实验报告

2021 年 6 月 4 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	9	实验名称	实现 R、I 型指令的 CPU 设计实验		
实验时间	2021.6.4	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_CPU(
    input clk,
   input rst,
    input clk_m,
   output [31:0]Inst_code,
   output [5:0]opcode,func,
   output [4:0]rs,
    output [4:0]rt,
    output [4:0]rd,
    output [4:0]shamt,
    output [15:0]imm,
    output [15:0]offset,
   output [31:0] ALU_F,
    output reg [2:0] ALU_OP,
    output ZF,
    output 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 rd_rt_s,
    output reg imm_s,
    output reg [31:0]PC,
    output reg rt_imm_s,
    output reg alu_mem_s,
```

```
     output [4:0]W_Addr,
     output [31:0]ALU_B,
     output [31:0]W_Data,
     output reg [31:0]imm_kz
    );
    wire [31:0]PC_new;

    initial PC = 32'h00000000;

    assign PC_new = PC + 4;
    assign W_Addr = (rd_rt_s) ? rt : rd;
    assign ALU_B = (rt_imm_s) ? imm_kz : R_Data_B;
   assign Mem_Addr = ALU_F[7:0];
   assign W_Data = (alu_mem_s) ? M_R_Data : ALU_F;

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

    RAM_B RAM1 (
      .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
    );

    ROM_B ROM1 (
      .clka(clk), // input clka
      .addra(PC[7:2]), // input [5 : 0] addra
      .douta(Inst_code) // output [31 : 0] douta
    );

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

    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];

    always @(*)
    begin
        ALU_OP = 3'b100;
        rd_rt_s = 1'b0;
        imm_s = 1'b0;
        rt_imm_s = 1'b0;
        alu_mem_s = 1'b0;
        Mem_Write = 1'b0;
        Write_Reg = 1'b1;

        if (opcode == 6'b000000) // R指令

        begin
            ALU_OP = 3'b100;
            rd_rt_s = 1'b0;
            imm_s = 1'b0;
            rt_imm_s = 1'b0;
            alu_mem_s = 1'b0;
            Mem_Write = 1'b0;
            Write_Reg = 1'b1;
            case (func)
                6'b100000:begin ALU_OP=3'b100; end
                6'b100010:begin ALU_OP=3'b101; end
                6'b100100:begin ALU_OP=3'b000; end
                6'b100101:begin ALU_OP=3'b001; end
                6'b100110:begin ALU_OP=3'b010; end
                6'b100111:begin ALU_OP=3'b011; end
                6'b101011:begin ALU_OP=3'b110; end
                6'b000100:begin ALU_OP=3'b111; end
            endcase
        end

        else // I指令

        begin
            case(opcode)
6'b001000:begin rd_rt_s=1; imm_s=1; rt_imm_s=1; ALU_OP=100; end  //addi
6'b001100:begin rd_rt_s=1; rt_imm_s=1; ALU_OP=000; end  //andi
6'b001110:begin rd_rt_s=1; rt_imm_s=1; ALU_OP=010; end  //xori
6'b001011:begin rd_rt_s=1; rt_imm_s=1; ALU_OP=110; end  //sltiu
```

```
6'b100011:begin rd_rt_s=1; imm_s=1; rt_imm_s=1; alu_mem_s=1;
                ALU_OP=100; end  //lw
6'b101011:begin imm_s=1; rt_imm_s=1; ALU_OP=100; Write_Reg=0;
                Mem_Write=1; end  //sw
          endcase
       end
   end

   // 0 拓展与符号拓展

   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;

endmodule
```

二、仿真测试代码

```
`timescale 1ns / 1ps

module TestR_I_CPU;

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

   // Outputs
   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;
```

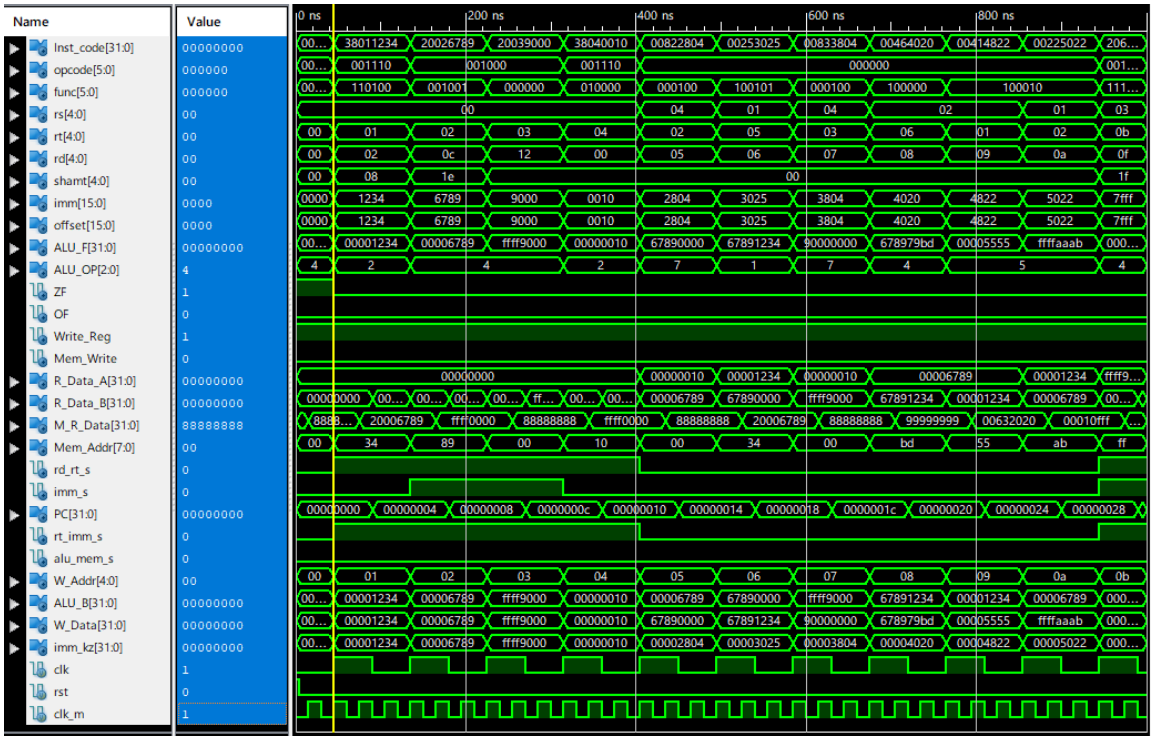
```
    wire [15:0] imm;
    wire [15:0] offset;
    wire [31:0] ALU_F;
    wire [2:0] ALU_OP;
    wire ZF;
    wire OF;
    wire Write_Reg;
    wire Mem_Write;
    wire [31:0] R_Data_A;
    wire [31:0] R_Data_B;
    wire [31:0] M_R_Data;
    wire [7:0] Mem_Addr;
    wire rd_rt_s;
    wire imm_s;
    wire [31:0] PC;
    wire rt_imm_s;
    wire alu_mem_s;
    wire [4:0] W_Addr;
    wire [31:0] ALU_B;
    wire [31:0] W_Data;
    wire [31:0] imm_kz;

    // Instantiate the Unit Under Test (UUT)
    R_I_CPU uut (
        .clk(clk),
        .rst(rst),
        .clk_m(clk_m),
        .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),
        .rd_rt_s(rd_rt_s),
        .imm_s(imm_s),
        .PC(PC),
        .rt_imm_s(rt_imm_s),
        .alu_mem_s(alu_mem_s),
        .W_Addr(W_Addr),
        .ALU_B(ALU_B),
        .W_Data(W_Data),
        .imm_kz(imm_kz)
    );
    initial begin
        // Initialize Inputs
        clk = 0;
        rst = 1;
        clk_m = 0;

        #5;
        rst = 0;
    end
    always #15 clk_m = ~clk_m;
    always #45 clk = ~clk;
endmodule
```

三、仿真波形



四、思考与探索

这次实验是基本上把所有的 R、I 型指令都实现了，建了一个 RAM 和一个 ROM 用来放指令和数据，下次只要再把 J 型跳转指令实现就可以完成最后的 CPU 设计了。基本上通过这次实验我对内部的那些控制信号有了更深度的认识，通过实践也解决了一些上课没能解决的问题。下次做 RIJ 型 CPU 的时候我再更加完善一下它吧。