

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

```
1 module register_file (  
2     input clk, rst,  
3     input [4:0] ra0,    //读端口0地址  
4     output reg [31:0] rd0, //读端口0数据  
5  
6     input [4:0] ra1,    //读端口1地址  
7     output reg [31:0] rd1, //读端口1数据  
8 )
```

```

9      input [4:0] ra2,    //读端口2地址
10     output reg [31:0] rd2, //读端口2数据
11
12     input [4:0] wa, //写端口地址
13     input we,    //写使能 高电平有效
14     input [31:0] wd //写端口数据
15 );
16
17 reg [31:0] regfile [0:31];
18
19 always@(*) begin
20     if(ra0 == 0) rd0 <= 32'h0;
21     else if(we && (wa == ra0)) rd0 <= wd;
22     else rd0 <= regfile[ra0];
23
24     if(ra1 == 0) rd1 <= 32'h0;
25     else if(we && (wa == ra1)) rd1 <= wd;
26     else rd1 <= regfile[ra1];
27
28     if(ra2 == 0) rd2 <= 32'h0;
29     else if(we && (wa == ra2)) rd2 <= wd;
30     else rd2 <= regfile[ra2];
31 end
32
33 integer i;
34
35 always @(posedge clk) begin
36     if(rst) begin
37         for(i = 0; i <= 31; i = i + 1) begin
38             regfile[i] <= 32'b0;
39         end
40     end
41     else if(we) begin
42         if(wa == 0) regfile[wa] <= 0;    //r0内容恒为0
43         else regfile[wa] <= wd;
44     end
45 end
46
47 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）

```

1  .data
2  out: .word 0xff
3  in: .word 0xff
4
5  .text
6  la a0, out
7  addi x5, x0, 0xf0 #addi
8  lw x5, 4(x0) #lw
9
10 li x6, 0xf
11 li x7, 0xf0
12 add x8, x6, x7 #add
13 sub x9, x6, x7 #sub

```

生成.coe文件并初始化 `ins.mem`

Rars运行结果如下：

Text Segment					Name	Number	Value
<input type="checkbox"/>	0x00003000	0xffffd517 auipc x10,0xfffffff	6: la a0, out		zero	0	0x00000000
<input type="checkbox"/>	0x00003004	0x00050513 addi x10,x10,0			ra	1	0x00000000
<input type="checkbox"/>	0x00003008	0xf000293 addi x5,x0,0x000000f0	7: addi x5, x0, 0xf0 #addi		sp	2	0x00002ffc
<input type="checkbox"/>	0x0000300c	0x00402283 lw x5,4(x0)	8: lw x5, 4(x0) #lw		gp	3	0x00001800
<input type="checkbox"/>	0x00003010	0x00f00313 addi x6,x0,15	10: li x6, 0xf		tp	4	0x00000000
<input type="checkbox"/>	0x00003014	0xf000393 addi x7,x0,0x000000f0	11: li x7, 0xf0		t0	5	0x000000ff
<input type="checkbox"/>	0x00003018	0x00730433 add x8,x6,x7	12: add x8, x6, x7 #add		t1	6	0x0000000f
<input type="checkbox"/>	0x0000301c	0x407304b3 sub x9,x6,x7	13: sub x9, x6, x7 #sub		t2	7	0x000000f0
					s0	8	0x000000ff
					s1	9	0xfffffff1
					a0	10	0x00000000
					a1	11	0x00000000

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

```

1  module IFID(
2      input clk,
3
4      input [31:0] pc_in,
5      input [31:0] inst,
6      output reg [31:0] IFIDpc,
7      output reg [31:0] IFIDinst
8  );
9
10 always@(posedge clk) begin
11     IFIDpc <= pc_in;
12     IFIDinst <= inst;
13 end
14
15 endmodule

```

IDEX.v

```
1  module IDEX(  
2      input clk,  
3  
4      input [31:0] IFIDpc,  
5      input [4:0] Rd, //also part of forwarding unit  
6      input [31:0] Imm,  
7      input [31:0] rf_rd0,  
8      input [31:0] rf_rd1,  
9  
10     //控制信号  
11     input rf_wr_en, //WB  
12     input alu_a_sel, alu_b_sel, //EX  
13     input [1:0] alu_ctrl, //EX  
14     input dm_rd_ctrl, //MEM  
15     input dm_wr_ctrl, //MEM  
16     input [1:0] rf_wr_sel, //WB  
17  
18     input [2:0] comp_ctrl, //EX  
19     input do_branch, do_jump, //EX  
20  
21     output reg [31:0] IDEXpc,  
22     output reg [4:0] IDEXRd,  
23     output reg [31:0] IDEXImm,  
24     output reg [31:0] IDEXrf_rd0,  
25     output reg [31:0] IDEXrf_rd1,  
26  
27     output reg IDEXrf_wr_en, reg IDEXalu_a_sel, reg IDEXalu_b_sel,  
28     output reg [1:0] IDEXalu_ctrl,  
29     output reg IDEXdm_rd_ctrl,  
30     output reg IDEXdm_wr_ctrl,  
31     output reg [1:0] IDEXrf_wr_sel,  
32     output reg [2:0] IDEXcomp_ctrl,  
33     output reg IDEXdo_branch, reg IDEXdo_jump,  
34  
35     //forwarding unit  
36     input [4:0] IFIDRs1,  
37     input [4:0] IFIDRs2,  
38  
39     output reg [4:0] IDEXRs1,  
40     output reg [4:0] IDEXRs2  
41 );  
42  
43 always@(posedge clk) begin  
44     IDEXpc <= IFIDpc;  
45     IDEXRd <= Rd;  
46     IDEXImm <= Imm;  
47     IDEXrf_rd0 <= rf_rd0;  
48     IDEXrf_rd1 <= rf_rd1;  
49  
50     IDEXrf_wr_en <= rf_wr_en;
```

```

51     IDEXalu_a_sel <= alu_a_sel;
52     IDEXalu_b_sel <= alu_b_sel;
53     IDEXalu_ctrl <= alu_ctrl;
54     IDEXdm_rd_ctrl <= dm_rd_ctrl;
55     IDEXdm_wr_ctrl <= dm_wr_ctrl;
56     IDEXrf_wr_sel <= rf_wr_sel;
57     IDEXcomp_ctrl <= comp_ctrl;
58     IDEXdo_branch <= do_branch;
59     IDEXdo_jump <= do_jump;
60
61     IDEXRs1 <= IFIDRs1;
62     IDEXRs2 <= IFIDRs2;
63 end
64
65 endmodule

```

EXMEM.v

```

1  module EXMEM (
2      input clk,
3
4      input [31:0] alu_out,
5      input [31:0] IDEXrf_rd1,
6      input [4:0] IDEXRd,
7      input [31:0] IDEXpc,
8
9      input IDEXrf_wr_en,
10     input IDEXdm_rd_ctrl,
11     input IDEXdm_wr_ctrl,
12     input [1:0] IDEXrf_wr_sel,
13
14     output reg [31:0] EXMEMalu_out,
15     output reg [31:0] EXMEMrf_rd1,
16     output reg [4:0] EXMEMRd,
17     output reg [31:0] EXMEMpc,
18
19     output reg EXMEMrf_wr_en,
20     output reg EXMEMdm_rd_ctrl, //MEM
21     output reg EXMEMdm_wr_ctrl, //MEM
22     output reg [1:0] EXMEMrf_wr_sel
23 );
24
25 always@(posedge clk) begin
26     EXMEMalu_out <= alu_out;
27     EXMEMrf_rd1 <= IDEXrf_rd1;
28     EXMEMRd <= IDEXRd;
29     EXMEMpc <= IDEXpc;
30
31     EXMEMrf_wr_en <= IDEXrf_wr_en;
32     EXMEMdm_rd_ctrl <= IDEXdm_rd_ctrl;
33     EXMEMdm_wr_ctrl <= IDEXdm_wr_ctrl;

```

```

34     EXMEMrf_wr_sel <= IDEXrf_wr_sel;
35 end
36
37 endmodule

```

MEMWB.v

```

1  module MEMWB(
2      input clk,
3
4      input EXMEMrf_wr_en,
5      input [1:0] EXMEMrf_wr_sel,
6      input [31:0] dm_dout,
7      input [31:0] EXMEMalu_out,
8      input [4:0] EXMEMRd,
9      input [31:0] EXMEMpc,
10
11     output reg MEMWBrf_wr_en,
12     output reg [1:0] MEMWBrf_wr_sel,
13     output reg [31:0] MEMWBdm_dout,
14     output reg [31:0] MEMWBalu_out,
15     output reg [4:0] MEMWBRd,
16     output reg [31:0] MEMWBpc
17 );
18
19 always@(posedge clk) begin
20     MEMWBrf_wr_en <= EXMEMrf_wr_en;
21     MEMWBrf_wr_sel <= EXMEMrf_wr_sel;
22     MEMWBdm_dout <= dm_dout;
23     MEMWBalu_out <= EXMEMalu_out;
24     MEMWBRd <= EXMEMRd;
25     MEMWBpc <= EXMEMpc;
26 end
27
28 endmodule

```

cpu.v(仅支持数据前递的非最终版本)

```

1  module cpu_pl(
2      input clk,
3      input rst,
4
5      //io_bus
6      output [31:0] io_addr,    //led和seg的地址
7      output [31:0] io_dout,    //输出led和seg的数据
8      output io_we,            //输出led和seg数据时的使能信号
9      input [31:0] io_din,      //输出led和seg数据时的使能信号
10
11     //debug_bus
12     input [7:0] m_rf_addr,     //存储器 (mem) 或寄存器堆 (rf) 的调试读口地址

```

```

13     output [31:0] rf_data, //从rf读取的数据
14     output [31:0] m_data, //从mem读取的数据
15
16     //PC/IF/ID
17     output [31:0] pc_out,
18     output [31:0] pcd,
19     output [31:0] ir,
20     output [31:0] pcin,
21
22     //ID/EX
23     output [31:0] pce,
24     output [31:0] a,
25     output [31:0] b,
26     output [31:0] imm_debug,
27     output [4:0] rd,
28     output [31:0] ctrl,
29
30     //EX/MEM
31     output [31:0] y,
32     output [31:0] bm,
33     output [4:0] rdm,
34     output [31:0] ctrlm,
35
36     //MEM/WB
37     output [31:0] ym,
38     output [31:0] mdr,
39     output [4:0] rdw,
40     output [31:0] ctrlw
41 );
42
43 wire branch;
44 wire [31:0] pc;
45 wire [31:0] pc_in;
46 wire [31:0] pc_plus4;
47 wire [31:0] inst;
48 wire [31:0] imm_out;
49 wire rf_wr_en;
50 wire alu_a_sel;
51 wire alu_b_sel;
52 wire [1:0] alu_ctrl;
53 wire dm_rd_ctrl;
54 wire dm_wr_ctrl;
55 wire dm_wr_ctrl_aft;
56 wire [1:0] rf_wr_sel;
57 wire [2:0] comp_ctrl;
58 wire do_branch;
59 wire do_jump;
60
61 reg [31:0] rf_wd;
62 wire [31:0] rf_rd0, rf_rd1;
63 wire [31:0] alu_a, alu_b, alu_out;
64 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 IEXdo_branch;
85 wire IEXdo_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;

```



```

116     2'b01: IDEXrf_rd0_fd = rf_wd;
117     2'b10: IDEXrf_rd0_fd = EXMEMalu_out;
118     default: IDEXrf_rd0_fd = IDEXrf_rd0;
119 endcase
120
121 case (ForwardB)
122     2'b00: IDEXrf_rd1_fd = IDEXrf_rd1;
123     2'b01: IDEXrf_rd1_fd = rf_wd;
124     2'b10: IDEXrf_rd1_fd = EXMEMalu_out;
125     default: IDEXrf_rd1_fd = IDEXrf_rd1;
126 endcase
127 end
128
129 //io_bus
130 assign dm_wr_ctrl_aft = (~(io_addr[10])) & EXMEMdm_wr_ctrl; //AND gate
131 assign dm_dout_aft = io_addr[10] ? io_din : dm_dout; //Mux
132 assign io_dout = rf_rd1;
133 assign io_we = io_addr[10] & dm_wr_ctrl; //AND gate
134 assign io_addr = alu_out;
135
136 assign alu_a = IDEXalu_a_sel ? IDEXrf_rd0_fd : IDEXpc;
137 assign alu_b = IDEXalu_b_sel ? IDEXImm : IDEXrf_rd1_fd;
138 assign pc_in = (pc >= 32'h3000 && pc <= 32'h33ff) ? pc : 32'h0;
139
140 assign pc_out = pc_in;
141 assign pcd = IFIDpc;
142 assign ir = IFIDinst;
143 assign pce = IDEXpc;
144 assign a = IDEXrf_rd0;
145 assign b = IDEXrf_rd1;
146 assign imm_debug = IDEXImm;
147 assign rd = IDEXRd;
148 assign ctrl = {13'b0, IDEXrf_wr_en, IDEXrf_wr_sel, 2'b0, IDEXdm_rd_ctrl, IDEXdm_wr_ctrl,
149 2'b0, IDEXdo_jump, IDEXdo_branch, 2'b0, alu_a_sel, alu_b_sel, 2'b0, alu_ctrl};
150
151 assign y = EXMEMalu_out;
152 assign bm = EXMEMrf_rd1;
153 assign rdm = EXMEMRd;
154 assign ctrlm = {18'b0, EXMEMdm_rd_ctrl, EXMEMdm_wr_ctrl, 12'b0};
155
156 assign ym = MEMWBalu_out;
157 assign mdr = MEMWBdm_dout;
158 assign rdw = MEMWBRd;
159 assign ctrlw = {29'b0, MEMWBrf_wr_en, MEMWBrf_wr_sel}; //这里只有两个控制信号
160
161 program_counter program_counter(clk, rst, branch, alu_out, pc, pcin, pc_plus4);
162
163 IFID IFID(clk, pc_in, inst, IFIDpc, IFIDinst);
164
165 register_file register_file(.clk (clk), .rst (rst), .ra0 (IFIDinst[19:15]), .ra1
166 (IFIDinst[24:20]), .ra2 (m_rf_addr[4:0]), .wa (MEMWBRd), .wd (rf_wd), .we
167 (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;

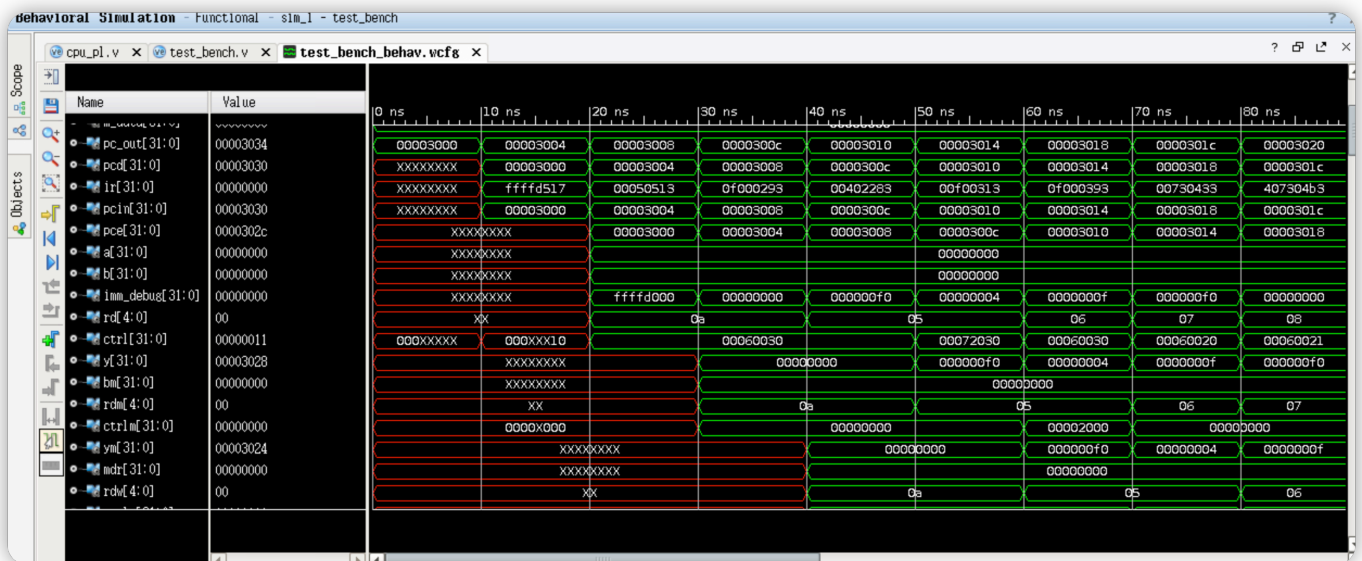
```

```

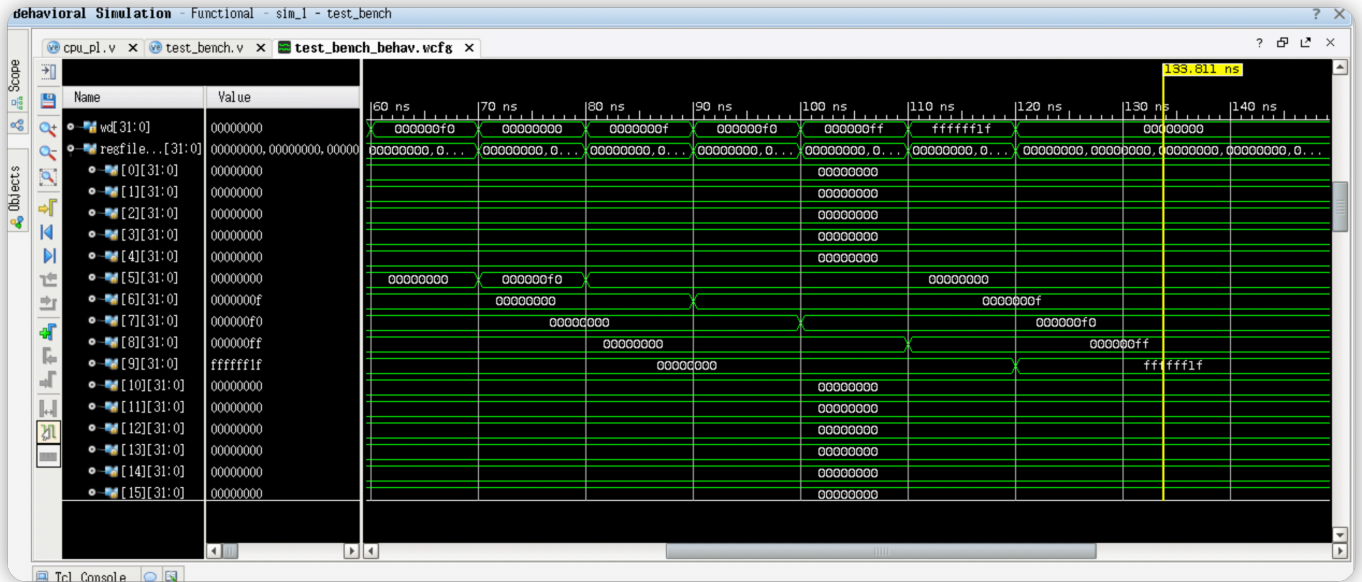
37
38 cpu_pl cpu_pl(clk, rst, io_addr, io_dout, io_we, io_din, m_rf_addr, rf_data, m_data,
pc_out, pcd, ir, pcin, pce, a, b, imm_debug, rd, ctrl, y, bm, rdm, ctrlm, ym, mdr, rdw,
ctrlw);
39
40 initial begin
41     rst = 1; m_rf_addr = 0;
42     clk = 1;#5
43     clk = 0;
44     rst = 0;
45
46     forever #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 **0xffffffff1f** 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）：

```

1  .data
2  out: .word 0xf00
3  in: .word 0xf00
4
5  .text
6  li x6, 0xf
7  li x7, 0xf0
8  add x8, x6, x7 #add
9  sub x9, x6, x7 #sub
10 lw x5, 4(x0)
11 add x5, x8, x5
12 sub x5, x9, x5

```

生成.coe文件并初始化 **data_mem** 和 **ins_mem**

hazard_detection.v

```

1 module hazard_detection(
2     input IDExdm_rd_ctrl,
3     input [4:0] IDExRd,
4     input [4:0] IFIDRs1,
5     input [4:0] IFIDRs2,

```

```

6     output PCWrite,
7     output IFIDWrite,
8     output stallpl
9 );
10
11 assign PCWrite = (IDEXdm_rd_ctrl && ((IDEXRd == IFIDRs1) || (IDEXRd == IFIDRs2))) ? 1'b0 :
12 1'b1;
13 assign IFIDWrite = (IDEXdm_rd_ctrl && ((IDEXRd == IFIDRs1) || (IDEXRd == IFIDRs2))) ? 1'b0
14 : 1'b1;
15 assign stallpl = (IDEXdm_rd_ctrl && ((IDEXRd == IFIDRs1) || (IDEXRd == IFIDRs2))) ? 1'b1 :
16 1'b0;
17
18 endmodule

```

pc.v

做如下更改：

```

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 always@(posedge clk or posedge rst) begin
14     if(rst) pc <= 32'h3000;
15     else if(!PCWrite) pc <= pc;
16     else if(br) pc <= alu_out;
17     else pc <= pc_plus4;
18
19     pcin <= pc;
20 end
21
22 assign pc_plus4 = pc + 32'h4;
23
24 endmodule

```

IFID.v

做如下更改：

```

1 module IFID(
2     input clk,
3
4     input [31:0] pc_in,

```

```

5     input [31:0] inst,
6     input IFIDWrite,
7
8     output reg [31:0] IFIDpc,
9     output reg [31:0] IFIDinst
10 );
11
12 always@(posedge clk) begin
13     if(!IFIDWrite) begin
14         IFIDpc <= IFIDpc;
15         IFIDinst <= IFIDinst;
16     end
17     else begin
18         IFIDpc <= pc_in;
19         IFIDinst <= inst;
20     end
21 end
22
23 endmodule

```

cpu_pl.v

做如下更改：

```

1 module cpu_pl(
2     input clk,
3     input rst,
4
5     //io_bus
6     output [31:0] io_addr,    //led和seg的地址
7     output [31:0] io_dout,    //输出led和seg的数据
8     output io_we,            //输出led和seg数据时的使能信号
9     input [31:0] io_din,      //输出led和seg数据时的使能信号
10
11     //debug_bus
12     input [7:0] m_rf_addr,    //存储器 (mem) 或寄存器堆 (rf) 的调试读口地址
13     output [31:0] rf_data,    //从rf读取的数据
14     output [31:0] m_data,     //从mem读取的数据
15
16     //PC/IF/ID
17     output [31:0] pc_out,
18     output [31:0] pcd,
19     output [31:0] ir,
20     output [31:0] pcin,
21
22     //ID/EX
23     output [31:0] pce,
24     output [31:0] a,
25     output [31:0] b,
26     output [31:0] imm_debug,
27     output [4:0] rd,

```

```

28     output [31:0] ctrl,
29
30     //EX/MEM
31     output [31:0] y,
32     output [31:0] bm,
33     output [4:0] rdm,
34     output [31:0] ctrlm,
35
36     //MEM/WB
37     output [31:0] ym,
38     output [31:0] mdr,
39     output [4:0] rdw,
40     output [31:0] ctrlw
41 );
42
43 wire branch;
44 wire [31:0] pc;
45 wire [31:0] pc_in;
46 wire [31:0] pc_plus4;
47 wire [31:0] inst;
48 wire [31:0] imm_out;
49 wire rf_wr_en;
50 wire alu_a_sel;
51 wire alu_b_sel;
52 wire [1:0] alu_ctrl;
53 wire dm_rd_ctrl;
54 wire dm_wr_ctrl;
55 wire dm_wr_ctrl_aft;
56 wire [1:0] rf_wr_sel;
57 wire [2:0] comp_ctrl;
58 wire do_branch;
59 wire do_jump;
60
61 reg [31:0] rf_wd;
62 wire [31:0] rf_rd0, rf_rd1;
63 wire [31:0] alu_a, alu_b, alu_out;
64 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;
116         2'b01: IDEXrf_rd0_fd = rf_wd;
117         2'b10: IDEXrf_rd0_fd = EXMEMalu_out;
118         default: IDEXrf_rd0_fd = IDEXrf_rd0;
119     endcase
120
121     case (ForwardB)
122         2'b00: IDEXrf_rd1_fd = IDEXrf_rd1;
123         2'b01: IDEXrf_rd1_fd = rf_wd;
124         2'b10: IDEXrf_rd1_fd = EXMEMalu_out;
125         default: IDEXrf_rd1_fd = IDEXrf_rd1;
126     endcase
127 end
128
129 //hazard detection
130 wire PCWrite;

```

```

131 wire IFIDWrite;
132 wire stallpl;
133
134 hazard_detection hazard_detection(IDEXdm_rd_ctrl, IDEXRd, IFIDinst[19:15],
IFIDinst[24:20], PCWrite, IFIDWrite, stallpl);
135
136 reg rf_wr_en_hd;
137 reg dm_wr_ctrl_hd;
138 always@(*) begin
139     if(stallpl) rf_wr_en_hd <= 1'b0;
140     else rf_wr_en_hd <= rf_wr_en;
141     if(stallpl) dm_wr_ctrl_hd <= 1'b0;
142     else dm_wr_ctrl_hd <= dm_wr_ctrl;
143 end
144
145 //io_bus
146 assign dm_wr_ctrl_aft = (~(io_addr[10])) & EXMEMdm_wr_ctrl; //AND gate
147 assign dm_dout_aft = io_addr[10] ? io_din : dm_dout; //Mux
148 assign io_dout = rf_rd1;
149 assign io_we = io_addr[10] & dm_wr_ctrl; //AND gate
150 assign io_addr = alu_out;
151
152 assign alu_a = IDEXalu_a_sel ? IDEXrf_rd0_fd : IDEXpc;
153 assign alu_b = IDEXalu_b_sel ? IDEXImm : IDEXrf_rd1_fd;
154 assign pc_in = (pc >= 32'h3000 && pc <= 32'h33ff) ? pc : 32'h0;
155
156 assign pc_out = pc_in;
157 assign pcd = IFIDpc;
158 assign ir = IFIDinst;
159 assign pce = IDEXpc;
160 assign a = IDEXrf_rd0;
161 assign b = IDEXrf_rd1;
162 assign imm_debug = IDEXImm;
163 assign rd = IDEXRd;
164 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};
165
166 assign y = EXMEMalu_out;
167 assign bm = EXMEMrf_rd1;
168 assign rdm = EXMEMRd;
169 assign ctrlm = {18'b0, EXMEMdm_rd_ctrl, EXMEMdm_wr_ctrl, 12'b0};
170
171 assign ym = MEMWBalu_out;
172 assign mdr = MEMWBdm_dout;
173 assign rdw = MEMWBRd;
174 assign ctrlw = {29'b0, MEMWBrf_wr_en, MEMWBrf_wr_sel}; //这里只有两个控制信号
175
176 program_counter program_counter(clk, rst, branch, alu_out, PCWrite, pc, pcin, pc_plus4);
177
178 IFID IFID(clk, pc_in, inst, IFIDWrite, IFIDpc, IFIDinst);
179

```

```

180 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));
181
182 imm imm(.inst (IFIDinst), .imm_out (imm_out));
183
184 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);
185
186 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,
187 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,
188 IFIDinst[19:15], IFIDinst[24:20], IDEXRs1, IDEXRs2);
189
190 alu alu(alu_a, alu_b, IDEXalu_ctrl, alu_out);
191
192 br br(.a (IDEXrf_rd0), .b (IDEXrf_rd1), .comp_ctrl (IDEXcomp_ctrl), .do_branch
    (IDEXdo_branch), .do_jump (IDEXdo_jump), .branch (branch));
193
194 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);
195
196 ins_mem ins_mem(.a (pc_in[9:2]), .spo (inst));
197
198 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));
199
200 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);
201
202 //the last mux
203 always@(*) begin
204     case (MEMWBrf_wr_sel)
205         2'b00: rf_wd = 32'h0;
206         2'b01: rf_wd = MEMWBpc + 32'h4;
207         2'b10: rf_wd = MEMWBalu_out;
208         2'b11: rf_wd = MEMWBdm_dout; //2'b11: rf_wd = dm_dout_aft;
209         default: rf_wd = 32'h0;
210     endcase
211 end
212
213 endmodule

```

仿真结果：

hazard_detection.v

```
1 module hazard_detection(  
2     input IDEXdm_rd_ctrl,  
3     input [4:0] IDEXRd,  
4     input [4:0] IFIDRs1,  
5     input [4:0] IFIDRs2,  
6     output PCWrite,  
7     output IFIDWrite,  
8     output stallpl,  
9  
10    input branch,  
11    output dFlush, eFlush  
12 );  
13  
14 assign PCWrite = (IDEXdm_rd_ctrl && ((IDEXRd == IFIDRs1) || (IDEXRd == IFIDRs2))) ? 1'b0 :  
15 1'b1;  
16 assign IFIDWrite = (IDEXdm_rd_ctrl && ((IDEXRd == IFIDRs1) || (IDEXRd == IFIDRs2))) ? 1'b0  
17 : 1'b1;  
18 assign stallpl = (IDEXdm_rd_ctrl && ((IDEXRd == IFIDRs1) || (IDEXRd == IFIDRs2))) ? 1'b1 :  
19 1'b0;  
20 assign dFlush = (branch) ? 1'b1 : 1'b0;  
21 assign eFlush = (branch) ? 1'b1 : 1'b0;  
22  
23 endmodule
```

IFID.v

```
1 module IFID(  
2     input clk,  
3  
4     input [31:0] pc_in,  
5     input [31:0] inst,  
6     input IFIDWrite,  
7  
8     output reg [31:0] IFIDpc,  
9     output reg [31:0] IFIDinst,  
10  
11    //hazard_detection  
12    input dFlush  
13 );  
14  
15 always@(posedge clk) begin  
16     if(dFlush) begin  
17         IFIDpc <= 32'b0;  
18         IFIDinst <= 32'b0;  
19     end  
20     else if(!IFIDWrite) begin  
21         IFIDpc <= IFIDpc;  
22         IFIDinst <= IFIDinst;  
23     end  
24 end
```

```

24     else begin
25         IFIDpc <= pc_in;
26         IFIDinst <= inst;
27     end
28 end
29
30 endmodule

```

IDEX.v

```

1  module IDEX(
2      input clk,
3
4      input [31:0] IFIDpc,
5      input [4:0] Rd, //also part of forwarding unit
6      input [31:0] Imm,
7      input [31:0] rf_rd0,
8      input [31:0] rf_rd1,
9
10     //控制信号
11     input rf_wr_en, //WB
12     input alu_a_sel, alu_b_sel, //EX
13     input [1:0] alu_ctrl, //EX
14     input dm_rd_ctrl, //MEM
15     input dm_wr_ctrl, //MEM
16     input [1:0] rf_wr_sel, //WB
17
18     input [2:0] comp_ctrl, //EX
19     input do_branch, do_jump, //EX
20
21     output reg [31:0] IDEXpc,
22     output reg [4:0] IDEXRd,
23     output reg [31:0] IDEXImm,
24     output reg [31:0] IDEXrf_rd0,
25     output reg [31:0] IDEXrf_rd1,
26
27     output reg IDEXrf_wr_en, reg IDEXalu_a_sel, reg IDEXalu_b_sel,
28     output reg [1:0] IDEXalu_ctrl,
29     output reg IDEXdm_rd_ctrl,
30     output reg IDEXdm_wr_ctrl,
31     output reg [1:0] IDEXrf_wr_sel,
32     output reg [2:0] IDEXcomp_ctrl,
33     output reg IDEXdo_branch, reg IDEXdo_jump,
34
35     //forwarding unit
36     input [4:0] IFIDRs1,
37     input [4:0] IFIDRs2,
38     input eFlush,
39
40     output reg [4:0] IDEXRs1,
41     output reg [4:0] IDEXRs2

```

```

42 );
43
44 always@(posedge clk) begin
45     if(eFlush) begin
46         IDEXpc <= 0;
47         IDEXRd <= 0;
48         IDEXImm <= 0;
49         IDEXrf_rd0 <= 0;
50         IDEXrf_rd1 <= 0;
51
52         IDEXrf_wr_en <= 0;
53         IDEXalu_a_sel <= 0;
54         IDEXalu_b_sel <= 0;
55         IDEXalu_ctrl <= 0;
56         IDEXdm_rd_ctrl <= 0;
57         IDEXdm_wr_ctrl <= 0;
58         IDEXrf_wr_sel <= 0;
59         IDEXcomp_ctrl <= 0;
60         IDEXdo_branch <= 0;
61         IDEXdo_jump <= 0;
62
63         IDEXRs1 <= 0;
64         IDEXRs2 <= 0;
65     end
66     else begin
67         IDEXpc <= IFIDpc;
68         IDEXRd <= Rd;
69         IDEXImm <= Imm;
70         IDEXrf_rd0 <= rf_rd0;
71         IDEXrf_rd1 <= rf_rd1;
72
73         IDEXrf_wr_en <= rf_wr_en;
74         IDEXalu_a_sel <= alu_a_sel;
75         IDEXalu_b_sel <= alu_b_sel;
76         IDEXalu_ctrl <= alu_ctrl;
77         IDEXdm_rd_ctrl <= dm_rd_ctrl;
78         IDEXdm_wr_ctrl <= dm_wr_ctrl;
79         IDEXrf_wr_sel <= rf_wr_sel;
80         IDEXcomp_ctrl <= comp_ctrl;
81         IDEXdo_branch <= do_branch;
82         IDEXdo_jump <= do_jump;
83
84         IDEXRs1 <= IFIDRs1;
85         IDEXRs2 <= IFIDRs2;
86     end
87 end
88
89 endmodule

```

cpu_pl.v

注意对br模块输入端口的更改。

```
1  module cpu_pl(
2      input clk,
3      input rst,
4
5      //io_bus
6      output [31:0] io_addr,    //led和seg的地址
7      output [31:0] io_dout,    //输出led和seg的数据
8      output io_we,            //输出led和seg数据时的使能信号
9      input [31:0] io_din,      //输出led和seg数据时的使能信号
10
11     //debug_bus
12     input [7:0] m_rf_addr,     //存储器 (mem) 或寄存器堆 (rf) 的调试读口地址
13     output [31:0] rf_data,     //从rf读取的数据
14     output [31:0] m_data,      //从mem读取的数据
15
16     //PC/IF/ID
17     output [31:0] pc_out,
18     output [31:0] pcd,
19     output [31:0] ir,
20     output [31:0] pcin,
21
22     //ID/EX
23     output [31:0] pce,
24     output [31:0] a,
25     output [31:0] b,
26     output [31:0] imm_debug,
27     output [4:0] rd,
28     output [31:0] ctrl,
29
30     //EX/MEM
31     output [31:0] y,
32     output [31:0] bm,
33     output [4:0] rdm,
34     output [31:0] ctrlm,
35
36     //MEM/WB
37     output [31:0] ym,
38     output [31:0] mdr,
39     output [4:0] rdw,
40     output [31:0] ctrlw
41 );
42
43 wire branch;
44 wire [31:0] pc;
45 wire [31:0] pc_in;
46 wire [31:0] pc_plus4;
47 wire [31:0] inst;
48 wire [31:0] imm_out;
```



```

49 wire rf_wr_en;
50 wire alu_a_sel;
51 wire alu_b_sel;
52 wire [1:0] alu_ctrl;
53 wire dm_rd_ctrl;
54 wire dm_wr_ctrl;
55 wire dm_wr_ctrl_aft;
56 wire [1:0] rf_wr_sel;
57 wire [2:0] comp_ctrl;
58 wire do_branch;
59 wire do_jump;
60
61 reg [31:0] rf_wd;
62 wire [31:0] rf_rd0, rf_rd1;
63 wire [31:0] alu_a, alu_b, alu_out;
64 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, MEMWBBrf_wr_en,
ForwardA, ForwardB);
112
113 always@(*) begin
114     case (ForwardA)
115         2'b00: IDEXrf_rd0_fd = IDEXrf_rd0;
116         2'b01: IDEXrf_rd0_fd = rf_wd;
117         2'b10: IDEXrf_rd0_fd = EXMEMalu_out;
118         default: IDEXrf_rd0_fd = IDEXrf_rd0;
119     endcase
120
121     case (ForwardB)
122         2'b00: IDEXrf_rd1_fd = IDEXrf_rd1;
123         2'b01: IDEXrf_rd1_fd = rf_wd;
124         2'b10: IDEXrf_rd1_fd = EXMEMalu_out;
125         default: IDEXrf_rd1_fd = IDEXrf_rd1;
126     endcase
127 end
128
129 //hazard detection
130 wire PCWrite;
131 wire IFIDWrite;
132 wire stallpl;
133
134 wire dFlush;
135 wire eFlush;
136
137 hazard_detection hazard_detection(IDEXdm_rd_ctrl, IDEXRd, IFIDInst[19:15],
IFIDInst[24:20], PCWrite, IFIDWrite, stallpl, branch, dFlush, eFlush);
138
139 reg rf_wr_en_hd;
140 reg dm_wr_ctrl_hd;
141 always@(*) begin
142     if(stallpl) rf_wr_en_hd <= 1'b0;
143     else rf_wr_en_hd <= rf_wr_en;
144     if(stallpl) dm_wr_ctrl_hd <= 1'b0;
145     else dm_wr_ctrl_hd <= dm_wr_ctrl;
146 end
147
148 //io_bus
149 assign dm_wr_ctrl_aft = (~(io_addr[10])) & EXMEMdm_wr_ctrl; //AND gate
150 assign dm_dout_aft = io_addr[10] ? io_din : dm_dout; //Mux

```

```

151 assign io_dout = rf_rd1;
152 assign io_we = io_addr[10] & dm_wr_ctrl;    //AND gate
153 assign io_addr = alu_out;
154
155 assign alu_a = IDEXalu_a_sel ? IDEXrf_rd0_fd : IDEXpc;
156 assign alu_b = IDEXalu_b_sel ? IDEXImm : IDEXrf_rd1_fd;
157 assign pc_in = (pc >= 32'h3000 && pc <= 32'h33ff) ? pc : 32'h0;
158
159 assign pc_out = pc_in;
160 assign pcd = IFIDpc;
161 assign ir = IFIDinst;
162 assign pce = IDEXpc;
163 assign a = IDEXrf_rd0;
164 assign b = IDEXrf_rd1;
165 assign imm_debug = IDEXImm;
166 assign rd = IDEXRd;
167 assign ctrl = {13'b0, IDEXrf_wr_en, IDEXrf_wr_sel, 2'b0, IDEXdm_rd_ctrl, IDEXdm_wr_ctrl,
168 2'b0, IDEXdo_jump, IDEXdo_branch, 2'b0, alu_a_sel, alu_b_sel, 2'b0, alu_ctrl};
169
169 assign y = EXMEMalu_out;
170 assign bm = EXMEMrf_rd1;
171 assign rdm = EXMEMRd;
172 assign ctrlm = {18'b0, EXMEMdm_rd_ctrl, EXMEMdm_wr_ctrl, 12'b0};
173
174 assign ym = MEMWBalu_out;
175 assign mdr = MEMWBdm_dout;
176 assign rdw = MEMWBrd;
177 assign ctrlw = {29'b0, MEMWBrf_wr_en, MEMWBrf_wr_sel}; //这里只有两个控制信号
178
179 program_counter program_counter(clk, rst, branch, alu_out, PCWrite, pc, pcin, pc_plus4);
180
181 IFID IFID(clk, pc_in, inst, IFIDWrite, IFIDpc, IFIDinst, dFlush);
182
183 register_file register_file(.clk (clk), .rst (rst), .ra0 (IFIDinst[19:15]), .ra1
184 (IFIDinst[24:20]), .ra2 (m_rf_addr[4:0]), .wa (MEMWBrd), .wd (rf_wd), .we
185 (MEMWBrf_wr_en), .rd0 (rf_rd0), .rd1 (rf_rd1), .rd2 (rf_data));
186
187 imm imm(.inst (IFIDinst), .imm_out (imm_out));
188
189 controller controller(IFIDinst, rf_wr_en, alu_a_sel, alu_b_sel, alu_ctrl, dm_rd_ctrl,
190 dm_wr_ctrl, rf_wr_sel, comp_ctrl, do_branch, do_jump);
191
192 IDEX IDEX(clk, IFIDpc, IFIDinst[11:7], imm_out, rf_rd0, rf_rd1, rf_wr_en_hd, alu_a_sel,
193 alu_b_sel, alu_ctrl, dm_rd_ctrl, dm_wr_ctrl_hd, rf_wr_sel, comp_ctrl, do_branch, do_jump,
194 IDEXpc,
195 IDEXRd, IDEXImm, IDEXrf_rd0, IDEXrf_rd1, IDEXrf_wr_en, IDEXalu_a_sel, IDEXalu_b_sel,
196 IDEXalu_ctrl, IDEXdm_rd_ctrl, IDEXdm_wr_ctrl, IDEXrf_wr_sel, IDEXcomp_ctrl,
197 IDEXdo_branch, IDEXdo_jump,
198 IFIDinst[19:15], IFIDinst[24:20], eFlush, IDEXRs1, IDEXRs2);
199
200 alu alu(alu_a, alu_b, IDEXalu_ctrl, alu_out);
201

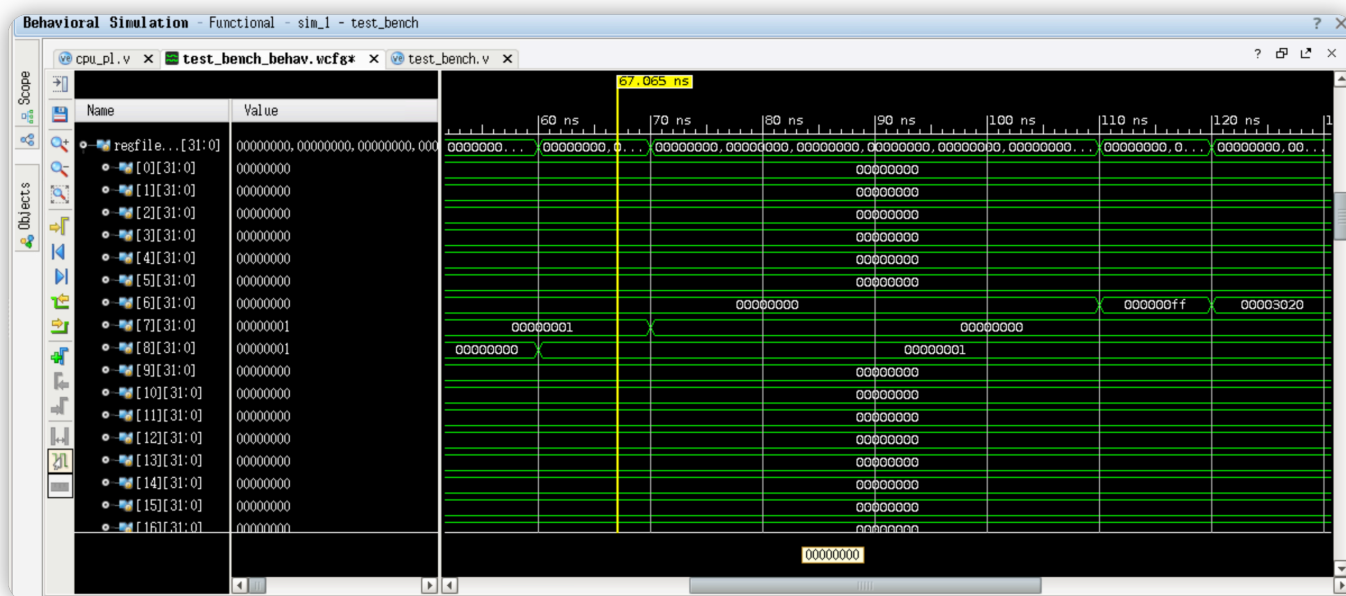
```

```

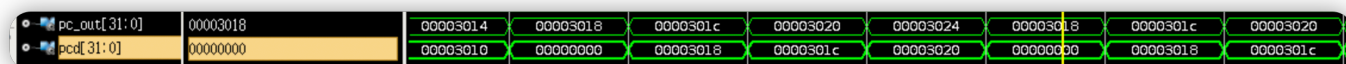
195 br br(.a (IDEXrf_rd0_fd), .b (IDEXrf_rd1_fd), .comp_ctrl (IDEXcomp_ctrl), .do_branch
    (IDEXdo_branch), .do_jump (IDEXdo_jump), .branch (branch));
196
197 EXMEM EXMEM(clk, alu_out, IDEXrf_rd1, IDEXRd, IDEXpc, IDEXrf_wr_en, IDEXdm_rd_ctrl,
    IDEXdm_wr_ctrl, IDEXrf_wr_sel, EXMEMalu_out, EXMEMrf_rd1, 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

```
1 module cpu_pl(
2     input clk,
3     input rst,
4
5     //io_bus
6     output [31:0] io_addr,    //led和seg的地址
7     output [31:0] io_dout,    //输出led和seg的数据
8     output io_we,            //输出led和seg数据时的使能信号
9     input [31:0] io_din,      //输出led和seg数据时的使能信号
10
11     //debug_bus
12     input [7:0] m_rf_addr,    //存储器 (mem) 或寄存器堆 (rf) 的调试读口地址
13     output [31:0] rf_data,    //从rf读取的数据
14     output [31:0] m_data,     //从mem读取的数据
15
16     //PC/IF/ID
17     output [31:0] pc_out,
18     output [31:0] pcd,
19     output [31:0] ir,
20     output [31:0] pcin,
21
22     //ID/EX
23     output [31:0] pce,
24     output [31:0] a,
25     output [31:0] b,
26     output [31:0] imm_debug,
27     output [4:0] rd,
28     output [31:0] ctrl,
29
30     //EX/MEM
31     output [31:0] y,
32     output [31:0] bm,
33     output [4:0] rdm,
34     output [31:0] ctrlm,
35
36     //MEM/WB
37     output [31:0] yw,
38     output [31:0] mdr,
39     output [4:0] rdw,
40     output [31:0] ctrlw
41 );
42
43 wire branch;
44 wire [31:0] pc;
45 wire [31:0] pc_in;
46 wire [31:0] pc_plus4;
47 wire [31:0] inst;
```

```

48 wire [31:0] imm_out;
49 wire rf_wr_en;
50 wire alu_a_sel;
51 wire alu_b_sel;
52 wire [1:0] alu_ctrl;
53 wire dm_rd_ctrl;
54 wire dm_wr_ctrl;
55 wire dm_wr_ctrl_aft;
56 wire [1:0] rf_wr_sel;
57 wire [2:0] comp_ctrl;
58 wire do_branch;
59 wire do_jump;
60
61 reg [31:0] rf_wd;
62 wire [31:0] rf_rd0, rf_rd1;
63 wire [31:0] alu_a, alu_b, alu_out;
64 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;
116         2'b01: IDEXrf_rd0_fd = rf_wd;
117         2'b10: IDEXrf_rd0_fd = EXMEMalu_out;
118         default: IDEXrf_rd0_fd = IDEXrf_rd0;
119     endcase
120
121     case (ForwardB)
122         2'b00: IDEXrf_rd1_fd = IDEXrf_rd1;
123         2'b01: IDEXrf_rd1_fd = rf_wd;
124         2'b10: IDEXrf_rd1_fd = EXMEMalu_out;
125         default: IDEXrf_rd1_fd = IDEXrf_rd1;
126     endcase
127 end
128
129 //hazard detection
130 wire PCWrite;
131 wire IFIDWrite;
132 wire stallpl;
133
134 wire dFlush;
135 wire eFlush;
136
137 hazard_detection hazard_detection(IDEXdm_rd_ctrl, IDEXRd, IFIDinst[19:15],
IFIDinst[24:20], PCWrite, IFIDWrite, stallpl, branch, dFlush, eFlush);
138
139 reg rf_wr_en_hd;
140 reg dm_wr_ctrl_hd;
141 always@(*) begin
142     if(stallpl) rf_wr_en_hd <= 1'b0;
143     else rf_wr_en_hd <= rf_wr_en;
144     if(stallpl) dm_wr_ctrl_hd <= 1'b0;
145     else dm_wr_ctrl_hd <= dm_wr_ctrl;
146 end
147
148 //io_bus
149 assign dm_wr_ctrl_aft = (~(io_addr[10])) & EXMEMdm_wr_ctrl; //AND gate

```

```

150 assign dm_dout_aft = io_addr[10] ? io_din : dm_dout;    //Mux
151 assign io_dout = rf_wd;
152 assign io_we = io_addr[10] & EXMEMdm_wr_ctrl;    //AND gate
153 assign io_addr = EXMEMalu_out;
154
155 assign alu_a = IDEXalu_a_sel ? IDEXrf_rd0_fd : IDEXpc;
156 assign alu_b = IDEXalu_b_sel ? IDEXImm : IDEXrf_rd1_fd;
157 assign pc_in = (pc >= 32'h3000 && pc <= 32'h33ff) ? pc : 32'h0;
158
159 assign pc_out = pc_in;
160 assign pcd = IFIDpc;
161 assign ir = IFIDinst;
162 assign pce = IDEXpc;
163 assign a = IDEXrf_rd0;
164 assign b = IDEXrf_rd1;
165 assign imm_debug = IDEXImm;
166 assign rd = IDEXRd;
167
168 wire fStall;
169 wire dStall;
170 assign fStall = !PCWrite;
171 assign dStall = !IFIDWrite;
172
173 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};
174
175 assign y = EXMEMalu_out;
176 assign bm = EXMEMrf_rd1;
177 assign rdm = EXMEMRd;
178 assign ctrlm = {30'b0, EXMEMdm_rd_ctrl, EXMEMdm_wr_ctrl};
179
180 assign yw = MEMWBalu_out;
181 assign mdr = MEMWBdm_dout;
182 assign rdw = MEMWBRd;
183 assign ctrlw = {29'b0, MEMWBrf_wr_en, MEMWBrf_wr_sel};
184
185 program_counter program_counter(clk, rst, branch, alu_out, PCWrite, pc, pcin, pc_plus4);
186
187 IFID IFID(clk, pc_in, inst, IFIDWrite, IFIDpc, IFIDinst, dFlush);
188
189 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));
190
191 imm imm(.inst (IFIDinst), .imm_out (imm_out));
192
193 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);
194

```



```

195 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,
196 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,
197 IFIDinst[19:15], IFIDinst[24:20], eFlush, IDEXRs1, IDEXRs2);
198
199 alu alu(alu_a, alu_b, IDEXalu_ctrl, alu_out);
200
201 br br(.a (IDEXrf_rd0_fd), .b (IDEXrf_rd1_fd), .comp_ctrl (IDEXcomp_ctrl), .do_branch
    (IDEXdo_branch), .do_jump (IDEXdo_jump), .branch (branch));
202
203 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);
204
205 ins_mem ins_mem(.a (pc_in[9:2]), .spo (inst));
206
207 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));
208
209 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);
210
211 //the last mux
212 always@(*) begin
213     case (MEMWBrf_wr_sel)
214         2'b00: rf_wd = 32'h0;
215         2'b01: rf_wd = MEMWBpc + 32'h4;
216         2'b10: rf_wd = MEMWBalu_out;
217         2'b11: rf_wd = MEMWBdm_dout; //2'b11: rf_wd = dm_dout_aft;
218         default: rf_wd = 32'h0;
219     endcase
220 end
221
222 endmodule

```

pdu.v

已经给出

top.v

```

1 module top(
2     input clk,
3     input rst,
4
5     //选择CPU工作方式
6     input run,

```

```

7      input step,
8
9      //输入switch的端口
10     input valid,
11     input [4:0] in,
12
13     //输出led和seg的端口
14     output [1:0] check, //led6-5:查看类型
15     output [4:0] out0,  //led4-0
16     output [2:0] an,    //8个数码管
17     output [3:0] seg,
18     output ready        //led7
19
20 );
21 wire clk_cpu;
22
23 wire [7:0] io_addr;
24 wire [31:0] io_dout;
25 wire io_we;
26 wire [31:0] io_din;
27
28 wire [7:0] m_rf_addr;
29 wire [31:0] rf_data;
30 wire [31:0] m_data;
31 wire [31:0] pc_out;
32 wire [31:0] pcd;
33 wire [31:0] ir;
34 wire [31:0] pcin;
35 wire [31:0] pce;
36 wire [31:0] a;
37 wire [31:0] b;
38 wire [31:0] imm_debug;
39 wire [4:0] rd;
40 wire [31:0] ctrl;
41 wire [31:0] y;
42 wire [31:0] bm;
43 wire [4:0] rdm;
44 wire [31:0] ctrlm;
45 wire [31:0] yw;
46 wire [31:0] mdr;
47 wire [4:0] rdw;
48 wire [31:0] ctrlw;
49
50 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,
51 pcin, pc_out, pcd, pce, ir, imm, mdr, a, b, y, bm, yw, rd, rdm, rdw, ctrl, ctrlm, ctrlw);
52 cpu_pl cpu_pl(clk_cpu, rst, io_addr, io_dout, io_we, io_din, m_rf_addr, rf_data, m_data,
pc_out,
53 pcd, ir, pcin, pce, a, b, imm_debug, rd, ctrl, y, bm, rdm, ctrlm, yw, mdr, rdw, ctrlw);
54
55 endmodule

```

hazard_test.asm

```
1  start:
2  sw x0, 0x408(x0)    #out1=0
3
4  #test data hazards
5  addi x1, x0, 1      #x1=1
6  addi x2, x1, 1      #x2=2
7  add x3, x1, x2      #x3=3
8  add x4, x1, x3      #x4=4
9  add x5, x1, x4      #x5=5
10 sw x5, 0x408(x0)    #out1=5
11
12 add x6, x1, x2      #x6=3
13 add x6, x6, x3      #x6=6
14 add x6, x6, x4      #x6=10
15 add x6, x6, x5      #x6=15
16 sw x6, 0x408(x0)    #out1=15
17
18 #test load-use hazard
19 lw x7, 0x40C(x0)    #x7=in
20 addi x8, x7, 1      #x8=in+1
21 addi x9, x8, -1     #x9=in
22 sw x9, 0x408(x0)    #out1=in
23
24 #test control hazard
25 beq x9, x0, start   #if (in==0) start
26 add x10, x9, x5
27 add x10, x10, x6
28 sw x10, 0x408(x0)   #out1=in+20
29 stop: jal x0, stop
30
31 #do not execute
32 add x11, x9, x10
33 add x12, x10, x11
34 add x13, x11, x12
```

fib_test.asm

```
1  .text
2      addi x1, x0, 1      #x1=1
3      add t1, x0, x0      #store fib series @t1
4
5  #### input f0
6      sw x1, 0x404(x0)    #rdy=1
7  l1:
8      lw t0, 0x410(x0)    #wait vld=1
9  #      addi a7, x0, 5      #for debug begin
10 #      ecall
11 #      mv t0, a0          #for debug end
12
```

```

13     beq t0, x0, l1
14     lw s0, 0x40c(x0) #s0=vin
15 #     addi a7, x0, 5    #for debug begin
16 #     ecall
17 #     mv s0, a0        #for debug end
18
19     sw s0, 0x408(x0) #out1=f0
20     sw s0, 0(t1)     #store f0
21     addi t1, t1, 4
22
23     sw x0, 0x404(x0) #rdy=0
24 l2:
25     lw t0, 0x410(x0) #wait vld=0
26 #     addi a7, x0, 5    #for debug begin
27 #     ecall
28 #     mv t0, a0        #for debug end
29
30     beq t0, x1, l2
31
32 ##### input f1
33     sw x1, 0x404(x0) #rdy=1
34 l3:
35     lw t0, 0x410(x0) #wait vld=1
36 #     addi a7, x0, 5    #for debug begin
37 #     ecall
38 #     mv t0, a0        #for debug end
39
40     beq t0, x0, l3
41     lw s1, 0x40c(x0) #s1=vin
42 #     addi a7, x0, 5    #for debug begin
43 #     ecall
44 #     mv s1, a0        #for debug end
45
46     sw s1, 0x408(x0) #out1=f1
47     sw s1, 0(t1)     #store f1
48     addi t1, t1, 4
49
50     sw x0, 0x404(x0) #rdy=0
51 l4:
52     lw t0, 0x410(x0) #wait vld=0
53 #     addi a7, x0, 5    #for debug begin
54 #     ecall
55 #     mv t0, a0        #for debug end
56
57     beq t0, x1, l4
58
59 ##### comput fi = fi-2 + fi-1
60 next:
61     add t0, s0, s1    #fi
62     sw t0, 0x408(x0) #out1=fi
63     sw t0, 0(t1)     #store fi
64     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

uart

```

FPGA0L uart beta 1.0
>

```

uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4
baud rate: 115200

segplay(sharing with led) hexplay

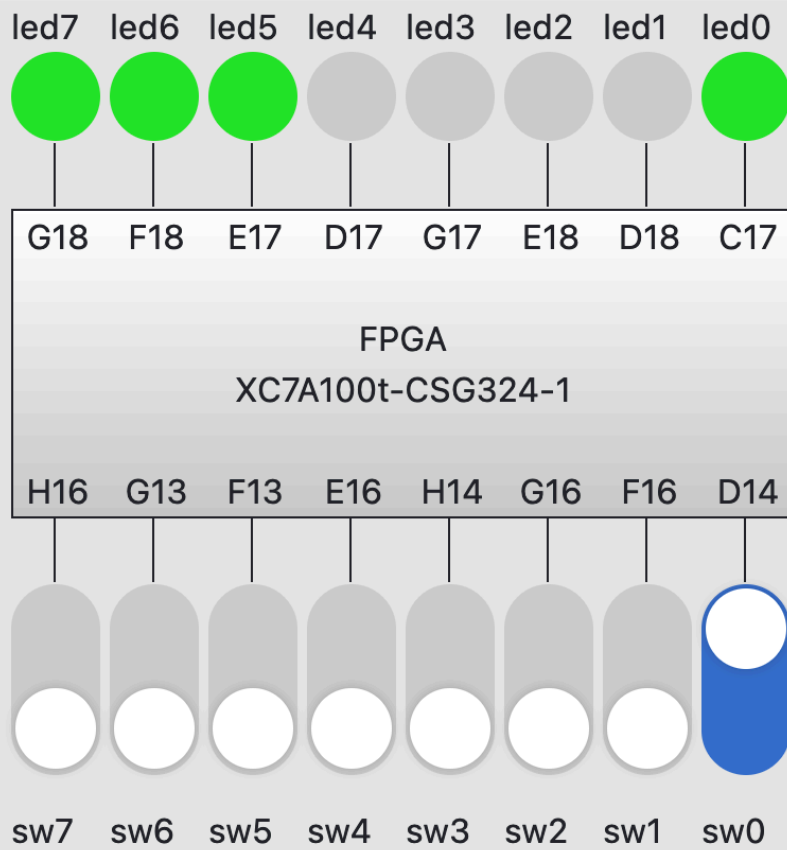
segplay pin: dot seg_gseg_f seg_eseg_dseg_cseg_bseg_a

soft clock button

None

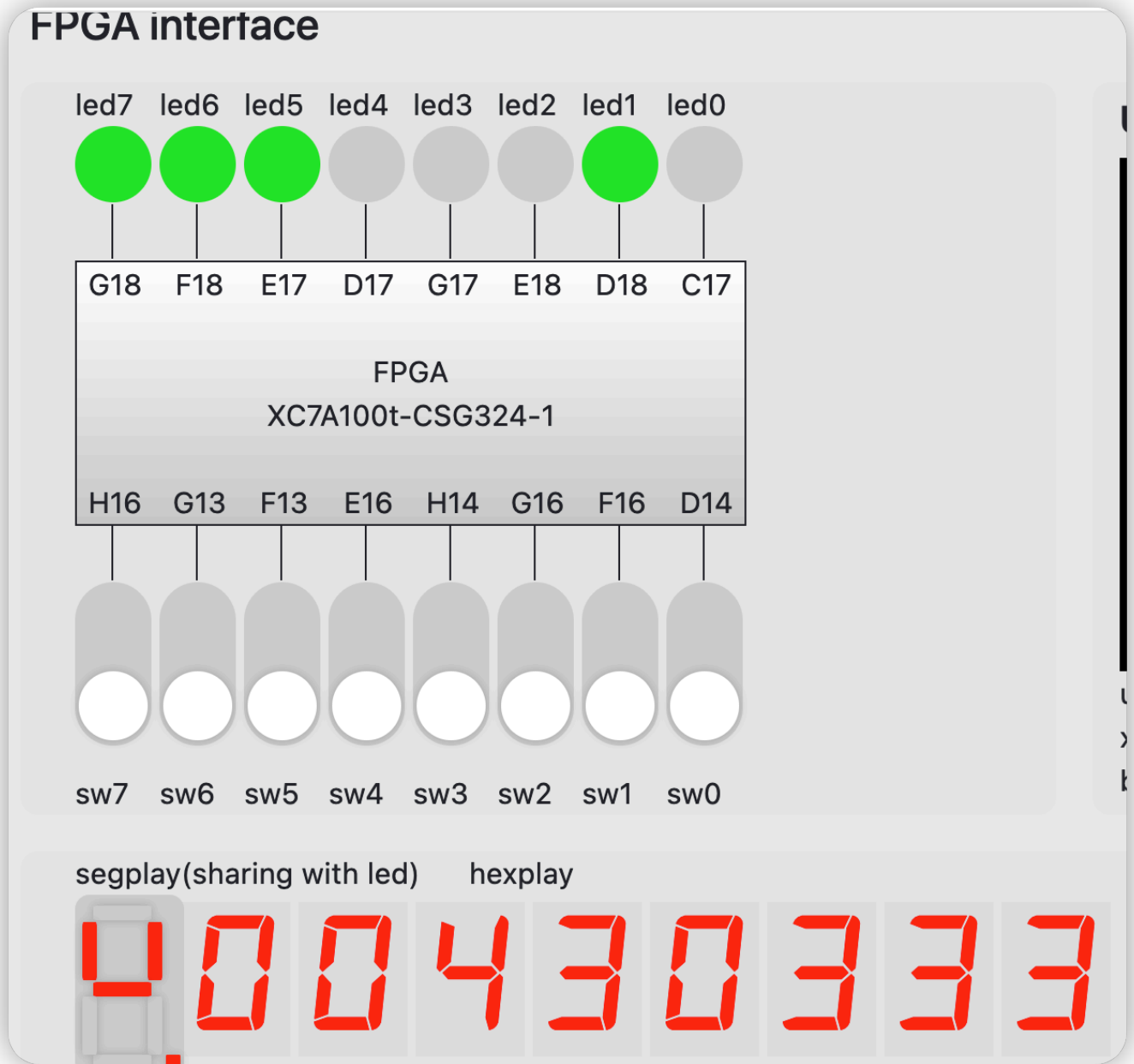
clk btn pins: clk_btn
xdc,ucf sym: R18

FPGA interface



segplay(sharing with led) hexplay





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。