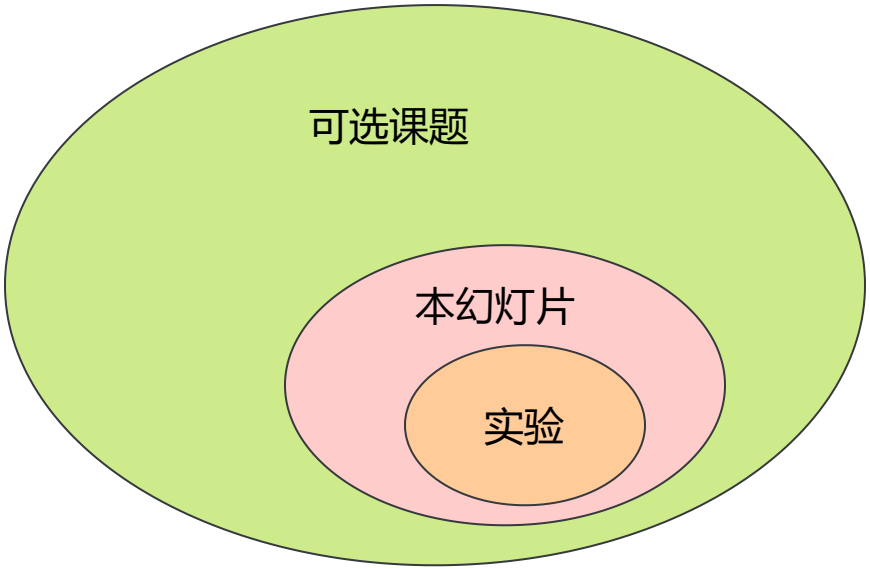
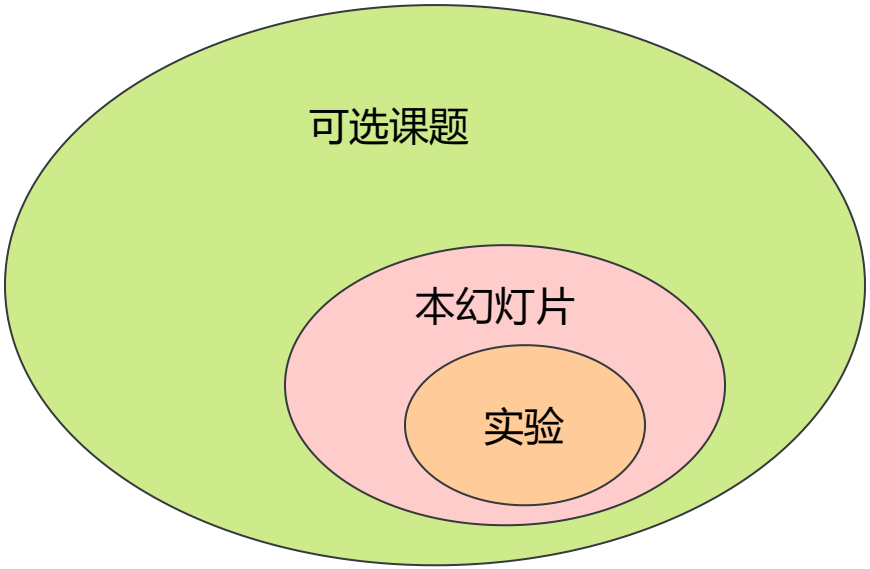
Final Project Topics

概览

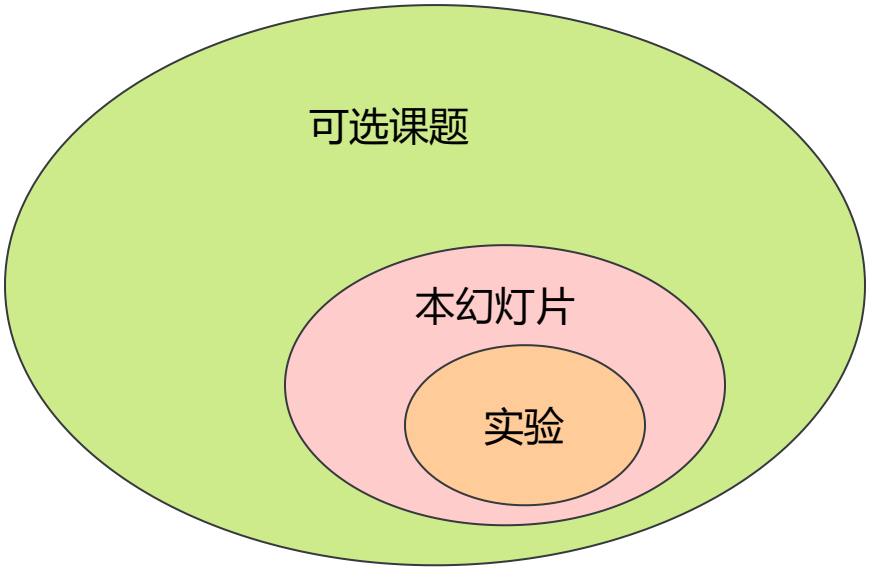


概览



选择自己感兴趣的方向进行探索

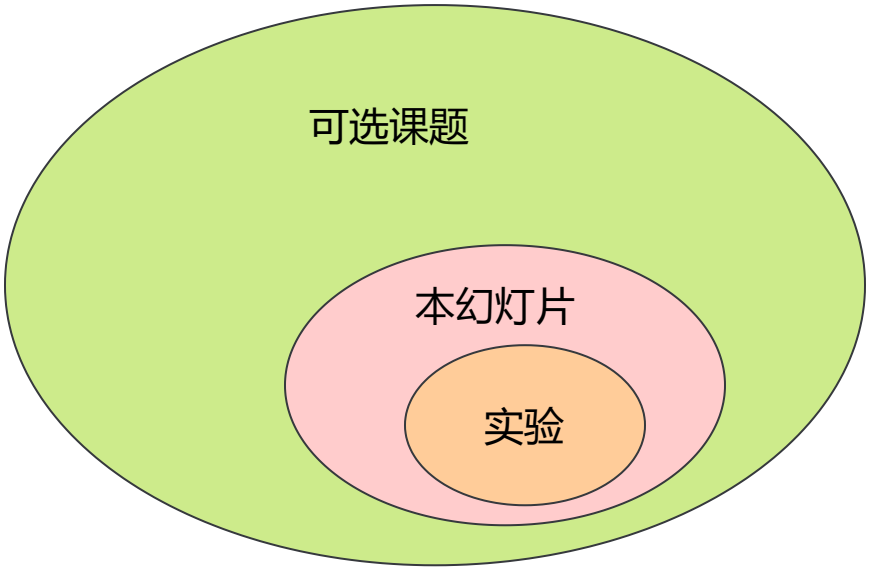
概览



选择自己感兴趣的方向进行探索

- 流水线

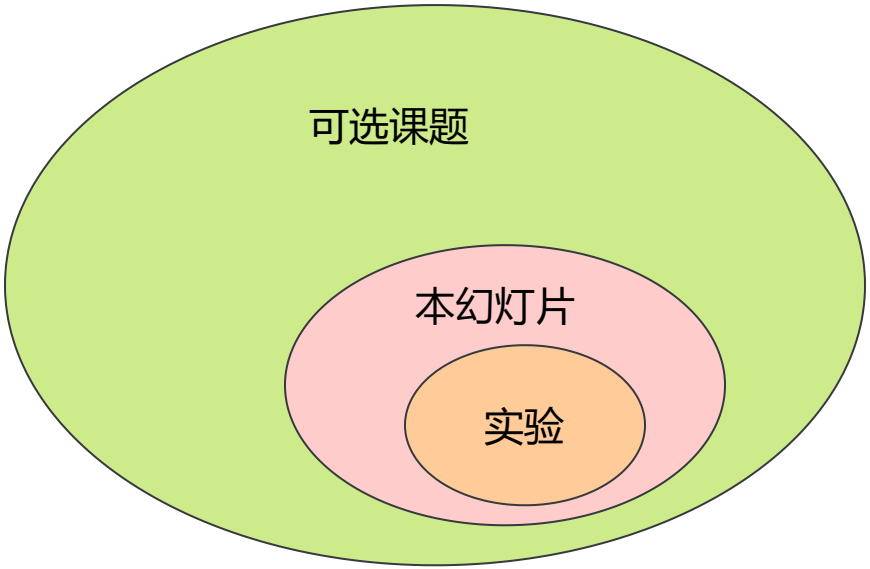
概览



选择自己感兴趣的方向进行探索

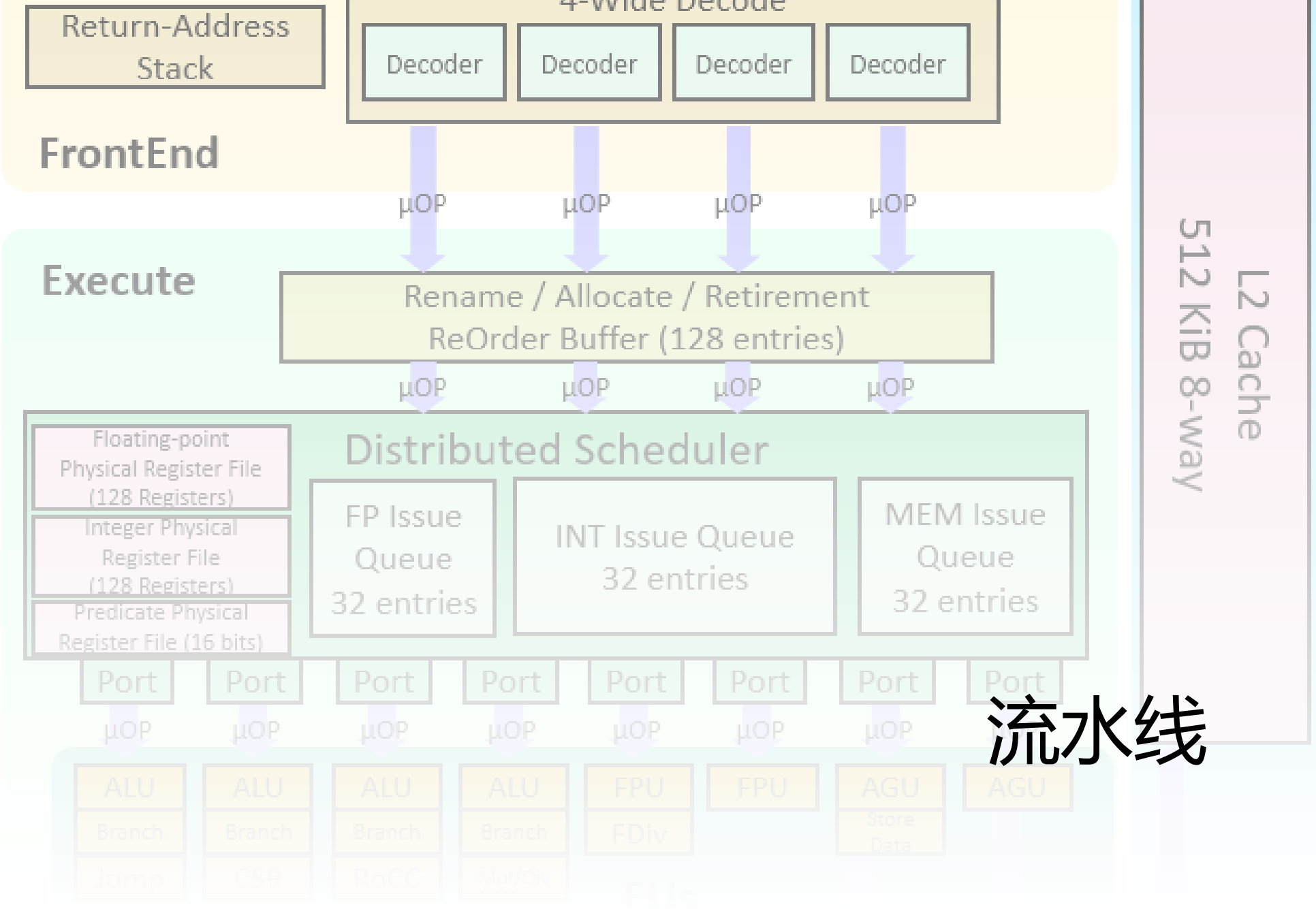
- 流水线
- 访存

概览



选择自己感兴趣的方向进行探索

- 流水线
- 访存
- 外设


Return-Address Stack
4-Wide Decode
Decoder
Decoder
Decoder
Decoder
FrontEnd
µOP
µOP
µOP
µOP
Execute
Rename / Allocate / Retirement
ReOrder Buffer (128 entries)
µOP
µOP
µOP
µOP
Floating-point Physical Register File (128 Registers)
Integer Physical Register File (128 Registers)
Predicate Physical Register File (16 bits)
Distributed Scheduler
FP Issue Queue 32 entries
INT Issue Queue 32 entries
MEM Issue Queue 32 entries
Port
Port
Port
Port
Port
Port
Port
Port
µOP
µOP
µOP
µOP
µOP
µOP
µOP
ALU
ALU
ALU
ALU
FPU
FPU
AGU
AGU
Branch
Branch
Branch
Branch
FDiv
Store Data
Jump
CSR
RoCC
Mul/Div
L2 Cache
512 KiB 8-way

流水线

顺序双发（Inorder Dual-issue）

流水线每个阶段可以容纳两条指令

所有指令顺序完成

难度：★★★☆☆

顺序双发（Inorder Dual-issue）

流水线每个阶段可以容纳两条指令

所有指令顺序完成

难度：★★★☆☆

- 需要处理更多的冲突和转发

顺序双发（Inorder Dual-issue）

流水线每个阶段可以容纳两条指令

所有指令顺序完成

难度：★★★☆☆

- 需要处理更多的冲突和转发
- 处理延迟槽

顺序双发（Inorder Dual-issue）

流水线每个阶段可以容纳两条指令

所有指令顺序完成

难度：★★★☆☆

- 需要处理更多的冲突和转发
- 处理延迟槽
- 需要缓冲能够每周期提供两条指令

顺序双发（Inorder Dual-issue）

流水线每个阶段可以容纳两条指令

所有指令顺序完成

难度：★★★☆☆

- 需要处理更多的冲突和转发
- 处理延迟槽
- 需要缓冲能够每周期提供两条指令
- 有很多成功的例子

乱序多发（Out of Order, OoO）

支持多发射乱序执行

通常是乱序四发

难度：★★★★★

乱序多发（Out of Order, OoO）

支持多发射乱序执行

通常是乱序四发

难度：★★★★★

- 流水线的架构将变得十分复杂

乱序多发（Out of Order, OoO）

支持多发射乱序执行

通常是乱序四发

难度：★★★★★

- 流水线的架构将变得十分复杂
- 需要处理写后写冲突

乱序多发（Out of Order, OoO）

支持多发射乱序执行

通常是乱序四发

难度：★★★★★

- 流水线的架构将变得十分复杂
- 需要处理写后写冲突
- 依然需要顺序提交

乱序多发（Out of Order, OoO）

支持多发射乱序执行

通常是乱序四发

难度：★★★★★

- 流水线的架构将变得十分复杂
- 需要处理写后写冲突
- 依然需要顺序提交
- 寄存器重命名

乱序多发（Out of Order, OoO）

支持多发射乱序执行

通常是乱序四发

难度：★★★★★

- 流水线的架构将变得十分复杂
- 需要处理写后写冲突
- 依然需要顺序提交
- 寄存器重命名
- 预测执行（speculative execution）

乱序多发（Out of Order, OoO）

支持多发射乱序执行

通常是乱序四发

难度：★★★★★

- 流水线的架构将变得十分复杂
- 需要处理写后写冲突
- 依然需要顺序提交
- 寄存器重命名
- 预测执行（speculative execution）
- **非常消耗硬件资源**

乱序多发（Out of Order, OoO）

支持多发射乱序执行

通常是乱序四发

难度：★★★★★

- 流水线的架构将变得十分复杂
- 需要处理写后写冲突
- 依然需要顺序提交
- 寄存器重命名
- 预测执行（speculative execution）
- **非常消耗硬件资源**
- 可参考的案例很少

乱序多发（Out of Order, OoO）

支持多发射乱序执行

通常是乱序四发

难度：★★★★★

- 流水线的架构将变得十分复杂
- 需要处理写后写冲突
- 依然需要顺序提交
- 寄存器重命名
- 预测执行（speculative execution）
- **非常消耗硬件资源**
- 可参考的案例很少

有野心的同学可以尝试 😀

多核（Symmetric Multi-Processor, SMP）

两个物理核心

难度：★★★★★★★★

多核（Symmetric Multi-Processor, SMP）

两个物理核心

难度：★★★★★★★★

- 不适合上龙芯杯：没有测试

多核（Symmetric Multi-Processor, SMP）

两个物理核心

难度：★★★★★★★★

- 不适合上龙芯杯：没有测试
- 需要考虑内存一致性模型

多核（Symmetric Multi-Processor, SMP）

两个物理核心

难度：★★★★★★★★

- 不适合上龙芯杯：没有测试
- 需要考虑内存一致性模型
  - 很多单核上的优化都会被限制

多核（Symmetric Multi-Processor, SMP）

两个物理核心

难度：★★★★★★★

- 不适合上龙芯杯：没有测试
- 需要考虑内存一致性模型
  - 很多单核上的优化都会被限制
- 核心之间需要通信：总线

多核（Symmetric Multi-Processor, SMP）

两个物理核心

难度：★★★★★★★

- 不适合上龙芯杯：没有测试
- 需要考虑内存一致性模型
  - 很多单核上的优化都会被限制
- 核心之间需要通信：总线
- 需要考虑缓存一致性

多核（Symmetric Multi-Processor, SMP）

两个物理核心

难度：★★★★★★★

- 不适合上龙芯杯：没有测试
- 需要考虑内存一致性模型
  - 很多单核上的优化都会被限制
- 核心之间需要通信：总线
- 需要考虑缓存一致性
- 测试

多核（Symmetric Multi-Processor, SMP）

两个物理核心

难度：★★★★★★★

- 不适合上龙芯杯：没有测试
- 需要考虑内存一致性模型
  - 很多单核上的优化都会被限制
- 核心之间需要通信：总线
- 需要考虑缓存一致性
- 测试

分支预测（Branch Prediction）

实现分支预测

难度：★☆☆☆☆

分支预测（Branch Prediction）

实现分支预测

难度：★☆☆☆☆

- 无条件跳转、有条件跳转

分支预测（Branch Prediction）

实现分支预测

难度：★☆☆☆☆

- 无条件跳转、有条件跳转
- 跳转地址预测（Branch Target Buffer, BTB）

分支预测（Branch Prediction）

实现分支预测

难度：★☆☆☆☆

- 无条件跳转、有条件跳转
- 跳转地址预测（Branch Target Buffer, BTB）
- `jal` / `jr`：硬件调用栈

分支预测（Branch Prediction）

实现分支预测

难度：★☆☆☆☆

- 无条件跳转、有条件跳转
- 跳转地址预测（Branch Target Buffer, BTB）
- `jal` / `jr`：硬件调用栈
- 仿真或测试验证预测准确率

浮点数运算（FPU）

添加浮点数指令

难度：★★☆☆☆

浮点数运算（FPU）

添加浮点数指令

难度：★★☆☆☆

- MIPS 中 CP1 是浮点数单元

浮点数运算（FPU）

添加浮点数指令

难度：★★☆☆☆

- MIPS 中 CP1 是浮点数单元
- 可以使用开源的 FPU

浮点数运算（FPU）

添加浮点数指令

难度：★★☆☆☆

- MIPS 中 CP1 是浮点数单元
- 可以使用开源的 FPU
- 可以自己实现 FPU：★★★★

浮点数运算（FPU）

添加浮点数指令

难度：★★☆☆☆

- MIPS 中 CP1 是浮点数单元
- 可以使用开源的 FPU
- 可以自己实现 FPU：★★★★
- 测试

硬件加速指令

添加自定义的 coprocessor 2

难度：★★☆☆☆

硬件加速指令

添加自定义的 coprocessor 2

难度：★★☆☆☆

- 提供加速特定运算的协处理器

硬件加速指令

添加自定义的 coprocessor 2

难度：★★☆☆☆

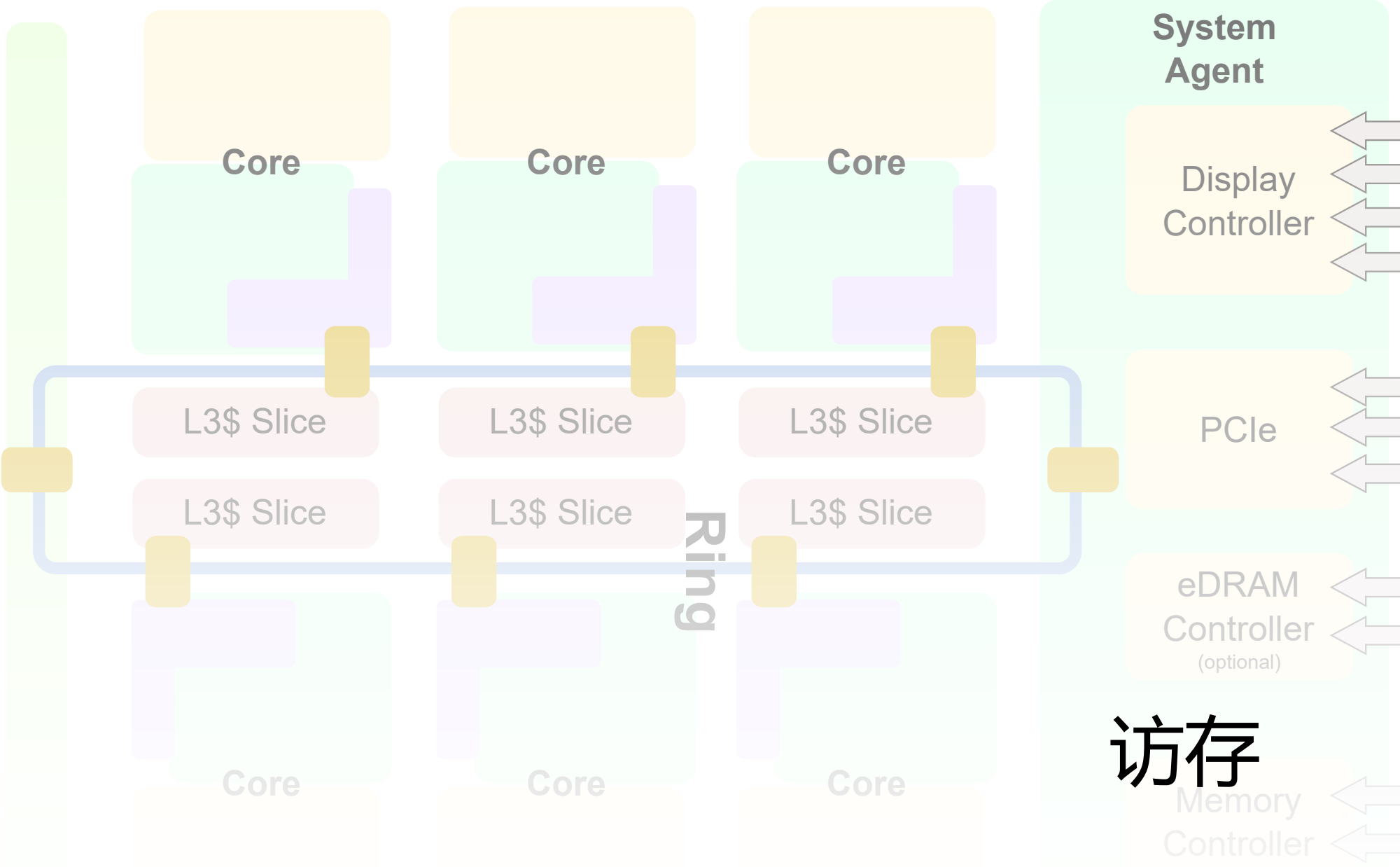
- 提供加速特定运算的协处理器
- 清华的 NontrivialMIPS 集成了 AES 模块

硬件加速指令

添加自定义的 coprocessor 2

难度：★★☆☆☆

- 提供加速特定运算的协处理器
- 清华的 NontrivialMIPS 集成了 AES 模块
- 设计利用协处理器的测试代码


System Agent
Core
Core
Core
Display Controller
L3$ Slice
L3$ Slice
L3$ Slice
PCIe
L3$ Slice
L3$ Slice
L3$ Slice
Ring
eDRAM Controller
(optional)
访存
Core
Core
Core
Memory Controller

LUTRAM/BRAM

使用 LUTRAM/BRAM 作为 cache 的存储

难度：★☆☆☆☆

LUTRAM/BRAM

使用 LUTRAM/BRAM 作为 cache 的存储

难度：★☆☆☆☆

- 不要再用数组了！

LUTRAM/BRAM

使用 LUTRAM/BRAM 作为 cache 的存储

难度：★☆☆☆☆

- 不要再用数组了！
  - 虽然有时候 Vivado 会自动使用 LUTRAM/BRAM 来实现数组，但需要按照一定的写法才能让 Vivado 正确识别出来
  - 否则就是拿 FF 实现

LUTRAM/BRAM

使用 LUTRAM/BRAM 作为 cache 的存储

难度：★☆☆☆☆

- 不要再用数组了！
  - 虽然有时候 Vivado 会自动使用 LUTRAM/BRAM 来实现数组，但需要按照一定的写法才能让 Vivado 正确识别出来
  - 否则就是拿 FF 实现
  - 用 RAM 模块可以确保综合出 LUTRAM/BRAM

流水线缓存（Pipelined Cache）

将访存切为多级流水线完成

降低访存延时

难度：★★☆☆☆

流水线缓存（Pipelined Cache）

将访存切为多级流水线完成

降低访存延时

难度：★★☆☆☆

- 怎样划分才是合理的?

流水线缓存（Pipelined Cache）

将访存切为多级流水线完成

降低访存延时

难度：★★☆☆☆

- 怎样划分才是合理的?
- 级数过多 → 阻塞和转发

流水线缓存（Pipelined Cache）

将访存切为多级流水线完成

降低访存延时

难度：★★☆☆☆

- 怎样划分才是合理的?
- 级数过多 → 阻塞和转发
- 测试

分库缓存（Banked Cache）

使用多个 RAM 模块作为 cache 的存储

提供更多的访存接口

难度：★★★☆☆

分库缓存（Banked Cache）

使用多个 RAM 模块作为 cache 的存储

提供更多的访存接口

难度：★★★☆☆

- 设计 RAM 的组织方式

分库缓存（Banked Cache）

使用多个 RAM 模块作为 cache 的存储

提供更多的访存接口

难度：★★★☆☆

- 设计 RAM 的组织方式
- 处理 bank conflict

分库缓存（Banked Cache）

使用多个 RAM 模块作为 cache 的存储

提供更多的访存接口

难度：★★★☆☆

- 设计 RAM 的组织方式
- 处理 bank conflict
- 测试

分库缓存（Banked Cache）

使用多个 RAM 模块作为 cache 的存储

提供更多的访存接口

难度：★★★☆☆

- 设计 RAM 的组织方式
- 处理 bank conflict
- 测试
- 需要超标量流水线

写缓冲（Store Buffer）

隐藏写操作的延时

可以做数据转发和写合并

难度：★★☆☆☆

写缓冲（Store Buffer）

隐藏写操作的延时

可以做数据转发和写合并

难度：★★☆☆☆

- 设计 FIFO

写缓冲（Store Buffer）

隐藏写操作的延时

可以做数据转发和写合并

难度：★★☆☆☆

- 设计 FIFO
- 测试

非阻塞缓存（Non-blocking Cache）

添加 fill buffer 和 victim buffer

在 cache miss 和 cache line 换出时，cache 能继续工作

难度：★★☆☆☆

非阻塞缓存（Non-blocking Cache）

添加 fill buffer 和 victim buffer

在 cache miss 和 cache line 换出时，cache 能继续工作

难度：★★☆☆☆

- 通常需要 cache 有多个端口

非阻塞缓存（Non-blocking Cache）

添加 fill buffer 和 victim buffer

在 cache miss 和 cache line 换出时，cache 能继续工作

难度：★★☆☆☆

- 通常需要 cache 有多个端口
  - BRAM 可以有两个独立的端口

非阻塞缓存（Non-blocking Cache）

添加 fill buffer 和 victim buffer

在 cache miss 和 cache line 换出时，cache 能继续工作

难度：★★☆☆☆

- 通常需要 cache 有多个端口
  - BRAM 可以有两个独立的端口
  - 分 bank

非阻塞缓存（Non-blocking Cache）

添加 fill buffer 和 victim buffer

在 cache miss 和 cache line 换出时，cache 能继续工作

难度：★★☆☆☆

- 通常需要 cache 有多个端口
  - BRAM 可以有两个独立的端口
  - 分 bank
- 测试

硬件预取（Hardware Prefetch）

设计并实现预取机制

难度：★★★☆☆

硬件预取（Hardware Prefetch）

设计并实现预取机制

难度：★★★☆☆

- 判断预取的时机

硬件预取（Hardware Prefetch）

设计并实现预取机制

难度：★★★☆☆

- 判断预取的时机
- 等距访存的识别

硬件预取（Hardware Prefetch）

设计并实现预取机制

难度：★★★☆☆

- 判断预取的时机
- 等距访存的识别
- 设计更多的策略

硬件预取（Hardware Prefetch）

设计并实现预取机制

难度：★★★☆☆

- 判断预取的时机
- 等距访存的识别
- 设计更多的策略
- 通过仿真和测试证明预取策略是有效的

自定 SoC

修改龙芯提供的 SoC

例如加宽 CPU 对外的访存带宽等

难度：★★★☆☆

自定 SoC

修改龙芯提供的 SoC

例如加宽 CPU 对外的访存带宽等

难度：★★★☆☆

- 增强你的 cache，和 SoC 配合

自定 SoC

修改龙芯提供的 SoC

例如加宽 CPU 对外的访存带宽等

难度：★★★☆☆

- 增强你的 cache，和 SoC 配合
- 用仿真或者测试证明性能的提升

外设

虚拟地址翻译

实现 TLB 和相关指令

MIPS 中没有硬件页表

难度：★☆☆☆☆

虚拟地址翻译

实现 TLB 和相关指令

MIPS 中没有硬件页表

难度：★☆☆☆☆

- 实现 VIPT 策略

虚拟地址翻译

实现 TLB 和相关指令

MIPS 中没有硬件页表

难度：★☆☆☆☆

- 实现 VIPT 策略
- 需要实现相关的 TLB 指令

虚拟地址翻译

实现 TLB 和相关指令

MIPS 中没有硬件页表

难度：★☆☆☆☆

- 实现 VIPT 策略
- 需要实现相关的 TLB 指令
- 二级 TLB

UART 串口通信

使用 Vivado 的 IP 核，实现开发板和你的电脑间的串口通信

难度：★☆☆☆☆

UART 串口通信

使用 Vivado 的 IP 核，实现开发板和你的电脑间的串口通信

难度：★☆☆☆☆

- 了解 UART 协议

UART 串口通信

使用 Vivado 的 IP 核，实现开发板和你的电脑间的串口通信

难度：★☆☆☆☆

- 了解 UART 协议
- 编写使用串口的测试代码

启动 PMON/UBoot

启动 PMON/UBoot，为启动操作系统做准备

需要串口通信

难度：★★☆☆☆

启动 PMON/UBoot

启动 PMON/UBoot，为启动操作系统做准备

需要串口通信

难度：★★☆☆☆

- 学习如何烧写硬件

启动 PMON/UBoot

启动 PMON/UBoot，为启动操作系统做准备

需要串口通信

难度：★★☆☆☆

- 学习如何烧写硬件
- 可能需要实现额外的指令：branch-likely、`cache`、`mul`

启动 PMON/UBoot

启动 PMON/UBoot，为启动操作系统做准备

需要串口通信

难度：★★☆☆☆

- 学习如何烧写硬件
- 可能需要实现额外的指令：branch-likely、`cache`、`mul`
- 没有仿真，可能需要上板调试 😅

启动 PMON/UBoot

启动 PMON/UBoot，为启动操作系统做准备

需要串口通信

难度：★★☆☆☆

- 学习如何烧写硬件
- 可能需要实现额外的指令：branch-likely、`cache`、`mul`
- 没有仿真，可能需要上板调试 😅
  - 使用 Verilator 进行仿真

启动 Linux

自行编译、启动 Linux shell

需要串口通信

难度：★★★★☆

启动 Linux

自行编译、启动 Linux shell

需要串口通信

难度：★★★★☆

- 有机会了解 Linux 的启动过程

VGA 显示

我们的开发板上有 VGA 接口

将其包装成外设，并接入 CPU

难度：★★★★☆

VGA 显示

我们的开发板上有 VGA 接口

将其包装成外设，并接入 CPU

难度：★★★★☆

- 需要跨时钟域

VGA 显示

我们的开发板上有 VGA 接口

将其包装成外设，并接入 CPU

难度：★★★★☆

- 需要跨时钟域
- 可能需要双缓冲区以避免屏幕撕裂现象

VGA 显示

我们的开发板上有 VGA 接口

将其包装成外设，并接入 CPU

难度：★★★★☆

- 需要跨时钟域
- 可能需要双缓冲区以避免屏幕撕裂现象
- 仿真和测试

VGA 显示

我们的开发板上有 VGA 接口

将其包装成外设，并接入 CPU

难度：★★★★☆

- 需要跨时钟域
- 可能需要双缓冲区以避免屏幕撕裂现象
- 仿真和测试
  - 使用 Verilator 和 OpenGL/SDL/... 进行仿真

键盘/鼠标/USB.....

发挥你的主观能动性 🙄

结语

暂定的 deadline 是第 16 周星期天

有可能可以往后调

中期报告欢迎大家报名

祝大家期末~~愉快~~ 😅