CS2110 Spring 2015

Homework 6

This assignment is due by:

Day: February 17, 2015

Time: 11:54:59pm

Rules and Regulations

Academic Misconduct

Academic misconduct is taken very seriously in this class. Homework assignments are collaborative. However, each of these assignments should be coded by you and only you. This means you may not copy code from your peers, someone who has already taken this course, or from the Internet. You may work with others **who are enrolled in the course,** but each student should be turning in their own version of the assignment. Be very careful when supplying your work to a classmate that promises just to look at it. If he/she turns it in as his own you will both be charged.

We will be using automated code analysis and comparison tools to enforce these rules. **If you are caught you will receive a zero and will be reported to Dean of Students.**

Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know ***IN ADVANCE*** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. No excuses, what you turn in is what we grade. In addition, your assignment must be turned in via T-Square. When you submit the assignment you should get an email from T-Square telling you that you submitted the assignment. If you do not get this email that means that you did not complete the submission process correctly. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over T-Square.

3. There is a random grace period added to all assignments and the TA who posts the assignment determines it. The grace period will last **at least one hour** and may be up to 6 hours and can end on a 5 minute interval; therefore, you are guaranteed to be able to submit your assignment before 12:55AM and you may have up to 5:55AM. As stated it can end on a 5 minute interval so valid ending times are 1AM, 1:05AM, 1:10AM, etc. **Do not ask us what the grace period is we will not tell you**. *So what you should take from this is not to start assignments on the last day and depend on this grace period past 12:55AM.* There is also no late penalty for submitting within the grace period. If you can not submit your assignment on T-Square due to the grace period ending then you will receive a zero, no exceptions.

General Rules

1. In addition any code you write (if any) must be clearly commented and the comments must be meaningful. You should comment your code in terms of the algorithm you are implementing we all know what the line of code does.
2. Although you may ask TAs for clarification, you are ultimately responsible for what you submit.
3. Please read the assignment in its entirety before asking questions.
4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.

Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files.
2. When preparing your submission you may either submit the files individually to T-Square (preferred) or you may submit an archive (zip or tar.gz only please) of the files.
3. If you choose to submit an archive please don't zip up a folder with the files, only submit an archive of the files we want.
4. Do not submit compiled files that is .class files for Java code and .o files for C code.

5. Do not submit links to files. We will not grade assignments submitted this way as it is easy to change the files after the submission period ends.

Objectives

The goal of this assignment is to help you become comfortable coding in the LC-3 assembly language. This will involve the creating small programs, reading input from the keyboard, printing to the console, and converting from high-level code to assembly.

Overview

A Few Requirements

1. Your code must assemble with NO WARNINGS OR ERRORS. To assemble your program, open the file with complx. It will complain if there are any issues.
2. Comment your code! This is especially important in assembly, because it's much harder to interpret what is happening later, and you'll be glad left notes to let you know sections of code or certain instructions are contributing to the code. Comment things like what registers are being used for and what not so intuitive lines of code are actually doing. To comment code in LC-3 assembly just type a semi-colon (;), and the rest of that line will be a comment. Avoid stating the obvious in your comments, it doesn't help in understanding what the code is doing.

Good Comment

```
ADD R3, R3, -1          ;counter--
BRp LOOP                ;if counter == 0 don't loop again
```

Bad Comment

```
ADD R3, R3, 1           ;Decrement R3
BRp LOOP                ;Branch to LOOP if positive
```

3. DO NOT assume that ANYTHING in the LC-3 is already zero. Treat the machine as if your program was loaded into a machine with random values stored in the memory and register file.
4. Following from 3. You can randomize memory by using the menu option State → Randomize. And then load your program by saying File → Load Over
5. You may NOT use the instructions JMP, JSR, or JSRR. Do not write any TRAPs.

6. Do NOT execute any data as if it were an instruction (meaning you should put .fills after halt).
7. Do not add any comments beginning with @plugin or change any comments of this kind.
8. Test your assembly. Don't just assume it works and turn it in.

Assembly Overview

For this assignment, you will be writing 4 assembly programs. The purpose of this assignment is to get your familiar and more comfortable with writing low-level assembly code.

For each part of this homework, we will supply you with pseudo-code, and you must write the equivalent assembly program. Please make your code as similar to the pseudo-code as you can!

You've been given framework files for this assignment, but I'll tell you a little bit about how the assembly files in this class look.

```
.orig x3000
        LEA R0, HW              ;Load the address of the string
        PUTS            ;Output the string
        HALT            ;Stop Running
        HW      .stringz “Hello World.\n”
.end
```

This is a simple assembly program that prints “Hello World.” and a new line to the console.

“;” denotes a comment. You don't need a semi-colon after every line, only before comments.

“.orig” is a pseudo-op. Pseudo-ops are special instructions for the assembler that are not actually assembly instructions. “.orig x3000” tells the assembler to place this block of code at x3000, which is where the LC-3 starts code execution. Therefore “LEA R0, HW” is at address x3000, “PUTS” is at address x3001, etc.

Next is your assembly program. You've seen this before. PUTS is just a pseudonym for a TRAP that prints a string whose address is stored in R0, and HALT is a TRAP that stops the LC-3.

“.stringz” is another pseudo-op the stores the following string at that set of memory locations, followed by a zero (That's what the 'z' is for). For example, 'H' is stored at x3003, 'e' is stored at x3004, etc.

".end" tells the assembler where to stop reading code for the current code block. Every .orig statement must have a corresponding .end statement.

Other pseudo-ops you should know are:

".fill [value]" - put the given value at that memory location (.fill x6000, .fill 7, etc).

".blkw [n]" - reserve the next n memory locations, filling them with 0.

After writing your assembly code, you can test it using Complx. Complx is a full featured LC-3 simulator, and we recommend running your code in the following fashion:

1. Go to the "State" menu and click on "Randomize". This randomly assigns values to every memory location and register, and prevents you from assuming values will be initialized to 0.
2. Go to the "File" menu, select "Load Over" and browse for your assembly file. This will load your code over the randomized memory. You will see your code load in the main window.
3. From here, you can run your code. Click the "Run" button to run your code automatically until a HALT instruction or breakpoint is hit. Click "Step" to execute one instruction at a time. Click "Next Line" to fast-forward through subroutines. Notice you can also step back.

One more thing, it is very important that you use Appendix A in the book to your advantage. Make sure you know what the instruction before you use it. Think about how you can set a register with a specific number, how can you load a value from a label, how can you compare if two numbers are equal, and so on.
Take a look at this instruction: LD R2, 5
What exactly is this instruction doing? Is it loading the number 5 into R2? Remember that the bits preceding the destination register are based on the PCOFFSET9, this is not an immediate value, you are merely loading data from an address location.

Part 1 – deMorgan's Law (demorgans.asm)

In LC-3 assembly you will notice that there is no OR operation. But that can easily be fixed by using deMorgan's Law. Your task is to store the value A | B in your assembly program. deMorgan's law: ~(~A & ~B) == (A | B).

Here's the pseudo-code:

```
/* A and B are the parameters, answer is the return
A = !A;
B = !B;
```

```
answer = A & B;
answer = !answer;
return answer;
```

A and B are labels for a .fill in the template file. You will need to load the values A and B into a register.
ANSWER is a label for another .fill in the template file. You will need to store the value you get for answer at the address labeled ANSWER.

A and B will be any integer in the range -32767 to 32767, inclusive.

Part 2 – Is Multiple (multiple.asm)

Write an assembly program that finds whether A is a multiple of B. If A is a multiple of B, then return 1. Otherwise return 0.

Here's the pseudo-code:

```
while(a > b) {
      a = a - b;
}
if (a == b) {
      return 1;
} else {
      return 0;
}
```

A and B are labels for a .fill in the template file and you will store your answer in the label ANSWER.

A and B are both integers in the range 1 to 32767, inclusive.

Part 3 – Number Count (count.asm)

Write a program that counts how many times a specific number is in an array and return ANSWER.

Here's the pseudo-code:

```
/* number and array are parameters, answer is the return value */

int count = 0;
int negative = 0;
for(int k = 0; k < LENGTH; k++)
{
      if(array[k] == number)
            count += 1;
}
answer = count;
return answer;
```

Again, number and *array are supplied as .fill'd values, and you must store your answer in ANSWER.

number is an integer in the range -32768 to 32767, inclusive.

Part 4 – BubbleSort (bubble.asm)

Sony has now given Marvel the rights to incorporate Spider-Man into the Avengers. However Tony Stark saw that Spidey is not yet fit for battle and decided to put him aside as their new intern in the Avengers tower.

One evening as Spidey sought to pass out the fellow Avengers their Identicards, a sudden tremor erupted in the tower, making him drop all of the cards. He is almost certain that Bruce Banner is responsible. Now all of his card numbers are out of order and he needs to sort them as fast as possible or Tony will make a complaint to Sony! Peter Parker is brilliant scientist so he knows that bubble sort is quick to write and fast on the LC-3 Assembler. Now you gotta help Spidey out by writing bubble sort!

Write a program that sorts an array location at address ARRAY of length LENGTH in ascending order.

Here's the pseudo-code:

```
for (int k = 0; k < LENGTH; k++) {
      int isSorted = 1;
      for (int i = 1; i < LENGTH - k; i++) {
            if (array[i] < array[i - 1]) {
                  int temp = array[i];
                  array[i] = array[i - 1];
                  array[i - 1] = temp;
                  isSorted = 0;
            }
      }
```

```
      if (isSorted) break;
}
```

Here, ARRAY and LENGTH are supplied as .fill'd values. The actual array will be defined in another .orig / .end block, as seen in the template file. You should change the array to test your code.
Helpful hint: press ctrl+v to open up a new view, then ctrl+g to go to x6000. Now you can watch your array change as you debug your instructions!

ARRAY is any valid address and LENGTH is in the range 0 to 100, inclusive.

Deliverables

Remember to put your name at the top of EACH file you submit.

demorgans.asm
multiple.asm
count.asm
bubble.asm