Floating Point

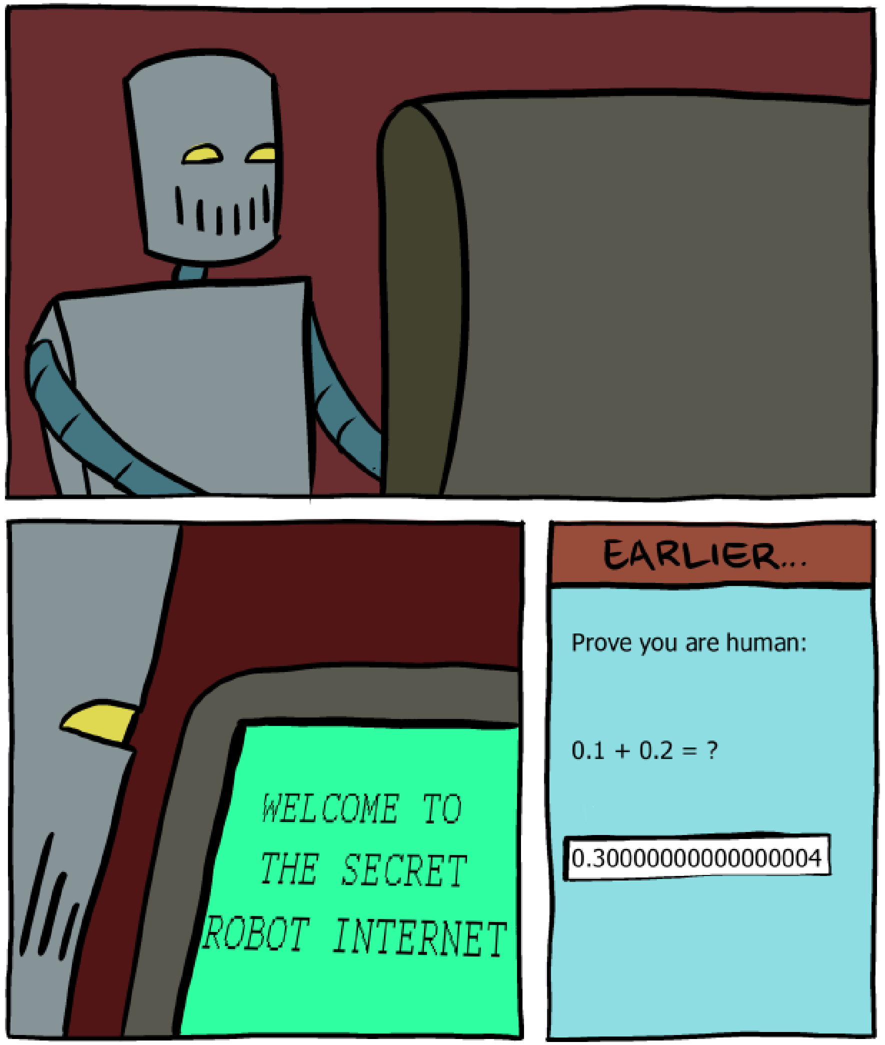
CSE 351 Winter 2024

Instructor:

Justin Hsia

Teaching Assistants:

Adithi Raghavan
Aman Mohammed
Connie Chen
Eyoel Gebre
Jiawei Huang
Malak Zaki
Naama Amiel
Nathan Khuat
Nikolas McNamee
Pedro Amarante
Will Robertson



http://www.smbc-comics.com/?id=2999

Relevant Course Information

- HW4 due tonight, HW5 due Friday, HW6 due Monday

- Lesson questions are graded on *completion*
  - Don’t change your answer afterward; misrepresents your understanding

- Lab 1a final late submissions due tonight at 11:59 pm
  - Submit `pointer.c` and `lab1Asynthesis.txt`
  - Make sure there are no lingering `printf` statements in your code!

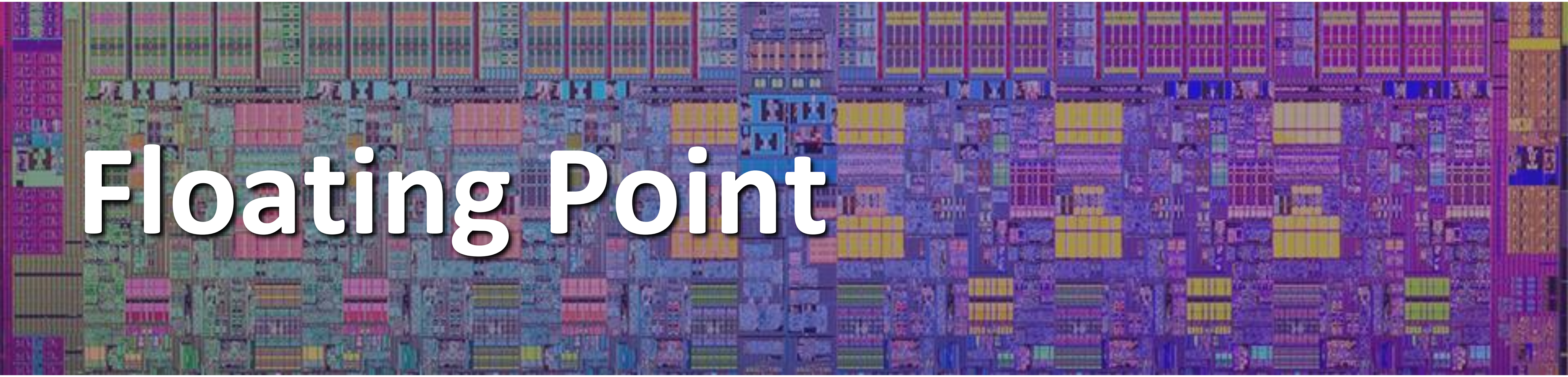
- Lab 1b due Monday (1/22)
  - Submit `aisle_manager.c`, `store_client.c`, and `lab1Bsynthesis.txt`

Lab 1b Aside: C Macros

- C macros basics:
  - Basic syntax is of the form: `#define NAME expression`
  - Allows you to use "`NAME`" instead of "`expression`" in code
    - Does naïve copy and replace *before* compilation – everywhere the characters "NAME" appear in the code, the characters "expression" will now appear instead
    - NOT the same as a Java constant
  - Useful to help with readability/factoring in code

- You'll use C macros in Lab 1b for defining bit masks
  - See Lab 1b starter code and Lesson 4 (card operations) for examples

Floating Point



Lesson Summary (1/2)

- Floating point approximates real numbers (large, small, & special):

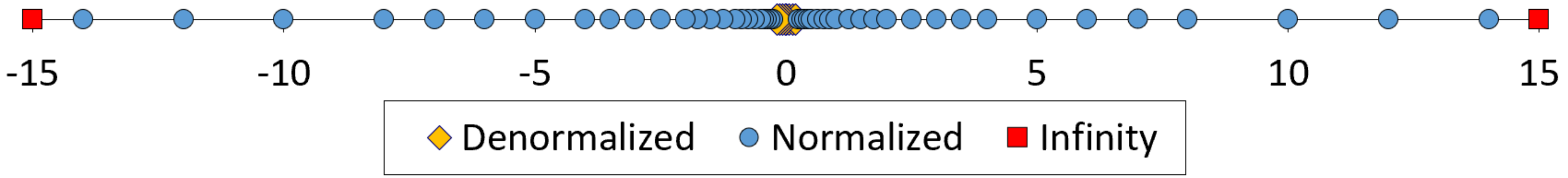
31	30 … 23	22 … 0
S	E (8)	M (23)

  - Normalized case: $\pm 1 \times \text{Mantissa} \times 2^{\text{Exponent}} = (-1)^{S} \times 1.M \times 2^{(E-\text{bias})}$
  - Mantissa approximates fractional portion
    - Size of mantissa field determines our representable ***precision***
    - Exceeding mantissa length causes ***rounding***
  - Exponent in biased notation (bias = $2^{w-1} - 1$)
    - Size of exponent field determines our representable ***range***
    - Outside of representable exponents is ***overflow*** and ***underflow***
  - `double` (64 bits: [S (1)|E (11)|M (52)]) available if more precision needed

E	M	Meaning
0b0…0	anything	± denorm num (including 0)
anything else	anything	± norm num
0b1…1	0	± ∞
0b1…1	non-zero	NaN

Lesson Summary (2/2)

- Limitations of FP affect programmers all the time (!)
  - Overflow, underflow, rounding
    - Rounding is a HUGE issue due to limited mantissa bits and gaps that are scaled by the value of the exponent

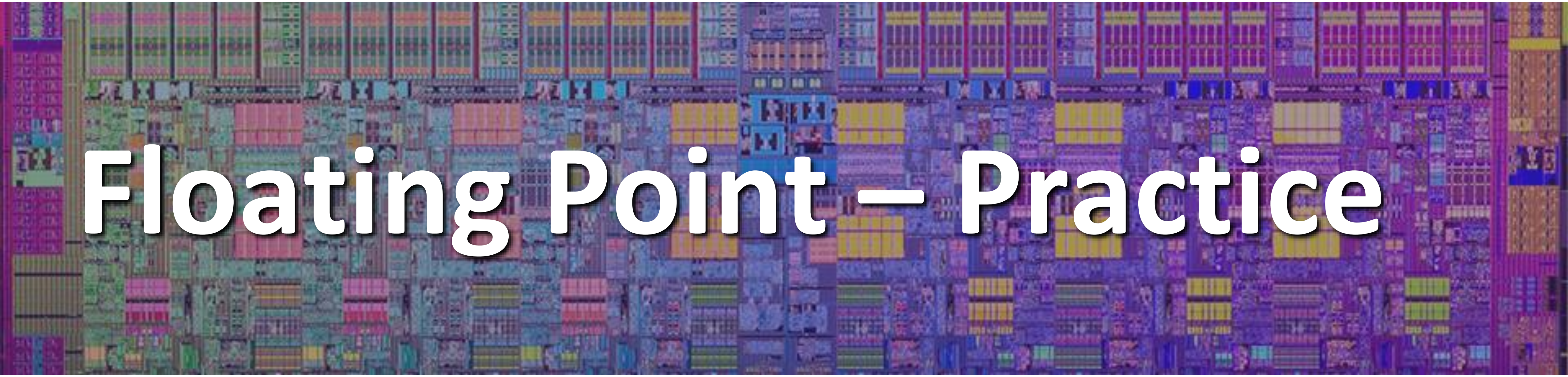


  - Floating point arithmetic is NOT associative or distributive
    - ∞ and NaN are valid operands, but can produce unintuitive results
  - Do NOT use equality (==) with floating point numbers
  - Converting between integral and floating point data types *does* change the bits
    - *e.g.*, `int i = 2; // stored as 0x00000002`, `float f = i; // stored as 0x40000000`

Lesson Q&A

- Learning Objectives:
  - Describe how the bits in floating point are organized and how they represent real numbers (and special cases).
  - Describe the distribution of representable values in floating point.
  - Explain the limitations of floating point and write C code that accounts for them.

- What lingering questions do you have from the lesson?
  - Chat with your neighbors about the lesson for a few minutes to come up with questions



Floating Point – Practice

$2^{-1} = 0.5$
$2^{-2} = 0.25$
$2^{-3} = 0.125$
$2^{-4} = 0.0625$

Polling Questions (1/2)

- What is the value encoded by the following floating point number?

0b $\underbrace{0}_{S}$ | $\underbrace{1000\ 0000}_{E}$ | $\underbrace{110\ 0000\ 0000\ 0000\ 0000\ 0000}_{M}$

  - bias = $2^{w-1}-1$ $= 2^7 - 1 = 127$
  - exponent = E – bias $= 2^7 - 127 = 128 - 127 = 1$
  - mantissa = 1.M $= 1.110\ldots0_2$

$(-1)^0 \times 1.11_2 \times 2^1 = 11.1_2 = \boxed{+3.5}$

- Convert the decimal number **-7.375 = -1.11011 x 2²** into floating point representation.

S = 1 , E = 2+127 = 129 = 0b 1000 0001 , M = 0b 11011 0...0

0b 1 1000 0001 1101 1000...0 = $\boxed{0\text{x C0EC 0000}}$

Polling Questions (2/2)

- What is the value of the following floats?
  - `0x00000000` ⇒ S=0, E=0, M=0 ⇒ $\boxed{+0}$
  - `0xFF800000` ⇒ S=1, E= all 1's, M=0 ⇒ $\boxed{-\infty}$

0b 1 | 1111 1111 | 1000 0...0
S E M

- For the following code, what is the smallest value of n that will encounter a limit of representation?

```
float f = 1.0;  // 2^0
for (int i = 0; i < n; ++i)
    f *= 1024;  // 1024 = 2^10
printf("f = %f\n", f);
```

13

E_{max} = 0xFE, Exp_{max} = 254 − 127 = 127

n	f
1	2^{10}
2	2^{20}
3	2^{30}
4	2^{40}
⋮	⋮

always M = 0

for $\boxed{n=13}$, we hit 2^{130}, which causes $\boxed{\text{overflow}}$

Homework Setup

- Let **`float`** `f = 1073741824;` *`// 2^30;`*
- What's the smallest power of 2 for g such that `f + g != f`?

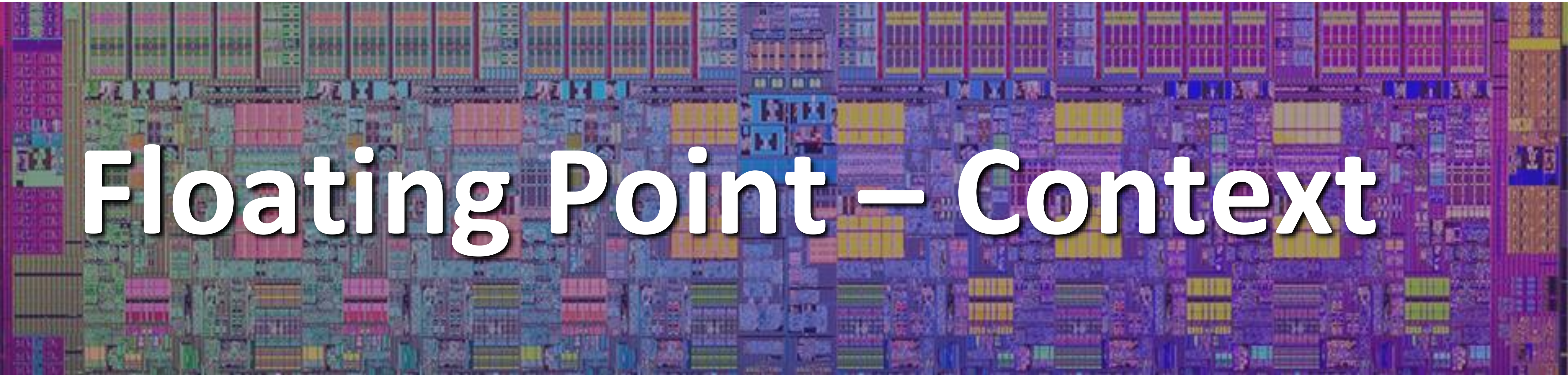
$$2^{30} = 1.0000\,0000\,0000\,0000\,0000\,000 \times 2^{30}$$

2^0 2^{-23}

$$2^{-23} \times 2^{30} = \boxed{2^7}$$

← smaller powers of 2 will get rounded off

Floating Point – Context



Floating Point Issues in Real Life

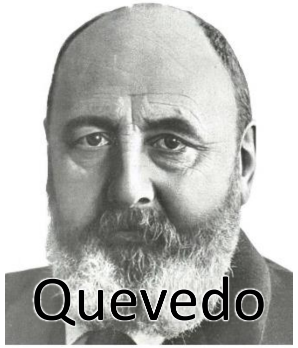
- **1991:** Patriot missile targeting error
  - Time in system stored in integer (tenths of a second since boot)
  - Converted to seconds by multiplying by 0.1 = $0.0\overline{0011}_2$ leading to erroneous time (error grows the longer system has been on)

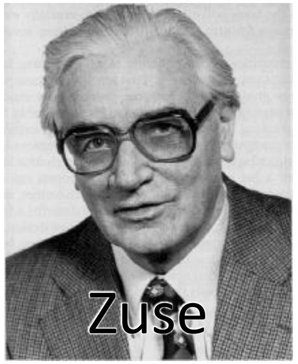
- **1996:** V88 Ariane 501 rocket exploded 37 seconds after launch
  - Reused code from Ariane 4 inertial reference platform
  - Overflow when converting a 64-bit floating point number to a 16-bit integer (not protected by extra lines of code)

- **Other related bugs:**
  - 1982: Vancouver Stock Exchange 50% error in less than 2 years due to truncation
  - 1994: Intel Pentium FDIV (floating point division) hardware bug costs company $475 million in recall

More on Floating Point History

- Early days
  - First design with floating-point arithmetic in 1914 by Leonardo Torres y Quevedo
  - Implementations started in 1940 by Konrad Zuse, but with differing field lengths (usually not summing to 32 bits) and different subsets of the special cases
- IEEE 754 standard created in 1985
  - Primary architect was William Kahan, who won a Turing Award for this work
  - Standardized bit encoding, well-defined behavior for *all* arithmetic operations





Floating Point in the “Wild”

- 3 formats from IEEE 754 standard widely used in computer hardware and languages
  - In C, called `float`, `double`, `long double`
- Common applications:
  - 3D graphics: textures, rendering, rotation, translation
  - “Big Data”: scientific computing at scale, machine learning
- Non-standard formats in domain-specific areas:
  - **Bfloat16:** training ML models; range more valuable than precision
  - **TensorFloat-32:** Nvidia-specific hardware for Tensor Core GPUs

Type	S bits	E bits	M bits	Total bits
Half-precision	1	5	10	16
Bfloat16	1	8	7	16
TensorFloat-32	1	8	10	19
Single-precision	1	8	23	32

Discussion Question

- Discuss the following question(s) in groups of 3-4 students
  - I will call on a few groups afterwards so please be prepared to share out
  - Be respectful of others' opinions and experiences

- How do you feel about floating point?
  - Do you feel like the limitations are acceptable?
  - Does this affect the way you'll think about non-integer arithmetic in the future?
  - Are there any changes or different encoding schemes that you think would be an improvement?

Group Work Time

- During this time, you are encouraged to work on the following:
  1) If desired, continue your discussion
  2) Work on the homework problems
  3) Work on the lab (if applicable)

- Resources:
  - You can revisit the lesson material
  - Work together in groups and help each other out
  - Course staff will circle around to provide support