IEEE Floating Point

- **IEEE Standard 754**
  - Established in 1985 as uniform standard for floating point arithmetic
    - Before that, many idiosyncratic formats
  - Supported by all major CPUs

- **Driven by numerical concerns**
  - Nice standards for rounding, overflow, underflow
  - Hard to make fast in hardware
    - Numerical analysts predominated over hardware designers in defining standard

Floating Point Representation

- **Numerical Form:**

$$(-1)^s \, M \; 2^E$$

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

- **Encoding**
  - MSB s is sign bit ***s***
  - **exp** field encodes ***E*** (but is not equal to E)
  - **frac** field encodes ***M*** (but is not equal to M)

s	exp	frac

Precision options

- **Single precision: 32 bits**

s	exp	frac
1	8-bits	23-bits

- **Double precision: 64 bits**

s	exp	frac
1	11-bits	52-bits

- **Extended precision: 80 bits (Intel only)**

s	exp	frac
1	15-bits	63 or 64-bits

"Normalized" Values

$v = (-1)^s \, M \, 2^E$

- **When: exp ≠ 000…0 and exp ≠ 111…1**

- **Exponent coded as a *biased* value: $E = Exp - Bias$**
  - *Exp*: unsigned value of exp field
  - *Bias* = $2^{k-1} - 1$, where *k* is number of exponent bits
    - Single precision: 127 (Exp: 1…254, E: -126…127)
    - Double precision: 1023 (Exp: 1…2046, E: -1022…1023)

- **Significand coded with implied leading 1: $M = 1.xxx…x_2$**
  - xxx…x: bits of frac field
  - Minimum when frac=000…0 (M = 1.0)
  - Maximum when frac=111…1 (M = 2.0 – ε)
  - Get extra leading bit for "free"

Normalized Encoding Example

$v = (-1)^s M\, 2^E$
$E = Exp - Bias$

- **Value:** `float F = 15213.0;`
  - $15213_{10} = 11101101101101_2$
    $= 1.1101101101101_2 \times 2^{13}$

- **Significand**

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

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

- **Exponent**

  $E = 13$

  $Bias = 127$

  $Exp = 140 = 10001100_2$

- **Result:**

s	exp	frac
0	10001100	11011011011010000000000

Denormalized Values

$v = (-1)^s M\, 2^E$
$E = 1 - Bias$

- **Condition:** exp = 000…0

- **Exponent value: *E* = 1 – Bias (instead of *E* = 0 – *Bias*)**
- **Significand coded with implied leading 0: *M* = 0.xxx…x$_2$**
  - **`xxx`…`x`**: bits of **`frac`**
- **Cases**
  - **`exp`** = 000…0, **`frac`** = 000…0
    - Represents zero value
    - Note distinct values: +0 and –0 (why?)
  - **`exp`** = 000…0, **`frac`** ≠ 000…0
    - Numbers closest to 0.0
    - Equispaced

Special Values

- **Condition: `exp` = `111`…`1`**

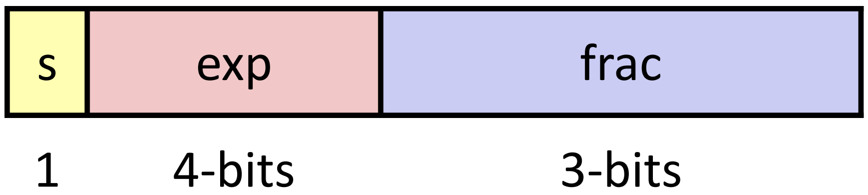
- **Case: `exp` = `111`…`1`, `frac` = `000`…`0`**
  - Represents value ∞ (infinity)
  - Operation that overflows
  - Both positive and negative
  - E.g., $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$

- **Case: `exp` = `111`…`1`, `frac` ≠ `000`…`0`**
  - Not-a-Number (NaN)
  - Represents case when no numeric value can be determined
  - E.g., sqrt(–1), $\infty - \infty$, $\infty \times 0$

Today: Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- **Example and properties**
- Rounding, addition, multiplication
- Floating point in C
- Summary

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, with a bias of 7
  - the last three bits are the `frac`

- **Same general form as IEEE Format**
  - normalized, denormalized
  - representation of 0, NaN, infinity

Dynamic Range (Positive Only)

$v = (-1)^s\, M\, 2^E$

n: E = Exp – Bias

d: E = 1 – Bias

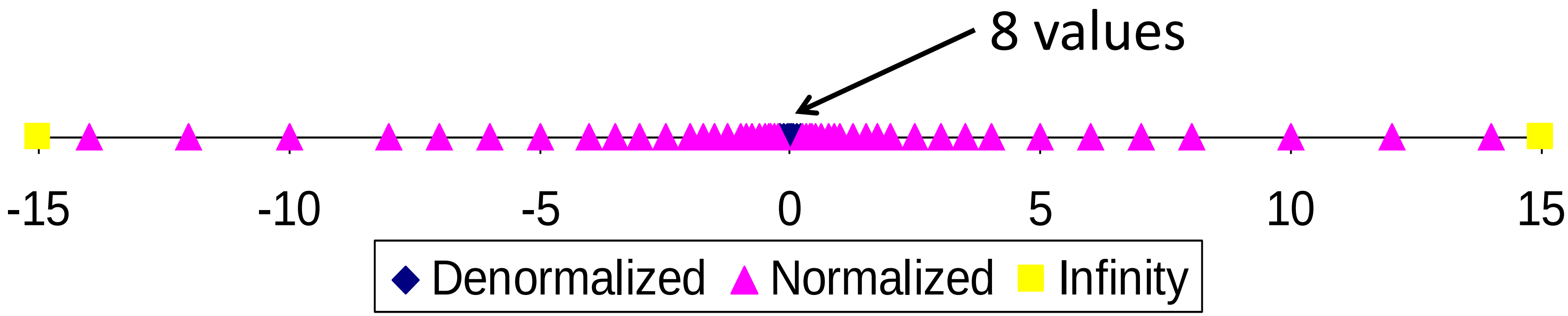
	s	exp	frac	E	Value	
Denormalized numbers	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	closest to zero
	0	0000	010	-6	2/8*1/64 = 2/512	
	…					
	0	0000	110	-6	6/8*1/64 = 6/512	
	0	0000	111	-6	7/8*1/64 = 7/512	largest denorm
Normalized numbers	0	0001	000	-6	8/8*1/64 = 8/512	smallest norm
	0	0001	001	-6	9/8*1/64 = 9/512	
	…					
	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
	0	1111	000	n/a	inf	

Distribution of Values

- **6-bit IEEE-like format**
  - e = 3 exponent bits
  - f = 2 fraction bits
  - Bias is $2^{3-1}-1 = 3$

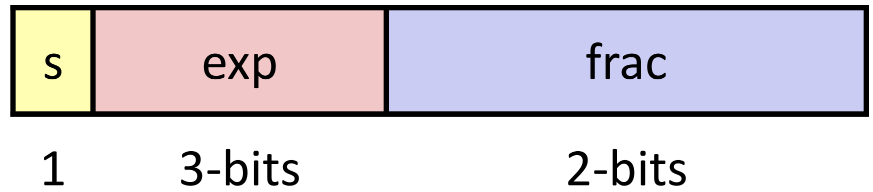
s	exp	frac
1	3-bits	2-bits

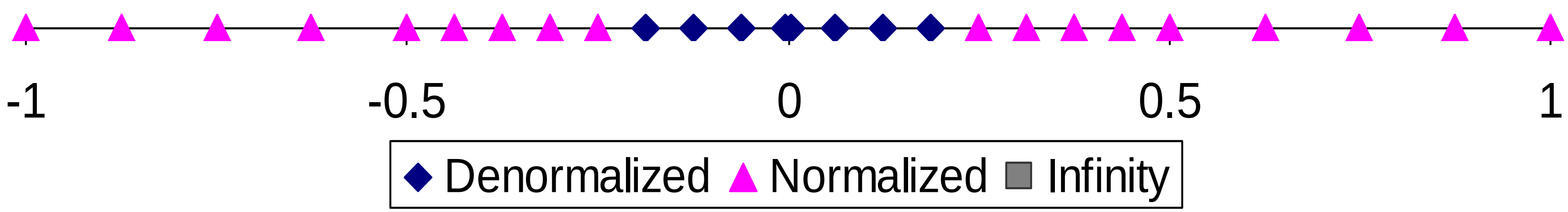
- **Notice how the distribution gets denser toward zero.**



Distribution of Values (close-up view)

- **6-bit IEEE-like format**
  - e = 3 exponent bits
  - f = 2 fraction bits
  - Bias is 3





Special Properties of the IEEE 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

Rounding

`1.BBGRXXX`

Guard bit: LSB of result

Sticky bit: OR of remaining bits

Round bit: 1st bit removed

- **Round up conditions**
  - Round = 1, Sticky = 1 → > 0.5
  - Guard = 1, Round = 1, Sticky = 0 → Round to even

Fraction	*GRS*	*Incr?*	*Rounded*
`1.0000000`	`000`	**N**	`1.000`
`1.1010000`	`100`	**N**	`1.101`
`1.0001000`	`010`	**N**	`1.000`
`1.0011000`	`110`	**Y**	`1.010`
`1.0001010`	`011`	**Y**	`1.001`
`1.1111100`	`111`	**Y**	`10.000`

FP Multiplication

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ \times\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- **Exact Result:** $(-1)^{s}\ M\ \ 2^{E}$
  - Sign s: $s1$ ^ $s2$
  - Significand M: $M1 \times M2$
  - Exponent E: $E1 + E2$

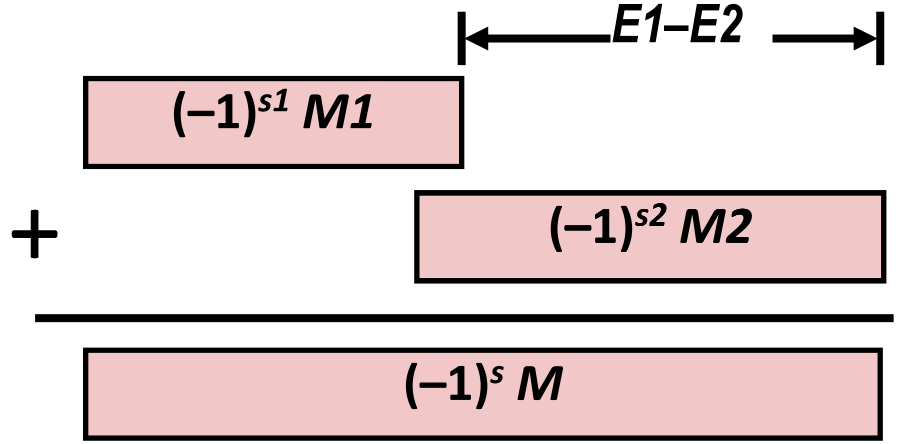
- **Fixing**
  - If $M \geq 2$, shift M right, increment E
  - If E out of range, overflow
  - Round M to fit `frac` precision

- **Implementation**
  - Biggest chore is multiplying significands

Floating Point Addition

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

Get binary points lined up



- **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

Today: Floating Point

- Background: Fractional binary numbers
- IEEE floating point standard: Definition
- Example and properties
- Rounding, addition, multiplication
- **Floating point in C**
- Summary

Floating Point in C

- **C Guarantees Two Levels**
  - **`float`** single precision
  - **`double`** double precision
- **Conversions/Casting**
  - Casting between **`int`**, **`float`**, and **`double`** changes bit representation
  - **`double`/`float` → `int`**
    - Truncates fractional part
    - Like rounding toward zero
    - Not defined when out of range or NaN: Generally sets to TMin
  - **`int` → `double`**
    - Exact conversion, as long as **`int`** has ≤ 53 bit word size
  - **`int` → `float`**
    - Will round according to rounding mode