Floating point

Today

- IEEE Floating Point Standard
- Rounding
- Floating Point Operations
- Mathematical properties

Next time

- The machine model

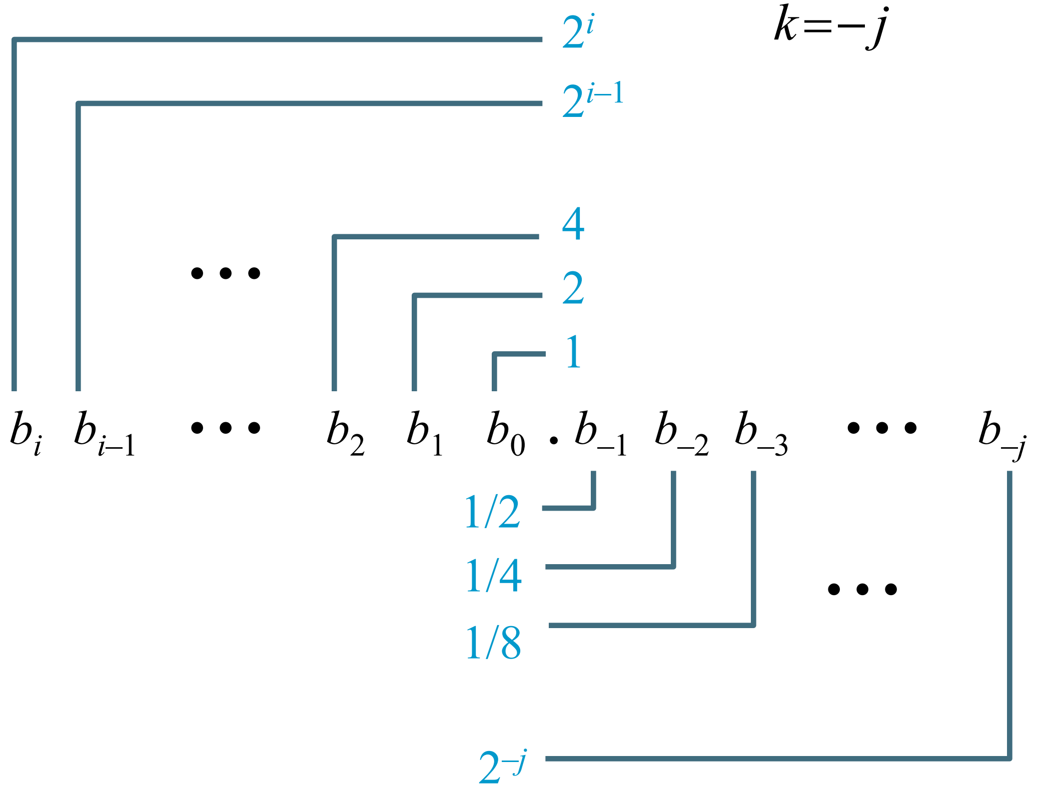
IEEE Floating point

- Floating point representations
  - Encodes rational numbers of the form $V=x*(2^y)$
  - Useful for very large numbers or numbers close to zero
- IEEE Standard 754 (*IEEE floating point*)
  - Established in 1985 as uniform standard for floating point arithmetic (started as an Intel's sponsored effort)
    - Before that, many idiosyncratic formats
  - Supported by all major CPUs
- Driven by numerical concerns
  - Nice standards for rounding, overflow, underflow
  - Hard to make go fast
    - Numerical analysts predominated over hardware types in defining standard

Fractional binary numbers

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

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



Fractional binary number examples

- Value Representation
  - ???? 101.11_2
  - ???? 10.111_2
  - ???? 0.111111_2
- Observations
  - Divide by 2 by shifting right (the point moves to the left)
  - Multiply by 2 by shifting left (the point moves to the right)
  - Numbers of form $0.111111\ldots_2$ represent those just below 1.0
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - We use notation $1.0 - \varepsilon$ to represent them

Representable numbers

- Limitation
  - Can only exactly represent numbers of the form $x/2^k$
  - Other numbers have repeating bit representations
- Value Representation
  - 1/3 $0.0101010101[01]\ldots_2$
  - 1/5 $0.001100110011[0011]\ldots_2$
  - 1/10 $0.0001100110011[0011]\ldots_2$

Floating point representation

- Numerical form
  - $V = (-1)^s * M * 2^E$
    - 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 is sign bit
  - `exp` field encodes E (note: *encode != is*)
  - `frac` field encodes M

Floating point precisions

- Encoding

s	exp	frac

  – Sign bit; `exp` (encodes E): k-bit; `frac` (encodes M): n-bit

- Sizes
  – Single precision: k = 8 exp bits, n= 23 frac bits
    - 32 bits total
  – Double precision: k = 11 exp bits, n = 52 frac bits
    - 64 bits total
  – Extended precision: k = 15 exp bits, n = 63 frac bits
    - Only found in Intel-compatible machines
    - Stored in 80 bits
      – 1 bit wasted
- Value encoded – three different cases, depending on value of `exp`

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 = $2^{k-1} - 1$, where k is number of exponent bits
- Significand coded with implied leading 1
  - M = $1.xxx\ldots x_2$ ($1+f$ & $f = 0.xxx_2$)
    - 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”

Normalized encoding example

- Value
  - Float F = 15213.0;
  - $15213_{10} = 11101101101101_2 = 1.1101101101101_2 \text{ X } 2^{13}$
- Significand
  - $M = 1.1101101101101_2$
  - frac = 11011011011010000000000
- Exponent
  - E = 13
  - Bias = 127
  - $\text{exp} = E + \text{Bias} = 140 = 10001100_2$

Floating Point Representation:

Hex:	4	6	6	D	B	4	0	0
Binary:	0100	0110	0110	1101	1011	0100	0000	0000
140:	100	0110	0					
15213:			110	1101	1011	01		

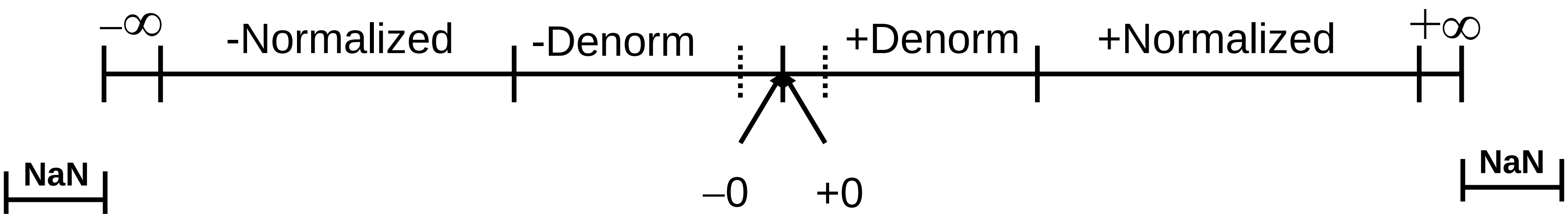
Denormalized values

- Condition
  - $\text{exp} = 000\ldots0$
- Value
  - Exponent value $E = 1 - \text{Bias}$
    - Note: not simply $E = -\text{Bias}$
  - Significand value $M = 0.\text{xxx}\ldots\text{x}_2$ ($0.f$)
    - xxx…x: bits of frac
- Cases
  - $\text{exp} = 000\ldots0$, $\text{frac} = 000\ldots0$
    - Represents value 0
    - Note that have distinct values +0 and –0
  - $\text{exp} = 000\ldots0$, $\text{frac} \neq 000\ldots0$
    - Numbers very close to 0.0

Special values

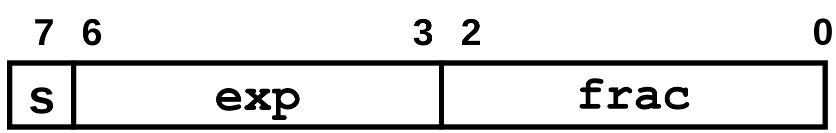
- Condition
  - exp = 111…1
- Cases
  - exp = 111…1, frac = 000…0
    - Represents value ∞(infinity)
    - Operation that overflows
    - Both positive and negative
    - E.g., $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$
  - exp = 111…1, frac $\neq$ 000…0
    - Not-a-Number (NaN)
    - Represents case when no numeric value can be determined
    - E.g., $\sqrt{-1}$, $(\infty - \infty)$

Summary of FP real number encodings



Tiny floating point example

- 8-bit Floating Point Representation
  - the sign bit is in the most significant bit.
  - the next four (k) bits are the exponent, with a bias of 7 ($2^{k-1}-1$)
  - the last three (n) bits are the frac
- Same General Form as IEEE Format
  - normalized, denormalized
  - representation of 0, NaN, infinity



Values related to the exponent

Normalized
$E = e - Bias$

Denormalized
$E = 1 - Bias$

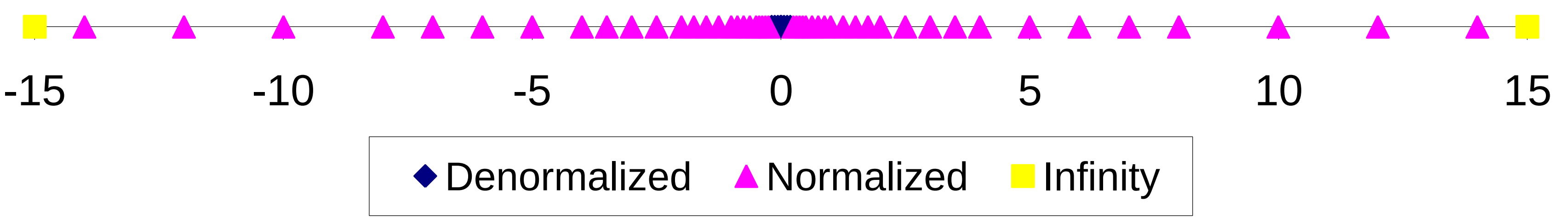
Exp	exp	E	2^E	
0	0000	-6	1/64	(denorms)
1	0001	-6	1/64	
2	0010	-5	1/32	
3	0011	-4	1/16	
4	0100	-3	1/8	
5	0101	-2	1/4	
6	0110	-1	1/2	
7	0111	0	1	
8	1000	+1	2	
9	1001	+2	4	
10	1010	+3	8	
11	1011	+4	16	
12	1100	+5	32	
13	1101	+6	64	
14	1110	+7	128	
15	1111	n/a		(inf, NaN)

Dynamic range

	s	exp	frac	*E*	Value	
	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	closest to zero
Denormalized numbers	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	
	…					
	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	
Normalized numbers	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	

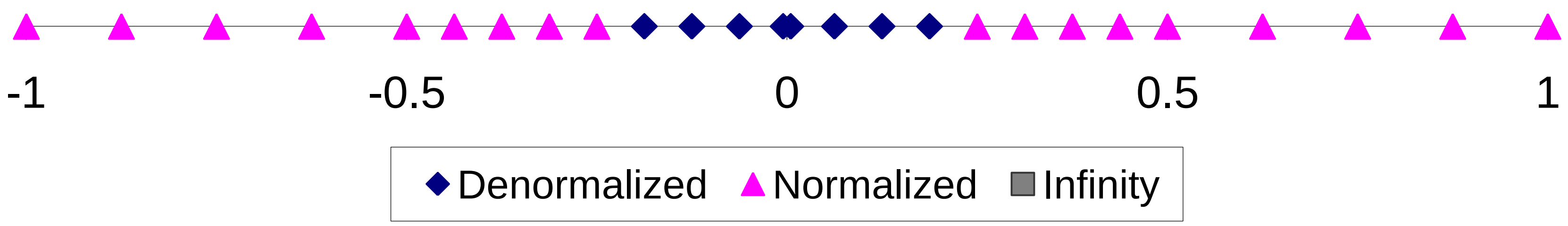
Distribution of values

- 6-bit IEEE-like format
  - e = 3 exponent bits
  - f = 2 fraction bits
  - Bias is 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
- Note: Smooth transition between normalized and de-normalized numbers due to definition E = 1 - Bias for denormalized values



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}

Description	exp	frac	Numeric Value
Largest Denormalized	00…00	11…11	$(1.0 - \varepsilon) \times 2^{-\{126,1022\}}$

- Single ~ 1.18×10^{-38}
- Double ~ 2.2×10^{-308}

Description	exp	frac	Numeric Value
Smallest Pos. Normalized	00…01	00…00	$1.0 \times 2^{-\{126,1022\}}$

- Just larger than largest denormalized

Description	exp	frac	Numeric Value
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}

Floating point operations

- Conceptual view
  - 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
  - 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)

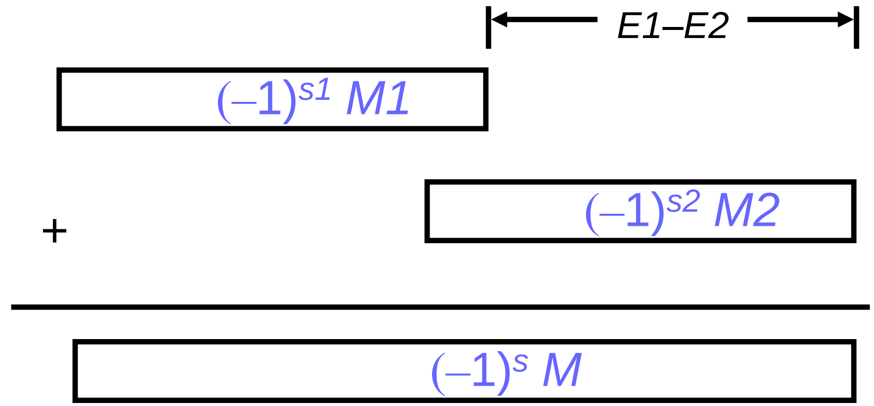
Value	Binary	Rounded	Action	Rounded Value
2 3/32	10.00011_2	10.00_2	(<1/2—down)	2
2 3/16	10.00110_2	10.01_2	(>1/2—up)	2 1/4
2 7/8	10.11100_2	11.00_2	(1/2—up)	3
2 5/8	10.10100_2	10.10_2	(1/2—down)	2 1/2

FP multiplication

- Operands
  - $(-1)^{s1}\ M^{1}\ \ 2^{E1}$ * $(-1)^{s2}\ M^{2}\ \ 2^{E2}$
- Exact result
  - $(-1)^{s}\ M\ \ 2^{E}$
  - Sign s: s1 ^ s2
  - Significand M: M1 * M2
  - Exponent E: E1 + E2
- Fixing
  - If M ≥ 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 addition

- Operands
  – (–1)s1 M1 2E1
  – (–1)s2 M2 2E2
  – Assume E1 > E2



- Exact Result
  – (–1)s M 2E
  – Sign s, significand M:
    • Result of signed align & add
  – Exponent E: E1
- Fixing
  – If M ≥ 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

Mathematical properties of FP add

- Compare to those of Abelian Group
  - Closed under addition? YES
    - But may generate infinity or NaN
  - Commutative? YES
  - Associative? NO
    - Overflow and inexactness of rounding
      - (3.14+1e10)-1e10=0 (rounding)
      - 3.14+(1e10-1e10)=3.14
  - 0 is additive identity? YES
  - Every element has additive inverse ALMOST
    - Except for infinities & NaNs
- Monotonicity
  - $a \geq b \Rightarrow a+c \geq b+c$? ALMOST
    - Except for NaNs

Math. properties of FP multiplication

- Compare to commutative ring
  - Closed under multiplication? YES
    - But may generate infinity or NaN
  - Multiplication Commutative? YES
  - Multiplication is Associative? NO
    - Possibility of overflow, inexactness of rounding
  - 1 is multiplicative identity? YES
  - Multiplication distributes over addition? NO
    - Possibility of overflow, inexactness of rounding
- Monotonicity
  - $a \geq b$ & $c \geq 0 \Rightarrow a * c \geq b * c$? ALMOST
    - Except for NaNs

Floating point in C

- C guarantees two levels
  - float   single precision
  - double double precision
- Conversions
  - int → float : maybe rounded
  - int → double : exact value preserved (double has greater range and higher precision)
  - float → double : exact value preserved (double has greater range and higher precision)
  - double → float : may overflow or be rounded
  - double → int : truncated toward zero (-1.999 → -1)
  - float → int : truncated toward zero

Floating point puzzles

- 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)(double) x`	*Yes*
`x == (int)(float) x`	*No (x = TMax)*
`d == (double)(float) d`	*No (d = 1e40)*
`f == (float)(double) f`	*Yes*
`f == -(-f);`	*Yes*
`1.0/2 == 1/2.0`	*Yes*
`d*d >= 0.0`	*Yes (may overflow)*
`(f+d)-f == d`	*No (f = 1.0e20, d = 1.0; f+d rounded to 1.0e20*

Summary

- IEEE Floating point has clear mathematical properties
  - Represents numbers of form M X 2E
  - Not the same as real arithmetic
    - Violates associativity/distributivity
    - Makes life difficult for compilers & serious numerical applications programmers