CS 2110 Timed Lab 4: Recursive Assembly

Your TAs

Fall 2019

Contents

Please take the time to read the entire document before starting the assignment. It is your responsibility to follow the instructions and rules.

x

1 Timed Lab Rules - Please Read

1.1 General Rules

1. You are allowed to submit this timed lab starting at the moment the assignment is released, until you are checked off by your TA as you leave the recitation classroom. Gradescope submissions will remain open until 7:15 pm - but you are not allowed to submit after you leave the recitation classroom under any circumstances. **Submitting or resubmitting the assignment after you leave the classroom is a violation of the honor code - doing so will automatically incur a zero on the assignment and might be referred to the Office of Student Integrity.**
2. Make sure to give your TA your Buzzcard before beginning the Timed Lab, and to pick it up and get checked off before you leave. **Students who leave the recitation classroom without getting checked off will receive a zero.**
3. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. **The information provided in this Timed Lab document takes precedence.** If in doubt, please make sure to indicate any conflicting information to your TAs.
4. Resources you are allowed to use during the timed lab:
   - Assignment files
   - Previous homework and lab submissions
   - Your mind
   - Blank paper for scratch work (please ask for permission from your TAs if you want to take paper from your bag during the Timed Lab)
5. Resources you are **NOT** allowed to use:
   - The Internet (except for submissions)
   - Any resources that are not given in the assignment
   - Textbook or notes on paper or saved on your computer
   - Email/messaging
   - Contact in any form with any other person besides TAs
6. **Before you start, make sure to close every application on your computer.** Banned resources, if found to be open during the Timed Lab period, will be considered a violation of the Timed Lab rules.
7. We reserve the right to monitor the classroom during the Timed Lab period using cameras, packet capture software, and other means.

1.2 Submission Rules

1. Follow the guidelines under the Deliverables section.
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 Gradescope.

3. 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.

1.3 Is collaboration allowed?

Absolutely NOT. No collaboration is allowed for timed labs.

2 Overview

In this timed lab, you will writing a subroutine that operates on a Linked List. Specifically, for each node in the linked list, for that node's value v, you will compute v!, otherwise known as the factorial of v, and sum all the v! together. At the same time, you will overwrite the value at the address of the nodes data with v!. As a refresher, the factorial of a natural number $v \in \mathbb{N}$, is defined as

$v! = v * (v-1) * (v-2) * \ldots * 2 * 1$

As an example, 5! is defined as

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

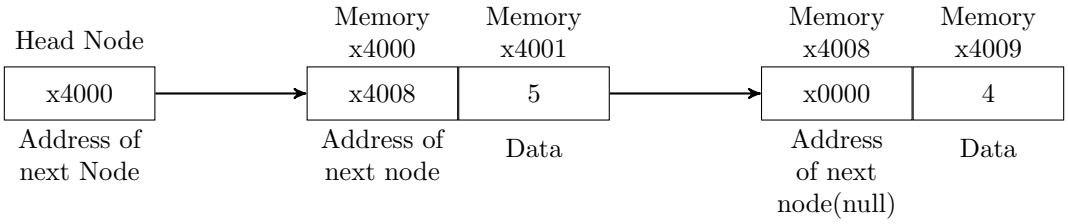
As an example, if you had the following node values for your linked list: $[2, 5, 3, 7]$ then what we are looking for you to produce is

$sum = 2! + 5! + 3! + 7!$
$sum = 2 + 120 + 6 + 5040$
$sum = 5168$

Pseudocode for what we are expecting you to do will be given later in the PDF. You will also be given a factorial subroutine in order to calculate the factorial of every node value. Keep in mind the linked list will be set up in a very similar way to the linked lists you've seen in previous homeworks. As an additional point, keep in mind you will be writing a subroutine, so you must follow proper calling conventions.

3 Linked List Data Structure and Example

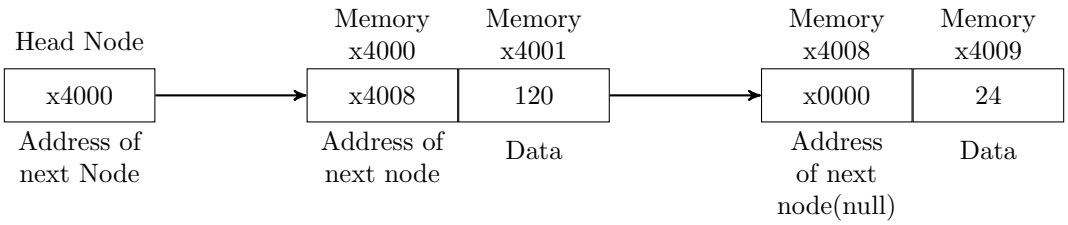
The below figure depicts how each node in our linked list is laid out. Each node will have two attributes: the next node, and a value for that node.



In this case, our expected return from your subroutine would be

$sum = 5! + 4!$
$sum = 120 + 24$
$sum = 144$

With resulting linked list



4 Instructions

Highlighted lines in the pseudocode are the new lines necessary to get from the previous checkpoint to the current checkpoint.
/* NOTE: Each of the variables in the pseudocode below can be accessed through labels in tl4.asm */

4.1 Checkpoint 1: Finding the Head (15 points)

Save the address of the head to the LISTHEAD label. LISTHEAD will be 0 by default (and should be nonzero if the head has already been written).
Pseudocode:

```
LISTHEAD = 0    //label
int sumFactorialsLL(Node head) {

    if (LISTHEAD == 0) {
        LISTHEAD = head
    }

    return 0    //output is not checked
}
```

4.2 Checkpoint 2: Replace head with factorial (15 points)

Overwrite the head's data with the factorial of the data. Also, add a check to make sure head is not NULL.

```
LISTHEAD = 0
int sumFactorialsLL(Node head) {

    if (head == NULL) { //AKA head == 0
        return 0
    }

    if (LISTHEAD == 0) {
        LISTHEAD = head
    }

    head.data = factorial(head.data)

    return 0    //output is not checked
}
```

4.3 Checkpoint 3: Recursive call (20 points)

Call sumFactorialsLL(head.next). The return value is not used in this checkpoint, but it will be used later.

```
LISTHEAD = 0
int sumFactorialsLL(Node head) {

    if (head == NULL) {     //AKA head == 0
        return 0
    }

    if (LISTHEAD == 0) {
        LISTHEAD = head
    }

    head.data = factorial(head.data)

    sumFactorialsLL(head.next)

    return 0    //output is not checked
}
```

4.4 Checkpoint 4: Finding the tail (15 points)

Save the address of the tail node to LISTTAIL. The tail node will have tail.next == NULL.

```
LISTHEAD = 0
LISTTAIL = 0
int sumFactorialsLL(Node head) {

    if (head == NULL) {     //AKA head == 0
        return 0
    }

    if (LISTHEAD == 0) {
        LISTHEAD = head
    }

    if (head.next == NULL) { //AKA head.next == 0
        LISTTAIL = head
    }

    head.data = factorial(head.data)

    sumFactorialsLL(head.next)

    return 0    //output is not checked
}
```

4.5 Checkpoint 5: Returning the sum of factorials (20 points)

Add together the new value of head.data and the return value of sumFactorialsLL. Return this value.

```
LISTHEAD = 0
LISTTAIL = 0
int sumFactorialsLL(Node head) {

    if (head == NULL) {     //AKA head == 0
        return 0
    }

    if (LISTHEAD == 0) {
        LISTHEAD = head
    }

    if (head.next == NULL) {    //AKA head.next == 0
        LISTTAIL = head
    }

    head.data = factorial(head.data)

    sum = sumFactorialsLL(head.next)

    return sum + head.data
```

4.6 Other Requirements (15 points)

Your subroutine must be a recursive subroutine that follows the LC-3 calling convention. Specifically, it must fulfill the following conditions:

- When your subroutine returns, every register must have its original value preserved (except R6).
- When your subroutine returns, the stack pointer (R6) must be decreased by 1 from its original value so that it now points to the return value.
- During the execution of your subroutine, you must make one call to `sumFactorialsLL` and one call to `factorial` (as described in the pseudocode).
  - This only applies to cases where the head is a non-null node. If the head is null, you do not need to call both subroutines.
  - If the autograder claims that you are making an unknown subroutine call to some label in your code, it may be that your code has two labels without an instruction between them. Removing one of the labels should appease the autograder.

5 Local Grader

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

- Mac/Linux Users:
  1. Navigate to the directory your timed lab 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`

The output of the autograder is an approximation of your score on this timed lab. It is a tool provided to students so that you can evaluate how much of the assignment expectations your submission fulfills. However, **we reserve the right to run additional tests, fewer tests, different tests, or change individual tests** - your final score will be determined by your instructors and no guarantee of tester output correlation is given.

6 Deliverables

Please upload the following files to Gradescope:

1. tl4.asm

Download and test your submission to make sure you submitted the right files

7 LC-3 Assembly Programming Requirements

7.1 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. You should not use a compiler that outputs LC3 to do this assignment.
11. **Test your assembly.** Don't just assume it works and turn it in.