Data Representation – Floating Point

CSCI 224 / ECE 317: Computer Architecture

Instructor:

Prof. Jason Fritts

Slides adapted from Bryant & O'Hallaron's slides

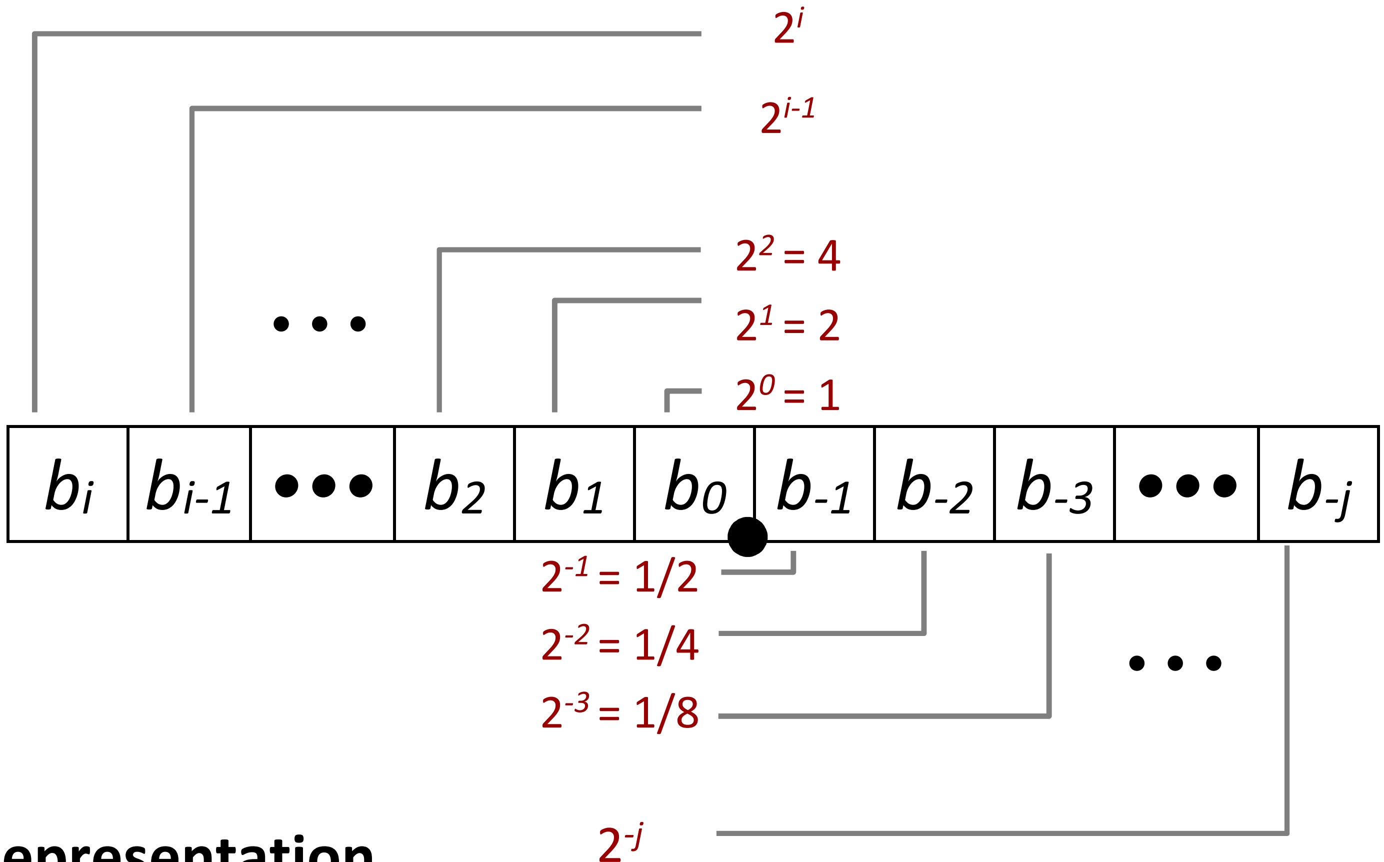
Today: Floating Point

- Background: Fractional binary numbers
- Example and properties
- IEEE floating point standard: Definition
- Floating point in C
- Summary

Fractional binary numbers

- **What is 1011.101_2?**

Fractional Binary Numbers



- **Representation**
  - Bits to right of "binary point" represent fractional powers of 2
  - Represents rational number:

$$\sum_{k=-j}^{i} b_k \times 2^k$$

Fractional Binary Numbers: Examples

- **Value** — **Representation**

Value	Representation	
$5\,{}^3/_4$	101.11_2	$= 4 + 1 + {}^1/_2 + {}^1/_4 = 5\,{}^3/_4$
$2\,{}^7/_8$	10.111_2	$= 2 + {}^1/_2 + {}^1/_4 + {}^1/_8 = 2\,{}^7/_8$
${}^{25}/_{64}$	0.011001_2	$= {}^1/_4 + {}^1/_8 + {}^1/_{64} = {}^{25}/_{64}$

- **Observations**
  - Divide by 2 by shifting right
  - Multiply by 2 by shifting left

- **Limitations**
  - Can only exactly represent numbers of the form $x/2^k$
  - Other rational numbers have repeating bit representations

Value	Representation
1/3	0.0101010101**[01]**…$_2$
1/5	0.001100110011**[0011]**…$_2$

Today: Floating Point

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

Tiny Floating Point Example

s	exp	frac
1	4-bits	3-bits

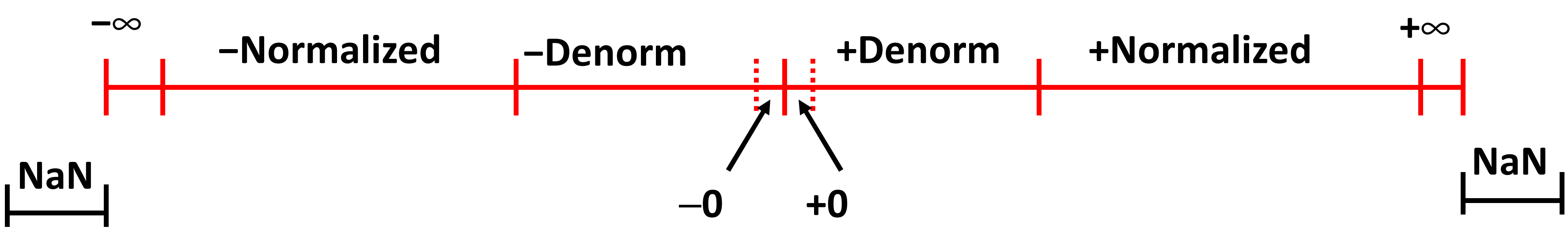
- **8-bit Floating Point Representation**
  - the sign bit is in the most significant bit
  - the next four bits are the exponent **`(exp)`**, with a bias of $2^{4-1} - 1 = 7$
  - the last three bits are the fraction **`(frac)`**

- **Exponent bias**
  - enable exponent to represent both positive and negative powers of 2
  - use half of range for positive and half for negative power
  - given k exponent bits, bias is then $2^{k-1} - 1$

Floating Point Encodings and Visualization

- **Five encodings:**
  - Two general forms: normalized, denormalized
  - Three special values: zero, infinity, NaN *(not a number)*

Name	Exponent (`exp`)	Fraction (`frac`)
zero	`exp` == 0000	`frac` == 000
denormalized	`exp` == 0000	`frac` != 000
normalized	0000 < `exp` < 1111	`frac` != 000
infinity	`exp` == 1111	`frac` == 000
NaN	`exp` == 1111	`frac` != 000



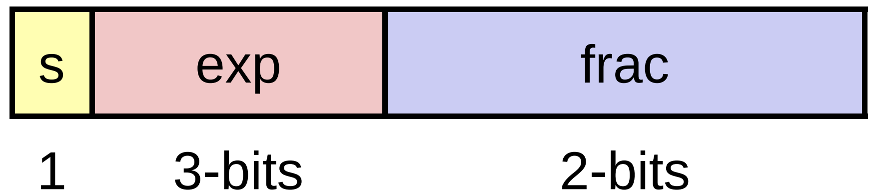
Dynamic Range (Positive Only)

	s	exp	frac	E	Value	
Denormalized numbers	0	0000	000	-6	0	*zero*
	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	*infinity*
	0	1111	xxx	n/a	NaN	*NaN* (not a number)

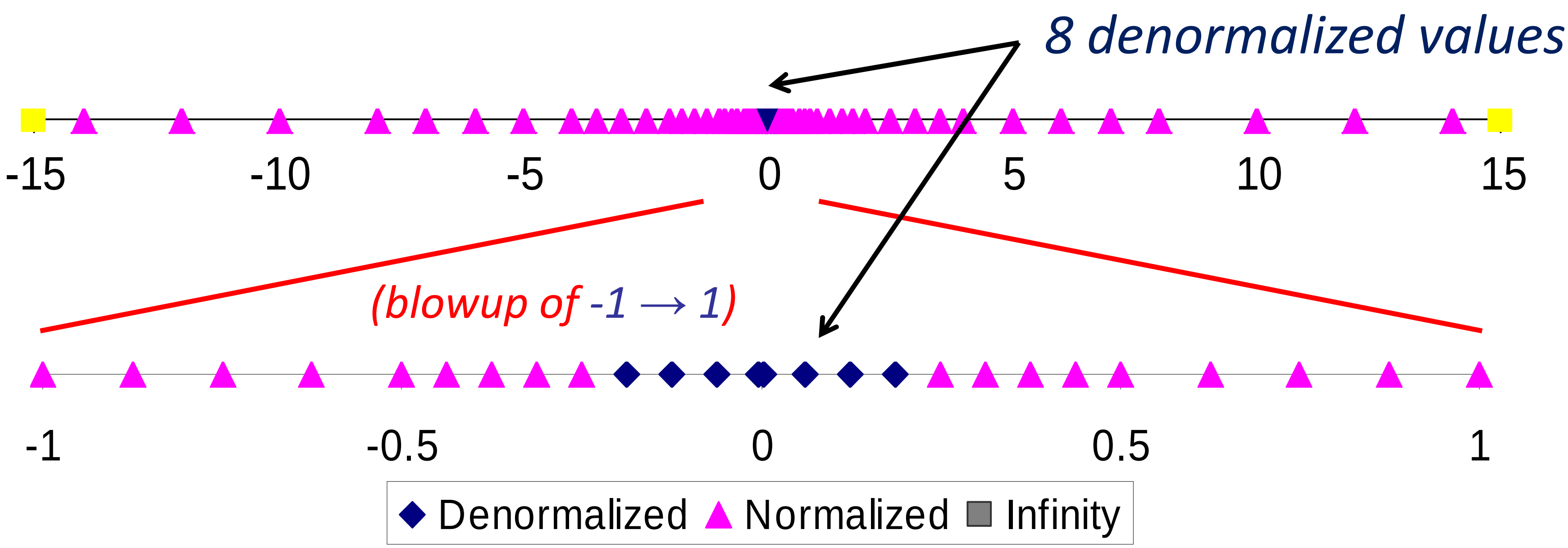
Distribution of Values

(reduced format from 8 bits to 6 bits for visualization)

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



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



Today: Floating Point

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

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

Precisions

- **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

- **Condition:** *exp* **≠ 000…0 and** *exp* **≠ 111…1**
- **Exponent coded as *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* = $\underline{1}.\text{xxx}\ldots\text{x}_2$**
  - xxx…x: bits of *frac*
- **Decimal value of normalized FP representations:**
  - Single-precision: $Value_{10} = (-1)^s \times 1.frac \times 2^{exp-127}$
  - Double-precision: $Value_{10} = (-1)^s \times 1.frac \times 2^{exp-1023}$

Normalized Encoding Example

- **Value:** **`float F = 15213.0;`**
  - $15213_{10} = 11101101101101_2$

    $= 1.1101101101101_2 \times 2^{13}$

shift binary point by K bits so that only one leading 1 bit remains on the left side of the binary point (here, shifted right by 13 bits, so K = 13), then multiply by 2^K (here, 2^{13})

- **Significand**

  M = **1.1101101101101**$_2$

  frac = **11011011011010000000000**$_2$

- **Exponent (*E* = *Exp – Bias*)**

  E = 13

  $Bias$ = 127

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

0	10001100	11011011011010000000000
s	exp	frac

Denormalized Values

- **Condition:** exp = 000…0
- **Exponent value:** $E = –Bias + 1$ **(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 very close to 0.0
    - Lose precision as get smaller
    - 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), $\infty - \infty$, $\infty \times 0$

Interesting Numbers {single,double}

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}			

Today: Floating Point

- Background: Fractional binary numbers
- Example and properties
- IEEE floating point standard: Definition
- **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

Today: Floating Point

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

Summary

- **Represents numbers of form M x 2^E**

- **One can reason about operations independent of implementation**
  - As if computed with perfect precision and then rounded

- **Not the same as real arithmetic**
  - Violates associativity/distributivity
  - Makes life difficult for compilers & serious numerical applications programmers