

# 实验 3-1 报告

第 28 小组  
付琳晴、李舒博

## 一、实验任务（10%）

设计支持 15 条 MIPS 指令（LUI、ADDU、ADDIU、BEQ、BNE、LW、OR、SLT、SLTI、SLTIU、SLL、SW、J、JAL、JR）的静态 5 级流水 CPU，实现延迟槽技术，解决控制相关和结构相关，不考虑数据相关。

## 二、实验设计（30%）

lab03 将每个模块按照老师的模板全部重写，外部整体框架和数据线路如下图 1，图 2。

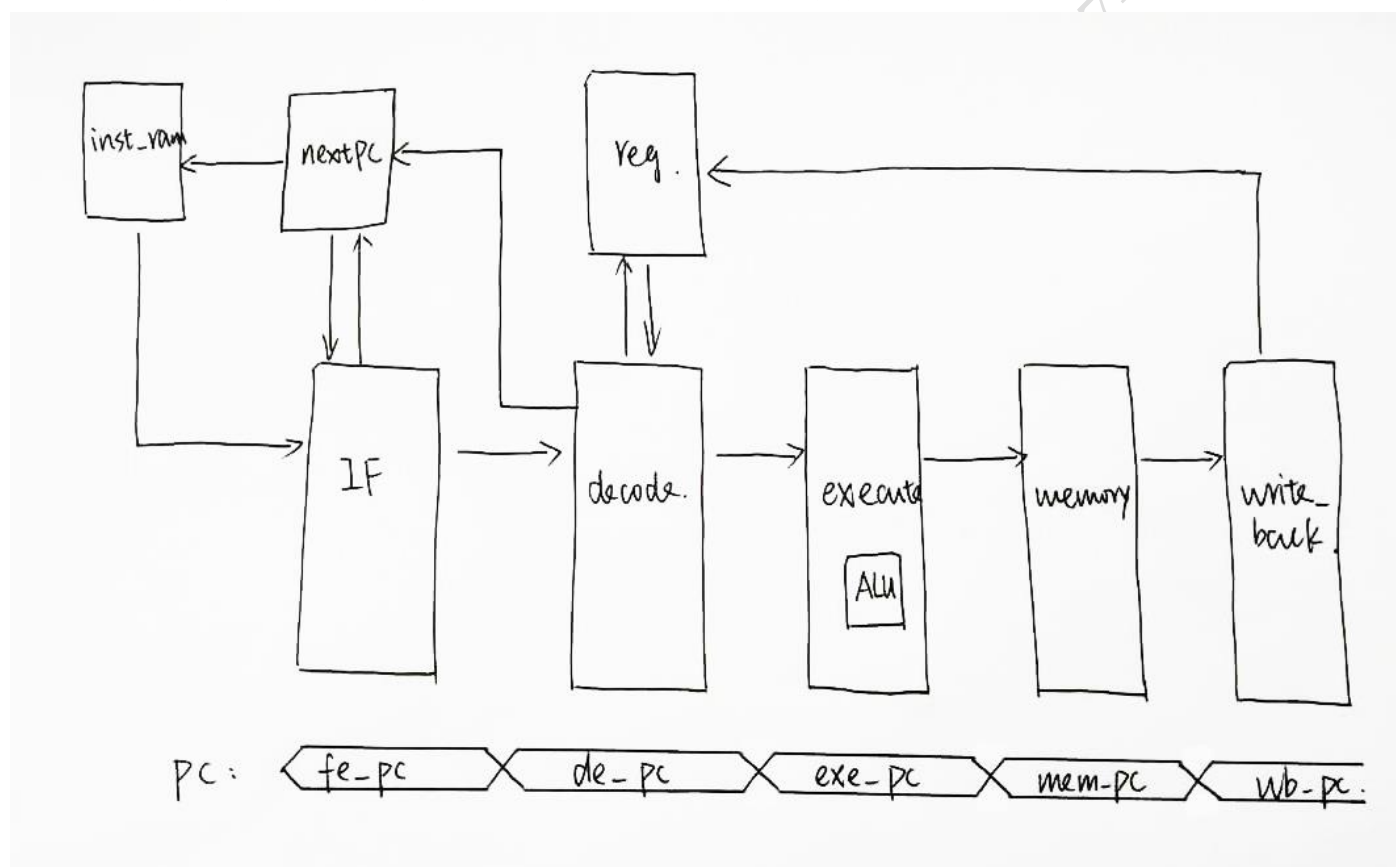


图 1. 五级流水整体框架



nextpc[31:0]	bfc59404	bfc59400	bfc59404	bfc59408	bfc5940c	bfc59410	bfc59414
de_pc[31:0]	bfc59400	bfc593fc	bfc59400	bfc59404	bfc59408	bfc5940c	bfc59410
exe_pc[31:0]	bfc593fc	bfc593f8	bfc593fc	bfc59400	bfc59404	bfc59408	bfc5940c
mem_pc[31:0]	bfc593f8	bfc593f4	bfc593f8	bfc593fc	bfc59400	bfc59404	bfc59408
wb_pc[31:0]	bfc593f4	bfc593f0	bfc593f4	bfc593f8	bfc593fc	bfc59400	bfc59404

图 4. 各级流水线的 pc 信号

下面对每一级流水模块设计进行细致介绍：

## (一) fetch\_stage

此阶段包括模块 nextpc\_gen 和 fetch\_stage. fetch-stage 输出 pc 当前指令的 pc 信号给 nextpc\_gen, nextpc\_gen 根据传来的 pc 信号和 decode\_stage 传来的分支指令选择信号计算出下一条指令的 pc, 由于 nextpc\_gen 中的信号全部采用 wire 类型, 因此能够在第一个周期内计算出下一条指令的地址并直接输出给 inst\_ram 进行取址, 即每条指令的取址信号总是在前一个周期便送出, 在下一个周期即取值阶段便已经取回指令。此设计影响了后续四个阶段模块都必须将输入信号用寄存器存后再用 (即寄存器加在每一级流水前面) 的设计方法。

nextpc\_gen 中输入了 decode\_stage 传来的分支指令选择信号, 如图 5。这些信号方便了 nextpc 的赋值。延迟槽技术的实现在 decode\_stage 阶段介绍。

```

35 module nextpc_gen(
36     input wire      resetn,
37
38     input wire [31:0] fe_pc,
39
40     input wire      de_br_taken,    //1: branch taken, go to the branch target
41     input wire      de_br_is_br,    //1: target is PC+offset
42     input wire      de_br_is_j,     //1: target is PC||offset
43     input wire      de_br_is_jr,    //1: target is GR value
44     input wire [15:0] de_br_offset,  //offset for type "br"
45     input wire [25:0] de_br_index,   //instr_index for type "j"
46     input wire [31:0] de_br_target,  //target for type "jr"
47 )

```

图 5. nextpc\_gen 分支选择信号

## (二) decode\_stage

在老师的设计中, 非常巧妙地将寄存器堆从 decode\_stage 中提了出来, 将 reg 和 decode\_stage 作为两个模块, 之间通过 wire 类型连接并传递数据。

decode 阶段除了来自 reg 的输入信号 de\_rf\_rdata1 和 de\_rf\_rdata2, 其余所有输入信号都需要寄存器存一下, 保证了输入数据恰好留到译码周期使用, 并且顺理成章地实现了延迟槽技术。例如: 第一条指令为分支跳转指令 bne, 该指令传到译码级, 对于 decode\_stage 内部的 de\_br\_taken 等将要传给 nextpc 的信号, 均在译码阶段才得到并传出, 此时 fetch\_pc 已经是下一条指令 (即延迟槽处指令), 意味着这些信号只能影响延迟槽指令的下一条指令的取出, 因此分支跳转会发生在延迟槽指令之后, 由此延迟槽技术实现。

老师给的代码框架保证了在译码级就判断出了会不会发生跳转, 即 branch 信号不需要从 exe 级别的 alu 中得出, 此时也能充分理解老师上课所提示的要把判断两个源操作数是否相等放在译码级进行的含义。

单周期设计的 control 模块和 alucontrol 模块放入了 deccode\_stage 模块内, 由此译码出各个控制信号, 传给后面流水级, 此处对老师的框架进行了小小的改动, 在 decode\_stage 输出信号新添加了一个 aluctr 信号, 方便后续选择 alu 的计算种类。不过, 按照原本设计是将译码出的所有控制信号传出, 但是最终害怕信号传乱, 且为了方便 debug, 还是选择了将 opcode 即指令的高 6 位一级一级传下去, 这样做的弊端是在后续流水级别还需要再次译码, 增加了电路的复杂性, 且没必要, 但为了方便第一次先这样写, 之后会修改的。

## (三) execute\_stage

前两个模块写好之后, 后面三个模块都非常好写, 只需要将信号传下去即可。

将 alu 模块放在了 execute\_stage 模块内部，由于在译码级已经判断了源操作数是否相等，因此此处 alu 就不需要输出 zero 信号了，反而更加简便。这里 alu 的两个输入使用的是已经用寄存器存过的数，也是为了保证数据留到 exe 级再使用。

另外，data\_sram\_addr 等需要输给 data\_ram 的信号在 exe 级就送出，将会在 mem 级得到返回给 memory\_stage，是恰好满足节拍的。

#### (四) memory\_stage

在该流水级，从 data\_ram 传回的 data\_sram\_rdata 恰好在 mem 级传回，因此该输入不需要用寄存器存一下再用，必须直接用。memory\_stage 中只需要多选器对于写回寄存器的数据源进行一下选择，一个是传回来的 data\_sram\_rdata，一个是 exe 级传来的 alu 计算结果 exe\_value，注意，exe\_value 需要转存一下才能使用。

#### (五) writeback\_stage

在该流水级，将输出信号给 reg 进行写操作，同时，该模块中输出信号 wb\_rf\_wen、wb\_rf\_wdata、wb\_rf\_waddr 需要作为 debug 信号与 reference 进行对照，此时终于明白 lab2 时候老师为什么强调的是用写回级 pc，pc 信号要沿周期传下来的真正意义，在流水线中得到充分体现。

以上五个框架完成并连接好后，便可实现五级流水功能。但代码中存在一个缺点：由于时间关系，没有按照老师提供的流水线代码设置 pipe\_valid、pipe\_allowin 等信号，因此不能成功的解决结构相关。但是思考了一下觉得目前还不会出现结构相关，而且也可以 pass，所以在第一阶段没有进行修改，但其实不严谨。想了想觉得老师的设计非常严谨，通过握手协议可以保证后面流水级阻塞后阻塞前面的流水级，在后续阶段将改成老师的风格。并且该风格可以通过设置 readygo 信号解决数据相关，细想之后觉得很容易实现，后续阶段会好好改的。

查阅资料的过程中发现了一个加入分支选择单元的设计，如图 6。

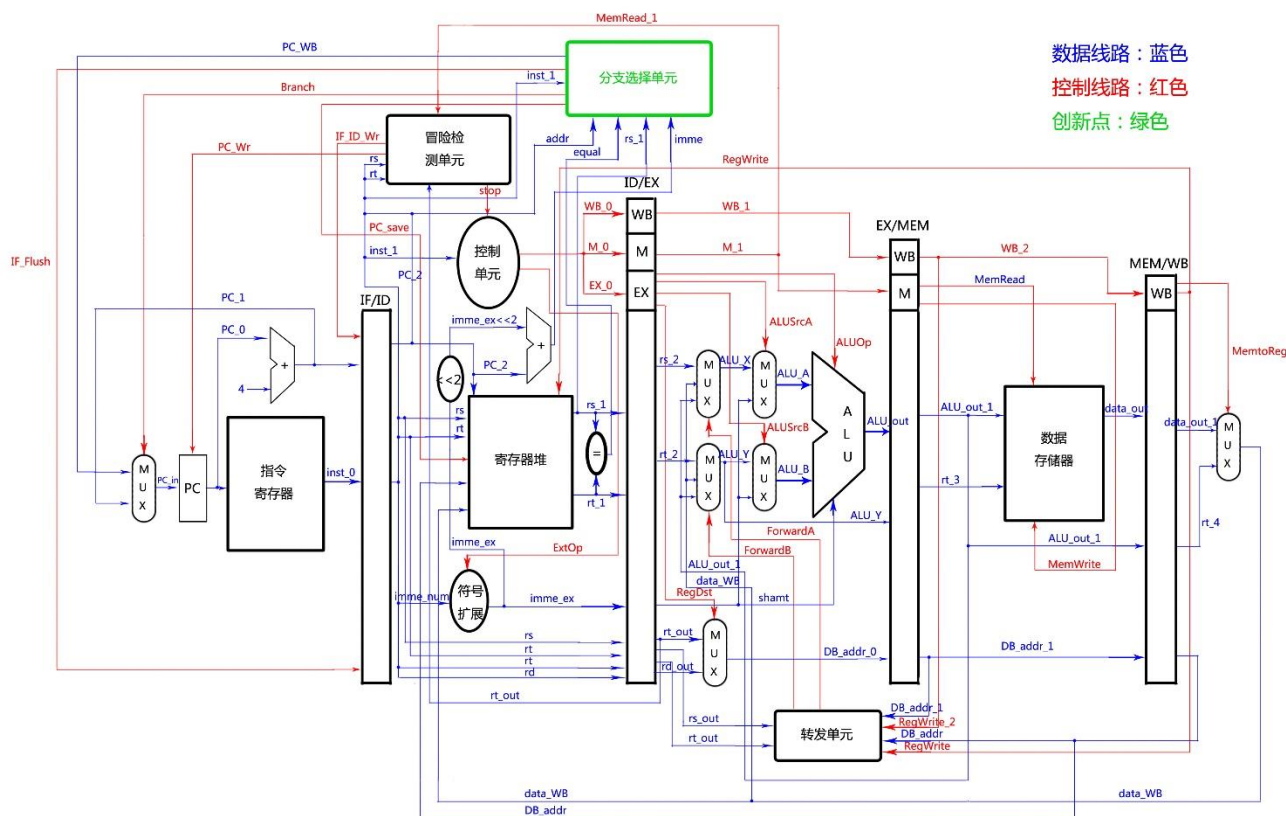


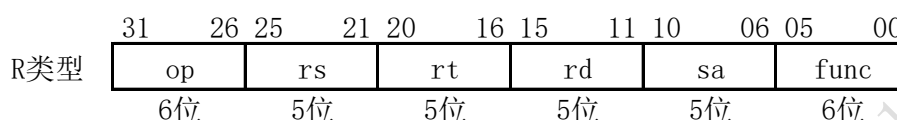
图 6. 分支选择单元设计

通过加入分支选择单元、冒险检测模块、转发单元来避免 hazard。看起来仿佛很妙（exe 阶段第一眼看上去有很多 MUX），可总模糊感觉冗余（不好写），为什么 exe 阶段需要使用那么多 mux，还需要额外加入三个单元？思考之后觉得大概是用粗略区分指令避免 hazard。那我们能不能在高层设计阶段就对指令进行划分？

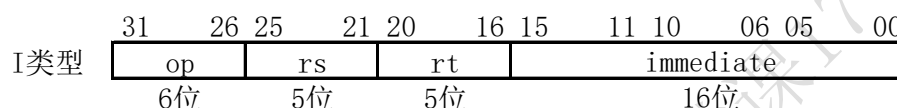
如果能在 instruction decode 阶段就区分 R/I 型指令、Branch 指令和 Jump 指令，事情应该就会简单得多。可这样仿佛离硬件电路又远了。

这样之后，把 MIPS 指令进行划分（想直接从全部指令都要载入的角度去看结构 orz，这是为什么看到老师给的 demo 想了半天还是没有照着写的原因...渣渣的确不会用那个结构去控制指令流动）：

R 型指令：



I 型指令：



J 型指令：

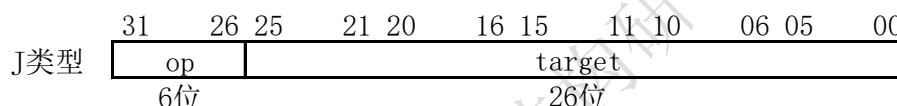


图 7. 指令类别

指令清楚之后考虑的是电路需要的器件。

1. 译码器。MIPS 寄存器堆中含  $2^5 = 32$  个通用寄存器，所以需要 5 位译码器。
2. 数据选择器 mux。分为 32 位 4 选 1，5 位 4 选 1。

【实现文件：Mux.v；模块：module Mux\_4\_32, module Mux\_4\_5】

3. 【实现文件：dm.v, im.v；模块：module dm\_4k, module im\_4k】。

4. alu（乘除法再单独写，乘法应该要实现有符号乘法和无符号乘法两种，除法器同理实现有符号除法和无符号除法两种），这里想用 alu 同时实现比较功能，即 SetLessThan，slt 指令。【实现文件：alu.v】

表 1. Aluop 信号对应

ALUop	运算	ALUop	运算
0000	sub	1000	sra 算术右移 rs 位
0001	or	1001	and
0010	add	1010	xor
0011	sll 逻辑左移 sa 位	1011	nor
0100	srl 逻辑右移 sa 位	1100	slt（不需经过 exe）
0101	sra 算术右移 sa 位		
0110	sllv 逻辑左移 rs 位		
0111	srlv 逻辑右移 rs 位		



5. 乘加、乘减器（从乘除法再变来，实现 MADD, MADDDU, MSUB, MSUBU 指令），羞耻地还没写！考虑要不要直接加进 ALU 里面。
6. 寄存器堆 GPR：32 个 32 位通用寄存器（其中 r0 被硬连线到 0，读 r0 始终为 0，写 r0 不会有任何影响。这样，如果某些指令需要丢弃运算结果，就可以把 r0 作为保存结果的目的寄存器。如果有些指令需要 0 为操作数，就可以把 r0 作为源数据寄存器）（r31 作为指令 JAL、BLTZAL、BGEZAL 在没有指定目的寄存器时的默认目的寄存器，也可以被用作通用寄存器）。单独放三个专用寄存器（程序计数寄存器 PC、乘法指令高位结果/除法指令余数寄存器 HI、乘法指令低位结果/除法指令商寄存器 LO）
- 【实现文件：GPR.v；模块：module RegFile】
- 【搭配文件：enable\_single\_register.v；模块：module enable\_single\_register】
7. 计算 next PC 的模块【实现文件：nPC.v；模块：module nPC】，以及载入 PC 的模块【实现文件：PC.v；模块：module PC】。
8. 输出数据扩展模块【实现文件：Dout\_extender.v；模块：module Dout\_extender】。

表 2. Extop 信号对应

extop	扩展方式
00	低位补零
01	零扩展
10	符号扩展

9. 当然要有顶层模块【实现文件：mips.v；模块：module mips】。
10. 控制模块最要命所以放在最后写【实现文件：ctrl.v；模块：module ctrl】
- 这个设计 ctrl 模块是关键！控制指令试图避免 hazard。

表 3. Wdsel 信号对应选择路径

Wdsel	选择路径
00	ALUout
01	Dout
10	pc+4

表 4. RegDst 信号对应选择路径

RegDst	选择路径
00	rt
01	rd
10	\$31

表 5. ALUsrc 信号对应选择路径

ALUsrc	选择路径
00	B
01	IMM
10	32' b0

表 6-1. 各条指令对应的控制信号

	PCWrite	IRWrite	WDSel	RegDst	EXTop	GPRWrite
lui	S0	S0	S4:ALUout	S4:rt	S2:低位补零	S4
addu	S0	S0	S4:ALUout	S4:rd		S4
addiu	S0	S0	S4:ALUout	S4:rt	S2:sign	S4

beq	S0+S6&zero=1	S0				
bne	S0+S6&zero=0	S0				
lw	S0	S0	S4:Dout	S4:rt	S2:sign	S4
or	S0	S0	S4:ALUout	S4:rd		S4
slt	S0	S0	S4:ALUout	S4:rd		S4
slti	S0	S0	S4:ALUout	S4:rt	S2:sign	S4
sltiu	S0	S0	S4:ALUout	S4:rt	S2:sign	S4
sll	S0	S0	S4:ALUout	S4:rd		S4
sw	S0	S0			S2:sign	
j	S0+S7	S0				
jal	S0+S7	S0	S7:PC_add_4	S7:\$31		S7
jr	S0+S7	S0				

PCWrite: 只有跳转指令可以不只是 S0

RegDst: R 型写回 rd 域, I 型写回 rt 域

表 6-2. 各条指令对应的控制信号

	ALUsrc	ALUop	nPCop	Dmwrite	din	dout
lui	S2:IMM	S2:or	S0:PC+4			
addu	S2:B	S2:add	S0:PC+4			
addiu	S2:IMM	S2:add	S0:PC+4			
beq	S2:B		S0:PC+4  ((S6&&zero=1):IMM_16			
bne	S2:B		S0:PC+4  ((S6&&zero=0):IMM_16			
lw	S2:IMM	S2:add	S0:PC+4			S3:none
or	S2:B	S2:or	S0:PC+4			
slt	S2:B	S2:slt	S0:PC+4			
slti	S2:IMM	S2:slt	S0:PC+4			
sltiu	S2:IMM	S2:slt	S0:PC+4			
sll	S2:B	S2 : sll	S0:PC+4			
sw	S2:IMM	S2:add	S0:PC+4	S5	S5:4'b1111	
j			S0:PC+4  S7:IMM_26			
jal			S0:PC+4  S7:IMM_26			
jr			S0:PC+4  S7:rs			

din、dout: 都在 controller 里。

### 三、实验过程（60%）

#### （一）实验流水账

付琳晴:








10月3日: 查阅《计算机组成与设计: 硬件软件接口》里的流水线章节。将多周期 cpu 改成单周期 cpu, 并重新规范了所有控制信号, 代码交给队友写。

10月7日: 发现队友写的代码跑的 pass 用的是老师的 trace 代码, 按照老师提供的模板重写代码。




10月8日: 代码完成, 开始 debug。

10月9日: debug 完成。





















李舒博：除了 Patterson 那本是 pdf，其他的大部分是翻墙乱看的 orz 直接截 chrome history 的图了  
10月3日：没有意识到写 cpu 麻烦程度的

- ☐ 上午11:32  verilog cpu tutorial - YouTube www.youtube.com
- ☐ 上午11:32  https://www.youtube.com/results?search\_query=verilog%20cpu%20tutorial%20 www.youtube.com
- ☐ 上午11:32  YouTube www.youtube.com
- ☐ 上午11:27  Building a CPU on an FPGA, part 1 - YouTube www.youtube.com
- ☐ 上午11:27  cpu verilog - YouTube www.youtube.com
- ☐ 上午11:27  https://www.youtube.com/results?search\_query=cpu%20verilog www.youtube.com
- ☐ 上午11:26  写cpu - YouTube www.youtube.com

10月4日：开始在 csdn 上看怎么写

- ☐ 上午8:38  数字电路设计之五级流水线设计 ( CPU ) - I AM BACK - CSDN博客 blog.csdn.net
- ☐ 上午8:38  https://www.baidu.com/link?url=MMYmFT6xwiJH7Lz5JWpQow8FmniGboBuovJNhXUWygEOIM9... www.baidu.com
- ☐ 上午8:38  五级流水\_百度搜索 www.baidu.com
- ☐ 上午8:38  百度一下，你就知道 www.baidu.com

然后开始慌了



















- ☐ 下午4:24  首页-中国科学院大学综合信息网 onestop.ucas.ac.cn
- ☐ 下午4:13  4729022[正版现货]计算机组成与设计硬件/软件接口(原书第5版)/计算机教材/计算机科学丛书-tmall... detail.tmall.com
- ☐ 下午4:13  https://s.taobao.com/search?initiative\_id=tbindexz\_20171004&ie=utf8&spm=a21bo.50862.201856... s.taobao.com
- ☐ 下午4:13  计算机组成与设计硬件软件接口\_淘宝搜索 s.taobao.com
- ☐ 下午4:02  readme.md - OneDrive onedrive.live.com
- ☐ 下午3:50  执行级 - OneDrive onedrive.live.com
- ☐ 下午3:50  研讨课 - OneDrive onedrive.live.com
- ☐ 下午3:49  计算机体系结构 - OneDrive onedrive.live.com
- ☐ 下午3:49  计算机体系结构 - OneDrive onedrive.live.com
- ☐ 下午3:49  onedrive.live.com onedrive.live.com
- ☐ 下午4:32  Lab03\_静态5级流水简单MIPS CPU实现.pdf course.ucas.ac.cn
- ☐ 下午4:32  A07\_交叉编译工具链安装.pdf course.ucas.ac.cn
- ☐ 下午4:32  LEC04\_仿真调试说明.pdf course.ucas.ac.cn
- ☐ 下午4:31  LEC03\_CPU实验开发环境使用说明.pdf course.ucas.ac.cn
- ☐ 下午4:25  Course: 计算机体系结构研讨课17-18秋季: Assignments course.ucas.ac.cn
- ☐ 下午4:24  Course: 计算机体系结构研讨课17-18秋季: Resources course.ucas.ac.cn
- ☐ 下午4:24  Course: 计算机体系结构研讨课17-18秋季: Home course.ucas.ac.cn
- ☐ 下午4:24  课程网站 course.ucas.ac.cn
- ☐ 下午4:24  SEP 教育业务接入平台 sep.ucas.ac.cn
- ☐ 下午4:24  SEP 教育业务接入平台 sep.ucas.ac.cn

晚上确定了用状态机写，把要写的 MIPS 指令的状态跳转做了，大致想了想要写哪几个模块，把图画



了。

10月5日:

<input type="checkbox"/>	下午8:48		YouTube	www.youtube.com
<input type="checkbox"/>	下午8:48		<a href="https://www.youtube.com/results?search_query=5%20stage%20pipeline">https://www.youtube.com/results?search_query=5%20stage%20pipeline</a>	www.youtube.com
<input type="checkbox"/>	下午8:46		1 Introduction to MIPS datapath - YouTube	www.youtube.com
<input type="checkbox"/>	下午8:44		21 Pipeline Control Summary in MIPS Datapath - YouTube	www.youtube.com
<input type="checkbox"/>	下午8:36		20 Pipelined Control with Interstage Buffers in MIPS Datapath - YouTube	www.youtube.com
<input type="checkbox"/>	下午8:32		19 Code Scheduling to Avoid Inserting nop stalls in MIPS Datapath - YouTube	www.youtube.com
<input type="checkbox"/>	下午8:28		18 Forwarding and Load Use Data Hazard in MIPS Datapath - YouTube	www.youtube.com
<input type="checkbox"/>	下午8:14		17 Data Hazard Example With add and sub instruction in MIPS Datapath - YouTube	www.youtube.com
<input type="checkbox"/>	下午8:04		16 Introduction to Piplining in MIPS Datapath Adding IF, ID, EX, MEM, WB Stages - YouTube	www.youtube.com
<input type="checkbox"/>	下午7:56		15 How J Type Jump Instruction is Executed on MIPS Datapath - YouTube	www.youtube.com
<input type="checkbox"/>	下午7:48		rtl_led - OneDrive	onedrive.live.com
<input type="checkbox"/>	下午7:48		lab_0_20161010 - OneDrive	onedrive.live.com
<input type="checkbox"/>	下午5:03		14 Why Do We Add Shift Left 2 in Beq Instruction in MIPS Datapath - YouTube	www.youtube.com
<input type="checkbox"/>	下午4:50		13 Executing Beq Instruction on MIPS Datapath - YouTube	www.youtube.com
<input type="checkbox"/>	下午4:45		12 Setting Control Lines RegDst, MemRead, MemtoReg, MemWrite, ALUOp, ALUsrc, RegWrite in...	www.youtube.com
<input type="checkbox"/>	下午4:33		11 Fixing MIPS Datapath with Multiplexors - YouTube	www.youtube.com
<input type="checkbox"/>	下午4:30		10 Adding Instruction Fetch to our Partial MIPS Datapath - YouTube	www.youtube.com
<input type="checkbox"/>	下午1:35		9 Executing an I Type Instruction In MIPS Datapath - YouTube	www.youtube.com
<input type="checkbox"/>	下午1:30		7 How to Fetch the Next Instruction from Memory - YouTube	www.youtube.com
<input type="checkbox"/>	下午1:26		6 ALU in MIPS Data Path - YouTube	www.youtube.com
<input type="checkbox"/>	下午1:19		5 Register File in MIPS Dath Path - YouTube	www.youtube.com
<input type="checkbox"/>	下午1:06		4 add Assembly Instruction to add Machine Instruction in MIPS PART2 - YouTube	www.youtube.com
<input type="checkbox"/>	下午12:54		3 MIPS Assembly to Machine Language PART 1 - YouTube	www.youtube.com
<input type="checkbox"/>	下午12:50		2 MIPS Memory Organization And Review of Language to Assembly to Machine Language - Yo...	www.youtube.com
<input type="checkbox"/>	下午12:47		computer engineering - YouTube	www.youtube.com
<input type="checkbox"/>	下午12:39		9 Executing an I Type Instruction In MIPS Datapath - YouTube	www.youtube.com
<input type="checkbox"/>	下午12:29		16 Introduction to Piplining in MIPS Datapath Adding IF, ID, EX, MEM, WB Stages - YouTube	www.youtube.com
<input type="checkbox"/>	下午12:29		Pipelines in MIPS processors - Overview - YouTube	www.youtube.com
<input type="checkbox"/>	下午12:28		5-Stage Pipeline Processor Execution Example - YouTube	www.youtube.com

视频历史太多了不截图了..orz

想了想 hazard 啥的，还是没什么头绪，希望状态机能解决这个问题 orz。昨天看视频看得很慌，就终于开始勤奋地写 bug。因为是从头做所以先写的肯定是 alu 和 regfile，enable\_single\_register。之后因为 mux 好写！就把 mux 写了。之后写的肯定是 extender！再之后写 PC 和 nPC。写 ctrl 写得很烦。

10月6日：看了看 piazza 之后想起了小伙伴和我讲过有 demo，本来开心到爆炸，但是看完感觉和前几天想出来的思路不一样（渣渣理解不到妙处不会写 只能用 YouTube 助攻硬上），在 csdn 和 github 上看别人的源码怎么搭的结构

10月7日：早上跑了板子。昨晚没敢跑因为怕 simulation 过了 bitstream 不过...怂。发现跑到 0014 就绿了又开始怕。想去 piazza 上问来着，可是发现之前应该有做完的人可是并没有人问这个问题，大概是自己睡傻了或者怎么样。后来想了想 simulation 只跑了 15 个 pass...不知道自己脑子里进的什么水。然后写了一点实验报告就开始摸鱼。

10月8日：小伙伴说接口没规范...可能是 trace 文件让它跑过的。吓得翻 trace...看到名字叫 goldentrace 的时候一声哀嚎。查了查发现是追踪文件？问了助教老师发现和 trace 其实没什么关系。自己还奇怪地避开了 hazard...

10月9日：开始重新写 cpu 血泪

10月10日：下午四点还在 simulation debug

## (二) 错误记录

### 1、错误 1

#### (1) 错误现象

```
1 module nPC (rs, imm_26, imm_16, PC, nPCop, nPC, PC_add_4);
2   input [31:0] rs;
3   input [25:0] imm_26;
4   input [15:0] imm_16;
5   input [31:0] PC;
6   input [1:0] nPCop;
7   output [31:0] nPC; //output to IM
8   output [31:0] PC_add_4; //output to $31
9   //zero and IFbeq are transmitted to Controller
10
11   assign PC_add_4 = PC; // it's different from single-cycle cpu
12
13   assign nPC = (nPCop == 2'b00)?(PC+h4):
14               (nPCop == 2'b01)?(PC[31:28],imm_26,2'b00):
15               (nPCop == 2'b10)?(PC+({14{imm_16[15]}},imm_16,2'b00));
16 endmodule
17
```

#### (2) 分析定位过程

非常客气 直接标出来了红波浪线

#### (3) 错误原因

rs 情况没有考虑

#### (4) 修正效果

在之后加 rs;

#### (5) 归纳总结（可选）

大概是智障类型的吧

### 2、错误 2

#### (1) 错误现象

```
1 module ALU(A,B,sa,C,ALUop,zero,big,smal);
2   input unsigned [31:0] A,B; //????
3   input unsigned [4:0] sa;
4   output unsigned[31:0] C;
5   input [3:0] ALUop;
6   output zero;
7   output big,smal;
8
```

#### (2) 分析定位过程

自己不确定 ALU 的输入 A,B,C 是 signed 还是 unsigned (加了个 sa 输入信号就不知道其他的该怎么办了???)

### (3) 错误原因

ALU 操作有符号和无符号数, 输入应该用什么?

### (4) 修正效果

用了 signed

### (5) 归纳总结 (可选)

不够了解, 改完记住了。

## 3、错误 3

### (1) 错误现象

```
24 type, addu, subu, ori, lv, sv, beq, lui, addi, addiu, slt, j, jal, jr, lb, lbu, lh, lhu, sb, sh, slti;  
25 dd, sub, sll, srl, sra, sllv, srlv, srav, AND, OR, XOR, NOR, andi, xori, sltiu, bne, blez, bgtz, bltz;  
26  
27  
28 add = (Rtype&&func==b100000); //it's better to be included in "head.v"  
29 sub = (Rtype&&func==b100010);  
30 sll = (Rtype&&func==b000000);  
31 srl = (Rtype&&func==b000010);  
32 sra = (Rtype&&func==b000011);  
33 sllv = (Rtype&&func==b000100);  
34 srlv = (Rtype&&func==b000110);  
35 srav = (Rtype&&func==b000111);  
36 AND = (Rtype&&func==b100100);  
37 OR = (Rtype&&func==b100101);  
38 XOR = (Rtype&&func==b100110);  
39 NOR = (Rtype&&func==b100111);  
40 andi = (op==b001100);  
41 xori = (op==b001110);  
42 sltiu = (op==b001011);  
43 bne = (op==b000101);  
44 blez = (op==b000110&&rt==b000000);
```

### (2) 分析定位过程

应该算是失误吧...觉得自己写的时候拼命想贴电路, 文件数量 (比起老师的 demo 文件数量) 本来就有点多...extender 都分在两个里面写, 想在 control 模块里解决一切, 反而看起来炸裂。

### (3) 错误原因

偷懒没写 header

### (4) 修正效果

羞耻地尚未修正。

### (5) 归纳总结 (可选)

请努力写 header, 用 include 很好的。

## 4、错误 4

### (1) 错误现象



### (2) 分析定位过程

之前的模块都没啥错了, 放心大胆 (小心翼翼 orz) 写一贯容易写漏写错的顶层 mips.v。没什么好方法, 就对着图数元件...非常笨了, 看了三遍。

### (3) 错误原因

寄存器是两个 微笑

#### (4) 修正效果

```
enable_single_register regA(clk,RD1,Aout,1'b1);  
enable_single_register rebB(clk,RD2,Bout,1'b2);
```

#### (5) 归纳总结 (可选)

我也不知道怎么避免顶层写漏...对着图写还是漏...查三遍才看出来 非常漏了。不过现在起码知道自己这儿会出错了? orz 无耻地自我辩解

### 5、错误 5

#### (1) 错误现象

mycpu 的 pc 信号与 wb\_rf\_wnum、wb\_rf\_wdata 信号错位。如图

```
Test begin!  
  
[ 2105 ns] Error!!!  
reference: PC = 0xbfc0038c, wb_rf_wnum = 0x04, wb_rf_wdata = 0x00000000  
mycpu      : PC = 0xbfc00000, wb_rf_wnum = 0x04, wb_rf_wdata = 0x00000004
```

#### (2) 分析定位过程

首先, wb\_rf\_wnum 信号对应正确, wdata 信号不正确, 说明这两个信号已经发生错位, 而 wb\_rf\_wnum 信号与 reference 信号对应, 说明正确。查看代码发现 writeback\_stage 模块中传入的 mem\_value 并没有用寄存器存一下, 直接就用, 因此 wire 信号当前周期瞬间传出去。而 wb\_rf\_waddr 用寄存器保存了是因为老师给的模板中定义了 reg 类型用来保存目的寄存器的信号作为提示。

纠正过 wb\_rf\_wnum、wb\_rf\_wdata 信号后进行仿真, 发现 pc 信号与这两个信号又一次错位。pc 取的是前一条指令的 pc, 但是想到 reference 信号变化是由于写回级处需要写寄存器, 而且 debug 用的 pc 信号在五级模块中都有对应的寄存器存, 是严格按照周期一拍一拍传下去的, 说明不是 pc 信号传晚了, 是 wb\_rf\_wnum、wb\_rf\_wdata 信号提早传出来了, 在 memory\_stage 模块时已经得到 wb\_rf\_wen 等一系列信号导致 reference 提前有效, 但这里仍不是信号提前传出的根源。

继续沿着流水线往回追溯直到在 decode\_stage 模块中发现, 所有控制信号和数据信号都是 wire 类型, 译码得到后直接输出, 也就意味着后面的几级流水全部提前了一个周期, 至此找到根因。

#### (3) 错误原因

出现的错误全部是由于在每个模块中, 输入信号没有全部先存一下再用, 导致 wire 类型信号直接接出。

#### (4) 修正效果

最开始修改的很凌乱。发现 decode\_stage 模块中有错误后仅补救式地将 decode\_stage 模块中输出给 execute\_stage 模块的信号均存一下再输出, 虽然出发点是让信号停一个周期再传给后一级, 但其实代码很乱。后意识到五个模块中全存在这个问题, 统一作出修改: 将每个模块中的输入信号全部用寄存器保存一遍, 再在该模块中使用。至此代码变得非常有规律。考虑了 data\_ram 是同步的, 取数据会晚一拍这个问题, 发现不需要考虑, 因为在我的设计中所有寄存器加在每个模块的前面, 意味着 data\_ram 的读信号将在 exe 级输出, 正好在 mem 级取回。

#### (5) 归纳总结

事实上老师在 piazza 上已经再三强调过这个问题, 但自己开始写的时候还是没有充分理解老师的意思。虽然考虑到可能会出现不同指令的控制信号、数据信号由于周期没有严谨对上发成冲突, 但还是会写错。事实上最终按照上述方法修改过后, 终于意识到老师给的模板和各种资料上的架构没有任何差别, 只是资料上两级之间有一个很多位的寄存器来保存信号数据, 而老师的架构是在模块内部用多个寄存器保存传入数据。而代码改过之后再看老师的提示豁然开朗。

## 6、错误 6

### (1) 错误现象

reg 读数据时无法立即返回值，波形显示高阻态。

### (2) 分析定位过程

查看波形发现寄存器在上个周期要向寄存器内写值，在下个周期才会写进去，但此时对寄存器进行了读操作，因此没有读出来，这个问题看似是个数据相关，但这前后两个指令其实已经岔开了很多，不该出现数据相关，意识到一定是对寄存器读早了，继续查看波形发现对寄存器读信号竟然发生在取值阶段，即五级流水的第一阶段。因为 nextpc 模块中 inst\_sram\_addr 信号已经提前送出，保证了在取值阶段当前指令已经取回，而 decode 模块迅速将取回的指令译码并读寄存器。由此意识到还是上述问题，输入信号没有全部先存一下再用，提前读了寄存器。

### (3) 错误原因

出现的错误全部是由于在每个模块中，输入信号没有全部先存一下再用，导致 wire 类型信号直接接出。

### (4) 修正效果

在 decode 模块中认真的把所有输入信号都加了寄存器后仿真，发现神奇地实现了延迟槽技术。细想之后意识到确实如此，因为修改过后 decode\_stage 中输给 nextpc 模块信号都是在第二个周期才传回去，也意味着将影响的是第二条指令即延迟槽指令的 nextpc，因此若发生跳转也会作为第三条指令执行，顺利地执行完延迟槽再执行跳转。

### (5) 归纳总结

以上两个错误证明了仔细阅读老师的建议是多么的重要。不过在写代码或者没有真正 debug 前，其实并不能理解老师给出的工程性建议意味着什么，而自己在 debug 过程中才能深深体会这些细节以及老师意见的宝贵性。

## 7、错误 7

### (1) 错误现象

写入目的寄存器的数据源选择错误，原本应该选择 data\_sram\_rdata，但结果选择了 alu 的计算结果。

### (2) 分析定位过程

发现是因为信号选择错误由此定位到 memory\_stage 模块中目的寄存器多选器处，原来错误代码如下图。

```
96 always @(posedge clk) begin
97     if (!resetn) begin
98         mem_st_value <= 32'b0;
99     end
100     else begin
101         mem_st_value <= (mem_out_op == 6'b100011)?data_sram_rdata:exe_value;//will change until 5th stage
102     end
103 end
104
```

首先，最明显的错就是直接使用了传入的信号作为一个数据源，导致两个数据源本身时钟就不统一。将错误的数据源更改为经过寄存器保存后的数据信号。

其次，代码的这种写法很危险。我不了解在时钟上升沿时用 mem\_out\_op 进行判断时，依照的是其上个时钟的值还是这个时钟的，因此这样写总觉得不踏实，索性换了一种写法，如下图。

```
104 */
105 assign mem_value = (mem_out_op == 6'b100011)?data_sram_rdata:mem_value_wire;//mem_st_value;
106
107
```

### (3) 错误原因

直接使用了模块的输入信号，导致两个数据错位。

#### (4) 修正效果

保证了当前选择的两个数据源在正确的时钟同时有效，而且用 assign 心里更踏实。

#### (5) 归纳总结

以上三个错误全部是由于错误地使用了输入信号，归根结底都是一个错误，但产生了千奇百怪的错误结果。

### 8、错误 8

#### (1) 错误现象

pc 跳转错误，多跳了一个指令。

#### (2) 分析定位过程

根据 pc 地址找到执行的跳转指令为 bne，查看波形发现 nextpc 值错误，比应该跳到的地址多了 4，即多出一条指令。定位到文件 nextpc\_gen 模块中，查看 nextpc 的赋值，原错误代码如下图。

```
70 assign nextpc = (  
71     (!resetrn)?32'hbfc00000:  
72     (de_br_taken & de_br_is_br)?fe_pc + 4 + sign_ex_left:// + {14{de_br_offset[15]},de_br_offset,2'b00}://beq bne  
73     (de_br_taken & de_br_is_j)?{pc4[31:28],de_br_index,2'b00}: //j jal  
74     (de_br_taken & de_br_is_jr)?de_br_target: //jr  
75     fe_pc+4  
76 );  
77
```

看到代码中的加 4 后恍然大悟。在前面问题中已经提到，decode\_stage 中输给 nextpc 模块控制信号和 pc 信号都是在第二个周期才传回去，也意味着此时传入的 pc 是延迟槽指令地址。

#### (3) 错误原因

nextpc 计算时 pc 多加了 4，忘记了此时传入的 pc 已经是延迟槽指令的 pc，即已经加过 4。

#### (4) 修正效果

更改了两种跳转情况的 nextpc 信号赋值，第一种是 beq 和 bne，第二种是 j 和 jal。修改后如下图。

```
62 assign nextpc = (  
63     (!resetrn)?32'hbfc00000:  
64     (de_br_taken & de_br_is_br)?fe_pc + sign_ex_left:// + {14{de_br_offset[15]},de_br_offset,2'b00}://beq bne  
65     (de_br_taken & de_br_is_j)?{fe_pc[31:28],de_br_index,2'b00}: //j jal  
66     (de_br_taken & de_br_is_jr)?de_br_target: //jr  
67     fe_pc+4  
68 );  
69
```

#### (5) 归纳总结

在上学期查看 MIPS 手册时很疑惑为什么 j、jal、beq、bne 的跳转地址高位用的是当前 pc 加 4 后的高位，即延迟槽指令 pc 的高位，当时对延迟槽不理解。如今意识到实现延迟槽技术是在实现五级流水过程中顺理成章的事情。

### 9、错误 9

#### (1) 错误现象

第一次跑仿真时失败，报错找不到 de\_pc 和 de\_inst 接口，以及报错找不到各个模块。报错如下图。



Simulation (13 errors)

sim\_1 (13 errors)

- [VRFC 10-426] cannot find port de\_inst on this module [mycpu\_top.v:172] (1 more like this)
- [VRFC 10-2063] Module <Control> not found while processing module instance <contr1> [decode\_stage.v:131] (9 more like this)
  - [VRFC 10-2063] Module <ALUcontrol> not found while processing module instance <aluctrl> [decode\_stage.v:146]
  - [VRFC 10-2063] Module <execute\_stage> not found while processing module instance <exe\_stage> [mycpu\_top.v:177]
  - [VRFC 10-2063] Module <memory\_stage> not found while processing module instance <mem\_stage> [mycpu\_top.v:207]
  - [VRFC 10-2063] Module <writeback\_stage> not found while processing module instance <wb\_stage> [mycpu\_top.v:231]
  - [VRFC 10-2063] Module <regfile\_2r1w> not found while processing module instance <regfile> [mycpu\_top.v:252]
  - [VRFC 10-2063] Module <inst\_ram> not found while processing module instance <inst\_ram> [soc\_lite\_top.v:143]
  - [VRFC 10-2063] Module <bridge\_1x2> not found while processing module instance <bridge\_1x2> [soc\_lite\_top.v:153]
  - [VRFC 10-2063] Module <data\_ram> not found while processing module instance <data\_ram> [soc\_lite\_top.v:177]
  - [VRFC 10-2063] Module <confreg> not found while processing module instance <confreg> [soc\_lite\_top.v:188]
- [XSIM 43-3322] Static elaboration of top level Verilog design unit(s) in library work failed.

## (2) 分析定位过程

仔细看了代码觉得没有问题，de\_pc 和 de\_inst 都进行了定义，而且已经注意到了放在`ifdef SIMU\_DEBUG 内。需要的模块文件都进行了添加，其文件分层等级在列表中均正确对应。最后索性把所有的 ifdef 块注释掉。

## (3) 错误原因

至今无解。怀疑是因为在各个模块没有再单独加入`def SIMU\_DEBUG 导致的，但是作为宏定义，就应该保证各个文件里都可以用啊，很奇怪。

## (4) 修正效果

不再报错。

## 10、错误 10

### (1) 错误现象

第一次跑仿真时所有信号在 resetn 之后变成高阻态或 0，导致 debug\_wb\_pc = 0xxxxxxxxx。

### (2) 分析定位过程

出错说明无法正确找到下一条指令，即 pc 初始化后无法传给 nextpc，并且 nextpc 也没有顺利地传给 pc。进而发现是因为仅对 pc 信号做了初始化，而 nextpc 没有初始化。

### (3) 错误原因

nextpc 未初始化，因此传给 pc 是高阻态，因此无法正确取回指令。

### (4) 修正效果

pc 等信号终于有波形。

## 11、错误 11

### (1) 错误现象

ERROR: [VRFC 10-91] fe\_pc is not declared [E:/CA/vivado/lab03\_lsb/02\_decode\_stage.v:75]

## (2) 分析定位过程

fe\_pc 未定义，说明 simu debug 模式有问题，查了小伙伴的 bug 过程发现可以把所有`ifdef 注释掉。

## (3) 错误原因

simu debug 模式可能没有开？或者接口有问题？其实在 top 文件里有定义但是？

## (4) 修正效果

不再报错。

## 12、错误 12

### (1) 错误现象

```
EERROR: [VRFC 10-2592] cannot index into non-array type wire for sign_extention  
[E:/CA/vivado/lab03_lsb/02_decode_stage.v:263]
```

### (2) 分析定位过程

见 (1)

### (3) 错误原因

未声明 wire [31:0] sign\_extention

### (4) 修正效果

不再报错。

## 13、错误 13

### (1) 错误现象

跑仿真时，debug\_wb\_pc = 0xxxxxxxxx

### (2) 分析定位过程

小伙伴说可能是 nextpc 没有初始化，可是已经初始化过了

### (3) 错误原因

尚未解决。

### (4) 修正效果

尚未解决。

## 四、实验总结（可选）

### （一）组员：付琳晴

这次老师给的模板好良心！原本毫无头绪，假期把多周期 cpu 改成单周期试图找点头绪，看到模板之后觉得简化成了连线题。写流水线前觉得这次代码量好大啊根本没法写完，但是按照模板规范完成后觉得确实简单不少。写代码加 debug 一共用了两天时间，很感人。发现自己能看懂波形，大概原因得益于有每级的 pc 信号而且每级信号都有加标注对应哪个阶段，debug 时能很快定位出错的阶段。说到底还是老师给的模板架构很完美。

有个小建议，希望体系结构的理论课能和实验课相结合。上学期计算机组成原理的课程安排很舒服，理论课老师会提前讲下一周实验课要做的内容，硬件结构图讲的很清楚，因此代码写起来很快，因为理解通透。这学期可能是因为两个课老师不同，感觉理论课和实验课是脱节的，流水线的知识都在自己查资料。缺少老师讲的环节导致我在写代码时候有点晕晕的，尤其是最开始摸不着头脑，无从下手，不过写完后自己确实完全理解了。希望老师能和理论课老师沟通一下，或者希望老师能在以后实验课上对要做的内容讲的更清晰一点。

以及，感觉现在每次发布的任务书有些要求很含糊，总是布置要实现什么功能，但是没有提示为了实现该功能要怎么做，导致每次刚开始着手写代码一脸茫然，不知所措。虽然我非常理解老师和助教这样做的原因，而且写出来代码再看任务书会有种大彻大悟的感觉，但是这种体验感真的太崩溃了。

---

## （二）组员：李舒博

是个重感冒发低烧强行写出来的 CPU...收获很大。坦白讲之前组成原理没觉得 CPU 很厉害（我知道每天都在用）...也没觉得很优美之类。看了很多视频（好的渣渣）想了很久才知道美在哪里。不过还是不知道 hazard 怎么处理，以及好像做完了好几次的？不知道下次实验报告能不能复制粘贴...

队友能读懂老师的 demo！想学！（想重新写！说得像积极不摸鱼又有时间一样）

休完 2017 最后假期的程序媛。希望这个实验之后的下个实验留活路。谢谢老师。先跪为敬 orz。

谢谢小伙伴告诉写的第一个智障 cpu！在周一下午开始重新写！实验课下午四点还在跑仿真 TuT 啊先交实验报告。