

15-213 Floating Point January 24, 2002

Topics

- IEEE Floating Point Standard
- Rounding
- Floating Point Operations
- Mathematical properties
- Suggested Reading: 2.4-2.5
- Suggested Practice Problems: 2.50 and 2.51

class04.ppt

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

class04.ppt

- 2 -

15-213 S'02(Based on CS 213 F'01)

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

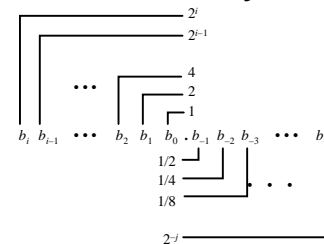
- Nice standards for rounding, overflow, underflow
- Hard to make go fast
 - Numerical analysts predominated over hardware types in defining standard

class04.ppt

- 3 -

15-213 S'02(Based on CS 213 F'01)

Fractional Binary Numbers



Representation

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

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

class04.ppt

- 4 -

15-213 S'02(Based on CS 213 F'01)

Fractional Binary Number Examples

Value **Representation**

5-3/4 101.11₂

2-7/8 10.111₂

63/64 0.111111₂

Observation

- Divide by 2 by shifting right
- Numbers of form 0.111111...₂ just below 1.0
– Use notation 1.0 – ε

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]₂

1/5 0.001100110011[0011]₂

1/10 0.0001100110011[0011]₂

class04.ppt

– 5 –

15-213 S'02(Based on CS 213 F'01)

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 is sign bit
- exp field encodes E
- frac field encodes M

Sizes

- Single precision: 8 exp bits, 23 frac bits
– 32 bits total
- Double precision: 11 exp bits, 52 frac bits
– 64 bits total

class04.ppt

– 6 –

15-213 S'02(Based on CS 213 F'01)

“Normalized” Numeric Values

Condition

- exp ' 000...0 and exp ' 111...1

Exponent coded as *biased* value

$E = \text{Exp} - \text{Bias}$

– 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 = $2^{m-1} - 1$, where m is the 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 ($M = 2.0 - \epsilon$)

– Get extra leading bit for “free”

class04.ppt

– 7 –

15-213 S'02(Based on CS 213 F'01)

Normalized Encoding Example

Value

Float $F = 15213.0$;

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

Significand

$M = 1.1101101101101_2$

frac = 110110110110000000000₂

Exponent

$E = 13$

Bias = 127

Exp = 140 = 10001100₂

Floating Point Representation (Class 02):

Hex: 4 6 6 D B 4 0 0
Binary: 0100 0110 0110 1101 1011 0100 0000 0000

140: 100 0110 0

15213: 1110 1101 1011 01

class04.ppt

– 8 –

15-213 S'02(Based on CS 213 F'01)

Denormalized Values

Condition

- $\text{exp} = 000\dots 0$

Value

- Exponent value $E = -\text{Bias} + 1$
- Significand value $m = 0.x_1x_2\dots x_n$
– $x_1\dots x_n$: bits of frac

Cases

- $\text{exp} = 000\dots 0, \text{frac} = 000\dots 0$
– Represents value 0
– Note that have distinct values $+0$ and -0
- $\text{exp} = 000\dots 0, \text{frac} \neq 000\dots 0$
– Numbers very close to 0.0
– Lose precision as get smaller
– “Gradual underflow”

class04.ppt

– 9 –

15-213 S'02(Based on CS 213 F'01)

Special Values

Condition

- $\text{exp} = 111\dots 1$

Cases

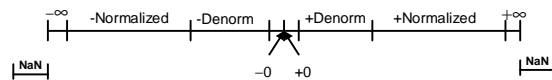
- $\text{exp} = 111\dots 1, \text{frac} = 000\dots 0$
– Represents value ∞ (infinity)
– Operation that overflows
– Both positive and negative
– E.g., $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$
- $\text{exp} = 111\dots 1, \text{frac} \neq 000\dots 0$
– Not-a-Number (NaN)
– Represents case when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

class04.ppt

– 10 –

15-213 S'02(Based on CS 213 F'01)

Summary of Floating Point Real Number Encodings



class04.ppt

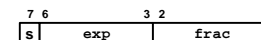
– 11 –

15-213 S'02(Based on CS 213 F'01)

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



class04.ppt

– 12 –

15-213 S'02(Based on CS 213 F'01)

Values related to the exponent

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)

class04.ppt

- 13 -

15-213 S'02(Based on CS 213 F'01)

Dynamic Range

	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	$6/8 * 1/64 = 6/512$
Normalized numbers	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	$8/8 * 1 = 1$
	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

class04.ppt

- 14 -

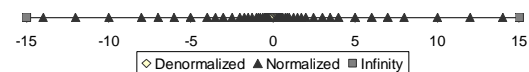
15-213 S'02(Based on CS 213 F'01)

Distribution of Representable Values

6-bit IEEE-like format

- K = 3 exponent bits
- n = 2 significand bits
- Bias is 3

Notice how the distribution gets denser toward zero.



class04.ppt

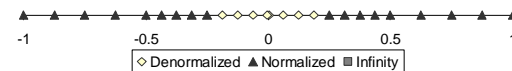
- 15 -

15-213 S'02(Based on CS 213 F'01)

Distribution of Representable Values (close-up view)

6-bit IEEE-like format

- K = 3 exponent bits
- n = 2 significand bits
- Bias is 3



class04.ppt

- 16 -

15-213 S'02(Based on CS 213 F'01)

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 $\gg 1.4 \times 10^{-45}$			
• Double $\approx 4.9 \times 10^{-324}$			
Largest Denormalized	00...00	11...11	$(1.0 - \epsilon) \times 2^{-(126,1022)}$
• Single $\gg 1.18 \times 10^{-38}$			
• Double $\approx 2.2 \times 10^{-308}$			
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	$(2.0 - \epsilon) \times 2^{(127,1023)}$
• Single $\gg 3.4 \times 10^{38}$			
• Double $\approx 1.8 \times 10^{308}$			

class04.ppt

- 17 -

15-213 S'02(Based on CS 213 F'01)

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

class04.ppt

- 18 -

15-213 S'02(Based on CS 213 F'01)

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	\$1.00	\$1.00	\$1.00	\$2.00	-\$1.00
• Round down ($\leftarrow\infty$)	\$1.00	\$1.00	\$1.00	\$2.00	-\$2.00
• Round up ($\rightarrow\infty$)	\$2.00	\$2.00	\$2.00	\$3.00	-\$1.00
• Nearest Even (default)	\$1.00	\$2.00	\$2.00	\$2.00	-\$2.00

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.

class04.ppt

- 19 -

15-213 S'02(Based on CS 213 F'01)

A 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

- 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)

class04.ppt

- 20 -

15-213 S'02(Based on CS 213 F'01)

Rounding Binary Numbers

Binary Fractional Numbers

- “Even” when least significant bit is 0
- Half way when bits to right of rounding position = 100...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

class04.ppt

– 21 –

15-213 S'02(Based on CS 213 F'01)

FP Multiplication

Operands

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

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

Exact Result

$$(-1)^s M 2^E$$

- Sign s : $s1 \wedge s2$
- 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

class04.ppt

– 22 –

15-213 S'02(Based on CS 213 F'01)

FP Addition

Operands

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

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

- Assume $E1 > E2$

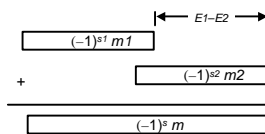
Exact Result

$$(-1)^s M 2^E$$

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

Fixing

- If $M = 2$, 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



class04.ppt

– 23 –

15-213 S'02(Based on CS 213 F'01)

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
- 0 is additive identity? YES
- Every element has additive inverse ALMOST
– Except for infinities & NaNs

Monotonicity

- $a = b \Rightarrow a+c = b+c$? ALMOST
– Except for infinities & NaNs

class04.ppt

– 24 –

15-213 S'02(Based on CS 213 F'01)

Algebraic Properties of FP Mult

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

- $a = b \ \& \ c = 0 \Rightarrow a * c = b * c$? ALMOST
 - Except for infinities & NaNs

class04.ppt

– 25 –

15-213 S'02(Based on CS 213 F'01)

Floating Point in C

C Guarantees Two Levels

`float` single precision
`double` double precision

Conversions

- Casting between `int`, `float`, and `double` changes numeric values
- Double or float to int
 - Truncates fractional part
 - Like rounding toward zero
 - Not defined when out of range
 - » Generally saturates to TMin or TMax
- int to double
 - Exact conversion, as long as int has = 53 bit word size
- int to float
 - Will round according to rounding mode

class04.ppt

– 26 –

15-213 S'02(Based on CS 213 F'01)

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` \nRightarrow `((d*2) < 0.0)` Yes!
- `d > f` \nRightarrow `-f < -d` No
- `d * d >= 0.0` Yes!
- `(d+f)-d == f` No: Not associative

class04.ppt

– 27 –

15-213 S'02(Based on CS 213 F'01)

Summary

IEEE Floating Point 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
 - Violates associativity/distributivity
 - Makes life difficult for compilers & serious numerical applications programmers

class04.ppt

– 28 –

15-213 S'02(Based on CS 213 F'01)