

## More technical matters: Floating-pt. arithmetic

What guarantees the "correct" solution of any numerical algorithm?

- ① Bug-free code (obviously)
- ② Convergent numerical algorithm (in infinite-precision)
- ③ "Stable" computers (recall  $\cos(2\pi 10^{20})$ ?)

① & ② are "human" problems

③ is a "computer" problem

[ Intel Flaw incorrectly computing quotients  
⇒ \$420 million dollars

[ Ariane 5 rocket - programmers didn't allocate enough memory  
⇒ all arrays at least 10000 long!

The question you should be asking yourself is:  
How do computers store numbers?

→ Binary / base-2 arithmetic

Decimal:  $10 = 1 \cdot 10 + 0 \cdot 1$

Binary:  $1010 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$

Binary addition:

$$\begin{aligned}23 &= 16 + 7 \\ &= 16 + 4 + 2 + 1 \\ &= 2^4 + 2^2 + 2^1 + 2^0 \\ &= (10111)_2\end{aligned}$$

$$\begin{aligned}17 &= 16 + 1 \\ &= 2^4 + 2^0 \\ &= (10001)_2\end{aligned}$$

$$\begin{aligned}17 + 23 &= 40 = 32 + 8 \\ &= 2^5 + 2^3 \\ &= (101000)_2\end{aligned}$$

$$\begin{array}{r}10111 \\ + 10001 \\ \hline \boxed{101000}\end{array}$$

Binary decimals:

$$\frac{7}{2} = 3.5 \text{ in decimal notation}$$

In binary notation:

$$\begin{aligned}3 &= (11)_2 \\ .5 &= (0.1)_2 \\ &\quad \quad \quad \uparrow \frac{1}{2} \\ .25 &= \frac{1}{4} = (0.01)_2 \\ &\quad \quad \quad \quad \quad \quad \uparrow \frac{1}{2^2}\end{aligned}$$

Each 0 or 1 is a "bit".

Fixed representation:  $xxxx.xxxx$  8 bit fixed

Not how computers store numbers

Floating point representation (like scientific notation)

$$\pm m \times 2^E, \quad m \in [1, 2)$$

Example:  $23 = (10111)_2$

$$= (1.0111 \times 2^4)$$

Every multiplication by 2 moves the decimal over by one.

$$\frac{1}{8} = 0.001 = 1.0 \times 2^{-3}$$

How many bits does a computer use?

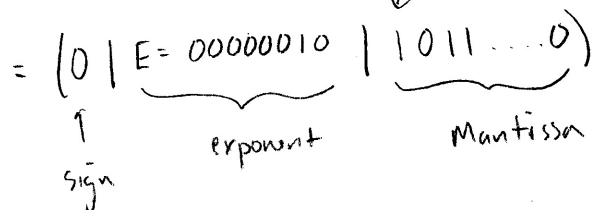
single precision "word" : 32 bits

1 bit for sign

8 bits for exponent

23 bits for mantissa (significand)

Ex:  $5.5 = 1.011 \times 2^2$



always 1 (can improve, except for 0)  
hidden-bit rep.

Non-repeating ~~decimals~~ binary are "floating-point numbers"

What is the precision of this?

Machine precision is the distance between  
1 and the next fp number. In this

case,

$$1 = (0 | E=0 | 1.0 \dots 0)$$

23 bits  
↑  $\frac{1}{2^{23}}$

$$1 + 2^{-23} = (0 | E=0 | 1.0 \dots 01)$$

$$\hookrightarrow \approx 1.2 \times 10^{-7}$$