CS2110 Summer 2016
Homework 06

Author: Austin Herring

This assignment is due by:

Day: June 28, 2015
Time: 11:55:00pm

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. After submitting you should be sure to download your submission into a brand new

folder and test if it works. No excuses if you submit the wrong files, 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 6-hour grace period added to all assignments. You may submit your assignment without penalty up until 11:55PM, or with 25% penalty up until 5:55AM. *So what you should take from this is not to start assignments on the last day and plan to submit right at 11:54AM.* You alone are responsible for submitting your homework before the grace period begins or ends; neither T-Square, nor your flaky internet are to blame if you are unable to submit because you banked on your computer working up until 11:54PM. The penalty for submitting during the grace period (25%) or after (no credit) is non-negotiable.

General Rules

1. Starting with the assembly homeworks, 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. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.

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.

5. If you find any problems with the assignment it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

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 unless otherwise noted.

2. When preparing your submission you may either submit the files individually to T-Square or you may submit an archive (zip or tar.gz only please) of the files (preferred). You can create an archive by right clicking on files and selecting the appropriate compress option on your system.

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. (See **Deliverables**).

4. Do not submit compiled files that is .class files for Java code and .o files for C code. Only submit the files we ask for in the assignment.

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.

Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class.

Quizzes, timed labs and the final examination are individual work.

Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using electronic computer programs to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student via electronic means. If you supply an electronic copy of your homework to another student and they are charged with copying you will also be charged. This includes storing your code on any site which would allow other parties to obtain your code such as but not limited to public repositories, etc.

Objectives

The goal of this assignment is to get you familiar with the LC-3 calling convention using assembly recursion. This will involve using the stack as seen in lecture to save the the return address and the old frame pointer.

Overview

A Few Requirements

1. Your code must assemble with NO WARNINGS
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.

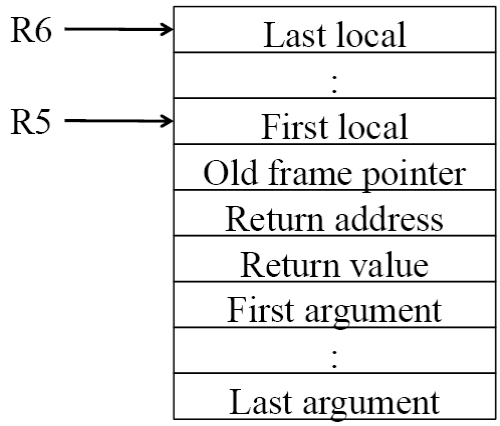
   Good Comment
   ADD R3, R3, -1 ;counter--
   BRp LOOP ;if counter == 0 don't loop again

   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. Use the LC-3 calling convention. This means that all local variables must be pushed onto the stack. Our autograder will be checking for this.
6. Do this assignment using recursion, else you will receive no credit.
7. Start the stack at xF000
8. Do not do bounds checking when pushing and popping
9. Stack pointer points to the last used stack location (Which means allocate positions first, then store onto the stack pointer).
10. Test your assembly. Don't just assume it works and turn it in.

LC-3 Calling Convention Overview

Since you will be using subroutines, you need a way of preserving the old return addresses and arguments before calling another subroutine which will trash these values. As you already know, the stack is a convenient place to keep them. It would make sense to use the stack pointer for figuring out where to push the next value, but it is extremely inconvenient for loading a particular value into a register if you have pushed a whole bunch of things into the stack. You would have to keep track of how many values got pushed into the stack to find the exact address since you will not necessarily be using those values in the order they were pushed into the stack. This is where the frame pointer comes in handy. In LC-3 calling convention, the caller pushes all the arguments into the stack before calling the subroutine. Then the callee reserves space for the return value, pushes the return address (R7), and pushes the old frame pointer (R5) into the stack. The frame pointer should then be modified to point to the address right above where the old frame pointer was stored on the stack. You now know precisely where the old frame pointer, return address, and arguments are stored in relative to the frame pointer regardless where the stack pointer is pointing at. Using this will make debugging and cleanup much easier. Lastly, make sure that the callee also saves R0 to R4 for the caller.



Part 1 Power

You will start by implementing a simple subroutine that returns the *nth* power of *x.* We are providing you with the power.asm file which already calls the subroutine to show you how to properly pass in arguments using the LC-3 calling convention. We are also providing you with the multiply subroutine, so you do not have to worry about writing that. This multiply subroutine follows the calling convention. This means that you will be getting your return value on top of the stack. Once you have finished implementing the power subroutine, you may change the values stored in X and N for testing.

Here is the C version of the subroutine:

```
int power(int x, int n) {
        if(n==0) {
                return 1;
        }
        int temp = power(x,n-1);
        return (x * temp);
}
```

Part 2 Evaluate

The eval function that you will write takes in a mathematical expression consisting of single digit numbers, multiplication, and addition as a string and returns the result of the evaluated expression. The given string will not have any spaces. Examples of valid expressions include:

"4+2*4"

"5*4+3*2"

"9+8+7+6*5+4+3+2+1"

Your eval function must adhere to the LC3 calling convention. The main function that initializes the stack and makes the initial call to eval is already written for you. You just need to write the eval function. The C code for this function is on the next page. In the C code, there are a couple instances where we say "*(str + i)". This is equivalent to "mem[str +i]" where we access the memory at the address str + i (aka, getting the i-th character of the string).

```
#include <stdio.h>
#include <string.h>
int eval(const char *str, int len) {
	int i = 0;
	int left;
	int right;
	while(i < len) {
		if(*(str + i) == '+') {
			left = eval(str, i);
			right = eval(str + i + 1, len - i - 1);
			return left + right;
		}
		i++;
	}
	i = 0;
	while(i < len) {
		if(*(str + i) == '*') {
			left = eval(str, i);
			right = eval(str + i + 1, len - i - 1);
			return left * right;
		}
		i++;
	}
	return *str - '0';
}

int main(int argc, char **argv) {
	int x = eval(argv[1], strlen(argv[1]));
	printf("%d\n", x);
	return 0;
}
```

Deliverables and Reminders

1. power.asm
2. eval.asm

- Remember to put your name at the top of EACH file you submit.
- Make sure to SAVE R0 – R4 in your implementation. Those registers should appear untouched to the user after they call your function.
- Pay attention to the order in which arguments and local variables are placed on the stack.
- START EARLY. If you wait until the last day, we will most likely not be able to help as much.