P7-MIPS 微体系——异常与中断实验报告

一、整体结构

P7 搭建的 MIPS 微体系在 Verilog 设计的流水线 CPU 基础上进行开发，支持的指令集是 MIPS-C3 指令集。本处理器为五级流水线设计：IF、ID、EX、MEM、WB，共有四（五）级流水寄存器：IF/ID、ID/EX、EX/MEM、MEM/WB（、GRF）。本流水线设计支持延迟槽。为了解决数据冒险而设计的转发数据来源全部来自某级流水线寄存器。本设计添加了 CP0、Bridge、计时器等硬件，可以支持中断与异常，并配有简单的外设——定时器来模拟外部中断。

二、模块规格

本系统顶层模块 mips 由四个子模块构成（CPU、Bridge、Timer0、Timer1），Bridge 主要沟通 CPU 与各个设备与主存，实现虚拟主存地址到各个设备的地址映射。

（一）CPU

信号名	方向	描述
clk	I	
reset	I	同步复位信号
CPU_Addr[31:0]	O	CPU 要读（写）的地址
CPU_WD[31:0]	O	CPU 要写的值
CPU_RD[31:0]	I	CPU 读出的值
CPU_total_WeEn	O	CPU 对 DM 和设备的总的写使能
DEV_Addr[31:0]	I	要读（写）DM 的地址（针对于 CPU 中的 DM）
DEV_WD[31:0]	I	要写 DM 的值（针对于 CPU 中的 DM）
DM_RD[31:0]	O	从 DM 中读出来的值
We_DM	I	针对于 DM 的写使能
HWInt[5:0]	I	6 路“原生态”硬件中断信号

MacroPC[31:0]	O	宏观 PC，此设计中 E 级

（二）Bridge

信号名	方向	描述
CPU_Addr[31:0]	I	CPU 要访问的地址（读/写）
CPU_WD[31:0]	I	CPU 要写入的值
CPU_RD[31:0]	O	CPU 从主存或设备中读出的值
CPU_total_WeEn	I	CPU 写主存和设备的总使能
DEV_Addr[31:0]	O	访问设备的地址
DEV_WD[31:0]	O	向设备中写入的值
DM_RD[31:0]	I	从 DM 中读出的值
Timer0_RD[31:0]	I	从 Timer0 读出的值
Timer1_RD[31:0]	I	从 Timer1 读出的值
We_DM	O	DM 写使能
We_Timer0	O	Timer0 写使能
We_Timer1	O	Timer1 写使能

（三）Timer

信号名	方向	描述
clk	I	
reset	I	同步复位信号
Addr[31:0]	I	访问 Timer 的地址
WE[31:0]	I	写 Timer 写使能
Din[31:0]	I	写入 Timer 的值
Dout[31:0]	O	从 Timer 中读出的值
IRQ	O	Timer 发出的中断信号

（一）GRF

表 1 GRF 模块接口

信号名	方向	描述
A1[4:0]	I	输入第一个读寄存器编号
A2[4:0]	I	输入第二个读寄存器编号
A3[4:0]	I	输入写寄存器编号
WrReg[31:0]	I	写入寄存器的内容
RD1[31:0]	O	读出编号为 A1 的寄存器的内容
RD2[31:0]	O	读出编号为 A2 的寄存器的内容
GRF_En	I	写寄存器使能信号，高电平可写寄存器
Clk	I	时钟信号
Rst	I	同步复位信号，将所有寄存器内容复位为 0
PC	I	输入当前执行指令的指令地址

表 2 GRF 功能定义

序号	功能	功能描述
1	读存储器	读 A1，A2 存储器内容
2	写存储器	使能信号有效时，将 WrReg 的内容写入 A3 存储器
3	同步复位	寄存器置 0

（二）ALU

表 3 ALU 模块接口

信号名	方向	描述
ALU_A[31:0]	I	输入操作数 A
ALU_B[31:0]	I	输入操作数 B
ALU_Out[31:0]	O	输出运算结果
ALUOp[4:0]	I	运算功能选择信号 00000　加

		00001　减 00010　或 00011　与 00100　异或 00101　或非 00110　Slt 00111　Sltu 01000　Lui 01001　Sll 01010　Srl 01011　Sra
AddOverFlow	0	加法溢出
SubOverFlow	0	减法溢出

表 4 ALU 功能定义

序号	功能	功能描述
1	运算	根据选择信号进行运算

（三）EXT

表 5 EXT 模块接口

信号名	方向	描述
In[15:0]	I	输入需要扩展的 16 位立即数
Out[31:0]	O	输出 32 位扩展结果
EXTOp	I	选择信号，0:零扩展，1：符号扩展

表 6 EXT 功能定义

序号	功能	功能描述
1	扩展	根据选择信号进行扩展

（四）IM

表 7 IM 模块接口

信号名	方向	描述
IM_Addr[31:0]	I	输入指令的地址
Instr[31:0]	O	输出当前 32 位指令

表 8 IM 功能定义

序号	功能	功能描述
1	读指令	读出 IM_Addr[31:0]处的指令

（五）NPC

表 9 NPC 模块接口

信号名	方向	描述
PC[4:0]	I	输入当前指令的地址
Imm16[15:0]	I	输入指令的 16 位立即数
Imm26[25:0]	I	输入指令的 26 位立即数
Reg	I	输入一个寄存器的值
NPCOp[2:0]	I	计算 NPC 的选择信号 000：PC+4 001：PC+4+signed_ext(Imm16||0^2) 010:PC[31:28]||Imm26||0^2 011:Reg 100:0x0000_4180（when ActivateCP0） 101:EPC(when CoolCP0)
NPC[31:0]	O	输出下一个地址

PC8_Out[31:0]	O	输出 PC+8
Stall	I	暂停信号，有效时冻结 PC，NPC <= PC
ActivateCP0	I	激活内核态信号
CoolCP0	I	关闭内核态信号

表 10 NPC 功能定义

序号	功能	功能描述
1	计算地址	计算 NPC，输出 PC+8

（六）DM

表 11 DM 模块接口

信号名	方向	描述
DM_Addr[31:0]	I	输入访问的地址
WrDM[31:0]	I	输入要向 DM 中写入的数据内容
WDM_En	I	DM 写使能信号，WDM_En 高电平有效可以写入
Byte_En[3:0]	I	独热编码的字节使能： Byte_En[x]=1,DM[DM_Addr]的第 X 字节可以写入 Byte_En[x]=0,DM[DM_Addr]的第 X 字节禁止写入 1111：SW 0011: SH 寄存器 15:0 写入地址 15:0 1100：SH 寄存器 15:0 写入地址 31:16 0001：SB 寄存器 7:0 写入地址 7:0 0010：SB 寄存器 7:0 写入地址 15:8 0100：SB 寄存器 7:0 写入地址 23:16 1000：SB 寄存器 7:0 写入地址 31:24
Clk	I	时钟信号
Rst	I	同步复位信号
DM_Out[31:0]	O	输出 DM_Addr 处的数据内容
PC[31:0]	I	输入当前执行指令的指令地址

表 12 DM 功能定义

序号	功能	功能描述
1	读数据	读 DM_Addr 处的数据内容
2	写数据	时钟上升沿到来时，若 WDM_En 信号有效，向 DM_Addr 处写数据
3	同步复位	时钟上升沿到来时，若 Rst 信号有效，则 DM 所有数据复位为 0

（七）BE_LOCATE(字节定位模块)

表 BE_EXT 模块接口

信号名	方向	描述
DM_Addr[31:0]	I	输入访问的地址
BE_LOCATE_Op [1:0]	I	选择 Addr[1:0]的译码方式： 00：按照 SW 指令译码 01：按照 SH 指令译码 10：按照 SB 指令译码
Byte_En[3:0]	O	计算出要写的字节

（八）DM_VALUE_EXT

表 BE_EXT 模块接口

信号名	方向	描述
DM_Addr[31:0]	I	输入访问的地址（内部对低两位以及选择信号进行译码）
DM_EXT_In[31:0]	I	输入按照字地址读出来的数据
DM_VALUE_Op[2:0]	I	数据扩展控制码： 000：无扩展 001：无符号字节数据扩展 010：符号字节数据扩展 011：无符号半字数据扩展 100：符号半字数据扩展
DM_True_Out[31:0]	O	扩展之后的 32 位数据内容

（九）IFU

表 13 DM 模块接口

信号名	方向	描述
Imm16[15:0]	I	输入指令的低 16 位立即数
Imm26[25:0]	I	输入指令的低 26 位立即数
Reg[31:0]	I	输入一个要将值传给 NPC 的寄存器的值
NPCOp[1:0]	I	选择计算 NPC 的方法： 00：PC+4 01：PC+4+signed_ext(Imm16||0^2) 10:PC[31:28]||Imm26||0^2 11:Reg
Rst	I	同步复位信号
Clk	I	时钟信号
Instr[31:0]	O	输出当前的指令
PC8_Out[31:0]	O	输出 PC+8
PC[31:0]	O	输出 PC

表 14 DM 功能定义

序号	功能	功能描述
1	计算 NPC	计算 NPC
2	读指令	输出当前指令

（十）COMP

表 COMP 模块接口

信号名	方向	描述
COMP1[31:0]	I	输入要判断的第一个数
COMP2[31:0]	I	输入要判断的第二个数

Zero	O	COMP1==COMP2 输出 1，反之 0
Large	O	COMP1(Rs)>0 为 1，否则为 0（符号数比较）
Little	O	COMP1(Rs)<0 为 1，否则为 0（符号数比较）
Equal	O	COMP1(Rs)==0 为 1，否则为 0（符号数比较）

表 COMP 功能定义

序号	功能	功能描述
1	判断相等	两数相等输出 1，反之 0
2	判断与 0 大小	

（十一）MDU 乘除模块

表 乘除模块接口

信号名	方向	描述
MDU_D1[31:0]	I	操作数 1（乘数\被除数）
MDU_D2[31:0]	I	操作数 2（乘数\除数）
Start	I	启动乘除法
WrHL[31:0]	I	要写入 LO，HI 寄存器的值
HI_En	I	写 HI 寄存器的写使能信号
LO_En	I	写 LO 寄存器的写使能信号
MD_En	I	当前指令是否为四条乘除指令之一，是 1 否 0
Clk	I	
Rst	I	
MDU_Sel[1:0]	I	选择乘法运算信号： 00 mult 01 multu 10 div 11 divu
R_HI[31:0]	O	读 HI 寄存器

R_LO[31:0]	O	读 LO 寄存器
Busy	O	繁忙信号

（十二）MUX

①MUX_EXT_Source

选择信号	输出
0	Imm16
1	经过零扩展的 Offset

②MUX_WrRegSel

选择信号	输出
000	ALU_Out
001	DM_Out
010	PC8_Out
011	MDU_Out
100	CP0_Out

③MUX_A3Sel

选择信号	输出
00	Rt
01	Rd
10	$31

④MUX_ALU_AOp

选择信号	输出
00	RD1
01	EXT_Out

⑤MUX_ALU_BOp

选择信号	输出
00	RD2
01	EXT_Out

⑥MUX_MDU_Out_Sel

选择信号	输出
0	HI
1	LO

三、流水线寄存器设计

（一）IF/ID

表 IF/ID 级流水寄存器端口设置

信号名	方向	描述
Instr_IF[31:0]	I	
PC8_Out_IF[31:0]	I	
PC_IF[31:0]	I	
Instr_ID[31:0]	O	
PC8_Out_ID[31:0]	O	
PC_ID[31:0]	O	
Clk	I	
Rst	I	用于流水线寄存器初始化
Stall	I	暂停信号，当 Stall 信号有效时不更新此寄存器
ActivateCP0	I	
CoolCP0	I	

ExcCode_True_F[4:0]	I	
ExcCode_D[4:0]	O	

（二）ID/EX

表 ID/EX 级流水寄存器端口设置

信号名	方向	描述
Instr_ID[31:0]	I	
RD1_ID[31:0]	I	
RD2_ID[31:0]	I	
EXT_Out_ID[31:0]	I	
PC8_Out_ID[31:0]	I	
PC_ID[31:0]	I	
Rx_ID[4:0]	I	
Instr_EX[31:0]	O	
RD1_EX[31:0]	O	
RD2_EX[31:0]	O	
EXT_Out_EX[31:0]	O	
PC8_Out_EX[31:0]	O	
PC_EX[31:0]	O	
Rx_EX[4:0]	O	
Clk	I	
Rst	I	用于流水线寄存器初始化，复位信号有效时，清零
Stall	I	暂停信号，当暂停信号有效时，Stall 信号有效，清零（NOP）
ActivateCP0	I	
CoolCP0	I	
ExcCode_True_D[4:0]	I	

ExcCode_E[4:0]	O	

（三）EX/MEM

表 EX/MEM 级流水寄存器端口设置

信号名	方向	描述
Instr_EX[31:0]	I	
ALU_Out_EX[31:0]	I	
MDU_Out_EX[31:0]	I	
CP0_Out_EX[31:0]	I	
RD2_EX[31:0]	I	
PC8_Out_EX[31:0]	I	
PC_EX[31:0]	I	
Rx_EX[4:0]	I	
Instr_MEM[31:0]	O	
MDU_Out_MEM[31:0]	O	
ALU_Out_MEM[31:0]	O	
CP0_Out_MEM[31:0]	O	
RD2_MEM[31:0]	O	
PC8_Out_MEM[31:0]	O	
PC_MEM[31:0]	O	
Rx_MEM[4:0]	O	
Clk	I	
Rst	I	初始化
ActivateCP0	I	

（四）MEM/WB

表 MEM/WB 级流水寄存器端口设置

信号名	方向	描述
Instr_MEM[31:0]	I	

ALU_Out_MEM[31:0]	I	
MDU_Out_MEM[31:0]	I	
DM_Out_MEM[31:0]	I	
CP0_Out_MEM[31:0]	I	
PC8_Out_MEM[31:0]	I	
PC_MEM[31:0]	I	
Rx_MEM[4:0]	I	
Instr_WB[31:0]	O	
ALU_Out_WB[31:0]	O	
MDU_Out_WB[31:0]	O	
DM_Out_WB[31:0]	O	
CP0_Out_WB[31:0]	O	
PC8_Out_WB[31:0]	O	
Rx_WB[4:0]	O	
PC_WB[31:0]	O	
Clk	I	
Rst	I	初始化
CoolCP0	I	

微系统设计文档

一、CP0 设计

信号名	方向	描述
A_RD[4:0]	I	读 CP0 寄存器堆的地址
A_WR[4:0]	I	写 CP0 寄存器堆的地址
WD[31:0]	I	写入 CP0 寄存器堆的值
RD[31:0]	O	从 CP0 寄存器堆读的值
We	I	写 CP0 寄存器堆写使能
PC[31:0]	I	传入的受害指令（如果是延迟槽，则是跳转指令）的地址，用于返回用户态时定位返回的地址
HWInt[5:0]	I	硬件中断信号，每一路对应一个硬件设备中断： HWInt[2]:interrupt HWInt[1]:Timer1 HWInt[0]:Timer0
ExcCode_[4:0]	I	输入异常识别码： 0：中断（需要配合中断异常请求信号，否则为默认值） 4：AdEL 5：AdES 10：RI 12：Ov
EPC[31:0]	O	保存传入的受害指令的地址，用于返回
IntReq	I	输出总的中断请求（只有中断）
Clk	I	时钟信号
Rst	I	同步复位
BD_	I	BD 位，如果 BD_为 1，则是延迟槽指令受害
ActivateCP0	I	中断异常状态触发信号
CoolCP0	I	中断异常状态结束信号

本设计中，将进出内核态的出入口设定在 E 级，所以将 CP0 设计为 E 级的一个部件。接下来可以看到，设定在 E 级带来的优势。

由于本实验中 CP0 中使用到的寄存器数量不多，而且 CP0 寄存器中不同位的行为存在差异于是，采取独立建模方法：

CP0 涉及到的功能有：①软件读寄存器，②软件写寄存器，③中断异常时对于部分位 BD、EXL、ExcCode、EPC 的写操作，④每周期都写的 Cause.IP 位，⑤输出允许下的中断信号。

```
//PRId
always@(posedge Clk)begin
   if(Rst)begin
      PRId <= 32'b0;
   end
end
//SR IE/IM
always@(posedge Clk)begin
   if(Rst)begin
      IM <= 6'b0;
      IE <= 1'b0;
   end
   else begin
      if(We)begin
         if(A_WR==`SR)begin
            IM <= WD[15:10];
            IE <= WD[0];
         end
      end
   end
end
//SR EXL
```

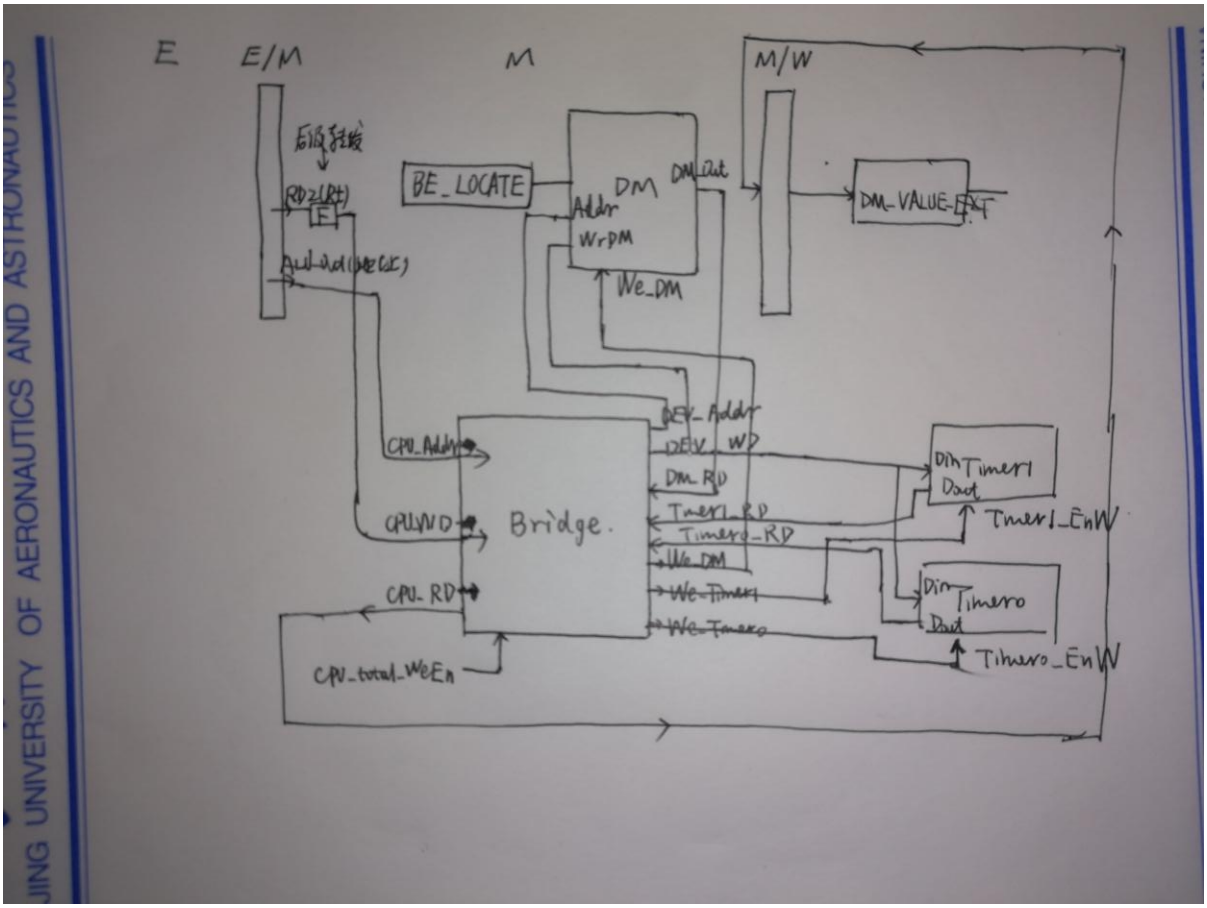
```
always@(posedge Clk)begin
   if(Rst)begin
      EXL <= 1'b0;
   end
   else begin
      if(We)begin//We,Activate,Cool 作为 E 级信号
         if(A_WR==`SR)begin
            EXL <= WD[1];
         end
      end
      else if(ActivateCP0)begin
         EXL <= 1'b1;
      end
      else if(CoolCP0)begin
         EXL <= 1'b0;
      end
   end
end
//EPC
always@(posedge Clk)begin
   if(Rst)begin
      EPC <= 32'b0;
   end
   else begin
      if(We)begin
         if(A_WR==`EPC)begin
            EPC <= WD;
         end
      end
```

```
            else if(ActivateCP0)begin
                EPC <= {PC[31:2],2'b0};
            end
        end
    end
    //Cause
    always@(posedge Clk)begin
        if(Rst)begin
            IP <= 6'b0;
            ExcCode <= 5'b0;
            BD <= 1'b0;
        end
        else begin
            IP <= HWInt;
            if(ActivateCP0)begin
                ExcCode <= ExcCode_;
                BD <= BD_;
            end
        end
    end
    //申请中断信号表达式
IntReq = (IE && (~EXL) && (|(IM & HWInt)));
```

二、桥与 IO 设计

信号名	方向	描述
CPU_Addr[31:0]	I	CPU 要访问的地址（读/写）
CPU_WD[31:0]	I	CPU 要写入的值
CPU_RD[31:0]	O	CPU 从主存或设备中读出的值
CPU_total_WeEn	I	CPU 写主存和设备的总使能

DEV_Addr[31:0]	O	访问设备的地址
DEV_WD[31:0]	O	向设备中写入的值
DM_RD[31:0]	I	从 DM 中读出的值
Timer0_RD[31:0]	I	从 Timer0 读出的值
Timer1_RD[31:0]	I	从 Timer1 读出的值
We_DM	O	DM 写使能
We_Timer0	O	Timer0 写使能
We_Timer1	O	Timer1 写使能



系统桥的功能是沟通 CPU 与各个设备（将主存视作一个设备）。将所有的设备寄存器都都映射到虚拟的“主存地址”，系统桥的功能便是实现这个虚拟映射。

系统桥可以实现的功能：①根据地址的范围判断属于哪一个设备②读出 CPU

操作的设备的寄存器值③实现 CPU 对设备的写操作。

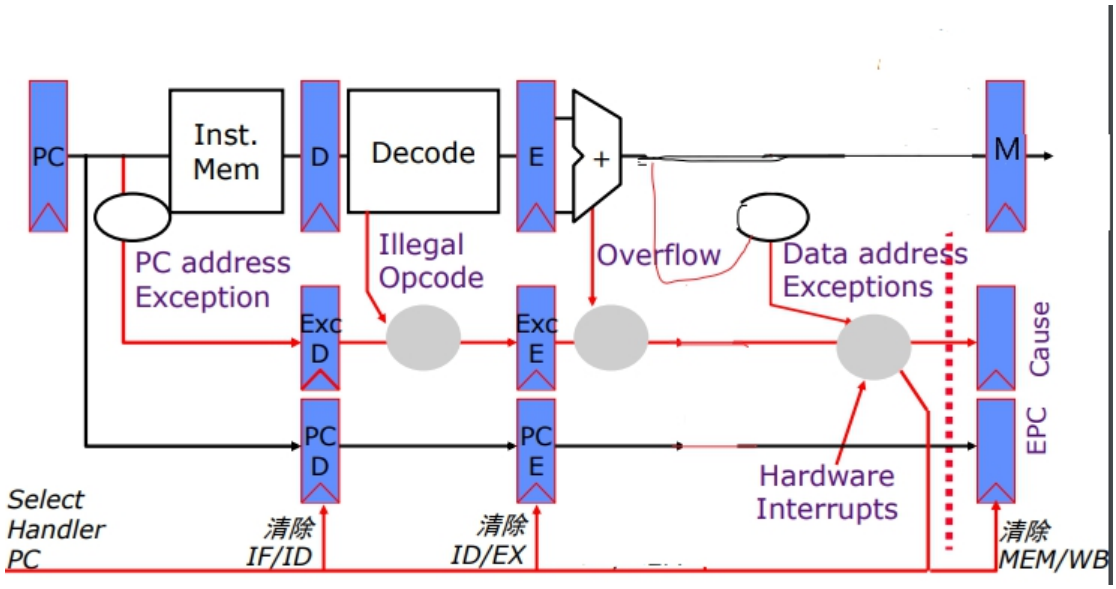
```
//传递读
assign CPU_RD = ((CPU_Addr>=`DM_Addr_start)&&(CPU_Addr<=`DM_Addr_end))? DM_RD:
                ((CPU_Addr>=`Timer0_Addr_start)&&(CPU_Addr<=`Timer0_Addr_end))? Timer0_RD:
                ((CPU_Addr>=`Timer1_Addr_start)&&(CPU_Addr<=`Timer1_Addr_end))? Timer1_RD:
                                                                                32'b0;
//传递写
assign We_DM = ((CPU_total_WeEn)&&((CPU_Addr>=`DM_Addr_start)&&(CPU_Addr<=`DM_Addr_end)));
assign We_Timer0 = ((CPU_total_WeEn)&&((CPU_Addr>=`Timer0_Addr_start)&&(CPU_Addr<=`Timer0_Addr_end)));
assign We_Timer1 = ((CPU_total_WeEn)&&((CPU_Addr>=`Timer1_Addr_start)&&(CPU_Addr<=`Timer1_Addr_end)));

assign DEV_WD = CPU_WD;
assign DEV_Addr = CPU_Addr;
```

三、异常与中断实现机制

中断与异常的口选在了 E 级，因为指令到达 E 级已经可以判断出所有异常情况，并且指令在 E 级时，并没有改变 DM（设备）和 GRF，而乘法的执行没有影响，回来再执行一遍等于重复写了 HILO 相同的内容（即使有时候前面可能多了一条跳转指令），完全没有影响，可以宏观上说 E 级时，指令还没有执行，返回时再执行。作为判断异常中断的点，E 级需要维护与特判的内容较少。ERET 返回时，选择在 E 级跳转，避免了如果在 D 级 EPC 还没有写需要转发（暂停）的情况。由于 E 级进行的中断异常的处理操作，其对于 NPC 的控制信号（Activate 与 Cool）要优先于 D 级的跳转指令。例如，E 级出错，D 级跳转，存在这个问题；E 级判断 ERET 后才清空前两级流水寄存器，E 级的 ERET 信号与 D 级的跳转也会存在 NPCOp 的优先级的判断问题。

主要分为三个部分：①异常信息和中断信息的收集与处理②得到 CP0 的激活与关闭信号的表达式③处理激活与关闭时的 CP0 等部件的一系列操作。

①异常信息和中断信息的收集与处理：建立 F\D,D\E 两级传递异常信息的流水线寄存器，上一级传递过来的被本级覆盖之后向下传递。中断主要是接入外部的中断信号，在 CP0 后判断允许的情况，如果能够允许，输出中断请求。E 级除了本身的异常会覆盖前面 D 传来的异常，作为最终的异常，还接入中断的申请和异常，判断 E 级的（异常||中断），如果成立，则 Activate。进入内核态。



②得到 CP0 的激活与关闭信号的表达式

```
assign ActivateCP0 =
((IntReqCP0)||(ExcCode_True_E!=5'b00000))? 1'b1: 1'b0;
assign CoolCP0 = (ERET_E)? 1'b1: 1'b0;
```

③处理激活与关闭时的 CP0 等部件的一系列操作

ActivateCP0 有效：受害 PC 传入 EPC，PC 指向 4180，EXL 置位，F\D,F\D_IntExc,D\E,D\E_IntExc，E\M 共 5 个流水线寄存器清零。

CoolCP0:EPC 写入 PC,EXL 恢复,F\D,F\D_IntExc,D\E,D\E_IntExc，4 个流水线寄存器清零。

采用宏观 BD 与宏观 PC 方法。

四、测试软件

见附录部分：

六、控制器设计

（一）模块设计

表 控制器模块接口

信号名	方向	描述
Instr[31:0]	I	
Zero	I	COMP1==COMP2 为 1，否则为 0
Large	I	COMP1(Rs)>0 为 1，否则为 0
Little	I	COMP1(Rs)<0 为 1，否则为 0
Equal	I	COMP1(Rs)==0 为 1，否则为 0
NPCOp[2:0]	O	计算 NPC 的选择信号， 000：PC+4 001：PC+4+signed_ext(Imm16||0^2) 010:PC[31:28]||Imm26||0^2 011:Reg 100:0x0000_4180 101:EPC
GRF_En	O	寄存器堆写使能信号,高电平可写
EXTOp	O	位扩展选择信号 0：零扩展 1：符号扩展
ALUOp[4:0]	O	运算功能选择信号 00000　加 00001　减 00010　或 00011　与 00100　异或 00101　或非 00110　Slt

		00111 Sltu 01000 Lui 01001 Sll 01010 Srl 01011 Sra
WDM_En	0	DM 写使能信号，高电平可写
A3Sel[1:0]	0	写寄存器的编号： 00：Rt 寄存器 01：Rd 寄存器 10：$31 寄存器
WrRegSel[2:0]	0	写寄存器的内容： 000：ALU_Out 001：DM_Out 010:PC8_Out 011:MDU_Out 100:CP0_Out
ALU_AOp[1:0]	0	ALU 操作数 A 来源 00:GRF.RD1 01：EXT.Out
ALU_BOp[1:0]	0	ALU 操作数 B 来源 00：GRF.RD2 01:EXT.Out
EXT_Source	0	选择被扩展的来源 0：16 位立即数 1: 5 位扩展后的结果
F_VALUE_WB_SEL[2:0]	0	选择从 WB 级向前转发的内容： 000: ALU_WB 001: DM_WB 010: IFU_WB(PC8)

		011: MDU_WB 100：CP0_WB
F_VALUE_MEM_SEL[1:0]	0	选择从 MEM 级向前转发的内容： 00: ALU_MEM 01: IFU_MEM 10: MDU_MEM 11: CP0_MEM
Start	0	当前是否是乘除法指令(MULT\MULTU\DIV\DIVU)： 0：不是 1：是，启动 MDU 运算
IsMD	0	当前是否是 MD 族指令 （MULT|MULTU|DIV|DIVU|MFHI|MFLO|MTHI|MTLO）： 0：不是 1：是
MDU_Out_Sel	0	选择 MDU 的输出结果： 0：输出 HI 1：输出 LO
HI_En	0	HI 寄存器写使能信号 1 可写
LO_En	0	LO 寄存器写使能信号 1 可写
BE_LOCATE_Op[1:0]	0	选择 Addr[1:0]的译码方式： 00：按照 SW 指令译码 01：按照 SH 指令译码 10：按照 SB 指令译码
DM_VALUE_Op[2:0]	0	数据扩展控制码： 000：无扩展 LW 001：无符号字节数据扩展 LBU 010：符号字节数据扩展 LB 011：无符号半字数据扩展 LHU 100：符号半字数据扩展 LH

MDU_Sel[1:0]	0	选择 MDU 的运算方式： 00：mult 01：multu 10：div 11：divu
LW	0	
LH	0	
LHU	0	
LB	0	
LBU	0	
SW	0	
SH	0	
SB	0	
B_type	0	
J_type	0	
ERET	0	
FakeInstr	0	

表 控制器功能定义

序号	功能	功能描述
1	识别指令	根据指令的 opcode 与 func 来识别指令
2	产生控制信号	根据指令产生控制信号

（二）实现机制

建立指令-控制信号对应表：

NPCOp[1:0]

GRF_En

EXTOp

ALUOp[4:0]

WDM_En

A3Sel[1:0]

WrRegSel[1:0]

ALU_AOp[1:0]

ALU_BOp[1:0]

EXT_Source

MUX_F_VALUE_WB[1:0]

MUX_F_VALUE_MEM[1:0]

Start

IsMD

MDU_Out_Sel

HI_En

LO_En

BE_LOCATE_Op

DM_VALUE_Op

MDU_Sel

(三)　控制信号流水

F	D	E	M	W
NPCOp				
GRF_En	GRF_En	GRF_En	GRF_En	GRF_En
	EXTOp			
		ALUOp		
			WDM_En	
	A3Sel			
				WrRegSel
		ALU_AOp		
		ALU_BOp		
	EXT_Source			

				F_VALUE_WB_SEL[1:0]
			F_VALUE_MEM_SEL	
	IsMD	Start		
		MDU_Out_Sel		
		HI_En		
		LO_En		
			BE_LOCATE_Op	
				DM_VALUE_Op

四、暂停转发机制构造

通过建立 AT 策略矩阵，比较 D 级指令的 Tuse 与 E、M、W 的指令 Tnew 进行比较，Tuse<Tnew 的点需要建立暂停机制，Stall 信号在 NPC 处优先级仅次于 Rst，在 IF/ID,ID/EX 两级流水线寄存器优先级与 Rst 相同。

通过计算 Tnew，当 E、M 的 Tnew 为 0 且写寄存器时，向前转发，W 级的 Tnew 一定为 0，所以写寄存器时也向前转发。

Addu 类：addu,subu,add,sub,and,or,xor,nor,slt,sltu

addiu 类：addiu,addi,andi,xori,ori,slti,sltiu

b 类：beq,bne,blez,bgtz,bltz,bgez

sll 类：sll,srl,sra

sllv 类：sllv,srlv,srav

lw 类：lw,lb,lbu,lh,lhu

sw 类：sw,sh,sb

	Tuse(D)			功能部件	Tnew			
	Rs	Rt	Rd		D	E	M	W
addu 类	1	1		ALU	2	1	0	0
Addiu 类	1			ALU	2	1	0	0
Lui				ALU	2	1	0	0
b 类	0	0						
Sll 类		1		ALU	2	1	0	0
Sllv 类	1	1		ALU	2	1	0	0
j								
jal				IFU	1	0	0	0
jr	0							
jalr	0			IFU	1	0	0	0
lw 类	1			DM	3	2	1	0

sw 类	1	2						
Mult 类	1	1						
mf 类				MDU	2	1	0	0
Mt 类	1							
eret								
mfc0				CP0	2	1	0	0
mtc0		1						

Rs

Tnew / Tuse	E			M			W		
	ALU	DM	IFU	ALU	DM	IFU	ALU	DM	IFU
	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

Rt

Tnew / Tuse	E			M			W		
	ALU	DM	IFU	ALU	DM	IFU	ALU	DM	IFU
	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

（一）暂停机制

同 P5，写出红色位点表达式

```
Stall_RS0_E1 = Tuse_RS0 && (Tnew_E==2'b01) && (Rs==Rx_EX)
&& (Rx_EX!=5'b00000) && (GRF_En_EX);
...
```

乘除法导致的暂停，Busy 或 Start 信号有效，并且当前 D 级的指令为乘除族：

```
Stall_MD = ((Busy||Start)&&(IsMD_ID));
Stall = Stall_RS || Stall_RT||Stall_MD;
```

（二）转发机制

转发思路：确定转哪一级寄存器、确定了级数后确定选择本级的什么值转发

①转发内容 MUX（当前级数的指令驱动,选择当前产生的最新结果）

注意从 DM 转发的内容一定是经过扩展后的新结果，并且添加转发 MDU_Out 的路径。

②确定转哪一级：后级只要产生结果并写寄存器就向前转，数据通路上的端口都要转

如要转发的端口：

D： COMP 的两个比较端口

RD1_ID(向下传递所以需要更新)

RD2_ID（向下传递所以需要更新）

E:

RD1_EX(用于计算)

RD2_EX（用于计算和向下传递）

M: DM.WrDM_MEM

③转发表达式的构建：

当前读数据的寄存器与转发级的指令写的寄存器是否相等

转发级的指令是否真的写寄存器

Tnew 是否为零

特判 0 寄存器

七、测试 CPU

见附录

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”？（Tips：什么是接口？和我们到现在为止所学的有什么联系？）

接口物理上可以理解为设备与 CPU 交互的寄存器堆，设备的功能非常多样化，而 CPU 的操作使以电平信号进行的，此时通过寄存器堆，将硬件的各种工作形式转化为能够为 CPU 操作的寄存器堆的状态变化。软件对硬件的操作实质上是对设备寄存器的读写操作，而寄存器堆的不同的状态决定了设备的工作状态。通过这些寄存器实现了指令（软件）对寄存器（硬件）的读写操作，从而使软件与硬件发生了交互。

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？

应该在 CPU 外部，DM 可以看做一个设备，通过系统桥与 CPU 联系起来，这一点上它和外在的设备是平权的，从逻辑上可以视作 CPU 外面的，其实物理上看 DM 体积比较大，功能单一，应该放在 CPU 外面。

3、BE 部件对所有的外设都是必要的吗？

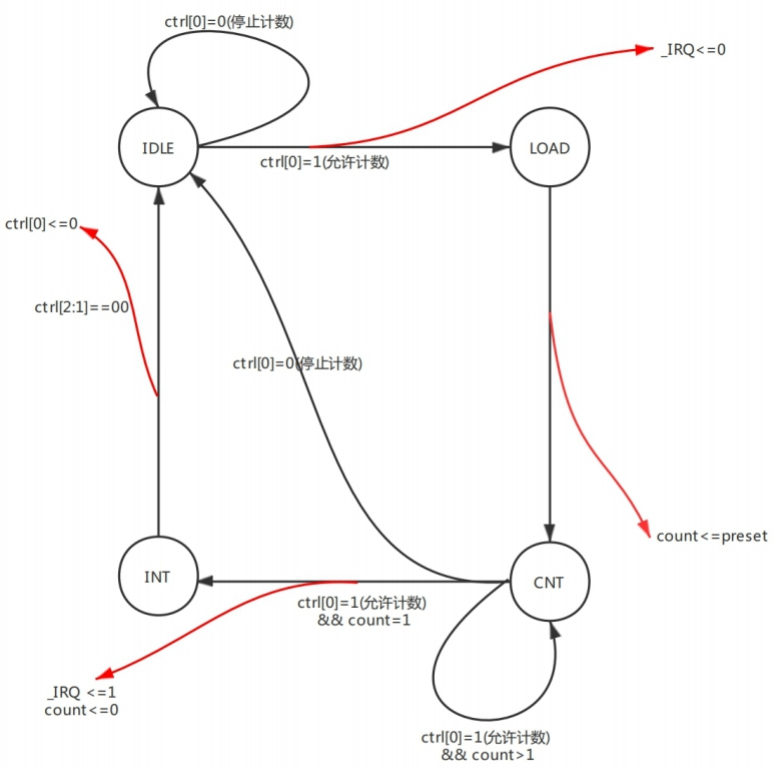
不是全都是必要的。在本实验中，由于只有 CPU 只需要对 DM 才会有半字、字节的操作，所以只需要在 DM 前面加一个属于 DM 自己的 BE 元件就可以。**如果设备不能够半字与字节操作，只能都整字操作，就不需要 BE**。但是为了使得系统桥更具有可扩展性，最好使他保留 BE 的信号。

4、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。

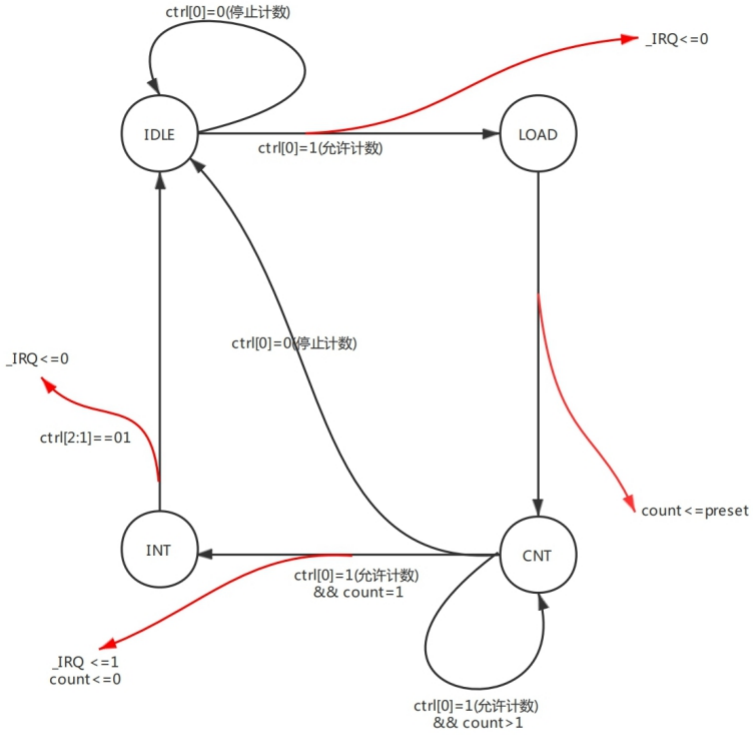
相同点：前三个状态的转移过程是相同的。

不同点：由 INT 向 IDLE 转移的那个上升沿，会根据 ctrl 的模式位的不同产生不同的操作。

状态转移图：



模式0



模式1

5、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。）

```
.data
   .globl store_scene
   store_scene: .space 160
.text
   li $s0,0x7f00 # timer0 基地址
   li $s1,0x7f10 # timer1 基地址
```

```
    mtc0 $0,$12
    # timer0
    sw $0,0($s0)
    li $t0,0x0080
    sw $t0,4($s0)# preset 0x80
    li $t0,0xfc01# SR
    mtc0 $t0,$12
    # timer1
    sw $0,0($s1)
    li $t0,0x0180
    sw $t0,4($s1)# preset 0x180
    li $t0,0xfc01# SR
    mtc0 $t0,$12
    # 允许计数，允许中断
    li $t0,9
    sw $t0,0($s0)
    sw $t0,0($s1)
.ktext 0x4180
    _entry:
        mfc0 $k1,$14 #先把 cause 的 IP 保存下来，之后每个周期都会写入
        j save_reg
        nop
    save_reg_back:
        # read ExcCode
        move $s0,$k1 #s0 保存进入时的 Cause

        mfc0 $k0,$13
        andi $k0,$k0,0x00ff
        srl $k0,$k0,2
```

```
    li $k1,0

    beq $k0,$k1,int_handler
    nop

int_handler:
    mfc0 $k0,$13
    andi $k0,$k0,0x0400
    srl $k0,$k0,10
    li $k1,1
    beq $k0,$k1,timer0_handler
    nop
    j timer1_handler
    nop
timer0_handler:
    li $k0,1
    li $s0,0x7f00
    sw $k0,0($s0)
timer1_handler:
    mfc0 $k0,$13
    andi $k0,$k0,0x0800
    srl $k0,$k0,11
    li $k1,1
    beq $k0,$k1,timer1_next
    nop
    # 否则结束
    j load_reg
    nop
timer1_next:
```

```
    li $k0,1
    li $s1,0x7f10
    sw $k0,0($s1)
    j load_reg
    nop

save_reg:
    la $k0,store_scene
    sw $1,0($k0)
    sw $2,4($k0)
    sw $3,8($k0)
    sw $4,12($k0)
    sw $5,16($k0)
    sw $6,20($k0)
    sw $7,24($k0)
    sw $8,28($k0)
    sw $9,32($k0)
    sw $10,36($k0)
    sw $11,40($k0)
    sw $12,44($k0)
    sw $13,48($k0)
    sw $14,52($k0)
    sw $15,56($k0)
    sw $16,60($k0)
    sw $17,64($k0)
    sw $18,68($k0)
    sw $19,72($k0)
    sw $20,76($k0)
    sw $21,80($k0)
    sw $22,84($k0)
```

```
    sw $23,88($k0)
    sw $24,92($k0)
    sw $25,96($k0)
    sw $28,108($k0)
    sw $29,112($k0)
    sw $30,116($k0)
    sw $31,120($k0)
    j reg_save_back
    nop
load_reg:
    la $k0,store_scene
    lw $1,0($k0)
    lw $2,4($k0)
    lw $3,8($k0)
    lw $4,12($k0)
    lw $5,16($k0)
    lw $6,20($k0)
    lw $7,24($k0)
    lw $8,28($k0)
    lw $9,32($k0)
    lw $10,36($k0)
    lw $11,40($k0)
    lw $12,44($k0)
    lw $13,48($k0)
    lw $14,52($k0)
    lw $15,56($k0)
    lw $16,60($k0)
    lw $17,64($k0)
    lw $18,68($k0)
    lw $19,72($k0)
```

```
    lw $20,76($k0)
    lw $21,80($k0)
    lw $22,84($k0)
    lw $23,88($k0)
    lw $24,92($k0)
    lw $25,96($k0)
    lw $28,108($k0)
    lw $29,112($k0)
    lw $30,116($k0)
    lw $31,120($k0)
    j return_
    nop
return_:
    eret
```

6、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？

鼠标和键盘是低速设备，通过发出中断的方式使得 CPU 知晓，并请求 IO 操作。按下键盘或者点一下鼠标，设备寄存器的内容便被改变了，此时设备会发出一个中断信号，此时 CPU 如果允许该中断，便会识别中断类型，分别执行不同的中断处理程序，把设备寄存器的读进去。例如敲击键盘，此时寄存器的值是按键的内容，CPU 中断执行时，把这个状态寄存器的值，读进去，此时便知道了键盘输入的内容。

	Tuse(D)			功能部件	Tnew			
	Rs	Rt	Rd		D	E	M	W
addu 类	1	1		ALU	2	1	0	0
Addiu 类	1			ALU	2	1	0	0
Lui				ALU	2	1	0	0
b 类	0	0						
Sll 类		1		ALU	2	1	0	0
Sllv 类	1	1		ALU	2	1	0	0
j								
jal				IFU	1	0	0	0
jr	0							
jalr	0			IFU	1	0	0	0
lw 类	1			DM	3	2	1	0
sw 类	1	2						
Mult 类	1	1						
mf 类				MDU	2	1	0	0
Mt 类	1							
eret								
mfc0				CP0	2	1	0	0
mtc0		1						

Tnew / Tuse	E			M				W			
	ALU	DM	IFU	ALU	MDU	DM	IFU	ALU	DM	IFU	MDU
	1	2	0	0	0	1	0	0	0	0	0
0	S1	S2	F01	F11	F	S4	F12	F21	F22	F23	
1	F	S3	F	F13	F	F	F15	F24	F25	F26	
2	F	F	F	F	F	F	F	F27	F28	F29	

Tnew / Tuse	E			M			W		
	ALU	DM	IFU	ALU	DM	IFU	ALU	DM	IFU
	1	2	0	0	1	0	0	0	0
0	S	S	F01（连续）	F11（间隔一个）	S	F12（间隔一个）	F21（间隔两个）	F22（间隔两个）	F23（间隔两个）
1	F	S	F	F13（连续）	F	F15（连续）	F24（间隔一个）	F25（间隔一个）	F26（间隔一个）
2	F	F	F	F	F	F	F27（连续）	F28（连续）	F29（连续）

Addu 类：addu,subu,add,sub,and,or,xor,nor,slt,sltu

addiu 类：addiu,addi,andi,xori,ori,slti,sltiu

b 类：beq,bne,blez,bgtz,bltz,bgez

sll 类：sll,srl,sra

sllv 类：sllv,srlv,srav

lw 类：lw,lb,lbu,lh,lhu

sw 类：sw,sh,sb

本设计中，通过 Tnew_E, Tnew_W 判断是否转发，确定转发点。利用 Tuse 与 Tnew 的关系，确定暂停点。

暂停：

S1:Addu, addiu， lui， sll, sllv, mf （mfc0）类-b 类, jr/jalr

S2:Lw 类-b 类,jr/jalr

S3:Lw 类-Addu,addiu，sll,sllv,sw,lw,mult,mt(mtc0)类

S4:Lw 类-xxx-b 类/jr/jalr

乘除暂停:mult\multv\div\divu-mult\multv\div\divu\mf\mt

转发:

Jal-jal(不可能)

B 类-jr，jalr(不可能)

Addu,addiu，lui，sll,sllv,mf 类-xxx-b，jr,jalr 类

Jal，jalr-nop-b 类($31,jalr 写入的寄存器)

Jal-nop-jr($31)（死循环）

Addu,addiu，lui，sll,sllv,mf(mfc0)类

-Addu,addiu，sll,sllv,sw,lw,mult,mt(mtc0)类

Addu,addiu，lui，sll,sllv,mf(mfc0)-sw 类（Rs,Rt）

Jal-cal(不可能)

Jal-load（不可能）

Jal-store（不可能）

Lw 类-xxx-xxx-b 类

Lw 类-xxx-xxx-jr，jalr 类

Lw 类-xxx-Addu,addiu，sll,sllv,sw,lw,mult,mt(mtc0)类

Lw 类-xxx-sw 类(Rs 冲突)

Lw 类-xxx-Lw 类

Lw 类-sw 类(Rt 冲突)

附录：

命令行导出 handler：

java -jar C:\Users\chl\Desktop\Mars.jar a db mc

CompactDataAtZero dump 0x00004180-0x00004ffc HexText

F:\ISE_files\P7_2020_12_23\code_handler.txt

F:\ISE_files\P7_2020_12_23\test_interrupt.asm

（一）支持 IO

```
.text
li $s0,0x987a
li $sp,0x2ffc
sh $s0,2($sp)
sb $s0,3($sp)
sw $s0,0($sp)
li $sp,0x7f04
sw $s0,0($sp)
sw $s0,-4($sp)
li $sp,0x7f14
sw $s0,0($sp)
sw $s0,-4($sp)
```

期望输出：

```
@00003000: $16 <= 0000987a
@00003004: $29 <= 00002ffc
@00003008: *00002ffc <= 987a0000
@0000300c: *00002ffc <= 7a7a0000
@00003010: *00002ffc <= 0000987a
@00003014: $29 <= 00007f04
@00003018: *00007f04 <= 0000987a（理论没有）
@0000301c: *00007f00 <= 0000000a（理论没有）
@00003020: $29 <= 00007f14
@00003024: *00007f14 <= 0000987a（理论没有）
@00003028: *00007f10 <= 0000000a（理论没有）
```

（二）异常

ExcCode	名称与助记符	指令或指令类型	描述与备注	备注
4	AdEL（取指异常）	所有指令	PC地址未4字节对齐	1. 测试时不会出现跳转到未加载指令的位置。2. 发生取指异常后视为nop直至提交至CP0.
		所有指令	PC地址超出0x3000-0x4ffc范围	
	AdEL（取数异常）	lw	取数地址未4字节对齐	
		lh、lhu	取数地址未2字节对齐	
		lh、lhu、lb、lbu	取Timer寄存器的值	
		Load类	计算地址加法溢出	
		Load类	取数地址超出DM、Timer0、Timer1的范围	
5	AdEs（存数异常）	sw	存数地址未4字节对齐	
		sh	存数地址未2字节对齐	
		sh、sb	存Timer寄存器的值	
		Store类	计算地址溢出	
		Store类	向计时器的Count寄存器存值	
		Store类	存数地址超出DM、Timer0、Timer1的范围	
10	RI（未知指令）	-	未知的指令码	1. 课上测试的未知指令的测试点中，非法指令的Opcode和Func码组合一定没有在正确指令集中出现。2. 发生RI异常后视为nop直至提交至CP0.
12	Ov（溢出异常）	add、addi、sub	算术溢出	store与load类算址溢出按照AdEL或AdES

```
.text
```

准备工作

```
li $s0,0x0001
li $s1,0xffff
lui $s2,0x7fff
ori $s2,$s2,0xffff
lui $s3,0xffff
ori $s3,$s3,0xefff
li $sp,0x0000
sw $s0,0($sp)
sw $s1,4($sp)
sw $s2,8($sp)
sw $s3,12($sp)
li $sp,0x2ffc
```

```
sw $s0,0($sp)
sw $s1,-4($sp)
sw $s2,-8($sp)
sw $s3,-12($sp)
li $sp,0x7f10
sw $s3,0($sp)
sw $s3,4($sp)

# AdEL,PC 没有字节对齐
li $ra,0x4fff
#jr $ra #人为修改成不对齐
nop

# AdEL,PC 越界
li $ra,0x5000
#jr $ra #人为修改越界
nop
# AdEL lw 取数没有四对齐
li $sp,0x0000
lw $t0,2($sp)
# AdEL lh,lhu 没有 2 对齐
li $sp,0x0004
lh $t0,3($sp)
lhu $t0,1($sp)
# AdEL lh,lhu,lb,lbu 取 Timer 寄存器的值
li $sp,0x7f10
lhu $t0,0($sp)
```

```
lh $t0,4($sp)
lh $t0,8($sp)
lh $t0,7($sp)# 测试双重异常
lbu $t0,3($sp)
# Load 类加法溢出
lui $sp,0x7fff
ori $sp,$sp,0xffff
lbu $t0,100($sp)
# load 超出 DM,Timer0,Timer1 的范围
li $sp,0x4000
lb $t0,0($sp)
li $sp,0x7f1b
lw $t0,0($sp)
# AdES sw 没有四对齐
li $sp,0x1000
sw $s2,3($sp)
# AdES sh 没有 2 对齐
sh $s3,3($sp)
# AdES sh,sb 存 Timer 寄存器
li $sp,0x7f00
sh $s3,2($sp)
sb $s3,3($sp)
# AdES Store 计算地址溢出
lui $sp,0x7fff
ori $sp,$sp,0xffff
sw $s1,200($sp)
# AdES store 向计时器的 count 寄存器存值
```

```
li $sp,0x7f00
sw $s2,8($sp)
# AdES store 存数地址超出 DM，Timer0，Timer1 的范围
li $sp,0x7f0b
sw $s2,0($sp)
li $sp,0x3000
sh $s2,4($sp)
# RI 位置指令
nop# 手动修改
nop# 手动修改
nop# 手动修改
# Ov 溢出
lui $t0,0x7fff
ori $t0,$t0,0xffff
lui $t1,0x7fff
ori $t1,$t1,0xffff

add $t2,$t0,$t1
addi $t2,$t0,0x7fff

lui $t3,0x8000
ori $t3,$t3,0x0000
lui $t4,0x7fff
ori $t4,$t4,0xffff
sub $t5,$t3,$t4

# 延迟槽异常
```

```
li $s7,1
li $s6,-1
li $s5,0
beq $s7,$s6,jump1
sub $t5,$t3,$t4

ori $t9,$0,0x1111# 测试是否执行
jump1:
ori $t0,$0,0x1234

j jump2
sub $t5,$t3,$t4
ori $t0,$0,0x1234# 测试是否执行
jump2:
ori $t0,$0,0x2323

# 暂停异常
div $s1,$s2
sub $t5,$t3,$t4
mflo $t0



.ktext 0x4004
label_4004:
ori $t0,$0,0xabab
ori $t1,$0,0xdeab
jr $ra
nop
```

预期结果：

```
 65@00003000: $16 <= 00000001
 75@00003004: $17 <= 0000ffff
 85@00003008: $18 <= 7fff0000
 95@0000300c: $18 <= 7fffffff
105@00003010: $19 <= ffff0000
115@00003014: $19 <= ffffefff
125@00003018: $29 <= 00000000
125@0000301c: *00000000 <= 00000001
135@00003020: *00000004 <= 0000ffff
145@00003024: *00000008 <= 7fffffff
155@00003028: *0000000c <= ffffefff
175@0000302c: $29 <= 00002ffc
175@00003030: *00002ffc <= 00000001
185@00003034: *00002ff8 <= 0000ffff
195@00003038: *00002ff4 <= 7fffffff
205@0000303c: *00002ff0 <= ffffefff
225@00003040: $29 <= 00007f10
255@0000304c: $31 <= 00004fff
275@00003054: $31 <= 00005000
295@0000305c: $29 <= 00000000
325@00004180: *00001ffc <= 00000000
335@00004184: *00002000 <= 00000000
355@00004188: $26 <= 00000002
365@0000418c: $26 <= 00000010
375@00004190: $26 <= 00003060
385@00004194: $26 <= 00003064
405@0000419c: $26 <= 00000000
415@000041a0: $27 <= 00000001
```

```
 465@00003064: $29 <= 00000004
 495@00004180: *00001ffc <= 00000000
 505@00004184: *00002000 <= 00000001
 525@00004188: $26 <= 00000002
 535@0000418c: $26 <= 00000010
 545@00004190: $26 <= 00003068
 555@00004194: $26 <= 0000306c
 575@0000419c: $26 <= 00000000
 585@000041a0: $27 <= 00000001
 655@00004180: *00001ffc <= 00000000
 665@00004184: *00002000 <= 00000001
 685@00004188: $26 <= 00000002
 695@0000418c: $26 <= 00000010
 705@00004190: $26 <= 0000306c
 715@00004194: $26 <= 00003070
 735@0000419c: $26 <= 00000000
 745@000041a0: $27 <= 00000001
 795@00003070: $29 <= 00007f10
 825@00004180: *00001ffc <= 00000000
 835@00004184: *00002000 <= 00000001
 855@00004188: $26 <= 00000002
 865@0000418c: $26 <= 00000010
 875@00004190: $26 <= 00003074
 885@00004194: $26 <= 00003078
 905@0000419c: $26 <= 00000000
 915@000041a0: $27 <= 00000001
 985@00004180: *00001ffc <= 00000000
 995@00004184: *00002000 <= 00000001
1015@00004188: $26 <= 00000002
1025@0000418c: $26 <= 00000010
```

```
1035@00004190: $26 <= 00003078
1045@00004194: $26 <= 0000307c
1065@0000419c: $26 <= 00000000
1075@000041a0: $27 <= 00000001
1145@00004180: *00001ffc <= 00000000
1155@00004184: *00002000 <= 00000001
1175@00004188: $26 <= 00000002
1185@0000418c: $26 <= 00000010
1195@00004190: $26 <= 0000307c
1205@00004194: $26 <= 00003080
1225@0000419c: $26 <= 00000000
1235@000041a0: $27 <= 00000001
1305@00004180: *00001ffc <= 00000000
1315@00004184: *00002000 <= 00000001
1335@00004188: $26 <= 00000002
1345@0000418c: $26 <= 00000010
1355@00004190: $26 <= 00003080
1365@00004194: $26 <= 00003084
1385@0000419c: $26 <= 00000000
1395@000041a0: $27 <= 00000001
1465@00004180: *00001ffc <= 00000000
1475@00004184: *00002000 <= 00000001
1495@00004188: $26 <= 00000002
1505@0000418c: $26 <= 00000010
1515@00004190: $26 <= 00003084
1525@00004194: $26 <= 00003088
1545@0000419c: $26 <= 00000000
1555@000041a0: $27 <= 00000001
1605@00003088: $29 <= 7fff0000
1615@0000308c: $29 <= 7fffffff
```

```
1645@00004180: *00001ffc <= 00000000
1655@00004184: *00002000 <= 00000001
1675@00004188: $26 <= 00000002
1685@0000418c: $26 <= 00000010
1695@00004190: $26 <= 00003090
1705@00004194: $26 <= 00003094
1725@0000419c: $26 <= 00000000
1735@000041a0: $27 <= 00000001
1785@00003094: $29 <= 00004000
1815@00004180: *00001ffc <= 00000000
1825@00004184: *00002000 <= 00000001
1845@00004188: $26 <= 00000002
1855@0000418c: $26 <= 00000010
1865@00004190: $26 <= 00003098
1875@00004194: $26 <= 0000309c
1895@0000419c: $26 <= 00000000
1905@000041a0: $27 <= 00000001
1955@0000309c: $29 <= 00007f1b
1985@00004180: *00001ffc <= 00000000
1995@00004184: *00002000 <= 00000001
2015@00004188: $26 <= 00000002
2025@0000418c: $26 <= 00000010
2035@00004190: $26 <= 000030a0
2045@00004194: $26 <= 000030a4
2065@0000419c: $26 <= 00000000
2075@000041a0: $27 <= 00000001
2125@000030a4: $29 <= 00001000
2155@00004180: *00001ffc <= 00000000
2165@00004184: *00002000 <= 00000001
2185@00004188: $26 <= 00000002
```

```
2195@0000418c: $26 <= 00000014
2205@00004190: $26 <= 000030a8
2215@00004194: $26 <= 000030ac
2235@0000419c: $26 <= 00000000
2245@000041a0: $27 <= 00000001
2315@00004180: *00001ffc <= 00000000
2325@00004184: *00002000 <= 00000001
2345@00004188: $26 <= 00000002
2355@0000418c: $26 <= 00000014
2365@00004190: $26 <= 000030ac
2375@00004194: $26 <= 000030b0
2395@0000419c: $26 <= 00000000
2405@000041a0: $27 <= 00000001
2455@000030b0: $29 <= 00007f00
2485@00004180: *00001ffc <= 00000000
2495@00004184: *00002000 <= 00000001
2515@00004188: $26 <= 00000002
2525@0000418c: $26 <= 00000014
2535@00004190: $26 <= 000030b4
2545@00004194: $26 <= 000030b8
2565@0000419c: $26 <= 00000000
2575@000041a0: $27 <= 00000001
2645@00004180: *00001ffc <= 00000000
2655@00004184: *00002000 <= 00000001
2675@00004188: $26 <= 00000002
2685@0000418c: $26 <= 00000014
2695@00004190: $26 <= 000030b8
2705@00004194: $26 <= 000030bc
2725@0000419c: $26 <= 00000000
2735@000041a0: $27 <= 00000001
```

```
2785@000030bc: $29 <= 7fff0000
2795@000030c0: $29 <= 7fffffff
2825@00004180: *00001ffc <= 00000000
2835@00004184: *00002000 <= 00000001
2855@00004188: $26 <= 00000002
2865@0000418c: $26 <= 00000014
2875@00004190: $26 <= 000030c4
2885@00004194: $26 <= 000030c8
2905@0000419c: $26 <= 00000000
2915@000041a0: $27 <= 00000001
2965@000030c8: $29 <= 00007f00
2995@00004180: *00001ffc <= 00000000
3005@00004184: *00002000 <= 00000001
3025@00004188: $26 <= 00000002
3035@0000418c: $26 <= 00000014
3045@00004190: $26 <= 000030cc
3055@00004194: $26 <= 000030d0
3075@0000419c: $26 <= 00000000
3085@000041a0: $27 <= 00000001
3135@000030d0: $29 <= 00007f0b
3165@00004180: *00001ffc <= 00000000
3175@00004184: *00002000 <= 00000001
3195@00004188: $26 <= 00000002
3205@0000418c: $26 <= 00000014
3215@00004190: $26 <= 000030d4
3225@00004194: $26 <= 000030d8
3245@0000419c: $26 <= 00000000
3255@000041a0: $27 <= 00000001
3305@000030d8: $29 <= 00003000
3335@00004180: *00001ffc <= 00000000
```

```
3345@00004184: *00002000 <= 00000001
3365@00004188: $26 <= 00000002
3375@0000418c: $26 <= 00000014
3385@00004190: $26 <= 000030dc
3395@00004194: $26 <= 000030e0
3415@0000419c: $26 <= 00000000
3425@000041a0: $27 <= 00000001
3505@000030ec: $ 8 <= 7fff0000
3515@000030f0: $ 8 <= 7fffffff
3525@000030f4: $ 9 <= 7fff0000
3535@000030f8: $ 9 <= 7fffffff
3565@00004180: *00001ffc <= 00000000
3575@00004184: *00002000 <= 00000001
3595@00004188: $26 <= 00000002
3605@0000418c: $26 <= 00000030
3615@00004190: $26 <= 000030fc
3625@00004194: $26 <= 00003100
3645@0000419c: $26 <= 00000000
3655@000041a0: $27 <= 00000001
3725@00004180: *00001ffc <= 00000000
3735@00004184: *00002000 <= 00000001
3755@00004188: $26 <= 00000002
3765@0000418c: $26 <= 00000030
3775@00004190: $26 <= 00003100
3785@00004194: $26 <= 00003104
3805@0000419c: $26 <= 00000000
3815@000041a0: $27 <= 00000001
3865@00003104: $11 <= 80000000
3875@00003108: $11 <= 80000000
3885@0000310c: $12 <= 7fff0000
```

```
3895@00003110: $12 <= 7fffffff
3925@00004180: *00001ffc <= 00000000
3935@00004184: *00002000 <= 00000001
3955@00004188: $26 <= 00000002
3965@0000418c: $26 <= 00000030
3975@00004190: $26 <= 00003114
3985@00004194: $26 <= 00003118
4005@0000419c: $26 <= 00000000
4015@000041a0: $27 <= 00000001
4065@00003118: $23 <= 00000001
4075@0000311c: $22 <= ffffffff
4085@00003120: $21 <= 00000000
4125@00004180: *00001ffc <= 00000000
4135@00004184: *00002000 <= 00000001
4155@00004188: $26 <= 00000002
4165@0000418c: $26 <= 80000030
4175@00004190: $26 <= 00003124
4185@00004194: $26 <= 00003128
4205@0000419c: $26 <= 00000000
4215@000041a0: $27 <= 00000001
4285@00004180: *00001ffc <= 00000000
4295@00004184: *00002000 <= 00000001
4315@00004188: $26 <= 00000002
4325@0000418c: $26 <= 00000030
4335@00004190: $26 <= 00003128
4345@00004194: $26 <= 0000312c
4365@0000419c: $26 <= 00000000
4375@000041a0: $27 <= 00000001
4425@0000312c: $25 <= 00001111
4435@00003130: $ 8 <= 00001234
```

```
4475@00004180: *00001ffc <= 00000000
4485@00004184: *00002000 <= 00000001
4505@00004188: $26 <= 00000002
4515@0000418c: $26 <= 80000030
4525@00004190: $26 <= 00003134
4535@00004194: $26 <= 00003138
4555@0000419c: $26 <= 00000000
4565@000041a0: $27 <= 00000001
4635@00004180: *00001ffc <= 00000000
4645@00004184: *00002000 <= 00000001
4665@00004188: $26 <= 00000002
4675@0000418c: $26 <= 00000030
4685@00004190: $26 <= 00003138
4695@00004194: $26 <= 0000313c
4715@0000419c: $26 <= 00000000
4725@000041a0: $27 <= 00000001
4775@0000313c: $ 8 <= 00001234
4785@00003140: $ 8 <= 00002323
4885@00004180: *00001ffc <= 00000000
4895@00004184: *00002000 <= 00000001
4915@00004188: $26 <= 00000002
4925@0000418c: $26 <= 00000030
4935@00004190: $26 <= 00003148
4945@00004194: $26 <= 0000314c
4965@0000419c: $26 <= 00000000
4975@000041a0: $27 <= 00000001
5105@0000314c: $ 8 <= 00000000
```

（三）转发暂停

```
.text
```

```
lui $s0,0x1fff
lui $s1,0x0fff
lui $s2,0xfe21
lui $s3,0xffff
ori $s0,$s0,0x0000
ori $s1,$s1,0x0010
ori $s2,$s2,0xabab
ori $s3,$s3,0x1111

#暂停
addiu $t0,$s1,0xff11dddd
bltz $t0,loop1
nop
ori $t9,$0,0x0011
addiu $t0,$s1,0xff11dddd
bgez $t0,loop1
nop
beq $t0,$t1,loop2
nop
srav $t9,$t1,$t2
bne $t0,$t1,loop3
nop
jr $t3
nop

loop3:
loop1:
addi $t0,$0,0x00003054
addu $t1,$t0,$0
```

```
loop2:
nop
jalr $t3,$t0
nop
sb $t1,0,($0)

sw $s0,0($0)
sw $s1,4($0)
sw $s2,8($0)
sw $s3,12($0)
sb $t1,3($0)
sb $t2,5($0)
sh $t3,6($0)
lbu $t0,3($0)
lbu $t1,4($0)
blt $t0,$t1,loop4
nop
xor $t1,$s1,$s2

loop4:
xor $t2,$s1,$s2

lh $t1,2($0)
lhu $t2,4($0)
mult $t1,$t2
mflo $t1
mfhi $t2
lbu $t1,5($0)
lb $t2,7($0)
```

```
divu $t1,$t2
mflo $t1
mflo $t2

ori $t5,$0,0x3110
lhu $t1,0($0)
nop
jr $t5
ori $t9,$0,0x1234
ori $t9,$0,0x2341
ori $t9,$0,0x3412
ori $t9,$0,0x4123
ori $t9,$0,0x1234
ori $t9,$0,0x2341
ori $t9,$0,0x3412
ori $t9,$0,0x4123
ori $t9,$0,0x1234
ori $t9,$0,0x2341
ori $t9,$0,0x3412
ori $t9,$0,0x4123

divu $s2,$s3
mflo $t2
div $s3,$s2
mflo $t2

mtlo $s2
mflo $t2
```

```
#转发
addi $t2,$0,0x315c
nop
jalr $t3,$t2
nop
ori $t1,$0,0x1111
ori $t1,$0,0x1122
ori $t1,$0,0x1331
ori $t1,$0,0x11a1
ori $t1,$0,0x1b11
ori $t1,$0,0x1c11
ori $t1,$0,0xe111
ori $t1,$0,0x111f
ori $t1,$0,0x1001
addu $t5,$0,$t3
jal loop5
nop
ori $t6,$0,0x123f

loop5:
ori $t8,$0,0x4000
blt $31,$t8,loop6
nop
srav $t5,$t2,$s1

loop6:
srlv $t5,$t2,$s1

lui $t4,0x01fe
```

```
mflo $t4
mtlo $t4
mflo $t4
lb $t4,8($0)

sllv $t5,$t2,$s2
ori $t5,$0,0
sh $s2,2($t5)
srlv $t6,$t2,$s2
ori $t6,0x0012
sh $s2,2($t6)

lh $t5,6($0)
sb $t5,0($t6)

lb $t2,8($0)
nop
nop
bgtz $t2,loop7
nop
slt $t9,$t1,$t2
slt $t9,$t2,$t1

loop7:
slt $t8,$t1,$t2
slt $t8,$t2,$t1
slti $t6,$t2,0x1111
slti $t7,$t2,0x0fff
sltiu $t7,$t2,0x0fff
sltiu $t6,$t2,0x1111
```

```
lbu $t6,13($0)
or $t6,$0,$0
sb $s2,9($t6)

lbu $t6,13($0)
multu $t5,$t6

lhu $t8,10($0)
nop
mthi $t8
```

```
# 补充测试新增转发暂停
lui $s0,0x1fff
lui $s1,0x0fff
lui $s2,0xfe21
lui $s3,0xffff
ori $s0,$s0,0x0000
ori $s1,$s1,0x0010
ori $s2,$s2,0xabab
ori $s3,$s3,0x1111

li $sp,0x0000
sw $s0,0($sp)
sw $s1,4($sp)
```

```
sw $s2,8($sp)
sw $s3,12($sp)

mtlo $s1
mflo $t0

li $a1,0xfc03
mtc0 $a1,$12
mfc0 $t0,$12

beq $s1,$t0,loop8
addu $t0,$t0,$t0
addi $t0,$t0,12

loop8:
slt $t0,$t1,$t2
lw $t9,0($sp)
mtc0 $a1,$12




div $s0,$s2
mflo $t0
mtc0 $a1,$12
mfc0 $t8,$12
mtc0 $a1,$12

mfc0 $t7,$13
sw $t7,40($0)
```

```
mtc0 $a1,$12
mfc0 $sp,$12
li $sp,0
sw $t7,48($sp)

lw $t5,4($0)
nor $t0,$s2,$s3
mtc0 $t5,$14
mfc0 $t8,$14
nop
nop
nop
```

预期结果：

```
65@00003000: $16 <= 1fff0000
75@00003004: $17 <= 0fff0000
85@00003008: $18 <= fe210000
95@0000300c: $19 <= ffff0000
105@00003010: $16 <= 1fff0000
115@00003014: $17 <= 0fff0010
125@00003018: $18 <= fe21abab
135@0000301c: $19 <= ffff1111
145@00003020: $ 1 <= ff110000
155@00003024: $ 1 <= ff11dddd
165@00003028: $ 8 <= 0f10dded
205@00003034: $25 <= 00000011
215@00003038: $ 1 <= ff110000
225@0000303c: $ 1 <= ff11dddd
235@00003040: $ 8 <= 0f10dded
275@00003068: $ 8 <= 00003054
```

```
285@0000306c: $ 9 <= 00003054
305@00003074: $11 <= 0000307c
325@00003054: $25 <= 00003054
365@0000307c: *00000000 <= 00000054
375@00003080: *00000000 <= 1fff0000
385@00003084: *00000004 <= 0fff0010
395@00003088: *00000008 <= fe21abab
405@0000308c: *0000000c <= ffff1111
415@00003090: *00000000 <= 54ff0000
425@00003094: *00000004 <= 0fff0010
435@00003098: *00000004 <= 307c0010
455@0000309c: $ 8 <= 00000054
465@000030a0: $ 9 <= 00000010
485@000030a4: $ 1 <= 00000000
525@000030b0: $ 9 <= f1deabbb
535@000030b4: $10 <= f1deabbb
545@000030b8: $ 9 <= 000054ff
555@000030bc: $10 <= 00000010
645@000030c4: $ 9 <= 00054ff0
655@000030c8: $10 <= 00000000
665@000030cc: $ 9 <= 00000000
675@000030d0: $10 <= 00000030
815@000030d8: $ 9 <= 00000000
825@000030dc: $10 <= 00000000
835@000030e0: $13 <= 00003110
845@000030e4: $ 9 <= 00000000
875@000030f0: $25 <= 00001234
885@00003110: $25 <= 00001234
895@00003114: $25 <= 00002341
905@00003118: $25 <= 00003412
```

```
915@0000311c: $25 <= 00004123
1045@00003124: $10 <= 00000000
1175@0000312c: $10 <= 00000000
1195@00003134: $10 <= fe21abab
1205@00003138: $10 <= 0000315c
1225@00003140: $11 <= 00003148
1245@0000315c: $ 9 <= 00001c11
1255@00003160: $ 9 <= 0000e111
1265@00003164: $ 9 <= 0000111f
1275@00003168: $ 9 <= 00001001
1285@0000316c: $13 <= 00003148
1295@00003170: $31 <= 00003178
1315@0000317c: $24 <= 00004000
1325@00003180: $ 1 <= 00000001
1365@00003190: $13 <= 00000000
1375@00003194: $12 <= 01fe0000
1385@00003198: $12 <= fe21abab
1405@000031a0: $12 <= fe21abab
1415@000031a4: $12 <= fffffab
1425@000031a8: $13 <= 018ae000
1435@000031ac: $13 <= 00000000
1435@000031b0: *00000000 <= abab0000
1455@000031b4: $14 <= 00000006
1465@000031b8: $14 <= 00000016
1465@000031bc: *00000018 <= 0000abab
1485@000031c0: $13 <= 0000307c
1485@000031c4: *00000014 <= 007c0000
1505@000031c8: $10 <= fffffab
1555@000031dc: $25 <= 00000000
1565@000031e0: $25 <= 00000001
```

```
1575@000031e4: $24 <= 00000000
1585@000031e8: $24 <= 00000001
1595@000031ec: $14 <= 00000001
1605@000031f0: $15 <= 00000001
1615@000031f4: $15 <= 00000000
1625@000031f8: $14 <= 00000000
1635@000031fc: $14 <= 00000011
1645@00003200: $14 <= 00000000
1645@00003204: *00000008 <= fe21abab
1665@00003208: $14 <= 00000011
1695@00003210: $24 <= 0000fe21
1765@0000321c: $16 <= 1fff0000
1775@00003220: $17 <= 0fff0000
1785@00003224: $18 <= fe210000
1795@00003228: $19 <= ffff0000
1805@0000322c: $16 <= 1fff0000
1815@00003230: $17 <= 0fff0010
1825@00003234: $18 <= fe21abab
1835@00003238: $19 <= ffff1111
1845@0000323c: $29 <= 00000000
1845@00003240: *00000000 <= 1fff0000
1855@00003244: *00000004 <= 0fff0010
1865@00003248: *00000008 <= fe21abab
1875@0000324c: *0000000c <= ffff1111
1905@00003254: $ 8 <= 0fff0010
1915@00003258: $ 5 <= 0000fc03
1935@00003260: $ 8 <= 0000fc03
1965@00003268: $ 8 <= 0001f806
1975@0000326c: $ 8 <= 0001f812
1985@00003270: $ 8 <= 00000000
```

```
1995@00003274: $25 <= 1fff0000
2135@00003280: $ 8 <= fffffffef
2155@00003288: $24 <= 0000fc03
2175@00003290: $15 <= 00000000
2175@00003294: *00000028 <= 00000000
2205@0000329c: $29 <= 0000fc03
2215@000032a0: $29 <= 00000000
2215@000032a4: *00000030 <= 00000000
2235@000032a8: $13 <= 0fff0010
2245@000032ac: $ 8 <= 00004444
2265@000032b4: $24 <= 0fff0010
```

（四）中断

```
.text

li $s0,0xfc01

mtc0 $s0,$12

# 打开中断的允许

# 正常中断

li $t1,1

li $t2,2

li $t3,3

li $t4,4

li $t5,5
```

```
li $t6,6

li $t7,7

li $t8,8

li $t9,9

# 延迟槽中断

j loop1

li $t1,4

li $t2,5

li $t4,6

loop1:

# div后面的中断

div $t2,$t3

j loop2

mflo $t2

li $t1,4

loop2:

# 异常和中断一起

ori $t2,$t3,100
```

```
lui $t3,0x7fff

ori $t3,$t3,0xffff

add $t4,$t3,$t3

nop

# 延迟槽后面的异常和中断一起

j loop3

add $t4,$t3,$t3

loop3:



.ktext 0x4180

mfc0 $k0,$14

addiu $k0,$k0,0

mtc0 $k0,$14

eret

Nop
```

附上 testbench:

```
module mips_tb;
```

```
// Inputs
reg clk;
reg reset;
reg interrupt;

// Outputs
wire [31:0] addr;

// Instantiate the Unit Under Test (UUT)
mips uut (
   .clk(clk),
   .reset(reset),
   .interrupt(interrupt),
   .addr(addr)
);

initial begin
   // Initialize Inputs
   clk = 0;
```

```
    reset = 0;
    interrupt = 0;
    #10;
    reset = 1;
    #15;
    reset = 0;
    // Wait 100 ns for global reset to finish
    #100;


    // Add stimulus here


end
   always #5 clk = ~clk;
    always@(posedge clk)begin
       if(addr==32'h3010) interrupt <= 1;
       else if(addr==32'h3020) interrupt <= 1;
       else if(addr==32'h3040) interrupt <= 1;
       else if(addr==32'h3058) interrupt <= 1;
       else interrupt <= 0;
```

```
        end

endmodule
```