CS 4400
Computer Systems

LECTURE 4

Representing floats

Floating-point arithmetic

Floating Point

- Floating-point representation encodes rational numbers of the form $V = x \times 2^y$.
- Useful for numbers very large and very close to 0, why?

Floating Point

- Until the 1980s, there were many different conventions for how to represent floats and the operations on them.
- Accuracy was not the biggest concern, what was?

Floating Point

- Around 1985, IEEE Standard 754 surfaced as a carefully crafted standard for floating point.
  – by Kahan et al., now supported by virtually all computers

Fixed-Point Fractional Numbers

- Decimal: $d_m\, d_{m-1} \cdots d_1\, d_0 . d_{-1}\, d_{-2} \cdots d_{-n}$ $\qquad d = \sum_{i=-n}^{m} 10^i \times d_i$

- Binary: $b_m\, b_{m-1} \cdots b_1\, b_0 . b_{-1}\, b_{-2} \cdots b_{-n}$ $\qquad b = \sum_{i=-n}^{m} 2^i \times b_i$

- Example: $101.11_2 = 2^2 + 2^0 + 2^{-1} + 2^{-2} = 5\ ¾$

- What is the effect of shifting the binary point right/left?
- Are all fractions representable?

Fixed-Point Fractional Numbers

- With finite-length encodings, there are decimal (and binary) fractions that cannot be represented exactly.
- $1/3 = 0.33333..._{10}$
- $1/5 = 0.001100110011..._{2}$

Question

Represent the value 51/32 as a fixed-point binary number.

A. 0.010011
B. 0.100101
C. 1.100011
D. 1.100110
E. It cannot be represented exactly

IEEE Floating-Point Representation

- Represents a number of the form $V = (-1)^s \times M \times 2^E$

- s: sign bit
- E: exponent (`exp`) field, weights by a power of 2
  - k bits ($k=8$ for single precision, $k=11$ for double)
- M: significand (`frac`) field, a fractional binary number
  - n bits ($n=23$ for single precision, $n=52$ for double)

- The value encoded by a given bit representation is divided into **three cases**, depending on the value of `exp`.

IEEE Floating-Point Representation

- Represents a number of the form $V = (-1)^s \times M \times 2^E$
- ***s***: sign bit
  - interpretation for numeric value 0 is special
- ***E***: exponent (`exp`) field, weights by a power of 2
  – *k* bits (*k=8* for single precision, *k=11* for double)
- ***M***: significand, a fractional binary number
  – ranges [1, 2) or [0, 1), depending on whether the `exp` field is 0
  – *n* bits (*n=23* for single precision, *n=52* for double), `frac` field

- The value encoded by a given bit representation is divided into **three cases**, depending on the value of `exp`.

Case 1: Normalized Values

- Occurs when bit pattern of `exp` is neither all 0s nor all 1s.

- `exp` field interpreted as a signed integer in biased form
  - Let e = unsigned number represented by `exp`
  - Bias = $2^{k-1} - 1$
  - The actual exponent value is $E = e - (2^{k-1} - 1)$.
  - For double ($k=11$), $-1022 \leq E \leq 1023$. For single ($k=8$)?

Case 1: Normalized Values

- `frac` field interpreted as fixed point fractional value $0 \leq f < 1$
  - The significand value: $M = 1 + f$
  - "Implied leading 1" representation gets additional bit for free
  - Thus, the range of M is [1,2).

Question

Recall: single precision uses 8 `exp` bits and 23 `frac` bits

$$E = e - (2^{k-1} - 1),\quad M = 1 + f,\quad V = (-1)^s \times M \times 2^E$$

What is V for 0 01111111 00000000000000000000000?

A. 0
B. 0.5
C. 1
D. 2
E. It is not a normalized value.

Case 2: Denormalized Values

- Occurs when bit pattern of `exp` = 0
- The exponent value: $\boldsymbol{E = 1 - (2^{k-1} - 1)}$
- The significand value $\boldsymbol{M = f}$
  - Without "implied leading 1".
  - Thus, the range of *M* is [0,1).

Case 2: Denormalized Values

- Why have denormalized numbers?
  - Can represent numeric value 0. Why cannot with normalized?
  - Can represent numbers very close to 0.
  - *Gradual underflow*—possible values are spaced evenly near 0.0

- Is there only 1 zero value?

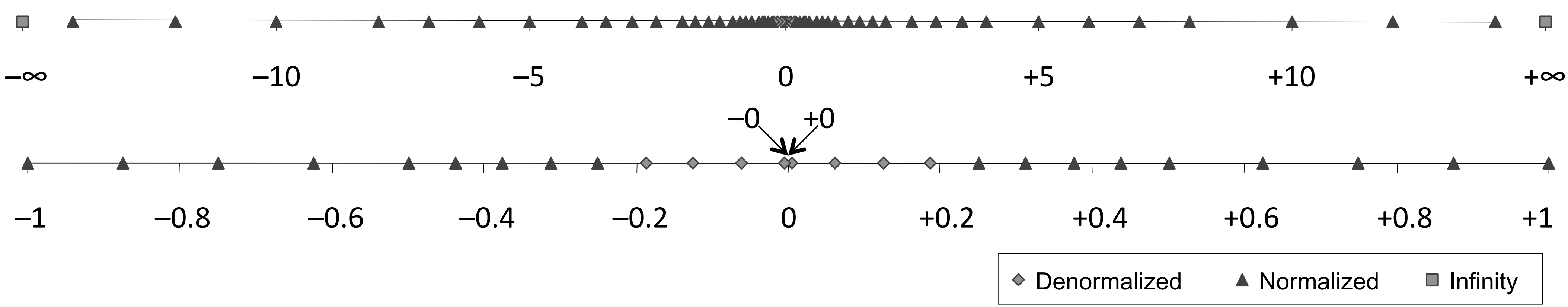
Case 3: Special Values

- Occurs when bit pattern of `exp` is all 1s (numeric value 255 for single or 2047 for double)

- When the `frac` = 0
  - Value is ∞ (s: positive or negative).
- When the `frac` field is nonzero:
  - Value is "`NaN`" (Not a Number)
  - Is there only 1 NaN value?

Example: 6-bit Format

- Assume a hypothetical 6-bit format with k=3 exponent bits and n=2 significand bits. What is the exponent bias?



- What are the normalized numbers with maximum magnitude?

  e = ?   E = ?   f = ?   M = ?   V = ?

- Are the representable numbers uniformly distributed?

Exercises: 5-bit Format

- Assume a hypothetical 5-bit format with *k=2* exponent bits and *n=2* significand bits. The exponent bias is $2^{k-1} - 1 = 1$.

s e_1e_0 f_1f_0	e	E	f	M	V
0 00 00					
0 00 10					
0 01 01					
0 10 11					
0 11 00					
0 11 10					

Exercises: 5-bit Format

- Assume a hypothetical 5-bit format with *k=2* exponent bits and *n=2* significand bits. The exponent bias is $2^{k-1} - 1 = 1$.

s e_1e_0 f_1f_0	e	E	f	M	V
0 00 00					
0 00 10					Case 1 – Normalized Values
0 01 01	1	1-1 = 0	¼	1 + ¼ = 1.25	$1.25*2^0 = 1.25$
0 10 11	2	2-1 = 1	½ + ¼	1 + ½ + ¼ = 1.75	$1.75*2^1 = 3.50$
0 11 00					
0 11 10					

Exercises: 5-bit Format

- Assume a hypothetical 5-bit format with *k=2* exponent bits and *n=2* significand bits. The exponent bias is $2^{k-1} - 1 = 1$.

Case 2 – Denormalized Values

s e_1e_0 f_1f_0	e	E	f	M	V
0 00 00		**1-1 = 0**	**0**	**0**	**0**
0 00 10		**1-1 = 0**	**½**	**½**	**0.5**
0 01 01	1	1-1 = 0	¼	1 + ¼ = 1.25	$1.25*2^0$ = 1.25
0 10 11	2	2-1 = 1	½ + ¼	1 + ½ + ¼ = 1.75	$1.75*2^1$ = 3.50
0 11 00					
0 11 10					

Exercises: 5-bit Format

- Assume a hypothetical 5-bit format with *k=2* exponent bits and *n=2* significand bits. The exponent bias is $2^{k-1} - 1 = 1$.

s e_1e_0 f_1f_0	e	E	f	M	V
0 00 00		1-1 = 0	0	0	0
0 00 10		1-1 = 0	½	½	0.5
0 01 01	1	1-1 = 0	¼	1 + ¼ = 1.25	$1.25 * 2^0 = 1.25$
0 10 11	2	2-1 = 1	½ + ¼	1 + ½ + ¼ = 1.75	$1.75 * 2^1 = 3.50$
0 11 00					∞
0 11 10					**NaN**

Case 3 – Special Cases

Properties of IEEE Floating Point

- The value **+0.0** always has a **bit pattern of all 0s**
- The **smallest denormalized value > 0** has a bit pattern consisting of **1 in LSB** and all **0s elsewhere**.
  - $M = f = 2^{-n}$, $E = 1 - (2^{k-1} - 1) = -2^{k-1} + 2$
  - $V = M \times 2^E = 2\text{\^{}}(-n-2^{k-1}+2)$

Properties of IEEE Floating Point

- The **largest denormalized value** has a bit pattern consisting of an **all-0** `exp` field and an **all-1** `frac` field.
  - $M = f = 1 - epsilon$, $E = 1 - (2^{k-1} - 1) = -2^{k-1} + 2$
  - $V = M \times 2^E = (1 - epsilon) \times 2\text{^}(-2^{k-1}+2)$

- The **smallest normalized value** > 0 has a bit pattern consisting of **1 in LSB of** `exp` field and all **0s elsewhere**.
  - $M = 1 + f = 1$, $E = e - (2^{k-1} - 1) = -2^{k-1} + 2$
  - $V = M \times 2^E = 2\text{^}(-2^{k-1}+2)$

Properties of IEEE Floating Point

- The value **1.0** has a bit pattern with **all but the MSB of the `exp` field set to 1** and **all other bits set to 0**
  - $M = 1 + f = 1,\ E = e - (2^{k-1} - 1) = 0$

- The **largest normalized value** has a bit pattern consisting of **0 in LSB of `exp`** field and all **1s elsewhere**
  - $M = 1 - f = 2 -$ epsilon, $E = e - (2^{k-1} - 1) = 2^{k-1} - 1$
  - $V = M \times 2^E = (2 -$ epsilon$) \times 2\text{^}(2^{k-1} - 1)$

Rounding

- The key problem is to define the direction to round a value that is between two possibilities.

- For a real value x, find the “closest” matching x‘ representable in floating-point format.
- Another approach is to determine representable values x^- and x^+ such that $x^- \leq x \leq x^+$ is guaranteed.

- IEEE floating-point format defines four rounding modes.
  – The default mode finds x'.
  – The other three can be used to compute x^- and x^+.

Rounding Modes

- `Round-to-even` (aka `round-to-nearest`) mode—default
  - rounds either upward or downward such that least-significant digit of the result is even, e.g., both \$1.50 and \$2.50 → \$2
- `Round-to-zero` mode
  - rounds positive numbers downward and negative numbers upward, giving value x'' such that $| x'' | \leq | x |$
- `Round-up` mode
  - rounds all numbers upward, giving value x^- such that $x^- \leq x$
- `Round-down` mode
  - rounds all numbers downward, giving value x^+ such that $x \leq x^+$

Floating-Point Operations

- The result of floating-point addition or multiplication is simply the exact result of the operation defined over real numbers, and then rounded (to be representable).

- Floating-point addition is not associative (single precision example)

  `(3.14 + 1e10) - 1e10` = `0.0`

  `3.14 + (1e10 - 1e10)` = `3.14`

- Floating-point multiplication is not associative or distributive over addition.

  `1e20 * (1e20 - 1e20)` = `0.0`

  `1e20 * 1e20 - 1e20 * 1e20` = `NaN`

Question

In C, all `int` values can be represented as `float` values.

A. True
B. False

Floating Point in C

- Single precision: float, double precision: double

- Round-to-even mode
  - C standard does not require IEEE format—no (standard) way to change rounding modes or get special values.
  - Most systems provide access to such features, but details vary

- Casting among types changes numeric values as follows:
  - int to float: may be rounded
  - int/float to double: exact numeric value is preserved
  - double to float: may overflow or be rounded
  - float/double to int: truncated toward zero, may overflow

Questions

Always true?

Assume: `int x, float f, double d`

A. `x == (int)(float)x`

B. `x == (int)(double)x`

C. `f == (float)(double)f`

D. `d == (double)(float)d`

E. `f == -(-f)`

F. `2/3 == 2/3.0`

G. `(d >= 0.0) || ((d*2) < 0.0)`

H. `(d+f) - d == f`

Extended Precision

- Floating-point registers of the IA32 processors use 80-bit extended-precision format (with x87, not SSE, know your architecture!).
  - $k=15$ exponent bits, $n=63$ fraction bits
- When normal single- and double-precision numbers are loaded from memory, they are converted to this format.
- Arithmetic is always performed in the extended format.
- Numbers are converted back to single- or double precision as they are stored to memory
- Can lead to undesirable consequences (see text).

Summary: Representing Information

- Groups of bits are interpreted differently for integers, real numbers, and character strings.
  - encoding and byte-ordering conventions differ across machines
- C is designed to accommodate a wide range of word sizes and encodings.
  - most (but not all) machines use two's complement and IEEE format
- In casting between signed and unsigned integers, the underlying bit patterns do not change.
- Due to finite encoding length, properties of computer arithmetic differ from those of integer/real arithmetic.

Summary: Representing Information

- *Overflow*—a result exceeds representable range
- *Underflow*—a floating-point value is so close to 0.0, it is represented as such
- Properties of computer arithmetic allow compilers to do many optimizations.
  - such as replacing `7*x` with `(x<<3)-x`
- Floating-point arithmetic must be used carefully because of its limited range and precision, as well as, because it does not obey some common math properties