

Lecture #10: Floating Point



2005-07-06

Andy Carle



Quote of the day

“95% of the folks out there are completely clueless about floating-point.”

James Gosling  
Sun Fellow  
Java Inventor  
1998-02-28



Review of Numbers

- Computers are made to deal with numbers
- What can we represent in N bits?
  - Unsigned integers:  
 $0$  to  $2^N - 1$
  - Signed Integers (Two's Complement)  
 $-2^{(N-1)}$  to  $2^{(N-1)} - 1$



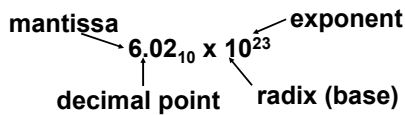
Other Numbers

- What about other numbers?
  - Very large numbers? (seconds/century)  
 $3,155,760,000_{10}$  ( $3.15576_{10} \times 10^9$ )
  - Very small numbers? (atomic diameter)  
 $0.00000001_{10}$  ( $1.0_{10} \times 10^{-8}$ )
  - Rationals (repeating pattern)  
 $2/3$  (0.666666666. . .)
  - Irrationals  
 $2^{1/2}$  (1.414213562373. . .)
  - Transcendentals  
 $e$  (2.718...),  $\pi$  (3.141...)



All represented in scientific notation

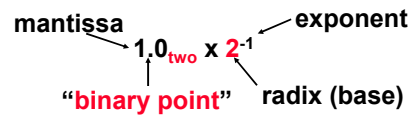
Scientific Notation (in Decimal)



- Normalized form: no leading 0s (exactly one digit to left of decimal point)
- Alternatives to representing 1/1,000,000,000
  - Normalized:  $1.0 \times 10^{-9}$
  - Not normalized:  $0.1 \times 10^{-8}$ ,  $10.0 \times 10^{-10}$



Scientific Notation (in Binary)

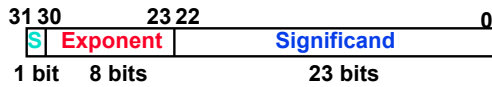


- Normalized mantissa always has exactly one “1” before the point.
- Computer arithmetic that supports it called **floating point**, because it represents numbers where binary point is not fixed, as it is for integers
- Declare such variable in C as float



### Floating Point Representation (1/2)

- Normal format:  $+1.xxxxxxxxxx_{two} * 2^{yyyy}_{two}$
- Multiple of Word Size (32 bits):



- S represents Sign
- Exponent represents y's
- Significand represents x's



Represent numbers as small as  $2.0 \times 10^{-38}$  to as large as  $2.0 \times 10^{38}$

CS 61C L10 Floating Point (7)

A. Carls, Summer 2005 © UC Berkeley

### Floating Point Representation (2/2)

- What if result too large? ( $> 2.0 \times 10^{38}$ )
  - **Overflow!**
  - Overflow  $\Rightarrow$  Exponent larger than represented in 8-bit Exponent field
- What if result too small? ( $>0, < 2.0 \times 10^{-38}$ )
  - **Underflow!**
  - Underflow  $\Rightarrow$  Negative exponent larger than represented in 8-bit Exponent field
- How to reduce chances of overflow or underflow?

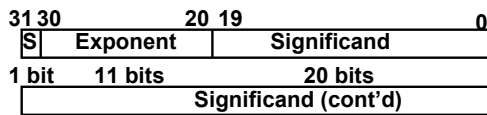


CS 61C L10 Floating Point (8)

A. Carls, Summer 2005 © UC Berkeley

### Double Precision Fl. Pt. Representation

- Next Multiple of Word Size (64 bits)



- **Double Precision** (vs. **Single Precision**)

- C variable declared as `double`
- Represent numbers almost as small as  $2.0 \times 10^{-308}$  to almost as large as  $2.0 \times 10^{308}$
- But primary advantage is greater accuracy due to larger significand



CS 61C L10 Floating Point (9)

A. Carls, Summer 2005 © UC Berkeley

### QUAD Precision Fl. Pt. Representation

- Next Multiple of Word Size (128 bits)
- Unbelievable range of numbers
- Unbelievable precision (accuracy)
- This is currently being worked on
- The version in progress has 15 bits for the exponent and 112 bits for the significand



CS 61C L10 Floating Point (10)

A. Carls, Summer 2005 © UC Berkeley

### IEEE 754 Floating Point Standard (1/4)

- Single Precision, DP similar
- Sign bit: 1 means negative, 0 means positive
- Significand:
  - To pack more bits, leading 1 implicit for normalized numbers
  - 1 + 23 bits single, 1 + 52 bits double
- Note: 0 has no leading 1, so reserve exponent value 0 just for number 0



CS 61C L10 Floating Point (11)

A. Carls, Summer 2005 © UC Berkeley

### IEEE 754 Floating Point Standard (2/4)

- Kahan wanted FP numbers to be used even if no FP hardware; e.g., sort records with FP numbers using integer compares
- Could break FP number into 3 parts: compare signs, then compare exponents, then compare significands
- Wanted it to be faster, single compare if possible, especially if positive numbers
- Then want order:
  - Highest order bit is sign (negative < positive)
  - Exponent next, so big exponent  $\Rightarrow$  bigger #
  - Significand last: exponents same  $\Rightarrow$  bigger #



CS 61C L10 Floating Point (12)

A. Carls, Summer 2005 © UC Berkeley

### IEEE 754 Floating Point Standard (3/4)

#### Negative Exponent?

- 2's comp?  $1.0 \times 2^{-1}$  v.  $1.0 \times 2^{+1}$  ( $1/2$  v.  $2$ )

1/2	0	1111 1111	000 0000 0000 0000 0000 0000
2	0	0000 0001	000 0000 0000 0000 0000 0000

- This notation using integer compare of  $1/2$  v.  $2$  makes  $1/2 > 2!$
- Instead, pick notation **0000 0001** is most negative, and **1111 1111** is most positive
- $1.0 \times 2^{-1}$  v.  $1.0 \times 2^{+1}$  ( $1/2$  v.  $2$ )

1/2	0	0111 1110	000 0000 0000 0000 0000 0000
2	0	1000 0000	000 0000 0000 0000 0000 0000



### IEEE 754 Floating Point Standard (4/4)

- Called **Biased Notation**, where bias is number subtracted to get real number

- IEEE 754 uses bias of 127 for single prec.

- Subtract 127 from Exponent field to get actual value for exponent

#### Summary (single precision):

31	30	23	22	0
S	Exponent		Significand	
1 bit	8 bits		23 bits	

- $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

- Double precision identical, except with exponent bias of 1023



?

0	0111 1101	0000 0000 0000 0000 0000 0000
---	-----------	-------------------------------

Is this floating point number:

- > 0?
- = 0?
- < 0?



### Understanding the Significand (1/2)

#### Method 1 (Fractions):

- In decimal:  $0.340_{10} \Rightarrow 340_{10}/1000_{10} \Rightarrow 34_{10}/100_{10}$

- In binary:  $0.110_2 \Rightarrow 110_2/1000_2 = 6_2/8_{10} \Rightarrow 11_2/100_2 = 3_{10}/4_{10}$

- Advantage: less purely numerical, more thought oriented; this method usually helps people understand the meaning of the significand better



### Understanding the Significand (2/2)

#### Method 2 (Place Values):

- Convert from scientific notation
- In decimal:  $1.6732 = (1 \times 10^0) + (6 \times 10^{-1}) + (7 \times 10^{-2}) + (3 \times 10^{-3}) + (2 \times 10^{-4})$
- In binary:  $1.1001 = (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4})$
- Interpretation of value in each position extends beyond the decimal/binary point
- Advantage: good for quickly calculating significand value; use this method for translating FP numbers



### Example: Converting Binary FP to Decimal

0	0110 1000	101 0101 0100 0011 0100 0010
---	-----------	------------------------------

- Sign: 0 => positive

- Exponent:

- $0110 1000_{\text{two}} = 104_{\text{ten}}$
- Bias adjustment:  $104 - 127 = -23$

- Significand:

- $1 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} + \dots$   
 $= 1 + 2^{-1} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-11} + 2^{-13} + 2^{-15} + 2^{-17} + 2^{-19} + \dots$   
 $= 1.0_{\text{ten}} + 0.666115_{\text{ten}}$

- Represents:  $1.666115_{\text{ten}} \times 2^{-23} \sim 1.986 \times 10^{-7}$   
 (about 2/10,000,000)



### Peer Instruction #1

What is the decimal equivalent of this floating point number?

1 1000 0001 111 0000 0000 0000 0000 0000



### Answer

What is the decimal equivalent of:

1 1000 0001 111 0000 0000 0000 0000 0000

S Exponent Significand

$$(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$$

$$(-1)^1 \times (1 + .111) \times 2^{(129-127)}$$

$$-1 \times (1.111) \times 2^2$$

$$-111.1$$

$$-7.5$$

1:	-1.75
2:	-3.5
3:	-3.75
4:	-7
5:	-7.5
6:	-15
7:	-7 * 2 <sup>129</sup>
8:	-129 * 2 <sup>7</sup>



### Converting Decimal to FP (1/3)

• Simple Case: If denominator is an exponent of 2 (2, 4, 8, 16, etc.), then it's easy.

• Show MIPS representation of -0.75

$$-0.75 = -3/4$$

$$-11_{\text{two}}/100_{\text{two}} = -0.11_{\text{two}}$$

$$\text{Normalized to } -1.1_{\text{two}} \times 2^{-1}$$

$$(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$$

$$(-1)^1 \times (1 + .100\ 0000 \dots 0000) \times 2^{(126-127)}$$

1 0111 1110 100 0000 0000 0000 0000 0000



### Converting Decimal to FP (2/3)

• Not So Simple Case: If denominator is not an exponent of 2.

• Then we can't represent number precisely, but that's why we have so many bits in significand: for precision

• Once we have significand, normalizing a number to get the exponent is easy.

• So how do we get the significand of a never-ending number?



### Converting Decimal to FP (3/3)

• Fact: All rational numbers have a repeating pattern when written out in decimal.

• Fact: This still applies in binary.

• To finish conversion:

• Write out binary number with repeating pattern.

• Cut it off after correct number of bits (different for single v. double precision).

• Derive Sign, Exponent and Significand fields.



### Example: Representing 1/3 in MIPS

• 1/3

$$= 0.33333\dots_{10}$$

$$= 0.25 + 0.0625 + 0.015625 + 0.00390625 + \dots$$

$$= 1/4 + 1/16 + 1/64 + 1/256 + \dots$$

$$= 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} + \dots$$

$$= 0.0101010101\dots_2 \times 2^0$$

$$= 1.0101010101\dots_2 \times 2^{-2}$$

• Sign: 0

• Exponent = -2 + 127 = 125 = 01111101

• Significand = 0101010101...

0 0111 1101 0101 0101 0101 0101 0101 0101



## Administrivia

- Midterm #1
  - Friday, 11:00am – 2:00pm
  - 277 Cory
  - You may bring with you:
    - The green sheet from COD or a photocopy thereof
    - One 8 1/2" x 11" note sheet with *handwritten* notes on *one side*
    - No books, calculators, other shenanigans
- Project 1 is due Sunday night
- HW4 will be due Tuesday



CS 61C L10 Floating Point (26)

A. Carls, Summer 2005 © UC Berkeley

## "Father" of the Floating point standard

### IEEE Standard 754 for Binary Floating-Point Arithmetic.



Prof. Kahan

1989  
ACM Turing  
Award Winner!

[www.cs.berkeley.edu/~wkahan/.../ieee754status/754story.html](http://www.cs.berkeley.edu/~wkahan/.../ieee754status/754story.html)



CS 61C L10 Floating Point (28)

A. Carls, Summer 2005 © UC Berkeley

## Representation for $\pm \infty$

- In FP, divide by 0 should produce  $\pm \infty$ , not overflow.
- Why?
  - OK to do further computations with  $\infty$   
E.g.,  $X/0 > Y$  may be a valid comparison
  - Ask math majors
- IEEE 754 represents  $\pm \infty$ 
  - Most positive exponent reserved for  $\infty$
  - Significands all zeroes



CS 61C L10 Floating Point (27)

A. Carls, Summer 2005 © UC Berkeley

## Representation for 0

- Represent 0?
  - exponent all zeroes
  - significand all zeroes
  - What about sign?
    - +0: 0 00000000 000000000000000000000000
    - -0: 1 00000000 000000000000000000000000
- Why two zeroes?
  - Helps in some limit comparisons
  - Ask math majors



CS 61C L10 Floating Point (28)

A. Carls, Summer 2005 © UC Berkeley

## Special Numbers

- What have we defined so far? (Single Precision)
- | Exponent | Significand    | Object        |
|----------|----------------|---------------|
| 0        | 0              | 0             |
| 0        | <u>nonzero</u> | <u>???</u>    |
| 1-254    | anything       | +/- fl. pt. # |
| 255      | 0              | +/- $\infty$  |
| 255      | <u>nonzero</u> | <u>???</u>    |
- Professor Kahan had clever ideas; "Waste not, want not"
    - Exp=0,255 & Sig!=0 ...



CS 61C L10 Floating Point (29)

A. Carls, Summer 2005 © UC Berkeley

## Representation for Not a Number

- What is `sqrt(-4.0)` or `0/0`?
  - If  $\infty$  not an error, these shouldn't be either.
  - Called Not a Number (NaN)
  - Exponent = 255, Significand nonzero
- Why is this useful?
  - Hope NaNs help with debugging?
  - They contaminate: `op(NaN,X) = NaN`



CS 61C L10 Floating Point (30)

A. Carls, Summer 2005 © UC Berkeley

### Representation for Denorms (1/2)

• **Problem:** There's a gap among representable FP numbers around 0

• **Smallest representable pos num:**

$$a = 1.0 \dots_2 * 2^{-126} = 2^{-126}$$

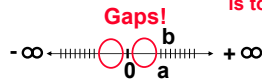
• **Second smallest representable pos num:**

$$b = 1.000 \dots_2 * 2^{-126} = 2^{-126} + 2^{-149}$$

$$a - 0 = 2^{-126}$$

$$b - a = 2^{-149}$$

Normalization and implicit 1 is to blame!



CS 61C L10 Floating Point (31)

A. Carls, Summer 2005 © UC Berkeley

### Representation for Denorms (2/2)

• **Solution:**

• We still haven't used Exponent = 0, Significand nonzero

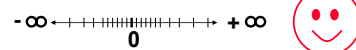
• Denormalized number: no leading 1, **implicit exponent = -126.**

• **Smallest representable pos num:**

$$a = 2^{-149}$$

• **Second smallest representable pos num:**

$$b = 2^{-148}$$



CS 61C L10 Floating Point (32)

A. Carls, Summer 2005 © UC Berkeley

### Peer Instruction 2

1. Converting float  $\rightarrow$  int  $\rightarrow$  float produces same float number
2. Converting int  $\rightarrow$  float  $\rightarrow$  int produces same int number
3. FP add is associative:  
 $(x+y)+z = x+(y+z)$



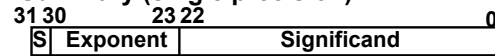
CS 61C L10 Floating Point (33)

A. Carls, Summer 2005 © UC Berkeley

### "And in conclusion..."

- Floating Point numbers approximate values that we want to use.
- IEEE 754 Floating Point Standard is most widely accepted attempt to standardize interpretation of such numbers
  - Every desktop or server computer sold since ~1997 follows these conventions

• **Summary (single precision):**



1 bit 8 bits

23 bits

•  $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$



CS 61C L10 Floating Point (34)

A. Carls, Summer 2005 © UC Berkeley

• Double precision identical, bias of 1023