Floating Point Arithmetics + Posits (Part 1)

by Uğur Çayoğlu

05. February 2020

STEINBUCH CENTRE FOR COMPUTING (SCC) &
INSTITUTE FOR METEOROLOGY AND CLIMATE RESEARCH (IMK-ASF)

Agenda

- Floating Point Arithmetics (Part 1)
- Posits (Part 2)

Take-home message of today!

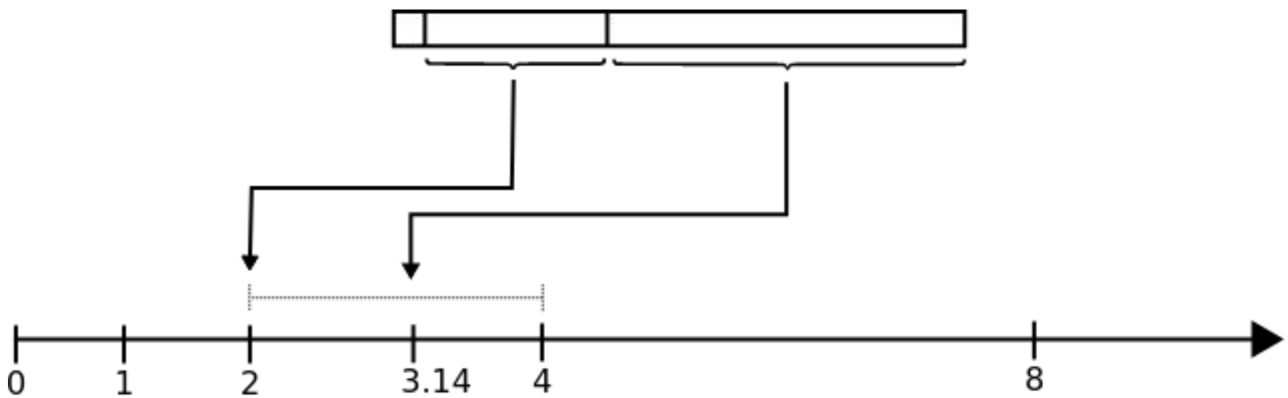
```
0.1 + 0.2 != 0.3
```

- The correct answer is not of interest
    - **Accept life** and get over it!
    - “If life gives you lemons, …”
    - Be consistent and **repeat mistakes!**
- Repeat mistakes **within one build**, across **multiple builds**, and across **multiple platforms**

Source_1

Structure and Format of Floating-Point Data

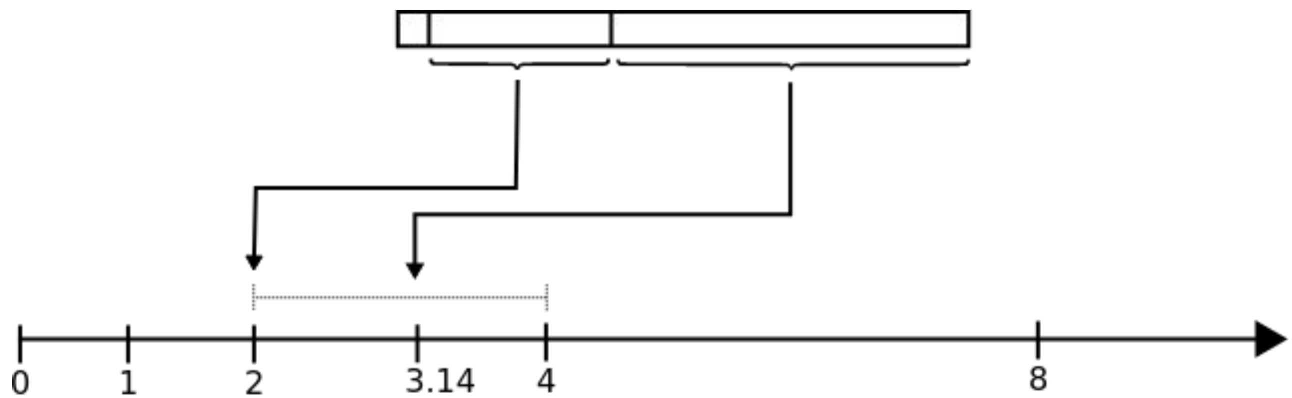
- Sign (1) + Exponent (8) + Mantissa (23) = IEEE Floating-Point Data (32)
  - Sign = Negative/Positive
  - Exponent = Power of two
  - Mantissa = Fraction (1/2^23 = 1/8,388,608 = eps)
- `x = (-1)^s * 2^(exp-127) * (1+f)`



Source_2

Some Weird Properties of Floating-Point Data

- There are some special values (i.e. denormals):
  - +infinite, -infinite,
  - +zero, -zero,
  - NaN (there are actually 16,777,216 NaN values (2 * mantissa))



Source_2

Guarantees while using Floating-Point Arithmetics

These operations are determined and will produce **reproducible** results…

- Addition
- Subtraction
- Division
- Multiplication
- Square root computation

… if we use the same **rounding mode**, same **inputs**, same **global settings** and same destination **precision**

STEINBUCH CENTRE FOR COMPUTING (SCC)
INSTITUTE FOR METEOROLOGY AND CLIMATE RESEARCH (IMK-ASF)

A Set of Problems Ordered by Your Likelihood of Experiencing it

1. Rounding modes
2. Composition
3. Per processor code
4. Precision modes
5. Denormals
6. Compiler differences / uninitialised data / transcendentals / square root estimates / conversions

Please check the trusted documentation of your programming language and compiler of choice for possibilities to prevent these problems

Please check your code before you blame floating point arithmetics…

- Check your **algorithms and data structures** before you blame floating point arithmetics about missing determinism

  - Random number generator working properly?
  - Flexible timing with thread scheduling?
  - Simulation time variable?
  - Code with undefined behaviour?

Source_3

Five Rounding modes

- Rounding mode for values in between two floating point values
- Runtime check necessary
- Thread bound
- C/C++: `#include <fenv.h>`

	11.5	**12.5**	**- 11.5**	**- 12.5**
TO_NEAREST_EVEN	12	12	-12	-12
TO_NEAREST_AWAY_ZERO	12	13	-12	-13
TOWARD_ZERO	11	12	-11	-12
PLUS_INFINITY	12	13	-11	-12
MINUS_INFINITY	11	12	-12	-13

Composition

- Optimization error
- Reordering of operations based on register/cache availability
- Throughput vs precision
- Use parenthesis for operation ordering
- C/C++: `fp/fast vs fp/precise`

```
a + b + c = ?

a + (b + c) != (a + b) + c
```


STEINBUCH CENTRE FOR COMPUTING (SCC)
INSTITUTE FOR METEOROLOGY AND CLIMATE RESEARCH (IMK-ASF)

Precision Modes

- Different precisions for intermediate representations
- In most common architectures (i.e. Intel, AMD)
  this is 24-, 53-, 64-bit for representing mantissa
- Depends on the feature set one uses
  - C78: Floats are doubles where fraction part is filled with zeros
  - C++98: Only if one of the operands is a double
  - C99: “Evaluation type may be wider than semantic type”

… basically it is up to the compiler and the compilation flags used.

Per processor code

- Depending on the feature optimizations available the results might differ
- SSE = 32bit intermediate precision
- SSE2 = 64bit intermediate precision
- Special instructions
  - i.e. fmadd

```
fmadd(a,b,c) = a * b + c

a * b + c * d = ?

t = c * d
fmadd(a,b,t) != a * b + c * d
```


Ugur Cayoglu

STEINBUCH CENTRE FOR COMPUTING (SCC)
INSTITUTE FOR METEOROLOGY AND CLIMATE RESEARCH (IMK-ASF)

Denormals

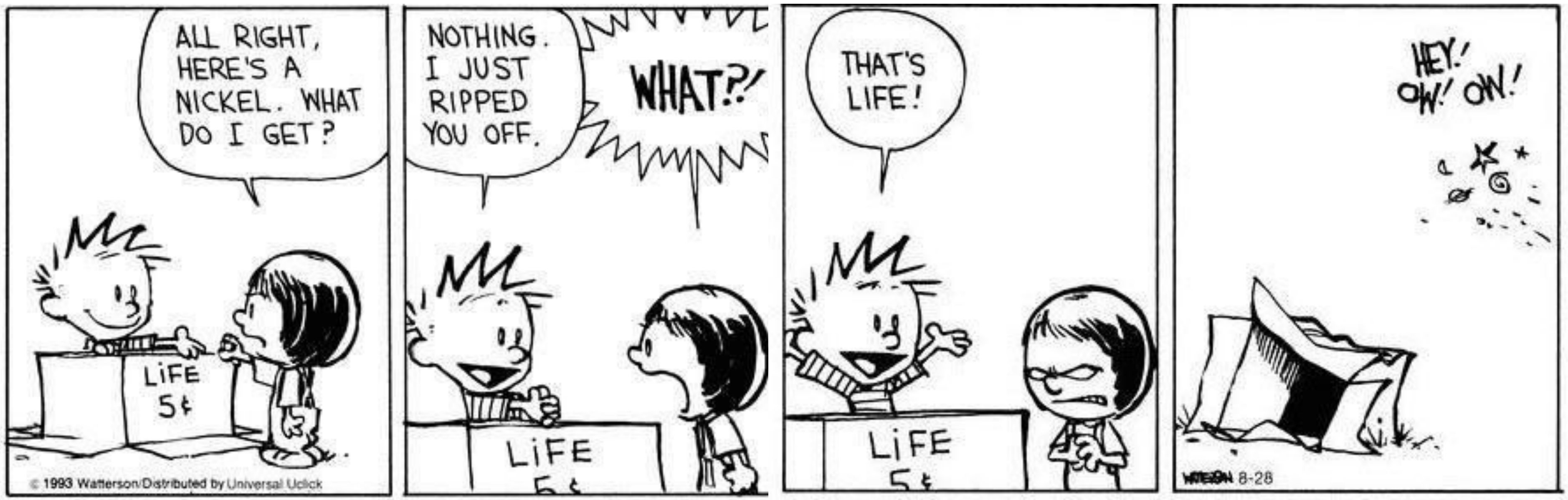
- Special values from 0 to 1
- `x = (-1)^s * 2^(exp-127) * (1+f)`
  - Smallest possible value is for f = 0, exp = 1 (since exp=0 is +zero)
  - 2^-126 = 1.175E-38 (realmin)
  - 2^-126 * 2^-23 = 1.4013e-45 (eps*realmin)
- All values between realmin and eps*realmin are called **denormals**
- Computations between these values are not hardware supported
- There is a flag to turn calculation of denormals off
  - All values below realmin will be set to 0
- Turn off if you want speed up, but results will change
  (> better don't)

Compiler differences / transcendentals / square root estimates / conversions

- Compile time/Runtime calculations (e.g. sin, tan, cos)
- Compiler: gcc, clang, visual c++, ...
- Square root estimates
  - There are square root estimates that are implemented by hardware manufacturers (rcpss, rcpps, rsqrtps, rsqrtss)
- Transcendentals like sin, cos, tan, pi, e can be hardware supported and might differ based on manufacturer
- Conversions e.g. printing commands might differ in output depending on operation order

STEINBUCH CENTRE FOR COMPUTING (SCC)
INSTITUTE FOR METEOROLOGY AND CLIMATE RESEARCH (IMK-ASF)

That's all folks. Thank you…



https://0.30000000000000004.com/

Source_4

Resources

Main source for presentation: http://randomascii.wordpress.com
What Every Computer Scientist Should Know About Floating-Point Arithmetic
https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html
Float Converter https://www.h-schmidt.net/FloatConverter/IEEE754.html

Pics:

1. https://i.imgur.com/SgnwoIn.jpg
2. https://fabiensanglard.net/floating_point_visually_explained/floating_point_window_pi.svg
3. https://static.inspiremore.com/wp-content/uploads/2017/01/24092529/Screen-Shot-2017-01-24-at-3.10.33-PM.png
4. https://twitter.com/calvinn_hobbes/status/528720685771026432

STEINBUCH CENTRE FOR COMPUTING (SCC)
INSTITUTE FOR METEOROLOGY AND CLIMATE RESEARCH (IMK-ASF)