支持异常与中断的CPU设计

设计思路：

1.先修改数据通路，改造成一个可以检测异常的cpu(因为之前的CPU是忽略了异常的产生，实际上可能会产生各种错误)。

ALU单元区分AriOV和DMOV的原因，可能一个DM型指令在加法的时候已经爆了，但是因为异常规定OV只检查add,sub,addi，即爆了的原因要纳入M级中，如果不区分的话，可能指令不是add，sub，addi，检测出了ov的异常，是不符合要求的。故要在ALU单元区分ALUOV和DMOV

2.然后根据异常生成对应的ExcCode进行流水，集中到CP0所在的M级，为接下来CP0的搭建做好准备

3.流水寄存器对异常的处理，生成nop指令，以及调整pc，保证宏观pc不会在异常时突变为0而是0x00004180

4.CP0的搭建

我们搭建的CP0允许6个外设的中断请求，因此IM的大小为6位，对应位置1表示允许来自该外设的中断请求，为0表示不理会该外设的中断请求，IP的大小也为6位，可以理解成外设，当IP的某位置1，表示该位对应的外设发出中断请求，若IM的对应位也是1，表示CP0要响应中断，因为IM表示具有发送中断的能力，IP表示发送了中断，一个具有发送中断的能力的设备发送了中断，自然完成了中断请求。

```
assign IntReq = (|((`IM)&(`HWInt)))&(!`EXL)&(`IE)
    //有设备，并且设备发送了一个中断请求&不在异常状态中&寄存器允许中断
```

5.Bridge的搭建

建立沟通cpu和外设的桥梁

6.实现mips整体架构，将用来的mips改为mips_cpu，组装好cpu与Bridge,Timer0,Timer1的mips整体架构，修改一些导线的名称

7.添加eret,mfc0,mtc0,syscall指令

8.对之前的行为进行适当的修正，完善，大概就完成了支持异常与中断cpu的设计

模块端口及其功能描述

Bridge模块

信号名	端口	描述
temp_m_data_addr[31:0]	I	来自于cpu的结果地址，传到桥里供桥选择从哪个外设读取数据
temp_m_data_wdata[31:0]	I	来自于cpu提供的写数据，传到桥里供桥选择写哪个外设
temp_m_data_byteen[3:0]	I	来自于cpu的使能信号
temp_m_data_rdata[31:0]	O	从外设中读取的数据，传回cpu供cpu进行相应处理

信号名	端口	描述
m_data_addr[31:0]	O	提供给顶层模块的输出
m_data_wdata[31:0]	O	提供给顶层模块的输出
m_data_byteen[3:0]	O	提供给顶层模块的输出
m_data_rdata[31:0]	I	从testbench中的dm读取到的DM的数据
TC0_data_rdata[31:0]	I	从timer0读取到的数据
TC1_data_rdata[31:0]	I	从timer1读取到的数据
TC0_WE	O	提供给timer0的使写能信号
TC1_WE	O	提供给timer1的使写能信号

TC模块

信号名	端口	描述
clk	I	
reset	I	
Addr[31:2]	I	
WE	I	
Din	I	
Dout	O	
IRQ	O	

mips_CPU模块:

·F级：

PC模块

信号名	端口	描述
clk	I	时钟信号
reset	I	同步复位信号
PCSrc[2:0]	I	000:pc=pc+4,001:pc=pc+4+sign_extend(offset||0^2),010:pc=pc=pc31...28||instr||0^2,011:pc=ra
immediate_32[31:0]	I	32位立即数
immediate_26[25:0]	I	j指令的26位立即数

信号名	端口	描述
ra[31:0]	I	返回的地址
D_PC[31:0]	I	D级PC，用于计算beq型跳转地址
stall	I	暂停信号,1:pc保持不变，即暂停,0:pc正常计算
Req	I	中断请求，即将进入异常处理程序
EPC_out	I	EPC结果，用于D_eret的跳转到EPC+4，因为EPC已经在F级中了
pc_out[31:0]	O	输出的pc

FD_reg模块

信号名	端口	描述
clk	I	时钟信号
reset	I	同步复位信号
stall	I	1:冻结，机器码保持为上一个周期的值，0:机器码正常进行读取
Req	I	中断请求，即将进入异常处理程序
F_PC[31:0]	I	F级的pc
F_Instr[31:0]	I	F级的指令
F_exception_code[3:0]	I	F级的异常码，用于流水，一直流水到CP0
F_isDelay	I	确定指令是否是延迟槽指令
D_PC[31:0]	O	流水到D级的pc
D_Instr[31:0]	O	流水到D级的指令
D_temp_exception_code[3:0]	O	流水的F级异常码
D_isDelay	O	流水到D级的确定指令是否为延迟槽指令

·D级：

GRF模块

信号名	端口	描述
A1[4:0]	I	读到RD1的寄存器的地址
A2[4:0]	I	读到RD2的寄存器的地址

信号名	端口	描述
A3[4:0]	I	进行写操作的寄存器的地址
WD[31:0]	I	写入的数据
clk	I	时钟信号
reset	I	同步复位信号
W_PC[31:0]	I	流水到的W级的pc，因为我们设定是只有在W级才会进行写寄存器的操作，所以写的pc应该就是一直随着流水线传递的pc
RD1[31:0]	O	从A1的寄存器读出的数据
RD2[31:0]	O	从A2的寄存器读出的数据

EXT模块

信号名	端口	描述
immediate_16[15:0]	I	16位的立即数
ExtOp	I	1:符号扩展,0:0扩展
immediate_32[31:0]	O	扩展成32位的立即数

D_ctrl模块

信号名	端口	描述
D_Instr[31:0]	I	流水到D级的指令
D_forward_RD1[31:0]	I	考虑了有转发的D级真实的会流水下去的输出
D_forward_RD2[31:0]	I	考虑了有转发的D级真实的会流水下去的输出，它俩用来比较，来提前beq的判断，减少开销
D_A3[4:0]	O	判断要写的寄存器，**如果**该条指令**非写寄存器**操作，我们就不需要关心A3的值，为了操作的连贯性，置0
RegDst[2:0]	O	000:写入rt寄存器，001:写入rd寄存器，010:写入31号寄存器，(提供选择信号供D_A3选择要写入的寄存器)
ExtOp	O	1：对符号数进行符号扩展，0：0扩展
PCSrc[2:0]	O	决定PC的计算方式,000：正常计算，001：beq型跳转，010:jal型跳转，011：jr型跳转，具体跳转方式参考pc模块
D_Tuse_rs[1:0]	O	判断从现在开始经过几个周期会用到GPR[rs]的数据

信号名	端口	描述
D_Tuse_rt[1:0]	O	判断从现在开始经过几个周期会用到GPR[rt]的数据
HILO_operation	O	判断D型指令是否为乘除槽相关指令
D_exception_RI	O	在D级判断指令是否是未知指令
D_exception_syscall	O	在D级判断指令是否为syscall
D_isDelay_operation	O	如果D级是beq,bne,jal,jr指令，就要将F_isDelay赋值为1
D_eret	O	判断D级的这条指令是否为eret指令

DE_reg模块

信号名	端口	描述
clk	I	时钟信号
reset	I	同步复位信号
stall	I	1：等价于插入了一个nop指令，0：正常进行
Req	I	中断请求，即将进入异常处理程序
D_forward_RD1[31:0]	I	考虑过转发后的rs寄存器真实数据
D_forward_RD2[31:0]	I	考虑过转发后的rt寄存器真实数据
D_immediate_32[31:0]	I	32位立即数
D_PC [31:0]	I	D级PC
D_Instr [31:0]	I	
D_A3 [4:0]	I	要进行写操作的寄存器，随流水线流水下去
D_exception_code[3:0]	I	D级的异常码
D_isDelay	I	流水的是否是延迟槽指令的信号
E_RD1[31:0]	O	随流水线流下去的rs寄存器真实数据
E_RD2[31:0]	O	随流水线流下去的rt寄存器真实数据
E_immediate_32[31:0]	O	随流水线流水下去的立即数
E_PC[31:0]	O	将D_PC的值流水下去
E_Instr[31:0]	O	将D_Instr的值流水下去
E_A3[4:0]	O	将D_A3的值流水下去

信号名	端口	描述
E_temp_exception_code[3:0]	O	E级的异常码
E_isDelay	O	流水的是否是延迟槽指令的信号

·E级：

ALU模块

信号名	端口	描述
ALUArithmetic	I	判断是否为add,addi,sub指令
ALUDM	I	判断是否为load,store型指令
SrcA[31:0]	I	ALU模块的第一个操作数
SrcB[31:0]	I	ALU模块的第二个操作数
ALUOp[2:0]	I	
ALUResult	O	计算结果
E_exception_AriOV	O	判断add,addi,sub指令是否产生了异常
E_exception_DMOV	O	判断DM型指令是否产生了异常

E_ctrl模块

信号名	端口	描述
E_Instr[31:0]	I	
E_PC[31:0]	I	
ALUOp[2:0]	O	决定运算操作
ALUSrc	O	1：选取流水到E级的32位立即数，0：选取考虑了转发后的rt寄存器数据E_forward_RD2
E_Tnew[1:0]	O	根据指令具体类型判断该指令产生的最终结果写入寄存器需要几个周期，00：与PC有关的指令操作，在D级就已经完成结果写入寄存器且随流水线传递了,01:ALU运算指令，在E级经过ALU产生E_ALUResult，再等一个周期才能经EM_reg寄存器流水到M级,10:DM型指令
E_WD[31:0]	O	根据E_Instr判断该条指令是什么指令，产生的结果是什么，判断是否能够产生所需需要的数据进行转发，因为E级只有pc是已经产生的，alu的计算结果和dm的访存结果都没产生，所以只能转发pc回去,通过该输出，我们就封装了让模块自己决定转发什么数据回去供D级使用
start	O	乘除槽启动的信号
MDOp[3:0]	O	乘除槽具体做的运算类型 0000:mult,0001:multu,0010:div,0011:divu,0100:mfhi,0101:mflo,0110:mthi,0111:mtlo,1111:不做乘除运算
ALUArithmetic	O	判断E_Instr是否为add,addi,sub指令
ALUDM	O	判断E_Instr是否为load,store型指令
E_mtc0	O	判断E_Instr是否为mtc0指令

MDU模块

信号名	端口	描述
clk	I	时钟信号
reset	I	同步复位信号
Op[3:0]	I	乘除槽运算类型
start	I	乘除槽启动信号
E_forward_RD1[31:0]	I	
E_forward_RD2[31:0]	I	
Req	I	异常中断信号，结合start信号判断乘除槽是否需要启动
Busy	O	
HI[31:0]	O	
LO[31:0]	O	

EM_reg模块

信号名	端口	描述
clk	I	时钟信号
reset	I	同步复位信号
E_PC[31:0]	I	
E_Instr[31:0]	I	
ALUResult_in[31:0]	I	
E_RD2[31:0]	I	
E_A3[4:0]	I	
E_Tnew[1:0]	I	
HI[31:0]	I	
LO[31:0]	I	
Req	I	异常中断信号
E_exception_code[3:0]	I	

信号名	端口	描述
E_isDelay	I	
E_exception_DMOV	I	
ALUResult_out[31:0]	O	
M_PC[31:0]	O	
M_Instr[31:0]	O	
M_RD2[31:0]	O	
M_A3[4:0]	O	
M_Tnew[1:0]	O	采用递减思路，免去了ctrl模块重复计算,M_Tnew={E_Tnew-1,0}
M_HI[31:0]	O	用于mfhi，mflo的转发
M_LO[31:0]	O	
M_temp_exception_code[3:0]	O	
M_isDelay	O	
M_exception_DMOV	O	

·M级：

M_ctrl模块

信号名	端口	描述
M_Instr[31:0]	I	
M_PC[31:0]	I	
M_ALUResult[31:0]	I	
M_WD[31:0]	O	决定转发回去的数据是什么,因为M级ALUResult和PC结果已经产生，所以根据指令类型判断当前面有转发需求时转发什么数据回去，相当于临时结果变量的存储
M_HI[31:0]	O	
M_LO[31:0]	O	
m_data_byteen[3:0]	O	字节使能信号
BEOp[2:0]	O	load类型指令取得的数据的扩展

信号名	端口	描述
M_mtc0	O	M级是mtc0指令，置1
M_eret	O	M级是eret指令，置1

BE模块

信号名	端口	描述
address[1:0]	I	M_ALUResult的最后两位，byte位
m_data_rdata[31:0]	I	从tb的DM中读出来的数据，需要根据指令类型进行扩展
BEOp[2:0]	I	000:lw,001:lb,011:lh
M_DMRD[31:0]	O	经过扩展的数据，可认为是从DM最终读出的正确数据

CP0模块

信号名	端口	描述
clk		
reset		
CP0_in[31:0]		
CP0_A1[4:0]		
CP0_A2[4:0]		
exception_code[3:0]		
M_PC[31:0]		
HWInt[5:0]		
CP0_WE		
EXLClr		
isDelay		
Req		
EPC_out[31:0]		
M_CP0_out[31:0]		

MW_reg模块

信号名	端口	描述
clk	I	
reset	I	
M_ALUResult[31:0]	I	
M_DMRD[31:0]	I	从内存中读取出来的数据
M_PC[31:0]	I	
M_Instr[31:0]	I	
M_A3[4:0]	I	
M_Tnew[1:0]	I	
M_HI[31:0]	I	
M_LO[31:0]	I	
M_CP0_out	I	
Req		
W_ALUResult[31:0]	O	
W_DMRD[31:0]	O	
W_A3[4:0]	O	
W_PC[31:0]	O	
W_Instr[31:0]	O	
W_Tnew[1:0]	O	其余均为流水信号，W_Tnew恒等于0
W_HI[31:0]	O	
W_LO[31:0]	O	
W_CP0_out	O	

·W级：

W_ctrl模块

信号名	端口	描述
W_Instr[31:0]	I	

信号名	端口	描述
W_ALUResult[31:0]	I	
W_DMRD[31:0]	I	
W_CP0_out[31:0]	I	
W_PC[31:0]	I	
W_HI[31:0]	I	
W_LO[31:0]	I	
W_WD[31:0]	O	此时alu的计算结果，pc，dm的访存结果均已经产生，根据需要挑选数据转发回去

Stall模块

信号名	端口	描述
D_Tuse_rs[1:0]	I	
D_Tuse_rt[1:0]	I	
E_Tnew[1:0]	I	
M_Tnew[1:0]	I	
D_A1[4:0]	I	
D_A2[4:0]	I	
E_A3[4:0]	I	
M_A3[4:0]	I	
HILO_operation	I	
start	I	
Busy	I	
D_eret	I	
E_mtc0	I	
E_Instr[31:0]	I	
M_mtc0	I	
M_Instr[31:0]	I	

信号名	端口	描述
stall	O	1:发送阻塞信号，进行pc，FD_reg，DE_reg寄存器的相应操作,0:无事发生

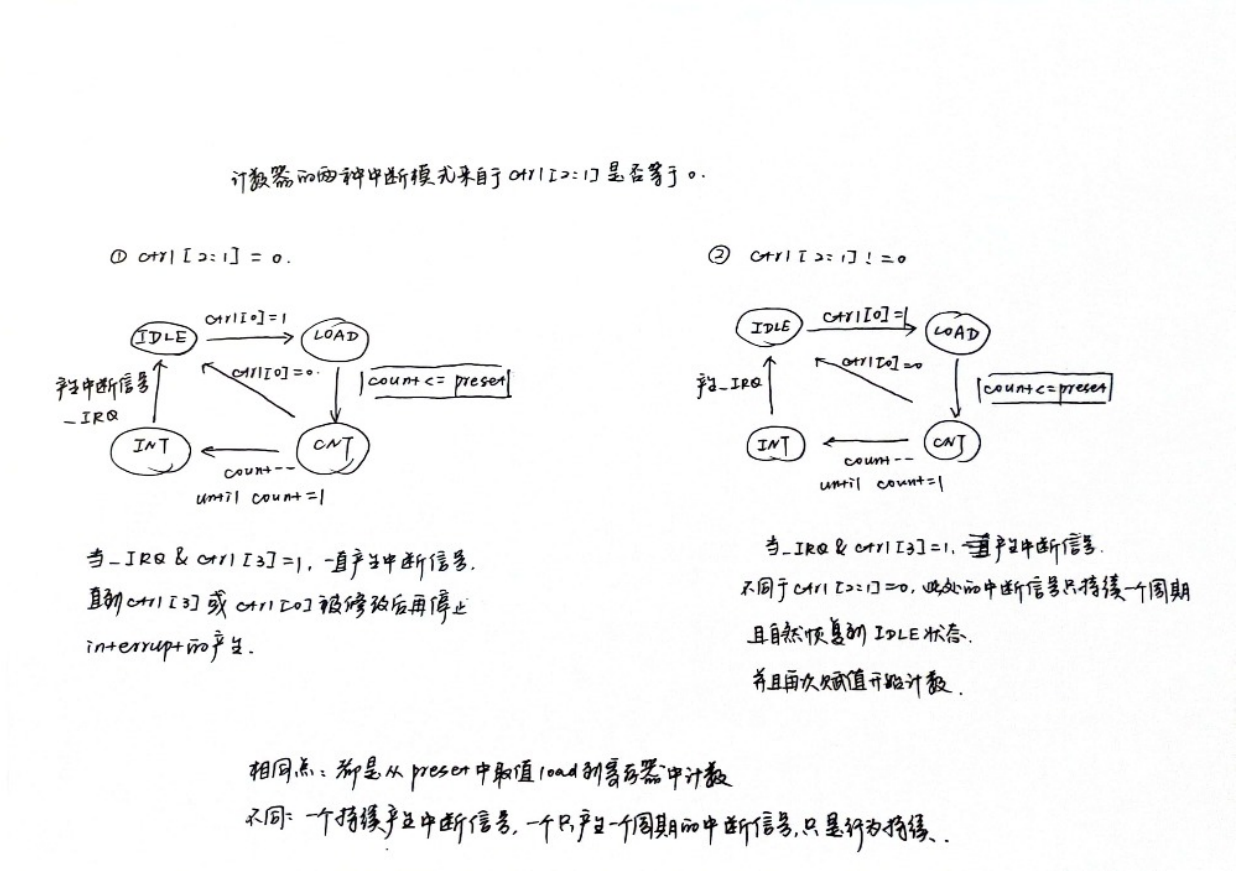
思考题

1.鼠标和键盘的输入信号会转化为不同的中断信号，如同本次实验的timer1,timer0一样，cpu根据中断信号执行对应的异常处理程序，就实现了输入信号被cpu知晓的要求。

2.我认为是可以的。本次实验指定异常处理程序的入口是0x00004180，其实也是用户根据要求自定义的入口，更改之后我们只需要实现当产生异常中断需求时跳到异常处理程序的入口的地址不一样而已，但是入口变动会使通用性降低，即不同指令的执行入口可能要更换很多个

3.外设有很多个，使用bridge，并在bridge通过多路选择器选择合适的数据回送至cpu即能实现功能。如果不使用桥，那么cpu就要直接与外设进行接口的设计，会使接口冗杂而多余，并且增加一个外设，就要重新设计一套读写逻辑，麻烦且逻辑不清晰，故采用桥的方式，由cpu输送要读取外设的地址到bridge里面由bridge选择正确的数据回送即可。

4.



5.宏观pc突变为0。因此清空流水线产生的空泡指令应该保留原指令的pc信息。

6.假如延迟槽指令出现了中断或者异常，$31的值已经被改变，在重新执行jalr指令的时候会与先前的结果不一样，因为$31寄存器的值发生了改变，所以会跳转到不对的pc值。