You may find this Vim tutorial useful
You may find this GDB debugging tutorial useful
You may find this ARM assembly tutorial useful

1. Read through the above tutorials to get a feel for our approach.
2. Open a terminal window in the directory containing sum.s with: *$vim sum.s*
    - The first thing you will notice is that comments begin with the // sign. Everything to the right of these on a line is ignored.
    - Next, you'll notice that there is a .data section and a .text section. The .data contains information stored in data memory.
        - Observe that the names of these data are mentioned at the bottom of the program. These are the “pointers” to the data. Loading the value stored in data memory requires you to load from the data at the bottom first, and then use the register you have loaded into as the address for another data load. We will talk about this more in the next few classes.
    - You'll also notice these "tags" along the left-hand side of the text. Tags are mnemonics, or short-hand, for data memory locations. We can use these tags to reference the data instead of specifying the memory location directly (this makes sense because you never actually know where data will wind up in memory until the code is assembled and linked – a topic we’ll get to later this week). Two important tags to notice:
        - Those in the middle of the text give names to parts of the code. In a way, you can specify an immediate value with these tags.
        - Those in data can be thought of as variable names, but they’re pointers which we will discuss more in the next few classes.
3. To run your file you must compile it in the terminal. Do so with:
    as -o sum.o sum.s
    gcc -o sum sum.o
    ./sum
4. The last line of the above code should result in an output to the terminal, let’s find out why. In the same terminal window, open the GDB debugger using:
    $gdb
    $file sum
    $start
    $disassemble
5. The last line should show you the compiled code with some extra information that we’ll get to later.
6. Set breakpoints at every *ldr* and *add* instruction using *break *address*. The address of each instruction is in hexadecimal on the left side of the sintruction. Then use *continue* and check the contents of your registers at every breakpoint using *info registers*. You’ll notice that their value changes at every breakpoint. Record some of the values with the print screen function.
7. Edit a copy of sum.s (call it sum2.s), change the three integers being summed, and rerun the program. Add breakpoints, run the code again and record the value of the registers before, during, after execution in order to demonstrate that it works.

8. Write an ARM assembly-language program (swap.s) that will swap the contents of two elements of an array, given the array and the indices of the elements to be swapped. Run the program and print screen snapshots of the data window before and after. Mark the variable locations whose contents have been exchanged. Some notes:
    - To declare an array, do something like the following:

      ar: .word 5, 17, -3, 22, 120, -1 // a six-element array
      arrend: .skip 24 //24 bytes are needed to store 6 words.

    - Note: the spaces after the commas are important!
    - For the indices, use two more integer variables declared in the data section with .word.

Hand In:

- your assembly files (sum2.s and swap.s)
    - be sure your name is written on the top of each file!
- annotated screenshots of relevant portions above (enough to show me that registers change)
- and your code is doing what you expect.

How I will test your code:

- I will run swap.s on a variety of different array inputs and confirm correct behavior