**Lab 8报告**

学号：2020K8009929034、2020K8009929019、2020K8009929037

姓名：胡康、李子恒、吕星宇

箱子号：05

一、实验任务（10%）

存储管理是现代操作系统的重要功能之一. 本实验通过三阶段来完成一个简单的MMU管理:

1. 完成TLB模块的设计；
2. 将TLB集成到现有CPU中， 实现TLB相关指令和控制寄存器；
3. 添加异常处理和虚实地址转换模块，使CPU能够处理MMU相关异常。

二、实验设计（40%）

（一）总体设计思路

1. TLB的设计本质和寄存器区别不大，可以理解成另类的寄存器堆，不过需要比对index来选出合适的输出。

2. 设计虚实地址转换模块加入到IF和EXE级，在cause\_bus上添加相关异常支持。

（二）重要模块1设计：TLB

1. 工作原理

TLB一共需要五类端口，取指和访存的两类查询端口和用于TLB指令和异常处理的两类读写端口，最后是专门给invtlb指令的端口。

取指和访存端口一致（在下面的接口定义中只介绍s端口，即取指为s0端口，访存为s1端口，两者端口功能上一致），通过输入的虚页号，虚地址的12位和asid信息来查找，并输出found，index等信号。

其工作原理为：通过传入的虚页号和asid对所有项进行查找比对：

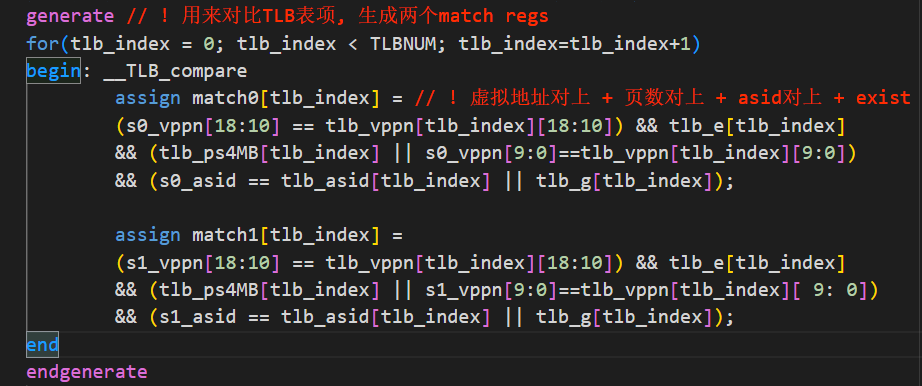


图1：tlb比对代码

并得到对应项的index，否则found=0：

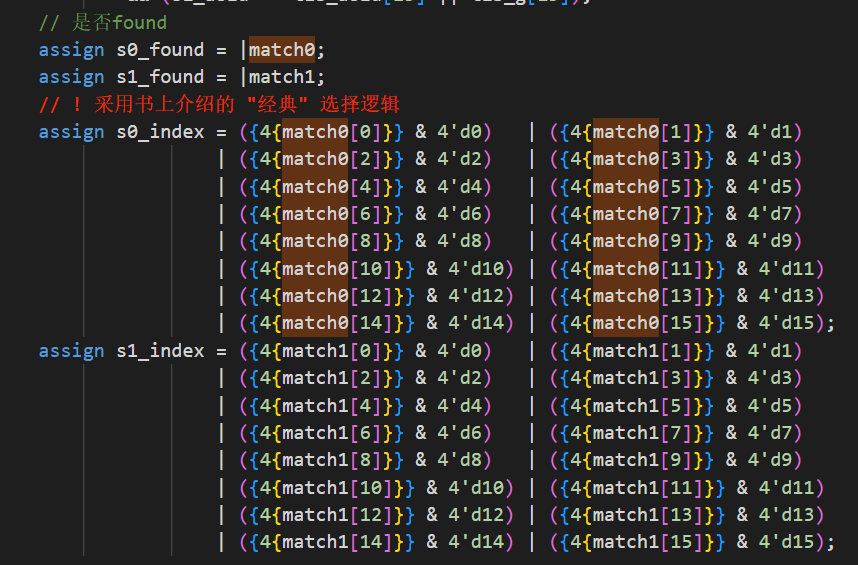


图2：tlb寻找index代码

随后根据对应项的ps位来判定是大页小页和奇偶项，最后输出：

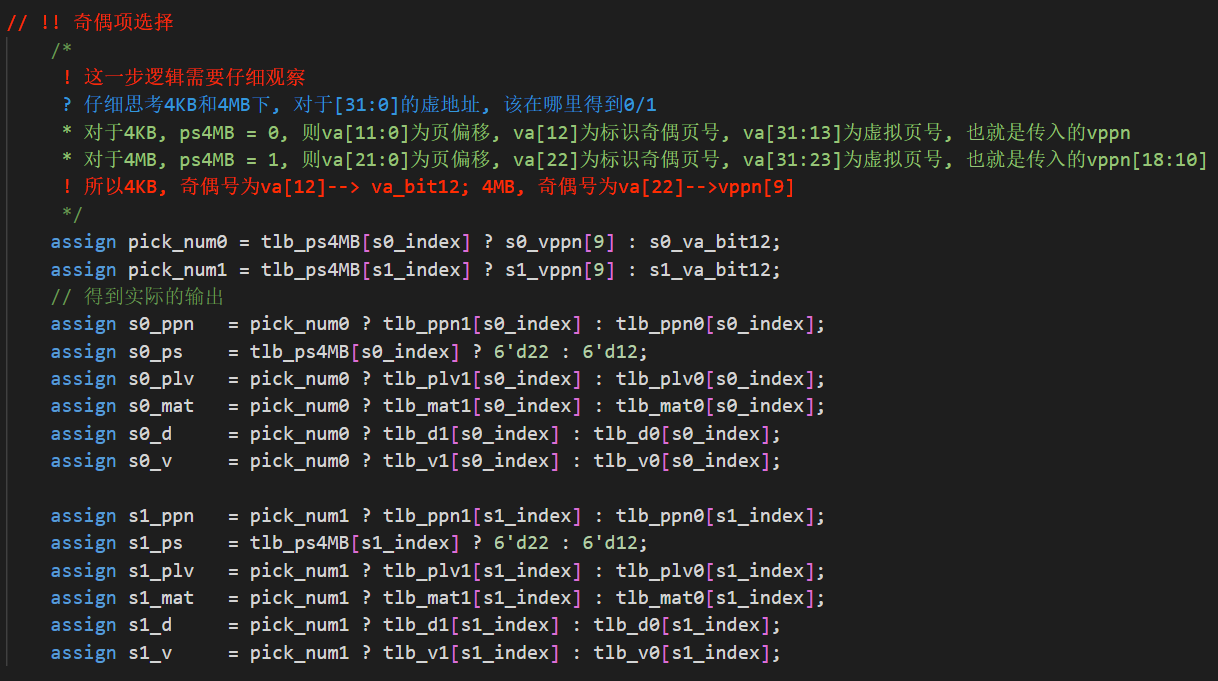


图3：tlb奇偶项选择代码

读写端口和寄存器十分类似，工作原理不再介绍（只给出写端口定义，读只需把we信号删去，除开index输入外的io反转，w前缀改成r即可）。

Invtlb指令则是通过op信号来选出所有需要被删除的项:

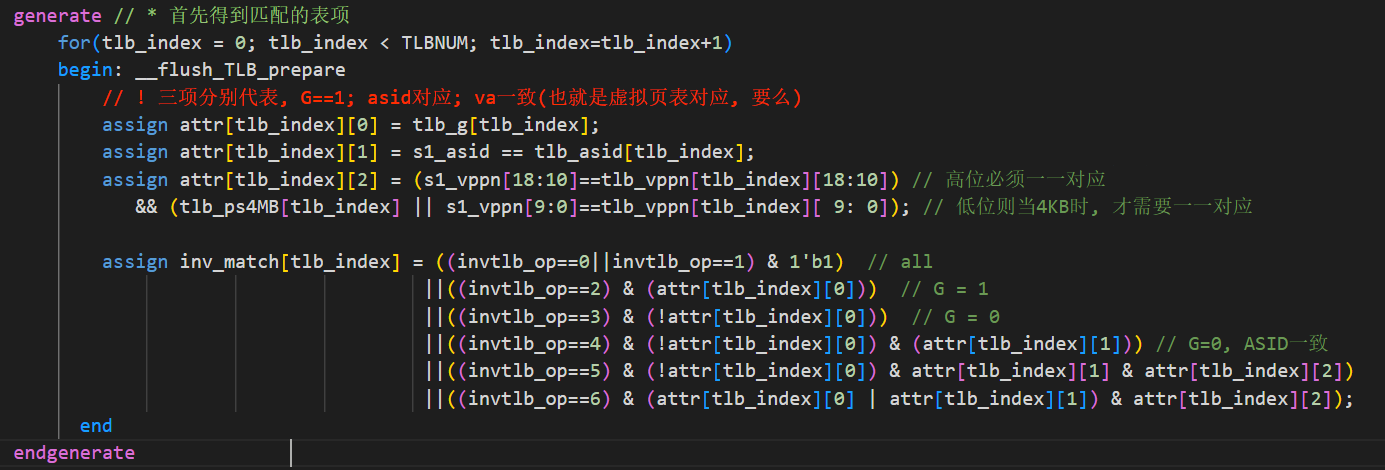


图4：inv指令match代码

然后清零：

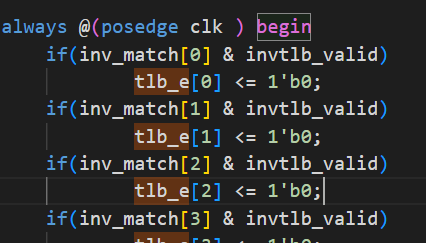


图5：inv指令清除tlb的存在位代码

1. 接口定义

表1：TLB接口信号表

| **名称** | **位宽** | **方向** | **功能描述** |
| --- | --- | --- | --- |
| clk | 1 | input | 时钟信号 |
| s\_vppn | 19 | input | 虚双页号，来自访存虚地址[31:13] |
| s\_va\_bit12 | 1 | input | 来自访存虚地址的第12位，在4KB页下用来选择奇偶页 |
| s\_asid | 10 | output | 标识进程 |
| s\_found | 1 | output | 是否找到对应页 |
| s\_index | log(表项) | output | 对应页的索引 |
| s\_ppn | 20 | output | 对应页的物理页号 |
| s\_ps | 6 | input | 对应页的大小页标志 |
| s\_plv | 2 | input | 对应页的权限 |
| s\_mat | 2 | input | 对应页的访存类型 |
| s\_d | 1 | output | 对应页的脏位 |
| s\_v | 1 | output | 对应页的有效位 |
| … | … | … | 分隔线------------------------------------------------------------- |
| we | 1 | input | 写使能信号 |
| w\_index | log(表项) | input | 写目的tlb项的索引 |
| w\_e | 1 | input | 写入tlb的存在位 |
| w\_ps | 6 | input | 写入tlb的页框大小位 |
| w\_vppn | 19 | input | 写入tlb的虚拟页框号 |
| w\_asid | 10 | input | 写入tlb的进程标识位 |
| w\_g | 1 | input | 写入tlb的全局有效位 |
| w\_ppn0/1 | 20 | input | 写入tlb的物理页号（奇偶） |
| w\_plv0/1 | 2 | input | 写入tlb的权限（奇偶） |
| w\_mat0/1 | 2 | input | 写入tlb的访存类型（奇偶） |
| w\_d0/1 | 1 | input | 写入tlb的脏位（奇偶） |
| w\_v0/1 | 1 | input | 写入tlb的有效位（奇偶） |
| … | … | … | … |
| invtlb\_valid | 1 | input | invtlb指令有效 |
| invtlb\_op | 5 | input | invtlb指令类型 |

1. 功能描述

支持IF和EXE阶段的访存需求，根据crmd相应位提供虚实地址转换功能。

1. 重要模块2设计：vaddr\_transfer虚实地址转换
2. 工作原理

通过传入的crmd确定翻译模式，通过传入的虚拟地址，操作类型，还有asid分别进行三类地址查找：

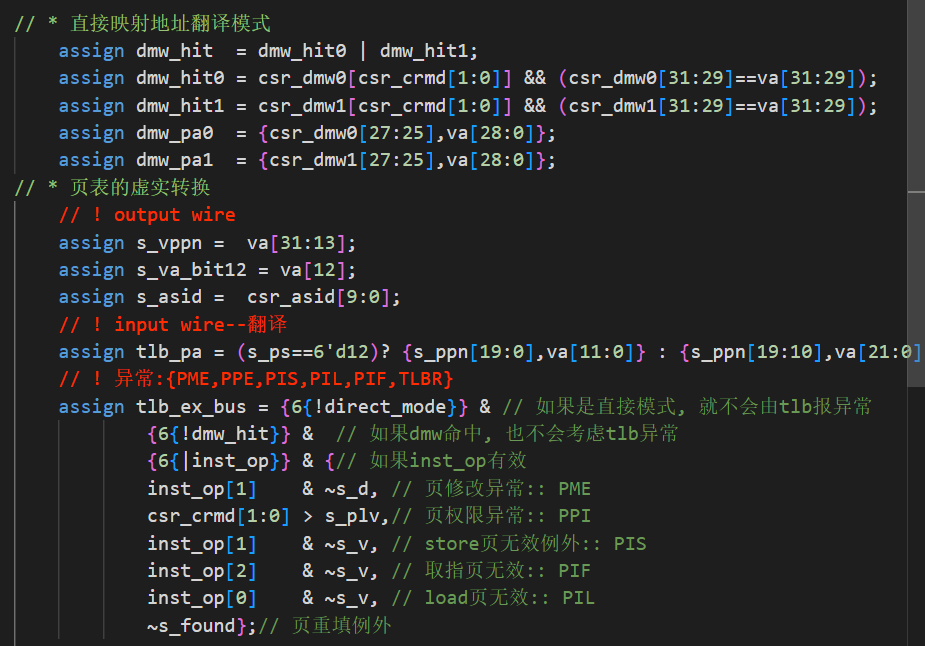


图6：vaddr\_transfer的三类地址查找

最后根据模式和命中来输出实地址，并输出tlb异常信号线。

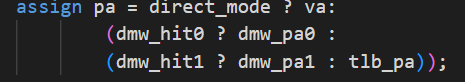


图7：选择输出物理地址

2、各通道接口信号定义

表2：Vaddr\_transfer接口信号表

| **名称** | **位宽** | **方向** | **功能描述** |
| --- | --- | --- | --- |
| va | 32 | input | 传入的虚拟地址 |
| inst\_op | 3 | input | 传入的操作类型{load, store, inst} |
| pa | 32 | output | 输出的物理地址 |
| tlb\_ex\_bus | 6 | output | 期间发生的异常{PME,PPE,PIS,PIL,PIF,TLBR} |
| s\_vppn | 19 | output | 和TLB相连（下列s\_均是），输出虚拟页号 |
| s\_va\_bit12 | 1 | output | 输出虚拟地址的第12位 |
| s\_asid | 10 | output | 输出当前的asid号 |
| s\_found | 1 | input | 从TLB输入，是否查找到 |
| s\_index | 4 | input | 查找到的index |
| s\_ppn | 20 | input | 查找到的页表项物理页号 |
| s\_ps | 6 | input | 查找到的页表项页框大小 |
| s\_plv | 2 | input | 查找到的页表项权限 |
| s\_mat | 2 | input | 查找到的页表项访存类型 |
| s\_d | 1 | input | 查找到的页表项的脏位 |
| s\_v | 1 | input | 查找到的页表项的有效位 |
| csr\_asid | 32 | input | 从csr传入的asid |
| csr\_crmd | 32 | input | 从csr传入的crmd |
| dmw\_hit | 1 | output | 是否有dmw命中，判断ade错 |
| csr\_dmw0 | 32 | input | dmw0信息 |
| csr\_dmw1 | 32 | input | dmw1信息 |

3、功能描述

与IF和EXE的访存地址交互, 生成物理地址。

三、实验过程（50%）

（一）实验流水账

2022.11.18 19：00——22：00 exp17写完

2022.11.21 20：00——21：00 exp17上板，debug

2022.11.23 15：30——19：30 完成exp18代码

2022.11.24 8：00——15：30 调试exp18代码，上板

2022.11.29 9：30——20：00 完成exp19代码

2022.11.30 11：00——20：58 exp19的debug

2022.12.01 13：00——20：20 完成三个实验报告

（二）错误记录

1、错误1：exp17上板错

（1）错误现象

数码管最后停在右侧的数码管从 0x00 累加到 0x0f的地方。

（2）分析定位过程

查询piazza得知，检查是否在不同的always块里对同一个寄存器赋值

（3）错误原因

在多个always块里对同一个寄存器赋值了:

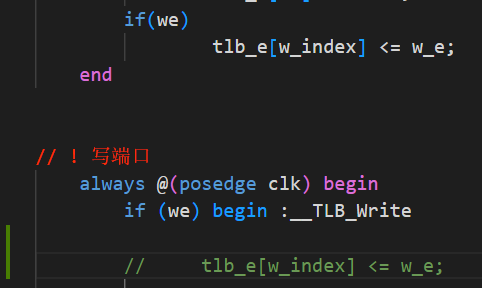


图8：错误1代码截图

（4）修正效果

将下面注释，并且移动到上方invtlb逻辑即可。

（5）归纳总结

这次出错是因为两者功能恰好分开，结果就变成按功能分成两个always块了，以后得设计清晰。

2、错误2：出现了始料未及的tlb异常

（1）错误现象

Pc出错：

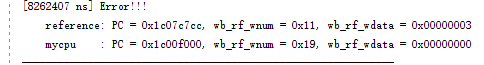


图9：错误2出错情况截图

（2）分析定位过程

查看波形发现是pc迟迟没有拉高we信号提交：

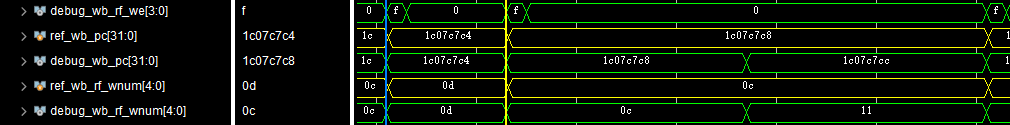


图10：错误2出错波形截图-1

查看指令为：

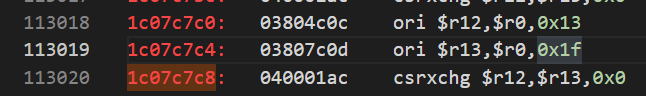


图11：错误2出错指令截图

理应产生写寄存器信号，但是没有，于是查找es级发现了问题：

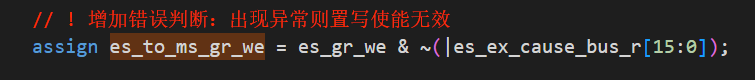


图12：错误2出错代码截图-1

那么是什么异常呢？答案是tlb异常：

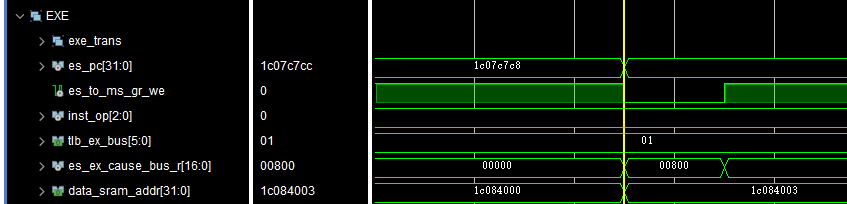


图13：错误2出错波形截图-2

在inst\_op为0的情况下，仍然将异常有效，所以会出bug。

（3）错误原因

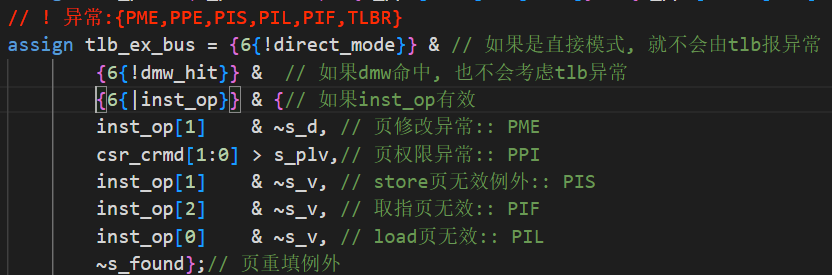


图14：错误2出错代码截图-2

图中本来是没有对{6{|inst\_op}}后来发现，哪怕是没有对总线访问，某些地址照样会产生异常，影响流水级。

（4）修正效果

需要inst\_op有效后，tlb异常线才能有效，这样就规避了非法异常的影响。

（5）归纳总结

需要考虑什么时候会发生异常，什么时候不可能发生某类异常，这样才能让处理器尽可能少的出现非法情况。

3、错误3：没有出对应的异常

（1）错误现象

如图，未发生异常跳转：

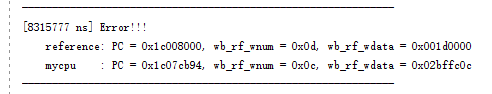


图15：错误3出错现象截图

（2）分析定位过程

首先查看汇编指令序列，如图9所示，虽然是b90地址出问题，但为b94指令出问题，因为我们期待这条ld指令触发异常，但是我在纸上跑了一遍寄存器，发现不应该出现异常。

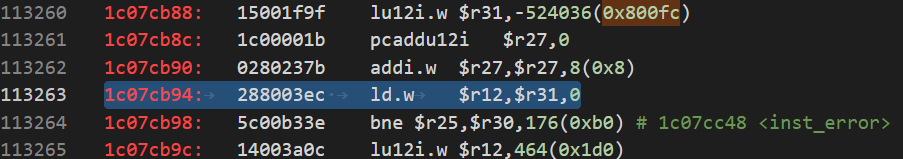


图15：错误3汇编指令序列

百思不得其解下，我查询了n71\_tlb\_ex.S文件，直接锁定了测试用例：

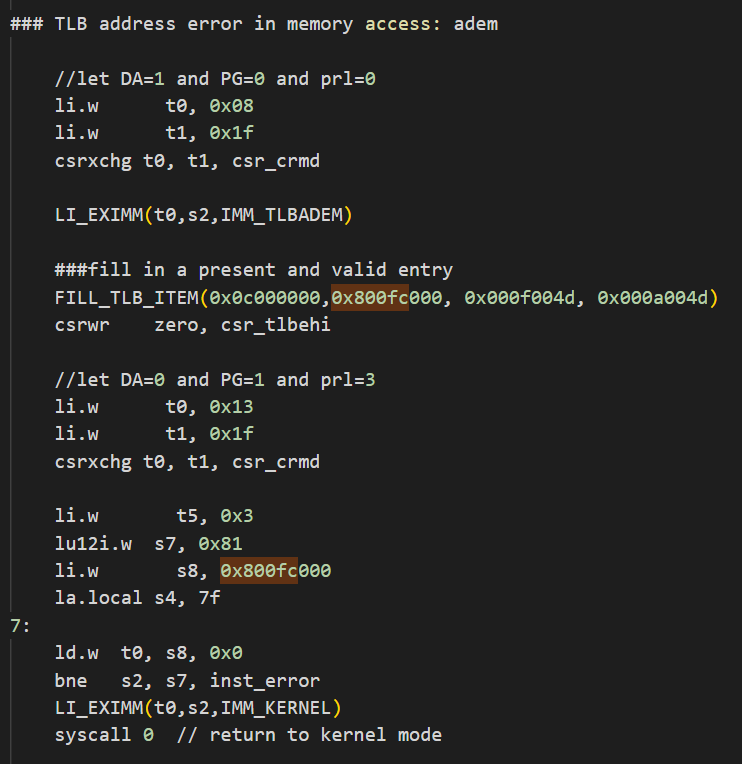


图16：错误3测试指令序列

原来是ADEM异常，可惜，我搜索了手册，没有找到ADEM异常说明。

（3）错误原因

上网搜索后发现，如果不是最高特权等级，访问0x80000000的地址是不被允许的。(地址空间限制)

（4）修正效果

加入相关异常的数据通路和判断（在IF阶段也是同理），直接pass！！



图17：错误3错误代码修正

4、错误4：地址错异常中BADV寄存器出错

（1）错误现象

在异常处理指令中BADV出错:

出错PC: 0x1c0084f8, ref:0x401fe000, mycpu: 0xffffffff

（2）分析定位过程

查询指令得知为csrrd $r12, 0x7

这是一个读BADV寄存器的指令, 那么第一个反映就是vaddr不对. 于是开始查询相关逻辑线路.

（3）错误原因

在原来的逻辑中, ms\_alu\_result是直接连到ms\_vaddr上了, 这也是之前没有出现过指令错的情况造成的后果:

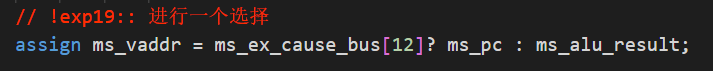


图18：错误4代码截图

（4）修正效果

在ex\_cause\_bus中新加入一位来表征是取指错还是访存错, 这样就完成了对错误地址的选择.

（5）归纳总结

这次出错是因为在之前的设计中, 设计人员没有考虑到后面的实现和实验完整性, 只是想通过测试造成的遗漏代码. 另一方面, 也是后续设计人员没有充分考虑前面设计而直接动手, 没有更多地和之前模块的设计人员沟通, 造成的结果. 所以在多人作业中, 合作和沟通是重中之重.

5、错误5：wb阶段处理异常有优先级安排

（1）错误现象

当异常传递总线上出现多个异常时, CSR.STAT寄存器内容出错.

（2）分析定位过程

STAT寄存器中存储的是ecode和subcode, 很直接地就可以定位到wb阶段的ecode和subecode逻辑.

（3）错误原因

修改异常返回逻辑的优先级即可.

（4）修正效果

将第十二位的IF/EXE地址错判断位放到最后即可, 因为这是一个判断位, 而不是决定发生地址异常的异常位, 所以放到相应的异常后面即可.

（5）归纳总结

由于异常的设计有一些问题, 导致总线部分也没有对异常有很好的适应, 最后需要判断异常发生的优先级, 整体看下来, 实现并不是很漂亮, 也许后续有时间可以修改得更好.

四、实验总结

这次任务最难的是而exp19，它是对整个总线和异常处理系统的挑战。由于初始异常的设计有些问题，在这个实验上花费的时间就多得超乎预期。此外，还有很多没预料的bug出现。此外，还需要看手册和PPT，这样才能迅速解决问题。