

同济大学计算机系

计算机组成原理实验报告



学 号 2151140

姓 名 王谦

专 业 信息安全

授课老师 张冬冬

一、实验内容

31 条指令单周期 CPU 设计

实验介绍：

在本次实验中，我们将使用 Verilog HDL 语言实现 31 条 MIPS 指令的 CPU 的设计和仿真。

实验目标：

- (1) 深入了解 CPU 的原理。
- (2) 画出实现 31 条指令的 CPU 的通路图。
- (3) 学习使用 Verilog HDL 语言设计实现 31 条指令的 CPU。

实验原理：

(*注意：为了便于使用 Mars 进行测试，CPU 设计做如下规定：Mars 中 CPU 采用冯诺依曼结构，我们使用它的 default 内存设置，故指令存储起始地址为：0x00400000，数据存储起始地址为 0x10010000，由于我们设计的 CPU 为哈佛结构，指令和数据的起始地址都为 0，所以最后请在 PC 模块、指令存储模块、数据存储模块进行地址映射（将逻辑地址转换为物理地址 0））

1) 需要实现的 31 条 MIPS 指令，见图

Mnemonic Symbol	Format						Sample
Bit #	31..26	25..21	20..16	15..11	10..6	5..0	
R-type	op	rs	rt	rd	shamt	func	
add	000000	rs	rt	rd	0	100000	add \$1,\$2,\$3
addu	000000	rs	rt	rd	0	100001	addu \$1,\$2,\$3
sub	000000	rs	rt	rd	0	100010	sub \$1,\$2,\$3
subu	000000	rs	rt	rd	0	100011	subu \$1,\$2,\$3
and	000000	rs	rt	rd	0	100100	and \$1,\$2,\$3
or	000000	rs	rt	rd	0	100101	or \$1,\$2,\$3
xor	000000	rs	rt	rd	0	100110	xor \$1,\$2,\$3
nor	000000	rs	rt	rd	0	100111	nor \$1,\$2,\$3
slt	000000	rs	rt	rd	0	101010	slt \$1,\$2,\$3
sltu	000000	rs	rt	rd	0	101011	sltu \$1,\$2,\$3
sll	000000	0	rt	rd	shamt	000000	sll \$1,\$2,10
srl	000000	0	rt	rd	shamt	000010	srl \$1,\$2,10
sra	000000	0	rt	rd	shamt	000011	sra \$1,\$2,10
slv	000000	rs	rt	rd	0	000100	slv \$1,\$2,\$3
sriv	000000	rs	rt	rd	0	000110	sriv \$1,\$2,\$3
srav	000000	rs	rt	rd	0	000111	srav \$1,\$2,\$3
jr	000000	rs	0	0	0	001000	jr \$31

Bit #	31..26	25..21	20..16	15..0	
I-type	op	rs	rt	immediate	
addi	001000	rs	rt	Immediate(- ~ +)	addi \$1,\$2,100
addiu	001001	rs	rt	Immediate(- ~ +)	addiu \$1,\$2,100
andi	001100	rs	rt	Immediate(0 ~ +)	andi \$1,\$2,10
ori	001101	rs	rt	Immediate(0 ~ +)	andi \$1,\$2,10
xori	001110	rs	rt	Immediate(0 ~ +)	andi \$1,\$2,10
lw	100011	rs	rt	Immediate(- ~ +)	lw \$1,10(\$2)
sw	101011	rs	rt	Immediate(- ~ +)	sw \$1,10(\$2)
beq	000100	rs	rt	Immediate(- ~ +)	beq \$1,\$2,10
bne	000101	rs	rt	Immediate(- ~ +)	bne \$1,\$2,10
slti	001010	rs	rt	Immediate(- ~ +)	slti \$1,\$2,10
sltiu	001011	rs	rt	Immediate(- ~ +)	sltiu \$1,\$2,10
lui	001111	00000	rt	Immediate(- ~ +)	Lui \$1, 10
Bit #	31..26	25..0			
J-type	op	Index			
j	000010	address			j 10000
jal	000011	address			jal 10000

2) 单周期数据通路设计的一般性方法

- (1) 阅读每条指令，对每条指令所需执行的功能与过程都有充分的了解
- (2) 确定每条指令在执行过程中所用到的部件
- (3) 使用表格列出指令所用部件，并在表格中填入每个部件的数据输入来源
- (4) 根据表格所涉及部件和部件的数据输入来源，画出整个数据通路

3) 前仿真

前仿真也称为功能仿真，主旨在于验证电路的功能是否符合设计要求，其特点是不考虑电路门延迟与线延迟，主要是验证电路与理想情况是否一致。可综合 FPGA 代码是用 RTL 级代码语言描述的，其输入为 RTL 级代码与 Testbench.

4) 后仿真

后仿真也称为时序仿真或者布局布线后仿真，是指电路已经映射到特定的工艺环境以后，综合考虑电路的路径延迟与门延迟的影响，验证电路能否在一定时序条件下满足设计构想的过程，是否存在时序违规。其输入文件为从布局布线结果中抽象出来的门级网表、Testbench 和扩展名为 SDO 或 SDF 的标准时延文件。SDO 或 SDF 的标准时延文件不仅包含门延迟，还包括实际布线延迟，能较好地反映芯片的实际工作情况。一般来说后仿真是必选的，检查设计时序与实际的 FPGA 运行情况是否一致，确保设计的可靠性和稳定性。选定了器件分配引脚后在做后仿真。

5) 下板测试 由于我们自行编写的指令 RAM 用来初始化内存的 initial 指令是不可综合的，无法在开发板上运行，所以，我们可以使用 Vivado 提供的 ip 核来替换我们的 ram，其可以使用一个 coe

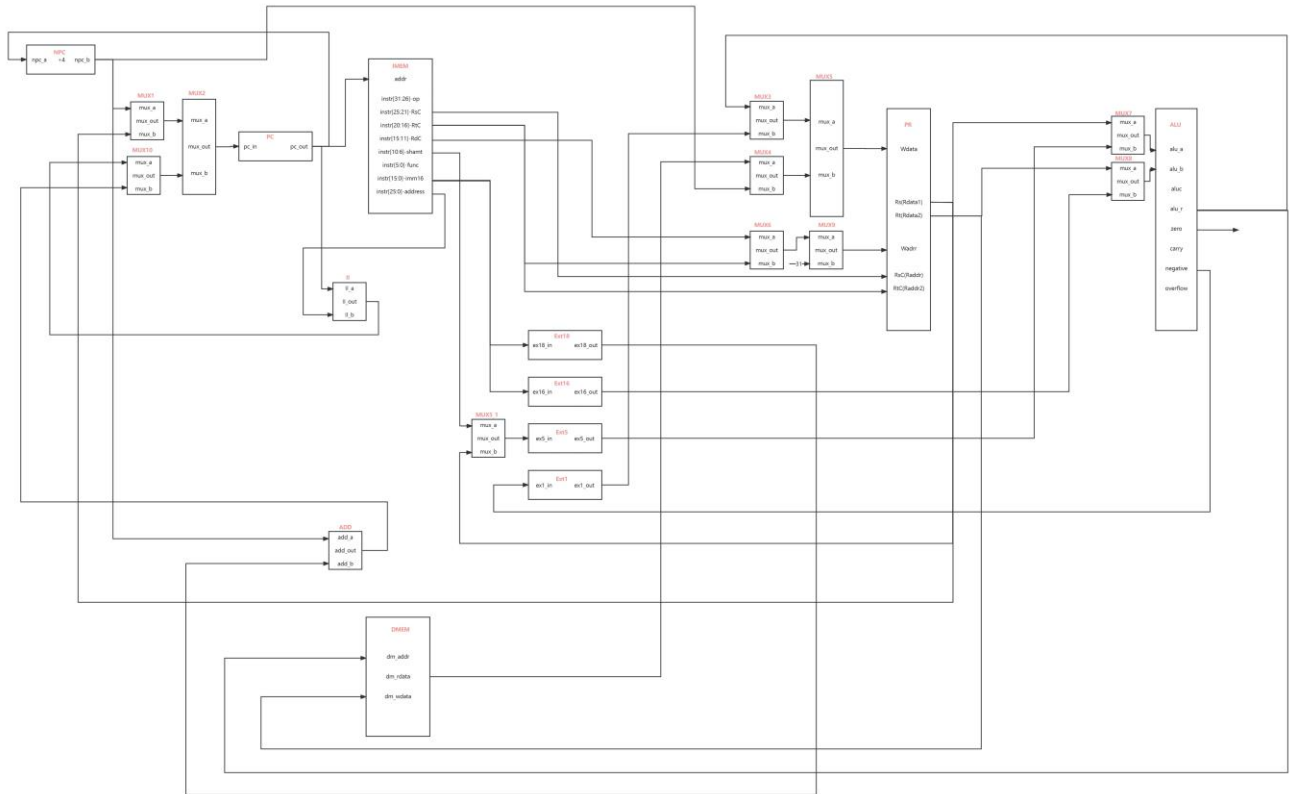
实验步骤：

- 1.新建 Vivado 工程，编写各个模块。
- 2.用 ModelSim 前仿真逐条测试所有指令。
- 3.用 ModelSim 进行后仿真测试。
- 4.配置 XDC 文件，综合下板，并观察实验现象。
- 5.按照要求书写实验报告

二、硬件逻辑图

	PC	NPC	IMEM	DMEM	RegFile								ALU					Ext16	Ext5	Ext18			Ext1	J				ADD			
					in				out				in					out	in	out	in	out	in	out	in				out	A	B
				Addr	Data	Wdata(大部分是Rd)	Waddr(大部分是RdC)	Raddr1(RsC)	Raddr2(RtC)	Rdata1(Rs)	Rdata2(Rt)	A	B	zero	carry	negative	overflow	in	out	in	out	in	out	in	out	A	B	A	B		
add	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	+														
addu	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	\														
sub	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	+														
subu	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	\														
and	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	\														
or	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	\														
xor	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	\														
nor	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	\	\														
slt	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Rs	Rt	\	\	+	\								AL+						
sltu	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	0 Rs	0 Rt	\	\	+	\								AL+						
sll	NPC	PC+4	PC			ALU	RdC	\	RtC	\	+	Ext5	Rt	\	\	\	\							0 shamt	+						
srl	NPC	PC+4	PC			ALU	RdC	\	RtC	\	+	Ext5	Rt	\	\	\	\							0 shamt	+						
sra	NPC	PC+4	PC			ALU	RdC	\	RtC	\	+	Ext5	Rt	\	\	\	\							0 shamt	+						
slv	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Ext5(Rs)	Rt	\	\	\	\							0 Rs[4:0]	+						
srlv	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Ext5(Rs)	Rt	\	\	\	\							0 Rs[4:0]	+						
srav	NPC	PC+4	PC			ALU	RdC	RsC	RtC	+	+	Ext5(Rs)	Rt	\	\	\	\							0 Rs[4:0]	+						
jr	Rs	PC+4	PC					RsC		+	\	Ext5(Rs)	Rt	\	\	\	\														
addi	NPC	PC+4	PC			ALU	RtC	RsC	\	+	\	Rs	Ext16	\	\	\	+	imm16	+												
addiu	NPC	PC+4	PC			ALU	RtC	RsC	\	+	\	Rs	Ext16	\	\	\	\	imm16	+												
andi	NPC	PC+4	PC			ALU	RtC	RsC	\	+	\	Rs	Ext16	\	\	\	\	0 imm16	+												
ori	NPC	PC+4	PC			ALU	RtC	RsC	\	+	\	Rs	Ext16	\	\	\	\	0 imm16	+												
xori	NPC	PC+4	PC			ALU	RtC	RsC	\	+	\	Rs	Ext16	\	\	\	\	0 imm16	+												
lw	NPC	PC+4	PC	ALU		DMEM	RtC	RsC	\	+	\	Rs	Ext16	\	\	\	\	imm16	+												
sw	NPC	PC+4	PC	ALU	+			RsC	RtC	+	+	Rs	Ext16	\	\	\	\	imm16	+												
beq	ADD/	PC+4	PC					RsC	RtC	+	+	Rs	Rt	+	\	\	\								imm16[0]^2	+			NPC		
bne	ADD/	PC+4	PC					RsC	RtC	+	+	Rs	Rt	+	\	\	\								imm16[0]^2	+			NPC		
slti	NPC	PC+4	PC			Ext1	RtC	RsC	\	+	\	RS	Ext16	\	\	\	\	imm16	+						AL+						
sltiu	NPC	PC+4	PC			Ext1	RtC	RsC	\	+	\	0 Ext1	\	\	\	\	\	imm16	+						AL+						
lui	NPC	PC+4	PC			ALU	RtC	\	\	\	\		Ext16	\	\	\	\	imm16 0	+												
j		PC+4	PC								\	\		\	\	\	\														
jal?		PC+4	PC			ADD	GPR[31]	\	\	\	\			\	\	\	\									PC31-28	IMEM25-0 0^2	+			
						NPC	31	\	\	\	\			\	\	\	\									PC31-28	IMEM25-0 0^2	+			
																													PC		
类型	4	1	1	1	1	4	3	1	1	\	\	2	2	\	\	\	\	1	\	2	\	1	\	1	1	\	\	1	1		
MUX	mux1					mux3	mux6					mux7	mux8							mux5_1											
	mux10					mux4	mux9																								
	mux2					mux5																									

	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	slv	srlv	srav	jr	addi	addiu	andi	ori	xori	lw	sw	beq	bne	slti	sltiu	lui	j	jal
aluc[3]	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1		0	0	0	0	0	0	0	0	0	1	1	1		
aluc[2]	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1		0	0	1	1	1	0	0	0	0	0	0	0		
aluc[1]	1	0	1	0	0	0	1	1	1	1	1	0	0	1	0	0		1	0	0	0	1	1	1	1	1	1	1	1	0	
aluc[0]	0	0	1	1	0	1	0	1	1	0	x(0)	1	0	x(0)	1	0		0	0	0	1	0	0	0	1	1	1	0	x(0)		
mux1_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	slv	srlv	srav		addi	addiu	andi	ori	xori	lw	sw	(beq)	(bne)	slti	sltiu	lui		
mux10_choose																													j	jal	
mux2_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	slv	srlv	srav	jr	addi	addiu	andi	ori	xori	lw	sw	(beq)	(bne)	slti	sltiu	lui		
mux3_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	slv	srlv	srav		addi	addiu	andi	ori	xori					slti	sltiu	lui		
mux4_choose																						lw					slti	sltiu	lui		
mux5_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	slv	srlv	srav		addi	addiu	andi	ori	xori					slti	sltiu	lui		
mux6_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	slv	srlv	srav		addi	addiu	andi	ori	xori					slti	sltiu	lui		
mux9_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	slv	srlv	srav		addi	addiu	andi	ori	xori	lw				slti	sltiu	lui		
mux7_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu								addi	addiu	andi	ori	xori	lw	sw	beq	bne	slti	sltiu			
mux8_choose	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	slv	srlv	srav									beq	bne					
mux5_1choose											sll	srl	sra																		
rf_we	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	slv	srlv	srav		addi	addiu	andi	ori	xori	lw				slti	sltiu	lui		jal
add_overflow																															
ext5_signed_ext											sll		sra	slv		srav															
ext16_signed_ext																		addi	addiu				lw	sw			slti	sltiu			
ext16_left																												lui			
dm_ena																							lw	sw							
dm_w																								sw							
dm_r																							lw								



三、模块建模

（该部分要求对实验中建模的所有模块进行功能描述，并列出各模块建模的verilog 代码）

PC:

```
`timescale 1ns / 1ps

module PC_reg(
    input clk,
    input rst,
    input ena,
    input [31:0] PC_in,
    output [31:0] PC_out
);
    reg [31:0] PC_r;
    always @(negedge clk or posedge rst)
    begin
        if (ena) begin
            if (rst)
                begin
                    PC_r <= 32'h0040_0000; //Mars 中的 PC 是从 0x0040_0000 开始
                end
            else
                PC_r <= PC_in;
        end
    end
    PC_out <= PC_r;
endmodule
```

```

        end
        else
        begin
            PC_r <= PC_in;
        end
    end
end

//assign PC_out = (ena && !rst) ? PC_r : 32'hz;
assign PC_out = (ena) ? PC_r : 32'hz;
endmodule

```

NPC:

```

`timescale 1ns / 1ps

module NPC(
    input [31:0] a,
    input rst,
    output [31:0] b
);

    assign b = rst ? a : a + 4;
endmodule

```

Regfiles:

```

module Regfiles(clk,rst,we,raddr1,raddr2,waddr,wdata,rdata1,rdata2);
input clk,rst,we;//we 高电位写
input[4:0]raddr1,raddr2,waddr;//raddr1-RsC raddr2-RtC waddr-RdC
input[31:0]wdata;//wdata-Rd
output [31:0]rdata1,rdata2;//rdata1-Rs rdata2-Rt

reg [31:0] array_reg[31:0];
integer i; // 定义整型变量 i, 用于循环迭代

always @(posedge clk or posedge rst) begin
    if (rst) begin // 复位时清空所有寄存器内容
        for (i = 0; i < 32; i = i + 1) begin
            array_reg[i] <= 32'h00000000;
        end
    end else begin
        if (we && waddr != 5'b0) begin // 写操作
            array_reg[waddr] <= wdata;
        end
    end
end
end

```

```

end
    // 读操作
assign rdata1 = array_reg[raddr1];
assign rdata2 = array_reg[raddr2];

endmodule

```

Ext:

```

`timescale 1ns / 1ps

module Extend18(
    input [15:0] a,
    //input signed_ext, //不需要无符号扩展
    output [31:0] b
);

    assign b = {{(14){a[15]}}, a[15:0], 2'b00};
endmodule

```

```

`timescale 1ns / 1ps

module Extend16(
    input [15:0] a,
    input signed_ext,
    //input left,
    output [31:0] b
);

    assign b = signed_ext ? {{(16){a[15]}}, a[15:0]} : {{(16){1'b0}},
a[15:0]};
endmodule

```

```

`timescale 1ns / 1ps

module Extend5(
    input [4:0] a,
    input signed_ext,
    output [31:0] b
);

    assign b = signed_ext ? {{(27){a[4]}}, a[4:0]} : {{(27){1'b0}}, a[4:0]};
endmodule

```

MUX:

```
`timescale 1ns / 1ps

module Mux(
    input [31:0] a,
    input [31:0] b,
    input choose,
    output [31:0] z
);
    reg [31:0] r;
    always @(*)
    begin
        case(chOOSE)
            1'b1:r <= a;
            1'b0:r <= b;
            default:r <= 5'bz;
        endcase
    end

    assign z = r;
endmodule
```

```
`timescale 1ns / 1ps

module Mux_5(
    input [4:0] a,
    input [4:0] b,
    input choose,
    output reg [4:0] z
);

    always @(*)
    begin
        case(chOOSE)
            1'b1:z <= a;
            1'b0:z <= b;
            default:z <= 5'bz;
        endcase
    end
endmodule
```

ALU:

```
`timescale 1ns / 1ps
```



```

module ALU(a,b,aluc,r,zero,carry,negative,overflow);
input [31:0] a,b;
input [3:0] aluc;
output reg [31:0] r;
output reg zero,carry,negative,overflow;

always@(*)
begin
    case(aluc)
        //add
        4'b0010:
            begin
                r=a+b;
                overflow=((a[31]==b[31])&&(~r[31]==a[31]))?1:0;
                zero=(r==0)?1:0;
                carry=0;
                negative=(r<0)?1:0;
            end
        //addu
        4'b0000:
            begin
                {carry,r}=a+b;
                zero=(r==0)?1:0;
                overflow=0;
                negative=(r[31]==1)?1:0;
            end
        //sub
        4'b0011:
            begin
                r=a-b;
                overflow=((a[31]==0 && b[31]==1 && r[31]==1) || (a[31]==1 &&
b[31]==0 && r[31]==0))?1:0;
                zero=(a==b)?1:0;
                carry=0;
                negative=(r<0)?1:0;
            end
        //subu
        4'b0001:
            begin
                {carry,r}=a-b;
                zero=(r==0)?1:0;
                overflow=0;
                negative=(r[31]==1)?1:0;
            end
    endcase
end

```

```

//and
4'b0100:
    begin
        r=a&b;
        zero=(r==0)?1:0;
        carry=0;
        overflow=0;
        negative=(r[31]==1)?1:0;
    end

//or
4'b0101:
    begin
        r=a|b;
        zero=(r==0)?1:0;
        carry=0;
        overflow=0;
        negative=(r[31]==1)?1:0;
    end

//xor
4'b0110:
    begin
        r=a^b;
        zero=(r==0)?1:0;
        carry=0;
        overflow=0;
        negative=(r[31]==1)?1:0;
    end

//nor
4'b0111:
    begin
        r=~(a|b);
        zero=(r==0)?1:0;
        carry=0;
        overflow=0;
        negative=(r[31]==1)?1:0;
    end

//slt
11'b1011:
    begin
        if(a[31]==1 && b[31]==0)
            r=1;
        else if(a[31]==0 && b[31]==1)
            r=0;
        else

```

```

        r=(a<b)?1:0;
    overflow=r;
    zero=(r==0)?1:0;
    carry=0;
    negative=(a<b)?1:0;
end
//sltu
4'b1010:
    begin
        r=(a<b)?1:0;
        carry=r;
        zero=(r==0)?1:0;
        overflow=0;
        negative=(r[31]==1)?1:0;
    end
//sll/slr
4'b1110:
    begin
        {carry,r}=b<<a;
        overflow=0;
        zero=(r==0)?1:0;
        negative=(r[31]==1)?1:0;
    end
//srl
4'b1101:
    begin
        r=b>>a;
        carry=b[a-1];
        overflow=0;
        zero=(r==0)?1:0;
        negative=(r[31]==1)?1:0;
    end
//sra
4'b1100:
    begin
        r=($signed(b))>>>a;
        carry=b[a];
        overflow=0;
        zero=(r==0)?1:0;
        negative=(r[31]==1)?1:0;
    end
//lui
4'b1000:
    begin

```

```

        r={b[15:0],16'b0};
        carry=0;
        overflow=0;
        zero=(r==0)?1:0;
        negative=(r[31]==1)?1:0;
    end
endcase
end
endmodule

```

II:

```

`timescale 1ns / 1ps

module II(
    input [3:0] a,
    input [25:0] b,
    output [31:0] r
);

    assign r = {a, b, 2'b00};
endmodule

```

ADD:

```

`timescale 1ns / 1ps

module add(
    input [31:0] a,
    input [31:0] b,
    output [31:0] r,
    output overflow
);

    assign r=a+b;
    assign overflow=((a[31]==b[31])&&(~r[31]==a[31]))?1:0; //照着以前写的
    ALU 搬过来的
endmodule

```

CPU + control + connect:

```

`timescale 1ns / 1ps

module cpu_top(
    input clk,
    input ena,

```

```

input rst,

input [31:0] IM_inst,
input [31:0] DM_rdata,

output DM_ena,
output DM_W,
output DM_R,
output [10:0] addr,
output [31:0] DM_wdata,
output [31:0] PC_out,
output [31:0] ALU_out
);

//imem
wire [31:0] imem;
assign imem = IM_inst;

//dmem
wire [31:0] dm_rdata = DM_rdata;
wire [31:0] dm_wdata;
reg [10:0] dm_addr;
wire dm_ena;
wire dm_w, dm_r;

//assign addr = dm_addr;

assign DM_ena = dm_ena;
assign DM_W = dm_w;
assign DM_R = dm_r;
assign DM_wdata = dm_wdata;

//pc_reg
//wire pc_rst;
//wire pc_ena;
wire [31:0] pc_in;
wire [31:0] pc_out;//
assign PC_out = pc_out;

//npc
wire [32:0] npc_a, npc_b;

```

```

//alu
wire ZF, CF, NF, OF;
wire [31:0] alu_a, alu_b, alu_r;
wire [4:0] aluc;
assign ALU_out = alu_r;

//regfiles
wire rf_clk = ~clk;
wire rf_we;
wire [4:0] RsC, RtC, RdC; //raddr1-RsC raddr2-RtC waddr-RdC
wire [31:0] Rd; //wdata-Rd
wire [31:0] Rs, Rt; //rdata1-Rs rdata2-Rt

//mux
wire [31:0] mux1_a, mux1_b, mux1_z, mux2_a, mux2_b, mux2_z, mux3_a,
mux3_b, mux3_z,
mux4_a, mux4_b, mux4_z, mux5_a, mux5_b, mux5_z, mux6_a,
mux6_b, mux6_z,
mux7_a, mux7_b, mux7_z, mux8_a, mux8_b, mux8_z, mux9_a,
mux9_b, mux9_z,
mux10_a, mux10_b, mux10_z;

wire mux1_choose, mux2_choose, mux3_choose,
mux4_choose, mux5_choose, mux6_choose,
mux7_choose, mux8_choose, mux9_choose,
mux10_choose;

//mux_5
wire [4:0] mux5_1a, mux5_1b, mux5_1z;
wire mux5_1choose;
//assign mux5_1a[4:0] = imem[10:6];

//Extend1
wire ext1_a;
wire [31:0] ext1_b;

//Extend5
wire [4:0] ext5_a;
wire ext5_signed_ext;
wire [31:0] ext5_b;

//Extend16
wire [15:0] ext16_a;

```

```

wire ext16_signed_ext;
//wire ext16_left;
wire [31:0] ext16_b;

//Extend18
wire [15:0] ext18_a;
wire ext18_signed_ext;
wire [31:0] ext18_b;

//II
wire [3:0] ii_a;
wire [25:0] ii_b;
wire [31:0] ii_r;

//add
wire [31:0] add_a;
wire [31:0] add_b;
wire [31:0] add_r;
wire add_overflow;

//connect
assign PC_out = pc_out;
assign npc_a = pc_out;
assign mux1_a = npc_b;
assign mux2_a = mux1_z;
assign pc_in = mux2_z;
assign ii_a = pc_out[31:28];
assign mux10_a = ii_r;
assign mux10_b = add_r;
assign mux2_b = mux10_z;
assign ii_b = imem[25:0];
assign mux1_b = Rs;
assign add_a = npc_b;
assign add_b = ext18_b;
assign mux4_b = npc_b;
assign mux4_a = dm_rdata;
assign mux5_b = mux4_z;
//assign mux11_a = alu_r;
//assign mux11_b = ext16_b;
assign mux3_a = alu_r;
assign mux3_b = ext1_b;
assign mux5_a = mux3_z;
assign Rd = mux5_z;
assign mux6_a = imem[15:11];

```

```

assign mux6_b = imem[20:16];
assign mux9_a = mux6_z;
assign mux9_b = 31;
assign RdC = mux9_z;
assign RsC = imem[25:21];
assign RtC = imem[20:16];
assign ext18_a = imem[15:0];
assign ext16_a = imem[15:0];
assign mux5_1a = imem[10:6];
assign mux5_1b = Rs;
assign ext5_a = mux5_1z;
assign ext1a = NF;
assign mux8_b = ext16_b;
assign mux7_b = ext5_b;
assign mux7_a = Rs;
assign mux8_a = Rt;
assign alu_a = mux7_z;
assign alu_b = mux8_z;
//assign dm_addr = alu_r[10:0];
assign dm_wdata = Rt;
assign addr = alu_r[10:0];

//31
wire _add, _addu, _sub, _subu, _and, _or, _xor, _nor;
wire _slt, _sltu, _sll, _srl, _sra, _sllv, _srlv, _sra, _j, _jr;
wire _addi, _addiu, _andi, _ori, _xori, _lw, _sw;
wire _beq, _bne, _slti, _sltiu, _lui, _j, _jal;

//decode
//1~17
assign _add =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100000)?1'b1:1'b0;
assign _addu =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100001)?1'b1:1'b0;
assign _sub =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100010)?1'b1:1'b0;
assign _subu =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100011)?1'b1:1'b0;
assign _and =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100100)?1'b1:1'b0;
assign _or =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100101)?1'b1:1'b0;
assign _xor =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100110)?1'b1:1'b0;

```



```

    assign _nor =
(imem[31:26]==6'b000000&&imem[5:0]==6'b100111)?1'b1:1'b0;

    assign _slt =
(imem[31:26]==6'b000000&&imem[5:0]==6'b101010)?1'b1:1'b0;
    assign _sltu =
(imem[31:26]==6'b000000&&imem[5:0]==6'b101011)?1'b1:1'b0;
    assign _sll =
(imem[31:26]==6'b000000&&imem[5:0]==6'b000000)?1'b1:1'b0;
    assign _srl =
(imem[31:26]==6'b000000&&imem[5:0]==6'b000010)?1'b1:1'b0;
    assign _sra =
(imem[31:26]==6'b000000&&imem[5:0]==6'b000011)?1'b1:1'b0;
    assign _sllv =
(imem[31:26]==6'b000000&&imem[5:0]==6'b000100)?1'b1:1'b0;
    assign _srlv =
(imem[31:26]==6'b000000&&imem[5:0]==6'b000110)?1'b1:1'b0;
    assign _srav =
(imem[31:26]==6'b000000&&imem[5:0]==6'b000111)?1'b1:1'b0;
    assign _jr =
(imem[31:26]==6'b000000&&imem[5:0]==6'b001000)?1'b1:1'b0;

//18~29
assign _addi = (imem[31:26]==6'b001000)?1'b1:1'b0;
assign _addiu = (imem[31:26]==6'b001001)?1'b1:1'b0;
assign _andi = (imem[31:26]==6'b001100)?1'b1:1'b0;
assign _ori = (imem[31:26]==6'b001101)?1'b1:1'b0;
assign _xori = (imem[31:26]==6'b001110)?1'b1:1'b0;
assign _lw = (imem[31:26]==6'b100011)?1'b1:1'b0;
assign _sw = (imem[31:26]==6'b101011)?1'b1:1'b0;
assign _beq = (imem[31:26]==6'b000100)?1'b1:1'b0;
assign _bne = (imem[31:26]==6'b000101)?1'b1:1'b0;
assign _slti = (imem[31:26]==6'b001010)?1'b1:1'b0;
assign _sltiu = (imem[31:26]==6'b001011)?1'b1:1'b0;
assign _lui = (imem[31:26]==6'b001111)?1'b1:1'b0;

//30 31
assign _j = (imem[31:26]==6'b000010)?1'b1:1'b0;
assign _jal = (imem[31:26]==6'b000011)?1'b1:1'b0;

//control
assign aluc[3] = _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv
|| _srav || _slti || _sltiu || _lui;

```

```

    assign aluc[2] = _and || _or || _xor || _nor || _sll || _srl || _sra ||
    _sllv || _srlv || _srav || _andi || _ori || _xori;
    assign aluc[1] = _add || _sub || _xor || _nor || _slt || _sltu || _sll
    || _sllv || _addi || _xori || _lw || _sw || _beq || _bne || _slti || _sltiu;
    assign aluc[0] = _sub || _subu || _or || _nor || _slt || _srl || _srlv
    || _ori || _beq || _bne || _slti;

    assign mux1_choose = _add || _addu || _sub || _subu || _and || _or ||
    _xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
    _srav || _addi || _addiu || _andi || _ori || _xori || _lw || _sw || _slti
    || _sltiu || _lui + ((_beq) && (!ZF)) + ((_bne) && (ZF));
    assign mux10_choose = (_j || _jal);
    assign mux2_choose = (_add || _addu || _sub || _subu || _and || _or ||
    _xor || _nor || _slt || _sltu || _sll || _srl || _sra || _jr || _sllv ||
    _srlv || _srav || _addi || _addiu || _andi || _ori || _xori || _lw || _sw
    || _slti || _sltiu || _lui) + ((_beq) && (!ZF)) + ((_bne) && (ZF));

    //assign mux11_choose = _add || _addu || _sub || _subu || _and || _or
    || _xor || _nor || _sll || _srl || _sra || _sllv || _srlv || _srav || _addi
    || _addiu || _andi || _ori || _xori;
    assign mux3_choose = _add || _addu || _sub || _subu || _and || _or ||
    _xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
    _srav || _addi || _addiu || _andi || _ori || _xori || _slti || _sltiu ||
    _lui;
    assign mux4_choose = _lw;
    assign mux5_choose = _add || _addu || _sub || _subu || _and || _or ||
    _xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
    _srav || _addi || _addiu || _andi || _ori || _xori || _slti || _sltiu ||
    _lui;

    assign mux6_choose = _add || _addu || _sub || _subu || _and || _or ||
    _xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
    _srav;
    assign mux7_choose = _add || _addu || _sub || _subu || _and || _or ||
    _xor || _nor || _slt || _sltu || _addi || _addiu || _andi || _ori || _xori
    || _lw || _sw || _beq || _bne || _slti || _sltiu;
    assign mux8_choose = _add || _addu || _sub || _subu || _and || _or ||
    _xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
    _srav || _beq || _bne;
    assign mux9_choose = _add || _addu || _sub || _subu || _and || _or ||
    _xor || _nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv ||
    _srav || _addi || _addiu || _andi || _ori || _xori || _lw || _slti || _sltiu
    || _lui;

```

```

assign mux5_1choose = _sll || _srl || _sra;

assign rf_we = _add || _addu || _sub || _subu || _and || _or || _xor ||
_nor || _slt || _sltu || _sll || _srl || _sra || _sllv || _srlv || _srav
|| _addi || _addiu || _andi || _ori || _xori || _lw || _slti || _sltiu ||
_lui || _jal;
assign add_overflow = 0;
assign ext5_signed_ext = 0;
assign ext16_signed_ext = _addi || _addiu || _lw || _sw || _slti || _sltiu;
//assign ext16_left = _lui;

assign dm_ena = _sw || _lw;
assign dm_r = _lw;
assign dm_w = _sw;

NPC npc(
    .a(npc_a),
    .rst(rst),
    .b(npc_b)
);

PC_reg pc_reg(
    .clk(clk),
    .rst(rst),
    .ena(ena),
    .PC_in(pc_in),
    .PC_out(pc_out)
);

Regfiles cpu_ref(
    .clk(rf_clk),
    .rst(rst),
    .we(rf_we), //高电平可以写入
    .raddr1(RsC),
    .raddr2(RtC),
    .waddr(RdC),
    .wdata(Rd),
    .rdata1(Rs),
    .rdata2(Rt)
);

ALU alu(
    .a(alu_a),
    .b(alu_b),

```

```
.aluc(aluc),
.r(alu_r),
.zero(ZF),
.carry(CF),
.negative(NF),
.overflow(OF)
);

//Mux
Mux mux1(
    .a(mux1_a),
    .b(mux1_b),
    .choose(mux1_choose),
    .z(mux1_z)
);
Mux mux2(
    .a(mux2_a),
    .b(mux2_b),
    .choose(mux2_choose),
    .z(mux2_z)
);
Mux mux3(
    .a(mux3_a),
    .b(mux3_b),
    .choose(mux3_choose),
    .z(mux3_z)
);
Mux mux4(
    .a(mux4_a),
    .b(mux4_b),
    .choose(mux4_choose),
    .z(mux4_z)
);
Mux mux5(
    .a(mux5_a),
    .b(mux5_b),
    .choose(mux5_choose),
    .z(mux5_z)
);
Mux mux6(
    .a(mux6_a),
    .b(mux6_b),
    .choose(mux6_choose),
    .z(mux6_z)
```

```

);
Mux mux7(
    .a(mux7_a),
    .b(mux7_b),
    .choose(mux7_choose),
    .z(mux7_z)
);
Mux mux8(
    .a(mux8_a),
    .b(mux8_b),
    .choose(mux8_choose),
    .z(mux8_z)
);
Mux mux9(
    .a(mux9_a),
    .b(mux9_b),
    .choose(mux9_choose),
    .z(mux9_z)
);
Mux mux10(
    .a(mux10_a),
    .b(mux10_b),
    .choose(mux10_choose),
    .z(mux10_z)
);
/*Mux mux11(
    .a(mux11_a),
    .b(mux11_b),
    .choose(mux11_choose),
    .z(mux11_z)
);*/

Mux_5 mux5_1(
    .a(mux5_1a),
    .b(mux5_1b),
    .choose(mux5_1choose),
    .z(mux5_1z)
);

Extend1 ext1(
    .a(ext1_a),
    .b(ext1_b)
);
Extend5 ext5(

```

```

        .a(ext5_a),
        .signed_ext(ext5_signed_ext),
        .b(ext5_b)
    );
    Extend16 ext16(
        .a(ext16_a),
        .signed_ext(ext16_signed_ext),
        // .left(ext16_left),
        .b(ext16_b)
    );
    Extend18 ext18(
        .a(ext18_a),
        .b(ext18_b)
    );
    II ii(
        .a(ii_a),
        .b(ii_b),
        .r(ii_r)
    );
    add add(
        .a(add_a),
        .b(add_b),
        .r(add_r),
        .overflow(add_overflow)
    );

endmodule

```

IMEM:

```

`timescale 1ns / 1ps

module IMEM(
    input [10:0] addr,
    output [31:0] instr
);

    dist_mem_gen_0 instr_mem(
        .a(addr),
        .spo(instr)
    );

endmodule

```

DMEM:

```

`timescale 1ns / 1ps

```

```

module DMEM(
    input clk,
    input ena, //高电位有效

    input Dmem_W, //控制写
    input Dmem_R, //控制读
    input [10:0] Dmem_addr, //输入的地址
    input [31:0] Dmem_wdata, //写入的数据
    output [31:0] Dmem_rdata //读取的数据
);

    reg [31:0] D_mem[0:1023];

    always @(posedge clk) begin
        if (Dmem_W && ena)
            begin
                D_mem[Dmem_addr] <= Dmem_wdata;
            end
        end

        assign Dmem_rdata = (Dmem_R && ena) ? D_mem[Dmem_addr] : 32'bz; //不可
        读时改为高阻
    endmodule

```

sccomp_dataflow:

```

`timescale 1ns / 1ps

module sccomp_dataflow(
    input clk_in,
    input reset,
    output [31:0] inst,
    output [31:0] pc
);

    wire [31:0] instr;
    wire dw, dr, dena;
    wire [31:0] w_data, r_data;
    wire [10:0] dm_addr;
    wire [10:0] im_addr;

    assign inst = instr;
    assign im_addr = (pc - 32'h0040_0000) / 4;

```

```

cpu_top sccpu(
    .clk(clk_in),
    .ena(1'b1),
    .rst(reset),

    .IM_inst(instr), //i ->
    .DM_rdata(r_data), //d ->

    .DM_ena(dena), //pc ->
    .DM_W(dw), //-> d
    .DM_R(dr), //-> d
    .addr(dm_addr),
    .DM_wdata(w_data), //-> d
    .PC_out(pc), //-> i
    .ALU_out(res) //-> d
);

IMEM imemory(
    .addr(im_addr),
    .instr(instr)
);

DMEM dmemory(
    .clk(clk_in),
    .ena(1'b1),

    .Dmem_W(dw),
    .Dmem_R(dr),

    .Dmem_addr(dm_addr),
    .Dmem_wdata(w_data),
    .Dmem_rdata(r_data)
);
endmodule

```

四、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

test:

```
`timescale 100ps / 1ps
```

```
module test();
```


[illegible]

```

        $fdisplay(file_open, "regfile13: %h",
test.uut.sccpu.cpu_ref.array_reg[13]);
        $fdisplay(file_open, "regfile14: %h",
test.uut.sccpu.cpu_ref.array_reg[14]);
        $fdisplay(file_open, "regfile15: %h",
test.uut.sccpu.cpu_ref.array_reg[15]);
        $fdisplay(file_open, "regfile16: %h",
test.uut.sccpu.cpu_ref.array_reg[16]);
        $fdisplay(file_open, "regfile17: %h",
test.uut.sccpu.cpu_ref.array_reg[17]);
        $fdisplay(file_open, "regfile18: %h",
test.uut.sccpu.cpu_ref.array_reg[18]);
        $fdisplay(file_open, "regfile19: %h",
test.uut.sccpu.cpu_ref.array_reg[19]);
        $fdisplay(file_open, "regfile20: %h",
test.uut.sccpu.cpu_ref.array_reg[20]);
        $fdisplay(file_open, "regfile21: %h",
test.uut.sccpu.cpu_ref.array_reg[21]);
        $fdisplay(file_open, "regfile22: %h",
test.uut.sccpu.cpu_ref.array_reg[22]);
        $fdisplay(file_open, "regfile23: %h",
test.uut.sccpu.cpu_ref.array_reg[23]);
        $fdisplay(file_open, "regfile24: %h",
test.uut.sccpu.cpu_ref.array_reg[24]);
        $fdisplay(file_open, "regfile25: %h",
test.uut.sccpu.cpu_ref.array_reg[25]);
        $fdisplay(file_open, "regfile26: %h",
test.uut.sccpu.cpu_ref.array_reg[26]);
        $fdisplay(file_open, "regfile27: %h",
test.uut.sccpu.cpu_ref.array_reg[27]);
        $fdisplay(file_open, "regfile28: %h",
test.uut.sccpu.cpu_ref.array_reg[28]);
        $fdisplay(file_open, "regfile29: %h",
test.uut.sccpu.cpu_ref.array_reg[29]);
        $fdisplay(file_open, "regfile30: %h",
test.uut.sccpu.cpu_ref.array_reg[30]);
        $fdisplay(file_open, "regfile31: %h",
test.uut.sccpu.cpu_ref.array_reg[31]);
        $fdisplay(file_open, "pc: %h", pc);
        $fdisplay(file_open, "instr: %h", inst);

    end;
end

sccomp_dataflow uut(

```

```

        .clk_in(clk_in),
        .reset(reset),
        .inst(inst),
        .pc(pc)
    );

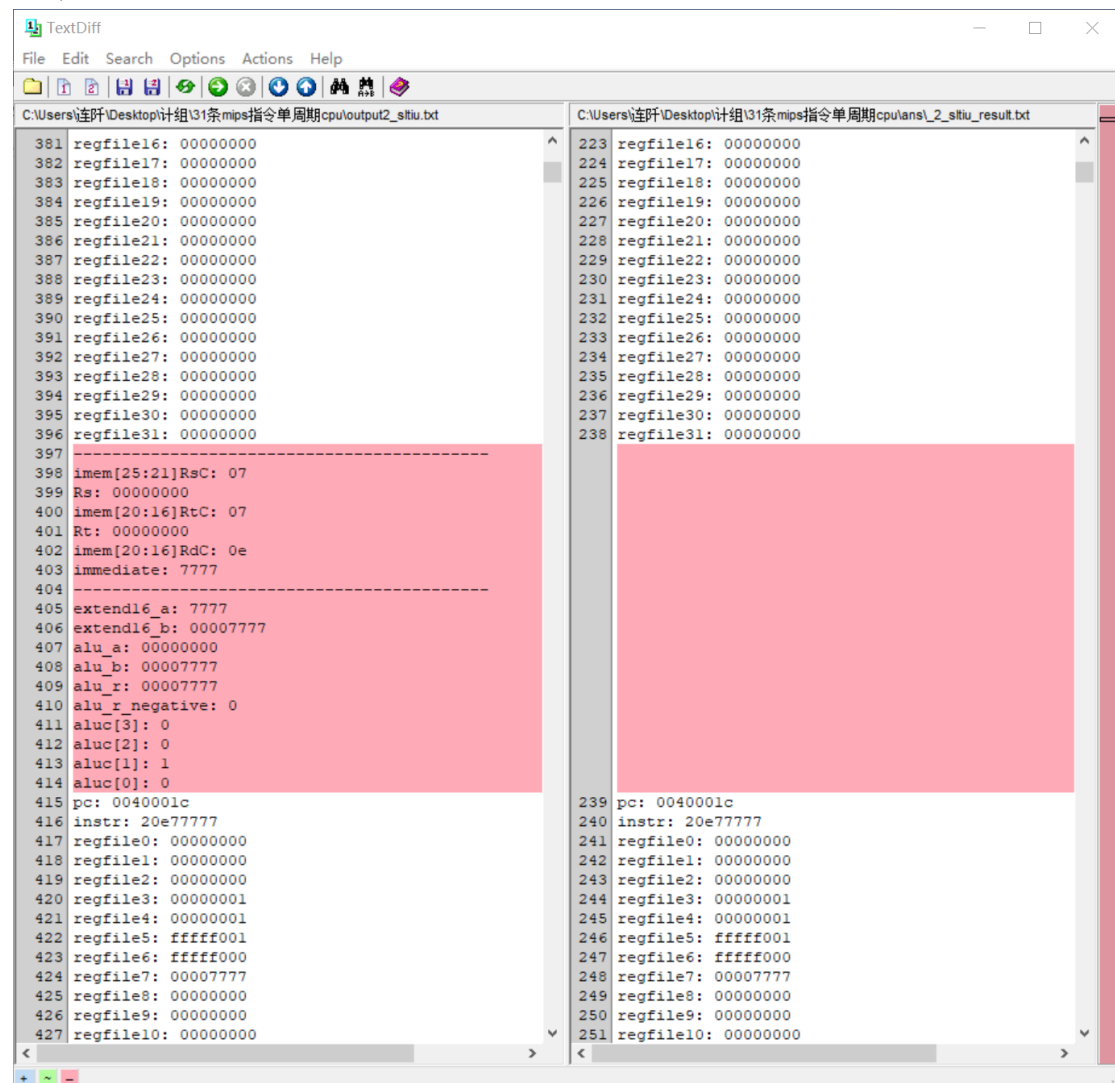
endmodule











































```

五、实验结果

前仿真：

比对通过：



 2_and.coe	2023/5/21 16:51	COE 文件
 2_lsw.coe	2023/5/21 16:56	COE 文件
 2_lsw2.coe	2023/5/21 17:00	COE 文件
 2_sll.coe	2023/5/21 17:04	COE 文件
 2_slt.coe	2023/5/21 19:03	COE 文件
 2_sltiu.coe	2023/5/21 19:06	COE 文件
 2_sra.coe	2023/5/21 18:58	COE 文件
 2_srlv.coe	2023/5/21 18:33	COE 文件
 3.5_beq.coe	2023/5/21 21:21	COE 文件
 3.5_bne.coe	2023/5/21 21:29	COE 文件
 3_j.coe	2023/5/21 19:11	COE 文件
 3_jal.coe	2023/5/21 21:43	COE 文件
 4_jr.coe	2023/5/21 22:15	COE 文件
 output2_addu.txt	2023/5/21 21:39	文本文档
 output2_and.txt	2023/5/22 18:11	文本文档
 output2_lsw.txt	2023/5/23 3:16	文本文档
 output2_lsw2.txt	2023/5/23 2:44	文本文档
 output2_sll.txt	2023/5/21 18:27	文本文档
 output2_slt.txt	2023/5/23 14:20	文本文档
 output2_sltiu.txt	2023/5/23 14:37	文本文档
 output2_sra.txt	2023/5/21 19:00	文本文档
 output2_srlv.txt	2023/5/21 18:35	文本文档
 output3.5_beq.txt	2023/5/23 10:10	文本文档
 output3.5_bne.txt	2023/5/23 10:12	文本文档
 output3_j.txt	2023/5/23 13:39	文本文档
 output3_jal.txt	2023/5/23 13:43	文本文档
 output4_jr.txt	2023/5/23 13:45	文本文档
 result2_add.txt	2023/5/21 16:30	文本文档
 result2_addu.txt	2023/5/21 16:36	文本文档
 result2_and.txt	2023/5/21 16:50	文本文档
 result2_lsw.txt	2023/5/21 16:55	文本文档
 result2_lsw2.txt	2023/5/21 16:59	文本文档
 result2_sll.txt	2023/5/21 17:03	文本文档
 result2_slt.txt	2023/5/21 19:02	文本文档
 result2_sltiu.txt	2023/5/21 19:05	文本文档
 result2_sra.txt	2023/5/21 18:58	文本文档
 result2_srlv.txt	2023/5/21 18:32	文本文档
 result3.5_beq.txt	2023/5/21 21:11	文本文档
 result3.5_bne.txt	2023/5/21 21:28	文本文档
 result3_j.txt	2023/5/21 19:10	文本文档
 result3_jal.txt	2023/5/21 21:42	文本文档
 result4_jr.txt	2023/5/21 22:15	文本文档

提交结果 AC:

上传作业

本实验截止时间：2023/5/24 0:00:00

还可以上传 1次

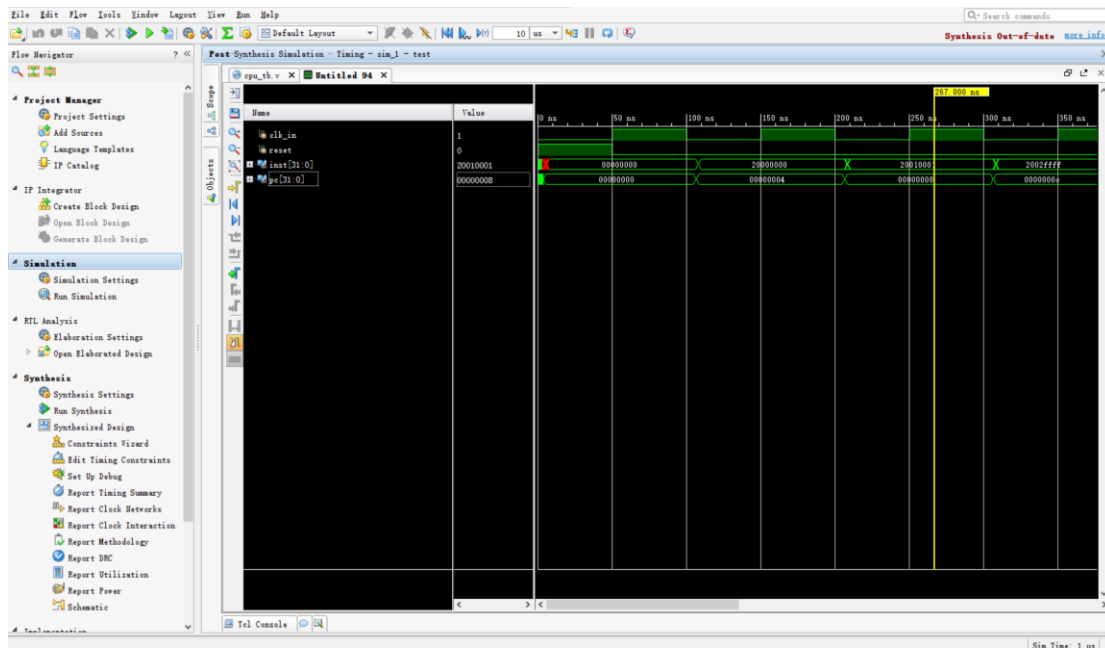
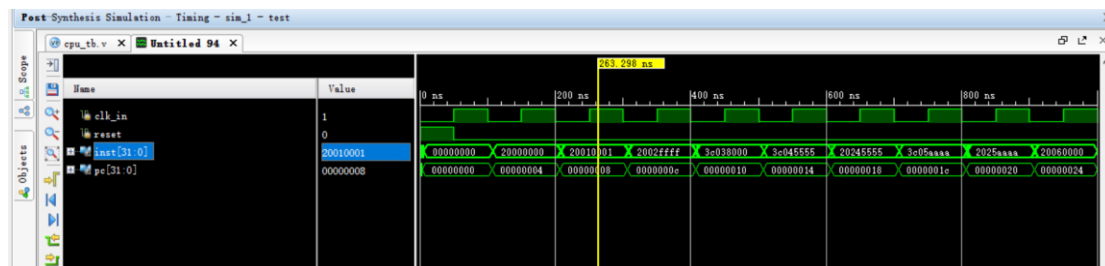
选择文件未选择文件

上传

本实验提交历史

实验	结果	提交时间	
9	AC	2023/5/23 14:58:56	--

后仿真:



下板:

```
3c1d1001
37bd0004
201c0000
3c1b1001
0c100413
00000000
20010002
20020002
```

