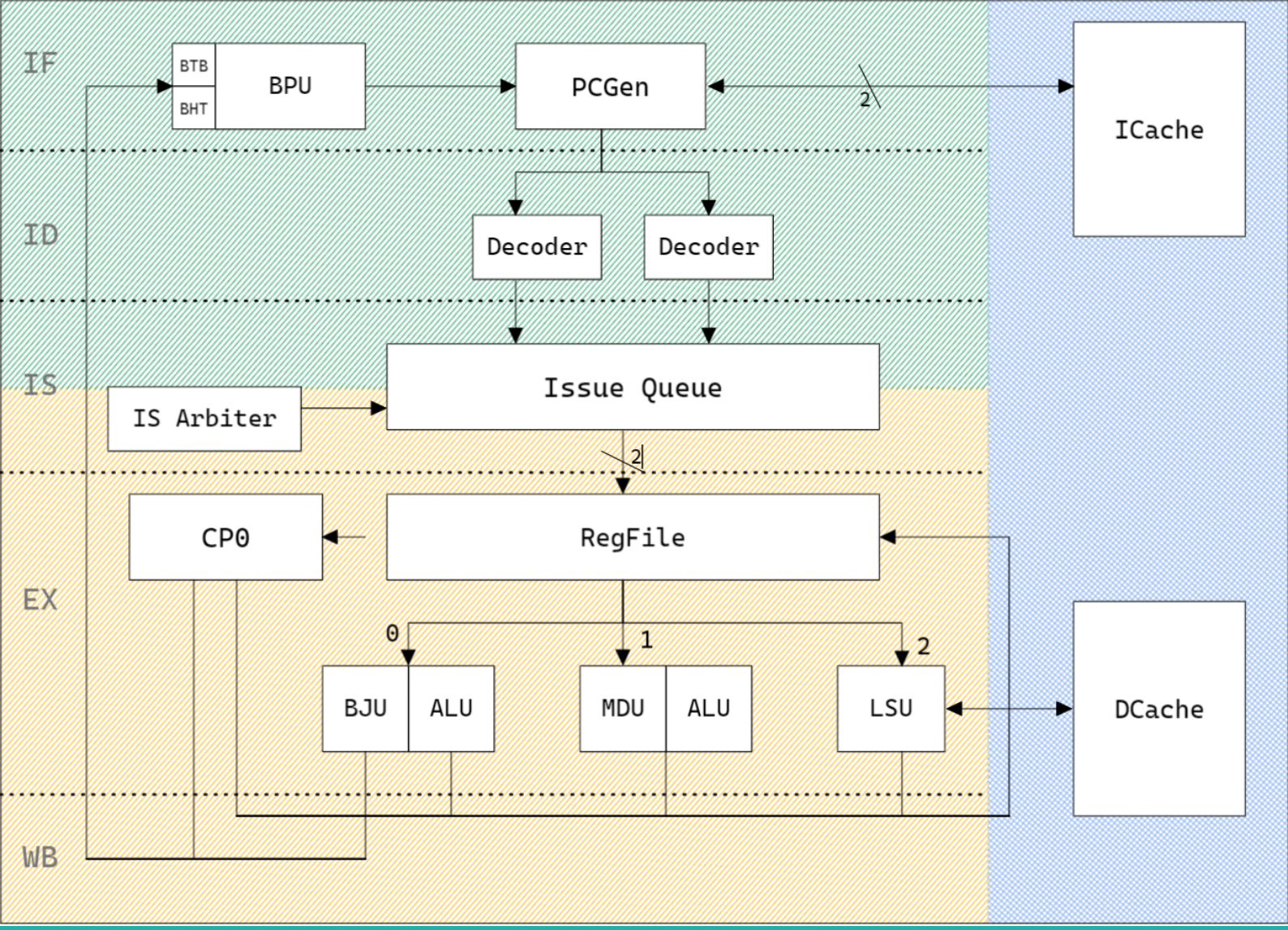

Marvelous MIPS Machine (MMM) 设计报告

第六届 “龙芯杯” 全国大学生计算机系统能力培养大赛

作者：潘子曰、王晶晶、汤尧、叶泽凯

组织：浙江大学 | Capable Chip Creator (CCC) 队

时间：August 5, 2022

指导老师：常瑞

目录

第 1 章 MMM 处理器简介

MMM (Marvelous MIPS Machine) 是一款基于 Chisel 语言开发的 MIPS32 架构处理器。MMM 处理器在设计上采用 KISS (Keep It Simple, Stupid) 原则，架构简单，可配置强。

MMM 处理器的主要设计亮点在于：

1. **精简的核内设计**：MMM 处理器采用顺序双发射五级流水线，结构精简紧凑。一方面，短流水级可以避免流水级过长造成的巨大 stall 开销，另一方面，精简的设计也让处理器保持了良好的时序，在龙芯实验箱上的频率可达 110MHz。
2. **软件辅助的处理器验证**：MMM 在设计过程中充分使用软件辅助处理器设计。在正确性方面分别采用了重庆大学陈泱宇同学开发的 `soc-simulator` [2] 和自主开发的基于 QEMU 的差分测试框架，在性能方面设计了基于 Verilator 的性能分析 (Profiling) 部件，用于量化分析处理器在性能测试时的执行状态和瓶颈。利用快速的软件验证，MMM 处理器的设计过程中始终保持明确的方向，节省了时间开销。
3. **优异的性能表现**：MMM 处理器在设计时着重优化了运行过程中最为严重的性能瓶颈：分支预测。MMM 处理器采用了基于 BHT+BTB 的经典分支预测器，在性能测试中平均分支预测准确率可达 88.24%，使得性能测试达到了 87.830 分。

第 2 章 MMM 处理器的设计与实现

2.1 概览

MMM 总体结构分为处理器核心、数据缓存、指令缓存、总线仲裁转换器。其中处理器内部分为前后端，前端包含两个阶段，分别为 IF 和 ID，后端包含三个阶段，分别为 IS, EX 和 WB，共五级流水线。

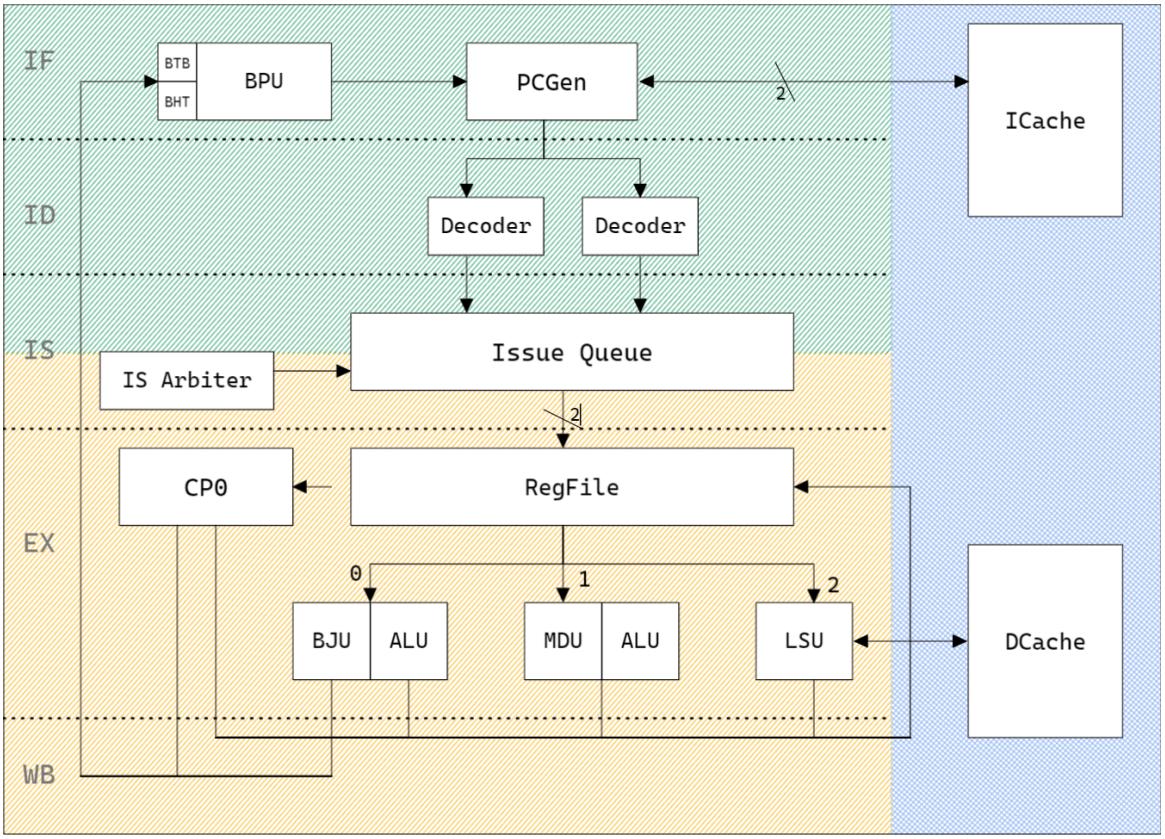

图 2.1: MMM 处理器的总体设计

如图 2.4所示，前端采取双发射的模式，从指令缓存中取出指令后，经由解码器解析送入发射队列。后端为双发射，发射仲裁器会在 IS 阶段判断当前发射队列队首的两条指令是否符合并行条件，具体取决于我们在 EX 阶段的设计：共分为三条通路，一条通路处理基本算数运算和跳转判断，一条通路处理乘除法和基本算数运算，第三条负责处理数据的访存，三条通路同时处理。在 WB 阶段，执行 CP0 和 RegFile 相关的寄存器写回操作。

为提高运行效率，在遇到分支跳转指令时，MMM 会使用分支预测器判断是否进行跳转，以减少判断错误造成的性能损失。

2.2 QMC 算法优化的译码器设计

QMC (Quine–McCluskey) 算法在功能上等同于卡诺图，用于最小化布尔函数。MMM 处理器使用了 Chisel 库中的 QMC 算法实现 [3]，用于最小化译码器 (Decoder) 的布尔逻辑。

经过实验验证，相比原来手工设计的译码器，QMC 算法优化的译码器组件获得了约 10% 的时序提升。

2.3 分支预测

分支预测单元 (**B**ranch **P**rediction **U**nit) 主要由两部分构成，BHT(分支历史表) 和 BTB(分支跳转目标缓冲)。二者均由双端口 BRAM 存储。其中，BHT 负责存储一组两位的向量，记录某个地址映射的跳转预测情况，BTB 负责存储某个地址映射的历史跳转目标地址。BHT 的两位预测向量更新机制如下图所示。主要有来自于两个阶段的更新。一为在 ID 阶段发现指令为普通指令但预测跳转，而为 WB 阶段提交的指令分支预测错误。

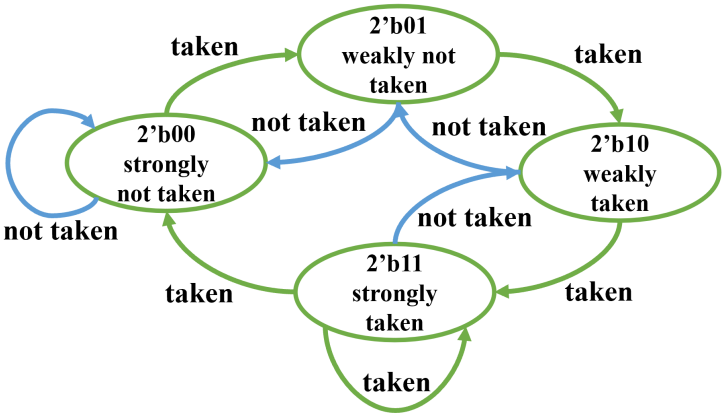

图 2.2: BHT 预测向量

BTB 的更新唯一来自于 WB 提交指令阶段。当跳转或分支指令的执行目标地址和预测的目标地址发生冲突时，就会更新 BTB 中存储的分支跳转目标历史。

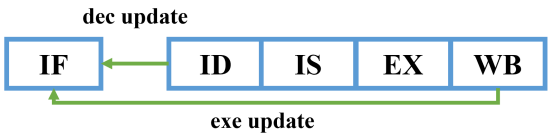

图 2.3: BHT 更新机制

为了解决顺序双发过程中 BHT 存在的双读单写问题，我们采取了两种策略进行试验。一为 split BRAM 机制，二为 BHT cache 机制，来进行 forwarding 操作。split BRAM 可以实现较高的预测准确率，BHT cache 的预测准确率也高于不采用 cache。其中 BHT cache 由四个寄存器组成的循环队列构成，记录最新的 BHT 更新结果。但二者在时序上都劣于最简单的不进行 forwarding 的机制。

在这样的 trade-off 下，我们选择简化时序。只使用最简单的一个双端口 BRAM 构建 BHT。一旦遇到写请求，就放弃对于第二条指令的读取、在这种 greedy 策略下，也能表现出可观的预测准确率。

2.4 缓存

缓存结构包括 DCache 和 ICache 。DCache : 直连，16KB，写分配，行位宽 256 bits，使用双端口 BRAM 实现。DCache : 直连，16KB，行位宽 256 bits，使用双端口 BRAM 实现。输入 Cache 的地址为

虚地址，在内部映射并储存为物理地址。

对于写操作，由于行位数大于字长，因此在执行写入操作时需要将整行读出，改写对应字（半字/字节）的数据，再将整行写回。为提高流水线效率，在 Cache 中分为三个流水段，第一个周期将传入的地址映射后输入 BRAM，在第二个周期返回命中结果和有效数据，并在数据有效时发送 valid 信号，若指令为 DCache 的写请求，则在第三个周期将整行数据写回。为了使写操作流水化，硬件资源至少要能同时一次读和一次写，选择使用双端口 BRAM 实现。

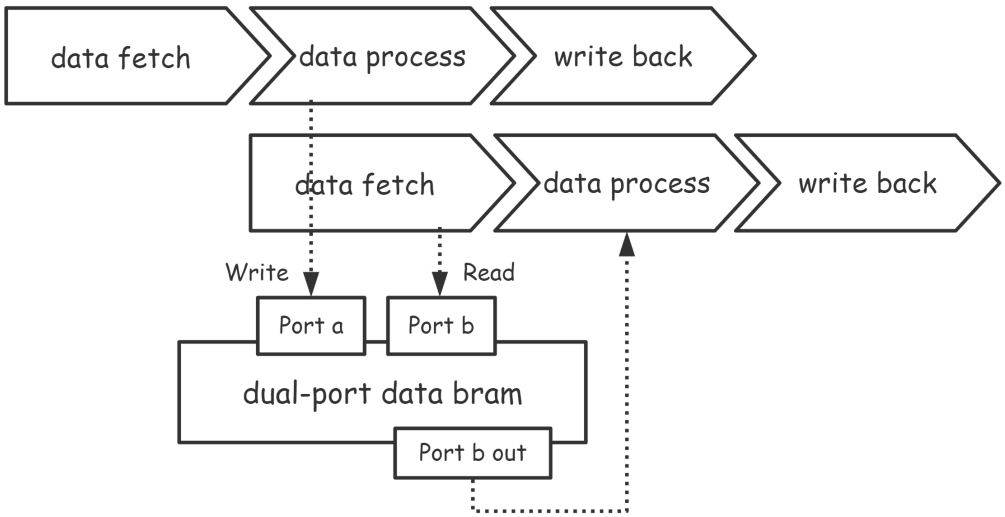

图 2.4: Cache 的流水线时序实现

Cache 的三个阶段都有 valid 信号相互制约，确保了数据流通的有效性和正确性。对于流水线中可能出现的数据冲突，如上一周期写入地址和下一周期访问地址相同的冲突，则将上一条请求写入数据直接定向到返回数据。

第 3 章 软件验证框架

3.1 soc-simulator

soc-simulator 是重庆大学陈泱宇同学设计的基于 Verilator 的仿真框架 [2]，模拟了 AXI 协议和龙芯提供的 confreg，并且支持 golden_trace 做差分测试。相比原有的仿真测试框架，soc-simulator 可以轻松地修改基于 C++ 的 testbench，对多发射结构的处理器仿真更有友好。此外，Verilator 的运行速度也比 XSim 更快，大幅提高了 MMM 处理器的仿真效率。

3.2 基于 QEMU 的差分测试框架

MMM 处理器还使用了自主设计的基于 QEMU 的差分测试框架，用于系统软件的仿真验证。MMM 处理器通过 QEMU 提供的 remote serial protocol 协议控制 QEMU 的运行，并且通过中断屏蔽和启用同步控制流，提高了差分测试的效率。

3.3 性能分析 (Profiling) 部件

MMM 处理器添加了多个性能分析 (Profiling) 部件，在仿真时可开启用于快速分析处理器运行时的状态和瓶颈。

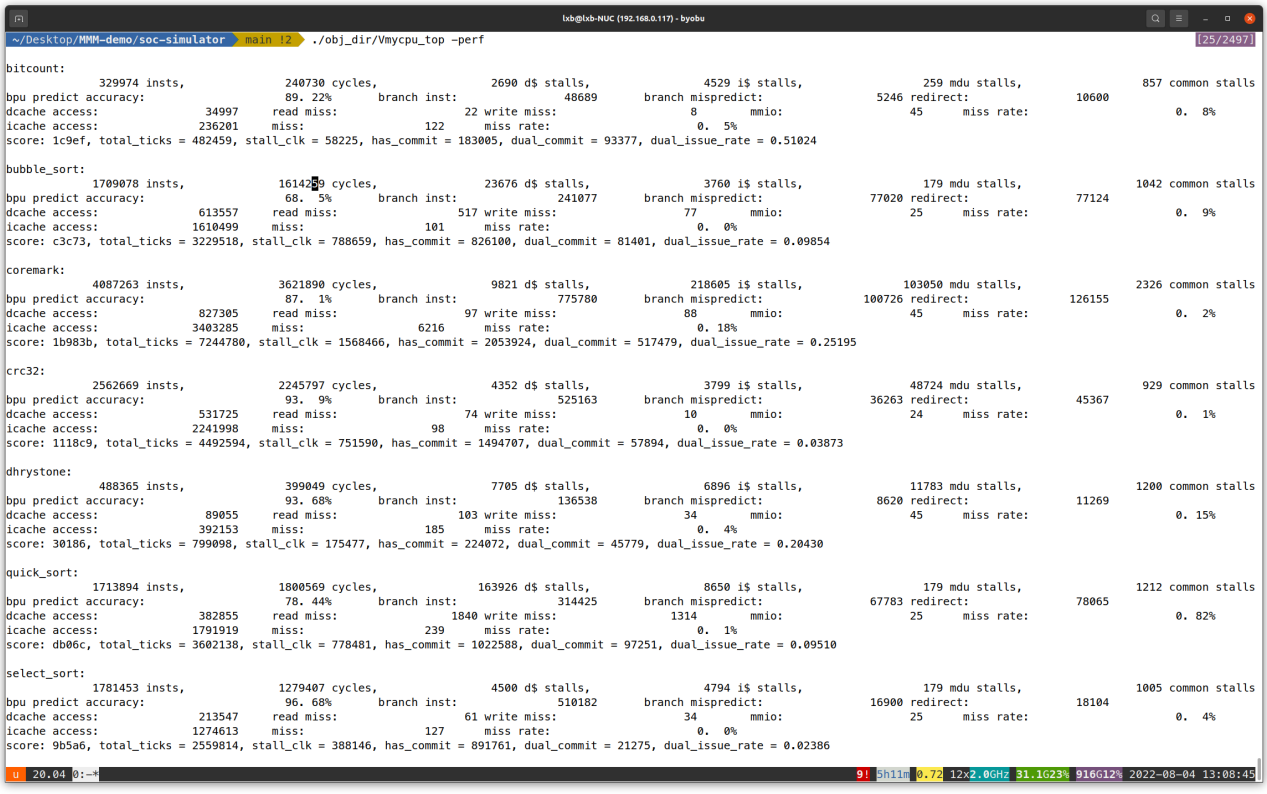

图 3.1: 性能分析结果

如图 3.1所示，MMM 处理器在仿真时统计了总共执行的指令数、分支跳转的指令数、周期、缓存的 Stall、缓存的访问次数、缓存的失中率、乘除法器的 Stall、分支预测器的准确率、失败的分支预测数、总共的跳转次数以及其他多个时钟统计指标。

第 4 章 总结与展望

MMM 处理器是一款小巧精悍的 MIPS 架构处理器，在本次龙芯杯系统能力初赛的性能测试中取得了 87.830 的成绩。

在未来几天内，我们将努力扩展 MMM 处理器使其适配 PMON、uCore 和 Linux 处理器，并且将尝试增加自定义的安全指令，以展现 MMM 处理器的扩展性。

参考文献

[1] *chipsalliance/rocket-chip: Rocket Chip Generator*. URL: https://github.com/chipsalliance/rocket-chip.

[2] *cyyself/soc-simulator: A Verilator based SoC simulator that allows you to define AXI Slave interface in software*. URL: https://github.com/cyyself/soc-simulator.

[3] *QMCMinimizer*. URL: https://www.chisel-lang.org/api/latest/chisel3/util/experimental/decode/QMCMinimizer$.