思考题

1. 什么是硬件软件接口

   指令集体系结构就是低层次软件和硬件之间的抽象接口，他可让程序员在不直接操纵硬件的前提下，就编写出可以让硬件实现想要达到的目的的正确的程序，包括指令，寄存器，访存和 IO。
2. 在现代计算机中，DM 和其他外设一样应该位于 CPU 外部（~~不然我上个月咋安的内存条~~），通过总线控制器连接到 CPU 供其访问。
3. 不是。因为不是所有外设都要求支持按字节访问
4. .text
   ori $s0,9#低四位分别是 1001
   ori $s1,1000
   sw $s0,0x7f10($0)
   sw $s0,0x7014($0)
   loop:#开始死循环
   beq $s0,$s0,loop
   nop

   .ktext 0x0004180
   ori $s0,9
   sw $s0,0x7f10($0)
   eret
5. 键盘在按键被按下或者抬起时都会发生中断，CPU 读取对应键盘端口的寄存器来得知发生了什么事件，通常如果一个按键被按下 CPU 读取到的扫描码是 X 那么该按键抬起时 CPU 读到的就是 X+0X80　　鼠标同理

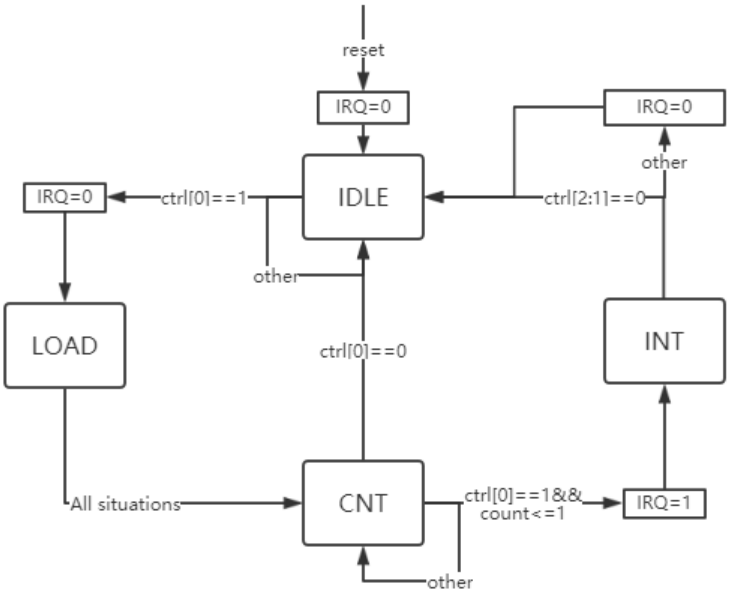
P6 CPU 文档

~~（P6 工程化写 bug 报告）~~

数据通路设计：

	地址或地址范围	备注
数据存储器	0x0000_0000至0x0000_2FFF	
指令存储器	0x0000_3000至0x0000_4FFF	
PC初始值	0x0000_3000	
Exception Handler入口地址	0x0000_4180	
定时器寄存器地址	0x0000_7F00至0x0000_7F0B 0x0000_7F10至0x0000_7F1B	定时器0的3个寄存器 定时器1的3个寄存器

6. 计时器的状态转移图如下



- 上图只表明状态的转换，对于定时器内部寄存器值的改变还需要参考其余说明
- 上图中带有IRQ的方框表示在状态转换过程中IRQ值的变化，并不代表状态，箭头上的表达式代表状态转换的条件
- IRQ的值表示的状态机内部的值，在屏蔽中断状态下不论状态机内部IRQ为何值，对外表现出IRQ恒为0
- 外部写入的优先级大于状态机自身状态转换的优先级，换一种说法，在外部向计时器写入数据的周期中，计时器的状态寄存器不会发生变化，也不会进行count自减的操作

SR寄存器

IM[7:2]：6位中断屏蔽位，分别对应6个外部中断

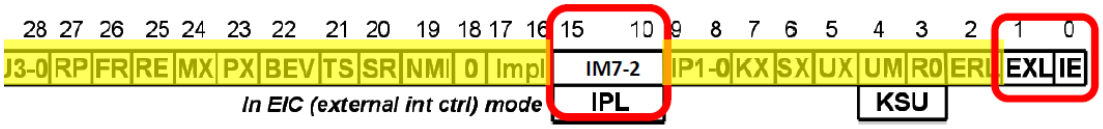
- 1-允许中断，0-禁止中断

IE：全局中断使能

- 1-允许中断；0-禁止中断

EXL：异常级

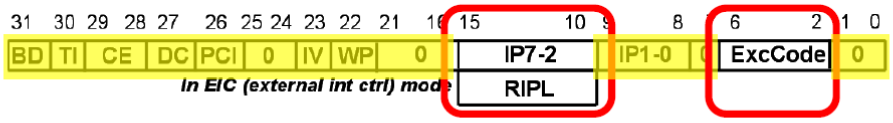
- 1-进入异常，不允许再中断；0-允许中断
- 注意：重入（在中断程序中仍然允许再次进行中断）需要OS的配合，重点是堆栈



CAUSE寄存器

- IP[7:2]：6位待决的中断位，分别对应6个外部中断
  - 记录当前哪些硬件中断正在有效
  - 1-有中断；0-无中断
- ExcCode[6:2]：异常编码，记录当前发生的是什么异常
  - 共计32种
  - 与课程相关的主要异常类型

ExcCode	助记符	描述
0	Int	中断
10	RI	不识别(非法)指令
12	Ov	算数指令导致的异常(如add)



正常各流水级部件

1. IFU

接口名称	类型	描述
clk	I	时钟信号
clr	I	复位信号
branch	I	跳转信号，为 1 时有效
npcout[31:0]	I	输入在跳转指令时需要跳转的地址
stall	I	暂停信号
instr	O	输出取出的指令
pc8	O	输出 pc+8 的值
intreq	I	中断信号

序号	功能名称	功能描述
1	复位	当 clr 信号有效时 pc 被置为 0x00003000
2.	取指令	取出 pc 对应地址里的指令
3	生成下一指令地址	若 branch 有效，下一指令地址为 npc 否则为 pc+4
4	stall 响应	若 stall 有效，锁定 pc 寄存器使其值不再改变
5.	intreq 响应	若 intreq 有效跳转至 npc（此时是 0x00004180）

2. grf

接口名	类型	描述
clk	I	时钟信号
reset	I	异步复位信号，1 为有效
we	I	写使能信号，1 为有效
a1[4:0]	I	RD1 端口输出的寄存器的编号
a2[4:0]	I	RD2 端口输出的寄存器的编号
a3[4:0]	I	要写入的寄存器的编号
wd	I	要写入选定寄存器的数据
rd1	O	输出 A1 端口选中寄存器的值(进行转发)
rd2	O	输出 A2 端口选中寄存器的值（进行转发）

功能描述

序号	功能名称	功能描述
1	复位	reset 信号有效时，所有寄存器的值置为 0
2	读寄存器	RD1,RD2 输出 A1,A2 选中寄存器的值
3	写寄存器	we 信号有效时 A3 选中的寄存器的值被写为 WD 的值

3. npc

接口名	类型	描述
npcop[1:0]	I	功能选择 00 依据 Imm 字段计算下一指令地址 01 依据指令后 26 位与 pc+4 高四位计算地址 02 输出 rd1 中的地址 03 0x00004180 04 eret 执行时需返回的地址
pc4	I	输入 pc+4

extout	I	输入 imm 进行 16 位扩展后的信号
rd1	I	输入 gpr 中 a1 选定寄存器的值
intsr	I	输入当前正在执行的指令
npcout	O	输出计算出的下一指令
intreq	I	中断信号，若有效 npcot 为 0x4180
epc	I	eret 跳转地址

功能描述

序号	功能名称	功能描述
1	实现立即数跳转	依据 Imm 字段计算下一指令地址
2	实现 j 类型跳转	依据指令后 26 位与 pc+4 高四位计算地址
3	实现寄存器跳转	输出 rd1 中的地址
4	实现 handler 跳转	输出 0x4180
5	实现 eret 跳转	输出 epc 的值（epc 值已经过显式转发）

4. ext

接口名	类型	描述
imm[15:0]	I	输入 16 位待扩展立即数
extout[31:0]	O	输出扩展后 32 位数据
extop[1:0]	I	功能选择 00 无符号扩展，01 有符号扩展，02Lui 扩展

功能说明

序号	功能名称	功能描述
1	无符号扩展	高位补 16 个 0
2	有符号扩展	高位补 16 个符号位
3	Lui 扩展	低位补 16 个 0

5. cmp

接口名	类型	描述
[31:0]a	I	输入待比较数 a
[31:0]	I	输入待比较数 b
[31:0]zero32	O	输出 zeroextent(a==b)

功能说明

序号	功能名称	功能描述
1	相等判断	若 a==b 时有效

6. alu

接口名	类型	描述
a[31:0]	I	输入 alu 的第一个操作数
b[31:0]	I	输入 alu 的第二个操作数
aluop[1:0]	I	alu 功能选择 00 输出 b，01 为加运算，10 为减运算，11 为或运算，
aluout[31:0]	O	输出运算结果
overflow	O	输出有无溢出

功能描述

序号	功能名称	功能描述
1	加运算	输出 a1+a2
2	减运算	输出 a1-a2
3	或运算	输出 a1|a2
4	输出 b	为提高 lui 等指令的性能并降低复杂度而预留

7. dm

接口名	类型	描述
addr[31:0]	I	DM 输入的地址
wd[31:0]	I	DM 要写入的数据
clk	I	时钟信号
clr	I	异步复位信号
we	I	写使能信号，1 为有效，将 A 端口选中的字写为 INPUT 中数据
dmout[31:0]	O	输出 addr 端口选中的字的值

功能描述

序号	功能名称	功能描述
1	复位	clr 信号有效时，所有主存的值置为 0
2	读主存	输出 addr 端口选中的字的值
3	写主存	we 有效时将 addr 端口选中的字写为 wd 中数据

异常处理单元

1. Exc_i

接口名	方向	描述
instr_d	I	d 级指令
pc8_i	I	I 级指令的 pc8 值
bd_i	O	branch delayed 信号，若为延迟槽指令则为 1
exccode_i	O	传递异常代码

功能说明

序号	功能名称	描述
1	判断取指异常	若 pc 越界或不对齐报 adel 异常
2	延迟槽指令判断	若为延迟槽指令则置 1

2. Exc_d

接口名	方向	描述
instr_d	I	输入 d 级指令
exccode_i	I	输入 i 级传递的 exccode
bd_i	I	输入是否延迟槽指令
exccode_d	O	输出 exccode
bd_d	O	输出 bd

功能说明

序号	功能名称	描述
1	检查不识别指令	若指令未被识别则报 ri 异常
2	传递前级异常	若本级无异常则传递前级 exccode

3. exc_e

接口名	方向	描述

instr_e	I	输入 e 级指令
overflow	I	输入 overflow 判断结果
bd_d	I	输入前级 bd
exccode_d	I	输入前级 exccode
bd_e	O	输出 bd
exccode_e	O	输出 exccode

功能说明

序号	功能名称	描述
1	检查算数溢出	若 add sub 算数溢出报 ov 异常
2	传递前级异常	若本级无异常则传递前级 exccode

4. exc_m

接口名	方向	描述
instr_m	I	输入 m 级指令
ao_m	I	输入 m 级地址总线
exccode_e	I	输入前级 exccode
bd_e	I	输入前级 bd
exccode_m	O	输出 exccode
bd_m	O	输出 bd

功能说明

序号	功能名称	描述
1	判断读/写异常	若读写地址不对齐/地址越界/写不该写的地址 报对应的异常
2	传递前级异常	若本级无异常则传递前级 exccode

协处理器单元 0

接口名	方向	描述
clk	I	时钟信号
clr	I	复位信号
a1	I	读寄存器编号
a2	I	写寄存器编号
wdbus	I	写入数据
pc8_m	I	写入 m 级异常/中断受害指令地址
bdin	I	写入 m 级异常/中断受害指令是否为延迟槽指令
hwint	I	输入 6 个外部设备中断信号
exccodein	I	输入 m 级异常/中断编号
we	I	CP0 寄存器写使能
exlclr	I	关中断使能
cp0_rd	O	cp0 读出数据
cp0_epc	O	epc 寄存器的值
intreq	O	中断信号

功能说明

序号	功能名称	描述
1	复位	若 clr 信号有效实施复位操作
2	外部中断响应	若外部中断到来且全局允许中断且不在中断时

		保存 epc 值为 pc_m(pc_m-4 若为延迟槽) 保存 cause 寄存器的值 exl 置位为 1 cause exccode 段置位 00000
3	异常响应	若 m 级指令异常且全局允许中断且不在中断时 保存 epc 值为 pc_m(pc_m-4 若为延迟槽) 保存 cause 寄存器的值 exl 置位为 1
4	关中断信号	若有效 exl 置位 0
5	写 CP0 寄存器	若写使能有效，写入对应编号的寄存器
6	读 CP0 寄存器	输出对应编号的寄存器值

Bridge 设计

接口名	方向	描述
we	I	写使能信号
addrbus	I	地址总线
wdbus	I	数据总线
bridge_rd	O	读出数据
hwint	O	输出中断位
t_addr	O	向定时器输出地址总线
t_we	O	定时器写使能
t_rd0	I	定时器 0 读出数据
t_rd1	I	定时器 1 读出数据
t_int0	I	定时器 0 中断信号
t_int1	I	定时器 1 中断信号

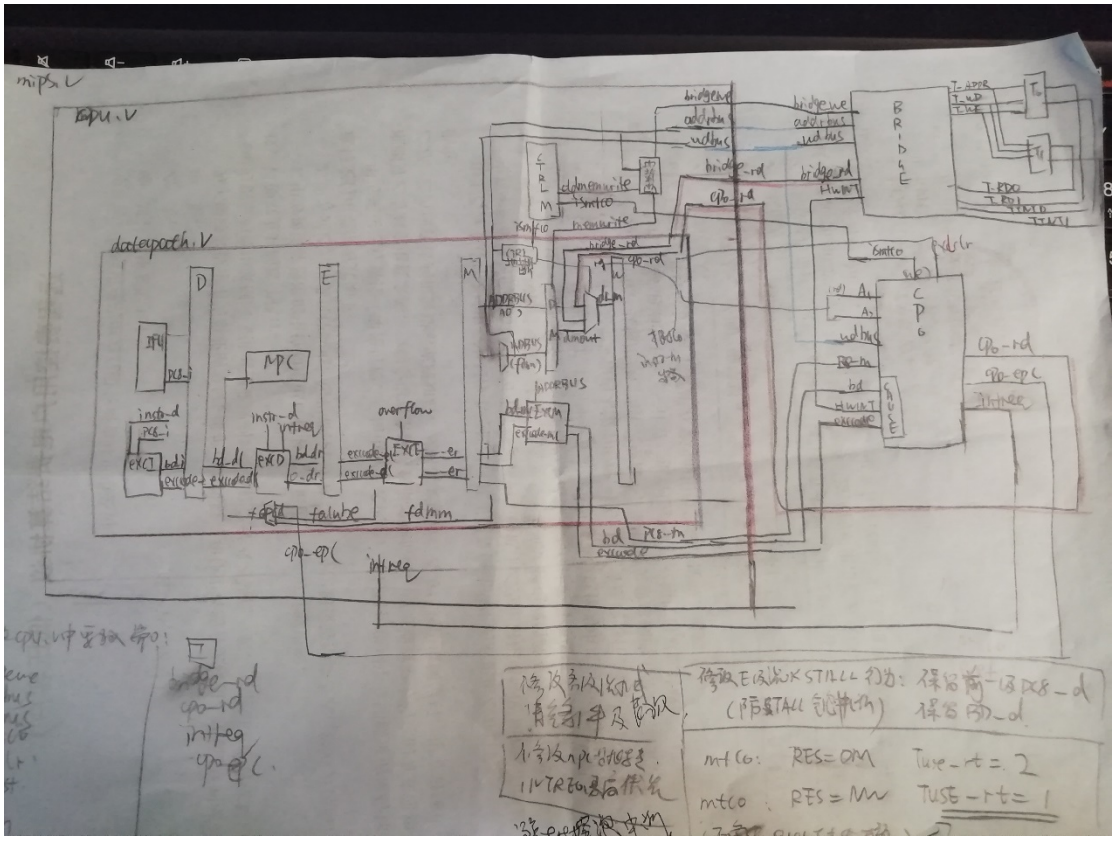
各级流水线寄存器

```
module Dpipe (clk,clr,en,npcout,iseret,
              instr_i,pc8_i,bd_i,exccode_i,
              instr_d,pc8_d,bd_d,exccode_d);
```

```
3  module Epipe(clk,clr,stall,
4  res_d,instr_d,a1_d,a2_d,a3_d,v1_d,v2_d,oth_d,pc8_d,bd_d,exccode_d,
5  res_e,instr_e,a1_e,a2_e,a3_e,v1_e,v2_e,oth_e,pc8_e,bd_e,exccode_e);
```

```
3  module Mpipe(clk,clr,
4  res_e,instr_e,a2_e,a3_e,v2_e,ao_e,pc8_e,bd_e,exccode_e,
5  res_m,instr_m,a2_m,a3_m,v2_m,ao_m,pc8_m,bd_m,exccode_m);
```

DATAPATH 及 CP0，BRIDGE,TIMER 缩略图



控制器设计：

控制器部分基本沿用 P5 设计

1.常规控制器设计

	addu	subu	ori	lw	sw	beq	lui	jal	jr	j
branch	0	0	0	0	0	1	0	1	1	1
regwrite	1	1	1	1	0	0	1	1	0	0
memwrite	0	0	0	0	1	0	0	0	0	0
alusrc	0	0	1	1	1	0	0	0	0	0
npcop	0	0	0	0	0	0	0	1	2	1
extop	0	0	0	1	1	1	2	0	0	0
aluop	1	2	3	1	1	0	0	0	0	0
regdst	0	0	1	1	1	0	1	2	0	0
memtoreg	0	0	0	1	1	0	2	3	0	0
otherctrl	0	0	0	0	0	0	0	0	0	0

新增另外两个信号：ismfc0,ismtc0

定义如下：

```
assign ismfc0=(op==`cop0&&rs==`mfc0_rs);
assign ismtc0=(op==`cop0&&rs==`mtc0_rs);
```

2.AT 控制器

指令	tuse_rs	tuse_rt	res
addu	1	1	alu
subu	1	1	alu

ori	1		alu
lw	1		dm
sw	1	2	nw
beq	0	0	nw
jr	0		nw
jal			pc
j			nw
lui			ext
mfc0			dm
mtc0		***1（以此保证 eret 不会引入新的暂停）***	nw
eret	单独转发	单独转发	nw

STALL 控制器：无任何修改

```
assign stall_rs0le=(tuse_rs==0)&&(res_e==`alu)&&(a1_d==a3_e)&&(a1_d!=0);
assign stall_rs02e=(tuse_rs==0)&&(res_e==`dm)&&(a1_d==a3_e)&&(a1_d!=0);
assign stall_rs12e=(tuse_rs==1)&&(res_e==`dm)&&(a1_d==a3_e)&&(a1_d!=0);
assign stall_rs0lm=(tuse_rs==0)&&(res_m==`dm)&&(a1_d==a3_m)&&(a1_d!=0);
assign stall_rs=stall_rs0le|stall_rs02e|stall_rs12e|stall_rs0lm;

assign stall_rt0le=(tuse_rt==0)&&(res_e==`alu)&&(a2_d==a3_e)&&(a2_d!=0);
assign stall_rt02e=(tuse_rt==0)&&(res_e==`dm)&&(a2_d==a3_e)&&(a2_d!=0);
assign stall_rt12e=(tuse_rt==1)&&(res_e==`dm)&&(a2_d==a3_e)&&(a2_d!=0);
assign stall_rt0lm=(tuse_rt==0)&&(res_m==`dm)&&(a2_d==a3_m)&&(a2_d!=0);
assign stall_rt=stall_rt0le|stall_rt02e|stall_rt12e|stall_rt0lm;

assign stall=stall_rs|stall_rt;
```

转发控制器：无任何修改

```
31  assign fv1dctrl=(res_e==`pc && a1_d==a3_e && a1_d!=0)?`fd_pc8_e:
32                  (res_e==`other && a1_d==a3_e &&a1_d!=0)?`fd_oth_e:
33                  (res_m==`alu && a1_d==a3_m && a1_d!=0)?`fd_ao_m:
34                  (res_m==`pc && a1_d==a3_m && a1_d!=0)?`fd_pc8_m:
35                  `fd_rd;
36  assign fv2dctrl=(res_e==`pc && a2_d==a3_e && a2_d!=0)?`fd_pc8_e:
37                  (res_e==`other && a2_d==a3_e &&a2_d!=0)?`fd_oth_e:
38                  (res_m==`alu && a2_d==a3_m && a2_d!=0)?`fd_ao_m:
39                  (res_m==`pc && a2_d==a3_m && a2_d!=0)?`fd_pc8_m:
40                  `fd_rd;
41  assign faluaectrl=(res_m==`pc &&a1_e==a3_m&&a1_e!=0)?`falue_pc8_m:
42                    (res_m==`alu &&a1_e==a3_m&&a1_e!=0)?`falue_ao_m:
43                    (res_w!=`nw && a1_e==a3_w&&a1_e!=0)?`falue_wd_w:
44                    `falue_v;
45  assign falubectrl=(res_m==`pc &&a2_e==a3_m&&a2_e!=0)?`falue_pc8_m:
46                    (res_m==`alu &&a2_e==a3_m&&a2_e!=0)?`falue_ao_m:
47                    (res_w!=`nw && a2_e==a3_w&&a2_e!=0)?`falue_wd_w:
48                    `falue_v;
49  assign fdmmctrl=(res_w!=`nw && a2_m==a3_w &&a2_m!=0)?1:0;
```

（epc 的转发在 datapath 中独立完成，不守 forward 控制器的控制）

测试程序

测试程序 1：测试了取地址异常和 epc 的独立转发，mtc0 与 mfc0 部分的转发

.text

ori $s0,0x3005

```
jr $s0
ori $t0,1
ori $t1,1
ori $t2,1
ori $t3,1
ori $t4,1

.ktext 0x4180
ori $a1,3
mfc0 $a0,$14
add $a0,$a0,$a1
mtc0 $a0,$14
eret
```

测试程序 2 测试了算数溢出异常

```
.text
lui $s0,0x7fff
add $s1,$s0,$s0
ori $t0,1

.ktext 0x4180
ori $a1,4
mfc0 $a0,$14
add $a0,$a0,$a1
mtc0 $a0,$14
eret
```

测试程序 3 测试了取数异常和 eret 后硬件添加的延迟槽

```
.text
ori $s0,1
lw $s1,0($s0)
ori $t0,1

.ktext 0x4180
ori $a1,4
mfc0 $a0,$14
add $a0,$a0,$a1
mtc0 $a0,$14
eret
ori $t7,1
```

测试程序 4 测试了定时器和外部中断的响应

```
    .text
    ori $s0,9#低四位分别是 1001
    ori $s1,1000
    sw $s0,0x7f10($0)
    sw $s0,0x7014($0)
```

```
    loop:#开始死循环
    beq $s0,$s0,loop
    nop

    .ktext 0x0004180
    ori $s0,9
    sw $s0,0x7f10($0)
    eret

.text
ori $s0,9
sw $s0,0($0)
ori $s1,1
sw $s0,0x7f00($0)
sw $s1,0x7f04($0)
ori $t0,1
ori $t1,2
lw $t3,0($0)#stall==========
ori $t3,4
ori $t4,5
ori $t5,6
lui $s0,0xffff
ori $s0,0xffff
lui $s1,0x8000
beq $s1,$0,tar
add $s0,$s0,$s1#===============
ori $t6,7
tar:
ori $t7,8

.ktext 0x00004180
mfc0 $1, $12
mfc0 $1, $13
mfc0 $1, $14
lui $s0,0x0000
lui $s1,0
sw $0,0x7f00($0)
sw $0,0x7f04($0)
ori $a0,$0,4
mtc0 $1,$14
nop
eret
```