**ECEGR 2220: Microprocessor Design**

**Spring 2018**

**LAB 4 REPORT**

**Name:**

**Performed by:**

**Don Le - Thanh Nguyen – Lauren Molina**

**Date of Report: 05/13/2018**

**SEATTLE UNIVERSITY**

**Department of Electrical and Computer Engineering**

**Part 1:**

We included 6 signals:

addsub\_result: result of adder\_subtracter

addsub\_carryout: the carry out of adder\_subtracter

shift\_result: results of sll and srl

and\_result: result of and

or\_result: result of or

final\_result: result of ALU

Fulladder, adders\_subtracter and shift register are included as necessary components.

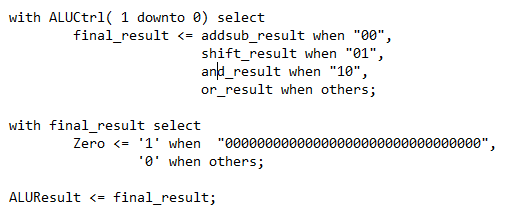
For adder\_subtracter and shift\_register, as we already included these as components, we used port map to utilize them.

In adder\_subtracter port map, we used index two of ALUCtrl to dedcide add or sub, if ALUCtrl (2) = 0 🡪 add, if ALUCtrl(2) =1 🡪 sub.

In shift\_register port map, we used index three of ALUCtrl to decide sll or srl, if ALUCtrl(3) = 0 🡪 sll, if ALUCtrl(3) = 1 🡪 srl . We used DataIn2(10 downto 6) for shamt.

We used logic *and* for and instruction, *or* for or instruction.

We used “with-select” statement to get ALU result and zero output.



From this point, we can have the truth table for ALU control input decoding:

|  |  |
| --- | --- |
| **ALU actions** | **ALU control input** |
| add / addi | 00000 |
| sub | 00100 |
| and / andi | 00010 |
| or / ori | 00011 |
| sll / slli | 00001 |
| srl / srli | 01001 |

**Part 2:**

In this part, our code will get from user a floating-point value of Fahrenheit temperature as an input, then convert it to temperatures in Celsius and Kelvin. A function name “converter” is called in the program to do the mathematical conversion between the temperatures, which later returns values of temperatures in Celsius and Kelvin.

We used registers ft1 for F temperatures (input), fa1 for C temperature, fa2 for K temperature.

Later, these temperature values will be moved to register fa0 for printing.

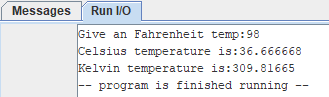


Figure 1. Screenshot of the program result.

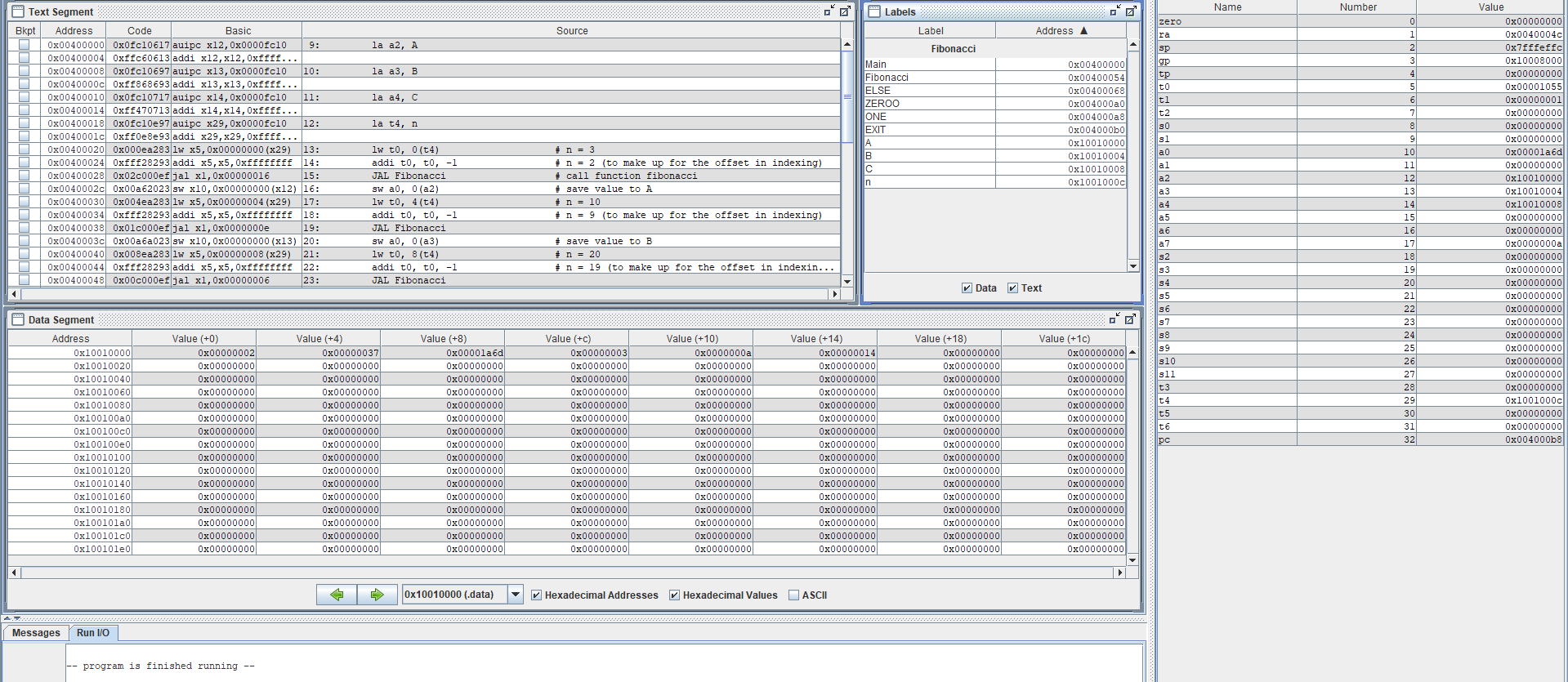


Figure 2. Correct conversion

As we can see, the program asked us for an input for F temperature. We used 98 degrees as our input. The results of conversion were 36.666 degrees C and 309.81665 degrees K, which were correct compared to data in figure 2. This shows that our program function properly.

1. Fobinacci

We made an array of the numbers for n and called the Fibonacci function 3 different times. Each time we called the Fibonacci function, we relabeled n to go to the next value in the main function. Within the Fibonacci function, we created 3 main statements to branch off. The first branch made the value 0 if n was less than 1. The second branch made the value 1 if n was equal to 1. The third branch saved the return address and n value from the main function to the stack and using n-1 and n-2 continuously called the Fibonacci function and added values together until the local n value in the function caused the Fibonacci function to end. At the end of the Fibonacci “else” function we re-loaded the original return address and deallocated the memory from the stack. The last step was to add Fibonacci(n-1) and Fibonacci(n-2) to get the actual value.



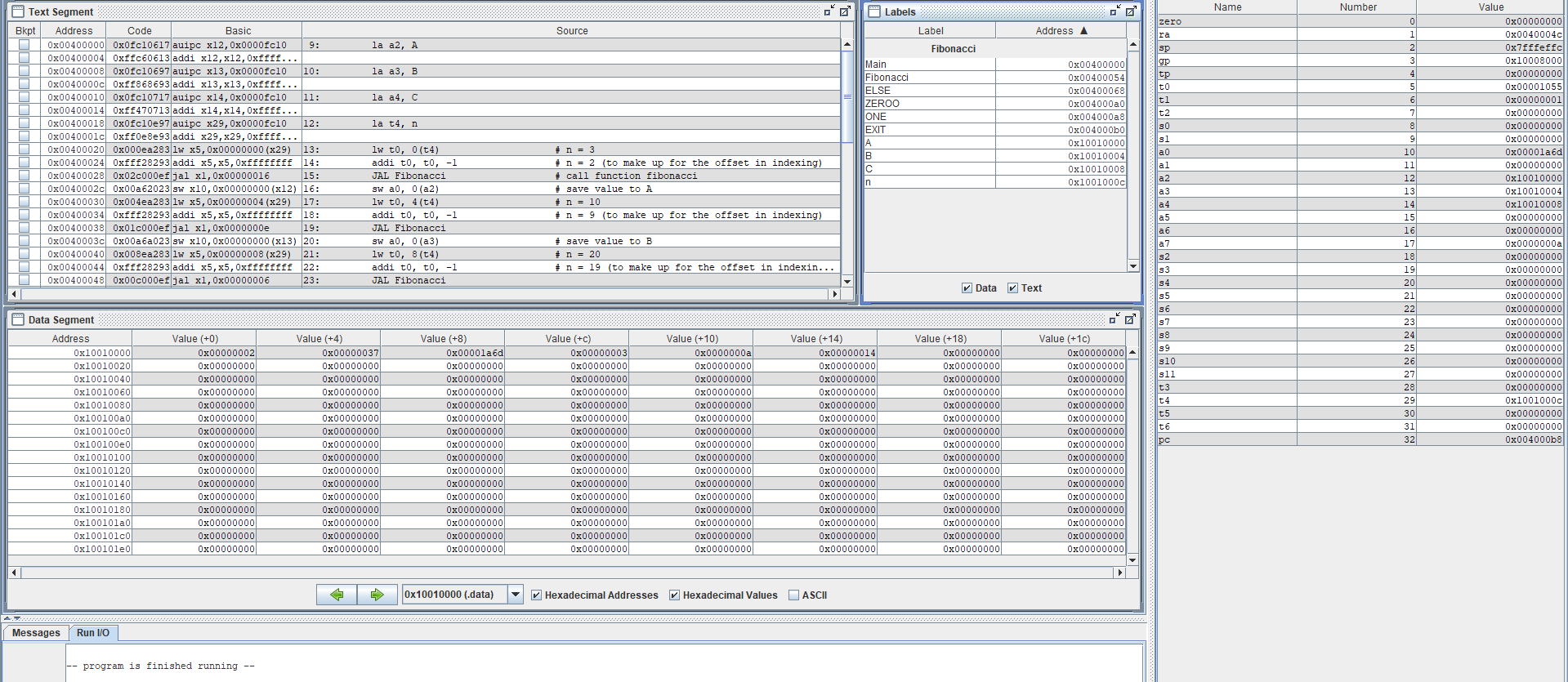


Figure 3. Results of RISC-V code for Fibonacci



This part is used to test those functions: ADD, ADDI, SUB, AND, ANDI, OR, ORI, SLL, SRL. Respectively, the test result for these functions are:

* + Test\_add (saved in register s0)
  + Test\_sub (saved in register s1)
  + Test\_addi (saved in register s2)
  + Test\_and (saved in register s3)
  + Test\_andi (saved in register s4)
  + Test\_or (saved in register s5)
  + Test\_ori (saved in register s6)
  + Test\_sll (saved in register s7)
  + Test\_srl (saved in register s8)

The two datain\_a = 0x01234567 and datain\_b = 0x11223344 are saved in registers a0, a1. The immediate used here is 0x00000011.

**TESTING ALU:**

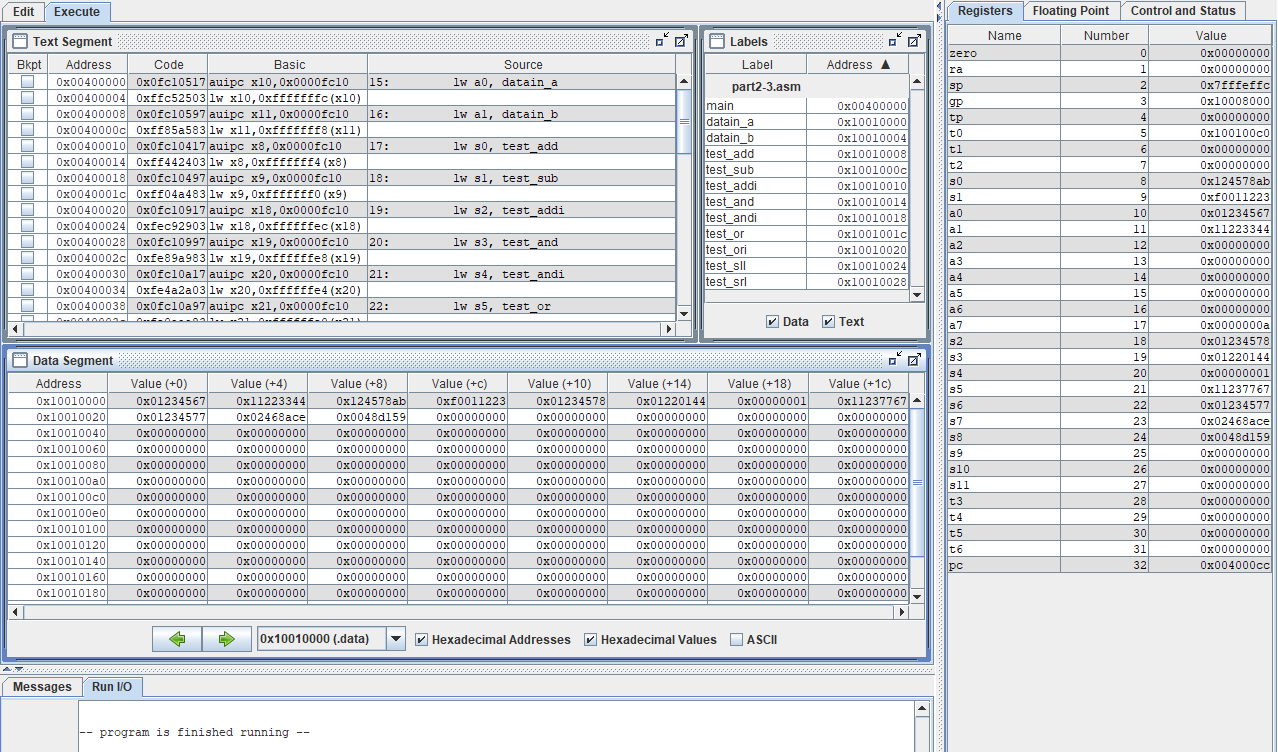


Figure 4. Results of RISC-V code to test ALU

As we can see from the results in figure 4.

* + Test\_add (saved in register s0) = **0x124578ab**
  + Test\_sub (saved in register s1) = **0xf0011223**
  + Test\_addi (saved in register s2) = **0x01234578**
  + Test\_and (saved in register s3) = **0x01220144**
  + Test\_andi (saved in register s4) = **0x00000001**
  + Test\_or (saved in register s5) = **0x11237767**
  + Test\_ori (saved in register s6) = **0x01234577**
  + Test\_sll (saved in register s7) = **0x02468ace**
  + Test\_srl (saved in register s8) = **0x0048d159**

Comparing these values to the expected values shown in the tALU.vhd code, and the values we got when running it, we could say our ALU.vhd and tALU.vhd function properly.

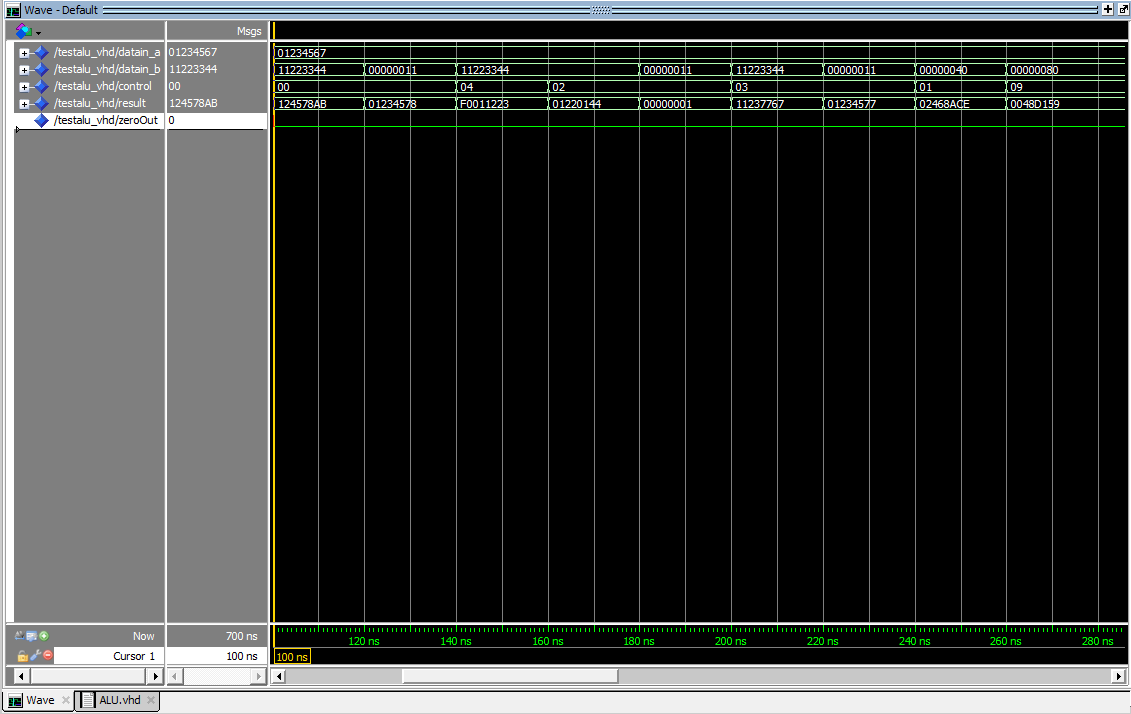


Figure 5. Values from running ALU

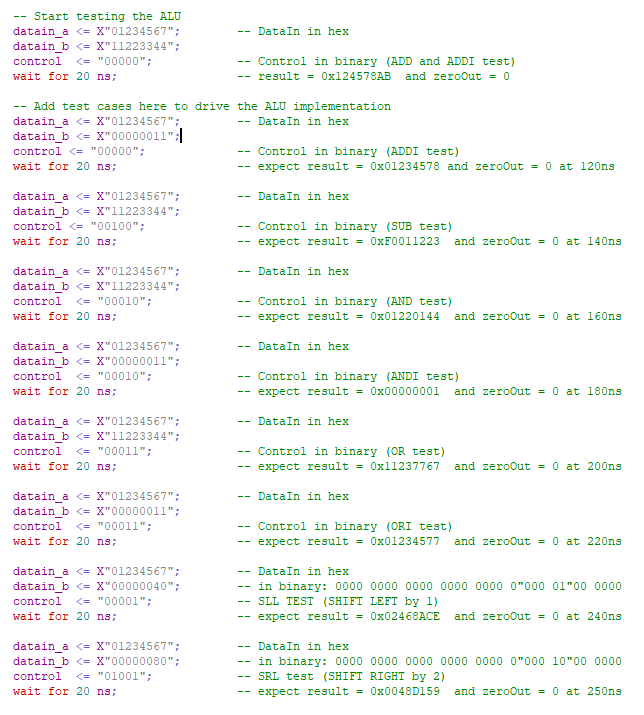


Figure 6. tALU code with expected values

* Link for our github:
* Link to our master branch:

<https://github.com/SU-ECEGR-2220/AVOCADOS/tree/master/Lab%204>

* Link to individual branches:

Thanh: <https://github.com/SU-ECEGR-2220/AVOCADOS/tree/Thanh/Lab%204>

Don: <https://github.com/SU-ECEGR-2220/AVOCADOS/tree/Don/Lab%204>

Lauren: <https://github.com/SU-ECEGR-2220/AVOCADOS/tree/Lauren/Lab%204>

* Note:
* Each individual built the program then saved and test on his/her own branch first. Then we came up with the best version of the program and pulled it to the master branch. You can find the contribution of each member towards the project in the individual branches. Thank you!