

一种减少寄存器堆读端口的的设计方法

(中国科学技术大学物理系微电子学教研室, 安徽 合肥 230026)

孙岩 金西

摘要: 介绍了一种给超标量流水线分配优先级并使用缓冲队列等方式来减少寄存器堆读端口的的方法, 从而大大减小了寄存器堆的面积和功耗, 并使用了寄存器合并技术, 将两个单精度数据合并写入寄存器堆从而加快了写寄存器堆的速度。本文以有三条流水线的浮点处理器为例, 说明了其实现过程。

关键词: 寄存器堆 超标量流水线 缓冲队列 数据前推

A Design to Reduce the Ports of the Register File

(Microelectronics Lab., Dept. of Physics, USTC, Hefei Anhui 230026, China)

Sun Yan Zhang Xin Jin xi

Abstract: In this paper, a method of reducing the read ports of the register file is implemented by setting the pipeline PRI and using buffer queue, and this method reduce the area and power consumption of the register file a lot. We also give an example of the implement of a vector floating point processor which has three pipelines.

Key Words: register file, superscalar pipeline ,buffer queue , data forwarding

1、引言

在现代微处理器中, 寄存器堆已经成为重要的组成部分, 并影响着微处理器的全面性能, 随着微处理器性能的不不断提高, 对寄存器堆的要求也在不断的提高。另一方面, 随着集成电路特征尺寸的不断缩小以及芯片的规模和速度的迅速提高, 功耗已成为集成电路设计中越来越重要的因素。而微处理器超标量体系结构的不断发展, 要求寄存器堆的端口越来越多, 以便多条流水线可以同时访问寄存器堆。随着端口的增加, 寄存器堆所占的面积迅速增加, 在Alpha 21464的设计中, 多端口寄存器堆所消耗的面积是64K主Cache的5倍多^[1]。许多研究已表明, 寄存器堆已经成为现代处理器功耗预算的重要组成部分, 而且还将随着指令并行执行水平的提高而迅速增长, 端口的增多不但使寄存器堆的动态功耗增加, 同时漏电流的通路也大大的增加使得静态功耗也大幅增加, 例如, 在摩托罗拉的M. CORE 结构中, 寄存器堆的功耗占了处理器总功耗的16% , 占数据路径功耗的42%^[2]。同时寄存器堆端口数的增多使得控制逻辑更加复杂, 系统的稳定性下降, 限制了时钟频率的提高, 这些都增加了设计的复杂程度。特别是对于嵌入式领域, 这些都是致命的缺点。

为了减少寄存器堆端口增加带来的影响, 许多减少寄存器堆面积、功耗和延时的方法都被提了出来。有些是将微处理器结构分割成几块, 每一块为一个功能

作者: e-mail: ysun@mail.ustc.edu.cn; 地址: 安徽合肥市中国科技大学东区物理楼712#; 电话: 05513600155

单元并包含寄存器堆的一个子集^{[3][4]}，但这些设计需要很复杂的控制逻辑将指令分配给各个功能块并控制各个块之间的交互。将寄存器堆分组也是减少寄存器堆端口的一个方法^[5]，并且在很多微处理器中得到应用。

2. 减少寄存器堆写端口的实现方法

2.1 概述分析

在一个浮点处理器中设计三条独立工作的流水线：浮点乘加流水线（FMA），进行除了除法和开方运算的所有算数和逻辑运算；除法开方流水线（DS），用于浮点的除法和开方运算；Load/Store流水线（LS），完成寄存器堆与浮点处理器外部的数据交换。这三条流水线对寄存器堆的写操作具有随机的特点，即三条流水线可能同时写寄存器堆。在没有冲突的情况下要保证三条流水线互不影响的同时写寄存器堆就需要考虑三条流水线将三个64位双精度浮点数写入寄存器堆的情况，这时就需要6个32位寄存器堆写端口，这将占用很大的面积并使功耗大大增加。

考虑到三条流水线同时写寄存器堆的机会很小，尤其是除法和开方流水线需要十几个或几十个时钟周期的迭代才能完成一次完整的运算，所以除法和开方流水线两次写寄存器堆之间要间隔多个时钟周期，也就是说写寄存器堆的概率比较小。所以我们将三条流水线分配优先级，当有两条或三条流水线同时写寄存器堆时优先级高的流水线先写寄存器堆，同时为了不影响优先级低的流水线中后续指令继续执行，将优先级低的流水线要写入寄存器堆中的数据暂时存入到一个缓冲队列中。三条流水线优先级按照它们使用频率和重要性进行分配，即Load/Store流水线优先级最高，乘加流水线次之，除法开方流水线的优先级最低。

2.2 缓冲队列的设计

缓冲队列的结构有“串入串出”寄存器队列（如图1所示）、“串入并出”寄存器队列（如图2所示）和“并入并出”寄存器队列（如图3所示）。“串入串出”寄存器队列只有一个写入口和一个输出口，这种队列结构简单但效率很低。

“串入并出”寄存器队列只有一个输入口，输出的数据可以在队列中的寄存器进行选择。“并入并出”寄存器队列是在前两种队列的基础上发展来的。在这种结构的队列中，可以同时有多个数据写入队列的寄存器中，在输出时可以选择不同的寄存器中的值进行输出。

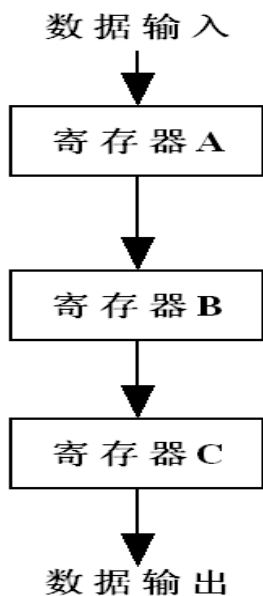


图1 “串入串出”寄存器队列

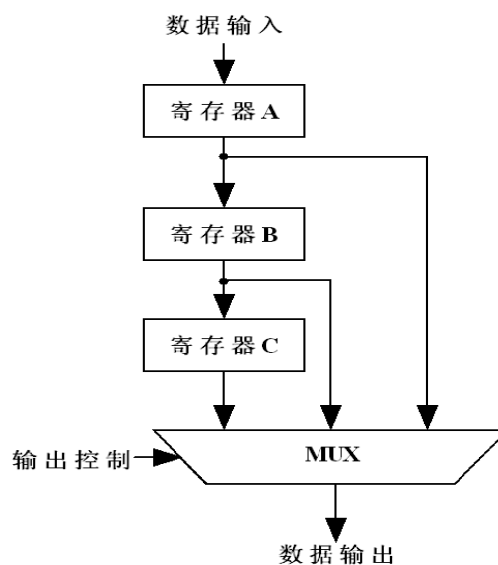


图2 “串入并出”寄存器队列

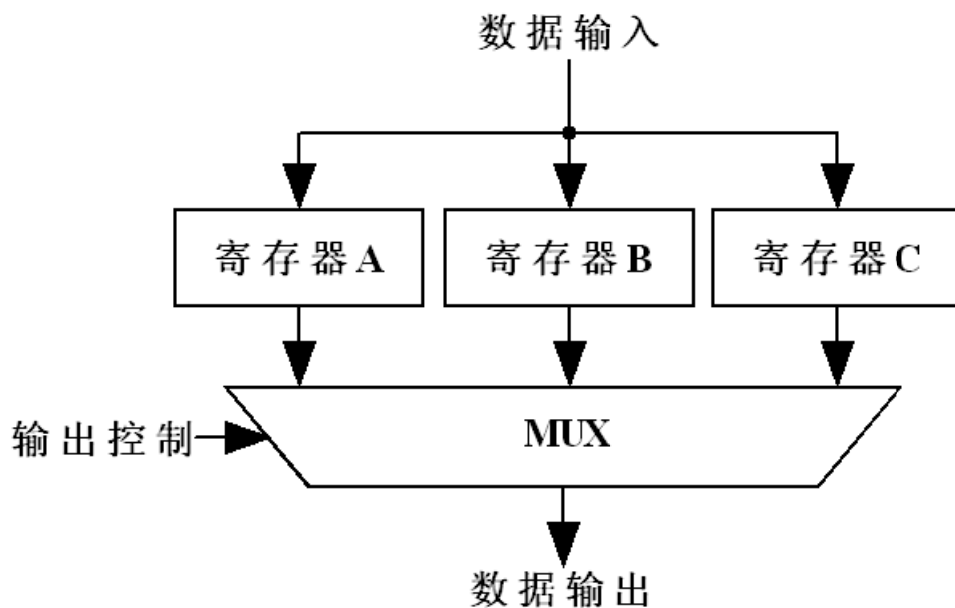


图3 “并入并出”寄存器队列

我们需要能够同时将多条流水线的值写入到缓冲队列中，并且按照先进先出的顺序输出的缓冲队列，并且根据所以我们使用“并入并出”寄存器队列的改进型缓冲队列，如图4所示。缓冲队列中的寄存器C为固定的输出端口，即数据只能从寄存器C输出，寄存器A、B、C的输出优先级为C>B>A，优先级低的寄存器中的数据将随着时钟传递到优先级高的寄存器中。例如，当三个寄存器中都含有有效数据，在下一个时钟上升沿到来的时候寄存器C中的数据输出，寄存器A和寄存器B中的数据分别写入到寄存器B和寄存器C中，同时可以有新的数据写入到寄存器A中。当缓冲队列为空时，如果要写入一个数据则将数据写入到寄存器C中，如果写入多个数据时按照输入数据来自流水线的优先级高低写入寄存器C、

B、A中。寄存器A、B、C的附加有效位V（Valid）表示该寄存器中是否存储着有效数据。

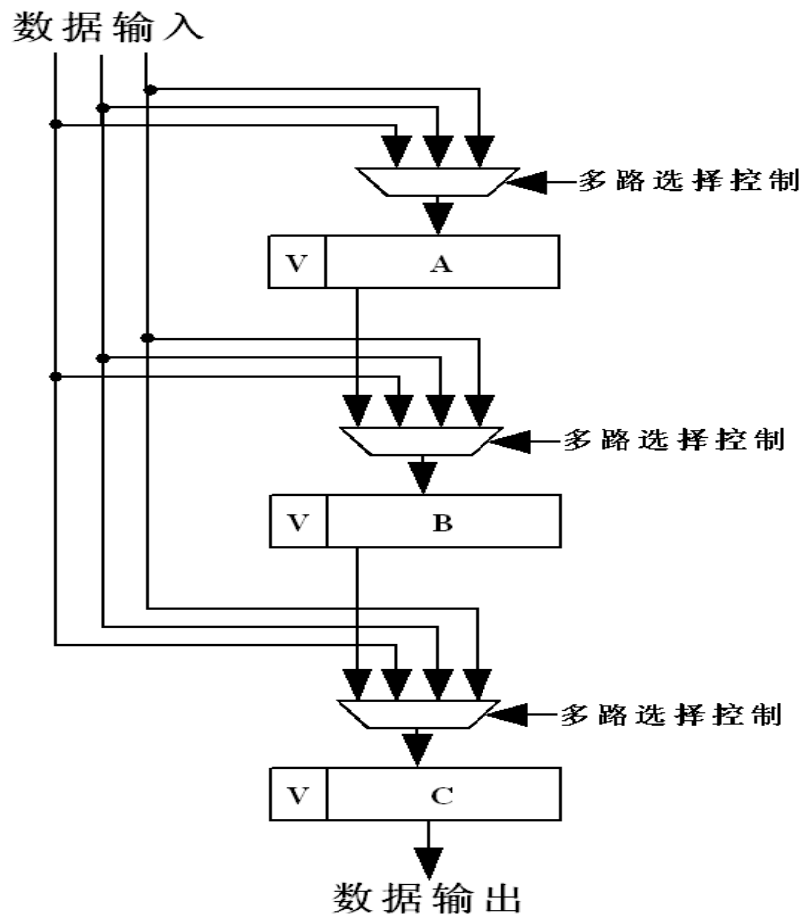


图4. 改进型“并入并出”缓冲队列

2.3减少寄存器堆读端口的结构的具体实现

从总体来说，当寄存器堆只有一个写端口时，如果有多条流水线在同一时钟周期内写寄存器堆就会产生冲突，这时优先级高的流水线先写寄存器堆，而优先级相对低的流水线先将要写入寄存器堆的数据写入缓冲队列，而不影响该流水线中后续指令的执行。当缓冲队列中存在有效数据时先将缓冲队列中的数据写入到寄存器堆中，同时将流水线中的数据写入到缓冲队列中。如果有多条流水线要进行写操作，但缓冲队列中没有足够的空间进行存储（两种情况：有两条流水线将要进行写寄存器堆操作而缓冲队列已满。三条流水线将要进行写寄存器堆操作而缓冲队列已满或只有一个空闲单元），那么优先级低的流水线就要先停止工作，直到有缓冲空间可以使用。

如图5所示，是减少寄存器堆读端口实现方法整体结构图。三个缓冲队列单元A、B、C都可以存储一个64位的双精度数据，为了同时可以方便的存储32位单精度数据，将A、B、C都分为2个32位的数据存储单元，每一个单元包含一个有效

位V来表示其中的32位数据是否有效，并包含了一个5位的寄存器号RN，寄存器号RN表示该单元中数据需要写入寄存器堆的地址。当存储的数据为双精度时，将64位的寄存器占满，当存储单精度数据时将占用其低32位。

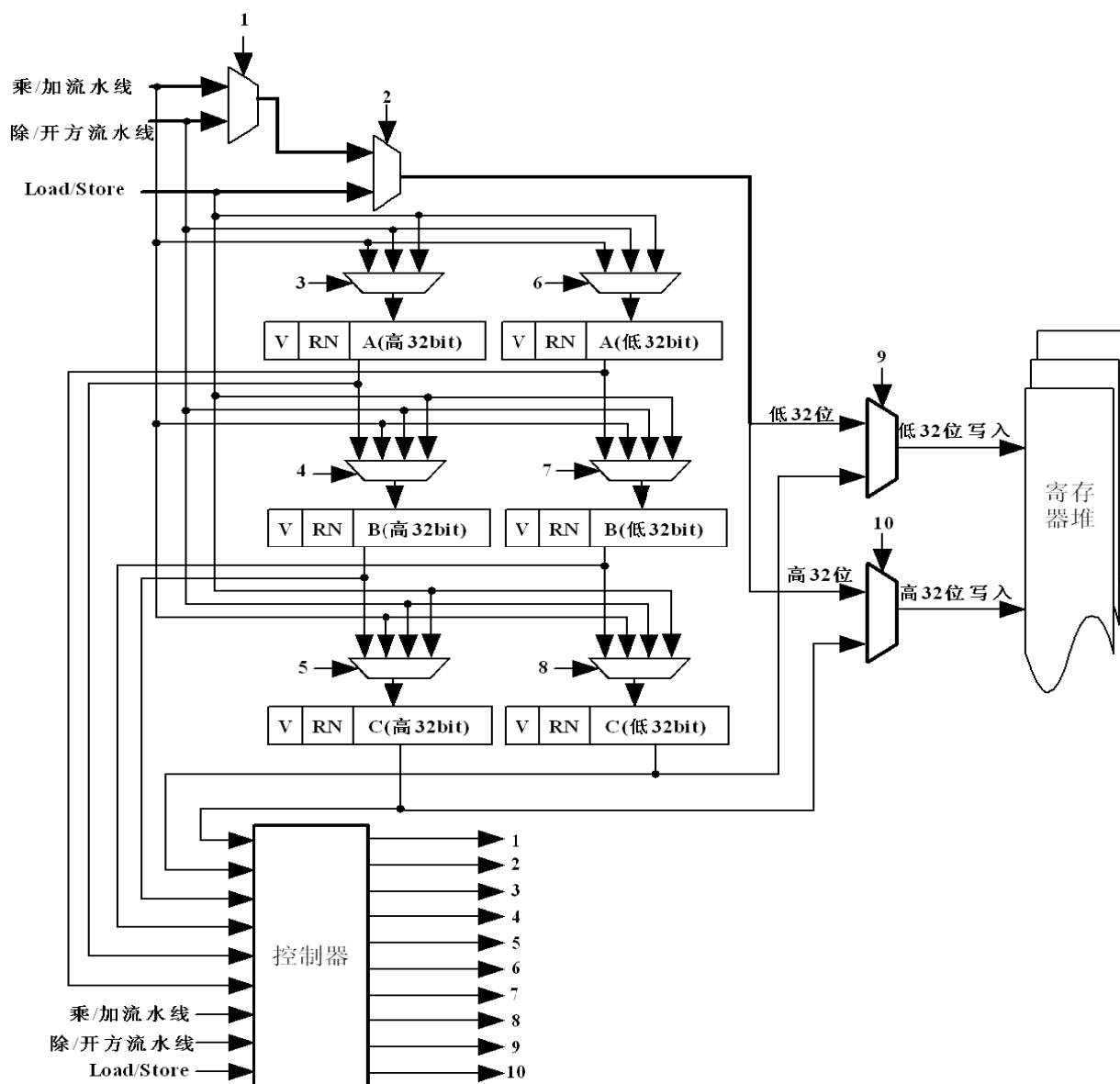


图5 减少寄存器堆读端口的结构

如果流水线中的数据可以直接写入到寄存器堆时无需写入到缓冲队列中，否则按照优先级顺序写入到缓冲队列。将缓冲队列中的数据写入寄存器堆时，按先进先出的原则，即优先级为C>B>A，写入的顺序也按此顺序进行：当A、B、C均为空或C将为空时将需要写入的数据写入到C中，如要写入两个数据则将数据写入到C与B中，如果优先级高的缓冲单元出现空闲状态，优先级低的单元中的数据将顺次前移。所以整体的优先级排列为：C>B>A>Load/Store流水线>乘/加流水线>除/开方流水线。

在这种结构中为了减小先写后读相关（RAW）所产生的后续指令等待读寄存器的时间，将缓冲队列设计为可提供数据前推的存储队列，即缓冲队列的3个寄存器都可以将存储的数据前推到执行级的最前端。在这种情况下，进入缓冲队列的数据就相当于写入了寄存器堆，而不会因为缓冲队列中数据晚些写入寄存器堆而引起后续指令从寄存器堆中读该数据而产生停滞。同时这也减小了读寄存器堆频率，降低了读寄存器端口所产生的资源冲突的频率。缓冲队列中每个寄存器的RN纪录了数据将要写入寄存器堆中的寄存器号，如果后续指令需要读取的寄存器其中的某个寄存器号相吻合，即可将该数据进行前推操作。

我们写入寄存器堆中的数据可能是64位双精度或32位单精度数据，所以我们的写端口要为64位，有很多情况下我们写入的数据位32位或是更小，所以我们将这个64位的写端口拆分成了两个独立的32位写端口。这时，如果我们只需写入一个32位数据，那么另一个32位写端口即可不工作，这样就减少了功耗和寄存器堆访问时间。可以同时写两个不相关的32位数据，提高了端口的利用率，提高了速度。如图所示，三个缓冲队列单元A、B、C均为两个32位存储器的组合，如果要存储一个64位双精度数据，则将它存储到A、B、C中完全没有被占用的64位寄存器，当只需存储一个32位单精度数据时，只需将它存储到缓冲队列单元中的低32位。为了使两个不相关的32位数据同时写入寄存器堆，所以当将要写入寄存器堆的数据为32位数据，另有缓冲队列中或三条流水线的写回级将写32位数据时，则将这两个独立的数据分别从寄存器堆写入端口的低32位和高32位写入寄存器堆。缓冲队列中的优先级高于三条流水线的优先级，所以缓冲队列中有数据时先将缓冲队列中的数据写入到寄存器堆，也就是将缓冲队列单元C中的数据写入，此时如果C中的数据为32位数据，则察看B、A和三条流水线要写入的数据是否为32位，如果是的话，将这两个32位数据写入寄存器堆。当缓冲队列为空时，三条流水按照优先级的顺序当将要写存储器堆的数据为32位时，如果有优先级更低的流水线同时要写32位数据时则将这两个数据同时写入寄存器堆。

控制单元（仲裁器）有九个输入端口分别检测三条流水线和缓冲队列的三个基本单元的状态。输出的控制信号控制数据的流向。控制信号9决定了是将流水线中的低32位数据还是缓冲队列中C的数据的低32位写入到寄存器堆中。控制信号10决定了是将流水线中的高32位数据还是缓冲队列中C的数据的高32位写入到寄存器堆中。通过控制信号9和10的控制可以将流水线中或缓冲队列中64位数据写入寄存器堆，也可以将流水线中的32位数据和缓冲队列中的32位数据同时写入到寄存器堆中。如果是将流水线中的数据写入到寄存器堆，控制信号1和2 决定了那条流水线中的数据写入到寄存器堆中。如果是将缓冲队列中的数据写入到寄存器堆，则是存储器C中的数据写入到寄存器堆中。控制信号3至8决定了哪些数据将写入到缓冲队列的基本单元A、B、C中的高32位和低32位。

2.4 缓冲队列的合并方法

为了提高缓冲队列的利用率，并减少写寄存器堆时进行的上述判断，我们在缓冲队列中也使用类似的存储方式。在存储器A、B和C中优先高的存储器只存储一个32位数据时，可以将只存储32位数据的优先级低的的寄存器中的值或是将要写入到缓冲队列的32位的值写入到优先级高的寄存器中的高32位中。图6给出一

个例子：图中A、B、C寄存器灰色的存储着有效数据，白色的为空闲寄存器。寄存器中的数字代表存储的数据，可以看出寄存器B中存储的是一个32位的单精度或整型数据。三条流水线中，乘加流水线将要写入32位数据，除/开方流水线不进行写操作，Load/Store流水线将要写入64位数据。在下一个时钟上升沿到来时，寄存器C中的64位数据写入到寄存器中，缓冲队列中其余数据顺次下移，来自Load/Store流水线的双精度数据写入到寄存器A，来自乘加流水线的32位数据写入到寄存器C的高32位，这避免了乘加流水线的停滞。数据的通路控制在流水线执行级的最后一级进行判断，因为这时即可知道流水线在下一个时钟周期是否将数据以及那种类型的数据写入到寄存器堆中。

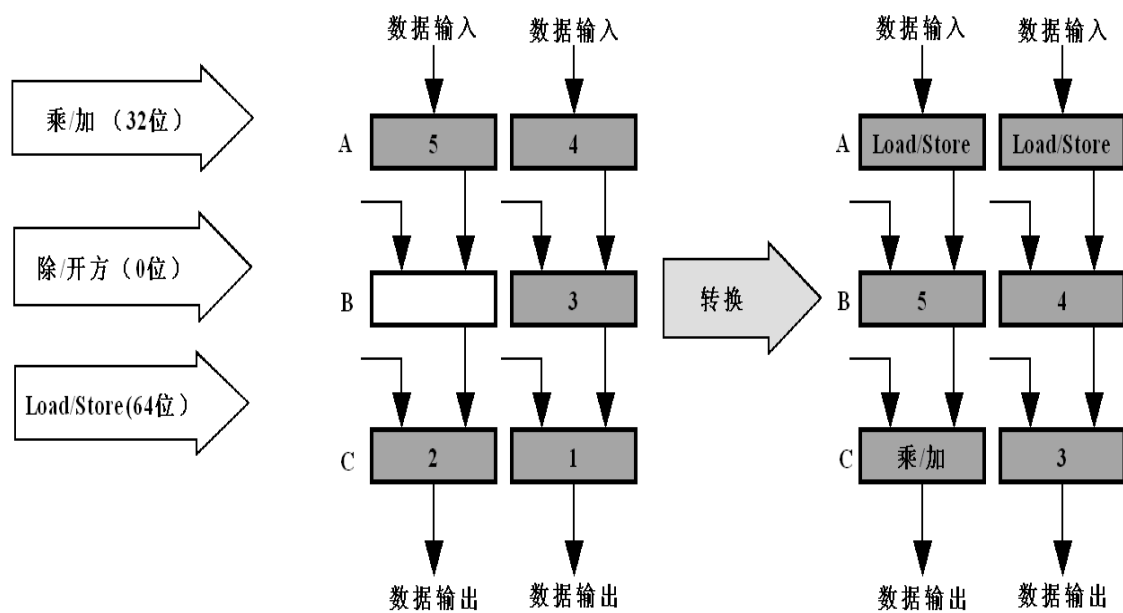


图6 缓冲队列中数据传递方式

由于采用了寄存器合并的方法，以缓冲队列中寄存器 C 为例，它可以存储一个 64 位双精度寄存器或两个 32 位数据，如果 C 中只有一个 32 位数据，它一定存储在低 32 位，则低 32 位中有效位 V 为 1，高 32 位有效位 C 为 0。如果有两个不相关的 32 位数据，高 32 位和低 32 位的有效位都为 1。当 C 中存储一个 64 位双精度数据时，我们让高 32 位有效位为 1，低 32 位有效位为 0，这时我们可以把 64 位当成一个整体处理，高 32 位中寄存器地址的 RN 的高 4 位为有效地址，这时寄存器号判断更简单。高、低 32 位的有效位所表示的意义如表 1 所示：

表 1. 高、低 32 位的有效位所表示的意义

高 32 位有效位	与低 32 位有效位	表示的意义
0	0	寄存器整体为空
0	1	只有一个 32 位单精度数据存储在低 32 位中
1	0	存储一个 64 位双精度数据
1	1	存储两个不相关 32 位单精度数据

2.5 缓冲队列中存储器个数使用分析

在本文中使用了3级的缓冲队列，每一级中包含了64位的数据位和12位的附加信息位。在某些应用中可以使用2级甚至1级的缓冲队列，例如有些协处理器使用的频率比较低或是对面积和功耗要求很高性能次之的微处理器都可以减少缓冲队列级数，这样可以减少该缓冲队列的面积和功耗，同时也简化了其控制逻辑。同理，对于流水线条数较多、性能要求较高的情况下可以增加缓冲队列的级数。

3. 结论:

本设计采用对多条流水线写寄存器堆时分配优先级，根据优先级的高低顺次写寄存器堆，并将暂时无法写入寄存器堆的数据存入到缓冲队列中，达到了减少寄存器堆读端口的目的，从而减少寄存器堆占用面积和功耗。将不能写入寄存器堆的数据存入了缓冲队列可以不影响后续指令的执行，在缓冲队列中的数据支持数据前推减少了后续指令因等待读取该数据所停滞的时间并且减少了对寄存器堆读端口的访问。在本设计中还使用了寄存器合并技术，将两个单精度数据合并写入寄存器堆从而加快了写寄存器堆的速度。最后分析了缓冲队列所需寄存器单元的数量。

参考文献:

1. R. P. Preston, R.W. Badeau, D.W. Bailey, et al. “ Design of an 8-wide superscalar RISC microprocessor with simultaneous multithreading”, IEEE International Solid-State Circuits Conference(ISSCC), Vol. 2, Page(s):266 – 500, February 2002.
2. J. Scott. “Designing the low - power Multicore architecture”. Power Driven micro- architecture Work shop at ISCA 98, Barcelona, Spain, June 1998.
3. G. S. Sohi, S. Breach, and T. N. Vijaykumar, “Multiscalar processors”. 22nd Annual International Symposium on Computer Architecture, Page(s):414 – 425, In ISCA-22, June 1995.
4. S. Palacharla, N. Jouppi, and J. E. Smith. “Complexity effective superscalar processors”. The 24th Annual International Symposium on Computer Architecture, Page(s):206 – 218, In ISCA-24 , page(s): 206–218, June 1997.
5. J. H. Tseng and K. Asanovi. “Banked Multiported Register Files for High-Frequency Superscalar Microprocessors”. 30th Annual International Symposium on Computer Architecture, page(s): 62 - 71, June 2003.

文章所述工作的背景:

该文章是本设计小组在进行“浮点协处理器”设计项目时为了减小协处理器的寄存器堆面积并提高其性能时所做的创新设计工作。我们参考了近年内减小寄存器堆面积的方法后提出了自己的创新方法。该方法现已实现，并达到了预期目标。

声明: 稿件内容属于作者的科研成果; 署名无争议; 引用他人成果已注明出处; 未公开发表过.