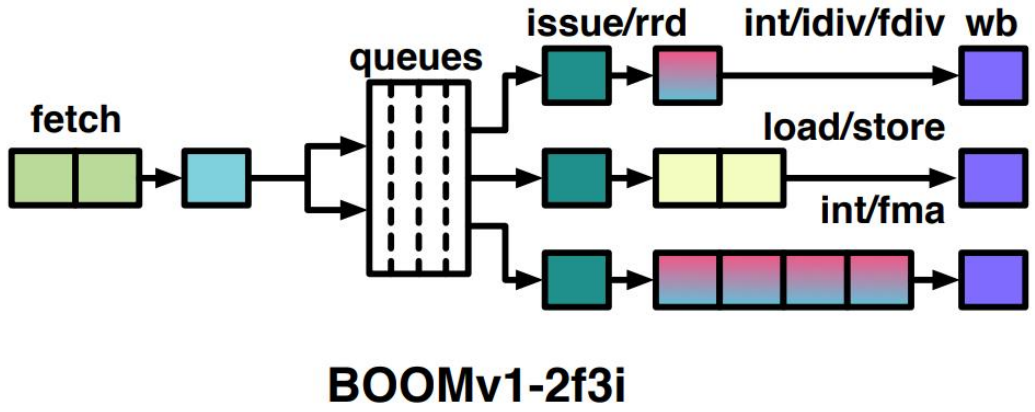
Overview

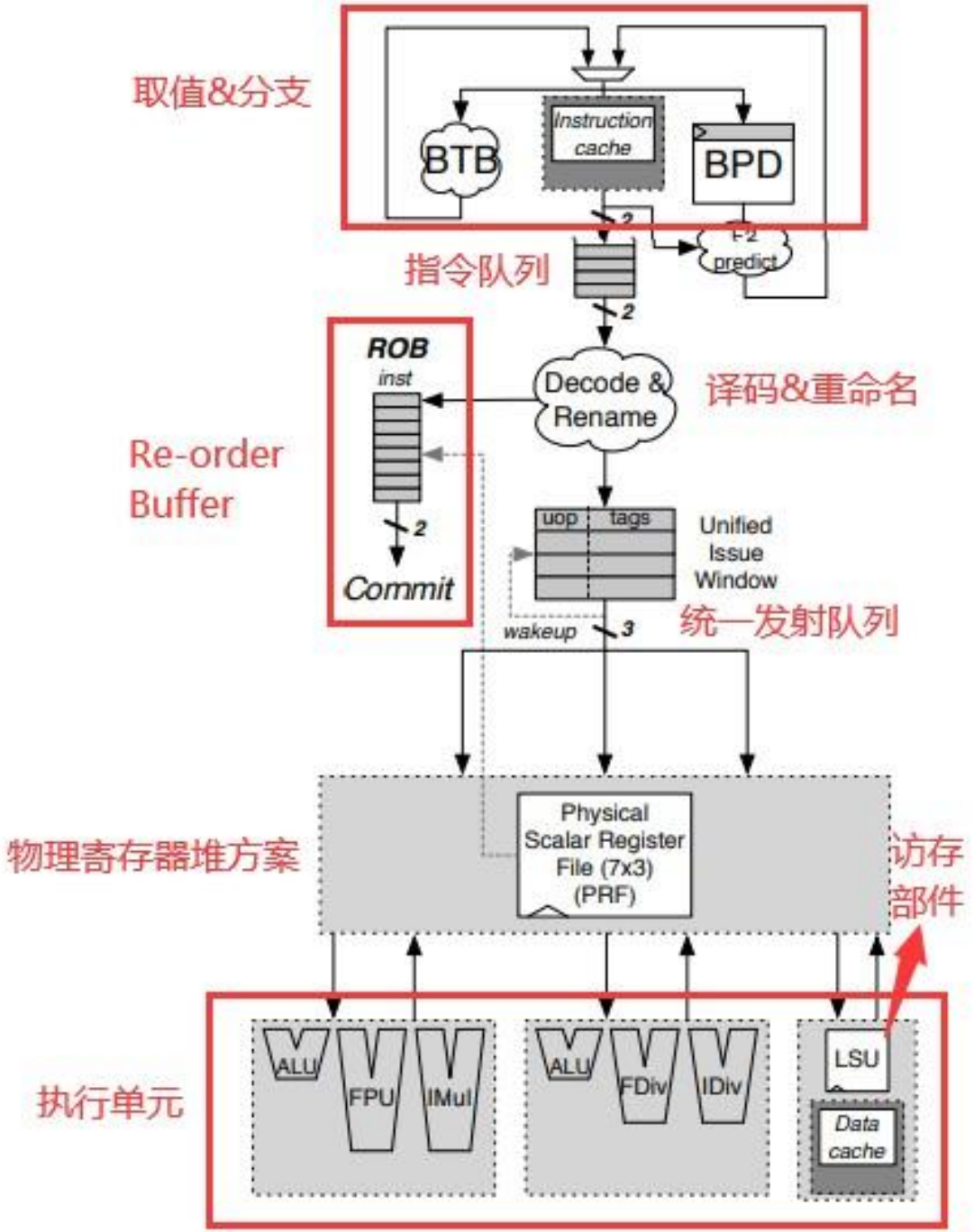
- 现代处理器设计
- 乱序执行
  - 只有真相关(RAW)才能阻塞执行
  - 假相关(WAW WAR)需要被消除
  - 引入ROB，从外界看仍是顺序执行
- 多发射
  - 一个周期同时执行多条指令，突破传统的限制
- 分支预测
  - 解决条件冲突，保证流水线不断裂
  - 速度越快，分支预测越重要

Overview

- BOOM v1 by UC.Berkeley
- Fetch - 2
- Decode/rename - 2
- Issue/register-read - 3
- Execute
- Memory
- Writeback - 3

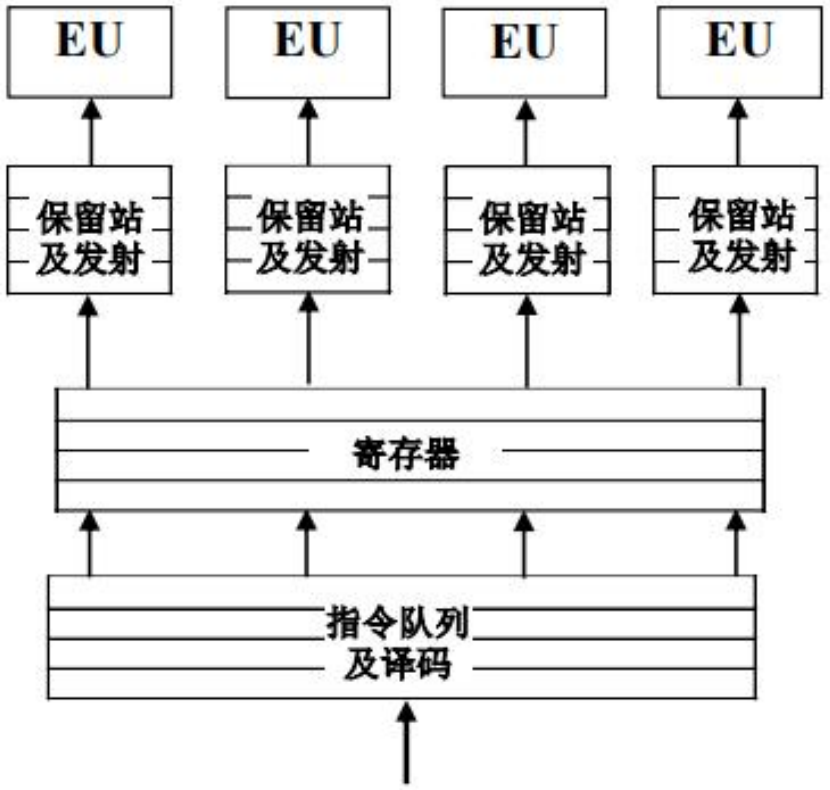


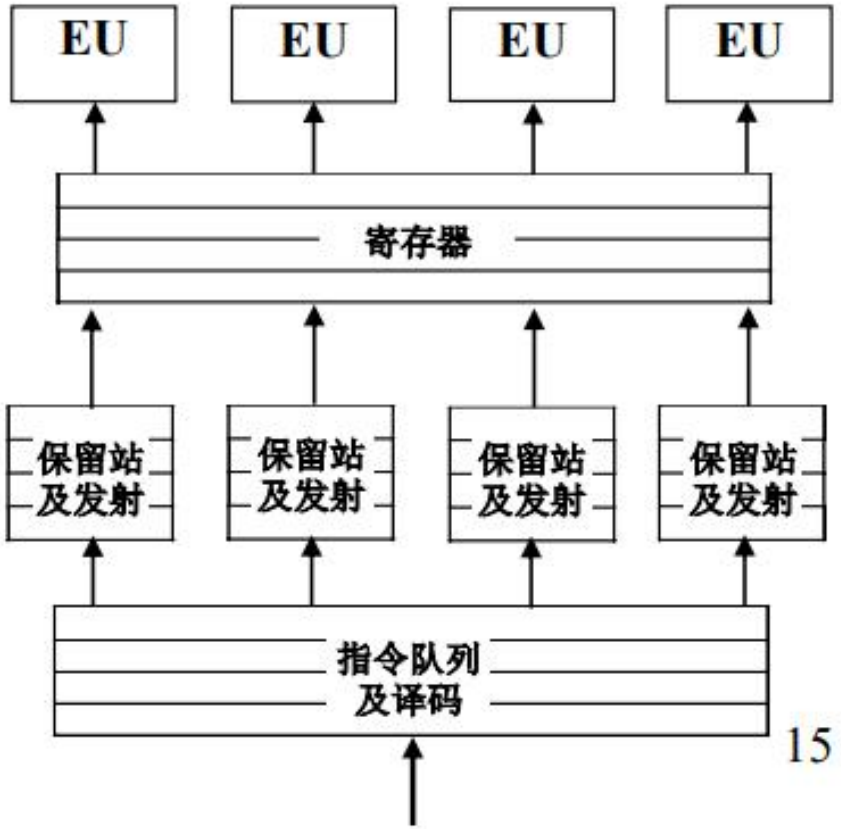
BOOMv1-2f3i



Out of Order

- 读取寄存器的时机 – 发射队列中记录“号”还是记录“值”





Register Renaming

- 记录一个号，到时候按照号找需要的值
- 重命名作用
  - WAR WAW 冲突消除
  - 异常恢复(中断 分支错误)
- 重命名思路
  - 写的寄存器重命名
  - 读的寄存器查表
- 数存哪里?
  - ROB
  - Physical Register File
  - Others

ROB

- 将逻辑寄存器号转换为寄存器号 或者ROB号
- 需要判断值在ROB中还是寄存器堆中
- 好实现，但浪费面积

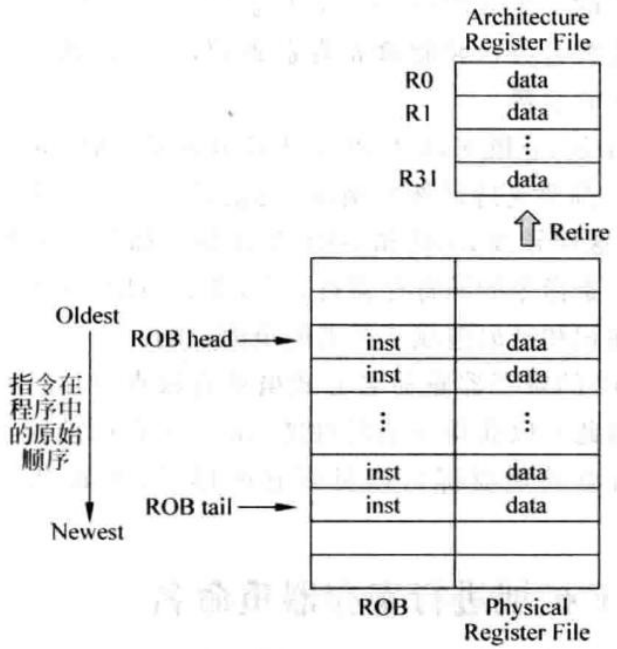


图 7.4 将 PRF 和 ROB 集成在一起，进行寄存器重命名

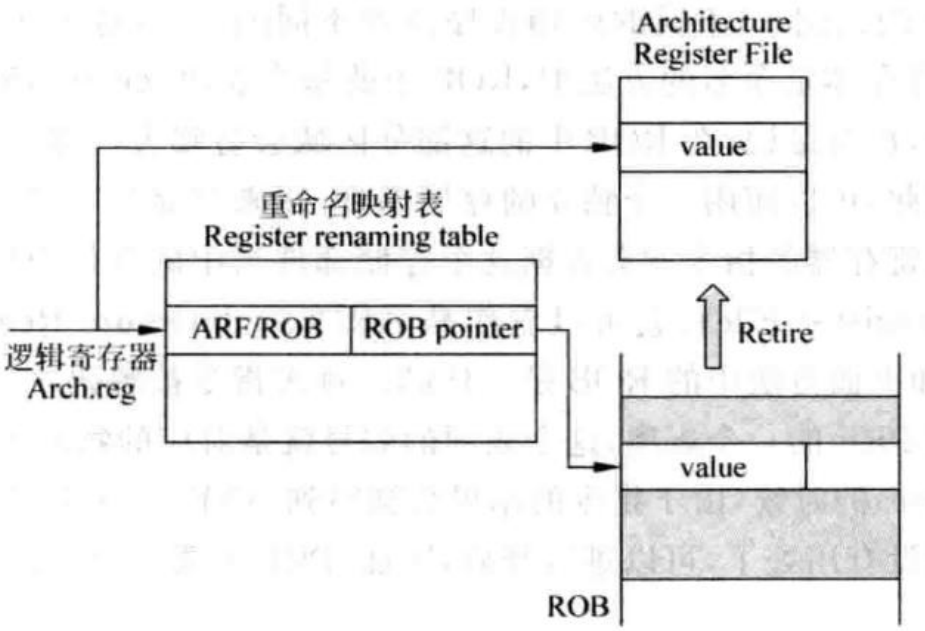


图 7.5 基于 ROB 进行寄存器重命名

Physical Register File

- 用一个大的寄存器堆记录数据
- 每个从大的寄存器堆中找一个空的寄存器号来记录写的数据
- 读的时候需要映射到大寄存器堆中一个寄存器号

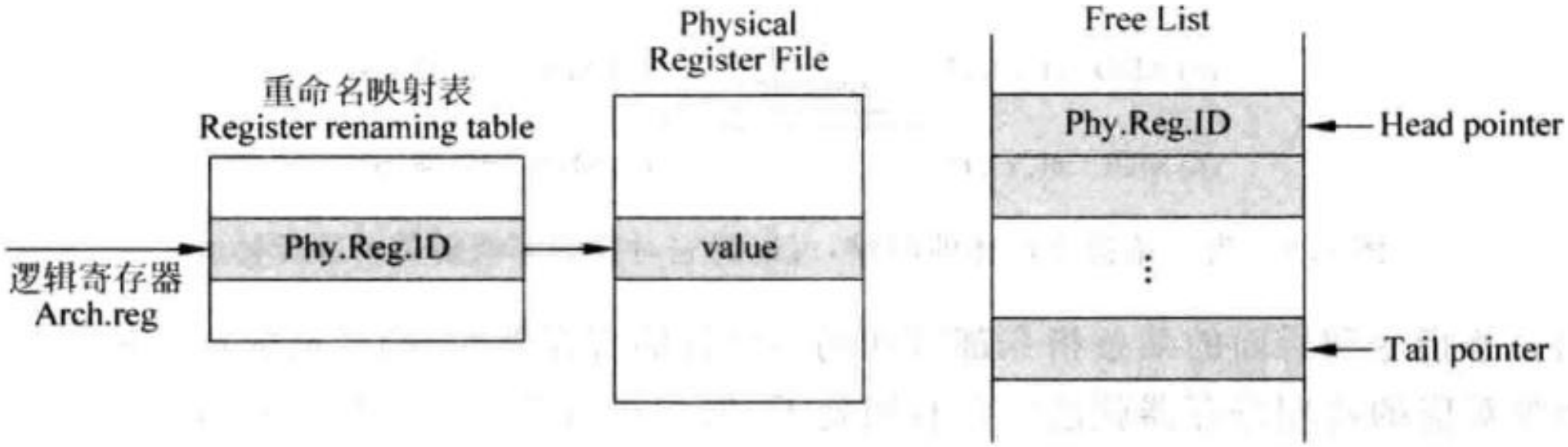


图 7.7　使用统一的 PRF 进行寄存器重命名

Register Alias Table

- RAT – 重命名表
- 有两种实现方式
  - SRAM
  - CAM
- Checkpoints用于恢复现场
- 此外ROB中也要记录即将Free的寄存器号

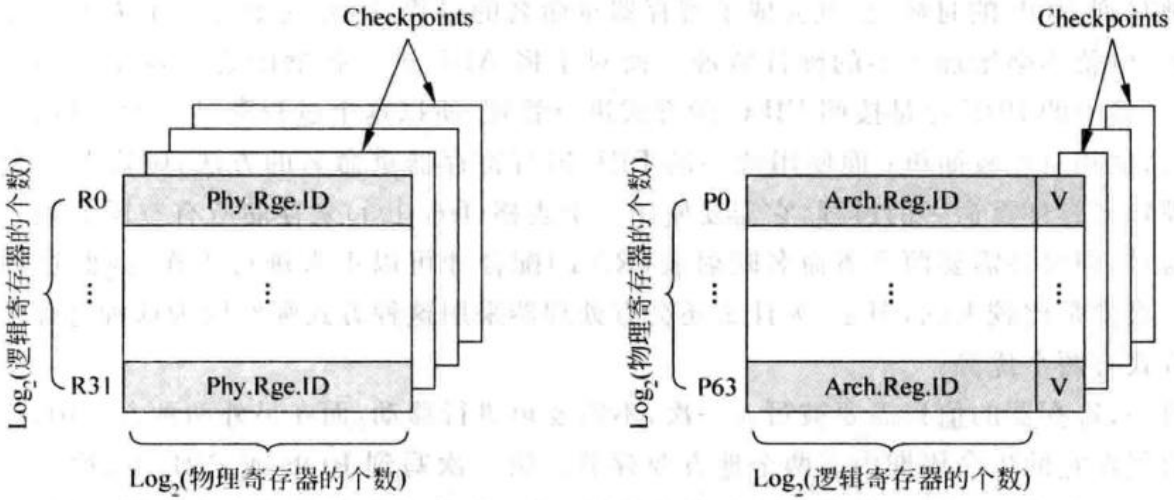


图 7.9 RAT 的两种物理实现方法

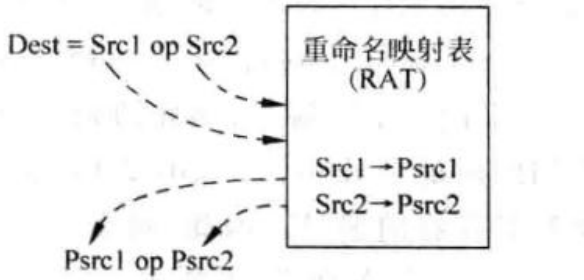


图 7.13 找到两个源寄存器对应的物理寄存器

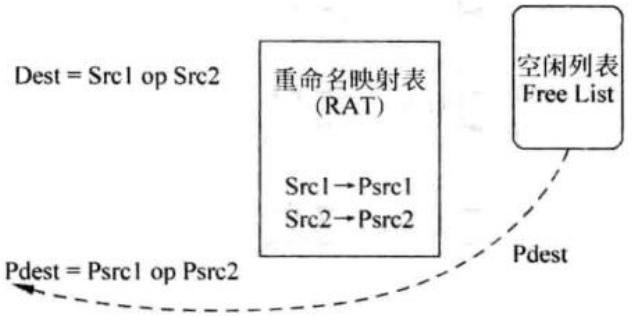


图 7.14 为指令的目的寄存器指定一个物理寄存器

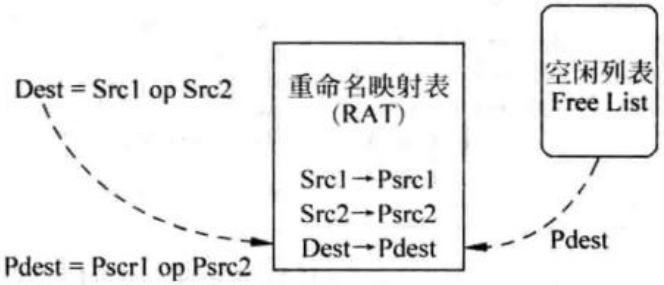


图 7.15 将新的映射关系写到 RAT 中

Superscalar Renaming

多发射下RAW问题

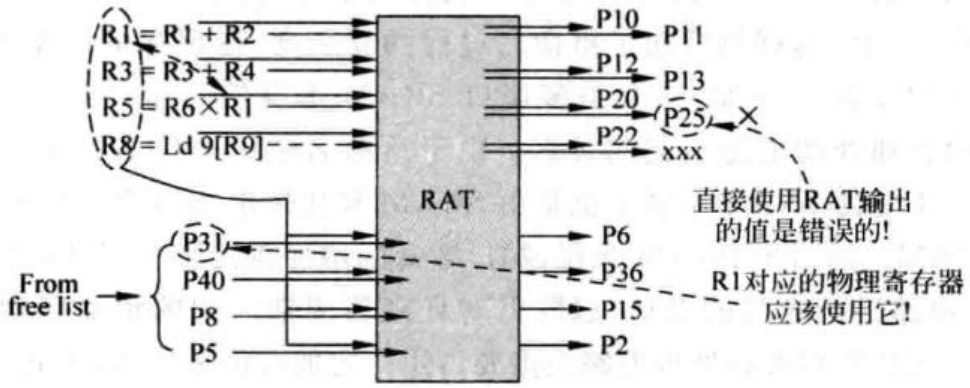


图 7.19 当四条指令之间存在 RAW 相关性时，遇到的问题

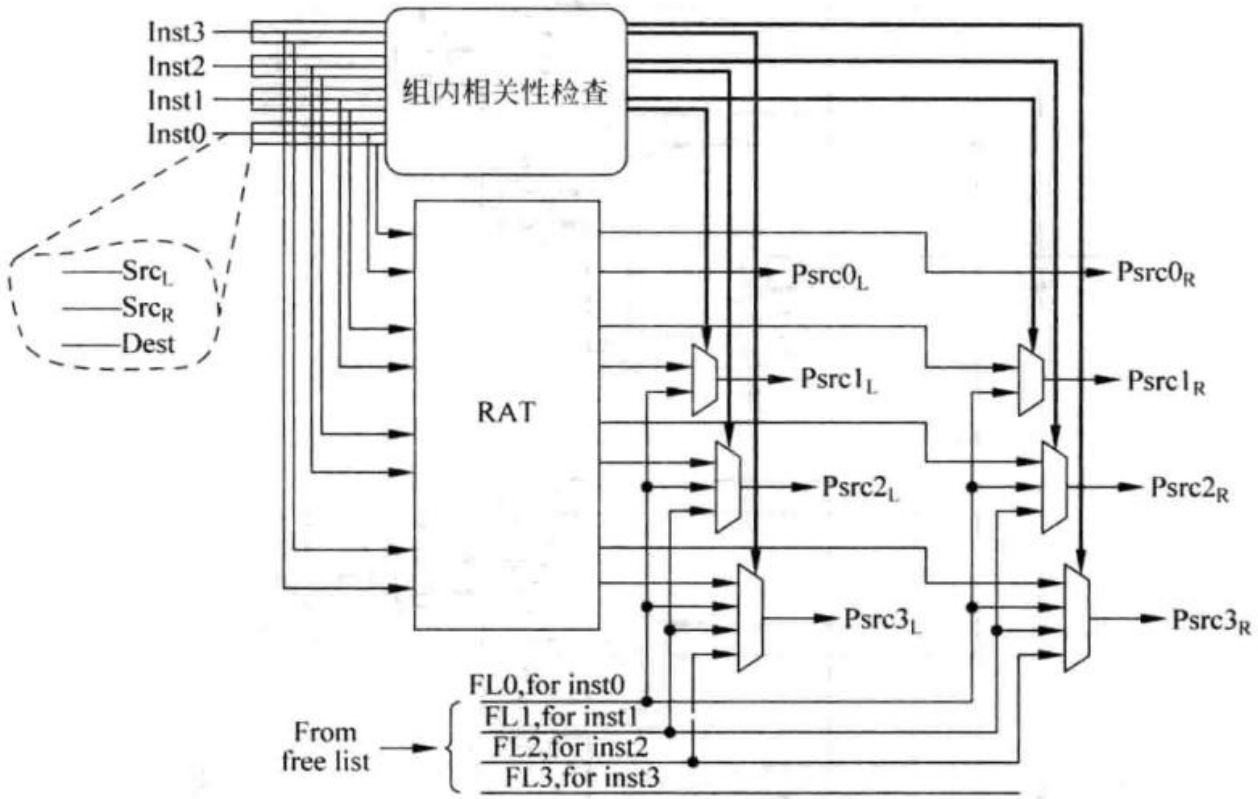


图 7.20 对 RAW 相关性进行检查的电路图

Superscalar Renaming

- 多发射下WAW问题
  - 先写的指令可以不写
  - Free的指令要弄对

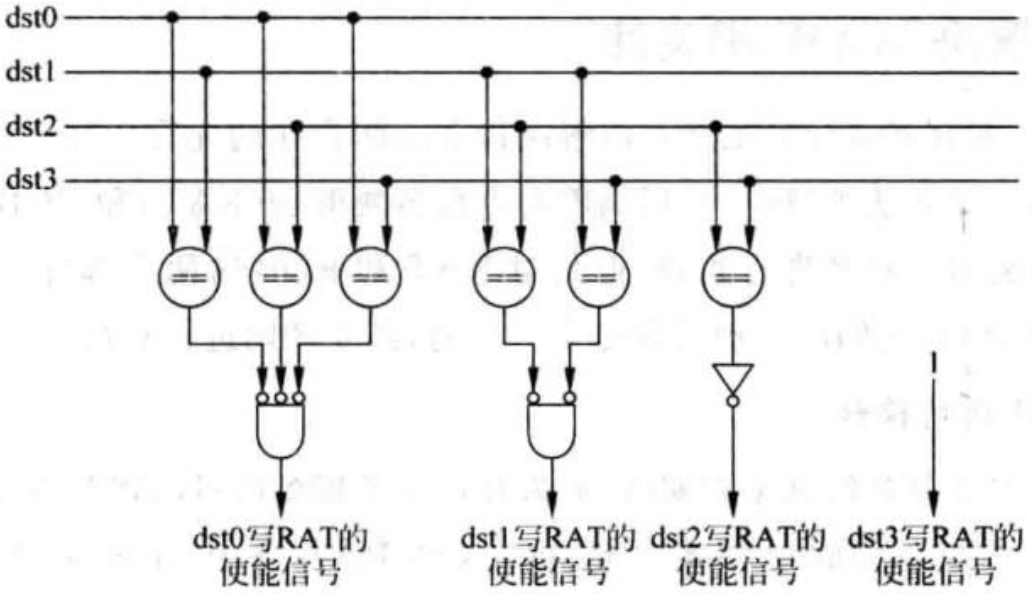


图 7.22 对于写 RAT 来说，实现 WAW 相关性检查的电路结构图

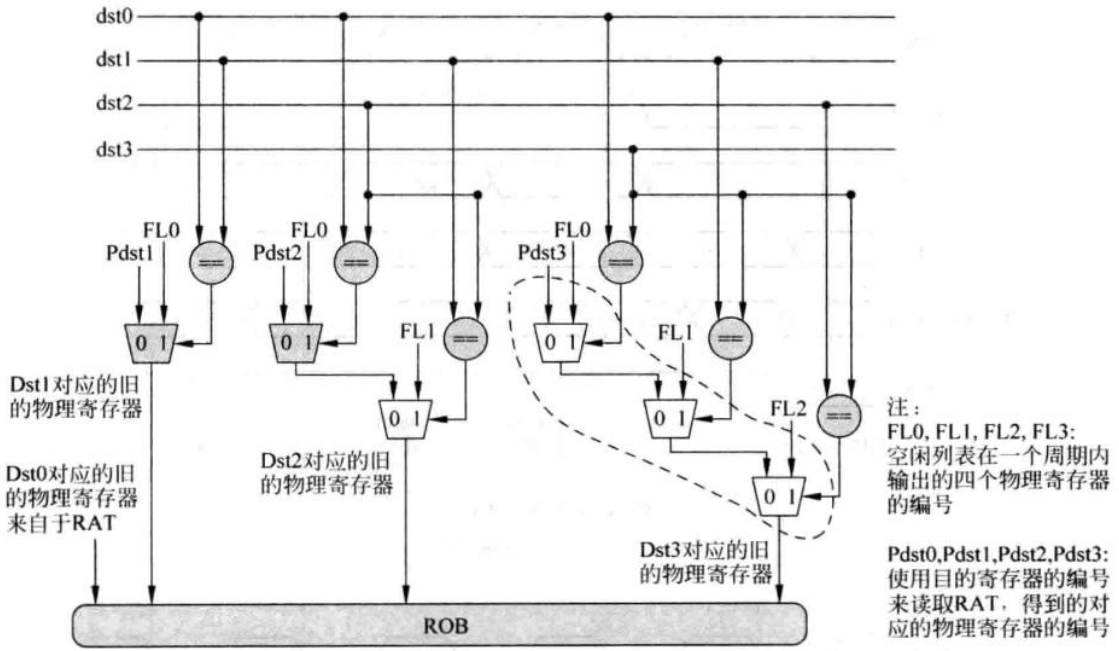


图 7.24 对于写 ROB 来说，进行 WAW 相关性的检查

Issue Queue

(1) 发射队列(Issue Queue),用来存储已经被寄存器重命名,但是没有被送到 FU 执行的指令,通常也被称为保留站(Reservation Station)[33];

(2) 分配(Allocation)电路,用来从发射队列中找到空闲的空间,将寄存器重命名之后的指令存储到其中,不同的发射队列的设计方法,会直接影响这部分电路的实现,在后文会进行详细的介绍;

(3) 选择(Select)电路,也称为仲裁(Arbiter)电路,如果在发射队列中存在多条指令的操作数都已经准备好了,那么这个电路会按照一定的规则,从其中找出最合适的指令,送到 FU 中去执行,这部分电路是发射阶段比较关键的部分,会直接影响整个处理器的执行效率;

(4) 唤醒(Wake-up)电路,当一条指令经过 FU 执行而得到结果数据时,会将其通知给发射队列中所有等待这个数据的指令,这些指令中对应的源寄存器就会被设置为有效的状态,这个过程就是唤醒。如果发射队列中一条指令的所有源操作数都有效了,则这个指令就处于准备好(ready)的状态,可以向选择电路发出申请。

Wake Up

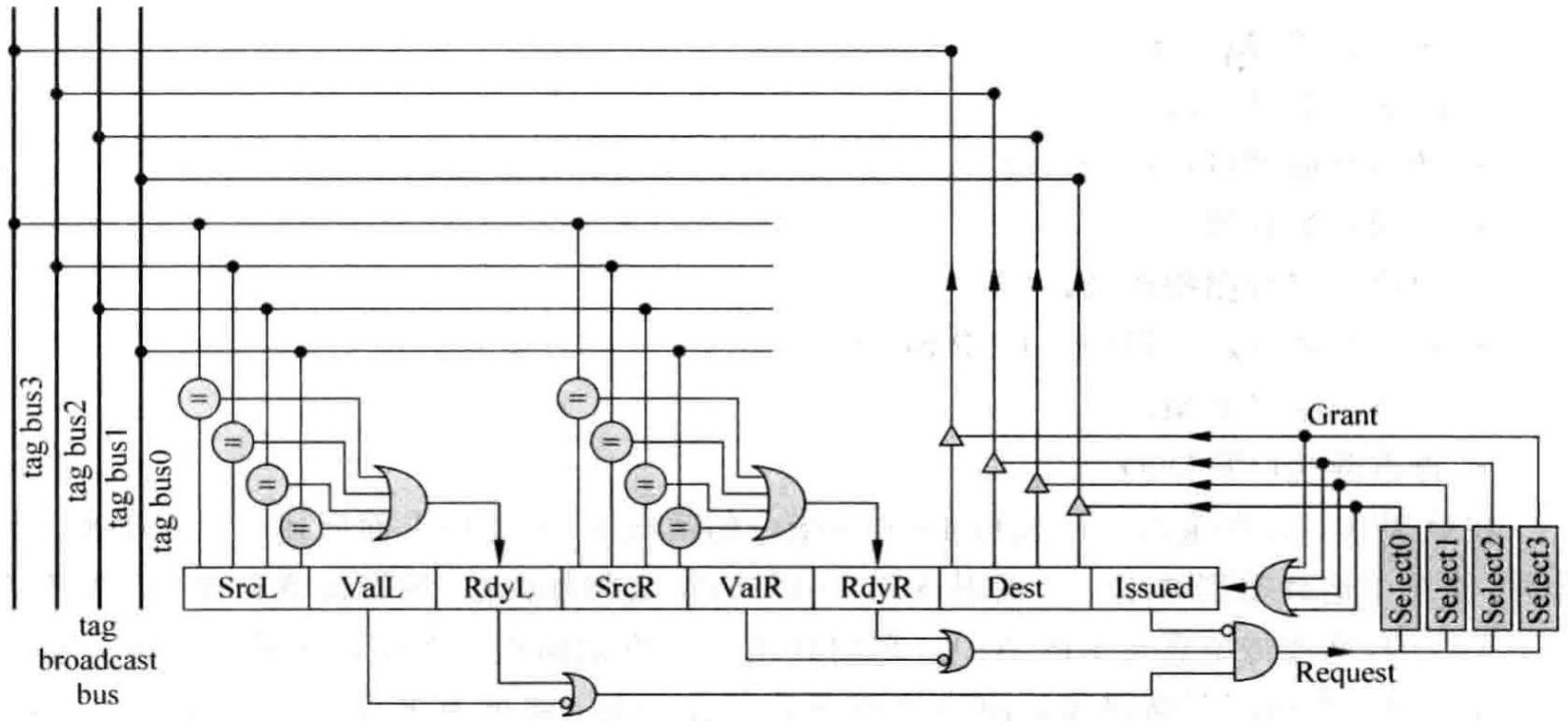


图 8.33　唤醒电路

Select & Allocation

- 每次往下压
- FIFO 读写

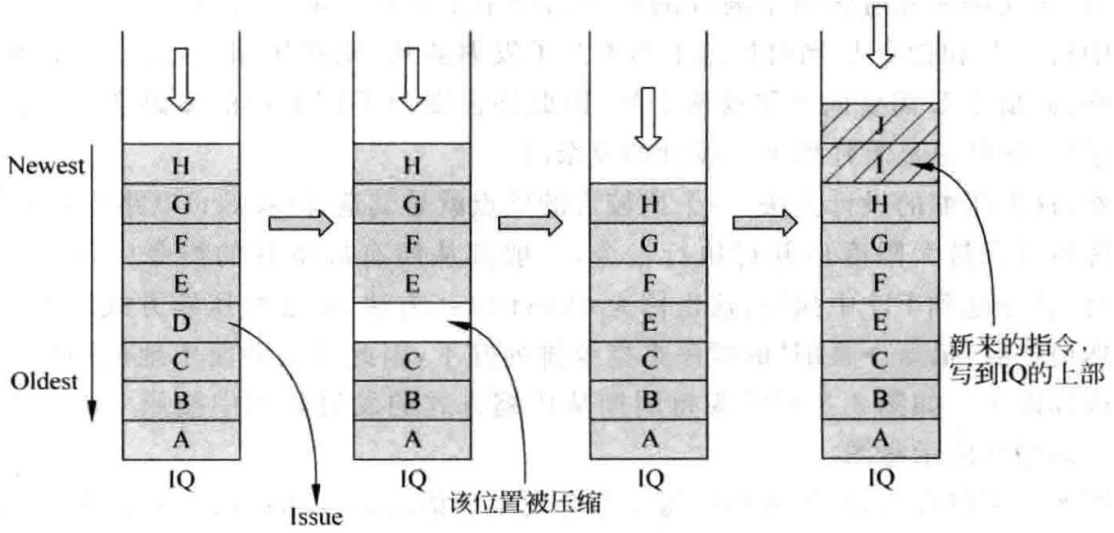
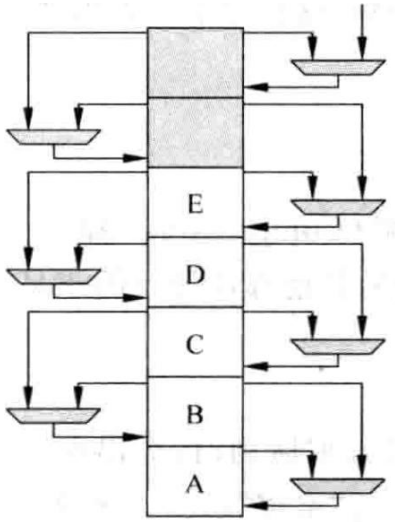


图 8.5　压缩方式的发射队列的工作原理



8.6　每周期可以压缩一个表项的发射队列

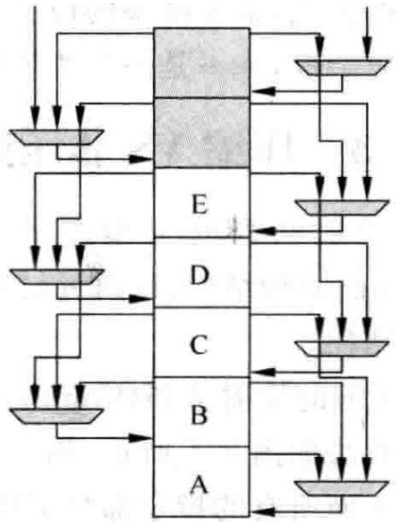


图 8.7　每周期可以压缩两个表项的发射队列

Commit

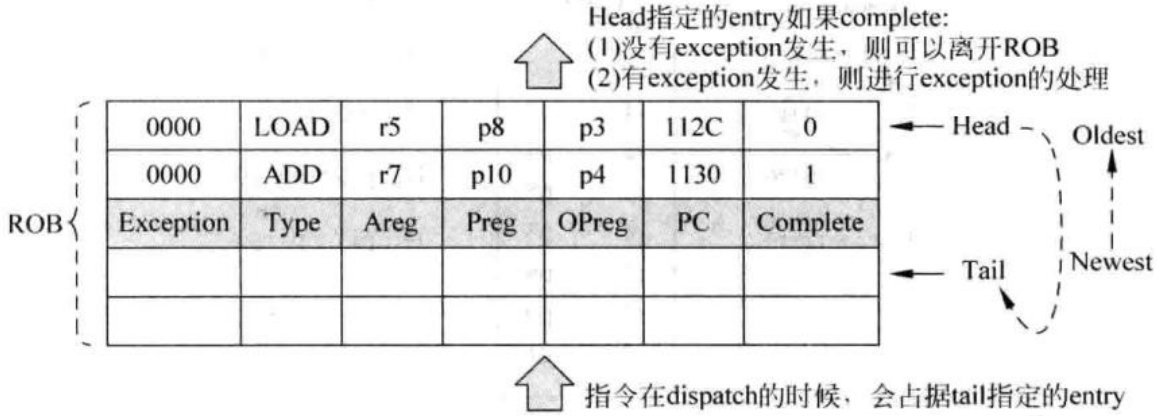


图 10.2　重排序缓存(Reorder B

(1) Complete：表示一条指令是否已经执行完毕；

(2) Areg：指令在原始程序中指定的目的寄存器，它以逻辑寄存器的形式给出；

(3) Preg：指令的 Areg 经过寄存器重命名之后，对应的物理寄存器的编号；

(4) OPreg：指令的 Areg 被重命名为新的 Preg 之前，对应的旧的 Preg，当指令发生异常(exception)而进行状态恢复时，会使用到这个值；

(5) PC：指令对应的 PC 值，当一条指令发生中断或者异常时，需要保存这条指令的 PC 值，以便能够重新执行程序；

(6) Exception：如果指令发生了异常，会将这个异常的类型写到这里，当指令要退休的时候，会对这个异常进行处理；

(7) Type：指令的类型会被记录到这里，当指令退休的时候，不同类型的指令会有不同的动作，例如 store 指令要写 D-Cache、分支指令要释放 Checkpoint 资源等。

Commit

- 管理处理器状态
- 写RAT表，标记为提交状态
- 处理Store指令，写入到cache
- 异常处理
- 分支预测错误恢复
- ROB中记录信息很多，要干的事情也很多，需要细致设计

Branch Prediction

- 识别Branch指令
- Pre decoder / fast decoder

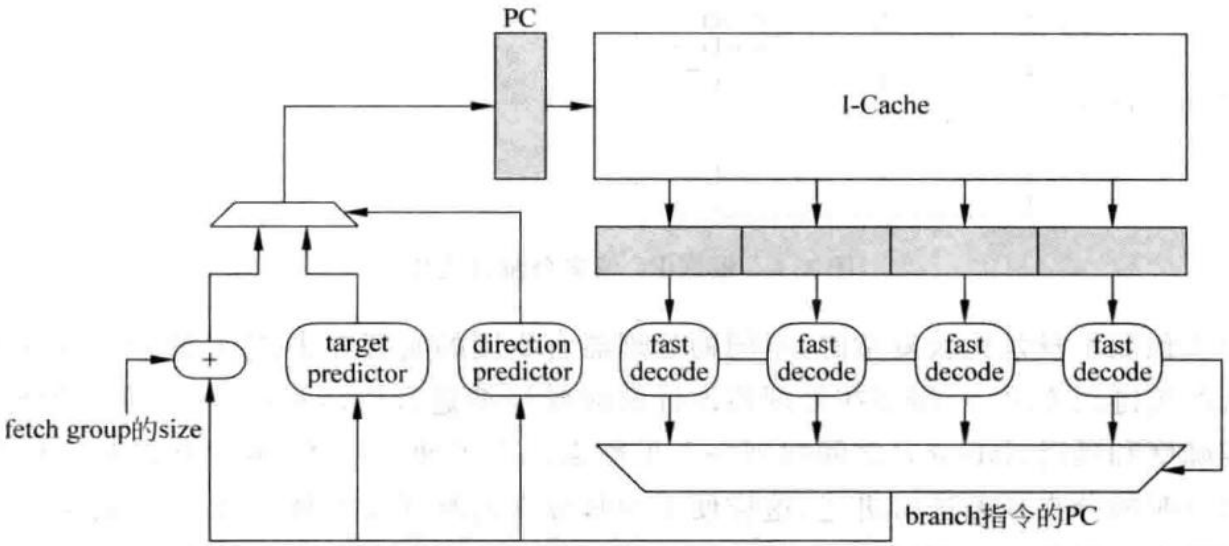


图 4.3　使用快速解码的方式来分辨分支指令

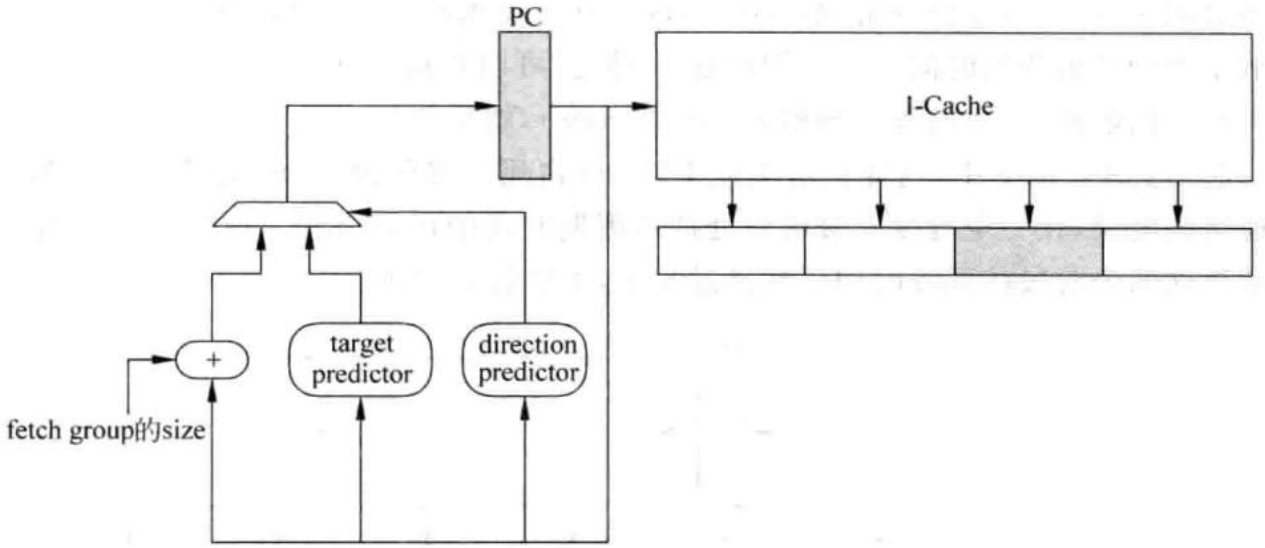


图 4.4　根据 PC 值来分辨分支指令

Branch Prediction

- PC相对 立即数 使用BTB

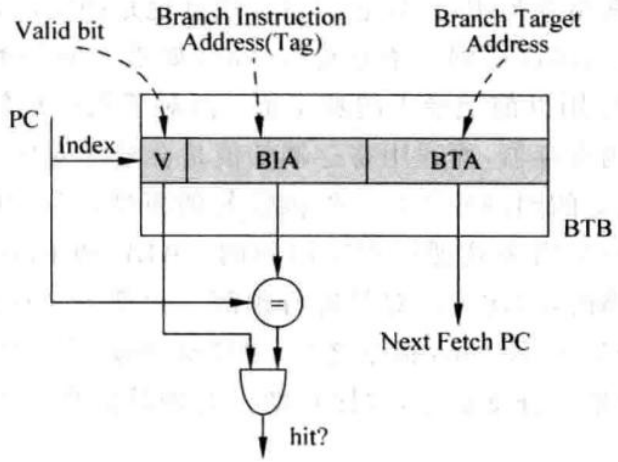


图 4.32 Direct-mapped BTB

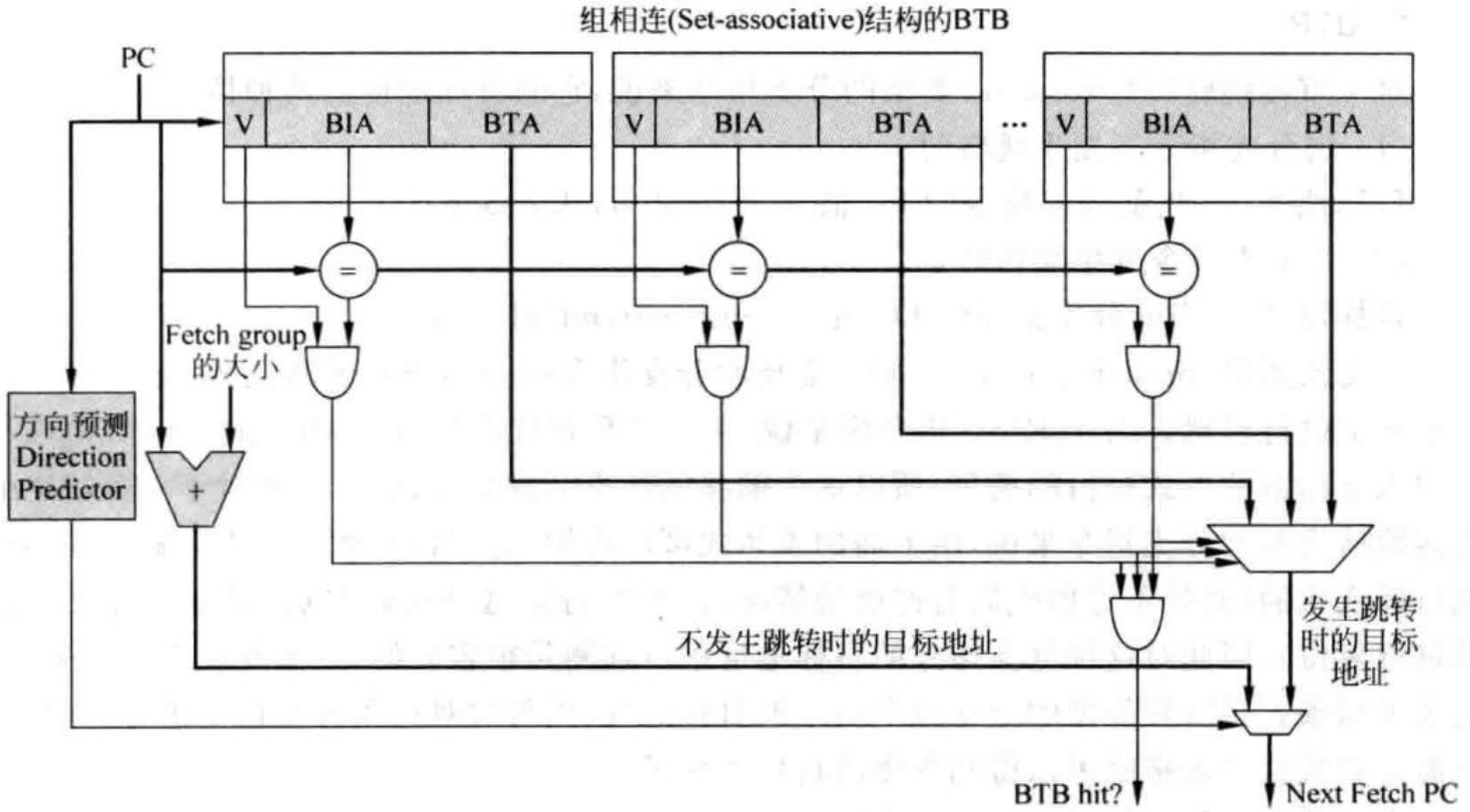


图 4.33 Set-associative BTB

Branch Prediction

- Return 跳转用 RAS 记录

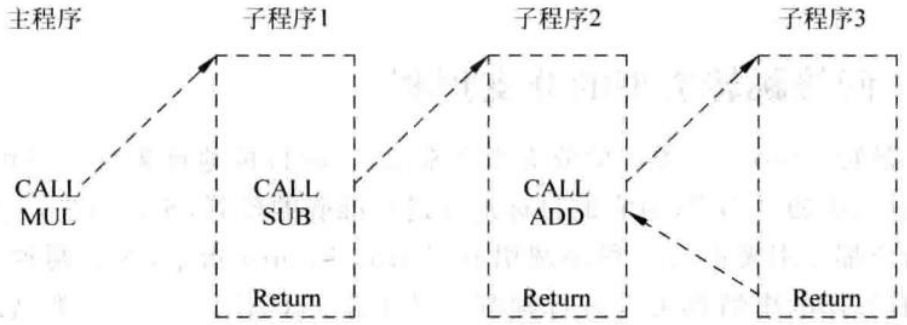


图 4.39　一个三级嵌套的子程序调用

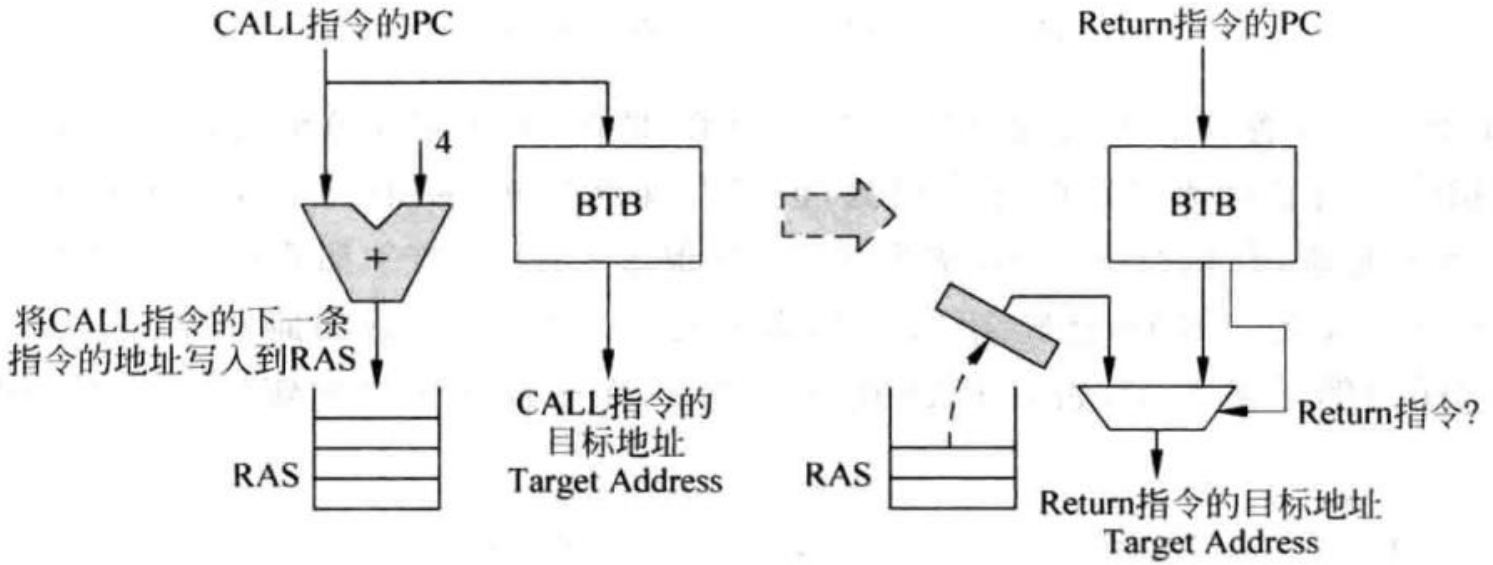


图 4.41　对 CALL/Return 指令进行分支预测

Branch Prediction

- Taken/Not Taken
- gShare
- gSelect
- TAGE

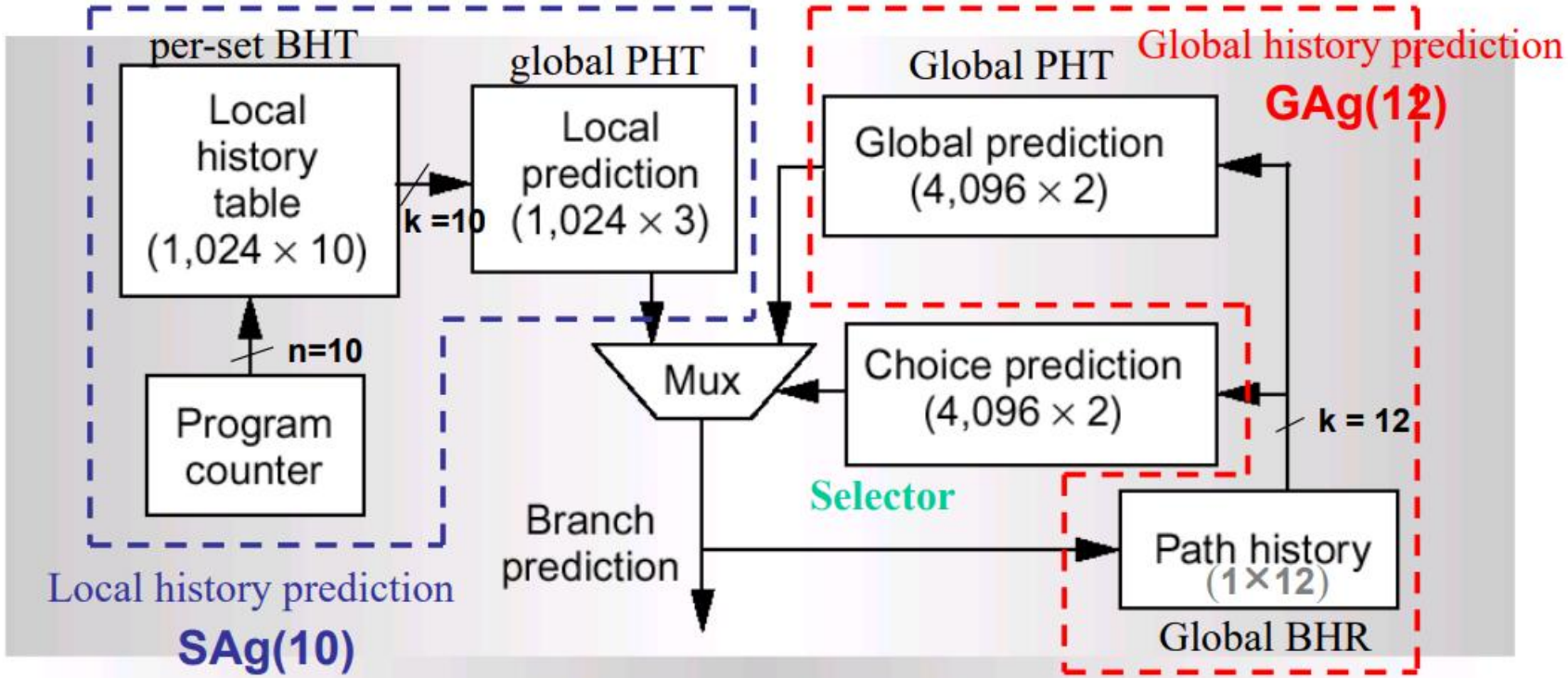


Figure 4. Block diagram of the 21264 tournament branch predictor. The local history prediction path is on the left; the global history prediction path and the chooser (choice prediction) are on the right.

Branch Prediction

- 需要一个表记录预测跳转的情况

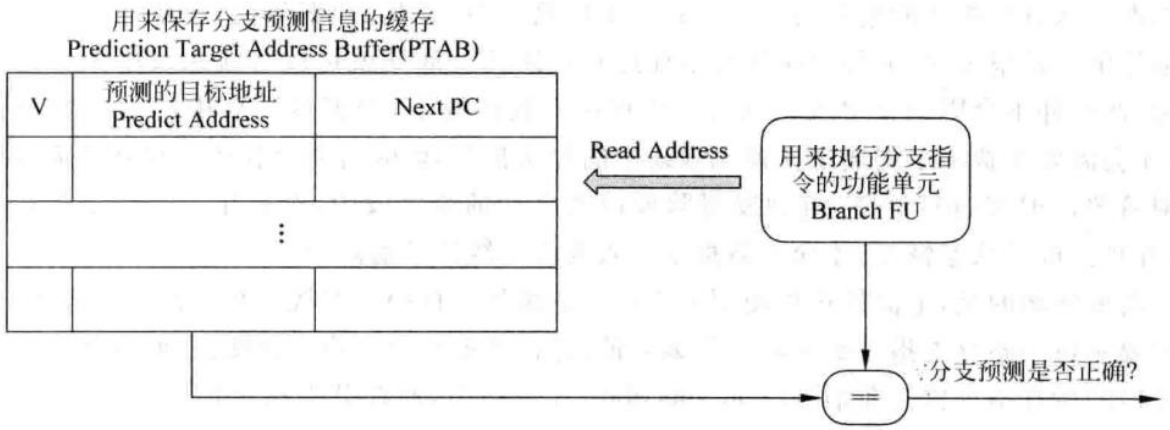


图 4.54　使用缓存来保存那些预测跳转的分支指令

在图 4.54 中，对方向预测是跳转(taken)的分支指令进行保存的地方称为 PTAB (Prediction Target Address Buffer)，它主要由三部分组成。

(1) Valid，表示 PTAB 中某个表项(entry)是否被占用，当一条方向预测为跳转的分支指令写入到 PTAB 中时，会将这一位置为 1，表示这个表项被占用；当这条分支指令到达流水线的执行阶段，完成了分支预测是否正确的检查时，这个表项就可以被释放了，因此将这一位清零。

(2) Predict Address，分支指令被预测的目标地址。

(3) Next PC，方向预测为跳转的分支指令的下一条指令的 PC 值，如果发现这样的分支指令预测错误，直接使用这个值就可以作为正确的取指令地址。

Branch Prediction

- 如何恢复执行
  - 等待提交
  - checkpoint

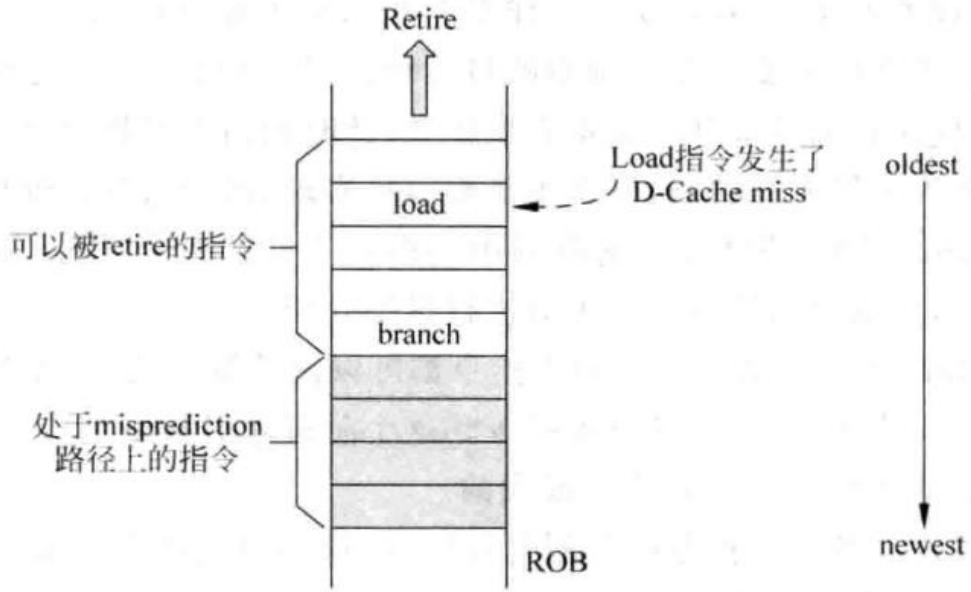


图 4.50　使用 ROB 对分支预测失败时的处理器进行状态恢复

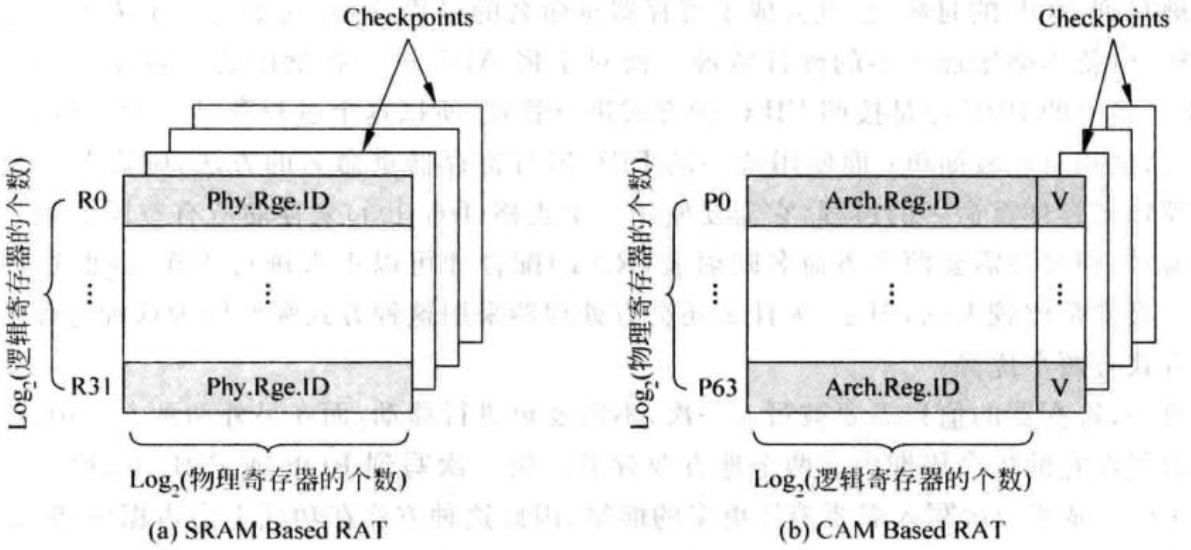


图 7.9　RAT 的两种物理实现方法

Conclusion

- 设计较为复杂
- 需要良好的团队合作
- 需要提前设计好框图，明确好部件，编写文档
- 注意编写可综合的代码
- 多搜搜资料

- 推荐参考书 《超标量处理器设计》姚永斌 清华大学出版社