CS61C Spring 2014 Discussion 6
Floating Point and CALL

1 Whatever Floats Your Boat

1.1 Overview

The IEEE 754 standard defines a binary representation method for floating point values that uses several encodings that you have already seen before. It's a sign and magnitude representation, with the magnitude portion further split into exponent and significand. The exponent is in bias notation (with a bias of 127) and the significand is akin to unsigned, but used to store a fraction instead of an integer.

s	eeeeeeee	mmmmmmmmmmmmmmmmmmmmmmm
Sign 1 bit	Exponent 8 bits	Significand (Mantissa) 23 bits

The above table shows the bit breakdown for the single precision (32-bit) representation defined by the standard. There is also a double precision encoding format that uses 64 bits. This behaves the same as the single precision but uses 11 bits for the exponent (and thus a bias of 1023) and 52 bits for the significand.

1.2 Conversion

Use this table to decode the value of the represented number:

Single Precision		**Double Precision**		**Encoded Value**
Exponent	Mantissa	Exponent	Mantissa	
0	0	0	0	zero
0	nonzero	0	nonzero	± denormalized number
1-254	anything	1-2046	anything	± normalized number
255	0	2047	0	± Infinity
255	nonzero	2047	nonzero	NaN

For normalized values, float $= (-1)^{sign} \times 2^{exponent-bias} \times 1.mantissa_2$
For denormalized values, float $= (-1)^{sign} \times 2^{-bias+1} \times 0.mantissa_2$

1.3 Exercises

1. How many zeroes can be represented using a float? 2 (± 0)

2. What is the largest finite positive value that can be stored using a single precision float? 0x7F7FFFFF $= (2 - 2^{-23}) \times 2^{127} \approx 3.4 \times 10^{38}$

3. What is the smallest positive value that can be stored using a single precision float? 0x00000001 $= 2^{-23} \times 2^{-126} \approx 1.4 \times 10^{-45}$

4. What is the smallest positive normalized value that can be stored using a single precision float? 0x00800000 $= 2^{-126} \approx 1.18 \times 10^{-38}$

5. Convert the following numbers from binary to decimal or from decimal to binary:
   0x0 = 0
   8.25 = 0x41040000
   0xF00 = $(2^{-11} + 2^{-12} + 2^{-13} + 2^{-14}) \times 2^{-127} \approx 5.381 \times 10^{-42}$
   39.5625 = 0x421E4000
   0xFF94BEEF = NaN
   $-\infty$ = 0xFF800000

2 Compile, Assemble, Link, Load, and Go!

2.1 Overview

2.2 Exercises

1. What is the Stored Program concept and what does it enable us to do?
   It is the idea that instructions are just the same as data, and we can treat them as such. This enables us to write programs that can manipulate other programs!

2. How many passes through the code does the Assembler have to make? Why?
   Two, one to find all the label adresses and another to convert all instructions while resolving any forward references using the collected label addresses.

3. What are the different parts of the object files output by the Assembler?
   Header: Size and position of other parts
   Text: The machine code
   Data: Binary representation of any data in the source file
   Relocation Table: Identifies lines of code that need to be "handled" by Linker
   Symbol Table: List of the files labels and data that can be referenced
   Debugging Information: Additional information for debuggers

4. Which step in CALL resolves relative addressing? Absolute addressing? Assembler, Linker.

5. What step in CALL may make use of the `$at` register? Assemble

6. What does RISC stand for? How is this related to pseudoinstructions?
   Reduced Instruction Set Computing. Minimal set of instructions leads to many lines of code. Pseudoinstructions are more complex instructions intended to make assembly programming easier for the coder. These are converted to TAL by the assembler.