TrivialMIPS 项目分享

“龙芯杯”第三届全国大学生系统能力培养大赛

清华大学计算机系 陈晟祺

harry-chen@outlook.com

内容大纲

CPU 架构设计与性能优化

“性能就像海绵里的水，只要愿挤，总还是有的”

——不是鲁迅

SoC 设计与硬软件适配

自己动手，丰衣足食

总结与经验

~~奋战三星期，造台计算机！~~

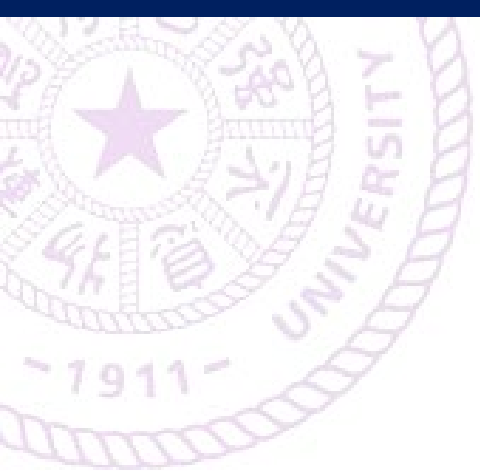
~~——刘卫东老师~~

奋战八星期，造台真正的计算机！

——杰哥

系统类课程之我见

“Because we can.”



CPU 架构设计与性能优化

“性能就像海绵里的水，只要愿挤，总还是有的”

——不是鲁迅

NonTrivialMIPS CPU

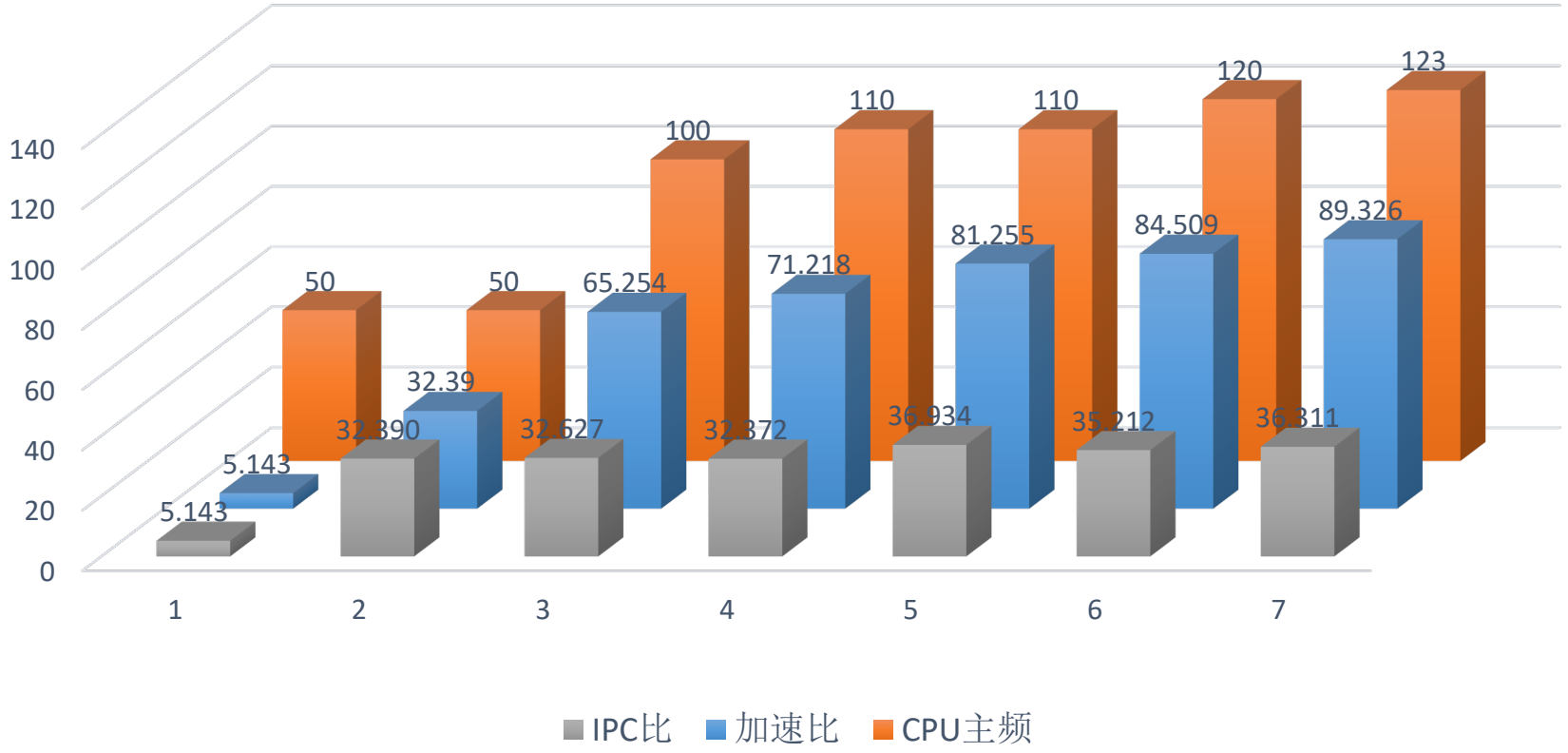
- **双发射**十级**动态调度**流水线
  - 增加读操作数流水段，取指、访存段各扩展为三级
  - 动态延迟部分指令执行至访存阶段，减少数据冲突
  - 2比特分支预测单元，减少控制冲突
- 基于 MIPS32 Release 1 指令集
  - 完整的指令、寄存器、MMU、异常支持
  - 部分 Release 2 内容：CP0 EBase、Config1 支持
- 扩展内容
  - CPU 参数化支持，灵活**可裁剪**功能支持
  - CP1 **浮点协处理器**，实现 IEEE 754 32位浮点运算
  - CP2 用户自定义协处理器，**加速密码学算法**

最终性能结果

- 最高 CPU 主频 123MHz

测试程序	加速比	IPC 绝对值	IPC比值
bitcount	97.886	1.1545	39.308
bubble_sort	85.152	0.9037	34.600
coremark	74.182	0.8278	30.139
crc32	93.048	0.9747	37.805
dhrystone	86.365	0.4983	35.107
quick_sort	75.984	0.8981	30.873
select_sort	101.278	1.2584	41.153
sha	102.928	1.1429	41.823
stream_copy	87.286	0.8598	35.544
stringsearch	94.136	0.5898	38.253
（几何）平均值	**89.326**	**0.8788**	**36.361**

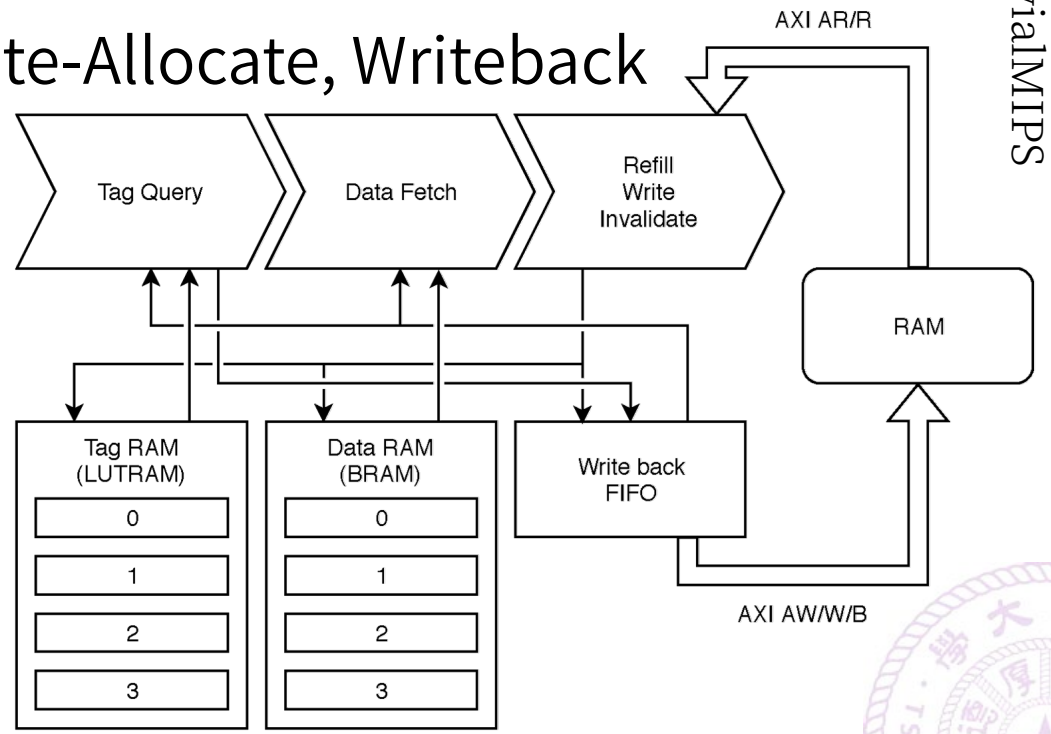
性能优化历程



- 结果均由 CI 自动化测试得到
  - 对每一个 Git 提交（共>500个）进行正确性与性能测试

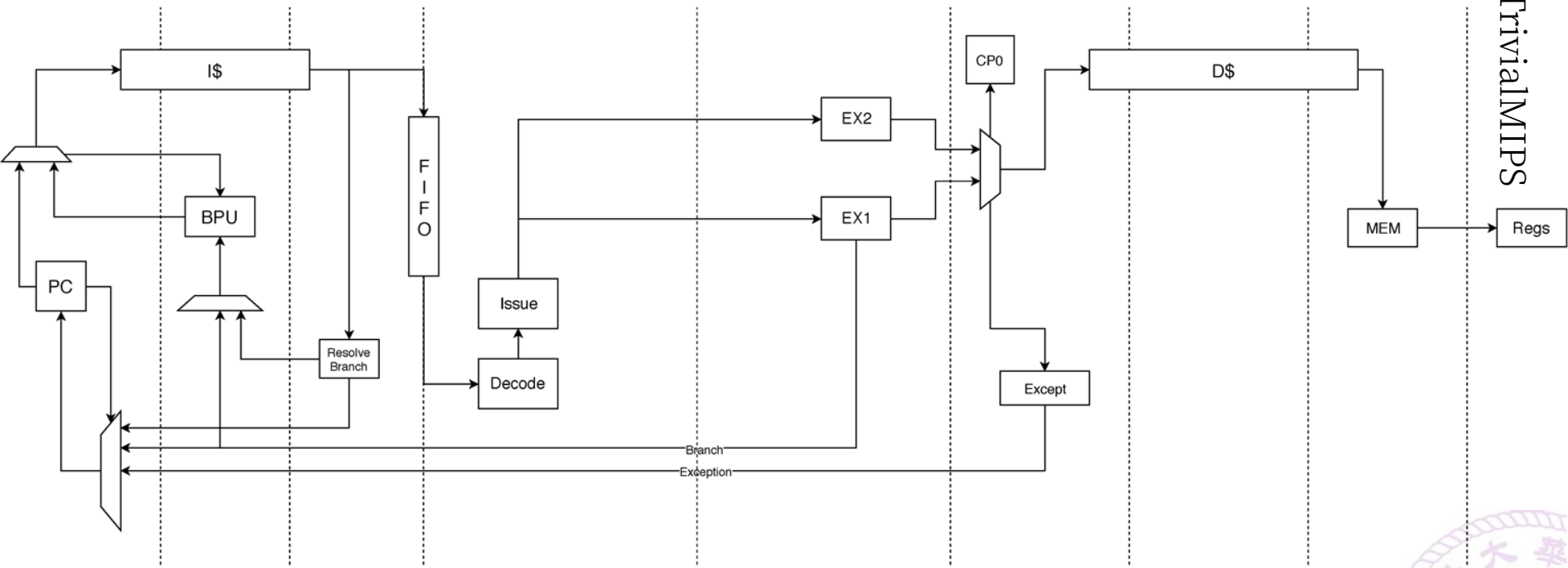
Cache 设计

- 指令 Cache
  - 4路组相联，16KB，虚拟索引物理标识（VIPT）
  - 三级流水
- 数据 Cache
  - 4路, 16KB, VIPT, Write-Allocate, Writeback
  - 三级流水
- Cache 相关指令
  - 查询 Cache 信息
  - 实现 Invalidation



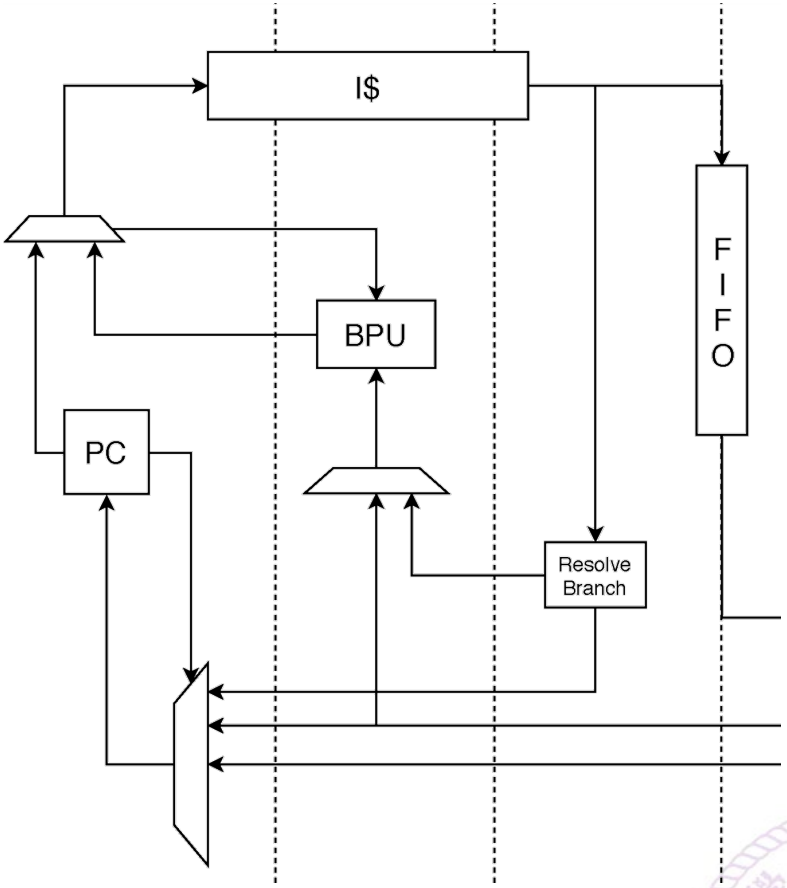
流水线架构

- 主频 110MHz，加速比 71.218，IPC 比 32.372
- 9级流水（3*取指+译码/发射+执行+3*访存+写回）



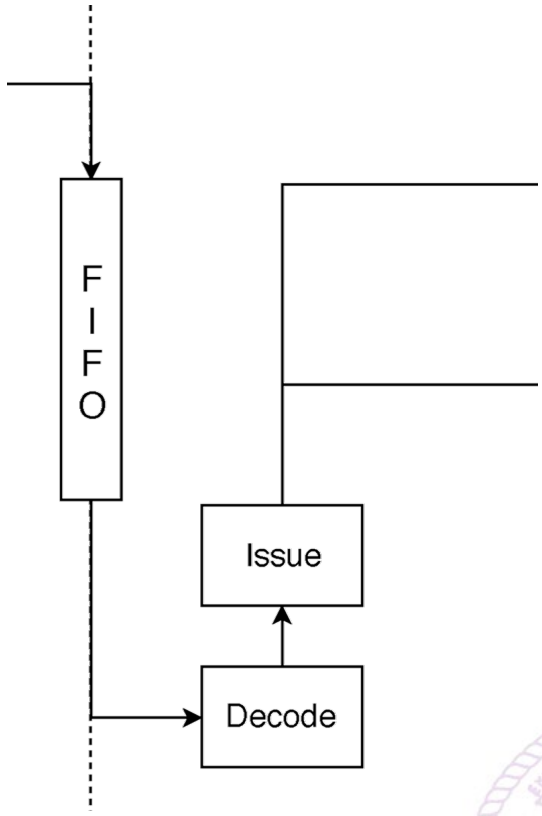
取指流水段

- 2比特分支预测
- 尽早解析分支
  - 确定是否为分支指令
  - 分支目标地址是否正确（JAL，BEQ等）
- 指令FIFO
  - 取指和执行解耦
  - 最大取指数为3
    - 任意指令+分支+延迟槽

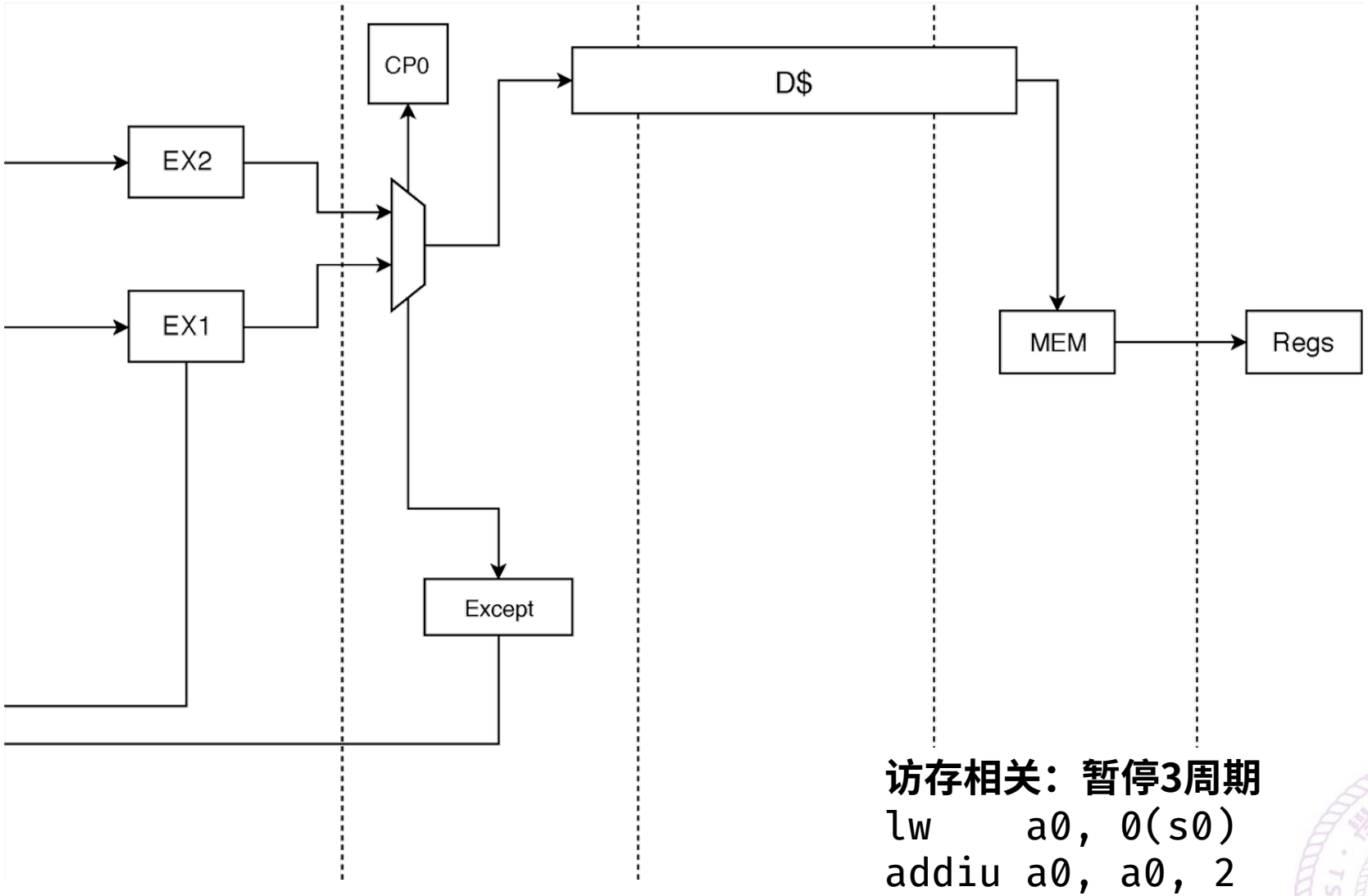


译码与发射流水段

- 发射阶段解决各类冲突
  - 发射包内两条指令的冲突（仅发射一条指令）
    - 数据冲突
    - 结构冲突
      - 乘法器/除法器
      - CP0、CP2
      - 访存
  - 需要暂停发射的情况
    - 数据冲突（主要为访存相关）
    - FIFO为空
    - 分支指令，延迟槽未在FIFO中
  - 分支和延迟槽必须一起发射



剩余流水段

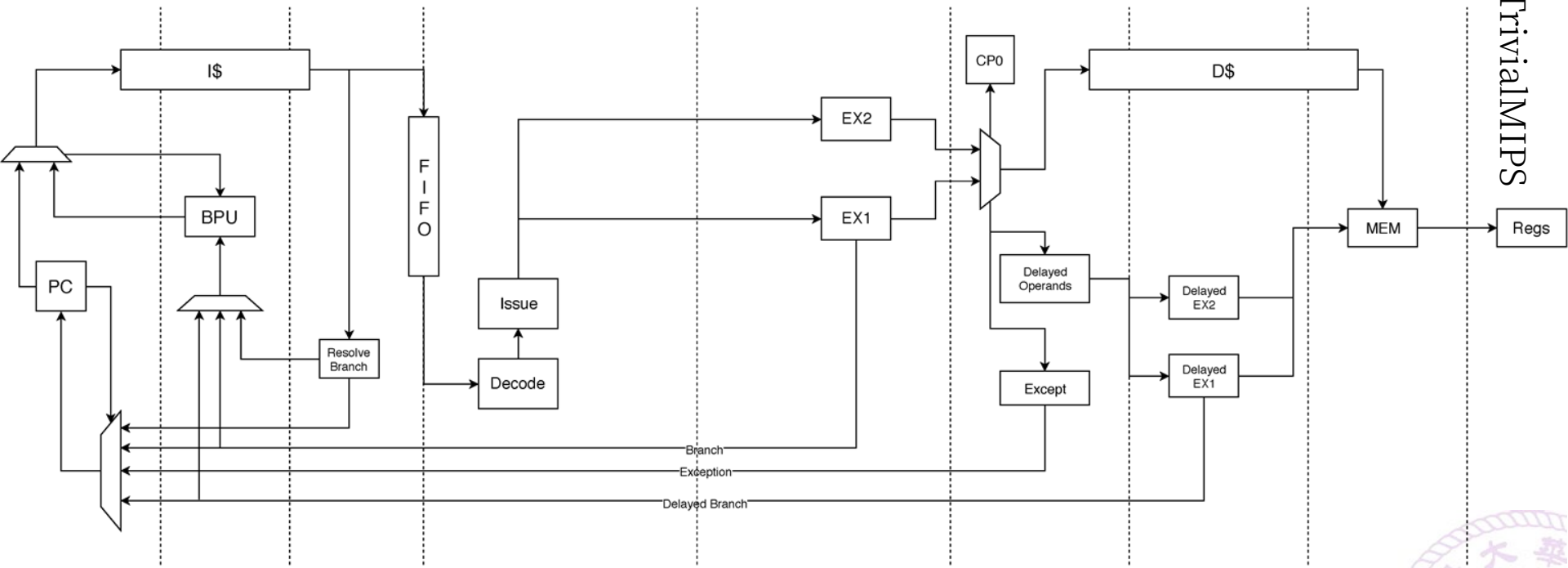


访存相关：暂停3周期

```
lw    a0, 0(s0)
addiu a0, a0, 2
```

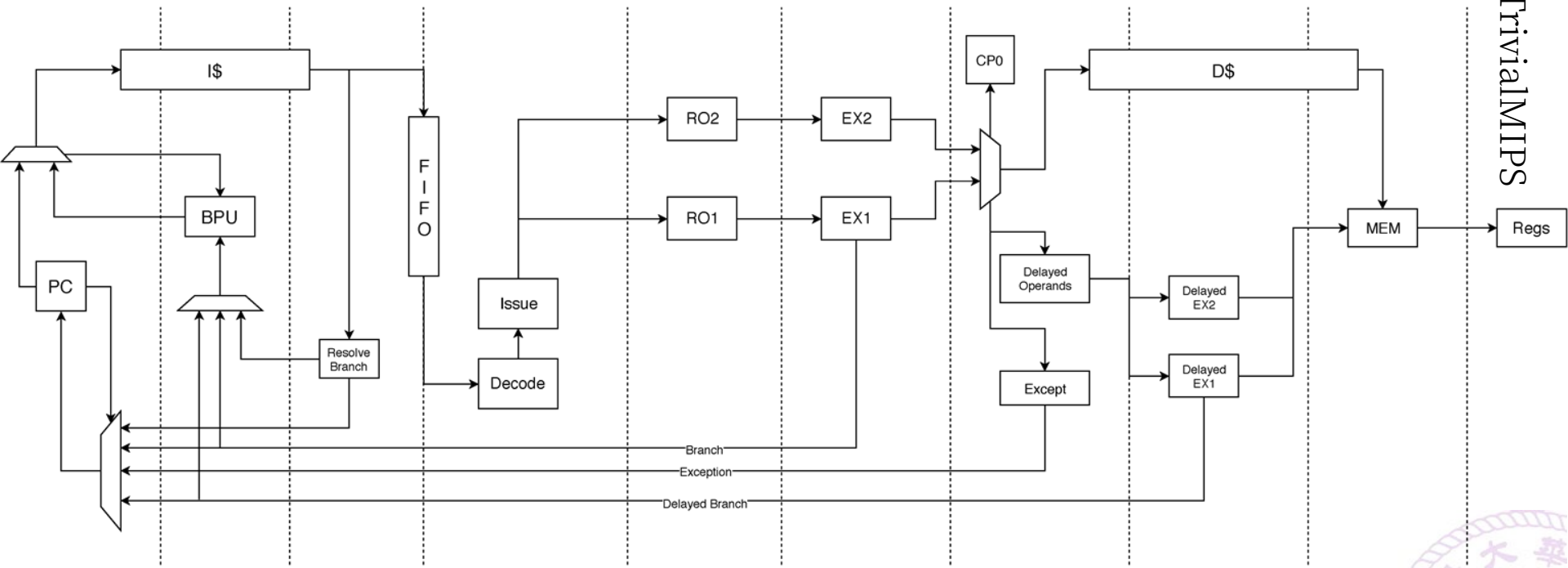
流水线架构改进（1）

- 主频 110MHz，加速比 81.255，IPC 比 36.934
- 增加部分指令延迟执行，减少访存引起数据冲突



流水线架构改进（2）

- 主频 120MHz，加速比 84.509，IPC 比 35.212
- 增加读操作数流水段



其他优化

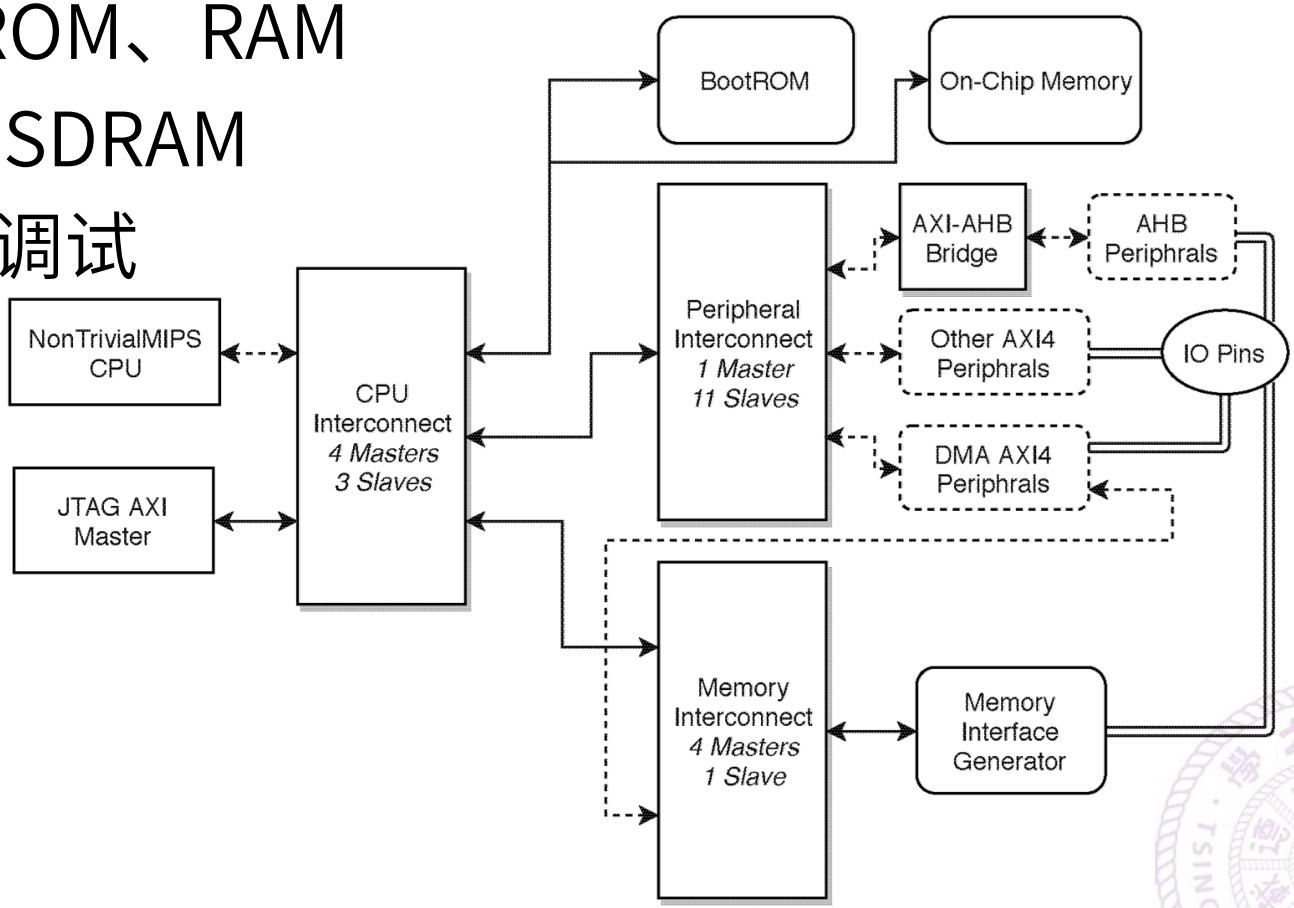
- 主频 123MHz，加速比 89.326，IPC 比 36.261
- Cache 相关
  - 只在 Cache 满时进行替换
  - 使用 PLRU 替换策略
- 多周期指令优化
- CPU 参数调整

SoC 设计与硬软件适配

自己动手，丰衣足食

SoC 整体架构

- 使用 Block Design 自行设计
- CPU@80MHz，外设@100MHz
- 内嵌 BootROM、RAM
- 使用 DDR3 SDRAM
- 支持 JTAG 调试



SoC 硬件外设

- 使用了实验板**全部**硬件接口
- 标准 Xilinx IP
  - 配置 Flash & 外置 SPI Flash（AXI Quad SPI）
  - 以太网（AXI Ethernet Lite）
  - VGA Framebuffer（AXI TFT Controller）
  - 串口（AXI UART 16550）
- 自行开发或移植
  - GPIO 控制器（移植于龙芯提供的 confreg）
  - LCD 控制器（移植于 NaiveMIPS 项目）
  - PS/2 控制器（移植于 Altera University Program）
  - Framebuffer DMA 加速（配合 Xilinx IP）
  - **USB 2.0 FS 控制器**（由开源项目大规模修改而来）

引导程序

- FPGA 从配置 Flash 中读取 bitstream 配置自身
- 初级引导程序（FSBL）— TrivialBootloader
  - 使用 C++ 编写，固化在 BootROM 中
  - 支持从配置 Flash、内存、串口灵活引导 ELF 文件
  - 捕获异常，打印必要信息
- 次级引导程序—U-Boot
  - 移植自 U-Boot 最新版（v2019.7）
    - 添加 USB 驱动
  - 由 FSBL 从配置 Flash 中加载
  - 支持从网络、两块 Flash、U盘引导多格式文件
  - 包含简单的性能测试、内存测试等工具
  - 支持串口/USB 键盘命令交互和美观的启动菜单

Linux 移植

- 基于 Linux v5.2.9（**最新**稳定版，8月17日发布）
  - 使用 gcc 9.2.0 交叉编译
- CPU 适配：基于 MIPS 4Kc
  - 去除 PERF/WATCH 等未实现的指令
  - 关闭 branch likely 等编译选项
  - TLB 完全符合标准，**无需修改**代码
- 板级支持
  - 初始化代码：串口、中断控制器、时钟
  - 设备树描述：自动加载对应驱动程序
  - 默认内核配置：裁剪尺寸、最大化性能

硬件驱动移植

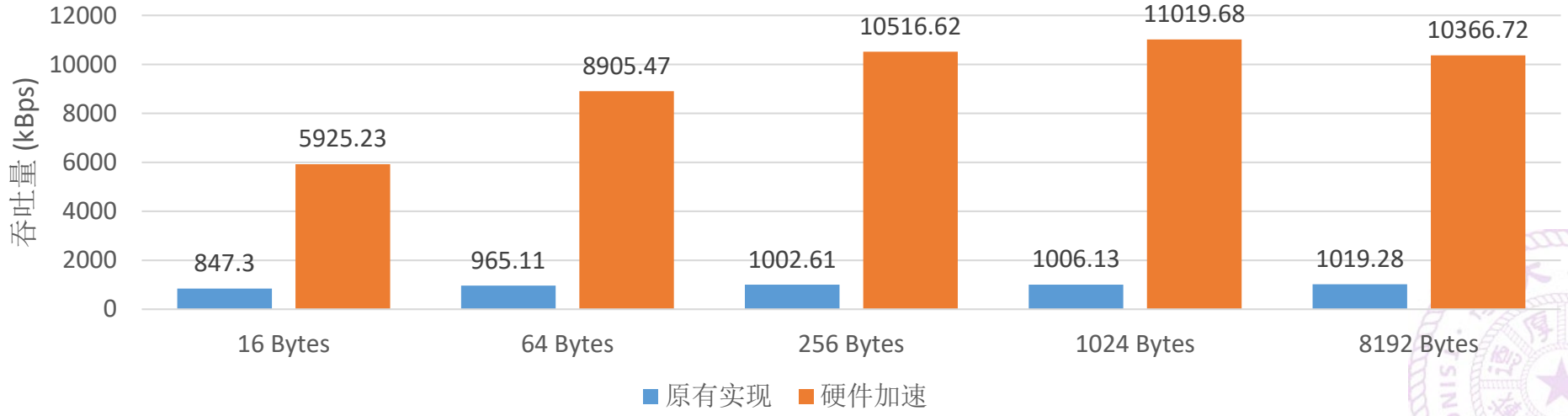
- USB 控制器驱动（**大赛首次适配**）
  - 移植自开源代码，适应硬件控制器修改
  - 注册为 Linux 标准 USB Host Controller
  - 可以正常外接 HID 设备（键盘）、存储设备（U盘）
- LCD 控制器驱动
  - 移植自 NaiveMIPS 项目，修复 bug
  - 支持向屏幕任意位置写入/读取像素数据
- Framebuffer 驱动
  - 基于 Xilinx Framebuffer 驱动修改
  - 在大量拷贝/填充时，使用硬件进行 DMA

Linux 用户态程序

- 基于嵌入式 Linux 工具包 Buildroot
  - Busybox 提供大部分命令行工具
  - 使用 NFS 挂载 rootfs
- 运行 Linux 命令行程序
  - GNU Coreutils (ls, ps, grep, awk, …)
  - GCC、LLVM、Rust 编译器
  - 网络工具 (ip, ping, mtr, wget, nc, …)
  - Python 2.7 (numpy, scipy, …)
- 支持图形化用户界面
  - 在 Framebuffer 驱动支持下运行 Xorg Server
  - 正常绘制窗口、捕获键盘鼠标输入事件
  - 顺利运行 X11 Window 程序，使用 Qt 编写演示程序

软硬件协同设计：密码学算法加速

- MIPS CPU 可选 CP2 协处理器
  - 标准规定可由厂商自定义功能
  - 接入开源硬件 AES 模块，使用 mtc2/mfc2 控制
- OpenSSL/libcrypto 中实现 AES 加解密算法
  - 使用硬件指令替换原有软件实现
  - 可被相关工具软件 (ssh/scp/wget) 等使用
- OpenSSL 测试加速最高 >11 倍



总结与经验

~~奋战三星期，造台计算机！~~

~~——刘卫东老师~~

奋战八星期，造台真正的计算机！

——杰哥

最终成果

- 精益求精的硬件设计
  - **双发射** CPU、十级**动态**流水线、低暂停 Cache
  - 主频最高 **123 MHz**，加速比 **89.326**，IPC 比 **36.261**
  - 独立设计 SoC，驱动**所有**硬件（**USB 零的突破**）
- 完整丰富的配套软件
  - TrivialBootloader 与 U-Boot 两级引导程序
  - **最新版** Linux 移植与**完整驱动程序**适配
  - 运行用户态命令行程序、**图形用户界面**
- 锦上添花的软硬件协同开发
  - 使用硬件 DMA 进行 Framebuffer 绘制加速
  - 实现 MIPS 标准 CP1 32位**浮点运算单元**
  - 使用 CP2 协处理器进行高达 **11x** 的密码学运算加速

获奖感想

- 兴趣永远是**最重要**的
  - 项目从2018年暑假开始，持续至今
  - CPU 主要编写者事实上主攻 ML 和数学
  - 多人合作最后终于达成“造台计算机”的目标
- **连续的**课程序列安排很有帮助
  - 18秋：组成原理/软件工程/编译原理 (双发射 + uCore)
  - 19春：操作系统 (rCore)
  - 19夏：专业实践 (重新设计 CPU + 性能调优 + Linux)
- 经验积累不可或缺
  - 第一届冠军队伍(NaiveMIPS)：张宇翔同学
  - 指导老师：陈康老师、李山山老师、刘卫东老师
  - 其他课程：陈渝老师、向勇老师……

经验：科学的分工

- 各取所长，明确职责
  - CPU / 硬件适配与展示 / SoC 与系统 / Cache 与优化
  - 制定统一接口或者遵照现有协议
  - 每部分工作都可以独立进行（编码、测试）
  - 互相了解各自工作
- 避免重叠或者强依赖关系。杜绝甩锅
  - “一个人写流水线，一个人写 ALU，一个人写访存”
  - “CPU 写了两个月，SoC 实在是没时间搞了”
  - “我一直在等他写完/修完 bug”

经验：源代码管理与版本控制

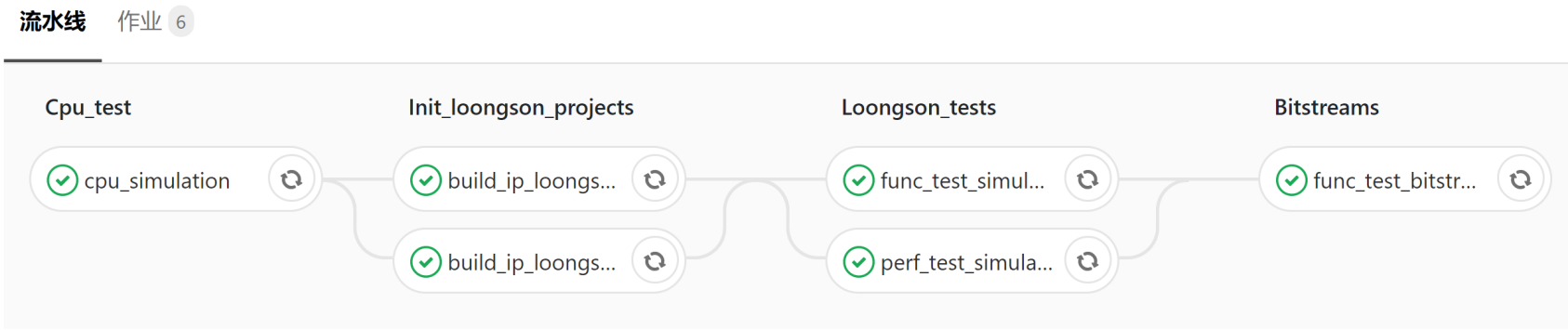
- **最**容易被忽略的
  - “只要功能实现完了就能拿满分了”
  - “又不是软件工程课，在乎这个做什么”
- …也是可能**最**致命的
  - 一年前的这几千行代码到底在做什么？
  - 某个bug是什么时候由谁引入的？
  - 不小心删掉了这一个月的成果怎么办？
- 最佳实践
  - 保持良好的代码风格（文件组织、注释、命名等）
  - 良好的团队合作（来源于科学的分工）
  - 正确使用版本控制工具（git + GitHub / 自建 GitLab）

经验：自动化测试

- 常见问题：
  - 改了一些代码，本来对的也不工作了
  - 自己的电脑太慢了，永远在等综合/编译
  - 助教检查工作量太大，容易有漏网之鱼
- 解决方案：基于 CI （持续集成）的自动化测试
  - 每次提交后自动向高性能服务器提交测试
  - 运行正确性测试（回归测试）、性能测试
  - 自动提取测试结果和指标，获得最终产物（如二进制）
- 实例：比赛中的 CPU 参数调优
  - 可调整：指令/数据 Cache 的尺寸/相连度/替换算法等
  - 人工过于繁琐，使用脚本进行参数的矩阵测试
  - 获得尽可能高的时钟频率和性能分数

自动化测试（例）

- 基于 GitLab CI 与 Linux CI 服务器




```
----PASS!!!
I$ miss count:         170
D$ miss count:         155
D$ access count:       33664
UD miss count:        3368
Instruction count:      122212
Cycle count:      210641
Branch count:       25941
Branch mispredict count:        2092
Memory (data) stall count:       89620
NoCF mispredict:           6
I$ miss rate:     0.000000
Branch miss rate: 0.080645
CPI:              1.723571
Performance test passed: Yes
Performance test cycles: 0x0001840c



================================================================
Test end!
```

```
----[10594875 ns] Number 8'd87 Functional Test Point PASS!!!
        [10602000 ns] Test is running, debug_wb_pc = 0x00000000
        [10612000 ns] Test is running, debug_wb_pc = 0x00000000
        [10622000 ns] Test is running, debug_wb_pc = 0x00000000
        [10632000 ns] Test is running, debug_wb_pc = 0x00000000
----[10633425 ns] Number 8'd88 Functional Test Point PASS!!!
        [10642000 ns] Test is running, debug_wb_pc = 0x00000000
        [10652000 ns] Test is running, debug_wb_pc = 0x00000000
        [10662000 ns] Test is running, debug_wb_pc = 0x00000000
        [10672000 ns] Test is running, debug_wb_pc = 0x00000000
----[10672055 ns] Number 8'd89 Functional Test Point PASS!!!
================================================================
Test end!
----PASS!!!
```

经验：定量分析

- 感谢 Patterson 大作《量化研究方法》
- 每一个指标都很重要
  - 内部：Cache 缺失率、暂停次数、分支预测失败率
  - 外部：加速比、CPI
- 记录与分析
  - 前述自动化测试过程可进行完整过程记录
  - 分析变化，比较得失
- 最终取得平衡
  - 永远需要 tradeoff
  - 并不意味着需要放弃某一部分

系统类课程之我见

“Because we can.”

一些拙见

- 突出课程核心，减少额外负担
  - 使用在线实验平台、自动化评测等手段辅助完成实验
  - 隐藏不必要细节，减少学习成本
- 紧贴时代潮流，增加前沿内容
  - 组成原理：MIPS → RISC-V
  - 操作系统：Legacy x86 → x86_64 / aarch64 / RV
  - 编译原理：文法分析 → 数据流分析与优化
- 增强课程关联，鼓励综合实践
  - 交叉讲授知识，不局限于课程本身
  - 设计含有挑战性的课程联合实验（见下）

清华系统类课程实验（独立）

- 计算机组成原理
  - MIPS32 CPU 与简单 SoC，无操作系统
- 编译原理
  - 实现简单的教学语言编译器前端
- 操作系统
  - （大部分）基于虚拟机的教学操作系统移植/开发
- ~~计算机网络原理~~
  - 基于 X86 的软件路由转发与路由协议实现
- 都很有趣，但是……缺点什么？

我们的小目标

- *“在自己造的 CPU 上运行自己写的操作系统，控制自己设计的硬件路由器，运行自己写的编译器，用编译器再完成对操作系统的编译。”*
- 已经达成：
  - 设计完整的 CPU：MIPS / RV32 / RV64
  - 运行操作系统：uCore / rCore （兼容 Linux 程序）
  - 撰写编译器：decaf 教学语言，Rust / Java 的编译器
  - 控制路由器：硬件路由转发平面，软件控制平面
- 暂时缺少比较综合的成果
- 胜利就在眼前！

Slides 分享

谢谢！