杭州電子科技大学

计算机组成原理（甲）

实 验 报 告

学　　院	网络空间安全学院
专　　业	网络工程
班　　级	19272401
学　　号	19061440
学生姓名	F001
教师姓名	袁理峰
完成日期	2020.01.08
成　　绩	

实验八 实现 R 型指令的 CPU 设计实验 （实验名称）

一、 实验目的

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
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、 实验原理

实现一个单周期 CPU，实现 8 条 R 型指令；

（1） 建立 R 型指令的数据通路；

（2） 构造顶层模块，含部件：指令存储器（实验七）、PC 及自增电路（实验七）、寄存器堆模块（实验四）、ALU 模块（实验三）、指令译码与控制单元：新增，根据指令码和功能码，为数据通路上各部件发送控制信号（置位或复位）

三、 实验环境

所用电脑的软硬件配置：自己的笔记本电脑、Windows10 操作系统
实验所用的软件：ISE design suite

四、 主要操作步骤及实验结果记录（不能光截图，要有相应的文字说明）

（对实验过程中的主要操作步骤进行描述，并随时记录实验过程中观察到的结果，必要时可辅助截图）
任务一：在 Xilinx ISE 中创建工程，编辑程序源代码，然后编译、综合；若编译出错，则需要重新修改程序代码，直至正确。在之前的实验基础之上，编写一个 CPU 模块，能够实现 8 条指定的 R 型指令。相关实验代码如下：

```
20 //////////////////////////////////////////////////////////////////
21 module Top_LED(clk,rst,SW,LED);
22 input clk,rst;
23 input [2:0]SW;
24 output reg[7:0]LED;
25 wire ZF,OF;
26 wire [31:0]ALU_F;
27 top_R_cpu test_cpu(rst,clk,ZF,OF,ALU_F);
28 always@(*)
29 begin
30 case(SW)
31 3'b000:LED=ALU_F[7:0];
32 3'b001:LED=ALU_F[15:8];
33 3'b010:LED=ALU_F[23:16];
34 3'b011:LED=ALU_F[31:24];
35 3'b100:begin LED[7:2]=0;LED[1]=OF;LED[0]=ZF;end
36 default:LED=0;
37 endcase
38 end
39 endmodule
40
41
```

上图为 TOP_LED 的代码

```
module top_R_cpu(input rst,input clk,output ZF,output OF,output [31:0]F);
reg write_reg;
wire [31:0]Inst_code;
wire [31:0]R_Data_A;
wire [31:0]R_Data_B;
reg [2:0]ALU_OP;
pc pc_connect(clk,rst,Inst_code);
Register_file R_connect(Inst_code[25:21],Inst_code[20:16],
                        Inst_code[15:11],write_reg,F,~clk,rst,
                        R_Data_A,R_Data_B);
ALU ALU_connect(R_Data_A,R_Data_B,F,ALU_OP,ZF,OF);
always@(*)
begin
 write_reg=0;
 ALU_OP=0;
 if(Inst_code[31:26]==0)
 begin
  case(Inst_code[5:0])
  6'b100000:ALU_OP=3'b100;
  6'b100010:ALU_OP=3'b101;
  6'b100100:ALU_OP=3'b000;
  6'b100101:ALU_OP=3'b001;
  6'b100110:ALU_OP=3'b010;
  6'b100111:ALU_OP=3'b011;
  6'b101011:ALU_OP=3'b110;
  6'b000100:ALU_OP=3'b111;
  endcase
  write_reg=1;
  end
  end
  endmodule
```

上图为 top_R_CPU 的代码

```
module pc(input clk,input rst,output [31:0]Inst_code);
reg [31:0]PC;
wire[31:0]PC_new;
initial
    PC<=32'h00000000;
Inst_ROM Inst_ROM1 (
  .clka(clk),
  .addra(PC[7:2]),
  .douta(Inst_code)
);
assign PC_new=PC+4;
always@(negedge clk or posedge rst)
begin
  if(rst)
     PC=32'h00000000;
  else PC={24'h000000,PC_new[7:0]};
end
endmodule
```

上图为 PC_connect 的代码

```
module Register_file(R_Addr_A,R_Addr_B,W_Addr,Write_Reg,W_Data,Clk,Reset,R_Data_A,R_Data_B);
input [4:0]R_Addr_A;
input [4:0]R_Addr_B;
input [4:0]W_Addr;
input Write_Reg;
input [31:0]W_Data;
input Clk;
input Reset;
output [31:0]R_Data_A;
output [31:0]R_Data_B;
reg [31:0]REG_Files[0:31];
reg [5:0]i;
initial//仿真过程中的初始化
begin
 for(i=0;i<=31;i=i+1)
  REG_Files[i]=0;
end
assign R_Data_A=REG_Files[R_Addr_A];
assign R_Data_B=REG_Files[R_Addr_B];
always@(posedge Clk or posedge Reset)
begin
 if(Reset)
  for(i=0;i<=31;i=i+1)
  REG_Files[i]=0;
  else
  if(Write_Reg&&W_Addr!=0)
  REG_Files[W_Addr]=W_Data;
  end
  endmodule
```

上图为 R_connect 的代码

```
module ALU(A,B,F,ALU_OP,ZF,OF);
input [31:0]A,B;
input [2:0]ALU_OP;
output reg ZF,OF;
output reg[31:0]F;
reg C32;
always@(*)
begin
 OF=1'b0;
 C32=1'b0;
 case(ALU_OP)
 3'b000:F=A&B;
 3'b001:F=A|B;
 3'b010:F=A^B;
 3'b011:F=~(A^B);
 3'b100:begin {C32,F}=A+B;OF=A[31]^B[31]^F[31]^C32;end
 3'b101:begin {C32,F}=A-B;OF=A[31]^B[31]^F[31]^C32;end
 3'b110:
 if(A<B)
   F=1;
 else
   F=0;
 3'b111:F=B<<A;
 endcase
 if(F==0)
  ZF=1;
 else
  ZF=0;
 end
 endmodule
```

上图为 ALU 的相关代码

任务二：编写激励代码，观察仿真波形，若验证逻辑有误，则修改代码，重新编译、仿真，直至正确。编写一段测试 8 条指令的汇编程序，使用实验六的汇编器，将其翻译成二进制机器码，并通过关联文件初始化指令寄存器。

```
27    // Inputs
28    reg rst;
29    reg clk;
30
31    // Outputs
32    wire ZF;
33    wire OF;
34    wire [31:0] F;
35
36    // Instantiate the Unit Under Test (UUT)
37    top_R_cpu uut (
38       .rst(rst),
39       .clk(clk),
40       .ZF(ZF),
41       .OF(OF),
42       .F(F)
43    );
44
45    initial begin
46       // Initialize Inputs
47       rst = 0;
48       clk = 0;
49
50       // Wait 100 ns for global reset to finish
51       #100;
52       clk=1;
53       // Add stimulus here
54
55       forever
56       begin
57           #50;
58           clk=~clk;
59       end
60 end
```

上图即为测试代码，经过调试后发现成功通过了语法检查，本次实验成功。

五、 实验分析总结及心得

经过验证，16 条指令的执行结果均正确。思考的相关问题如下：

实现 sllrd,rt,shamt 指令将 rt 寄存器的数据进行逻辑左移，左移的位数由字段 shamt 指定，可以加入 rs_shamt 信号，控制位移量是 rs，地址数据还是 shamt。

assign ALU_A = (rs_shamt)?shamt_kz:R_Data_A;

ALU ALU_1(ALU_OP,ALU_A,R_Data_B,ALU_F,ZF,OF);

在 case(func)中加入：

6'b000000:begin ALU_OP=3'b111;Set_ZF=1;Set_OF=0;rs_shamt=1'b1; end;

本实验实现的 sltu 指令是对无符号数的比较置位指令，如果需要实现有符号数的比较置位指令——slt 指令，可以通过以下的方式进行实现。若想实现有符号数的比较置位指令，可以先根据有符号数的最高位进行分类：

若两正，则和无符号数比较置位无异

若一正一负，则可直接得出结论。

若两负，则将余下位数进行比较置位，再将结果取反即可。

要实现 srav 对（有符号）数据的算术右移指令，可以先读取数据符号位，设右移位数为 n，从数据最右边开始，将其每一位覆盖为其左边 n 位的数的数值，当其左边 n 位数不存在时，改为复制符号位即可。