EC 413 Fall 2019

Project Report

Team #9

Members:

- Andreas Francisco
- Layan Bahaidarah

Overview:

The project's objective was to utilize the knowledge we have attained throughout the semester in our labs and lectures, and make a CPU implementation. Using our knowledge of Verilog, consisting of sequential and combinational logic, our group was able to complete the given project. After debugging, additions and changes, we were successful in creating an implementation that fetches, decodes, executes, accesses memory and write's back an instruction as needed.

Design Choices:

Our implementation mostly depends on combinational logic, with the regfile, fetch and ram modules as the exception due to their dependency on the clock. The decision to use combinational logic is to make the code of each module more concise and to avoid using any unnecessary logic operators.

Key Modules:

Our project contains 6 modules, Fetch, Decode, ALU, Regfile, Ram and Top module. The first module the processor will use is the fetch. It will provide the place where the next instruction is after every positive edge clock cycle since we only want to go to the next instruction after we are sure we have completed the current one. It does so by setting the program counter (PC) to either PC + 4 or target PC.

The second module of our processor is the RAM module. It will fetch the instruction and any other required data from memory using combinational logic. It does so by indexing the addresses specified by the 13 most significant bits of d_address and i_address to make sure we are not jumping to the middle of another instruction. Additionally, the RAM module will write to d_address's 13 most significant bits at every positive edge of the clock if wEn = 1'b1.

The Decode module will interpret the instruction fetched from RAM. Specifically, it will output the necessary results that register file, ALU, RAM, muxes and Top module will need later on and will compute the future address of the next instruction.

The regfile will read the data from the registers specified by the instruction. Also, it will write back to any registers in the end of the cycle if required by the instruction.

The ALU will then perform any arithmetic operation necessary on the outputs of the register file, immediate or program counter. Additionally, any writebacks will be performed after, to see if the instruction requires any additional information to be sent back to earlier stages before fetch moves the program counter again.

Finally, it is important to point out that each module acts on its own. The actions that were described above are all performed due to the Top module connecting the individual modules to each other. Therefore, each stage can interact with another if needed and the instruction can be executed properly.

Teaming:

The tasks were mostly equally distributed. With Layan focusing on part 1 of the project and Andreas focusing on part 2 of the project. After everything was finished Andreas focused on debugging the project whereas Layan wrote the project report.

Concluding Remarks:

We tested our project against all of the given vmh files and we obtained the correct output. Thus, we are very proud of our implementation. However, because of the time constraints we weren’t able to implement additional optimizations to our project. If we had more time we would really have liked to implement an out of order processor since we looked at various different ways of optimizing the running time of programs but didn’t really get the chance to implement them in Verilog.