Floating Point numbers

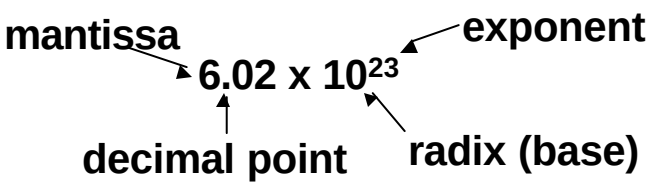
Review of Numbers

- **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)**

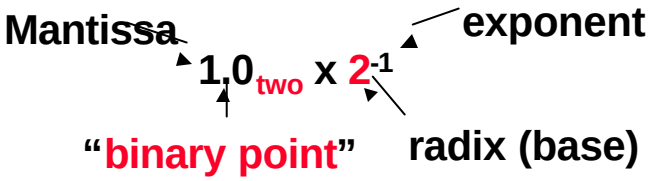
    $-2^{(N-1)}$ to $2^{(N-1)} - 1$
- **What about other numbers?**
  - **Very large numbers? (seconds/century)**
    $3{,}155{,}760{,}000_{10}$ ($3.15576_{10} \times 10^9$)
  - **Very small numbers? (atomic diameter)**
    0.00000001_{10} ($1.0_{10} \times 10^{-8}$)
  - **Rational (repeating pattern) 2/3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

Scientific Notation: Review



- Normalized form: no leadings 0s (exactly one digit to left of decimal point)
- Alternatives to representing 1/1,000,000,000
  - Normalized: 1.0×10^{-9}
  - Not normalized: 0.1×10^{-8}, 10.0×10^{-10}

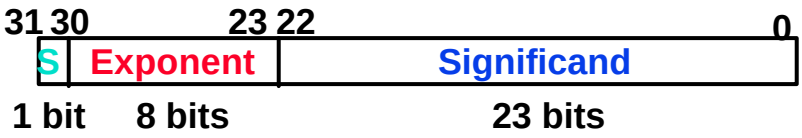
Scientific Notation: Binary Numbers



- Computer arithmetic that supports it called floating point, because it represents numbers where binary point is not fixed, as it is for integers
  - Declare such variable in C as `float`

Floating Point (FP) Representation (1/2)

- Normal format: $+1.xxxxxxxxxx_{two} * 2^{yyyy}{}_{two}$
- Multiple of Word Size (32 bits)



- S represents Sign

  Exponent represents y's

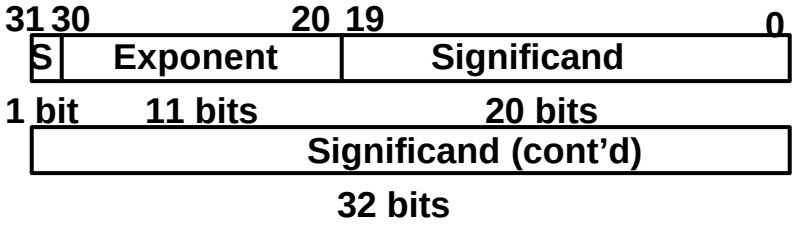
  Significand represents x's
- Represent numbers as small as 2.0×10^{-38} to as large as 2.0×10^{38}

FP Representation (2/2)

- What if result too large? (> 2.0×10^{38})
  - Overflow!
  - Overflow => Exponent larger than represented in 8-bit Exponent field
- What if result too small? (>0, < 2.0×10^{-38})
  - Underflow!
  - Underflow => Negative exponent larger than represented in 8-bit Exponent field
- How to reduce chances of overflow or underflow?

Double Precision FP Representation

- Next Multiple of Word Size (64 bits)



- Double Precision (vs. Single Precision)
  - C variable declared as `double`
  - Represent numbers almost as small as 2.0×10^{-308} to almost as large as 2.0×10^{308}
  - But primary advantage is greater accuracy due to larger significand

IEEE 754 FP Standard (1/4)

- Single Precision, DP similar
- Sign bit: 1 means negative
  0 means positive
- Significand:
  - To pack more bits, leading 1 implicit for normalized numbers
  - 1 + 23 bits single, 1 + 52 bits double
  - always true: Significand < 1 (for normalized numbers)
- Note: 0 has no leading 1, so reserve exponent value 0 just for number 0

IEEE 754 FP Standard (2/4)

- Kahan wanted FP numbers to be used even if no FP hardware; e.g., sort records with FP numbers using integer compares
- Could break FP number into 3 parts: compare signs, then compare exponents, then compare significands
- Wanted it to be faster, single compare if possible, especially if positive numbers
- Then want order:
  - Highest order bit is sign (negative < positive)
  - Exponent next, so big exponent => bigger #
  - Significand last: exponents same => bigger #

IEEE 754 FP Standard (3/4)

- Negative Exponent?
  - 2's comp? 1.0×2^{-1} v. $1.0 \times 2^{+1}$ (1/2 v. 2)

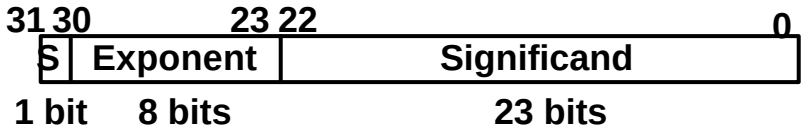
1/2	0	1111 1111	000 0000 0000 0000 0000 0000
2	0	0000 0001	000 0000 0000 0000 0000 0000

  - This notation using integer compare of 1/2 v. 2 makes 1/2 > 2!
- Instead, pick notation 0000 0001 is most negative, and 1111 1111 is most positive
  - 1.0×2^{-1} v. $1.0 \times 2^{+1}$ (1/2 v. 2)

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

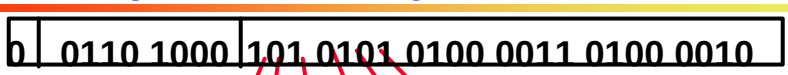
IEEE 754 FP Standard (4/4)

- Called Biased Notation, where bias is number subtract to get real number
  - IEEE 754 uses bias of 127 for single prec.
  - Subtract 127 from Exponent field to get actual value for exponent
  - 1023 is bias for double precision
- Summary (single precision):



- $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$
  - Double precision identical, except with exponent bias of 1023

Example: Converting FP to Decimal



- Sign: 0 => positive
- Exponent:
  - $0110\ 1000_{two} = 104_{ten}$
  - Bias adjustment: 104 - 127 = -23
- Significand:
  - $1 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3} + 0x2^{-4} + 1x2^{-5} + \ldots$
    $= 1 + 2^{-1} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-14} + 2^{-15} + 2^{-17} + 2^{-22}$
    $= 1.0 + 0.666115$
- Represents: $1.666115_{ten} * 2^{-23} \sim 1.986 * 10^{-7}$

Converting Decimal to FP

- Simple Case: If denominator is an exponent of 2 (2, 4, 8, 16, etc.), then it's easy.
- Show MIPS representation of -0.75
  - -0.75 = -3/4
  - $-11_{two}/100_{two} = -0.11_{two}$
  - Normalized to $-1.1_{two} \times 2^{-1}$
  - $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$
  - $(-1)^1 \times (1 + .100\ 0000\ ...\ 0000) \times 2^{(126-127)}$

1	0111 1110	100 0000 0000 0000 0000 0000

Another Example

- 1/3
  - $= 0.33333\ldots_{10}$
  - $= 0.25 + 0.0625 + 0.015625 + 0.00390625 + 0.0009765625 + \ldots$
  - $= 1/4 + 1/16 + 1/64 + 1/256 + 1/1024 + \ldots$
  - $= 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} + 2^{-10} + \ldots$
  - $= 0.0101010101\ldots_2 * 2^0$
  - $= 1.0101010101\ldots_2 * 2^{-2}$

0	0111 1101	0101 0101 0101 0101 0101 010

Representation for +/- Infinity

- In FP, divide by zero should produce +/- infinity, not overflow.
- Why?
  - OK to do further computations with infinity e.g., X/0 > Y may be a valid comparison
  - Ask math majors
- IEEE 754 represents +/- infinity
  - Most positive exponent reserved for infinity
  - Significand all zeroes

Representation for 0

- Represent 0?
  - exponent all zeroes
  - significand all zeroes too
  - What about sign?
  - `+0: 0 00000000 00000000000000000000000`
  - `-0: 1 00000000 00000000000000000000000`
- Why two zeroes?
  - Helps in some limit comparisons

Special Numbers

- What have we defined so far? (Single Precision)

Exponent	Significand	Object
0	0	0
0	nonzero	???
1-254	anything	+/- fl. pt. #
255	0	+/- infinity
255	nonzero	???

Representation for Not a Number

- What do I get if I calculate

  `sqrt(-4.0)` or `0/0`?

  - If infinity is not an error, these shouldn't be either.
  - Called Not a Number (NaN)
  - Exponent = 255, Significand nonzero

- Why is this useful?
  - Hope NaNs help with debugging?
  - They contaminate: op(NaN,X) = NaN

Special Numbers (cont'd)

- What have we defined so far? (Single Precision)?

Exponent	Significand	Object
0	0	0
0	nonzero	???
1-254	anything	+/- fl. pt. #
255	0	+/- infinity
255	nonzero	NaN

Representation for Denorms (1/2)

- Problem: There's a gap among representable FP numbers around 0
  - Smallest representable pos num:

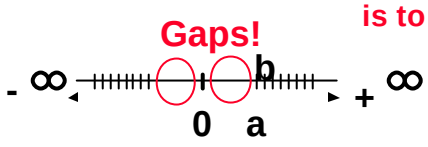
    $a = 1.0\ldots_2 * 2^{-126} = 2^{-126}$
  - Second smallest representable pos num:

    $b = 1.000\ldots\ldots1_2 * 2^{-126} = 2^{-126} + 2^{-149}$

  $a - 0 = 2^{-126}$

  $b - a = 2^{-149}$

Normalization and implicit 1 is to blame!

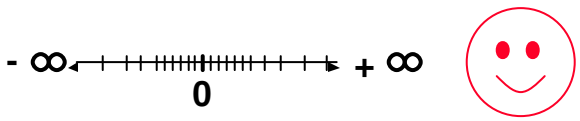


Representation for Denorms (2/2)

- **Solution:**
  - We still haven't used Exponent = 0, Significand nonzero
  - Denormalized number: no leading 1, implicit exponent = -126.
  - Smallest representable pos num:

    $a = 2^{-149}$
  - Second smallest representable pos num:

    $b = 2^{-148}$



Question

What is the decimal equivalent of:

1	1000 0001	111 0000 0000 0000 0000 0000

A: A: -3.5
B: B: -3.75
C: C: -7
D: D: -7.5
E: E: -15
F: F: -7 * 2^129

Answer

What is the decimal equivalent of:

1	1000 0001	111 0000 0000 0000 0000 0000
S	Exponent	Significand

$(-1)^{S} \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

$(-1)^{1} \times (1 + .111) \times 2^{(129-127)}$

$-1 \times (1.111) \times 2^{(2)} = -111.1 = -7.5$

A: A: -3.5
B: B: -3.75
C: C: -7
D: D: -7.5
E: E: -15
F: F: -7 * 2^129