超标量处理器中重排序缓冲器的研究

张 鹤
(同济大学 微电子中心，上海 200092)

摘 要：介绍了重排序缓冲器实现思想与硬件结构，并提出了增加结果锁存器和将重排序缓冲器由集中式改为分布式的设计，来降低重排序缓冲器设计复杂度的方案。

关键词：重排序缓冲器；寄存器重命名；超标量；乱序执行；低复杂度
中图分类号：TP302 **文献标识码**：A

Study of reorder buffer in superscalar processors

ZHANG He
(Microelectronics Center,Tongji University,Shanghai 200092,China)

Abstract: This paper introduces the design and architecture of ROB. Some approaches for reducing the ROB complexity are proposed. One is increasing retention latches, the other is relying on distributed ROB instead of centralized one.

Key words: reorder buffer; register renaming; superscalar, out-of-order; low-complexity

随着科技的发展，人们对计算机处理速度的要求越来越高。为了提高微处理器的性能，人们提出了超标量流水线技术。超标量技术是指 CPU 在每个时钟周期内可以完成一条以上指令的并行执行技术[1]。实现多指令并行执行的关键在于乱序执行。乱序执行是指不按照指令原始顺序执行的技术。而实现指令乱序执行的关键是用到了寄存器重命名技术。寄存器重命名是用一个或者多个虚拟寄存器来代替真实的数据寄存器的硬件猜测方法。重排序缓冲器 ROB(Reorder Buffer)是超标量处理器中基于寄存器重命名技术的一种实现结构，在 MIPS R10000、Intel Pentium 系列处理器和 IBM Power 系列处理器中得到了广泛的应用。

本文在介绍重排序缓冲器原理和实现的基础上，分析了重排序缓冲器的复杂度问题，即重排序缓冲器是一个占用大量寄存器资源并且有多个并行读写端口的复杂器件，耗费大量的功耗，容易出现不必要的延时。文中通过适当的改进方法，优化了重排序缓冲器的结构和性能。

1 超标量处理器中的寄存器重命名技术

1.1 超标量处理器

现今的超标量处理器都是通过一个时钟周期内发射多条指令并且乱序执行这些指令来实现多指令并行技术的。

一般流水线技术的局限在于采用了按序发射指令的机制，指令必须按照原来的顺序执行。即如果流水线中出现停顿，那么后续指令则无法前行。因此，若 2 条紧挨着的指令存在相关关系就会引起停顿。对于有多个功能部件的机器，会造成这些功能的闲置。乱序执行技术允许指令不按照原来的发射顺序执行，哪条指令的操作数已经就绪，就可以提前执行。这样可有效提高指令的执行效率。

1.2 寄存器重命名

乱序执行带来的数据相关和控制相关的问题，实质上是由于使用了共同的寄存器资源保存了不同的变量值所引发的[2]。若采用更名的方法，用 2 个不同的寄存器来保存这 2 个变量，出现相关的 2 条指令就能并发执行，不会引起性能的损失。Tomasulo 算法就是一种基于寄存器重命名的动态调度算法，并在超标量流水线中得到广泛应用。

2 重排序缓冲器的分析及相关问题的提出

2.1 重排序缓冲器的原理

乱序执行会带来数据冲突和控制冲突，数据冲突

问题可以单纯地采用保留站的调度技术予以解决，控制冲突的解决则要利用分支预测技术，而分支预测常常会带来预测的失败。如果仅仅允许乱序执行而不顺序写回的话，就会出现不可恢复的预测失败。因此，在Tomasulo算法的基础上，又采用了一种基于硬件的猜测技术—重排序缓冲器。

重排序缓冲器本质上也是一种寄存器重命名技术，重排序缓冲器提供了额外的寄存器，就像Tomasulo算法中的保留站扩展了寄存器堆一样。重排序缓冲器在指令执行结束和指令提交之间保存指令的执行结果。因此，重排序缓冲器可以为指令提供源操作数，就像Tomasulo算法中的保留站一样提供操作数[3]。在Tomasulo算法中，一旦一条指令写入它的执行结果，任何后边的指令都可以在寄存器堆中读取该结果，在指令提交之前，数据寄存器不用得到更新，由重排序缓冲器在指令执行完毕和提交之前提供操作数。如图1所示，重排序机制类似于一种旁路机制，这样可将在重排序缓冲器中保存的某个指令的结果送到等待它的执行单元中，不用经过数据寄存器[4]。

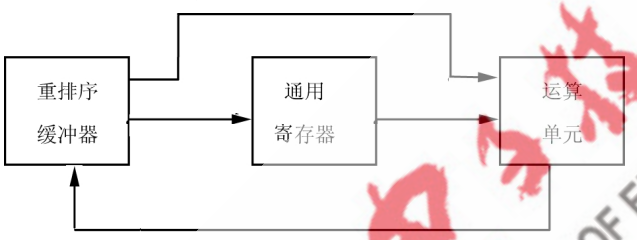

图1 重排序缓冲器提供操作数

相比提供操作数，重排序缓冲器另一个最大的作用是保证指令有序提交和异常行为恢复执行[4]。提交阶段必须按一开始程序的最初顺序提交结果到指定的存储单元。在指令译码之后、乱序发射之前，每条指令已经在重排序缓冲器中按序申请到存储单元。当指令执行完毕，重排序缓冲器就可以保证最终结果顺序提交。一旦出现中断或异常行为，则废除未提交的剩余所有执行结果，进入中断处理或从正确的指令处重新开始执行。如图2所示。

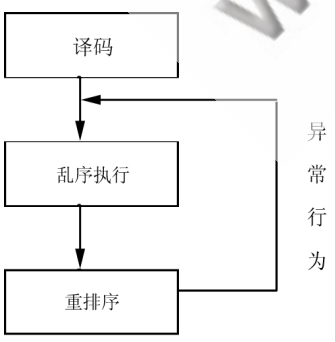

图2 异常行为处理

2.2 重排序缓冲器的硬件实现

重排序缓冲器是用于乱序执行后恢复指令原来顺序的一种硬件结构，以达到指令结果顺序提交的目的。可以把重排序缓冲器看作是一个包含头指针和尾指针的堆栈FIFO[5]。每条指令进入流水线的时候，按照程序的最先顺序都在重排序缓冲器中依次占据了一列条目，等指令执行完毕，按照先入先出的顺序依次提交指令。当发生中断或者异常行为时，也能恢复原来的执行顺序。重排序缓冲器主要由下列几个字段构成：

(1) 指令字段：记录指令；

(2) 目标寄存器字段：记录指令结果要写入的数据寄存器的地址；

(3) 数值字段：记录指令的最后结果，如果结果还没有得出，则记录得出该结果的指令在ROB中的地址；

(4) 完成字段：记录指令是否执行完毕。

重排序缓冲器在超标量流水线中的实现如图3所示。运算单元FU(Floating Unit)计算出结果后，将其写到重排序缓冲器中。同时通过前置定向总线FB(Forwarding Bus)送给发射队列IQ(Instruction Queue)中正在等待该操作数的指令。重排序缓冲器中存储的结果将在指令退休时送给相应的数据寄存器ARF(Architecture Register Files)中。在指令发射时，操作数的值按照具体情况，既可以从数据寄存器中获得，也可以从重排序缓冲器中获得，如果值还没有计算出来，指令要在发射队列中等待，直到前置定向总线将运算结果送到发射队列指定的位置。

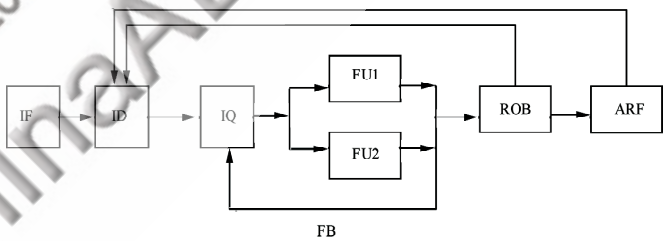

图3 重排序缓冲器在超标量流水线中的实现

3 重排序缓冲器的复杂性和优化方案

3.1 重排序缓冲器的复杂性

这类的重排序缓冲器通常以允许多端口同时读写的寄存器堆的方式实现。在一个N指令发射的超标量流水线中重排序缓冲器至少需要$2N$个读端口，以供N条指令可以在一个时钟周期内读取所需要的$2N$个操作数。至少需要N个写端口，以供N条指令一个时钟周期内将结果写入重排序缓冲器中。

多端口寄存器堆的实现会加剧设计的复杂度，并且实际的端口数有时还会因为出现双倍位宽的操作数而加倍。在通常情况下，可以采用双重排序缓冲器并行存在的方式，以减少每个重排序缓冲器端口的数量。但是，双倍位宽和单倍位宽的数据常常一起存在，倘若采用双ROB的形式将不可避免的带来极大地浪费。

另外，随着端口数的增加，寄存器堆的面积也呈快速增长趋势，这必将导致时序的延迟和功耗的增加。因此，合理地优化重排序缓冲器的结构设计日益成为人

们关心的焦点。

3.2 重排序缓冲器结构的优化

一个优化的方法是去除重排序缓冲器上原操作数读端口用以简化寄存器堆的设计。当指令的结果从运算单元送到重排序缓冲器中的同时，通过前置定向总线送到指令发射单元中等待该操作数的指令队列。之后，该操作数提交给数据寄存器，指令分配单元无法再从重排序缓冲器中读此操作数。当操作数需要第二次被提取时，而该操作数还没有被提交给数据寄存器时，常常会因为无法读重排序缓冲器中的值，出现延迟，造成流水线的阻塞。因此，可以参考计算机中缓存 Cache 的设计方法，如图 4 所示，加一个存储单元数远少于重排序缓冲器的数据锁存器 RL(Retention Latches)以存储指令结果，这样执行结束的指令结果在写入重排序缓冲器的同时也并行写入了数据锁存器。为了在增加数据锁存器的基础上而不增加设计的复杂度，数据锁存器的存储量应该尽可能的小，不妨采用 LRU 算法进行锁存器内部数据的替换管理。LRU 是广泛应用于计算机缓存 Cache 中的一种数据替换算法，为了减少替换那些可能不久要用到的信息，需要记录数据块的使用次数来预测未来的使用情况[6]。这样在指令被重排序缓冲器提交给数据寄存器之前，数据锁存器可以有效解决操作数延迟获得的问题。

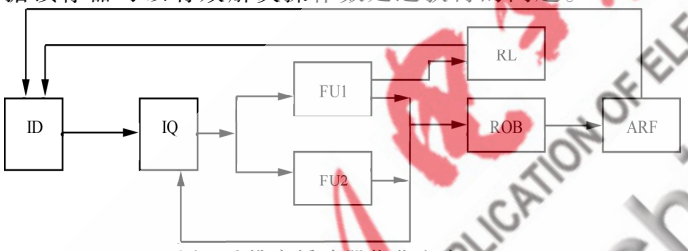

图 4 重排序缓冲器优化方案一

另一个优化的方法，从 Tomasulo 算法的分布式保留站中得到启发，将重排序缓冲器由集中式改为分布式，如图 5 所示。这样可以将一个多端口读写的重排序缓冲器简化成多个单端口读写的重排序缓冲器。这种方法简化了原来集中式寄存器堆多端口同时读写的复杂度，但也带来了因为多个分散的重排序缓冲器中指令提交的排序问题。为此，可以在译码时给每一条指令加上标志位，指明该指令在程序中的顺序，并且标注该指令发送给哪一个运算单元。这样，在每个缓冲器提交指令或者预测失败时，便能在多缓冲器间协调读取指令或者结果。

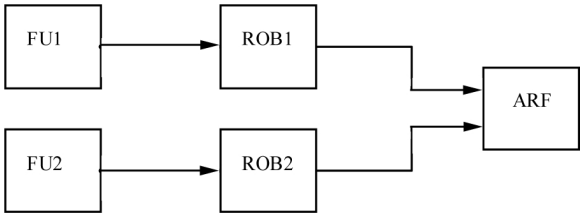

图 5 重排序缓冲器优化方案二

3.3 性能分析

根据对各种模型的分析，列出表 1 中的顺序执行、改进前的硬件猜测法以及改进后的硬件猜测法 3 种实现形式，采用 Verilog HDL 语言对结构进行描述，采用 Synopsy 公司的 Design Compiler 工具在 0.18 μm 的 CMOS 工艺下进行综合和时序分析，得出性能的比较如表 1 所列。

表1 实现比较

使用算法	路径延迟 /ns	面积/mm²
顺序执行	3.85	1.68
改进前的硬件猜测法	3.21	1.98
加锁存器的硬件猜测法	3.39	1.72
分布式硬件猜测算法	3.28	1.82

从表 1 中可以看出，在普通的顺序执行算法下，硬件路径延迟较大，但是面积最小；改进前的硬件猜测算法路径延迟最小，但是面积最大，带来了设计上复杂度和功耗损失较大的问题；采用改进后的硬件猜测算法以后，虽然路径延迟比改进前大，但是面积相对减少很多，并且路径的延迟相比顺序执行仍然减少了不少。因此，采用改进后的硬件猜测算法在减少面积的情况下得到了可观的效果。

超标量处理器中的重排序缓冲器对于实现指令的乱序执行和顺序提交起着重要的作用。随着对指令并行度要求的不断提高，重排序缓冲器的设计结构会越来越复杂。本文提出了几点改进方法，通过减少单个缓冲器上读端口的数目，有效地降低了重排序缓冲器的设计复杂度，这对将来的超标量处理器的设计有着深远影响。

参考文献

[1] STEVEN W, NADER B. Modeled and measured instruction fetching performance for superscalar microprocessors. IEEE transaction on parallel and distributed, 1998,9(6).

[2] JOHN P,MIKKO H L.现代处理器设计——超标量处理器基础[M].北京:电子工业出版社,2004.

[3] JOHN L H,DAVID A P.计算机系统结构——量化研究方法(第3版)[M].北京:电子工业出版社,2004.

[4] 邓正宏,康慕宁,罗旻.超标量微处理器的研究和应用[J].微电子学与计算机,2004,21(9):59-63.

[5] TAREK M T, WILLS D S. An instruction throughput model of Superscalar Processors[J]. IEEE Transactions On Computers,2008, 57(3).

[6] 李学干.计算机系统结构[M].西安:西安交通大学出版社,2007.

(收稿日期：2009-03-18)