**实验报告**

2019 年 6 月 4 日 成绩：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 於文卓 | 学号 | 17061833 | 班级 | 17052317 |
| 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 | 冯建文 | 指导老师 | 冯建文 | 机位号 |  |
| 实验序号 | 10 | 实验名称 | 实现R-I-J型指令的CPU设计实验 | | |
| 实验时间 | 6月4日 | 实验地点 | 1-225 | 实验设备号 | #9 |
| **一、实验目的与要求** | | | | | |
| 1. 实验目的： 2. 掌握MIPS R型、I型和J型指令的综合数据通路设计 3. 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 4. 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 5. 掌握无条件转移指令和条件转移指令的实现方法 6. 编程实现MIPS的部分J型、I型和R型转移指令的功能 7. 实验要求： 8. 实验九的基础上，编写一个CPU模块，除了能够实现实验九的8条R型指令、6条I型指令外，还要求能够实现新的5条J型指令 9. 将实验九的工程拷贝至新目录下,成为一个新工程；修改ROM\_B和RAM\_B的初始化关联文件为新工程下的\*.coe文件。 10. 定义一些控制和数据信号,添加PC的四选一数据通道和移位部件、地址加法器部件,重新对各模块进行逻辑连接。 11. 修改和扩充CPU模块中指令译码、指令执行控制部分的代码，完善CPU模块 | | | | | |
| **二、实验设计与程序代码** | | | | | |
| 1. 模块设计说明   （1）在CPU的顶层模块中调用之前设计好的取指令模块、寄存器模块、ALU模块、译码模块、数据存储器模块；  （2）在取指令模块中重新生成指令寄存器ROM  （3） 在数据存储器模块中重新生成ROM  （4）在译码模块中，对指令的各个字段和各个信号进行  （5）再设计一个顶层模块，调用消抖模块、数码管模块和cpu的顶层模块，用于板级验证   1. 实验程序源代码及注释等 2. CPU顶层模块   **module** TOP(  **input** clk,  **input** clk\_100M,  **input** rst\_,  **output** [31:0]f,  **output** [31:0]M\_R\_DATA,  **output** OF,  **output** ZF,  **output** set\_of,  **output** set\_zf  *// output [31:0]PC\_new,*  *// output [31:0]Inst\_code,*  *// output [31:0]R\_DATA\_A,*  *// output [31:0]ALU\_B,*  *// output rt\_imm\_s,*  *// output [15:0]imm*  *//*  );    wire write\_reg;  wire mem\_write;  wire [31:0]Inst\_code;  *//各种指令字段*  wire [4:0]rs;  wire [4:0]rt;  wire [4:0]rd;  wire [15:0]imm;  wire [25:0]address;  *//各种信号*  wire imm\_s;  wire rt\_imm\_s;  wire [1:0]w\_r\_s;  wire [1:0]PC\_s;  wire [1:0]wr\_data\_s;  wire [2:0]alu\_op;  wire [31:0]imm\_data;  *//译码器参数*  *//寄存器堆参数*  wire [31:0]R\_DATA\_A;  wire [31:0]R\_DATA\_B;  wire [4:0]W\_Addr;  wire [31:0]W\_Data;  *//ALU参数*  wire [31:0]ALU\_B;  *//取指令模块 得到32位指令*  wire [31:0]PC\_new;  fetch\_inst quzhiling(clk,rst\_,PC\_s,R\_DATA\_A,imm\_data,address,Inst\_code,PC\_new);  decoding yima(Inst\_code,ZF,rs,rt,rd,imm,imm\_s,rt\_imm\_s,address,w\_r\_s,PC\_s,wr\_data\_s,mem\_write,write\_reg,alu\_op,set\_of,set\_zf);  **assign** W\_Addr=(w\_r\_s[1])?5'b11111:((w\_r\_s[0])?**rt:**rd);  *// PC+4-->$31*  register\_file jicunqi(rs,rt,W\_Addr,W\_Data,write\_reg,clk,rst\_,R\_DATA\_A,R\_DATA\_B);  *//imm\_s为1的时候 有符号数扩展 0无符号扩展*  **assign** imm\_data=(imm\_s)?{{16{imm[15]}},imm}:{{16{1'b0}},imm};  *//ALU*  **assign** ALU\_B = (rt\_imm\_s)?**imm\_data:**R\_DATA\_B;  AUL alu(R\_DATA\_A,ALU\_B,alu\_op,ZF,OF,f);  *//数据存储器*  DATA\_RAM data\_ram(clk\_100M,mem\_write,rst\_,f[7:0],R\_DATA\_B,M\_R\_DATA);  **assign** W\_Data=(wr\_data\_s[1])?**PC\_new:**(wr\_data\_s[0]?**M\_R\_DATA:**f);  **endmodule**  取指令  **module** fetch\_inst(  **input** clk,  **input** rst\_,  *//PC四选一*  **input** [1:0]PC\_s,    **input** [31:0]R\_DATA\_A,  **input** [31:0]imm\_data,  **input** [25:0]address,  **output** [31:0]Inst\_code,  **output** reg [31:0]PC  );  wire [31:0]PC\_new;  **initial**  **begin**  PC <= 32'h00000000;  **end**  R\_I\_CPU rom (  *//按下去是负的 但是理论上按下去就取指令 所以按下去要变成正的*  .clka(clk), *// input wire clka*  .addra(PC[7:2]), *// input wire [5 : 0] addra*  .douta(Inst\_code) *// output wire [31 : 0] douta*  );  **assign** PC\_new = PC + 4;    *//上跳沿的时候取指令 下跳沿打入新的PC*  **always**@(**negedge** clk **or** **negedge** rst\_)  **begin**  **if**(!rst\_)PC <= 32'h0000\_0000;  **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[31:28],address,1'b0,1'b0};  **endcase**  **end**  **end**  **endmodule**  寄存器堆  **module** register\_file(  **input** [4:0]R\_Addr\_A,*//读地址A 0-32*  **input** [4:0]R\_Addr\_B,*//读地址B 0-32*  **input** [4:0]W\_Addr,*//写地址 0-32*  **input** [31:0]W\_Data,*//写数据 32位的数据*  **input** Write\_Reg,*//写信号*  **input** clk,  **input** reset\_,  **output** [31:0]R\_Data\_A,*//读到A数据*  **output** [31:0]R\_Data\_B*//读到B数据*  );  **integer** i;  reg [31:0]REG\_Files[0:31];*//32个寄存器堆，每个32位*  **initial**  **begin**  **for**(i=0;i<31;i=i+1)  REG\_Files[i]=0;  **end**    **assign** R\_Data\_A = REG\_Files[R\_Addr\_A];*//读操作A*  **assign** R\_Data\_B = REG\_Files[R\_Addr\_B];*//读操作B*    *//写操作*  **always**@(**posedge** clk **or** **negedge** reset\_)  **begin**  **if**(!reset\_)  **begin**  **for**(i=0;i<31;i=i+1)  REG\_Files[i] <= 32'h0000\_0000;  **end**  **else**  **begin**  **if**(Write\_Reg && W\_Addr!=0)  REG\_Files[W\_Addr] <= W\_Data;  **end**    **end**  **endmodule**  ALU    数据存储器  **module** DATA\_RAM(  **input** clk,  **input** Mem\_Write,  **input** rst\_,  **input** [7:0]Mem\_Addr,  **input** [31:0]M\_W\_DATA,  **output** [31:0]M\_R\_DATA  );  data\_ram DATA\_RAM  (  .clka(clk),  .wea(Mem\_Write),  .addra(Mem\_Addr[7:2]),  .dina(M\_W\_DATA),  .douta(M\_R\_DATA)  );  **endmodule**  **译码模块**  **module** decoding(  *//传入指令*  **input** [31:0]Inst\_code,  *//传入ZF 作为相等与否的判断*  **input** ZF,  *//分配指令的各个字段*  **output** reg[4:0]rs,  **output** reg[4:0]rt,  **output** reg[4:0]rd,  **output** reg[15:0]imm,  *//各类信号*  *//rd\_rt\_s 变成了 w\_r\_s*  *//alu\_mem\_s 变成了 wr\_data\_s*  *//output reg rd\_rt\_s,*  **output** reg imm\_s,  **output** reg rt\_imm\_s,  **output** reg [25:0]address,  **output** reg [1:0]w\_r\_s,  **output** reg [1:0]PC\_s,*//选择PC*  **output** reg [1:0]wr\_data\_s,    **output** reg mem\_write,*//数据存储器写入信号*  **output** reg write\_reg,*//寄存器堆写入信号*  **output** reg[2:0]alu\_op,  **output** reg set\_of,  **output** reg set\_zf    );    **always**@(\*)  **begin**  *//R型*  **if**(Inst\_code[31:26]==0)  **begin**  rs = Inst\_code[25:21];  rt = Inst\_code[20:16];  rd = Inst\_code[15:11];  wr\_data\_s=2'b00;  w\_r\_s = 2'b00;  rt\_imm\_s = 0;  mem\_write = 0;  *//func字段*  **case**(Inst\_code[5:0])  6'b100000:**begin** alu\_op=3'b100;set\_of=1'b1;write\_reg=1;PC\_s=2'b00;set\_zf=1'b1;**end**  6'b100010:**begin** alu\_op=3'b101;set\_of=1'b1;write\_reg=1;PC\_s=2'b00;set\_zf=1'b1;**end**  6'b100100:**begin** alu\_op=3'b000;set\_of=1'b0;write\_reg=1;PC\_s=2'b00;set\_zf=1'b1;**end**  6'b100101:**begin** alu\_op=3'b001;set\_of=1'b0;write\_reg=1;PC\_s=2'b00;set\_zf=1'b1;**end**  6'b100110:**begin** alu\_op=3'b010;set\_of=1'b0;write\_reg=1;PC\_s=2'b00;set\_zf=1'b1;**end**  6'b100111:**begin** alu\_op=3'b011;set\_of=1'b0;write\_reg=1;PC\_s=2'b00;set\_zf=1'b1;**end**  6'b101011:**begin** alu\_op=3'b110;set\_of=1'b0;write\_reg=1;PC\_s=2'b00;set\_zf=1'b1;**end**  6'b000100:**begin** alu\_op=3'b111;set\_of=1'b0;write\_reg=1;PC\_s=2'b00;set\_zf=1'b1;**end**  *//R型无条件跳转 rs-->PC*  6'b001000:**begin** PC\_s=2'b01; **end**  **endcase**  **end**  *//I型立即数寻址指令*  **if**(Inst\_code[31:29]==3'b001)  **begin**  imm = Inst\_code[15:0];  rt = Inst\_code[20:16];  rs = Inst\_code[25:21];  mem\_write = 0;  rt\_imm\_s = 1;  w\_r\_s = 2'b01;  write\_reg = 1;  wr\_data\_s=2'b00;  PC\_s=2'b00;  **case**(Inst\_code[31:26])  6'b001000:**begin** imm\_s = 1;alu\_op=3'b100; **end**  6'b001100:**begin** imm\_s = 0;alu\_op=3'b000; **end**  6'b001110:**begin** imm\_s = 0;alu\_op=3'b010; **end**  6'b001011:**begin** imm\_s = 0;alu\_op=3'b110; **end**  **endcase**  **end**  *//I型存取数指令*  **if**((Inst\_code[31:30]==2'b10)&&(Inst\_code[28:26]==3'b011))  **begin**  imm = Inst\_code[15:0];  rs = Inst\_code[25:21];  rt = Inst\_code[20:16];  rt\_imm\_s = 1;  imm\_s = 1;  PC\_s=2'b00;  **case**(Inst\_code[31:26])  6'b100011:**begin** w\_r\_s=2'b01;wr\_data\_s=2'b01;mem\_write=0;write\_reg=1;alu\_op=3'b100;**end**  6'b101011:**begin** mem\_write=1;write\_reg=0;alu\_op=3'b100;**end**  **endcase**  **end**  *//I型跳转指令*  **if**(Inst\_code[31:27]==5'b00010)  **begin**  imm = Inst\_code[15:0];  rs = Inst\_code[25:21];  rt = Inst\_code[20:16];  imm\_s = 1;  rt\_imm\_s=0;  alu\_op = 3'b101;  write\_reg = 0;  mem\_write = 0;  *//OP字段*  **case**(Inst\_code[31:26])  *//相等转移 相等 ZF=1*  6'b000100:**begin** PC\_s=(ZF?2'b10:2'b00);**end**  *//不相等转移 不相等 ZF=0*  6'b000101:**begin** PC\_s=(ZF?2'b00:2'b10);**end**  **endcase**  **end**  *//J型跳转指令*  **if**(Inst\_code[31:27]==5'b00001)  **begin**  address = Inst\_code[25:0];  PC\_s=2'b11;  **case**(Inst\_code[31:26])  6'b000010:**begin** w\_r\_s=2'b00;write\_reg=0;mem\_write=0; **end**  6'b000011:**begin** w\_r\_s=2'b10;wr\_data\_s=2'b10;write\_reg=1;mem\_write=0;**end**  **endcase**  **end**  **end**    **endmodule**   1. 板级验证顶层模块   **module** CPU\_TOP(  **input** clk,  **input** scan\_clk,  **input** rst\_,  **input** choiceScan,  **output** [7:0]wei,  **output** [7:0]duan,  **output** reg OF,  **output** reg ZF  );    wire tmp\_of;  wire tmp\_zf;  wire set\_of;  wire set\_zf;  wire [31:0]F;  wire [31:0]M\_R\_DATA;  wire [31:0]scanData;  **assign** scanData = choiceScan?**M\_R\_DATA:**F;  **always**@(\*)  **begin**  **if**(set\_of) OF=tmp\_of;  **if**(set\_zf) ZF=tmp\_zf;  **end**  wire xd\_clk;  xd m0(rst\_,scan\_clk,clk,xd\_clk);  *//assign xd\_clk = ~xd\_clk;*  TOP m1(~xd\_clk,scan\_clk,rst\_,F,M\_R\_DATA,tmp\_of,tmp\_zf,set\_of,set\_zf);  scan m2(scan\_clk,rst\_,scanData,wei,duan);  **endmodule**  消抖模块    数码管模块 | | | | | |
| **三、实验仿真** | | | | | |
| 1. 仿真代码   **module** baogaotest1;  *// Inputs*  reg clk;  reg clk\_100M;  reg rst\_;  *// Outputs*  wire [31:0] f;  wire [31:0] M\_R\_DATA;  wire OF;  wire ZF;  wire set\_of;  wire set\_zf;    wire [31:0]PC\_new;  wire [31:0]Inst\_code;  wire [31:0]R\_DATA\_A;  wire [31:0]ALU\_B;  wire rt\_imm\_s;  wire [15:0]imm;  *// Instantiate the Unit Under Test (UUT)*  TOP uut (  .clk(clk),  .clk\_100M(clk\_100M),  .rst\_(rst\_),  .f(f),  .M\_R\_DATA(M\_R\_DATA),  .OF(OF),  .ZF(ZF),  .set\_of(set\_of),  .set\_zf(set\_zf),  .PC\_new(PC\_new),  .Inst\_code(Inst\_code),  .R\_DATA\_A(R\_DATA\_A),  .ALU\_B(ALU\_B),  .rt\_imm\_s(rt\_imm\_s),  .imm(imm)  );  **initial** **begin**  *// Initialize Inputs*  clk = 0;  clk\_100M = 0;  rst\_ = 0;  *// Wait 100 ns for global reset to finish*  #100;  rst\_ = 1;  *// Add stimulus here*  **end**  **always** #20 clk = ~clk;  **always** #5 clk\_100M = ~clk\_100M;   1. 仿真波形   程序1    程序2     1. 仿真结果分析   对于程序1，由于在RAM中的数据全为1，可以看到ALU\_F每隔几条指令后就输出00000001、00000002、00000003…..  对于程序2，可以从Inst\_code字段中看到执行完指令1540fffa后，下一条取到的指令为8d0b0000，  数据存储器中的数据为fffffff1,fffffff2….  仿真结果符合预期，仿真正确。 | | | | | |
| **四、电路图** | | | | | |
|  | | | | | |
| **五、引脚配置（约束文件）** | | | | | |
| //如果边沿输入信号，还要在";"前加上：| CLOCK\_DEDICATED\_ROUTE = FALSE  //譬如: NET "clk" LOC = N17 | IOSTANDARD = LVCMOS18 | CLOCK\_DEDICATED\_ROUTE = FALSE; //BT\_S[7]  //HCS-A01板卡上的实际开关  NET "scan\_clk" LOC = E3 |IOSTANDARD = LVCMOS18 |CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "clk" LOC = P18 |IOSTANDARD = LVCMOS18 |CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "rst\_" LOC = N17 |IOSTANDARD = LVCMOS18 |CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "OF" LOC = U6 |IOSTANDARD = LVCMOS18;  NET "ZF" LOC = R5 |IOSTANDARD = LVCMOS18;  NET "choiceScan" LOC = V5 |IOSTANDARD = LVCMOS18;    NET "wei[7]" LOC = C9 |IOSTANDARD = LVCMOS18;  NET "wei[6]" LOC = C10 |IOSTANDARD = LVCMOS18;  NET "wei[5]" LOC = D10 |IOSTANDARD = LVCMOS18;  NET "wei[4]" LOC = C11 |IOSTANDARD = LVCMOS18;  NET "wei[3]" LOC = M17 |IOSTANDARD = LVCMOS18;  NET "wei[2]" LOC = J14 |IOSTANDARD = LVCMOS18;  NET "wei[1]" LOC = K13 |IOSTANDARD = LVCMOS18;  NET "wei[0]" LOC = P14 |IOSTANDARD = LVCMOS18;  NET "duan[7]" LOC = F14 |IOSTANDARD = LVCMOS18;  NET "duan[6]" LOC = N14 |IOSTANDARD = LVCMOS18;  NET "duan[5]" LOC = J13 |IOSTANDARD = LVCMOS18;  NET "duan[4]" LOC = G13 |IOSTANDARD = LVCMOS18;  NET "duan[3]" LOC = F13 |IOSTANDARD = LVCMOS18;  NET "duan[2]" LOC = G14 |IOSTANDARD = LVCMOS18;  NET "duan[1]" LOC = M13 |IOSTANDARD = LVCMOS18;  NET "duan[0]" LOC = H14 |IOSTANDARD = LVCMOS18; | | | | | |
| **六、思考与探索** | | | | | |
| 1. 程序1实验结果记录：  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 序号 | 指令 | 执行结果 | 标志 | | 下条指令地址 | 结论 | | OF | ZF | | 1 | 00004020 | 00000000 | 0 | 1 | 00000004 | 正确 | | 2 | 00004820 | 00000000 | 0 | 1 | 00000008 | 正确 | | 3 | 200a0014 | 00000014 | 0 | 0 | 0000000c | 正确 | | 4 | 8d2b0010 | 00000010 | 0 | 0 | 00000010 | 正确 | | 5 | 010b4020 | 00000001 | 0 | 0 | 00000014 | 累加0+1=1正确 | | 6 | 21290004 | 00000004 | 0 | 0 | 00000018 | 正确 | | 7 | 214affff | 00000013 | 0 | 0 | 0000001c | 正确 | | 8 | 11400001 | 00000013 | 0 | 0 | 00000020 | 正确 | | 9 | 08000003 | 00000000 | 0 | 0 | 0000000c | 指令循环 正确 | | 10 | 8d2b0010 | 00000014 | 0 | 0 | 00000010 | 正确 | | 11 | 010b4020 | 00000002 | 0 | 0 | 00000014 | 累加1+1=2正确 | | 12 | 21290004 | 00000008 | 0 | 0 | 00000018 | 正确 | | 13 | 214affff | 00000012 | 0 | 0 | 0000001c | 正确 | | 14 | 11400001 | 00000012 | 0 | 0 | 00000020 | 正确 | | 15 | 08000003 | 00000000 | 0 | 0 | 0000000c | 正确 |   ……..   1. 实验结论：   每次按下P18按钮，都能正确显示ALU的F值和R\_DATA\_A的值，并且表示LED灯亮起正确。  实验正确   1. 问题与解决方案： 2. 拷贝指令寄存器的时候，无法读出指令——需要重新生成IP核 3. 用按钮作为时钟信号的时候，按下一次读出多条指令——需要对按钮进行防抖 4. 通过设置set\_of 和 set\_zf 来输出正确的of 和 zf 5. 2'b11:PC<={PC\_new[31:28],address,1'b0,1'b0};正确   2'b11:PC<={PC\_new[31:28],address,0,0};错误  （5）根据指令OP字段进行译码时一定要仔细，容易写错   1. 思考题：     执行 beq $t2,$zero,Loop2时，offest字段为0001H，PC为0000\_0020，PC加上offset\*4为0000\_0024H，是Loop2的PC，所以与转移目标地址一致    要判断rs的正负，通过译码器模块判断R\_DATA\_A的数据的正负，如果是负数，此条指令的实现与jal相同。   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | | | | | | |