**实验报告**

年 月 日 成绩：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 王坤 | 学号 | 17051225 | 班级 | 17052312 |
| 专业 | 计算机科学与技术 | | 课程名称 |  | |
| 任课老师 | 章复嘉 | 指导老师 |  | 机位号 |  |
| 实验序号 | 实验10 | 实验名称 | R\_I\_J型CPU的实现 | | |
| 实验时间 |  | 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MIPS R型、I型和J型指令的综合数据通路设计**    2. **掌握各种转移类指令的控制流和指令流的多路选通控制方法；**    3. **掌握J型、I型和R型转移指令的指令格式和寻址方式，学习转移地址的产生方法**    4. **掌握无条件转移指令和条件转移指令的实现方法；**    5. **编程实现MIPS的部分J型、I型和R型转移指令的功能** | | | | | |
| **二、实验设计与程序代码** | | | | | |
| 1. 模块设计说明   本实验是R\_I\_J型实验,相关模块如ALU,寄存器堆模块在前面的实验报告已经给出,本实验报告只给出顶层模块和译码器模块   1. 实验程序源代码及注释等   //顶层模块  `timescale 1ns / 1ps  module CPU\_R\_I\_J(SW,Output\_Data,ALU\_F);    parameter ADDR = 5;//地址位宽  parameter SIZE = 32;//数据位宽  //输出数据  output [SIZE:1] Output\_Data;  output [32:1] ALU\_F;      //时钟clk  input [5:0] SW;  //指令地址  reg [SIZE:1] PC;  wire [SIZE:1] PC\_new;    wire [SIZE:1] ZOF;  //寄存器堆地址  reg [ADDR:1]R\_Addr\_A;//A读端口寄存器地址  reg [ADDR:1]R\_Addr\_B;//B读端口寄存器地址n  wire [ADDR:1]W\_Addr;//写寄存器地址  wire [SIZE:1]W\_Data;    wire [SIZE:1]R\_Data\_A;//A端口读出数据  wire [SIZE:1]R\_Data\_B;//B端口读出数据    //ALU  wire [SIZE:1] ALU\_F,ALU\_B;  wire CF,//进借位标志  SF,//符号标志  PF,//奇偶标志uijm  ZF,//零标志  OF;//溢出标志    //译码以及控制单元信号  wire Write\_Reg; //寄存器写入信号  wire [3:0] ALU\_OP;//运算符编码  wire Mem\_Write,rt\_imm\_s,imm\_s;  wire [1:0] PC\_s,alu\_mem\_s,rd\_rt\_s;  //指令存储器  wire [SIZE:1] Inst\_code,imm\_data;  //数据存储器  wire [SIZE:1] M\_W\_Data,M\_R\_Data;  wire [6:1] Mem\_Addr;  //指令类型  wire [6:1] OP; //公用OP  wire [5:1] rs,rt,rd,shamt; //R型指令  wire [6:1] func; //R型指令功能  wire [16:1] imm; //    //实例化寄存器堆模块  RegFile RF\_Test(  .Clk(~SW[0]),  .reset(SW[1]),  .Write\_Reg(Write\_Reg), //译码控制给出  .R\_Addr\_A(Inst\_code[26:22]),  .R\_Addr\_B(Inst\_code[21:17]),  .W\_Addr(W\_Addr),  .W\_Data(W\_Data),  .R\_Data\_A(R\_Data\_A),  .R\_Data\_B(R\_Data\_B)  );    //实例化ALU模块  ALU ALU\_Test(  .OP(ALU\_OP),//运算符  .A(R\_Data\_A),//A操作数  .B(ALU\_B),//B操作数  .F(ALU\_F),//ALU\_F作为中间变量    .ZF(ZF),//零标志  .CF(CF),//进借位标志  .OF(OF),//溢出标志  .SF(SF),//符号标志  .PF(PF)//奇偶标志  );      //指令译码器    yimaqi yimaqi (  .Inst\_code(Inst\_code),  .OP(Inst\_code[32:27]),  .func(Inst\_code[6:1]),  .write\_reg(Write\_Reg),  .ALU\_OP(ALU\_OP),  .Mem\_Write(Mem\_Write),  .alu\_mem\_s(alu\_mem\_s), //wr\_data\_s  .rt\_imm\_s(rt\_imm\_s),  .imm\_s(imm\_s),  .rd\_rt\_s(rd\_rt\_s), //w-r-s  .PC\_s(PC\_s)  );  //PC模块  Get\_Inst get\_inst(  .inst\_code(Inst\_code),  .clk(SW[0]),  .PC(PC)  );    //数据存储器  RAM\_B RAM\_B (  .clka(SW[0]), // input clka  .wea(Mem\_Write), // input [0 : 0] wea  .addra(ALU\_F[6:1]), // input [5 : 0] addra  .dina(M\_W\_Data), // input [31 : 0] dina  .douta(M\_R\_Data) // output [31 : 0] douta  );  // assign Mem\_Addr = ALU\_F;  assign W\_Data = (alu\_mem\_s[1])?(PC\_new):((alu\_mem\_s[0])?M\_R\_Data:ALU\_F);  assign M\_W\_Data = R\_Data\_B;  assign W\_Addr = (rd\_rt\_s[1])?5'b11111:((rd\_rt\_s[0])?Inst\_code[21:17]:Inst\_code[16:12]);  assign imm\_data = (imm\_s)?{{16{Inst\_code[16]}},Inst\_code[16:1]}:{{16{1'b0}},Inst\_code[16:1]};  assign ALU\_B = (rt\_imm\_s)?imm\_data:R\_Data\_B;    assign PC\_new = PC+4;;    assign Output\_Data=W\_Data;  // always@(posedge SW[2] or posedge SW[3] or posedge SW[4])  //begin  // if(SW[2]) begin Output\_Data<=W\_Data; end  //else if(SW[3]) begin Output\_Data<={30'b0,OF,ZF}; end  //else if(SW[4]) begin Output\_Data<=ALU\_F; end  //end      always@(negedge SW[0])  begin  if(SW[1]) begin PC<=0; end  else begin  case(PC\_s)  2'b00 : PC <=PC\_new;  2'b01 : PC <=R\_Data\_A;  2'b10 : PC <=PC\_new+(imm\_data<<2);  2'b11 : PC <={PC\_new[32:29],Inst\_code[26:1],2'b00};  endcase  end  end  endmodule  //译码器模块  `timescale 1ns / 1ps  module yimaqi(Inst\_code,OP,func,write\_reg,ALU\_OP,Mem\_Write,alu\_mem\_s,rt\_imm\_s,imm\_s,rd\_rt\_s,PC\_s);  input [31:0] Inst\_code;  input [5:0]OP;  input [5:0]func;  output reg write\_reg;  output reg [3:0]ALU\_OP;  output reg Mem\_Write,rt\_imm\_s,imm\_s;  output reg [1:0] alu\_mem\_s,rd\_rt\_s,PC\_s;  always@(\*)  begin  if(OP==6'b000000)  begin  case(func)  6'b001000 : begin ALU\_OP <= 4'b0100;write\_reg<=1'b0;  Mem\_Write<=0;alu\_mem\_s<=0;rt\_imm\_s<=0;imm\_s<=1;  rd\_rt\_s<=0;PC\_s<=2'b01;end //跳转    6'b100000 : begin ALU\_OP <= 4'b0100;write\_reg<=1'b1;  Mem\_Write<=0;alu\_mem\_s<=0;rt\_imm\_s<=0;imm\_s<=1;  rd\_rt\_s<=0;PC\_s<=2'b0;end  6'b100010 : begin ALU\_OP <= 4'b0101;write\_reg<=1'b1;  Mem\_Write<=0;alu\_mem\_s<=0;rt\_imm\_s<=0;imm\_s<=1;  rd\_rt\_s<=0;PC\_s<=2'b0;end  6'b100100 : begin ALU\_OP <= 4'b0000;write\_reg<=1'b1;  Mem\_Write<=0;alu\_mem\_s<=0;rt\_imm\_s<=0;imm\_s<=1;  rd\_rt\_s<=0;PC\_s<=2'b0;end  6'b100101 : begin ALU\_OP <= 4'b0001;write\_reg<=1'b1;  Mem\_Write<=0;alu\_mem\_s<=0;rt\_imm\_s<=0;imm\_s<=1;  rd\_rt\_s<=0;PC\_s<=2'b0;end  6'b100110 : begin ALU\_OP <= 4'b0010;write\_reg<=1'b1;  Mem\_Write<=0;alu\_mem\_s<=0;rt\_imm\_s<=0;imm\_s<=1;  rd\_rt\_s<=0;PC\_s<=2'b0;end  6'b100111 : begin ALU\_OP <= 4'b0011;write\_reg<=1'b1;  Mem\_Write<=0;alu\_mem\_s<=0;rt\_imm\_s<=0;imm\_s<=1;  rd\_rt\_s<=0;PC\_s<=2'b0;end  6'b101011 : begin ALU\_OP <= 4'b0110;write\_reg<=1'b1;Mem\_Write<=0;  alu\_mem\_s<=0;rt\_imm\_s<=0;imm\_s<=1;  rd\_rt\_s<=0;PC\_s<=2'b0;end  6'b000100 : begin ALU\_OP <= 4'b0111;write\_reg<=1'b1;Mem\_Write<=0;  alu\_mem\_s<=0;rt\_imm\_s<=0;imm\_s<=1;  rd\_rt\_s<=0;PC\_s<=2'b0;end    endcase  end  else  begin  case(OP)  6'b001000 : begin ALU\_OP <= 4'b0100;write\_reg<=1'b1;  Mem\_Write<=0;alu\_mem\_s<=0;rt\_imm\_s<=1;imm\_s<=1;rd\_rt\_s<=1; PC\_s<=2'b0;  end  6'b001100 : begin ALU\_OP <= 4'b0000;write\_reg<=1'b1;  Mem\_Write<=0;alu\_mem\_s<=0;rt\_imm\_s<=1;imm\_s<=0;rd\_rt\_s<=1;PC\_s<=2'b0;  end  6'b001110 : begin ALU\_OP <= 4'b0010;write\_reg<=1'b1;  Mem\_Write<=0;alu\_mem\_s<=0;rt\_imm\_s<=1;imm\_s<=0;rd\_rt\_s<=1;PC\_s<=2'b0;  end  6'b001011 : begin ALU\_OP <= 4'b0110;write\_reg<=1'b1;  Mem\_Write<=0;alu\_mem\_s<=0;rt\_imm\_s<=1;imm\_s<=0;rd\_rt\_s<=1;PC\_s<=2'b0;  end  6'b100011 : begin ALU\_OP <= 4'b0100;write\_reg<=1'b1;  Mem\_Write<=0;alu\_mem\_s<=1;rt\_imm\_s<=1;imm\_s<=1;rd\_rt\_s<=1;PC\_s<=2'b0;  end  6'b101011 : begin ALU\_OP <= 4'b0100;write\_reg<=1'b0;  Mem\_Write<=1; rt\_imm\_s<=1;imm\_s<=1;PC\_s<=2'b0;  end      6'b000100 : begin ALU\_OP <= 4'b0101;write\_reg<=1'b0;  Mem\_Write<=0;alu\_mem\_s<=1;rt\_imm\_s<=0;imm\_s<=1;  if(Inst\_code[25:21]==Inst\_code[20:16])  begin PC\_s<=2'b10;end  else begin PC\_s<=2'b0; end    end  6'b000101 : begin ALU\_OP <= 4'b0101;write\_reg<=1'b0;  Mem\_Write<=0;alu\_mem\_s<=1;rt\_imm\_s<=0;imm\_s<=1;  if(Inst\_code[25:21]==Inst\_code[20:16])  begin PC\_s<=2'b0;end  else begin PC\_s<=2'b10; end  end  6'b000010 : begin write\_reg<=1'b0;  Mem\_Write<=0;PC\_s<=2'b11;  end  6'b000011 : begin write\_reg<=1'b1;  Mem\_Write<=0;alu\_mem\_s<=2'b1X;rd\_rt\_s<=2'b1X;PC\_s<=2'b11;  end  endcase  end  end  endmodule | | | | | |
| **三、实验仿真** | | | | | |
| 1. 仿真代码   `timescale 1ns / 1ps  module demo;  // Inputs  reg [5:0] SW;  // Outputs  wire [32:1] Output\_Data;  wire [32:1] ALU\_F;  // Instantiate the Unit Under Test (UUT)  CPU\_R\_I\_J uut (  .SW(SW),  .Output\_Data(Output\_Data),  .ALU\_F(ALU\_F)  );  always #10 SW[0]=~SW[0];  initial begin  // Initialize Inputs  SW = 0;  // Wait 100 ns for global reset to finish  #100;    // Add stimulus here  SW[1] = 1;#10; SW[1] =0;#10;  end    endmodule   1. 仿真波形          1. 仿真结果分析   仿真就结果与分析结果完全一致 | | | | | |
| **四、电路图** | | | | | |
|  | | | | | |
| **五、引脚配置（约束文件）** | | | | | |
| NET "Output\_Data[31]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[30]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[29]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[28]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[27]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[26]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[25]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[24]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[23]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[22]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[21]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[20]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[19]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[18]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[17]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[16]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[15]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[14]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[13]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[12]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[11]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[10]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[9]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[8]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[7]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[6]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[5]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[4]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[3]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[2]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[1]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[32]" IOSTANDARD = LVCMOS18;  NET "Output\_Data[32]" LOC = R1;  NET "Output\_Data[31]" LOC = P2;  NET "Output\_Data[30]" LOC = P1;  NET "Output\_Data[29]" LOC = N2;  NET "Output\_Data[28]" LOC = M1;  NET "Output\_Data[27]" LOC = M2;  NET "Output\_Data[26]" LOC = L1;  NET "Output\_Data[25]" LOC = J2;  NET "Output\_Data[24]" LOC = G1;  NET "Output\_Data[23]" LOC = E1;  NET "Output\_Data[22]" LOC = D2;  NET "Output\_Data[21]" LOC = A1;  NET "Output\_Data[20]" LOC = L3;  NET "Output\_Data[19]" LOC = G3;  NET "Output\_Data[18]" LOC = K4;  NET "Output\_Data[17]" LOC = G4;  NET "Output\_Data[16]" LOC = K1;  NET "Output\_Data[15]" LOC = J1;  NET "Output\_Data[14]" LOC = H2;  NET "Output\_Data[13]" LOC = G2;  NET "Output\_Data[12]" LOC = F1;  NET "Output\_Data[11]" LOC = E2;  NET "Output\_Data[10]" LOC = D1;  NET "Output\_Data[9]" LOC = B1;  NET "Output\_Data[8]" LOC = B2;  NET "Output\_Data[7]" LOC = N3;  NET "Output\_Data[6]" LOC = M3;  NET "Output\_Data[5]" LOC = K3;  NET "Output\_Data[4]" LOC = H3;  NET "Output\_Data[3]" LOC = N4;  NET "Output\_Data[2]" LOC = L4;  NET "Output\_Data[1]" LOC = J4;  NET "SW[5]" IOSTANDARD = LVCMOS18;  NET "SW[4]" IOSTANDARD = LVCMOS18;  NET "SW[3]" IOSTANDARD = LVCMOS18;  NET "SW[2]" IOSTANDARD = LVCMOS18;  NET "SW[1]" IOSTANDARD = LVCMOS18;  NET "SW[0]" IOSTANDARD = LVCMOS18;  NET "SW[5]" LOC = R4;  NET "SW[4]" LOC = AA4;  NET "SW[3]" LOC = AB6;  NET "SW[2]" LOC = T5;  NET "SW[1]" LOC = V8;  NET "SW[0]" LOC = AA8;  NET "SW[0]" CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "SW[1]" CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "SW[4]" CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "SW[5]" CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "SW[3]" CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "SW[2]" CLOCK\_DEDICATED\_ROUTE = FALSE; | | | | | |
| **六、思考与探索** | | | | | |
| 1. 实验结果记录：   0000\_0000  0000\_0014  0000\_000a  0000\_0010  0000\_0014  0000\_0000  0000\_0014  0000\_000a  1440\_fffc  0000\_2020  0000\_0009  0000\_0008  Ad2b\_0000  214a\_ffff   1. 实验结论：   实验结果与分析结果完全一致   1. 问题与解决方案：   唯一问题:没有问题  解决方案:我也想写点在这里,但是,这三个实验的坑在前面两个R 和R\_I型实验已经踩完了,所以这个实验非常顺利,把指令编码模块写好就好了,就是端口的更改有点麻烦,但是没有像上一个实验一样,写错端口,还行!   1. 思考题： 2. 实验结果与分析结果完全一致 3. Offset的转移地址公式为 PC+4+offset->PC ,address的转移地址公式为{(PC+4)高四位,adderss,0,0}->PC,结果与分析结果完全一致 4. 这条指令的大概意思就是一个式子:if rs<0 PC+4+offset->rt   因此,为了实现这条指令,指令译码的w\_r\_s=1, t\_imm\_s = 1, wr\_data\_s = 0, 在ALU\_A与PC\_new之间加一条选择数据通路,设置一个控制信号,信号有效,则吧PC\_new送入ALU\_B,这样就可以实现BLTZAL这条指令了 | | | | | |