基于 MIPS 指令集的乱序四发射 CPU 设计与实现

哈尔滨工业大学（深圳）1 队

指导教师：仇洁婷　薛睿

哈尔滨工业大学（深圳）

2020 年 8 月 19 日

目录

目录

主体工作概述

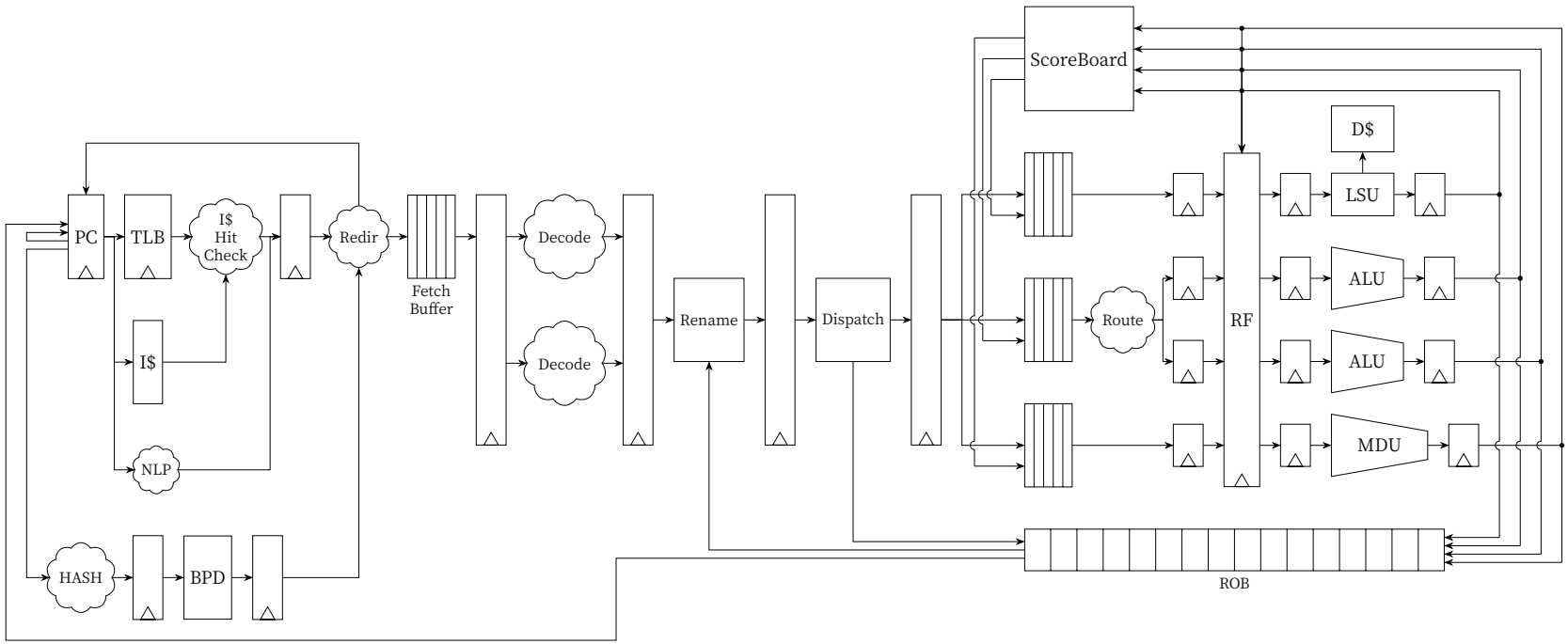
- 初赛
  - 实现大赛官方要求的 57 条指令 + 系统测试所需指令，支持精确中断和异常
  - 通过功能测试 + 性能测试 + 系统测试
  - 4 路组相联，4KiB I-Cache + 4 路组相联，16KiB D-Cache
  - 简易分支历史缓冲（Next Line Predictor），尝试 TAGE 预测器
  - 运行频率75MHz，性能得分38.8
- 决赛
  - 添加 GShare 分支预测和分支历史缓冲（BTB）
  - 优化关键路径，频率提升至88MHz，IPC 比值27.5（后续提升至29，非提交版本）
  - 提前 `Load/Store` 指令对于其他指令的唤醒
  - 添加 TLB，通过系统测试第三档，并启动 PMON 命令行（但频率受影响，故最终提交版本无 TLB）

目录

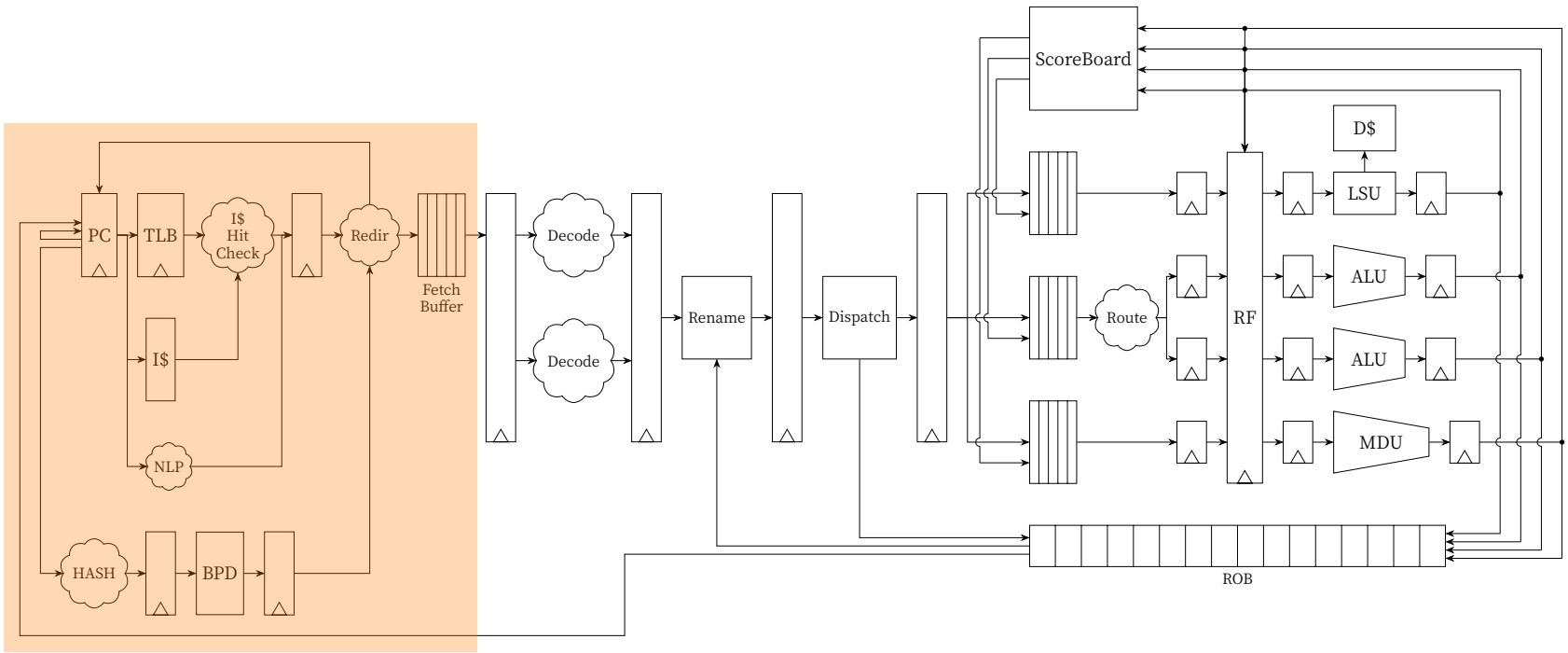
流水线总体结构

- 主流水线架构
  - 前端：取指段、译码及重命名段、分派段
  - 后端：保留站、发射仲裁、执行单元、重定序缓冲
- 关键技术
  - Overriding 分支预测
  - 寄存器重命名
  - 发射队列的设计和指令的仲裁
  - 唤醒与旁路
  - 非阻塞 D-Cache
  - 投机执行的状态恢复

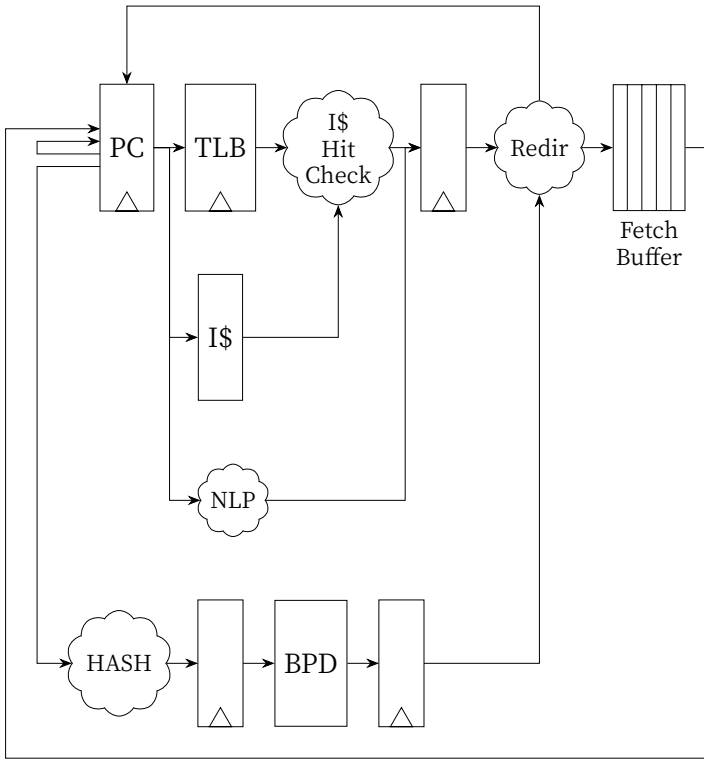
流水线总体结构


ScoreBoard
D$
LSU
ALU
ALU
MDU
RF
Route
PC
TLB
I$ Hit Check
Redir
Fetch Buffer
Decode
Decode
Rename
Dispatch
I$
NLP
HASH
BPD
ROB

取指段

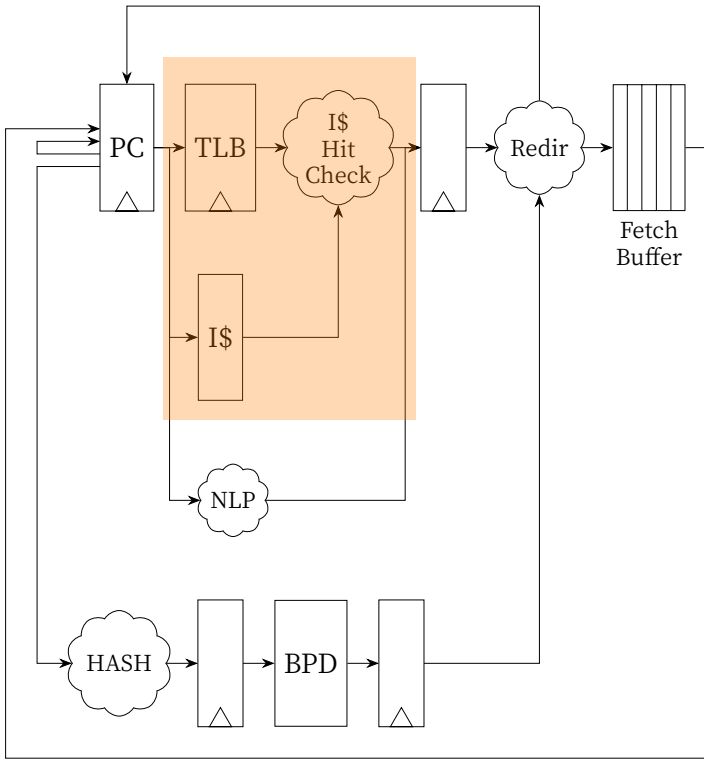

PC
TLB
I$ Hit Check
Redir
Fetch Buffer
I$
NLP
HASH
BPD
Decode
Decode
Rename
Dispatch
ScoreBoard
Route
RF
D$
LSU
ALU
ALU
MDU
ROB

取指段



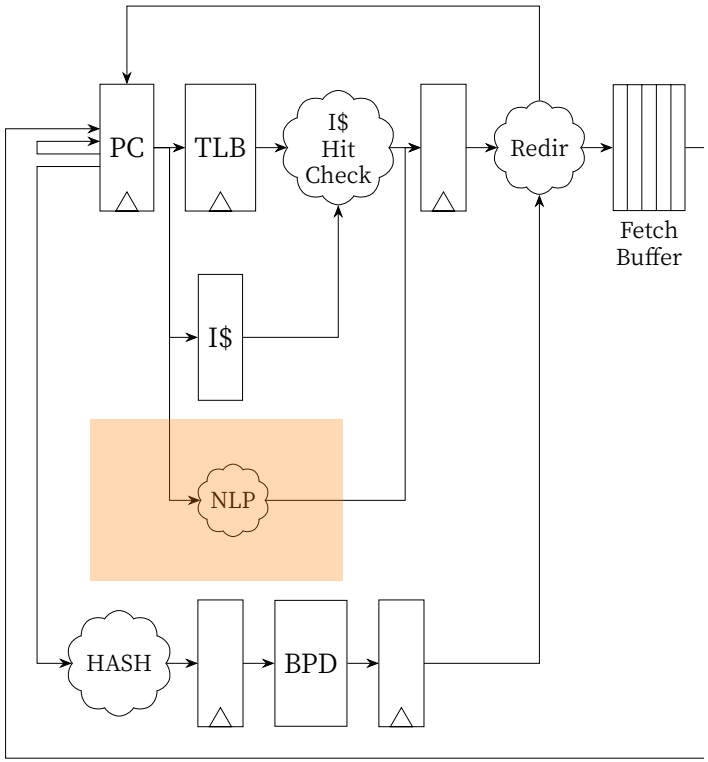
- 指令缓存
- 下址预测器
- 二级分支预测器
- 指令缓冲

取指段：指令缓存



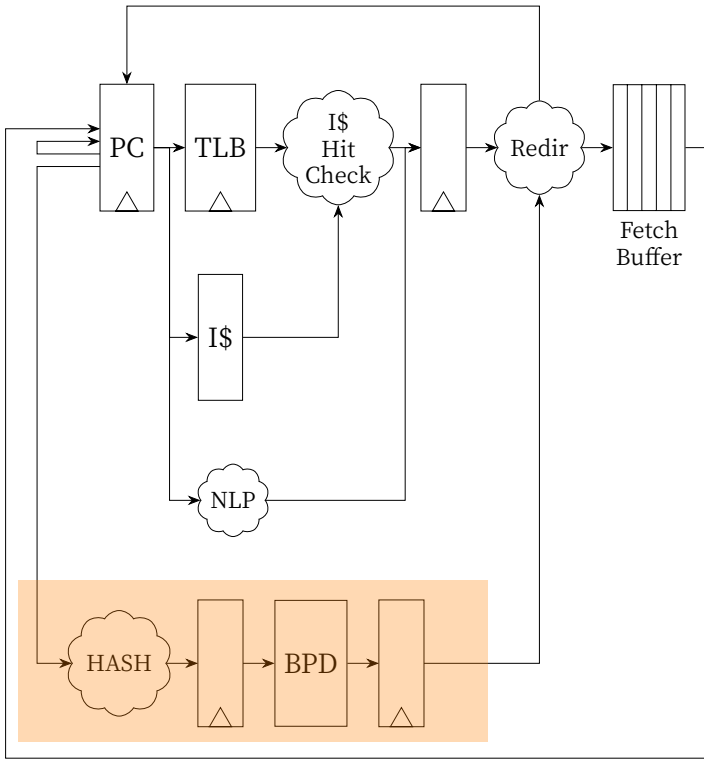
- 指令缓存设计
  - 每项 16 字节，四路组相联，64 行，4KiB
  - VIPT
  - 伪-LRU 替换算法

取指段：下址预测器



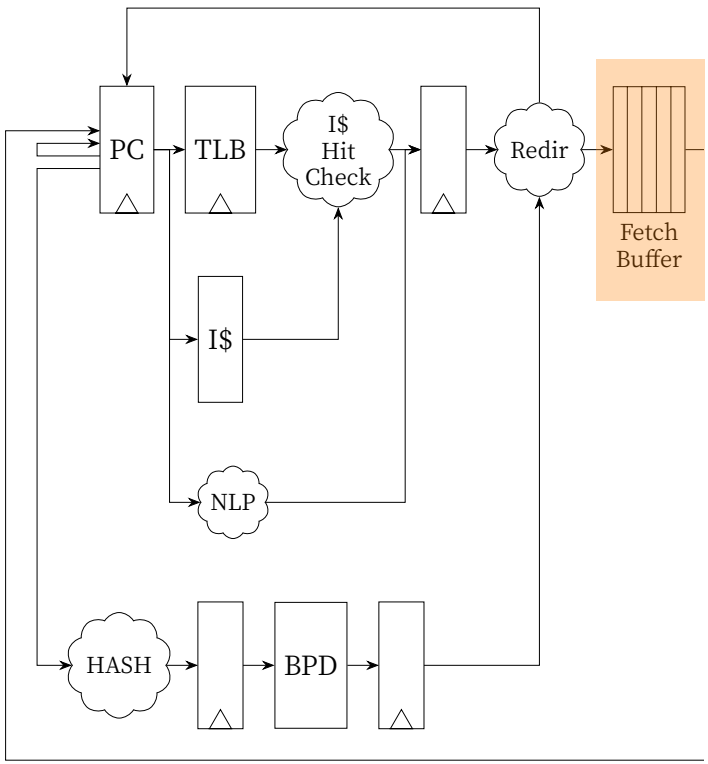
- 下址预测器设计
  - 32 路全相联结构
  - 二位饱和计数器
  - 异步读取、同步写入
  - FIFO 替换算法

取指段：主分支预测器



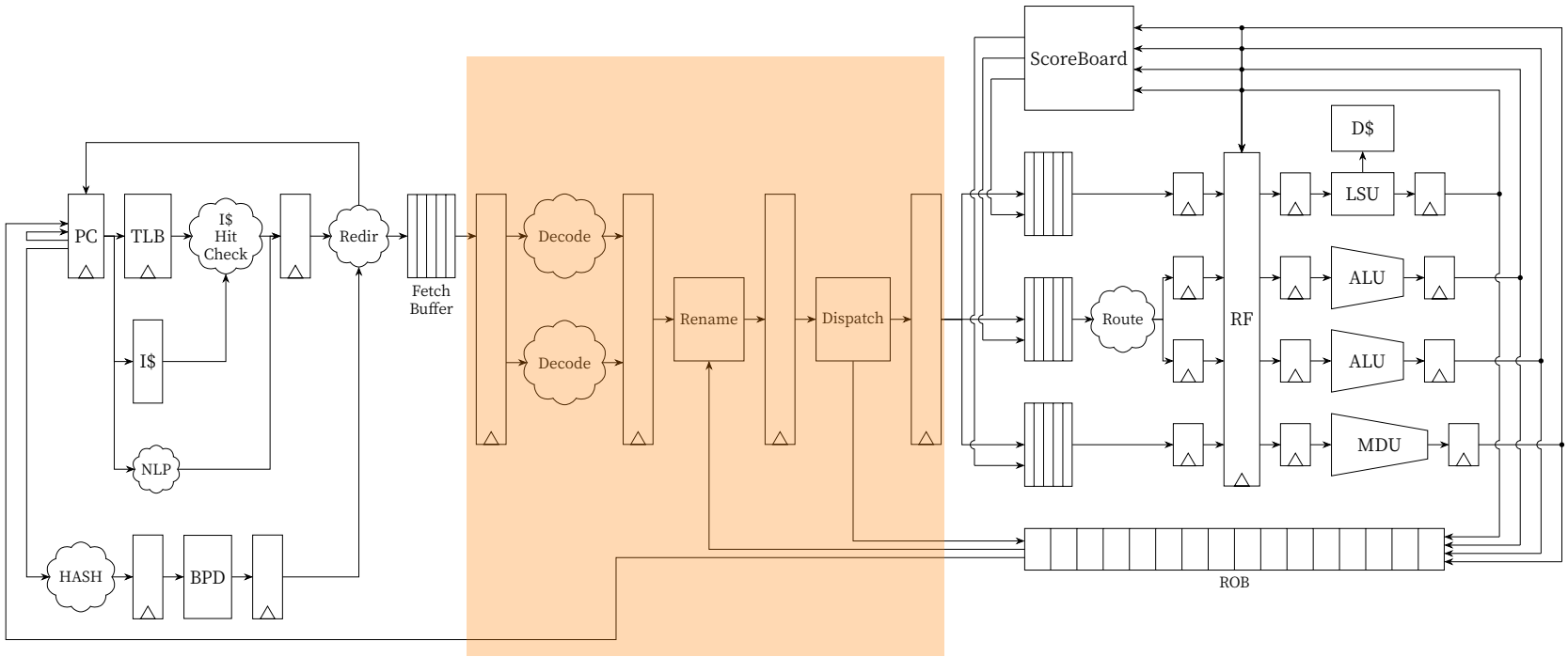
- 下址预测器设计
  - 主预测器使用 GShare 分支预测算法
  - 全局历史长度为 4
  - 使用低 2 位与 PC 的低 12 位进行异或，寻址 PHT
  - 当主预测器中没有有效记录时，则拒绝做出预测
  - 尝试过实现 TAGE 预测器，但效果不好

取指段：主分支预测器

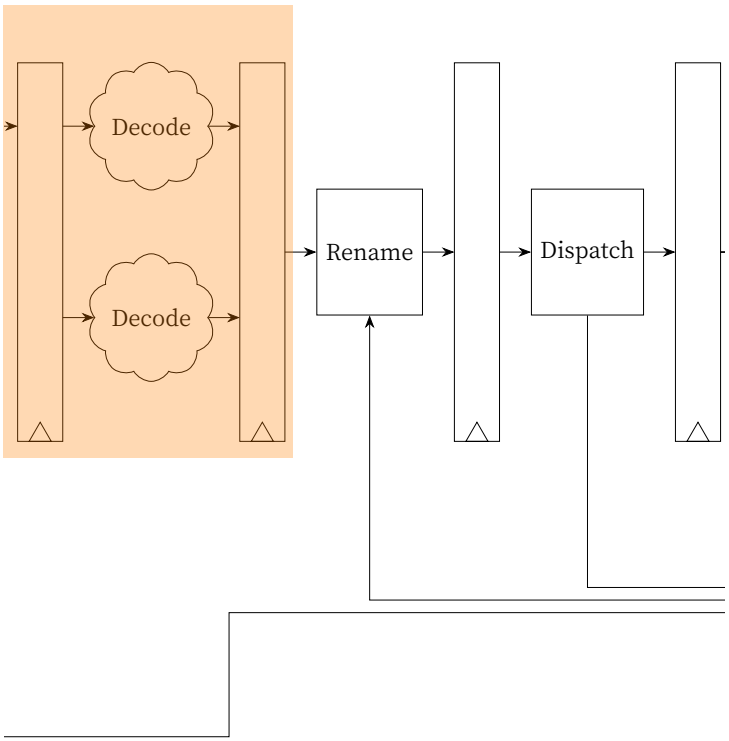


- 指令缓冲
  - 16 项循环队列
  - 二进二出，队列透传
  - 保证跳转指令和延迟槽同步出队
  - 用于前后端解耦

译码、重命名、分发

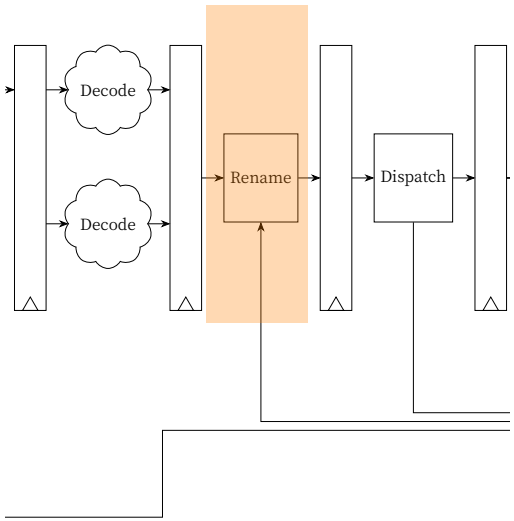

ScoreBoard
D$
LSU
ALU
ALU
MDU
RF
Route
PC
TLB
I$ Hit Check
Redir
Fetch Buffer
Decode
Decode
Rename
Dispatch
I$
NLP
HASH
BPD
ROB

译码、重命名、分发



- 指令在此阶段被译码为微操作码 (uOP)
- 一周期最多同时译码 2 条指令
- 对于乘除法指令，其目的寄存器有 2 个，故一周期只能译码 1 条 *
- 乘法指令在译码阶段，被拆分成 MULTHI/MULTLO 形式；除法同理

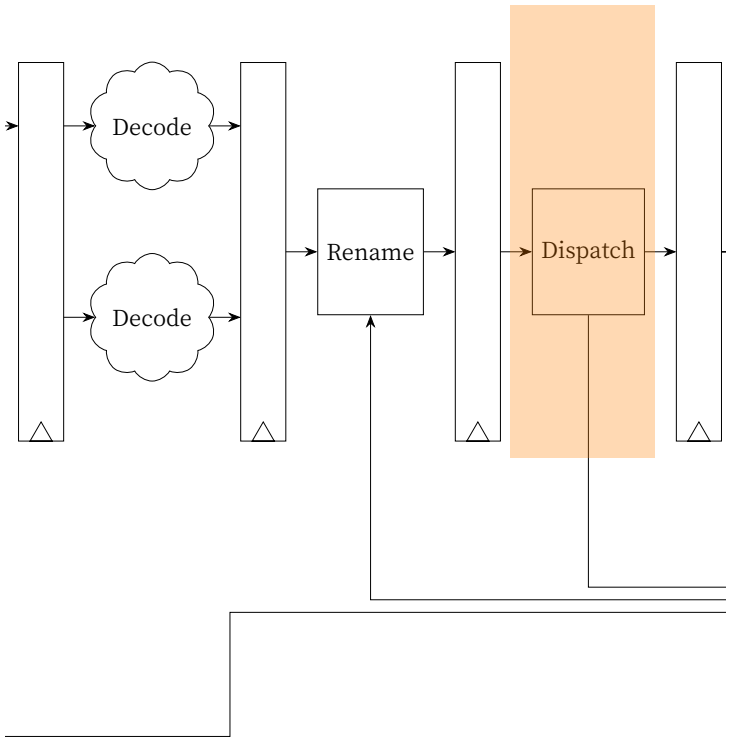
译码、重命名、分发



- 寄存器重命名流程：
  - 编码 Free Vector，获取新的目的物理寄存器
  - 从映射表中读出旧的映射关系
  - 向映射表写入新的映射关系
  - 更新 Free Vector

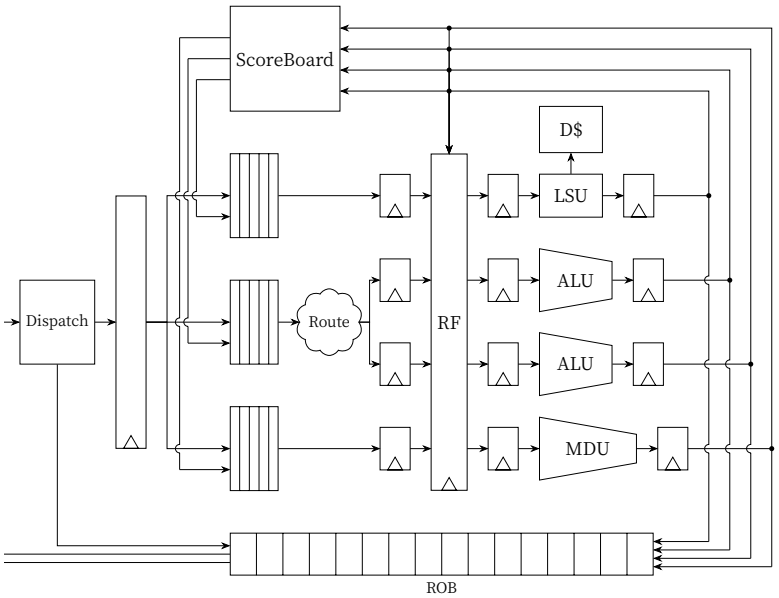
- 解决指令之间的假相关
- 采用统一的物理寄存器文件 + 映射表
- 使用 Free Vector 来管理寄存器的分配状态
  - 使用 2 个优先编码器对 Free Vector 进行优先编码，即可分配出两个空余的物理寄存器
  - 使用 Committed Free Vector 来管理已提交的寄存器分配状态
- 使用映射表（Map Table）来记录映射关系
  - 使用 Committed Map Table 来管理已提交的寄存器映射关系
- 状态恢复：直接使用 Committed 的表格覆盖当前的表格即可

译码、重命名、分发



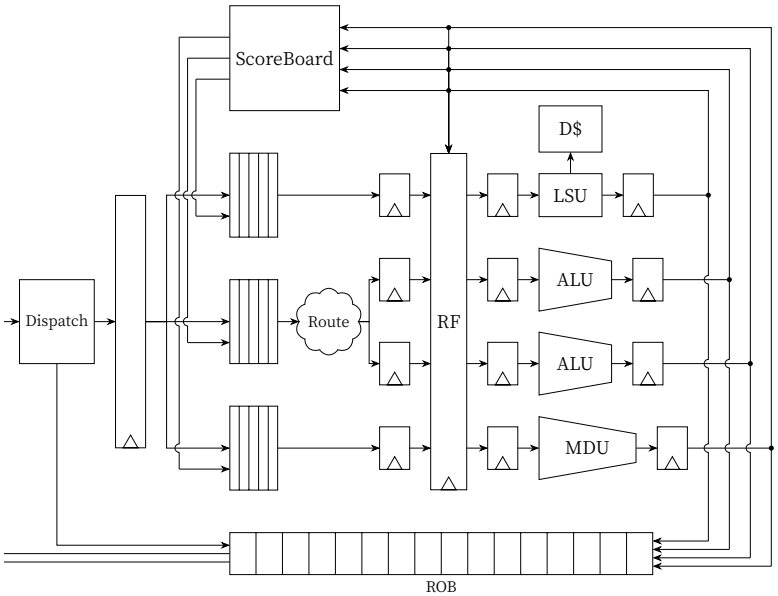
- 将指令路由到正确的指令队列中
  - 通过译码阶段得到的指令信息，生成对应指令队列的写使能信号
  - 每周期最多分发 2 条指令
  - 对于 CP0 指令，由于 CP0 寄存器未进行重命名，没有恢复的措施，所以必须等待处理器处于确定状态（无潜在的异常指令和分支预测错误）才可执行，故在此阶段，若有 CP0 指令，则必须等待 ROB 为空时，才可进入指令队列。
- 在此阶段，同时将指令顺序地写入 ROB，以在乱序执行之前记录指令原本的顺序

发射与执行：概述



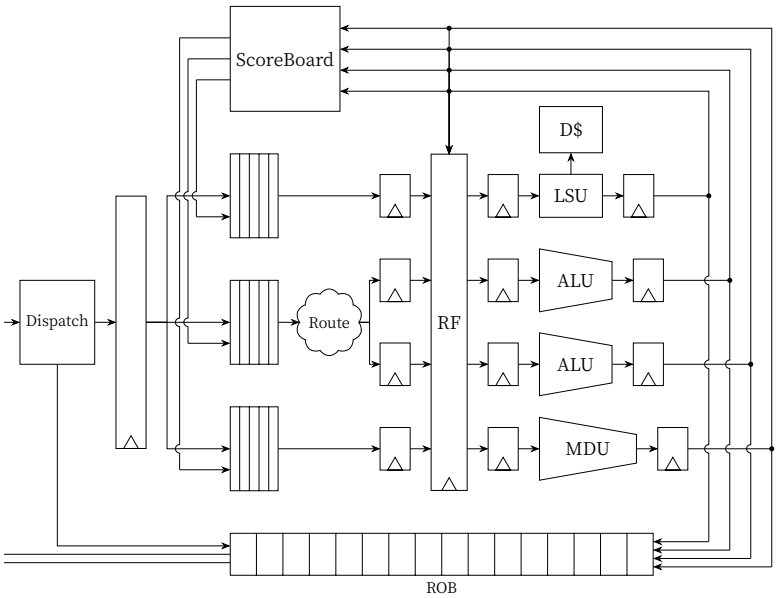
- 3 个发射队列，容量为 8
  - 整数指令：每周期发射 2 条
  - 乘除法指令：每周期发射 1 条
  - 访存指令：每周期发射 1 条
- 4 条执行流水线
  - 整数 *2
  - 乘除 *1
  - 访存 *1
- 部分旁路
  - ALU to ALU

发射与执行：指令队列



- 发射队列为 2 进 2 出（算术）或 2 进 1 出（访存/乘除法）
- 队列中指令顺序严格按照指令的前后排序
- 采用压缩式的队列，不存在气泡
- 仲裁逻辑相对简单，只需优先编码器即可完成（发射与仲裁并未成为 CPU 中的关键路径）
- Load/Store 指令采用半乱序发射的方式，避免相关性：
  - Store 必须在队列首部，才可发射；
  - Store 之后的 Load 不得提前于 Store 发射

发射与执行：Scoreboard

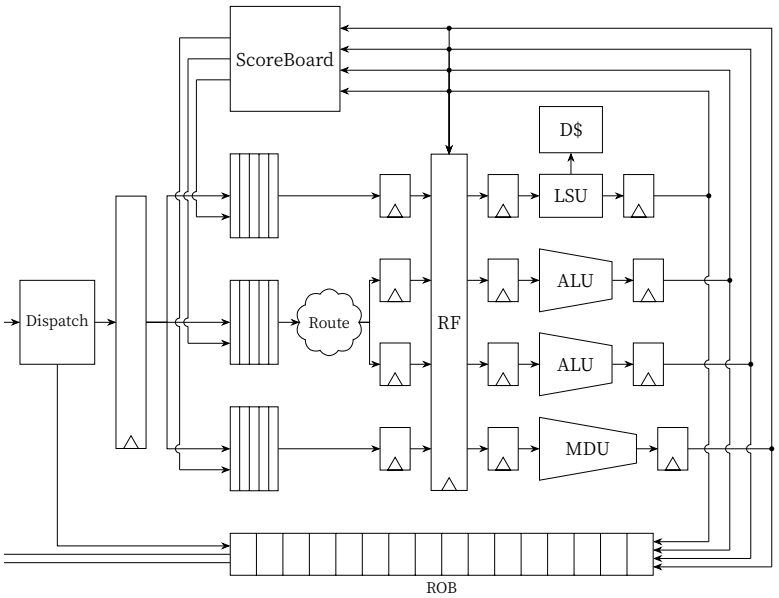


Scoreboard 为每个指令队列维护一个向量，指示操作数的状态。
写 Scoreboard 的时机：

1. 指令被仲裁电路选中，可以发射：此时写 Scoreboard 相当于唤醒可以从旁路网络获取操作数的指令
2. 指令写回结果的同时：此时写 Scoreboard 则告知那些不可从旁路网络获取操作数的指令，其操作数准备就绪

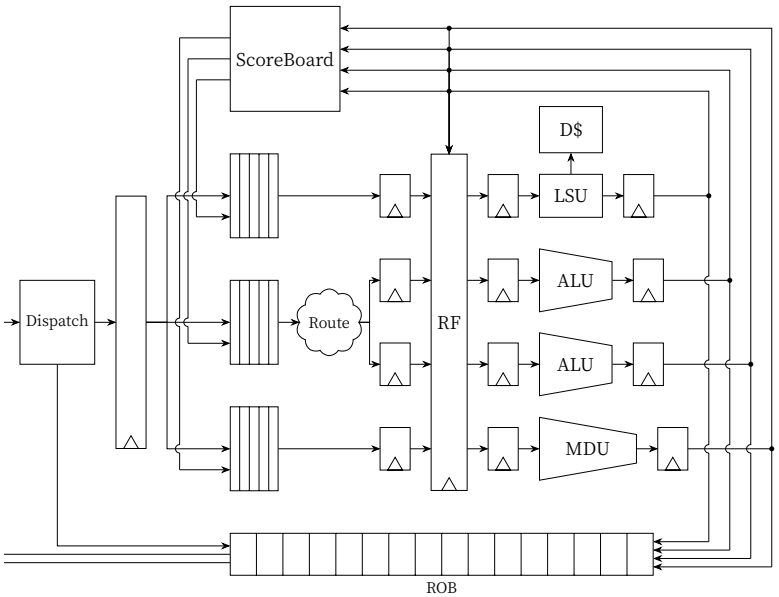
发射队列中的指令通过读 Scoreboard 获知操作数的状态，以设置自身的 Ready 位

发射与执行：仲裁



仲裁的原则是 Oldest-First，加以部分限制（例如 `Load/Store` 的相关性限制）

发射与执行：执行

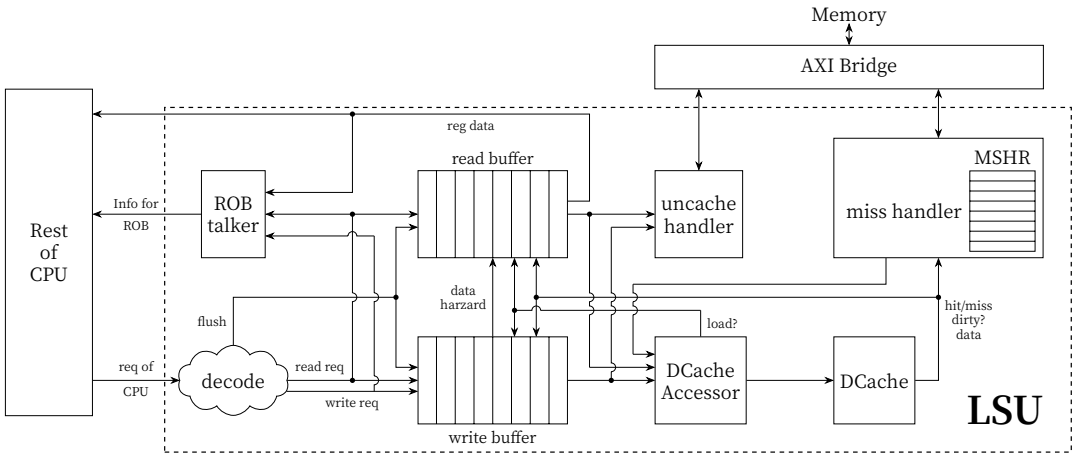


执行阶段和标量五级流水类似：

- 读 PRF（物理寄存器堆）
- 执行
- 写回

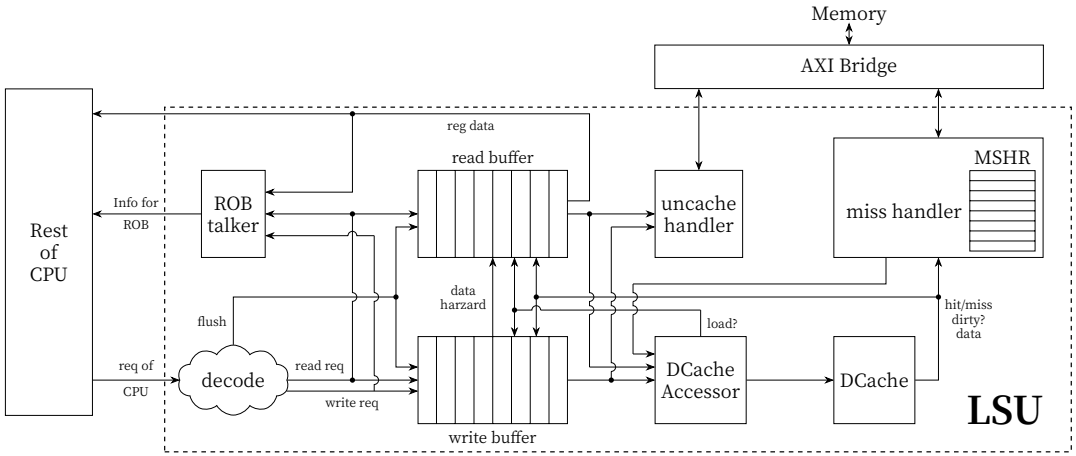
不同的功能单元，其流水线级数不尽相同。

发射与执行：非阻塞访存模块设计



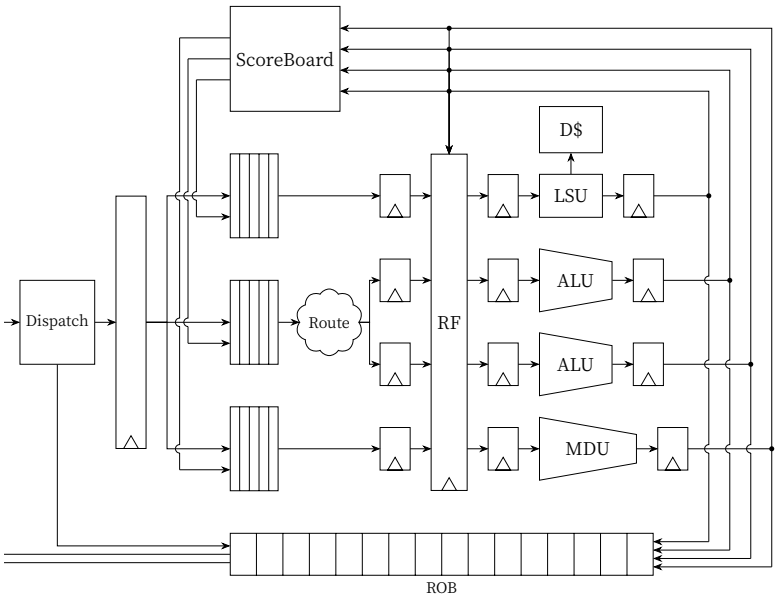
- 写缓存队列
  - 16 行循环队列
  - 顺序发射
- 读缓存队列
  - 16 行缓冲槽
  - cache 读乱序发射
  - 读后写依赖读请求锁定
  - cache 读写缓存数据预取
  - 一次 miss，多处标记

发射与执行：非阻塞访存模块设计



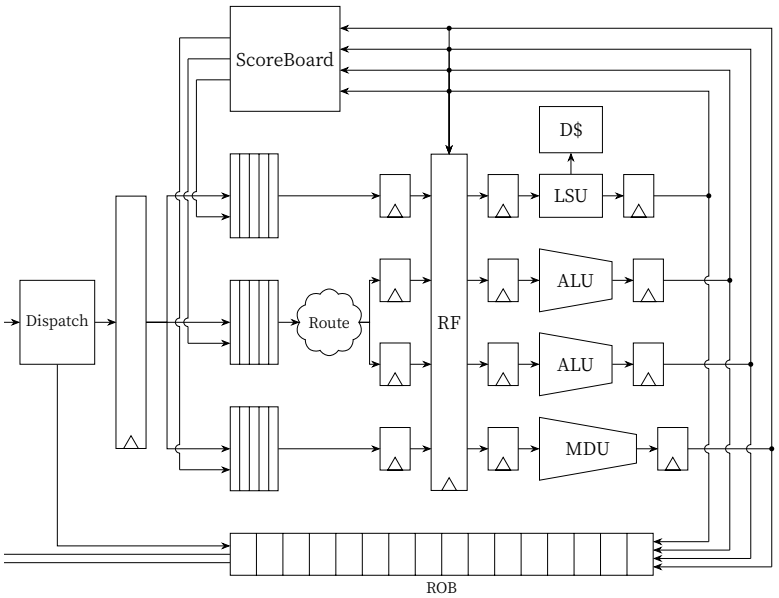
- 数据 cache
  - 两级流水化读写
  - 二路组相联，单路 8KiB
  - invalidate 指令支持
- 缺失处理单元
  - 16 表项 MSHR
  - 丢弃相同缺失请求
  - 脏写处理
- uncache 访存模块

提交与状态恢复: 重定序缓冲



- 用于顺序提交指令
- 32 项，每项两条指令，共 64 条
- 每个周期最多提交 2 条指令
- 跳转指令和延迟槽同时提交，其他指令能交则交

提交与状态恢复



- 每个周期最多提交 2 条指令
- 在指令提交时，对是否造成异常和分支预测失败进行判断
- 若发生异常和失败的预测，则刷新整条流水线，并
  - 恢复重命名映射表
  - 清空 Scoreboard
  - 恢复分支预测历史记录
- 对于外部中断：当且仅当有能够提交的指令时，才进行响应

目录

性能分析

- 部分测试程序上表现很好，发挥出了乱序执行的优势
- 在排序任务和访存负担较重的程序上表现极为糟糕
- 右表是在 88M 频率下的运行结果
- `crc32`/`select_sort` 表现极佳
- `bubble_sort`/`quick_sort` 表现糟糕
- 原因分析
  - 分支预测失败带来损失过大
  - `Load`/`Store` 流水线与算术指令流水线之间并未实现旁路

测试程序	得分	IPC 比值
crc32	73.0	41.3
select_sort	72.3	41.0
sha	60.1	34.1
stream_copy	51.8	29.4
bitcount	48.9	27.4
dhrystone	46.6	26.4
stringsearch	44.4	25.2
coremark	37.8	21.4
bubble_sort	33.7	19.1
quick_sort	33.2	18.8

PMON 的尝试

```
sp=8/00c000FREQ
RTC time invalid, reset to epoch.
FREI
DONE
DEVI
ENVI
MAPV
in envinit
nvram=bfc00000
NVRAM is invalid!
NVRAM@bfc00000
STDV
80100000:  memory between 82fff800-83000000  is already been allocated,heap is already above this point
SBDD
DINI
==gpio: gpio status 0
NETI
RTCL
====before configure
in configure
mainbus0 (root)
localbus0 at mainbus0
dmfe0 at localbus0: address 00:98:76:64:32:19
in if attach
loopdev0 at mainbus0out configure
====before init ps/2 kbd
devconfig done.
ifinit done.
domaininit done.
init_proc....
HSTI
SYMI
SBDE

 * PMON2000 Professional *
Configuration [FCR,EL,NET]
Version: PMON2000 2.1 (ls1b) #51: Sun Aug 16 18:53:46 PDT 2020 .
Supported loaders [srec, elf, bin]
Supported filesystems [mtd, net, fs/yaffs2, fat, fs, disk, socket, tty, ram]
This software may be redistributed under the BSD copyright.
Copyright 2000-2002, Opsycon AB, Sweden.
Copyright 2005, ICT CAS.
CPU unidentified @ 99.99 MHz / Bus @ 99.99 MHz
Memory size 128 MB (128 MB Low memory,   0 MB High memory) .
Primary Instruction cache size 0kb (2 line, 1 way)
Primary Data cache size 0kb (2 line, 1 way)

BEV1
BEV2
BEV3
BEV0
BEV in SR set to zero.

NAND DETE
NAND device: Manufacturer ID: 0xec, Chip ID: 0xf1 (Samsung NAND 128MiB 3,3V 8-bit)
NAND_ECC_NONE selected by board driver. This is not recommended !!
NANDFlash info:
erasesize       131072 B
writesize       2048 B
oobsize         64 B
PMON>
PMON>
```

目录

总结、反思与展望

走过的弯路，都是宝贵的财富

- 在时间极度紧张的情况下（20 天左右），使用 System Verilog 语言，完成了主流水线的构建、Cache 的添加、总线的连接，并通过了所有的测试
- 完成了一个真正意义上的乱序执行四发射处理器
- 在一定程度地牺牲了 IPC 的情况下，提升了主频
- 流水线功能基本正确，但性能差强人意
- 是一次有意义的探索
- 过深的流水线导致分支预测失败后，流水线启动时间过长，效率急剧降低
- 未实现完整的旁路网络，导致在某些访存密集型任务上效率低于常规的标量处理器

一些简单的思考

- 在这次设计中，我们发现，往往在设计前期难以对流水线的设计进行相应的性能评估，而只有在真正 RTL 编码完毕，进行性能仿真时，甚至综合、实现、上板后，才能得到相关的数据，此时再修改架构设计，为时已晚，成本过高。
- 是否有某种手段，能够在设计的前期，即真正的 RTL 实现之前，对架构的性能，特别是流水线的效率，进行时序精确级的、量化的评估，以做出相应的取舍？
- GEM5 等模拟器是否可以完成此类任务？在体系结构的研究中，一般采用何种手段？

谢谢！

项目代码及设计文档：
https://github.com/Superscalar-HIT-Core/SHIT-Core-NSCSCC2020

恳请批评指正！