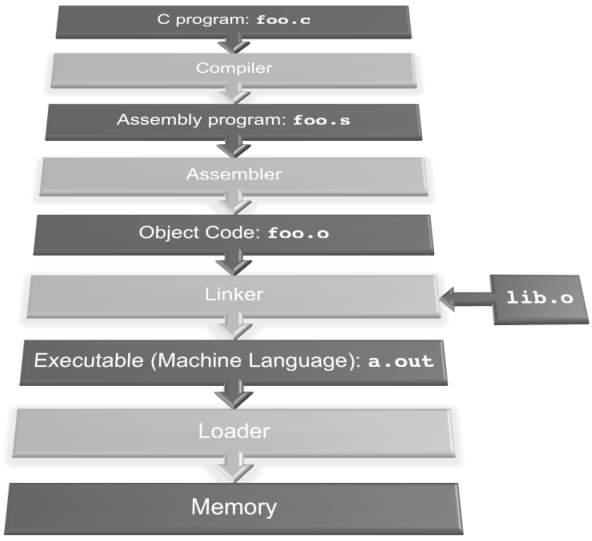
CS61C Summer 17 Discussion 3 – CALL & Floating Point

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

1.1 Overview



1.2 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 addresses 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.

2 Floating Point

2.1 Overview

The IEEE 754 standard defines a binary representation for floating point values using three fields:

- The *sign* determines the sign of the number (0 for positive, 1 for negative)
- The *exponent* is in **biased notation** with a bias of 127
- The *significand or mantissa* is akin to unsigned, but used to store a fraction instead of an integer

The below table shows the bit breakdown for the single precision (32-bit) representation.

Sign	Exponent	Significand
1 bit	8 bits	23 bits

There is 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.
How a float is interpreted depends on the values in the exponent and significand fields:

For normalized floats:
Value $= (-1)^{Sign} * 2^{Exp-Bias} * 1.significand_2$

For denormalized floats:
Value $= (-1)^{Sign} * 2^{Exp-Bias+1} * 0.significand_2$

Exponent	Significand	Meaning
0	Anything	Denorm
1-254	Anything	Normal
255	0	Infinity
255	Nonzero	NaN

2.2 Exercises

1. How many zeroes can be represented using a float?
   2
2. What is the largest finite positive value that can be stored using a single precision float?
   0x7F7FFFFF $= (2 - 2^{-23}) * 2^{127}$
3. What is the smallest positive value that can be stored using a single precision float?
   0X00000001 $= 2^{-23} * 2^{-126}$
4. What is the smallest positive normalized value that can be stored using a single precision float? 0X00800000 $= 2^{-126}$
5. What is the smallest floating point greater than 1? 2? 4? 32?

   $1 + 2^{-23}, 2 + 2^{-22}, 4 + 2^{-21}, 32 + 2^{-18}$

   - Now, for any power of 2, where x = 2^y, what is the difference between x and the next largest floating point?
     2^{-23+y}
6. Cover the following numbers from binary to decimal or from decimal to binary:
   - 0x00000000 = 0
   - 8.25 = 0X41040000
   - 0x00000F00 = $(2^{-12}+2^{-13}+2^{-14}+2^{-15})*2^{-126}$
   - 39.5625 = 0x421E4000
   - 0xFF94BEEF = NaN
   - -∞ = 0xFF800000