中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2020K8009907032 姓名：唐嘉良 专业：计算机科学与技术

实验序号：04 实验名称：定制RISC-V功能型处理器设计

注1：撰写此Word格式实验报告后以PDF格式保存在~/COD-Lab/reports目录下。文件命名规则：prjN.pdf，其中“prj”和后缀名“pdf”为小写，“N”为1至4的阿拉伯数字。例如：prj1.pdf。PDF文件大小应控制在5MB以内。此外，实验项目5包含多个选做内容，每个选做实验应提交各自的实验报告文件，文件命名规则：prj5-projectname.pdf，其中“-”为英文标点符号的短横线。文件命名举例：prj5-dma.pdf。具体要求详见实验项目5讲义。

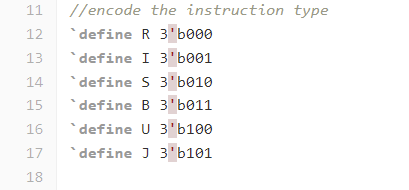
注2：使用git add及git commit命令将实验报告PDF文件添加到本地仓库master分支，并通过git push推送到GitLab远程仓库master分支（具体命令详见实验报告）。

注3：实验报告模板下列条目仅供参考，可包含但不限定如下内容。实验报告中无需重复描述讲义中的实验流程。

1. **逻辑电路结构与仿真波形的截图及说明（比如关键RTL代码段{包含注释}及其对应的逻辑电路结构图、相应信号的仿真波形和信号变化的说明等）**

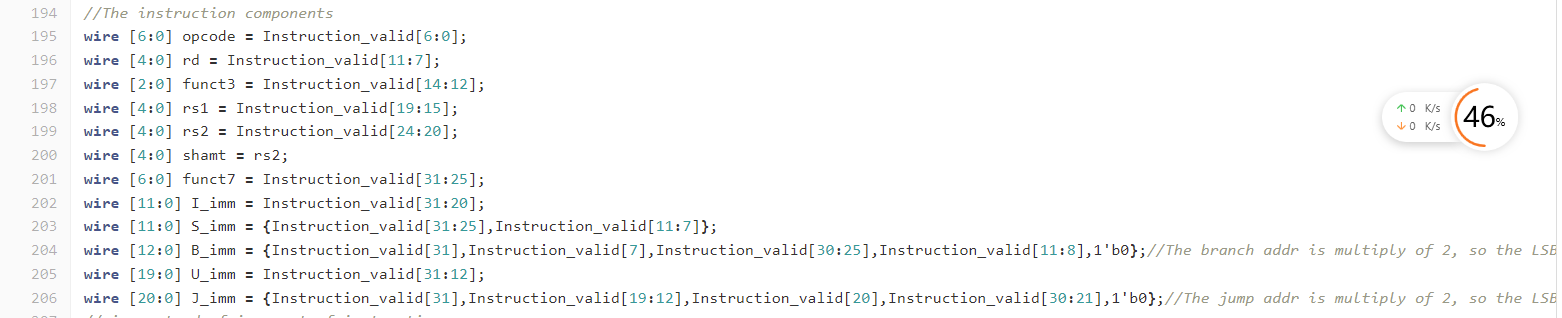
本实验需要修改的地方并不是太多，最主要的就是指令译码、ALUop与RF\_wdata等信号的修改。先看译码！

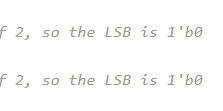
RISC-V指令格式已经足够工整，结合指令手册上的指令分类，可以把指令分为以下几类（这里通过定义宏来表示）：



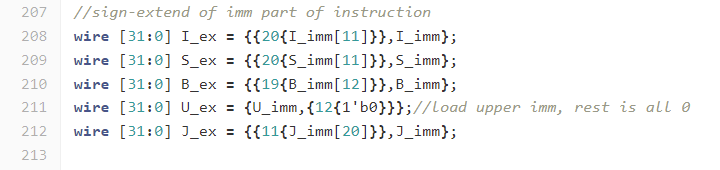
这在一定程度上减轻了写代码的负担。但是在硬件层面具有一些劣势，将在下面详细说明。

在指令分类之前，我们先解析指令：

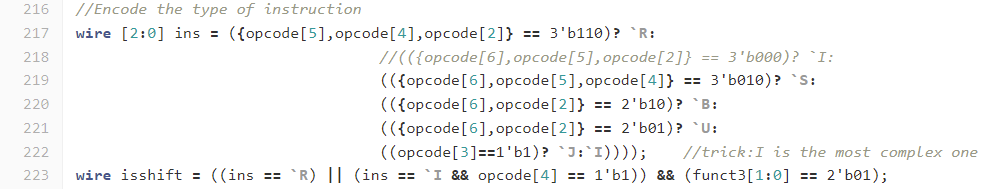




可以看到，我们利用\_imm后缀的几个变量提取并暂时存放指令中的立即数部分，并成功将所有可能的部分提取了出来。这里只要对照指令组成写即可，注意到B和J类型指令的LSB补0。

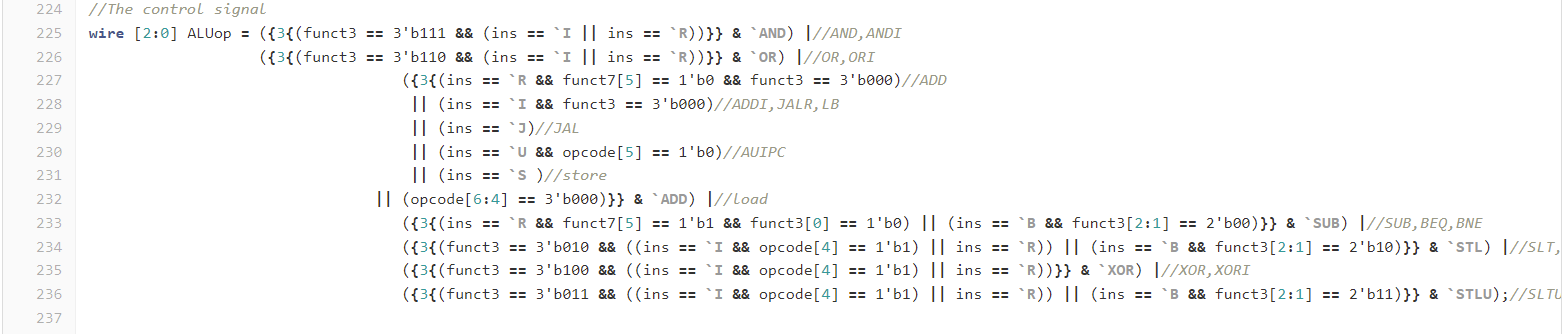


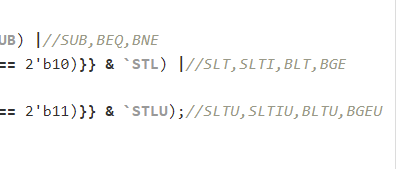
随后进行指令对应操作数的确定。这里注意：ISBJ四大类指令都是符号扩展，而U类指令则是低12位填充全0。



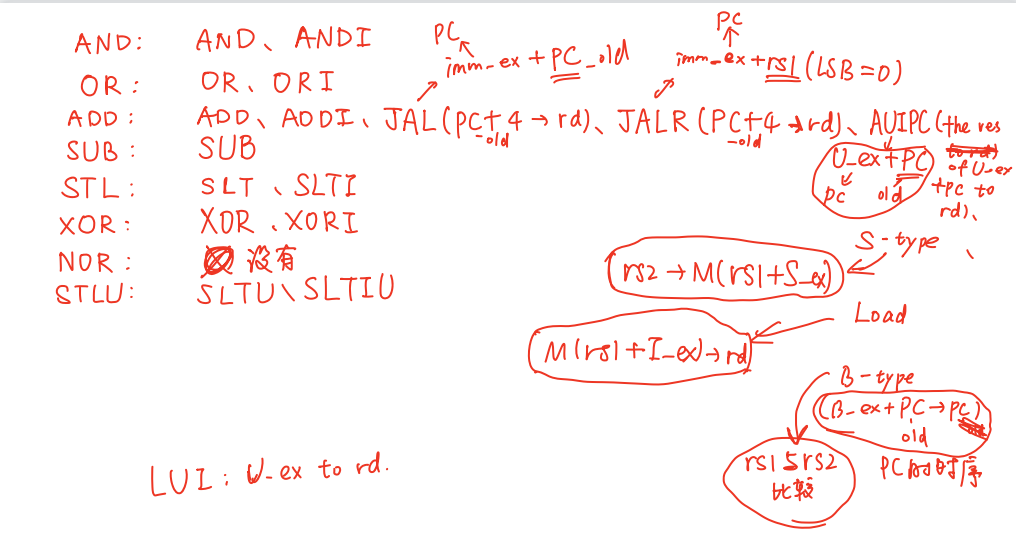
在指令译码部分，我们根据指令的opcode码特征进行了分类，并对指令是否是移位指令进行了判断。这里用到了一个小技巧：三目运算符最后一个判断是前面条件之并取补，那么可以把译码最复杂的I类指令放在最后判断，减少了一定工作量和代码量，让整体代码风格更加简洁易读。

之所以要判断是否是移位指令，是因为移位指令比较繁，这里拿出来单独判断，方便后续工作。

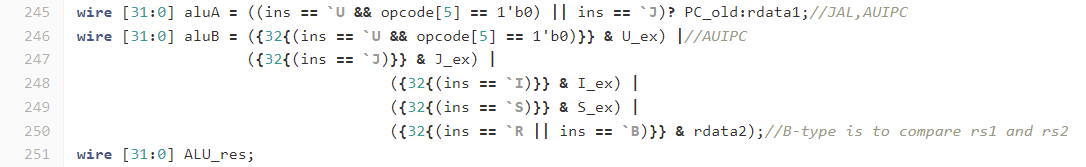




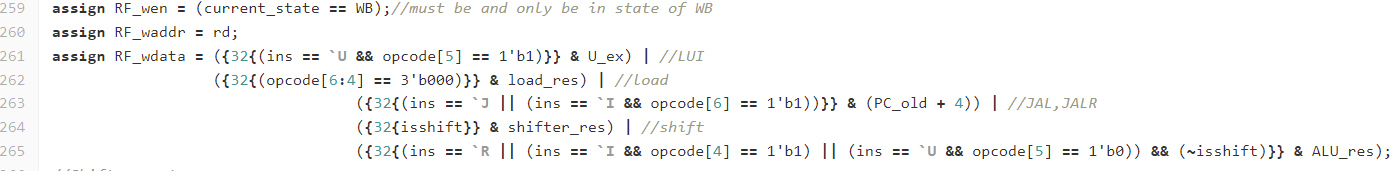
这里（ALUop的计算）是主要工作量所在，每个ALUop对应的指令已经在注释中指明，其中ADD最为复杂。为此，我总结了一张粗略的表如下：



借助这张表以及一些归纳，我将一些同类指令进行了合并判断，减少了一些判断逻辑和代码量。这里就能看到之前宏定义的缺陷：在ins判断这里每次都要进行三位判断！事实上，如果将ins归纳换为各类归纳，每次只需要进行一位判断。这里就可以看出对硬件并没有那么友好，仅仅是代码更清晰易读而已。

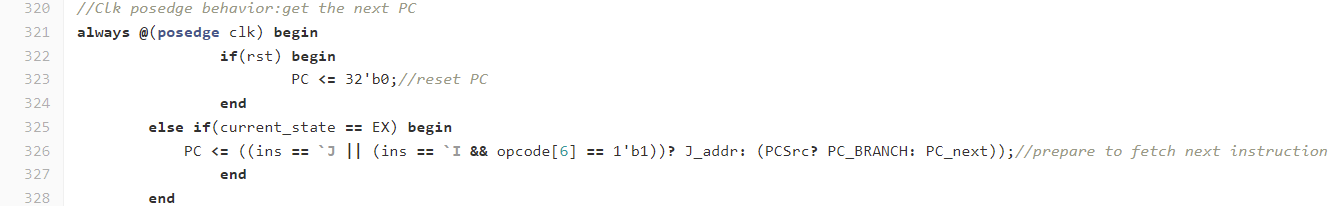


与此共同需要改写的还有ALU的两个端口：aluA和aluB。根据指令类型和各类指令的功能，我们确定究竟是PC\_old还是rdata1加上U/J/I/S\_ex还是rdata2。

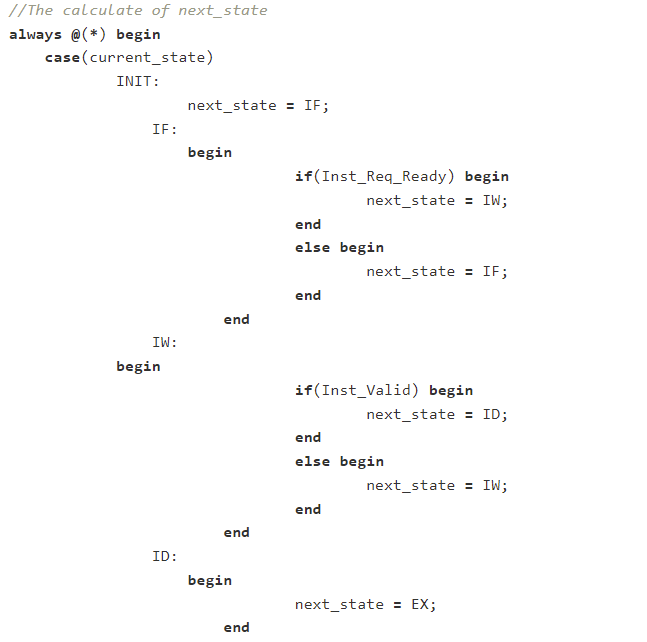


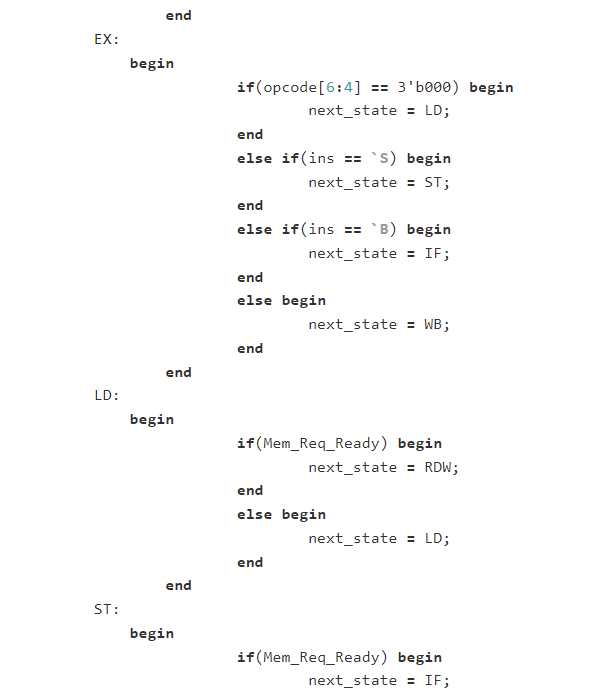
当且仅当目前状态处于WB时拉高RF\_wen。此处无需赘述，主要改变在于RF\_wdata。由于LUI是将U\_ex写入reg，load类指令是将计算出来的load\_res（与mips时计算方法一样）写入reg，JAL和JALR是将旧PC的值+4写入reg，shift类型指令则是将移位器结果写入reg中。除此以外，其余写回寄存器堆的指令均是将ALU\_res写入寄存器。

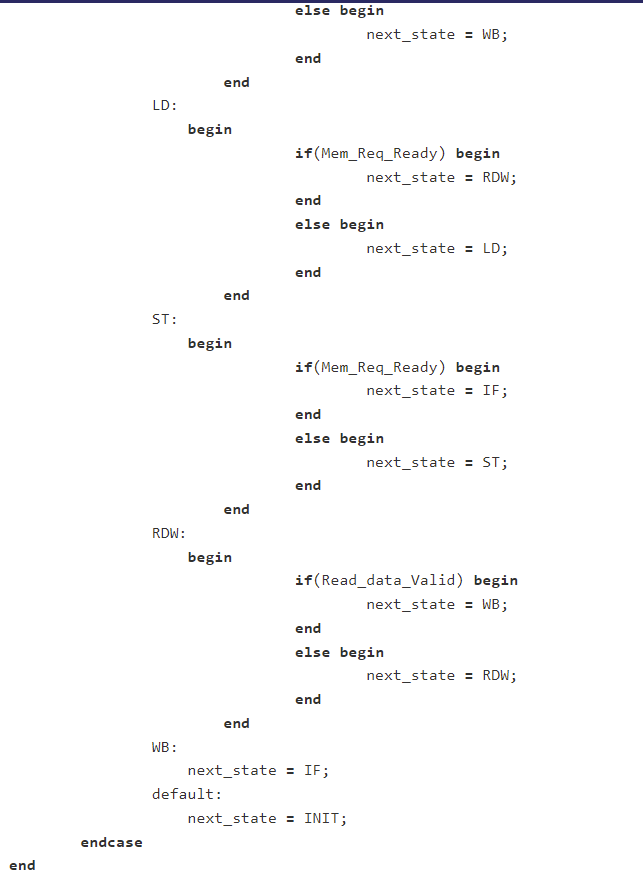
关于PC的跳转，实际上与mips差别不大，在于删去了nop指令的特判:



关于状态机的修改，差别也并不大，注意没有nop指令，按照课件所提及的进行改变即可:







改变主要在于EX状态的条件判断部分，这里的修改是平凡的，不再赘述。



最后我增加了一个内存访问数的性能计数器，用以比较RISC-V和MIPS两大指令集，详见五。

1. **实验过程中遇到的问题、对问题的思考过程及解决方法（比如RTL代码中出现的逻辑bug，逻辑仿真和FPGA调试过程中的难点等）**

遇到了一个问题：因为RF\_wdata判断错误而导致各种信号发生错误。这个bug极其难de，因为寄存器堆是会保存错误的，尽管在写入的时候不会产生错误信号。于是需要从出错点往前寻找很多个时钟周期才能定位到当时的写入错误。

**三、对讲义中思考题（如有）的理解和回答**

无思考题。至于RISC-V和MIPS指令集的区别，详见五。

**四、在课后，你花费了大约\_\_\_6\_\_\_小时完成此次实验。**

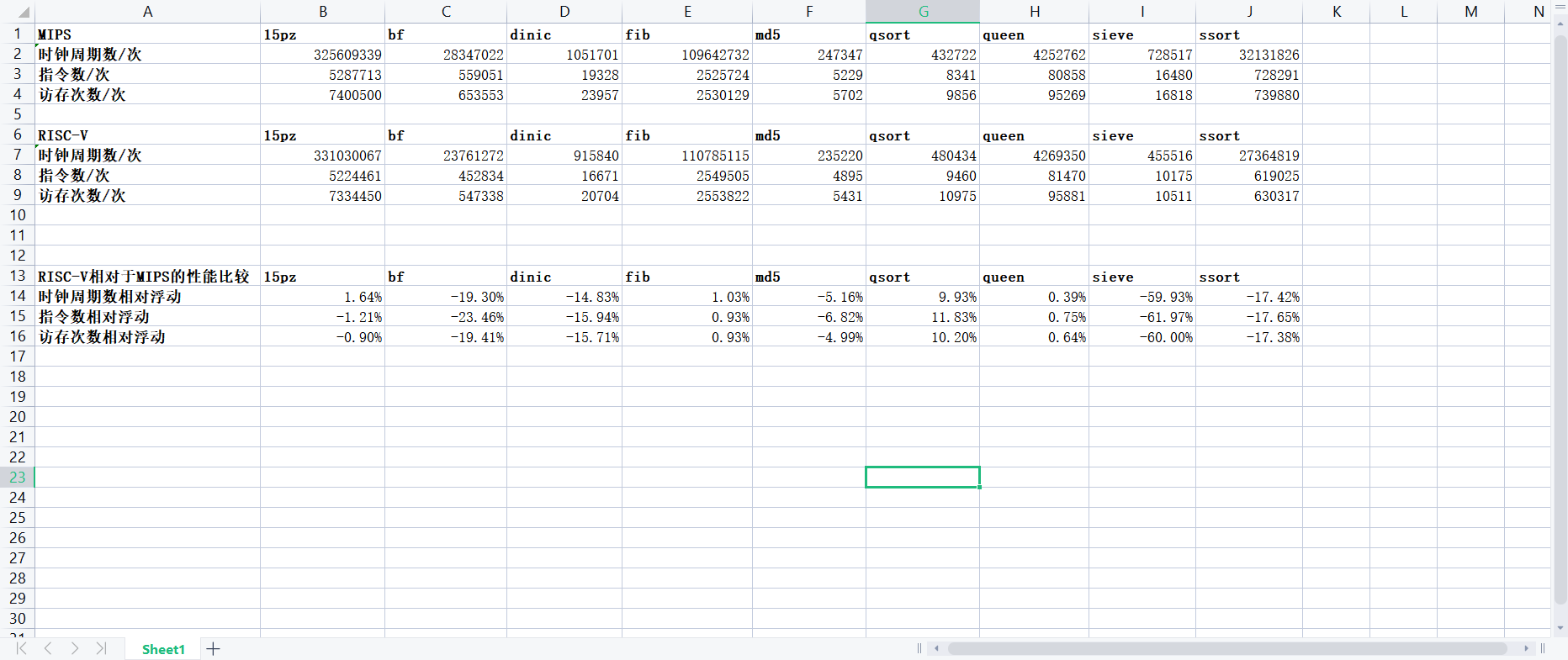
要修改的地方不多，但是不得不说也有一定复杂度。

代码编写和debug等实验主体过程总计耗时**6h**（忽视serve云平台运行时间）。

总共耗时**9h**（算上serve云平台运行时间）。

**五、对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，对实验项目的建议，对提供帮助的同学的感谢，以及其他想与任课老师交流的内容等）**

**对RISC-V和MIPS指令集区别的思考和分析**：



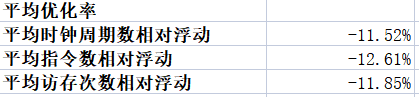
将9个样例的3个性能计数器的数据统计成表，可以得到一些结论。

一般来说，RISC-V指令集下的CPU性能比MIPS强，完成一个操作所需指令较少。

从表格中我们可以看到，完成同一个程序，RISC-V所需指令数基本上都比MIPS要少，并且有几个样例甚至少很多。可见完成一个操作平均所需指令数更少。在sieve中甚至优化了60%左右。为此，我在网上查询相关信息，却只得到了“MIPS指令集垂直性太好导致实现同一操作需要更多指令”的回答。

当然，不乏例外情况：fib、qsort和queen三个样例出现反常。可见RISC-V指令集的性能并非严格优于MIPS；然而，即便是这三个反常样例，所多出的比例也不高。最多的是ssort，也仅仅劣化了17%，相比sieve的60%并不算高。

为了验证我的猜想，我对九个样例的性能表现求了平均，得到平均性能提升率，如下：

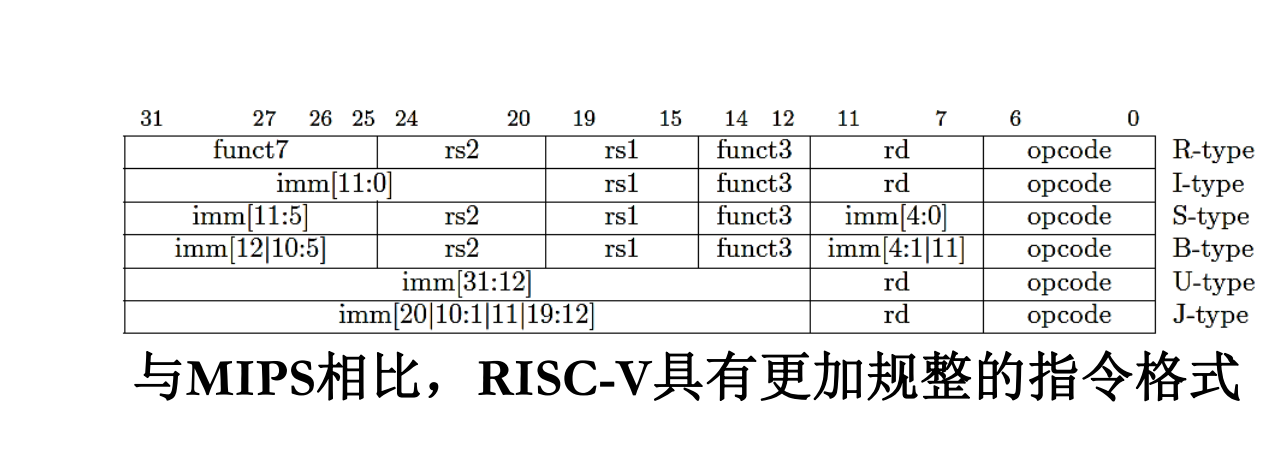


猜想得到验证。进一步地，我们可以看出**RISC-V指令集下的CPU在平均性能（时钟周期数、指令数、访存次数）上比MIPS优化了约12%**。由于实验课样例很有限，该结论并不能保证正确性，但是大致趋势是可以看出来的。

此外，RISC-V有个明显的优点是**指令格式规整，所以译码成本比MIPS要低；另外，基于RISC-V指令集的CPU数据通路也更加简单。**事实上在代码编写过程中我就意识到了这一点。

一个思考是：RISC-V到底有没有“NOP指令”？事实上，RISC-V中也有类似MIPS的NOP指令的指令，只不过本实验并没有留给我们编写。

另一个思考是：为什么RISC-V的指令里立即数会被划分为好几块？



第一个想到的是，在R-type指令中，指令已经被预先确定为了这样的六段，在设计其它指令的时候由于没有func7等因素，为了保持指令规整，必须将立即数拆分开来填入不同的部分。此外，助教学长还提示我：这样拆分会让立即数的各个部分可能取值减少，但我对此存疑。

在上个实验的实验报告中，我忘记了提一个建议。**在此，强烈建议：删除矩阵乘法和水仙花数！删除矩阵乘法和水仙花数！删除矩阵乘法和水仙花数！**这两个样例实在太耗时了，严重影响同学们的云平台测试体验（例如，推到serve上跑，然后睡觉，结果第二天睡醒一看，发现failed）。建议寻找其它的具有相同或者更好的测试能力但是耗时较短的测试样例作为替代。

本次实验帮助我深刻体会到了RISC-V指令集的特点——格式规整、精简、性能优越等，并借助基于RISC-V的CPU的实现，对比MIPS，分析总结了他们之间的异同和优缺点。

**致谢**：感谢孙维铭同学，帮助我de出了**一个极其隐蔽的bug！该bug横跨几千纳秒的波形图！**