UM-SJTU JOINT INSTITURE

Intro to Computer Organization
(VE370)

Project 2

Group Report - Pipelined CPU

prof. Zheng

Group 14

Kaiyang Chen 518021910962
Xinmiao Yu 518021910792
Zhouxiang Fu 518370910101
Shutong Zhang 517370910180

DATE: 12 Nov. 2020

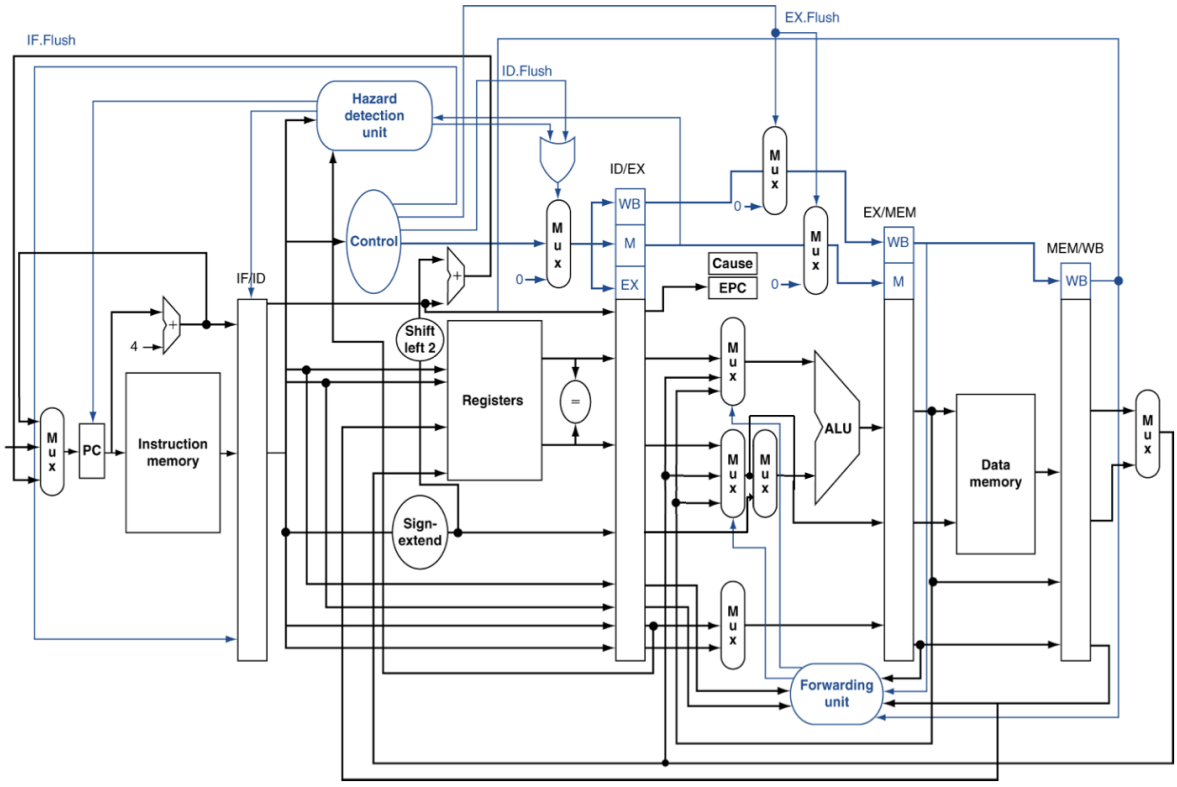
1. Objectives

Model pipelined implementation of MIPS computer in Verilog that support a subset of MIPS instruction set including:

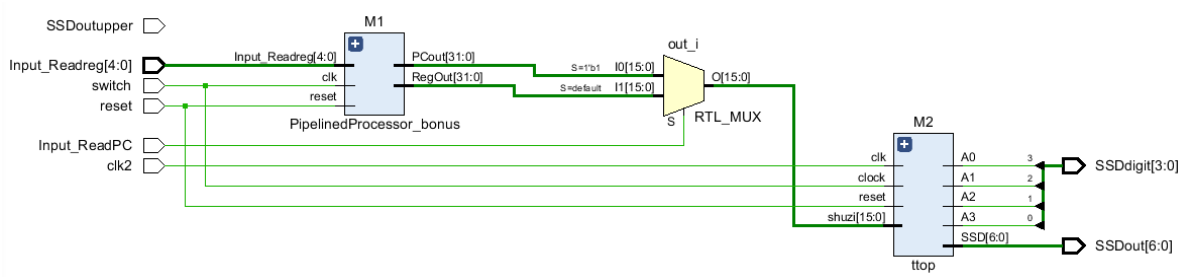
- The memory-reference instructions load word (*lw*) and store word (*sw*)
- The arithmetic-logical instructions add, *addi*, sub, and, *andi*, or, and *slt*
- The jumping instructions branch equal (*beq*), branch not equal (*bne*), and jump (*j*)

2. Implementation

2.1 Block Diagram for Pipelined implementation of MIPS architecture

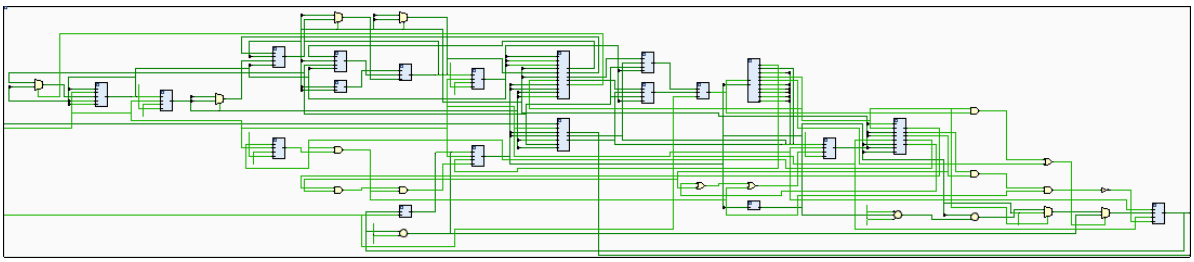


2.2 The RTL Schematic of the Verilog Design



RTL diagram of general design

Signal Name	Direction	Description
reset	Input	Reset signal used for initialization
clk	Input	Clock signal used to control the change of PC
stall	Input	Stall signal used to decide whether we should stop to update PC in order to add NOP instructions.
pcin	Input	Input of PC
pcout	Output	Output of PC



RTL diagram of pipeline implementation

Signal Name	Direction	Description
reset	Input	Reset signal used for initialization
clk	Input	Clock signal used to control all of the registers
Input_ReadPC	Input	Control signal of whether to check PC value
PCout	Output	Value of PC that we want to check its value
Regout	Output	Value of reg that we want to check its value

```
module PipelinedProcessor_bonus(
    input clk,
    input [4:0] Input_Readreg,
    input reset,
    input Input_ReadPC,
    output [31:0] PCout,
    output [31:0] RegOut
    );
    //WIRE IF STAGE
    wire    [31:0]  pc_in_if,
                    pc_out_if,
                    pcplus4_if,
                    instruction_if,
                    branchjumpaddress_if;
    wire            branchjump_if,// if there is branch or jump, then 1
                    if_id_write_hazard,
                    pc_write_hazard,
                    pc_write_2; // beq + data hazard, R-type

    //WIRE ID STAGE
    wire    [31:0]  pcplus4_id,
                    instruction_id,
```

```
                    regdata1_id,
                    regdata2_id,
                    regdata1final_id,
                    regdata2final_id,
                    branchaddress_id,
                    jumpaddress_id,
                    imm_signEx_id,
                    eq_mux1_id,
                    eq_mux2_id;
    wire    [25:0]  jump_inst_id;
    wire    [5:0]   op_id;
    wire    [4:0]   rs_id,
                    rt_id,
                    rd_id;
    wire    [15:0]  immi_id;
    wire            branch_id,//if there is successful branch, then 1(beq or
bne)
                    beq_id, // delete bne
                    regwrite_id,
                    memtoreg_id,
                    memwrite_id,
                    memread_id,
                    regdst_id,
                    ALUsrc_id,
                    com_res_id,
                    jump,
                    pc_write_3_id;
                //  regdst_id1, ALUsrc_id1, regwrite_id1, memtoreg_id1,
memwrite_id1, memread_id1, ALUop_id1,
           // regdata1_id1, regdata2_id1, imm_signEx_id1, rs_id1, rt_id1,
rd_id1;
    wire    [1:0]   ALUop_id;

    //WIRE EX STAGE
    wire            regwrite_ex,
                    memtoreg_ex,
                    memwrite_ex,
                    memread_ex,
                    regdst_ex,
                    ALUsrc_ex,
                    pc_write_3_ex,
                    exzero; // seems like useless?
    wire    [1:0]   ALUop_ex;
    wire    [31:0]  regdata1final_ex,
                    regdata2final_ex,
                    immi_signEx_ex,
                    ALUin1final_ex,
                    ALUin2final_ex,
                    ALUresult_ex,
                    ALUin2_ex;
    wire    [5:0]   func_ex;
    wire    [4:0]   rs_ex,
                    rt_ex,
                    rd_ex,
                    registerrd_ex;
    wire    [3:0]   ALUctrl_ex;
```

```
    //WIRE MEM
    wire    [31:0]  ALUresult_mem,
                    regdata2final_mem,
                    writedata_mem,
                    dataread_mem;
    wire    [4:0]   registerrd_mem;
    wire            regwrite_mem,
                    memtoreg_mem,
                    memwrite_mem,
                    memread_mem;

    //WIRE WB
    wire            regwrite_wb,
                    memtoreg_wb,
                    memread_wb;
    wire    [4:0]   registerrd_wb;
    wire    [31:0]  writeback_wb,
                    memdata_wb,
                    ALUresult_wb;

    //WIRE FORWARDING
    wire    [1:0]   forwardingeqin1,
                    forwardingeqin2;
    wire            memSrc;
    wire    [1:0]   forwardingALUin1,
                    forwardingALUin2;

    //WIRE Hazard
    wire            stall, // =1 when hazard
                    IF_flush_id;

// blocks

    // in IF STAGE
    PC PC_(
        .reset(reset),
        .clk(clk),
        .stall(~(pc_write_hazard & pc_write_2 & (pc_write_3_id &
pc_write_3_ex))), // when hazard, pc_write_hazard=0 // stall=0, write
        .pcin(pc_in_if),
        .pcout(pc_out_if)
        );
    InstrMem InsrtMem_(
        .reset(reset),
        .address(pc_out_if),
        .instruction(instruction_if)
        );
    assign pcplus4_if = pc_out_if + 4;
    assign pc_in_if = (branchjump_if) ? branchjumpaddress_if : pcplus4_if; //
MUX: decide which is pcin

    // IF/ID Reg
    register #(64) IF_ID(
        .write(if_id_write_hazard & pc_write_2 & (pc_write_3_id &
pc_write_3_ex)), //  when hazard if_id_write_hazard = 0
        .flush(IF_flush_id), // if need to insert bubble or fulsh, zero all (?)
        .clock(clk),
        .Din({pcplus4_if, instruction_if}),
```

```
        .Dout({pcplus4_id, instruction_id})
    );

    // in ID STAGE
        // wires
    assign op_id = instruction_id[31:26];
    assign rs_id = instruction_id[25:21];
    assign rt_id = instruction_id[20:16];
    assign rd_id = instruction_id[15:11];
    assign immi_id = instruction_id[15:0];
    assign jump_inst_id = instruction_id[25:0];


    RegFile M4(
        .clk(clk),
        .reg_write(regwrite_wb),
        .read_reg1(rs_id),
        .read_reg2(rt_id),
        .write_reg(registerrd_wb),
        .Input_Readreg(Input_Readreg),
        .write_data(writeback_wb),
        .read_data1(regdata1_id),
        .read_data2(regdata2_id),
        .RegOut(RegOut)
    );
    MUX4_2 firstComparatorMUX(
        .control(forwardingeqin1),
        .in1(regdata1_id),
        .in2(writeback_wb),
        .in3(ALUresult_mem),
        .out(eq_mux1_id)
    );

   MUX4_2 secondComparatorMUX(
        .control(forwardingeqin2),
        .in1(regdata2_id),
        .in2(writeback_wb),
        .in3(ALUresult_mem),
        .out(eq_mux2_id)
    );
        // in top.v
    comparator com_id_(
        .reset(reset),
        .I1(eq_mux1_id),
        .I2(eq_mux2_id),
        .res(com_res_id)
    );
    control control_(
        .inst(op_id),
        .equal(com_res_id),
        .IFflush(IF_flush_id),
        .regdst(regdst_id),
        .jump(jump),
        .branch(beq_id), // only beq, no bne
        .memread(memread_id),
        .memtoreg(memtoreg_id),
        .aluop(ALUop_id),
        .memwrite(memwrite_id),
```

```
        .alusrc(ALUsrc_id),
        .regwrite(regwrite_id)
    );
    SignExtend signextend_(
        .imm_in(immi_id),
        .out(imm_signEx_id)
    );
    assign branchaddress_id = pcplus4_id+(imm_signEx_id<<2);
    assign jumpaddress_id = {pcplus4_id[31:28], jump_inst_id, 2'b00};
            // for IF
    assign branchjump_if = (beq_id & com_res_id) || jump;
    assign branchjumpaddress_if = (beq_id) ? branchaddress_id : jumpaddress_id;
    register #(120) ID_EX(
        .write(1'b1),
        .flush(stall | ~pc_write_2 | ~(pc_write_3_id & pc_write_3_ex)),
        .clock(clk),
        .Din({regdst_id, ALUsrc_id, regwrite_id, memtoreg_id, memwrite_id,
memread_id, ALUop_id, ALUsrc_id,
           regdata1_id, regdata2_id, imm_signEx_id, rs_id, rt_id, rd_id}),
        .Dout({regdst_ex, ALUsrc_ex, regwrite_ex, memtoreg_ex, memwrite_ex,
memread_ex, ALUop_ex, ALUsrc_ex,
            regdata1final_ex, regdata2final_ex, immi_signEx_ex, rs_ex, rt_ex,
rd_ex})
    );
    register #(1) ID_EX_pc_write(
        .write(1'b1),
        .flush(1'b0),
        .clock(clk),
        .Din(pc_write_3_id),
        .Dout(pc_write_3_ex)
    );
    // in EX STAGE
    assign registerrd_ex = (regdst_ex) ? rd_ex : rt_ex; // bottom MUX for Rt/Rd
    assign func_ex = immi_signEx_ex[5:0]; // used for ALU control
    assign ALUin2final_ex = (ALUsrc_ex) ? immi_signEx_ex : ALUin2_ex;

    MUX4_2 firstALUMUX(
        .control(forwardingALUin1),
        .in1(regdata1final_ex),
        .in2(writeback_wb),
        .in3(ALUresult_mem),
        .out(ALUin1final_ex)
    );
    MUX4_2 secondALUMUX(
        .control(forwardingALUin2),
        .in1(regdata2final_ex),
        .in2(writeback_wb),
        .in3(ALUresult_mem),
        .out(ALUin2_ex)
    );
    ALUControl ALUcontrol_( // assume 2'b11 as and
        .func(func_ex),
        .op(ALUop_ex),
        .ctrl(ALUctrl_ex)
    );
    ALU ALU_(
        .ALU_control(ALUctrl_ex),
        .a(ALUin1final_ex),
```

```
        .b(ALUin2final_ex),
        .zero(exzero),
        .result(ALUresult_ex)
    );
    // EX/MEM Reg
        // Din={forwardB=regdata2final_ex, aluresult=ALUresult_ex,
dst=registerrd_ex, memread=memread_ex, memtoreg=memtoreg_ex, memwrite, regwrite}
    register #(73) EX_MEM(
        .write(1'b1),
        .flush(1'b0),
        .clock(clk),
        .Din({registerrd_ex, ALUin2_ex, ALUresult_ex, memread_ex,memwrite_ex,
memtoreg_ex, regwrite_ex}),
        .Dout({registerrd_mem, regdata2final_mem, ALUresult_mem, memread_mem,
memwrite_mem, memtoreg_mem, regwrite_mem})
    );
    // in MEM STAGE
    assign writedata_mem = (memSrc) ? writeback_wb : regdata2final_mem;
    DataMemory datamem_(
        .clk(clk),
        .mem_read(memread_mem),
        .mem_write(memwrite_mem),
        .address(ALUresult_mem),
        .data_write(writedata_mem),
        .data_read(dataread_mem)
    );
    // MEM/WB Reg
    register #(71) MEM_WB(
        .write(1'b1),
        .flush(1'b0),
        .clock(clk),
        .Din({dataread_mem, ALUresult_mem, memtoreg_mem, regwrite_mem,
registerrd_mem}),
        .Dout({memdata_wb, ALUresult_wb, memtoreg_wb, regwrite_wb,
registerrd_wb})
    );
    // in WB STAGE
    assign writeback_wb = (memtoreg_wb) ? memdata_wb : ALUresult_wb;
    // Forwarding
    ForwardingUnit forward_(
        .IDEXRs(rs_ex),
        .IDEXRt(rt_ex),
        .IDEXRst(registerrd_ex),
        .EXMEMDst(registerrd_mem),
        .MEMWBDst(registerrd_wb),
        .IFIDRs(rs_id),
        .IFIDRt(rt_id),


        .MEMWBMemRead(memread_wb),
        .MEMWBRegWrite(regwrite_wb),
        .EXMEMRegWrite(regwrite_mem),
        .EXMEMMemWrite(memwrite_mem),
        .IFIDBeq(beq_id),

        .AluA(forwardingALUin1),
        .AluB(forwardingALUin2),
        .EqA(forwardingeqin1),
```

```
        .EqB(forwardingeqin2),
        .MemSrc(memSrc)

    );
    //Hazard of lw, in "top.v"
    hazarddetection hazard_(
        .reset(reset),
        .rs_id(rs_id),
        .rt_id(rt_id),
        .rt_ex(rt_ex),
        .rd_ex(rd_ex),
        .regWrite_ex(regwrite_ex),
        .IDEXmemread(memread_ex),
        .IFIDwrite(if_id_write_hazard),
        .PCwrite(pc_write_hazard),
        .hazard(stall), // hazard=1 when hazard,
        .PCwrite2(pc_write_2), // pc__write2=0,if beq data hazard
        .PCwrite3(pc_write_3_id),
        .beq_id(beq_id)
    );

     assign PCout = pc_out_if;
endmodule
```

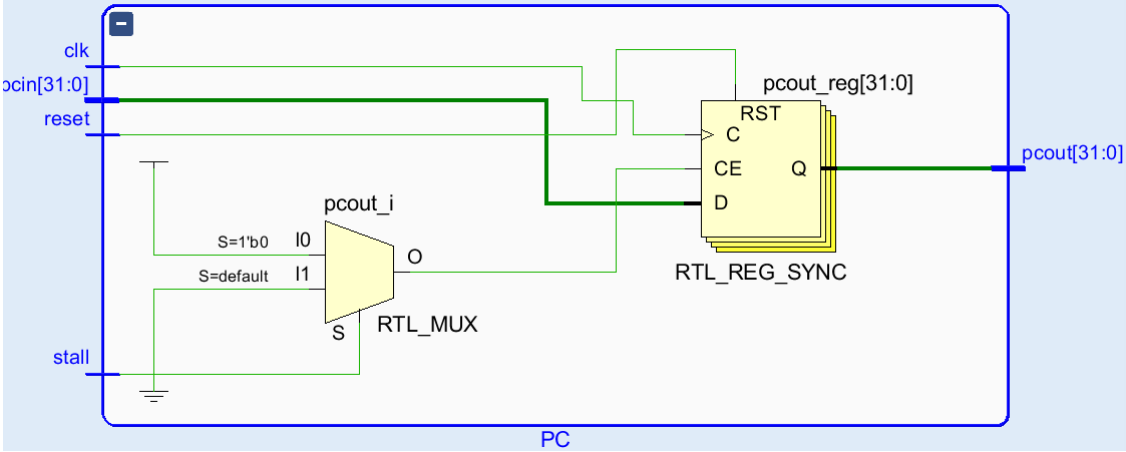
2.3 Detailed Design for Each Components

PC

Signal Name	Direction	Description
reset	Input	Reset signal used for initialization
clk	Input	Clock signal used to control the change of PC
stall	Input	Stall signal used to decide whether we should stop to update PC in order to add NOP instructions.
pcin	Input	Input of PC
pcout	Output	Output of PC

```
module PC(
    input       reset, clk,stall,
    input       [31:0]  pcin,
    output reg  [31:0]  pcout
    );

    always @ (posedge clk) begin
        if (reset)
            pcout <= 32'b0;
        else if (!stall)
            pcout <= pcin;
    end
```

```
endmodule
```

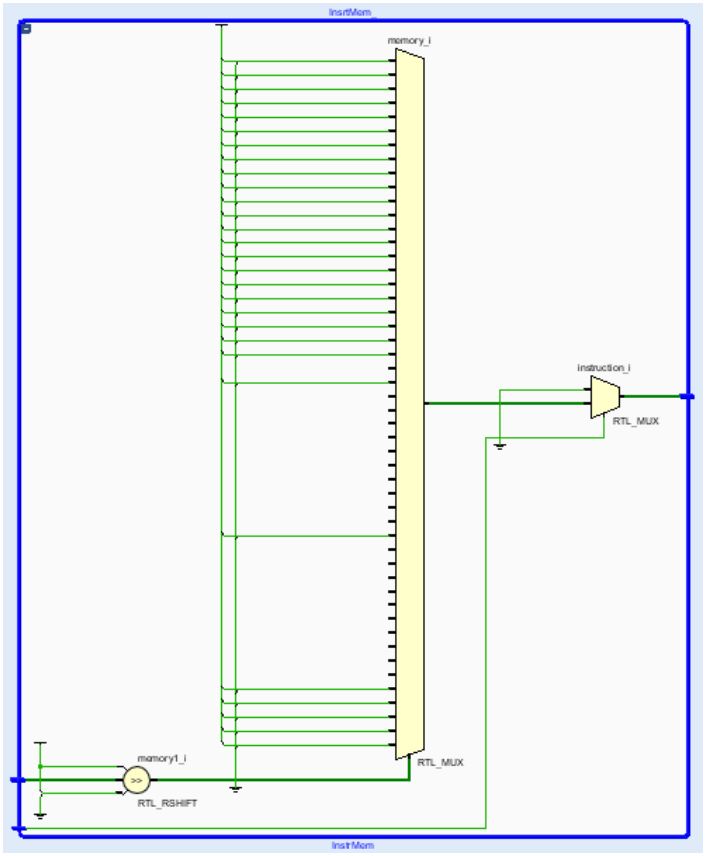


Instruction Memory

Signal Name	Direction	Description
reset	Input	Reset signal used for initialization
address	Input	Input of instruction memory
instruction	Output	Output of instruction memory

```
module InstrMem(
    input       reset,
    input       [31:0]  address,
    output      [31:0]  instruction
    );
    wire [31:0] memory [0:49];
//    reg [7:0] temp[199:0];
//    integer i;
//    initial begin
//        for (i = 0; i < 128; i = i + 1)
//           memory[i] = 32'b0;
//        $readmemb("C:/Users/Administrator/Desktop/ve370-
p2/InstructionMem_for_P2_Demo_bonus.txt", temp);
//        for (i = 0; i < 128; i=i+1) begin
//            memory[i][31:24] = temp[4*i];
//            memory[i][23:16] = temp[4*i+1];
//            memory[i][15:8] = temp[4*i+2];
//            memory[i][7:0] = temp[4*i+3];
//        end
//    end


assign memory[0] = 32'b00100000000010000000000000100000; //addi $t0, $zero, 0x20
assign memory[1] = 32'b00100000000010010000000000110111; //addi $t1, $zero, 0x37
assign memory[2] = 32'b00000001000010011000000000100100; //and $s0, $t0, $t1
assign memory[3] = 32'b00000001000010011000000000100101; //or $s0, $t0, $t1
assign memory[4] = 32'b10101100000010000000000000000100; //sw $s0, 4($zero)
assign memory[5] = 32'b10101100000010000000000000001000; //sw $t0, 8($zero)
assign memory[6] = 32'b00000001000010011000100000100000; //add $s1, $t0, $t1
assign memory[7] = 32'b00000001000010011001000000100010; //sub $s2, $t0, $t1
```

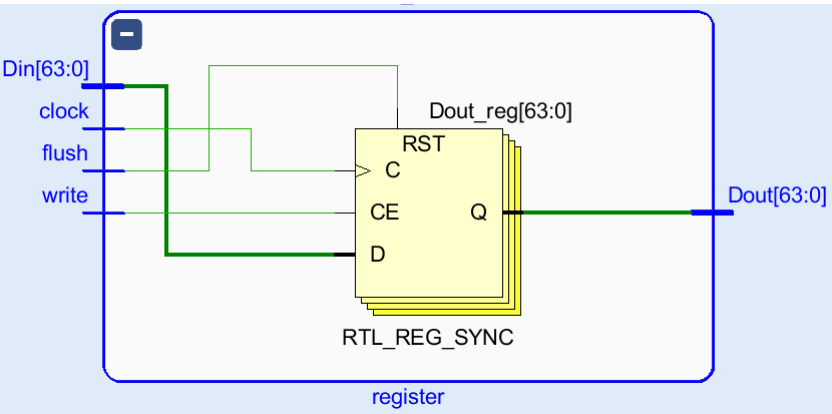
```
assign memory[8] = 32'b00010010001100100000000000001001; //beq $s1, $s2, error0
assign memory[9] = 32'b10001100000100010000000000000100; //lw $s1, 4($zero)
assign memory[10] = 32'b00110010001100100000000001001000; //andi $s2, $s1, 0x48
assign memory[11] = 32'b00010010001100100000000000001001; //beq $s1, $s2, error1
assign memory[12] = 32'b10001100000100110000000000001000;//lw $s3, 8($zero)
assign memory[13] = 32'b00010010000100110000000000001010; //beq $s0, $s3, error2
assign memory[14] = 32'b00000001001010001101000000101010; //slt $s4, $s2, $s1
(Last)
assign memory[15] = 32'b00010010100000000000000000001111; //beq $s4, $0, EXIT
assign memory[16] = 32'b00000010001000001001000000100000; //add $s2, $s1, $0
assign memory[17] = 32'b00001000000000000000000000001110; //j Last
assign memory[18] = 32'b00100000000010000000000000000000; //addi $t0, $0,
0(error0)
assign memory[19] = 32'b00100000000010010000000000000000; //addi $t1, $0, 0
assign memory[20] = 32'b00001000000000000000000000011111; //j EXIT
assign memory[21] = 32'b00100000000010000000000000000001; //addi $t0, $0,
1(error1)
assign memory[22] = 32'b00100000000010010000000000000001; //addi $t1, $0, 1
assign memory[23] = 32'b00001000000000000000000000011111; //j EXIT
assign memory[24] = 32'b00100000000010000000000000000010; //addi $t0, $0,
2(error2)
assign memory[25] = 32'b00100000000010010000000000000010; //addi $t1, $0, 2
assign memory[26] = 32'b00001000000000000000000000011111; //j EXIT
assign memory[27] = 32'b00100000000010000000000000000011; //addi $t0, $0,
3(error3)
assign memory[28] = 32'b00100000000010010000000000000011; //addi $t1, $0, 3
assign memory[29] = 32'b00001000000000000000000000011111; //j EXIT


    assign instruction = (reset == 1 ) ? 32'b0 : memory[address >> 2];
endmodule
```



Registers

Signal Name	Direction	Description
write	Input	Write signal used to decide whether to update the registers
clock	Input	Clock signal used to control the change of registers
flush	Input	Flush signal used to decide whether we should set all values in registers to be zero in order to add bubbles.
Din	Input	Input of register
Dout	Output	Output of register

```
module register(write, flush, clock, Din, Dout);
parameter N = 32;
input write, flush, clock;
input [N-1:0] Din;
output [N-1:0] Dout;
reg [N-1:0] Dout;
initial begin
Dout <= 0;
end

always @ (posedge clock)begin
if (flush == 1) Dout <= 0;
else if (write == 0) Dout <= Dout;
else Dout <= Din;
end
endmodule
```

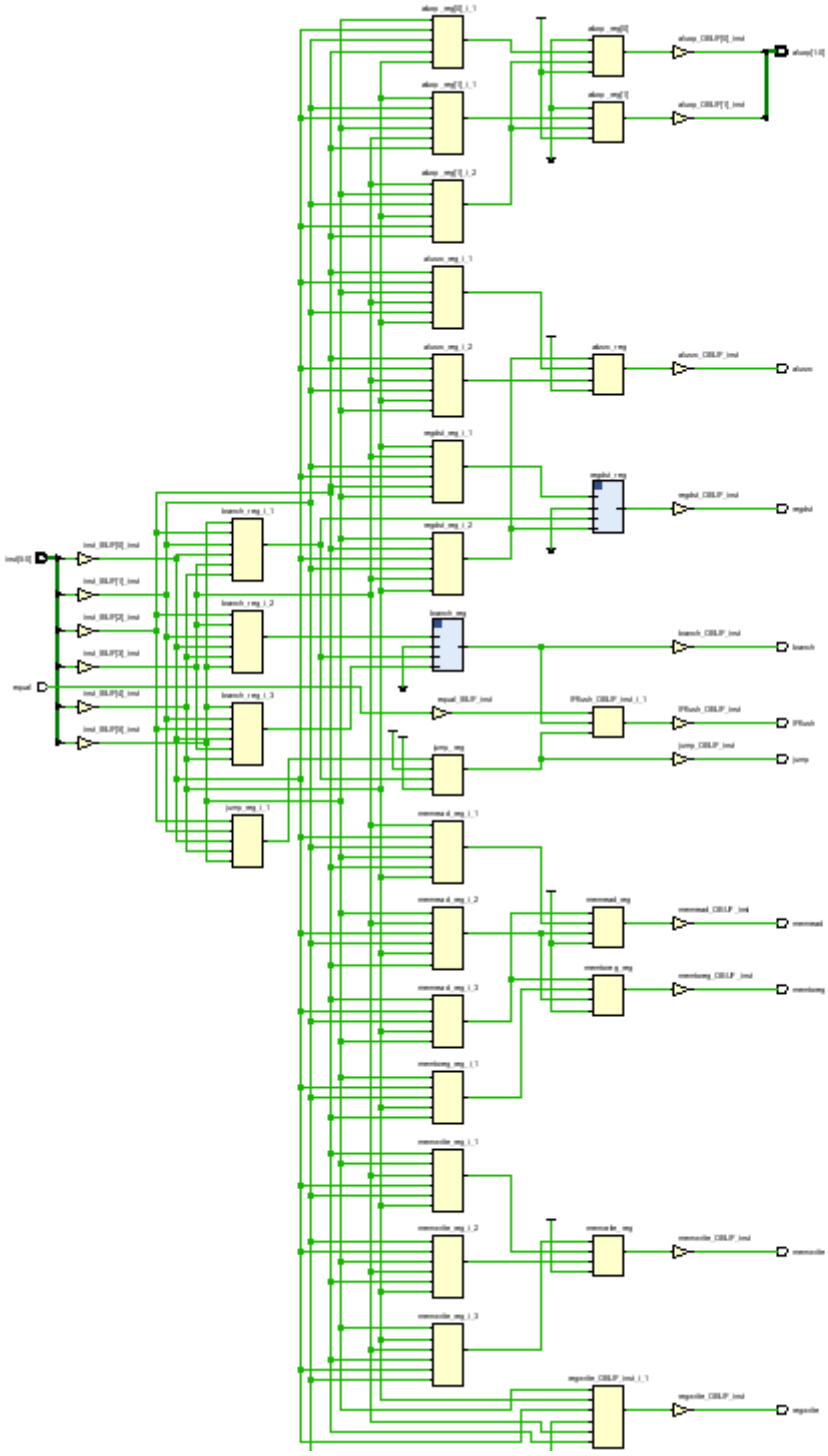


Control

Signal Name	Direction	Description
inst	Input	Operation signal used to decide the category of the instruction
equal	Input	Signal used to check whether two data from registers file are equal
IFflush	Output	Flush signal used to decide whether we should set all values in IF/ID registers to be zero in order to add bubbles.
regdst	Output	Decide the destination of write register
jump	Output	Show whether it is a jump instruction
branch	Output	Show whether it is a branch instruction
memread	Output	Show whether we need to read from data memory
memtoreg	Output	Show whether we should write back the data from data memory
aluop	Output	Control signal for ALU control
memwrite	Output	Show whether we need to write data memory
alusrc	Output	Decide the second input of ALU
regwrite	Output	Show whether we need to write back to register files

```
module control(inst, equal, IFflush, regdst, jump, branch, memread, memtoreg,
aluop, memwrite, alusrc, regwrite);
input [5:0] inst;
input equal; // output of module comparator
output IFflush;
output regdst, jump, branch, memread, memtoreg, memwrite, alusrc, regwrite;
output [1:0] aluop;
reg regdst, jump, branch, memread, memtoreg, memwrite, alusrc;
reg [1:0] aluop;
initial begin
branch = 0;
jump = 0;
memread = 0;
memtoreg = 0;
memwrite = 0;
alusrc = 0;
end
always @ (inst)begin
// r-type
if (inst == 0) begin regdst = 1; jump = 0; branch = 0; memread = 0; memtoreg =
0; memwrite = 0; alusrc = 0; aluop = 2; end
// addi
else if (inst == 8) begin regdst = 0; jump = 0; branch = 0; memread = 0;
memtoreg = 0; memwrite = 0; alusrc = 1; aluop = 0;end
// lw
else if (inst == 35) begin regdst = 0; jump = 0; branch = 0; memread = 1;
memtoreg = 1; memwrite = 0; alusrc = 1; aluop = 0; end
// sw
```

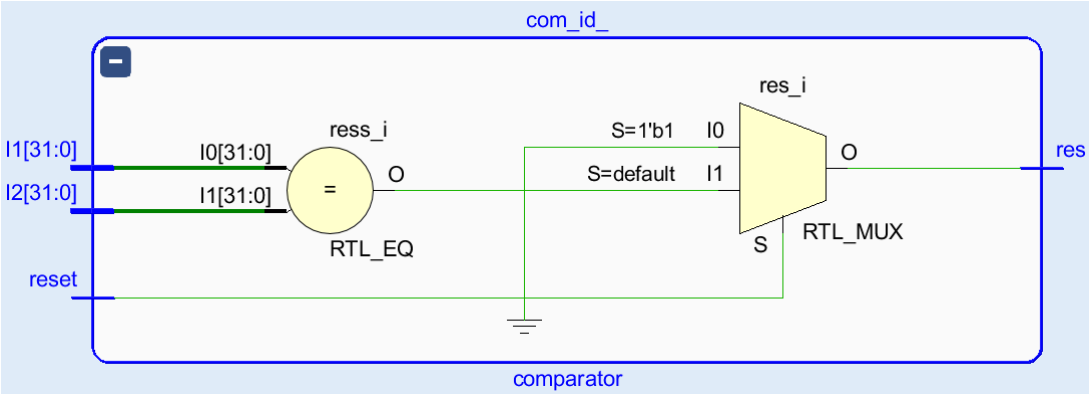
```
else if (inst == 43) begin regdst = 0; jump = 0; branch = 0; memread = 0;
memtoreg = 1; memwrite = 1; alusrc = 1; aluop = 0; end
// andi
else if (inst == 12) begin regdst = 0; jump = 0; branch = 0; memread = 0;
memtoreg = 0; memwrite = 0; alusrc = 1; aluop = 3; end
// beq
else if (inst == 4) begin regdst = 0; jump = 0; branch = 1; memread = 0;
memtoreg = 0; memwrite = 0; alusrc = 0; aluop = 1; end
// jump
else if (inst == 2) begin regdst = 0; jump = 1; branch = 0; memread = 0;
memtoreg = 0; memwrite = 0; alusrc = 0; aluop = 1; end
end
assign regwrite = (inst == 0) || (inst == 8) || (inst == 35) || (inst == 12) ? 1
: 0;
assign IFflush = ((branch == 1) && (equal == 1)) || (jump == 1) ? 1 : 0;
endmodule
```



Comparator

Signal Name	Direction	Description
reset	Input	Reset signal used for initialization
I1	Input	first data
I2	Input	second data
res	Output	Output to show whether two inputs are equal

```
module comparator(reset, I1, I2, res);
input [31:0] I1, I2;
input reset;
output res;
wire ress;
assign ress = (I1 == I2) ? 1 : 0;
assign res = reset ? 0 : ress;
endmodule
```

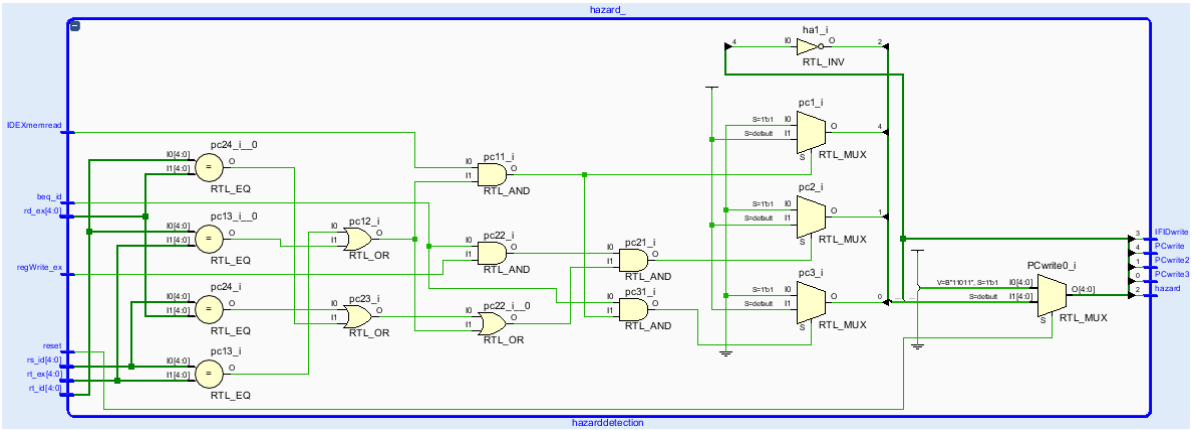


Hazard Detection Unit

Signal Name	Direction	Description
reset	Input	Reset signal used for initialization
rs_id	Input	The address of Rs in id stage
rt_id	Input	The address of Rt in id stage
rt_ex	Input	The address of Rt in ex stage
rd_ex	Input	The address of Rd in ex stage
regWrite_ex	Input	The address of write register in ex stage
IDEXmemread	Input	Memread signal in ex stage
beq_id	Input	Beq signal in id stage
PCwrite	Output	Control signal to decide whether to update PC
PCwrite2	Output	Control signal to decide whether to update PC
PCwrite3	Output	Control signal to decide whether to update PC
IFIDwrite	Output	Control signal to decide whether to update IFID register
hazard	Output	Show whether there is a hazard

```
module hazarddetection(reset, rs_id, rt_id, rt_ex, rd_ex, regWrite_ex,
IDEXmemread, PCwrite, IFIDwrite, hazard, PCwrite2, beq_id, PCwrite3);
input [4:0] rs_id, rt_id, rt_ex, rd_ex;
input IDEXmemread, reset, regWrite_ex, beq_id;
output PCwrite, IFIDwrite, hazard, PCwrite2, PCwrite3;
wire pc1, if1, ha1,pc2, pc3;
// pc2 used for beq data hazard, one previous R-type
```

```
// pc2 for beq data hazard, with lw

assign pc1 = ((IDEXmemread == 1) && ((rs_id == rt_ex) || (rt_id == rt_ex))) ? 0
: 1;
assign if1 = PCwrite;
assign ha1 = ~PCwrite;

assign pc2 = (beq_id && (regWrite_ex == 1) && ( ((rs_id == rd_ex) || (rt_id ==
rd_ex)) || ((rs_id == rt_ex) || (rt_id == rt_ex)))) ? 0 : 1;
assign pc3 = ( beq_id && ((IDEXmemread == 1) && ((rs_id == rt_ex) || (rt_id ==
rt_ex))) ) ? 0 : 1;
assign {PCwrite, IFIDwrite, hazard, PCwrite2, PCwrite3} = (reset == 1) ?
5'b11011 : {pc1, if1, ha1, pc2, pc3};
endmodule
```

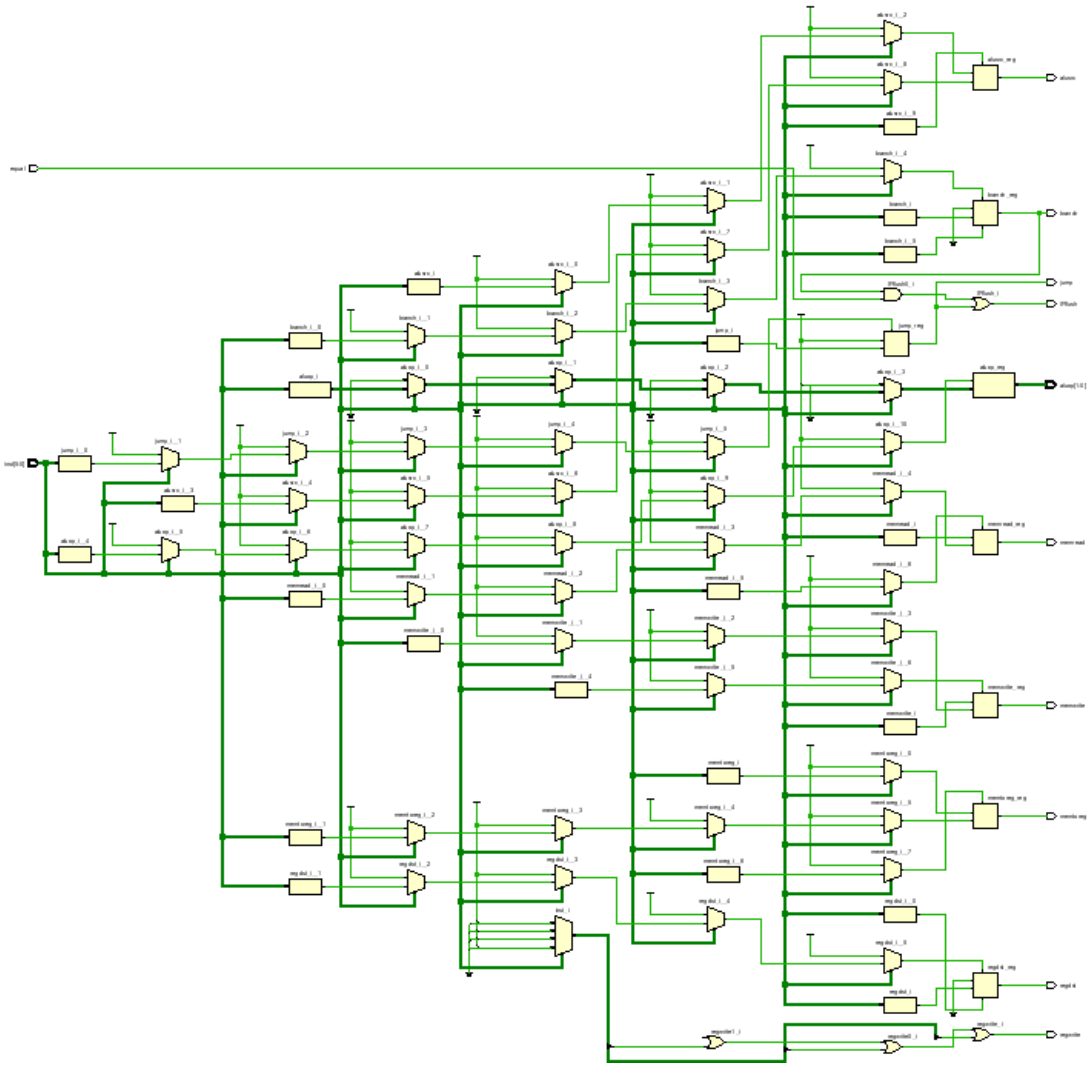


Register Files

Signal Name	Direction	Description
clk	Input	Clock signal used to control Regfile
reg_write	Input	Signal used to show whether we update the Regfiles
read_reg1	Input	Address of first reading data from Regfiles
read_reg2	Input	Address of second reading data from Regfiles
write_reg	Input	Address of reg to be written
Input_Readreg	Input	Address of reg that we want to check its value
write_data	Input	The data that write back to Regfiles
read_data1	Output	First reading data
read_data2	Output	Second reading data
RegOut	Output	Value of reg that we want to check its value

```
module RegFile(
    input clk, reg_write,
    input [4:0] read_reg1, read_reg2, write_reg, Input_Readreg,
    input [31:0] write_data,
    output [31:0] read_data1, read_data2, RegOut
    );
```

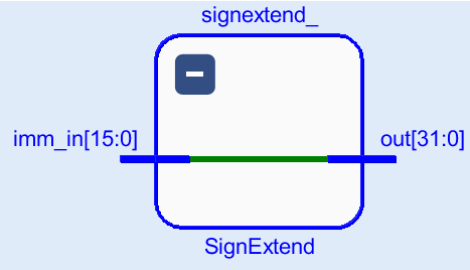
```
    reg [31:0] file [0:31];
    integer i;

    initial begin
        for (i = 0; i < 32; i = i + 1)
            file[i] = 32'b0;  // initialize the registers
    end

    assign read_data1 = file[read_reg1];
    assign read_data2 = file[read_reg2];
    assign RegOut = file[Input_Readreg];
    always @(negedge clk) begin
        if (reg_write == 1)
            file[write_reg] <= write_data;
    end
endmodule
```



Sign Extend Block

Signal Name	Direction	Description
imm_in	Input	Input of SignExtended
out	Output	Output of SignExtended

```
module SignExtend(
    input [15:0] imm_in,
    output [31:0] out
    );
    assign out[15:0] = imm_in[15:0];
    assign out[31:16] = imm_in[15] ? 16'b1111111111111111 : 16'b0;
endmodule
```

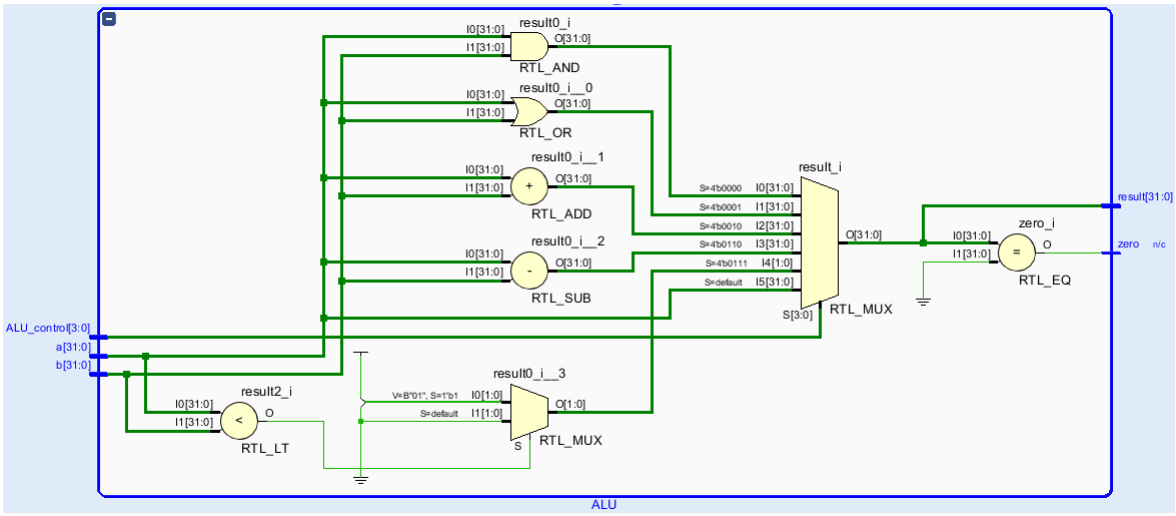


ALU

Signal Name	Direction	Description
ALU_control	Input	Control signal that decide the operation of ALU
a	Input	First input data
b	Input	Second input data
zero	Output	Show whether the operation result is zero
result	Output	Operation result

```
module ALU(
    input       [3:0]   ALU_control,
    input       [31:0]  a, b,
    output       zero,
    output reg  [31:0]  result
    );

    assign zero = (result == 0);

    initial begin
        result = 32'b0;  //to initialize the result
    end

    always @ (a or b or ALU_control) begin
        case (ALU_control)
            4'b0000:
                result = a & b;  // and
            4'b0001:
                result = a | b;  // or
            4'b0010:
                result = a + b;  // add
            4'b0110:
                result = a - b;  // sub
            4'b0111:
                result = (a < b) ? 1:0;  // slt
            default:
```

```
                result = a;
        endcase
    end
endmodule
```

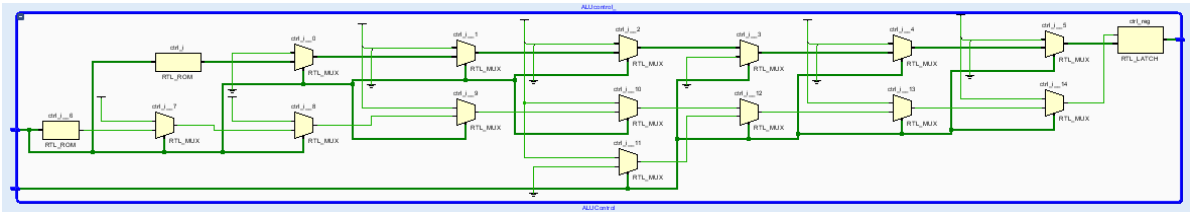


ALU Control

Signal Name	Direction	Description
funct	Input	The category of instruction
op	Input	More detailed information
ctrl	Output	Control signal for ALU

```
module ALUControl(
    input [5:0] func, // function
    input [1:0] op,   // operation
    output reg [3:0] ctrl
    );
    always @(func or op) begin
        if (op == 2'b00)
            ctrl = 4'b0010; //add
        else if (op == 2'b01)
            ctrl = 4'b0110; //minus
        else if (op == 2'b10)
            begin
                if (func == 6'b100000)
                    ctrl = 4'b0010;
                else if (func == 6'b100010)
                    ctrl = 4'b0110;
                else if (func == 6'b100100)
                    ctrl = 4'b0000;
                else if (func == 6'b100101)
                    ctrl = 4'b0001;
                else if (func == 6'b101010)
                    ctrl = 4'b0111;
            end
        else if (op == 2'b11) //and
            ctrl = 4'b0000;
    end
```

```
endmodule
```

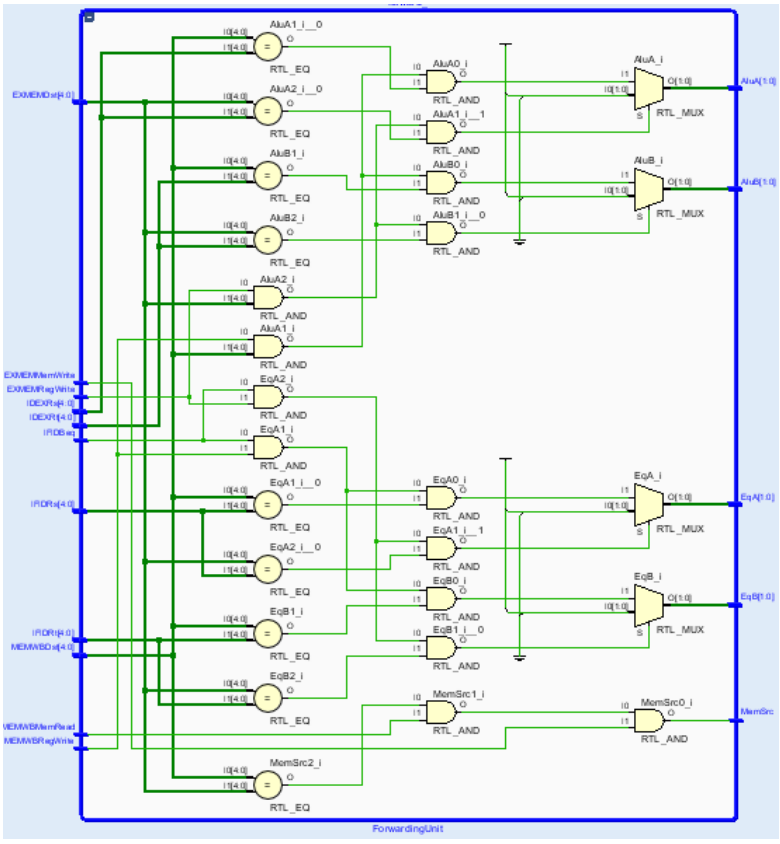


Forwarding Unit

Signal Name	Direction	Description
IDEXRs	Input	Rs in ID/EX Register
IDEXRt	Input	Rt in ID/EX Register
IDEXRst	Input	Rst in ID/EX Register
EXMEMDst	Input	Rst in EX/MEM Register
MEMWBDst	Input	Rst in MEM/WB Register
IFIDRs	Input	Rs in IF/ID Register
IFIDRt	Input	Rt in IF/ID Register
MEMWBMemRead	Input	MemRead in MEM/WB Register
MEMWBRegWrite	Input	RegWrite in MEM/WB Register
EXMEMRegWrite	Input	RegWrite in EX/MEM Register
EXMEMMemWrite	Input	MemWrite in EX/MEM Register
IFIDBeq	Input	Beq in IF/ID Register
MemSrc	Output	Help to forward the lw data to following sw instruction
AluA	Output	Control signal for mux of first input signal of ALU
AluB	Output	Control signal for mux of second input signal of ALU
EqA	Output	Control signal for mux of first input signal of comparator
EqB	Output	Control signal for mux of second input signal of comparator

```
module ForwardingUnit(
    input[4:0] IDEXRs,IDEXRt,IDEXRst,EXMEMDst,MEMWBDst,IFIDRs,IFIDRt,
    input MEMWBMemRead,MEMWBRegWrite,EXMEMRegWrite,EXMEMMemWrite,IFIDBeq,
    output reg MemSrc,
    output reg[1:0] AluA,AluB,EqA,EqB
        );

    initial begin
        AluA = 2'b00;
        AluB = 2'b00;
```

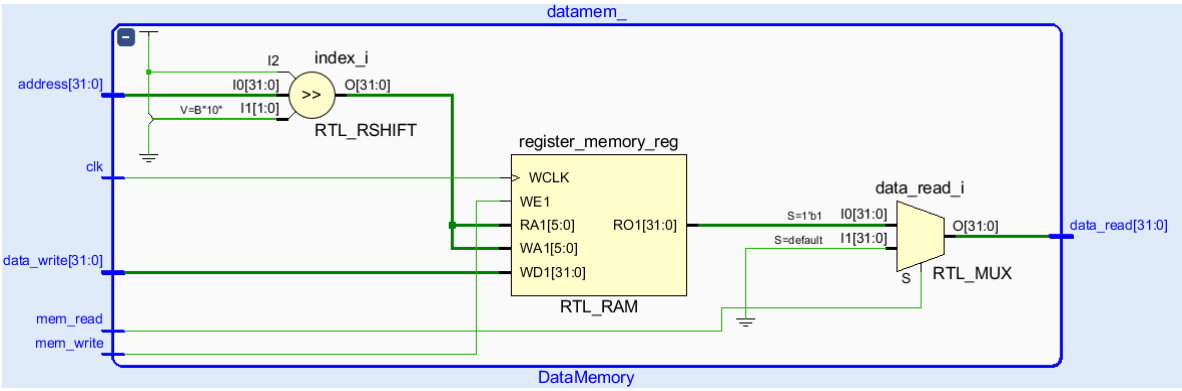
```
        EqA = 2'b00;
        EqB = 2'b00;
        MemSrc = 1'b0;
    end

    always @(*) begin
        if(EXMEMRegWrite && EXMEMDst && EXMEMDst == IDEXRs) AluA = 2'b10;
        else if(MEMWBRegWrite && MEMWBDst && MEMWBDst == IDEXRs) AluA = 2'b01;
        else AluA = 2'b00;

        if(EXMEMRegWrite && EXMEMDst && EXMEMDst == IDEXRt) AluB = 2'b10;
        else if(MEMWBRegWrite && MEMWBDst && MEMWBDst == IDEXRt) AluB = 2'b01;
        else AluB = 2'b00;

        if(MEMWBDst == EXMEMDst && MEMWBMemRead && EXMEMMemWrite) MemSrc = 1'b1;
        else MemSrc = 1'b0;

        if((IFIDBeq) && EXMEMRegWrite && EXMEMDst == IFIDRs) EqA = 2'b10;
        else if((IFIDBeq) && MEMWBRegWrite && MEMWBDst == IFIDRs) EqA = 2'b01;
        else EqA = 2'b00;

        if((IFIDBeq) && EXMEMRegWrite && EXMEMDst == IFIDRt) EqB = 2'b10;
        else if((IFIDBeq) && MEMWBRegWrite && MEMWBDst == IFIDRt) EqB = 2'b01;
        else EqB = 2'b00;

    end
endmodule
```



Data Memory

Signal Name	Direction	Description
mem_read	Input	Control signal that decide whether we should read from data memory
mem_write	Input	Control signal that decide whether we should write to data memory
clk	Input	Clock signal used to control the change of data memory
address	Input	The reg address that we want to operate on
data_write	Input	The data that we want to write
data_read	Output	Read data from data memory

```
module DataMemory(
    input       mem_read, mem_write, clk,
    input       [31:0]  address, data_write,
    output      [31:0]  data_read
    );
    parameter        size = 64;
    wire     [31:0]  index;
    reg      [31:0]  register_memory [0:size-1];
    integer i;

    assign index = address >> 2;
    initial begin
        for (i = 0; i < size; i = i + 1)
            register_memory[i] = 32'b0;  // initialize the memory
    end

    always @ (negedge clk) begin
        if (mem_write == 1'b1) begin
            register_memory[index] = data_write;  //write data one by one
        end
    end

    assign data_read = (mem_read == 1'b1) ? register_memory[index]:32'b0;
endmodule
```



3 FPGA implementation: SSD and Top Module Design

In this part, we will introduce the design of our SSD and how we print the results onto the FPGA board.

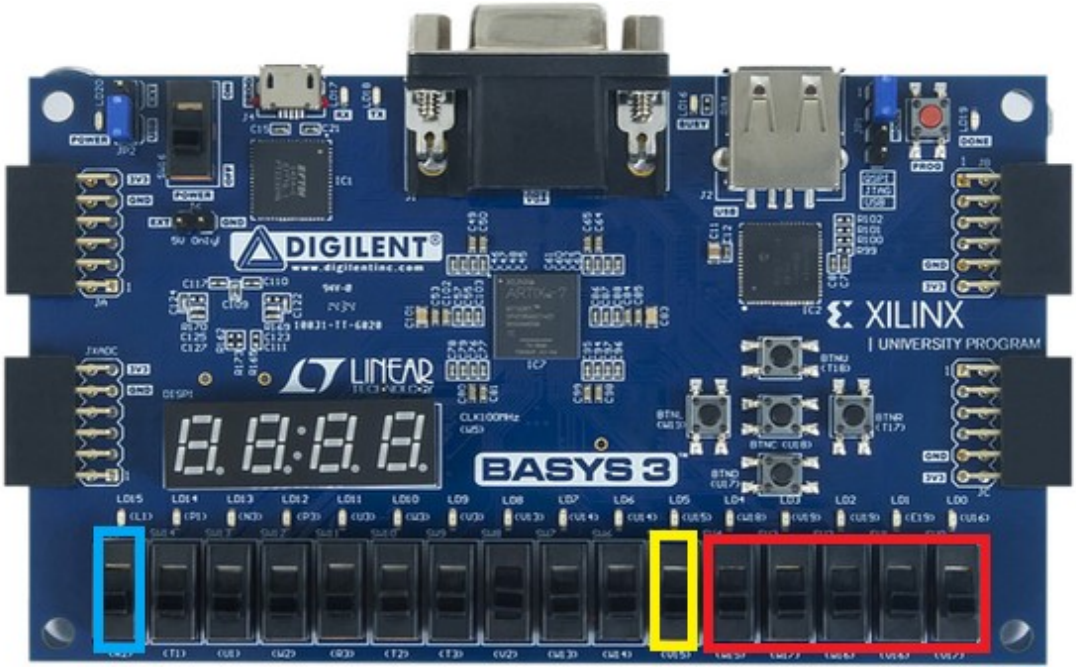


Figure X: A brief view of the FPGA board we use

The picture above is the FPGA board we used in this project. To display adequate and appropriate information, the highlighted switches are used.

- The blue-framed switch represents the input signal "clk" of the pipeline processor module.
- The five red-framed switches determines which register we want to read. They represent the 5-bit input signal "Input_Readreg" of the pipelined processor module.
- The yellow-framed switch determines whether SSD displays the address of PC or the value in certain register. It represents the "Input_ReadPC" signal of the pipeline processor module. When it is turned on, SSD will print the PC value regardless of the condition of red-framed switches. When it is turned off, SSD will print value stored in chosen register.

Signal Name	Direction	Description
Input_Readreg	Input	Control signal that decide which register to read
Input_ReadPC	Input	Control signal that decide whether to print PC value
clk2	Input	FPGA original clock signal
reset	Input	Reset signal used for initialization
SSDout	Output	Control signal for 7 segments of a single SSD digit
SSDdigit	Output	Control signal for 4 SSD digits

```
module Demo(Input_Readreg, Input_ReadPC, switch, clk2, reset, SSDout, SSDdigit);
input [4:0] Input_Readreg;
input Input_ReadPC;
```

```
input switch;
input SSDoutupper;
input clk2;
input reset;
output [6:0] SSDout;
output [3:0] SSDdigit;
wire A0, A1, A2, A3;
wire useless;
wire [31:0] pcout, regout;
wire [31:0] out;
PipelinedProcessor_bonus M1 (switch, Input_Readreg, reset, useless, pcout,
regout);
assign out = (Input_ReadPC == 1'b1) ? pcout : regout;
ttop M2 (clk2, reset, switch, out[15:0], SSDout, A0, A1, A2, A3);
assign SSDdigit = {A0, A1, A2, A3};
endmodule
```

3.1 Components of Top module design

The idea of SSD displayer is to update every digit one-by-one dynamically at a relatively high frequency. So the top module is made of three components. They are clock divider, ring counter, and SSD.

3.1.1 Clock divider

Clock divider take the and generate a new clock cycle signal of appropriate frequency and send it to the ring counter.

Signal Name	Direction	Description
reset	Input	reset signal used for initialization
clock	Input	Clock cycle signal of the original FPGA board
clock_output	Output	Clock signal used in ring counter

```
module divider_1 (reset, clock, clock_out);
parameter N = 50000000;
input clock, reset;
output clock_out;
reg [33:0] Q;
reg clock_out;
always @ (posedge clock) begin
if (reset == 1'b1)
begin
Q = 0;
clock_out = 0;
end
else if (Q == N-1) begin
Q = 0;
clock_out = ~clock_out;
```

```
end
else begin
Q = Q + 1;
end
end
endmodule
```

3.1.2 Ring Counter

The ring counter and the clock divider together help display the number in a stable format. Ring counter itself update every digit one-by-one dynamically at a relatively high frequency. It contains a counter which generates a periodic signal counting from 0 to 3, and a decoder to transfer this signal to the 4-bit control signal, each bit corresponds to an SSD digit.

Signal Name	Direction	Description
I	input(decoder)	Periodic signal counting from 0 to 3
D	output(decoder)	Signal transferring to ring counter
reset	input(counter)	Reset signal used for initialization
clock	input(counter)	Clock signal generated by clock divider
Q	output(counter)	Signal transferring to decoder
clock	input(ring counter)	Clock signal generated by clock divider
reset	input(ring counter)	Reset signal used for initialization
P	output(ring counter)	Signal controlling 4 SSD digits

```
module Decoder_2_4in (I, D);
input [1:0] I;
output [3:0] D;
reg [3:0] D;
always @ (I) begin
case (I)
2'b00: D=4'b1110;
2'b01: D=4'b1101;
2'b10: D=4'b1011;
2'b11: D=4'b0111;
default D=4'b1111;
endcase
end
endmodule

module counter_2_bit (reset, clock, Q);
input reset, clock;
output [1:0] Q;
reg [1:0] Q;
always @ (posedge reset or posedge clock) begin
```

```
if (reset == 1'b1) Q=0;
else Q=Q+1;
end
endmodule

module ring_counter(clock, reset, P);
input reset, clock;
output [3:0] P;
wire [1:0] Q;
counter_2_bit bb(reset, clock, Q);
Decoder_2_4in cc(Q, P);
endmodule
```

3.1.3 SSD & Synthesized Top Module

The component SSD outputs a 7-bit output signal to control the display number of SSD.
The top module combines the three components together, thus we succeed to print our result of pipeline processor onto the FPGA board.

Signal Name	Direction	Description
clk	input	Clock cycle signal of the original FPGA board
reset	input	Reset signal used for initialization
shuzi	input	Number expected to be printed on SSD
SSD	output	Control signal of seven segment displaying unit
A0-A3	outputs	Control signal of each 4 SSD digits

```
module ttop(clk, reset, shuzi, SSD, A0, A1, A2, A3);
input clk, reset;
input [15:0] shuzi;
output [6:0] SSD;
output A0, A1, A2, A3;
wire [3:0] A;
wire [3:0] sh3, sh2, sh1, sh0;
wire clcc, clout;
assign A = {A0, A1, A2, A3};
assign sh3 = shuzi[15:12];
assign sh2 = shuzi[11:8];
assign sh1 = shuzi[7:4];
assign sh0 = shuzi[3:0];
reg [3:0] sh;
reg [6:0] SSD;
divider_1 #(200000) N4 (reset, clk, clcc);
ring_counter MM(clcc, reset, A);
always @ (sh)begin
if (sh == 0) SSD = 1;
else if (sh == 1) SSD = 79;
else if (sh == 2) SSD = 18;
else if (sh == 3) SSD = 6;
else if (sh == 4) SSD = 76;
else if (sh == 5) SSD = 36;
else if (sh == 6) SSD = 32;
else if (sh == 7) SSD = 15;
```

```
else if (sh == 8) SSD = 0;
else if (sh == 9) SSD = 4;
else if (sh == 10) SSD = 8;
else if (sh == 11) SSD = 96;
else if (sh == 12) SSD = 49;
else if (sh == 13) SSD = 66;
else if (sh == 14) SSD = 48;
else SSD = 56;
end
always @ (posedge clcc) begin
case (A)
4'b0111: sh = sh0;
4'b1011: sh = sh3;
4'b1101: sh = sh2;
4'b1110: sh = sh1;
default: sh = 0;
endcase
end
endmodule
```

4 Simulation Scenario

We use 50ns as the half length of a clock in the test bench. Each half period from 0ns the following information will be printed out:

- Current clock time
- PC address
- The result stored in register $t0 to $t9 and $s0 to $s7

```
module simgroup;
parameter half_period = 50;
reg clk, reset;
reg [4:0] Input_Readreg;
reg Input_ReadPC;
wire [31:0] PCout, RegOut;
PipelinedProcessor_bonus UUT (clk, Input_Readreg, reset, Input_ReadPC, PCout,
RegOut);
initial begin
#0 clk = 0; reset = 1; Input_Readreg = 1; Input_ReadPC = 0;
#60 reset = 0;
end
initial begin
$monitor("Time: %d, CLK = %h, PC = %h, \n [$s0] = %h, [$s1] = %h, [$s2] = %h, \n
[$s3] = %h, [$s4] = %h, [$s5] = %h, \n [$s6] = %h, [$s7] = %h, [$t0] = %h, \n
[$t1] = %h, [$t2] = %h, [$t3] = %h, \n [$t4] = %h, [$t5] = %h, [$t6] = %h, \n
[$t7] = %h, [$t8] = %h, [$t9] = %h \n",
$stime, clk, UUT.PC_.pcout, UUT.M4.file[16], UUT.M4.file[17], UUT.M4.file[18],
UUT.M4.file[19], UUT.M4.file[20], UUT.M4.file[21], UUT.M4.file[22],
UUT.M4.file[23], UUT.M4.file[8], UUT.M4.file[9], UUT.M4.file[10],
UUT.M4.file[11], UUT.M4.file[12], UUT.M4.file[13], UUT.M4.file[14],
UUT.M4.file[15], UUT.M4.file[24], UUT.M4.file[25]);
```

```
$monitor("Time: %d, CLK = %h, datamem_(0) = %h,  datamem_(1) = %h, datamem_(2) =
%h, stall = %h, if_id_write_hazard=%h, pc_write_hazard=%h, memread_ex=%h,
rs_id=%h, rt_id=%h, rt_ex=%h, instruction_id=%h", $stime, clk,
UUT.datamem_.register_memory[0], UUT.datamem_.register_memory[1],
UUT.datamem_.register_memory[2], UUT.stall, UUT.if_id_write_hazard,
UUT.pc_write_hazard, UUT.memread_ex, UUT.rs_id, UUT.rt_id, UUT.rt_ex,
UUT.instruction_id);
end
always #half_period clk = ~clk;
initial #4000 $stop;
endmodule
```

5 Textual Result and Explanation

We use the demo for bonus part as our source code. And the code is as follows:

00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00100000 00001001 00000000 00110111 //addi $t1, $zero, 0x37
00000001 00001001 10000000 00100100 //and $s0, $t0, $t1
00000001 00001001 10000000 00100101 //or $s0, $t0, $t1
10101100 00010000 00000000 00000100 //sw $s0, 4($zero)
10101100 00001000 00000000 00001000 //sw $t0, 8($zero)
00000001 00001001 10001000 00100000 //add $s1, $t0, $t1
00000001 00001001 10010000 00100010 //sub $s2, $t0, $t1
00010010 00110010 00000000 00001001 //beq $s1, $s2, error0
10001100 00010001 00000000 00000100 //lw $s1, 4($zero)
00110010 00110010 00000000 01001000 //andi $s2, $s1, 0x48
00010010 00110010 00000000 00001001 //beq $s1, $s2, error1
10001100 00010011 00000000 00001000 //lw $s3, 8($zero)
00010010 00010011 00000000 00001010 //beq $s0, $s3, error2
00000010 01010001 10100000 00101010 //slt $s4, $s2, $s1 (Last)
00010010 10000000 00000000 00001111 //beq $s4, $0, EXIT
00000010 00100000 10010000 00100000 //add $s2, $s1, $0
00001000 00000000 00000000 00001110 //j Last
00100000 00001000 00000000 00000000 //addi $t0, $0, 0(error0)
00100000 00001001 00000000 00000000 //addi $t1, $0, 0
00001000 00000000 00000000 00011111 //j EXIT
00100000 00001000 00000000 00000001 //addi $t0, $0, 1(error1)
00100000 00001001 00000000 00000001 //addi $t1, $0, 1
00001000 00000000 00000000 00011111 //j EXIT
00100000 00001000 00000000 00000010 //addi $t0, $0, 2(error2)
00100000 00001001 00000000 00000010 //addi $t1, $0, 2
00001000 00000000 00000000 00011111 //j EXIT
00100000 00001000 00000000 00000011 //addi $t0, $0, 3(error3)
00100000 00001001 00000000 00000011 //addi $t1, $0, 3
00001000 00000000 00000000 00011111 //j EXIT

```
Time:         0, CLK = 0, PC = xxxxxxxx,
 [$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000,
 [$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:        50, CLK = 1, PC = 00000000,
 [$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000,
 [$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       100, CLK = 0, PC = 00000000,
 [$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000,
 [$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

At the very beginning, all registers in regfiles are 0 and PC is unknown. At the first clock cycle PC is also reset.

```
Time:       150, CLK = 1, PC = 00000004,
 [$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000,
 [$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       200, CLK = 0, PC = 00000004,
 [$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000,
 [$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

```
Time:       250, CLK = 1, PC = 00000008,
 [$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000,
 [$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       300, CLK = 0, PC = 00000008,
 [$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000,
 [$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

```
Time:       350, CLK = 1, PC = 0000000c,
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000,
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       400, CLK = 0, PC = 0000000c,
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000,
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

Until this cycle nothing is changed in the reg file. This is because the first instruction has not arrived at WB stage.

```
Time:       450, CLK = 1, PC = 00000010,
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000000,
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       500, CLK = 0, PC = 00000010,
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

At this cycle, we can see that *addi* function is correct, because $t0 changes from 0 to 0x20 at negative edge, which can make it more stable.

```
Time:       550, CLK = 1, PC = 00000014,
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000000, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       600, CLK = 0, PC = 00000014,
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

This picture also shows the correctness of *addi*, since $t1 changes from 0 to 0x37.

Time: 650, CLK = 1, PC = 00000018,
[$s0] = 00000000, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time: 700, CLK = 0, PC = 00000018,
[$s0] = 00000020, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

We can see that and function is also correct, since $s0 changes from 0 to 0x20.

Time: 750, CLK = 1, PC = 0000001c,
[$s0] = 00000020, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time: 800, CLK = 0, PC = 0000001c,
[$s0] = 00000037, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

We can see that or function is also correct, since $s0 changes from 0x20 to 0x37.

Time: 850, CLK = 1, PC = 00000020,
[$s0] = 00000037, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time: 900, CLK = 0, PC = 00000020,
[$s0] = 00000037, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

```
Time:       950, CLK = 1, PC = 00000024,
[$s0] = 00000037, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       1000, CLK = 0, PC = 00000024,
[$s0] = 00000037, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

Nothing happens because it is *sw* function.

```
Time:       1050, CLK = 1, PC = 00000024,
[$s0] = 00000037, [$s1] = 00000000, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       1100, CLK = 0, PC = 00000024,
[$s0] = 00000037, [$s1] = 00000057, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

add function is correct, since $s1 changes from 0 to 0x57.

```
Time:       1150, CLK = 1, PC = 00000028,
[$s0] = 00000037, [$s1] = 00000057, [$s2] = 00000000,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       1200, CLK = 0, PC = 00000028,
[$s0] = 00000037, [$s1] = 00000057, [$s2] = fffffffe9,
[$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
[$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
[$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
[$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
[$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

sub function is correct, since $s2 changes from 0 to 0xE9.

```
Time:       1250, CLK = 1, PC = 0000002c,
 [$s0] = 00000037, [$s1] = 00000057, [$s2] = ffffffe9,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       1300, CLK = 0, PC = 0000002c,
 [$s0] = 00000037, [$s1] = 00000057, [$s2] = ffffffe9,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       1350, CLK = 1, PC = 0000002c,
 [$s0] = 00000037, [$s1] = 00000057, [$s2] = ffffffe9,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       1400, CLK = 0, PC = 0000002c,
 [$s0] = 00000037, [$s1] = 00000057, [$s2] = ffffffe9,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

Nothing changes here because this is a *beq* function. And PC doesn't change for one more cycle. This because the rs of *beq* is the destination of last instruction *addi*. And a *nop* is added.

```
Time:       1450, CLK = 1, PC = 00000030,
 [$s0] = 00000037, [$s1] = 00000057, [$s2] = ffffffe9,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       1500, CLK = 0, PC = 00000030,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = ffffffe9,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

$s1 changes from 0x57 to 0x37, which shows that *sw* and *lw* functions are both correct.

```
Time:       1550, CLK = 1, PC = 00000030,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = ffffffe9,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       1600, CLK = 0, PC = 00000030,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = ffffffe9,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

A *nop* is added here since there is *beq* whose rs is the same as the destination of *lw*.

```
Time:       1650, CLK = 1, PC = 00000034,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = ffffffe9,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       1700, CLK = 0, PC = 00000034,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

Addi is correct, because $s2 changes from 0xE9 to 0.

```
Time:       1750, CLK = 1, PC = 00000038,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       1800, CLK = 0, PC = 00000038,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

Nothing happens here because it is the write back stage of *beq*.

```
Time:       1850, CLK = 1, PC = 00000038,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       1900, CLK = 0, PC = 00000038,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       1950, CLK = 1, PC = 00000038,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000000, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2000, CLK = 0, PC = 00000038,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

Two *nops* are added here because of the destination of *lw* is the same as rt of the next *beq*. Obviously, *sw*, and *lw* are successful because $s3 changes from 0 to 0x20.

```
Time:       2050, CLK = 1, PC = 0000003c,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2100, CLK = 0, PC = 0000003c,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

```
Time:       2150, CLK = 1, PC = 00000040,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2200, CLK = 0, PC = 00000040,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2250, CLK = 1, PC = 00000040,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2300, CLK = 0, PC = 00000040,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

A *nop* is added here because $s4 is the rs of *beq* and destination of *slt*.

```
Time:       2350, CLK = 1, PC = 00000044,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2400, CLK = 0, PC = 00000044,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

$s4 changes from 0 to 1, which shows the correctness of *slt* function ,which operates when $s2 is less than $s1.

```
Time:       2450, CLK = 1, PC = 00000048,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2500, CLK = 0, PC = 00000048,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2550, CLK = 1, PC = 00000038,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2600, CLK = 0, PC = 00000038,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

Function *j* is correct because PC changes from 48 to 38.

```
Time:       2650, CLK = 1, PC = 0000003c,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000000,
 [$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2700, CLK = 0, PC = 0000003c,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037,
 [$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

```
Time:       2750, CLK = 1, PC = 00000040,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037,
 [$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

 Time:       2800, CLK = 0, PC = 00000040,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037,
 [$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2850, CLK = 1, PC = 00000040,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037,
 [$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2900, CLK = 0, PC = 00000040,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037,
 [$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       2950, CLK = 1, PC = 0000007c,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037,
 [$s3] = 00000020, [$s4] = 00000001, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       3000, CLK = 0, PC = 0000007c,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037,
 [$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

At this point, *beq* operates, since PC changes from 40 to 7c. And $s4 changes from 0x1 to 0. The program ends.

```
Time:       3050, CLK = 1, PC = 00000080,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037,
 [$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000

Time:       3100, CLK = 0, PC = 00000080,
 [$s0] = 00000037, [$s1] = 00000037, [$s2] = 00000037,
 [$s3] = 00000020, [$s4] = 00000000, [$s5] = 00000000,
 [$s6] = 00000000, [$s7] = 00000000, [$t0] = 00000020,
 [$t1] = 00000037, [$t2] = 00000000, [$t3] = 00000000,
 [$t4] = 00000000, [$t5] = 00000000, [$t6] = 00000000,
 [$t7] = 00000000, [$t8] = 00000000, [$t9] = 00000000
```

Above is all of our text results.

6 Conclusion and Discussion

In this project, we implement the single cycle pipelined implantation of MIPS processor through Verilog. And our processor can support many instructions like sw, lw, beq, j, add, and so on. Suffering from the tough time of writing code and debugging, we all have a better understanding of MIPS processor, as well as the advantages and disadvantages of both modes.

For the single cycle processor, it is much easier to implement and there are almost no strange problems compared with pipelined implementation. However, the efficiency of single cycle is much slower because there are many modules which are not in use. But for the pipelined implantation, though it is much faster, since all parts operate at the same time, we need to spend much more time to deal with all kinds of problems like forwarding and hazard detection. In general, there is always a tradeoff. If we want the processor to save more time, then we must spend more time.