ZenCove 项目分享

“龙芯杯”第六届全国大学生系统能力培养大赛

崔轶锴　张为　王拓为
清华大学计算机系

内容大纲


CPU 架构设计
清华大学 Tsinghua University
龙芯杯项目展示——ZenCove
3


微架构优化
清华大学 Tsinghua University
龙芯杯项目展示——ZenCove
10


SoC 架构设计
清华大学 Tsinghua University
龙芯杯项目展示——ZenCove
14


系统软件
清华大学 Tsinghua University
龙芯杯项目展示——ZenCove
16


总结致谢
清华大学 Tsinghua University
龙芯杯项目展示——ZenCove
20

CPU 架构设计

设计模式

真的高级
真的优雅

非传统硬件描述语言 SpinalHDL

- 基于高级语言 Scala 构建
- 具有更强大的表达能力、抽象能力和面向对象能力

设计模式

彻底解耦
即插即用

非传统硬件描述语言 SpinalHDL

- 基于高级语言 Scala 构建
- 具有更强大的表达能力、抽象能力和面向对象能力

借鉴 VexRiscv 的设计理念

- Plugin 概念 → 功能模块真正分离
- 自动流水线工具 → Plugin 即插即用
- Service 系统 → Plugin 动态交互
- 高度可配置 → 所有主要功能参数化支持

微架构

乱序乱序
乱中有序

乱序多发射

- 标准配置下为 4 发射
- 前端 5 级流水：IF1, IF2, ID, RENAME, DISPATCH
- 后端 4 条流水：INT × 2, MUL/DIV × 1, MEMORY × 1
- 支持指令缓存和数据缓存：8 KB, 2 路组相联

微架构

小孩子才做选择题
我全都要

乱序多发射

- 标准配置下为 4 发射
- 前端 5 级流水：IF1, IF2, ID, RENAME, DISPATCH
- 后端 4 条流水：INT × 2, MUL/DIV × 1, MEMORY × 1
- 支持指令缓存和数据缓存：8 KB, 2 路组相联

基于 MIPS32 Release 1 指令集

- 完整支持单核运行 Linux 所需 92 条指令
- 支持 23 个 CP0 寄存器
- 支持 12 种异常类型
- 支持 TLB, CACHE 等特权指令

乱序的挑战

欲戴皇冠
必承其重

挑战之一：设计逻辑复杂

- 远较顺序处理器复杂的设计逻辑和冲突问题
- 如何解决乱序产生的新的数据相关？
- 如何处理部分访存指令的强顺序性？
- 如何突破复杂逻辑导致的较大门延迟对主频的限制？
- 没有统一的设计准则，需要根据实际需求 Trade off

挑战之二：实现调试困难

- 以处理器的状态恢复为例
- 乱序处理器流水线深度大，十分依赖预测技术
- 预测失败，错误指令需要抹去，造成痕迹也要消除
- 预测技术越激进，恢复电路越复杂，实现调试越困难

因为乱

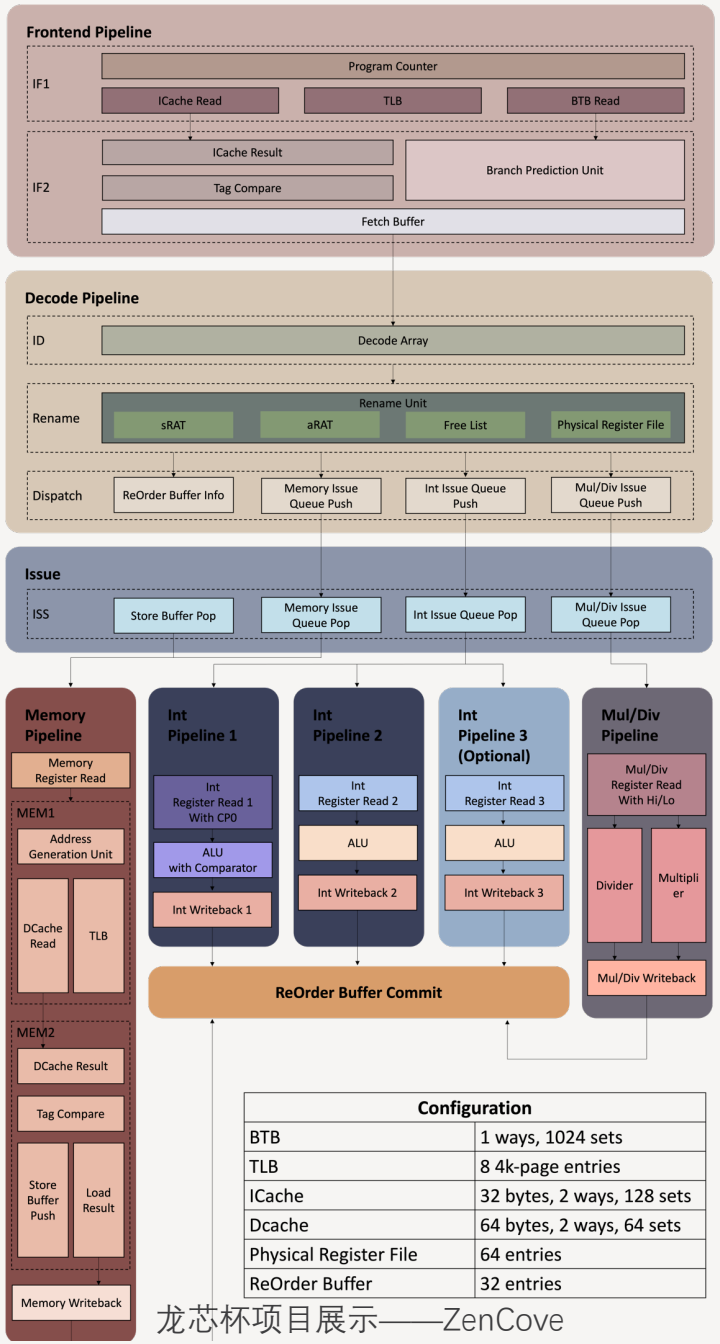
顺序处理器 NonTrivialMIPS 性能

NSCSCC 2019 特等奖

性能得分纪录保持者

处理器主频 123 M			
测试程序	加速比	IPC	IPC 比值
bitcount	97.886	1.155	39.308
bubble_sort	85.152	0.904	34.600
coremark	74.182	0.828	30.139
crc32	93.048	0.975	37.805
dhrystone	86.365	0.498	35.107
quick_sort	75.984	0.898	30.873
select_sort	101.278	1.258	41.153
sha	102.928	1.143	41.823
stream_copy	87.286	0.860	35.544
stringsearch	94.136	0.590	38.253
几何平均	89.325	0.879	36.361

ZenCove Architecture



Configuration	
BTB	1 ways, 1024 sets
TLB	8 4k-page entries
ICache	32 bytes, 2 ways, 128 sets
Dcache	64 bytes, 2 ways, 64 sets
Physical Register File	64 entries
ReOrder Buffer	32 entries

所以快

乱序处理器 ZenCove 性能

频率↓9.8%，加速比↑2.4%，IPC↑13.5%

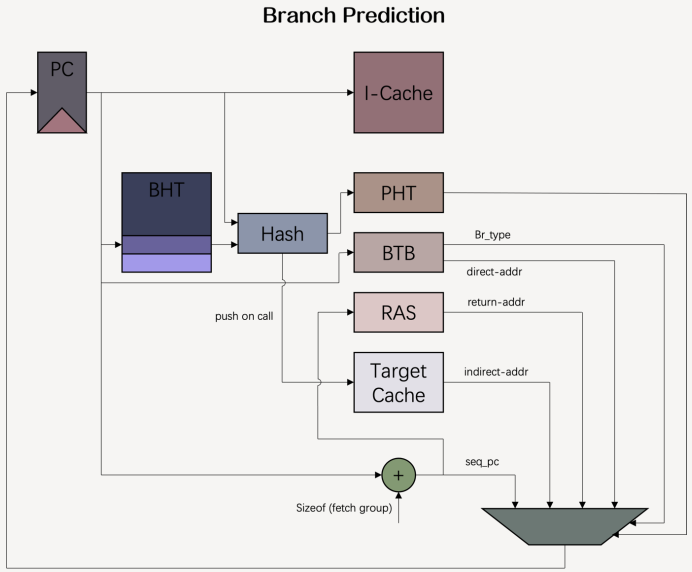
7 个程序 IPC > 1，平均 IPC ≈ 1

处理器主频 111 M			
测试程序	加速比	IPC	IPC 比值
bitcount	85.865	1.122	38.185
bubble_sort	102.764	1.209	46.254
coremark	65.926	0.815	29.677
crc32	111.846	1.298	50.336
dhrystone	74.887	0.479	34.503
quick_sort	76.803	1.006	34.566
select_sort	93.746	1.291	42.195
sha	112.323	1.382	50.556
stream_copy	105.482	1.151	47.624
stringsearch	98.915	0.687	44.532
几何平均	91.495	0.997	41.242

微架构优化

分支预测器

- 分支目标缓冲区（BTB）
- 分支方向预测器（PHT）
- 调用返回预测器（RAS）



反复实验寻找最佳方案

- 实验 4 种不同方案：2 位饱和计数器局部预测器、Gshare、Gselect 全局预测器、局部/全局混合预测器
- 最终选用 Gselect 全局预测器

处理器主频 100 M	
架构	加速比
朴素实现	32.065
分支预测（无 RAS）	79.481
分支预测	82.874

提升 158%
提升 4.3%

分支预测

过去是最好的预言

基于实际痛点

- LOAD 指令和后续指令的 RAW 相关十分常见
- LOAD 指令执行周期不确定，后续指令唤醒不及时

设计优化方案

- 实际执行中 Cache 缺失为小概率事件
- 假设 Cache 总是命中，提前唤醒后续指令
- 如果 Cache 缺失，暂停流水线

处理器主频 100 M	
架构	加速比
朴素实现	79.183
推测唤醒	82.874

提升 4.7%

推测唤醒

我愿意赌
因为我知道不会输

基于实际痛点

- 实际程序执行中 RAW 相关十分常见
- 乱序处理器中等待指令写回时间过长

设计优化方案

- 指令执行阶段开始时需要操作数，结束时得到操作数
- 在功能单元的输入和输出之间构建通路
- 实现背靠背执行相关指令

处理器主频 100 M	
架构	加速比
朴素实现	79.391
数据前传	82.874

提升 4.4%

数据前传

车到山前必有路
船到桥头自然直

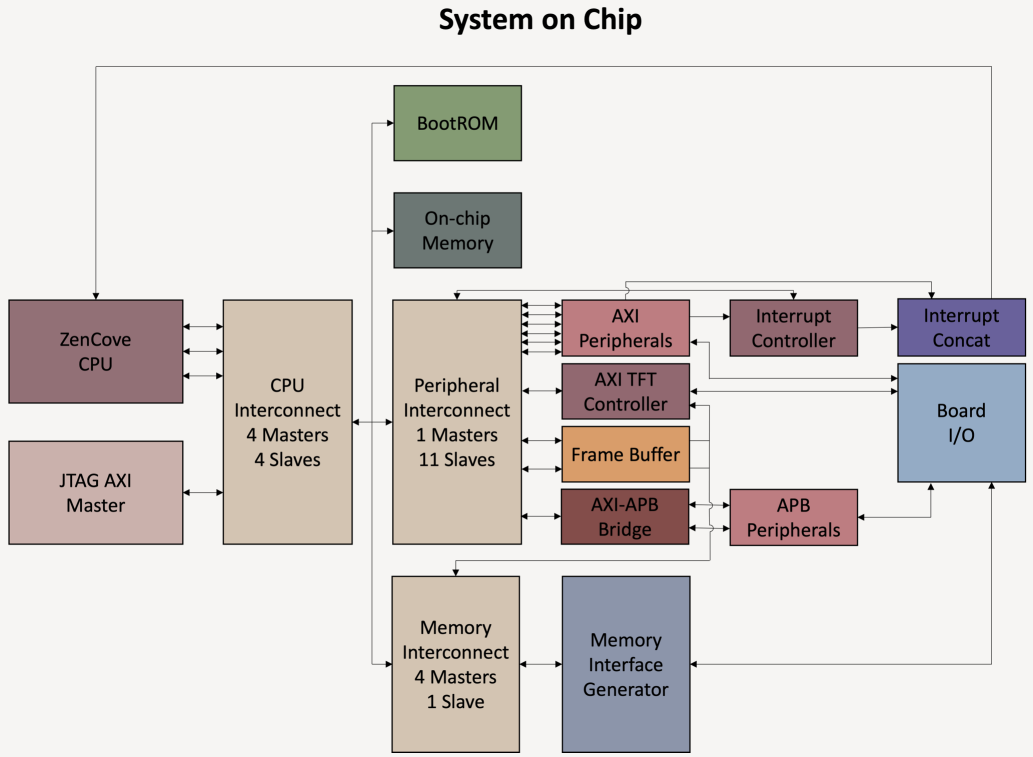
SoC 架构设计

外设驱动

人尽其才
物尽其用

使用除 USB 外全部硬件接口
支持外设如下：

- DRAM
- 串口
- FLASH
- 以太网
- GPIO
- PS2
- VGA Framebuffer
- NT35510 LCD



系统软件

第 1 级引导程序 TrivialBootloader*

- 完成基本的内存检查和初始化
- 从 FLASH 拷贝 ELF 格式文件到内存并引导
- 启动 U-Boot 和 uCore 操作系统

第 2 级引导程序 U-Boot

- 适配最新稳定版本 v2022.04
- 从网口通过 tftp 协议加载 Linux 操作系统到内存并引导
- 包含丰富外设，具备基本交互功能

* NonTrivialMIPS 项目 https://github.com/trivialmips/TrivialMIPS_Software

Mission I
引导程序

师父领进门
修行靠个人

uCore-mips 操作系统

- 拥有完整 Kernel 功能
- 自带少量用户态程序
- 通过串口进行简单交互

Mission II
uCore

瘦死的骆驼比马大

Linux 操作系统

- 适配最新发布版本 v5.19-rc4
- 移植 VGA 驱动以支持 Framebuffer 和 DMA
- 移植并修复 LCD 驱动以适配 LCD 屏
- 使用 Buildroot 完成用户程序构建
- 文件系统中配置 micropython 和小游戏 ascii_invaders

龙芯杯历史上首次运行完整 Linux 的乱序处理器

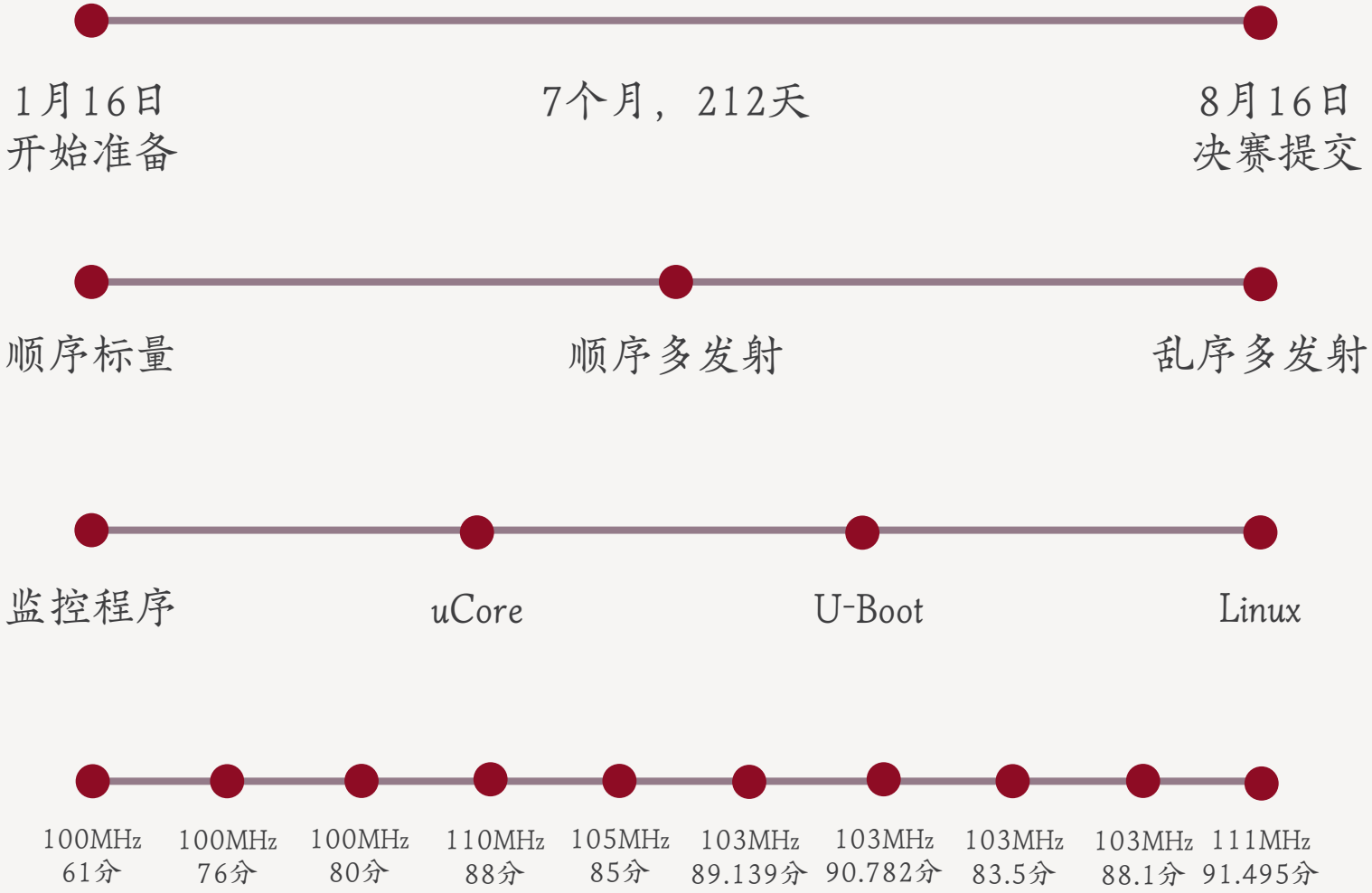
Mission III
Linux

你们对力量一无所知

总结致谢

总结自己

凡是过往
皆为序章



致谢朋友

站在巨人的肩膀上

感谢刘卫东、陈康、李山山、陆游游等老师的指导

感谢陈嘉杰、高一川、陈晟祺、黄嘉良等学长的帮助

感谢隔壁队伍同学的陪伴

核心亮点

- 突破边界：龙芯杯历史上首次完整运行 Linux 的乱序处理器
- 极致性能：在微架构上深度优化，实现 111 MHz，91.495 分
- 外设丰富：成功驱动了实验平台上除 USB 外的所有硬件外设

谢谢大家
欢迎提问

附1：

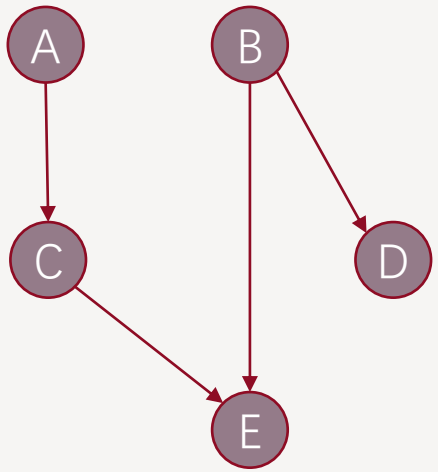
乱序的原因

Why did you want to climb Mount Everest?
Because it is there.

为什么选择乱序

- 贴近实际：现代处理器几乎全部为乱序超标量
- 效率更高：避免了大量顺序执行导致的不必要等待
- 突破边界：龙芯杯历史上没有真正完善的乱序处理器

```
指令A   add    r3,   r2,   r1
指令B   mul    r5,   r4,   r2
指令C   mul    r6,   r3,   r1
指令D   add    r10,  r5,   r4
指令E   add    r7,   r5,   r6
```



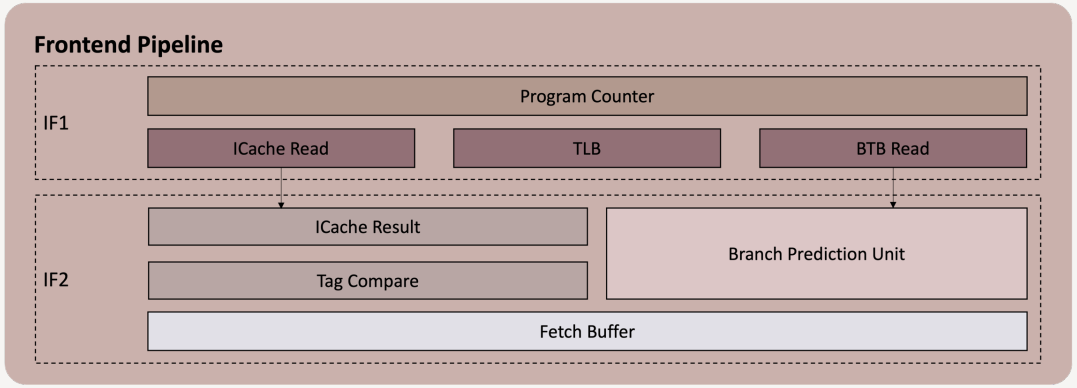
附2：微架构——取指流水线

取指令1 IF1

- TLB 查询
- 地址翻译
- I-Cache 查询第 1 阶段

取指令2 IF2

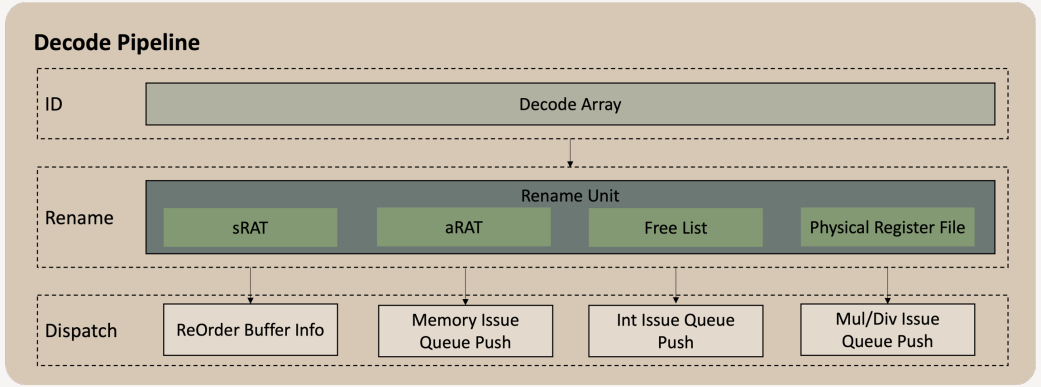
- I-Cache 查询第 2 阶段
- 根据缓存缺失情况进行填充



附2：微架构——译码流水线

指令译码 ID

- 每条指令只产生一个微码



寄存器重命名 RENAME

- 基于统一的物理寄存器
- 2 个重命名映射表以支持精确异常
- 基于 FIFO 管理空闲寄存器列表

指令派发 DISPATCH

- 根据指令类型将指令装入对应发射队列
- 装入 ROB

附2：微架构——后端流水线

发射

- 分布式发射队列，共同选择避免重复

读寄存器

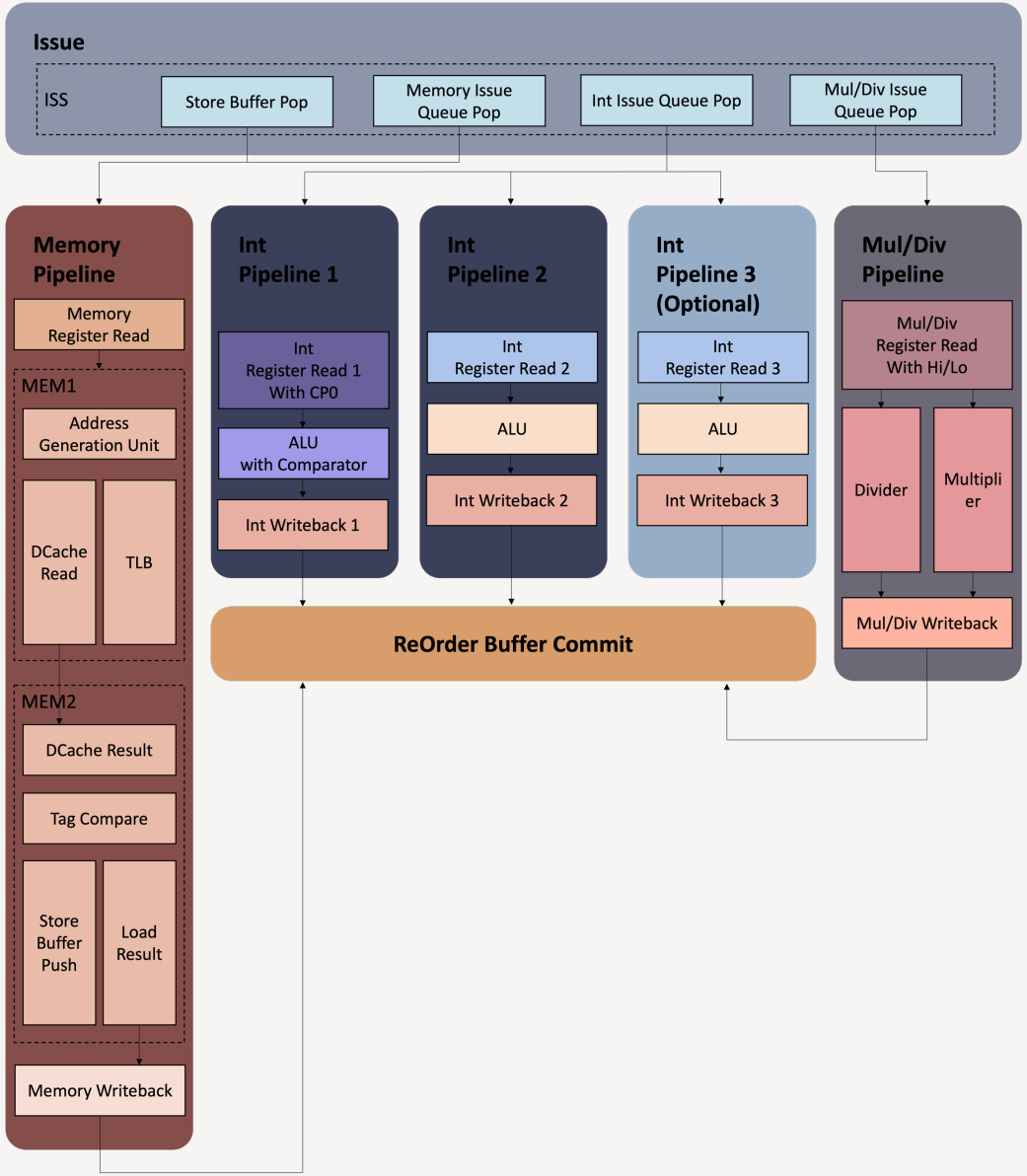
- 读取物理寄存器堆
- 整数流水线 进行 RAW 相关唤醒

执行

- 计算指令执行结果

写回

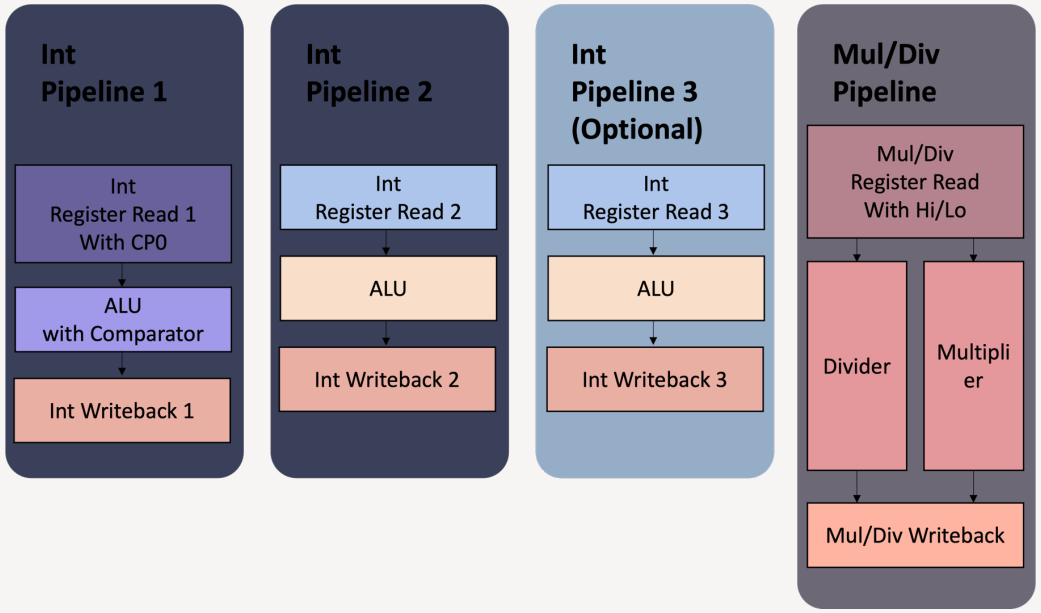
- 将执行结果写入物理寄存器堆和 ROB



附2：微架构——算术流水线

整数流水线

- 标准配置为 3 条（第 3 条可选）
- 对不同流水线进行分工以简化逻辑
- 第 1 条流水线读 CP0 且包含比较器
- 指令发射纯乱序
- 进行 RAW 相关指令唤醒



乘除法流水线

- 处理 Hi/Lo 寄存器读写
- 指令发射纯顺序（FIFO 队列）
- 不同发射队列间形成乱序

附2：微架构——访存流水线

访存阶段1 MEM1

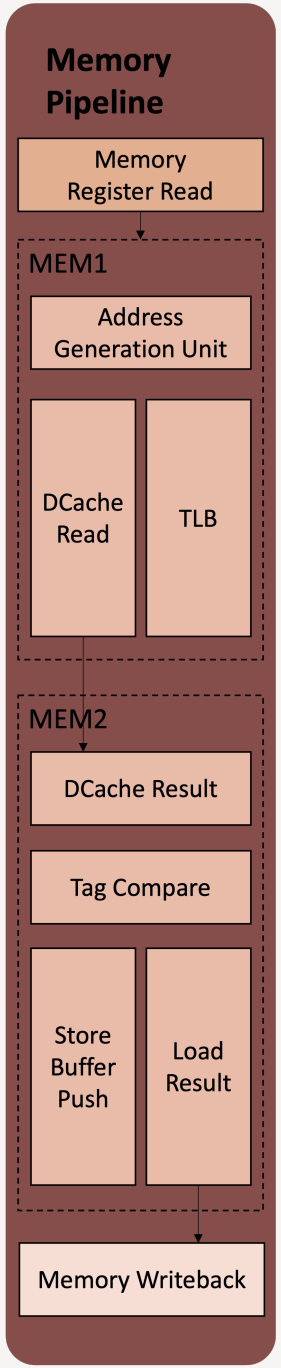
- TLB 查询
- 地址翻译
- D-Cache 查询第 1 阶段

访存阶段2 MEM2

- D-Cache 查询第 2 阶段
- 根据缓存缺失情况进行填充

访存缓冲区 Store Buffer

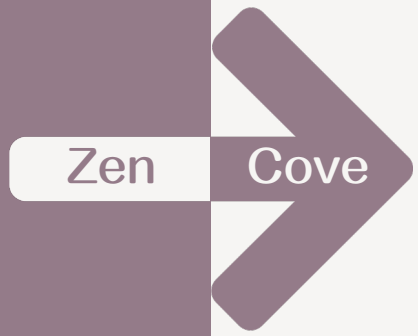
- 处理 STORE 指令和 uncached 地址段 LOAD 指令
- 在 MEM2 进行对应操作



附3：性能对比

优化前

架构	加速比
无分支预测	32.065
无推测唤醒	79.183
无数据前传	79.391



优化后

架构	提升
分支预测	158%
推测唤醒	4.7%
数据前传	4.4%