Computer Organization

Lab 3

Zhengyu Li

Zlr25

Caleb Lee

Calzy8

Nicholas Bouckaert

Nabyq6

Objective:

The objective of this lab was to used the previously implemented MIPS simulator and extend it to use a pipelined version. The five stages of this are IF which is instruction fetch, ID which is instruction decode, EX which is execution, MEM which is memory access, and WB which is writeback.In IF the instruction is fetched from memory and loaded into the instruction register IR using the program counter. The pc then is increased by 4 to select the next instruction until a syscall is reached. The ID stage decodes the instruction by pulling out the source, target, and destination registers from the instruction as well as the offset or immediate if there exists one. The EX stage just executes the instruction based on the constraints and instruction given. The MEM stage is used with load/store instructions and reads or stores in memory accordingly. The WB stage the output from the ALU is written back to the register.

Distribution:

This lab Nicholas Bouckaert was the main programmer for the lab program. Whenever he needed assistance, Caleb and Zhengyu helped with various parts of all five stages. With the help of the lab rubric, the lab went pretty smooth and all members were able to understand the basic flow of the program. The lab report was equally distributed among the members as well.

Milestones:

The first two instructions IF and ID were pretty straightforward with the way the lab document explained it and weren’t too complicated. EX would be what we consider to be our first milestone, in the since that it was similar to the lab 1 code except we had to figure out where to use ID\_EX.A and ID\_EX.B for each of the instructions and at the start took some time but once we had a rhythm going it start to speed up. Essentially it was replacing each rd, rt, and rs with the ID\_EX.A and ID\_EX.B and the imm and offsets with ID\_EX.imm. MEM and WB were together a milestone in that they both weren’t complicated alone but put together we would consider them a milestone. MEM was pretty much the same as the EX in terms of figuring out with values to replace instead of rs, and the immediate and offset values. Lastly, WB was pretty simple in that we had to determine if it was a register-register or register-immediate instructions or store or load function and connection the output from MEM to the proper register.

Implementation:

For the implementation portion of this lab we used the outline structure provided to us in the lab document with four pipeline registers and the five functions. We then started coding implementation of each step one at a time starting with Instruction Fetch Function(IF). This was simply executed by using variables already created in the MIPS simulator and storing the into the proper pipeline register. This function will also update the program counter accordingly. Next, we formated each of the subsequent functions, we mainly used switch statements to check pieces of the instruction to determine which instruction was being called. Once completed we moved onto the Instruction decode(ID) stage, we simply needed to check and see what type of function was getting called, was it an immediate, register to register or memory. After completing that portion of the code we moved onto the Execution stage, this implementation took a little long as we need to have a function for each and every function and in memory we needed to check for each of the different access instructions. Within the filled out switch statement after decoding the instruction we stored the value of registers within A and B or immediate to the correct pipeline register. In this stage we mostly used code from a previous lab( this is denoted with comments within the code), using the logic form those execution calls, we changed which variables were used to place outputs into ALU output and inputs as the A,B, and immediate. Within the memory access stage, we need to load a value using the location held in ALU Output for load instructions and store that value in LMD. For stores we used the ALU Output location and stored the value of B in that location. Finally we reach the last and final stage which is the writeback stage where we could store either the ALU Output or LMD according to which instructions was called in the correct register by decoding the instruction one final time. Along with the execution described here we need to advance the value held within each IR of the pipeline registers each time a stage was entered and make sure that the variables held within the pipeline registers would be advanced as need be.

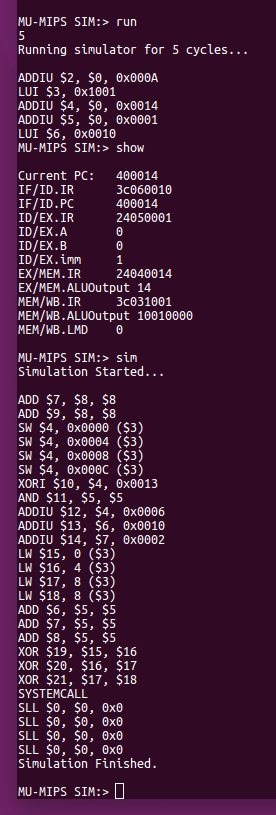


Figure 1: All the instructions running and the pipeline registers status after simulating for only 5 cycles.

Conclusion:

Overall this lab was pretty straightforward after you understood the idea of the pipeline stages. Implementing the pipeline stages was extremely helpful in solidifying the idea of a pipeline system. There were five main stages that we need to implement, those being instruction fetch, instruction decode, execution, memory access and writeback stage. Each of theses stages broke was a breakdown of a single stage we had already implemented in the MIPS Simulator before. This made understanding how to code each portion fairly straightforward while also seeing how this would speed up the pipeline. One thing i did notice about the MIPS.h file was some of the register names where incorrected/flipped when compared to the lab document. This how did not requires much trouble shooting and was a very simple fix about observing the compiler errors we received. Overall this lab should us where the value is in implementing a pipelined systemed.