

CS61C : Machine Structures

Lecture 3.2.1

Floating Point

2004-07-06

Kurt Meinz

inst.eecs.berkeley.edu/~cs61c



CS 61C L3.2.1 Floating Point 1 (1)

K. Meinz, Summer 2004 © 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



CS 61C L3.2.1 Floating Point 1 (2)

K. Meinz, Summer 2004 © UCB

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 $2^{(N-1)} - 1$



CS 61C L3.2.1 Floating Point 1 (3)

K. Meinz, Summer 2004 © UCB

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)
 $\frac{2}{3}$ (0.666666666. . .)
 - Irrationals
 $2^{1/2}$ (1.414213562373. . .)
 - Transcendentals
 e (2.718...), π (3.141...)



All represented in scientific notation

CS 61C L3.2.1 Floating Point 1 (4)

K. Meinz, Summer 2004 © UCB

Scientific Notation (in Decimal)

mantissa 6.02_{10} $\times$ 10^{23} 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: 0.1×10^{-8} , 10.0×10^{-10}



CS 61C L3.2.1 Floating Point 1 (5)

K. Meinz, Summer 2004 © UCB

Scientific Notation (in Binary)

mantissa 1.0_{two} $\times$ 2^{-1} exponent
“binary point” radix (base)

- 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`

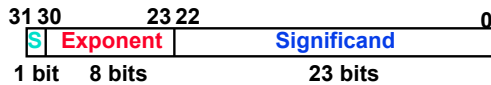


CS 61C L3.2.1 Floating Point 1 (6)

K. Meinz, Summer 2004 © UCB

Floating Point Representation (1/2)

- Normal format: $+1.xxxxxxxx_{two} \cdot 2^{yyyy_{two}}$
- Multiple of Word Size (32 bits):



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



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

CS 61C L3.2.1 Floating Point 1 (7)

K. Meinel, Summer 2004 © UCB

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?

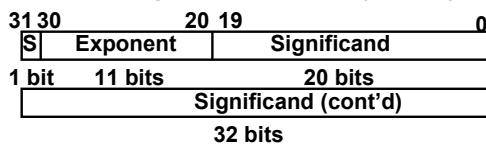


CS 61C L3.2.1 Floating Point 1 (8)

K. Meinel, Summer 2004 © 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



CS 61C L3.2.1 Floating Point 1 (9)

K. Meinel, Summer 2004 © UCB

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



CS 61C L3.2.1 Floating Point 1 (10)

K. Meinel, Summer 2004 © UCB

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 L3.2.1 Floating Point 1 (11)

K. Meinel, Summer 2004 © UCB

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 L3.2.1 Floating Point 1 (12)

K. Meinel, Summer 2004 © UCB

IEEE 754 Floating Point Standard (3/4)

Negative Exponent?

- 2's comp? 1.0×2^{-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
- 1.0×2^{-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

Cal

CS 61C L3.2.1 Floating Point 1 (13)

K. Meinel, Summer 2004 © UCB

IEEE 754 Floating Point Standard (4/4)

- 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):

31	30	23	22	0
S	Exponent		Significant	

1 bit 8 bits

23 bits

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

- Double precision identical, except with exponent bias of 1023

Cal

CS 61C L3.2.1 Floating Point 1 (14)

K. Meinel, Summer 2004 © UCB

Question

- Why is this fp number not == 0?

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

Cal

CS 61C L3.2.1 Floating Point 1 (15)

K. Meinel, Summer 2004 © UCB

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

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

Cal

CS 61C L3.2.1 Floating Point 1 (16)

K. Meinel, Summer 2004 © UCB

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

Cal

CS 61C L3.2.1 Floating Point 1 (17)

K. Meinel, Summer 2004 © UCB

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)

Cal

CS 61C L3.2.1 Floating Point 1 (18)

K. Meinel, Summer 2004 © UCB

Think it over

What is the decimal equivalent of this floating point number?

1 1000 0001 111 0000 0000 0000 0000 0000

- 1: -1.75
- 2: -3.5
- 3: -3.75
- 4: -7
- 5: -7.5
- 6: -15
- 7: $-7 * 2^{129}$
- 8: $-129 * 2^7$



CS 61C L3.2.1 Floating Point 1 (19)

K. Meinel, Summer 2004 © UCB

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: $-7 * 2^{129}$
- 8: $-129 * 2^7$



CS 61C L3.2.1 Floating Point 1 (20)

K. Meinel, Summer 2004 © UCB

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



CS 61C L3.2.1 Floating Point 1 (21)

K. Meinel, Summer 2004 © UCB

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 neverending number?



CS 61C L3.2.1 Floating Point 1 (22)

K. Meinel, Summer 2004 © UCB

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.



CS 61C L3.2.1 Floating Point 1 (23)

K. Meinel, Summer 2004 © UCB

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



CS 61C L3.2.1 Floating Point 1 (24)

K. Meinel, Summer 2004 © UCB

"Father" of the Floating point standard

IEEE Standard 754 for Binary Floating-Point Arithmetic.



Prof. Kahan

www.cs.berkeley.edu/~wkahan/.../ieee754status/754story.html



CS 61C L3.2.1 Floating Point 1 (26)

K. Meina, Summer 2004 © UCB

Representation for $\pm \infty$

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



CS 61C L3.2.1 Floating Point 1 (28)

K. Meina, Summer 2004 © UCB

Representation for 0

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



CS 61C L3.2.1 Floating Point 1 (27)

K. Meina, Summer 2004 © UCB

Special Numbers

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



CS 61C L3.2.1 Floating Point 1 (28)

K. Meina, Summer 2004 © UCB

Representation for Not a Number

- What is $\text{sqrt}(-4.0)$ or $0/0$?
 - If ∞ 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: $\text{op}(\text{NaN}, X) = \text{NaN}$



CS 61C L3.2.1 Floating Point 1 (29)

K. Meina, Summer 2004 © UCB

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:

$$b = 1.000 \dots 1_2 * 2^{-126} = 2^{-126} + 2^{-149}$$

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

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

Normalization and implicit 1 is to blame!



CS 61C L3.2.1 Floating Point 1 (30)

K. Meina, Summer 2004 © UCB

