**实验8报告**

学号: 2018K8009929043 2018K8009929035

姓名: 曾冕 张翔雨

箱子号： 06

一、实验任务（10%）

本实验主要在lab7的前提下添加了对例外信号的处理，额外定义了cp0寄存器，并根据cause和status两个寄存器的位特性对内部的微信号进行了赋值，同时对不同流水级之间的总线信号进行修改，主要添加了延迟槽检测信号，例外信号和刷新信号等。

二、实验设计（40%）

（一）总体设计思路

本次的CPU在lab7（运算类指令实现）的基础上加入了对例外指令的支持。例外指令的实现主要涉及到了cp0寄存器的内容，在本小组的CPU设计中，这个模块是单独列出的分模块（类似regfile），在WB模块中完成的例化（这样方便后期的额外cp0寄存器添加），同时沿用讲义的信号名称，在不同流水级的传递中增添了流水级报错和是否延迟槽等各种信号，具体涉及的分模块涉及思路如下：

1. IF 模块设计思路：增添取指例外信号，是否处于延迟槽检测信号和例外代码等信号，并向下一级流水传递，同时next\_pc更改为四路选择
2. ID模块设计思路：例行对新指令译码（注意这里的译码参照了范例） ，增添了分支延迟槽，系统调用信号和对应例外代码，以及cp0寄存器的地址选择。
3. EXE模块设计思路：除了例外的例行信号处理外，对执行级的写内存和cp0寄存器添加了额外的写使能判断。
4. MEM模块设计思路： 除了例外的例行信号处理外，选择在访存级报出例外（eret和ms\_ex）信号
5. WB模块设计思路：在此阶段例化cp0寄存器，进行cp0写或读操作，同时输出flush信号和eret\_flush信号
6. Cp0模块设计思路： 按照讲义指导，将对应位信号写入

最后的总体设计思路如下图1所示。

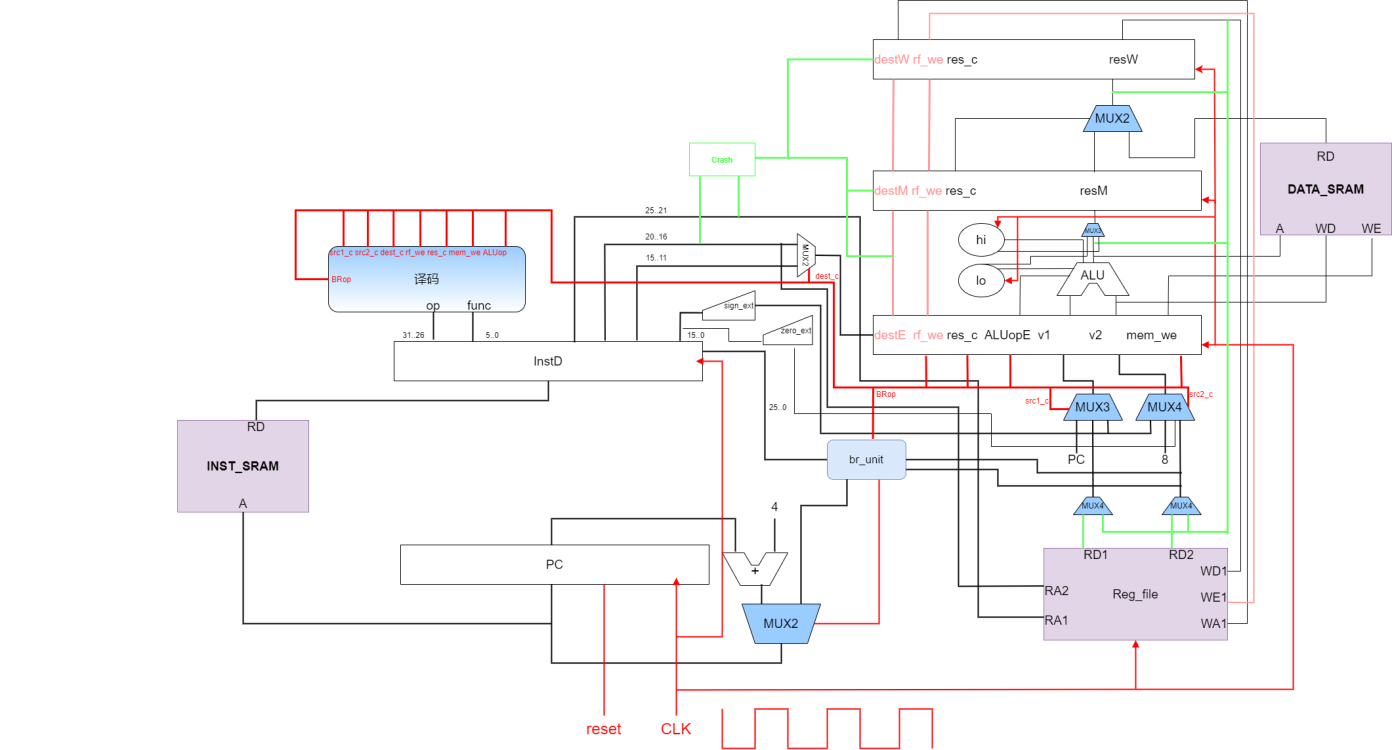


图01: 整体数据通路设计

（二）重要模块1设计：IF模块

1. 工作原理  
    根据上一条指令最后的WB传出信号，决定新PC取值，同时处理取值阶段可能出现的例外，以及延迟槽信号判断
2. 接口定义（仅展示lab8更新）

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| fs\_to\_ds\_bus | OUT | 71 | 到译码级流水，新增三个信号 |
| C0\_epc | IN | 32 | Eret操作返回的读出的epc的值 |
| ds\_branch\_op | IN | 1 | 是否处于延迟槽 |
| Eret\_flush | IN | 1 | Eret指令的刷新信号 |
| Ws\_ex | IN | 1 | WB阶段是否出错（例外跳转） |

1. 功能描述

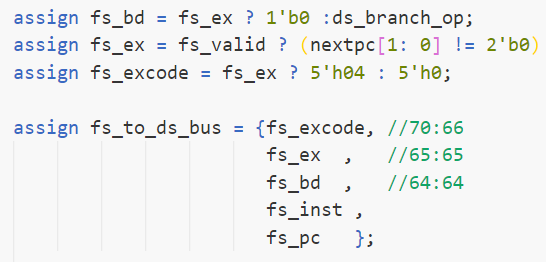
IF阶段例外处理，延迟槽判断以及向下一级流水的信号代码如下图01所示:

图01：IF阶段的例外处理代码

其中，读指令例外的例外代码位04，所以将该例外代码向下传递

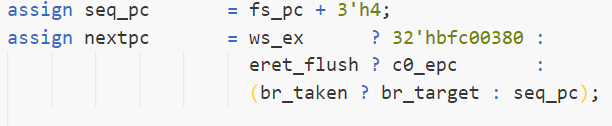
IF阶段还需要对PC的取值进行确定，使用了一个4路选择器，如下图02所示：

图02：nextpc的四选1代码

其中，如果是非eret的例外，下一拍应该是在例外处理代码的pc段0xbfc00380 。如果是eret指令，则将对应的epc值赋给nextpc，剩下两个则是正常跳转指令的pc选取

Ready\_go信号暂时不需要处理。

（三）重要模块2设计：ID模块

1. 工作原理

（lab8新增）传递出转移指令特征到if，完成是否在延迟槽的判断，完成mfc0等指令中有关cp0寄存器选择信号的拼接

1. 接口定义（仅展示lab6更新）

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| ds\_to\_es\_bus | OUT | 177 | 向exe的流水，具体增添部分见3的功能描述 |
| ds\_branch\_op | OUT | 1 | 延迟槽判断指令 |
| fs\_to\_ds\_bus | IN | 71 | 到译码级流水，新增三个信号 |
| Flush | IN | 1 | 流水线刷新信号 |

1. 功能描述

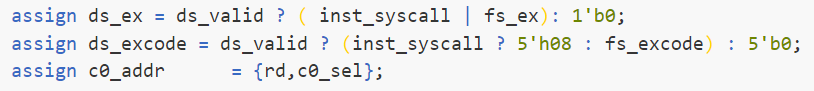
首先，对于译码阶段可能产生的例外信号以及上一级流水下来的例外信号的处理如下图3所示：

图03：译码阶段例外信号处理

根据mfc0等指令在指令手册上的要求对c0\_addr进行拼接处理。\_

转移槽判断指令代码处理如下图04所示：（列出所有的跳转指令）

图04：转移槽指令判断

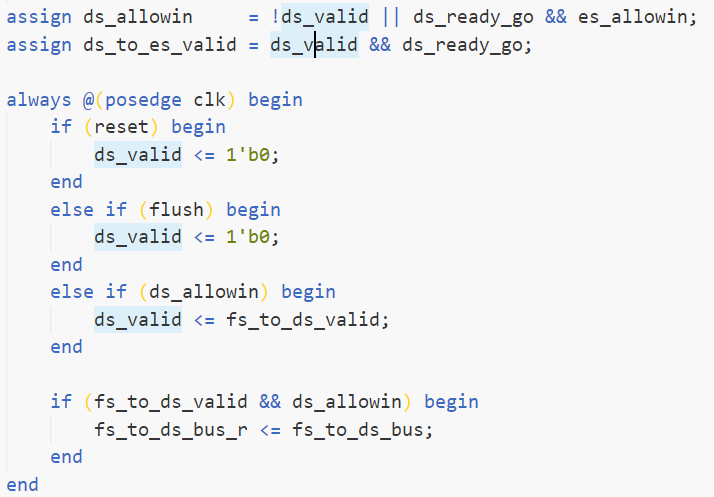
由于涉及到flush信号带来的刷新信号影响，将有效信号进行改变处理如下图05所示：

图05：转移槽指令判断

如果有flush信号 该流水级valid信号置0，流水正常运行，但流水级传递的数据无效化，与被阻塞的ready\_go情况有差异（后面几级的流水若涉及到需要有效信号改变的，使用了同种处理方式，故之后的只是介绍，不会再单独放图）

而ready\_go信号的处理也需要适当改变为如下形式，以保证在刷新时使得流水正常运行只是传递的信息无效：

assign ds\_ready\_go =    flush ? 1'b1 : !crash;

（四）重要模块3设计：EXE模块（仅描述lab8更新）

1. 工作原理

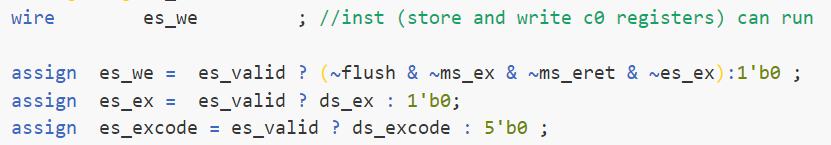
Lab8中，exe阶段需要判断后面两个流水是否出现例外需要刷新的情况，以保证在该阶段对hi和lo寄存器以及内存的写操作不会执行，产生错误。

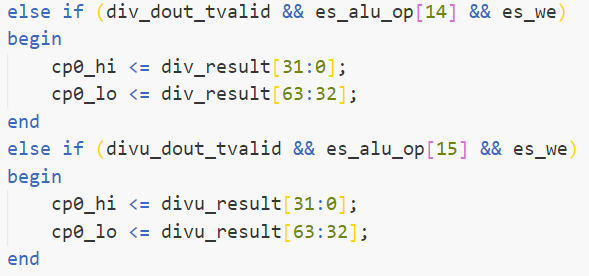
1. 接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| ds\_to\_es\_bus | IN | 177 | 来自ID的流水，具体功能略 |
| flush | IN | 1 | 刷新信号 |
| ms\_ex | IN | 1 | 上一条指令的例外信号 |
| ms\_eret | IN | 1 | 上一条指令是否为eret信号 |

3、功能描述

在EXE阶段需要定义额外的使能信号，来对store类指令和写cp0寄存器的指令进行控制。代码实现如下图所示：

图06：例外信号生成和指令使能信号处理

只有在本条指令不会产生例外，且上一条指令既不产生例外也不是eret指令的情况下，才会正常执行store和写hi，lo寄存器的指令。

在写cp0寄存器/向内存store内容时，更改条件，加入es\_we信号如下图7，8所示（仅展示部分代码）：

图07：写cp0代码段更新内容

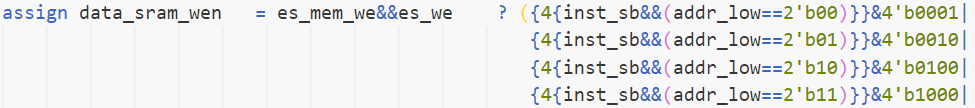


图08：store类指令更新内容

（五）重要模块4设计：MEM模块

1. 工作原理

本次实验中MEM阶段无需实现新功能，只需完成例外发生时流水级的刷新以及例外信号的传递即可。

1. 接口定义（仅展示lab8更新）

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| es\_to\_ms\_bus | IN | 98 | EXE阶段流水过来的数据，包括例外信息及代码 |
| ms\_to\_ws\_bus | OUT | 91 | Mem阶段向下一级的流水 |
| ms\_ex | OUT | 1 | 向外输出（报错）例外信号 |
| eret | OUT | 1 | 向外输出(报错)eret信号 |
| flush | IN | 1 | 刷新信号 |

1. 功能描述

此阶段例外的处理较为简单如下图9所示：

图9:MEM阶段的例外处理

在本次CPU设计中，错误的报出设置在了写回级，所以在接口处可以看到向外输出了ms\_ex信号和eret信号

（六）重要模块5设计：cp0模块

1. 工作原理

根据特权指令对各种cp0寄存器进行读写操作

2、接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| Mtc0\_we | IN | 1 | MTC0信号有效 |
| C0\_addr | IN | 8 | 选择哪一个cp0寄存器 |
| C0\_wdata | IN | 32 | 写入cp0寄存器的内容 |
| Wb\_ex | IN | 1 | 发生了例外（流水下来的例外信号） |
| Eret\_flush | IN | 1 | Eret刷新信号 |
| Wb\_bd | IN | 1 | 例外在延迟槽内发生（延迟槽信号） |
| Ext\_int\_in | IN | 6 | 硬件中断，暂时不需要使用 |
| Wb\_excode | IN | 5 | 例外信息（流水下来的） |
| Wb\_pc | IN | 32 | 该条指令的pc |
| C0\_status | OUT | 32 | 对应的status寄存器信息 |
| C0\_cause | OUT | 32 | 对应的cause寄存器信息 |
| C0\_epc | OUT | 32 | 对应的epc寄存器信息 |
| C0\_count | OUT | 32 | 对应的count寄存器信息 |
| C0\_compare | OUT | 32 | 对应的compare寄存器信息 |

1. 功能描述

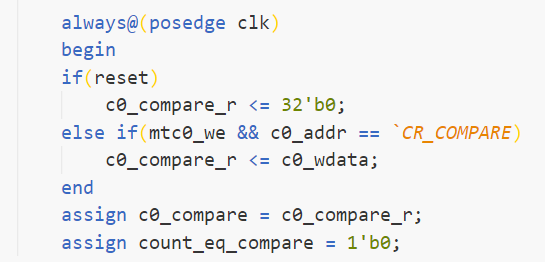
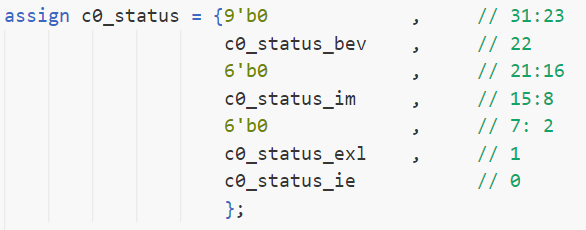
主要功能实现参照了讲义部分，讲义上已有的部分略去，展示一下compare域的写法和最后的拼接如下图10,11,12所示：

图10:compare域的处理



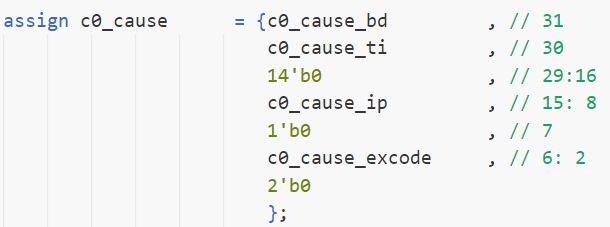
图11:cp0\_status的拼接

图12:cp0\_status的拼接

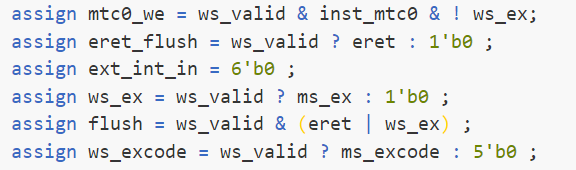
（七）重要模块6设计：WB模块

1. 功能描述（lab8更新）

例化cp0，对cp0寄存器进行读写操作，输出例外信号，双flush信号和epc信号。

1. 接口定义（仅展示lab8更新）

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| ms\_to\_ws\_bus | IN | 91 | EXE阶段流水过来的数据，包括例外信息及代码 |
| ws\_ex | OUT | 1 | 向外输出（报错）例外信号 |
| Eret\_flush | OUT | 1 | ERET的特殊刷新信号 |
| flush | OUT | 1 | 刷新信号 |
| C0\_epc | OUT | 32 | Cp0中epc的值 ，用于IF阶段取指令 |

1. 功能描述

写回级的错误信号处理代码如下图13所示：

图13:WB阶段的例外处理信号

硬件中断部分还不需要涉及，所以这里暂时赋值为0

前面模块中一直用到的刷新（flush）信号实在这里定义的，只要出现eret或者例外，就会向外输出刷新信号，其中如果是eret指令，还会额外给出一个eret级别的刷新信号，用于传递回IF阶段供next\_pc取值使用并传入cp0寄存器堆中以供使用。

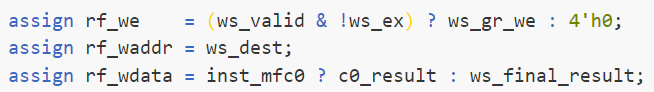
同样的，由于对cp0的读操作是要写到一个正常寄存器中，对于regfile部分作出如下处理：

图14:regfile相关的数据更新

其中需要对最后的写数据进行一个二选一，以及寄存器写使能信号多了一个确保没发生例外的判断条件

三、实验过程（50%）

（一）实验流水账

10.28晚上大约0.5小时 小组成员完成了cp0.v文件的编写

10.29 8:00-16:00设计代码并调试bug完成实验

10.31晚上 开始写实验报告，大约4小时半完成

（二）错误记录

1、错误1：写入epc的值出错

1. 错误现象

写入epc的值为全0

（2）分析定位过程

发现是epc的错误而cp0的写法是讲义上给出的应该不会错，所以转而去查找写入数据，最后发现在exe阶段的总线中数据正确，但是在其后一个阶段写入数据（既传入传出出错）就发生了错误，最后找到是exe阶段的es\_final\_result出了问题。

（3）错误原因

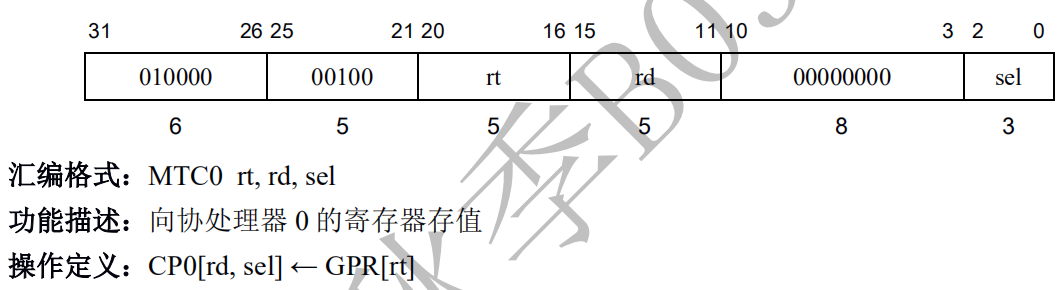
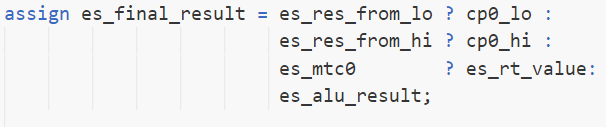
在mtc0指令的时候，没有正确更新exe阶段的final\_result的值（在exe阶段需要将原来的三路选择改为4路选择）

图15:mtc0指令手册上的操作定义

|  |
| --- |
|  |
|  |  |

（4）修正效果

将代码修改为下图所示的4路选择：

图16:修改后的代码图

正确运行到结束

四、实验总结（可选）

本次实验完成的例外信号处理主要涉及添加各个流水级的例外信号，例外代码和刷新信号，以及最后在WB段和MEM段的例外上报。最困难也是最陌生的部分cp0寄存器的代码在实验讲义中已经被很详细的定义了，减少了工作量。然后是关于例外的处理中需要刷新流水线这个操作如何实现的问题，这也是本次实验总体设计上最关键的一部分，在前面的实验中有了对实验框架的不断加深的理解，发现了流水级valid信号在刷新流水线操作中的可用之处，通过将valid信号置0，可以有效地将流水级之中的信息无效化，让流水线空转。

在本次实验中，我们小组将cp0单独列出作为一个分部分处理，这样方便后期其他例外/特权指令等的判断，但由于忘记了vivado添加源代码的操作，导致一直不能例化cp0文件报错，浪费了一部分时间。像这样的低级错误在本次实验中也多次出现，由于添加的信号比较多，导致出现大量的位宽定义或取值错误，以及信号命名错误，在下次实验中应该注意避免。