

$$\begin{array}{ccccc} \text{Sign} & & & \text{Exponent} & \\ \downarrow & & & \downarrow & \\ (-1)^s & M & \times & 2^E & \\ \uparrow & \uparrow & & \uparrow & \\ \text{Significand} & & & \text{Base} & \end{array}$$

- If I tell you that there is a number where:
  - Fraction = 0101
  - $s = 1$
  - $E = 10$
  - You could reconstruct the number as  $(-1)^1 1.0101 \times 2^{10}$

# Primer: Floating Point Representation

- In binary:  $(-1)^s M 2^E$

- Normalized form:

  - $1 \leq M < 2$

  - $M = 1.\textcolor{red}{b_0b_1b_2b_3\dots}$   
**Fraction**

- Encoding

  - MSB  $s$  is sign bit  $s$
  - $exp$  field encodes **Exponent** (but not exactly the same, more later)
  - $frac$  field encodes **Fraction** (but not exactly the same, more later)

Diagram illustrating the components of the floating point representation formula  $(-1)^s M \times 2^E$ :

- Sign** (orange) points to the sign bit  $s$  in  $(-1)^s$ .
- Exponent** (green) points to the exponent  $E$  in  $2^E$ .
- Significand** (red) points to the significand  $M$ .
- Base** (blue) points to the base  $2$  in  $2^E$ .





# 6-bit Floating Point Example

$$v = (-1)^s M 2^E$$

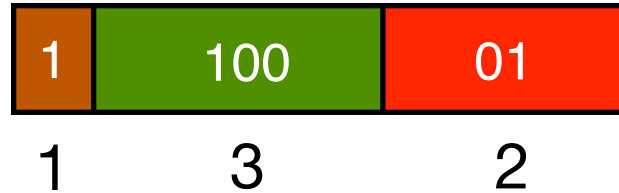


- *exp* has 3 bits, interpreted as an unsigned value
  - If *exp* were *E*, we could represent exponents from **0 to 7**
  - How about negative exponent?
  - Subtract a bias term:  $E = \text{exp} - \text{bias}$  (i.e.,  $\text{exp} = E + \text{bias}$ )
  - bias is always  $2^{k-1} - 1$ , where *k* is number of exponent bits
- Example when we use 3 bits for *exp* (i.e., *k* = 3):
  - bias = 3
  - If *E* = -2, *exp* is 1 (001<sub>2</sub>)
  - Reserve 000 and 111 for other purposes (more on this later)
  - We can now represent exponents from **-2 (exp 001) to 3 (exp 110)**

E	exp
<del>-3</del>	<del>000</del>
-2	001
-1	010
0	011
1	100
2	101
3	110
<del>4</del>	<del>111</del>

# 6-bit Floating Point Example

$$v = (-1)^s M 2^E$$



- *frac* has 2 bits, append them after “1.” to form M
  - *frac* = 10 implies M = 1.10
- Putting it Together: An Example:

$$-10.1_2 = (-1)^1 1.01 \times 2^1$$

↓
↓
↓

E	exp
<del>3</del>	<del>000</del>
-2	001
-1	010
0	011
1	100
2	101
3	110
<del>4</del>	<del>111</del>

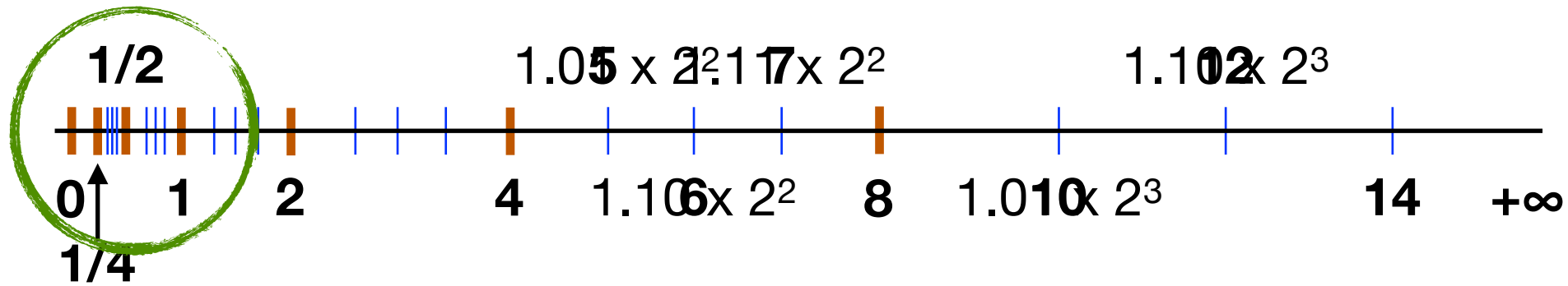
# Representable Numbers (Positive Only)

$$v = (-1)^s M 2^E$$



E	exp	E	exp
<del>-3</del>	<del>000</del>	1	100
-2	001	2	101
-1	010	3	110
0	011	<del>4</del>	<del>111</del>

- Uneven interval (c.f., fixed interval in fixed-point)
  - More dense toward 0, sparser toward infinite
  - Allow encoding small and large numbers at the same time



# Representable Numbers (Positive Only)

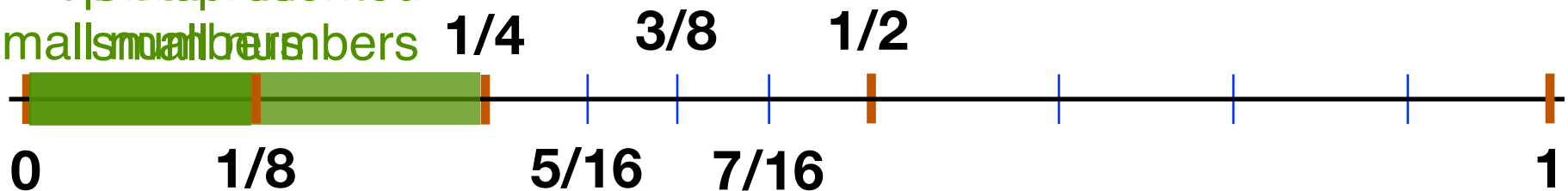
$$v = (-1)^s M 2^E$$



<del>E</del>	<del>exp</del>	E	exp
<del>-3</del>	<del>000</del>	1	100
-2	001	2	101
-1	010	3	110
0	011	<del>4</del>	<del>111</del>

- Always round to 0 is inelegant
- Using 000 for *exp* doesn't solve it either

Unrepresented  
small numbers



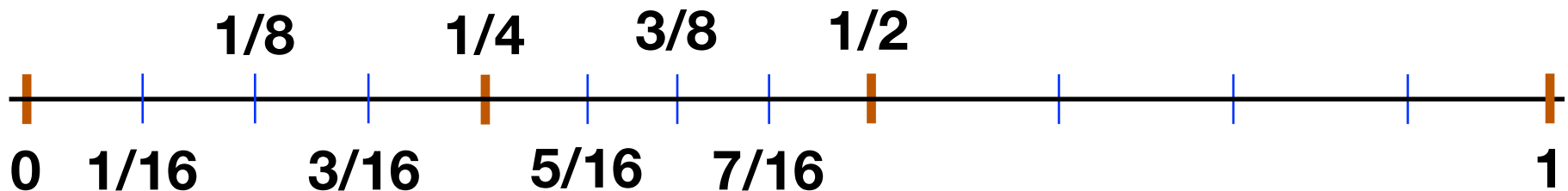
# Subnormal (De-normalized) Numbers

$$v = (-1)^s M 2^E$$



E	exp	E	exp
-3	000	1	100
-2	001	2	101
-1	010	3	110
0	011	<del>4</del>	<del>111</del>

- Idea: Evenly divide between 0 and 1/4 rather than exponentially decreasing **when  $exp = 0$**  (subnormal/denormalized numbers)
- $E = (exp + 1) - bias$  (instead of  $exp - bias$ )
- $M = 0.frac$  (instead of  $1.frac$ )
- Subnormal numbers allow graceful underflow



$$= (-1)^0 0.01 \times 2^{(0+1-3)} = 1/16$$

# Special Values

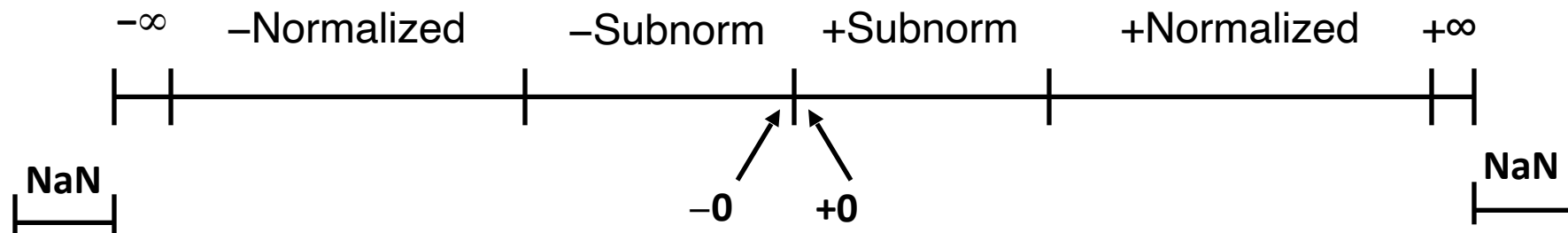
$$v = (-1)^s M 2^E$$



E	exp	E	exp
-2	000	1	100
-2	001	2	101
-1	010	3	110
0	011		<del>111</del>

- There are many special values in scientific computing
  - $\pm \infty$ , Not-a-Numbers (NaNs) (e.g.,  $0 / 0$ ,  $0 / \infty$ ,  $\infty / \infty$ ,  $\sqrt{-1}$ ,  $\infty - \infty$ ,  $\infty \times 0$ , etc.)
- $\text{exp} = 111$  is reserved to represent these numbers
- $\text{exp} = 111$ ,  $\text{frac} = 00$ 
  - $\pm \infty$  (depending on the  $s$  bit). Overflow results.
  - Arithmetic on  $\infty$  is exact:  $1.0/0.0 = -1.0/-0.0 = +\infty$ ,  $1.0/-0.0 = -\infty$
- $\text{exp} = 111$ ,  $\text{frac} \neq 00$ 
  - Represent NaNs

# Visualization: Floating Point Encodings



Infinite Amount of Real Numbers



Finite Amount of Floating Point Numbers

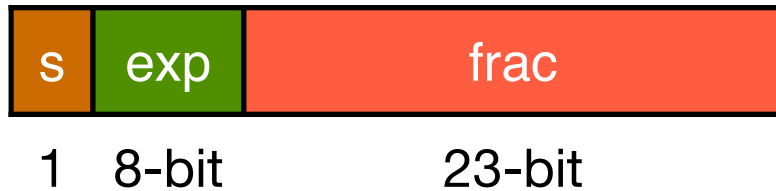
# Today: Floating Point

- Background: Fractional binary numbers and fixed-point
- Floating point representation
- **IEEE 754 standard**
- Rounding, addition, multiplication
- Floating point in C
- Summary



# IEEE 754 Floating Point Standard

- Single precision: 32 bits



- Double precision: 64 bits



# IEEE Floating Point

- **IEEE Standard 754**

- Established in 1985 as uniform standard for floating point arithmetic
  - Before that, many idiosyncratic formats
- Supported by all major CPUs (and even GPUs and other processors)

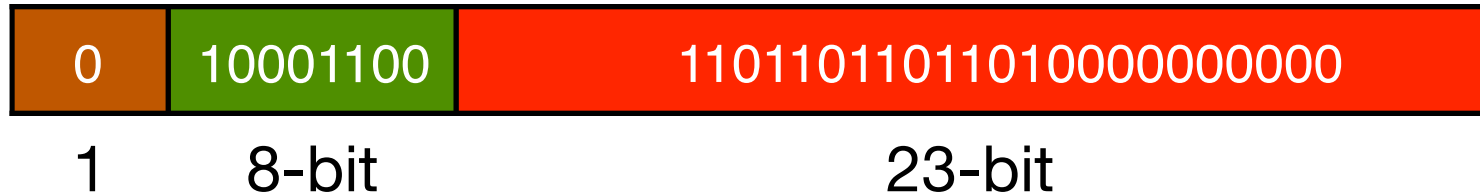
- **Driven by numerical concerns**

- Nice standards for rounding, overflow, underflow
- Hard to make fast in hardware
  - Numerical analysts predominated over hardware designers in defining standard

# Single Precision (32-bit) Example

$$v = (-1)^s M 2^E$$

$$\text{bias} = 2^{(8-1)} - 1 = 127$$



$$\begin{aligned} 15213_{10} &= 11101101101101_2 \\ &= (-1)^0 1.1101101101101_2 \times 2^{13} \end{aligned}$$

$$\text{exp} = E + \text{bias} = 140_{10}$$