For the first program (first.s) I follow the example code provided:

@ first program

.section .data

.section .text

.globl \_start

\_start:

mov r1, #5 @ load r1 with 5

sub r1, r1, #1 @ subtract 1 from r1

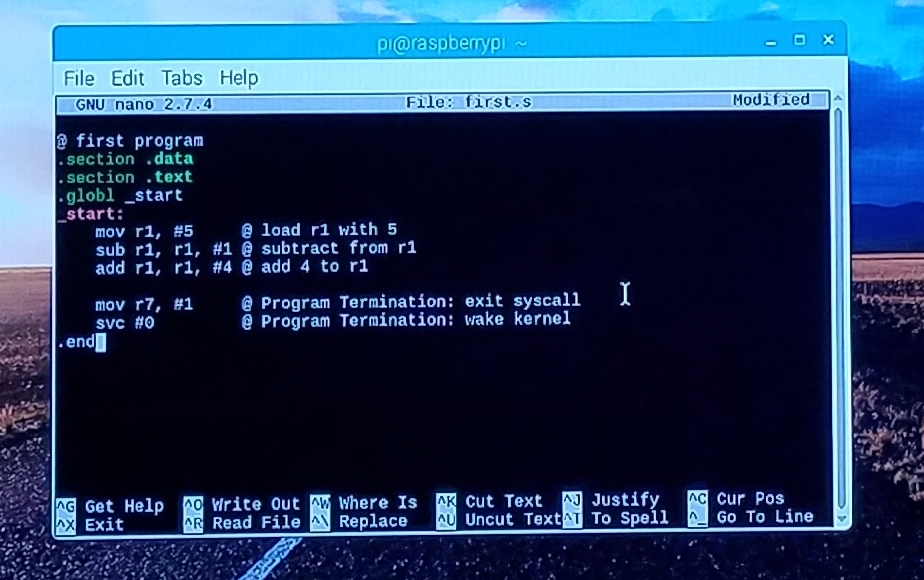
add r1, r1, #4 @ add 4 to r1

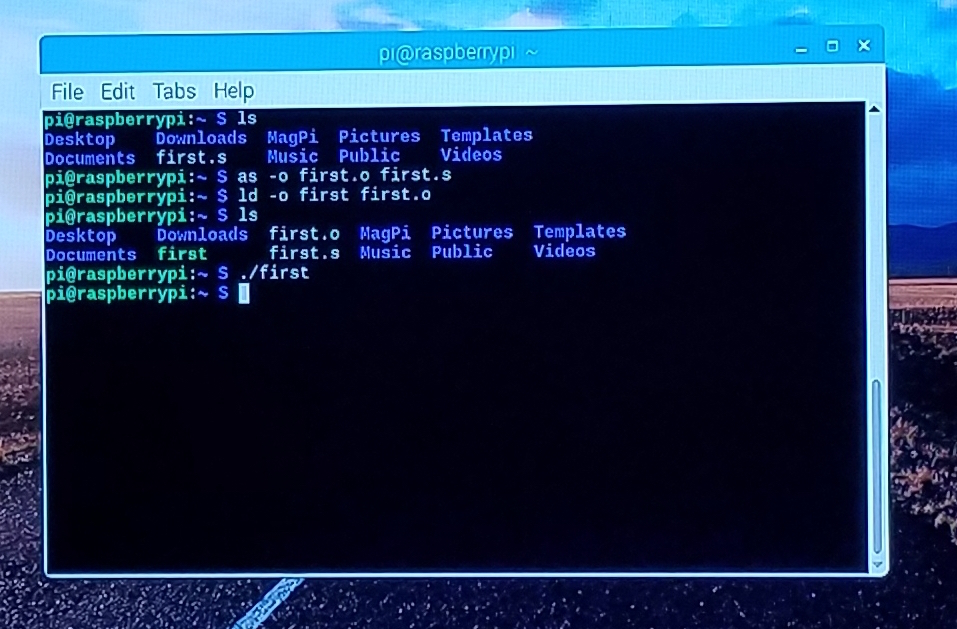
mov r7, #1 @ Program Termination: exit syscall

svc #0 @ Program Termination: wake kernel

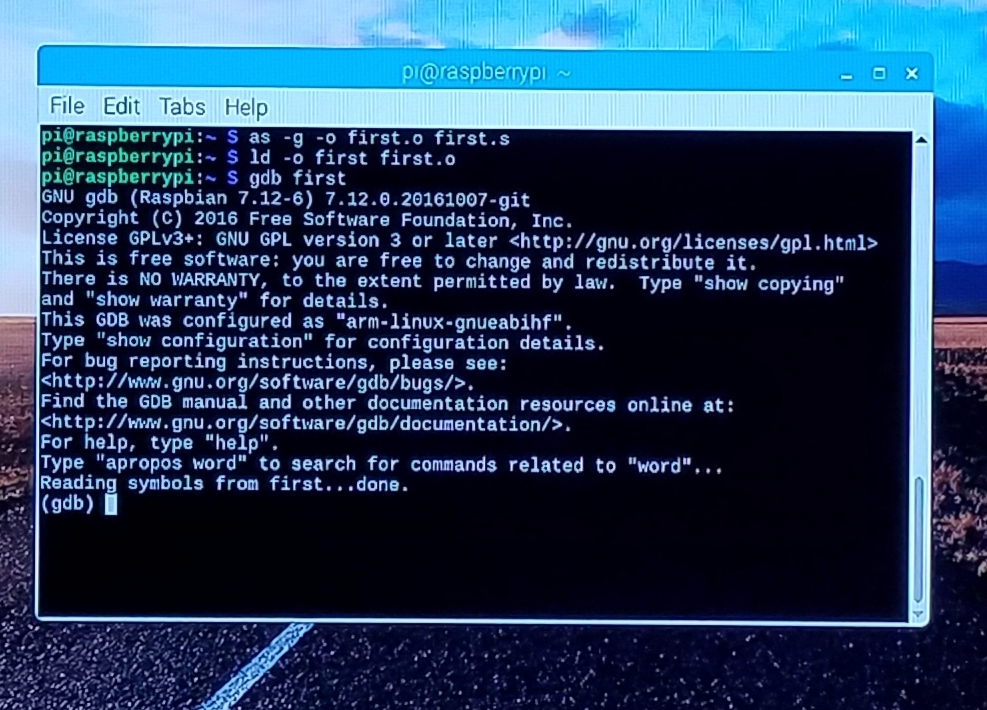
.end

While it is clear that the @ symbol indicates in line comment. I could not figure out what the .section and .globl lines do.

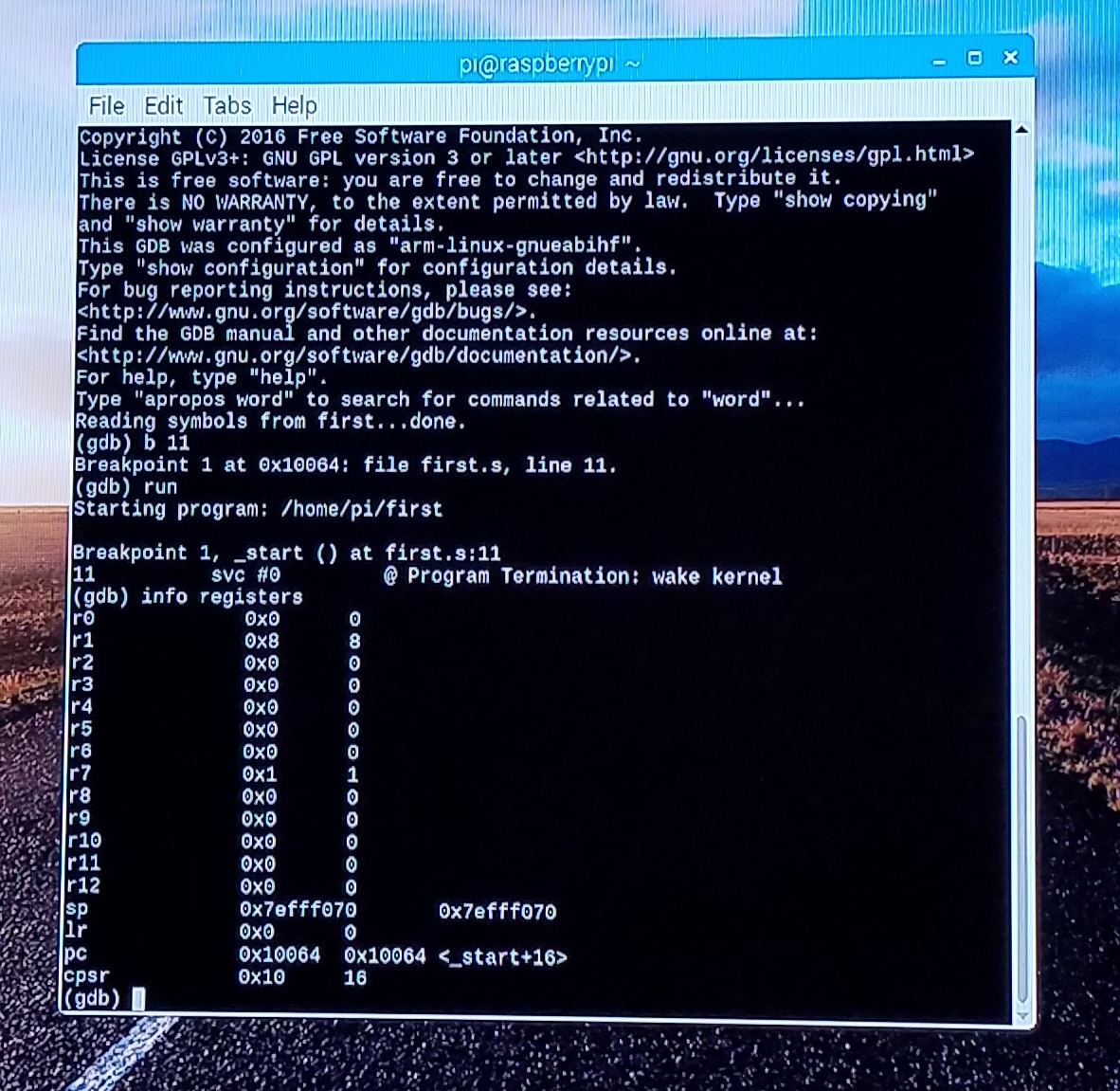




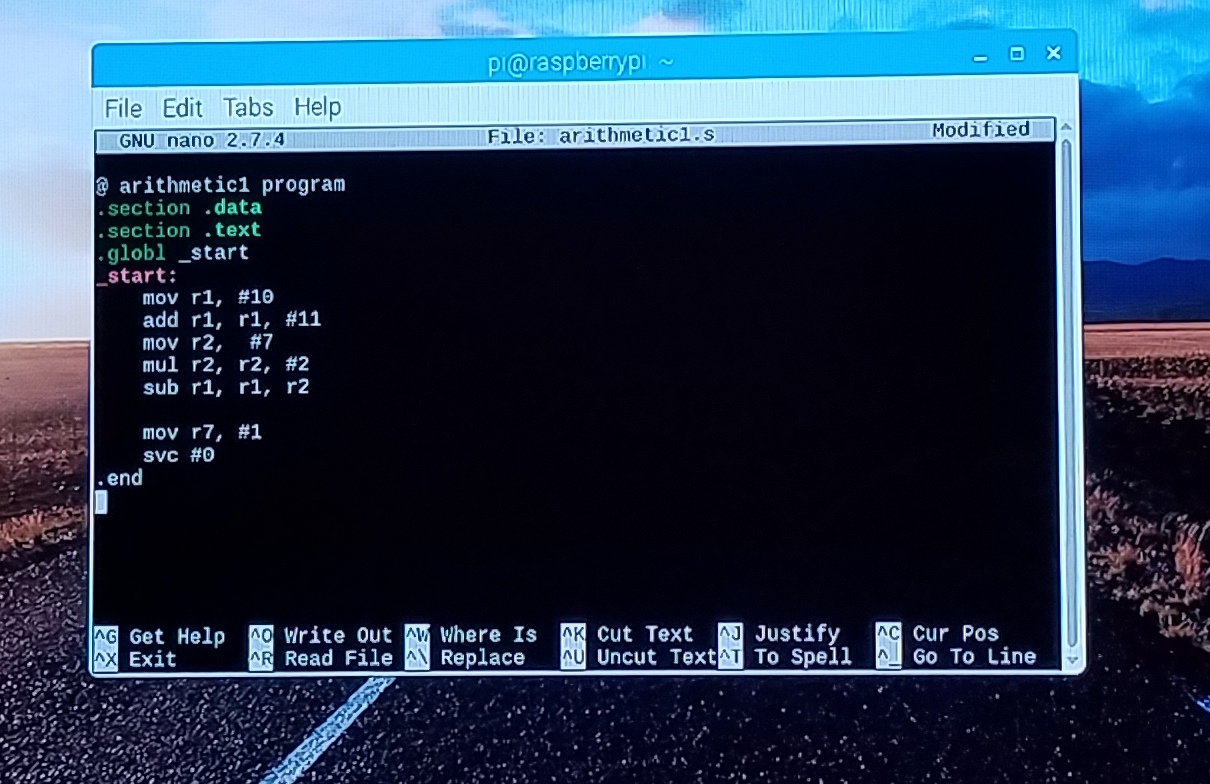
After assembling and loading the first program, everything seemed to run smoothly. However, there were no results echoed to the screen: this is because the code did not produces any output.



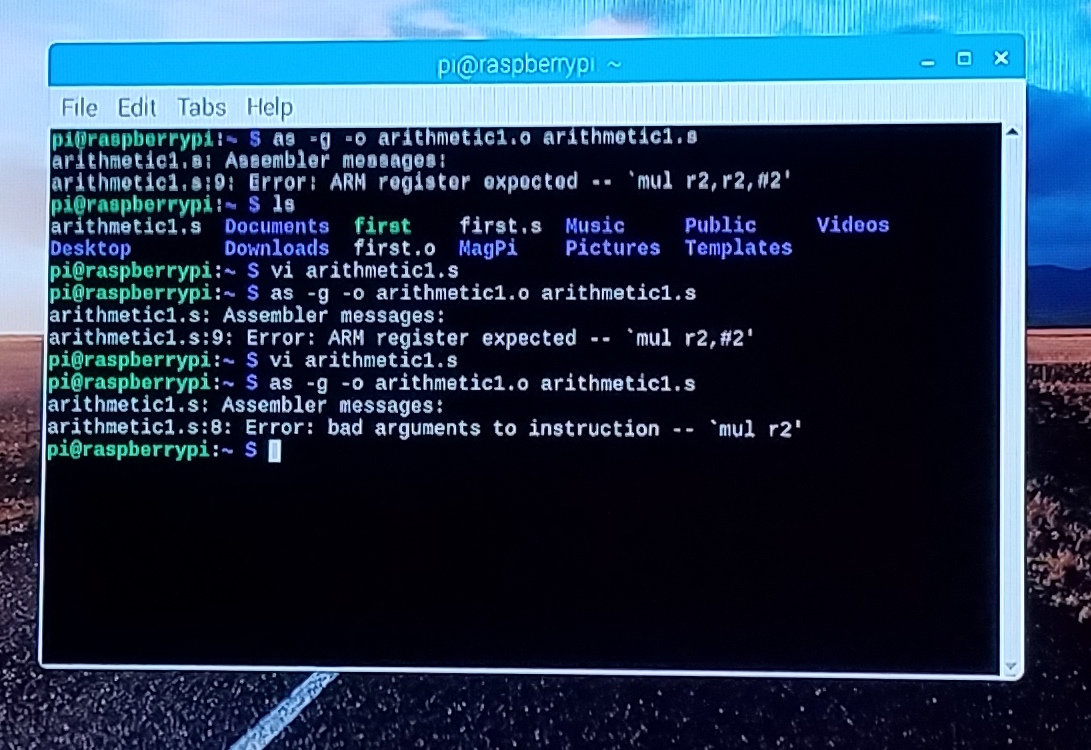
Since running the program did not produce any output, I had to access the GNU Debugger to make sure the code ran properly.



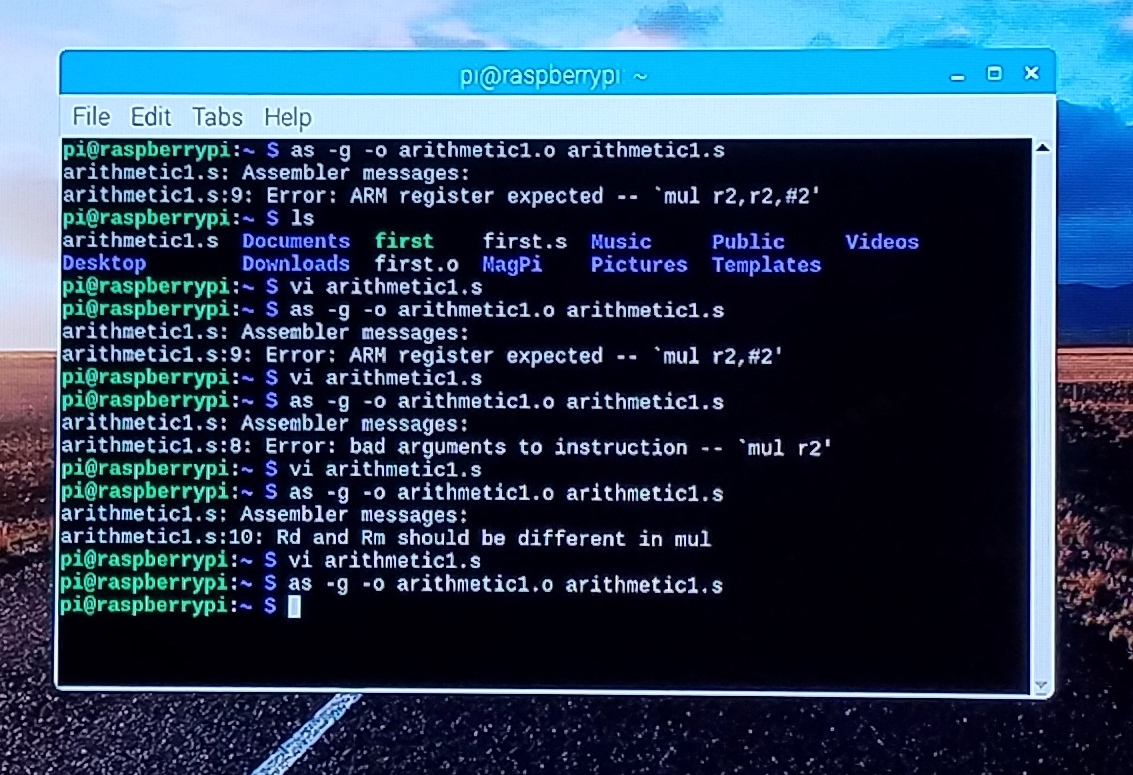
Adding a breakpoint to line 11 allowed us to see the registers at that point of execution. At that point, we can see that r1 is 8, which is the expected value, because 5 minus 1 plus 4 equals 8. In addition, r7 has the value 1. This is also expected, since I set that value in line 10 of our code.



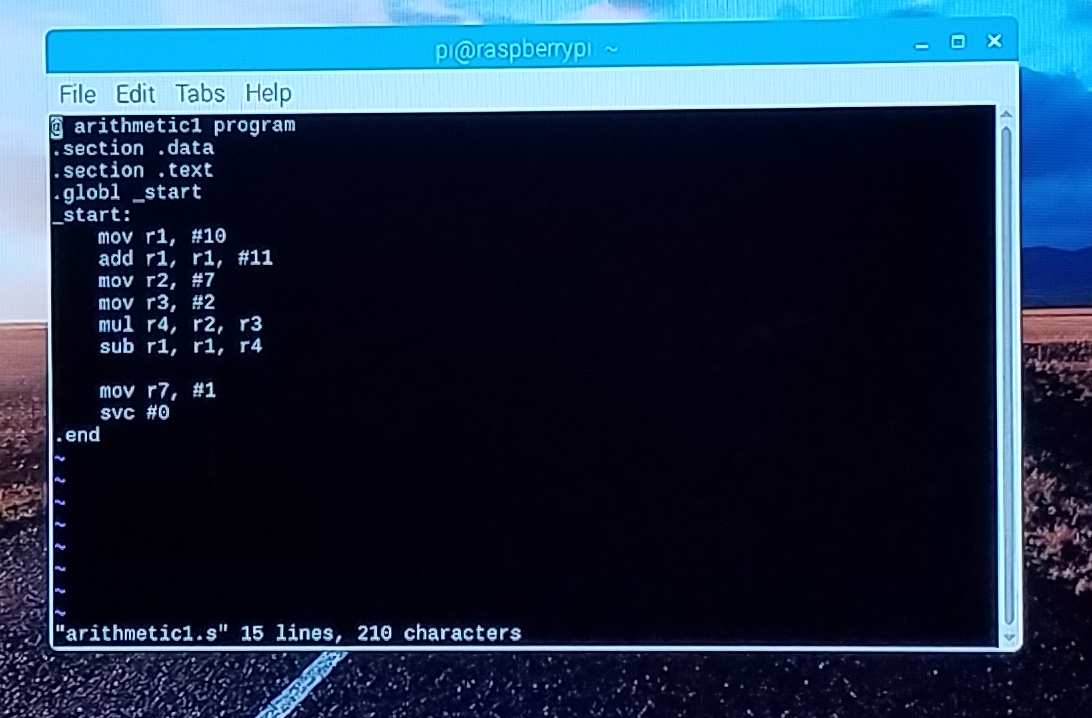
In my first attempt at the arithmetic1 program, I tried to use the an immediate value for the mul function. I assumed that the mul function could take the same parameters as the add function. However, when I tried to assemble the program, I ran into some errors.



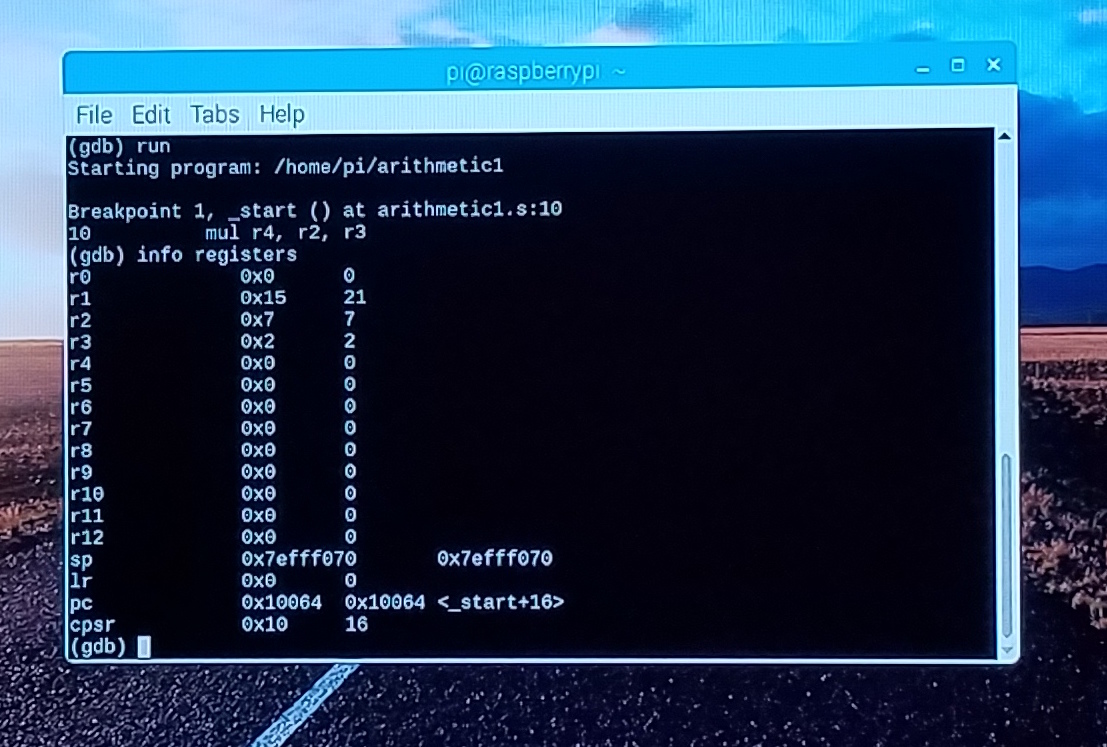
After researching that the mul function took 3 parameters: a destination register and two input registers, rather than an immediate value, I changed the code to use registers to provide the values.



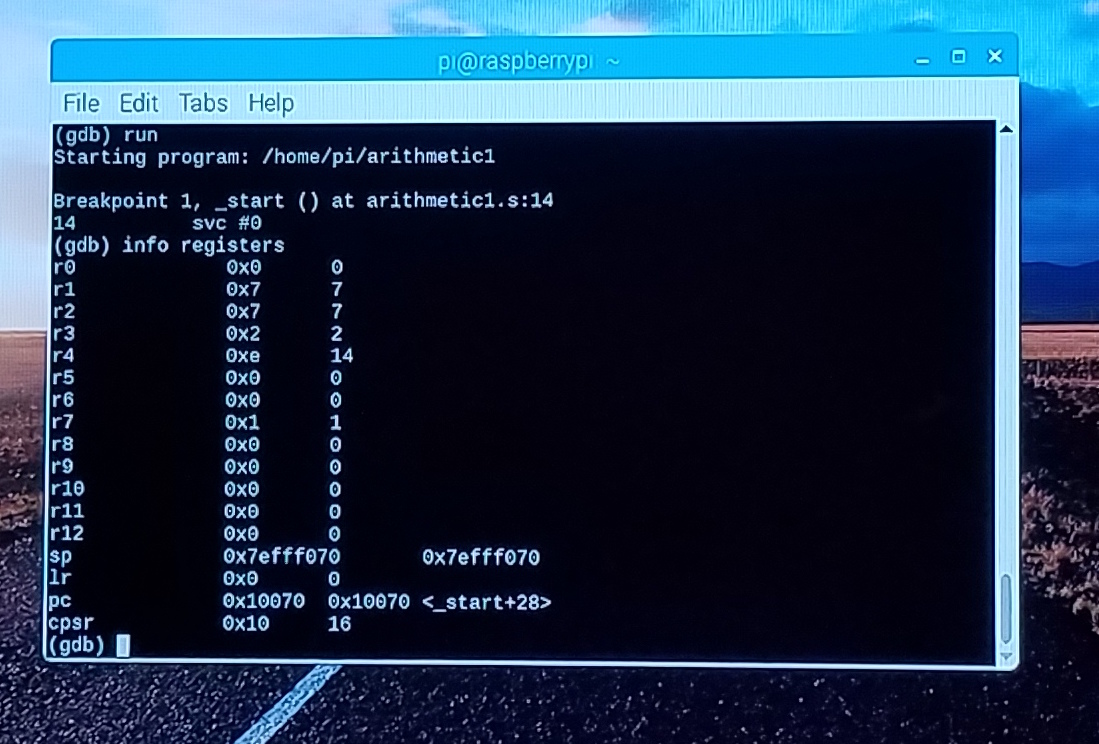
When I tried to assemble the program again, I found that the destination register could not be the same as the input registers.



After fixing those errors, this ended up being my final code. This code did not throw errors when assembled and loaded. However, we need to check the gdb to make sure the code functions as expected.



After running the code with gbd and setting the break point at line 10, the register values make sense. r1 should have a value of 21, because 10+11 is 21. r2 and r3 were both set to 7 and 2 respectively and were unchanged throughout the code, so 7 and 2 make sense.



At the completion of line 14, it makes sense that r4 is 14 because 2 \* 7 equals 14. r1 is now 7 because 21 minus 14 is 7. Also, it makes sense that r7 is 1 because I set that value. The code functions as expected.

@ arithmetic1 program

.section .data

.section .text

.globl \_start

\_start:

mov r1, #10

add r1, r1, #11

mov r2, #7

mov r3, #2

mul r4, r2, r2

sub r1, r1, r4

move r7, #1

svc #0

.end