实验报告

2021 年 6 月 26 日　　成绩：_____

姓名	刘合	学号		班级	
专业	计算机科学与技术		课程名称	计算机组成原理课程设计	
任课老师		指导老师		机位号	
实验序号	7，8，9	实验名称	基于 RISC-V 架构的 RV32I 指令集的 cpu 的设计实现		
实验时间	2021.6.26	实验地点		实验设备号	

一、实验目的与要求

1、 **实验目的：**

1. 掌握 RISC-V 的转移指令的数据通路设计，掌握指令流和数据流的控制方法;

2. 学习依据新增指令，修改多周期 CPU 的系统结构的方法，具备连接各模块构建整机的能力:

3. 掌握基于有限状态机， 实现控制单元的设计方法

4. 在实现 RISC-V 的 R 型和 I 型运算类指令、U 型传送类指令、访存指令的基础上，进一步实现转移类指令 beq、jal、jalr 的功能。

2、 实验要求：

1. 编写用于测试的汇编程序 RIUSJB_ test.s,可以自行设计，也可以采用前述范例程序将其翻译成机器指令代码，写入 RIUSJB_test.coe 文件备用。:

2. 将实验 8 的工程另存为一个新工程,用 RIUSJB_est.coe 重新生成指令存储器模块,用测试数据初始化数据存储器。

3. 设置 PC 多路选择器，新建 PCO 寄存器，添加相对转移地址加法器，改造 W_Data 的多路数据选择器为四选一数据选择器;修改二级指令译码模块 ID2 和控制单元 CU; 将它们与其他模块进行正确的连接，构建 RIUSJB_CPU 模块，如图 11.73 所示。

4. 对 RIUSJB_ CPU 模块进行仿真，在 rst_n 之后，每来一个 clk,观察指令执行的每步骤(状态)是否符合预期，确保 RIUSJB_CPU 能正确执行目标指令集上的测试程序。

5. 针对使用的实验板卡,设计 RIUSJB_CPU 的板级验证实验方案,编写顶层测试模块,要求至少能观察 PC、IR、W_ Data、MDR、标志位。这需要修改 RIUSJB_CPU 模块的输出

端口，将内部模块的相应信号引出。可参照实验 8 方案设计。

6. 复位后，按 clk 按键，会按步骤执行指令，验证每条指令需要几个周期执行完，执行结果是否正确，将实验结果记录到表 11.26 中，尤其记录转移指令。

7. 撰写实验报告，格式见附录，重点内容包括:对仿真结果进行分析:描述你设计的板级验证实验方案、模块结构与连接;说明你的板级操作过程;分析记录下来的板级实验结果、得到有效结论。请力所能及回答或实践本实验的“思考与探索”部分。

3.实验步骤

1. 编写用于测试的汇编程序 RIUSJB_test.s，可以自行设计，也可以采用前述范例程序;

2. 使用汇编器和反汇编器，将其翻译成机器指令代码,写入 RIUSJB_test.coe 文件备用;

3.新建数据存储器的 COE 文件 RIUSJB_data.coe 备用，初始化为想要测试的数据;

4. 将实验 8 的工程另存为一个新工程;

5. 用RIUSJB_test.coe 重新初始化指令存储器的 IP 核模块 IM,用RIUSJB_data.coe;重新初始化数据存储器;

6. 新建 PC0 寄存器，与 PC 正确连接;

7. 新建 PC 的三选一多路选择器、相对转移地址加法器，并与 PC、PC0、ImmU 等其他部件进行正确连接;

8.改造 W_Data 的多路数据选择器为四选一数据选择器;

9.修改二级指令译码模块 ID2，添加对 beq、jal 和 jalr 指令的译码;

10.修改控制单元 CU 的有限状态机，添加 beq、jal 和 jalr 指令的状态及其控制;

11. 将各模块正确连接，构建 RIUSJB_CPU 模块，如图 11.73 所示:

12.编写激励仿真代码，对 RIUSJB_CPU 模块进行仿真测试，首先 rst_n=0，然后置 1，之后，周期性地产生 clk,观察输出信号，分析指令执行的每一步骤(状态)是否符合预期。确保 RIUSJB_CPU 能正确执行目标指令集上的程序。

13.依据实际板卡情况，设计 RIUS_CPU 模块的板级验证实验方案，然后据此编写一个顶层测试模块。按照要求至少能观察 PC、IR、 MDR、写入寄存器堆的值 W_Data。

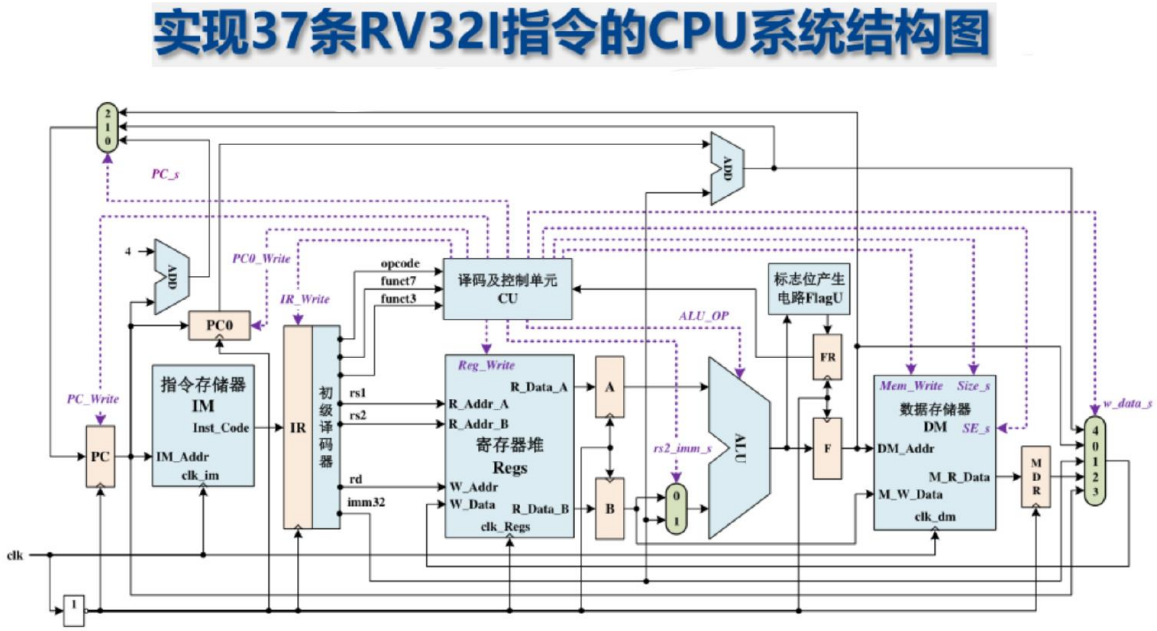
14.新建管脚约束文件，进行相应的引脚配置。

15. 生成*.bit 文件，下载到实验设备的 FPGA 芯片中。

16. 板级实验:按照你所设计的实验方案，操作输入设备、观察输出设备，预期的验证操作如下:

(1)按 rst_n 按键，将 PC 和 IR 清零，接下来将从 0 号单元开始执行指令;

(2)按动时钟键 clk,每按--下，指令就执行一步，观察 PC 和 IR 的值，判断每条指令需要几个 clk 才能执行完;观察各部件值，判断指令执行结果是否正确。



二、实验设计与程序代码

1、 模块设计说明

程序包含了 20 个模块 2 个存储器 IP 核：

1.CPU 顶层模块：有序调用其他模块的程序；

2.PC 地址模块：当前指令所在存储器中的地址

3.PC0 寄存模块：暂存当前还未+4 的指令 PC 值

4. IM 存储器 IP 核：存储指令数据

5. IR 模块：读取并暂存指令

6.ID1 初级译码模块：对指令进行初步分类

7.ID2 二级译码模块：对指令进行细分，处理特殊指令

8.CU 模块：根据传入的指令控制状态的转移和读写标志的输出

9.ImmU 立即数拼接:根据 opcode 拼接出对应指令的立即数

10.Reg 寄存器堆模块:保存参与运算的中间数据

11.zcA 暂存模块：暂存 A 的数据

12.zcB 暂存模块：暂存 B 的数据

13.ALU 运算模块：实现 10 种逻辑算数运算功能

14. zcF 暂存模块：暂存 F 的数据

15.PC_threeone 三选一数据选择模块：选择输出要输回到 PC 地址处的值

16.DM 存储器 IP 核：存储需要对其操作的数据

17.MDR 寄存器模块：暂存 MDR 的值

18.fourone2 四选一数据选择模块：根据信号选择要寄存到寄存器堆中的数据

19.FR 信号模块：输出显示 ZF,SF,CF,OF

20.sevenone 七选一数据选择模块：根据外部输入 SW 的值，选择要显示的数据

21.分频器模块：是为了给数码管扫描显示模块分配时钟频率；

22.数码管扫描显示模块：是为了把 BCD 码数据以十进制数字显示出来

2.实验程序源代码及注释等

```
module CPU_D(clk_1M,CLR,rst_n,clk,SW,FR,AN,seg/*Choice*/);
    input clk_1M;
    input CLR;
    input rst_n;
    input clk;
    input [2:0]SW;
    output [3:0]FR;
    output [7:0]AN;
    output [7:0]seg;
    wire clk_fim;
    wire PC_Write;
    wire [31:0]PC_in;
    wire [31:0]PC_Out;
    wire [31:0]PC_1;
    wire [31:0]Inst_Code;
    wire IR_Write;
    wire [31:0]inst;
```

```
wire [4:0]rs1;
wire [4:0]rs2;
wire [4:0]rd;
wire [6:0]I_fmt;
wire [6:0]opcode;
wire [2:0]funct3;
wire [6:0]funct7;
wire IS_R,IS_IMM,IS_LUI,IS_LW,IS_SW,IS_BEQ,IS_JAL,IS_JALR;
wire [3:0]ALU_OP_in;
wire Reg_Write;
wire [3:0]ALU_OP;
wire rs2_imm_s;
wire [1:0]w_data_s;
wire PC0_Write;
wire [1:0]PC_s;
wire [31:0]imm32;
wire [31:0]W_Data;
wire [31:0]R_Data_A;
wire [31:0]R_Data_B;
wire [31:0]ALU_A;
wire [31:0]ALU_B;
wire [31:0]B;
wire [31:0]ALU_F;
wire ZF,SF,CF,OF;
wire [31:0]F;
wire Mem_Write;
wire [31:0]M_R_Data;
wire [31:0]MDR_data;
wire [31:0]Choice;
//output [31:0]Choice;
```

```
  wire CLK;
  assign clk_fim=~clk;
  PC aa(rst_n,PC_Write,clk_fim,PC_in,PC_Out);
  PC0 tt(clk_fim,rst_n,PC0_Write,PC_Out,PC_1);
  RAM_B IM(         //存储器
.clka(clk),        // input wire clka
.addra(PC_Out[7:2]),    // input wire [5 : 0] addra
.douta(Inst_Code)    // output wire [31 : 0] douta
);
  IR bb(Inst_Code,clk_fim,rst_n,IR_Write,inst);
  ID1 cc(inst,rs1,rs2,rd,I_fmt,opcode,funct3,funct7);
  ID2 qq(opcode,funct3,funct7,IS_R,IS_IMM,IS_LUI,IS_LW,
            IS_SW,IS_BEQ,IS_JAL,IS_JALR,ALU_OP_in);
  CU dd(clk,rst_n,ZF,IS_R,IS_IMM,IS_LUI,IS_LW,IS_SW,IS_BEQ,IS_JAL,
            IS_JALR,ALU_OP_in,PC_Write,PC0_Write,IR_Write,Reg_Write,rs2_imm_s,
            w_data_s,Mem_Write,PC_s,ALU_OP);
  ImmU ee(funct3,inst,I_fmt,imm32);
  Reg ff(rst_n,rs1,rs2,rd,Reg_Write,W_Data,clk_fim,R_Data_A,R_Data_B);
  zcA gg(rst_n,R_Data_A,clk_fim,ALU_A);
  zcB hh(rst_n,R_Data_B,clk_fim,ALU_B);
  assign B=rs2_imm_s?imm32:ALU_B;
  ALU jj(ALU_OP,ALU_A,B,ALU_F,ZF,SF,CF,OF);
  zcF kk(rst_n,ALU_F,clk_fim,F);
  PC_threeone vv(PC_s,imm32,PC_Out,PC_1,F,PC_in);
  RAM_C DM(
.clka(clk),        // input wire clka
.wea(Mem_Write),           // input wire [0 : 0] wea
.addra(F[7:2]),    // input wire [5 : 0] addra
.dina(ALU_B),        // input wire [31 : 0] dina
.douta(M_R_Data)    // output wire [31 : 0] douta
```

```
);
    MDR rr(M_R_Data,clk_fim,rst_n,MDR_data);
    fourone2 ll(w_data_s,imm32,F,MDR_data,PC_Out,W_Data);
    FR mm(clk_fim,rst_n,ZF,SF,CF,OF,FR);
    sevenone nn(SW,PC_Out,inst,W_Data,ALU_A,ALU_B,F,MDR_data,Choice);
    fpq oo(CLR,clk_1M,CLK);
    smgsm pp(CLK,CLR,Choice,AN,seg);
endmodule

module PC(rst_n,PC_Write,clk_fim,PC_in,PC_Out);//程序计数器 PC
      input rst_n;
      input PC_Write;
      input clk_fim;
      input [31:0]PC_in;
      output reg [31:0]PC_Out;
      initial
          begin
                PC_Out<=0;
          end
      always@(posedge clk_fim or negedge rst_n)
      begin
           if(!rst_n)
                PC_Out<=0;
           else
                begin
                    if(PC_Write==1)
                          PC_Out<=PC_in;
                end
      end
endmodule
```

```
module PC0(clk_fim,rst_n,PC0_Write,PC_Out,PC_1);
	input clk_fim;
	input rst_n;
	input PC0_Write;
	input [31:0]PC_Out;
	output reg [31:0]PC_1;
	always@(posedge clk_fim or negedge rst_n)
	begin
		if(!rst_n)
			PC_1<=0;
		else
		  begin
			if(PC0_Write)
				PC_1<=PC_Out;
		  end
	end
endmodule

module IR(Inst_Code,clk_fim,rst_n,IR_Write,inst);
	input [31:0]Inst_Code;
	input clk_fim;
	input rst_n;
	input IR_Write;
	output reg [31:0]inst;
	always@(posedge clk_fim or negedge rst_n)
	begin
		if(!rst_n)
			inst<=0;
		else
```

```
            begin
                if(IR_Write==1)
                    inst<=Inst_Code;
            end
        end
endmodule

module ID1(inst,rs1,rs2,rd,I_fmt,opcode,funct3,funct7);
        input [31:0]inst;
        output [4:0]rs1;
        output [4:0]rs2;
        output [4:0]rd;
        output [6:0]I_fmt;
        output [6:0]opcode;
        output [2:0]funct3;
        output [6:0]funct7;
        wire R_Type,I_Type,S_Type,B_Type,U_Type,J_Type,JALR_Type;
    assign I_fmt={R_Type,I_Type,S_Type,B_Type,U_Type,J_Type,JALR_Type};
    `define OP_R    7'b0110011
    assign R_Type=(opcode==`OP_R);
    `define OP_I    7'b0010011
    `define OP_LW 7'b0000011
    assign I_Type=(opcode==`OP_I||opcode==`OP_LW);
    `define OP_S    7'b0100011
    assign S_Type=(opcode==`OP_S);
    `define OP_B    7'b1100011
    assign B_Type=(opcode==`OP_B);
    `define OP_U1    7'b0110111
    assign U_Type=(opcode==`OP_U1);
    `define OP_J    7'b1101111
```

```
    assign J_Type=(opcode==`OP_J);
    `define OP_JALR    7'b1100111
    assign JALR_Type=(opcode==`OP_JALR);
    assign opcode=inst[6:0];
    assign rs1=inst[19:15];
    assign rs2=inst[24:20];
    assign rd=inst[11:7];
    assign funct3=inst[14:12];
    assign funct7=inst[31:25];
endmodule

module ID2(opcode,funct3,funct7,IS_R,IS_IMM,IS_LUI,IS_LW,IS_SW,
                    IS_BEQ,IS_JAL,IS_JALR,ALU_OP_in);
        input [6:0]opcode;
        input [2:0]funct3;
        input [6:0]funct7;
        output reg IS_R;
        output reg IS_IMM;
        output reg IS_LUI;
        output reg IS_LW;
        output reg IS_SW;
        output reg IS_BEQ;
        output reg IS_JAL;
        output reg IS_JALR;
        output reg [3:0]ALU_OP_in;
          always@(*)
          begin
          case(opcode)
7'b0110011:begin //10R
      IS_R=1;IS_IMM=0;IS_LUI=0;IS_LW=0;IS_SW=0;IS_BEQ=0;IS_JAL=0;IS_JALR=0;
```

```
        ALU_OP_in={funct7[5],funct3};
                end
7'b0010011:begin//9I
        IS_IMM=1;IS_R=0;IS_LUI=0;IS_LW=0;IS_SW=0;IS_BEQ=0;IS_JAL=0;IS_JALR=0;
        ALU_OP_in=(funct3==3'b101)?{funct7[5],funct3}:{1'b0,funct3};
                end
7'b0110111:begin//lui
        IS_LUI=1;IS_R=0;IS_IMM=0;IS_LW=0;IS_SW=0;IS_BEQ=0;IS_JAL=0;IS_JALR=0;
        ALU_OP_in=4'b0000;
                end
7'b0000011:begin//lw
        IS_LW=1;IS_R=0;IS_IMM=0;IS_LUI=0;IS_SW=0;IS_BEQ=0;IS_JAL=0;IS_JALR=0;
        ALU_OP_in=4'b0000;
                end
7'b0100011:begin//sw
        IS_SW=1;IS_R=0;IS_IMM=0;IS_LUI=0;IS_LW=0;IS_BEQ=0;IS_JAL=0;IS_JALR=0;
        ALU_OP_in=4'b0000;
                end
7'b1100011:begin//beq
        IS_SW=0;IS_R=0;IS_IMM=0;IS_LUI=0;IS_LW=0;IS_BEQ=1;IS_JAL=0;IS_JALR=0;
        ALU_OP_in=4'b0000;
                end
7'b1101111:begin//jal
        IS_SW=0;IS_R=0;IS_IMM=0;IS_LUI=0;IS_LW=0;IS_BEQ=0;IS_JAL=1;IS_JALR=0;
        ALU_OP_in=4'b0000;
                end
7'b1100111:begin//jalr
        IS_SW=0;IS_R=0;IS_IMM=0;IS_LUI=0;IS_LW=0;IS_BEQ=0;IS_JAL=0;IS_JALR=1;
        ALU_OP_in=4'b0000;
                end
```

```
default:begin
        IS_R=0;IS_IMM=0;IS_LUI=0;IS_LW=0;IS_SW=0;IS_BEQ=0;IS_JAL=0;IS_JALR=0;
        ALU_OP_in=4'b0000;
                end
         endcase
         end
endmodule

module CU(clk,rst_n,ZF,IS_R,IS_IMM,IS_LUI,IS_LW,IS_SW,IS_BEQ,IS_JAL,
                IS_JALR,ALU_OP_in,PC_Write,PC0_Write,IR_Write,Reg_Write,
                rs2_imm_s,w_data_s,Mem_Write,PC_s,ALU_OP);
            input clk;
            input rst_n;
            input ZF;
            input IS_R,IS_IMM,IS_LUI,IS_LW,IS_SW,IS_BEQ,IS_JAL,IS_JALR;
            input [3:0]ALU_OP_in;
            output reg PC_Write;
            output reg PC0_Write;
            output reg IR_Write;
            output reg Reg_Write;
            output reg rs2_imm_s;
            output reg [1:0]w_data_s;
            output    Mem_Write;
            output reg [1:0]PC_s;
            output reg [3:0]ALU_OP;
            reg [3:0]ST;
            reg [3:0]Next_ST;
            parameter Idle=4'b0000,S1=4'b0001,S2=4'b0010,S3=4'b0011,
              S4=4'b0100,S5=4'b0101,S6=4'b0110,S7=4'b0111,S8=4'b1000,S9=4'b1001,
              S10=4'b1010,S11=4'b1011,S12=4'b1100,S13=4'b1101,S14=4'b1110;
```

```
//第一段：状态转移，在 clk 的边沿进行状态转移，是同步时序逻辑电路
always@(negedge rst_n or posedge clk)
begin
    if(!rst_n)//初始状态
       ST<=Idle;//初始状态
    else
       ST<=Next_ST;//clk 的上跳沿，进行状态转移
end
assign Mem_Write=(Next_ST==S10);
//第二段：次态函数，对次态的阻塞式赋值，是组合逻辑函数
always@(*)
begin
    Next_ST=Idle;
    case(ST) //根据状态转移图进行次态赋值
          Idle:Next_ST=S1;
          S1:begin
                   if(IS_LUI)Next_ST=S6;
                   else if(IS_JAL)Next_ST=S11;
                   else Next_ST=S2;
              end
          S2:begin
                   if(IS_R)Next_ST=S3;
                   else if(IS_IMM) Next_ST=S5;
                   else if(IS_BEQ) Next_ST=S13;
                   else Next_ST=S7;
              end
          S3:begin
                   Next_ST=S4;
              end
          S4:begin
```

```
            Next_ST=S1;
        end
    S5:begin
            Next_ST=S4;
        end
    S6:begin
            Next_ST=S1;
        end
    S7:begin
            if(IS_LW) Next_ST=S8;
            else if(IS_SW) begin Next_ST=S10;end
            else Next_ST=S12;
        end
    S8:begin
            Next_ST=S9;
        end
    S9:begin
            Next_ST=S1;
        end
    S10:begin
            Next_ST=S1;
        end
    S11:begin
            Next_ST=S1;
        end
    S12:begin
            Next_ST=S1;
        end
    S13:begin
            Next_ST=S14;
```

```
                end
            S14:begin
                    Next_ST=S1;
                end
            default:Next_ST=S1;
        endcase
    end
    //第三段：输出函数，在 clk 的上跳沿，根据下一状态进行控制信号的非阻塞式赋值
    always@(negedge rst_n or posedge clk)
    begin
        if(!rst_n)      //全部信号初始化为 0
        begin
            PC_Write<=1'b0;   PC0_Write<=1'b0; IR_Write<=1'b0;   Reg_Write<=1'b0;
            ALU_OP<=4'b0000; rs2_imm_s<=1'b0; w_data_s<=2'b00; PC_s<=2'b00;
        end
        else
        begin
            case(Next_ST)
                Idle:begin
    PC_Write<=1'b0; PC0_Write<=1'b0; IR_Write<=1'b0;     Reg_Write<=1'b0; //初始状态
    ALU_OP<=4'b0000;   rs2_imm_s<=1'b0;   w_data_s<=2'b00;   PC_s<=2'b00;
                    end
                S1:begin
    PC_Write<=1'b1;     PC0_Write<=1'b1;     IR_Write<=1'b1;     Reg_Write<=1'b0;
    ALU_OP<=4'b0000;                rs2_imm_s<=1'b0;                w_data_s<=2'b00;
PC_s<=2'b00;//ALU_OP,rs2_imm_s,w_data_s 无效，取任意值
                    end
                S2:begin
PC_Write<=1'b0;     PC0_Write<=1'b0;     IR_Write<=1'b0;     Reg_Write<=1'b0;
```

```
ALU_OP<=4'b0000;  rs2_imm_s<=1'b0;  w_data_s<=2'b00;  PC_s<=2'b00;//下面 4 个都无效，取任意值
                    end
                S3:begin
PC_Write<=1'b0;  PC0_Write<=1'b0;  IR_Write<=1'b0;  Reg_Write<=1'b0;
ALU_OP<=ALU_OP_in;  rs2_imm_s<=1'b0;  w_data_s<=2'b00;
PC_s<=2'b00;//w_data_s,PC_s 无效，取任意值
                    end
                S4:begin
PC_Write<=1'b0;  PC0_Write<=1'b0;  IR_Write<=1'b0;  Reg_Write<=1'b1;
ALU_OP<=4'b0000;  rs2_imm_s<=1'b0;  w_data_s<=2'b00;
PC_s<=2'b00;//ALU_OP,rs2_imm_s,PC_s 无效，取任意值
                    end
                S5:begin
 PC_Write<=1'b0;  PC0_Write<=1'b0;  IR_Write<=1'b0;  Reg_Write<=1'b0;
 ALU_OP<=ALU_OP_in;  rs2_imm_s<=1'b1;  w_data_s<=2'b00;
PC_s<=2'b00;//w_data_s,PC_s 无效，取任意值
                    end
                S6:begin
PC_Write<=1'b0;  PC0_Write<=1'b0;  IR_Write<=1'b0;  Reg_Write<=1'b1;
ALU_OP<=4'b0000;  rs2_imm_s<=1'b0;  w_data_s<=2'b01;
PC_s<=2'b00;//ALU_OP,rs2_imm_s,PC_s 无效，取任意值
                    end
                S7:begin
PC_Write<=1'b0;  PC0_Write<=1'b0;  IR_Write<=1'b0;  Reg_Write<=1'b0;
ALU_OP<=4'b0000;  rs2_imm_s<=1'b1;  w_data_s<=2'b00;
PC_s<=2'b00;//w_data_s,PC_s 无效，取任意值
                    end
                S8:begin
PC_Write<=1'b0;  PC0_Write<=1'b0;  IR_Write<=1'b0;  Reg_Write<=1'b0;
```

```
ALU_OP<=4'b0000;  rs2_imm_s<=1'b0;  w_data_s<=2'b00;  PC_s<=2'b00;//下面 4 个都无效，取任意值
                    end
                S9:begin
PC_Write<=1'b0;  PC0_Write<=1'b0;  IR_Write<=1'b0;  Reg_Write<=1'b1;
ALU_OP<=4'b0000;  rs2_imm_s<=1'b0;  w_data_s<=2'b10;
PC_s<=2'b00;//ALU_OP,rs2_imm_s,PC_s 无效，取任意值
                    end
                S10:begin
PC_Write<=1'b0;  PC0_Write<=1'b0;  IR_Write<=1'b0;  Reg_Write<=1'b0;
ALU_OP<=4'b0000;  rs2_imm_s<=1'b0;  w_data_s<=2'b00;  PC_s<=2'b00;//下面 4 个都无效，取任意值
                    end
                S11:begin
PC_Write<=1'b1;  PC0_Write<=1'b0;  IR_Write<=1'b0;  Reg_Write<=1'b1;
ALU_OP<=4'b0000;  rs2_imm_s<=1'b0;  w_data_s<=2'b11;
PC_s<=2'b01;//ALU_OP,rs2_imm_s 无效，取任意值
                    end
                S12:begin
PC_Write<=1'b1;  PC0_Write<=1'b0;  IR_Write<=1'b0;  Reg_Write<=1'b1;
ALU_OP<=4'b0000;  rs2_imm_s<=1'b0;  w_data_s<=2'b11;
PC_s<=2'b10;//ALU_OP,rs2_imm_s 无效，取任意值
                    end
                S13:begin
PC_Write<=1'b0;  PC0_Write<=1'b0;  IR_Write<=1'b0;  Reg_Write<=1'b0;
ALU_OP<=4'b1000;  rs2_imm_s<=1'b0;  w_data_s<=2'b00;  PC_s<=2'b00;//w_data_s，PC_s 无效，取任意值
                    end
                S14:begin
PC_Write<=ZF;  PC0_Write<=1'b0;  IR_Write<=1'b0;  Reg_Write<=1'b0;
```

```
ALU_OP<=4'b0000;                    rs2_imm_s<=1'b0;                    w_data_s<=2'b00;
PC_s<=2'b01;//ALU_OP,rs2_imm_s,w_data_s 无效，取任意值
                                end
                endcase
            end
        end
endmodule

module ImmU(funct3,inst,I_fmt,imm32);
    input [2:0]funct3;
    input [31:0]inst;
    input [6:0]I_fmt;
    output reg [31:0]imm32;
    wire [31:0]I_imm;
    wire [31:0]S_imm;
    wire [31:0]B_imm;
    wire [31:0]U_imm;
    wire [31:0]J_imm;
    wire [31:0]JALR_imm;
    assign I_imm={{20{inst[31]}},inst[31:20]};
    assign S_imm={{20{inst[31]}},inst[31:25],inst[11:7]};
    assign B_imm={{20{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0};
    assign U_imm={inst[31:12],{12{1'b0}}};
    assign J_imm={{12{inst[31]}},inst[19:12],inst[20],inst[30:21],1'b0};
    always@(*)
        begin
            case(I_fmt)
                7'b0100000: begin //I
                                if(funct3==3'b101||funct3==3'b001)
                                    imm32={{27{1'b0}},inst[24:20]};
```

```
                                    else
                                        imm32=I_imm;
                                end
                        7'b0010000: imm32=S_imm;//S
                        7'b0001000: imm32=B_imm;//B
                        7'b0000100: imm32=U_imm;//U
                        7'b0000010: imm32=J_imm;//J
                        7'b0000001: begin
                                    if(funct3==3'b101||funct3==3'b001)
                                        imm32={{27{1'b0}},inst[24:20]};
                                    else
                                        imm32=I_imm;
                                end
                        default:imm32=0;
                endcase
        end
endmodule

module
Reg(rst_n,R_Addr_A,R_Addr_B,W_Addr,Reg_Write,W_Data,clk_fim,R_Data_A,R_Data_B
);
    input rst_n;
    input [4:0]R_Addr_A;
    input [4:0]R_Addr_B;
    input [4:0]W_Addr;
    input Reg_Write;
    input [31:0]W_Data;
    input clk_fim;
    output [31:0]R_Data_A;
    output [31:0]R_Data_B;
```

```
    integer i;
    wire clk;
    reg[31:0]REG_Files[0:31];//寄存器阵列
    assign R_Data_A=REG_Files[R_Addr_A];//读 A 操作
    assign R_Data_B=REG_Files[R_Addr_B];//读 B 操作
    always@(posedge clk_fim or negedge rst_n)
    begin
        if(!rst_n)
                begin
                    for(i=0;i<31;i=i+1) REG_Files[i]<=32'h0000_0000;//初始化 32 寄存器
                end
        else
            begin
                if(Reg_Write&&W_Addr!=0)//上跳沿
                  REG_Files[W_Addr]<=W_Data;//写入寄存器
            end
    end
endmodule

module zcA(rst_n,Data_A,clk_fim,ALU_A);//A 暂存器
    input rst_n;
    input [31:0]Data_A;
    input clk_fim;
    output reg [31:0]ALU_A;
    always@(negedge rst_n or posedge clk_fim)
    begin
        if(!rst_n)
            ALU_A<=32'b0;
        else
            ALU_A<=Data_A;
```

```
    end
endmodule

module zcB(rst_n,Data_B,clk_fim,ALU_B);//B 暂存器
    input rst_n;
    input [31:0]Data_B;
    input clk_fim;
    output reg [31:0]ALU_B;
    always@(negedge rst_n or posedge clk_fim)
    begin
        if(!rst_n)
            ALU_B<=32'b0;
        else
            ALU_B<=Data_B;
    end
endmodule

module ALU(ALU_OP,ALU_A,ALU_B,ALU_F,ZF,SF,CF,OF); //ALU 功能模块
    input [3:0]ALU_OP;
    input [31:0]ALU_A;
    input [31:0]ALU_B;
    output [31:0]ALU_F;
    output   ZF;//是否为 0，结果为 0 时，ZF=1;否则，ZF=0
    output   SF;//符号标志位，与运算结果的最高位相同，正数时，SF=0
    output   CF;//进位/借位，最高位产生的进位 C32，加法：C32=1 时，CF=1;减法：C32=0
时，CF=1   无符号
    output   OF;//溢出标志位，有溢出，OF=1；带符号
    reg [31:0]ALU_F;
    reg C32;//最高进位
    reg ZF,SF,CF,OF;
```

```
    always@(*)
    begin
        C32=0;
        case(ALU_OP)
                4'b0000:begin                {C32,ALU_F}=ALU_A+ALU_B;                end
//加法
                4'b0001:begin                  ALU_F=ALU_A<<ALU_B;                   end
//左移
                4'b0010:begin      ALU_F=($signed(ALU_A)<$signed(ALU_B))?1:0;      end
//有符号数比较
                4'b0011:begin                  ALU_F=(ALU_A<ALU_B)?1:0;              end
//无符号数比较
                4'b0100:begin                  ALU_F=ALU_A^ALU_B;                    end
//异或
                4'b0101:begin                  ALU_F=ALU_A>>ALU_B;                   end
//逻辑右移，高位补零
                4'b0110:begin                  ALU_F=ALU_A|ALU_B;                    end
//按位或
                4'b0111:begin                  ALU_F=ALU_B&ALU_A;                    end
//按位与
                4'b1000:begin                {C32,ALU_F}=ALU_A-ALU_B;                end
//减法
                //4'b1101:begin                ALU_F=ALU_A>>>ALU_B;                  end
//算术右移，高位补 ALU_A[31]
                4'b1101:begin ALU_F=($signed(ALU_A))>>>ALU_B; end
        endcase
         ZF = ALU_F==0;//F 全为 0，则 ZF=1
         SF = ALU_F[31];//符号标志,取 F 的最高位
         CF = C32; //进位借位标志
```

```
            OF = ALU_A[31]^ALU_B[31]^ALU_F[31]^C32;//溢出标志公式
        end
endmodule

module zcF(rst_n,ALU_F,clk_fim,F);//F 暂存器
        input rst_n;
        input [31:0]ALU_F;
        input clk_fim;
        output reg [31:0]F;
        always@(negedge rst_n or posedge clk_fim)
        begin
            if(!rst_n)
                F<=32'b0;
            else
                F<=ALU_F;
        end
endmodule

module PC_threeone(PC_s,imm32,PC_Out,PC_1,F,PC_in);
          input [1:0]PC_s;
          input [31:0]imm32;
          input [31:0]PC_Out;
          input [31:0]PC_1;
          input [31:0]F;
          output reg [31:0]PC_in;
          always@(*)
          begin
              case(PC_s)
                        2'b00:begin PC_in=PC_Out+4; end
                        2'b01:begin PC_in=PC_1+imm32; end
```

```
                2'b10:begin PC_in=F; end
            endcase
        end
endmodule

module MDR(M_R_Data,clk_fim,rst_n,MDR_data);
    input [31:0]M_R_Data;
    input clk_fim;
    input rst_n;
    output reg [31:0]MDR_data;
    always@(negedge rst_n or posedge clk_fim)
    begin
        if(!rst_n)
            MDR_data<=32'b0;
        else
            MDR_data<=M_R_Data;
    end
endmodule

module fourone2(w_data_s,imm32,F,MDR_data,PC_Out,W_Data);
    input [1:0]w_data_s;
    input [31:0]imm32;
    input [31:0]F;
    input [31:0]MDR_data;
    input [31:0]PC_Out;
    output reg [31:0]W_Data;
    always@(*)
    begin
        case(w_data_s)
                2'b00:begin W_Data=F; end
```

```
                2'b01:begin W_Data=imm32; end
                2'b10:begin W_Data=MDR_data; end
                2'b11:begin W_Data=PC_Out; end
            endcase
    end
endmodule

module FR(clk_fim,rst_n,ZF,SF,CF,OF,FR);//FR 标志寄存器
    input clk_fim;
    input rst_n;
    input ZF,SF,CF,OF;
    output reg [3:0]FR;
    always@(negedge rst_n or posedge clk_fim)
    begin
        if(!rst_n)
            FR<=4'b0;
        else
            begin
                FR[3]=ZF;
                FR[2]=SF;
                FR[1]=CF;
                FR[0]=OF;
            end
    end
endmodule

module sevenone(SW,PC_Out,inst,W_Data,ALU_A,ALU_B,F,MDR_data,Choice);
      input [2:0]SW;
      input [31:0]PC_Out;
      input [31:0]inst;
```

```
    input [31:0]W_Data;
    input [31:0]ALU_A;
    input [31:0]ALU_B;
    input [31:0]F;
    input [31:0]MDR_data;
    output reg [31:0]Choice;
    always@(*)
    begin
        case(SW)
                3'b000:begin Choice=PC_Out; end
                3'b001:begin Choice=inst; end
                3'b010:begin Choice=W_Data; end
                3'b011:begin Choice=ALU_A; end
                3'b100:begin Choice=ALU_B; end
                3'b101:begin Choice=F; end
                3'b110:begin Choice=MDR_data; end
                default:begin Choice=PC_Out;end
        endcase
    end
endmodule

module fpq(CLR,clk_1M,CLK);              //分频器模块
   input CLR;
   input clk_1M;
   output reg CLK;
   reg [8:0]counter;
   always@(negedge CLR or posedge clk_1M)
      begin
         if(!CLR)
           begin
```

```
            counter<=9'd0;
            CLK<=1'b0;
        end
    else if(counter==9'd500)
            begin
                CLK<=~CLK;
                counter<=9'd0;
            end
    else
            counter<=counter+1'b1;
end
endmodule

module smgsm(CLK,CLR,Choice,AN,seg);//数码管扫描显示模块
    input CLK; //扫描的间隔始终，控制数码管轮流点亮的频率
    input CLR; //复位信号，用于初始化状态
    input[31:0]Choice;
    output reg [7:0]AN;//位选
    output reg [7:0]seg; //段选
    reg [3:0] data;
    reg [2:0] bit_sel;   //数码管计数器指示，000~111  最左到最右
    always@(negedge CLR or posedge CLK)
    begin
        if(!CLR)
                bit_sel<=3'b000;
        else
            bit_sel<=bit_sel+1'b1;
    end
    always@(*)
        begin
```

```
        case(bit_sel)
            3'b000:data<=Choice[3:0];
            3'b001:data<=Choice[7:4];
            3'b010:data<=Choice[11:8];
            3'b011:data<=Choice[15:12];
            3'b100:data<=Choice[19:16];
            3'b101:data<=Choice[23:20];
            3'b110:data<=Choice[27:24];
            3'b111:data<=Choice[31:28];
            default:data<=data;
        endcase
    end
always@(*)
    begin
        case(bit_sel)
            3'b000:AN=8'b11111110;
            3'b001:AN=8'b11111101;
            3'b010:AN=8'b11111011;
            3'b011:AN=8'b11110111;
            3'b100:AN=8'b11101111;
            3'b101:AN=8'b11011111;
            3'b110:AN=8'b10111111;
            3'b111:AN=8'b01111111;
            default:AN=8'b11111111;
        endcase
    end
always@(*)
    begin
        case(data)
                0:seg<=8'b00000011;
```

```
                1:seg<=8'b10011111;
                2:seg<=8'b00100101;
                3:seg<=8'b00001101;
                4:seg<=8'b10011001;
                5:seg<=8'b01001001;
                6:seg<=8'b01000001;
                7:seg<=8'b00011111;
                8:seg<=8'b00000001;
                9:seg<=8'b00001001;
                10:seg<=8'b00010001;
                11:seg<=8'b11000001;
                12:seg<=8'b01100011;
                13:seg<=8'b10000101;
                14:seg<=8'b01100001;
                15:seg<=8'b01110001;
            default:seg<=8'b11111111;
        endcase
    end
endmodule
```

三、实验仿真

1、 仿真代码

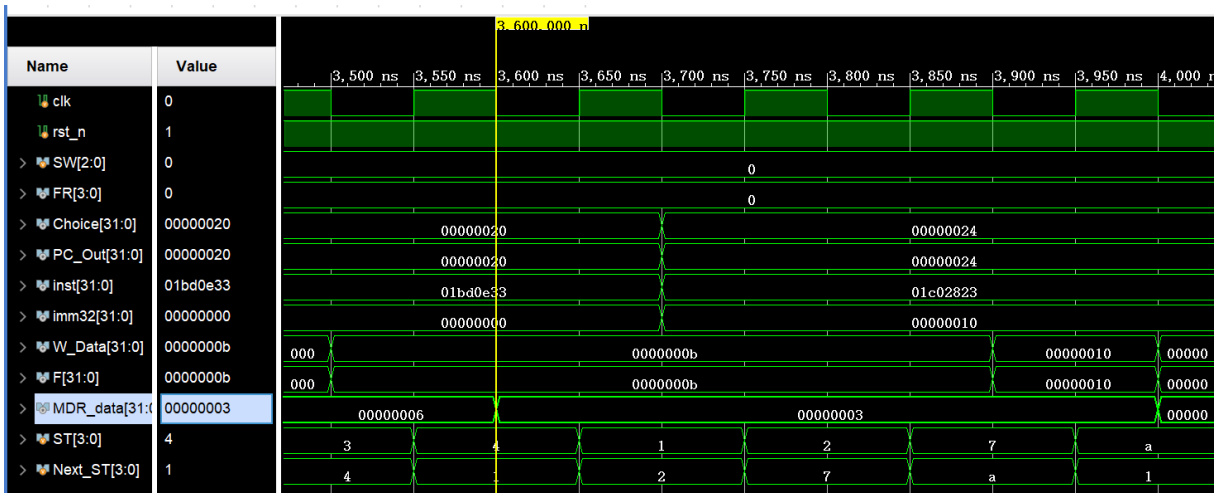
```
module CPU7fz();
//INPUT
reg clk;
reg rst_n;
reg [2:0]SW;
//OUTPUT
wire [3:0]FR;
wire [31:0]Choice;
```

```
initial
begin
    rst_n=0;clk=0;SW=3'b000;
    #25 rst_n=1;
end
always #50 clk=~clk;
CPU_D    ss(/*clk_1M,CLR,*/rst_n,clk,SW,FR,/*AN,seg*/Choice);
endmodule
```

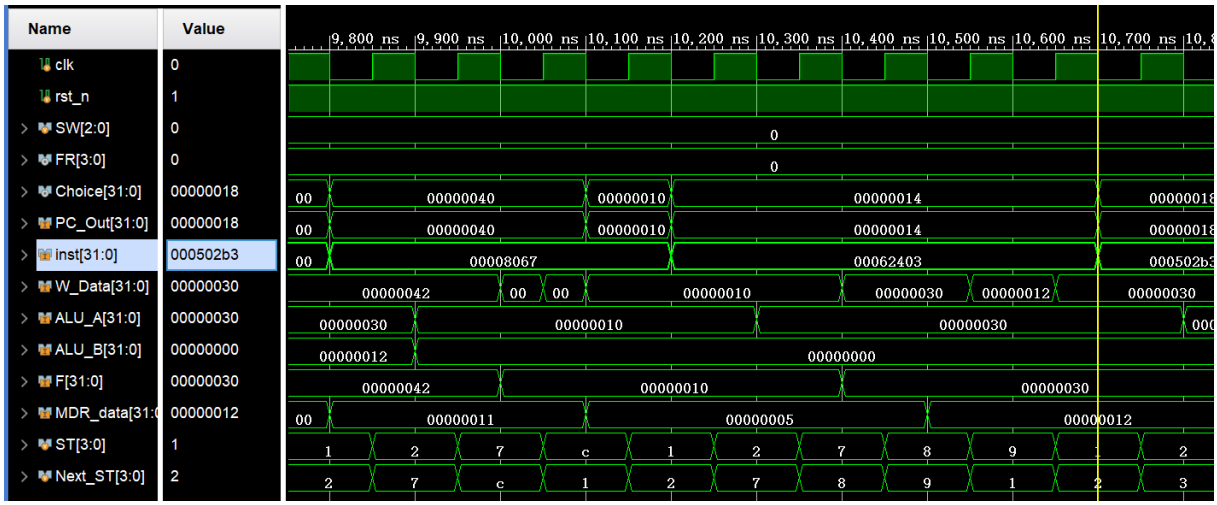
2、 仿真波形

（运行仿真时，波形截图）

实验八：



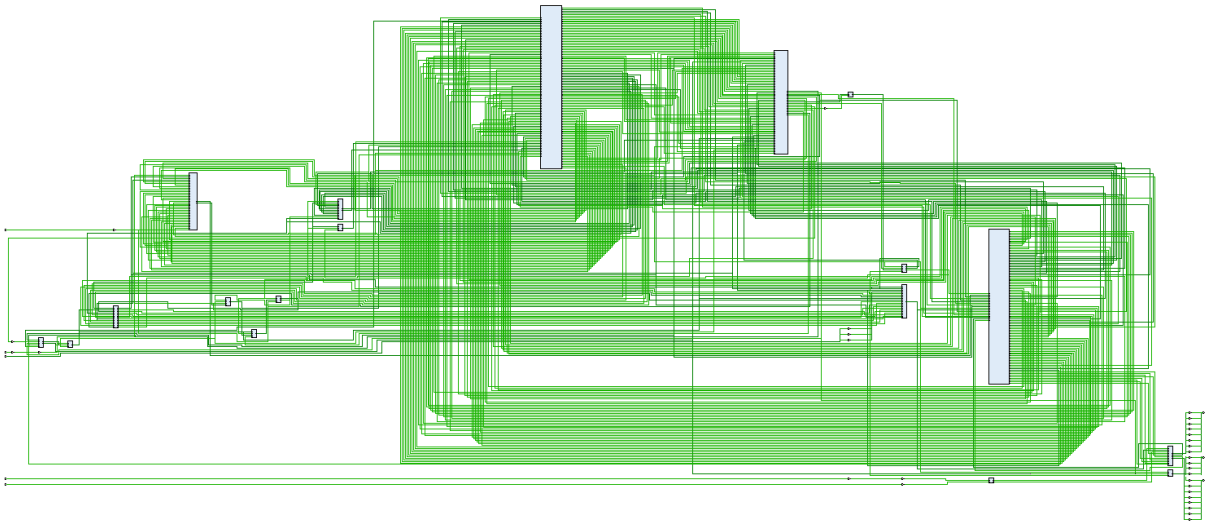
实验九：



3、 仿真结果分析

指令功能都符合预期的效果。逻辑成功。

四、电路图



五、引脚配置

（引脚约束文件的内容，描述主要配置情况）

set_property IOSTANDARD LVCMOS18 [get_ports clk]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[7]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[6]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[5]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[4]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[3]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[2]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[1]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[0]}]

set_property IOSTANDARD LVCMOS18 [get_ports {FR[3]}]

set_property IOSTANDARD LVCMOS18 [get_ports {FR[2]}]

set_property IOSTANDARD LVCMOS18 [get_ports {FR[1]}]

set_property IOSTANDARD LVCMOS18 [get_ports {FR[0]}]

set_property IOSTANDARD LVCMOS18 [get_ports {seg[7]}]

set_property IOSTANDARD LVCMOS18 [get_ports {seg[6]}]

```
set_property IOSTANDARD LVCMOS18 [get_ports {seg[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {SW[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports clk_1M]
set_property IOSTANDARD LVCMOS18 [get_ports CLR]
set_property IOSTANDARD LVCMOS18 [get_ports rst_n]
set_property PACKAGE_PIN E3 [get_ports clk_1M]
set_property PACKAGE_PIN N17 [get_ports CLR]
set_property PACKAGE_PIN U13 [get_ports {FR[3]}]
set_property PACKAGE_PIN R13 [get_ports {FR[2]}]
set_property PACKAGE_PIN U14 [get_ports {FR[1]}]
set_property PACKAGE_PIN T15 [get_ports {FR[0]}]
set_property PACKAGE_PIN C9 [get_ports {AN[7]}]
set_property PACKAGE_PIN C10 [get_ports {AN[6]}]
set_property PACKAGE_PIN D10 [get_ports {AN[5]}]
set_property PACKAGE_PIN C11 [get_ports {AN[4]}]
set_property PACKAGE_PIN M17 [get_ports {AN[3]}]
set_property PACKAGE_PIN J14 [get_ports {AN[2]}]
set_property PACKAGE_PIN K13 [get_ports {AN[1]}]
set_property PACKAGE_PIN P14 [get_ports {AN[0]}]
set_property PACKAGE_PIN F14 [get_ports {seg[7]}]
set_property PACKAGE_PIN N14 [get_ports {seg[6]}]
set_property PACKAGE_PIN J13 [get_ports {seg[5]}]
set_property PACKAGE_PIN G13 [get_ports {seg[4]}]
```

```
set_property PACKAGE_PIN F13 [get_ports {seg[3]}]
set_property PACKAGE_PIN G14 [get_ports {seg[2]}]
set_property PACKAGE_PIN M13 [get_ports {seg[1]}]
set_property PACKAGE_PIN H14 [get_ports {seg[0]}]
set_property PACKAGE_PIN U18 [get_ports rst_n]
set_property PACKAGE_PIN U17 [get_ports clk]
set_property PACKAGE_PIN V5 [get_ports {SW[2]}]
set_property PACKAGE_PIN T4 [get_ports {SW[1]}]
set_property PACKAGE_PIN V6 [get_ports {SW[0]}]
set_property PULLDOWN true [get_ports clk_1M]
set_property PULLDOWN true [get_ports CLR]
set_property PULLDOWN true [get_ports rst_n]
set_property PULLDOWN true [get_ports {SW[2]}]
set_property PULLDOWN true [get_ports {SW[1]}]
set_property PULLDOWN true [get_ports {SW[0]}]
set_property PULLDOWN true [get_ports clk]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_1M]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets CLR]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets rst_n]
```

六、思考与探索

1、 实验结果记录：

序号	PC	IR	汇编指令	预期执行结果	指令 clk 数	W_Data	MDR
1	0	01000f93	li t6,16	00000000	4	00000000	00000005
2	4	000fad03	lw s10,0(t6)	00000005	5	00000010	00000005
3	8	004fad83	lw s11,4(t6)	00000006	5	00000010	00000006
4	c	01bfa023	sw s11,0(t6)	00000006	4	00000016	00000006
5	10	01afa223	sw s10,4(t6)	00000005	4	00000015	00000005
6	14	000fad03	lw s10,0(t6)	00000006	5	00000010	00000006
7	18	004fad83	lw s11,4(t6)	00000005	5	00000010	00000005

（实验操作的过程及结果记录）

实验 8 结果记录：

8	1c	01bd0e33	add t3,s10,s11	0000000b	4	00000000b	00000003

序号	PC	IR	汇编指令	执行结果	下条指令地址

实验 9 结果记录：

1	0	01000513	li a0,16	00000000	4
2	4	00306593	ori a1,zero,3	00000003	8
3	8	03004613	xori a2,zero,48	00000000	c
4	c	008000ef	jal ra,14	00000014	14
5	14	000502b3	add t0,a0,zero	00000005	18
6	20	0002ae03	lw t3,0(t0)	00000005	24
7	34	fedff06f	j 20	00000020	20
8	30	00030463	beqz t1,38	00000038	38
9	3c	00008067	ret	00000010	10

2、 实验结论：

实验结果符合预期结果，逻辑正确，实验成功。

3、 问题与解决方案：

问题：立即数拼接有问题，有些特殊指令的立即数拼接未考虑到，没有做相应的处理和识别。

结局方案：添加了三条 I 特殊指令的识别代码，成功解决此问题。

4、 思考题：

1. 你的 CPU 能否正确执行新增的三条转移指令?以你的测试程序为例，举例说明。

答：可以正确执行三条转移指令，两次从地址 34 跳回地址 20，一次从地址 40 跳回地址 10。

5. 说说你在实验中碰到了哪些问题，你是如何解决的?

答：问题：立即数拼接有问题，有些特殊指令的立即数拼接未考虑到，没有做相应的处理和识别。

结局方案：添加了三条 I 特殊指令的识别代码，成功解决此问题。