Verilog 流水线 CPU 实验报告

一、CPU 设计方案综述

（一）总体设计概述

使用 Logisim 开发一个的 MIPS 流水线 PU，总体概述如下：

1. 此 CPU 为 32 位 CPU
2. 此 CPU 支持的指令集为：{ lb、lbu、lh、lhu、lw、sb、sh、sw、add、addu、 sub、 subu、 mult、 multu、 div、 divu、 sll、 srl、 sra、 sllv、 srlv、srav、and、or、xor、nor、addi、addiu、andi、ori、 xori、lui、slt、slti、sltiu、sltu、beq、bne、blez、bgtz、 bltz、bgez、j、jal、jalr、jr、mfhi、mflo、mthi、mtlo}
3. nop 机器码为 0x00000000
4. 运算指令均不支持溢出
5. 对于 b 类和 j 类指令， 流水线设计必须支持延迟槽
6. PC 的初始地址 0x00003000

（二）关键模块定义

1.PC

（一）端口设置

表 1 PC 端口设置

序号	信号名	方向	描述
1	clk	I	时钟信号
2	PC[31:0]	I	当前 PC 值
3	reset	I	异步复位信号，将 PC 值置为 0x00003000 0：无效 1：复位
4	PC_en	I	PC 写入信号，判断是否冻结 PC

			0：冻结 1：写入
5	NPC[31:0]	O	更新后 PC 的值，即下一条指令所在的地址

(二)功能定义

表 2 PC 功能定义

序号	功能	描述
1	存储指令的地址	保存当前执行指令在 IM 中的地址

2.IM

（一）端口设置

表 3 IM 端口设置

序号	信号名	方向	描述
1	PC[31:0]	I	当前 PC 值
2	Instr[31:0]	O	将 IM 中存储的指令输出

（二）功能定义

表 4 IM 功能定义

序号	功能	描述
1	输出指令	根据 PC 的值，取出 IM 中的指令

3. IFID

（一）端口定义

表 5-IDIF 功能定义

序号	信号名	方向	描述
1	clk	I	时钟信号
2	IFID_en	I	使能信号（反向暂停信号）
3	reset	I	同步复位信号
4	PCF[31:0]	I	PC 在 F 级的值
5	InstrF[31:0]	I	instr 在 F 级的值
6	PCD[31:0]	O	PC 在 D 级的值
7	InstrD[31:0]	O	instr 在 D 级的值

（二）功能定义

表 6-IFID 功能定义

序号	功能名称	功能描述
1	传递	当时钟信号处于上升沿且暂停信号无效时，指令和 PC 输出赋值为输入
2	清零	当重置信号有效时，将输出清零

4.NPC

（一）端口说明

表 7 NPC 端口说明

序号	信号名	方向	描述
1	jr	I	是否是 jr 指令
2	j_jal	I	是否是 j/jal 指令
3	beq	I	指令是不是 beq 0：不是 1：是
4	PCF[31:0]	I	F 级 PC 值
5	JrJump[31:0]	I	$ra 中储存的 PC 值，用于 jr 指令
6	Imm32[31:0]	I	beq 指令中 0-15 位的 32 位符号扩展
7	instr[31:0]	I	IM 中要执行的指令
8	PCD[31:0]	I	D 级 PC 值
9	NPC[31:0]	O	更新后的 PC 值，指向下一条应该执行的指令

（二）功能定义

表 8 NPC 功能定义

序号	功能	描述
1	更新 PC 的值	当 Zero 和 beq 皆为 1 时， PC = PC + 4 + imm32 * 4 当 jr 为 1 时

		PC = PC_jr 当 j_jal 为 1 时 PC = {PC[31:28], Instr[25:0], 2'b00} 否则，PC = PC +4

5. GRF

（一）端口说明

表 9 GRF 端口说明

序号	信号名	方向	描述
1	A1[4:0]	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD1
2	A2[4:0]	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD2
3	A3[4:0]	I	5 位地址输入信号，指定 32 个寄存器中的一个，作为 RD 的写入地址
4	WD[31:0]	I	32 位写入到 A3 的数据
5	clk	I	时钟信号
6	clr	I	异步复位信号 0：无效 1：清零
7	GRF_WE	I	写使能信号 1：可向 GRF 中写入数据 0：不能向 GRF 中写入数据
8	RD1	O	输出 A1 指定的寄存器的 32 位数据
9	RD2	O	输出 A2 指定的寄存器的 32 位数据
10	PC[31:0]	I	题目要求，输出相应 PC 值

（二）功能定义

表 10 GRF 功能定义

序号	功能	描述
1	异步复位	reset 为 1 时，将所有寄存器清零
2	读数据	将 A1 和 A2 地址对应的寄存器的值分别通过 RD1 和 RD2 读出，读出$31 的值，方便执行 jr 指令
3	写数据	当 WE 为 1 且时钟上升沿来临时，将 WD 写入到 A3 对应的寄存器内部

6.CMP

(一)端口说明

表 11 CMP 端口说明

序号	信号名	方向	描述
1	A[31:0]	I	操作数 A
2	B[31:0]	I	操作数 B
3	Zero	O	A==B？

（二）功能描述

表 12 CMP 功能定义

序号	功能	描述
1	判断 A 和 B 是否相等	若 A 等于 B，Zero=1，否则置零

7.EXT

（一）端口说明

表 13 EXT 端口说明

序号	信号名	方向	描述
1	imm16[15:0]	I	代扩展的 16 位信号
2	EXTOp	I	无符号或符号扩展选择信号 0：无符号扩展 1：符号扩展
3	imm32[31:0]	O	扩展后的 32 位的信号

（二）功能定义

表 14 EXT 功能定义

序号	功能	描述
1	无符号扩展	当 EXTOp 为 0 时，将 imm16 无符号扩展输出
2	符号扩展	当 EXTOp 为 1 时，将 imm16 符号扩展输出

8.IDEX

(一)端口说明

表 15 IDEX 端口说明

序号	信号名	方向	描述
1	reset	I	同步复位信号
2	flush	I	清除信号，用于暂缓
3	clk	I	时钟信号
4	RD1D[4:0]	I	RD1 在 D 级的值
5	RD2D[4:0]	I	RD2 在 D 级的值
6	InstrD[31:0]	I	Instr 在 D 级的值
7	MemtoRegD[1:0]	I	MemtoReg 在 D 级的值
8	MemWriteD	I	MemWritre 在 D 级的值
9	A3D[4:0]	I	A3 在 D 级的值
10	WDD[31:0]	I	WD 在 D 级的值
11	ALUOpD[2:0]	I	ALUOp 在 D 级的值
12	ALUSrcD	I	ALUSrc 在 D 级的值
13	RD1E[4:0]	O	RD1 在 E 级的值
14	RD2E[4:0]	O	RD2 在 E 级的值
15	InstrE[31:0]	O	Instr 在 E 级的值
16	MemtoRegE[1:0]	O	MemtoReg 在 E 级的值
17	MemWriteE	O	MemWritre 在 E 级的值
18	A3E[4:0]	O	A3 在 E 级的值
19	WDE[31:0]	O	WD 在 E 级的值
20	ALUOpE[2:0]	O	ALUOp 在 E 级的值

21	ALUSrcE	O	ALUSrc 在 E 级的值

（二）功能定义

表 16-IFID 功能定义

序号	功能名称	功能描述
1	传递	当时钟信号处于上升沿且暂停信号无效时，指令和 PC 输出赋值为输入
2	清零	当重置信号有效时，将输出清零
3	冲刷	当暂缓信号有效时，将输出清零

9. ALU

（一）端口说明

表 17 ALU 端口说明

序号	信号名	方向	功能描述
1	SrA[3:0]	I	与运算的第一个数
2	SrB[31:0]	I	与运算的第二个数
3	ALUOp[2:0]	I	决定 ALU 做何种操作 000：无符号加 001：无符号减 010：或 011:LUI（加载至最高位）
4	Result[31:0]	O	运算的结果
5	shamt[4:0]	I	传递的位移数

（二）功能定义

表 18 ALU 功能定义

序号	功能	描述
1	加运算	Ans = A + B
2	减运算	Ans = A - B
3	或运算	Ans = A | B

4	LUI’运算	Ans’ = imm16||0^{16}

10.EXMEM

（一）端口说明

表 19 EXMEM 端口说明

序号	信号名	方向	描述
1	reset	I	同步复位信号
2	PCE[31:0]	I	PC 在 E 级的值
3	clk	I	时钟信号
4	InstrE31:0]	I	Instr 在 E 的值
5	MemtoRegE[1:0]	I	MemtoReg 在 E 级的值
6	MemWriteE	I	MemWritre 在 E 级的值
7	A3E[4:0]	I	A3 在 E 级的值
8	WDE[31:0]	I	WD 在 E 级的值
9	ResE[31:0]	I	Res 在 E 的值
10	imm32E[31:0]	I	imm32 在 E 值
11	InstrM31:0]	O	Instr 在 M 级的值
12	MemtoRegEM[1:0]	O	MemtoReg 在 E 级的值
13	MemWriteM	O	MemWritre 在 E 级的值
14	A3M[4:0]	O	A3 在 M 级的值
15	WDM[31:0]	O	WD 在 M 级的值
16	ResM[31:0]	O	ResM 在 M 级的值
17	PCM31:0]	O	PC 在 M 的值
18	imm32M:0]	O	imm32 在 M 的值

（二）功能定义

表 20FID 功能定义

序号	功能名称	功能描述
1	传递	当时钟信号处于上升沿且暂停信号无效时，指令和 PC 输

		出赋值为输入
2	清零	当重置信号有效时，将输出清零

10.DM

（一）端口说明

表 19 DM 端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	异步复位信号 0：无效 1：内存值全部清零
3	MemWrite	I	写使能信号 0：禁止写入 1：允许写入
4	Aaddress[31:0]	I	读取或写入信号地址
5	WD[31:0]	I	32 位写入数据
6	Sel[1:0]	I	读取/写入模式 00：字 01：字节 10：半字
7	RD[31:0]	O	32 位读出数据

（二）功能定义

表 20 DM 功能定义

序号	功能	描述
1	异步复位	当 reset 为 1 时，DM 中所有数据清零
2	写入数据	当 MemWrite 有效时，时钟上升沿来临时，WD 中数据写入 Address 对应的 DM 地址中
3	读出数据	RD 永远读出 Address 对应的 DM 地址中的值

11.MEWB

(一)功能定义

用于存储连接 M 级和 W 级 ~~FW~~

12.MulDiv

(一)功能定义

用于进行乘除运算

(二)端口说明

表 21 MulDiv 端口说明

序号	信号名	方向	描述
1	IR [31:0]	I	当前阶段传入的操作码
2	clk	I	时钟信号
3	reset	I	复位信号
4	D1[31:0]	I	第一个操作数
5	D2[31:0]	I	第二个操作数
6	busy	O	表示计算单元正在计算的信号
7	start	O	启动计算的信号
8	HIOut[31:0]	O	HI 寄存器输出
9	LOOut[31:0]	O	LO 寄存器输出

（三）功能定义

表 22MulDiv 功能定义

序号	功能名称	功能描述
1	进行乘除法运算	当根据当前操作码，计算出的控制信号是 start 时，开始进行计算，并将下一跳 M 类指令冻结在 D 级，清除 E 级信息，冻结 PC。并输出 busy 为 1 表示正在计算。
2	写 HI、LO 寄存	当 MDWrite 信号有效时，将 rs 的值写入 HI 或 LO

	器	
3	输出 HI、LO 寄存器的值	将 HI、LO 寄存器的值输出

13.Controller

（一）端口说明

表 25 Controller 端口说明

序号	信号名	方向	描述
1	Insre[5:0]	I	IM 中的指令
2	jr	O	instr 是否为 jr 信号 0：不是 1：是
3	ALUOp[2:0]	O	ALU 的控制信号 000：无符号加 001：无符号减 010：或 011:LUI（加载至最高位）
4	RegWrite	O	GRF 写使能信号 0：禁止写入 1：允许写入
5	MemWrite	O	DM 的写入信号 0：禁止写入 1：允许写入
6	beq	O	instr 是否为 beq 信号 0：不是 1：是
7	AluSrc	O	参与 ALU 运算的第二个数，来自 GRF 还是 imm 0：来自 GRF 1：imm

8	WhichtoReg[1:0]	O	将何种数据写入 GRF？ 00：ALU 计算结果 01：DM 读出信号 10：PC+4
9	RegDst[1:0]	O	GRF 写入地址选择信号 00：Rd 01：Rt 10：$r31
10	J_jal	O	instr 是否为 j/jal 信号 0：不是 1：是
11	SignExt	O	对 imm16 进行扩展的方式 0：0 扩展 1：符号扩展

（二）功能定义

表 26　Controller 功能定义

序号	功能	描述
1	判断指令类型	根据 op 和 func，判断 instr[31:0]具体是哪条指令
2	转换控制信号	利用数据通路，分析对应指令下哪些信号的选择

（三）真值表

表 17 Controller 对应的真值表

端口	addu	subu	ori	lw	sw	lui	beq	j	jal	jr
op	000000	000000	001101	100011	101011	001111	000100	000010	000011	000000
func	100001	100011								001000
AluOp	000	001	010	000	000	011	000	000	000	000
RegWrite	1	1	1	1	0	1	0	0	1	0
MemWrite	0	0	0	0	1	0	0	0	0	0
beq	0	0	0	0	0	0	1	0	0	0
ALUSrc	0	0	1	1	1	0	0	0	0	0

WhichtoReg	00	00	00	01	00	00	00	00	10	00
RegDst	00	00	01	01	01	01	01	00	10	00
SignExt	0	0	0	1	1	0	1	0	0	0
j_jal	0	0	0	0	0	0	0	1	1	0
jr	0	0	0	0	0	0	0	0	0	1

（三）重要机制实现方法

二、测试方案

（一）典型测试样例

（1）测试代码

```
lui $1,0xf0ff
ori $2,$1,0xabbd
sll $3,$1,3
sra $4,$1,3
srl $5,$1,3
addu $6,$4,$3
add $7,$4,$3
sub $8,$3,$4
nor $22,$3,$2
subu $8,$3,$4
or $9,$3,$4
nor $10,$9,$8
xor $11,$9,$8
mult $1,$2
mflo $12
mfhi $13
mtlo $5
```

```
mthi $6
mfhi $5
mflo $6
multu $1,$2
mflo $5
mfhi $6
div $1,$2
mflo $5
mfhi $6
divu $1,$2
mflo $5
mfhi $6
beq $5,$6,loop
nop
andi $9,$4,9
loop:jal loop1
nop
nop
nop
sll $0,$0,0
loop1:xori $10,$31,100
ori $9,$0,0x30a8
jalr $5,$9
nop
and $4,$3,$4
srav $19,$12,$9
srlv $20,$12,$9
sllv $21,$12,$9
sw $1,0($0)
sb $2,4($0)
```

```
sb $2,5($0)
sh $2,6($0)
lb $22,0($0)
lbu $22,1($0)
lh $23,2($0)
lhu $24,4($0)
lw $25,4($0)
addi $4,$4,4
addiu $5,$5,5
andi $6,$6,6
xori $7,$7,7
lui $4,3
ori $5,$0,4
slti $6,$5,4
sltiu $7,$5,4
slti $6,$5,5
sltiu $7,$5,5
bne $4,$5,loop2
nop
lui $4,4
loop3:
blez $0,loop4
nop
lui $4,3
loop2:blez $0,loop3
nop
loop4:
jal loop5
nop
lui $4,4
```

```
bgtz $31,loop6
nop
lui $4,4
loop5:
jr $31
nop
lui $4,4
loop6:
lui $4,0xf000
bltz $4,loop7
nop
lui $4,4
loop7:
slt $9,$4,$5
sltu $10,$5,$4
ori $16,0x316c
jalr $3,$16
nop
nop
add $4,$4,$4
j end
nop
sll $4,$4,3
end:
nop
```

（2）标准输出结果

```
@00003000: $ 1 <= f0ff0000
@00003004: $ 2 <= f0ffabbd
```

@00003008: $ 3 <= 87f80000

@0000300c: $ 4 <= fe1fe000

@00003010: $ 5 <= 1e1fe000

@00003014: $ 6 <= 8617e000

@00003018: $ 7 <= 8617e000

@0000301c: $ 8 <= 89d82000

@00003020: $22 <= 08005442

@00003024: $ 8 <= 89d82000

@00003028: $ 9 <= ffffe000

@0000302c: $10 <= 00001fff

@00003030: $11 <= 7627c000

@00003038: $12 <= 41430000

@0000303c: $13 <= 00e113f0

@00003048: $ 5 <= 8617e000

@0000304c: $ 6 <= 1e1fe000

@00003054: $ 5 <= 41430000

@00003058: $ 6 <= e2dfbfad

@00003060: $ 5 <= 00000001

@00003064: $ 6 <= ffff5443

@0000306c: $ 5 <= 00000000

@00003070: $ 6 <= f0ff0000

@0000307c: $ 9 <= 00000000

@00003080: $31 <= 00003088

@00003094: $10 <= 000030ec

@00003098: $ 9 <= 000030a8

@0000309c: $ 5 <= 000030a4

@000030a8: $19 <= 00414300

@000030ac: $20 <= 00414300

@000030b0: $21 <= 43000000

@000030b4: *00000000 <= f0ff0000

@000030b8: *00000004 <= 000000bd

@000030bc: *00000004 <= 0000bdbd

@000030c0: *00000004 <= abbdbdbd

@000030c4: $22 <= 00000000

@000030c8: $22 <= 00000000

@000030cc: $23 <= fffff0ff

@000030d0: $24 <= 0000bdbd

@000030d4: $25 <= abbdbdbd

@000030d8: $ 4 <= fe1fe004

@000030dc: $ 5 <= 000030a9

@000030e0: $ 6 <= 00000000

@000030e4: $ 7 <= 8617e007

@000030e8: $ 4 <= 00030000

@000030ec: $ 5 <= 00000004

@000030f0: $ 6 <= 00000000

@000030f4: $ 7 <= 00000000

@000030f8: $ 6 <= 00000001

@000030fc: $ 7 <= 00000001

@00003120: $31 <= 00003128

@00003128: $ 4 <= 00040000

@00003144: $ 4 <= f0000000

@00003154: $ 9 <= 00000001

@00003158: $10 <= 00000001

@0000315c: $16 <= 0000316c

@00003160: $ 3 <= 00003168

@0000316c: $ 4 <= e0000000

（3）搭建的 CPU 运行结果

38@00003000: $ 1 <= f0ff0000

42@00003004: $ 2 <= f0ffabbd

46@00003008: $ 3 <= 87f80000

50@0000300c: $ 4 <= fe1fe000

54@00003010: $ 5 <= 1e1fe000

58@00003014: $ 6 <= 8617e000

62@00003018: $ 7 <= 8617e000

66@0000301c: $ 8 <= 89d82000

70@00003020: $22 <= 08005442

74@00003024: $ 8 <= 89d82000

78@00003028: $ 9 <= ffffe000

82@0000302c: $10 <= 00001fff

86@00003030: $11 <= 7627c000

118@00003038: $12 <= 41430000

122@0000303c: $13 <= 00e113f0

134@00003048: $ 5 <= 8617e000

138@0000304c: $ 6 <= 1e1fe000

170@00003054: $ 5 <= 41430000

174@00003058: $ 6 <= e2dfbfad

226@00003060: $ 5 <= 00000001

230@00003064: $ 6 <= ffff5443

282@0000306c: $ 5 <= 00000000

286@00003070: $ 6 <= f0ff0000

302@0000307c: $ 9 <= 00000000

306@00003080: $31 <= 00003088

314@00003094: $10 <= 000030ec

318@00003098: $ 9 <= 000030a8

326@0000309c: $ 5 <= 000030a4

334@000030a8: $19 <= 00414300

338@000030ac: $20 <= 00414300

342@000030b0: $21 <= 43000000

342@000030b4: *00000000 <= f0ff0000

346@000030b8: *00000004 <= 000000bd

350@000030bc: *00000004 <= 0000bdbd

354@000030c0: *00000004 <= abbdbdbd

362@000030c4: $22 <= 00000000

366@000030c8: $22 <= 00000000

370@000030cc: $23 <= fffff0ff

374@000030d0: $24 <= 0000bdbd

378@000030d4: $25 <= abbdbdbd

382@000030d8: $ 4 <= fe1fe004

386@000030dc: $ 5 <= 000030a9

390@000030e0: $ 6 <= 00000000

394@000030e4: $ 7 <= 8617e007

398@000030e8: $ 4 <= 00030000

402@000030ec: $ 5 <= 00000004

406@000030f0: $ 6 <= 00000000

410@000030f4: $ 7 <= 00000000

414@000030f8: $ 6 <= 00000001

418@000030fc: $ 7 <= 00000001

454@00003120: $31 <= 00003128

470@00003128: $ 4 <= 00040000

482@00003144: $ 4 <= f0000000

498@00003154: $ 9 <= 00000001

502@00003158: $10 <= 00000001

506@0000315c: $16 <= 0000316c

514@00003160: $ 3 <= 00003168

522@0000316c: $ 4 <= e0000000

因为时钟周期 4，以及 20s 的复位信号 所以总共经过了 125 个 cycle

由计组提供的辅助测试中

```
standard pipeline-cycle: 125
    slow pipeline-cycle: 157
  accepted cycle range: [118, 147]
```

满足条件

三、思考题

（一） 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？

因为 32 位和 32 位做乘法的结果可能超过 32 位了，直接存会有溢出，所以多加了 HI,LO，如果直接 mult $1,$2,$3, $1 可能存不下结果。整合进 ALU 的话，对 HI,LO 的处理不方便了，ALU 的接口更多了，比较复杂。

除上述差异外，乘除部件由于乘除槽的原因还需要时钟信号等，由此可以看出乘除法本质上是和 ALU 不同的部件，根据高内聚低耦合的原则，应该对其单独建模

（二） 参照你对延迟槽的理解，试解释 “乘除槽”。

乘除法进入 E 级之后，进行运算，5 or 10 个周期之后才能出结果，在这个过程中，后面一条指令进入 D 级，如果是不相干的指令就继续执行。如果相关，就要等 5 or 10 个周期结束之后才能继续运算。将和该指令计算结果姓关的指令暂停，不相干的指令放走。

（三） 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）

字符串的读写、更改时。如”abcdefg”按照字访问，想取出一个字符，需要先取字，再取字符，而字节访问就比较简单，可以直接取一个字符。原因在于字符串的一个 ASCII 码就对应一个字节。此时按字节访问内存相对于按字访问内存性能上更有优势

（四） 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？

如果不需要写寄存器，直接将 A 译码为为 0，这样甚至可以直接省略 we。

（五） 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？？

MulDiv 的指令之间的冲突。应在在 D 级检测是否该指令要使用 MulDiv，暂停的条件是要使用 MDU 并且 MDU 处于 start 或 busy 的状态

```
lui $1,0xf0ff
ori $2,$1,0xabbd
sll $3,$1,3
sra $4,$1,3
srl $5,$1,3
addu $6,$4,$3
add $7,$4,$3
sub $8,$3,$4
nor $22,$3,$2
subu $8,$3,$4
or $9,$3,$4
nor $10,$9,$8
xor $11,$9,$8
mult $1,$2
mflo $12
mfhi $13
mtlo $5
mthi $6
mfhi $5
mflo $6
```

```
multu $1,$2
mflo $5
mfhi $6
div $1,$2
mflo $5
mfhi $6
divu $1,$2
mflo $5
mfhi $6
```

（六） 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。

按照教程的说法，将指令分类，枚举两条相关的指令，因为

根据前文的表格可以构建出需要测试的转发和暂停模块机制

一．D 级 rs

1.D 级 Rs 与 E 级 jal

2.D 级 Rs 与 E 级 jalr

3.D 级 Rs 与 E 级 mflo/mfhi

4.D 级 Rs 与 M 级 cal_r

5.D 级 Rs 与 M 级 cal_i

6.D 级 Rs 与 M 级 jal

7.D 级 Rs 与 M 级 jalr

8.D 级 Rs 与 M 级 mflo/mfhi

9.D 级 Rs 与 W 级 cal_r

10.D 级 Rs 与 W 级 cal_i

11.D 级 Rs 与 W 级 load rt

12.D 级 Rs 与 W 级 jal

13.D 级 Rs 与 W 级 jalr

14.D 级 Rs 与 W 级 jalr

每一项又分为：cal_r,cal_i,ld,st,beq,jr,jalr

二．D 级 Rt

1.D 级 Rt 与 E 级 jal

2.D 级 Rt 与 E 级 jalr

3.D 级 Rt 与 E 级 mflo/mfhi

4.D 级 Rt 与 M 级 cal_r

5.D 级 Rt 与 M 级 cal_i

6.D 级 Rt 与 M 级 jal

7.D 级 Rt 与 M 级 jalr

8.D 级 Rt 与 M 级 mflo/mfhi

9.D 级 Rt 与 W 级 cal_r

10.D 级 Rt 与 W 级 cal_i

11.D 级 Rt 与 W 级 load rt

12.D 级 Rt 与 W 级 jal

13.D 级 Rt 与 W 级 jalr

14.D 级 Rt 与 W 级 jalr

每一项又分为：cal_r,st,beq

三．E 级 Rs

1.E 级 Rs 与 M 级 cal_r

2.E 级 Rs 与 M 级 cal_i

3.E 级 Rs 与 M 级 jal

4.E 级 Rs 与 M 级 jalr

5.E 级 Rs 与 M 级 mflo/mfhi

6.E 级 Rs 与 W 级 cal_r

7.E 级 Rs 与 W 级 cal_i

8.E 级 Rs 与 W 级 load

9.E 级 Rs 与 W 级 jal

10.E 级 Rs 与 W 级 jalr

11.E 级 Rs 与 W 级 mflo/mfhi

每一项又分为：cal_r,cal_i,ld,st

四．E 级 Rt

1.E 级 Rt 与 M 级 cal_r

2.E 级 Rt 与 M 级 cal_i

3.E 级 Rt 与 M 级 jal

4.E 级 Rt 与 M 级 jalr

5.E 级 Rt 与 M 级 mflo/mfhi

6.E 级 Rt 与 W 级 cal_r

7.E 级 Rt 与 W 级 cal_i

8.E 级 Rt 与 W 级 load

9.E 级 Rt 与 W 级 jal

10.E 级 Rt 与 W 级 jalr

11.E 级 Rt 与 W 级 mflo/mfhi

每一项又分为：cal_r,st

五．M 级 Rt

1.M 级 Rt 与 W 级 cal_r

2.M 级 Rt 与 W 级 cal_i

3.M 级 Rt 与 W 级 load

4.M 级 Rt 与 W 级 jal

5.M 级 Rt 与 W 级 jalr

6.M 级 Rt 与 W 级 mflo/mfhi

每一项又分为：st

暂停：

一．Beq_rs/rt

(1)E 级 cal_r_rd

(2) E 级 cal_i_rt

(3) E 级 load_rt

(4) M 级 load_rt

二．Cal_r_rs/rt

E 级 load_rt

三．Cal_i_rs

E 级 load_rt

四．load_rs

E 级 load_rt

五．store_rs

E 级 load_rt

六．jr_rs

(1)E 级 cal_r_rd

(2) E 级 cal_i_rt

(3) E 级 load_rt

(4) M 级 load_rt

七. mult multu div divu

mflo mfhi mtlo mthi 导致的暂

（七） 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

分布式译码，减少寄存器的接口，防止编写失误

将指令分类，便于统一管理和理解，也方便转发暂停的控制

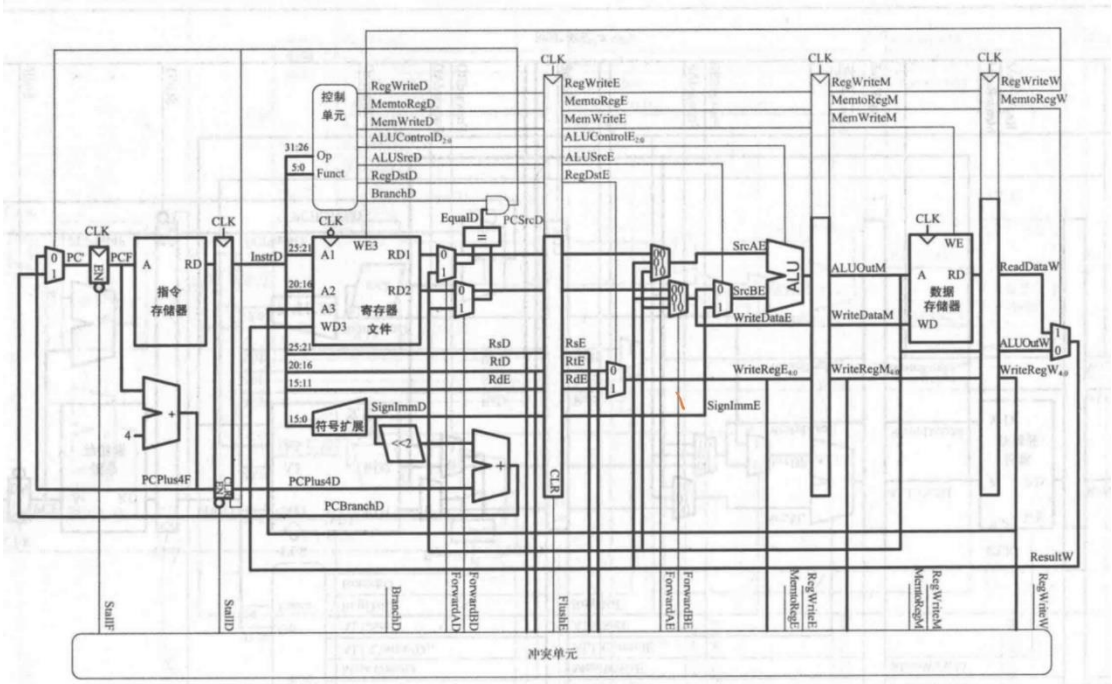

图7-58　处理所有冲突的流水线处理器