CSE12-02 Lab 3: Looping with RISC-V Assembly

Due Friday, May 24, 2024, 11:59 PM

Minimum Submission Requirements

Ensure that your Gradescope submission contains the student-completed file:

- Lab3.asm

Objective

This lab will introduce you to the RISC-V assembly language programming using RARS. You will write a program with nested loops in the provided file *Lab3.asm* to write a specific pattern to a file by the name of *lab3_output.txt* (which is generated by running your *Lab3.asm* file and does not exist prior to this action)

Make sure you properly follow the steps mentioned in the RARS site to download and install RARS on your machine. Please approach a TA or tutor in your lab section if you have issues installing RARS on your machine!

Since this is your very first foray into assembly programming, please read this document thoroughly without skipping any sections!

Resources

Much like how a high-level program has a specific file extension (.c for C, .py for Python) RARS RARS-based RISC-V programs have an .asm extension.

In the Lab3 folder in the course Google Slide, you will see 7 assembly files. They are meant to be read *(and understood*) **in sequence**:

1. *firstRARSprogram.asm* – This program just prints "Hello World" on the output console.

2. *add.asm* – This program accepts two integers as user inputs and prints their addition result on the output console.

3. *multiply.asm* – This program accepts two integers as user inputs and prints their multiplication result on the output console. This file specifically shows you how to do loops in assembly

4. *fileWriteDemo.asm* – This program creates a file for write, and then writes a line of text into the file. Basically, it shows the basics of writing a file in RARS.

5. *fileReadDemo.asm* – This program reads the file generated by the above .asm file and reads out its contents on the output console. Basically, it shows the basics of doing a file Read in RARS.

6. *patternDisplayDemo.asm* – This program inputs a number ***n*** from the user and generates a "* * * ….." Pattern depending on the value of ***n***. This pattern is written into a file. Understanding how this code works will help you in writing the pattern required for this lab assignment into a file as well.

7. *Lab3.asm* – Some **starter code** has already been provided in this file which you will need to complete and then submit to your Gradescope autograding portal. The starter code mainly revolves around macros that have been created for your benefit to create and write to the file *lab3_output.txt*.

Please download these files and make sure to open them in **RARS Text Editor only**. Else the comments and other important code sections won't be properly highlighted and can be a hindrance to learning assembly language intuitively. Steps on opening, assembling, and running a .asm file are provided later in this document.

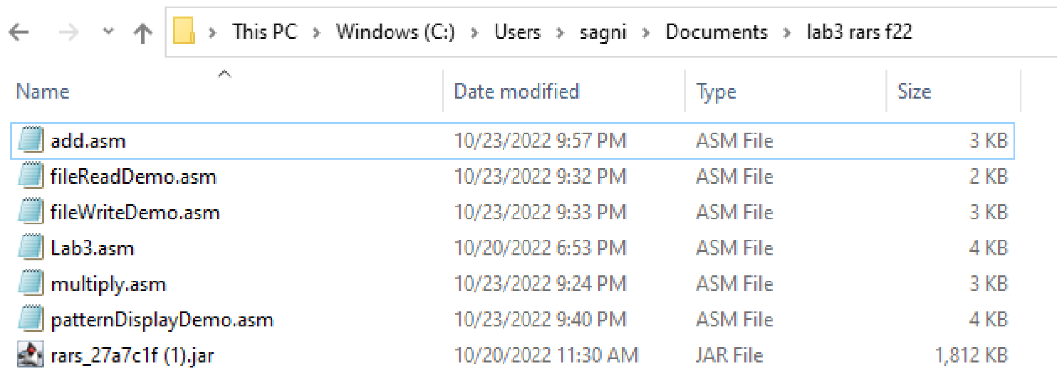
These 7 files have enough comments in the source code to jumpstart your understanding of RISC-V assembly programming if the lectures have not yet covered certain topics in assembly programming.

Beyond these three files, ***you should have all the required resources in the Lecture Slides themselves, in the ppt files which have the title slide "Von Neuman and RISC- V". These slides are very self-explanatory and it is encouraged you start reading them even if the instructor hasn't started discussing them in lecture.***

For the usage of macros (which are utilized heavily in this lab to generate ecalls), please also refer to the RARS documentation on macros and ecalls as well. For lab 3, ***you don't even need to know what the inside of a macro block looks like so long as you know just what it is supposed to do overall.***

Working in RARS

Helpful tip: For lab3 and lab4, it is very useful to create two separate folders in your machine, lab3 and lab4. Make each folder the workspace for your respective lab. So, for the given lab, place the provided three
.asm files in the Lab3 folder along with a copy of the .jar RARS application file. This is where you will create your Lab3.asm file as well.

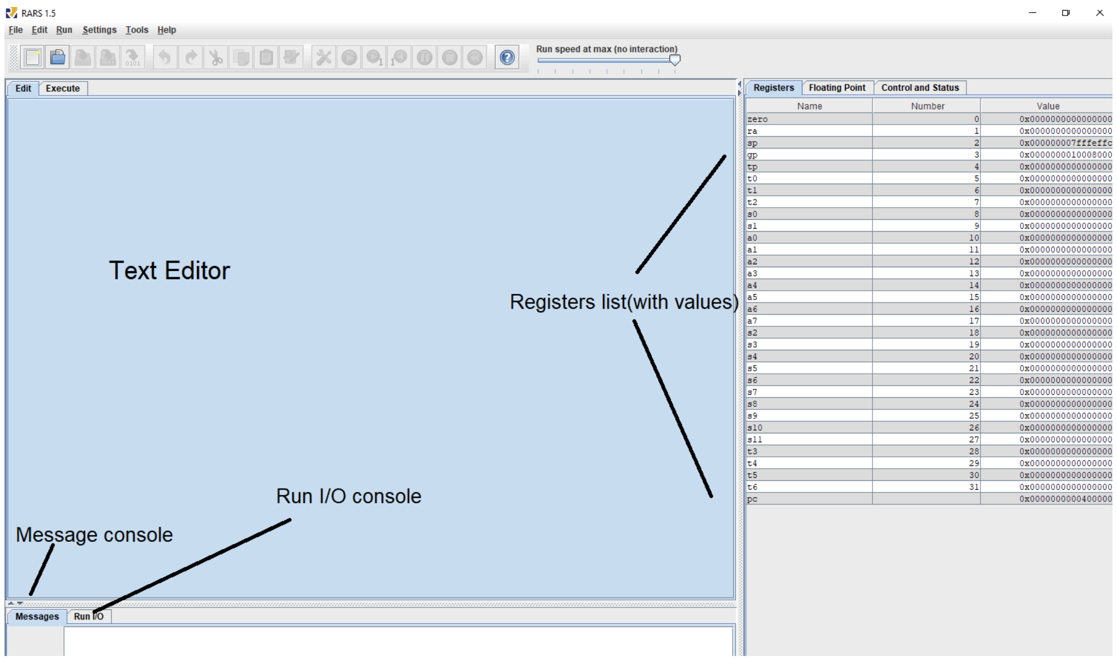

This PC › Windows (C:) › Users › sagni › Documents › lab3 rars f22

Name	Date modified	Type	Size
add.asm	10/23/2022 9:57 PM	ASM File	3 KB
fileReadDemo.asm	10/23/2022 9:32 PM	ASM File	2 KB
fileWriteDemo.asm	10/23/2022 9:33 PM	ASM File	3 KB
Lab3.asm	10/20/2022 6:53 PM	ASM File	4 KB
multiply.asm	10/23/2022 9:24 PM	ASM File	3 KB
patternDisplayDemo.asm	10/23/2022 9:40 PM	ASM File	4 KB
rars_27a7c1f (1).jar	10/20/2022 11:30 AM	JAR File	1,812 KB

Figure 1 Ideal workspace setup for lab3/lab4

Henceforth, run all .asm files pertinent to Lab3 on this local copy of the .jar RARS application. For reference, see my Lab3 workspace below on Windows

Open the RARS application. You should get the window below.

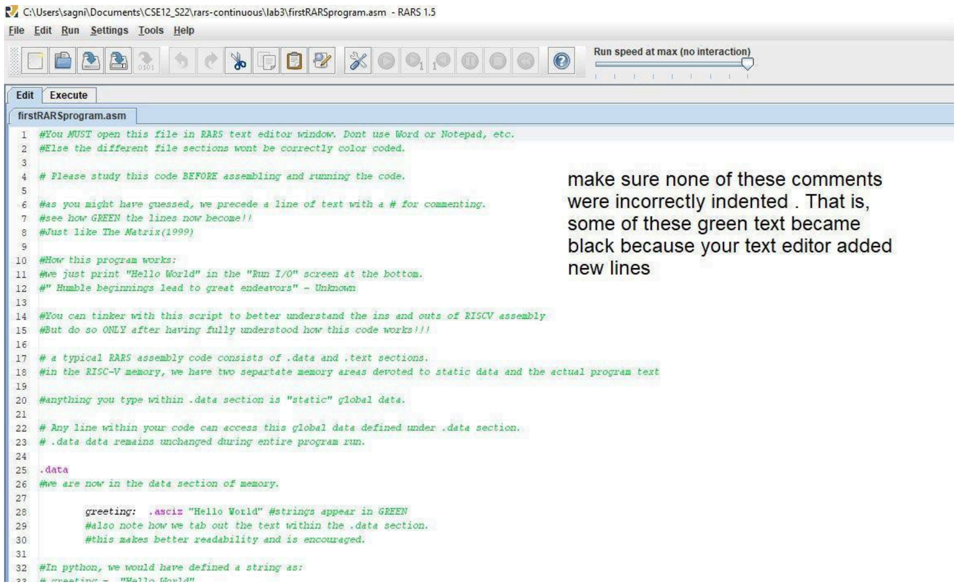

RARS 1.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Registers Floating Point Control and Status
Text Editor
Registers list(with values)
Run I/O console
Message console
Messages Run I/O

Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x000000007fffeffc
gp	3	0x0000000010008000
tp	4	0x0000000000000000
t0	5	0x0000000000000000
t1	6	0x0000000000000000
t2	7	0x0000000000000000
s0	8	0x0000000000000000
s1	9	0x0000000000000000
a0	10	0x0000000000000000
a1	11	0x0000000000000000
a2	12	0x0000000000000000
a3	13	0x0000000000000000
a4	14	0x0000000000000000
a5	15	0x0000000000000000
a6	16	0x0000000000000000
a7	17	0x0000000000000000
s2	18	0x0000000000000000
s3	19	0x0000000000000000
s4	20	0x0000000000000000
s5	21	0x0000000000000000
s6	22	0x0000000000000000
s7	23	0x0000000000000000
s8	24	0x0000000000000000
s9	25	0x0000000000000000
s10	26	0x0000000000000000
s11	27	0x0000000000000000
t3	28	0x0000000000000000
t4	29	0x0000000000000000
t5	30	0x0000000000000000
t6	31	0x0000000000000000
pc		0x0000000000400000

Figure 2 Opening the RARS application

Let us open firstRARSprogram.asm by clicking File -> Open.

Make sure the comments (which appear in green) are properly indented and haven't been misaligned when you download the file from Google Drive. They should appear as shown below:

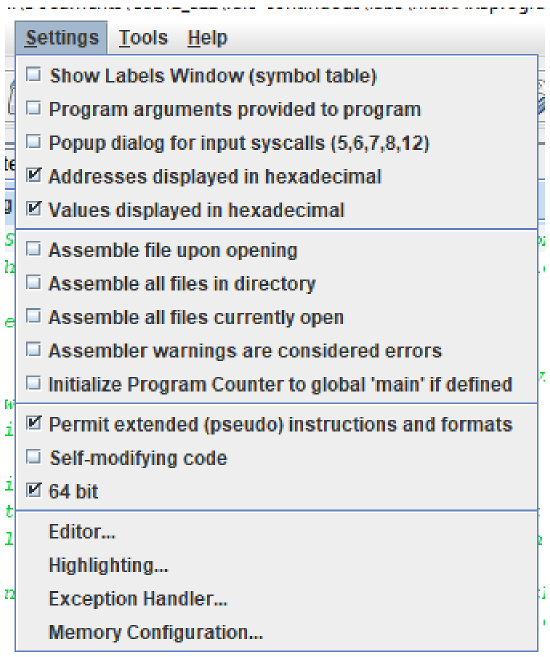

C:\Users\sagni\Documents\CSE12_S22\rars-continuous\lab3\firstRARSprogram.asm - RARS 1.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
firstRARSprogram.asm
make sure none of these comments were incorrectly indented . That is, some of these green text became black because your text editor added new lines

```
#You MUST open this file in RARS text editor window. Dont use Word or Notepad, etc.
#Else the different file sections wont be correctly color coded.

# Please study this code BEFORE assembling and running the code.

#as you might have guessed, we precede a line of text with a # for commenting.
#see how GREEN the lines now become!!
#Just like The Matrix(1999)

#How this program works:
#we just print "Hello World" in the "Run I/O" screen at the bottom.
#" Humble beginnings lead to great endeavors" - Unknown

#You can tinker with this script to better understand the ins and outs of RISCV assembly
#But do so ONLY after having fully understood how this code works!!!

# a typical RARS assembly code consists of .data and .text sections.
#in the RISC-V memory, we have two separtate memory areas devoted to static data and the actual program text

#anything you type within .data section is "static" global data.

# Any line within your code can access this global data defined under .data section.
# .data data remains unchanged during entire program run.

.data
#we are now in the data section of memory.

        greeting:  .asciz "Hello World" #strings appear in GREEN
        #also note how we tab out the text within the .data section.
        #this makes better readability and is encouraged.

#In python, we would have defined a string as:
# greeting =  "Hello World"
```

Figure 3 Opening an asm file on RARS

Make sure to thoroughly read the entire contents of this file in the text editor. Verbose comments have been provided to guide you along in explaining each step in the source code. This will be the norm for the other .asm files in the Lab3 folder in Google Drive as well.

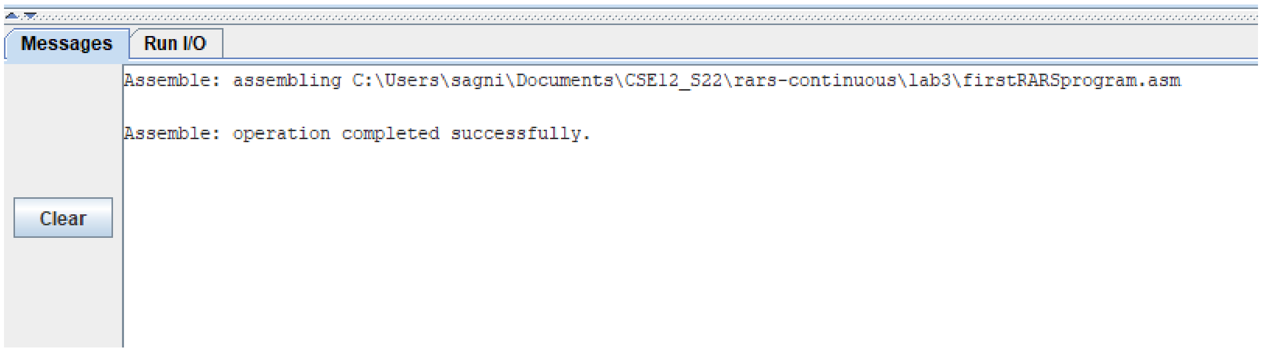
After you have read and understood the source code, it is time to assemble the program. Before you assemble, go to Settings and make sure you have the exact following options checked (or unchecked). For this course, you are allowed to use pseudo instructions. Pseudo instructions are those instructions that are not native to the RISC-V instruction set but the RARS environment has defined these new ones by a combination of actual RISC-V instructions. Permit pseudo instructions (this actually makes coding easier for you) and **64-bit operation**. **This should be done for every RARS code in CSE12!**



- [ ] Show Labels Window (symbol table)
- [ ] Program arguments provided to program
- [ ] Popup dialog for input syscalls (5,6,7,8,12)
- [✓] Addresses displayed in hexadecimal
- [✓] Values displayed in hexadecimal
- [ ] Assemble file upon opening
- [ ] Assemble all files in directory
- [ ] Assemble all files currently open
- [ ] Assembler warnings are considered errors
- [ ] Initialize Program Counter to global 'main' if defined
- [✓] Permit extended (pseudo) instructions and formats
- [ ] Self-modifying code
- [✓] 64 bit

Figure 4 RARS setting

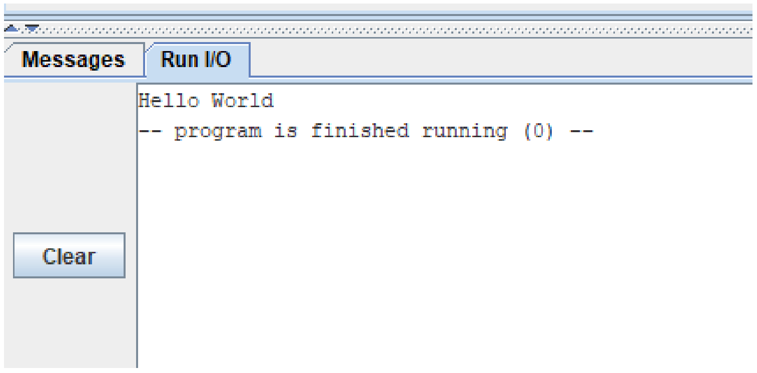
Now click on Assemble (the Wrench and screwdriver icon). If correctly assembled, your Messages window should show the following information:



```
Assemble: assembling C:\Users\sagni\Documents\CSE12_S22\rars-continuous\lab3\firstRARSprogram.asm

Assemble: operation completed successfully.
```

Figure 5 Successful assembly

Now Click on the Run button to Run the program. You will get the following output:

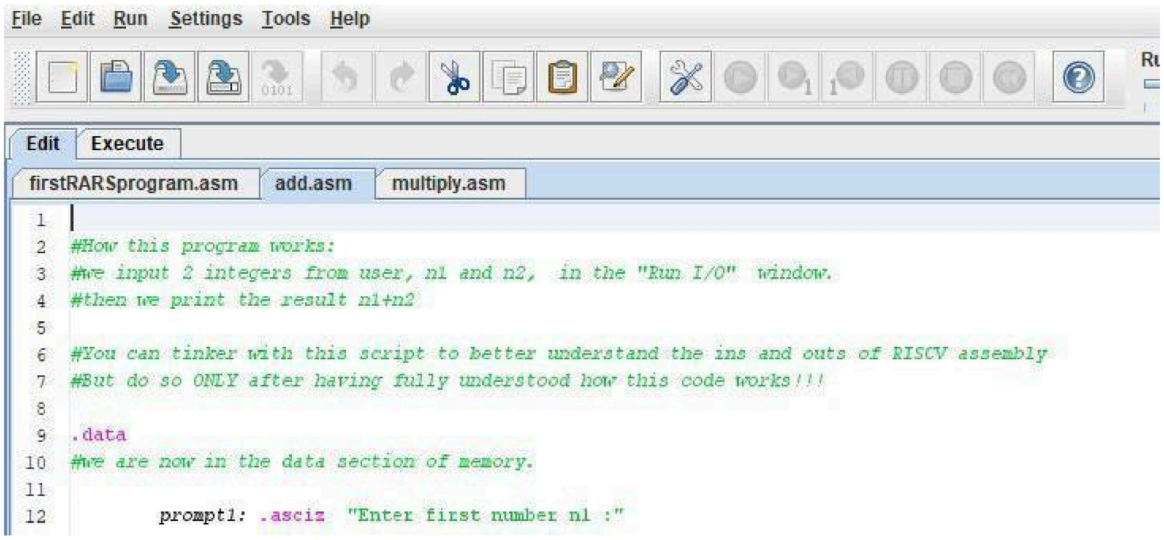


```
Hello World
-- program is finished running (0) --
```

Figure 6 Successful Runtime

Now try running the other .asm files.

One word of caution when your text editor contains multiple opened files is to make sure of assembling the correct assembly file. For example, in the window below, multiple files are open and I want to only assemble and run add.asm. Then my tab for add.asm should be highlighted as shown below. Only then can I click Assemble, then Run.



```

#How this program works:
#we input 2 integers from user, n1 and n2,  in the "Run I/O"  window.
#then we print the result n1+n2

#You can tinker with this script to better understand the ins and outs of RISCV assembly
#But do so ONLY after having fully understood how this code works!!!

.data
#we are now in the data section of memory.

        prompt1: .asciz  "Enter first number n1 :"
```

Figure 7 Multiple tabs open

A helpful feature in RARS

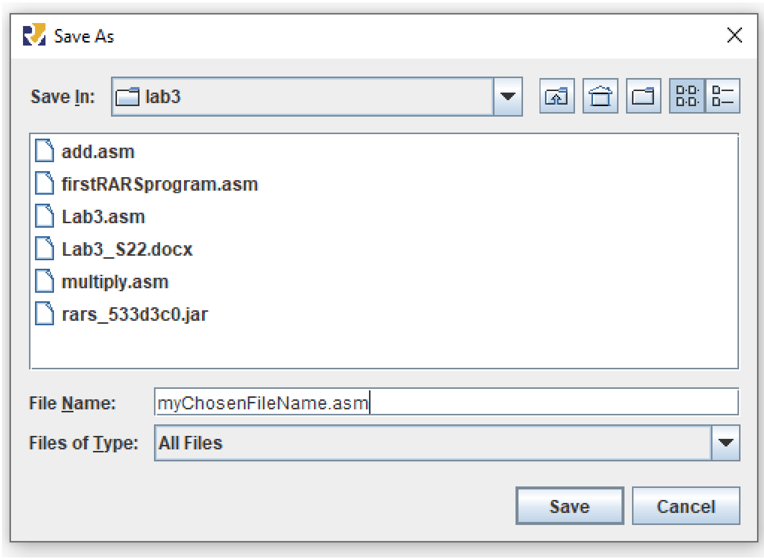
RARS has a helpful feature where instead of Running the entire program at once, you can **Run One Step At A Time**. The corresponding button is beside the Run button. This allows you to sequentially execute each line of code and see how it affects the values of the Registers as they appear to the right of your screen.

Regarding Macros

The file *multiply.asm* makes extensive use of macros to help create a more readable main program section (Instructions on how to use macros are provided in the file comments). So does the source code in the files *fileWriteDemo.asm*, *fileReadDemo.asm and patternDisplayDemo.asm* (we will discuss more on the aspect of file reads and writes that these .asm files do shortly). Based on how we define a macro in the source code, it is tempting to confuse it with a function. However**,** ***macros are NOT functions***! Whenever you place multiple instances of the same macro in your code, you are copying the macro's contents in those code areas the same number of times.

Naming new files in RARS

When you want to open a new file on RARS, go to File->New. The default file name riscv1.asm shows up on the tab. When you save this file, you MUST make sure that you are explicitly defining the correct extension(.asm) as shown below.



File Name: myChosenFileName.asm
Files of Type: All Files

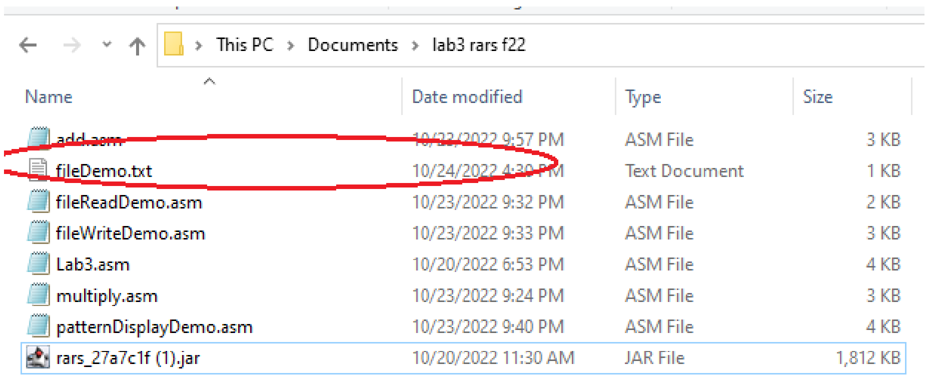
Figure 8 Saving a new file in RARS

About file reads and writes in RARS

File creation and manipulation is a very common part of the learning curve whenever you learn a new high-level programming language, be it C or Python. For lab 3, we will be writing the display pattern to a file so that it is more convenient for the auto grader. The auto grader would do a file text equality check between the file generated by your lab3 source code and the one in its deck and accordingly provide you points (or not!).

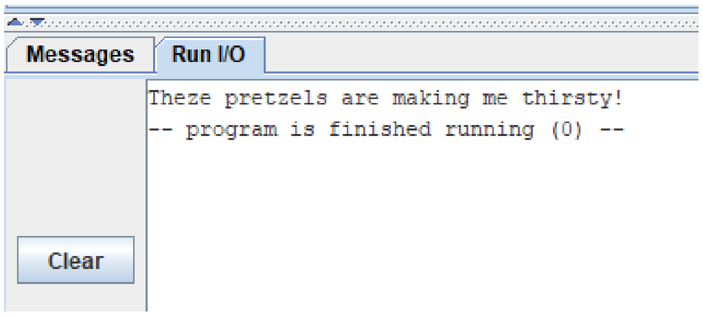
To give you a demo, we have two reference assembly source code files: *fileWriteDemo.asm* and *fileReadDemo.asm.* The former file creates a file with the name *fileDemo.txt.* The following text is written into *fileDemo.txt*: "These pretzels are making me thirsty!". The latter file *fileReadDemo.asm* contains code to open *fileDemo.txt* and extract this text to display on the output console of RARS.

The following two images show the results of having run *fileWriteDemo.asm* and then *fileReadDemo.asm.*



Name	Date modified	Type	Size
add.asm	10/23/2022 9:57 PM	ASM File	3 KB
fileDemo.txt	10/24/2022 4:30 PM	Text Document	1 KB
fileReadDemo.asm	10/23/2022 9:32 PM	ASM File	2 KB
fileWriteDemo.asm	10/23/2022 9:33 PM	ASM File	3 KB
Lab3.asm	10/20/2022 6:53 PM	ASM File	4 KB
multiply.asm	10/23/2022 9:24 PM	ASM File	3 KB
patternDisplayDemo.asm	10/23/2022 9:40 PM	ASM File	4 KB
rars_27a7c1f (1).jar	10/20/2022 11:30 AM	JAR File	1,812 KB

Figure 9 A new file generated in my workspace after running fileWriteDemo.asm. Note the file size to be shown as 1KB despite us having written only 38 bytes of data into it. That is because a file also contains metadata and a header generated by your OS as well.



```
Theze pretzels are making me thirsty!
-- program is finished running (0) --
```

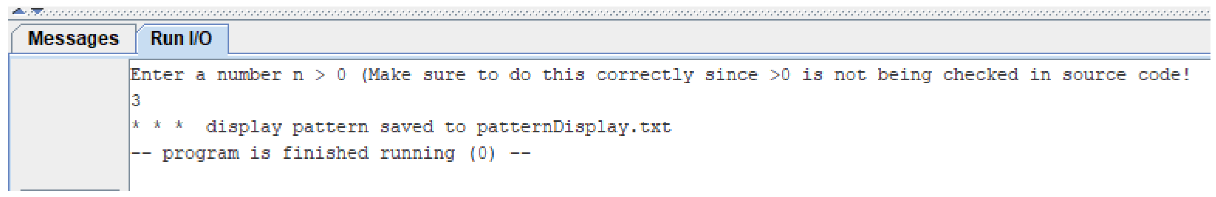
Figure 10 RARS output console after running fileReadDemo.asm

Both *fileWriteDemo.asm* and *fileReadDemo.asm* use many macros within the source code to make the main .text section of the code more programmer friendly in terms of writing. *For the purposes of lab 3, you **DO NOT** need to understand **WHAT** these macros are doing within their definition block.* It suffices to know simply what the result of executing a macro in your source code simply does. However, understanding the macros does help to build your foundation in RARS programming well.

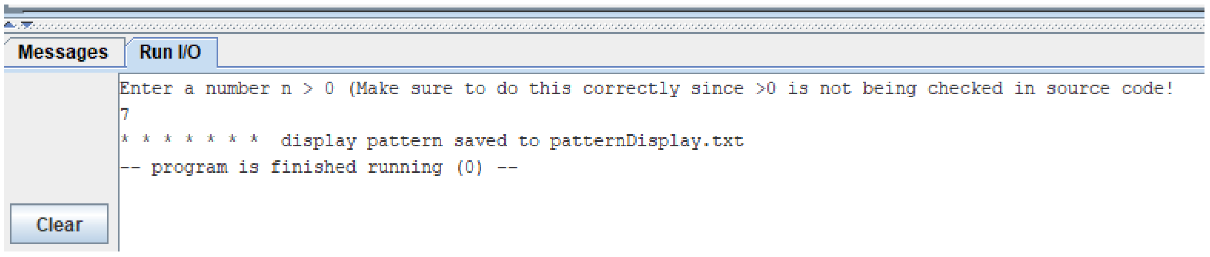
One thing to note is that since lab3 does not focus on proper function coding in RISC-V assembly, it can get very difficult to keep track of random unintentional instances of your registers to change value. For instance, in C or Python, you can define a variable *temp,* assign it a specific value, and rest assured that this variable does not change from the assigned value during code compilation or runtime unless explicitly told to. However, in a large source code in assembly, working with a limited number of registers means that it is very difficult to keep track of each individual register value unless you are very careful.

We will deal with register preservation in lab4 but in lab3, you will only be asked to ensure that **you do not use specific registers in your *Lab3.asm* source code. The list of these taboo registers will be highlighted in the section later on Lab3**

Besides the aforementioned 2 files related to file writing and reading, we also have a 3rd .asm file, *patternDisplayDemo.asm.* The source code in this file, once run, asks as input an integer n and then prints the pattern "* " n a number of times horizontally.



```
Enter a number n > 0 (Make sure to do this correctly since >0 is not being checked in source code!
3
* * *  display pattern saved to patternDisplay.txt
-- program is finished running (0) --
```



```
Enter a number n > 0 (Make sure to do this correctly since >0 is not being checked in source code!
7
* * * * * * *  display pattern saved to patternDisplay.txt
-- program is finished running (0) --
```

Figure 11 Output console after running patternDisplayDemo.asm for user input n=3 and 7. In both cases, make sure to check the contents of the created file patternDisplay.txt as well

Similar to *patternDisplayDemo.asm,* Lab3.asm will also make use of loops (nested loops to be precise) to generate a pattern based on the value of n inputted by the user. Thus, you should thoroughly read and understand the working of source code in *patternDisplayDemo.asm.*

Lab3 Programming Assignment

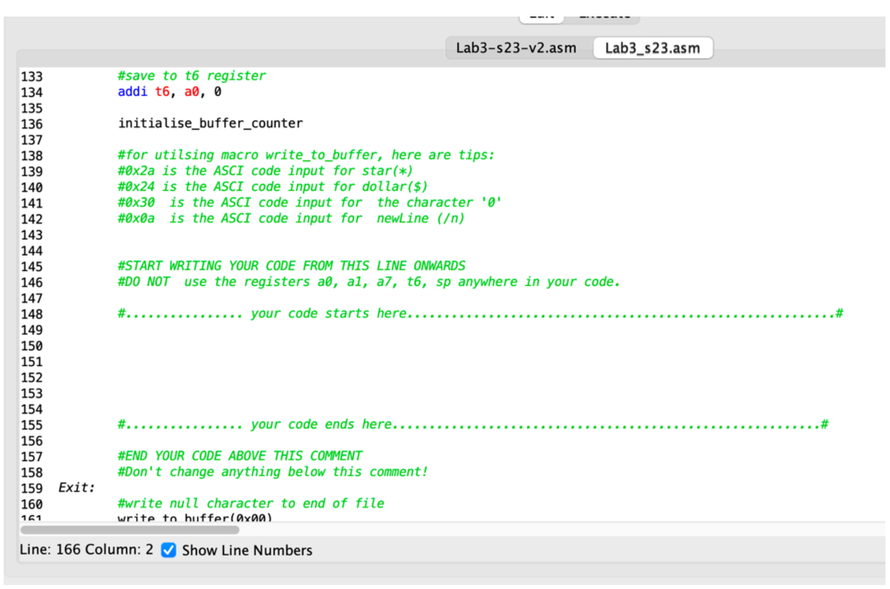
This program will print out a pattern with stars (asterisks, ASCII hex code 0x2a) dollar sign (ASCII hex code 0x24), and the character ‘0’(ASCII hex code 0x30).

1. It will first prompt for the height of the pattern (i.e., the number of rows in the pattern). If the user enters an invalid input such as a negative number or zero, an error message will be printed and the user will be prompted again. These error messages are listed in the starter code in the Lab3.asm file provided.

2. Then your program should generate the following pattern, a "tattooed right-angled triangle", using stars, dollars, ’0’, and newline (ascii hex code 0x0a) characters. Refer to test cases in the test cases sub folder in the Lab3 folder for examples.

3. This entire pattern generated from the user input of n is written to the file *lab3_output.txt*.

The actual task of opening the file *lab3_output.txt* and writing the contents to it is borne by macros used in the starter code included in the *Lab3.asm* file. Consider the screenshot of Lab3.asm file below:


Lab3-s23-v2.asm Lab3_s23.asm

```
        #save to t6 register
        addi t6, a0, 0

        initialise_buffer_counter

        #for utilsing macro write_to_buffer, here are tips:
        #0x2a is the ASCI code input for star(*)
        #0x24 is the ASCI code input for dollar($)
        #0x30  is the ASCI code input for  the character '0'
        #0x0a  is the ASCI code input for  newLine (/n)


        #START WRITING YOUR CODE FROM THIS LINE ONWARDS
        #DO NOT  use the registers a0, a1, a7, t6, sp anywhere in your code.

        #................ your code starts here..........................................................#






        #................ your code ends here..........................................................#

        #END YOUR CODE ABOVE THIS COMMENT
        #Don't change anything below this comment!
Exit:
        #write null character to end of file
        write_to_buffer(0x00)
```

Line: 166 Column: 2 ☑ Show Line Numbers

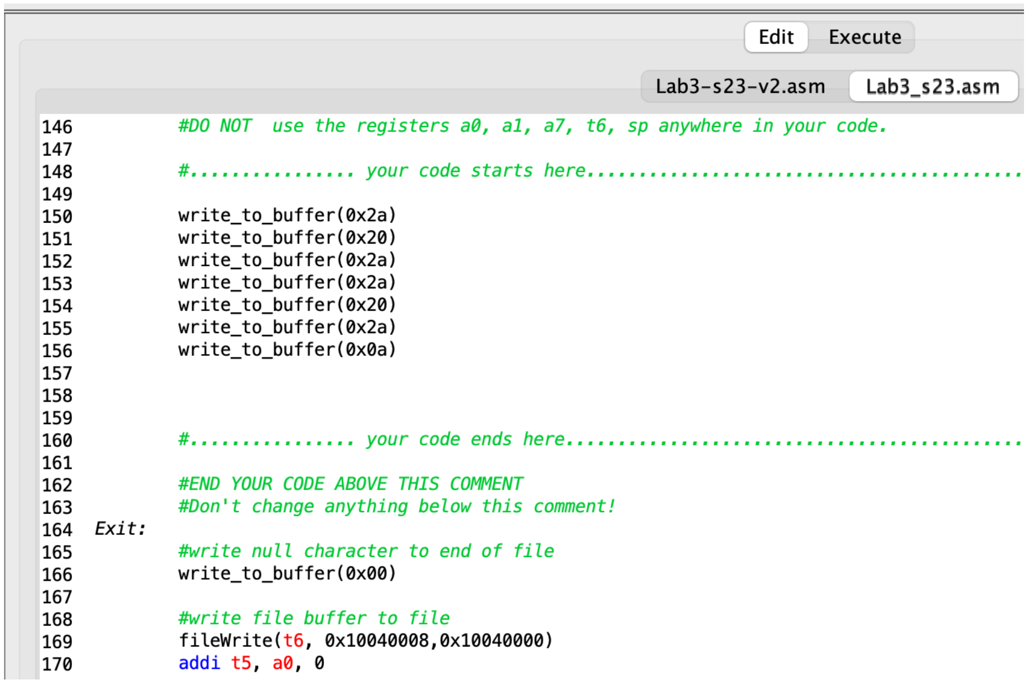
Figure 12 Lab3.asm screenshot

As you can see, you should write down your code ONLY within the area indicated by the comments.

The way this code works regarding file manipulation is as follows:

When a file is created in RARS, it is assigned a file descriptor ID, an integer number. So future references to this file through macros is only by referring to this file descriptor ID number. Once we create a file, we first need to set aside memory space within our RISC-V memory where data to be written to the file is kept. This space we call the memory buffer. In Lab3.asm, we have defined the memory space starting from address 0x10040000 as our internal memory buffer. Specifically, we hold a double word (64 bits) at this address which keeps track of how many bytes we intend to finally write to the file. 0x10040008 onwards, we start collecting the bytes that will be written into the file *lab3_output.txt.*

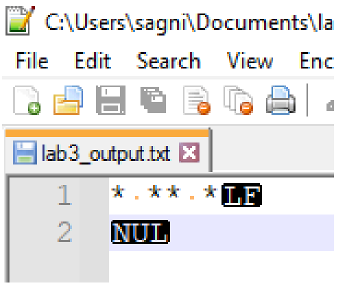
In your student code, you can update the file buffer with any character with the macro write_to_buffer. For example, I want to write the character sequence "*<blank space>**<blank space>*\n" to my memory buffer within Lab3.asm's student code section. Then I would need to write the following student code as shown next:



```
        #DO NOT  use the registers a0, a1, a7, t6, sp anywhere in your code.

        #................ your code starts here........................................

        write_to_buffer(0x2a)
        write_to_buffer(0x20)
        write_to_buffer(0x2a)
        write_to_buffer(0x2a)
        write_to_buffer(0x20)
        write_to_buffer(0x2a)
        write_to_buffer(0x0a)



        #................ your code ends here..........................................

        #END YOUR CODE ABOVE THIS COMMENT
        #Don't change anything below this comment!
Exit:
        #write null character to end of file
        write_to_buffer(0x00)

        #write file buffer to file
        fileWrite(t6, 0x10040008,0x10040000)
        addi t5, a0, 0
```

Figure 13 Modified Lab3.asm screenshot

Run this Lab3.asm file and open the generated *lab3_output.txt* in a text editor. Specifically, if you are using Notepad++(which is strongly recommended), make sure to apply the setting: ***View->Show Symbol ->Show All Characters***. This will make characters such as null and newline visible.

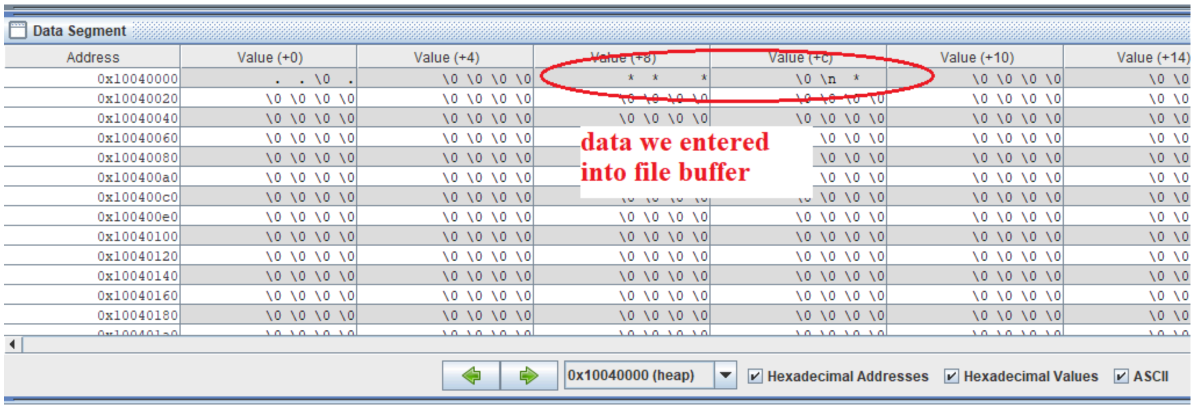


```
1   * . * * . * LF
2   NUL
```

Figure 14 lab3_output.txt screenshot from running modified Lab3.asm

As you can see, the blank space appears as an orange dot, newline as LF (Line Feed), and null as NUL.

You can see these characters as they reside in the file buffer in memory too on RARS as shown below. If you go to the execute window after running this modified Lab3.asm, selecting 0x1004000 for view, and enabling ASCII view, you will get the following screenshot:



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x10040000	. . \0 .	\0 \0 \0 \0	* * *	\0 \n *	\0 \0 \0 \0	\0 \0
0x10040020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040060	\0 \0 \0 \0	\0 \0 \0 \0	[illegible]	\0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040080	\0 \0 \0 \0	\0 \0 \0 \0	[illegible]	\0 \0 \0	\0 \0 \0 \0	\0 \0
0x100400a0	\0 \0 \0 \0	\0 \0 \0 \0	[illegible]	\0 \0 \0	\0 \0 \0 \0	\0 \0
0x100400c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x100400e0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040100	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040120	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040140	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040160	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040180	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0

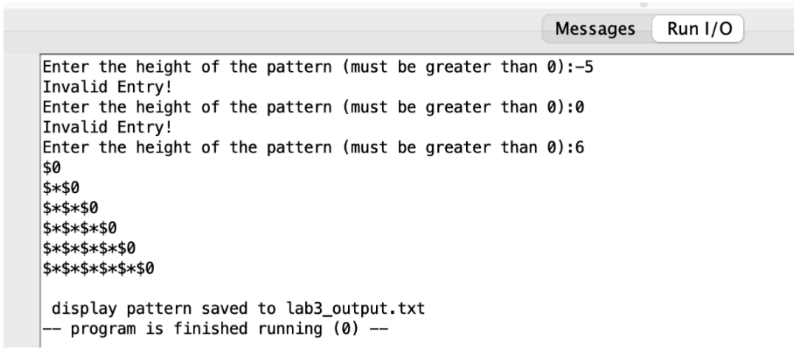
Figure 15 " ** *\n" data as it resides in our file buffer*

Note that within each individual cell in the Data Segment matrix above, we should read the bytes from right to left.

NOTE: For your student starter code, you MUST NOT use any of the following registers: a0, a1, a7, t6, sp. Using the registers t0 through t5 should be enough for the Lab3 assignment.

Example of running the actual correct Lab3.asm source code

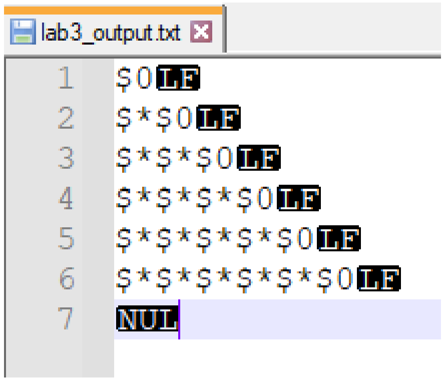
The following is a screenshot showing the runtime of the actual solved Lab3.asm code:



```
Enter the height of the pattern (must be greater than 0):-5
Invalid Entry!
Enter the height of the pattern (must be greater than 0):0
Invalid Entry!
Enter the height of the pattern (must be greater than 0):6
$0
$*$0
$*$*$0
$*$*$*$0
$*$*$*$*$0
$*$*$*$*$*$0

 display pattern saved to lab3_output.txt
-- program is finished running (0) --
```

Figure 16 Solved Lab3.asm runtime demo

When we open the generated *lab3_output.txt* file, you should get :



```
$0LF
$*$0LF
$*$*$0LF
$*$*$*$0LF
$*$*$*$*$0LF
$*$*$*$*$*$0LF
NUL
```

Figure 17 lab3_output.txt screenshot

Your student code MUST display the prompts and error messages in response to user input EXACTLY as shown in Figure 16. Please make use of the provided strings in the .data section of the starter code in Lab3.asm to make sure you do not use any other type of sentence!

NOTE: Although you are not required to print each row in the pattern on your output console, doing so (as shown in Figure 16) will greatly help in the real-time code debugging. So, it is strongly advised to do so.

Test Cases

The Lab3 folder in Google Drive contains some test cases in the test cases subfolder for the case when user input was n=1, 3, 6, 9, 30. Make sure your code output generates the exact same alignment of characters as provided there for the corresponding n input in your student code.

Automation

Note that our grading script is automated, so it is imperative that your program's output matches the specification exactly. The output that deviates from the spec will cause a point deduction.

Files to be submitted to your Lab3 Gradescope portal

Lab3.asm

-This file contains your pseudocode and assembly code. Include a header comment as indicated in the documentation guidelines here.

A Note About Academic Integrity

This is the lab assignment where most students start to get flagged for cheating. Please review the pamphlet on Academic Dishonesty and look at the examples in the first lecture for acceptable and unacceptable collaboration.

You should be doing this assignment completely all by yourself!

Grading Rubric (100 points total)

The following rubric applies provided you have fulfilled all criteria in **Minimum Submission Requirements**. Failing any criteria listed in that section would result in an automatic grade of zero ***which cannot be legible for applying for a regrade request.***

20 pt Lab3.asm ***assembles*** without errors (so *even if you submit Lab3.asm having written absolutely no student code, you would still get 20 pts!*)

80 pt output in file *lab3_output.txt* matches the specification:

20 pt error check zero and negative heights ***using the convention shown in*** Figure 16

20 pt prompts the user until a correct input is entered **as shown in** Figure 16

20 pt number of rows match user input (i.e. if n=6, the pattern would have 6 row

20 pt correct sequence of stars and dollars and '0's on each row

Legal Notice

All course materials and relevant files located in the Lab3 folder in the course Google Drive must not be shared by the students outside of the course curriculum on any type of public domain site or for financial gain. Thus, if any of the Lab3 documents is found in any type of publicly available site (e.g., GitHub, stack Exchange), or for monetary gain (e.g., Chegg), then the original poster will be cited for misusing CSE12 course-based content and will be reported to UCSC for academic dishonesty.

In the case of sites such as Chegg.com, we have been able to locate course material shared by a previous quarter student. Chegg cooperated with us by providing the student's contact details, which was sufficient proof of the student's misconduct leading to an automatic failing grade in the course.