CPU简易设计文档

该文档适用于P5P6P7，以求更快速度的增添指令

一. 主控制信号

1.指令与控制信号的对应

指令	NPCSel	ExtOp	RegDst	ALUSrc	ALUControl	MemWrite	RegWriteSel	RegWrite	TuseRs	TuseRt	Tnew	Op	Func	DataExtOp
位数	[1:0]	[1:0]	[1:0]	[1:0]	[2:0]	[0:0]	[1:0]	[0:0]	[1:0]	[1:0]	[1:0]	[5:0]	[5:0]	[2:0]
ADD	0	0	1	0	0	0	0	1	1	1	1	000000	100000	0
SUB	0	0	1	0	1	0	0	1	1	1	1	000000	100010	0
AND	0	0	1	0	4	0	0	1	1	1	1	000000	100100	0
OR	0	0	1	0	2	0	0	1	1	1	1	000000	100101	0
SLT	0	0	1	0	5	0	0	1	1	1	1	000000	101010	0
SLTU	0	0	1	0	6	0	0	1	1	1	1	000000	101011	0
ORI	0	0	0	1	2	0	0	1	1	3	1	001101		0
ADDI	0	0	0	1	0	0	0	1	1	3	1	001000		0
ANDI	0	0	0	1	4	0	0	1	1	3	1	001100		0
LUI	0	2	0	1	0	0	0	1	3	3	1	001111		0
LW	0	1	0	1	0	0	1	1	1	3	2	100011		0
SW	0	1	0	1	0	1	0	0	1	2	"0"	101011		0
JAL	2	0	2	0	0	0	2	1	3	3	0	000011		0
JR	3	0	0	0	0	0	0	0	0	3	"0"	000000	001000	0
BEQ	1	0	0	0	1	0	0	0	0	0	"0"	000100		0
BNE	4	0	0	0	1	0	0	0	0	0	"0"	000101		0
SLL	0	0	1	2	3	0	0	1	1	3	1	000000	000000	0
LB	0	1	0	1	0	0	1	1	1	3	2	100000		2
LH	0	1	0	1	0	0	1	1	1	3	2	100001		4
SB	0	1	0	1	0	3	0	0	1	2	"0"	101000		0
SH	0	1	0	1	0	2	0	0	1	2	"0"	101001		0

乘除指令的加装

指令	NPCSel	ExtOp	RegDst	ALUSrc	ALUControl	MemWrite	RegWriteSel	RegWrite	TuseRs	TuseRt	Tnew	Op	Func	DataExtOp	MDControl	MDDataOp
MULT	0	0	0	0	0	0	0	0	1	1	"0"	000000	011000	0	1	0
MULTU	0	0	0	0	0	0	0	0	1	1	"0"	000000	011001	0	2	0
DIV	0	0	0	0	0	0	0	0	1	1	"0"	000000	011010	0	3	0
DIVU	0	0	0	0	0	0	0	0	1	1	"0"	000000	011011	0	4	0
MTLO	0	0	0	0	0	0	0	0	1	3	"0"	000000	010011	0	5	0
MTHI	0	0	0	0	0	0	0	0	1	3	"0"	000000	010001	0	6	0
MFLO	0	0	1	0	0	0	3	1	3	3	1	000000	010010	0	7	0
MFHI	0	0	1	0	0	0	3	1	3	3	1	000000	010000	0	7	1

中断异常指令的加装

指令	NPCSel	ExtOp	RegDst	ALUSrc	ALUControl	MemWrite	RegWriteSel	RegWrite	TuseRs	TuseRt	Tnew	Op	Func	Rs	CP0Write	ExcCode	EXLClr
SYSCALL	0	0	0	0	0	0	0	0	3	3	0	000000	001100		0	8	0
ERET	0	0	0	0	0	0	0	0	3	3	0	010000(CP0指令)	011000		0	0	1
MTC0	0	0	0	0	0	0	0	0	3	2	0	010000		00100	1	0	0
MFC0	0	0	0	0	0	0	4	1	3	3	2	010000		00000	0	0	0

注：(1)Tnew的“0”，Tuse的3代表无意义

(2)此处ALUControl是宏观整体行为，未考虑不同异常，其进一步针对不同改装码的改装见下“五”

(3)新加装且第一个表格未提到的信号均为0

2.各个控制信号的意义

取值	功用	0	1	2	3	4	5	6	7
nPCSel	下一条地址	正常PC+4	小跳转 偏移offset	大跳转 imm26	GRF值				
ExtOp	立即数扩展	无/0扩展	符号扩展	立即数置高位					
RegDst	写入GRF地址	Rt	Rd	31					

取值	功用	0	1	2	3	4	5	6	7
ALUSrc	ALU第二计算数选择	GRF读出的第二个数	立即数	Rt(移位操作)					
ALUControl	ALU计算操作	无/加法	减法	或	逻辑左移	与	有符号小于置1	无符号小于置1	
MemWrite	是否写入存储器&处理写入数据	否	SW 是	SH 是	SB 是				
RegWrite	是否写入寄存器	否	是						
RegWriteSel	写入寄存器的值选择	ALU计算结果	存储器读出结果	PC+8	乘除模块HILO读出结果	CP0协处理器读出结果			
DataExtOp	处理存储器读出数据	无处理	无符号扩展字节	有符号扩展字节(LB)	无符号扩展半字	有符号扩展半字(LH)			
MDControl	乘除模块操作&判断是否为乘除指令	非乘除指令	有符号乘	无符号乘	有符号除	无符号除	写入LO	写入HI	读出HI/LO
MDDataOp	乘除模块读出选择	无/LO	HI						
CP0Write	是否写入CP0模块	否	是						
ExcCode	D级产生的(因特殊指令而产生的)异常码								
EXLClr	是否清空M级前流水线寄存器	否	是(ERET专属)						

注：MemWrite信号需配合Addr[1:0]进一步判断产生Byteen信号

二. 暂停转发矩阵

利用AT法判断每条指令的TuseRT TuseRS Tnew

Tnew	E	E	E	M	M	W
TuseRs|rt	0	1	2	0	1	0
0	F(Forward)	S(Stall)	S	F	S	F
1	0	0	S	F	F	F
2	F	F	F	F	F	F

整体思路：D级强制阻塞，后级但凡出现冒险一定能通过转发解决，尽可能多转发，每级皆转发（共五个转发多路选择器RD1,RD2,ALUA,ALUB,StoreData）

注：暂停一共只有四种情况 无过多值得注意的地方

三. 有关延迟槽

1.B/J型指令会用到延迟槽，紧随其后的一条指令不会清除而是会执行，因此JAL应链接下两条指令，即$PC+8$;

2.无需考虑分支冒险，仅考虑数据冒险即可

四.整体通路结构

图片导入略麻烦 见思考题

乘除模块的延时信号已经发出立即进行阻塞判断

五.ALUControl的改编

为应对P7出现的ExcCode码 需要将一个操作（如ADD）细分为若干个操作 以判断出相应的异常码

ALUControl	NUM	适用指令
ADD_unsign	0	其余一切无需计算指令
ADD_sign	1	ADDI ADD
SUB_sign	2	SUB
OR	3	OR ORI
AND	4	AND ANDI
SLT	5	SLT
SLTU	6	SLTI
ADD_LW	7	LW
ADD_LH	8	LH
ADD_LB	9	LB
ADD_SW	10	SW
ADD_SH	11	SH
ADD_SB	12	SB

六.细节(犯过错的地方)

1.VPC即完整错误指令码 无需截取31：2

2.内部异常无需考虑IM

3.注意PC的优先级，也可以说注意全部流水线寄存器的优先级，即reset > req > exlclr > stall

七.思考题

(一).输入输出设备

1.键盘控制流程

(1)每一个按键对应两个独特的编码，按下时产生的编码叫通码，松开时的编码叫断码，由键盘编码器统一控制。键盘编码器监控是否有键按下或弹起，若有键按下，向键盘控制器发送此键的通码；若有键弹起，则发送断码。
(2)键盘控制器将编码保存到自己的输出缓存区中，然后通过中断控制器向CPU发送键盘中断信号(不同外设的中断信号肯定不同，课上做出了很大的简化)
(3)CPU未关中断的情况下响应，中断控制器再发送中断向量号，CPU根据向量号定位中断服务程序，并进入
(3)压栈保护上下文，从键盘输出缓存区读取扫描码，**若不从输出缓存区读取数据的话，键盘控制器是无法继续工作的**，读取后判断这个扫描码是否是有多字节的，若有多字节，需要先保存下来等待接下来的扫描码组合成为完整的扫描码。
(4)寻址调用相应的键处理程序。每个扫描码都有一个键处理程序入口，寻址并进入，执行特定的操作
(5)键处理程序：对于普通键的处理，主要的功能为将扫描码转化成ASCII码，然后放进缓冲区中；对于操作控制键如CTRL SHIFT等的处理，设置一个寄存器(mode/leds)的相应位，当检测到他们的通码时，相应位置1，而检测到断码时，相应位置0(因此我们使用组合键时要先按下控制键,程序为控制键置好按下状态，再处理后来的键时会检查这些标识，是否有控制键按下，来执行不同的操作)
(6)退出中断，执行原任务

2.鼠标控制流程

(1)鼠标不只有左键右键滚轮侧键，还需要有光标位置的判定，因此在图形化界面的不同的交互位置和不同的点击松开按键结合都将产生不同的编码，大致流程与上无异。

3.总结

外设其内部均有一些微处理器控制自己与CPU的信息交互，每个设备都需要有驱动程序，有的在电脑系统内，如无线键鼠，驱动程序就在USB头上，叫免驱设备；有的需另外安装，若没有驱动程序的话，CPU是无法识别内容的。

(二).自定义中断处理程序

我认为是不妥当的，在自定义时程序有可能会占据CPU存储空间的其他地方，如占据DM段，外设段等，这样会使得访存时出现冲突。中断程序的入口地址是一个固定的地址，不会因任何因素变化而变化，而在设计系统之初就提前制定好地址，并不允许用户私自更改，是便于维护的，合乎道理的。

(三).为何与外设通信用BE

系统桥相当于CPU与外设沟通的桥梁。外设的种类是无穷无尽的，而 CPU 的指令集却是有限的。我们并不能总是因为新加入了一个外设，就专门为这个外设增加新的 CPU 指令。我们希望的是，尽管外设多种多样，但是 CPU 可以用统一的方法访问它们，这个方法在MIPS中便是为每个外设配置一块CPU地址空间，使用L/S型指令向不同的地址读出/写入数据，来交换外设寄存器所蕴含的信息。为了实现这个目标，我们设计了系统桥。
系统桥是连接 CPU 和外设的功能设备，它会给 CPU 提供一种接口，使得 CPU 可以像读写普通存储器一样（即按地址读写）来读写复杂多变的外设。系统桥统一且简化了 CPU 的对外接口，CPU 不必为每种外设单独提供接口，符合高内聚，低耦合的设计思想。

(四).关于计时器

初始化相同，即先将倒计时数存在PRESET寄存器，后将行为模式存入CTRL寄存器，在允许计数时先将PRESET寄存器的值赋值给COUNT寄存器，后每个周期使COUNT寄存器值减一，当减到0时，两种模式产生异常。
模式0：减到0时中断产生并持续有效，将计数器写使能变为0，直到计数器写使能为1时(由CPU修改)，清除中断信号，重新载入初值，再次开始计时。
模式1：减到0时中断产生并只存在一个时钟周期，下一个时钟周期重新载入计数器初值，再次开始计时。

图如下

模式0:

模式0

模式1:

模式1

(五).倘若中断信号流入的时候，在检测宏观 PC 的一级如果是一条空泡（你的 CPU 该级所有信息均为空）指令，此时会发生什么问题？在此例基础上请思考：在 P7 中，清空流水线产生的空泡指令应该保留原指令的哪些信息？

会发生的情况为，在异常处理程序返回时EPC中返回的值为0，导致程序错乱，显然不符合逻辑。空泡指令应保留被阻塞的当前指令的PC值以及判断其是否为延迟槽指令的BD位控制信号，以便在空泡中断返回后回到正确的指令上执行。至于为何要保留BD位，则是由于当延迟槽指令被阻塞时中断，中断的气泡就应该像中断的延迟槽指令一样，向EPC写入PC-4(跳转指令的地址)，否则返回时会返回到延迟槽指令，进而顺序进行，与前跳转指令逻辑不符。

(六).为什么 jalr 指令为什么不能写成 jalr 31,31?

jarl为跳转并链接指令，行为是$PC<-GPR[rs];GPR[rd]<-PC+4$
按照正常转发阻塞逻辑，其GPR[rs]在D级使用，TuseRs为0，而GPR[rd]在D级产生，其Tnew为0，因此当出现rs=rd时，转发需要在D级进行，此时将PC+4转发给PC，从意义上看便是顺序执行下一条指令，影响了PC原本应跳转到的GPR[rs]的值，因此jalr指令的$rs!=rd$