实验 2-1 报告

学号：2016K8009908007
姓名：薛峰

一、实验任务

实验目的：

（一）、设计一款静态 5 级流水简单 MIPS CPU，可以不用考虑数据相关；

（二）、至少支持 lab1 要求的 19 条机器指令；

（三）、要求实现 MIPS 架构的延迟槽技术；

检验方法：

要求仿真和上板运行 lab2_func_1 通过。

二、实验设计

（一）总体设计思路

该项目总体分为以下四个模块：mycpu_top，控制模块（cpu_control），寄存器堆(reg_file)和 ALU。

其中 mycpu_top 模块为顶层模块，控制其他三个模块进行工作。除 mycpu_top 模块外，其余三个模块同之前实验，基本未作变动(仅规范了控制模块代码风格)，因此在此仅介绍 mycpu_top 模块。

（二）mycpu_top 模块设计

1、工作原理

为实现五级流水线，首先需要明确各个流水级的任务。

取指 IF：根据 PC 取指令；

译码 ID：指令取回，根据指令生成控制信号；读寄存器堆；判断 ALU 的输入；判断写寄存器堆的地址；根据 ID 阶段产生的控制信号控制 PC 的更新；

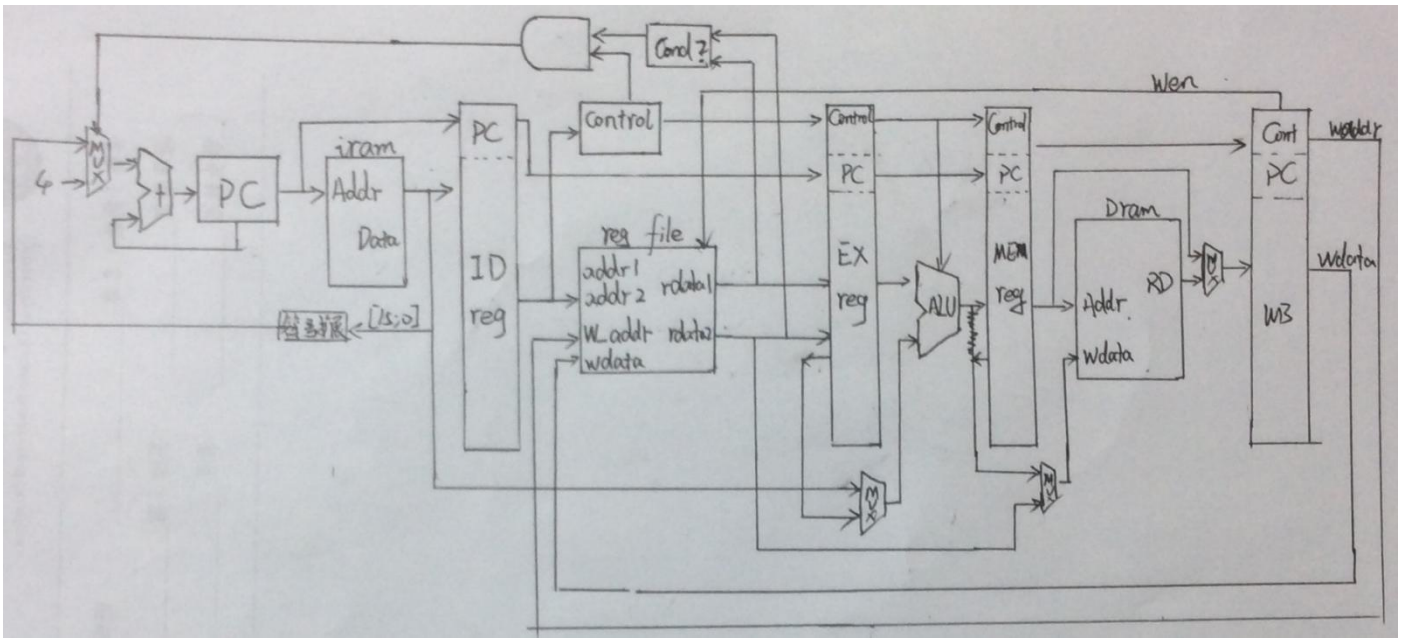
执行 EX：将 ALU 的操作数送入 ALU，产生运算结果；判断分支指令是否跳转；判断写内存的内容；

访存 MEM：写内存或读内存；

写回 WB：内存数据取回；写寄存器。

其中主要需要考虑的是如何用多个寄存器来存放中间结果，例如在对于某条指令在 ID 阶段产生的控制信号 MemtoReg 在 WB 阶段用到，因此需要几个寄存器保存 MemtoReg 的值。并且对于存放中间结果的代码部分应理清思路，时序关系容易出错。

2、结构设计图



3、功能描述

由于本实验不考虑数据相关，因此还未加入 allowin，valid 等信号。

项目的关键在于中间结果的流动，即如何保持一条指令的控制信号、PC、Inst、ALU_Result 等值在流动的过程中保持一致。

首先对于 IF 阶段，其主要任务是等待指令信号的到来；对于 ID 阶段，将指令传入控制模块，由此产生控制信号，并且读寄存器堆；对于 EX 阶段，将 ID 阶段产生的控制信号、读寄存器堆的值以及指令保存到相应的寄存器中，并且将操作数传入 ALU，根据结果判断是否跳转；对于 MEM 阶段，传递本阶段和 WB 阶段所需的控制信号，并进行访存操作；对于 WB 阶段，传递所需的控制信号，写寄存器堆。

对于 PC 的更新，如果是跳转指令或者分支指令，在 ID 阶段便可以判断是否要跳转，如果不跳转，则 PC 正常递增；如果跳转，则 PC 跳转至指定值，此时延迟槽指令已经进入流水线中。通过这种办法可以实现延迟槽，从而避免了周期浪费。

三、实验过程

（一）实验流水账

时间	记录
9 月 20 日 19：00~21：30	一、阅读课程网站上的资料及讲义，构建整个大致框架，并思考各个阶段的任务，需要哪些寄存器，并如何实现。
9 月 21 日 14：10~17：30	一、根据画的大致结构图开始写 rtl 代码 mycpu_top 部分； 二、根据讲义上的代码风格重新写控制模块。
9 月 23 日	一、进行行为仿真，根据波形，golden_trace 和 test.s 修改代码，最终行为仿真通过；

10：30~16：20	二、上板验证通过。
9 月 25 日 10：10~12：20	一、整理代码，使其整齐规范； 二、完成实验报告。

（二）错误记录

1、错误 1

（1）错误现象

PC 的值与指令的值不对应，指令比 PC 延迟一拍。

（2）分析定位过程

查看波形发现 PC 的值无误，因此错误在取指。

（3）错误原因

对于一个 PC 值，指令在这条 PC 流到 ID 的时候才会到来。因此，指令要存入 EX 时候的寄存器，而我在 ID 时便将指令存入寄存器，所以存入的是上一个 PC 对应的指令。

（4）修正效果

将指令存入 EX 阶段的寄存器，PC 和指令便能对应上。

（5）归纳总结

这个错误是没有考虑清楚时序关系造成的。

2、错误 2

（1）错误现象

仿真刚开始，控制台打 Error。

（2）分析定位过程

根据控制台打印出的指令的位置，对比 golden_trace 发现一条跳指令后第三条指令进入流水线，然而这条指令本不应该执行。

（3）错误原因

之前 PC 更新是根据 WB 阶段时候的状态进行的，因此跳转指令之后会有四条指令进入流水线。然而本实验延迟槽只允许跳转指令后的一条指令进入流水线。

（4）修正效果

使 PC 的值根据 ID 阶段产生的控制信号更新，由此若指令跳转，则只有跳转指令之后的一条指令进入延迟槽。

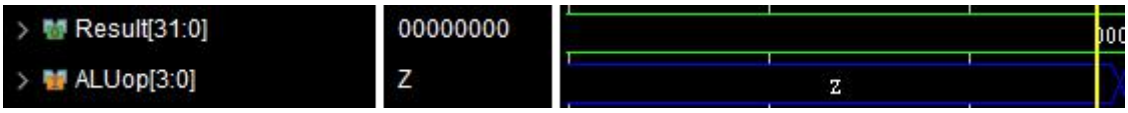
3、错误 3

（1）错误现象

行为仿真中发现 ALU 的两个输入正确，操作信号正确，但输出一直是 0，如下图：

（2）分析定位过程

查看 ALU 模块里的信号，发现 ALUop 一直是 Z，于是判断是接口处出现问题。



（3）错误原因

检查后发现 mucpu_top 中 ALUOp 位宽是 3，而 ALU 模块中位宽是 4，是因为修改控制模块时将 ALUop 的位

宽改了，导致接口处信号位宽不匹配，因此 ALUop 才会一直为 Z。

（4）修正效果

将位宽改为一致之后 ALU 正常运行。

4、错误 4

（1）错误现象

进行仿真过程中，控制台打印出的 Error。

（2）分析定位过程

根据控制台打印出的错误时间向前推，通过对比 golden_trace，发现一处 lw 指令出错，数据未存入寄存器堆中。

（3）错误原因

访存后，内存数据一个周期之后也就是写回的时才能到来，而我在 MEM 阶段就去取数据因此取出来的是错误数据。

（4）修正效果

在写回 WB 阶段取从内存中取来的数据后，lw 指令通过测试。

（5）归纳总结

这个错误是因为没有仔细考虑时序的关系，导致取数据出错。

5、错误 5

（1）错误现象

控制台不断打印错误，但测试仍继续执行，如下图：

```
run: Time (s): cpu = 00:00:10 ; elapsed = 00:00:11 . Memory (MB): peak = 921.961 ; gain = 14
run 200 us
--------------------------------------------------------------------
[ 201695 ns] Error(  1)!!! Occurred in number 8'd00 Functional Test Point!
--------------------------------------------------------------------
        [ 202000 ns] Test is running, debug_wb_pc = 0xbfc006b8
        [ 212000 ns] Test is running, debug_wb_pc = 0xbfc36ab0
        [ 222000 ns] Test is running, debug_wb_pc = 0xbfc37280
        [ 232000 ns] Test is running, debug_wb_pc = 0xbfc37a50
        [ 242000 ns] Test is running, debug_wb_pc = 0xbfc38220
        [ 252000 ns] Test is running, debug_wb_pc = 0xbfc389f0
        [ 262000 ns] Test is running, debug_wb_pc = 0xbfc391c0
        [ 272000 ns] Test is running, debug_wb_pc = 0xbfc39990
        [ 282000 ns] Test is running, debug_wb_pc = 0xbfc3a160
        [ 292000 ns] Test is running, debug_wb_pc = 0xbfc3a930
        [ 302000 ns] Test is running, debug_wb_pc = 0xbfc3b100
        [ 312000 ns] Test is running, debug_wb_pc = 0xbfc3b8d0
        [ 322000 ns] Test is running, debug_wb_pc = 0xbfc3c0a0
--------------------------------------------------------------------
[ 323435 ns] Error(  2)!!! Unknown, Functional Test Point numbers are unequal!
--------------------------------------------------------------------
====================================================================
```

（2）分析定位过程

piazza 上一位同学的问题与我的相似，根据助教老师个同学的回答判断是一出 store 指令出错。于是在打印出 Error 的时间往前推，发先是溢出 store 指令出错。

（3）错误原因

发现存入内存的数据为 1，而非正确的值。于是检查代码发现 data_sram_wdata_MEM 信号未定义位宽，所以存入内存的数据才是 1。

（4）修正效果

将 data_sram_wdata_MEM 信号的位宽定义为 32 之后行为仿真通过。

（5）归纳总结

在对信号进行定义时，要注意信号的位宽。

四、实验总结

实验二的第一阶段因为不需要考虑数据相关，因此实验难度不大。主要难点还在于对于流水线框架的构思。如果思路清晰，框架结构合理本实验很快便能完成，并且后续的实验也会很轻松。本实验主要还是为之后的实验打基础。

另外修改了控制模块的代码风格，更改过之后发现代码清晰，添加指令和 debug 都很方便。