**中国科学技术大学计算机学院**

**《计算机组成原理实验》报告**



实验题目：\_\_\_\_多周期CPU\_\_\_\_\_

学生姓名：\_\_\_\_\_\_\_李昱祁\_\_\_\_\_\_

学生学号：\_\_\_\_\_PB18071496\_\_\_\_

完成日期：\_\_\_\_\_2020/05/25\_\_\_\_

计算机实验教学中心制

2019年09月

【实验目的】

1.理解计算机硬件的基本组成、结构和工作原理；

2.掌握数字系统的设计和调试方法；

3.熟练掌握数据通路和控制器的设计和描述方法。

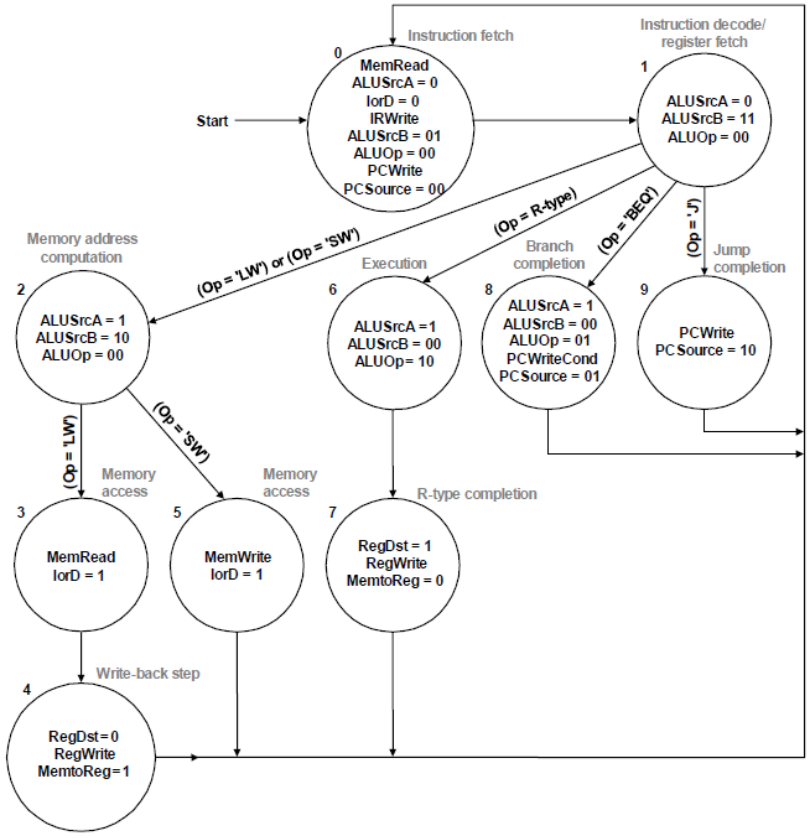
【实验环境】

1. Vivado
2. PC一台
3. Linux操作系统

【实验过程】

1. 多周期 CPU 设计
2. ***逻辑设计：***

多周期 CPU 在单周期的基础上改进而成，主要根据不同的指令，设计了不同的状态来完成。控制状态图如下：



此外，为了实现addi 指令，我们还要额外设计其状态转换如下：

IF -> ID -> MAC -> AIC

其中 MAC 阶段（实际上是 lw/sw的“Memory address computation”状态）可以实现寄存器内容与立即数相加，符合 addi 指令的要求；新设一个上图中不存在的状态“AIC”，完成数据写回寄存器堆的操作。

本次实验中，指令与数据放在同一个存储器中，因此在该存储器的“读地址”端口需要加入选择器以确定我们读的是指令段还是数据段内容；并且，PC 的自增需要通过主 ALU 来完成，所以 ALU 的第一个操作数来源也要通过选择器来判断。

1. ***核心代码：***

（其中 isIF，done 信号和 pc\_out 信号用于仿真检查，分别表示当前状态是否为取指令、最终存储器0x08地址的值、以及指令指针 pc 的值）

`timescale 1ns / 1ps

module *cpu\_multi\_cycle*( //多周期CPU

input clk,

input rst,

output isIF,

output [31:0] done,

output [31:0] pc\_out

);

// 状态机信号

localparam IF = 4'b0000,

ID = 4'b0001,

MAC = 4'b0010,

MAL = 4'b0011,

WB = 4'b0100,

MAS = 4'b0101,

EX = 4'b0110,

RC = 4'b0111,

BC = 4'b1000,

JC = 4'b1001,

AIC = 4'b1010;

//指令

localparam ADD = 6'b000000,

ADDI = 6'b001000,

LW = 6'b100011,

SW = 6'b101011,

BEQ = 6'b000100,

J = 6'b000010;

//内部信号

reg PCwe,IorD,MemRead,MemWrite,MemtoReg,IRWrite,ALUSrcA,RegWrite,RegDst;

reg [1:0] PCSource,ALUSrcB;

reg [2:0] alu\_m;

reg [3:0] state,next\_state;

reg [32:0] pc,mdr,a,b,alu\_out,instr;

wire zf;

wire [31:0] pc\_in,pc\_jump,MemData;

wire [31:0] rd1,rd2;

wire [31:0] alu\_result;

wire [31:0] mem\_addr;

wire [31:0] wd\_rf;

wire [31:0] alu\_a,alu\_b;

wire [4:0] wa\_rf;

wire [31:0] signed\_extend\_imm,s\_e\_shift\_imm;

dist\_mem\_gen\_0 ram(

.a(mem\_addr[10:2]),

.d(b),

.clk(clk),

.we(MemWrite),

.spo(MemData),

.dpra(2),

.dpo(done)

);

register\_file rf(

.clk(clk),

.rst(rst),

.we(RegWrite),

.ra1(instr[25:21]),

.ra2(instr[20:16]),

.wa(wa\_rf),

.wd(wd\_rf),

.rd1(rd1),

.rd2(rd2)

);

alu ALU(

.a(alu\_a),

.b(alu\_b),

.m(alu\_m),

.y(alu\_result),

.zf(zf)

);

assign isIF = (state == IF);

assign mem\_addr = IorD ? alu\_out : pc;

assign wa\_rf = RegDst ? instr[15:11] : instr[20:16];

assign wd\_rf = MemtoReg ? mdr : alu\_out;

assign alu\_a = ALUSrcA ? a : pc;

assign alu\_b = ALUSrcB[1] ?

(ALUSrcB[0] ? s\_e\_shift\_imm : signed\_extend\_imm) : (ALUSrcB[0] ? 4 :b );

assign pc\_in = PCSource[1] ?

(PCSource[0] ? 0 : pc\_jump) : (PCSource[0] ? alu\_out : alu\_result);

assign pc\_out = pc;

assign signed\_extend\_imm = instr[15] ? {{16{1'b1}},instr[15:0]} : {{16{1'b0}},instr[15:0]};

assign s\_e\_shift\_imm = {signed\_extend\_imm[29:0],2'b00};

assign pc\_jump = {pc[31:28],instr[25:0],2'b00};

always@(posedge clk,posedge rst)

begin

if(rst)

begin

pc = 0;

instr = 0;

mdr = 0;

a = 0;

b = 0;

alu\_out = 0;

end

else

begin

if(PCwe)

pc = pc\_in;

if(IRWrite)

instr = MemData;

mdr = MemData;

a = rd1;

b = rd2;

alu\_out = alu\_result;

end

end

always@(posedge clk,posedge rst)

begin

if(rst)

begin

state = IF;

end

else

begin

state = next\_state;

end

end

//next\_state logic

always@(\*)

begin

case(state)

IF: next\_state = ID;

ID:

begin

case(instr[31:26])

ADD:next\_state = EX;

ADDI:next\_state = MAC;

LW:next\_state = MAC;

SW:next\_state = MAC;

BEQ:next\_state =BC;

J:next\_state = JC;

default:next\_state = IF;

endcase

end

MAC:

begin

case(instr[31:26])

ADDI:next\_state = AIC;

LW:next\_state = MAL;

SW:next\_state = MAS;

default:next\_state = IF;

endcase

end

MAL:next\_state = WB;

WB:next\_state = IF;

MAS:next\_state = IF;

EX:next\_state = RC;

RC:next\_state = IF;

BC:next\_state = IF;

JC:next\_state = IF;

AIC:next\_state = IF;

default:next\_state = IF;

endcase

end

//output logic

always@(\*)

begin

ALUSrcA = 0;

ALUSrcB = 2'b00;

alu\_m = 3'b000;

MemRead = 0;

IorD = 0;

IRWrite = 0;

MemWrite = 0;

MemtoReg = 0;

RegWrite = 0;

RegDst = 0;

PCwe = 0;

PCSource = 2'b00;

case(state)

IF:

begin

MemRead = 1'b1;

IorD = 1'b0;

IRWrite = 1'b1;

ALUSrcA = 1'b0;

ALUSrcB = 2'b01;

alu\_m = 3'b000;

PCwe =1'b1;

PCSource = 2'b00;

end

ID:

begin

ALUSrcA = 1'b0;

ALUSrcB = 2'b11;

alu\_m = 3'b000;

end

MAC:

begin

ALUSrcA = 1;

ALUSrcB = 2'b10;

alu\_m = 3'b000;

end

MAL:

begin

MemRead = 1;

IorD = 1;

end

WB:

begin

RegDst = 0;

RegWrite = 1;

MemtoReg = 1;

end

MAS:

begin

MemWrite = 1;

IorD = 1;

end

EX:

begin

ALUSrcA = 1;

ALUSrcB = 2'b00;

alu\_m = 3'b000;

end

RC:

begin

RegDst = 1;

RegWrite = 1;

MemtoReg = 0;

end

BC:

begin

ALUSrcA = 1;

ALUSrcB = 2'b00;

alu\_m = 3'b001;

PCSource = 2'b01;

PCwe = (zf==1);

end

JC:

begin

PCwe = 1;

PCSource = 2'b10;

end

AIC:

begin

RegDst = 0;

RegWrite = 1;

MemtoReg = 0;

end

default: ;

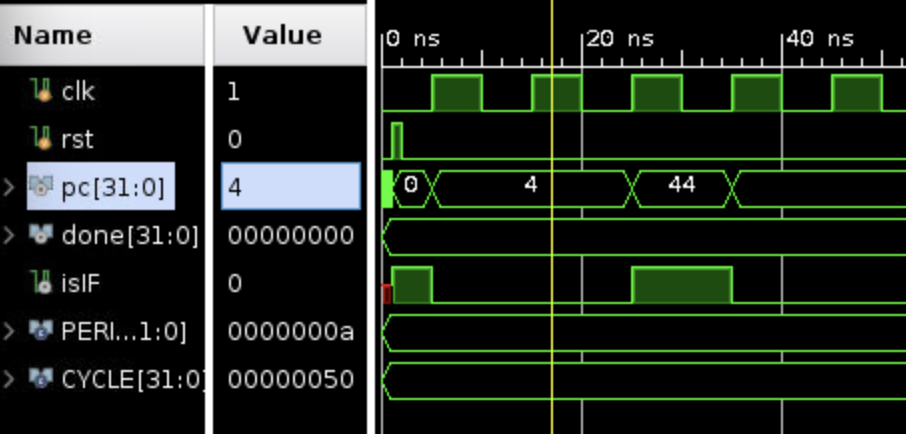
endcase

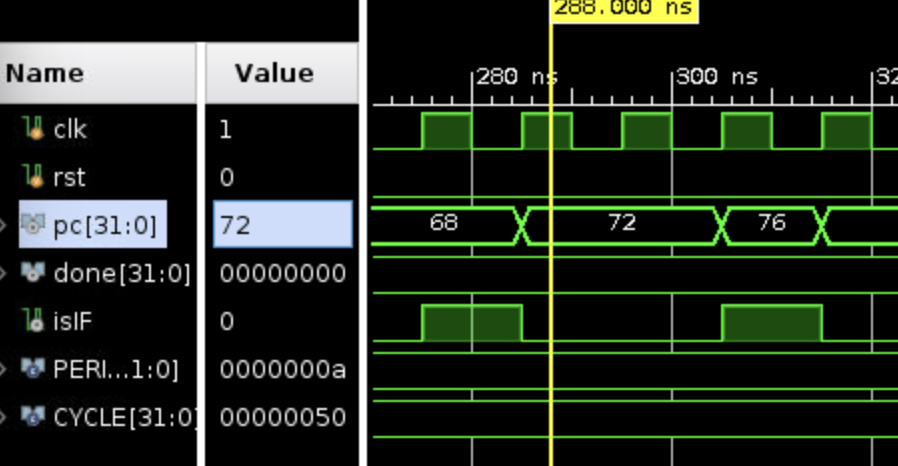
end

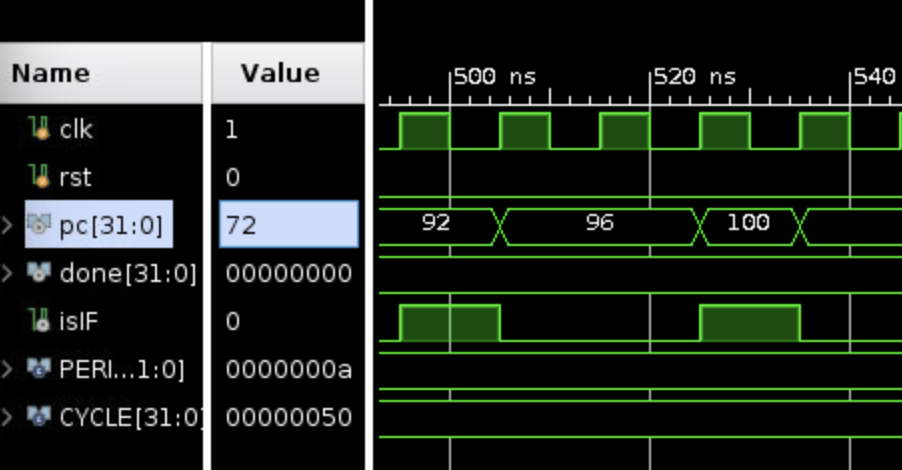
endmodule

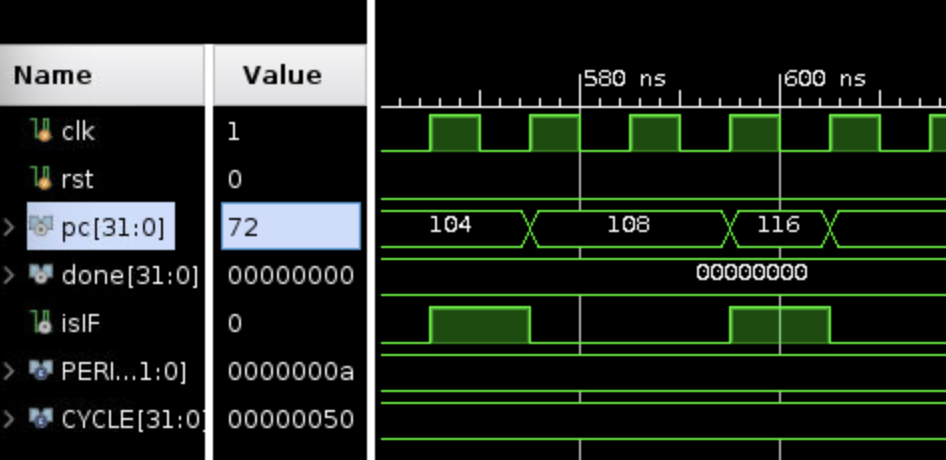
1. ***仿真结果：***

首先我们来看几次周期跳转。没有经历取指令 IF 阶段的 pc 值对应的指令都是被跳过的指令，对应地址从 0 到 44、68 到 76、92 到 100、104 到 116等跳转，如下面几张图所示：

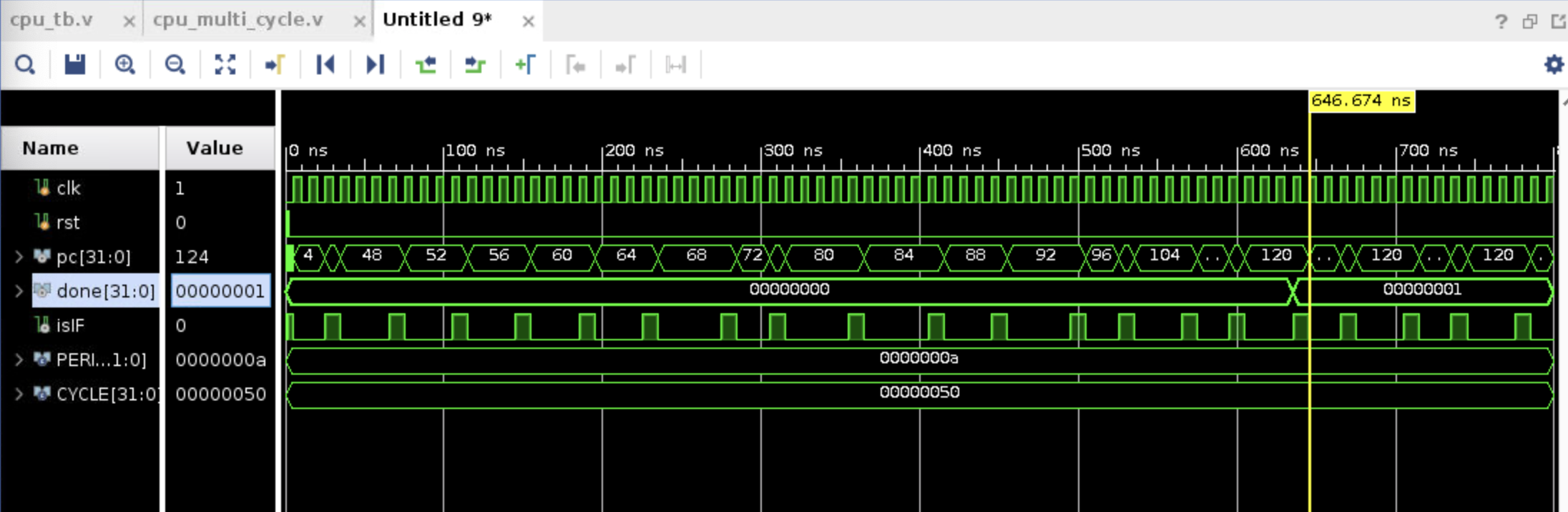








之后再查看存储器地址 0x08 的内容，最终在执行完 116 对应的 sw 指令后，由0变为1：



顺利通过了测试，验证了 CPU 设计的正确性。

1. Debug 模块
2. ***逻辑设计：***

根据实验讲义中的要求，类比上一次实验直接修改即可

1. ***核心代码：***

module *dbu*(

input succ , step ,inc , m\_rf , dec,clk,rst,

input [2:0] sel,

output run ,

output reg [15:0] led,

output reg [31:0] display

);

reg [7:0] SSEG\_CA ;

reg [7:0] SSEG\_AN ;

reg succ\_r1,succ\_r2,step\_r1,step\_r2,inc\_r1,inc\_r2 , dec\_r1,dec\_r2;

wire inc\_edg,dec\_edg,step\_edg;

reg [14:0] cnt;

wire clk10 ;

reg [2:0] sel\_SSEG;

wire [7:0] a,b,c,d,e,f,g,h;

reg [31:0] m\_rf\_addr ;

wire en\_m\_rf\_addr;

wire PC\_we,PC\_write,PCwritecond,IorD ,MemtoReg, MemWrite,IRWrite,RegDst,RegWrite,ALUop,ALUsrcA, zf;

wire [1:0] ALUsrcB ,PCsource;

wire [5:0] op;

wire [4:0] rs , rt;

wire [15:0] imm;

wire [31:0] pc\_in ,pc\_out, rd0 , rd1,alu\_rslt,alu\_reg\_data ,memdata, mem\_reg\_data,rd0\_tmp,rd1\_tmp ;

cpu\_multi\_cycle\_debug cpu(.clk(clk),.rst(rst),.run(run),.m\_rf\_addr(m\_rf\_addr),.en\_m\_rf\_addr(en\_m\_rf\_addr),.PC\_write(PC\_write),.PCwritecond(PCwritecond),.IorD(IorD) ,.MemtoReg(MemtoReg) ,.MemWrite(MemWrite),.IRWrite(IRWrite),.RegDst(RegDst),.RegWrite(RegWrite),.ALUop(ALUop),.ALUsrcA (ALUsrcA), .zf(zf),.imm(imm), .op(op),.rs(rs),.rt(rt) ,.PC\_we(PC\_we), .ALUsrcB (ALUsrcB),.PCsource(PCsource),.pc\_in(pc\_in) ,.pc\_out(pc\_out), .rd0(rd0) , .rd1(rd1),.rd0\_tmp(rd0\_tmp) , .rd1\_tmp(rd1\_tmp),.alu\_rslt(alu\_rslt),.alu\_reg\_data(alu\_reg\_data) ,.memdata(memdata), .mem\_reg\_data(mem\_reg\_data));

assign run = succ | (~succ & step\_edg);

assign en\_m\_rf\_addr = (~sel[2])&(~sel[1])&(~sel[0]) ;

always@(negedge clk)

step\_r1 <= step;

always@(negedge clk)

step\_r2 <= step\_r1;

assign step\_edg = step\_r1 & (~step\_r2);

always@(posedge clk)

inc\_r1 <= inc;

always@(posedge clk)

inc\_r2 <= inc\_r1;

assign inc\_edg = inc\_r1 & (~inc\_r2);

always@(posedge clk)

dec\_r1 <= dec;

always@(posedge clk)

dec\_r2 <= dec\_r1;

assign dec\_edg = dec\_r1 & (~dec\_r2);

always@(posedge clk,posedge rst)

begin

case (sel)

3'b000:begin

if(m\_rf)

begin

display <= memdata;

end

else begin

display <= rd0\_tmp;

end

led <= {2'b00, ALUsrcB ,PCsource, PC\_we, IorD ,MemtoReg, MemWrite,IRWrite,RegDst,RegWrite,ALUop,ALUsrcA, zf};

end

3'b001:begin

display <= pc\_out ;

led <= {2'b00, ALUsrcB ,PCsource, PC\_we, IorD ,MemtoReg, MemWrite,IRWrite,RegDst,RegWrite,ALUop,ALUsrcA, zf};

end

3'b010:begin

display <= {op,rs,rt,imm} ;

led <= {2'b00, ALUsrcB ,PCsource, PC\_we, IorD ,MemtoReg, MemWrite,IRWrite,RegDst,RegWrite,ALUop,ALUsrcA, zf};

end

3'b011:begin

led <= {2'b00, ALUsrcB ,PCsource, PC\_we, IorD ,MemtoReg, MemWrite,IRWrite,RegDst,RegWrite,ALUop,ALUsrcA, zf};

display <= mem\_reg\_data ;

end

3'b100:begin

display <= rd0 ;

led <= {2'b00, ALUsrcB ,PCsource, PC\_we, IorD ,MemtoReg, MemWrite,IRWrite,RegDst,RegWrite,ALUop,ALUsrcA, zf};

end

3'b101:begin

display <= rd1 ;

led <= {2'b00, ALUsrcB ,PCsource, PC\_we, IorD ,MemtoReg, MemWrite,IRWrite,RegDst,RegWrite,ALUop,ALUsrcA, zf};

end

3'b110:begin

display <= alu\_reg\_data ;

led <= {2'b00, ALUsrcB ,PCsource, PC\_we, IorD ,MemtoReg, MemWrite,IRWrite,RegDst,RegWrite,ALUop,ALUsrcA, zf};

end

default: begin

display <= 0;

led <= 0;

end

endcase

end

always@(posedge clk ,posedge rst)

begin

if (rst) begin

m\_rf\_addr <= 0;

end

else if (inc\_edg)

begin

m\_rf\_addr <= m\_rf\_addr + 4;

end

else if (dec\_edg)

begin

m\_rf\_addr <= m\_rf\_addr - 4;

end

end

endmodule

【实验总结】

1. 复习了多周期 CPU 的相关知识，并且进行了实践设计
2. 在根据数据通路，用 Verilog 编写 CPU 模块时，深切体会到了数据通路的电路级描述和 Verilog 语言级描述的区别：后者抽象程度更高

【附录】

本次提交目录结构组织如下：