Array Sum

1. Save the copy of the array-sum program that uses a subroutine. Run it to verify that it works and look through data memory and the register information with GDB to get a feel for things. and then make the following changes in it.
2. The *endarr* label, as discussed in the lab, points to the array in data memory. Modify the summing function in array-sum so that the second argument is the array length in words (the number of integers in it). Use the *len* pointer to reference the size of the array, stored in data memory, which calculates the length of the array on the fly.
3. Add a *.asciz* declaration of an information message that will label the output value and print it with *bl printf*. This is the same function as is defined in C.
    a. Make sure you create a pointer to the message at the bottom of the *.text* section.
    b. Make sure you store and load the link register (*lr*) before and after calling printf.
    c. Do not rely on registers *r0-r3* after printf as they are unprotect registers.
4. Some changes for *arrsum*:
    a. Improve the loop design so that the loop test is at the end of the loop, with an initial jump to it before entering the loop (it should still be a while-loop, allowing for a possibly empty array). Why is this better?
    b. Run the program and print a snapshot of data memory in GDB.

Insertion Sort in Assembly

1. Make a copy of array-sum, and use the copy as a starting point implementation insertion sort in ARM assembly language. The C++ code for insertion sort is a useful starting point. Translate it into assembly language and test it.
2. Some Notes:
    a. The isort routine must be a subroutine called from the main program, and the array address and the number of integers in the array must be passed in registers.
    b. You may use a for-loop, but ARM relies heavily on pointers so it may be useful to get comfortable with them. You should not need new variables in data memory.
    c. Before starting, it is important to have an understanding of the calling convention and protected registers of ARM assembly. The stack must be used when you run out of scratch registers to either hold data or hold a protected register. The latter is more efficient, something we will discuss later.
    d. When your routine works, which you can see by looking at the array in GDB data memory after the call to sort it, add another subroutine that runs through the array and prints its elements. This will be a simple modification of the array-summing code that also uses the *printf* instruction which we have discussed. Have your main program call this output routine before and after sorting, and label each line of output appropriately, using *.asciz* strings.
3. Grab a snapshot of your output window before and after the run, and in each case, highlight the portion of the data area that holds the array. Grab a snapshot of the console too.

Extra Credit

- Implement the recursive implementation of insertion sort. I will have shared the code.

Hand In

- A PDF report containing a brief written description of each of your programs, alongside before-and-after screenshots.
- Your modified array-sum
- Your isort.s
- Optionally, your recursive insertion sort.