计算机组成原理实验报告(P7)

一、 MIPS微系统设计方案综述

（一）、总体设计概述

本实验的MIPS微系统在P6实现的五级流水线CPU上增加CP0中断相关的寄存器与控制。并在CPU外增加系统桥和两个计时器的外设构成一个简单的MIPS系统。支持的指令集在MIPS-C3的基础上增加了{eret, mfc0, mtc0}三条新指令。可以支持除浮点运算外的绝大多数定点类程序的运行，并支持模拟简单的中断及异常过程。

为了实现这些功能微系统的设计分为四级，第一级最外层连接CPU，Bridge（系统桥）和两个Timer计时器，第二级主要对于CPU而言，其内部可划分为包含各个流水级的数据通路、冒险控制器、CP0寄存器、中断异常控制器。 其中数据通路、冒险控制器与P6相同。

对于各级指令，采用分布式译码的方式，在每一级中设置独立但完全相同的Controller, 并通过采取阻塞转发机制，处理冲突和各类冒险，在保证指令执行正确性的前提下，提高效率。最后，通过自己编写的对针对各种情形下中断和异常的测试程序，验证其正确性并debug。

异常的触发

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

（二）、关键模块定义

在P6的基础上，增加了如下的模块

1.CP0（M级中）

采用与课程ppt完全相同的模块接口：

设计CP0：模块接口

信号名	方向	用途	产生来源及机制
A1[4:0]	I	读CP0寄存器编号	执行MFC0指令时产生
A2[4:0]	I	写CP0寄存器编号	执行MTC0指令时产生
DIn[31:0]	I	CP0寄存器的写入数据	执行MTC0指令时产生 数据来自GPR
PC[31:2]	I	中断/异常时的PC	PC
ExcCode[6:2]	I	中断/异常的类型	异常功能部件
HWInt[5:0]	I	6个设备中断	外部硬件设备(如鼠标、键盘)
We	I	CP0寄存器写使能	执行MTC0指令时产生
EXLSet	I	用于置位SR的EXL(EXL为1)	流水线在W阶段产生
EXLClr	I	用于清除SR的EXL(EXL为0)	执行ERET指令时产生
clk	I	时钟	
rst	I	复位	
IntReq	O	中断请求，输出至CPU控制器	是HWInt/IM/EXL/IM的函数
EPC[31:2]	O	EPC寄存器输出至NPC	
DOut[31:0]	O	CP0寄存器的输出数据	执行MFC0指令时产生，输出数据至GPR

对于CP0内部的四个寄存器，设计如下

1.SR

31——8	7——2	1	0
others	IM中断掩码位	EXL（为1时进入内核态）	IE（全局中断允许信号）

2.Cause

31	30——16	15——10	9——8	6——2	1——0
BD（是否延迟槽异常或中断）	others	IP域（中断请求信号）	others	Exc异常信息	others

3.EPC

31——0
返回地址

4.PRID

31——0
自定义标签

2.Bridge（顶层模块）

名称	I\O	端口大小	说明
PrAddr	I	31:2	内存地址
PrWD	I	31:0	写入内存的数据
interrupt	I	1	外部中断信号
IRQ1	I	1	TC1中断请求信号
IRQ2	I	1	TC2中断请求信号
We1	O	1	TC1写使能信号
We2	O	1	TC2写使能信号
HWInt	O	7:2	中断请求
PrRD1	I	31:0	TC1读出数据
PrRD2	I	31:0	TC2读出数据
PrRD	O	31:0	写入CPU的数据

3.Timer1&Timer2(顶层模块)

名称	I\O	端口大小	说明
addr	I	31:0	宏观pc（调试时使用）
clk	I	1	时钟信号
reset	I	1	同步复位信号
Addr	I	31：2	地址信号
WE	I	1	写使能信号
Din	I	31：0	写入数据信号
Dout	O	31：0	读出数据信号
IRQ	O	1	中断请求位

1.F级

1.PC

名称	I\O	端口大小	说明
PC	O	31:0	当前指令指针
NPC	I	31:0	下一个指令指针
clk	I	1	时钟信号
reset	I	1	同步复位信号
PCEnD	I	1	指令寄存器使能

2. NPC

名称	I\O	端口大小	说明
PC	I	31:0	当前指令指针
NPCOp	I	1:0	00:PC+4, 01:beq, 10:jal, 11:jr
IMM	I	25:0	jal 写入地址
RA	I	31:0	jr 跳转地址
PC4	O	31:0	PC+4
NPC	O	31:0	下一个指令指针

3.IM

名称	I\O	端口大小	说明
Iaddr	I	31:0	pc指针
Instr	O	31:0	当前指令

2.F\D寄存器

名称	I\O	端口大小	说明
clk	I	1	时钟信号
reset	I	1	同步复位信号
FDEn_FD_I	I	1	寄存器使能端
Instr_FD_I	I	31:0	FD指令输入（F级当前指令）
PC_FD_I	I	31:0	FDpc输入（F级当前pc值）
PC_FD_O	O	31:0	FDpc输出（D级当前pc值）
Instr_FD_O	O	31:0	FD指令输出（D级当前指令）

3. D级

1.GRF

名称	I\O	端口大小	说明
A1	I	4:0	rs地址（25:21）
A2	I	4:0	rt地址（20:16）
A3	I	4:0	WR选择信号 00：rs（20:16），01：rd（15:10），$ra （5'h1f）
WD	I	31:0	写入信号
RD1	O	31:0	rs寄存器的值
RD2	O	31:0	rt寄存器的值

2. Extender

名称	I\O	端口大小	说明
ExtOp	I	1:0	00: signed_extend 01: unsigned_extend 10: lui({in[15:0]},{16{0}})
in	I	15:0	Instr[15:0]
out	O	32:0	输出结果

3.Acoder

对各个指令的AT信息进行译码

名称	I\O	端口大小	说明
Opcode	I	5:0	Opcode选择信号
Funct	I	5:0	Funct选择信号
Tusers	O	2:0	rs的使用时间
Tusert	O	2:0	rt的时钟时间
Tnew	O	2:0	产生结果的时间

4.D\E寄存器

名称	I\O	端口大小	说明
clk	I	1	时钟信号
reset	I	1	同步复位信号
Instr_DE_I	I	31:0	FD指令输入（D级当前指令）
PC_DE_I	I	31:0	FDpc输入（D级当前pc值）
PC_DE_O	O	31:0	FDpc输出（E级当前pc值）
Instr_DE_O	O	31:0	FD指令输出（E级当前指令）
RD1_DE_I	I	31:0	value of rs(D级)
RD2_DE_I	I	31:0	value of rt(D级)
Ext_DE_I	I	31:0	Ext resut(D级)
RD1_DE_I	O	31:0	value of rs(E级)
RD2_DE_I	O	31:0	value of rt(E级)
Ext_DE_O	O	31:0	Ext resut(E级)
Tnew_DE_I	I	2:0	冒险控制Tnew(D级)
Tnew_DE_O	O	2:0	冒险控制Tnew(E级)

5. E级

1.ALU

名称	I\O	端口大小	说明
SrcA	I	31:0	参与运算的A信号
SrcB	I	31:0	参与运算的B信号
Zero	O	31:0	beq相等信号
result	O	31:0	计算结果
ALUControl	I	31:0	ALU控制信号

2.WD_MUX

对写入的目标寄存器进行选择 2'b00选择rt寄存器， 2'b01选择rd寄存器 2'b10选择$31.

3. ALU_SRC_MUX

对ALU的第二个输入信号进行选择，0时选择rt寄存器的值作为输入，1时选择Extender的结果作为输入.

6.E\M寄存器

名称	I\O	端口大小	说明
clk	I	1	时钟信号
reset	I	1	同步复位信号
Instr_EM_I	I	31:0	EM指令输入（E级当前指令）
PC_EM_I	I	31:0	EMpc输入（E级当前pc值）
PC_EM_O	O	31:0	EMpc输出（M级当前pc值）
Instr_EM_O	O	31:0	EM指令输出（M级当前指令）
ALURS_EM_I	I	31:0	ALU计算结果(E级)
WD_EM_I	I	31:0	WD结果(E级)
Dst_EM_I	I	31:0	可能写入的寄存器地址(E级)
ALURS_EM_O	O	31:0	ALU计算结果(M级)
WD_EM_O	O	31:0	WD结果(M级)
Dst_EM_O	O	31:0	可能写入的寄存器地址(M级)
Tnew_EM_I	I	2:0	冒险控制Tnew(E级)
Tnew_EM_O	O	2:0	冒险控制Tnew(M级)

7.M级

1. DM

名称	I\O	端口大小	说明
scr	I	1	写数据使能
sel	I	1	RAM工作使能
A	I	4:0	写入地址
D1	I	31:0	写入的数据
D2	O	31:0	输出的数据
clk	I	1	时钟信号
reset	I	1	同步复位信号

8.M\W寄存器

名称	I\O	端口大小	说明
clk	I	1	时钟信号
reset	I	1	同步复位信号
Instr_MW_I	I	31:0	MW指令输入（E级当前指令）
PC_MW_I	I	31:0	MWpc输入（E级当前pc值）
PC_MW_O	O	31:0	MWpc输出（M级当前pc值）
Instr_MW_O	O	31:0	MW指令输出（M级当前指令）
ALURS_MW_I	I	31:0	ALU计算结果(M级)
RD_MW_I	I	31:0	WD结果(M级)
Dst_MW_I	I	31:0	可能写入的寄存器地址(M级)
ALURS_MW_O	O	31:0	ALU计算结果(W级)
RD_MW_O	O	31:0	WD结果(W级)
Dst_MW_O	O	31:0	可能写入的寄存器地址(W级)
Tnew_MW_I	I	2:0	冒险控制Tnew(M级)
Tnew_MW_O	O	2:0	冒险控制Tnew(W级)

9.W级

1.LTC

名称	I\O	端口大小	说明
addr	I	31:0	读取的地址（根据最后两位判断指令）
M_type	I	1:0	Load选择信号
Din	I	31:0	原始数据（1w）
Dout	I	31:0	选择后的数据（lw, lh, lb）

2. WDMUX

对写回GRF的数据进行选择 2'b00选择由ALU直接产生的结果， 2'b01选择从DM中读取的数据，2'b01写入PC值（PC+8）.

10. Controller

对于每一级中均独立存在的Controller,其控制信号的组合逻辑如下

名称	I\O	端口大小	说明
Opcode	I	5:0	opcode字段
Func	I	4:0	function字段

控制信号及对应的指令如下：

Instru	Opcode	RFWr	WRSel	WDSel	ALU_SrcSel	DMWr	ALUOp	NPCOp	ExtOp	M_type
R_type（or and addu subu sll sllv)	000000	1	01	00	0	0	ALUControl	00	x	00
ori	001101	1	00	00	1	0	011	00	01	00
lw	100011	1	00	01	1	0	000	00	00	11
lh	100001	1	00	01	1	0	000	00	00	10
lb	100000	1	00	01	1	0	000	00	00	01
sw	101011	0	x	00	1	1	000	00	00	11
sh	101001	0	x	00	1	1	000	00	00	10
sb	101000	0	x	00	1	1	000	00	00	01
beq	000100	0	x	00	0	0	001	01	x	00
lui	001111	1	00	00	1	0	000	00	10	00
addiu	001001	1	00	00	1	0	000	00	00	00
jal	000011	1	10	10	x	0	000	10	x	00
jr	000000	0	x	11	x	0	000	11	x	00
j	000010	0	x	00	x	x	000	10	x	00

（二）冒险控制

在本次CPU设计中，由于哈佛体系下DM,IM的分离，不存在结构冒险，对于跳转冒险，由于仅包含beq这一由寄存器的值决定是否发生跳转的指令，故将跳转指令的冒险归入数据冒险之中。

使用D级中的ACoder部件，各指令AT信息译码：

指令	使用寄存器编号	写入寄存器编号	Tusers	Tusert	Tnew
addu、add、sub、subu、or、and、nor、xor、sllv、srav、srlv、slt、sltu	rs,rt	rd	1	1	2
sll、sra、srl	rs	rd	5	1	2
load(lw、lh、lb、lhu、lbu、mfc0)	rs	rt	1	5	3
sw(sw、sh、sb、mtc0)	rs,rt	/	1	2	0
ori、addiu、addi、andi、xori、slti、sltiu	rs	rt	1	5	2
lui	/	rt	5	5	2
beq、bgez、bgtz、blez、bltz、bne	rs,rt	/	0	0	0
jr	rs	/	0	0	0
jal	/	$31	5	5	2
jalr	rs	rd	0	5	2
j、eret	/	/	5	5	0
mthi、mtlo	rs	hi/lo	1	0	0
mflo、mfhi	/	rd	5	5	2

二、测试方案

1.原指令正确性检验

借用部分P6的测试程序，对MIPS3C指令集中的指令进行检验，保证在新增异常中断的机制后，原指令执行的正确性。

```
#BM转发测试_rs
ori $t1, $t1, 1
addu $t1, $t1, $t2
nop
addi $t3, $3, 2
add $t4, $t3, $t2
nop
addi $t4, $t4, 2
sub $t5, $t4, $t6
```

```
nop
sllv $t5, $t5, $t1
sra $t6, $t5, 1
nop
srav $t7, $t7, $t3
sltiu $t5, $t7, 123
nop
xori $t6, $t5, 13
nor $t7, $t6, $t5
nop
addiu $t2, $t2, 1
subu $t2, $t2, $t3
nop
lui $t1,  1
nop
lw $t2, 4($0)
addu $t3, $t2, $t1
nop

#BM转发测试_rt
ori $t1, $t1, 1
addu $t1, $t2, $t1
nop
addi $t3, $3, 2
add $t4, $t2, $t3
nop
addi $t4, $t4, 2
sub $t5, $t6, $t4
nop
sllv $t5, $t5, $t1
sra $t6, $t5, 1
nop
srav $t7, $t7, $t3
sltiu $t5, $t7, 123
nop
xori $t6, $t5, 13
nor $t7, $t5, $t6
nop
addiu $t2, $t2, 1
subu $t2, $t3, $t2
nop
lui $t1,  1
addu $0, $t2, $1
subu $t2, $0, $t3
addu $t1, $t1, $t2
nop
nop
lw $t2, 4($0)
addu $t3, $t2, $t1
nop

#BW转发测试_rs
ori $t1, $t1, 1
nop
addu $t1, $t1, $t2
nop
addi $t3, $3, 2
nop
```

```
add $t4, $t3, $t2
nop
addi $t4, $t4, 2
nop
sub $t5, $t4, $t6
nop
sllv $t5, $t5, $t1
nop
sra $t6, $t5, 1
nop
srav $t7, $t7, $t3
nop
sltiu $t5, $t7, 123
nop
xori $t6, $t5, 13
nop
nor $t7, $t6, $t5
nop
addiu $t2, $t2, 1
nop

#BW转发测试_rt
ori $t1, $t1, 1
nop
addu $t1, $t2, $t1
nop
addi $t3, $3, 2
nop
add $t4, $t2, $t3
nop
addi $t4, $t4, 2
nop
sub $t5, $t6, $t4
nop
sllv $t5, $t5, $t1
nop
sra $t6, $t5, 1
nop
srav $t7, $t7, $t3
nop
sltiu $t5, $t7, 123
nop
xori $t6, $t5, 13
nop
nop

#BW与BM转发的优先顺序测试_rs
andi $t3, $t3, 0
andi $t2, $t2, 0
ori $t3, 1
ori $t2, 2
srav $t2, $t2, $t3
slti $t3, $t3, 3
or $t2, $t2, $t3
and $t2, $t3, $t2
sltu $t4, $t2, $t3
slt $t2, $t4, $t3
subu $t5, $t1, $t2
```

```
nop

#BW与BM转发的优先顺序测试_rt
andi $t3, $t3, 0
andi $t2, $t2, 0
ori $t3, 1
ori $t2, 2
srav $t2,$t3, $t2
slti $t3, $t3, 3
or $t2, $t2, $t3
and $t2, $t3, $t2
sltu $t4, $t3, $t2
slt $t2, $t3, $t4
subu $t5, $t2, $t1
nop

# CW指令测试
ori $t2, $t2, 3
sw $t2, 16($0)
lw $t3, 16($0)
sw $t3, 20($0)
and $t2, $t2, $0
addiu $t2, $t2, 4
sw $t2, 0($0)
lw $t2, 0($0)
sw $t3, 0($t2)
ori $t3, $t3, 4
sw $t3, 4($0)
lw $0, 0($t2)
sw $0, 4($0)


# 跳转指令及延迟槽测试
andi $t6, $t6, 0
andi $t7, $t7, 0
ori $t6, $t6, 1
ori $t7, $t7, 1
beq $t6, $t7, label_beq
nop
addi $t1, $t1, 1
label_beq:
addu $t1, $t1, $t1
beq $t1, $t1, nobe_beq
nop
addu $t1, $t1, $t1
nop
nobe_beq:
nop
nop
andi $t6, $t6, 0
andi $t7, $t7, 0
ori $t6, $t6, 1
ori $t7, $t7, 1
blt $t6, $t7, label_blt
nop
addi $t1, $t1, 1
label_blt:
addu $t1, $t1, $t1
```

```
beq $t1, $t1, nobe_blt
nop
addu $t1, $t1, $t1
nop
nobe_blt:

and $t1, $t1, $0
ori $t1, $t1, 100
jal funct
addiu $ra, $ra, 4
addiu $t1, $t1, 2
j end
nop
funct:
sw $t2, 0($0)
jr $ra
nop
end:


# 乘除指令及冲突
ori $t1, $t1, -1
ori $t2, $t2, -2
sw $t2, 0($0)
lw $t3, 0($0)
divu $t1, $t3
mfhi $s4
addu $s3, $s4, $t1
mflo $s3
ori $t6, $t6, 1
div $t1, $t2
mfhi $s4
mflo $s3
mult $t1, $t2
mfhi $s4
mflo $s3
lw $t1, 0($0)
multu $t1, $t2
ori $s2, $s2, 1
ori $t2, $t2, 1
andi $s4, $s4, 0
mfhi $s4
addu $s4, $s4, $s4
mflo $s3
addu $s3, $s3, $4
addi $t2, $t2, 1
mfhi $t7
lw $t3, 0($0)
mult $t3, $3
ori $t5, $t5, 1
mfhi $t4
mflo $t6
mthi $t5
mtlo $t6
mfhi $t7
mflo $t9
```

2. 新增指令检验

对新增的3个指令的正确性进行检验

```
ori $t1, $t1, 1
ori $t2, $t2, 2
ori $t3, $t3, 3
ori $t4, $t4, 4
mtc0 $t1, $14
mtc0 $t2, $13
mfc0 $1, $13
mfc0 $0, $14
mfc0 $1, $14
mtc0 $t3, $14
mtc0 $t1, $12
mtc0 $t2, $13
mfc0 $1, $12
mfc0 $2, $13

ori $ra, $ra, 0x3044
mtc0 $ra, $14
eret
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $t1, $t1, 1
addi $ra, $ra, 0x0004
mtc0 $ra, $14 # 循环使EPC的值每次执行这条指令时+4
eret
```

```
div $1, $2
mult $2, $3
divu $4, $5
```

3.各类异常检验

这里的检验均借用讨论区大佬的P7专用的魔改Mars。

首先写了一个简单的软件处理程序，检验产生异常的原因是否正确， 并使得异常处理后，返回地址为受害指令的下一条指令地址

```
.ktext 0x4180
mfc0 $1, $14
mfc0 $2, $13    #检验产生异常的原因
lui $30, 0x8000
and $2, $2, $30  #对Cause寄存器的值进行BD掩码
beq $2, $30 else # 对BD位是否为1进行分支
nop
addi $1, $1, 4  # 若BD位为0， EPC写入EPC+4
j end
nop
else:
addi $1, $1, 8 #若BD位为1， EPC写入EPC+8
end:
mtc0 $1, $14
mfc0 $2, $13
eret            #处理完异常，直接返回
div $1, $2
mult $2, $3
divu $4, $5    #检验eret后的指令是否不执行
```

其后的异常检验都以此作为handler软件，将异常分类为延迟槽中的异常（special）、ALU计算溢出异常、存取指令异常、未知指令异常，并分别进行进行测试， 将结果与魔改版的Mars进行比对。

```
# 延迟槽指令异常
ori $t1, $t1, 1
beq $t1, $t1, label
sw $t1, 0($t1)
label:
addi $t2, $t2, 1

ori $t1, $t1, 1
bgez $t1, label1
sw $t1, 0($t1)
label1:
addi $t2, $t2, 1

ori $t1, $t1, 1
bltz $t1, label2
sw $t1, 0($t1)
label2:
addi $t2, $t2, 1
```

```
ori $t1, $t1, 1
blez $t1, label3
sw $t1, 0($t1)
label3:
addi $t2, $t2, 1

ori $t1, $t1, 1
j label4
sw $t1, 0($t1)
label4:
addi $t2, $t2, 1

ori $t1, $t1, 1
jal label5
sw $t1, 0($t1)
label5:
addi $t2, $t2, 1

ori $t1, $t1, 1
jal label6
sw $t1, 0($t1)
label6:
addi $t2, $t2, 1
nop
nop
nop


# 地址异常
and $t1, $t1, $0
ori $t1, $t1, 1
lh $t1, 0($t1)
addi $t2, $t2, 1

and $t1, $t1, $0
lui $t1, 0x5678
sw $t1, 0($t1)
addi $t2, $t2, 1

and $t1, $t1, $0
addi $t1, $t1, -1
sw $t1, 5($t1)
addi $t2, $t2, 1

and $t1, $t1, $0
addi $t1, $t1, -1
sw $t1, 5($t1)
addi $t2, $t2, 1

and $t1, $t1, $0
ori $t1, $t1, 2
sw $t1, 0($t1)
addi $t2, $t2, 1

and $t1, $t1, $0
lui $t1, 0x1234
sw $t1, 0($t1)
addi $t2, $t2, 1
```

```
and $t1, $t1, $0
addi $t1, $t1, -1
sw $t1, 5($t1)
addi $t2, $t2, 1

and $t1, $t1, $0
addi $t1, $t1, -1
sb $t1, 6($t1)
addi $t2, $t2, 1

and $t1, $t1, $0
addi $t1, $t1, -1
sh $t1, 7($t1)
addi $t2, $t2, 1

and $t1, $t1, $0
addi $t1, $t1, -1
sw $t1, 5($t1)
addi $t2, $t2, 1

and $t1, $t1, $0
lui $t1, 0x7fff
addi $t1, $t1, 0xffff
lb $t1, 100($t1)

and $t1, $t1, $0
addi $t1, $t1, 0x7f00
lw $t2, 0($t1)
lw $t2, 4($t1)
lw $t2, 8($t1)
lh $t2, 0($t1)
lb $t2, 4($t1)
lhu $t2, 8($t1)
lbu $t2, 12($t1)
lw $t2, 12($t1)

and $t1, $t1, $0
addi $t1, $t1, 0x7f10
lw $t2, 0($t1)
lw $t2, 4($t1)
lw $t2, 8($t1)
lh $t2, 0($t1)
lb $t2, 4($t1)
lhu $t2, 8($t1)
lbu $t2, 12($t1)
lw $t2, 12($t1)
nop
nop
nop

# 计算类指令溢出异常
lui $t1, 0x7fff
addi $t1, $t1, 0xfff2
addi $t1, $t1, 1233
ori $t2, $t2, 123
addi $t1, $t1, 100
addu $t3, $t2, $t1
```

```
add $t3, $t2, $t1
lui $t4, 0x8000
subu $t5, $t2, $t4
sub $t5, $t2, $t4

and $t1, $t1, $0
and $t2, $t2, $0
and $t3, $t3, $0
and $t4, $t4, $0
and $t5, $t5, $0

lui $t1, 0x7fff
addi $t1, $t1, 0xfff6
addi $t1, $t1, 4567
ori $t2, $t2, 234
addi $t1, $t1, 110
addu $t3, $t2, $t1
add $t3, $t2, $t1
lui $t4, 0x8000
addi $t4, $t4, 0x0023
subu $t5, $t2, $t4
sub $t5, $t2, $t4
nop
nop
nop
nop

# 未知指令异常
ori $t1, $t1, 1
ori $t2, $t2, 3
bgezal $t1, label
mul $t2, $t5, $t6
addi $t3, $t3, 1
bltzalr $t1, $t2, $t3
addi $t3, $t1, 1
bltzalr $t1, $t2, $t3
addi $t3, $t3, 1
bgezal $t1, label
mul $t2, $t5, $t6
bgezal $t1, label
mul $t2, $t5, $t6
addi $t3, $t3, 1
addi $t3, $t3, 1
addi $t3, $t3, 1
bgezal $t1, label
mul $t2, $t5, $t6
bgezal $t1, label
mul $t2, $t5, $t6
addi $t3, $t3, 1
addi $t3, $t3, 1
addi $t3, $t3, 1
bgezal $t1, label
mul $t2, $t5, $t6
bgezal $t1, label
mul $t2, $t5, $t6
bgezal $t1, label
mul $t2, $t5, $t6
addi $t3, $t3, 1
```

```
mul $t2, $t5, $t6
mul $t2, $t5, $t6
mul $t2, $t5, $t6
mul $t2, $t5, $t6
addi $t3, $t3, 1
addi $t3, $t3, 1
bgezal $t1, label
mul $t2, $t5, $t6
bgezal $t1, label
mul $t2, $t5, $t6
bgezal $t1, label
```

4.中断检验

使用上述测试程序，通过在tb中通过检测宏观pc产生中断信号，并与架构基本相同的同学的CPU进行比对。

5.与定时器的IO检验

```
.ktext 0x4180
mfc0 $1, $14
mfc0 $2, $13
lui $30, 0x8000
and $2, $2, $30
beq $2, $30 else
nop
addi $1, $1, 4
j end
nop
else:
addi $1, $1, 8
end:
mtc0 $1, $14
mfc0 $2, $13
eret

.text
ori $t1, $t1, 4
ori $t2, $t2, 10
sh $t2, 0x7f00($t1)
sb $t2, 0x7f00($0)
sb $t2, 0x7f00($0)
sw $t2,  0x7f00($t1)
lw $t3,  0x7f00($t1)
ori $t1, $t1, 8
sw $t2, 0x7f00($t1)
ori $t4, $t4, 1
sw $t4, 0x7f00($0)

and $t1, $0, $0
and $t2, $0, $0
and $t3, $0, $0
and $t4, $0, $0
```

```
ori $t1, $t1, 4
ori $t2, $t2, 10
sh $t2, 0x7f10($t1)
sb $t2, 0x7f10($0)
sb $t2, 0x7f10($0)
sw $t2,  0x7f10($t1)
lw $t3,  0x7f10($t1)
ori $t1, $t1, 8
sw $t2, 0x7f10($t1)
ori $t4, $t4, 1
sw $t4, 0x7f10($0)
```

三、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？）

接口是CPU与外界进行交互的信息通道，我们在P6及以前的CPU仅支持从外部一次性读入一段指令序列，执行一定的运算，与外界没有交互能力。而实际使用的CPU在进行正确计算的基础上，要求能够通过软件接口，接受操作系统及软件的控制；通过硬件接口，接受各种输入设备（鼠标、键盘等）的输入信号。在CPU封装完成后，CPU将作为一个整体对外不可见，而仅保留相应的接口以接受控制与信息输入。在本次实验中，我们的MIPS微系统支持了几乎最简单的IO接口（外部中断）作为学习的示例。实际使用的CPU有着复杂的接口与接口控制。

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。

实际的DM应位于内存当中，而位于CPU内部的存储部件一般为L1高速缓存，CPU读写的信息直接与此高速缓存交互，此后该高速缓存再通过其余的多级cache最终实现与DM的交互。

3.BE 部件对所有的外设都是必要的吗?

并不是必要的，比如在本次实验中, DM处在CPU内部的M级中，而在实际中，CPU读写高速缓存没有通过BE，实际的系统桥的作用在于：由于CPU运算频率很高，而外部不同的设备的运转频率一般都小于CPU主频且参差不齐，故而需要对这些部件的输入输出进行控制，而对于信息交互的速度没有较高的要求，此时这些设备将通过Bridge与CPU桥联。对于交互频繁且运转频率可以很高的设备(如高速缓存)，则由于电学限制离CPU越近越好，故而对于这些设备而言，BE不是必要的。

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

详见计时器说明文档

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：

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。

```
.ktext 0x4180
ori $t5, $t5, 9
sw $t5, 0x7f00($0) # 设置写使能为1
eret

.text
ori $t1, $t1, 4
ori $t2, $t2, 10
sw $t2,  0x7f00($t1) # 设定初值寄存器
lw $t3,  0x7f00($t1)
ori $t4, $t4, 9
sw $t4, 0x7f00($0)   # 初始化定时器的模式
```

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?

当键盘或鼠标的一个键被按下时，鼠标或键盘微处理器便根据其位置，将字符信号转换成二进制码，通过总线传给主机和显示器。如果CPU正在进行其它的工作，就先将键入的内容送往内存中的键盘或鼠标的缓冲区。在需要读取数据时，将并不断通过polling的方式尝试读取其中的数据：

- **Input:** Read from keyboard into `$v0`

```
                lui   $t0, 0xffff # ffff0000
Waitloop:       lw    $t1, 0($t0) # control reg
                andi  $t1,$t1,0x1
                beq   $t1,$zero, Waitloop
                lw    $v0, 4($t0) # data reg
```

- **Output:** Write to display from `$a0`

```
                lui   $t0, 0xffff # ffff0000
Waitloop:       lw    $t1, 8($t0) # control reg
                andi  $t1,$t1,0x1
                beq   $t1,$zero, Waitloop
                sw    $a0,12($t0) # data reg
```

当该循环结束时时，$v0中将保存着键盘或鼠标中存储的信息传入CPU。