计算机组成原理实验报告

16061007 王文珺

一、 CPU 设计文档

（一） 数据通路设计

数据通路的设计和 p5 基本没有任何区别，仍然为构建表格，在表头写出每一个功能模块，以及它们的数据输入，由于笔者在数据通路上仅仅增加了 DM 前后的扩展模块，用于搭配 load 和 store 类指令使用，以及在 ALU 的模块并列增加了乘除模块，并且在进入 M 级流水时加入了一个多选器来选择 ALU 计算结果或是从 HI 和 LO 寄存器读出的数据，其他的数据通路仍然为下表所示。

级别	部件	输入	lw	sw	addu	subu	ori	beq	j	jal	jr	MUX	0	1	2
IF级部件	PC		ADD4	ADD4	ADD4	ADD4	ADD4	ADD4/NPC	NPC	NPC	RF.RD1	MUX_PC	ADD4	NPC	RF.RD1(MF_PC_F)
	ADD4		PC	PC	PC	PC	PC	PC	PC	PC	PC				
	IM		PC	PC	PC	PC	PC	PC	PC	PC					
F/D级流水线寄存器	IR		IM	IM	IM	IM	IM	IM	IM	IM	IM				
	NPC		ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4					
D级部件	RF	A1	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D			IR[rs]@D				
		A2		IR[rt]@D	IR[rt]@D	IR[rt]@D		IR[rt]@D							
	EXT		IR[116]@I	IR[116]@D			IR[116]@D								
	CMP	D1						RF.RD1				MF_CMP1_D			
		D2						RF.RD2				MF_CMP2_D			
	NPC	PC4						NPC@D	NPC@D	NPC@D					
		126						IR[116]@I	IR[126]@I	IR[126]@D					
D/E级流水线寄存器	V1		RF.RD1	RF.RD1	RF.RD1	RF.RD1	RF.RD1					MF_CMP1_D			
	V2			RF.RD2	RF.RD2	RF.RD2						MF_CMP2_D			
	A1		IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D	IR[rs]@D								
	A2			IR[rt]@D	IR[rt]@D	IR[rt]@D									
	A3		IR[rt]@D		IR[rd]@D	IR[rd]@D	IR[rt]@D			31		MUX_A3_E	IR[rt]@D	IR[rd]@D	31
	E32		EXT	EXT			EXT								
	PC4									NPC@D					
E级部件	ALU	A	V1@E	V1@E	V1@E	V1@E	V1@E					MF_ALU_A_E			
		B	E32@E	E32@E	V2@E	V2@E	E32@E					MUX_ALU_B	V2@E(MF_ALU_B_E)	E32@E	
E/M级流水线寄存器	V2			V2@E								MF_V2_M			
	A2			A2@E											
	A0		ALU	ALU	ALU	ALU	ALU								
	A3		A3@E		A3@E	A3@E	A3@E			A3@E					
	PC4									PC4@E					
M级部件	DM	A	A0@M	A0@M	A0@M	A0@M	A0@M								
		WD		V2@M								MF_WD_M			
M/W级流水线寄存器	A3		A3@M		A3@M	A3@M	A3@M								
	PC4									PC4@M					
	A0				A0@M	A0@M	A0@M								
	DR		DM												
W级部件	RF	A3	A3@W		A3@W	A3@W	A3@W			A3@W					
		WD	DR@W		A0@W	A0@W	A0@W			PC4@W		MUX_RF_WD	DR@W	A0@W	PC4@W

增加的数据通路如下图所示：

```
MultDiv MD(
.clk(clk),
.reset(reset),
.Data1(ALUA),
.Data2(ALUB),
.HI(HI),
.LO(LO),
.Busy(Busy),
.Start(Start),
.Instr(Instr_E)
);

MUX_32b_3 MUX_MDALU(
.in_1(AO),
.in_2(HI),
.in_3(LO),
.select(MD_Sel),
.out(FAO)
);

BE_EXT BEEXT(
.instr(Instr_M),
.ADDR(AO_M[1:0]),
.BE(BE)
);

Data_EXT DATAEXT(
.instr(Instr_W),
.Data(DM_W),
.A(Aout),
.Dout(Dout)
);
```

其中的 MultDiv 是乘除模块，数据来源和 ALU 相同。BE_EXT 负责处理写入 DM 的数据，DATA_EXT 负责处理 DM 读出的数据。

对于转发和暂停机制，由于我们之前使用的需求者供给者模型同样适用，而且新加入的

指令也没有产生新类型的转发，所以所有的指令都可以对应到已有的各种转发和暂停中。下图均为 p5 的策略矩阵。

指令	Tuse		功能部件	Tnew		
	rs	rt		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				
jal			PC	0	0	0
jr	0					
j						
	{0, 1}	{0, 1, 2}				

rs	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	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

Tuse_rs0=beq+jr

Tuse_rs1=addu+subu+ori+lw+sw

rt	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	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

Tuse_rt0=beq

Tuse_rt1=addu+subu

Tuse_rt2=sw

新增指令的具体的实现转发和暂停控制信号我们在控制器设计模块进行详细阐述，接下来给出数据通路中所有部件的端口说明。

IF：

序号	端口名称	I\O	功能名称	功能描述
1	Clk	I	时钟	提供时钟信号
2	Reset	I	复位	当复位信号有效时，PC 被设置为 0x00003000
3	Branch	I	跳转	当 Branch 信号有效时，将跳转地址赋给 PC
4	Stall	I	暂停	当 Stall 信号有效时，冻结 PC
5	nextPC	I	跳转地址	提供跳转地址
6	Instr	O	指令	输出正在执行的指令
7	ADD4	O	下条地址	非跳转情况下的下条指令地址
8	Bfail	I	B 类指令失败	当 bfail 有效时，pc 下条地址为正常+4

D_PIPE:

序号	端口名称	I\O	功能名称	功能描述

1	Clk	I	时钟	提供时钟信号
2	Reset	I	复位	当复位信号有效时，复位清零 D 级流水寄存器
3	Instr	I	指令	将上级指令存入 D 级流水寄存器
4	Stall	I	暂停	当 Stall 信号有效时，冻结 D 级流水寄存器
5	ADD4	I	下条地址	将上级指令的下条地址存入 D 级流水寄存器
6	Instr_D	O	D 级指令	输出 D 级流水寄存器的指令
7	PC4_D	O	D 级下条地址	输出 D 级流水寄存器的指令的下条地址

NPC:

序号	端口名称	I\0	功能名称	功能描述
1	Imm16	I	16 位立即数	提供 16 位立即数
2	Imm26	I	26 位立即数	提供 26 位立即数
3	PC4	I	当前 PC+4	提供当前的 PC+4
4	jrpc	I	跳转地址	提供从寄存器读出的地址
5	NPC_Sel	I	选择信号	0000/beq 跳转，0001/j，jal 跳转，0010/jr 跳转 0011/bne 跳转，0100/blez 跳转，0101/bltz 跳转 0110/bgez 跳转，0111/bgtz 跳转
6	CMPResult	I	比较结果	0/两数据不相等，1/两数据相等
7	OVERZERO	I	Rs 大于 0	用于判断 b 类指令跳转成功与否
8	BELOWZERO	I	Rs 小于 0	用于判断 b 类指令跳转成功与否
9	Bfail	O	B 类指令失败	指令为 b 类但是跳转条件无效时为 1
10	nPC	O	下条地址	输出下条地址

GRF:

序号	端口名称	I\0	功能名称	功能描述
1	Rreg1	I	寄存器 1 选择	通过多路选择器选择寄存器 1
2	Rreg2	I	寄存器 2 选择	通过多路选择器选择寄存器 2
3	Wreg	I	选择写入寄存器	当控制信号 RegWrite 有效时， 在时钟上升沿将 Wdata 中的数据写入 Wreg 通过 Decoder 选择的寄存器中，
4	RegWrite	I	写入控制	
5	Wdata	I	写入的数据	
6	Rdata1	O	数据 1 输出	输出寄存器 1 的数据
7	Rdata2	O	数据 2 输出	输出寄存器 2 的数据
8	clk	I	时钟	时钟信号
9	reset	I	复位	当 reset 信号有效时，将所有寄存器值清零
10	PrePC	I	当前地址	用于输出当前指令地址

CMP:

序号	端口名称	I\0	功能名称	功能描述
1	CMPD1	I	比较数 1	提供第一个比较数
2	CMPD2	I	比较数 2	提供第二个比较数
3	CMPResult	O	比较结果	比较结果：0/不相等 1/相等
4	OVERZERO	O	Rs 大于 0	Rs 大于 0 时此信号为 1
5	BELOWZERO	O	Rs 小于 0	Rs 小于 0 的此信号为 1

EXT：

序号	端口名称	I\0	功能名称	功能描述
1	Imm16	I	16 位立即数	提供被扩展的立即数
2	ExtOp	I	扩展信号	0/无符号扩展 1/符号扩展
3	Imm32	O	32 位立即数	输出扩展后的立即数

E_PIPE：

序号	端口名称	I\0	功能名称	功能描述
1	Clk	I	时钟	提供时钟信号
2	Reset	I	复位	当复位信号有效时，复位清零 E 级流水寄存器
3	Instr_D	I	D 级指令	将 D 级指令存入 E 级流水寄存器
4	Stall	I	暂停	当 stall 信号有效，复位清零 E 级流水寄存器
5	PC4_D	I	D 级下条地址	将 D 级下条地址存入 E 级流水寄存器
6	Resin	I	写入 res_e	将 res_e 的值存入 E 级流水寄存器
7	RF_RD_1	I	数据 1	将数据 1 存入 E 级流水寄存器
8	RF_RD_2	I	数据 2	将数据 2 存入 E 级流水寄存器
9	E32	I	32 位立即数	将 32 位立即数存入 E 级流水寄存器
10	MUX_A3	I	写入寄存器	将写入寄存器存入 E 级流水寄存器
11	Res_E	O	E 级 res_e	输出当前 E 级 res_e
12	Instr_E	O	E 级指令	输出当前 E 级指令
13	PC4_E	O	E 级下条地址	输出当前 E 级下条地址
14	V1_E	O	E 级数据 1	输出当前 E 级数据 1
15	V2_E	O	E 级数据 2	输出当前 E 级数据 2
16	A1_E	O	E 级寄存器 1	输出当前 E 级寄存器 1
17	A2_E	O	E 级寄存器 2	输出当前 E 级寄存器 2
18	A3_E	O	E 级写入寄存器	输出当前 E 级写入寄存器
19	E32_E	O	E 级 32 位立即数	输出当前 E 级 32 位立即数
20	Shamt	O	移位	输出用于移位的指令段

ALU:

序号	端口名称	I\0	功能名称	功能描述
1	A	I	数据 1	提供运算的数据 1
2	B	I	数据 2	提供运算的数据 2
3	Shamt	I	移位	提供移位的数据
4	ALUOp	I	计算信号	0000/加法，0001/减法，0010/或，0011/加载至高 16 位，0100/逻辑左移，0101/逻辑可变左移，0110/算术右移，0111/算术可变右移，1000/逻辑右移，1001/逻辑可变右移，1010/与，1011/异或，1100/或非，1101/有符号小于置 1，1110/无符号小于置 1
5	C	O	计算结果	输出计算结果

MultDiv:

序号	端口名称	I\0	功能名称	功能描述

1	Clk	I	时钟	提供时钟信号
2	Reset	I	复位	当复位信号有效时，复位清零 HI,LO 寄存器
3	Instr	I	指令	提供操作信号
4	Data1	O	数据 1	提供运算的数据 1
5	Data2	O	数据 2	提供运算的数据 2
6	HI	O	HI 寄存器	HI 寄存器的值
7	LO	O	LO 寄存器	LO 寄存器的值
8	Busy	O	正在计算	乘法持续 5 个周期，除法持续 10 个周期
9	Start	O	计算启动	识别到正在处理乘除法指令时有效 1 个周期

M_PIPE：

序号	端口名称	I\0	功能名称	功能描述
1	Clk	I	时钟	提供时钟信号
2	Reset	I	复位	当复位信号有效时，复位清零 M 级流水寄存器
3	Instr_E	I	E 级指令	将 E 级指令存入 M 级流水寄存器
4	PC4_E	I	E 级下条地址	将 E 级下条地址存入 M 级流水寄存器
5	res_E	I	E 级 res_e	将 res_e 的值存入 M 级流水寄存器
6	AO	I	计算结果	将 ALU 输出结果存入 M 级流水寄存器
7	V2_E	I	写入数据	将写入数据存入 M 级流水寄存器
8	A2_E	I	数据对应寄存器	将数据对应寄存器存入 M 级流水寄存器
9	A3_E	I	写入寄存器	将写入寄存器存入 M 级流水寄存器
10	res_M	O	M 级 res_m	输出当前 M 级 res_m
11	Instr_M	O	M 级指令	输出当前 M 级指令
12	PC4_M	O	M 级下条地址	输出当前 M 级下条地址
13	V2_M	O	M 级写入数据	输出当前 M 级写入数据
14	A2_M	O	M 级数据寄存器	输出当前 M 级数据对应寄存器
15	A3_M	O	M 级写入寄存器	输出当前 M 级写入寄存器
16	AO_M	O	M 级计算结果	输出当前 M 级计算结果

BE_EXT：

序号	端口名称	I\0	功能名称	功能描述
1	Instr	I	指令	通过该指令判断是否是 store 型指令
2	ADDR	I	地址低两位	提供地址低两位判断哪些字节需要写入
3	BE	O	写入信号	1111/sw 四个字节全部写入 1100/sh 写入到前两个字节 0011/sh 写入到后两个字节 1000/sb 写入到第一个字节 0100/sb 写入到第二个字节

				0010/sb 写入到第三个字节 0001/sb 写入到第四个字节

DM：

序号	端口名称	I\0	功能名称	功能描述
1	reset	I	复位	当复位信号有效时，DM 中的数据被全部清零
2	Address	I	32 位地址	利用 Splitter 选择 DM 中存储的某条数据
3	WriteData	I	写入的数据	当 MemWrite 信号有效时，将 WriteData 写入 Address 所选择的地址在 DM 中的位置
4	MemWrite	I	数据写入	
5	ReadData	O	读取的数据	输出 DM 被选择的位置所保存的数据
6	clk	I	时钟	时钟信号
7	PC	I	当前地址	用于输出当前指令地址

Data_EXT：

序号	端口名称	I\0	功能名称	功能描述
1	Instr	I	指令	通过该指令判断是否是 load 指令
2	Data	I	数据	提供 DM 读出的数据
3	A	I	地址低两位	通过该地址判断读出哪些字节
4	Dout	O	处理数据	经过处理的字节

W_PIPE:

序号	端口名称	I\0	功能名称	功能描述
1	Clk	I	时钟	提供时钟信号
2	Reset	I	复位	当复位信号有效时，复位清零 W 级流水寄存器
3	Instr_M	I	M 级指令	将 E 级指令存入 W 级流水寄存器
4	PC4_M	I	M 级下条地址	将 E 级下条地址存入 W 级流水寄存器
5	res_M	I	M 级 res_m	将 res_m 的值存入 W 级流水寄存器
6	AO_M	I	计算结果	将 ALU 输出结果存入 W 级流水寄存器
7	DM	I	读出数据	将 DM 读出数据存入 W 级流水寄存器
8	A3_M	I	写入寄存器	将写入寄存器存入 W 级流水寄存器
9	res_W	O	W 级 res_w	输出当前 W 级 res_w
10	Instr_W	O	W 级指令	输出当前 W 级指令
11	PC4_W	O	W 级下条地址	输出当前 W 级下条地址
12	A3_W	O	W 级写入寄存器	输出当前 W 级写入寄存器
13	AO_W	O	W 级计算结果	输出当前 W 级计算结果
14	DM_W	O	W 级读出数据	输出当前 W 级读出数据

（二） 控制器设计

控制器设计方面，主要由两种控制器组成，一种生成是正常的控制信号，一种生成转发和暂停信号。正常的控制信号以按照下表，通过和之前 project 一样的方法生成。相比较于 p5，p6 的控制信号仅仅增加了一个 AO_Sel 在 ALU 和 MultDiv 模块中选择相应的数据写入 M

级流水寄存器，并且对 NPC_Sel 信号位数进行的扩展以适应新增的各种指令。

	addu	subu	ori	lui	sw	lw	beq	jal	j	jr	nop	movz
RegWrite	1	1	1	1	0	1	0	1	0	0	0	1
A3_E_Sel	01	01	00	00	00	00	0	10	00	00	00	01
MemWrite	0	0	0	0	1	0	0	0	0	0	0	0
Branch	0	0	0	0	0	0	1	1	1	1	0	0
ALU_B_Sel	0	0	1	1	1	1	0	0	0	0	0	0
RF_WD_Sel	01	01	01	01	00	00	00	10	00	00	00	01
NPC_Sel	0000	0000	0000	0000	0000	0000	0000	0001	0001	0010	0000	0000
ExtOp	0	0	0	0	1	1	0	0	0	0	0	0
ALUOp	0000	0001	0010	0011	0000	0000	0000	0000	0000	0000	0000	0000
AO_Sel	00	00	00	00	00	00	00	00	00	00	00	00

	lb	lbu	lh	lhu	sb	sh	add	sub	mult	multu	div	divu
RegWrite	1	1	1	1	0	0	1	1	0	0	0	0
A3_E_Sel	00	00	00	00	00	00	01	01	00	00	00	00
MemWrite	0	0	0	0	1	1	0	0	0	0	0	0
Branch	0	0	0	0	0	0	0	0	0	0	0	0
ALU_B_Sel	1	1	1	1	1	1	0	0	0	0	0	0
RF_WD_Sel	00	00	00	00	00	00	01	01	00	00	00	00
NPC_Sel	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
ExtOp	1	1	1	1	1	1	0	0	0	0	0	0
ALUOp	0000	0000	0000	0000	0000	0000	0000	0001	0000	0000	0000	0000
AO_Sel	00	00	00	00	00	00	00	00	00	00	00	00

	sll	srl	sra	sllv	srlv	srav	and	xor	nor	or	addi	addiu
RegWrite	1	1	1	1	1	1	1	1	1	1	1	1
A3_E_Sel	01	01	01	01	01	01	01	01	01	01	00	00
MemWrite	0	0	0	0	0	0	0	0	0	0	0	0
Branch	0	0	0	0	0	0	0	0	0	0	0	0
ALU_B_Sel	0	0	0	0	0	0	0	0	0	0	1	1
RF_WD_Sel	01	01	01	01	01	01	01	01	01	01	01	01
NPC_Sel	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
ExtOp	0	0	0	0	0	0	0	0	0	0	0	0
ALUOp	0100	1000	0110	0101	1001	0111	1010	1011	1100	0010	0000	0000
AO_Sel	00	00	00	00	00	00	00	00	00	00	00	00

	andi	xori	slt	slti	sltu	sltiu	bne	blez	bltz	bgez	bgtz	jalr
RegWrite	1	1	1	1	1	1	0	0	0	0	0	1
A3_E_Sel	00	00	01	00	01	00	0	0	0	0	0	01
MemWrite	0	0	0	0	0	0	0	0	0	0	0	0
Branch	0	0	0	0	0	0	1	1	1	1	1	1
ALU_B_Sel	1	1	0	1	0	1	0	0	0	0	0	0
RF_WD_Sel	01	01	01	01	01	01	00	00	00	00	00	10
NPC_Sel	0000	0000	0000	0000	0000	0000	0011	0100	0101	0110	0111	0010
ExtOp	0	0	0	1	0	1	0	0	0	0	0	0
ALUOp	1010	1011	1101	1101	1110	1110	0000	0000	0000	0000	0000	0000
AO_Sel	00	00	00	00	00	00	00	00	00	00	00	00

	mfhi	mflo	mthi	mtlo
RegWrite	1	1	0	0
A3_E_Sel	01	01	00	00
MemWrite	0	0	0	0
Branch	0	0	0	0
ALU_B_Sel	0	0	0	0
RF_WD_Sel	01	01	00	00
NPC_Sel	0000	0000	0000	0000
ExtOp	0	0	0	0
ALUOp	0000	0000	0000	0000
AO_Sel	01	10	00	00

冲突控制器方面，继续采用高小鹏老师讲义中的方法，设置 E 级 Res 寄存器的初值，并进行简单传递，把每条新加的指令都通过其 rs 和 rt 的 Tuse 和 Tnew 把其判定条件并入原本已有的暂停和转发中即可。

通过策略矩阵，在 p5 构造的暂停条件的基础上并入所有新增指令。

```
assign Tuse_rs0=(op==`BEQ)||(op==`BNE)||(op==`special&&func==`JR)||(op==`special&&func==`JALR)||(op==`BLEZ)||(op==`BGTZ)||(op==`BLTZBGEZ&&(jump==`BLTZ||jump==`BGEZ));
assign Tuse_rs1=(op==`special&&func==`ADDU)||(op==`special&&func==`ADD)||(op==`special&&func==`AND)||(op==`special&&func==`OR)||(op==`special&&func==`NOR)
                ||(op==`special&&func==`XOR)||(op==`special&&func==`SUBU)||(op==`special&&func==`SLT)||(op==`special&&func==`SLTU)
                ||(op==`special&&func==`SUB)||(op==`ORI)||(op==`XORI)||(op==`ADDI)||(op==`ADDIU)||(op==`ANDI)||(op==`special&&func==`SLLV)
                ||(op==`special&&func==`SRAV)||(op==`SLTI)||(op==`SLTIU)||(op==`special&&func==`MULT)||(op==`special&&func==`MULTU)
                ||(op==`special&&func==`SRLV)||(op==`LUI)||(op==`special&&func==`DIV)||(op==`special&&func==`DIVU)||(op==`special&&func==`MTHI)||(op==`special&&func==`MTLO)
                ||(op==`SW)||(op==`SB)||(op==`SH)||(op==`LW)||(op==`LB)||(op==`LBU)||(op==`LH)||(op==`LHU)
                ||(op==`special&&func==`MOVZ);
assign Tuse_rt0=(op==`BEQ)||(op==`BNE)||(op==`special&&func==`MOVZ)||(op==`BLEZ)||(op==`BGTZ)||(op==`BLTZBGEZ&&(jump==`BLTZ||jump==`BGEZ));
assign Tuse_rt1=(op==`special&&func==`ADDU)||(op==`special&&func==`ADD)||(op==`special&&func==`AND)||(op==`special&&func==`OR)||(op==`special&&func==`NOR)
                ||(op==`special&&func==`XOR)||(op==`special&&func==`SUBU)||(op==`special&&func==`SLT)||(op==`special&&func==`SLTU)
                ||(op==`special&&func==`SUB)||(op==`special&&func==`SLL)||(op==`special&&func==`SRA)||(op==`special&&func==`MULT)||(op==`special&&func==`MULTU)
                ||(op==`special&&func==`SRL)||(op==`special&&func==`SLLV)||(op==`special&&func==`SRAV)||(op==`special&&func==`DIV)||(op==`special&&func==`DIVU)
                ||(op==`special&&func==`SRLV);
assign Tuse_rt2=(op==`SW)||(op==`SB)||(op==`SH);
```

在并入所有新增指令时都需要考虑该指令用到了 rs 还是 rt，他的 Tnew 在哪里产生，并且并

入相应的 Tuse 行列。

至于暂停，此处增加了一个乘除的暂停判断，共同并入暂停条件

```
wire [4:0] A1=Instr_D[25:21];
wire [4:0] A2=Instr_D[20:16];

assign Stall_rs0_E1=Tuse_rs0 & (res_e==`ALU) & (A1==A3_E) & (A1!=5'b00000);
assign Stall_rs0_E2=Tuse_rs0 & (res_e==`DM) & (A1==A3_E) & (A1!=5'b00000);
assign Stall_rs1_E2=Tuse_rs1 & (res_e==`DM) & (A1==A3_E) & (A1!=5'b00000);
assign Stall_rs0_M1=Tuse_rs0 &(res_m==`DM) & (A1==A3_M) & (A1!=5'b00000);
assign Stall_rs= Stall_rs0_E1 | Stall_rs0_E2 | Stall_rs1_E2 | Stall_rs0_M1;

assign Stall_rt0_E1=Tuse_rt0 & (res_e==`ALU) & (A2==A3_E) & (A2!=5'b00000);
assign Stall_rt0_E2=Tuse_rt0 & (res_e==`DM) & (A2==A3_E) & (A2!=5'b00000);
assign Stall_rt1_E2=Tuse_rt1 & (res_e==`DM) & (A2==A3_E) & (A2!=5'b00000);
assign Stall_rt0_M1=Tuse_rt0 &(res_m==`DM) & (A2==A3_M) & (A2!=5'b00000);
assign Stall_rt= Stall_rt0_E1 | Stall_rt0_E2 | Stall_rt1_E2 | Stall_rt0_M1;

assign Stall_multdiv=(op==`special&(func==`DIV|func==`DIVU|func==`MULT|func==`MULTU
                   |func==`MFHI|func==`MFLO|func==`MTHI|func==`MTLO))&(Start!=0|Busy!=0);

assign stall=Stall_rs|Stall_rt|Stall_multdiv;
```

转发机制和 p5 相同，没有进行任何的更改。

```
assign F_CMP1_D=((A1_D==A3_E)&(res_e==`PC)&(A1_D!=0))?`E2D_PC:
                ((A1_D==A3_M)&(res_m==`PC)&(A1_D!=0))?`M2D_PC:
                ((A1_D==A3_M)&(res_m==`ALU)&(A1_D!=0))?`M2D_ALU:
                ((A1_D==A3_W)&(res_w==`ALU)&(A1_D!=0))?`W2D_ALU:
                ((A1_D==A3_W)&(res_w==`DM)&(A1_D!=0))?`W2D_DM:
                ((A1_D==A3_W)&(res_w==`PC)&(A1_D!=0))?`W2D_PC:0;

assign F_CMP2_D=((A2_D==A3_E)&(res_e==`PC)&(A2_D!=0))?`E2D_PC:
                ((A2_D==A3_M)&(res_m==`PC)&(A2_D!=0))?`M2D_PC:
                ((A2_D==A3_M)&(res_m==`ALU)&(A2_D!=0))?`M2D_ALU:
                ((A2_D==A3_W)&(res_w==`ALU)&(A2_D!=0))?`W2D_ALU:
                ((A2_D==A3_W)&(res_w==`DM)&(A2_D!=0))?`W2D_DM:
                ((A2_D==A3_W)&(res_w==`PC)&(A2_D!=0))?`W2D_PC:0;

assign F_ALU_A_E=((A1_E==A3_M)&(res_m==`PC)&(A1_E!=0))?`M2E_PC:
                 ((A1_E==A3_M)&(res_m==`ALU)&(A1_E!=0))?`M2E_ALU:
                 ((A1_E==A3_W)&(res_w==`ALU)&(A1_E!=0))?`W2E_ALU:
                 ((A1_E==A3_W)&(res_w==`DM)&(A1_E!=0))?`W2E_DM:
                 ((A1_E==A3_W)&(res_w==`PC)&(A1_E!=0))?`W2E_PC: 0;

assign F_ALU_B_E=((A2_E==A3_M)&(res_m==`PC)&(A2_E!=0))?`M2E_PC:
                 ((A2_E==A3_M)&(res_m==`ALU)&(A2_E!=0))?`M2E_ALU:
                 ((A2_E==A3_W)&(res_w==`ALU)&(A2_E!=0))?`W2E_ALU:
                 ((A2_E==A3_W)&(res_w==`DM)&(A2_E!=0))?`W2E_DM:
                 ((A2_E==A3_W)&(res_w==`PC)&(A2_E!=0))?`W2E_PC: 0;


assign F_WD_M=((A2_M==A3_W)&(res_w==`ALU)&(A2_M!=0))?`W2M_ALU:
              ((A2_M==A3_W)&(res_w==`DM)&(A2_M!=0))?`W2M_DM:
              ((A2_M==A3_W)&(res_w==`PC)&(A2_M!=0))?`W2M_PC:0;
```

由于不同的控制信号是在不同的级别使用的，所以这里使用了分布式译码，仅在需要此控制信号的级别传入该信号。即实例化四个 control 单元。

（三） 测试程序

```
#共支持 11 条指令
#Add,Addu,Sub,Subu,And,Or,Xor,Nor;
#Mult,Multu,Div,Divu,Mfhi,Mflo,Mthi,Mtlo,Madd;
#Sll,Srl,Sra,Sllv,Srlv,Srav;
#Slt,Sltu;
#Jr,Jalr;
#Addi,Addiu,Andi,Ori,Xori,Lui;
#Slti,Sltiu;
#wire Lw,Lb,Lbu,Lh,Lhu,Sw,Sb,Sh;
#wire Beq,Bne,Blez,Bgtz;
#wire Bgezal,Bltz,Bgez;
#wire J,Jal;
#功能测试
ori $t0,$0,520
ori $t1,$t0,233

lui $t0,520
lui $t1,0xffff
ori $t1,$t1,0xffff

#addu,subu
addu $t2,$t0,$t0#++
addu $t3,$t0,$t1#+-
addu $t4,$t1,$t1#--
subu $t5,$t0,$t2#++
subu $t6,$t0,$t1#+-
subu $t7,$t1,$t4#--

#add,sub
ori $s0,$0,1
ori $s1,$0,4
ori $s2,$0,15
lui $t0,0xfda3
ori $t0,$t0,0x34f5 #-
ori $t1,$0,0x234 #+
lui $t2,0x424
ori $t2,$t2,32853
add $a0,$t0,$t1 #-+
add $a1,$t0,$t0 #--
add $a2,$t1,$t1 #++
subu $a0,$t0,$t1 #-+
subu $a1,$t1,$t0 #+-
```

```
subu $a2,$t1,$t2 #++
subu $a3,$t2,$t1 #++
lui $t3,0xf39f
subu $a1,$t0,$t3 #--
subu $a2,$t3,$t0

#and,or,xor,nor
ori $s0,$0,1
ori $s1,$0,4
ori $s2,$0,15
lui $t0,0xfda3
ori $t0,$t0,0x34f5 #-
ori $t1,$0,0x234 #+
lui $t2,0x424
ori $t2,$t2,32853
and $a0,$t0,$t1 #-+
and $a1,$t0,$t0 #--
and $a2,$t1,$t1 #++
or $a0,$t0,$t1 #-+
or $a1,$t1,$t0 #+-
or $a2,$t1,$t2 #++
xor $a3,$t2,$t1 #++
xor $a0,$t0,$t1 #-+
xor $a1,$t1,$t0 #+-
xor $a2,$t1,$t2 #++
lui $t3,0xf39f
or $a1,$t0,$t3 #--
xor $a2,$t3,$t0
nor $a0,$t0,$t1 #-+
nor $a1,$t1,$t0 #+-
nor $a2,$t1,$t2 #++

#mult
lui $t0,0xfeaf
ori $t0,$t0,0x5254
lui $t1,0x0243
ori $t1,$t1,0x323f
mult $t0,$t0 #--
mfhi $a0
mflo $a1
mult $t1,$t0 #+-
mfhi $a0
mflo $a1
mult $t1,$t1 #-+
```

```
mfhi $a1
mflo $a0

#multu
lui $t0,0xfeaf
ori $t0,$t0,0x5254
lui $t1,0x0243
ori $t1,$t1,0x323f
multu $t0,$t0
mfhi $a0
mflo $a1
multu $t1,$t0
mfhi $a0
mflo $a1
multu $t1,$t1
mfhi $a0
mflo $a1

#div,divu
lui $t0,0xfeaf
ori $t0,$t0,0x5254
lui $t1,0x0243
ori $t1,$t1,0x323f
lui $t2,0xe120
ori $t2,$t2,0x300
ori $t3,$0,0x99
div $t0,$t1 #-+
mfhi $a0
mflo $a1
div $t1,$t0
mfhi $a0
mflo $a1
div $t0,$t2 #--
mfhi $a0
mflo $a1
div $t2,$t0
mfhi $a0
mflo $a1
div $t1,$t3 #++
mfhi $a0
mflo $a1
div $t0,$t1 #-+
mfhi $a0
mflo $a1
```

```
div $t1,$t0
mfhi $a0
mflo $a1
div $t0,$t2 #--
mfhi $a0
mflo $a1
div $t2,$t0
mfhi $a0
mflo $a1
div $t1,$t3 #++
mfhi $a0
mflo $a1

#mthi,mtlo
lui $t0,0xfeaf
ori $t0,$t0,0x5254
lui $t1,0x0243
ori $t1,$t1,0x323f
mult $t0,$t0
mthi $t0
mtlo $t1
mfhi $a0
mflo $a1
mult $t1,$t0
mthi $t1
mfhi $a0
mult $t1,$t1
mtlo $t0
mfhi $a0
mflo $a1

#sll,srl,sra
ori $t0,$t0,123
sll $t0,$t0,4
srl $t0,$t0,4
ori $t1,$t1,0xffff
sll $t1,$t1,2
srl $t1,$t1,2
sra $t0,$t0,3  #+
lui $t0,0xf234
sra $t1,$t0,3 #-
lui $t0,0x23
sra $t0,$t0,4 #+
```

```
#sllv,srlv,srav
ori $s0,$0,1
ori $s1,$0,4
ori $s2,$0,15
lui $t0,0xfda3
ori $t0,$t0,0x34f5
ori $t1,$0,0x234
lui $t2,0x424
ori $t2,$t2,32853
sllv $a0,$t0,$s0 #-
sllv $a1,$t1,$s1 #+
sllv $a3,$t2,$s2 #+
srlv $a0,$t0,$s0
srlv $a1,$t1,$s1
srlv $a3,$t2,$s2
srav $a0,$t0,$s0
srav $a0,$t0,$s1
srav $a1,$t1,$s1
srav $a3,$t2,$s2

#slt,sltu
ori $t0,$0,2
ori $t1,$0,1
ori $t2,$0,2
lui $t3,0xffff
slt $a0,$t2,$t1#++
slt $a1,$t3,$t1#-+
ori $t4,$t3,0x1234
slt $a2,$t3,$t4 #--
sltu $a0,$t2,$t1 #++
sltu $a1,$t3,$t1 #-+
sltu $a2,$t3,$t4
sltu $a3,$t4,$t3

#jalr
ori $t0,$0,1
jal jalr_loop
ori $t0,$ra,0
ori $t2,$0,2
addu $a1,$a1,$a0
j jalr_end
ori $t6,$t6,6
jalr_loop:
addu $a0,$0,$ra
```

```
jalr $a0,$ra
ori $t4,$a0,0
ori $t5,$t5,5
jalr_end:

#addi,addiu,andi,xori
ori $s0,$0,1
ori $s1,$0,4
ori $s2,$0,15
lui $t0,0xfda3
ori $t0,$t0,0x34f5 #-
ori $t1,$0,0x234 #+
lui $t2,0x424
ori $t2,$t2,32853
addi $a0,$t0,0xea4
addi $a1,$t0,-134
addi $a2,$t1,0xf53f
addi $a3,$t1,533
addiu $a0,$t0,0xea4
addiu $a1,$t0,-134
addiu $a2,$t1,0xf53f
addiu $a3,$t1,533
andi $a0,$t0,0xea4
andi $a1,$t0,-134
andi $a2,$t1,0xf53f
andi $a3,$t1,533
xori $a0,$t0,0xea4
xori $a1,$t0,-134
xori $a2,$t1,0xf53f
xori $a3,$t1,533

#slti,sltiu
ori $t0,$0,2
ori $t1,$0,1
lui $t3,0xffff
slti $a0,$t1,2 #+
slti $a1,$t0,1
slti $a2,$t0,-100
slti $a3,$t3,2390
sltiu $a0,$t1,2 #+
sltiu $a1,$t0,1
sltiu $a2,$t0,-100 #-
sltiu $a3,$t3,2390
```

```
#sw,sh,sb,lh.lhu,lb,lbu
ori $20,$0,1
lui $21,0xffff
ori $21,$21,0xffff
lui $3,0xf3f4
ori $3,$3,0x71f2
sw $3,0($0)
sh $3,8($0)
sh $3,10($0)
sb $3,4($0)
sb $3,5($0)
sb $3,6($0)
sb $3,7($0)
lh $2,0($0)
lh $2,2($0)
lhu $2,0($0)
lhu $2,2($0)
lb $2,0($0)
lb $2,1($0)
lb $2,2($0)
lb $2,3($0)
lbu $2,0($0)
lbu $2,1($0)
lbu $2,2($0)
lbu $2,3($0)

#bne
ori $t0,$0,1
bne $t0,$t0,bne_label1
ori $t1,$0,1
ori $t2,$0,2
bne $t0,$t2,bne_label1
ori $t3,$0,3
ori $t4,$0,4
bne_label1:ori $t3,$t3,3

#blez,bgtz
ori $a1,$0,1
lui $a2,0xf433
blez $a1,blez_label1
ori $t0,$t0,1
ori $t1,$t1,2
ori $a0,$0,0
blez $a0,blez_label1
```

```
ori $t2,$t2,2
ori $t3,$t3,3
blez_label1:ori $t4,$t4,4
blez $a2,bgez_label2
ori $t5,$t5,5
ori $t6,$t6,6
blez_label2:ori $t7,$t7,7
bgtz $a2,bgtz_label3
ori $t0,$t0,1
ori $t1,$t1,2
bgtz $a0,bgtz_label3
ori $t2,$t2,2
ori $t3,$t3,3
bgtz $a1,bgtz_label3
ori $t5,$t5,5
ori $t6,$t6,6
bgtz_label3:ori $t4,$t4,4

#bltz,bgez
ori $a1,$0,1
lui $a2,0xf433
bgez $a2,bgez_label1
ori $t0,$t0,1
ori $t1,$t1,2
ori $a0,$0,0
bgez $a0,bgez_label1
ori $t2,$t2,2
ori $t3,$t3,3
bgez_label1:ori $t4,$t4,4
bgez $a1,bgez_label2
ori $t5,$t5,5
ori $t6,$t6,6
bgez_label2:ori $t7,$t7,7
bltz $a1,bgez_label3
ori $t0,$t0,1
ori $t1,$t1,2
bltz $a0,bgez_label3
ori $t2,$t2,2
ori $t3,$t3,3
bltz $a2,bgez_label3
ori $t5,$t5,5
ori $t6,$t6,6
bgez_label3:ori $t4,$t4,4
```

```
ori $t0,$0,8
sw $t1,-8($t0)
sw $t2,0($t0)
sw $t3,8($t0)
lw $t4,-8($t0)
lw $t5,0($t0)
lw $t6,8($t0)

 #测试跳转及延迟槽
jal loop1
ori $t0,$0,233
nop
j test
ori $t0,$0,555
ori $t0,$0,342
test:
ori $t0,0
ori $t1,$0,1
beq $t0,$t1,test1
ori $t2,$0,1
nop
beq $t1,$t2,test1
ori $t3,3
ori $t4,4
test1:
ori $t0,$0,0

#冲突测试
#需要初始化内存单元
ori $t0,$0,16
sw $t0,0($0)
ori $t0,$0,432
sw $t0,4($0)
ori $t0,$0,858
sw $t0,8($0)
ori $t0,$0,656
sw $t0,12($0)
ori $t0,$0,8419
sw $t0,16($0)
ori $t0,$0,3420
sw $t0,20($0)
ori $t0,$0,29812
sw $t0,24($0)
```

```
ori $t0,$0,1812
sw $t0,28($0)
ori $t0,$0,213
sw $t0,32($0)

#addu
#addu RSuse,RTuse 在 E
ori $t1,$0,516
ori $t2,$0,233
#R-M-RS
subu $t0,$t1,$t2
addu $t3,$t0,$t1
#R-M-RT
subu $t0,$t2,$t1
addu $t3,$t1,$t0
#R-W-RS
subu $t0,$t1,$t2
nop
addu $t3,$t0,$t1
#R-W-RT
subu $t0,$t2,$t1
nop
addu $t3,$t1,$t0
#I-M-RS
ori $t0,$t1,101
addu $t3,$t0,$t1
#I-M-RT
ori $t0,$t1,102
addu $t3,$t1,$t0
#I-W-RS
ori $t0,$t1,103
nop
addu $t3,$t0,$t1
#I-W-RT
ori $t0,$t1,104
nop
addu $t3,$t1,$t0
#LD-M-RS
lw $t0,0($0)
addu $t3,$t0,$t1
#LD-M-RT
lw $t0,4($0)
addu $t3,$t1,$t0
#LD-W-RS
```

```
lw $t0,8($t0)
nop
addu $t3,$t0,$t1
#LD-W-RT
lw $t0,12($t0)
nop
addu $t3,$t1,$t0
#JAL-M-RS
jal loop1
addu $t2,$ra,$t0
#JAL-M-RT
jal loop2
addu $t2,$t0,$ra
#JAL-W-RS
jal loop1
nop
addu $t2,$ra,$t0
#JAL-W-RT
jal loop2
nop
addu $t2,$t0,$ra
#MU-M-RS
mult $t0,$t0
mfhi $a0
addu $t2,$a0,$t0
#MU-M-RT
mflo $a0
addu $t2,$t1,$a0
#MU-W-RS
mult $t1,$t1
mfhi $a1
nop
addu $t2,$a1,$t0
#MU-W-RT
mflo $a2
nop
addu $t0,$t0,$a2

#ori
#ori RSuse 在 E
#R-M-RS
addu $t0,$t1,$t2
ori $t1,$t0,233
#R-W-RS
```

```
addu $t0,$t1,$t2
nop
ori $t1,$t0,242
#I-M-RS
lui $t0,454
ori $t1,$t0,234
#I-W-RS
lui $t0,49
nop
ori $t1,$t0,34
#LD-M-RS
lw $t0,16($0)
ori $t0,342
#LD-W-RS
lw $t0,20($0)
nop
ori $t0,984
#JAL-M-RS
jal loop1
ori $t0,$ra,5995
#JAL-W-RS
jal loop2
nop
ori $t0,$ra,488
#MU-M-RS
mult $t0,$t0
mfhi $a0
ori $t0,$a0,3415
#MU-W-RS
mult $t1,$t1
mflo $a1
nop
ori $t0,$a1,328

#lw
#lwRSuse 在 E
ori $t0,$0,0
ori $t1,$0,4
#R-M-RS
addu $t0,$t0,$t1
lw $t3,0($t0)
#R-W-RS
addu $t0,$t0,$t1
nop
```

```
lw $t3,0($t0)
#I-M-RS
ori $t0,$0,16
lw $t3,0($t0)
#I-W-RS
ori $t0,$0,20
nop
lw $t3,0($t0)
#LD-M-RS
lw $t0,40($0)
lw $t3,0($t0)
#LD-W-RS
lw $t0,0($0) #t0=4
nop
lw $t3,0($t0)
#SD-M-RS
sw $t3,36($0)
lw $t4,36($0)
#SD-W-RS
sw $t3,40($0)
nop
lw $t4,40($0)

#beq
#beq,RSuse/RTuse 在 D
#R-E-RS
ori $t0,$0,0
ori $t1,$0,0
ori $t2,$0,1
ori $t3,$0,2
addu $t0,$t0,$t2
beq $t0,$t1,wrong
nop
#R-E-RT
ori $t0,$0,0
addu $t1,$t1,$t3
beq $t0,$t1,wrong
nop
#R-M-RS
ori $t0,$0,0
ori $t1,$0,0
addu $t0,$t0,$t2
nop
beq $t0,$t1,wrong
```

```
nop
#R-M-RT
ori $t0,$0,0
addu $t1,$t1,$t3
nop
beq $t0,$t1,wrong
nop
#R-W-RS
ori $t0,$0,0
ori $t1,$0,0
addu $t0,$t0,$t2
nop
nop
beq $t0,$t1,wrong
nop
#R-W-RT
ori $t0,$0,0
addu $t1,$t1,$t3
nop
nop
beq $t0,$t1,wrong
nop
#LD-E-RS
ori $t0,$0,1
ori $t1,$0,1
lw $t0,100($0)
beq $t0,$t1,wrong
nop
#LD-M-RS
ori $t0,$0,1
ori $t1,$0,1
lw $t0,100($0)
nop
beq $t0,$t1,wrong
nop
#LD-W-RS
ori $t0,$0,1
ori $t1,$0,1
lw $t0,100($0)
nop
nop
beq $t0,$t1,wrong
nop
#JAL-E-RS
```

```
ori $t0,$0,0
ori $ra,$0,0
jal loop1
beq $t0,$ra,wrong
nop
#JAL-M-RS
ori $ra,$0,0
jal loop4
nop
loop4:beq $t0,$ra,wrong
nop
#JAL-W-RS
ori $ra,$0,0
jal loop5
nop
loop5:nop
beq $t0,$ra,wrong
nop
#MU-E-RS
lui $t0,0xfe23
ori $t0,$t0,0x24f5
lui $t1,0x948
ori $t1,$t1,0x8840
ori $a0,$0,0
mult $t1,$t1
mfhi $a0
beq $a0,$0,wrong
nop
#MU-E-RT
ori $a1,$0,0
mflo $a1
beq $0,$a1,wrong
nop
#MU-M-RS
mult $t0,$t0
ori $a0,$0,0
mfhi $a0
nop
beq $a0,$0,wrong
nop
#MU-M-RT
ori $a1,$0,0
mflo $a1
nop
```

```
beq $0,$a1,wrong
nop
#MU-W-RS
mult $t0,$t0
ori $a0,$0,0
mfhi $a0
nop
nop
beq $a0,$0,wrong
nop
#MU-W-RT
ori $a1,$0,0
mflo $a1
nop
nop
beq $0,$a1,wrong
nop


#sw
#sw,RSuse 在 E,RTuse 在 M
#R-M-RS
addu $t0,$0,4
sw $t4,0($t0)
#R-M-RT
addu $t3,$0,423
sw $t3,0($t0)
#R-W-RS
addu $t0,$t0,4
nop
sw $t3,0($t0)
#R-W-RT
addu $t3,$t0,24214
nop
sw $t3,0($t0)
#I-M-RS
ori $t0,$0,16
sw $t3,0($t0)
#I-M-RT
ori $t3,$0,235
sw $t4,0($t0)
#I-W-RS
ori $t0,$0,20
nop
```

```
sw $t3,0($t0)
#I-W-RT
ori $t3,$0,9885
nop
sw $t3,0($t0)
#LD-M-RS
lw $t0,64($0)
sw $t3,0($t0)
#LD-M-RT
lw $t0,24($0)
sw $t0,32($0)
#LD-W-RS
lw $t0,80($0)
nop
sw $t3,4($t0)
#LD-W-RT
lw $t3,32($0)
nop
sw $t3,0($0)
#JAL-M-RT
jal loop1
sw $ra,0($0)
#JAL-W-RT
jal loop3
nop
loop3:
sw $ra,0($0)
#MU-W-RT
lui $t1,0x948
ori $t1,$t1,0x8840
ori $a0,$0,0
mult $t1,$t1
mfhi $a0
sw $a0,0($0)

#mult
#mult,RSuse,RTuse 在 E
lui $t1,0xf284
ori $t1,$t1,516
lui $t2,0x344
ori $t2,$t2,233
#R-M-RS
addu $a0,$t1,$t2
mult $a0,$t1
```

```
mfhi $a0
#R-M-RT
addu $a1,$t1,$t1
mult $t2,$a1
mflo $a2
#R-W-RS
addu $a0,$t2,$t2
mult $a0,$t1
nop
mfhi $a1
#R-W-RT
addu $a0,$t1,$t2
mult $a0,$t1
nop
mfhi $a0
#I-M-RS
ori $t0,$0,0x324
mult $a0,$t0
mfhi $a0
#I-M-RT
ori $t0,$0,0x4242
mult $t0,$a1
mflo $a1
#I-W-RS
ori $t0,$0,0x3343
mult $a0,$t0
nop
mfhi $a0
#I-W-RT
ori $t0,$0,0xf242
mult $t0,$a1
nop
mflo $a1
#LD-M-RS
lw $t0,0($0)
mult $t0,$a1
mfhi $a0
#LD-M-RT
lw $t0,4($0)
mult $a0,$t0
mflo $a2
#LD-W-RS
lw $t0,8($0)
mult $t0,$a1
```

```
mfhi $a1
#LD-W-RT
lw $t0,12($0)
mult $a1,$t0
mflo $a1
#JAL-M-RS
ori $a0,0x4248
jal loop1
mult $ra,$a0
mflo $a1
#JAL-M-RT
jal loop2
mult $a1,$ra
mfhi $a2
#JAL-W-RS
jal loop1
nop
mult $ra,$a2
mflo $a3
#JAL-W-RT
jal loop2
nop
mult $a3,$ra
mfhi $t0

j end
nop

wrong:
ori $s0,$0,2333

loop1:
jr $ra
nop
nop

loop2:
jr $ra
nop
nop
end:
```

指令方面因为无法全面的覆盖所有指令的转发，这里采用的是测试每个指令的功能性以及各种类别指令的转发和暂停。以上为按照这个原则的测试代码。

二、 思考题

1. 为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

我觉得单独的乘除法部件需要单独出来的原因是与指令行为相关的，在乘除指令中我们需要将其计算结果分别存入 HI 寄存器和 LO 寄存器，而指令中并没有指明其寄存器编号的数据段，所以需要独立的 HI 寄存器和 LO 寄存器，并且乘除法在实际实现中是比其他的运算需要更多的时间，倘若整合到 ALU，势必会大大增加流水线的周期，对于 cpu 的执行效率来说是不利的，所以需要有单独的乘除法部件。独立的 LO 和 HI 我认为一是为了配合乘除相关指令，二是由于在汇编指令中乘除相关指令一半会配合取 HI 寄存器和 LO 寄存器的操作，为了转发方便节省周期数，需要有独立的 HI 和 LO 寄存器。

2. 参照你对延迟槽的理解，试解释“乘除槽”

根据我对延迟槽的理解，跳转指令需要经过一个周期的判断和地址输送才能改变下一个地址来读取下一条指令，所以势必需要一个周期进行暂停，所以在一个周期去处理一条不相关指令时可行的，可以提高 cpu 的执行效率。所以我们称这个跳转的空档为延迟槽，原因来源于跳转指令的延迟。而对于乘除指令，其在进入 E 级部件后也会产生 5-10 个周期的延迟，其相关指令势必要进行暂停，但是在这个空档中是可以处理一些不相关指令的，也可以提高 cpu 的执行效率。所以相比较延迟槽，这个进行乘除法操作的延迟，可以被称为乘除槽。

3. 为何上文文末提到的 lb 等指令使用的数据扩展模块应在 MEM/WB 之后，而不能在 DM 之后?

在计组理论课上，我们得到的一般性结论是 MEM 部件的延迟最高，而流水线的执行周期是根据整条流水线中延迟最高的部件决定的，倘若把数据扩展模块放在 DM 之后的话，其势必会增加 MEM 部件的延迟，同时增加了整条流水线的执行周期，对于 cpu 的运行效率有很大影响。但是对于有转发的情况，也就是最差情况来讲，数据的经过通路是相同的，所以对于时钟周期好像又没有影响，在考虑到之前提到过转发数据应当从流水级寄存器转发，并且将数据扩展模块放在 DM 后需要生成新的控制信号会使数据通路更加复杂，所以此数据扩展模块应该在 MEM/WB 之后。

4. 举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑 C 语言中字符串的情况）

当我们需要进行 sb 操作，即对 DM 写入字节时，倘若 DM 为按字访问，我们需要提出该字地址的整个字，然后修改其中相应字节的内容然后再存回去。而我们倘若把 DM 分成四块，即每个字分成四个字节，在存储字节操作上我们可以直接将相应字节存入相应地址，而不需要把数据读出再操作。虽然在我们实现中看不出来时间的差异，但是在实际中按字节访问在效率和时间利用上是比按字访问内存更有优势的。

5. 如何概括你所设计的 CPU 的设计风格？为了对抗复杂性你采取了哪些抽象和规范手段？

我认为自己的 CPU 设计风格更加倾向于规划者型，也就是所谓的全知全能安排一切，通过分析每条指令的 rs 和 rt 的 Tuse 和其产生的 Tnew 来分析冲突，然后构建相应的 Stall 信号。在添加新信号的时依循以下的顺序进行操作来进行冲突的规范。1.将新增指令的添加到相应的 Tuse 信号中。2.构建新增指令的 Tnew。仅仅用这两步就可以完成一条新增指令的冲突分析。因为在最初的冲突构建中已经考虑到了所有的冲突情况，把所有条件抽象出来而非一个一个写清楚。所以在新增指令到来时，只需要将指令添加到相对应的抽象信号中即可。在代码量上并没有什么增加。

6. 你对流水线 CPU 设计风格有何见解？

结合亲身的 cpu 设计历程，在设计中善于使用宏定义是很有优势的，用简洁明了的名称来代替二进制编码虽然在代码开头需要更多处理，但是在后面的编写过程，尤其是条件判断和信号生成上会更加的直观明显。我把这理解为一个包装的过程，通过适当的包装把 cpu 需要使用的组件尽可能的简化，这样在顶层实现会很简单明了。并且在刚开始设计的时候，最好能够考虑到它的可扩展性，体现在控制信号的位数上，或者是否可能整合某几个信号，比如 cmp 单元我曾经设计两个信号分别表示大于 0 和小于 0，其实完全可以用一个两位信号，00 代表等于 0，,01 大于 0,10 小于 0 来表示，简化端口数量对于后面的处理和各个单元部件的连接有很大好处。

7. 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

序号	测试类型	前序指令	前序指令位置	冲突位置	冲突寄存器	测试样例	解决
1	R-E-RS	subu	E	CMP	Rs	Subu $1,$2,$3 Beq $1,$4,label	暂停 M2D 转发 AO@M
2	R-E-RT	subu	E	CMP	Rt	Subu $1,$2,$3 Beq $4,$1,label	暂停 M2D 转发 AO@M
3	R-M-RS	Subu	M	ALU	Rs	Subu $1,$2,$3 Addu $4,$1,$2	M2E 转发 AO@M
4	R-M-RT	Subu	M	ALU	Rt	Subu $1,$2,$3 Addu $4,$2,$1	M2E 转发 AO@M
5	R-M-RS	Subu	M	CMP	Rs	Subu $1,$2,$3 Instr 无关 Beq $1,$4,label	M2D 转发 AO@M
6	R-M-RT	Subu	M	CMP	Rt	Subu $1,$2,$3 Instr 无关 Beq $4,$1,label	M2D 转发 AO@M

7	R-W-RS	Subu	W	ALU	Rs	Subu $1,$2,$3
						Instr 无关
						Addu $4,$1,$2
							W2E 转发 AO@W

8	R-W-RT	Subu	W	ALU	Rt	Subu $1,$2,$3
						Instr 无关
						Addu $4,$2,$1
							W2E 转发 AO@W

9	R-W-RS	Subu	W	CMP	Rs	Subu $1,$2,$3 Instr 无关 Instr 无关 Beq $1,$4,label	W2D 转发 AO@W
0	R-W-RT	Subu	W	CMP	Rt	Subu $1,$2,$3 Instr 无关 Instr 无关	W2D 转发 AO@W

						Beq $4,$1,label	

11	R-W-RS	Subu	W	ALU	Rs	Subu $1,$2,$3
						Instr 无关
						Sw $4,0($1)
							W2E 转发 AO@W

12	R-W-RT	subu	W	DM	Rt	Subu $1,$2,$3 Sw $1,0($4)	W2M 转发 AO@W
13	I-E-RS	Ori	E	CMP	Rs	Ori $1,$2,100 Beq $1,$3,label	暂停 M2D 转发 AO@M
14	I-E-RT	Ori	E	CMP	Rt	Ori $1,$2,100 Beq $3,$1,label	暂停 M2D 转发 AO@M
15	I-M-RS	Ori	M	ALU	Rs	Ori $1,$2,$100 Addu $4,$1,$2	M2E 转发 AO@M
16	I-M-RT	Ori	M	ALU	Rt	Ori $1,$2,$100 Addu $4,$2,$1	M2E 转发 AO@M
17	I-M-RS	Ori	M	CMP	Rs	Ori $1,$2,100 Instr 无关 Beq $1,$3,label	M2D 转发 AO@M
18	I-M-RT	Ori	M	CMP	Rt	Ori $1,$2,100 Instr 无关 Beq $3,$1,label	M2D 转发 AO@M
19	I-W-RS	Ori	W	ALU	Rs	Ori $1,$2,$100 Instr 无关 Addu $4,$1,$2	W2E 转发 AO@W
20	I-W-RT	Ori	W	ALU	Rt	Ori $1,$2,$100 Instr 无关 Addu $4,$2,$1	W2E 转发 AO@W
21	I-W-RS	Ori	W	CMP	Rs	Ori $1,$2,100 Instr 无关 Instr 无关	W2D 转发 AO@W

						Beq $1,$3,label	
22	I-W-RT	Ori	W	CMP	Rt	Ori $1,$2,100 Instr 无关 Instr 无关 Beq $3,$1,label	W2D 转发 AO@W
23	I-W-RS	Ori	W	ALU	Rs	Ori $1,$2,100 Instr 无关 Sw $4,0($1)	W2D 转发 AO@W
24	I-W-RT	Ori	W	DM	Rt	Ori $1,$2,100 Sw $1,0($4)	W2D 转发 AO@W
25	LD-E-RS	Lw	E	CMP	Rs	Lw $1,0($2) Beq $1,$3,label	暂停两个周期 W2D 转发 DR@W
26	LD-E-RT	Lw	E	CMP	Rt	Lw $1,0($2) Beq #3,$1,label	暂停两个周期 W2D 转发 DR@W
27	LD-M-RS	Lw	M	ALU	Rs	Lw $1,0($2) Addu $3,$1,$2	暂停一个周期 W2E 转发 DR@W
28	LD-M-RT	Lw	M	ALU	Rt	Lw $!,0($2) Addu $3,$2,$1	暂停一个周期 W2E 转发 DR@W
29	LD-M-RS	Lw	M	CMP	Rs	Lw $1,0($2) Instr 无关 Beq $1,$3,label	暂停一个周期 W2D 转发 DR@W
30	LD-M-RT	Lw	M	CMP	Rt	Lw $1,0($2) Instr 无关 Beq $3,$1,label	暂停一个周期 W2D 转发 DR@W

31	LD-W-RS	Lw	W	ALU	Rs	Lw $1,0($2)
						Instr 无关
						Addu $3,$1,$2
							W2E 转发 DR@W

32	LD-W-RT	Lw	W	ALU	Rt	Lw $1,0($2) Instr 无关	W2E 转发 DR@W

						Addu $3,$2,$1	

33	LD-W-RS	Lw	W	ALU	Rs	Lw $1,0($2)
						Instr 无关
						Sw $3,0($1)
							W2E 转发 DR@W

34	LD-W-RT	Lw	W	DM	Rt	Lw $1,0($2)
						Instr 无关
						Sw $1,0($3)
							W2E 转发 DR@W

35	LD-W-RS	Lw	W	CMP	Rs	Lw $1,0($2) Instr 无关 Instr 无关 Beq $1,$3,label	W2D 转发 DR@W
36	LD-W-RT	Lw	W	CMP	Rt	Lw $1,0($2) Instr 无关 Instr 无关 Beq $3,$1,label	W2D 转发 DR@W
37	JAL-E-RS	Jal	E	CMP	Rs	Jal label Beq $ra,$2,label	D2D 转发 PC8
38	JAL-E-RT	Jal	E	CMP	Rt	Jal label Beq $2,$ra,label	D2D 转发 PC8
39	JAL-M-RS	Jal	M	ALU	Rs	Jal label Addu $2,$ra,$1	M2E 转发 PC8
40	JAL-M-RT	Jal	M	ALU	Rt	Jal label Addu $2,$1,$ra	M2E 转发 PC8
41	JAL-M-RS	Jal	M	CMP	Rs	Jal label Instr 无关 Beq $2,$ra,label	M2D 转发 PC8
42	JAL-M-RT	Jal	M	CMP	Rt	Jal label Instr 无关 Beq $ra,$2,label	M2D 转发 PC8

43	JAL-W-RS	Jal	W	ALU	Rs	Jal label Instr 无关 Addu $2,$ra,$1	W2E 转发 PC8
44	JAL-W-RT	Jal	W	ALU	Rt	Jal label Instr 无关 Addu $2,$1,$ra	W2E 转发 PC8
45	JAL-W-RS	Jal	W	CMP	Rs	Jal label Instr 无关 Instr 无关 Beq $ra,$2,label	W2D 转发 PC8
46	JAL-W-RT	Jal	W	CMP	Rt	Jal label Instr 无关 Instr 无关 Beq $2,$ra,label	W2D 转发 PC8
47	JAL-W-RS	Jal	W	ALU	Rs	Jal label Instr 无关 Sw $2,0($ra)	W2E 转发 PC8
48	JAL-W-RT	Jal	W	DM	Rt	Jal Sw $ra,0($2)	W2E 转发 PC8

P6 新增的相关指令除去已有的 R 型，B 型，J 型，LOAD，STORE 以外，需要重新考虑的就是乘除相关的八条指令，其中 mult，multu，div，divu 的转发和 R 型指令相同都是在 E 级需求，并且不产生后续转发。Mfhi 和 mflo 指令不会产生前面的转发，因为读寄存器是 hi 和 lo，而其后续的转发是从 ALU 出现的，和其他 R 型指令的转发相同。至于 mthi 和 mtlo 指令，其只会在前序产生转发，因为其写入 hi 和 lo 寄存器就结束了。整体来说乘除相关只会在 E 级前需求寄存器值，在 E 级后生成新的寄存器值，无需构造新的转发冲突测试。