RISC-V CPU 设计报告

作者 ： 肖云轩

516030910268

1 设计架构

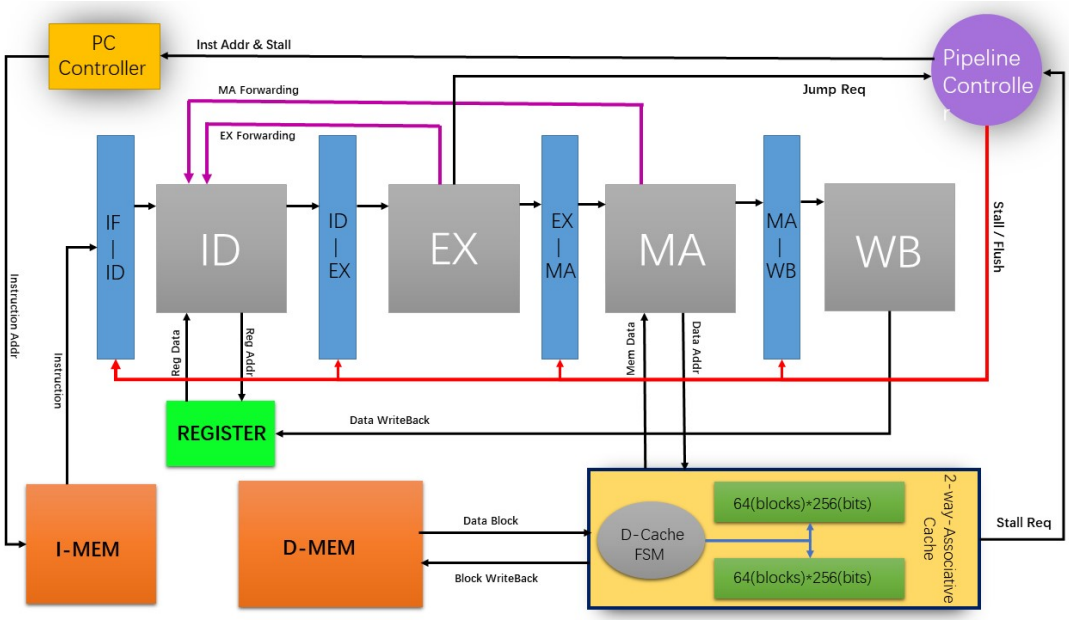

图 1: 流水线架构

在实现了传统的 MIPS 五级流水线的基础上，进行了如下改进:

1. 增加了 EX、MA 两条 Forwarding 线，WB 阶段 Forwarding 由一个 Cycle 内寄存器先写后读实现，解决了除 Load/Store 之外的 RAW Harzard 问题。

2. 内存采用哈佛结构，指令内存与数据内存分开，解决了数据与指令同时访存的 Structual Hazard

3. 实现了一个 data cache 模块，内部由状态机控制时序，采用块大小为 2MB(64*256bit) 的两路组关联 cache，LRU 替换策略

4. 增加了流水线控制单元，监听 cache 模块 stall 信号以及 branch 指令跳转信号，控制流水线暂停、清空、以及 PC 取指等操作。

2 D-cache 设计

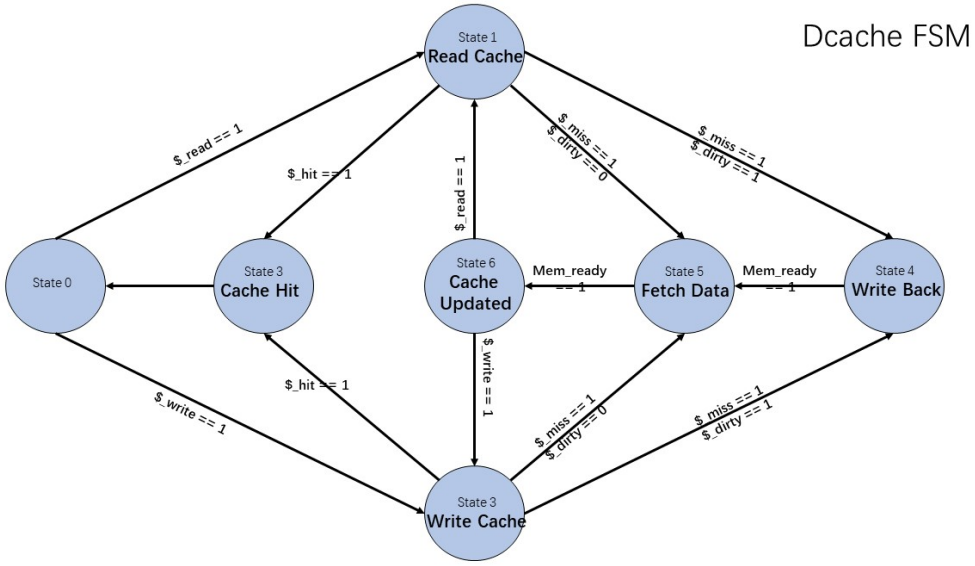

图 2: cache 状态

2.1 功能描述

此设计为一级两路组关联 Cache，因此其命中率较高，大概有 80% 左右的命中率，剩余 20% 情况均摊到 miss 或者 miss 且 dirty 的情况。而我们使用的是 core 等待 cache 响应的机制，因此要提高 cpu 的速率，应考虑尽量缩短 core 对于 cache 进行访问的时间。

为了便于控制 cache 与 core 和 memory 的交互，以及控制时序逻辑的需要，dcache 使用了如图所示的状态机控制内部功能模块。

替换方式为 LRU，块大小为 256bits，保证了时间本地性与空间本地性。

2.2 未来可能的改进：

1.ID 阶段传输指令 Load/Store 地址，cache 进行数据预取减少停顿。

2. 令状态机转换成为流水，搭配改进 1 大幅度增加 cache 的吞吐率。

3. 增加 victim cache。

3 思考与致谢

本次 CPU 设计大作业中，通过动手实践与模拟初步实现了五级流水，对流水的效率问题有了更深的思考。为了使流水线效率提升，要保证各流水级时间均衡，减少跳转延迟等，其中最关键的问题是内存访问延迟，其花销远大于跳转失败所花费的 cycle，故增加了 cache 模块。在编写过程中对 cache 与内存交互机制感触颇深。

然而，在编写过程中也经历了很多痛苦的 bug，最棘手的便是维护时序以及 always 块互相触发。比如三个 always 块敏感信号互相触发，而每个 always 块中的信号数目特别多，使得 debug 进程花费了很长时间，导致之后没有能够上板子，我感到非常遗憾。

最后感谢两位助教的指导与帮助，感谢范舟与吴章昊同学的测试代码，以及所有向我提供过帮助的同学！曾记得我们七八个人在党团活动室肝代码到凌晨，没有与你们分享经验与交流这个大作业会变得更加艰难，也希望我们能记住这段艰难而又美好的回忆！