Floating point

Jinyang Li

a few slides are adapted from Bryant & O'Hallaron

What we’ve learnt and what’s ahead

- Bit representation of integers
  - Can only represent a finite set of integers (signed, unsigned)
  - Overflow
- Big representation of real numbers
  - Can only represent a finite set of numbers
  - Discretization of real numbers (loss of precision)
  - Overflow
  - Rounding

How to represent real numbers?

- The decimal system

12.34

$$x = \sum_{i=-u}^{i=v} x_i * 10^i$$

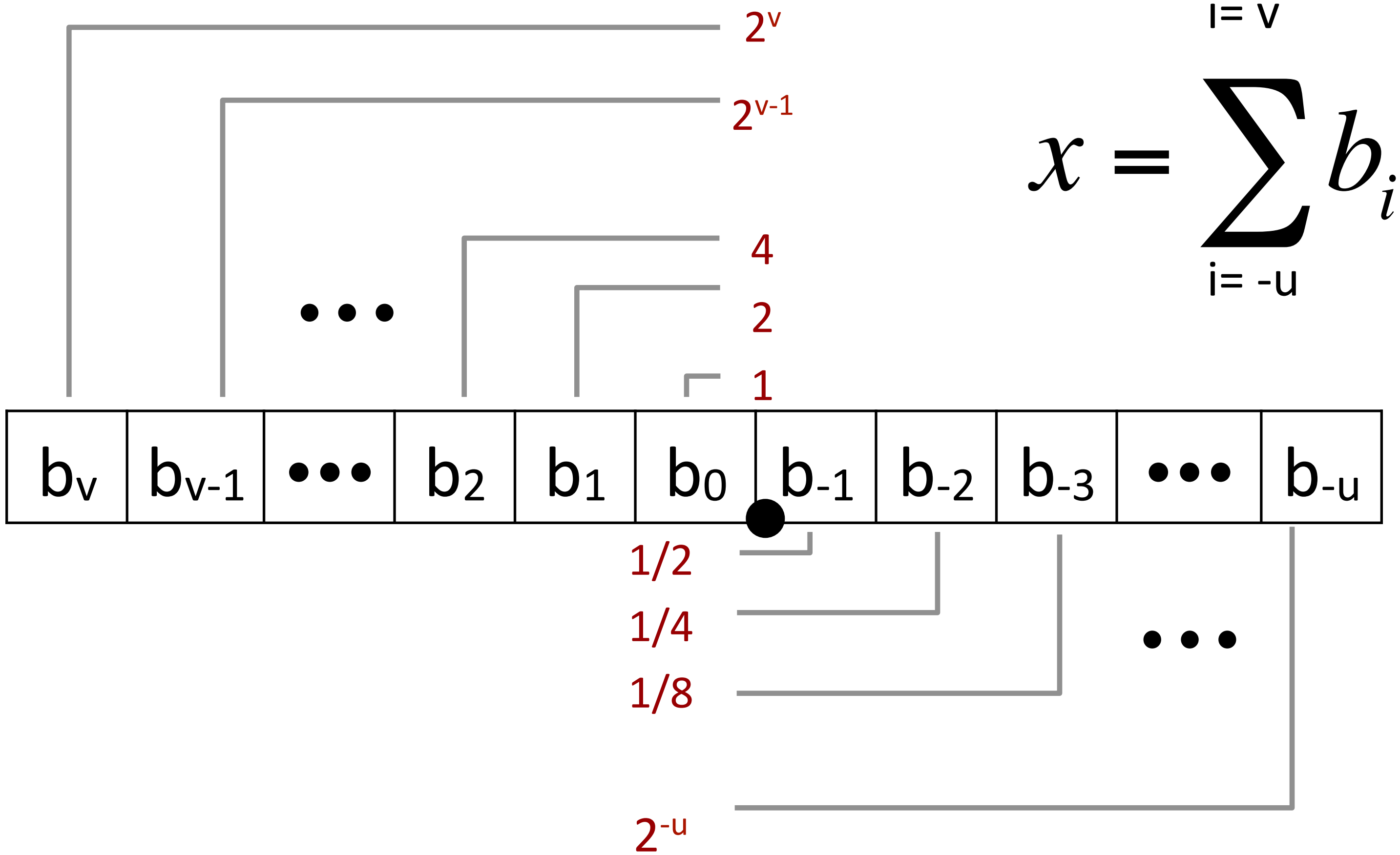
10 (10^1)

1 (10^0)

0.1 (10^-1)

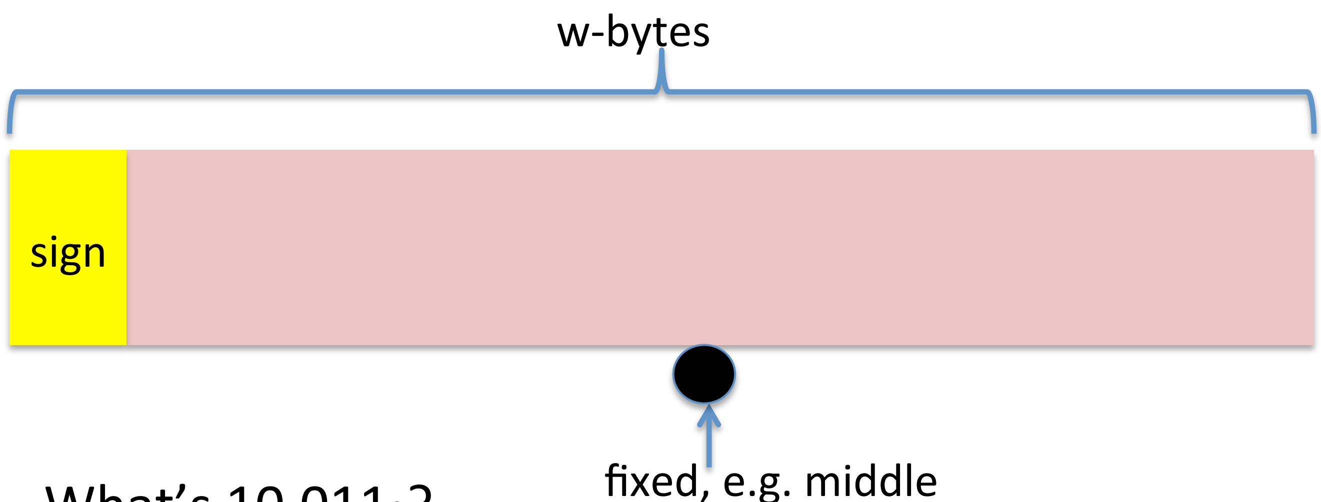
0.01 (10^-2)

Generalize to binary representation



$$x = \sum_{i=-u}^{i=v} b_i * 2^i$$

Naive approach: fixed point representation



- What's 10.011_2?
- 2 3/8
- What's 0.1_{10} in binary?
- 0.00011001....
- What's $0.11111111....111_2$?
- ½ + ¼ + 1/8 + ... 1/(2^16) = 1-(1/2^16)→ 1
- Largest 4-byte fractional numbers?
- 2^15-2^(-16)

Limitations of fixed point

- Useful in certain settings (embedded device)
- Limited range and precision: e.g. using 4-byte
  - largest number 2^15
  - fixed precision 1/(2^-16)
- Not efficient: Small numbers have many zeros after radix points
  - 1/10 **0.0001100110011[0011]**$\ldots_2$

IEEE Floating point

- It's a standard (convention)
  - A group of people get together in the 80s to pick a convention as the standard
- 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

FP: taking inspiration from scientific notation

$$M \cdot 10^{E}$$

Mantissa or significand

- In normalized form, 1 <= M < 10
  - Why normalizing it?
- The most compact way of writing a real number
- Normalized form cannot represent 0!

Floating point representation

sign bit

$$(-1)^{s} \cdot M \cdot 2^{E}$$

encodes E, but not identical to E

encodes M, but not identical to M

single precision

s	exp (8-bit)	frac (23-bit)

Floating point: normalized encoding

s	exp (8-bit)	frac (23-bit)

$$(-1)^{s} \cdot M \cdot 2^{E}$$

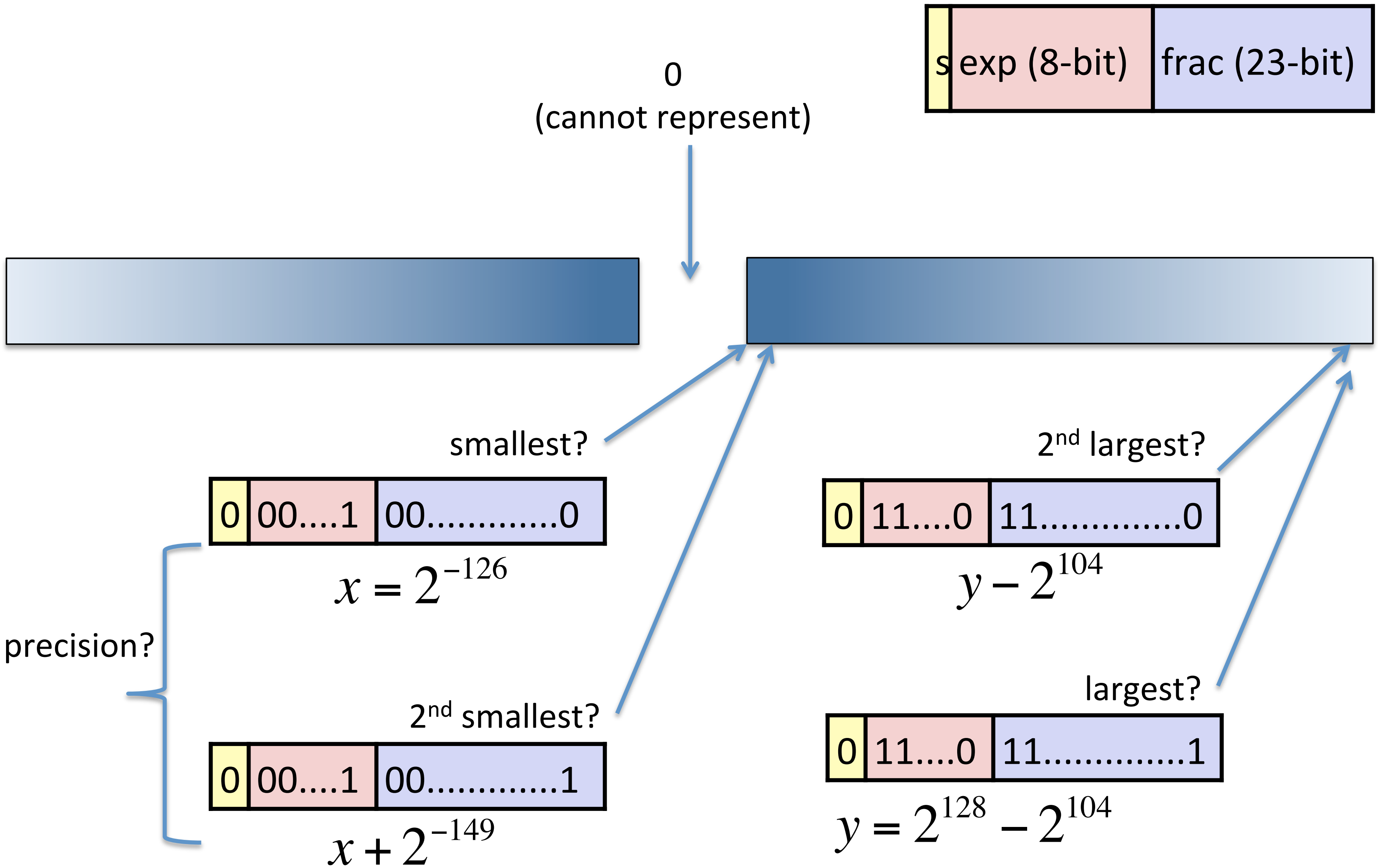
- When: exp ≠ 000…0 and exp ≠ 111…1
- E = exp – bias

  8-bit unsigned   Constant, 127

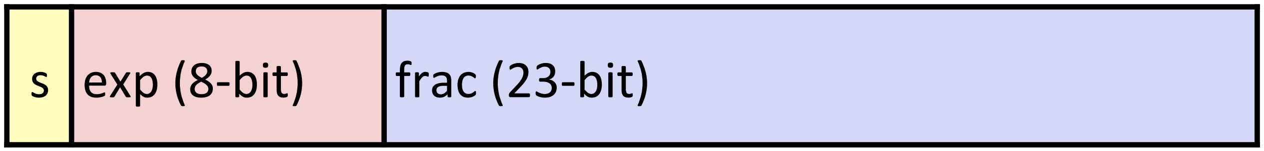
  – Range of E: [-126, 127]
- M = 1.frac

  – Range of M: [1, 2) (normalized)

FP: normalized encoding



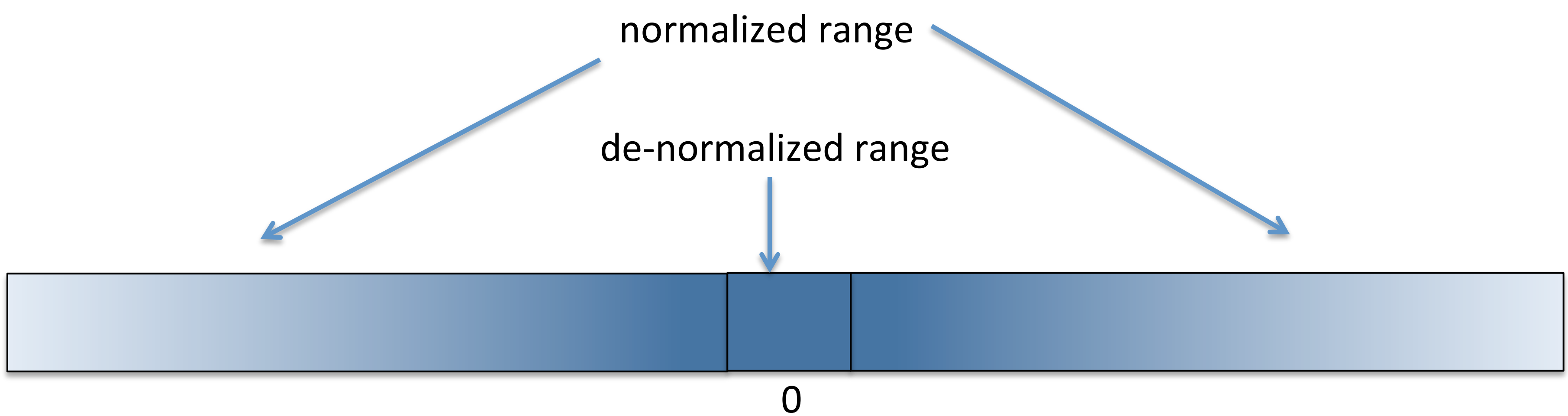
Floating point: de-normalized encoding



$$(-1)^s \cdot M \cdot 2^E$$

- When: exp = 000…0
- E = -126
- M = 0.frac
  – Range of M: [0, 1)
- What's zero like?
  – 0000…000
  – 1000…000

Floating point: denormalized

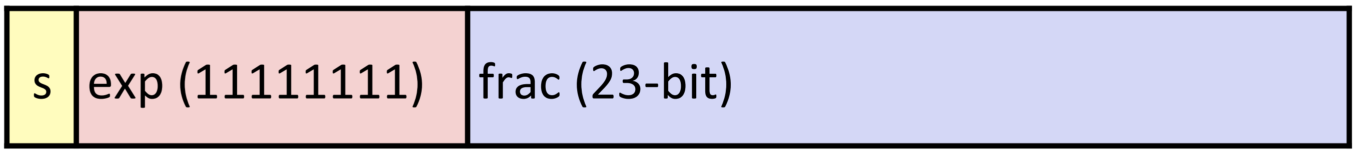


- Smallest de-normalized value?

  2^{-149} | 0 | 00.. | 00............1 |

- How about precision?
  - Fixed at 2^{-149}

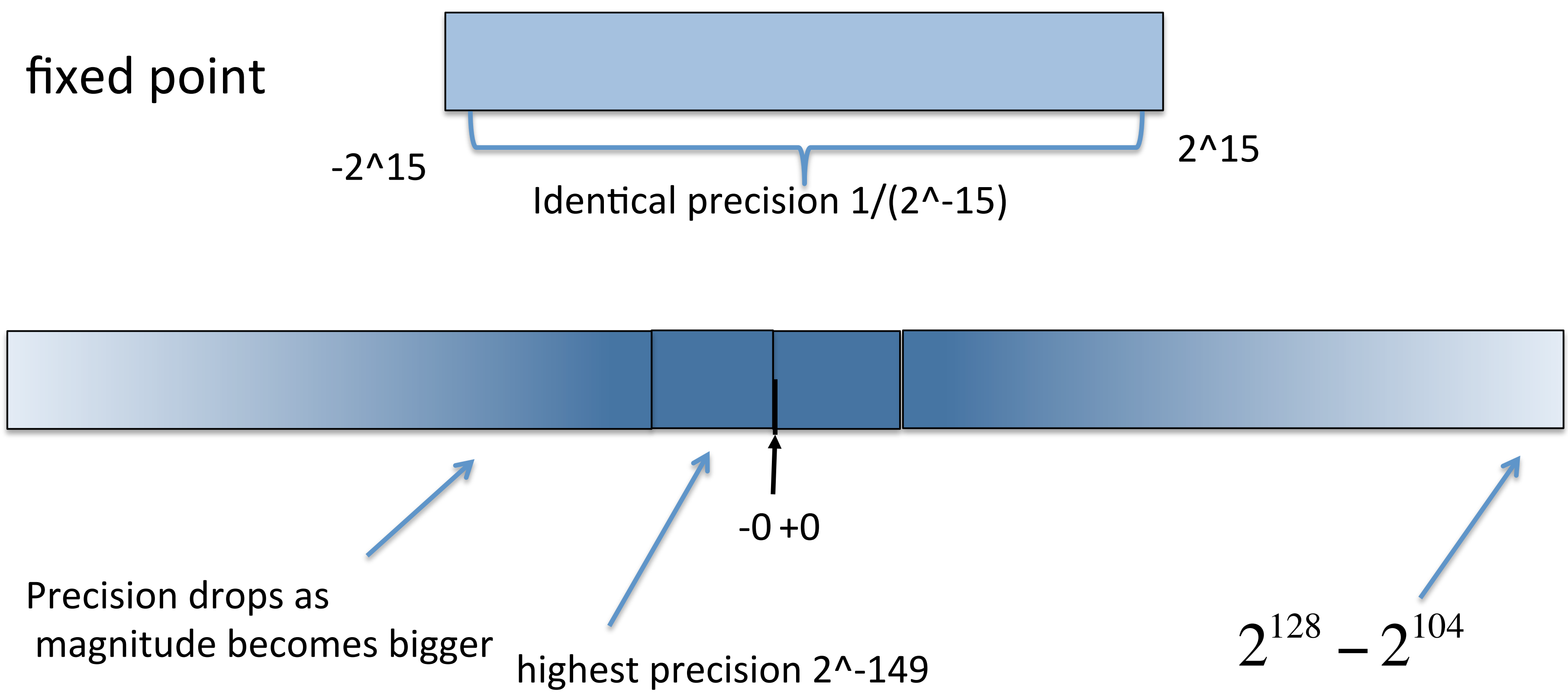
Floating point: special values



- When: `exp` = **111…1**
- ∞: `exp` = **111…1**, `frac` = **000…0**
  - Operation that overflows
  - E.g., 1.0/0.0 = −1.0/−0.0 = +∞, 1.0/−0.0 = −∞
- NaN: `exp` = **111…1**, `frac` ≠ **000…0**
  - Not-a-Number (NaN)
  - E.g., sqrt(–1), ∞ – ∞, ∞ × 0

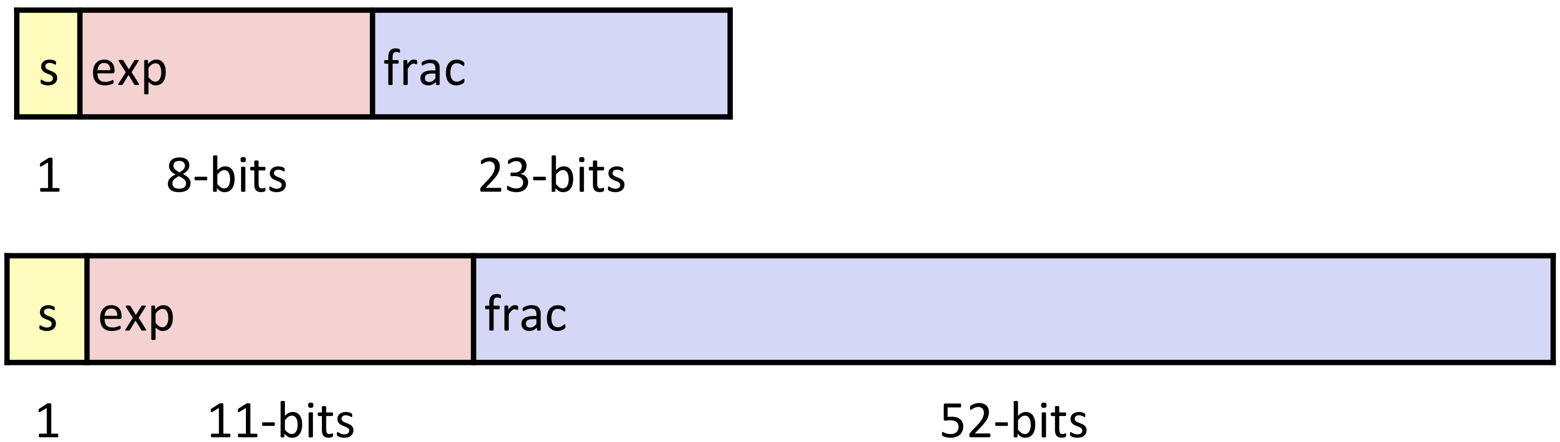
floating point “floats”

- “floating” as radix point is not at a fixed position



Adjacent floats have adjacent bit representations

Single, double precision



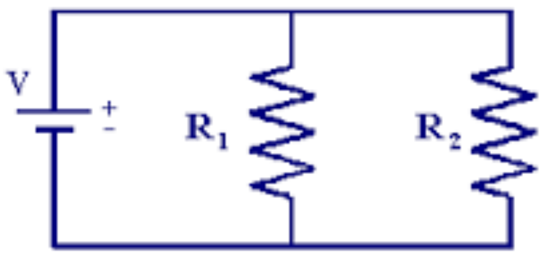
- Single precision
  - highest precision: 2^-149
  - Highest magnitude: ≈ 2^128
- Double precision
  - highest precision: 2^(-52-1022) = 2^(-1074)
  - Highest magnitude: ≈ 2^1025

Floating point operations

- Addition, subtraction, multiplication, division etc.
- 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!

Why divide by zero = ∞?

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



$$\textit{Parallel resistance} = \frac{1}{\frac{1}{R1} + \frac{1}{R2}}$$

If R1 or R2 is 0, overall resistance should be 0

Rounding: round-to-even

- Default Rounding Mode
- Decimal examples:

7.8949999	7.89	(Less than half way)
7.8950001	7.90	(Greater than half way)
7.8950000	7.90	(Half way—round up)
7.8850000	7.88	(Half way—round down)

Round-to-even: binary numbers

- Example: Round to nearest 1/4

Binary	Rounded	Action
10.00011_2		
10.00110_2		
10.00100_2		
10.10100_2		

Floating point addition

- Commutative? x+y == y+x?
- Associative? (x+y)+z = x + (y+z)?
  - Overflow:

```
(3.14+1e10)-1e10 = 0
 3.14+(1e10-1e10) = 3.14
```

  - Rounding
- Every number has an additive inverse?
  - Yes except for ∞ and NaN
- Monoticity? a ≥ b ⇒ a+c ≥ b+c?

Floating point multiplication

- Commutative? x* y == y*x?
- Associative? (x*y)*z = x * (y*z)?
  - Overflow:

  `(1e20*1e20)*1e-20`= `inf`,
  `1e20*(1e20*1e-20)`= `1e20`
  - Rounding
- (x+y)*z = x*z + y*z?
  - `1e20*(1e20-1e20)`= `0.0`, `1e20*1e20 – 1e20*1e20` = `NaN`
- Monoticity? a ≥ b ⇒ a*c ≥ b*c?

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 0.1 rounding error (1991)
  - Ariane 5 explosion due to overflow in converting from double to int (1996)