

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**
- **Background**
- **Preliminary**
- **Experiment**
- **Results**
- **Conclusion**

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

Intel FPGA Monitor Program - Q1 : address_map_arm.srec [Paused]

File Edit Actions Windows Help

Memory

Go to address (hex or symbol name): 1000 [Go] Query Devices

Address	Hex	Dec	Hex	Dec	Hex	Dec
0x00001000	0	0	1	1	2	2
0x00001010	5	5	6	6	13	13
0x00001020	21	34	55	89		
0x00001030	144	233	377	610		
0x00001040	122958614	747425447	235939846	4036032012		
0x00001050	4048148856	2682828723	828931711	3757708471		
0x00001060	100821291	4180026169	97628723	1208357690		
0x00001070	1071643407	4227799975	429466863	4021482316		
0x00001080	400704290	1855184656	3771203845	2575988592		
0x00001090	3038563839	2919094287	2661993245	4291821173		
0x000010A0	2506105990	1677865587	540456744	1949651154		
0x000010B0	4246099949	815539583	4142442967	4261117367		
0x000010C0	75514444	134973460	823128512	115169536		
0x000010D0	4268748543	375789281	4235541695	3472682222		
0x000010E0	820132889	446646034	523978611	2926472112		
0x000010F0	321499685	2146856455	416033975	4242487711		
0x00001100	235982872	214208146	2901762064	2151944456		
0x00001110	341920983	2630794591	429494859	3754479504		
0x00001120	85891170	59955059	922858290	2601526052		
0x00001130	423102955	4160225277	427898023	4285003775		
0x00001140	67162152	3767274672	429392092	2190909640		
0x00001150	423644843	4278114285	4284867293	2097139613		
0x00001160	398070064	656474272	808165889	1099997716		
0x00001170	4169730735	2876787224	1870138291	4286576639		
0x00001180	1150812876	3940074509	2148200136	1086959196		
0x00001190	3489456047	3755802095	3472712447	4292878152		
0x000011A0	3004250654	846232675	570438051	288596506		
0x000011B0	2146908159	3870751206	4294469561	4294462429		
0x000011C0	615797820	2755677353	320554150	2819936900		
0x000011D0	3736797023	3732799231	3757899743	3487489533		
0x000011E0	556906195	399551946	694231914	2735866664		
0x000011F0	2086907779	4284864263	2142764897	930610110		
0x00001200	176196614	1214562957	135032138	1422709385		
0x00001210	424189912	4276943583	4253027327	285033117		
0x00001220	159426626	755039664	546706240	287399949		
0x00001230	3215359990	981172219	409362119	103733175		
0x00001240	3230011016	3423394004	1301323964	335708224		
0x00001250	3439118820	2823748991	3742624682	4297878351		
0x00001260	270770443	2520050	868426010	2155914848		
0x00001270	3154075183	4286549223	4288916479	3011223691		
0x00001280	787228456	2718361328	1084278024	2613631268		

Registers

Reg	Value
PC	0x00000014
R0	0xFFFFFFFF
R1	0x00000000
R2	0xFF200040
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000001
R11	0xFFD02000
R12	0x00000000
R13	0xFFFFFFFF
R14	0xFFFFFFFF
R15	0x00000013

Disassembly / Breakpoints / Memory / Watches / Trace

Terminal

```

TMS UART link established using cable "DE-9C [USB-1]", device 2, instance 0x02
arm-11c86-eabi-objdump -d -O -M reg-names-std "C:/ece463/Project_2/Q1/address_map_arm.srec" | tee
"C:/ece463/Project_2/Q1/address_map_arm.ssf.objdump"
Source code loaded.
Program stopped @ 0x00000014

```

Info & Errors

```

arm-11c86-eabi-objdump -d -O -M reg-names-std "C:/ece463/Project_2/Q1/address_map_arm.srec" | tee
"C:/ece463/Project_2/Q1/address_map_arm.ssf.objdump"
Source code loaded.
Program stopped @ 0x00000014

```

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.

Intel FPGA Monitor Program - Project_2_Q2 : address_map_arm.srec [Paused]

File Edit Actions Windows Help

Q2 - Subroutine (50 points)

Memory

Go to address (hex or symbol name): 1000 [Go] Query Devices

Load file

Select a file: C:\ece463\Project_2\Q2\list2.sv

File type: Delimited hex value format

Delimiter character: .

Value size (bytes): 4

Start address (hex): 1000

Load

Address	Hex	Dec	Hex	Dec	Hex	Dec
0x00001000	00000020	00000002	00000012	00000012	00000012	00000012
0x00001010	0000001A	0000001A	0000003B	0000003B	0000003B	0000003B
0x00001020	00000123	00000123	00000123	00000123	00000123	00000123
0x00001030	000002F5	000002F5	00000321	00000321	00000323	00000323
0x00001040	00000372	00000372	00000837	00000837	00000A11	00000A11
0x00001050	00000A1A	00000A1A	000009FF	000009FF	000009FF	000009FF
0x00001060	000023AB	000023AB	00000E2A	00000E2A	0000ABCD	0000ABCD
0x00001070	0000ABC0	0000ABC0	0000C4FF	0000C4FF	0000CBA	0000CBA
0x00001080	0000FEDC	785DC9A0	83C60105	9988F8D0		
0x00001090	84E0197F	AD05010F	9F8F8D0	FFC7F5		
0x000010A0	956024C2	64023273	20368728	74352662		
0x000010B0	F016578D	32FE7F7F	F87F7F7F	F8FBFA7		
0x000010C0	048042AC	06080614	310FFC0	06050080		
0x000010D0	F6F8F8FF	DFFC084D	F753CBF	CF8C88E		
0x000010E0	30E24001	1A811082	1F84773	AF070230		
0x000010F0	8BFF8F8F	7FF81187	F7F8F8F	FFF27FFF		
0x00001100	8CA820C0	0CC48E92	ACF56010	80441108		
0x00001110	C8C8F8F8	90FEFF8F	FFFFF8F8	D0F8F8F8		
0x00001120	3334A23A	89926225	370A8A92	98102C24		
0x00001130	F8F8F8F8	FFF8F8F8	FFF8F8F8	FFF8F8F8		
0x00001140	D40D0D28	E8C8C8D0	19800DC	8296A0C8		
0x00001150	F7F8F8F8	F8F8F8D0	FFFFF8D0	7CF8F8D0		
0x00001160	178A1150	77110A80	3028A01	419A814		
0x00001170	F7F8F8F8	AD77F8F8	6F77F8F8	FF77F8F8		
0x00001180	4A980204	EAD8C4D0	80C9A480	40C8A0C8		
0x00001190	CF78F8F8	F8F8F8F8	CF8F8F8F	FF8F8F8F		
0x000011A0	B3113A1E	32700730	20002003	113A2A1A		
0x000011B0	7FF7377F	87637F66	FFFF777F	FFFF777F		
0x000011C0	24C8F8F8	A40C2A0	8A50A86	A8140234		
0x000011D0	DEA8F8F8	DE70FEFF	DFFCF8F8	CF8E8D8D		
0x000011E0	21803053	12796349	2961236A	A3120328		
0x000011F0	F7F8F8F8	FFF8F8F8	F8F8F8F8	3777F8F8		
0x00001200	0A8080C6	40C8C4D0	80C5D4A	54CC5289		
0x00001210	FCC8F8F8	F8C8F8F8	F87F8F8F	F8F8F8D0		
0x00001220	65689722	D03E2E30	D3A134E	11216011		
0x00001230	8F48F8F8	3A78F8F8	F7F8F8F8	3078F8F8		
0x00001240	C0860E88	CC08D4D	4090A8C	14028840		
0x00001250	CF8C8F8F	A84E8F7F	D8E8E8EA	FF8E8F7F		
0x00001260	10032023	02E49332	3A7011A	82E10A80		
0x00001270	B8FF8E2F	FFF8F8F8	F7A3A8FF	8378F8F8		
0x00001280	78C2A0C8	8A80C4D0	40A0C4D0	8C05A418		

Registers

Reg	Value
PC	0x000000C8
R0	0xFFFFFFFF
R1	0x00000000
R2	0x00001004
R3	0x00000020
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x7050C8A0
R9	0x00000005
R10	0xFFD02000
R11	0x00000000
R12	0x00000000
R13	0xFFFFFFFF
R14	0xFFFFFFFF
R15	0x00000013

Disassembly / Breakpoints / Memory / Watches / Trace

Terminal

```

TMS UART link established using cable "DE-9C [USB-1]", device 2, instance 0x02
arm-11c86-eabi-objdump -d -O -M reg-names-std "C:/ece463/Project_2/Q2/address_map_arm.srec" | tee
"C:/ece463/Project_2/Q2/address_map_arm.ssf.objdump"
Source code loaded.
Program stopped @ 0x000000C8

```

Info & Errors

```

arm-11c86-eabi-objdump -d -O -M reg-names-std "C:/ece463/Project_2/Q2/address_map_arm.srec" | tee
"C:/ece463/Project_2/Q2/address_map_arm.ssf.objdump"
Source code loaded.
Program stopped @ 0x000000C8

```

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.