**实验4&5报告**

学号：2018K8009929021

姓名：袁欣怡

箱子号：10

1. 实验任务

Lab4: 了解流水线冲突产生的原因，并在上次给出的无阻塞 CPU设计代码的基础上，设计阻塞五级流水线CPU。

Lab5: 针对流水线CPU中“写后读”这一特殊情况，实现前递技术，来更高性能地解决问题。

1. 实验设计

由于本实验在lab3的基础上修改代码而得，所以以下着重阐述lab4&5相对lab3的重大改动部分，改动较小的部分不再赘述。

## 总体设计思路

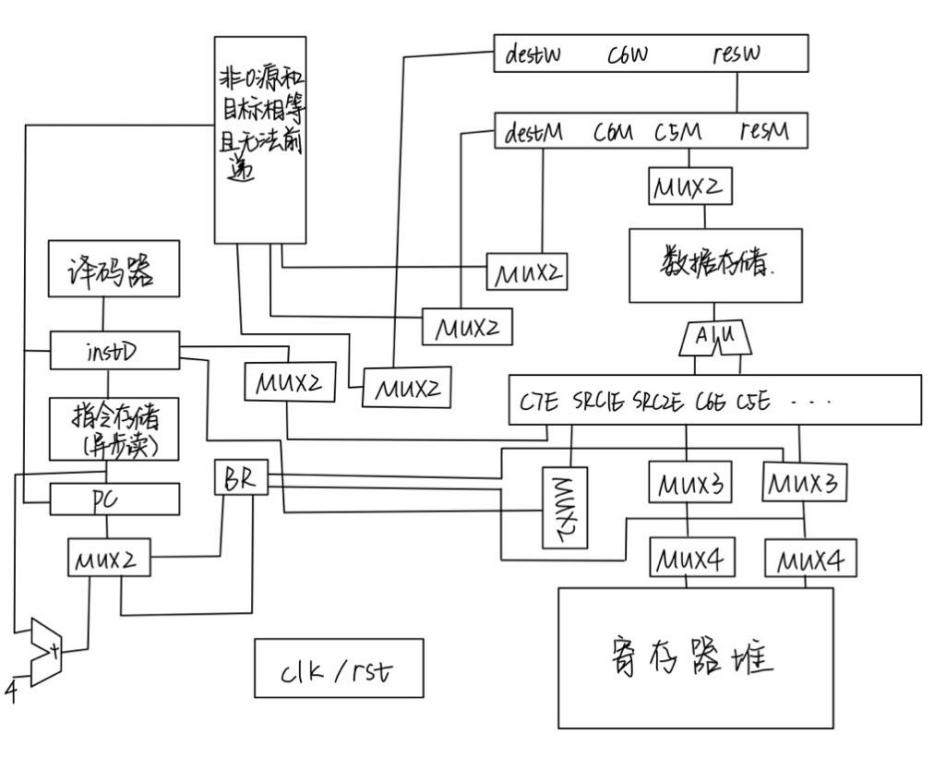


图1 硬件结构设计图

图示

描述已自动生成

图2 五级流水示意图

## 重要模块设计1: ID\_stage

1. 工作原理

从IF模块获取指令进行译码，判断指令格式的类型、ALU如何操作、是否需要加载、PC是否需要跳转等等，并将结果送至相应的模块。

相较于之前的无阻塞五级流水设计，lab4中设计的ID状态需要利用EXE、MEM、WB状态传数据，同时判断是否有“写后读”的行为，并判断是否需要进行阻塞；lab5中设计的ID状态需要添加旁路设计，让前面的指令直接把生成的结果传递给后面的指令。

在lab4中新介入了两个valid信号es\_to\_ms\_valid和 ms\_to\_ws\_valid，以及两条总线es\_to\_ms\_bus和ms\_to\_ws\_bus。需要注意的是，在ID模块中添加了新的端口之后，需要在cpu\_top中调用ID模块时相应添加新的端口。

根据分析，发生阻塞的原因有两种：寄存器读地址1和写地址冲突，或者寄存器读地址2和写地址冲突，而这些情况会发生在addu, subu, slt, sltu, and, or, xor, nor, bne, beq这些指令中。为了解决这个问题，设计了hazard, src1\_harzard和src2\_harzard三个变量，其中hazard结合了MIPS不同指令产生源操作数冲突的不同，src1\_hazard和src2\_hazard判断姨妈阶段的寄存器读地址与EXE、MEM、WB状态的寄存器地址是否有冲突，判断是否会出现写后读的现象。若hazard置为1，则ds\_ready\_go置为0。

在实现阻塞之后，还需要考虑CPU完成阻塞之后如何进行恢复。我采取的方法是判断EXE、MEM、WB三个阶段内的valid信号，当流水线发生阻塞之后，es\_to\_ms\_valid, ms\_to\_ws\_valid 和 ws\_to\_rf三个信号会从1变成0，阻塞完成后将hazard\_rst信号职位1，hazard信号置为0，数据恢复流动。

在lab5中，数据通路增加旁路设计，来让前面的指令直接把已经生成出来的结果直接转给后面的指令，采用“流水级组合逻辑的结果传递到译码级寄存器读出处”的方案。同时，还会通过后续阶段的valid信号和gr\_we信号来控制ID模块中rs\_value和rt\_value的值。

另外需要注意的是，假如EXE模块正在运行LW指令且会和当前ID模块正在运行的指令发生影响时，需要将ready\_go信号设置成“0”。

1. 接口定义

表1 ID模块接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| es\_allowin | IN | 1 | EXE模块允许接受ID传值 |
| ds\_allowin | OUT | 1 | 允许IF模块向ID模块传递数据 |
| fs\_to\_ds\_valid | IN | 1 | IF模块可以向ID模块传值 |
| fs\_to\_ds\_bus | IN | 64 | IF模块向ID模块传递数据（指令码及地址） |
| ds\_to\_es\_valid | OUT | 1 | 允许ID模块向ES模块传递数据 |
| ds\_to\_es\_bus | OUT | 136 | ID模块向EXE模块传递数据 |
| br\_bus | OUT | 33 | 输出是否跳转和branch的target给IF模块 |
| ws\_to\_rf\_bus | IN | 38 | WB模块向ID模块传递的需要写回REG FILE的信息 |
| es\_to\_ms\_bus | IN | 71 | EXE模块向MEM模块传递数据 |
| ms\_to\_ws\_bus | IN | 70 | MEM模块向WB模块传递数据 |
| es\_to\_ms\_valid | IN | 1 | EXE模块可以向MEM模块传值 |
| ms\_to\_ws\_valid | IN | 1 | MEM模块可以向WB模块传值 |
| out\_es\_valid | IN | 1 | 接收es\_valid是否为1 |
| out\_ms\_valid | IN | 1 | 接收ms\_valid是否为1 |

1. 功能描述

将IF模块获取的指令进行译码，并处理PC是否需要跳转，将结果返还给IF模块。将译码后的数据和控制信号传通过数据总线传递给EXE模块,EXE模块在下一时钟周期接受。写回阶段的数据也通过该模块传递给寄存器堆。此外，该模块会利用EXE、MEM、WB传回的数据，判断是否有“写后读”的行为、是否要进行阻塞。采用了前递的方式来减少CPU的阻塞时间，缩短运行时间。

## 重要模块设计2: WB\_stage

1. 工作原理

将从 MEM 模块获取的写回指令相应的执行。确定是否有写回指令，并进行相应的操作。同时，该模块还会将几个重要信号传递给 debug 模块，用于调试 CPU 的正确性。给ws\_to\_rf\_bus的位宽增加两位，用来传递ws\_valid和ws\_gr\_we两个信号给ID模块。

1. 接口定义

表2 WB模块接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| reset | IN | 1 | 复位信号 |
| ws\_allowin | OUT | 1 | WB模块允许接受MEM传值 |
| ms\_to\_ws\_valid | IN | 1 | MEM模块可以向WB模块传值 |
| ms\_to\_ws\_bus | IN | 70 | MEM模块向WB模块传递数据 |
| ws\_to\_rf\_bus | OUT | 40 | WB模块向寄存器堆模块（通过ID模块）传递数据（包括ws\_gr\_we、ws\_valid等） |
| debug\_wb\_pc | OUT | 32 | debug显示PC |
| Debug\_wb\_rf\_wen | OUT | 4 | debug显示寄存器堆写使能 |
| Debug\_wb\_rf\_wnum | OUT | 5 | debug显示寄存器堆写地址 |
| Debug\_wb\_rf\_wdata | OUT | 32 | debug显示寄存器堆写数据 |

1. 功能描述

确定是否需要写回操作，并相应地传递数据，同时和debug模块一起，辅助调试CPU的正确性。

## 重要模块设计3: mycpu.h

1. 工作原理

CPU头文件，包含很多位宽的宏定义。

1. 功能描述

在本实验中，给ws\_to\_rf\_bus的位宽增加两位，即将该文件中的WS\_TO\_RF\_BUS\_WD改为40。

1. 实验过程
2. 实验流水账

9.29 18:00-20:00 阅读讲义

10.11 20:30-22:30 进行lab4

10.12 8:30-11:30 进行lab4、lab5

10.12 14:00-17:00 撰写实验报告

1. 错误记录
2. 错误1:
3. 错误现象：PC值错误

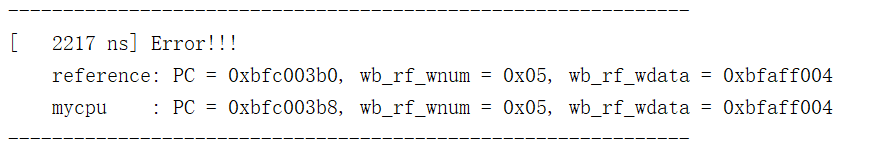


图3 PC值出错

1. 分析定位过程：

判断出当时出现“写后读”的情况，阻塞没有设置成功，跳转时没有跳转到正确的PC值。

1. 错误原因：

阻塞信号hazard没有考虑到全部的分支跳转指令，导致没有正确阻塞，让寄存器读到了错误的数据。

1. 修正效果

修正后可以进入阻塞态，正确完成分支跳转指令。

1. 错误2:
2. 错误现象：波形停止

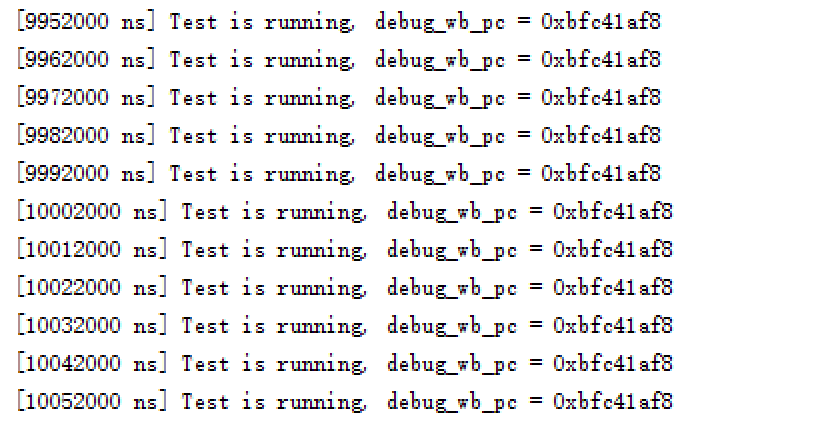


图4 PC值不发生变化

1. 分析定位过程：

PC值没有变化，推测是阻塞时没有恢复，于是发现hazard信号有一处拼写错误，导致没有赋值成功，一旦进入阻塞状态就无法跳出。

1. 错误原因：

阻塞信号拼写错误，导致阻塞信号赋值失败，一直没有从阻塞状态中跳出，进入新的状态。

1. 修正效果

修正后可以从阻塞态中恢复，完成后面的指令。

1. 错误3:
2. 错误现象：值为X
3. 分析定位过程：

在检查错误的过程中还发现新添加的es\_to\_ms\_valid和 ms\_to\_ws\_valid值始终为X，没有成功赋值。

1. 错误原因：

在ID模块中添加了es\_to\_ms\_valid和 ms\_to\_ws\_valid两个信号，但是没有在cpu\_top.v中调用ID模块时添加相应的端口，导致没有赋值。

1. 修正结果：

修正后信号的值恢复正常。仿真成功通过。

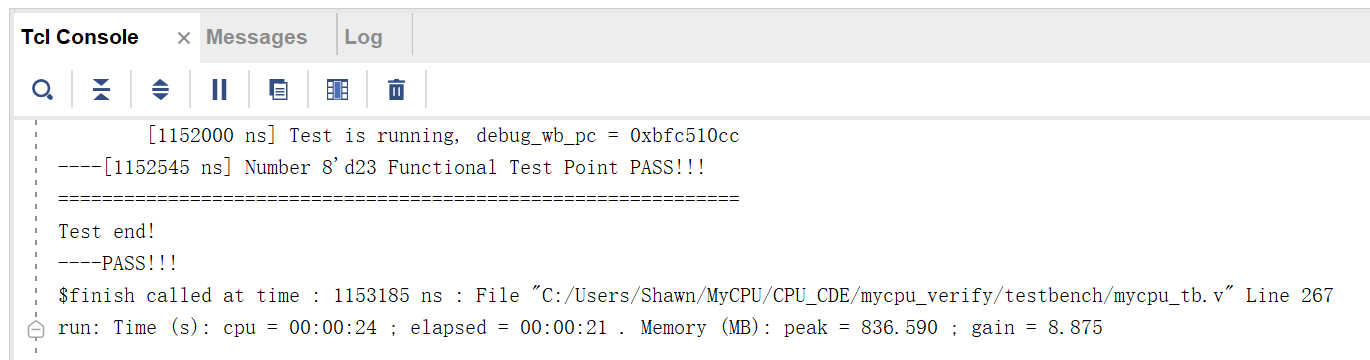


图5 仿真通过

1. 对比分析

有无前递技术的仿真时间比较：

无前递技术（lab4）:1,310,945ns

图片包含 文字

描述已自动生成

图6 无前递技术仿真时间

有前递技术（lab5）: 1,153,185ns（见图5）

从仿真时间不难看出，在运行相同的测试程序时，采用前递技术的流水线所用的时间缩短了12.03%，采用前递的运行效率提高。

1. 实验总结

修改所有错误后，上板成功，且采用前递技术的CPU仿真时间相对没有前递技术的有所减少。

通过这几次实验，学习了debug的常用方法，希望以后可以少出错。为了减少debug时花费的时间，应该先想清楚整个设计的框架再开始写，而不是边写边改。