**Lab 10**

**EE 4305**

**Computer Architecture**

**Professor: Taek Kwon**

**Mat Maternowsky**

**Lingbo Meng**

**Objective:**

The objective of this lab was to add the final three modules: execute, control, and memory. The execute module takes the inputs from the decode module and performs the ALU operations. It takes the inputs and produces alu\_result, branch\_addr, and branch\_decision. The ALU functions are controlled by a two bit ALUOp code. The control module will take the op code portion of the instruction and generate nine control signals. The control signals are determined by whether the instruction is R-format, lw, sw, beq, addi, or jmp. We had to modify the decode module to include a write clock and a write\_data signal that carries the value written to the register. The memory module is similar to the fetch module in its implementation. It will receive an address from the alu\_result and write the decoded data to memory. For testing we used three switches to show registers rs, rt, and rd as well as immediate, alu\_result, read\_data, and write\_data. We had two buttons: one for a manual clock and one for cycling through test values. We used three separate test assembly codes, one for r-type, one for lw, sw, and one for beq loops.

**Data Path and Control Signals:**

The program counter decides which address to read from the instruction memory in the fetch module. From there each type of instruction follows its own path. The path it follows is determined by the Op code or instruction[31:26]. These are the control signals.

**Lw:**

For load word, the instruction goes into readreg1 and readreg2, and rt is used for write register since RegDst is ‘0’. The immediate gets sign extended and is sent to the ALU with the ALUSrc = ‘1’. The data from rs is also sent to the alu and the added with immediate since ALUOP = “00” and the result is stored in rt and used as memory address.MemRead = ‘1’ and MemtoReg is ‘1’ so the read data is sent to the write data in the register.

**Sw:**

For store word, the instruction goes into readreg1 and readreg2, and it doesn’t care about write register. The immediate is sign extended and sent to the ALU with the ALUSRc = ‘1’. The ALU adds rs and immediate since ALUOP = “00” and the result is stored in rt and used as the memory address. The rt information is then written to memory since MemWrite = ‘1’. RegWrite =’0’ so no value is written to the registers. **Addi:**

For add immediate, the instruction goes into readreg1 and readreg2, and rt is used for write register since RegDst is ‘0’. The immediate is sign extended and is sent to the ALU with ALUSrc = ‘1’. The data from rs is added with the immediate since ALUOP = “00” and is stored in rt and used for memory address. Nothing is written or read in memory since MemWrite = ‘0’ and MemRead = ‘0’. Instead the result or rt from the ALU is sent as WriteData for the registers since MemtoReg = ‘0’.

**R-format:**

For R-format, the instruction goes into readreg1, readreg2 and writereg, rd is used for write register since RegDst is ‘1’. Instruction[15:0] goes to sign extension and sent to ALU with ALU Src=’0’, so immediate won’t be used in R-format. ALUop=’10’ and result will depend on the value of ALU control. The result from ALU will goes to writedata and write data in register(index).

**Beq:**

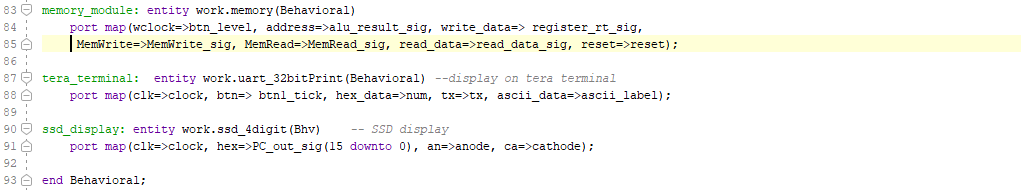
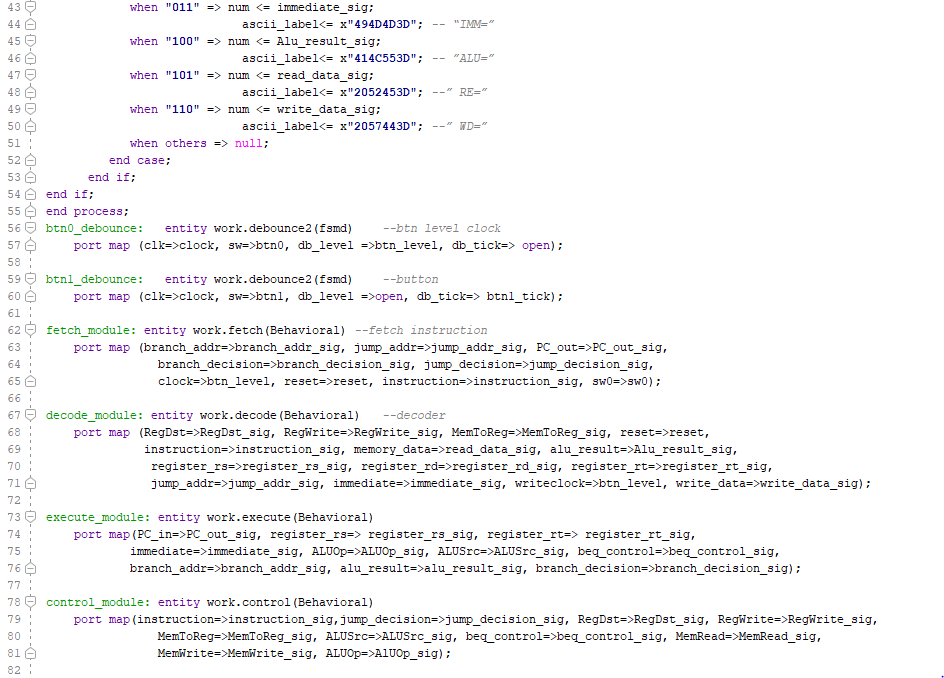
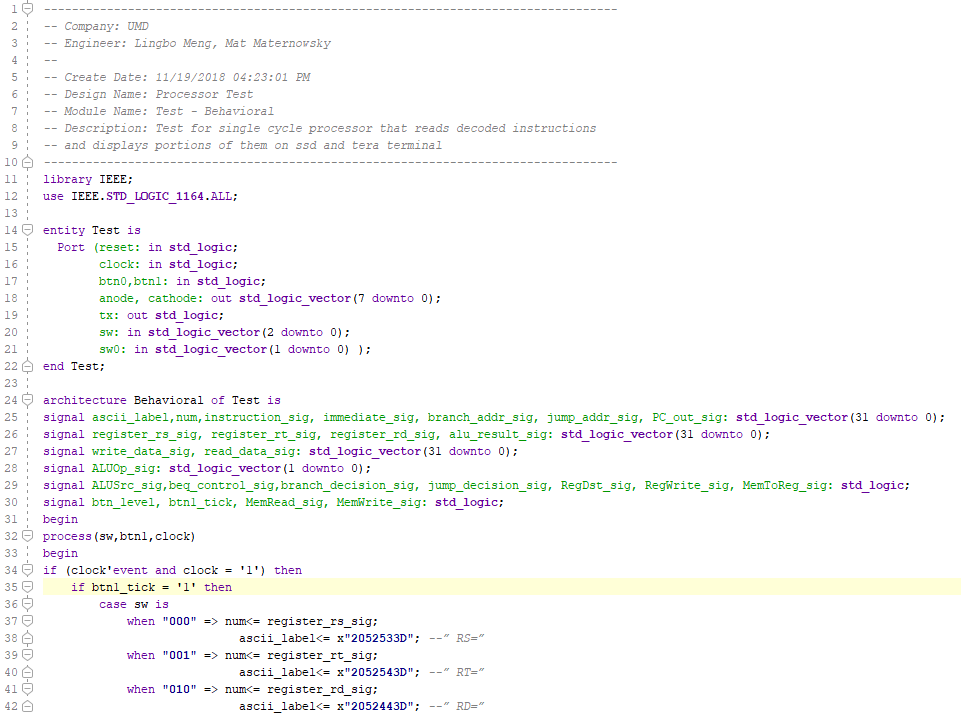
For branch, the instruction goes to readreg1 and readreg2, it doesn’t matter whether rd or rt will be the destination register when writing back. Immediate will be added to PC+1 after it is sign extented. ALU Src is ‘0’ so both readreg1(rs) and readreg2(rt) will come to ALU. ALUop=”01” so the result will be subtracting. When rs = rt and the result is zero, and beq\_control is ‘1’ for beq instruction, so the branch signal will be triggered and MUX choose the result Imm+PC+1 and use it as the next address.

**J:**

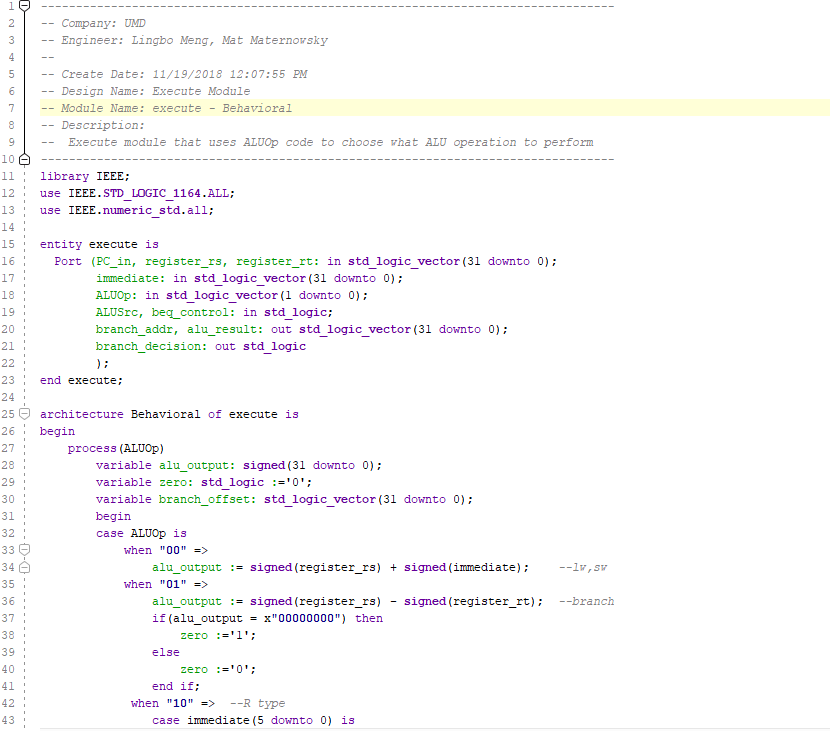
For jump, Opcode goes to control module and jump signal=’1’, then jump address will be goes to MUX and choose by jump signal. Mux return jump address as next address for PC.

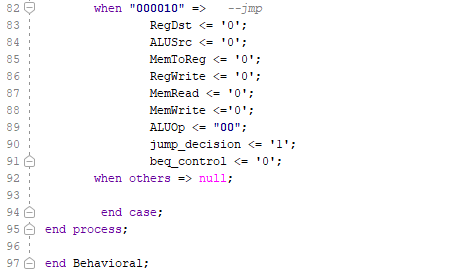
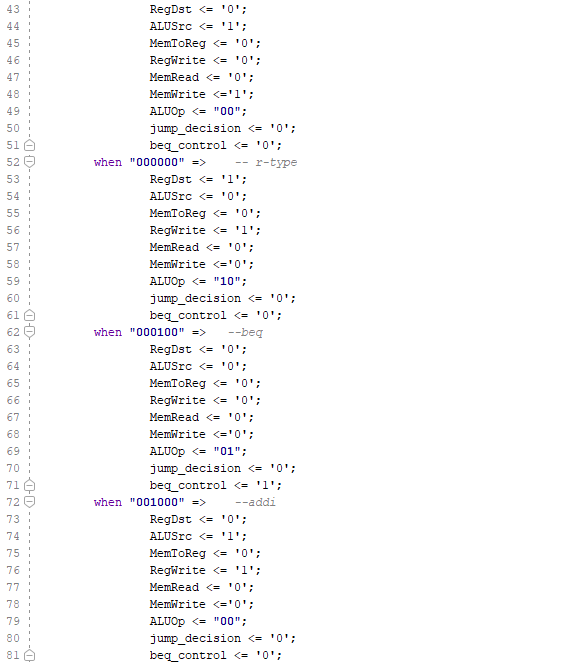
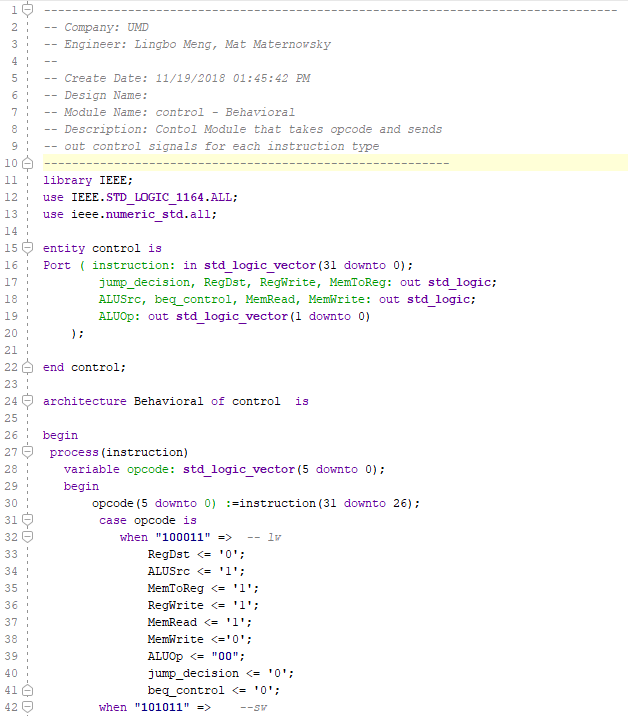
**Conclusion:**

This lab added the final portions to our MIPs processor. It helped our understanding of how the control signals work to route the instruction through the processor. We can now see the broad picture of how our processor functions. It helped to understand how the memory functions and returns the values to registers, and how the ALU operates based on control signals.

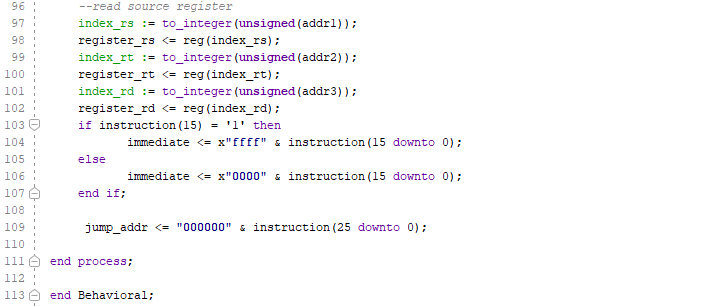
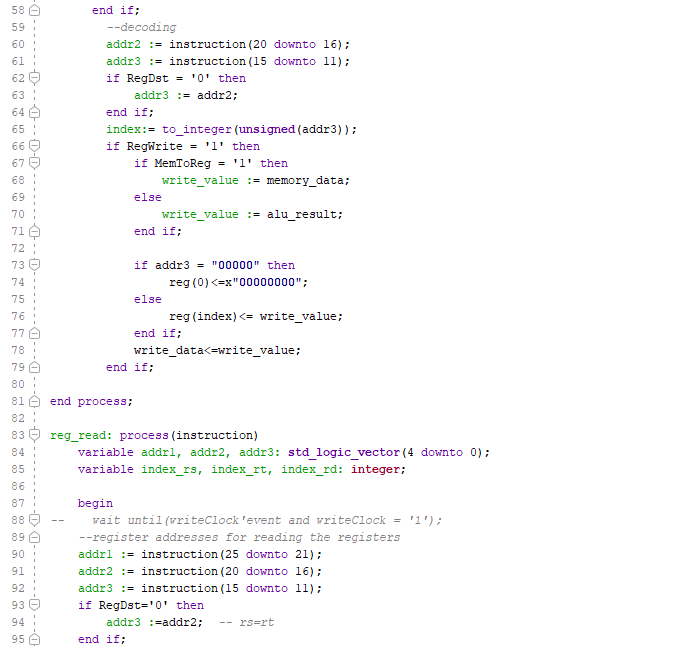
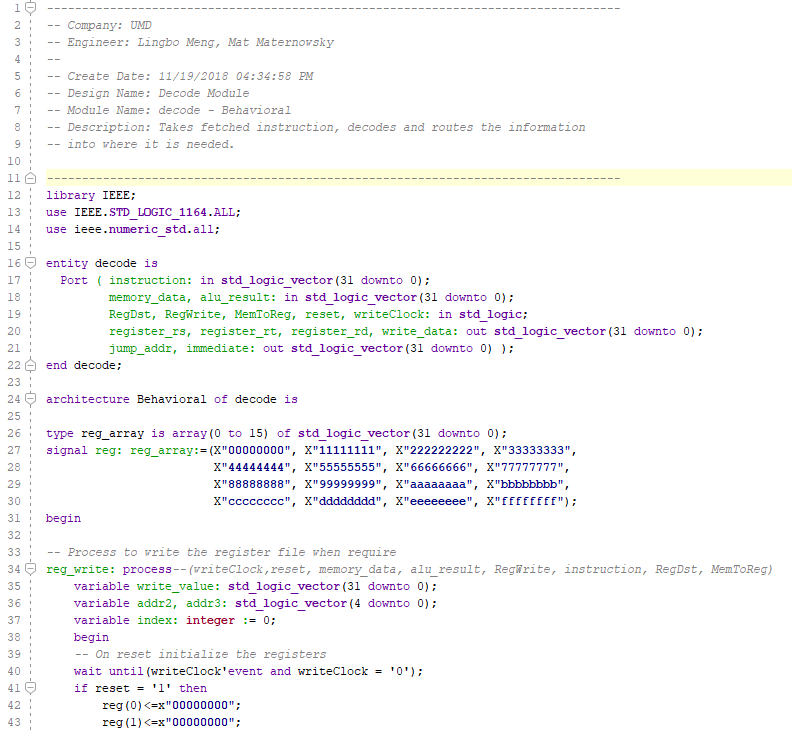
**Test Module**

**Execute Module**

****

**Control Module**

**Decode Module**

****