**实验七报告**

学号 2017K8009929044

2017K8009929025

姓名 李昊宸 李颖彦

箱子号 33

一、实验任务（10%）

(1) 深入理解译码和执行两个流水级的工作过程。

(2) 掌握在流水线 CPU中添加转移和访存类指令的方法。

二、实验设计（40%）

（一）总体设计思路

本次实验是添加三类指令：转移类指令，取数类指令，存数类指令。因此我们将其分为三个模块，逐个完善。添加指令的主要过程为译码，加入已有数据通路/创建新的数据通路。

（二）重要模块1设计：转移类指令模块

1. 工作原理

新增指令并译码，并相应的更新数据通路。

1. 功能描述

转移类指令可细分为三种：

1. BGEZ,BGTZ,BLEZ,BLTZ
2. J,JALR
3. BLTZAL,BGEZAL
4. 具体代码实现

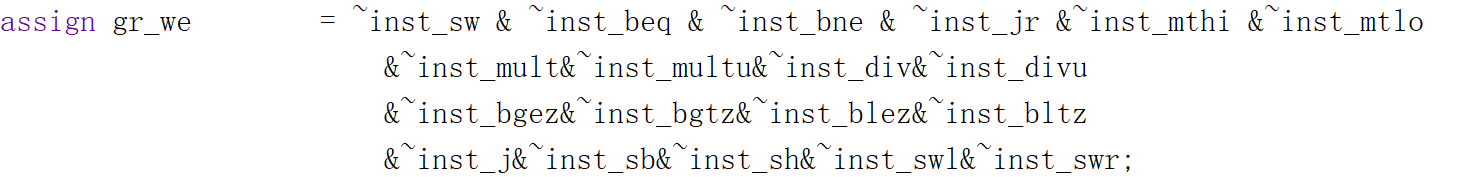
译码依照mips规定即可。

重点是数据通路的更新：

1. 第一类

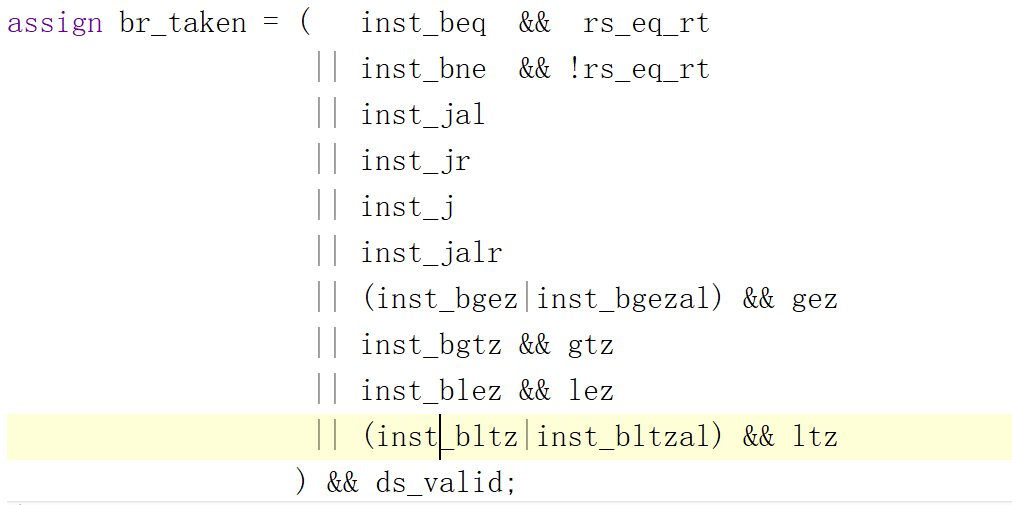
Bgez，bgtz这类指令可以复用beq的数据通路，稍作修改即可

1. Gr\_we：



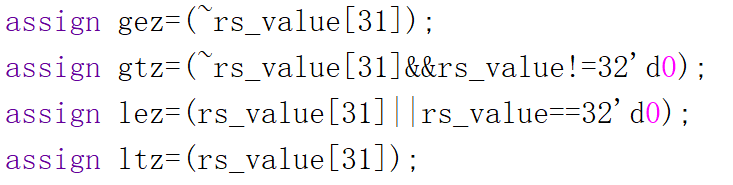
因为gr\_we代表着写寄存器操作，而bltz等指令并没有写寄存器

1. Br\_taken：



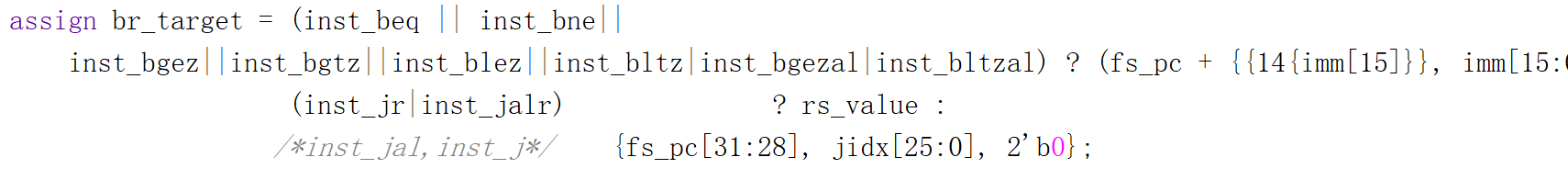
Br\_taken作为跳转的控制信号，势必要进行修改。

为此，新增四个信号，分别为gez,gtz,lez,ltz，顾名思义，代表数据与0比较的结果。



依照补码的特性，从最高位便可确定正负。

1. Br\_target



加入新指令并复用beq的通路即可。

B) j与jalr

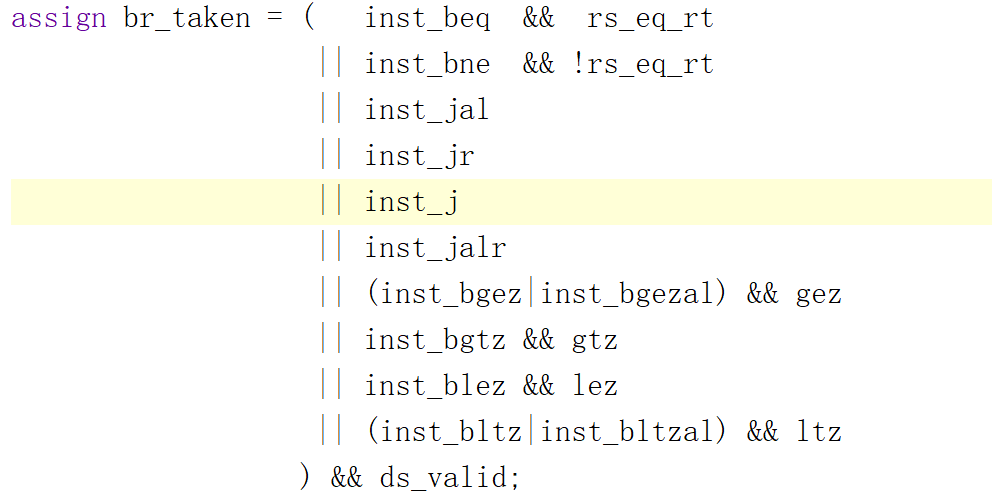
J指令作为一跳绝对跳转的指令，功能较为简单

1. gr\_we

前已有截图，不再赘述

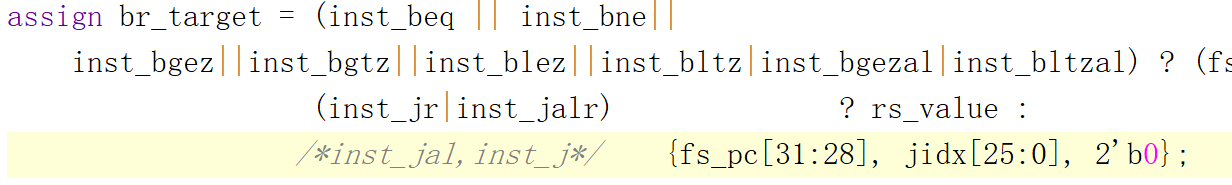
1. br\_taken

直接加入即可



1. br\_target

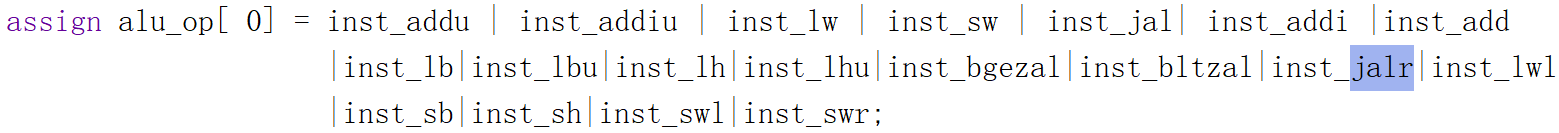
此处复用jal的数据通路



Jalr指令

1. alu相关

与jal相同，需要使用加法计算返回地址



第一个操作数是pc

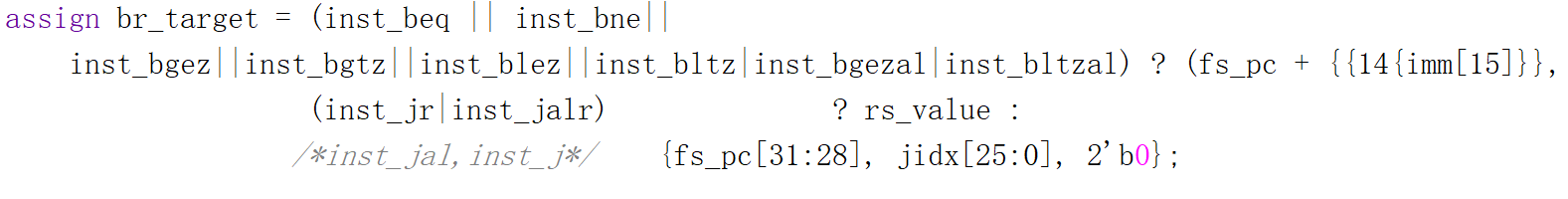


第二个操作数是8



1. br\_taken 与br\_target

复用jr的通路即可



1. bgezal与bltzal
2. br\_taken与br\_target

直接复用bgez与bltz的通路即可

1. alu相关

因为link，所以必须要记录返回值，这点与jal相同，复用其数据通路。

1. 重要模块2设计：取数模块

1、工作原理

在内存中取出数据之后写入寄存器中，主要复用lw的数据通路，需要新增一些特殊的数据通路。

2、具体代码实现

1）id阶段：

要修改的信号有alu\_op[0],load\_op,src2\_is\_imm，dest\_is\_rt，将这组的信号全部加入即可。并将这六条指令加入到id\_to\_exe,exe\_to\_mem的总线中。

2）mem阶段：

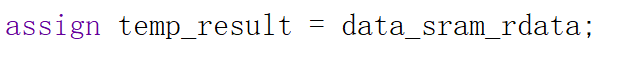
主要修改在mem模块。

1.首先定义loc信号：

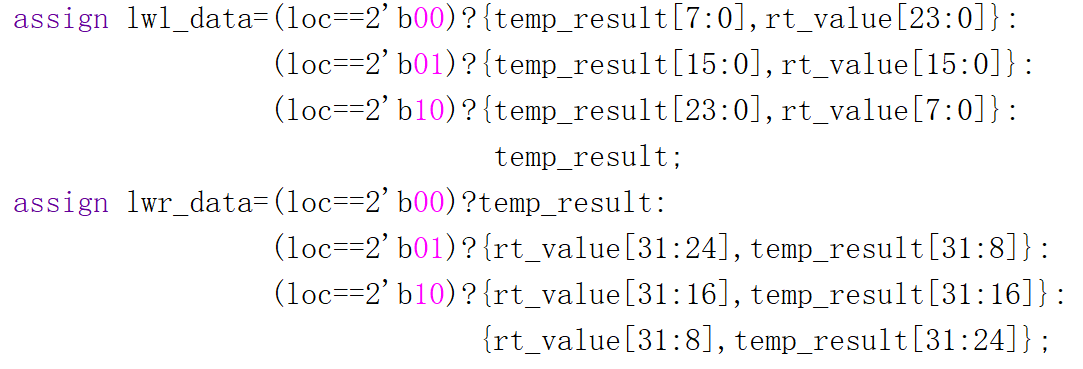


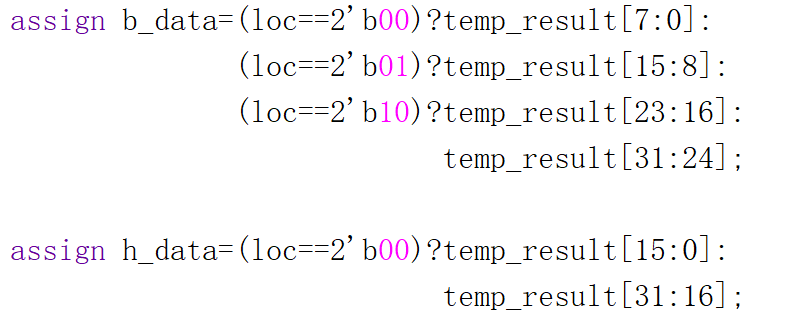
地址的末两位

2.将ram中读出的数据设为temp\_result

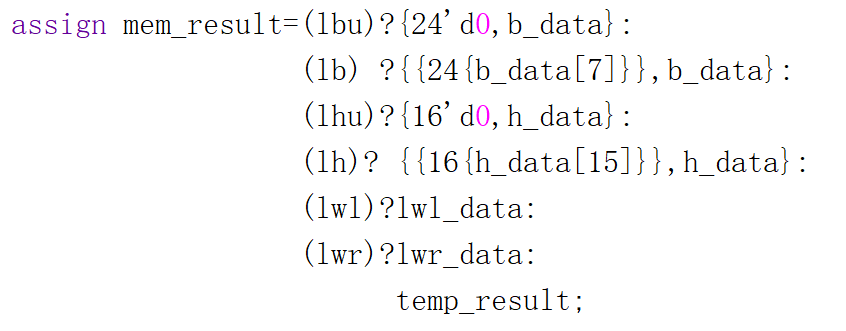


3.数据选择器





最后汇总：



1. 重要模块3设计：存数模块

1、工作原理

将寄存器中的数据写入内存中，主要复用sw的数据通路，需要新增一些特殊的数据通路。

2、具体代码实现

1）id阶段：

要修改的信号有gr\_we,mem\_we，存数类指令不写寄存器，但需要内存写使能。并将指令信号由id传至exe。

2）exe阶段：

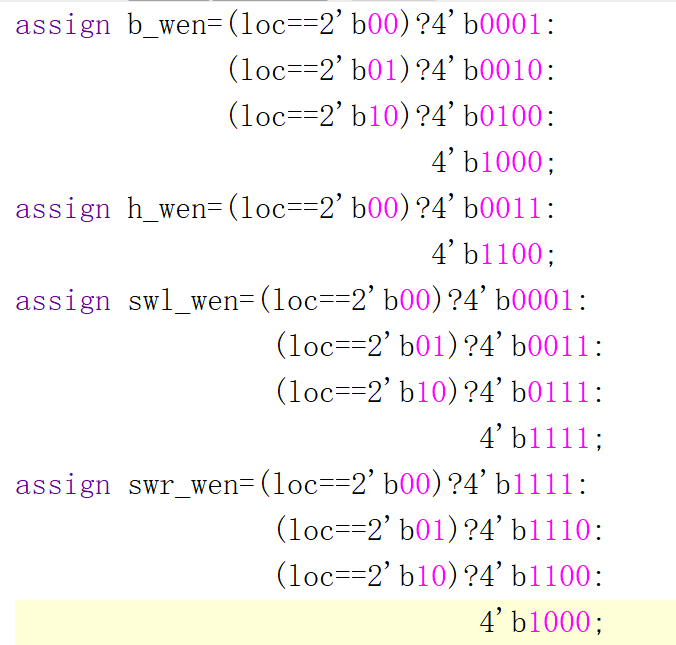
主要修改在exe模块。

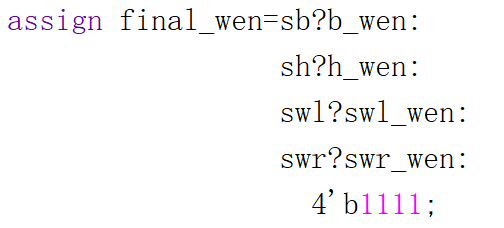
1.首先定义loc信号：



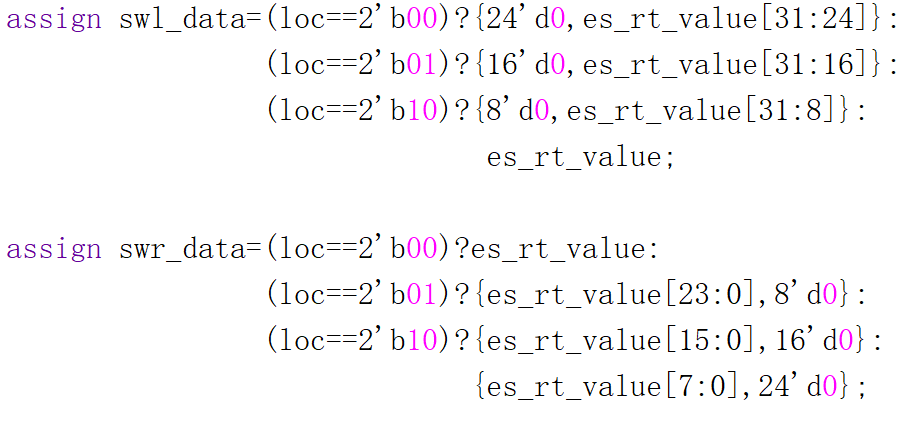
地址的末两位

2. 依据指令定义确定写使能：

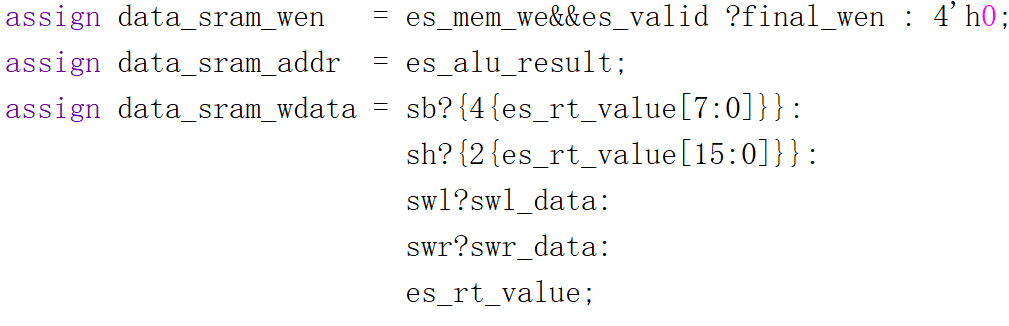




3.按swl和swr的要求对数据做特殊处理



4.汇总



值得一提，老师讲义上对数据的处理十分巧妙，采用{4{es\_rt\_value[7:0]}}形式，相较于按情况补0的做法，不仅功能完全实现，而且十分简洁，不需要无用的地线。

三、实验过程（50%）

（一）实验流水账

2019/10/15 14:00 – 17:00 加入指令

2019/10/15 18:00 – 20:00 debug相关

（二）错误记录

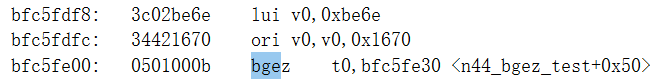
1、错误1：不该转移时转移

（1）错误现象

仿真出错，pc在该不应该进行跳转时跳转

（2）分析定位过程

查看反汇编代码



由bfc5fdf8的指令可知v0寄存器必为负数，此时bfc5fe00不应该发生跳转，可实际上发生了跳转。应该是判定出了问题。

（3）错误原因

一开始比较的指令如下：



乍一看没有问题，但仔细一想，verilog默认为无符号数，那么无论如何都是成立的！

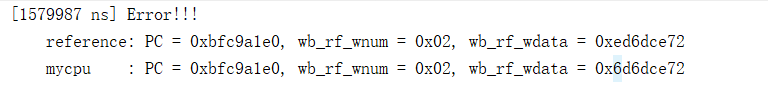
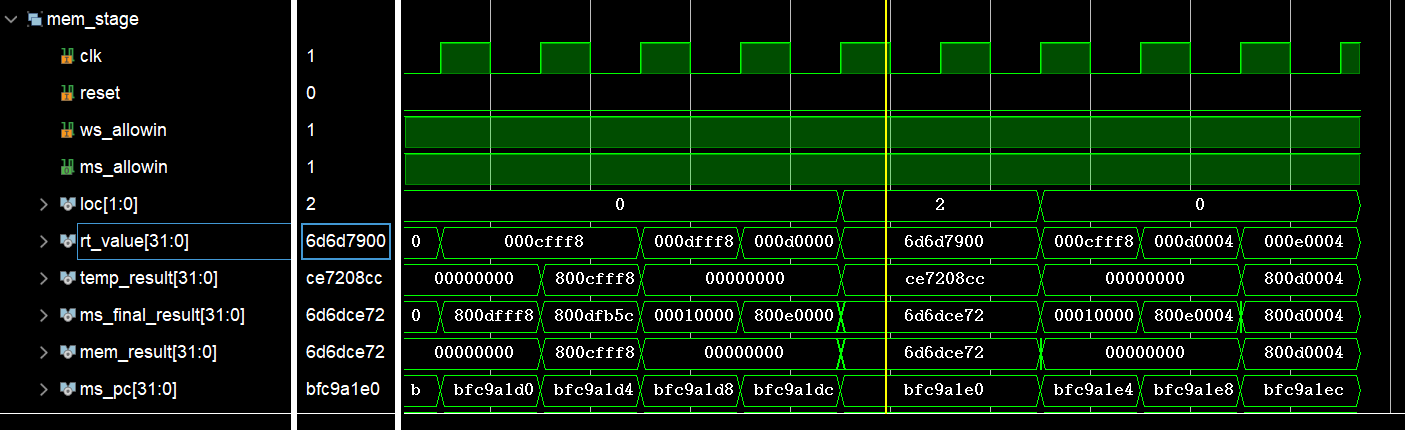
（4）修正效果



依照补码的特性按照最高为来判断即可。

2、错误2：e变6

（1）错误现象

在上图中，我们看到rf\_wdata仅有一位之差

（2）分析定位过程

一拍大腿，原来最高位由1变成了0，什么时候会有这种情况呢？

逐级回溯，发现在上一级的结果仍是正确的，那只能是信号传递出了问题。

（3）错误原因



原来数据通路少了一位，导致最高位由1变0

（4）修正效果

注意代码中的宽度位定义的宏减1，减去1一定不能忘记，所以原来的宏需要加1.

3、错误3：比对错误

（1）错误现象

在仿真过程中，发现myCPU 的PC值与refference不同。

（2）分析定位过程

一般来说，进行比对时，除非是跳转指令跳转错误，否则不会出现PC值不同的情况。查看反汇编语句后发现，在出错范围内并没有跳转语句跳转错误，地址跳转正确，但是仿真文件对比时所取的时间是正确时间的前一个周期。无处下手，于是查看了仿真文件。文件中描述取样是在寄存器写使能信号拉高和寄存器号非0的时候进行对比。冷静分析后发现，由于inst\_j指令没有加入到写使能信号的控制端，导致在写回模块时，控制寄存器写回的gr\_we信号非0，即拉高，恰巧j指令的rd位是有效的，导致仿真文件提前采样。

（3）错误原因

inst\_j指令的写使能信号忘记修改。

（4）修正效果

修改后，仿真文件正确采样。

四、实验总结（可选）

这部分写出了无数个bug，但感谢golden\_trace，对bug进行精确定位，大大加快了debug的速度。