Computer Organization

Lab 5

Zhengyu Li

Zlr25

Caleb Lee

Calzy8

Nicholas Bouckaert

Nabyq6

Objective:

The objective of this lab was to extend the pipelined MIPS Simulator that we developed in Lab 4. For this portion of the simulator we want it to be able to handle branch and jump instructions and execute the program appropriately. To do this we implemented both branch taken and not taken cases.

Distribution:

For this portion of the lab most of the coding was done by Zhengyu Li with the

assistance and communication of Caleb and Nicholas. We all would talk about the proper way we feel the lab should be implementing these next steps. The report then fell upon Nicholas and Caleb to write up, for the most part. As Nicholas will be coding the next lab.

Milestones:

First major milestone was understanding exactly what the goal of the lab was to do. As the whole idea was clear at first, then we discussed different implementation methods. As one person would understand how to code it in one way while the other would see it a different way of coding. So as a group we had to come to a consensus on the approach. Other than that after coming to a consensus the implementation was rather straight forward.

The second milestone was revising lab 4 due to the implementation not working with certain methods. Certain instructions with double data dependencies wouldn’t work correctly due to registers not updating all the way in the WB stage.

The last milestone was making sure each of the instructions operated as properly and branch and jump instructions went to the proper address.

Implementation:

For the new implementation of the previous lab, used different flags that acted as control signals for the IF, ID, EX, and MEM stage. In the ID stage, we don’t update the register ID\_EX.IR to IF\_ID.IR until we know the stall has finished, and if there is a stall, then ID\_EX.IR is set to 0. In the EX and MEM stages, the respective registers are updated i.e. EX\_MEM.IR is forwarded the ID\_EX.IR register and MEM\_WB.IR is forwarded the value from EX\_MEM.IR. In the EX and MEM stages, the destination registers are logged in order for ID to properly check the source registers, and if either EX\_MEM or MEM\_WB registers are 0, then the stage returns to avoid any redundant code.

For the branching, each branch calculates the proper result, and if there is a branch, then a branch flag is set. If the branch flag is set, then in the ID stage, ID\_EX.IR is set to 0, the branch flag is set to 0, and the function returns like in the EX and MEM stages in order to avoid redundant code. For getting the new address with branching and jumps , the CURRENT\_STATE of the PC is updated, and due to this the IF stage properly grabs to correct instruction. If no branch is done, then the code continues on as normal.

Conclusion:

In this lab we learned how to handle branch and jump instructions in MIPS instruction simulator with IF, ID, EX, MEM, and WB stages. This was done through a branch flag that represents there is a stall in ID stage of the next instruction, and generate the new instruction address in EX stage. We also learned how to handle different outcome of branch instructions.



