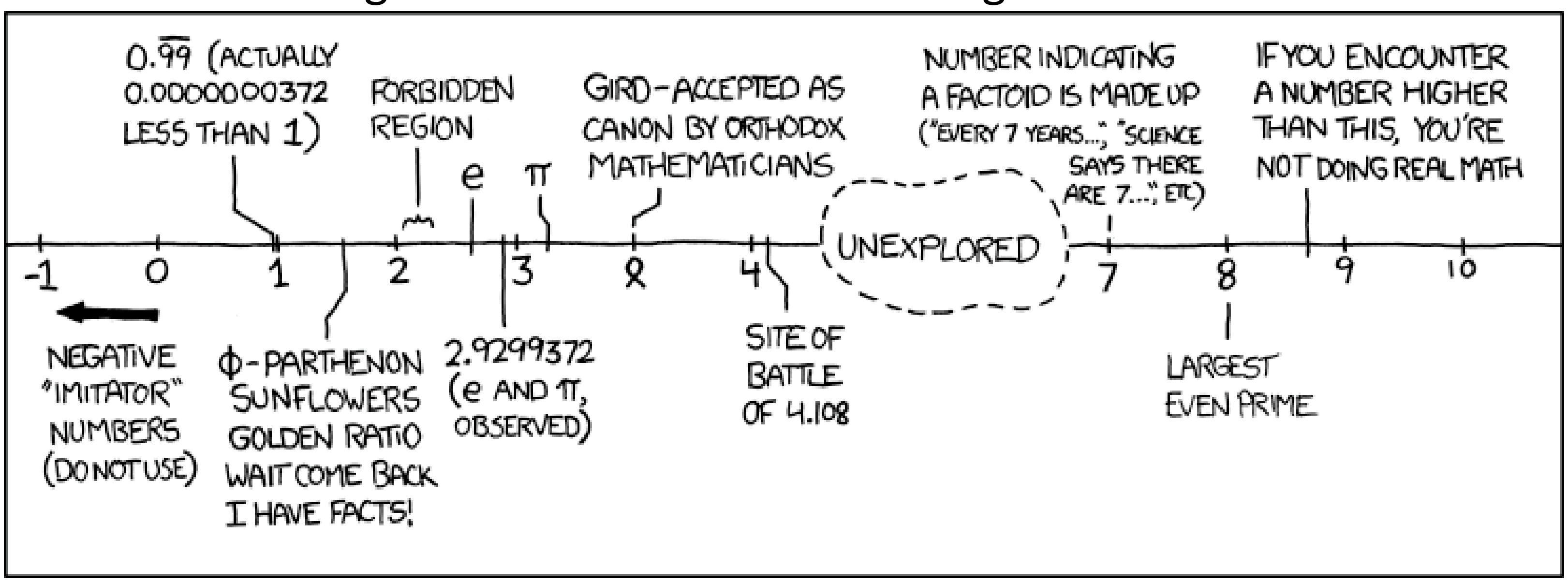
Floating Point I

CSE 351 Spring 2022

Instructor:	**Teaching Assistants:**		
Ruth Anderson	Melissa Birchfield	Jacob Christy	Alena Dickmann
	Kyrie Dowling	Ellis Haker	Maggie Jiang
	Diya Joy	Anirudh Kumar	Jim Limprasert
	Armin Magness	Hamsa Shankar	Dara Stotland
	Jeffery Tian	Assaf Vayner	Tom Wu
	Angela Xu	Effie Zheng	



http://xkcd.com/899/

Relevant Course Information

- hw4 due Friday (4/08) @ 11:59 pm
- hw5 due Monday (4/11) @ 11:59 pm
- Lab 1a due Monday (4/11) @ 11:59 pm
  - Submit `pointer.c` and `lab1Asynthesis.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 Wed 11:59 pm
- Lab 1b, due 4/18
  - 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 Lecture 4 slides (card operations) for examples

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

Review Questions

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

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

  0b 0 | 1000 0000 | 110 0000 0000 0000 0000 0000

  - bias = $2^{w-1}-1$
  - exponent = E – bias
  - mantissa = 1.M

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:

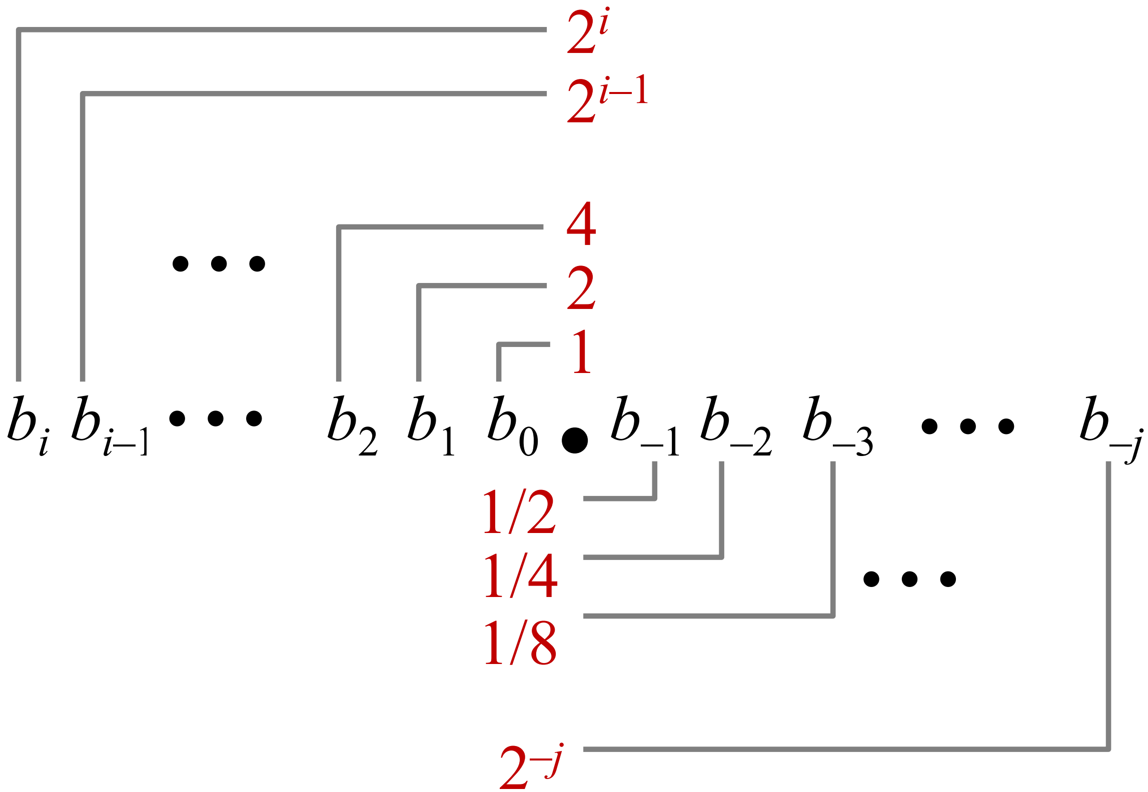
Example 6-bit representation:

`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?
  - What is the encoding and value of the largest (most positive) number?
  - What is the smallest number greater than 2 that we can represent?

Fractional Binary Numbers



- **Representation**
  - Bits to right of "binary point" represent fractional powers of 2
  - Represents rational number: $$\sum_{k=-j}^{i} b_k \cdot 2^k$$

Fractional Binary Numbers

- Value Representation
  - 5 and 3/4 `101.11`$_2$
  - 2 and 7/8 `10.111`$_2$
  - 47/64 `0.101111`$_2$
- Observations
  - Shift left = multiply by power of 2
  - Shift right = divide by power of 2
  - Numbers of the form `0.111111`$\ldots_2$ are just below 1.0
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - Use notation $1.0 - \varepsilon$

Limits of Representation

- Limitations:
  - Even given an arbitrary number of bits, can only **exactly** represent numbers of the form $x * 2^y$ (y can be negative)
  - Other rational numbers have repeating bit representations

Value:	Binary Representation:
• 1/3 = $0.333333\ldots_{10}$ =	$0.01010101[01]\ldots_2$
• 1/5 =	$0.001100110011[0011\ ]\ldots_2$
• 1/10 =	$0.0001100110011[0011\ ]\ldots_2$

Fixed Point Representation

- Implied binary point. Two example schemes:
  - #1: the binary point is between bits 2 and 3
    - $b_7\ b_6\ b_5\ b_4\ b_3$ [.] $b_2\ b_1\ b_0$
  - #2: the binary point is between bits 4 and 5
    - $b_7\ b_6\ b_5$ [.] $b_4\ b_3\ b_2\ b_1\ b_0$
- Which scheme is best?

Floating Point Representation

- Analogous to scientific notation
  - In Decimal:
    - Not 12000000, but 1.2×10^7 In C: 1.2e7
    - Not 0.0000012, but 1.2×10^{-6} In C: 1.2e-6
  - In Binary:
    - Not 11000.000, but 1.1×2^4
    - Not 0.000101, but 1.01×2^{-4}
- We have to divvy up the bits we have (e.g., 32) among:
  - the sign (1 bit)
  - the mantissa (significand)
  - the exponent

Binary Scientific Notation (Review)

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

exponent

binary point

radix (base)

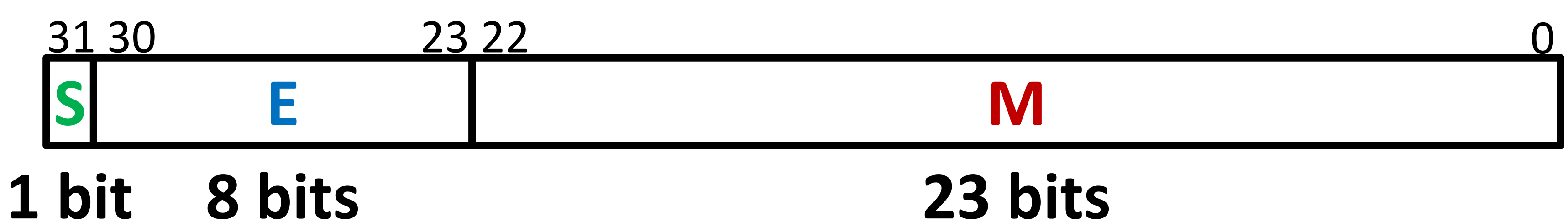
- *Normalized form*: exactly one digit (non-zero) to left of binary point
- Computer arithmetic that supports this called floating point due to the "floating" of the binary point
  - Declare such variable in C as `float` (or `double`)

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**

Floating Point Encoding (Review)

- Use normalized, base 2 scientific notation:
  - Value: $\pm 1 \times \text{Mantissa} \times 2^{\text{Exponent}}$
  - Bit Fields: $(-1)^{S} \times 1.M \times 2^{(E-\text{bias})}$
- Representation Scheme:
  - Sign bit (0 is positive, 1 is negative)
  - 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 (Review)

- Use biased notation
  - Read exponent as unsigned, but with *bias* of $2^{w-1}-1$ = 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

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

The Mantissa (Fraction) Field (Review)

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 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}

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 $2^{\mathrm{E}-\text{bias}}$.
  3. Multiply the sign $(-1)^{\mathrm{S}}$.
  4. Multiply out the exponent by shifting the binary point.
  5. Convert from binary to decimal.

- Decimal → FP
  1. Convert decimal to binary.
  2. Convert binary to normalized scientific notation.
  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

Challenge Question

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

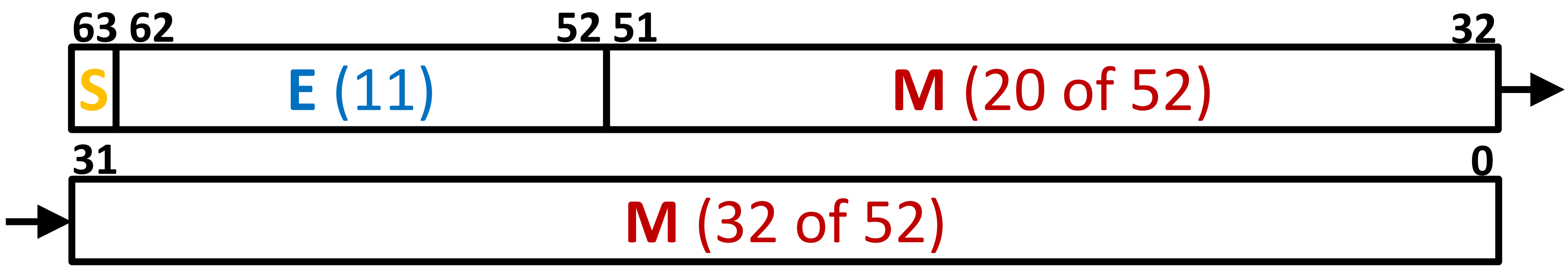
$$1.01_2 \times 2^0 + 1.11_2 \times 2^2$$

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;`
    - `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 $2^{10}-1 = 1023$
  - **Advantages:** greater precision (larger mantissa), greater range (larger exponent)
  - **Disadvantages:** more bits used, slower to manipulate

Current Limitations

- Largest magnitude we can represent?
- Smallest magnitude we can represent?
  - Limited *range* due to width of E field

- What happens if we try to represent $2^0 + 2^{-30}$?
  - 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*