[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/38 (2006.01)
G06F 15/80 (2006.01)

[12] 发明专利申请公布说明书

[21] 申请号 200810144567.0

[43] 公开日 2009 年 2 月 25 日

[11] 公开号 CN 101373425A

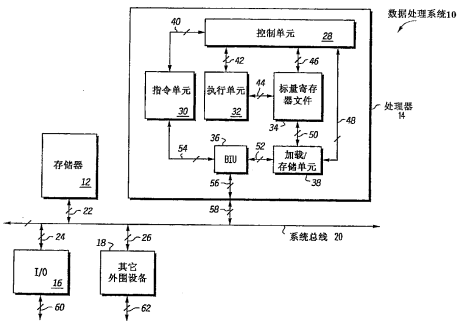
[22] 申请日 2004.8.30
[21] 申请号 200810144567.0
分案原申请号 200480024425.0
[30] 优先权
[32] 2003. 9. 8 [33] US [31] 10/657,797
[71] 申请人 飞思卡尔半导体公司
地址 美国得克萨斯
[72] 发明人 威廉·C·莫耶
詹姆斯·M·诺里斯
菲利普·E·马伊
肯特·唐纳德·莫特
雷蒙德·B·埃西克四世
布赖恩·杰夫里·卢卡斯
[74] 专利代理机构 中原信达知识产权代理有限责任公司
代理人 刘光明 穆德骏
权利要求书 4 页 说明书 55 页 附图 12 页

[54] 发明名称

用于执行 SIMD 运算的数据处理系统及其方法

[57] 摘要

提供一种用于执行 SIMD 运算的数据处理系统及其方法。可使用各种加载与存储指令在寄存器文件(元件 34)中的寄存器和存储器(元件 12)间传送多个矢量分量。可使用 cnt 参数表示将向或从存储器传送的分量的总数，并且可使用 rcnt 参数表示向或从寄存器文件中的单个寄存器传送的矢量分量的最大数量。另外，这些指令可使用各种不同的寻址模式。可独立于该寄存器分量大小指定该存储器分量大小，使得指令中的源与目标的大小可不相同。通过一些指令，可发起矢量流，并条件入队或出队。可提供截短或舍入字段，使得在传送时可截短或舍入源数据分量。另外，可在传送时对源数据分量进行有符号或无符号扩展。



1. 一种数据处理系统，其包括：

存储操作数的存储器；

至少一个通用寄存器；以及

用于执行至少第一条指令和该第一条指令之后的第二条指令的处理器电路，所述第一条指令发起要在至少一个通用寄存器中排队的来自存储器的数据分量流的传送，其中，所述第二条指令包括至少第一源操作数，并且所述处理器电路基于作为所述第二条指令的源操作数的所述至少一个通用寄存器，使来自存储器的数据分量流的下一部分有条件地加载到所述至少一个通用寄存器中。

2. 一种在数据处理系统中的方法，其包括：

提供存储操作数的存储器；

提供至少一个通用寄存器；以及

执行至少第一条指令；

发起要在至少一个通用寄存器中排队的来自存储器的数据分量流的传送，以响应所述第一条指令；

执行所述第一条指令之后的第二条指令，所述第二条指令包括至少第一源操作数；以及

根据作为所述第二条指令的源操作数的所述至少一个通用寄存器，使来自存储器的数据分量流的下一部分有条件地加载到所述至少一个通用寄存器中。

3. 一种数据处理系统，其包括：

存储操作数的存储器；

至少一个通用寄存器；以及

用于执行多条指令的处理器电路，所述多条指令中的第一条发起要在至少一个通用寄存器中排队的来自存储器的数据分量流的传送，并且，所述处理器电路基于对应于所述多条指令的适当子集中预定指

令的所述多条指令中的第二条，使来自存储器的数据分量流的下一部分有条件地加载到所述至少一个通用寄存器中。

4. 一种在数据处理系统中的方法，其包括：

提供存储操作数的存储器；

提供至少一个通用寄存器；以及

执行多条指令，所述多条指令的第一条发起要在至少一个通用寄存器中排队的来自存储器的数据分量流的传送；以及

基于对应于所述多条指令的适当子集中预定指令的所述多条指令中的第二条，有条件地加载数据分量流的下一部分。

5. 一种数据处理系统，其包括：

存储操作数的存储器；

至少一个通用寄存器；以及

用于执行多条指令的处理器电路，所述多条指令中的第一条发起要在存储器中排队的来自至少一个通用寄存器的数据分量流的传送，所述处理器电路基于对应于所述多条指令的适当子集中预定指令的所述多条指令中的第二条，使所述数据分量流的下一部分有条件地存储到存储器。

6. 一种在数据处理系统中的方法，其包括：

提供存储操作数的存储器；

提供至少一个通用寄存器；以及

执行多条指令，所述多条指令的第一条发起要在存储器中排队的来自至少一个通用寄存器的数据分量流的传送；以及

根据作为所述多条指令的第二条的目标操作数的所述至少一个通用寄存器，使所述数据分量流的下一部分有条件地存储到存储器。

7. 一种数据处理系统，其包括：

存储操作数的存储器；

至少一个通用寄存器；以及

用于执行至少第一条指令和该第一条指令之后的第二条指令的处理器电路，所述第一条指令发起要在存储器中排队的来自至少一个通用寄存器的数据分量流的传送，其中，所述第二条指令包括至少第一目标操作数，并且，所述处理器电路基于作为所述第二条指令的目标操作数的所述至少一个通用寄存器，使所述数据分量流的下一部分有条件地存储到存储器。

8. 一种数据处理系统，其包括：

存储操作数的存储器；

至少一个通用寄存器；以及

执行至少第一条指令和该第一条指令后的第二条指令的处理器电路，所述第一条指令发起要在至少一个通用寄存器中排队的来自存储器的数据分量流的传送，其中，所述第一条指令进一步指定将要传送的数据分量的数量，并且，所述处理器电路基于作为所述第二条指令的源操作数的所述至少一个通用寄存器，有条件地加载所述数据分量流的下一部分。

9. 一种数据处理系统，其包括：

存储操作数的存储器；

至少一个通用寄存器；以及

执行至少第一条指令和该第一条指令后的第二条指令的处理器电路，所述第一条指令发起要在存储器中排队的来自至少一个通用寄存器的数据分量流的传送，其中，所述第一条指令进一步指定将要传送的数据分量的数量，并且，所述处理器电路基于作为所述第二条指令的目标操作数的所述至少一个通用寄存器，使数据分量流的下一部分有条件地存储到存储器。

10. 一种数据处理系统，其包括：

存储操作数的存储器；

至少一个通用寄存器；以及

执行多条指令的处理器电路，所述多条指令的第一条发起在所述存储器和所述至少一个通用寄存器间的数据分量流的传送，这些数据分量将在所述存储器和所述至少一个通用寄存器中的至少一个中进行排队，并且，所述处理器电路有条件地执行使所述数据分量流的下一部分的存储和加载中的至少一个，以响应所述多条指令的第二条中的控制字段。

用于执行 SIMD 运算的数据处理系统及其方法

本申请是申请号为 200480024425.0，申请日为 2004 年 8 月 30 日的中国专利申请“用于执行 SIMD 运算的数据处理系统及其方法”的分案申请。

技术领域

本发明主要涉及数据处理系统，更具体地，涉及在数据处理系统中使用的指令。

背景技术

通过允许在矢量的多个分量上执行并行运算，可实现提高数据处理系统的性能。目前可用的一种处理器类型为矢量处理器，其利用执行矢量运算的矢量寄存器。然而，尽管矢量处理器允许更高的性能，但对于使用标量通用寄存器的处理器而言，增加了复杂度和成本。也就是，矢量处理器中的矢量寄存器文件通常包括 N 个矢量寄存器，其中每个矢量寄存器包括 M 个寄存器的库用于保存 M 个分量。例如，在本领域公知的一种矢量寄存器中，该寄存器文件包括 32 个矢量寄存器，其中，每个寄存器包括 32 个 64 位寄存器的库，从而等于需要 1024 个 64 位寄存器。

目前另一种可用的矢量寄存器是单指令多数据（SIMD）标量处理器（也称为“短矢量机”），其允许有限的矢量处理，同时使用任何现有的标量通用寄存器（GPR）。例如，在具有 32 个标量 64 位 GPR 的数据处理系统中，每一标量寄存器可以保存 2 个 32 位分量，4 个 16 位分量，或 8 个 8 位分量，从而能够执行 2 个 32 位分量运算，4 个 16 位分量运算，或 8 个 8 位分量运算。因此，与矢量处理器相比（根据上述示例，其每次可对寄存器库中的 32 个 64 位寄存器执行运算），

尽管每次运算的分量数目受到限制，但减少了所需的硬件。然而，在当前的 SIMD 标量处理器中，存在大量的与向标量寄存器传送用于执行的矢量分量和向存储器传送回多个矢量分量相关联的开销。该开销限制了运算的有效吞吐量，因此，存在对于在存储器与寄存器间加载和存储多个矢量分量的改良方法的需求。

附图说明

通过示例说明本发明，并不受限于附图，在附图中，相同的引用号码表示相同的元素，并且其中：

图 1 以框图的形式表示遵照本发明的一个实施例的数据处理系统；

图 2-11 表示可由图 1 的数据处理系统执行的矢量加载和存储指令，其遵照本发明的各种实施例；

图 12 表示示例存储器部分，其遵照本发明的一个实施例；

图 13-17 表示一组标量 GPR 中的条目的示例，其遵照本发明的各种实施例；

图 18 表示存储器部分随时间变化的示例，其遵照本发明的一个实施例；

图 19-26 表示一组标量 GPR 中条目的示例，其遵照本发明的各种实施例；

图 27-33 表示可由图 1 的数据处理系统执行的矢量加载和存储指令，其遵照本发明的替换实施例；

图 34 表示存储器部分的示例，其遵照本发明的一个实施例；和

图 35 表示一组标量 GPR 中条目的示例，其遵照本发明的一个实施例。

技术人员明白，图中的元件出于简化和说明的目的而表示，并不一定按比例绘制。例如，图中某些元件的尺寸相对于其它元件夸大，以有助于理解本发明的实施例。

具体实施方式

如此处所使用的，术语“总线”指用于传送一种或多种信息，例如，数据、地址、控制或状态的多个信号或导线（conductor）。此处讨论的导线可表示或说明为单个导线、多个导线、单向导线或双向导线。然而，不同的实施例可变化这些导线的实现方式。例如，可使用多个分离的单向导线，而不是双向导线，反之亦然。另外，可用以串行或时间复用方式传送信号的单个导线代替多个导线。同样地，承载多个信号的单个导线可分离为承载这些信号的子集的各种不同导线。因此，对于传送信号存在多个选项。

当将信号、状态位或类似的装置分别呈现其逻辑真或假状态时，使用术语“肯定”（assert）或“设置”（set）以及“否定”（negate）（或“非肯定”（deasert）或“清除”（clear））。如果该逻辑真状态为逻辑水平一，则逻辑假状态为逻辑水平零。并且如果逻辑水平真状态为逻辑水平零，则逻辑水平假状态为逻辑水平一。另外，在数字前的“0x”表明该数字以其 16 进制或 16 基底形式表示。数字前的“%”表明该数字以其二进制或 2 基底形式表示。

如上所述，存在对于降低从存储器向寄存器和从寄存器向存储器传送多个矢量分量的开销的需求。因此，此处将讨论多种不同的矢量加载和存储指令，其允许以改进的方式从或向存储器传送矢量分量。此外，使用此处说明的指令可允许降低的开销，向现有的设计增加最小的硬件复杂度。例如，如同在许多类型的数据处理系统中的使用，标量寄存器文件可与这些指令一同使用，从而无需成本更高的独立矢量寄存器文件。另外，这些指令可允许降低的软件复杂度，同时还提供执行效率。另外，此处的指令可用于解决各种问题，例如，存储器中矢量分量的对齐，将存储器分量的大小扩展至寄存器分量以提高的精确度的需要，存储器的任意矢量长度对于固定长度，或与检索/存储矢量数据所需的加载/存储指令数关联的开销（例如，在发出与执行指令时）。此处的这些指令还可用于解决这些问题的组合，或别的问题

或问题的组合，如通过在下面参考图 1-35 提供的说明将要理解的那样。

图 1 以框图的形式表示根据本发明的一个实施例的数据处理系统 10。数据处理系统 10 包括处理器 14、存储器 12、输入/输出（I/O）16、其它外围设备 18 以及系统总线 20。存储器 12 通过导线 22 双向连接至系统总线 20，I/O 16 通过导线 24 双向连接至系统总线 20，其它外围设备 18 通过导线 26 双向连接至系统总线 20，处理器 14 通过导线 58 连接至系统总线 20。在一个实施例中，其它外围设备 18 可包括一个或多个外围设备，其中，每一外围设备可为任何类型的外围设备，例如，通用异步收发器（UART），实时时钟（RTC），键盘控制器，其它存储器，等等。其它外围设备 18 的一些或全部能够通过导线 62 传送数据处理系统 10 外部的信息。I/O 电路 16 可包括任何类型的 I/O 电路，其通过例如导线 60，接收或提供数据处理系统 10 的外部信息。存储器 12 可为任何类型的存储器，例如，只读存储器（ROM），随机存取存储器（RAM），非易失性存储器（例如，flash）等。数据处理系统 10 可包括除表示的那些之外的其它元件，或可包括比表示出的那些更多或更少的元件。例如，数据处理系统 10 可包括任意数量的存储器或处理器。

处理器 14 可为任何类型的处理器，例如，微处理器、微控制器、数字信号处理器，等等。在一个实施例中，处理器 14 可称为处理器内核。在另一实施例中，处理器 14 可为多处理器数据处理系统中多个处理器中的一个。另外，尽管没有这样表示，但处理器 14 可为流水线处理器。在图 1 中所示的实施例中，处理器 14 包括控制单元 28、指令单元 30、执行单元 32、标量寄存器文件 34、总线接口单元（BIU）36 以及加载/存储单元 38。控制单元 28 通过导线 40 双向连接至指令单元 30，通过导线 42 连接至执行单元 32，通过导线 46 连接至标量寄存器文件 34，并且通过导线 48 连接至加载/存储单元 38。执行单元 32 通过导线 44 双向连接至标量寄存器文件 34，并且标量寄存器文件 34 通过导线 50 双向连接至加载/存储单元 38。BIU 36 通过导线 54 双向连接至指令

单元 30，并且通过导线 52 连接至加载/存储单元 38。处理器 14 通过连接至导线 58 的导线 56，能够与系统总线 20 双向通信。注意，处理器 14 可包括比表示出的更多的电路，其中，附加电路可连接至导线 58。也就是，导线 56 可通过全部或部分导线 58 与系统总线 20 通信。还要注意，全部或部分处理器 14 可称为处理电路。

在操作中，指令单元 30 通过 BIU 36 和系统总线 20 从例如存储器 12 的存储器取回指令，并从或向控制单元 28 接收和提供控制信息。指令单元 30 可为本领域所公知的任何类型的指令单元，并如本领域所公知的那样操作，并且因此不在此处详细说明。指令单元 30 因此向控制单元 28 提供指令，控制单元通过，例如，执行单元 32 和加载/存储单元 38，控制这些接收的指令的执行，执行单元 32 和加载/存储单元 38 这两个单元可在必要时直接地或通过控制单元 28 与标量寄存器文件 34 通信。例如，控制单元 28 通过加载/存储单元 38 和 BIU 36，能在需要执行指令时从存储器（例如存储器 12）向标量寄存器文件 34 中的寄存器加载数据，并且能在需要执行指令时将来自寄存器文件 34 中寄存器的数据存储至存储器（例如存储器 12）。例如，在一个实施例中，加载/存储单元 38 能够通过导线 50，基于控制单元 28 通过导线 48 提供的控制信息，直接与标量寄存器文件 34 通信（以读取和写入数据）。执行单元 32 可在需要执行通过控制单元 28 从指令单元 30 接收的指令时，使用在标量寄存器文件 34 中存储的数据，执行算术、逻辑、移位或其它运算操作，并将结果存储至标量寄存器 34 中的寄存器。执行单元 32 可包括，例如，算术逻辑单元（ALU）、浮点单元，等等。

标量寄存器文件 34 包括 N 个通用寄存器（GPR），其中，N 可为大于或等于一的任何整数。在一个实施例中，标量寄存器文件 34 包括 32 个 64 位寄存器。如同此处所使用，标量寄存器表示具有一维映射并因此仅保存一行数据（例如，1×M 位寄存器）的寄存器，其中，M 为大于或等于一的任何整数。在一个实施例中，M 为 64，并且每个寄存器因此能够存储 64 位的量。标量寄存器文件 34 可通过导线 46 向或从

控制单元 28 提供或接收控制信息。

处理器 14 的操作通常可由本领域的普通技术人员理解。因此在此处将不详细说明处理器 14，除了在理解参考图 2-36 说明的各种实施例中需要的那些部分外。还要注意，可在需要执行此处说明的加载与存储指令时修改具有存储在通用寄存器文件中的操作数的数据处理系统的现有设计。此外，注意，由于可使用标量寄存器文件，可更新目前的现有设计，以允许此处说明的指令。（然而，此处说明的实施例可与任何类型的寄存器文件一同使用，并不仅限于标量寄存器文件。）

图 2-11 表示各种不同的矢量分量加载与存储指令，其可由处理电路执行，例如，由图 1 的处理器 14 等执行。例如，每一指令可由指令单元 30 取回，其在适当时将它们提供至控制单元 28。控制单元 28 因此可在需要时指示加载/存储单元 38 和执行单元 28 执行接收的指令，在需要存储数据时使用寄存器文件 34，如下详述。注意，参考后面的图，提供使用在图 2-11 中表示的指令的示例。另外，注意，如此处所使用，矢量分量（或寄存器分量）指一个最多为标量 GPR 大小，但也可以小于 GPR 大小的分量。例如，在标量寄存器文件 34 包括 64 位寄存器（M=64）的情况下，矢量分量可为 64 位的大小或更小。例如，矢量分量可为一个字节（8 位），使得一个 64 位 GPR 能够保存 8 个矢量分量。另外，矢量分量可为半字（16 位），使得一个 64 位 GPR 能够保存 4 个矢量分量。类似地，矢量分量可为字（32 位），使得 64 位 GPR 可保存 2 个分量。还要注意，如此处所使用，字节由“b”表示，半字由“h”表示，并且字由“w”表示。（注意，在替换实施例中，字或半字的定义可能不同。例如，字可以指 16 位，而不是 32 位。然而，如此处所使用，为了便于说明，字指 32 位。）

图 2 表示加载矢量分量（lvex）指令，其用来自存储器的分量加载该寄存器文件（例如，标量寄存器文件 34）中的一个目标寄存器。在一个实施例中，可利用附加控制字段进一步限定该操作，例如 s/u 字段、

ms 字段以及 ds 字段，在图 2 中将每一个说明为紧随操作符“lvex”，并由句点分隔。在一个实施例中，具有这些控制字段中的每一个；然而在替换实施例中，可能存在这些控制字段中的一些或者完全不存在。在一个实施例中，ms 字段对应于存储器分量大小字段，ds 字段对应于目标分量大小。例如，在一个实施例中，该存储器分量大小（ms）字段可设置为 mb、mh 或 mw 中的一个，以分别表明该将要加载的存储器分量是否为字节、半字或字。类似地，目标分量大小（ds）字段用于表示在目标寄存器中每一加载的分量将具有的大小。

图 2 的 lvex 指令还包括三个操作数：rD、rA 以及 rB。该指令 lvex 利用来自存储器的矢量分量加载目标寄存器 rD。寄存器 rA 指向存储器中的第一个分量，并且在图 2 的实施例中，在寄存器 rB 中指定附加参数。因此，每一 rD、rA 和 rB 可表明标量寄存器文件 34 中 GPR 中的一个。在寄存器 rB 中存储的附加参数包括计数（cnt）值和跨度（stride）值，使得 lvex 指令在具有 ds 大小的 rD 字段中用来自存储器的“cnt”个分量（由 rA 指向这些分量的第一个）加载寄存器 rD，使得 rD 中的每一目标分量具有 ds 大小。

还可指定跨度值，其指示存储器中的分量如何分隔。也就是，存储器中的这些分量距离“stride”个大小 ms 的单位。例如，如果存储器中的分量相邻，则可使用跨度 1 得到 “cnt”个大小为“ms” 的相邻分量。例如，如果这些分量距离三个分量放置，则可使用跨度 3 获得“cnt”个元素，彼此间距离 3 个“ms”大小的单位。例如，如果 ms 表示字（例如，mw），则跨度 3 将在存储器中从 rA 所指的存储器位置开始每三个字取一个字。另外，该跨度值可为有符号值，其中，负值将向低位地址位置取分量，不是向高位地址位置取分量。

在一个实施例中，目标大小（ds）应当大于或等于存储器分量大小（ms）。如果该目标大小（ds）大于该存储器大小（ms），则 s/u 字段可用于适当地扩展每个存储器分量。“s”用于表示这些分量是有

符号的，并且应该因此能够进行符号扩展，而“u”可用于表示这些分量是无符号的，并且应当因此能够进行零扩展。例如，在“lvex.u.mh.dw rD,rA,rB”的情况下，rD 中（字大小）的每一目标位置的上半字可归零，如下面详细说明。另外，注意，在一个实施例中，如果“cnt*ds”小于目标寄存器 rD 的大小，则 rD 剩余的低阶部分归零。可替换地，rD 的任何剩余部分，例如，rD 的剩余的低阶部分，可由任何预定值填充（例如，零或任何其它预定值）。另外，在一个实施例中，如果 ds 小于 ms，则在加载至 rD 时，可对每一分量进行舍入或截短（其中，该 lvex 指令还可包括截短或舍入控制字段，如下对于存储指令的说明）。

在说明的实施例中，可将 s/u 字段、ms 字段以及 ds 字段编码为操作码或子操作码的部分。因此，在说明的实施例中，该 lvex 指令包括表示该操作码的字段、表示 rD 的字段、表示 rA 的字段、表示 rB 的字段以及表示子操作码的字段。然而，在替换实施例中，注意，s/u、ms 以及 ds 字段还可在 rB 中连同 cnt 和 stride（跨度）一起表示。

图 3 表示图 2 的 lvex 指令的替换实施例，其中不是将 rB 表示为操作数，其中，rB 包括附加参数，而是作为该指令的操作码和子操作码的全部或部分提供所有这些参数。例如，可在该指令的子操作码部分提供 cnt、stride 以及控制字段（s/u，ms 和 ds）。

注意，上面提供的对于控制字段 s/u、ms 和 ds 的说明适用于所有此处说明的加载指令。在使用该指令 lvex 时，在此处说明的其它加载指令中，可存在这些控制字段的全部、一些或是完全不存在。还需要注意，对于此处的讨论，rD 通常指目标寄存器，rA 指向存储器中第一个存储器分量，并且 rB，如果在该指令中存在的话，存储该指令的一些或全部附加参数。注意，rB 还可称为控制寄存器，并且不必是 GPR 中的一个。类似地，rA 可为控制寄存器或另一存储器位置，并且不必是 GPR 中的一个。还要注意，在此处说明的所有指令（加载与存储）中，这些控制字段可编码为该指令的操作码或子操作码的一部分或可

位于寄存器 rB 中。如果它们位于寄存器 rB 中，则这些控制字段通常不出现在操作符之后，并由句点分隔，而将仅出现在寄存器 rB 中。还要注意，如果需要更多的参数或控制字段，此处的任何指令可表示，如必要的话，多于一个的寄存器，用于表示指令参数和控制字段，或者在共同的或不相连的存储位置中可包含附加信息。

图 4 表示加载多个矢量分量（lmvex）指令，其用来自存储器的分量加载多个目标寄存器。在使用该 lmvex 指令时，该 lmvex 指令还可包括 s/u 字段、ms 字段，以及 ds 字段，每一个在图 4 中表示为紧随操作符“lmvex”，并由句点分隔。图 4 的 lmvex 指令还包括三个操作数：rD，rA 和 rB。该 lmvex 指令用“cnt”个来自存储器的矢量分量加载第一目标寄存器 rD，和所需的多个跟随 rD 之后的寄存器（其中，寄存器 rA 指向存储器中的第一个分量），使得任何目标寄存器中的每一目标分量的大小为 ds。例如，如果 9 个半字（16 位）大小的分量将加载至 rD（其中，每一 GPR 为 64 位），则需要 3 个 GPR。在一个实施例中，rD 表示第一个寄存器，并且如果需要更多，则使用额外的相邻寄存器。例如，如果 rD 表示该 9 个分量的寄存器 R4，则使用寄存器 R4，R5 和 R6。

在寄存器 rB 中存储的附加参数包括 cnt、stride（跨度）、skip（跳跃）和 skip_cnt。参数 cnt 与 stride 与上述参考图 1 的说明相同。也就是，存储器中将要加载的分量可距离“stride”个 ms 大小的单位放置。该 lmvex 指令从存储器距离“stride”个（ms 大小的）单位加载分量，直到取回“skip_cnt”个分量。此时，下一个分量位于“skip”个 ms 大小的单位处，并且重置该记录 skip_cnt 的机制（例如，计数器）。因此，每次取回 skip_cnt 个分量时（从该指令的起始或从上次重置 skip_cnt 计数器），下一分量就位于“skip”（而不是“stride”）个 ms 大小的单位处。在一个实施例中，处理器 14（例如，控制单元 28 中）中的计数器设置为“skip_cnt-1”，并在每次取数后递减，直到该计数器终止（到达零），并再次设置为“skip_cnt-1”，以继续取回位于“skip”个

单位处的分量，直到取回“cnt”个分量。（可替换地，可将该计数器设置为零，并逐次增加，直到该计数器到达 skip_cnt-1，替换实施例可使用其它方法记录 skip_cnt。）注意，stride 和 skip 均可为有符号量。另外，在一个实施例中，在从存储器加载“cnt”个分量后，最后一个寄存器中任何剩下的未填充字节用预定值填充（例如，零）。

在一个实施例中，该目标大小（ds）应大于或等于该存储器分量大小（ms）。如果该目标大小（ds）大于该存储器分量大小（ms），则该 s/u 字段可用于适当地扩展每一存储器分量，如上对于 lvex 的说明。然而，在一个实施例中，如果 ds 小于 ms，则每个分量可在加载至 rD 时舍入或截短，如下对于存储指令的说明。

图 5 表示图 4 的 lmvex 指令的替换实施例，其中，不是将 rB 表示为操作数，其中 rB 包括附加参数，而是可将所有这些参数作为该指令子操作码的全部或部分提供。例如，cnt、stride、skip、skip_cnt 以及控制字段（s/u、ms 和 ds）可全部在该指令的子操作码或操作码部分指示。但在另一替换实施例中（未示出），各个指令参数可分配至指令字段或 rB 中的字段。例如，当希望使该参数能够基于计算结果或基于系统状态动态修改时，可使用分配至 rB 中的字段。其它参数可为静态特性，并且可编码至该指令中的字段。

注意，对于此处说明的其余字段，任何附加参数将位于 rB 中，其中 rB 作为操作数提供至该指令。然而，应当理解，在替换实施例中，该指令可具有不同的格式，如上所述。

图 6 表示另一加载多矢量分量（lmvex2）指令，其也用来自存储器的分量加载多个目标寄存器。在使用上述指令时，该 lmvex2 指令还可包括 s/u 字段、ms 字段和 ds 字段，每个在图 6 中表示为跟随操作符“lmvex2”，并由句点分隔。图 6 的 lmvex2 还包括三个操作数：rD，rA 和 rB。该指令 lmvex2 用来自存储器的“cnt”个矢量分量（其中，

寄存器 rA 指向存储器中的第一个分量）加载该目标寄存器 rD（和所需的多个跟随 rD 的寄存器），使得任一目标寄存器中的每一目标分量的大小为 ds。例如，如果 9 个半字（16 位）的分量将加载至 rD（其中，每一 GPR 为 64 位），则至少需要 3 个 GPR，并且如进一步所述，则还需要超过 3 个的附加 GPR。在一个实施例中，rD 表示该第一个寄存器，并且如果需要更多，则使用附加的连续寄存器。例如，如果 rD 表示这 9 个分量的寄存器 R4，则至少使用寄存器 R4，R5 和 R6。

在寄存器 rB 中存储的用于 lmvex2 的附加参数包括 cnt、rcnt、stride 和 skip。参数 cnt 和 stride 如同上面对于 lmvex 的说明。也就是，存储器中将要加载的分量可位于“stride”个 ms 大小的单位处，并且 cnt 定义要加载的分量的总数。该 rcnt 参数定义将加载至每一寄存器中目标分量的最大数。（因此，所需的寄存器总数取决于 cnt 和 rcnt。）该 lmvex2 指令从存储器加载总共“cnt”个分量至一个或更多目标寄存器（按照需要的数量），其中，仅有“rcnt”个分量加载至每一目标寄存器。（因此，在一个实施例中，rcnt 小于或等于可包含在单个 GPR 中的 ds 大小的目标分量的总数，并且 rcnt 还小于或等于 cnt。）

例如，该 lmvex2 指令从存储器间隔“stride”个（ms 大小的）单位加载分量至目标寄存器（rD），直到取回“rcnt”个分量至单个寄存器。此时，如果还没有取回总共“cnt”个分量，则下一个“分量”位于“skip”个 ms 大小的单位处。来自存储器相隔“stride”个（ds 大小的）单位位置的分量然后加载至另一目标寄存器（例如，rD＋1 等），直到取回“rcnt”个分量至该目标寄存器（或者直到取回“cnt”个分量，无论哪一个先发生）。在取回“rcnt”分量后，如果还没有取回“cnt”个分量，则继续将位于“skip”个单位处的下一个分量加载至，例如 rD＋2。当已经取回“cnt”个分量时，加载停止。因此注意，存储器中对于给定目标寄存器的分量距离“stride”个 ms 大小的单位，在将“skip”个 ms 大小的单位加至前一目标寄存器的最后一个分量的地址后，取回后面寄存器的分量。还要注意，stride 和 skip 均为有符号的量，允许负

stride 和 skip。

注意，在替换实施例中，可使用 skip_cnt 参数，使得在取回 skip_cnt 个分量后，下一分量位于“skip”个 ms 大小的单位处。按照这种方式，在将“rcnt”个分量载入至目标寄存器中后，如果已取回 skip_cnt 个分量，或者基于 stride，如果从该指令的起始或从上次重置 skip_cnt 以来还没有加载 skip_cnt 个分量，则基于 skip 取回后面寄存器的下一个分量。

如上对于 lvex 的说明，处理器 14 中的计数器（例如，在控制单元 28 中）可用于在加载矢量分量期间记录“cnt”。类似地，处理器 14 中的计数器还可用于在加载每一目标寄存器期间记录“rcnt”。替换实施例可使用其它方法记录 cnt 和 rcnt。另外，在一个实施例中，在取回“rcnt”个分量之后，每一目标寄存器中的其余字节由预定值填充，例如零等。另外，在一个实施例中，在从存储器加载“cnt”个分量后，所使用的最后一个寄存器中的剩余字节用预定值填充（例如，零）。

在一个实施例中，该目标大小（ds）应大于或等于该存储器分量大小（ms）。如果该目标大小（ds）大于该存储器分量大小（ms），则该 s/u 字段可用于适当地扩展每一存储器分量，如同对于上面指令的上述说明。然而，在一个实施例中，如果 ds 小于 ms，则在加载至 rD 时，每一分量可被舍入或截短，如下面对于该存储指令的说明。

图 7 表示加载矢量分量流（lstrmvex）指令，其发起从存储器向与目标寄存器关联的指定流加载矢量分量流。在使用上述指令时，该 lstrmvex 指令也可包括 s/u 字段、ms 字段和 ds 字段，每一个在图 7 中表示为跟随操作符“lstrmvex”，并由句点分隔。图 7 的 lstrmvex 指令还包括三个操作数：rD，rA 和 rB。lstrmvex 的附加参数还包括 cnt、rcnt、stride、skip 和 skip_cnt。参数 cnt、rcnt、stride、skip 和 skip_cnt 通常与对于 lvex，lmvex 和 lmvex2 的上述说明相同。

lstrmvex 指令用来自存储器的总共“cnt”个分量加载多个 rD 的实例（其中，寄存器 rA 指向存储器中的第一个分量），使得该目标寄存器中的每一目标分量的大小为 ds。cnt 和 rcnt 的值确定目标 rD 实例的数量。该 lstrmvex 指令从间隔“stride”个 ms 大小的单位的存储器加载矢量分量，直到取回“skip_cnt”个分量。此时，将“skip * ms”添加至最后一个取回的地址（而不是“stride * ms”），以获得将要加载的下一分量的地址。另外，此时重置记录 skip_cnt 的机制（例如，计数器）。因此，每次取回“skip_cnt”个分量时（自从该指令的起始或从上次重置该 skip_cnt 计数器），由“skip”，而不是“stride”参数指示下一分量的位置。

该“rcnt”参数定义每寄存器实例的目标分量数。（因此，在一个实施例中，rcnt 小于或等于单个 GPR 中 ds 大小的目标分量的总数，并且 rcnt 小于或等于 cnt。）例如，该 lstrmvex 指令从存储器（根据参数 stride、skip 以及 skip_cnt，如上所述）加载分量至 rD（即，rD 的第一个实例），直到取回“rcnt”个分量。此时，如果还没有取回“cnt”个分量，则下一分量位于“stride”个 ms 大小的单位处（或者，如果取回 skip_cnt 分量，则下一分量位于“skip”个，而不是“stride”个 ms 大小的单位处）。该 lstrmvex 指令然后继续从存储器（根据参数 stride、skip 和 skip_cnt）向不同的 rD 实例（即，rD 的第二个实例，覆盖以前的 rD 值）加载分量，直到取回“rcnt”个分量（或直到取回“cnt”个分量，无论哪一个先发生）。如果，在取回“rcnt”个分量后，并还没有取回“cnt”个分量，则继续加载下一分量至 rD（rD 的另一实例）。当取回“cnt”个分量时，加载停止。rD 的每一实例对应于从存储器取回的 rcnt 个分量的唯一集合。随着 rD 每一实例的出队，用分量流中的下一 rcnt 个分量集合加载下一实例，直到加载该流的总共 cnt 个分量。

在一个实施例中，rD 的实例基于后面指令中 rD 的使用或位置出队。例如，在一个实施例中，如果 rD 用作后面指令中特定源寄存器，

则 rD 实例出队。因此，在执行后面指令后（rD 作为特定源寄存器），将在 rD 中加载下一个“rcnt”个分量（假设还没有达到“cnt”个分量）。然而，如果 rD 没有用作特定源寄存器（即，如果其为不同的目标寄存器，或其为目标寄存器），则 rD 不出队，并且不将下一“rcnt”个分量加载至 rD。在替换实施例中，以其它方式指示是否出队的选项，例如，通过指令中的标志，或通过不同的操作码等（例如，对应于要求出队的指令的特殊子集等）。可选出队的示例将在后面参考图 23-26 说明。

如上对于上面指令的说明，处理器 14 中的计数器（例如，在控制单元 28 中）可在加载矢量分量期间用于记录“cnt”。类似地，处理器 14 中的寄存器还可在加载每一目标寄存器期间记录“rcnt”。计数器还可用于记录 skip_cnt。替换实施例可使用其它方法记录 cnt，rcnt，和 skip_cnt。另外，在一个实施例中，在加载“rcnt”个分量后，每一目标寄存器或寄存器实例中的任何剩余字节由预定值填充，例如零。另外，在一个实施例中，从存储器加载“cnt”个分量后，所使用的最后一个目标寄存器或寄存器实例中的剩余字节用预定值填充（例如零）。还要注意，stride 和 skip 可为有符号量，允许负的 stride 和 skip。

在一个实施例中，目标大小（ds）应大于或等于存储器分量大小（ms）。如果该目标大小（ds）大于该存储器分量大小（ms），则 s/u 字段可用于适当地拓展每一存储器分量，如对于上面指令的上述说明。然而，在一个实施例中，如果 ds 小于 ms，则在加载至 rD 时，可舍入或截短每一分量。

注意，在替换实施例中，lstrmvex 指令的各个参数的每一个可为可选的，并且此处说明的其它指令的替换实施例也可能如此，例如 lvex、lmvex 和 lmvex2 指令。

图 8 表示存储矢量分量（stvex）指令，其从源寄存器向存储器存

储分量。该 stvex 指令可认为是 lvex 指令的对应。在一个实施例中，该 stvex 指令使用附加控制字段，例如，s/u 字段、ms 字段、ss 字段和 h/l 字段，在图 8 中将每一个表示为跟随操作符“stvex“，并由句点分隔。在一个实施例中，这些控制字段的每一个都存在；然而，在替换实施例中，仅存在这些控制字段中的一些，或者完全没有。在一个实施例中，s/u 字段（有符号或无符号扩展）和 ms 字段（存储器大小）与上面对于加载指令的说明相同。在一个实施例中，ss 字段对应于源分量大小字段，其指示源寄存器中每一源分量的大小。例如，在一个实施例中，该 ss 字段可设置为 sb、sh 或 sw 之一，以分别指示将存储至存储器的源分量是否为字节、半字或字。

图 8 的 stvex 指令还包括三个操作数：rS、rA 和 rB。在图 8 的实施例中，在寄存器 rB 中指定了附加参数，其中这些附加参数包括 cnt 和 stride。（因此，在使用上面的加载指令时，每一 rS、rA 和 rB 可指示标量寄存器 34 中 GPR 中的一个。）该 stvex 指令从源寄存器 rS 向存储器存储“cnt”个 ss 大小的分量，其中，第一个分量将存储在由寄存器 rA 指向的存储器位置，使得存储器中的每一目标分量具有 ms 大小（并且可选地截短或拓展，如 s/u 字段和 h/l 字段指示）。

还可指定跨度（stride）值，其指示存储器中的分量如何分隔。也就是，在存储时，存储器中的分量将距离“stride”个 ms 大小的单位。例如，跨度 1 可用于存储“cnt”个相邻的“ms”大小的分量。类似地，跨度 3 可用于向存储器存储“cnt”个“ms”大小的分量，每一个彼此间距离 3 个“ms”大小的单位。另外，注意，该跨度值可为有符号值。

在一个实施例中，h/l 字段为截短控制字段。例如，如果 ss 大于 ms（即，源大小大于存储器大小），则该 h/l 字段可用于指示如何截短该源分量。例如，如果该 h/l 字段为“h”，则截短该源分量的低阶部分。然而，如果该 h/l 字段为“l”，则截短该源分量的高阶部分。注意，在替换实施例中，stvex 指令还包括舍入控制字段（除了或替换该截短

控制字段），使得如果，例如，ss 大于 ms，则对各值进行舍入，而不是截短。例如，在一个实施例中，可基于源分量的低阶部分，舍入该源分量的高阶部分。

在一个实施例中，如果该源分量大小（ss）小于该存储器分量大小（ms），则该 s/u 字段可用于在存储至存储器中时适当地扩展每一源分量。可使用“s”指示分量是有符号的，并且因此应当进行符号扩展，而使用“u”指示分量是无符号的，并因此应当进行零扩展。

在表示的实施例中，此处说明的任何存储指令的 s/u 字段、ms 字段，ss 字段和 h/l 字段可编码为操作码或子操作码的一部分。因此，在表示的实施例中，stvex 指令包括指示该操作码的字段、指示 rS 的字段、指示 rA 的字段、指示 rB 的字段和指示子操作码的字段。然而，在替换实施例中，注意，s/u、h/l、ms、和 ss 字段，连同 cnt 和 stide，还可在 rB 中指示。另外，在使用该加载指令时，每一附加参数还可编码为操作码或子操作码的一部分，无需指定寄存器，例如 rB。因此注意，在替换实施例中，此处说明的任何存储指令可具有各种不同的格式。另外注意，将参考在寄存器 rB 中存储附加参数说明该存储指令。（如上所述，如果需要附加参数和控制字段，则在该指令中可指示多个寄存器或存储位置。）

注意，上面提供的对于控制字段 s/u、ms、ss 和 h/l 的说明适用于此处说明的所有存储指令。在使用指令 stvex 时，在此处说明的其它存储指令中可存在这些控制字段中的全部，一些，或完全不存在。还要注意，对于此处的讨论，rS 通常指源寄存器，rA 指向将要存储第一个分量的存储器位置，并且 rB，如果在该指令中存在的话，保存该指令的一些或全部附加参数。

图 9 表示存储多矢量分量（strmvex）指令，其从多个源寄存器向存储器存储分量。该 stmvex 指令可认为是 lmvex 指令的存储对应。在

一个实施例中，该 stmvex 指令使用附加控制字段，例如 s/u 字段、ss 字段、ds 字段和 h/l 字段，每一个在图 9 中表示为跟随操作符“stmvex”，并由句点分隔。上面提供的对于 stvex 指令的这些字段的说明也适用于该 stmvex 指令。

图 9 的 stmvex 指令还包括三个操作数：rS，rA 和 rB。在图 9 的实施例中，在寄存器 rB 中指定附加参数。其中，这些附加的参数包括 cnt、stride、skip 和 skip_cnt。该 stmvex 指令从多个源寄存器 rS（以及 rS 之后的寄存器，例如，rS+1，rS+2，按照需要）向存储器存储“cnt”个 ss 大小的分量，其中，寄存器 rA 指向将要存储的第一个分量的地址，并且使得存储器中的每一分量具有 ms 的大小（并且可选地截短或扩展，根据 s/u 和 h/l 字段的指示）。

在使用 stmvex 时，跨度值可指示如何分隔存储器中的分量。因此，来自 rS 的源分量存储至存储器，使得它们距离“stride”个（ms 大小的）单位放置，直到存储了“skip_cnt”个分量。此时，向最后一个存储的分量的地址增加“skip*ms”（而不是“stride*ms”），以获得将要存储的下一个分量的地址。另外，此时，重置用于记录 skip_cnt 的机制（例如，计数器）。因此，每次存储“skip_cnt”个分量时（从该指令的开始或从上次重置 skip_cnt 计数器），将要存储的下一个分量的地址由 skip 参数指示，而不是由 stride 参数指示。在将来自 rS 的源分量存储至存储器时，将来自下一源寄存器的源分量（例如，rS+1 等）基于 stride、skip 和 skip_cnt 参数存储至存储器，直到存储了“cnt”个分量。

如上对于加载指令的说明，处理器 14 中的计数器（例如控制单元 28 中）可用于适当地记录 cnt 和 skip_cnt。例如，计数器可用于指示 skip_cnt 个分量，其中，在存储了 skip_cnt 个分量后，重置该计数器，并且将下一 skip_cnt 个分量存储在起始处由 skip 指示的位置。继续如此，例如，直到存储了 cnt 个分量（其可由，例如，另一计数器指示）。

替换实施例可使用其它方法记录 cnt 和 skip_cnt。

注意，在一个实施例中，如果 ss 小于 ms，则可适当地扩展每一源分量，如同 s/u 字段（如对于 stvex 的说明）所指示，以生成存储器目标分量。另外，在一个实施例中，如果 ss 大于 ms，则适当地截短每一源分量，如同 h/l 字段（如对于 stvex 的说明）所指示，以生成存储器目标分量。

图 10 表示另一存储多矢量分量（stmvex2）指令，其从多个目标寄存器向存储器存储分量。该 stmvex2 指令可视为 lmvex2 指令的存储对应。在一个实施例中，该 stmvex2 指令使用附加控制参数，例如 s/u 字段、ss 字段、ds 字段和 h/l 字段，每一个在图 10 中表示为跟随操作符“stmvex2”，并由句点分隔。上面对于 stvex 和 stmvex 指令的这些控制字段提供的说明也适用于 stmvex2 指令。

图 10 的 stmvex2 指令还包括三个操作数：rS、rA 和 rB。在图 10 的实施例中，在 rB 中指定附加参数，其中，这些附加参数包括 cnt、rcnt、stride 和 skip。该 stmvex2 指令从多个源寄存器 rS（以及 rS 之后的寄存器，例如，rS+1、rS+2，按照需要）向存储器存储“cnt”个 ss 大小的分量，其中，寄存器 rA 指向将要存储的第一个分量的地址，并且使得存储器中的每一分量具有 ms 的大小（并且可选地截短或扩展，如 s/u 和 h/l 字段所指示）。该 rcnt 参数定义将要存储的每个源寄存器的源分量的数量，并且该 cnt 参数定义将要存储的分量的总数。

在使用 stmvex2 时，跨度（stride）值可指示如何分隔存储器中的分量。因此，来自 rS 的源分量存储至存储器，使得它们距离“stride”个（ms 大小的）单位放置。在一个实施例中，在存储“rcnt”个来自给定源寄存器（rS）的分量后，如果要存储其余的分量，则向最后一个存储的分量的地址增加“skip*ms”（而不是“stride*ms”），以获得将要存储的下一个分量的地址。然后将来自下一个源寄存器（例如，

rS+1 等）的“rcnt”个源分量利用所指示的“stride”值存储至存储器。继续如此，直到存储了“cnt”个分量。注意，在一个实施例中，如“rcnt*ss”小于该源寄存器的大小，则忽略每一源寄存器的其余低阶部分。还要注意，“stride”和“skip”可为有符号的量，从而允许负的 stride 和 skip。

注意，在替换实施例中，还可使用 skip_cnt 参数，使得在存储了 skip_cnt 个分量之后，下一个分量位于“skip”个 ms 大小的单位处（而不是“stride”个 ms 大小的单位处）。按照这种方式，在向存储器存储了“rcnt”个分量后，如果存储了 skip_cnt 个分量，或者基于 stride，如果自从该指令的起始或从上次重置 skip_cnt 以来没有存储 skip_cnt 个分量，则基于 skip 将来自后面寄存器的下一个分量存储至存储器中的位置。

如上对于该加载指令的说明，处理器 14 中的计数器（例如，控制单元 28 中）可用于适当地记录 cnt 和 rcnt。替换实施例可使用其它方法记录 cnt 和 rcnt。

注意，在一个实施例中，如果 ss 小于 ms，则适当扩展每一源分量，如同 s/u 字段所指示（如对于 stvex 的说明），以生成存储器目标分量。另外，在一个实施例中，如果 ss 大于 ms，则适当地截短每一源分量，如同 h/l 字段所指示（如同对于 stvex 和 stmvex 的说明），以生成存储器目标分量。还可以可选地指定舍入，如前对于 stvex 指令的说明。

图 11 表示存储矢量分量流（ststrmvex）指令，其发起从指定流向存储器存储矢量分量流。该 ststrmvex 指令可视为 lstrmvex 指令的存储对应。在一个实施例中，该 ststrmvex 指令使用附加控制字段，例如 s/u 字段、ss 字段、ds 字段和 h/l 字段，每一个在图 11 中表示为跟随操作符“ststrmvex”，并由句点分隔。对于 stvex、stmvex 和 stmvex2 指令的这些字段的说明在这里也适用于 ststrmvex 指令。

图 11 的 ststrmvex 指令还包括三个操作数：rS、rA 和 rB。在图 11 的实施例中，附加参数在寄存器 rB 中指定，其中，这些附加参数包括 cnt、rcnt、stride、skip 和 skip_cnt。该 ststrmvex 指令向存储器存储具有总共“cnt”个分量的源寄存器 rS 的多个实例，其中，rA 指向第一个将要存储的分量的地址。该“cnt”参数因此定义将要存储的分量的总数，并且“rcnt”参数定义每个源寄存器 rS 的实例的源分量数。该 ststrmvex 指令从 rS（即，rS 的第一个实例）向存储器存储分量，使得它们距离“stride”个（ms 大小的）单位，直到存储了“skip_cnt”个分量。此时，将“skip*ms”添加至最后一个存储的分量的地址（而不是“stride*ms”），以获得下一分量的地址。另外，此时，重置记录 skip_cnt 的机制（例如，计数器）。因此，每次存储“skip_cnt”个分量时（自从该指令的起始或从上次重置该 skip_cnt 计数器），由“skip”参数，而不是“stride”参数指示下一分量的位置。

另外，在从 rS 的给定实例存储“rcnt”个分量后，如果还有分量要存储（即，还没有存储“cnt”个分量），则将“stride*ms”添加至最后一个存储的分量的地址（除非已经存储了“skip_cnt”个分量，在这种情况下，将“skip*ms”，而不是“stride*ms”添加至最后一个存储的分量的地址），以获得从 rS 的下一个实例存储的第一个分量的地址（例如，在执行 rS 用作目标寄存器的指令后存储）。例如，在一个实施例中，在 rS 用作指令的目标寄存器时实例入队。例如，如果 rS 用作后面指令的目标寄存器，并且还没有存储“cnt”个分量，则根据 stride、skip 和 skip_cnt 参数将来自 rS（该后面指令的目标寄存器）的“rcnt”个分量（包括，例如，后面指令结果的全部或部分）存储至存储器。在此实施例中，如果 rS 没有用作指令的目标寄存器，则其在执行该指令时不入队（并因此不存储为该矢量流的一部分）。在替换实施例中，可使用其它机制，例如指令中的标志，或者操作码（例如，对应于要求入队的指令的特殊子集等）指示 rS 的实例是否入队。

注意，在一个实施例中，如果“rcnt*ss”小于源寄存器的大小，

则忽略该源寄存器的剩余低阶部分。还要注意，“stride”和“skip”可为有符号的量，从而允许负的 stride 和 skip。

如上对于加载指令的说明，处理器 14 中的计数器（例如，在控制单元 28 中）可用于适当地记录 cnt、rcnt、skip 和 skip_cnt。替换实施例可使用其它方法记录这些参数。

注意，在一个实施例中，如果 ss 小于 ms，则每一源分量被适当地扩展，如 s/u 字段所指示（如对于 stvex 的说明），以生成存储器目标分量。另外，在一个实施例中，如果 ss 大于 ms，则每一源分量被适当地截短，如 h/l 字段所指示（如对于 stvex 和 stmvex 的说明），以生成存储器目标分量。可替换地，可使用舍入生成存储器目标分量。

将参考图 12-16 说明上述各种不同加载指令的示例。图 12 表示存储器的一部分 100，其可对应于，例如，图 1 的存储器 12 的全部或一部分。为了说明简单起见，将存储器部分 100 示出为 128 位宽的存储器，但在给定实施例中的实际配置可以不同。例如，存储器部分 100 包括矩阵 102，其第一行存储在位置 0x12-0x16（对应于半字 A、B 和 C），第二行存储在位置 0x22-0x26（对应于半字 D、E 和 F），并且第三行存储在位置 0x32-0x36（对应于半字 G、H 和 I）。存储器部分 100 包括矩阵 106，其第一行存储在位置 0x54-0x58（对应于半字 K、L 和 M），第二行存储在位置 0x64-0x68（对应于半字 N、O 和 P），第三行存储在位置 0x74-0x78（对应于半字 Q、R 和 S）。存储器部分 100 包括矩阵 108，其第一行存储在位置 0x8A-0x8E（对应于半字 aa、bb 和 cc），第二行存储在位置 0x9A-0x9E（对应于半字 dd、ee 和 ff），第三行存储在位置 0xAA-0xAE（对应于半字 gg、hh 和 ii）。存储器部分 100 包括存储在位置 0x46-0x4E 的矢量 104，其对应于半字 pp、qq、rr、ss 和 tt。存储器部分 100 还包括分别存储在位置 0xA2、0xA3、0xA4 和 0xA5 的字节大小的分量 T、U、V 和 W。

注意，在替换实施例中，存储器部分 100 可以不同方式寻址。另外，存储器部分 100 可为任何大小，并可保存任何类型的信息，还可位于数据处理系统的任何位置。

当矩阵存储为图 12 中的矩阵 102 和 106 时，上述加载和存储指令可用于执行，例如，矩阵运算。也就是，注意到每一矩阵 102 和 106 存储为 3×3 二维矩阵。另外，在一个实施例中，矩阵运算的结果还可以相同的格式存储，如 3×3 二维矩阵，例如，矩阵 108。因此，注意到，下面的伪码可用于执行矩阵 102 和矩阵 106 第一行的和，以获得最终的第一行矩阵 108。

/*R7 存储 0x12，并且 R8 存储 cnt 值 3 和 stride 值 1*/

lvex.u.mh.dh R0，R7，R8

/*R9 存储 0x54，并且 R10 存储 cnt 值 3 和 stride 值 1*/

lvex.u.mh.dh R1，R9，R10

add.h R2，R0，R1 /*对 R0 和 R1 中四个独立的半字分量求和，并将结果存储至 R2*/

/*R11 存储 0x8A，并且 R12 存储 cnt 值 3 和 stride 值 1*/

stvex.u.mh.sh R2，R11，R12

注意，在此实施例中，R7 存储矩阵 102 的第一行的起始地址（0x12），并且 R8 用作 rB，以存储上面参考图 2 和图 3 说明的 cnt 和 stride 参数。类似地，R9 存储矩阵 106 的第一行的起始地址（0x54），并且 R10 用作 rB，以存储 cnt 和 stride 参数。R11 用于存储 R2 的第一分量将要存储的位置（0x8A），并且 R12 用于存储 cnt 和 stride 参数。

图 13 表示执行上述伪码后的寄存器文件（例如，图 1 的标量寄存器文件 34 等）。注意，利用执行第一条 lvex 指令，将矩阵 102 的 A、B 和 C 加载至 R0，R0 的最后一个字段由预定值填充（例如，当前实施例中的零）。类似地，利用执行第二条 lvex 指令，将矩阵 106 的 K、L

和 M 加载至 R1。因此，注意，仅利用单个指令即可容易地加载每一矩阵的第一行，其允许指定 cnt 和 stride。该 add.h 指令对 R0 和 R1 中的四个独立的半字分量进行求和，并且将四个最终的半字分量放入 R2。注意，lvex 指令允许用零填充 R0 和 R1 的最后一个未使用的字段（因为 cnt*ds 小于 64 位，R0 和 R1 的宽度），这两个寄存器的和在 R2 的不想要的字段中导致零。按照这种方式，不是在 R2 的不想要的字段中具有未知值（或由于未知值的和造成出现不希望的异常条件），而是为已知的预定值（在此示例中为零）。然后使用 stvex 指令将 R2 的头 3 个字段存储至存储器，使得 aa=A+K，bb=B+L 和 cc=C+H。

然后通过下面的伪码完成矩阵 102 和 106 的和。

```
add     R7，0x10        /*R7 现在存储 0x22*/
add     R9，0x10        /*R9 现在存储 0x64*/
add     R11，0x10       /*R11 现在存储 0x9A*/
lvex.u.mh.dh    R0，R7，R8
lvex.u.mh.dh    R1，R9，R10
add.h               R2，R0，R1
stvex.u.mh.sh   R2，R11，R12
add     R7，0x10        /*R7 现在存储 0x32*/
add     R9，0x10        /*R9 现在存储 0x74*/
add     R11，0x10       /*R11 现在存储 0xAA*/
lvex.u.mh.dh    R3，R7，R8
lvex.u.mh.dh    R4，R9，R10
add.h               R5，R0，R1
stvex.u.mh.sh   R5，R11，R12
```

头三个求和指令增加了 R7，R9 和 R11 的值，使得它们均分别指向每一矩阵 102、106 和 108 的第二行的第一个分量。图 14 表示执行上面代码后图 13 的寄存器文件。头两个 lvex 指令后，D、E 和 F 载入

至 R0，并且 N、O 和 P 载入至 R1。该 add.h 指令对 R0 和 R1 中的四个独立半字分量求和，并将最终的四个半字分量放入 R2。然后由 stvex 指令将该结果存储至矩阵 108 中的存储器，使得 dd=D+N、ee=E+O 以及 ff=F+P。第五至第七条求和指令再一次地增加了 R7、R9 和 R11 的值，使得它们均分别指向每一矩阵 102、106 和 108 的第三行的第一个分量。注意，用于加载矩阵 102 和 106 的第三行的 lvex 指令将它们分别加载至 R3 和 R4，而不是 R0 和 R1。应当理解，替换实施例可按照需要使用这些寄存器中的任一个，以执行这些操作。例如，可连续覆盖同样的寄存器（例如，R0 和 R1）或可使用不同的寄存器（例如 R3 和 R4）。最后一条存储指令将结果存储至矩阵 108，使得 gg=G+Q、hh=H+R 以及 ii=I+S。

注意，在替换实施例中，可使用不同的代码和指令对矩阵 102 和 106 进行求和，并将结果存储至 108。例如，上面的伪码可写在循环中，使得在循环体中对于这些矩阵的每一行具有两个 lvex 和一个 stvex 指令，其中，该循环执行 3 次，每行一次。可替换地，可使用不同的矢量加载和存储指令将所有的矩阵 102 和矩阵 106 一次送入寄存器文件。这可使用，例如 lmvex 或 lmvex2 指令完成。例如，图 15 和 16 表示将矩阵 102 加载至寄存器 R0、R1 和 R2 的两个示例。

图 15 表示例如使用 lvmvex 指令加载矩阵 102 后的寄存器文件。在下面的 lvmex 指令中，假设 R7 指向矩阵 102 的起始（R7 存储 0x12），并且 R8 对应于 rB，其存储 cnt 值 9，stride 值 1，skip 值 6 和 skip_cnt 值 3。因此，可使用下列指令：

lmvex.u.mh.dh R0，R7，R8 /*cnt=9，stride=1，skip=6 且 skip_cnt=3*/

因此，如图 15 中所示，分量 A、B 和 C 载入至 R0，每一个在存储器中连续，因为这些分量距离“stride*ms”或 1 个半字放置。然后，由于加载了 skip_cnt 个（即，3 个）分量，将位于“skip*ms”（即，6

个半字）处，而不是“stride*ms”（即，1 个半字）处的下一个分量 D 作为 R0 的第四个分量加载。然后，分量 E 和 F 加载至 R1，其为 R0 之后的寄存器，直到再一次地加载 skip_cnt 分量，从而需要“跳跃（skip）”至下一个分量 G，其位于 6 个半字处。继续如此，直到加载了“cnt”个分量。因此，最后一个分量 I 加载至 R2 的第一个字段。注意，R2 的其余字段由预定值填充（在此示例中为零）。

图 16 表示使用例如，lmvex2 指令加载矩阵 102 后的寄存器文件，在该指令中指定 cnt、rcnt、skip 和 skip_cnt。在下面的 lmvex 指令中，假设 R7 指向矩阵 102 的起始（R7 存储 0x12），并且 R8 对应于 rB，其存储 cnt 值 9，rcnt 值 3，stride 值 1 和 skip 值 6。因此，可使用下面的指令：

```
lmvex2.u.mh.dh      R0,R7,R8   /*cnt=9,rcnt=3,stride=1,skip=6*/
```

因此，如图 16 中所示，分量 A、B 和 C 载入至 R0，每一个在存储器中连续，因为跨度（stride）为“stride*ms”或 1 个半字。此时，在 R0 中载入 rcnt 个分量（即，3 个），并且因此，从“skip*ms”（即，6 个半字）位置处加载下一分量 D。此外，不将下一分量 D 作为 R0 的第四个分量加载，如同在图 15 的前一示例中利用 lvmvex 指令的做法，而是作为 R1（其为 R0 之后的寄存器）的第一个分量加载，因为 R0 已经具有“rcnt”个分量。分量 E 和 F 也加载至 R1，对于 R1 完成值为 3 的 rcnt。由于已经在 R1 中加载了 rcnt 个分量（并且，还没有加载 cnt 个分量，即 9 个），因而从由 skip 确定的位置，即 6 个半字处的位置，将下一个分量 G 取回至 R2。另外，分量 H 和 I 也载入至 R2。此时，已经加载了 cnt 个分量。还要注意，R0，R1 和 R2 的每一未使用的字段由预定值填充（在此例中为零）。

在替换实施例中，如上所述，lvmvex2 指令还可允许指定 skip_cnt 字段。例如，还可使用 lmvex2 指令，如上利用值为 3 的 skip_cnt，如

图 16 所示加载矩阵 102。在此示例中（仍然指图 16 和上面的具有所有同样参数和附加的 skip_cnt=3 的 lvmvex2 指令），分量 A、B 和 C 载入至 R0，每一个在存储器中连续，因为这些分量距离“stride*ms”或 1 个半字放置。然后，由于已经加载了 skip_cnt 个（即，3 个）分量，则加载位于“skip*ms”（即，6 个半字）处，而不是位于“stride*ms”（即，1 个半字）处的下一分量 D。另外注意，仅在每一寄存器中加载 rcnt 个（即，3 个）分量。因此，不将下一分量 D 作为 R0 的第四分量加载，如同图 15 的前一示例使用 lmvex 指令那样，而是作为 R1（其为 R0 后面的寄存器）的第一个分量加载。然后分量 E 和 F 也加载至 R1。由于再一次地加载了 skip_cnt 个分量（并且还没有加载 cnt 个分量，即 9 个），则下一分量 G 位于 6 个半字处。另外，由于已经将 rcnt 个分量加载至 R1，则将下一分量 G 连同 H 和 I 一起加载至 R2。此时，加载了 cnt 个分量。还要注意，R0，R1 和 R2 的每一未使用的字段由预定值填充（在此例中为零）。

因此，采用上述加载指令的各种实施例，诸如矩阵 102 的矩阵可容易地以多种方式从存储器移动至寄存器文件。还要注意，使用 rcnt 参数允许减少在使用传统编程方法从存储器加载矢量分量时所需的掩码运算。

在替换实施例中，上述指令可比此处说明的指令包括更多，更少或不同的参数。例如，在不同的实施例中，该 lmvex2 指令可以考虑，也可以不考虑 skip_cnt 参数。还要注意，上面提供的示例指令将 s/u 字段设置为 u，使得如果有必要的话，在存储至寄存器时，对这些分量进行零扩展。在一个实施例中，可将 s/u 字段设置为“s”，或者省略该字段。例如，在一个实施例中，如果 ms=ds，则省略或忽略该 s/u 字段。

图 16 还表示加载指令的示例，其从存储器部分 100 的位置 0xA2、0xA3 和 0xA4 向 R4 加载 T、U 和 V。例如，可使用下面的 lvex 指令，其中假设 R10 存储起始值 0xA2 并且 R11 存储 cnt 值 3 和 stride 值 1。

lvex.u.mb.dh R4，R10，R11

注意，存储器大小（mb）与目标大小（dh）是不同的。也就是，在此处说明的所有指令中，可独立地设置每一存储器大小和目标大小（在存储指令的情况下为源大小），使得它们相同或者不同。在上面的示例中，注意，存储器大小设置为 mb，表明存储器中每一分量（T、U 和 V）的大小为字节。然而，目标大小设置为 dh，以表示 R4 中目标大小为半字，其大于字节。因此，在 R4 中加载 cnt 个分量（3 个分量），其中，适当地扩展每一分量 T、U 和 V，以填充半字目标大小。在此示例中，它们按照“.u”所示进行零扩展（即，无符号扩展）。然而，可使用“.s”对每一分量进行有符号扩展。另外，注意，由于“cnt*ds”为 3 个半字（即 46 位），并且从而小于 R4 的大小，则 R4 的其余未使用部分由预定值填充（在此示例中为零）。

图 17 表示寄存器文件的示例，其中，利用值为 1 的 stride 和值为 2 的 stride 加载存储器部分 100 的矢量 104。在一个示例中，假设通过存储 0x46，R7 指向 pp，矢量 104 的起始，并且 R10 存储 cnt 值 5，stride 值 1 和 skip_cnt 值 5。注意，skip 的值在此示例中无关紧要，因为 cnt=skip_cnt（即从存储器正在加载单个矢量，无需不断地“跳跃”）。因此可使用下面的指令将矢量 104 加载至寄存器文件。

lmvex.u.mh.dh R1，R7，R10 /*cnt=5，stride=1，skip_cnt=5*/

在上面的示例中，将 cnt 个（5 个）分量加载至寄存器文件，从 R1 开始，然后是 R2。由于 stride＝1，并且后面的分量距离“stride”个 ms 大小的单位放置，则将要加载的分量彼此距离 1 个半字。因此，5 个连续的半字（pp、qq、rr、ss 和 tt）载入至寄存器 R1 和 R2。还要注意，R2 的其余未使用部分由预定值填充（在此例中为零）。另外注意，对于 s/u 字段可使用“.s”代替。或者可替换地，不提供该 s/u 字

段。

在另一实施例中，假设 R7 指向 pp（存储 0x46），并且 R10 存储 cnt 值 3 和 stride 值 2。因此，可使用下面的指令每隔一个分量向该寄存器加载矢量 104。

lvex.u.mh.dh R4，R7，R10 /*cnt=3，stride=2*/

在此示例中，可使用 lvex 指令，因为 cnt*ms（3 隔半字）适合单个寄存器（所示实施例中的 64 位寄存器）。利用此 lvex 指令，将 cnt 个分量（3 个分量）加载至 R4，其中，后面的分量距离“stride”个 ms 大小的单位放置。因此，在此示例中，后面的分量以距离 2 个半字（“stride*ms”）放置，因此，在加载每一分量后，下一分量从 2 个半字处取回。例如，从 0x46 取回 pp，从 0x4A（0x46+0x2）取回 rr。如此继续，直到加载 3 个分量。还要注意，R4 的未使用部分由预定值填充（在此示例中为零）。

图 18 表示处于四个不同时刻（time＝0，time＝1，time＝2，time＝3）的存储器的一部分 200，其可对应于，例如，图 1 的存储器 12 的全部或部分（或者可位于数据处理系统的任何地方）。在说明的实施例中，存储器部分 200 示出为 64 位宽存储器。注意，在替换实施例中，可以各种不同方式将存储器部分 200 格式化为具有不同的宽度。存储器部分 200 分别在存储器位置 0x08，0x0A，0x0C，0x0E 和 0x10 存储分量 C_4，C_3，C_2，C_1 和 C_0。分量 t、u 和 v 分别存储在 0x12，0x14 和 0x16。存储器部分 200 还在 time=0 时分别在位置 0x20，0x22，0x24，0x26 和 0x28 存储 X_{-4}，X_{-3}，X_{-2}，X_{-1} 和 X_0。在一个实施例中，这些值的每一个可以指以特定采样速率采样的数字采样信号的各种时间采样。例如，在 time=0，当前采样由 X_0 表示，并且一个时间延迟（即，1/采样速率）前的采样由 X_{-1} 表示，两个时间延迟（即，2*1/采样速率）前的采样由 X_{-2} 表示，等等。

接下来，在 time=1，当前采样由 X_1 表示，并且一个时间延迟（即，1/采样速率）前的采样由 X_0 表示，两个时间延迟（即，2*1/采样速率）前的采样由 X_{-1} 表示，等。注意，最新的采样覆盖最老的时间采样，在示出 time=1 的示例中其为 X_{-4}。因此，在 time=2，当前采样 X_2 覆盖 X_{-3}，并且，在 time=3，当前采样 X_3 覆盖 X_{-2}。因此，存储位置 0x20，0x22，0x24，0x26 和 0x28 可实现循环缓冲，其连续地存储进来的数字采样信号的时间采样。在替换实施例中，可对于不同的原因或应用实现循环缓冲。

在一个实施例中，可使用修改的 lmvex 加载指令按照某种方式将该循环缓冲的值正确地送入寄存器文件的寄存器（如将参考图 19-22 的说明），该方式为这些值处于用于计算的正确的希望的顺序，例如，在执行滤波函数时。例如，下面的等式对应于滤波函数，其中，Y_0，Y_1，Y_2 和 Y_3 分别对应于在 time=0，time=1，time=2，time=3 时的结果：

Eq.1 $Y_0=C_0*X_0+C_1*X_{-1}+C_2*X_{-2}+C_3*X_{-3}+C_4*X_{-4}$

Eq.2 $Y_1=C_0*X_1+C_1*X_0+C_2*X_{-1}+C_3*X_{-2}+C_4*X_{-3}$

Eq.3 $Y_2=C_0*X_2+C_1*X_1+C_2*X_0+C_3*X_{-1}+C_4*X_{-2}$

Eq.4 $Y_3=C_0*X_3+C_1*X_2+C_2*X_1+C_3*X_0+C_4*X_{-1}$

在一个实施例中，该修改的 lmvex 指令称为一次跳跃加载多个矢量分量（lmvex_skip_once）指令，如图 27 中所示。图 27 的 lmvex_skip_once 指令包括 s/u 字段、ms 字段和 ds 字段，其就是按照上面对于 lmvex 指令的说明操作。该 lmvex_skip_once 指令还采用三个操作数：rD、rA 和 rB。该 lmvex_skip_once 指令使用附加参数 cnt、stride、skip 和 skip_cnt，其中，它们存储在由 rB 表示的一个或多个寄存器中，或由该指令的子操作码指定。除了 skip_cnt 附加参数外，所有操作数和附加参数以与对于 lvmex 指令的说明相同的方式操作。如上所述，该 lmvex 指令取回 skip_cnt 个分量，并且然后使用“skip”参数取回下一个分量。此时，必须重置该 skip_cnt 计数器，使得在再一次跳跃前取回

下一 skip_cnt 个分量。如此继续，直到加载了 cnt 个分量。与 lmvex 指令不同，lmvex_skip_once 指令在取回 skip_cnt 个分量后并不重置该 skip_cnt 计数器。因此，在取回 skip_cnt 个分量后，下一个分量位于 skip 个 ms 大小的单位处。然而，从该指针开始，skip_cnt 计数器持续递减（即，不重置），直到取回“cnt”个分量。

例如，如将参考图 19-22 的说明，下面的伪码段可用于计算上面等式 1、2、3 和 4 的滤波结果 Y_0，Y_1，Y_2 和 Y_3。

/*初始化 R12，以存储 cnt 值 5 和 stride 值 1*/

/*初始化 R13，以存储 cnt 值 5，stride 值 1，skip_cnt 值 5 和 skip 值-4*/

/*初始化 R1，以存储 0x08*/

/*初始化 R2，以存储 0x20*/

lmvex.u.mh.dh R6，R1，R12 /*将 C_4-C_1 加载至 R6，将 C_0 加载至 R7*/

loop：

lmvex_skip_once.u.mh.dh R8，R2，R13 /*在第一次迭代中，将 X_{-4}-X_{-1} 加载至 R8，并将 X_0 加载至 R9*/

vmac.h R10，R6，R8 /*在第一次迭代中，将 $C_4*X_{-4}+C_3*X_{-3}+C_2*X_{-2}+C_1*X_{-1}$ 加载至 R10*/

vmac.h R11，R7，R9 /*在第一次迭代中，将 C_0*X_0 存储至 R11*/

add R10，R10，R11 /*在第一次迭代中，将 Y_0 存储至 R10*/

/*在 R10 中的值可存储至存储器*/

add R2，R2，2 /*将 R2 中的指针向前移动 2*/

/*修改 R13 中的值，使得“skip_cnt=skip_cnt-1”*/

branch loop

循环由上面代码段中的标签“loop：”表示，在进入该循环之前，，

用 cnt 值 5 和 stride 值 1 初始化 R12，用 cnt 值 5，stride 值 1，skip_cnt 值 5 和 skip 值-4 初始化 R13。因此，寄存器 R12 和 R13 可在后面的矢量加载指令中用作“rB”。初始化 R1 指向常矢量的起始（位于 0x08 的 C_4）。初始化 R2 指向循环缓冲的起始，其在 time=0 时为 0x20，如图 18 所示。因此注意，循环的第一次迭代对应于加载至对应于 time=0 时寄存器文件（如图 19 所示）的值。

第一条 lmvex 指令根据 R12 中存储的参数，将 C_4-C_1 加载至 R6，并将 C_0 加载至 R7，R12 在前面初始化；并且控制字段如上参考图 4 和 5 的说明。然后，循环内的第一次迭代开始于 lmvex_skip_once 指令。在该第一次迭代中，对应于 time＝0，该 lmvex_skip_once 指令根据 R12 中存储的参数和这些控制字段，将 X_{-4}-X_{-1} 加载至 R8，并且将 X_0 加载至 R9，如图 19 中的 R8 和 R9 所示，其中，R12 在前面初始化。注意，由于在该第一次迭代中，skip_cnt=cnt，skip 可视为“无关紧要”，因为并不需要该 skip 值。也就是，当该 skip_cnt 计数器终止时，cnt 计数器也终止，并从而不需要“跳跃”即可完成该指令。

第一条矢量乘积累加指令（vmac.h）对寄存器 R6 和 R8 的半字分量执行矢量乘积累加运算，并将结果存储至 R10。因此，R10 现在存储 $C_4*X_{-4}+C_3*X_{-3}+C_2*X_{-2}+C_1*X_{-1}$，如图 19 中的 R10 所示。第二条矢量乘积累加指令对寄存器 R7 和 R9 的半字分量执行矢量乘积累加运算，并将结果存储至 R11。因此，R11 现在存储 $C_0*X_0+0*0+0*0+0*0$，其就等于 C_0*X_0。注意，在当前的示例中，通过每一个加载矢量指令（lmvex 和 lmvex_skip_once），寄存器（例如 R7 和 R9）的其余未使用的部分清零，从在 R11 中产生正确结果。否则，如果例如，反而从存储器 200 加载值 t，u 和 v ，则 R11 可能为不正确的值。

然后，第一条求和指令对 R10 和 R11 的值求和，并将结果存储至 R10（此运算并未在图 19 中示出）。然后可在需要时存储对应于上面等式 1 的 Y_0 的该值，例如回到存储器 200。第二条求和指令将循环缓

冲的起始指针向前移动至下一个半字，使得对于 time=1，该起始指针现在指向地址 0x22 的 X_{-3}，如图 18 和 20 所示。接下来，修改 R13，使得 skip_cnt=skip_cnt-1。因此。经过上面代码段的第一次迭代，skip_cnt 现在等于 4。可使用循环计数器或附加指令（未包括在上面的代码段中）指示是否退出该循环。如果不退出该循环，则代码返回“loop”，并开始下一次迭代。

值 C_4-C_0 保持在 R6 和 R7 中（如图 20 的 R6 和 R7 所示）。该循环的第二次迭代开始于 lmvex_skip_once 指令。在该第二次迭代中，对应于 time=1，该 lmvex_skip_once 指令根据存储在 R12 中的参数和控制字段，将 X_3-X_0 加载至 R8，并将 X_1 加载至 R9，如图 20 中的 R8 和 R9 所示。在此次迭代中，由于 skip_cnt 为 4，并且 rA 指向 X_{-3} 为第一个要加载的分量，加载 4 个分量：X_{-3}-X_0。此时，由于加载了 skip_cnt（4）个分量，并且还没有加载 cnt（5）个分量，则从“skip”个 ms 大小（半字）的单位的位置处加载下一分量。因此，在此示例中，值为-4 的 skip 允许从 0x28 - 4*2 = 0x20 加载分量 X_1。（注意，存储器 200 可半字寻址；因此，skip 个 ms 大小的单位处表示-4 个半字处。）注意，如上所述，在 time=1 时，存储器 200 中的最近分量存储在 0x20，其为前一最老的时间采样的位置。因此，lmvex_skip_once 指令以从最老到最新的时间采样顺序，从存储器位置 0x20-0x28 向 R8 和 R9 正确地加载了用于计算 Y_1 所需的 5 个分量。

第一条矢量乘积累加指令（vmac.h）对寄存器 R6 和 R8 的半字分量执行矢量乘积累加运算，并将结果存储至 R10。因此，R10 现在存储 $C_4*X_{-3}+C_3*X_{-2}+C_2*X_{-1}+C_1*X_0$，如图 20 中的 R10 所示。第二条矢量乘积累加指令对寄存器 R7 和 R9 的半字分量执行矢量乘积累加运算，并将结果存储至 R11。因此，R11 现在存储 $C_0*X_1+0*0+0*0+0*0$，其就等于 C_0*X_1。注意，在当前的示例中，通过每一加载矢量指令（lmvex 和 lmvex_skip_once），寄存器（例如 R7 和 R9）的其余未使用部分清零，从在 R11 中产生正确结果。否则，如果例如，反而从存储器 200 加载

值 a、b 或 c（或者 t、u 和 v），则 R11 可能为不正确的值，。

然后，第一条求和指令对 R10 和 R11 的值求和，并将结果存储至 R10（此运算并未在图 20 中示出）。然后可在需要时存储对应于上面等式 2 的 Y_1 的该值，例如回到存储器 200。第二条求和指令将循环缓冲的起始指针向前移动至下一个半字，使得对于 time=2，该起始指针现在指向地址 0x24 的 X_{-2}，如图 18 和 21 所示。接下来，修改 R13，使得 skip_cnt=skip_cnt-1。因此。经过上面代码段的第二次迭代，skip_cnt 现在等于 3。可使用循环计数器或附加指令（未包括在上面的代码段中）指示是否退出该循环。如果不退出该循环，则代码返回至“loop”，并开始下一次迭代。

第三次迭代类似于第一次和第二次迭代操作。对于第三次迭代，其对应于 time=2，lmvex_skip_once 指令将 X_{-2}，X_{-1}，X_0，X_1，X_2 加载至寄存器 R8 和 R9，如图 21 所示。因此，一旦加载了分量 X_{-2}，X_{-1} 和 X_0（开始于 0x24），就已经加载了 skip_cnt 个分量，并且使用 skip 加载 X_1 和 X_2。后面的矢量乘积累加指令导致 $C_4*X_{-2}+C_3*X_{-1}+C_2*X_0+C_1*X_1$ 存储在 R10 中，并且导致 C_0*X_2 存储在 R11 中。第一条求和指令将对应于 Y_2 的结果存储在 R10 中，其中，然后可将 Y_2 存储至存储器，如果需要的话。第二条求和指令然后将该指针向前移动至起始位置 0x26，如图 22 中所示。另外，修改 R13，使得 skip_cnt=skip_cnt-1。因此。经过上面代码段的第三次迭代，skip_cnt 现在等于 2。可使用循环计数器或附加指令（未包括在上面的代码段中）指示是否退出该循环。如果不退出该循环，则代码返回至“loop”，并开始下一次迭代。

通过上面参考图 4 和 5 所述的 lmvex 指令，重置该记录 skip_cnt 的机制（例如，计数器），使得当加载下一 skip_cnt 个分量时（并且假设还没有加载 cnt 个分量），从 skip*ms 个单位处加载下一分量。然而，在当前示例中，使用 lvmex_skip_once 指令，使得当加载 skip_cnt 个分量时，不重置记录 skip_cnt 的机制。按照这种方式，在加载全部 cnt 个

分量前，将不会再一次地使用跳跃。这允许在后面的迭代中，正确地加载所有分量，如将所述，例如通过第四次迭代（对应于 time=3）。

第四次迭代类似于前面的迭代操作。对于第四次迭代，对应于 time=3，lmvex_skip_once 指令将 X_{-1}，X_0，X_1，X_2 和 X_3 加载至寄存器 R8 和 R9，如图 22 所示。在此次迭代中，注意，加载的第一个分量是来自位置 0x26 的 X_{-1}（如图 18 对于 time=3 的表示）。加载 X_{-1} 和 X_0 之后，就已经加载了 skip_cnt（其等于 2）个分量；因此，使用 skip 将指针绕回位置 0x20，因为还没有加载 cnt（5）个分量。还要注意，并没有重置 skip_cnt 机制。按照这种方式，即使加载了下一 skip_cnt（2）个分量，也不进行跳跃。如果在下一 skip_cnt 个分量后对于此次迭代进行跳跃，则在跳跃前仅取回分量 X_1 和 X_2，从而遗漏了 X_3。因此，尽管在前面的迭代中，在第一次跳跃后是否重置 skip_cnt 并不重要，但在某些迭代中，例如此次迭代，其可确定是否取回正确的分量。因此，可以知道如何将 lmvex_skip_once 指令用于正确地执行循环缓冲，例如，对于滤波器应用。还要注意，能够以多种不同的方法书写上面的代码段。例如，该循环可以展开，而不是写为循环形式。在此示例中，可以对于 time=0，1，2 使用 lmvex 指令，而仅对于 time=3 和 4 使用 lmvex_skip_once 指令。

在图 22 中可以看出，一旦加载了 X_{-1}，X_0，X_1，X_2 和 X_3，后面的矢量乘积累加指令导致 $C_4*X_{-1}+C_3*X_0+C_2*X_1+C_1*X_2$ 存储在 R10 中，并且导致 C_0*X_3 存储在 R11 中。第一条求和指令将对应于 Y_3 的结果存储在 R10 中，其中，然后可将 Y_3 存储至存储器，如果需要的话。此时，流程按照上面对于前面迭代的说明继续进行。在一个实施例中，在将 R12 修改为 skip_cnt 为 0 的指针后，可退出该循环，使得重置 skip_cnt，例如，返回 5，并重复整个过程。

因此，能够理解如何将矢量加载指令，连同 skip、skip_cnt 和 cnt 用于执行循环缓冲。在一个实施例中，可使用循环缓冲加载多个矢量

分量（lmvex_cb）指令，如图 28 所示。该指令类似于上面参考图 4 和 5 说明的 lmvex 指令，除了在 rB 中存储的附加参数为 buffer_size（缓冲大小）和 offset（偏移）。（在使用 lmvex 指令时，注意，在替换实施例中，可作为操作码或子操作码部分包括该附加参数）。图 29 表示如何使用这些附加参数定义循环缓冲。例如，在使用上述加载指令时，rA 表示存储器中存储该循环缓冲的存储器位置的起始，如图 29 所示。该 buffer_size 表示“缓冲分量总数*ms”，并且 offset 定义在该循环缓冲中哪里是将要加载的第一个分量。另外注意，尽管并未示出，还可使用循环缓冲存储多个矢量分量（stmvex_cb）指令，作为 lmvex_cb 的存储对应。对于 lmvex_cb 和 stmvex_cb 指令，在某些实施例中，由“offset”表示的当前起始值（current_start）在每次遍历该循环缓冲后，即在处理了 cnt 个分量后，自动增加或降低预定数量的分量，作为执行 lmvex_cb 或 stmvex_cb 指令的一部分。

图 30 表示循环缓冲加载分量流（lstrmvex_cb）指令，其类似于上面 lstrmvex 指令操作，除了附加参数包括 buffer_size 和 offset（如上面参考图 29 的说明），而不是 cnt、skip 和 skip_cnt。正如上面的 lstrmvex 指令，该 lstrmvex_cb 指令从存储器发起矢量分量流，其中，例如根据 rD 何时用作后面指令的特定源操作数，这些分量可以出队。因此，除了关于寻址模式外，上面对于 lstrmvex 的相同讨论此处也适用于 lstrmvex_cb。还要注意，尽管没有说明，但也可以使用循环缓冲存储矢量分量流（ststrmvex_cb）指令，作为 lstrmvex_cb 的存储对应。对于 lstrmvex_cb 和 ststrmvex_cb 指令，在一些实施例中，每次遍历该循环缓冲后，由“offset”表示的当前起始值（current_start）自动递增预定数量的分量。

lmvex_cb、stmvex_cb、lstrmvex_cb 和 ststrmvex_cb 指令的可选指示符包括 rcnt 和 stride（未在图 29 和 30 中示出），它们如同前面对于 lmvex、lmvex2、stmvex、stmvex2、lstrmvex 和 ststrmvex 指令的说明操作。这允许控制加载至寄存器 rD 的每一实例，或从源寄存器 rS 的

每一示例存储的循环缓冲分量的数量，并且控制遍历访问该循环缓冲的跨度。例如，可使用跨度执行分数速率滤波运算，或抽取滤波器。

图 23-26 表示如何使用 lstrmvex 指令（如上参考图 7 的说明）执行，例如，矩阵相乘的一个示例。对于此示例，将使用图 12 的存储器部分 100，和矩阵 102、106 和 108。图 23-26 表示在矩阵相乘期间的不同时间点的寄存器文件内容，例如图 1 的标量寄存器文件 34 的内容等。

在当前示例中，矩阵 102 将乘以矩阵 106，结果将对应于矩阵 108。例如，在执行矩阵相乘时，下列等式用于结果矩阵 108 的头两列。

Eq.5 aa = A*K + B*N + C*Q

Eq.6 dd = D*K + E*N + F*Q

Eq.7 gg = G*K + H*N + I*Q

Eq.8 bb = A*L + B*O + C*R

Eq.9 ee = D*L + E*O + F*R

Eq.10 hh = G*L + H*O + I*R

在上面的等式中，注意，在获得 aa、dd 和 gg 时，需要从矩阵 102 出队分量（因为 A、B、C 用于获得 aa；D、E、F 用于获得 dd；并且然后 G、H、I 用于获得 gg），同时不需要使矩阵 106 出队（因为需要 K、N、Q 获得每个 aa、dd 和 gg）。因此，下面的伪码提供如何使用指令，例如 lstrmvex 指令，获得结果矩阵 108 的一个示例。注意，在下面的伪码示例中，仅当 rD 作为指令（例如，指令目标，源 1，源 2）的第一个源（源 1）时才执行出队。

/*初始化 R6，以存储值为 9 的 cnt，值为 3 的 rcnt，值为 1 的 stride，值为 6 的 skip，和值为 3 的 skip_cnt*/

/*初始化 R4，以存储 0x12，即矩阵 102 的第一行的起始*/

/*初始化 R7，以存储值为 9 的 cnt，值为 3 的 rcnt，值为 8 的 stride，

值为-15 的 skip，和值为 3 的 skip_cnt*/

/*初始化 R5，以存储 0x54，即矩阵 106 的第一列的起始*/

lstrmvex.u.mh.dh R1，R4，R6 /*加载矩阵 102 的第一行：A，B，C*/

lstrmvex.u.mh.dh R2，R5，R7 /*加载矩阵 106 的第一列：K，N，Q*/

vmac.h R10，R1，R2 /*在 R10 中存储 aa，并使矩阵 102 的 R1 出队*/

vmac.h R11，R1，R2 /*在 R11 中存储 dd，并使矩阵 102 的 R1 出队*/

vmac.h R12，R2，R1 /*在 R12 中存储 gg，并使矩阵 104 的 R2 出队*/

lstrmvex.u.mh.dh R1，R4，R6 /*加载矩阵 102 的第一行：A，B，C*/

vmac.h R13，R1，R2 /*在 R13 中存储 bb，并使矩阵 102 的 R1 出队*/

vmac.h R14，R1，R2 /*在 R14 中存储 ee，并使矩阵 102 的 R1 出队*/

vmac.h R15，R2，R1 /*在 R15 中存储 hh，并使矩阵 104 的 R2 出队*/

上面的伪码可用于获得 aa、dd、gg、bb、ee 和 hh，如上面的等式 5-8 所述。如在此示例中所使用，假设所有值为无符号的半字。首先，初始化 R6，以存储值为 9 的 cnt，值为 3 的 rcnt，值为 1 的 stride，值为 6 的 skip，和值为 3 的 skip_cnt，其中，R6 可用作 lstrmvex 指令的 rB 寄存器。如图 23 所示，将 R4 初始化为指向位于 0x12 处的分量 A，即图 12 的矩阵 102 的第一行的第一个分量。初始化 R7，以存储值为 9 的 cnt，值为 3 的 rcnt，值为 8 的 stride，值为-15 的 skip，和值为 3 的 skip_cnt，其中，R7 可用作 lstrmvex 指令的 rB 寄存器。如图 23 所示，将 R5 初始化为指向 0x54 处的分量 K，即矩阵 106 的第一列的第一个

分量。

第一条 lstrmvex 指令发起矢量分量流，其依次加载矩阵 102 的每一行。首先，该指令将矩阵 102 的第一行加载至 R1（如图 23 中所示）。例如，分量 A、B 和 C 彼此距离 1 半字（因此需要值为 1 的 stride）。值为 3 的 rcnt 每次仅向 R1 加载 3 个分量（其余部分清零），并且在加载 skip_cnt 个分量（此例中为 3 个）之后，则在 C（0x16）位置增加 8*ms（8 半字）的跳跃，以在位置 0x22（0x16+0x08）获得下一分量 D 的地址。当 R1 用作后面指令的源 1 时，如下所述，则其出队，使得矩阵 102 的下一行（如 rcnt、stride 和 skip_cnt 所定义）加载至 R1。

第二条 lstrmvex 指令发起矢量分量流，其依次向 R2 加载矩阵 106 的每一列。首先，该指令将矩阵 106 的第一列加载至 R2（如图 23 中所示）。例如，分量 K、N 和 Q 在存储器中彼此距离 8 个半字（因此需要值为 8 的 stride）。值为 3 的 rcnt 每次仅向 R2 加载 3 个分量（其余部分清零），并且在加载 skip_cnt 个分量（此例中为 3 个）之后，则将 Q（0x74）位置增加-15*ms（-15 或-0x0F 个半字）的跳跃，以在位置 0x56（0x74-0x0F*2）获得下一分量 D 的地址，其中，L 为矩阵 106 的第二列的第一个分量。当 R2 用作后面指令的源 1 时，如下所述，则其出队，使得矩阵 106 的下一列（如 rcnt、stride 和 skip_cnt 所定义）加载至 R2。

第一条乘积累加指令（vmac）指令将 A*K + B*N + C*Q（其等于 aa）存储至 R10，如图 24 中所示。由于 R1 用作源 1，因而其根据第一条 lstrmvex 指令（和 R6 中的参数）出队，使得矩阵 102 的下一行（D、E 和 F）存储至 R1，而其余部分清零。此时，注意，由于 R2 没有用作源 1，则其不出队；因此，K、N 和 Q 保持在 R2 中，并且矩阵 106 的下一列也不存储至 R2。

第二条乘积累加指令（vmac）指令将 D*K + E*N + F*Q（其等于

dd）存储至 R11，如图 25 中所示。由于 R1 用作源 1，其再一次地根据第一条 lstrmvex 指令（和 R6 中的参数）出队，使得矩阵 102 的下一行（G、H 和 I）存储至 R1，而其余部分清零。此时，已经加载了对应于第一条 lstrmvex 指令的 9 个分量，从而已经加载了“cnt”个分量，并且此指令流终止。由于 R2 没有用作源 1，则其不出队；因此，K、N 和 Q 保持在 R2 中，并且矩阵 106 的下一列也不存储至 R2。此时，仅加载了对应于 lstrmvix 指令的 3 个分量，从而对于此指令没有加载 cnt 个分量。

第三条累加指令（vmac）指令将 G*K + H*N + I*Q（其等于 gg）存储至 R11，如图 26 中所示。由于 R2 现在用作源 1，因此其根据第二条 lstrmvex 指令（和 R7 中的参数）出队，使得矩阵 106 的下一列（L、O 和 R）存储至 R2，而其余部分清零。此时，已经加载了矩阵 106 的 9 个分量中的 6 个。另外，由于通过第一条 lstrmvex 指令已经加载了（矩阵 102 的）“cnt”个分量，因而 R1 并不改变，并仍然保存 G、H 和 I。（此外，即使通过第一条 lstrmvex 指令还没有加载“cnt”个分量，R1 仍然不改变，因为由于 R1 用作源 2，其将不出队。）

第三条 lstrmvex 指令发起新的矢量分量流，其与第一条 lstrmvex 指令相同。也就是，该第三条 lstrmvex 指令依次将矩阵 102 的每一行加载至 R1，如上所述。后面三条 vmac 指令分别向 R13、R14 和 15 存储值 bb、ee 和 hh。注意，在使用头三条 vmac 指令时，第一条 lstrmvex 指令通过第三条 lstrmvex 指令之后的头两条 vmac 指令出队，第二条 lstrmvex 指令通过第三条 lstrmvex 指令之后的第三条 vmac 指令出队。此时，矩阵 106 的最后一列（M、P 和 S）将加载至 R2（未示出），并且对于第二条 lstrmvex 指令的加载将停止，因为此时将要加载“cnt”个（即 9 个）分量。可使用类似的指令获得 cc、ff 和 ii。可将值 aa、dd、gg、bb、ee、hh、cc、ff 和 ii 分别存储回至矩阵 108 的存储位置 0x8A，0x9A，0xAA，0x8C，0x9C，0xAC，0x8E，0x9E，0xAE。如此处所述，可使用矢量存储指令执行这些存储。

注意，在上面的伪码示例中，这些分量根据后面指令中 rD 的位置出队（即 rD 是否用作源 1 或是源 2）。然而要注意，在替换实施例中，可使用其它因子指示这些分量是否出队。例如，可使用不同的操作码指示出队，或者使用其它的指示符，例如标志位等。此外，在一个实施例中，rD 在任何时候用作为后面指令的源时出队，不是仅在其为源 1 时出队。在又一实施例中，仅当使用同一 rD 作为源操作数的指令为一个或多个由处理单元 14 执行的所有指令的适当子集中的预定指令时执行出队。也就是，在此实施例中，实际上仅有指令或指令功能的子集作为它们执行的一部分执行出队，而由处理器单元 14 执行的其余指令不执行出队，不论 rD 是否用作源操作数。该执行出队的子集可为所有执行的指令的预定子集，并可基于数据处理系统 10 的需要确定。在又一实施例中，出队可由用于控制该出队操作的指令中的控制字段值确定。

尽管在前面说明图 23-26 的段落中已经说明了出队，但类似的入队操作同样适用于当前发明的实施例，并以基本类似的方式操作。可基于入队源寄存器 rS 作为后面指令目标的位置，或者基于在由处理单元 14 执行的所有指令的适当子集中的一个或多个预定指令中成为目标操作数，来执行入队。也就是，在一个实施例中，实际上仅有指令或指令功能的子集作为它们执行的一部分执行入队，由处理器 14 执行的其余指令并不执行入队，无论 rS 是否用作目标操作数。该执行入队的子集可为执行的全部指令的预定子集，并可基于数据处理系统 10 的需要确定。在又一实施例中，入队可由用于控制该入队操作的指令中的控制字段值确定。

图 31 表示可用于快速傅立叶变换（FFT）的指令。图 31 表示加载多个矢量分量 FFT（lmvex_fft）指令，其具有控制字段 s/u、ms 和 ds 以及操作数 rD，rA 和 rB。这些控制字段和操作数的每一个如上对于其它加载指令的说明操作。该 lmvex_fft 指令还使用在 rB 中存储的附加

参数“radix”（基数），并且可选地如上对于其它加载矢量分量指令的说明执行在 rB 中存储的 rcnt 控制字段。在使用上面的指令时，在替换实施例中，“radix”（和“rcnt”，如果存在的话）可为操作码或子操作码的一部分，并且因此，rB 可能不是必要的。该 lmvex_fft 指令还可使用截短或舍入字段，如上所述。

图 32 表示存储多个矢量分量 FFT（stmvex_fft）指令，其也具有控制字库 s/u、ms 和 ss 以及操作数 rS、rA 和 rB。这些控制字段和操作数中的每一个如上对于其它存储指令的说明操作。该 stmvex_fft 指令还可使用截短或舍入字段，如上所述，并可选地如上对于其它存储矢量分量指令的说明执行 rcnt 控制字段。该 stmvex_fft 指令还使用在 rB 中存储的附加参数“radix”。在使用上面的指令时，在替换实施例中，“radix”可为该操作码或子操作码的一部分，并且因此，rB 可能不是必要的。该 stmvex_fft 指令可视为 lmvex_fft 指令的存储对应。

重新参看该 lmvex_fft 指令，“radix”参数可用于根据 FFT 寻址算法向寄存器或寄存器组加载一系列数据，如本领域所公知。例如，对于 FFT，公知将以 X_0，X_1，X_2，X_3，X_4，X_5，X_6 和 X_7 为顺序的数据进行“比特翻转”，形成 X_0，X_4，X_6，X_2，X_1，X_5，X_3，X_7，以进行特定 FFT 计算。在该 lmvex_fft 指令中，该“radix”定义分量的数量，以及这些分量的特定存储顺序。例如，图 34 表示存储器部分 300，其可为，例如图 1 的存储器 12 的一部分。存储器部分 300 分别包括存储在位置 0x16-0x24 的分量 X_0，X_1，X_2，X_3，X_4，X_5，X_6 和 X_7。存储器部分 300 还包括存储在位置 0x44-0x52 的分量 Y_0，Y_4，Y_6，Y_2，Y_1，Y_5，Y_3 和 Y_7。

图 35 表示诸如图 1 的标量寄存器文件 34 的寄存器文件的一部分。例如，基数 8 的 lvmex_fft 指令可用于以比特翻转形式将 X 个分量送入寄存器 R1 和 R2（如图 35 所示），其中，R1 由 rD 指定，并且将要加载的第一个存储器位置（0x16）由 rA 指定。类似地，基数 8 的 lmvex_fft

可用于将比特翻转的 Y 个分量以连续顺序送入寄存器 R4 和 R5（如图 35 所示），其中，R4 由 rD 指定，并且将要加载的第一个存储器位置（0x44）由 rA 指定。因此，该 lmvex_fft 指令可以比特翻转的形式加载分量，其中，这些分量可以一开始就进行比特翻转，或者不翻转。

类似地，stmvex_fft 指令可用于将这些分量与比特翻转形式存储至存储器。例如，基数 8 的该 stmvex_fft 指令可用于将比特翻转的 X 个分量从 R1 和 R2 存储至位置 0x16-0x24 的存储器，使得存储器中的这些分量与 R1 和 R2 中的那些分量相比没有进行比特翻转。类似地，该 stmvex_fft 指令可用于将连续的 Y 个分量从 R4 和 R5 存储至位置 0x44-0x52 的存储器，使得存储器中的这些分量与 R4 和 R5 中的那些分量相比进行了比特翻转。

图 33 表示加载矢量分量流 FFT（lmstrmvex_fft）指令，其具有控制字段 s/u、ms 和 ds 以及操作数 rD、rA 和 rB。这些控制字段和操作数的每一个如上对于 lmstrmvex 指令的说明操作。该 lmstrmvex_fft 指令还使用存储在 rB 中的附加参数“radix”，并且能够如上对于其它加载矢量分量指令的说明可选地执行存储在 rB 中的 rcnt 控制字段。在利用上面的指令时，在替换实施例中，“radix”（和“rcnt”，如果存在的话）可为操作码或子操作码的一部分，并且因此，rB 可不是必要的。“radix”定义将要加载的分量的总数。在利用 lmstrmvex_fft 指令时，该 lmstrmvex_fft 从存储器发起矢量分量流，其中，以比特翻转形式或以来自比特翻转的连续顺序加载这些分量，并且根据例如，后面指令中 rD 的位置使这些分量出队。注意，尽管没有表示出来，但可使用对应的存储指令发起以比特翻转形式向存储器存储分量流，例如对于 FFT 应用。

因此，可以明白，如何将各种不同类型的矢量加载和存储指令用于具有一组寄存器的系统，例如，标量寄存器文件等。也就是，上面的指令可用于现有设计或仅需对现有设计的执行单元（例如执行单元

32）和控制单元（例如控制单元 28）进行少量的改动。注意，此处说明的每一条指令可具有不同的格式，并且甚至可分为多条指令。另外，在从或向寄存器传送矢量分量时，可使用附加参数指示将哪一个寄存器分量传送至存储器（例如对于矢量存储指令），或者寄存器中的哪一个寄存器分量存储从存储器加载的矢量分量（例如对于矢量加载指令）。此外，可使用附加参数定义如何将矢量分量放置在寄存器中，或者从存储器的何处读取矢量分量。例如，可提供起始位置指示符指示向或从寄存器中的何处开始传送。另外，对于寄存器可给定跨度，使得能够以非相邻的方式在寄存器中向或从寄存器传送参数。另外，此处说明的所有参数和控制字段可出现在任何这些指令的任何组合之中。

还要注意，上面已经讨论了不同的寻找模式，例如使用 stride、skip、skip_cnt 等的模式，或者使用 buffer_size 和 offset 的模式，或者使用基数（radix）的模式。然而，在替换实施例中，可使用不同的寻址模式。例如，可代之以指定阵列维数，例如，阵列的维数等（例如，2-维 4×4 阵列，3-维 5×5 阵列等）。因此，与上面实施例一同说明的寻址模式可用于以各种模式加载或存储数据或数据阵列，包括例如，zig-zag 模式和其它规则模式，如上所述。与在当前 SIMD 处理器中所需的开销相比，此处说明的指令还可降低向或从存储器传送存储器分量所需的开销。例如，在能够进行 SIMD 运算的当前处理器中，需要大量的指令向或从存储器传送矢量分量（例如，矢量或数据阵列或其它规则模式的数据）。然而，可利用上述矢量加载和存储指令完成相同的分量传送，其中，降低的指令数量（或者仅用一条指令）可用于完成此传送。

下面的段落提供本发明的各种方面，形式和实施例的说明。每一说明仅提供各种实施例的示例，目的并不是限制上面讨论的任何说明或实施例。

根据本发明的一个方面，一种数据处理系统具有用于存储操作数的存储器和至少一个通用寄存器。处理器电路执行一条或多条指令，并且该一条或多条指令中的至少一条在该存储器和该至少一个通用寄存器间传送数据分量。该至少一条或多条指令指定存储器中数据分量的大小，其独立于指定该至少一个通用寄存器中数据分量的大小。在一种形式中，该一条或多条指令包括独立字段，其用于分别存储用于存储器的第一数据大小指定符和用于该至少一个通用寄存器的第二数据大小指定符。在另一形式中，该一条或多条指令指定用于定义存储器的第一数据大小指定符和该至少一个通用寄存器的第二数据大小指定符的存储位置。在一种形式中，该存储位置为存储器中的位置和该存储器外部的处理器寄存器位置中的任何一个。在另一形式中，该存储位置为该数据处理系统的控制寄存器。在一种形式中，该存储位置为该数据处理系统的寄存器，算术、逻辑和移位操作由该数据处理系统工具执行。该存储器可进一步包括将在该存储器和该至少一个通用寄存器间传送的多个数据分量。在一种形式中，该多个数据分量在该存储器中相邻。在另一形式中，该多个数据分量在存储器中不相邻。在一种形式中，该至少一个通用寄存器中的每一个保存多个数据分量。在另一形式中，该至少一个通用寄存器中的每一个包括具有一维存储器映射的标量寄存器。

当该至少一条或多条指令指定存储器中的源数据分量的大小大于该至少一个通用寄存器中的目标数据分量的大小时，该处理器电路可截短存储器中的源数据分量的一部分。存储器中该源数据分量被截短的部分可为该存储器中源数据分量的高阶部分。存储器中该源数据分量被截短的部分可为该存储器中源数据分量的低阶部分。当该至少一条或多条指令指定存储器中的源数据分量的大小大于该至少一个通用寄存器中的目标数据分量的大小时，该处理器电路可基于该存储器中源数据分量低阶部分的值，舍入（round）该存储器中源数据分量的高阶部分。当该至少一条或多条指令指定存储器中的源数据分量的大小小于该至少一个通用寄存器中的目标数据分量的大小时，该处理器电

路可将预定数据值放入该至少一个通用寄存器的目标数据分量没有被来自该存储器的源数据分量填充的部分。在一种形式中，该处理器电路通过使用零扩展将预定数据值放入该至少一个通用寄存器的目标数据分量的一部分。在另一种形式中，该处理器电路通过使用符号扩展将预定数据值放入该至少一个通用寄存器的目标数据分量的一部分。在还一形式中，该处理器电路通过用预定比特值填充该至少一个通用寄存器的目标数据分量的低阶数据部分，将预定数据值放入该至少一个通用寄存器的目标数据分量的一部分，其中，该预定比特值为零值。

当该至少一条或多条指令指定存储器中的目标数据分量的大小小于该至少一个通用寄存器中的源数据分量的大小时，该处理器电路可截短该至少一个通用寄存器中源数据分量的一部分。在一种形式中，该处理器电路截短该至少一个通用寄存器中源数据分量的高阶部分。在另一种形式中，该处理器电路截短该至少一个通用寄存器中源数据分量的低阶部分。当该至少一条或多条指令指定存储器中的目标数据分量的大小小于该至少一个通用寄存器中的源数据分量的大小时，该处理器电路可根据该源数据分量的低阶部分的值，舍入在该至少一个通用寄存器中的源数据分量的高阶部分。当该至少一条或多条指令指定存储器中的目标数据分量的大小大于该至少一个通用寄存器中的源数据分量的大小时，该处理器电路可将预定数据值放入该存储器中目标数据分量没有被该至少一个通用寄存器中源数据分量填充的部分。在另一形式中，该处理器电路通过使用零扩展将预定数据值放入该存储器中未被填充的目标数据分量部分，或者通过使用符号扩展将预定数据值放入该存储器中未被填充的目标数据分量部分。在另一形式中，通过将预定比特值放置在该目标数据分量的低阶数据部分，将预定数据值放置在该存储器中目标数据分量的未填充部分。

根据另一方面，用于在数据处理系统中加载和存储数据分量的方法包括提供存储操作数的存储器和提供至少一个通用寄存器。执行一条或多条指令，该一条或多条指令中的至少一条引起在该存储器和该

至少一个通用寄存器间的数据分量的传送。该至少一条或多条指令指定该存储器中数据分量的大小，其独立于指定该至少一个通用寄存器中数据分量的大小。通过执行该至少一条或多条指令中的独立字段指定数据分量的大小，第一字段表示用于存储器的第一数据大小指定符，第二字段表示用于该至少一个通用寄存器的第二数据大小指定符。通过执行该至少一条或多条指令中的预定字段指定数据分量的大小，该预定字段表示存储位置，其定义用于存储器的第一数据大小指定符和用于该至少一个通用寄存器的第二数据大小指定符。当该至少一条或多条指令指定存储器中的源数据分量的大小大于该至少一个通用寄存器中的目标数据分量的大小时，可截短该存储器中源数据分量的一部分。在另一种形式中，当该至少一条或多条指令指定存储器中的源数据分量的大小大于该至少一个通用寄存器中的目标数据分量的大小时，根据该存储器中的源数据分量低阶部分的值，舍入该存储器中的源数据分量的高阶部分。在另一形式中，数据处理系统包括存储器和用于执行数据处理指令的处理器。至少一条数据处理指令包括指定存储在该存储器中的数据分量大小的控制信息，其独立于指定存储在该存储器外部的数据处理系统中的至少一个存储位置的数据分量的大小。

在另一形式中，该系统中的处理器电路执行至少第一条指令和该第一条指令之后的第二条指令。该第一条指令在存储器和至少一个通用寄存器间传送数据分量流，这些数据分量将在该存储器和该至少一个通用寄存器的至少一个中进行排队。该第二条指令包括至少第一源操作数，并且基于作为该第二条指令的源操作数的该至少一个通用寄存器，使该数据分量流的一部分条件出队。当该至少一个通用寄存器用作由该第二条指令指定的预定类型功能的源操作数时，可执行该数据分量一部分的条件出队。在另一种形式中，基于该第二条指令的控制字段值执行该部分数据分量的条件出队。该第二条指令还可进一步包括第二源操作数，并且当该至少一个通用寄存器用作该第一源操作数时，执行该条件出队。在一种形式中，当该至少一个通用寄存器作

为该第二源操作数时，不执行该条件出队。

根据另一方面，数据处理系统中数据分量的选择性出队方法包括提供存储操作数的存储器和提供至少一个通用寄存器。执行至少第一条指令和该第一条指令后的第二条指令。在该存储器和该至少一个通用寄存器间传送数据分量流，以响应该第一条指令。对该存储器和该至少一个通用寄存器中的至少一个中的数据分量进行排队。执行该第一条指令之后的第二条指令。该第二条指令包括至少第一源操作数。根据作为该第二条指令的源操作数的该至少一个通用寄存器，使数据分量流的一部分条件出队。当该至少一个通用寄存器用作由该第二条指令指定的预定类型功能的第二源操作数时，使该部分数据分量流条件出队。可在该第二条指令中提供第二源操作数，并且在当该至少一个通用寄存器作为第一源操作数时使该部分数据分量流条件出队。在一种形式中，当该至少一个通用寄存器用作该第二源操作数时不执行出队。

根据另一方面，数据处理系统包括存储操作数的存储器和至少一个通用寄存器。处理器电路执行多条指令，该多条指令中的第一条在该存储器和该至少一个通用寄存器间传送数据分量流，并在该存储器和该至少一个通用寄存器中的至少一个中对这些数据分量进行排队。数据分量流的一部分条件出队，以响应对应于该多条指令的适当子集中预定指令的该多条指令中的第二条。该处理器电路可基于作为该多条指令的第二条的源操作数的该至少一个通用寄存器，进一步使该部分数据分量流条件出队。该处理器电路可基于该多条指令的第二条的控制字段值，进一步使该部分数据分量流条件出队。

根据另一方面，数据处理系统中数据分量的选择性出队方法包括提供存储操作数的存储器和提供至少一个通用寄存器。执行多条指令，该多条指令的第一条在该存储器和该至少一个通用寄存器间传送数据分量流。在该存储器和该至少一个通用寄存器中的至少一个中对该数

据分量流进行排队。使该数据分量流的一部分条件出队，以响应对应于该多条指令的适当子集中预定指令的该多条指令的第二条。

根据另一方面，数据处理系统包括存储操作数的存储器和至少一个通用寄存器。处理器电路执行多条指令。该多条指令的第一条在该存储器和至少一个通用寄存器间传送数据分量流，这些数据分量将在该存储器和该至少一个通用寄存器中的至少一个中进行排队。根据作为该多条指令的第二条的目标操作数的该至少一个通用寄存器，使该数据分量流的一部分条件入队。

根据另一方面，数据处理器中数据分量的选择性入队方法包括提供存储操作数的存储器和提供至少一个通用寄存器。执行多条指令，该多条指令的第一条在该存储器和该至少一个通用寄存器间传送数据分量流，这些数据分量将在该存储器和该至少一个通用寄存器中的至少一个中进行排队。根据作为该多条指令的第二条的目标操作数的该至少一个通用寄存器，使该数据分量流的一部分条件入队。

根据另一方面，数据处理系统包括存储操作数的存储器和至少一个通用寄存器。该处理器电路执行至少第一条指令和该第一条指令后的第二条指令。该第一条指令在存储器和至少一个通用寄存器间传送数据分量流，这些数据分量将在该存储器和该至少一个通用寄存器中的至少一个中进行排队。该第二条指令包括至少第一目标操作数。基于作为该第二条指令的目标操作数的该至少一个通用寄存器，使该数据分量流的一部分条件入队。

根据另一方面，数据处理系统包括存储操作数的存储器和至少一个通用寄存器。该处理器电路执行至少第一条指令和该第一条指令后的第二条指令。该第一条指令在该存储器和该至少一个通用寄存器间传送数据分量流，这些数据分量将在该存储器和该至少一个通用寄存器中的至少一个中进行排队。该第一条指令进一步指定将要传送的数

据分量的数量，并且基于作为该第二条指令的源操作数的该至少一个通用寄存器，使来自该部分数据分量流的的多个数据分量条件出队。

根据另一方面，数据处理系统包括存储操作数的存储器和至少一个通用寄存器。该处理器电路执行至少第一条指令和该第一条指令后的第二条指令。该第一条指令在存储器和至少一个通用寄存器间传送数据分量流，这些数据分量将在该存储器和该至少一个通用寄存器中的至少一个中进行排队。该第一条指令进一步指定将要传送的数据分量的数量，并且基于作为该第二条指令的目标操作数的该至少一个通用寄存器，使来自该部分数据分量流的的多个数据分量条件入队。

根据另一方面，数据处理系统包括存储操作数的存储器、至少一个通用寄存器和处理器电路。该处理器电路包括多条指令。该多条指令中的第一条在存储器和至少一个通用寄存器间传送数据分量流，这些数据分量将在该存储器和该至少一个通用寄存器中的至少一个中进行排队。条件执行该数据分量流的一部分的入队和出队中的至少一个，以响应该多条指令的第二条中的控制字段。

根据另一方面，数据处理系统包括存储操作数的存储器和多个通用寄存器，其中，每个通用寄存器保存多个数据分量。处理器电路执行一条或多条指令，该一条或多条指令中的至少一条在该存储器和该多个通用寄存器中的至少两个间传送多个数据分量，其中，该至少一条或多条指令指定将要在该多个通用寄存器中的至少两个中的每一个和该存储器间传送的寄存器分量的数量。该一条或多条指令除了指定将要传送的寄存器分量的数量外，还指定将加载或存储哪一个寄存器分量。在另一种形式中，当在该多个通用寄存器中的至少两个中的每一个和该存储器间仅传送该多个数据分量的子集时，用预定值填充任何未指定的数据分量。

根据另一方面，在数据处理系统中传送数据分量的方法包括在存

储器中存储操作数。在多个通用寄存器的每一个中存储多个数据分量。该数据处理系统执行一条或多条指令。该一条或多条指令中的至少一条引起在该存储器和该多个通用寄存器中的至少两个间的多个数据分量的传送。该至少一条或多条指令指定将要在该多个寄存器中的至少两个中的每一个和该存储器间传送的寄存器分量的数量。

根据另一方面，数据处理系统包括存储操作数的存储器和多个通用寄存器，其中，每个通用寄存器保存多个数据分量。处理器电路执行一条或多条指令，该一条或多条指令中的至少一条在该存储器和该多个通用寄存器中的至少两个间传送多个数据分量。该至少一条或多条指令指定将要传送该多个通用寄存器中的至少两个中的哪些数据分量。当仅传送多个数据分量的子集时，该处理器电路用预定值填充任何未指定的数据分量。在一种形式中，由该至少一条或多条指令指定的数据分量在该多个通用寄存器中的至少两个中连续排列。在另一种形式中，由该至少一条或多条指令指定的数据分量在该多个通用寄存器中的至少两个中不是连续排列。

根据另一方面，在数据处理系统中传送数据分量的方法包括在存储器中存储操作数和在多个通用寄存器中的每一个中保存多个数据分量。执行一条或多条指令，该一条或多条指令中的至少一条在该存储器和该多个通用寄存器中的至少两个间引起多个数据分量的传送，其中，至少一条或多条指令指定将要传送该多个通用寄存器中的至少两个中的哪些数据分量。

根据另一方面，一种数据处理系统包括存储操作数的存储器和多个通用寄存器，其中，每一通用寄存器保存多个数据分量。处理器电路执行一条或多条指令，该一条或多条指令中的至少一条在该存储器和该多个通用寄存器中的至少两个间传送多个数据分量。该至少一条或多条指令既指定将要在该多个寄存器中的至少两个中的每一个和该存储器间传送的数据分量的数量，还进一步指定将要传送的数据分量

的总数。

根据另一方面，数据处理系统包括存储操作数的存储器和至少一个通用寄存器。处理器电路执行一条或多条指令，该一条或多条指令中的至少一条用于在该存储器和该至少一个通用寄存器间传送多个数据分量，其中，该一条或多条指令指定：（a）存储器中连续数据分量间的第一部分中数据分量间的第一偏移；（b）将在该存储器和该至少一个 GPR 间传送的数据分量的第一数量；以及（c）该存储器中数据分量的第一部分和第二部分间的第二偏移。在一种形式中，该一条或多条指令中的这一条还指定存储器中数据分量的数据分量大小。在另一形式中，该一条或多条指令中的这一条还指定该存储器中数据分量的大小，其独立于指定该至少一个通用寄存器中数据分量的大小。该处理器电路基于该存储器中数据分量的大小，确定将要传送的数据分量的总数。该处理器电路可基于该至少一个通用寄存器中数据分量的大小，确定将要传送的数据分量的总数。该一条或多条指令中的这一条可进一步指定将要在该存储器和该至少一个通用寄存器间传送的数据分量的总数。该数据处理系统可进一步包括第一通用寄存器和第二通用寄存器，其中，该一条或多条指令中的这一条在该存储器和第一通用寄存器与第二通用寄存器间传送数据分量，以响应执行该一条或多条指令中的该条指令。该一条或多条指令中的这一条指令可进一步指定将要在该存储器和第一通用寄存器与第二通用寄存器间传送的数据分量的总数。如果传送的数据分量的总数并没有完全填充该第二通用寄存器，则处理器电路可用预定值填充任何剩余比特位置的一部分。该一条或多条指令中的这一条指令可独立地指定将要在该存储器和该第一与第二通用寄存器中的每一个间传送的数据分量的数量。如果传送的数据分量的总数并没有完全填充该第二通用寄存器，则该处理器电路可用预定值填充任何剩余比特位置的至少一部分。该一条或多条指令中的这一条指令可进一步包括指示符，其中，该处理器电路使用该第二偏移不超过一次，同时传送第一数量的数据分量。当该一条或多条指令中的这一条指定仅使用该第二偏移一次时，该处理器电路可

利用循环缓冲发送存储器中的数据分量。该一条或多条指令中的这一条可进一步包括指示符，其中，如果将要传送的数据分量的第一数量是将要传送的数据分量的第一部分的两倍，则该处理器电路使用该第二偏移多于一次。

根据另一方面，数据处理系统包括存储操作数的存储器和至少一个通用寄存器。处理器电路执行一条或多条指令，该一条或多条指令中的至少一条在该存储器和该至少一个通用寄存器间传送数据分量，其中，该一条或多条指令的一条指定基数指示符，用于在该存储器和该至少一个通用寄存器间以比特翻转的顺序执行传送一个或多个数据分量。

根据另一方面，一种使用多个寻址模式的方法提供存储操作数的存储器。提供至少一个通用寄存器。执行一条或多条指令，该一条或多条指令中的至少一条在该存储器和该至少一个通用寄存器间传送数据分量。该一条或多条指令中的至少一条指定该存储器中连续数据分量的第一部分中数据分量间的第一偏移。该一条或多条指令中的至少一条指定将在该存储器和该至少一个通用寄存器（GPR）间传送的数据分量的第一数量。该一条或多条指令中的至少一条可进一步指定该存储器中数据分量的第一部分和第二部分间的第二偏移。该一条或多条指令中的至少一条可用于进一步指定存储器中数据分量的数据分量大小。该一条或多条指令中的至少一条可用于进一步指定该存储器中数据分量的大小，其独立于指定该至少一个通用寄存器中数据分量的大小。处理器基于该存储器中数据分量的大小，用于确定将要传送的数据分量的总数。该处理器可基于该至少一个通用寄存器中数据分量的大小，确定将要传送的数据分量的总数。该一条或多条指令中的至少一条指令可用于进一步指定将要在该存储器和该至少一个通用寄存器间传送的数据分量的总数。可提供第一通用寄存器和第二通用寄存器并且在该存储器和第一通用寄存器与第二通用寄存器间传送数据分量，以响应执行该一条或多条指令中的该条指令。该一条或多条指令

中的这一条指令可用于进一步指定将要在该存储器和第一通用寄存器与第二通用寄存器间传送的数据分量的总数。在一种形式中，如果传送的数据分量的总数并没有完全填充该第二通用寄存器，则用预定值填充该第二通用寄存器中任何剩余的未填充的比特位置的至少一部分。该一条或多条指令中的这一条指令可用于进一步独立地指定将要在该存储器和该第一与第二通用寄存器中的每一个间传送的数据分量的数量。在一种形式中，如果传送的数据分量的总数并没有完全填充该第二通用寄存器，则利用诸如零的预定值填充任何剩余比特位置的至少一部分。可在该一条或多条指令中的这一条中提供指示符，其中，作为该指示符的响应，处理器仅使用该第二偏移一次，来传送第一数量的数据分量。

根据另一方面，当该一条或多条指令中的这一条指定仅使用该第二偏移一次时，则可利用循环缓冲，在该处理器的控制之下，发送存储器中的数据分量。可在该一条或多条指令中的这一条中提供指示符，其中，作为该指示符的响应，如果将要传送的第一数量的数据分量是将要传送的存储器中数据分量的第一部分中的数据分量数量的两倍，则该处理器使用该第二偏移多于一次。可该一条或多条指令的这一条中提供基数指示符，用于在该存储器和该至少一个通用寄存器间以比特翻转的顺序执行一个或多个数据分量的传送。

根据另一方面，一种数据处理寻址方法包括提供存储操作数的存储器和提供至少一个通用寄存器。执行一条或多条指令，该一条或多条指令中的至少一条在该存储器和该至少一个通用寄存器间传送数据分量。该一条或多条指令中的至少一条在该一条或多条指令的这一条中指定基数指示符，该基数指示符在该存储器和该至少一个通用寄存器间以比特翻转的顺序执行一个或多个数据分量的传送。

根据另一方面，一种数据处理系统寻址方法包括提供存储操作数的存储器和提供至少一个通用寄存器。提供处理器电路以执行一条或

多条指令。该一条或多条指令中的至少一条在该存储器和该至少一个通用寄存器间传送数据分量，其中，该一条或多条指令中的一条执行以比特翻转的顺序将预定数据分量存储至存储器中，并将该预定数据分量传送至该至少一个通用寄存器。该一条或多条指令中的至少一条进一步指定将要传送至该至少一个通用寄存器中的数据分量的数量。在另一形式中，该一条或多条指令中的至少一条进一步指定该存储器中数据分量的大小，其独立于指定该至少一个通用寄存器中数据分量的大小。

根据另一方面，一种数据处理系统寻址方法包括提供存储操作数的存储器和提供至少一个通用寄存器。提供处理器电路以执行一条或多条指令，该一条或多条指令中的至少一条在该存储器和该至少一个通用寄存器间传送数据分量，其中，该一条或多条指令中的一条在该存储器中以连续顺序存储预定数据分量，并且以比特翻转顺序将该预定数据分量传送至该至少一个通用寄存器。

根据另一方面，一种数据处理系统寻址方法包括提供存储操作数的存储器和提供至少一个通用寄存器。提供处理器电路以执行一条或多条指令，该一条或多条指令中的至少一条在该存储器和该至少一个通用寄存器间传送数据分量，其中，该一条或多条指令中的一条在该至少一个通用寄存器中执行以比特翻转顺序存储预定数据分量，并且将该预定数据分量传送至该存储器。

根据另一方面，一种数据处理系统寻址方法包括提供存储操作数的存储器和提供至少一个通用寄存器。提供处理器电路以执行一条或多条指令，该一条或多条指令中的至少一条在该存储器和该至少一个通用寄存器间传送数据分量，其中，该一条或多条指令中的一条在该至少一个通用寄存器中以连续顺序存储预定数据分量，并且以比特翻转顺序将该预定数据分量传送至该存储器。

在前述说明中，参考特定实施例、示例和方面说明了本发明。然而，本领域的普通技术人员明白，可进行各种修改和变化，而不脱离权利要求说明的本发明的精神与范围。例如，框图可包括与已表示的方框不同的方框，并且可具有更多或更少的方框，或不同的布置。另外，作为另一示例，此处说明的指令可具有不同于那些已经说明的指令的各种不同格式，并使用不同的控制字段和参数。每一条指令甚至可作为多条指令执行。另外，作为另一示例，此处说明的寄存器可为位于该数据处理系统任何地方的任何类型的存储电路。因此，说明书与附图将视为说明意义，而不是限定意义，并且所有这样的改变将包含在本发明的范围之内。

已经针对特定实施例说明了益处、优势和问题的解决方案。然而，这些益处、优势和问题的解决方案以及使任何益处、优势、问题的解决方案出现或变得更加显著的任何要素将不被视为任何或所有权利要求的关键的、必须的或本质的特征或要素。如此处所使用，术语“包括”，或其任何变体，将涵盖非排它性的包涵，使得包括一系列要素的过程、方法、物品或装置不仅包括那些要素，还可包括没有明确列出的，或这些过程、方法、物品或装置固有的其它要素。

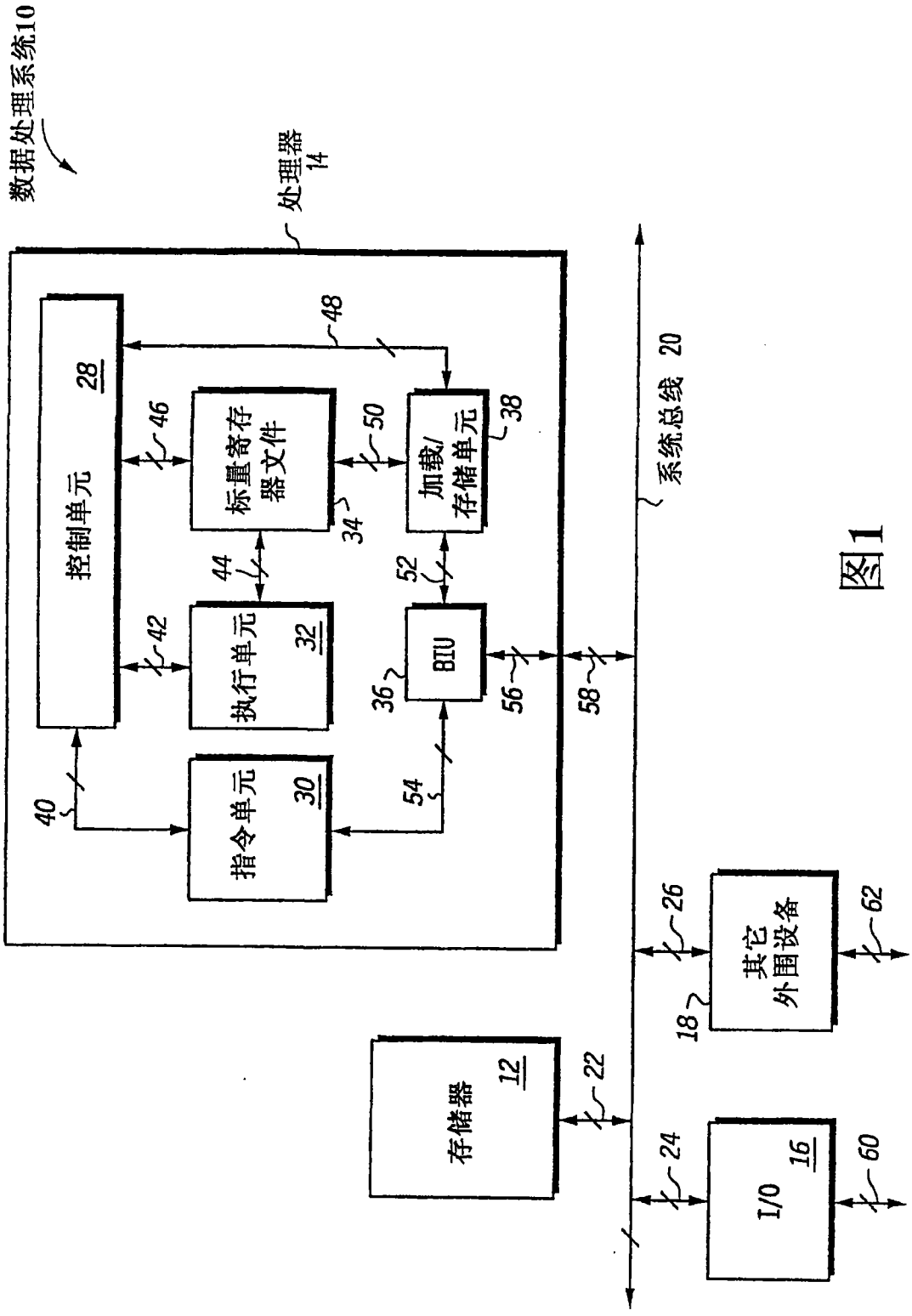

数据处理系统10
处理器
14
控制单元
28
标量寄存器文件
加载/存储单元
38
执行单元
32
BIU
指令单元
30
系统总线 20
存储器
12
其它外围设备
I/O
16
40
42
44
46
48
50
52
54
56
58
36
34
18
22
24
26
60
62

图1

lvex.[s/u].[ms].[ds] rD, rA, rB

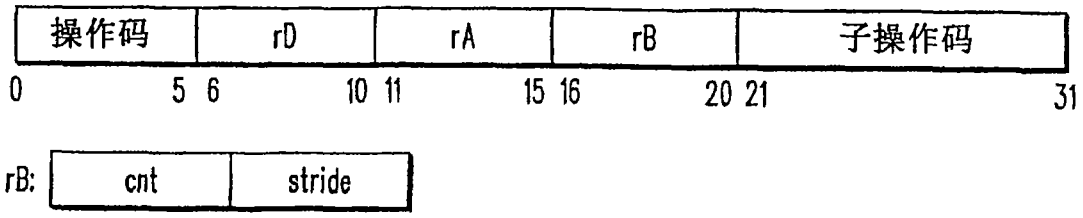


图2

lvex.[s/u].[ms].[ds] rD, rA, cnt, stride

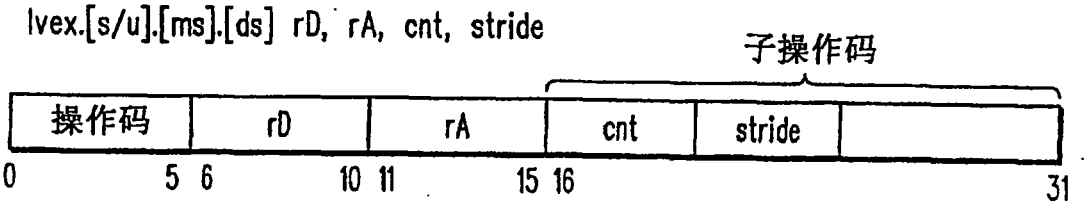


图3

lmvex.[s/u].[ms].[ds] rD, rA, rB

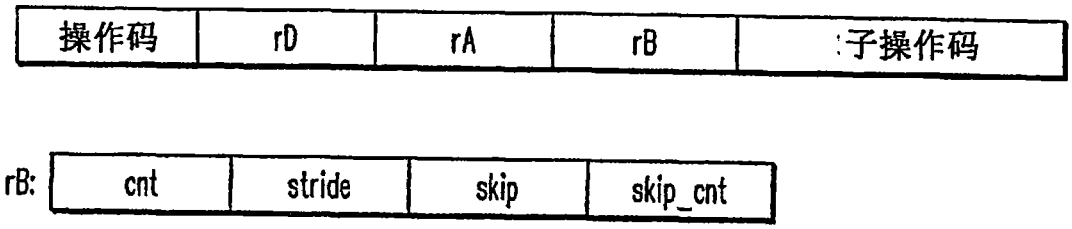


图4

lmvex.[s/u].[ms].[ds] rD, rA, cnt, stride, skip, skip_cnt

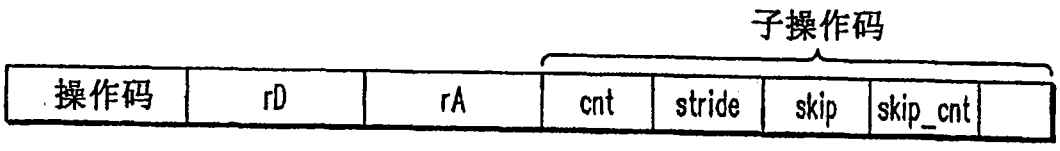


图5

lmvex2.[s/u].[ms].[ds] rD, rA, rB

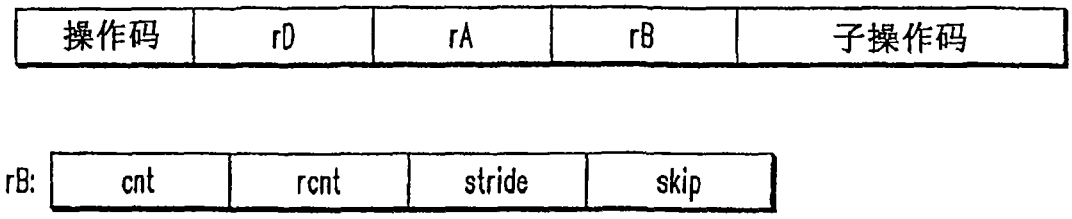


图6

lstrmvex.[s/u].[ms].[ds] rD, rA, rB

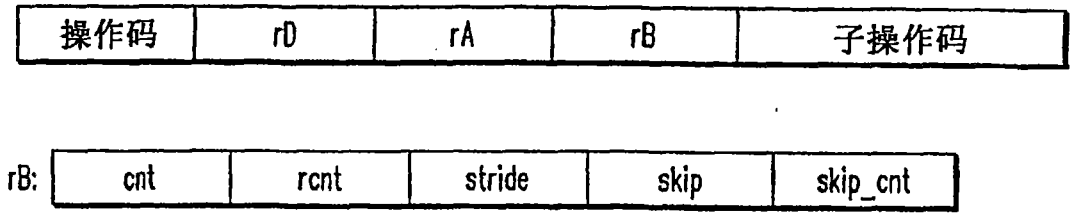


图7

stvex.[s/u].[ms].[ss].[h/l] rS, rA, rB

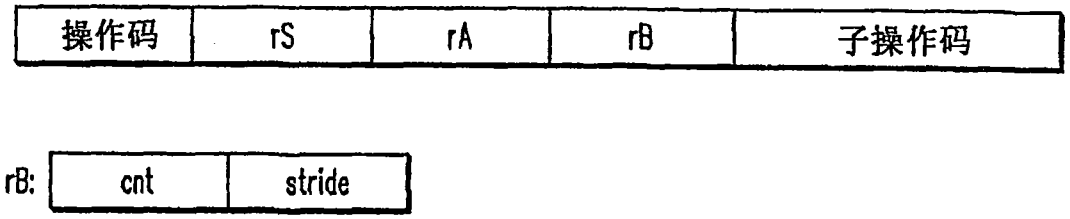


图8

stmvex.[s/u].[ms].[ss].[h/l] rS, rA, rB

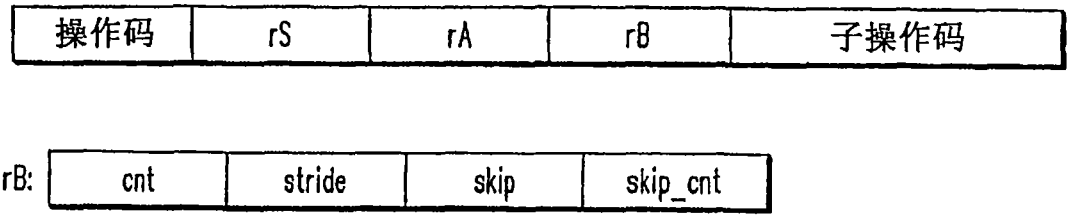


图9

stmvex2.[s/u].[ms].[ss].[h/l] rS, rA, rB

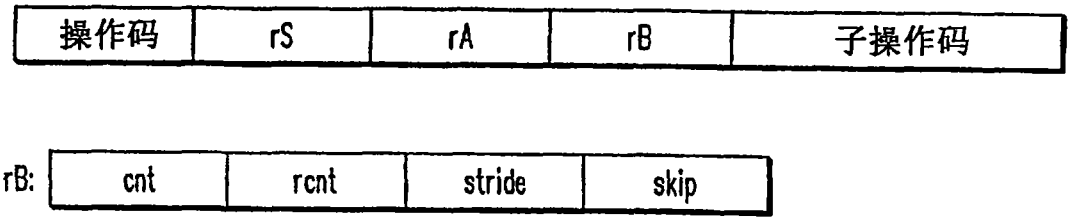


图10

ststrmvex.[s/u].[ms].[ss].[h/l] rS, rA, rB

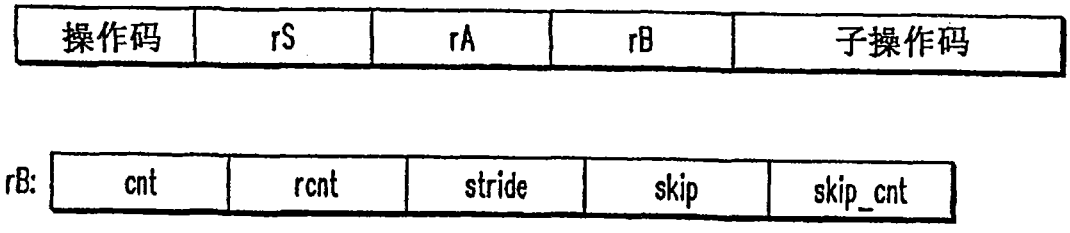


图11

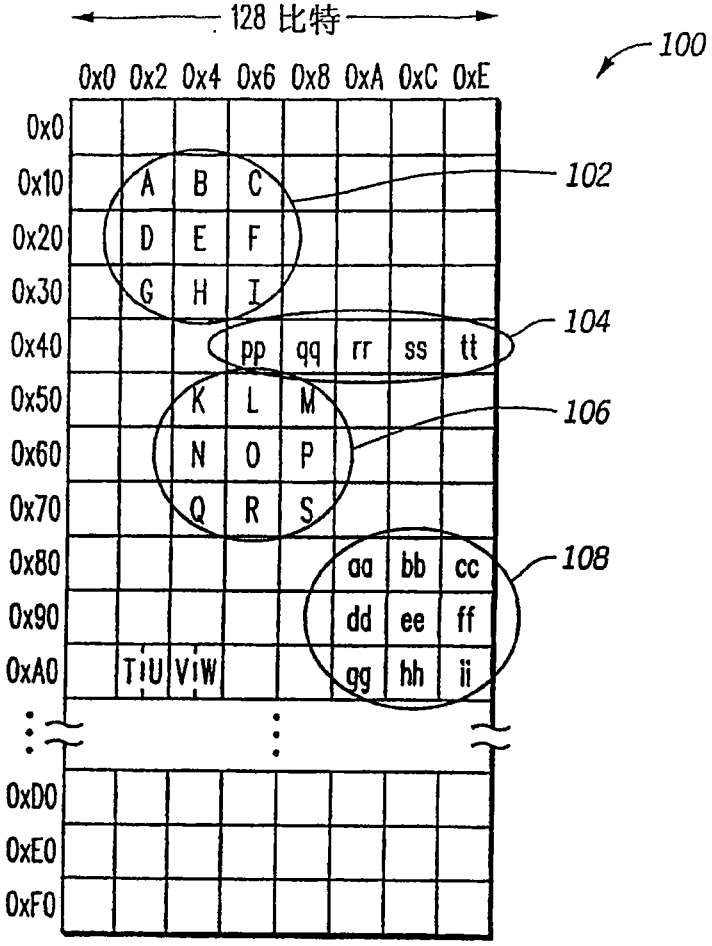

128 比特
100
0x0 0x2 0x4 0x6 0x8 0xA 0xC 0xE
0x0
0x10 A B C
0x20 D E F
0x30 G H I
0x40 pp qq rr ss tt
0x50 K L M
0x60 N O P
0x70 Q R S
0x80 aa bb cc
0x90 dd ee ff
0xA0 T U V W gg hh ii
0xD0
0xE0
0xF0
102
104
106
108

图12

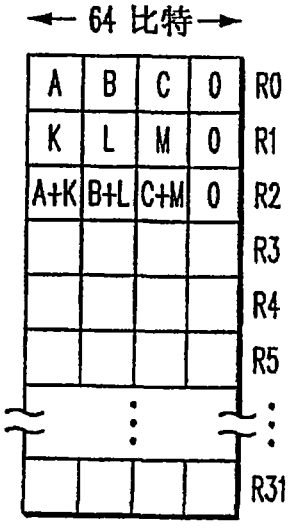

64 比特
A B C 0 R0
K L M 0 R1
A+K B+L C+M 0 R2
R3
R4
R5
R31

图13

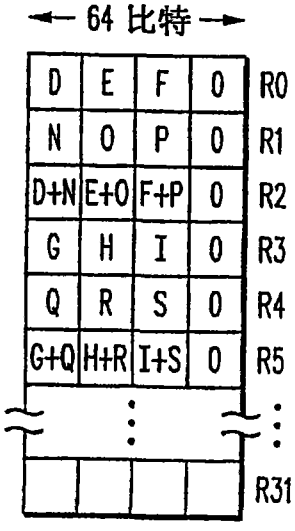

64 比特
D E F 0 R0
N O P 0 R1
D+N E+O F+P 0 R2
G H I 0 R3
Q R S 0 R4
G+Q H+R I+S 0 R5
R31

图14

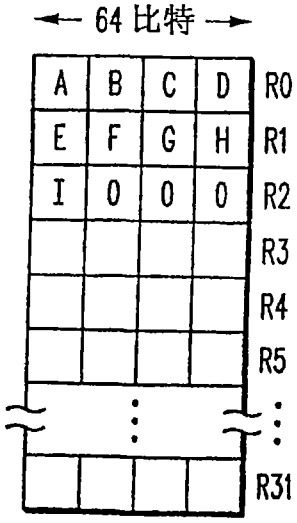

64 比特
A B C D R0
E F G H R1
I 0 0 0 R2
R3
R4
R5
R31

图15

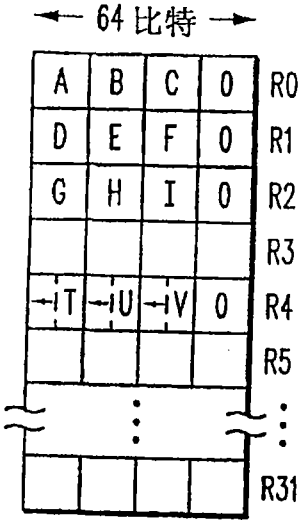

64 比特
A B C 0 R0
D E F 0 R1
G H I 0 R2
R3
T U V 0 R4
R5
R31

图16

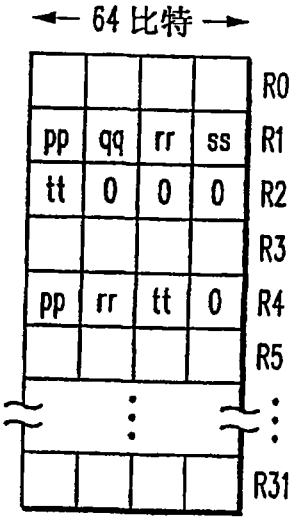

64 比特
R0
pp qq rr ss R1
tt 0 0 0 R2
R3
pp rr tt 0 R4
R5
R31

图17

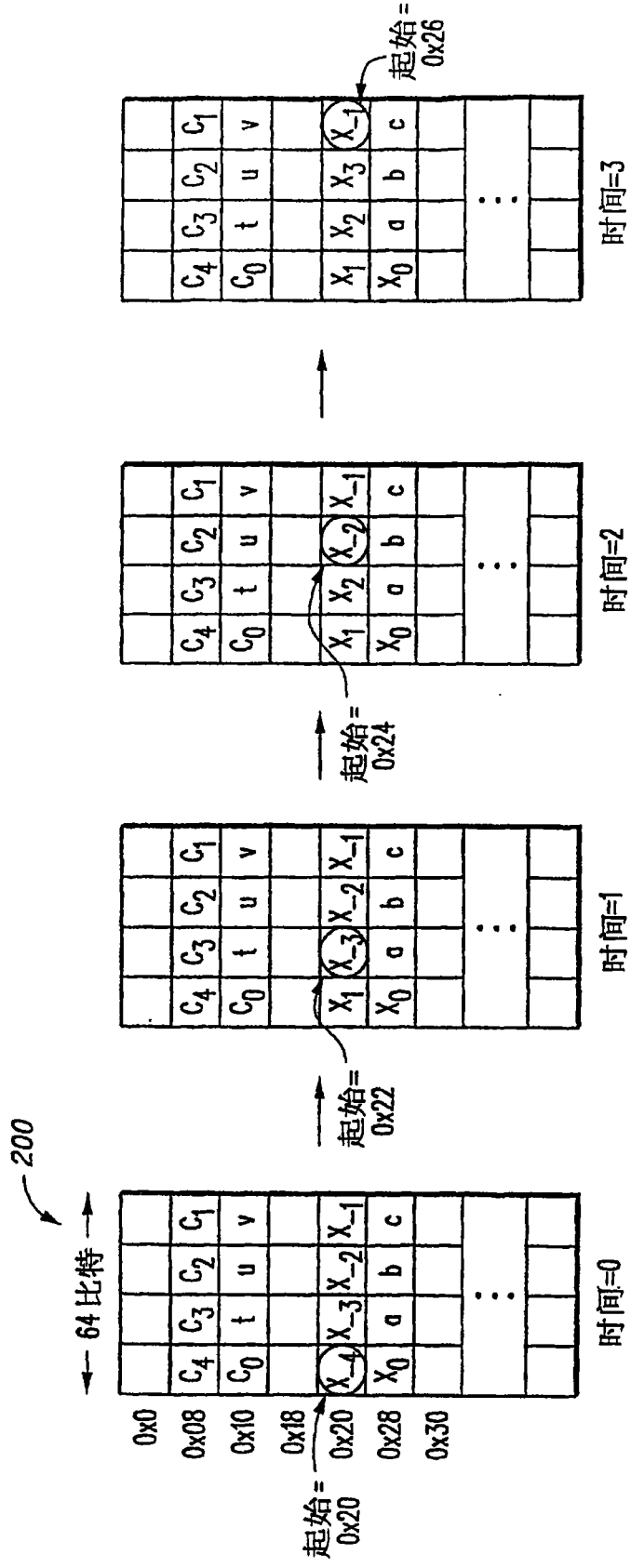

200
64 比特
0x0
0x08
0x10
0x18
0x20
0x28
0x30
起始=
0x20
起始=
0x22
起始=
0x24
起始=
0x26
时间=0
时间=1
时间=2
时间=3

图18

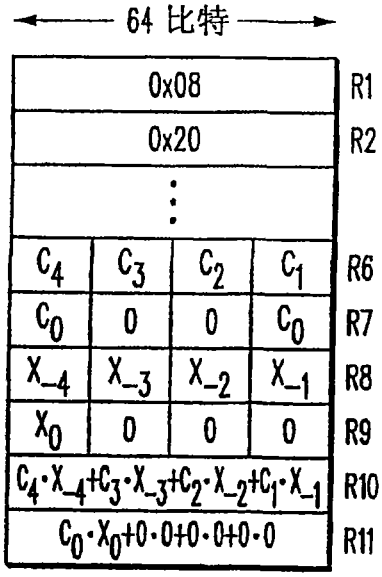


图19

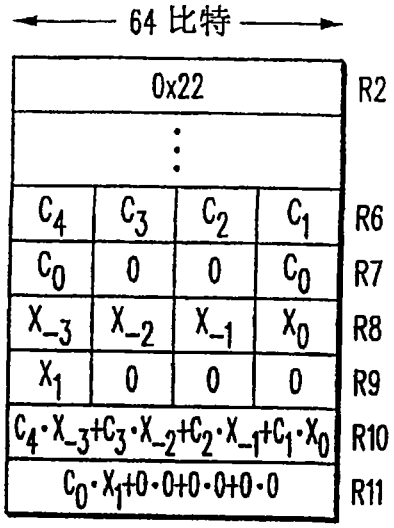


图20

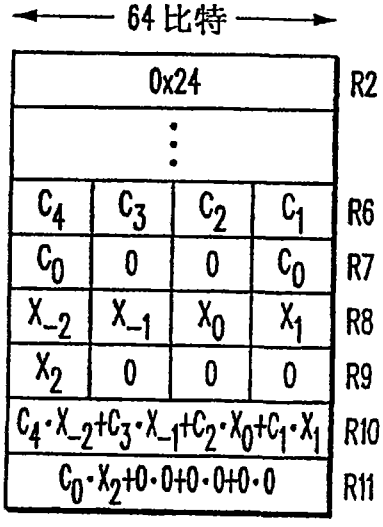


图21

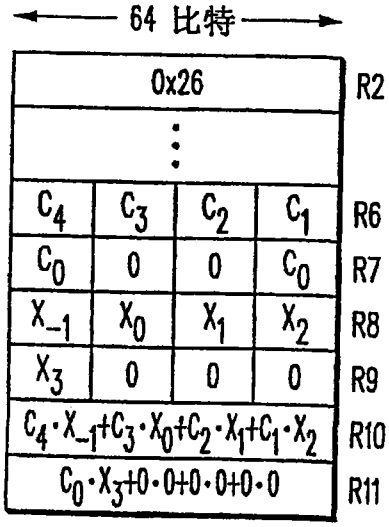


图22

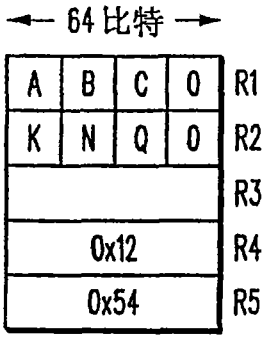


图23

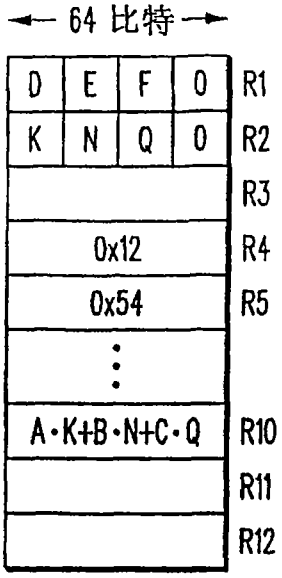


图24

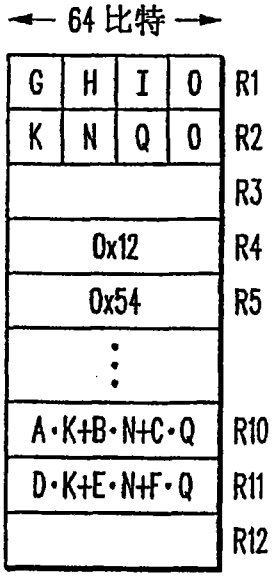


图25

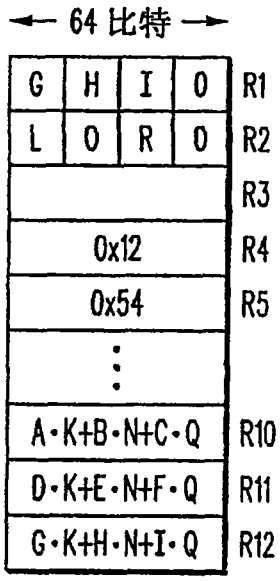


图26

lmvex_skip_once.[s/u].[ms].[ds] rD, rA, rB

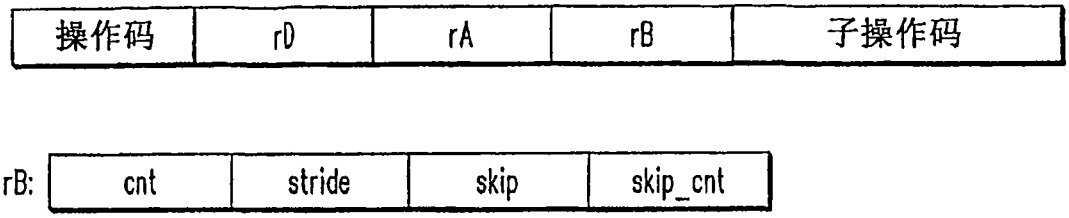


图27

lmvex_cb.[s/u].[ms].[ds] rD, rA, rB

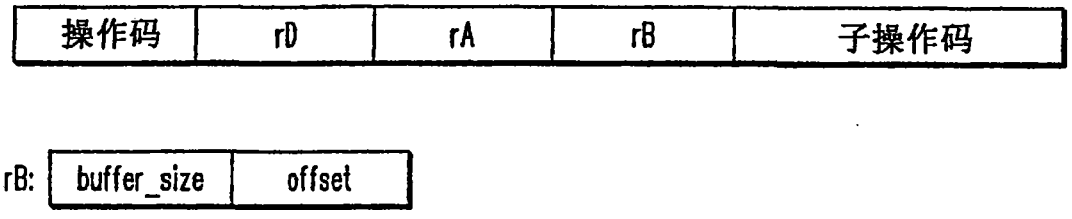


图28

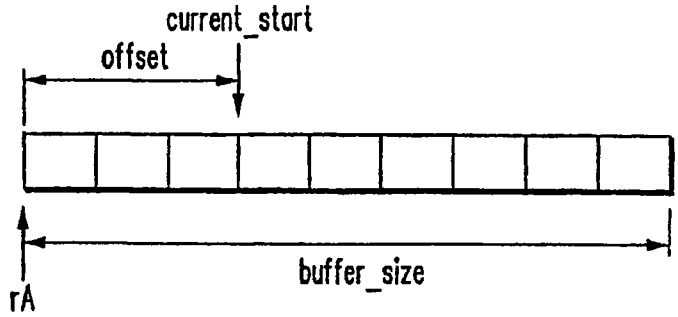


图29

lstrmvex_cb.[s/u].[ms].[ds] rD, rA, rB

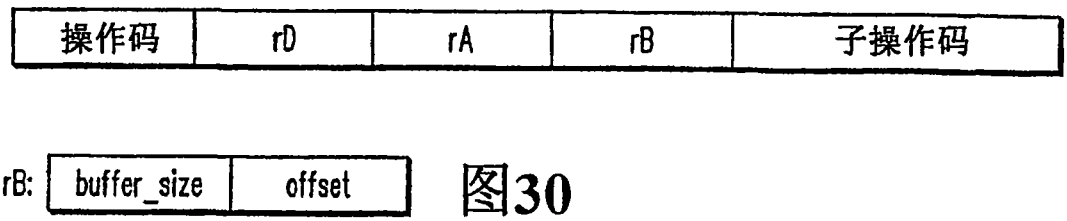


图30

lmvex_fft.[s/u].[ms].[ds] rD, rA, rB





图31

stmvex_fft.[s/u].[ms].[ss] rS, rA, rB

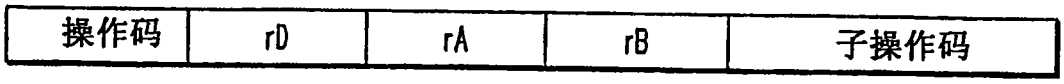




图32

lmstrmvex_fft.[s/u].[ms].[ds] rD, rA, rB





图33

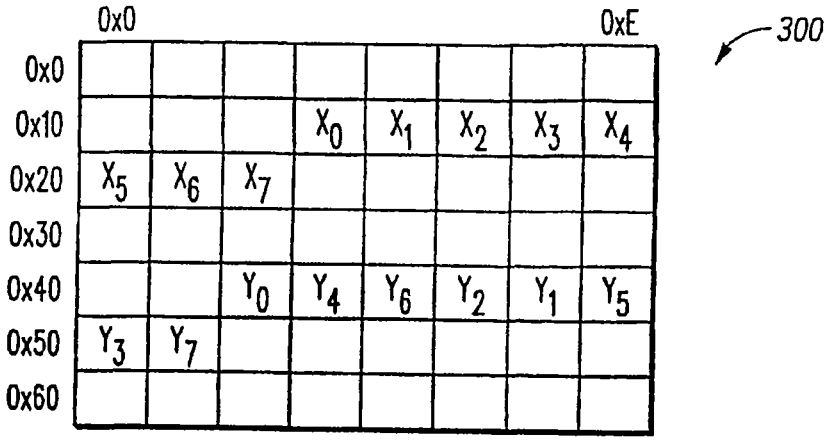


图34

X_0	X_4	X_6	X_2	R1
X_1	X_5	X_3	X_7	R2
				R3
Y_0	Y_1	Y_2	Y_3	R4
Y_4	Y_5	Y_6	Y_7	R5

图35