[Chap.2-3] Representing and Manipulating Information

Young Ik Eom (yieom@skku.edu, 031-290-7120)
Distributing Computing Laboratory
Sungkyunkwan University
http://dclab.skku.ac.kr

Contents

- Introduction
- Information storage
- Integer representations
- Integer arithmetic
- Floating point
- Summary

Floating Point

The problem

- How to represent fractional values with finite # of bits?
  - 0.1
  - 2.718281828...
  - 3.14159265358979323846264338327950288...

- How to represent very small (close to 0) numbers ($|V| \ll 1$) and very large numbers ($|V| \gg 0$)
  - 0.0000000000000000000000000000000000000000000...01
  - 0.123×10^{-60}
  - 1.234×10^{90}

Floating Point

The problem

- 3 fields for representation of FP numbers
  - Sign
  - Exponent
  - Fraction (significand, mantissa)

s	exp	frac

Floating Point

Warming up

- Example-1)
  - $0.1_{(10)} = 0.00011001100110011[0011]_{(2)}$
  - $0.110011001100\ldots \times 2^{-3}$ ($\Leftrightarrow 0.00011001100\ldots \times 2^{0}$)
  - (s, exp, frac) with 16-bits FPN = 0 <u>11101</u> 1100110011

Excess-16 exp: 11101 + 10000 = 01101

Normalization

- Example-2)
  - $5.1_{(10)} = 101.00011001100110011[0011]_{(2)}$
  - $0.10100011001100110011\ldots \times 2^{3}$
  - (s, exp, frac) with 16-bits FPN = 0 00011 1010001100

Excess-16 exp: 00011 + 10000 = 10011

Floating Point

Warming up

- Example-3) 16-bit integer 1
  - $1_{(10)} = 0000000000000001_{(2)}$

- Example-4) 16-bit float 1.0
  - $1.0_{(10)} = 1.000000000000000_{(2)}$
  - $0.1000000000000000\ldots \times 2^1$
  - (s, exp, frac) with 16-bits FPN = 0000011000000000

Excess-16 exp: 00001 + 10000 = 10001

Floating Point

Fractional decimal numbers

- Representation
  - $d_m d_{m-1} \cdots d_1 d_0 . d_{-1} d_{-2} \cdots d_{-(n-1)} d_{-n}$
- Value
  - $d = \sum_{i=-n}^{m} 10^i \times d_i$
- Example) 12.34
  - $1\times10^1 + 2\times10^0 + 3\times10^{-1} + 4\times10^{-2} = 12\frac{34}{100}$

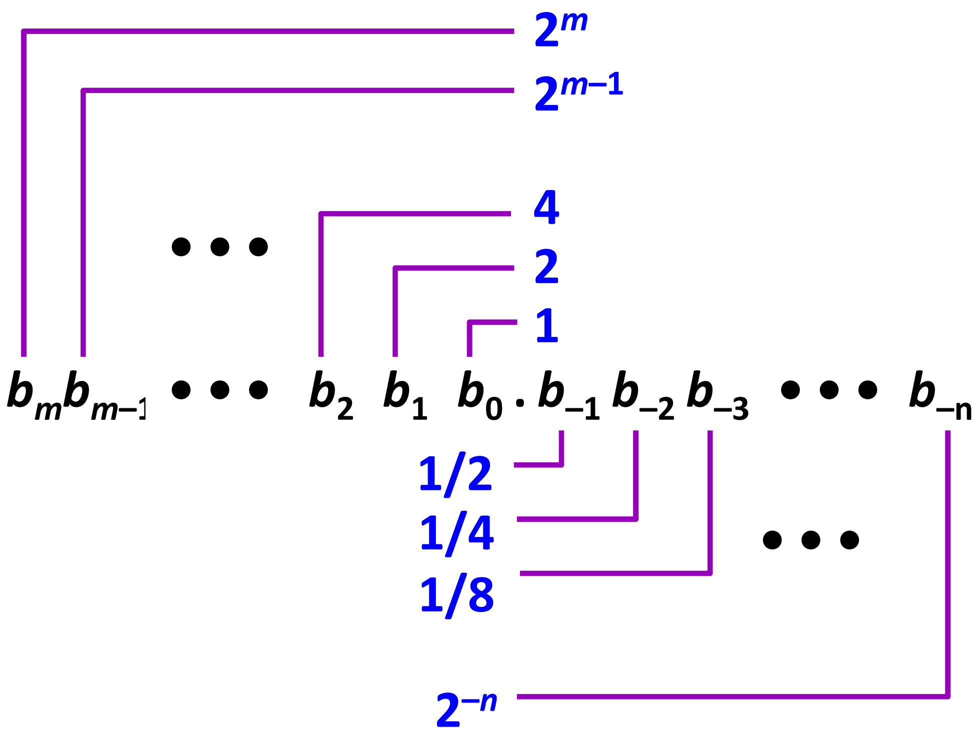
Floating Point

Fractional binary numbers

- Representation
  - $b_m b_{m-1} \cdots b_1 b_0 . b_{-1} b_{-2} \cdots b_{-(n-1)} b_{-n}$
- Value
  - $b = \sum_{i=-n}^{m} 2^i \times b_i$
- Example)
  - $101.11_2 = 1\times2^2 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} = 5\frac{3}{4}$

Floating Point

■ Fractional binary numbers



$$b = \sum_{i=-n}^{m} b_i \cdot 2^i$$

Floating Point

Examples)

Value	Representation
$5\frac{3}{4}$	**101.11_2**
$2\frac{7}{8}$	**10.111_2**
$\frac{63}{64}$	**0.111111_2**

Observations

- Divide by 2 by shifting right
- Multiply by 2 by shifting left
- Numbers of form **$0.111111\ldots_2$** just below **1.0**
  - $1/2 + 1/4 + 1/8 + \cdots + 1/2^i + \cdots \rightarrow 1.0$
  - Use notation **1.0** – ε

Floating Point

Representable numbers

- Can only exactly represent numbers of the form $\mathbf{x/2^k}$
- Other numbers have repeating bit representations

Value	Representation
1/3	$0.0101010101[01]\cdots_2$
1/5	$0.001100110011[0011]\cdots_2$
1/10	$0.0001100110011[0011]\cdots_2$

Floating Point

IEEE FP representation

- Encodes rational numbers of the form $\mathbf{V} = \mathbf{x} * \mathbf{2^y}$
- Useful for very large numbers ($|\mathbf{V}| \gg \mathbf{0}$) or numbers very close to 0 ($|\mathbf{V}| \ll \mathbf{1}$)

- IEEE Standard 754 (1985-)
  - Standard for representing floating-point numbers
  - Sponsored by Intel and IEEE
  - William Kahan (1933-)
    - ✓ The father of floating point
    - ✓ Emeritus professor of EECS at the Univ. of California, Berkeley
    - ✓ Turing award, 1989

Floating Point

IEEE FP representation

- Represents numbers in a form $(-1)^s \times M \times 2^E$
  - The sign s determines whether the number is negative (s = 1) or positive (s = 0)
    - ✓ The interpretation of the sign bit for numeric value 0 is handled as a special case
  - The significand M is a fractional binary number that ranges [1, 2)
  - The exponent E weights the value by a power of 2
- Encoding

s	exp	frac

  - MSB is sign bit
  - The **exp** encodes E
  - The **frac** encodes M

Floating Point

IEEE FP representation

- Encoding

s	exp	frac

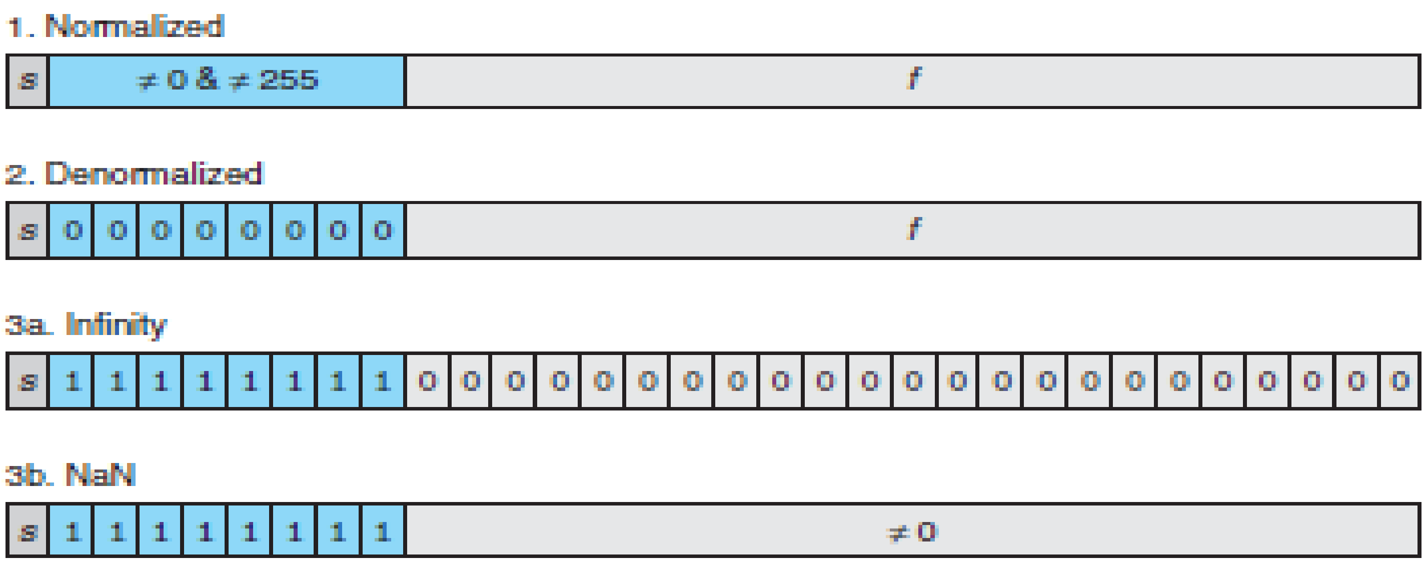
  - MSB is sign bit
  - The **exp** encodes E
  - The **frac** encodes M

- Sizes
  - Single precision
    - ✓ 8 **exp** bits, 23 **frac** bits (32bits total)
  - Double precision
    - ✓ 11 **exp** bits, 52 **frac** bits (64bits total)

Floating Point

■ IEEE FP representation

- Encoding: 3 cases
  - Normalized values
  - Denormalized values
  - Special values



Floating Point

IEEE FP representation

- Encoding: Normalized 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 ($2^{k-1} - 1$)
      - 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 + f = $1.xxx…x_2$ ($0 \le f < 1$)**
      - Minimum when 000…0 (**M** = 1.0)
      - Maximum when 111…1 (**M** = 2.0 - ε)
    - ✓ Get extra leading bit for "free"

Floating Point

IEEE FP representation

- Encoding: Normalized values (Example) **float f = 2003.0;**
  - Value
    2003_{10} = $11111010011_2 = 1.1111010011_2 \times 2^{10}$
  - Significand
    M **=** $1.\underline{1111010011}_2$
    frac = $\underline{1111010011}0000000000000_2$
  - Exponent
    E **= 10**
    Exp **=** $E + Bias = 10 + 127 = 137 = 10001001_2$

```
[Floating Point Representation]
Hex          4      4      F      A      6      0      0      0
Binary    0100   0100   1111   1010   0110   0000   0000   0000
Exp        100   0100   1
frac                    1111   1010   0110
```

Floating Point

IEEE FP representation

- Encoding: Denormalized values
  - Condition
    - ✓ **exp** = 000…0
  - Exponent and significand
    - ✓ Exponent value **E = 1 – Bias**
    - ✓ Significand value **M = 0.xxx…x$_2$** (no implied leading 1)
  - Cases
    - ✓ **exp = 000…0, frac = 000…0**
      - Represents value 0
      - Note that we have distinct values +0 and -0
    - ✓ **exp = 000…0, frac ≠ 000…0**
      - Numbers very close to 0.0
      - **Gradual underflow** property
        (possible numeric values are spaced evenly near 0.0)

Floating Point

IEEE FP representation

- Encoding: Special values
  - Condition
    - ✓ **exp** = 111…1
  - Cases
    - ✓ **exp = 111…1, frac = 000…0**
      - Represents value $\pm\infty$ (infinity)
      - Represents results that overflow
      - Eg) 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)
      - Used when the result of an operation cannot be represented as a real number or infinity
      - Eg) sqrt(-1), $\infty - \infty$, $\infty * 0$, …

Floating Point

Tiny FP example

- 8-bit FP representation
  - The sign bit is in the most significant bit
  - The next four bits are the **exp**, with a bias of 7 ($2^3 - 1$)
  - The last three bits are the **frac**

- Same general form as IEEE format
  - Normalized, denormalized
  - Representation of 0, NaN, infinity

Floating Point

- **Tiny FP example:** Values related to the exponent (*Bias* = 7)

Description	Exp	exp	E = Exp - Bias	2^E
Denormalized	0	0000	-6	1/64
Normalized	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
inf, NaN	15	1111	-	-

Floating Point

Tiny FP example: Dynamic range

Description	Bit representation	e	E	f	M	V
Zero	0 0000 000	0	-6	0	0	0
Smallest positive	0 0000 001	0	-6	1/8	1/8	1/512
	0 0000 010	0	-6	2/8	2/8	2/512
	0 0000 011	0	-6	3/8	3/8	3/512
	0 0000 110	0	-6	6/8	6/8	6/512
Largest denorm.	0 0000 111	0	-6	7/8	7/8	7/512
Smallest norm.	0 0001 000	1	-6	0	8/8	8/512
	0 0001 001	1	-6	1/8	9/8	9/512
	0 0110 110	6	-1	6/8	14/8	14/16
	0 0110 111	6	-1	7/8	15/8	15/16
One	0 0111 000	7	0	0	8/8	1
	0 0111 001	7	0	1/8	9/8	9/8
	0 0111 010	7	0	2/8	10/8	10/8
	0 1110 110	14	7	6/8	14/8	224
Largest norm.	0 1110 111	14	7	7/8	15/8	240
Infinity	0 1111 000	-	-	-	-	+∞

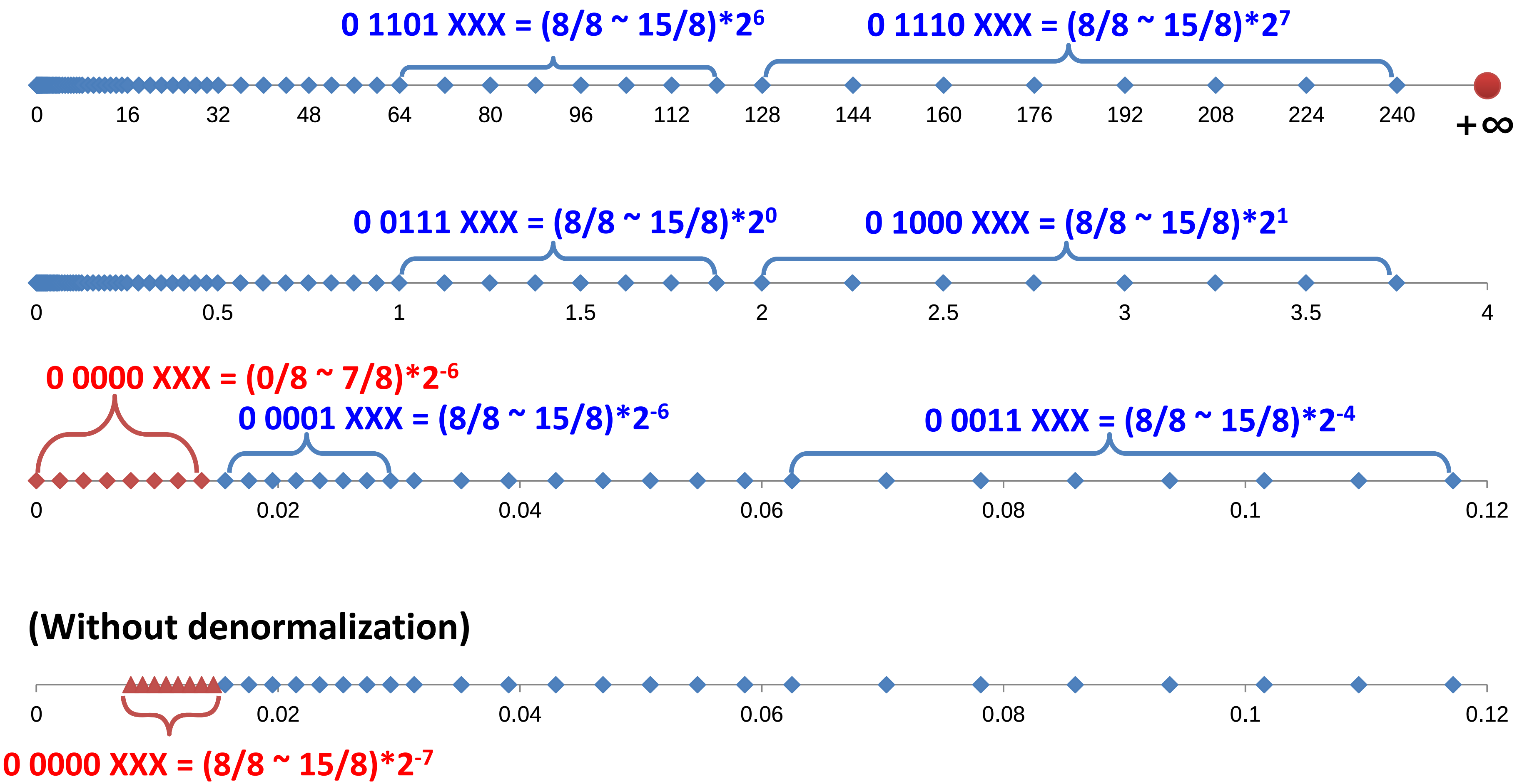
Floating Point

Tiny FP example

		Exponent			Fraction		Value		
Description	Bit representation	e	E	2^E	f	M	$2^E \times M$	V	Decimal
Zero	0 0000 000	0	−6	$\frac{1}{64}$	$\frac{0}{8}$	$\frac{0}{8}$	$\frac{0}{512}$	0	0.0
Smallest pos.	0 0000 001	0	−6	$\frac{1}{64}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{512}$	$\frac{1}{512}$	0.001953
	0 0000 010	0	−6	$\frac{1}{64}$	$\frac{2}{8}$	$\frac{2}{8}$	$\frac{2}{512}$	$\frac{1}{256}$	0.003906
	0 0000 011	0	−6	$\frac{1}{64}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{3}{512}$	$\frac{3}{512}$	0.005859
	⋮								
Largest denorm.	0 0000 111	0	−6	$\frac{1}{64}$	$\frac{7}{8}$	$\frac{7}{8}$	$\frac{7}{512}$	$\frac{7}{512}$	0.013672
Smallest norm.	0 0001 000	1	−6	$\frac{1}{64}$	$\frac{0}{8}$	$\frac{8}{8}$	$\frac{8}{512}$	$\frac{1}{64}$	0.015625
	0 0001 001	1	−6	$\frac{1}{64}$	$\frac{1}{8}$	$\frac{9}{8}$	$\frac{9}{512}$	$\frac{9}{512}$	0.017578
	⋮								
	0 0110 110	6	−1	$\frac{1}{2}$	$\frac{6}{8}$	$\frac{14}{8}$	$\frac{14}{16}$	$\frac{7}{8}$	0.875
	0 0110 111	6	−1	$\frac{1}{2}$	$\frac{7}{8}$	$\frac{15}{8}$	$\frac{15}{16}$	$\frac{15}{16}$	0.9375
One	0 0111 000	7	0	1	$\frac{0}{8}$	$\frac{8}{8}$	$\frac{8}{8}$	1	1.0
	0 0111 001	7	0	1	$\frac{1}{8}$	$\frac{9}{8}$	$\frac{9}{8}$	$\frac{9}{8}$	1.125
	0 0111 010	7	0	1	$\frac{2}{8}$	$\frac{10}{8}$	$\frac{10}{8}$	$\frac{5}{4}$	1.25
	⋮								
	0 1110 110	14	7	128	$\frac{6}{8}$	$\frac{14}{8}$	$\frac{1792}{8}$	224	224.0
Largest norm.	0 1110 111	14	7	128	$\frac{7}{8}$	$\frac{15}{8}$	$\frac{1920}{8}$	240	240.0
Infinity	0 1111 000	—	—	—	—	—	—	∞	—

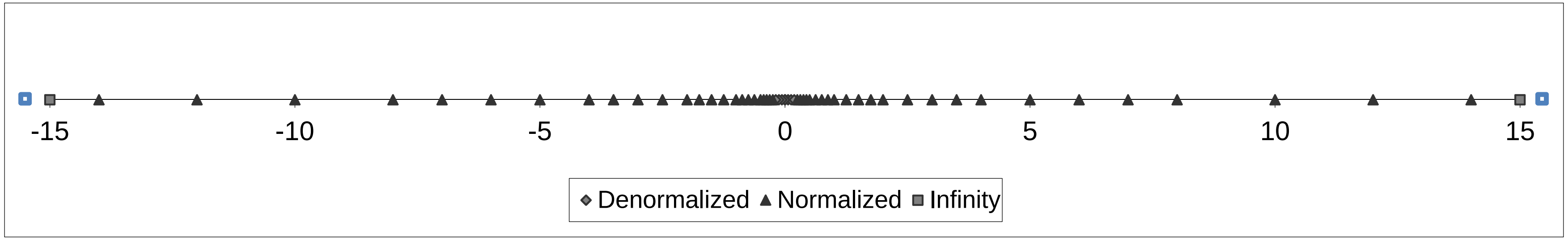
Floating Point

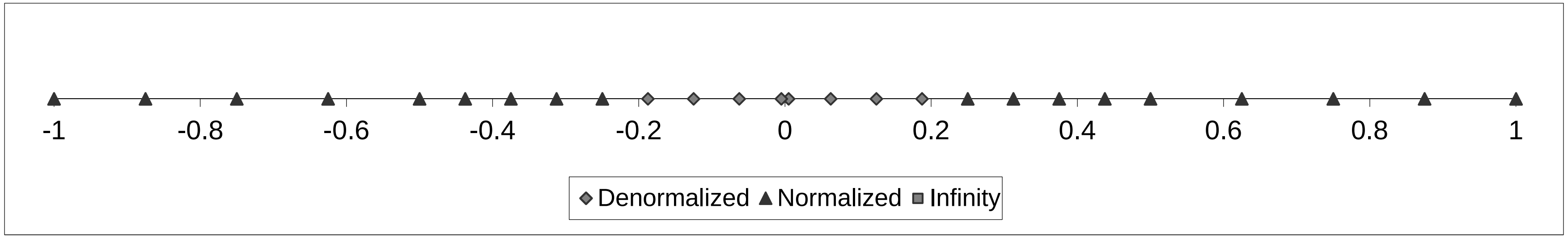
■ Tiny FP example: Encoded values (nonnegative)



Floating Point

- **Tiny FP example: Encoded values for 6-bit FP**
  - 3 exponent bits and 2 fraction bits (bias = 3)





Floating Point

■ Interesting numbers

Description	exp	frac	Numeric value
Zero	000 … 00	000 … 00	0.0
Smallest Positive Denormalized	000 … 00	000 … 01	Single: $2^{-23} \times 2^{-126} \approx 1.4 \times 10^{-45}$ Double: $2^{-52} \times 2^{-1022} \approx 4.9 \times 10^{-324}$
Largest Denormalized	000 … 00	111 … 11	Single: $(1.0 - \varepsilon) \times 2^{-126} \approx 1.18 \times 10^{-38}$ Double: $(1.0 - \varepsilon) \times 2^{-1022} \approx 2.2 \times 10^{-308}$
Smallest Positive Normalized	000 … 01	000 … 00	Single: 1.0×2^{-126}, Double: 1.0×2^{-1022} (Just larger than largest denormalized)
One	011 … 11	000 … 00	1.0
Largest Normalized	111 … 10	111 … 11	Single: $(2.0 - \varepsilon) \times 2^{127} \approx 3.4 \times 10^{38}$ Double: $(2.0 - \varepsilon) \times 2^{1023} \approx 1.8 \times 10^{308}$

Floating Point

Special properties

- FP zero same as integer zero
  - All bits = 0

- To compare two FP numbers
  - Can (almost) use unsigned integer comparison with some considerations
    - ✓ Must first compare sign bits
    - ✓ Must consider -0 = +0
    - ✓ NaNs problematic
      - Will be greater than any other values
    - ✓ Otherwise OK
      - Denormalized vs normalized
      - Normalized vs infinity

Floating Point

Rounding

- A systematic method of finding the "closest" matching value **x**' that can be represented in the desired FP format, for a given value **x**
- IEEE FP format defines 4 different rounding modes
  - Round-to-even (round-to-nearest)
  - Round-toward-zero
  - Round-down
  - Round-up

Floating Point

Rounding

- Round-to-even mode (default)
  - Attempts to find a closest match
  - For halfway values
    - ✓ Rounds upward or downward such that the LSB of the result is even
    - ✓ Avoids the statistical bias (for example, in averaging)
- Round-toward-zero
  - Rounds positive numbers downward and negative numbers upward
  - Takes the value $\mathbf{x}^*$ such that $|\mathbf{x}^*| \leq |\mathbf{x}|$
- Round-down
  - Always rounds downward (takes the value $\mathbf{x}^-$ such that $\mathbf{x}^- \leq \mathbf{x}$)
- Round-up
  - Always rounds upward (takes the value $\mathbf{x}^+$ such that $\mathbf{x}^+ \geq \mathbf{x}$)

Floating Point

Rounding

- Example)

Mode	$1.40	$1.60	$1.50	$2.50	$–1.50
Round-to-even	$1	$2	$2	$2	$–2
Round-toward-zero	$1	$1	$1	$2	$–1
Round-down	$1	$1	$1	$2	$–2
Round-up	$2	$2	$2	$3	$–1

Floating Point

- **Rounding: Round-to-even on binary fractional numbers**
  - Consider
    - LSB 0: even
    - LSB 1: odd
  - Example) Rounding to the nearest quarter
    - $10.00011_{(2)} \rightarrow 10.00_{(2)}$ $\quad 2\frac{3}{32} \rightarrow 2$
    - $10.00110_{(2)} \rightarrow 10.01_{(2)}$ $\quad 2\frac{3}{16} \rightarrow 2\frac{1}{4}$
    - $10.11100_{(2)} \rightarrow 11.00_{(2)}$ $\quad 2\frac{7}{8} \rightarrow 3$
    - $10.10100_{(2)} \rightarrow 10.10_{(2)}$ $\quad 2\frac{5}{8} \rightarrow 2\frac{1}{2}$

Floating Point

FP operations: Multiplication

- Operands
  $(-1)^{\mathbf{s1}}\ \mathbf{M1}\ 2^{\mathbf{E1}} \quad \times \quad (-1)^{\mathbf{s2}}\ \mathbf{M2}\ 2^{\mathbf{E2}}$
- Exact Result
  $(-1)^{\mathbf{s}}\ \mathbf{M}\ 2^{\mathbf{E}}$
  - 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

Floating Point

FP operations: Addition

- Operands
  $(-1)^{s1}\ \mathbf{M1}\ 2^{\mathbf{E1}}$ + $(-1)^{s2}\ \mathbf{M2}\ 2^{\mathbf{E2}}$ (Assume **E1** > **E2**)
- Exact Result
  $(-1)^{s}\ \mathbf{M}\ 2^{\mathbf{E}}$
  - 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

Floating Point

FP operations: Some properties

- FP addition is not associative
  - (3.14 + 1e10) – 1e10 → 0.0 (3.14 is lost due to rounding)
  - 3.14 + (1e10 – 1e10) → 3.14
- FP addition satisfies monotonicity
  - If a ≥ b, then x + a ≥ x + b, for any a, b, x other than NaN
- FP multiplication is not associative
  - (1e20*1e20)*1e-20 → ∞ (in single precision arithmetic)
  - 1e20*(1e20*1e-20) → 1e20
- FP multiplication does not distribute over addition
  - 1e20*(1e20 - 1e20) → 0.0
  - 1e20*1e20 – 1e20*1e20 → NaN

Floating Point

FP in C

- C standard provides two FP types
  - **float** and **double**
- C standard does not require the machine to use IEEE FP
  - Also, no standard methods to change the rounding mode or to get special values such as -0, $+\infty$, $-\infty$, or NaN
- Most systems provide the header files and libraries to provide access to these features
  - But, the details vary from one system to another

Floating Point

FP in C

- 3^{rd} FP types in ISO C99: **long double**
  - Equivalent to **double** type in many machines
  - 80-bit extended precision format in Intel-compatible machines
    - ✓ 15 **exp** bits and 63 **frac** bits (1 bit wasted)

Floating Point

FP in C

- Type conversions
  - Casting between **int**, **float**, and **double** changes numeric values
  - From **int** to **float**
    - ✓ May be rounded
  - From **int** or **float** to **double**
    - ✓ Exact conversion
  - From **double** to **float**
    - ✓ May overflow or may be rounded
  - From **float** or **double** to **int**
    - ✓ Rounded toward 0
    - ✓ May overflow
      - No specifications in C standard

Floating Point

FP puzzles

```
int x, float f, double d; //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;
```

Floating Point

- **Ariane 5 tragedy (Jun. 4, 1996)**
  - Exploded 37 seconds after liftoff
  - Satellites worth $500 million

- **Why?**
  - Computed horizontal velocity as FP number
  - Converted to 16-bit integer
    - Careful analysis of **Ariane 4** trajectory proved 16-bit is enough
  - Reused a module from 10-year-old SW
    - Overflowed for **Ariane 5**
    - No precise specification for the SW

Summary

- **IEEE FP has clear mathematical properties**
  - Represents numbers of form $M \times 2^E$
  - Can reason about operations independent of implementation
    - As if computed with perfect precision and then rounded
  - Not the same as real arithmetic
    - Lacks associativity/distributivity
    - Makes life difficult
      for compilers and serious numerical applications programmers