CPE 271L

Final Project

Design of a Simple Central Processing Unit

12/12/2024

*Aidan Lett*

*Partner*

**Introduction**

Starting with the basics, what is a central processing unit or CPU? To put it simply it is the “brain” of a computer system. A CPU has the ability to contain all of the necessary components to process inputs, provide outputs, and store data value. A CPU processes and executes instructions given to it by programs written by us. CPU’s have a few different kinds of architectures including ARM and x86, and new CPUs can contain multiple cores to allow for parallel processing. Another important specification of a CPU is its clock speed, you could think of this as the metronome of a computer it helps to keep operations running at a consistent speed and also tells us how many operations can be run in a certain amount of time, this is measure in hertz and more commonly megahertz. For example the DE10-Lite board that we have been using in the lab this year has a 50 megahertz clock meaning in theory it could compute fifty million operations per second. Cache stores frequently used data, this improves access speeds. There are many other things that can be included in a CPU as well you can have multiple threads which can help to handle multiple tasks simultaneously, and integrated graphics. To sum it all up the CPU is the “brain” of a computer and allows it to execute programs.

**Hardware Description**

The hardware used for this project was the same as the hardware we have been using all semester for this lab which is a Terasic DE10-Lite FPGA board. This is a ALTERA MAX 10

based FPGA board that served as our reprogrammable circuit board. This board allows the user to use its features in the instances of embedded systems and computer organization. The DE10-Lite board has a range of advantages that depend on how the board is being used. One of the biggest advantages is its versatility, it can be reconfigured to deal with various project requirements. As far as from a budget standpoint, it’s pretty cost effective, especially considering the board versatility. This board also offers support for OpenCL, allowing users to FPGA-targeted programs in higher-level languages. In addition to this, its small size makes it perfect for projects where size comes at a premium. Now given all of the benefits we have to keep in mind that there are still a fair amount of downsides with this board. Notably its small size can restrict the amount of logic and memory available. On top of this there are the limitations of having limited inputs and outputs which can pose a problem if a project needs more than what's available. The need for an external USB-Blaster can also be a drawback, having a built in solution for this would be nice. But for the purpose of teaching and learning this board is a great middle ground as far as price and functionality, with built- in seven-segment displays, switches, push-buttons, and LED’s.

**Software Description**

In most of or lab’s this semester we used a software called Quartus Prime, this is a programmable logic design suite developed by Intel. Quartus Prime offers a variety of functions including HDL analysis and synthesis, examining register-transfer level diagrams, and running simulations that closely mimic hardware behavior. One of the big advantages of this software is its strong compatibility with FPGA devices, this allows VHDL code to be easily transferred onto physical hardware to get more tangible outputs through the LED’s and seven segment displays on our board. In addition to this the software has an integrated Waveform simulation tool which allows users to visualize signal transitions over time and better understand the relationship between those signals all without needing to use a physical device. Another notable feature is the ability to have logic circuit visual editing, this was super helpful during the first few labs where our understanding was still limited. Despite all of these positives there are some downsides as well, most notably the software does not work well on MacOS systems, as well as the high barrier of entry as far as price and understanding. Still, when the user has an understanding of how to use the software Quartus prime can be a very versatile and effective tool.

**VHD File Description**

The structure of the VHD files can be summed up like this, at the highest level there is the Simple CPU Template, which incorporates several underlying components, the ALU, Control Unit, Memory\_8\_by\_32, Program Counter, and Register. We’ll take a closer look at all of these. Starting with the ALU, its operation mirrors a standard arithmetic logic unit. It takes two values and uses a given opcode to determine the correct operation whether that be addition, subtraction, AND , OR, or simply passing an input. This logic is implemented via “if” statements that check the 3-bit opcode and execute the corresponding operation. The Control Unit is one of the more complex elements, as it has multiple roles. It acts as a finite state machine, navigates through various instructions and interprets them to initiate loading, adding, and storing sequences. Each sequence goes through multiple states, and the control unit also controls the activation of individual load bits within those states. The memory component stores 32 values that are 8-bits each. Inside it initializes a signal “z” and defines individual addresses for each memory location. It also writes new values to memory locations and provides outputs from specific addresses. The program counter collaborates with the ALU to increment a “counter” signal by one. This serves as a cue for the CU to proceed to the next instruction in the sequence. The register component takes an input value and stores it temporarily, it is controlled by a 1-bit load signal that will decide if the value is kept. The register is key for holding values until the next instruction is executed. The CPU itself integrates all of these pieces. It exposes various internal outputs while connecting the individual components described above. Signals map the input and output of each module, and then the port maps link them together, at times multiple instances of the same component are used to create a fully functioning CPU.

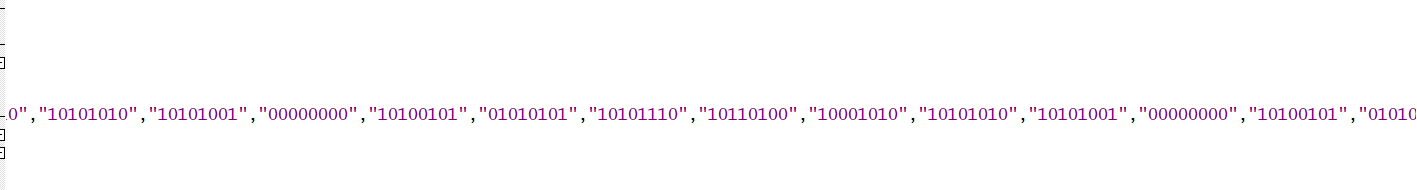
**Problems and Solutions**

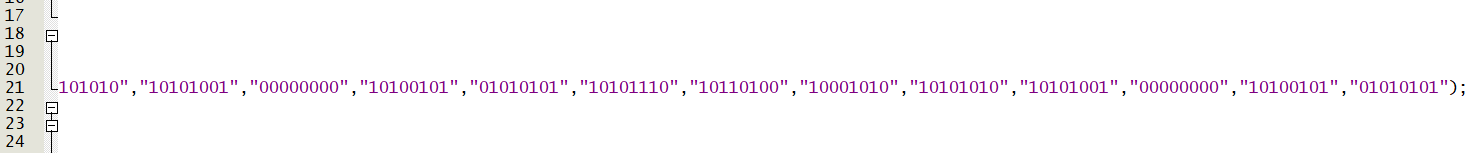
Getting all of the tasks done properly for this project was challenging to say the least. We encountered several hurdles along the way. Our first major problem was correctly setting up the finite state machine. For a while we were having trouble integrating the three instructions and their terms into the FSM. After struggling for a while we looked through the handout again and found that the solution was pretty much given to us and we were just overlooking it. We went through and carefully reviewed the provided logic and were able to find the states and sequences for each instruction. Though the instructions for store were not listed we were able to figure it out by looking at the diagrams. We broke it down into loading the MDRO, writing to memory, then incrementing the program counter. Our other big problem showed up as we were programming the control unit. We spent a long while running into compilation errors. We had lots of syntax errors in our decision statements related to the decoding instructions. The error messages kept sending us on a wild goose chase as well sending us several lines below the actual error locations, which just made the whole situation worse. We ended up spending a lot of time just going through the code for it line by line, and with a lot of trial and error we eventually got there, and our control unit code finally compiled.

**Finished code**

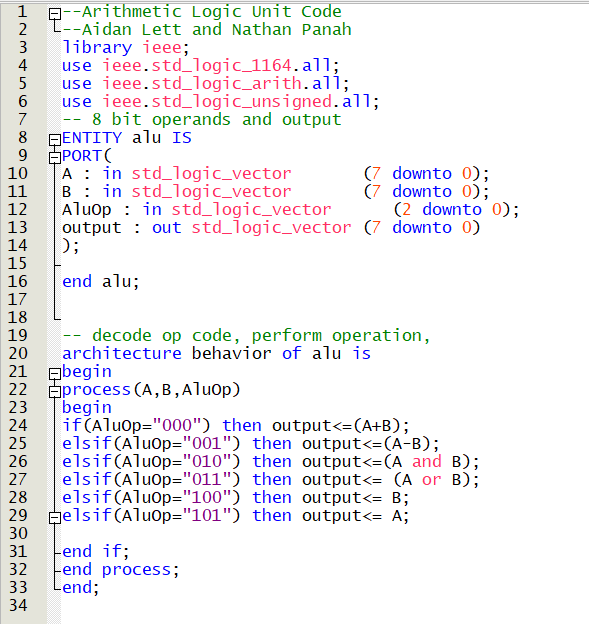
*Memory\_8\_by\_32:*



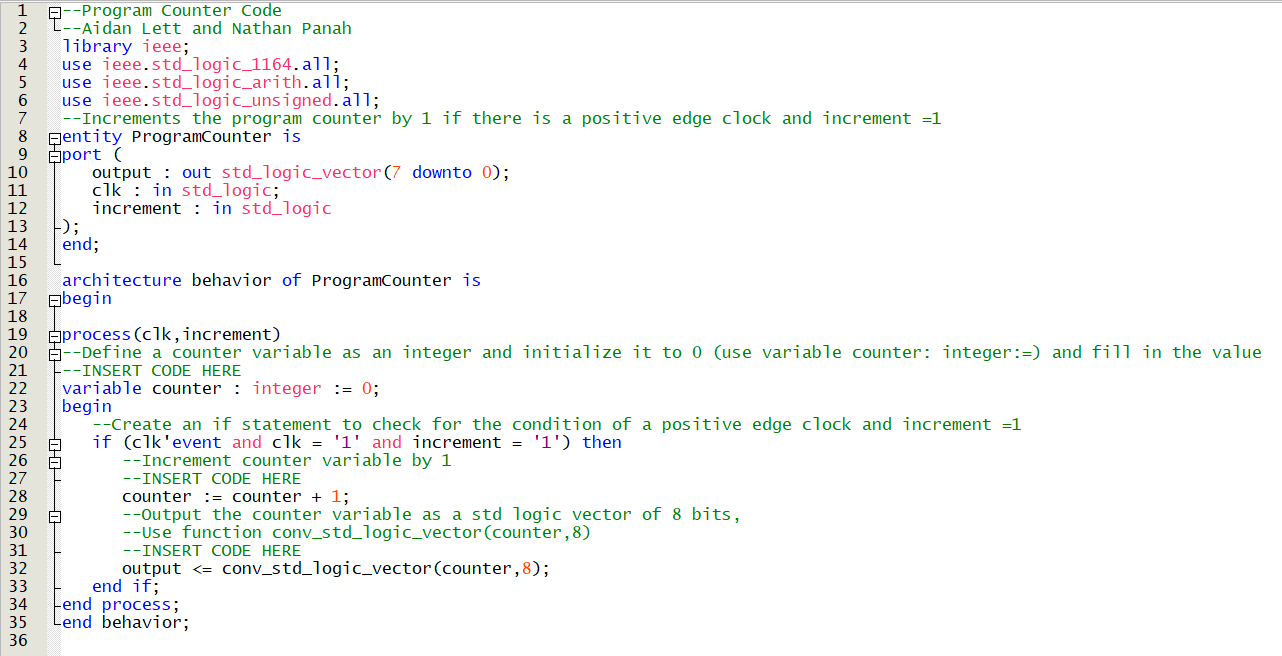


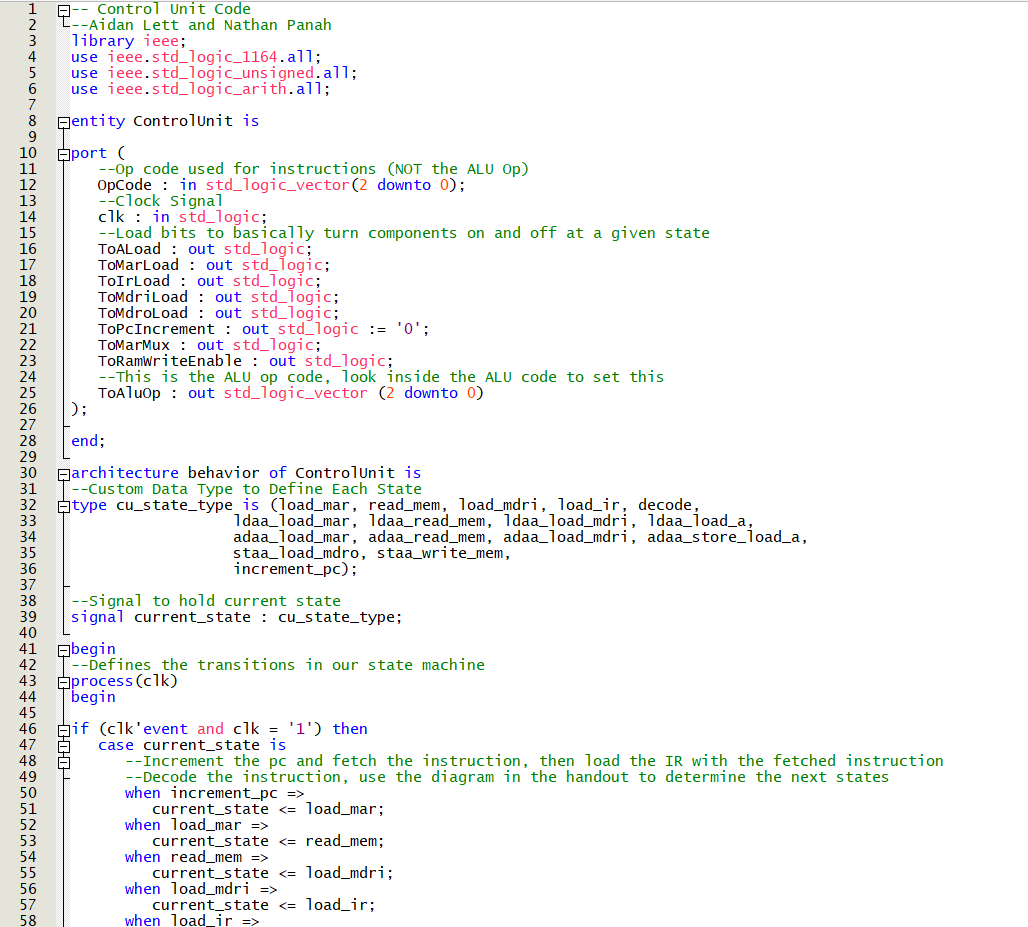


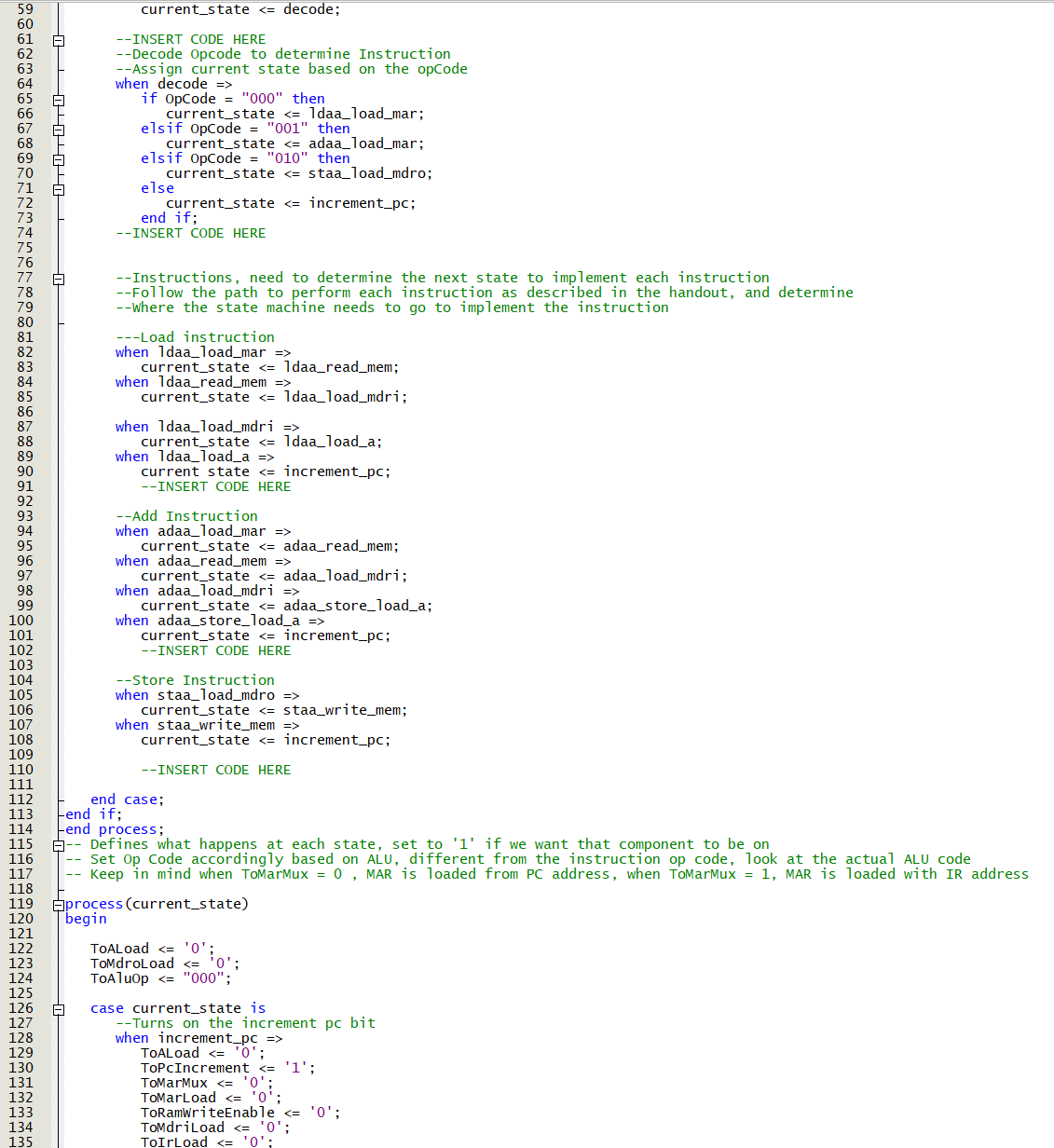
*ALU:*

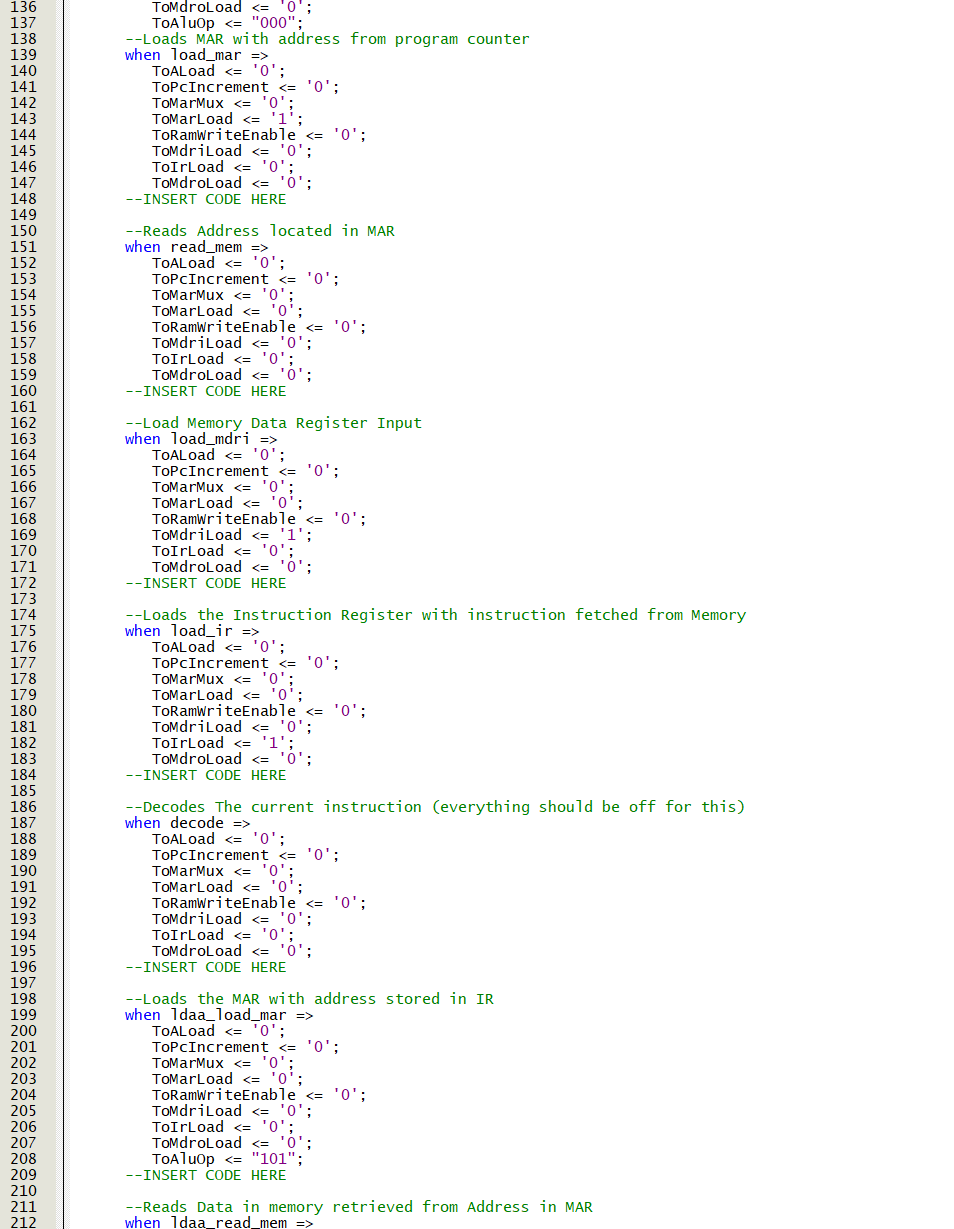


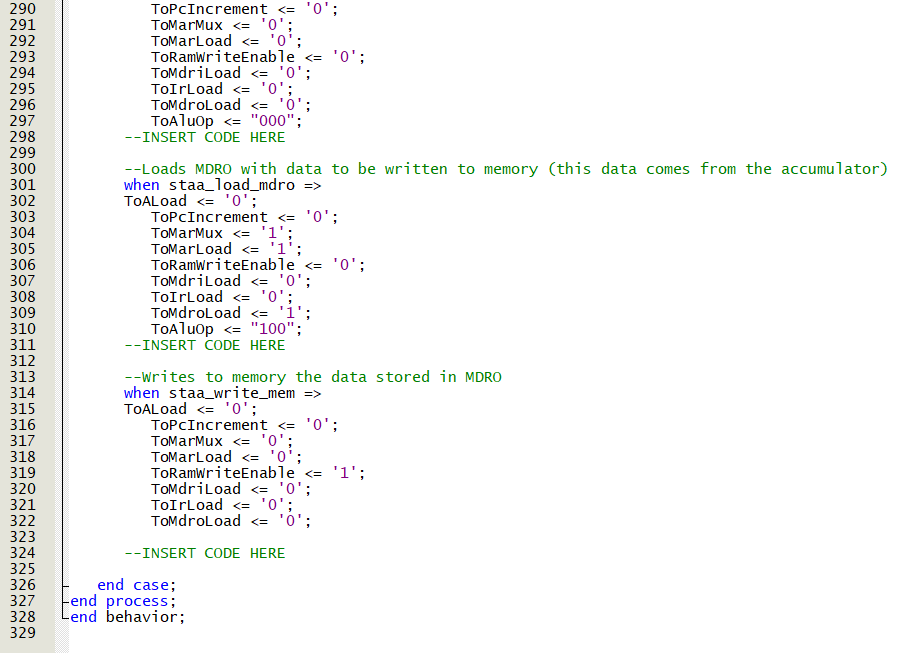
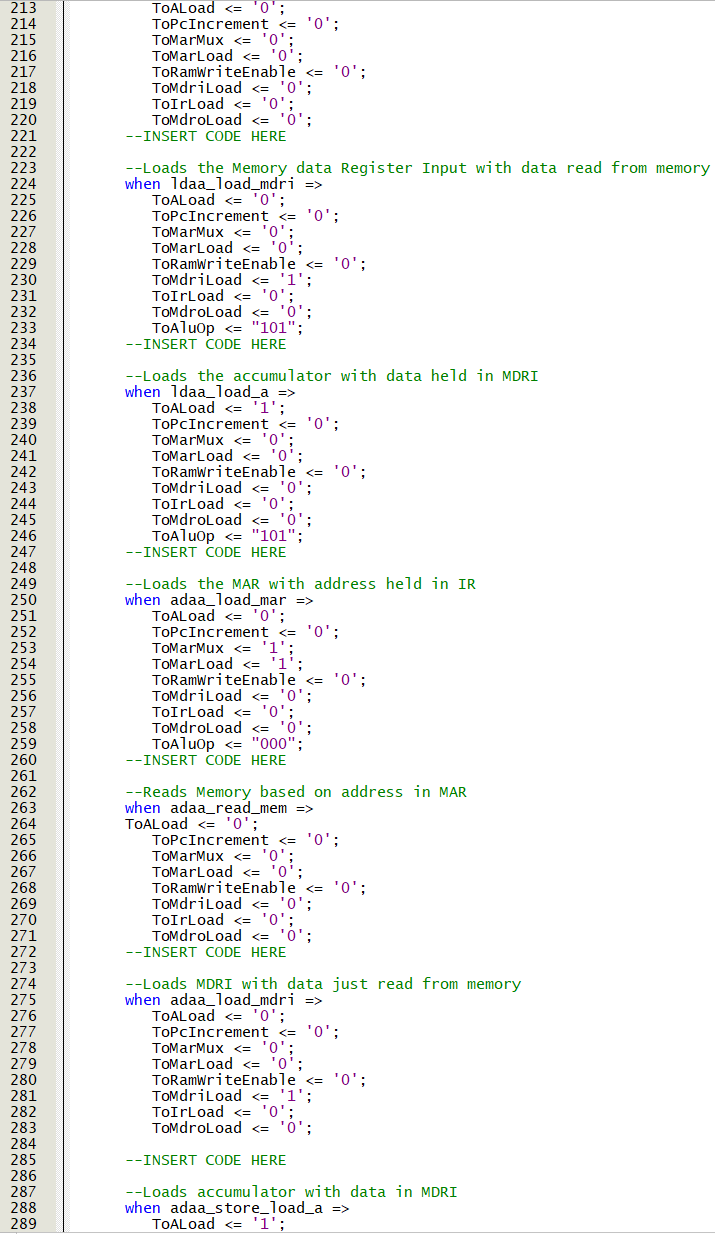
*Program counter:*

**

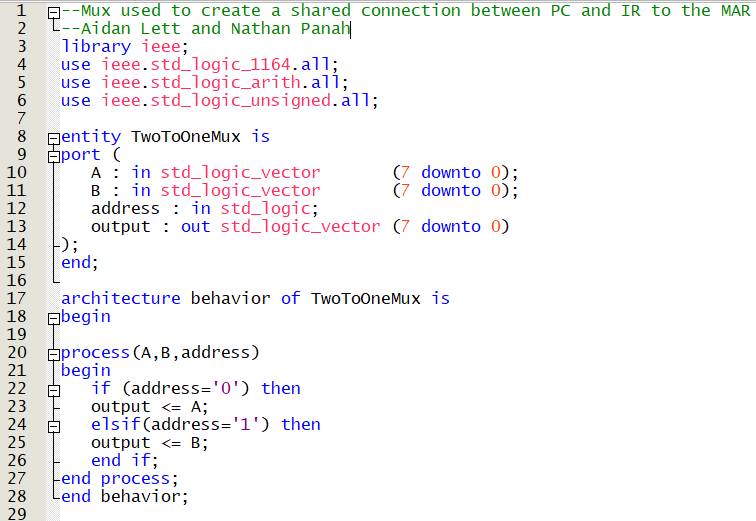
*Control Unit:* 



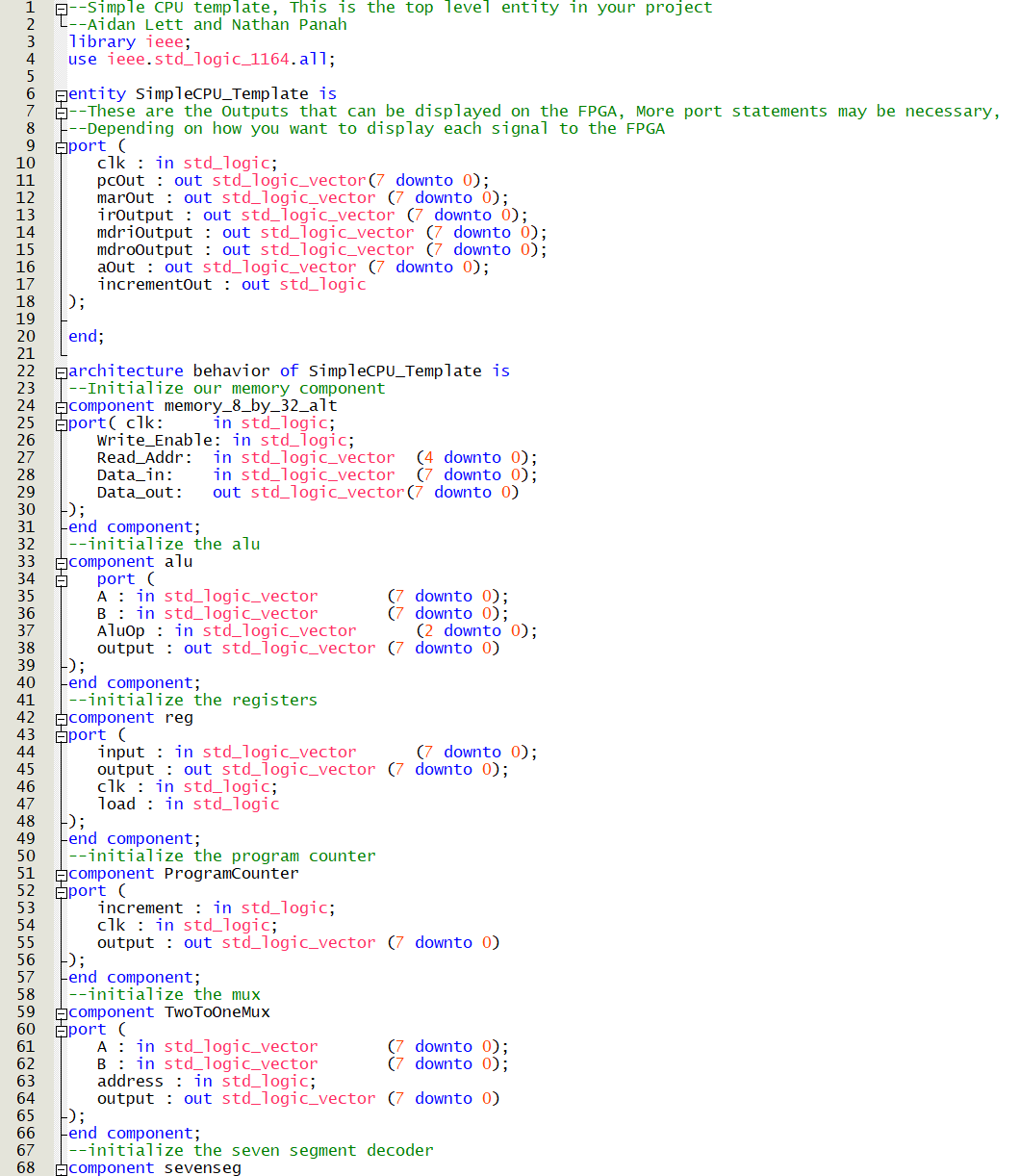


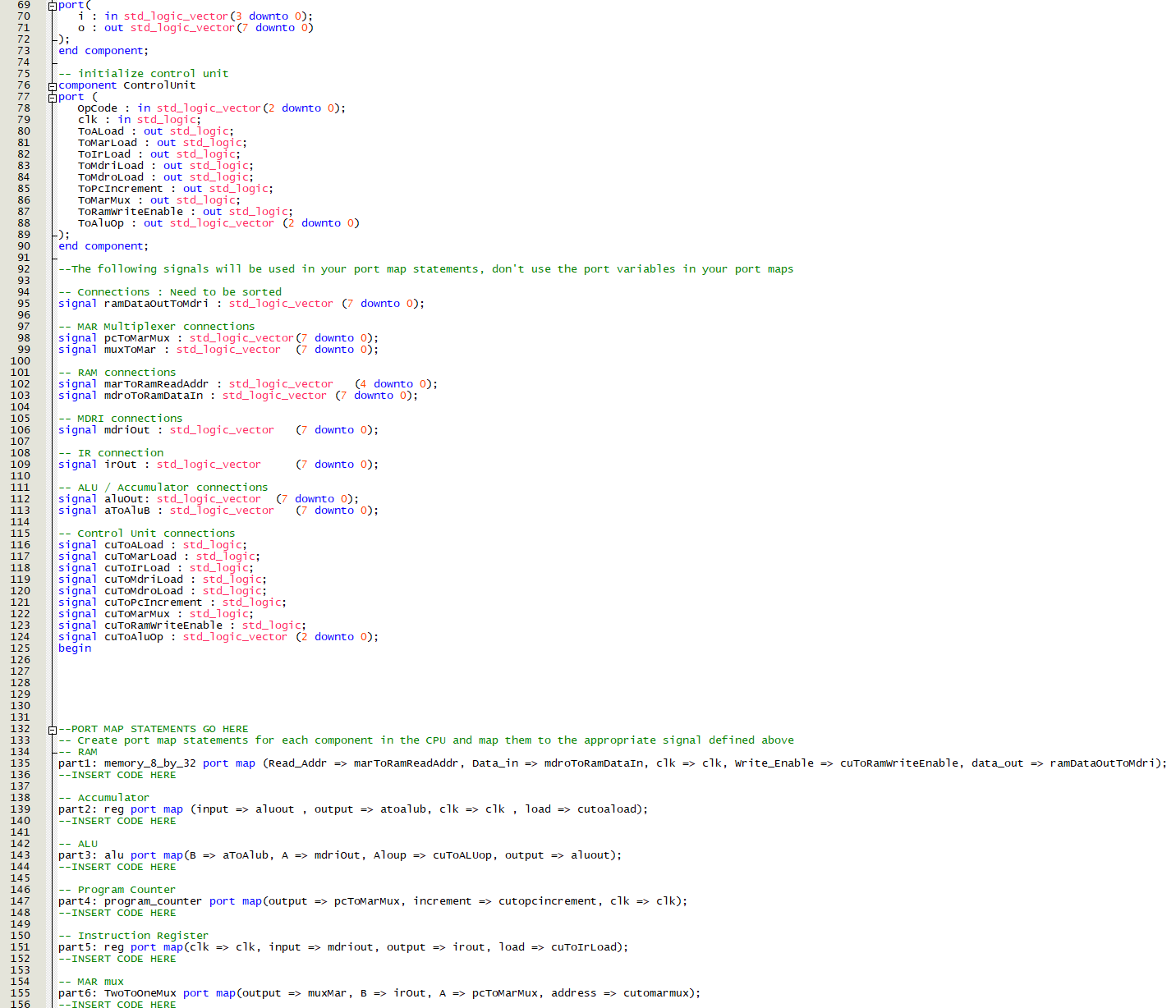


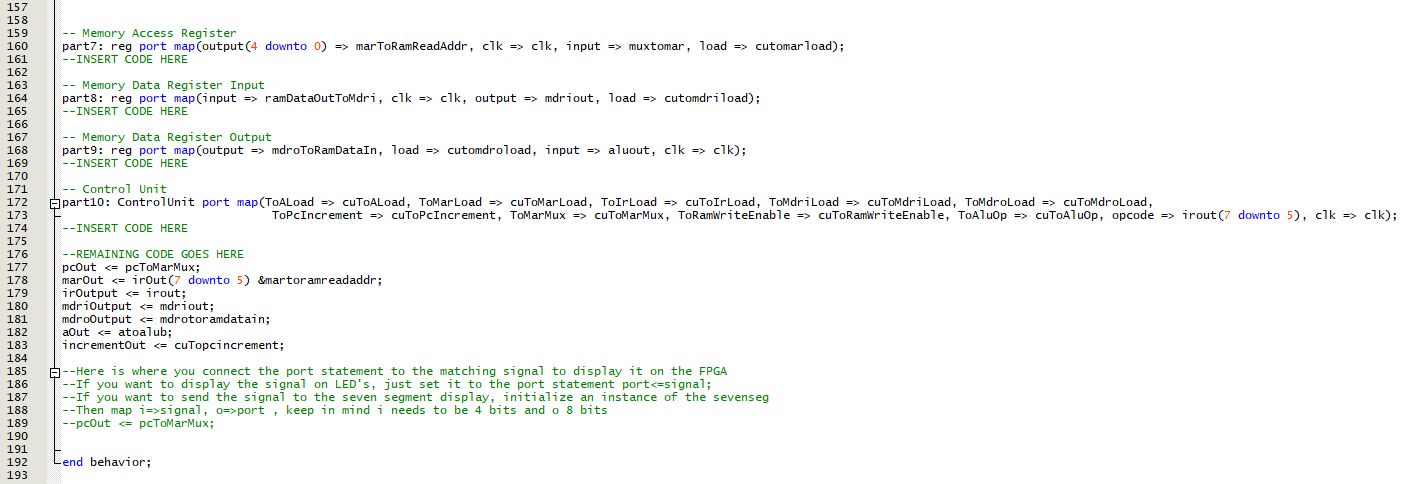
*TwoToOneMux:*



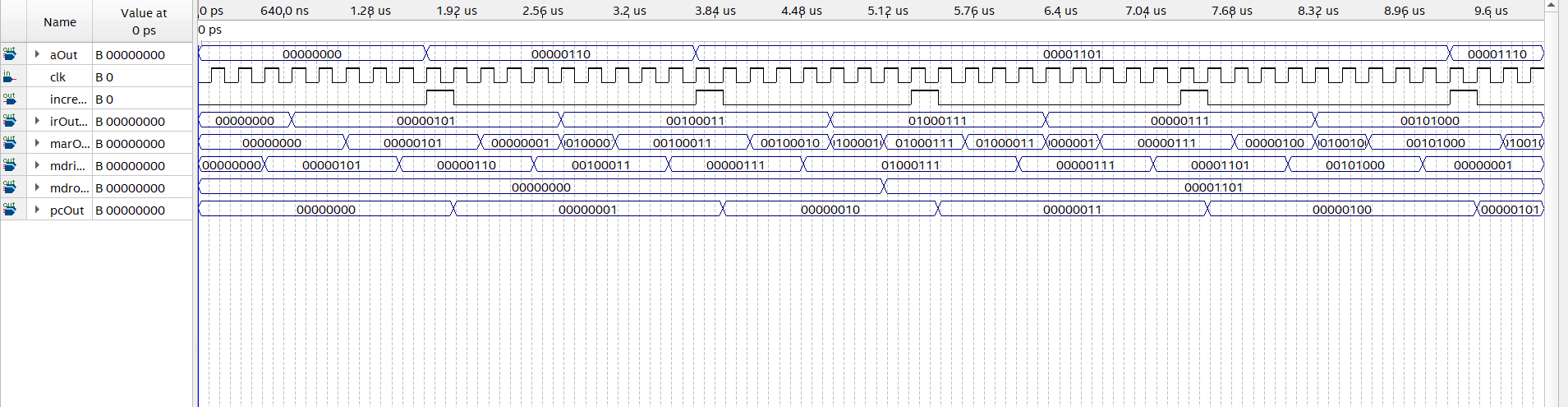
*Simple CPU Template:*





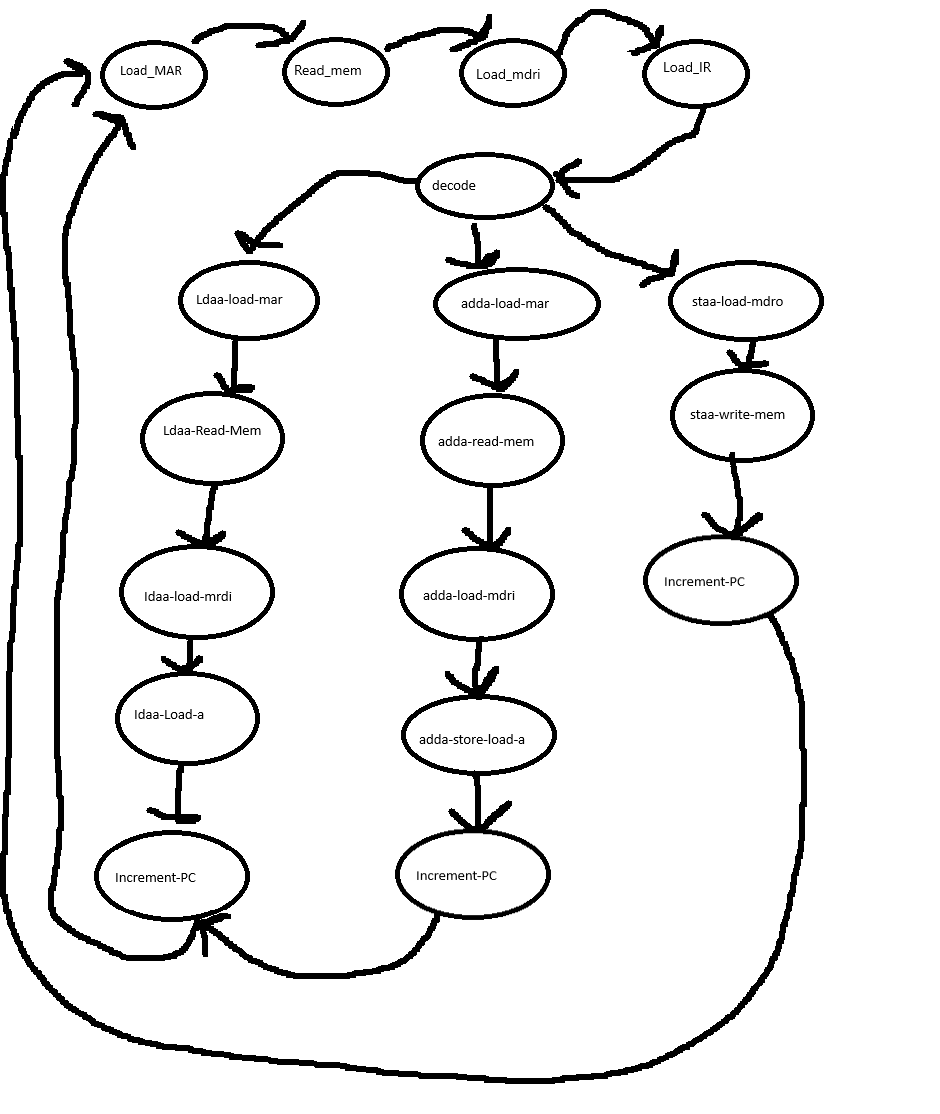


*Waveform Simulation:*

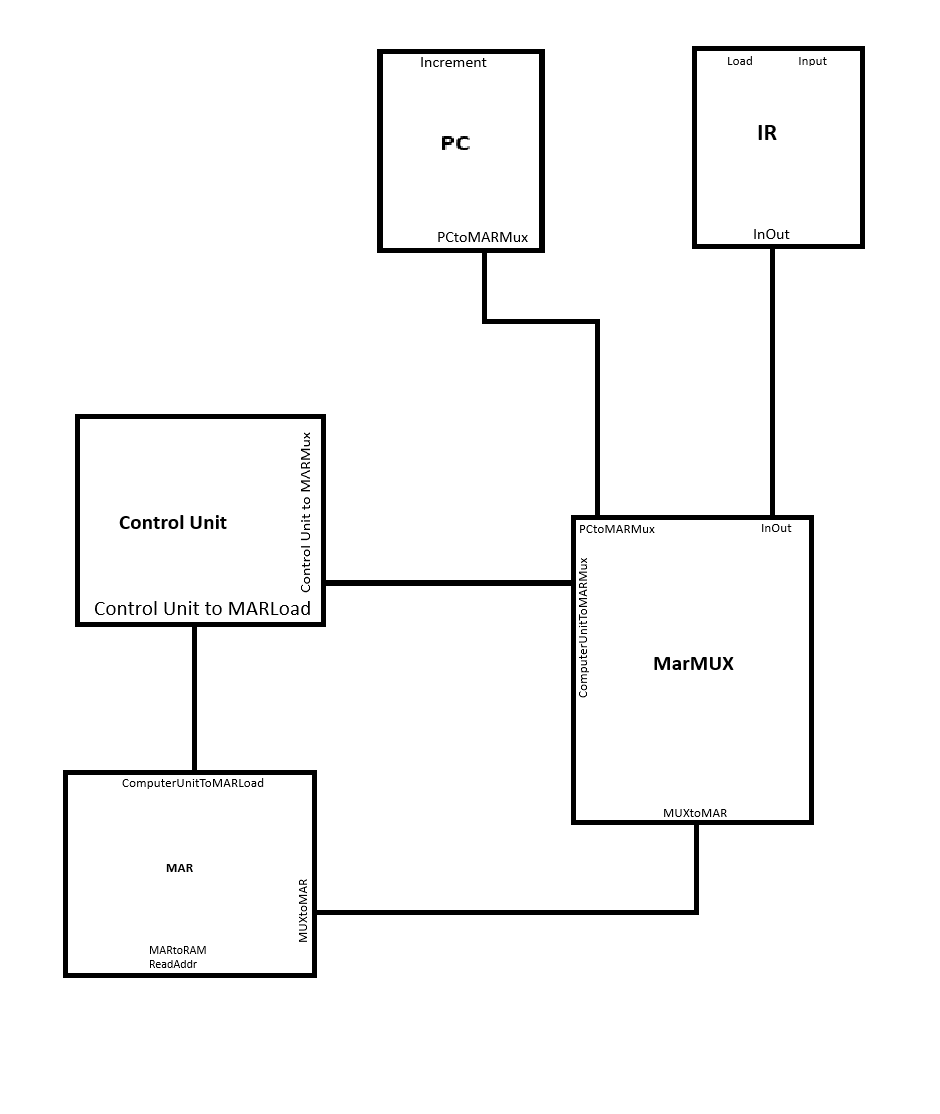


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Memory Data** | | | | | |
| **RAM** | **Binary** | **OpCode** | **Address** | **Value** | **Accumulator** |
| 0 | 00000101 | 000 – loadA | 00101 – 5 | 6 | 6 |
| 1 | 00100011 | 001 – addA | 00011 – 3 | 7 | 13 |
| 2 | 01000111 | 010 – storeA | 00111 – 7 | 13 | -- |
| 3 | 00000111 | 000 – loadA | 00111 – 7 | 13 | 13 |
| 4 | 00101000 | 001 – addA | 01000 – 8 | 1 | 14 |
| 5 | 00000110 | 000-loadA | 00110-6 | 7 | 7 |
| 6 | 00010100 | 000-loadA | 10100-20 | 21 | 21 |
| 7 | 00001101 | 000-loadA | 01101-13 | 14 | 12 |
| 8 | 00000001 | 000-loadA | 00001-1 | 1 | 1 |

**Finite State Machine Diagram**



**Block Diagram**



**Conclusion**

This project gave us valuable insight into the intricacies of a CPU and emphasized the importance of the many different components that contribute to the functionality of the CPU. This final project served as a comprehensive demonstration of how each component has a distinct purpose, but are separate independent entities. We extensively utilized VHDL files and followed the handout provided in order to successfully complete this project. We used various different tools in VHDL like port maps to illustrate the CPUs overall structure. Throughout the project, we encountered several challenges in working with VHDL and implementing various aspects but we eventually managed to successfully deal with them. The missing lines of code from the skeleton code provided was filled in with the assistance of comments in the code, which made the process much easier. Creating the finite state machine required a careful examination of the slides and handouts to piece together an accurate design. The block diagram construction was inspired by the handout that was provided to us. We focused on correctly linking components and their associated inputs/outputs. Overall, this final project really showed the complexity and unity of the components that form a CPU. It highlights the vital role that a CPU plays in executing tasks. This hands-on experience brought us a deeper understanding of how individual components can come together to create such a powerful component.