浙江大学

本科实验报告

课程名称：计算机组成

姓　名：TANG ANNA YONGQI

学　院：计算机科学与技术学院

专　业：计算机科学与技术（中加班）留学生

学　号：3180300155

生活照：

指导教师：刘海风，洪奇军

2020 年 5 月 24 日

Lab 10 – Multicycle CPU Datapath Design

Name: Anna Yongqi Tang **ID:** 3180300155 **Major:** 计算机科学与技术（中加班）留学生
Course: Computer Organization
Date: 2020-05-24 **Instructor:** 洪奇军

1. Method and Experimental Steps

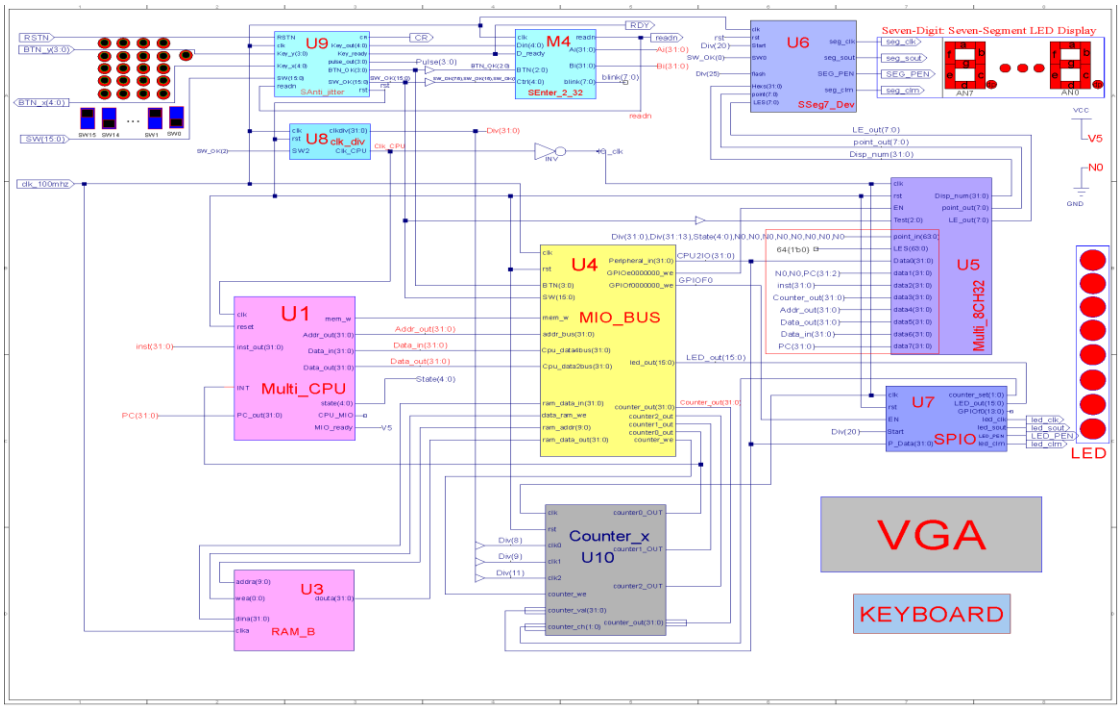


Figure 1 - topMod.sch

Lab 10 is a continuation of the multicycle CPU design from lab 9. Like before, an HDL implementation was used to implement the top module instead of the schematic pictured above. The main task of this experiment was to replace the Multi_CPU's M_datapath ngc file with an actual implementation. The .ucf for this program came from the provided courseware, and is linked to topMod.v. Synthesis had minimal warnings, and implementation was successful. A programmable file has been generated and is ready for testing on the SWORD board. The above diagram was retrieved from the lab PowerPoint. No schematic has been drawn for this experiment.

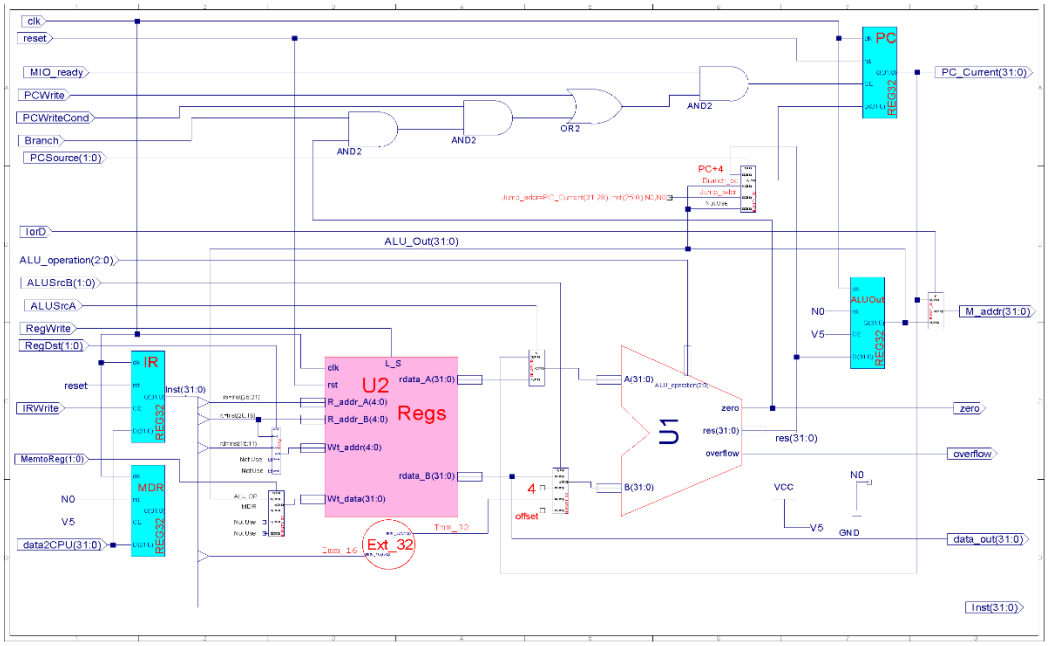


Figure 2 – M_datapath.sch

Instead of using the suggested schematic (pictured above), the M_datapath was constructed in HDL as well. The big differences of the multicycle datapath are the IR, MDR, and ALUOut. The ALU and Regs modules come from the single cycle implementation in lab 4. This datapath also supports 9 instructions: add, sub, and, or, slt, nor, lw, sw, beq and j. Lastly, M_datapath is a module of the multi_cpu component.

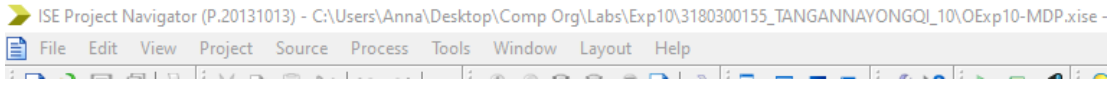


Figure 3 – 3180300155_TANGANNAYONGQI_10

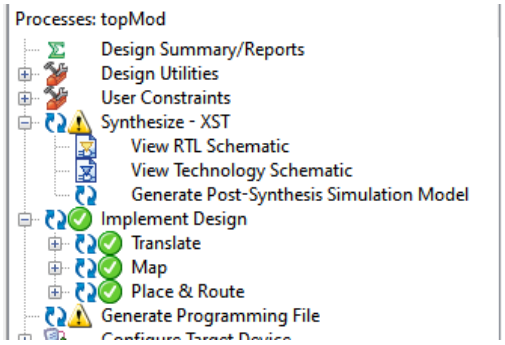


Figure 4 - .bit file generation

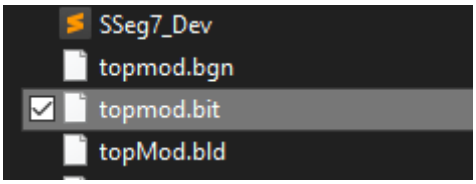


Figure 5 - .bit file generated in directory

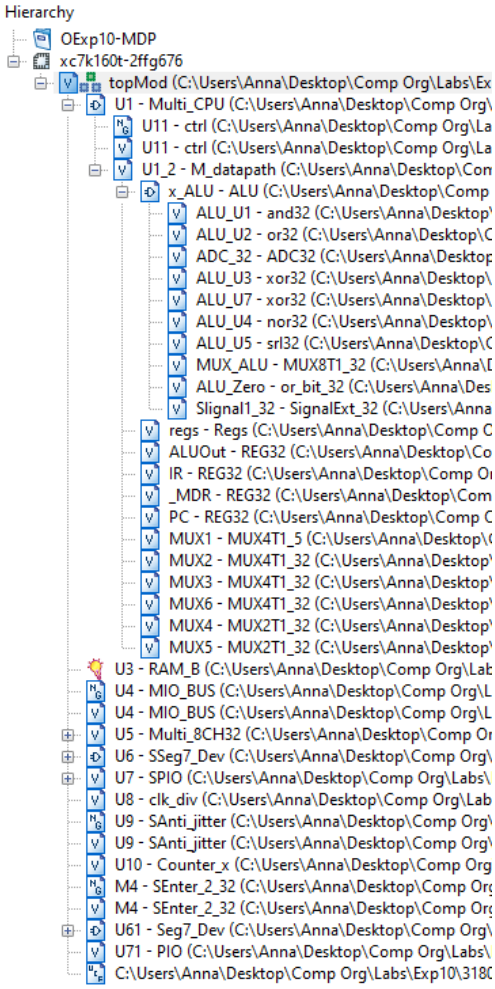


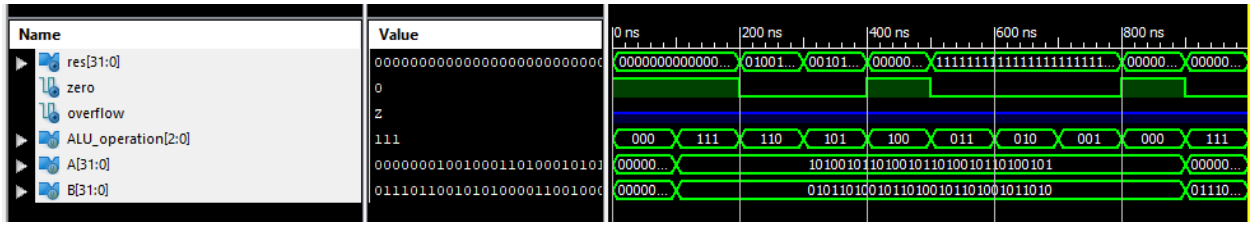
Figure 6 - file hierarchy

2. Simulations and Observations

This lab requires a MIPS program (MCPU_DEMO9.coe) to be tested on the multicyle CPU. The demo will be performed later in a future exercise. The ALU, Regs and M_datapath modules were simulated in this lab.

ALU Simulation

The code used for this simulation was taken from previous labs. The ALU_operation shows that all nine instructions have been tested.

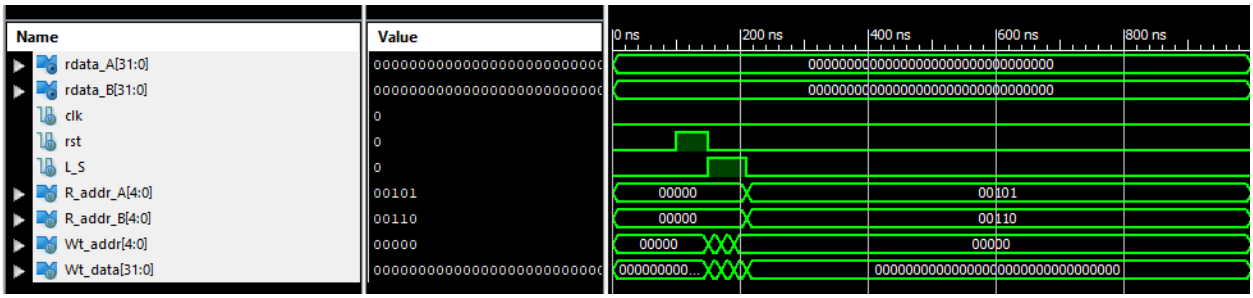


aluSim.v

```
module ALU_ALU_sch_tb();

// Inputs
   reg [2:0] ALU_operation;
   reg [31:0] A;
   reg [31:0] B;

// Output
   wire [31:0] res;
   wire zero;
   wire overflow;

// Bidirs

// Instantiate the UUT
   ALU UUT (
              .ALU_operation(ALU_operation),
              .res(res),
              .zero(zero),
              .overflow(overflow),
              .A(A),
              .B(B)
   );
// Initialize Inputs
       initial begin
              A = 0;
              B = 0;
              ALU_operation = 0;

              #100;
              // Wait 100 ns for global reset to finish

              // Add stimulus here
              A=32'hA5A5A5A5;
              B=32'h5A5A5A5A;
              ALU_operation =3'b111;
              #100;
              ALU_operation =3'b110;
              #100;
              ALU_operation =3'b101;
              #100;
              ALU_operation =3'b100;
              #100;
              ALU_operation =3'b011;
              #100;
              ALU_operation =3'b010;
              #100;
              ALU_operation =3'b001;
              #100;
              ALU_operation =3'b000;
              #100;
              A=32'h01234567;
              B=32'h76543210;
              ALU_operation =3'b111;
       end
endmodule
```

Regs Simulation

The code used for this simulation was also taken from a previous lab.



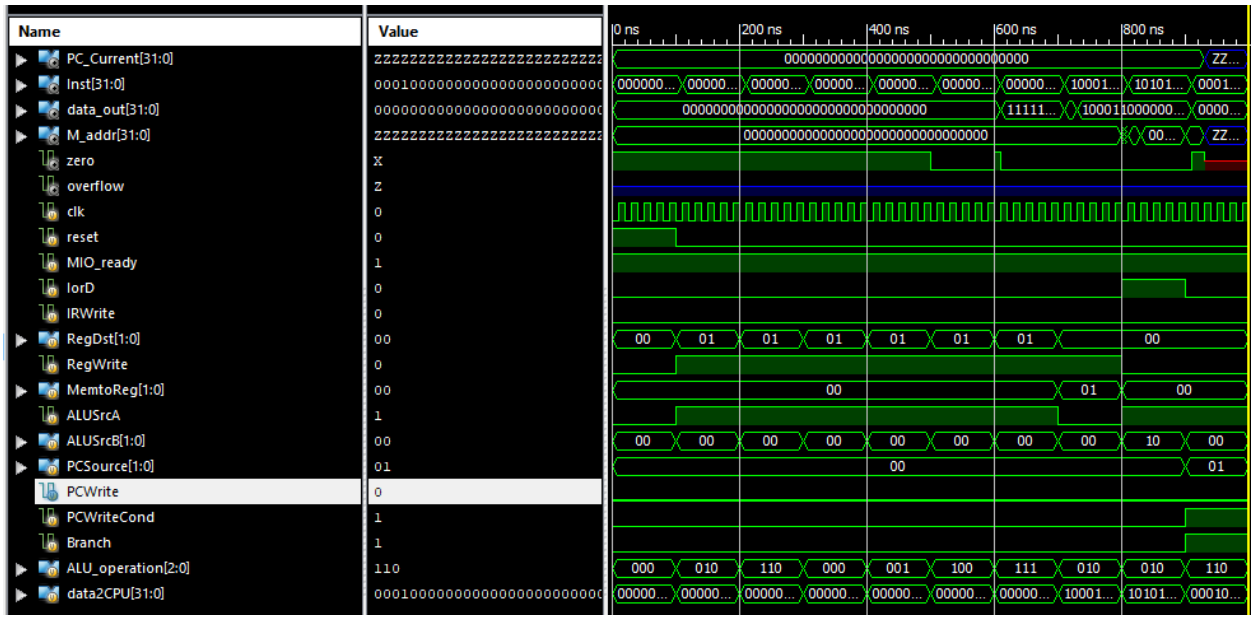
regsSim.v

```
module regsSim;

	// Inputs
	reg clk;
	reg rst;
	reg L_S;
	reg [4:0] R_addr_A;
	reg [4:0] R_addr_B;
	reg [4:0] Wt_addr;
	reg [31:0] Wt_data;

	// Outputs
	wire [31:0] rdata_A;
	wire [31:0] rdata_B;

	// Instantiate the Unit Under Test (UUT)
	Regs uut (
		.clk(clk),
		.rst(rst),
		.L_S(L_S),
		.R_addr_A(R_addr_A),
		.R_addr_B(R_addr_B),
		.Wt_addr(Wt_addr),
		.Wt_data(Wt_data),
		.rdata_A(rdata_A),
		.rdata_B(rdata_B)
	);

	initial begin
		clk = 0;
		rst = 0;
		L_S = 0;
		R_addr_A = 0;
		R_addr_B = 0;
		Wt_addr = 0;
```

```
            Wt_data = 0;

            // Wait 100 ns for global reset to finish
            #100;

            // Add stimulus here
            rst = 1;
            #50;
            rst = 0;
            L_S = 1;
            R_addr_A = 0;
            R_addr_B = 0;
            Wt_addr = 5;
            Wt_data = 32'hA5A5A5A5;
            #20;
            L_S = 1;
            R_addr_A = 0;
            R_addr_B = 0;
            Wt_addr = 6;
            Wt_data = 32'h55AA55AA;
            #20;
            L_S = 1;
            R_addr_A = 0;
            R_addr_B = 0;
            Wt_addr = 0;
            Wt_data = 32'hAAAA5555;
            #20;
            L_S = 0;
            R_addr_A = 5;
            R_addr_B = 6;
            Wt_addr = 0;
            Wt_data = 0;
            #20;
      end

endmodule
```

M_Datapath Simulation

Eight instructions; add, sub, and, or, nor, slt, lw, sw and beq were simulated. The instruction encoding was first fed into the data2CPU port, and the control signals were set by `signals. For each state/cycle of the multicycle FSM, the control signals and ALU_operation were updated. This can be observed with how different instruction classes require a different number of cycles.



M_DatapathSim.v

```
module M_datapathSim;

	// Inputs
	reg clk;
	reg reset;
	reg MIO_ready;
	reg IorD;
	reg IRWrite;
	reg [1:0] RegDst;
	reg RegWrite;
	reg [1:0] MemtoReg;
	reg ALUSrcA;
	reg [1:0] ALUSrcB;
	reg [1:0] PCSource;
	reg PCWrite;
	reg PCWriteCond;
	reg Branch;
	reg [2:0] ALU_operation;
	reg [31:0] data2CPU;

	// Outputs
	wire [31:0] PC_Current;
	wire [31:0] Inst;
	wire [31:0] data_out;
	wire [31:0] M_addr;
	wire zero;
	wire overflow;

	// Instantiate the Unit Under Test (UUT)
	M_datapath uut (
		.clk(clk),
		.reset(reset),
		.MIO_ready(MIO_ready),
		.IorD(IorD),
		.IRWrite(IRWrite),
```

```
            .RegDst(RegDst),
            .RegWrite(RegWrite),
            .MemtoReg(MemtoReg),
            .ALUSrcA(ALUSrcA),
            .ALUSrcB(ALUSrcB),
            .PCSource(PCSource),
            .PCWrite(PCWrite),
            .PCWriteCond(PCWriteCond),
            .Branch(Branch),
            .ALU_operation(ALU_operation),
            .PC_Current(PC_Current),
            .data2CPU(data2CPU),
            .Inst(Inst),
            .data_out(data_out),
            .M_addr(M_addr),
            .zero(zero),
            .overflow(overflow)
      );

      initial begin
            // Initialize Inputs
            `define signals {PCWrite, PCWriteCond, IorD, IRWrite, MemtoReg, PCSource,
ALUSrcB, ALUSrcA, RegWrite, RegDst}
            clk = 0;
            reset = 1;
            MIO_ready = 1;
            IorD = 0;
            IRWrite = 0;
            RegDst = 0;
            RegWrite = 0;
            MemtoReg = 0;
            ALUSrcA = 0;
            ALUSrcB = 0;
            PCSource = 0;
            PCWrite = 0;
            PCWriteCond = 0;
            Branch = 0;
            ALU_operation = 0;
            data2CPU = 0;
            #100;

            reset = 0;

            //add r3, r2, r2
            data2CPU = 32'b000000_00010_00010_00011_00000_100000;
            `signals = 14'b1_00_1000_0010_000;
            ALU_operation = 3'b000;
            `signals = 14'b0_00_0000_0110_000;
            `signals = 14'b0_00_0000_0001_000;
            ALU_operation = 3'b010;
            `signals = 14'b0_00_0000_0001_101;
            #100;

            //sub r4, r0, r3
            data2CPU = 32'b000000_00000_00011_00100_00000_100010;
            `signals = 14'b1_00_1000_0010_000;
            ALU_operation = 3'b000;
            `signals = 14'b0_00_0000_0110_000;
```

```
`signals = 14'b0_00_0000_0001_000;
ALU_operation = 3'b110;
`signals = 14'b0_00_0000_0001_101;
#100;

//and r5, r3, r4
data2CPU = 32'b000000_00100_00011_00101_00000_100100;
`signals = 14'b1_00_1000_0010_000;
ALU_operation = 3'b000;
`signals = 14'b0_00_0000_0110_000;
`signals = 14'b0_00_0000_0001_000;
`signals = 14'b0_00_0000_0001_101;
#100;

//or r6, r2, r4
data2CPU = 32'b000000_00100_00010_00110_00000_010110;
`signals = 14'b1_00_1000_0010_000;
 ALU_operation = 3'b000;
`signals = 14'b0_00_0000_0110_000;
`signals = 14'b0_00_0000_0001_000;
ALU_operation = 3'b001;
`signals = 14'b0_00_0000_0001_101;
#100;

//nor r1, r0, r0
data2CPU = 32'b000000_00000_00000_00001_00000_100111;
`signals = 14'b1_00_1000_0010_000;
ALU_operation = 3'b000;
`signals = 14'b0_00_0000_0110_000;
`signals = 14'b0_00_0000_0001_000;
ALU_operation = 3'b100;
`signals = 14'b0_00_0000_0001_101;
#100;

//slt r2, r0, r1
data2CPU = 32'b000000_00000_00001_00010_00000_101010;
`signals = 14'b1_00_1000_0010_000;
 ALU_operation = 3'b000;
`signals = 14'b0_00_0000_0110_000;
`signals = 14'b0_00_0000_0001_000;
ALU_operation = 3'b111;
`signals = 14'b0_00_0000_0001_101;
#100;

//lw r1, 4(r0)
 data2CPU = 32'b100011_00000_00001_00000_00000_000100;
`signals = 14'b1_00_1000_0010_000;
 ALU_operation = 3'b000;
`signals = 14'b0_00_0000_0110_000;
`signals = 14'b0_00_0000_0101_000;
 ALU_operation = 3'b010;
`signals = 14'b0_01_0000_0101_000;
`signals = 14'b0_00_0010_0000_100;
#100;

//sw r1, 8(r0)
data2CPU = 32'b101011_00000_00001_00000_00000_001000;
`signals = 14'b1_00_1000_0010_000;
```

```
              ALU_operation = 3'b000;
              `signals = 14'b0_00_0000_0110_000;
              `signals = 14'b0_00_0000_0101_000;
              ALU_operation = 3'b010;
              `signals = 14'b0_01_0000_0101_000;
             #100;

             //beq r0, r0, 4
             data2CPU = 32'b000100_00000_00000_00000_00000_000100;
             `signals = 14'b1_00_1000_0010_000;
             ALU_operation = 3'b000;
             `signals = 14'b0_00_0000_0110_000;
             `signals = 14'b0_10_0000_1001_000;
             ALU_operation = 3'b110;
             Branch = 1;
             #100;

      end

      always begin
             clk=0;
             #10;
             clk=1;
             #10;
      end

endmodule
```

3. Conclusion

This week's lab was an overview to a multicycle CPU's datapath. It reinforced the material that was taught in class, but there were some differences such as the lack of the A and B registers. Comparing the simulations of a single cycle datapath to the one in this lab, we can tell that this implementation is a lot more efficient in runtime. I look forward to continuing recreating the other modules from scratch, and adjusting to using HDL instead of schematics for the top module.

4. Source Code

All modules and components were either retrieved from previous labs or directly taken from the given files.

M_datapath – M_datapath_IO.v

```
                  module M_datapath(input clk,
                                    input reset,

                                    input MIO_ready,
                                    input IorD,
                                    input IRWrite,
                                    input[1:0] RegDst,
                                    input RegWrite,
```

```
                                    input[1:0]MemtoReg,
                                    input ALUSrcA,

                                    input[1:0]ALUSrcB,
                                    input[1:0]PCSource,
                                    input PCWrite,
                                    input PCWriteCond,
                                    input Branch,
                                    input[2:0]ALU_operation,

                                    output[31:0]PC_Current,
                                    input[31:0]data2CPU,
                                    output[31:0]Inst,
                                    output[31:0]data_out,
                                    output[31:0]M_addr,

                                    output zero,
                                    output overflow
                                   );

wire [31:0] rdata_A, rdata_B, ALU_Out, MDR, w_reg_data, Alu_A, Alu_B, res, PC_Next;
wire[4:0] reg_Rs_addr_A = Inst[25:21];
wire[4:0] reg_Rt_addr_B = Inst[20:16];
wire[4:0] reg_rd_addr = Inst[15:11];
wire[4:0] reg_Wt_addr;
wire[15:0] imm = Inst[15:0];
wire[31:0] imm_32 = {{16{imm[15]}},imm};
wire N0 = 1'b0, V5 = 1'b1;
wire CE;

assign CE = MIO_ready && (PCWrite || (PCWriteCond && zero&&Branch));
assign data_out = rdata_B;

ALU x_ALU(.A(Alu_A),
          .B(Alu_B),
          .ALU_operation(ALU_operation),
          .res(res),
          .zero(zero),
          .overflow(overflow)
          );

Regs regs(.clk(clk),
          .rst(reset),
          .R_addr_A(reg_Rs_addr_A),  //Inst(25:21)
          .R_addr_B(reg_Rt_addr_B),     //Inst(20:16)
          .Wt_addr(reg_Wt_addr),
          .Wt_data(w_reg_data),
          .L_S(RegWrite),
          .rdata_A(rdata_A),
          .rdata_B(rdata_B)
           );

REG32  ALUOut(.clk(clk),
              .rst(N0),
              .CE(V5),
              .D(res),
              .Q(ALU_Out)
              );
```

```
REG32 IR (.clk(clk),
                  .rst(reset),
                  .CE(V5),
                  .D(data2CPU),
                  .Q(Inst)
                   );

REG32  _MDR(.clk(clk),
          .rst(N0),
          .CE(V5),
          .D(data2CPU),
          .Q(MDR)
          );

REG32 PC (.clk(clk),
          .rst(reset),
          .CE(CE),
          .D(PC_next),
          .Q(PC_Current)
                    );

MUX4T1_5 MUX1(.I0(reg_Rt_addr_B),       //reg addr=IR[21:16]
              .I1(reg_rd_addr),  //reg addr=IR[15:11]
              .I2(5'b11111),            // not use
              .I3(5'b00000),            // not use
              .s(RegDst),
              .o(reg_Wt_addr)
               );


MUX4T1_32    MUX2(.I0(ALU_Out),         //ALU OP
                 .I1(MDR),
                 .I2(32'h00000000),     // not use
                 .I3(32'h00000000),     // not use
                 .s(MemtoReg),
                 .o(w_reg_data)
                );

MUX4T1_32    MUX3(.I0(data_out),              //reg out B
                 .I1(32'h00000004),                  //4 for PC+4
                 .I2(imm_32[31:0]),
                 .I3({imm_32[29:0],N0,N0}),
                                  .s(ALUSrcB),
                 .o(Alu_B)
                );

MUX4T1_32    MUX6(.I0(res[31:0]),
                 .I1(ALU_Out[31:0]),
                 .I2({PC_Current[31:28],Inst[25:0],N0,N0}),
                 .I3(32'h00000000),
                 .s(PCSource),
                 .o(PC_Next)
                 );

MUX2T1_32    MUX4(.I0(rdata_A),  // reg out A
                 .I1(PC_Current), // PC
                 .s(ALUSrcA),
```

```
                .o(Alu_A)
                );

MUX2T1_32    MUX5(.I0(PC_Current),      //IF
                .I1(ALU_Out),   //access memory
                .s(IorD),
                .o(M_addr)
                );

endmodule
```