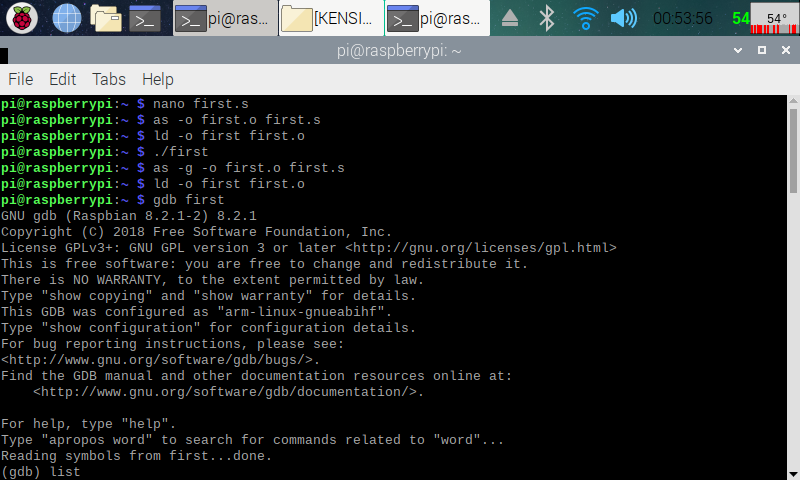
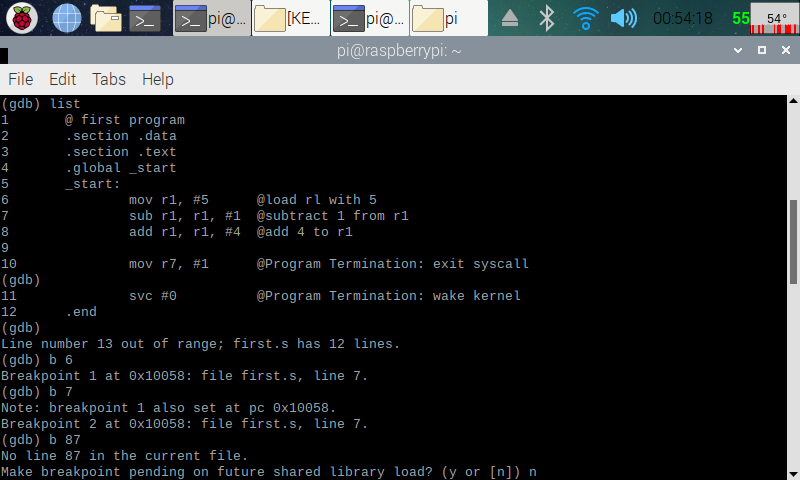
**ARM Assembler in Raspberry Pi**

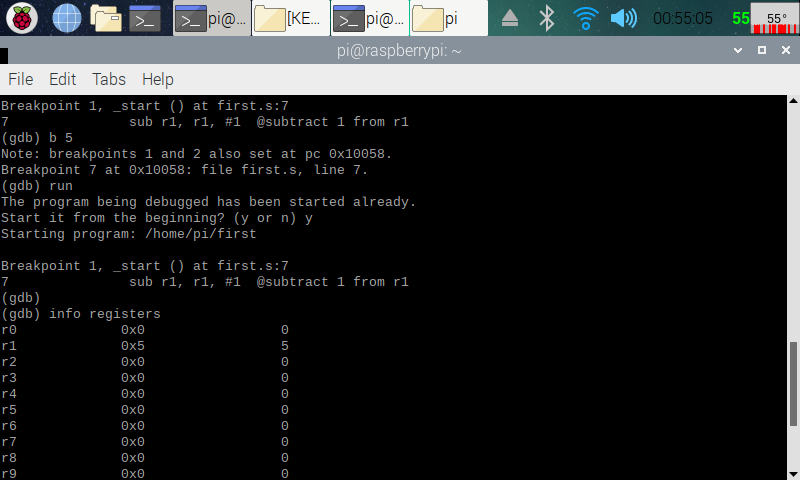
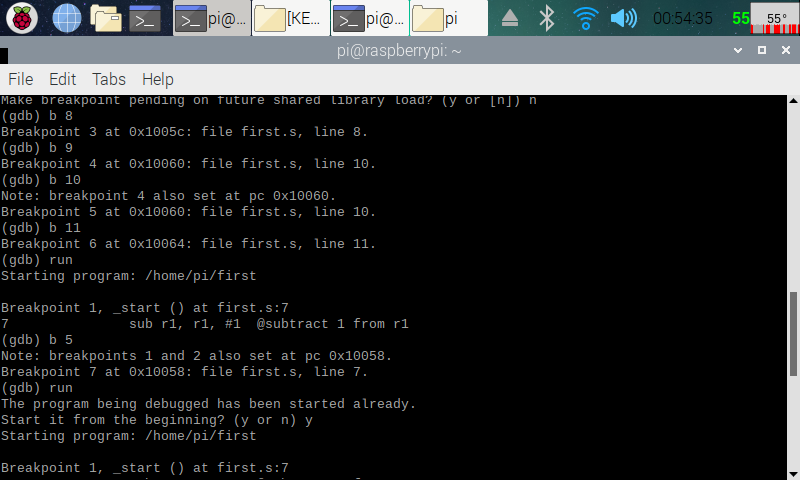
**Part 1: First Program:**



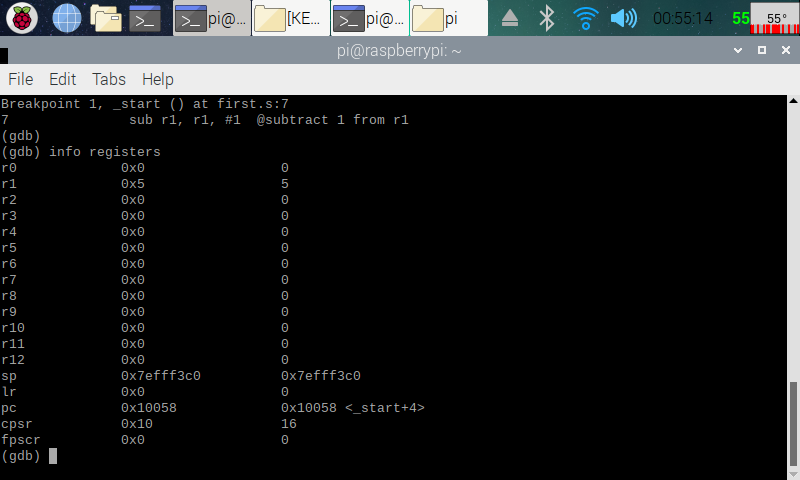
Here (in the screenshot above), I have created the First assmbly program (nano first.s). I then assembled the program (as -o first.o first.s) and linked it (ld -o first first.s). After that, I ran the program (./first), and no output was displayed, because any program that minipulated data between the CPU registers and memory will not use the IO. To see what the program is doing, I used the GDB (GNU Debugger). To do this, I used the debugger to link the machine code to the source code (as -g -o first.o first.s), then assembled it like I did before.i then ran the code with the debugger (gdb first), which displayed some of th debugger information before continueing.



Here (in the screenshot above), I displayed my code in the debugger ((gdb) list), and added breakpoints in between the lines of the codes ((gdb) b “a line number”). This way I can see what is happening in the registry later.

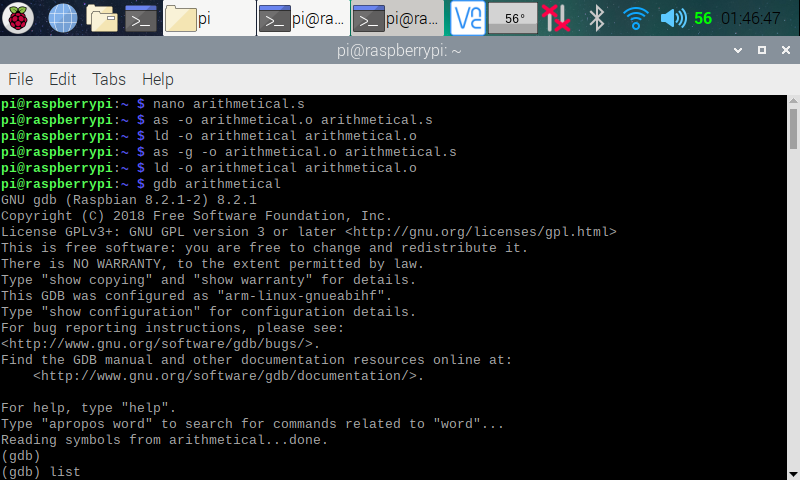


Here (in the two screenshots above), I was playing around with the breakpoints, and after I was done adding breakpoints to the lines that I wanted, I ran the debugger ((gdb) run). The debugger then goes through the first breakpoint. I then pulled up the register information ((gdb) info registers).

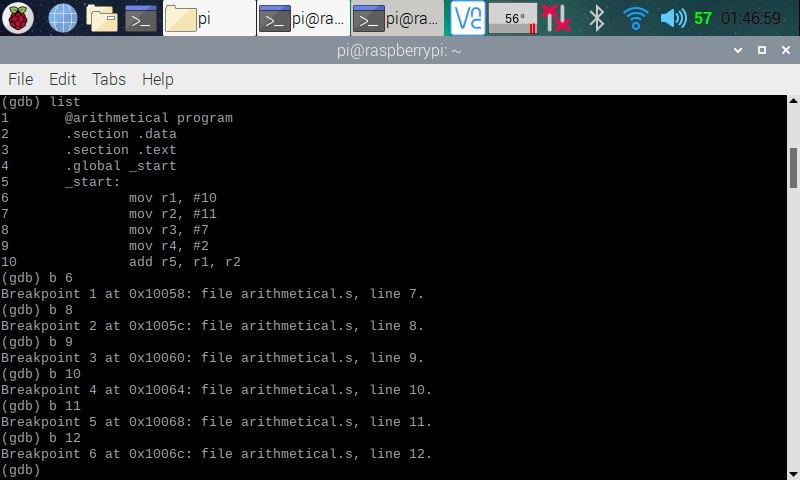


Here (in the screenshot above), I know that the code is working, because after going through the first breakpoint, you can see that 5 was indeed added into register 1 (r1) by the code (mov r1, #5). This is the end of part one.

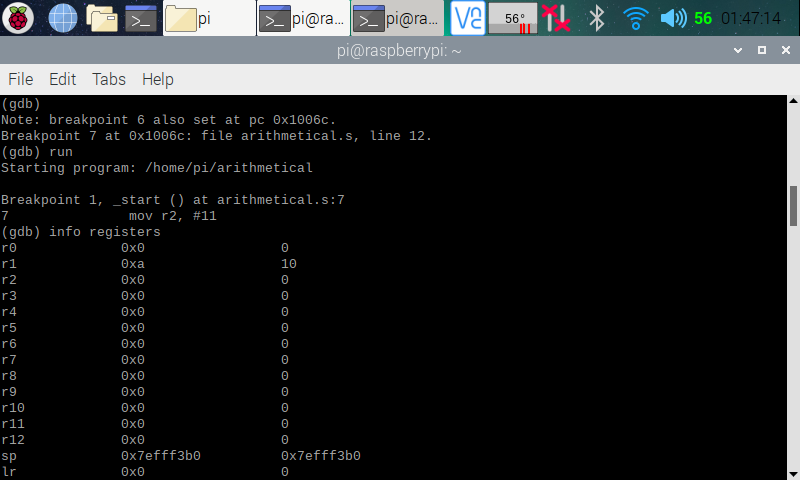
**Part 2: Arithmetical Program:**

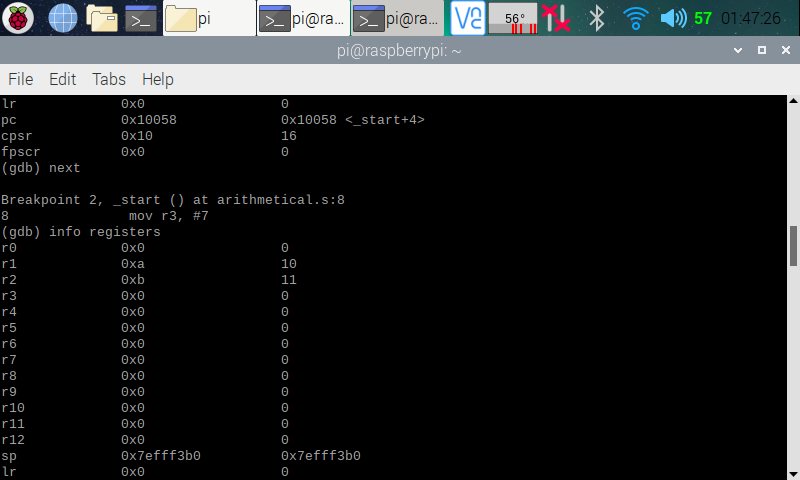
****

Here (in the screenshot above), I made the arithmetical program (nano arithmetical.s). I then assembled, linked and ran it with the GDB debugger like I did with the First program.

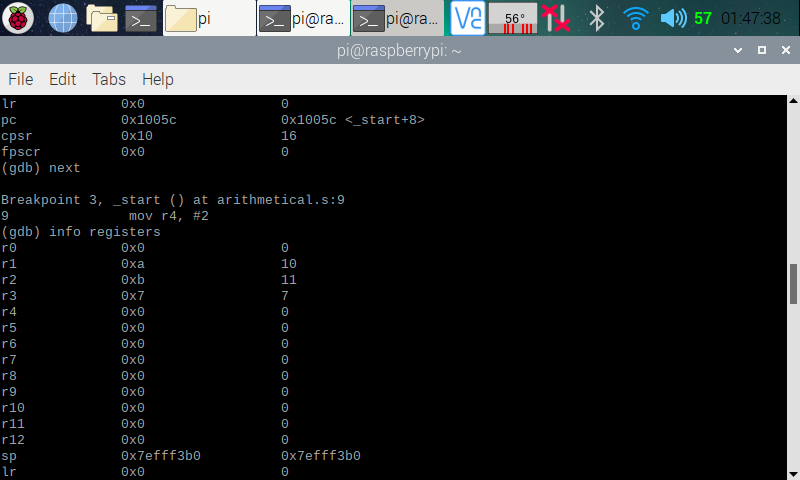


Here (in the screenshot above), I pulled up the code on the debugger and added in the breakpoints like I did in part 1.

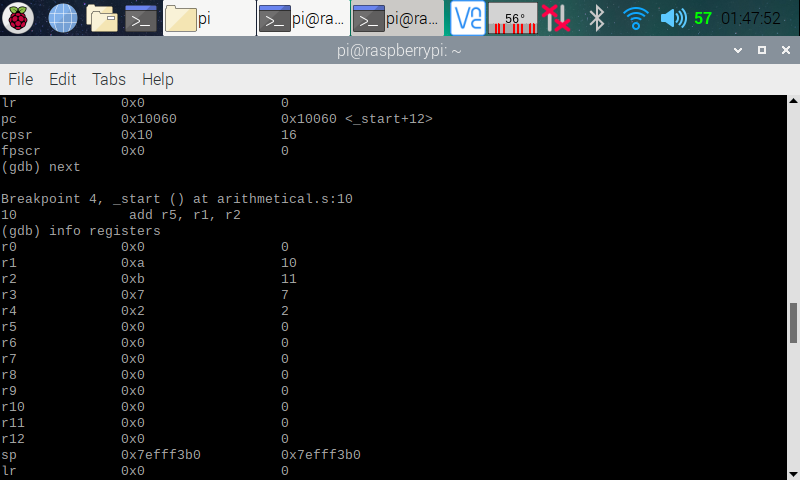
Here (in the screenshot above), I ran the code through the debugger and pulled up the register information. In the register, you see that, after executing line 6 (with the breakpoint moving on to line 7 [(gdb) next]) of the code, the number 10 has been added into register one (mov r1, #10). Register one shows 0xa, with “a” representing 10 in hexadecimal format. On the column over, 10 being showed in decimal format.

****

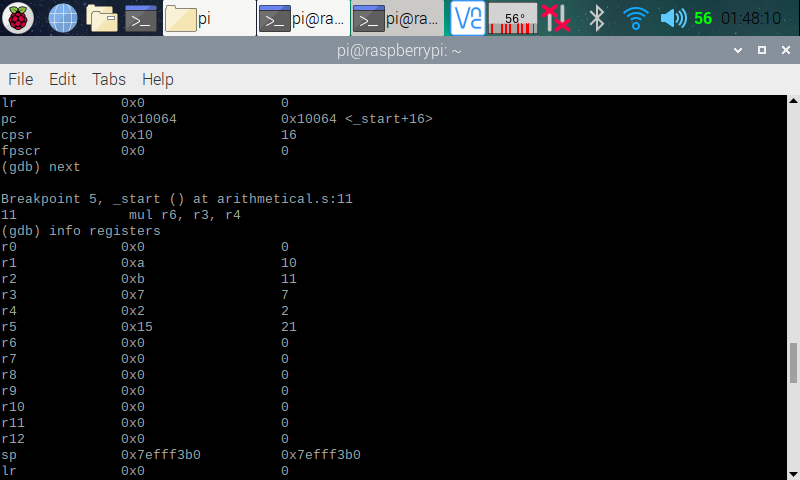
Here (in the screenshot above), in the register, you see that, after executing line 7 (with the breakpoint moving on to line 8) of the code, the number 11 has been added into register two (mov r2, #11). Register two shows 0xb, with “b” representing 11 in hexadecimal format. On the column over, 11 is being displayed in decimal format.



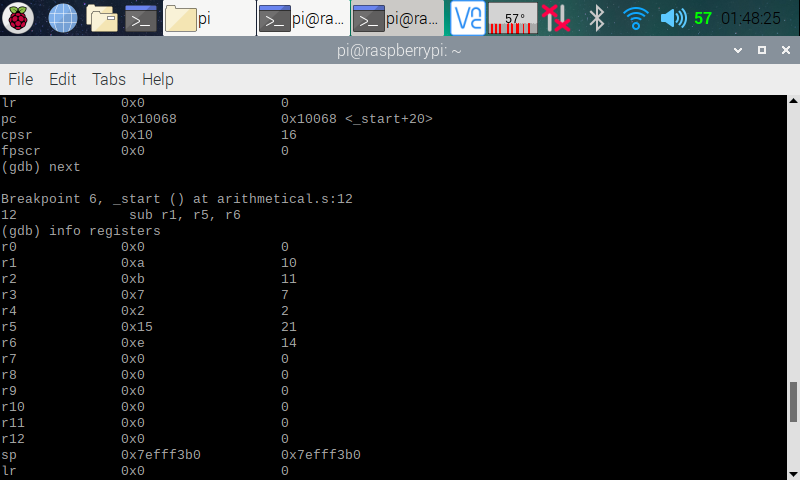
Here (in the screenshot above), in the register, you see that, after executing line 8 (with the breakpoint moving on to line 9) of the code, the number 7 has been added into register three (mov r3, #7). Register three shows 0x7, with “7” representing 7 in hexadecimal format. On the column over, 7 is being displayed in decimal format.



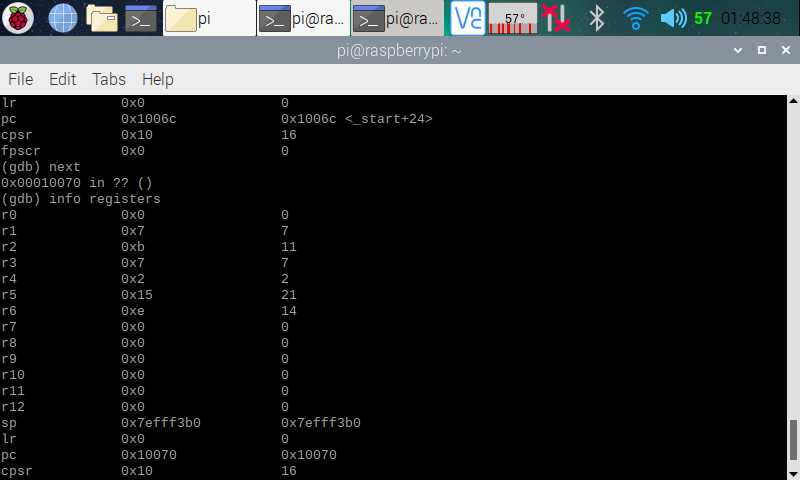
Here (in the screenshot above), in the register, you see that, after executing line 9 (with the breakpoint moving on to line 10) of the code, the number 2 has been added into register four (mov r4, #2). Register four shows 0x2, with “2” representing 2 in hexadecimal format. On the column over, 2 is being displayed in decimal format.



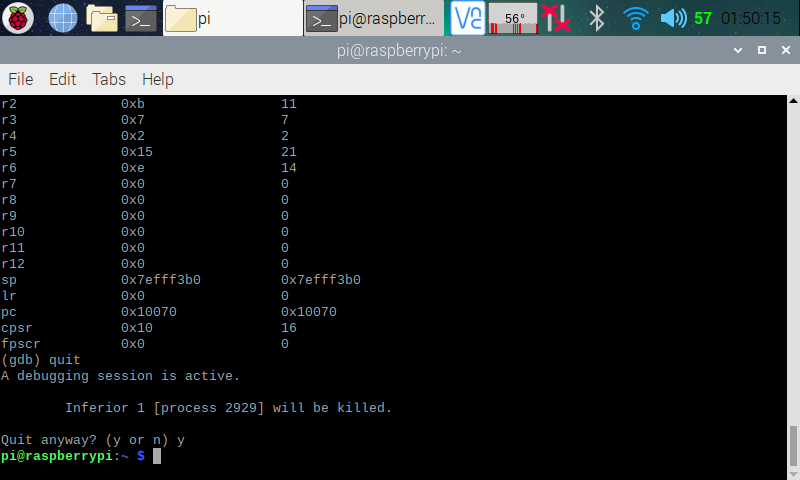
Here (in the screenshot above), in the register, you see that, after executing line 10 (with the breakpoint moving on to line 11) of the code, the number 21 has been added into register five (add r5, r1, r2). Register four shows 0x15, with “15” representing 21 in hexadecimal format. On the column over, 21 is being displayed in decimal format. We are getting 21 because we added r2 (11) to r1 (10), which gave us 21, and stored it into r5.



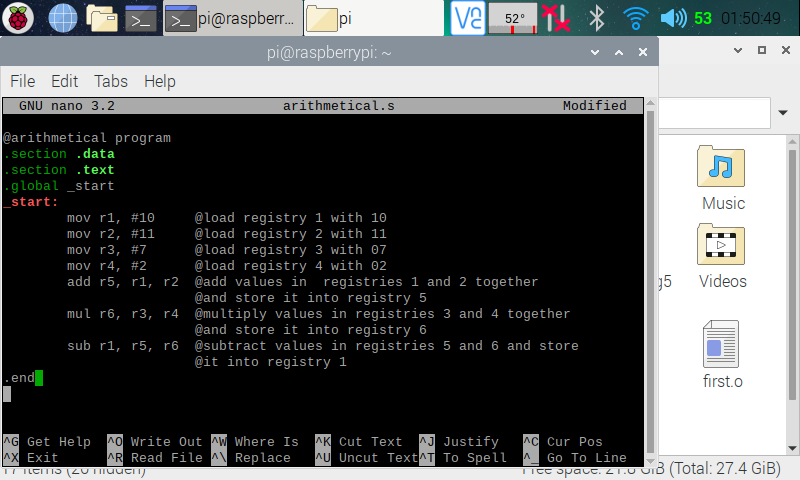
Here (in the screenshot above), in the register, you see that, after executing line 11 (with the breakpoint moving on to line 12) of the code, the number 14 has been added into register six (mul r6, r3, r4). Register six shows 0xe, with “e” representing 14 in hexadecimal format. On the column over, 14 is being displayed in decimal format. We are getting 14 because we multiplied r3 (7) by r4 (2), which gave us 14, and stored it into r6.



Here (in the screenshot above), in the register, you see that, after executing line 12 (with the breakpoint moving on to line 13) of the code, the number 7 has been added into register one (sub r1, r5, r6). Register one shows 0x7, with “7” representing 7 in hexadecimal format. On the column over, 7 is being displayed in decimal format. We are getting 7 because we subtracted r6 (14) from r5 (21), which gave us 7, and stored it into r1.



Here (in the screenshot above), after knowing that the program runs correctly, I left the debugger ((gdb) quit).

****

Here (in the screenshot above), I went back to the arithmetical program and commented through my code. This is the end of part two.