ECE199JL: Introduction to Computer Engineering **Fall 2012**
Homework 12 **Due: at start of lecture on Friday 16 November**

Instruction Set Architecture and the LC-3

Please do problems 4.10, 5.9, 5.14, 5.18 (only for LDR and STI), and 5.26 (a), (b), and (d), 5.34, and 5.40 from the textbook.

Here is an additional problem for this week:

1. Using the LC-3 Tools
Download the two files `hw12-probA.bin` and `hw12-probB.asm` from the class web site onto an EWS lab machine.

A. Let's start by translating a file containing machine instructions coded as ASCII 0s and 1s into a file with just the bits to be loaded into the LC-3 memory.

We have given you a file called `hw12-probA.bin` for this purpose. The `.bin` suffix implies that the file contains binary instruction data (0s and 1s). Take a look at the file in an editor. Notice that the file includes comments explaining the program and comments explaining some of the instructions. Comments start with semicolons—";"—and continue to the end of the line. Comments do not become part of the program.

The first line of bits specifies the starting address of the program. In our file, it is x3000. The remaining bit patterns are converted from ASCII into bits and stored into consecutive locations in the LC-3 memory when the program is loaded into the simulator. Each line of bits contains exactly 16 bits—the number that can be stored in one LC-3 memory location. If a line contains fewer bits, or contains more bits, the conversion tool will produce an error message that identifies the improperly written line. Spaces are ignored when the conversion tool converts from ASCII 0s and 1s into an object file. We use spaces to separate the fields of the instructions in order to make the file easier for humans to read. For data words, we have separated the high eight bits from the low eight bits to make the extended ASCII characters easier to identify.

Convert the `.bin` file into an object file, `hw12-probA.obj`, by typing:

```
lc3convert hw12-probA
```

Now launch the simulator and tell it to load the program (the `.obj` file) that you just created by typing:

```
lc3sim hw12-probA
```

After an introductory message, the simulator halts the LC-3, then loads the object file that you indicated into memory starting at address x3000 (as indicated in the `.obj` file). You'll also see a warning, which we explain in **Part B** below—please ignore the warning for now.

Whenever the LC-3 stops, the simulator prints the current state of the LC-3 registers and gives you information about the memory location referenced by the PC. Recall that this location holds the next instruction to be executed by the processor.

After loading a file, the PC is automatically changed to point to the first instruction in the file that has been loaded, which in our case is x3000. If you need to review the current state of the registers, you can use the `printregs` command, which for convenience can be abbreviated to just the first letter. Type "p" and press <ENTER>.

Notice that the PC has changed to x3000, as already mentioned. The instruction at address x3000 is the first one from our file. The last line printed by the simulator shows the address and contents of this memory location. The contents are shown both as hexadecimal and as the best guess the simulator can make about the meaning of the bits. When the bits encode a valid and useful instruction, as in the case being discussed, the simulator shows that instruction in LC-3 assembly language.

If you want to see more of a program, you can use the `list` command. Type "l" and press <ENTER>.

Notice that the simulator shows you memory locations before and after the current PC value. If you press <ENTER> again, the simulator shows a consecutive group of memory locations.

If you want to look at a specific location in memory in this way, you can give additional arguments to the `list` command. Try typing "l 048e"—the code shown is executed whenever your program executes the HALT system call.

Next try "l 3000 3027"—this command tells the simulator to print the entire binary program that we loaded earlier (which we happen to know fills 40 (hex x28) memory locations).

You could just read the secret message from this printout. Instead, however, use the `continue` command (type "c" and press <ENTER>) to tell the simulator to let the LC-3 execute instructions until it reaches the HALT trap at the end of our program.

Write down and turn in the secret word for this part of the question.

At this point, you may want to just play with some of the other simulator commands. Type "h" for the `help` command.

Ready to quit? Type "q" and press <ENTER>. Didn't work? That's deliberate. When you quit the simulator, you lose all of the LC-3 state, which can be really inconvenient. So you have to type the whole word—"quit"—and then press <ENTER>.

B. Your first lab will use binary, but your last lab in our class will require you to write a program in LC-3 assembly language, which we will explain in the coming weeks. In this part, you'll learn to use the assembler in conjunction with the simulator that you have already used in **Part A**.

We have given you an assembly file called `hw12-probB.asm`. The `.asm` suffix implies that the file contains LC-3 assembly code, which an assembler can change into binary instruction data (0s and 1s). Take a look at the file in an editor. Notice that this file also includes comments using the same syntax as did the binary file.

Unlike the binary file, though, the instructions are written in a more human-friendly form (not 0s and 1s!), and the string at the bottom is like a C string.

Note also the use of the label "MSG" in the LEA instruction and the definition of the label in front of the string. One of the main advantages of an assembler, as you will soon learn, is that you can give symbolic names to memory locations and let the computer do all of the easy-but-painful offset calculations on your behalf. These symbols are stored in another file when you assemble the code, and the simulator makes them available to you when you load the program. Let's go ahead and assemble this code by typing:

```
lc3as hw12-probB
```

The assembler produces two files: an object file, which is the same as before—a file containing 0s and 1s to be loaded to LC-3 memory; and a symbol file (`.sym`). Take a look at the symbol file by typing:

```
cat hw12-probB.sym
```

Notice that the assembler has assigned a memory address to the symbol MSG.

Now start the simulator by typing:

```
lc3sim hw12-probB
```

This time the simulator does not give you any warning when it loads the program, because the corresponding symbol file has been found and loaded as well.

Type "l 3000" to look at the program. Notice that the simulator recognizes the symbol both when it is used by an instruction (look at x3000) and marks the location of the symbol (look at x3003).

The simulator provides a mechanism for you to stop your program in the middle of execution whenever a

particular address is reached. Set a debugging breakpoint in your program by typing "b s 3002", which means: `breakpoint` command `set` breakpoint at address x3002. You can type the longer version if you enjoy typing (`breakpoint set 3002`).

Type "b l" to see a list of breakpoints.

Type "l 3000" again. Notice the "B" to the left of address x3002; that mark indicates that a breakpoint has been set at this location.

Type "c" to let the LC-3 execute instructions. Notice that it stops when it hits the breakpoint, just before it executes the HALT trap at address x3002.

Type "reset" to reset the simulator (clear the state and re-load the last program). The `reset` command cannot be abbreviated. The "r" command is used to set register values.

Type "b l" and notice that your breakpoint is gone. Reset discards all breakpoints, too.

Let's go one instruction at a time. Use the `next` command by typing "n" and pressing <ENTER>. The simulator executes a single LC-3 instruction and stops again. Press <ENTER> to execute another one—the simulator allows you to repeat certain commands, such as `list` and `next`, by simply pressing <ENTER>. One more command and we hit the HALT trap.

You can keep pressing <ENTER> if you want: what's happening is that the LC-3 is executing the HALT trap handler, which simulates a real machine halt after writing the machine control register appropriately, as described in the textbook. We will explain this sort of code later in our class.

Quit the simulator and edit the `.asm` file to replace "Hello, world!" with your own message. Just change the string, and be sure to keep the quotation marks.

Assemble your new program. If you get errors, it might be easiest to just try again on a fresh copy of the file at this point rather than trying to debug. Once you have the file assembled, load it into the simulator and let it run. Congratulations!

Turn in a copy of your code for this part of the question.

C. There is nothing to turn in for this part. Using a lab machine, assemble your program from **Part B**, then type:

```
lc3sim-tk hw12-probB
```

Play with the graphical version of the simulator.