NOOP概要设计

- 余子濠

乱序多发射的组原实验

- 大家都学习过乱序多发射
- 为什么目前为止没有高校有乱序多发射的组原实验?
  - 实验设计不scale
  - 步骤分解不明确
  - 基础设施不完善

紧耦合集中式控制是不scale的原因

- 单周期的译码模块
  - ID阶段译码出所有控制信号
    - is_j, is_mem, is_mem_write, is_syscall…
- 多周期的状态机
  - 状态机控制所有模块的行为
    - 40+条指令
- 流水线的全局阻塞
  - 一个模块阻塞, 所有模块都要跟着阻塞
  - 各个模块都有不同的阻塞需求
    - 取指/访存阻塞, 乘除法多周期出结果, load-use冒险, 中断/异常
- 规模小的时候很奏效, 规模大了难以维护

教科书上”目光短浅”的设计

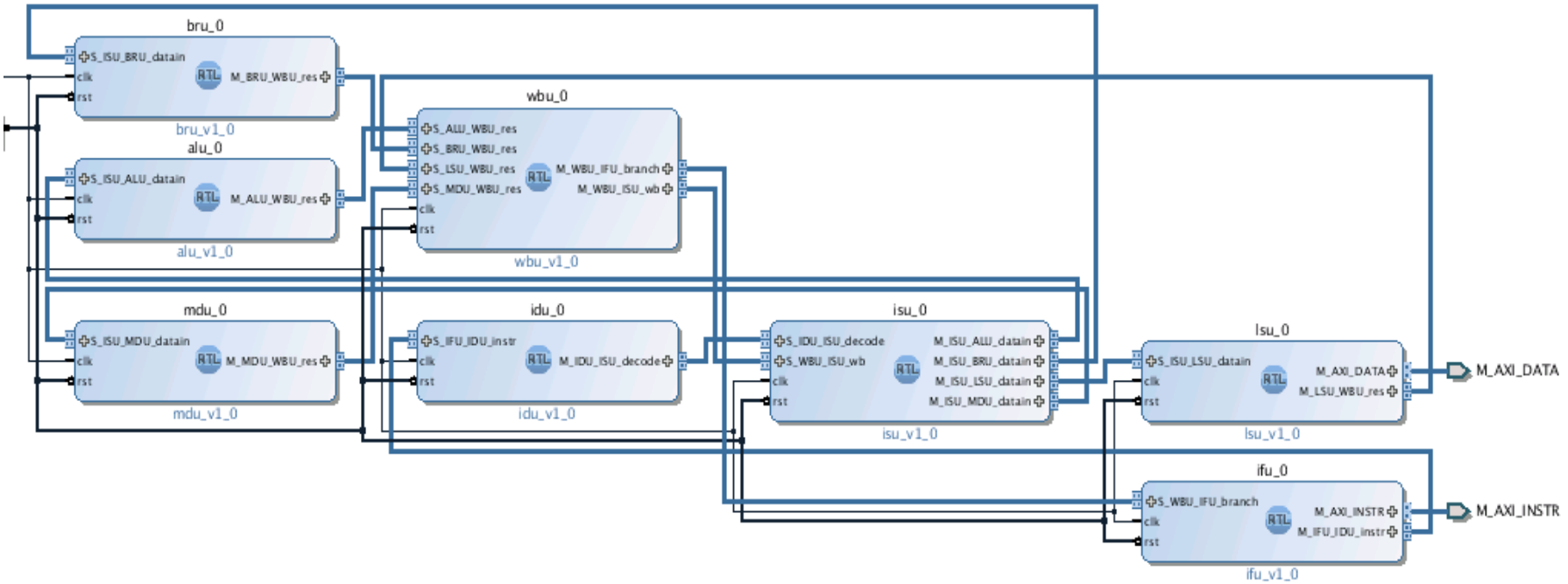
- MEM阶段
  - 额外的转发源
- 在ID阶段判断分支跳转结果
  - 额外的转发目标
- 延迟槽
  - 中断/异常, 多发射的取指/分支预测

```
lw $1, 100($2)
bne $1, $3, 400
add $1, $1, $1
```

- 乱序多发射处理器需要重新审视这些设计

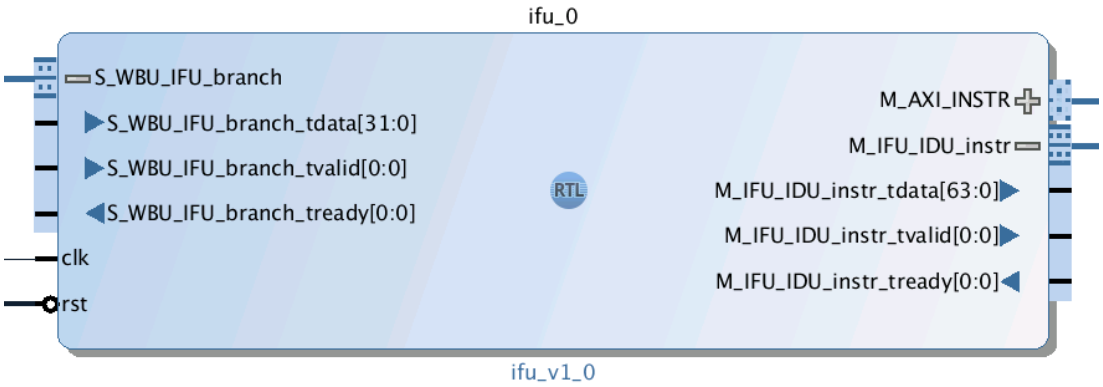
设计的第一个问题 - 松耦合

- 点对点的消息传递
- 采用AXI stream作为消息通道的协议



AXI-stream

- AXI-stream接口
  - tdata - 消息内容, master -> slave
    - 当前实现的各种信号的组合
  - tvalid - 消息有效的标志, master -> slave
  - tready - slave可接收消息的标志, slave -> master



消息通道命令约定

M - 通道的master端
S - 通道的slave端

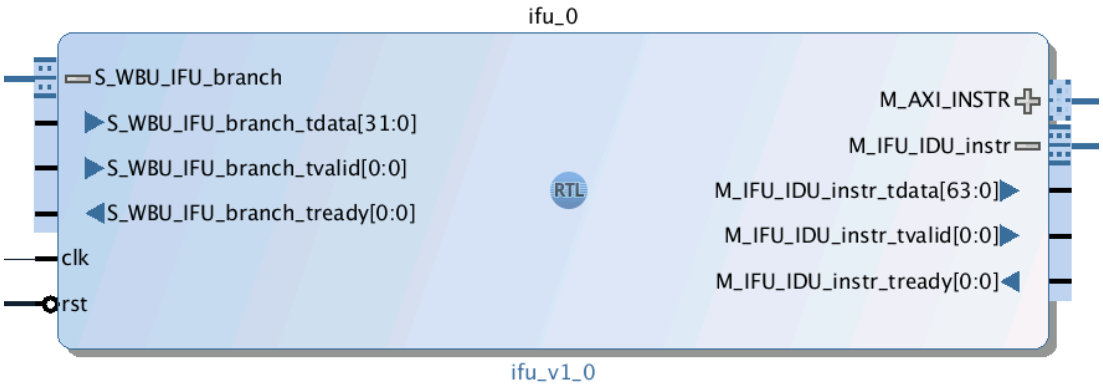
slave模块

AXIS信号

M_IFU_IDU_instr_tdata

master模块

消息通道标识



握手信号

- tvalid和tready组成一对握手信号
  - 解耦的关键
- tvalid表示模块内部当前产生有效消息
- tready表示下游模块能否接受该有效信息
- tvalid和tready同时有效 = 消息成功发送/接收

- 例子 - 乘除法需要多周期出数
  - M_MDU_WBU_res_tvalid = 1’b0
  - S_IDU_MDU_datain_tready = 1’b0

通过握手信号实现各种微结构

- 单周期
  - 除FU外各消息通道的tvalid恒为1, FU的tvalid取决于译码情况
  - 无视tready, 认为下游模块总能接收消息
- 带阻塞的单周期/多周期
  - IFU收到WBU的完成信号后再取下一条指令
- 流水线
  - IFU不等WBU的完成信号, 一直取指
  - 阻塞 = tready置为0 -> 告诉上游模块我不能接受新消息
  - 冲刷 = tvalid置为0 -> 告诉下游模块我没有消息要发给你

分布式控制是乱序多发射的必要条件

- 各个模块几乎都有队列/buffer来支撑乱序执行的功能
  - IQ, rename table, LSQ, ROB
- 模块可以一直工作, 直到队列/buffer满/空为止
  - 上游是下游的生产者, 下游是上游的消费者
- 阻塞与冲刷随时会出现

- 没有握手信号, 几乎不可能实现乱序执行

设计的第二个问题 - 步骤分解

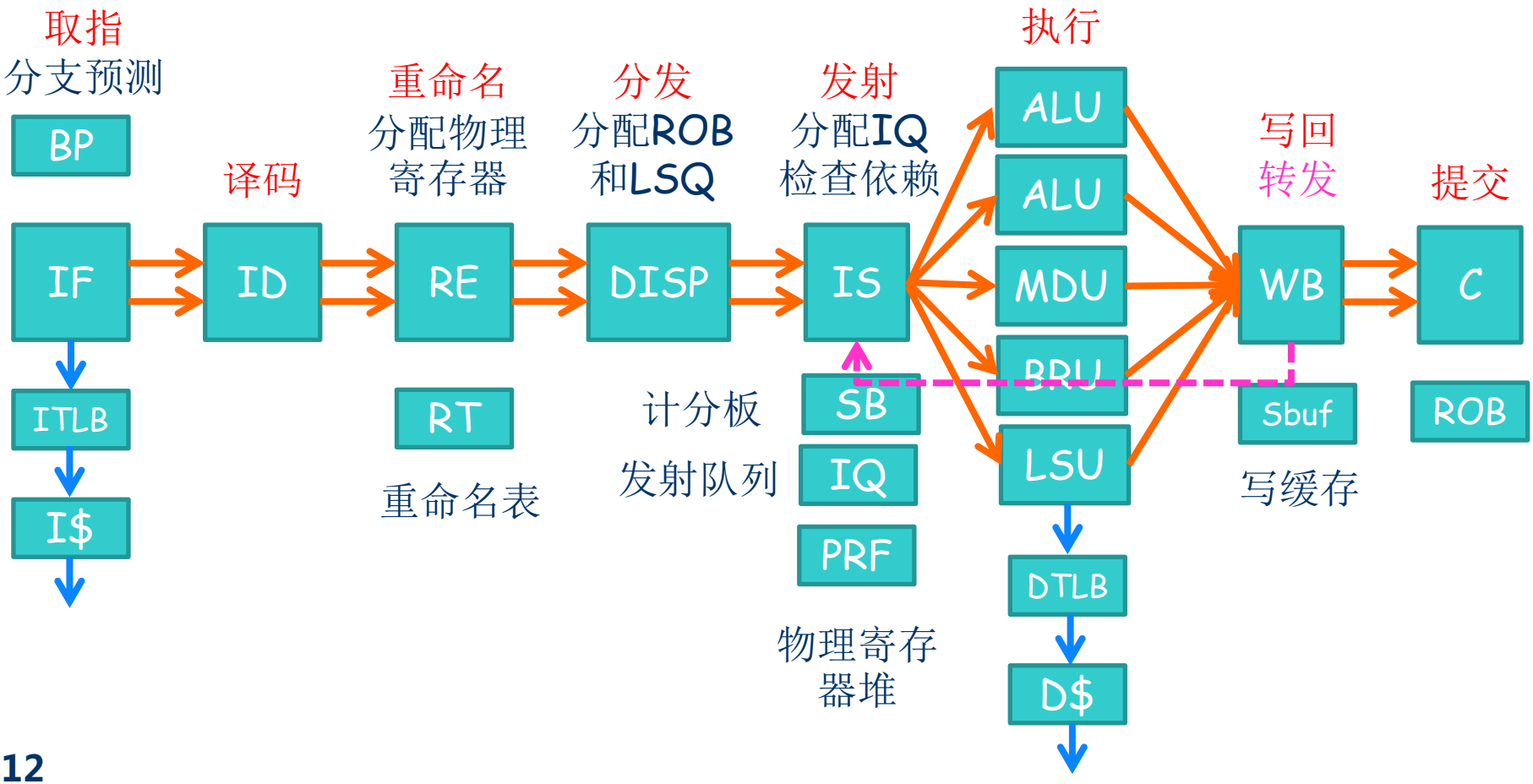
- 那么多模块/功能, 要先实现哪些?
- 选择的原则的是什么?

- 基本原则 - 实现尽量少的模块, 使得设计可以尽快达到下一个可以验证的阶段

- 我们可以从最终设计出发, 反推得到步骤的分解
  - 删除尽量少的模块达到一个更简单的, 可以验证的阶段

最终架构 - 双发射的IO2I

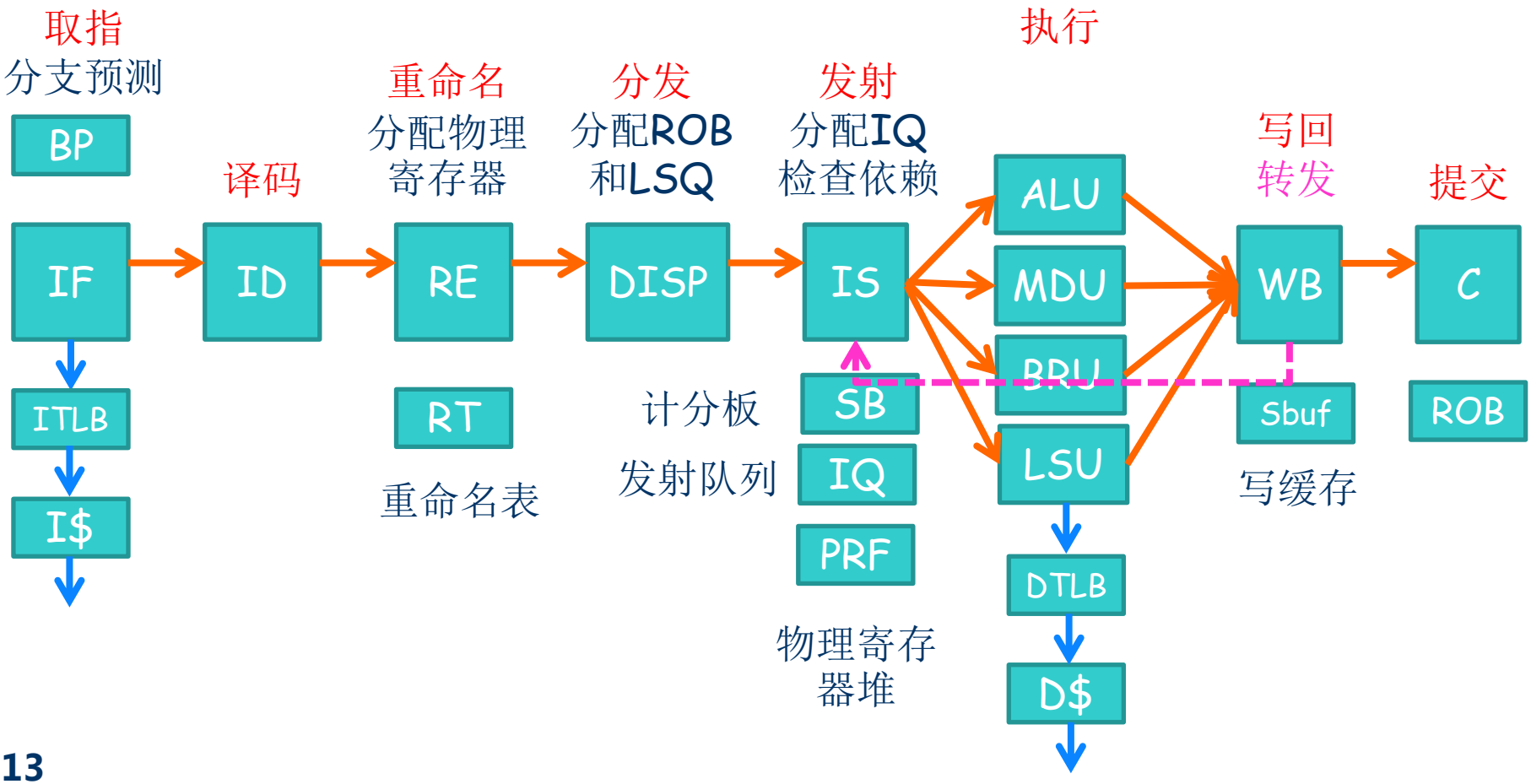
取指	发射	执行	提交	
I	I	I	I	I4
I	I	O	I	I2OI
I	O	O	I	IO2I



单发射的IO2I

取指	发射	执行	提交	
I	I	I	I	I4
I	I	O	I	I2OI
I	O	O	I	IO2I

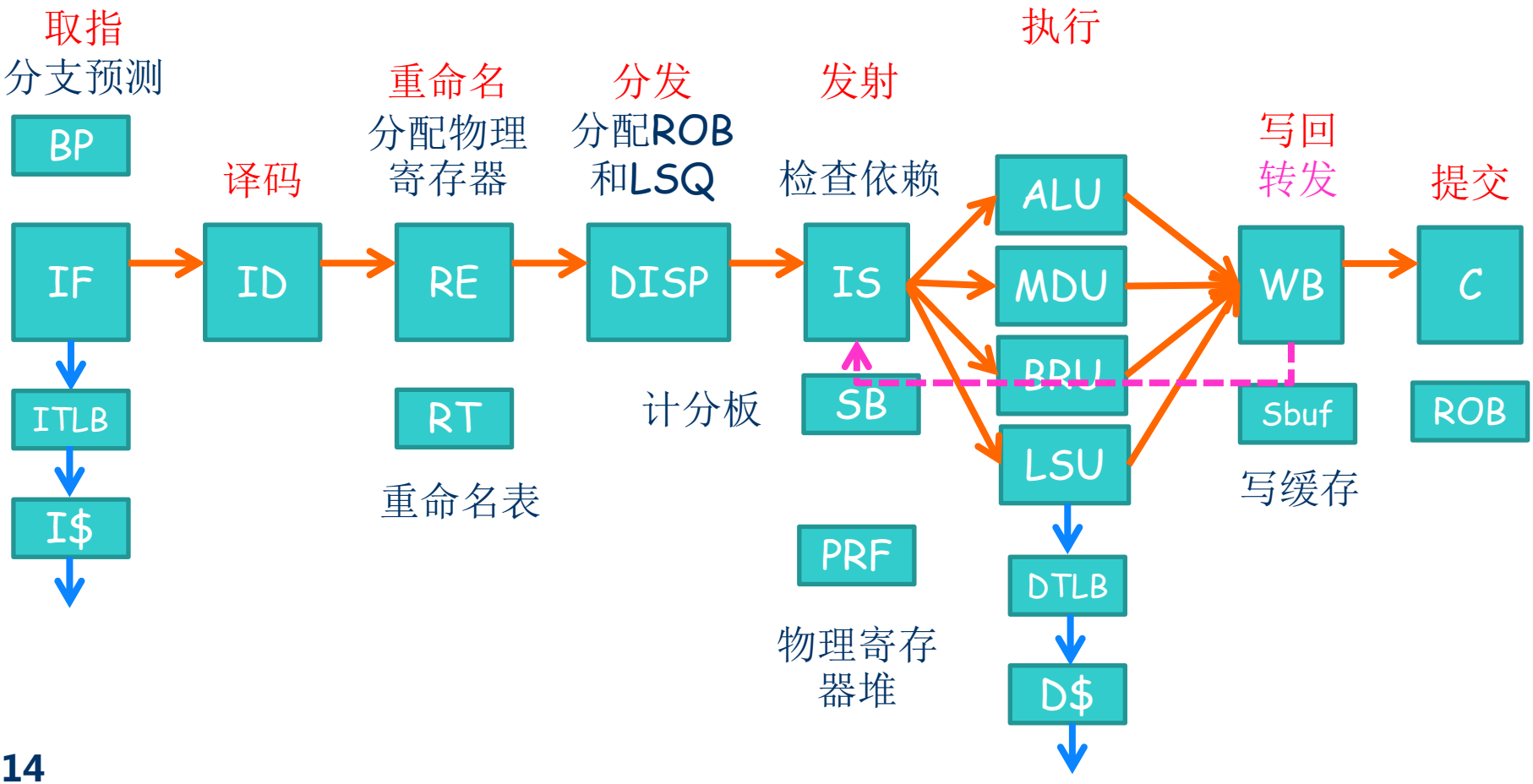
各种宽度减为1
去掉1个ALU



I2OI

取指	发射	执行	提交	
I	I	I	I	I4
I	I	O	I	I2OI
I	O	O	I	IO2I

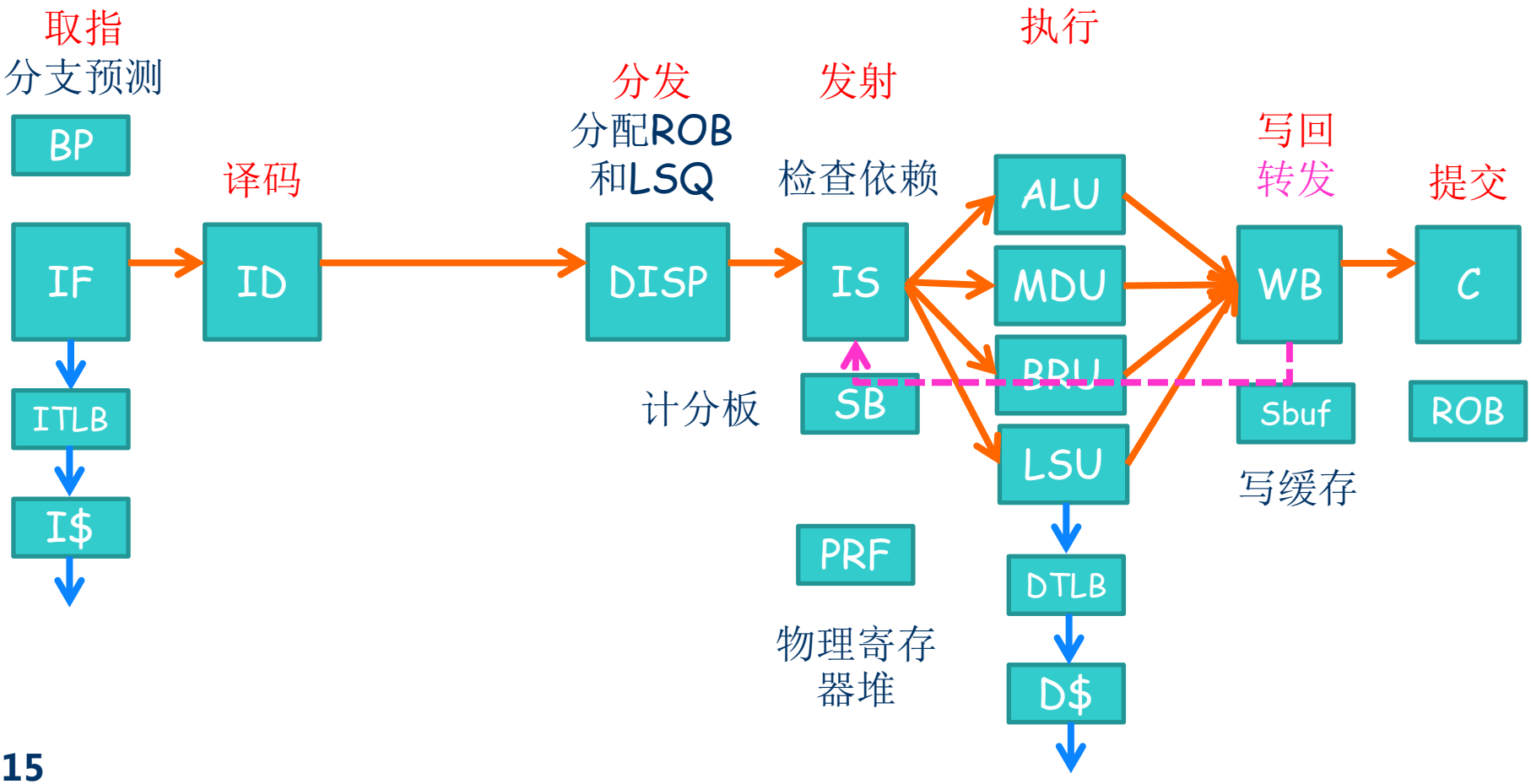
去掉IQ，顺序发射乱序执行



不带rename的I2OI

取指	发射	执行	提交	
I	I	I	I	I4
I	I	O	I	I2OI
I	O	O	I	IO2I

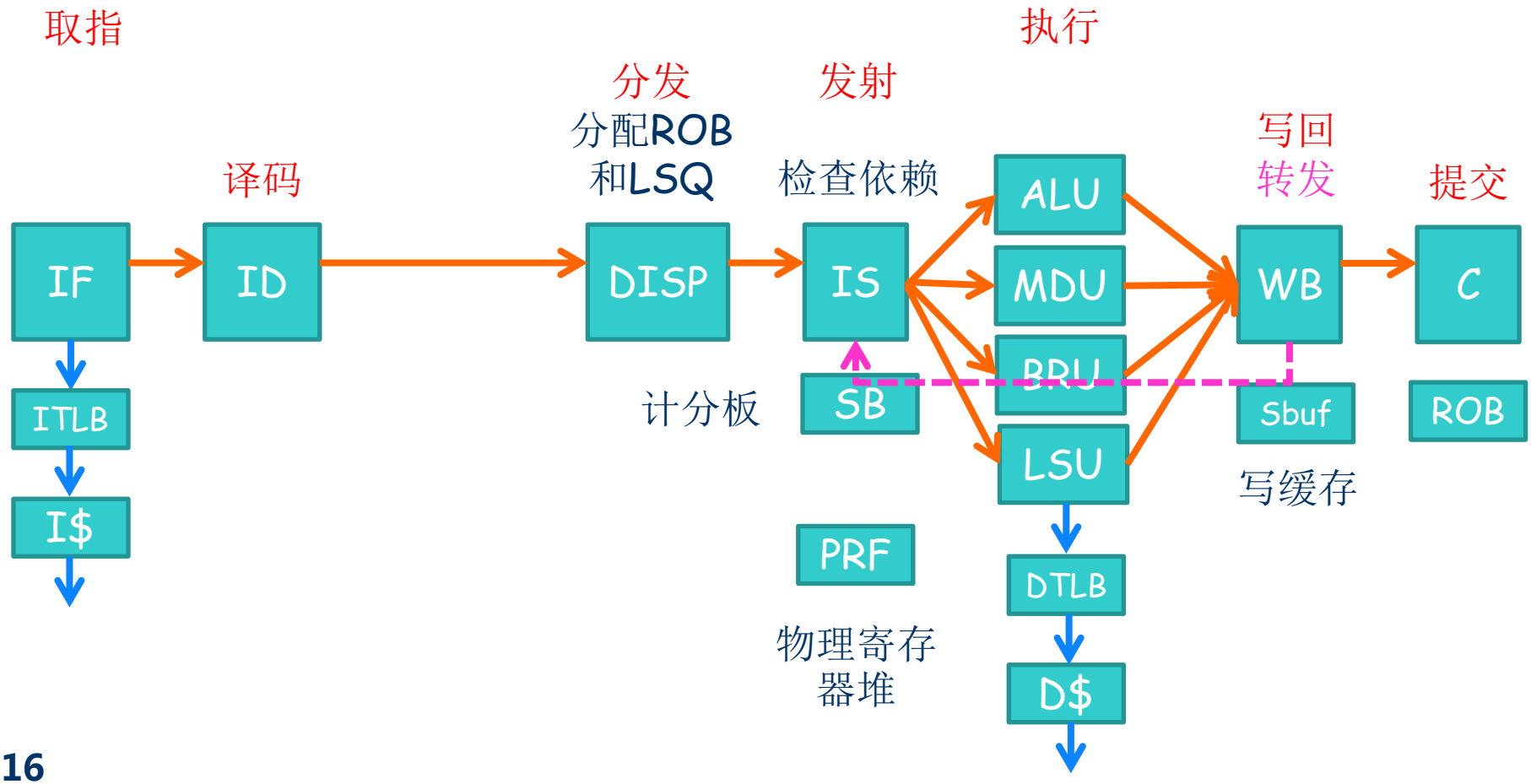
去掉rename阶段，ARF = PRF，SB还需检测WAR和WAW



不带分支预测的I2OI

取指	发射	执行	提交	
I	I	I	I	I4
I	I	O	I	I2OI
I	O	O	I	IO2I

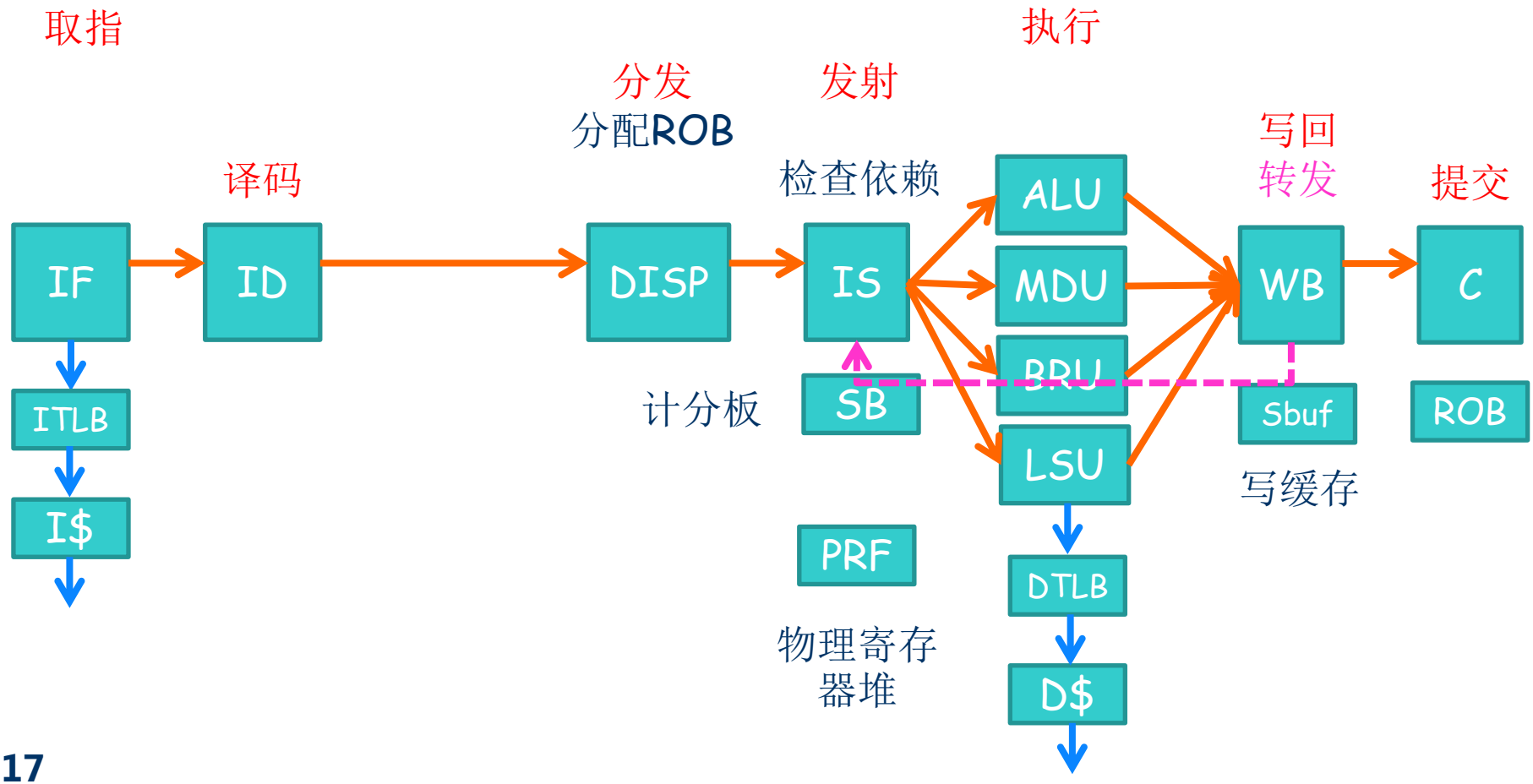
去掉BP和ROB的相关功能(标志, 回滚), 遇到分支就阻塞



LSQ/Sbuf只有一项的I2OI

取指	发射	执行	提交	
I	I	I	I	I4
I	I	O	I	I2OI
I	O	O	I	IO2I

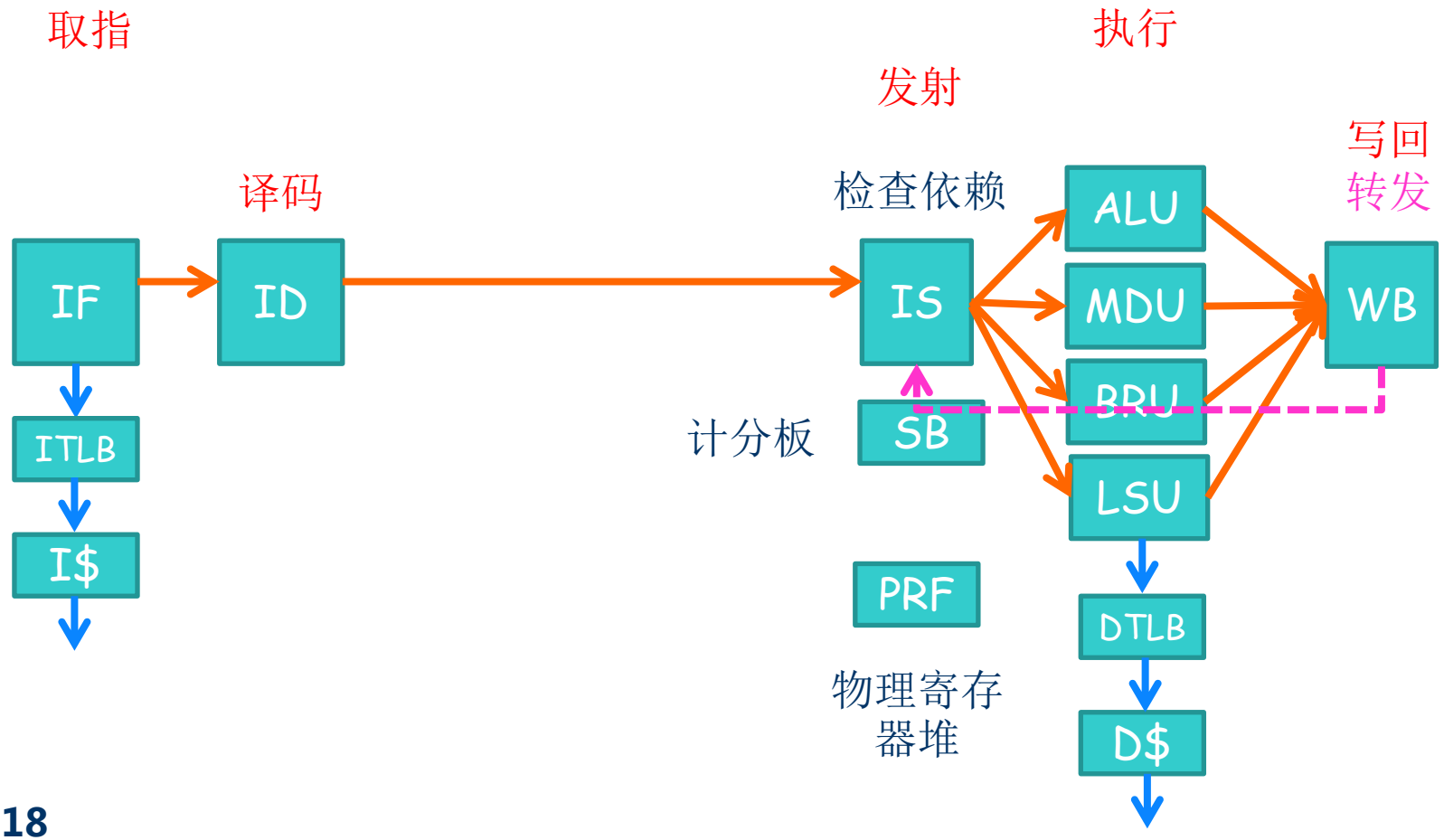
只有一项，无需分配



I4

取指	发射	执行	提交	
I	I	I	I	I4
I	I	O	I	I2OI
I	O	O	I	IO2I

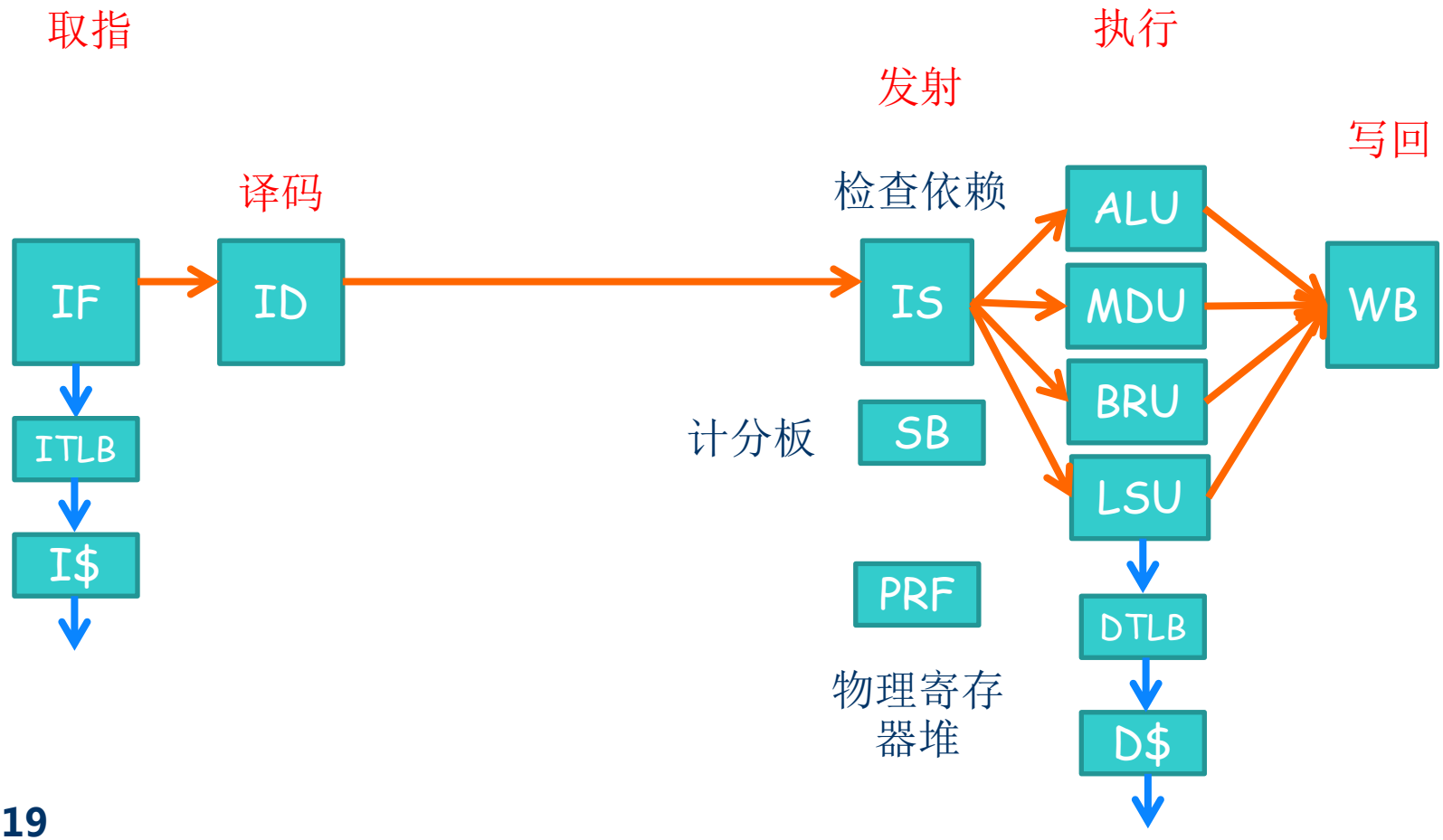
去掉DISP和C阶段，去掉LSQ，ROB，Sbuf；SB不需要检测WAW和WAR



不带转发的I4

取指	发射	执行	提交	
I	I	I	I	I4
I	I	O	I	I2OI
I	O	O	I	IO2I

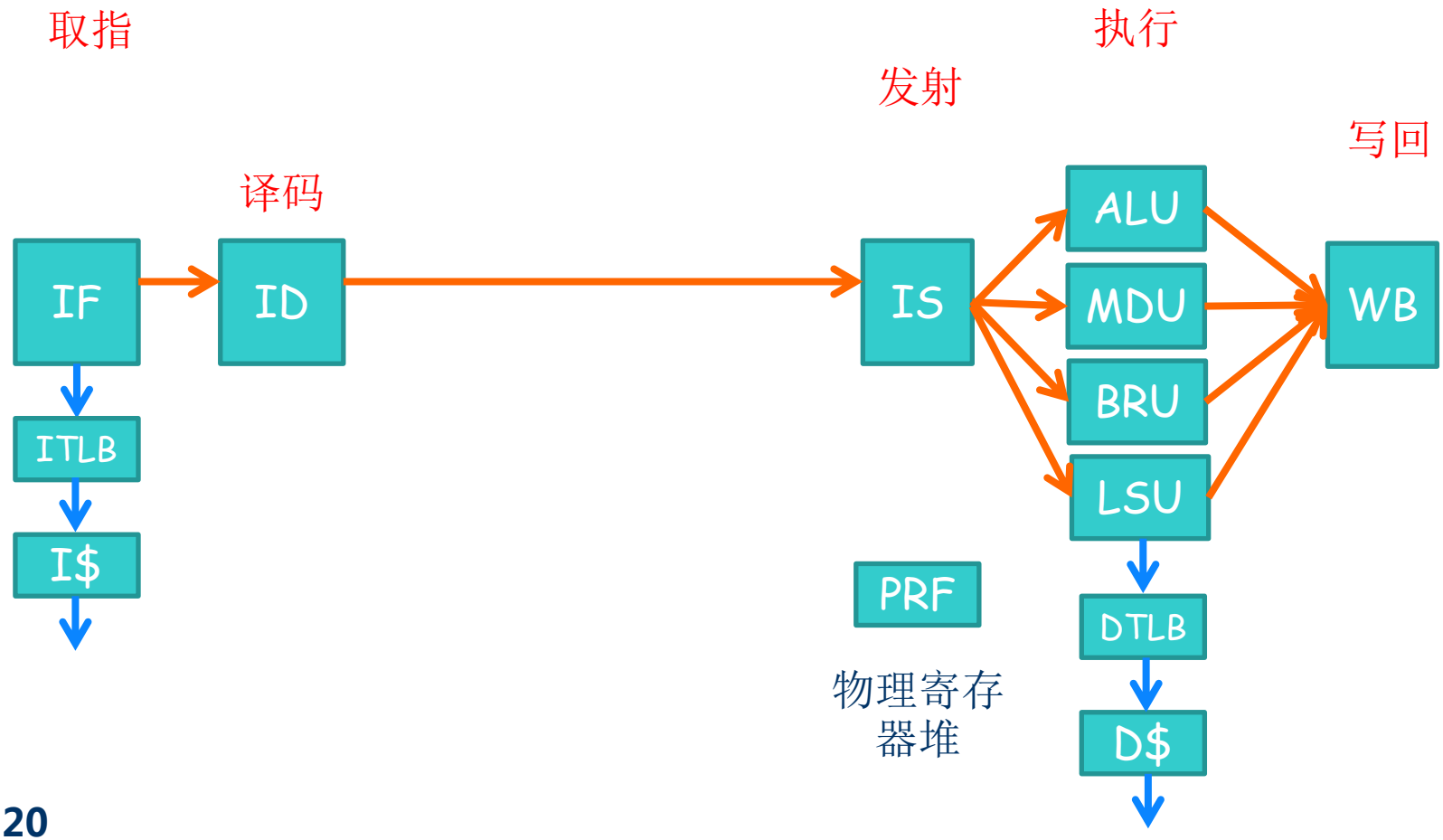
去掉转发，寄存器写回
PRF才发射RAW的指令



带阻塞的单周期

取指	发射	执行	提交	
I	I	I	I	I4
I	I	O	I	I2OI
I	O	O	I	IO2I

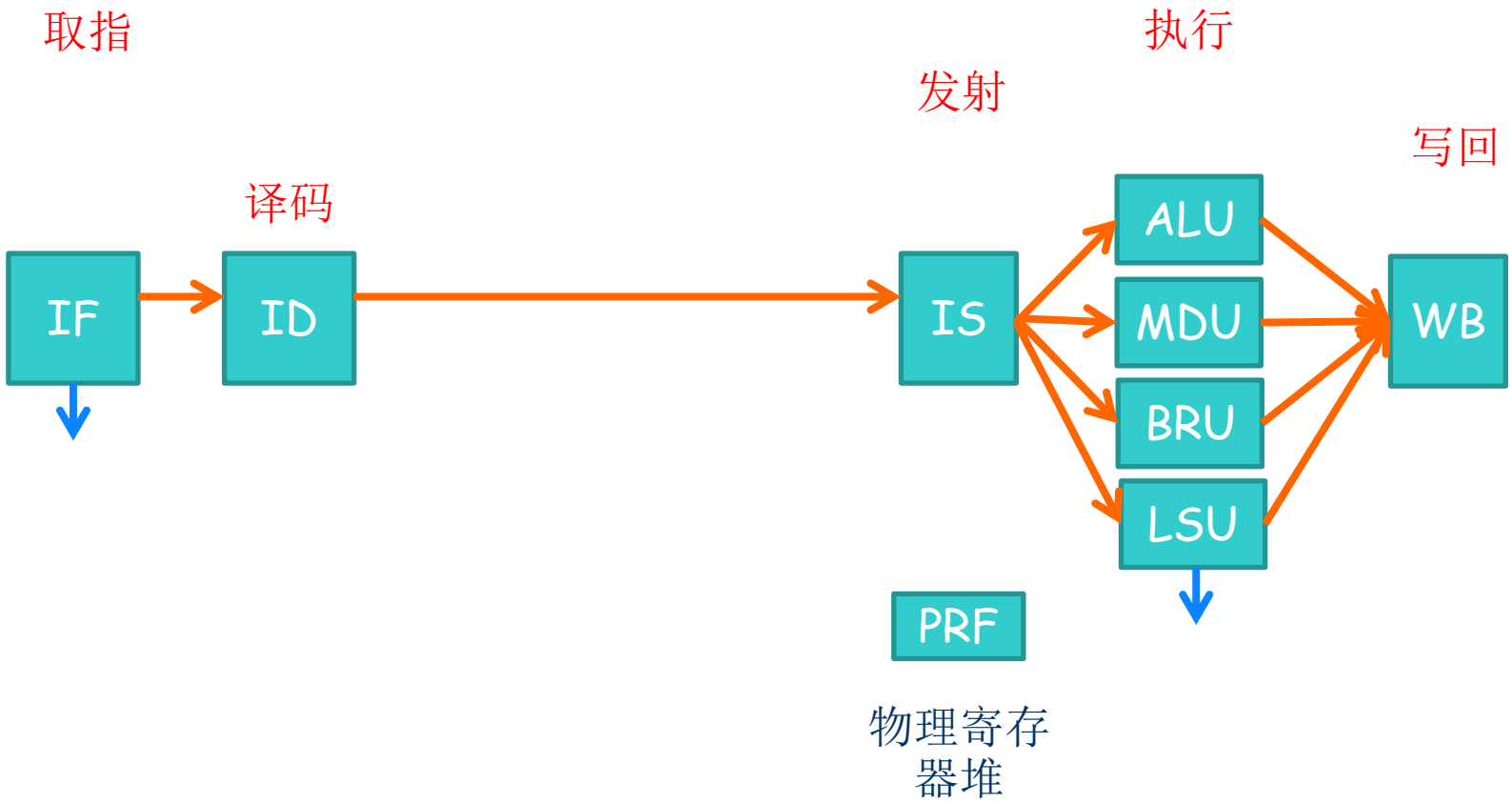
去掉SB



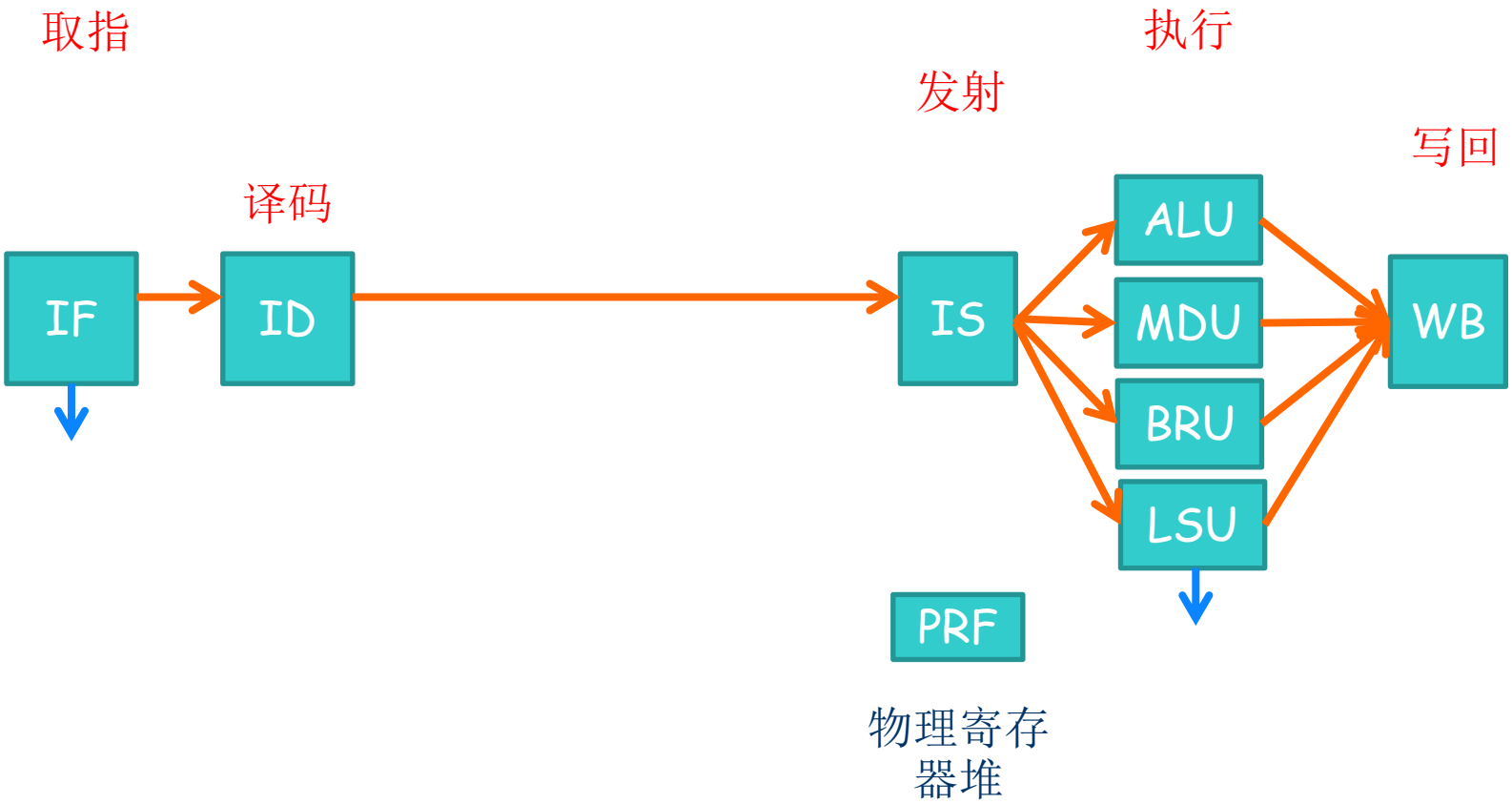
纯单周期

取指	发射	执行	提交	
I	I	I	I	I4
I	I	O	I	I2OI
I	O	O	I	IO2I

不支持$和TLB，只能运行在LUTRAM上



从零开始的第一步



重新审视教科书上的设计

- 增加IS阶段
- 去掉MEM阶段, 其功能移到EX阶段的LSU中
- ID阶段只需要译码出IS需要使用的信号
  - B_src, ext_type, rd_sel
  - 其它信号(如is_j, is_load)包含在fu_type和fu_op中
- 统一在EX阶段的BRU中判断跳转结果
  - 跳转目标在WB/C阶段反馈给IF
  - 性能损失由BP弥补
  - 最后有时间再优化
- 去掉延迟槽
- 只在WB和IS之间实现转发
  - I2OI就可以实现最大限度的转发
- 通过SB记录/检测数据依赖

设计的第三个问题 - 基础设施

- 如何快速开发?
  - 用描述文件描述接口/定义/功能
  - 用脚本生成重复代码
  - 利用编译器/综合器的优化
  - 利用工具和框架(如vivado中的block design)

- 如何(快速)测试实现是否正确?
  - 如何在verilog中实现assert?
  - 工具链 -> benchmark
  - 随机指令发生器
  - 输出到外设(串口, 显示器)
  - 一键仿真平台/一键上板

设计的第三个问题 - 基础设施(2)

- 不正确的时候如何(快速)调试?
  - cross checking
  - 断点/监视点
  - 如何在zynq上实现一个monitor?
  - 如何冻结panic现场?
- 如何在正确实现中找到性能瓶颈?
  - performance counter
- 如何记录/维护开发过程
  - github, issue, pull request

设计总结

- 松耦合
  - 直接决定设计的尽头
- 步骤分解
  - 可行地一步步迈向目标
  - 体验项目成长的过程
  - 第一个能完整跑起来的baseline是最最最重要的
- 基础设施
  - 快速地开发/测试/调试

- 好的设计能让细节更可控 != 没有繁琐的细节
  - 所有细节和探索都应该在github上记录, 这是做实验最宝贵的财富

做事的方法

- must have - 不做项目就无法/很难推进的事情
  - 永远优先解决
  - 举例 - 第一个能跑起来的baseline
- nice to have - 做了有好处, 不做项目也可以推进的事情
  - 随着项目的推进转变为must have再做
  - 举例 - 分支预测/$, 各种目前碰不到的新指令
- 其它 - 做了对项目也没好处的事情
  - 永远不要做
  - 举例 - 自己写flash/spi

- 时刻记住做系统实验的目标
  - (1) 实现一个能正确跑程序的计算机系统
  - (2) 在(1)的基础上让计算机尽可能跑得快

超越龙芯也许不是梦

- NOOP是第一款面向教学的乱序多发射处理器
  - BOOM性能强大, 但对白手起家的我们帮助很有限
- 我们有强大的顾问
  - jyy/yzh把握软/硬件的设计和大方向
    - 已经能超越绝大部分学校的队伍(清北复交, 中科大?)
  - zyy/whz熟悉前沿的研究
    - 有多少队伍知道DRRIP一用平均性能就能直接提高20%-30%?
- 龙芯开源计划
  - GS232是5级流水乱序双发射
  - 我们也许能超越它
    - 设计更易于教学/更早开源/性能更好

讨论