Project 1 Write Up - Finding Fibonacci Numbers Using MARIE

Using the MARIE architecture and associated assembly language, I was able to successfully build a Fibonacci Sequence calculator that runs using only the concepts that we have learned so far in Computer Architecture. My algorithm runs with the use of six variables: “ONE”, “userInput”, “currTotal”, “firstIndexValue”, “secondIndexValue”, and “COUNTER”. The variable “ONE” was used as a basic iterator throughout the program that simply incremented the “COUNTER” value by 1 for convenience purposes. Another element, “userInput” (or “n” in the provided instructions handout), was responsible for storing the number of Fibonacci numbers the user wanted to have calculated as specified at the beginning of the program. The “currTotal” element was used to record and output the next number in the Fibonacci Sequence. The only reason why “currTotal” was able to be calculated, however, was by using “firstIndexValue” and “secondIndexValue”, whose values were sent to “currTotal” in an alternating fashion so that previously held elements could still be accessed for the actual Fibonacci addition process without the implementation of an array structure. Finally, the “COUNTER” was a device used to alternate between “firstIndexValue” and “secondIndexValue” by having its stored value switch between 0 and 1. Basically, if the “COUNTER” was equal to 0, the “firstIndexValue” was added to “currTotal”. Likewise, if the “COUNTER” was equal to 1, the “secondIndexValue” would instead be added to “currTotal”.

The actual implementation of my algorithm begins with a simple input validation to ensure that the user enters a positive value over 0 which is stored in “userInput”. The algorithm then proceeds to function by alternating between “firstIndexValue” and “secondIndexValue” using “COUNTER” so that previous members of the Fibonacci Sequence are remembered and can be added to “currTotal” to determine what the next number of the Fibonacci Sequence will be later on in the program. This series of events is continuously repeated the number of times that the user had specified at the beginning of the program so that it can find the number that they were looking for in the Fibonacci Sequence. By the time you reach the final output of the program, every number in the sequence up to, and including, the value the user had specified at the beginning of the program would have been printed out. To get a better understanding of how my algorithm works, a real-world example that I came up with would be when you are hiking through the woods and you come across a fork in the road. At that point you decide to go take a left (“firstIndexValue”) and make note of your decision (“COUNTER”) but at the next fork you decide to take a right (“secondIndexValue”) and you make note of that decision as well (“COUNTER”). At each junction, you would also make sure to keep a log of your decisions so that you could base your next one off the trend produced from the previous two and have a reference of them all at the end of your journey (“currTotal”). You would then continue to repeat this process, being sure to make alternating choices at each fork in the road, until you reach the end where you are able to look back at your log (“currTotal”) and have a complete list of each choice that you made along the way.

In terms of the problems that I encountered while trying to complete this assignment, my biggest challenge was probably just trying to figure out exactly how to use each of the MARIE assembly commands. I’ve never written anything in assembly before and I’ve noticed that I learn better with more of a hands-on approach rather than just strictly reading from the textbook, so I started this assignment out by writing a quick addition program in assembly. After I felt comfortable with that, I began experimenting with other commands, such as Skipcond and Jump, and eventually I was able to find success. A more specific problem that I had with assembly code was with the Jump command because I thought that I could just count out the lines in decimal notation that I wanted to access and use that as my argument. Eventually, I realized that the assembler was expecting the argument to be in Hex and this not only fixed a major problem in my program, but it also cured a whole lot of frustration that had built up inside me. I also had trouble with trying to figure out the logic of having two one’s print out in a row without hard coding the first value of the Fibonacci Sequence. In the end, I was able to resolve this by setting “secondIndexValue” as 0 to start unlike “firstIndexValue” which had been set to 1. This additional alternating scheme helped increment “currTotal” to a 1 to begin with and then maintain it for one more iteration before “firstIndexValue” and “secondIndexValue” were set to the previously held value by “currTotal” for their respective iterations after the first and second initial rounds in the program. Writing code in assembly definitely gave me much more of an appreciation for the high-level programming languages that I’m used to like C++ and Java.

As for my acknowledgements, I did not collaborate with anyone on this project and everything in my program is my own code. The only resources that I used to help design my algorithm was the information found in the textbook and my notes, which acted as a reference for me as I tried to learn and implement the new assembly language commands in MARIE. I am also happy to point out that there was no hard coding in my application, and it should be able to complete its operation for any value, within the allocated memory space of MARIE, that the user is looking for in the Fibonacci Sequence.