Lab5 - Cover Page

Computer Architecture

CPE 315 - Section 05

Lab Section 06

4-30-16

Chad Benson

Nghia Nguyen

CHad Benson

Nghia Nguyen

Introduction ( & purpose of project ):

Lab 5 continues where Lab 4 left off: where as Lab 4 decoded mips instructions, Lab 5 implements a mips simulator and actually processes and computes each instruction, storing the results in registers. The lab helps the student gain a better understanding of how a computer works at the assembly level, keeping track of the clock cycles that occur during each instruction execution along with updating the program counter.

Functional Requirements ( What is to be accomplished ):

To implement the mips simulator, we first needed to load a mips binary file into a buffer. Since this was required in the last lab, we just reused the same code from before. Next, we executed one instruction at a time, keeping track of what happened to the clock cycle, program counter, and register values based on the instruction. Additionally, besides printing out all the results to the console, we setup of the program such that the user can choose whether to run through the entire program, step through the program ( line by line ), or quit the program.

Approach used ( algorithms & methods ):

We used a do-while loop to run the general state of the program, prompting the user for a command and then processing each instruction. If the user enters 0, then the program quits. If the user enters 1, then the whole program runs its course and then halts when a syscall of 10 in $v0 is executed and then all the register values are printed to console, including the number of instructions, number of cycles, etc. If the user enters 2, then the program simply steps through each instruction, printing the results one at a time.

We reused the code from lab 4, which decoded each instruction. Keeping that same structure, we added coded to compute each instruction. For example, if the instruction was an “add” instruction, we first loaded the rs and rt values from their respective registers, added them together, and then stored the result in the destination register. We used a structure, mips\_sim, to keep track of the register values, program counter, total number of cycles, and so on. All of the mips instructions on the first page of the green sheet were implemented.

Source Code:

See zip file.

Discussion ( difficulties and or concerns with reliability or security ) :

It took us some time to set up the general structure of the program. We also were not sure about reading the instructions as bytes, as discussed in class… we instead used an unsigned int, which seemed to work.

Summary:

Lab 5 helped us gain a deeper understanding of how a computer is implemented at the assembly level. We also learned how to implement most, if not all mips bare metal instructions in the C language. This lab, in conjunction with the last lab, has help us gain a better general picture of what things look like “under the hood.”