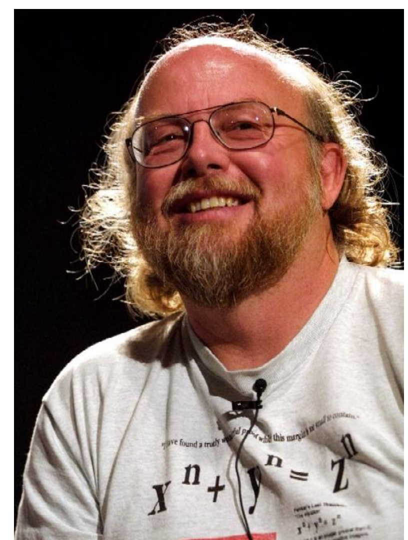
Topics for Module 1

- Part 1: Course Logistics
- **Part 2: Review number representation**
  - Fixed point numbers
  - → Floating point numbers
    - Textbook Section 3.5 (up to “Floating Point Addition) and the Elaboration part at the end of the section
    - Worksheet on floating point number representation

Quote of the day

"95% of the folks out there are completely clueless about floating-point."

James Gosling
Sun Fellow
Java Inventor
1998-02-28



Review of Numbers

- **Computers are made to deal with numbers**
- **What can we represent in N bits?**
  - Unsigned integers:

    0 to $2^N - 1$
  - Signed Integers (Two's Complement)

    $-2^{(N-1)}$ to $2^{(N-1)} - 1$

Other Numbers

- **What about other numbers?**
  - Very large numbers? (seconds/century)
    $3,155,760,000_{10}$ ($3.15576_{10} \times 10^9$)
  - Very small numbers? (atomic diameter)
    0.00000001_{10} ($1.0_{10} \times 10^{-8}$)
  - Rationals (repeating pattern)
    2/3 (0.666666666. . .)
  - Irrationals
    $2^{1/2}$ (1.414213562373. . .)
  - Transcendentals
    e (2.718...), π (3.141...)
- **All represented in scientific notation**

Scientific Notation (in Decimal)

exponent

$$6.02_{10} \times 10^{23}$$

decimal point radix (base)

- **Scientific Notation:**
  - exactly one digit to the left of decimal point
- **Normalized scientific notation : no leadings 0s**
- **Alternatives to representing 1/1,000,000,000**
  - Normalized: 1.0×10^{-9}
  - Not normalized: $0.1 \times 10^{-8}, 10.0 \times 10^{-10}$

Scientific Notation (in Binary)

$$1.0_{two} \times 2^{-1}$$

exponent

"binary point"

radix (base)

- **Floating point**
  - Computer arithmetic that represents numbers where binary point is not fixed
  - Declare such variable in C as `float`

“Father” of the Floating point standard

IEEE Standard 754 for Binary Floating-Point Arithmetic.

Prof. Kahan

1989
ACM Turing Award Winner!

`www.cs.berkeley.edu/~wkahan/ieee754status/`

Single-Precision Floating Point

- **Normal format: $\pm 1.\text{xxxxxxxxxx}_{two} * 2^{yyyy}{}_{two}$**

31	30 … 23	22 … 0
S	**Exponent**	**Fraction**
1 bit	8 bits	23 bits

- **Sign and Magnitude Representation (32 bits)**
  - S: Sign (1 for negative, 0 for positive)
  - Exponent: yyyy
  - Fraction: xxxx…xxxx (Significand: 1+0.xxxx…xxxx)

    => 24 bits
- **Real value = $(-1)^S x (1+F) x 2^{(E-127)}$**

Why this Order of Fields

- **Sort records with FP numbers using integer compares**
- **Break FP number into 3 parts**
  - compare signs: negative < positive
  - compare exponents: big exponent => bigger number
  - compare significands for same exponents

Biased Exponent

- **Bias: the number subtracted from the normal and unsigned representation to get the real value**
- **IEEE 754 bias = 2^{8-1}-1=127 for single precision**
  - Stored exponent = real exponent + bias
- **0 < E < 255**
- **Magnitude of numbers**
  - 2^{-126} to 2^{127}
- **Why biased?**
  - negative exponents appear big in 2’s complements

Negative Exponent

- 2's compliment? 1.0 x 2^{-1} vs. 1.0 x2^{+1} (1/2 vs. 2)

1/2	0	1111 1111	000 0000 0000 0000 0000 0000
2	0	0000 0001	000 0000 0000 0000 0000 0000

  - 1/2 seems bigger than 2
- Desirable notation for exponent
  - 0000 0000 : smallest
  - 1111 1111 : largest
- Biased exponent:
  - -1: 126; 1: 128

1/2	0	0111 1110	000 0000 0000 0000 0000 0000
2	0	1000 0000	000 0000 0000 0000 0000 0000

Single-Precision Range

- **Exponents 00000000 and 11111111 reserved**
- **Smallest value**
  - Exponent: 00000001
    $\Rightarrow$ actual exponent = 1 – 127 = –126
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$
- **Largest value**
  - exponent: 11111110
    $\Rightarrow$ actual exponent = 254 – 127 = +127
  - Fraction: 111…11 $\Rightarrow$ significand $\approx$ 2.0 (see next slide)
  - $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

Review: Sum of Geometric Series

- **A geometric sequence/series: {a, ar, ar^2, ar^3, ... }**

where:

  - a is the first term, and
  - r is the factor between the terms (called the "common ratio")

- **Sum of a geometric series:** a + ar + ar^2 + ... + ar$^{(n-1)}$

$$\sum_{k=0}^{n-1}(ar^k) = a\left(\frac{1-r^n}{1-r}\right)$$

- **$0.1111...1_{two} = 2^{-1}+2^{-2}+..+2^{-n}$, where: $a=2^{-1}$, $r = 2^{-1}$**

$=(2^{-1})((1-(2^{-1})^n)/(1-2^{-1})) = 1 - 2^{-n}$

Overflow and Underflow

- **Overflow**
  - result too large (> 3.4×10^{38})
  - Exponent larger than represented in 8-bit Exponent field
- **Underflow**
  - result too small? (>0, < 1.2×10^{-38})
  - Negative exponent larger than represented in 8-bit Exponent field
- **How to reduce chances of overflow or underflow?**

Double Precision Fl. Pt. Representation

- **Multiple Word Size (64 bits)**

31	30 20	19 0
S	Exponent	Fraction
1 bit	11 bits	20 bits

Fraction (cont'd)
32 bits

- **Double Precision (vs. Single Precision)**
  - C variable declared as `double`
  - Bias=2^{11-1}-1=1023
  - [2.0 x 10^{-308}, 2.0 x 10^{308}]
  - greater accuracy due to larger significand

Double-Precision Range

- **Exponents 0000…00 and 1111…11 reserved**
- **Smallest value**
  - Exponent: 00000000001
    $\Rightarrow$ actual exponent = 1 – 1023 = –1022
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$
- **Largest value**
  - Exponent: 11111111110
    $\Rightarrow$ actual exponent = 2046 – 1023 = +1023
  - Fraction: 111…11 $\Rightarrow$ significand $\approx$ 2.0
  - $\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{+308}$

Floating-Point Precision

- **Relative precision**
  - all fraction bits are significant
  - Single: approx 2^{23}
    - Equivalent to $23 \times \log_{10}2 \approx 23 \times 0.3 \approx 7$ decimal digits of precision
  - Double: approx 2^{52}
    - Equivalent to $52 \times \log_{10}2 \approx 52 \times 0.3 \approx 16$ decimal digits of precision

Single Precision vs. Double Precision

	Single Precision	**Double Precision**
C programming	`float, real`	`double`
Size	32 bits, 1 word	64 bits, 2 words
Exponent	8 bits	11 bits
Fraction	23 bits	52 bits
Bias	127	1023
Range	10^{-38} to 10^{38}	10^{-308} to 10^{308}

- **Double Precision has greater range and accuracy**
  - Fit all 32-bit in fraction

Understanding the Significand

- **Method 1 (Fractions):**
  - In decimal: $0.340_{10} \Rightarrow 340_{10}/1000_{10}$
    $\Rightarrow 34_{10}/100_{10}$
  - In binary: $0.110_2 \Rightarrow 110_2/1000_2 = 6_{10}/8_{10}$
    $\Rightarrow 11_2/100_2 = 3_{10}/4_{10}$
- **Method 2 (Place Values):**
  - Convert from scientific notation
  - In decimal: $1.6732 = (1 \times 10^0) + (6 \times 10^{-1}) + (7 \times 10^{-2}) + (3 \times 10^{-3}) + (2 \times 10^{-4})$
  - In binary: $1.1001 = (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4})$
  - Interpretation of value in each position extends beyond the decimal/binary point

Special Numbers

- What have we defined so far for single precision floating point numbers?

Exponent	Fraction	Object
0	0	???
0	Nonzero	???
1-254	Anything	+/- fl. Pt #
255	0	???
255	Nonzero	???

Representation for $\pm \infty$

- **In FP, divide by 0 should produce $\pm \infty$, not overflow.**
- **Why?**
  - OK to do further computations with ∞
  - e.g., $X/0 > Y$ may be a valid comparison
- **Representation**
  - Exponent: 255 (for S.P), 2047 (for D.P)
  - Fraction: 0

Representation for 0

- **Represent 0?**
  - Exponent: 0
  - Fraction: 0
  - What about sign?
  - `+0: 0 00000000 00000000000000000000000`
  - `-0: 1 00000000 00000000000000000000000`
- **Why two zeroes?**
  - Helps in some limit comparisons

Representation for Not a Number

- **What is `sqrt(-4.0)` or `0/0`?**
  - If ∞ not an error, these shouldn't be either.
  - Called Not a Number (NaN)
  - Exponent: 255, Fraction: nonzero

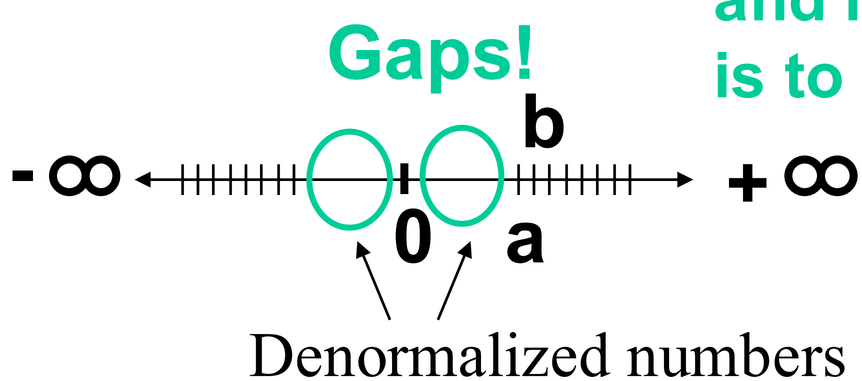
- **Why is this useful?**
  - help with debugging
  - op(NaN, X) = NaN

Representation for Denorms (1/2)

- **Problem: a gap among representable FP numbers around 0**
  - Smallest representable positive num:

    $a = 1.0\ldots_2 * 2^{-126} = 2^{-126}$
  - $a - 0 = 2^{-126}$

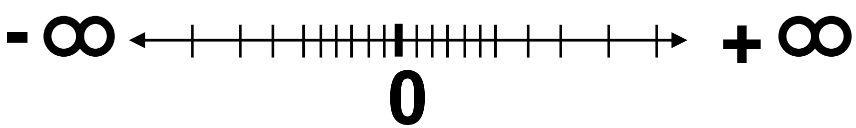
Normalization and implicit 1 is to blame!



Representation for Denorms (2/2)

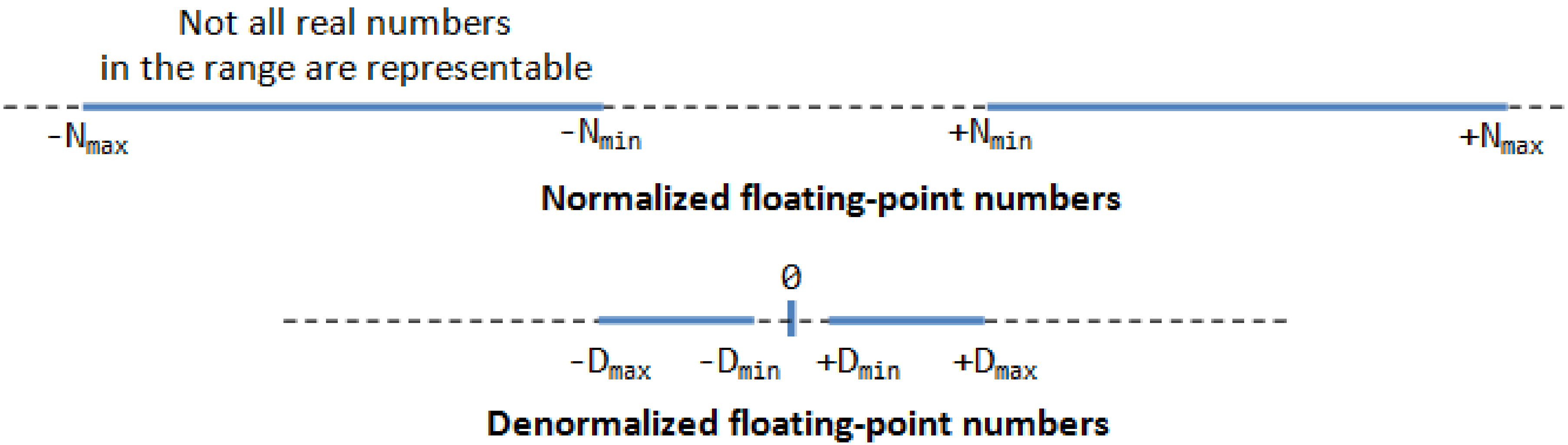
Solution for Denormalized numbers

- Gradual underflow
- Representation: Exponent = 0, Fraction=nonzero
- Interpretation: no leading 1 before binary point, implicit exponent = -126.
- Smallest representable positive num: 2^{-149}
- Second smallest representable positive num: 2^{-148}



Normalized vs. Denormal Numbers

https://www3.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html



IEEE 754 Floating-Point Numbers

S	Exponent	Fraction

$$(-1)^S x(1+F)x2^{(E\text{-bias})}$$

Single Precision		**Double Precision**		**Object Represented**
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	Non-zero	0	Nonzero	± Denormalized number
1 - 254	Anything	1 - 2046	Anything	± FP number
255	0	2047	0	± infinity
255	Nonzero	2047	Nonzero	NaN

Converting Binary Fractions to Decimal FP

0	0110 1000	101 0000 0000 0000 0000 0010

- **Sign: 0 => positive**
- **Exponent:**
  - $0110\ 1000_{two} = 104_{ten}$
  - Bias adjustment: 104 - 127 = -23
- **Significand:**
  - $1 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3} + 1x2^{-22} + \ldots$
    $=1+2^{-1}+2^{-3}+2^{-22}$
- **Final answer:**

$(1+2^{-1}+2^{-3}+2^{-22}) * 2^{-23}$

Converting Decimal Fractions to Binary

Simple Case

- denominator is power of 2 (2, 4, 8, 16, etc.)

Show F.P. representation of -0.75

- $-0.75 = -3/4$
- $-11_{two}/100_{two} = -0.11_{two}$
- Normalized to $-1.1_{two} \times 2^{-1}$
- $(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent}-127)}$
- $(-1)^1 \times (1 + .100\ 0000\ \ldots\ 0000) \times 2^{(126-127)}$

1	**0111 1110**	**100 0000 0000 0000 0000 0000**

Converting Decimal Fractions to Binary

- **Complicated case**
  - Denominator is not power of 2
  - Cannot represent the number precisely
- **How to get the significand of a never-ending number?**
  - Fact:
    - All rational numbers have a repeating pattern in decimal and binary
  - For conversion:
    - Write out binary number with repeating pattern.
    - Cut it off after correct number of bits
      - different for single vs. double precision
    - Derive Sign, Exponent and Significand fields

Converting Decimal to Binary FP: Method1

- **1/3**

$= 0.33333\ldots_{10}$

$= 0.25 + 0.0625 + 0.015625 + 0.00390625 + \ldots$

$= 1/4 + 1/16 + 1/64 + 1/256 + \ldots$

$= 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} + \ldots$

$= 0.0101010101\ldots_2 * 2^0$

$= 1.0101010101\ldots_2 * 2^{-2}$

- Sign: 0
- Exponent = -2 + 127 = 125 = 01111101
- Fraction = 0101010101…

0	**0111 1101**	**0101 0101 0101 0101 0101 010**

Converting Decimal Fractions to Binary FP: Method 2

- **Use a table with columns old, bit, and new**
- **Work across each row**
  - Multiply the old entry by 2 to get a new value
  - Put the integer part of the new value in the bit entry
  - Put the fraction part in the new entry
- **If the new entry is non-zero, copy it to the old entry in the next row down and continue; o/w, done.**
- **Example**

old	bit	new
.375	0	.75
0.75	1	.5
.5	1	0

So the result is .011

In-class Exercises

- Convert 20_{ten} to single-precision floating point format

- Convert 0.1_{ten} to single-precision floating point format

Answer

- $20_{ten} = 10100_{two} = 1.01 * 2^{4} = 0\ 10000011\ 010000000\ldots$
  - Biased exponent = 4+127 = 131

- $0.1_{ten} = 0.0\ 0011\ 0011\ 0011\ \ldots_{two}$

  $= 1.1\ 0011\ 0011\ 0011\ldots * 2^{-4}$

0 01111011 1 0011 0011 0011 0011 0011 00