MIPS 微体系实验报告

一、CPU 设计方案综述

（一）总体设计概述

使用 Veriloh 开发一个的 MIPS 微系统，总体概述如下：

a. MIPS 处理器须为流水线设计，MIPS 微系统须支持中断和异常。

b. 除本文明确的规范和补充声明外，MIPS 微系统设计以《See MIPS Run Linux》(下文简称《SMRL》)作为标准。《SMRL》的标准与 MARS 的行为存在一定差异，在测试时不以 MARS 为准。

c. MIPS 处理器位流水线 CPU 有以下要求

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，mfc0、mtc0、eret}
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. CPU

新增模块，顶层 CP0

1. CP0

（一）端口设置

端口	方向	位宽	描述
A1	I	5	指定 4 个寄存器中的一个，将其存储的数据读出到 RD
A2	I	5	指定 4 个寄存器中的一个，作为写入的目标寄存器
WDin	I	32	写入寄存器的数据信号
PC	I	32	目前传入的 EPC 值
EXCCode	I	5	目前传入的 EXCCode 值
Delay	I	32	目前传入的 BD 值
HWInt	I	6	外部硬件中断信号
WE	I	1	写使能信号，高电平有效
EXLClr	I	1	传入 eret 指令时将 SR 的 EXL 位置 0 ，高电平有效
clk	I	1	时钟信号
reset	I	1	同步复位信号
Request	O	1	输出当前的中断请求
EPCOut	O	32	输出当前 EPC 寄存器中的值
WDout	O	32	输出 A 指定的寄存器中的数据
Response	O	1	检测 CPU 是否对外部中断产生响应，从而决定是否去写 0x0000_7f20

(一)功能说明

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来且同步复位信号有效时，将所有寄存器的值设置为 0x00000000。
2	读数据	读出 A1 地址对应寄存器中存储的数据到 RD；当 WE 有效时会将 WD 的值会实时反馈到对应的 RD，当 ERET 有

		效时会将 EXL 置 0，即内部转发。
3	写数据	当 WE 有效且时钟上升沿到来时，将 WD 的数据写入 A2 对应的寄存器中。
4	中断处理	根据各种传入信号和寄存器的值判断当前是否要进行中断，将结果输出到 Request 。

以下是 P6 构建的模块

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 寄存器	当 MDWrite 信号有效时，将 rs 的值写入 HI 或 LO
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

2. Bridge

系统桥是处理 CPU 与外设（两个计时器）之间信息交互的通道

CPU 中 store 类指令需要储存的数据经过存储模块处理后会通过 m_data_addr, m_data_byteen, m_data_wdata 三个信号输出到桥中，桥会根据写使能 m_data_byteen 和地址 m_data_addr 来判断到底写的是内存还是外设，然后给出正确的写使能

load 类指令则是全部把地址传递给每个外设和 DM 中，然后桥根据地址选择从应该反馈给 CPU 从哪里读出来的数据，然后 DE 在处理读出的数据，反馈正确的结果

（三）重要机制实现方法

1. 转发机制

首先我们对每条指令根据其得到的结果类型进行一个分类：

```
//Forwarding
   assign   resOp =  (cal_r|cal_i|shift)?`ALU: //the
source of the value of the reg, namely Tnew
                (load)?`DM:
                (jal|jalr)?`PC:
                (mflo)?`LOout:
                (mfhi)?`HIout:`NW
```

其在转发中相当于需要转发数据的选择信号编码信息，而其在暂缓中相当于 T_new 的用法

接下来分析需要转发的位点。当某一部件需要使用 GPR 中的值时，如果此时这

个值存在于后续某个流水线寄存器中，而还没来得及写入 GPR，我们就需要通过转发（旁路）机制将这个值从流水线寄存器中送到该部件的输入处。

根据我们对数据通路的分析，这样的位点有：

1．D 级比较器的两个输入（含 NPC 逻辑中寄存器值的输入）；

2．E 级 ALU 的两个输入；

3．M 级 DM 的输入。

因此我们对于每个周期各个阶段的指令进行转发的判断与选择，并对选择信号进行编码

```
assign MALUAE = (A1E == 0)                    ?0:
                ((A1E == A3M) && ( resOpM ==`ALU))?`ME_ALU:
                ((A1E == A3M) && (resOpM == `PC))  ?`ME_PC:
                ((A1E == A3M) && (resOpM ==`HIout))?`ME_HI:
                ((A1E == A3M) && (resOpM ==`LOout))?`ME_LO:
                ((A1E == A3W) && (resOpW == `ALU))?`WE_ALU:
                ((A1E == A3W) && (resOpW == `PC)) ?`WE_PC:
                ((A1E == A3W) && (resOpW == `DM)) ?`WE_DM:
                ((A1E == A3W ) && (resOpW ==`HIout))?`WE_HI:
                ((A1E == A3W) && (resOpW ==`LOout))?`WE_LO:
                                              0;
   //MFALUBE
   assign MALUBE = (A2E == 0)                    ?0:
                ((A2E == A3M) && (resOpM == `ALU))?`ME_ALU:
                ((A2E == A3M) && (resOpM ==`PC))?`ME_PC:
                ((A2E == A3M) && (resOpM ==`HIout))?`ME_HI:
                ((A2E == A3M) && (resOpM ==`LOout))?`ME_LO:
                ((A2E == A3W) && (resOpW == `ALU))?`WE_ALU:
                ((A2E == A3W) && (resOpW == `PC))  ?`WE_PC:
```

```
            ((A2E == A3W) && (resOpW == `DM))  ?`WE_DM:
            ((A2E == A3W) && (resOpW ==`HIout))?`WE_HI:
            ((A2E == A3W) && (resOpW ==`LOout))?`WE_LO:
                                        0;

//MCMP1D
assign MCMP1D = (A1D == 0)                  ?0:
            ((A1D == A3M) && (resOpM == `ALU))?`MD_ALU:
            ((A1D == A3M) && (resOpM == `PC))  ?`MD_PC:
            ((A1D == A3M) && (resOpM ==`HIout))?`MD_HI:
            ((A1D == A3M) && (resOpM ==`LOout))?`MD_LO:
            ((A1D == A3W) && (resOpW == `ALU))?`WD_ALU:
            ((A1D == A3W) && (resOpW == `PC))  ?`WD_PC:
            ((A1D == A3W) && (resOpW == `DM))  ?`WD_DM:
            ((A1D == A3W) && (resOpW ==`HIout))?`WD_HI:
            ((A1D == A3W) && (resOpW==`LOout)) ?`WD_LO:
                                        0;
//MFCMP2D
assign MCMP2D = (A2D == 0)                  ?0:
            ((A2D == A3M) && (resOpM == `ALU))?`MD_ALU:
            ((A2D == A3M) && (resOpM == `PC))  ?`MD_PC:
            ((A2D == A3M) && (resOpM ==`HIout))?`MD_HI:
            ((A2D == A3M) && (resOpM ==`LOout))?`MD_LO:
            ((A2D == A3W) && (resOpW == `ALU))?`WD_ALU:
            ((A2D == A3W) && (resOpW == `PC))  ?`WD_PC:
            ((A2D == A3W) && (resOpW == `DM))  ?`WD_DM:
            ((A2D == A3W) && (resOpW ==`HIout))?`WD_HI:
            ((A2D == A3W) && (resOpW ==`LOout))?`WD_LO:
                                        0;
```

```
assign MDMM =  (A2M == 0)                    ?0:
            ((A2M == A3W) && (resOpW == `ALU))?`WM_ALU:
            ((A2M == A3W) && (resOpW == `PC))  ?`WM_PC:
            ((A2M == A3W) && (resOpW == `DM))  ?`WM_DM:
            ((A2M == A3W) && (resOpW ==`HIout))?`WM_HI:
            ((A2M == A3W) && (resOpW ==`LOout))?`WM_LO:
                                               0;
```

对应的就有以下数据的选择,以 CMP 的选择为例

```
MUX_10_32 mfcm2d(
      .in0(RD2),
      .in1(ResM),
      .in2(PC4M+4),
      .in3(ResW),
      .in4(PC4W+4),
      .in5(MemRDW),
      .in6(HIM),
      .in7(LOM),
      .in8(HIW),
      .in9(LOW),
      .sel(MCMP2D),
      .out(MFCMP2D)
   );
```

2. 暂停机制

按照 T_{use} 和 T_{new}，对每个进入 D 阶段的指令以及正在运行执行的指令进行判断，以指令为导向，判断是否应该暂停

如前文所说，得到 resOp 可以看作 T_{new} 的一种形式

```
   stall_rs0_e1 = Tuse_rs0 && (resOpE==`ALU ||
resOpE==`HIout || resOpE==`LOout) && (Instr[`rs]==A3E);
```

```
      stall_rs0_e2 = Tuse_rs0 && (resOpE==`DM) &&
(Instr[`rs]==A3E);
      stall_rs1_e2 = Tuse_rs1 && (resOpE==`DM) &&
(Instr[`rs]==A3E);
      stall_rs0_m1 = Tuse_rs0 && (resOpM==`DM) &&
(Instr[`rs]==A3M);
      stall_rs = stall_rs0_e1 || stall_rs0_e2 ||
stall_rs1_e2 || stall_rs0_m1;

      stall_rt0_e1 = Tuse_rt0 && (resOpE==`ALU ||
resOpE==`HIout || resOpE==`LOout) && (Instr[`rt]==A3E);
      stall_rt0_e2 = Tuse_rt0 && (resOpE==`DM) &&
(Instr[`rt]==A3E);
      stall_rt1_e2 = Tuse_rt1 && (resOpE==`DM) &&
(Instr[`rt]==A3E);
      stall_rt0_m1 = Tuse_rt0 && (resOpM==`DM) &&
(Instr[`rt]==A3M);
      stall_rt = stall_rt0_e1 || stall_rt0_e2 ||
stall_rt1_e2 || stall_rt0_m1;

      stall = stall_rs||stall_rt;
```

因此可以得到以下暂停表达式的构造

暂停转发表如下

IF/ID当前指令		
指令类型	源寄存器	Tuse
beq	rs/rt	0
cal_r	rs/rt	1
cal_i	rs	1
load	rs	1
store	rs	1
store	rt	2
jr	rs	0
jalr	rs	0

ID/EX					EX/MEM					MEM/WB				
Tnew					Tnew					Tnew				
cal_r	cal_i	load	jal	jalr	cal_r	cal_i	load	jal	jalr	cal_r	cal_i	load	jal	jalr
1/rd	1/rt	2/rt	0/31	0/rs	0/rd	0/rt	1/rt	0/31	0/rs	0/rd	0/rt	0/rt	0/31	0/rs

IF/ID当前指令			ID/EX			EX/MEM
			Tnew			Tnew
指令类型	源寄存器	Tuse	cal_r	cal_i	load	load
			1/rd	1/rt	2/rt	1/rt
beq	rs/rt	0	暂停	暂停	暂停	暂停
cal_r	rs/rt	1			暂停	
cal_i	rs	1			暂停	
load	rs	1			暂停	
store	rs	1			暂停	
store	rt	2				
jr	rs	0	暂停	暂停	暂停	暂停
jalr	rs	0	暂停	暂停	暂停	暂停

						ID/EX			EX/MEM					MEM/WB					
						Tnew			Tnew					Tnew					
流水级	源寄存器	涉及指令	MUX	控制信号	输入0	jal	jalr	mflo mfhi	cal_r	cal_i	jal	jalr	mflo mfhi	cal_r	cal_i	load	mflo mfhi	jal	jalr
IR_D	rs	cal_r,cal_i,ld,st,beq,jr,jalr	MFRSD	ForwardRSD	RF.RD1	PC8_E	PC8_E	MD_out	AO_M	AO_M	PC8_M	PC8_M	MDO_M	MUX_WD	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
IR_D	rt	cal_r,st,beq	MFRTD	ForwardRTD	RF.RD2	PC8_E	PC8_E	MD_out	AO_M	AO_M	PC8_M	PC8_M	MDO_M	MUX_WD	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
IR_E	rs	cal_r,cal_i,ld,st	MFRSE	ForwardRSE	RS_E				AO_M	AO_M	PC8_M	PC8_M	MDO_M	MUX_WD	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
ALU	rt	cal_r,st	MFRTE	ForwardRTE	RT_E				AO_M	AO_M	PC8_M	PC8_M	MDO_M	MUX_WD	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W
IR_M DM	rt	st	MFRTM	ForwardRTM	RT_M									MUX_WD	MUX_WD	MUX_WD	MUX_WD	PC8_W	PC8_W

3．分布式译码

与 P4、P5 不同，P6 支持的指令多达 50+如果按照集中译码，再将给类信号传递下去，由于指令的增多必然也会造成要流水的信号增多，导致流水线寄存器的接口过多，流水时容易出错。因此，选用分布式译码的方式，在需要译码的地方实例化一个 decoder，（decoder 为译码模块，包含了所有需要的信号），每次需要流水的只有指令。尽管这种方法在实际运用时会运用大量的集体管，造成成本的上涨，但在此课程中以“高内聚低耦合”为指导原则，暂不考虑成本建造难度等方面

4．异常判断机制

将异常码 ExcCode、是否处于延迟槽中的判断信号 Delay 和当前 PC（如果时取指地址异常则传递错误的 PC 值）一直跟着流水线到达 M 级直至提交至 CP0，由 CP0 综合判断分析是否响应该异常

如果需要响应该异常，则 CP0 输出 Request 信号置为 1，此时 FD、DE、DM、MW 寄存器响应 Request 信号，清空 Instr，将 PC 值设为 0x4180，然后输入 F 级的 NPC 也被置为 0x4180，下一条指令从 0x4180 开始执行

当外设和系统外部输入中断信号时，CP0 同样也会确认是否响应该中断，然后把 Req 置为 1，执行相同的操作

当系统外部输入中断信号时，CP0 还会输出一个 Response 信号指示是否响应外部中断信号，如果响应则系统会相应去写 0x7f20 地址，从而时外部中断信号停止

同时有关传入的异常 PC 需要按照以下原理传递

1. 如果发生异常的指令是延迟槽指令，那么我们要求返回程序时仍然返回这条指令所属的跳转指令。也就是说“异常延迟槽回到跳转”。
2. 如果发生异常的指令是跳转指令，那么要求执行完延迟槽。。
3. 如果发生异常的指令是乘除指令的下一条，我们允许乘除指令不被撤回。也就是对于 M 错误指令，W 乘除指令的情况，此时乘除槽正在计算，本来在异常处理时可能会覆盖乘除槽的结果，但是我们约定不会这么做。但是注意，如果是 E 乘除指令，M 错误指令，要保证乘除指令不执行。

5．空泡处理

我在阻塞时我们会往流水线中插入 nop，这个 nop 的 pc 和 bd 信号都是 0。此时宏观 PC 会显示错误的值。并且如果此时发生了中断，就会导致 EPC 存入错误的值。

按照道理来讲，如果插入了 nop，它的 PC 和 bd 应该是目前这条指令的值。比如指令序列是 D-add, E-nop, M-lw，那么 nop 的 pc 是 add。这样当 nop 到 M 级时的宏观 PC 才是正确的。

```
//F-D reg
always@(posedge clk)begin
     if(reset || Request)
     begin
        InstrD<=0;
        PC4D<=Request ? 32'h0000_4184 : 32'd0;
        EXCCodeD<=0;
        DelayD<=0;
```

```
        end
        else if(en  && (stall_md == 1'b0) )
        begin
           InstrD<=InstrF;
           PC4D<=ADD4;
           EXCCodeD<=EXCCodeF;
           DelayD<=DelayF;
        end

    ////////////////////

    //D-E reg
    always@(posedge clk)
      begin
        if(reset||flush||stall_md||Request || stall)
        begin
           RD1E <= 0;
           RD2E <= 0;
           imm32E <= 0;
           PC4E <= (stall_md||stall)? PC4D : (Request ?
32'h0000_4184 :0);
           InstrE <= 0;
           checkE <= 0;
           DelayE <= (stall_md || stall) ? DelayD : 0;
           EXCCodeE <= 0;
        end
        else
        begin
           RD1E <= RD1D;
           RD2E <= RD2D;
```

```
            imm32E <= imm32D;
            PC4E <= PC4D;
            InstrE <= InstrD;
            checkE <= checkD;
            DelayE <= DelayD;
            EXCCodeE <= EXCCodeD;
         end
      end

//////////////////////

//E-M Reg
always@(posedge clk)
   begin
      if(reset || Request)
      begin
         ResM <= 0;
         MemWDM <= 0;
         PC4M <= (Request? 32'h0000_4184 :32'd0);
      HIM <= 0;
         LOM <= 0;
         InstrM <= 0;
         checkM <= 0;
         DelayM <= 0;
         EXCCodeM <= 0;
         Exc_AddrM <= 0;
      end
      else
      begin
         MemWDM <= MemWDE;
```

```
            ResM <= ResE;
            PC4M <= PC4E;
            HIM <= HIE;
            LOM <= LOE;
            InstrM <= InstrE;
            checkM <= checkE;
            DelayM <= DelayE;
            EXCCodeM <= EXCCodeE;
            Exc_AddrM <= Exc_AddrE;
        end
```

二、测试方案

（一）取值异常

```
.text

li $28, 0
li $29, 0

# jr PC mod 4 not 0
la $1, label1
la $2, label1
addiu $1, $1, 1
jr $1
nop
label1:

# jr PC < 0x3000
```

```
li $1, 0x2996
la $2, label2
jr $1
nop
label2:

# jr PC > 0x4ffc
li $1, 0x4fff
la $2, label3
jr $1
nop
label3:

end:j end

.ktext 0x4180
mfc0 $12, $12
mfc0 $13, $13
mfc0 $14, $14
mtc0 $2, $14
eret
ori $1, $0, 0
```

（二）其他的异常

```
.text

li $28, 0
li $29, 0

lw $1, 1($0)
```

```
lh $1, 1($0)
lhu $1, 1($0)

lh $1, 0x7f00($0)
lhu $1, 0x7f04($0)
lb $1, 0x7f08($0)
lbu $1, 0x7f10($0)
lb $1, 0x7f14($0)
lb $1, 0x7f18($0)

li $2, 0x7fffffff
lw $1, 1($2)
lh $1, 1($2)
lhu $1, 1($2)
lb $1, 1($2)
lbu $1, 1($2)

lw $1, 0x3000($0)
lh $1, 0x4000($0)
lhu $1, 0x6000($0)
lb $1, 0x7f0c($0)
lbu $1, 0x7f1c($0)

sw $1, 1($0)
sh $1, 1($0)

sh $1, 0x7f00($0)
sb $1, 0x7f04($0)
sh $1, 0x7f08($0)
sb $1, 0x7f10($0)
```

```
sh $1, 0x7f14($0)
sb $1, 0x7f18($0)

li $2, 0x7fffffff
sw $1, 1($2)
sh $1, 1($2)
sb $1, 1($2)

sw $1, 0x7f08($0)
sh $1, 0x7f08($0)
sb $1, 0x7f08($0)
sw $1, 0x7f18($0)
sh $1, 0x7f18($0)
sb $1, 0x7f18($0)

sw $1, 0x3000($0)
sh $1, 0x4000($0)
sh $1, 0x6000($0)
sb $1, 0x7f0c($0)
sb $1, 0x7f1c($0)

msub $1, $2

li $1, 0x7fffffff
add $1, $1, $1
addi $1, $1, 1
li $1, 0x80000000
add $1, $1, $1
addi $1, $1, -1
sub $1, $1, $2
```

```
sub $1, $2, $1

end:j end


.ktext 0x4180
mfc0 $12, $12
mfc0 $13, $13
mfc0 $14, $14
addi $14, $14, 4
mtc0 $14, $14
eret
ori $1, $0, 0
```

（三）计时器异常

```
.text

li $28, 0
li $29, 0

lw $1, 1($0)
lh $1, 1($0)
lhu $1, 1($0)

lh $1, 0x7f00($0)
lhu $1, 0x7f04($0)
lb $1, 0x7f08($0)
lbu $1, 0x7f10($0)
lb $1, 0x7f14($0)
```

```
lb $1, 0x7f18($0)

li $2, 0x7fffffff
lw $1, 1($2)
lh $1, 1($2)
lhu $1, 1($2)
lb $1, 1($2)
lbu $1, 1($2)

lw $1, 0x3000($0)
lh $1, 0x4000($0)
lhu $1, 0x6000($0)
lb $1, 0x7f0c($0)
lbu $1, 0x7f1c($0)

sw $1, 1($0)
sh $1, 1($0)

sh $1, 0x7f00($0)
sb $1, 0x7f04($0)
sh $1, 0x7f08($0)
sb $1, 0x7f10($0)
sh $1, 0x7f14($0)
sb $1, 0x7f18($0)

li $2, 0x7fffffff
sw $1, 1($2)
sh $1, 1($2)
sb $1, 1($2)
```

```
sw $1, 0x7f08($0)
sh $1, 0x7f08($0)
sb $1, 0x7f08($0)
sw $1, 0x7f18($0)
sh $1, 0x7f18($0)
sb $1, 0x7f18($0)

sw $1, 0x3000($0)
sh $1, 0x4000($0)
sh $1, 0x6000($0)
sb $1, 0x7f0c($0)
sb $1, 0x7f1c($0)

msub $1, $2

li $1, 0x7fffffff
add $1, $1, $1
addi $1, $1, 1
li $1, 0x80000000
add $1, $1, $1
addi $1, $1, -1
sub $1, $1, $2
sub $1, $2, $1

end:j end


.ktext 0x4180
mfc0 $12, $12
mfc0 $13, $13
```

```
mfc0 $14, $14
addi $14, $14, 4
mtc0 $14, $14
eret
ori $1, $0, 0
```

（四）延迟槽异常

```
.text

li $28, 0
li $29, 0

li $1, 1
bne $0, $0, end
lw $1, 1($0)
li $1, 1
bne $0, $0, end
lh $1, 1($0)
li $1, 1
bne $0, $0, end
lhu $1, 1($0)

li $1, 1
bne $0, $0, end
lh $1, 0x7f00($0)
li $1, 1
bne $0, $0, end
lhu $1, 0x7f04($0)
li $1, 1
```

```
bne $0, $0, end
lb $1, 0x7f08($0)
li $1, 1
bne $0, $0, end
lbu $1, 0x7f10($0)
li $1, 1
bne $0, $0, end
lb $1, 0x7f14($0)
li $1, 1
bne $0, $0, end
lb $1, 0x7f18($0)

li $2, 0x7fffffff
li $1, 1
bne $0, $0, end
lw $1, 1($2)
li $1, 1
bne $0, $0, end
lh $1, 1($2)
li $1, 1
bne $0, $0, end
lhu $1, 1($2)
li $1, 1
bne $0, $0, end
lb $1, 1($2)
li $1, 1
bne $0, $0, end
lbu $1, 1($2)

li $1, 1
```

```
bne $0, $0, end
lw $1, 0x3000($0)
li $1, 1
bne $0, $0, end
lh $1, 0x4000($0)
li $1, 1
bne $0, $0, end
lhu $1, 0x6000($0)
li $1, 1
bne $0, $0, end
lb $1, 0x7f0c($0)
li $1, 1
bne $0, $0, end
lbu $1, 0x7f1c($0)

li $1, 1
bne $0, $0, end
sw $1, 1($0)
li $1, 1
bne $0, $0, end
sh $1, 1($0)

li $1, 1
bne $0, $0, end
sh $1, 0x7f00($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f04($0)
li $1, 1
bne $0, $0, end
```

```
sh $1, 0x7f08($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f10($0)
li $1, 1
bne $0, $0, end
sh $1, 0x7f14($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f18($0)

li $2, 0x7fffffff
li $1, 1
bne $0, $0, end
sw $1, 1($2)
li $1, 1
bne $0, $0, end
sh $1, 1($2)
li $1, 1
bne $0, $0, end
sb $1, 1($2)

li $1, 1
bne $0, $0, end
sw $1, 0x7f08($0)
li $1, 1
bne $0, $0, end
sh $1, 0x7f08($0)
li $1, 1
bne $0, $0, end
```

```
sb $1, 0x7f08($0)
li $1, 1
bne $0, $0, end
sw $1, 0x7f18($0)
li $1, 1
bne $0, $0, end
sh $1, 0x7f18($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f18($0)

li $1, 1
bne $0, $0, end
sw $1, 0x3000($0)
li $1, 1
bne $0, $0, end
sh $1, 0x4000($0)
li $1, 1
bne $0, $0, end
sh $1, 0x6000($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f0c($0)
li $1, 1
bne $0, $0, end
sb $1, 0x7f1c($0)

li $1, 1
bne $0, $0, end
msub $1, $2
```

```
li $1, 0x7fffffff
li $11, 1
bne $0, $0, end
add $1, $1, $1
li $11, 1
bne $0, $0, end
addi $1, $1, 1
li $1, 0x80000000
li $11, 1
bne $0, $0, end
add $1, $1, $1
li $11, 1
bne $0, $0, end
addi $1, $1, -1
li $11, 1
bne $0, $0, end
sub $1, $1, $2
li $11, 1
bne $0, $0, end
sub $1, $2, $1

end:j end


.ktext 0x4180
mfc0 $12, $12
mfc0 $13, $13
```

```
mfc0 $14, $14
addi $14, $14, 8
mtc0 $14, $14
eret
ori $1, $0, 0
```

三、思考题

（一） 我们计组课程一本参考书目标题中有“硬件/软件接口”接口字样，那么到底什么是“硬件/软件接口”？（Tips：什么是接口？和我们到现在为止所学的有什么联系？）

硬件/软件接口”是指令。硬件实现物理功能，并按照规约可以被相应的指令所操控。软件通过规约使用相应的指令操控硬件完成相应的功能，从而达到软件所期望的效果，同时也包括了有关异常的处理程序。指令在这个过程中实现了硬件软件的对接，因此是“硬件/软件接口”。

（二） BE 部件对所有的外设都是必要的吗？

仅对需要按字节或半字访问内存的外设必要

（三） 请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并分别针对每一种模式绘制状态转移图。

见计时器文档

（四） 请开发一个主程序以及定时器的 exception handler。整个系统完成如下功能：

1. 定时器在主程序中被初始化为模式 0；

2.定时器倒计数至 0 产生中断；

3.handler 设置使能 Enable 为 1 从而再次启动定时器的计数器。2 及 3 被无限重复。

4.主程序在初始化时将定时器初始化为模式 0，设定初值寄存器的初值为某个值，如 100 或 1000。（注意，主程序可能需要涉及对 CP0.SR 的编程，推荐阅读过后文后再进行。）

```
.text
li $12, 0x0401 #get SR
mtc0 $12, $12
li $1, 100
li $2, 9
sw $1, 0x7f04($0)
sw $2, 0x7f00($0)

dead_loop:
j dead_loop
nop

.ktext 0x4180
li $1, 100
li $2, 9
sw $1, 0x7f04($0)
sw $2, 0x7f00($0)
```

```
eret
```

（五） 。请查阅相关资料，说明鼠标和键盘的输入信号是如何被 CPU 知晓的？

鼠标点击、位置移动或按下键盘时，将产生中断信号，从而使 CPU 进入相应的中断处理程序。

有许多人说写 7f20 需要延一个周期，但是笔者和两个舍友均未延一个周期也都过了 P7，不延迟半个周期在识别到 interrupt 时确实没办法让中断停下，但之后从异常处理程序返回时，是可以让中断停下的，我认为这里可能与 P7 实现方式有关

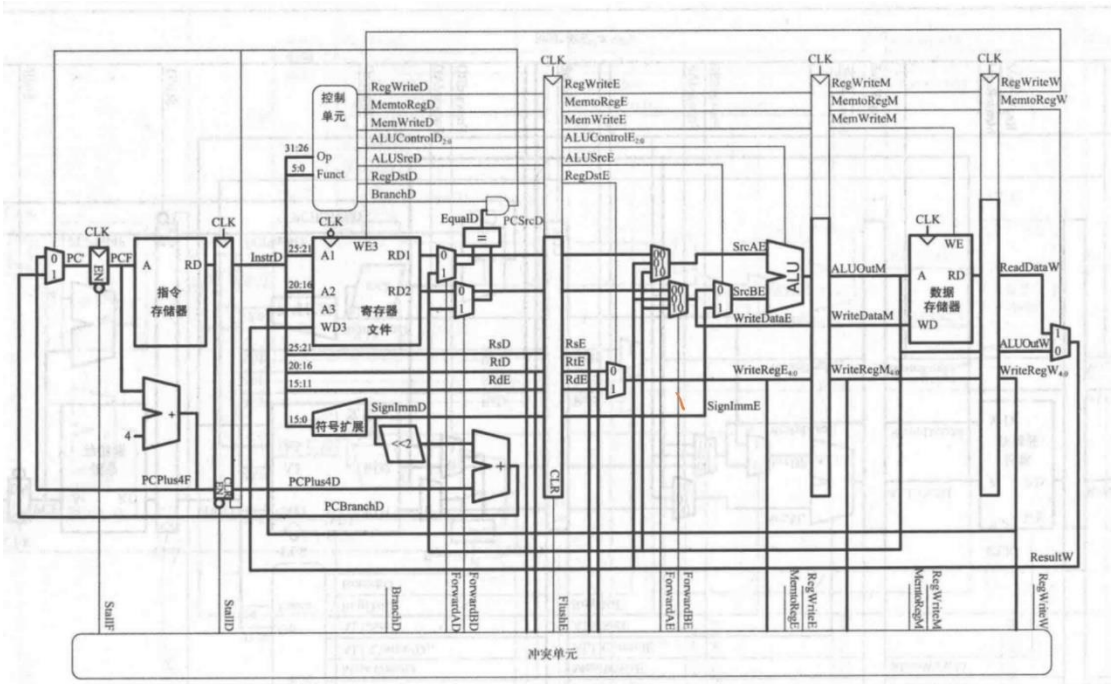

图7-58　处理所有冲突的流水线处理器