Task4:
ARM assembler

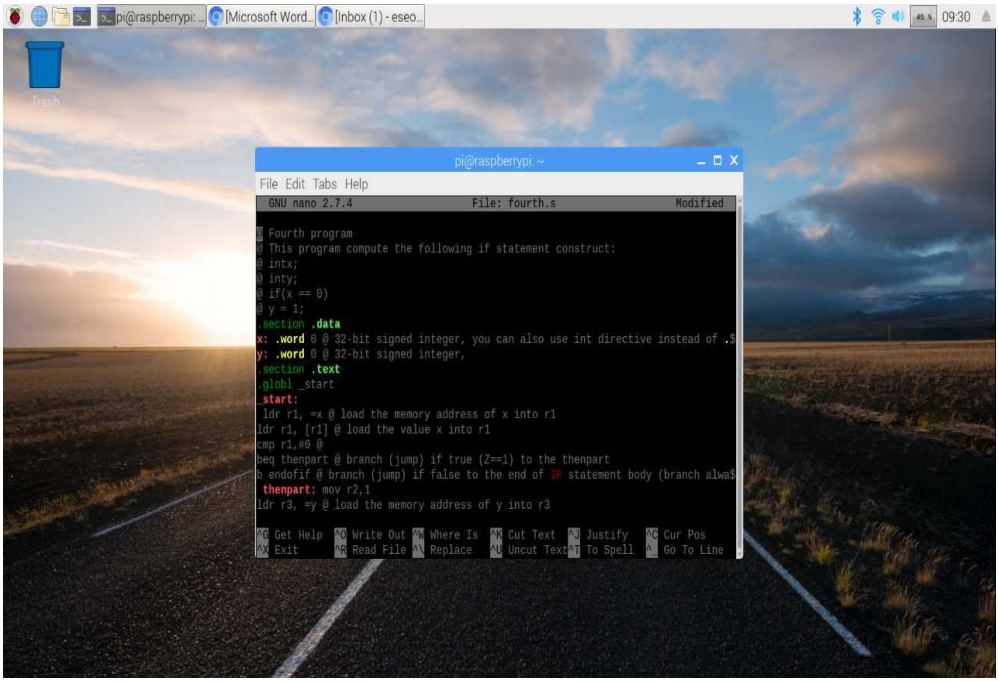
These instructions assume that you have a Raspberry Pi up and running. After starting up your Raspberry Pi, you should have a GUI interface to Raspbian, the Linux variant operating system for these little machines. For this tutorial, you will primarily use the terminal window and type commands into it. You will also use an editor (more on that below).

a) Part1: fourth program:

1. Open the **Terminal** (command line) in Raspberry Pi, click on the 4th icon to the left on the top bar.
2. Type the following in the terminal window, along with the file name:

nano fourth.s

Note: Control-w writes the file (save); control-x exits the editor.

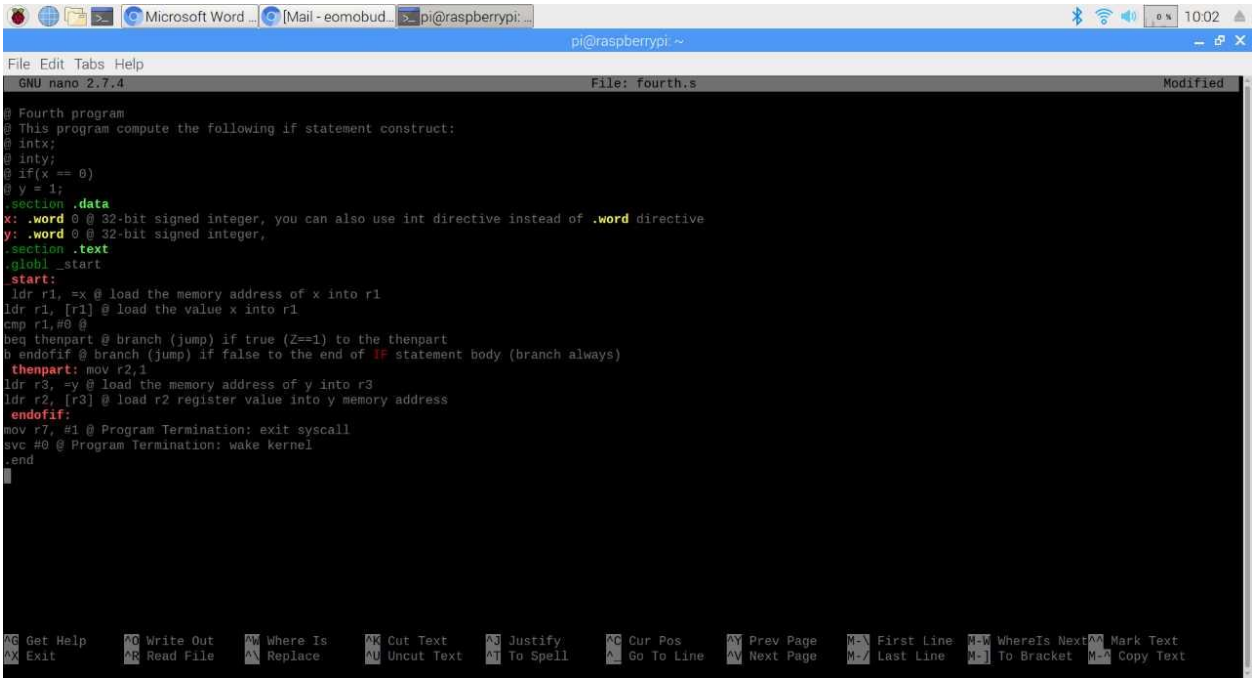


- We had to search how to use scrot to save and screenshot our work.
- This screenshot shows what happened after we used scrot.

3. Write the following program using nano and save it (control-w)

See appendix section to see how signed integer is represented

- The screenshot below shows the program inputted into the Raspberry Pi.



- According to the assignment, we had to move all the values between the registers.

-This was the program after we typed it into the Raspberry Pi.
-We had to make some slight changes in order to get the program to run as there were some typos in the instructions.

4. To **assemble** the file, type the following command:

Debug your program by doing the following in the terminal window:

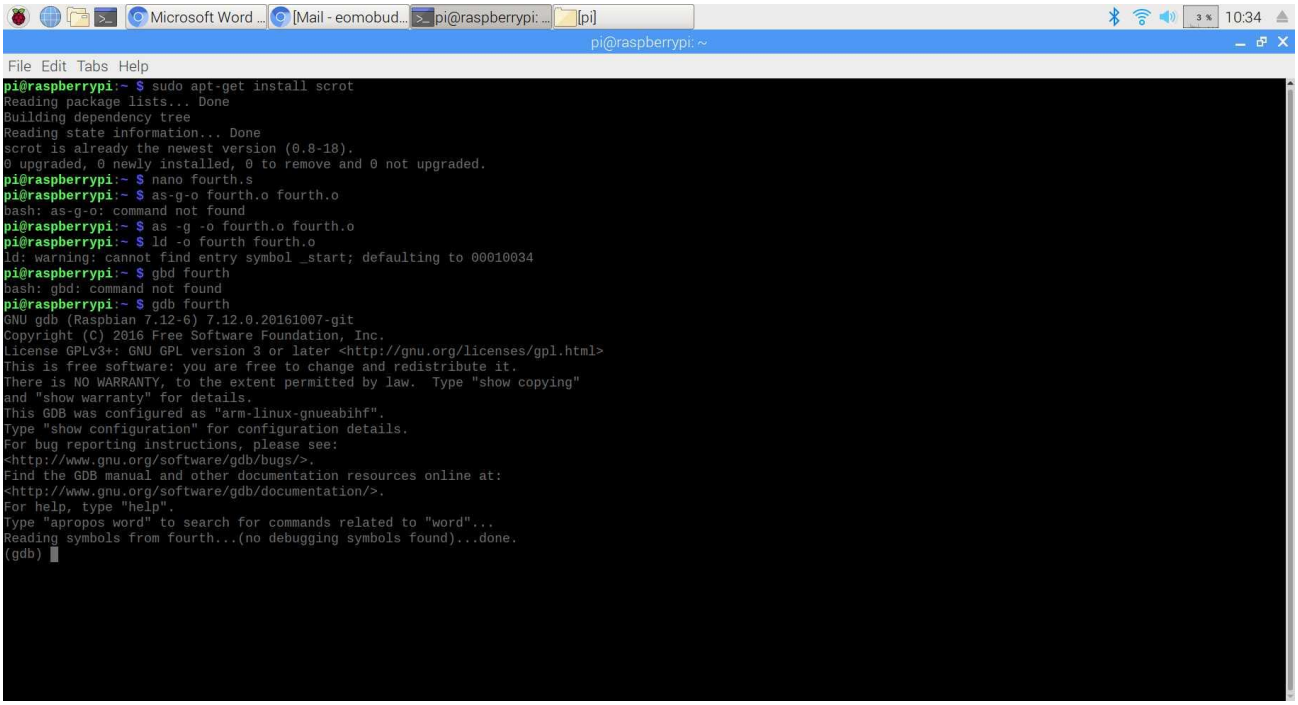
as -g -o fourth.o fourth.s

The linker command stays the same: **ld -o fourth fourth.o**

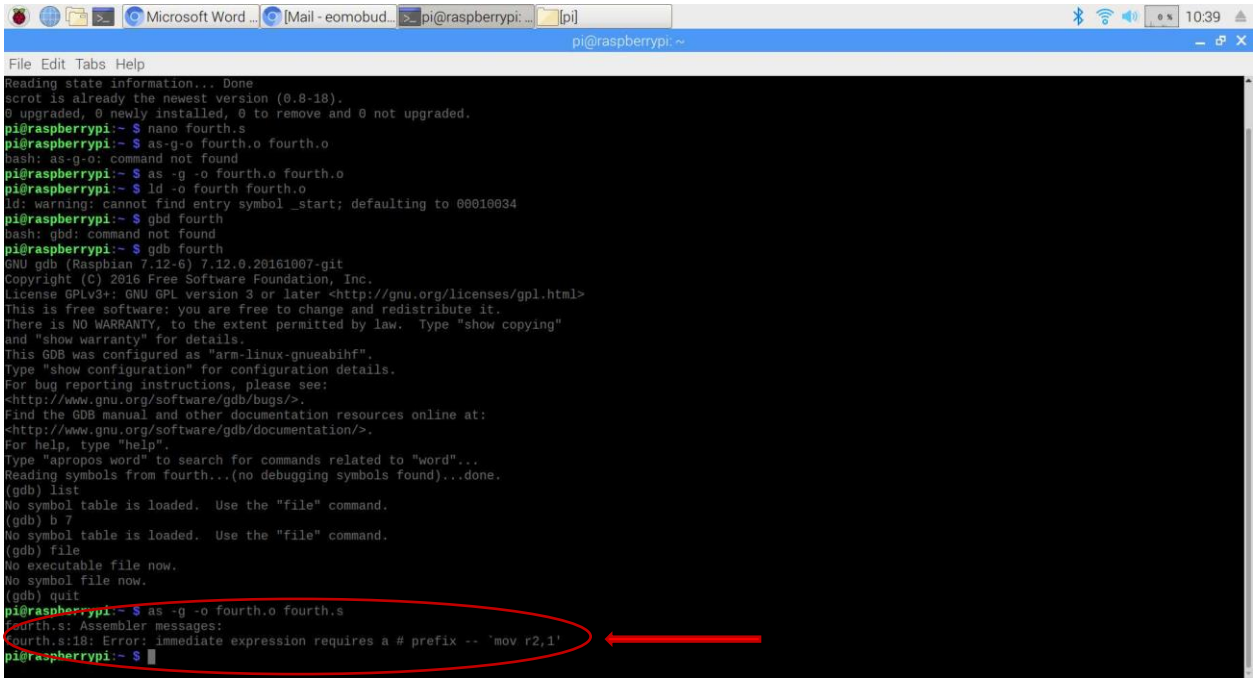
To launch the GNU Debugger, type the command gdb followed by the executable file name at the prompt:

gdb fourth

After displaying the license and warranty statements, the prompt is shown as (gdb). See the following figure:

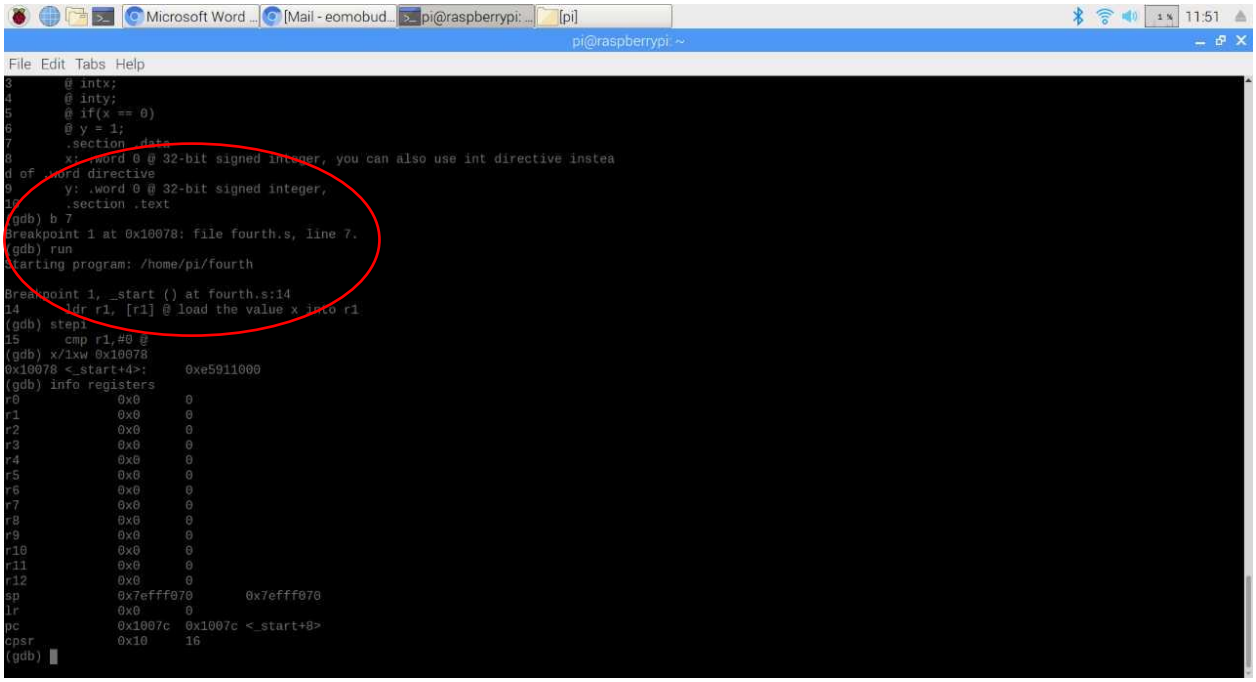


- We ran into a few problems trying to assemble and link the program. These screenshots show the errors we encountered.
- We had to fix the program so that it could find "_start"



- After an hour of trying to fix the problems, we finally managed to fix one, however, another one popped up.

- There was an error with the part of the code “mov r2,1” so we had to fix that as well.
- We fixed it by rewording the code and adding a Direct Operand
- After that process was complete we used (gdb) stepi and this is shown in the screenshot below.



- Before we could go through the program, we had to set breakpoints.

- These were set in the program so that we are able to see what data is being moved to the registers and what values are getting changed as the program progresses.

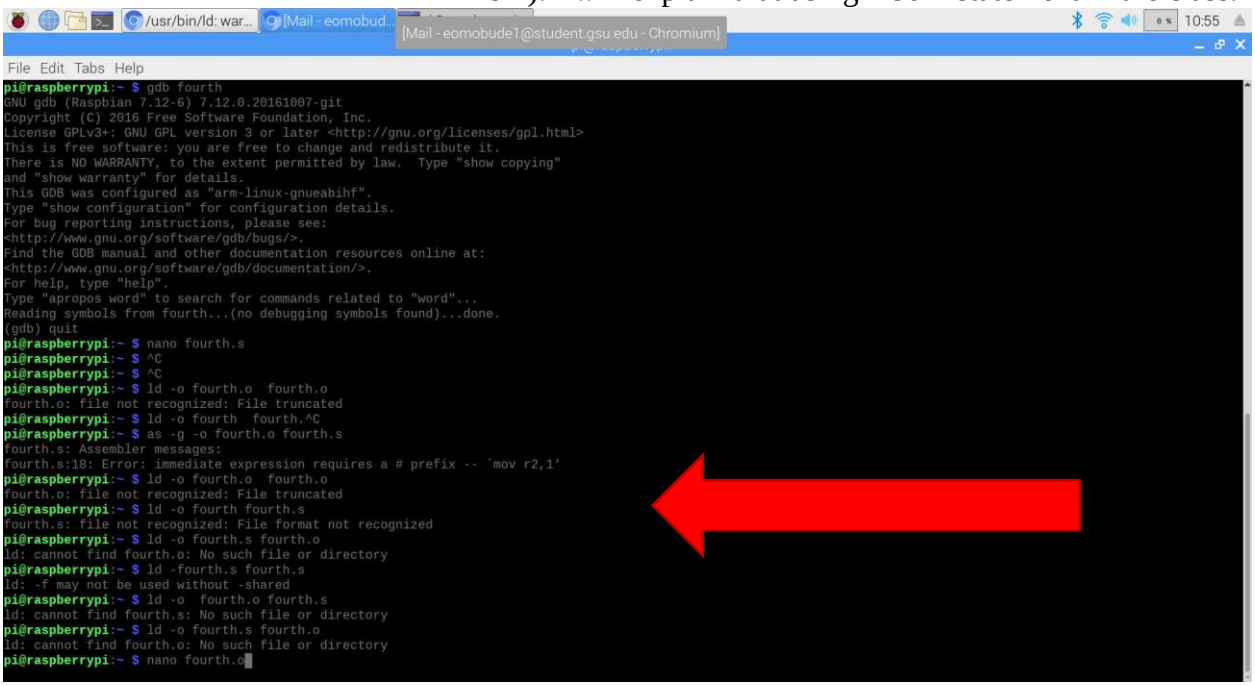
- We checked the values by using ‘stepi’ command to see each line running one by one and also ‘info registers’ to check the content of the registers.
- Breakpoint 1 was created at line #7
- Breakpoint 2 was created at line #14

b) **Part2:** Using the fourth.s program as a reference to update, assemble, link, run, and debug the new program.

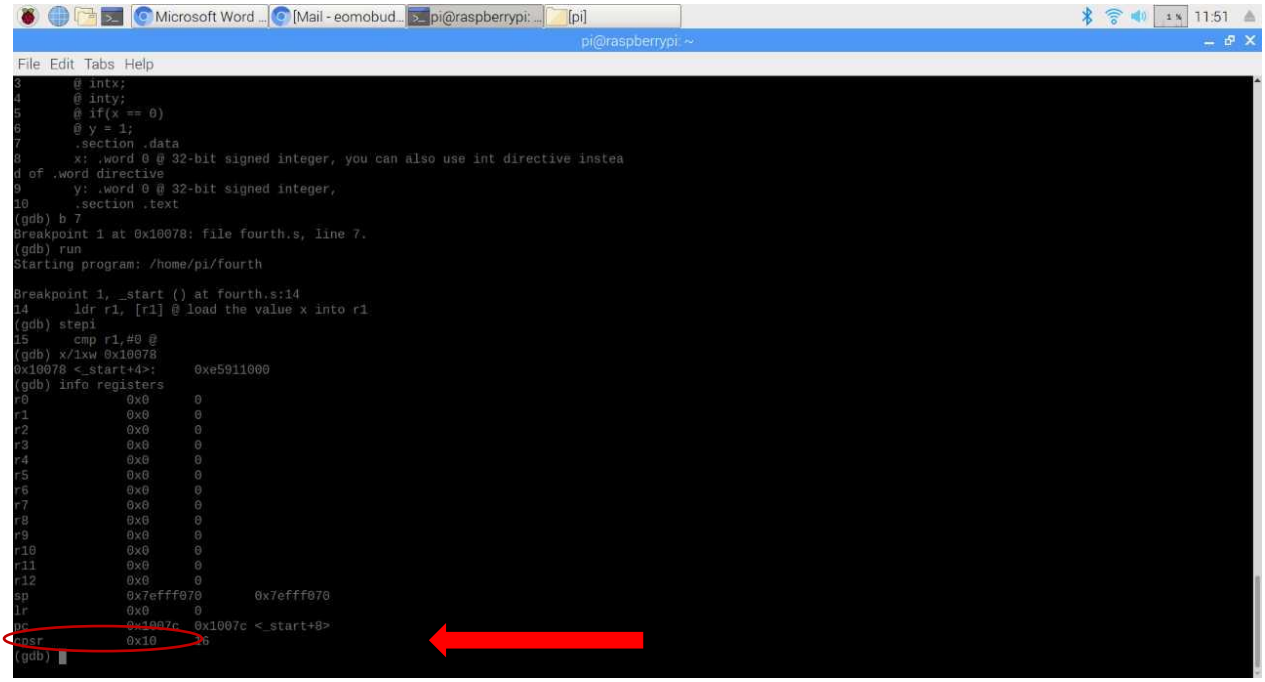
- Is the program in part one efficient?
    - No the program is not efficient because it includes an instruction that does not allow it to run correctly. To be specific, the code contains back-to-back branches (**beq** followed by **b**). We want to avoid this, since branches may cause a delay slot.

- What did we do?
    - We replaced **beq** with **bne(**branch on not equal (Z==0)) and we removed the line **b endofif**

Note: What you did here is you jumped when the condition was false which is different from what we do in the high level languages. In Other words, we reversed the Boolean expression using DE Morgan Law (< is >=, > is <=, == is !=, <= is > and >= is <). I will explain that using X86 if statement in the class.



-After we implemented this step of removing the “b endofif” line, we had to fix a few of the errors In the code because it would not run. We had to add an immediate prefix. We also changed “1” to “#1” in line s:18 and then tried to run the program again.



-The z flag is in the cpsr register which is at the bottom. The Z flag is 1 because the outcome in r1 is 0.

-There is a value of 16 in one of the registers.

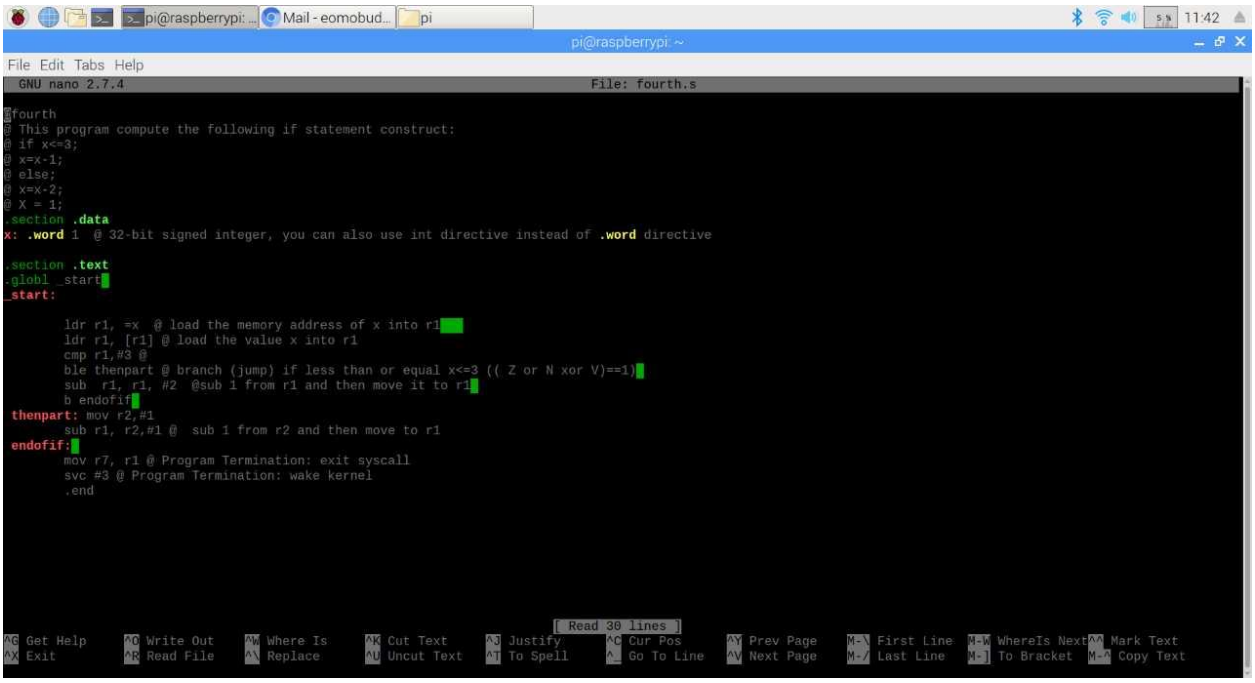
c) **Part3:** Using the fourth.s program as a reference to edit, assemble, link, run, and debug the new program.

Write a program that calculates the following expression:

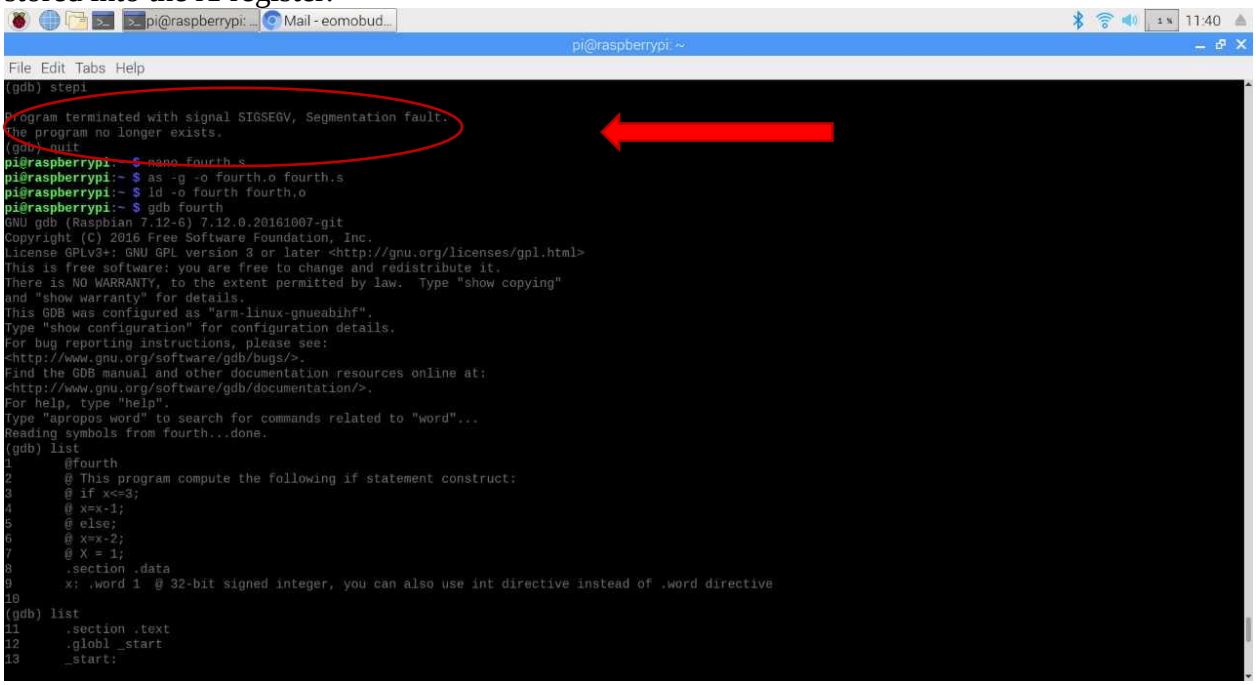
if X <= 3
X = X – 1
else
X = X-2

-We ran into a lot of problems trying to get the program to run. It took hours of coding and reprogramming to finally create a program that worked. This is what we came up with.

-

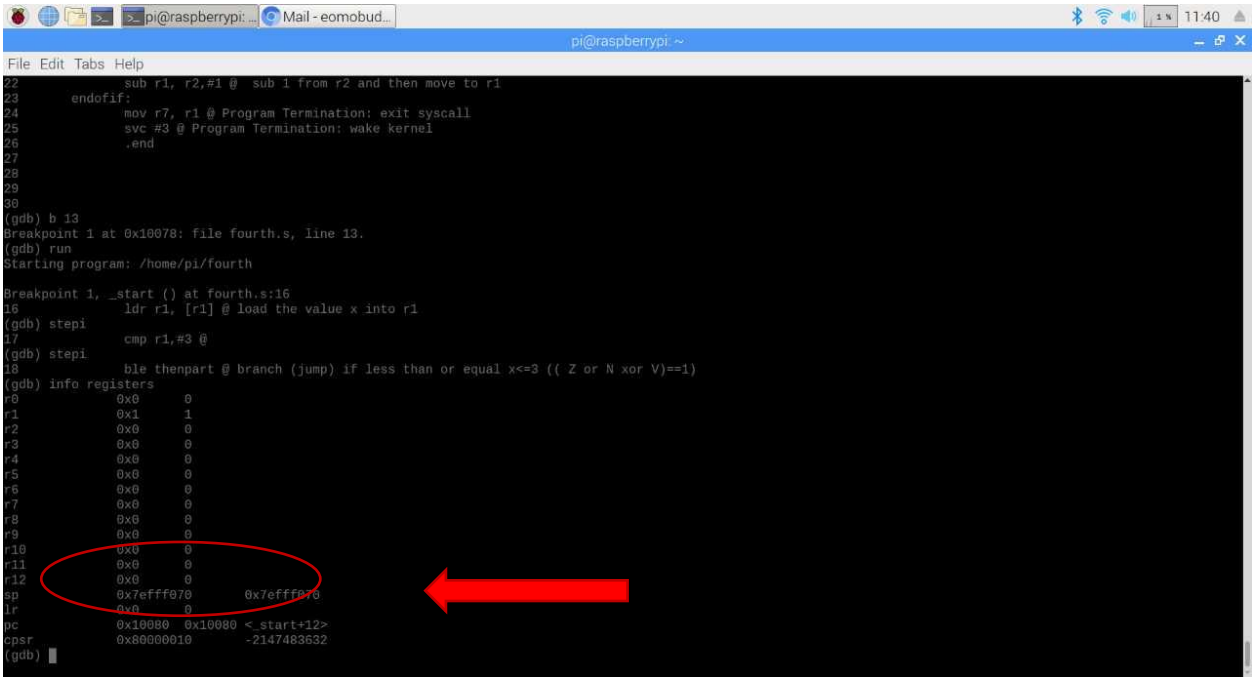


-We used the Fourth.s program as a guide and then replaced some things. For example, we removed the second variable and just left the code with one variable (x).We ended up using “sub r1, r1, #2” and “sub r1, r2, #1” to get the final result. stored into the r1 register.



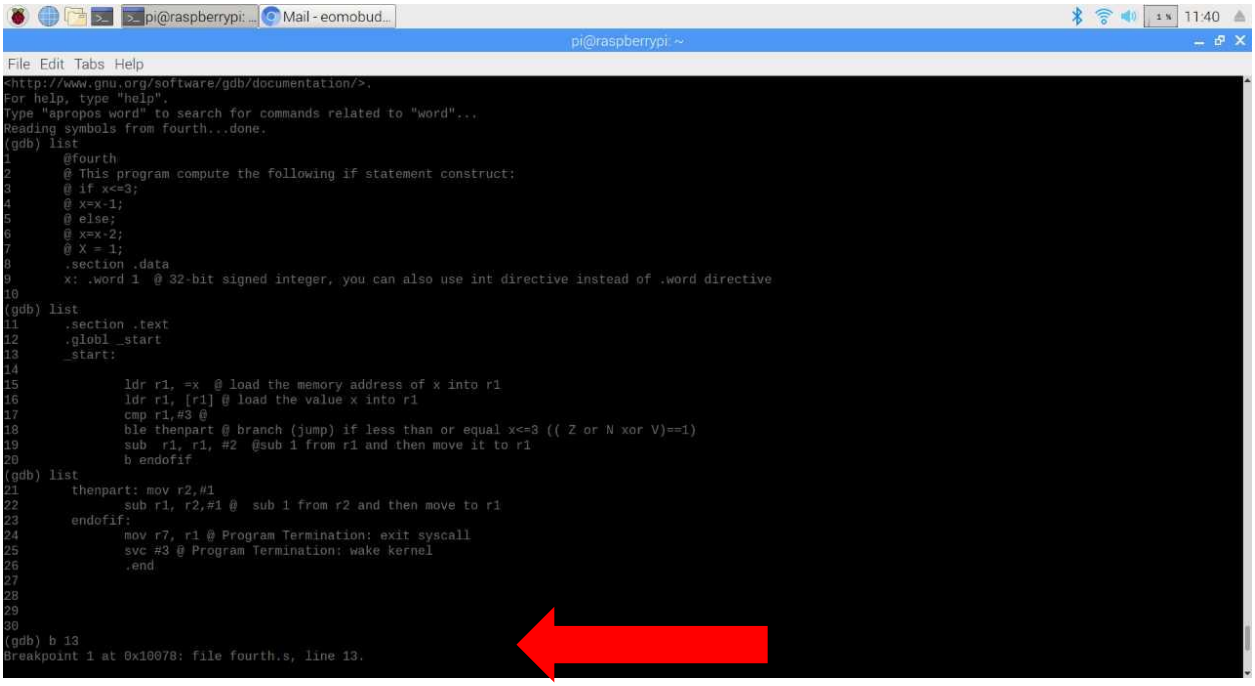
-The program kept getting terminated due to that and it is shown at the top of this screenshot.

-We had to type “quit” several times and redo the whole process because we kept typing “stepi” too many times.



-We observed on the first command that it shows the stored value inside the memory(0x7efff070) but another command gives a long output with the same \x7efff\.

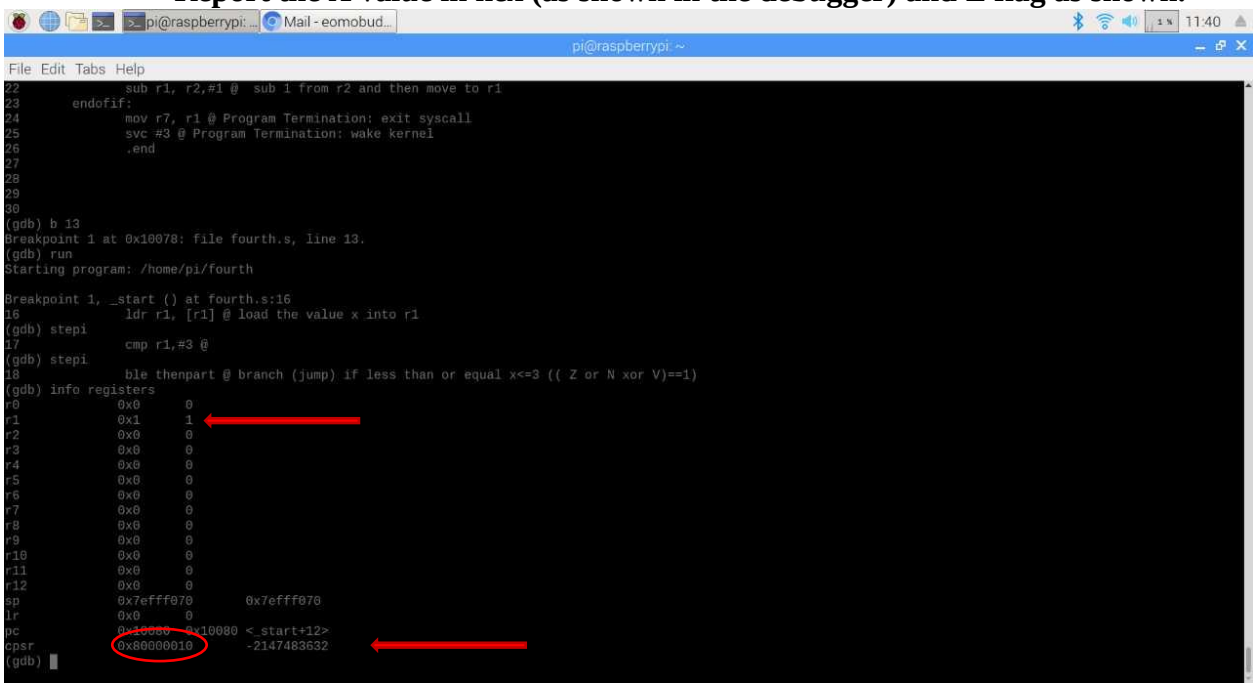
-Breakpoints were set at s:13 and s:16



-We ran into a lot of problems after changing the variables for the new code.
-After a lot of trial and error, we were finally able to figure out how to get the code to assemble and run.
-We set a breakpoint at line s:13 :0x10078

Assume that X is 32-bit integer memory variables and assign X 1.

- **Use the debugger to verify the result in the memories and the Register.**
- **Report the X value in hex (as shown in the debugger) and Z flag as shown.**



-We confirmed that r1 had the value of 1 as shown in this screenshot.
-The flags are shown in the bottom next to the cpsr line. This is where the values are stored in the registers.
-cpsr is the register where the total value(solution) of the equation is stored which is
-R1 contains the signed value 0
-R2 contains the first half of the program and the value is 0
-R3 contains the second half of the program and the value is 0

The most confusing part of this assignment was figuring out how to run the program and create my own program in part 3.