流水线 CPU——verilog 实现

17373124 闫苗

1. 总体架构

数据通路总体设计：见数据通路 pdf

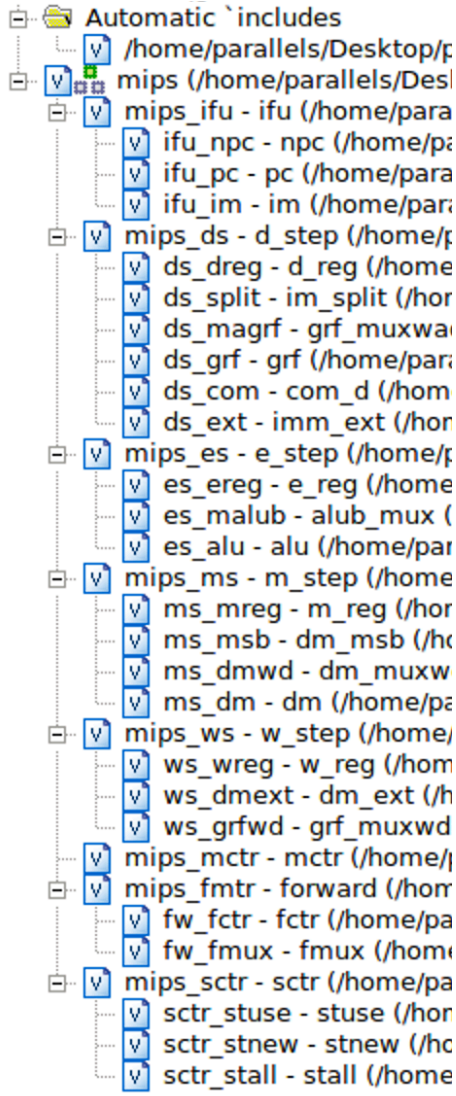
层次结构总体设计：

基础数据通路：

ifu— d 级— e 级— m 级— w 级

控制器：

主控制器、转发控制器、暂停控制器



指令支持总体设计：

本 cpu 支持指令集

{addu, subu, sll, srl, slt, ori, lui, lw, lb, sw, sb, j, jal, jalr, jr, beq}

解决三大冒险总体设计：

Pipelining Hazards

A *hazard* is a situation that prevents starting the next instruction in the next clock cycle

1) *Structural hazard*
   - A required resource is busy (e.g. needed in multiple stages)
2) *Data hazard*
   - Data dependency between instructions
   - Need to wait for previous instruction to complete its data read/write
3) *Control hazard*
   - Flow of execution depends on previous instruction

（1） 结构冒险。已经解决，cpu 中的 grf 和 dm 部件，可以独立的支持读和写。

（2） 数据冒险。最大程度的使用转发，尽力转发！迫不得已使用暂停。

（3） 控制冒险。采用延迟槽。

2. 基础数据通路搭建

PC

表 1 PC 接口定义表

文件	模块接口定义
PC.v	input [31:0] PC_in input clk, input reset, input update_PC, output reg [31:0] PC_out

IM

表 2 IM 接口定义表

文件	模块接口定义

IM.v	input [31:0] PC; // instruction address
	output [31:0] out; // instruction

IF/ID（F/D 级流水线寄存器）

表 3 IF/ID 接口定义表

文件	模块接口定义
IF_D.v	input clk, input reset, input en, //写入使能信号，和 stall 相关 input [31:0] PC8, input [31:0] IR_in, output [31:0] IR_out, output [31:0] PC8_out

D_ForwardMux

表 4 D 转发器接口定义表

文件	模块接口定义

MUX.v	input [2:0] ForwardRSD, input [2:0] ForwardRTD, input [31:0] RF_RD1, input [31:0] RF_RD2, input [31:0] ALU_out_M, input [31:0] PC8_E, input [31:0] PC8_M, input [31:0] WriteData_GRF, output [31:0] MFRSD, output [31:0] MFRTD

Controller

表 5 控制器接口定义表

文件	模块接口定义
Controller.v	input [5:0] op, input [5:0] fun, output [1:0] RegDst, output AluSrc, output MemtoReg, output RegWrite, output MemWrite, output MemRead, output [2:0] ExtOp, output [3:0] ALUOp, output Branch, output [1:0] jump, output is_jal

Write_back_GRF_Addr_MUX

表 6 写入寄存器地址的多路选择器接口定义表

文件	模块接口定义
MUX.v	input [4:0] rd,

	input [4:0] rt, input [4:0] ra, input [1:0] RegDst, output reg [4:0] WriteAddr_GRF

PC_Mux

表 7 PC 多路选择器信号生成模块接口定义表

文件	模块接口定义
ALU_ctl.v	input Branch, input [1:0] jump, input [31:0] MFRSD, input [31:0] MFRTD, output reg [2:0] MPC

EXT

表 8 EXT 接口定义表

文件	模块接口定义
EXT.v	input [15:0] imm16, input [2:0] EXTOp, output [31:0] ext_output

ID/EXE（D/E 级流水线寄存器）

表 9 ID/EXE 接口定义表

文件	模块接口定义
ALU.v	input [31:0] IR_in, input [31:0] RD1_in, input [31:0] RD2_in, input [31:0] EXT_in, input [31:0] PC8, input [4:0] WriteAddr_GRF, input [4:0] A1_D, input [4:0] A2_D, input [1:0] Tnew_D, input RegWriteEn_in,

	output [31:0] IR_out, output [31:0] RD1_out, output [31:0] RD2_out, output [31:0] EXT_out, output [31:0] PC8_out, output [1:0] Tnew_E, output [4:0] A1_E, output [4:0] A2_E, output [4:0] WriteAddr_GRF_E, output RegWriteEn_out, input clk, input reset, input clr

ALUSrc_Mux

表 10 ALUSrc_Mux 接口定义表

文件	模块接口定义
ALU.v	input ALUSrc, input [31:0] RD2, input [31:0] imm32, output [31:0] ALU_B

判断是需要转发后的 GRF[rt]还是拓展后的 imm。

ALU

表 11 ALU 接口定义表

文件	模块接口定义
ALU.v	input [31:0] A, input [31:0] B, output reg [31:0] C, input [3:0] ALUOp

E_ForwardMux

表 12 E 级转发器接口定义表

文件	模块接口定义

MUX.v	input [2:0] ForwardRSE, input [2:0] ForwardRTE, input [31:0] RS_E, input [31:0] RT_E, input [31:0] WriteData_GRF, input [31:0] ALU_out_M, input [31:0] PC8_M, output [31:0] MFRSE, output [31:0] MFRTE

EXE_MEM（E/M 级流水线寄存器）

表 13 EXE_MEM 接口定义表

文件	模块接口定义
EXE_MEM.v	input clk, input reset, input [31:0] IR_in, input [31:0] Add_in, input [31:0] RD2_in, input [31:0] PC8_in, input [1:0] Tnew_E, input [4:0] WriteAddr_GRF_E, input [4:0] A1_E, input [4:0] A2_E, input RegWriteEn_in, output [31:0] IR_out, output [31:0] Add_out, output [31:0] RD2_out, output [31:0] PC8_out, output [1:0] Tnew_M, output [4:0] WriteAddr_GRF_M, output [4:0] A1_M, output [4:0] A2_M,

	output RegWriteEn_out

M_ForwardMux

表 14 M 级转发器接口定义表

文件	模块接口定义
MUX.v	input [31:0] RT_M, input [31:0] WriteData_GRF, input [1:0] ForwardRTM, output [31:0] MFRTM

DM

表 15 DM 接口定义表

文件	模块接口定义
DM.v	input [31:0] A, input [31:0] WD, input [31:0] PC, input WE, input clk, input Reset, output [31:0] RD

MEM_WB

表 16 M/W 级流水线寄存器接口定义表

文件	模块接口定义
MEM_WB.v	input clk, input reset, input [31:0] IR_in, input [31:0] Add_in, input [31:0] DR_in, input [31:0] PC8, input [1:0] Tnew_M, input [4:0] WriteAddr_GRF_M, input RegWriteEn_in,

	output [31:0] IR_out, output [31:0] Add_out, output [31:0] DR_out, output [31:0] PC8_out, output [1:0] Tnew_W, output [4:0] WriteAddr_GRF_W, output RegWriteEn_out

Write_back_GRF_MUX

表 17 写回寄存器的值多路选择器接口定义表

文件	模块接口定义
MUX.v	input MemtoReg, input jal, input [31:0] PC8_W, input [31:0] ALU_out_W, input [31:0] DM_W, output reg [31:0] WriteData_GRF

AT

表 18 解码器接口定义表

文件	模块接口定义
AT.v	input [31:0] IR, output Tuse_RS0, output Tuse_RS1, output Tuse_RT0, output Tuse_RT1, output Tuse_RT2, output reg [1:0] Tnew_D, output [4:0] A1, output [4:0] A2

Forward

表 19 转发信号控制接口定义表

文件	模块接口定义

AT.v	input RegWriteEn_E, input RegWriteEn_M, input RegWriteEn_W, input [4:0] A1_D, input [4:0] A2_D, input [4:0] A1_E, input [4:0] A2_E, input [4:0] A2_M, input [4:0] A3_E, input [4:0] A3_M, input [4:0] A3_W, input [1:0] Tnew_E, input [1:0] Tnew_M, input [1:0] Tnew_W, input jal, output [2:0] ForwardRSD, output [2:0] ForwardRTD, output [2:0] ForwardRSE, output [2:0] ForwardRTE, output [1:0] ForwardRTM

Stall

表 20 暂停信号控制接口定义表

文件	模块接口定义
stall.v	input [31:0] IR_D, input Tuse_RS0, input Tuse_RS1, input Tuse_RT0, input Tuse_RT1, input Tuse_RT2, input [1:0] Tnew_E, input [1:0] Tnew_M, input [4:0] A3_E,

	input [4:0] A3_M, input RegWriteEn_E, input RegWriteEn_M, output UpdatePC, output IF_ID_EN, output ID_EXE_Clr

3．数据冒险分析处理

数据冒险是针对流水线写入寄存器的周期延后，而设计的补救策略，分为转发和暂停。为了整体效率，优先转发，或者说能转发则转发，迫不得已才暂停。

而转发和暂停的选择条件是根据数据需求和供给的匹配时间来判断的。

暂停:结果产生的时间晚于指令到达需求者时(前)的时间 。

转发:结果产生的时间早于/等于指令到达需求者时(前)的时间。

1. 转发

转发分为数据需求点和供给点。

需求点：

对数据的需求一共只有五种情况：D 级对 rs,rt 寄存器的需求，E 级对 rs,rt 寄存器的需求，M 级对 rt 寄存器的需求。

指令对应其中的一个或两个需求而已。如 beq 指令需求 ID 阶段的 rs,rt 寄存器，addu 指令需求 EX 阶段的 rs,rt 寄存器。

在需求点添加 mux forward，每一级只用在最先使用处添加一个即可。对于每个需求点，其后的所有各级流水线寄存器均需向其转发数据。（跳转指令在 D 级也有转发，在难点处重点分析考虑）

供给点，通常（除跳转指令）是 M 级的 alu 写入结果、W 级的 dm 输出结果和尚未写入寄存器堆的结果。

供给点转发的三个条件，一个注意：

1. 新生成（有效的）
2. 写入（写入信号）寄存器堆的数据
3. 与读寄存器编号对比相同。

注意：写入地址为$0 时，该路数据不被转发转发写入。

命名规范：

如 mfrsd 表示含义如下，其相应的控制信号为 frsd：

mf：MUX Forward

rs：rs 寄存器

d：D 级

如 mfalube 表示含义如下，其相应的控制信号为 falube：

mf：MUX Forward

alub：ALU 的 B 输入

e：E 级

优先级：mux 的 0 输入必然是本级流水线寄存器。其余输入端口数字越大优先级越高，越是最新产生的数据。

流水级	源寄存器	涉及指令	转发 mux	控制信号	输入				输出
					0	3	2	1	
D	rs	beq jalr jr	mfrsd	frsd	rf_r1	npc8@M	ao@M	wd@W	v1@D
	rt	beq	mfrtd	frtd	rf_r2	npc8@M	ao@M	wd@W	v2@D
E	rs	cal_r cal_i load store	mfalua1e	falua1e	v1@E	npc8@M	ao@M	wd@W	a_alu
	rt	cal_r	mfalua2e	falua2e	v2@E	npc8@M	ao@M	wd@W	v2_alu
M	rt	store	mfa2m	fa2m	v2@M			wd@W	v2_dm

2. 暂停

Tuse 和 Tnew 构造策略矩阵。Tuse 表示输入需求时间，Tnew 表示输出供给时间

把各种指令根据它们对寄存器堆的读取行为进行分类，同一类指令的工作流程是相似的。

- Cal_r 类（R-R，寄存器与寄存器进行计算）：add,addu,or,subu，sll，srl 等。
- Cal_i 类（R_I，寄存器与常数进行计算）：addi,lui,ori 等。
- Beq 类（通过 CMP 比较器判断跳转）：beq,bne,bgez 等。
- Load 类（读取内存的值）：lw,lb,lbu,lh 等。
- Save 类（保存值到内存）：sw,sb,sh 等。
- J 类（四条指令各自都有微妙差别，推荐四条指令分开处理）：j，jal，jalr，jr。

其中，只有 cal_r, cal_i, load 三类指令和 jal, jalr 会产生结果。

Tuse 矩阵：

指令进入 D 级后，其后的某个功能部件再经过多少 cycle 就必须要使用寄存器值，该值是根据指令静止确定的。（1 表示 e 级；2 表示 m 级）

	Tuse	
	rs	rt
Cal_r	1	1
Cal_i	1	/
beq	0	0
load	1	/
store	1	2
j	/	/
jal	/	/
jr	0	/
jalr	0	/

Tnew 矩阵，该值是随时钟周期变化，逐级减 1 直至 0：

指令	功能部件	E	M	W
Cal_r/rd	alu	1	0	0
Cal_i/rt	alu	1	0	0
load/rt	dm	2	1	0

store	/	/	/	/
beq	/	/	/	/
j	/	/	/	/
jal	pc	0	0	0
jr	/	/	/	/
jalr	pc	0	0	0

构造策略矩阵：

将以上两表分别做为两坐标综合，构造 rs 策略矩阵。

暂停条件：

1. Tnew > Tuse
2. 新产生的数据将会被写入寄存器堆（grf_we_（各级）==1）
3. 新产生的数据写入的寄存器标号，与使用的寄存器标号相同。

其中省略 Tnew 为 0 的纵列（npc8），Tuse 为 2 的横行（store）（对暂停问题没有实际影响），且省略部分指令写入的 rt 寄存器。

D 级当前指令			E 级（a3）(Tnew)			M 级(as3) (Tnew)
指令类型	源寄存器	Tuse	cal_r 1/rd	cal_i 1/rt	load 2/rt	load 1/rt
beq	rs/rt	0	STALL	STALL	STALL	STALL
cal_r	rs/rt	1			STALL	
cal_i	rs	1			STALL	
load	rs	1			STALL	
store	rs	1			STALL	
jr	rs	0	S	S	S	S
jalr	rs	0	S	S	S	S

(jr 和 jalr 作为特殊情况，特殊考虑，在测试时也应该构造特殊的测试程序)

由表中得种暂停情况：

`stall_rs0_e, stall_rs0_m, stall_rs1_e,`

`stall_rt0_e, stall_rt0_m, stall_rt1_e`

`rs` 或者 `rt` 寄存器中任意一个，发生了不能由转发解决的数据冒险时，都应该暂停。

当判断为暂停时（本实验采用 d 级指令固定指令暂停），pc 处和流水寄存器控制信号的相应设置：

- 冻结 PC，不让 PC 的值改变
- 让 E 级流水级寄存器清零，等于插入了一个 NOP 指令（clr）
- 冻结 D 级流水级寄存器，不让它的值改变。

代码实现（暂停的条件应当较高，这样才能保证整个 cpu 在较高的频率运行）：

4．控制冒险分析处理

采用延迟槽，硬件无需处理调度控制冒险，编译调度指令。

跳转时 jal 和 jalr 写入寄存器堆的值应当是 pc+8。

beq 实现时，npc 处 pc+offset 即可（当 d 级的 imm 传入 ifu 单元时，pc 值已经是 pc+4）

5．控制器 ctr

控制信号在 ctr 被产生之后，跟随各级流水寄存器，直至使用。

表 8 R 型指令控制器真值表

function	100001	100011	000000	000010	101010
opcode	000000	000000	000000	000000	000000
功能	addu	subu	sll	srl	slt
is_j[2:0]	000	000	000	000	000
grf_we	1	1	1	1	1
grf_waslt[1:0]	00 rd	00	00	00	00
grf_wdslt[1:0]	00 ALU.C	00	00	00	00
ext_slt[1:0]	x	x	x	x	x
alu_bslt	0	0	0	0	0
alu_op[2:0]	add (000)	sub (001)	shft_l (011)	shft_r (100)	comless (101)
dm_we	0	0	0	0	0

dm_wdslt[1:0]	x	x	x	x	x

表 8 I 型指令控制器真值表

function	n/a					
opcode	001101	001111	100011	100000	101011	101000
功能	ori	lui	lw	lb	sw	sb
is_j[2:0]	000	000	000	000	000	000
grf_we	1	1	1	1	0	0
grf_waslt[1:0]	01 rt	01	01	01	x	x
grf_wdslt[1:0]	00	00	01 dmo	01 dmo	x	x
ext_slt[1:0]	00 0ext	10 lzero	01 signed	01	01	01
alu_bslt	1 EXT.O	1	1	1	1	1
alu_op[2:0]	or (010)	or (010)	add (000)	add (000)	add (000)	add (000)
dm_we	0	0	0	0	1	1
dm_wdslt	x	x	x	x	0 rf_r2	1 B_zero
dm_ext	x	x	0	1	x	x

表 8 跳转型指令控制器真值表

function	n/a			001000	001001
opcode	000100	000010	000011	000000	000000
功能	beq	j	jal	jr	jalr
is_j[2:0]	001	010	010	011	010
grf_we	0	0	1	0	
grf_waslt[1:0]	x	x	10 0x1f	x	
grf_wdslt[1:0]	x	x	10	x	

			NPC.PC4		
ext_slt[1:0]	01	x	x	x	
alu_bslt	0	x	x	x	
alu_op[2:0]	x	x	x	x	
dm_we	0	0	0	0	
dm_wdslt[1:0]	x	x	x	x	

6．测试

课下作业要求指令集覆盖：

{ addu, subu, ori, lw, sw, beq, lui, j, jal, jr, nop }

下面列举部分，全部测试见压缩包中的 test 部分。

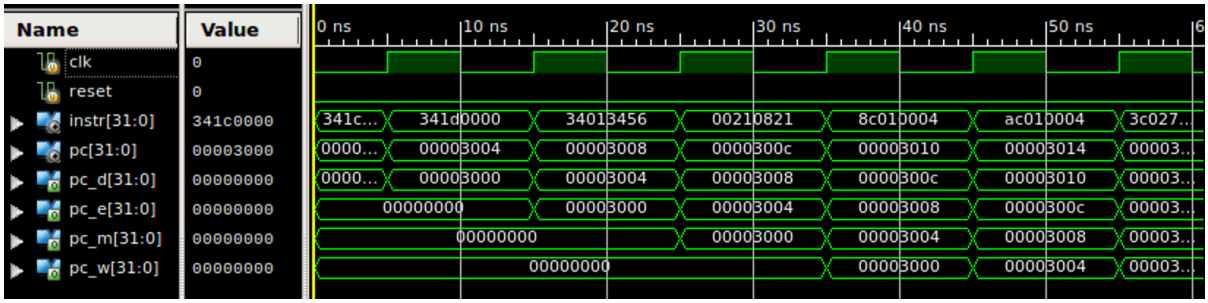
基础全指令代码测试

代码：

```
ori     $gp, $zero, 0
ori     $sp, $zero, 0
ori     $at, $zero, 0x3456
addu    $at, $at, $at
lw      $at, 4($zero)
sw      $at, 4($zero)
lui     $v0, 0x7878
subu    $v1, $v0, $at
lui     $a1, 0x1234
ori     $a0, $zero, 5
nop
sw      $a1, -1($a0)
lw      $v1, -1($a0)
beq     $v1, $a1, loc_0x3044
nop
j       loc_0x3084
nop
loc_0x3044:
ori     $a3, $v1, 0x404
```

```
beq     $a3, $v1, loc_0x3084
nop
lui     $t0, 0x7777
ori     $t0, $t0, 0xffff
move    $zero, $t0
ori     $zero, $zero, 0x1100
addu    $t2, $a3, $a2
ori     $t0, $zero, 0
ori     $t1, $zero, 1
ori     $t2, $zero, 1
loc_0x3070:
addu    $t0, $t0, $t2
beq     $t0, $t1, loc_0x3070
nop
jal     loc_0x3088
nop
addu    $t2, $t2, $t2
loc_0x3084:
j       loc_0x3084
nop
loc_0x3088:
addu    $t2, $t2, $t2
jr      $ra
nop
```

基本调试之后，程序正确运行



45@00003000: $28 <= 00000000

55@00003004: $29 <= 00000000

65@00003008: $ 1 <= 00003456

75@0000300c: $ 1 <= 000068ac

85@00003010: $ 1 <= 00000000

85@00003014: *00000004 <= 00000000

105@00003018: $ 2 <= 78780000

115@0000301c: $ 3 <= 78780000

125@00003020: $ 5 <= 12340000

135@00003024: $ 4 <= 00000005

145@0000302c: *00000004 <= 12340000

165@00003030: $ 3 <= 12340000

215@00003044: $ 7 <= 12340404

255@00003050: $ 8 <= 77770000

265@00003054: $ 8 <= 7777ffff

295@00003060: $10 <= 12340404

305@00003064: $ 8 <= 00000000

315@00003068: $ 9 <= 00000001

325@0000306c: $10 <= 00000001

335@00003070: $ 8 <= 00000001

375@00003070: $ 8 <= 00000002

415@0000307c: $31 <= 00003084

435@00003090: $10 <= 00000002

465@00003084: $10 <= 00000004

该测试覆盖：课上要求的基本指令最基础测试。

跳转指令测试：

代码：

```
lui $t0,0xffff
ori $t0,$t0,0xffff
sw $t0,0($0)
li $t0,0x12345678
ori $t1,1
ori $s1,1
beq $t1,$s1,end
```

```
ori $t2,2
addu $s0,$t0,$0
begin:
jr $ra
nop
addu $s2,$t0,$s0
end:
jal begin
nop
addu $s3,$t0,$s1
jal fun
nop
newbegin:
beq $t1,$t2,newend
nop
addu $t1,$t1,$t1
j newbegin
nop
lw $s4,0($0)
newend:
beq $t1,$t1,newend
nop
fun:
subu $t3,$t2,$t1
jr $ra
nop
```

基本调试之后，程序正确运行：

45@00003000: $ 8 <= ffff0000

55@00003004: $ 8 <= ffffffff

55@00003008: *00000000 <= ffffffff

75@0000300c: $ 1 <= 12340000

85@00003010: $ 8 <= 12345678

95@00003014: $ 9 <= 00000001
105@00003018: $17 <= 00000001
135@00003020: $10 <= 00000002
145@00003034: $31 <= 0000303c
185@0000303c: $19 <= 12345679
195@00003040: $31 <= 00003048
215@00003068: $11 <= 00000001
265@00003050: $ 9 <= 00000002

基础测试和跳转指令测试成功，在调试过程中，发现问题主要集中在转发和暂停部分。

问题原因分析：基本指令和跳转指令是通过 p4 的模块，能确保是基本正确的。所以最可能出现 bug 的地方也确实是新增的转发和暂停。（虽然控制冒险也是新增要求，但加了延迟槽之后并不要求 hdl 代码实现，应当写 mips 时注意手动调整。）

下面全面的列出所有数据冒险冲突的情形，以及相应的代表性测试样例，以求进行全面的测试。（这也是课下的思考题。）

暂停全面测试：

类型	测试类型	前序指令	冲突位置	冲突寄存器	测试序列
R 型	LD-E-RS	lw	D	RS	lw $4,0($5) addu $4,$4,$5
	LD-E-RT	lw	D	RT	lw $4,0($5) addu $4,$5,$4
I 型	LD-E-RS	lw	D	RS	lw $4,0($5) ori $5,$4,0xffff
LD 型	LD-E-RS	lw	D	RS	lw $4,0($5) lw $3,4($4)
ST 型	LD-E-RS	lw	D	RS	lw $4,0($5) sw $3,4($4)

JR	R-E-RS	addu	D	RS	addu $4,$4,$5 jr $4
	I-E-RS	ori	D	RS	ori $4,0xffff jr $4
	LD-E-RS	lw	D	RS	lw $4,0($5) jr $4
	LD-M-RS	lw	D	RS	lw $4,0($5) nop jr $4
B 型	R-E-RS	addu	D	RS	addu $4,$4,$5 beq $4,$5,loop
	I-E-RS	ori	D	RS	ori $4,0xffff beq $4,$5,loop
	LD-E-RS	lw	D	RS	lw $4,0($5) beq $4,$5,loop
	LD-M-RS	lw	D	RS	lw $4,0($5) nop beq $4,$5,loop
	R-E-RT	addu	D	RT	addu $4,$4,$5 beq $5,$4,loop
	I-E-RT	ori	D	RT	ori $4,0xffff beq $5,$4,loop
	LD-E-RT	lw	D	RT	lw $4,0($5) beq $5,$4,loop
	LD-M-RT	lw	D	RT	lw $4,0($5) nop beq $5,$4,loop

代码：

```
ori $4,$4,4
ori $3,$3,0x1234
sw $4,0($0)
```

```
sw $3,4($0)

lw $4,0($5)
addu $4,$4,$5

lw $4,4($5)
addu $4,$5,$4

lw $4,0($5)
ori $5,$4,8

lw $4,4($5)
lw $3,4($4)

lw $4,0($5)
sw $3,4($4)

jal fun
nop
fun:
ori $s3,$s3,16
addu $ra,$ra,$s3
jr $ra
nop

ori $10,0x3044
jr $10
nop

nop
ori $11,24
addu $10,$10,$11
sw $10,0($5)
```

```
lw $22,0($5)
jr $22
nop

ori $5,$5,16
ori $21,$21,0x3094
sw $21,0($5)
lw $4,0($5)
nop
jr $4
nop

addu $4,$4,$5
loop0:
beq $4,$5,loop0
nop

ori $5,0xffff
ori $4,0xffff
beq $4,$5,loop1
nop
loop1:


lw $4,0($25)
beq $4,$5,loop2
nop
loop2:

lw $4,0($25)
nop
beq $4,$5,loop3
nop
```

```
loop3:

addu $4,$4,$5
beq $5,$4,loop4
nop
loop4:

ori $4,0xffff
beq $5,$4,loop5
nop
loop5:

lw $4,0($25)
beq $5,$4,loop6
nop
loop6:

lw $4,0($25)
nop
loop7:
beq $5,$4,loop7
nop
```

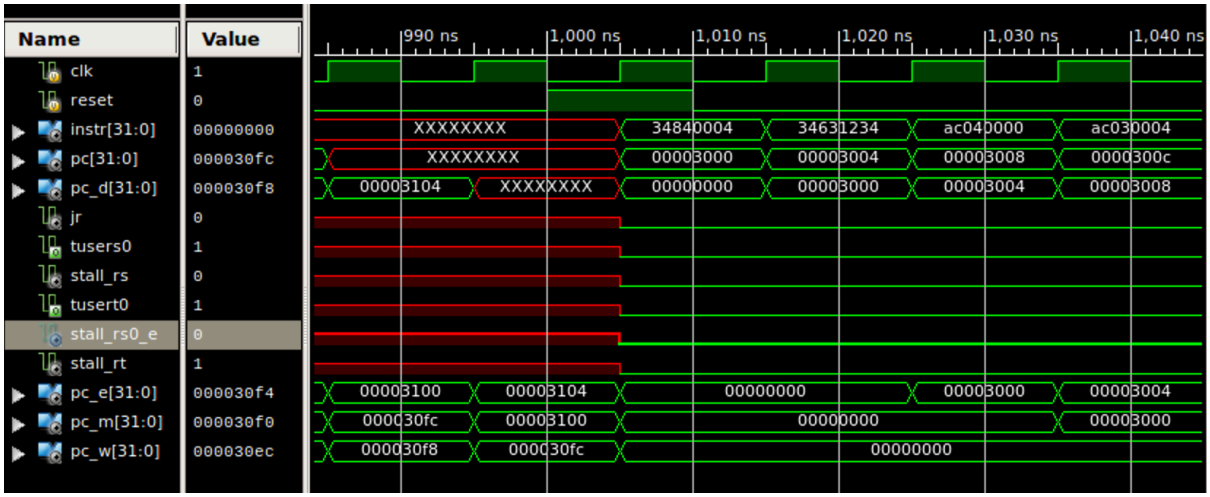
testbench 代码：

```
      initial begin
      // Initialize Inputs
      clk = 0;
      reset = 0;
      // Wait 100 ns for global reset to finish
      #1000;
     reset =1 ;
      #10;
      reset =0;
```

```
        // Add stimulus here
    end
        always #5 clk= ~clk;
```

测试结果：

逐一比对后，stall 全部正确，reset 也为正确。



display 输出：

45@00003000: $ 4 <= 00000004
55@00003004: $ 3 <= 00001234
55@00003008: *00000000 <= 00000004
65@0000300c: *00000004 <= 00001234
85@00003010: $ 4 <= 00000004
105@00003014: $ 4 <= 00000004
115@00003018: $ 4 <= 00001234
135@0000301c: $ 4 <= 00001234
145@00003020: $ 4 <= 00000004
165@00003024: $ 5 <= 0000000c
175@00003028: $ 4 <= 00000000
195@0000302c: $ 3 <= 00001234
205@00003030: $ 4 <= 00000000
215@00003034: *00000004 <= 00001234
235@00003038: $31 <= 00003040
255@00003040: $19 <= 00000010
265@00003044: $31 <= 00003050

305@00003050: $10 <= 00003044
345@00003044: $31 <= 00003060
385@00003060: $11 <= 00000018
395@00003064: $10 <= 0000305c
395@00003068: *0000000c <= 0000305c
415@0000306c: $22 <= 0000305c
475@00003060: $11 <= 00000018
485@00003064: $10 <= 00003074
485@00003068: *0000000c <= 00003074
505@0000306c: $22 <= 00003074
565@00003078: $ 5 <= 0000001c
575@0000307c: $21 <= 00003094
575@00003080: *0000001c <= 00003094
595@00003084: $ 4 <= 00003094
645@00003094: $ 4 <= 000030b0
685@000030a0: $ 5 <= 0000ffff
695@000030a4: $ 4 <= 0000ffff
735@000030b0: $ 4 <= 00000004
785@000030bc: $ 4 <= 00000004
835@000030cc: $ 4 <= 00010003
875@000030d8: $ 4 <= 0001ffff
915@000030e4: $ 4 <= 00000004
965@000030f0: $ 4 <= 00000004

转发部分测试（根据转发部分的表格制造）：

代码：

```
ori $1,$1,1
ori $2,$2,0xff
lui $3,0xffff
ori $3,$3,0xffff
sw $3,0($0)
```

```
sw $2,4($0)


#beq, rs, rt
loop0:
addu $s0,$1,$0
addu $s0,$s0,$1 # for wd_w
addu $s1,$1,$0 # for ao_m , should stall in the following line
beq $s0,$s1,loop0
ori $s2,$s2,2
addu $s1,$s1,$1
beq $s1,$s2,loop1
nop


#jr, rs, s0 ==2
fun:
jr $ra
nop
loop1:
jal fun  # npc8_m
nop

ori $6,$6,14
addu $ra,$ra,$6 # wd_w
addu $ra,$ra,$s0 # ao_m
jr $ra
nop


#cal_r, e step, rs, rt,
lui $3,1 # wd_w
addu $2,$2,$2 # ao_m
```

```
subu $1,$2,$3
nop

addu $2,$2,$2 # wd_w
jal next1 # npc8_m
subu $1,$2,$ra
next1:

addu $2,$2,$2 # wd_w
jal next2 # npc8_m
subu $1,$ra,$2
next2:


# cal_i, e step, rt
lui $4,1 # wd_w
addu $4,$2,$2 # ao_m ; selected
ori $4,$4,0xff

lui $4,1 # wd_w ; selected
addu $2,$2,$2 # ao_m
ori $4,$4,0xff

addu $ra,$2,$2 # wd_w
jal next3 # npc8_m
ori $ra,0xff
next3:


# load, rs, e step, has exception
ori $4,$0,0 # wd_w
ori $4,$0,4  # ao_m, no stall here
lw $5,0($4)
```

```
addu $4,$0,$0 # wd_w, selected
ori $3,$0,4  # ao_m, no stall here
lw $5,0($4)

ori $ra,$0,0 # wd_w
jal next4  # npc8_m, no stall here
lw $5,0($ra)  # exception !!
next4:


# store rs, e step, has exception
ori $4,$0,0 # wd_w
ori $4,$0,4  # ao_m, no stall here
sw $5,0($4)

addu $4,$0,$0 # wd_w, selected
ori $3,$0,4  # ao_m, no stall here
sw $5,0($4)

ori $ra,$0,0 # wd_w
jal next5  # npc8_m, no stall here
sw $5,0($ra)  # exception !!
next5:


#store, rt, m step
addu $ra,$4,$2
ori $ra,$0,0xfff
jal next6 # wd_w
sw $ra,4($0)
next6:
```

```
addu $ra,$4,$2
jal next7
ori $ra,$0,0xfff
next7:
sw $ra,4($0)

jal next8
nop
next8:
ori $ra,$0,0xfff
addu $ra,$4,$2
sw $ra,4($0)
```

仿真结果与 mips 中的结果比对之后，确认正确。

结果：

```
 45@00003000: $ 1 <= 00000001
 55@00003004: $ 2 <= 000000ff
 65@00003008: $ 3 <= ffff0000
 75@0000300c: $ 3 <= ffffffff
 75@00003010: *00000000 <= ffffffff
 85@00003014: *00000004 <= 000000ff
105@00003018: $16 <= 00000001
115@0000301c: $16 <= 00000002
125@00003020: $17 <= 00000001
155@00003028: $18 <= 00000002
165@0000302c: $17 <= 00000002
205@00003040: $31 <= 00003048
245@00003048: $ 6 <= 0000000e
255@0000304c: $31 <= 00003056
265@00003050: $31 <= 00003058
315@0000305c: $ 3 <= 00010000
325@00003060: $ 2 <= 000001fe
335@00003064: $ 1 <= ffff01fe
```

```
355@0000306c: $ 2 <= 000003fc
365@00003070: $31 <= 00003078
375@00003074: $ 1 <= ffffd384
385@00003078: $ 2 <= 000007f8
395@0000307c: $31 <= 00003084
405@00003080: $ 1 <= 0000288c
415@00003084: $ 4 <= 00010000
425@00003088: $ 4 <= 00000ff0
435@0000308c: $ 4 <= 00000fff
445@00003090: $ 4 <= 00010000
455@00003094: $ 2 <= 00000ff0
465@00003098: $ 4 <= 000100ff
475@0000309c: $31 <= 00001fe0
485@000030a0: $31 <= 000030a8
495@000030a4: $31 <= 000030ff
505@000030a8: $ 4 <= 00000000
515@000030ac: $ 4 <= 00000004
525@000030b0: $ 5 <= 000000ff
535@000030b4: $ 4 <= 00000000
545@000030b8: $ 3 <= 00000004
555@000030bc: $ 5 <= ffffffff
565@000030c0: $31 <= 00000000
575@000030c4: $31 <= 000030cc
585@000030c8: $ 5 <= 00000000
595@000030cc: $ 4 <= 00000000
605@000030d0: $ 4 <= 00000004
605@000030d4: *00000004 <= 00000000
625@000030d8: $ 4 <= 00000000
635@000030dc: $ 3 <= 00000004
635@000030e0: *00000000 <= 00000000
655@000030e4: $31 <= 00000000
665@000030e8: $31 <= 000030f0
665@000030ec: *000030f0 <= 00000000
```

```
685@000030f0: $31 <= 00000ff0
695@000030f4: $31 <= 00000fff
705@000030f8: $31 <= 00003100
705@000030fc: *00000004 <= 00003100
725@00003100: $31 <= 00000ff0
735@00003104: $31 <= 0000310c
745@00003108: $31 <= 00000fff
745@0000310c: *00000004 <= 00000fff
765@00003110: $31 <= 00003118
785@00003118: $31 <= 00000fff
795@0000311c: $31 <= 00000ff0
795@00003120: *00000004 <= 00000ff0
```

转发全面测试：

类型	测试类型	前序指令	冲突位置	冲突寄存器	测试序列
R 型(以 addu 为例)	R-W-RS	addu	D	RS	addu $4,$4,$5 nop nop addu $4,$4,$5
	I-W-RS	ori	D	RS	ori $4,$5,0xffff nop nop addu $4,$4,$5

	LD-W-RS	lw	D	RS	lw $4,0($5)
					nop
					nop
					addu $4,$4,$5

	JAL-W-RS	jal	D	RS	jal loop nop nop

					addu $1,$31,$1

	R-W-RT	addu	D	RT	addu $4,$4,$5
					nop
					nop
					addu $4, $5,$4

	I-W-RT	ori	D	RT	ori $4,$5,0xffff nop nop addu $4,$5,$4

	LD-W-RT	lw	D	RT	lw $4,0($5)
					nop
					nop
					addu $4,$5,$4

	JAL-W-RT	jal	D	RT	jal loop nop nop addu $1,$1,$31
	R-W-RS	addu	E	RS	addu $4,$4,$5 nop addu $4,$4,$5 nop
	I-W-RS	ori	E	RS	ori $4,$5,0xffff nop addu $4,$4,$5 nop

	LD-W-RS	lw	E	RS	lw $4,0($5)
					nop
					addu $4,$4,$5
					nop

	JAL-W-RS	jal	E	RS	jal loop nop addu $1,$31,$1

					nop
	R-M-RS	addu	E	RS	addu $4,$4,$5 addu $4,$4,$5 nop
	I-M-RS	ori	E	RS	ori $4,$5,0xffff addu $4,$4,$5 nop
	JAL-M-RS	jal	E	RS	jal loop addu $1,$31,$1 nop

	R-W-RT	addu	E	RT	addu $4,$4,$5
					nop
					addu $4, $5,$4
					nop

	I-W-RT	ori	E	RT	ori $4,$5,0xffff nop addu $4,$5,$4 nop

	LD-W-RT	lw	E	RT	lw $4,0($5)
					nop
					addu $4,$5,$4
					nop

	JAL-W-RT	jal	E	RT	jal loop nop addu $1,$1,$31 nop

	R-M-RT	addu	E	RT	addu $4,$4,$5
					addu $4,$5,$4
					nop

	I-M-RT	ori	E	RT	ori $4,$5,0xffff addu $4,$5,$4 nop

	JAL-M-RT	jal	E	RT	jal loop addu $1,$1,$31 nop
I 型(以 ori 为例)	R-W-RS	addu	D	RS	addu $4,$4,$5 nop nop ori $4,$4,0xffff
	I-W-RS	ori	D	RS	ori $4,$5,0xffff nop nop ori $4,$4,0x0000
	LD-W-RS	lw	D	RS	lw $4,0($5) nop nop ori $4,$4,0xffff
	JAL-W-RS	jal	D	RS	jal loop nop nop ori $1,$31,0xffff
	R-W-RS	addu	E	RS	addu $4,$4,$5 nop ori $4,$4,0xffff nop
	I-W-RS	ori	E	RS	ori $4,$5,0xffff nop ori $4,$4,0x0f0f nop
	LD-W-RS	lw	E	RS	lw $4,0($5) nop ori $4,$4,0xffff nop

	JAL-W-RS	jal	E	RS	jal loop nop ori $1,$31,0xffff nop
	R-M-RS	addu	E	RS	addu $4,$4,$5 ori $4,$4,0xffff nop
	I-M-RS	ori	E	RS	ori $4,$5,0xffff ori $4,$4,0xf0f0 nop
	JAL-M-RS	jal	E	RS	jal loop ori $1,$31,0xfff0 nop

LD 型	R-W-RS	addu	D	RS	addu $4,$4,$5
					nop
					nop
					lw $5,0($4)

	I-W-RS	ori	D	RS	ori $4,$5,0xffff nop nop lw $5,0($4)

	LD-W-RS	lw	D	RS	lw $4,0($5)
					nop
					nop
					lw $5,0($4)

	JAL-W-RS	jal	D	RS	jal loop nop nop lw $5,0($31)
	R-W-RS	addu	E	RS	addu $4,$4,$5 nop

					lw $5,0($4) nop
	I-W-RS	ori	E	RS	ori $4,$5,0xffff nop lw $5,0($4) nop

	LD-W-RS	lw	E	RS	lw $4,0($5)
					nop
					lw $5,0($4)
					nop

	JAL-W-RS	jal	E	RS	jal loop nop lw $5,0($31) nop

	R-M-RS	addu	E	RS	addu $4,$4,$5
					lw $5,0($4)
					nop

	I-M-RS	ori	E	RS	ori $4,$5,0xffff lw $5,0($4) nop
	JAL-M-RS	jal	E	RS	jal loop lw $5,0($31) nop

ST 型	R-W-RS	addu	D	RS	addu $4,$4,$5
					nop
					nop
					sw $5,0($4)

	I-W-RS	ori	D	RS	ori $4,$5,0xffff nop nop sw $5,0($4)

	LD-W-RS	lw	D	RS	lw $4,0($5)
					nop
					nop
					sw $5,0($4)

	JAL-W-RS	jal	D	RS	jal loop nop nop sw $5,0($31)

	R-W-RT	addu	D	RT	addu $4,$4,$5
					nop
					nop
					sw $4,0($5)

	I-W-RT	ori	D	RT	ori $4,$5,0xffff nop nop sw $4,0($5)

	LD-W-RT	lw	D	RT	lw $4,0($5)
					nop
					nop
					sw $4,0($6)

	JAL-W-RT	jal	D	RT	jal loop nop nop sw $31,0($5)

	R-M-RS	addu	E	RS	addu $4,$4,$5
					sw $5,0($4)
					nop

	I-M-RS	ori	E	RS	ori $4,$4,$5 sw $5,0($4) nop
	JAL-M-RS	jal	E	RS	jal loop sw $5,0($31)

					nop

	R-W-RS	addu	E	RS	addu $4,$4,$5
					nop
					sw $5,0($4)
					nop

	I-W-RS	ori	E	RS	ori $4,$5,0xffff nop sw $5,0($4) nop

	LD-W-RS	lw	E	RS	lw $4,0($5)
					nop
					sw $5,0($4)
					nop

	JAL-W-RS	jal	E	RS	jal loop nop sw $5,0($31) nop

	R-W-RT	addu	E	RT	addu $4,$4,$5
					nop
					sw $4,0($5)
					nop

	I-W-RT	ori	E	RT	ori $4,$5,0xffff nop sw $4,0($5) nop

	LD-W-RT	lw	E	RT	lw $4,0($5)
					nop
					sw $4,0($6)
					nop

	JAL-W-RT	jal	E	RT	jal loop nop sw $31,0($5)

					nop

	R-W-RT	addu	M	RT	addu $4,$4,$5
					sw $4,0($5)
					nop
					nop

	I-W-RT	ori	M	RT	ori $4,$5,0xffff sw $4,0($5) nop nop

	LD-W-RT	lw	M	RT	lw $4,0($5)
					sw $4,0($6)
					nop
					nop

	JAL-W-RT	jal	M	RT	jal loop sw $31,0($5) nop nop
JR	R-W-RS	addu	D	RS	addu $4,$4,$5 nop nop jr $4
	I-W-RS	ori	D	RS	ori $4,$5,0xffff nop nop jr $4
	LD-W-RS	lw	D	RS	lw $4,0($5) nop nop jr $4
	JAL-W-RS	jal	D	RS	jal loop nop nop

					jr $31
	R-M-RS	addu	D	RS	addu $4,$4,$5 nop jr $4
	I-M-RS	ori	D	RS	ori $4,$5,0xffff nop jr $4
	JAL-M-RS	jal	D	RS	jal loop nop jr $31
	JAL-E-RS	jal	D	RS	jal loop jr $31
B 型	R-W-RS	addu	D	RS	addu $4,$4,$5 nop nop beq $4,$3,loop
	I-W-RS	ori	D	RS	ori $4,$5,0xffff nop nop beq $4,$3,loop
	LD-W-RS	lw	D	RS	lw $4,0($5) nop nop beq $4,$3,loop
	JAL-W-RS	jal	D	RS	jal loop nop nop beq $31,$3,loop
	R-M-RS	addu	D	RS	addu $4,$4,$5 nop beq $4,$3,loop
	I-M-RS	ori	D	RS	ori $4,$5,0xffff

					nop beq $4,$3,loop
	JAL-M-RS	jal	D	RS	jal loop nop beq $31,$3,loop
	JAL-E-RS	jal	D	RS	jal loop beq $31,$3,loop
	R-W-RT	addu	D	RT	addu $4,$4,$5 nop nop beq $3,$4,loop
	I-W-RT	ori	D	RT	ori $4,$5,0xffff nop nop beq $3,$4,loop
	LD-W-RT	lw	D	RT	lw $4,0($5) nop nop beq $3,$4,loop
	JAL-W-RT	jal	D	RT	jal loop nop nop beq $3,$31,loop
	R-M-RT	addu	D	RT	addu $4,$4,$5 nop beq $3,$4,loop
	I-M-RT	ori	D	RT	ori $4,$5,0xffff nop beq $3,$4,loop
	JAL-M-RT	jal	D	RT	jal loop nop beq $3,$31,loop

	JAL-E-RT	jal	D	RT	jal loop beq $3,$31,loop

将上述完全测试逐一检测之后，确定转发代码基本正确。

7.思考题

在本实验中你遇到了哪些不同指令组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？请有条理的罗列出来。

主要考虑数据冲突。暂停和转发部分不同指令组合产生的冲突已经在上两表中。