

**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**

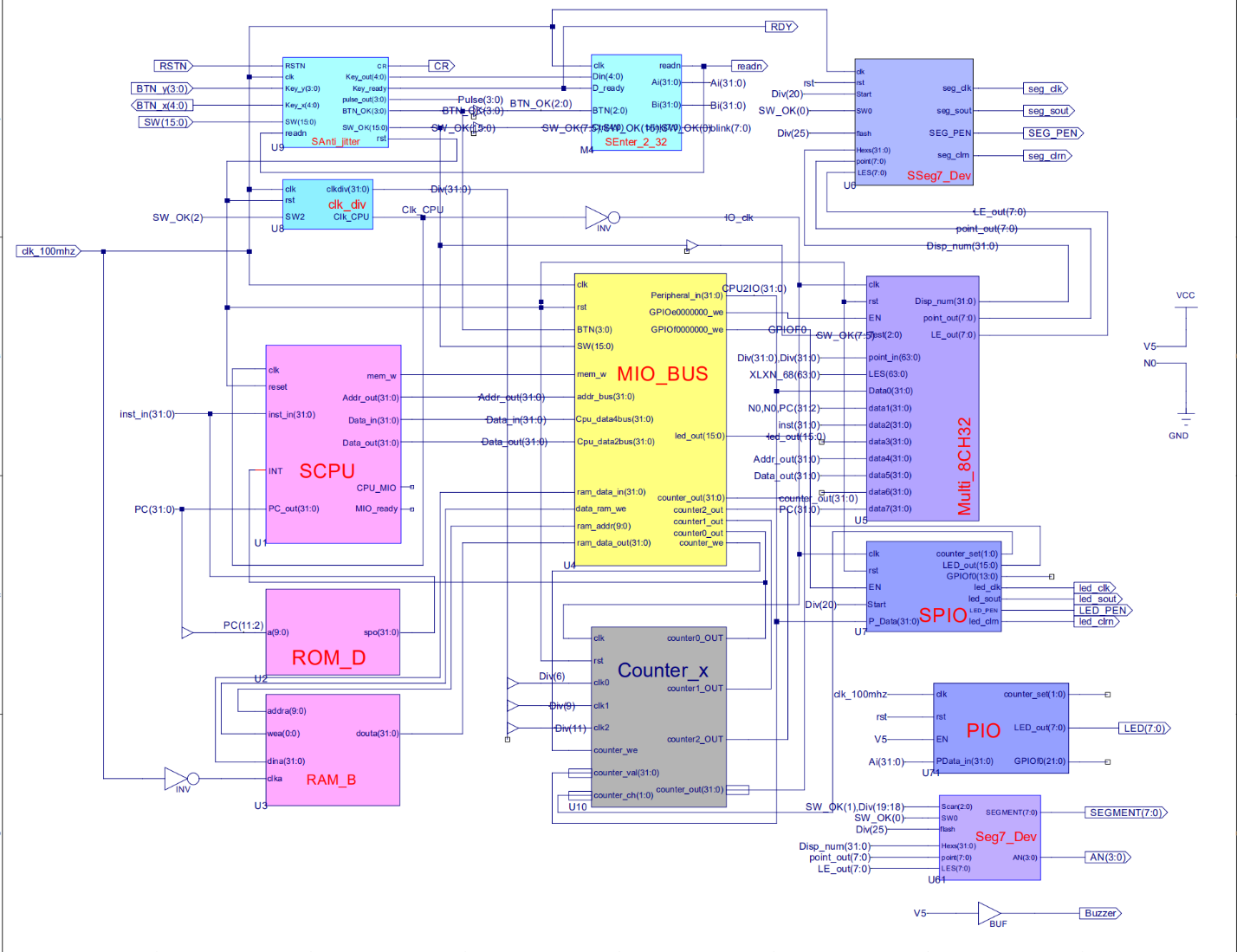


Figure 1 - topMod.sch

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.

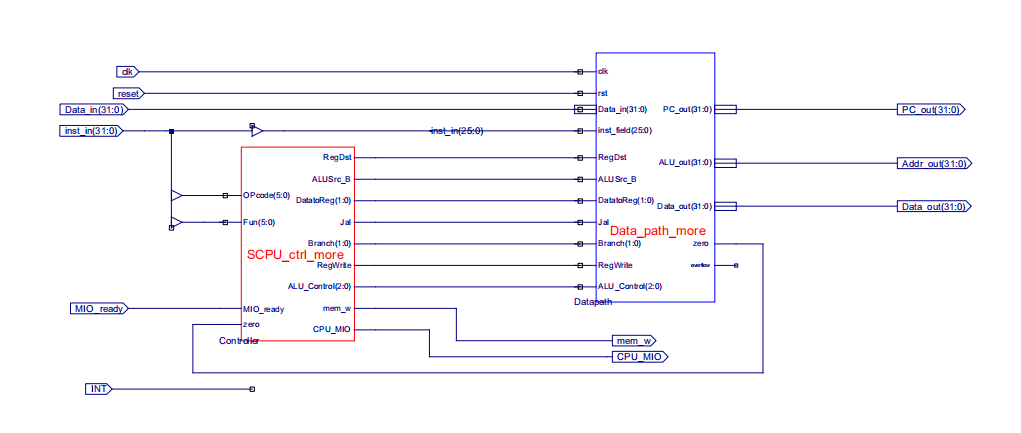


Figure 2 – SCPU\_more.sch

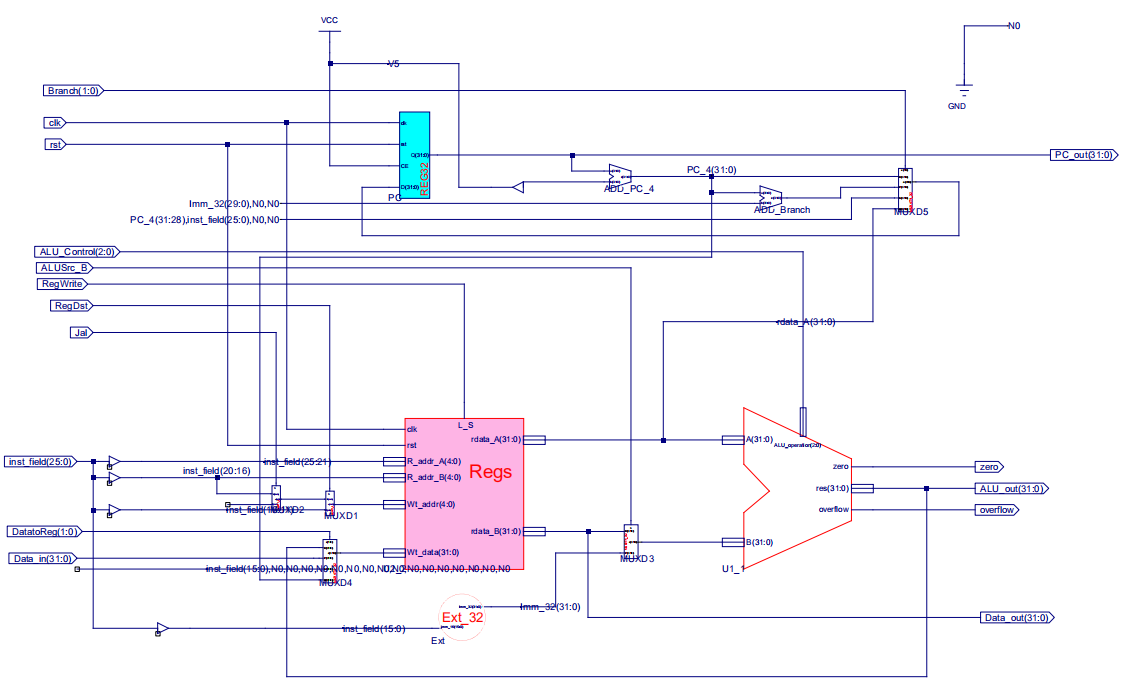


Figure 3 – Data\_path\_more.sch



Figure 4 – 3180300155\_TANGANNAYONGQI\_07

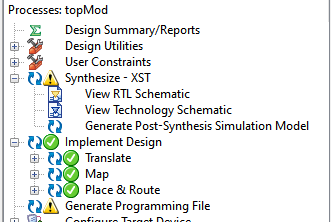
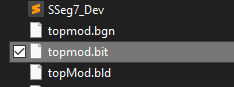
 

Figure 5 - .bit file generation Figure 6 - .bit file generated in directory

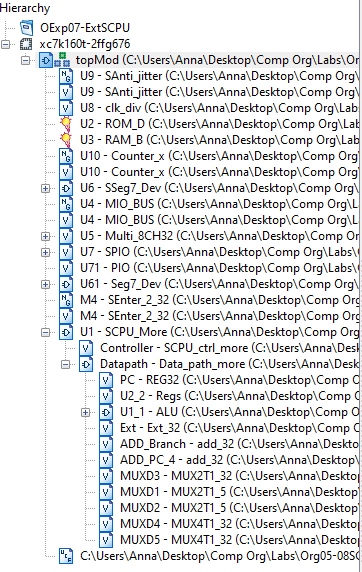


Figure 7 - file hierarchy

**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.

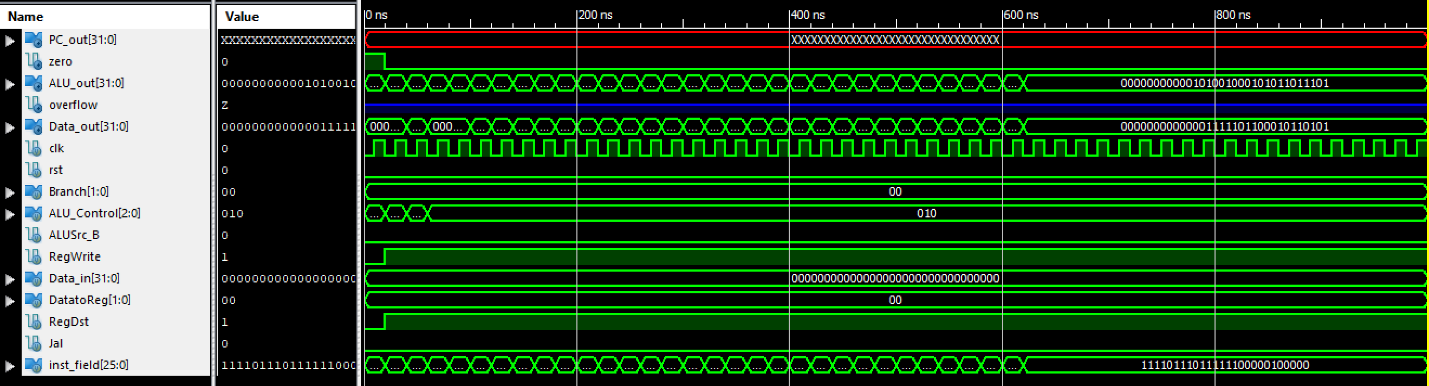


Figure 8 – Datapath Simulation

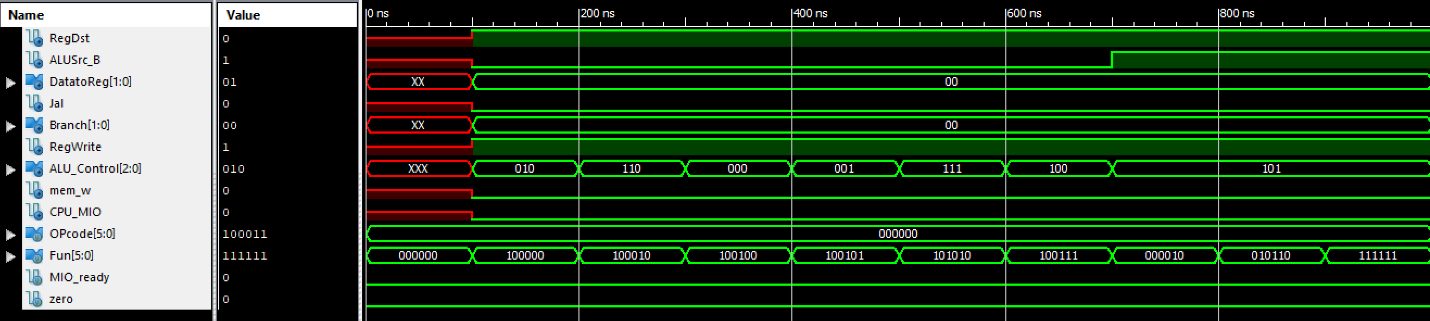


Figure 9 – Controller Simulation

*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.

1. **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