May 7, 2019

Term Project Report

PREPARED fOR dR. cARROLL cSE 2441

pREPARED BY Bailey Brown

**Table of Contents**

List of Figures/Tables ....……………………………......…………………………………….….. 2

Introduction ……………………..……………………………………………………………….. 3

Project Overview……………...………………...……………………………….……….. 3

Project Status…………………..…………………………………………………….…… 3

Report Overview…....……………………...…………………………………………….. 4

System Design ……………………………….……………..……………………………….…… 4

System-level Description ………………………………….……………………….…….. 4

Subsystem Descriptions …………………………………….…………………………… 5

Hierarchical Design Structure …………………………………………………………… 7

Controller Design ……………………………………………..………………………………..... 8

Functional Description …………………………………………………………………... 8

State Diagrams ………………………………………………...………………………… 8

Verilog Code …………………………………………………..………………………… 9

Alternative Design Considerations ………………………………………………..…..………… 9

Alternatives Considered & Final Decision…………………….……..………..………… 9

Integration and Test Plan …………………………………………………………...……….….. 10

Integration Strategy …………………………………………………..………………… 10

Test Strategy ………………………………………………………………………….… 10

Simulation Results …………………………….…………………………..………….… 10

Test Results ……………………………………………………………………………... 11

Conclusion …………………………………………………………………………..…………. 11

Resolution of Design/Implementation ….....….………………………………………... 11

Lessons Learned …...…………………………………………………………………… 11

**List of Figures and Tables**

Schematic diagram of processor at system level pg. 4

Schematic Diagram of control unit pg. 5

Verilog code for instruction decoder pg. 5

Schematic diagram of instruction register pg. 5

Schematic diagram of program counter pg. 6

Schematic diagram of accumulator pg. 6

Schematic diagram of address selector pg. 6

Schematic diagram of arithmetic logic unit (ALU) pg. 7

Diagram of hierarchical design structure pg. 7

State diagram of controller pg. 8

Verilog code of controller pg. 9

Simulation results of processor pg. 10

Processor output table pg. 11

Introduction (1)

Project Overview (1.1)

The goal of this project was to design and implement a processor that could execute a program consisting of a limited number of instructions. This project was separated into two parts; each part required that we execute a program with sixteen instructions; in part A we had three unique instructions, increment, clear, and jump, and in part B we had six, the previous three in addition to load, store and add. The requirements of part A were to integrate the TRISC components we had previously designed in laboratory exercises and homework assignments. This consisted of a program counter, accumulator, instruction register, control unit, and RAM that was given to us. Part B required us to integrate an address selector, an arithmetic logic unit with a buffer register, and a redesigned control unit that could handle the program with a wider instruction set.

The design portion of this project was done on Quartus II software. Designs consist of both block/schematic diagrams and Verilog code. Throughout the semester, we designed and tested various parts of the processor individually. This project focused on teaching us how to use them to create new components and as components themselves and making them work together, especially making use of buses. It was required that we had Memory Data In, Memory Data Out, Address, and Control buses. These were used to transfer data from one component to another, because it would have been virtually impossible to connect them directly.

The testing portion of this project was done mostly on the DE1. Once we finished our design, we had to assign outputs to various pins on the DE1 so we could understand visually what the processor was doing. The clock of the processor was assigned to a key button so that we could see it traverse through each state individually. The DE1 was extremely useful with debugging and finding problems in the designs. Once the processor was working, it was also used to display the correct output sequence.

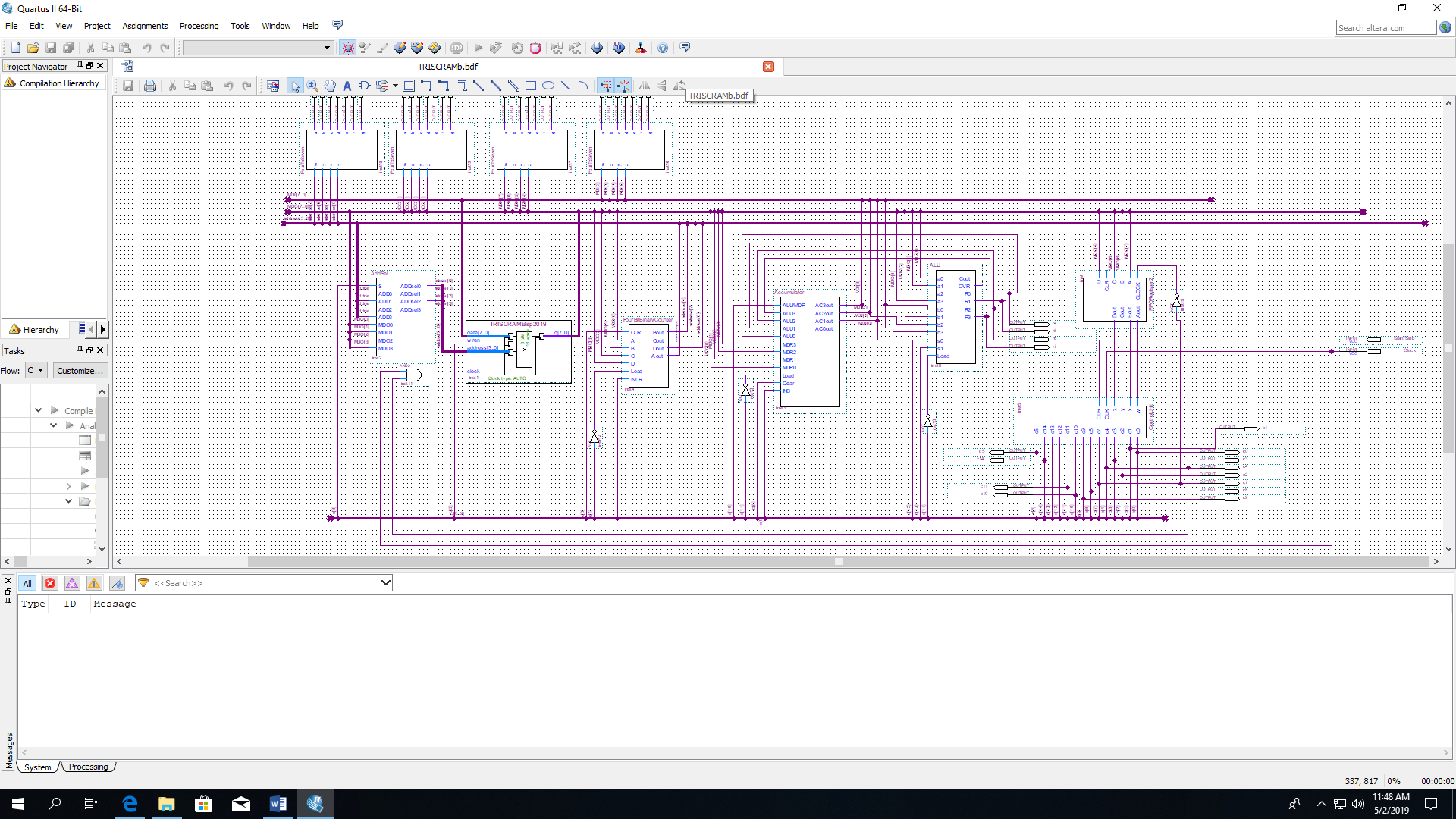
Project Status (1.2)

I was able to complete both parts of the project by their assigned early due dates. My processor was able to run and successfully execute the programs given to us with the required instructions. One of the most difficult parts was loading in the data to each component in the correct order because the input and output order of the four bits varied across each component. I was not able to fix this inconsistency in my design and had to rely on testing different wiring patterns to see which way was correct. Besides this, there were no other major problems faced that I could not fix.

Report Overview (1.3)

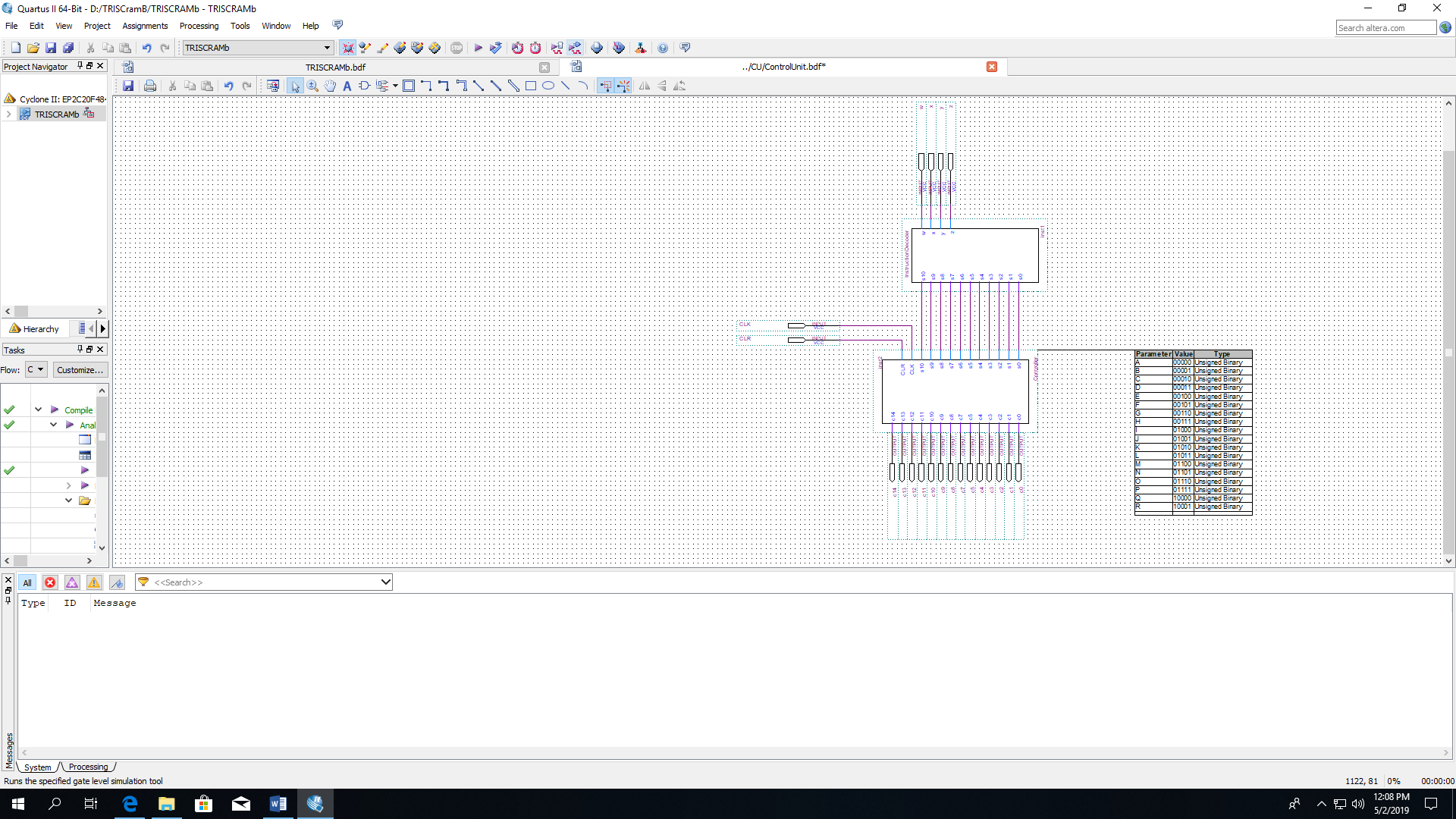
This report will provide more detail about the project explained above. The functionality and design of the processor, as well as each component individually, will be discussed in detail. Understanding how every part worked on its own was key in getting them to work together. The controller of the processor and how it was designed and used will be specifically focused on since it is the most significant component. Following this, I will discuss alternatives to my design, but more specifically, why I chose not to go with them. Finally, I will thoroughly go over my strategy for integrating and testing these components.

System Design (2)

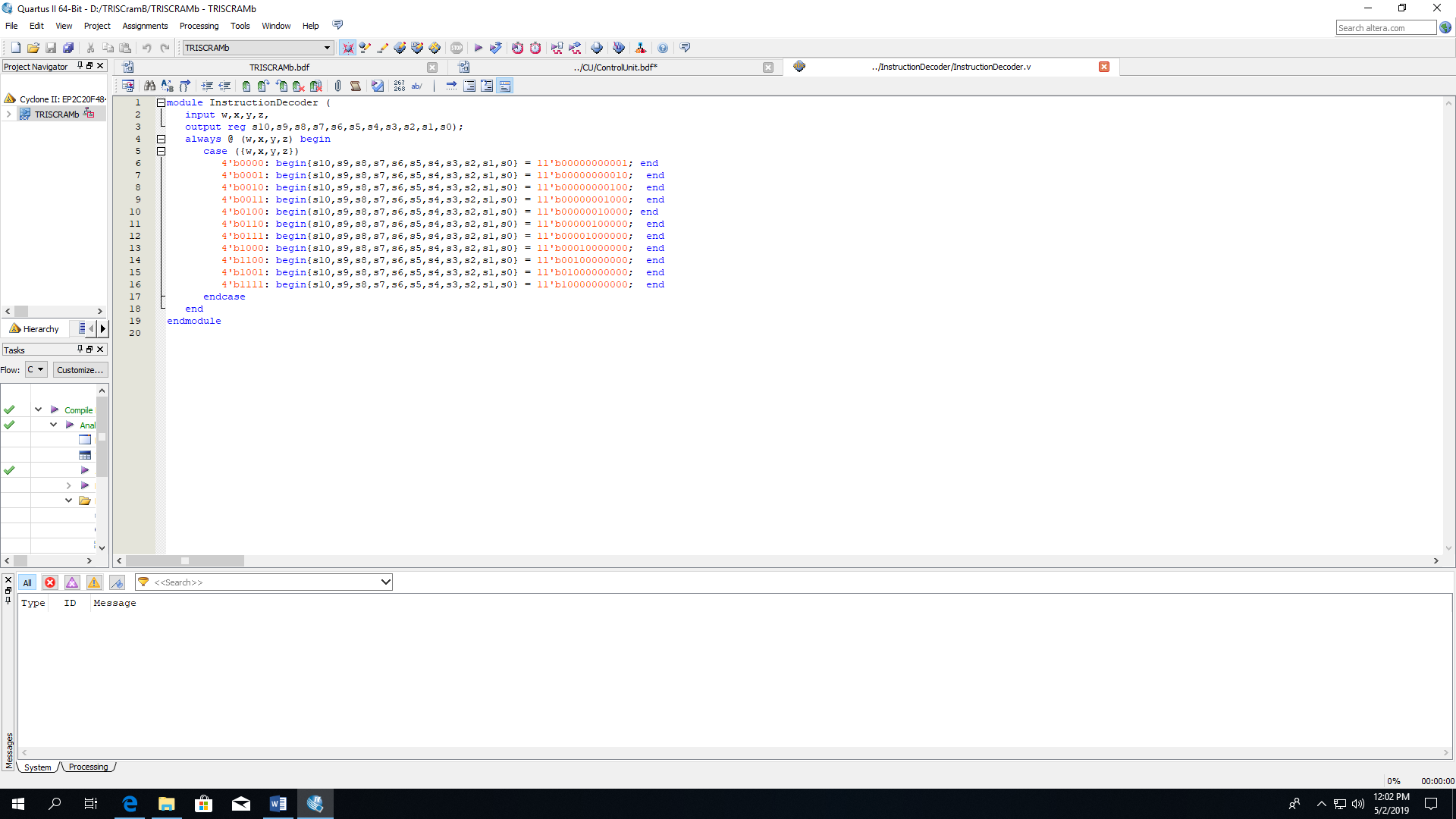
System Level Description (2.1)

Every action performed by the processor is determined by the control unit on the far right. A cycle begins with the RAM reading in the instruction from a memory location specified by the address selector. For this part, the memory address is given by the program counter. The four-bit op code for the instruction is sent to the instruction register to be decoded and fed to the control unit. The control unit then sends control signals to the control bus and to the respective components that need to perform an action. As this was happening, the program counter would increment the address the RAM would read from so it could perform the next instruction. The increment instruction will send a signal that increments accumulator. Clear will clear the accumulator, setting its counter back to zero. Jump will load the data stored in the memory data out bus to the program counter so it will output it to the address bus, so the RAM can read from this address next. Load will have the RAM read from the address in the memory data out bus and load that data into the accumulator. Store will enable the RAM write cycle and execute it so it will write the data stored at the address specified in the memory data out bus. Finally, add will take the data stored in the memory location specified by the memory data out bus, send it to ALU to be added to the data in the memory data in bus, and load the result into the accumulator. After each instruction is complete, the instruction register will fetch the op code at the next address in the RAM and the cycle would start again.

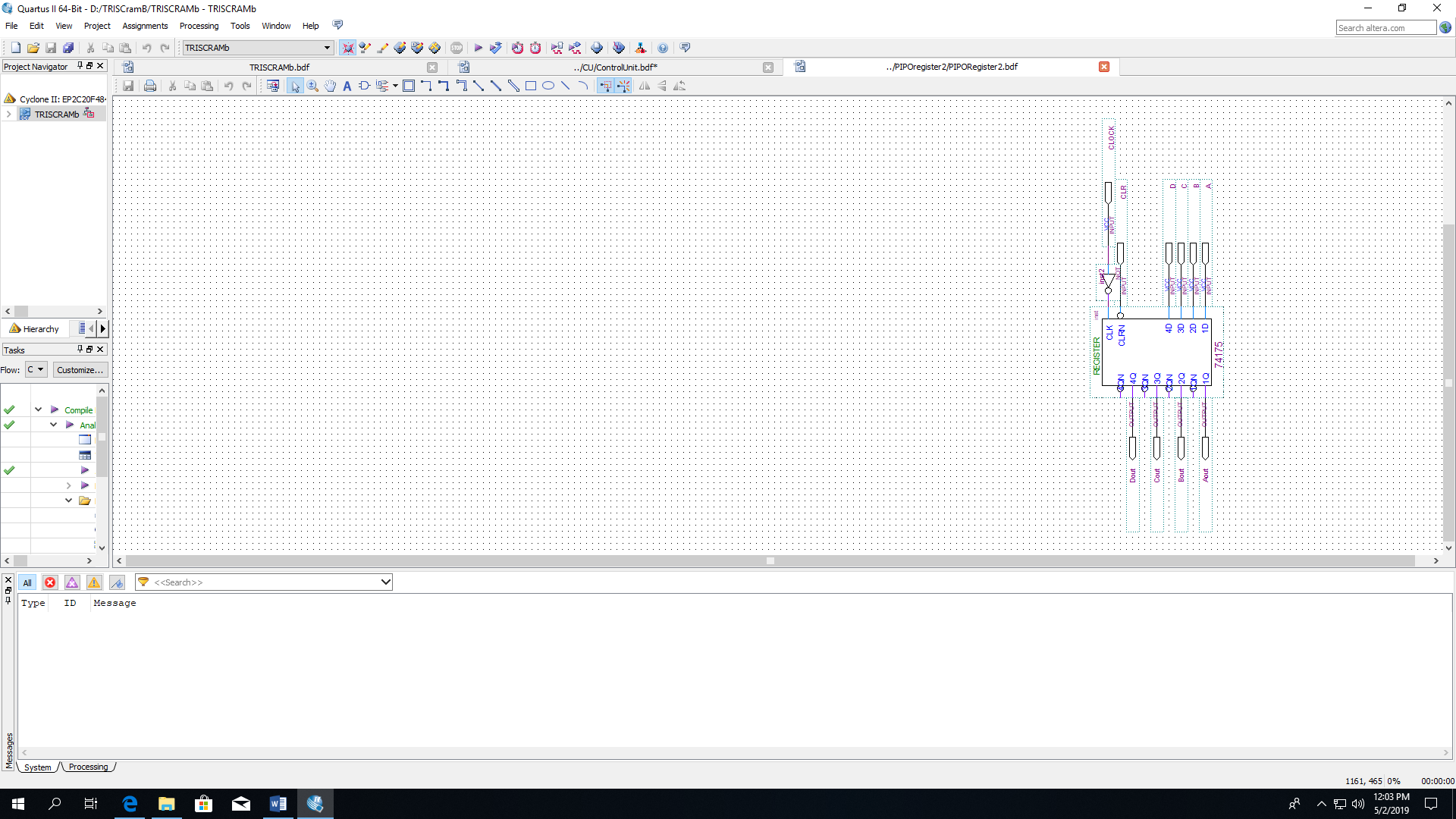
Subsystem Descriptions (2.2)



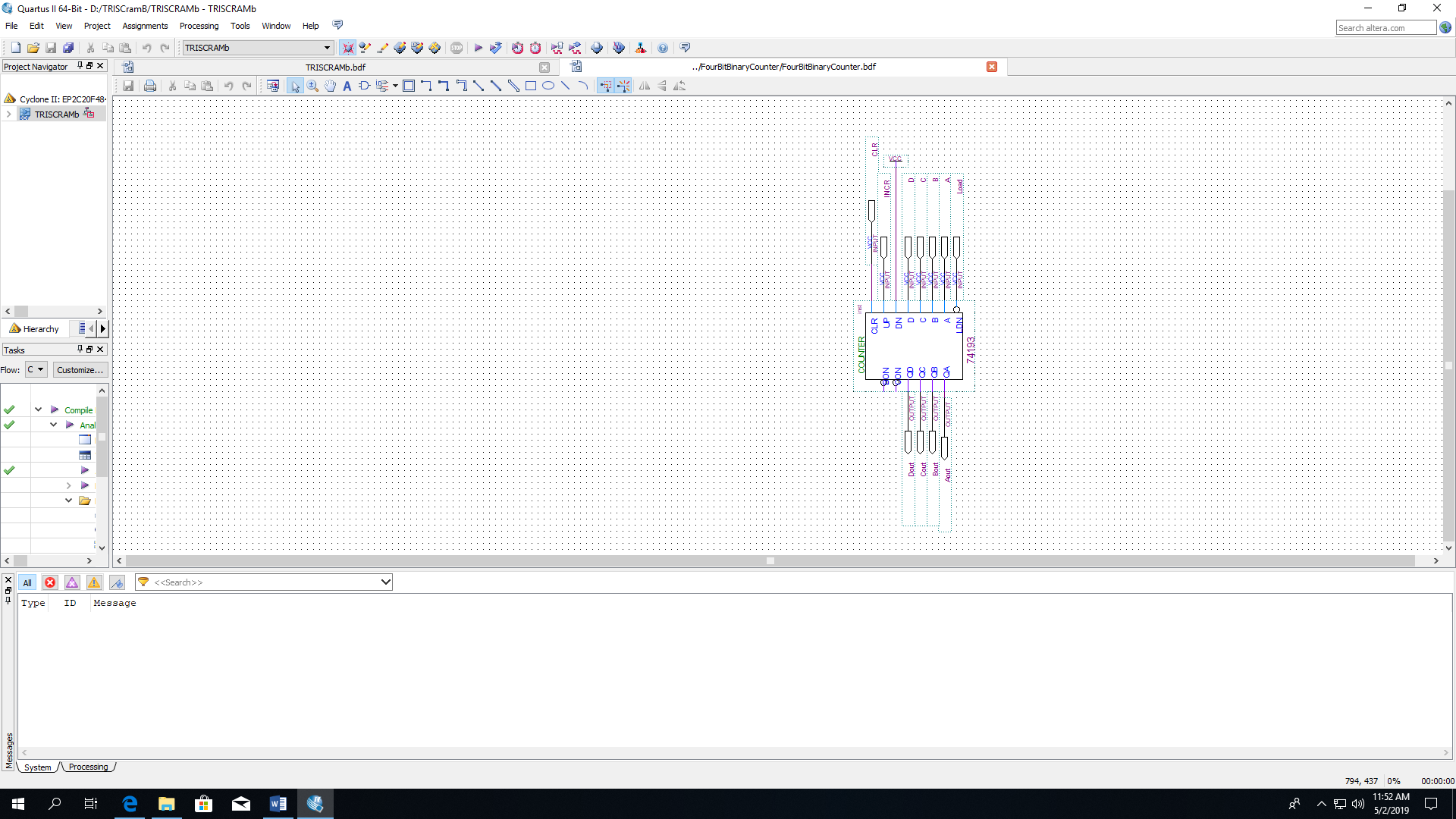
The control unit is the most significant part of the processor. It has an instruction decoder that interprets the op code for the controller beneath it. The controller dictates everything the processor does and will be discussed in the most detail later in the report.



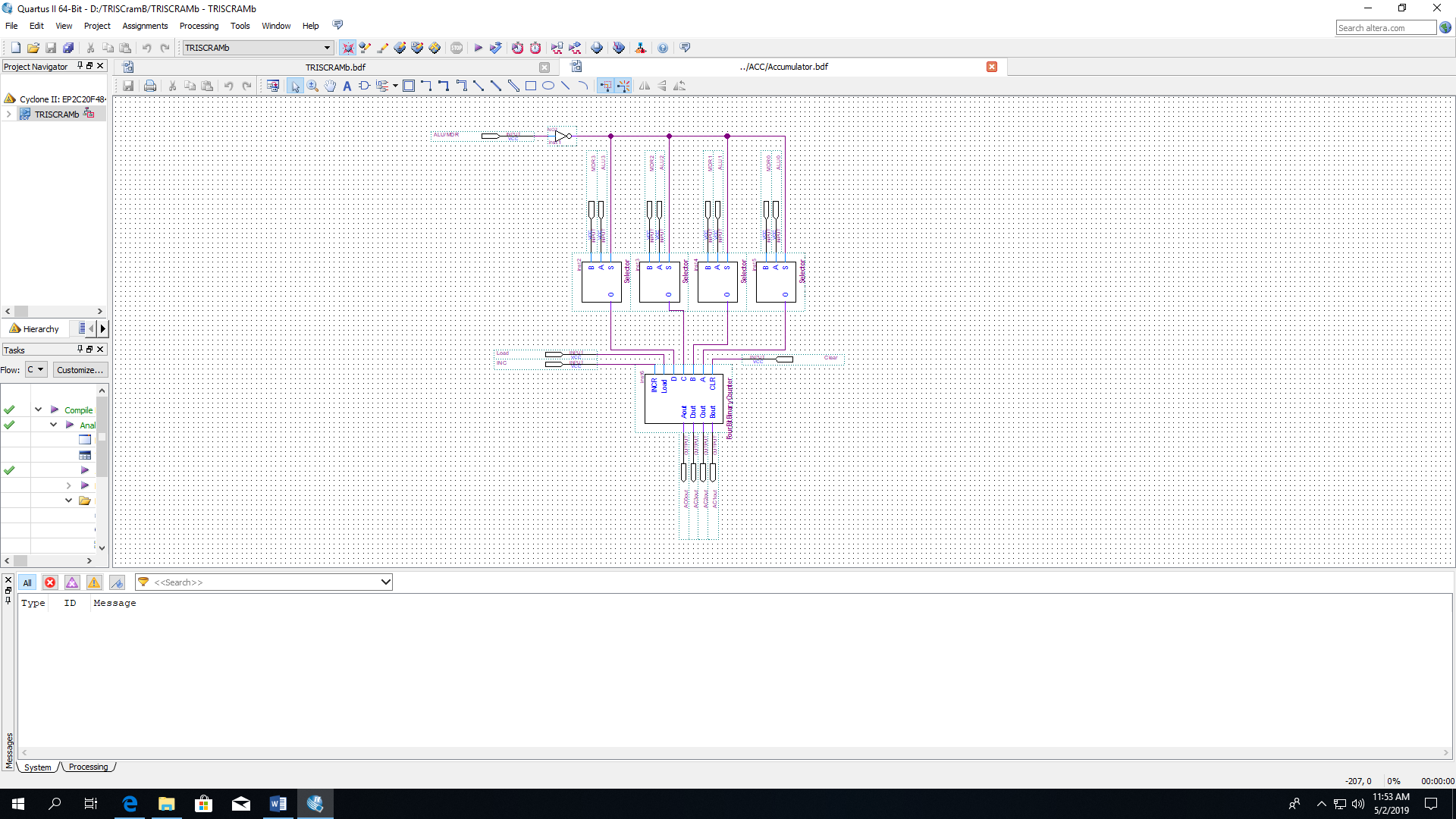
The instruction decoder is basically a four to eleven-bit decoder. The four-bit op code is the input and it outputs an eleven-bit active high sequence to the controller, so it understands which instruction it needs to execute.



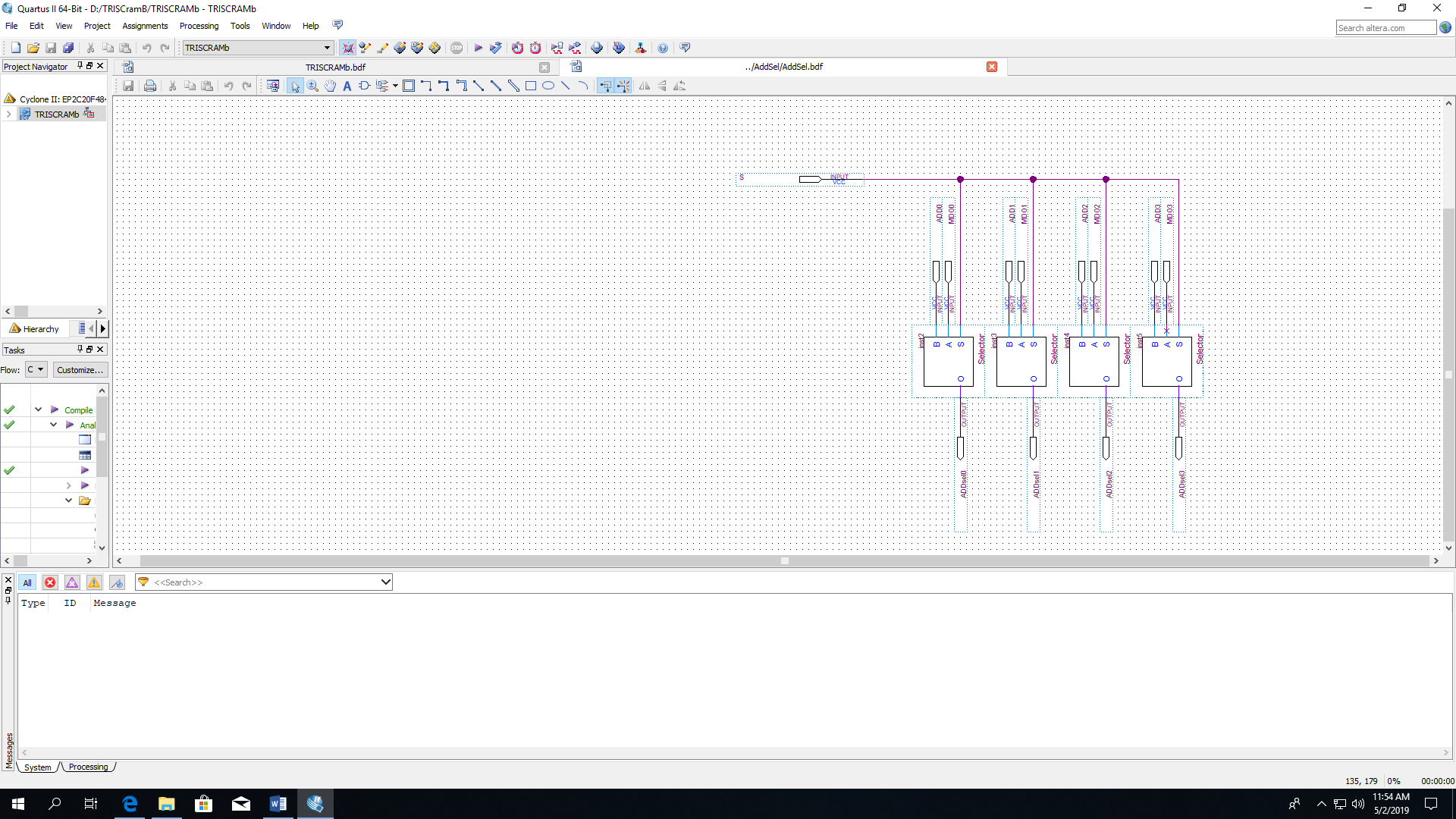
The instruction register is only a four-bit binary register. When data is read from the RAM and sent to the memory data out bus, the instruction register receives the instruction op code in the last four bits. The input is held in the register until the control unit triggers its load and the code is output to the instruction decoder.



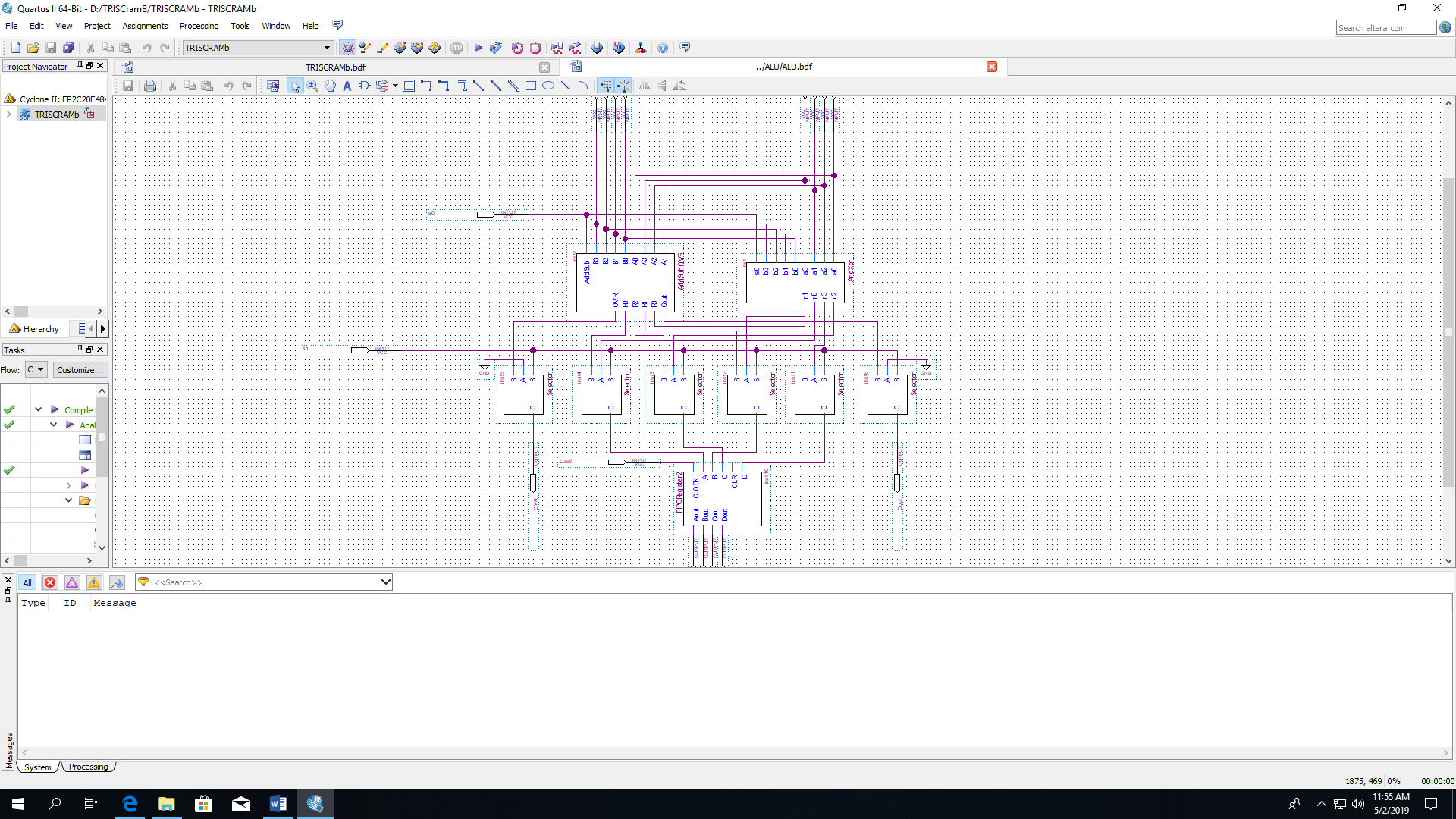
The program counter is an active low four-bit binary counter. It starts at zero and its purpose is to increment the address in the RAM so we can fetch the instruction op code at every address. When the processor needs to move to the next address, the control signal to increment the counter is sent and the program counter outputs the next address value to the address bus and is read into the address input of the RAM.



The accumulator is a four-bit binary counter that can take input from two different sources. Eight bits are loaded into four two-to-one multiplexers and the control unit will send a signal to select which input to use based on the instruction it is given. The inputs that are selected are then loaded into a four-bit binary counter and sent to the memory data in bus. New data will be sent every time this counter is incremented or loaded.

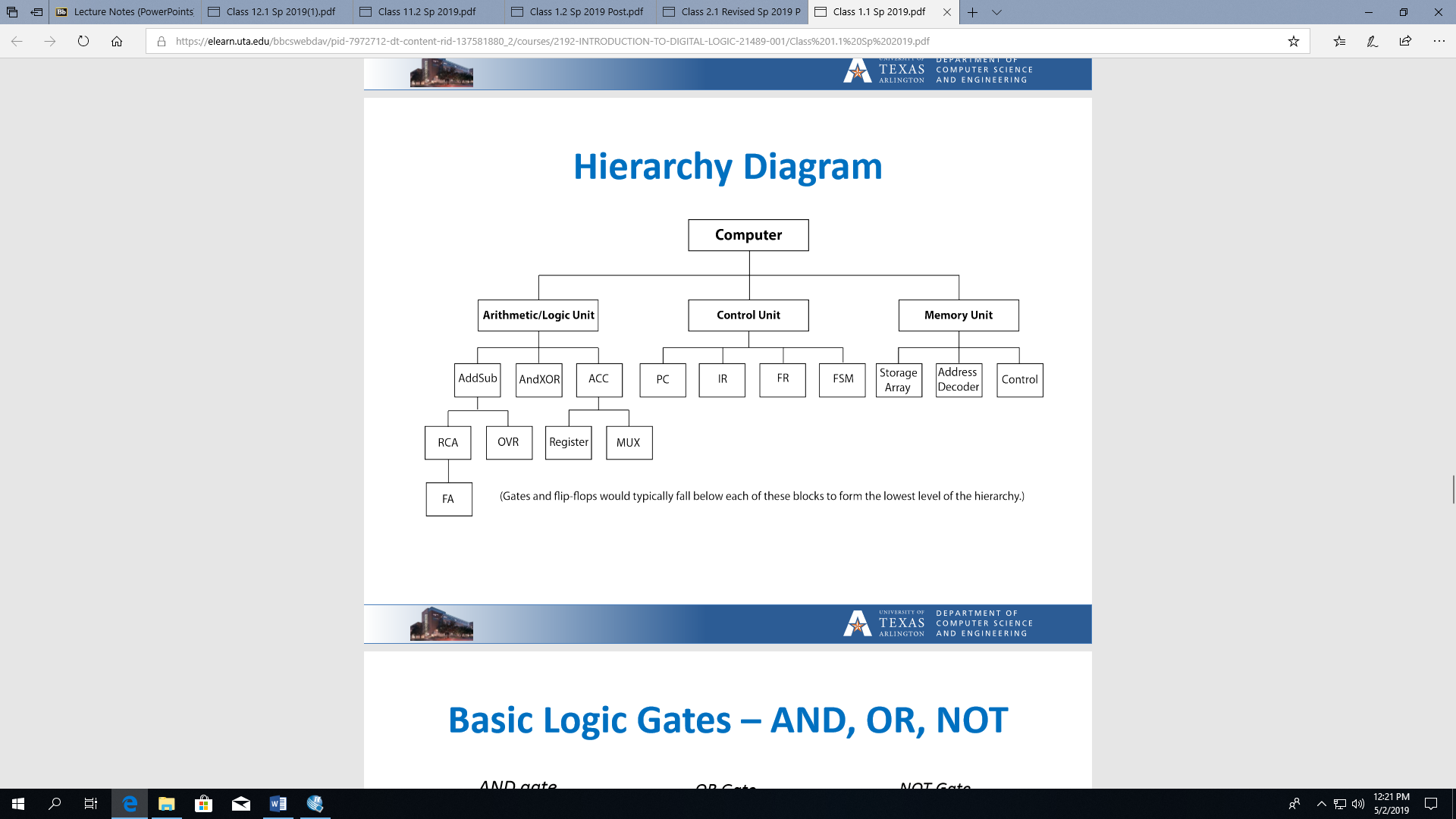


The address selector consists of four two-to-one multiplexers. The purpose is to select which source, the memory data out bus or address bus, the RAM would get the next address to read an instruction from. When needed, the control unit will signal which input data will be used and the RAM will read from the address the selector outputs.



Finally, the last component added was an arithmetic logic unit (ALU). It has two sets of four-bit numbers, one coming from the memory data out bus and the other coming from the memory data in bus. These two numbers can have four different operations (add, subtract, AND or XOR) done on them, depending on the signals sent to the two selector inputs from the control unit. For our project, we added the two numbers and then loaded them into a four-bit binary register called a buffer register that would output the result directly into the accumulator.

Hierarchical Design Structure (2.3)



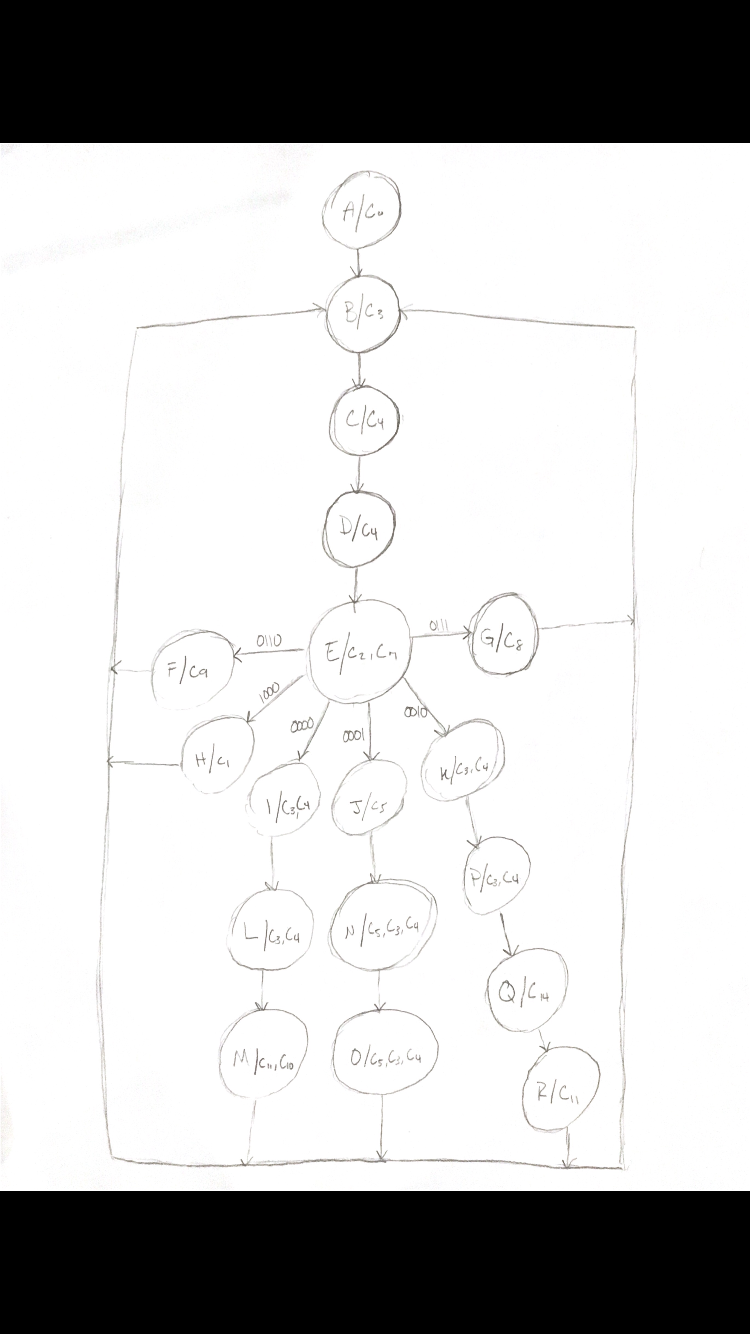
The diagram above shows the hierarchical design of the processor. It starts at the highest level and shows the lower level components needed to make each work.

Controller Design (3)

Functional Description (3.1)

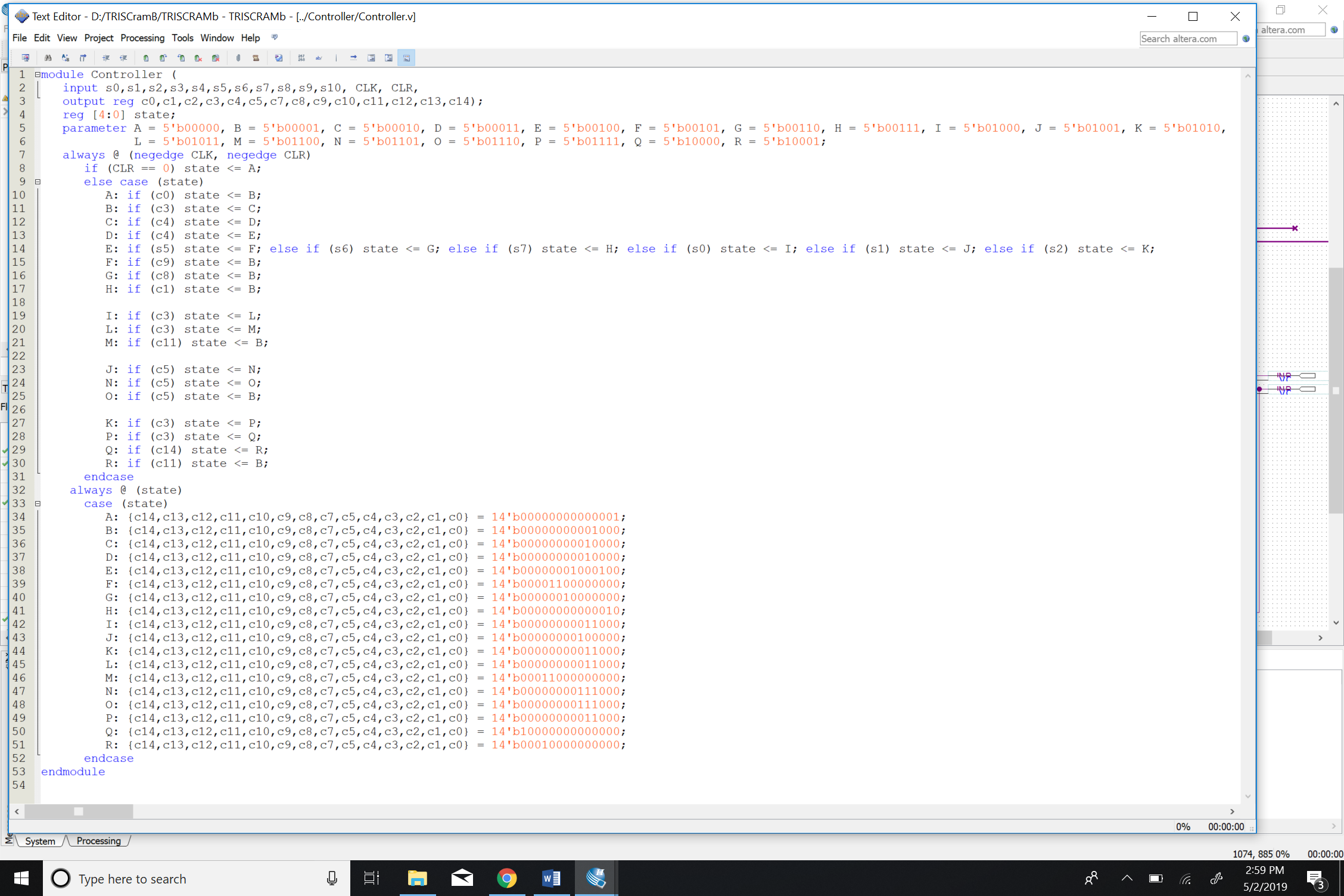
The purpose of the controller is to control the processor and send the correct signals to each component based on the instruction it is given. The controller is a finite state machine and everything the processer does is based on how the controller traverses through its various states. The controller is given 11 inputs from the instruction decoder and uses this to determine the output sequence, which is different for every instruction.

State Diagram (3.2)



Creating a state diagram was the first step in the design process for the controller. This state diagram was used to determine how the controller would be designed using Verilog. As seen from the diagram, the controller starts in a beginning state A that is triggering the output that clears the program counter. The controller will only be in this state when the system clear button is pushed, starting the entire process over again at address zero. The controller will move to a new state every time the clock is toggled. When it moves to state B, the control signal that selects the memory address source to be loaded into the RAM is triggered. The next two states will load and transfer the data at that address in the RAM to the memory data out bus so the instruction register will have access to it. The last four digits of this data is the op code for the instruction. The next state will increment the program counter and load the op code in the instruction register to be decoded for the controller. From this point, the controller will perform said instruction, and the direction it takes through the state diagram is dependent on which instruction was given.

Verilog Code (3.3)



Alternative Design Considerations (4)

Alternatives Considered and Final Decision (4.1)

When I first began creating the state diagram for the controller for part B, I noticed that the three instructions we added all included executing the RAM read cycle from the address in the memory data out bus. I thought it would make sense for all these instructions to share these states since they were almost identical. Attempting to differentiate which instructions required these states and the specific instruction that was being executed after it went through these states became, in my opinion, unnecessarily difficult. It also made my state diagram look cluttered and hard to understand. Because of this, I decided to make each instruction have a separate path after state E. This made my state diagram easier to follow, and although my Verilog code was longer, it was easier to write.

Integration and Testing (5)

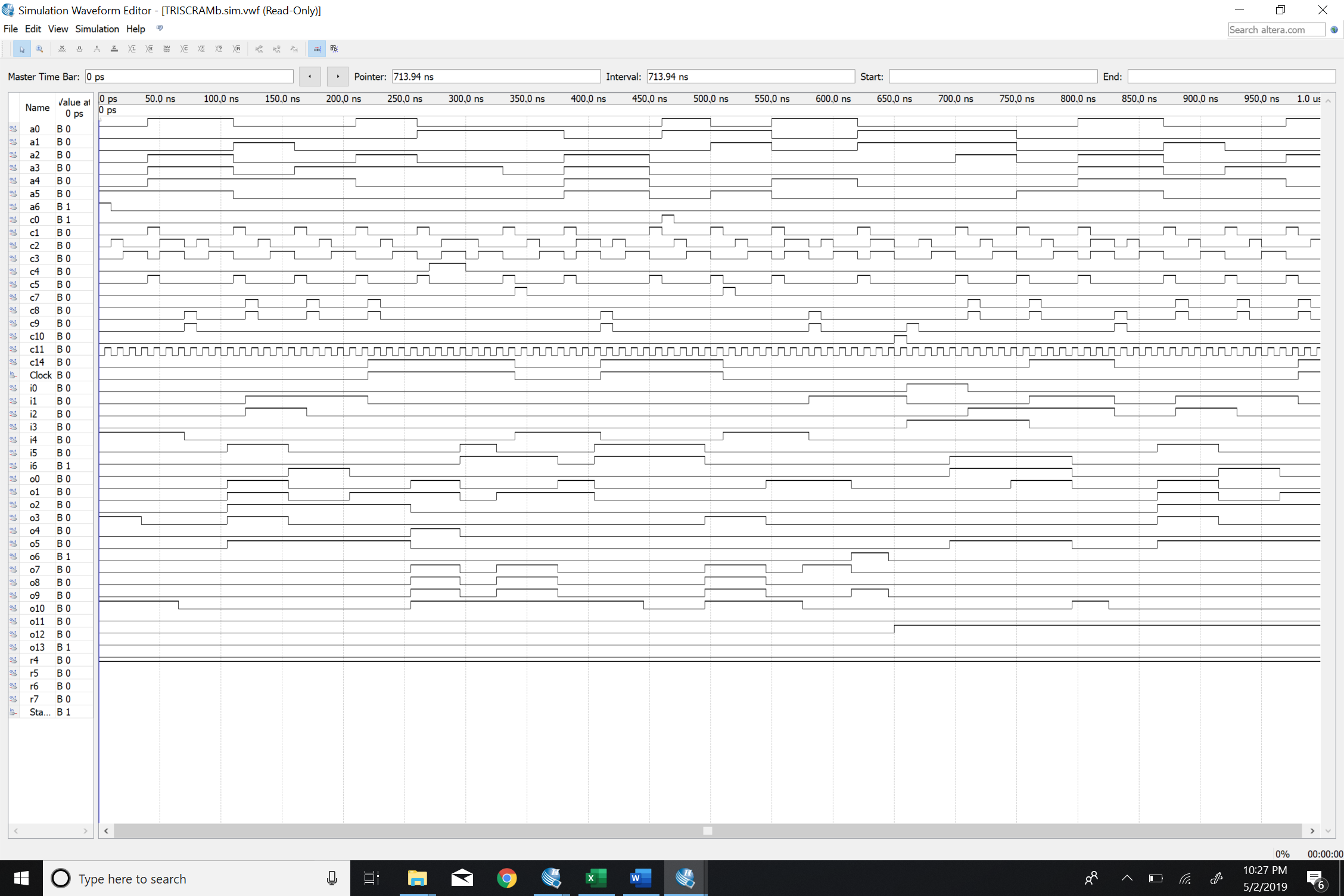
Integration Strategy (5.1)

Integrating the separate parts and getting them to work together was much more difficult than I imagined. My initial strategy was to test the processor using simulation after adding each component. This was not as efficient as I’d hoped, though, because there was a very limited amount of states that could be tested without all the components. The first meaningful simulation was done after integrating the instruction register, controller, RAM, and program counter. After that, I added in the accumulator to complete part A. For part B I decided to integrate the address selector and ALU at the same time because I thought it would be more of a hassle to do one at a time.

Another source of difficulty was figuring out ff the components were active high or active low. I often found when I when my processor was stalling it was because I was not sending out the correct signal to load of the component. Adding a not gate fixed this problem whenever it occurred, but figuring out which components needed one while integrating them took a lot of time

Testing Strategy (5.2)

I originally planned on using the simulation software in Quartus to make sure my design was working before testing it on the DE1. I did this when I had only the first four components, but I quickly realized it was hard to interpret the simulation, especially after adding in the accumulator. I decided to program the DE1 and do the remaining testing on it. I used the remaining LED pins we had to see how and if data was moving between the components; I connected them to wires in between components so I would know where data had stopped moving or if it was going in or out in the wrong order. This was really helpful, and debugging was much easier than it was when I tried using simulation.

Simulation Results (5.3)  


Test Results on DE1 (5.4)

|  |  |  |  |
| --- | --- | --- | --- |
| Memory Address | Memory Data Out | | Memory Data In |
| 0 | 0 | 8 | 8 |
| 1 | 6 | 1 | 9 |
| 2 | 6 | 2 | A |
| 3 | 6 | 3 | B |
| 4 | 1 | 9 | B |
| 5 | 7 | 5 | 0 |
| 6 | 0 | 9 | B |
| 7 | 8 | B | B |
| 8 |  |  |  |
| 9 |  |  |  |
| A |  |  |  |
| B | 7 | 0 | 0 |
| C | 0 | A | A |
| D | 2 | 8 | 2 |
| E | 6 | E | 3 |
| F | 6 | F | 4 |

Conclusion (6)

Resolutions (6.1)

I did not have any problems arise that I could not eventually figure out. My processor worked fine, and I was able run a program and execute the increment, clear, jump, load, store, and add instructions. Designing the processor, specifically the controller, was a challenge, but I am satisfied with the design I decided on and feel that it was best for this project. I faced several other challenges as well, most notably figuring out input and output order of the bits, figuring out the load of the components, and getting every component to work together. It took a lot of debugging and making use of the LEDs on the DE1 to figure out where the problems were originating, but I was able to correct all my mistakes.

Lessons Learned (6.2)

The most important lesson I will take away from this project is the value in having consistency in the design across all components that will be integrated. If I had focused on this from the beginning, I would have had significantly less problems to deal with. I also learned how important starting projects early are. This project took way more hours of work than I expected, and if I had not been given the incentive to start and finish both parts early, I do not think I would have given myself enough time to finish. Overall, I enjoyed this project and believe I will take away useful skills and concepts that will benefit me in my future classes and career.