Douglass Chen, Eric Gauderman, Matthew Pohlmann, Angelica Huyen Tran

1 May 2014

EE357

**Final Project Report**

**General Structure**

A long-word is 32 bits. As each of our instructions is 32 bits, we can allocate 1 long-word for each instruction.

Machine Code, Program Counter, and Start Location: We allocate 11 long-words at a location #CODE for the machine code and 1 long-word at location #PPC for the program counter. At the beginning of the program, we load the 11 instruction lines into #CODE and then load the first line of #CODE, our first machine code line, into #PPC, our program counter.

Registers: We allocate 8 long-words for the registers. Each long-word is labelled ‘Rx’ where x ranges from 0 to 7. We set #R0 to 0. It stays 0 for the entire program.

Instructions: Instructions are implemented using subroutines. To process each instruction, we first logically shift the program counter (the current line) to only contain the opcode (the first 6 bits). We then compare the opcode against the 8 instructions. If the opcode of the current instruction matches the opcode of a specific instruction, we jump to that subroutine. Once we finish the subroutine, we increment the program counter (unless we are branching or loading, in which we load a specific memory location into the program counter under specific conditions).

To parse the instruction to obtain the necessary information (the registers and the immediate value if applicable) we load the instruction into a separate register and perform shifts to access the necessary bits. After ANDing this value with the correct number of 1’s (to clear any unnecessary bits) and ensuring sign extension on immediate values, we store these values separately into different registers for easy access.

We have several subroutines to help our code: GET\_REG\_D2, SAVE\_D2\_TO\_REG\_D3, and WAIT. When we process an instruction, we are only given a register number, which we then must use to access the current register address. GET\_REG\_D2 takes this register number, retrieves the correct register address, and moves the value at that address into D2. In SAVE\_D2\_TO\_REG\_D3, D2 contains a value and D3 contains a register number. This subroutine saves the value in D2 into the correct register obtained from the register number in D3. The logic is the same to GET\_REG\_D2. WAIT processes a delay loop to provide an interval between commands.

**Difficulties**

Our largest difficulty was designing and setting up the infrastructure for our emulator. Once we completed our design, implementing the instructions, the program counter, and the machine code was mostly straightforward. Our other major obstacle was debugging why our branching did not work. When we retrieved the immediate value, we would clear the first 12 bits of the instruction to 0’s. However if the immediate value was negative, we forgot to extend the sign and would add the incorrect value to the program counter. We fixed this by performing an arithmetic shift. This sign extension was added to all of our instructions that involve immediate values, including ADDI, SUBI, and LOAD.

**Team Contributions**

We divided the work evenly between each team members. For a more specific breakdown, see below.

Douglass Chen:

Eric Gauderman:

Matthew Pohlmann:

Angelica Huyen Tran: