RISC-V 流水线 CPU Lab2 实验报告

PB16060130　　顾健鑫

一、实验目的

本次实验根据实验一设计好的 CPU 数据通路，用 Verilog 实现进行实现，并以通过 testbench 的 3 个测试文件为本次实验的最终目标。

二、实验环境和工具

本次实验由于开发板的硬件适配原因，使用的软件环境为 Vivado 2017.4 的版本。在创建工程项目时，直接选择导入提供好的模块框架文件并在完成代码的编写后，导入测试文件用于测试。

三、实验内容和过程

本次实验在实现的时候，由于已经设计好了 Hazard Unit 的 Flush 和 Stall 操作逻辑，因此我在实际做时直接从第三阶段开始。另外由于在做 Lab1 时已经整理出来了每条指令在 ControlUnit 应有的输出，因此 CPU 的核心模块 ControlUnit 的工作量有显著降低，主要以将整理的表格转化为 Verilog 代码即可。另外的其余模块则是按照 Lab01 设计报告的描述进行实现。

在完成代码编写后我先进行了多次综合，查看所有的 warning 并进可能的多消除。最后仅留下由于逻辑上的不使用导致的综合阶段的简化，如不存在 stall WB 段的操作所导致 WB 段寄存器中的 stall_ff 以及相关的用来暂存上周期数据的 32 位寄存器的移除等。

调试的时候我则是以 testbench 文件为准直接进行调试，并通过追踪 3 号寄存器的值的变化来确定具体在几号测试中发生错误并跳转至“fail”，之后再通过阅读对应测试号的具体汇编代码，添加别的信号用于具体的错误侦测和代码修改。具体的调试经过将在第四部分中详述。

四、实验总结

本次实验感慨颇多。

自己踩的深坑：过于相信 testbench 输出的指令存储器文件，并且有点一根筋，后来在偶然间点开了仿真界面的指令存储器才发现每个 testbench 文件在输出时都会把第一条指令多输出一次，导致我参考的汇编代码地址和实际代码地址错了 4 个字节，也就是前后一个周期。这直接导致我浪费了几乎两天的时间在这个问题上。最开始发现这个问题是因为发现 PCD 和指令不同步，即 PCD 为上一个周期的指令的 PC。（其实是没有问题的）以至于我对 Memory 的读写特性都产生了怀疑，为此还特意去修改了 testbench.v 文件去测试顺序读取时两个 Memory 的读取结果，结果自然也是本身没有问题的，但我仍然是以输出的 txt 为参考标准，因此最后得出的结论是两个 Memory 是异步读的。在此基础上，我去修改了 ID 段寄存器的输出方式，并对 HazardUnit 的 stall 和 flush 逻辑进行了修改（把对的改成了错的）。在中在完全反常理的仿真结果下，同一个地址前后两个周期读出的结果不同，让我在此产生怀疑并去查看了仿真界面下的 Memory 具体值，这才发现自己干了两天的

无用功，发现了一堆本不是问题的问题。至此跳出深坑开始进入正式的调试阶段。

在排除了 Memory 的疑虑后调试整体相对顺利，在第一个 testbench 文件中耗时相对较长些，主要的问题在于设计报告中所犯的 HazardUnit 的设计错误。在设计报告中，我对分支指令执行时的处理结果描述如下。

对于分支指令的处理设计如下

1. 当 BranchE 或 JalrE 为 1 时，即分支成功，由于统一采用预测失败的策略，因此要 flush 流水线的 ID 和 IF 段。
2. 当 JalD 为 1 且 BranchE 和 JalrE 均为 0 时，根据优先级判断，JalD 分支成功，此时仅 flush 流水线的 IF 段即可。

但这是有问题的，由于这个周期置 1，下个周期才会对段寄存器置 0，因此实际上 IF 段并不需要 Flush，其本身就直接接收了分支指令的地址，继续运行即可。相比较下，应该整体再右移一个流水段，即 Branch 或 Jalr 指令 Flush 流水线的 ID 和 EX 段，Jal 指令 Flush 流水线的 ID 段，即可达到目标。这个问题解决后 testbench1 就通过了。

另外两个问题分别出现在 testbench2 和 testbench3 中，每个文件各一处。分别是 1、Data Memory 的在写操作，仅仅对 WE 做了特殊的移位处理，但并没有对输入的数据做移位处理。导致 373 号测试失败（先 sb 一个 16 位数据到字对齐地址+2 的位置，再从相同的地址 lh 出数据放到指定寄存器中）。在这里 sb 对应的 WE 应为 4’b0100，而这种情形下，直接传输过来的数据 StoreDataM 是不符合要求的，为了正确的存储，还需要将其左移 WE*8 位以保证结果的正确。2、由于手抖使得 ControlUnit 中 SRA 指令的 case 判断条件多打了一个 0，这导致 613 号测试中的 SRA 测试中 ControllUnit 输出全为 0（default 值）。

整体总结，由于自己跳进了深坑并且一根筋的参考一个来源，导致做了很多的无用功，以后在调试时应该参考多方信息再做工作。尤其是在遇到反直觉的情况时，如 memory 模块的代码确实是同步读但无论怎么测都是异步读的情况。

五、改进意见

本次实验由于已经有了助教提供好的框架和设计图，整体实现流程相比去年的 COD 的 MIPS 流水线要少考虑很多东西。在有详细设计报告的前提下，实现起来难度降低了很多。另外还是希望能在提供测试文件时多一些说明，比如输出的 memory 具体值的 txt 中会将首地址多输出一次这样的情况。

仿真结果展示

分别为 testbench1、2、3 的仿真波形图，可以看到最后 3 号寄存器结果变为 1，并且 PC_IN、PCF 和 PCD 已经开始循环，其中 PCD 对应的指令地址加上 x10080 后即为给出的汇编代码中的 success 的指令地址。