杭州电子科技大学计算机学院

实验报告

实验项目：实现 R 型指令的 CPU 设计实验

课程名称：计算机组成原理

姓名： 盛竹青 学号： 12051742 同组姓名： 学号：

实验位置（机号）：

实验日期： 指导教师： 章复嘉

实验内容（算法、程序、步骤和方法）	1、实验目的
	（1）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
	（2）掌握完整的单周期 CPU 顶层模块的设计方法
	（3）实现 MIPS R 型的指令
	2、实验仪器
	ISE
	
	3、步骤、方法
	module
	work8(rst,Clk,F,OF,ZF,PC,PC_new,M_R_Data,R_Addr_A,R_Addr_B,W_Addr,ALU_OP,R_Data_A,R_Data_B);
	input wire Clk,rst;
	output wire [31:0] R_Data_A,R_Data_B;
	output wire [31:0] F;
	output wire OF,ZF;
	output reg [2:0] ALU_OP;
	output reg [31:0] PC;
	output reg [31:0]PC_new;
	reg [31:0]A,B;
	output wire[31:0] M_R_Data;
	output wire[4:0]R_Addr_A,R_Addr_B,W_Addr;
	assign R_Addr_A=M_R_Data[25:21];
	assign R_Addr_B=M_R_Data[20:16];
	assign W_Addr=M_R_Data[15:11];
	work4 RAM (
	.addr_A(M_R_Data[25:21]), //数据 1 地址
	.addr_B(M_R_Data[20:16]), //数据 2 地址
	.Data_A(R_Data_A),
	.Data_B(R_Data_B),
	.Clk(Clk),
	.Reset(rst),
	.W_Data(F),
	.W_addr(M_R_Data[15:11]), //写地址
	.WorR(!Clk)

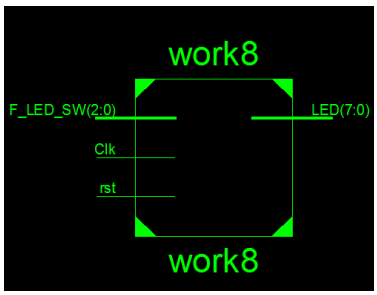
```
    );

Inst_ROM1 myROM (
  .clka(Clk), // input clka
  .addra(PC[7:2]), // input [5 : 0] addra
  .douta(M_R_Data) // output [31 : 0] douta
);

work3 ALU (
    .A(R_Data_A),
    .B(R_Data_B),
    .ALU_OP(ALU_OP),
    .F(F),
        .OF(OF),
        .ZF(ZF)
    );
        initial
        begin
        PC=32'h0000_0000;
        PC_new=32'h0000_0000;
        end

        always@(posedge Clk or posedge rst)  //取指令操作
        begin
        if(rst)
                PC_new[31:0]=32'h0000_0000;
        else
                PC_new=PC+4;
        end
        always@(negedge Clk or posedge rst)  //取指令操作
        begin
        if(rst)
                PC[31:0]=32'h0000_0000;
        else
                PC=PC_new;
        end

        always@(*)
        begin
        if(Clk)
        begin
                        case(M_R_Data[5:0])  //指令译码
                                6'b100000:ALU_OP=3'b100;
```

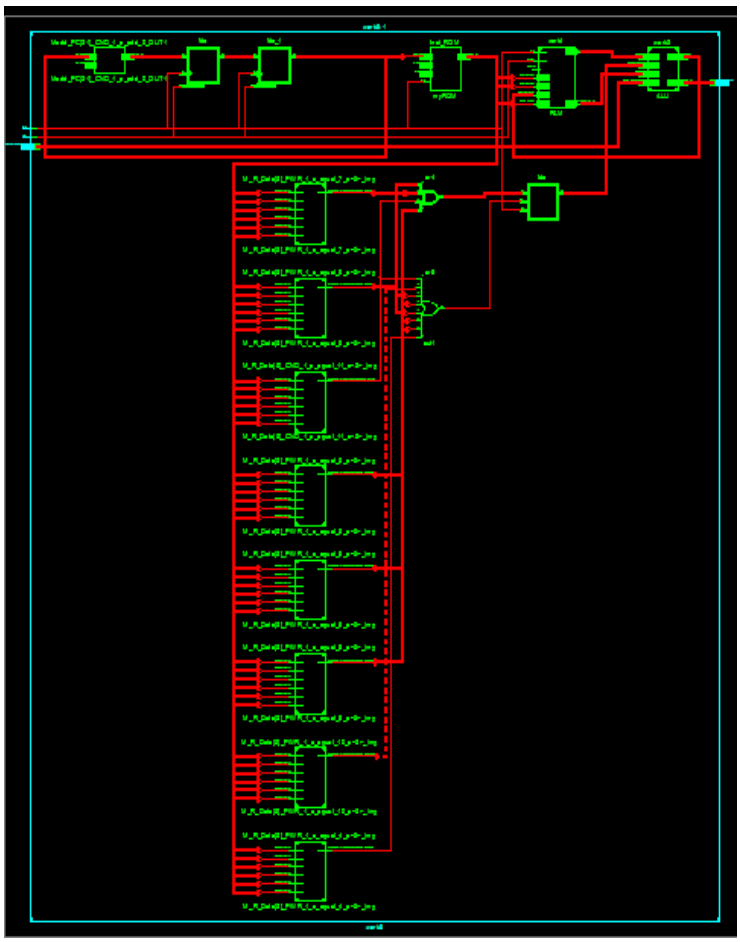
	6'b100010:ALU_OP=3'b101; 6'b100100:ALU_OP=3'b000; 6'b100101:ALU_OP=3'b001; 6'b100110:ALU_OP=3'b010; 6'b100111:ALU_OP=3'b011; 6'b101011:ALU_OP=3'b110; 6'b000100:ALU_OP=3'b111; endcase end end endmodule

操作过程及结果	1、操作过程
	实验过程和描述：
	
	module tt;
	
	// Inputs
	reg rst;
	reg Clk;
	
	// Outputs
	wire [31:0] F;
	wire OF;
	wire ZF;
	wire [31:0] PC;
	wire [31:0] PC_new;
	wire [31:0] M_R_Data;
	wire [4:0] R_Addr_A;
	wire [4:0] R_Addr_B;
	wire [4:0] W_Addr;
	wire [2:0] ALU_OP;
	wire [31:0] R_Data_A;

```
wire [31:0] R_Data_B;

// Instantiate the Unit Under Test (UUT)
work8 uut (
	.rst(rst),
	.Clk(Clk),
	.F(F),
	.OF(OF),
	.ZF(ZF),
	.PC(PC),
	.PC_new(PC_new),
	.M_R_Data(M_R_Data),
	.R_Addr_A(R_Addr_A),
	.R_Addr_B(R_Addr_B),
	.W_Addr(W_Addr),
	.ALU_OP(ALU_OP),
	.R_Data_A(R_Data_A),
	.R_Data_B(R_Data_B)
);

initial begin
	// Initialize Inputs
	rst = 0;
	Clk = 0;

	// Wait 100 ns for global reset to finish
	#20;
	rst=1;
	#30;
	Clk=1;rst=0;
		#50;
	Clk=0;
		#50;
	Clk=1;
		#50;
	Clk=0;
		#50;
	Clk=1;
		#50;
	Clk=0;
		#50;
	Clk=1;
		#50;
```

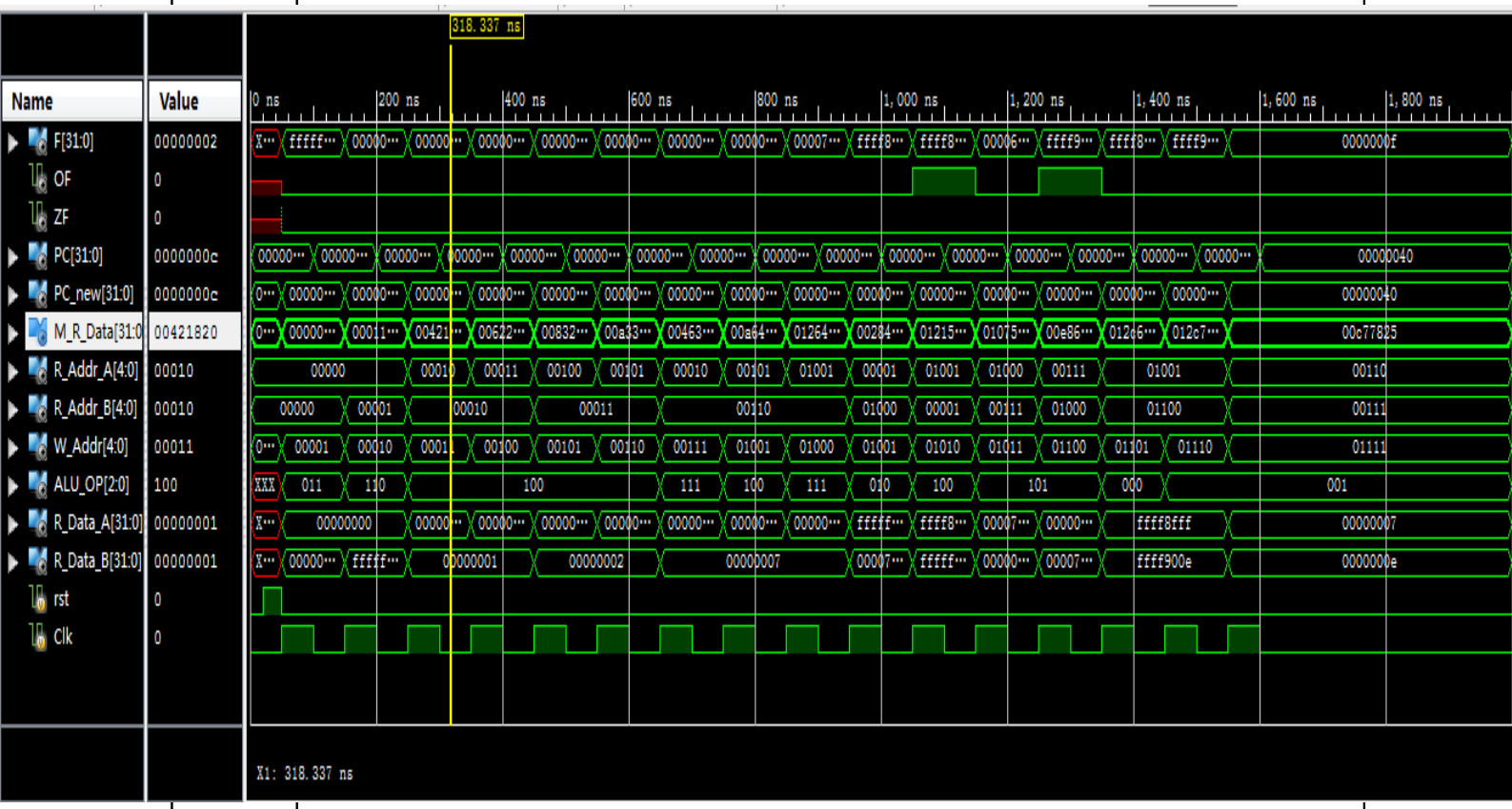
```
Clk=0;
        #50;
Clk=1;
        #50;
Clk=0;
        #50;
Clk=1;
        #50;
Clk=0;
        #50;
Clk=1;
        #50;
Clk=0;
        #50;
Clk=1;
        #50;
Clk=0;
        #50;
Clk=1;
    #50;
Clk=0;
        #50;
Clk=1;
        #50;
Clk=0;
        #50;
Clk=1;
        #50;
Clk=0;
        #50;
Clk=1;
        #50;
Clk=0;
        #50;
Clk=1;
        #50;
Clk=0;
        #50;
Clk=1;
        #50;
Clk=0;
        #50;
Clk=1;
```





```
                #50;
        Clk=0;
                #50;
        Clk=1;
                #50;
        Clk=0;
        // Add stimulus here

    end

endmodule
```

二、结果



实验体会

本实验要求实现 R 型指令的 CPU 设计，也就是要把之前做好的 ALU 运算器以及存储器一起调用进来组合在一起，最开始也是准备把指令存储器模块调用进来的，指令存取模块是不需要调用进来的，应该直接在顶层模块里重新编写。

实验中遇到的问题有很多，最开始是顺序的问题，在一个时钟信号里面要处理这么多的过程，不知道要怎么开始，以一个什么样的顺序进行下去，指导书上提示说把一个时钟信号分为三个阶段，上跳沿，CLk 持续，以及下跳沿，分工完成任务。与以前的模块调用不一样，这次的两个模块调用需要改写模块里的代码，因为很多输入和输出都需要关掉，有些输入需要重新添加，在顶层模块里也需要添加一些零时变量以利于个个模块中的信号链接，由于在之前的存储器实验中并没有对其初始化值，导致最开始仿真的时候，一直没有取到值，仿真一直有问题，最后得知，只需对 RAM 存储器的第一个单元赋值即可。

本实验中遇到最大的问题是后续连续的 CLK 信号来临时，没有得到正确的运算结果，再经过一步步的检测后发现，是因为 ALU 运算器是实时运算的，所以检测列表应该是检测所有输入端口的信号变化，而不是只检测运算操作信号，这个问题解决后，仿真正常通过。

以上就是本实验中遇到的问题。

课后思考题：

1、经过实验分析和实际仿真测试可以得出下表记录结果：

序号	指令	执行结果	标志		结论
			ZF	OF	
1	00000827	ffffffff	0	0	正确
2	0001102b	00000001	0	0	正确
3	00421820	00000002	0	0	正确
4	00622020	00000003	0	0	正确
5	00832820	00000005	0	0	正确
6	00a33020	00000007	0	0	正确
7	00463804	0000000E	0	0	正确
8	00a64820	0000000C	0	0	正确
9	01264004	00007000	0	0	正确
10	00284826	FFFF8FFF	0	0	正确
11	01215020	FFFF8FFE	0	0	正确
12	01075822	00006FF2	0	0	正确
13	00e86022	FFFF900E	0	0	正确
14	012c6824	FFFF800E	0	0	正确
15	012c7025	FFFF9FFF	0	0	正确
16	00c77825	0000000F	0	0	正确

2、sll rd ,rt ,shamt 这条指令是要实现 rt 内容逻辑左移，该操作与 R 型指令里第 8 条指令操作是一样的，只是要左移的位数字段不同，所以，可以 sll 和 sllv 这两个操作码字段的不同，来分别决定左移字段位数是哪一个。

3、对于有符号的比较置位指令 slt，我认为在实现上是在 ALU 运算器里先对进来的操作数进行符号位判断正负，若一正一负那就直接得出结果了，若不是，则可以用比较运算符来进行运算。

	4、srav 这个指令，若是对有符号的算数右移，则要先判断操作数的正负，若是正数，算数右移的时候高位补 0，若是负数，高位补 1 即可。
指导教师评议	成绩：　　指导教师签名：