1. Adder.v: Adder module takes two 32-bit data, “data1\_in” and “data2\_in”, as input, and adds these two values to output a 32 bits data, “data\_o”. One of the input data bus is from the current PC, and the other is hardwired to the integer 4 (from the CPU.v side). “data\_o” then sends the output, which is PC+4, back to PC whenever “data1\_in” signal changes.
2. ALU\_Control.v: ALU\_Control has two inputs and one output. The first input, “funct\_i”, which is 10-bit, is the [31:25, 14:12] (funct7, funct3) bits from the instruction. The second input, “ALUOp\_i”, is from the Control unit, which is 2-bit. In ALU\_Control, we first split the “funct\_i” signal into “funct7”, “funct3” with the help of wire. The module will first consider the “ALUOp\_i” signal. If “ALUOp\_i” is 2’b00, then we know the instruction is I-type, and if it isn’t, we know it’s R-type. Next, if it is I-type, we check “funct3”. If “funct3” is 3'b000, we set the output, “ALUCtrl\_o”, to perform ADDI. Otherwise, if “funct3” is 3'b101, “ALUCtrl\_o” will be set to perform SRAI. Moreover, if the instruction is R-type, we’ll check “funct7”. If “funct7” is all 0, we know it’s among AND, XOR, SLL, ADD. Otherwise, it’s SUB or MUL. If it’s among AND, XOR, SLL, and ADD, we check “funct3” to know which one is it. If it’s SUB or MUL, we check “funct7” to know which one is it. “ALUCtrl\_o” will send a 3-bit signal to the ALU every time “funct\_i”, “ALUOp\_i”, “funct7”, or “funct3” get updated.
3. ALU.v: ALU has two 32-bit data input, “data1\_i”and “data2\_i”. “data1\_i” is from the rs1 of the register file. “data2\_i” is from the MUX32 module. Additionally, ALU has a 3-bit input, “ALUCtrl\_i”, which is sent by ALU\_Control to control which operation ALU should perform. With these three inputs, ALU will perform arithmetic or logical operations determined by “ALUCtrl\_i” on “data1\_i” and “data2\_i”, and output result whenever “ALUCtrl\_i”, “data1\_i”, or “data2\_i” get updated. The result will be output as “data\_o”. There is also another output “Zero\_o”, but it’s not used and implemented in this homework.
4. Control.v: Control takes one input, “Op\_i”, which is the Opcode of the instruction from the instruction memory, and outputs “ALUOp\_o” to ALU-Control, “ALUSrc\_o” to MUX32, and “RegWrite\_o” to Registers. We can check the 5th bit of “Op\_i” to know if it’s I-type. If the 5th bit of “Op\_i” is 0, we know it’s I-type, so “ALUSrc\_o” will be set to 1, and “ALUOp\_o” will be set to 2’b00 to signify ALU\_Control that it’s a I-type. Otherwise “ALUSrc\_o” will be set to 0 and “ALUOp\_o” to 2’b10. In every case, “RegWrite\_o” will always be 1 since all 8 instruction require writing back to the register file. The outputs will refresh as soon as “Op\_i” refreshes.
5. CPU.v: CPU has 3 inputs “clk\_i”, “rst\_i”, and “start\_i”. They are all from testbench to control PC in CPU. There are many wires in CPU. “pc\_o” connects the output of PC to Instruction memory and Adder. “pc\_i” connects the output of Adder to PC. Adder’s “data2\_in” port here in CPU is hardwired to 4. “instr\_o” is from the output of instruction memory, and splits into 6 wires. The first is “Op\_i”, which is 7-bit, is the [6:0] bits of “instr\_o” and connects to Control. The second is “RS1addr\_i”, which is 5-bit, is the [19:15] bits of “instr\_o” and connects to Registers’ “RS1addr\_i” port. The third is “RS2addr\_i”, which is 5 bits, is the [24:20] bits of “instr\_o” and connects to Registers’ “RS2addr\_i” port. The forth is “RDaddr\_i”, which is also 5 bits, is the [11:7] bits of “instr\_o” and connects to Registers’ “RDaddr\_i” port. The fifth is “data\_i”, which is the [31:20] bits of “instr\_o” and connects to Sign\_Extend. The sixth wire is “funct\_i”, is the concatenation of instr\_o[31:25] and instr\_o[14:12], and connects to ALU\_Control’s “funct\_i” port. “ALUOp\_o” connects the output from Control’s “ALUOp\_o” port to ALU\_Control’s “ALUOp\_i” port, which is 3-bit. “ALUSrc\_o” connects the output of Control’s “ALUSrc\_o” port to “select\_i“port in MUX32, which is 1-bit. “RegWrite\_o” connects the output of Control to Registers’ “RegWrite\_i” port, which is 1-bit. “RS1data\_o” is a 32-bit data bus from register file rs1 and connects to ALU “data1\_i” port. “RS2data\_o” is another 32-bit data bus from register file rs2 and connects to “data1\_i” of MUX32. “data\_o” connects the output of Sign\_Extend, which is 32 bits, to “data2\_i” port of MUX32. “ALUCtrl\_i” connects the output of ALU\_Control “ALUCtrl\_o” to ALU, and it is 3-bit. “ALUresult” connects the output of ALU “data\_o” and send it back to “RDdata\_i” port of Registers, which is 32-bit.
6. MUX32 has three inputs and one output. Two of the inputs are 32-bit data inputs, which are “data1\_i” from rs1 of register file, and “data2\_i” from Sign-Extend. Another input is 1-bit “select\_i” signal. If “select\_i” is 1, MUX32 will output “data2\_i” as “data\_o” to perform an I-type instruction. If “select\_i” is 0, MUX32 will output “data1\_i” as “data\_o” to perform an R-type instruction.
7. Sign\_Extend only has 1 input and 1 output. The input “data\_i” is 12-bit, and the 32-bit output “data\_o” will copy data from “data\_i” to its lower bits and fill the rest of the bits with the bit of data\_i[11].

Dev. Environment:

OS: Windows 10

Compiler: iverilog

Text editing with VSCode