CS270 Recitation 8
"More LC-3 Programming"

Goals

To help students with the LC-3 programming assignment on floating-point addition and subtraction, or more specifically:

1. To resolve any problems you might have with the LC-3 assembler and simulator.
2. To explain the LC-3 assignment in more detail and answer any questions you might have.
3. To provide some algorithmic hints about left and right shift, without giving you the code!

Due to limitations on time and resource, this recitation is not intended as a help session to debug your LC-3 code!

The Assignment

1) The teaching assistant will show how to call one LC-3 function from another, by setting parameters and saving and restoring the return address stored in the R7 register.

2) The teaching assistant will present the algorithms for implementing left and right shift, then students will implement either of these and ask questions:

LEFT SHIFT

Step A) Load the bit pattern and number of bits to shift into registers.
Step B) Return the bit pattern unchanged if the number of bits is negative or zero.
Step C) Create a loop that executes number of bits times.
Step D) Each time through loop add the bit pattern to itself, which is the same as shifting left.
Step E) When loop is complete, return the shifted value.

RIGHT SHIFT

Step A) Load the bit pattern and number of bits to shift into registers.
Step B) Initialize all the other registers you will need, and write comments about the register allocation.
Step C) Create a variable for the source mask and destination mask, set to 0x0001 in both cases.
Step D) Shift the source mask left by the number of bits, by calling or copying code from the left shift function.
Step E) Create a loop that executes until the source mask is zero.
Step F) Each time through the loop, do the following:

- Check if the bit in the source mask is also set in the bit pattern.
- If so, set the bit corresponding to the destination mask in the result.
- Shift the source and destination masks left by one bit.

3) The teaching assistant will talk about how to do incremental development for floating point addition, as follows:

Step A) Make sure you can extract the sign, exponent, and mantisaa fields from both floating point operands.
Step B) Make sure you can construct the floating point result from the sign, exponent, and mantissa fields.
Step C) Start with positive operands to avoid needing any 2's complement conversion code.
Step D) Start with identical exponents to avoid needing any normalization code for operands.
Step E) Write the code to add the mantissas together to get the mantissa for the sum.
Step F) Write the code to normalize the resulting sum, at least in the case where right shift is required.
Step G) Test out 5.5 + 6.25 = 11.75. The associated hexadecimal values are 0x4580, 0x4640, and 0x49E0.
Step H) Implement 2's complement conversion for the operands and result, including setting the result sign.
Step I) Test out 5.5 + -2.5 = 3.0, you must figure out the hexadecimal values for this test case.
Step J) Implement normalization of operands, you can assume the first operand exponent >= second operand exponent.
Step K) Test out 11.25 + 6.5 = 17.75, you must figure out the hexadecimal values for this test case.
Step L) Implement floating point subtraction by negating the second operand and calling the addition function.

NOTE: The current implementation we are using for testing has 8 lines of code for left shift and 21 for right shift.