Computer Architecture: Lab 1

Student 1: Steven Howell **CWID:** A20236687

Student 2: Seth **CWID:** A20271944

• Section 1: Introduction – In this lab, we were introduced to the implementation of RISC-V and the many functions that they offer. We were introduced to many instructions like AND, XOR, ADD, and many more. The software used was Cygwin along with a C compiler and GNU. Most of our compilation happened in the ISA code, however, we also had the SIM and Shell files as well.

• Section 2: Baseline Design - The baseline design that we were provided was 3 ISA instructions and the simple simulator file. Continuing to build the lab involved implementing most of the RISC-V 32-bit architecture instruction set as well as building a corresponding simulator to handle the instructions. Inside the simulator, we added code to support multiple new opcodes for the instructions as well as the funct3 and funct7 and I'm determining the instructions.

• Section 3: Detailed Design - Our detailed design implemented a total of 38 instructions ADDI, XORI, ORI, ANDI, SLTI, SLLI, SLTIU, LB, LH, LW, LBU, LHU, SRLI, SRAI, LUI, AUIPC, SB, SH, SW, ADD, SLL, SRA, SLT, SLTU, XOR, OR, AND, SUB, BNE, BEQ, BLT, BGE, BLTU, BGEU, JALR, JAL. The ANDI, XOR, and ORI instructions perform the and, or, and xor operations between the current register information and the immediate value. The ADDI instructions preforms the addition operation between the current called register and the immediate input. The SLTI or set less than immediate sets the value of the current register to less than the value of the immediate. SLLI or shift left logical immediate shifts the current register left by the value of the immediate. SLTIU performs the same operation as SLTI however the result zero extends to 32 bits. The Load Byte instructions take a byte value from the given address. The load half instructions take a half (16-bit) value from the given address. The load word loads all 32 bytes from the address. LBU performs the LB operation however it zero-extends the result to 32 bits. LHU performs the LH operation however it zero-extends the result to 32 bits. The LB, LH, LW, LHU, LBU, functions all employed memory_read_32 to read the value from the memory. SRLI or shift right logical immediate shifts the current register right by the value of the immediate. SRAI or shift right arithmetic immediate performs the SRLI operation however it msb-extends the result. LUI or load upper imm shifts the immediate by 12 bits and loads it into the next register. The auipc shifts the immediate value by 12 bits then adds it to the PC value and loads it into the next register. Store Byte takes a memory address and a value and writes the byte value into the given address. Store Half takes a memory address and a value and writes the 16-bit value into the given address. Store Word takes a memory address and a value and writes the 32-bit value into the given address. The ADD function takes the value from 2 different registers and sums them together then pushes it to the next register. The SLL function shifts the value of the first register left by the value of the second register. The SRL function shifts the value of the first register right by the value of the second register. SRA shifts register 1 by the last 5 bits of register 2 and stores it into the next register. SLT compares the current value of the register and the value of register two and sets it less than register two. SLTU does the operation of SLT but zero extends the final result. XOR, OR, and AND do the and, xor, and or logical operations between resister 1 and register 2 storing the result in the next register. The SUB function subtracts the value of register 1 and register 2 and stores them in the next register. BNE function compares the value of the registrars and if they are not equal then the immediate value is added to the PC. BEQ function compares the value of the registrars and if they are equal then the immediate value is added to the PC. BLT function compares the value of the registrars and if register 1 is less than register 2 then the immediate value is added to the PC. BGE function compares the value of the registrars and if register 1 is greater than or equal to register 2 then the immediate value is added to the PC. BLTU function compares the value of the registrars and if register 1 is less than register 2 then the immediate value is added to the PC and zero extends the stored result. BGEU function compares the value of the registrars and if register 1 is greater than or equal to register 2 then the immediate value is added to the PC then zero extends the stored results. JAL jumps and links the result. It takes the current PC value and links it to the next register and then takes the immediate and sums the current state storing it in the next PC value. JALR takes the current state and links it to the next register then takes the immediate and sums it with the current register 1 and stores it as the next register state. Finally, ECALL sends the program controls to OS.

The simulator had to have adjustments made to allow for proper control of the functions. The Funct3 is the controller that differentiates between the opcodes and sends the programs to different actions. Funct7 or equivalent had to also be implemented to allow programs with the same opcode and the same Funt3 code to be differentiated such as ADD and SUB. The simulator for several processes had to be completed and integrated with the data read such as U and J processes and some opcodes has to be added to the process controller in the simulator since it laked functionality.

• Section 4: Testing Strategy - We had a set of instructions that were programmed in and had to be tested. Professor Stine sent us a link to a website called luplab that helps us see the values of all RISC-V instructions helps us see what the encode/decode values are for each instruction. Using that, we were able to do a test harness that allows us to use the execution of each instruction. While watching the execution of each instruction set, we can clearly see how each of our instructions behaves while comparing its output with luplab.

• Section 5: Evaluation - RISC-V is a very important concept to grasp, as RISC-V and ARM are absolutely essential for Computer Architecture. This is the purpose of this lab. Therefore, the assessment of this lab is rather straightforward. We were given a set of codes from the RISC-V greencard files on Github. This was helpful to understand and write the properties of each instruction.