浙江大学

本科实验报告

课程名称：计算机组成

姓　　名：胡亮泽

学　　院：计算机科学与技术学院

　　系：计算机科学与技术系

专　　业：计算机科学与技术

学　　号：3120102116

指导教师：姜晓红

2014 年　4　月　16　日

浙江大学实验报告

课程名称：Computer Organization 实验类型：综合

实验项目名称：Lab6： R 指令实现

学生姓名：胡亮泽 专业：计算机科学与技术

学号：3120102116

同组学生姓名：王艺 指导老师：姜晓红

实验地点：东 4-509 实验日期：2014 年 4 月 16 日

一、 实验目的和要求

Combine the Register File, ALU Controller, ALU and other units to implement R-type instruction processor , and simulate it.

二、 实验内容和原理

方法：实验通过输入一条 32 二位的 R 指令，并模拟各个寄存器在 R 指令输入后在仿真中的输出结果，来模拟 R 指令在计算机中实现的环境。

本实验中 top 模块文件如下：

```
module top(input wire clk,
          input wire rst,
          input wire[31:0] I,
          output wire[31:0]A, B,
          output wire[31:0]result
    );

wire[2:0] ALUoper;
wire zero;

assign zero=0;

RegFile m0(clk,rst,I[25:21],I[20:16],I[15:11],result,A,B,1'b1);
alu m1(A,B,ALUoper,result,zero);
```

```
aluc m2(2'b10,I[5:0],ALUoper);

endmodule
```

在仿真中，将在 clk 上升沿时读取目标寄存器的值，并在 clk 下降沿时将结果写入目标寄存器。

I 为 32 位的指令序列。而 A,B 为从目标寄存器中读取到的数据。Result 为指令运算结果。

RegFile 为寄存器控制器。将 I 指令中对应的位，I[25：21],I[20：26],I[25：11]分别代表的读入寄存器和写入寄存器输入，并将结果 A,B 输出来得到结果。其中最后一个 1’b1 代表写入操作永远是允许的。

Alu 和 aluc 模块在以前的实验中已经详细描述过，这里不多做赘述。

RegFile 模块代码如下：

```
module RegFile(input wire clk,
             input wire rst,
             input wire[4:0] regA,regB,regW,
             input wire[31:0] Wdat,
             output reg[31:0] Adat,Bdat,
             input wire RegWrite
    );

reg[31:0] reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg10,
    reg11,reg12,reg13,reg14,reg15,reg16,reg17,reg18,reg19,
    reg20,reg21,reg22,reg23,reg24,reg25,reg26,reg27,reg28,
    reg29,reg30,reg31,reg32;



initial begin
    reg1=1;
    reg2=2;
    reg3=3;reg4=4;reg5=5;reg6=6;reg7=7;reg8=8;reg9=9;reg10=10;

reg11=11;reg12=12;reg13=13;reg14=14;reg15=15;reg16=16;reg17=17;re
g18=18;reg19=19;

reg20=20;reg21=21;reg22=22;reg23=23;reg24=24;reg25=25;reg26=26;re
g27=27;reg28=28;
    reg29=29;reg30=30;reg31=31;reg32=32;
end


always @(negedge clk) begin

if(RegWrite) begin
case(regW)
5'b00000: reg1<=Wdat;
5'b00001: reg2<=Wdat;
5'b00010: reg3<=Wdat;
5'b00011: reg4<=Wdat;
5'b00100: reg5<=Wdat;
5'b00101: reg6<=Wdat;
5'b00110: reg7<=Wdat;
```

```
5'b00111: reg8<=Wdat;
5'b01000: reg9<=Wdat;
5'b01001: reg10<=Wdat;
5'b01010: reg11<=Wdat;
5'b01011: reg12<=Wdat;
5'b01100: reg13<=Wdat;
5'b01101: reg14<=Wdat;
5'b01110: reg15<=Wdat;
5'b01111: reg16<=Wdat;
5'b10000: reg17<=Wdat;
5'b10001: reg18<=Wdat;
5'b10010: reg19<=Wdat;
5'b10011: reg20<=Wdat;
5'b10100: reg21<=Wdat;
5'b10101: reg22<=Wdat;
5'b10110: reg23<=Wdat;
5'b10111: reg24<=Wdat;
5'b11000: reg25<=Wdat;
5'b11001: reg26<=Wdat;
5'b11010: reg27<=Wdat;
5'b11011: reg28<=Wdat;
5'b11100: reg29<=Wdat;
5'b11101: reg30<=Wdat;
5'b11110: reg31<=Wdat;
5'b11111: reg32<=Wdat;
endcase

end
end

always @(posedge clk) begin

case(regA)
5'b00000: Adat<=reg1;
5'b00001: Adat<=reg2;
5'b00010: Adat<=reg3;
5'b00011: Adat<=reg4;
5'b00100: Adat<=reg5;
5'b00101: Adat<=reg6;
5'b00110: Adat<=reg7;
5'b00111: Adat<=reg8;
5'b01000: Adat<=reg9;
5'b01001: Adat<=reg10;
5'b01010: Adat<=reg11;
5'b01011: Adat<=reg12;
5'b01100: Adat<=reg13;
5'b01101: Adat<=reg14;
5'b01110: Adat<=reg15;
5'b01111: Adat<=reg16;
5'b10000: Adat<=reg17;
5'b10001: Adat<=reg18;
5'b10010: Adat<=reg19;
5'b10011: Adat<=reg20;
5'b10100: Adat<=reg21;
5'b10101: Adat<=reg22;
5'b10110: Adat<=reg23;
5'b10111: Adat<=reg24;
5'b11000: Adat<=reg25;
5'b11001: Adat<=reg26;
5'b11010: Adat<=reg27;
5'b11011: Adat<=reg28;
```

```
5'b11100: Adat<=reg29;
5'b11101: Adat<=reg30;
5'b11110: Adat<=reg31;
5'b11111: Adat<=reg32;
endcase

case(regB)
5'b00000: Bdat<=reg1;
5'b00001: Bdat<=reg2;
5'b00010: Bdat<=reg3;
5'b00011: Bdat<=reg4;
5'b00100: Bdat<=reg5;
5'b00101: Bdat<=reg6;
5'b00110: Bdat<=reg7;
5'b00111: Bdat<=reg8;
5'b01000: Bdat<=reg9;
5'b01001: Bdat<=reg10;
5'b01010: Bdat<=reg11;
5'b01011: Bdat<=reg12;
5'b01100: Bdat<=reg13;
5'b01101: Bdat<=reg14;
5'b01110: Bdat<=reg15;
5'b01111: Bdat<=reg16;
5'b10000: Bdat<=reg17;
5'b10001: Bdat<=reg18;
5'b10010: Bdat<=reg19;
5'b10011: Bdat<=reg20;
5'b10100: Bdat<=reg21;
5'b10101: Bdat<=reg22;
5'b10110: Bdat<=reg23;
5'b10111: Bdat<=reg24;
5'b11000: Bdat<=reg25;
5'b11001: Bdat<=reg26;
5'b11010: Bdat<=reg27;
5'b11011: Bdat<=reg28;
5'b11100: Bdat<=reg29;
5'b11101: Bdat<=reg30;
5'b11110: Bdat<=reg31;
5'b11111: Bdat<=reg32;
endcase

end

endmodule
```

定义了 32 个寄存器变量并赋予 1~32 的初值。通过 always 分别在 clk 上升沿和下降沿时，通过 regA,regB,regW 译码后，将对应的值从寄存器中读出或者写入寄存器。

三、 实验过程和数据记录

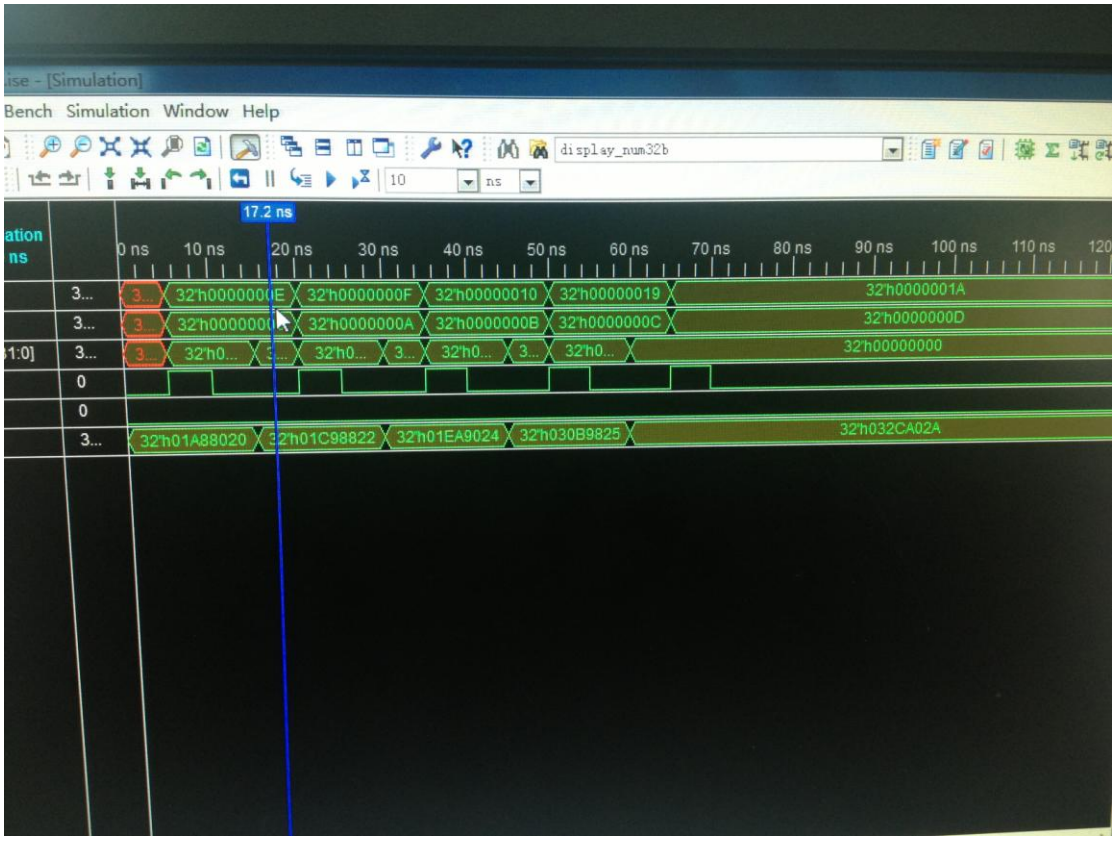
实验中仿真的代码如下所示：

```
module tester;

    // Inputs
    reg clk;
    reg rst;
```

```
    reg [31:0] I;

    // Outputs
    wire [31:0] A;
    wire [31:0] B;
    wire [31:0] result;

    // Instantiate the Unit Under Test (UUT)
    top uut (
        .clk(clk),
        .rst(rst),
        .I(I),
        .A(A),
        .B(B),
        .result(result)
    );


    initial begin
        // Initialize Inputs
        clk = 0;
        rst = 0;
        I = 32'h01AB_8020;
        clk = 1;
        #5;
        clk = 0;
        #5;
        I = 32'h01C9_8822;
        #5;
        clk=1;
        #5;
        clk=0;
        #5;
        I = 32'h01EA_9024;
        #5;
        clk=1;
        #5;
        clk=0;
        #5;
        I = 32'h030B_9825;
        #5;
        clk=1;
        #5;
        clk=0;
        #5;
        I = 32'h032C_A02A;



        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

    end

endmodule
```

如 initial 中模块所示，每隔 15 秒 clk 将经历一个周期，而 I 在每个周期中将会改变一次值。

以下为仿真结果：



前三行分别为 A,B 寄存器和 result 计算结果，第四行为 clk,最后一行为指令。

在每次指令数据改变的时候，result 结果也会改变，而 A,B 寄存器输出值将会在时钟上升沿到来的时候做出对应的改变。

从各条指令中对应的寄存器值来看，结果完全符合要求。比如第一条加法指令，并对应 13 号寄存器和 8 号寄存器，其中存储了 14 和 9，对应结果为 E 和 9，并计算得到结果 16 进制的 19.

其他四条指令以此类推，不多做赘述。

四、 实验结果

在实验过程中最主要的问题是因为变量数量太多，导致 ISE 无法输出所有变量的仿真结果。对于这个问题，应该可以采用版本较新的 ISE 进行仿真来解决。实验本身难度不大，只是重复一些类似的代码时比较麻烦，容易犯一些低级错误。只要前面几次实验没有出现大问题，好好完成任务，这次的实验就很容易得到预期结果。

五、 讨论与心得

本次实验用比较抽象的方法模拟了 R 指令在计算机中实现的过程。对此，我

对 R 指令的理解更加深刻了。