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：02 实验名称：简单功能型处理器设计

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。

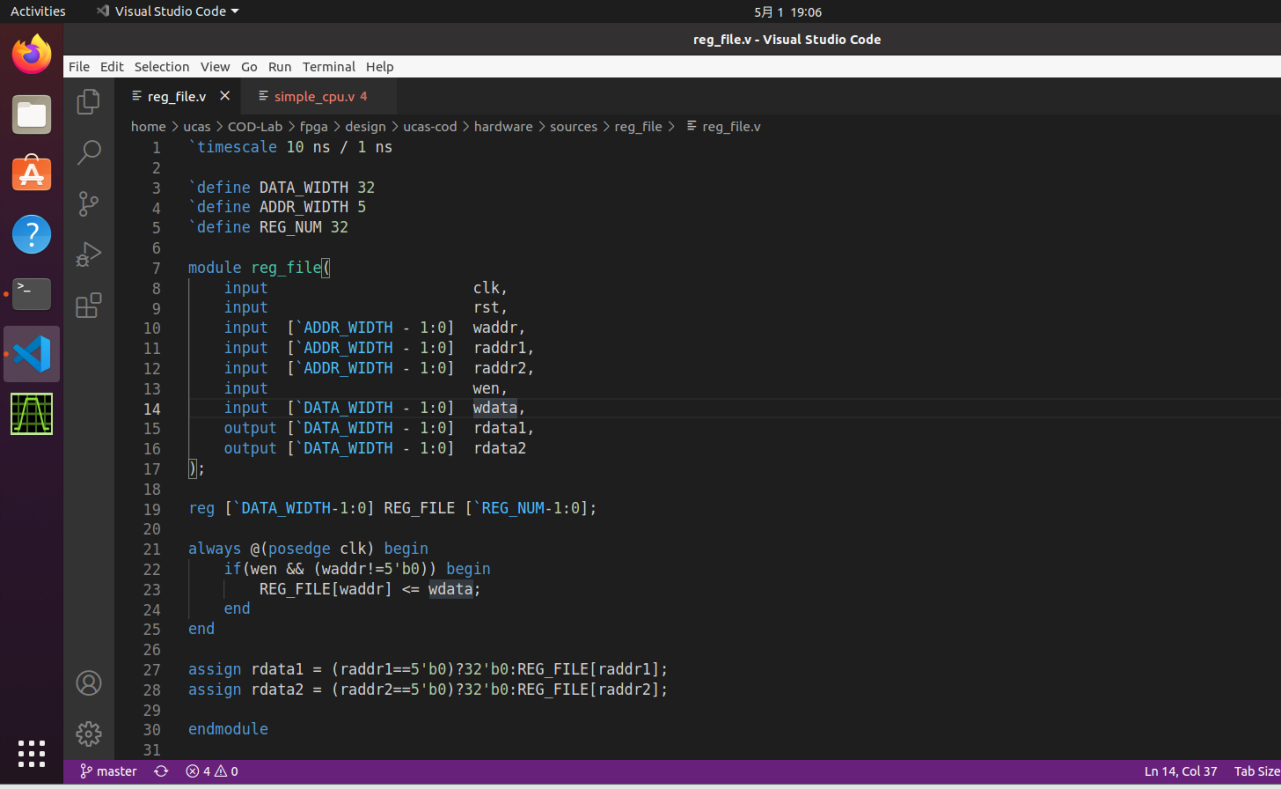
注2：使用git add及git commit命令将实验报告PDF文件添加到本地仓库master分支，并通过git push推送到GitLab远程仓库master分支（具体命令详见实验报告）。

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）**

为完成单周期CPU实验，我首先对reg\_file.v以及alu.v这两份寄存器堆以及运算器代码进行了修改以便复用，然后利用它们和新写的shifter.v移位器构造了单周期处理器的代码。

1. **reg\_file相应代码修改**

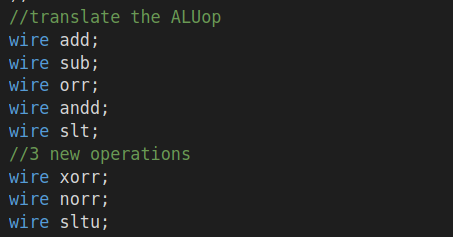


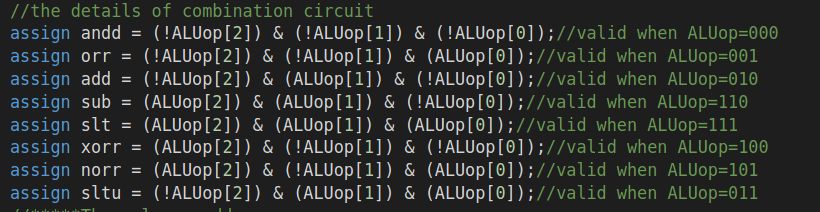
本以为reg\_file.v不需要进行修改，谁知在后来云平台的语法检查中出了错误，发现是reg\_file模块没有加复位信号rst导致的，这是由于在实验一中我的reg\_file.v代码并没有设置rst接口。因此reg\_file的修改仅是在于rst端口信号的加入。

后来发现了，是因为我在调用的时候自作主张地加上了rst信号端口。其实不需要修改寄存器代码。

1. **alu相应代码修改**

对于alu而言需要进行修改的内容是增加三个ALUop信号，也就是按位或非nor、无符号数比较sltu以及按位异或xor.修改后的关键代码段如下：

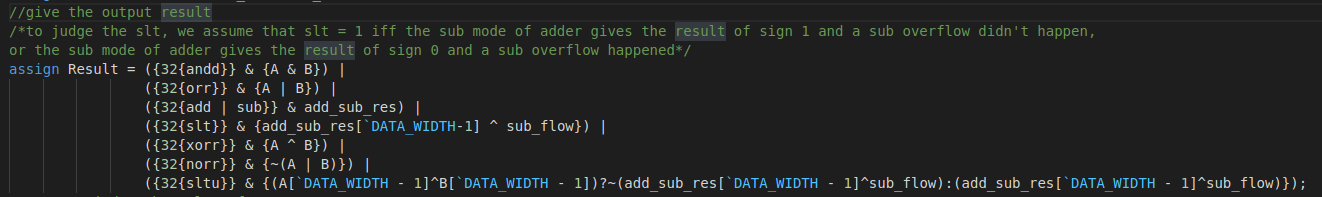




**译码信号**：新增三个译码信号xorr,norr和sltu。如此命名是因为xor以及nor是verilog的保留字。

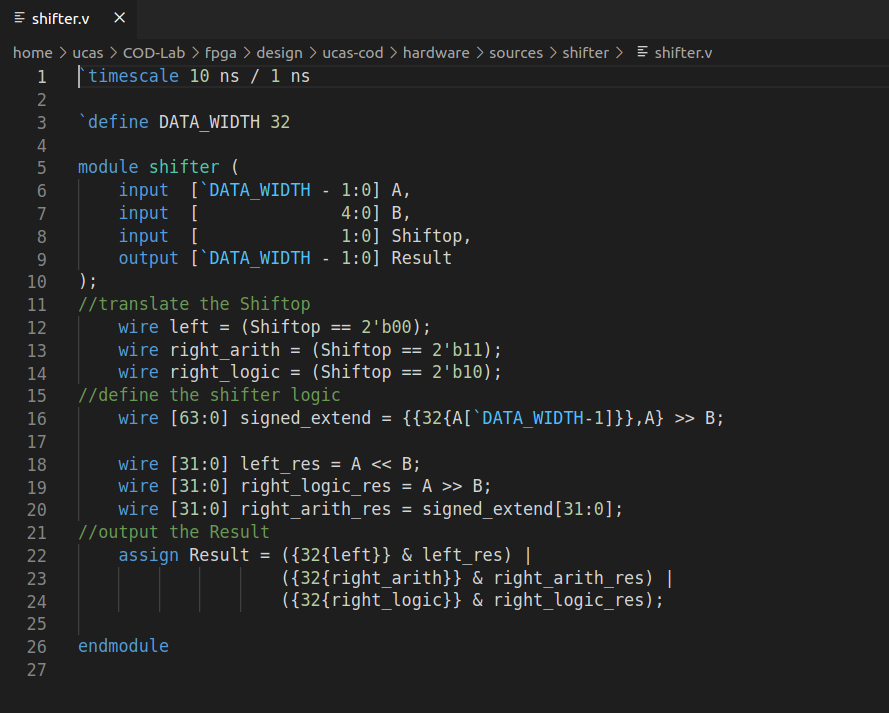
IMG_258

**加法器信号**：将减法的选择信号从单独的ALUop[2]改为ALUop[2] | sltu，以便在sltu的译码信号下加法器执行减法功能。



**Result信号**：将Result这一output信号改为上图所示。增加了xor、nor以及sltu的结果。对于sltu，考虑A与B符号位，如果符号相同，则判断规则同slt，若符号不同，有两种可能：A小B大，结果大且溢出或者小且不溢出；A大B小，结果大且不溢出或者小且溢出。发现这两种情况等价于add\_sub\_res[31]和sub\_flow同号，这是与slt相反的。综上，根据同类合并的原则，我用三目运算符给sltu下的Result进行选择赋值。

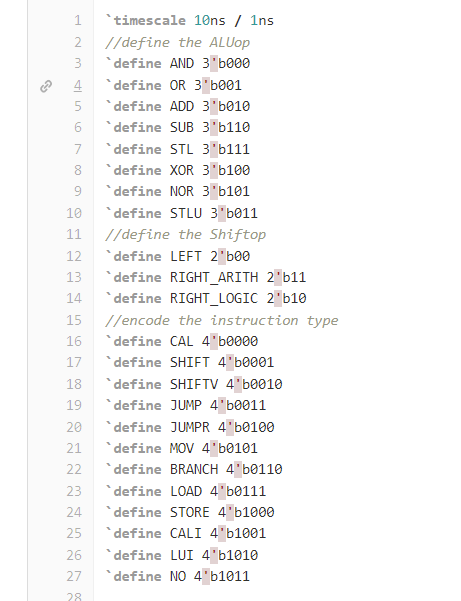
1. **shifter代码实现**



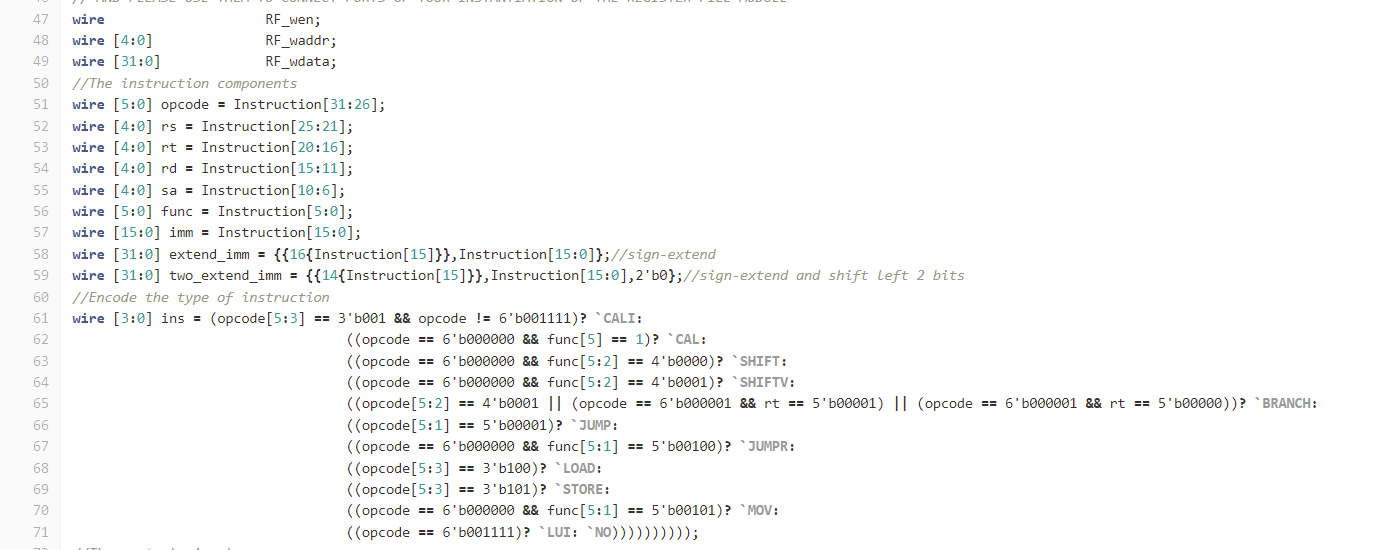
与ALU类似地，我们利用left, right\_arith和right\_logic产生译码信号，然后利用verilog的移位运算符轻而易举地实现了左移和逻辑右移。关键在于算数右移，这里我采取了一点代码技巧：先将被移位数进行符号扩展，再对符号扩展后的操作数进行逻辑右移，取低32位作为算术右移的结果。

Result的确定方法同ALU，是简单的，在此不再赘述。

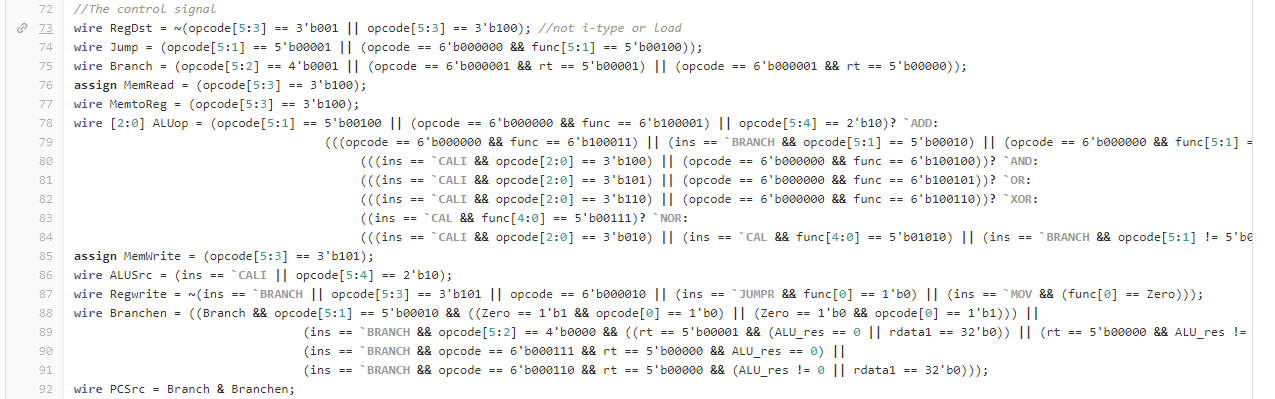
1. **simple\_cpu**

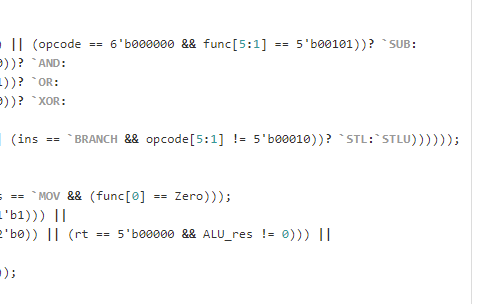


**宏定义**：考虑到需要对ALU与shifter这两个模块进行复用，我们的宏定义中包括了ALUop、shifterop的译码；另一方面，考虑将指令译码，我们又定义出12种指令译码,以此定义出Ins\_Type的宏。

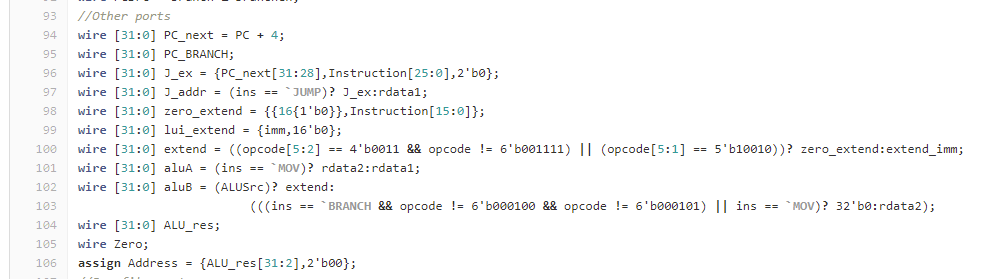


**指令译码**：此处我们将Instruction信号进行译码，分析指令的组成部分，并且将指令分类。对应相同控制信号的同类指令进行合并，根据事先写出的指令译码表进行译码。这部分工作有点困难且复杂，我一度写错了译码表。这里我采用先将指令分类的方法来进行之后的操作，是因为这样可以方便后续control signal和其它端口信号的判断方式统一，便于规范化、统一化操作，也增强了代码的可读性。

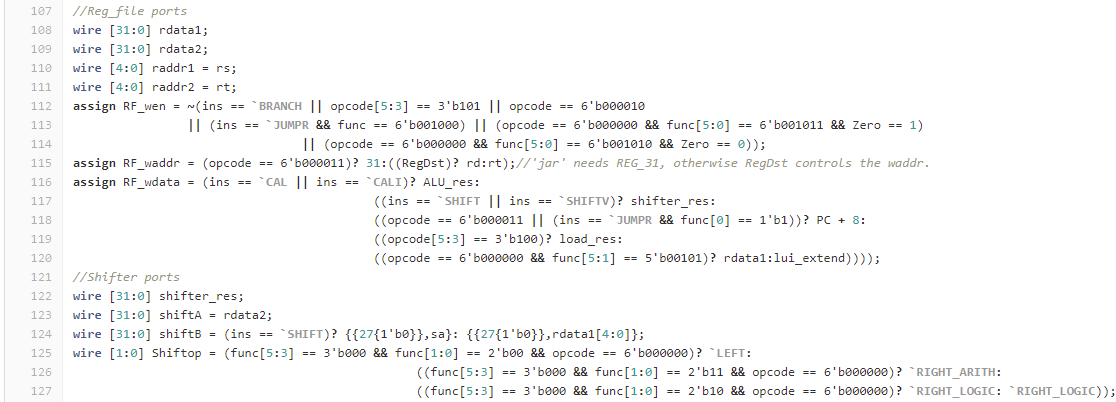




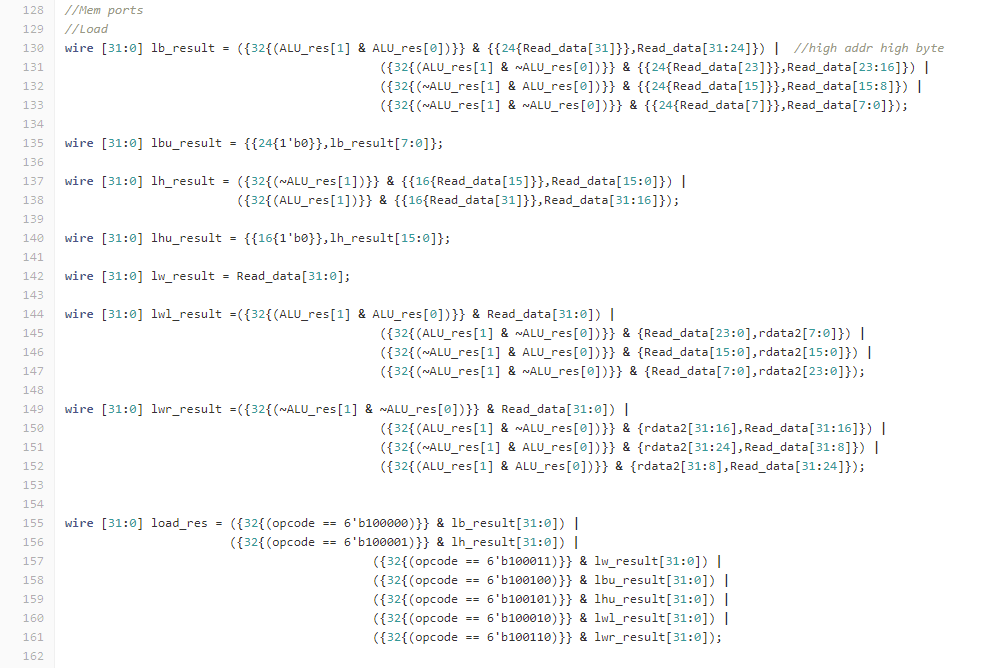
**控制信号：**对于control unit的实现，这里我依照助教学姐的意见进行了一定的优化：对于1bit宽的信号赋值，不再使用三目运算符，而是直接采取表达式赋值，简化了代码。对照每种指令对应的控制信号表和理论课上讲过的数据通路，我对控制信号依次进行了赋值，这里注意并非每种指令对应唯一的控制信号，对于一些特殊地指令仍然要单独讨论，否则必然会出错。个人认为这是单周期处理器最艰难的部分之一，但是却没有太多的思维含量，只需要总结每条指令的控制信号即可，关键在于复杂程度较高，但是细心分类即可。



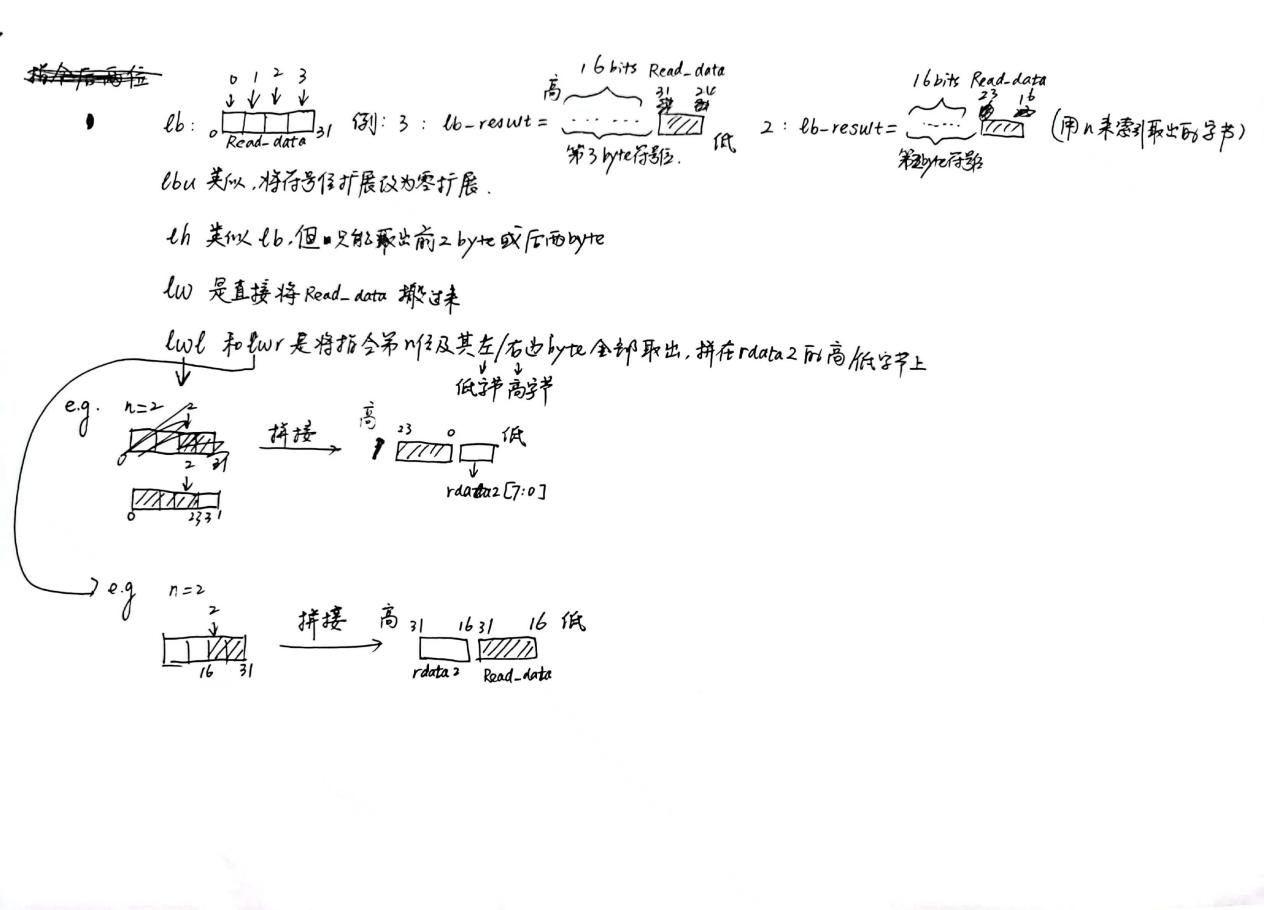
**其它端口信号**：除却控制信号以外，整个数据通路还有其它很多端口需要定义或赋值。对照PPT上的数据通路和组件结构，挨个写出即可，注意同类指令的合并。此时之前的指令分类和宏起到了作用，依照这种清楚、可读性高的写法，在写此部分代码的过程中我的思路非常清晰，一气呵成。

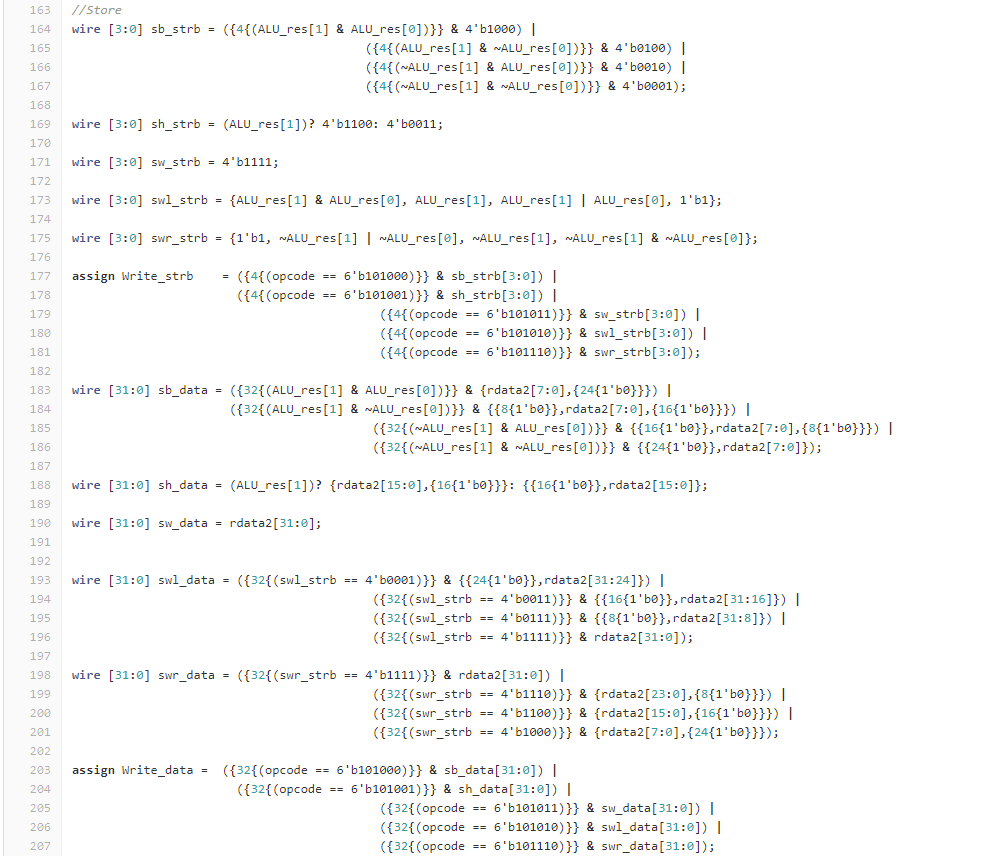


**寄存器堆和移位器端口：**这里大部分端口是显然的（比如raddr1和addr2），但是RF的写使能、写地址和写数据都需要进行一番计算、分类和总结。和前面的译码、分类一样，是一个体力活，需要关照每条指令，耗费时间较长且bug较多，后来根据波形图成功debug，在此不再赘述。

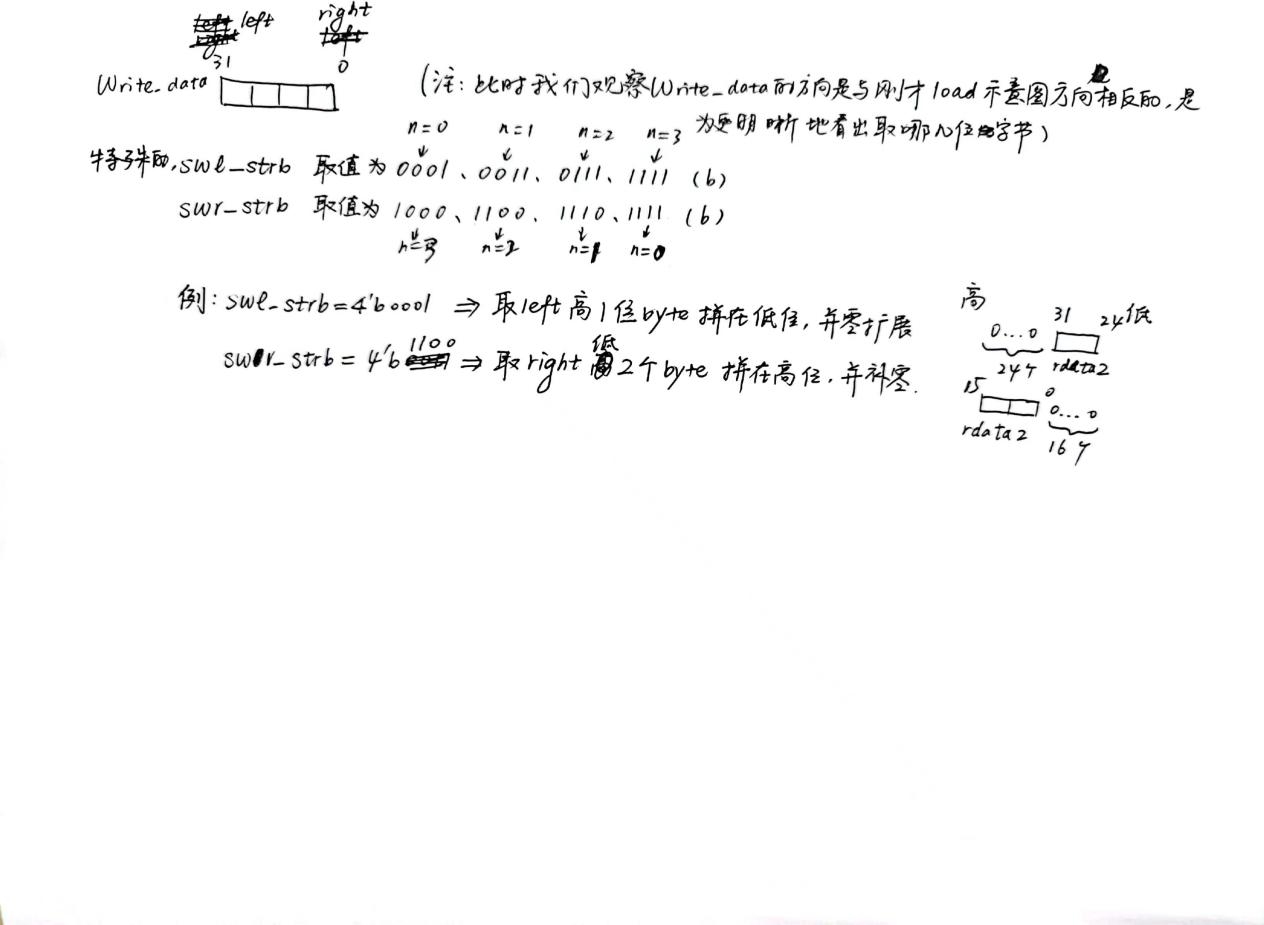


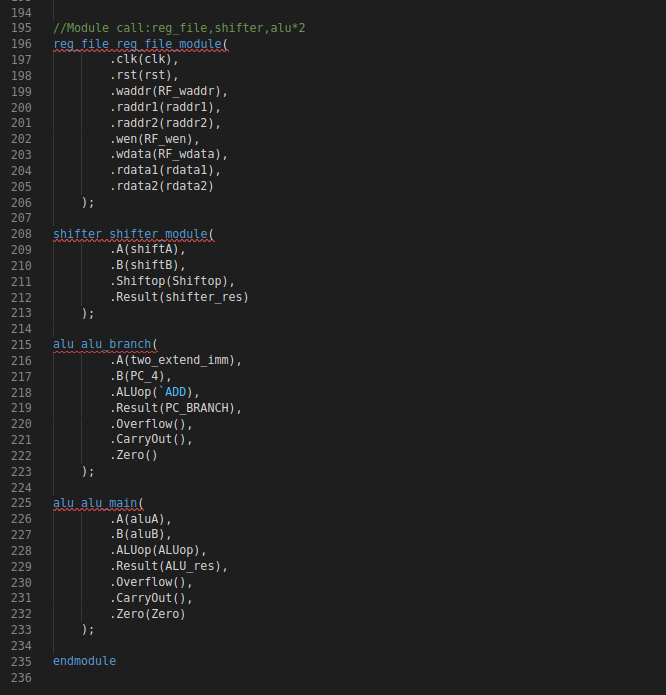
**Load指令：**这里为方便根据ALU运算结果低两位进行赋值，我将它们单独设置在n中存储，后面根据load指令（signed的和unsigned的）来决定是进行符号扩展还是零扩展，并且根据n的两位的情况来确定加载位。其中lwl和lwr指令较为特殊，根据指令手册，画出示意图如下：



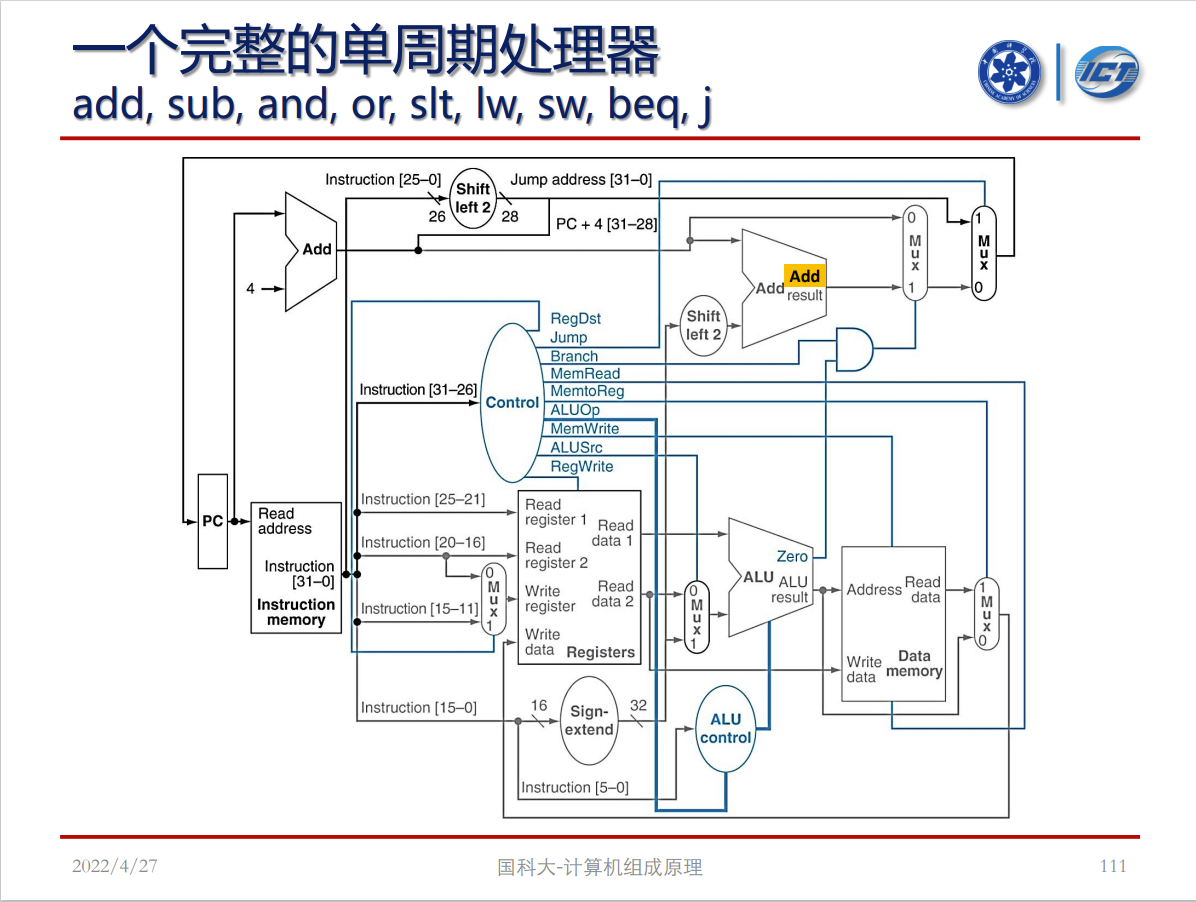


**Store指令：**由于比load指令多出一个步骤，我们需要先确定strb信号的取值，以此控制内存写的数据位，swl和swr同样特殊，根据指令示意图，画出示意图如下：





**模块调用：**调用寄存器堆reg\_file.v和移位器shifter.v，这里注意运算器alu.v需要调用两次，因为PC需要单独使用一个加法器进行跳转地址的计算。



如图所示即我的代码对应的数据通路以及逻辑电路示意图。唯一的不同点在于最上面的shifter，我并没有使用这一个shifter，而是直接拼接，节省了模块调用。

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）

首先，因为是第一次写百行量级的verilog代码，而且距离上次实验已经很长时间，我对verilog语法的掌握变得生疏起来，在最先的多次提交中都因为rtl\_chk语法错误而烦恼，后来经过李衍君同学的点醒，我在本地虚拟机安装的vscode中找到了检查verilog语法的插件iverilog，并实现了就地检查语法，不必多次提交serve等待检查结果。

其次，在行为仿真阶段，一开始我只通过了一个样例，有多达29个样例均未通过，后来通过查看仿真波形来逐步debug。在debug的过程中我采取的方法是先将未通过样例的错误波形信号逐个点开，总结哪几个wire或者reg出了问题，再回头专门检查这几个wire或reg。为此，我从波形图界面点开Instruction以及和这几个信号有关的所有信号（数据通路上的相关信号），根据指令本身（在纸上手动译码出指令内容），分析每个信号在该指令下是否正确。个人认为这种方法便捷高效，能非常精确地定位bug来源。但是在此过程中出了一个很深的bug：我在ALU的修改中错误地认为按位或|运算符比加法+运算符优先级高，导致我的ALU内部信号出错。为此我盯着代码看了两个小时。事实说明，对语法的熟练掌握不仅能提高写代码效率和减少出错，在debug时也不会产生如此难以意识到的错误（因为完全没有意识到这个错误，我甚至一度怀疑是查看波形图的软件本身出了故障）。

最后，也是当时比较困扰我的问题出在bit\_gen的环节，serve网站上的报错显示我有multi-drive问题，但是苦于我不了解这个问题，在网上查找了半天也没有太多收获，一直到凌晨2点才在群里询问助教老师，最终得到了陈欲晓老师的解答，豁然开朗，第二天早晨就修改simple\_cpu.v代码消除了多驱动问题——问题原来出在我在模块内部又以wire的身份对多个output端口进行了赋值，产生了多驱动，事实上直接对其进行assign即可。这个问题的根源在于，我本意是对信号都采用wire定义的同时赋值，这样节省了assign的步骤，大大降低了代码量，但是没有意识到不能对模块output做出类似操作。

与bug们经此一战，我对verilog代码的掌握程度空前提高，并且经过自己的实践、探索和思考，熟悉了一套debug、解决多驱动等问题的方法，并且养成了良好的代码习惯。只能说，计算机真是一门实践出真知的学科！

1. 对讲义中思考题（如有）的理解和回答



**思考：**图中所示ALUop编码的特点在于对于常见和常用的运算（如加减、按位与、按位或），其编码简单且有效位数低。另外，不难察觉到一个事实：类似的运算之间编码几乎都只相差1bit（这令人不禁联想到格雷码，这样的编码有利于合并相似运算操作，化简逻辑表达式，这是在数字电路课程中提及过的思想），综合其它指令的对应op信号判断来看，图中的ALUop编码已经达到了比较优化的情况，可优化空间已经较小。

但是这并非代表不可优化。如果硬要说有什么优化手段，我认为R-type和I-type指令的共同点在于func和opcode低三位一致，因此可以定义一个新的wire类型的选择信号，用opcode[3]来选通该选择信号，这样，利用新信号，结合func或opcode低三位来确定每个指令对应的ALUop编码。这是一种可行的方案，但是空间成本增加了。只能说这种力度较低的优化可以做但没必要。

1. 在课后，你花费了大约\_\_\_18\_\_\_小时完成此次实验。

修改alu.v与reg\_file.v文件花费了0.5h，写出shifter.v文件花费1h,写出simple\_cpu.v文件以及最后的debug和解决问题花费了16h左右（事实上，我有一天坐在座位上从下午15:00一直写到了凌晨2：00而未休息或者吃饭，只为写simple\_cpu.v）

1. 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）
2. 本实验过程中遇到的障碍之一是**实验课课件上的示意图有错误**......实验课课件上所示的数据通路基本不全，而且有时候没有显示关键信号，具有一定迷惑性，刚开始的时候按照实验课课件写，逐渐意识到不对劲，发现错误百出。最后，我根据理论课课件才找到正确且完整的示意图，成功写出正确代码。
3. 课堂验收的时候出了点问题。因为熬了一周的夜，前一天晚上还熬夜复习拓扑，早八期中考试，上了整整一天的课，中午也没睡觉，傍晚实验课验收的时候脑子不太清醒，对助教老师的提问有点懵，脑子没转过来，反应有点慢，而且jr指令指错了信号。在此声明：此单周期CPU代码为本人原创，验收时表现不佳纯属个人身体状态和熬夜原因。而且写完后在验收前没来得及remake一遍当时写的代码和指令表。
4. 致谢：感谢李衍君同学，在语法检查上提醒了我vscode中iverilog插件的存在，让我能够快速进行语法错误修改。