**实验4-2报告**

学号：2016K8009929029

姓名：张丽玮

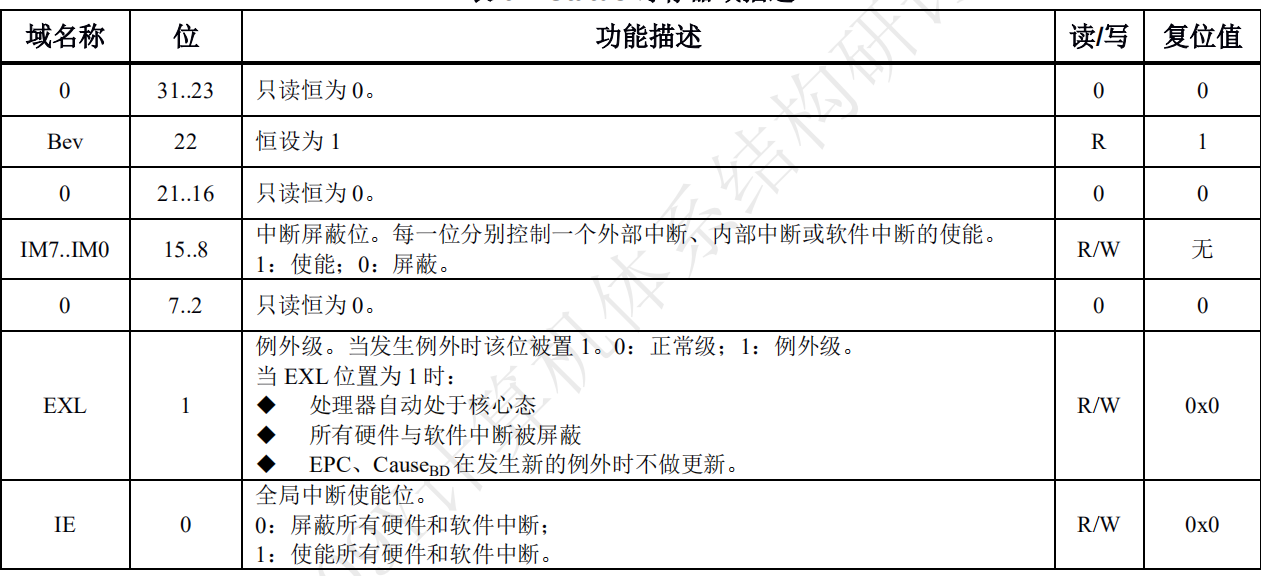
一、实验任务（10%）

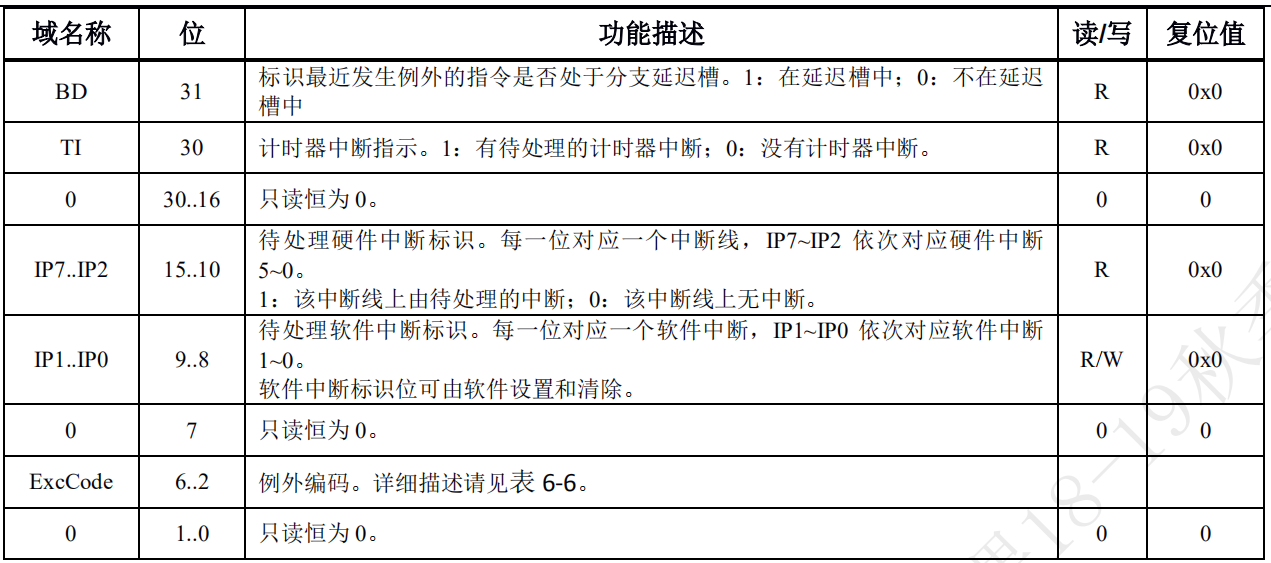
这个实验主要目的在于使得CPU支持例外中断。第一部分主要实现三个指令：MTC0、MFC0、ERET 。同时要对cp0寄存器的作用有所了解，并且实现CP0 寄存器 STATUS、CAUSE、EPC。除此之外还有SYSCALL指令，也就是要完成对系统调用的支持。

增加break指令；增加地址错、整数溢出、保留指令例外支持；增加 CP0 寄存器 COUNT、COMPARE、BADVADDR；增加时钟中断支持，时钟中断要求固定绑定在硬件中断 5 号上；完成 lab4-2 功能测试。

1. 实验设计（30%）

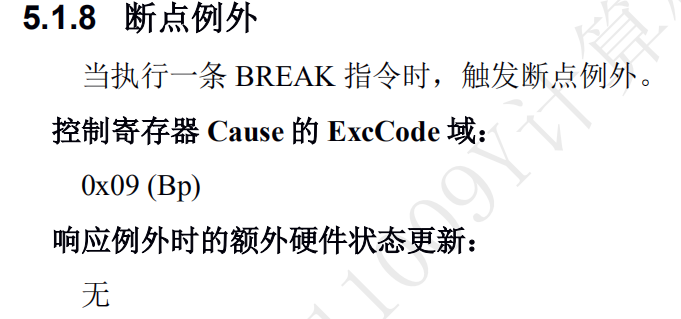
总的来说，是添加mtc0、mfc0、eret和syscall这几个指令，为了实现这些指令，需要增加cp0寄存器。这里epc用于返回系统调用发生的地址，从而在eret的时候回到epc寄存器的地址；cause是用于判断是哪一种例外（通过exccode域），而status寄存器的IM7-IM0位可以控制中断的屏蔽打开。





BadVAddr 寄存器是一个只读寄存器，用于记录最近一次导致发生地址错例外的虚地址，实际上是用来记录下地址错例外的虚地址；当一条 ADD、ADDI 或 SUB 指令执行结果溢出时，触发整型溢出例外；保留指令例外是触发未实现的指令时的例外。

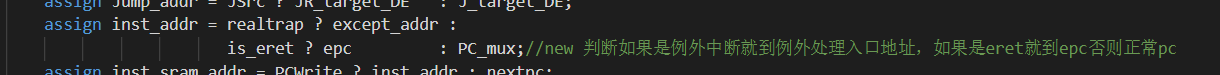
而break指令属于断点例外。



1. fetch模块/next\_pc

fetch模块并没有增添信号，因为目前的例外并不会在取指阶段发生，也并不会受到影响。

但是计算nextpc的逻辑需要改变，如下：



如果是trap例外发生，那么就跳转到例外发生地址

except\_addr = 32'hbfc00380

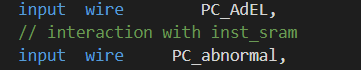
如果是eret指令（这里is\_eret是一个信号），那么需要回到epc地址。

这样通过nextpc基本完成了一个例外中断的地址跳转。

---

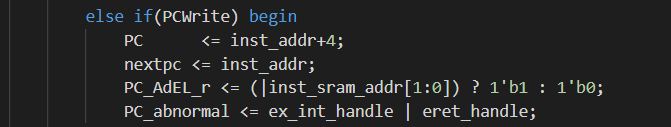
fetch增加了这三个个信号，相应的逻辑如下：

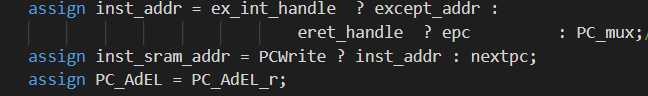






实际上是用来判断取指是否延迟。而这两个信号在nextpc的逻辑中也很重要：

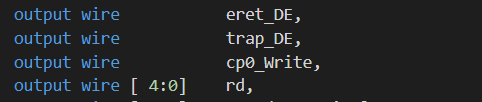




eret表示eret指令，而ex\_int\_handle表示其他例外中断指令。这实际上就是一个是否要跳转380例外地址的判定。与之前不同的是，需要将信号传入五级流水。

1. decode模块

decode模块是一个译码操作模块，增加的接口较多：



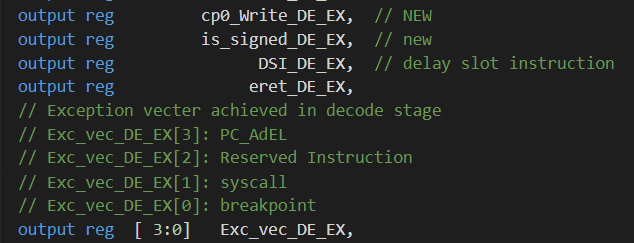


这里实际上进行了一个cp0寄存器内容的输入（cp0寄存器内容在cp0reg模块有处理），然后通过control处理输出一些例外中断信号（判断是否是eret,是否是trap）

因此这一部分主要在control模块里详细叙述。

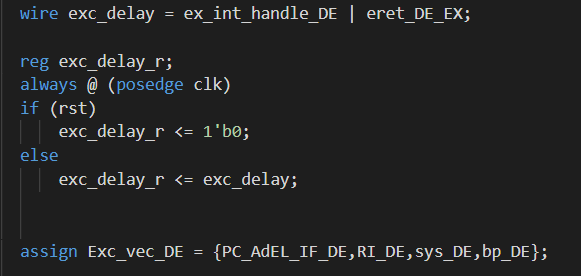
----

同样增加了很多接口



还有fetch模块传出的adel、DSI、ex\_int\_handle信号。

主要增加的逻辑：一是判断延迟，二是判断地址是否错误。

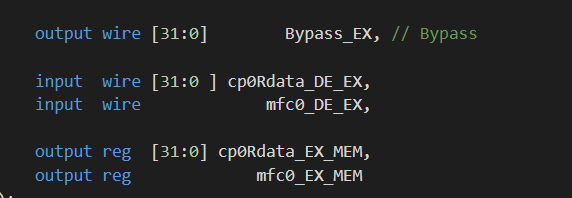




control模块同样有变动，将在下面详述。

1. execute模块

这是一个执行模块，增加了如下几个接口：





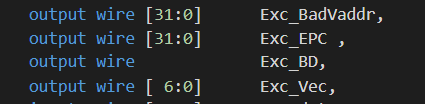
增添的逻辑语句仅仅是这一句：



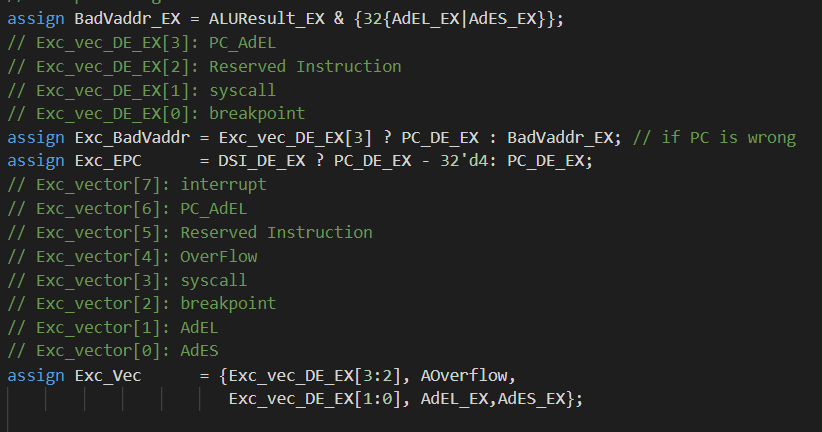
判断是否是mfc0指令，如果是，就取出cp0rdata的值，否则正常输出ALU的结果。

-----

除却从DE模块传入的信号和相应的传出信号，增加的信号如下：



很明显可以看出作用，主要是用来判断是否是地址错或者整型溢出。

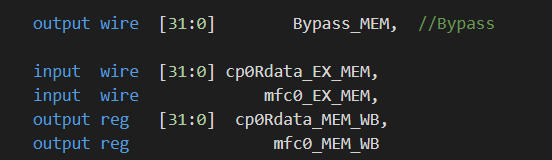




1. memory模块

简单明了的存指操作，接口命名可以看出数据来源和去向。主体部分就是一个时钟节拍控制的赋值操作。

主要增加的接口和EX模块类似：



而实际山增添的操作也就是对于是否是mfc0指令的判断。



---

并无太多变动，只是传输信号。

1. writeback模块

一个写回模块。不需要时钟信号控制，直接赋值即可。

写回模块类似上面的操作，这里主要影响了写回的regwdata寄存器，同样是判断是否是mfc0寄存器，是就取cp0rdata的值。

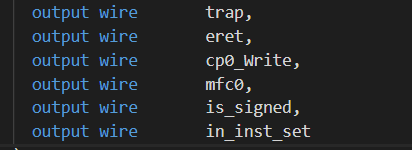


---

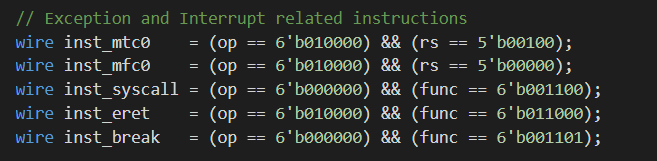
并无太多变动，只是传输信号。

1. control模块

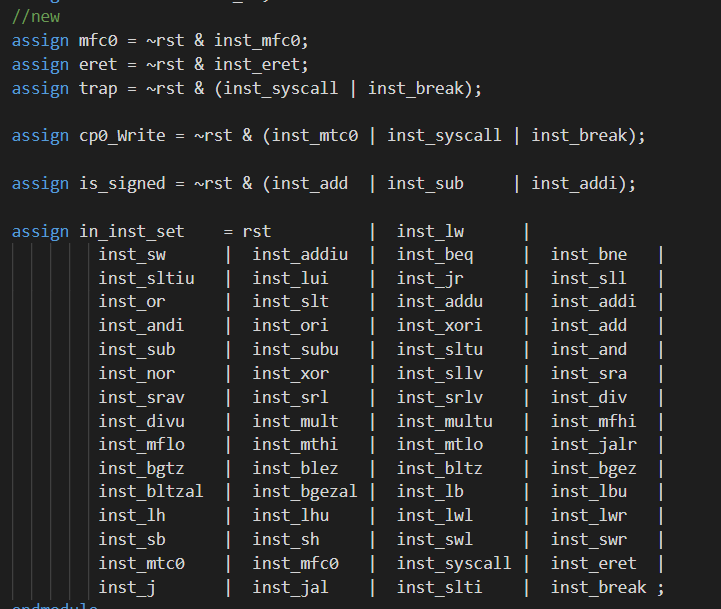
decode的主要功能模块。这次主要增加了这几个接口输出：



同样也增加了这几条指令的判断信号：



除了mfc0指令在regwrite信号里增添，其他都在下方额外增加语句判断信号。



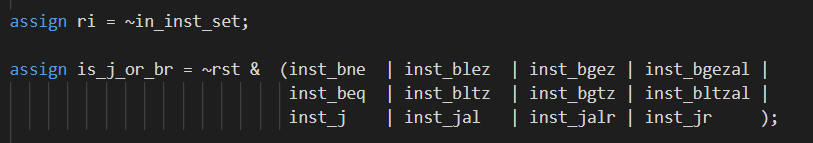
trap用来判断是否是syscall或者break指令，这两个都表示要返回epc地址。Is\_signed和in\_inst\_set在4-1目前没有用到，适用于判断4-2的ALU运算的整形溢出例外。

--

由于增加了指令，control模块也要有相应改动：



这是一个syscall和break指令的判断和信号输出。

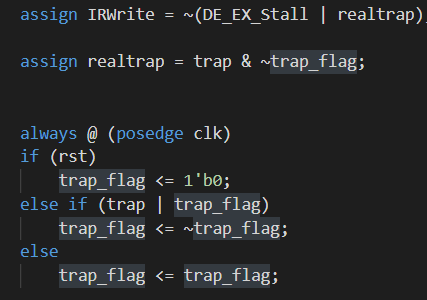


判断是否是j或者br指令输出。ri表示已经实现的指令以外的指令，用以处理保留指令例外。

1. Bypass模块

这是一个为了实现指令相关时延迟取指的旁路模块，

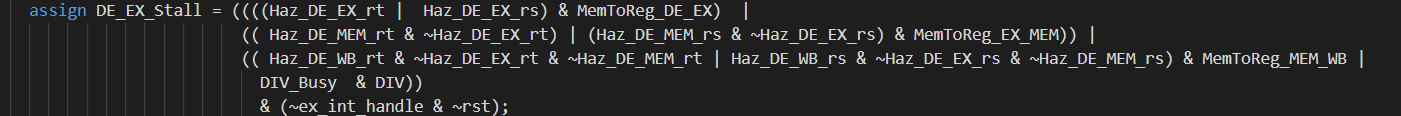
这一模块只增加了对trap的逻辑操作：

。

这是一个对陷入的追加判断。一旦陷入，陷入处理程序被规定在各自的进程上下文中执行。

---

上面那个trap\_flag在这一次已经删除了。实际的变动仅仅是在这个延迟判断中增加了ex\_int\_handle信号。



1. divider模块

一个运用迭代的除法器。busy和done信号分别表示除法是否正在运行和是否已经做完，从而在除法进行时对于其他操作阻塞处理。

实际上是一个辗转相除法的处理，利用时钟周期，和count，完成一个busy和done的判断从而达到阻塞效果。

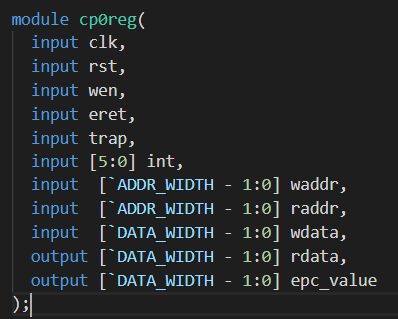
1. Multiplier模块

此模块原本实现了booth算法的乘法器，但是由于是和div除法器一样利用了时钟周期，进行了一个状态机的操作，而导致在测试mul\_tb的时候由于时钟延迟，没有办法获得正确结果。后来只是简单采用了乘号。（具体代码可见multiply.v文件）。

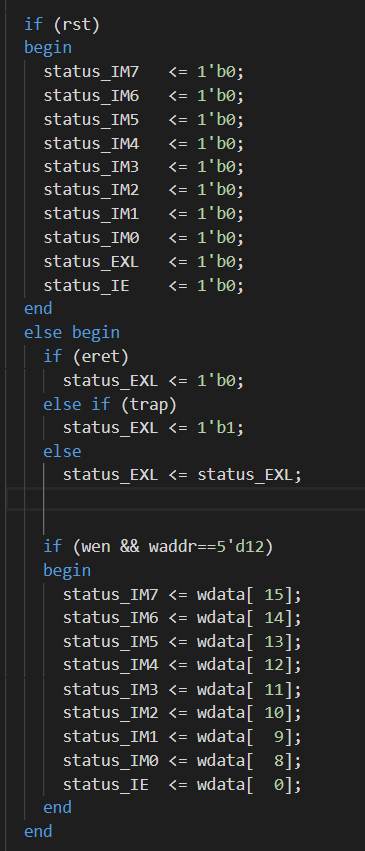
等之后时间富裕之后再添加信号busy和done等尝试完成这个乘法器。

1. Cp0reg模块

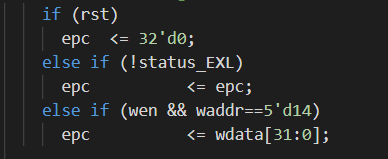
Cp0寄存器的处理模块，也可以说是这次例外中断的核心模块之一。



输入以下信号，用以判断是否是eret或者陷入指令，并且会返回epc的值。

具体操作的时候，将status和cause寄存器每部分单独罗列出来判断再整合（列举status的操作） 

除了对于status和cause寄存器的操作，还有epc寄存器

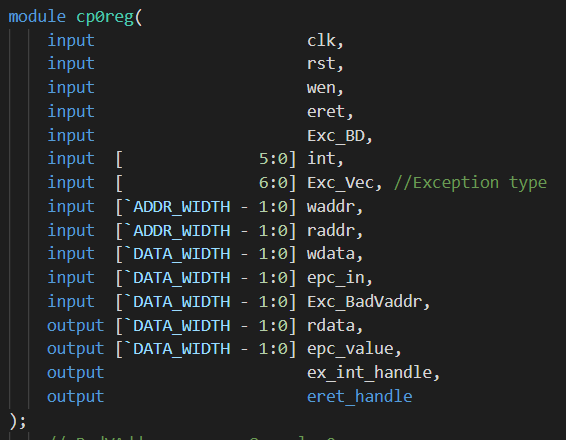


一旦EXL位为1那么在发生新的例外的时候epc的值并不进行更新。

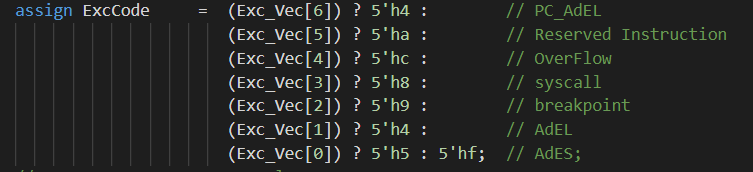
最后通过rdata将值输出出去。

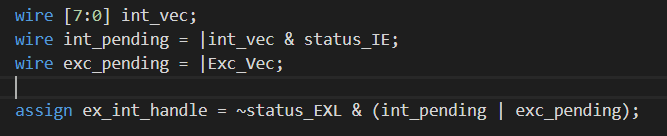
---

增加几个例外中断的信号



并且增加对应的cp0寄存器逻辑。







将IP7和时钟中断联系起来，根据时钟中断回改。

输入的int默认为6’b000000，对于跑4-2测试。但如果要进行记忆游戏等，就要将这个打开。

三、实验过程（60%）

（一）实验流水账

11月14日，那两天在写12306并没有动cpu

11月15日，回顾一下ppt，看了一下MIPS指令手册以及cp0寄存器的详细情况

11月16日，下午5点到晚上12点，开始动工cpu

11月17日，下午4点到晚上9点，中断例外基本框架写完，开始跑测试，失败

11月18日，下午4点到晚上11点，debug，通过69条测试

11月19日，下午5点到晚上10点，企图增加后续指令，结果69测试点都跑不过了……回退回去

11月23日，开始看任务书……因为我之前都在爆肝12306以及准备人工智能考试

11月24日，下午6点到第二天3点，找出之前回退前的版本继续写，写出基本代码进行测试，69测试点ERROR

11月25日，下午3点到第二天3点，研究为什么连69都跑不过，找到问题，继续测试

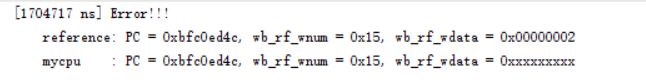
11月26日，下午6点到晚上12点，PASS

（二）错误记录

具体描述实验过程中的错误，环境问题、仿真阶段、上板阶段的都可以记录。

1、错误1

（1）错误现象



IMG_256

第35个测试点开始报错

（2）分析定位过程

有Z调Z，HI和LO的连接在更改过程中出了问题

（3）错误原因

删除注释冗余内容时不注意删掉了之前代码的寄存器定义

（4）修正效果

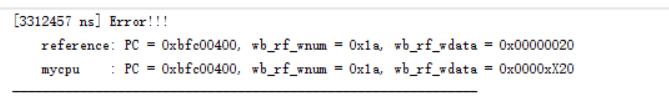
增添回来，完成。

（5）归纳总结（可选）

代码删改需备份，需谨慎。

2、错误2

（1）错误现象



前68测试点全过之后第69个测试点过程中的问题

（2）分析定位过程

寻找中间部分xx的原因

（3）错误原因

这是在我加了break信号和其他例外之后出现的问题，初步判断为新加入的信号扰乱了之前的逻辑。

（4）修正效果

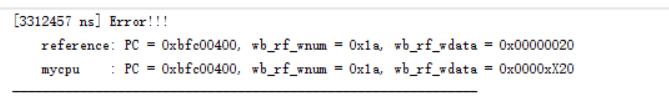
将后增添的部分先行删除，回退到原先版本。

（5）归纳总结（可选）

其实应该更仔细分析，这个xx一定不是因为这条指令出错，而是之前某条指令开始写ram的时候出现了问题，导致这次取data出现xx。因为这周比较爆炸所以没来得及处理这一问题，留待下次解决。

3、错误3

（1）错误现象



前68测试点全过之后第69个测试点过程中的问题

（2）分析定位过程

寻找中间部分xx的原因

（3）错误原因

这是在我加了break信号和其他例外之后出现的问题，初步判断为新加入的信号扰乱了之前的逻辑。

（4）修正效果

这一次有空去找原因了，原因是我之前把硬件例外的6位默认为0，而继续往下写的时候以为要实现硬件中断支持，所以把这个默认值注释掉了，这才导致了xx的出现。重新修正默认值之后就通过了。

（5）归纳总结（可选）

可能还是要按部就班一点，不能一激动就注释掉了，完了过段时间自己又忘了这茬事情……

4、错误4

（1）错误现象

PC全是0

（忘记截图了）

这是我困扰了几小时的问题

（2）分析定位过程

Gold\_trace生成问题？

（3）错误原因

说实话没有找到究竟是为什么，甚至我跑上一次4-1的代码也全是0，没有跳转没有ERROR没有PASS，重新生成了trace文件也不得行

（4）修正效果

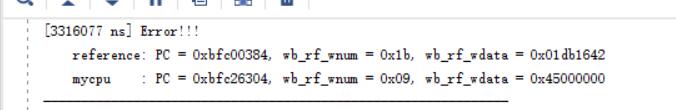
放弃修正，重新解压了一个框架从头来过……

（5）归纳总结（可选）

可能我每次实验都得重新解压一个框架吧？

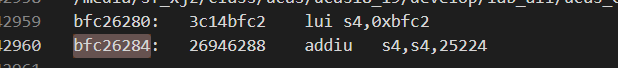
5、错误5

（1）错误现象



（2）分析定位过程

查看波形和test指令

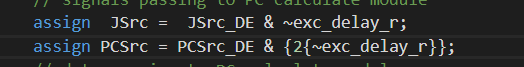




发现确实触发例外跳到了380，但是马上就跳回去了，猜测这是跳转延迟的问题。

（3）错误原因

实际上在execute模块这两句出了问题



之前没有写delay信号。

（4）修正效果

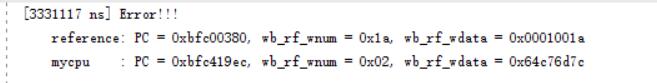
修正了之后69号测试点pass了

（5）归纳总结（可选）

添加信号的时候逻辑要更严谨一点，考虑周全一些。

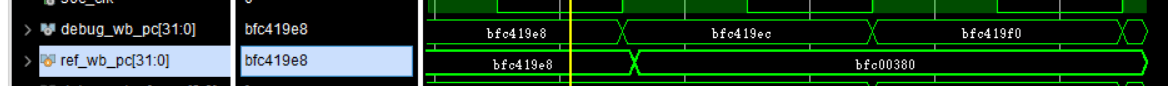
6、错误6

（1）错误现象



（2）分析定位过程

查看波形和test指令

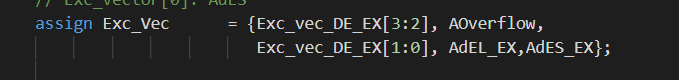


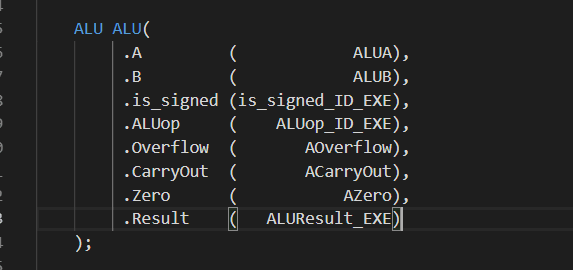
IMG_256

发现是整型溢出却没有跳转

（3）错误原因

查找ex模块的问题，然后在模块内发现我之前alu输出信号不需要用到overflow信号因此注释掉了，但实际上这一次判断需要有。





（4）修正效果

修正了之后其实还有问题，但是我觉得ex模块没有问题了，最后发现是我调用ex模块的时候忘记写is\_signed接口了，加上之后就没问题了。

（5）归纳总结（可选）

粗心使人de不出bug或者de越来越多的bug

四、实验总结（可选）

这一次的实验报告写的真的非常简陋了……因为要准备人工智能期中考试和数据库系统12306的实验验收（orz），实验做的比较仓促，里面可能还有些东西自己也还没完全弄明白，可能需要拖到4-2的时候再自己搞清楚。因为实际上本来有直接把4-2也写了的野心，但是写了之后发现甚至连4-1的测试都跑不过了，说明理解方面可能还是有些问题。

按部就班来，以及代码注意备份。说实话当时写了后面内容之后改回原本能过的版本花了我很久很久……要是改不回来就凉了啊orz。

因为用的control模块总体控制，每级一个模块，很多信号，越来越多的信号要逐层传递，无论是写的时候还是debug的时候都非常麻烦，可能是时候考虑一下老师给的代码风格了。

---

仍然比较仓促，不过debug中也有很多收获就是了。人最开心的大概就是从closed到open了。