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 6 月 5 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 5：流水线 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 6 月 5 日

一、操作方法与实验步骤

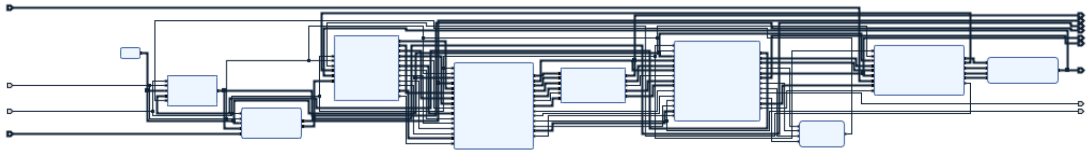
Lab5-1:流水线处理器集成

实验目的：

1.理解流水线 CPU 的基本原理和组织结构
2.掌握五级流水线的工作过程和设计方法
3.理解流水线 CPU 停机的原理
4.设计流水线测试程序

设计过程：

根据流水线集成设计图完成 Pipeline_CPU.v 文件的编写



代码如下所示（部分接口为后续 debug 时加上的，实验中并不做要求）：

Pipeline_CPU.v

```
module Pipeline_CPU (
    input rst,
    input clk,
    input [31:0] Data_in,
    input [31:0] Inst_IF,
    output [31:0] PC_out_IF,
    output [31:0] PC_out_ID,
```

```
    output [31:0] PC_out_EX,
    output [31:0] Inst_ID,
    output [31:0] Addr_out,
    output [31:0] Data_out,
    output [31:0] Data_out_WB,
    output MemRW_Mem,
    output MemRW_EX,

    output [31:0] zero,
    output [31:0] ra,
    output [31:0] sp,
    output [31:0] gp,
    output [31:0] tp,
    output [31:0] t0,
    output [31:0] t1,
    output [31:0] t2,
    output [31:0] s0,
    output [31:0] s1,
    output [31:0] a0,
    output [31:0] a1,
    output [31:0] a2,
    output [31:0] a3,
    output [31:0] a4,
    output [31:0] a5,
    output [31:0] a6,
    output [31:0] a7,
    output [31:0] s2,
    output [31:0] s3,
    output [31:0] s4,
    output [31:0] s5,
    output [31:0] s6,
    output [31:0] s7,
    output [31:0] s8,
    output [31:0] s9,
    output [31:0] s10,
    output [31:0] s11,
    output [31:0] t3,
    output [31:0] t4,
    output [31:0] t5,
    output [31:0] t6,
    output [31:0] Rs1_out_IDEX,
    output [31:0] Rs2_out_IDEX,
    output [31:0] Imm_out_IDEX
    );
```

```
wire [31:0] PC_out_EXMem;
wire PCSrc;
wire RegWrite_out_MemWB;
wire [4:0] Rd_addr_out_MemWB;
wire [31:0] Rd_addr_out_ID;
wire [31:0] Rs1_out_ID;
wire [31:0] Rs2_out_ID;
wire [31:0] Imm_out_ID;
wire ALUSrc_B_ID;
wire [2:0] ALU_control_ID;
wire Branch_ID;
wire BranchN_ID;
wire MemRW_ID;
wire Jump_ID;
wire [1:0] MemtoReg_ID;
wire RegWrite_out_ID;
wire [31:0] PC_out_IDEX;
wire [4:0] Rd_addr_out_IDEX;
wire ALUSrc_B_out_IDEX;
wire [2:0] ALU_control_out_IDEX;
wire Branch_out_IDEX;
wire BranchN_out_IDEX;
wire Jump_out_IDEX;
wire [1:0] MemtoReg_out_IDEX;
wire RegWrite_out_IDEX;
wire [31:0] PC4_out_EX;
wire zero_out_EX;
wire [31:0] ALU_out_EX;
wire [31:0] Rs2_out_EX;
wire [31:0] PC4_out_EXMem;
wire [4:0] Rd_addr_out_EXMem;
wire zero_out_EXMem;
wire Branch_out_EXMem;
wire BranchN_out_EXMem;
wire Jump_out_EXMem;
wire [1:0] MemtoReg_out_EXMem;
wire RegWrite_out_EXMem;
wire [31:0] PC4_out_MemWB;
wire [31:0] ALU_out_MemWB;
wire [31:0] DMem_data_out_MemWB;
wire [1:0] MemtoReg_out_MemWB;

Pipeline_IF IF(
.clk_IF(clk),.rst_IF(rst),.en_IF(1'b1),
```

```
.PC_in_IF(PC_out_EXMem),.PCSrc(PCSrc),
.PC_out_IF(PC_out_IF)
);

IF_reg_ID IF_reg_ID(
.clk_IFID(clk),.rst_IFID(rst),.en_IFID(1'b1),
.PC_in_IFID(PC_out_IF),.inst_in_IFID(Inst_IF),
.PC_out_IFID(PC_out_ID),.inst_out_IFID(Inst_ID)
);

Pipeline_ID ID(
.clk_ID(clk),.rst_ID(rst),
.RegWrite_in_ID(RegWrite_out_MemWB),
.Rd_addr_ID(Rd_addr_out_MemWB),.Wt_data_ID(Data_out_WB),
.Inst_in_ID(Inst_ID),
.Rd_addr_out_ID(Rd_addr_out_ID),
.Rs1_out_ID(Rs1_out_ID),.Rs2_out_ID(Rs2_out_ID),
.Imm_out_ID(Imm_out_ID),
.ALUSrc_B_ID(ALUSrc_B_ID),.ALU_control_ID(ALU_control_ID),
.Branch_ID(Branch_ID),.BranchN_ID(BranchN_ID),
.MemRW_ID(MemRW_ID),.Jump_ID(Jump_ID),
.MemtoReg_ID(MemtoReg_ID),.RegWrite_out_ID(RegWrite_out_ID),
.zero(zero),
.ra(ra),
.sp(sp),
.gp(gp),
.tp(tp),
.t0(t0),
.t1(t1),
.t2(t2),
.s0(s0),
.s1(s1),
.a0(a0),
.a1(a1),
.a2(a2),
.a3(a3),
.a4(a4),
.a5(a5),
.a6(a6),
.a7(a7),
.s2(s2),
.s3(s3),
.s4(s4),
.s5(s5),
```

```
.s6(s6),
.s7(s7),
.s8(s8),
.s9(s9),
.s10(s10),
.s11(s11),
.t3(t3),
.t4(t4),
.t5(t5),
.t6(t6)
);

ID_reg_Ex ID_reg_Ex(
.clk_IDEX(clk),.rst_IDEX(rst),.en_IDEX(1'b1),
.PC_in_IDEX(PC_out_ID),
.Rd_addr_IDEX(Rd_addr_out_ID[4:0]),
.Rs1_in_IDEX(Rs1_out_ID),.Rs2_in_IDEX(Rs2_out_ID),
.Imm_in_IDEX(Imm_out_ID),
.ALUSrc_B_in_IDEX(ALUSrc_B_ID),.ALU_control_in_IDEX(ALU_control_ID),
.Branch_in_IDEX(Branch_ID),.BranchN_in_IDEX(BranchN_ID),
.MemRW_in_IDEX(MemRW_ID),.Jump_in_IDEX(Jump_ID),
.MemtoReg_in_IDEX(MemtoReg_ID),.RegWrite_in_IDEX(RegWrite_out_ID),
.PC_out_IDEX(PC_out_IDEX),
.Rd_addr_out_IDEX(Rd_addr_out_IDEX),
.Rs1_out_IDEX(Rs1_out_IDEX),.Rs2_out_IDEX(Rs2_out_IDEX),
.Imm_out_IDEX(Imm_out_IDEX),
.ALUSrc_B_out_IDEX(ALUSrc_B_out_IDEX),.ALU_control_out_IDEX(ALU_control_out_IDEX),
.Branch_out_IDEX(Branch_out_IDEX),.BranchN_out_IDEX(BranchN_out_IDEX),
.MemRW_out_IDEX(MemRW_EX),
.Jump_out_IDEX(Jump_out_IDEX),
.MemtoReg_out_IDEX(MemtoReg_out_IDEX),
.RegWrite_out_IDEX(RegWrite_out_IDEX)
);

Pipeline_Ex Ex(
.PC_in_EX(PC_out_IDEX),
.Rs1_in_EX(Rs1_out_IDEX),.Rs2_in_EX(Rs2_out_IDEX),
.Imm_in_EX(Imm_out_IDEX),
.ALUSrc_B_in_EX(ALUSrc_B_out_IDEX),
.ALU_control_in_EX(ALU_control_out_IDEX),
.PC_out_EX(PC_out_EX),
.PC4_out_EX(PC4_out_EX),
.zero_out_EX(zero_out_EX),
.ALU_out_EX(ALU_out_EX),
```

```
.Rs2_out_EX(Rs2_out_EX)
);

Ex_reg_Mem Ex_reg_Mem(
.clk_EXMem(clk),.rst_EXMem(rst),.en_EXMem(1'b1),
.PC_in_EXMem(PC_out_EX),.PC4_in_EXMem(PC4_out_EX),
.Rd_addr_EXMem(Rd_addr_out_IDEX),
.zero_in_EXMem(zero_out_EX),
.ALU_in_EXMem(ALU_out_EX),
.Rs2_in_EXMem(Rs2_out_EX),
.Branch_in_EXMem(Branch_out_IDEX),
.BranchN_in_EXMem(BranchN_out_IDEX),
.MemRW_in_EXMem(MemRW_EX),
.Jump_in_EXMem(Jump_out_IDEX),
.MemtoReg_in_EXMem(MemtoReg_out_IDEX),
.RegWrite_in_EXMem(RegWrite_out_IDEX),
.PC_out_EXMem(PC_out_EXMem),
.PC4_out_EXMem(PC4_out_EXMem),
.Rd_addr_out_EXMem(Rd_addr_out_EXMem),
.zero_out_EXMem(zero_out_EXMem),
.ALU_out_EXMem(Addr_out),
.Rs2_out_EXMem(Data_out),
.Branch_out_EXMem(Branch_out_EXMem),
.BranchN_out_EXMem(BranchN_out_EXMem),
.MemRW_out_EXMem(MemRW_Mem),
.Jump_out_EXMem(Jump_out_EXMem),
.MemtoReg_out_EXMem(MemtoReg_out_EXMem),
.RegWrite_out_EXMem(RegWrite_out_EXMem)
);

Pipeline_Mem Mem(
.zero_in_Mem(zero_out_EXMem),
.Branch_in_Mem(Branch_out_EXMem),
.BranchN_in_Mem(BranchN_out_EXMem),
.Jump_in_Mem(Jump_out_EXMem),
.PCSrc(PCSrc)
);

Mem_reg_WB Mem_reg_WB(
.clk_MemWB(clk),.rst_MemWB(rst),.en_MemWB(1'b1),
.PC4_in_MemWB(PC4_out_EXMem),
.Rd_addr_MemWB(Rd_addr_out_EXMem),
.ALU_in_MemWB(Addr_out),
.DMem_data_MemWB(Data_in),
```

```
    .MemtoReg_in_MemWB(MemtoReg_out_EXMem),
    .RegWrite_in_MemWB(RegWrite_out_EXMem),
    .PC4_out_MemWB(PC4_out_MemWB),
    .Rd_addr_out_MemWB(Rd_addr_out_MemWB),
    .ALU_out_MemWB(ALU_out_MemWB),
    .DMem_data_out_MemWB(DMem_data_out_MemWB),
    .MemtoReg_out_MemWB(MemtoReg_out_MemWB),
    .RegWrite_out_MemWB(RegWrite_out_MemWB)
    );

    Pipeline_WB WB(
    .PC4_in_WB(PC4_out_MemWB),
    .ALU_in_WB(ALU_out_MemWB),
    .DMem_data_WB(DMem_data_out_MemWB),
    .MemtoReg_in_WB(MemtoReg_out_MemWB),
    .Data_out_WB(Data_out_WB)
    );
endmodule
```

后续将提供的模块（如 Pipeline_IF）导入（需要导入.v 和.edf 文件以实现模块功能），形成流水线集成 CPU 核，替换至 Lab2 所建立的 SOC 工程中并修改顶层文件中部分接口。修改后的顶层文件代码如下：

CSSTE.v

```
module CSSTE(
    input clk_100mhz,
    input RSTN,
    input [3:0] BTN_y,
    input [15:0] SW,
    output [3:0] Blue,
    output [3:0] Green,
    output [3:0] Red,
    output HSYNC,
    output VSYNC,
    output [15:0] LED_out,
    output [7:0] AN,
    output [7:0] segment
    );
    wire [31:0] clkdiv;
    wire [31:0] PC_out;
    wire [15:0] SW_OK;
    wire [31:0] Addr_out;
    wire [31:0] Data_in;
    wire [31:0] Data_out;
    wire Clk_CPU;
```

```
    wire [31:0] Inst_in;
    wire rst;
    wire [9:0] ram_addr;
    wire [31:0] ram_data_in;
    wire [31:0] RAM_B_0_douta;
    wire U4_data_ram_we;
    wire [3:0] BTN_OK;
    wire [31:0] counter_out;
    wire counter0_out;
    wire counter1_out;
    wire counter2_out;
    wire U7_EN;
    wire U5_EN;
    wire counter_we;
    wire [31:0] Peripheral_in;
    wire [7:0] point;
    wire [7:0] les;
    wire [31:0] disp_num;
    wire [1:0] counter_set;
    wire [31:0] PC_out_ID;
    wire [31:0] inst_ID;
    wire [31:0] PC_out_EX;
    wire MemRW_EX;
    wire MemRW_Mem;
    wire [31:0] Data_out_WB;

    wire [31:0] zero;
    wire [31:0] ra;
    wire [31:0] sp;
    wire [31:0] gp;
    wire [31:0] tp;
    wire [31:0] t0;
    wire [31:0] t1;
    wire [31:0] t2;
    wire [31:0] s0;
    wire [31:0] s1;
    wire [31:0] a0;
    wire [31:0] a1;
    wire [31:0] a2;
    wire [31:0] a3;
    wire [31:0] a4;
    wire [31:0] a5;
    wire [31:0] a6;
    wire [31:0] a7;
```

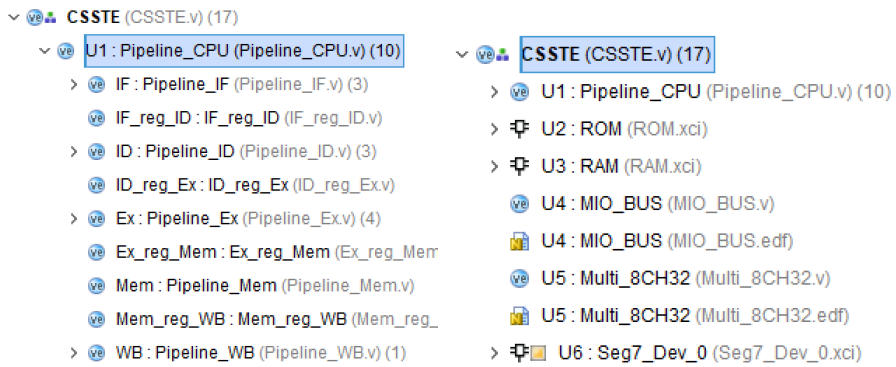
```
    wire [31:0] s2;
    wire [31:0] s3;
    wire [31:0] s4;
    wire [31:0] s5;
    wire [31:0] s6;
    wire [31:0] s7;
    wire [31:0] s8;
    wire [31:0] s9;
    wire [31:0] s10;
    wire [31:0] s11;
    wire [31:0] t3;
    wire [31:0] t4;
    wire [31:0] t5;
    wire [31:0] t6;
    wire [31:0] Rs1_out_IDEX;
    wire [31:0] Rs2_out_IDEX;
    wire [31:0] Imm_out_IDEX;

    Pipeline_CPU U1
        (.clk(Clk_CPU),.rst(rst),
         .Inst_IF(Inst_in),
         .Data_in(Data_in),
         .PC_out_IF(PC_out),
         .PC_out_ID(PC_out_ID),
         .Inst_ID(inst_ID),
         .PC_out_EX(PC_out_EX),
         .MemRW_EX(MemRW_EX),
         .MemRW_Mem(MemRW_Mem),
         .Data_out(Data_out),
         .Addr_out(Addr_out),
         .Data_out_WB(Data_out_WB),
        .zero(zero),
        .ra(ra),
        .sp(sp),
        .gp(gp),
        .tp(tp),
        .t0(t0),
        .t1(t1),
        .t2(t2),
        .s0(s0),
        .s1(s1),
        .a0(a0),
        .a1(a1),
        .a2(a2),
```

```
        .a3(a3),
        .a4(a4),
        .a5(a5),
        .a6(a6),
        .a7(a7),
        .s2(s2),
        .s3(s3),
        .s4(s4),
        .s5(s5),
        .s6(s6),
        .s7(s7),
        .s8(s8),
        .s9(s9),
        .s10(s10),
        .s11(s11),
        .t3(t3),
        .t4(t4),
        .t5(t5),
        .t6(t6),
        .Rs1_out_IDEX(Rs1_out_IDEX),
        .Rs2_out_IDEX(Rs2_out_IDEX),
        .Imm_out_IDEX(Imm_out_IDEX)
        );

    ROM U2
        (.a(PC_out[11:2]),.spo(Inst_in));

    RAM U3
        (.addra(ram_addr),.clka(~clk_100mhz),.dina(ram_data_in),
         .douta(RAM_B_0_douta),.wea(U4_data_ram_we));

    MIO_BUS U4
        (.clk(clk_100mhz),.rst(rst),.BTN(BTN_OK),.SW(SW_OK),
         .mem_w(MemRW_Mem),.Cpu_data2bus(Data_out),.addr_bus(Addr_out),
         .ram_data_out(RAM_B_0_douta),.led_out(LED_out),.counter_out(counter_out),
         .counter0_out(counter0_out),.counter1_out(counter1_out),.counter2_out(counter2_o
ut),
         .Cpu_data4bus(Data_in),.ram_data_in(ram_data_in),.ram_addr(ram_addr),
         .data_ram_we(U4_data_ram_we),.GPIOf0000000_we(U7_EN),.GPIOe0000000_we(U5_EN),
         .counter_we(counter_we),.Peripheral_in(Peripheral_in));

    Multi_8CH32 U5
        (.clk(~Clk_CPU),.rst(rst),.EN(U5_EN),.Test(SW_OK[7:5]),
```

```
        .point_in({clkdiv,clkdiv}),.LES(64'b0),.Data0(Peripheral_in),.data1({2'b0,PC_out
[31:2]}),
        .data2(Inst_in),.data3(counter_out),.data4(Addr_out),.data5(Data_out),
        .data6(Data_in),.data7(PC_out),
        .point_out(point),.LE_out(les),.Disp_num(disp_num));

    Seg7_Dev_0 U6
        (.disp_num(disp_num),.point(point),.les(les),.scan(clkdiv[18:16]),
        .AN(AN),.segment(segment));

    SPIO U7
        (.clk(~Clk_CPU),.rst(rst),.Start(clkdiv[20]),.EN(U7_EN),.P_Data(Peripheral_in),
        .counter_set(counter_set),.LED_out(LED_out));

    clk_div U8
        (.clk(clk_100mhz),.rst(rst),.SW2(SW_OK[2]),.SW8(SW_OK[8]),.STEP(SW_OK[10]),
        .clkdiv(clkdiv),.Clk_CPU(Clk_CPU));

    SAnti_jitter U9
        (.clk(clk_100mhz),.RSTN(RSTN),.readn(1'b0),.Key_y(BTN_y),.SW(SW),
        .rst(rst),.BTN_OK(BTN_OK),.SW_OK(SW_OK));

    Counter_x U10
        (.clk(~Clk_CPU),.rst(rst),.clk0(clkdiv[6]),.clk1(clkdiv[9]),.clk2(clkdiv[11]),
        .counter_we(counter_we),.counter_val(Peripheral_in),.counter_ch(counter_set),
        .counter0_OUT(counter0_out),.counter1_OUT(counter1_out),.counter2_OUT(counter2_o
ut),
        .counter_out(counter_out));

    VGA U11
        (.clk_25m(clkdiv[1]),.clk_100m(clk_100mhz),.rst(rst),
        .PC_IF(PC_out),.inst_IF(Inst_in),
        .PC_ID(PC_out_ID),.inst_ID(inst_ID),
        .PC_Ex(PC_out_EX),
        .MemRW_Ex(MemRW_EX),
        .MemRW_Mem(MemRW_Mem),
        .Data_out(Data_out),
        .Addr_out(Addr_out),
        .Data_out_WB(Data_out_WB),
        .hs(HSYNC),.vs(VSYNC),.vga_r(Red),.vga_g(Green),.vga_b(Blue),
       .zero(zero),
       .ra(ra),
       .sp(sp),
       .gp(gp),
```

```
        .tp(tp),
        .t0(t0),
        .t1(t1),
        .t2(t2),
        .s0(s0),
        .s1(s1),
        .a0(a0),
        .a1(a1),
        .a2(a2),
        .a3(a3),
        .a4(a4),
        .a5(a5),
        .a6(a6),
        .a7(a7),
        .s2(s2),
        .s3(s3),
        .s4(s4),
        .s5(s5),
        .s6(s6),
        .s7(s7),
        .s8(s8),
        .s9(s9),
        .s10(s10),
        .s11(s11),
        .t3(t3),
        .t4(t4),
        .t5(t5),
        .t6(t6),
        .Rs1_out_IDEX(Rs1_out_IDEX),
        .Rs2_out_IDEX(Rs2_out_IDEX),
        .Imm_out_IDEX(Imm_out_IDEX)
        );
endmodule
```

集成完成之后，工程层次如下图所示：



后续生成 bit 文件，上板验证集成环境是否建立成功。

Lab5-2:流水线 CPU 实现（不含 hazard）

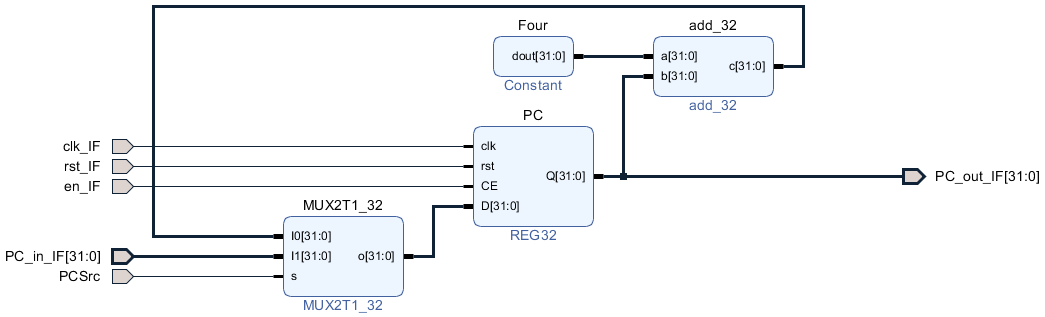
实验目的：

1.理解流水线 CPU 的基本原理和组织结构
2.掌握五级流水线的工作过程和设计方法
3.理解流水线的设计原理
4.设计流水线测试程序

设计过程：

根据相应的接口和该模块的功能，编写 verilog 代码实现

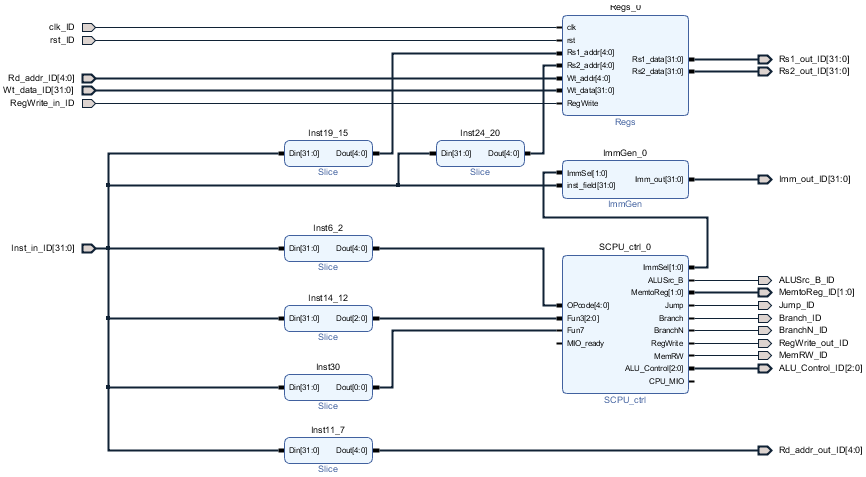
Pipeline_IF.v：




```
module Pipeline_IF(
    input clk_IF,
    input rst_IF,
    input en_IF,
    input [31:0] PC_in_IF,
    input PCSrc,
    output [31:0] PC_out_IF
    );
    wire [31:0] mux_out;
    wire [31:0] add_out;

    MUX2T1_32 MUX(
    .I0(add_out),
    .I1(PC_in_IF),
    .s(PCSrc),
    .o(mux_out)
    );

    REG32 PC(
    .clk(clk_IF),
    .rst(rst_IF),
    .CE(en_IF),
```

```
    .D(mux_out),
    .Q(PC_out_IF)
    );

    ADD32 ADD(
    .a(32'h00000004),
    .b(PC_out_IF),
    .o(add_out)
    );
Endmodule
```

Pipeline_ID.v:



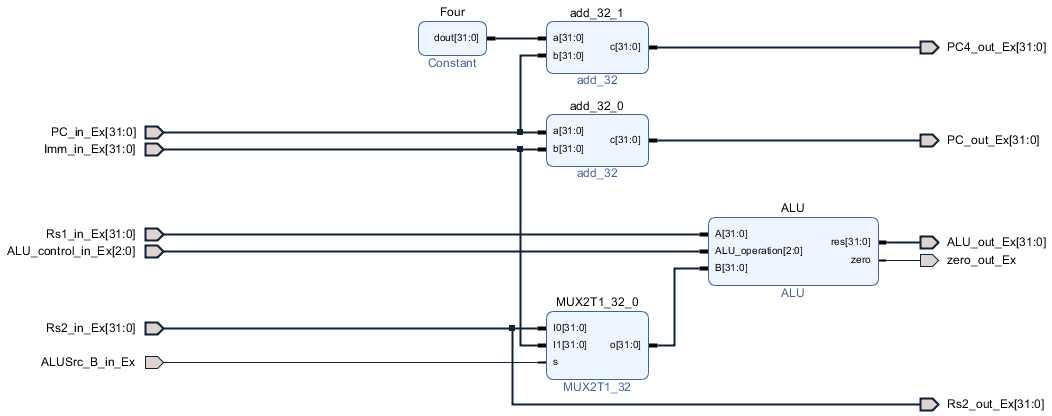

```
module Pipeline_ID(
    input clk_ID,
    input rst_ID,
    input RegWrite_in_ID,
    input [4:0] Rd_addr_ID,
    input [31:0] Wt_data_ID,
    input [31:0] Inst_in_ID,
    output [31:0] Rd_addr_out_ID,
    output [31:0] Rs1_out_ID,
    output [31:0] Rs2_out_ID,
    output [31:0] Imm_out_ID,
    output ALUSrc_B_ID,
    output [2:0] ALU_control_ID,
    output Branch_ID,
    output BranchN_ID,
    output MemRW_ID,
    output Jump_ID,
    output [1:0] MemtoReg_ID,
    output RegWrite_out_ID,
```

```
    output [31:0] zero,
    output [31:0] ra,
    output [31:0] sp,
    output [31:0] gp,
    output [31:0] tp,
    output [31:0] t0,
    output [31:0] t1,
    output [31:0] t2,
    output [31:0] s0,
    output [31:0] s1,
    output [31:0] a0,
    output [31:0] a1,
    output [31:0] a2,
    output [31:0] a3,
    output [31:0] a4,
    output [31:0] a5,
    output [31:0] a6,
    output [31:0] a7,
    output [31:0] s2,
    output [31:0] s3,
    output [31:0] s4,
    output [31:0] s5,
    output [31:0] s6,
    output [31:0] s7,
    output [31:0] s8,
    output [31:0] s9,
    output [31:0] s10,
    output [31:0] s11,
    output [31:0] t3,
    output [31:0] t4,
    output [31:0] t5,
    output [31:0] t6
    );

    wire [2:0] ImmSel; //*
    assign Rd_addr_out_ID[4:0] = Inst_in_ID[11:7];

    regs regs(
    .clk(clk_ID),
    .rst(rst_ID),
    .Rs1_addr(Inst_in_ID[19:15]),
    .Rs2_addr(Inst_in_ID[24:20]),
    .Wt_addr(Rd_addr_ID),
    .Wt_data(Wt_data_ID),
```

```
.RegWrite(RegWrite_in_ID),
.Rs1_data(Rs1_out_ID),
.Rs2_data(Rs2_out_ID),
.zero(zero),
.ra(ra),
.sp(sp),
.gp(gp),
.tp(tp),
.t0(t0),
.t1(t1),
.t2(t2),
.s0(s0),
.s1(s1),
.a0(a0),
.a1(a1),
.a2(a2),
.a3(a3),
.a4(a4),
.a5(a5),
.a6(a6),
.a7(a7),
.s2(s2),
.s3(s3),
.s4(s4),
.s5(s5),
.s6(s6),
.s7(s7),
.s8(s8),
.s9(s9),
.s10(s10),
.s11(s11),
.t3(t3),
.t4(t4),
.t5(t5),
.t6(t6)
);

ImmGen ImmGen(
.ImmSel(ImmSel),
.inst_field(Inst_in_ID),
.Imm_out(Imm_out_ID)
);

SCPU_ctrl SCPU_ctrl(
```

```
    .OPcode(Inst_in_ID[6:2]),
    .Fun3(Inst_in_ID[14:12]),
    .Fun7(Inst_in_ID[30]),
    .ImmSel(ImmSel),
    .ALUSrc_B(ALUSrc_B_ID),
    .MemtoReg(MemtoReg_ID),
    .Jump(Jump_ID),
    .Branch(Branch_ID),
    .BranchN(BranchN_ID),
    .RegWrite(RegWrite_out_ID),
    .MemRW(MemRW_ID),
    .ALU_Control(ALU_control_ID)
    );
endmodule
```

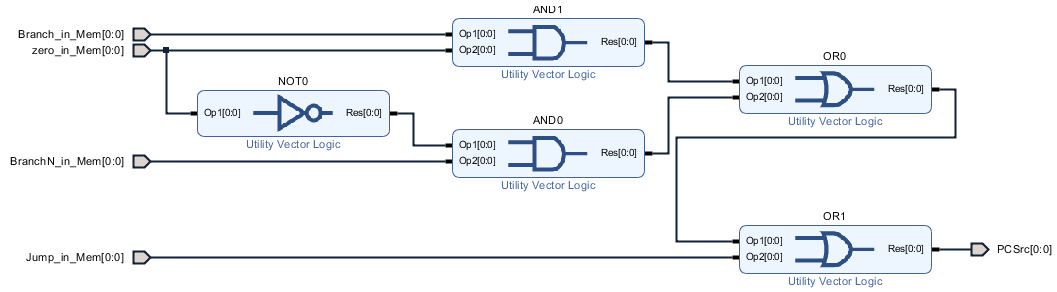
Pipeline_Ex.v:




```
module Pipeline_Ex(
    input[31:0] PC_in_EX,
    input[31:0] Rs1_in_EX,
    input[31:0] Rs2_in_EX,
    input[31:0] Imm_in_EX,
    input ALUSrc_B_in_EX,
    input[2:0] ALU_control_in_EX,
    output wire [31:0] PC_out_EX,
    output wire [31:0] PC4_out_EX,
    output wire zero_out_EX,
    output wire [31:0] ALU_out_EX,
    output wire [31:0] Rs2_out_EX
    );

    wire [31:0]mux_out;

    assign Rs2_out_EX = Rs2_in_EX;
```

```
    ADD32 ADD1(
    .a(32'h00000004),
    .b(PC_in_EX),
    .o(PC4_out_EX)
    );

    ADD32 ADD2(
    .a(PC_in_EX),
    .b(Imm_in_EX),
    .o(PC_out_EX)
    );

    MUX2T1_32 MUX(
    .I0(Rs2_in_EX),
    .I1(Imm_in_EX),
    .s(ALUSrc_B_in_EX),
    .o(mux_out)
    );

    ALU ALU(
    .A(Rs1_in_EX),
    .B(mux_out),
    .ALU_operation(ALU_control_in_EX),
    .res(ALU_out_EX),
    .zero(zero_out_EX)
    );
endmodule
```

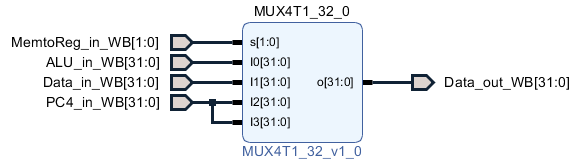
Pipeline_Mem.v:




```
module Pipeline_Mem(
    input zero_in_Mem,
    input Branch_in_Mem,
    input BranchN_in_Mem,
    input Jump_in_Mem,
    output PCSrc
```

```
    );

    assign PCSrc = Jump_in_Mem | (Branch_in_Mem & zero_in_Mem) | ((~zero_in_Mem) &
BranchN_in_Mem);

endmodule
```

Pipeline_WB.v:




```
module Pipeline_WB(
    input [31:0] PC4_in_WB,
    input [31:0] ALU_in_WB,
    input [31:0] DMem_data_WB,
    input [1:0] MemtoReg_in_WB,
    output [31:0] Data_out_WB
    );

    MUX4T1_32 MUX(
    .s(MemtoReg_in_WB),
    .I0(ALU_in_WB),
    .I1(DMem_data_WB),
    .I2(PC4_in_WB),
    .I3(PC4_in_WB),
    .o(Data_out_WB)
    );

endmodule
```

IF_reg_ID.v:

```
module IF_reg_ID(
    input clk_IFID,
    input rst_IFID,
    input en_IFID,
    input [31:0] PC_in_IFID,
    input [31:0] inst_in_IFID,
    output reg [31:0] PC_out_IFID,
    output reg [31:0] inst_out_IFID
    );
```

```
    always @(posedge clk_IFID or posedge rst_IFID)
                    if (rst_IFID == 1'b1) begin
            PC_out_IFID <= 32'h00000000;
            inst_out_IFID <= 32'h00000000;
        end
                    else if(en_IFID == 1'b0) begin
            PC_out_IFID <= PC_out_IFID;
            inst_out_IFID <= inst_out_IFID;
        end
        else begin
            PC_out_IFID <= PC_in_IFID;
            inst_out_IFID <= inst_in_IFID;
        end
endmodule
```

ID_reg_Ex.v:

```
module ID_reg_Ex(
        input clk_IDEX,
        input rst_IDEX,
        input en_IDEX,
        input [31:0] PC_in_IDEX,
        input [4:0] Rd_addr_IDEX,
        input [31:0] Rs1_in_IDEX,
        input [31:0] Rs2_in_IDEX,
        input [31:0] Imm_in_IDEX ,
        input ALUSrc_B_in_IDEX ,
        input [2:0] ALU_control_in_IDEX,
        input Branch_in_IDEX,
        input BranchN_in_IDEX,
        input MemRW_in_IDEX,
        input Jump_in_IDEX, //*
        input [1:0] MemtoReg_in_IDEX,
        input RegWrite_in_IDEX,
                    output reg [31:0] PC_out_IDEX,
        output reg [4:0] Rd_addr_out_IDEX,
        output reg [31:0] Rs1_out_IDEX,
        output reg [31:0] Rs2_out_IDEX,
        output reg [31:0] Imm_out_IDEX ,
        output reg ALUSrc_B_out_IDEX ,
        output reg [2:0] ALU_control_out_IDEX,
        output reg Branch_out_IDEX,
        output reg BranchN_out_IDEX,
        output reg MemRW_out_IDEX,
        output reg Jump_out_IDEX, //*
```

```
        output reg [1:0] MemtoReg_out_IDEX,
        output reg RegWrite_out_IDEX
    );

    always @(posedge clk_IDEX or posedge rst_IDEX) begin
          if (rst_IDEX == 1'b1)  begin
                    PC_out_IDEX <= 32'b0;
                    Rs1_out_IDEX <= 32'b0;
                    Rs2_out_IDEX <= 32'b0;
                    Imm_out_IDEX <= 32'b0;
                    Rd_addr_out_IDEX <= 5'b0;
                    ALUSrc_B_out_IDEX <= 1'b0;
                  ALU_control_out_IDEX <= 3'b0;
                  Branch_out_IDEX <= 1'b0;
                    BranchN_out_IDEX <= 1'b0;
                    MemRW_out_IDEX <= 1'b0;
                    Jump_out_IDEX <= 1'b0;
                  MemtoReg_out_IDEX <= 2'b0;
                  RegWrite_out_IDEX <= 1'b0;
        end
        else if (en_IDEX) begin
                    PC_out_IDEX <= PC_in_IDEX;
                    Rs1_out_IDEX <= Rs1_in_IDEX;
                  Rs2_out_IDEX <= Rs2_in_IDEX;
                    Imm_out_IDEX <= Imm_in_IDEX;
                    Rd_addr_out_IDEX <= Rd_addr_IDEX;
                    ALUSrc_B_out_IDEX <= ALUSrc_B_in_IDEX;
                    ALU_control_out_IDEX <= ALU_control_in_IDEX;
                    Branch_out_IDEX <= Branch_in_IDEX;
                    BranchN_out_IDEX <= BranchN_in_IDEX;
                    MemRW_out_IDEX <= MemRW_in_IDEX;
                  Jump_out_IDEX <= Jump_in_IDEX;
                    MemtoReg_out_IDEX <= MemtoReg_in_IDEX;
                    RegWrite_out_IDEX <= RegWrite_in_IDEX;
        end
    end
endmodule
```

Ex_reg_Mem.v:

```
module Ex_reg_Mem(
        input clk_EXMem,
        input rst_EXMem,
        input en_EXMem,
        input [31:0] PC_in_EXMem,
```

```
        input [31:0] PC4_in_EXMem,
        input [4:0] Rd_addr_EXMem,
        input zero_in_EXMem,
        input [31:0] ALU_in_EXMem,
        input [31:0] Rs2_in_EXMem,
        input Branch_in_EXMem,
        input BranchN_in_EXMem,
        input MemRW_in_EXMem,
        input Jump_in_EXMem, //*
        input [1:0] MemtoReg_in_EXMem,
        input RegWrite_in_EXMem,
        output reg [31:0] PC_out_EXMem,
        output reg [31:0] PC4_out_EXMem,
        output reg [4:0] Rd_addr_out_EXMem,
        output reg zero_out_EXMem,
        output reg [31:0] ALU_out_EXMem,
        output reg [31:0] Rs2_out_EXMem,
        output reg Branch_out_EXMem,
        output reg BranchN_out_EXMem,
        output reg MemRW_out_EXMem,
        output reg Jump_out_EXMem,  //*
        output reg [1:0] MemtoReg_out_EXMem,
        output reg RegWrite_out_EXMem
    );

always @(posedge clk_EXMem or posedge rst_EXMem)
          if (rst_EXMem == 1'b1) begin
      PC_out_EXMem <= 32'b0;
      PC4_out_EXMem <= 32'b0;
      Rd_addr_out_EXMem <= 5'b0;
      zero_out_EXMem <= 1'b0;
      ALU_out_EXMem <= 32'b0;
      Rs2_out_EXMem <= 32'b0;
      Branch_out_EXMem <= 1'b0;
      BranchN_out_EXMem <= 1'b0;
      MemRW_out_EXMem <= 1'b0;
      Jump_out_EXMem <= 1'b0;
      MemtoReg_out_EXMem <= 2'b0;
      RegWrite_out_EXMem <= 1'b0;
          end
          else if (en_EXMem == 1'b1) begin
      PC_out_EXMem <= PC_in_EXMem;
      PC4_out_EXMem <= PC4_in_EXMem;
      Rd_addr_out_EXMem <= Rd_addr_EXMem;
```

```
      zero_out_EXMem <= zero_in_EXMem;
      ALU_out_EXMem <= ALU_in_EXMem;
      Rs2_out_EXMem <= Rs2_in_EXMem;
      Branch_out_EXMem <= Branch_in_EXMem;
      BranchN_out_EXMem <= BranchN_in_EXMem;
      MemRW_out_EXMem <= MemRW_in_EXMem;
      Jump_out_EXMem <= Jump_in_EXMem;
      MemtoReg_out_EXMem <= MemtoReg_in_EXMem;
      RegWrite_out_EXMem <= RegWrite_in_EXMem;
          end
endmodule
```

Mem_reg_WB.v:

```
module Mem_reg_WB(
    input clk_MemWB,
    input rst_MemWB,
    input en_MemWB,
    input [31:0] PC4_in_MemWB,
    input [4:0]  Rd_addr_MemWB,
    input [31:0] ALU_in_MemWB,
    input [31:0] DMem_data_MemWB,
    input [1:0] MemtoReg_in_MemWB,
    input RegWrite_in_MemWB,
    output reg [31:0] PC4_out_MemWB,
    output reg [4:0] Rd_addr_out_MemWB,
    output reg [31:0] ALU_out_MemWB,
    output reg [31:0] DMem_data_out_MemWB,
    output reg [1:0] MemtoReg_out_MemWB,
    output reg RegWrite_out_MemWB
);

    always @(posedge clk_MemWB or posedge rst_MemWB)
        if (rst_MemWB == 1'b1) begin
                    PC4_out_MemWB <= 32'b0;
                    Rd_addr_out_MemWB <= 5'b0;
                  ALU_out_MemWB <= 32'b0;
                    DMem_data_out_MemWB <= 32'b0;
                    MemtoReg_out_MemWB <= 2'b0;
                  RegWrite_out_MemWB <= 1'b0;
          end
        else if (en_MemWB == 1'b1) begin
                    PC4_out_MemWB <= PC4_in_MemWB;
                  Rd_addr_out_MemWB <= Rd_addr_MemWB;
                  ALU_out_MemWB <= ALU_in_MemWB;
```

```
                DMem_data_out_MemWB <= DMem_data_MemWB;
               MemtoReg_out_MemWB <= MemtoReg_in_MemWB;
               RegWrite_out_MemWB <= RegWrite_in_MemWB;
        end
endmodule
```

全部模块编写完成后，替换 lab5-1 中提供的各模块 ip 核，集成流水线 CPU。（此处为了 debug，额外添加了寄存器等接口，因此代码行数较多。）生成 bit 文件后，上板验证，

Lab5-3:流水线 CPU 实现（含 hazard）

截至本报告提交时，我还没有写出能跑通 hazard 的流水线 CPU，因此把现有的含有 bug 的源代码放在报告末尾处。（当前版本的代码，跑含 hazard 的指令 h.coe 时行为表现与不含冒险的代码基本一致，因此不在实验结果分析模块另外说明上板结果。）

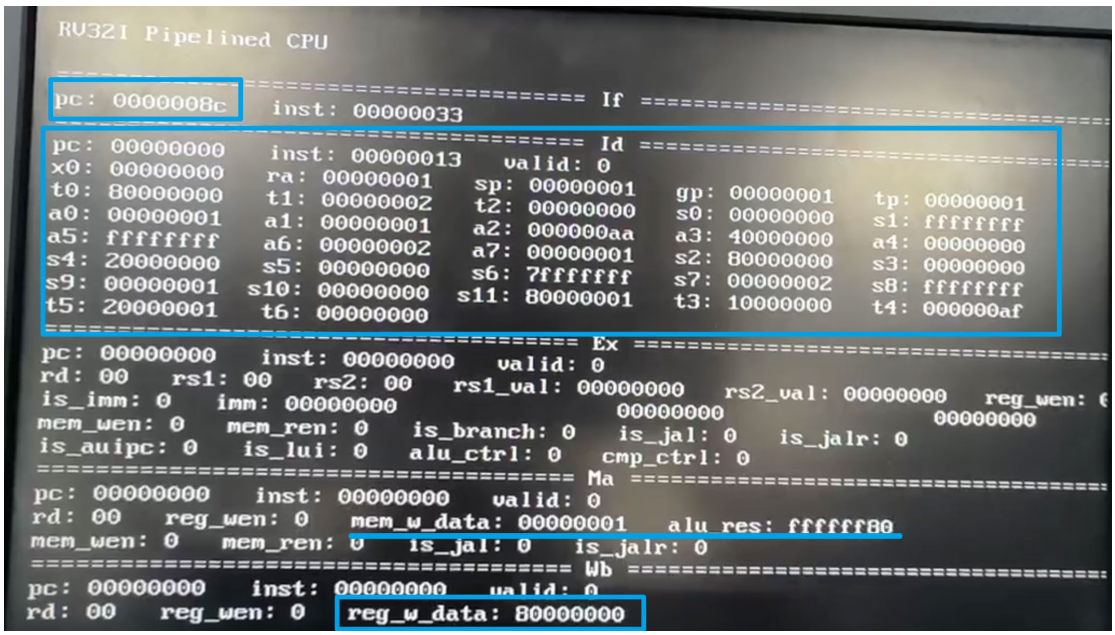
二、实验结果与分析

不含 hazard 版本（p.coe）：

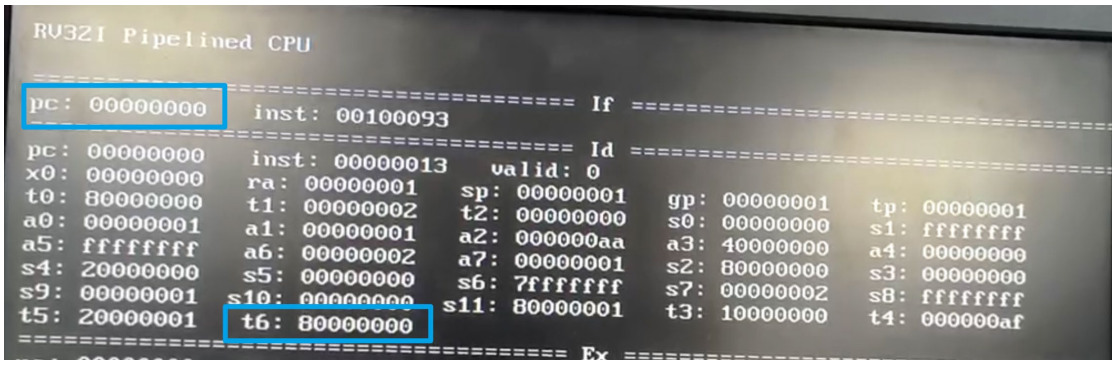
经过逐步比对，该流水线 CPU 的行为完全符合预期。

本报告内为了便于说明，只截取程序结束时的两个连续时钟周期，通过比对 ID 阶段各寄存器的值和期望值来说明流水线符合预期。（因为提供的观察接口并没有全部接入，因此图中有部分数据是观察不到并且默认为零的。这里将几个接入的数据标出，便于观察。）

PC = 0x8c：



PC = 0x00：



可以看到，图一中ID阶段显示寄存器t6的值还未写入，为零；而WB阶段返回数据reg_w_data为0x80000000，恰好为下一指令执行时被写回 t6 的数据。

更多的验证结果可以比对 ID 阶段各寄存器的值和下方的指令（0x68 处 andi 指令的结果在图上标错了，现用粉色字迹更正，）

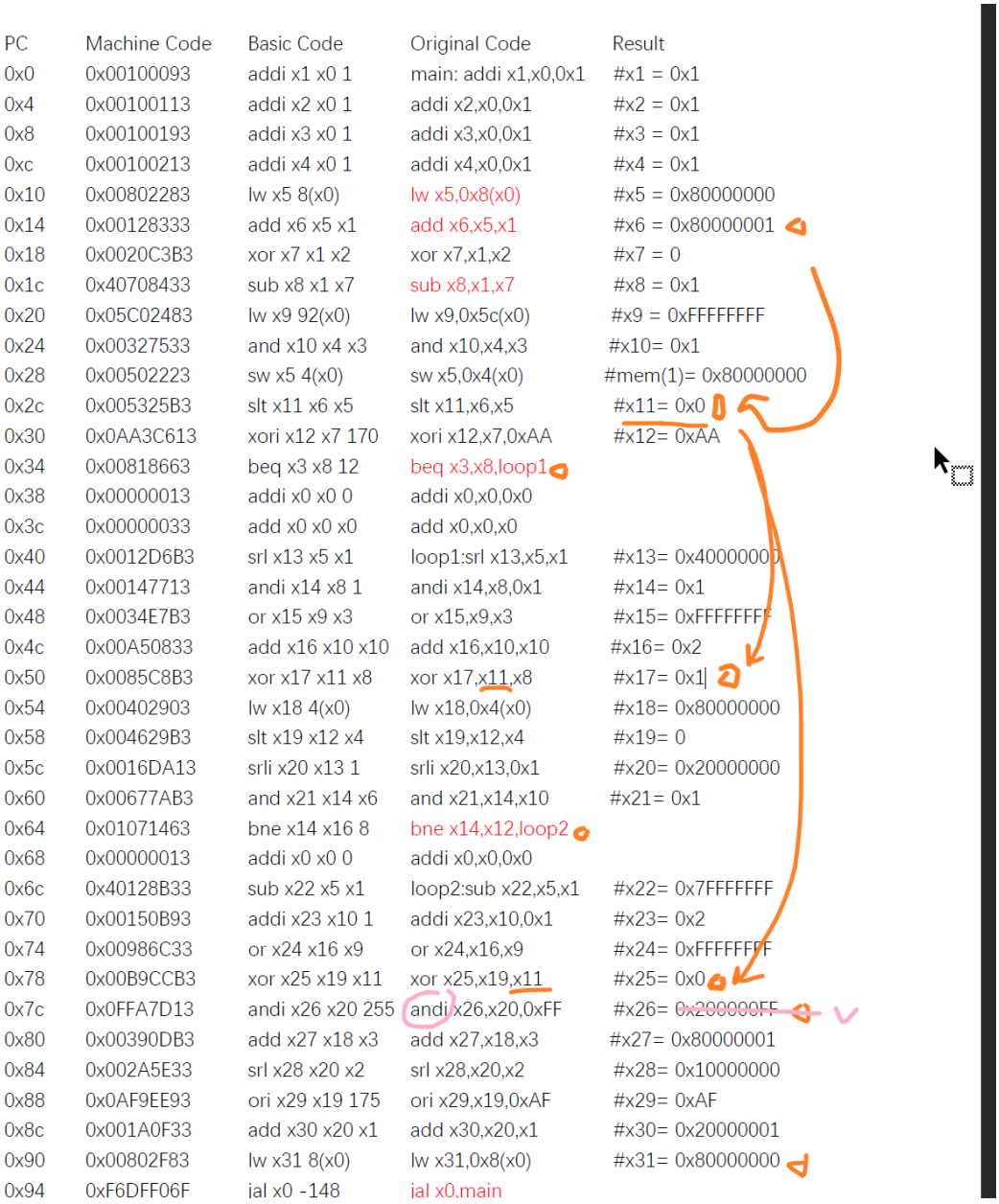
PC	Machine Code	Basic Code	Original Code	Result
0x0	0x00100093	addi x1 x0 1	main: addi x1,x0,0x1	#x1 = 0x1
0x4	0x00100113	addi x2 x0 1	addi x2,x0,0x1	#x2 = 0x1
0x8	0x00100193	addi x3 x0 1	addi x3,x0,0x1	#x3 = 0x1
0xc	0x00100213	addi x4 x0 1	addi x4,x0,0x1	#x4 = 0x1
0x10	0x00802283	lw x5 8(x0)	lw x5,0x8(x0)	#x5 = 0x80000000
0x14	0x00108333	add x6 x1 x1	add x6,x1,x1	#x6 = 0x2
0x18	0x0020C3B3	xor x7 x1 x2	xor x7,x1,x2	#x7 = 0
0x1c	0x40110433	sub x8 x2 x1	sub x8,x2,x1	#x8 = 0
0x20	0x05C02483	lw x9 92(x0)	lw x9,0x5c(x0)	#x9 = 0xFFFFFFFF
0x24	0x00327533	and x10 x4 x3	and x10,x4,x3	#x10= 0x1
0x28	0x00502223	sw x5 4(x0)	sw x5,0x4(x0)	#mem(1)= 0x80000000
0x2c	0x005325B3	slt x11 x6 x5	slt x11,x6,x5	#x11= 0x1
0x30	0x0AA3C613	xori x12 x7 170	xori x12,x7,0xAA	#x12= 0xAA
0x34	0x0012D6B3	srl x13 x5 x1	srl x13,x5,x1	#X13= 0x40000000
0x38	0x00147713	andi x14 x8 1	andi x14,x8,0x1	#x14= 0
0x3c	0x0034E7B3	or x15 x9 x3	or x15,x9,x3	#x15= 0xFFFFFFFF
0x40	0x00A50833	add x16 x10 x10	add x16,x10,x10	#x16= 0x2
0x44	0x0085C8B3	xor x17 x11 x8	xor x17,x11,x8	#x17= 0x1
0x48	0x00402903	lw x18 4(x0)	lw x18,0x4(x0)	#x18= 0x80000000
0x4c	0x004629B3	slt x19 x12 x4	slt x19,x12,x4	#x19= 0
0x50	0x0016DA13	srli x20 x13 1	srli x20,x13,0x1	#x20= 0x20000000
0x54	0x00677AB3	and x21 x14 x6	and x21,x14,x6	#x21= 0
0x58	0x40128B33	sub x22 x5 x1	sub x22,x5,x1	#x22= 0x7FFFFFFF
0x5c	0x00150B93	addi x23 x10 1	addi x23,x10,0x1	#x23= 0x2
0x60	0x00986C33	or x24 x16 x9	or x24,x16,x9	#x24= 0xFFFFFFFF
0x64	0x00B9CCB3	xor x25 x19 x11	xor x25,x19,x11	#x25= 0x1
0x68	0x0FFA7D13	andi x26 x20 255	andi x26,x20,0xFF	#x26= 0x000000FF
0x6c	0x00390DB3	add x27 x18 x3	add x27,x18,x3	#x27= 0x80000001
0x70	0x002A5E33	srl x28 x20 x2	srl x28,x20,x2	#x28= 0x10000000
0x74	0x0AF9EE93	ori x29 x19 175	ori x29,x19,0xAF	#x29= 0xAF
0x78	0x001A0F33	add x30 x20 x1	add x30,x20,x1	#x30= 0x20000001
0x7c	0x00802F83	lw x31 8(x0)	lw x31,0x8(x0)	#x31= 0x80000000

含 hazard 版本（h.coe）：

（因为我并没有写出能跑通的 CPU，这里仅给出不能跑通的结果并进行分析。）

下图是跑的指令，其中橙色标记处是和我的流水线 CPU 表现结果相比有出入的地方：

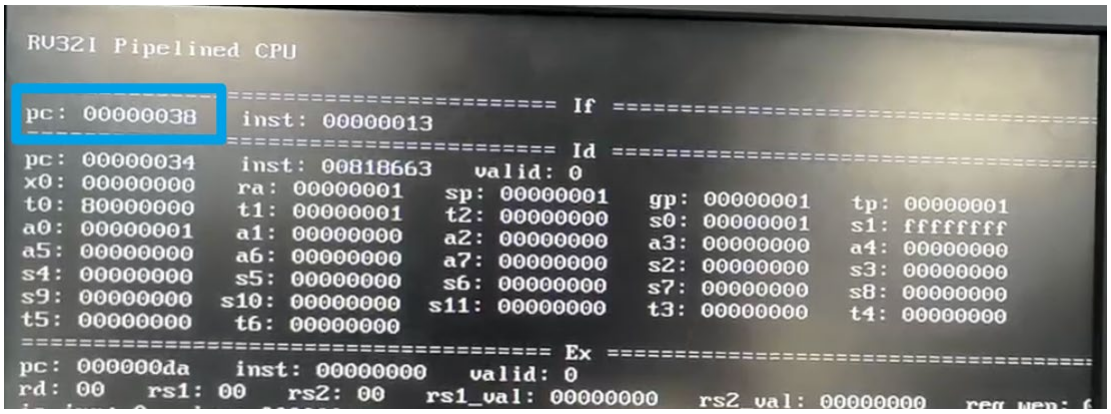
1. 0x14 处指令是第一次出现冒险的位置，由于上一条指令写回寄存器 x5，本条指令使用寄存器 x5 内的值，而实际上在（未能实现 hazard 功能的）流水线 CPU 内，上一条指令还没经过 WB 阶段，寄存器 x5 内的值还未更正，因此值出现不同。
2. 而在跳转指令处（如 0x34），流水线还未跳转便获取了下一条指令，导致中间个别本不该执行的指令也一并执行。
3. 几个有箭头的位置表明，上一步寄存器没有存放正确的值导致这一步结果出错（但是带入寄存器内实际值时，这几条指令都是正常执行的。）
4. 寄存器 t6 内没有值是因为实际跑的指令没有 0x90 这一条。

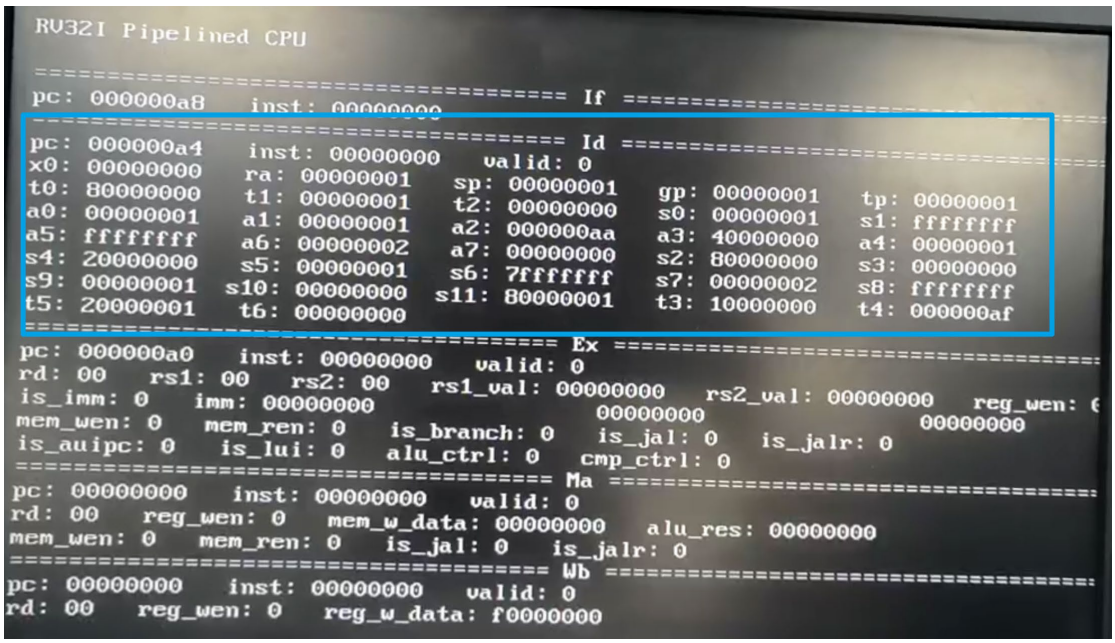


PC	Machine Code	Basic Code	Original Code	Result
0x0	0x00100093	addi x1 x0 1	main: addi x1,x0,0x1	#x1 = 0x1
0x4	0x00100113	addi x2 x0 1	addi x2,x0,0x1	#x2 = 0x1
0x8	0x00100193	addi x3 x0 1	addi x3,x0,0x1	#x3 = 0x1
0xc	0x00100213	addi x4 x0 1	addi x4,x0,0x1	#x4 = 0x1
0x10	0x00802283	lw x5 8(x0)	lw x5,0x8(x0)	#x5 = 0x80000000
0x14	0x00128333	add x6 x5 x1	add x6,x5,x1	#x6 = 0x80000001
0x18	0x0020C3B3	xor x7 x1 x2	xor x7,x1,x2	#x7 = 0
0x1c	0x40708433	sub x8 x1 x7	sub x8,x1,x7	#x8 = 0x1
0x20	0x05C02483	lw x9 92(x0)	lw x9,0x5c(x0)	#x9 = 0xFFFFFFFF
0x24	0x00327533	and x10 x4 x3	and x10,x4,x3	#x10= 0x1
0x28	0x00502223	sw x5 4(x0)	sw x5,0x4(x0)	#mem(1)= 0x80000000
0x2c	0x005325B3	slt x11 x6 x5	slt x11,x6,x5	#x11= 0x0
0x30	0x0AA3C613	xori x12 x7 170	xori x12,x7,0xAA	#x12= 0xAA
0x34	0x00818663	beq x3 x8 12	beq x3,x8,loop1	
0x38	0x00000013	addi x0 x0 0	addi x0,x0,0x0	
0x3c	0x00000033	add x0 x0 x0	add x0,x0,x0	
0x40	0x0012D6B3	srl x13 x5 x1	loop1:srl x13,x5,x1	#x13= 0x40000000
0x44	0x00147713	andi x14 x8 1	andi x14,x8,0x1	#x14= 0x1
0x48	0x0034E7B3	or x15 x9 x3	or x15,x9,x3	#x15= 0xFFFFFFFF
0x4c	0x00A50833	add x16 x10 x10	add x16,x10,x10	#x16= 0x2
0x50	0x0085C8B3	xor x17 x11 x8	xor x17,x11,x8	#x17= 0x1
0x54	0x00402903	lw x18 4(x0)	lw x18,0x4(x0)	#x18= 0x80000000
0x58	0x004629B3	slt x19 x12 x4	slt x19,x12,x4	#x19= 0
0x5c	0x0016DA13	srli x20 x13 1	srli x20,x13,0x1	#x20= 0x20000000
0x60	0x00677AB3	and x21 x14 x6	and x21,x14,x10	#x21= 0x1
0x64	0x01071463	bne x14 x16 8	bne x14,x12,loop2	
0x68	0x00000013	addi x0 x0 0	addi x0,x0,0x0	
0x6c	0x40128B33	sub x22 x5 x1	loop2:sub x22,x5,x1	#x22= 0x7FFFFFFF
0x70	0x00150B93	addi x23 x10 1	addi x23,x10,0x1	#x23= 0x2
0x74	0x00986C33	or x24 x16 x9	or x24,x16,x9	#x24= 0xFFFFFFFF
0x78	0x00B9CCB3	xor x25 x19 x11	xor x25,x19,x11	#x25= 0x0
0x7c	0x0FFA7D13	andi x26 x20 255	andi x26,x20,0xFF	#x26= 0x000000FF
0x80	0x00390DB3	add x27 x18 x3	add x27,x18,x3	#x27= 0x80000001
0x84	0x002A5E33	srl x28 x20 x2	srl x28,x20,x2	#x28= 0x10000000
0x88	0x0AF9EE93	ori x29 x19 175	ori x29,x19,0xAF	#x29= 0xAF
0x8c	0x001A0F33	add x30 x20 x1	add x30,x20,x1	#x30= 0x20000001
0x90	0x00802F83	lw x31 8(x0)	lw x31,0x8(x0)	#x31= 0x80000000
0x94	0xF6DFF06F	jal x0 -148	jal x0,main	

下面是实际流水线 CPU 实际跑的结果：

图一处，0x38 指令本该被跳过。
图二 ID 阶段的寄存器可以用来比对上述结果。

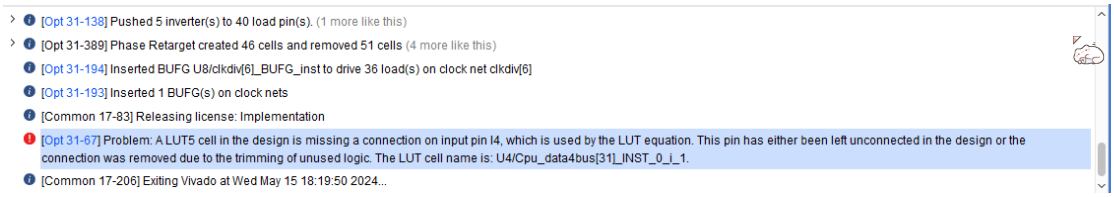




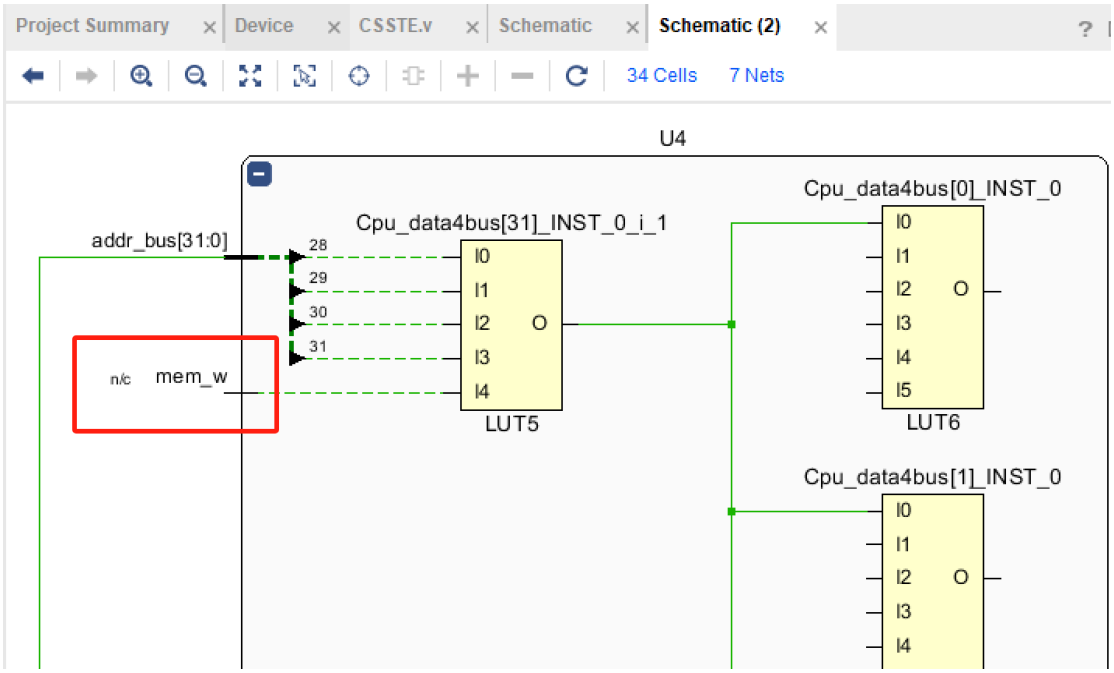
三、讨论、心得

本次实验非常非常的麻烦，也非常非常的困难！很遗憾截至上交报告时，还是没有写出正确的含冒险的流水线 CPU。过程中记录的一些报错和解决办法如下：

1. 执行的时候出现报错（如图）：



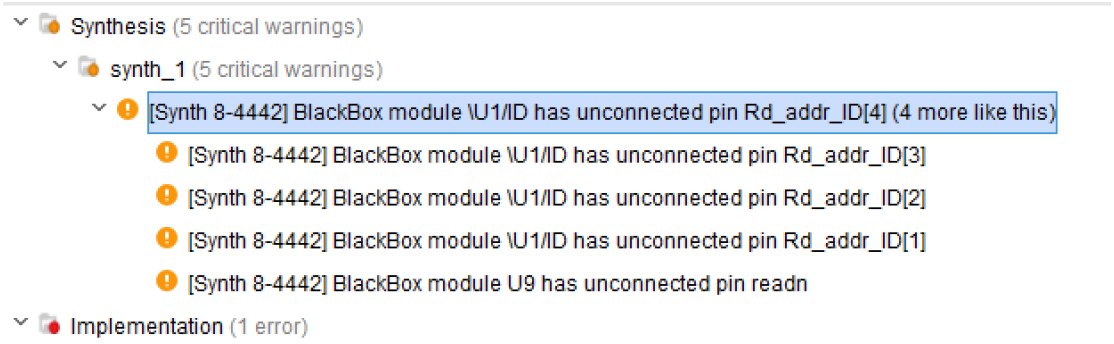
先是根据网上的信息（https://support.xilinx.com/s/article/72980?language=en_US）进行排查，发现 U4 模块线没接好（如下图 1）。改正后仍有错误，最后发现是命名不一致导致的（如下图二）。




```
Pipeline_CPU U1
    (.clk(clk_100mhz),.rst(rst),
     .Inst_IF(Inst_in),
     .Data_in(Data_in),
     .PC_out_IF(PC_out),
     .PC_out_ID(PC_out_ID),
     .Inst_ID(inst_ID),
     .PC_out_EX(PC_out_EX),
     .MemRW_EX(MemRW_EX),
     .MemRW_Mem(MemRW_Mem),
     .Data_out(Data_out),
     .Addr_out(Addr_out),
     .Data_out_WB(Data_out_WB)
    );
```

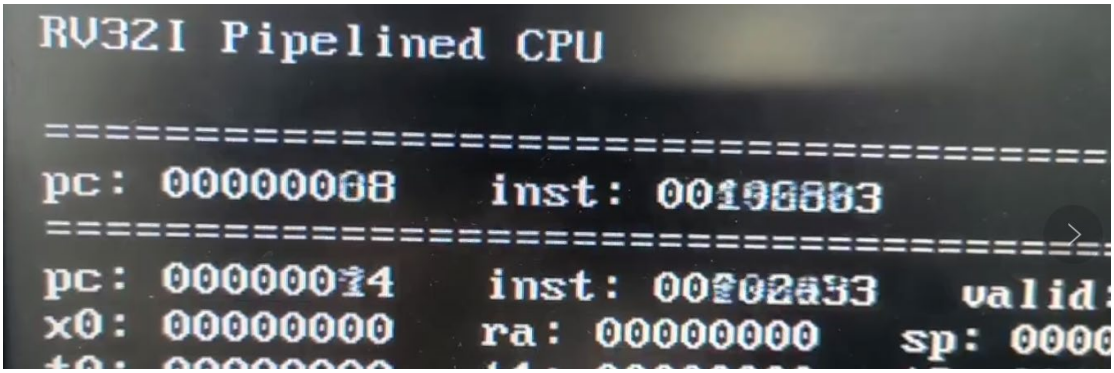
```
    (.addra(ram_addr),.clka(~clk_100mhz),.dina(ram_data_in),
     .douta(RAM_B_0_douta),.wea(U4_data_ram_we));

MIO_BUS U4
    (.clk(clk_100mhz),.rst(rst),.BTN(BTN_OK),.SW(SW_OK),
     .mem_w(MemRW),.Cpu_data2bus(Data_out),.addr_bus(Addr_out),
     .ram_data_out(RAM_B_0_douta),.led_out(LED_out),.counter_out(counter_out),
     .counter0_out(counter0_out),.counter1_out(counter1_out),.counter2_out(counter2_out),
     .Cpu_data4bus(Data_in),.ram_data_in(ram_data_in),.ram_addr(ram_addr),
     .data_ram_we(U4_data_ram_we),.GPIOf0000000_we(U7_EN),.GPIOe0000000_we(U5_EN),
     .counter_we(counter_we),.Peripheral_in(Peripheral_in));

Multi_8CH32 U5
    (.clk(~Clk_CPU),.rst(rst),.EN(U5_EN),.Test(SW_OK[7:5]),
```

2. 下图所示报错：



检查后发现是信号的位宽有错（因此上图报错没有报 0 位）

3. 在 lab5-1 阶段，最初上板时显示非常奇怪（vga 显示的数字疯狂闪动，没有固定值；板上则是不管怎么改变 SW 开关，七段管显示的数据都没有任何变化）：



检查后发现是 CPU 信号接错了（应该是 Clk_CPU，如图）

```
Pipeline_CPU U1
    (.clk(Clk_CPU),.rst(rst),
     .Inst_IF(Inst_in),
     .Data_in(Data_in),
```

4. 此外还有一个常见错误就是信号的位数接错了。

四、个人生活照片

五、debug 版本 hazard 流水线

```
module stall(
    input rst_stall,
    input [4:0]Rs1_addr_ID,
    input [4:0]Rs2_addr_ID,
    input RegWrite_out_IDEX,
    input RegWrite_out_EXMem,
```

```
    input [4:0]Rd_addr_out_IDEX,
    input [4:0]Rd_addr_out_EXMem,
    input Rs1_used,
    input Rs2_used,
    input Branch_ID,
    input BranchN_ID,
    input Jump_ID, //*
    input Branch_out_IDEX,
    input BranchN_out_IDEX,
    input Jump_out_IDEX,
    input Branch_out_EXMem,
    input BranchN_out_EXMem,
    input Jump_out_EXMem,
    output reg en_IF,
    output reg en_IFID,
    output reg NOP_IFID,
    output reg NOP_IDEX
    );\

    /* data hazard */
    reg Data_stall;
    always @(*) begin
        if(RegWrite_out_EXMem && Rs1_used && (Rs1_addr_ID != 5'b0) && (Rd_addr_out_EXMem
== Rs1_addr_ID))
            Data_stall = 1'b1;
        else if(RegWrite_out_EXMem && Rs2_used && (Rs2_addr_ID != 5'b0) &&
(Rd_addr_out_EXMem == Rs2_addr_ID))
            Data_stall = 1'b1;
        else if(RegWrite_out_IDEX && Rs1_used && (Rs1_addr_ID != 5'b0) &&
(Rd_addr_out_IDEX == Rs1_addr_ID))
            Data_stall = 1'b1;
        else if(RegWrite_out_IDEX && Rs2_used && (Rs2_addr_ID != 5'b0) &&
(Rd_addr_out_IDEX == Rs2_addr_ID))
            Data_stall = 1'b1;
        else
            Data_stall = 1'b0;
    end

    /* Control hazard */
    reg Control_stall;
    always @(*) begin
        if((Branch_ID || BranchN_ID || Jump_ID) || (Branch_out_IDEX || BranchN_out_IDEX
|| Jump_out_IDEX) || (Branch_out_EXMem || BranchN_out_EXMem || Jump_out_EXMem))
            Control_stall = 1'b1;
```

```
        else
            Control_stall = 1'b0;
    end

    /* stall */
    always @(*) begin
        if(rst_stall == 1'b1) begin
            en_IF = 1'b1;
            en_IFID = 1'b1;
            NOP_IDEX = 1'b0;
            NOP_IFID = 1'b0;
        end else begin
            if (Control_stall == 1'b1)
                NOP_IFID = 1'b1;   //Insert nop
            else
                NOP_IFID = 1'b0;
            if (Data_stall == 1'b1) begin
                en_IF = 1'b0;
                en_IFID = 1'b0;     //Hold pc and if/id reg
                NOP_IDEX = 1'b1;    //insert nop and disable REGWrite,MemRW
            end else begin
                en_IF = 1'b1;
                en_IFID = 1'b1;
                NOP_IDEX = 1'b0;
            end
        end
    end
endmodule
```