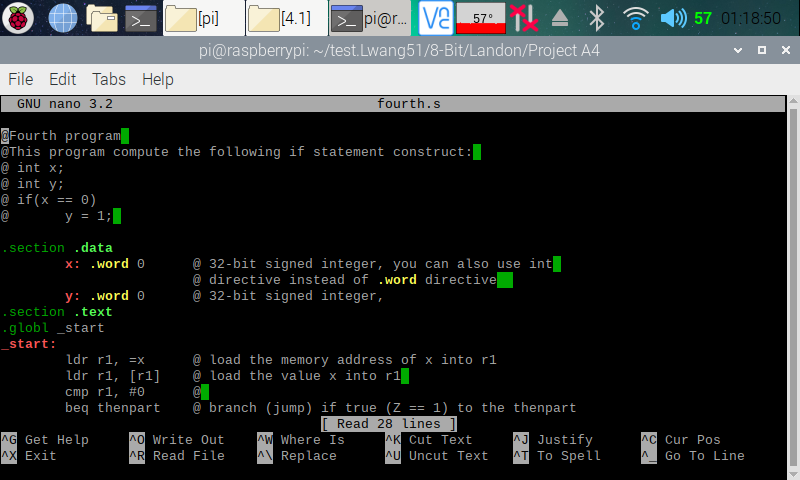
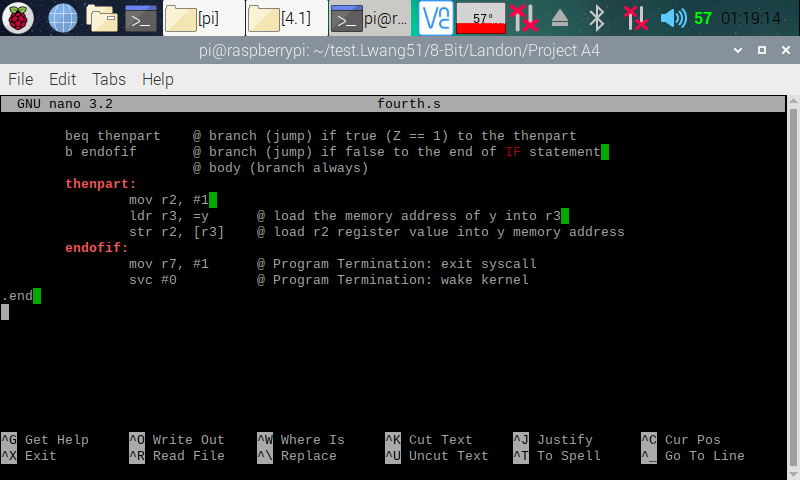
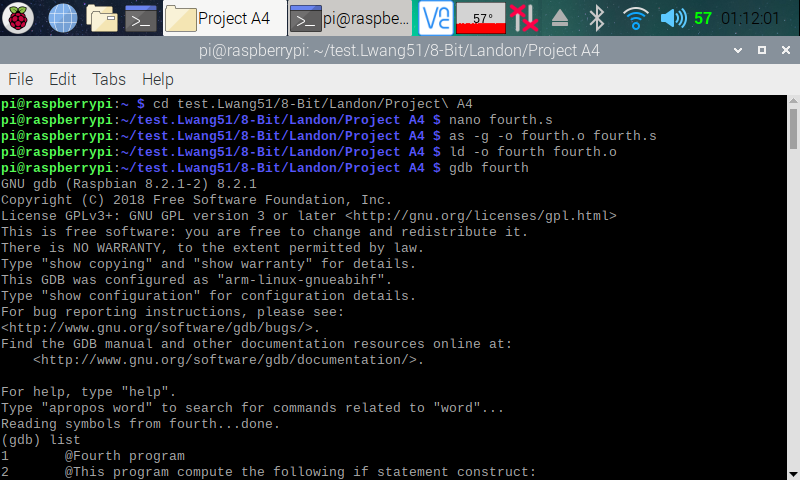
**ARM Assembly Programming**

**Part 1: Fourth Program**

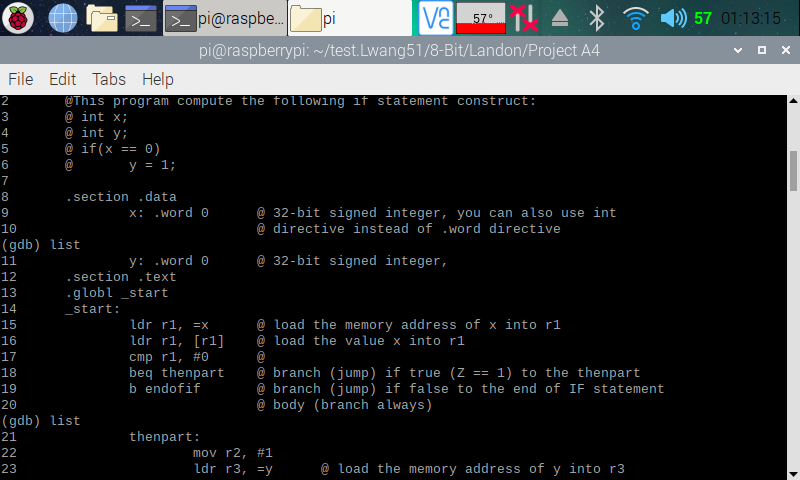


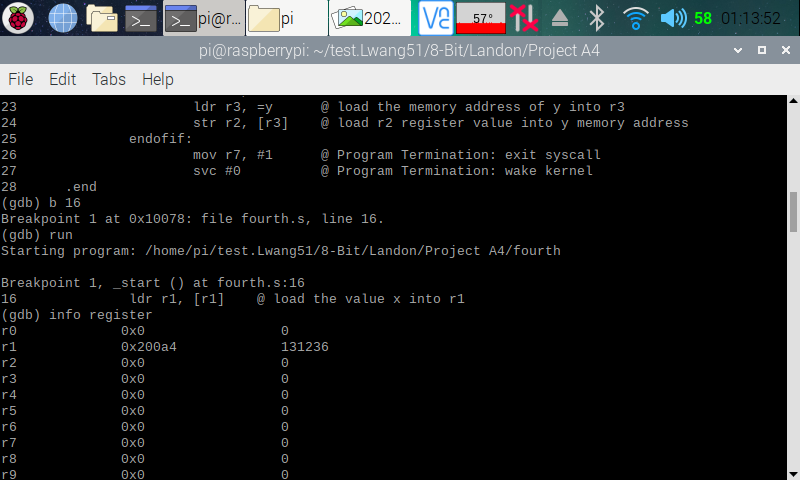


Here (in the two screenshots above), I copied and pasted the codes from the ARM Assembly Programming A4 document and used the nano editor to create a program on my Raspberry PI.

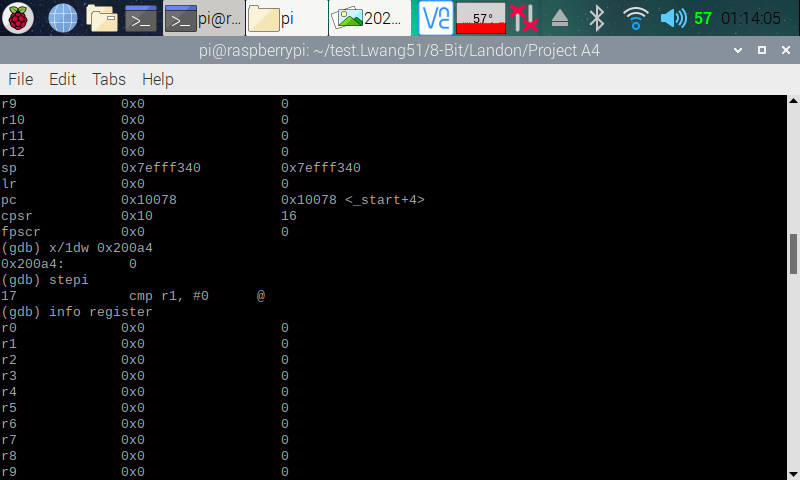


Here (in the screenshot above), I assembled and linked the fourth.s program. I then entered the GDB debugger.

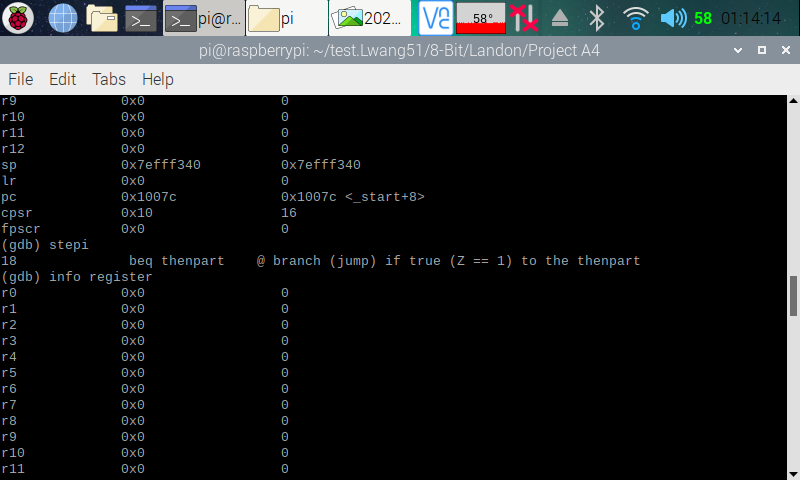


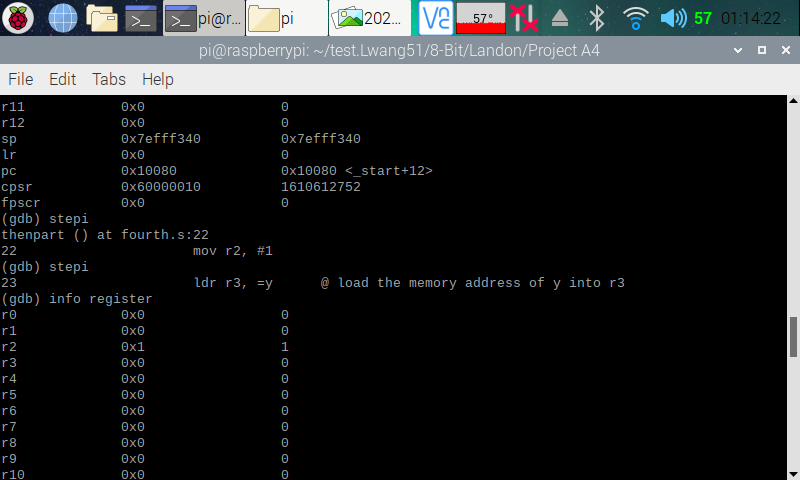


Here (in the two screenshots above), I displayed my program out in the debugger for easy reference, then set a breakpoint at line 16. I then ran the debugger. Once the debugger executes the line 15 of the code, I pulled up the register information, and we can see that a memory address has been loaded onto r1 as expected from the code (ldr, r1, =x).

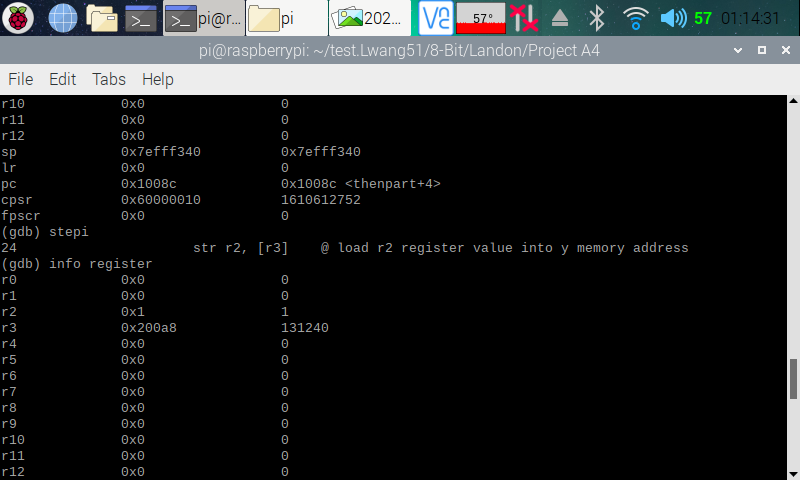


Here (in the screenshot above), I pulled up the memory of the memory address that was loaded onto r1, and we can see that it points to 0 as expected, as we assigned the number 0 to variable ‘x’. I then stepped over to the next line so that line 16 (ldr r1, [r1]) can execute. I then pulled up the register information, and we can see that 0 has now been loaded onto r1.

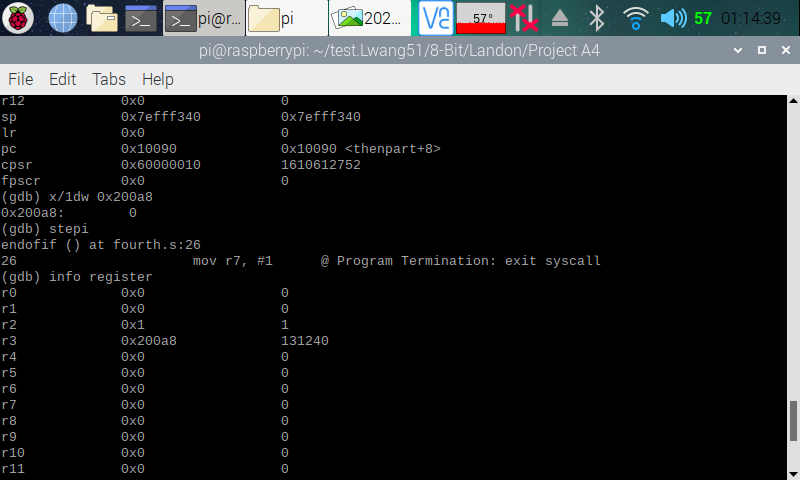




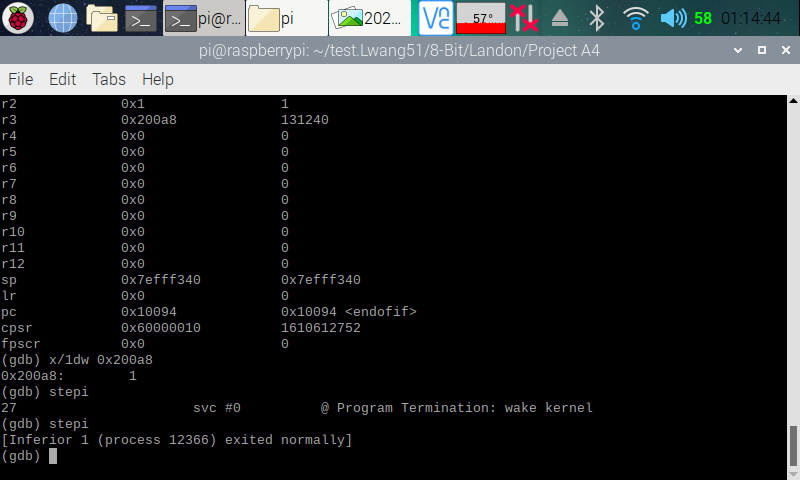
Here (in the screenshot above), I stepped over to the next line so that line 17 can (cmp r1, #0) can run. I then pulled up the register information. Looking at the CPSR flag register, we can see that the zero flag (second bit), has been raised. This means that r1 equals 0, which is true, because 0 = 0. I then stepped over to the next line so that line 18 (beq thenpart) can execute. We can see that the program is executing correctly, because it branched to thenpart to set a breakpoint instead of continuing with setting the breakpoint at line 19 (b endofif). This is because we have beq, which means branch if equal, in which it was. I then stepped over to the next line, so that line 22 (mov r2, #1) can execute. I pulled up the register information, and we can see that the number one has been moved to r2.



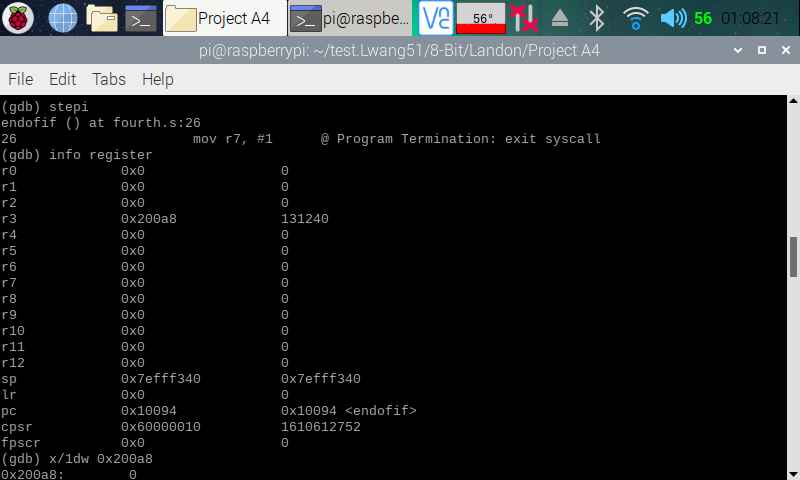
Here (in the screenshot above), I stepped over to the next line so that line 23 (ldr r3, =y) can execute. I pulled up the register information, and we can see that a memory address has indeed been loaded onto r3.



Here (in the screenshot above), I pulled up the memory with the memory address in r3, and we can see that it is pointing to 0. This was expected, because we assigned the number 0 to variable ‘y’. I then stepped over to the next line so that line 24 (str r2, [r3]) can execute. I changed up the code here a little, because I believe there was some issue. The original code for line 24 was ldr r2, [r3], which did not produce the correct output as stated in the comment (this is shown at the end of this section). I then pulled up the register and we can see that there is a memory address in r3.

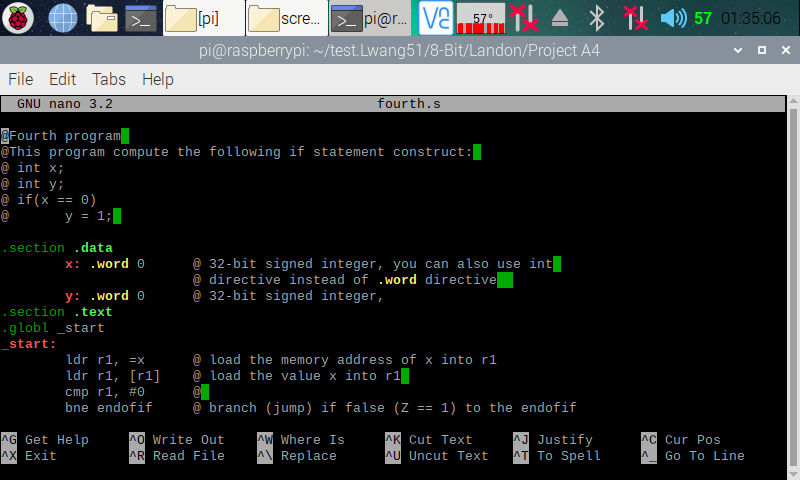


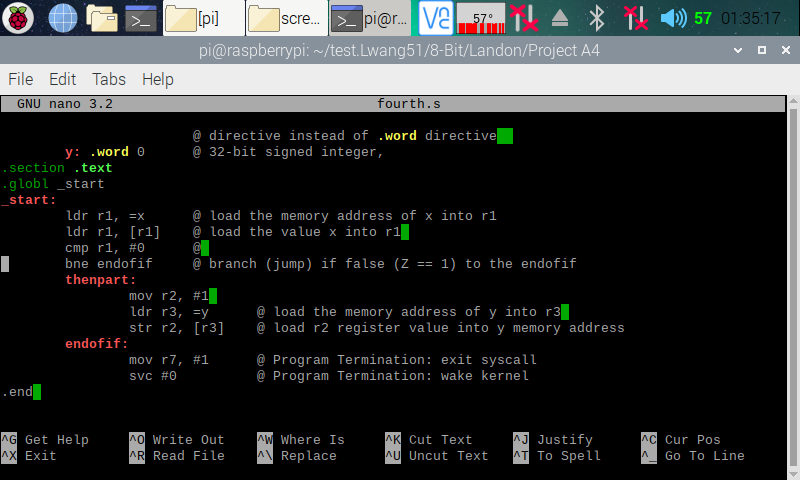
Here (in the screenshot above), I pulled up the memory using the memory location in r3, and we can see that it now stores a 1 instead of the 0. We now know that our program is executing correctly.



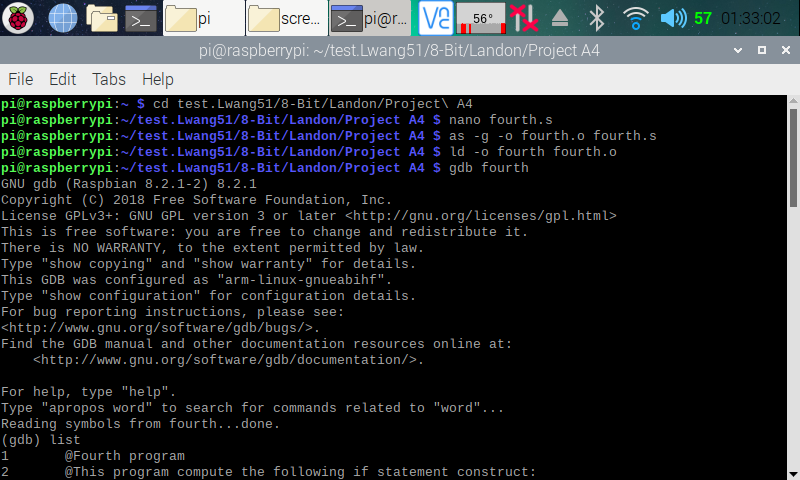
Here (in the screenshot above), is the register from the program when line 24 had ldr r2, [r3] instead of str r2, [r3]. I pulled up the memory register and the memory and we can see that the memory address points at a 0, which was not what the comment wanted the program to do. The comment stated to load r2 register value (which should be 1) into ‘y’ memory address. However, ldr r2, [r3] will load the r3 value, which is 0 onto r2, hence the reason why we are seeing a 0 in r2 instead of 1. Because of this, I suspected that there was a typo of some kind, so I changed up the code.

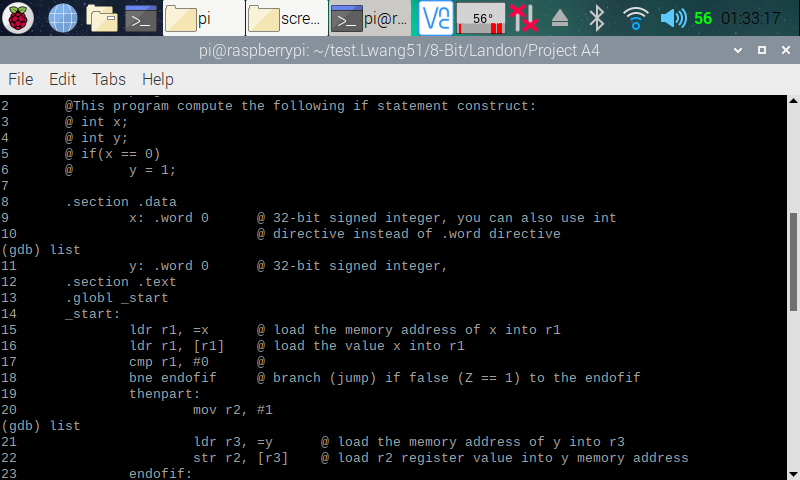
**Part 1: Fourth Program (Improved)**



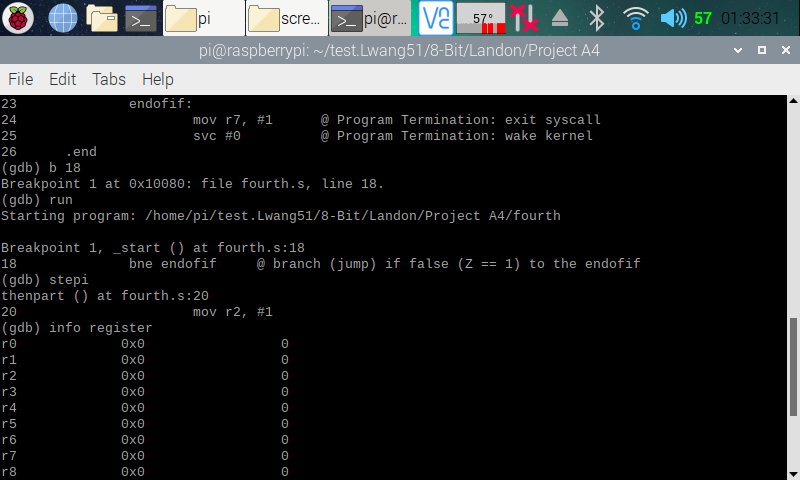


Here (in the two screenshots above), I copied and pasted the codes from the ARM Assembly Programming A4 document and used the nano editor to create a program on my Raspberry PI.

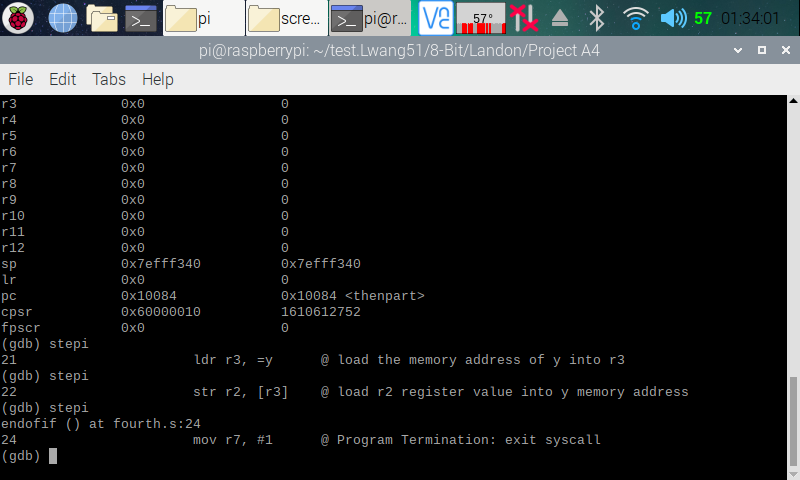




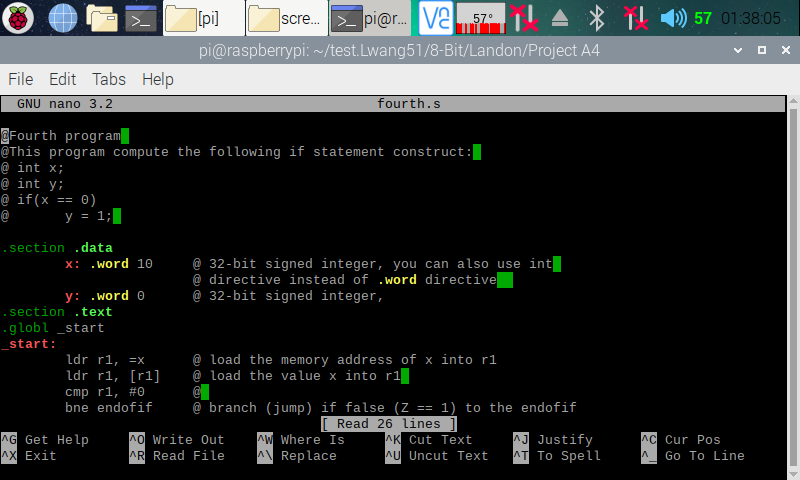
Here (in the two screenshots above), I reassembled and relinked the fourth program after changing “BEQ” to “BNE” and removing the original line 19 (b endofif) in the code. I then entered the GDB debugger and displayed the code for easy reference.



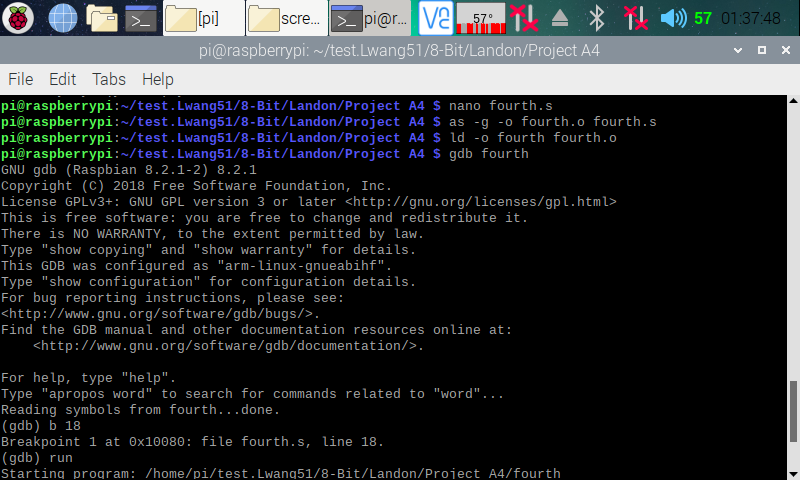
Here (in the screenshot above), I set the breakpoint to line 18, since we already proved earlier that the codes before line 18 works perfectly and that we only need to check the coded after line 17. I then ran the program and stepped over to the next line of code so that line 18 (bne endofif) can execute. We can see that it did not jump, because the zero flag is raised in the CPSR register (as shown earlier). This means that r1 and 0 are equal, and they are (0 = 0). It will only branch if r1 and 0 are not equal.



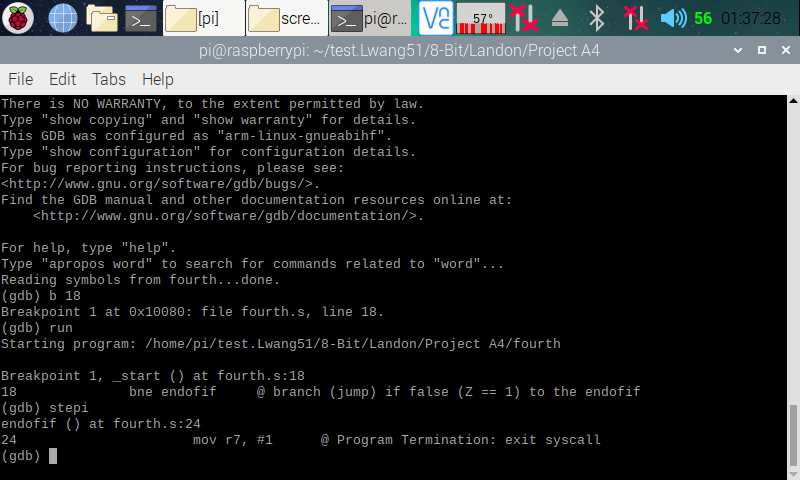
Here (in the screenshot above), I ran through the rest of the program, and everything ran as expected and like the old version of the program.



Here (point he screenshot above), I I went back to the code and changed the variable ‘x’ to equal 10 instead to 0 to verify that my code is running correctly.

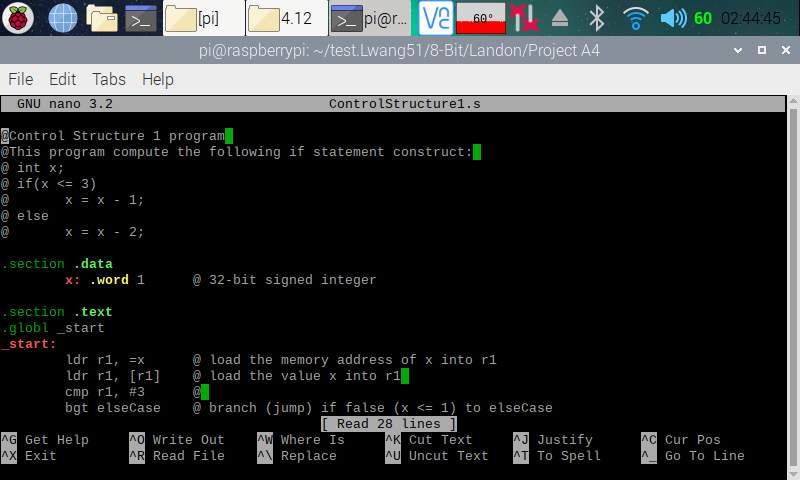


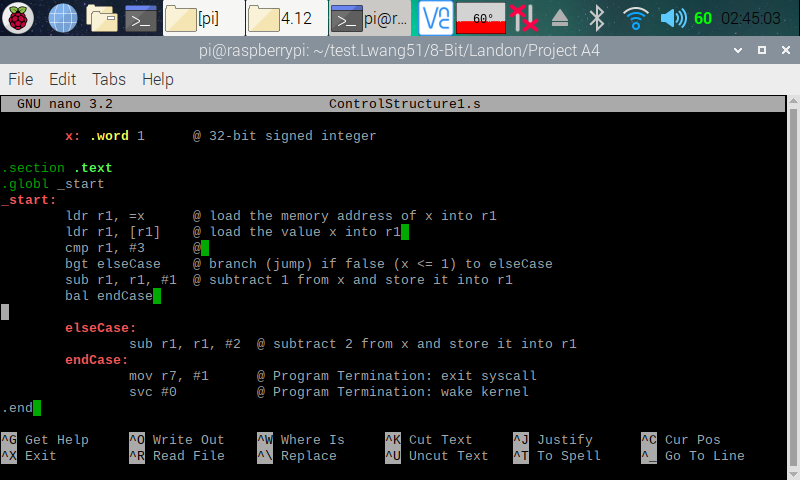
Here (in the screenshot above), I reassembled and relinked my program and entered the GDB debugger. I set a breakpoint at line 18 and ran the program.



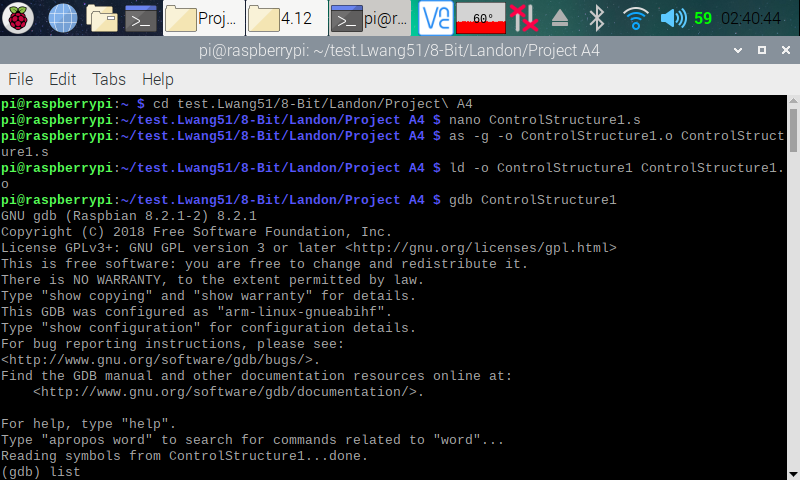
Here (in the program above), I stepped over to the next line, and we can see that the program branched to endofif, because the zero flag in the CPSR register was not set, meaning r1, which stores x (10) does not equal 0, and this is correct (10 != 0). I now know that my program is running correctly.

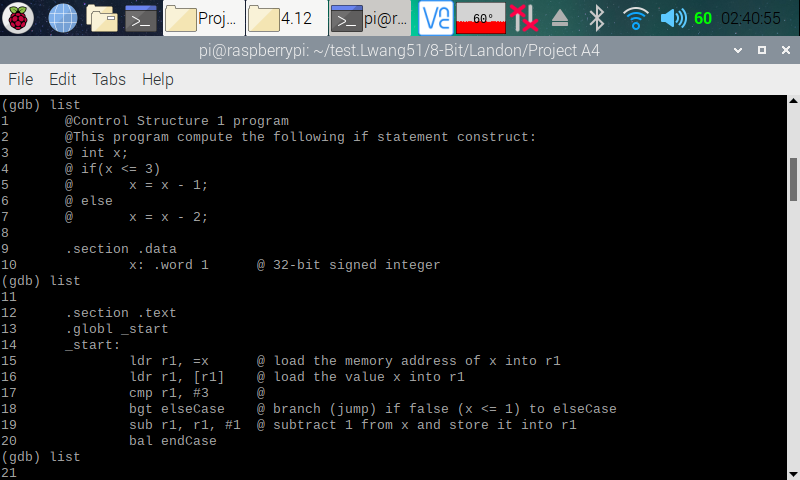
**Part 3: Control Structure 1 Program**

****

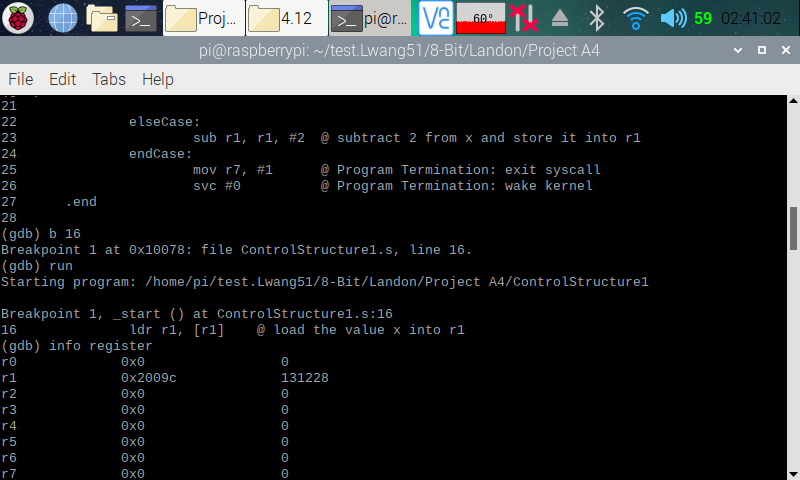
****

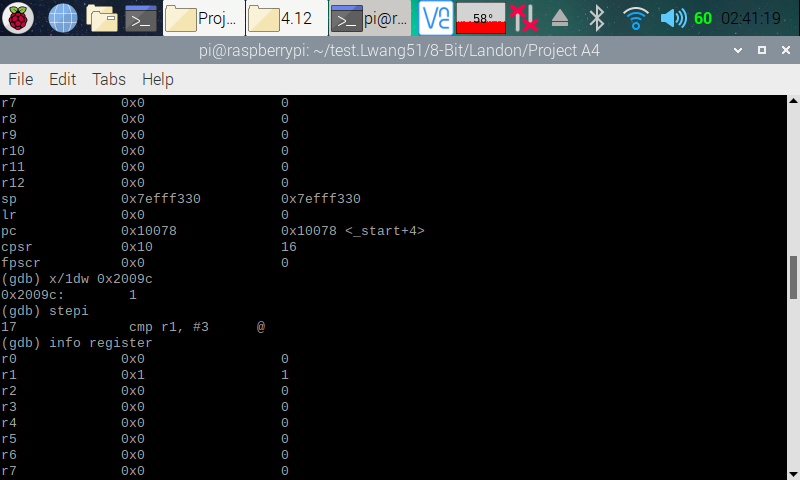
Here (in the two screenshots above), I copied and pasted the codes from the ARM Assembly Programming A4 document and used the nano editor to create a program on my Raspberry PI.



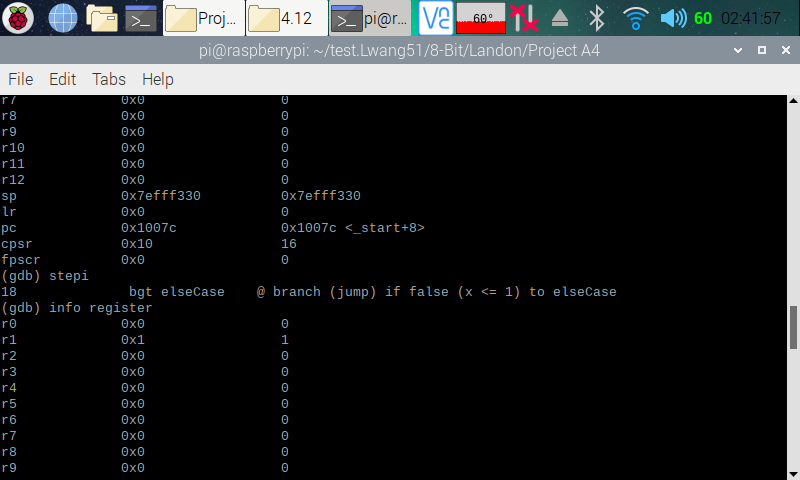


Here (in the screenshot above), I assembled and linked the ControlStructure1.s program. I then entered the GDB debugger. In the debugger, I displayed my code for easy reference.

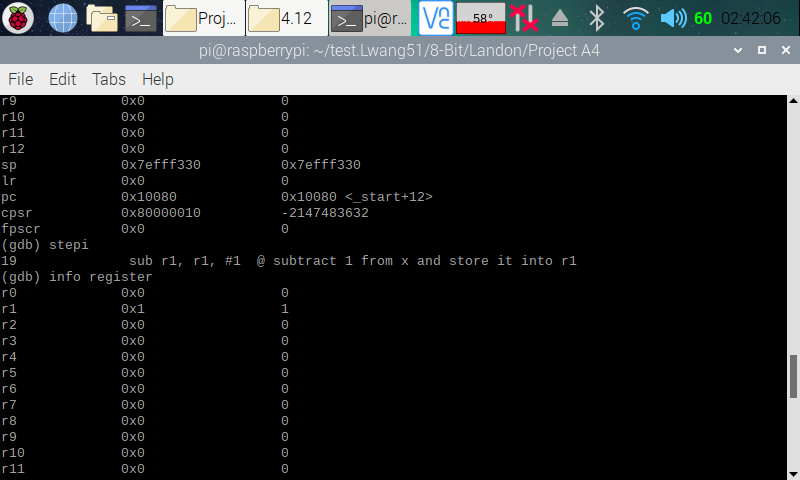




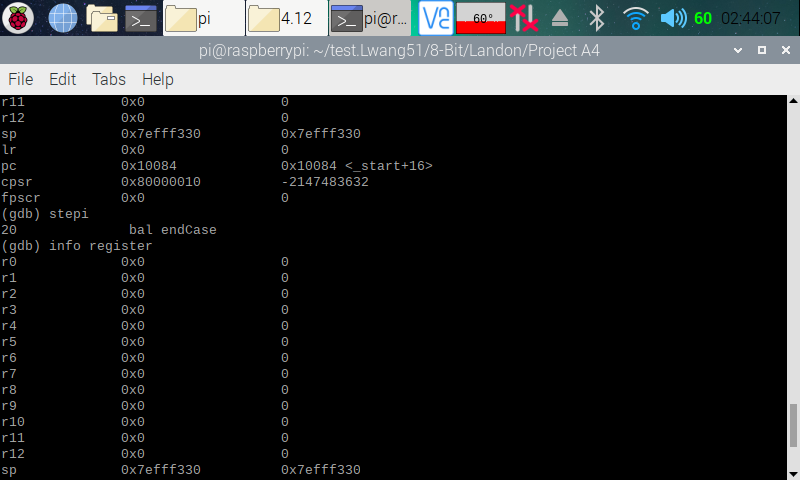
Here (in the two screenshots above). I set a breakpoint on line 16 and ran the program. I then pulled up the register information, and we can see that a memory address has been loaded to r1. I then pulled up the memory using the memory address, and we can see that it is currently pointing to 1. This was as expected, because we assigned the number 1 t variable ‘x’. I then stepped over to the next line so that line 16 (ldr r1, [r1]) can execute. I then pulled up the register information, and we can see that 1 is now loaded onto r1.



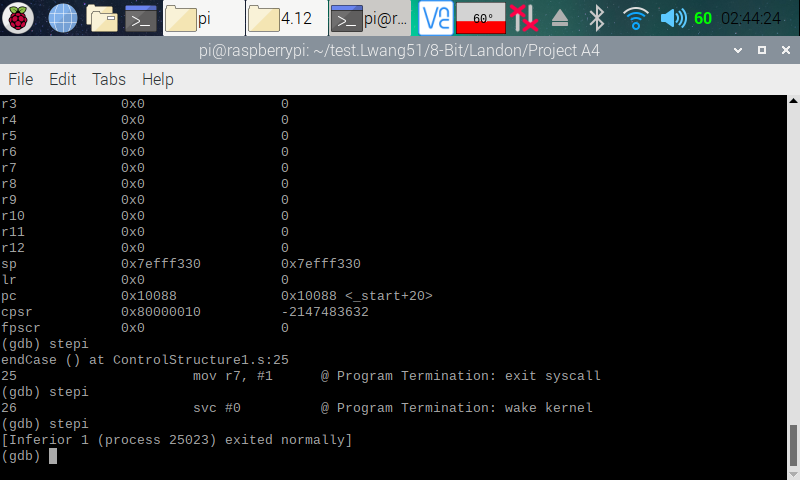
Here (in the screenshot above), I stepped over to the next line so that line 17 (cmp r1, #3) can execute. I then pull up the register information.



Here (in the screenshot above), I can see that the zero flag is set in the CPSR flag. I then stepped over to the next line so that line 18 (bgt elseCase) can execute. The program did not branch, because 1 was not greater than 3. As a result, the breakpoint was set on line 19.



Here (in the screenshot above), I stepped over to the next line so that line 19 (sub r1, r1, #1) can execute. I pulled up the register, can we can see that r1 now stores 0 in it as expected (1 – 1 = 0). The x value in hex is 0, and the Z flag is 1.



Here (in the screenshot above), I continued with the program until it hits an end. We have 1 stored in r1, and now I know my program is running correctly.