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　4 月 27 日

Lab 6 – CPU Design & Instruction 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: 2020-04-27 **Instructor:** 洪奇军

1. Method and Experimental Steps

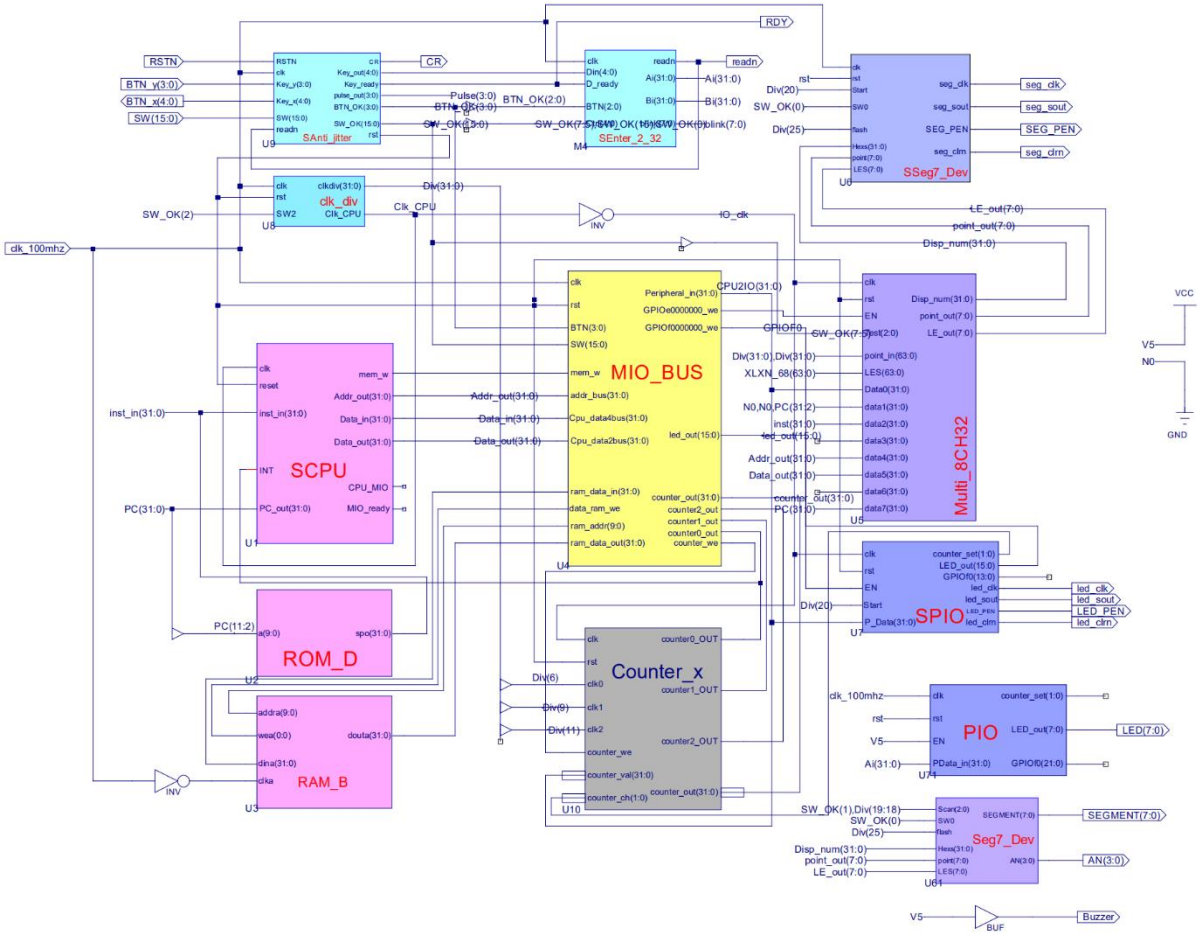

Figure 1 - topMod.sch

This depicts the completion of lab 7. The purpose of this week's experiment was to add extensions to the controller and datapath so that it can process different classes of instructions. Jump, memory reference and branch instructions are now supported as well. The control signals are slightly different from the previous version of the controller. The .ucf for this program came from the provided courseware, and is linked to topMod.sch. Synthesis had minimal warnings, and implementation was successful. A programmable file has been generated and is ready for testing on the SWORD board.

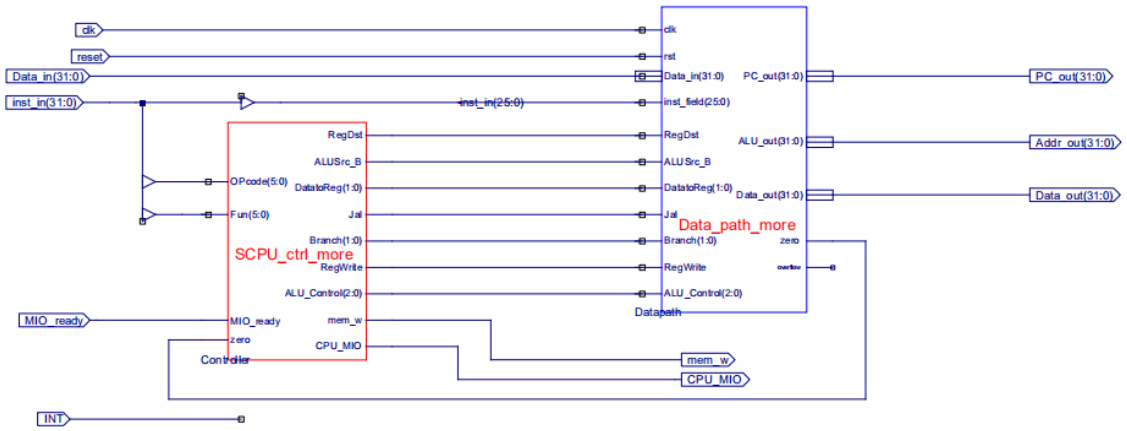

Figure 2 – SCPU_more.sch

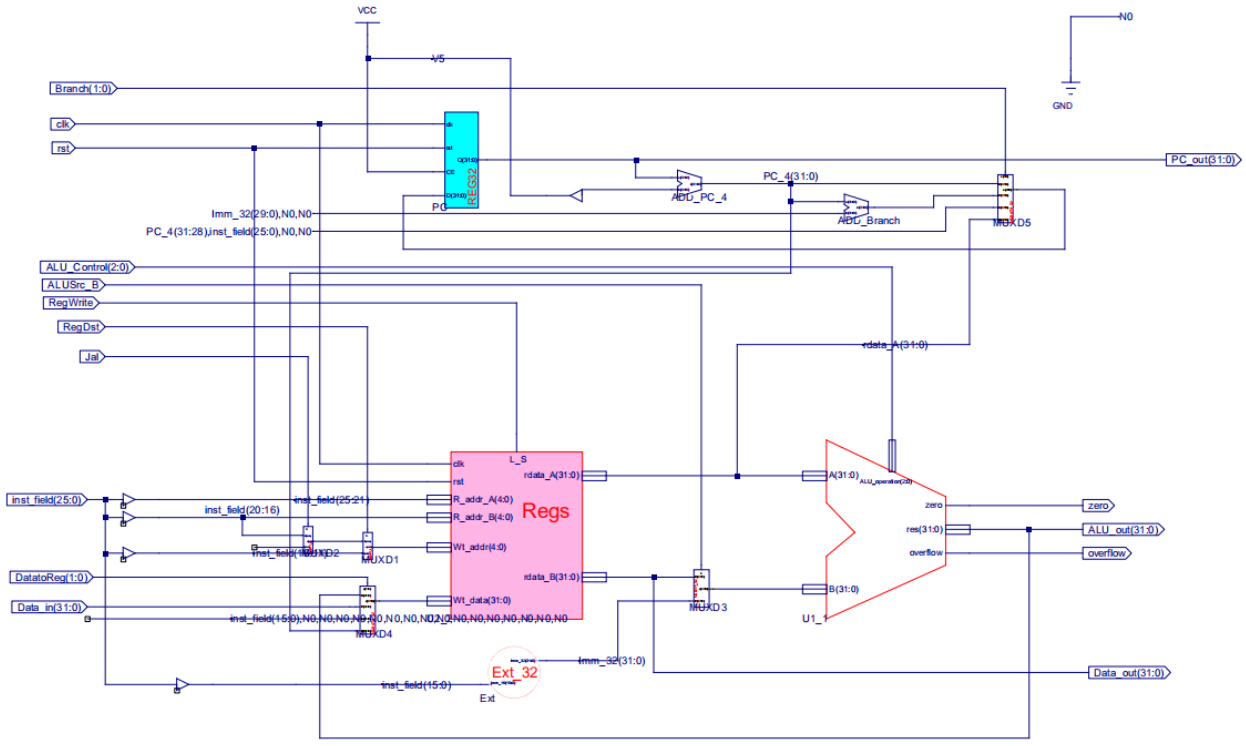

Figure 3 – Data_path_more.sch

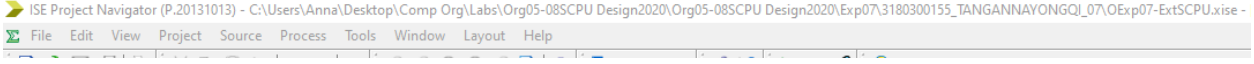

Figure 4 – 3180300155_TANGANNAYONGQI_07

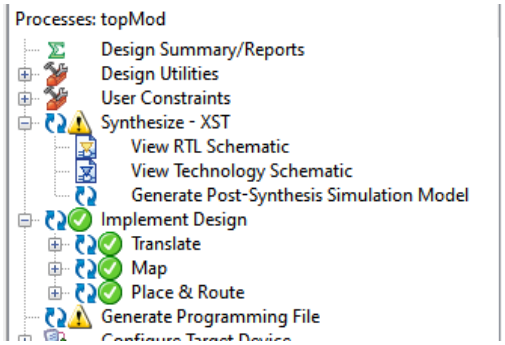

Figure 5 - .bit file generation

Figure 6 - .bit file generated in directory

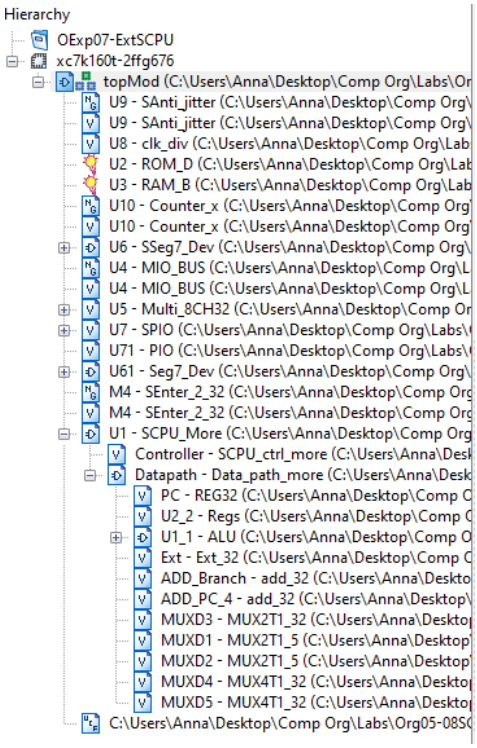

Figure 7 - file hierarchy

2. Simulations and Observations

This lab requires a MIPS program to be designed and tested on the CPU. The DEMO program and datapath testing will be performed later. There is an inconsistency with the simulation below, compared to the one presented in the PowerPoint. I was unable to track down what caused the issue and used the Verilog Test Fixture presented in the slides as a benchmark. I also used last lab's simulation code for the controller, with the extended instructions added in.

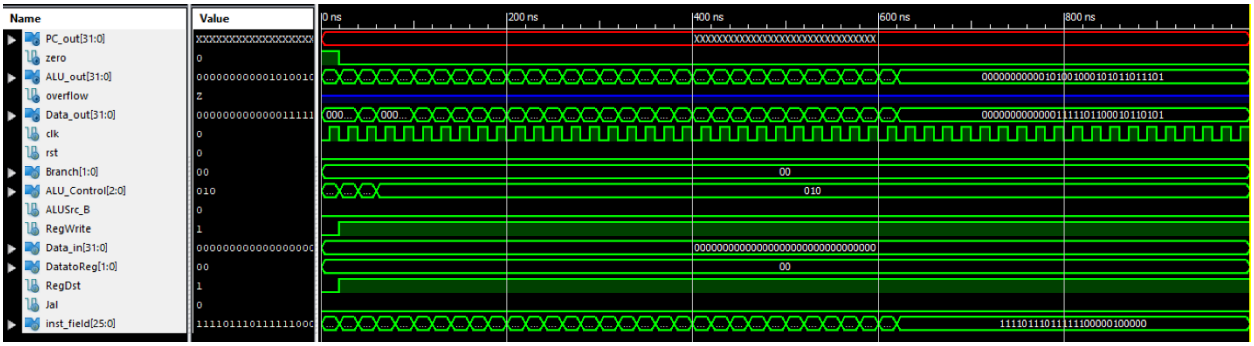

Figure 8 – Datapath Simulation

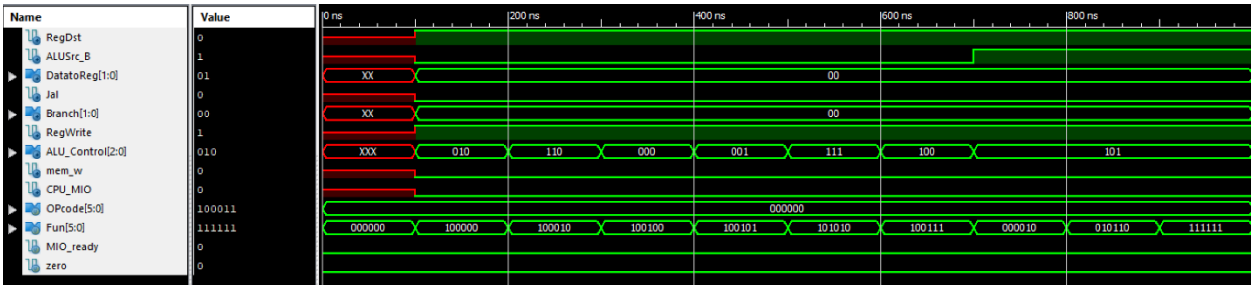

Figure 9 – Controller Simulation

Datapath Simulation

```
`timescale 1ns / 1ps

module Data_path_more_Data_path_more_sch_tb();

// Inputs
   reg clk;
   reg rst;
   reg [1:0] Branch;
   reg [2:0] ALU_Control;
   reg ALUSrc_B;
   reg RegWrite;
   reg [31:0] Data_in;
   reg [1:0] DatatoReg;
   reg RegDst;
   reg Jal;
   reg [25:0] inst_field;

// Output
   wire [31:0] PC_out;
   wire zero;
   wire [31:0] ALU_out;
   wire overflow;
   wire [31:0] Data_out;

// Bidirs

// Instantiate the UUT
   Data_path_more UUT (
              .PC_out(PC_out),
```

```
            .clk(clk),
            .rst(rst),
            .Branch(Branch),
            .ALU_Control(ALU_Control),
            .zero(zero),
            .ALU_out(ALU_out),
            .overflow(overflow),
            .Data_out(Data_out),
            .ALUSrc_B(ALUSrc_B),
            .RegWrite(RegWrite),
            .Data_in(Data_in),
            .DatatoReg(DatatoReg),
            .RegDst(RegDst),
            .Jal(Jal),
            .inst_field(inst_field)
   );
// Initialize Inputs
      initial begin
            clk = 0;
            rst = 0;
            Branch = 0;
            ALU_Control = 0;
            ALUSrc_B = 0;
            RegWrite = 0;
            Data_in = 0;
            DatatoReg = 0;
            RegDst = 0;
            Jal = 0;
            inst_field = 0;

            #20

            rst = 0;

            ALU_Control = 3'b100;
            RegWrite = 1;
            RegDst = 1;
            inst_field = 26'b00000_00000_00001_00000_100111;
            #20;

            ALU_Control = 3'b111;
            inst_field = 26'b00000_00001_00010_00000_101010;
            #20;

            Branch = 0;
            ALU_Control = 3'b010;
            ALUSrc_B = 0;
            RegWrite = 1;
            RegDst = 1;
            inst_field = 26'b00010_00010_00011_00000_100000;
            #20;
            inst_field = 26'b00011_00010_00100_00000_100000;
            #20;
            inst_field = 26'b00100_00011_00101_00000_100000;
            #20;
            inst_field = 26'b00101_00100_00110_00000_100000;
            #20;
            inst_field = 26'b00110_00101_00111_00000_100000;
```

```
            #20;
            inst_field = 26'b00111_00110_01000_00000_100000;
            #20;
            inst_field = 26'b01000_00111_01001_00000_100000;
            #20;
            inst_field = 26'b01001_01000_01010_00000_100000;
            #20;
            inst_field = 26'b01010_01001_01011_00000_100000;
            #20;
            inst_field = 26'b01011_01010_01100_00000_100000;
            #20;
            inst_field = 26'b01100_01011_01101_00000_100000;
            #20;
            inst_field = 26'b01101_01100_01110_00000_100000;
            #20;
            inst_field = 26'b01110_01101_01111_00000_100000;
            #20;
            inst_field = 26'b01111_01110_10000_00000_100000;
            #20;
            inst_field = 26'b10000_01111_10001_00000_100000;
            #20;
            inst_field = 26'b10001_10000_10010_00000_100000;
            #20;
            inst_field = 26'b10010_10001_10011_00000_100000;
            #20;
            inst_field = 26'b10011_10010_10100_00000_100000;
            #20;
            inst_field = 26'b10100_10011_10101_00000_100000;
            #20;
            inst_field = 26'b10101_10100_10110_00000_100000;
            #20;
            inst_field = 26'b10110_10101_10111_00000_100000;
            #20;
            inst_field = 26'b10111_10110_11000_00000_100000;
            #20;
            inst_field = 26'b11000_10111_11001_00000_100000;
            #20;
            inst_field = 26'b11001_11000_11010_00000_100000;
            #20;
            inst_field = 26'b11010_11001_11011_00000_100000;
            #20;
            inst_field = 26'b11011_11010_11100_00000_100000;
            #20;
            inst_field = 26'b11100_11011_11101_00000_100000;
            #20;
            inst_field = 26'b11101_11100_11110_00000_100000;
            #20;
            inst_field = 26'b11110_11101_11111_00000_100000;
            #20;
            end

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

endmodule
```

Controller Simulation

```
module SCPU_ctrl_moreSim;

	// Inputs
	reg [5:0] OPcode;
	reg [5:0] Fun;
	reg MIO_ready;
	reg zero;

	// Outputs
	wire RegDst;
	wire ALUSrc_B;
	wire [1:0] DatatoReg;
	wire Jal;
	wire [1:0] Branch;
	wire RegWrite;
	wire [2:0] ALU_Control;
	wire mem_w;
	wire CPU_MIO;

	// Instantiate the Unit Under Test (UUT)
	SCPU_ctrl_more uut (
		.OPcode(OPcode),
		.Fun(Fun),
		.MIO_ready(MIO_ready),
		.zero(zero),
		.RegDst(RegDst),
		.ALUSrc_B(ALUSrc_B),
		.DatatoReg(DatatoReg),
		.Jal(Jal),
		.Branch(Branch),
		.RegWrite(RegWrite),
		.ALU_Control(ALU_Control),
		.mem_w(mem_w),
		.CPU_MIO(CPU_MIO)
	);

	initial begin
		// Initialize Inputs
		OPcode = 0;
		Fun = 0;
		MIO_ready = 0;
		zero = 0;

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

		// Add stimulus here
		OPcode = 6'b000000; //ALU??,?? ALUop=2'b10; RegDst=1; RegWrite=1
		Fun = 6'b100000;    //add,??ALU_Control=3'b010
		#100;
		Fun = 6'b100010;    //sub,??ALU_Control=3'b110
		#100;
		Fun = 6'b100100;    //and,??ALU_Control=3'b000
		#100;
		Fun = 6'b100101;    //or,??ALU_Control=3'b001
		#100;
```

```
            Fun = 6'b101010;    //slt,??ALU_Control=3'b111
            #100;
            Fun = 6'b100111;    //nor,??ALU_Control=3'b100
            #100;
            Fun = 6'b000010;    //srl,??ALU_Control=3'b101
            #100;
            Fun = 6'b010110;    //xor,??ALU_Control=3'b011
            #100;
            Fun = 6'b111111;    //??
            #100;
            OPcode = 6'b100011;//load??,?? ALUop=2'b00, RegDst=0,
            #100;           // ALUSrc_B=1, MemtoReg=1, RegWrite=1
            OPcode = 6'b101011;
            #100; //store??,??ALUop=2'b00, mem_w=1, ALUSrc_B=1
            OPcode = 6'b000100;//beq??,?? ALUop=2'b01, Branch=1
            #100;
            OPcode = 6'b000010;//jump??,?? Jump=1
            #100;
            OPcode = 6'b001010; //slti??,??ALUop=2'b11, RegDst=0,
            #100;              //ALUSrc_B=1, RegWrite=1

            OPcode = 6'h3f;            //??
            Fun = 6'b000000;           //??
      end

endmodule
```

3. Conclusion

This lab was conceptually simple, and it was just basically an extension of lab 6. The simulation was also a bit odd to understand. I am looking forward to implementing this completed lab onto the SWORD board, and seeing what output comes out after running the demo MIPS program.

4. Source Code

SCPU_ctrl_more.v

```
module SCPU_ctrl_more(
                    input[5:0]OPcode,   //Opcode
                    input[5:0]Fun,      //Function
                    input MIO_ready,  //CPU Wait
                    input zero,
                    output reg RegDst,
                    output reg ALUSrc_B,
                    output reg [1:0]DatatoReg,
                    output reg Jal,
                    output reg [1:0]Branch,
                    output reg RegWrite,
                    output reg [2:0]ALU_Control,
                    output reg mem_w,
                    output reg CPU_MIO
    );

`define CPU_ctrl_signals
{RegDst,ALUSrc_B,DatatoReg,Jal,Branch,RegWrite,ALU_Control,mem_w,CPU_MIO}
       always @* begin
         case(OPcode)
             6'b000000: begin
```

```
               case(Fun)
                    6'b100000: `CPU_ctrl_signals = 13'b1000000101000; //add
                  6'b100010: `CPU_ctrl_signals = 13'b1000000111000; //sub
                  6'b100100: `CPU_ctrl_signals = 13'b1000000100000; //and
                  6'b100101: `CPU_ctrl_signals = 13'b1000000100100; //or
                  6'b100110: `CPU_ctrl_signals = 13'b1000000101100; //xor
                  6'b100111: `CPU_ctrl_signals = 13'b1000000110000; //nor
                  6'b101010: `CPU_ctrl_signals = 13'b1000000111100; //slt
                  6'b000010: `CPU_ctrl_signals = 13'b1100000110100; //srl
                  6'b001000: `CPU_ctrl_signals = 13'b1000011000000; //jr
                  6'b001001: `CPU_ctrl_signals = 13'b1011111100000; //jalr
               endcase
             end

      6'b100011: begin `CPU_ctrl_signals = 13'b0101000101000; end //load
      6'b101011: begin `CPU_ctrl_signals = 13'b0101000001010; end //store

      6'b000100: begin
             if (zero == 1'b1) `CPU_ctrl_signals = 13'b0000001011000; //beq
             else `CPU_ctrl_signals = 13'b0000000011000;
             end

      6'b000101: begin
             if (zero == 1'b0) `CPU_ctrl_signals = 13'b0000000011000; //bne
             else `CPU_ctrl_signals = 13'b0000001011000;
             end

      6'b000010: begin `CPU_ctrl_signals = 13'b0000010000000; end //jump

      6'b001010: begin `CPU_ctrl_signals = 13'b0100000111100; end //slti

      6'b001110: begin `CPU_ctrl_signals = 13'b0100000101100; end //xori

      6'b000011: begin `CPU_ctrl_signals = 13'b0011110100000; end //jal

      default: begin `CPU_ctrl_signals = 13'b0000000000000; end
      endcase
      end

endmodule
```