**Lab 6报告**

学号：2020K80099290？？、2020K8009929019、2020K8009929037

姓名：胡康、李子恒、吕星宇

箱子号：05

一、实验任务（10%）

在lab5的基础上，完成以下任务：

1. 添加csr系列指令和CSR系列寄存器，使CPU能够和CSR交互；

2. 增加syscall，ertn指令，实现系统调用处理；

3. 增加多种异常处理的支持。

二、实验设计（40%）

（一）总体设计思路

1. CSR单独设计，照任务书上推荐设置在WB阶段，实际分析感觉在EXE阶段较优；

2. csr指令和现有指令有部分共享通路；

3. 需要增加额外的flush信号来控制整个流水线上的指令流；

4. ？？

（二）重要模块1设计：CSR模块

处理csr系列指令，能够正确写到CSR寄存器中正确的地址，并能够读取。

1. 工作原理

其工作行为实际上和寄存器堆一致，不过由于其内由多种不同功能的寄存器组成，每一位都有特殊的含义，所以需要详细到每一位上写入。

1. 接口定义

表1：csr信号表

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| reset | IN | 1 | 重置信号 |
| clk | IN | 1 | 时钟信号 |
| csr\_num | IN | 14 | csr寄存器传入“序号/地址” |
| csr\_rvalue | OUT | 32 | csr读出值 |
| csr\_we | IN | 1 | csr写使能信号 |
| csr\_wmask | IN | 32 | csr写入值的掩码 |
| csr\_wvalue | IN | 32 | csr写入值 |
| ws\_ex | IN | 1 | ws阶段触发例外 |
| ws\_ecode | IN | 6 | ws阶段传入的异常码 |
| ws\_esubcode | IN | 9 | ws阶段传入的异常码（辅助） |
| ws\_pc | IN | 32 | ws传入的pc（ra等） |
| ertn | IN | 1 | syscall返回信号 |
| ex\_entry | OUT | 32 | 例外地址输出信号（后续会删除） |
| era\_entry | OUT | 32 | 返回地址输出值（后续会并入rvalue） |

1. 功能描述

接受csr系列指令，完成value的读出和写入。模块本身并不需要很多思考，主要是参照手册的标准来完成代码，讲义中也有很多代码可以参考。主要倒是一些控制信号需要思索一番，详见后续错误1。

1. 重要模块2设计：syscall-ertn处理模块（？？可能要和时钟中断一起讲，暂缺）

1、工作原理

？？

2、功能描述

？？

3、注意事项

？？

1. 重要模块3设计：？？

1、工作原理

？？

2、功能描述

？？

3、注意事项

？？

三、实验过程（50%）

（一）实验流水账

2022.10.12 12：30——22：16 学习csr设计，并完成代码部分。（未运行）

2022.10.13 10：00——11：30 debug，仿真通过；

2022.10.14 14：00——20：00 完成exp12部分实验报告的书写；

2022.10.9 20：30——22：00 ？？

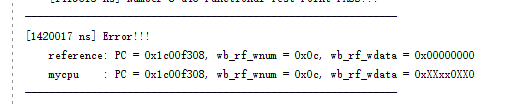
2022.10.11 8：30——9：30 ？？

（二）错误记录

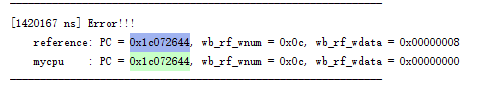
1、错误1：信号逻辑+位宽错误合集

（1）错误现象

wb\_rf\_wdata出错：



图x：错误1中出错截图



图x：错误1中出错截图

（2）分析定位过程

先从pc找到当前指令的行为是什么，然后手动推导其行为，最后自后向前搜查波形问题，找到对与错变化的信号，再从源代码中寻找错误。

（3）错误原因

分析div\_valid的来源，发现是来源于bus\_r（es\_mul\_div\_op），这需要一次clk才能将bus中的值传入bus\_r，这本是正

比如ds\_ready\_go的逻辑错误：



图x：错误1中出错代码截图

应该要对两者取与，而不是或，是两者都不发生时，才是ready\_go，否则就是该阶段被阻塞。

还有一些位宽问题，在car.v文件中，crmd\_rvalue忘记声明，直接使用assign了，结果只有一位。检查之后estat\_rvalue的位宽也没有声明。

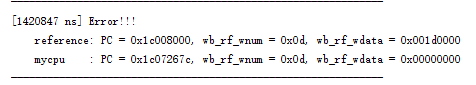
（4）修正效果

位宽修改合适，逻辑重调即可，基本上不需要动脑子的活。

2、错误2：syscall-ertn操作中PC跳转出错

（1）错误现象

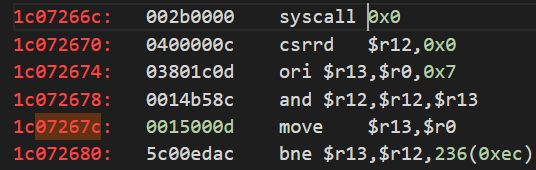
Pc跳转出错：



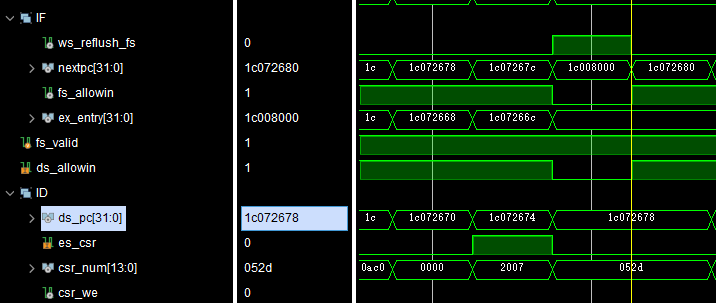
图x：错2中出错截图

（2）分析定位过程

pc出错，只能是syscall了，但是奇怪的是之前有一个syscall却成功了，查看上下代码并结合波形发现其出错原因



图x：错2中出错代码截图



图x：错2中出错波形截图

（3）错误原因

很明显，在黄线之前的两个周期。es\_csr拉高，导致IF阶段的pc没有传递给ID阶段。其原因是，后面的syscall恰好碰见一个csr指令，结果产生了阻塞导致ds\_ready\_go为0，从而使得IF阶段的pc没能正确地进入ID阶段。在wb提交syscall指令后，中断信号丢失，pc组合地恢复到pc+4的情况，这就造成了错误。

（4）修正效果

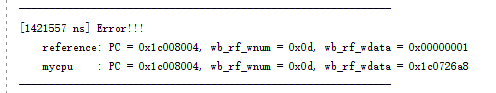
图x：错2中出错代码修改截图

或上flush信号即可。

3、错误3：syscall-ertn操作中syscall与st指令冲突出错

（1）错误现象

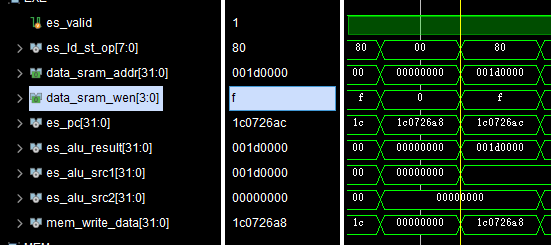
Ld结果出错：



图x：错3中出错截图

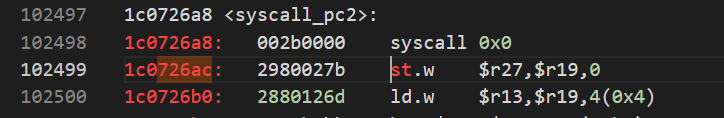
（2）分析定位过程

ld指令出错，目前唯一的可能就是之前的store指令出了问题，找到写入对应值的st.w指令：



图x：错3中出错波形截图

这不是这个阶段会出现的bug，所以只能是和syscall之间的问题。查指令发现：



图x：错3中出错代码截图

（3）错误原因

只增加了访存信号结果计算模块，而没有相应地修改ID阶段的各个控制信号。

Syscall生效时会flush之前阶段的信号，但是st指令会在es阶段完成操作，这就需要在es和ms之间加一条信号流，从而取消这种相关。

（4）修正效果



图x：错3中出错代码截图

这样就可以避免这种情况。

4、错误4：？？

（1）错误现象

？？

（2）分析定位过程

？？

（3）错误原因

？？

（4）修正效果

？？

四、实验总结（可选）

小组合作要搞好分工，另外还要理解好小组使用的代码。

Syscall-ertn系列指令流中，很容易出现由指令顺序造成的bug，属于开始写时只考虑功能，未考虑实际中所有出问题的可能性，这一部分应该由小组成员详细讨论决定。