

Assignment 2 — Machine Languages

Due: Monday, October 5th

The LC-3 is an intentionally minimalistic computer architecture featuring simplified versions of the major hardware and software components of “real” modern computers. Nevertheless, in theory, all computable functions can be expressed in the machine language of the LC-3.

Deliverables:

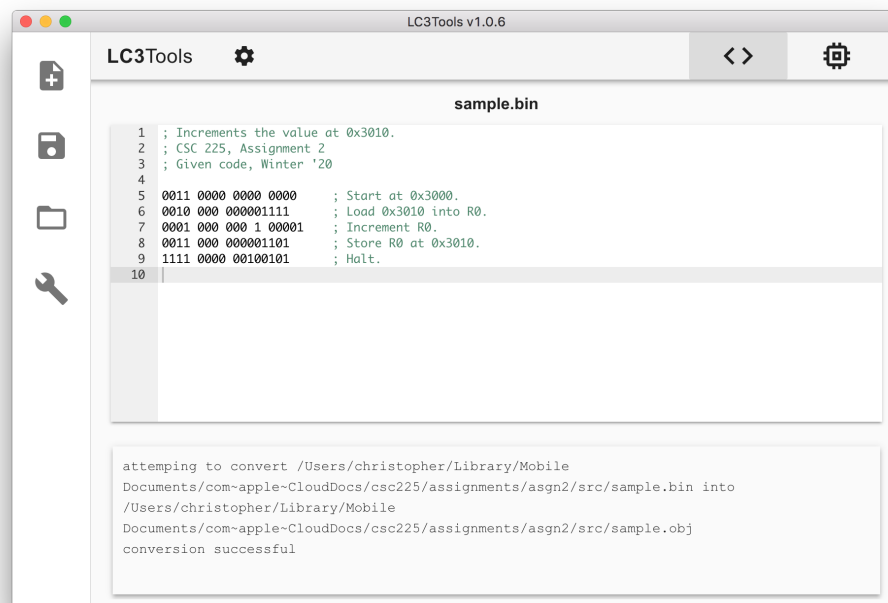
GitHub Classroom: <https://classroom.github.com/a/YkxhLrxv>

Required Files: rotate.bin, reverse.bin



Optional Files: none

Part 1: The LC-3 Simulator



Begin by installing LC3Tools as described here: <https://github.com/chiragsakhuja/lc3tools#quick-start>. This program is available for all major operating systems; it is shown below as it appears on macOS, having loaded and converted the given sample.bin:



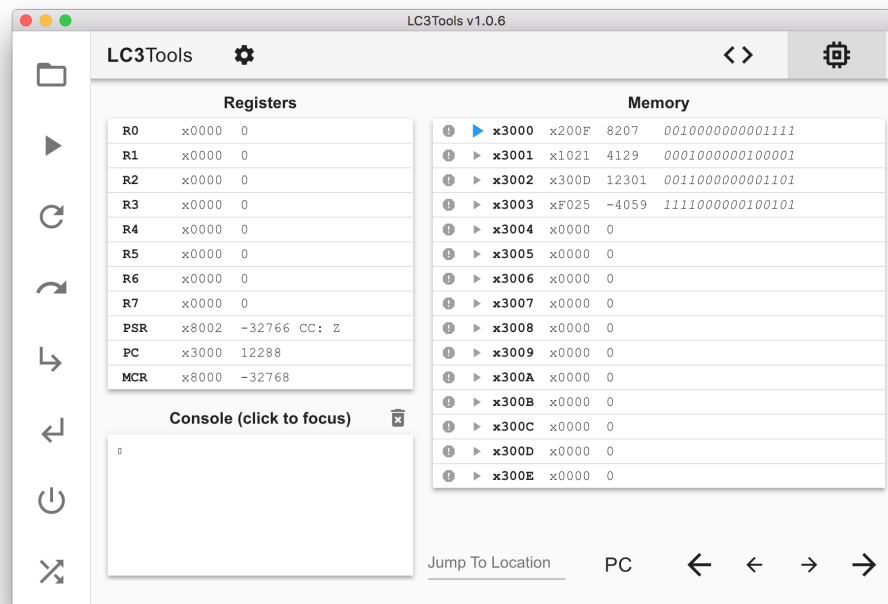
This initial view is the *editor*, which allows you to load, edit, and save machine code.

- The  button loads a machine code program from a file.
- The  button saves the current program.

Recall that true machine code is encoded by the bits ‘0’ and ‘1’; it is not written using the characters ‘0’ and ‘1’. It also can’t contain any extra whitespace or comments for readability. The editor thus also includes a button to convert textual “machine code”, as shown above, into the appropriate actual binary data.

- The  button converts the current program into binary and loads it into the simulator.
- The  button switches over to the simulator.

Once a machine code program has successfully been converted into binary, switching to the *simulator* view shows the program loaded into memory and the PC set accordingly:



Additionally, the simulator displays the entire state of the LC-3:

- Each register is shown along with its value, displayed in both hexadecimal and decimal. Clicking on any register value edits it manually.
- Each memory location is shown with both its address and its value, displayed in both hexadecimal and decimal. Additionally, the simulator realizes that some of the data in memory originated as binary instructions, and displays those instructions. Clicking on any memory value edits it manually.
- The ← and → buttons scroll through memory.
- The small ► button next to each memory location sets the PC to its address.

At this point, the machine code program is ready to be run:

- The ► button begins execution from the current PC.
- The <> button switches back to the editor.

Complete documentation of the LC3Tools editor and simulator can be found here: <https://github.com/chiragsakhuja/lc3tools/blob/master/docs/GuideToUsingLC3Tools.pdf>.

Part 2: The LC-3 ISA

In order to write your own machine code programs for the LC-3, you will likely refer often to the documentation of its valid machine instructions. Their formats and descriptions can be found in the LC-3's ISA, which has been rehosted here: <http://users.csc.calpoly.edu/~cesiu/csc225/slides/lc3isa.pdf>.

Instructions following these formats can be written and converted by the LC3Tools editor, then loaded and run in the LC3Tools simulator, according to the same procedure described above.

A *significant* amount of the work you will do in this course will involve writing, running, and debugging LC-3 programs in this environment. If you have any questions about any of the steps described so far, please do not hesitate to ask.

Part 3: Rotating Bits

Recall that a bitwise *shift* moves every bit of a binary value to the left or to the right. In a left shift, this means that the most significant bit is discarded, whereas in a right shift, the least significant bit is discarded. In a bitwise *rotation*, the bit that would have been discarded is instead placed at the other end.

Complete the machine code program in `rotate.bin`:

- Your program's machine code must begin at memory location `0x3000`.
- After running your program to completion, the value at memory location `0x3050` must have been rotated once to the left.
- It must be possible to rerun your program by manually resetting the PC to `0x3000`. It should not require that the LC-3 be reinitialized or that any files be reloaded.

For example, suppose that, prior to running your program, memory location `0x3050` contains the value:

1001 1001 1001 1001

Then, after running your program to completion, it should contain the value:

0011 0011 0011 0011

Think carefully about which instructions you will use to accomplish this task¹. While it is not a requirement, a clear, concise program should be able to do this using fewer than 10 instructions (including the final `HALT`).

Part 4: Reversing Bits

Complete the machine code program in `reverse.bin`:

- Your program's machine code must begin at memory location `0x3000`.
- After running your program to completion, the value at memory location `0x3050` must be unchanged, while memory location `0x3051` must contain the bits of `0x3050` in reverse order.
- It must be possible to rerun your program by manually resetting the PC to `0x3000`. It should not require that the LC-3 be reinitialized or that any files be reloaded.

For example, suppose that, prior to running your program, memory location `0x3050` contains the value:

1111 1010 0000 0000

Then, after running your program to completion, memory location `0x3051` should contain the value:

0000 0000 0101 1111

A clear, concise program should be able to do this using fewer than 15 instructions.

Part 5: Submission

The following files are required and must be pushed to your GitHub Classroom repository by the deadline:

- `rotate.bin` — A working machine code program for rotating binary values, as specified.
- `reverse.bin` — A working machine code program for reversing binary values, as specified.

The following files are optional:

- `none`

Any files other than these will be ignored.

¹The runtime of a program is directly proportional to the number of instructions that must be executed.