IEEE-754

Junseo Shin

April 11, 2024

Outline

Introduction

Should We Trust Computers?

- The error in floating point is a form of “noise” in the data.
- The computer automatically “preprocesses” the data in a way that is not always what you want.

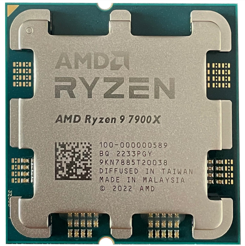
How does a computer operate on numbers?

$$178956970 - 178957034$$

How does a computer operate on numbers?

178956970 — 178957034

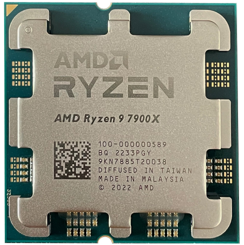
⇓



How does a computer operate on numbers?

$$178956970 - 178957034$$

$$\Downarrow$$



$$\Downarrow$$

$$= -64$$

Adding/Subtracting two 32 bit numbers

$$
\begin{array}{rrrrrrrrr}
 & (0)000 & 1010 & 1010 & 1010 & 1010 & 1010 & 1010 & 1010 \\
- & (0)000 & 1010 & 1010 & 1010 & 1010 & 1010 & 1110 & 1010 \\
\hline
 & (1)000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0100 & 0000
\end{array}
$$

Adding/Subtracting two 32 bit numbers

$$\begin{array}{r} (0)000\ 1010\ 1010\ 1010\ 1010\ 1010\ 1010\ 1010 \\ -\ (0)000\ 1010\ 1010\ 1010\ 1010\ 1010\ 1110\ 1010 \\ \hline (1)000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100\ 0000 \end{array}$$

$$= -2^6 = -64$$

Multiplying two binary numbers

$$170 \times 170 = 28900$$

Multiplying two binary numbers

$$170 \times 170 = 28900$$

$$
\begin{array}{ccccccccccccccc}
 & & & & & & & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
 & & & & & & \times & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
\cline{7-15}
 & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & \\
 & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & \\
 & & & & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & & & \\
 & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & & & \\
 & & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & & & & & \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & & & & & \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & & & & & & & \\
\hline
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0
\end{array}
$$

Dividing two binary numbers

```
                        1 0 1 0 1 0 1 0
          ______________________________
10101010 ) 1 1 1 0 0 0 0 1 1 1 0 0 1 0 0
           1 0 1 0 1 0 1 0
           ---------------
           0 0 1 1 0 1 1 1 1 1
               1 0 1 0 1 0 1 0
               ---------------
               0 0 1 1 0 1 0 1 0 0
                   1 0 1 0 1 0 1 0
                   ---------------
                   0 0 1 0 1 0 1 0 1 0
                       1 0 1 0 1 0 1 0
                       ---------------
                       0 0 0 0 0 0 0 0
```

Problems with Binary Integers

No way to represent rational numbers

Problems with Binary Integers

No way to represent rational numbers

- Division is not accurate

$$3/2 = 1.5 \rightarrow 101/10 = 1.\cancel{1} = 1$$

Problems with Binary Integers

No way to represent rational numbers

- Division is not accurate

$$3/2 = 1.5 \rightarrow 101/10 = 1.\cancel{1} = 1$$

- Real-Valued Functions (e.g. $\sin(x), e^x, \sqrt{x}$)

IEEE-754: Floating Point Arithmetic

From Decimal to Floating Point

Scientific Notation: $n \times 10^m$

- $1234 = 1.234 \times 10^3$
- $0.01234 = 1.234 \times 10^{-2}$

From Decimal to Floating Point

Scientific Notation: $n \times 10^m$

- $1234 = 1.234 \times 10^3$
- $0.01234 = 1.234 \times 10^{-2}$

Floating Point: (sign, significand, exponent)

$$(-1)^s \times m \times 2^e$$

Converting Decimal to Floating Point

Decimal $\rightarrow$ Binary $\rightarrow$ Floating Point

$$\begin{aligned} 7.45 &\rightarrow 7 + 0.45 \\ &\rightarrow 111.01110011001100110011001100110011\ldots \end{aligned}$$

1 bit for sign, 8 bits for exponent, 23 bits for significand

Converting Decimal to Floating Point

Decimal $\rightarrow$ Binary $\rightarrow$ Floating Point

$$
\begin{aligned}
7.45 &\rightarrow 7 + 0.45 \\
&\rightarrow 111.01110011001100110011001100110011\ldots
\end{aligned}
$$

$$
\overbrace{\boxed{+}}^{\text{sign}}\ \underbrace{1.1101\ldots}_{\text{significand}} \times 2^{\overbrace{\boxed{2}}^{\text{exponent}}}
$$

1 bit for sign, 8 bits for exponent, 23 bits for significand

$$
\boxed{0}\boxed{1000\ 0001}\boxed{1101\ 1100\ 1100\ 1100\ 1100\ 110}0110\ldots
$$

Error in Floating Point

Machine Epsilon

$$\epsilon = 2^{-(p-1)}$$

- Single Precision: $p = 24$: $\epsilon = 2^{-23} \approx 1.19 \times 10^{-7}$

$$1 + \epsilon = 1.0000\ 0000\ 0000\ 0000\ 0000\ 001$$

- Double Precision: $p = 53$: $\epsilon = 2^{-52} \approx 2.22 \times 10^{-16}$

Python Example

```
python3
>>> 0.1 + 0.3
0.30000000000000004
>>> 0.3 / 0.10
2.9999999999999996
```

Simulating Larger Precision

GNU Multiple Precision Arithmetic Library (GNU MPFR)

- Used in Mathmatica
- Follows IEEE-754 standard but with arbitrary precision
- Does this solve the problem?

Stirling's Approximation vs Factorial

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^{n}$$

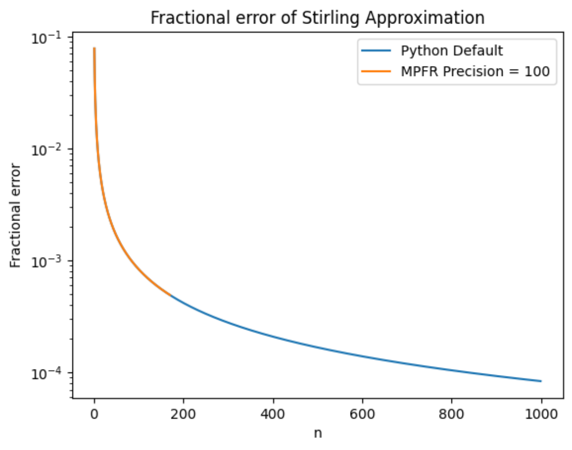
Stirling's Approximation vs Factorial

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Python Default (Float64, $p = 53$) vs MPFR ($p = 100$)

- $n = 171 \to$ Overflow in Float64
- At $n = 100$, Fractional Error of 0.000833 for both!
- Fractional Error of 10^{-6} at $n = 83334$

Float64 vs MPFR



Patriot Defense Missile Failure

- February 25, 1991: Patriot missile defense system failed to intercept SCUD missile
- 28 soldiers died

Simulating Precision Loss

- Integer to 24 bit floating point

Hours	Seconds	Calculated Time (Seconds)	Inaccuracy (Seconds)	Approximate Shift In Range Gate (Meters)
0	0	0	0	0
1	3600	3599.9966	.0034	7
8	28800	28799.9725	.0275	55
20[a]	72000	71999.9313	.0687	137
48	172800	172799.8352	.1648	330
72	259200	259199.7528	.2472	494
100[b]	360000	359999.6667	.3433	687

[a]Continuous operation exceeding about 20 hours—target outside range gate

[b]Alpha Battery ran continuously for about 100 hours

Reference

Ieee standard for floating-point arithmetic.
IEEE Std 754-2019 (Revision of IEEE 754-2008), pages 1–84, 2019.

David Goldberg.
What every computer scientist should know about floating-point arithmetic.
ACM Computing Surveys, 23:5–48, Mar 1991.

[2] [1]

- https://floating-point-gui.de/
- Patriot Missile Defense Software Problem Led to System Failure at Dhahran, Saudi Arabia
- Floating Point Converter