P7 - MIPS 微体系——异常与中断

一. 整体结构

1. 处理器应支持 MIPS-lite4 指令集。

MIPS-C4 = {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、ERET、MFC0、MTC0}

2. 处理器为流水线设计。

二. 模块规格

1. pc. v

文件	模块接口定义
pc.v	module pc(input clk, input reset, input en, input[31:0] next_pc, output reg[31:0] pc, output [4:0] excode_F);

模块接口

信号名	方向	功能描述
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
en	I	使能信号
next_pc	I	更新的 PC（时钟上升沿更新）

Pc	I	PC

功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，PC 被设置为 0x00003000
2	更新 pc	时钟上升沿时改变 pc=next_pc

2. im.v

文件	模块接口定义
im.v	module im(input [31:0] PC, output[31:0] instruction);

模块接口

信号名	方向	功能描述
PC[31:0]	I	32 位 PC
Instruction[31:0]	O	32 位当前指令

功能定义

序号	功能名称	功能描述
1	取指令	根据 PC 从 IM 中取出指令

3. ID.v

模块接口

文件	模块接口定义
ID.v	module ID(input clk, input reset, input en, input [31:0] Instr, input [31:0] PC, output reg[31:0] IR_D, output reg[31:0] PC_D, output reg[31:0] PC4_D, output reg[31:0] PC8_D, output [4:0] excode_D);

功能定义

序号	功能名称	功能描述
1	IF/ID 流水线寄存器	保存 PC,IR 等信号的值

4.grf.v

文件	模块接口定义
grf.v	module grf(input clk, input reset, input RegWrite, input [4:0] RA1, input [4:0] RA2, input [4:0] WA, input [31:0] WD, input [31:0] PC, output [31:0] RD1, output [31:0] RD2);

模块接口

信号名	方向	功能描述
WD[31:0]	I	写入数据的输入
RA1[4:0]	I	读寄存器地址 1
RA2[4:0]	I	读寄存器地址 2
WA[4:0]	I	写寄存器地址
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
PC[31:0]	I	当前 PC
RegWrite	I	是否可以写入控制信号(随时都可以读出) 1：可以写 0：不可以写
RD1[31:0]	O	32 位数据输出 1
RD2[31:0]	O	32 位数据输出 2

功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，所有寄存器被设置为 0x00000000
2	读寄存器	根据输入的寄存器地址读出 32 位数据

3	写寄存器	根据输入的地址，把输入的数据写进所选的寄存器

5. cmp. v

文件	模块接口定义
cmp. v	module cmp(input [31:0] D1, input [31:0] D2, output Equal);

模块接口

信号名	方向	功能描述
D1[31:0]	I	输入 1
D2[31:0]	I	输入 2
Equal	0	判断两个输入是否相等

功能定义

序号	功能名称	功能描述
1	比较器	比较两个输入是否相等

6. ext. v

文件	模块接口定义
ext. v	module ext(input [15:0] in, input [1:0] ExtOp, output reg [31:0] out);

模块接口

信号名	方向	功能描述
In[15:0]	I	16 位数据输入
Out[31:0]	0	32 位数据输出
ExtOp[1:0]	I	扩展方式选择信号

功能定义

序号	功能名称	功能描述
1	高位符号扩展	高 16 位补符号位
2	高位 0 扩展	高 16 位补 0

3.	低位 0 扩展	低 16 位补 0

7. npc.v

文件	模块接口定义
npc.v	module npc(input [31:0] PC4, input [31:0] PC4D, input [25:0] I26, input [31:0] MFRSD, input [31:0] EPC, input Zero, input more, input less, input if_beq, input if_bne, input if_bgtz, input if_blez, input if_bgez, input if_bltz, input if_j,//j 或 jal input[1:0] PC_sel, input Interrupt, output reg[31:0] next_pc);

模块接口

信号名	方向	功能描述
PC4	I	PC+4 的值(对应于无跳转 直接执行下一句)
PC4D	I	D 级 PC+4
I26	I	26 位立即数
MFRSD	I	转发 PC 的 MUX 结果（jr jalr 需要转发）
Zero	I	比较两个数是否相等的结果
Branch	I	判断是不是 beq 类指令
If_j	I	判断是不是 j/jal 指令
PC_sel[1:0]	I	PC 的选择信号
Next_pc	O	更新的 pc 值

功能定义

序号	功能名称	功能描述
1	更新 PC	更新 PC

8. controller.v （分布式译码 实例化 4 个）

文件	模块接口定义
controller.v	module controller(input [5:0] op, input [5:0] func, input [4:0] rs, input [4:0] rt, output reg[3:0] ALUCtrl, output reg[1:0] RegDst, output reg ALUASrc, output reg ALUBSrc, output reg RegWrite, output reg MemRead, output reg MemWrite, output reg [2:0] MemtoReg, output reg [1:0]ExtOp, output reg if_beq, output reg if_bne, output reg if_blez, output reg if_bgez, output reg if_bltz, output reg if_bgtz, output reg if_j, output reg [1:0]PCsel, output reg if_sh, output reg if_sb, output reg[2:0] dataOp, output reg[1:0] multdivOp, output reg start, output reg if_mthi, output reg if_mtlo, output reg if_mfhi, output reg if_mflo, output reg if_1, output reg if_2, output reg if_3, output reg cpOWE);

模块接口

信号名	方向	功能描述
Op[5:0]	I	6 位 opcode 段
Func[5:0]	I	6 位 func 段
ALUCtrl[3:0]	O	ALU 控制信号
RegDst[1:0]	O	写地址控制 选择 RT,RD
ALUASrc	O	ALU 第一操作数选择控制

ALUBSrc	0	ALU 第二操作数选择控制
RegWrite	0	GRF 写入控制
MemRead	0	DM 读信号
MemWrite	0	DM 写信号
MemToReg[1:0]	0	GRF 写入数据的选择信号
ExtOp	0	高位扩展方式选择信号
If_beq	0	判断是否为 beq 指令的信号
If_bne	0	判断是否为 bne 指令的信号
If_bgez	0	判断是否为 bgez 指令的信号
If_blez	0	判断是否为 blez 指令的信号
If_bgtz	0	判断是否为 bgtz 指令的信号
If_bltz	0	判断是否为 bltz 指令的信号
If_j	0	判断是不是 jal/j 指令 是则为 1
PC_sel[1:0]	0	PC 选择信号
If_sh	0	判断是否为 sh 指令的信号
If_sb	0	判断是否为 sb 指令的信号
dataOp	0	数据扩展方式控制信号
multdivOp	0	乘除法方式控制信号
Start	0	乘除法开始信号
If_mthi	0	判断是否为 if_mthi 指令的信号
If_mtlo	0	判断是否为 if_mtlo 指令的信号
If_mfhi	0	判断是否为 if_mfhi 指令的信号
If_mflo	0	判断是否为 if_mflo 指令的信号

功能定义

序号	功能名称	功能描述
1	产生控制信号	产生控制信号

9. EX. v

文件	模块接口定义
EX. v	module EX(input clk, input reset, input en, input over, input [31:0] IR_D, input [31:0] PC_D, input [31:0] PC4_D, input [31:0] PC8_D, input [31:0] RF_RD1, input [31:0] RF_RD2, input [31:0] EXT, input Zero,

	input more, input less, output reg[31:0] IR_E, output reg[31:0] PC_E, output reg[31:0] PC4_E, output reg[31:0] PC8_E, output reg[31:0] RS_E, output reg[31:0] RT_E, output reg[31:0] EXT_E, output [6:2] excode_E, output reg Zero_E, output reg more_E, output reg less_E);

功能定义

序号	功能名称	功能描述
1	ID/EX 流水线寄存器	保存 PC,IR 等信号的值

10.alu.v

文件	模块接口定义
alu.v	module alu(input [31:0] A, input [31:0] B, input [3:0] ALUCtrl, output reg[31:0] Result);

模块接口

信号名	方向	功能描述
A[31:0]	I	32 位输入数据 1
B[31:0]	I	32 位输入数据 2
ALUCtrl[3:0]	I	控制信号 000：与 001：或 010：加 011：减 100：移位
Result[31:0]	O	32 位数据输出

功能定义

序号	功能名称	功能描述
1	与	A&B
2	或	A|B
3	加	A+B
4	减	A-B
5	异或	A^B
6	或非	~(A|B)
7	逻辑左	B << A[4:0]
8	逻辑右	B >> A[4:0]
9	算数右	$signed($signed(B) >>> A[4:0]);
10	符号数小于置一	($signed(A)<$signed(B)) ? 32'b1 : 32'b0;
11	无符号数小于置一	(A<B) ? 32'b1 : 32'b0;

11. Mult_Div.v

文件	模块接口定义
Mult_Div.v	module Mult_Div(input clk, input reset, input [31:0] A, input [31:0] B, input [1:0] op, input start, input if_mthi, input if_mtlo, output reg Busy, output [31:0] High, output [31:0] Low);

模块接口

信号名	方向	功能描述
Clk	I	时钟信号
Reset	I	复位信号
A	I	输入 A
B	I	输入 B
Op	I	运算方式选择
Start	I	开始信号
If_mthi	I	判断是不是 mthi
If_mtlo	I	判断是不是 mtlo
Busy	O	忙碌信号
High	O	High 寄存器
Low	O	Low 寄存器

功能定义

序号	功能名称	功能描述
1	无符号乘	无符号乘
2	符号乘	符号乘
3	无符号除	无符号除
4	符号除	符号除

12.MEM.v

文件	模块接口定义
MEM.v	module MEM(input clk, input reset, input en, input [31:0] IR_E, input [31:0] PC_E, input [31:0] PC4_E, input [31:0] PC8_E, input [31:0] ALU, input [31:0] Mult_Div, input [31:0] RT_E, output reg[31:0] IR_M, output reg[31:0] PC_M, output reg[31:0] PC4_M, output reg[31:0] PC8_M, output reg[31:0] AO_M, output reg[31:0] MDO_M, output reg[31:0] RT_M);

功能定义

序号	功能名称	功能描述
1	EX/MEM 流水线寄存器	保存 PC,IR 等信号的值

13.dm.v

文件	模块接口定义
dm.v	module dm(input clk, input reset, input MemWrite, input MemRead,

	input [31:0] MemAddr, input [31:0] WD, input [31:0] PC, output [31:0] RD);

模块接口

信号名	方向	功能描述
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
MemWrite	I	读写控制信号 1：写操作
MemRead	I	读写控制信号 1：读操作
MemAddr[31:0]	I	操作寄存器地址
WD[31:0]	I	输入（写入内存）的 32 位数据
PC[31:0]	I	当前 PC
RD[31:0]	O	32 位数据输出

功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，所有数据被设置为 0x00000000
2	读	根据输入的寄存器地址读出数据
3	写	根据输入的地址，把输入的数据写入

14.WB.v

模块接口

文件	模块接口定义
WB.v	module WB(input clk, input reset, input en, input [31:0] IR_M, input [31:0] PC_M, input [31:0] PC4_M, input [31:0] PC8_M, input [31:0] AO_M, input [31:0] MDO_M, input [31:0] DM,

	output reg[31:0] IR_W, output reg[31:0] PC_W, output reg[31:0] PC4_W, output reg[31:0] PC8_W, output reg[31:0] AO_W, output reg[31:0] MDO_W, output reg[31:0] DR_W);

功能定义

序号	功能名称	功能描述
1	MEM/WB 流水线寄存器	保存 PC,IR 等信号的值

15. DataExt.v

文件	模块接口定义
DataExt.v	module DataExt(input [31:0] Din, input [2:0] dataOp, input [1:0] Addr, output reg[31:0] Dout);

模块接口

信号名	方向	功能描述
Din[31:0]	I	32 位输入
dataOp[2:0]	I	扩展方式控制信号
Addr[1:0]	I	地址信号
Dout[31:0]	O	扩展结果

功能定义

序号	功能名称	功能描述
1	扩展	扩展
2	无扩展	无扩展
3	无符号字节数据扩展	无符号字节数据扩展
4	符号字节数据扩展	符号字节数据扩展
5	无符号半字数据扩展	无符号半字数据扩展
6	符号半字数据扩展	符号半字数据扩展

16. mux.v

模块接口

文件	模块接口定义
mux.v	module mux(input [31:0] EXT_E, input [31:0] IR_E, input [31:0] IR_W, input [31:0] DR_Wnew, input [31:0] AO_W, input [31:0] MDO_W, input [31:0] PC8_W, input [31:0] CP0_W, input [31:0] MFRSE, input [31:0] MFRTE, input [31:0] High, input [31:0] Low, input ALUasel, input ALUbsel, input if_mfhi, input if_mflo, input [1:0] RegDst, input [2:0] MemtoReg, output reg[31:0] ALU_A, output reg[31:0] ALU_B, output reg[4:0] MUX_A3, output reg[31:0] MUX_WD, output reg[31:0] MD_out);

功能定义

序号	功能名称	功能描述
1	多路选择器	各级多路选择器 ALU_A,ALU_B,MUX_WD,MUX_A3

17.forward_mux.v

模块接口

文件	模块接口定义
Forward_mux.v	module forward_mux(input [31:0] RS_E, input [31:0] RT_E, input [31:0] RT_M, input [31:0] WD,

	input [31:0] AO_M, input [31:0] MDO_M, input [31:0] CP0_M, input [31:0] MD_out, input [31:0] PC8_E, input [31:0] PC8_M, input [31:0] PC8_W, input [31:0] RF_RD1, input [31:0] RF_RD2, input [31:0] EPCout, input [2:0] ForwardRSD, input [2:0] ForwardRTD, input [2:0] ForwardRSE, input [2:0] ForwardRTE, input [2:0] ForwardRTM, input [2:0] ForwardERET, output reg[31:0] MFRSD, output reg[31:0] MFRTD, output reg[31:0] MFRSE, output reg[31:0] MFRTE, output reg[31:0] MFRTM, output reg[31:0] MFERET);

功能定义

序号	功能名称	功能描述
1	各级转发 MUX	转发信号的选择 MFRSD,MFRTD,MFRSE,MFRTE,MFRTM

18. hazardUnit.v

模块接口

文件	模块接口定义
hazardUnit.v	module hazardUnit(input [31:0] IR_D, input [31:0] IR_E, input [31:0] IR_M, input [31:0] IR_W, input Busy, input start, output IR_D_en, output IR_E_clr, output PC_en, output [2:0]ForwardRSD,

	output [2:0]ForwardRTD, output [2:0]ForwardRSE, output [2:0]ForwardRTE, output [2:0]ForwardRTM, output [2:0]ForwardERET);

功能定义

序号	功能名称	功能描述
1	冒险控制单元	产生转发和暂停的控制信号

19. ExcCode.v

模块接口

文件	模块接口定义
ExcCode.v	module ExcCode(input clk, input reset, input en, input [4:0] ExcCodeIn, output reg[4:0] ExcCodeOut);

功能定义

序号	功能名称	功能描述
1	异常代码传递	异常代码传递

20. CP0.v

模块接口

文件	模块接口定义
CP0.v	module CP0(input[4:0] A1,//读 CP0 寄存器编号 执行 MFC0 指令时产生 input[4:0] A2,//写 CP0 寄存器编号 执行 MTC0 指令时产生 input[31:0] DIn,//CP0 寄存器的写入数据 执行 MTC0 指令时产生 数据来自 GPR input[31:0] PC,//中断/异常时的 PC input[31:0] IR_M,//指令

	input Zero, input more, input less, input if_bd, input[6:2] ExcCode,//中断/异常的类型 input[5:0] HWInt,//6 个设备中断 input We,//CP0 写使能 执行 MTC0 指令时产生 input EXLSet,//用于置位 SR 的 EXL(EXL 为 1) 流水线在 M 阶段产生 input EXLClr,//用于清除 SR 的 EXL(EXL 为 0) 执行 ERET 指令时产生 input clk, input reset, output Interrupt,//中断和异常 是 HWInt/IM/EXL/IE 的函数 output[31:0] EPC,//EPC 寄存器输出至 NPC output[31:0] DOut //CP0 寄存器的输出数据 执行 MFC0 指令时产生，输出数据至 GRF);

功能定义

序号	功能名称	功能描述
1	CP0	CP0

21.CPU.v

模块接口

文件	模块接口定义
CPU.v	module CPU(input clk, input reset, input [7:2] HWInt, input [31:0] PrRD, input [6:2] ExcCode, output [31:0] PrAddr, output [31:0] PrWD, output PrWe);

功能定义

序号	功能名称	功能描述
1	CPU	CPU

22.Bridge.v

模块接口

文件	模块接口定义
Bridge.v	module Bridge(input [31:0] PrAddr, input PrWE, output [31:0] PrRD, input [31:0] PrWD, output [31:0] DEV_Addr, output [31:0] DEV_WD, input [31:0] DEV0_RD, input [31:0] DEV1_RD, output WeDEV0, output WeDEV1);

功能定义

序号	功能名称	功能描述
1	Bridge	Bridge

23.DEV0.v

模块接口

文件	模块接口定义
DEV0.v	module DEV0(input clk, input reset, input [31:0] Addr, input WE, input [31:0] DataIn, output [31:0] DataOut, output IRQ);

功能定义

序号	功能名称	功能描述
1	DEV0	计时器

24.DEV1.v

模块接口

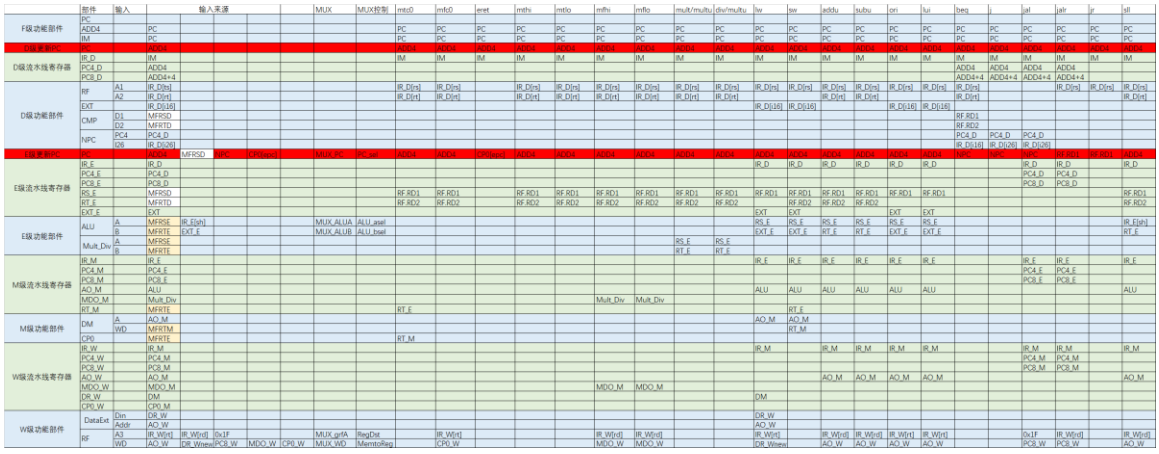
文件	模块接口定义
DEV1.v	module DEV1(input clk, input reset, input [31:0] Addr, input WE, input [31:0] DataIn, output [31:0] DataOut, output IRQ);

功能定义

序号	功能名称	功能描述
1	DEV1	计时器

三．控制器设计

数据通路如下



由此可见需要以下几个 MUX 多路选择器

1.GRF 的 WA 端选择 Rd,Rt 需要一个 MUX，控制信号 RegDst[1:0]

2.GRF 的 WD 输入端，有三种选择：RF.RD2，ALU 的输出，lui 指令直接对 imm16 后边补 16 位 0，需要 2 选 4MUX,选择信号 MemToReg[1:0]

3.扩展方式的选择（符号扩展，0 扩展）选择信号 EXTOp[1:0]

4. ALU 的 A 端两种选择，RF.RD1 或 IR_E[sh]的输出，选择信号 ALUASrc

5.ALU 的 B 端两种选择，RF.RD2 或 EXT 的输出，选择信号 ALUBSrc

6.j/jal 指令 跳转地址的选择 if_j

7.PC 的选择信号 PCsel[1:0]

8.beq 类指令 跳转地址的选择 Branch

除了上述 Branch, ALUASrc, ALUBSrc, EXTOp[1:0], MemToReg[1:0], RegDst[1:0] ,if_j, PC_sel[1:0] 还有三个读写控制信号，RegWrite 是 GRF 写入信号，

MemRead, MemWrite 是 DM 读写信号，ALUCtrl[2:0]是 ALU 控制信号，所以控制器 Controller 需要设计这 12 个控制信号。

信号名	方向	功能描述
Op[5:0]	I	6 位 opcode 段
Func[5:0]	I	6 位 func 段
ALUCtrl[3:0]	O	ALU 控制信号
RegDst[1:0]	O	写地址控制 选择 RT, RD
ALUASrc	O	ALU 第一操作数选择控制
ALUBSrc	O	ALU 第二操作数选择控制
RegWrite	O	GRF 写入控制
MemRead	O	DM 读信号
MemWrite	O	DM 写信号
MemToReg[1:0]	O	GRF 写入数据的选择信号
ExtOp	O	高位扩展方式选择信号
If_beq	O	判断是否为 beq 指令的信号
If_bne	O	判断是否为 bne 指令的信号
If_bgez	O	判断是否为 bgez 指令的信号
If_blez	O	判断是否为 blez 指令的信号
If_bgtz	O	判断是否为 bgtz 指令的信号
If_bltz	O	判断是否为 bltz 指令的信号
If_j	O	判断是不是 jal/j 指令 是则为 1
PC_sel[1:0]	O	PC 选择信号
If_sh	O	判断是否为 sh 指令的信号
If_sb	O	判断是否为 sb 指令的信号

dataOp	0	数据扩展方式控制信号
multdivOp	0	乘除法方式控制信号
Start	0	乘除法开始信号
If_mthi	0	判断是否为 if_mthi 指令的信号
If_mtlo	0	判断是否为 if_mtlo 指令的信号
If_mfhi	0	判断是否为 if_mfhi 指令的信号
If_mflo	0	判断是否为 if_mflo 指令的信号

画出如下表格

name	lw	sw	beq	lui	ori	jal	j	addu	subu	jr	sll	Jalr
Op5	1	1	0	0	0	0	0	0	0	0	0	0
Op4	0	0	0	0	0	0	0	0	0	0	0	0
Op3	0	1	0	1	1	0	0	0	0	0	0	0
Op2	0	0	1	1	1	0	0	0	0	0	0	0
Op1	1	1	0	1	0	1	1	0	0	0	0	0
Op0	1	1	0	1	1	1	0	0	0	0	0	0
Func5								1	1	0	0	0
Func4								0	0	0	0	0
Func3								0	0	1	0	1
Func2								0	0	0	0	0
Func1								0	1	0	0	0
Func0								1	1	0	0	1
RegDst[1:0]	00	00	00	00	00	10	00	01	01	01	01	01
ALUASrc	0	0	0	0	0	0	0	0	0	0	1	0
ALUBSrc	1	1	0	1	1	0	0	0	0	0	0	0
RegWrite	1	0	0	1	1	1	0	1	1	1	1	1
MemRead	1	0	0	0	0	0	0	0	0	0	0	0
MemWrite	0	1	0	0	0	0	0	0	0	0	0	0

MemToReg[1:0]	01	00	00	00	00	10	00	00	00	00	00	10
EXTOp[1:0]	00	00	00	10	01	00	00	00	00	00	00	00
If_beq	0	0	1	0	0	0	0	0	0	0	0	0
ALUCtrl[3:0]	0010	0010	0111	0010	0001	0111	0111	0010	0011	0111	0100	0000
If_j	0	0	0	0	0	1	1	0	0	0	0	0
PC_sel[1:0]	00	00	10	00	00	10	10	00	00	01	00	01

分布式译码 实例化四级控制器（译码器）

```
controller
my_controllerD(.op(IR_D[`op]),.func(IR_D[`func]),.rt(IR_D[`rt]),.Ext
Op(EXTop),.if_beq(if_beq),.if_bne(if_bne),.if_blez(if_blez),.if_bgtz
(if_bgtz),.if_bgez(if_bgez),.if_bltz(if_bltz),.if_j(if_j),.PCsel(PC_
sel));

controller
my_controllerE(.op(IR_E[`op]),.func(IR_E[`func]),.rt(IR_D[`rt]),.ALU
Ctrl(ALUCtrl),.ALUASrc(ALUASrc),.ALUBSrc(ALUBSrc),.multdivOp(multdiv
Op),.start(start),.if_mthi(if_mthi),.if_mtlo(if_mtlo),.if_mfhi(if_mf
hi),.if_mflo(if_mflo));

controller
my_controllerM(.op(IR_M[`op]),.func(IR_M[`func]),.rt(IR_D[`rt]),.Mem
Read(MemRead),.MemWrite(MemWrite),.if_sh(if_sh),.if_sb(if_sb));

controller
my_controllerW(.op(IR_W[`op]),.func(IR_W[`func]),.rt(IR_D[`rt]),.Reg
Dst(RegDst),.RegWrite(RegWrite),.MemtoReg(MemtoReg),.dataOp(dataOp));
```

四．冒险处理单元设计

需求时间——供给时间模型。

Tuse

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

Tnew

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/rd	0/rd	0/rt	1/rt	0/31	0/rd	0/rd	0/rt	0/rt	0/31	0/rd

暂停

IF/ID 当前指令			ID/EX			EX/MEM
指令类型	源寄存器	Tuse	Tnew			Tnew
			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

由此可以写出各种控制信号的表达式如下

```
`define cal_r_D (IR_D[`op]==`R&&IR_D[`func]!=`jalr&&IR_D[`func]!=`jr&&IR_D!=0)
`define cal_r_E (IR_E[`op]==`R&&IR_E[`func]!=`jalr&&IR_E[`func]!=`jr&&IR_E!=0)
`define cal_r_M (IR_M[`op]==`R&&IR_M[`func]!=`jalr&&IR_M[`func]!=`jr&&IR_M!=0)
`define cal_r_W (IR_W[`op]==`R&&IR_W[`func]!=`jalr&&IR_W[`func]!=`jr&&IR_W!=0)

`define cal_i_D (IR_D[`op]==`lui||IR_D[`op]==`ori)
`define cal_i_E (IR_E[`op]==`lui||IR_E[`op]==`ori)
`define cal_i_M (IR_M[`op]==`lui||IR_M[`op]==`ori)
`define cal_i_W (IR_W[`op]==`lui||IR_W[`op]==`ori)

`define load_D (IR_D[`op]==`lw)
`define load_E (IR_E[`op]==`lw)
`define load_M (IR_M[`op]==`lw)
`define load_W (IR_W[`op]==`lw)

`define store_D (IR_D[`op]==`sw)
`define store_E (IR_E[`op]==`sw)
`define store_M (IR_M[`op]==`sw)
`define store_W (IR_W[`op]==`sw)

`define beq_D (IR_D[`op]==`beq)
`define beq_E (IR_E[`op]==`beq)
`define beq_M (IR_M[`op]==`beq)
`define beq_W (IR_W[`op]==`beq)
```

```
reg stall=0;
wire stall_b,stall_cal_r,stall_cal_i,stall_load,stall_store,stall_jr,stall_jalr,stall_busy,stall_mfmt;

assign stall_b = (`beq_D & `cal_r_E & (IR_D[`rs]==IR_E[`rd]||IR_D[`rt]==IR_E[`rd]))||
                 (`beq_D & `cal_i_E & (IR_D[`rs]==IR_E[`rt]||IR_D[`rt]==IR_E[`rt]))||
                 (`beq_D & `load_E  & (IR_D[`rs]==IR_E[`rt]||IR_D[`rt]==IR_E[`rt]))||
                 (`beq_D & `load_M  & (IR_D[`rs]==IR_M[`rt]||IR_D[`rt]==IR_M[`rt]));
assign stall_cal_r = (`cal_r_D) && (`load_E) && (IR_D[`rs]==IR_E[`rt]||IR_D[`rt]==IR_E[`rt]);
assign stall_cal_i = (`cal_i_D) && (`load_E) && (IR_D[`rs]==IR_E[`rt]);
assign stall_load =  (`load_D)  && (`load_E) && (IR_D[`rs]==IR_E[`rt]);
assign stall_store = (`store_D) && (`load_E) && (IR_D[`rs]==IR_E[`rt]);
assign stall_jr = (`jr_D & (`cal_r_E) & (IR_D[`rs]==IR_E[`rd]))||
                  (`jr_D & (`cal_i_E) & (IR_D[`rs]==IR_E[`rt]))||
                  (`jr_D & (`load_E)  & (IR_D[`rs]==IR_E[`rt]))||
                  (`jr_D & (`load_M)  & (IR_D[`rs]==IR_M[`rt]));
assign stall_jalr =  (`jalr_D & (`cal_r_E) & (IR_D[`rs]==IR_E[`rd]))||
                     (`jalr_D & (`cal_i_E) & (IR_D[`rs]==IR_E[`rt]))||
                     (`jalr_D & (`load_E)  & (IR_D[`rs]==IR_E[`rt]))||
                     (`jalr_D & (`load_M)  & (IR_D[`rs]==IR_M[`rt]));
assign stall_busy = (IR_D[`op]==`R&(IR_D[`func]==`mult||IR_D[`func]==`multu||IR_D[`func]==`div||IR_D[`func]==`divu||
IR_D[`func]==`mflo||IR_D[`func]==`mfhi||IR_D[`func]==`mthi||IR_D[`func]==`mtlo)) & Busy;
assign stall_mfmt = (IR_D[`op]==`R&(IR_D[`func]==`mult||IR_D[`func]==`multu||IR_D[`func]==`div||IR_D[`func]==`divu||
   IR_D[`func]==`mflo||IR_D[`func]==`mfhi||IR_D[`func]==`mthi||IR_D[`func]==`mtlo))
   &(IR_E[`op]==`R&(IR_E[`func]==`mult||IR_E[`func]==`multu||IR_E[`func]==`div||IR_E[`func]==`divu));
always@(*) begin
   stall <= stall_b||stall_cal_r||stall_cal_i||stall_load||stall_store||stall_jr||stall_jalr||stall_busy||stall_mfmt;
end
```

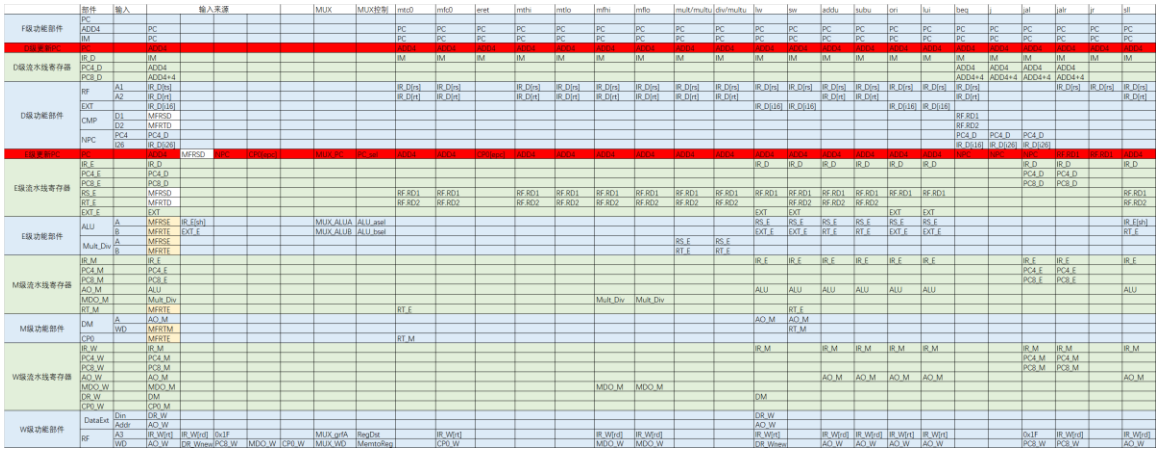
转发

						ID/EX			EX/MEM					MEM/WB					
						Tnew			Tnew					Tnew					
						jal	jalr	mflo mfhi	cal_r	cal_i	jal	jalr	mflo mfhi	cal_r	cal_i	load	mflo mfhi	jal	jalr
流水级	源寄存器	涉及指令	MUX	控制信号	输入0	0/31	0/rd	0/rd	0/rd	0/rt	0/31	0/rd	0/rd	0/rd	0/rt	0/rt	0/rd	0/31	0/rd
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
					0	3	3	6	1	1	4	4	7	2	2	2	2	5	5

由此可以写出各种控制信号的表达式如下

```
    assign ForwardRSD = (`RSD & `jal_E   & IR_D[`rs]==31 & IR_D[`rs]!=0) ? 3 :
                        (`RSD & `jalr_E  & IR_D[`rs]==IR_E[`rd] & IR_D[`rs]!=0) ? 3 :
                        (`RSD & `mf_E    & IR_D[`rs]==IR_E[`rd] & IR_D[`rs]!=0) ? 6 :
                        (`RSD & `cal_r_M & IR_D[`rs]==IR_M[`rd] & IR_D[`rs]!=0) ? 1 :
                        (`RSD & `cal_i_M & IR_D[`rs]==IR_M[`rt] & IR_D[`rs]!=0) ? 1 :
                        (`RSD & `jal_M   & IR_D[`rs]==31        & IR_D[`rs]!=0) ? 4 :
                        (`RSD & `jalr_M  & IR_D[`rs]==IR_M[`rd] & IR_D[`rs]!=0) ? 4 :
                        (`RSD & `mf_M    & IR_D[`rs]==IR_M[`rd] & IR_D[`rs]!=0) ? 7 :
                        (`RSD & `cal_r_W & IR_D[`rs]==IR_W[`rd] & IR_D[`rs]!=0) ? 2 :
                        (`RSD & `cal_i_W & IR_D[`rs]==IR_W[`rt] & IR_D[`rs]!=0) ? 2 :
                        (`RSD & `load_W  & IR_D[`rs]==IR_W[`rt] & IR_D[`rs]!=0) ? 2 :
                        (`RSD & `mf_W    & IR_D[`rs]==IR_W[`rd] & IR_D[`rs]!=0) ? 2 :
                        (`RSD & `jal_W   & IR_D[`rs]==31        & IR_D[`rs]!=0) ? 5 :
                        (`RSD & `jalr_W  & IR_D[`rs]==IR_W[`rd] & IR_D[`rs]!=0) ? 5 : 0 ;


    assign ForwardRTD = (`RTD & `jal_E   & IR_D[`rt]==31 & IR_D[`rt]!=0) ? 3 :
                        (`RTD & `jalr_E  & IR_D[`rt]==IR_E[`rd] & IR_D[`rt]!=0) ? 3 :
                        (`RTD & `mf_E    & IR_D[`rt]==IR_E[`rd] & IR_D[`rt]!=0) ? 6 :
                        (`RTD & `cal_r_M & IR_D[`rt]==IR_M[`rd] & IR_D[`rt]!=0) ? 1 :
                        (`RTD & `cal_i_M & IR_D[`rt]==IR_M[`rt] & IR_D[`rt]!=0) ? 1 :
                        (`RTD & `jal_M   & IR_D[`rt]==31        & IR_D[`rt]!=0) ? 4 :
                        (`RTD & `jalr_M  & IR_D[`rt]==IR_M[`rd] & IR_D[`rt]!=0) ? 4 :
                        (`RTD & `mf_M    & IR_D[`rt]==IR_M[`rd] & IR_D[`rt]!=0) ? 7 :
                        (`RTD & `cal_r_W & IR_D[`rt]==IR_W[`rd] & IR_D[`rt]!=0) ? 2 :
                        (`RTD & `cal_i_W & IR_D[`rt]==IR_W[`rt] & IR_D[`rt]!=0) ? 2 :
                        (`RTD & `load_W  & IR_D[`rt]==IR_W[`rt] & IR_D[`rt]!=0) ? 2 :
                        (`RTD & `mf_W    & IR_D[`rt]==IR_W[`rd] & IR_D[`rt]!=0) ? 2 :
                        (`RTD & `jal_W   & IR_D[`rt]==31        & IR_D[`rt]!=0) ? 5 :
                        (`RTD & `jalr_W  & IR_D[`rt]==IR_W[`rd] & IR_D[`rt]!=0) ? 5 : 0;


  assign ForwardRSE = (`RSE & `cal_r_M & (IR_E[`rs]==IR_M[`rd]) & IR_E[`rs]!=0) ? 1 :
                      (`RSE & `cal_i_M & (IR_E[`rs]==IR_M[`rt]) & IR_E[`rs]!=0) ? 1 :
                      (`RSE & `jal_M   & (IR_E[`rs]==31)        & IR_E[`rs]!=0) ? 4 :
                      (`RSE & `jalr_M  & (IR_E[`rs]==IR_M[`rd]) & IR_E[`rs]!=0) ? 4 :
                      (`RSE & `mf_M    & (IR_E[`rs]==IR_M[`rd]) & IR_E[`rs]!=0) ? 7 :
                      (`RSE & `cal_r_W & (IR_E[`rs]==IR_W[`rd]) & IR_E[`rs]!=0) ? 2 :
                      (`RSE & `cal_i_W & (IR_E[`rs]==IR_W[`rt]) & IR_E[`rs]!=0) ? 2 :
                      (`RSE & `load_W  & (IR_E[`rs]==IR_W[`rt]) & IR_E[`rs]!=0) ? 2 :
                      (`RSE & `mf_W    & (IR_E[`rs]==IR_W[`rd]) & IR_E[`rs]!=0) ? 2 :
                      (`RSE & `jal_W   & (IR_E[`rs]==31)        & IR_E[`rs]!=0) ? 5 :
                      (`RSE & `jalr_W  & (IR_E[`rs]==IR_W[`rd]) & IR_E[`rs]!=0) ? 5 : 0;

  assign ForwardRTE = (`RTE & `cal_r_M & (IR_E[`rt]==IR_M[`rd]) & IR_E[`rt]!=0) ? 1 :
                      (`RTE & `cal_i_M & (IR_E[`rt]==IR_M[`rt]) & IR_E[`rt]!=0) ? 1 :
                      (`RTE & `jal_M   & (IR_E[`rt]==31)        & IR_E[`rt]!=0) ? 4 :
                      (`RTE & `jalr_M  & (IR_E[`rt]==IR_M[`rd]) & IR_E[`rt]!=0) ? 4 :
                      (`RTE & `mf_M    & (IR_E[`rt]==IR_M[`rd]) & IR_E[`rt]!=0) ? 7 :
                      (`RTE & `cal_r_W & (IR_E[`rt]==IR_W[`rd]) & IR_E[`rt]!=0) ? 2 :
                      (`RTE & `cal_i_W & (IR_E[`rt]==IR_W[`rt]) & IR_E[`rt]!=0) ? 2 :
                      (`RTE & `load_W  & (IR_E[`rt]==IR_W[`rt]) & IR_E[`rt]!=0) ? 2 :
                      (`RTE & `mf_W    & (IR_E[`rt]==IR_W[`rd]) & IR_E[`rt]!=0) ? 2 :
                      (`RTE & `jal_W   & (IR_E[`rt]==31)        & IR_E[`rt]!=0) ? 5 :
                      (`RTE & `jalr_W  & (IR_E[`rt]==IR_W[`rd]) & IR_E[`rt]!=0) ? 5 : 0;


 assign ForwardRTM = (`RTM & `cal_r_W & (IR_M[`rt]==IR_W[`rd]) & IR_M[`rt]!=0) ? 2 :
                     (`RTM & `cal_i_W & (IR_M[`rt]==IR_W[`rt]) & IR_M[`rt]!=0) ? 2 :
                     (`RTM & `load_W  & (IR_M[`rt]==IR_W[`rt]) & IR_M[`rt]!=0) ? 2 :
                     (`RTM & `mf_W    & (IR_M[`rt]==IR_W[`rd]) & IR_M[`rt]!=0) ? 2 :
                     (`RTM & `jal_W   & (IR_M[`rt]==31) & IR_M[`rt]!=0) ? 5 :
                     (`RTM & `jalr_W  & (IR_M[`rt]==IR_W[`rd]) & IR_M[`rt]!=0) ? 5 : 0;


  assign ForwardERET = ((IR_D==32'h42000018)&&(IR_M[`op]==6'b010000&&IR_M[`rs]==5'b00100&&IR_M[`rd]==14)) ? 1 : 0;
```

更新后的数据通路 加入了转发 MUX



Forward_mux 代码如下

```
always@(*) begin
        case(ForwardRSD)
            3'b000 : MFRSD <= RF_RD1;
            3'b001 : MFRSD <= AO_M;
            3'b010 : MFRSD <= WD;
            3'b011 : MFRSD <= PC8_E;
            3'b100 : MFRSD <= PC8_M;
            3'b101 : MFRSD <= CP0_M;
            3'b110 : MFRSD <= MD_out;
            3'b111 : MFRSD <= MDO_M;
            default : MFRSD <= 0;
        endcase

        case(ForwardRTD)
            0: MFRTD <= RF_RD2;
            1: MFRTD <= AO_M;
            2: MFRTD <= WD;
            3: MFRTD <= PC8_E;
```

```
        4: MFRTD <= PC8_M;
        5: MFRTD <= CP0_M;
        6: MFRTD <= MD_out;
        7: MFRTD <= MDO_M;
        default: MFRTD <= 0;
    endcase

    case(ForwardRSE)
        0:MFRSE <= RS_E;
        1:MFRSE <= AO_M;
        2:MFRSE <= WD;
        3:MFRSE <= 0;
        4:MFRSE <= PC8_M;
        5:MFRSE <= CP0_M;
        6:MFRSE <= 0;
        7:MFRSE <= MDO_M;
        default:MFRSE <= 0;
    endcase

    case(ForwardRTE)
        0:MFRTE <= RT_E;
        1:MFRTE <= AO_M;
        2:MFRTE <= WD;
        3:MFRTE <= 0;
        4:MFRTE <= PC8_M;
        5:MFRTE <= CP0_M;
        6:MFRTE <= 0;
```

```
            7:MFRTE <= MDO_M;
            default:MFRTE <= 0;
        endcase

        case(ForwardRTM)
            0:MFRTM <= RT_M;
            1:MFRTM <= 0;
            2:MFRTM <= WD;
            3:MFRTM <= 0;
            4:MFRTM <= 0;
            5:MFRTM <= 0;
            6:MFRTM <= 0;
            7:MFRTM <= 0;
            default:MFRTM <= 0;
        endcase

        case(ForwardERET)
            0:MFERET <= EPCout;
            1:MFERET <= MFRTM;
            default:MFERET <= 0;
        endcase
    end
```

五．中断异常设计

ExcCode.v

模块接口

文件	模块接口定义
ExcCode.v	module ExcCode(input clk, input reset, input en, input [4:0] ExcCodeIn, output reg[4:0] ExcCodeOut);

ExcCode 用来传递每一级的异常编码

本project需要支持的异常:

ExcCode	助记符	描述
0	Int	中断
4	AdEL	取数或取指时地址错误
5	AdES	存数时地址错误
10	RI	不认识的（或者非法的）指令码
12	Ov	自陷形式的整数算术指令（例如add）导致的溢出

CP0 行为规范

1. 本实验要求支持 SR、Cause、EPC、PRId 四个 CP0 寄存器。
2. CP0 的位置不做明确要求，需要自行设计
3. 模块规格可以参考课件上的设计，不作硬性要求。
4. SR 寄存器行为需要与课件上行为保持一致。
5. Cause 寄存器在课件的基础上需要增加 BD 位，和 ExcCode 位（因此需要对顶层模块做相应的修改）。行为规范参考 《See MIPS Run Linux》。
6. EPC 寄存器架构与课件上保持一致。异常发生时 EPC 存入的值参考《See MIPS Run Linux》，中断发生时 EPC 存入的值可以自行设计。
7. PRId 寄存器的值不作要求。

CP0.v

模块接口

文件	模块接口定义
CP0.v	module CP0(input[4:0] A1,//读 CP0 寄存器编号 执行 MFC0 指令时产生 input[4:0] A2,//写 CP0 寄存器编号 执行 MTC0 指令时产生 input[31:0] DIn,//CP0 寄存器的写入数据 执行 MTC0 指令时产生 数据来自 GPR input[31:0] PC,//中断/异常时的 PC input[31:0] IR_M,//指令 input Zero, input more, input less, input if_bd, input[6:2] ExcCode,//中断/异常的类型 input[5:0] HWInt,//6 个设备中断 input We,//CP0 写使能 执行 MTC0 指令时产生 input EXLSet,//用于置位 SR 的 EXL(EXL 为 1) 流水线在 M 阶段产生 input EXLClr,//用于清除 SR 的 EXL(EXL 为 0) 执行 ERET 指令时产生 input clk, input reset, output Interrupt,//中断和异常 是 HWInt/IM/EXL/IE 的函数 output[31:0] EPC,//EPC 寄存器输出至 NPC output[31:0] DOut //CP0 寄存器的输出数据 执行 MFC0 指令时产生，输出数据至 GRF);

六. 主程序

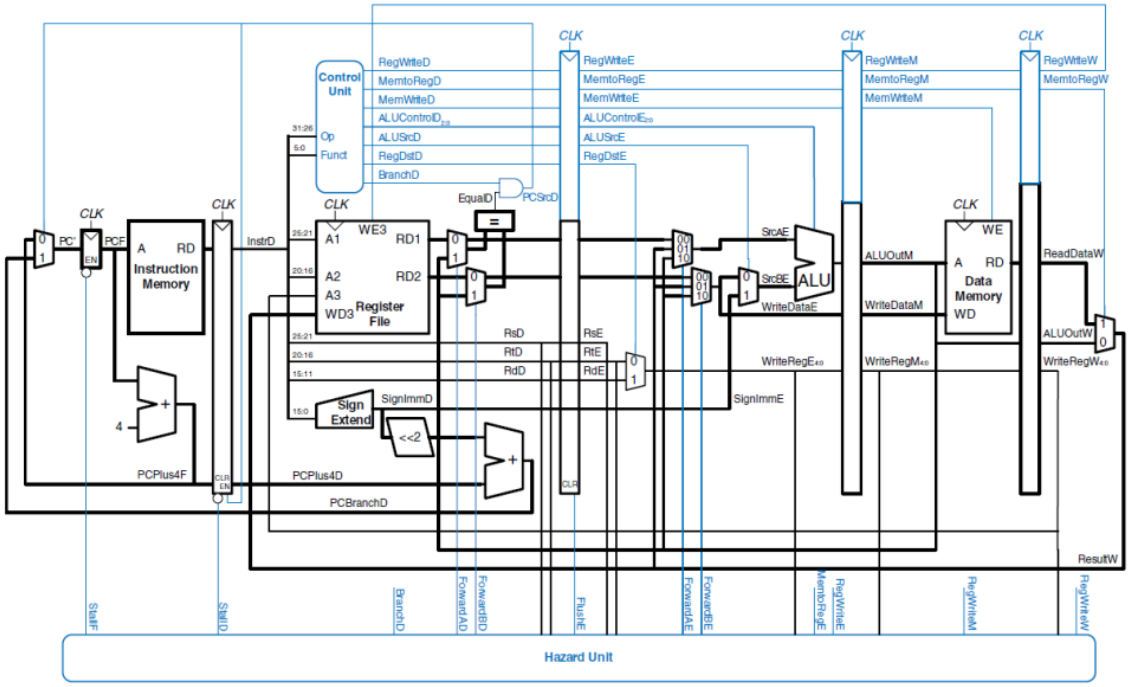


Figure 7.58 Pipelined processor with full hazard handling

数据通路主要采用如上架构　区别是分布式译码

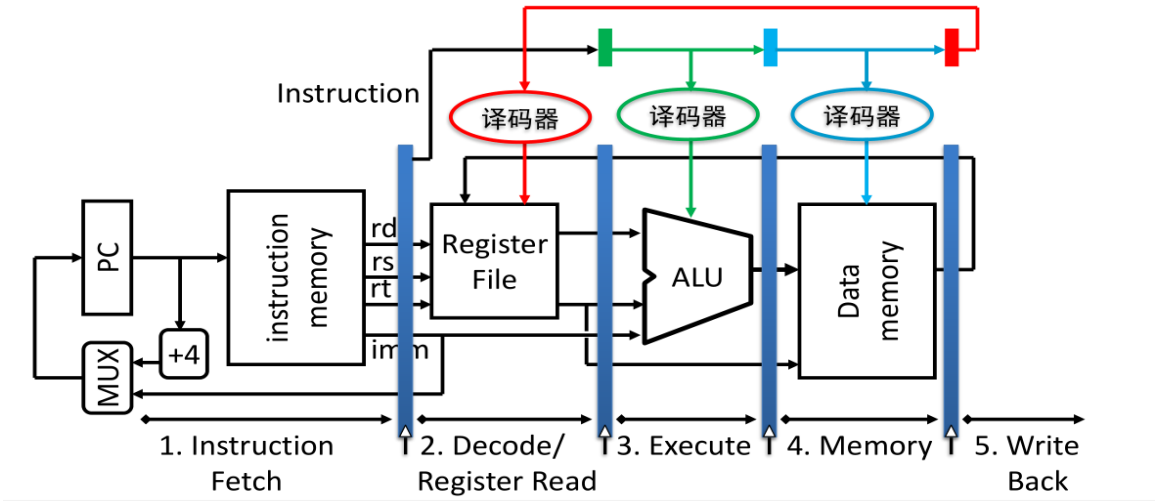
1. 流水线的设计以追求性能为第一目标，因此必须尽最大可能**支持转发**以解决数据冒险。这一点在本 project 的最终成绩中所占比重较大，课上测试时会通过测试程序所跑的**总周期数**进行判定，望大家慎重对待。

2. 对于 b 类和 j 类指令， 流水线设计必须**支持延迟槽**，因此设计需要注意使用 **PC+8**。

3. 为了解决数据冒险而设计的转发数据来源必须是**某级流水线寄存器，不允许**对功能部件的输出直接进行转发。

4.分布式译码

- 分布式式控制器
  - 控制器分布在多个流水线阶段
  - 每级控制器只产生该级功能部件相关的译码信号
  - 流水指令



需要以下几个 MUX 多路选择器

1.GRF 的 WA 端选择 Rd,Rt 需要一个 MUX，控制信号 RegDst[1:0]

2.GRF 的 WD 输入端，有三种选择：RF.RD2，ALU 的输出，lui 指令直接对 imm16 后边补 16 位 0，需要 2 选 4MUX,选择信号 MemToReg[1:0]

3. ALU 的 A 端两种选择，RF.RD1 或 IR_E[sh]的输出，选择信号 ALUASrc

4.ALU 的 B 端两种选择，RF.RD2 或 EXT 的输出，选择信号 ALUBSrc

1.mux.v

模块接口

文件	模块接口定义
mux.v	module mux(input [31:0] EXT_E, input [31:0] IR_E, input [31:0] IR_W, input [31:0] DR_Wnew,

	input [31:0] AO_W, input [31:0] MDO_W, input [31:0] PC8_W, input [31:0] CPO_W, input [31:0] MFRSE, input [31:0] MFRTE, input [31:0] High, input [31:0] Low, input ALUasel, input ALUbsel, input if_mfhi, input if_mflo, input [1:0] RegDst, input [2:0] MemtoReg, output reg[31:0] ALU_A, output reg[31:0] ALU_B, output reg[4:0] MUX_A3, output reg[31:0] MUX_WD, output reg[31:0] MD_out);

2.CPU.v

文件	模块接口定义
CPU.v	module CPU(input clk, input reset, input [7:2] HWInt, input [31:0] PrRD, input [6:2] ExcCode, output [31:0] PrAddr, output [31:0] PrWD, output PrWe);

3.tb

```
module test;

    // Inputs

    reg clk;

    reg reset;

    // Instantiate the Unit Under Test (UUT)

    mips uut (
```

```
        .clk(clk),
        .reset(reset)
    );
    initial begin
        clk = 0;
        reset = 1;
        #12 reset = 0;
    end
    always #10 clk = ~clk;
endmodule
```

4.外设 计时器

	地址或地址范围	备注
数据存储器	0x0000_0000至0x0000_2FFF	
指令存储器	0x0000_3000至0x0000_4FFF	
PC初始值	0x0000_3000	
Exception Handler入口地址	0x0000_4180	
定时器寄存器地址	0x0000_7F00至0x0000_7F0B 0x0000_7F10至0x0000_7F1B	定时器0的3个寄存器 定时器1的3个寄存器

功能描述及内部结构

TC 的内部基本结构如图 1-1 所示。TC 由控制寄存器、初值寄存器、32 位计数器及中断产生逻辑构成。

1) 控制寄存器决定该计数起停控制等。

2) 初值寄存器为 32 位计数器提供初始值。

3) 根据不同的计数模式，在计数为 0 后，计数器或者自动装填初值并重新倒计数，或者保持在 0 值直至计数器使能再次被设置为 1。

4) 使用 store 类指令修改 TC 寄存器值的优先级高于 TC 自修改的优先级。

5) 当计数器计数时，若计数器使能被 store 类指令修改为 0 则停止计数。

6) 当计数器工作在模式 0 并且在中断允许的前提下，当计数器计数值为 0 时，中断产生逻辑产生中断请求(IRQ 为 1)。

计数模式

2.1. 模式 0

当计数器倒计数为 0 后，计数器停止计数，此时控制寄存器中的使能 Enable 自动变为 0。当使能 Enable 被设置为 1 后，初值寄存器值再次被加载至计数器，计数器重新启动倒计数。模式 0 通常用于产生定时中断。例如，为操作系统的时间片调度机制提供定时。模式 0 下的中断信号将持续有效，直至控制寄存器中的中断屏蔽位被设置为 0。

2.2. 模式 1

当计数器倒计数为 0 后，初值寄存器值被自动加载至计数器，计数器继续倒数计数。模式 1 通常用于产生周期性脉冲。例如，可以用模式 1 产生步进电机所需的步进控制信号。不同于模式 0，模式 1 下计数器每次计数循环中只产生一周期的中断信号。

表 3-1 Timer/Counter 寄存器

偏移	寄存器	寄存器描述	R/W	复位值
0h	CTRL	控制寄存器	R/W	0
4h	PRESET	初值寄存器	R/W	0
8h	COUNT	计数值寄存器	R	0

3.1. 控制寄存器(CTRL)

当读取 CTRL 寄存器时，未定义位始终为 0；当写入 CTRL 寄存器时，未定义位被忽略。

表 3-2 控制寄存器格式

Bit mnemonic	Bit No.	Description	R/W	Value After Reset
Reserved	31:4	保留	—	0
IM	3	中断屏蔽 0：禁止中断 1：允许中断	R/W	0
Mode	2:1	模式选择 00：方式 0 01：方式 1 10：未定义 11：未定义	R/W	00
Enable	0	计数器使能 0：停止计数 1：允许计数	R/W	0

1.在允许计数器计数前，应首先停止计数；然后加载初值寄存器；再允许计数。

2.无论哪种模式，如果不需要产生中断，则应屏蔽中断。

DEV0.v

模块接口

文件	模块接口定义
DEV0.v	module DEV0(

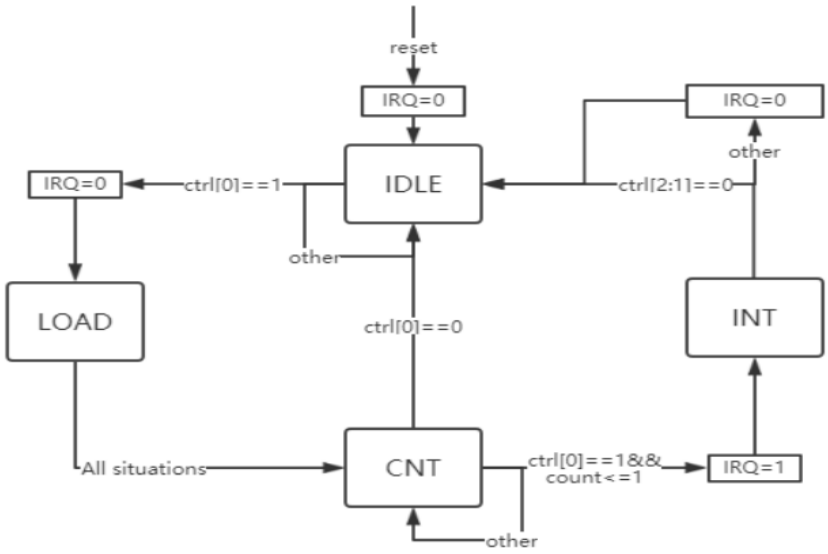
	input clk, input reset, input [31:0] Addr, input WE, input [31:0] DataIn, output [31:0] DataOut, output IRQ);

DEV1.v

模块接口

文件	模块接口定义
DEV1.v	module DEV1(input clk, input reset, input [31:0] Addr, input WE, input [31:0] DataIn, output [31:0] DataOut, output IRQ);

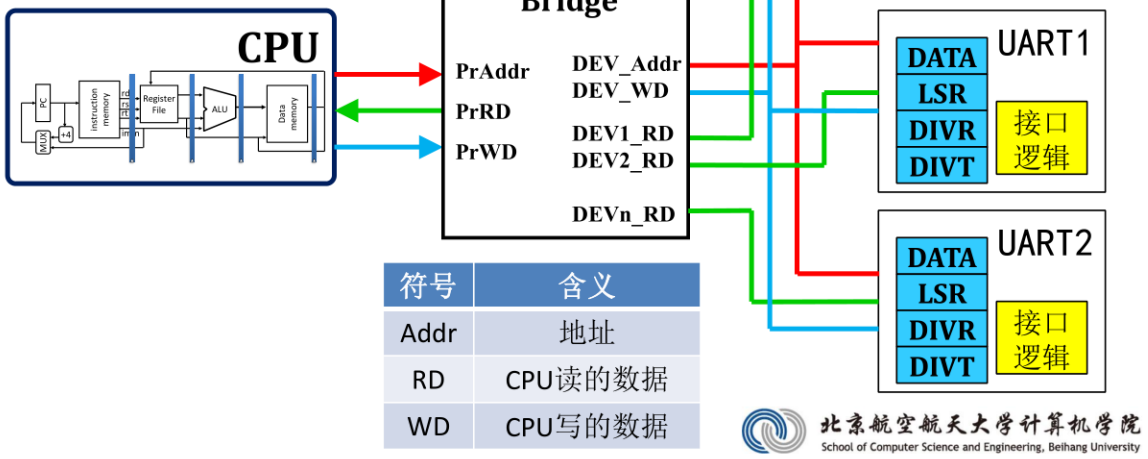
6. 计时器的状态转移图如下



5.Bridge

增加新模块：Bridge

- Bridge：类似与网络switch
  - CPU侧：1组接口。设备侧：N组接口
- 1组地址/写数据，N组读数据
  - CPU读：数据汇聚
  - CPU写：数据派发



符号	含义
Addr	地址
RD	CPU读的数据
WD	CPU写的数据

北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

Bridge.v

模块接口

文件	模块接口定义
Bridge.v	module Bridge(input [31:0] PrAddr, input PrWE, output [31:0] PrRD, input [31:0] PrWD, output [31:0] DEV_Addr, output [31:0] DEV_WD, input [31:0] DEV0_RD, input [31:0] DEV1_RD, output WeDEV0, output WeDEV1);

七．测试程序

命令行导出：java -jar E:\Mars4_5.jar a db mc CompactDataAtZero dump 0x00004180-0x00004ffc HexText E:\code_handler.txt E:\mips4.asm

(1)转发测试

(2)暂停测试

(3)异常中断测试

1.ADEL

```
  (1).ktext 0x4180
    mfc0   $k0, $14
    addu   $k0, $k0, 4
    mtc0   $k0, $14
    eret
  .text
    ori $28, $0, 0x0000
    ori $29, $0, 0x0000
    lui $8, 0x7000
    lui $9, 0xf000
    lw $9,3($0)
    sub $10, $8,$9
    or  $10, $8, $9
  (2).ktext 0x4180
```

```
  mfc0   $k0, $14
  addu   $k0, $k0, 4
  mtc0   $k0, $14
  eret
.text
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  lui $8, 0x7000
  lui $9, 0xf000
  lh $9,3($0)
  sub $10, $8,$9
  or  $10, $8, $9
```

(3).text

```
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  ori $8, 0x7fffffff
  lui $9, 0x1
  add $10, $8,$9
  lw $a0,0x1000($8)
  or  $10, $8, $9
```

2.ADES

(1).text

```
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
```

```
  ori $8, 0x7f00
  lui $9, 0xf000
  sw $9,3($0)
  sub $10, $8,$9
  or  $10, $8, $9
```

(2).text

```
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  ori $8, 0x7f00
  lui $9, 0xf000
  sh $9,1($0)
  sub $10, $8,$9
  or  $10, $8, $9
```

3.RI

在其他测试的机器码中插入 ffffffff

4.Ov

```
.ktext 0x4180
  mfc0   $k0, $14
  sub $8,$8,$8
  mtc0   $k0, $14
  eret
.text
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
```

```
  lui $8, 0x7fff
  lui $9, 0x7fff
  add $10, $8,$9
  or  $10, $8, $9
.text
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  lui $8, 0x7fff
  lui $9, 0x7fff
  addi $10, $8,0x7fff0000
  or  $10, $8, $9
.text
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  lui $8, 0x7fff
  ori $8, $8, 0xffff
  addi   $10, $8, 1
  ori $a0,$0,100
.text
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  lui $8, 0x7000
  lui $9, 0xf000
  sub $10, $8,$9
```

```
  or  $10, $8, $9
.text
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  ori $8, 0x7f00
  lui $9, 0xf000
  lw $9,0($8)
  sub $10, $8,$9
  or  $10, $8, $9
.text
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  ori $8, 0x7f00
  lui $9, 0xf000
  sh $9,4($8)
  sub $10, $8,$9
  or  $10, $8, $9
```

5.延迟槽

（1）.ktext 0x4180

```
  mfc0 $1,$13
  sub $9,$9,$9
  eret
.text
  ori $28, $0, 0x0000
```

```
  ori $29, $0, 0x0000
  ori $8, 0x7fffffff
  ori $9, 0x1000
  j eee
  add $10, $8,$9
  lw $a0,0x1000($8)
  eee:
  or  $10, $8, $9
(2) .ktext 0x4180
  mfc0 $1,$13
  sub $9,$9,$9
  eret
.text
  ori $8, 0x7fffffff
  ori $9, 0x1000
  ori $t1 0x00007f00
  ori $a0,0x0009
  ori $a3,0xfc01
  beq $9,$8,eee
  add $10,$8,$9
  ori $a1,2
  sw $a1,4($t1)
  eee:
  sw $a0,0($t1)
```

（3）.ktext 0x4180

```
 mfc0 $1,$13
 sub $9,$9,$9
 eret
.text
 ori $8, 0x7fff0000
 ori $9, 0x7fff0000
 ori $a0,0x0009
 ori $a3,0xfc01
 beq $9,$8,eee
 add $10,$8,$9
 ori $a1,2
 sw $a1,4($t1)
 eee:
 sw $a0,0($t1)
```

（4）.ktext 0x4180

```
 mfc0 $1,$13
 sub $8,$8,$8
 sub $9,$9,$9
 eret
.text
 ori $8, 0x7fff0000
 ori $9, 0x7fff0000
 ori $a0,0x0009
```

```
ori $a3,0xfc01
beq $9,$8,eee
add $10,$8,$9
ori $a1,2
sw $a1,4($t1)
eee:
sw $a0,0($t1)
```

（5）

```
.ktext 0x4180
mfc0 $1,$13
sub $9,$9,$9
eret
.text
ori $28, $0, 0x0000
ori $29, $0, 0x0000
ori $8, 0x7fffffff
ori $9, 0x1000
j eee
add $10, $8,$0
lw $a0,0x1000($8)
eee:
or $10, $8, $9
j end
add $10,$8,$9
end:
```

```
  ori $a0,$0,0
(6) .ktext 0x4180
  mfc0 $1,$13
  sub $9,$9,$9
  eret
.text
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  ori $8, 0x7fffffff
  ori $9, 0x1000
  j eee
  add $10, $8,$0
  lw $a0,0x1000($8)
  eee:
  or  $10, $8, $9
  j end
  sh $1,1($0)
  end:
  ori $a0,$0,0
(7) .text
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  ori $8, 0x7fffffff
  ori $9, 0x1000
```

```
  j eee
  add $10, $8,$0
  lw $a0,0x1000($8)
  eee:
  or  $10, $8, $9
  j end
  sw $1,1($0)
  end:
  ori $a0,$0,0
(8) .text
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  ori $8, 0x7fffffff
  ori $9, 0x1000
  j eee
  add $10, $8,$0
  lw $a0,0x1000($8)
  eee:
  or  $10, $8, $9
  j end
  lh $1,1($0)
  end:
  ori $a0,$0,0
(9) .text
```

```
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  ori $8, 0x7fffffff
  ori $9, 0x1000
  j eee
  add $10, $8,$0
  lw $a0,0x1000($8)
  eee:
  or  $10, $8, $9
  j end
  lw $1,1($0)
  end:
  ori $a0,$0,0
(10).text
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  ori $8, 0x7fffffff
  ori $9, 0x1000
  j eee
  add $10, $8,$0
  lw $a0,0x1000($8)
  eee:
  or  $10, $8, $9
  j end
```

```
    lhu $1,1($0)
    end:
    ori $a0,$0,0
```

6.中断

```
  （1）.ktext 0x4180
    mfc0 $1,$13
    sub $9,$9,$9
    mtc0 $0,$12
    eret
  .text
    ori $8, 0x7fffffff
    ori $9, 0x1000
    ori $t1 0x00007f00
    ori $a0,0x0009
    ori $a3,0xfc01
    ori $a1,2
    sw $a1,4($t1)
    sw $a0,0($t1)
    mtc0 $a3,$12
    or  $10, $8, $9
    ori $28, $0, 0x0000
    ori $29, $0, 0x0000
    ori $28, $0, 0x0010
```

```
  ori $29, $0, 0x0111
  nop
  nop
  nop
(2) .ktext 0x4180
  mfc0 $1,$13
  sub $9,$9,$9
  mtc0 $0,$12
  eret
.text
  ori $8, 0x7fffffff
  ori $9, 0x1000
  ori $t1 0x00007f00
  ori $a0,0x0009
  ori $a3,0xfc01
  ori $a1,2
  sw $a1,4($t1)
  sw $a0,0($t1)
  add $11,$8,$9
  mtc0 $a3,$12
  or  $10, $8, $9
  ori $28, $0, 0x0000
  ori $29, $0, 0x0000
  ori $28, $0, 0x0010
```

```
  ori $29, $0, 0x0111
  nop
  nop
  nop
(3) .ktext 0x4180
  mfc0 $1,$13
  sub $9,$9,$9
  sub $29,$29,$29
  mtc0 $0,$12
  ori $v1,$0,0x00007f00
  sw $a3,0($v1)
  eret
.text
  ori $8, 0x7fffffff
  ori $9, 0x1000
  add $11,$8,$9
  ori $t1 0x00007f00
  ori $a0,0x0009
  ori $a3,0xfc01
  ori $a1,1
  sw $a1,4($t1)
  sw $a0,0($t1)
  mtc0 $a3,$12
  nop
```

```
  ori $a0,1111
  j eee
  ori$29, $0, 0x0111
  eee:
  or  $10, $8, $9
  ori$28, $0, 0x0000
  ori$29, $0, 0x0000
  ori $29,0x1000
  lui $v1,1
  j end
  add $10,$8,$29
  end:
  lui $a0,1
  lui $a2,2
```

7.乘除相关

（1）.ktext 0x4180

```
  mfc0 $1,$13
  sub $a0,$a0,$a0
  eret
.text
  ori$8, 0x7fffffff
  ori $9, 0x1000
  div $8,$9
  lui $a0,0x7f00
```

```
    add $a0,$a0,$a0
    mthi $a0
    mflo $s7
    ori $a1,1
    ori $v0,1
    addu $a1,$v0,$v0
    nop
    ori $s0,11
  (2) .ktext 0x4180
    mfc0 $1,$13
    mtc0 $0,$12
    mflo $s2
    mfhi $s1
    ori $v1,$0,0x00007f00
    sw $a3,0($v1)
    eret
  .text
    ori $t1,0x00007f00
    ori $a0,0x0009
    ori $a3,0xfc01
    ori $a1,3
    sw $a1,4($t1)
    sw $a0,0($t1)
    mtc0 $a3,$12
```

```
  mult $t1,$a3
  mthi $a3
  mfhi $a2
  ori $29,0x1000
  lui $v1,1
```

（3）mult 中断

```
.ktext 0x4180
  mfc0 $1,$13
  mtc0 $0,$12
  mflo $s2
  mfhi $s1
  ori $v1,$0,0x00007f00
  sw $a3,0($v1)
  eret
.text
  ori $t1,0x00007f00
  ori $a0,0x0009
  ori $a3,0xfc01
  ori $a1,3
  sw $a1,4($t1)
  sw $a0,0($t1)
  mtc0 $a3,$12
  mult $t1,$a3
  mthi $a3
```

```
  mfhi $a2
  ori $29,0x1000
  lui $v1,1
(4) .ktext 0x4180
  mfc0 $1,$13
  mtc0 $0,$12
  mflo $s2
  mfhi $s1
  ori $v1,$0,0x00007f00
  sw $a3,0($v1)
  eret
.text
  ori $t1,0x00007f00
  ori $a0,0x0009
  ori $a3,0xfc01
  ori $a1,4
  sw $a1,4($t1)
  sw $a0,0($t1)
  mtc0 $a3,$12
  mult $t1,$a3
  mthi $a3
  mfhi $a2
  ori $29,0x1000
  lui $v1,1
```

```
(5) .ktext 0x4180
  mfc0 $1,$13
  mtc0 $0,$12
  mflo $s2
  mfhi $s1
  ori $v1,$0,0x00007f00
  sw $a3,0($v1)
  eret
.text
  ori $t1,0x00007f00
  mult $t1,$t1
  ori $a0,0x0009
  ori $a3,0xfc01
  ori $a1,3
  sw $a1,4($t1)
  sw $a0,0($t1)
  mtc0 $a3,$12
  mult $t1,$a3
  #mthi $a3
  #mfhi $a2
  ori $29,0x1000
  lui $v1,1
（6）.ktext 0x4180
  mfc0 $1,$13
```

```
  mtc0 $0,$12
  mflo $s2
  mfhi $s1
  ori $v1,$0,0x00007f00
  sw $a3,0($v1)
  eret
.text
  ori $t1,0x00007f00
  mult $t1,$t1
  ori $a0,0x0009
  ori $a3,0xfc01
  ori $a1,8
  sw $a1,4($t1)
  sw $a0,0($t1)
  mtc0 $a3,$12
  ori $29,0x1000
  mthi $a3
  ori $a0,0x000b
  sw $a0,0($t1)
  mthi $a3
  mfhi $a2
  ori $29,0x1000
  mult $t1,$t1
  div $a2,$a2
```

```
  lui $a0,1
110@00003000: $ 9 <= 00007f00
150@00003008: $ 4 <= 00000009
170@0000300c: $ 7 <= 0000fc01
190@00003010: $ 5 <= 00000008
270@00003020: $29 <= 00001000
310@00003028: $ 4 <= 0000000b
370@00003034: $ 6 <= 0000fc01
390@00003038: $29 <= 00001000
550@00004180: $ 1 <= 00000000
590@00004188: $18 <= 3f010000
610@0000418c: $17 <= 00000000
630@00004190: $ 3 <= 00007f00
870@00003044: $ 4 <= 00010000
```

八．思考题

1. 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”？

   硬件

   目标：CPU 与外设(被控对象)在硬件上连接构成一个有机整体

   方法：I/0 接口电路(接口、接口控制器)

   软件

   目标：控制设备工作方式，完成信息传送

方法：接口控制程序(或驱动程序)

硬软件接口就是计算机硬件与软件的交互手段，人类与电脑等信息机器或人类与程序之间的接口称为用户界面。电脑等信息机器硬件组件间的接口叫硬件接口。电脑等信息机器软件组件间的接口叫软件接口。硬件接口指的是两个硬件设备之间的连接方式。硬件接口既包括物理上的接口，还包括逻辑上的数据传送协议。软件不同部分之间的交互接口。通常就是所谓的 API——应用程序编程接口，其表现的形式是源代码。

2. 在我们设计的流水线中，DM 处于 CPU 内部，请你考虑现代计算机中它的位置应该在何处。

在主存中，CPU 通过高速缓存 cache 与主存交换数据。

3. BE 部件对所有的外设都是必要的吗？

不是，本次加的 timer 只能用 lw,sw,用不到 BE 了。

4. 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：

定时器在主程序中被初始化为模式 0；定时器倒计数至 0 产生中断；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。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。）

```
.ktext 0x4180
  mfc0 $1,$13
  mflo $s2
  mfhi $s1
  ori $v1,$0,0x00007f00
  sw $a0,0($v1)
```

```
    eret
  .text
    ori $t1,0x00007f00
    mult $t1,$t1
    ori $a0,0x0009
    ori $a3,0xfc01
    ori $a1,2
    sw $a1,4($t1)
    sw $a0,0($t1)
    mtc0 $a3,$12
    mult $t1,$a3
    ori $29,0x1000
    lui $v1,1
    lui $v0,2
    lui $s1,3
    lui $s0,4
```

5. 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？

设备实际上包括两部分接口控制器（也称为接口芯片）和设备主体，设备主体不直接与主机连接，而是通过接口控制器与主机连接。鼠标和键盘的输入信号相当于中断，当键盘、鼠标有信息时，产生一个中断然后中断例程会从端口读入数据到寄存器。CPU 接收到中断请求之后进入中断处理程序获得鼠标键盘的信息。