ELEC-5200

# Computer Architecture

FROM: Jacob Howard

TO: Dr. Harris & Tucker Johnston

DUE DATE: 11/11/21

## CPU Design – Part 4

| **Operation** | **Format** | **Opcode (3-bits)** | **Destination Reg. (3-bits)** | **Source Reg. (3-bits)** | **Target Reg. (3-bits)** | **F4** | **Description** |
|---|---|---|---|---|---|---|---|
| | | | | *Arithmatic Instructions* | | | |
| **add** | R | 000 | Rd | $Rs_1$ | $Rs_2$ | 0000 | *Adds: Rd = Rs+Rt* |
| **sub** | R | 000 | Rd | $Rs_1$ | $Rs_2$ | 0001 | *Subtracts: Rd = Rs - Rt* |
| **and** | R | 000 | Rd | $Rs_1$ | $Rs_2$ | 0010 | *Logic AND: Rd = Rs&Rt* |
| **or** | R | 000 | Rd | $Rs_1$ | $Rs_2$ | 0011 | *Logic OR: Rd = Rs\|Rt* |
| **xor** | R | 000 | Rd | $Rs_1$ | $Rs_2$ | 0100 | *Logic XOR: Rd = Rs^Rt* |
| **sl** | R | 000 | Rd | $Rs_1$ | ---- | 0101 | *Shifts left: Rd = Rs<<1* |
| **sr** | R | 000 | Rd | $Rs_1$ | ---- | 0110 | *Shifts right: RD = Rs>>1* |
| | | | | *Immediate Arithmatic (extra un-unique instructions)* | | | |
| **addi** | i | 001 | Rd | $Rs_1$ | *Immidiate* | 0000 | *Add Immediate: Rd = Rs+imm* |
| **andi** | i | 001 | Rd | $Rs_1$ | *Immidiate* | 0001 | *Logic AND immediate: Rd = Rs&imm* |
| **ori** | i | 001 | Rd | $Rs_1$ | *Immidiate* | 0010 | *Logic OR immediate: Rd = Rs\|imm* |
| **xori** | i | 001 | Rd | $Rs_1$ | *Immidiate* | 0011 | *Logic XOR immediate: Rd = Rs^imm* |
| **sli** | i | 001 | Rd | $Rs_1$ | *Immidiate* | 0100 | *Shift left immediate:* |
| **sri** | i | 001 | Rd | $Rs_1$ | *Immidiate* | 0101 | *Shift right immediate:* |
| | | | | | | | |
| | | | | *Load/store instructions* | | | |
| **lw** | i | 010 | Rd | $Rs_1$ | *Immidiate* | 0000 | *Load word: Rd = Rs1+imm* |
| **sw** | s | 011 | ---- | $Rs_1$ | $Rs_2$ | 0000 | *Store word: M[R[Rs1]+imm](15:0)=[Rs2](15:0)* |
| | | | | **Branches/Jumps** | | | |
| **operation** | **Type: sb** | **Opcode 3-bits** | **3-bits** | **3-bits** | ***4-bits*** | **F4** | ***Description*** |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **beq** | sb | 100 | $Rs_1$ | $Rs_2$ | *immidiate* | 0000 | *Branch if equal to: if(Rs==Rt)PC=PC+imm* |
| **bgt** | sb | 100 | $Rs_1$ | $Rs_2$ | *immidiate* | 0001 | *Branch if greater than: If($Rs_1$>$Rs_2$)PC=PC+imm* |
| **bge** | sb | 100 | $Rs_1$ | $Rs_2$ | *immidiate* | 0010 | *Branch if greater than or equal to: If(Rs1>=Rs2)PC=PC+imm* |
| **blt** | sb | 100 | $Rs_1$ | $Rs_2$ | *immidiate* | 0011 | *Branch if less than: If(Rs1<Rs2)PC=PC+imm* |
| | | | | | | | |
| **Operation** | Type: uj | Opcode 3-bits | 3-bits | 10-bis | *Description* | | |
| **jal** | uj | 101 | Rd | Immediate | *Jump and link: Rd=PC+4;PC=Rs1+imm* | | |
| | | | | | | | |
| **Operation** | **Format** | **Opcode (3-bits)** | **Destination Reg. (3-bits)** | **Source Reg. (3-bits)** | **Target Reg. (3-bits)** | **F4** | **Description** |
| **jalr** | i | 110 | Rd | $Rs_1$ | *immidiate* | 0000 | *Jump and link register: Rd=PC+4;PC=Rs1+imm* |
| ***Halt*** | | | | | | | |
| **halt** | - | 111 | - | - | - | 0000 | *Halts the processor* |

## Instructions

Above is an overview of the instructions made in Part 1 of the CPU Design Project. This will be a helpful table to reference for this part of the project (shown above due to format erros in word). You may also reference below for the instruction format table. This will also be usefuil to reference to understand overall design.

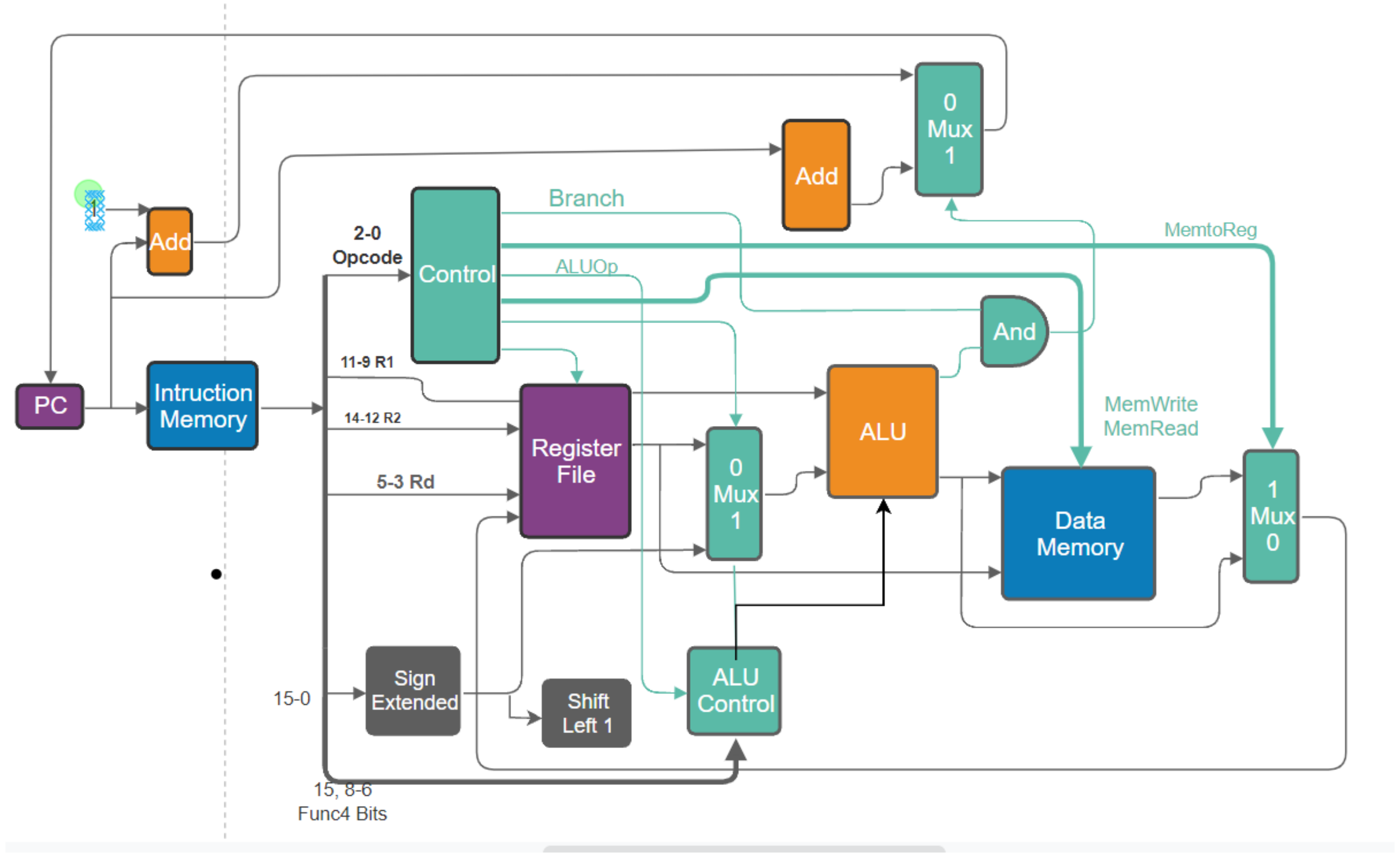
In this project, we were required to connect all of our Verilog datapath modules together and write test instructions to verify functionality.

| Type | 15th bit | 14-12 | 11-9 | 8-6 | 5-3 | 2-0 bits |
|---|---|---|---|---|---|---|
| R-Type | Funct4[3] | R2 | R1 | Func4[2:0] | Rd | opcode |
| I-Tpye | Imm[3:0] | | R1 | Func4[2:0] | rd | opcode |
| S-Type | Imm[3] | R2 | R1 | Func4[2:0] | Imm[2:0] | opcode |
| SB-Type | Imm[3] | R2 | R1 | Func4[2:0] | Imm[2:0] | opcode |
| UJ-Type | Imm[9:0] | | | | rd | opcode |

*Format Table*

## Datapath

Below is an image of the current datapath being used in the design. The goal for this part of the project is to connect everything together and verify instruction functionality. (*Note that the ALU Control block should only not be connected to the MUX, but to the ALU. When trying to fix this, the datapath software I was using would not let me remove the green wire connecting the control to the mux.*)

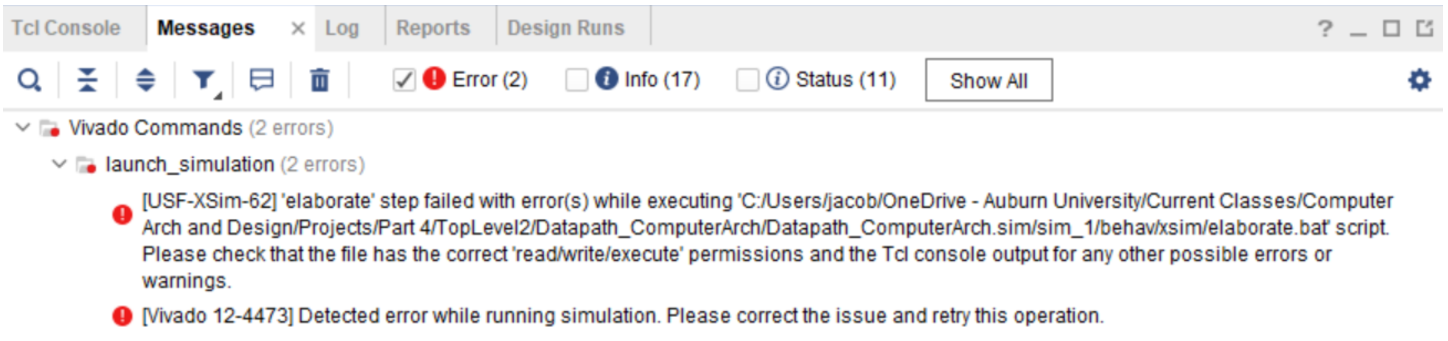


*Datapath*

## Testing

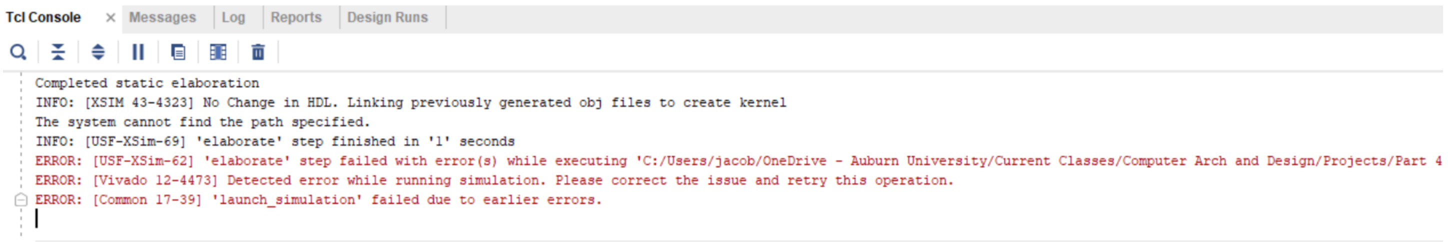
Previously, I had written two modules to connect all DataPath blocks together. I have now converted the top-level design to one module named *"cpu"*. Though I have created a simplified module to connect every block and verified connections, I receive to undescriptive vivado errors. I have spent many ours trying to fix all issues, but I am not sure what left to verify. Indicidul blocks function as expected during testing and I have verified the top-level cpu module multiple times. I have gone to the TA for assistance and had peers review my top-level module, but I still receive simulation erros.

The errors and tlc consol can be seen below in a labeled screenshot. My top-level module code is also shown below and correctly labeled. I will show my test case code I had written for the top-level cpu module. Lastly, I will show some example instructions I was trying to run. The example instructions are labbled and shown in binary and assembly with descriptions.



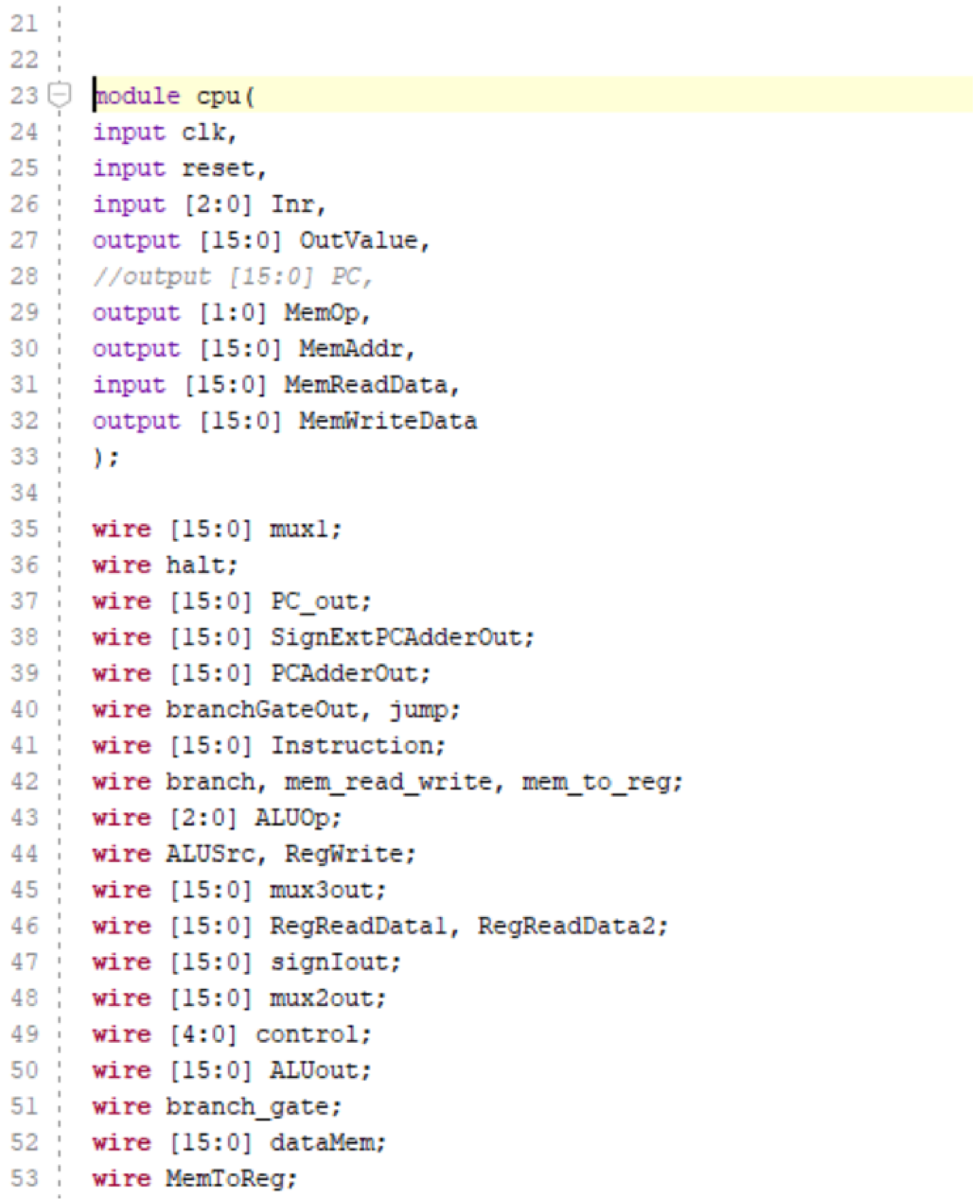
```
Vivado Commands (2 errors)
  launch_simulation (2 errors)
    [USF-XSim-62] 'elaborate' step failed with error(s) while executing 'C:/Users/jacob/OneDrive - Auburn University/Current Classes/Computer Arch and Design/Projects/Part 4/TopLevel2/Datapath_ComputerArch/Datapath_ComputerArch.sim/sim_1/behav/xsim/elaborate.bat' script. Please check that the file has the correct 'read/write/execute' permissions and the Tcl console output for any other possible errors or warnings.
    [Vivado 12-4473] Detected error while running simulation. Please correct the issue and retry this operation.
```

*Vivado Simulation Erros*



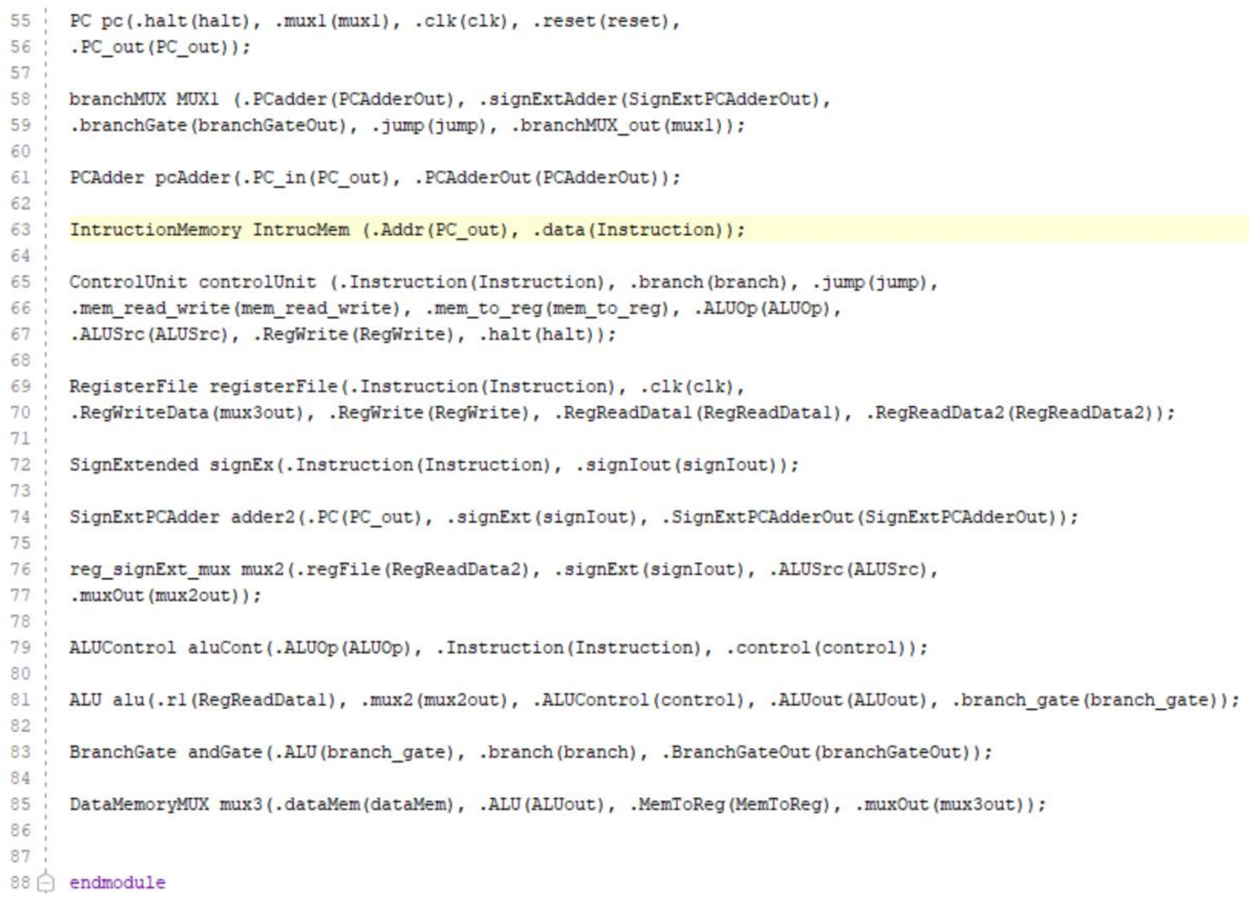
```
Completed static elaboration
INFO: [XSIM 43-4323] No Change in HDL. Linking previously generated obj files to create kernel
The system cannot find the path specified.
INFO: [USF-XSim-69] 'elaborate' step finished in '1' seconds
ERROR: [USF-XSim-62] 'elaborate' step failed with error(s) while executing 'C:/Users/jacob/OneDrive - Auburn University/Current Classes/Computer Arch and Design/Projects/Part 4
ERROR: [Vivado 12-4473] Detected error while running simulation. Please correct the issue and retry this operation.
ERROR: [Common 17-39] 'launch_simulation' failed due to earlier errors.
```

*TLC Consol Erros*



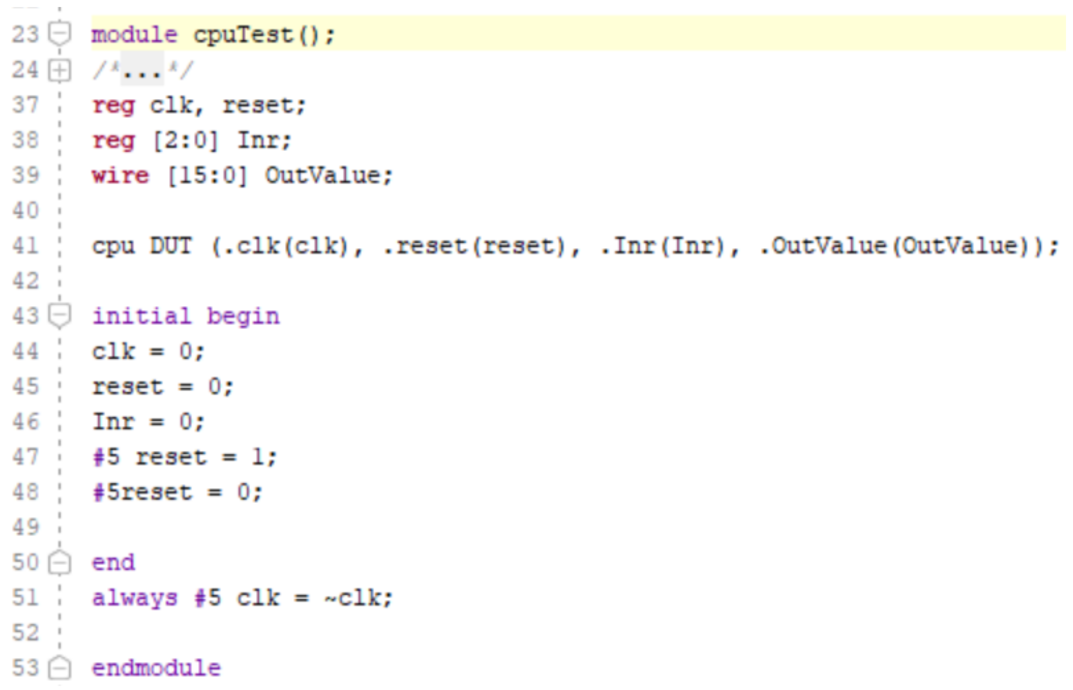
```
module cpu(
input clk,
input reset,
input [2:0] Inr,
output [15:0] OutValue,
//output [15:0] PC,
output [1:0] MemOp,
output [15:0] MemAddr,
input [15:0] MemReadData,
output [15:0] MemWriteData
);

wire [15:0] mux1;
wire halt;
wire [15:0] PC_out;
wire [15:0] SignExtPCAdderOut;
wire [15:0] PCAdderOut;
wire branchGateOut, jump;
wire [15:0] Instruction;
wire branch, mem_read_write, mem_to_reg;
wire [2:0] ALUOp;
wire ALUSrc, RegWrite;
wire [15:0] mux3out;
wire [15:0] RegReadData1, RegReadData2;
wire [15:0] signIout;
wire [15:0] mux2out;
wire [4:0] control;
wire [15:0] ALUout;
wire branch_gate;
wire [15:0] dataMem;
wire MemToReg;
```

Top-Level “CPU” Code (Lines 23-53)



```
55  PC pc(.halt(halt), .mux1(mux1), .clk(clk), .reset(reset),
56  .PC_out(PC_out));
57
58  branchMUX MUX1 (.PCadder(PCAdderOut), .signExtAdder(SignExtPCAdderOut),
59  .branchGate(branchGateOut), .jump(jump), .branchMUX_out(mux1));
60
61  PCAdder pcAdder(.PC_in(PC_out), .PCAdderOut(PCAdderOut));
62
63  IntructionMemory IntrucMem (.Addr(PC_out), .data(Instruction));
64
65  ControlUnit controlUnit (.Instruction(Instruction), .branch(branch), .jump(jump),
66  .mem_read_write(mem_read_write), .mem_to_reg(mem_to_reg), .ALUOp(ALUOp),
67  .ALUSrc(ALUSrc), .RegWrite(RegWrite), .halt(halt));
68
69  RegisterFile registerFile(.Instruction(Instruction), .clk(clk),
70  .RegWriteData(mux3out), .RegWrite(RegWrite), .RegReadData1(RegReadData1), .RegReadData2(RegReadData2));
71
72  SignExtended signEx(.Instruction(Instruction), .signIout(signIout));
73
74  SignExtPCAdder adder2(.PC(PC_out), .signExt(signIout), .SignExtPCAdderOut(SignExtPCAdderOut));
75
76  reg_signExt_mux mux2(.regFile(RegReadData2), .signExt(signIout), .ALUSrc(ALUSrc),
77  .muxOut(mux2out));
78
79  ALUControl aluCont(.ALUOp(ALUOp), .Instruction(Instruction), .control(control));
80
81  ALU alu(.r1(RegReadData1), .mux2(mux2out), .ALUControl(control), .ALUout(ALUout), .branch_gate(branch_gate));
82
83  BranchGate andGate(.ALU(branch_gate), .branch(branch), .BranchGateOut(branchGateOut));
84
85  DataMemoryMUX mux3(.dataMem(dataMem), .ALU(ALUout), .MemToReg(MemToReg), .muxOut(mux3out));
86
87
88  endmodule
```

*Top-Level "CPU" Code (Lines 55-8)*



```
module cpuTest();
/*...*/
reg clk, reset;
reg [2:0] Inr;
wire [15:0] OutValue;

cpu DUT (.clk(clk), .reset(reset), .Inr(Inr), .OutValue(OutValue));

initial begin
clk = 0;
reset = 0;
Inr = 0;
#5 reset = 1;
#5reset = 0;

end
always #5 clk = ~clk;

endmodule
```

*Top-Level "CPU" Testbench*

| Imm | R1 | Func | Rd | Op | ASSEMBLY | DESC. |
|---|---|---|---|---|---|---|
| 0001 | 001 | 000 | 001 | 001 | addi x1, 1, x1 | storing 1 into x1 |

| Func[3] | R2 | R1 | Func[2:0] | Rd | Op | ASSEMBLY | DESC. |
|---|---|---|---|---|---|---|---|
| 0 | 001 | 001 | 000 | 001 | 000 | dd x1, x1, x1 | adding 1+1 and storing 2 back into x1 |
| 0 | 001 | 001 | 001 | 001 | 000 | sub x1, x1, x1 | subtracting 2-2 and storing 0 into x1 |

| Imm | R1 | Func | Rd | Op | ASSEMBLY | DESC. |
|---|---|---|---|---|---|---|
| 0001 | 001 | 000 | 010 | 001 | addi x1, 1, x2 | storing 1 into x2 for testing next arithmatic instruction |

| Func[3] | R2 | R1 | Func[2:0] | Rd | Op | ASSEMBLY | DESC. |
|---|---|---|---|---|---|---|---|
| 0 | 010 | 001 | 010 | 000 | 000 | and x2, x1, x1 | AND’ing 1 and 0 and storing 0 into x1 |

| Imm | R1 | Func | Rd | Op | ASSEMBLY | DESC. |
|---|---|---|---|---|---|---|
| 0001 | 000 | 010 | 001 | 001 | andi x0, x1, 1 | AND’ing im 1 and 0 and storing in x1 |
| 0001 | 000 | 010 | 001 | 001 | ori x0, x1, 1 | OR’ing im 1 and 0 and storing in x1 |

| Imm[3] | R2 | R1 | Func[2:0] | Imm[2:0[ | Op | ASSEMBLY | DESC. |
|---|---|---|---|---|---|---|---|
| 0 | 001 | 001 | 000 | 001 | 100 | beq x1, x1, 1 | branching 1 line |
| 0 | 000 | 010 | 001 | 001 | 100 | bgt x0, x2, 2 | not branching 2 lines bc 0<1 |

| Imm[9:0] | Rd | Op | ASSEMBLY |
|---|---|---|---|
| 0000000001 | 011 | 101 | Jal x3, 1 |

| Imm[3] | R1 | Func[2:0] | Rd | Op | ASSEMBLY |
|---|---|---|---|---|---|
| 001 | 001 | 000 | 010 | 110 | jalr x1, x2, 1 |

| Don’t Care | Op | ASSEMBLY | DESC. |
|---|---|---|---|
| xxxxxxxxxxxxx | 111 | Halt | Halts cpu |

| Imm | R1 | Func | Rd | Op | ASSEMBLY | DESC. |
|---|---|---|---|---|---|---|
| 0001 | 100 | 000 | 100 | 001 | addi x4, x4, 1 | Using addi to test the halt (storing 1 into x4) |

*Test Instructions*

*(Stored in .mem file as 16-bit words)*