计算机组成原理 实验报告

> 姓名：张艺耀
>
> 学号：PB20111630

> 实验日期：2022-4-19

实验题目

流水线CPU设计

实验目的

理解流水线CPU的结构和工作原理

掌握流水线CPU的设计和调试方法，特别是流水线中数据相关和控制相关的处理

熟练掌握数据通路和控制器的设计和描述方法

实验平台

FPGAOL Vivado Mac + VSCode-remote + SSH + VLab

实验过程

Step 1 : 修改Lab4寄存器堆模块，使其满足写优先

原CPU采用Ripes稍加改进的数据通路（参见课程ppt）新添加的流水线数据通路大致为教材上的数据通路

register_file.v

```
module register_file (
    input clk, rst,
    input [4:0] ra0,     //读端口0地址
    output reg [31:0] rd0,    //读端口0数据

    input [4:0] ra1,     //读端口1地址
    output reg [31:0] rd1,    //读端口1数据

```

```
    input [4:0] ra2,    //读端口2地址
    output reg [31:0] rd2,  //读端口2数据

    input [4:0] wa, //写端口地址
    input we,   //写使能 高电平有效
    input [31:0] wd //写端口数据
);

reg [31:0] regfile [0:31];

always@(*) begin
    if(ra0 == 0) rd0 <= 32'h0;
    else if(we && (wa == ra0)) rd0 <= wd;
    else rd0 <= regfile[ra0];

    if(ra1 == 0) rd1 <= 32'h0;
    else if(we && (wa == ra1)) rd1 <= wd;
    else rd1 <= regfile[ra1];

    if(ra2 == 0) rd2 <= 32'h0;
    else if(we && (wa == ra2)) rd2 <= wd;
    else rd2 <= regfile[ra2];
end

integer i;

always @(posedge clk) begin
    if(rst) begin
        for(i = 0; i <= 31; i = i + 1) begin
            regfile[i] <= 32'b0;
        end
    end
    else if(we) begin
        if(wa == 0) regfile[wa] <= 0;   //r0内容恒为0
        else regfile[wa] <= wd;
    end
end

endmodule
```

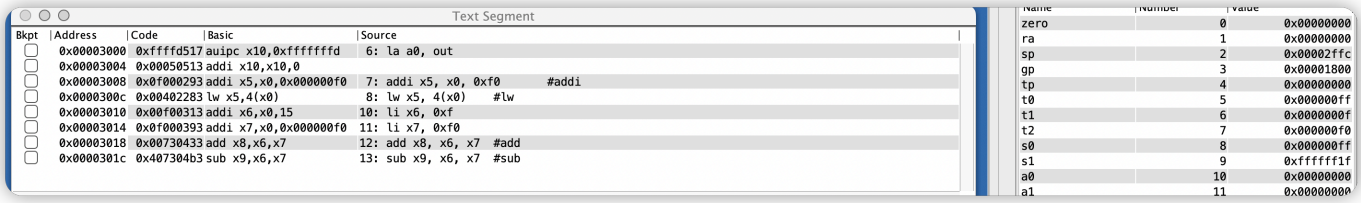
Step 2.1 : 前递单元

一段测试代码（暂时不含停顿和分支的情况）

```
.data
out: .word 0xff
in: .word 0xff

.text
la a0, out
addi x5, x0, 0xf0 #addi
lw x5, 4(x0)  #lw

li x6, 0xf
li x7, 0xf0
add x8, x6, x7  #add
sub x9, x6, x7  #sub
```

生成.coe文件并初始化`ins.mem`

Rars运行结果如下：



增加流水线寄存器 设计流水线基本框架。以下的代码是**仅含前递处理的版本 不支持停顿和控制冒险**。上面的测试用汇编也仅是为了测试这段代码。

IFID.v

```
module IFID(
    input clk,

    input [31:0]pc_in,
    input [31:0] inst,
    output reg [31:0] IFIDpc,
    output reg [31:0] IFIDinst
);

always@(posedge clk) begin
    IFIDpc <= pc_in;
    IFIDinst <= inst;
end

endmodule
```

IDEX.v

```
module IDEX(
    input clk,

    input [31:0] IFIDpc,
    input [4:0] Rd, //also part of forwarding unit
    input [31:0] Imm,
    input [31:0] rf_rd0,
    input [31:0] rf_rd1,

    //控制信号
    input rf_wr_en, //WB
    input alu_a_sel, alu_b_sel, //EX
    input [1:0] alu_ctrl,   //EX
    input dm_rd_ctrl,   //MEM
    input dm_wr_ctrl,   //MEM
    input [1:0] rf_wr_sel,  //WB

    input [2:0] comp_ctrl,  //EX
    input do_branch, do_jump,   //EX

    output reg [31:0] IDEXpc,
    output reg [4:0] IDEXRd,
    output reg [31:0] IDEXImm,
    output reg [31:0] IDEXrf_rd0,
    output reg [31:0] IDEXrf_rd1,

    output reg IDEXrf_wr_en, reg IDEXalu_a_sel,reg IDEXalu_b_sel,
    output reg [1:0] IDEXalu_ctrl,
    output reg IDEXdm_rd_ctrl,
    output reg IDEXdm_wr_ctrl,
    output reg [1:0] IDEXrf_wr_sel,
    output reg [2:0] IDEXcomp_ctrl,
    output reg IDEXdo_branch, reg IDEXdo_jump,

    //forwarding unit
    input [4:0] IFIDRs1,
    input [4:0] IFIDRs2,

    output reg [4:0] IDEXRs1,
    output reg [4:0] IDEXRs2
);

always@(posedge clk) begin
    IDEXpc <= IFIDpc;
    IDEXRd <= Rd;
    IDEXImm <= Imm;
    IDEXrf_rd0 <= rf_rd0;
    IDEXrf_rd1 <= rf_rd1;

    IDEXrf_wr_en <= rf_wr_en;
```

```
    IDEXalu_a_sel <= alu_a_sel;
    IDEXalu_b_sel <= alu_b_sel;
    IDEXalu_ctrl <= alu_ctrl;
    IDEXdm_rd_ctrl <= dm_rd_ctrl;
    IDEXdm_wr_ctrl <= dm_wr_ctrl;
    IDEXrf_wr_sel <= rf_wr_sel;
    IDEXcomp_ctrl <= comp_ctrl;
    IDEXdo_branch <= do_branch;
    IDEXdo_jump <= do_jump;

    IDEXRs1 <= IFIDRs1;
    IDEXRs2 <= IFIDRs2;
end

endmodule
```

EXMEM.v

```
module EXMEM (
    input clk,

    input [31:0] alu_out,
    input [31:0] IDEXrf_rd1,
    input [4:0] IDEXRd,
    input [31:0] IDEXpc,

    input IDEXrf_wr_en,
    input IDEXdm_rd_ctrl,
    input IDEXdm_wr_ctrl,
    input [1:0] IDEXrf_wr_sel,

    output reg [31:0] EXMEMalu_out,
    output reg [31:0] EXMEMrf_rd1,
    output reg [4:0] EXMEMRd,
    output reg [31:0] EXMEMpc,

    output reg EXMEMrf_wr_en,
    output reg EXMEMdm_rd_ctrl, //MEM
    output reg EXMEMdm_wr_ctrl, //MEM
    output reg [1:0] EXMEMrf_wr_sel
);

always@(posedge clk) begin
    EXMEMalu_out <= alu_out;
    EXMEMrf_rd1 <= IDEXrf_rd1;
    EXMEMRd <= IDEXRd;
    EXMEMpc <= IDEXpc;

    EXMEMrf_wr_en <= IDEXrf_wr_en;
    EXMEMdm_rd_ctrl <= IDEXdm_rd_ctrl;
    EXMEMdm_wr_ctrl <= IDEXdm_wr_ctrl;
```

```
    EXMEMrf_wr_sel <= IDEXrf_wr_sel;
end

endmodule
```

MEMWB.v

```
module MEMWB(
    input clk,

    input EXMEMrf_wr_en,
    input [1:0] EXMEMrf_wr_sel,
    input [31:0] dm_dout,
    input [31:0] EXMEMalu_out,
    input [4:0] EXMEMRd,
    input [31:0] EXMEMpc,

    output reg MEMWBrf_wr_en,
    output reg [1:0] MEMWBrf_wr_sel,
    output reg [31:0] MEMWBdm_dout,
    output reg [31:0] MEMWBalu_out,
    output reg [4:0] MEMWBRd,
    output reg [31:0] MEMWBpc
);

always@(posedge clk) begin
    MEMWBrf_wr_en <= EXMEMrf_wr_en;
    MEMWBrf_wr_sel <= EXMEMrf_wr_sel;
    MEMWBdm_dout <= dm_dout;
    MEMWBalu_out <= EXMEMalu_out;
    MEMWBRd <= EXMEMRd;
    MEMWBpc <= EXMEMpc;
end

endmodule
```

cpu.v(仅支持数据前递的非最终版本)

```
module cpu_pl(
    input clk,
    input rst,

    //io_bus
    output [31:0] io_addr,   //led和seg的地址
    output [31:0] io_dout,  //输出led和seg的数据
    output io_we,   //输出led和seg数据时的使能信号
    input [31:0] io_din,    //输出led和seg数据时的使能信号

    //debug_bus
    input [7:0] m_rf_addr,  //存储器（mem）或寄存器堆（rf）的调试读口地址
```

```
    output [31:0] rf_data,  //从rf读取的数据
    output [31:0] m_data,   //从mem读取的数据

    //PC/IF/ID
    output [31:0] pc_out,
    output [31:0] pcd,
    output [31:0] ir,
    output [31:0] pcin,

    //ID/EX
    output [31:0] pce,
    output [31:0] a,
    output [31:0] b,
    output [31:0] imm_debug,
    output [4:0] rd,
    output [31:0] ctrl,

    //EX/MEM
    output [31:0] y,
    output [31:0] bm,
    output [4:0] rdm,
    output [31:0] ctrlm,

    //MEM/WB
    output [31:0] ym,
    output [31:0] mdr,
    output [4:0] rdw,
    output [31:0] ctrlw
);

wire branch;
wire [31:0] pc;
wire [31:0] pc_in;
wire [31:0] pc_plus4;
wire [31:0] inst;
wire [31:0] imm_out;
wire rf_wr_en;
wire alu_a_sel;
wire alu_b_sel;
wire [1:0] alu_ctrl;
wire dm_rd_ctrl;
wire dm_wr_ctrl;
wire dm_wr_ctrl_aft;
wire [1:0] rf_wr_sel;
wire [2:0] comp_ctrl;
wire do_branch;
wire do_jump;

reg [31:0] rf_wd;
wire [31:0] rf_rd0, rf_rd1;
wire [31:0] alu_a, alu_b, alu_out;
wire [31:0] dm_dout;
```

```
65  wire [31:0] dm_dout_aft;
66
67  //5-stage pipeline
68  wire [31:0] IFIDpc;
69  wire [31:0] IFIDinst;
70
71  wire [31:0] IDEXpc;
72  wire [4:0] IDEXRd;
73  wire [31:0] IDEXImm;
74  wire [31:0] IDEXrf_rd0;
75  wire [31:0] IDEXrf_rd1;
76  wire IDEXrf_wr_en;
77  wire IDEXalu_a_sel;
78  wire IDEXalu_b_sel;
79  wire [1:0] IDEXalu_ctrl;
80  wire IDEXdm_rd_ctrl;
81  wire IDEXdm_wr_ctrl;
82  wire [1:0] IDEXrf_wr_sel;
83  wire [2:0] IDEXcomp_ctrl;
84  wire IDEXdo_branch;
85  wire IDEXdo_jump;
86
87  wire [31:0] EXMEMalu_out;
88  wire [31:0] EXMEMrf_rd1;
89  wire [4:0] EXMEMRd;
90  wire EXMEMrf_wr_en;
91  wire EXMEMdm_rd_ctrl;
92  wire EXMEMdm_wr_ctrl;
93  wire [1:0] EXMEMrf_wr_sel;
94  wire [31:0] EXMEMpc;
95
96  wire MEMWBrf_wr_en;
97  wire [1:0] MEMWBrf_wr_sel;
98  wire [31:0] MEMWBdm_dout;
99  wire [31:0] MEMWBalu_out;
100 wire [4:0] MEMWBRd;
101 wire [31:0] MEMWBpc;
102
103 /*forwarding unit*/
104 wire [4:0] IDEXRs1;
105 wire [4:0] IDEXRs2;
106 wire [1:0] ForwardA;
107 wire [1:0] ForwardB;
108 reg [31:0] IDEXrf_rd0_fd;
109 reg [31:0] IDEXrf_rd1_fd;
110
111 forward forward(IDEXRs1, IDEXRs2, IDEXRd, EXMEMRd, MEMWBRd, EXMEMrf_wr_en, MEMWBrf_wr_en,
    ForwardA, ForwardB);
112
113 always@(*) begin
114     case (ForwardA)
115         2'b00: IDEXrf_rd0_fd = IDEXrf_rd0;
```

```
        2'b01: IDEXrf_rd0_fd = rf_wd;
        2'b10: IDEXrf_rd0_fd = EXMEMalu_out;
        default: IDEXrf_rd0_fd = IDEXrf_rd0;
    endcase

    case (ForwardB)
        2'b00: IDEXrf_rd1_fd = IDEXrf_rd1;
        2'b01: IDEXrf_rd1_fd = rf_wd;
        2'b10: IDEXrf_rd1_fd = EXMEMalu_out;
        default: IDEXrf_rd1_fd = IDEXrf_rd1;
    endcase
end

//io_bus
assign dm_wr_ctrl_aft = (~(io_addr[10])) & EXMEMdm_wr_ctrl;  //AND gate
assign dm_dout_aft = io_addr[10] ? io_din : dm_dout;    //Mux
assign io_dout = rf_rd1;
assign io_we = io_addr[10] & dm_wr_ctrl;    //AND gate
assign io_addr = alu_out;

assign alu_a = IDEXalu_a_sel ? IDEXrf_rd0_fd : IDEXpc;
assign alu_b = IDEXalu_b_sel ? IDEXImm : IDEXrf_rd1_fd;
assign pc_in = (pc >= 32'h3000 && pc <= 32'h33ff) ? pc : 32'h0;

assign pc_out = pc_in;
assign pcd = IFIDpc;
assign ir = IFIDinst;
assign pce = IDEXpc;
assign a = IDEXrf_rd0;
assign b = IDEXrf_rd1;
assign imm_debug = IDEXImm;
assign rd = IDEXRd;
assign ctrl = {13'b0, IDEXrf_wr_en, IDEXrf_wr_sel, 2'b0, IDEXdm_rd_ctrl, IDEXdm_wr_ctrl,
2'b0, IDEXdo_jump, IDEXdo_branch, 2'b0, alu_a_sel, alu_b_sel, 2'b0, alu_ctrl};

assign y = EXMEMalu_out;
assign bm = EXMEMrf_rd1;
assign rdm = EXMEMRd;
assign ctrlm = {18'b0, EXMEMdm_rd_ctrl, EXMEMdm_wr_ctrl, 12'b0};

assign ym = MEMWBalu_out;
assign mdr = MEMWBdm_dout;
assign rdw = MEMWBRd;
assign ctrlw = {29'b0, MEMWBrf_wr_en, MEMWBrf_wr_sel}; //这里只有两个控制信号

program_counter program_counter(clk, rst, branch, alu_out, pc, pcin, pc_plus4);

IFID IFID(clk, pc_in, inst, IFIDpc, IFIDinst);

register_file register_file(.clk (clk), .rst (rst), .ra0 (IFIDinst[19:15]), .ra1
(IFIDinst[24:20]), .ra2 (m_rf_addr[4:0]), .wa (MEMWBRd), .wd (rf_wd), .we
(MEMWBrf_wr_en), .rd0 (rf_rd0), .rd1 (rf_rd1), .rd2 (rf_data));
```

```
165
166 imm imm(.inst (IFIDinst), .imm_out (imm_out));
167
168 controller controller(IFIDinst, rf_wr_en, alu_a_sel, alu_b_sel, alu_ctrl, dm_rd_ctrl,
    dm_wr_ctrl, rf_wr_sel, comp_ctrl, do_branch, do_jump);
169
170 IDEX IDEX(clk, IFIDpc, IFIDinst[11:7], imm_out, rf_rd0, rf_rd1, rf_wr_en, alu_a_sel,
    alu_b_sel, alu_ctrl, dm_rd_ctrl, dm_wr_ctrl, rf_wr_sel, comp_ctrl, do_branch, do_jump,
    IDEXpc,
171 IDEXRd, IDEXImm, IDEXrf_rd0, IDEXrf_rd1, IDEXrf_wr_en, IDEXalu_a_sel, IDEXalu_b_sel,
    IDEXalu_ctrl, IDEXdm_rd_ctrl, IDEXdm_wr_ctrl, IDEXrf_wr_sel, IDEXcomp_ctrl,
    IDEXdo_branch, IDEXdo_jump,
172 IFIDinst[19:15], IFIDinst[24:20], IDEXRs1, IDEXRs2);
173
174 alu alu(alu_a, alu_b, IDEXalu_ctrl, alu_out);
175
176 br br(.a (IDEXrf_rd0), .b (IDEXrf_rd1), .comp_ctrl (IDEXcomp_ctrl), .do_branch
    (IDEXdo_branch), .do_jump (IDEXdo_jump), .branch (branch));
177
178 EXMEM EXMEM(clk, alu_out, IDEXrf_rd1, IDEXRd, IDEXpc, IDEXrf_wr_en, IDEXdm_rd_ctrl,
    IDEXdm_wr_ctrl, IDEXrf_wr_sel, EXMEMalu_out, EXMEMrf_rd1, EXMEMRd, EXMEMpc,
    EXMEMrf_wr_en, EXMEMdm_rd_ctrl, EXMEMdm_wr_ctrl, EXMEMrf_wr_sel);
179
180 ins_mem ins_mem(.a (pc_in[9:2]), .spo (inst));
181
182 data_mem data_mem(.a (EXMEMalu_out[9:2]), .d (EXMEMrf_rd1), .dpra (m_rf_addr), .clk
    (clk), .we (dm_wr_ctrl_aft), .spo (dm_dout), .dpo (m_data));
183
184 MEMWB MEMWB(clk, EXMEMrf_wr_en, EXMEMrf_wr_sel, dm_dout, EXMEMalu_out, EXMEMRd, EXMEMpc,
    MEMWBrf_wr_en, MEMWBrf_wr_sel, MEMWBdm_dout, MEMWBalu_out, MEMWBRd, MEMWBpc);
185
186 //the last mux
187 always@(*) begin
188     case (MEMWBrf_wr_sel)
189         2'b00: rf_wd = 32'h0;
190         2'b01: rf_wd = MEMWBpc + 32'h4;
191         2'b10: rf_wd = MEMWBalu_out;
192         2'b11: rf_wd = MEMWBdm_dout;  //2'b11: rf_wd = dm_dout_aft;
193         default: rf_wd = 32'h0;
194     endcase
195 end
196
197 endmodule
```

forward.v

```
1 module forward(
2     input [4:0] IDEXRs1, IDEXRs2, IDEXRd, EXMEMRd, MEMWBRd,
3     input EXMEMrf_wr_en, MEMWBrf_wr_en,
4     output reg [2:0] ForwardA, ForwardB
5 );
```

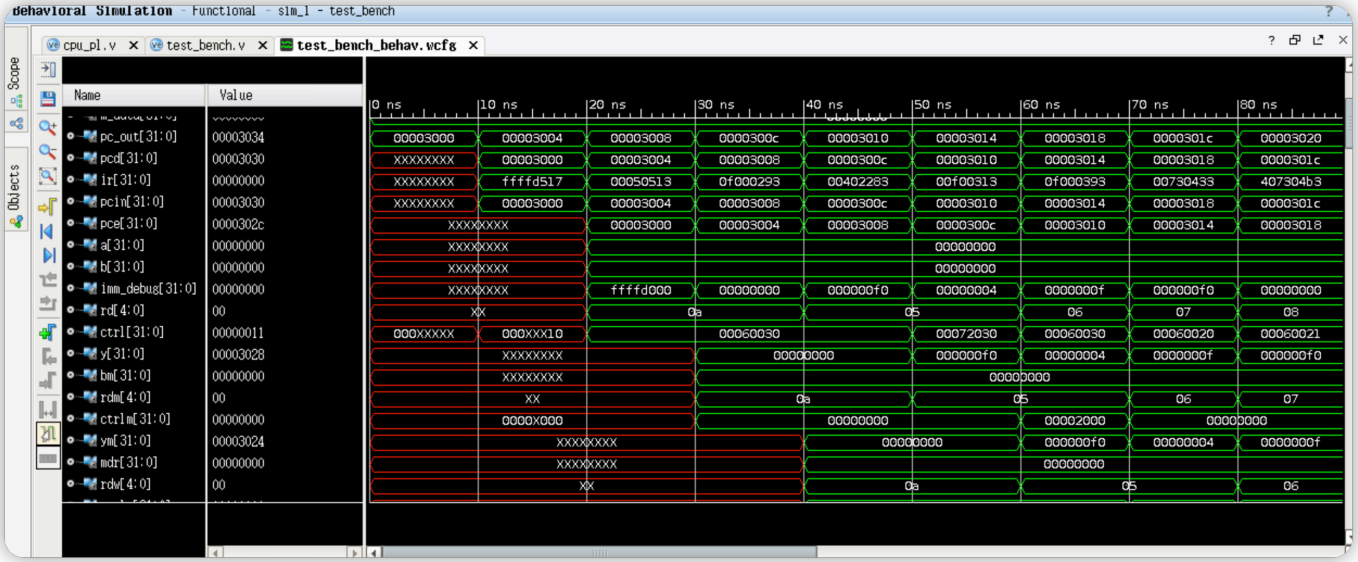
```
6
7  always@(*) begin
8      if(EXMEMrf_wr_en && (EXMEMRd != 0) && EXMEMRd == IDEXRs1) ForwardA = 2'b10;
9      else if(MEMWBrf_wr_en && (MEMWBRd != 0) && MEMWBRd == IDEXRs1) ForwardA = 2'b01;
10     else ForwardA = 2'b00;
11
12     if(EXMEMrf_wr_en && (EXMEMRd != 0) && EXMEMRd == IDEXRs2) ForwardB = 2'b10;
13     else if(MEMWBrf_wr_en && (MEMWBRd != 0) && MEMWBRd == IDEXRs2) ForwardB = 2'b01;
14     else ForwardB = 2'b00;
15 end
16
17 endmodule
```

test_bench.v

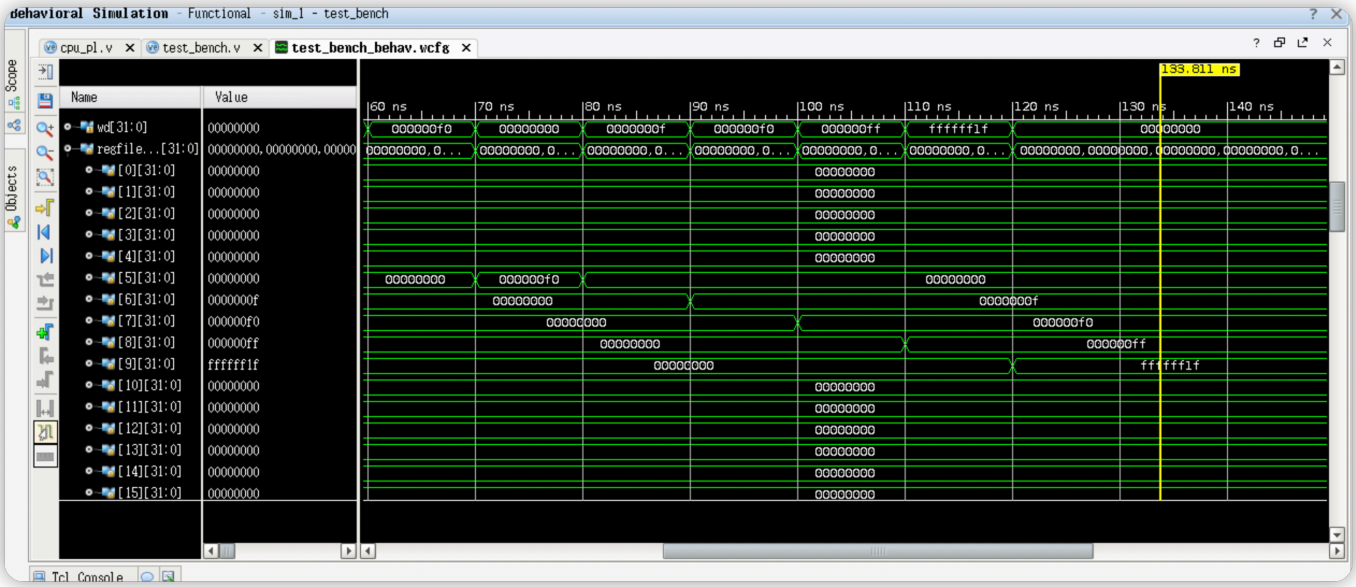
```
1  module test_bench();
2  reg clk;
3  reg rst;
4
5  wire [31:0] io_addr;
6  wire [31:0] io_dout;
7  wire io_we;
8  reg [31:0] io_din;
9
10 reg [7:0] m_rf_addr;
11 wire [31:0] rf_data;
12 wire [31:0] m_data;
13 wire [31:0] pc_out;
14 wire [31:0] pcd;
15 wire [31:0] ir;
16 wire [31:0] pcin;
17
18 //ID/EX
19 wire [31:0] pce;
20 wire [31:0] a;
21 wire [31:0] b;
22 wire [31:0] imm_debug;
23 wire [4:0] rd;
24 wire [31:0] ctrl;
25
26 //EX/MEM
27 wire [31:0] y;
28 wire [31:0] bm;
29 wire [4:0] rdm;
30 wire [31:0] ctrlm;
31
32 //MEM/WB
33 wire [31:0] ym;
34 wire [31:0] mdr;
35 wire [4:0] rdw;
36 wire [31:0] ctrlw;
```

```
37
38  cpu_pl cpu_pl(clk, rst, io_addr, io_dout, io_we, io_din, m_rf_addr, rf_data, m_data,
    pc_out, pcd, ir, pcin, pce, a, b, imm_debug, rd, ctrl, y, bm, rdm, ctrlm, ym, mdr, rdw,
    ctrlw);
39
40  initial begin
41      rst = 1; m_rf_addr = 0;
42      clk = 1;#5
43      clk = 0;
44      rst = 0;
45
46      forever #5 clk = ~clk;
47  end
48
49  initial begin
50      #180 $finish;
51  end
52
53  endmodule
```

仿真结果如下：



通过简单查看寄存器判断程序的正确性：



x5寄存器在MEMWBpc的下个时钟周期写入f0 说明addi指令运行正确。下一个时钟周期将4(x0)处值存到f0，这里未初始化数据存储器故默认初始化为0，可以看到x5在此时变为0，ld结果正确；之后将0xf和0xf0分别存入x6和x7寄存器，二者相加存到x8寄存器。在这里触发了前递（因为x6和x7值被更改后立刻被使用）x6对应MEMWB段前递、x7对应EXMEM段前递。x8计算结果为`0xff`。x9计算结果为`0xffffff1f`与Rars结果相符。

Step 2.2： 冒险检测单元

测试代码（含有停顿情况但不含分支情况）：

```
.data
out: .word 0xf00
in: .word 0xf00

.text
li x6, 0xf
li x7, 0xf0
add x8, x6, x7  #add
sub x9, x6, x7  #sub
lw x5, 4(x0)
add x5, x8, x5
sub x5, x9, x5
```

生成.coe文件并初始化`data_mem`和`ins_mem`

hazard_detection.v

```
module hazard_detection(
    input IDEXdm_rd_ctrl,
    input [4:0] IDEXRd,
    input [4:0] IFIDRs1,
    input [4:0] IFIDRs2,
```

```
    output PCWrite,
    output IFIDWrite,
    output stallpl
);

assign PCWrite = (IDEXdm_rd_ctrl && ((IDEXRd == IFIDRs1) || (IDEXRd == IFIDRs2))) ? 1'b0 : 1'b1;
assign IFIDWrite = (IDEXdm_rd_ctrl && ((IDEXRd == IFIDRs1) || (IDEXRd == IFIDRs2))) ? 1'b0 : 1'b1;
assign stallpl = (IDEXdm_rd_ctrl && ((IDEXRd == IFIDRs1) || (IDEXRd == IFIDRs2))) ? 1'b1 : 1'b0;

endmodule
```

pc.v

做如下更改：

```
module program_counter(
    input clk,
    input rst,
    input br,
    input [31:0] alu_out,
    input PCWrite,

    output reg [31:0] pc,
    output reg [31:0] pcin,
    output [31:0] pc_plus4
);

always@(posedge clk or posedge rst) begin
    if(rst) pc <= 32'h3000;
    else if(!PCWrite) pc <= pc;
    else if(br) pc <= alu_out;
    else pc <= pc_plus4;

    pcin <= pc;
end

assign pc_plus4 = pc + 32'h4;

endmodule
```

IFID.v

做如下更改：

```
module IFID(
    input clk,

    input [31:0]pc_in,
```

```
    input [31:0] inst,
    input IFIDWrite,

    output reg [31:0] IFIDpc,
    output reg [31:0] IFIDinst
);

always@(posedge clk) begin
    if(!IFIDWrite) begin
        IFIDpc <= IFIDpc;
        IFIDinst <= IFIDinst;
    end
    else begin
        IFIDpc <= pc_in;
        IFIDinst <= inst;
    end
end

endmodule
```

cpu_pl.v

做如下更改：

```
module cpu_pl(
    input clk,
    input rst,

    //io_bus
    output [31:0] io_addr,   //led和seg的地址
    output [31:0] io_dout,  //输出led和seg的数据
    output io_we,   //输出led和seg数据时的使能信号
    input [31:0] io_din,    //输出led和seg数据时的使能信号

    //debug_bus
    input [7:0] m_rf_addr,  //存储器（mem）或寄存器堆（rf）的调试读口地址
    output [31:0] rf_data,  //从rf读取的数据
    output [31:0] m_data,   //从mem读取的数据

    //PC/IF/ID
    output [31:0] pc_out,
    output [31:0] pcd,
    output [31:0] ir,
    output [31:0] pcin,

    //ID/EX
    output [31:0] pce,
    output [31:0] a,
    output [31:0] b,
    output [31:0] imm_debug,
    output [4:0] rd,
```

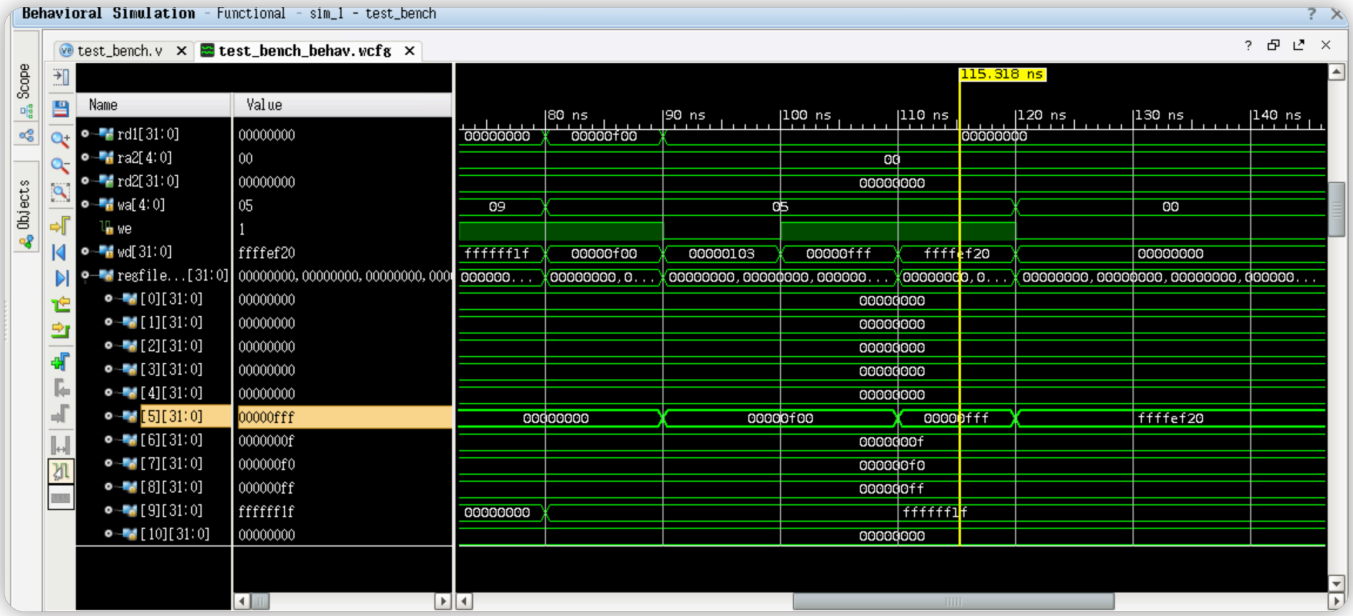
```
    output [31:0] ctrl,

    //EX/MEM
    output [31:0] y,
    output [31:0] bm,
    output [4:0] rdm,
    output [31:0] ctrlm,

    //MEM/WB
    output [31:0] ym,
    output [31:0] mdr,
    output [4:0] rdw,
    output [31:0] ctrlw
);

wire branch;
wire [31:0] pc;
wire [31:0] pc_in;
wire [31:0] pc_plus4;
wire [31:0] inst;
wire [31:0] imm_out;
wire rf_wr_en;
wire alu_a_sel;
wire alu_b_sel;
wire [1:0] alu_ctrl;
wire dm_rd_ctrl;
wire dm_wr_ctrl;
wire dm_wr_ctrl_aft;
wire [1:0] rf_wr_sel;
wire [2:0] comp_ctrl;
wire do_branch;
wire do_jump;

reg [31:0] rf_wd;
wire [31:0] rf_rd0, rf_rd1;
wire [31:0] alu_a, alu_b, alu_out;
wire [31:0] dm_dout;
wire [31:0] dm_dout_aft;

//5-stage pipeline
wire [31:0] IFIDpc;
wire [31:0] IFIDinst;

wire [31:0] IDEXpc;
wire [4:0] IDEXRd;
wire [31:0] IDEXImm;
wire [31:0] IDEXrf_rd0;
wire [31:0] IDEXrf_rd1;
wire IDEXrf_wr_en;
wire IDEXalu_a_sel;
wire IDEXalu_b_sel;
wire [1:0] IDEXalu_ctrl;
```

```
wire IDEXdm_rd_ctrl;
wire IDEXdm_wr_ctrl;
wire [1:0] IDEXrf_wr_sel;
wire [2:0] IDEXcomp_ctrl;
wire IDEXdo_branch;
wire IDEXdo_jump;

wire [31:0] EXMEMalu_out;
wire [31:0] EXMEMrf_rd1;
wire [4:0] EXMEMRd;
wire EXMEMrf_wr_en;
wire EXMEMdm_rd_ctrl;
wire EXMEMdm_wr_ctrl;
wire [1:0] EXMEMrf_wr_sel;
wire [31:0] EXMEMpc;

wire MEMWBrf_wr_en;
wire [1:0] MEMWBrf_wr_sel;
wire [31:0] MEMWBdm_dout;
wire [31:0] MEMWBalu_out;
wire [4:0] MEMWBRd;
wire [31:0] MEMWBpc;

//forwarding unit
wire [4:0] IDEXRs1;
wire [4:0] IDEXRs2;
wire [1:0] ForwardA;
wire [1:0] ForwardB;
reg [31:0] IDEXrf_rd0_fd;
reg [31:0] IDEXrf_rd1_fd;

forward forward(IDEXRs1, IDEXRs2, IDEXRd, EXMEMRd, MEMWBRd, EXMEMrf_wr_en, MEMWBrf_wr_en,
ForwardA, ForwardB);

always@(*) begin
    case (ForwardA)
        2'b00: IDEXrf_rd0_fd = IDEXrf_rd0;
        2'b01: IDEXrf_rd0_fd = rf_wd;
        2'b10: IDEXrf_rd0_fd = EXMEMalu_out;
        default: IDEXrf_rd0_fd = IDEXrf_rd0;
    endcase

    case (ForwardB)
        2'b00: IDEXrf_rd1_fd = IDEXrf_rd1;
        2'b01: IDEXrf_rd1_fd = rf_wd;
        2'b10: IDEXrf_rd1_fd = EXMEMalu_out;
        default: IDEXrf_rd1_fd = IDEXrf_rd1;
    endcase
end

//hazard detection
wire PCWrite;
```

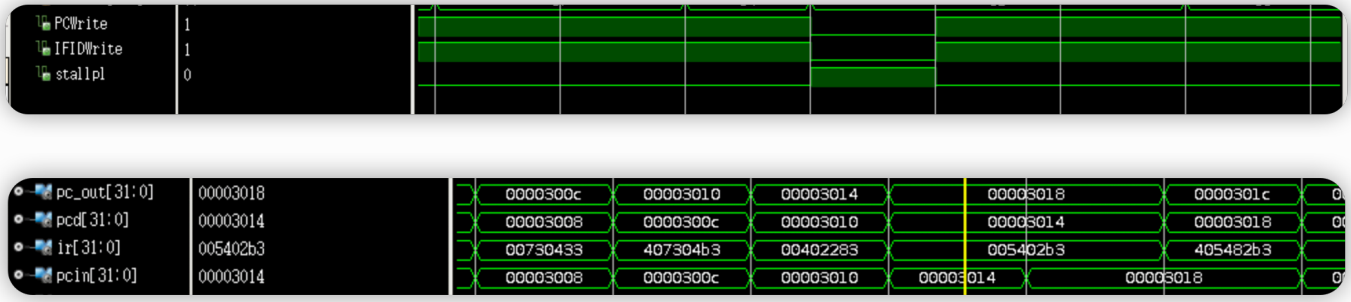
```
wire IFIDWrite;
wire stallpl;

hazard_detection hazard_detection(IDEXdm_rd_ctrl, IDEXRd, IFIDinst[19:15],
IFIDinst[24:20], PCWrite, IFIDWrite, stallpl);

reg rf_wr_en_hd;
reg dm_wr_ctrl_hd;
always@(*) begin
    if(stallpl) rf_wr_en_hd <= 1'b0;
    else rf_wr_en_hd <= rf_wr_en;
    if(stallpl) dm_wr_ctrl_hd <= 1'b0;
    else dm_wr_ctrl_hd <= dm_wr_ctrl;
end

//io_bus
assign dm_wr_ctrl_aft = (~(io_addr[10])) & EXMEMdm_wr_ctrl;  //AND gate
assign dm_dout_aft = io_addr[10] ? io_din : dm_dout;    //Mux
assign io_dout = rf_rd1;
assign io_we = io_addr[10] & dm_wr_ctrl;    //AND gate
assign io_addr = alu_out;

assign alu_a = IDEXalu_a_sel ? IDEXrf_rd0_fd : IDEXpc;
assign alu_b = IDEXalu_b_sel ? IDEXImm : IDEXrf_rd1_fd;
assign pc_in = (pc >= 32'h3000 && pc <= 32'h33ff) ? pc : 32'h0;

assign pc_out = pc_in;
assign pcd = IFIDpc;
assign ir = IFIDinst;
assign pce = IDEXpc;
assign a = IDEXrf_rd0;
assign b = IDEXrf_rd1;
assign imm_debug = IDEXImm;
assign rd = IDEXRd;
assign ctrl = {13'b0, IDEXrf_wr_en, IDEXrf_wr_sel, 2'b0, IDEXdm_rd_ctrl, IDEXdm_wr_ctrl,
2'b0, IDEXdo_jump, IDEXdo_branch, 2'b0, alu_a_sel, alu_b_sel, 2'b0, alu_ctrl};

assign y = EXMEMalu_out;
assign bm = EXMEMrf_rd1;
assign rdm = EXMEMRd;
assign ctrlm = {18'b0, EXMEMdm_rd_ctrl, EXMEMdm_wr_ctrl, 12'b0};

assign ym = MEMWBalu_out;
assign mdr = MEMWBdm_dout;
assign rdw = MEMWBRd;
assign ctrlw = {29'b0, MEMWBrf_wr_en, MEMWBrf_wr_sel}; //这里只有两个控制信号

program_counter program_counter(clk, rst, branch, alu_out, PCWrite, pc, pcin, pc_plus4);

IFID IFID(clk, pc_in, inst, IFIDWrite, IFIDpc, IFIDinst);

```

```
register_file register_file(.clk (clk), .rst (rst), .ra0 (IFIDinst[19:15]), .ra1 (IFIDinst[24:20]), .ra2 (m_rf_addr[4:0]), .wa (MEMWBRd), .wd (rf_wd), .we (MEMWBrf_wr_en), .rd0 (rf_rd0), .rd1 (rf_rd1), .rd2 (rf_data));

imm imm(.inst (IFIDinst), .imm_out (imm_out));

controller controller(IFIDinst, rf_wr_en, alu_a_sel, alu_b_sel, alu_ctrl, dm_rd_ctrl, dm_wr_ctrl, rf_wr_sel, comp_ctrl, do_branch, do_jump);

IDEX IDEX(clk, IFIDpc, IFIDinst[11:7], imm_out, rf_rd0, rf_rd1, rf_wr_en_hd, alu_a_sel, alu_b_sel, alu_ctrl, dm_rd_ctrl, dm_wr_ctrl_hd, rf_wr_sel, comp_ctrl, do_branch, do_jump, IDEXpc,
IDEXRd, IDEXImm, IDEXrf_rd0, IDEXrf_rd1, IDEXrf_wr_en, IDEXalu_a_sel, IDEXalu_b_sel, IDEXalu_ctrl, IDEXdm_rd_ctrl, IDEXdm_wr_ctrl, IDEXrf_wr_sel, IDEXcomp_ctrl, IDEXdo_branch, IDEXdo_jump,
IFIDinst[19:15], IFIDinst[24:20], IDEXRs1, IDEXRs2);

alu alu(alu_a, alu_b, IDEXalu_ctrl, alu_out);

br br(.a (IDEXrf_rd0), .b (IDEXrf_rd1), .comp_ctrl (IDEXcomp_ctrl), .do_branch (IDEXdo_branch), .do_jump (IDEXdo_jump), .branch (branch));

EXMEM EXMEM(clk, alu_out, IDEXrf_rd1, IDEXRd, IDEXpc, IDEXrf_wr_en, IDEXdm_rd_ctrl, IDEXdm_wr_ctrl, IDEXrf_wr_sel, EXMEMalu_out, EXMEMrf_rd1, EXMEMRd, EXMEMpc, EXMEMrf_wr_en, EXMEMdm_rd_ctrl, EXMEMdm_wr_ctrl, EXMEMrf_wr_sel);

ins_mem ins_mem(.a (pc_in[9:2]), .spo (inst));

data_mem data_mem(.a (EXMEMalu_out[9:2]), .d (EXMEMrf_rd1), .dpra (m_rf_addr), .clk (clk), .we (dm_wr_ctrl_aft), .spo (dm_dout), .dpo (m_data));

MEMWB MEMWB(clk, EXMEMrf_wr_en, EXMEMrf_wr_sel, dm_dout, EXMEMalu_out, EXMEMRd, EXMEMpc, MEMWBrf_wr_en, MEMWBrf_wr_sel, MEMWBdm_dout, MEMWBalu_out, MEMWBRd, MEMWBpc);

//the last mux
always@(*) begin
    case (MEMWBrf_wr_sel)
        2'b00: rf_wd = 32'h0;
        2'b01: rf_wd = MEMWBpc + 32'h4;
        2'b10: rf_wd = MEMWBalu_out;
        2'b11: rf_wd = MEMWBdm_dout;  //2'b11: rf_wd = dm_dout_aft;
        default: rf_wd = 32'h0;
    endcase
end

endmodule
```

仿真结果：



可见lw之后的addi可以正常运行得到正确结果（阻塞了一个时钟周期）。



Step 2.3 : 控制冒险

测试代码（含分支情况）：

```
.data
out: .word 0xf00
in: .word 0xf00

.text
li x7, 0x1
li x8, 0x1

sub x7, x7, x8  #sub
beqz x7, HERE

add x6, x7, x8
sub x6, x7, x8

HERE:
li x6, 0xff
jal x6, HERE
```

hazard_detection.v

```
module hazard_detection(
    input IDEXdm_rd_ctrl,
    input [4:0] IDEXRd,
    input [4:0] IFIDRs1,
    input [4:0] IFIDRs2,
    output PCWrite,
    output IFIDWrite,
    output stallpl,

    input branch,
    output dFlush, eFlush
);

assign PCWrite = (IDEXdm_rd_ctrl && ((IDEXRd == IFIDRs1) || (IDEXRd == IFIDRs2))) ? 1'b0 : 1'b1;
assign IFIDWrite = (IDEXdm_rd_ctrl && ((IDEXRd == IFIDRs1) || (IDEXRd == IFIDRs2))) ? 1'b0 : 1'b1;
assign stallpl = (IDEXdm_rd_ctrl && ((IDEXRd == IFIDRs1) || (IDEXRd == IFIDRs2))) ? 1'b1 : 1'b0;
assign dFlush = (branch) ? 1'b1 : 1'b0;
assign eFlush = (branch) ? 1'b1 : 1'b0;

endmodule
```

IFID.v

```
module IFID(
    input clk,

    input [31:0]pc_in,
    input [31:0] inst,
    input IFIDWrite,

    output reg [31:0] IFIDpc,
    output reg [31:0] IFIDinst,

    //hazard_detection
    input dFlush
);

always@(posedge clk) begin
    if(dFlush) begin
        IFIDpc <= 32'b0;
        IFIDinst <= 32'b0;
    end
    else if(!IFIDWrite) begin
        IFIDpc <= IFIDpc;
        IFIDinst <= IFIDinst;
    end
```

```
    else begin
        IFIDpc <= pc_in;
        IFIDinst <= inst;
    end
end

endmodule
```

IDEX.v

```
module IDEX(
    input clk,

    input [31:0] IFIDpc,
    input [4:0] Rd, //also part of forwarding unit
    input [31:0] Imm,
    input [31:0] rf_rd0,
    input [31:0] rf_rd1,

    //控制信号
    input rf_wr_en, //WB
    input alu_a_sel, alu_b_sel, //EX
    input [1:0] alu_ctrl,   //EX
    input dm_rd_ctrl,   //MEM
    input dm_wr_ctrl,   //MEM
    input [1:0] rf_wr_sel,  //WB

    input [2:0] comp_ctrl,  //EX
    input do_branch, do_jump,   //EX

    output reg [31:0] IDEXpc,
    output reg [4:0] IDEXRd,
    output reg [31:0] IDEXImm,
    output reg [31:0] IDEXrf_rd0,
    output reg [31:0] IDEXrf_rd1,

    output reg IDEXrf_wr_en, reg IDEXalu_a_sel,reg IDEXalu_b_sel,
    output reg [1:0] IDEXalu_ctrl,
    output reg IDEXdm_rd_ctrl,
    output reg IDEXdm_wr_ctrl,
    output reg [1:0] IDEXrf_wr_sel,
    output reg [2:0] IDEXcomp_ctrl,
    output reg IDEXdo_branch, reg IDEXdo_jump,

    //forwarding unit
    input [4:0] IFIDRs1,
    input [4:0] IFIDRs2,
    input eFlush,

    output reg [4:0] IDEXRs1,
    output reg [4:0] IDEXRs2
```

```
);

always@(posedge clk) begin
    if(eFlush) begin
        IDEXpc <= 0;
        IDEXRd <= 0;
        IDEXImm <= 0;
        IDEXrf_rd0 <= 0;
        IDEXrf_rd1 <= 0;

        IDEXrf_wr_en <= 0;
        IDEXalu_a_sel <= 0;
        IDEXalu_b_sel <= 0;
        IDEXalu_ctrl <= 0;
        IDEXdm_rd_ctrl <= 0;
        IDEXdm_wr_ctrl <= 0;
        IDEXrf_wr_sel <= 0;
        IDEXcomp_ctrl <= 0;
        IDEXdo_branch <= 0;
        IDEXdo_jump <= 0;

        IDEXRs1 <= 0;
        IDEXRs2 <= 0;
    end
    else begin
        IDEXpc <= IFIDpc;
        IDEXRd <= Rd;
        IDEXImm <= Imm;
        IDEXrf_rd0 <= rf_rd0;
        IDEXrf_rd1 <= rf_rd1;

        IDEXrf_wr_en <= rf_wr_en;
        IDEXalu_a_sel <= alu_a_sel;
        IDEXalu_b_sel <= alu_b_sel;
        IDEXalu_ctrl <= alu_ctrl;
        IDEXdm_rd_ctrl <= dm_rd_ctrl;
        IDEXdm_wr_ctrl <= dm_wr_ctrl;
        IDEXrf_wr_sel <= rf_wr_sel;
        IDEXcomp_ctrl <= comp_ctrl;
        IDEXdo_branch <= do_branch;
        IDEXdo_jump <= do_jump;

        IDEXRs1 <= IFIDRs1;
        IDEXRs2 <= IFIDRs2;
    end
end

endmodule
```

cpu_pl.v

注意对br模块输入端口的更改。

```
module cpu_pl(
    input clk,
    input rst,

    //io_bus
    output [31:0] io_addr,   //led和seg的地址
    output [31:0] io_dout,  //输出led和seg的数据
    output io_we,   //输出led和seg数据时的使能信号
    input [31:0] io_din,    //输出led和seg数据时的使能信号

    //debug_bus
    input [7:0] m_rf_addr,  //存储器（mem）或寄存器堆（rf）的调试读口地址
    output [31:0] rf_data,  //从rf读取的数据
    output [31:0] m_data,   //从mem读取的数据

    //PC/IF/ID
    output [31:0] pc_out,
    output [31:0] pcd,
    output [31:0] ir,
    output [31:0] pcin,

    //ID/EX
    output [31:0] pce,
    output [31:0] a,
    output [31:0] b,
    output [31:0] imm_debug,
    output [4:0] rd,
    output [31:0] ctrl,

    //EX/MEM
    output [31:0] y,
    output [31:0] bm,
    output [4:0] rdm,
    output [31:0] ctrlm,

    //MEM/WB
    output [31:0] ym,
    output [31:0] mdr,
    output [4:0] rdw,
    output [31:0] ctrlw
);

wire branch;
wire [31:0] pc;
wire [31:0] pc_in;
wire [31:0] pc_plus4;
wire [31:0] inst;
wire [31:0] imm_out;
```

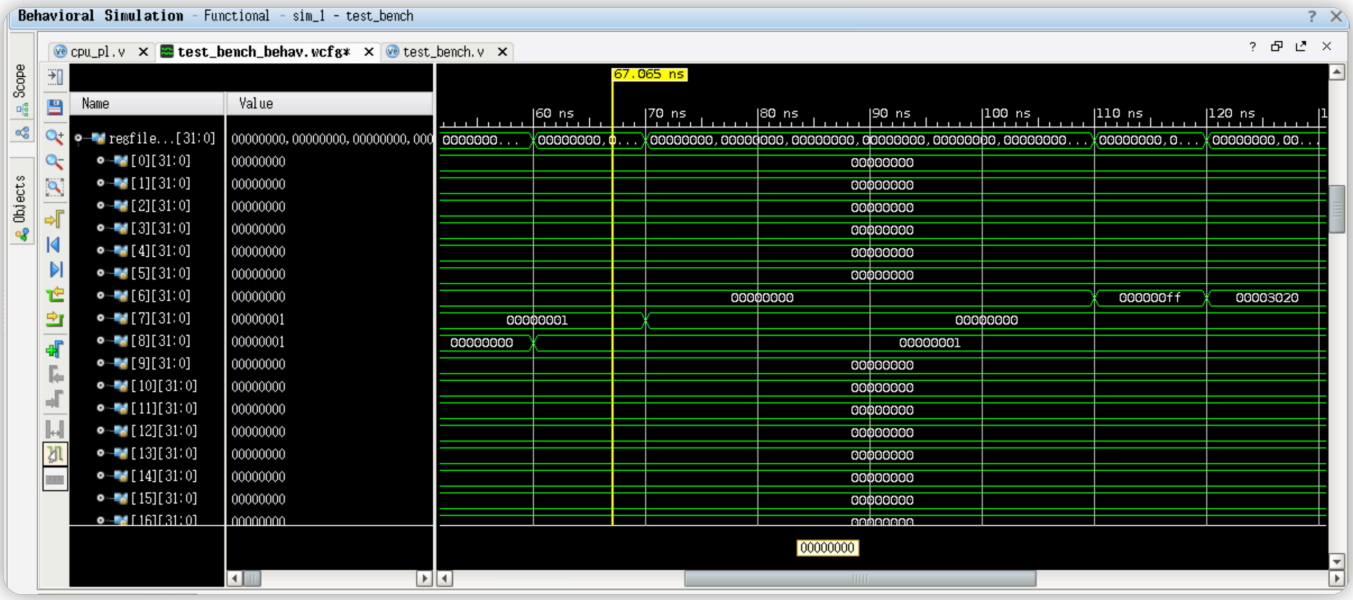
```
wire rf_wr_en;
wire alu_a_sel;
wire alu_b_sel;
wire [1:0] alu_ctrl;
wire dm_rd_ctrl;
wire dm_wr_ctrl;
wire dm_wr_ctrl_aft;
wire [1:0] rf_wr_sel;
wire [2:0] comp_ctrl;
wire do_branch;
wire do_jump;

reg [31:0] rf_wd;
wire [31:0] rf_rd0, rf_rd1;
wire [31:0] alu_a, alu_b, alu_out;
wire [31:0] dm_dout;
wire [31:0] dm_dout_aft;

//5-stage pipeline
wire [31:0] IFIDpc;
wire [31:0] IFIDinst;

wire [31:0] IDEXpc;
wire [4:0] IDEXRd;
wire [31:0] IDEXImm;
wire [31:0] IDEXrf_rd0;
wire [31:0] IDEXrf_rd1;
wire IDEXrf_wr_en;
wire IDEXalu_a_sel;
wire IDEXalu_b_sel;
wire [1:0] IDEXalu_ctrl;
wire IDEXdm_rd_ctrl;
wire IDEXdm_wr_ctrl;
wire [1:0] IDEXrf_wr_sel;
wire [2:0] IDEXcomp_ctrl;
wire IDEXdo_branch;
wire IDEXdo_jump;

wire [31:0] EXMEMalu_out;
wire [31:0] EXMEMrf_rd1;
wire [4:0] EXMEMRd;
wire EXMEMrf_wr_en;
wire EXMEMdm_rd_ctrl;
wire EXMEMdm_wr_ctrl;
wire [1:0] EXMEMrf_wr_sel;
wire [31:0] EXMEMpc;

wire MEMWBrf_wr_en;
wire [1:0] MEMWBrf_wr_sel;
wire [31:0] MEMWBdm_dout;
wire [31:0] MEMWBalu_out;
wire [4:0] MEMWBRd;
```

```
wire [31:0] MEMWBpc;

//forwarding unit
wire [4:0] IDEXRs1;
wire [4:0] IDEXRs2;
wire [1:0] ForwardA;
wire [1:0] ForwardB;
reg [31:0] IDEXrf_rd0_fd;
reg [31:0] IDEXrf_rd1_fd;

forward forward(IDEXRs1, IDEXRs2, IDEXRd, EXMEMRd, MEMWBRd, EXMEMrf_wr_en, MEMWBrf_wr_en,
ForwardA, ForwardB);

always@(*) begin
    case (ForwardA)
        2'b00: IDEXrf_rd0_fd = IDEXrf_rd0;
        2'b01: IDEXrf_rd0_fd = rf_wd;
        2'b10: IDEXrf_rd0_fd = EXMEMalu_out;
        default: IDEXrf_rd0_fd = IDEXrf_rd0;
    endcase

    case (ForwardB)
        2'b00: IDEXrf_rd1_fd = IDEXrf_rd1;
        2'b01: IDEXrf_rd1_fd = rf_wd;
        2'b10: IDEXrf_rd1_fd = EXMEMalu_out;
        default: IDEXrf_rd1_fd = IDEXrf_rd1;
    endcase
end

//hazard detection
wire PCWrite;
wire IFIDWrite;
wire stallpl;

wire dFlush;
wire eFlush;

hazard_detection hazard_detection(IDEXdm_rd_ctrl, IDEXRd, IFIDinst[19:15],
IFIDinst[24:20], PCWrite, IFIDWrite, stallpl, branch, dFlush, eFlush);

reg rf_wr_en_hd;
reg dm_wr_ctrl_hd;
always@(*) begin
    if(stallpl) rf_wr_en_hd <= 1'b0;
    else rf_wr_en_hd <= rf_wr_en;
    if(stallpl) dm_wr_ctrl_hd <= 1'b0;
    else dm_wr_ctrl_hd <= dm_wr_ctrl;
end

//io_bus
assign dm_wr_ctrl_aft = (~(io_addr[10])) & EXMEMdm_wr_ctrl;  //AND gate
assign dm_dout_aft = io_addr[10] ? io_din : dm_dout;    //Mux
```

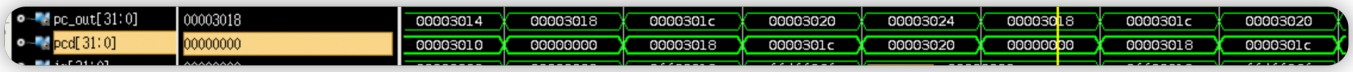
```
assign io_dout = rf_rd1;
assign io_we = io_addr[10] & dm_wr_ctrl;    //AND gate
assign io_addr = alu_out;

assign alu_a = IDEXalu_a_sel ? IDEXrf_rd0_fd : IDEXpc;
assign alu_b = IDEXalu_b_sel ? IDEXImm : IDEXrf_rd1_fd;
assign pc_in = (pc >= 32'h3000 && pc <= 32'h33ff) ? pc : 32'h0;

assign pc_out = pc_in;
assign pcd = IFIDpc;
assign ir = IFIDinst;
assign pce = IDEXpc;
assign a = IDEXrf_rd0;
assign b = IDEXrf_rd1;
assign imm_debug = IDEXImm;
assign rd = IDEXRd;
assign ctrl = {13'b0, IDEXrf_wr_en, IDEXrf_wr_sel, 2'b0, IDEXdm_rd_ctrl, IDEXdm_wr_ctrl,
2'b0, IDEXdo_jump, IDEXdo_branch, 2'b0, alu_a_sel, alu_b_sel, 2'b0, alu_ctrl};

assign y = EXMEMalu_out;
assign bm = EXMEMrf_rd1;
assign rdm = EXMEMRd;
assign ctrlm = {18'b0, EXMEMdm_rd_ctrl, EXMEMdm_wr_ctrl, 12'b0};

assign ym = MEMWBalu_out;
assign mdr = MEMWBdm_dout;
assign rdw = MEMWBRd;
assign ctrlw = {29'b0, MEMWBrf_wr_en, MEMWBrf_wr_sel}; //这里只有两个控制信号

program_counter program_counter(clk, rst, branch, alu_out, PCWrite, pc, pcin, pc_plus4);

IFID IFID(clk, pc_in, inst, IFIDWrite, IFIDpc, IFIDinst, dFlush);

register_file register_file(.clk (clk), .rst (rst), .ra0 (IFIDinst[19:15]), .ra1
(IFIDinst[24:20]), .ra2 (m_rf_addr[4:0]), .wa (MEMWBRd), .wd (rf_wd), .we
(MEMWBrf_wr_en), .rd0 (rf_rd0), .rd1 (rf_rd1), .rd2 (rf_data));

imm imm(.inst (IFIDinst), .imm_out (imm_out));

controller controller(IFIDinst, rf_wr_en, alu_a_sel, alu_b_sel, alu_ctrl, dm_rd_ctrl,
dm_wr_ctrl, rf_wr_sel, comp_ctrl, do_branch, do_jump);

IDEX IDEX(clk, IFIDpc, IFIDinst[11:7], imm_out, rf_rd0, rf_rd1, rf_wr_en_hd, alu_a_sel,
alu_b_sel, alu_ctrl, dm_rd_ctrl, dm_wr_ctrl_hd, rf_wr_sel, comp_ctrl, do_branch, do_jump,
IDEXpc,
IDEXRd, IDEXImm, IDEXrf_rd0, IDEXrf_rd1, IDEXrf_wr_en, IDEXalu_a_sel, IDEXalu_b_sel,
IDEXalu_ctrl, IDEXdm_rd_ctrl, IDEXdm_wr_ctrl, IDEXrf_wr_sel, IDEXcomp_ctrl,
IDEXdo_branch, IDEXdo_jump,
IFIDinst[19:15], IFIDinst[24:20], eFlush, IDEXRs1, IDEXRs2);

alu alu(alu_a, alu_b, IDEXalu_ctrl, alu_out);

```

```
195 br br(.a (IDEXrf_rd0_fd), .b (IDEXrf_rd1_fd), .comp_ctrl (IDEXcomp_ctrl), .do_branch
    (IDEXdo_branch), .do_jump (IDEXdo_jump), .branch (branch));
196
197 EXMEM EXMEM(clk, alu_out, IDEXrf_rd1, IDEXRd, IDEXpc, IDEXrf_wr_en, IDEXdm_rd_ctrl,
    IDEXdm_wr_ctrl, IDEXrf_wr_sel, EXMEMalu_out, EXMEMrf_rd1, EXMEMRd, EXMEMpc,
    EXMEMrf_wr_en, EXMEMdm_rd_ctrl, EXMEMdm_wr_ctrl, EXMEMrf_wr_sel);
198
199 ins_mem ins_mem(.a (pc_in[9:2]), .spo (inst));
200
201 data_mem data_mem(.a (EXMEMalu_out[9:2]), .d (EXMEMrf_rd1), .dpra (m_rf_addr), .clk
    (clk), .we (dm_wr_ctrl_aft), .spo (dm_dout), .dpo (m_data));
202
203 MEMWB MEMWB(clk, EXMEMrf_wr_en, EXMEMrf_wr_sel, dm_dout, EXMEMalu_out, EXMEMRd, EXMEMpc,
    MEMWBrf_wr_en, MEMWBrf_wr_sel, MEMWBdm_dout, MEMWBalu_out, MEMWBRd, MEMWBpc);
204
205 //the last mux
206 always@(*) begin
207     case (MEMWBrf_wr_sel)
208         2'b00: rf_wd = 32'h0;
209         2'b01: rf_wd = MEMWBpc + 32'h4;
210         2'b10: rf_wd = MEMWBalu_out;
211         2'b11: rf_wd = MEMWBdm_dout;  //2'b11: rf_wd = dm_dout_aft;
212         default: rf_wd = 32'h0;
213     endcase
214 end
215
216 endmodule
```

仿真结果：



pcd清零后跳转到 `0x3018` 结果正确。



Step 2.4 : CPU Debug及IO-bus的完善和烧写测试

cpu_pl.v

```
module cpu_pl(
    input clk,
    input rst,

    //io_bus
    output [31:0] io_addr,   //led和seg的地址
    output [31:0] io_dout,  //输出led和seg的数据
    output io_we,   //输出led和seg数据时的使能信号
    input [31:0] io_din,    //输出led和seg数据时的使能信号

    //debug_bus
    input [7:0] m_rf_addr,  //存储器（mem）或寄存器堆（rf）的调试读口地址
    output [31:0] rf_data,  //从rf读取的数据
    output [31:0] m_data,   //从mem读取的数据

    //PC/IF/ID
    output [31:0] pc_out,
    output [31:0] pcd,
    output [31:0] ir,
    output [31:0] pcin,

    //ID/EX
    output [31:0] pce,
    output [31:0] a,
    output [31:0] b,
    output [31:0] imm_debug,
    output [4:0] rd,
    output [31:0] ctrl,

    //EX/MEM
    output [31:0] y,
    output [31:0] bm,
    output [4:0] rdm,
    output [31:0] ctrlm,

    //MEM/WB
    output [31:0] yw,
    output [31:0] mdr,
    output [4:0] rdw,
    output [31:0] ctrlw
);

wire branch;
wire [31:0] pc;
wire [31:0] pc_in;
wire [31:0] pc_plus4;
wire [31:0] inst;
```

```
wire [31:0] imm_out;
wire rf_wr_en;
wire alu_a_sel;
wire alu_b_sel;
wire [1:0] alu_ctrl;
wire dm_rd_ctrl;
wire dm_wr_ctrl;
wire dm_wr_ctrl_aft;
wire [1:0] rf_wr_sel;
wire [2:0] comp_ctrl;
wire do_branch;
wire do_jump;

reg [31:0] rf_wd;
wire [31:0] rf_rd0, rf_rd1;
wire [31:0] alu_a, alu_b, alu_out;
wire [31:0] dm_dout;
wire [31:0] dm_dout_aft;

//5-stage pipeline
wire [31:0] IFIDpc;
wire [31:0] IFIDinst;

wire [31:0] IDEXpc;
wire [4:0] IDEXRd;
wire [31:0] IDEXImm;
wire [31:0] IDEXrf_rd0;
wire [31:0] IDEXrf_rd1;
wire IDEXrf_wr_en;
wire IDEXalu_a_sel;
wire IDEXalu_b_sel;
wire [1:0] IDEXalu_ctrl;
wire IDEXdm_rd_ctrl;
wire IDEXdm_wr_ctrl;
wire [1:0] IDEXrf_wr_sel;
wire [2:0] IDEXcomp_ctrl;
wire IDEXdo_branch;
wire IDEXdo_jump;

wire [31:0] EXMEMalu_out;
wire [31:0] EXMEMrf_rd1;
wire [4:0] EXMEMRd;
wire EXMEMrf_wr_en;
wire EXMEMdm_rd_ctrl;
wire EXMEMdm_wr_ctrl;
wire [1:0] EXMEMrf_wr_sel;
wire [31:0] EXMEMpc;

wire MEMWBrf_wr_en;
wire [1:0] MEMWBrf_wr_sel;
wire [31:0] MEMWBdm_dout;
wire [31:0] MEMWBalu_out;
```

```
wire [4:0] MEMWBRd;
wire [31:0] MEMWBpc;

//forwarding unit
wire [4:0] IDEXRs1;
wire [4:0] IDEXRs2;
wire [1:0] ForwardA;
wire [1:0] ForwardB;
reg [31:0] IDEXrf_rd0_fd;
reg [31:0] IDEXrf_rd1_fd;

forward forward(IDEXRs1, IDEXRs2, IDEXRd, EXMEMRd, MEMWBRd, EXMEMrf_wr_en, MEMWBrf_wr_en,
ForwardA, ForwardB);

always@(*) begin
    case (ForwardA)
        2'b00: IDEXrf_rd0_fd = IDEXrf_rd0;
        2'b01: IDEXrf_rd0_fd = rf_wd;
        2'b10: IDEXrf_rd0_fd = EXMEMalu_out;
        default: IDEXrf_rd0_fd = IDEXrf_rd0;
    endcase

    case (ForwardB)
        2'b00: IDEXrf_rd1_fd = IDEXrf_rd1;
        2'b01: IDEXrf_rd1_fd = rf_wd;
        2'b10: IDEXrf_rd1_fd = EXMEMalu_out;
        default: IDEXrf_rd1_fd = IDEXrf_rd1;
    endcase
end

//hazard detection
wire PCWrite;
wire IFIDWrite;
wire stallpl;

wire dFlush;
wire eFlush;

hazard_detection hazard_detection(IDEXdm_rd_ctrl, IDEXRd, IFIDinst[19:15],
IFIDinst[24:20], PCWrite, IFIDWrite, stallpl, branch, dFlush, eFlush);

reg rf_wr_en_hd;
reg dm_wr_ctrl_hd;
always@(*) begin
    if(stallpl) rf_wr_en_hd <= 1'b0;
    else rf_wr_en_hd <= rf_wr_en;
    if(stallpl) dm_wr_ctrl_hd <= 1'b0;
    else dm_wr_ctrl_hd <= dm_wr_ctrl;
end

//io_bus
assign dm_wr_ctrl_aft = (~(io_addr[10])) & EXMEMdm_wr_ctrl;  //AND gate
```

```
assign dm_dout_aft = io_addr[10] ? io_din : dm_dout;    //Mux
assign io_dout = rf_wd;
assign io_we = io_addr[10] & EXMEMdm_wr_ctrl;    //AND gate
assign io_addr = EXMEMalu_out;

assign alu_a = IDEXalu_a_sel ? IDEXrf_rd0_fd : IDEXpc;
assign alu_b = IDEXalu_b_sel ? IDEXImm : IDEXrf_rd1_fd;
assign pc_in = (pc >= 32'h3000 && pc <= 32'h33ff) ? pc : 32'h0;

assign pc_out = pc_in;
assign pcd = IFIDpc;
assign ir = IFIDinst;
assign pce = IDEXpc;
assign a = IDEXrf_rd0;
assign b = IDEXrf_rd1;
assign imm_debug = IDEXImm;
assign rd = IDEXRd;

wire fStall;
wire dStall;
assign fStall = !PCWrite;
assign dStall = !IFIDWrite;

assign ctrl = {fStall, dStall, dFlush, eFlush, 2'b0 , ForwardA , 2'b0, ForwardB, 1'b0,
IDEXrf_wr_en, IDEXrf_wr_sel, 2'b0, IDEXdm_rd_ctrl, IDEXdm_wr_ctrl, 2'b0, IDEXdo_jump,
IDEXdo_branch, 2'b0, alu_a_sel, alu_b_sel, 2'b0, alu_ctrl};

assign y = EXMEMalu_out;
assign bm = EXMEMrf_rd1;
assign rdm = EXMEMRd;
assign ctrlm = {30'b0, EXMEMdm_rd_ctrl, EXMEMdm_wr_ctrl};

assign yw = MEMWBalu_out;
assign mdr = MEMWBdm_dout;
assign rdw = MEMWBRd;
assign ctrlw = {29'b0, MEMWBrf_wr_en, MEMWBrf_wr_sel};

program_counter program_counter(clk, rst, branch, alu_out, PCWrite, pc, pcin, pc_plus4);

IFID IFID(clk, pc_in, inst, IFIDWrite, IFIDpc, IFIDinst, dFlush);

register_file register_file(.clk (clk), .rst (rst), .ra0 (IFIDinst[19:15]), .ra1
(IFIDinst[24:20]), .ra2 (m_rf_addr[4:0]), .wa (MEMWBRd), .wd (rf_wd), .we
(MEMWBrf_wr_en), .rd0 (rf_rd0), .rd1 (rf_rd1), .rd2 (rf_data));

imm imm(.inst (IFIDinst), .imm_out (imm_out));

controller controller(IFIDinst, rf_wr_en, alu_a_sel, alu_b_sel, alu_ctrl, dm_rd_ctrl,
dm_wr_ctrl, rf_wr_sel, comp_ctrl, do_branch, do_jump);

```

```
IDEX IDEX(clk, IFIDpc, IFIDinst[11:7], imm_out, rf_rd0, rf_rd1, rf_wr_en_hd, alu_a_sel,
alu_b_sel, alu_ctrl, dm_rd_ctrl, dm_wr_ctrl_hd, rf_wr_sel, comp_ctrl, do_branch, do_jump,
IDEXpc,
IDEXRd, IDEXImm, IDEXrf_rd0, IDEXrf_rd1, IDEXrf_wr_en, IDEXalu_a_sel, IDEXalu_b_sel,
IDEXalu_ctrl, IDEXdm_rd_ctrl, IDEXdm_wr_ctrl, IDEXrf_wr_sel, IDEXcomp_ctrl,
IDEXdo_branch, IDEXdo_jump,
IFIDinst[19:15], IFIDinst[24:20], eFlush, IDEXRs1, IDEXRs2);

alu alu(alu_a, alu_b, IDEXalu_ctrl, alu_out);

br br(.a (IDEXrf_rd0_fd), .b (IDEXrf_rd1_fd), .comp_ctrl (IDEXcomp_ctrl), .do_branch
(IDEXdo_branch), .do_jump (IDEXdo_jump), .branch (branch));

EXMEM EXMEM(clk, alu_out, IDEXrf_rd1, IDEXRd, IDEXpc, IDEXrf_wr_en, IDEXdm_rd_ctrl,
IDEXdm_wr_ctrl, IDEXrf_wr_sel, EXMEMalu_out, EXMEMrf_rd1, EXMEMRd, EXMEMpc,
EXMEMrf_wr_en, EXMEMdm_rd_ctrl, EXMEMdm_wr_ctrl, EXMEMrf_wr_sel);

ins_mem ins_mem(.a (pc_in[9:2]), .spo (inst));

data_mem data_mem(.a (EXMEMalu_out[9:2]), .d (EXMEMrf_rd1), .dpra (m_rf_addr), .clk
(clk), .we (dm_wr_ctrl_aft), .spo (dm_dout), .dpo (m_data));

MEMWB MEMWB(clk, EXMEMrf_wr_en, EXMEMrf_wr_sel, dm_dout_aft, EXMEMalu_out, EXMEMRd,
EXMEMpc, MEMWBrf_wr_en, MEMWBrf_wr_sel, MEMWBdm_dout, MEMWBalu_out, MEMWBRd, MEMWBpc);

//the last mux
always@(*) begin
    case (MEMWBrf_wr_sel)
        2'b00: rf_wd = 32'h0;
        2'b01: rf_wd = MEMWBpc + 32'h4;
        2'b10: rf_wd = MEMWBalu_out;
        2'b11: rf_wd = MEMWBdm_dout;  //2'b11: rf_wd = dm_dout_aft;
        default: rf_wd = 32'h0;
    endcase
end

endmodule
```

pdu.v

已经给出

top.v

```
module top(
    input clk,
    input rst,

    //选择CPU工作方式
    input run,
```

```
    input step,

    //输入switch的端口
    input valid,
    input [4:0] in,

    //输出led和seg的端口
    output [1:0] check,  //led6-5:查看类型
    output [4:0] out0,    //led4-0
    output [2:0] an,     //8个数码管
    output [3:0] seg,
    output ready          //led7

);
wire clk_cpu;

wire [7:0] io_addr;
wire [31:0] io_dout;
wire io_we;
wire [31:0] io_din;

wire [7:0] m_rf_addr;
wire [31:0] rf_data;
wire [31:0] m_data;
wire [31:0] pc_out;
wire [31:0] pcd;
wire [31:0] ir;
wire [31:0] pcin;
wire [31:0] pce;
wire [31:0] a;
wire [31:0] b;
wire [31:0] imm_debug;
wire [4:0] rd;
wire [31:0] ctrl;
wire [31:0] y;
wire [31:0] bm;
wire [4:0] rdm;
wire [31:0] ctrlm;
wire [31:0] yw;
wire [31:0] mdr;
wire [4:0] rdw;
wire [31:0] ctrlw;

pdu pdu(clk, rst, run, step, clk_cpu, valid, in, check, out0, an, seg, ready, io_addr,
io_dout, io_we, io_din, m_rf_addr, rf_data, m_data,
pcin, pc_out, pcd, pce, ir, imm, mdr, a, b, y, bm, yw, rd, rdm, rdw, ctrl, ctrlm, ctrlw);
cpu_pl cpu_pl(clk_cpu, rst, io_addr, io_dout, io_we, io_din, m_rf_addr, rf_data, m_data,
pc_out,
pcd, ir, pcin, pce, a, b, imm_debug, rd, ctrl, y, bm, rdm, ctrlm, yw, mdr, rdw, ctrlw);

endmodule
```

hazard_test.asm

```
start:
sw x0, 0x408(x0)    #out1=0

#test data hazards
addi x1, x0, 1  #x1=1
addi x2, x1, 1  #x2=2
add x3, x1, x2  #x3=3
add x4, x1, x3  #x4=4
add x5, x1, x4  #x5=5
sw x5, 0x408(x0) #out1=5

add x6, x1, x2  #x6=3
add x6, x6, x3  #x6=6
add x6, x6, x4  #x6=10
add x6, x6, x5  #x6=15
sw x6, 0x408(x0)    #out1=15

#test load-use hazard
lw x7, 0x40C(x0) #x7=in
addi x8, x7, 1   #x8=in+1
addi x9, x8, -1  #x9=in
sw x9, 0x408(x0)    #out1=in

#test control hazard
beq x9, x0, start  #if (in==0) start
add x10, x9, x5
add x10, x10, x6
sw x10, 0x408(x0)    #out1=in+20
stop: jal x0, stop

#do not execute
add x11, x9, x10
add x12, x10, x11
add x13, x11, x12
```

fib_test.asm

```
.text
    addi x1, x0, 1    #x1=1
    add t1, x0, x0    #store fib series @t1

#### input f0
    sw x1, 0x404(x0)  #rdy=1
l1:
    lw t0, 0x410(x0)  #wait vld=1
#    addi a7, x0, 5    #for debug begin
#    ecall
#    mv t0, a0         #for debug end

```

```
    beq t0, x0, l1
    lw s0, 0x40c(x0)  #s0=vin
#    addi a7, x0, 5    #for debug begin
#    ecall
#    mv s0, a0         #for debug end

    sw s0, 0x408(x0)  #out1=f0
    sw s0, 0(t1)      #store f0
    addi t1, t1, 4

    sw x0, 0x404(x0)  #rdy=0
l2:
    lw t0, 0x410(x0)  #wait vld=0
#    addi a7, x0, 5    #for debug begin
#    ecall
#    mv t0, a0         #for debug end

    beq t0, x1, l2

#### input f1
    sw x1, 0x404(x0)  #rdy=1
l3:
    lw t0, 0x410(x0)  #wait vld=1
#    addi a7, x0, 5    #for debug begin
#    ecall
#    mv t0, a0         #for debug end

    beq t0, x0, l3
    lw s1, 0x40c(x0)  #s1=vin
#    addi a7, x0, 5    #for debug begin
#    ecall
#    mv s1, a0         #for debug end

    sw s1, 0x408(x0)  #out1=f1
    sw s1, 0(t1)      #store f1
    addi t1, t1, 4

    sw x0, 0x404(x0)  #rdy=0
l4:
    lw t0, 0x410(x0)  #wait vld=0
#    addi a7, x0, 5    #for debug begin
#    ecall
#    mv t0, a0         #for debug end

    beq t0, x1, l4

#### comput fi = fi-2 + fi-1
next:
    add t0, s0, s1    #fi
    sw t0, 0x408(x0)  #out1=fi
    sw t0, 0(t1)      #store fi
    addi t1, t1, 4
```

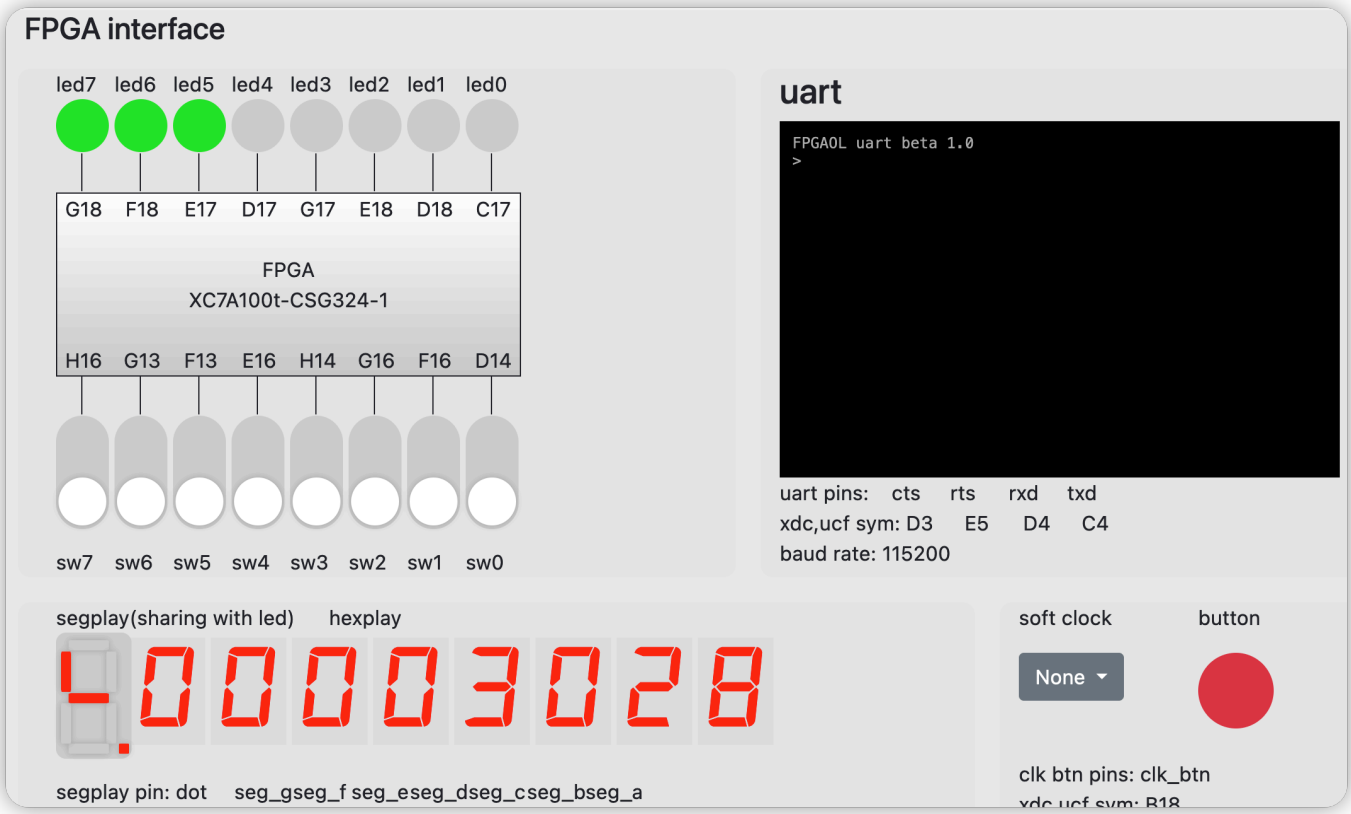
```
65
66      add s0, x0, s1
67      add s1, x0, t0
68
69      sw x1, 0x404(x0)  #rdy=1
70  l5:
71      lw t0, 0x410(x0)  #wait vld=1
72  #    addi a7, x0, 5    #for debug begin
73  #    ecall
74  #    mv t0, a0         #for debug end
75
76      beq t0, x0, l5
77      sw x0, 0x404(x0)  #rdy=0
78  l6:
79      lw t0, 0x410(x0)  #wait vld=0
80  #    addi a7, x0, 5    #for debug begin
81  #    ecall
82  #    mv t0, a0         #for debug end
83
84      beq t0, x1, l6
85      jal x0, next
```

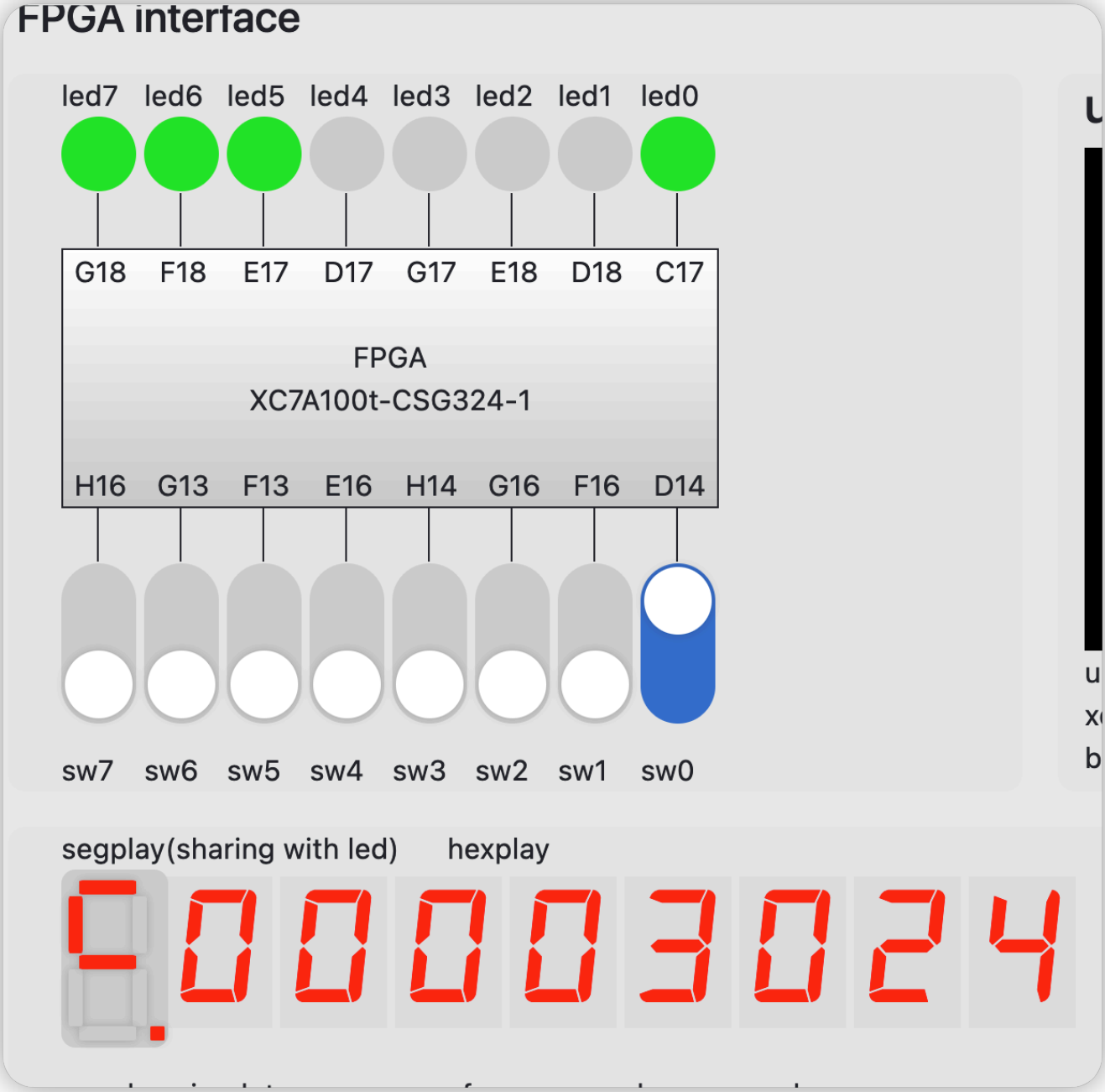
PC.v

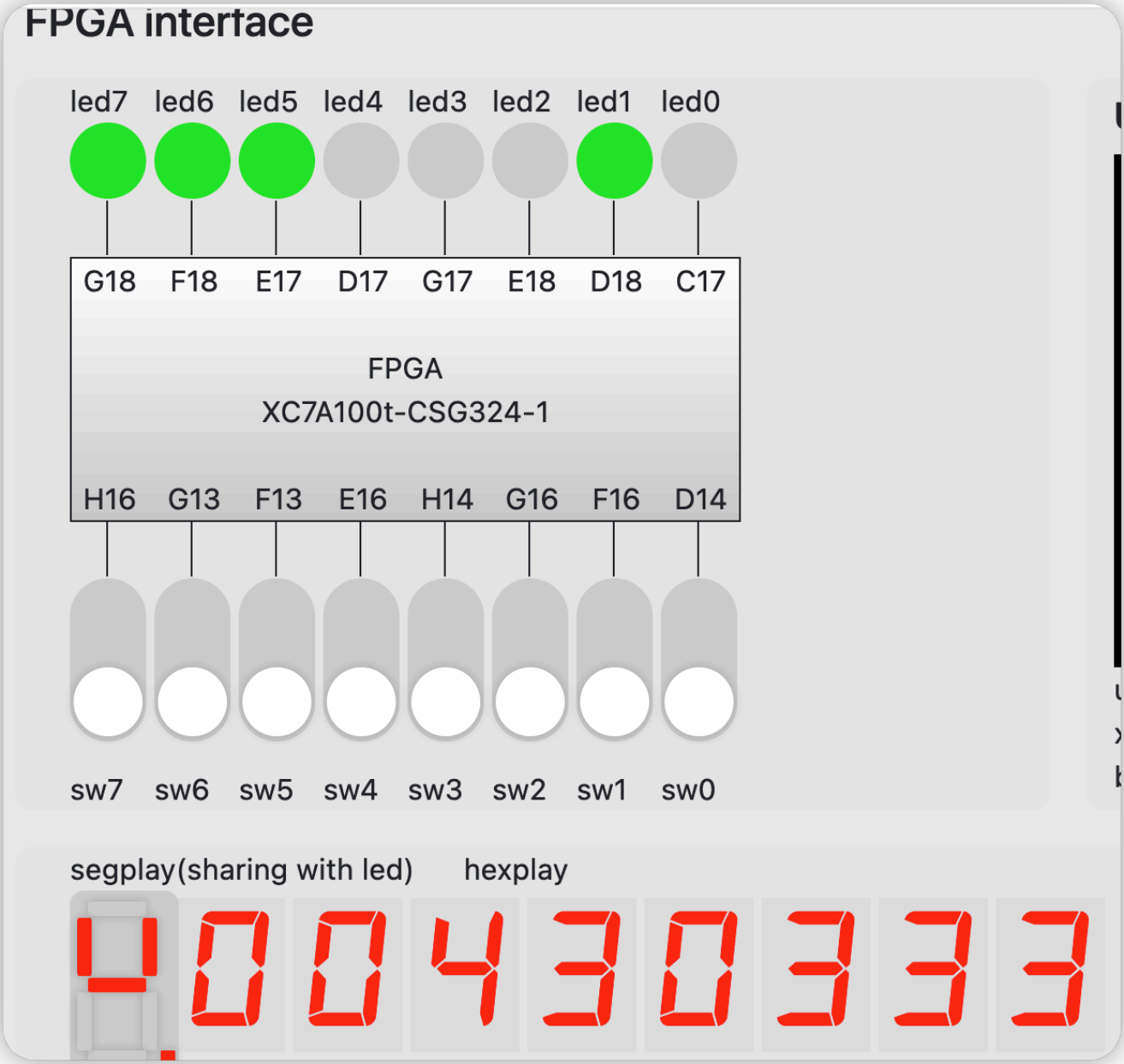
```
1  module program_counter(
2      input clk,
3      input rst,
4      input br,
5      input [31:0] alu_out,
6      input PCWrite,
7
8      output reg [31:0] pc,
9      output reg [31:0] pcin,
10     output [31:0] pc_plus4
11 );
12
13
14 always@(*) begin
15     if(!PCWrite) ;
16     else if(br) pcin <= alu_out;
17     else pcin <= pc_plus4;
18 end
19
20 always@(posedge clk or posedge rst) begin
21     if(rst) pc <= 32'h3000;
22     else pc <= pcin;
23 end
24
25 // always@(posedge clk or posedge rst) begin
26 //     if(rst) pc <= 32'h3000;
27 //     else if(!PCWrite) pc <= pc;
```

```
28  //     else if(br) pc <= alu_out;
29  //     else pc <= pc_plus4;
30  //     pcin <= pc;
31  // end
32
33  assign pc_plus4 = pc + 32'h4;
34
35  endmodule
```

烧写测试：




FPGA interface
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
4 0 0 0 0 3 0 2 4



以上为IFIDPC流水线寄存器在PC为0x3028时的结果，与演示视频一致。其他寄存器类似故省略。

检测beq不发生同理 与演示视频一致。

实验结果

见实验过程。

心得体会

本次实验需在实验4的基础上增加**四个流水线寄存器**、**forward模块**和**hazard_detection模块**。实验过程中有许多细节需要注意，比如io_bus的端口应仔细斟酌。同时，实验也没有给出完整的数据通路（jal数据通路），需要根据我们的理解接线。最后，将整个工程模块化可以加快设计cpu的速度、提高debug的效率。