Floating Point I

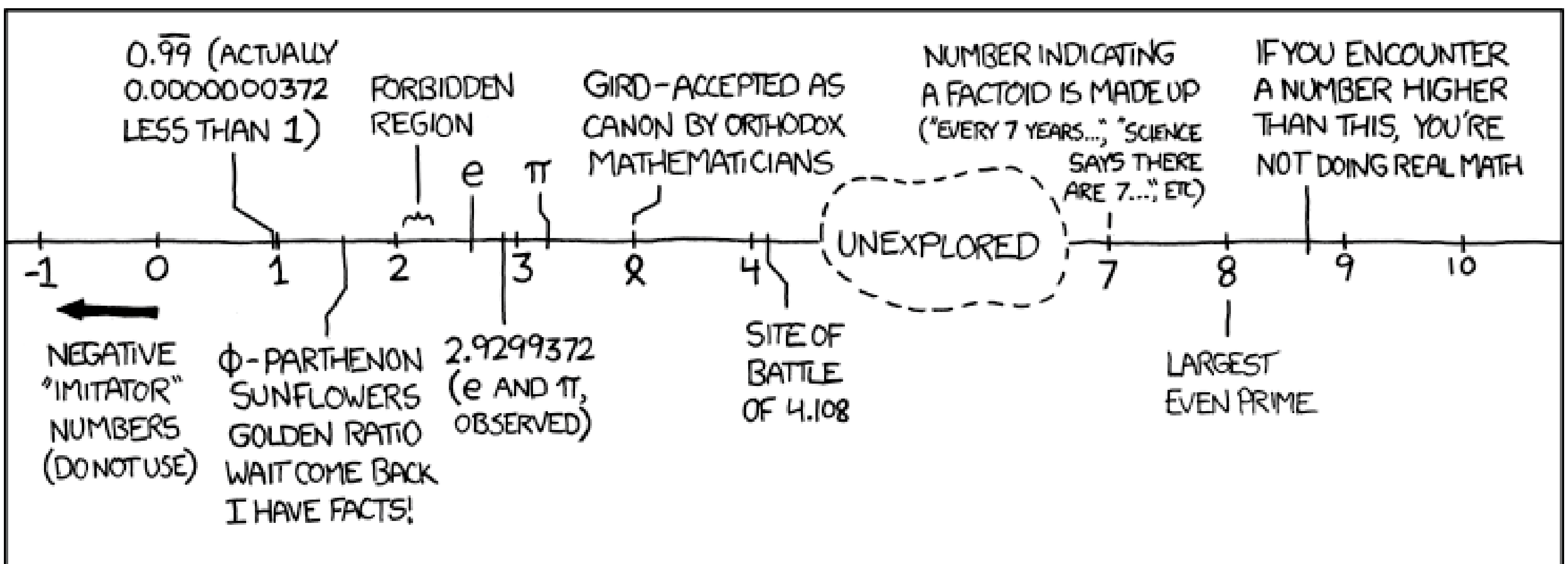
CSE 351 Winter 2021

Instructor:

Mark Wyse

Teaching Assistants:

Kyrie Dowling, Catherine Guevara, Ian Hsiao, Jim Limprasert, Armin Magness, Allie Pfleger, Cosmo Wang, Ronald Widjaja



http://xkcd.com/899/

Administrivia

- hw5 due Wednesday, hw6 due Friday

Monday – No Class!

- Lab 1a due tonight at 11:59 pm
  - Submit `pointer.c` and `lab1Areflect.txt`
  - Make sure you submit *something* to Gradescope before the deadline and that the file names are correct
  - Can use late day tokens to submit up until Mon 11:59 pm
- Section worksheet due tonight at 11:59 pm!
- Lab 1b due next Friday (1/22)
  - Submit `aisle_manager.c`, `store_client.c`, and `lab1Breflect.txt`

Reading Review

- Terminology:
  - normalized scientific binary notation
  - trailing zeros
  - sign, mantissa, exponent ↔ bit fields S, M, and E
  - `float`, `double`
  - biased notation (exponent), implicit leading one (mantissa)
  - rounding errors

- Floating Point Simulator
  - https://www.h-schmidt.net/FloatConverter/IEEE754.html
- Questions from the Reading?

Review Questions

$2^{-1} = .5$
$2^{-2} = .25$
$2^{-3} = .125$

- Convert 11.375_{10} to normalized binary scientific notation

$2^3 + 2^1 + 2^0 + 2^{-2} + 2^{-3} = 1011.011_2$

1.011011×2^3

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

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

  - bias = $2^{w-1}-1$ = 127 (w = 8)
  - exponent = E – bias   128 − 127 = 1
  - mantissa = 1.M

$(-1)^0 \times 1.110\ldots \times 2^1 \rightarrow$ equation

11.10

$2^1 + 2^0 + 2^{-1}$

$= 3.5$

Number Representation Revisited

- What can we represent in one word?
  - Signed and Unsigned Integers
  - Characters (ASCII)
  - Addresses

- How do we encode the following:
  - Real numbers (*e.g.*, 3.14159)
  - Very large numbers (*e.g.*, 6.02×10^{23})
  - Very small numbers (*e.g.*, 6.626×10^{-34})
  - Special numbers (*e.g.*, ∞, NaN)

Floating Point

Floating Point Topics

- **Fractional binary numbers**
- **IEEE floating-point standard**
- Floating-point operations and rounding
- Floating-point in C

- There are many more details that we won't cover
  - It's a 58-page standard…

Representation of Fractions

- "Binary Point," like decimal point, signifies boundary between integer and fractional parts:

Example 6-bit representation: xx.yyyy

2^1 2^0 2^{-1} 2^{-2} 2^{-3} 2^{-4}

- Example: $10.1010_2 = 1\times2^1 + 1\times2^{-1} + 1\times2^{-3} = 2.625_{10}$

Representation of Fractions

- "Binary Point," like decimal point, signifies boundary between integer and fractional parts:

Example 6-bit representation: **xx.yyyy**

2^{1} 2^{0} 2^{-1} 2^{-2} 2^{-3} 2^{-4}

- In this 6-bit representation:
  - What is the encoding and value of the smallest (most negative) number?

    $\varnothing \rightarrow 00.0000$

  - What is the encoding and value of the largest (most positive) number?

    $11.1111 = 4 - 2^{-4} = 3.9375$

    $2^{1} + 2^{0} + \underline{2^{-1} + 2^{-2} + 2^{-3} + 2^{-4}}$

  - What is the smallest number greater than 2 that we can represent?

    $10.0000 \rightarrow 10.0001 = 2.0625$

    2^{4}

Scientific Notation (Binary)

mantissa

exponent

normalize →

$$\underbrace{1.01_2}_{\text{mantissa}} \times 2^{-1}$$

binary point

radix (base)

- *Normalized form*: exactly one digit (non-zero) to left of binary point

- Computer arithmetic that supports this called floating point due to the "floating" of the binary point
  - Declare such variable in C as `float` (or `double`)

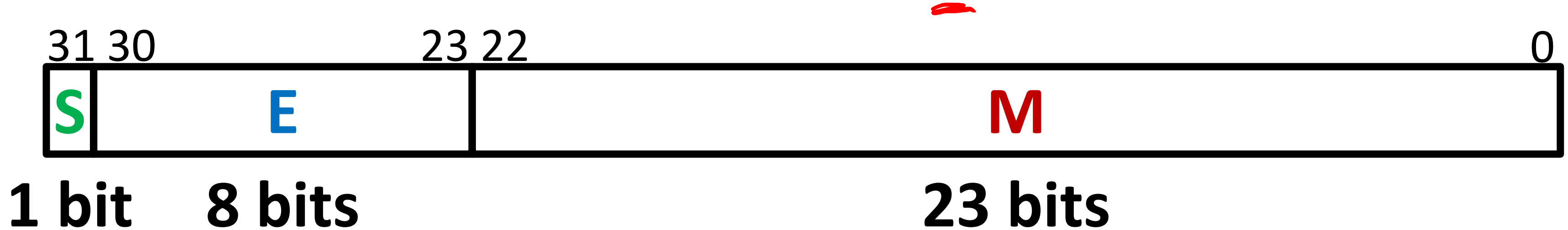
IEEE Floating Point

- IEEE 754 (established in 1985)
  - Standard to make numerically-sensitive programs portable
  - Specifies two things: *representation scheme* and result of *floating point operations*
  - Supported by all major CPUs
- Driven by numerical concerns
  - **Scientists**/numerical analysts want them to be as **real** as possible
  - **Engineers** want them to be **easy to implement** and **fast**
  - Scientists mostly won out:
    - Nice standards for rounding, overflow, underflow, but...
    - Hard to make fast in hardware
    - **Float operations can be an order of magnitude slower than integer ops**

FLOP → FLOPS

Floating Point Encoding

- Use normalized, base 2 scientific notation:
  - Value: $\pm 1 \times \text{Mantissa} \times 2^{\text{Exponent}}$
  - (S,E,M) Bit Fields: $(-1)^{S} \times 1.M \times 2^{(E-\text{bias})}$
- Representation Scheme:
  - Sign bit (0 is positive, 1 is negative) → S
  - Mantissa (a.k.a. significand) is the fractional part of the number in normalized form and encoded in bit vector **M**
  - Exponent weights the value by a (possibly negative) power of 2 and encoded in the bit vector **E**



The Exponent Field

float: w=8

- Use biased notation
  - Read exponent as unsigned, but with *bias* of $\underline{2^{w-1}-1}$ = 127
  - Representable exponents roughly ½ positive and ½ negative
  - Exp = E – bias ↔ E = Exp + bias
    - Exponent 0 (Exp = 0) is represented as E = 0b 0111 1111

=127

Exp = 127 − 127

Exp 1 = 128 − 127 → E = 0b 1000 0000

−63 = 64 − 127 → E = 0b 0100 0000

- Why biased?
  - Makes floating point arithmetic easier
  - Makes somewhat compatible with two's complement hardware

The Mantissa (Fraction) Field

31	30 ... 23	22 ... 0
S	E	M
1 bit	8 bits	23 bits

$$(-1)^{S} \times (1\,.\,M) \times 2^{(E-\text{bias})}$$

- Note the implicit leading 1 in front of the M bit vector
  - Example: 0b 0011 1111 1100 0000 0000 0000 0000 0000
    is read as $1.1_2 = 1.5_{10}$, *not* $0.1_2 = 0.5_{10}$
  - Gives us an extra bit of *precision*
- Mantissa "limits"
  - Low values near M = 0b0…0 are close to 2^{Exp}
  - High values near M = 0b1…1 are close to 2^{Exp+1}

$2^{Exp} * (2 - 2^{-23})$

$= 2^{Exp+1} - \underbrace{2^{Exp-23}}$

$2^{Exp} * 1.00\ldots0$

$2^{Exp} * 1.\underline{11\ldots1}$

Normalized Floating Point Conversions

- FP → Decimal
  1. Append the bits of M to implicit leading 1 to form the mantissa.
  2. Multiply the mantissa by $2^{E-\text{bias}}$.
  3. Multiply the sign $(-1)^S$.
  4. Multiply out the exponent by shifting the binary point.
  5. Convert from binary to decimal. ...xxxx.yyyy...
- Decimal → FP
  1. Convert decimal to binary. ...xxxx.yyyy...
  2. Convert binary to normalized scientific notation. $1.xxx \times 2^{Exp}$
  3. Encode sign as S (0/1).
  4. Add the bias to exponent and encode E as unsigned.
  5. The first bits after the leading 1 that fit are encoded into M.

Practice Question

- Convert the decimal number **-7.375** into floating point representation $= -(4+2+1+0.25+0.125)$

$= -111.011_2$

$(-1)^1_{10} \times (1.11011_2 \times 2^2)$

$S=1 \quad M=110110\ldots0 \quad E=2+127=129$

$= 0b\ 1000\ 0001$

$1\ 1000\ 0001\ 110\ 1100 \ldots 0$

$0xC0EC0000$

Challenge Question

- Find the sum of the following binary numbers in normalized scientific binary notation:

$$1.01_2 \times 2^0 + 1.11_2 \times 2^2$$

$0.0101 \times 2^2 + 1.11 \times 2^2$

$$\begin{array}{r} 0.0101 \\ +\ 1.1100 \\ \hline 10.0001_2 \end{array}$$

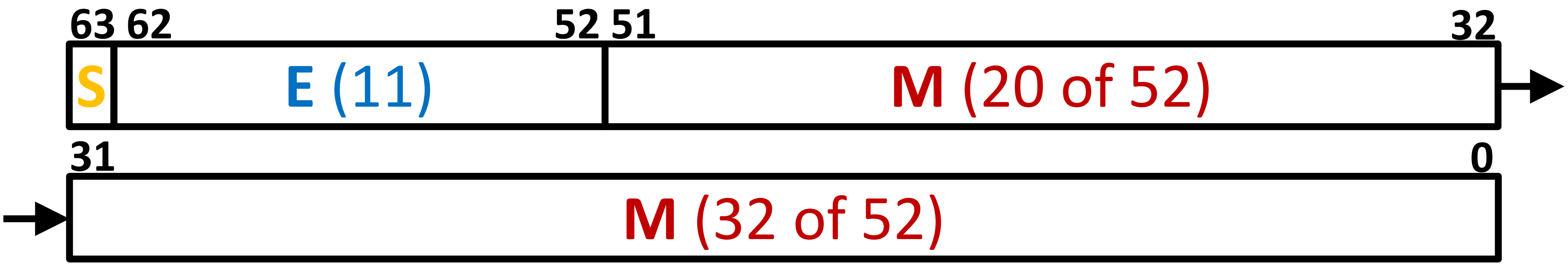
$10.0001_2 \times 2^2 = \boxed{1.00001 \times 2^3}$

Precision and Accuracy

- Precision is a count of the number of bits in a computer word used to represent a value
  - Capacity for accuracy
- Accuracy is a measure of the difference between the *actual value of a number* and its computer representation
  - *High precision permits high accuracy but doesn't guarantee it. It is possible to have high precision but low accuracy.*
  - **Example:** `float pi = 3.14;` 3.14........
    - `pi` will be represented using all 24 bits of the mantissa (highly precise), but is only an approximation (not accurate)

Need Greater Precision?

- Double Precision (vs. Single Precision) in 64 bits



  - C variable declared as `double`
  - Exponent bias is now $2^{10}–1 = 1023$ $2^{w-1} - 1 \rightarrow w = 11$
  - **Advantages:** greater precision (larger mantissa), greater range (larger exponent)
  - **Disadvantages:** more bits used, slower to manipulate

Current Limitations

- Largest magnitude we can represent? E & M all 1
- Smallest magnitude we can represent? E & M all Ø
  exponent of −127
  - Limited *range* due to width of E field

- What happens if we try to represent $2^0 + 2^{-30}$?
  > M bits
  - Rounding due to limited *precision*: stores 2^0

- There is a need for *special cases*
  - How do we represent the value zero?
  - What about ∞ and NaN?

Summary

- Floating point approximates real numbers:

31	30 … 23	22 … 0
S	E (8)	M (23)

  - Handles large numbers, small numbers, special numbers
  - 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*
  - Mantissa approximates fractional portion of binary point
    - Size of mantissa field determines our representable *precision*
    - Implicit leading 1 (normalized) except in special cases
    - Exceeding length causes *rounding*