浙江大學

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　6 月 06 日

Lab 12 – Multicycle CPU Controller Extension

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: 2020-06-06 **Instructor:** 洪奇军

1. Method and Experimental Steps

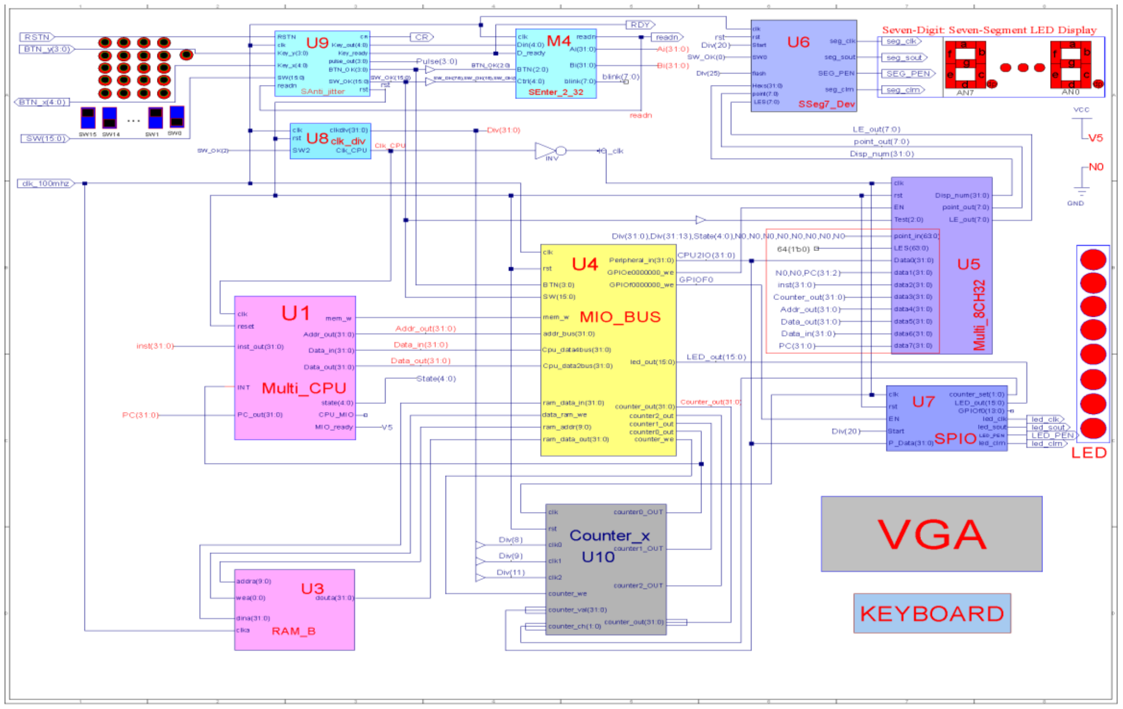
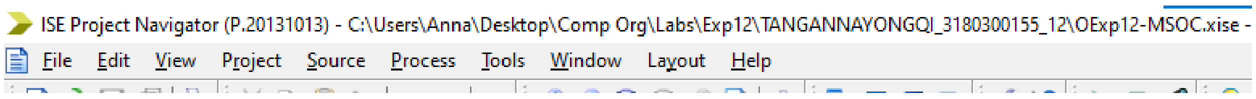


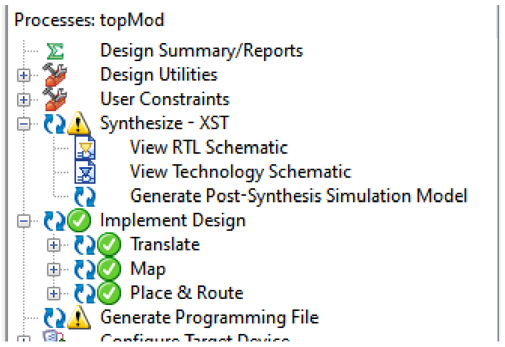
Figure 1 - topMod.sch

Lab 12 is an extension of lab 11, which more i-type and jump instructions are implemented. 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. The CPU currently supports add, sub, and, or, xor, nor, slt, srl, jr, jal, addi, andi, ori, xori, lui, lw, sw, beq, bne, slti, j and jal. For each state/circle in the FSM, we define different value controls. The opcodes are also set accordingly in this module. After the controller finishes processing each module, it goes back to the instruction fetch stage.



ISE Project Navigator (P.20131013) - C:\Users\Anna\Desktop\Comp Org\Labs\Exp12\TANGANNAYONGQI_3180300155_12\OExp12-MSOC.xise -

Figure 3 – 3180300155_TANGANNAYONGQI_12

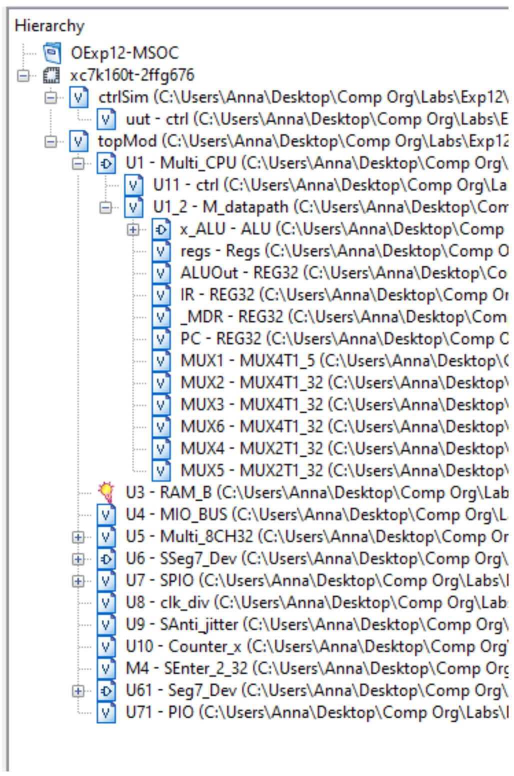


```
Processes: topMod
  Design Summary/Reports
  Design Utilities
  User Constraints
  Synthesize - XST
    View RTL Schematic
    View Technology Schematic
    Generate Post-Synthesis Simulation Model
  Implement Design
    Translate
    Map
    Place & Route
  Generate Programming File
  Configure Target Device
```

Figure 4 - .bit file generation

```
SSeg7_Dev
topmod.bgn
topmod.bit
topMod.bld
```

Figure 5 - .bit file generated in directory



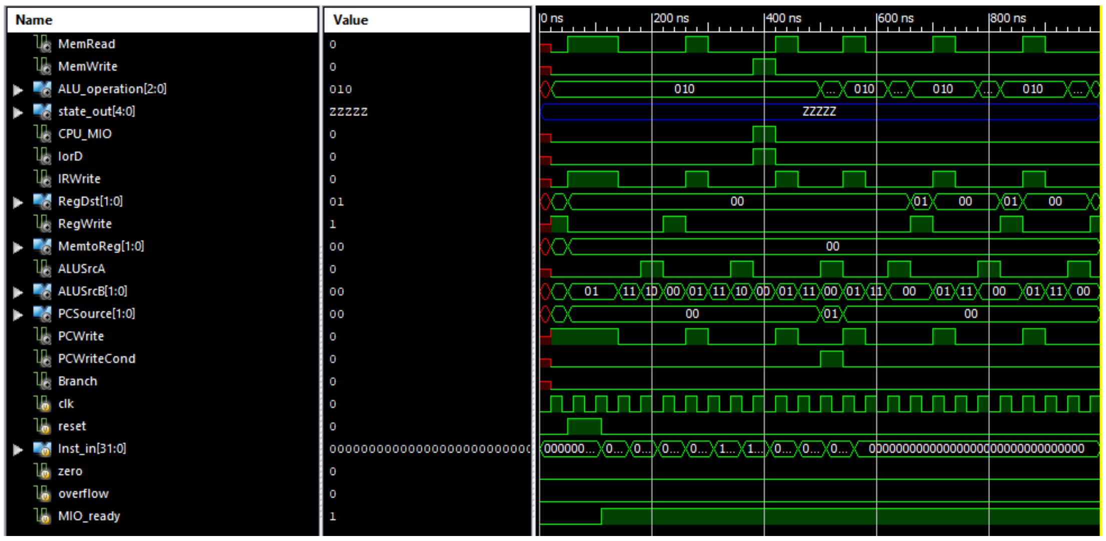
```
Hierarchy
  OExp12-MSOC
  xc7k160t-2ffg676
    ctrlSim (C:\Users\Anna\Desktop\Comp Org\Labs\Exp12\
      uut - ctrl (C:\Users\Anna\Desktop\Comp Org\Labs\E
    topMod (C:\Users\Anna\Desktop\Comp Org\Labs\Exp12
      U1 - Multi_CPU (C:\Users\Anna\Desktop\Comp Org\
        U11 - ctrl (C:\Users\Anna\Desktop\Comp Org\La
        U1_2 - M_datapath (C:\Users\Anna\Desktop\Com
          x_ALU - ALU (C:\Users\Anna\Desktop\Comp
          regs - Regs (C:\Users\Anna\Desktop\Comp O
          ALUOut - REG32 (C:\Users\Anna\Desktop\Co
          IR - REG32 (C:\Users\Anna\Desktop\Comp Or
          _MDR - REG32 (C:\Users\Anna\Desktop\Com
          PC - REG32 (C:\Users\Anna\Desktop\Comp C
          MUX1 - MUX4T1_5 (C:\Users\Anna\Desktop\
          MUX2 - MUX4T1_32 (C:\Users\Anna\Desktop\
          MUX3 - MUX4T1_32 (C:\Users\Anna\Desktop\
          MUX6 - MUX4T1_32 (C:\Users\Anna\Desktop\
          MUX4 - MUX2T1_32 (C:\Users\Anna\Desktop\
          MUX5 - MUX2T1_32 (C:\Users\Anna\Desktop\
      U3 - RAM_B (C:\Users\Anna\Desktop\Comp Org\Lab
      U4 - MIO_BUS (C:\Users\Anna\Desktop\Comp Org\L
      U5 - Multi_8CH32 (C:\Users\Anna\Desktop\Comp Or
      U6 - SSeg7_Dev (C:\Users\Anna\Desktop\Comp Org\
      U7 - SPIO (C:\Users\Anna\Desktop\Comp Org\Labs\
      U8 - clk_div (C:\Users\Anna\Desktop\Comp Org\Lab
      U9 - SAnti_jitter (C:\Users\Anna\Desktop\Comp Org\
      U10 - Counter_x (C:\Users\Anna\Desktop\Comp Org
      M4 - SEnter_2_32 (C:\Users\Anna\Desktop\Comp Org
      U61 - Seg7_Dev (C:\Users\Anna\Desktop\Comp Org\
      U71 - PIO (C:\Users\Anna\Desktop\Comp Org\Labs\
```

Figure 6 - file hierarchy

2. Simulations and Observations

This lab requires a MIPS program (mem.coe) to be tested on the multicyle CPU. The demo will be performed later in a future exercise. The controller module was simulated in this lab. I have simulated the add, addi, beq, j, lw, sw, jal, bne and lui instructions, and the results seemed to be consistent.

ctrl Simulation



ctrlSim.v

```
module ctrlSim;

	// Inputs
	reg clk;
	reg reset;
	reg [31:0] Inst_in;
	reg zero;
	reg overflow;
	reg MIO_ready;

	// Outputs
	wire MemRead;
	wire MemWrite;
	wire [2:0] ALU_operation;
	wire [4:0] state_out;
	wire CPU_MIO;
	wire IorD;
	wire IRWrite;
	wire [1:0] RegDst;
	wire RegWrite;
	wire [1:0] MemtoReg;
	wire ALUSrcA;
	wire [1:0] ALUSrcB;
	wire [1:0] PCSource;
	wire PCWrite;
	wire PCWriteCond;
	wire Branch;

	// Instantiate the Unit Under Test (UUT)
	ctrl uut (
		.clk(clk),
		.reset(reset),
		.Inst_in(Inst_in),
        .zero(zero),
        .overflow(overflow),
        .MIO_ready(MIO_ready),
        .MemRead(MemRead),
        .MemWrite(MemWrite),
        .ALU_operation(ALU_operation),
        .state_out(state_out),
        .CPU_MIO(CPU_MIO),
        .IorD(IorD),
        .IRWrite(IRWrite),
        .RegDst(RegDst),
        .RegWrite(RegWrite),
        .MemtoReg(MemtoReg),
        .ALUSrcA(ALUSrcA),
        .ALUSrcB(ALUSrcB),
        .PCSource(PCSource),
        .PCWrite(PCWrite),
        .PCWriteCond(PCWriteCond),
        .Branch(Branch)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        reset = 0;
        Inst_in = 0;
        zero = 0;
        overflow = 0;
        MIO_ready = 0;

        // Wait 100 ns for global reset to finish
        #50;
        reset=1;
        #60;
        reset=0;
        MIO_ready=1;
        Inst_in = 32'h014B4820; //add t1, t2, t3
        #50;
        Inst_in = 32'h2014003f; //addi s4, zero, 3f
        #50;
        Inst_in = 32'h11600005; //beq t3, zero, 5
        #50;
        Inst_in = 32'h0800000c; //j 12
        #50;
        Inst_in = 32'h8D69FFFF; //lw t1, 0xffff(t3)
        #50;
        Inst_in = 32'hAD71FFFF; //sw s1, 0xffff(t3)
        #50;
        Inst_in = 32'h0C00BFAF; //jal bfaf
        #50;
        Inst_in = 32'h15700005; //bne s0 5
        #50;
        Inst_in = 32'h3C0B0001; //lui t3 1
        #50;
        Inst_in = 32'h00000000;
        #50;
    end
    always begin
```

```
                clk=0;
                #20;
                clk=1;
                #20;
        end

endmodule
```

3. Conclusion

This week’s lab was a continuation of last week’s lab and concludes the lab portion of the Computer Organization course. Both the single cycle and multicycle CPUs have been completed, and I look forward to comparing their performances in a later analysis. It is a shame that I will not be able to implement these onto the SWORD board, but I greatly enjoyed doing these labs. It was a good supplement to what I have learned in the theoretical portion of this course.

4. Source Code

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

Controller – multi_ctrl_IO.v

```
module ctrl(input  clk,
                       input  reset,
                       input  [31:0] Inst_in,
                       input  zero,
                       input  overflow,
                       input  MIO_ready,
                       output reg MemRead,
                       output reg MemWrite,
                       output reg[2:0]ALU_operation,
                       output [4:0]state_out,

                       output reg CPU_MIO,
                       output reg IorD,
                       output reg IRWrite,
                       output reg [1:0]RegDst,
                       output reg RegWrite,
                       output reg [1:0]MemtoReg,
                       output reg ALUSrcA,
                       output reg [1:0]ALUSrcB,
                       output reg [1:0]PCSource,
                       output reg PCWrite,
                       output reg PCWriteCond,
                       output reg Branch
```

```
                      );

wire Rtype, LS, IBeq, Jump, Load, Store;
wire[5:0] OP = Inst_in[31:26];
reg[3:0] state;
reg[1:0] ALUop;

parameter IF = 4'b0000,  ID = 4'b0001, Mem_Ex = 4'b0010, Mem_RD =
4'b0011,
                LW_WB = 4'b0100,  Mem_W = 4'b0101,  R_Exc = 4'b0110,
R_WB = 4'b0111,
                Beq_Exc = 4'b1000, J = 4'b1001, I_Exc = 5'b01010, I_WB
= 5'b01011,
         Lui_Exc = 5'b01100, Bne_Exc = 5'b01101, Jr = 5'b01110, Jal =
5'b01111,
         Jalr = 5'b10000,  Error = 4'b1111;

`define Datapath_signals {PCWrite, PCWriteCond,IorD, MemRead,
MemWrite,IRWrite, MemtoReg, PCSource, ALUSrcA, ALUSrcB, RegWrite,
RegDst, Branch, ALUop, CPU_MIO}

parameter  value0 = 20'b10010100000010000000,    value1 =
20'b00000000000110000000,
             value2 = 20'b00000000001100000000,  value3 =
20'b00110000000000000001,
             value4 = 20'b00000001000001000000,  value5 =
20'b00101000000000000001,
             value6 = 20'b00000000001000000100,  value7 =
20'b00000000000001010000,
             value8 = 20'b01000000011000001010,  value9 =
20'b10000000100000000000,
                  value10 = 20'b00000000001100000110, value11 =
20'b00000000000001000000,
           value12 = 20'b00000010001111000000,   value13 =
20'b01000000011000000010,
           value14 = 20'b10000000110000000000,   value15 =
20'b10000011100001100000,
           value16 = 20'b10000011110001000000,   value17 =
20'b10000011100001100000;

parameter AND=3'b000, OR=3'b001, ADD=3'b010, SUB=3'b110, NOR=3'b100,
SLT=3'b111, XOR=3'b011, SRL=3'b101;

/*
assign Rtype = ~|OP;             //if OP=000000 then Rtype = 1
assign LS = (OP == 6'b10x011) ? 1 : 0;//if OP=10x011 then      LS = 1
assign IBeq = (OP == 6'b000100) ? 1 : 0;    //if OP=000100 then   Ibeq
= 1
```

```
assign Jump = (OP == 6'b000010) ? 1 : 0;    //if OP=000010 then  Jump =
1
assign Load = (OP == 6'b100011) ? 1 : 0;    //if OP=100011 then  Load =
1
assign Store = (OP == 6'b101011) ? 1 : 0;   //if OP=101011 then  Store
= 1
*/

always @ (posedge clk or posedge reset)
      if (reset==1) state <= IF;
      else
            case(state)
                  IF: if(MIO_ready) state <= ID;
                         else state <= IF;
                  ID: case (Inst_in[31:26])
                              6'b000000:
             begin
                 case(Inst_in[5:0])
                     6'b001000:  state <= Jr;          //Jr
                     6'b001001:  state <= Jalr;        //Jalr
                     default:    state <= R_Exc;       //R-type OP
                 endcase
             end
             6'b100011: state <= Mem_Ex;     //Lw
             6'b101011: state <= Mem_Ex;     //Sw
             6'b001000: state <= I_Exc;      //Addi
             6'b001100: state <= I_Exc;      //Andi
             6'b001101: state <= I_Exc;      //Ori
             6'b001110: state <= I_Exc;      //Xori
             6'b001010: state <= I_Exc;      //Slti
             6'b001111: state <= Lui_Exc;    //Lui
             6'b000100: state <= Beq_Exc;    //Beq
             6'b000101: state <= Bne_Exc;    //Bne
             6'b000010: state <= J;         //Jump
             6'b000011: state <= Jal;       //Jal
             default:   state <= Error;
                       endcase
                  Mem_Ex: if(Inst_in[29]) state <= Mem_W;
                               else state <= Mem_RD;
                  Mem_RD: state <= LW_WB;
                  LW_WB: state <= IF;
                  Mem_W: state <= IF;
                  R_Exc: state <= R_WB;
                  R_WB: state <= IF;
                  I_Exc: state <= I_WB;
                  I_WB:  state <= IF;
                  Lui_Exc: state <= IF;
                  Beq_Exc: state <= IF;
```

```
                Bne_Exc:   state <= IF;
                Jal: state <= IF;
                Jr: state <= IF;
                J: state <= IF;
                Error: state <= Error;
                default: state <= Error;
           endcase

always @ * begin
    case(state)                  //state
        IF:         `Datapath_signals = value0;
        ID:           `Datapath_signals = value1;
        Mem_Ex:    `Datapath_signals = value2;
        Mem_RD:       `Datapath_signals = value3;
        LW_WB:      `Datapath_signals = value4;
        Mem_W:      `Datapath_signals = value5;
        R_Exc:      `Datapath_signals = value6;
        R_WB:       `Datapath_signals = value7;
        Beq_Exc:  `Datapath_signals = value8;
        J:       `Datapath_signals = value9;
        I_Exc:        `Datapath_signals = value10;
        I_WB:    `Datapath_signals = value11;
        Lui_Exc:  `Datapath_signals = value12;
        Bne_Exc:  `Datapath_signals = value13;
        Jr:           `Datapath_signals = value14;
        Jal:        `Datapath_signals = value15;
        Jalr:       `Datapath_signals = value16;
        default:  `Datapath_signals = value0;
    endcase
end

always @ * begin
    case(ALUop)
        2'b00: ALU_operation = 3'b010;//add????
        2'b01: ALU_operation = 3'b110;//sub????
        2'b10:
        case (Inst_in[5:0])
            6'b100000: ALU_operation = ADD;
            6'b100010: ALU_operation = SUB;
            6'b100100: ALU_operation = AND;
            6'b100101: ALU_operation = OR;
            6'b100111: ALU_operation = NOR;
            6'b101010: ALU_operation = SLT;
            6'b000010: ALU_operation = SRL;       //shfit 1bit right
            6'b000000: ALU_operation = XOR;
            default:   ALU_operation = ADD;
        endcase
        2'b11:
             case (Inst_in[31:26])
                 6'b001010: ALU_operation = SLT;       //slti
            6'b001000: ALU_operation = ADD;      //addi
            6'b001100: ALU_operation = AND;      //andi
            6'b001101: ALU_operation = OR;       //ori
            6'b001110: ALU_operation = XOR;      //xori
            default:   ALU_operation = ADD;
        endcase
    endcase
end

endmodule
```