Project7 Mips微体系

一、CPU设计方案综述

（一）总体设计概述

- 处理器为32位处理器，5级流水线设计
- 支持的指令集为`MIPS-C3={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}`
- 所有运算型指令均需考虑溢出带来的异常
- 需要考虑延迟槽
- 支持同步复位

（二）关键模块定义

1.IFU

- 介绍：介绍：内部包括PC（程序计数器）、IM（指令存储器）及相关逻辑，PC用寄存器实现，具有同步复位功能，复位值为起始地址，IM容量为16KB
- 端口定义：

端口	方向	位宽	功能描述
clk	I	1	时钟控制信号
reset	I	1	同步复位信号
npc	I	1	下一个PC
WE	I	1	IFU使能信号
PC	O	32	当前指令地址
Instr	O	32	当前指令

- 功能定义

序号	功能名称	功能描述
1	同步复位	在时钟上升沿到来时，如果`reset`有效，则进行复位，复位PC位0x0000_3000
2	取指令	PC从IFU取出相应的指令
3	更新PC值	在时钟上升沿到来时，如果WE有效，则将PC更新为npc
4	暂停	当WE=0时，代表暂停操作

2.GRF

- 介绍：用具有写使能的寄存器实现，寄存器总数为32个，具有同步复位功能，其中0号寄存器始终为0
- 端口定义：

端口	方向	位宽	功能描述
clk	I	1	时钟信号
reset	I	1	复位信号
A1	I	5	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
A2	I	5	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
A3	I	5	5位地址输入信号，指定32个寄存器中的一个作为写入的目标寄存器
WE	I	1	写使能信号
WD	I	32	32位写入数据
PC	I	32	当前指令地址
RD1	O	32	A1所对应的寄存器的值
RD2	O	32	A2所对应的寄存器的值

- 功能定义：

序号	功能名称	功能定义
1	同步复位	在时钟上升沿到来时，如果reset有效时，则进行复位，将所有寄存器中存储的数值清零
2	读数据	读出A1、A2地址对应寄存器中所存储的数据到RD1、RD2
3	写数据	当WE有效且时钟上升沿来临时，将WD写入A3所对应的寄存器中

3.DM

- 介绍：容量为4KB，具有同步复位功能
- 端口定义：

端口	方向	位宽	功能描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
WE	I	1	写使能信号
Addr	I	32	写入地址
WD	I	32	写入的数据
PC	I	32	当前指令地址
RD	O	32	读出的数据

- 功能定义：

序号	功能名称	功能描述
1	复位	在时钟上升沿到来时，若reset有效，则进行复位
2	写入数据	当WE = 1时，将WD写入Addr地址中
3	读数据	RD始终为Addr地址的值

4.ALU

- 介绍：提供32位加、减、或运算以及大小比较功能，不检测溢出。
- 端口定义：

端口	方向	位宽	功能描述
SrcA	I	32	操作数1
SrcB	I	32	操作数2
ALUOP	I	3	ALU运算控制信号
Result	O	32	运算结果

- 功能定义：

序号	功能名称	功能描述
1	或运算	当ALUOP = 1，进行或运算
2	加法运算	当ALUOP = 2，进行加法运算
3	减法运算	当ALUOP = 3，进行减法运算
4	置高位操作	当ALUOP = 4，将B右移16位

5.Controller

- 介绍：产生控制信号
- 端口定义：

端口	方向	位宽	功能描述
I	I	32	当前指令
addu	O	1	addu信号
subu	O	1	subu信号
ori	O	1	ori信号
lw	O	1	lw信号
sw	O	1	sw信号
beq	O	1	beq信号
lui	O	1	lui信号
jal	O	1	jal信号
jr	O	1	jr信号
j	O	1	j信号
rs	O	5	当前指令的rs
rt	O	5	当前指令的rt
rd	O	5	当前指令的rd，当I=jal时，rd=31
R	O	1	除jalr、jr外的R型运算指令
load	O	1	需要从内存中取值的信号
store	O	1	需要存入内存的信号
type_i	O	1	I型运算指令
j_l	O	1	跳转并链接信号的值
j_ra	O	1	需要用到rs寄存器的跳转指令
j_2reg	O	1	需要用到双寄存器的跳转指令

- 功能定义：

序号	功能名称	功能定义
1	产生控制信号	产生控制信号

6.Ext

- 介绍：支持16bit到32bit的零拓展以及符号拓展
- 端口定义：

端口	方向	位宽	功能描述
imm_16	I	16	16位待拓展的立即数
ExtOp	I	1	拓展控制信号
Result	O	32	拓展后的结果

- 功能定义：

序号	功能名称	功能描述
1	零拓展	当ExtOp = 0时，进行零拓展
2	符号拓展	当ExtOp = 1时，进行符号拓展

7.IF_ID

- 介绍；I级和D级间的流水线寄存器，同时产生D级的控制信号
- 端口定义：

端口	方向	位宽	功能描述
reset	I	1	复位信号
clk	I	1	时钟控制信号
F_I	I	32	IFU指令
F_PC	I	32	IFU的PC
D_I	O	32	D级指令
D_PC	O	32	D级PC
D_Branch	O	4	跳转信号

- 功能定义：

序号	功能名称	功能描述
1	流水F_I	在时钟上升沿到来时，将F_I赋给D_I
2	流水F_PC	在时钟上升沿到来时，将F_PC赋给D_PC
3	生成D级控制信号	生成D级控制信号

8.ID_EX

- 介绍：D级和E级间的流水线寄存器，同时产生E级的控制信号
- 端口定义：

端口	方向	位宽	功能描述
clk	I	1	时钟控制信号
reset	I	1	复位信号
D_I	I	32	D级指令
D_PC	I	32	D级PC
D_RD1	I	32	D级RD1
D_RD2	I	32	D级RD2
E_I	O	32	E级指令
E_PC	O	32	E级PC
E_RD1	O	32	E级RD1
E_RD2	O	32	E级RD2
ExtOp	O	1	拓展信号
E_s	O	1	SrcB选择信号
E_ALUOP	O	1	ALU控制信号

- 功能定义：

序号	功能名称	功能描述
1	流水D_I	在时钟上升沿到来时，将D_I赋给E_I
2	流水D_PC	在时钟上升沿到来时，将D_PC赋给E_PC
3	流水D_RD1	在时钟上升沿到来时，将D_RD1赋给E_RD1
4	流水D_RD2	在时钟上升沿到来时，将D_RD2赋给E_RD2
5	产生E级控制信号	产生E级ExtOp、E_s、E_ALUOP信号

9.EX_MEM

- 介绍：E级和M级间的流水线寄存器，同时产生M级的控制信号
- 端口定义：

端口	方向	位宽	功能描述
clk	I	1	时钟控制信号
reset	I	1	复位信号
E_I	I	32	E级指令
E_PC	I	32	E级PC
E_AO	I	32	E级AO
E_RD2	I	32	E级RD2
M_WE	O	1	M级写入信号
M_I	O	32	M级指令
M_PC	O	32	M级PC
M_AO	O	32	M级AO
M_RD2	O	32	M级RD2

- 功能定义：

序号	功能名称	功能描述
1	流水E_I	在时钟上升沿到来时，将E_I赋给M_I
2	流水E_PC	在时钟上升沿到来时，将E_PC赋给M_PC
3	流水E_AO	在时钟上升沿到来时，将E_AO赋给M_AO
4	流水E_RD2	在时钟上升沿到来时，将E_RD2赋给M_RD2
5	产生M级控制信号	产生M级控制信号

10.MEM_WB

- 介绍：M级和W级间的流水线寄存器，同时产生W级的控制信号
- 端口定义：

端口	方向	位宽	功能描述
reset	I	1	复位信号
clk	I	1	时钟控制信号
M_I	I	32	M级指令
M_PC	I	32	M级PC
M_RD	I	32	M级RD
M_AO	I	32	M级AO
WE	O	1	GRF写入信号
ra	O	1	ra写入信号
RegDst	O	1	写入rd信号
GRF_PC	O	32	GRF的PC
GRF_I	O	32	GRF的I
W_RD	O	32	W级RD
W_AO	O	32	W级AO
WD_S	O	3	GRF写入数据选择信号

- 功能定义：

序号	功能名称	功能描述
1	流水M_I	在时钟上升沿到来时，将M_I赋给GRF_I
2	流水M_PC	在时钟上升沿到来时，将M_PC赋给GRF_PC
3	流水M_RD	在时钟上升沿到来时，将M_RD赋给W_RD
4	流水M_AO	在时钟上升沿到来时，将M_AO赋给W_AO
5	产生W级控制信号	产生W级控制信号

11.SFU

- 介绍：暂停 (Stall) 和 转发 (Forward) 的控制单元，产生两者的控制信号。
- 端口定义：

端口	方向	位宽	功能描述
d_I	I	32	D级指令
ex_I	I	32	E级指令
mem_I	I	32	M级指令
wb_I	I	32	W级指令
Stall	O	1	暂停信号
RD1_s	O	4	RD1选择信号 0：GRF[rs] 1：E级的PC + 8 2：M级的PC + 8 3：M级的AO
RD2_s	O	4	RD2选择信号 0：GRF[rs] 1：E级的PC + 8 2：M级的PC + 8 3：M级的AO
SrcA_s	O	4	SrcA选择信号 0：ID_EX寄存器的RD1 2：M级的PC + 8 3：M级的AO 4：W级的WD
SrcB_s	O	4	SrcB选择信号 0：ID_EX寄存器的RD2 2：M级的PC + 8 3：M级的AO 4：W级的WD
M_Data_s	O	4	M_Data选择信号 0：EX_MEM的AO 4：W级的WD

- 功能定义：

序号	功能名称	功能描述
1	D级RD1、RD2的选择信号	当D级比较器的两个输入与之前指令产生数据冲突时，进行转发
2	E级SrcA、SrcB的选择信号	当E级ALU的两个来源于之前指令产生数据冲突时，进行转发
3	M级Data的选择信号	当M级Data于之前指令产生数据冲突时，进行转发
4	暂停	当发生控制冲突时，IFU和IF_ID将暂停，同时清空ID_EX

12、MDU

- 乘除单元
- 端口定义：

端口	方向	位宽	功能描述
A	I	32	SrcA
B	I	32	SrcB
MDU_OP	I	32	乘除运算控制信号
start	I	1	乘除运算开始信号
clk	I	1	时钟信号
reset	I	1	复位信号
Busy	O	1	忙碌信号
hi	O	32	当前HI寄存器的值
lo	O	32	当前LO寄存器的值

- 功能定义：

序号	功能名称	功能描述
1	有符号乘	将`A * B`的运算结果放到`{HI,LO}`中，`A,B`被看作为有符号数
2	无符号乘	将`A * B`的运算结果放到`{HI,LO}`中，`A、B`被看作为无符号数
3	有符号除	将`A / B`的运算结果放到`{HI,LO}`中，`A、B`被看作为有符号数
4	无符号除	将`A / B`的运算结果放到`{HI,LO}`中，`A、B`被看作为无符号数
5	写`lo`寄存器	将A写入`lo`寄存器
6	写`hi`寄存器	将`A`写入`hi`寄存器

13、NPC

- 产生下一个PC
- 端口定义：

端口	方向	位宽	功能描述
D_PC	I	32	D级PC
PC	I	32	当前PC
imm_1	I	32	16位符号拓展后的结果
D_RD1	I	32	D级RD1
imm_jal	I	32	jal指令对应的立即数拓展
EPC	I	32	CP0接口的EPC
D_Branch	I	4	跳转控制信号
zero	I	1	相等信号
gez	I	1	大于等于0信号
gz	I	1	大于0信号
lz	I	1	小于0信号
lez	I	1	小于等于0信号
IntReq	I	1	中断请求
npc	O	32	下一个PC值

- 功能定义：

序号	功能名称	功能描述
1	产生下一个PC	根据控制信号产生下一个PC

14、CMP

- 比较器，放在D级
- 端口定义：

端口	方向	位宽	功能描述
A	I	32	D级转发后的rs值
B	I	32	D级转发后的rt值
zero	O	1	相等信号
gez	O	1	A>=0信号
gz	O	1	A>0信号
lez	O	1	A<=0信号
lz	O	1	A<0信号

- 功能定义：

序号	功能名称	功能描述
1	判断是否相等	判断是否相等
2	判断是否大于等于0	判断是否大于等于0
3	判断是否大于0	判断是否大于0
4	判断是否小于等于0	判断是否小于等于0
5	判断是否小于0	判断是否小于0

15、Bridge

- 系统桥：负责与外部设备沟通
- 端口定义：

端口	方向	位宽	功能描述
PrAddr	I	32	写入外设的地址
PrWD	I	32	写入外设的数据
DEV0_RD	I	32	timer0读出数据
DEV1_RD	I	32	timer1读出数据
PrRD	O	32	输出到CPU的数据
DEV_WD	O	32	写入外设的数据
DEV_Addr	O	32	写入外设的地址
HitDEV0	O	32	需要写入timer0
HitDEV1	O	32	需要写入timer1

- 功能定义：

序号	功能名称	功能描述
1	产生相应信号	产生相应信号

16、CP0

- 协处理器，包含4个32位寄存器，用于支持中断和异常
- 端口定义：

端口	方向	位宽	功能描述
clk	I	1	时钟信号
reset	I	1	复位信号
A1	I	5	读入CP0寄存器编号
A2	I	5	写入CP0寄存器编号
Din	I	32	写入CP0的数据
PC	I	32	当前PC值
ExcCode	I	5	异常类型
HWInt	I	6	6个外部中断
WE	I	1	写入使能信号
EXLClr	I	1	返回用户态信号
nBD	I	1	下一个分支延迟信号
IntReq	O	1	异常跳转信号
EPC	O	32	异常的PC值
DOut	O	32	读出的CP0寄存器的值

- 功能定义：

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IntReq。

（三）重要机制实现方法

利用Tuse-Tnew法解决数据冒险

- 先分析每个指令的Tuse和Tnew，然后根据Tuse和Tnew产生暂停以及转发信号
- 如果Tuse<Tnew，则优先考虑转发，其次考虑暂停
- 暂停
  - `beq`需要用到`rs,rt`寄存器，所以对于任何**需要写入寄存器**的指令，都需要考虑暂停和转发，其中如果和`lw`发生数据冲突，需要暂停两个周期
  - `jr`需要用到`rs`寄存器，所以对于任何**需要写入寄存器**的指令，都要考虑暂停和转发，与`beq`的情况类似
  - 除跳转指令外**任何需要用到寄存器进行计算**的指令与`lw`指令之如果发生了数据冲突，需要暂停
- 转发
  - D级与W级的转发已通过GRF内部转发实现，故无需考虑
  - D级比较器的两个输入的转发来源
    - E级的PC + 8
    - M级的PC + 8
    - M级的AO
  - SrcA、SrcB的转发来源
    - M级的PC + 8
    - M级的AO
    - W级的WD
  - M_Data的转发来源
    - W级的WD

M 级处理中断

- 将异常信号流水到 M 级，同时结合外部中断信号，判断此时是否要进行中断，如果进行中断就清空所有流水寄存器，并将 nPC 改为 0x00004180。
- 这样做 MEM/WB 流水线寄存器的指令如果需要写入 GRF，可以恰好写入而不用再特殊处理。
- 当D级为eret时，而E或M级向CP0的14号寄存器进行写入操作时，需要进行暂停操作。

测试样例

```
.text

li      $a0,0x3001
jr      $a0
TAG_1:  la  $a0,TAG_1
addi    $a0,$a0,1
li      $a0,0x3002
jr      $a0
TAG_2:  la  $a0,TAG_2
addi    $a0,$a0,1
li      $a0,0x3003
jr      $a0
TAG_3:  la  $a0,TAG_3
addi    $a0,$a0,1
```

```
li      $a0,0x2001
jr      $a0
TAG_4:  la  $a0,TAG_4
addi    $a0,$a0,1
li      $a0,0x2002
jr      $a0
TAG_5:  la  $a0,TAG_5
addi    $a0,$a0,1
li      $a0,0x2003
jr      $a0
TAG_6:  la  $a0,TAG_6
addi    $a0,$a0,1

li      $a0,0x5001
jr      $a0
TAG_7:  la  $a0,TAG_7
addi    $a0,$a0,1
li      $a0,0x5002
jr      $a0
TAG_8:  la  $a0,TAG_8
addi    $a0,$a0,1
li      $a0,0x5003
jr      $a0
TAG_9:  la  $a0,TAG_9
addi    $a0,$a0,1

li      $a0,0x6001
jr      $a0
TAG_10: la  $a0,TAG_10
addi    $a0,$a0,1
li      $a0,0x6002
jr      $a0
TAG_11: la  $a0,TAG_11
addi    $a0,$a0,1
li      $a0,0x6003
jr      $a0
TAG_12: la  $a0,TAG_12
addi    $a0,$a0,1

li      $a0,0x0000
jr      $a0
TAG_13: la  $a0,TAG_13
addi    $a0,$a0,1
li      $a0,0x5000
jr      $a0
TAG_14: la  $a0,TAG_14
addi    $a0,$a0,1
li      $a0,0x6000
jr      $a0
TAG_15: la  $a0,TAG_15
addi    $a0,$a0,1

li      $a0,0x3001
jalr    $30,$a0
TAGN_1: la  $a0,TAGN_1
addi    $a0,$a0,1
li      $a0,0x3002
```

```
jalr    $30,$a0
TAGN_2: la  $a0,TAGN_2
addi    $a0,$a0,1
li      $a0,0x3003
jalr    $30,$a0
TAGN_3: la  $a0,TAGN_3
addi    $a0,$a0,1

li      $a0,0x2001
jalr    $30,$a0
TAGN_4: la  $a0,TAGN_4
addi    $a0,$a0,1
li      $a0,0x2002
jalr    $30,$a0
TAGN_5: la  $a0,TAGN_5
addi    $a0,$a0,1
li      $a0,0x2003
jalr    $30,$a0
TAGN_6: la  $a0,TAGN_6
addi    $a0,$a0,1

li      $a0,0x5001
jalr    $30,$a0
TAGN_7: la  $a0,TAGN_7
addi    $a0,$a0,1
li      $a0,0x5002
jalr    $30,$a0
TAGN_8: la  $a0,TAGN_8
addi    $a0,$a0,1
li      $a0,0x5003
jalr    $30,$a0
TAGN_9: la  $a0,TAGN_9
addi    $a0,$a0,1

li      $a0,0x6001
jalr    $30,$a0
TAGN_10:la  $a0,TAGN_10
addi    $a0,$a0,1
li      $a0,0x6002
jalr    $30,$a0
TAGN_11:la  $a0,TAGN_11
addi    $a0,$a0,1
li      $a0,0x6003
jalr    $30,$a0
TAGN_12:la  $a0,TAGN_12
addi    $a0,$a0,1

li      $a0,0x0000
jalr    $30,$a0
TAGN_13:la  $a0,TAGN_13
addi    $a0,$a0,1
li      $a0,0x5000
jalr    $30,$a0
TAGN_14:la  $a0,TAGN_14
addi    $a0,$a0,1
li      $a0,0x6000
jalr    $30,$a0
TAGN_15:la  $a0,TAGN_15
```

```
addi    $a0,$a0,1

addi    $s2,$0,1000
mtc0    $s2,$14
mfc0    $s2,$14
addi    $s2,$2,1
mfc0    $s2,$14
lb      $s1,0($s2)
mfc0    $s2,$14
sb      $s2,0($0)
mfc0    $s2,$14
beq     $s2,0,TAGLST1
addi    $1,$1,1
addi    $1,$1,1
TAGLST1:

addi    $s2,$0,1000
mtc0    $s2,$14
mfc0    $s2,$14
nop
addi    $s2,$2,1
mfc0    $s2,$14
nop
lb      $s1,0($s2)
mfc0    $s2,$14
nop
sb      $s2,0($0)
mfc0    $s2,$14
nop
beq     $s2,0,TAGLST2
addi    $1,$1,1
addi    $1,$1,1
TAGLST2:

addi    $s2,$0,1000
mtc0    $s2,$14
mfc0    $s2,$14
nop
nop
addi    $s2,$2,1
mfc0    $s2,$14
nop
nop
lb      $s1,0($s2)
mfc0    $s2,$14
nop
nop
sb      $s2,0($0)
mfc0    $s2,$14
nop
nop
beq     $s2,0,TAGLST3
addi    $1,$1,1
addi    $1,$1,1
TAGLST3:

lui     $s2,1
mtc0    $s2,$14
```

```
mfc0    $s2,$14
addi    $s2,$s2,1
mtc0    $s2,$14
mfc0    $s2,$14
lw      $s2,0($0)
mtc0    $s2,$14
mfc0    $s2,$14

lui     $s2,1
nop
mtc0    $s2,$14
mfc0    $s2,$14
addi    $s2,$s2,1
nop
mtc0    $s2,$14
mfc0    $s2,$14
lw      $s2,0($0)
nop
mtc0    $s2,$14
mfc0    $s2,$14

lui     $s2,1
nop
nop
mtc0    $s2,$14
mfc0    $s2,$14
addi    $s2,$s2,1
nop
nop
mtc0    $s2,$14
mfc0    $s2,$14
lw      $s2,0($0)
nop
nop
mtc0    $s2,$14
mfc0    $s2,$14

nop
nop
nop

.ktext 0x4180
    mfc0    $t7,$12
    mfc0    $t7,$13
    mfc0    $t7,$14
    addu    $t7,$0,$a0
    mtc0    $t7,$14
    eret
    eret
```

二、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？）

- “硬件/软件接口”是指令（机器码）。硬件/软件接口指将二者联系起来的部分，比如本实验中CPU正常运行的用户态和进入异常的内核态之间的连接部分，或者是两者之间的一个桥梁。

2. BE 部件对所有的外设都是必要的吗?
   - 不是，只有对字节/半字有存取需求时才有必要。
3. 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移图。
   - 见计时器说明文档。
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：

   （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。

```
#main
.text
addi $1, $0, 9
addi $2, $0, 0x7f00                  #address of Timer0
addi $3, $0, 0x100                   #for preset
addi $4, $0, 0xfc02                   #for initial SR in CP0
mtc0 $4, $13                        #save SR
sw $3, 4($2)                         #save preset
sw $1, 0($2)                         #save control

Loop:
beq $0, $0, end
nop
end:
j Loop
nop

#exception handler
.ktext 4180
sw $24, 0($sp)                      #protect now
subi $sp, $sp, 4
sw $25, 0($sp)
subi $sp, $sp, 4
sw $26, 0($sp)
subi $sp, $sp, 4
sw $27, 0($sp)
subi $sp, $sp, 4
sw $28, 0($sp)
subi $sp, $sp, 4
sw $29, 0($sp)
subi $sp, $sp, 4
sw $30, 0($sp)
subi $sp, $sp, 4

mfc0 $26, $13
andi $27, $26, 0x0400                    #interrupt at normal instruction
andi $28, $26, 0x80000400               #interrupt at delay instruction
ori $29, $0, 0x400
ori $30, $0, 0x80000400
beq $27, $29, handler
```

```
nop
beq $28, $30, handler
nop

handler:
addi $25, $0, 9
addi $24, $0, 0x7f00
sw $25, 0($24)                                  #save control again

addi $sp, $sp, 4                                #return before interrupt
lw $30, 0($sp)
addi $sp, $sp, 4
lw $29, 0($sp)
addi $sp, $sp, 4
lw $28, 0($sp)
addi $sp, $sp, 4
lw $27, 0($sp)
addi $sp, $sp, 4
lw $26, 0($sp)
addi $sp, $sp, 4
lw $25, 0($sp)
addi $sp, $sp, 4
lw $24, 0($sp)

eret
```

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?
    - 点解鼠标或敲击键盘时，会产生一个与实验中的interrupt类似的一个信号，当鼠标或键盘操作时，产生外部中断信号，从而进入内核态。鼠标与键盘产生的中断信号不同，从而跳转至不同的内核程序，调用不同的程序。