ECE260: Fundamentals of Computer Engineering

Floating Point Numbers & Representation

James Moscola
Dept. of Engineering & Computer Science
York College of Pennsylvania

Floating-Point Numbers

- **Floating point numbers are numbers that have "floating" decimal points**
- **Used to represent non-integer numbers, including:**
  - Real numbers
    - Examples: 99.2 3.14159265359
  - Very small numbers such as fractions
    - Examples: 0.00187 -0.1211
  - Very large numbers that cannot be represented using using the provided word size
    - Examples: 987.02×10^{9} -0.002×10^{-4}
- **In many programming languages, declare floating point numbers using *float* or *double* keyword**
  - Two different floating-point representations: ***single-precision*** and ***double-precision***

Single-Precision Floating-Point Representation

- Represented using a single 32-bit word
  - Sign bit (s) specifies sign of the floating-point value
    - 0 indicates a positive / 1 indicates a negative
  - Includes 8-bit exponent and 23-bit fraction
  - Value of floating-point number is computed as: with a **bias of 127** for single-precision values

$$x = (-1)^{s} \times (1 + \text{fraction}) \times 2^{(\text{exponent - bias})}$$

index	31	30–23	22–0
	s	exponent	fraction
	1 bit	8 bits	23 bits

index 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Double-Precision Floating-Point Representation

- Represented using a ***TWO*** 32-bit words (totaling 64-bits)
  - Sign bit (s) specifies sign of the floating-point value
    - 0 indicates a positive / 1 indicates a negative
  - Includes 11-bit exponent and 52-bit fraction
  - Value of floating-point number is computed as: with a **bias of 1023** for double-precision values

$$x = (-1)^s \times (1 + \text{fraction}) \times 2^{(\text{exponent - bias})}$$

index	31	30–20	19–0
	s	exponent	fraction
	1 bit	11 bits	20 bits

fraction (continued)
32 bits

Bias? 😕 What the heck is that?

- Floating-point numbers can be negative AND also have a negative exponent
  - Example: -0.002×10^{-4}
- The sign bit of a floating-point number, (s), indicates the sign of the entire value, not the exponent
- How about embedding a second sign bit in the exponent (bit 30) and storing as 2's complement?
  - Meh .. it would work, but it would make comparing floating-point values difficult
    - Direct comparison of binary floating-point values would not be possible since negative numbers would "appear" larger than positive numbers
- Instead, bias the exponent value by the largest positive value and adjust exponent when interpreting value of floating-point number
  - Enables DIRECT comparison of binary floating-point numbers

Single-Precision Range

- Exponents 0000_0000_{two} and 1111_1111_{two} are reserved
  - Exponent 0_{ten} with a fraction of 0_{ten} indicates the value zero
  - Exponent 255_{ten} with a fraction of 0_{ten} indicates the value ∞
  - Exponent 255_{ten} with any nonzero fraction indicates the value NaN (Not a Number)

$$x = (-1)^s \times (1 + \text{fraction}) \times 2^{(\text{exponent - bias})}$$

- Smallest value in single-precision range:
  - Binary exponent value: 0000_0001_{two}
    Actual exponent after biasing: $1_{ten} - 127_{ten} = -126_{ten}$
  - Binary fraction value: $000\_0000\_0000\_0000\_0000\_0000_{two}$
    Significand: $1_{ten} + 0_{ten} = 1_{ten}$
  - Final value: $\pm$ significand $\times 2^{\text{actual\_exponent}}$
    $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$

Single-Precision Range (continued)

- Largest value in single-precision range:
  - Binary exponent value: 1111_1110_{two}
    Actual exponent after biasing: $254_{ten} - 127_{ten} = +127_{ten}$
  - Binary fraction value: $111\_1111\_1111\_1111\_1111\_1111_{two}$
    Fraction value: $\approx 1_{ten}$
    Significand: $1_{ten} + 1_{ten} \approx 2_{ten}$
  - Final value: $\pm$ significand $\times 2^{\text{actual\_exponent}}$
    $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

Double-Precision Range

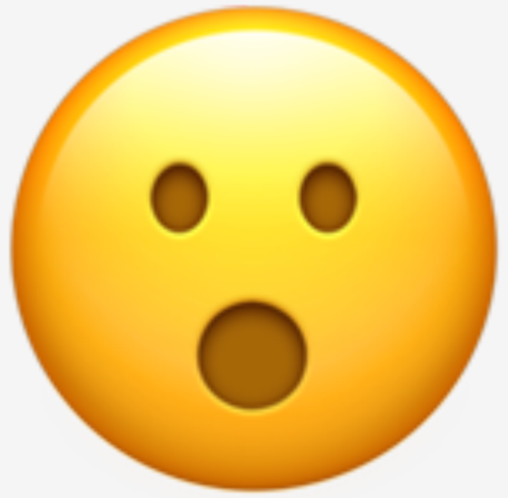
- Exponents $000\_0000\_0000_{two}$ and $111\_1111\_1111_{two}$ are reserved
  - Exponent 0_{ten} with a fraction of 0_{ten} indicates the value zero
  - Exponent 2047_{ten} with a fraction of 0_{ten} indicates the value ∞
  - Exponent 2047_{ten} with any nonzero fraction indicates the value NaN (Not a Number)

$$x = (-1)^s \times (1 + \text{fraction}) \times 2^{(\text{exponent - bias})}$$

- Smallest value in double-precision range:
  - Binary exponent value: $000\_0000\_0001_{two}$
    Actual exponent after biasing: $1_{ten} - 1023_{ten} = -1022_{ten}$
  - Binary fraction value: $000\_0000\_0000 \ldots \ldots 0000\_0000\_0000_{two}$ (52 bits of zeros)
    Significand: $1_{ten} + 0_{ten} = 1_{ten}$
  - Final value: $\pm$ significand $\times 2^{\text{actual\_exponent}}$
    $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$

Double-Precision Range (continued)

- **Largest value in double-precision range:**
  - Binary exponent value: $111\_1111\_1110_{two}$
    Actual exponent after biasing: $2046_{ten} - 1023_{ten} = +1023_{ten}$
  - Binary fraction value: $111\_1111\_1111 \ldots \ldots 1111\_1111\_1111_{two}$ (52 bits of ones)
    Fraction value: $\approx 1_{ten}$
    Significand: $1_{ten} + 1_{ten} \approx 2_{ten}$
  - Final value: $\pm$ significand $\times 2^{\text{actual\_exponent}}$
    $\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{+308}$



Floating-Point Precision

- Single-precision
  - Approximately 2^{-23}
  - Can represent a value x and $(x \pm 2^{-23})$, but not the numbers in between
- Double-precision
  - Approximately 2^{-52}
  - Can represent a value x and $(x \pm 2^{-52})$, but not the numbers in between

Floating-Point Example (bin -> float)

- What number is represented by the single-precision floating-point value?

alt. index		7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23
index	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1 bit	8 bits								23 bits																						

$$x = (-1)^s \times (1 + \text{fraction}) \times 2^{(\text{exponent - bias})}$$

- Sign bit: 1 (negative number)
  Exponent: $1000\_0001_{two} = 129_{ten}$
  Fraction: $010\_0000\_0000\_0000\_0000\_0000_{two}$
  $(0 \times 2^{-1}) + (1 \times 2^{-2}) + \ldots + (0 \times 2^{-23}) = 0 + (1 \times ¼) + \ldots + 0 = ¼ = .25$

$x = (-1)^s \times (1 + \text{fraction}) \times 2^{(\text{exponent - bias})}$
$= (-1)^1 \times (1 + .25) \times 2^{(129 - 127)}$
$= -1 \times 1.25 \times 2^2$
$= -5.0$

Floating-Point Example (float -> bin)

- What is the binary representation of the decimal number 29.28125 in single-precision float format?
  - Step #1 – Rewrite whole number portion in binary: $29_{ten} = 11101_{two}$
  - Step #2 – Rewrite fractional portion in binary: $0.28125_{ten} = 010\_0100\_0000\_0000\_0000\_0000_{two}$
    - set bit -1? // $2^{-1} = 0.5$ // 0.5 > 0.28125 (no, too big) // 0xx_xxxx_xxxx_xxxx_xxxx_xxxx
    - set bit -2? // $2^{-2} = 0.25$ // 0.25 ≤ 0.28125 (yes, it fits) // 01x_xxxx_xxxx_xxxx_xxxx_xxxx
      - 0.28125 - 0.25 = 0.03125 remains
    - set bit -3? // $2^{-3} = 0.125$ // 0.125 > 0.03125 (no, too big) // 010_xxxx_xxxx_xxxx_xxxx_xxxx
    - set bit -4? // $2^{-4} = 0.0625$ // 0.0625 > 0.03125 (no, too big) // 010_0xxx_xxxx_xxxx_xxxx_xxxx
    - set bit -5? // $2^{-5} = 0.03125$ // 0.03125 ≤ 0.03125 (yes, it fits) // 010_01xx_xxxx_xxxx_xxxx_xxxx
      - 0.03125 - 0.03125 = 0 remains (DONE)
    - set bit -6 through bit -23 to 0 // 010_0100_0000_0000_0000_0000

Floating-Point Example (float -> bin) (continued)

- What is the binary representation of the decimal number 29.28125 in single-precision float format?
  - Step #3 – Combine the rewritten components: $11101.01001000000000000000000_{two}$
  - Step #4 – Normalize the value (shift decimal): $1.11010100100000000000000_{two} \times 2^4$
    - Normalized value represents the (1 + fraction), drop the leading 1 from the normalized value
  - Step #5 – Determine sign bit and exponent bits:
    - Original value, 29.28125, was positive so: $s = 0$
    - Exponent from normalized value = 4, add to bias to determine exponent bits
      - exponent = 4 + bias = 4 + 127 = 131_{ten} = 1000_0011_{two}
  - Step #6 – Put it all together:
    - s _ exponent _ fraction = $0\_1000\_0011\_110\_1010\_0100\_0000\_0000\_0000_{two}$
      = $0100\_0001\_1110\_1010\_0100\_0000\_0000\_0000_{two}$
      = $41EA4000_{hex}$