Computer Architecture HW4 Report

Environment

Windows, using iverilog.
Specifically, the following commands are used:

Compalition:

```
C:\iverilog\bin\iverilog.exe -o test CPU.v Control.v Adder.v PC.v
Instruction_Memory.v Registers.v MUX32.v Sign_Extend.v ALU_Control.v ALU.v
testbench.v
```

Execution:

```
C:\iverilog\bin\vvp.exe test
```

Implementation

The following files were not changed:
- testbench.v
- PC.v
- Registers.v
- Instruction_Memory.v

The following files were changed:
- CPU.v
- Adder.v
- Control.v
- ALU_Control.v
- Sign_Extend.v
- ALU.v
- MUX32.v
- MUX5.v

CPU.v (and other changed files):
Wires for PC and instruction are called inst_addr and inst, respectively.
Wires for outputs that do not matter are: meh1, meh2, meh3, meh4, and always_true (always true).
Wires for funct7 and funct3, which are then merged to become funct, are called fseven and fthree.

Control takes the first 7 bits of instruction and outputs ALUOp, which ALU_Control will use. In addition, ALUSrc_o is the inverse of the 5th bit of instruction.

Add_PC takes the PC, adds 4 to it, and outputs the result to PC.pc_I input.

PC takes Add_PC's output and returns the corresponding instruction's address.

Instruction_Memory takes the instruction's address and returns the actual instruction.

Registers takes rs1, rs2, and rd as register addresses, as well as write data and the write signal.
It outputs the read data for rs1 and rs2.

MUX_ALUSrc takes read data 2 from register and the sign extended value from Sign_Extend and uses Control.ALUSrc_o to select between them.

Sign_Extend takes imm (that is, the last 12 bits) and sign extends it to 32 bits.

ALU takes the two values, one from Mux, and one from read data 1, and performs an operation on those two depending on ALUControl.ALUCtrl_o.

ALU_Control takes {funct7, funct3} and outputs a control signal corresponding to the operation needed:
or: 000, and: 001, add: 010, sub: 011, mul: 101, addi: 100.
Note that the ALU does the same operation for both 010 (add) and 100 (addi).