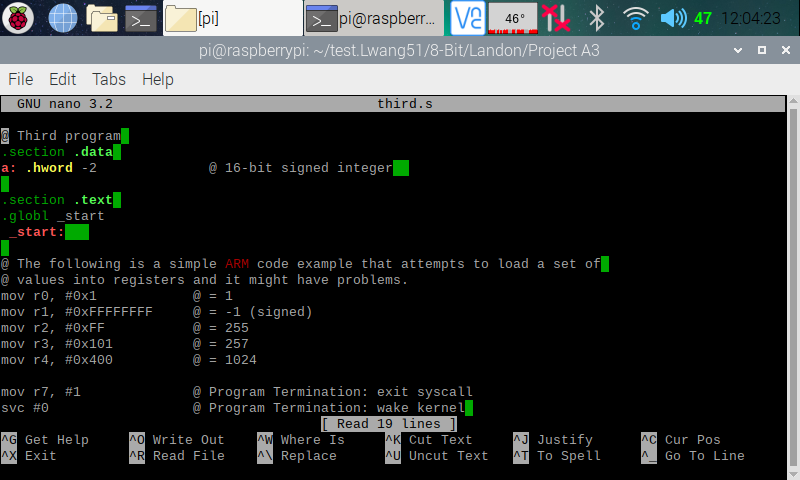
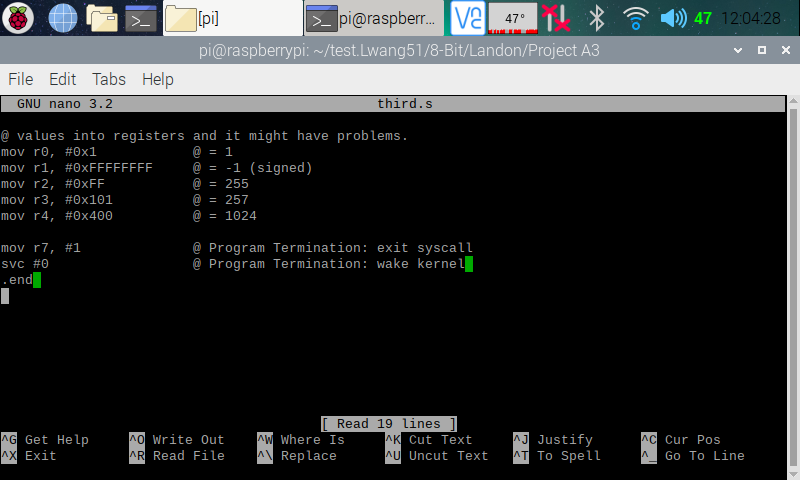
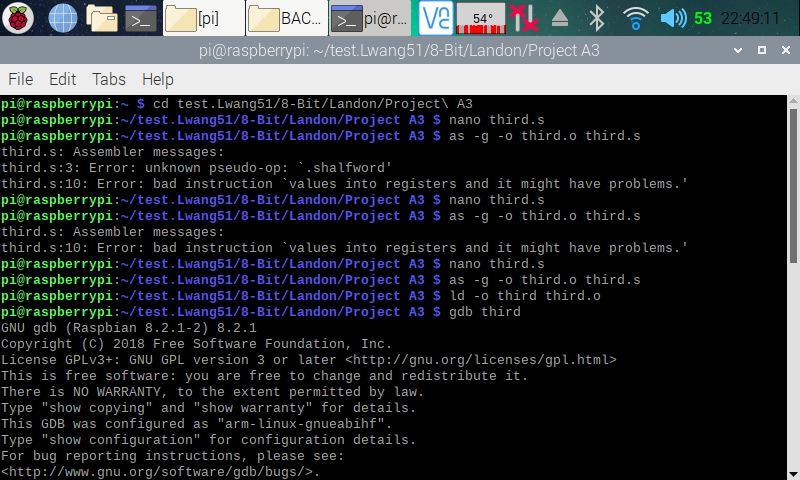
**ARM Assembly Programming**

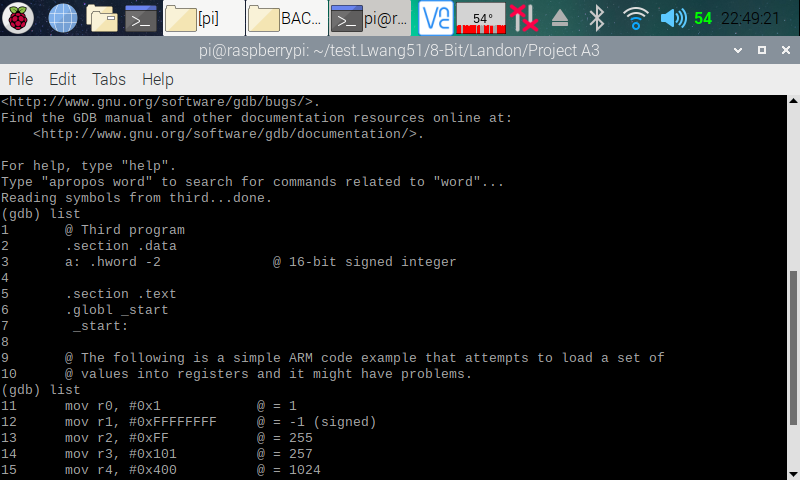
**Part 1: Third Program**



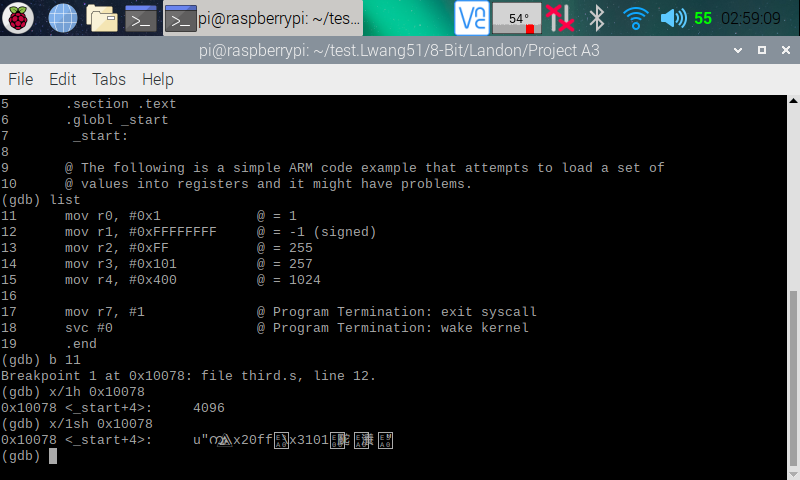


Here (in the two screenshots above), I copied and pasted the Third program from the ARM Assembly Programming A3 document and used the nano editor to create a program on my Raspberry PI.



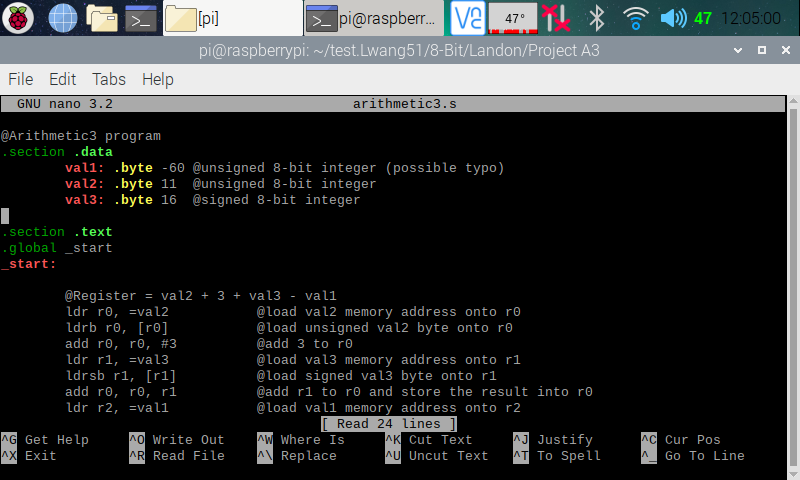


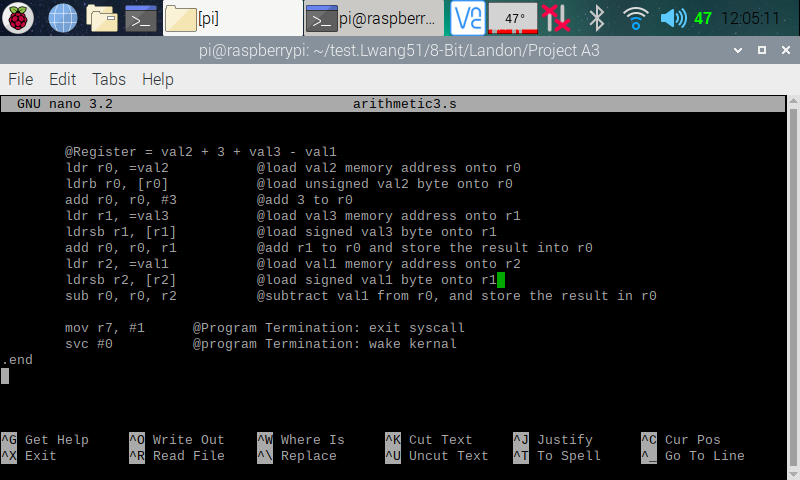
Here (in the two screenshots above), I exited out from the nano editor and assembled and linked the Third program. I got two error while attempt to assemble the program. My first error was on line three of my program. I got this error, because there was no such thing as .shalfword in ARM. Since ARM supports operation on different datatypes, there was no need to add in the ‘s’ in halfword. Along with this, to declare a halfword in ARM, we simply just type hword. To fix this error, we simply just change “a: .shalfword -2” to “a: .hword -2”. The second error was cause by a missing comment sign. I just simply added in a ‘@’ to resolve the error. After assembling and linking the program, I entered the GDB debugger.



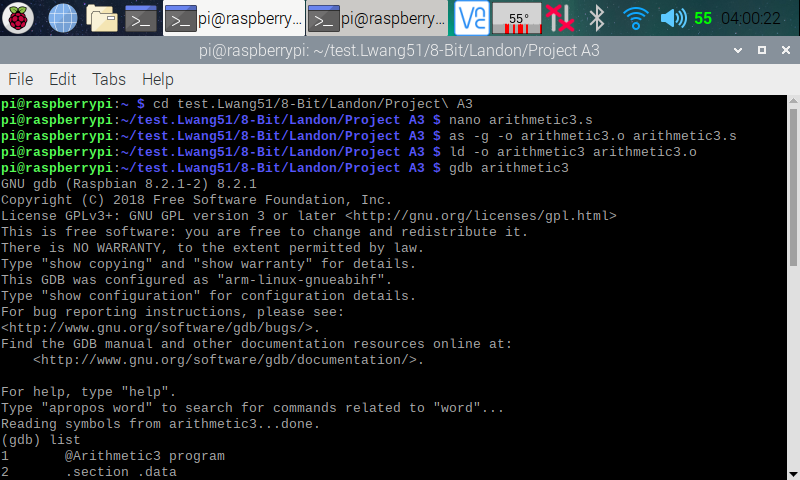
Here (in the screenshot above), I displayed my code, then placed a breakpoint at line 11 (which the debugger automatically moved it to line 12 since line 11 was an invalid point). I then ran the program and entered the memory using the memory address given to me when I set the breakpoint. When I used “x/1h 0x10078” to access the memory, I got a number (4096), and when I used “x/1sh 0x10078” to access the memory, I got many weird symbols.

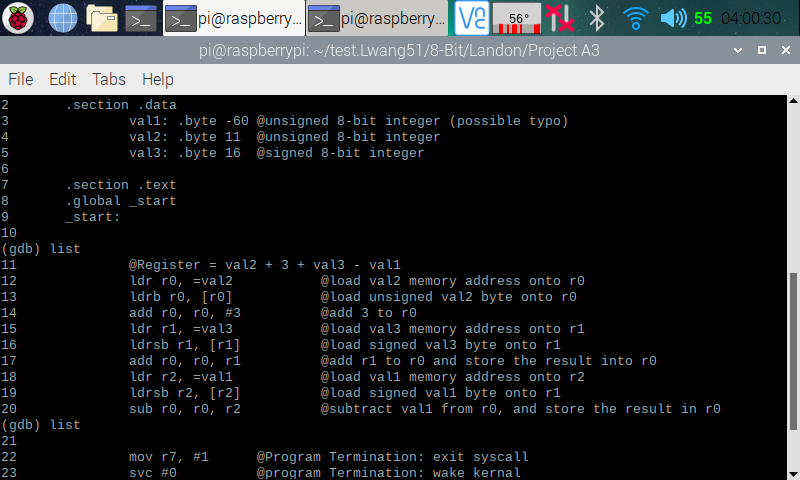
**Part 2: Arithmetic3 Program**



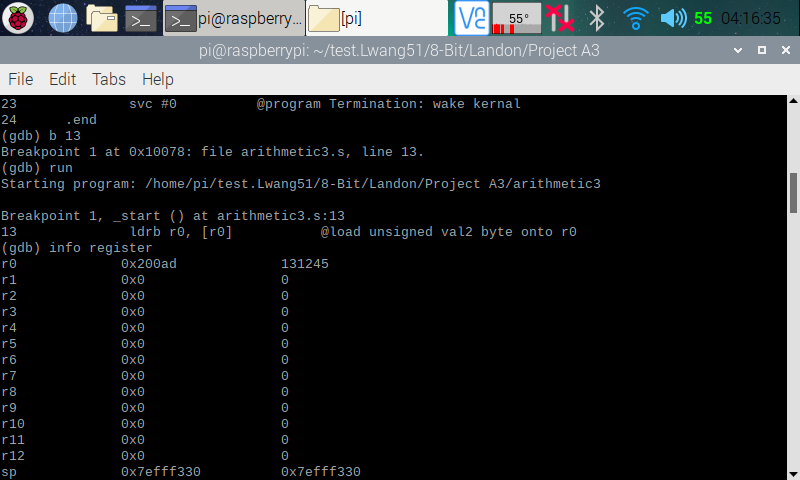


Here (in the two screenshots above), is my code for the Arithmetic3 program.

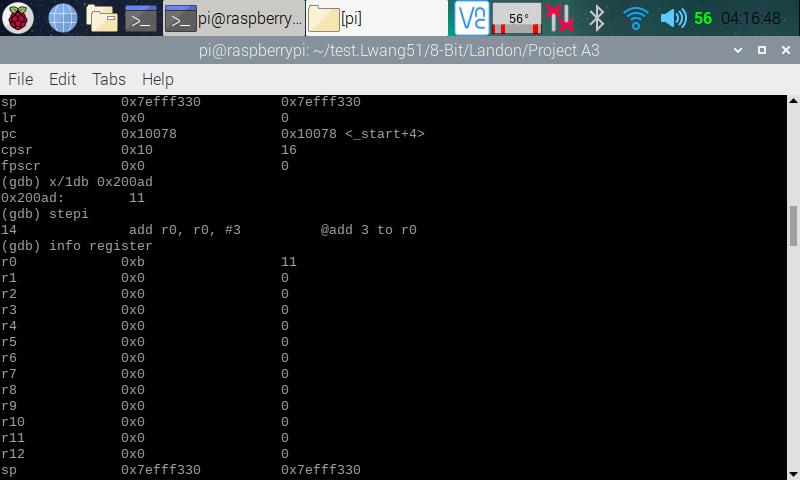




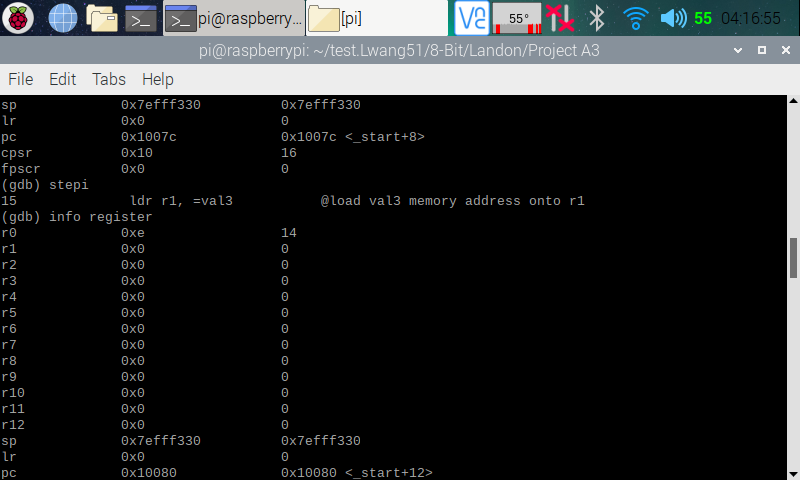
Here (in the two screenshots above), I exited the nano editor, then assembled and linked my program. I then entered the debugger. In the debugger I used **List** to display my codes for reference.



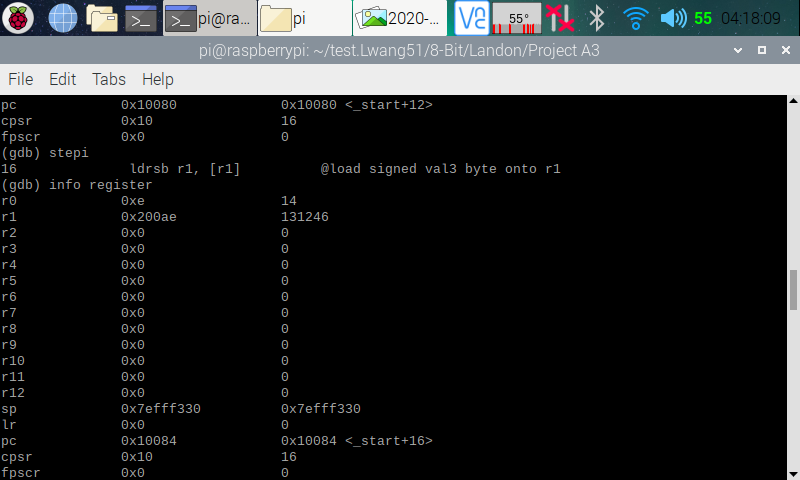
Here (in the screenshot above), I set a breakpoint at line 13, then ran the program. I then opened the register information, and we can see that a memory address has been placed into r0.



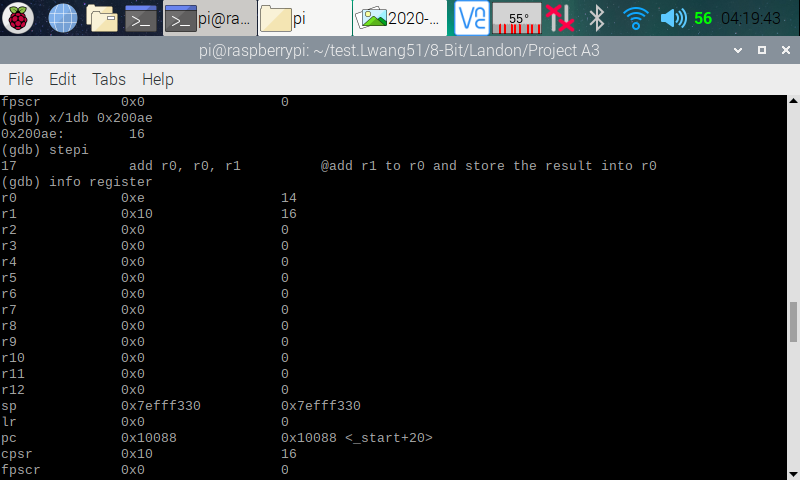
Here (in the screenshot above), I used the memory address placed into r0 to check if it is pointing at the right memory place. The memory that I pulled up shows 11, which is what I was expecting. I then stepped over to the next line so that line 13 can execute. I then pulled up the register, and we can see that number 11 has been added to r0, which is what I wanted.



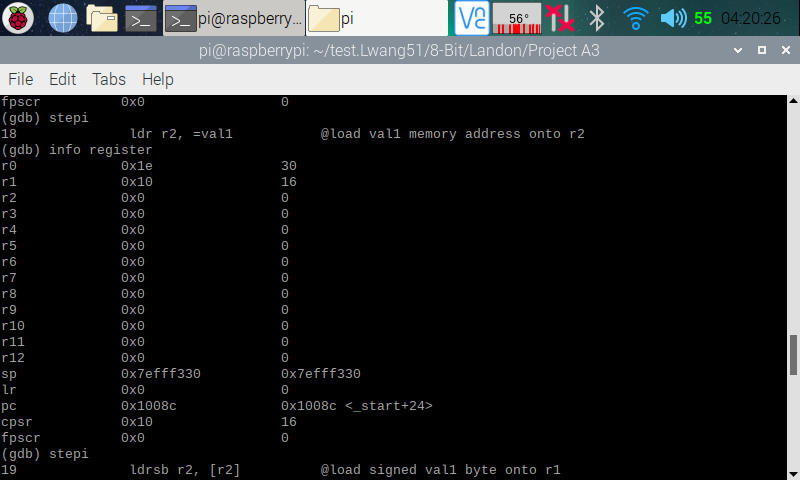
Here (in the screenshot above), I stepped over to the next line so that line 14 can execute. I then opened the register information, and we can see that r0 now holds 14 (11 + 3), and that is what I was expecting.



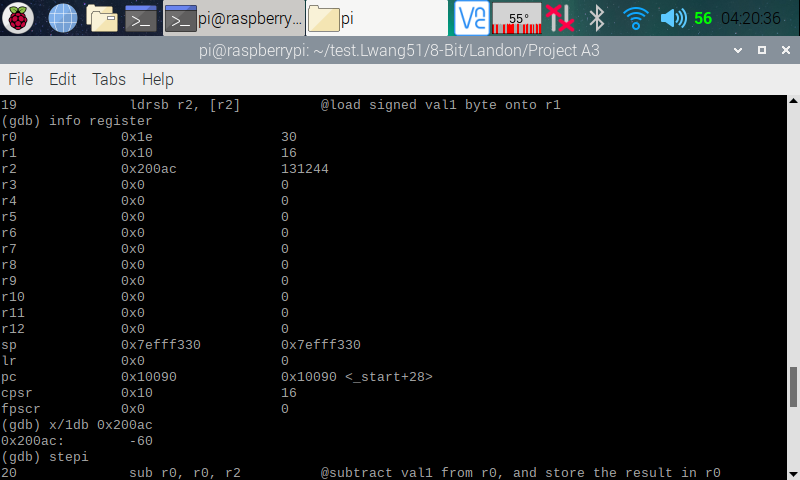
Here (in the screenshot above), I stepped over to the next line so that line 15 can execute. I then opened the register information, and we can see that a memory address has been placed into r1.



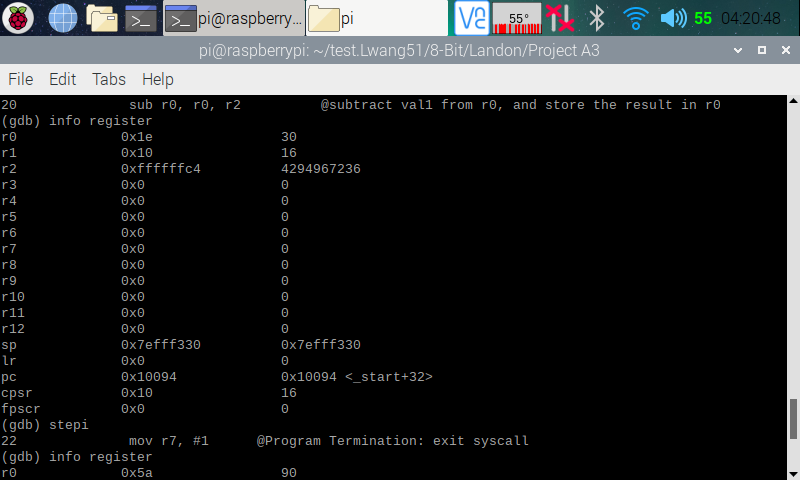
Here (in the screenshot above), I pulled up the memory, and we can see that it is pointing to 16 as expected. I then stepped over to the next line so that line 16 can execute. I then opened the register information, and we can see that 16 has been loaded into r1.



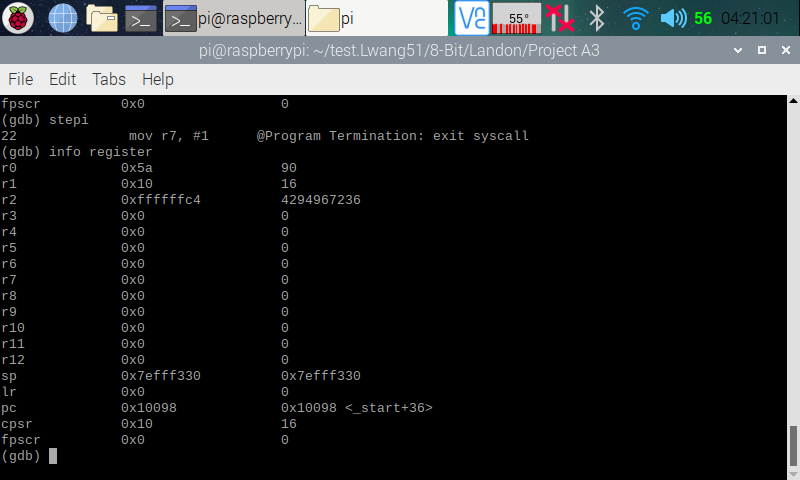
Here (in the screenshot above), I stepped over to the next line so that line 17 can execute. I then opened the register information, and we can see that r0 now stores a 30 (14 + 16), which was expected. I then stepped over to the next line so that line 18 can execute.



Here (in the screenshot above), I opened the register information, and we can see that a memory address has been loaded onto r2. I then used that memory address to pull up the memory, and we can see that it is pointing at -60 as expected. I then stepped over to the next line so that line 19 can execute.



Here (in the screenshot above), I pulled up the register information, and we can see that FFFFFFC4 or 4294967236 (two’s complement of -60) is loaded onto r2 as expected. I then stepped over to the next line so that line 20 can execute.



Here (in the screenshot above), I opened the register information, and we can see that 90 (30 - -60) is now stored in r0, which is what I wanted. I now know that I have the right program.