CIT 593 – Module 06 Assignment

Assembly Programming Instructions

Contents

Assignment Overview 3
Learning Objectives 3
Advice 3
Getting Started 4
- Codio Setup 4
  - Open the PennSim Window 4
  - Start PennSim in the Terminal Command Line 6
  - Troubleshooting Codio Issues 7
  - Run multiply.asm in PennSim 8
- Starter Code 10

Requirements 11
- General Requirements 11
- Part 1: While Loops in Assembly 11
- Part 2: Subroutines in Assembly 11
- Part 3: Working with Data Memory 12
- Extra Credit 12

Suggested Approach 13
- High Level Overview 13
- Great High Level Overview, but I really need a Slightly More Detailed Overview 14
  - Part 1: While Loops in Assembly 14
  - Part 2: Subroutines in Assembly 16
  - Part 3: Working with Data Memory 17
  - Part 4: Extra Credit 18

Submission 19
- Where to put the files 19
- Pre-Submission Test 19
- The Actual Submission 19
  - Codio Submission 19
  - Gradescope Submission 19
  - Academic Integrity Agreement: 19

Grading 20
- Main Assignment 20
- Extra Credit 20
- An Important Note of Plagiarism 20

FAQ 21
Quick Hints 21
Resources 21

Assignment Overview

In this assignment, we will start programming in the LC4 assembly language. We also introduce PennSim, a computer that can assemble and run LC4 programs. Finally, we introduce Codio, a remote hosted Linux platform that provides a standardized environment to ensure consistent grading.

Learning Ob

This assignment will [illegible] pics:

- Program a sir[illegible]C4 Assembly
- Create a subr[illegible]bly
- Work with Data Memory in LC4 Assembly
- (optionally) introduce the concept of pointers early

Advice

- Start early
- Ask for help early
- Do not try to do it all in one day

Getting Started

Codio Setup

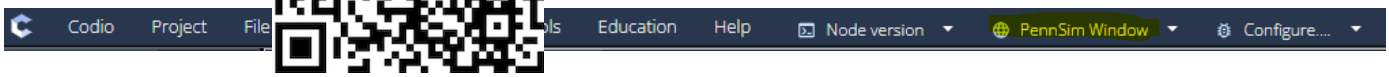
This is the section about all the steps to get into PennSim.

Be sure to open Codi[illegible]signment page in Canvas.

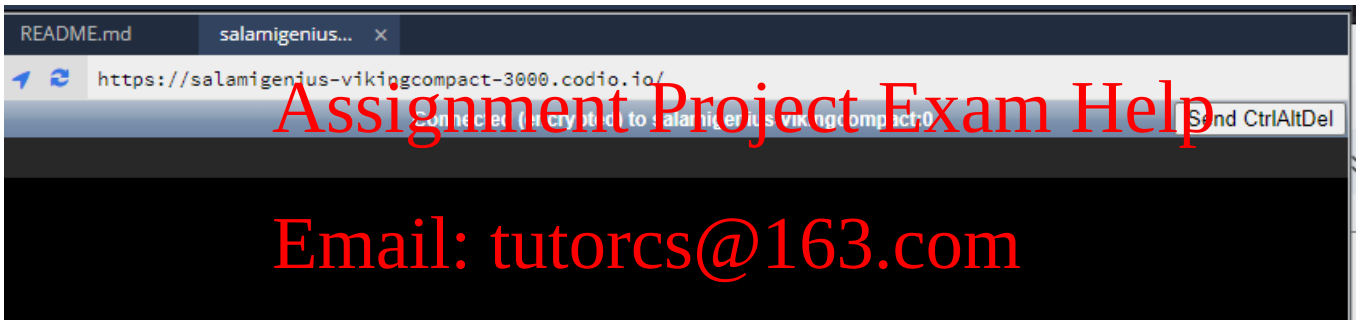
Open the PennS[illegible]

PennSim runs as a G[illegible]o open a window to display the GUI.

1. Along the top [illegible]e icon labeled as PennSim Window



2. This should open a new Codio Tab using a `fourword-randomname-3000` with a black screen. The Penn Sim Window is now open, but we still need to launch the program itself.

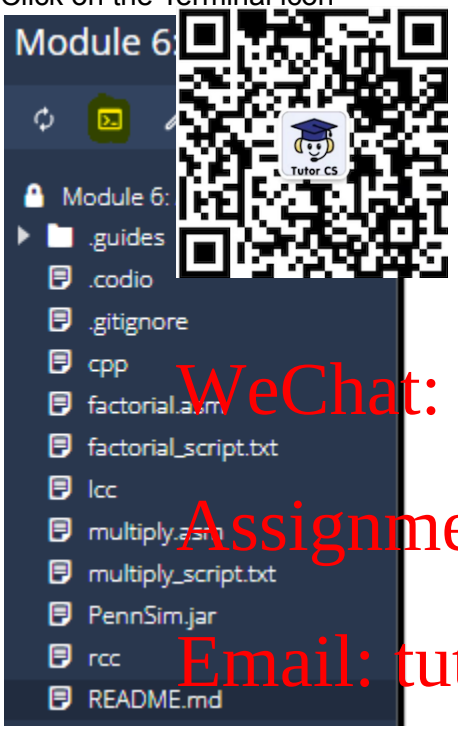


3. You may find having multiple Codio tabs cumbersome. If so, select the PennSim Window dropdown menu arrow, then choose New Browser Tab. Following steps 1 and 2 again will open PennSim Window in a new browser tab which you may find easier to navigate.

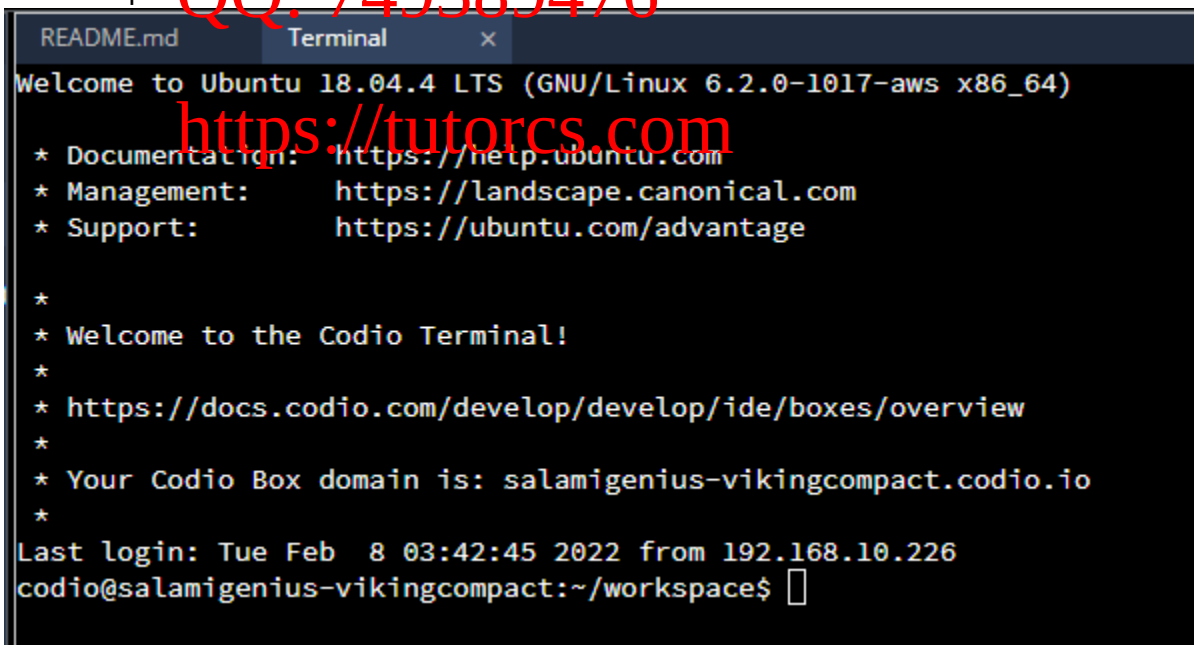
4. If you get Error 502 (Bad Gateway), contact course staff right away. Do not attempt to resolve this on your own.
5. If you get error 401 (Unauthorized), you may need to accept cookies and/or switch to Google Chrome browser. Adding [*.]codio.com as an allowed third-party for cookies should be sufficient.

Start PennSim in the Terminal Command Line

1. On the left side of the screen, you can see the File Tree. This shows all the files in Codio.
2. Click on the Terminal icon



3. This will open a Linux terminal command line in the main window.



4. Type this command to launch PennSim

```
java -jar PennSim.jar
```

5. Go to the PennSim Window you created in the previous section.

Troubleshooting

In the event the PennSim Window freezes and refreshing Codio doesn't help, go back to the terminal and do Control+C to stop the PennSim program. Then run this command in the terminal

```
pkill -f PennSim
```

Then rerun the launch command in Step 4.

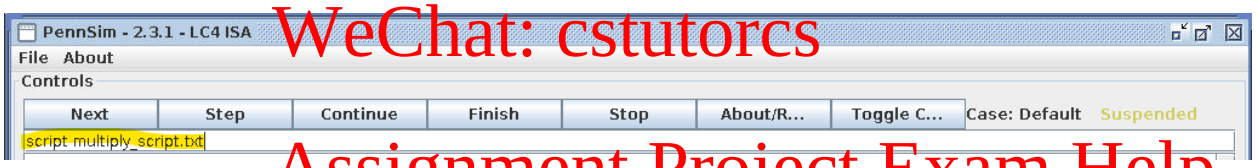
You may also attempt to go to Project->Restart Box to reboot the computer. Do not select Reset Box as this will delete your work.

If your keys appear to be stuck on CAPSLOCK even though you didn't press the button, you can try to Restart Box or use the Toggle Case button to have PennSim un-capitalize the characters.

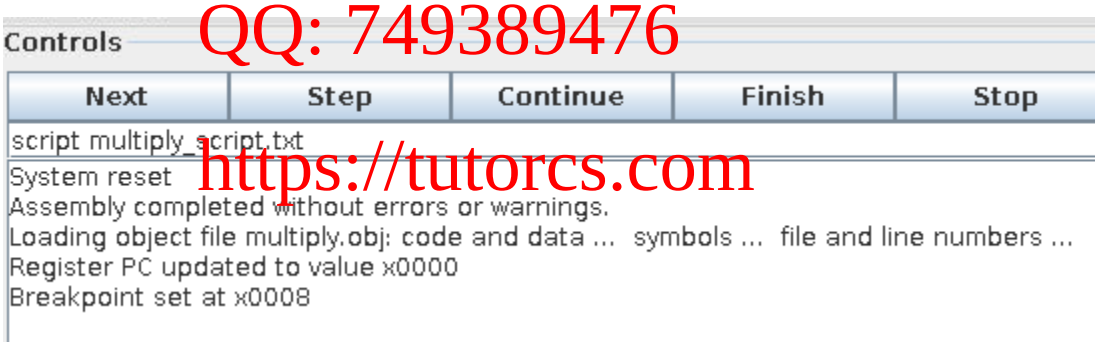
Run `multiply.asm` in PennSim

1. From the File Tree, click on `multiply.asm`. This will open a new Codio tab named `multiply.asm` containing the contents of the file.
    a. Review the contents of the file. Hopefully this looks familiar from lecture (if not, review the content).
2. From the File Tree, click on `multiply_script.txt`. This opens a new Codio tab displaying the contents of the file.
    a. This file contains the commands you must enter in PennSim to **`reset`** the simulator, load the program, and **`load`** it into memory. Be sure to understand what each command does.
3. In the PennSim window, go to the command line in the Controls section and enter

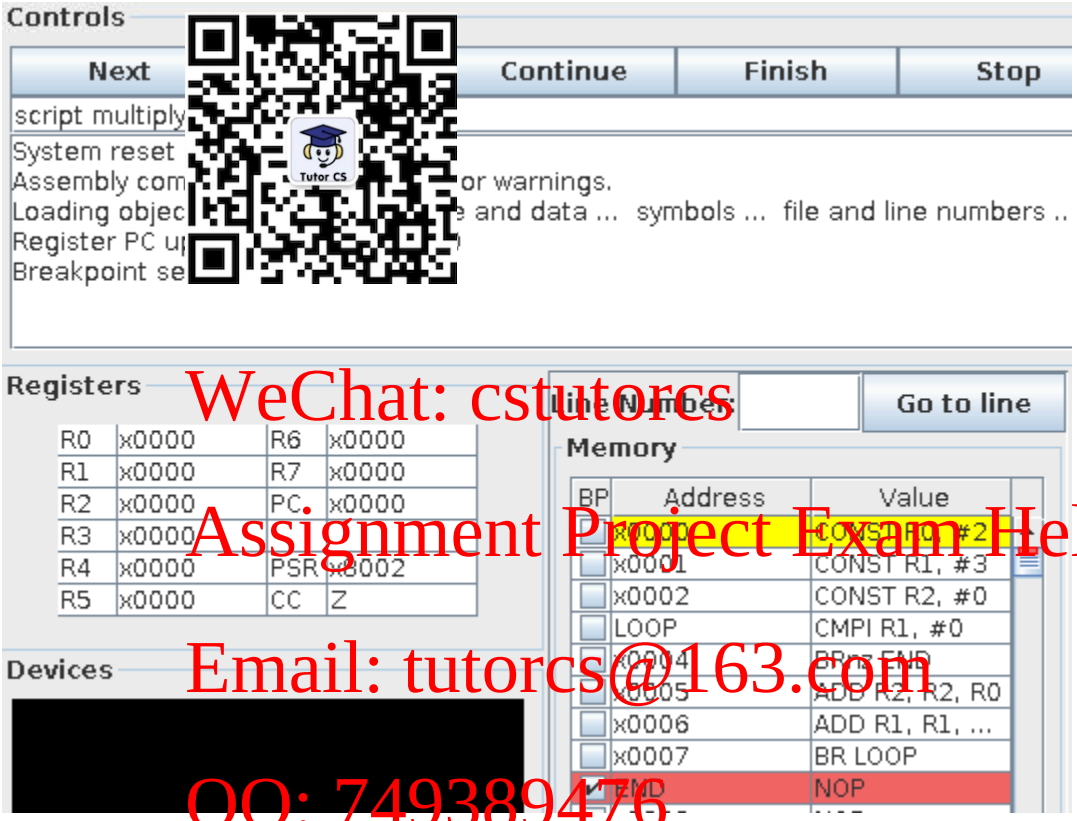
    ```
    script multiply_script.txt
    ```



4. This will run all the commands in the script line-by-line, outputting the result to the Console Log. See if you can match the commands in the script file to the messages in the Console Log output. Note how the script file automates manually typing many commands. The script files will save you a lot of manual work so you can focus on programming instead!



5. Press Step to run the program line-by-line. Carefully examine how each instruction updates the registers, PC, PSR, CC, etc, and also that the yellow highlight bar moves to each instruction.



Starter Code

We have provided some starter files. You may need to modify some files and generate completely new files to succeed in this assignment.

`factorial.asm`	- contains the pseudocode for the While Loops in Assembly mplete Part 1 here.
`factorial_script`	mple commands to assemble and load the factorial he While Loops in Assembly problem. Complete
`multiply.asm`	ram from lecture
`multiply_script.`	t file to automate commands
`PennSim.jar`	- Runs your programs, debugging, etc.

Requirements

General Requirements

- Your script files **MUST** contain all the necessary commands to assemble and load your programs, as the `multiply` example in Codio.
- Your program e requirements of the problem when loaded into PennSim and e button.
- Your program ny Exception unless otherwise noted.
- Your program LC4 Assembly.
- You **MUST** us B.
- You **MUST** comment your code so we can grade it. This will also help with partial credit.
- You **MUST** use `END` as the label that indicates the end of the program.
- You **MUST** submit to Codio and Gradescope as outlined in the Submission section.
- You **SHOULD** do all the work in Codio. Do not attempt to run these programs locally. TAs will not assist you if you are trying to do the projects outside of Codio.
    - Codio provides a standard environment to ensure consistent functionality.
    - Codio backs up your code. You can restore deleted/modified files by going to Tools->Code Playback.
    - TAs can login and view your code for asynchronous debugging.
    - Different operating systems handle endianness differently. Your submission **MUST** work in the Codio environment, which is where we will be performing all tests.

Part 1: While Loops in Assembly

- You **MUST** use the filenames `factorial.asm` and `factorial_script.txt`.
- You **MUST** implement the pseudocode algorithm as provided unless otherwise noted:
    - You **MAY** set `B` to `#1` for `A = 0`.
    - You **MAY** set `B` to `#-1` for values of `A` less than `0`
- You **MAY** hardcode `A = 5` testing purposes. We will change the value of `A` (and only A) ourselves to check your work.

Part 2: Subroutines in Assembly

- You **MUST** use the filenames `factorial_sub.asm` and `factorial_sub_script.txt`.
- You **MUST** use a subroutine to calculate the factorial and it **MUST** be called `SUB_FACTORIAL`.
- You **MUST** follow the provided pseudocode (this includes the factorial algorithm from Part 1) unless otherwise noted.
- Your subroutine **MUST** only take a single argument, `A`.
- You **MUST** determine the largest value of `A` that your subroutine can correctly calculate the factorial (call it `X` for reference)

- Your subroutine **MUST** check the value of `A`:
  - If the value of `A` is less than `0`, your subroutine **MUST** return `#-1`.
  - If the value of `A` is greater than `X`, your subroutine **MUST** return `#-1`.
  - Otherwise, your subroutine **MUST** return the factorial of `A`.
    - If `A` is `0`, your subroutine **MAY** return `#1`.
- Your subroutine **MUST** store the return value in `R1` when it returns.
- For this Part only, you **MUST** execute the subroutine again once you have returned from the subroutine.

Part 3: Working with Data Memory

- You **MUST** use the filenames `dmem_fact.asm` and `dmem_fact_script.txt`.
- You **MUST** pre-load Data Memory, starting at address `x4020`, with the following values: `#6`, `#5`, `#8`, `#10`, `#-5`
- You **MAY** assume that we will only test five values in Data Memory starting at this address.
- For each value in Data Memory, you **MUST** do the following:
  - Load Data Memory as the value of `A` to pass to your subroutine
  - Follow the same factorial requirements in Part 2. You **SHOULD NOT** change the subroutine in this problem.
  - Store the returned value into the same Data Memory address, overwriting the original value.

Extra Credit

- You **MUST** use the filenames `dmem_fact_ec.asm` and `dmem_fact_ec_script.txt`.
- You **MUST** pre-load Data Memory, starting at address `x4020`, with the following values: `#6`, `#5`, `#8`, `#10`, `#-5`
- You **MUST** modify your subroutine from Part 3 to accept an address as the input value, rather than a number to calculate.
  - Your subroutine therefore **MUST** perform the loading and storing steps.
  - You **MUST** follow the other factorial requirements in Part 2.

Suggested Approach

This is a *suggested* approach. You are not required to follow this approach as long as you follow all of the other requirements.

High Level Ov

Work on one problem

1. Be sure you u o works: where the File Tree is, how to use the Terminal to st get the desired tab configuration
2. Run the multi sense of how PennSim works.
3. Read the Pen in Canvas).
4. Get the While Loops in Assembly working. This is just a basic while loop, to introduce you to the programming construct. Be sure to test it thoroughly.
5. In Subroutines in Assembly, copy your While Loops in Assembly solution into a new file and expand it to work as a subroutine. We also introduce if/then/else block constructs here, so be sure you handle the edge cases.
6. In Working With Data Memory in Assembly, set up pre-loaded data in Data Memory, then have your program load the data and pass it to the subroutine. The subroutine does the factorial calculation and returns the result to the main program. The main program stores the calculated result back into Data Memory and does this for five different values.
7. Attempt the optional extra credit.

Great High Level Overview, but I really need a Slightly More Detailed Overview

Okay, I guess we can give some more details.

Part 1: While Lo

File Setup

You will need to mod nd `factorial_script.txt` for this problem.

The pseudo-code bel `rial.asm` starter code) describes the mathematical operati ial. When the algorithm below is completed, the variable `B` will contain `A!` of the given positive value of `A`, the factorial is defined as `5 x 4 x 3 x 2 x 1 = 120`.

Here is the pseudo-code for the factorial algorithm:


```
A = 5 ; // example to do 5!
B = A ; // B=A! when while loop completes
while (A > 1){
      A = A - 1 ;
      B = B * A ;
}
```


Your Task

Your task for this part is to implement this algorithm using LC4 Assembly. Use `R0` to hold variable `A`, and `R1` to hold variable `B`.

In your File Tree find `factorial.asm` and open it. Implement the factorial algorithm within this file. Test it using the other script file `factorial_script.txt`.

You may hardcode `A` to have the value `5` for testing purposes, but when we grade your assignment, we will try out different numbers to ensure your algorithm is working. Be certain to run your program in PennSim and make sure it is working for different values of `A`.

For this problem, we just want you to implement the while-loop algorithm as outlined in the pseudocode. But we recognize that some students are uncomfortable returning an "incorrect" result. If you've been paying attention, you may have noticed that the algorithm would return 0! = 0. Therefore, you may, if you choose, set `B = 1`, but this is completely optional and not required.

Likewise, when `A < 0`, your program just needs to end gracefully. That is, it should stop on the `END` label without throwing an Exception. You may, if you choose, setup your while loop to return `-1` to indicate "unable to calculate", but this is completely optional and not required in this problem.

In your program and in your script file be certain to set a breakpoint labeled `END`. This will ensure you program ends, instead of requiring an infinite loop to stop execution or throwing an Exception. Also, be certain to comment your code to help us understand the flow of your program as we grade.

Don't move on to Part 2 until your program calculates factorials correctly (aside from the one exceptions listed here), ends on the `END` label, and does not throw exceptions. Be sure to comment your program.

Part 2: Subroutines in Assembly

File Setup

For this problem, you'll convert your factorial program from Part 1 into a subroutine and call it using `JSR`.

You will need to crea…`torial_sub.asm` and `factorial_sub_scr`…`blem`. After you've completed Part 1, copy the file `factorial.asm` in th…licking on the file, clicking `copy` and then right-clicking once ag…nd pressing `paste`. Rename this copied file as `factorial_sub.asm`…s with `factorial_script.txt`, and rename the copy as `factorial_`…xt, you'll need to open and edit this new script file to ensure it assemble…`al_sub.asm` instead of `factorial.asm`.

Your Task

Your task for this Part is to make your While Loop into a Subroutine.

Start by adding the label `SUB_FACTORIAL` to the top of your factorial program. Remove any `CONST` instructions you may have that would set the variable `A` (register `R0`) inside your subroutine. We want `A` to be an argument to your subroutine, so its value must be set before the subroutine is called.

Inside the subroutine, replace the `END` label with a `RET` instruction. If you have done this correctly, when the subroutine returns, `B` (`R1`) holding the return value of the subroutine. Of course, your program should still end after the subroutine returns, be sure to implement that.

Above your subroutine code, implement the following pseudocode to call your subroutine:

```
MAIN
        A = 6 ; // this is the only thing a user would change
        B = sub_factorial (A) ;

        // your sub_factorial subroutine you just wrote goes here
END
```

After you return from the subroutine, make certain to "jump over" your subroutine to a new `END` label, so that your subroutine isn't executed twice! Make certain to set `END` as a breakpoint in your script file. At this point, your While Loop should be converted to a Subroutine.

Next, you will need to add some checks to test if the user has provided a value of `A` that your subroutine cannot perform the calculation correctly. First, determine `X`, the largest value of `A` that your factorial subroutine will calculate the correct factorial. Add an if/else statement to the start of your subroutine to ensure `A` is both a positive number and also is less than or equal to the largest number your assembly program can work with. If `A <= 0` OR `A > X`, then set `B = -1` and immediately return from the subroutine without attempting to find the factorial.

As a hint, first determine if we as programmers should consider `A` as signed or unsigned. This will help you correctly identify `X` for your program.

Part 3: Working with Data Memory

For this problem, you will have to review the example of working with data memory in lecture.

File setup

After you've completed Part 2, copy `factorial_sub.asm` and paste it with the new name `dmem_fact.asm`. Like`al_sub_script.txt` and paste it with the new name `dmem_fact_sc`ate `dmem_fact_script.txt` to ensure assembles and loads `dmem_fact`corial_sub.asm`.

Your Task

Use the `.FILL` direct`ows of data memory starting address `x4020` with the numbers `6`, `5`, `8`, `

Write a short assembly program that does the following for each of the five rows of data memory that you've populated:

1. load the data from Data Memory into `A`,
2. call the subroutine you've created in Part 2 on each of those rows, and
3. after the factorial subroutine returns on each row, store the calculated factorial back to the same data memory row, overwriting the original number.

The specific implementation of the `MAIN` part of your program is up to you, but you must call your subroutine for each value of `A`. As an example of how the first row of data memory should look after your program completes, address `x4020` should have the number `#720`.

Be careful with manipulating the address with `.CODE`, `.DATA`, and `.ADDR`.

Part 4: Extra Credit

For this optional part you will need to generate two more files: `dmem_fact_ec.asm` and `dmem_fact_ec_script.txt`.

Create a new program in `dmem_fact_ec.asm` that allows your subroutine `SUB_FACTORIAL` to take in a data mem *of a value* as its only argument. The new `SUB_FACTORIAL` shou e from data memory, find its factorial, and store the result back in dat without a return value. Do this for the same five values.

Submission

Where to put the files

Leave all the files in the working directory where the starter code started.

Pre-Submissio

There are no pre-sub

The Actual Sul

There are two parts t[illegible]ment: Codio and Gradescope.

Codio Submission

When you are ready (before the deadline), go to Education -> Mark Complete.

Gradescope Submission

You will need to create a single page PDF and upload it to Gradescope.

Match every question to this single page.

Do not submit a copy of your code in the PDF.

This PDF requires two things:

1. **Academic Integrity statement and signature.** You can use this as a template (an entire word document template is available on Canvas):

   Academic Integrity Agreement:

   By submitting this agreement, I certify that I have completed this homework assignment **on my own** (without collaboration with another student or unauthorized outside source) and have not **plagiarized** on this assignment (in accordance with Penn's Code of Academic Integrity).

   _____ (your name, just type it in)

2. **A screenshot of your Completed Codio workspace.** It must show:
   a. your Codio username
   b. the Module number for this assignment (do not reuse the screenshot between assignments; you will get a 0)
   c. A screenshot of an indication that the workspace is complete:
      i. The Warning that pops up after Marking Complete and typing "yes", OR
      ii. The Education dropdown menu showing that Mark as Completed is inactive (greyed out)

Grading

Main Assignment

This assignment is worth 140 points, which will be normalized to 100% for gradebook purposes.

All problems have pa

20 points for script fil mments

40 points for While L blem

40 points for Subrout blem

40 points for Working roblem

Do note that we are only grading for correctness, not efficiency.

Extra Credit

The extra credit is worth 5 percentage points. The maximum score on this assignment is 105%. As usual, the extra credit does not have partial credit.

An Important Note of Plagiarism

- We will scan your assignment files for plagiarism using an automatic plagiarism detection tool.

- If you are unaware of the plagiarism policy, make certain to check the syllabus to see the possible repercussions of submitting plagiarized work (or letting someone submit yours).