**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 | 计算机组成原理 | |
| 任课老师 | 章复嘉 | 指导老师 |  | 机位号 |  |
| 实验序号 |  | 实验名称 | R型指令CPU | | |
| 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型指令的数据通路设计，掌握指令流和数据流的控制方法；    2. 掌握完整的单周期CPU顶层模块的设计方法；   实现MIPS R型指令的功能 | | | | | |
| **二、实验设计与程序代码** | | | | | |
| 1. 模块设计说明   本实验只提供了顶层模块和译码器模块,其他的模块在前面的实验已经给出   1. 实验程序源代码及注释等   `timescale 1ns / 1ps  module CPU(SW,Output\_Data);    parameter ADDR = 5;//地址位宽  parameter SIZE = 32;//数据位宽  //输出数据  output [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  reg [ADDR:1]W\_Addr;//写寄存器地址  reg [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;  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  //指令存储器  wire [SIZE:1] Inst\_code;    //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    //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(Inst\_code[16:12]),  .W\_Data(ALU\_F),  .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(R\_Data\_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)  );  //PC模块  Get\_Inst get\_inst(  .inst\_code(Inst\_code),  .clk(SW[0]),  .reset(SW[1]),  .PC(PC)  );    // assign {OP,rs,rt,rd,shamt,func} = Inst\_code;    always@(posedge SW[2] or posedge SW[3])  begin  if(SW[2]) begin Output\_Data<=ALU\_F; end  else if(SW[3]) begin Output\_Data<={31'b0,OF,ZF}; end  end  endmodule  //译码器模块  `timescale 1ns / 1ps  module yimaqi(OP,func,write\_reg,ALU\_OP);  input [5:0]OP;  input [5:0]func;  output reg write\_reg;  output reg [3:0]ALU\_OP;  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=(func==0)?1'b0:1'b1;  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] PC\_Data;  // Instantiate the Unit Under Test (UUT)  CPU uut (  .SW(SW),  .Output\_Data(Output\_Data),  .PC\_Data(PC\_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 | | | | | |
| **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：   FFFF\_FFFF **0000\_0001 0000\_0002 0000\_0003 0000\_0005 0000\_0007**  **0000\_000E 0000\_000C 0000\_7000 FFFF\_8FFF FFFF\_8FFE(只记录了前十个结果)**   1. 实验结论：   实验的结果与与预测完全一致   1. 问题与解决方案：   问题一:仿真的时候,需要20个左右的脉冲才能重存储器里面读出来一条指令?  解决方案:目前还没有找到合适的解决方案,而且主要是这种情况是出在仿真的时候,在进行板级验证的时候并没有出现这种情况,按一下按钮就能出来数!  问题二:在刚刚做完的时候,如何怎么修改代码,都只能读出来 0000\_0000?  解决方案: 在把前几个模块合并的时候,使用的是”add sourse”,而不是重新去新建一个,而这,对于IP核是不行的,只能通过新建一个文件才能解决问题!   1. 思考题： 2. 各条指令的执行结果与分析完全一致 3. 由于alu模块存在逻辑左移的功能,因此可以通过在alu模块加一个端口shamt ,然后在ALU模块的逻辑左移模块将逻辑左移运算那个地方的左移位数改成shamt即可实现功能! 4. 对于无符号数的比较可直接使用大小符号进行判断,但是对于有符号数则不能这样做,如果需要对有符号数进行大小比较,需要先对数的首位进行比较,如果首位相同再对剩余的位数使用大小符号进行大小比较即可实现! 5. 算术右移和逻辑右移的区别在于逻辑右移只需要高位补零即可,但是算术右移需要根据最高位为’1’或者’0’进行补零或者一!因此如果进行算术右移需要先对最高位尽行判断,如果为’1’则补’1’,为’0’则补’0’1 | | | | | |