


# p4\_log

## 模块化层次化

- 整体结构与p3中logisim的分层一致，顶层端口定义为mips.v
- 不同：
  - 需要增加同步复位信号reset
  - clk留端口，而不是内置
- 需要支持的指令集为：add, sub, ori, lw, sw, beq, lui, jal, jr, nop
- 后期添加的指令为：slt,sll,lh,sh,lbu

## 设计方案

- 模块化、端口定义等等大致和p3一致
- 连线的时候的命名：推荐使用模块+输出端口命名法，小写字母表示，每一个模块的线放一块，这样输入的时候一目了然  连线命名
- 注意存储器地址访问的/4细节，思考题中的[11:2]应该就是出于这个考虑

## datapath

- 所有的MUX放在同一个MUX.v的文件中，命名为：MUX\_SelNum\_bit(默认为32位), 如 MUX\_4,MUX\_4\_26,MUX\_2\_5;

## RF

信号名	IO	描述	备注
clk	I	时钟信号	
reset	I	异步复位	
WE	I	写使能信号	
A1[4:0]	I	读出地址1	
A2[4:0]	I	读出地址2	
A3[4:0]	I	写入地址	
WD[31:0]	I	写入的数据	
RD1[31:0]	O	输出A1地址寄存器中的值	
RD2[31:0]	O	输出A2地址寄存器中的值	

## NPC

信号名	IO	描述	备注
PC[31:0]	I	当前指令地址	

信号名	IO	描述	备注
Simm26[25:0]	I	地址偏移	
ra[31:0]	I	返回地址	
CmpOut	I	B类指令的比较结果	
NPCOp[2:0]	I	NPC功能选择： 000: 顺序+4 001: B类跳转 010: jal/j 011: jr/jalr	
NPC[31:0]	O	下一条指令地址	

EXT

信号名	IO	描述	备注
imm16[15:0]	I	16位立即数	
EXTOp	I	EXT功能选择： 0: 0拓展 1: 符号位拓展	lui指令的高位拓展在ALU里执行
S0imm32	O	拓展后的32位立即数	

ALU

信号名	IO	描述	备注
ALUOp[3:0]	I	ALU功能选择	
A[31:0]	I	运算数1	
B[31:0]	I	运算数2	
shamt[4:0]	I	位移值	
ALUout[31:0]	O	计算结果	


DM

信号名	IO	描述	备注
clk	I	时钟信号	
reset	I	异步复位	
WE	I	写使能信号	
A[31:0]	I	待操作数据的地址	

信号名	IO	描述	备注
DMOp[2:0]	I	DM功能选择： 000: word 001: halfword 010: byte 1xx: 无符号拓展load	最高位置一表示无符号位拓展
DMout[31:0]	输出的32位数据		

指令	部件	NPC				IM	RF				ALU		DM	
	输入信号	PC32	imm26	ra32	BeqYes?	A	A1	A2	A3	WD	A	B	A	WD
add	PC					PC	rs	rt	rd	ALUout	RD1	RD2		
sub	PC					PC	rs	rt	rd	ALUout	RD1	RD2		
ori	PC					PC	rs		rt	ALUout	RD1	0imm16		
slt	PC					PC	rs	rt	rd	ALUout	RD1	RD2		
lw	PC					PC	rs		rt	DMout	RD1	Simm16	ALUout	
sw	PC					PC	rs	rt			RD1	Simm16	ALUout	RD2
beq	PC		imm16		Zero	PC	rs	rt			RD1	RD2		
j	PC		add26			PC								
jal	PC		add26			PC			0x1f	PC4				
jr	PC			RD1		PC	rs							
lui	PC					PC			rt	ALUout		0imm16		
lb	PC					PC	rs		rt	DMout	RD1	Simm16	ALUout	
lbu	PC					PC	rs		rt	DMout	RD1	Simm16	ALUout	
sb	PC					PC	rs	rt			RD1	Simm16	ALUout	RD2
sll	PC					PC	rs	rt	rd	ALUout	RD1	RD2		
总合计	PC	imm16 add26	RD1	Zero	PC	rs	rt	rt	ALUout		RD1	RD2	ALUout	RD2
								rd	DMout			0imm16		
								0x1f	PC4			Simm16		

controller

- 对于控制器，我采用先用操作码识别出不同的指令，再去记录下每个控制信号所对应的指令，因为这样更加符合电路的直观（或运算），也更符合我对未来的想象（）控制信号表

测试方案

- 还是用p3时候的测试数据，但是需要多多注意的是这个东西肉眼debug其实还好一点，但更加适合自动化评测，开始手搓！

增量开发

slt

- 处理ctrl，ALU，先填表再连线

sll

- shamt传入ALU，其他就没什么问题了

lh, sh, lhu

- 加上DMOp，在前面有定义

思考题

- 阅读下面给出的 DM 的输入示例中（示例 DM 容量为 4KB，即 32bit × 1024字），根据你的理解回答，这个 addr 信号又是从哪里来的？地址信号 addr 位数为什么是 [11:2] 而不是 [9:0]

- 从ALUout输出过来，是按字节寻址的指令，但我们的IM是按字寻址的，addr应该是4的整数倍，所以左移两位，按字存取。
- 思考上述两种控制器设计的译码方式，给出代码示例，并尝试对比各方式的优劣。
  - 第二种是最后靠信号输出的，更符合电路的或逻辑，便于后续开发，且更加符合电路
- 在相应的部件中，复位信号的设计都是同步复位，这与 P3 中的设计要求不同。请对比同步复位与异步复位这两种方式的 reset 信号与 clk 信号优先级的关系。
  - 异步复位reset的优先级高于clk，不管clk是否处于上升沿，都会复位（所以需要加入触发条件中）。
  - 而同步复位的reset优先级低于clk，只有clk到来才会生效。
- C 语言是一种弱类型程序设计语言。C 语言中不对计算结果溢出进行处理，这意味着 C 语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持 C 语言，MIPS 指令的所有计算指令均可以忽略溢出。请说明为什么在忽略溢出的前提下，addi 与 addiu 是等价的，add 与 addu 是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的 Operation 部分。
  - addi和add，在operate中加入了溢出检测
  - 以add为例
 

```

temp ← (GPR[rs]31 || GPR[rs]31..0) + (GPR[rt]31 || GPR[rt]31..0)
if temp32 ≠ temp31 then
    SignalException(IntegerOverflow)
else
    GPR[rd] ← temp
endif
          
```
  - 若R[rs]和R[rt]的30都是1且31相同的话，就分别对应正数和负数的溢出情况，则这么计算后 temp<sub>32</sub>!temp<sub>31</sub>，不相等，所以溢出。
  - 检测溢出后就会抛出异常，但没有检测溢出的话，还是正常计算，所以和addiu，addu一样。