浙江大学

本科实验报告

课程名称:	计算机体系结构
姓 名:	
学 院:	计算机科学与技术学院
系:	计算机科学与技术系
专 业:	计算机科学与技术
学 号:	
指导教师:	卜凯

2022年 10月 12日

浙江大学实验报告

课程名称: 计算机体系结构 实验类型: 综合

实验项目名称: Lab1: Pipelined CPU supporting RISC-V RVI32 Instructions

学生姓名: 专业: 计算机科学与技术 学号:

同组学生姓名: 指导老师: 卜凯

实验地点: 曹西 301 实验日期: <u>2022</u>年 <u>10</u>月 <u>12</u>日

1 Tasks and requirements

1.1 Tasks

We will have 5 main tasks in Lab-1, those are:

- 1. Understand RISC-V RV32I instructions.
- 2. Master the design methods of pipelined CPU executing RV32I instructions.
- 3. Master the method of **Pipeline Forwarding Detection** and **bypass unit** design.
- 4. Master the methods of 1-cycle stall of **Predict-not-taken branch** design.
- 5. Master methods of program verification of Pipelined CPU executing RV32I instructions.

1.2Requirements

This experiment would be based on SWORD development board, with xc7k325tffg676-2L FPGA.

We are given a Verilog project which most parts have been finished, but still need to modify some files:

- 1. **RV32core.v**, as CPU core module.
- 2. **CtrlUnit.v**, the control unit of the core.
- 3. HazardDetectionUnit.v, in which we need to realize forwarding and branch

prediction functions.

4. **Code2Inst.v**, because still some instructions in the preset program are not included in. But the modification is **not necessary**.

When this work is finished, we shall verify both the simulation results and the on-board results.

2 Contents and principles

All of the principles are based on RISC-V 5-stage pipeline CPU.

2.1RISC-V instructions

RISC-V ISA has many types of instructions, but all of them have a fixed length of **32-bit**, which makes the hardware design easier. Here we are about to deal with the following types of instructions:

31	25	24 20	19 15	5 14 12	11 7	6	0
	funct7	rs2	rs1	funct3	rd	opcode	\square R
	imm[11:	0]	rs1	funct3	rd	opcode	
	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S
	imm[12 10:5]	rs2	rs1	funct3	imm[4:1 11]	opcode	□ B
		imm[31:12]			rd	opcode	
	imm[20 10:	1 11]	imm[19:	12]	rd	opcode	

R-type instructions, mainly store register arithmetic results into register.

I-type instructions, store register and immediate arithmetic results into register, or load memory data into register.

S-type instructions, are to save register data to memory.

B-type instructions, deal with branch in program.

U-type instructions, deal with very large immediate.

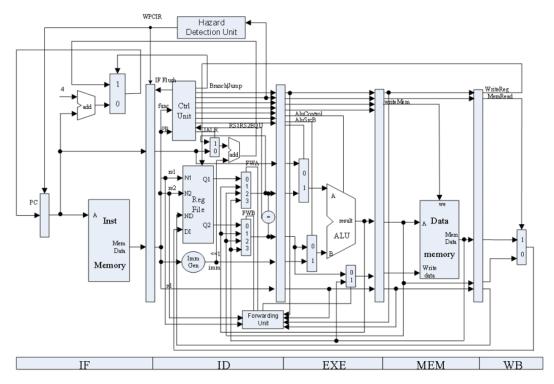
J-type instructions, mean jump to another address.

Each instruction would have different **opcode**, **funct3** or **funct7** bits, and we can read these bits to distinguish different instructions. This process is mainly dealt in **control unit** of CPU. After determining the control signals, the **data path** structure

would complete the work that the instruction informed.

2.2Data path and control unit

Following is the data path we need to implement.



The data path is typical **5-stage**, that is, divided by some registers into 5 stages: **IF**(Instruction Fetch), **ID**(Instruction Decode), **EXE**(Execute), **MEM**(Memory), **WB**(Write Back).

IF stage. We fetch the instruction from instruction memory.

ID stage. We try to decode the instruction and give out control signals.

EXE stage. We do ALU operations.

MEM stage. We write data into memory, or get data from memory.

WB stage. We write data back into registers.

Pipelining could improve CPU efficiency clearly, however, it may also cause some problems. There are 3 types of **hazards**. **Structure hazard** does not occur in our data path structure. **Data hazard** can be solved by bubbling, or forwarding. And we will try to avoid **control hazard** by prediction and flush.

Control unit is used to identify each instruction and give out control signals. We can see lots of control signals are transported to every stage and intervene in that

stage's operation. However, most of the signals are not important to this experiment. Now we mainly concern the signals that are used to implement **forwarding** and **prediction**.

The signals are introduced in more concerned parts following.

2.3Forwarding

Forwarding technique is used to solve data hazards. Using **hazard optype** signal to distinguish instruction types, we could achieve 3 types of **forwarding**:

- 1. An instruction using a specific register at ID stage after an ALU instruction which writes into the register at EXE stage.
- 2. An instruction using a specific register at ID stage after an ALU instruction which writes into the register at MEM stage.
- 3. An instruction using a specific register at ID stage after a LOAD instruction which writes into the register at MEM stage.

The forwarding unit will produce a forward control signal which control the **FWA** and **FWB MUXs** of ID stage, and then the required register data which is maintained in EXE or MEM stage could be forwarded to avoid data hazard.

However, there is one situation that forwarding cannot be used. We must **bubble**(stall):

4. An ALU or LOAD instruction using a specific register at ID stage after a LOAD instruction which writes into the register at EXE stage.

That's because the required data could be get only at MEM stage.

If this happened, we would let the ALU or LOAD instruction stop at ID stage, and let a bubble instruction go into EXE stage. Then the situation is converted to condition 3, which is mentioned above.

2.4Predict not-taken

Predictions is used to solve control hazards. We here will use **predict not-taken** method, which means we will predict all branch instructions as not branching -- the next instruction after a branch instruction will be fetched without any consideration.

Added a compare unit at ID stage, we could distinguish the branch instruction will

really branch or not. If not, then we naturally execute the next instruction. However, if branch, we have to invalidate the current instruction at ID stage.

To do this, we use **flush**. We could make all control signals that would affect the registers or memory invalid, then the instruction which was wrong would not make any effect on our CPU running conditions. It follows the correct instructions.

3 Steps and data records

3.1 Completed Verilog source files

3.1.1 RV32core.v

The filled blanks are as following.

These codes are mainly about MUXs used in data path.

```
1 //IF
2 MUX2T1_32 mux_IF(.I0(PC_4_IF),.I1(jump_PC_ID),.s(Branch_ctrl),.o(next_PC_IF));
3
4 //ID
5 MUX4T1_32 mux_forward_A(.I0(rs1_data_reg),.I1(ALUout_EXE),.I2(ALUout_MEM),.I3(Datain_MEM),.s(forward_ctrl_A),.o(rs1_data_ID));
6 MUX4T1_32 mux_forward_B(.I0(rs2_data_reg),.I1(ALUout_EXE),.I2(ALUout_MEM),.I3(Datain_MEM),.s(forward_ctrl_B),.o(rs2_data_ID));
7 //EXE
9 MUX2T1_32 mux_A_EXE(.I0(rs1_data_EXE),.I1(PC_EXE),.s(ALUSrc_A_EXE),.o(ALUA_EXE));
10 MUX2T1_32 mux_B_EXE(.I0(rs2_data_EXE),.I1(Imm_EXE),.s(ALUSrc_B_EXE),.o(ALUB_EXE));
11 MUX2T1_32 mux_forward_EXE(.I0(rs2_data_EXE),.I1(Datain_MEM),.s(forward_ctrl_ls),.o(Dataout_EXE));
```

3.1.2 CtrlUnit.v

33

34

The filled blanks are as following.

```
1 //instruction types decode(B, L, S, U, J types)
 2 wire BEQ = Bop & funct3_0;
 3 wire BNE = Bop & funct3_1;
 4 wire BLT = Bop & funct3 4;
 5 wire BGE = Bop & funct3_5;
 6 wire BLTU = Bop & funct3 6;
 7 wire BGEU = Bop & funct3_7;
 9 wire LB = Lop & funct3_0;
10 wire LH = Lop & funct3_1;
11 wire LW = Lop & funct3_2;
12 wire LBU = Lop & funct3_4;
13 wire LHU = Lop & funct3_5;
14
15 wire SB = Sop & funct3_0;
16 wire SH = Sop & funct3_1;
17 wire SW = Sop & funct3_2;
18
19 wire LUI = opcode == 7'b0110111;
20 wire AUIPC = opcode == 7'b0010111;
21
22 wire JAL = opcode == 7'b1101111;
23 assign JALR = opcode == 7'b1100111;
25 //control signals
26 assign Branch = (JAL | JALR | B_valid) & cmp_res;
27 assign cmp_ctrl = {3{B_valid}} & funct3;
28 assign ALUSrc_A = JAL | JALR | AUIPC;
29 assign ALUSrc_B = I_valid | L_valid | S_valid | AUIPC | LUI;
30 assign rs1use = R_valid | I_valid | B_valid | L_valid | S_valid | JALR;
31 assign rs2use = R_valid | B_valid | S_valid;
```

32 assign hazard_optype = {2{R_valid | I_valid | JAL | JALR | AUIPC | LUI}} & 2'b01 | {2{L_valid}} & 2'b10 |

{2{S_valid}} & 2'b11;

3.1.3 HazardDetectionUnit.v

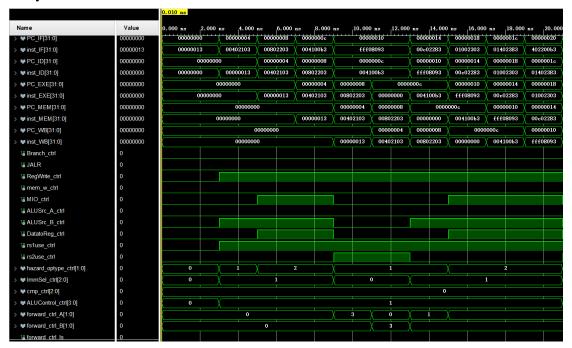
This module is as following.

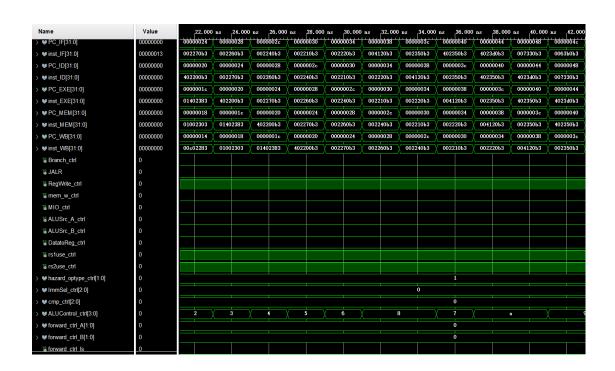
```
1 module HazardDetectionUnit(
        input clk,
        input Branch_ID, rs1use_ID, rs2use_ID,
        input[1:0] hazard_optype_ID,
      input[4:0] rd EXE, rd MEM, rs1 ID, rs2 ID, rs2 EXE,
       output PC_EN_IF, reg_FD_EN, reg_FD_stall, reg_FD_flush,
            reg_DE_EN, reg_DE_flush, reg_EM_EN, reg_EM_flush, reg_MW_EN,
       output forward_ctrl_ls,
       output[1:0] forward_ctrl_A, forward_ctrl_B
                 //according to the diagram, design the Hazard Detection Unit
        reg[1:0] hazard_optype_EXE, hazard_optype_MEM;
       always@(posedge clk) begin
          hazard_optype_MEM <= hazard_optype_EXE & {2{~reg_EM_flush}};
           hazard_optype_EXE <= hazard_optype_ID & {2{~reg_DE_flush}};
16
       parameter hazard_optype_ALU = 2'd1;
       parameter hazard optype LOAD = 2'd2;
       parameter hazard_optype_STORE = 2'd3;
21
       wire rs1_forward_1 = rs1use_ID && rd_EXE != 5'b00000 && rd_EXE == rs1_ID && hazard_optype_EXE == hazard_optype_ALU;
        wire rst_forward_stall = rsluse_ID && rd_EXE != 5'b00000 && rd_EXE == rst_ID && hazard_optype_EXE == hazard_optype_LOAD && hazard_optype_ID != hazard_optype_STORE;
       wire rs1_forward_2 = rsluse_ID && rd_MEM != 5'b00000 && rd_MEM == rs1_ID && hazard_optype_MEM == hazard_optype_ALU;
wire rs1_forward_3 = rsluse_ID && rd_MEM != 5'b00000 && rd_MEM == rs1_ID && hazard_optype_MEM == hazard_optype_LOAD;
        assign forward_ctrl_A = {2{rs1_forward_1}} & 2'b01 |
                                  {2{rs1_forward_2}} & 2'b10 |
                                   {2{rs1_forward_3}} & 2'b11 ;
        wire rs2_forward_1 = rs2use_ID && rd_EXE != 5'b00000 && rd_EXE == rs2_ID && hazard_optype_EXE == hazard_optype_ALU;
        wire rs2_forward_2 = rs2use_ID && rd_EXE != 5'b00000 && rd_EXE == rs2_ID && hazard_optype_EXE == hazard_optype_EXD && hazard_optype_EXD != hazard_optype_ID != hazard_optype_STORE;
wire rs2_forward_2 = rs2use_ID && rd_MEM != 5'b00000 && rd_MEM != rs2_ID && hazard_optype_MEM == hazard_optype_ALU;
wire rs2_forward_3 = rs2use_ID && rd_MEM != 5'b00000 && rd_MEM != rs2_ID && hazard_optype_MEM == hazard_optype_LOAD;
        assign forward_ctrl_B = \{2\{rs2\_forward\_1\}\}\ \&\ 2'b01\ |
                                   {2{rs2_forward_3}} & 2'b11 ;
        assign forward_ctrl_ls = rd_MEM == rs2_EXE && rd_MEM != 5'b00000 && hazard_optype_EXE == hazard_optype_STORE && hazard_optype_MEM == hazard_optype_LOAD;
42
        assign PC_EN_IF = ~(rs1_forward_stall | rs2_forward_stall);
        assign reg_FD_stall = rs1_forward_stall | rs2_forward_stall;
        assign reg_FD_flush = Branch_ID;
assign reg_DE_flush = rs1_forward_stall | rs2_forward_stall;
        assign reg_EM_flush = 1'b0;
        assign reg_FD_EN = 1'b1;
        assign reg_DE_EN = 1'b1;
        assign reg_EM_EN = 1'b1;
        assign reg_MW_EN = 1'b1;
```

You shall see the 2.3 and 2.4 chapters to get some explanation.

3.2Implementation results

Here we record all our behavioral simulation results. Most will not be used for analysis.





Name	Value	44.000 ns 46.000 ns 48.000 ns 50.000 ns 52.000 ns 54.000 ns 56.000 ns 58.000 ns 60.000 ns 62.000 ns														
> ₩ PC_IF[31:0]	00000000	00000050	00000054	00000058	00000056	, 000000000	00000064	00000068	00000066	X 00000070	00000074	00000078				
> W inst_IF[31:0]	00000013	00000033	ffd50093	00f27093	00f26093	00f24093	00f22093	00121093	00225093	40c35093	fff33093	fff3b093				
> W PC_ID[31:0]	00000000	0000004c	00000050	00000054	00000058	0000005c	00000060	00000064	00000068	0000006с	00000070	00000074				
> W inst_ID[31:0]	00000000	0063Р0Р3	00000033	ffd50093	00f27093	00f26093	00£24093	00£22093	00121093	00225093	40c35093	fff33093				
> W PC_EXE[31:0]	00000000	00000048	0000004c	00000050	00000054	00000058	0000005c	00000060	00000064	00000068	0000006c	00000070				
> ₩ inst_EXE[31:0]	00000000	007330ь3	0063Р0Р3	00000033	ffd50093	00f27093	00£26093	00£24093	00f22093	00121093	00225093	40c35093				
> ₩ PC_MEM[31:0]	00000000	00000044	00000048	0000004c	00000050	00000054	00000058	0000005c	00000060	00000064	00000068	0000006				
> ₩ inst_MEM[31:0]	00000000	4023d0ь3	007330ь3	0063Р0Р3	00000033	ffd50093	00£27093	00f26093	00f24093	00f22093	00121093	00225093				
> W PC_WB[31:0]	00000000	00000040	00000044	00000048	0000004c	00000050	00000054	00000058	0000005c	00000060	00000064	00000068				
> w inst_WB[31:0]	00000000	402350ь3	4023d0ь3	007330ь3	0063ь0ь3	00000033	ffd50093	00£27093	00f26093	00f24093	00f22093	00121093				
18 Branch_ctrl	0															
18 JALR	0															
16 RegWrite_ctrl	0															
16 mem_w_ctrl	0															
18 MIO_ctrl	0															
18 ALUSrc_A_ctrl	0															
16 ALUSrc_B_ctrl	0															
18 DatatoReg_ctrl	0															
18 rs1use_ctrl	0															
₩ rs2use_ctrl	0															
→ hazard_optype_ctrl[1:0]	0							1								
→ ImmSel_ctrl[2:0]	0		0	χ ' '					1							
→ cmp_ctrl[2:0]	0								0							
> ₩ ALUControl_ctrl[3:0]	0	9	X	1	3	4	5	8	6	χ) a	Χ				
→ forward_ctrl_A[1:0]	0								0							
→ w forward_ctrl_B[1:0]	0								0							
la forward ctrl Is	0															

Name	Value		66.000		00 ns	70.000		72.000		.000		000 ns	78.000		.000		00 ns	84.0	
> W PC_IF[31:0]	00000000	00000	=	00000080	_\	000084	0000		000000	=	0000009	_/_	00009с	000000	=(000000a4	_\	000ac	
> ⊌ inst_IF[31:0]	00000013	00520	==	00420663	=/=	000013	0042	1863	005216	=	0000001	/	42c863	005246	=(00000013	_/\	36863	
> ₩ PC_ID[31:0]	00000000	00000		0000007c		0000		==	000000	_		0000090		000000	^		0000a0		
> ₩ inst_ID[31:0]	00000000	fff3l		00520863	_\	420663	0000		004218	63	0052166	__	000013	0042c8	53 X	00524663	V 000	00013	
> ₩ PC_EXE[31:0]	00000000	00000	=	00000078	_\	00007c			000000	\equiv	0000008	_/_		00090		0000009c	_	00	
> ₩ inst_EXE[31:0]	00000000	fff33	==	fff3b093	_\	00520863		00420663			0042186	__	521663 00008c	000000	/\	0042c863	_\	24663	
> ₩ PC_MEM[31:0]	00000000		^		_^_	000078	0000007с			0000				^	00000090		_\	0000009с	
> ₩ inst_MEM[31:0]	00000000	40c35	=	fff33093	_\	fff3b093		00520863		00420663		/	421863	00521663		00000013	0042c863		
> ₩ PC_WB[31:0]	00000000	00000	==	00000070	_^_	000074	0000		000000	=		0000080		0000001	=(`		000090		
> ₩ inst_WB[31:0]	00000000	00225	093	40c35093	ff	£33093	fff3	ь093	005208	63	0042066	3 / 00	000013	004218	j3 /	00521663	000	00013	
1 Branch_ctrl	0																	_	
18 JALR	0																		
1⊌ RegWrite_ctrl	0								Щ										
1 mem_w_ctrl	0																	4_	
18 MIO_ctrl	0																		
18 ALUSrc_A_ctrl	0																		
1 ALUSrc_B_ctrl	0								Щ										
□ DatatoReg_ctrl	0																		
1 rs1use_ctrl	0																		
Is2use_ctrl Is2use_ctrl Is2use_ctrl Is3use_ctrl Is3us	0						$ldsymbol{ldsymbol{ldsymbol{ldsymbol{eta}}}$												
> ₩ hazard_optype_ctrl[1:0]	0	1			0		(1		X	0)	\square X \square	i	Χ	0		X =	1	
> W ImmSel_ctrl[2:0]	0	1			2		(1		X	2	?		1	χ	2		X	1	
> W cmp_ctrl[2:0]	0				0				X	i		X	Ó	χ	4		χ	0	
> W ALUControl_ctrl[3:0]	0	9			0		(1	i	X	0	,	$\overline{}$	i	Х	0		χ	1	
> W forward_ctrl_A[1:0]	0													0					
> W forward_ctrl_B[1:0]	0													0					
I forward ctrl Is	0																		

Name	Value	I	86.00	ns	88.000	ns	90.000	ns	92.000	ns 94.0	00 ns	96.000	ns	98.000	ns 100.0	00 ns 102.0	00 ns 104.	
> W PC_IF[31:0]	00000000	<u>) ' </u>	ооооро	, 0000	0064	, 9990	ООБс	0000	0000	00000064	, 990	000cc	, popo	00040	, 000000944	00000046	000000e0	
> w inst_IF[31:0]	00000013	0	063e663	0000	00013	0052	5863	0042	d663	00000013	006	3f863	0073	37663	00000013	00425663	00000013	
> W PC_ID[31:0]	00000000	(0	000000ac 00000 00736863 0063e663 000000a0 000000ac 00000013 00736863 000000a0 00524663 00000013		О0000000 О063e663 О000000				00Ъс	00	0000c0	000c0		000сс	000	00040) 000	
> w inst_ID[31:0]	00000000	(0							5863	0042d663	000	00013	0063£863		00737663	00000013	00425663 000d0	
> W PC_EXE[31:0]	00000000	00			00000		0000		000000Ъс	χ	0000	000c0		000000cc	000			
> w inst_EXE[31:0]	00000000	(0			36863	0063	8e663	0000	0013	00525863	004	2d663	0000	00013	0063f863	00737663	00000013	
> ₩ PC_MEM[31:0]	00000000	χ			000000ac 13 00736863		000000ь0		00Р0	000	000Ъс	X	0000	100c0	000000cc	00		
> W inst_MEM[31:0]	00000000) o							00000013	005	00525863		24663	00000013	0063f863	00737663		
> W PC_WB[31:0]	00000000) o	000009c	χ	0000	000a0 00000013		000000ac 00736863		00	0000Р0	000Ъ0		000Ъс	000	000c0	000000сс	
> W inst_WB[31:0]	00000000) o	042c863	0052	24663					0063e663	000	00013	005	25863	0042d663	00000013	0063f863	
18 Branch_ctrl	0																	
18 JALR	0																	
18 RegWrite_ctrl	0																	
18 mem_w_ctrl	0																	
18 MIO_ctrl	0																	
18 ALUSrc_A_ctrl	0																	
18 ALUSrc_B_ctrl	0																	
18 DatatoReg_ctrl	0																	
18 rs1use_ctrl	0																	
18 rs2use_ctrl	0																	
> W hazard_optype_ctrl[1:0]	0	χ		0		X	i)		•)	χ	1	X		0) i	χö	
> ₩ ImmSel_ctrl[2:0]	0	χ		2		(<u> </u>	i			2	χ	1	Χ		2	<u> </u>	2	
> ₩ cmp_ctrl[2:0]	0	χ		6		X	0			5	χ	0	χ		7	(5	
> ₩ ALUControl_ctrl[3:0]	0	χ		0		X	i			0	χ	1	χ		0	ì	Ó	
> ₩ forward_ctrl_A[1:0]	0														0			
→ forward_ctrl_B[1:0]	0														0			
la forward ctrl Is	0																	

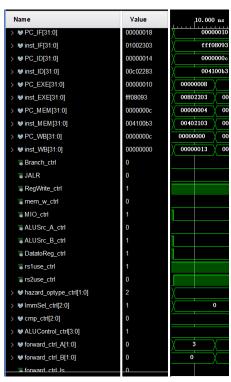
Name	Value																0 ns 12	6.0		
> ₩ PC_IF[31:0]	00000000	\\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \	0000ec	* 666 0	00£0	60000	810	000000£c	<u> </u>	000100	<u> </u>	0104	00000	108	00000	10c	0000	0110	000001:	14
> W inst_IF[31:0]	00000013	00	00c000ef		0013	01802403		00802e23	01	c02083	02801023		02002083		02800223		02402083		01a0108	
> W PC_ID[31:0]	00000000	00	0000e8	χ	0000	100ec		81000000	00	0000fc	0000	0100	00000	104	00000	108	0000	010c	000001	10
> w inst_ID[31:0]	00000000	00	000040ъ7		000ef	00000	013	01802403	00	B02e23	01c0	2083	02801	1023	0200	2083	0280	0223	0240208	33
> W PC_EXE[31:0]	00000000	000000de 00000013 000000 00425663		0000	000e8	X	0000	100ec	0000		0000	00fc	00000	100	00000	0104	0000	0108	0000010	0 c
> w inst_EXE[31:0]	00000000			0000	140ъ7	00c00	00ef	00000013	01	B02403	0080	2e23	01c02	2083	0280	1023	0200	2083	028002	23
> W PC_MEM[31:0]	00000000			000dc		000000e8 000040b7		000000ec			0000	8100	00000	Ofc	00000	100	0000	0104	0000010	80
> W inst_MEM[31:0]	00000000							00c000ef	00	00000013 0		01802403		2e23	01c02083		02801023		0200208	33
> W PC_WB[31:0]	00000000	000	000040	X	0000			000000e8	X	0000	00000ec		81000000		000000fc		00000100		0000010	
> w inst_WB[31:0]	00000000	00	000013	0042	5663	00000013		000040ъ7	00	c000ef	0000	0013	01802403		00802e23		01c02083		02801023	
18 Branch_ctrl	0																			
18 JALR	0																			
□ RegWrite_ctrl	0																			
le mem_w_ctrl	0																			
18 MIO_ctrl	0																			
18 ALUSrc_A_ctrl	0																			
1 ALUSrc_B_ctrl	0																			
□ DatatoReg_ctrl	0																			
% rs1use_ctrl	0																			
la rs2use_ctrl	0																			
> W hazard_optype_ctrl[1:0]	0			1				2	X	3) <u>2</u>	X	3		2		(:	3		
> W ImmSel_ctrl[2:0]	0	\mathcal{X}	5	X	3	X .		1	X	4) i	X	4		(i		('	1		
> w cmp_ctrl[2:0]	0														0					
> W ALUControl_ctrl[3:0]	0	X_{-}	c	X	ь	X .											1			
> W forward_ctrl_A[1:0]	0														0					
> W forward_ctrl_B[1:0]	0														0					
I forward ctrl Is	0																			



4 Analysis of the results

Here we only consider some typical results.

4.1 Successful forwarding



We will see the simulation in 9-11ns. During this period, the instructions in pipeline are:

IF 0xfff08093, addi x1, x1, -1

ID 0x004100b3, add x1, x2, x4

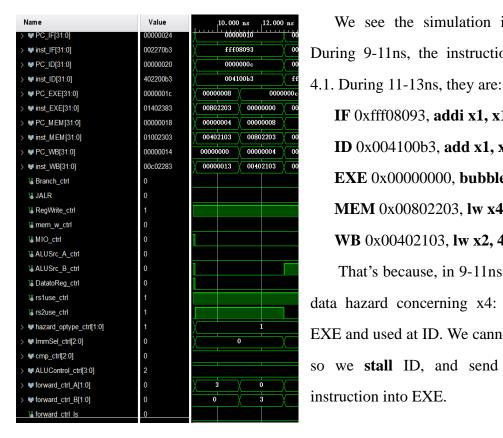
EXE 0x00802203, **lw** x4, 8(x0)

MEM 0x00402103, **lw x2**, **4**(**x0**)

WB 0x00000013, addi x0, x0, 0

Here we can see **forward_control_A** is 3. That is, the forwarding condition 3: LOAD x2 at MEM and using x2 at ID. That's a successful detection and forwarding.

4.2Stall to solve data hazard

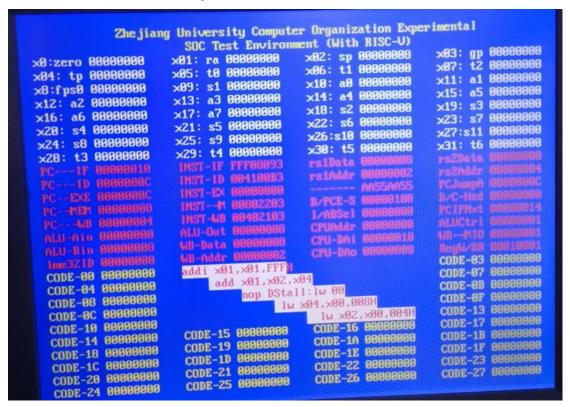


We see the simulation in 9-13ns. During 9-11ns, the instructions are as

IF 0xfff08093, addi x1, x1, -1 **ID** 0x004100b3, add x1, x2, x4 **EXE** 0x00000000, **bubble MEM** 0x00802203, **lw** x4, 8(x0) **WB** 0x00402103, **lw x2**, **4**(**x0**)

That's because, in 9-11ns, there is a data hazard concerning x4: LOAD at EXE and used at ID. We cannot forward, so we stall ID, and send a bubble instruction into EXE.

This result also is saved by us on FPGA:



That's the explained condition.

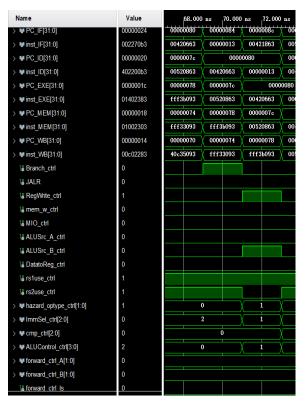
```
Zhejiang University Computer
                                                       ion Experimental
                        SOC Test Environment (With
                                           ent (With RISC-V)
×02: sp 00000008
x0:zero 00000000
                      x01: ra 00000000
                                                                  x03: gp 00000000
x04: tp 00000010
                      x05: t0 00000000
                                            x06: t1 00000000
                                                                  x07: t2 00000000
x8:fps0 000000000
                      x09: s1 00000000
                                            ×10: a0 00000000
                                                                 ×11: a1 00000000
x12: a2 00000000
                      x13: a3 000000000
                                            ×14: a4 00000000
                                                                 ×15: a5 00000000
×16: a6 00000000
                      ×17: a7 00000000
                                           x18: s2 000000000
                                                                 x19: s3 00000000
x20: s4 00000000
                      x21: s5 000000000
                                            x22: s6 00000000
                                                                 x23: s7 000000000
x24: s8 00000000
                      x25: s9 000000000
                                            x26:s10 000000000
                                                                 x27:s11 000000000
x28: t3 00000000
                      x29: t4 00000000
                                            x30: t5 000000000
                                                                 x31: t6 000000000
PC---IF 00000014
                      INST-IF 00C02283
                                            rs1Data 00000000
PC---ID 00000010
PC--EXE 0000000C
                                            rs1Addr 00000001
                      INST-EX 004100B3
INST-M 00000000
                                            ----- AA55AA55
B/PCE-S 00000100
PC-MEM 0000000C
                                                                 D/C-Hzd 00000000
PC---WB 00000008
                       INST-WB 00802203
                                            I/ABSel 00010001
ALU-Ain 00000008
                      ALU-Out 00000008
                                            CPUAddr 80000000
                                                                 ALUCtr1
ALU-Bin 00000010
                                            CPU-Dai 00000010
                                                                 WR--MIO 00000000
                      WB-Addr 00000004
                                            CPU-DAO 00000000
                                                                 RegN/DR 98818881
                                                                 CODE-03 00000000
CODE-00 00000000
                      lw x05,x00,00CH
                                                                 CODE-07
CODE-04 00000000
                           addi ×01,×01,FFF
                                                                         00000000
                                add x01,x02,x04
CODE-08 00000000
                                                                 CODE-0B 00000000
                                     mop DStall: lw 00
                                                                 CODE-0F 00000000
CODE-8C 8888888
                                             lw x04,x00,008H
CODE-10 00000000
                                                                 CODE-13 00000000
                      CODE-15 000000000
                                            CODE-16 00000000
                                                                 CODE-17 00000000
CODE-14 00000000
                                            CODE-1A 00000000
CODE-18 000000000
                      CODE-19 00000000
                                                                 CODE-1B 00000000
                                            CODE-1E 00000000
                      CODE-1D 00000000
                                                                 CODE-1F 00000000
CODE-1C 00000000
CODE-20 00000000
                                            CODE-22 00000000
                      CODE-21 00000000
                                                                 CODE-23 000000000
CODE-24 00000000
                      CODE-25 00000000
                                            CODE-26 00000000
                                                                 CODE-27 00000000
```

The next time snap, we can see we only stall the ID instruction for one step.

```
Zhejiang University Computer Organization Experimental
                       SOC Test Environment (With RISC-U)
x0:zero 00000000
                                          x02: sp 00000008
                                                                ×03: gp 00000000
                     x01: ra 00000000
                                                                x07: t2 000000000
x04: tp 00000010
                                           x06: t1 000000000
                     x05: t0 00000000
                                                                x11: a1 00000000
x8:fps0 00000000
                                          ×10: a0 000000000
                     x09: s1 000000000
                                                                x15: a5 000000000
x12: a2 00000000
                     x13: a3 00000000
                                          x14: a4 000000000
                                                                x19: s3 00000000
                                           x18: s2 000000000
×16: a6 000000000
                     ×17: a7 000000000
                     x21: s5 000000000
                                                                x23: s7 000000000
                                          x22: s6 000000000
x20: s4 000000000
                                                                x27:s11 00000000
x24: s8 00000000
                                          x26:s10 000000000
                     x25: s9 000000000
x28: t3 00000000
                                          x30: t5 00000000
                                                                x31: t6 000000000
                     x29: t4 00000000
                      INST-IF 01002303
                                           rs1Data 00000000
PC--- IF 00000018
                      INST-ID 00C02283
                                           rs1Addr 00000000
PC---ID 00000014
PC-EXE 00000010
                      INST-EX FFF08093
                                                                PCJumpA 00000020
                                                   AASSAASS
                      INST--M 004100B3
                                           B/PCE-S 00000100
                                                                D/C-Hzd 00000000
PC - MEM 0000000C
                      INST-WB 00000000
ALU-Out 00000018
PC -- WB 0000000C
                                                                ALUCtr1
ALU-Ain 00000018
                      WB-Data 00000010
                                           CPU-DAI 0000000F
                                                                WR--MIO 00000000
RecW/DR 00000001
                      WB-Addr 00000000
 mm321D 000000000
                                           CPU-DAo 00000010
                      lw x06,x00,010H
 CODE-00 00000000
                                                                CODE-03 00000000
 CODE-04 00000000
                          lw x05,x00,00CH
                                                                CODE-07 00000000
                               addi ×01,×01,FFF
 CODE-08 00000000
                                                                CODE-0B 00000000
 CODE-0C 00000000
                                     add x01,x02,x04
                                                                CODE-0F 00000000
                                          nop DStall: lw 00
 CODE-10 00000000
                                                                CODE-13 000000000
                                           CODE-16 00000000
                      CODE-15 00000000
 CODE-14 00000000
                                                                CODE-17 000000000
                      CODE-19 00000000
                                           CODE-1A 00000000
 CODE-18 00000000
                                                                CODE-1B 90000000
 CODE-1C 00000000
                      CODE-1D 00000000
                                           CODE-1E 00000000
                                                                CODE-1F 000000000
                      CODE-21 00000000
 CODE-20 00000000
                                           CODE-22 00000000
                                                                CODE-23 00000000
 CODE-24 00000000
                      CODE-25 00000000
                                           CODE-26 00000000
                                                                CODE-27 000000000
```

It runs normally after that stall.

4.3 successful branch not-taken prediction

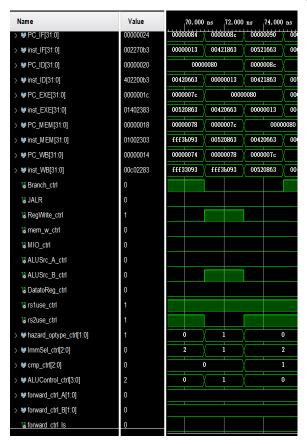


We see the simulation in 67-73ns. During 67-69ns, the instructions in pipeline are:

IF 0x00420663, beq x4, x4, 12 ID 0x00520863, bne x4, x4, 16 EXE 0xfff3b093, sltiu x1, x7, -1 MEM 0xfff33093, sltiu x1, x6, -1 WB 0x40c35093, srai x1, x6, 12

Now we get a branch instruction at ID, and we get the compare result: not taken. Then we normally execute the IF instruction, which is predicted to be fetched, in 71-73ns.

4.4Unsuccessful branch not-taken prediction



We see the simulation in 69-75ns. During 69-71ns, the instructions in pipeline are:

IF 0x0000013, addi x0, x0, 0 ID 0x00420663, beq x4, x4, 12 EX 0x00520863, bne x4, x4, 16 MEM 0xfff3b093, sltiu x1, x7, -1 WB 0xfff33093, sltiu x1, x6, -1

We get the compare result in ID: taken. Then we should not execute the instruction currently at IF. When this instruction is in EXE in 73-75ns, we **flush** it, that is, set all control signals to invalid.

This result also is saved by us on FPGA:

```
Zhejiang University Computer Organization Experimental
                        SOC Test Environment (With RISC-U)
                                                                x03: gp 00000000
x07: t2 0FFF0000
                                           x02: sp 00000008
                      x01: ra 00000001
x0:zero 00000000
                      x05: t0 00000014
                                           x06: t1 FFFF0000
×04: tp 00000010
                                                                ×11: a1 00000000
                                           ×10: a0 00000000
                      x09: s1 00000000
x8:fps0 00000000
                                           ×14: a4 00000000
                                                                x15: a5 000000000
                      x13: a3 000000000
×12: a2 00000000
                                                                ×19: s3 00000000
                                           ×18: s2 00000000
                      ×17: a7 000000000
x16: a6 00000000
                                                                x23: s7 000000000
                                           x22: s6 000000000
                      x21: s5 000000000
x20: s4 00000000
                                           x26:s10 000000000
                                                                x27:s11 000000000
                      x25: s9 000000000
×24: s8 00000000
                                                                x31: t6 000000000
                                           x30: t5 000000000
                      x29: t4 00000000
INST-IF 000000013
x28: t3 00000000
                                           rs1Bata 80000010
        8888888
                                           rs1Addr 000000004
                       INST-ID 88428663
PC---ID 00000000
                       INST-EX 00520863
PC--EXE 0000007C
PC--NEM 00000078
                                                                 B/C-Hzd 98888881
                                            B PCE-S 00018181
                      INST-M FFF38893
INST-WB FFF33893
CPU-DAI 88888888
                      ALU-Out 98989881
ALU-Ain 80000018
                      WB-Data 88888881
ALU-Bin 20000014
                                            CPU-DAD 88888888
                      WB-Addr 88888881
                                                                 CODE-83 60888888
                      nop JStall:addi0
                                                                  CODE-07 88888888
CODE-00 00000000
                           beq x04,x04,000C
                                                                  CODE-88 888888888
CODE-04 00000000
                                beq x04,x05,0010
                                                                  CODE-OF 00000000
CODE-08 00000000
                                      sltiu x01,x07,FFF
                                                                 CODE-13 00000000
                                              sltiu x01,x06,F
CODE-0C 00000000
                                                                  CODE-17 00000000
 CODE-10 00000000
                                            CODE-16 00000000
                       CODE-15 00000000
                                                                  CODE-1B 000000000
 CODE-14 00000000
                                             CODE-1A 00000000
                                                                  CODE-1F 00000000
                       CODE-19 000000000
                                             CODE-1E 00000000
 CODE-18 00000000
                       CODE-1D 00000000
                                                                  CODE-23 888888888
                                             CODE-22 00000000
 CODE-1C 00000000
                                                                  CODE-27 00000000
                       CODE-21 00000000
 CODE-20 00000000
                                             CODE-26 00000000
                       CODE-25 000000000
 CODE-24 00000000
```

We see the branch in ID is taken, but an invalid instruction has entered IF.

```
Zhejiang University Computer Organization Experimental
                       SOC Test Environment (With RISC-U)
                                                                x03: gp 00000000
x07: t2 0FFF0000
                     ×01: ra 00000001
                                          x02: sp 00000008
0:zero 00000000
                                          x06: t1 FFFF0000
                    x05: t0 00000014
04: tp 00000010
                                                                ×11: a1 00000000
                                          ×10: a0 000000000
                     x09: s1 000000000
8:fps0 00000000
                                                                ×15: a5 00000000
                                          ×14: a4 00000000
12: a2 00000000
                     ×13: a3 000000000
                                                                x19: s3 000000000
                                                   88888888
                                          x18: s2
                     ×17: a7 00000000
16: a6 00000000
                                                                x23: s7 000000000
                                          x22: s6 000000000
                     x21: s5 000000000
20: s4 00000000
                                                                x27:s11 00000000
                                          x26:s10 000000000
                     x25: s9 00000000
24: s8 000000000
                                          x30: t5 000000000
                                                                x31: t6 000000000
                     x29: t4 00000000
28: t3 00000000
                                          rs1Data 00000010
                     INST-IF 00521663
   -1F 00000099
                     INST-ID 00421863
   -ID 8888888C
                     INST-EX 00000013
                     INST--M 00420663
                                           B/PCE-S 00000100
                                                                 PCIFNxt 88888894
                                           I/ABSel 00020000
CPUAddr 00000000
                     INST-WB 00520863
                                           CPU-DAI FFFFFFBF
                     UB-Data 80000000
                                                                 Reold/DR 999000000
                                           CPU-DAO 00000010
                     WB-Addr 00000010
mm3210 000000010
                                                                 CODE-03 000000000
                     bne x04,x05,000C
CODE-00 000000000
                                                                 CODE-07 08000000
                          bne x04,x04,0010
CODE-84 00000000
                                                                 CODE-0B 00000000
                               nop JStall:addi0
CODE-88 88888888
                                                                 CODE-0F 00000000
                                     beq x04,x04,000C
CODE-8C 888888888
                                                                 CODE-13 00000000
                                          beq x04,x05,0010
CODE-10 00000000
                                                                 CODE-17 00000000
                                           CODE-16 00000000
                     CODE-15 000000000
CODE-14 00000000
                                                                 CODE-1B 00000000
                                           CODE-1A 00000000
                      CODE-19 000000000
CODE-18 00000000
                                                                 CODE-1F 00000000
                                           CODE-1E 00000000
                      CODE-1D 00000000
CODE-1C 00000000
                                                                 CODE-23 00000000
                                           CODE-22 000000000
                      CODE-21 00000000
CODE-25 00000000
CDDE-20 00000000
                                                                 CODE-27 00000000
                                           CODE-26 000000000
```

When it is in other stages, all control signals concerning it have been set invalid.

5 Discussion and Conclusion

5.1 Problems

5.1.1 Problems concerning Verilog language

Problem 1. Of '&' operator

In control unit, I first wrote a line as follows:

assign cmp_ctrl = B_valid & funct3;

That's a fault. I forgot that *B_valid* and *funct3* do not have the same bits, consequently using the meaning of '&&' operator as '&'. In the simulation I found *cmp_ctrl* had only one bit available, and then corrected this problem:

$$assign\ cmp_ctrl = \{3\{B_valid\}\}\ \&\ funct3;$$

5.1.2 Problems concerning the module

Problem 1. Forwarding of x0 register

At first, my forwarding unit didn't specify x0 register from other registers. However, as x0 is fixed to 0, forwarding x0 may cause some problem:

addi x0, x0, 1

add x1, x0, x0

If using this program, then x1 would become 2 instead of a correct 0 as x0 = 1 is forwarded. After comparing my program with my teammate, I corrected it.

5.1.3 Problems concerning the experiment guides

Problem 1. Of ALUSrc_A signal

In the data path shown in guides, $ALUSrc_A = 1$ means that using register data as input. However, in corresponded simulation results, $ALUSrc_A = 1$ means that using PC as input. Here I chose the second way, to achieve more readability when comparing with typical simulation results.

5.2 Achievements and conclusion

In this experiment, I reviewed RISC-V 5-stage pipeline CPU structure, and implemented forwarding and prediction functions. I get more familiar with Verilog language and Xilinx Vivado software as well. The experiment is successful.