

Lecture 15
Floating Point I

2006-10-02



Lecturer SOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

#16 Cal rolls over OSU ⇒
Behind the arm of Nate Longshore's 341 yds passing & 4 TDs, the Bears roll 41-13. Recall they stopped our winning streak LAST year... Next wk #11 Oregon @ home. Go Bears!



calbears.cstv.com/sports/m-footbl/recaps/093006aab.html
CS61C L15 Floating Point I (1) Garcia, Fall 2006 © UCB

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



CS61C L15 Floating Point I (2) Garcia, Fall 2006 © UCB

Review of Numbers

- Computers are made to deal with numbers
- What can we represent in N bits?
 - 2^N things, and no more! They could be...
 - Unsigned integers:
 0 to $2^N - 1$
(for $N=32$, $2^N - 1 = 4,294,967,295$)
 - Signed Integers (Two's Complement)
 $-2^{(N-1)}$ to $2^{(N-1)} - 1$
(for $N=32$, $2^{(N-1)} = 2,147,483,648$)



CS61C L15 Floating Point I (3) Garcia, Fall 2006 © UCB

What about other numbers?

1. Very large numbers? (seconds/millennium)
⇒ $31,556,926,000_{10}$ ($3.1556926_{10} \times 10^{10}$)
2. Very small numbers? (Bohr radius)
⇒ $0.000000000529177_{10}m$ ($5.29177_{10} \times 10^{-11}$)
3. Numbers with both integer & fractional parts?
⇒ 1.5

First consider #3.

...our solution will also help with 1 and 2.



CS61C L15 Floating Point I (4) Garcia, Fall 2006 © UCB

Representation of Fractions

“Binary Point” like decimal point signifies boundary between integer and fractional parts:

Example 6-bit representation: $xx.yyyy$
 2^1 2^0 2^{-1} 2^{-2} 2^{-3} 2^{-4}

$$10.1010_2 = 1x2^1 + 1x2^{-1} + 1x2^{-3} = 2.625_{10}$$

If we assume “fixed binary point”, range of 6-bit representations with this format:
0 to 3.9375 (almost 4)



CS61C L15 Floating Point I (5) Garcia, Fall 2006 © UCB

Fractional Powers of 2

i	2^{-i}
0	1.0
1	0.5
2	0.25
3	0.125
4	0.0625
5	0.03125
6	0.015625
7	0.0078125
8	0.00390625
9	0.001953125
10	0.0009765625
11	0.00048828125
12	0.000244140625
13	0.0001220703125
14	0.00006103515625
15	0.000030517578125



CS61C L15 Floating Point I (6) Garcia, Fall 2006 © UCB

Representation of Fractions with Fixed Pt.

What about addition and multiplication?

Addition is straightforward:

$$\begin{array}{r} 01.100 \quad 1.5_{10} \\ 00.100 \quad 0.5_{10} \\ \hline 10.000 \quad 2.0_{10} \end{array}$$

Multiplication a bit more complex:

$$\begin{array}{r} 01.100 \quad 1.5_{10} \\ 00.100 \quad 0.5_{10} \\ \hline 00 \ 000 \\ 000 \ 00 \\ 0110 \ 0 \\ 00000 \\ \hline 00000 \\ 0000110000 \\ \hline \text{HI} \quad \text{LOW} \end{array}$$

Where's the answer, 0.11? (need to remember where point is)



CS61C L15 Floating Point I (7)

Garcia, Fall 2006 © UCB

Representation of Fractions

So far, in our examples we used a "fixed" binary point what we really want is to "float" the binary point. Why?

Floating binary point most effective use of our limited bits (and thus more accuracy in our number representation):

example: put 0.1640625 into binary. Represent as in 5-bits choosing where to put the binary point.

... 000000.001010100000...

Store these bits and keep track of the binary point 2 places to the left of the MSB

Any other solution would lose accuracy!

With floating point rep., each numeral carries a exponent field recording the whereabouts of its binary point.

The binary point can be outside the stored bits, so very large and small numbers can be represented.



CS61C L15 Floating Point I (8)

Garcia, Fall 2006 © UCB

Scientific Notation (in Decimal)

mantissa → 6.02₁₀ × 10²³ ← exponent
 decimal point ← radix (base)

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



CS61C L15 Floating Point I (9)

Garcia, Fall 2006 © UCB

Scientific Notation (in Binary)

mantissa → 1.0_{two} × 2⁻¹ ← exponent
 "binary point" ← radix (base)

- Computer arithmetic that supports it called **floating point**, because it represents numbers where the binary point is not fixed, as it is for integers
 - Declare such variable in C as `float`

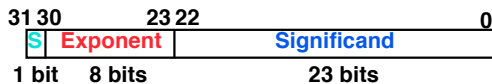


CS61C L15 Floating Point I (10)

Garcia, Fall 2006 © UCB

Floating Point Representation (1/2)

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



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

- Represent numbers as small as 2.0×10^{-38} to as large as 2.0×10^{38}

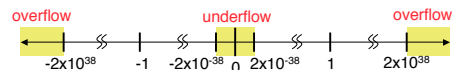


CS61C L15 Floating Point I (11)

Garcia, Fall 2006 © UCB

Floating Point Representation (2/2)

- What if result too large? ($> 2.0 \times 10^{38}, < -2.0 \times 10^{38}$)
 - **Overflow!** ⇒ Exponent larger than represented in 8-bit Exponent field
- What if result too small? ($> 0 \ \& \ < 2.0 \times 10^{-38}, < 0 \ \& \ > -2.0 \times 10^{-38}$)
 - **Underflow!** ⇒ Negative exponent larger than represented in 8-bit Exponent field



- What would help reduce chances of overflow and/or underflow?

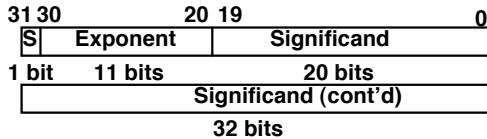


CS61C L15 Floating Point I (12)

Garcia, Fall 2006 © UCB

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 2.0×10^{-308} to almost as large as 2.0×10^{308}
- But primary advantage is greater accuracy due to larger significand



CS61C L15 Floating Point I (13)

Garcia, Fall 2006 © UCB

QUAD Precision Fl. Pt. Representation

- Next Multiple of Word Size (128 bits)
 - Unbelievable **range** of numbers
 - Unbelievable **precision** (accuracy)
- Currently being worked on (IEEE 754r)
 - Current version has 15 exponent bits and 112 significand bits (113 precision bits)

- **Oct-Precision?**

- Some have tried, no real traction so far

- **Half-Precision?**

- Yep, that's for a short (16 bit)

en.wikipedia.org/wiki/Quad_precision

en.wikipedia.org/wiki/Half_precision



CS61C L15 Floating Point I (14)

Garcia, Fall 2006 © UCB

Administrivia...Midterm in 2 weeks!

- **Midterm 2050 VLSB Mon 2006-10-16 @ 7-10pm**

- Conflicts/DSP? Email Head TA Scott, cc Dan

- How should we study for the midterm?

- Form study groups...don't prepare in isolation!
- Attend the review session (2006-10-15 @ 2pm in 105 Northgate)
- Look over HW, Labs, Projects, class notes!
- Write up your handwritten 1-page study sheet
- Go over old exams – HKN office has put them online (link from 61C home page)
- Attend TA office hours and work out hard probs

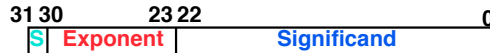


CS61C L15 Floating Point I (15)

Garcia, Fall 2006 © UCB

IEEE 754 Floating Point Standard (1/3)

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
- always true: $0 < \text{Significand} < 1$ (for normalized numbers)

- Note: 0 has no leading 1, so reserve exponent value 0 just for number 0



CS61C L15 Floating Point I (17)

Garcia, Fall 2006 © UCB

IEEE 754 Floating Point Standard (2/3)

- IEEE 754 uses “**biased exponent**” representation.

- Designers wanted FP numbers to be used even if no FP hardware; e.g., sort records with FP numbers using integer compares

- Wanted bigger (integer) exponent field to represent bigger numbers.

- 2's complement poses a problem (because negative numbers look bigger)

- We're going to see that the numbers are ordered EXACTLY as in sign-magnitude

- I.e., counting from binary odometer 00...00 up to 11...11 goes from 0 to +MAX to -0 to -MAX to 0



CS61C L15 Floating Point I (18)

Garcia, Fall 2006 © UCB

IEEE 754 Floating Point Standard (3/3)

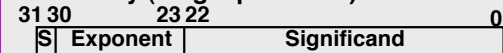
- Called **Biased Notation**, where bias is number subtract to get real number

- IEEE 754 uses bias of 127 for single prec.

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

- 1023 is bias for double precision

- **Summary (single precision):**



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

- Double precision identical, except with exponent bias of 1023 (half, quad similar)



CS61C L15 Floating Point I (19)

Garcia, Fall 2006 © UCB

"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



CS61C L15 Floating Point I (20)

Garcia, Fall 2006 © UCB

Example: Converting Binary FP to Decimal

0 0110 1000 101 0101 0100 0011 0100 0010

- Sign: 0 => positive
- Exponent:
 - 0110 1000_{two} = 104_{ten}
 - Bias adjustment: 104 - 127 = -23
- Significand:
 - $1 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3} + 0x2^{-4} + 1x2^{-5} + \dots$
 - $= 1 + 2^{-1} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-11} + 2^{-13} + 2^{-15} + 2^{-17} + 2^{-19} + \dots$
 - $= 1.0 + 0.666115$
- Represents: $1.666115_{ten} * 2^{-23} \sim 1.986 * 10^{-7}$
(about 2/10,000,000)



CS61C L15 Floating Point I (21)

Garcia, Fall 2006 © UCB

Example: Converting Decimal to FP

-2.340625 x 10¹

1. Denormalize: -23.40625
2. Convert integer part:
 - $23 = 16 + (7 = 4 + (3 = 2 + (1))) = 10111_2$
3. Convert fractional part:
 - $.40625 = .25 + (.15625 = .125 + (.03125)) = .01101_2$
4. Put parts together and normalize:
 - $10111.01101 = 1.011101101 * 2^4$
5. Convert exponent: $127 + 4 = 10000011_2$

1 1000 0011 011 1011 0100 0000 0000 0000



CS61C L15 Floating Point I (22)

Garcia, Fall 2006 © UCB

Understanding the Significand (1/2)

• Method 1 (Fractions):

- In decimal: $0.340_{10} \Rightarrow \frac{340_{10}}{1000_{10}} \Rightarrow \frac{34_{10}}{100_{10}}$
- In binary: $0.1110_2 \Rightarrow \frac{1110_2}{1000_2} = \frac{6_{10}}{8_{10}} \Rightarrow \frac{11_2}{100_2} = \frac{3_{10}}{4_{10}}$
- Advantage: less purely numerical, more thought oriented; this method usually helps people understand the meaning of the significand better



CS61C L15 Floating Point I (23)

Garcia, Fall 2006 © UCB

Understanding the Significand (2/2)

• Method 2 (Place Values):

- Convert from scientific notation
- In decimal: $1.6732 = (1x10^0) + (6x10^{-1}) + (7x10^{-2}) + (3x10^{-3}) + (2x10^{-4})$
- In binary: $1.1001 = (1x2^0) + (1x2^{-1}) + (0x2^{-2}) + (0x2^{-3}) + (1x2^{-4})$
- 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



CS61C L15 Floating Point I (24)

Garcia, Fall 2006 © UCB

"And in conclusion..."

- Floating Point lets us:
 - Represent numbers containing both integer and fractional parts; makes efficient use of available bits.
 - Store approximate values for very large and very small #s.
- 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):

31 30 23 22 0

S	Exponent	Significand	0
---	----------	-------------	---

1 bit 8 bits 23 bits

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



CS61C L15 Floating Point I (27)

Garcia, Fall 2006 © UCB

• Double precision identical, except with exponent bias of 1023 (half, quad similar)