**CS 4341.002 Team Project**

**Team Name: We Are the Protoss**

**Spring 2016**

Contents-

1. File Listing
2. How to import into ModelSim
3. File Descriptions

1. File Listing-

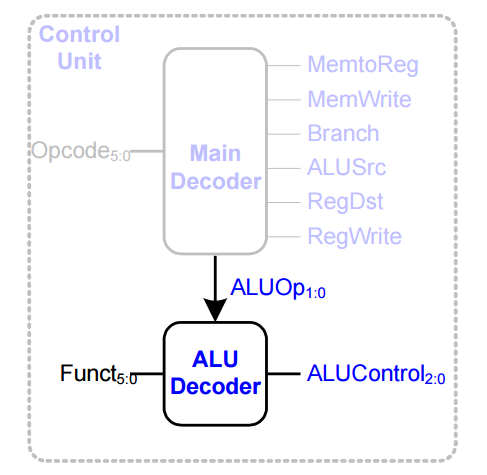
* ALUDecoder.v
* ALUDecoderTest.v
* Data\_Memory.v
* Instruction\_memory.v
* MainALU.v
* MainDecoder.v
* MainDecoderTester.v
* Multiplexer.v
* README.md
* Reg\_Test.v
* Register\_File.v
* ShiftLeftTwo.v
* SignExtender.v
* SignExtenderTest.v
* SimpleAdder.v
* SingleCycleProcessor.v

2. How to import into ModelSim-

1. Verify all files in the File Listing section are present in the zipped folder
2. Open ModelSim and choose File -> New -> Project
3. Choose any name and directory
4. Choose “Add Existing File” from the prompt
5. Navigate to our folder with all of our code and select all of the \*.v files
6. Compile all (every file should complete successfully)
7. You now have all the code necessary to run and test our program

3. File Descriptions-

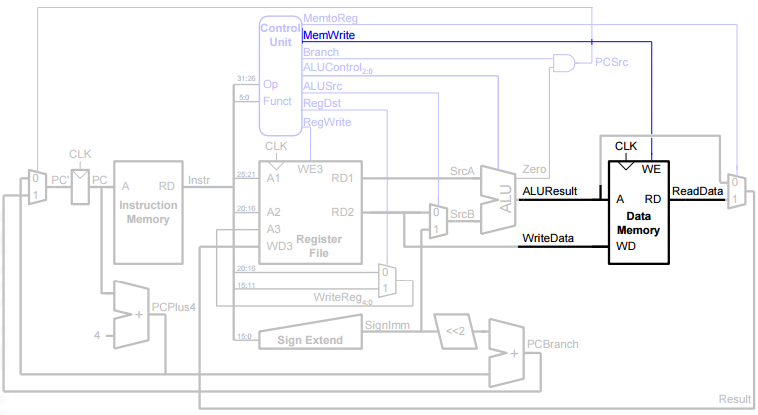
ALUDecoder.v



Takes in the ALUOp code and Funct code to choose an ALUControl. The ALUOp has a meaning of either add, subtract, or function. Funct Chooses a special function such as SLT. The output is ALUControl, which is sent to the Main ALU to control how the ALU handles its inputs.

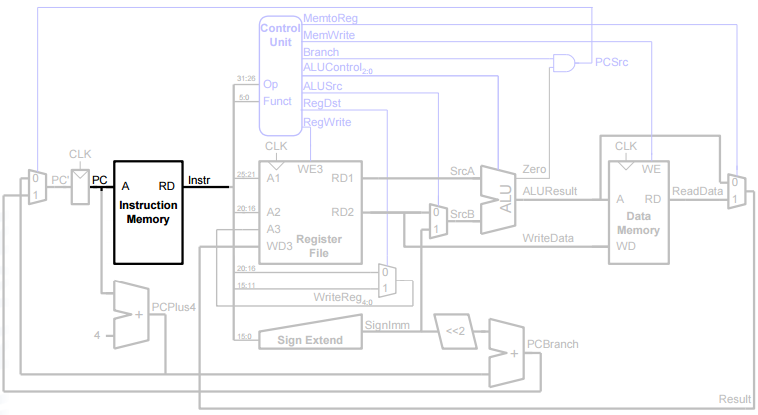
Data\_Memory.v

This is where our programs holds its main memory for all data. writeData is the data that is stored in memory at address addr if memWrite is true. readData is output from the address addr if memRead is true.



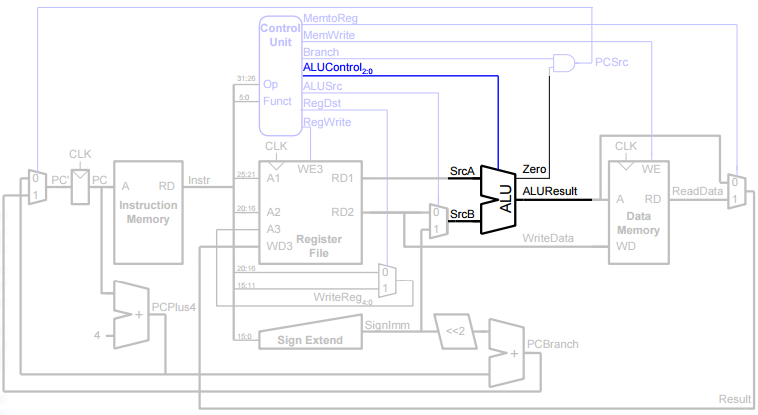
Instruction\_memory.v

This holds the set of instructions that we want to execute. addr holds the address we are at. instruct is the output that holds the encoded instruction in memory.



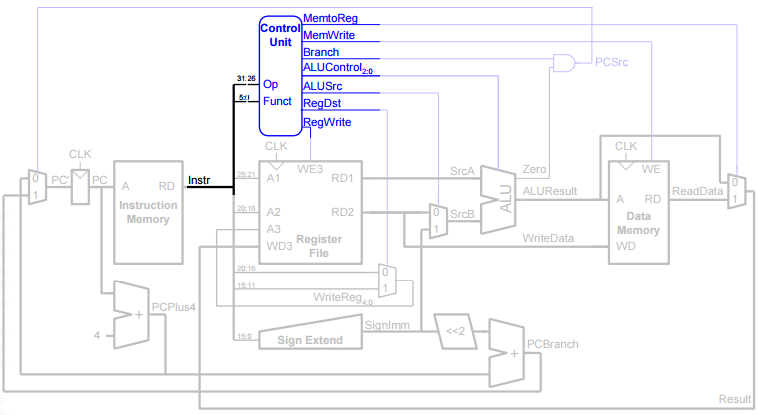
MainALU.v

This is where we handle the calculations needed to use the MIPS processor. SrcA and SrcB are the operands. ALUControl chooses what we do with the operands. ALUResult and Zero are the outputs that depends on the result of the operation executed.



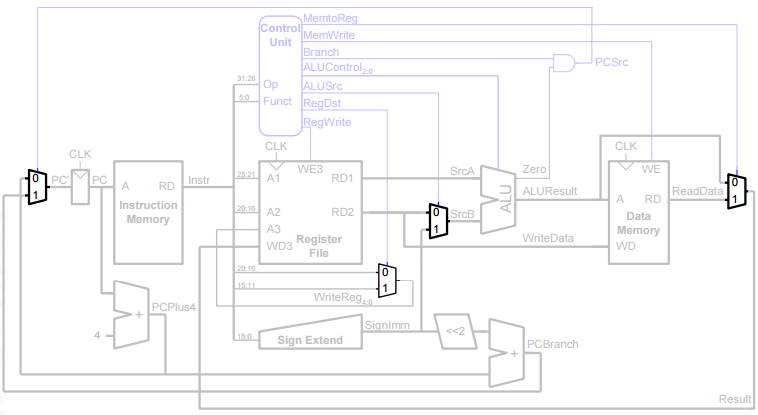
MainDecoder.v

This is our control unit that changes control signals depending on the opcode. These modify the execution of the datapath.



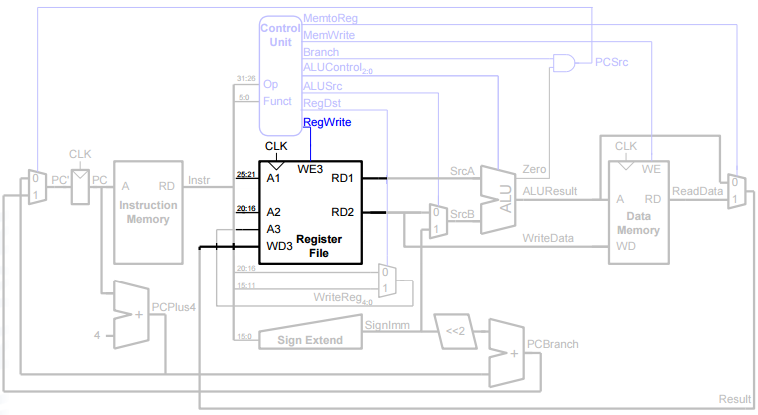
Multiplexer.v

This is a simple multiplexer module that selects between two inputs depending on the selection signal. The selected signal is then output to the next module.



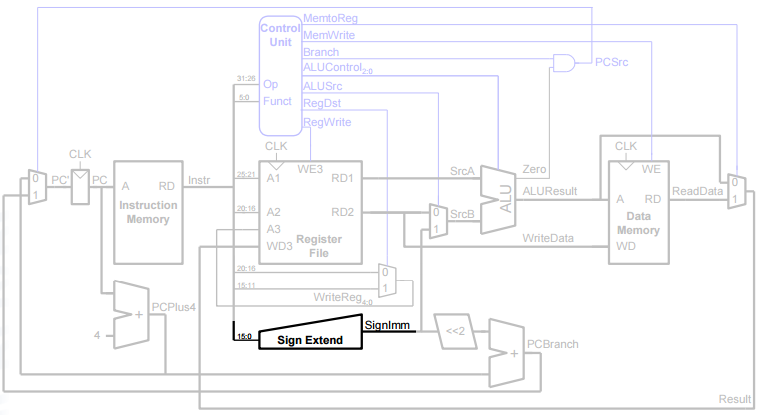
Register\_File.v

This holds our MIPS registers for usage in instruction operands. The outputs are chosen depending on the reg inputs. There are 32 registers, just like you would expect in MIPS. regwrite decides if we should write back to the register file from the writeback stage.



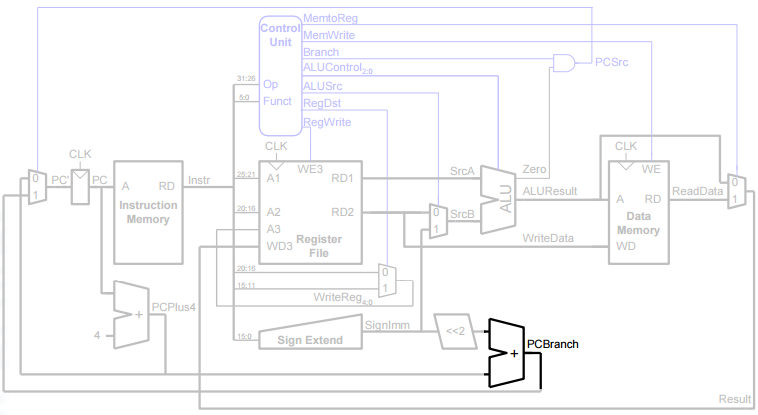
SignExtender.v

This is a simple sign extender from 16 bits to 32 bits so that we can extend the values preserving their sign.



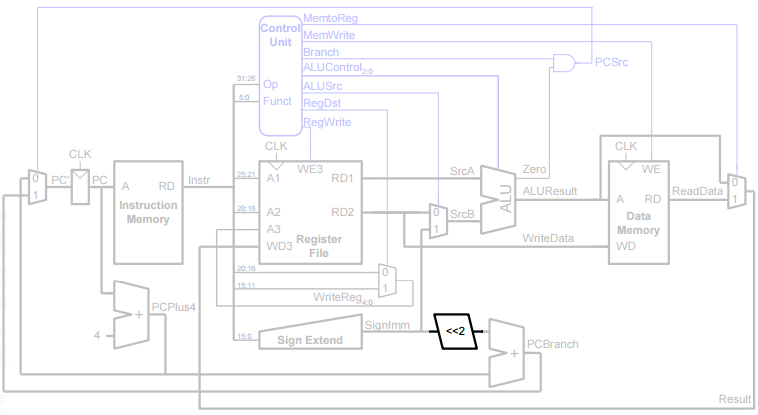
SimpleAdder.v

This is just an adder that outputs the addition of the 2 inputs. This is used to increment the PC value with and without branching.



ShiftLeftTwo.v

This is just a simple shifter that appends two 0’s to the right side of the input.



SingleCycleProcessor.v

This module is the collection of all other modules that drives the entire circuit. You will see a reference to all other modules here. The program counter is also changed here depending on a branch, jump, or just a +4 advancement. Many wires are defined here to serve as an intermediate location between the various modules.

Our current implementation simulates a single cycle processor, where we have 5 stages each advanced on a positive clock edge. We have a select number of instructions that can be executed, where we exclude complex instructions and pseudo instructions that would require more advanced decoding.

