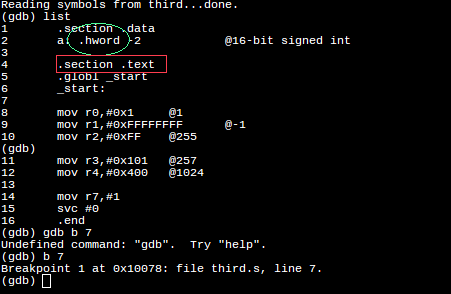
ARM Programming Report 3

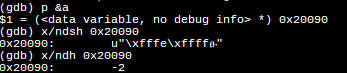
For assignment 3 we explored how the ARM handles storing signed and unsigned integers in the memory and its registers. Along with declaring signed variables. Creating a new file with ‘nano third.s’ I replicated the code for third.s, declaring the variable a and moving various unsigned values and a signed value to separate registers. However, upon assembling the following errors occur:



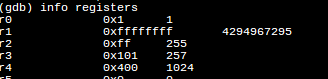
I suspected the reason for the errors is the lack of white space for .section .data/.text and the syntax for declaring signed halfword values. Placing white space between .section and its counterparts did alleviate the first and third errors. But for halfword I tested .shword and to see it functions similar to the x86 style of declaring signed variables. While it didn’t assemble either, I decided to assemble using .hword instead, which did assemble.



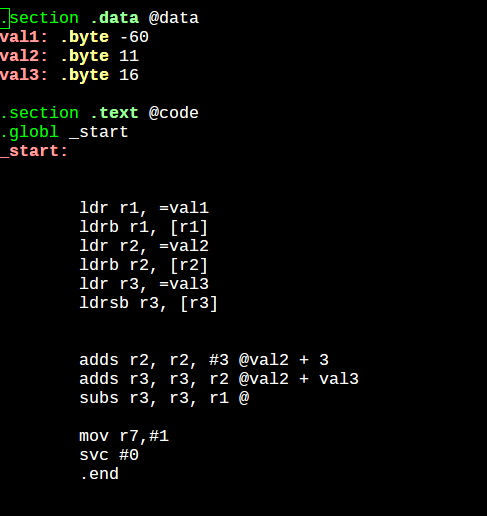
After running third in the debugger and checking the memory where variable “a” was stored, I was able to conclude that .hword has implied signed storage for signed variables.



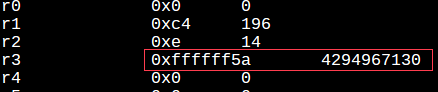
I also observed that registers store similarly to x86, thus interpreting all hex values as unsigned.



For the arithmetic3 assignment I was tasked to create three variables, 2 unsigned and 1 signed, and preform the following function: val2 + 3 + val3 – val1 and store in an arbitrary register. Formatted like this:

****

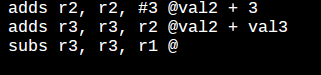
Checking the register during the arithmetic revealed that when the ARM 2’s complements a value to represent a negative under the SUB operand, it extends 1 to the most significant bit after the value. Thus, despite the true value being signed +90(5A), ARM still adds the leading extension.



Another thing that occurred was the lack of bits in the cpsr register. Without it it wouldn’t be possible to check the flags.



We had looked up how to have the bits representing the flags and learned that an ‘s’ must be appended to the instruction operator for each operation you want to observe flag changes with. With the flag order being NF ZF CF OF



Since 8h in binary is 1000, we were able to observe that the negative flag was set from the most significant bit from FFFFFF5A being 1.

