Microcontrollers

ECE 346 Project

Final Project

By: Patrick Hurley

04/24/2024

Honor Pledge:

"I pledge to support the Honor System of Old Dominion University. I will refrain from any form of academic dishonesty or deception, such as cheating or plagiarism. I am aware that as a member of the academic community, it is my responsibility to turn in all suspected violators of the Honor System. I will report to an Honor Council hearing if summoned."

Signature: Patrick Hurley

Table of Contents

Introduction

For the final project, we were tasked with writing a separate program that would fulfill the instructions for each part. The first part wanted us to create a Fibonacci sequence given the length of the sequence from key switches. We would give an input number, and that number would represent how long the Fibonacci sequence would be. The second part wanted us to create a program that would bubble sort a list of numbers. This part also wanted us to create subroutines that would take care of the sorting.

Background

The first part of the final project wanted us to create a program that would take the input of four switches in binary. The input would range from zero (0000) to fifteen (1111). We would then use this 4-bit binary input to determine how long the Fibonacci sequence needs to be. This sequence would then be stored in the memory starting at memory address 0x1000. A Fibonacci sequence is when you start with a pair of numbers, we started with 0 as the first number and 1 as the second number. To get the next number in the sequence you would add together the two previous numbers, and then that would be your next number. For example, when you add the first number 0 and the second number 1, you will get the number 1 as your next number in the sequence. Next, you would then take the second number, which is 1, and the third number, which is also 1, to then get the fourth number. The fourth number would be 2. We would keep doing this until we get to the Nth sequence. The second part of the final project involved me creating a program that would perform a bubble sort on a list of numbers in a subroutine. A bubble sort is when you compare two numbers in a list and swap them if the first number is larger. This would result in the smaller numbers going to the beginning of the list, and the larger numbers going to the end of the list. Ultimately, you would have a sorted list of numbers from smallest to largest.

Preliminary

The preliminary work I had to do consisted of figuring out what a Fibonacci sequence is, and how to implement it into code. I would also have to learn how to implement a bubble sort into a subroutine. I had to learn how to pass the base address, through the stack, to the subroutine so it would know where the data is located to be worked on. I also had to learn how to insert data from external files directly into the memory.

Experiment

To implement a Fibonacci sequence I have to create many different branches that would take on many different roles. The first branch I had was the LOOP branch, which was in charge of setting the starting variables. While I didn't have a stack pointer, R3 acted as a stack pointer. I would hold the memory address that I would insert data into, and would move to the next memory address with data is inserted. R4 was my first number, which was set to 0, and R5 was the second number in the Fibonacci sequence, which was set to 1. The CHECK_KEY branch was responsible for checking if the key was pressed. When pressed it would jump over to the FIBONACCI branch, and when not pressed, it would keep looping until pressed. The

FIBONACCI branch is responsible for loading the value from the switches(N) and for starting the counter. The counter starts at the value 1 because we already have the first two values of the sequence. This branch will also check to see if the value of N is 0 or not. If the value is 0 we will go to the FIB_0 branch, here we will insert the first number of the sequence into the memory and then jump back to LOOP. If the value of N is greater than zero we will then branch to FIB_1 and insert the first two values into the memory. Next, we will then go to the FIB_LOOP. This loop is responsible for checking to see if the counter is equal to N, adding the two numbers together, shifting the new values onto the registers, and getting rid of the old values. We will keep looping until the counter matches N, then we will go back to the LOOP branch and wait for another input from the switches.

Here is the Code for Q1:

```
.include    "address_map_arm.s"

.text       /* executable code follows */
.global     _start
_start:
   LDR     R1, =SW_BASE
   LDR     R2, =KEY_BASE


LOOP:
   LDR     R3, =0x1000
   MOV     R4, #0
   MOV     R5, #1

CHECK_KEY:
   LDR     R9, [R2]
   CMP     R9, #0
   BGT     FIBONACCI
   B       CHECK_KEY

FIBONACCI:
   LDR     R6, [R1]
   MOV     R8, #1
   CMP     R6, R4
   BNE     FIB_1

FIB_0:
   STR     R4, [R3], #4
   B       LOOP

FIB_1:
   STR     R4, [R3], #4
```

```
   STR     R5, [R3], #4

FIB_LOOP:
   CMP     R8, R6
   BEQ     LOOP
   ADD     R7, R5, R4
   MOV     R4, R5
   MOV     R5, R7
   ADD     R8, R8, #1
   STR     R5, [R3], #4
   B       FIB_LOOP
```

For the second part, we have to create a bubble sorting algorithm in a subroutine. The main project file is responsible for creating the stack pointer and for pushing the starting memory address onto the stack. The main file also has a BL statement to branch over to the bubble sort subroutine and a STOP branch for when the program is finished sorting. The bubble sort subroutine starts with the B_SORT branch. This branch is responsible for setting the correct starting values for each of the registers. R1 holds the base address for where the unsorted list is stored. R3 holds the first values in the list, which are the values that show how many data points are in the set. We set R4 to the same value as R3 so that we can change R4. We also save the address of the first data point in the list to R6. The R2 variable acts as a memory address pointer. the first data point. R4 and R5 are the counters for the inner and outer for loops for the sorting algorithm. The LOOP1 branch is responsible for resetting the R5 counter, and for setting R2 to the address of the first data point in the list. This loop also will subtract one from R4 and check to see if the value after is 0, if the value is zero then the sorting algorithm is finished and will branch to RETURN. LOOP2 is responsible for incrementing the inner for loop counter, R5, and loading the two values we will compare. If the first value is larger we will go to the SWAP branch. This branch will swap the two values in the memory and will loop back to the LOOP2 branch. When the values of R5 are greater than the number of values in the list, then we will jump back to LOOP1.

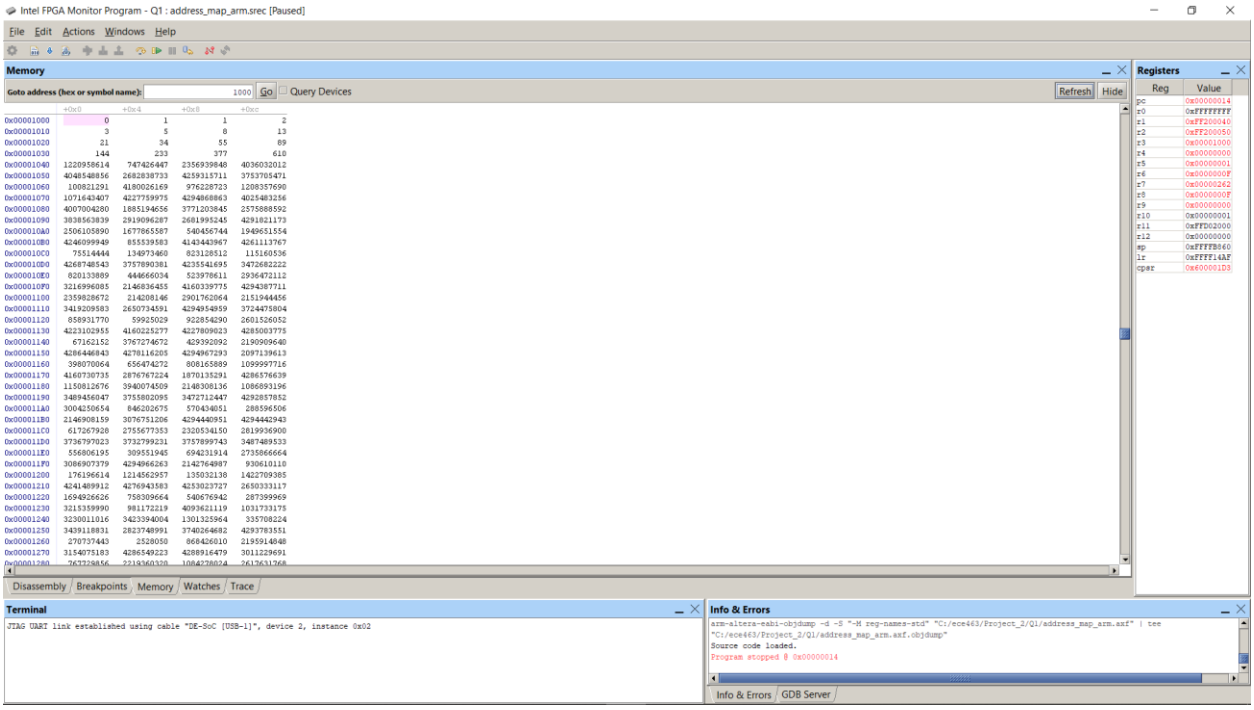
Here is the Code for Q2:
MAIN:

```
.include    "address_map_arm.s"
.include    "bubblesort.s"

.text       /* executable code follows */
.global     _start
_start:
        MOV             SP, #DDR_END - 3
        LDR             R12, =0x1000
        STR             R12, [SP], #-4
```

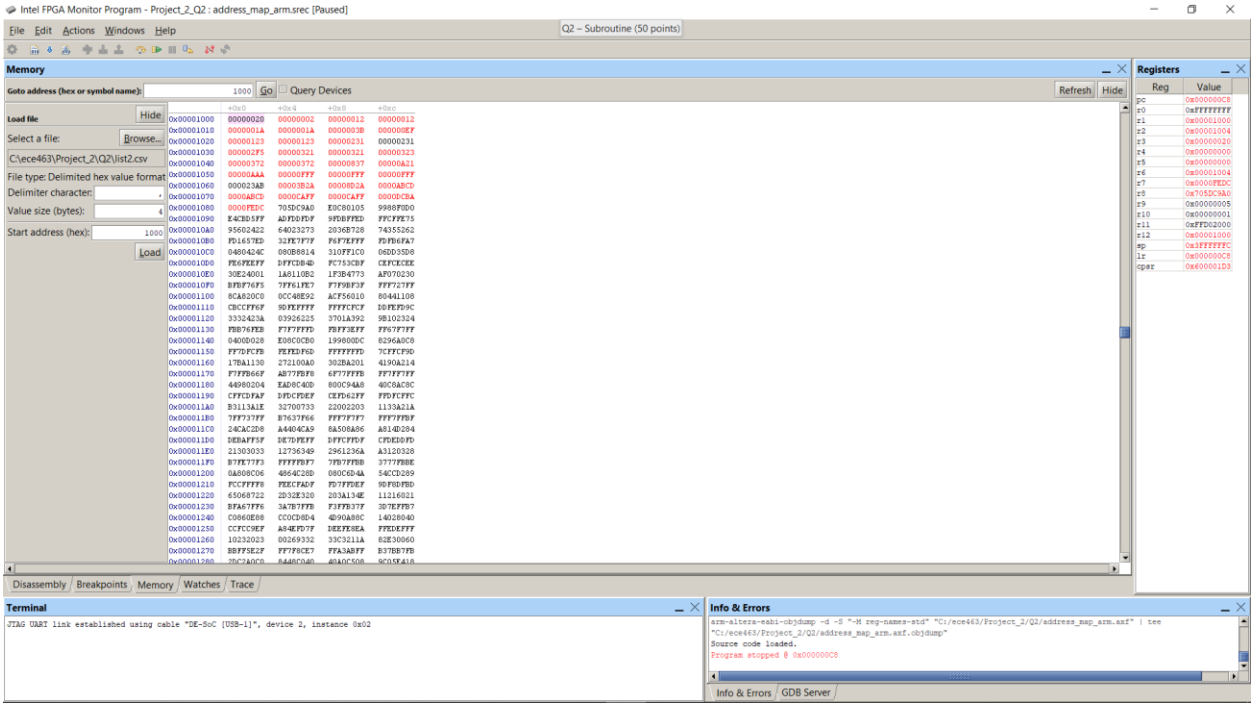
```
        BL              B_SORT

STOP:
        B               STOP

SORT SUBROUTINE:

.text       /* executable code follows */

B_SORT:
        ADD             SP, SP, #4      //pop the base address
        LDR             R1, [SP]
        MOV             R2, R1          //copy to additional
        LDR     R3, [R2]
   MOV     R4, R3      //Number of elements
   ADD     R6, R2, #4  //address of starting digit

LOOP1:
   MOV     R5, #0
   MOV     R2, R6
   SUB     R4, R4, #1
   CMP     R4, #0
   BEQ          RETURN
LOOP2:
   ADD     R5, R5, #1
   CMP     R5, R3
   BGT     LOOP1

   LDR     R7, [R2]   //First element
   LDR     R8, [R2, #4]!    //Second element

   CMP     R7, R8
   BGT     SWAP
   B       LOOP2
SWAP:
   STR     R7, [R2]
   STR     R8, [R2,#-4]
   B       LOOP2

RETURN:
BX LR
```

NOTE: The source code is available in the project .zip folder.

Results



The Values in this screenshot match exactly what was shown in the Question 1 project instructions when the value of the switches was set to 1111, or 15. In the screenshot, I have also switched the values of the memory to decimal to more easily compare to what was given in the project slides.



This screen show shows the memory where the list of values was stored. The order of the values has been changed because of the bubble sort, and the order of the values matches the memory address that was shown in the project instruction.

NOTE: The Full screenshot of the registers and memory separate and together will be in the project Zip file.

Conclusion

In conclusion, the results were exactly how they were shown in the project instruction slides. The first part showed the correct length of the Fibonacci sequence and also showed that my sequence algorithms correctly created the next values in the sequence. In the second part of the project I was able to insert the data values into the memory, and my bubble sort routine was able to sort the values from smallest to largest. The program even went back to the main file and stopped at the STOP branch. If I could do something different I would try to learn how to implement a different sorting algorithm instead of using one I already knew how to do from an earlier lab.