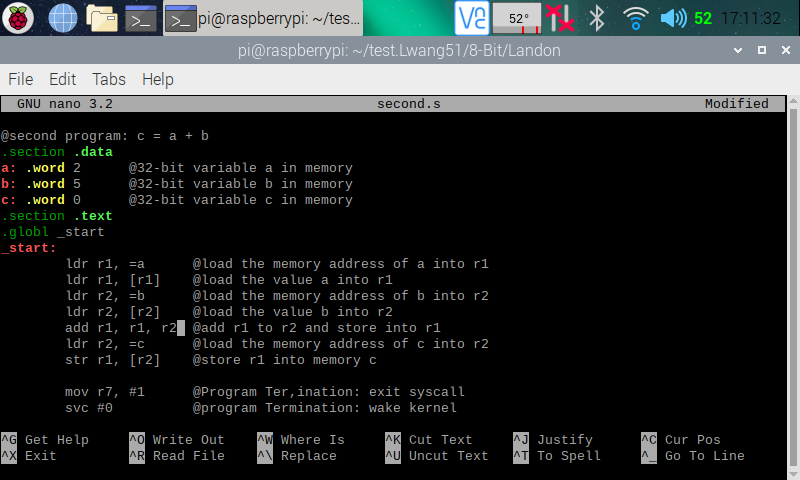
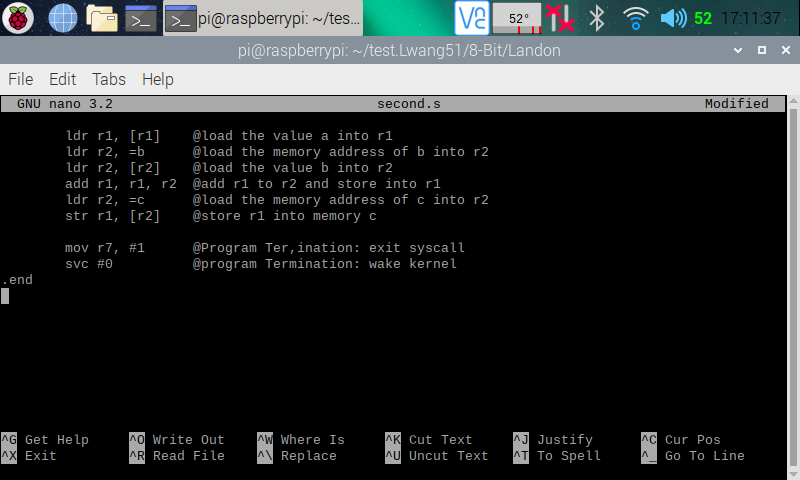
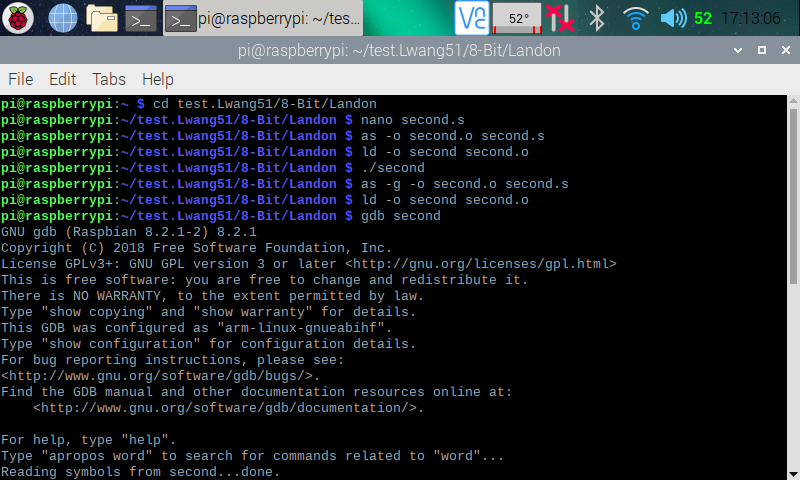
**ARM Assembly Programming**

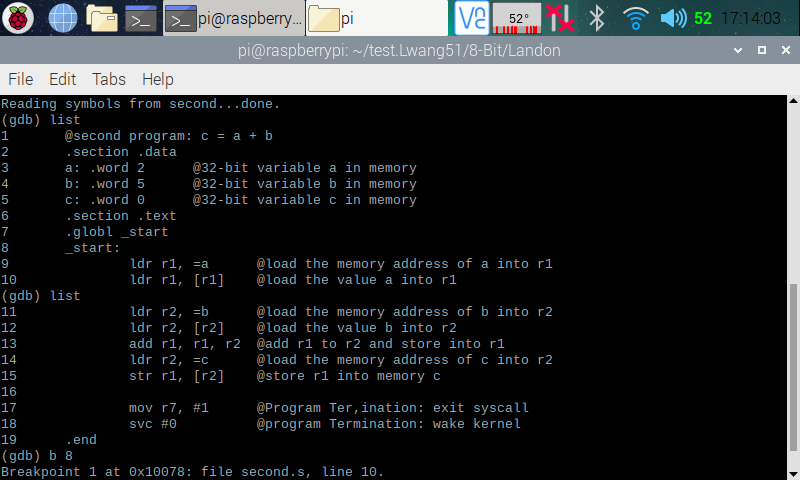
**Part One: Second Program:**



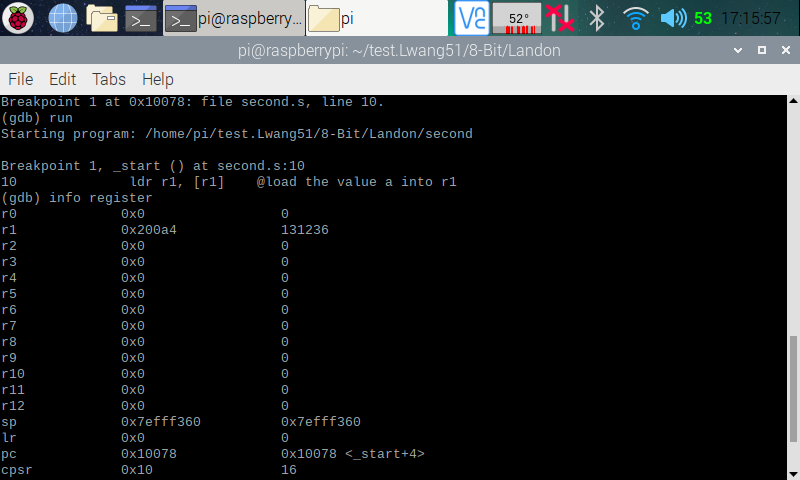




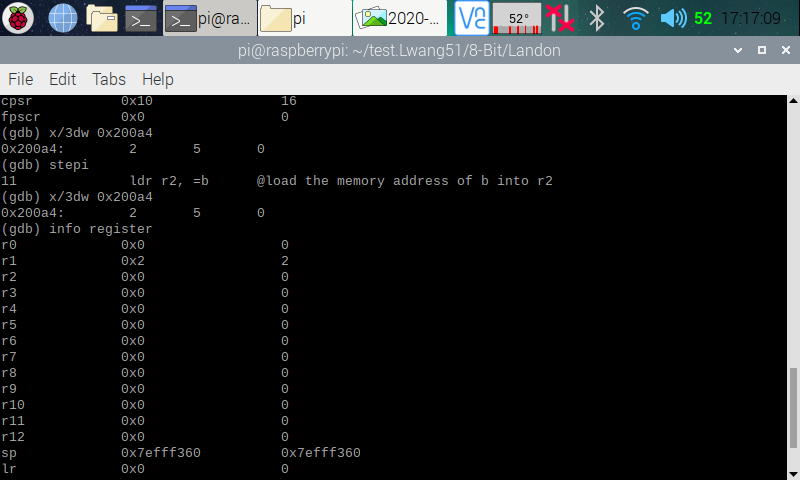
Here (in the three screenshots above), I created the Second program using the nano editor. I then assembled and linked the programs using the instructions **as -o second.o second.s** and **ld -o second second.o** respectively. After that, I ran the program using the instruction **./second,** but nothing is shown, because data was only manipulated between the CPU registers and memory. To see if my program is running correctly, I went into the debugger using the instruction **gdb second**.



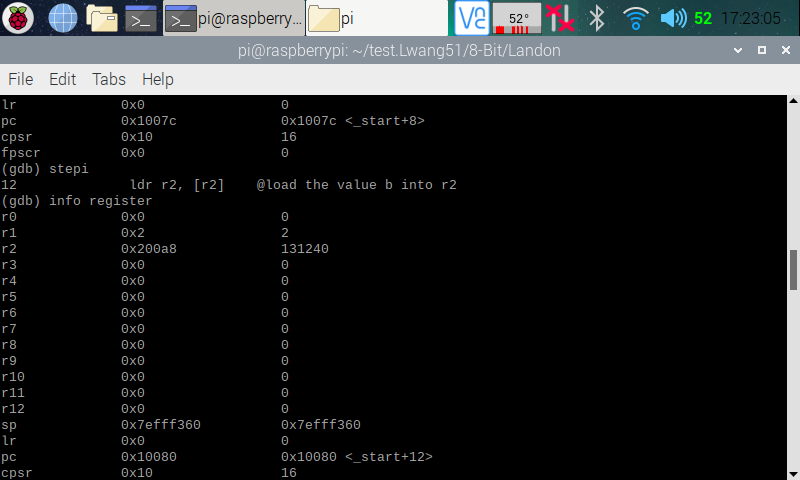
Here (in the screenshot above), I used **list** to show my program instructions, then set a breakpoint at line 8 using **b 8** (which automatically moved the breakpoint to line 10).



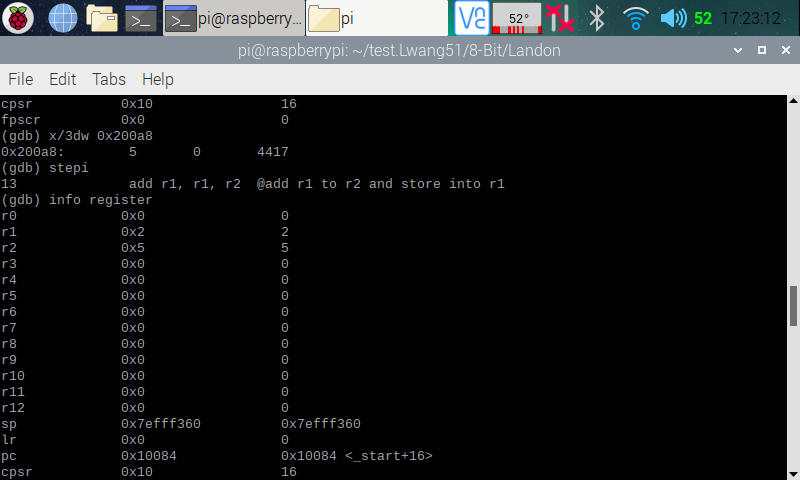
Here (in the screenshot above), I ran the debugger using **run**, and it stopped at line 10 where the breakpoint is set. I then pulled up the register using **info register**, and in the register, we can see the line 9 of the program executed correct, because a memory address (0x200a4) was loaded onto register 1.



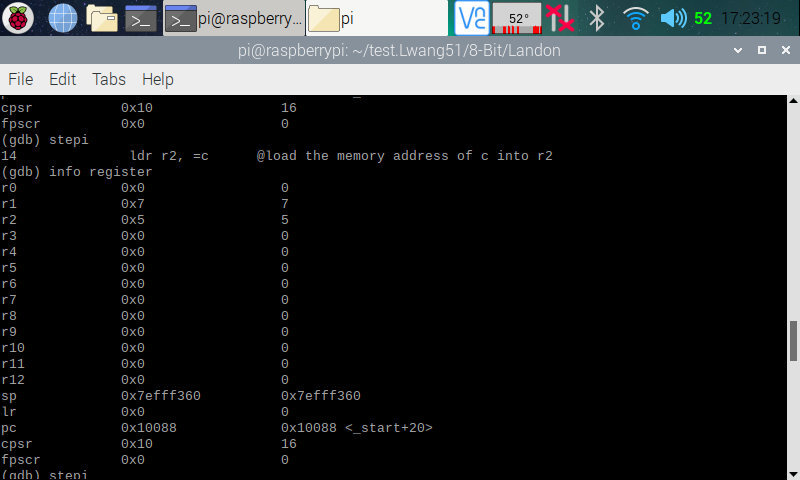
Here (in the screenshot above), I went into the memory to double check if it was pointing to the right memory address using **x/3dw 0x200a4**. The memory pulled up contained 2 5 0, which was the numbers we initialized a, b, and c with. We now know that the right memory address was loaded. I then stepped over to the next line (line 11) of code using **stepi**, so that line 10 (which loaded value a into r1) will run. I pulled up the register information, and we can see that 2 have been loaded into r1.



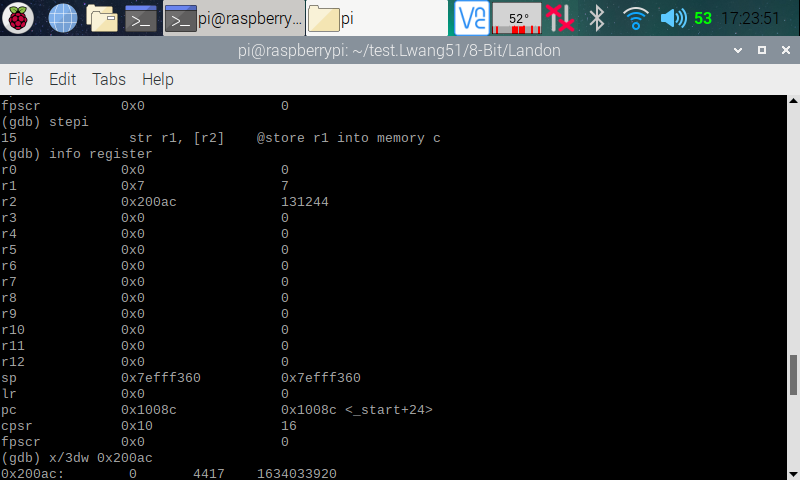
Here (in the screenshot above), I stepped over to the next line (line 12), so that line 11 (which loaded memory address of b into r2) will run. I pulled up the register information, and we can see that a memory address (0x200a8) has indeed been added into r2.



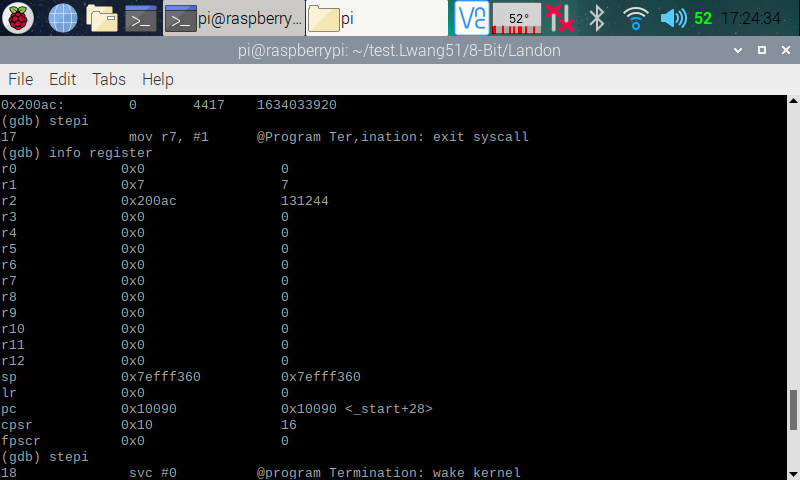
Here (in the screenshot above), I pulled up the memory that was just loaded onto r2 using the memory address (0x200a8), and we can see that it is pointing to the right part of the memory with 5 0 4417 in it. I then stepped over to the next line (line 13) of the code, so that line 12 (which loads b’s value into r2) will run. I pulled up the register information, and we can see that r2 is now loaded with 5.



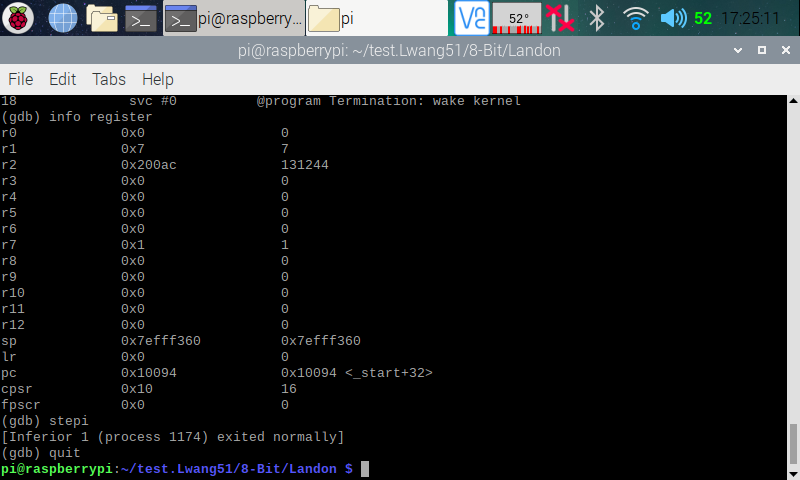
Here (in the screenshot above), I stepped over to the next line (line 14), so that line 13 (which adds r2 to r1) will run. I pulled up the register information, and we can see that 7 (from 2 + 5) is now stored in r1.



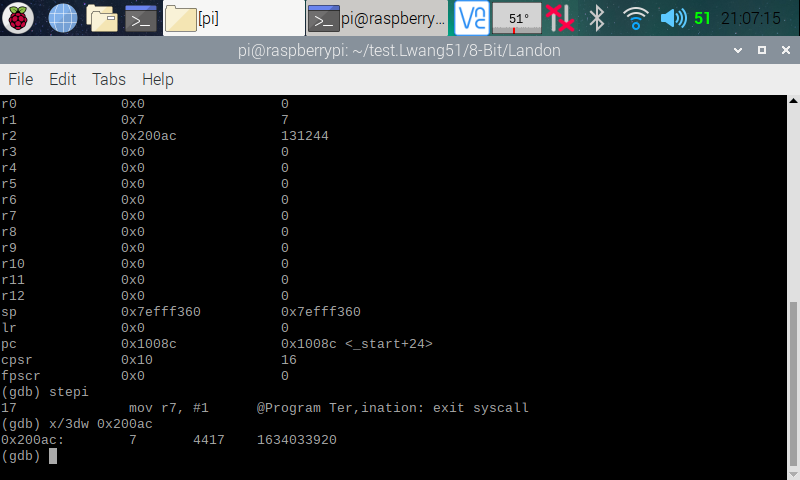
Here (in the screenshot above), I stepped over to the next line (line 15), so that line 14 (which loaded memory address of c into r2) will run. I pulled up the register information, and we can see that a memory address (0x200ac) has indeed been added into r2.



Here (in the screenshot above), I stepped over to the next line (line 17 [line 16 was blank]), so that line 15 (which stores r1 into memory c) will run.

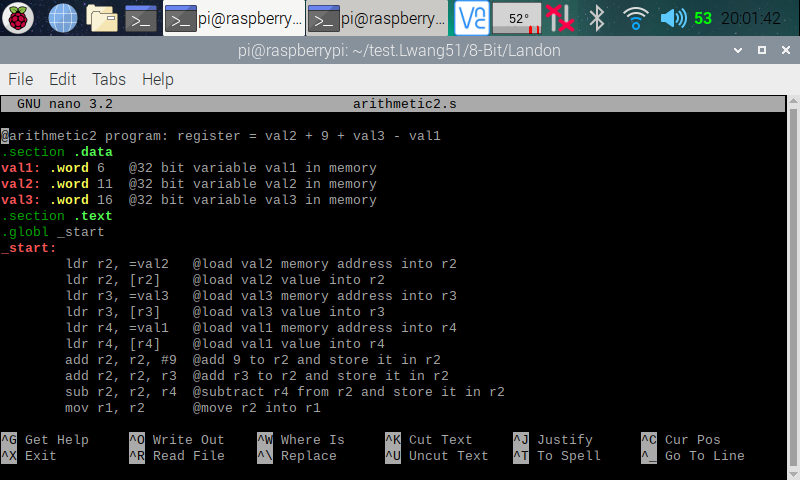


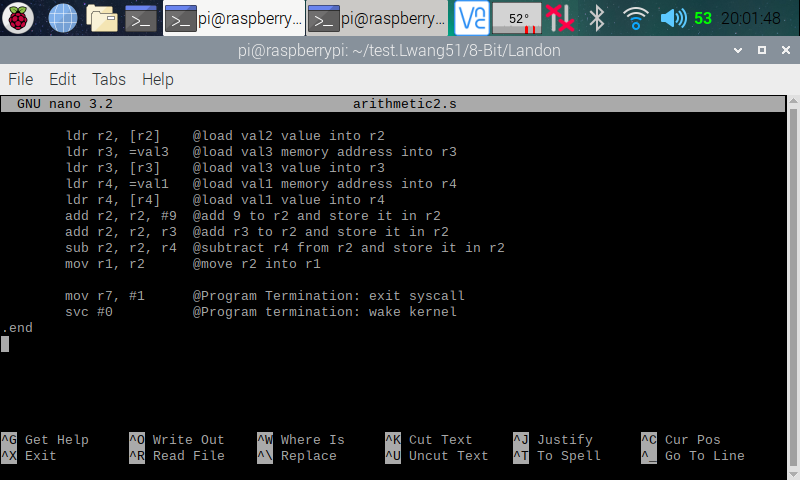
Here (in the screenshot above), I went through the rest of the debugger and exited after it went through the code.

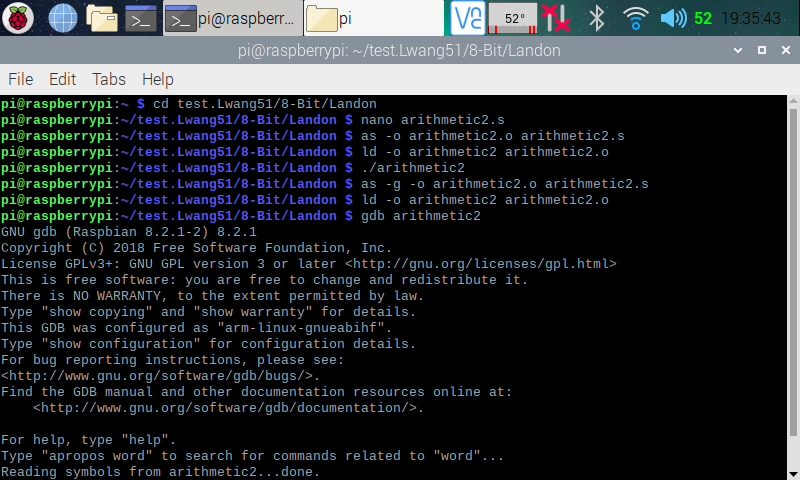


Here (in the screenshot above), I went back to the debugger and went through it again. At the end, I used the memory address of c (0x200ac) to pull it up to check if r1 was stored into memory c. In memory c, we can see that we do indeed have a 7 stored in it. We now know that our code is running correctly.

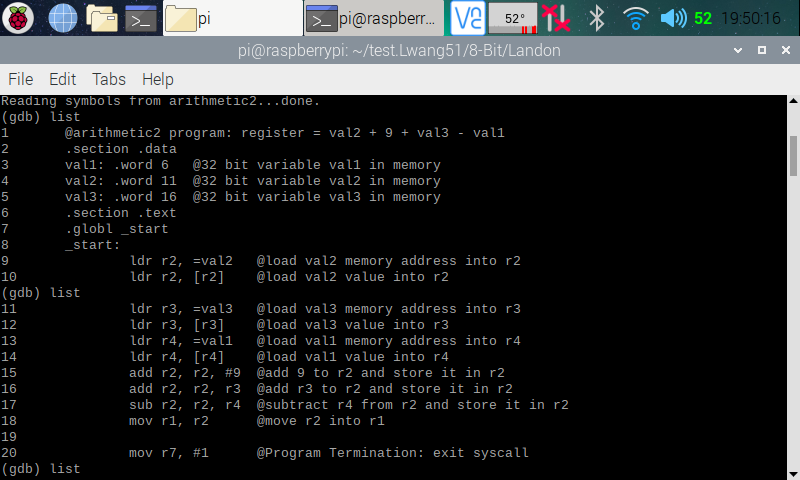
**Part Two: Arithmetic2 Program:**

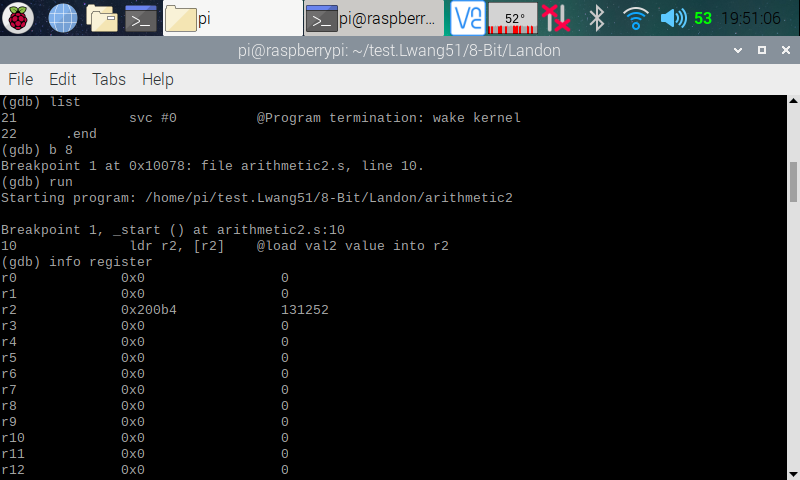




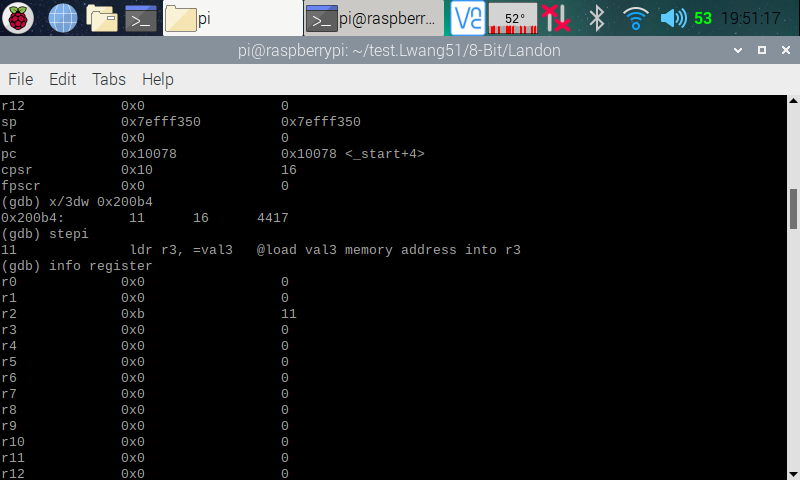


Here (in the three screenshots above), I created the Arithmetic program using the nano editor. I then assembled and linked the programs using the instructions **as -o arithmetic.o arithmetic.s** and **ld -o arithmetic arithmetic.o** respectively. After that, I ran the program using the instruction **./ arithmetic,** but nothing is shown, because data was only manipulated between the CPU registers and memory. To see if my program is running correctly, I went into the debugger using the instruction **gdb arithmetic**.

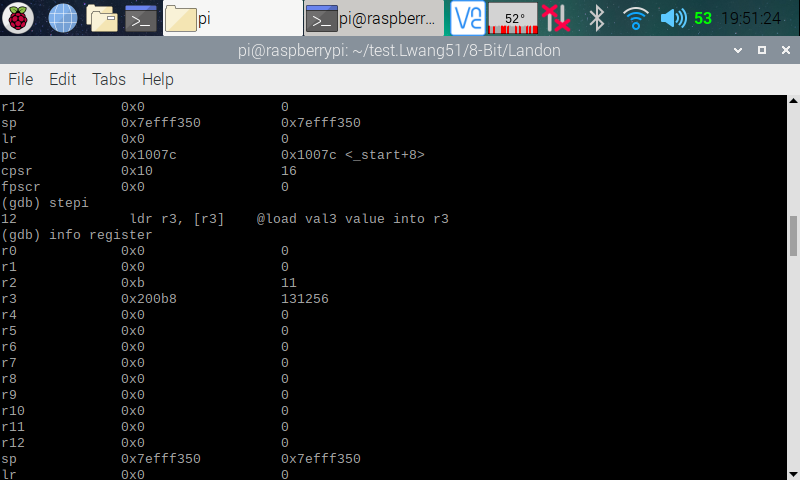




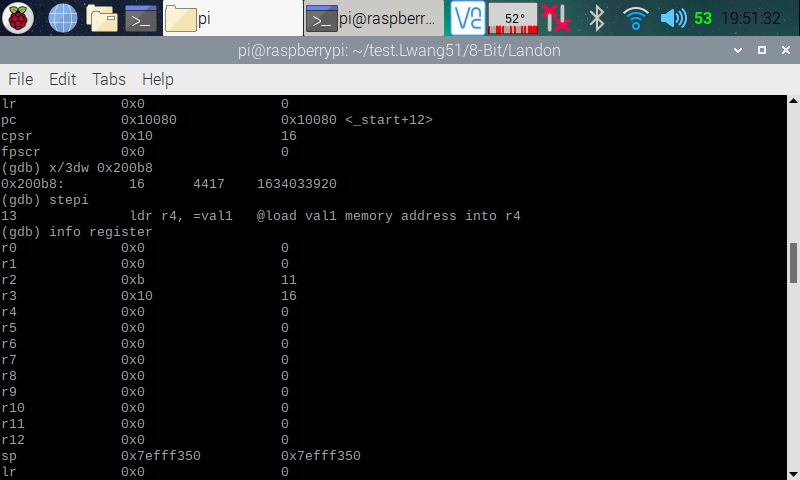
Here (in the two screenshots above), I used **list** to show my program instructions, then set a breakpoint at line 8 using **b 8** (which automatically moved the breakpoint to line 10). I then ran the debugger using **run**, and it stopped at line 10 where the breakpoint is set. I then pulled up the register using **info register**, and in the register, we can see the line 9 of the program executed correct, because a memory address (0x200b4) was loaded onto register 2.



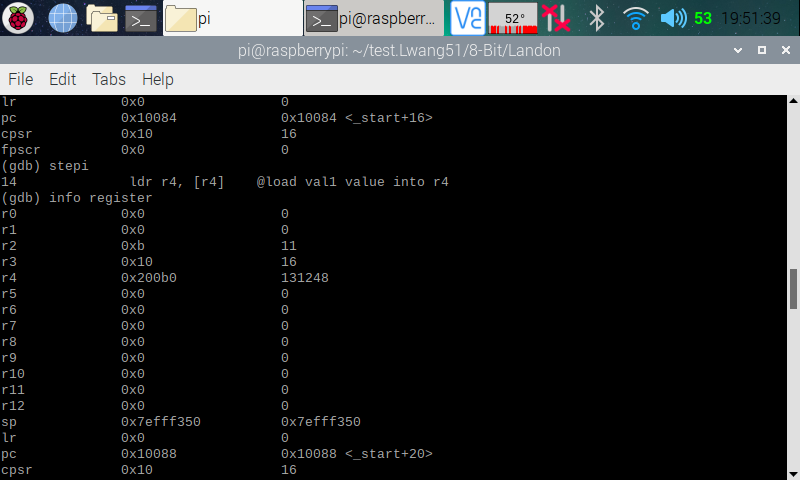
Here (in the screenshot above), I pulled up the memory (0x200b4) using the its address, and we can see that in the memory, we have the values 11, 16, and 4417 which are values of val2 and val3. We now know that we have the right memory address. I then stepped over to the next line (line 11), so that line 10 (which loaded value of val2 into r2) will run. I pulled up the register information, and we can see that 11 is now loaded onto r2.



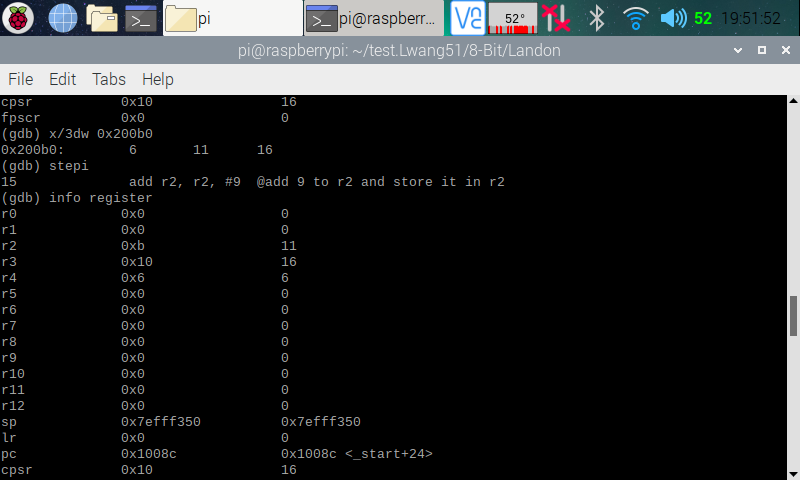
Here (in the screenshot above), I stepped over to the next line (line 12), so that line 11 (which loaded memory address of val3 into r3) will run. I pulled up the register information, and we can see that a memory address (0x200b8) has indeed been added into r3.



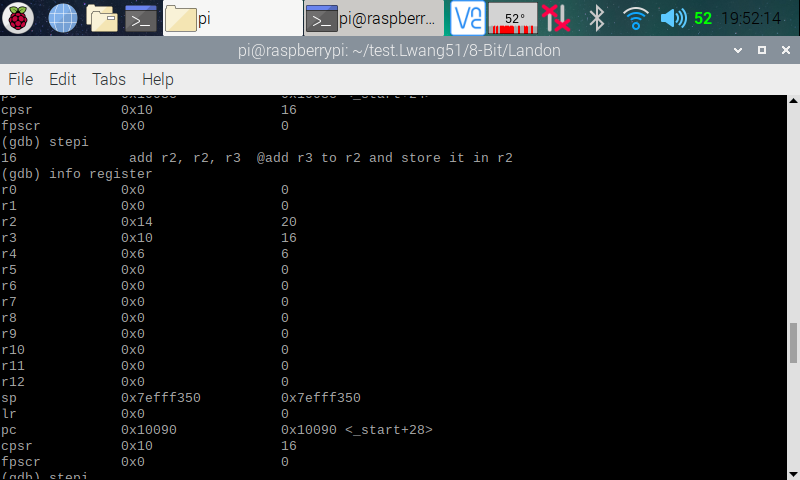
Here (in the screenshot above), I pulled up the memory (0x200b8) using the its address, and we can see that in the memory, we have the values 16, 4417, and 1634033920 which are values of val3. We now know that we have the right memory address. I then stepped over to the next line (line 13), so that line 12 (which loaded value of val3 into r3) will run. I pulled up the register information, and we can see that 16 is now loaded onto r3.



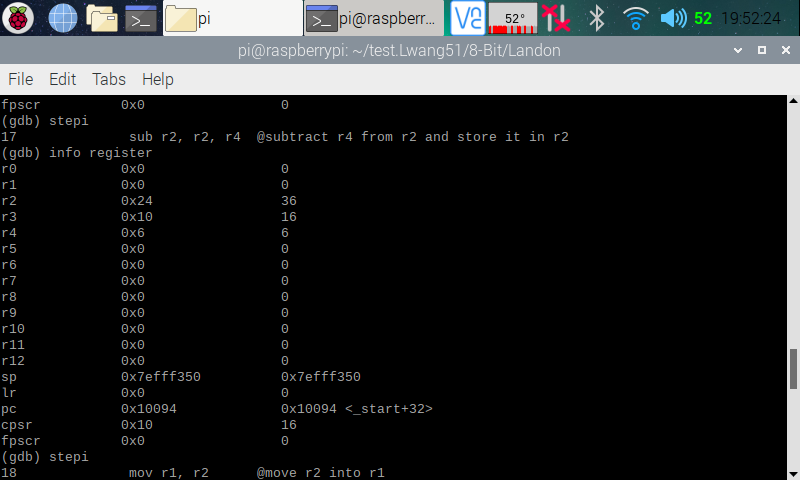
Here (in the screenshot above), I stepped over to the next line (line 14), so that line 13 (which loaded memory address of val1 into r4) will run. I pulled up the register information, and we can see that a memory address (0x200b0) has indeed been added into r4.



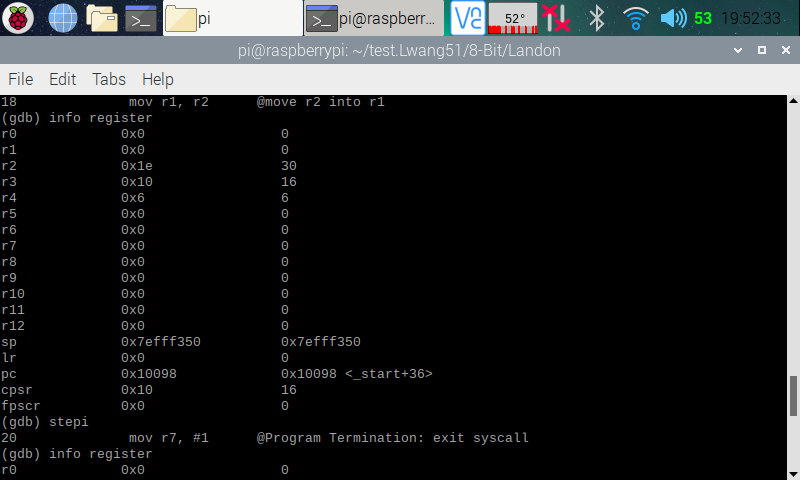
Here (in the screenshot above), I pulled up the memory (0x200b0) using the its address, and we can see that in the memory, we have the values 6, 11, 16 which are values of val1, val2, and val3. We now know that we have the right memory address. I then stepped over to the next line (line 15), so that line 14 (which loaded value of val1 into r4) will run. I pulled up the register information, and we can see that 6 is now loaded onto r4.



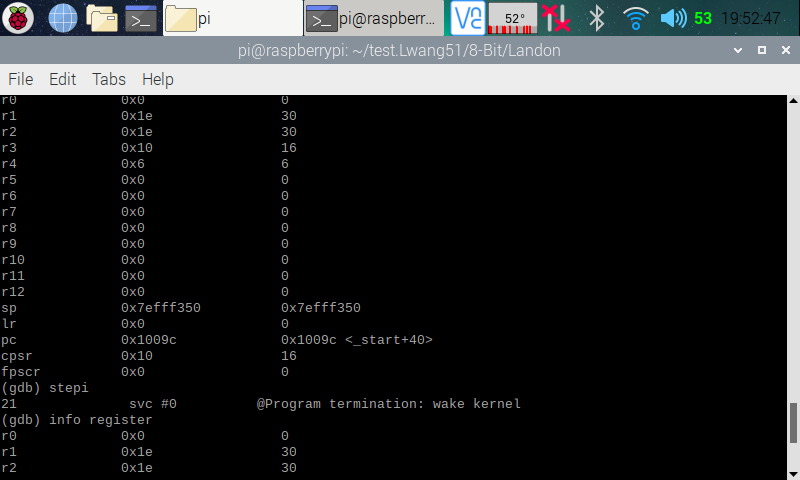
Here (in the screenshot above), I stepped over to the next line (line 16), so that line 15 (which added 9 to r2) will run. I pulled up the register information, and we can see that 20 (11 + 9) is now stored in r2.



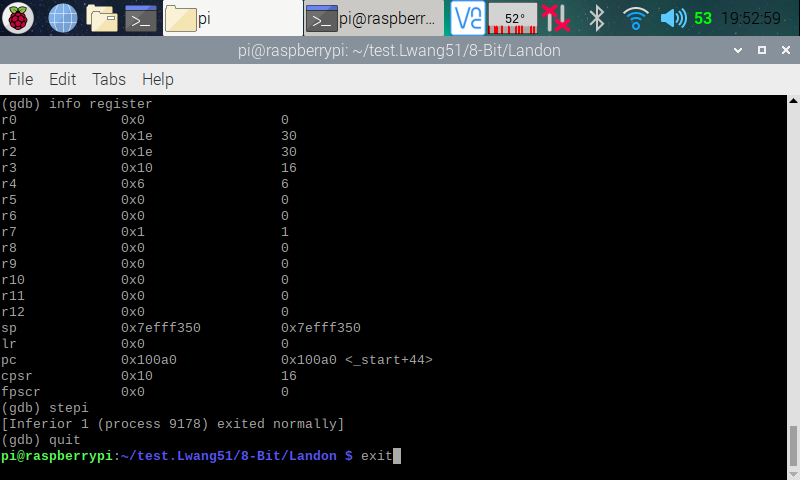
Here (in the screenshot above), I stepped over to the next line (line 17), so that line 16 (which added r3 to r2) will run. I pulled up the register information, and we can see that 36 (20 + 16) is now stored in r2. I then stepped over to the next line (line 18), so that line 17 (which subtracted r4 from r2) will run.



Here (in the screenshot above), I pulled up the register information, and we can see that 30 (36 - 6) is now stored in r2. I then stepped over to the next line (line 20 [line 19 was blank]), so that line 18 (moving r2 to r1) will run.



Here (in the screenshot above), I pulled up the register information, and we can see that 30 is now stored in r1. I then continued to step over to the next line until I reached the end of the program.



Here (in the screenshot above), I went through the rest of the debugger and exited it after it went through the code. I then exited the terminal.