计算机组成课程设计 P7 实验报告

一、 CP0 设计

CP0 端口描述		
端口名	端口方向	端口描述
IR_D[31:0]	I	D 级 IR
IR_M[31:0]	I	M 级 IR
IR_W[31:0]	I	W 级 IR
DIn[31:0]	I	mtc0 的数据
PC[31:0]	I	异常中断 PC 值
EXCcode[6:2]	I	异常中断代码
HWInt[15:10]	I	6 个外设的命中
EXLSET	I	中断异常标志
clk	I	时钟信号
reset	I	复位信号
IntReq	O	中断异常请求标志
EPC[31:0]	O	EPC 寄存器的值
Dout[31:0]	O	mfc0 输出的值

CP0 功能描述		
序号	功能名称	功能描述
1	产生中断异常请求	当外设有中断请求且非中断屏蔽时，或者有异常时，产生中断异常请求
2	写 SR\CAUSE\EPC\PRID	当写使能有效，且为对应寄存器号时，将 DIn 写入相应寄存器
3	读 SR\CAUSE\EPC\PRID	当 mfc0，且为对应寄存器号时，将寄存器值输出值 Dout
4	置 exl	当 EXLSET 为 1 时，将 exl 置 0，不再允许异常中断

二、 桥与 IO

BRIDGE		
		BRIDGE 端口描述
端口名	端口方向	端口描述
PrBE[3:0]	I	CPU 写外设的字节写使能
Addr[31:0]	I	CPU 写外设的地址
Dout[31:0]	I	CPU 写外设的数据
PrWE	I	CPU 写外设的全局写使能
PrRD[31:0]	O	外设的输出值
HWInt[7:2]	O	外设的中断命中
RD_timer0[31:0]	I	外设 0 的回写数据
RD_timer1[31:0]	I	外设 1 的回写数据
IRQ_timer0	I	外设 0 的中断请求
IRQ_timer1	I	外设 1 的中断请求
Addr_time[31:0]	O	写外设的地址
WE_timer0	O	写外设 0 的写使能
WE_timer1	O	写外设 1 的写使能
Dout_time[31:0]	O	输入到外设的数据

		BRIDGE 功能描述
序号	功能名称	功能描述
1	写外设	根据地址范围，将 CPU 的数据写入对应的外设
2	写 CPU	将外设的数据筛选并回写至 CPU
3	传递中断请求	将外设的中断请求传递给 CPU

三、 测试程序

```
.text

ori $1,$0,0x1234

lui $2,0x5678

or $3,$2,$1
```

```
#正确性

#mtc0 mfc0

mtc0 $1,$13

mtc0 $1,$12

mtc0 $1,$14

mtc0 $1,$15

mfc0 $4,$13

mfc0 $5,$12

mfc0 $6,$14

mfc0 $7,$15

#设备

ori $1,$0,0x7f00

ori $2,$0,0x7f10

sw $3,0($1)

sw $3,4($1)

sw $4,0($2)

sw $4,4($2)

lw $3,0($1)

lw $3,4($1)

lw $3,8($1)

lw $4,0($2)

lw $4,4($2)

lw $4,8($2)

#转发暂停
```

```
mfc0 $1,$13

beq $1,$1,label1  #beq-mfc0

nop

lui $12,0x1234

label1:

lui $11,0x1234

beq $1,$0,label1

nop

mfc0 $2,$12

nop

beq $2,$0,label1 #beq-x-mfc0

nop

lui $12,0x1234

mfc0 $1,$13

bgez $1,label2  #bgez-mfc0

nop

lui $12,0x1234

label2:

lui $11,0x1234

mfc0 $1,$14

nop

blez $1,label3 #blez-x-mfc0

nop

lui $12,0x1234
```

```
label3:

lui $11,0x1234

mfc0 $1,$13

nop

nop

blez $1,label4 #blez-x-x-mfc0

nop

lui $12,0x1234

label4:

lui $11,0x1234

ori $1,$0,0x30ec

mtc0 $1,$14

mfc0 $2,$14  #mfc0-mtc0

jr $2   #jr-mfc0

nop

nop

lui $12,0x1234

ori $1,$0,0x310c

mtc0 $1,$14

mfc0 $2,$14

nop

jr $2   #jr-x-mfc0

nop

nop
```

```
lui $12,0x1234

ori $1,$0,0x3130

mtc0 $1,$14

mfc0 $2,$14

nop

nop

jr $2   #jr-x-x-mfc0

nop

nop

lui $12,0x1234

ori $1,$0,0x3150

mtc0 $1,$14

mfc0 $2,$14

jalr $3,$2   #jalr-mfc0

nop

nop

nop

lui $12,0x1234

ori $1,$0,0x3170

mtc0 $1,$14

mfc0 $2,$14

nop

jalr $3,$2   #jalr-x-mfc0

nop
```

```
nop

lui $12,0x1234

ori $1,$0,0x3198

mtc0 $1,$14

mfc0 $2,$14

nop

nop

jalr $3,$2   #jalr-x-x-mfc0

nop

nop

nop

lui $12,0x1234

mfc0 $1,$14

addu $3,$1,$2 #cal-mfc0

subu $3,$1,$2 #cal-x-mfc0

addu $3,$1,$2 #cal-x-x-mfc0

ori $1,$0,4

mtc0 $1,$14

mfc0 $1,$14

sw $2,0($1) #sw-mfc0

sw $3,4($1) #sw-x-mfc0

sw $12,8($1) #sw-x-x-mfc0

mfc0 $1,$14

lw $2,0($1) #lw-mfc0
```

```
lw $2,4($1) #lw-x-mfc0

lw $2,8($1) #lw-x-x-mfc0

mfc0 $1,$13

mult $1,$2 #mult-mfc0

mult $1,$3 #mult-x-mfc0

mult $1,$2 #mult-x-x-mfc0

mfc0 $2,$14

beq $1,$2 label5 #beq-mfc0

nop

lui $12,0x1234

label5:

lui $11,0x1234

mfc0 $2,$12

nop

beq $0,$2,label6 #beq-x-mfc0

nop

lui $12,0x1234

label6:

lui $11,0x1234

mfc0 $3,$12

nop

nop

beq $1,$3,label7 #beq-x-x-mfc0

nop
```

```
lui $12,0x1234

label7:

lui $11,0x1234

mfc0 $1,$12

addu $3,$2,$1 #cal-mfc0

addu $3,$2,$1 #cal-x-mfc0

addu $3,$2,$1 #cal-x-x-mfc0

mfc0 $2,$14

mult $1,$2 #mult-mfc0

mult $1,$2 #mult-x-mfc0

mult $1,$2 #mult-x-x-mfc0

mfc0 $1,$13

sw $1,0($0) #sw-mfc0

sw $1,4($0) #sw-x-mfc0

sw $1,8($0) #sw-x-x-mfc0

mfc0 $2,$13

mtc0 $2,$14 #mtc0-mfc0

mtc0 $2,$14 #mtc0-x-mfc0

mtc0 $2,$14 #mtc0-x-x-mfc0
```

四、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”？**

硬件接口（hardware interface）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。接口（软件类接口）是指对协定进行定义的引用类型。其他类型实现接口，以保证它们支持某些操作。接口指定必须由类提供的成员或实现它的其他接口。与类相似，接口可以包含方法、属性、索引器和事件作为成员。

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。**

DM 应该与 CPU 分离。DM 需要大量存储空间，而 CPU 需要复杂的运算逻辑，将两者分离可以降低实现难度。

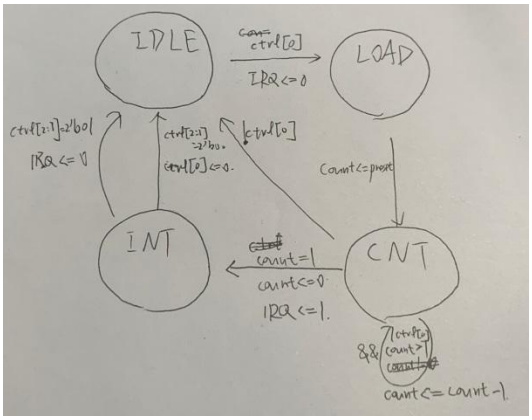
3. **BE 部件对所有的外设都是必要的吗？**

不是，对于不需要以字节为单位来存储的外设是不需要的，如 timer。

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**

两种中断模式共同点：计数器使能有效时，每个周期值减一；减至零时产生中断；重置时恢复为 preset 寄存器中的初始值。

不同点：模式 0 进入中断后将计数器使能置 0，直到计数器使能置 1 时结束中断；模式 1 进入中断后下个周期结束中断，再下一个周期恢复初始值。



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。）

主程序设计：

exception_handler.asm | main_exception.asm

```
1  ori $s0,0x7F10
2  ori $t1,10
3  sw $t1,4($s0)   #preset = 10
4  ori $t2,9
5  sw $t2,0($s0)   #ctrl[3:0] = 1001
6  ori $t3,0xfa01  #IM[7:2]=6'b111111, EXL=0, IE=1
7  mtc0 $t3,$12    #write to SR
8  nop
9  nop
10 nop
11 jal loop
12 nop
13 loop: jr $ra
14 nop
15
```

Handler 设计：

exception_handler.asm* | main_exception.asm

```
.ktext 0x4180
ori $t0,0x80000200      #value of Cause when BD=1 and interrupt
ori $t1,0x00000200      #value of Cause when BD=0 and interrupt
mfc0 $s0,$13
beq $s0,$t0,handler
nop
beq $s0,$t1,handler
nop
j end
handler: ori $s0,0x7f10 #if cause==t0 or t1, execute handler
ori $t2,9
sw $t2,0($s0)   #set ctrl[3:0] to 4'b1001, begin counting
end: 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？

键盘鼠标输入通过中断请求进行输入。敲击键盘/移动鼠标时发出中断信号，经过中断控制器到达 CPU，CPU 根据不同中断号执行不同的中断响应程序，并进行相应的 IO 操作，把读取的信息写入寄存器，最后存入内存。

根据策略矩阵构造暂停条件的一般性方法

rs策略矩阵

T_{use} \ T_{new}	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	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

```
Stall_RS0_e1 = . . .
Stall_RS0_e2 = . . .
Stall_RS0_m1 = . . .
Stall_RS1_e2 = . . .

Stall_RS = Stall_RS0_e1 |
           Stall_RS0_e2 |
           . . .
```

+

rt策略矩阵

T_{use} \ T_{new}	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
	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

```
Stall_RT0_e1 = . . .
Stall_RT0_e2 = . . .
Stall_RT0_m1 = . . .
Stall_RT1_e2 = . . .

Stall_RT = Stall_RT0_e1 |
           Stall_RT0_e2 |
           . . .
```

↓

```
Stall = Stall_RS |
        Stall_RT
```

Q:
如何表示T_{new}与T_{use}？

北京航空航天大学计算机学院

构造指令集的T_{use}

- 思路：结合流水线架构，逐条指令构造T_{use}及T_{new}
- T_{use}注意事项
  - 1）只关注每条指令的操作语义
  - 2）指令可能有2个不同的T_{use}，如sw
  - 3）指令集或流水线架构的变化，均可能导致T_{use}变化
    - 例如：流水线从5级变为6级且第5级为访存，则sw的rt将会延后1级被使用，故rt的T_{use}会变为{0,1,2,3}

	T_{use}	
	rs	rt
add	1	1
sub	1	1
andi	1	
ori	1	
lw	1	
sw	1	2
beq	0	0
jr	0	
	{0,1}	{0,1,2}

构造指令集的T_{new}

- 思路：结合流水线架构，逐条指令构造T_{use}及T_{new}
- T_{new}注意事项：
  - 一旦减为0，则不再继续减少！
    - 0：有效结果已经产生了
    - 非0：有效结果尚未产生
- 为了便于分析，用产生结果的功能部件来代表指令
  - 例如，ALU可以代表所有的计算类指令

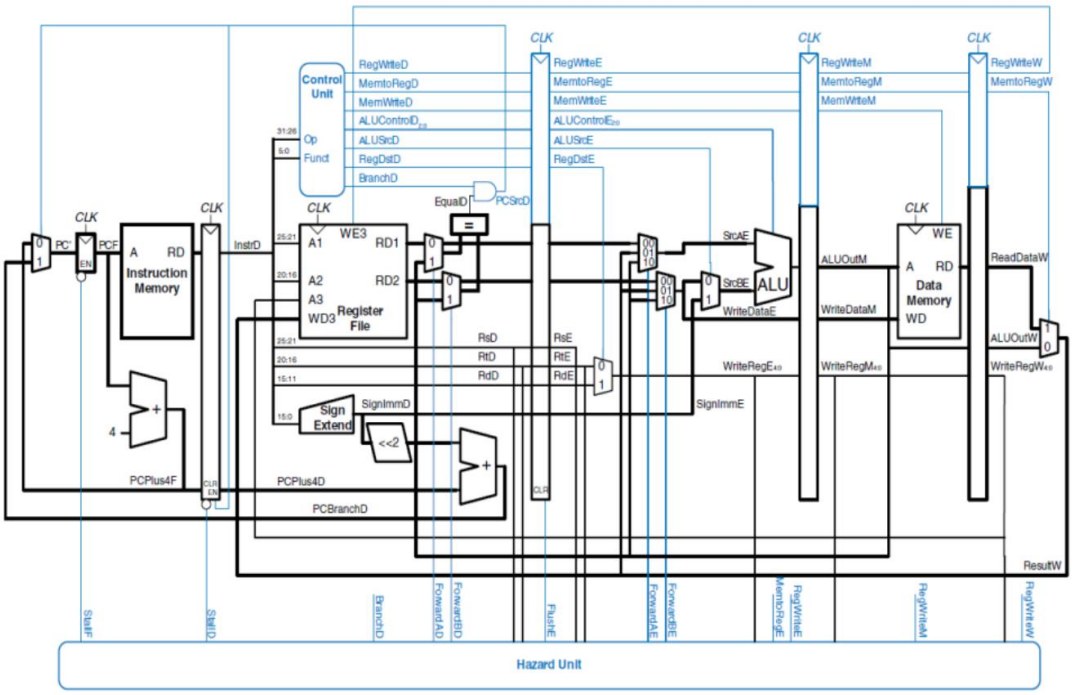
指令	功能部件	T_{new}		
		E	M	W
add	ALU	1	0	0
sub	ALU	1	0	0
andi	ALU	1	0	0
ori	ALU	1	0	0
lw	DM	2	1	0
sw				
beq				
jal	PC	0	0	0

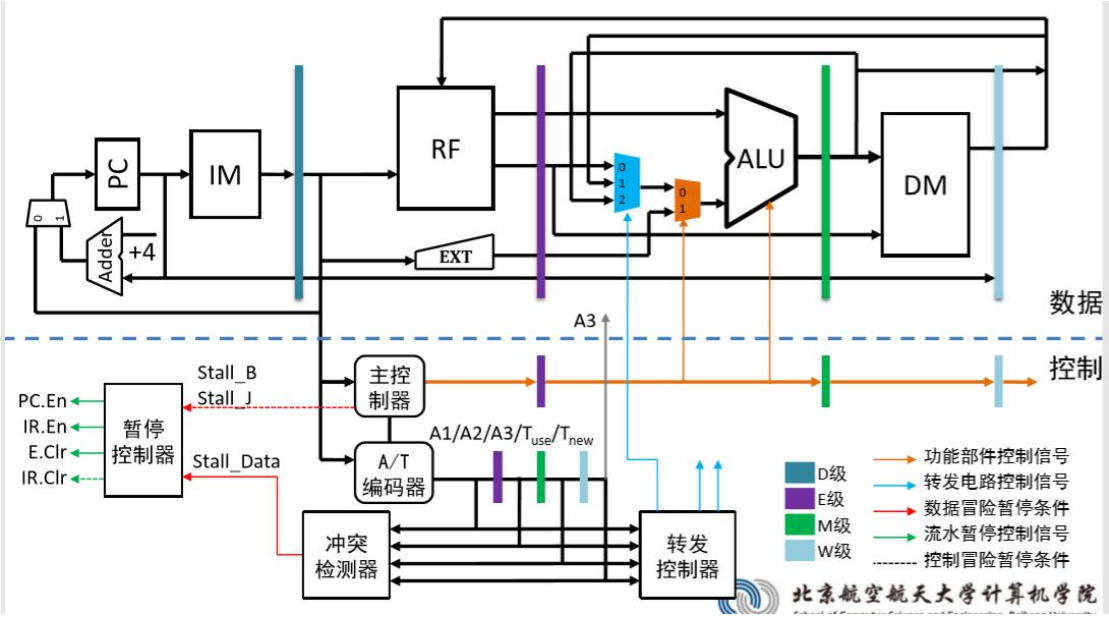
产生结果的功能部件

E			M			W		
ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
1	2	0	0	1	0	0	0	0

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

	Tuse		Tnew	功能部件
	rs	rt		
addu	1	1	1	ALU
subu	1	1	1	ALU
ori	1		1	ALU
lui	1	1	1	ALU
sw	1	2		
lw	1		2	DM
j				
jal			0	PC
jr	0			
beq	0	0		


CLK
Control Unit
RegWriteD
MemtoRegD
MemWriteD
ALUControlD
ALUSrcD
RegDstD
BranchD
Op
Funct
EqualD
PCSrcD
PC'
PCF
Instruction Memory
InstrD
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
RsD
RtD
RdD
Sign Extend
SignImmD
<<2
PCPlus4F
PCPlus4D
PCBranchD
RegWriteE
MemtoRegE
MemWriteE
ALUControlE
ALUSrcE
RegDstE
RsE
RtE
RdE
SignImmE
SrcAE
SrcBE
ALU
WriteDataE
WriteRegE
RegWriteM
MemtoRegM
MemWriteM
ALUOutM
WriteDataM
WriteRegM
WE
Data Memory
WD
RegWriteW
MemtoRegW
ReadDataW
ALUOutW
WriteRegW
ResultW
StallF
StallD
BranchD
ForwardAD
ForwardBD
FlushE
ForwardAE
ForwardBE
MemtoRegE
RegWriteE
RegWriteM
RegWriteW
Hazard Unit


PC
IM
Adder
+4
RF
EXT
ALU
DM
数据
控制
A3
主控制器
Stall_B
Stall_J
PC.En
IR.En
E.Clr
IR.Clr
暂停控制器
Stall_Data
A/T 编码器
A1/A2/A3/Tuse/Tnew
冲突检测器
转发控制器
D级
E级
M级
W级
功能部件控制信号
转发电路控制信号
数据冒险暂停条件
流水暂停控制信号
控制冒险暂停条件
北京航空航天大学计算机学院