浙江大学

本科实验报告

课程名称：	计算机组成
姓　　名：	姜雨童
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
邮　　箱：	3220103450@zju.edu.cn
QQ　　号：	1369218489
电　　话：	19550103468
指导教师：	马德
报告日期：	2024 年 5 月 20 日

浙江大学实验报告

课程名称：　计算机组成　　实验类型：　综合

实验项目名称：　Lab 4：单周期 CPU

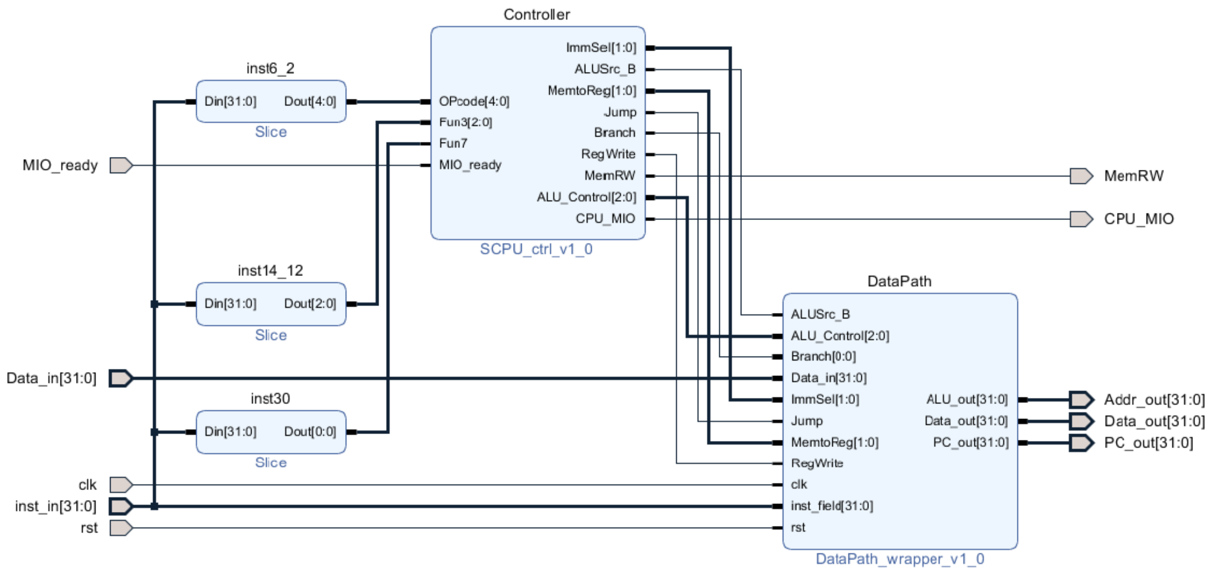
学生姓名：　姜雨童　学号：　33220103450　同组学生姓名：　/

实验地点：　紫金港东四 509 室　实验日期：　2024　年　4　月　15　日

一、操作方法与实验步骤

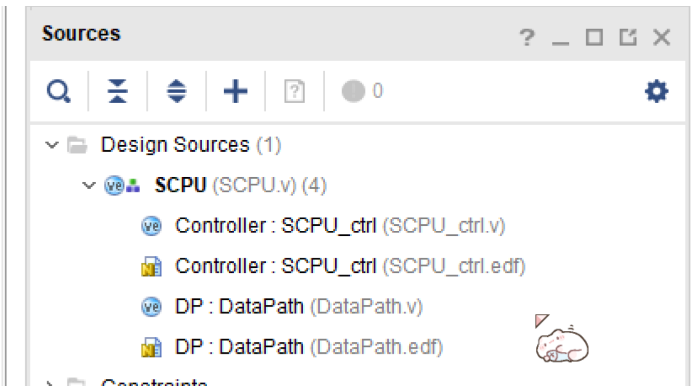
Lab04-0：集成 CPU 核

实验任务：利用数据通路和控制器两个 IP 核集成设计 CPU，逻辑原理图如下



编写代码如下：

```
module SCPU(
    input MIO_ready,
    input [31:0] Data_in,
    input clk,
    input [31:0] Inst_in,
    input rst,
    output MemRW,
    output CPU_MIO,
    output [31:0] Addr_out,
    output [31:0] Data_out,
    output [31:0] PC_out
    );
    wire [1:0] ImmSel;
    wire ALUSrc_B;
    wire [1:0] MemtoReg;
    wire Jump;
    wire Branch;
    wire RegWrite;
    wire [2:0] ALU_Control;

    SCPU_ctrl Controller(
        .OPcode(Inst_in[6:2]),.Fun3(Inst_in[14:12]),.Fun7(Inst_in[30]),.MIO_ready(MIO_rea
dy),
        .ImmSel(ImmSel),.ALUSrc_B(ALUSrc_B),.MemtoReg(MemtoReg),.Jump(Jump),
        .Branch(Branch),.RegWrite(RegWrite),.MemRW(MemRW),.ALU_Control(ALU_Control),.CPU_
MIO(CPU_MIO)
        );

    DataPath DP(
        .ALUSrc_B(ALUSrc_B),.ALU_Control(ALU_Control),.Branch(Branch),.Data_in(Data_in),
        .ImmSel(ImmSel),.Jump(Jump),.MemtoReg(MemtoReg),.RegWrite(RegWrite),.clk(clk),
        .inst_field(Inst_in),.rst(rst),.ALU_out(ALU_out),.Data_out(Data_out),.PC_out(PC_o
ut)
        );
endmodule
```
模块结构如图所示：


Sources
0
Design Sources (1)
SCPU (SCPU.v) (4)
Controller : SCPU_ctrl (SCPU_ctrl.v)
Controller : SCPU_ctrl (SCPU_ctrl.edf)
DP : DataPath (DataPath.v)
DP : DataPath (DataPath.edf)
Constraints

Lab04-3：CPU 设计（数据通路、控制器、指令集扩展）

（备注：Lab04-1 和 Lab04-2 经修改后得到 Lab04-3，此处不对前两个实验进行赘述）

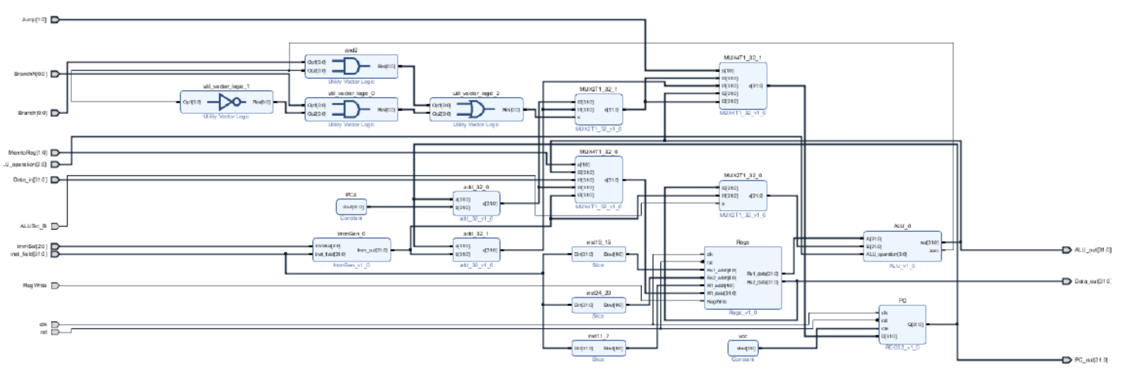
实验任务一：设计实现数据通路和控制器

要求至少包含以下指令：

R-Type： add, sub, and, or, xor, slt, sltu,srl, sra, sll；
I-Type： addi, andi, ori, xori, slti, sltiu,srli,srai,slli,lw,jalr;
S-Type： sw；
B-Type： beq,bne；
J-Type： Jal；
U-Type： lui；

根据数据通路图和控制信号真值表等内容可以完成该部分代码的编写。

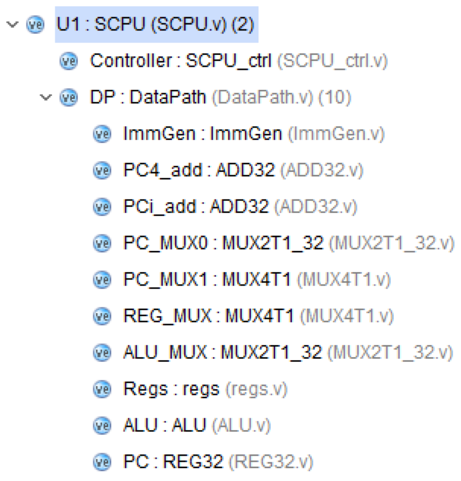
Datapath参考设计



控制信号真值表

nst[31:0]	Banch	Branch N	Jump	ImmSel	ALUSrc_B	ALU_Control	MemRW	RegWrite	MemtoReg
add	0	0	0	*	Reg (0)	Add	Read	1	ALU(0)
sub	0	0	0	*	Reg (0)	Sub	Read	1	ALU(0)
(R-R Op)	0	0	0	*	Reg (0)	*(Op)*	Read	1	ALU(0)
addi	0	0	0	I	Imm (1)	Add	Read	1	ALU(0)
lw	0	0	0	I	Imm (1)	Add	Read	1	Mem(1)
sw	0	0	0	S	Imm (1)	Add	Write	0	*
beq	0	0	0	B	Reg (0)	sub	*	0	*
beq	1	*	0	B	Reg (0)	sub	*	0	*
bne	0	0	0	B	Reg (0)	sub	*	0	*
bne	*	1	0	B	Reg (0)	sub	*	0	*
jalr	*	*	2	I	Imm (1)	Add	*	1	PC+4(2)
jal	*	*	1	J	Imm (1)	*	*	1	PC+4(2)
lui	0	0	0	U	*	*	*	1	Imm(3)

单周期流水线 CPU 的构架如下图所示：



```
U1 : SCPU (SCPU.v) (2)
    Controller : SCPU_ctrl (SCPU_ctrl.v)
    DP : DataPath (DataPath.v) (10)
        ImmGen : ImmGen (ImmGen.v)
        PC4_add : ADD32 (ADD32.v)
        PCi_add : ADD32 (ADD32.v)
        PC_MUX0 : MUX2T1_32 (MUX2T1_32.v)
        PC_MUX1 : MUX4T1 (MUX4T1.v)
        REG_MUX : MUX4T1 (MUX4T1.v)
        ALU_MUX : MUX2T1_32 (MUX2T1_32.v)
        Regs : regs (regs.v)
        ALU : ALU (ALU.v)
        PC : REG32 (REG32.v)
```

代码实现：

SCPU.v

```
module SCPU(
    input MIO_ready,
    input [31:0] Data_in,
    input clk,
    input [31:0] Inst_in,
    input rst,
    output MemRW,
    output CPU_MIO,
    output [31:0] Addr_out,
    output [31:0] Data_out,
    output [31:0] PC_out
    );
    wire [2:0] ImmSel;
    wire ALUSrc_B;
    wire [1:0] MemtoReg;
    wire [1:0] Jump;
    wire Branch;
    wire BranchN;
    wire RegWrite;
    wire [3:0] ALU_Control;

    SCPU_ctrl Controller(
        .OPcode(Inst_in[6:2]),.Fun3(Inst_in[14:12]),.Fun7(Inst_in[30]),.MIO_ready(MIO_rea
dy),
        .ImmSel(ImmSel),.ALUSrc_B(ALUSrc_B),.MemtoReg(MemtoReg),.Jump(Jump),
        .Branch(Branch),.BranchN(BranchN),.RegWrite(RegWrite),.MemRW(MemRW),.ALU_Control(
ALU_Control),.CPU_MIO(CPU_MIO)
        );
    DataPath DP(
        .ALUSrc_B(ALUSrc_B),.ALU_operation(ALU_Control),.Branch(Branch),.BranchN(BranchN)
,.Data_in(Data_in),
        .ImmSel(ImmSel),.Jump(Jump),.MemtoReg(MemtoReg),.RegWrite(RegWrite),.clk(clk),
        .inst_field(Inst_in),.rst(rst),.ALU_out(Addr_out),.Data_out(Data_out),.PC_out(PC_
out)
        );
endmodule
```

SCPU_ctrl.v

```
module SCPU_ctrl(
    input[4:0]OPcode,
    input[2:0]Fun3,
    input Fun7,
    input MIO_ready,
    output reg [2:0]ImmSel,
    output reg ALUSrc_B,
    output reg [1:0]MemtoReg,
    output reg [1:0]Jump,
    output reg Branch,
    output reg BranchN,
    output reg RegWrite,
    output reg MemRW,
    output reg [3:0]ALU_Control,
    output reg CPU_MIO
    );

    reg [1:0]ALUop;

    always@* begin
        case(OPcode)
        5'b00000:begin ImmSel<=3'b001; end  // I-lw
        5'b00100:begin ImmSel<=3'b001; end  // I-addi,slti...
        5'b11001:begin ImmSel<=3'b001; end  // I-jalr
        5'b01000:begin ImmSel<=3'b010; end  // S-sw
        5'b11000:begin ImmSel<=3'b011; end  // SB-beq,bne
        5'b11011:begin ImmSel<=3'b100; end  // J-jal
        5'b00101:begin ImmSel<=3'b000; end  // U
        5'b01101:begin ImmSel<=3'b000; end  // U-lui
         default:begin ImmSel<=3’000; end
        endcase
    end

    always@* begin
```

```
        case(OPcode)
        5'b01100:begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Jump,ALUop} = 9'b000100010; end
// R
        5'b01000:begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Jump,ALUop} = 9'b100010000; end
// sw
        5'b11000:begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Jump,ALUop} = 9'b000000001; end
// beq,bne
        5'b01101:begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Jump,ALUop} = 9'b011100000; end
// lui
        5'b00101:begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Jump,ALUop} = 9'b011100000; end
// U
        5'b00000:begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Jump,ALUop} = 9'b101100000; end
// lw - data_in from memory
        5'b11001:begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Jump,ALUop} = 9'b110101000; end
// jalr - store next PC
        5'b11011:begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Jump,ALUop} = 9'b110100100; end
// jal
        5'b00100:begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Jump,ALUop} = 9'b100100011; end
// I
       default: begin {ALUSrc_B,MemtoReg,RegWrite,MemRW,Jump,ALUop} = 9'b000100010; end
        endcase
        Branch = 1'b0;
        BranchN = 1'b0;
        if(OPcode==5'b11000 && Fun3==3'b000) Branch =  1'b1;
        if(OPcode==5'b11000 && Fun3==3'b001) BranchN = 1'b1;
    end

    always @* begin
        case(ALUop)
        2'b00: ALU_Control = 4'b0010 ;  // add
        2'b01: ALU_Control = 4'b0110 ;  // sub
        2'b10:  // R
            case({Fun3,Fun7})
            4'b0000: ALU_Control = 4'b0010 ;    // add
            4'b0001: ALU_Control = 4'b0110 ;    // sub
            4'b0010: ALU_Control = 4'b1110 ;    // sll
            4'b0100: ALU_Control = 4'b0111 ;    // slt
            4'b0110: ALU_Control = 4'b1001 ;    // sltu
            4'b1000: ALU_Control = 4'b1100 ;    // xor
            4'b1010: ALU_Control = 4'b1101 ;    // srl
            4'b1011: ALU_Control = 4'b1111 ;    // sra
            4'b1100: ALU_Control = 4'b0001 ;    // or
            4'b1110: ALU_Control = 4'b0000 ;    // and
           default:  ALU_Control = 4’b0000 ;
```

```
            endcase
        2'b11:  // I
            case(Fun3)
            3'b000: ALU_Control = 4'b0010 ;
            3'b010: ALU_Control = 4'b0111 ;
            3'b011: ALU_Control = 4'b1001 ;
            3'b100: ALU_Control = 4'b1100 ;
            3'b110: ALU_Control = 4'b0001 ;
            3'b111: ALU_Control = 4'b0000 ;
            3'b001: ALU_Control = 4'b1110 ;
            3'b101:
                case(Fun7)
                1'b0: ALU_Control = 4'b1101 ;
                1'b1: ALU_Control = 4'b1111 ;
                Endcase
             default:ALU_Control = 4’b1110;
            endcase
        endcase
        CPU_MIO = 0;
    end
endmodule
```

DataPath.v

```
module DataPath(
    input clk,
    input rst,
    input [31:0] inst_field,
    input [31:0] Data_in,
    input [3:0] ALU_operation,
    input [2:0] ImmSel,
    input [1:0] MemtoReg,
    input ALUSrc_B,
    input [1:0] Jump,
    input Branch,
    input BranchN,
    input RegWrite,
    output [31:0] PC_out,
    output [31:0] Data_out,
    output [31:0] ALU_out
    );
    wire [31:0] PC4;
    wire [31:0] NextPC;
    wire [31:0] PCi;
    wire [31:0] imm_out;
```

```verilog
wire [31:0] mux0_out;
wire [31:0] Wt_data;
wire [31:0] PC_in;
wire [31:0] ALU_in;
wire ALU_zero;
wire [31:0] Rs1_data;
assign PC4 = 32'd4;

ImmGen ImmGen(
    .ImmSel(ImmSel),.inst_field(inst_field),
    .Imm_out(imm_out));

ADD32 PC4_add(
    .A(PC_out),.B(PC4),.o(NextPC));

ADD32 PCi_add(
    .A(PC_out),.B(imm_out),.o(PCi));

MUX2T1_32 PC_MUX0(
    .I0(NextPC),.I1(PCi),
    .s((Branch & ALU_zero) | (BranchN & ~ALU_zero)),
    .o(mux0_out));

MUX4T1 PC_MUX1(
    .I0(mux0_out),.I1(PCi),.I2(ALU_out),.I3(mux0_out),
    .s(Jump),.o(PC_in));

MUX4T1 REG_MUX(
        .I0(ALU_out),.I1(Data_in),.I2(NextPC),.I3(imm_out),
        .s(MemtoReg),.o(Wt_data));

MUX2T1_32 ALU_MUX(
    .I0(Data_out),.I1(imm_out),.s(ALUSrc_B),.o(ALU_in));

regs Regs(
    .clk(clk),.rst(rst),
    .Rs1_addr(inst_field[19:15]),.Rs2_addr(inst_field[24:20]),
    .Wt_addr(inst_field[11:7]),.Wt_data(Wt_data),
    .RegWrite(RegWrite),
    .Rs1_data(Rs1_data),.Rs2_data(Data_out));

ALU ALU(
    .A(Rs1_data),.B(ALU_in),.ALU_operation(ALU_operation),
    .res(ALU_out),.zero(ALU_zero));
    REG32 PC(
        .clk(clk),.rst(rst),
        .CE(1'b1),.D(PC_in),.Q(PC_out));
endmodule
```

DataPath 模块中调用子模块的代码如下：

ImmGen.v

```
module ImmGen(
    input wire [2:0] ImmSel,
    input wire [31:0] inst_field,
    output reg [31:0] Imm_out
    );
    always@(*) begin
        case(ImmSel)
            3'b001: Imm_out = {{20{inst_field[31]}},inst_field[31:20]}; // addi\lw(I)
            3'b010: Imm_out =
{{20{inst_field[31]}},inst_field[31:25],inst_field[11:7]};// sw(S)
            3'b011: Imm_out =
{{20{inst_field[31]}},inst_field[7],inst_field[30:25],inst_field[11:8],1'b0};// beq(SB)
            3'b100: Imm_out =
{{12{inst_field[31]}},inst_field[19:12],inst_field[20],inst_field[30:21],1'b0};//
jal(UJ)
            3'b000: Imm_out = {inst_field[31:12], 12'h000};// lui(U)
           Default: Imm_out = {32’h00000000};
        endcase
    end
endmodule
```

regs.v (调用 Lab1)

```
module regs(
    input clk, rst, RegWrite,
    input [4:0] Rs1_addr, Rs2_addr, Wt_addr,
    input [31:0] Wt_data,
    output [31:0] Rs1_data, Rs2_data
    );

    reg [31:0] register [1:31];
    integer i;

    assign Rs1_data = (Rs1_addr == 0) ? 0 : register[Rs1_addr]; // read
    assign Rs2_data = (Rs2_addr == 0) ? 0 : register[Rs2_addr];

    always@(posedge clk or posedge rst)
```

```
        begin if(rst == 1) for (i=1;i<32;i=i+1) register[i] <= 0; //reset
               else if((Wt_addr != 0) && (RegWrite == 1))
                   register[Wt_addr] <= Wt_data; //write
        end
endmodule
```

ALU.v

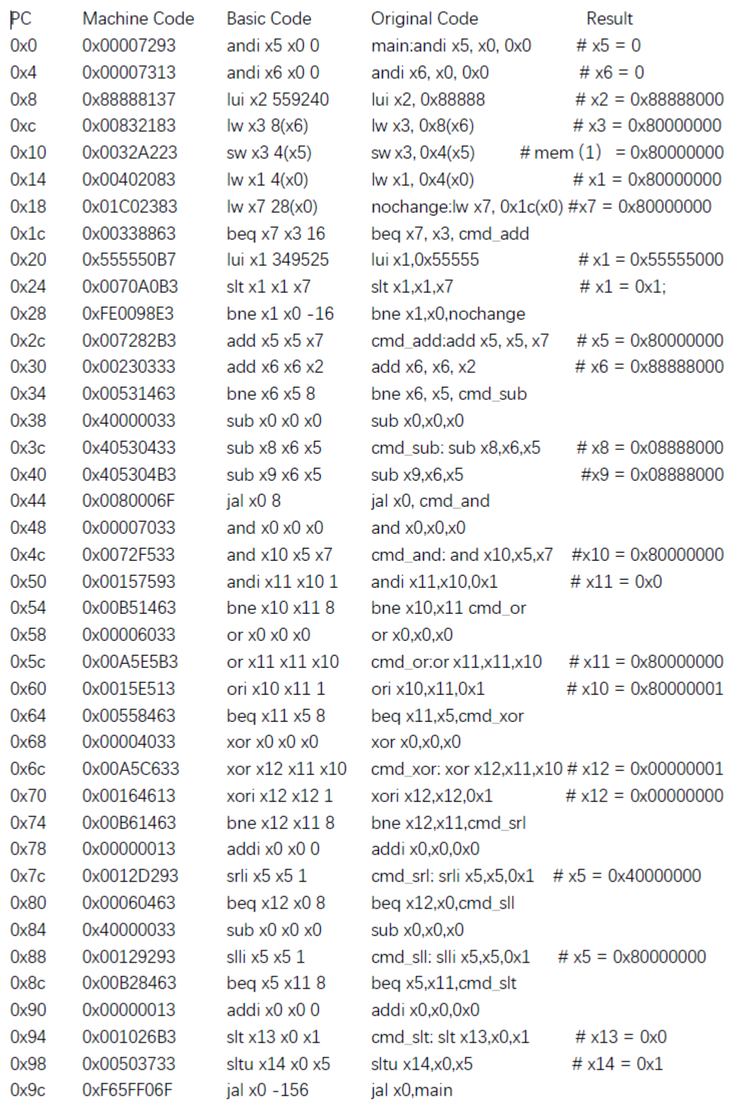
```
module ALU(
    input [31:0] A,
    input [2:0] ALU_operation,
    input [31:0] B,
    output reg [31:0] res,
    output zero
    );
    wire [31:0] res_and, res_or, res_add, res_sub, res_nor, res_slt;
    parameter one = 32'h00000001, zero_0 = 32'h00000000;

    assign res_and = A & B;
    assign res_or = A | B;
    assign res_add = A + B;
    assign res_sub = A - B;
    assign res_slt = (A < B) ? one : zero_0;

    always @(A or B or ALU_operation)
        case(ALU_operation)
            3'b000: res = res_and;
            3'b001: res = res_or;
            3'b010: res = res_add;
            3'b011: res = A ^ B;
            3'b100: res = ~(A | B);
            3'b101: res = B >> 1;
            3'b110: res = res_sub;
            3'b111: res = res_slt;
            default: res = 32'hx;
        endcase
        assign zero = (res == 0) ? 1:0;
endmodule
```

REG32.v

```
module REG32(
    input clk,
    input rst,
    input CE,
    input [31:0] D,
    output reg [31:0] Q
    );
    always @(posedge clk or posedge rst)
        if(rst == 1'b1) Q <= 32'b0;
        else if(CE) Q <= D;
endmodule
```

实验任务二：设计指令集测试方案并完成测试

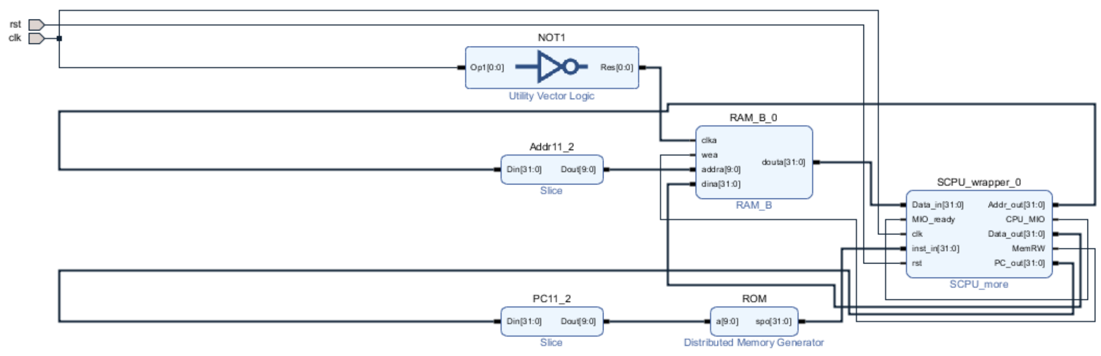
本次实验我使用了助教提供的指令来进行测试，指令代码及意义如下图所示：



PC	Machine Code	Basic Code	Original Code	Result
0x0	0x00007293	andi x5 x0 0	main:andi x5, x0, 0x0	# x5 = 0
0x4	0x00007313	andi x6 x0 0	andi x6, x0, 0x0	# x6 = 0
0x8	0x88888137	lui x2 559240	lui x2, 0x88888	# x2 = 0x88888000
0xc	0x00832183	lw x3 8(x6)	lw x3, 0x8(x6)	# x3 = 0x80000000
0x10	0x0032A223	sw x3 4(x5)	sw x3, 0x4(x5)	# mem (1) = 0x80000000
0x14	0x00402083	lw x1 4(x0)	lw x1, 0x4(x0)	# x1 = 0x80000000
0x18	0x01C02383	lw x7 28(x0)	nochange:lw x7, 0x1c(x0)	#x7 = 0x80000000
0x1c	0x00338863	beq x7 x3 16	beq x7, x3, cmd_add	
0x20	0x555550B7	lui x1 349525	lui x1,0x55555	# x1 = 0x55555000
0x24	0x0070A0B3	slt x1 x1 x7	slt x1,x1,x7	# x1 = 0x1;
0x28	0xFE0098E3	bne x1 x0 -16	bne x1,x0,nochange	
0x2c	0x007282B3	add x5 x5 x7	cmd_add:add x5, x5, x7	# x5 = 0x80000000
0x30	0x00230333	add x6 x6 x2	add x6, x6, x2	# x6 = 0x88888000
0x34	0x00531463	bne x6 x5 8	bne x6, x5, cmd_sub	
0x38	0x40000033	sub x0 x0 x0	sub x0,x0,x0	
0x3c	0x40530433	sub x8 x6 x5	cmd_sub: sub x8,x6,x5	# x8 = 0x08888000
0x40	0x405304B3	sub x9 x6 x5	sub x9,x6,x5	#x9 = 0x08888000
0x44	0x0080006F	jal x0 8	jal x0, cmd_and	
0x48	0x00007033	and x0 x0 x0	and x0,x0,x0	
0x4c	0x0072F533	and x10 x5 x7	cmd_and: and x10,x5,x7	#x10 = 0x80000000
0x50	0x00157593	andi x11 x10 1	andi x11,x10,0x1	# x11 = 0x0
0x54	0x00B51463	bne x10 x11 8	bne x10,x11 cmd_or	
0x58	0x00006033	or x0 x0 x0	or x0,x0,x0	
0x5c	0x00A5E5B3	or x11 x11 x10	cmd_or:or x11,x11,x10	# x11 = 0x80000000
0x60	0x0015E513	ori x10 x11 1	ori x10,x11,0x1	# x10 = 0x80000001
0x64	0x00558463	beq x11 x5 8	beq x11,x5,cmd_xor	
0x68	0x00004033	xor x0 x0 x0	xor x0,x0,x0	
0x6c	0x00A5C633	xor x12 x11 x10	cmd_xor: xor x12,x11,x10	# x12 = 0x00000001
0x70	0x00164613	xori x12 x12 1	xori x12,x12,0x1	# x12 = 0x00000000
0x74	0x00B61463	bne x12 x11 8	bne x12,x11,cmd_srl	
0x78	0x00000013	addi x0 x0 0	addi x0,x0,0x0	
0x7c	0x0012D293	srli x5 x5 1	cmd_srl: srli x5,x5,0x1	# x5 = 0x40000000
0x80	0x00060463	beq x12 x0 8	beq x12,x0,cmd_sll	
0x84	0x40000033	sub x0 x0 x0	sub x0,x0,x0	
0x88	0x00129293	slli x5 x5 1	cmd_sll: slli x5,x5,0x1	# x5 = 0x80000000
0x8c	0x00B28463	beq x5 x11 8	beq x5,x11,cmd_slt	
0x90	0x00000013	addi x0 x0 0	addi x0,x0,0x0	
0x94	0x001026B3	slt x13 x0 x1	cmd_slt: slt x13,x0,x1	# x13 = 0x0
0x98	0x00503733	sltu x14 x0 x5	sltu x14,x0,x5	# x14 = 0x1
0x9c	0xF65FF06F	jal x0 -156	jal x0,main	

线上仿真测试：

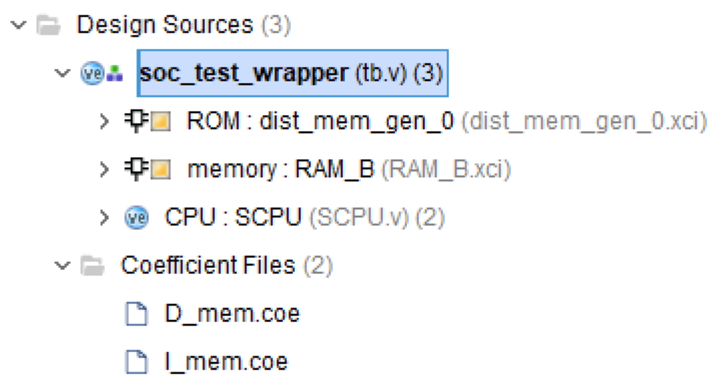
为了进行仿真测试，首先搭建一个线上测试的平台 soc_test_wrapper。
根据下图，调用 SCPU、ROM 和 RAM，编写 verilog 代码。



代码如下：

```
module soc_test_wrapper(
    input rst,
    input clk
    );
    wire [31:0] PC_out;
    wire [31:0] Inst_in;
    wire MemRW;
    wire [31:0] Addr_out;
    wire [31:0] Data_in;
    wire [31:0] Data_out;
    wire CPU_MIO;
    dist_mem_gen_0 ROM(
        .a(PC_out[11:2]),.spo(Inst_in)
        );

    RAM_B memory(
        .clka(~clk),.wea(MemRW),.addra(Addr_out[11:2]),.dina(Data_out),.douta(Data_in)
        );

    SCPU CPU(
        .Data_in(Data_in),.MIO_ready(CPU_MIO),.clk(clk),
        .Inst_in(Inst_in),.rst(rst),
        .Addr_out(Addr_out),.CPU_MIO(CPU_MIO),.Data_out(Data_out),
        .MemRW(MemRW),.PC_out(PC_out)
        );
endmodule
```

整体的框架入下图所示：


Design Sources (3)
soc_test_wrapper (tb.v) (3)
ROM : dist_mem_gen_0 (dist_mem_gen_0.xci)
memory : RAM_B (RAM_B.xci)
CPU : SCPU (SCPU.v) (2)
Coefficient Files (2)
D_mem.coe
I_mem.coe

编写仿真文件后，进行验证（具体结果放在第二部分-实验结果与分析），仿真文件如下：

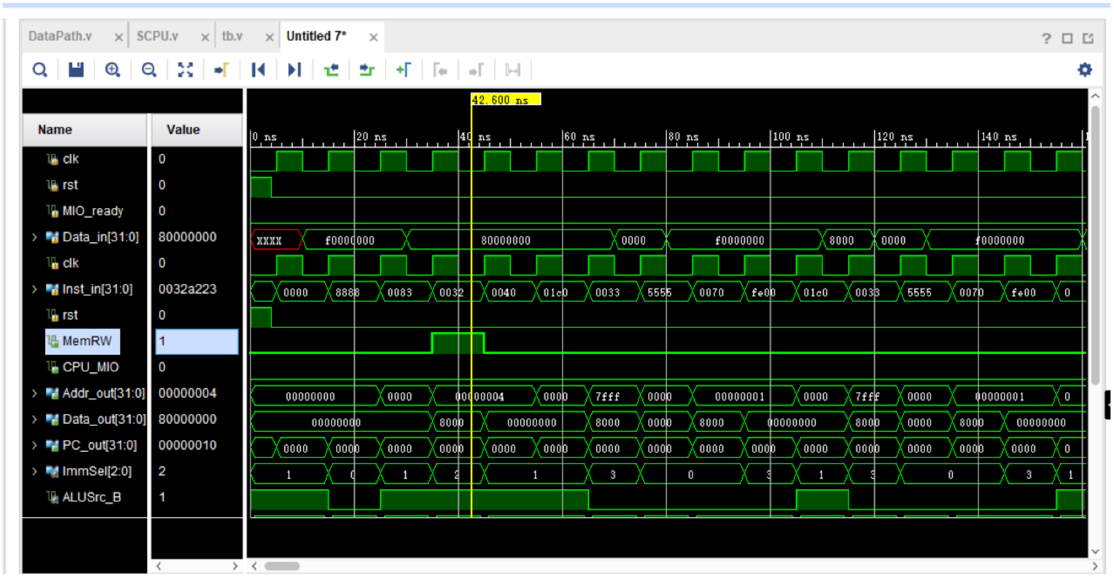
```
module cpu_tb();
    reg clk;
    reg rst;
    soc_test_wrapper u(
        .clk(clk),.rst(rst)
    );
    always #5 clk = ~clk;
    initial begin
        clk = 0;
        rst = 1;
        #4;
        rst = 0;
    end
endmodule
```

线下上板验证：
用本次实验制作的单周期 CPU 核替换实验二搭建 SOC 平台时使用的 CPU 核，生成 bit 文件后进行上班验证。

二、实验结果与分析

线上仿真测试：

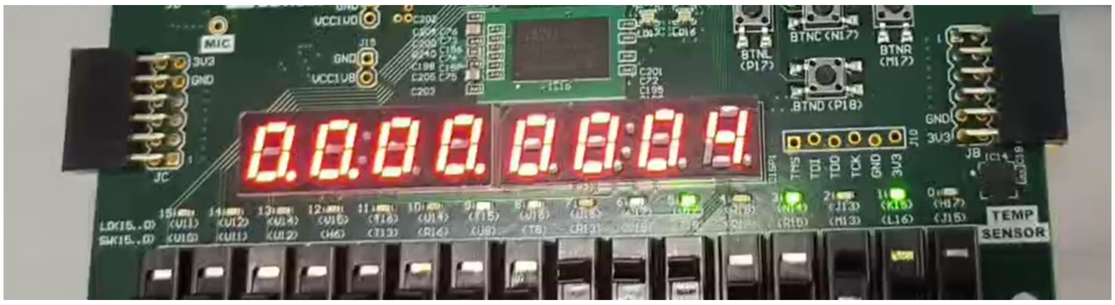
对照测试的指令比较各个指令阶段种种信号的值，发现结果符合预期，说明单周期 CPU 可以正常运行。



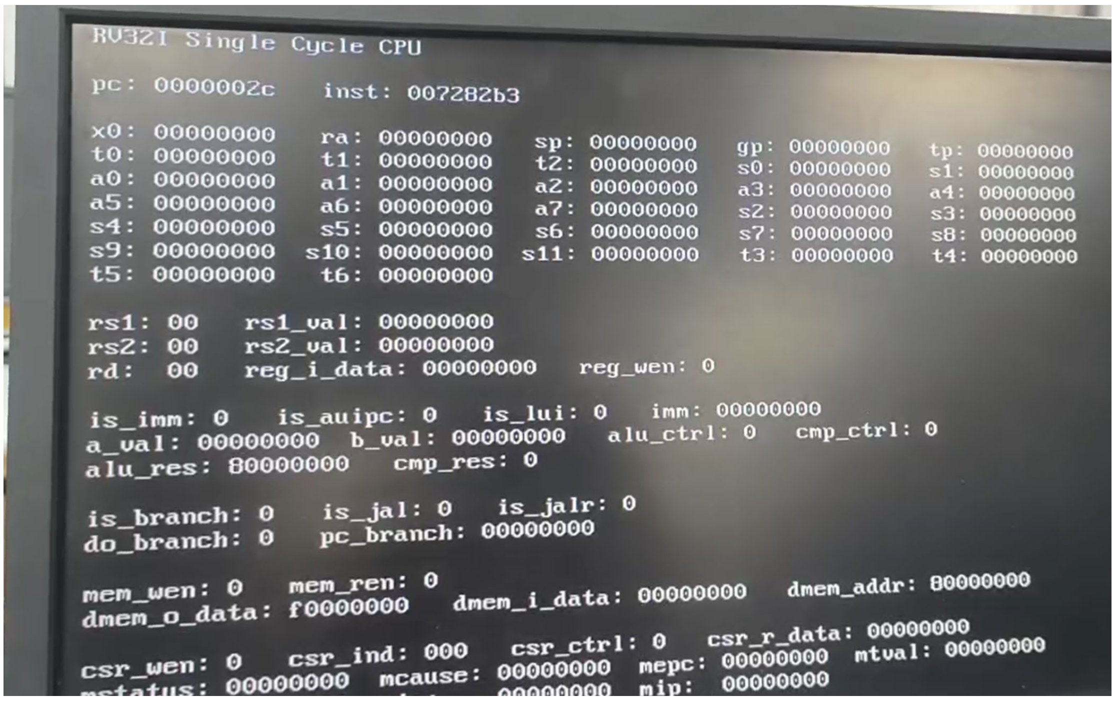
```
Name            Value
clk             0
rst             0
MIO_ready       0
Data_in[31:0]   80000000
clk             0
Inst_in[31:0]   0032a223
rst             0
MemRW           1
CPU_MIO         0
Addr_out[31:0]  00000004
Data_out[31:0]  80000000
PC_out[31:0]    00000010
ImmSel[2:0]     2
ALUSrc_B        1
```

线下上板验证：

上板图片：



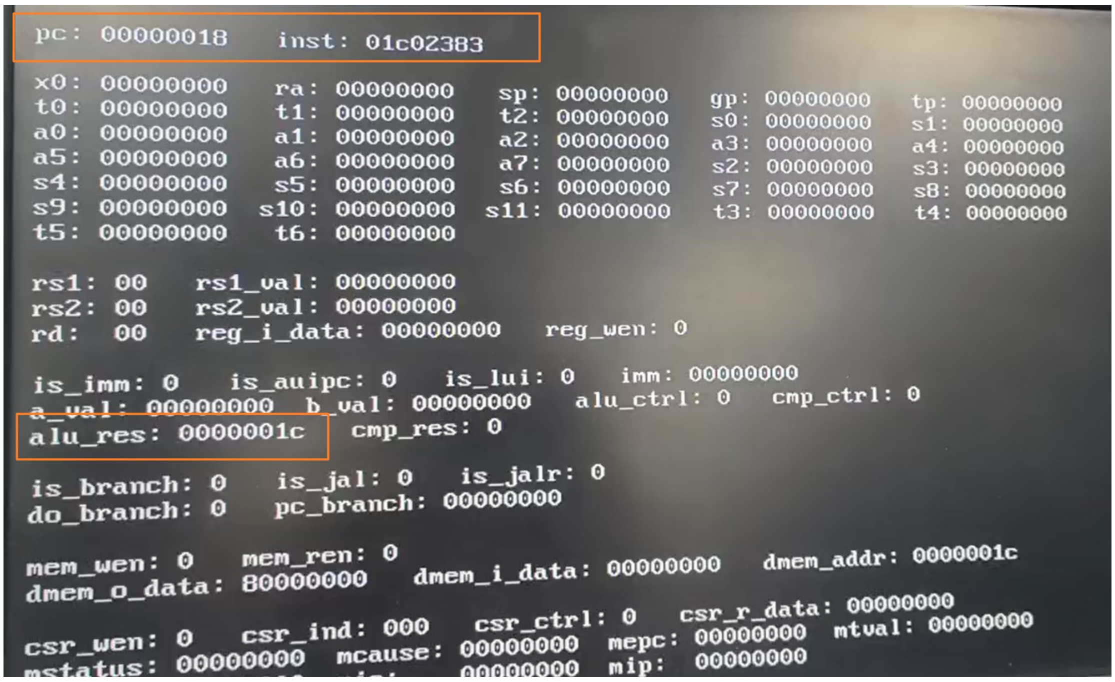
Vga 显示图片：



```
RV32I Single Cycle CPU

pc: 0000002c    inst: 007282b3

x0: 00000000    ra: 00000000    sp: 00000000    gp: 00000000    tp: 00000000
t0: 00000000    t1: 00000000    t2: 00000000    s0: 00000000    s1: 00000000
a0: 00000000    a1: 00000000    a2: 00000000    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000000    s3: 00000000
s4: 00000000    s5: 00000000    s6: 00000000    s7: 00000000    s8: 00000000
s9: 00000000   s10: 00000000   s11: 00000000    t3: 00000000    t4: 00000000
t5: 00000000    t6: 00000000

rs1: 00   rs1_val: 00000000
rs2: 00   rs2_val: 00000000
rd:  00   reg_i_data: 00000000   reg_wen: 0

is_imm: 0   is_auipc: 0   is_lui: 0   imm: 00000000
a_val: 00000000  b_val: 00000000   alu_ctrl: 0   cmp_ctrl: 0
alu_res: 80000000   cmp_res: 0

is_branch: 0   is_jal: 0   is_jalr: 0
do_branch: 0   pc_branch: 00000000

mem_wen: 0   mem_ren: 0
dmem_o_data: f0000000   dmem_i_data: 00000000   dmem_addr: 80000000

csr_wen: 0   csr_ind: 000   csr_ctrl: 0   csr_r_data: 00000000
mstatus: 00000000   mcause: 00000000   mepc: 00000000   mtval: 00000000
                               00000000   mip:  00000000
```

测试验证后发现运行结果完全符合预期，说明该 SCPU 上板验证通过。
下面截取部分过程说明：

```
0x18  0x01C02383  lw x7 28(x0)  lw x7, 0x1c(x0)  #x7=0x80000000
```



```
pc: 00000018    inst: 01c02383
x0: 00000000   ra: 00000000   sp: 00000000   gp: 00000000   tp: 00000000
t0: 00000000   t1: 00000000   t2: 00000000   s0: 00000000   s1: 00000000
a0: 00000000   a1: 00000000   a2: 00000000   a3: 00000000   a4: 00000000
a5: 00000000   a6: 00000000   a7: 00000000   s2: 00000000   s3: 00000000
s4: 00000000   s5: 00000000   s6: 00000000   s7: 00000000   s8: 00000000
s9: 00000000  s10: 00000000  s11: 00000000   t3: 00000000   t4: 00000000
t5: 00000000   t6: 00000000

rs1: 00   rs1_val: 00000000
rs2: 00   rs2_val: 00000000
rd:  00   reg_i_data: 00000000   reg_wen: 0

is_imm: 0   is_auipc: 0   is_lui: 0   imm: 00000000
a_val: 00000000   b_val: 00000000   alu_ctrl: 0   cmp_ctrl: 0
alu_res: 0000001c   cmp_res: 0

is_branch: 0   is_jal: 0   is_jalr: 0
do_branch: 0   pc_branch: 00000000

mem_wen: 0   mem_ren: 0
dmem_o_data: 80000000   dmem_i_data: 00000000   dmem_addr: 0000001c

csr_wen: 0   csr_ind: 000   csr_ctrl: 0   csr_r_data: 00000000
mstatus: 00000000   mcause: 00000000   mepc: 00000000   mtval: 00000000
                                00000000   mip: 00000000
```

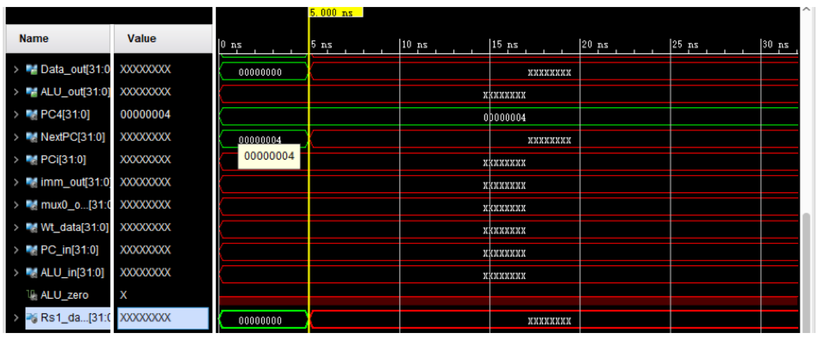
```
0x1c  0x00338863  beq x7 x3 16  beq x7, x3, cmd_add
```

运行发现可以成功跳转（从 1c 跳转到 2c）。

三、讨论、心得

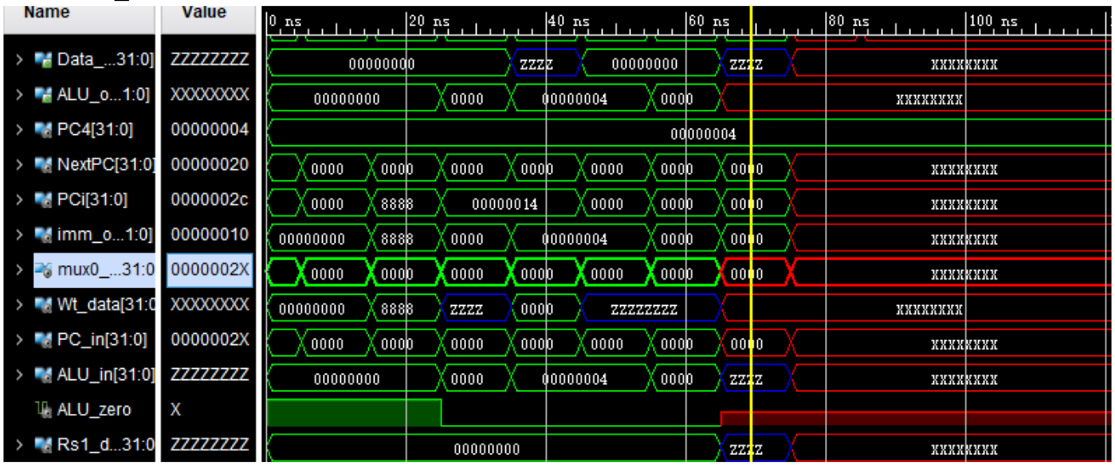
构建 SCPU 花费了相当久的时间，中间也经历了痛苦的 debug 过程，最后能成功写出来，在喜极而泣的同时也感觉非常有成就感！
下面列举几个碰到的问题和解决办法：

1. <u>在搭建仿真测试平台 soc_test_wrapper 的时候，仿真发现很多数据没有输出（显示成红色的叉）。</u>初步推测是连接错误，后续检查是因为在扩展指令集后，ImmSel、Jump 等信号的位数发生了变化，但是修改代码的时候并没有把他们改过来。修改之后问题得到解决。

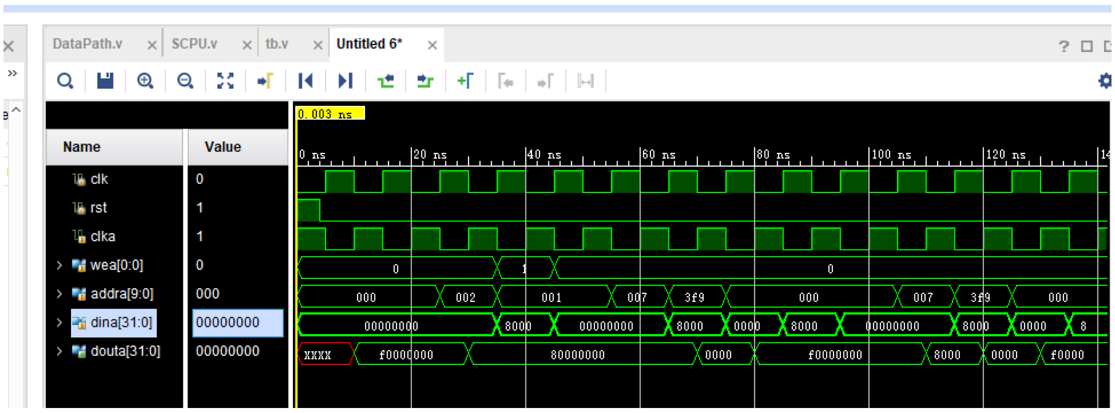


Name	Value
Data_out[31:0	XXXXXXXX
ALU_out[31:0]	XXXXXXXX
PC4[31:0]	00000004
NextPC[31:0]	XXXXXXXX
PCi[31:0]	XXXXXXXX
imm_out[31:0	XXXXXXXX
mux0_o...[31:0	XXXXXXXX
Wt_data[31:0]	XXXXXXXX
PC_in[31:0]	XXXXXXXX
ALU_in[31:0]	XXXXXXXX
ALU_zero	X
Rs1_da...[31:0	XXXXXXXX

2. 初次测试使用的是 lab2 的，含有很多 R 指令的测试文件，仿真没有发现问题，但是修改为 lab4 提供的测试文件时，发现跑了一段之后就出错了，很多输出出现了叉。检查后发现原因是此处的跳转指令执行后 PC 未知，进一步检查发现是因为 ALU_zreo 未知（输入的 A 和 B 出现了问题）。后续发现是因为给的 RAM_B 的 IP 核是空的，需要自己生成并利用 D_mem.coe 进行初始化。



修改后问题得到解决：



3. 上板验证时，发现部分指令无法跳转（比如 0x74 这条无意义指令仍旧执行了）。检查发现是 case 语句组的部分没有编写 default 语句，导致时间上出了差错，修改后可以正常运行跳转语句。

四、个人生活照片