Computer Organization

Lab 1

Zhengyu Li

Zlr25

Caleb Lee

Calzy8

Nicholas Bouckaert

Nabyq6

Objective:

The goal of the lab was to create two different functions that will handle instructions being sent to them. The first one we decide to complete was the print function, that handled the printing of an instruction that was sent and what instructions were used to complete it. The second function we code up was handle instruction, which is going to execute the instruction that is being sent. This instruction is that call every cycle for a new instruction set.

Distribution:

The workload was distributed evenly among the group, one would work on code, lab report, and translating instructions for the coder. Each of us would look and understand what the other did before moving on.

Milestones:

The first big milestone we had was figuring out the best way to do the print instruction. We were given the instruction to do a mem\_read\_32() on the address given in the function, but we didn’t quite know how to implement it but figured it out using the manual on Canvas. The second milestone was completing the handle\_instruction() function. This took a long time and we had to use the manual for each individual opcode to understand how to effectively program and simulate the MIPS instruction. We went by instruction by instruction reading how to properly run it.

Implementation:

For our print\_instruction() function, we used first used the mem\_read\_32() function and used addr which was a argument in the print function to read in the hexadecimal instruction format. Then, since the instruction length was 32 bits, we parsed through the result of the mem\_read\_32() function to store each value of the general purpose registers, immediate values, and the opcodes. After that, we used a giant switch statement to properly handle each case according the the opcode and the ending bit values which further identified the MIPS instruction. In each case we used an sprintf() to form a string that was the same format as usual MIPS instructions and at the very end of the function printed the string out.

The handle instruction was very similar but the few differences it had were that we used the mem\_read\_32() function but instead of an address given as a parameter we used the CURRENT\_STATE.PC defined in the header to read in the instructions. We then needed to increase the NEXT\_STATE.PC to CURRENT\_STATE.PC + 4 due to the instructions being 4 bytes long. After that we used the same concept as in the print\_instruction() being that it was a large switch statement that handled each case accordingly to update registers and perform jumps and branches.

Conclusion:

We have learned how to program specific instructions in c and how to print the results appropriately. We also learned three types of instructions such as I type, J type, R type and follows the document to develop c code and calculates the address of each register. We learned how to translate a 32-bit hexadecimal instruction format into a readable format using opcodes, register numbers, immediate and offset values.