ECSE 324 Lab 2 Report

Lab 2 consisted not only in writing assembly code but C code as well that could call upon assembly to perform the given task. The code in assembly consisted of a stack class that pushes and pops elements from an array, a class that inputs a number and computes its Fibonacci, a MAX_2 class taken from the lab description which is called upon by C code which inputs two numbers and outputs the bigger one and a subroutine that outputs the maximum value of a given array. The classes in C are simply two different approaches in finding the maximum value of a set, one calling assembly code and the other doing it all in C.

1) Subroutines

- 1.1) The stack

  Our goal here was to rewrite push and pop instructions in Assembly. This was fairly simple to implement, as we used the stack pointer SP to store and load values from the stack. The flow of our program is the following:
  - A few instructions to load our NUMBERS into necessary registers.
  - Using the STR instruction on SP, we push the values that we want.
  - Using the LDMIA instruction on SP, we pop the values that were just pushed, and load them in R0 through R2.

- 1.2) The subroutine calling convention

  Once we had properly understood how to pass data from the main routine to the subroutine and back, changing our code to include a subroutine was fairly easy. The process remained similar overall but required some code to be moved and some adjustments to be made.

  First, the usual loading of data is done at the beginning. R0 contains the first number of the list, and R2 the number of elements in the list. The MAX subroutine is then called. It is composed of a for loop which decrements R2 at every iteration. At each iteration, a value of the list is compared to the current maximum value and is set as the new maximum value if it exceeds it. This maximum value is stored in R0. When the for loop terminates, the subroutine finishes, and returns to the main section of the code.

- 1.3) Fibonacci calculation using recursive subroutine calls

  The Fibonacci sequence is simple in theory, we set up a base case, if the input is either 0 or 1 then we output 0 and 1 respectively, otherwise we use recursion inside the method, each time changing the input to (n-1) and (n-2) until the base case is satisfied. This means we will need two recursive calls within the FIBONACCI subroutine for (n-1) and (n-2). In order for the program to remember what the previous iteration of the Fibonacci subroutine yielded, we used the stack implementation mentioned above to push and pop LR, otherwise the program simply kept on doing calculations without remembering the previous inputs, meaning that it would only work with the base case inputs. Another issue we faced but didn't address as it was a minor one is the following one:

The Fibonacci series is most usually expressed the following way: 1, 1, 2, 3, 5, ...

The nth number of the Fibonacci series, or Fib(n), is the number at position n in the series. Fib(1) = Fib(2) = 1, Fib(3) = 2, etc.

However, our program will return Fib(1) = 1, Fib(2) = 2, Fib(3) = 3. The reason for this is that the program interprets Fib(1) as the number in position 1 in the list of Fibonacci numbers, starting from position 0 (and not from position 1). Fib(1) is therefore interpreted as the second number of the list, Fib(4) is interpreted as the fifth number of the list, etc.

When we expressed this issue to the TA during the demo, he assured us this was not a problem.

2) C Programming

- 2.1) Pure C

  This class only operates using C code and doesn't call assembly at all. We input an array of any given size and we assume the first value (at index 0) to be the greatest(max_val) one in the set. Using a for loop we then iterate through the whole array and compare each value we come across to the current maximum value, if any number is bigger than max_val, that number becomes the maximum value and we keep going until we have gone through all the values in the array, at which point we return max_val.

- 2.2) Calling an assembly subroutine from C

  The structure of the C code using the subroutine is very similar to Pure C, however instead of comparing max_val and the numbers in the array within the C program we are going to be calling upon the assembly class MAX_2 given in the lab description. The code operates very similarly using a for loop, however when the time comes to compare two values we call MAX_2 with two inputs (the current maximum value and the next value in the array) inside the for loop and when all the numbers have been checked return the final result.

We found C Programming to be easier to handle than pure assembly, since we are familiar with higher level languages, calling a simple subroutine inside a C class was much more intuitive than writing the entire program in assembly as we did in section 1.2. One issue with our code is that it will throw an error if the input array is of size 0. An easy way to fix this would be to return an error message if such an event occurs, by surrounding the code with a try/catch statement.