ECE 120: Introduction to Computing **Spring 2019 ZJUI**

due by 11:59:59 p.m. on Monday 13 May 2019

Lab 5: Computing the Greatest Common Denominator with LC-3

Your task in this lab is to write a short LC-3 program to compute the Greatest Common Denominator (GCD) of two numbers using Euclid's algorithm. C code for the algorithm was provided in Homework 10, and a copy has been provided in the `lab5` subdirectory of your Subversion repository (perform `svn update` to obtain it). Your program must load the two numbers from memory, calculate the GCD of the two numbers using Euclid's algorithm, and write the result back to a third memory location. The program requires fewer than 20 instructions, but you may use more, particularly if this assignment is your first programming experience.

The objective for this lab is for you to understand how to express computation using LC-3 instructions and how to encode those instructions in bits. Towards that end, your program must be written in binary and converted into executable form using the `lc3convert` tool. You may wish to read and perform some of the exercises in the **Lab 5 Preparation Guide** in order to prepare you for the lab, but these steps are not required and will not be graded.

The Task

Read the positive numbers stored in memory locations x3100 and x3101, use Euclid's algorithm to calculate the GCD of the two numbers (illustrated by the C subroutine provided to you in `gcd.c`), and store the GCD to memory location x3102.

Specifics

- Your code must be written in LC-3 assembly language and must be contained in a single file called `lab5.bin`. We **will not grade** files with any other name.
- Your program must load its two inputs from memory locations x3100 and x3101.
- You may assume that both input values are positive.
- Your program must store the GCD of the two input values to memory location x3102.
- You may use any registers for this program, but we suggest that you avoid using R7.
- Your code must be well-commented:
    - at the top of your program, you must include a paragraph explaining the purpose of your program and how you approach the problem (**in your own words**);
    - below the description, you must include a table describing how each register is used within your program (only the registers that you use),
    - each line of bits should include spaces between the opcode and each operand/constant field, and
    - each line of bits must include a translation into assembly or RTL.
    - Follow the style of examples provided to you in class and in the textbook.
- Do not forget to include a HALT (TRAP x25) at the end of your program.

Testing

Four test scripts have been provided to you. Once you have written your program, you can use these scripts to test your program using the LC-3 simulator. For example, for the first test, execute

```
lc3sim -s test1 > myout1
```

And then

```
diff out1 myout1
```

The second command compares your program's results with those of a correct program. Some output will be produced, as the final register values are almost definitely different, but the contents of the memory locations (printed using the simulator's "dump" command) should be identical. If they are not, your program has failed the test.

At a minimum, you should try all four tests provided. We suggest that you also create your own tests and make sure that your program produces the correct answers.

Submitting your Lab

Add your **`lab5.bin`** file to your Subversion **`lab5`** subdirectory, then commit it to hand in your final program. Do not add **`.obj`** nor **`.sym`** files to your repository. Note that we will use computer-based comparison tools, so please write your own program and **do not share code with any other student**.

Grading Rubric

Functionality (55%)

- 5% program reads inputs from correct locations and stores result to correct location
- 10% program does not write to any memory location other than x3102
- 25% program produces correct result when M[x3100] > M[x3101]
- 15% program produces correct result when M[x3100] < M[x3101]

Style (20%)

- 5% - program swaps inputs when necessary using a conditional (rather than copying remainder of execution based on ordering of inputs)
- 10% - program uses two nested loops to calculate GCD (**`for`** loop shown in the C code, and a second loop for calculating remainder)
- 5% - program calculates remainder in one place (serving the purpose of both the initialization and the update of the **`for`** loop in the C code)

Comments, Clarity, and Write-up (25%)

- 5% - program includes a paragraph explaining what it does and how it works (these are given to you; you just need to document your work **in your own words**)
- 10% - program includes a table of registers explaining how each is used in the program
- 10% - code is clear and well-commented

Note that some categories in the rubric may depend on other categories and/or criteria. For example, if your code does not read the inputs from the correct memory locations, you will receive no functionality points.