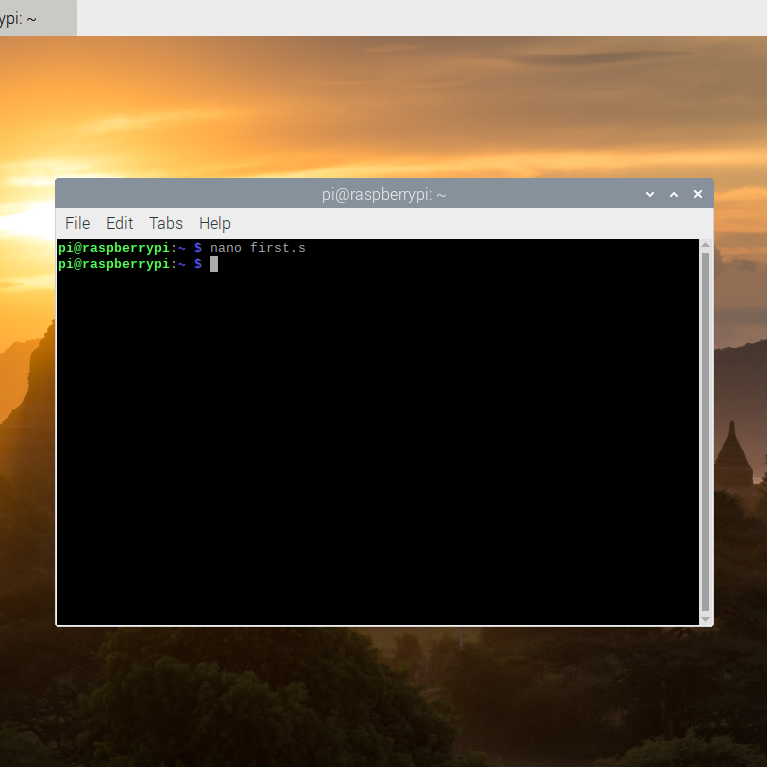
**Task 4: Raspberry PI Installation and ARM Assembly Programming**

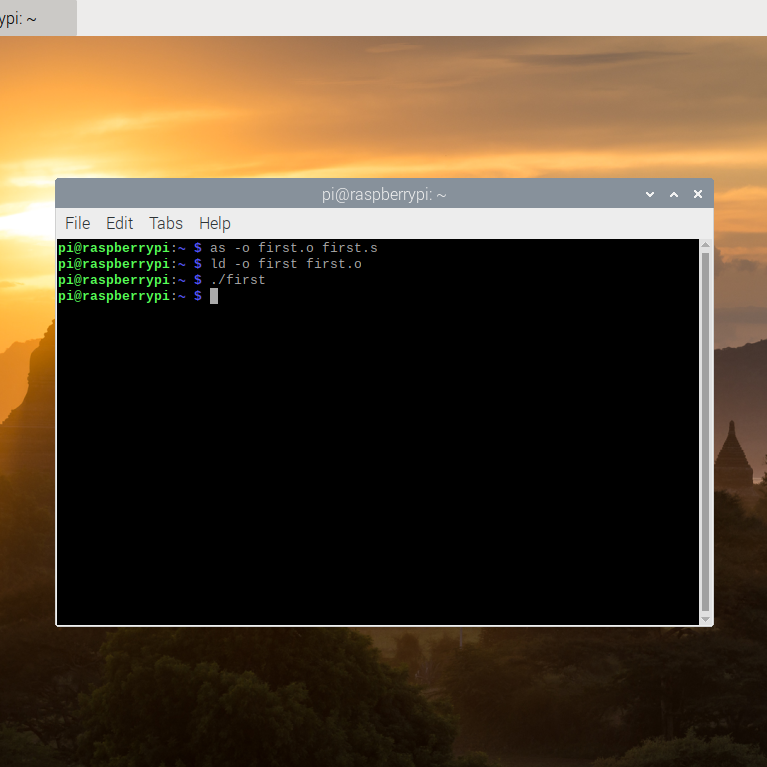
**Part 1 - First Program**

After downloading and installing Raspbian Buster with Desktop (the newest edition of Raspbian), we proceeded to follow the tasks to practice utilizing the ARM Assembler.

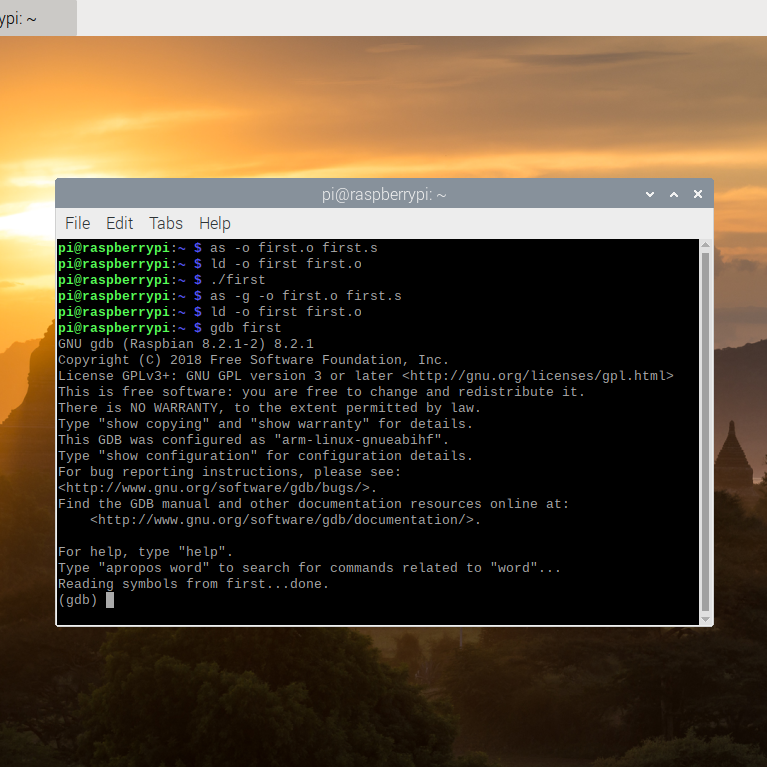


Utilizing the default nano text editor by way of the terminal window, we wrote and saved the following program:

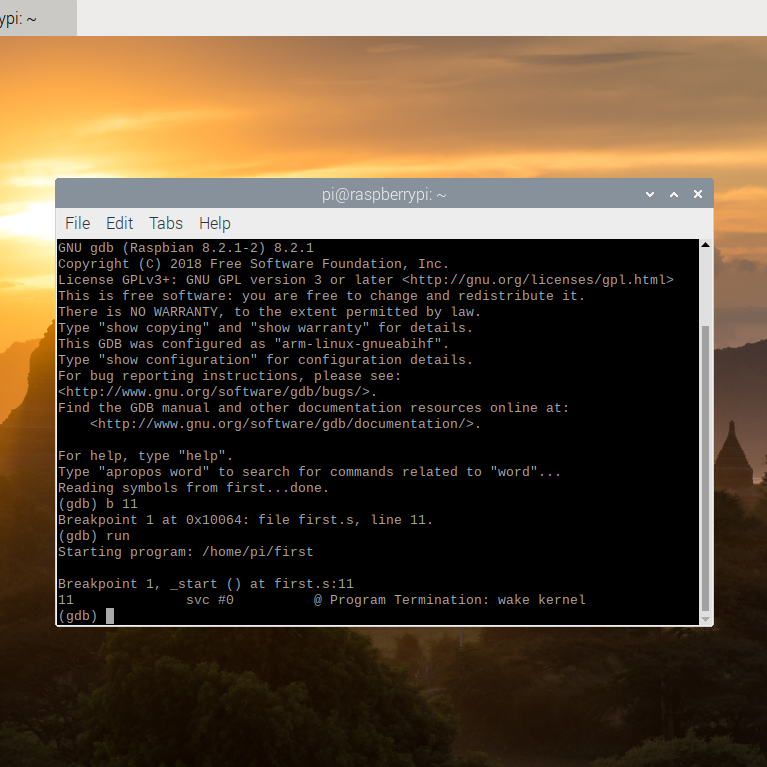




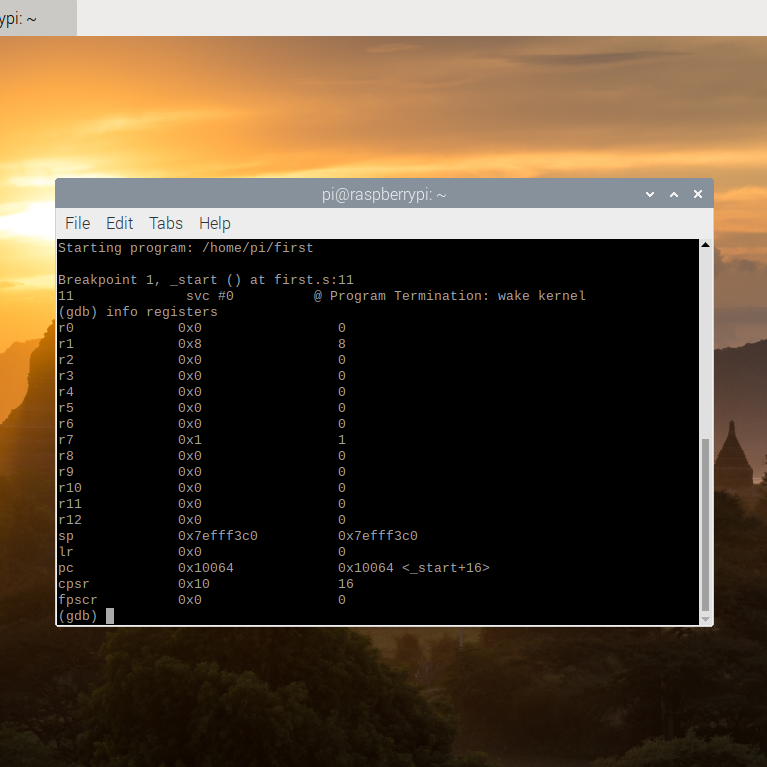
After composing the file, we then assembled and linked it in order the receive an executable file. Then we executed it. We did not see any output when the program was ran, however, this was expected. As the entire program takes place in the registry, we will not receive any output.



Next, we practiced utilizing the GDB (GNU DeBugger) in order to get an idea on how our First Program was being executed. Preparing to use the GDB required us to re-assemble the program with an additional flag “-g”. No additional steps were needed when the program was re-linked. After the new executable was generated, we launched the GDB with the particular file named “first” which we wished for the debugger to examine (utilizing “gdb first”).



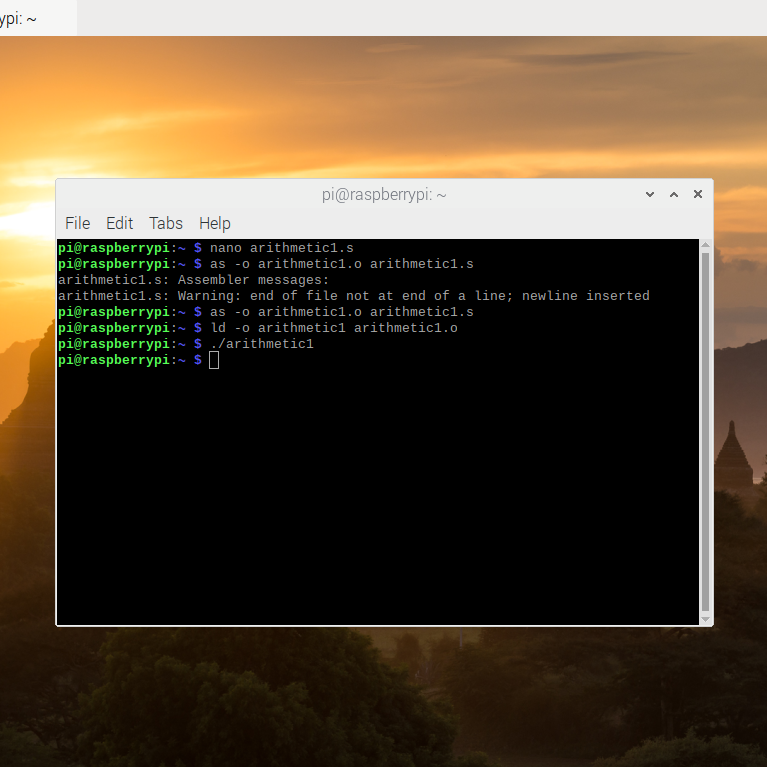
If the program already executes as expected, the debugger does not provide much information without an additional step, setting a breakpoint. By halting the program during its execution, we were able to see how the program was completing its intended task or tasks. To stop the program prior to the execution of line 11, we gave the GDB the command “b 11”, and then we ran the program from its beginning until the breakpoint utilizing “run”.



In order to see the generated values in the CPU registers (before line 11 was executed), we needed to utilize the command “info registers”. The information was then displayed as illustrated above. Register 1 has a value of 8. This was expected: as the register was first set to 5, then it became 4 when 1 was subtracted from it, and then finally 4 was added to that value, resulting it 8. We also found that register 7 held a value of 1, which was expected due to the inclusion of the command “mov r7, #1”.

**Part 2 - Calculate the Expression**

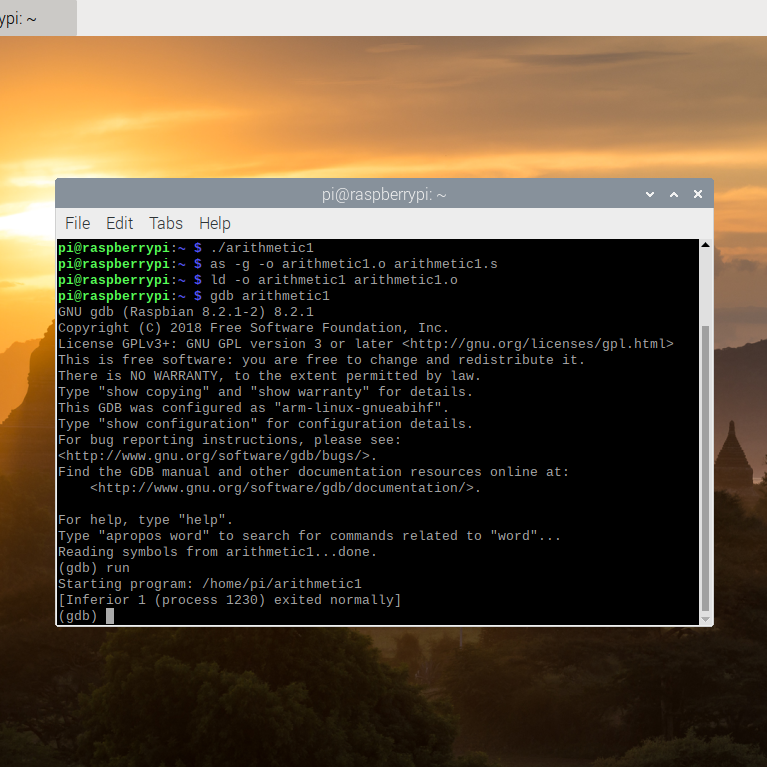
With this better understanding of the process, we began Part 2 in which we created, assembled, linked, ran, and debugged our own program which calculated the expression A = (A+B) - (C\*D) where A=10, B=11, C=7, and D=2.



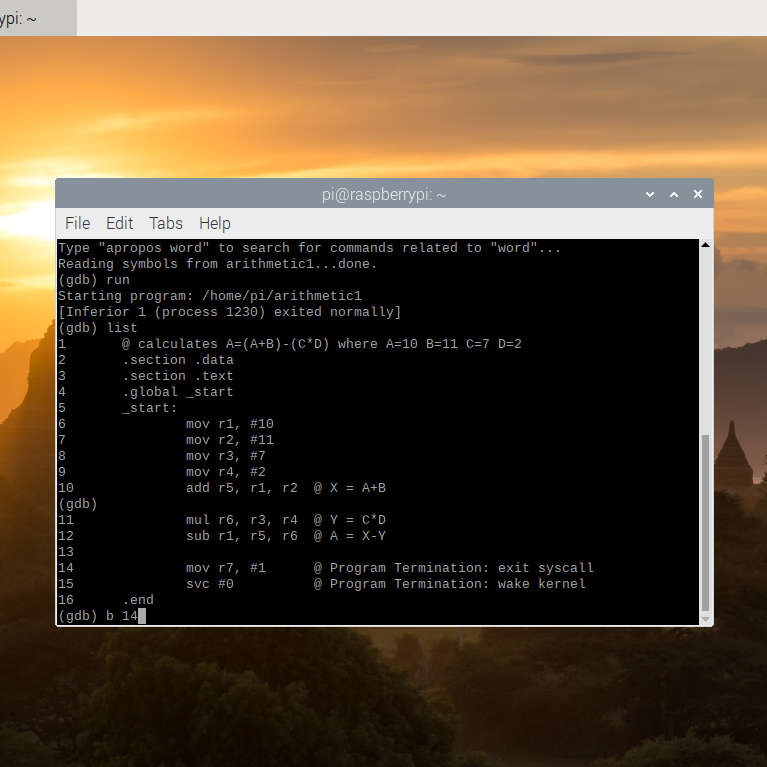
As in Part 1, we composed the program through the terminal by way of the nano text editor, before assembling it. This time when composing the program though, we realized an interesting fact. When composing a program, an extra, empty line must be included following the end of the program, or the assembler will take it upon itself to add the line for the user. After assembly, we linked the program to create the executable file and ran that file.

For reference, the program we utilized was as follows:

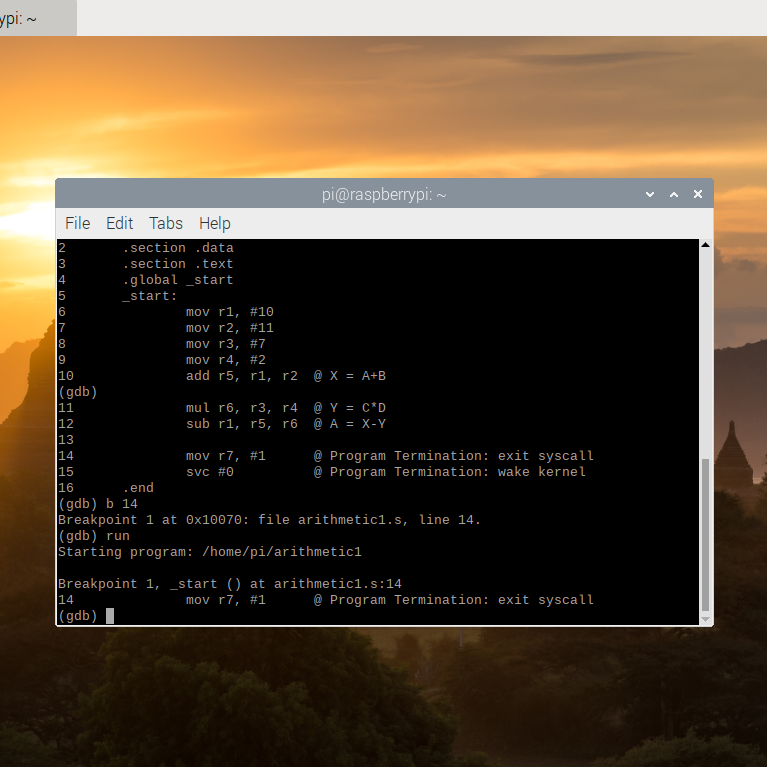




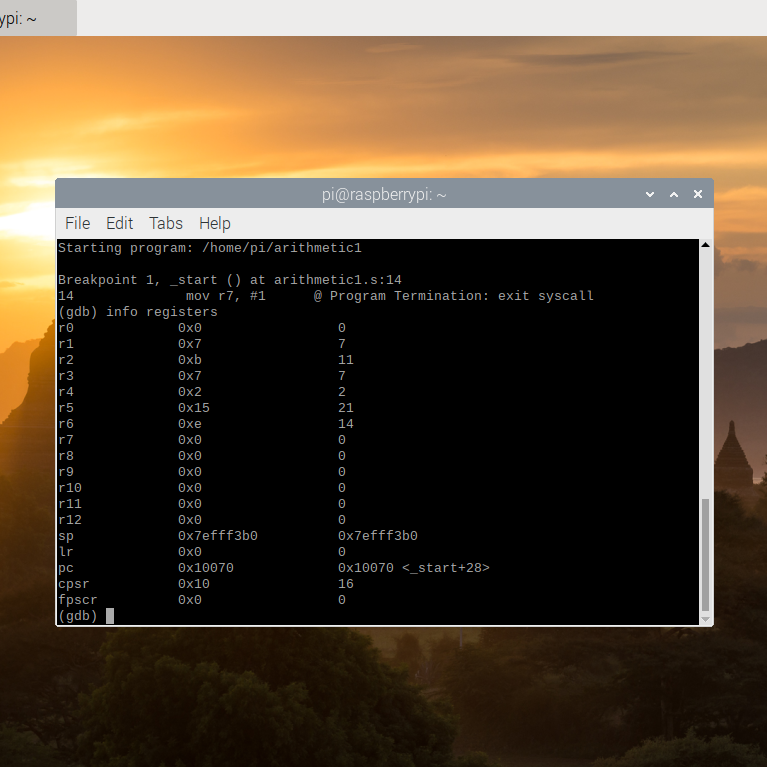
As with before, we were not able to “see” the program at work, and therefore took the steps to launch the GNU Debugger so as to get a better idea of how our program was functioning.

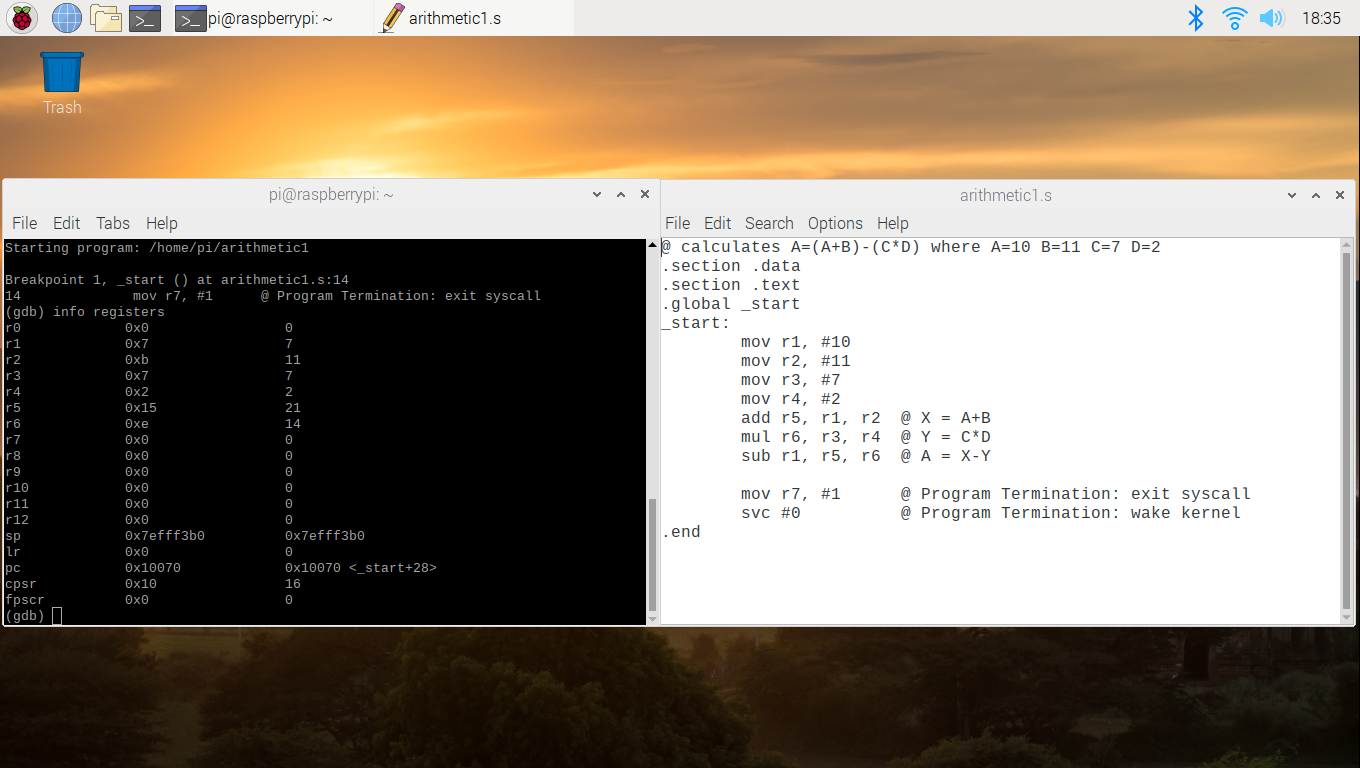


To see where we ought to insert our breakpoint, we utilized the “list” command, allowing us to see the entirety of the program with the lines numbered. We then chose to halt the program before line 14 had a chance to execute.



We then ran the program before utilizing the “info registers” command (whose results are shown in the next figure).





Comparing the values stored in the registers with the program itself, we could see that each of the values were expected. Registers 2 through 4 held the values 11, 7, and 2 respectively which were their assigned values. Register 5 held the value of 22 - the result of our initial A added to B (10+11). Register 6 read as 14 which was the result of C\*D (7\*2). Register 1, however, held A’s final value of 7 which was the expected result of [(10+11) - (7\*2)].