



## P6 课下 & 课上总结

2021-12-08

首页 > 北航计算机组成原理

### H<sub>1</sub> P6：较完善的流水线 CPU 设计

#

通过阅读本文，您可以大致了解 2021 年秋季北航计算机组成课程 P6 课下搭建的思路，以及课上测试的题目内容、难度和解题思路

**!** P6 课上测试的主要内容是对课下用 Verilog 搭建的流水线 CPU 进行强测，同时添加一些新指令

题目每年都会发生变化，题意描述可能与原题有一定差异

## 太陽曰く燃えよカオス

後ろから這いより隊G

作詞: 畑畠貴

作曲: 田中秀和

編曲: 田中秀和

我々 (うー!) 寄れ依ねー (にゃー!) (我们我们(鸣-!) 息息在那(喵-!))

世界はDark! (こずみっく!) 全世界的Dark (cosmic)

(うー! にゃー! うー! にゃー! うー! にゃー! レツツにゃー!) ((J・ω・)) 喵-- (/・ω・)/喵--(J・

心ぬるぬる異形の神々 (心灵也扭扭曲是异形的神明神明)



私もあなたも怖がりの物すき (我和你都恐怖事物爱好者)

形むるむる異教徒逃げれば (形状也黏糊糊异教徒都逃得没影)

私とあなたは炎天下でおデート (我和你的约会就在炎炎烈日之下进行)

たのしいね 燃けるぞ溶けるぞ (很高兴呢!烧起来了!要融化了!)

(生き物じゃなーい) ((根本不是活物))

骸 (むくろ) ひやひや其れでも好きです (心里冰冷冰冷但还是喜欢)

私がいちばん欲しいのは内緒よ (我最想要什么那是秘密)

姿ばやばや其れでは行きましょ (身着邋遢邋遢就这样前进吧)

私にいちばん似合うかな不条理 (我的心里最最合适的东西是没有道理)

鏡には (だんだん 誰なの?) (在那镜子里 (谁 谁 是谁啊))

うつらない (誰なんだ? やだやだ) (啥都看不清 (到底是谁?讨厌讨厌))

欲望は言葉にしなくちゃ (うー!) (欲望如果不说出来的话)

ほら (にゃー!) 、消えちゃうでしょう? (レツツにゃー!) (你看 就会消失殆尽)

太陽なんか眩しくって (总觉得太阳太耀眼)

闇のほうが無限です (どきどき) (黑暗才更无限 (心跳心跳))

太陽ばっか眩しくって (太阳笨蛋总是太耀眼)

闇のほうがす・て・き (にゃんだ~?) (黑暗才更美丽 (为喵?))

CHAOS (けいおす) CHAOS I wanna CHAOS (CHAOS CHAOS I WANNA CHAOS)

燃えよ混沌無敵です (わくわく) (燃烧吧 混沌是无限的(兴奋不已))

CHAOS CHAOS una sera CHAOS (CHAOS CHAOS una sera CHAOS)

燃えるようなき・も・ち (にゃんで~?) (燃烧了起来的情感 (为喵?))

我々! (うー!) 遺れ破れ! (にゃー!) (我们我们(鸣-!) 要去打败(喵-!))

世界はDark! (こずみっく!) (全世界的Dark (cosmic))

(うー! にゃー! うー! にゃー! うー! にゃー! レツツにゃー!) ((嗚-! 喵-! 呜-! 喵-! 呜-! 喵-! Let's喵-!))

枕ばかばか無貌 (むぼう) の神々 (枕头也暖和暖和是没脸的神明)

私とあなたと目的は違うの (我和你去的目的是毫不相同完全背离)

絆むかむか無理矢理されがち (羁绊是无价无价往往无理也有道理)

わたしがあなたの守るけどおピンチ (我会永远守护着你即便是你身陷危机)

おかしいな逃げるぞ呻くぞ (真是奇怪呢 一边逃跑一边呻吟)

(狙われてるーぅ) ((被人盯得紧紧))

帳 (とばり) おりおり此所まで来ました (本子也折叠折叠最后终于来到此地)

私はだまって敵たちを蹴散らす (我就这样不声不响把敌人都消灭干净)

焰 (ほむら) めらめら此所では駄目です (火焰在熊熊燃烧呆在这里不行不行)

私にだまってスカウトはしないで (我才不会去做那种一言不发的侦察兵)

望むなら (ぜん ぜん 全滅!) (假若有希冀 (全全 全灭吧))

しましょうね (敵なんだ? やれやれ) (去做别犹豫 (都是敌人?哎呀哎呀))

願わくば跡形もなくね (うー!) (愿望这东西 完全也看不到形迹 (鸣-!))

あら (にゃー!) 、要らないでしょう? (レツツにゃー!) (哎呀 是否真要放弃? (Let's喵-!))

現在なんと危険だった (现在这里太过于危险)

光もんを下さい (ぴかぴか) (光明请拿到我面前 (闪亮闪亮))

現在なんと危険だった (现在这里太过于危险)

光もんでし・げ・き (にゃるビーム?) (光明刺激再多·一点 (喵射线?))

MADNESS MADNESS You wanna MADNESS (MADNESS MADNESS You wanna MADNESS)

呼べば冒瀧過激です (ぞくぞく) (呼喊冒瀧打破界限 (接连不断))

MADNESS MADNESS ennu MADNESS (MADNESS MADNESS ennu MADNESS)

呼べばきっとお・し・まい (にやんです!) (呼喊是结束的·瞬间 (是喵!))  
這いよる! (うー!) 混沌! **Test Blog** (潜行潜行 (呜-!) 混沌混沌(喵-!))



ニャルラトホ! (てっふー!) (奈亚拉托提普 (te-pu))

太陽曰く燃えよカオス

TVアニメ「這いよれ! ニャル子さん」OP

作詞: 畑亜貴

作曲/編曲: 田中秀和(MONACA)

歌: 後ろから這いより隊G (ニャル子 (阿澄佳奈))

×クー子(松来未祐)×珠緒(大坪由佳)

(うー! にゃー! うー! にゃー! うー! にゃー! ((J · ω ·)) 唔 -- (/ · ω ·)/喵---(J · ω ·) 唔 -- (/ · ω ·)/

うー! にゃー! うー! にゃー! うー! にゃー! ((J · ω ·)) 唔 -- (/ · ω ·)/喵---(J · ω ·) 唔 -- (/ · ω ·)/

うー! にゃー! レッツにゃー! ) ((J · ω ·)) 唔 -- (/ · ω ·)/喵---(J · ω ·) 让我们唔喵! )

太陽なんか眩しくって (总觉得太阳太耀眼)

闇のほうが無限です (どきどき) (黑暗才更美丽 (小鹿乱撞))

太陽ばっか眩しくって (太阳笨蛋总是太耀眼)

闇のほうがす・て・き (にやんだ~?) (黑暗才更美丽 (为喵?))

CHAOS (けいおす) CHAOS I wanna CHAOS (CHAOS CHAOS I WANNA CHAOS)

燃えよ混沌 無敵です (つくわく) (燃烧吧混沌 无敌的(兴奋不已))

CHAOS CHAOS una sera CHAOS (CHAOS CHAOS una sera CHAOS)

燃えるようなき・も・ち (にやんで~?) (燃烧了起来的情感 (为喵?))

這いよる! (うー!) 混沌! (にゃー!) (潜行潜行 (呜-!) 混沌混沌(喵-!))

ニャルラトホ! (てっふー!) (奈亚拉托提普 (te-pu))

~END~ (-终-)



## Something

▶ 太陽曰く燃えよカオス

00:00 / 03:53



## H<sub>2</sub>课下部分 #

P5 已经完成了流水线的大部分框架性工作了，P6 要做的主要就是三件事：按照教程要求改造 DM，添加乘除槽以及添加大量指令

要求实现的指令集为 **MIPS-C3**，即 LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、SUB、SUBU、MULT、MULTU、DIV、DIVU、SLL、SRL、SRA、SLLV、SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO，可以看出已经包括了除了浮点数运算和 CPU 协处理器之外的所有指令

第一件事情，也是最容易做的就是添加运算类的和跳转类的指令了，由于支持的指令非常多，建议用宏的方式，定义每一条指令的 `opcode` 和 `funct`，例如单独建立一个文件 `def.v` 用于定义所有的控制信号和指令的 `opcode` 和 `funct`，尽量避免在程序主体部分出现 magic number

跳转类的只需要添加 b 类跳转，由于按照 P5 的架构，由 D 级的 CMP 判断是否需要跳转，那么我们只需要修改 `CMP` 即可支持所有的指令

运算类的需要修改 `ALU`，这里非常容易出现有符号数、无符号数，`$signed`，`$unsigned` 的符号问题，建议根据课下给出的弱测数据逐一进行功能测试，确保无问题

由于 P5 的阻塞和转发是对于某一类指令做的，添加上面两类指令时必然不会出现其它新的转发阻塞问题，因此只需要将他们加到原有分类中即可

需要特别注意 `SLL`、`SRL`、`SRA`、`SLLV`、`SRLV`、`SRAV` 这些指令，因为如果不可变的移位指令如 `SLL`、`SRL`、`SRA`，符号是一方面问题，更重要的是他们无需使用 `$rs` 寄存器，在阻塞的时候需要特别考虑

对于上面的问题，解决方案是单独分出两类，然后阻塞特殊判断，代码如下：

● ● Verilog

```
1 // control.v
2 assign shift_s = sll | srl | sra;
3 assign shift_v = sllv | sr1v | srav;
4
5 // stall.v
6 assign D_Tuse_rt = (D_branch) ? 8'd0 :
7             (D_calc_r | D_md) ? 8'd1 :
8             (D_store) ? 8'd2 :
9             8'd3;
10 assign D_Tuse_rs = (D_branch | D_jump_reg) ? 8'd0 :
11             (D_calc_i | (D_calc_r & !D_shift_s) | D_load | D_stor
12             8'd3;
```

第二件事情是实现乘除槽，我们需要模拟乘除法的时延，因此乘法需要慢 5 个周期，除法需要慢 10 个周期，乘除槽有天然的 HI 和 LO 两个寄存器，因此除了 `mflo`, `mfhi`, `mthi`, `mtlo` 不需要考虑转发，但是需要考虑阻塞，我们这样做

```

1 // control.v
2 assign md = mult | multu | div | divu;
3 assign mt = mtlo | mthi;
4 assign mf = mflo | mfhi;
5
6 // stall.v
7 wire E_stall_mdu = ((D_md | D_mt | D_mf) && (E_MDU_Busy | E_MDU_Start));
8 assign Stall = E_stall_rs | E_stall_rt | M_stall_rs | M_stall_rt | E_stal

```

## P6 课下 & 课堂总结

注意在乘除槽处于运算过程中时，如果 D 级出现了任何与乘除槽有关的指令，我们都必须

2021-12-08

阻塞！

此外对于乘除槽内部，我们维护一个 `tmp_HI` 和 `tmp_LO`，乘法需要慢 5 个周期，除法需要慢 10 个周期只是假象，我们在第一个周期就能算出结果，只是我们给他存入 `tmp_HI` 和 `tmp_LO` 里面，到规定的时延之后我们再把他们转存到 `HI` 和 `LO` 里面，像这样

```

1 if(cycle_cnt == 0) begin
2     if(Start) begin
3         Busy <= 1;
4         if(MDUOp == `MDU_multu || MDUOp == `MDU_mult) cycle_cnt <= 5;
5         else if(MDUOp == `MDU_div || MDUOp == `MDU_divu) cycle_cnt <= 10;
6
7         if(MDUOp == `MDU_multu) {tmp_HI, tmp_LO} <= D1 * D2;
8         else if(MDUOp == `MDU_mult) {tmp_HI, tmp_LO} <= $signed(D1) * $signed(D2);
9         else if(MDUOp == `MDU_div) begin
10             tmp_LO <= $signed(D1) / $signed(D2);

```



然后就是正常的状态机，每一次时钟上升沿 `cycle_cnt` 都会减 1，减到 1 的时候把

`tmp_HI` 和 `tmp_LO` 转存

```

1   if(cycle_cnt == 1) begin
2       L0 <= tmp_L0;
3       HI <= tmp_HI;
4       Busy <= 1'b0;
5       cycle_cnt <= 0;
6   end else begin
7       cycle_cnt <= cycle_cnt - 1;
8   end

```



第三件事 DM 改造，课程组今年已经给出了标准的 DM 实现，我们只需要调用对应的接口即可

## P6 课下 & 课上总结

这里我们用 **BE** 处理要写入的数据，使其符合规范，用 **DE** 处理读出的数据，使其符合 **lh, lb, lhu, lbu, lw, lw** 的要求

BE 的端口

信号名称	方向	功能描述
BEOp[1:0]	输入	控制信号
Addr[31:0]	输入	地址信息，用于处理半字、字节
rt_data[31:0]	输入	读取的寄存器数据，待处理
DMWrEn	输入	写使能
m_data_byteen[3:0]	输出	控制写入半字、字节的位置位置
m_data_wdata[31:0]	输出	待写入数据

DE 的端口

信号名称	方向	功能描述
DEOp[1:0]	输入	控制信号
Addr[31:0]	输入	地址信息，用于处理半字、字节
m_data_rdata[31:0]	输入	<b>mips_txt.v</b> 返回的 DM 中的数据
DMRD[31:0]	输出	处理之后的正确的读取数据

Verilog

```
1 // 与 DM 交互
2 M_BE _be(
3     .BEOp(M_BEOp),
4     .Addr(M_ALUAns),
5     .rt_data(M_FWD_rt_data),
6     .m_data_byteen(m_data_byteen),
7     .m_data_wdata(m_data_wdata)
8 );
9
10 assign m_inst_addr = M_PC;
```



注意今年的 P6 也不需要在寄存器堆里面输出写入信息了，同样是调用 `mips_tb.v` 的接口即可

这一部分仔细读一读教程，然后看一看提供的源代码就行，问题不大



## H<sub>2</sub> 课上部分 #

题目的考察离不开 P5 的那几种类型，逐一进行解释

个人的一些想法：

关于课上测试的准备，自己练习加的指令不求多，因为反正不可能考到原题，要更加熟悉你自己设计的流水线的架构，理解阻塞和转发的过程，做到心中有图，然后自己可以想一想如何添加，多想几种方法，找一个最方便的形成自己的套路，就足以应对课上测试了

并不一定需要真正把指令加到 CPU 里面，这有可能会引入潜在的 bug，而且你没有.class 文件也没法做测试，出了问题除了让自己血压升高没别的好处

下面的总结和套路也仅供参考，但是至少顺利通过了 P5 和 P6 的课上测试

## H<sub>3</sub> 第一类：运算类 #

### H<sub>4</sub> shl (swap HI LO) #

这是第一次课上考试的题目，要求是交换 HI、LO 寄存器的值

可以直接这样加（注意非阻塞赋值的运用），非常容易

Test Blog



Verilog

```
1 if(MDUOp == `MDU_shl) begin  
2     HI <= LO;  
3     LO <= HI;  
4 end
```

## H<sub>4</sub> bds (big divide(unsigned) small) #

### P6 课下 & 课上总结

给出 rs 和 rt 两个寄存器，用较大的那个数除以较小的那个数，注意是无符号除法

2021-12-08

这是第二次课上考试的题目，也非常容易（注意无符号）

Verilog

```
1 if(MDUOp == `MDU_bds) begin  
2     if(D1 > D2) begin  
3         tmp_LO <= D1 / D2;  
4         tmp_HI <= D1 % D2;  
5     end else begin  
6         tmp_LO <= D2 / D1;  
7         tmp_HI <= D2 % D1;  
8     end  
9 end
```

可以看出主要是关于乘除槽的计算，只要熟悉乘除槽的工作原理，第一题不难做出

## H<sub>3</sub> 第二类：条件跳转 #

### H<sub>4</sub> bezal #

指令描述

special	rs	rt	0	0	funct
000000	rs 寄存器地址	rt 寄存器地址	00000	00000	110001

指令行为

若 `GPR[rt] = 0`，则跳转到 `GPR[rs]`，并且链接到 `GPR[31]`



## 解法

这是第二次上机考试的题目

对于条件跳转的题目，有两个不同性质：条件链接 / 无条件链接，清空 / 不清空延迟槽，本题显然是条件链接 + 不清空延迟槽

而且由于在 D 级我们就可以获得是否跳转，因此我们只需要修改 `NPC` 和 `CMP` 两个元件，明确 \*\* `CMP` 是用来比较产生是否跳转的结果的，`NPC` 是根据 `CMP` 的结果决定跳转到哪里去的 \*\*，然后相应修改两个元件即可

2021-12-08

我强烈建议大家课下搭建 `CPU` 时把是否跳转的 `b_jump` 信号跟着流水线一直传递到最后，因为条件链接需要根据 `b_jump` 信号来决定本条指令是写 `5'd31` 还是 `5'd0` 寄存器，在把 `b_jump` 连到 `control` 里面后只需要这样即可

Verilog



```
1 // 要向寄存器中写入的值
2 assign GRFWDSel = (jump_link | branch_link) ? `WDSel_pc8 :
3                               (load) ? `WDSel_dmrld :
4                               (mf) ? `WDSel_mduans :
5                               `WDSel_aluans;
6
7 // 要写入哪个寄存器
8 assign GRFA3 = (calc_r | jalr | mf) ? rd :
9                               (calc_i | load) ? rt :
10                             (jal | (branch_link && b_jump)) ? 5'd31 :
11                               5'd0;
```

对于清空延迟槽，P5 已经说过了，要注意在处于阻塞状态时不能立即清空延迟槽指令，因为这时候你得到的 `b_jump` 未必是正确的，代码如下：

Verilog



```
1 // D_CMP.v
2 assign flush = (CMPOp == `CMP_bltzall && !jump);
3
4 // mips.v
```

对于本题而言，跳转方式反而与 **jr, jalr** 类似，但是还是需要 **CMP** 的判断，因此考虑给 **NPC** 和 **CMP** 都添加新的控制信号，然后 **CMP** 按 **rtData < 0** 比较，**NPC** 按 **jr** 跳转，就可以轻松解决本题

#### H<sub>4</sub> 类似的往年题

#

#### H<sub>5</sub> blezalc

#

### P6 课下 & 课上总结

小于等于 0 时跳转并链接，显然也是跳转并链接并且不清空延迟槽，按照上述套路只需要改 **CMP** 即可，因为这条指令的跳转方式跟其他的 b 型指令一致

2021-12-08

#### H<sub>3</sub> 第三类：条件储存

#

属于是大 Boss 了，不论是什么样的条件储存题目，都有这样的特点：**我们至少在 M 级之前都无法确切得知我们要写的寄存器和要写的值是什么**，这就是条件储存的“条件”

对于这类题目，我们认为最保险的情况就是**当啥都知道的时候（即 W 级时），在一起处理所有的问题，即要写啥要写什么这两个问题**

但是问题在于，在 W 级之前怎么办？

**当我们不确定自己要写啥寄存器时，把当前要写的寄存器设置为 0 总是最保险的**，这是因为我们不会处理向 0 号寄存器里面写的转发

至于要写啥有需要分成两种情况：一是一开始就知道要写的是 DM 的数据，二是那种如果满足某种情况就写 DM 数据，否则我们写其他数据（寄存器啊，PC+8 啊，PC+4 啊各种乱七八糟的...），但是由于我们是在 W 级才处理这些问题，所以无论写的是啥乱七八糟的东西，我们可以确定拿到的一定是正确的数据，结论是在 W 级之前，我们设置的写入寄存器的数据是啥都无所谓！

为什么呢？这就是下面要说的阻塞，我们对于条件储存的指令，阻塞的策略是：如果后面有这条指令，前面的寄存器需要读**后面这条指令可能需要写的寄存器**，在这里 **lhogeZ** 就是指 rt 和 31 号寄存器，再例如说下面的 **lwmx** 指令就是 4 号和 5 号寄存器，**总之只要条件储存指令在流水线 W 级之前，并且后面的那条指令要读后面这条指令可能需要写的**

寄存器，就必须把它阻塞在 D 级，这样的话条件储存指令在 W 级之前根本不会向前转发

因此根本不需要特别关心设置的写入寄存器的数据是啥，阻塞时这样设置即可

Verilog

```
1   wire E_stall_rs = ((E_lwmx ? (D_rs_addr == 5'd4 || D_rs_addr == 5'd5) :  
2   wire E_stall_rt = ((E_lwmx ? (D_rt_addr == 5'd4 || D_rt_addr == 5'd5) :  
3  
4   wire M_stall_rs = ((M_lwmx ? (D_rs_addr == 5'd4 || D_rs_addr == 5'd5) :  
5   wire M_stall_rt = ((M_lwmx ? (D_rt_addr == 5'd4 || D_rt_addr == 5'd5) :  
6  
7   assign Stall = E_stall_rs | E_stall_rt | M_stall_rs | M_stall_rt;
```

然后是条件如何判断？我们选择在 W 级判断，直接把判断逻辑写在 `mips.v` 或者 `datapath` 里面，生成一个 `check` 信号，然后连入 W 级的 `control` 里面去判断写什么，写哪里

但是这个时候前面 DEM 级的 `control` 会出现 `check` 信号是浮空值 `z` 的情况，这时候我们需要特殊用 `==` 语法进行判断，即得到浮空值，并且当前指令是条件储存，那么这时把要写的寄存器地址赋值成 `5'd0`，以 `lhnez` 指令和 `lwer` 指令为例，示例代码如下：

Verilog

```
1 // mips.v  
2 reg W_lhnez_check;  
3 reg [31:0] W_lwer_dst;  
4  
5 always @(*) begin  
6     W_lwer_dst = (W_rt_data + W_DMRD) & 32'h1e;  
7 end  
8  
9 always @(*) begin  
10    count1 = 0;
```

`lwer` 指令还稍微有些不同，我们在 W 级确定的是写入的地址，因此我们传入的不再是 `check` 信号，而是寄存器地址

里面还有一些别的其它指令，后面会再说，先说一下 `lwer` 和 `lhonz` 以及本次课上考  
试指令的具体描述

H<sub>4</sub> `lwer` #

### 文字描述

从内存读出数 (`memword`)，经表达式

```
(memword + GRF[rt]) & 0x1e
```

算出要写的寄存器编号（必为偶数），再将 `memword` 写入该寄存器

### 解法

这是 P6 第一次课上考试的题目

关键代码已经如上，具体思路就是在 W 级判断，其余数据相关指令阻塞到 D 级，注意这里的数据相关型指令是指要读的寄存器地址为偶数的指令

H<sub>4</sub> `lhonz` #

### 文字描述

从内存读出半字数据，比较该数据中数字 1 和 0 的个数的多少，1 多的情况下将该数据写入 rt 寄存器，否则将 PC+4 写入 31 号寄存器

### 解法

思路还是上面那样，总之在 W 级再判断，然后其他的数据相关指令阻塞到 D 级不会出问题，大致框架定下来之后，其他的就是细化如何判断，阻塞逻辑是什么（就是什么数据相关的指令），然后把 `check` 信号连到 `control` 里面，注意一下特判浮空值，然后就不会有问题了

H<sub>4</sub> `lwmx` #

### 指令描述

opcode	base	Test	Blog	rt	offset	↑
101010	base 寄存器地址			rt 寄存器地址	16 位偏移量	🔍

## 指令行为

如果从内存中读出的 `memword` 值大于 `GPR[rt]` 则将 `memword` 写入 5 号寄存器中，否则写入 4 号寄存器中

## 解法

### D6 课下 & 课上总结

首先明确数据相关的指令就是指需要读 4 号 5 号寄存器的指令，这些指令需要在 `stall.v` 中添加逻辑让其阻塞在 D 级

然后把判断放在 W 级，这时候我们需要把 `rt_data` 数据跟着流水线传递到 W 级（这时候的 `rt_data` 一定是正确的，想一想为什么？）

判断就是比大小，注意是有符号比较，然后生成 `check` 信号传入 W 级 `control`，小心 DEM 级 `control` 的浮空值

在 `control` 里面添加逻辑根据 `check` 信号是 `1'b0, 1'b1, 1'bz` 设置正确的待写入寄存器地址

待写入寄存器的值从一开始就很明确，是读出的 `memword`，所以没啥好说的

## H<sub>4</sub> 往届考试的相似题型

#

### H<sub>5</sub> lwid

#

读出来的 `memword` 如果满足 `memword[1:0]==0 && memword >= 32'h3000` 就写到 31 号寄存器，否则写到 rt 寄存器

典型的条件写，待写入的值很明确，就是 `memword`，只需要在 W 级完成判断逻辑，在 `stall.v` 中完成阻塞逻辑即可

## H<sub>3</sub> 第四类：一些奇奇怪怪不能归入某一类的题目

#

### H<sub>4</sub> movz

#

当  $\text{GRF}[rt] > 0$  时，将  $\text{GRF}[rs]$  写入  $\text{GRF}[rd]$ ，否则不写

## 解法

好像是一条 R 型计算类指令，但是如果  $\text{GRF}[rt] \leq 0$  时又不去写计算结果了

根据 Harahan 大佬的意见，本题可以看成计算类指令，但是需要在  $\text{ALU}$  中重新加入  $\text{ALU\_check}$  判断  $\text{GRF}[rt] > 0$ ，然后信号沿着流水线传递，去确定写不写寄存器

H<sub>4</sub>jap

## P6 课下 & 课上总结

#

2021-12-08

### 文字描述

从编号为 29 号的寄存器读出数据，作为地址将 PC+8 写入内存

维护这个以 29 号寄存器的值为栈顶指针的栈（这个栈和内存里的栈一样是倒着长得），即让 29 号寄存器的值减 4 再写入 29 号寄存器

完成与 j 指令一样的跳转操作

### RTL 语言描述

```
Addr <- GRF[29]
mem[addr] <- PC + 8
GRF[29] <- GRF[29] - 4
PC <- PC[31:28]||imm26||00
```

## 解法

这一题的操作非常多，贯穿了整个数据通路，所以我们应当找到一条**关键路径**，这条路径最长而且是这条指令的中心任务

我认为关键路径应当是  $\text{GRF}[29] \leftarrow \text{GRF}[29] - 4$ ，其他的什么跳转和写内存都只是路径上顺便完成的任务而已

首先到了 D 级，我们应当顺便利用  $\text{NPC}$  完成跳转操作，因为是无条件跳转，因此这个任务是没有难度的，直接改  $\text{NPCOp}$  即可

然后到了 E 级，我们利用 ALU 完成  $GRF[29] + 0$  操作，得到的 ALUAns 是要写入的地址，而要写入寄存器的值敲好就是 ALUAns - 4，这一点特判即可，关于阻塞，这里我们在 E 级就需要用到 GRF[29] 的值，即  $T_{use}\{jap\} = 1$

然后到了 M 级，我们写入 DM 只需要改变 DMWrEn，然后把 PC+8 作为写入数据即可，这时也应当特判

然后在 W 级，正常写入

需要注意的是：我们从 E 级开始就需要处理由于 **jap** 指令导致的 29 号寄存器数据的转发了

## P6 课下 & 课上总结

在上面的过程中，我们多次用到了特判，因此把 **jap** 归入哪一类指令都是不合适的，因此我们考虑单开一类，特殊处理

⌚ 北航计算机组成原理 ⌚ Verilog ⌚ MIPS ⌚ 流水线CPU ⌚ P6课上 ⌚ P6课下

📅 更新于 2021-12-15 📈 阅读次数 1153 次

❤ 赞赏

请我喝[茶]~(￣▽￣)~\*

👤 本文作者：FL @ FlyingLandlord's Blog

🔗 本文链接：<http://flyinglandlord.github.io/2021/12/08/BUAA-CO-2021/P6/P6课上&课下/>

COPYRIGHT 版权声明： 本站所有文章除特别声明外，均采用 ©BY-NC-SA 许可协议。转载请注明出处！

上一篇

北航计算机组成原理

P5课上测试1&2

下一篇

北航计算机组成原理

P7课下&课上总结



## P6 课下 & 课上总结

2021-12-08

### 最新评论



1 花露水 @ 3 hours ago

P7过了高低得给您磕一个Orz<img src="https://cdn.jsdelivr.net/gh/MiniValine/tieba@master/tieba-3...

2 pigKiller @ 4 days ago

P7过了高低得给您磕一个

3 Harry Potter @ 5 days ago

P7过了高低得给你磕一个

4 花露水 @ 68 days ago

大佬保佑我今晚pre不挂

5 do @ 2022-11-20

十分感谢，这篇帮我省下了大量琢磨P7的时间。（话说这篇的实现方法和roife学长的好像）

6 Anonymous @ 2022-06-15

哈哈，现在再看，我现在连OS的Extra都懒得做，当年还兴致勃勃的去做不加分的P8 人的变化可真大...

7 陈俊一 @ 2021-12-13

p6一遍过前来还愿

9 Edge @ 2021-11-23  
但愿P4—遍过

10 Edge @ 2021-11-23  
一神！！！

## 随机文章

1 北航计算机组成原理  
Verilog 学习笔记 (1)

2 北航操作系统  
BUAA-OS Lab2上机

3 北航计算机组成原理  
Pre课上测试游记

4 北航计算机组成原理  
P8课下&课上总结

5 北航计算机组成原理  
P6课下&课上总结

6 北航计算机组成原理  
MIPS汇编哈密顿回路

7 北航计算机组成原理  
P4课上测试

8 计算机科学 › 二进制杂谈 › Theme Shoka Documentation  
主题特殊功能测试

9 北航操作系统  
lab0-lab2总结

10 北航计算机组成原理  
P7课下&课上总结