**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 |  | 指导老师 |  | 机位号 |  |
| 实验序号 | 实验九 | 实验名称 | R\_I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型和I型指令的综合数据通路设计    2. 掌握数据流的多路选通控制方法    3. 掌握取数指令lw和存数指令sw指令的寻址方式及其有效地址产生方法    4. 实现MIPS的部分 I型和R型指令的功能   ： | | | | | |
| **二、实验设计与程序代码** | | | | | |
| 1. 模块设计说明   本实验是R\_I型CPU的实现,相关模块如CPU,寄存器堆,取指令模块在前面实验报告已经给出,  本实验只给出顶层模块,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(SW,Output\_Data);    parameter ADDR = 5;//地址位宽  parameter SIZE = 32;//数据位宽  //输出数据  output reg [SIZE:1] Output\_Data;        //时钟clk  input [5:0] SW;  //指令地址  wire [SIZE:1] PC;    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  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,alu\_mem\_s,rt\_imm\_s,imm\_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 (  .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),  .rt\_imm\_s(rt\_imm\_s),  .imm\_s(imm\_s),  .rd\_rt\_s(rd\_rt\_s)  );  //PC模块  Get\_Inst get\_inst(  .inst\_code(Inst\_code),  .clk(SW[0]),  .reset(SW[1]),  .PC(PC)  );    //数据存储器  RAM\_A RAM\_A (  .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?M\_R\_Data:ALU\_F;  assign M\_W\_Data = R\_Data\_B;  assign W\_Addr = (rd\_rt\_s)?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 Output\_Data=W\_Data;  always@(posedge SW[2] or posedge SW[3])  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  endmodule  //译码器模块  `timescale 1ns / 1ps  module yimaqi(OP,func,write\_reg,ALU\_OP,Mem\_Write,alu\_mem\_s,rt\_imm\_s,imm\_s,rd\_rt\_s);  input [5:0]OP;  input [5:0]func;  output reg write\_reg;  output reg [3:0]ALU\_OP;  output reg Mem\_Write,alu\_mem\_s,rt\_imm\_s,imm\_s,rd\_rt\_s;  always@(\*)  begin  if(OP==6'b000000)  begin  case(func)  6'b100000 : ALU\_OP <= 4'b0100;  6'b100010 : ALU\_OP <= 4'b0101;  6'b100100 : ALU\_OP <= 4'b0000;  6'b100101 : ALU\_OP <= 4'b0001;  6'b100110 : ALU\_OP <= 4'b0010;  6'b100111 : ALU\_OP <= 4'b0011;  6'b101011 : ALU\_OP <= 4'b0110;  6'b000100 : ALU\_OP <= 4'b0111;  endcase  write\_reg<=1'b1;  Mem\_Write<=0;  alu\_mem\_s<=0;  rt\_imm\_s<=0;  imm\_s<=1;  rd\_rt\_s<=0;  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;  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;  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;  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;  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;  end  6'b101011 : begin ALU\_OP <= 4'b0100;write\_reg<=1'b0;  Mem\_Write<=1; rt\_imm\_s<=1;imm\_s<=1;  end  endcase  end  end  endmodule | | | | | |
| **三、实验仿真** | | | | | |
| 1. 仿真代码   `timescale 1ns / 1ps  ////////////////////////////////////////////////////////////////////////////////  // Company:  // Engineer:  //  // Create Date: 11:08:38 05/30/2019  // Design Name: CPU\_R\_I  // Module Name: E:/ISE/Term\_PC/CPU\_R\_I/demo.v  // Project Name: CPU\_R\_I  // Target Device:  // Tool versions:  // Description:  //  // Verilog Test Fixture created by ISE for module: CPU\_R\_I  //  // Dependencies:  //  // Revision:  // Revision 0.01 - File Created  // Additional Comments:  //  ////////////////////////////////////////////////////////////////////////////////  module demo;  // Inputs  reg [5:0] SW;  // Outputs  wire [32:1] Output\_Data;  // Instantiate the Unit Under Test (UUT)  CPU\_R\_I uut (  .SW(SW),  .Output\_Data(Output\_Data)  );    always #1 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\_1234  0000\_6789  FFFF\_9000  0000\_0010  6789\_0000  6789\_1234  9000\_0000  6789\_79BD  0000\_5555  FFFF\_AAAB  0000\_0FFF  FFFF\_1000  0000\_AAAB  0000\_0000  0000\_0001   1. 实验结论：   实验结果与分析完全一致   1. 问题与解决方案：   唯一问题:仿真结果一致为X,仿真失败后,找不出问题在哪  解决方案:首先是查看代码有没有明显的错误,但是代码都是直接copy R型指令实验的,理论上是不存在问题的,然后,排查了代码错误的可能,然后找不到问题所在的,然后,迫于无奈,对这些模块挨个进行仿真,然后,按照书上的R\_I型数据流图,进行人工跑,最终发现,还是端口太多了导致端口的写入混乱,一些模块的端口的输入错了,经过一番改正,正确了!   1. 思考题： 2. 实验结果与分析完全一致 3. Lui指令的实现,首先,将指令编码rt\_imm\_s设置为1,然后在alu模块设计一个独立的运算即ALU\_F<={imm,16’b},然后alu\_mem\_s设置为0即可实现lui指令 | | | | | |