EC413 Class Project Part 2

Laura Morgan

The goal of this project is to design and implement a single-cycle, non-pipeline MIPS CPU. For the first part of this project the register file, the arithmetic logic unit, and the data memory were implemented. For the second, and final part of this project, the control unit and instruction fetch logic components were implemented. The control units purpose is to identify the instruction class and derive the logic signals needed to execute the instruction. The instruction fetch logic component consists of the program counter sending the current instruction address to memory to fetch the instruction.

The first thing I did when implementing the control unit was to find the ALUOp by setting it equal to instruction [31:26]. Finding the ALUOp determines which type of instruction it is so I made a few different if statements comparing the ALUOp to specific cases. For example, the ALUOp for load word is 100011 so if the ALUOp equals 100011 it would then set the logic signals for a load word instruction. The instructions that I implemented that depended on the ALUOp were load word, store word, branch and addi. I made the default case all R type instructions. Once the instruction type was identified, the logic signals were then set to correspond with that instruction type.

When I implemented the instruction fetch logic, I ended up breaking it up into two different modules that I called Instruction\_Fetch and Program\_Counter. The Instruction\_Fetch module acted like the instruction memory. I did this by making a case statement based on the value of the program counter. For example, if the program counter is equal to two it would set instruction equal to whichever instruction is under case two. The Program\_Counter module is responsible for incrementing the counter on every positive edge of the clock. For all instructions that occur, other than branch, the program counter increments by one each time. If a branch instruction occurred and the ALU\_zero is equal to one (which should occur if the instruction is a branch) then the program counter value is equal to the sign extend plus the original value of the program counter.

All of the modules that make up the single-cycle mips processor that I implemented in this project were all connected together in the CPU module and run with the CPU testbench. All other testbench’s that I included in my submission were used just to test the individual modules. In the CPU module, it first starts by sending the program counter to the Instruction\_Fetch module to get the instruction. That instruction is then sent to the Control\_Unit module to be identified and to have all of the logic signals set according to the instruction type. A mux is then implemented to choose what the write address of the register file will be (the mux chooses this depending on the RegDest logic signal set by the control unit). Next is the registerFile which stores the array of registers in the regfile. 16 bits of the instruction are then sent to the sign extend to extend it to 32 bits. If the 16 bit value is negative then 16 ones are added otherwise 16 zeros are added. Another mux is then implemented to determine if the second input going into the ALU should be the output from the sign extend or the read data 2 output from the registerFile. The output of this mux is dependent on the ALUSrc logic signal set by the control unit. The ALU module is then implemented which performs the arithmetic logic that is determined by the ALU\_Control (the ALU\_Control is instruction[3:0]). An and gate is then implemented which ands the Branch logic signal and the zero output of the ALU module. This and gate only outputs a one if both the Branch logic signal and zero output of the ALU are equal to one. Then comes the data memory module which sets the address register in ram equal to writeData if the MemWrite logic signal is one (such as in store word). If MemRead is set to one then ReadData is set to the address register in ram (such as in load word). Another mux is then implemented to determine what gets sent to Write Data in the register file. This mux depends on the logic signal MemToReg set by the control unit. If MemToReg is zero then the ALU result is sent to Write Data. If it is one then the Read Data output from Data Memory is sent to Write Data. The last module that is implemented in my CPU design is the program counter which increments the program counter by one each time and if the instruction is a branch it then it adds the sign extend value to the program counter value. This new program counter value is then sent back to the instruction memory module and the entire process starts again.