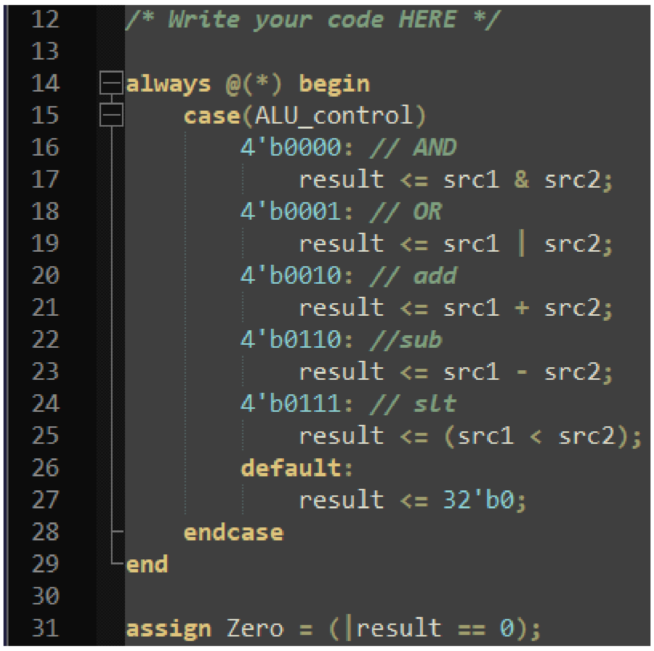
Lab04-Report

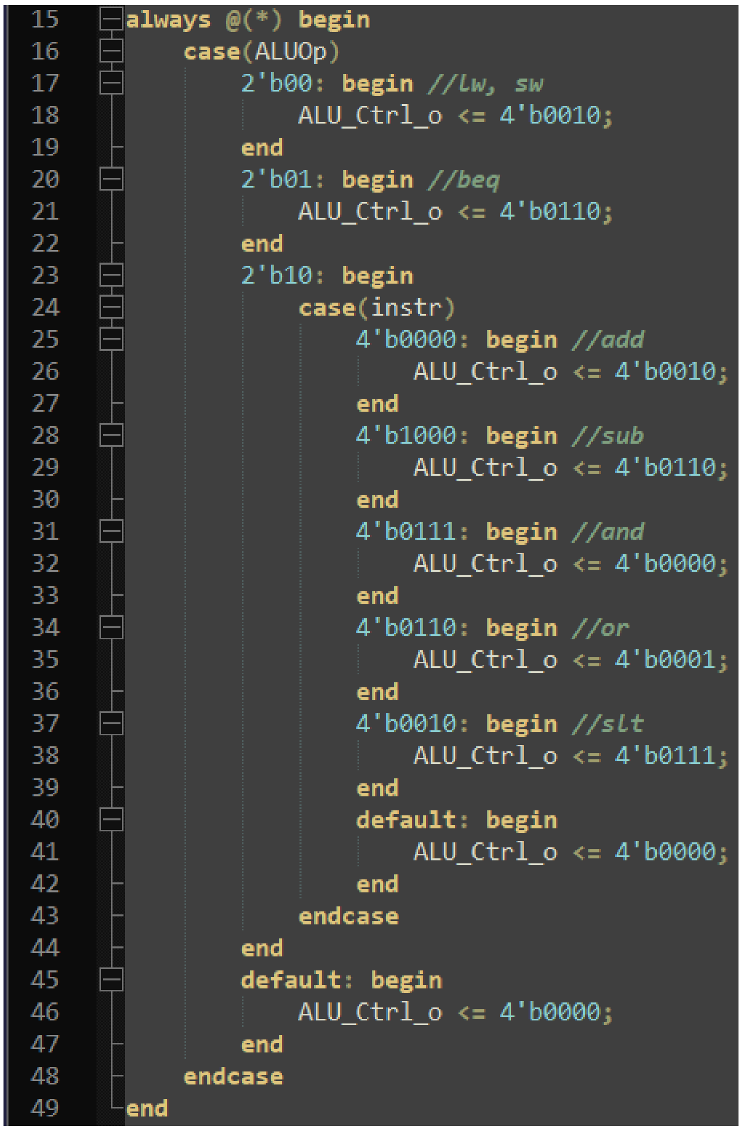
1. Detailed description of the implementation

1. alu.v: perform corresponding action using a case statement, and assign zero = 1 iff all result bits is 0.



```verilog
/* Write your code HERE */

always @(*) begin
    case(ALU_control)
        4'b0000: // AND
            result <= src1 & src2;
        4'b0001: // OR
            result <= src1 | src2;
        4'b0010: // add
            result <= src1 + src2;
        4'b0110: //sub
            result <= src1 - src2;
        4'b0111: // slt
            result <= (src1 < src2);
        default:
            result <= 32'b0;
    endcase
end

assign Zero = (|result == 0);
```

2. ALU_Ctrl.v: identify lw, sw, beq by ALUOp at first, then identify arithmetic operations by instr.



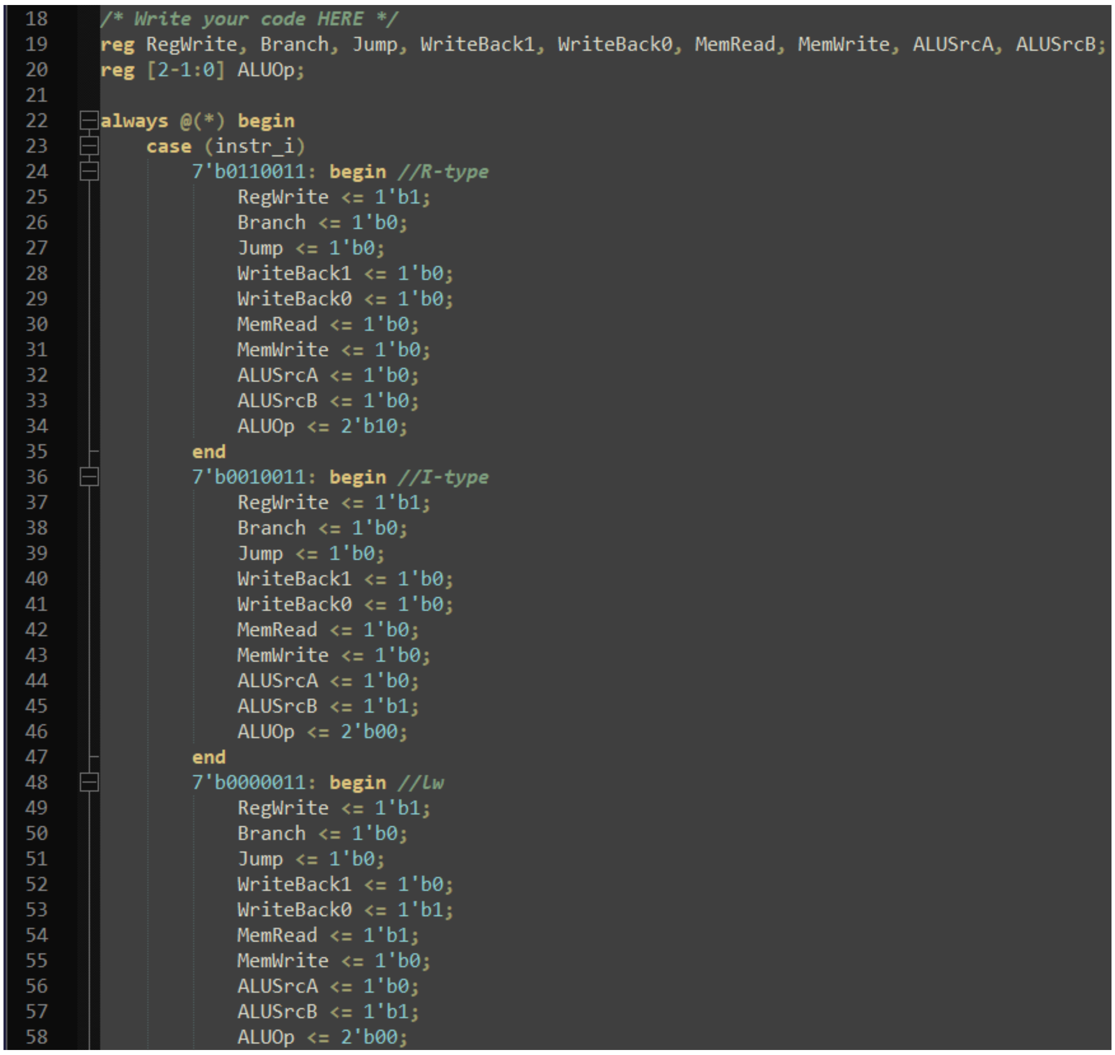
```verilog
always @(*) begin
    case(ALUOp)
        2'b00: begin //lw, sw
            ALU_Ctrl_o <= 4'b0010;
        end
        2'b01: begin //beq
            ALU_Ctrl_o <= 4'b0110;
        end
        2'b10: begin
            case(instr)
                4'b0000: begin //add
                    ALU_Ctrl_o <= 4'b0010;
                end
                4'b1000: begin //sub
                    ALU_Ctrl_o <= 4'b0110;
                end
                4'b0111: begin //and
                    ALU_Ctrl_o <= 4'b0000;
                end
                4'b0110: begin //or
                    ALU_Ctrl_o <= 4'b0001;
                end
                4'b0010: begin //slt
                    ALU_Ctrl_o <= 4'b0111;
                end
                default: begin
                    ALU_Ctrl_o <= 4'b0000;
                end
            endcase
        end
        default: begin
            ALU_Ctrl_o <= 4'b0000;
        end
    endcase
end
```

3. Decoder.v: assign all control signals according to this chart. And I declared the output as reg to assign them in a loop.

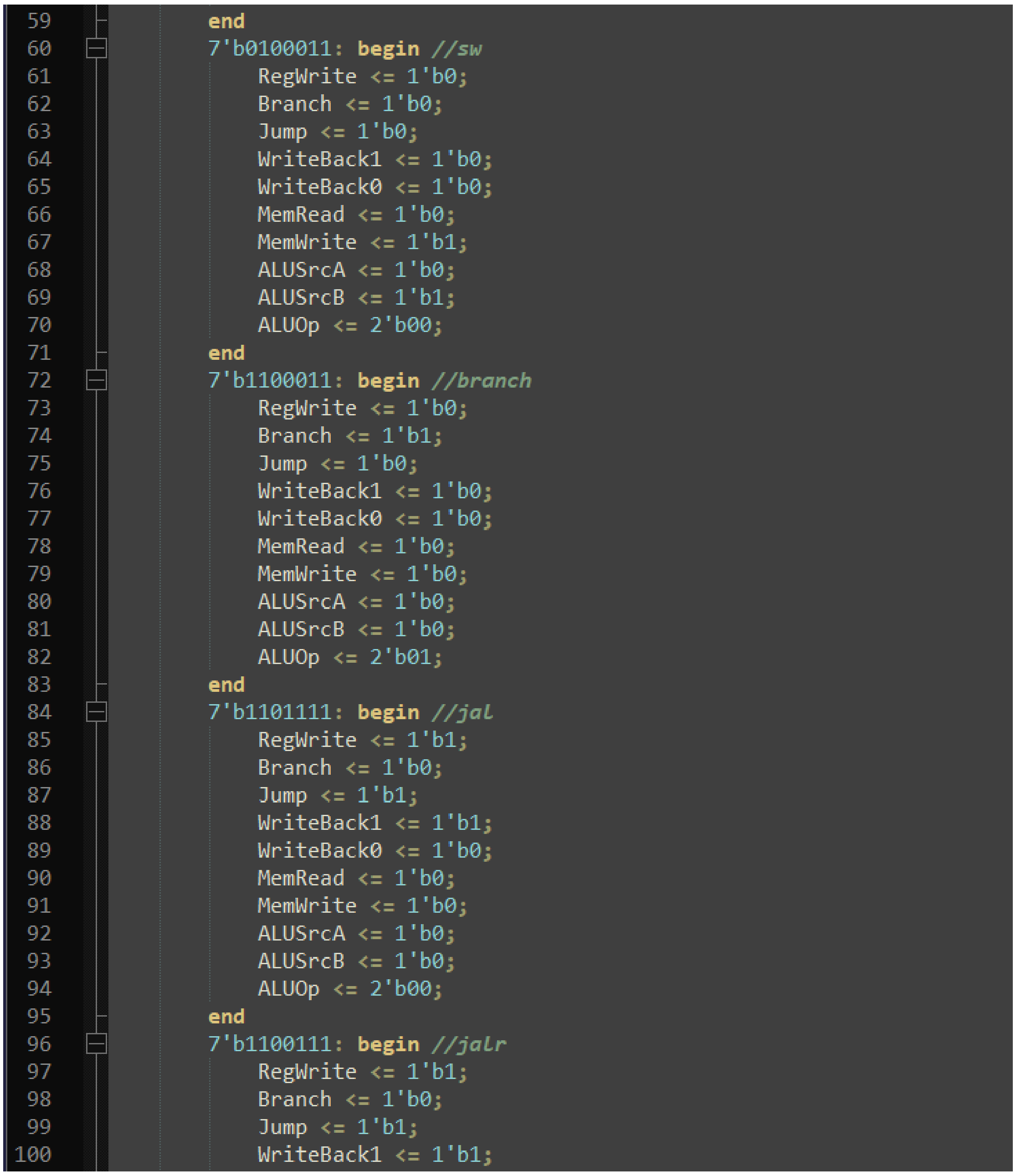
Implement instructions

- R-type
  - add
  - slt
- I-type
  - addi
- lw
- sw
- beq
- jal & jalr

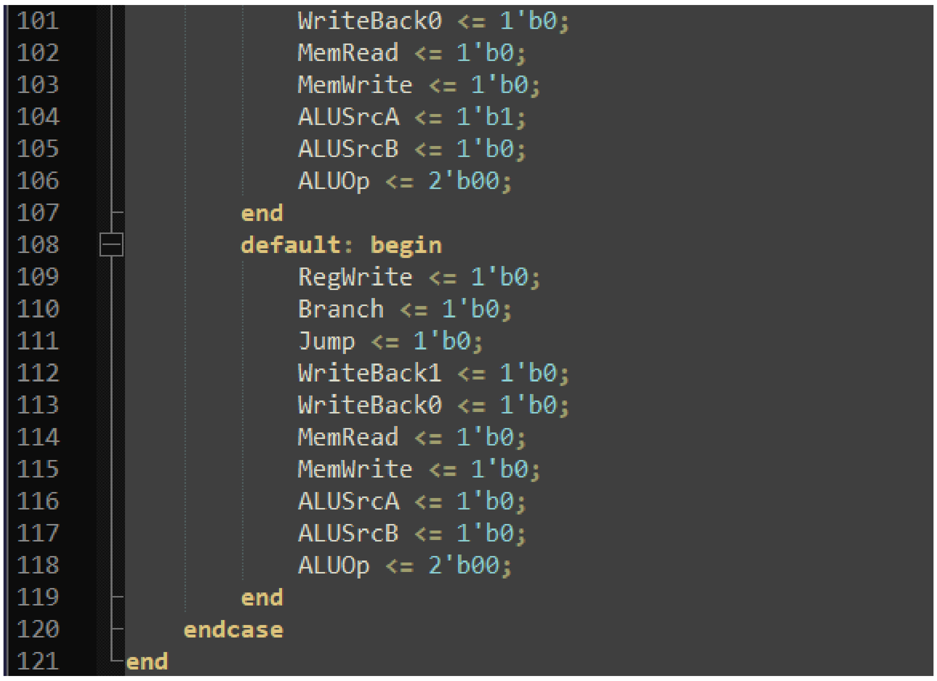
Instr	RW	B	J	WB	MR	MW	Src	Op	code
R-type	1	0	0	00	0	0	00	10	0110011
addi	1	0	0	00	0	0	01	00	0010011
Load	1	0	0	01	1	1	01	00	0000011
Store	X	0	0	X	0	0	01	00	0100011
Branch	X	1	0	X	0	0	00	01	1100011
JAL	1	0	1	10	0	0	00	X	1101111
JALR	1	0	1	10	0	0	10	X	1100111



```
/* Write your code HERE */
reg RegWrite, Branch, Jump, WriteBack1, WriteBack0, MemRead, MemWrite, ALUSrcA, ALUSrcB;
reg [2-1:0] ALUOp;

always @(*) begin
    case (instr_i)
        7'b0110011: begin //R-type
            RegWrite <= 1'b1;
            Branch <= 1'b0;
            Jump <= 1'b0;
            WriteBack1 <= 1'b0;
            WriteBack0 <= 1'b0;
            MemRead <= 1'b0;
            MemWrite <= 1'b0;
            ALUSrcA <= 1'b0;
            ALUSrcB <= 1'b0;
            ALUOp <= 2'b10;
        end
        7'b0010011: begin //I-type
            RegWrite <= 1'b1;
            Branch <= 1'b0;
            Jump <= 1'b0;
            WriteBack1 <= 1'b0;
            WriteBack0 <= 1'b0;
            MemRead <= 1'b0;
            MemWrite <= 1'b0;
            ALUSrcA <= 1'b0;
            ALUSrcB <= 1'b1;
            ALUOp <= 2'b00;
        end
        7'b0000011: begin //lw
            RegWrite <= 1'b1;
            Branch <= 1'b0;
            Jump <= 1'b0;
            WriteBack1 <= 1'b0;
            WriteBack0 <= 1'b1;
            MemRead <= 1'b1;
            MemWrite <= 1'b0;
            ALUSrcA <= 1'b0;
            ALUSrcB <= 1'b1;
            ALUOp <= 2'b00;
```

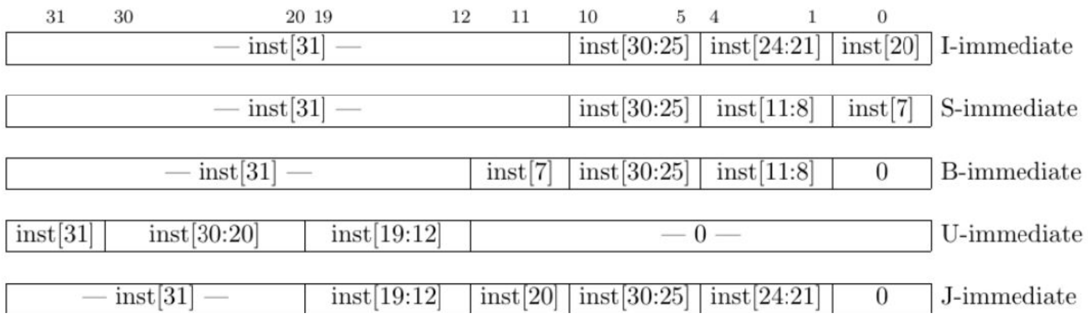


```
        end
        7'b0100011: begin //sw
            RegWrite <= 1'b0;
            Branch <= 1'b0;
            Jump <= 1'b0;
            WriteBack1 <= 1'b0;
            WriteBack0 <= 1'b0;
            MemRead <= 1'b0;
            MemWrite <= 1'b1;
            ALUSrcA <= 1'b0;
            ALUSrcB <= 1'b1;
            ALUOp <= 2'b00;
        end
        7'b1100011: begin //branch
            RegWrite <= 1'b0;
            Branch <= 1'b1;
            Jump <= 1'b0;
            WriteBack1 <= 1'b0;
            WriteBack0 <= 1'b0;
            MemRead <= 1'b0;
            MemWrite <= 1'b0;
            ALUSrcA <= 1'b0;
            ALUSrcB <= 1'b0;
            ALUOp <= 2'b01;
        end
        7'b1101111: begin //jal
            RegWrite <= 1'b1;
            Branch <= 1'b0;
            Jump <= 1'b1;
            WriteBack1 <= 1'b1;
            WriteBack0 <= 1'b0;
            MemRead <= 1'b0;
            MemWrite <= 1'b0;
            ALUSrcA <= 1'b0;
            ALUSrcB <= 1'b0;
            ALUOp <= 2'b00;
        end
        7'b1100111: begin //jalr
            RegWrite <= 1'b1;
            Branch <= 1'b0;
            Jump <= 1'b1;
            WriteBack1 <= 1'b1;
```

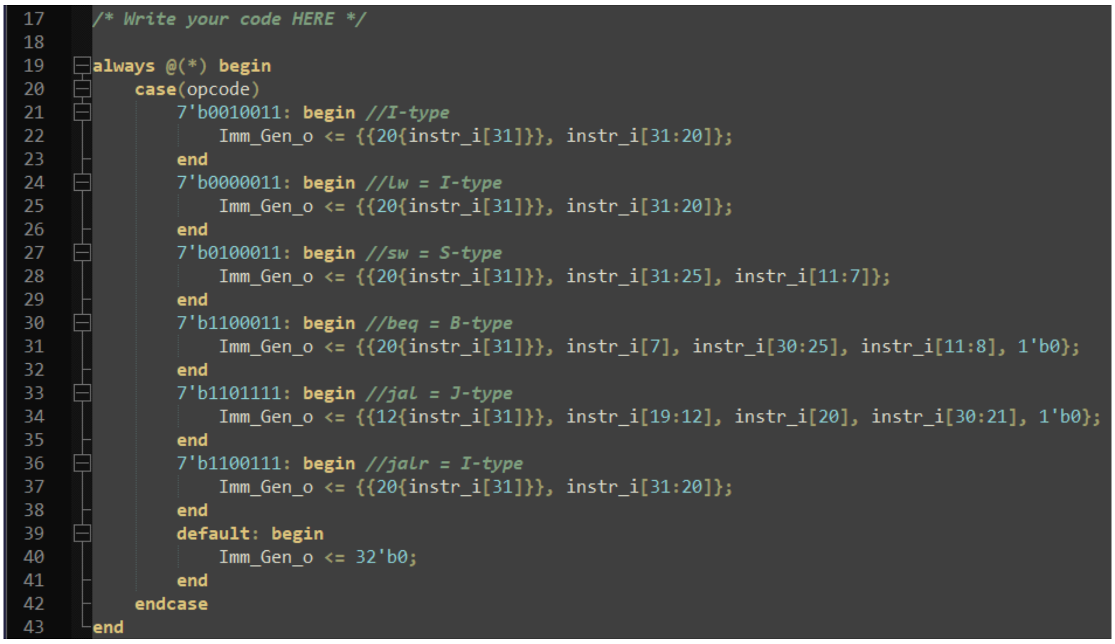


```
                WriteBack0 <= 1'b0;
                MemRead <= 1'b0;
                MemWrite <= 1'b0;
                ALUSrcA <= 1'b1;
                ALUSrcB <= 1'b0;
                ALUOp <= 2'b00;
            end
            default: begin
                RegWrite <= 1'b0;
                Branch <= 1'b0;
                Jump <= 1'b0;
                WriteBack1 <= 1'b0;
                WriteBack0 <= 1'b0;
                MemRead <= 1'b0;
                MemWrite <= 1'b0;
                ALUSrcA <= 1'b0;
                ALUSrcB <= 1'b0;
                ALUOp <= 2'b00;
            end
        endcase
end
```

4. Imm_Gen.v: implement *each* type of imm by this sheet.



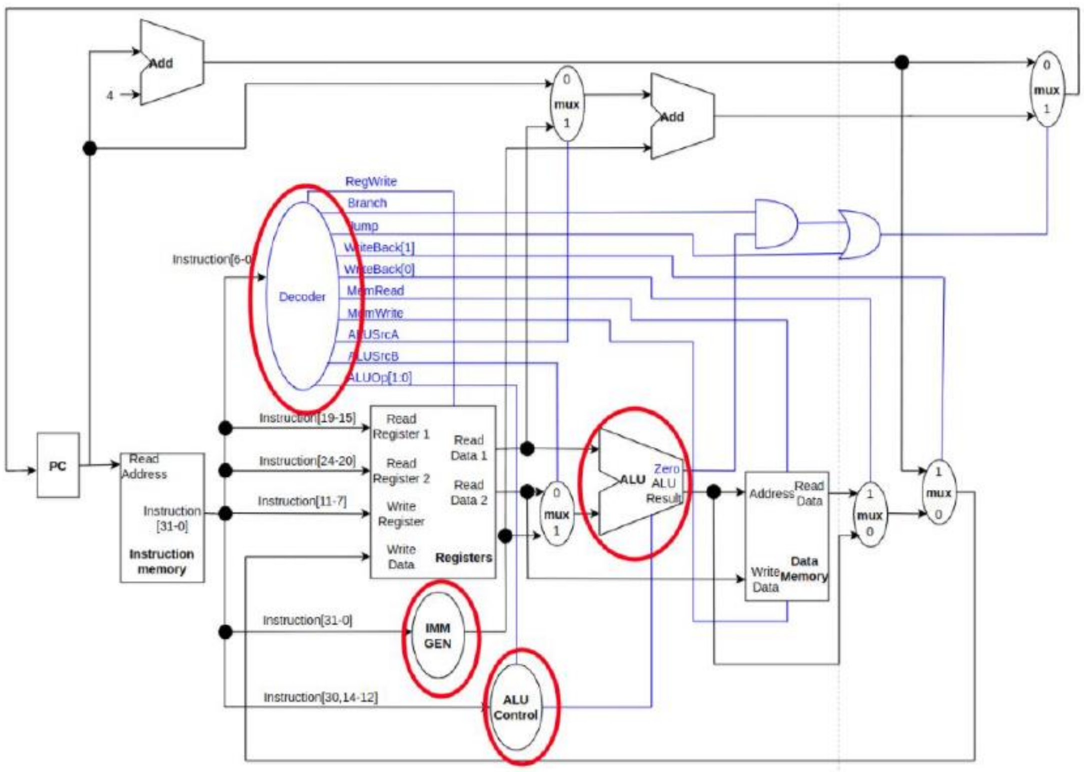
31	30 20	19 12	11	10 5	4 1	0	
— inst[31] —				inst[30:25]	inst[24:21]	inst[20]	I-immediate
— inst[31] —				inst[30:25]	inst[11:8]	inst[7]	S-immediate
— inst[31] —			inst[7]	inst[30:25]	inst[11:8]	0	B-immediate
inst[31]	inst[30:20]	inst[19:12]	— 0 —				U-immediate
— inst[31] —		inst[19:12]	inst[20]	inst[30:25]	inst[24:21]	0	J-immediate

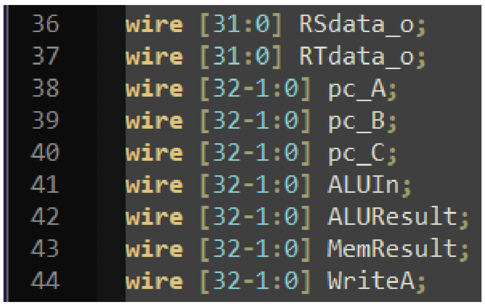


```
/* Write your code HERE */

always @(*) begin
    case(opcode)
        7'b0010011: begin //I-type
            Imm_Gen_o <= {{20{instr_i[31]}}, instr_i[31:20]};
        end
        7'b0000011: begin //lw = I-type
            Imm_Gen_o <= {{20{instr_i[31]}}, instr_i[31:20]};
        end
        7'b0100011: begin //sw = S-type
            Imm_Gen_o <= {{20{instr_i[31]}}, instr_i[31:25], instr_i[11:7]};
        end
        7'b1100011: begin //beq = B-type
            Imm_Gen_o <= {{20{instr_i[31]}}, instr_i[7], instr_i[30:25], instr_i[11:8], 1'b0};
        end
        7'b1101111: begin //jal = J-type
            Imm_Gen_o <= {{12{instr_i[31]}}, instr_i[19:12], instr_i[20], instr_i[30:21], 1'b0};
        end
        7'b1100111: begin //jalr = I-type
            Imm_Gen_o <= {{20{instr_i[31]}}, instr_i[31:20]};
        end
        default: begin
            Imm_Gen_o <= 32'b0;
        end
    endcase
end
```

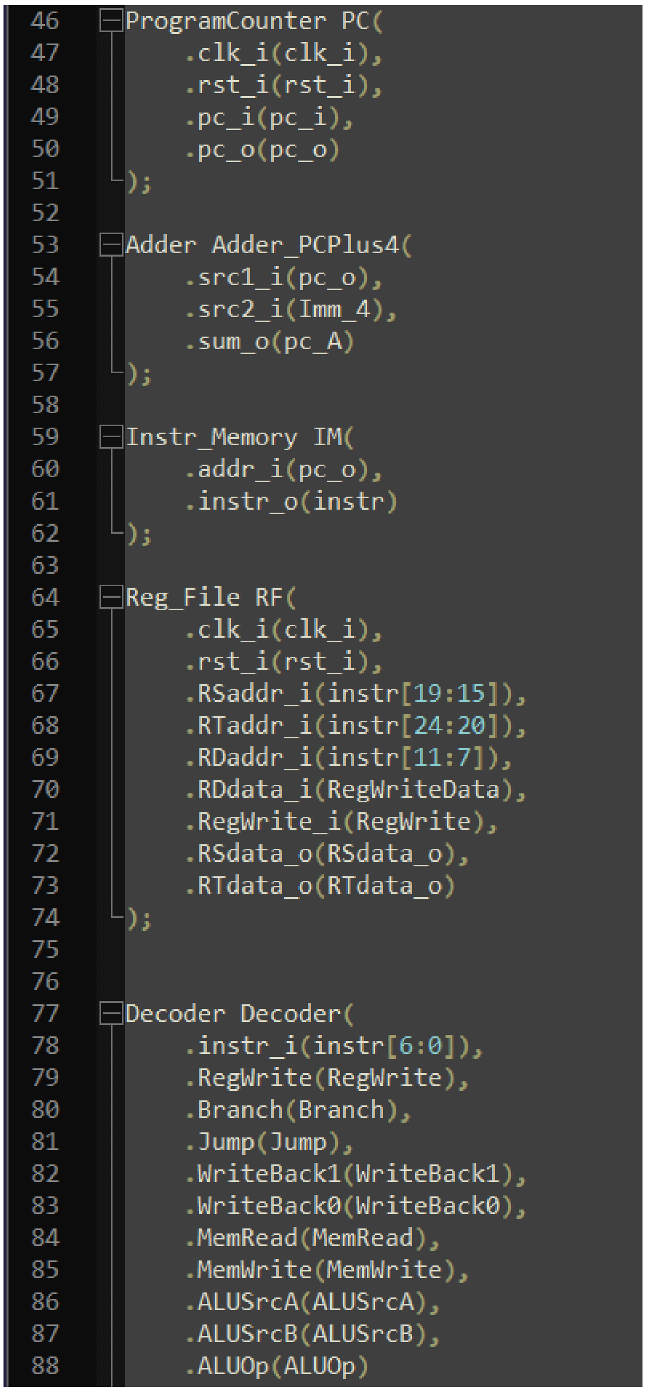
5. Simple_Single_CPU.v: *added these wires to connect*

between the modules, and connect according to the graph below.

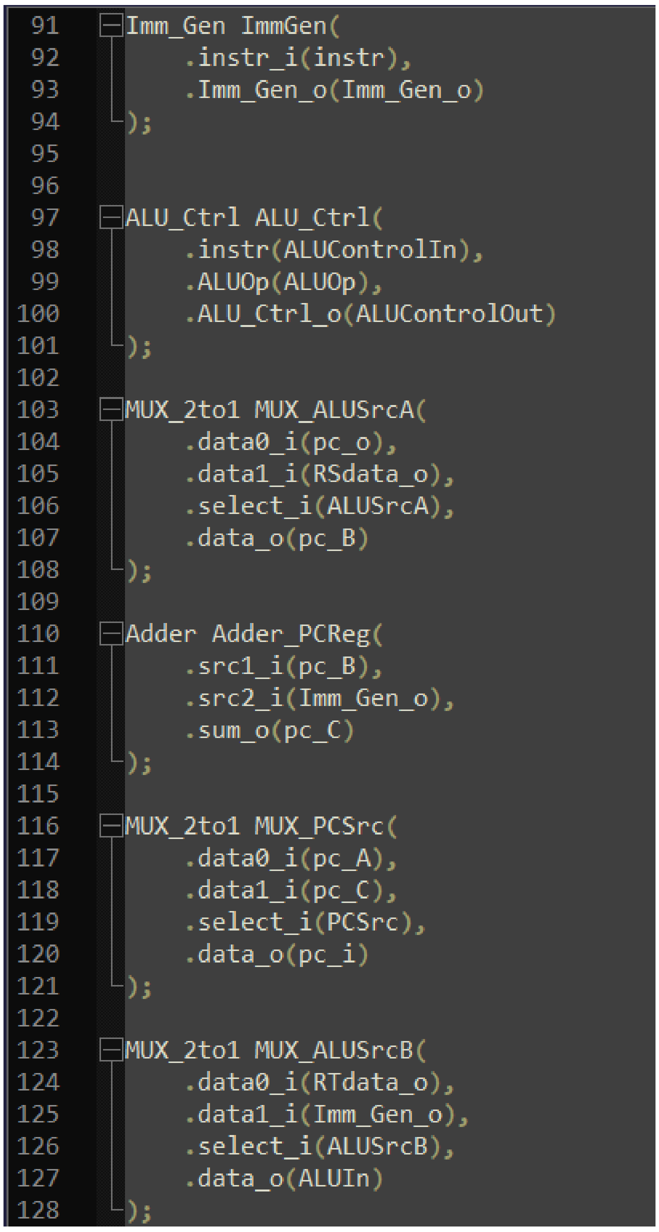


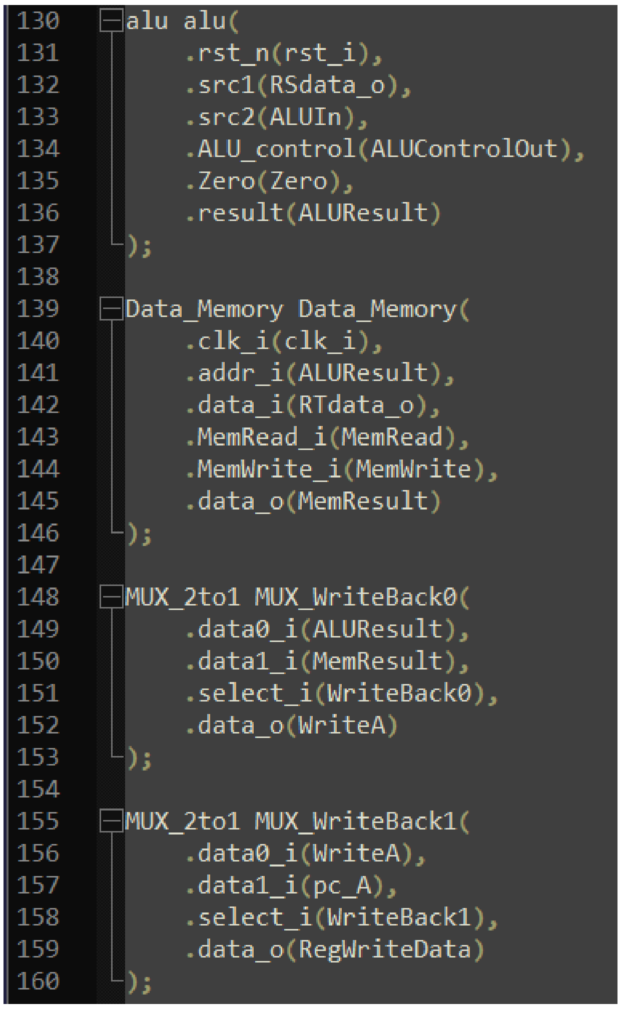


```
wire [31:0] RSdata_o;
wire [31:0] RTdata_o;
wire [32-1:0] pc_A;
wire [32-1:0] pc_B;
wire [32-1:0] pc_C;
wire [32-1:0] ALUIn;
wire [32-1:0] ALUResult;
wire [32-1:0] MemResult;
wire [32-1:0] WriteA;
```



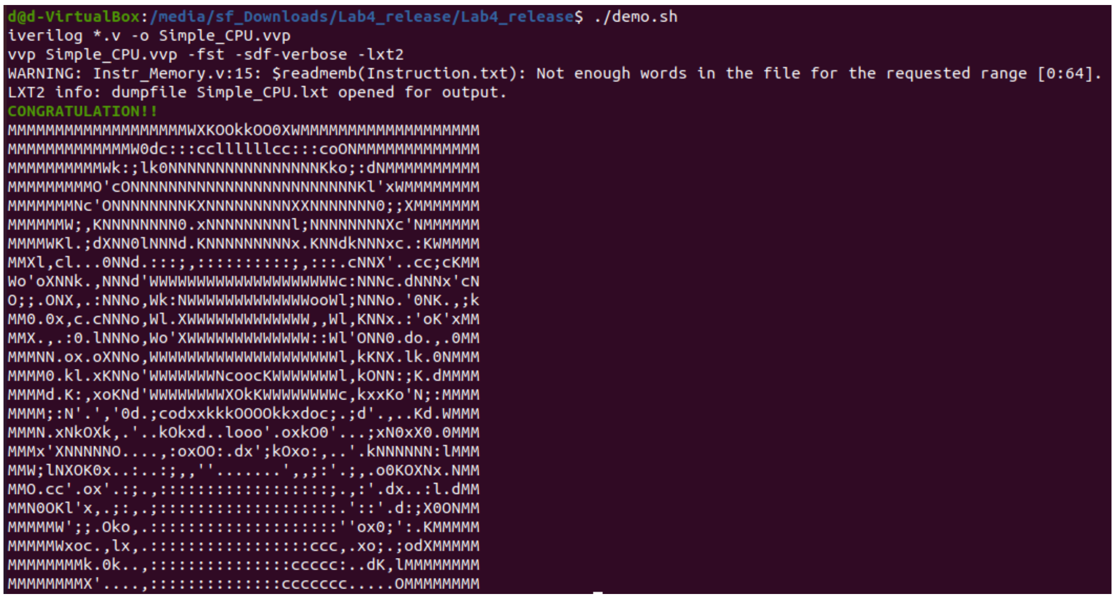
```
ProgramCounter PC(
    .clk_i(clk_i),
    .rst_i(rst_i),
    .pc_i(pc_i),
    .pc_o(pc_o)
);

Adder Adder_PCPlus4(
    .src1_i(pc_o),
    .src2_i(Imm_4),
    .sum_o(pc_A)
);

Instr_Memory IM(
    .addr_i(pc_o),
    .instr_o(instr)
);

Reg_File RF(
    .clk_i(clk_i),
    .rst_i(rst_i),
    .RSaddr_i(instr[19:15]),
    .RTaddr_i(instr[24:20]),
    .RDaddr_i(instr[11:7]),
    .RDdata_i(RegWriteData),
    .RegWrite_i(RegWrite),
    .RSdata_o(RSdata_o),
    .RTdata_o(RTdata_o)
);


Decoder Decoder(
    .instr_i(instr[6:0]),
    .RegWrite(RegWrite),
    .Branch(Branch),
    .Jump(Jump),
    .WriteBack1(WriteBack1),
    .WriteBack0(WriteBack0),
    .MemRead(MemRead),
    .MemWrite(MemWrite),
    .ALUSrcA(ALUSrcA),
    .ALUSrcB(ALUSrcB),
    .ALUOp(ALUOp)
```


```
Imm_Gen ImmGen(
        .instr_i(instr),
        .Imm_Gen_o(Imm_Gen_o)
);


ALU_Ctrl ALU_Ctrl(
        .instr(ALUControlIn),
        .ALUOp(ALUOp),
        .ALU_Ctrl_o(ALUControlOut)
);

MUX_2to1 MUX_ALUSrcA(
        .data0_i(pc_o),
        .data1_i(RSdata_o),
        .select_i(ALUSrcA),
        .data_o(pc_B)
);

Adder Adder_PCReg(
        .src1_i(pc_B),
        .src2_i(Imm_Gen_o),
        .sum_o(pc_C)
);

MUX_2to1 MUX_PCSrc(
        .data0_i(pc_A),
        .data1_i(pc_C),
        .select_i(PCSrc),
        .data_o(pc_i)
);

MUX_2to1 MUX_ALUSrcB(
        .data0_i(RTdata_o),
        .data1_i(Imm_Gen_o),
        .select_i(ALUSrcB),
        .data_o(ALUIn)
);
```



```
alu alu(
    .rst_n(rst_i),
    .src1(RSdata_o),
    .src2(ALUIn),
    .ALU_control(ALUControlOut),
    .Zero(Zero),
    .result(ALUResult)
);

Data_Memory Data_Memory(
    .clk_i(clk_i),
    .addr_i(ALUResult),
    .data_i(RTdata_o),
    .MemRead_i(MemRead),
    .MemWrite_i(MemWrite),
    .data_o(MemResult)
);

MUX_2to1 MUX_WriteBack0(
    .data0_i(ALUResult),
    .data1_i(MemResult),
    .select_i(WriteBack0),
    .data_o(WriteA)
);

MUX_2to1 MUX_WriteBack1(
    .data0_i(WriteA),
    .data1_i(pc_A),
    .select_i(WriteBack1),
    .data_o(RegWriteData)
);
```

2. Implementation result



```
d@d-VirtualBox:/media/sf_Downloads/Lab4_release/Lab4_release$ ./demo.sh
iverilog *.v -o Simple_CPU.vvp
vvp Simple_CPU.vvp -fst -sdf-verbose -lxt2
WARNING: Instr_Memory.v:15: $readmemb(Instruction.txt): Not enough words in the file for the requested range [0:64].
LXT2 info: dumpfile Simple_CPU.lxt opened for output.
CONGRATULATION!!
MMMMMMMMMMMMMMMMMMMWXKOOkkOO0XWMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMW0dc:::ccllllllcc:::coONMMMMMMMMMMMMM
MMMMMMMMMMWk:;lk0NNNNNNNNNNNNNNNNNNKko;:dNMMMMMMMMMM
MMMMMMMMMO'cONNNNNNNNNNNNNNNNNNNNNNNNNNKl'xWMMMMMMMM
MMMMMMMNc'ONNNNNNNNKXNNNNNNNNNXXNNNNNNN0;;XMMMMMMM
MMMMMMW;,KNNNNNNNN0.xNNNNNNNNNl;NNNNNNNNXc'NMMMMMM
MMMMWKl.;dXNN0lNNNd.KNNNNNNNNNx.KNNdkNNNxc.:KWMMMM
MMXl,cl...0NNd.:::;,:::::::::;,:::.cNNX'..cc;cKMM
Wo'oXNNk.,NNNd'WWWWWWWWWWWWWWWWWWWWc:NNNc.dNNNx'cN
O;;.ONX,.:NNNo,Wk:NWWWWWWWWWWWWWooWl;NNNo.'0NK.,;k
MM0.0x,c.cNNNo,Wl.XWWWWWWWWWWWWW,,Wl,KNNx.:'oK'xMM
MMX.,.:0.lNNNo,Wo'XWWWWWWWWWWWWW::Wl'ONN0.do.,.0MM
MMMNN.ox.oXNNo,WWWWWWWWWWWWWWWWWWWWWl,kKNX.lk.0NMMM
MMMM0.kl.xKNNo'WWWWWWWNcoocKWWWWWWWl,kONN:;K.dMMMM
MMMMd.K:,xoKNd'WWWWWWWWXOkKWWWWWWWWc,kxxKo'N;:MMMM
MMMM;:N'.','0d.;codxxkkkOOOOkkxdoc;.;d'.,..Kd.WMMM
MMMN.xNkOXk,.'..kOkxd..looo'.oxkO0'...;xN0xX0.0MMM
MMMx'XNNNNNO....,:oxOO:.dx';kOxo:,..'.kNNNNNN:lMMM
MMW;lNXOK0x..:..:;,,''.......',,;:'.;,.o0KOXNx.NMM
MMO.cc'.ox'.:;.,:::::::::::::::::::;.,:'.dx..:l.dMM
MMN0OKl'x,.;:,.;:::::::::::::::::::::.'::'.d:;X0ONMM
MMMMMW';;.Oko,.:::::::::::::::::::::''ox0;':.KMMMMM
MMMMMWxoc.,lx,.:::::::::::::::::ccc,.xo;.;odXMMMMM
MMMMMMMMk.0k..,::::::::::::::ccccc:..dK,lMMMMMMMMM
MMMMMMMMX'....,::::::::::::::ccccccc.....OMMMMMMMMM
```

3. Problems encountered and solutions

- The slt result was wrong in ALU.v.
  - ✓ This is solved after I changed the input reg to signed, so that the less than (<) operator can work.
- The jalr result is wrong, result in all instruction undefined after jalr.
  - ✓ I found that I implemented jalr imm according to J-type imm, but after searching on the Internet, I found jalr should use I-type imm. Corrected the implementation in Imm_Gen.v and this problem is solved.