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.P7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

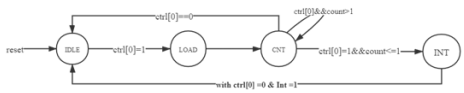
系统桥相当于CPU与外设沟通的桥梁。外设的种类是无穷无尽的，而 CPU 的指令集却是有限的。我们并不能总是因为新加入了一个外设，就专门为这个外设增加新的 CPU 指令。我们希望的是，尽管外设多种多样，但是 CPU 可以用统一的方法访问它们，这个方法在MIPS中便是为每个外设配置一块CPU地址空间，使用L/S型指令向不同的地址读出/写入数据，来交换外设寄存器所蕴含的信息。为了实现这个目标，我们设计了系统桥。
系统桥是连接 CPU 和外设的功能设备，它会给 CPU 提供一种接口，使得 CPU 可以像读写普通存储器一样（即按地址读写）来读写复杂多变的外设。系统桥统一且简化了 CPU 的对外接口，CPU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

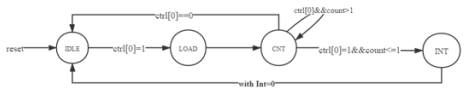
初始化相同，即先将倒计时数存在PRESET寄存器，后将行为模式存入CTRL寄存器，在允许计数时先将PRESET寄存器的值赋值给COUNT寄存器，后每个周期使COUNT寄存器值减一，当减到0时，两种模式产生异常。
模式0：减到0时中断产生并持续有效，将计数器写使能变为0，直到计数器写使能为1时(由CPU修改)，清除中断信号，重新载入初值，再次开始计时。
模式1：减到0时中断产生并只存在一个时钟周期，下一个时钟周期重新载入计数器初值，再次开始计时。

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$rs != rd$

八.Tube控制器原理

在人眼暂留时间内不断切换select(0001 0010 0100 1000) 并根据切换的select找到相应的四位data，并按照规律转化为八段数码管显示

九.UART协议

另外，代码中对于 preset 的注释可能令人产生疑惑，预设值明明使得 q 提前产生，为何会说 "lagging the sample point for a while"？原因在于这里所说的延迟是相对发送方的，接收方的采样点必然要落后于发送方。

十.P8前言

当阅读至此，微处理器的设计便来到了最后且最高深的一关，我们之前开发的CPU，只能在IM里接受一段来自汇编的指令码，并用开发软件自带的输出功能向外显示信息，这对于一个真正意义上的处理器显然是不合格的。一个合格的处理器应当具备真正的IO接口，并通过CPU交流信息，实现各种逻辑上的肉眼可见的直观的输入输出。在P7中，我们实现了系统桥，可以识别响应外部中断，这为P8奠定了基础，然而外部中断究竟从何而来，什么时候该响应中断，CPU如何与真正的外设交互；同时P7我们编写的异常处理程序可能只是简单的跳过或重复执行当前指令，这显然是为了图方便进行的操作，在P8中，为实现某些功能，需要我们编写一段简单却具有实际功能的异常处理程序，这些问题马上将会得到答案。在P8中，我们需要完成IP核的替换与改装，对GPIO，UART,TUBE这三类输入输出外设模块的添加，为其编写驱动程序，并学习到软件与硬件如何配合工作，相辅相成的实现最终要求。让我们一起走向最后的关底。

P8思考题

1.微系统接入乘除法器与外设驱动模块的方案

乘除法器：教程中提供了用加减法和移位操作代替verilog中*/%的运算的乘除核，但是我们需要和原先的乘除模块进行耦合，才能正确实现乘除法器。

乘除核信号

老四位代表mult，multu，div，divu
1.in_op, in_sign:这两个信号共同组合出四种需要用到乘除核的乘除指令，而我之前只用一个控制信号MDControl控制，因此需要组合逻辑进行转化
2.in_valid:当有老四位到来且非中断时置1，实际与原ready信号等价，因此assign即可
3.in_ready:当乘除核中可进行乘除法计算，即未被使用中时置1，与~busy等价，因此$assign busy = in_ready$即可
4.out_ready:外界此时能接受乘除法结果，对于本实验而言恒1即可，外界不存在不能接受的情况
5.out_valid:当计算结果合法，即计算完成后置1，相应修改为当out_valid==1时，lo与hi寄存器写入相应计算结果
6.其余数据输入输出信号易于理解

外设驱动模块：本实验共需要四种外设，即计时器Timer，可读可写；通用输入输出设备GPIO(其中包含8位用户按键user_key,一般用于选择工作模式，只读；64位拨码开关dip_switch，一般用于计算数的选择，只读；32位LED灯，一般用于结果的显示，只写)；2组四位八段数码管Tube，同样用于结果的显示，只写；UART串口，用于串口通信，可以实现串口回显，以及和LED,Tube相同的显示结果的功能。
每种外设都有各自的外设寄存器，用于和CPU的交互，同样也许会有一些内部的独立寄存器，为了使其接入CPU中，我们需要使用P7中系统桥的概念，将桥作为与外设寄存器交换信息的桥梁，每个外设对应CPU地址的部分空间，一一分配好地址空间后，便可以通过L和S型指令精确的找到相应的外设并进行寄存器堆与外设寄存器的信息交换。
由于DM使用了IP核，变为同步读同步写模式，读出结果滞后一个周期，因此如教程所言需将MW流水寄存器中的LoadData短接处理。**但值得注意的是，由于系统桥沟通的外设仍为同步读异步写模式，在读取数据时会使得和DM不一致(具体体现为为改DM而短接MW级的LoadData同样可能来自各个外设)，因此需在系统桥内对于外设的读取数据进行滞后一周期的处理，具体实现方法很简单，即将地址和各个外设的读取值先用寄存器存储起来，将寄存器值接入LoadData的选择即可。同时因此还要考虑原先在M级的DataExt模块需改装到W级，很关键！！**
外设寄存器可以存储数据，无论是CPU存入的(此时相对于CPU角度该寄存器只写)，还是外设内部计算得来而存入的(此时相对于CPU角度该寄存器只读)，也可以存储状态信号来控制外设，外设通过各自的外设寄存器做到了外设与CPU相沟通。可以做一个比喻，桥相当于车道，数据相当于跑的汽车，根据目的地不同走向不同的车道，而外设寄存器就相当于目的地的车位。

2.控制与外设交互的汇编程序

```
# 0x7f68 0~7 represent user_key lb $xx, 0($s0)
# 0x7f70 sw $xx, 0($s1) represent led_light
# 0x7f60 ALUB preset lw $xx, 0($s2)
# 0x7f64 ALUA lw $xx, 0($s2)
# 0x7f50 Tube (sign no use) sw $xx, 0($s4)
# 0x7f30 UART data
ori $s0, $0, 0xfff1
mtc0 $s0, $12
loop:
    lw $s0, 0x7f68($0)
    lw $s1, 0x7f64($0) # alua
    lw $s2, 0x7f60($0) # alub preset
    andi $t3, $s0, 0x0001 # cnt or maths
    andi $t4, $s0, 0x0002 # uart or led tube
    andi $t6, $s0, 0x00fc # 1111 1100 alucontrol
    andi $t7, $s0, 0x0004 # high or low
    beq $t3, $0, maths  # 1 cnt ; 0 maths
    nop
cnt:
    lw $s3, 0x7f08($0) # count
    # sw $s2, 0x7f04($0) # preset
    sw $s2, 0x7f04($0) # load preset
    beq $t7, $0, high
    nop
low:     # model 0
    ori $t3, $0, 0x0009
    j preend
    nop
high: # model 2
    ori $t3, $0, 0x000d
preend:
    sw $t3, 0x7f00($0) # ctrl

    beq $t4, $0, cntled
    nop
cntuart:
    beq $t2, $s3, loop
    nop
pass:
    lw $t2, 0x7f08($0)
    sw $s3, 0($0)

    lb $s3, 3($0)
    sw $s3, 0x7f30($0)
    jal wait
    nop
    lb $s3, 2($0)
    sw $s3, 0x7f30($0)
    jal wait
    nop
    lb $s3, 1($0)
    sw $s3, 0x7f30($0)
    jal wait
    nop
    lb $s3, 0($0)
    sw $s3, 0x7f30($0)
    jal wait
    nop
    j loop
    nop
cntled:
    sw $s3, 0x7f50($0) # tube
    sw $s3, 0x7f70($0) # led_light
    j loop
    nop

maths:
jia:
    ori $s6, $0, 4
    bne $t6, $s6, jian
    nop
    add $s3, $s1, $s2
    j print
    nop
jian:
    ori $s6, $0, 8
    bne $t6, $s6, cheng
    nop
    sub $s3, $s1, $s2
    j print
    nop
cheng:
    ori $s6, $0, 16
    bne $t6, $s6, chu
    nop
    mult $s1, $s2
    mflo $s3
    j print
    nop
chu:
    ori $s6, $0, 32
    bne $t6, $s6, qie
    nop
    div $s1, $s2
    mflo $s3
    j print
    nop
qie:
    ori $s6, $0, 64
    bne $t6, $s6, huo
    nop
    and $s3, $s1, $s2
    j print
    nop
huo:
    ori $s6, $0, 128
    bne $t6, $s6, loop
    nop
    or $s3, $s1, $s2
    j print
    nop

print:
    beq $t4, $0, aluled
    nop
aluuart:
    beq $t2, $s3, loop
```

```
    nop
    sw $s3, 0($0)
    lw $t2, 0($0)

    lb $s3, 3($0)
    sw $s3, 0x7f30($0)
    jal wait
    nop
    lb $s3, 2($0)
    sw $s3, 0x7f30($0)
    jal wait
    nop
    lb $s3, 1($0)
    sw $s3, 0x7f30($0)
    jal wait
    nop
    lb $s3, 0($0)
    sw $s3, 0x7f30($0)
    jal wait
    nop
    j loop
    nop
aluled:
    sw $s3, 0x7f50($0) # tube
    sw $s3, 0x7f70($0) # led_light
    j loop
    nop

wait:
    lw $t1, 0x7f34($0)
    andi $t1, $t1, 0x0020 #
    beq $t1, $0, wait
    nop
    jr $31
    nop
.ktext
    # timer
    lw $k1, 0x7f00($0)
    ori $k1, $k1, 0x0001
    sw $k1, 0x7f00($0)
    # uart back
    lw $k0, 0x7f30($0)
    sw $k0, 0x7f30($0)
    eret
```

大致思路为，首先根据第一位按键选择功能是计时器or计算器，若为计时器要根据第三位按键选择工作模式高向低计时or低向高计时，若为计算器要根据三到八位按键选择计算类型，加减乘除且或，最后根据第二位按键选择输出位置led&tube or uart进行输出。从最初选择功能时便不断使用条件判断if语句将指令一层层的分岔，执行各自的操作，最后可较为统一(如六种计算指令可跳转到同一部分的输出语句)的输出，然而由于计时器的特殊性，体现在若输入结果变化则应立刻归位重新计时，以及串口回显的特殊性，体现在只有当输出结果变化时再回显，需要对输出逻辑进行适当的修改以满足要求，这时输出函数就不能跳转到同一个了，因此针对不同的功能将有特殊性的处理。而串口回显功能我使用了进入异常中断处理的方法，不进入异常同样可行，智者见智。

3.犯过的错

1.M级DataExt模块没有改装到W级
2.计时器总是少一秒，计时器的异常产生与异常处理模块没配合好
3.IM没减3000
4.汇编逻辑混乱，难以debug