10

Floating Point Arithmetic

Want: Something like the real numbers... in a computer

Have: Integers, made of bits

$23 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$

How should we even represent fractions?

Idea: Keep going down past exponent zero

$23.625 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$

So: Could store
- a fixed number of bits with exponents >= zero
- a fixed number of bits with exponents < zero

Suppose we use a 64-bit integer, with 32 bits >= 1 and 32 bits < 1.

What is the smallest number we can represent?

2^{-32}

What is the biggest number we can represent?

$2^{31} + 2^{30} + \cdots + 2^1 + 2^0 + 2^{-1} + \cdots + 2^{-32}$

What's our range then?

$\sim 10^{-10} \ldots \quad 10^9$ ← not a lot

What happens if we multiply the largest number by 2?

Option (1): Lie (=truncate the number)
→Option (2): Error

What happens if we divide the smallest number by 2?

Option (1): Lie (=round the number back up)
→ Option (2): Lie (=round the number down to 0)
Option (3): Error

How many accurate decimal digits do we have in a number near 10^9 ?

about 9

How many accurate digits do we have in a number near 10^{-9} ?

$2^{-29} \approx 10^{-9}$ Rounding starts at $2^{-33} \approx 10^{-10}$

→ about one (!)

This is called fixed-point arithmetic, and it's pretty bad.

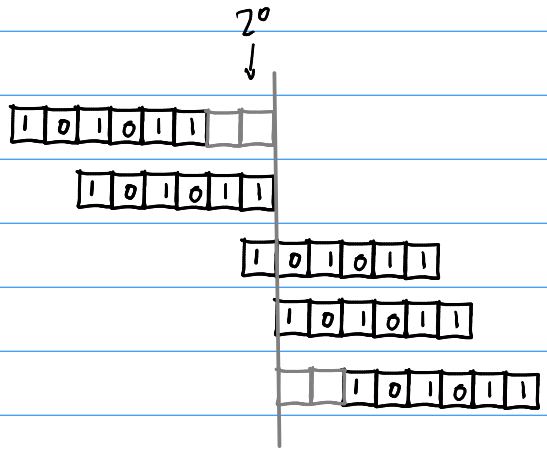
Should be able to do better.

Idea: Set a few bits aside to store the largest exponent. How?

$1 \cdot 2^{213} + 0 \cdot 2^{213-1} + 1 \cdot 2^{213-2} + \dots$ ← 213 is the highest exponent.

$1 \cdot 2^{-6} + 0 \cdot 2^{-6-1} + 1 \cdot 2^{-6-2} +$

What is the:



	exponent?	"significand"?	value?
	7	$(1.01011)_2 = 1.34375$	$1.34375 \cdot 2^7$
	5	— " —	$1.34375 \cdot 2^5$
	0	— " —	$1.34375 \cdot 2^0$
	-1	— " —	$1.34375 \cdot 2^{-1}$
	-3	— " —	$1.34375 \cdot 2^{-3}$

In our 64-bit example:

- 1 bit for sign (+/-)
- 11 bits for largest exponent
- 52 bits for "bits"

Exponent ranges from -1022 to 1023

This is called "double precision".

What is (very roughly) the smallest number we can represent?

$2^{-1022} \approx 10^{-308}$

lies: - denormals
- "implicit one"
- "missing exponents"

What is (very roughly) the largest number we can represent?

$2^{1023} \approx 10^{307}$

How many accurate decimal digits do we have in the largest representable number?

Rounding starts at $2^{1023-52} \approx 10^{292}$

$307 - 292 \approx 15$ digits

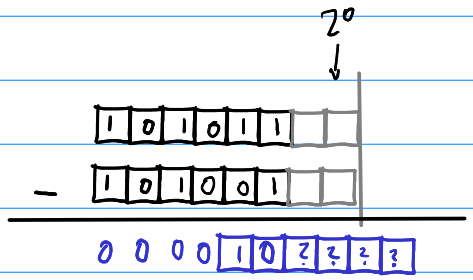
How many accurate decimal digits do we have in the smallest representable number?

Rounding starts at $2^{-1022-52} \approx 10^{-324}$

$-308 - (-324) \approx 15$ digits

Same relative accuracy for numbers of every magnitude: Yay!

So what could possibly go wrong?



How many accurate (binary) digits are there in the above result?

2/6 ← not good

Called "catastrophic cancellation"

Demo: Catastrophic Cancellation

Demo: Picking apart a floating point number

Demo: Floating point vs. program logic