**Lab 6报告**

学号 2020K8009908024

姓名 陈卓 史文轩 安金鹏

箱子号 69

一、实验任务（10%）

该实验的目的是在CPU中增加异常和中断的支持，包括设计状态控制寄存器，增加各种异常和中断的处理电路，增加CSR相关的指令，以及设计相关的阻塞和前递。

二、实验设计（40%）

（一）总体设计思路

该实验中，状态寄存器被单独设计为一个模块（csr.v）。所有读写csr模块的行为都被安排在写回级。

预取值模块中，增加了对取指地址错（ADEF）这一例外的判断，并把判断结果随pc值传到取值级。

取指模块中，增加了跳转到例外处理地址的功能。当出现异常或中断时，next\_pc的值取csr\_pc的值，并在下一时钟周期跳转至该地址。

译码模块中，增加了对csr相关指令、syscall指令和ertn指令的译码，于实验13中又添加了break指令、与读计数器有关的指令。由于csrwr、csrxchg会写入rd寄存器，导产生写后读相关，因此也增加了相关的阻塞部分；同时，对系统调用（SYS）、断点（BRK）、指令不存在（INE）这些例外的表示信号以及从状态寄存器模块传过来的中断信号（INT）都在译码级得到，并不断往后传到写回模块中处理。

执行模块中，首先是增加了对地址非对齐（ALE）这一例外的判断，并不断后传；其次是添加了一个64bit的计数器，该计数器用于满足rdcntvl\_w和rdcntvh\_w指令的要求，读出计数器中的计数。

写回模块中，由于读写csr模块的行为都在这个模块，因此增加了一些向csr模块传递的信号；同时，在这个阶段会对例外和中断信号做一些简单的处理和解析，反馈给csr模块，比如区分异常的ecode和esubcode信号。

当执行例外处理地址跳转以及返回异常发生地址时，由于我们将csr的读写安排在写回级，因此当异常指令到达协会级时，后面本不应该执行的指令已经进入流水线，因此，我们需要清空整个流水级，使得误取的指令不产生执行效果。

（二）重要模块1设计：CSR寄存器模块

1. 工作原理

该模块包含了所有实现的状态寄存器，包括CRMD、PRMD、ESTAT、ERA、EENTRY、SAVE0~3、ECFG、BADV、TID、TCFG、TVAL、TICLR。我们没有将各个状态寄存器分散到各流水级，而是集中到一个模块，与其他模块并列，这样不仅设计起来更加简洁，也方便调试。

所有读写该模块的行为都被安排在写回级，这样可以避免产生读写状态寄存器的写后读相关。

在写状态寄存器时，我们以每个状态寄存器的域为单元，对状态寄存器进行修改。

在读状态寄存器时，我们以每个状态寄存器作为一个单元，根据读地址csr\_num读取相关寄存器的内容。

1. 接口定义

|  |  |  |
| --- | --- | --- |
| 信号名称 | 输入\输出 | 作用 |
| clk | input | 时钟信号 |
| reset | input | 重置信号 |
| csr\_we | input | 状态寄存器写使能信号，在指令为csrwr、csrxchg时有效 |
| csr\_num | input | 状态寄存器地址 |
| csr\_wdata | input | 状态寄存器写数据 |
| csr\_wmask | input | 状态寄存器写掩码 |
| csr\_rvalue | output | 状态寄存器读数据 |
| wb\_ex | input | 写回级指令异常信号，在指令出现异常时有效 |
| ertn\_flush | input | 写回级为ertn指令信号 |
| wb\_pc | input | 写回级pc，用于传输异常发生地址 |
| csr\_pc | output | 用于给nextpc传输例外入口地址以及返回异常发生地址 |
| wb\_ecode | input | 例外类型一级编码 |
| wb\_esubcode | input | 例外类型二级编码 |
| wb\_vaddr | input | 出现地址相关例外时，出错的虚地址 |
| has\_int | output | 硬件或软件出现的有效的中断信号 |

3、功能描述

CRMD寄存器记录当前特权等级以及当前全局中断使能。

PRMD寄存器记录发生异常之前CRMD寄存器记录的特请点击和全局中断使能。

ESTAT寄存器的IS域记录中断状态，Ecode和Esubcode域用于记录例外类型。

ERA寄存器记录发生异常时的PC。

EENTRY寄存器用于配置除TLB重填异常之外的例外处理地址入口和中断入口

SAVE0~3用于暂存系统软件数据。

ECFG寄存器记录控制各中断的局部使能位，其LIE域有13位，每一位控制一个中断源。

BADV寄存器记录发生TLB重填异常和地址错误相关异常时，出错的虚地址。

TID、TCFG、TVAL、TICLR

当发生异常或中断时，相应的标志寄存器会被改动。此外，csrrd、csrwr、csrxchg指令可以将地址为csr\_num的状态寄存器的值写入通用寄存器，csrwr、csrxchg还会将通用寄存器中读出的数据与掩码相与并写入地址为csr\_num的状态寄存器，实现对状态寄存器的修改。

（三）重要模块2设计：阻塞模块

1、工作原理

由于所有读写状态寄存器的行为都在写回级，因此不会产生状态寄存器的写后读相关。

但csrrd、csrwr、csrxchg指令直至写回级才会修改通用寄存器，因此仍然会产生通用寄存器的写后读相关。因此，当csrrd、csrwr、csrxchg在执行级、访存级、写回级且译码级的指令与之有数据相关时，译码级的指令会被阻塞至上述三条指令离开流水线。

2、接口定义

执行级、访存级、写回级设置信号inst\_csr，判断当前流水级是否为上述三条指令且有效，这三个信号会被传给译码级。传输寄存器写地址rd的接口已在前面的实验设置，这里复用就行。

3、功能描述

当执行级、访存级、写回级输入的inst\_csr有效，且rd与rf\_raddr1或rf\_raddr2相同时，将ds\_ready\_go信号拉低，使得译码级指令被阻塞。

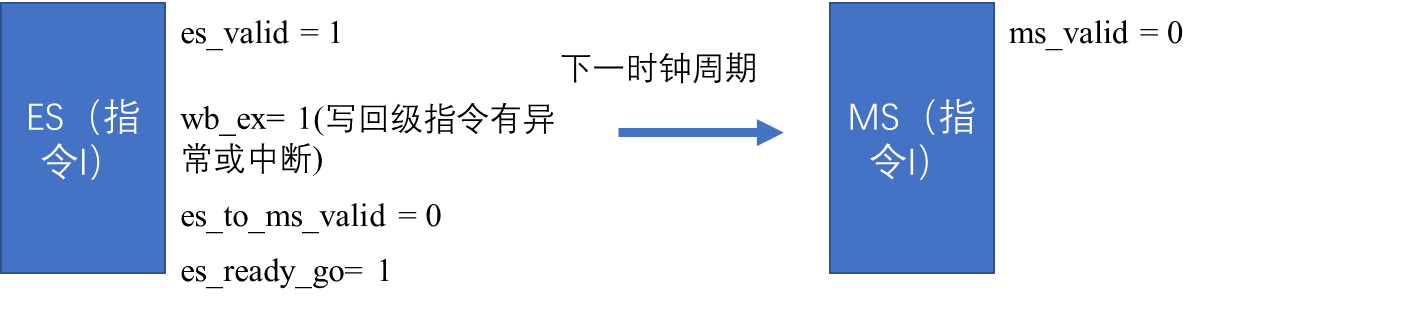
（三）重要模块3设计：控制相关的处理

1、工作原理

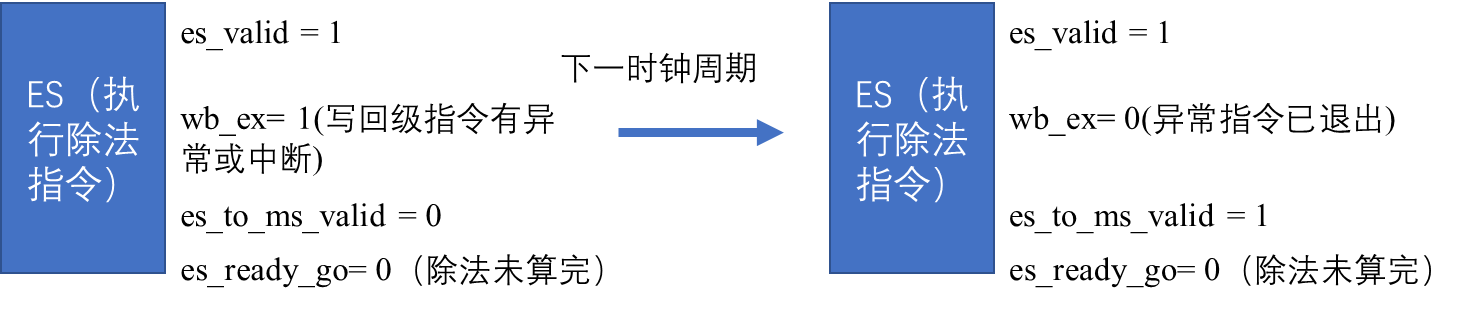
本实验中，在跳转至例外处理入口以及返回异常发生地址时，会发生控制相关。解决方法是清空流水级，使得误取的指令不产生执行效果

2、功能描述

当写回级的指令有异常或中断时，wb\_ex或ertn\_flush信号拉高。此时取指、译码、执行、访存级的ready\_go信号被置为1，xx\_to\_xx\_valid的信号被置为0。这样，当进入下一个时钟周期时，译码级、执行级、访存级、写回级的xx\_valid信号均变为0，就不会产生执行效果。



将ready\_go信号置为1是因为在写回级指令有异常或中断时，其前面的流水级可能会出现阻塞的情况，如除法指令，以及译码级的写后读相关指令。若不置为1，则会导致以下情况（以除法指令为例）：



这样，当除法指令算完，进入访存级时，ms\_valid仍为1，因此到写回级时，除法指令就会将结果写回寄存器，产生执行效果。因此，需要在发生异常或中断时将ready\_go信号置为1。

此外，st指令在访存级已经产生执行效果。因此，当执行级指令为st时，需要检查访存级或写回级的指令是否发生异常或中断。若发生，则data\_sram\_we被置为0，这样数据就不会被写入内存，产生执行效果。

三、实验过程（50%）

（一）实验流水账

10月16日，进行实验12的设计。

10月17日和18日，进行实验12的debug。

10月18日，仔细阅读了第七章相关部分的讲义。

10月19日，进行了设计和代码书写并开始debug。

10月20日，实验13debug完成。

（二）错误记录

1、错误1：ds\_ready\_go信号错误

（1）错误现象：

PC出错

图形用户界面, 应用程序

描述已自动生成

（2）分析定位过程

相关的汇编指令如下：

1c072668: 29800199 st.w $r25,$r12,0

1c07266c: 002b0000 syscall 0x0

1c072670: 0400000c csrrd $r12,0x0

1c072674: 03801c0d ori $r13,$r0,0x7

1c072678: 0014b58c and $r12,$r12,$r13

PC出错与syscall指令有关。

图形用户界面, 文本, 应用程序

描述已自动生成

可以看到，syscall指令到写回级时，next\_pc是正确的地址，但到下一时钟周期时却并未赋给fs\_pc。

fs\_pc赋值如下：

always @(posedge clk) begin

    if (reset) begin

        fs\_pc <= 32'h\_1bfffffc;

    end

    else if (to\_fs\_valid && fs\_allowin) begin

        fs\_pc <= next\_pc;

    end

调出fs\_allowin信号，发现为0，因此一开始以为是allowin信号出错，做了如下修改：

else if (to\_fs\_valid && (fs\_allowin || wb\_ex || wb\_ertn\_flush)) begin

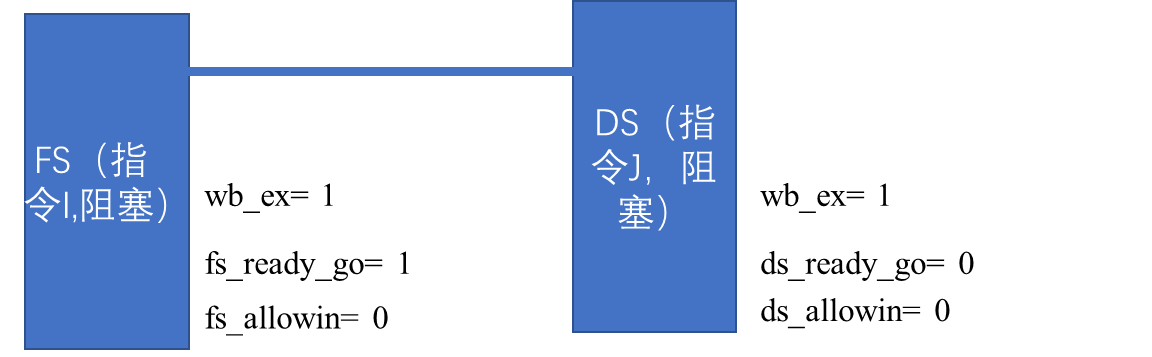
但测试仍未通过。

后来查看汇编代码，发现csrrd和and有数据相关，csrrd在访存级，and在译码级，产生了阻塞。

由fs\_pc的赋值可知，此时ds\_allowin为0，导致fs\_allowin为0。因此，next\_pc的值并未传给fs\_pc，导致pc错误。

（3）错误原因

当写回级指令为异常或中断时，ds\_ready\_go在阻塞的情况下为0，导致以下情况：



fs\_allowin为0导致fs\_pc不更新，进而导致pc出错。

（4）修正效果

在写回级指令异常或中断时，将ds\_ready\_go拉高

修改前：

assign ds\_ready\_go    = !(

    es\_blk\_valid  &&(es\_rf\_dest == rf\_raddr1 || es\_rf\_dest == rf\_raddr2) ||

    es\_csr\_blk\_valid &&(es\_rf\_dest == rf\_raddr1 || es\_rf\_dest == rf\_raddr2) ||

    ms\_csr\_blk\_valid &&(ms\_rf\_dest == rf\_raddr1 || ms\_rf\_dest == rf\_raddr2) ||

    ws\_csr\_blk\_valid &&(rf\_waddr == rf\_raddr1 || rf\_waddr == rf\_raddr2)) ;

修改后：

assign ds\_ready\_go    = !(

    es\_blk\_valid  && !wb\_ertn\_flush && !wb\_ex && (es\_rf\_dest == rf\_raddr1 || es\_rf\_dest == rf\_raddr2) ||

    es\_csr\_blk\_valid && !wb\_ertn\_flush && !wb\_ex && (es\_rf\_dest == rf\_raddr1 || es\_rf\_dest == rf\_raddr2) ||

    ms\_csr\_blk\_valid && !wb\_ertn\_flush && !wb\_ex && (ms\_rf\_dest == rf\_raddr1 || ms\_rf\_dest == rf\_raddr2) ||

    ws\_csr\_blk\_valid && !wb\_ertn\_flush && !wb\_ex && (rf\_waddr == rf\_raddr1 || rf\_waddr == rf\_raddr2)) ;

2、错误2：es\_ready\_go信号错误

（1）错误现象：

PC出错

图形用户界面, 应用程序

描述已自动生成

（2）分析定位过程

相关的汇编指令如下：

1c0726f4: 002b0000 syscall 0x0

1c0726f8: 0020418c div.w $r12,$r12,$r16

1c0726fc: 5c00733e bne $r25,$r30,112(0x70) # 1c07276c <inst\_error>

PC出错与syscall指令有关。

根据上个错误的经验，猜测又是ready\_go的问题。

原来的代码如下：

assign es\_ready\_go    =

    (es\_inst\_div\_w|es\_inst\_mod\_w)     ? signed\_dout\_tvalid   :

    (es\_inst\_div\_wu|es\_inst\_mod\_wu)  ? unsigned\_dout\_tvalid :

    1'b1;

而汇编指令中的除法指令存在阻塞的现象，导致出现错误。

（3）错误原因

当写回级指令为异常或中断时，es\_ready\_go在阻塞的情况下为0，导致以下情况：

图形用户界面

低可信度描述已自动生成

此时fs\_allowin为0，进而导致next\_pc的值没有赋给fs\_pc。

（4）修正效果

在写回级指令异常或中断时，将es\_ready\_go拉高

修改后：

assign es\_ready\_go    =

    (es\_inst\_div\_w|es\_inst\_mod\_w) && !wb\_ex && !wb\_ertn\_flush     ? signed\_dout\_tvalid   :

    (es\_inst\_div\_wu|es\_inst\_mod\_wu) && !wb\_ex && !wb\_ertn\_flush  ? unsigned\_dout\_tvalid :

    1'b1;

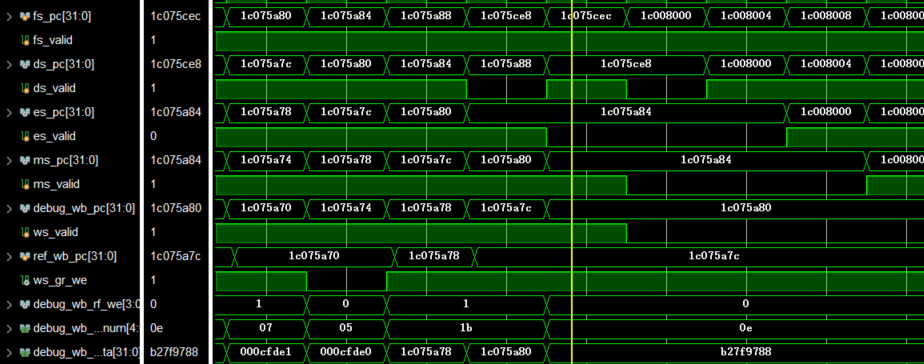
3、错误3：写回的数据出错

（1）错误现象：

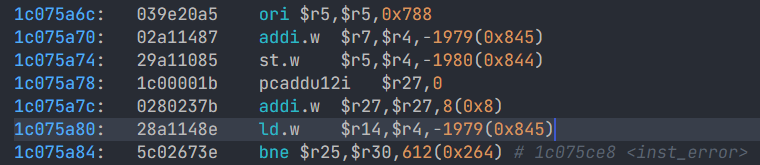
PC出错，ref的PC信号为0x1c075a7c，debug\_wb\_pc信号为0x1c075a80.

（2）分析定位过程

我们截取错误的波形如下图：



相关的汇编指令如下图：



我们可以看到出错的指令是一条ld.w指令，而且由于ref的指令在该ld指令之前都还是与debug信号相同的，我们只能考虑ld指令出错的可能性。再在此基础上合理怀疑ld在本阶段的写回状态：经查看，我发现debug\_wb\_rf\_we信号在ld指令处于写回阶段时是显示为0的，然而，ws\_gr\_we信号却显示为1，这显然不合理。因此，我查看了这两个信号的赋值代码如下：





可以发现错误应该就是在这里，由于我们在实际写回寄存器的时候，译码级接受的写回级的前递信号是rf\_we，因而实际上是写回了，但是该指令本来是不该写回的（由于这里的测试是在ALE异常处理的过程中，该条指令被标记了异常），而导致了错误。

（3）错误原因

由于未很好地区分不该写回的地方写回debug\_wb\_rf\_we信号和ws\_gr\_we信号，导致不该写回的地方发生了写回。

（4）修正效果

将rf\_we信号更正为出现异常时为0（异常不写回）。



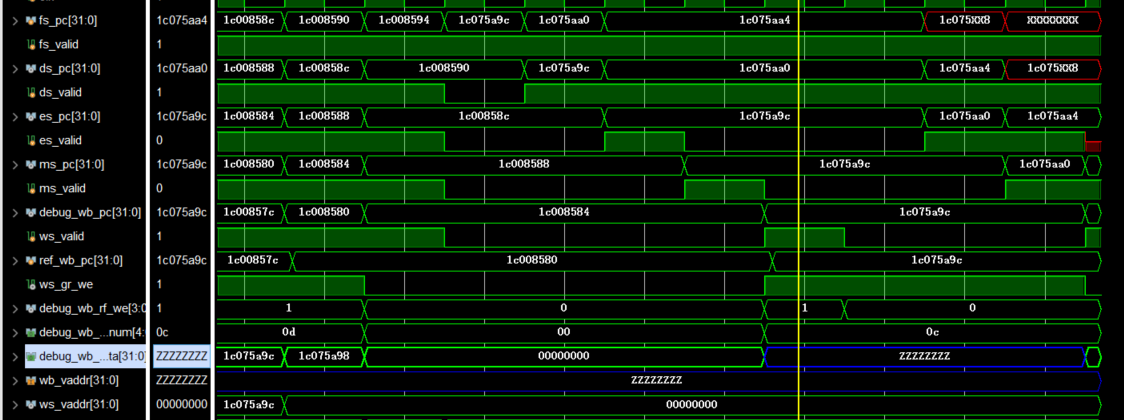
4、错误4：顶层信号没对应好

（1）错误现象：

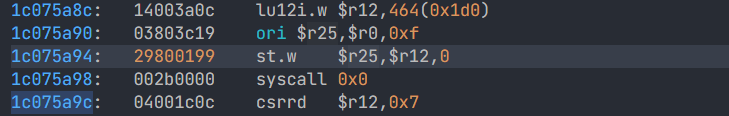
写回数据出现ZZZZZZZZ的情况。

（2）分析定位过程

我们截取错误的波形如下图：



相关的汇编指令如下图：



我们可以看到出错的指令是一条csrrd指令，此时写回的数据应该是从csr模块中的某个控制状态寄存器中读取出来的。但是无论如何读取，都不该是ZZZZZZZZ这种数据。一般来说， 出现Z信号都是由于模块间线没连好造成的。检查后发现是在顶层模块中实例化写回级模块和csr模块时变量名字没有对上，从而导致了错误。

（3）错误原因

顶层模块中实例化写回级模块和csr模块时变量名字没有对上，使得在写回级接受来自csr模块的寄存器数据时实际上没有连到一起，进而出现了高阻信号Z

（4）修正效果

将模块中的相关部分都改成一个变量名。

四、实验总结（可选）

本次实现了中断和例外等异常的处理，虽然有关控制状态寄存器的大部分代码都在讲义中给出了，但是本次实验由于需要消化理解的内容比较多，实现的细节也比较多，因而还是给我们组带来了不小的麻烦。本次实验的第一部分由陈卓同学完成，第二部分由史文轩同学完成，在我们都仔细阅读讲义之后，陈卓同学给我们组剩余二人仔细讲解了自己实现的内容以及一些细节，总体上来说交接的比较成功，在第二个同学实现自己的部分的时候，并没有因为陈卓同学的代码出现什么bug。在完成整个实验后，我们组的三位同学又进行了深入的交流，使得组内的每个成员都对实现的内容有比较充分的了解。