实验报告

2021 年 6 月 11 日　　成绩：_____

姓名	******	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	曾虹	指导老师	曾虹	机位号	31
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	2021.6.11	实验地点	一教 225	实验设备号	31

一、实验程序源代码

```
`timescale 1ns / 1ps

module R_I_J_CPU(
    input clk,
    input rst,
    input clk_m,
    output reg [31:0]PC,
    output reg [31:0]PC_next,
    output [31:0]Inst_code,
    output [5:0]opcode,func,
    output [4:0]rs,rt,rd,shamt,
    output [15:0]imm,offset,
    output [31:0] ALU_F,
    output reg [2:0] ALU_OP,
    output ZF,OF,
    output reg Write_Reg,
    output reg Mem_Write,
    output [31:0]R_Data_A,
    output [31:0]R_Data_B,
    output [31:0]M_R_Data,
    output [7:0]Mem_Addr,
    output reg imm_s,
    output reg rt_imm_s,
    output [4:0]W_Addr,
    output [31:0]ALU_B,
    output [31:0]W_Data,
    output reg [31:0]imm_kz,
    output reg [1:0]PC_s,
    output reg[1:0]w_r_s,
```

```
output reg[1:0]wr_data_s,
output [25:0]address
);
 wire [31:0]PC_new;

 initial PC = 32'h00000000;
 assign PC_new = PC + 4;

 ROM_B ROM(
 .clka(clk), // input clka
 .addra(PC[7:2]), // input [5 : 0] addra
 .douta(Inst_code) // output [31 : 0] douta
);

 RAM_B RAM (
 .clka(clk_m), // input clka
 .wea(Mem_Write), // input [0 : 0] wea
 .addra(Mem_Addr[7:2]), // input [5 : 0] addra
 .dina(R_Data_B), // input [31 : 0] dina
 .douta(M_R_Data) // output [31 : 0] douta
);

 assign opcode =  Inst_code[31:26];
 assign rs =  Inst_code[25:21];
 assign rt =  Inst_code[20:16];
 assign rd=  Inst_code[15:11];
 assign shamt = Inst_code[10:6];
 assign func =  Inst_code[5:0];
 assign imm= Inst_code[15:0];
 assign offset= Inst_code[15:0];
 assign address = Inst_code[25:0];

 always @(*)
case (PC_s)
    2'b00: PC_next = PC_new;
    2'b01: PC_next = R_Data_A;
    2'b10: PC_next = PC_new + (imm_kz<<2);
    2'b11: PC_next = {PC_new[31:28],address,2'b00};
endcase

 always @(negedge clk or posedge rst)
 begin
   if (rst)
      PC = 32'h00000000;
```

```
      else
        PC = PC_next;
    end;


    always @(*)
      begin

      ALU_OP = 3'b100;
      imm_s = 1'b0;
      rt_imm_s = 1'b0;
      Write_Reg = 1'b1;
     Mem_Write = 1'b0;
      PC_s = 2'b00;
      w_r_s = 2'b00;
      wr_data_s = 2'b00;


   if (opcode==6'b000000)   //R 指令
    begin
      case (func)
      6'b100000:begin ALU_OP=3'b100;end  //add
      6'b100010:begin ALU_OP=3'b101;end //sub
      6'b100100:begin ALU_OP=3'b000;end //and
      6'b100101:begin ALU_OP=3'b001;end //or
      6'b100110:begin ALU_OP=3'b010;end //xor
      6'b100111:begin ALU_OP=3'b011;end //nor
      6'b101011:begin ALU_OP=3'b110;end //stlu
      6'b000100:begin ALU_OP=3'b111;end //sllv
      6'b001000:begin Write_Reg=0;Mem_Write=0;PC_s = 2'b01; end   //jr
      endcase
   end
    else
      begin
      case(opcode)
      6'b001000:begin
w_r_s=2'b01;imm_s=1;rt_imm_s=1;ALU_OP=3'b100;end  //addi
      6'b001100:begin    w_r_s=2'b01;rt_imm_s=1;ALU_OP=3'b000;    end
//andi
      6'b001110:begin        w_r_s=2'b01;rt_imm_s=1;ALU_OP=3'b010;end
//xori
      6'b001011:begin    w_r_s=2'b01;rt_imm_s=1;ALU_OP=3'b110;    end
//sltiu
      6'b100011:begin
```

```
w_r_s=2'b01;imm_s=1;rt_imm_s=1;wr_data_s=2'b01;ALU_OP=3'b100;      end
//lw
       6'b101011:begin
imm_s=1;rt_imm_s=1;ALU_OP=3'b100;Write_Reg=0;Mem_Write=1; end  //sw
       6'b000100:begin ALU_OP=3'b101;PC_s = (ZF)?2'b10:2'b00; Write_Reg
= 1'b0;end  //beq
       6'b000101:begin ALU_OP=3'b101;PC_s = (ZF)?2'b00:2'b10; Write_Reg
= 1'b0;end  //bne
       6'b000010:begin Write_Reg=0;Mem_Write=0;PC_s = 2'b11; end  //j
       6'b000011:begin
w_r_s=2'b10;wr_data_s=2'b10;Write_Reg=1;Mem_Write=0;PC_s = 2'b11; end
//jal
       endcase
       end
   end;

   always @(*)
       begin
       if(imm_s==1'b0)
           begin
           imm_kz={{16{1'b0}},imm};
           end
       if(imm_s==1'b1)
           begin
           case(imm[15])
           1'b1:imm_kz={{16{1'b1}},imm};
           1'b0:imm_kz={{16{1'b0}},imm};
           endcase
       end
   end;

   assign W_Addr=(w_r_s[1]) ? 5'b11111 : ((w_r_s[0])?rt:rd);
   assign ALU_B=(rt_imm_s)?imm_kz:R_Data_B;
  assign Mem_Addr=ALU_F[7:0];
   assign    W_Data    =    (wr_data_s[1])?PC_new    :((wr_data_s[0])?
M_R_Data:ALU_F);


   REGS
REGS_1(R_Data_A,R_Data_B,W_Data,rs,rt,W_Addr,Write_Reg,rst,~clk);
   ALU ALU_1(ALU_OP,R_Data_A,ALU_B,ALU_F,ZF,OF);

endmodule
```

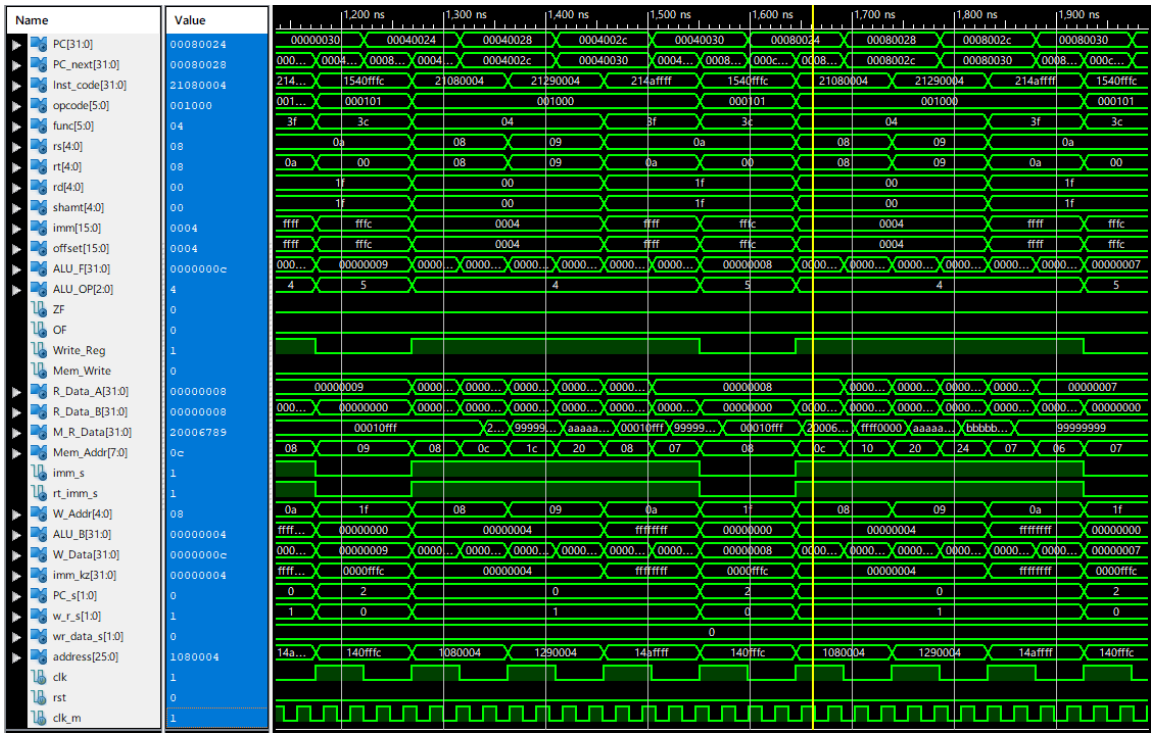
二、仿真测试代码

```
`timescale 1ns / 1ps

module test;

    // Inputs
    reg clk;
    reg rst;
    reg clk_m;

    // Outputs
    wire [31:0] PC;
    wire [31:0] PC_next;
    wire [31:0] Inst_code;
    wire [5:0] opcode;
    wire [5:0] func;
    wire [4:0] rs;
    wire [4:0] rt;
    wire [4:0] rd;
    wire [4:0] shamt;
    wire [15:0] imm;
    wire [15:0] offset;
    wire [31:0] ALU_F;
    wire [2:0] ALU_OP;
    wire ZF;
    wire OF;
    wire Write_Reg;
    wire Mem_Write;
    wire [31:0] R_Data_A;
    wire [31:0] R_Data_B;
    wire [31:0] M_R_Data;
    wire [7:0] Mem_Addr;
    wire imm_s;
    wire rt_imm_s;
    wire [4:0] W_Addr;
    wire [31:0] ALU_B;
    wire [31:0] W_Data;
    wire [31:0] imm_kz;
    wire [1:0] PC_s;
    wire [1:0] w_r_s;
    wire [1:0] wr_data_s;
    wire [25:0]address;
```

```
    // Instantiate the Unit Under Test (UUT)
    R_I_J_CPU uut (
        .clk(clk),
        .rst(rst),
        .clk_m(clk_m),
        .PC(PC),
        .PC_next(PC_next),
        .Inst_code(Inst_code),
        .opcode(opcode),
        .func(func),
        .rs(rs),
        .rt(rt),
        .rd(rd),
        .shamt(shamt),
        .imm(imm),
        .offset(offset),
        .ALU_F(ALU_F),
        .ALU_OP(ALU_OP),
        .ZF(ZF),
        .OF(OF),
        .Write_Reg(Write_Reg),
        .Mem_Write(Mem_Write),
        .R_Data_A(R_Data_A),
        .R_Data_B(R_Data_B),
        .M_R_Data(M_R_Data),
        .Mem_Addr(Mem_Addr),
        .imm_s(imm_s),
        .rt_imm_s(rt_imm_s),
        .W_Addr(W_Addr),
        .ALU_B(ALU_B),
        .W_Data(W_Data),
        .imm_kz(imm_kz),
        .PC_s(PC_s),
        .w_r_s(w_r_s),
        .wr_data_s(wr_data_s),
        .address(address)
    );
   always #13 clk_m=~clk_m;
    always #47 clk=~clk;
    initial begin
        // Initialize Inputs
        clk = 0;
        rst = 1;
        clk_m = 0;
```

```
        #3;
      rst=0;
    end

endmodule
```

三、仿真波形



四、思考与探索

这次实验是把 MIPS 机上的 R、I、J 型指令都实现了，依靠 shamt 字段实现的移位也成功实现，第一次利用 Verilog 语言写成这么大的程序还是很有成就感的。三种指令在逻辑上的指令写出来后，我对 MIPS 指令的译码、执行与取指令过程也有了全新的认识，对 CPU 内的控制信息也有了更全面的了解。实践学习的效果我认为还是很大的。