实验 4&5 报告

学号：2018K8009929021
姓名：袁欣怡
箱子号：10

一、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。

二、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。

（一）总体设计思路

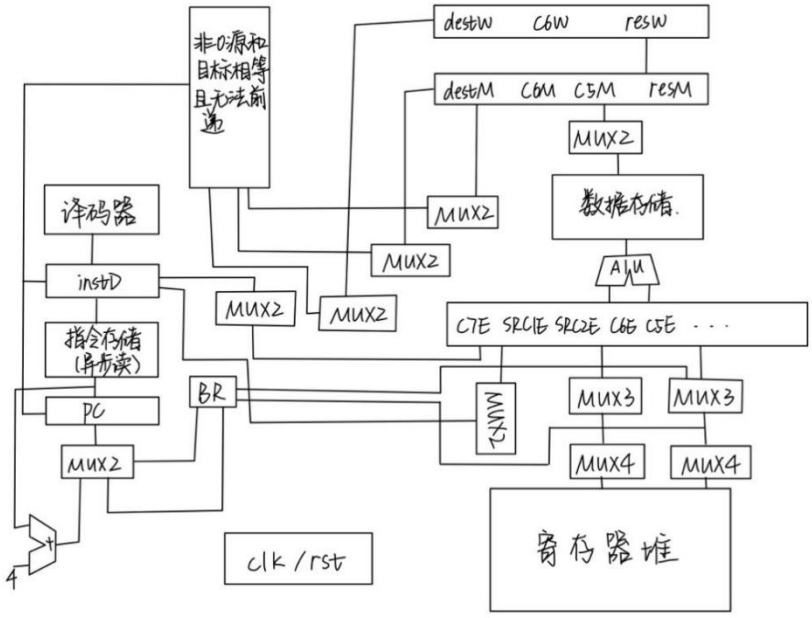

图 1 硬件结构设计图

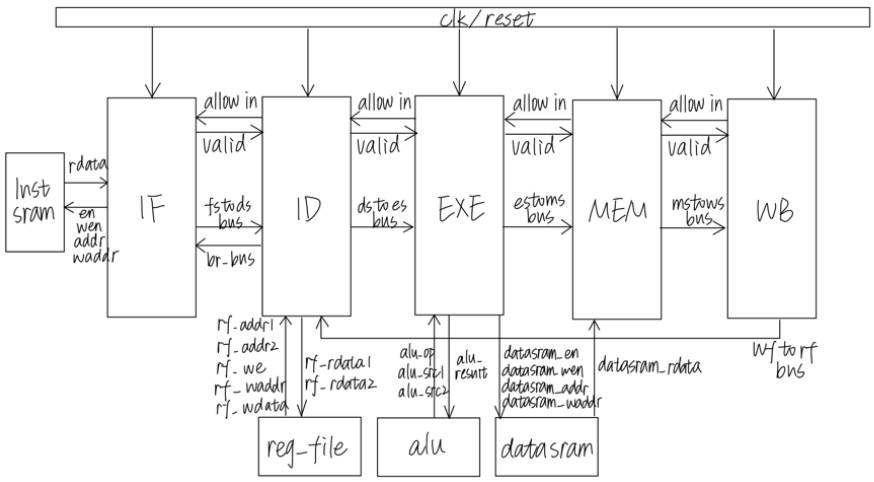

图 2 五级流水示意图

（二）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”。

2. 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

3. 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。

（三）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。

2. 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

3. 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。

（四）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。

2. 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。

三、实验过程

（一）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

（二）错误记录

1. 错误 1：

（1） **错误现象：**PC 值错误

```
------------------------------------------------------------
[   2217 ns] Error!!!
    reference: PC = 0xbfc003b0, wb_rf_wnum = 0x05, wb_rf_wdata = 0xbfaff004
    mycpu    : PC = 0xbfc003b8, wb_rf_wnum = 0x05, wb_rf_wdata = 0xbfaff004
------------------------------------------------------------
```

图 3 PC 值出错

（2） **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。

（3） **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。

（4） **修正效果**

修正后可以进入阻塞态，正确完成分支跳转指令。

2. 错误 2：

(1) **错误现象：**波形停止

```
[9952000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[9962000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[9972000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[9982000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[9992000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[10002000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[10012000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[10022000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[10032000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[10042000 ns] Test is running, debug_wb_pc = 0xbfc41af8
[10052000 ns] Test is running, debug_wb_pc = 0xbfc41af8
```

图 4 PC 值不发生变化

（2） **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。

（3） **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。

（4） **修正效果**

修正后可以从阻塞态中恢复，完成后面的指令。

3. **错误 3**:

（1） **错误现象：**值为 X

（2） **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。

（3） **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。

（4） **修正结果：**

修正后信号的值恢复正常。仿真成功通过。

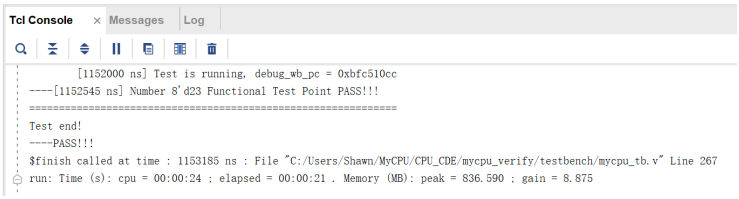

图 5 仿真通过

（三）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

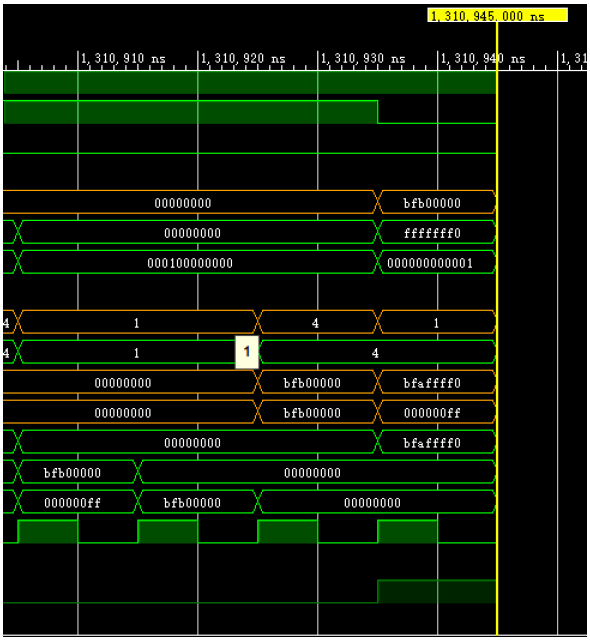

图 6 无前递技术仿真时间

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。

四、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。