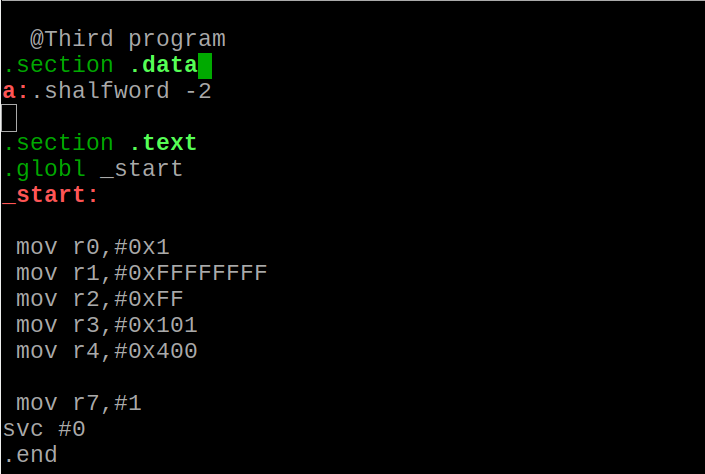
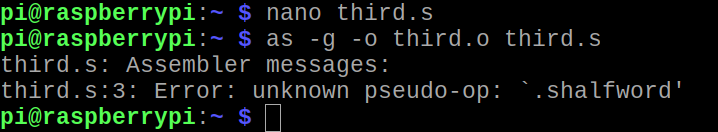
**ARM Assembly Programming:**

**Part 1:**

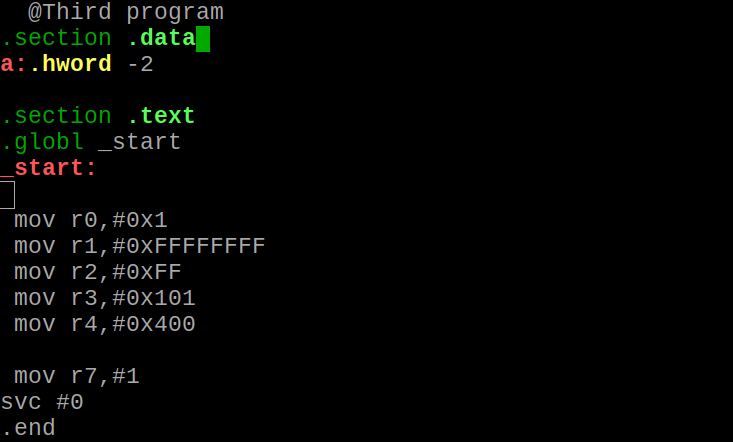
We started the assembly programming by opening a terminal, then opened a third.s file to write our program. We used the directive, .data, to indicate that we were going to declare some variables. We created a 2-byte signed memory size at location **a** initialized with -2. We then loaded the registers with some signed hexadecimal integers.



After we created the program, we tried to assemble it to get an object file and link, unfortunately, we couldn’t assemble it because there was an error in the program with the following message:

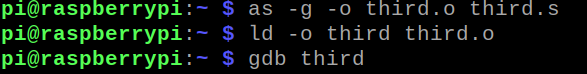


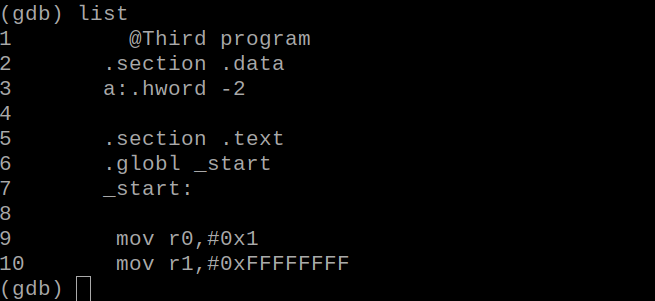
This error indicated that the ‘shalfword’ wasn’t a keyword and therefore can’t be used as a memory size. We fixed this error using the following program:



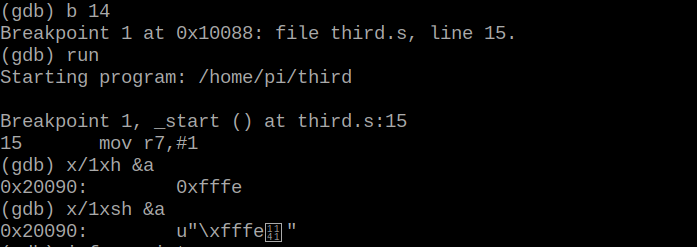
After the fix, we assembled and linked the program again, fortunately, it was able to assemble and link. We launched the gdb debugger for third.

We ran the **list** command to make sure our program was correct.



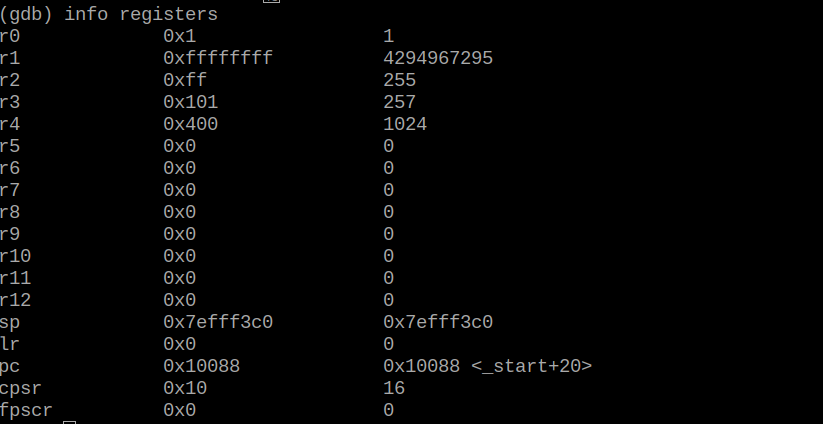


Since we didn’t load the variable **a** into any register, it didn’t matter where we set the breakpoint as long as it is outside the **.data section**. We set a breakpoint at line 14 using the syntex **b 14** to avoid stepping an instruction each time since we will be showing the content of the registers too. After that, we ran the program using **run**. After the program was successfully executed, we checked the memory location to make sure that the integer initialized (-2) was present in the 2-byte sized memory. We used both **x/1xh** and **x/1xsh** accompaning with the address of **a** (**&a)** to display the content of the memory in hexadecimal.



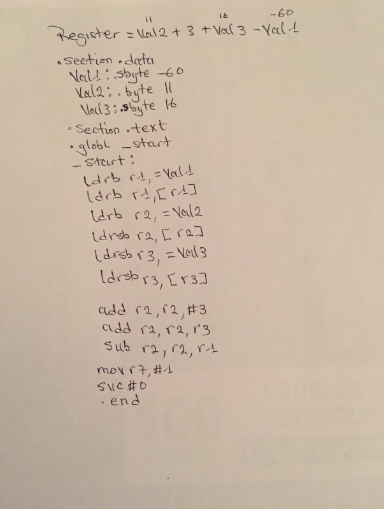
We observed that viewing the content of the memory using the halfword (x/1xh) is different from the signed halfword (x/1xsh), however, they both seem to provide us with the 2-byte memory with the integer -2 in hexadecimal.

We also examine the content of the registers to make sure that everything that we loaded stayed in the registers.

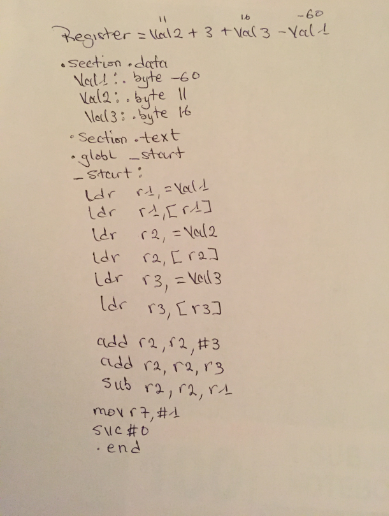


**Part 2:**

We created a second program called arithmetic3.s using Part 1 as a guide. We first planned what our program should do using the equation given as **Register = val2 + 3 + val3 - val1.** Given that Val1 is initialized with with -60, we automatically assume that we are dealing with a signed integer, so on a piece of paper, we illustrated what the program should look like.

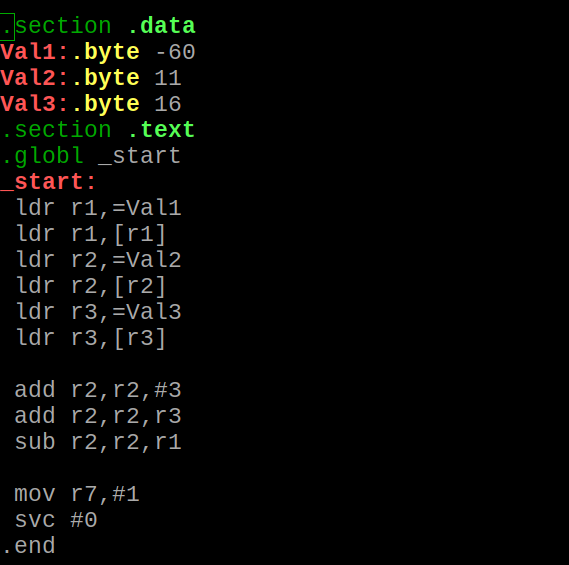


However, we got an error while trying to load the registers as a signed integer using **ldrsb** as suggested in the handout. As a result of trying to fix the program, we ended up with the following program:

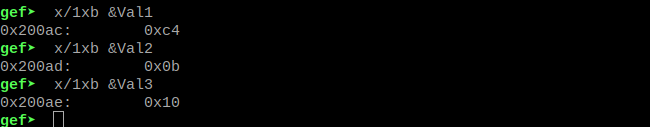


To determine the output of the program, we first opened an arithmetic3.s file to write our program. We used the directive, **.data**, to indicate that we were going to declare some variables. We created 3 byte variables: Val1, Val2, Val3. Val1 was initialized with -60, Val2 was initialized with 11, and Val3 was initialized with 16.

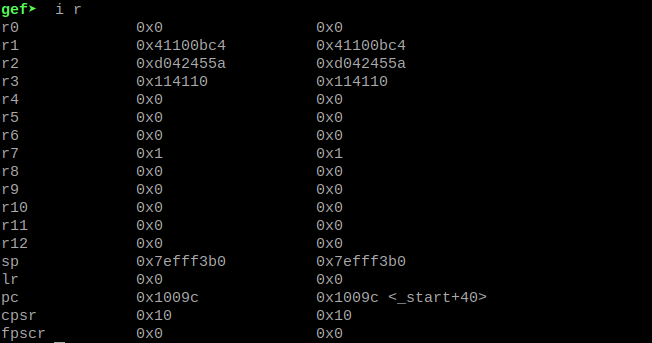
To use variable **val1**, we had to load (**ldr**) the memory address of **val1** into register r1. Then, we had to load the value of **val1** (**[r1]**) into register r1. After we loaded all the variables, we added **3** to the value of **r2**. We added the value of **r3** to the value of **r2**. Finally, we subtracted the value of **r1** from the value of **r2.**



After the code, we assembled, linked and debugged using **(gdb arithmetic3)**. We listed the program to make sure that everything was correct, then we set a breakpoint at 20 **(b 20 )** and ran the program. We examined the content of the memory using **x\1xb &Val1,Val2,Val3.**



We then checked the value in register 2 (hexadecimal) in **info registers** to make sure that the result from the program was correct.



We then checked for the flags

