计算机组成原理 P5 实验报告

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Verilog 实现的流水线 MIPS - CPU，支持的指令集包含{addu、subu、ori、lw、sw、beq、lui、j、jal、jr、nop}。为了实现这些功能，CPU 在顶层模块 mips 下并列包含了 DATAPATH、FORWARD_CONTROL、STOP_CONTROL 三个模块。

DATAPATH 模块下分五个流水级 F、D、E、M、W。F 级包含了 PC、IM 部件和用于 PC 选择的 MUX_PcSel、用于 pc+4 的 add4 两个小部件，D 级包含了 GRF、EXT、NPC、CMP 部件和用于写入寄存器 A3 选择的功能性部件 MUX_RegA3Sel；E 级包含了 ALU 部件和用于 ALUB 选择的功能性部件 MUX_AluBSel；M 级包含了 DM 部件；W 级连接到 D 级寄存器，包含了用于存入寄存器数据选择的功能性部件 MUX_RegDataSel。相邻两流水级之间还各设置了一个流水线寄存器 regD、regE、regM、regW，用于存储流水的信息。此外，为满足数据冒险的转发需求，还设置了五个转发多路选择器：D 级 MFCMP1D、MFCMP2D 分别用于选择需进入 CMP 部件和 E 级寄存器的两个寄存器值；E 级 MFALUAE、用于选择参与 ALU 运算的第一个数据，MFALUBE 用于选择参与 ALU 运算的第二个来自寄存器的数据和进入 M 级寄存器的数据；MFDM 用于选择写入 DM 的数据。

FORWARD_CONTROL 模块用于生成转发信号控制转发。

STOP_CONTROL 模块用于生成暂停信号控制暂停。

此外，工程文件中包含了名为 define 的.v 文件用于宏定义。本流水线 CPU 主要采用分布式译码，主要将指令与 A3 流水，所有需要指令信息的部件下均包括了译码部件 DECODE，用于产生改位置指令操作所需信息。

（二）关键模块定义

1. DATAPATH

（1） PC

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位至 0x00003000）
en	I	使能信号，为 1 时 PC 正常工作，为 0 时 PC 冻结
pcin[31:0]	I	32 位输入次指令地址
pc[31:0]	O	32 位输出当前指令地址

（2） IM

功能描述	指令存储器，内含 32*1024 字存储器，根据输入的地址输出指令	
信号名	方向	描述
pc[31:0]	I	32 位输入当前指令地址
instr[31:0]	O	32 位输出指令信号

（3） GRF

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
instr[31:0]	I	31 位输入 D 级指令，译码出写使能
instr_W[31:0]	I	31 位输入 W 级指令，译码出写使能
A3_W[4:0]	I	写入寄存器编号
RegData[31:0]	I	回写寄存器的值
pc4[31:0]	I	32 位输入当前 pc+4，用于 display
RD1[31:0]	O	第一个寄存器编号读出的值
RD2[31:0]	O	第二个寄存器编号读出的值

（4） CMP

功能描述	用于前置的 beq 跳转判断

信号名	方向	描述
D1[31:0]	I	32 位输入用于判断的第一个数据
D2[31:0]	I	32 位输入用于判断的第一个数据
zero	O	1 位输出判断结果，为 0 表示不相等，1 表示相等

（5） EXT

信号名	方向	功能
instr[31:0]	I	32 为输入当前指令，译码出操作选择信号和用于操作的立即数
ext[31:0]	O	32 位输出扩展结果

（6） NPC

信号名	方向	描述
pc4[31:0]	I	32 位输入当前 pc+4
instr[31:0]	I	32 位输入当前指令，译码出操作选择信号
zero	I	辅助选择 NPC（目前仅为 beq 使用）
ra[31:0]	I	32 位输入来自寄存器的指令地址
npc[31:0]	O	32 位输出用于跳转的次指令地址

（7） ALU

信号名	方向	描述
A[31:0]	I	第一个 32 位操作数
B[31:0]	I	第二个 32 位操作数
instr[31:0]	I	32 位输入当前指令，译码出操作选择信号
C[31:0]	O	32 位输出计算结果

（8） DM

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位）

pc4[31:0]	I	32 位输入当前 pc+4
instr[31:0]	I	32 位输入当前指令
Addr[31:0]	I	32 位输入写入 DM 的地址
Data[31:0]	I	32 位输入写入的数据
DMRD[31:0]	O	32 位输出读出的数据

（9） regD

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位）
instr[31:0]	I	32 位输入当前指令
pc4[31:0]	I	32 位输入当前 pc+4
en	I	使能信号，1 则寄存器正常工作，0 则冻结寄存器所存内容
instr_D[31:0]	O	32 位输出 D 级指令
pc4_D[31:0]	O	32 位输出 D 级 pc+4

（10） regE

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位）
clr	I	清空信号，1 则清除寄存器中所有值，0 则寄存器正常工作
instr[31:0]	I	32 位输入当前指令
V1[31:0]	I	32 位输入经转发选择后的 rs 寄存器值
V2[31:0]	I	32 位输入经转发选择后的 rt 寄存器值
ext[31:0]	I	32 位输入 ext 扩展结果
pc4[31:0]	I	32 位输入当前 pc+4
A3[4:0]	I	5 位输入当前指令对应写入寄存器编号
instr_E[31:0]	O	32 位输出 E 级指令
V1_E[31:0]	O	32 位输出 E 级 V1

V2_E[31:0]	O	32 位输出 E 级 V2
ext_E[31:0]	O	32 位输出 E 级 ext
pc4_E[31:0]	O	32 位输出 E 级 pc+4
A3_E[4:0]	O	5 位输出 E 级指令对应的写入寄存器编号

（11） regM

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位）
instr[31:0]	I	32 位输入当前指令
V2[31:0]	I	32 位输入 V2
ALUC[31:0]	I	32 位输入 ALU 计算结果
pc4[31:0]	I	32 位输入当前 pc+4
A3[4:0]	I	5 位输入当前指令对应写入寄存器编号
instr_M[31:0]	O	32 位输出 M 级指令
V2_M[31:0]	O	32 位输出 M 级 V2
ALUC_M[31:0]	O	32 位输出 M 级 ALU 计算结果
pc4_M[31:0]	O	32 位输出 M 级 pc+4
A3_M[4:0]	O	5 位输出 M 级指令对应的写入寄存器编号

（12） regW

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位）
instr[31:0]	I	32 位输入当前指令
pc4[31:0]	I	32 位输入当前 pc+4
ALUC[31:0]	I	32 位输入 ALU 计算结果
DMRD[31:0]	I	32 位输入 DM 读取结果
A3[4:0]	I	5 位输入当前指令对应写入寄存器编号

instr_W[31:0]	O	32 位输出 W 级指令
pc4_W[31:0]	O	32 位输出 W 级 pc+4
ALUC_W[31:0]	O	32 位输出 W 级 ALU 计算结果
DMRD_W[31:0]	O	32 位输出 W 级 DM 读取结果
A3_W[4:0]	O	5 位输出 W 级指令对应的写入寄存器编号

（13） add4

功能描述	用于 pc+4	
信号名	方向	描述
pc[31:0]	I	32 位输入当前指令地址
pc4[31:0]	O	32 位输出 pc+4

（14） MUX_PcSel

功能描述	用于选择使用普通 pc+4 或跳转指令地址	
信号名	方向	描述
pc4[31:0]	I	32 位输入 pc+4
instr[31:0]	I	32 位输入当前指令，译码出选择信号
npc[31:0]	I	32 位输入当前指令，译码用于选择进入 pc 的次指令地址
pcin[31:0]	O	32 位输出进入 pc 的次指令地址

（15） MUX_RegA3Sel

功能描述	用于选择写入寄存器编号	
信号名	方向	描述
instr[31:0]	I	32 位输入当前指令，译码出选择信号
A3[4:0]	O	5 位输出当前指令对应写入寄存器编号

（16） MUX_AluBSel

功能描述	用于选择进入 ALU 计算的 B 数据	
信号名	方向	描述

instr[31:0]	I	32 位输入当前指令，译码出选择信号
V2[31:0]	I	32 位输入经转发选择后的 rt 寄存器数据
ext[31:0]	I	32 位输入 ext 计算结果
ALUB[31:0]	O	32 位输出进入 ALU 的 B 数据

（17） MUX_RegDataSel

功能描述	用于选择存入寄存器的数据	
信号名	方向	描述
instr[31:0]	I	32 位输入当前指令，译码出选择信号
ALUC[31:0]	I	32 位输入 ALU 计算结果
DMRD[31:0]	I	32 位输入 DM 读出数据
pc4[31:0]	I	32 位输入当前 pc+4
RegData[31:0]	O	32 位输出存入寄存器数据

（18） MFCMP1D

功能描述	用于选择转发来的 rs 数据进入 CMP 和 E 级寄存器	
信号名	方向	描述
mfcmp1dSel[3:0]	I	4 位输入转发选择信号
RD1[31:0]	I	32 位输入当前位置指令从寄存器读取的数据
ALUC_M[31:0]	I	32 位输入来自 M 级的 ALU 计算结果
pc4_M[31:0]	I	32 位输入来自 M 级的 pc4
ALUC_W[31:0]	I	32 位输入来自 W 级的 ALU 计算结果
DMRD_W[31:0]	I	32 位输入来自 W 级的 DM 读取结果
pc4_W[31:0]	I	32 位输入来自 W 级的 pc4
mfcmp1[31:0]	O	32 位输出选择出的当前指令的真正 rs 读取数据

（19） MFCMP2D

功能描述	用于选择转发来的 rt 数据进入 CMP 和 E 级寄存器	
信号名	方向	描述

mfcmp2dSel[3:0]	I	4 位输入转发选择信号
RD2[31:0]	I	32 位输入当前位置指令从寄存器读取的数据
ALUC_M[31:0]	I	32 位输入来自 M 级的 ALU 计算结果
pc4_M[31:0]	I	32 位输入来自 M 级的 pc4
ALUC_W[31:0]	I	32 位输入来自 W 级的 ALU 计算结果
DMRD_W[31:0]	I	32 位输入来自 W 级的 DM 读取结果
pc4_W[31:0]	I	32 位输入来自 W 级的 pc4
mfcmp2[31:0]	O	32 位输出选择出的当前指令的真正 rt 读取数据

（20） MFALUAE

功能描述	用于选择转发来的真正寄存器值进入 ALU 作为 A	
信号名	方向	描述
mfaluaeSel[3:0]	I	4 位输入转发选择信号
V1_E[31:0]	I	32 位输入来自 E 级寄存器流水的 V1 值
ALUC_M[31:0]	I	32 位输入来自 M 级的 ALU 计算结果
pc4_M[31:0]	I	32 位输入来自 M 级的 pc4
ALUC_W[31:0]	I	32 位输入来自 W 级的 ALU 计算结果
DMRD_W[31:0]	I	32 位输入来自 W 级的 DM 读取结果
pc4_W[31:0]	I	32 位输入来自 W 级的 pc4
mfalua[31:0]	O	32 位输出选择出的当前指令进入 ALU 的真正寄存器值

（21） MFALUBE

功能描述	用于选择转发来的真正寄存器值进入 ALU 的 B 值选择	
信号名	方向	描述
mfalubeSel[3:0]	I	4 位输入转发选择信号
V2_E[31:0]	I	32 位输入来自 E 级寄存器流水的 V2 值
ALUC_M[31:0]	I	32 位输入来自 M 级的 ALU 计算结果
pc4_M[31:0]	I	32 位输入来自 M 级的 pc4
ALUC_W[31:0]	I	32 位输入来自 W 级的 ALU 计算结果

DMRD_W[31:0]	I	32 位输入来自 W 级的 DM 读取结果
pc4_W[31:0]	I	32 位输入来自 W 级的 pc4
mfalub[31:0]	O	32 位输出选择出的当前指令用于 ALUB 选择的真正寄存器值

（22） MFDM

功能描述	用于选择转发来的真正数据用于写入 DM	
信号名	方向	描述
mfdmSel[3:0]	I	4 位输入转发选择信号
V2_M[31:0]	I	32 位输入来自 M 级寄存器流水的 V2 值
ALUC_W[31:0]	I	32 位输入来自 W 级的 ALU 计算结果
pc4_W[31:0]	I	32 位输入来自 W 级的 pc4
DMRD_W[31:0]	I	32 位输入来自 W 级的 DM 读取结果
mfdm[31:0]	O	32 位输出选择出的当前指令用于存入 DM 的真正数据

2. FORWARD_CONTROL

功能描述	立足 D 级，用于转发控制信号的生成	
信号名	方向	描述
instr_D[31:0]	I	32 位输入位于 D 级的指令
instr_E[31:0]	I	32 位输入位于 E 级的指令
instr_M[31:0]	I	32 位输入位于 M 级的指令
instr_W[31:0]	I	32 位输入位于 W 级的指令
A3_M[4:0]	I	5 位输入 M 级指令的写入寄存器地址
A3_W[4:0]	I	5 位输入 W 级指令的写入寄存器地址
mfcmp1dSel[3:0]	O	4 位输出 cmp1 转发数据选择信号
mfcmp2dSel[3:0]	O	4 位输出 cmp2 转发数据选择信号
mfaluaeSel[3:0]	O	4 位输出 ALUA 转发数据选择信号
mfalubeSel[3:0]	O	4 位输出 ALUB 转发数据选择信号
mfdmSel[3:0]	O	4 位输出 DM 转发数据选择信号

3. STOP_CONTROL

功能描述	放眼全局，用于暂停控制信号的生成	
信号名	方向	描述
instr_D[31:0]	I	32 位输入位于 D 级的指令
instr_E[31:0]	I	32 位输入位于 E 级的指令
instr_M[31:0]	I	32 位输入位于 M 级的指令
A3_E[4:0]	I	5 位输入 E 级指令的写入寄存器地址
A3_M[4:0]	I	5 位输入 M 级指令的写入寄存器地址
enPC	O	输出用于暂停的 PC 使能信号
enD	O	输出用于暂停的 D 级流水线寄存器使能信号
clrE	O	输出用于暂停的 E 级流水线寄存器清空信号

4. DECODE

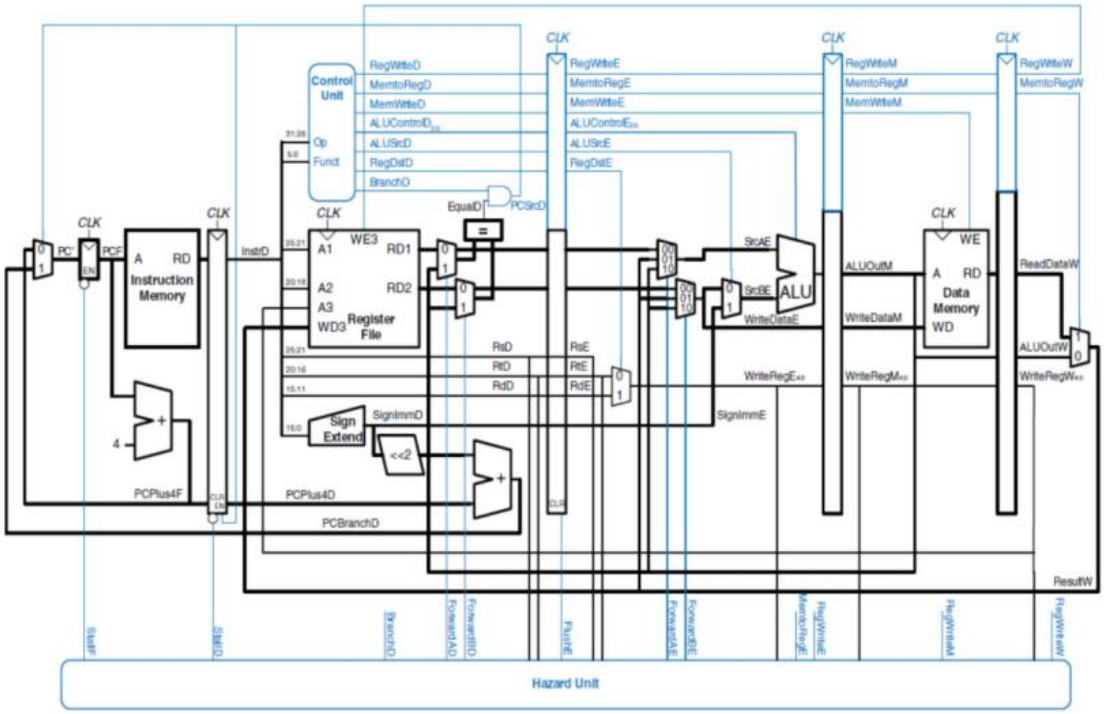
功能描述	用于译码产生各种控制信号	
信号名	方向	描述
instr[31:0]	I	32 位输入指令信号
NPCOp[2:0]	O	3 位输出 npc 操作选择信号
EXTOp[3:0]	O	4 位输出 ext 操作选择信号
ALUOp[3:0]	O	4 位输出 ALU 操作选择信号
RegWrite	O	1 位输出寄存器写使能信号
MemWrite	O	1 位输出 DM 写使能信号
RegA3Sel[2:0]	O	3 位输出写入寄存器编号选择信号
RegDataSel[2:0]	O	3 位输出写入寄存器数据选择信号
AluBSel[2:0]	O	3 位输出 ALUB 选择信号
PcSel[1:0]	O	2 位输出 pc 次指令地址选择信号
Tuse_rs0	O	表示指令经 0 周期使用 rs 寄存器值
Tuse_rs1	O	表示指令经 1 周期使用 rs 寄存器值
Tuse_rt0	O	表示指令经 0 周期使用 rt 寄存器值
Tuse_rt1	O	表示指令经 1 周期使用 rt 寄存器值

Tuse_rt2	O	表示指令经 2 周期使用 rt 寄存器值
Tnew[2:0]	O	表示指令产生写入寄存器值的部件

（三）数据通路的综合

1. 所有指令的指令级别数据通路

		addu	subu	ori	lw	sw	beq	lui	j	jal	jr	nop	MUX	
PC		ADD4	ADD4	ADD4	ADD4	ADD4	ADD4|NPC	ADD4	NPC	NPC	RD1	ADD4	NPC	
IM		PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	
ADD4		PC	PC	PC	PC	PC	PC	PC	PC	PC		PC	PC	
D级	instr	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	
	pc4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4		ADD4	ADD4	
RF	A1	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D				instr[25:21]@D		instr[25:21]@D	
	A2	instr[20:16]@D	instr[20:16]@D			instr[20:16]@D	instr[20:16]@D						instr[20:16]@D	
EXT	imm			instr[25:0]@D	instr[25:0]@D	instr[25:0]@D		instr[25:0]@D					instr[25:0]@D	
NPC	pc4						pc4@D		pc4@D	pc4@D			pc4@D	
	imm						instr[25:0]@D		instr[25:0]@D	instr[25:0]@D			instr[25:0]@D	
CMP	ra										RD1		RD1	
	D1						RD1						MFCMP1D	
	D2						RD2						MFCMP2D	
E级	V1	RD1	RD1	RD1	RD1	RD1							MFCMP1D	
	V2	RD2	RD2			RD2							MFCMP2D	
	A1	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D	instr[25:21]@D							instr[25:21]@D	
	A2	instr[20:16]@D	instr[20:16]@D			instr[20:16]@D							instr[20:16]@D	
	A3	instr[15:11]@D	instr[15:11]@D	instr[20:16]@D	instr[20:16]@D			instr[20:16]@D		1f			MUX_A3	
	ext			EXT.ext	EXT.ext	EXT.ext		EXT.ext					EXT.ext	
	pc4									pc4@D			pc4@D	
ALU	A	V1@E	V1@E	V1@E	V1@E	V1@E		ext@E					MFALUAE	
	B	V2@E	V2@E	ext@E	ext@E	ext@E							MUX_ALUB	MFALUBE
	V2					V2@E							MFALUBE	
	A2					A2@E							A2@E	
M级	ALUO	ALU.C	ALU.C	ALU.C	ALU.C	ALU.C		ALU.C					ALU.C	
	A3	A3@E	A3@E	A3@E	A3@E			A3@E		A3@E			A3@E	
	pc4									pc4@D			pc4@E	
DM	MemAddr				ALUO	ALUO@M							ALUO@M	
	MemData					V2@M							MFWDM	
W级	A3	A3@M	A3@M	A3@M	A3@M			A3@M		A3@M			A3@M	
	pc4									pc4@D			pc4	
	DMRD				DM[ALUO]								DM[ALUO]	
	ALUO	ALUO@M	ALUO@M	ALUO@M				ALUO@M					ALUO@M	
RF	A3	A3@M	A3@M	A3@M	A3@M			A3@W		A3@W			A3@M	
	RegData	ALUO@W	ALUO@W	ALUO@W	DMRD@W			ALUO@W		pc4@W			MUX_RegData	



（四）重要机制实现方法

1. 跳转

NPC 模块译码出 NPC 的计算控制信号，计算出 npc 后输出到 F 级 MUX_PcSel，MUX_PcSel 对来自 D 级寄存器的指令（即与 NPC 同指令）译码判断是否为跳转指令，若是则输出 npc 至 PC；反之输出当前 F 级 pc+4（来自 add4 部件的输出）至 PC。

2. 暂停

	rs	rt	功能部件	E	M	W
addu	1	1	ALU	1	0	0
subu	1	1	ALU	1	0	0
ori	1		ALU	1	0	0
lw	1		DM	2	1	0
sw	1	2				
beq	0	0				
lui			ALU	1	0	0
j						
jal			PC	0	0	0
jr	0					
nop						

	Tnew	E			M			W		
		ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
rs	Tuse	1	2	0	0	1	0	0	0	0
	0	S	S	F	F	S	F	F	F	F
	1	F	S	F	F	F	F	F	F	F
	Tnew	E			M			W		
		ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
rt	Tuse	1	2	0	0	1	0	0	0	0
	0	S	S	F	F	S	F	F	F	F
	1	F	S	F	F	F	F	F	F	F
	2	F	F	F	F	F	F	F	F	F

构建策略矩阵，当 Tnew>Tuse 时必须使用暂停，产生暂停信号，此时需要冻结 PC 的值，冻结 D 级流水线的值并清零 E 级流水线。

暂停信号由 rs、rt 寄存器的暂停信号取并集，当 Tnew 与 Tuse 满足矩阵中暂停条件，且写入寄存器与读取寄存器相同时相应暂停信号置 1。

对 D、E、M 级指令分别译码即可得到 Tuse 与 Tnew 值。

3. 转发

沿用 Tnew 信号（此处似乎与教程和课件内容不甚相符），本设计在宏定义时不同 Tnew 信号反映该级目前指令产生写入寄存器值的部件，因此在转发控制中只需要根据该级处指令写入寄存器编号与 D 级指令读取寄存器编号相同、该级指令产生写入寄存器值的部件两个信号判断转发哪一数据。

对 M、W 级指令分别译码即可得到该两级产生的写入寄存器数据来源，从而明确转发的数据来源。

此处需注意，如果写入寄存器为 0 号寄存器，不进行转发。

二、测试方案

（一）典型测试样例

Tuse_rs=0，Tnew_E=ALU，暂停	
ori $2, $0, 0x3000 ori $3, $0, 0x2fff ori $4, $0, 1 addu $5, $3, $4 ## beq $5, $2, label ## ori $10, $0, 5 ori $11, $0, 6 label: ori $12, $0, 0x302c ## jr $12 ## ori $1, $0, 9 lui $9, 0x2222 ori $12, $0, 326	@00003000: $ 2 <= 00003000 @00003004: $ 3 <= 00002fff @00003008: $ 4 <= 00000001 @0000300c: $ 5 <= 00003000 @00003014: $10 <= 00000005 @0000301c: $12 <= 0000302c @00003024: $ 1 <= 00000009 @0000302c: $12 <= 00000146
Tuse_rs=0，Tnew_E=DM，暂停	
ori $1, $0, 256 ori $6, $0, 256 ori $5, $0, 4	@00003000: $ 1 <= 00000100 @00003004: $ 6 <= 00000100 @00003008: $ 5 <= 00000004

sw $1, 4($5)
lw $10, 4($5) ##
beq $10, $6, label ##
nop
ori $20, $0, 2
label:
lui $4, 0x1234
	@0000300c: *00000008 <= 00000100
	@00003010: $10 <= 00000100
	@00003020: $ 4 <= 12340000

Tuse_rs=0，Tnew_E=PC（似乎这样的搭配只有延迟槽里跳转的情况）	
Tuse_rs=0，Tnew_M=ALU，转发	
ori $2, $0, 0x3000 ori $3, $0, 0x2fff ori $4, $0, 1 addu $5, $3, $4 ## ori $30, $0, 56 beq $5, $2, label ## ori $10, $0, 5 ori $11, $0, 6 label: ori $12, $0, 0x3034 ## ori $29, $0, 58 jr $12 ## ori $1, $0, 9 lui $9, 0x2222 ori $12, $0, 326	@00003000: $ 2 <= 00003000 @00003004: $ 3 <= 00002fff @00003008: $ 4 <= 00000001 @0000300c: $ 5 <= 00003000 @00003010: $30 <= 00000038 @00003018: $10 <= 00000005 @00003020: $12 <= 00003034 @00003024: $29 <= 0000003a @0000302c: $ 1 <= 00000009 @00003034: $12 <= 00000146
Tuse_rs=0，Tnew_M=DM，暂停	
ori $1, $0, 0x3038 ori $5, $0, 0x3038 ori $2, $0, 4 sw $1, -4($2)	@00003000: $ 1 <= 00003038 @00003004: $ 5 <= 00003038 @00003008: $ 2 <= 00000004 @0000300c: *00000000 <= 00003038

lw $10, -4($2) ##
ori $3, $0, 123
beq $10, $5, label ##
nop
ori $25, $0, 256
label:
lw $6, -4($2) ##
lui $26, 0x1234
jr $6 ##
nop
ori $23, $0, 125
ori $24, $0, 156
	@00003010: $10 <= 00003038
	@00003014: $ 3 <= 0000007b
	@00003024: $ 6 <= 00003038
	@00003028: $26 <= 12340000
	@00003038: $24 <= 0000009c

Tuse_rs=0，Tnew_M=PC，转发	
ori $5, $0, 0x3010 ori $1, $0, 256 jal label ## nop ori $2, $0, 255 label: beq $31, $5, label2 ## nop ori $6, $0, 247 label2: lui $9, 0x1234	@00003000: $ 5 <= 00003010 @00003004: $ 1 <= 00000100 @00003008: $31 <= 00003010 @00003020: $ 9 <= 12340000
Tuse_rs=0，Tnew_W=ALU，转发	
ori $2, $0, 0x3000 ori $3, $0, 0x2fff ori $4, $0, 1 addu $5, $3, $4 ##	@00003000: $ 2 <= 00003000 @00003004: $ 3 <= 00002fff @00003008: $ 4 <= 00000001 @0000300c: $ 5 <= 00003000

ori $27, $0, 256
ori $22, $0, 355
beq $5, $2, label ##
ori $10, $0, 5
ori $11, $0, 6
label:
ori $12, $0, 0x303c ##
addu $23, $2, $4
subu $24, $23, $4
jr $12 ##
ori $1, $0, 9
lui $9, 0x2222
ori $12, $0, 326
	@00003010: $27 <= 00000100
	@00003014: $22 <= 00000163
	@0000301c: $10 <= 00000005
	@00003024: $12 <= 0000303c
	@00003028: $23 <= 00003001
	@0000302c: $24 <= 00003000
	@00003034: $ 1 <= 00000009
	@0000303c: $12 <= 00000146

Tuse_rs=0，Tnew_W=DM，转发	

ori $1, $0, 0x3040
ori $5, $0, 0x3040
ori $2, $0, 4
sw $1, -4($2)
lw $10, -4($2) ##
ori $16, $0, 258
ori $3, $0, 123
beq $10, $5, label ##
nop
ori $25, $0, 256
label:
lw $6, -4($2) ##
lui $26, 0x1234
addu $17, $16, $3
jr $6 ##
	@00003000: $ 1 <= 00003040
	@00003004: $ 5 <= 00003040
	@00003008: $ 2 <= 00000004
	@0000300c: *00000000 <= 00003040
	@00003010: $10 <= 00003040
	@00003014: $16 <= 00000102
	@00003018: $ 3 <= 0000007b
	@00003028: $ 6 <= 00003040
	@0000302c: $26 <= 12340000
	@00003030: $17 <= 0000017d
	@00003040: $24 <= 0000009c

nop ori $23, $0, 125 ori $24, $0, 156	
Tuse_rs=0，Tnew_W=PC，转发	
ori $5, $0, 0x3010 ori $1, $0, 256 jal label ## nop ori $2, $0, 255 label: ori $10, $0, 0x1234 beq $31, $5, label2 ## nop ori $6, $0, 247 label2: lui $9, 0x1234	@00003000: $ 5 <= 00003010 @00003004: $ 1 <= 00000100 @00003008: $31 <= 00003010 @00003014: $10 <= 00001234 @00003024: $ 9 <= 12340000
Tuse_rs=1，Tnew_E=ALU，转发	
ori $1, $0, 256 ori $2, $0, 255 ## addu $4, $2, $1 ##	@00003000: $ 1 <= 00000100 @00003004: $ 2 <= 000000ff @00003008: $ 4 <= 000001ff
Tuse_rs=1，Tnew_E=DM，暂停	

ori $1, $0, 4
ori $2, $0, 0x1234
sw $2, 4($1)
lw $3, 4($1) ##
addu $4, $3, $1 ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001234
	@00003008: *00000008 <= 00001234
	@0000300c: $ 3 <= 00001234
	@00003010: $ 4 <= 00001238

Tuse_rs=1，Tnew_E=PC，转发	
ori $1, $0, 154 ori $2, $0, 111	@00003000: $ 1 <= 0000009a @00003004: $ 2 <= 0000006f

jal label ##	@00003008: $31 <= 00003010
addu $3, $31, $1 ##	@0000300c: $ 3 <= 000030aa
addu $4, $1, $2	@00003014: $ 5 <= 00000093
label:	
ori $5, $0, 147	
Tuse_rs=1，Tnew_M=ALU，转发	
ori $1, $0, 256	@00003000: $ 1 <= 00000100
ori $2, $0, 255 ##	@00003004: $ 2 <= 000000ff
ori $3, $0, 289	@00003008: $ 3 <= 00000121
addu $4, $2, $1 ##	@0000300c: $ 4 <= 000001ff
Tuse_rs=1，Tnew_M=DM，转发	
ori $1, $0, 4	@00003000: $ 1 <= 00000004
ori $2, $0, 0x1234	@00003004: $ 2 <= 00001234
sw $2, 4($1)	@00003008: *00000008 <= 00001234
lw $3, 4($1) ##	@0000300c: $ 3 <= 00001234
ori $25, $0, 156	@00003010: $25 <= 0000009c
addu $4, $3, $1 ##	@00003014: $ 4 <= 00001238
Tuse_rs=1，Tnew_M=PC，转发	
ori $1, $0, 154	@00003000: $ 1 <= 0000009a
ori $2, $0, 111	@00003004: $ 2 <= 0000006f
jal label ##	@00003008: $31 <= 00003010
addu $4, $1, $2	@0000300c: $ 4 <= 00000109
lui $8, 0x1235	@00003014: $ 3 <= 000030aa
label:	@00003018: $ 5 <= 00000093
addu $3, $31, $1 ##	
ori $5, $0, 147	
Tuse_rs=1，Tnew_W=ALU，转发	
ori $1, $0, 256	@00003000: $ 1 <= 00000100
ori $2, $0, 255 ##	@00003004: $ 2 <= 000000ff

lui $5, 0x1256 ori $3, $0, 289 addu $4, $2, $1 ##	@00003008: $ 5 <= 12560000 @0000300c: $ 3 <= 00000121 @00003010: $ 4 <= 000001ff
Tuse_rs=1，Tnew_W=DM，转发	

ori $1, $0, 4
ori $2, $0, 0x1234
sw $2, 4($1)
lw $3, 4($1) ##
ori $25, $0, 156
lui $26, 0x1475
addu $4, $3, $1 ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001234
	@00003008: *00000008 <= 00001234
	@0000300c: $ 3 <= 00001234
	@00003010: $25 <= 0000009c
	@00003014: $26 <= 14750000
	@00003018: $ 4 <= 00001238

Tuse_rs=1，Tnew_W=PC，转发	

ori $1, $0, 154
ori $2, $0, 111
jal label ##
addu $4, $1, $2
lui $8, 0x1235
label:
lui $9, 0x58
addu $3, $31, $1 ##
ori $5, $0, 147
	@00003000: $ 1 <= 0000009a
	@00003004: $ 2 <= 0000006f
	@00003008: $31 <= 00003010
	@0000300c: $ 4 <= 00000109
	@00003014: $ 9 <= 00580000
	@00003018: $ 3 <= 000030aa
	@0000301c: $ 5 <= 00000093

Tuse_rt=0，Tnew_E=ALU，暂停	
ori $2, $0, 0x3000 ori $3, $0, 0x2fff ori $4, $0, 1 addu $5, $3, $4 ## beq $2, $5, label ## ori $10, $0, 5	@00003000: $ 2 <= 00003000 @00003004: $ 3 <= 00002fff @00003008: $ 4 <= 00000001 @0000300c: $ 5 <= 00003000 @00003014: $10 <= 00000005 @0000301c: $12 <= 0000302c

ori $11, $0, 6 label: ori $12, $0, 0x302c	
Tuse_rt=0，Tnew_E=ALU，暂停	

ori $1, $0, 256
ori $6, $0, 256
ori $5, $0, 4
sw $1, 4($5)
lw $10, 4($5) ##
beq $6, $10, label ##
nop
ori $20, $0, 2
label:
lui $4, 0x1234
	@00003000: $ 1 <= 00000100
	@00003004: $ 6 <= 00000100
	@00003008: $ 5 <= 00000004
	@0000300c: *00000008 <= 00000100
	@00003010: $10 <= 00000100
	@00003020: $ 4 <= 12340000

Tuse_rt=0，Tnew_E=PC（似乎这样的搭配只有延迟槽里跳转的情况）	
Tuse_rt=0，Tnew_M=ALU，转发	
ori $2, $0, 0x3000 ori $3, $0, 0x2fff ori $4, $0, 1 addu $5, $3, $4 ## ori $30, $0, 56 beq $2, $5, label ## ori $10, $0, 5 ori $11, $0, 6 label: ori $12, $0, 0x3034	@00003000: $ 2 <= 00003000 @00003004: $ 3 <= 00002fff @00003008: $ 4 <= 00000001 @0000300c: $ 5 <= 00003000 @00003010: $30 <= 00000038 @00003018: $10 <= 00000005 @00003020: $12 <= 00003034
Tuse_rt=0，Tnew_M=DM，暂停	
ori $1, $0, 0x3038 ori $5, $0, 0x3038	@00003000: $ 1 <= 00003038 @00003004: $ 5 <= 00003038

ori $2, $0, 4
sw $1, -4($2)
lw $10, -4($2) ##
ori $3, $0, 123
beq $5, $10, label ##
nop
ori $25, $0, 256
label:
ori $27, $0, 145
	@00003008: $ 2 <= 00000004
	@0000300c: *00000000 <= 00003038
	@00003010: $10 <= 00003038
	@00003014: $ 3 <= 0000007b
	@00003024: $27 <= 00000091

Tuse_rt=0，Tnew_M=PC，转发	
ori $5, $0, 0x3011 ori $1, $0, 256 jal label ## nop ori $2, $0, 255 label: beq $5, $31, label2 ## nop ori $6, $0, 247 label2: lui $9, 0x1234	@00003000: $ 5 <= 00003011 @00003004: $ 1 <= 00000100 @00003008: $31 <= 00003010 @0000301c: $ 6 <= 000000f7 @00003020: $ 9 <= 12340000
Tuse_rt=0，Tnew_W=ALU，转发	
ori $2, $0, 0x3000 ori $3, $0, 0x2fff ori $4, $0, 1 addu $5, $3, $4 ## ori $27, $0, 256 ori $22, $0, 355 beq $2, $5, label ##	@00003000: $ 2 <= 00003000 @00003004: $ 3 <= 00002fff @00003008: $ 4 <= 00000001 @0000300c: $ 5 <= 00003000 @00003010: $27 <= 00000100 @00003014: $22 <= 00000163 @0000301c: $10 <= 00000005

ori $10, $0, 5 ori $11, $0, 6 label: addu $23, $2, $4	@00003024: $23 <= 00003001
Tuse_rt=0，Tnew_W=DM，转发	

ori $1, $0, 0x3040
ori $5, $0, 0x3040
ori $2, $0, 4
sw $1, -4($2)
lw $10, -4($2) ##
ori $16, $0, 258
ori $3, $0, 123
beq $5, $10, label ##
nop
ori $25, $0, 256
label:
lw $6, -4($2)
	@00003000: $ 1 <= 00003040
	@00003004: $ 5 <= 00003040
	@00003008: $ 2 <= 00000004
	@0000300c: *00000000 <= 00003040
	@00003010: $10 <= 00003040
	@00003014: $16 <= 00000102
	@00003018: $ 3 <= 0000007b
	@00003028: $ 6 <= 00003040

Tuse_rt=0，Tnew_W=PC，转发	
ori $5, $0, 0x3010 ori $1, $0, 256 jal label ## nop ori $2, $0, 255 label: ori $10, $0, 0x1234 beq $5, $31, label2 ## nop ori $6, $0, 247 label2:	@00003000: $ 5 <= 00003010 @00003004: $ 1 <= 00000100 @00003008: $31 <= 00003010 @00003014: $10 <= 00001234 @00003024: $ 9 <= 12340000

lui $9, 0x1234	
Tuse_rt=1，Tnew_E=ALU，转发	
ori $1, $0, 256 ori $2, $0, 255 ## addu $4, $1, $2 ##	@00003000: $ 1 <= 00000100 @00003004: $ 2 <= 000000ff @00003008: $ 4 <= 000001ff
Tuse_rt=1，Tnew_E=DM，暂停	

ori $1, $0, 4
ori $2, $0, 0x1234
sw $2, 4($1)
lw $3, 4($1) ##
addu $4, $1, $3 ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001234
	@00003008: *00000008 <= 00001234
	@0000300c: $ 3 <= 00001234
	@00003010: $ 4 <= 00001238

Tuse_rt=1，Tnew_E=PC，转发	

ori $1, $0, 154
ori $2, $0, 111
jal label ##
addu $3, $1, $31 ##
addu $4, $1, $2
label:
ori $5, $0, 147
	@00003000: $ 1 <= 0000009a
	@00003004: $ 2 <= 0000006f
	@00003008: $31 <= 00003010
	@0000300c: $ 3 <= 000030aa
	@00003014: $ 5 <= 00000093

Tuse_rt=1，Tnew_M=ALU，转发	
ori $1, $0, 256 ori $2, $0, 255 ## ori $3, $0, 289 addu $4, $1, $2 ##	@00003000: $ 1 <= 00000100 @00003004: $ 2 <= 000000ff @00003008: $ 3 <= 00000121 @0000300c: $ 4 <= 000001ff
Tuse_rt=1，Tnew_M=DM，转发	

ori $1, $0, 4
ori $2, $0, 0x1234
sw $2, 4($1)
lw $3, 4($1) ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001234
	@00003008: *00000008 <= 00001234
	@0000300c: $ 3 <= 00001234

ori $25, $0, 156 addu $4, $1, $3 ##	@00003010: $25 <= 0000009c @00003014: $ 4 <= 00001238
Tuse_rt=1，Tnew_M=PC，转发	

ori $1, $0, 154
ori $2, $0, 111
jal label ##
addu $4, $1, $2
lui $8, 0x1235
label:
addu $3, $31, $1 ##
ori $5, $0, 147
	@00003000: $ 1 <= 0000009a
	@00003004: $ 2 <= 0000006f
	@00003008: $31 <= 00003010
	@0000300c: $ 4 <= 00000109
	@00003014: $ 3 <= 000030aa
	@00003018: $ 5 <= 00000093

Tuse_rt=1，Tnew_W=ALU，转发	
ori $1, $0, 256 ori $2, $0, 255 ## lui $5, 0x1256 ori $3, $0, 289 addu $4, $1, $2 ##	@00003000: $ 1 <= 00000100 @00003004: $ 2 <= 000000ff @00003008: $ 5 <= 12560000 @0000300c: $ 3 <= 00000121 @00003010: $ 4 <= 000001ff
Tuse_rt=1，Tnew_W=DM，转发	

ori $1, $0, 4
ori $2, $0, 0x1234
sw $2, 4($1)
lw $3, 4($1) ##
ori $25, $0, 156
lui $26, 0x1475
addu $4, $1, $3 ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001234
	@00003008: *00000008 <= 00001234
	@0000300c: $ 3 <= 00001234
	@00003010: $25 <= 0000009c
	@00003014: $26 <= 14750000
	@00003018: $ 4 <= 00001238

Tuse_rt=1，Tnew_W=PC，转发	
ori $1, $0, 154 ori $2, $0, 111 jal label ##	@00003000: $ 1 <= 0000009a @00003004: $ 2 <= 0000006f @00003008: $31 <= 00003010

addu $4, $1, $2
lui $8, 0x1235
label:
lui $9, 0x58
addu $3, $1, $31 ##
ori $5, $0, 147
	@0000300c: $ 4 <= 00000109
	@00003014: $ 9 <= 00580000
	@00003018: $ 3 <= 000030aa
	@0000301c: $ 5 <= 00000093

Tuse_rt=2，Tnew_E=ALU，转发	

ori $1, $0, 4
ori $5, $0, 256
addu $3, $1, $1 ##
sw $3, 4($5) ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 5 <= 00000100
	@00003008: $ 3 <= 00000008
	@0000300c: *00000104 <= 00000008

Tuse_rt=2，Tnew_E=DM，转发	

ori $1, $0, 4
ori $2, $0, 0x1243
sw $2, -4($1)
lw $3, -4($1) ##
sw $3, 8($1) ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001243
	@00003008: *00000000 <= 00001243
	@0000300c: $ 3 <= 00001243
	@00003010: *0000000c <= 00001243

Tuse_rt=2，Tnew_E=PC，转发	

ori $1, $0, 4
jal label ##
sw $31, 4($1) ##
addu $3, $1, $1
label:
ori $5, $0, 156
	@00003000: $ 1 <= 00000004
	@00003004: $31 <= 0000300c
	@00003008: *00000008 <= 0000300c
	@00003010: $ 5 <= 0000009c

Tuse_rt=2，Tnew_M=ALU，转发	
ori $1, $0, 4 ori $5, $0, 256 addu $3, $1, $1 ## ori $4, $0, 28	@00003000: $ 1 <= 00000004 @00003004: $ 5 <= 00000100 @00003008: $ 3 <= 00000008 @0000300c: $ 4 <= 0000001c

sw $3, 4($5) ##	@00003010: *00000104 <= 00000008
Tuse_rt=2，Tnew_M=DM，转发	

ori $1, $0, 4
ori $2, $0, 0x1243
sw $2, -4($1)
lw $3, -4($1) ##
ori $9, $0, 156
sw $3, 8($1) ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001243
	@00003008: *00000000 <= 00001243
	@0000300c: $ 3 <= 00001243
	@00003010: $ 9 <= 0000009c
	@00003014: *0000000c <= 00001243

Tuse_rt=2，Tnew_M=PC，转发	

ori $1, $0, 4
jal label ##
nop
addu $3, $1, $1
label:
sw $31, 4($1) ##
ori $5, $0, 156
	@00003000: $ 1 <= 00000004
	@00003004: $31 <= 0000300c
	@00003010: *00000008 <= 0000300c
	@00003014: $ 5 <= 0000009c

Tuse_rt=2，Tnew_W=ALU，转发	

ori $1, $0, 4
ori $5, $0, 256
addu $3, $1, $1 ##
ori $4, $0, 28
addu $7, $1, $5
sw $3, 4($5) ##
	@00003000: $ 1 <= 00000004
	@00003004: $ 5 <= 00000100
	@00003008: $ 3 <= 00000008
	@0000300c: $ 4 <= 0000001c
	@00003010: $ 7 <= 00000104
	@00003014: *00000104 <= 00000008

Tuse_rt=2，Tnew_W=DM，转发	

ori $1, $0, 4
ori $2, $0, 0x1243
sw $2, -4($1)
lw $3, -4($1) ##
ori $9, $0, 156
	@00003000: $ 1 <= 00000004
	@00003004: $ 2 <= 00001243
	@00003008: *00000000 <= 00001243
	@0000300c: $ 3 <= 00001243
	@00003010: $ 9 <= 0000009c

addu $5, $1, $2 sw $3, 8($1) ##	@00003014: $ 5 <= 00001247 @00003018: *0000000c <= 00001243
Tuse_rt=2，Tnew_W=PC，转发	

ori $1, $0, 4
jal label ##
nop
addu $3, $1, $1
label:
addu $8, $1, $1
sw $31, 4($1) ##
ori $5, $0, 156
	@00003000: $ 1 <= 00000004
	@00003004: $31 <= 0000300c
	@00003010: $ 8 <= 00000008
	@00003014: *00000008 <= 0000300c
	@00003018: $ 5 <= 0000009c

三、思考题

（一）流水线冒险

1. 在采用本节所述的控制冒险处理方式下，PC 的值应当如何被更新？请从数据通路和控制信号两方面进行说明。

答：

从数据通路方面，由于寄存器值比较提前至位于 D 级，考虑将 NPC 部件同样置于 D 级，这样便于保证比较与跳转操作的接续性，此处注意 beq 判断为 0 时 npc 应输出 pc4+4，jal 等写入寄存器的指令地址也应该是 pc4+4。由于延迟槽的使用，在未产生跳转需求时应保持 pc 在每个周期+4 的操作，因此我在 F 级设置了 add4 部件，将 pc 输出值+4 后再输入到 pc 中。当产生跳转需求时，PC 的输入端口面临 pc4 和 npc 两个选择，因此我设置了 MUX_PcSel 多路选择器用于选择次指令地址。由于需要暂停控制，还需要从暂停控制器接到 PC 的通路，接入使能信号用于控制暂停。

从控制信号方面，将 D 级指令信号译码得到 NPC 操作的控制信号，目前分为三种（`bType_npc,`jType_npc,`rType_npc），在 MUX_PcSel 多路选择器中需要译码得到的选择信号，`pc4_pc 表示使用 pc+4（此处 pc 是 F 级的 pc），`npc 表

示使用跳转指令地址（此处 pc 是 D 级的 pc）。PC 中的使能信号为暂停信号取反，当使能信号为 1 时 PC 正常工作，为 0 时 PC 值冻结。

2. 对于 jal 等需要将指令地址写入寄存器的指令，为什么需要回写 PC+8？

答：

由于使用了延迟槽，CPU 会自动执行跳转指令后一个指令，该指令是在编译过程中编译器优化加入的，因此返回至跳转位置时应执行的是跳转指令后第二条指令，即 pc+8，所以回写入寄存器的指令应是 pc+8。

（二）数据冒险的分析

1. 为什么所有的供给者都是存储了上一级传来的各种数据的流水级寄存器，而不是由 ALU 或者 DM 等部件来提供数据？

答：

因为可能产生死循环：若供给者为 ALU，需求者也为 ALU，此时 ALU 的输出端口与输入端口相接产生错误。而所有供给者都设置为流水级寄存器在我看来是为了功能上的协调统一，否则相同的效率下暂停与转发的控制将更加复杂难以操作。

（三）AT 法处理流水线数据冒险

1. 如果不采用已经转发过的数据，而采用上一级中的原始数据，会出现怎样的问题？试列举指令序列说明这个问题。

答：

若上一级的数据来自 W 级转发而之后并未使用转发过的数据，则可能会导致数据的错误。比如序列：

```
ori $1, $0, 0x1234
addu $1, $2, $3  （$2+$3=0x5678）
xxx
xxx
addu $5, $1, $2
```

当第二条指令位于 W 级时，第五条指令位于 D 级，此时原始数据 RD1 为 0x1234，转发后数据 V1 为 0x5678，下一个时钟周期，若使用原始数据，第五条指令到达 E 级，此时参与计算的$1 为 0x1234，但实际上第二条指令刚刚写入了

寄存器，$1 值实际为 0x5678，因此产生错误。

2. 我们为什么要对 GPR 采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢？

答：

若某一时钟周期，GRF 需要写入寄存器，同时有新的指令需要在 D 级读取该寄存器的值，时钟上升沿到来时，两操作并行导致读取的寄存器值仍为原来的值，可能产生错误。根据理论课知识，如果不采用内部转发，似乎需要使用反时钟，使得用于写入的正时钟出于下降沿时，用于读取的反时钟出于上升沿，此时再读取可得到已写入的寄存器值（但似乎这样过于复杂，且时钟沿的延迟不易把控？因此放弃这一方案采用内部转发）。

3. 为什么 0 号寄存器需要特殊处理？

答：

因为 0 号寄存器无法写入，且任何时候读出均为 0，不涉及数据冲突与冒险，若未进行特殊处理，当某一指令需要写入 0 寄存器，且后面指令需要读取 0 号寄存器时，根据普通转发机制，产生转发，将要写入的值转发，导致参与运算或比较的数据不是应读出的 0，可能产生错误。

4. 什么是“最新产生的数据”？

答：

我的理解是在目前时序中最后写入寄存器的值，也即在使用数据的部件之后离它越近，产生的数据越新，而流水下来的数据最“旧”。

5. 在 AT 方法讨论转发条件的时候，只提到了“供给者需求者的 A 相同，且不为 0”，但在 CPU 写入 GRF 的时候，是有一个 we 信号来控制是否要写入的。为何在 AT 方法中不需要特判 we 呢？为了用且仅用 A 和 T 完成转发，在翻译出 A 的时候，要结合 we 做什么操作呢？

答：

我的理解是供给者既然已经称为了供给者，说明指令在它的位置上产生的值在之后一定会有写入寄存器的操作（若不被转发覆盖），因此不需要写使能的特判。为了仅用 AT 完成转发，我认为在翻译 A 时，若 WE 为 1，则照常翻译 A，若 WE 为 0，则直接将 A 翻译成 0，即写入 0 寄存器，在判断转发时也将直接被

判定为不转发。但其实我的设计中并没有使用此方法，而是在 T 的翻译中加入了写使能的考虑，由于 Tnew 代表产生写入寄存器值的部件，我将 ALU、PC、DM 三个部件信号宏定义为非 0 值，对于不写入寄存器的指令，其 Tnew 记为 0，在选择转发内容时，0 即代表不转发，从而完成转发的控制。

（四）在线测试相关说明

1. 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。此思考题请同学们结合自己测试 CPU 使用的具体手段，按照自己的实际情况进行回答。

答：

在实验中，我的测试是根据策略矩阵进行构造的，在文档的前面列出了每种可能的一个典型测试样例。针对每个 Tuse 和 Tnew，有固定数目的指令组合，我根据这些指令组合构造了大量测试样例，但实际上，每个测试样例只能测试一种跳转或暂停操作，由于水平欠缺，不会进行自动化测试，无法保证测试的操作覆盖每个寄存器，这是本实验的不足之处。此外，考虑到之后需要添加大量指令，手动构造测试样例工作将变得异常繁重，或将带来巨大隐患。