CS 2110 Homework 7
Recursive Assembly

Sam Gilson, Chris Messina, Matthew Musselman, Joshua Viszlai

Fall 2019

Contents

1 Overview

In this homework, you will be writing assembly code for the following tasks:

- `factorial()`: Compute the factorial of a number recursively
- `recursiveGCD()`: Compute the Greatest Common Divisor of two integers recursively
- `multiplyParameters()`: Given the number of parameters passed into the function as the first argument, multiply the rest of the arguments together and return it
- `linkedListSumOrFactorial()`: Given a linked list, replace each nodes value with either the sumtorial or factorial of the original value

You should follow the LC-3 calling convention when writing these functions. A detailed description of the calling convention is available in Part 5 of this PDF.

2 Instructions

2.1 Part 1

~~2.2 Factorial~~

For the first part of this homework, we will be writing a function that computes the factorial of a number.

2.2.1 Pseudocode

For the factorial subroutine, we will be taking a single parameter and computing the factorial of that number, which you may recognize in the following format: $7! = 7 * 6 * 5 * 4 * 3 * 2 * 1 = 5040$. The pseudocode is as follows:

```
factorial(int a)
{
    if (a == 1)
    {
        return a;
    }

    var result = a * factorial(a - 1);
    return result;
}
```

2.3 Part 2

~~2.4 RecursiveGCD~~

For this part, you will be recursively determining the greatest common divisor of two integers passed in as arguments. The greatest common divisor of two integers a and b is defined as the greatest integer that divides both a and b.

2.4.1 Pseudocode

Following is the pseudocode for determining the GCD of two integers a and b.

```
recursiveGCD(int a, int b)
{
    if (a < b) {
        return recursiveGCD(b, a);
    }
    else if (a == b) {
        return a;
    }
    else {
        return recursiveGCD(a - b, b);
    }
}
```

2.5 Part 3

~~2.6 multiplyParameters~~

You should complete the `multiplyParameters` subroutine. As in the description, `multiplyParameters` is a bit different than normal subroutines as it takes in a variable number of arguments. The first argument, `n`, has a range of `[0, MAX_NUM_OF_ARGS]` and is the count of remaining arguments (`MAX_NUM_OF_ARGS` is just a very large constant that states how many arguments the subroutine could potentially accept). Again, as in the description, the remaining arguments can be visualized as an array that is stored on the stack instead of in another region of memory. This function should multiply all the arguments in our variable array together and return the result of this multiplication (or 1 if there are zero arguments). Below is a diagram of the stack to help you visualize how these arguments have been stored by the caller and a few examples to make sure you understand the concept.

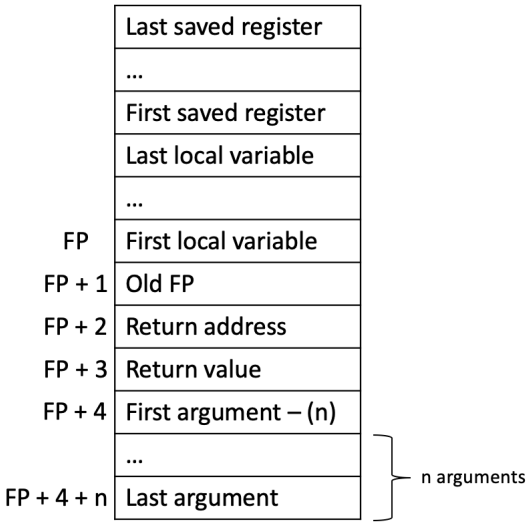


Figure 1: Example Stack – Very cool and useful!

2.6.1 Example 1

```
multiplyParameters(4, 7, 3, 1, 14)
```

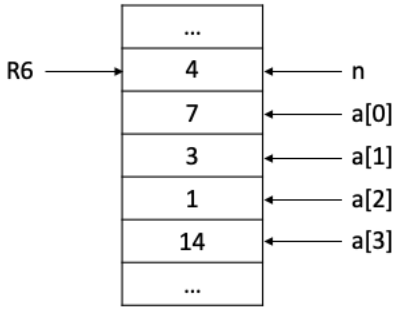


Figure 2: The argument portion of the stack frame for this example.

2.6.2 Example 2

```
multiplyParameters(2, 7, 3)
```

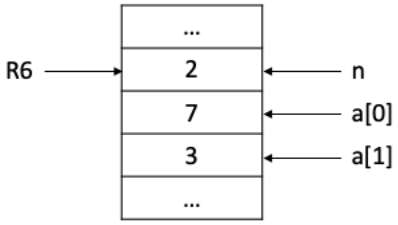


Figure 3: The argument portion of the stack frame for this example.

2.7 Part 4

2.8 Factorial Or Sumtorial Replacement

For this part of the homework, you will be writing a function to replace the value at each node of a linked list with either the factorial or sumtorial of the original value, then return the current node. You will perform the sumtorial if the value of the node is even, and perform the factorial if it is odd. As an example, 4 would get replaced with sumtorial(4), which is defined as

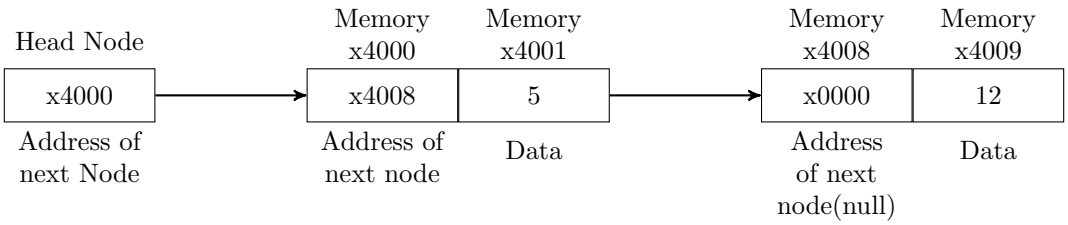
$sumtorial(4) = 4 + 3 + 2 + 1 = 10$

As a second example, if the value of the node was 5, it would get replaced with 5!, which is defined as

$5! = 5 * 4 * 3 * 2 * 1 = 120$

2.9 Linked List Data Structure

The below figure depicts how each node in our linked list is laid out. Each node will have two attributes: the next node, and a value for that node.



2.9.1 Pseudocode

The pseudocode for this function is as follows. **Note: null has a value of 0.** Since memory address zero is used to hold part of the trap vector, we can use zero as a value distinguished from all of the memory addresses in our linked list.

```
sumtorialAndFactorialLL(Node head) {
     if (head == null) {
         return head
     }

     if (head.data % 2 == 0) {
         head.data = sumtorial(head.data)
     } else {
         head.data = factorial(head.data)
     }
     sumtorialAndFactorialLL(head.next)
     return head;
}
```

3 Checker

To run the autograder locally, follow the steps below depending upon your operating system:

- Mac/Linux Users:
  1. Navigate to the directory your homework is in. **In your terminal, not in your browser**
  2. Run the command `sudo chmod +x grade.sh`
  3. Now run `./grade.sh`
- Windows Users:
  1. On **docker quickstart**, navigate to the directory your homework is in
  2. Run `./grade.sh`

Note: The checker may not reflect your actual grade on this assignment. We reserve the right to update the checker as we see fit when grading.

4 Deliverables

Please turn in the following files to gradescope:

1. `factorialLinkedList.asm`
2. `gcd.asm`
3. `multiplyArgs.asm`

5 LC-3 Assembly Programming Requirements

5.1 Overview

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. **If your code does not assemble you WILL get a zero for that file.**
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 you left yourself notes on what certain instructions are contributing to the code. Comment things like what registers are being used for and what less intuitive lines of code are actually doing. To comment code in LC-3 assembly just type a semicolon (;), and the rest of that line will be a comment.
3. 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
   ```

4. **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.
5. Following from 3. You can randomize the memory and load your program by doing File - Randomize and Load.
6. Use the LC-3 calling convention. This means that all local variables, frame pointer, etc... must be pushed onto the stack. Our autograder will be checking for correct stack setup.
7. Start the stack at xF000. **The stack pointer always points to the last used stack location.** This means you will allocate space **first**, then store onto the stack pointer.
8. Do NOT execute any data as if it were an instruction (meaning you should put .fills after **HALT** or RET).
9. Do not add any comments beginning with @plugin or change any comments of this kind.
10. **Test your assembly.** Don't just assume it works and turn it in.

6 Rules and Regulations

6.1 General Rules

1. Starting with the assembly homeworks, any code you write must be meaningfully commented. You should comment your code in terms of the algorithm you are implementing; we all know what each 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.

6.2 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 Canvas/Gradescope or you may submit an archive (zip or tar.gz only please) of the files. You can create an archive by right clicking on files and selecting the appropriate compress option on your system. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.

3. 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.

4. Do not submit links to files. The autograder does not understand it, and we will not manually grade assignments submitted this way as it is easy to change the files after the submission period ends.

6.3 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 Canvas/Gradescope. 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 Canvas/Gradescope.

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 Canvas/Gradescope, 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.

6.4 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 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. This includes simply e-mailing it to them so they can look at it. 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 (Github), pastebin, etc. If you would like to use version control, use github.gatech.edu

6.5 Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, share algorithms and pseudo-code, as well as help each other debug code. What you shouldn't be doing, however, is pair programming where you collaborate with each other on a single instance of the code. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, because it is frequently the case that the recipient will simply modify the code and submit it as their own.

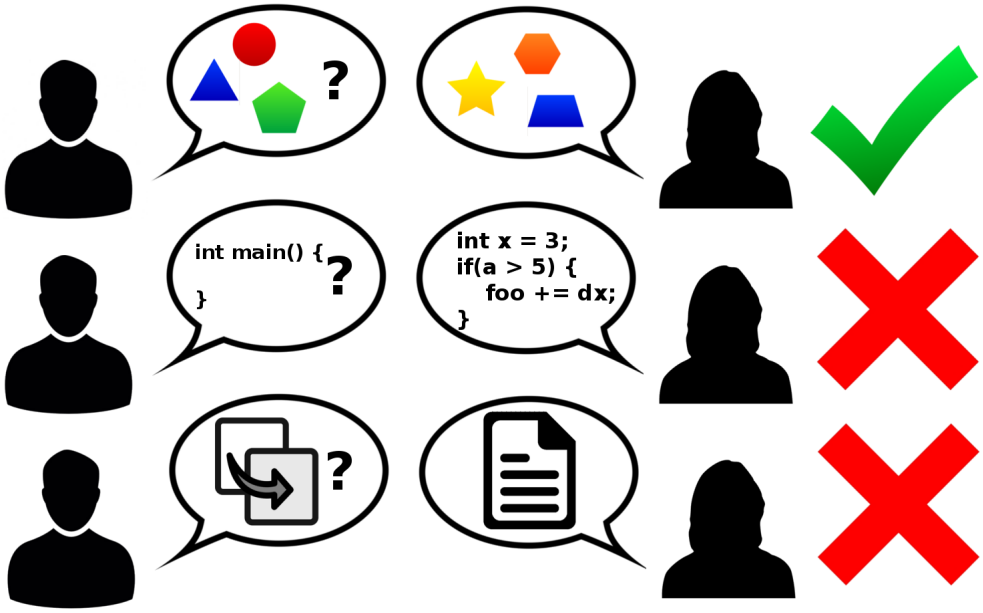


Figure 4: Collaboration rules, explained colorfully