Floating Point Representation and Arithmetic

Topics

- Chapter 2.4
- IEEE Floating Point Standard
- Floating Point Arithmetic

Announcements

- Buffer Lab available on moodle, due Friday Oct 24
  - Similar to bomb lab: download binary and input strings
- Recitation Exercises #3 on floating point due next Monday Oct 20 in recitation
  - On moodle, TA discussed floating point on Monday
- Midterms graded, return in recitation next Monday
  - Can pick them up in TA office hours Thursday & Friday
- Essential that you read the textbook in detail & do the practice problems
  - Chapter 2.4 Floating Point
  - Then move on to Chapter 4

Recap: Representing Real Numbers

In decimal notation,

$$470.35_{10} = 4*10^2 + 7*10^1 + 0*10^0 + 3*10^{-1} + 5*10^{-2}$$

$$= 4.7035*10^2 \quad \text{(scientific notation)}$$

Similarly, in binary point:

$$5\ 3/4 = 5 + ¾ = 4 + 1 + ½ + ¼$$

$$= 1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 1*2^{-2}$$

$$= 101.11_2$$

$$= 1.0111*2^2 \quad \text{(~floating point notation)}$$

- Can rewrite a "real number with a binary point" as $1.XXXXXXX * 2^E$, where exponent E ranges from + to - values

Recap: Floating Point Representation

Numerical Form

- Real number = $(-1^s) * M * 2^E$
  - Sign bit ***s*** determines whether number is negative or positive
  - Significand ***M=1.XXXXX*** normally a fractional value in range [1.0,2.0).
  - Exponent ***E*** weights value by power of two

Encoding

- MSBit is sign bit
- `exp` field encodes ***E***
- `frac` field encodes ***M***

Single Precision Floating Point (32 bits)

- `exp` field = 8 bits, `frac` field = 23 bits

Double Precision Floating Point (64 bits)

- `exp` field = 11 bits, `frac` field = 52 bits

Single Precision Intuition

Real number = $(-1^s) * M * 2^E$

- **Encoding**
  - **MSB is sign bit, `exp` field encodes *E*, `frac` field encodes *M***
  - **Single precision: 8 `exp` bits, 23 `frac` bits**

Want exponent to vary from negative to positive values

- **Thus fractional values will be represented too, not just positive powers of 2**
- **A simple two's complement interpretation of `exp` gives a range of -128 to +127**
  - **But need to represent special floating point values like +/- ∞**
  - **Could use lowest negative value '10000000', but this is hard to detect**
  - **Instead, choose `exp`= '11111111' to signify special values**
  - **But this is -1 in two's complement, in the middle of FP range**

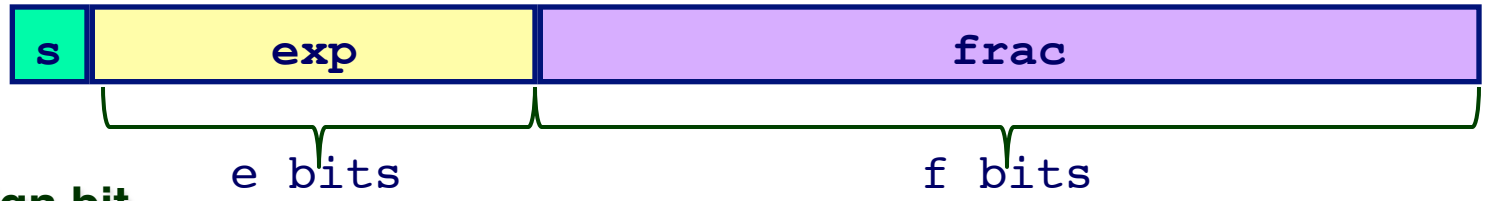
Single Precision Intuition

Real number = $(-1^s) * M * 2^E$

Want exponent to vary from negative to positive values

- **Choose unsigned representation for `exp`**
  - **Unsigned `exp` ranges from 0 to +255**
  - **Reserve `exp`= '11111111' for special values like +/- ∞, easy to detect, and is at upper edge of exponential range, not in the middle**
    - **`exp` now ranges from 0 to +254**
  - **To make E range from positive to negative, subtract a Bias = half the range = $2^{k-1} - 1 = 127$**
  - **E = `exp` – Bias, so E ranges from -127 to +127**
    - **But E's range is actually -126 to +127, because `exp`= all 0's reserved for *de-normalized (next slide)***

IEEE Floating Point Summary



- MSB `s` is sign bit
- `exp` field encodes E, and is `e` bits wide
- `frac` field encodes M, and is `f` bits wide

Bias = $2^{e-1}-1$, where e is # of exponent bits.

Floating point Value = $(-1)^S * M * 2^E$, except special cases.

3 Encoding cases:

If (`exp`!=all 0's && `exp`!=all 1's): // Normalized case

E = `exp`-Bias, M = 1.`frac`, i.e. Value = $(-1)^S * (1.\texttt{frac}) * 2^{\texttt{exp}-\text{Bias}}$

Else if (`exp`==all 1's): // Special cases

if (`frac`==all 0's): Value = +/- ∞ (infinity)

else Value = NAN

Else if (`exp`==all 0's): // De-normalized case for extra precision near 0

E = 1-Bias, M = 0.`frac`, i.e. Value = $(-1)^S * (0.\texttt{frac}) * 2^{1-\text{Bias}}$

Normalized Encoding Example

Value

`Float F = 15213.0;`

- $15213_{10} = 11101101101101_2 = 1.1101101101101_2 \times 2^{13}$

Significand

$M = 1.\underline{1101101101101}_2$

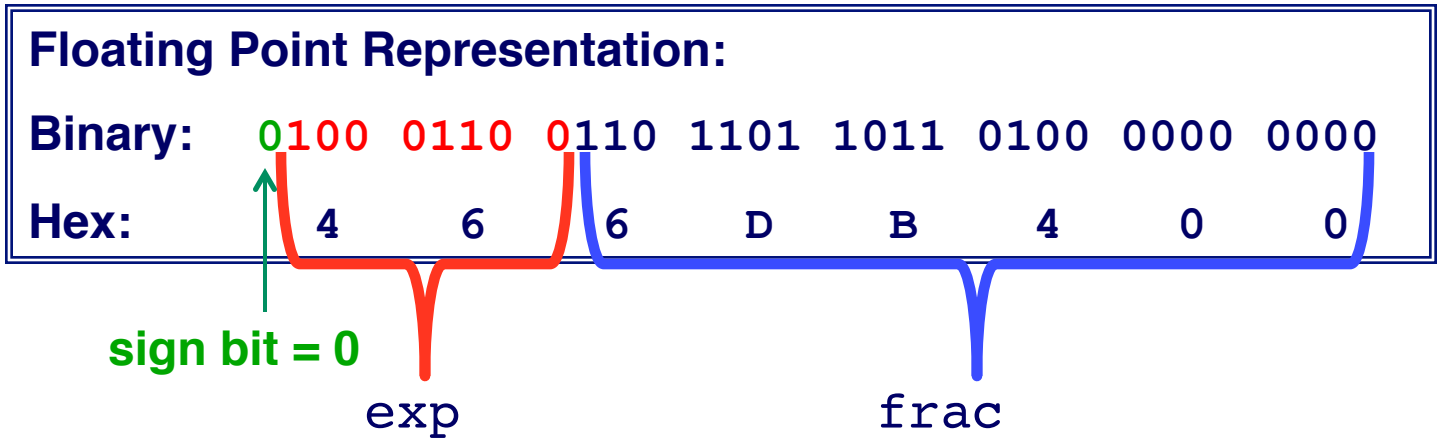
`frac=` $\underline{1101101101101}0000000000_2$

Exponent

$E = 13$

$Bias = 127$

$Exp = E + Bias = 140 = 10001100_2$



"Normalized" Numeric Values

Condition

- `exp` ≠ `000…0` and `exp` ≠ `111…1`

Exponent coded as *biased* value

E = *Exp* – *Bias*

- *Exp* : unsigned value denoted by `exp`
- *Bias* : Bias value
  - » Single precision: 127 (*Exp*: 1…254, *E*: -126…127)
  - » Double precision: 1023 (*Exp*: 1…2046, *E*: -1022…1023)
  - » in general: *Bias* = $2^{e-1} - 1$, where e is number of exponent bits

Significand coded with implied leading 1

M = `1.xxx…x`$_2$

- `xxx…x`: bits of `frac`
- Minimum when `000…0` (*M* = 1.0)
- Maximum when `111…1` (*M* = 2.0 – ε)
- Get extra leading bit for "free"

Denormalized Values Intuition

Range of normalized values for 32-bit single precision

- `exp` = 1 to +254, so E = -126 to +127
- So smallest possible value we can represent is ~ 2^{-126}
- = ~ 10^{-38}
- In some cases, this is not enough precision

Reserve `exp`=all 0's to get more precision near 0

- Let Exponent value *E* = 1–*Bias*
  - = -126 for single precision
- Let Significand value *M* = $0.xxx\ldots x_2$
  - `xxx…x`: bits of `frac`
- So now we can express 0.xxx…x * 2^{-126}
- Since the `frac` field 0.xxx…x has 23 bits, we get an additional 23 bits of precision!
  - i.e. if `frac=0000...0001` = 2^{-23}, then we get $2^{-23} * 2^{-126} = 2^{-149}$
  - ~ 10^{-45} (compare to ~ 10^{-38})

Denormalized Values

Condition

- `exp` = `000…0`

Value

- Exponent value $E = 1 - Bias$
- Significand value $M = 0.xxx\ldots_2$
  - `xxx…x`: bits of `frac`

Cases

- `exp` = `000…0`, `frac` = `000…0`
  - Represents value 0
  - Note that have distinct values +0 and –0
- `exp` = `000…0`, `frac` ≠ `000…0`
  - Numbers very close to 0.0
  - Lose precision as get smaller
  - “Gradual underflow”

Special Values

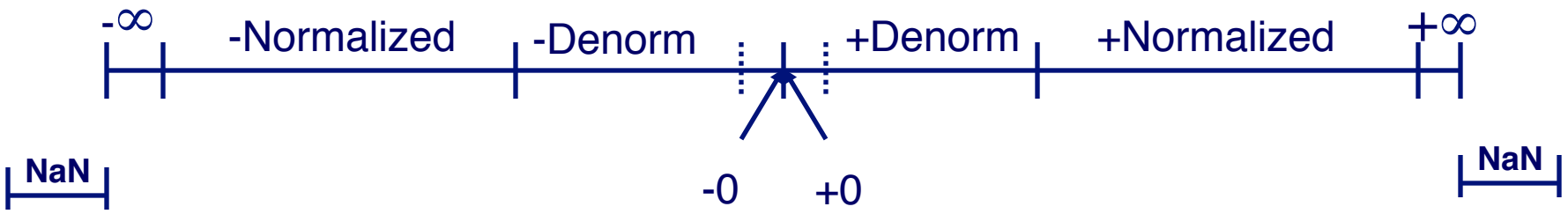
Condition

- `exp` = `111...1`

Cases

- `exp` = `111...1`, `frac` = `000...0`
  - Represents value ∞ (infinity)
  - Operation that overflows
  - Both positive and negative
  - E.g., 1.0/0.0 = −1.0/−0.0 = +∞, 1.0/−0.0 = −∞
- `exp` = `111...1`, `frac` ≠ `000...0`
  - Not-a-Number (NaN)
  - Represents case when no numeric value can be determined
  - E.g., sqrt(–1), ∞ − ∞

Summary of Floating Point Real Number Encodings

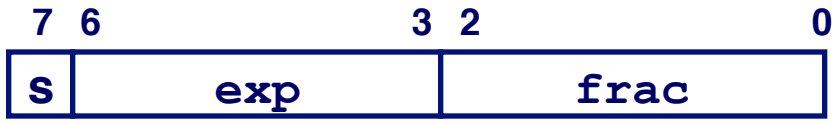


Tiny Floating Point Example

8-bit Floating Point Representation

- the sign bit is in the most significant bit.
- the next four bits are the exponent, so Bias = $2^{4-1}-1 = 7$.
- the last three bits are the `frac`

- **Same General Form as IEEE Format**
  - normalized, denormalized
  - representation of 0, NaN, infinity



Values Related to the Exponent

	Exp	exp	E=(exp -Bias)[1]	2^E	
denormalized	0	0000	-6	1/64	(denorms)
	1	0001	-6	1/64	
	2	0010	-5	1/32	
	3	0011	-4	1/16	
	4	0100	-3	1/8	
	5	0101	-2	1/4	
	6	0110	-1	1/2	
normalized	7	0111	0	1	
	8	1000	+1	2	
	9	1001	+2	4	
	10	1010	+3	8	
	11	1011	+4	16	
	12	1100	+5	32	
	13	1101	+6	64	
	14	1110	+7	128	
special values	15	1111	n/a		(inf, NaN)

highest normalized precision is 1/64,
i.e. 1.frac*2^(exp-Bias) is most precise when exp=1 (E=-6) and frac=0.

But we want more precision!
So use denormalized.

[1] normalized only

Dynamic Range

extra precision by `frac` bits

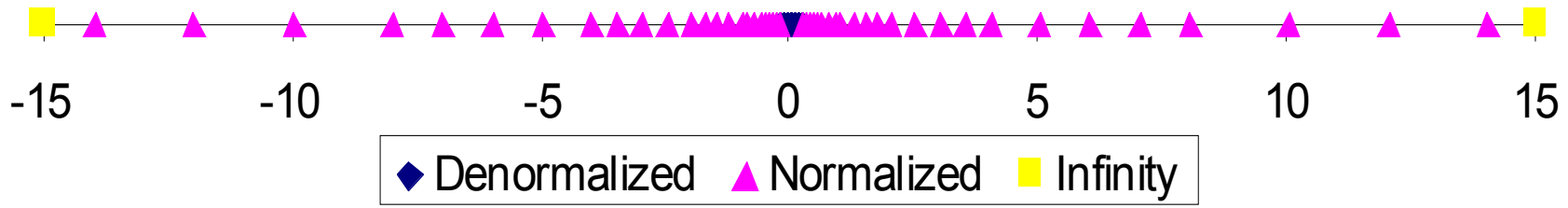
	s	exp	frac	E	Value	
Denormalized numbers Value= 0.frac*2^E, E=1-Bias	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	← closest to zero
	0	0000	010	-6	2/8*1/64 = 2/512	
	…					
	0	0000	110	-6	6/8*1/64 = 6/512	
	0	0000	111	-6	7/8*1/64 = 7/512	← largest denorm
	0	0001	000	-6	8/8*1/64 = 8/512	← smallest norm
	0	0001	001	-6	9/8*1/64 = 9/512	
	…					
Normalized Numbers Value= 1.frac*2^E, E=exp-Bias	0	0110	110	-1	14/8*1/2 = 14/16	
	0	0110	111	-1	15/8*1/2 = 15/16	← closest to 1 below
	0	0111	000	0	8/8*1 = 1	
	0	0111	001	0	9/8*1 = 9/8	← closest to 1 above
	0	0111	010	0	10/8*1 = 10/8	
	…					
	0	1110	110	7	14/8*128 = 224	
	0	1110	111	7	15/8*128 = 240	← largest norm
special values	0	1111	000	n/a	+ inf	

Distribution of Values

6-bit IEEE-like format

- e = 3 exponent bits
- f = 2 fraction bits
- Bias is 3

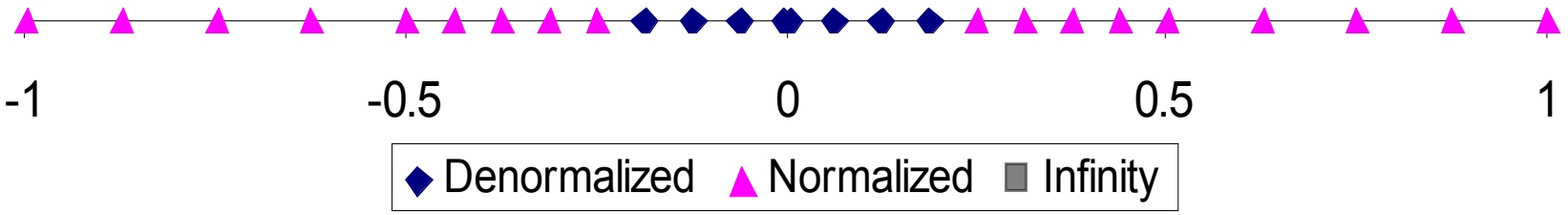
Notice how the distribution gets denser toward zero.



Distribution of Values (close-up view)

6-bit IEEE-like format

- e = 3 exponent bits
- f = 2 fraction bits
- Bias is 3



Floating Point Arithmetic Operations

Conceptual View

- Examples: FP Addition, FP Multiplication
- First compute exact result
- Make it fit into desired precision
  - Possibly overflow if exponent too large
  - Possibly *round* to fit into `frac`

Rounding Modes (illustrate with $ rounding)

	$1.40	$1.60	$1.50	$2.50	–$1.50
Zero	$1	$1	$1	$2	–$1
Round down (-∞)	$1	$1	$1	$2	–$2
Round up (+∞)	$2	$2	$2	$3	–$1
Nearest Even (default)	$1	$2	$2	$2	–$2

Note:

1. Round down: rounded result is close to but no greater than true result.
2. Round up: rounded result is close to but no less than true result.

Closer Look at Round-To-Even

Default Rounding Mode

- Hard to get any other kind without dropping into assembly
- All others are statistically biased
  - Sum of set of positive numbers will consistently be over- or under-estimated

Applying to Other Decimal Places / Bit Positions

- When exactly halfway between two possible values
  - Round so that least significant digit is even
- E.g., round to nearest hundredth

1.2349999	1.23	(Less than half way)
1.2350001	1.24	(Greater than half way)
1.2350000	1.24	(Half way—round up)
1.2450000	1.24	(Half way—round down)

Rounding Binary Numbers

Binary Fractional Numbers

- "Even" when least significant bit is 0
- Half way when bits to right of rounding position = $100\ldots_2$

Examples

- Round to nearest 1/4 (2 bits right of binary point)

Value	Binary	Rounded	Action	Rounded Value
2 3/32	10.00011_2	10.00_2	(<1/2—down)	2
2 3/16	10.00110_2	10.01_2	(>1/2—up)	2 1/4
2 7/8	10.11100_2	11.00_2	(1/2—up)	3
2 5/8	10.10100_2	10.10_2	(1/2—down)	2 1/2

FP Multiplication

Operands

$(-1)^{s1}\ M1\ \ 2^{E1}$ * $(-1)^{s2}\ M2\ \ 2^{E2}$

Exact Result

$(-1)^{s}\ M\ \ 2^{E}$

- **Sign** s**:** $s1$ ^ $s2$
- **Significand** M**:** $M1 * M2$
- **Exponent** E**:** $E1 + E2$

Fixing

- **If** $M \geq 2$**, shift** M **right, increment** E
- **If** E **out of range, overflow**
- **Round** M **to fit** `frac` **precision**

Implementation

- **Biggest chore is multiplying significands**

FP Multiplication - 8 bit example

- Recall: 1 bit for sign, 4 bits for exponent `exp`, 3 bits for fraction `frac`
- 9/512 * 224
- 9/512 = 0.000001001 = $1.001 * 2^{-6}$
- 224 = 11100000. = $1.110 * 2^{7}$

$9/512 * 224 = (1.001 * 1.110) * 2^{(-6+7)}$

$= (1.001 * 1.110) * 2^{1}$

$= (1.001 * (2^{0} + 2^{-1} + 2^{-2})) * 2^{1}$

$= (1.001 + 1.001>>1 + 1.001>>2) * 2^{1}$

$= 1.111110 * 2^{1}$

$= 1.111 * 2^{1}$ (only 3 bits for `frac`, so Round Zero)

= 15/8 * 2 = 15/4 = 3.75

The "precise" answer is ~3.94

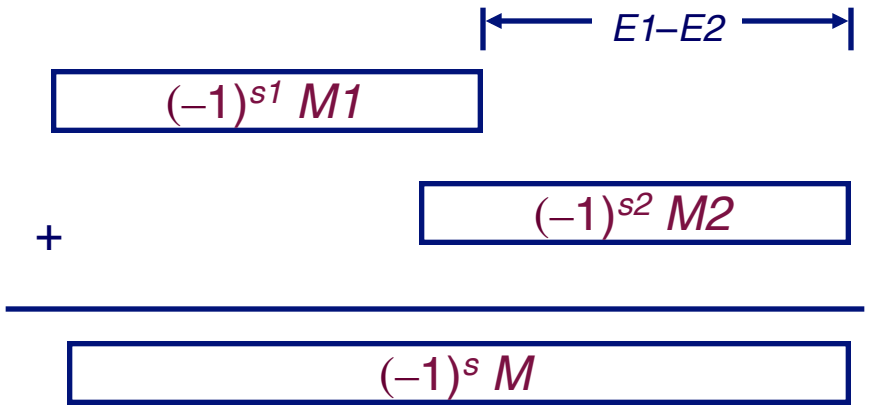
FP Addition

Operands

$(-1)^{s1}\ M1\ \ 2^{E1}$

$(-1)^{s2}\ M2\ \ 2^{E2}$

- **Assume $E1 > E2$**



Exact Result

$(-1)^{s}\ M\ \ 2^{E}$

- **Sign s, significand M:**
  - **Result of signed align & add**
- **Exponent E:** $E1$

Fixing

- **If $M \geq 2$, shift M right, increment E**
- **if $M < 1$, shift M left k positions, decrement E by k**
- **Overflow if E out of range**
- **Round M to fit `frac` precision**

FP Addition - 8 bit example

- 9/512 + 224
- 9/512 = 1.001 * 2^{-6} = 0.000001001
- 224 = 1.110 * 2^{7} = 11100000.
- 9/512+224 = 11100000.000001001

  = 1.1100000000001001 * 2^{7}

  = 1.110 * 2^{7} (3 bits `frac`)

  = 224 (9/512 is rounded away!)

Implication of this rounding effect:

- Suppose you added 9/512 50,000 times to 224:
  Then 9/512 + 9/512 + … + 9/512 + 224 > 224

  But 9/512 + (9/512+(…+(9/512+224))))))) = 224 !!!

  So floating point addition is not associative!

Supplementary Slides

Floating Point Puzzles - practice

- For each of the following C expressions, either:
  - Argue that it is true for all argument values
  - Explain why not true

```
int x = …;

float f = …;

double d = …;
```

Assume neither `d` nor `f` is NaN

- `x == (int)(float) x`
- `x == (int)(double) x`
- `f == (float)(double) f`
- `d == (float) d`
- `f == -(-f);`
- `2/3 == 2/3.0`
- `d < 0.0  ⇒  ((d*2) < 0.0)`
- `d > f  ⇒  -f > -d`
- `d * d >= 0.0`
- `(d+f)-d == f`

Answers to Floating Point Puzzles

```
int x = …;

float f = …;

double d = …;
```

Assume neither `d` nor `f` is NAN

• `x == (int)(float) x`	No: 24 bit significand
• `x == (int)(double) x`	Yes: 53 bit significand
• `f == (float)(double) f`	Yes: increases precision
• `d == (float) d`	No: loses precision
• `f == -(-f);`	Yes: Just change sign bit
• `2/3 == 2/3.0`	No: 2/3 == 0
• `d < 0.0 ⇒ ((d*2) < 0.0)`	Yes!
• `d > f ⇒ -f > -d`	Yes!
• `d * d >= 0.0`	Yes!
• `(d+f)-d == f`	No: Not associative

Interesting Numbers

Description	exp	frac	Numeric Value
Zero	00…00	00…00	0.0
Smallest Pos. Denorm.	00…00	00…01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
■ Single ≈ 1.4×10^{-45}			
■ Double ≈ 4.9×10^{-324}			
Largest Denormalized	00…00	11…11	$(1.0 - \varepsilon) \times 2^{-\{126,1022\}}$
■ Single ≈ 1.18×10^{-38}			
■ Double ≈ 2.2×10^{-308}			
Smallest Pos. Normalized	00…01	00…00	$1.0 \times 2^{-\{126,1022\}}$
■ Just larger than largest denormalized			
One	01…11	00…00	1.0
Largest Normalized	11…10	11…11	$(2.0 - \varepsilon) \times 2^{\{127,1023\}}$
■ Single ≈ 3.4×10^{38}			
■ Double ≈ 1.8×10^{308}			

Special Properties of Encoding

FP Zero Same as Integer Zero

- All bits = 0

Can (Almost) Use Unsigned Integer Comparison

- Must first compare sign bits
- Must consider -0 = 0
- NaNs problematic
  - Will be greater than any other values
  - What should comparison yield?
- Otherwise OK
  - Denorm vs. normalized
  - Normalized vs. infinity