[19]中华人民共和国国家知识产权局

[51]Int. Cl[7]
G06F 12/00
G06F 13/14

[12]发明专利申请公开说明书

[21]申请号 99815655.8

[43]公开日 2002年5月8日

[11]公开号 CN 1348564A

[22]申请日 1999.11.15 [21]申请号 99815655.8
[30]优先权
[32]1998.11.16 [33]US [31]60/108,930
[32]1999.11.12 [33]US [31]09/439,860
[86]国际申请 PCT/US99/27017 1999.11.15
[87]国际公布 WO00/29956 英 2000.5.25
[85]进入国家阶段日期 2001.7.16
[71]申请人 因芬尼昂技术股份公司
地址 德国慕尼黑
[72]发明人 H·斯特拉科夫斯基
P·斯扎贝尔斯基

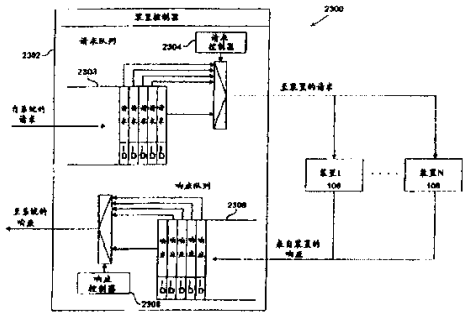
[74]专利代理机构 中国专利代理(香港)有限公司
代理人 吴立明 王忠忠

权利要求书1页 说明书32页 附图页数31页

[54]发明名称 分优先级访问外部装置的方法和设备

[57]摘要

按照本发明,一种分优先级访问外部装置的设备,包括请求队列(2303),用于存储外部装置的任何数量的请求的装置的请求,连接 到请求队列的请求队列控制器单元(2304)用于读取存储在那里的任 何请求,响应队列(2306)用于存储来自外部装置的任何数量的响应。该设备还包括连接到响应队列的响应队列控制器(2308),用于读取 存储在那里的任何请求,其中各个响应及其相关的请求用一组识别编 号(150)联系在一起,识别编号表明发出请求的请求装置的特定的 组和相应的响应是注定的,其中的相应队列控制器和请求队列控制器 单元使用存储在组优先级选择器寄存器(154)内的优先级编号,优 先级化每个存储的请求和响应,这样,具有较高优先级的请求或者响 应就可以绕过具有较低优先级的请求或者响应。



ISSN1008-4274

权　利　要　求　书

1. 一种分优先级访问外部装置的设备，包括：

请求队列，用于存储外部装置的任何数量的请求的装置的请求；

连接到请求队列的请求队列控制器单元，用于读取存储在其中的任何请求；

响应队列，用于存储来自外部装置的任何数量的响应；

连接到响应队列的响应队列控制器，用于读取存储在其中的任何请求，其中各个响应及其相关的请求用一组标识编号联系在一起，标识编号表明发出请求的请求装置的特定的组和相应的响应的目的地，其中的相应队列控制器和请求队列控制器单元使用存储在组优先级选择器寄存器内的优先级编号，优先级化每个存储的请求和响应，这样，具有较高优先级的请求或者响应就可以绕过具有较低优先级的请求或者响应。

2. 如权利要求 1 的设备，还包括活锁计数器寄存器，用于在这些情形下启动：具有较高优先级的请求或者响应绕过具有较低优先级的请求或者响应，因此，就可以基本上避免活锁情况。

01.07.16

说　明　书

分优先级访问外部装置的方法和设备

发明背景

本发明总的涉及计算系统。更具体地，本发明涉及访问计算系统如多处理器计算机系统等内的共享的资源。更进一步说，本发明描述分优先级访问外部装置的方法和设备。

发明背景

在基本的计算机系统中，中央处理器或 CPU，按照存储在相应的存储器内的预定的程序或指令集来运行，除了存储处理器借助运行的指令集或程序外，还提供了在处理器存储器或相关的附加的存储器中的存储空间，以便于中央处理器在处理过程中处理信息。基于哪一个处理器用于执行程序，附加的存储器用作由处理器生成的信息的存储器和临时信息的存储器，或“中间结果暂存器（scratchpad）”。另外，为了利用系统的输出装置，相关的存储器提供放置处理器操作指令集的输出信息的位置。

在为了访问内存，很多组件（处理器、硬盘等）必须共享公共的总线的系统中，很可能出现存储器访问冲突。尤其是在多处理器计算机系统中，系统使用不同的处理器同时的操作，访问内存或共享的资源，变得复杂了。由于各个处理器或处理器系统很可能同时请求访问同一个内存，处理器之间的冲突通常是不可避免的。实质上，在多处理器计算机系统中，两个或者多个处理器或处理器系统周期性的导致关于公共内存的存储指令的重叠。

解决共享内存的存储器访问请求的冲突的传统方法包括，一种情况是，各个处理器使用完全冗余的内存，并隔离处理器系统。但是，这种解决冲突的存储器访问请求的方法经常不能发挥多处理器系统的优势。如果按照同一数据提供并行的计算的方式操作，系统中一个处理器支持其它处理器的操作，这样的多处理器系统极其高效。传统地，这种处理器系统可能是分时的（time shared），其中的处理器竞争访问共享的资源如内存，或者处理器系统是双重端口的，其中的各处理器由自己的内存总线，比如其它的已经给予访问时，对其进行排队。

已经有多种方法用于避免上述的冲突问题。一种方法是，通过顺序操作处理器或者通过分时操作处理器，来达到避免冲突。这种方法中，为了避免冲突，处理器只是的“依次”访问共享的资源。通常使用的这种系统包括“通行铃（passing the ring）”或“标记系统”，系统中，可能冲突的处理器被系统简单的轮询，按照预定的与关于一组用户的通行铃同样的顺序。

不幸的是，使用顺序的处理器访问方法对整个计算机系统有很大的限制。这种限制源自于系统轮询竞争的处理器所用的实际时间。另外，如果单个的处理器在运行并请求访问共享的内存，比如，由于系统单步调试序列，处理器访问共享的资源时，在每个内存周期后有一个延时产生。

另一个传统的避免冲突的方法是依靠在计算机系统内的处理器中建立优先级。这种方法假定各个处理器具有分配给它的关于系统重要性层次的优先权。每次出现冲突时，存储器控制器只为具有最高优先权的处理器提供访问。比如，在一个具有两个处理器的系统中，第一和第二处理器访问共享的存储器，该存储器一般是要求周期性刷新存储器所保留的存储数据的动态 RAM（DRAM）型存储器。一般地，DRAM 型存储器由单独的独立更新系统刷新。在这样的多处理系统中，两个处理器和更新系统都争相访问公共存储器。系统存储器控制器将处理存储器访问请求冲突，或者命令，正如分配给处理器的不同的优先权和更新系统所决定的那样。尽管这种系统解决了冲突，并且在一定程度上比纯粹的顺序冲突避免系统的效率更高，但该系统仍然缺乏灵激活。

另一种解决冲突的方法包括在存储器控制器中加入决策能力。遗憾的是，存储器控制器的决策部分要在时钟系统的控制和时序下操作，问题就来自于，在存储器控制器能够授权访问公共存储器之前，用来执行真正的决策的实际时间。

不幸的是，执行真正的决策的问题很大程度上削弱了传统的存储器控制器授权访问多条型存储器系统的能力。在多存储体（multi-bank）型存储器系统中，实际的存储器核心被分成指定的区域和组，其中存储着要检索的数据。虽然准备了更快更有效的存储器访问，但传统的存储器控制器在处理多存储体型存储装置时所要求的复杂性

实际上减缓了整个系统的总的访问速度。

如上所述，很显然，可以期望有一种分优先级访问外部装置的方法和设备。

发明概述

按照本发明，公开了一种分优先级访问外部装置的方法和设备。分优先级访问外部装置的设备包括请求队列，用于存储外部装置的任何数量的请求的装置的请求；连接到请求队列的请求队列控制器单元，用于读取存储在那里的任何请求；响应队列，用于存储来自外部装置的任何数量的响应。该设备还包括连接到响应队列的响应队列控制器，用于读取存储在那里的任何请求；其中各个响应及其相关的请求用一组识别编号联系在一起，识别编号表明发出请求的请求装置的特定的组和相应的响应的目的地；其中的相应队列控制器和请求队列控制器单元使用存储在组优先级选择器寄存器内的优先级编号，优先级化每个存储的请求和响应，这样，具有较高优先级的请求或者响应就可以绕过（bypass）具有较低优先级的请求或者响应

参考说明书的其余部分和附图，可以更进一步的理解本发明的特性和优点。

附图简述

下面借助附图中的实例来解释本发明，而不意味着限制；附图中相同的标号代表相同的单元，其中：

图 1A 示出了符合本发明一个实施方案的通用控制器的概略的实现；

图 1B 示出了如图 1A 所示的通用控制器的具体实现；

图 1C 示出了符合本发明一个实施方案的连接到通用控制器的地址空间控制器；

图 1D 示出了如图 1C 所示的地址空间控制器的具体实现；

图 1E 示出了按照本发明一个实施方案的典型的请求/响应 ID 编号；

图 2A 示出了按照本发明一个实施方案的一般的通用指令；

图 2B 示出了适合于请求存储页读指令的如图 2A 中所示的具体的通用指令；

图 2C 示出了由图 2B 所示的典型指令的指令组成之间提供适当的

时序间隔所构成的顺序指令的示例;

图 3 示出了按照本发明一个实施方案的资源标志;

图 4 示出了按照本发明一个实施方案，通用控制器详细处理访问共享的资源的流程图;

图 5 示出了按照本发明一个实施方案，通用控制器确定资源的状态和执行操作的步骤的处理;

图 6 示出了按照本发明一个实施方案，通用控制器根据处理确定操作步骤之间适当的时序的处理;

图 7A 和 7B 示出了按照本发明一个实施方案，页命中/错过控制器;

图 8 示出了按照本发明一个实施方案的存储体访问控制器;

图 9A 示出了按照本发明一个实施方案的基于多处理器系统的典型的 SLDRAM;

图 9B 为一个时序图示出了如图 9A 所示的多处理器的典型的 SLDRAM 总线交易;

图 10 是一个符合本发明一个实施方案的存储器控制器的方框图;

图 11 是一个符合本发明一个实施方案的约束模块的方框图;

图 12 是符合本发明一个实施方案的典型的 SLDRAM 指令时序图;

图 13A-13C 示出了存储器指令的重排序的时间线，按照本发明的一个特殊的实施方案;

图 14 是一个按照本发明的一个特殊实施方案设计的存储器控制器的部分方框图;

图 15 是一个按照本发明的一个特殊实施方案设计的重排序电路的方框图;

图 16，如图 15 的重排序电路的更详细的方框图;

图 17 是一个符合本发明的一个特殊实施方案的指令队列项内容图;

图 18 是一个地址移位器的特殊实施方案的方框图;

图 19 是一个符合本发明的一个特殊实施方案的数据队列项内容图;

图 20 示出了一个冲突检测系统，也就是如图 15 所示的冲突检测

系统的另一个实现；

图 21 示出了一个典型的时序图，说明每次读/写指令到目标装置是如何涉及数据包传输的；

图 22 示出了一个预测器系统，它具有 N 页时钟，存储上一次发布到特定页的指令与预测的下一个内存的访问之间的时间；和

图 23 示出了按照本发明一个实施方案的具有装置访问优先权的装置控制器。

图 24，示出的表 4，汇总了按照本发明一个实施方案的约束模块执行的调度器。

优选实施方案详述

在具有几个装置如处理器，共享公共的资源的系统中，已经使用了多种方法来避免当不止一个装置请求访问共享资源时的冲突。一种方法是，通过顺序操作处理器和通过分时处理器来解决冲突。这时，为了避免冲突，处理器只是“依次地”访问共享的资源。通常使用的这种系统包括“通行铃”或“标记系统”，系统中可能冲突的处理器被系统简单的轮询，按照预定的与关于一组用户的通行铃同样的顺序。

遗憾的是，使用顺序的处理器访问方法通常会对整个计算机系统有很大的限制，由于用于轮询竞争的处理器的实际时间总量。

另一个传统的避免冲突的方法是依靠在计算机系统内的处理器中建立优先级。这种方法假定各个处理器具有分配给它的关于系统重要性层次的优先权。尽管这种系统解决了冲突，并且在一定程度上比纯粹的顺序冲突避免系统的效率更高，但该系统仍然缺乏灵激活。

另一种解决冲突的方法包括在存储器控制器中加入决策能力。遗憾的是，决策逻辑的复杂性要求在控制器能够授权访问共享的存储器之前执行实际决策的所使用的总的时间。

复杂逻辑降低了系统性能这个问题在多芯片模式存储系统中更加严重，系统中具有在各具不同操作特点的互联的多个存储装置中分布的内存。由于传统的逻辑不能用来补偿在不同的存储装置中固有的各个不同的访问特点，因而，整个系统的性能下降。

总的来说，如图 1A 所示，本发明可以根据系统 100 来说明，系统 100 具有请求装置 102，它通过用于向任何数量和类型的共享的资

源 108 提供访问的系统总线 106 连接到通用装置 106 的请求装置控制器 104。在一个实施方案中，系统总线 106 通过相关的系统接口层 110 连接到通用控制器 104，而通用控制器 104 通过共享的资源接口 109 依次连接到共享的资源 108。一般地，根据任何请求系统 102 发出的共享资源请求和共享资源操作的特征参数 113，通用控制器 104 被用来确定共享资源 108 的状态。

在这些情况下：当请求系统 102 是一个多处理器系统中的处理器，请求访问作为存储装置 108 的共享的资源 108，存储装置 108 被连接到它上面的其它处理器所共享，为了完成要求的资源请求，通用控制器 104 确定执行操作的步骤。当存储器装置 108 是，比如 SDRAM，操作一般包括预充电、关闭页、打开页和读页或写页。

一旦确定了操作的具体步骤，通用控制器 104 就确定各步骤之间的适当的时序，为了避免诸如数据冲突或其它类型的冲突。在优选的实施方案中，时序部分地根据存储在如查对表内的共享存储装置的操作特征。之后，适当的排好序的访问指令由通用控制器发布，然后由共享的资源作出回应。

在下面的本发明的详细描述中，为了完全地理解本发明，提供了多个特殊的实施方案。然而，对本领域技术人员来说是很显然的，可以在没有这些具体细节或使用改变的单元或步骤，来实现本发明。另一方面，对已经熟知的处理、步骤、组件和电路，不再详细说明，免得使本发明显得晦涩难懂。

现在，将根据作为处理器与共享资源之间的联络的存储器控制器来描述本发明。但是，应该指出的是，本发明也可以象能够控制访问不论共享与否的任何资源的通用控制器来实现。这些资源不必一定是存储器，实事上，本发明也可以用于控制访问共享的系统总线，比如在多处理器系统中提供交通控制，减少总线访问延时，从而提高系统总线带宽的效率。

现在请参见图 1B，系统 100 具有请求装置 102，如处理器能够通过系统总线 106 连接到通用控制器 104。控制器 104 依次连接到共享的资源 108，如存储器 108 可以采用很多形式，诸如 DRAM、SDRAM、SLDRAM、EDO、FPM、RDRAM 等。在所述的实施方案中，系统总线 106 包括单向地址总线 106－1，用于把处理器 102 发出的存储器地址请求

传送到通用控制器 104。系统总线 106 包括单向指令总线 106－2，它与地址总线 106－1 连接，传输关于存储地址的指令。比如，当处理器 102 请求一个存储在存储器 108 内的特定存储位置的可执行指令时，处理器象指令总线 106－2 输出读请求（称为系统指令），几乎同时，在地址总线 106－1 上得到相应的存储地址请求（称为系统地址）。系统地址和系统指令都被包括在控制器 104 内的可配置的系统接口 110 所接收。可以注意到，意味着通过配置，系统接口 110 可以用于处理接收到的无论是存储器 108 要求的何种形式和方式的系统指令和系统地址。这样，处理器 102 要求的数据就可以存储在任何编号和类型的连接到控制器 104 的存储装置中，无需要求处理器 102 为每个存储装置生成定制的请求。

在所述的实施方案中，系统接口 110 用于把接收到的系统指令和系统地址转换为被叫作通用指令 200 的指令，如图 2A 所示的示例。在一个实现中，当共享的资源是 DRAM 型存储装置（包括 SLDRAM、SDRAM、EDO、DRAM 等），通用指令 200 由 5 个数据字段构成，包括执行存储器 108 的任何存储器访问所要求的所有操作。这些操作包括由预充电数据字段 202 标识的预充电操作，用于表示是特定行否需要预充电。其它操作包括激活数据字段 204、读数据字段 206、写数据字段 208 和刷新数据字段 210。如果，比如说，存储器 208 具有存储体 1 当前激活（即在读或写之后打开）的存储页 1，以及随后的处理器指令要求读出存储在存储体 1 的 2 页上的数据并输出到处理器 102。这时，为了处理器 102 执行请求的指令，页 1 必须关闭（即页 1 被预充电），页 2 必须被激活，页 2 被激活后，读页 2。所以，如图 2B 所示的通用指令 212，由通用指令发生器 110 生成，通用指令发生器 110 具有数据字段 202、204 和 206，设置为“1”表示“执行相关的操作”，而数据字段 208 和 210 设置为“0”表示“不执行相关的操作”（即“NOP”，无操作）。

回过来参见图 1B，由于存储器 108 的访问是动态的这一特征，其中的多个不同的请求装置共同访问存储器 108，存储器 108 的状态是不断改变的。存储器的状态，意思是为在特定的存储位置完整的执行特定的操作，必须已知存储器位置的状态。比如，如果一个特定的存储页关闭了，为了执行读操作，该存储页必须被打开。因此，为了确

定特定地址位置的当前状态，大多数在特定位置已经执行了的当前操作，用资源标志 300 标识，如图 3 所示。在本发明的一个实施方案中，资源标志 300 包括地址字段 302，用于标识特定的存储器位置；包括最近发布的指令字段 304，用于识别在 302 中标识的地址的最近发布的指令；以及包括最近的指令数据字段 306 的时间。比如说，存储器地址 ADDs 的资源标志 308 表示在时间 5Φ（代表 5 个时钟周期）发布了读页面指令，而资源标志 310 表示对同一个存储器地址 ADDs，在时间 10Φ，在 ADDs，在存储页上执行了写页面指令。通过跟踪存储器地址 ADDs 的状态，通用控制器 104 得知在 ADDs 的存储页已经打开，因而不用要求打开页面的操作。

使用存储在资源标志缓冲器 114 内的标志 300 所提供的状态信息，连接到可配置的系统接口 112 的指令定序器 114 提供通用指令 200 的指令项 202－210 之间的适当的时间间隔，以便提供排好序的指令 220，如图 2C 所示，在指令项 202－204 和 204－206 之间分别具有时间间隔 t_1 和 t_2。应该注意到，由于指令项 208 和 210 是“NOP”型字段，排好序的指令 220 包括任何引用这样的字段，只要求时间周期实际上等于项 202 到 206 要求的时钟周期相加的时间 $t_1 + t_2$。这样，指令定序器 114 就能够在处理器 102 和存储器 108 之间提供优化的指令和数据流。

在本发明的另一个实施方案中，当共享的资源 108 是多存储体型存储装置，如 SDRAM，或当共享的资源是多装置存储装置如多芯片模式，比如，资源标志缓冲器 114 能够存储在特定存储体或装置内的所有打开页的资源标志。在一种实现中，比较器（未示出）检测系统地址中的存储体编号或装置标识号，并比较页地址和具有标志缓冲器 114 的内容的系统地址。如果比较没有“命中”（即地址不相匹配），通用控制器 104 就必须关闭使用来自标志缓冲器 114 的地址的旧页面，根据新的系统指令打开新页面。

在通用控制器 104 要服务多个不同的装置的情况下，可以期望能够选择那些只与关于输入系统地址的特定装置相关的操作参数。当通用控制器服务多个不同的装置时，地址空间控制器 120 连接到通用控制器 104，如图 1C 所示。在所述的实施方案中，地址空间控制器 120 提供只选择与输入系统地址相关的装置的特定参数的能力。在一个特

殊的实现中，如图 1D 所示，地址空间控制器 120 包括比较器 122，用于比较输入系统地址和区域地址范围缓冲器 124 的内容，缓冲器 124 标识哪一个装置（或者哪一个存储区域）与输入的系统地址相关。一旦具体的装置或者区域被标识，就选定一组装置参数寄存器 126 和 128（各自连接到范围缓冲器 124 并包含具体装置的装置特殊的参数）其中之一。在有些实施方案中，被选定的装置参数寄存器的内容输入到 LUT118。这样，任何数量的不同的装置可以得到通用控制器 104 的服务，从而，各装置的特定的操作参数被标识，并用于优先级化相应的通用指令的顺序。

还应该指出，有时连接到通用控制器中的一个装置忙，不能接受新指令，能够选择等待在指令队列中的任何其它指令是有益的。在本发明的一些实施方案中，装置的每次响应和通用控制器的每次请求都具有相关的 ID 编号 150,在所述的实施方案中一个长度为 5 位的数据字，如图 1E 所示。ID 编号 150 通过配置为包括长为 2 位的组选择器字段 152 和长为 3 位的请求标号字段 153。组选择器（GS）确定请求属于哪一组特定的系统（即处理器，比方说），而请求标号（RN）表明组选择器字段 152 内所标识的相关的组的响应或者请求的标号，这样来自同一个收发器的连续的请求就具有连续的请求标号。

在一些实施方案中，一组优先级选择器寄存器 154 包括请求或者响应组的各自的优先级值，从而具有更高优先级的请求或响应将代替具有较低优先级值的请求或响应。这样，当较低优先级的请求或响应不能在下一个时钟周期内处理时，具有更高优先级的请求或响应就能够绕过具有较低优先级值的请求或响应。为了防止所谓的活锁，活锁计数器寄存器 156 包含关于具有较高优先级的连续的请求（或响应）的编号，能够绕过具有较低优先级的请求（或响应）。用这样的方法，较低优先级的请求（或响应）就不能急需时钟周期的实际编号。

还应该注意到，为了同时优化指令和数据流控制，应该认识到每个共享的资源都有与之相关的操作特性的设置（如访问时间，比如在是 DRAM 型装置时的 CAS 延时）。当不止一个共享的资源被通用控制器 104 服务时，每个共享的资源具有不同的操作特性的设置，在有些实施方案中，这些特性的设置被存储在连接到指令定序器 116 的查对表（LUT）118 内。指令定序器 116 使用由 LUT118 和存储在资源标志

缓冲器 114 的资源标志提供的信息，适当地排序指令项 202－210，形成排好序的指令 220。当共享的资源实际上是一组存储装置如多芯片模式，其中的每个装置可以具有完全不同地操作特性时，尤其是这样。

现在请参见图 4，按照本发明的实施方案通用控制器访问共享的资源的过程 400 的详细流程图。过程 400 从 402 开始，系统生成访问共享资源的指令。但共享的资源是基于存储装置的 DRAM，该操作包括预充电、刷新、关闭、打开、读和写。比方说，处理器通过生成系统指令（即读页面）和相关的系统地址，请求存储在共享的存储器内的存储页，相关的地址表明被请求的页面被存储的位置。在优选的实施方案中，资源的状态是在 404 确定的，使用，比如与共享的存储器内的激活的存储位置相关的资源标志。接下来，在 406，决定了为执行要求的共享资源的请求所要求的操作的步骤。在 408，通用控制器基于执行所要求的请求的要求的操作步骤，生成通用指令。比如，为了执行读页面操作，先前打开的页面必须关闭，新的页面被激活，然后执行读操作，所有这些操作被理解为在一个单的通用指令结构内。一旦通用控制器构成了通用指令，就使用共享资源的资源标志和特殊的操作特性，然后在 410，通用控制器确定在通用指令的不同项之间的适当的时序。排好序的指令在 412 发布到共享的资源，使用有些实施方案的实际步骤。最后，在 414，共享的资源响应排好序的指令，比如，提供存储在由系统地址表示的位置内的数据。

在本发明的一个实施方案中，通用控制器使用如图 5 所示的过程 500 确定资源的状态（402）和操作的步骤（404）。过程 500 从 502 开始，资源划分标识符（即存储器地址寄存器）与资源标识符（即资源标志地址字段 202）比较。在 504，如果判定“命中”了（即新指令的地址和当前的标志地址字段相匹配），下一条指令（数据操作）就在 506 发布。反之，如果新指令的地址和当前的标志地址字段不相匹配（即未命中），就在 508 判断旧页面是否是打开的。如果旧页面是打开的，就在 510 关闭旧页面，在 512 打开新页面。但是，如果旧页面没有打开，新页面就在 512 被打开，以及无论什么情况，一旦新页面被打开，下一条指令（数据操作）就在 506 发布。

在本发明的一个实施方案中，通用控制器根据如图 6 所示的过程

600 确定操作步骤之间的适当的时序（410）。过程 600 从 602 开始，通用控制器比较指令的新步骤的第一指令和特定资源的最近的上一步骤的最后一条条指令。在 604，通用控制器确定通用指令项之间的时序约束，通过比较新的通用指令的第一指令项和最近的上一条通用指令的最后的指令项。在一个实施方案中，通用控制器使用如表 1 所示的二维数组的形式的 2 索引地址查对表（LUT），其中数组的第一行代表旧的（即最近的上一条）指令，第一列代表新的指令。比如，参见表 1，如果旧指令是读页面以及如果新指令是页面关闭，新指令页面关闭和旧指令读页面的交点（即 5Φ）提供来那个操作之间最少的可以允许的时间量（即最少的实际发布时间）。一般地，存储在 LUT 内的信息是由共享资源制造商提供的。

表 1

		页关闭	页打开	读	写
新指令	页关闭			5φ	
	页打开				
	读				
	写				

一旦确定了特定通用指令项的资源的实际约束，就将在 606 判断是否还有其它指令项包括在通用指令中。如果没有了其它指令项，在 608，就存储通用项及其相关的时序说明。反之，如果还有其它指令项包括在通用指令中，控制就返回到 604，确定该项的相应的物理时序约束。

但是，为了跟踪具有存储体编号的共享的存储器 108 内的物理页的状态，比如说，大量的资源标志要求大量的专用于资源标志的缓冲器 114 的高速缓存器。这将减缓通用控制器 104 的执行，由于它要求大量的时间用于检索存储器的特定页的特定资源标志，每个存储器可能位于分散的位置。参见图 7A，在一个实施方案中，页面命中/错过控制器 702 被包括在通用控制器 104，用于减少页面寄存器 704 的数量 M，在多存储体存储器 706 中 M 比存储体的数量 N 小，因为不是每个存储体条都有它在 M 页寄存器 704 内的表达。在操作中，各 M 页寄存器 704 存储打开页的地址和状态数据，以及随机页寄存器编号发生器 708 生成小于或等于对应页寄存器的 M 的随机整数，被打开页的状

态所代替。比较器 710 并行地比较具有存储体编号的输入系统地址与所有 M 个寄存器的页地址，具有四种可能的结果。

1） 如果比较器 710 表明命中，请求的存储体的要求的页就是打开的，并准备好访问；

2） 如果比较器 710 表明命中存储体，错过页面，通用控制器 104 就必须使用页寄存器的页地址关闭旧页，使用系统地址的页地址打开新页；

3） 如果比较器 710 表明错过了存储体和页，通用控制器 104 就必须关闭存储体的所有页，其编号由随机页号发生器给定，使用系统地址打开新页，以及最后访问请求的存储体；和

4） 错过了存储体和页，但至少有一个页寄存器没有被使用，该页就将被利用以及新页将被打开。

在一些实施方案中，随机页寄存器编号发生器 708 被最近最少使用（LUR）比较器 712 代替，如图 7B 所示，判断 M 个寄存器 704 中的哪一个最长时间没有使用了（即最近最少使用）。

除了跟踪多存储体存储器 704 中物理页的状态，如图 8 所示的存储体访问控制器 800 还包括 N 条寄存器 802，对应包括在多存储体存储器 704 内的存储体 N 的数量。存储体寄存器 802 包括存储体编号字段 804，定义存储体的标识编号，关于存储体的信息被存储。存储体寄存器 802 页包括存储体状态字段，表明由存储体编号字段 804 内的存储体编号识别的特定的存储体的状态。在一个特定的实施方案中，存储体状态字段 806 可以取表 2 中所示的那些值。

表 2

存储体寄存器单元	说明
存储体编号	识别存储在存储体寄存器内的信息

存储体状态	表示存储体的状态: “00” －无效输入 “01” －存储体计数器值递减直到其值为0。如果存储体计数器值大于0，禁止访问该存储体。 “10” －存储体被关闭。 “01” － 存储体计数器值递减直到其值为0。如果存储体计数器值大于0，禁止访问所有存储体
存储体定时器	如果存储体计数器值大于 0，根据存储体状态值，对存储体的访问是被禁止的。

随着高速数据包定位的存储器的发展，如同步链接动态随机存取存储器（SLDRAM），其传输总线数据速度达到 400－800Mb/x/pin，存储器访问冲突带来的问题大大增加了。如图 9A 示出了按照本发明实施方案的基于多处理器系统 900 的一个典型 SLDRAM。多处理器 900 包括通过系统总线 906 连接到控制器 904 的处理器。通用控制器 904 通过 SLDRAM 总线依次连接到同步链接 DRAM（SDRAM）908 和 SLDRAM910，SLDRAM 总线由单向指令总线 912 和双向数据总线 914 构成。应该指出，尽管图 9A 中只示出了两个 SLDRAM，任何数量的 SLDRAM 都可以通过总线 912 和 914 连接到通用控制器 904。在一些情况下，SLDRAMs 可以表现为缓冲模式，包括任何适当的数量的 SLDRAMs，如，该议题下的 SLDRAM908。连接通用控制器 904 和各个 SLDRAMs908 和 910 的初始化/同步（I/S）总线 916，为通用控制器 904 生成的初始信号和同步信号提供信号通道。

在本发明的一个实施方案中，打包的指令、地址和来自通用控制器 904 的控制信息，被选择性地送到指令总线 912 上的 SLDRAM908 和 SLDRAM910。数据总线 914 用于把打包的写指令从通用控制器 904 传输到 SLDRAM908 和 SLDRAM910 中被选定的那个。或者，数据总线 914 也用于把打包的读指令从 SLDRAM908 和 SLDRAM910 中被选定的一个传输到通用控制器 904。应该指出的是，指令总线 912 和数据 914 一般是以同样的速度操作的，即 400MB/s/p、600 MB/s/p、800 MB/s/p，

等等。

多个控制信号由通用控制器 904 产生，并由指令总线 912 传输，这些信号包括，比如，差动自由运行时钟信号（CCLK）、FLAG（标志）信号、指令地址信号 CA、LISTEN（监听）信号、LINKON（链路开）信号和 RESET（复位）信号。一般地，数据包指令由 4 个连续的 10 位字组成，其中指令的第一字由 FLAG 信号的第一位“1”表示。在优选的实施方案中，差动自由运行时钟 CCLK 的两个边都被 SLDRAM908 和 SLDRAM910 使用，用来锁存指令字。通过监视指令总线 912 上的输入信号，SLDRAM908 和 910 响应为 HIGH 的 LISTEN 信号。可以改变地，SLDRAM908 和 910 响应为 LOW 的 LISTEN 信号，通过输入电源备用模式。LINKON 信号和 RESET 信号分别用于关闭电源和上电，获悉被选择的 SLDRAM908 和 910 中的一个的状态，正如所期望的。

对本说明的剩余部分，将只详细讨论 SLDRAM908，但是，认为适当的任何数量的 SLDRAMs 都可以连接到通用控制器 904。如上所述，典型的 SLDRAM 装置，如 SLDRAM908，被分层地排列为存储体、列、行、位以及存储区域。重要的，应该理解到各个可以实际观察到的分层，彼此之间有不同的操作特性。这些操作特性包括但不局限于这些参数：存储器访问时间、芯片调用时间、数据检索时间等。应该指出，多存储体存储器中的各存储体一般具有相同的操作特性，而定义为不同装置的区域如不同的存储器类型或不同的存储器组，各具不同指令和数据延时。比如说，本地存储器组可以直接连接到存储器控制器，而第二非本地存储器组位于一块板上，其中的干预驱动器增加了指令和数据延时，相对本地存储器组来说。在其它情况下，用于构成多芯片模式的不同的存储器芯片可以看作是不同的存储器区域。

更进一步参见图 9A 所示的系统，SLDRAM908 是一个具有 4 个存储器芯片的多芯片模式，芯片 A、B、C、D 各自能够被指令总线 912、数据总线 914、I/S 总线 916 单独访问。由于各个存储器芯片 A-D 可以具有不同的操作特性（一般由制造商提供）为了优化调度指令和数据包，通用控制器 904 能够使用特定分层的和/或存储区域的操作特性。

借助示例，图 9B 示出了具有代表性的时序框图，按照图 9 所示的多处理器系统 99 的典型的 SLDRAM 交易。操作中，处理器通常生成处理器指令包，如读指令 950 和写指令 952，于是 SLDRAM908 的适当

的存储体作出响应。一般地，读指令 950 和写指令 952 是根据生成这些指令地处理器的特定的要求在系统总线 906 上管道传输的，不是为了优化 SLDRAM908 的性能。系统时钟 CLKsys（未示出）提供必要的时序信号。

在本例中，处理器 902a 生成读指令 950，该指令具有位于 SLDRAM 908 的内存芯片 A 的内存地址 MA_1，而处理器 902b 产生一个写指令 952，该指令同样具有位于 SLDRAM 908 的内存芯片 A 的内存地址 MA_2。在本例中，读指令 950 比写指令 952 先输出到系统总线 906。通用控制器 904 先接收读指令 950，接着根据该指令本身和指令地址 MA_1，用存储于通用控制器 904 的目的地址的详细信息执行指令。一旦确定了最小传送时间，通用控制器 904 就生成与接收到的处理器指令 950 相应的 SLDRAM 指令包 READ960，并发布到指令总线 912。

通常，SLDRAM 指令包由四个 10 位字组成，如表 3 所示典型情况，一个 64M SLDRAM，具有 8 个存储体，1024 行地址和 128 列地址。如图示，有 3 位是存储体，10 位是行地址，7 位是列地址。必须指出的是，许多其它结构和密度都是可能的，并且可以在所述的 40 位格式调节，为任何被认为适当的其它任何格式。上电时，通用控制器 904 根据 SLDRAM 对诸如存储单元、行、列及其相关的操作特性等因素的轮询，形成指令包，然后把它存储起来。

指令包的第一个字包含芯片的 ID 位。SLDRAM 将忽略与本地 ID 不匹配的所有指令。芯片的 ID 由通用控制器 904 在通电时用初始化和同步信号确定。这样，通用控制器 904 借助生成单独的芯片触发信号和联系逻辑，唯一寻址多处理器系统 900 中的各 SLDRAM。

表 3

SLDRAM 指令包结构

标志	CA9	CA8	CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0
1	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	CMD 5
0	CMD 4	CMD 3	CMD 2	CMD 1	CMD 0	BNK 2	BNK 1	BNK 0	RW9	RW8

0	ROW 7	ROW 6	ROW 5	ROW 4	ROW 3	ROW 2	ROW 1	ROW 0	0	0
0	0	0	0	COL 6	COL 5	COL 4	COL 3	COL 2	COL 1	COL 0

由于读指令 950 和写指令 952 是管道传输的，通用控制器 904 在接收读指令 950（或者已经存储在缓冲器中）后才接收写指令 952，随后根据写指令 952 传送 SLDRAM 指令包 WRITE 962。通用控制器 904 用 MA_2 的详细特征数据以及读命令 960 的发布时间（即，发布时间）来生成最小发布时间和 WRITE 962 的数据偏置，以避免与先前传送的读指令 960 相干扰，由于两个指令都用到了同一个存储体（A）。

以这种方法，至少根据特定的目标寻址装置的操作特性以及指令和数据包流的当前状态，通用控制器 904 可以动态地调度 SLDRAM 指令包的发布。

现在参照图 10 示出了按照本发明实施方案的的存储控制器 1000 方框图，应当说明的是，存储控制器 1000 仅仅是图 1 中的通用控制器 104 的一种可能的实施方案，因此不能被理解为本发明范围的限制。存储控制器 1000 包含系统接口 1002，它通过系统总线 906 把处理器 902 和存储调度器 1006（被叫做调度器）连接起来。在本发明的一各实施方案中，系统接口 1002 用于将处理器 902 生成的存贮指令包的传输及其相应的写数据包提供给存储器指令包调度器 1004。如果调度器 1006 表明所有内部缓冲器均满，无法容纳新指令，系统接口 1002 将拒绝任何新指令，直到调度器 1006 表示准备号接受新指令。

一个同步链接媒介存取控制器（SLiMAC）1008 提供调度器 1006 和 SLDRAM 908 之间的物理接口。更具体地说，SLiMAC 1008 包含一个指令接口 1010 和一个数据接口 1012，分别通过指令总线 912 和数据总线 914 把 SLiMAC 1008 和 SLDRAM 908 连接起来。在本发明优选的实施方案中，指令接口 1010 按照相关的指令时钟 CCLK，将存储指令从 SLiMAC 1008 传送到 SLDRAM 908。需要说明的是，在一些实施方案中，SLiMAC 1008 结合了一种时钟加倍装置，它用一种接口时钟信号 ICLK（可以达到大约 100MHz）来生成指令时钟信号 CCLK，一般能达到 200MHz。

在本发明的一个实施方案中，数据接口 1012 接受和传送数据都

通过数据总线 914。需要说明的是，数据总线 914 的宽度大得足以支持所需的 SLDRAM。因此，为了提供所需的带宽，SLiMAC 1008 包括足够多的数据接口。例如，如果数据总线 914 是 32 位宽（例如，每个 SLDRAM16 位），那么 SLiMAC 1008 可以包括 2 个数据接口，每个能够处理与 SLDRAM 相关的 16 位。按这种方法，SLiMAC 1008 包括的数据接口的数目可以与连接的 SLDRAM 的特定配置很好地相匹配。

与指令接口 1010 几乎相同的方式，SLiMAC 1008 可以提供一个数据时钟信号 DCLK，同时读数据从 SLDRAM 908 传送到 SLiMAC 1008。在本发明的一种实施方案中，数据时钟 DCLK 是通过用时钟加倍装置把接口时钟 ICLK 的频率从大约 100MHz 加倍到大约 1000MHz 来产生。需要指出的是，接口时钟信号 ICLK，指令时钟信号 CCLK，和数据时钟信号 DCLK 全部是同相的。

在本发明的一个优选实施方案中，调度器 1006 包含了一个约束程序块 1016，用来从连接到其上的系统接口 1002 接收系统指令及其相关的系统地址数据。约束程序块 1016 向重排序块 1018 提供 SLDRAM 指令包数据及其相关的时序信息。写缓冲器 1020 从系统接口 1002 接收写数据。由于受调度器 1006 控制，读数据通过读缓冲器 1022 从数据接口 1012 传输，该缓冲器与数据总线相连，用来向系统接口 1002 提供读数据。与初始化/同步化（I/S）总线 916 相连的 I/S 程序块 1024 按需要向 SLDRAM 908 提供合适的初始化和/或同步化信号。

操作中，调度器 1006 接收由处理器 902 产生的管道传输存储指令包。一般地，存储指令包由存储指令及其相关的存储地址组成。在本发明的一个实施方案中，调度器 1006 把与接收到的新指令相关的存储地址译码，以便决定存储指令和相关数据包（如果有的话）所指的目标地址。一旦译码，调度器 1006 用所存储的目标地址特定装置的特征数据以及与就在先前发布的存储指令相关的信息，来传送一个新的 SLDRAM 指令包。这个新的 SLDRAM 指令包输出到指令总线 912，并且最终送到由 SLDRAM 指令包中包含的芯片 ID 所指定的 SLDRAM。

作为调度过程的一部分，调度器 1006 确定在前一个发布的指令后，发布新指令前所需的最少时间。由于，如上所述，分级标准，比如 SLDRAM 的一个存储单元，可以有不同的操作特性（通常由制造商提供），调度器 1006 轮询各个在初始化中运行的 SLDRAM。在一些实

施方案中，如果为了确定操作参数，相连的存储器件不允许轮询，那么存储详细参数（比如时序）可以直接写入约束程序块寄存器 1016。一旦轮询了 SLDRAM，调度器 1006 就将存储装置详细信息，以后将用来改进适当的调度协议。以这种方法，调度器 1006 能够向任何数量和类型的 SLDRAM 恰当地提供调度服务，而无须借助硬件或其它费时而昂贵的步骤。

图 11 的示意图中示出了按照本发明一个实施方案的约束程序块 1100。应该理解，约束程序块 1100 不过是如图 10 所示的约束程序块 1016 一个可能的实施方案，不能看作是限制。约束程序块 1100 包括连接到系统接口 1002 的地址译码器 1102，用于译码接收到的新的与处理器 902 生成的新存储器指令相关的地址信号。译码后的新地址信号输入到序列标志寄存器 1104，寄存器 1104 中存储关于 SLDRAM 存储体的状态和相关信息，尽管有时只是存储这些信息的一部分。序列标志寄存器 1104 输入到选择器 1106，根据译码后的新指令地址通过选择的虚拟存储体的相关的数据，送到查对表（LUT）1108。

约束程序块 1100 还包括页连接到系统接口 1102 的区域比较器 1110，用于接收新的地址信号提供区域标识符表示新指令地址所在的存储器区域。这样，约束程序块 1100 就能够，至少根据部分的存储区域的特定特性数据，为新的存储指令提供最佳的调度协议。区域比较器 1110 在输入新指令信号时，向 LUT1108 提供区域标识符。LUT1108 依次提供最小的增量发布时间和数据偏置，用于将新指令及其相关的新地址转换为 SLDRAM 指令包。应该注意到，最小的增量发布时间是指发布新指令相对于只发布旧指令的增量时间（在时钟周期内）。数据偏置时间是表示，为了在发布新指令后，接收关于新指令的读数据包，在时钟周期内的增量时间。

在本发明的一个实施方案中，约束程序块 1100 包括 16 个序列标志存储体寄存器，LUT1108 能够存储 4 个时序区域的 4 个不同的参数设置，各具 16 个相关的寄存器。

图 12 是 SLDRAM 总线信号的时序图 1200，按照本发明实施方案的接收到的处理器指令的响应。应该指出，表 4 汇总了由约束程序块 1100 识别不同的生成的信号执行的调度器。还应该注意到，存储器指令的形式为{指令，地址}，其中“指令”代表要执行的指令，“地址”

是相应的存储位置。

参见表 4 和图 12，在系统时钟周期Φ_1期间，第一指令{OPENPAGE, 1000}在地址译码器 302 和区域比较器 1110 同时接收到。对该示例，地址译码器 1102 把指令 OPENPAGE 的地址“1000”译码为“100”和“400”，区域比较器 1110 判断该地址被包括在存储器区域 0 之内。由于指令 OPENPAGE 是接收到的第一条指令，没有“命中”任何虚拟的存储体 B_0-13，与之对应的替代计数器设定为“0”。在所述的实施方案中，替代计数器根据伪随机计数方案更新，而在其它实施方案中，可以使用随机计数或其它适当的方案。由于第一指令{OPENPAGE, 1000}是开放型指令，没有相应的最小增量发布时间或数据偏置，因此，地址 1000 的页面在第一指令时钟周期ΦC_1是打开的。

在下一个系统时钟周期Φ_2，指令{READ, 1000}在约束程序块 1100 被接收到，地址译码器 1102 将其译码为 100 和 400（即，从前一个时钟周期开始读取在存储地址 1000 打开的页），又使区域比较器 1110 设定区域标识符为 REGION1。然而，这种情况下，已经存储在 B_0 寄存器内的先前的或其它被称作“旧指令”的指令导致在 B_0“命中”，使选择器输出“READ”作为“旧指令”输入到 LUT1108。其它输入包括区域比较器 1104 生成的区域标识符 REGION1 和“新指令”输入如 READ。LUT1108 利用存储的特征数据生成 3 个指令周期Φ_3的最小的增量发布时间，表明至少 3 个指令时钟周期必须分开指令{OPENPAGE, 1000}和指令{READ, 1000}的发布。

这样，在约束程序块 1100 接收到的各个存储指令包，根据存储在 LUT1108 的特征数据和至少部分根据紧邻的先前发布的指令。

现在将描述按照本发明的具体实施方案的从约束块接收到的指令的重排序。图 13A-13C 是时间线 1302 和 1304，通过简单的指令重排序示例，解释一些借助重排序存储指令可以实现的优点，按照本发明的具体的实施方案。每条时间线示出了对应两个不同的存储体的 4 条读指令。CMD0 和 CMD1 是针对相关存储器的存储体 1 的读指令。CMD2 和 CMD3 是针对相关存储器的存储体 2 的读指令。时间线 1302 表示存储指令排列在指令总线上，依次连接存储器控制器和存储器，其中的指令是存储器控制器从系统处理器中接收的；CMD0 占据时隙 0，CMD1 占据时隙 3，CMD2 占据时隙 4，CMD3 占据时隙 7。每个时隙代表一个

时钟周期。

如上所述，给同一个存储体的指令，在先前发布的指令的发布和调节服务之间必须具有一些小的延时。图 13A 中每对指令之间由两个时隙表明了这一点。如图示，如果 4 条读指令按照图 13A 所示的顺序发送到存储器，指令总线将在可以利用的 4 个时钟周期没有被使用，即时隙 1、2、5 和 6。正如将要讨论的，根据本发明，通过重排序指令至少可以部分改善这种低效率现象。

图 13B 和图 13C 的时间线 1304 和 1306，分别说明了根据本发明具体实施方案的图 13A 的指令重排序和由此获得的至少部分优点。示例中，为了简化起见，没有考虑数据总线上的冲突。但是，如下所述，必须注意考虑存储指令的有效重排序。由于 CMD2 和 CMD3 与 CMD0 和 CMD1 是针对不同的存储体，两对指令之间的存储访问延时是不相关的并可以忽略。也就是说，指令必须象时间线 1304 所示的那样重新排列，CMD2 排在紧接着 CMD0 之后的时隙 1，CMD3 排在紧接着 CMD1 之后的时隙 4。这是因为在发布 CMD0 和 CMD2 或者 CMD1 和 CMD3 之间不需要任何延时，由于他们针对不同的存储体。但是，如图 13C 所示可以理解到的，最小的延时时间，如两个时钟周期，针对同一个存储体的一对指令之间的延时必须保留。也就是说，指令的重排序并不包括试图减少对同一存储体的连续的指令之间的延时。

重排序指令的结果如图 13C 所示，其中，4 条指令在 5 个时钟周期内发布，只有两个时隙没有利用。当然，将可以理解到，给另一个存储体的第五条存储指令可以插入到时隙 2，从而进一步提高指令总线的利用率。

图 14 为根据本发明具体实施方案设计的存储器控制器的部分方框图。重排序电路 1400 从系统处理器接收到输入的存储指令序列，即 1、2、3。按照具体的实施方案，存储指令通过约束电路（未示出）传输到重排序电路 1400，如前所述，约束电路施加发布时间限制到选择的指令相对于给相关存储器的同一逻辑存储体的其它指令。指令被重排序在指令队列 1402 中，由此把指令发布到存储器。本例中，指令被重排序到序列 1、3、2 中。

原来的存储指令序列，即 1、2、3，被存储在读数据电路 1406 中的先进先出（FIFO）存储器 1404。在 FIFO1404 中的序列用于重排

序从存储器接受到的数据，按照存储器控制器原来接受到的指令的顺序。但是，应该说明，有些处理器期望有序的数据，而其它处理器期望无序的数据，因此，通过把 FIFO1404 切换为要求的开或者关，就可以支持任何类型的数据顺序。这是必要的，因为处理器“期望”接收到的数据的顺序和原来传输到存储器控制器的指令的顺序相符。

另外，因为来自存储器的数据可能被存储器控制器接收到的顺序与处理器传输存储质量的原来的顺序不相符，第三序列就存储在数据队列 1408。该序列（本示例中是 3、2、1）代表符合指令序列 1、3、2 的数据，将被读数据电路 1406 接收的顺序。根据指令队列序列和已知的关于存储器的不同的逻辑存储体的延时，重排序电路 1400 计算数据队列序列。当存储器将数据按照存储在数据队列 1408（即 3、2、1）中的序列传输到存储器控制器时，数据被存储在读数据缓冲器 1410，并根据 FIFO1404 和数据队列 1408 的信息重排序，这样，数据就按照原来的指令序列顺序即 1、2、3，传输到处理器。

图 15 为根据本发明的一个具体实施方案设计的那场控制器中的重排序电路 1500 的方框图。重排序电路包括指令队列 1502，它存储和重排序从系统处理器接收到的指令。指令队列 1502 计算每条指令的发布时间、发布指令和从队列中删除指令，使用关于存储器中同一逻辑存储体的指令发布时间限制以及数据总线利用限制。

数据队列 1504 存储数据项表示对应发布的存储指令的的数据出现时间，计算队列中每个新输入的数据出现时间，并在相应的存储处理结束时删除队列输入。

比较器阵列 1506 执行冲突检测功能，其中指令准备从指令队列 1502（如图通过多路复用器 1508 传送的）发布的数据出现时间与在数据队列 1504 中提交的先前发布的指令的数据出现时间比较。如果检测到了冲突，就延时发布指令。

图 16 是图 15 所示的重排序电路 1500 的更详细的方框图。指令队列 1502 包括 6 个指令队列项 1602，每个单元存储 61 位关于特定存储指令的信息，如图 17 所示。指令字段 1702 包含 40 位指令包，详细说明存储指令。指令发布时间（C_d）字段 1704 为 6 位，表示指令可能被发布前的时钟周期内的增量时间。在 1704 内的值由如上所述的限制电路确定，并涉及到对应存储器内同一逻辑存储体的最近的存

储指令。也就是说，在 C_d 字段内的值表示到同一存储体地两条指令之间的延时。关于每个存储体的要求的延时的信息存储在限制电路内并主要由存储器的物理特性确定。只要在指令队列中，C_d 字段就对每个具有一些例外的时钟周期递减一次。比如，给同一逻辑存储体的连续的指令之间的延时是不能改变的。因此，如果针对特定存储体的指令的 C_d 字段达到 0 并没有发布，到同一存储体的其它指令的 C_d 字段也不能确定，直到第一指令被发布。

数据出现时间（D_d）字段 1706 为 6 位，表示存储指令从指令队列出现的时间到相应的数据传输之间的时钟周期的增量时间。D_d 字段 1706 可能在指令队列中不能改变。指令 ID 字段 1708 是 5 位的，唯一的识别指令包 1702 中的指令。该信息具有 FIFO 和数据队列中的相应的信息，用于跟踪哪一个信息包是哪一个，以及哪些数据对应哪些信息包，从而使指令和数据的重排序有效。逻辑存储体(B)字段 1710 使 3 位的，表示指令包是针对存储器内的哪一个逻辑存储体。最后，字符组指示器(D_b)字段 1712 是 1 位的，表示数据是否被请求或写，占据一个或两个时钟周期。

回过来参见图 16，指令队列的操作由指令队列控制器 1604 来控制。控制器 1604 跟踪可以利用的队列项 1602，控制输入指令插入到特定的队列 1602，通过自由位置指示器 1606。控制器 1604 也便于将指令队列项信息插入到数据队列 1504，一旦对应的指令已经发布。根据具体的实施方案，指令插入到队列 1502，无须考虑在指令或数据总线上的自由时隙的可使用性。

指令可以通过多路复用器 1608 从指令队列项 1602 中的任何一个发布到指令总线上，如果 Cd 的计数为 0 并且数据总线上没有表示有冲突。也就是说，必须识别指令总线和/或数据总线上的空时隙。如果指令没有读或写指令（因此不要求数据总线资源），就只需要指令总线时隙。如果指令是读或者写，在指令总线和数据总线上的时隙都需要。

控制器 1604 内的 0 比较器 1610 用于作出第一判断，即 Cd = 0。减法器 1612 用于，每个时钟周期，从每个指令队列项 1602 的 Cd 计数中减去“1”，除非出现如上所述的例外，即对没有发布的特定指令 Cd = 0。在队列控制器 1604 使用所有队列项的 Cd 和 B 字段的情形

下，生成一个屏蔽信号（M），防止发给同一逻辑存储体的所有指令的 Cd 计数被减少。

根据具体的实施方案，如果有两个队列项的 Cd = 0，就发布具有最高优先级的那一个（如最就的那个）。地址移位器 1614 确定队列中的指令的优先级，下面将结合附图 18 更详细地说明。按照另一个具体的实施方案，如果新指令到达指令队列时其 Cd 计数已经为 0，该指令可以通过多路复用器 1608 直接传输到存储器。如果 Cd 计数为非 0 或者有其它 Cd = 0 且优先级更高的指令存储在指令队列，新指令就被存储在指令队列项 1602 内。但是，如果指令队列为空，新指令就可以被立即发布（如果 Cd 等于 0）.

对读或写指令，使用包含准备要发布的指令的指令队列项 1602 的 D_d 和 D_b 字段，检测冲突。相应指令的出现时间和宽度通过多路复用器 1508 传送到比较器阵列 1506，多路复用器 1508 由队列控制器 1604 轮流控制。也就是说，队列控制器 1604 控制多路复用器 1508 传输队列项中指令发布时间即 Cd 为 0 的指令的数据出现时间和宽度（一个或者两个时钟周期）。宽度表示为一个或者两时钟周期，用加法器 1616 将 D_b 位加上数据出现时间 D_d，得到 $D_d + 1$ 为“0”（表示一个时钟周期）或者为“1”（表示两个时钟周期）。然后，数据出现时间和宽度就在比较器阵列 1506 中与存储在数据队列 1504 中的先前发布的 5 条指令的数据出现时间和宽度相想比较。按照具体的实施方案，比较器阵列 1506 包括 2 × 10 个并行的比较器阵列。

图 18 是图 16 所示的地址移位器 1614 的具体实施方案的方框图。如上所述，新指令按照空位置指示器 1606 被插入到任何自由指令队列项 1602。新指令要插入的指令队列项 1602 的地址被插入到具有最高优先级的第一个空位置（A0-A5）。结果是，地址移位器 1614 内的位置 A0 存储还没有发布的旧指令的队列项地址。当指令从指令队列发布，地址移位器 1614 内的相应项目旧被删除，并且较低优先级指令的地址移位到较高的优先级位置。如上所述，当指令队列中的指令的 Cd 计数达到 0，该指令可以被发布。但是，如果不止一条指令的 Cd = 0，最旧的指令，即具有最高优先级的指令，正如地址移位器 1614 内的其地址的位置所表明的，就被发布。

图 16 中的数据队列 1504 包括 5 个队列项 1652，各单元存储关于

先前发布的存储指令的 12 位信息，如图 19 所示。数据出现时间（D_d）字段 1902 是 6 位的，表示在指令队列发布指令到接收到相应的数据之间的时钟周期内的增量时间。每个指令队列项 1652 的 D_d 计数被每个时钟减，使用其中一个减法器 1564，直到其值为 0。当 $D_d = 0$ 时，相应的数据在数据总线上。因此，应该理解到，在任何给定时间，只有一个数据队列项 1652 的 $D_d = 0$。D_d 计数达到 0 后，将对应的数据队列中的信息从数据队列 1504 中删除。

指令 ID 字段 1904 是 5 位的，它唯一的标识发布的指令对应的数据。该信息对于按照指令原来传输到存储器控制器的顺序重排序数据是很有用的。最后，字符组指示器（D_b）字段 1906 是 1 位的，表明数据是否占据了一个或者两个时钟周期。

回过去参见图 16，如上所述，每个数据队列项 1652 的数据出现时间（D_d）和宽度在比较器阵列 1506 中与准备发布的，即 Cd = 0 的指令队列 1502 中的指令的 D_d 和宽度比较。宽度被表示为一个或者两个时钟周期，通过用加法器 1656 把 D_b 位与数据出现时间相加，加法器向 $D_d + 1$ 提供“0”（表示一个时钟周期）或者“1”（表示两个时钟周期）。如果比较的结果表明没有冲突，就从指令队列发布指令。

数据队列控制器 1658 控制数据队列 1504 的操作。空位置指示器 1660 和指令队列控制器 1604 一起，便于将新的数据队列项信息插入到数据队列项 1652。空位置指示器也方便于从数据队列项 1652 删除信息，当相应的存储器访问已经结束时。0 比较器 1662 和字符组指示器 1664 用于判定所有数据队列项 1652 的 D_d 何时为 0 和数据传输何时不再占据数据总线，以及因而相应的信息什么时候可以从数据队列中删除。

根据本发明的另一个具体实施方案，冲突检测变得更复杂，尽管使用了二维的比较器和多路复用器数组。这种方法比前述的一维方法更加深入细致，它查看了指令队列中的所有单元，而不仅仅查看准备被发布的指令。其调度指令时不仅考虑先前发布的指令，还考虑数据总线上数据包的顺序。

为了插入新指令，必须比较指令管道的准备发布部分内的每组两个连续的阶段，看是否能够将新指令插入其中。比较实际是确定指令可以被插入的范围。其范围如下：

$CLEN_x$ = 指令长度

$T_{cstart}=t_{cA}+CLEN_A$ (1)

$T_{cend}=t_{cB}$ (2)

其中的 t_{cA}和 t_{cA}是连续的管道单元 A 和 B 的发布时间。管道单元 A 在管道单元 B 之前，因此其发布时间是两个发布时间中中较小的。如果要插入，在单元 A 和 B 之间至少必须有一个打开的时隙。因此:

$N= T_{cend}- T_{cstart}+1$ (3)

(其中，N=单元 A 和 B 之间发布时隙的数量)

$LEN<=t_{cb} - t_{ca} - CLEN_A$ (4)

在硬件中很容易简化实现这样的情况:

$(t_{cB} - CLEN_A) - (t_{cA}+CLEN_A) => 0$ (5)

范围的起点和终点还规定了相关数据时隙的可能的范围。该范围必须与数据管道内的各组连续的单元比较，看是否有重叠，以及新的范围是什么。该比较存在 5 中不同的情况。

情况 0

在这种情况下，由数据时隙 t_{dA}和 t_{dB}描述的范围完全在两个连续的项 M 和 N 之外。这时:

$t_{dA}+CLEN_A => t_{dN}$ (6)

或者，其中的 DLENX=数据长度，

$t_{dB}<= t_{dM}+DLEN_M$ (7)

在 M 和 N 之间不可能有数据时隙。

情况 1

在这种情况下，由数据时隙 t_{dA}和 t_{dB}描述的范围完全在两个连续的项 M 和 N 之内。这时:

$t_{dA}+CLEN_A => t_{dM}+DLEN_M$ (8)

和

$t_{dB} - CLEN+DLEN<= t_{dN}$ (其中 CLEN 是新指令长度，DLEN 是时隙内的新数据长度) (9)

在这种情况下的可能的最早的数据时隙是 $t_{dA}+LEN_A$，具有对应的指令发布时间 $t_{cA}+CLEN_A$

情况 2

在这种情况下，由数据时隙 t_{dA}和 t_{dB}描述的范围跨越了项 M。这

时:

$t_{dA}+CLEN_A<t_{dM}+DLEN_M$ （10）

和

$t_{dB}-CLEN+DLEN>t_{dM}+DLEN_M$ 和 $t_{dB}-CLEN+DLEN<t_{dM}$ （11）

在这种情况下的可能的最早的数据时隙是 $t_{dM}+DLEN_M+1$，具有对应的指令发布时间 $t_{dM}+CLEN_M-DATA\_OFFSET$，其中 DATA_OFFSET 是指令发布时间和数据占用之间的时间。

情况 3

在这种情况下，由数据时隙 t_{dA} 和 t_{dB} 描述的范围跨越了项 N。这时:

$t_{dA}+CLEN_A>t_{dM}+DLEN_M$ （12）

和

$t_{dA}+CLEN_A+DLEN<t_{dN}$ （13）

因此，在这种情况下的可能的最早的数据时隙是 $t_{dA}+CLEN_M$，具有对应的指令发布时间 $t_{cA}+CLEN_A+1$。应该指出，情况 1 也被包括在这种情况内。

情况 4

在这种情况下，由数据时隙 t_{dA} 和 t_{dB} 描述的范围，包括在由项 M 和项 N 定义的范围内。这时:

$t_{dA}+CLEN_A<t_{dM}+DLEN_M$ （14）

和

$tdB-LEN>Ct_{dN}$ （15）

因此，在这种情况下的可能的最早的数据时隙是 $t_{dM}+DLEN_M$，具有对应的指令发布时间 $t_{cM}+CLEN_A+DATA\_OFFSET$，其中 $DATA\_OFFSET=t_{dA}-t_{Ca}$。

很显然，作为调度目的情况 1 和情况 3 是一样的，因为总是使用可能的最早时隙。所以组合的情况就是情况 3。类似的情况 2 和情况 4 是一样的，由于期望的结果是 $t_{dM}+LEN_M$。在这种情况下，必须简化地表示 t_{dM} 被由 t_{dA} 和 t_{dB} 给出地范围所跨越。输入指令的附加的可能的最早发布时间（tc）和数据时隙（td）必须被考虑。必须比较每对指令管道的每对数据管道:

```
if(((t_cB – CLEN) => (t_cA +CLEN_A)) && (t_c <= (t_cA + CLEN_A))){
        if(((t_dA + CLEN_A) <= (t_dM + DLEN_M)) && ((t_dB – DLEN – (t_dM + DLEN_M)) >=
0)){
                t_d = t_dM + DLEN_M;
                t_c = t_cA – t_dA + t_dM + DLEN_M;
        }
        else if(((t_dN - (t_dA + CLEN_A + DLEN_A)) >= 0) && (t_dA + CLEN_A) >= (t_dM +
        DLEN_M)){
                t_d = t_dA + CLEN_A;
                t_c = t_cA + CLEN_A;
        }
        else {
                t_d = IMPOSSIBLE;
                t_c = IMPOSSIBLE;
        }
}
else if(((t_cB – CLEN) => t_c) && (t_c > (t_cA + CLEN_A))){
        if((t_d < (t_dM + DLEN_M)) && ((t_dB – DLEN – (t_dM + DLEN_M)) >= 0)){
                t_d = t_dM + DLEN_M;
                t_c = t_c – t_d + t_dM + DLEN_M;
        }
        else if(((t_dN - (t_d + DLEN)) >= 0) && t_d >= (t_dM + DLEN_M)){
                t_d = t_d;
                t_c = t_c;
        }
        else {
                t_d = IMPOSSIBLE;
                t_c = IMPOSSIBLE;
        }
}
else {


        t_d = IMPOSSIBLE;
        t_c = IMPOSSIBLE;
}
```

因此，对指令管道，需要的操作是：

$t_{cB} - CLEN => t_{cA} + CLEN_A$
$t_{cB} - CLEN => t_c$
$t_c + CLEN <= t_{cB}$
$t_c > t_{cA} + CLEN_A$
$t_c <= t_{cA} + CLEN_A$

而对数据管道需要的操作是：

$t_{dA} + CLEN_A <= t_{dM} + DLEN_M$
$t_{dA} + CLEN_A >= t_{dM} + DLEN_M$
$t_{dB} - DLEN >= t_{dM} + DLEN_M$
$t_{dN} >= t_{dA} + CLEN_A + DLEN_A$
$t_d < t_{dM} + DLEN_M$
$t_{dN} >= t_d + DLEN$
$t_d >= t_{dM} + DLEN_M$

因此，判断逻辑由上面所定义的比较器单元的阵列组成。优化的选择是最早的指令发布时间，这由简单的优先级编码来确定。

重定序器管道控制逻辑必须动态地判断指令和数据管道的各单元要进行什么操作。

在等待的指令管道内，每个管道单元具有 4 各可能的操作：从先前的管道（管道接近）读取，保持当前管道（管道保持），从下一个单元读取（管道备份），和从输入指令总线读取。正如前面所定义 4 种情况在管道内的不同的点存在多种设定。同步链接媒介存取控制器（SLiMAC）所发布的单元被定义为项 0，而发布的最快的单元被定义为项 M.重定序器判断逻辑发现在当前管道中的最佳插入点是在项 N-1 和 N 之间，就在项 N 插入。

情况 1－保持

管道保持就是不发布到同步链接媒介存取控制器或者没有插入新的指令。

情况 2－保持&插入

这种情况就是不发布到同步链接媒介存取控制器（SLiMAC），但要在管道中插入新的指令。如果是在项 N 插入，管道将从项 0 到项 N－1 保持，在项 N 插入，并从项 N+1 到项 M 备份。

情况 3－发布

这种情况是从单元 0 发布到 SLiMAC，管道的复位将提前，因而单元 0 将包含单元 1 的内容，单元 1 将包含单元 2 的内容，以此类推，项 M-1 将包含项 M 的内容。

情况 4－发布&插入

这种情况是从单元 0 发布到 SLiMAC 并在项 N 插入。这时，单元 0 到项 N-2 进行提前操作，项 N-1 进行插入操作，项 N 到项 M 保持。对单元进行提前操作就是存储来自该单元后面项的数据，在项 N 插入（在当前管道的项 N-1 和项 N 之间插入项）实际上意味着插入的项将在更新的管道的位置 N-1 结束。

图 20 示出了一个冲突检测系统 2000，它是图 15 所示的冲突检测系统的另一种实现方案。在这个实施方案中，冲突检测系统 2000 重排序指令，以便达到最佳的指令顺序，基于目标响应约束和在启动程序控制器和目标子系统之间确定数据传输的最佳时隙。因为指令的重排序不会导致总线上不同数据包的冲突，如果一指令数据传输涉及到特定指令将导致要求的数据冲突，插入检测器 2002 就禁止发布该特

定指令的。在所述的实施方案中，冲突检测系统 2000 包括连接到指令队列 2004 的冲突检测器 2002。

在所述的实施方案中，冲突检测器 2002 检测“将要发布的”指令（存储在指令队列 2004 中的指令）和“已经发布的”指令（存储在数据队列 2006）中所有的可能的数据冲突。在所述的实施方案中，有 N 个指令队列 2004，各自连接到多路复用器 2008。N 个指令队列 2004 中的每一个通过排列用来存储这些将要发布的指令，时间因素“d_time_{ND}”，表明指令被发布到目标装置后，什么时候数据传输将出现在通用控制器 104 与目标装置（即共享的资源）108 之间的数据总线上；字符组字节（b_{ND}）表明数据字符组传输；和读/写字节（rw_{ND}）。在所述的实施方案中，数据队列 2006 存储时间因素“d_time_{D}”表明对已经发布到目标装置的请求，什么时候数据传输将出现在通用控制器 104 与目标装置 108 之间的数据总线上。指令队列 2006 还存储字符组字节（b_{ND}）和读/写字节（rw_{ND}）。

在优选的实施方案中，冲突检测系统 2000 包括队列和链接控制器单元 2010，用于存储和重排序那些将要发布的指令。队列和控制器单元 2010 还计算新的指令发布时间和数据出现在数据总线上的时间。队列和控制器单元 2010 也从指令队列将发布了的项传输到数据队列，以及当指令被发布后系哦那个指令队列中将其删除。当对存储器的访问结束后，队列和控制器单元 2010 也从数据队列中将数据项删除。

参见图 21，每个对目标装置的读/写指令都有关于它的数据包传输，指令被发布到目标装置前，根据其时序信息检查新数据包 ND（新数据），看能否插入到数据队列中而没有冲突。如图 21 所示的示例中，一个发布了的数据包 D 已经被放在数据队列中，一个新的数据包 ND 与发布了的数据包 D 相比较。应该指出，关于发布了的数据包 D 和新的数据包 ND 都呈现字符组访问。因此，在本示例中，在不导致冲突的前提下，新的数据包 ND 如何插入到发布了的数据包 D，有两种可能性。新的数据包 ND 可以放在发布了的数据包 D 的右侧或者左侧。

这个特定的示例说明了存储器控制器的冲突检测同时支持非字符组和字符组数据传输（即，4 数据流）。由于数据总线的的双向传输特性，一个时钟周期必须插入到连续的读－写或者写－读传输之

间。

应该注意到，这里有很多可能的结果，现其中的一部分罗列如下。

1） 如果 ND 放在 D 之前或之后，在 D 和 ND 之间不存储冲突。

2） 在连续的读－写或者写－读数据传输之间，插入了一个时钟周期。每个指令单元和数据队列存储“rw”位，表明操作是否是“读数据”（rw＝0）或者“写数据”（rw＝1）。

3） 数据包由一个数据流(没有字符组传输)或者 4 个数据流(字符组传输）。每个指令单元和数据队列存储“burst”位，表明操作是否是“字符组传输”（burst＝1）或者“没有字符组传输”（burst＝0）。

对每条将要发布的指令，必须比较要发布的数据包和发布了的数据包对：

```
//初始化变量
collision = NO;
//根据字符但位确定来自指令队列的新数据色的结束
if    (burst_ND = 1) then d_time_end_ND = d_time_ND + 3
else                      d_time_end_ND = d_time_ND

for i=1 to last_element_from_Data_Queue
begin
//根据字符组位确定来自数据队列的数据色的结束
if    (burst_D[i]  = 1) then d_time_end_D[i]    = d_time_D[i] + 3
else                         d_time_end_D[i]    = d_time_D[i]
//在一个周期内必须执行的两个连续的读/写式写/读指令之间
if    (rw_D[i] = rw_ND) then
begin
        d_time_end_D[i]    = d_time_end_D[i]    + 1
        d_time_end_ND      = d_time_end_ND      + 1
end
//冲突检测
      if   NOT((d_time_ND > d_time_end_D[i]) or (d_time_D[i] >
d_time_end_ND))  collision = YES;
        end.
```

在本发明的其它实施方案中，公开了一种预测两个连续的存储器访问之间的时间的装置和方法，允许很快的计算新指令的最早的“指令发布时间”。请看图 22，它示出了一个预测器系统 2200，该系统

具有 N 页计时器 2202，存储上一次发布指令到特定页与预定下一次访问该内存之间的时间。下一次访问同一页可以是“关闭”、“打开”、“写”或“读”。输入的新指令（比如，读）选择一特定的页的表明特定页访问在发布之前要等待多长时间的计时器。之后，同一条新指令就选择适当的时序查对表 2204 的内容，必须插入到该指令（读）与可能的下一条访问同一页的指令（关闭、打开、写或读）之间。计时器的分辨率为一个时钟周期。

时序查对表－数据存储时间，表明指令发布后多少个周期，数据总线上的数据将有效。如果新指令是非激活的，每个周期所有页计数器的值直到其值达到“0”。

现在请参见图 23，在本发明的另一个实施方案中示出了具有根据本发明一个实施方案的装置访问优先级区分器 2302 的装置控制器 2300。在所述的实施方案中，优先级区分器 2302 包括请求队列 2303 适合接收和存储任何数量的装置请求，连接到一个请求控制单元 2304，部分用于从请求队列 2303 的任何位置取特定的响应，并把所取到的响应传输到多个共享装置 108 中适当的那一个。在所述的实施方案中，优先级区分器 2302 还包括响应队列 2306，用来接收和存储连接到响应控制器单元 2308 的共享的任何装置 108 的响应，响应控制器单元 2308 用于选择发送到请求的装置 102 的特定的存储的响应。

在优选的实施方案中，每个响应和请求具有与之相关的 ID 号 150，如图 1E 所示，这样，每个请求及其相关的响应具有相同的 ID 号 150。如前所述，ID 号 150 包括 5 个数据位，其中的第一和第二数据位是组选择器字段 152，识别特定的响应/请求所属的请求装置的组（如在多处理器计算机环境中的处理器组）。还是如上所述，请求号字段（RN）153 提供由组选择器字段 152 识别的相关的请求的装置的请求和/或响应的编号，这样，来自同一请求装置的连续的请求，比如，具有连续的请求编号字段 153。

在操作中，请求以及响应控制器 2304 和 2308 都分别结合组优先级选择器寄存器 154、活锁计数器寄存器 156 和重排序寄存器 2312 结合。组优先级选择器寄存器 154 包括通过 RN152 识别的特定的请求/响应组的优先级信息，在一个实施方案中，值为“3”表示最高的优

先级，值为 0 表示最低的优先级，这样具有较高优先级的请求就能绕过具有较低优先级的请求。

为避免活锁情况，活锁计数器寄存器 156 包含关于多少具有较高优先级的连续的请求（响应）能够绕过具有较低优先级的请求（响应）的信息。应该指出，活锁计数器寄存器 156 只有在具有较高优先级的请求绕过具有较低优先级的请求时，才是激活的。实际上，如果在适当的队列中没有具有较低优先级的请求（响应），活锁计数器寄存器 156 就为非激活的。

尽管只是详细地说明了本发明的很少的几个实施方案，但我们应该能够理解到，在不背离本发明的实质和超出本发明范围的前提下，本发明可以用许多具体的形式实施。因此，所给出的示例是解释性的，而不应该看作是限制；本发明也不局限于在这里给出的细节，却可以在附后的权利要求的范围内改变。

说 明 书 附 图

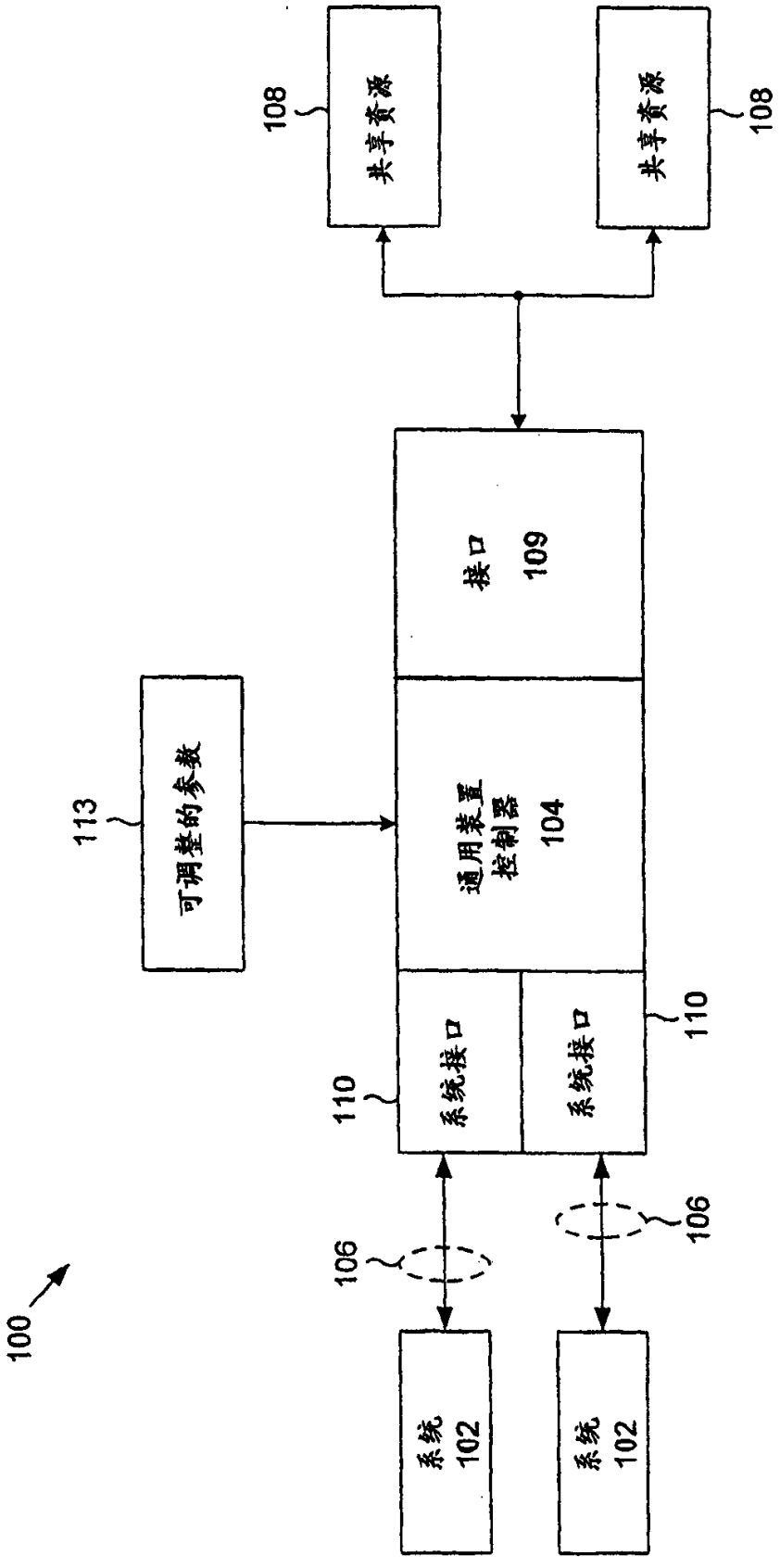

100
系统 102
系统 102
106
106
系统接口
系统接口
110
110
通用装置控制器 104
接口 109
可调整的参数
113
共享资源
共享资源
108
108

图 1A

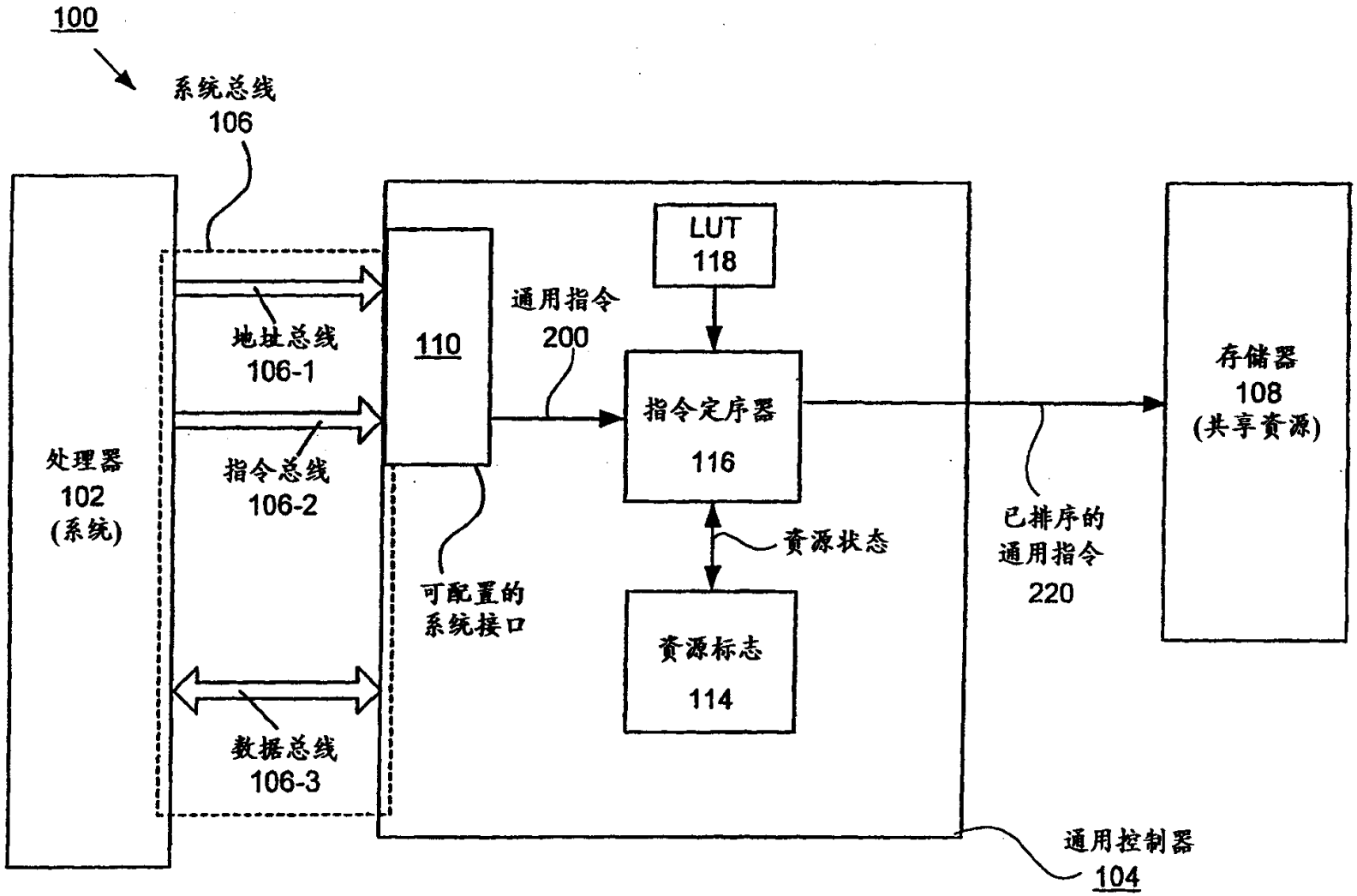

100
系统总线
106
处理器
102
(系统)
地址总线
106-1
指令总线
106-2
数据总线
106-3
110
可配置的
系统接口
通用指令
200
LUT
118
指令定序器
116
资源状态
资源标志
114
已排序的
通用指令
220
存储器
108
(共享资源)
通用控制器
104

图 1B

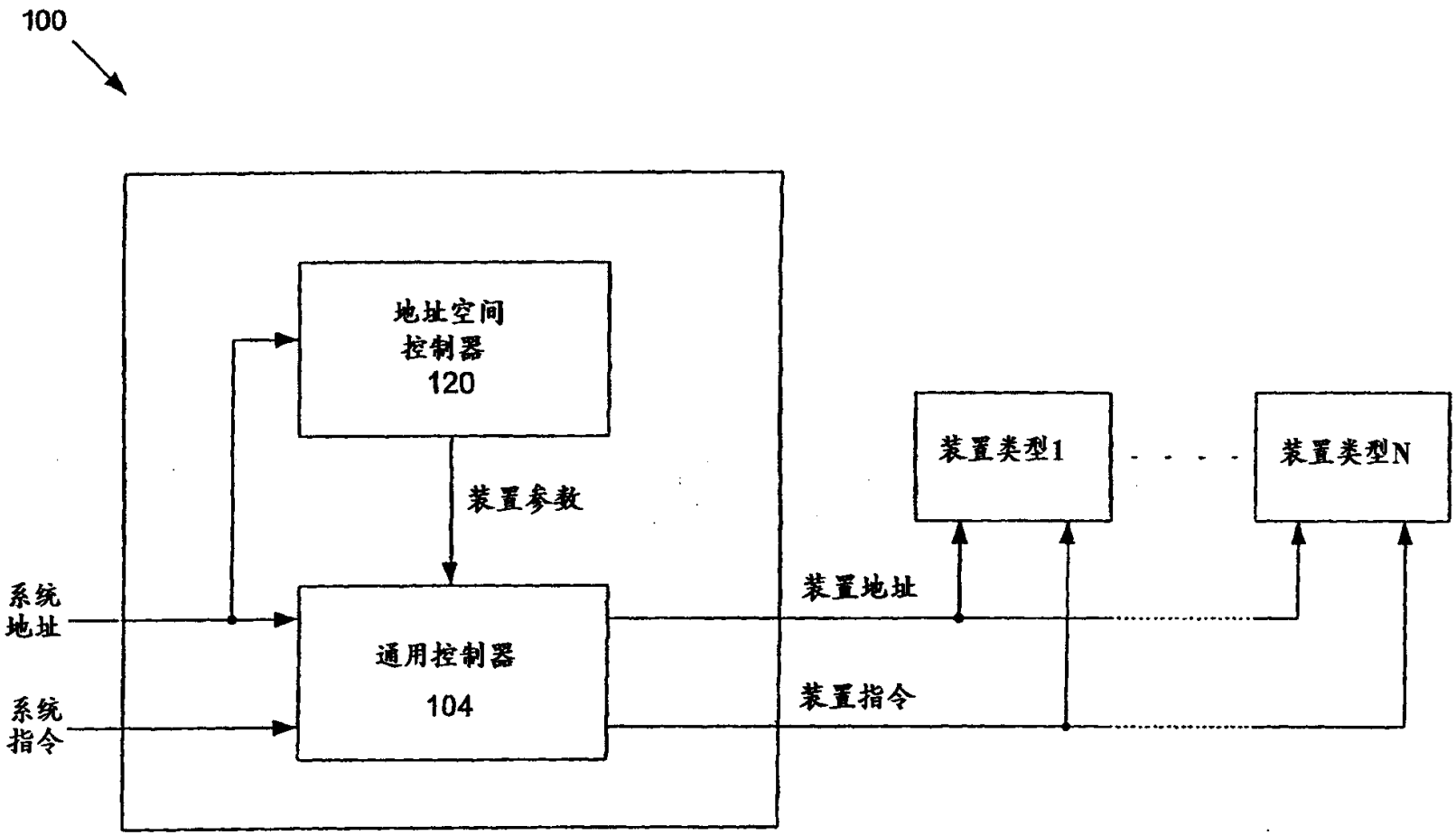

100
地址空间
控制器
120
装置参数
系统
地址
系统
指令
通用控制器
104
装置地址
装置指令
装置类型1
装置类型N

图 1C

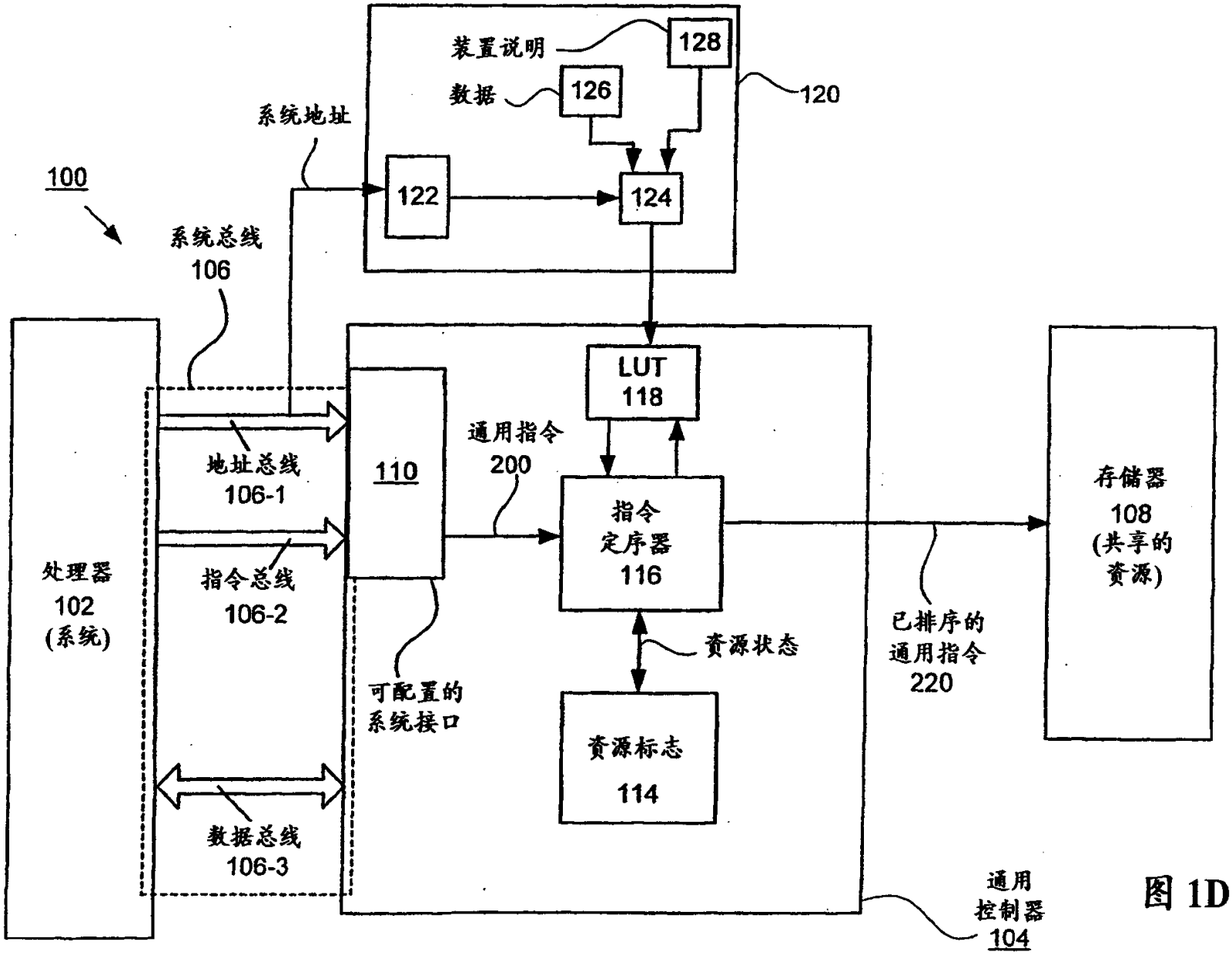
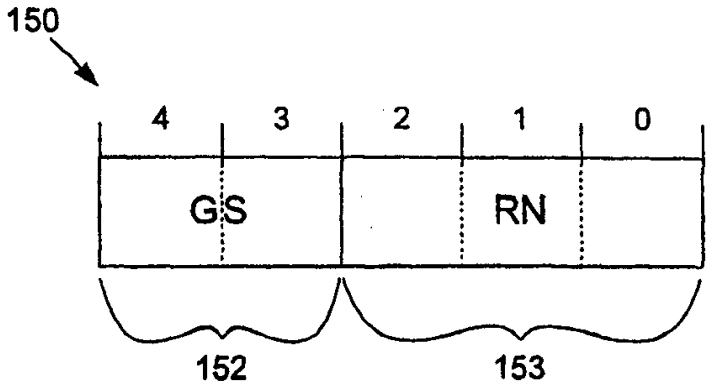

装置说明
128
数据
126
120
系统地址
100
122
124
系统总线
106
LUT
118
通用指令
200
地址总线
106-1
110
存储器
108
(共享的
资源)
处理器
102
(系统)
指令总线
106-2
指令
定序器
116
资源状态
已排序的
通用指令
220
可配置的
系统接口
资源标志
114
数据总线
106-3
通用
控制器
104

图 1D



RN - 请求编号
GS - 组选择器

图 1E.1

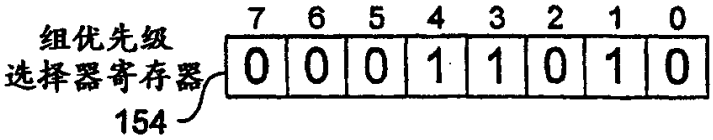
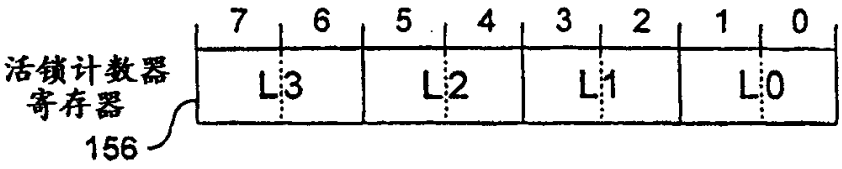


图 1E.2



L3...L0 - 活锁优先级“00”-“11”

图 1E.3

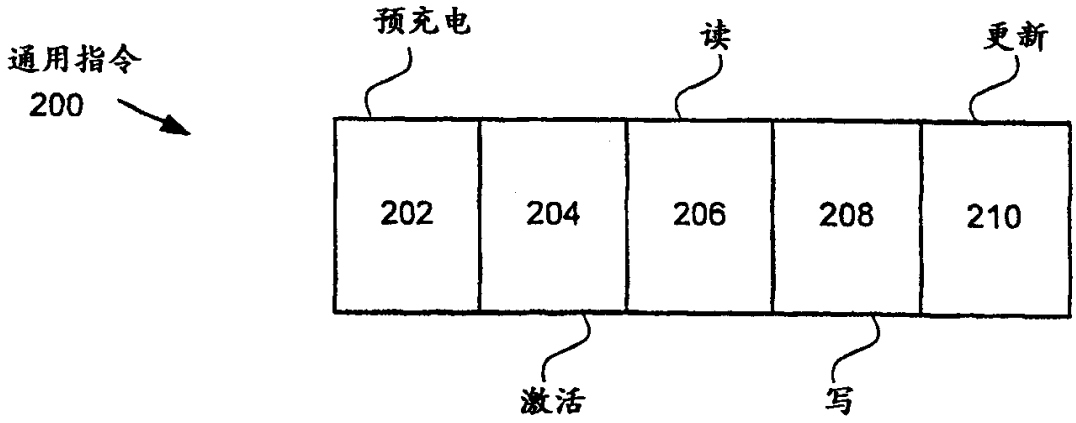

通用指令
200
预充电
读
更新
202
204
206
208
210
激活
写

图 2A

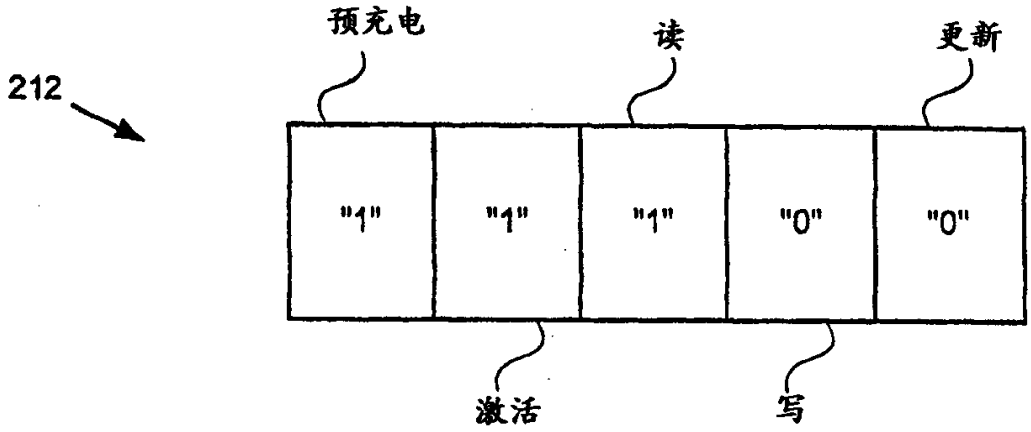

212
预充电
读
更新
"1"
"1"
"1"
"0"
"0"
激活
写

图 2B

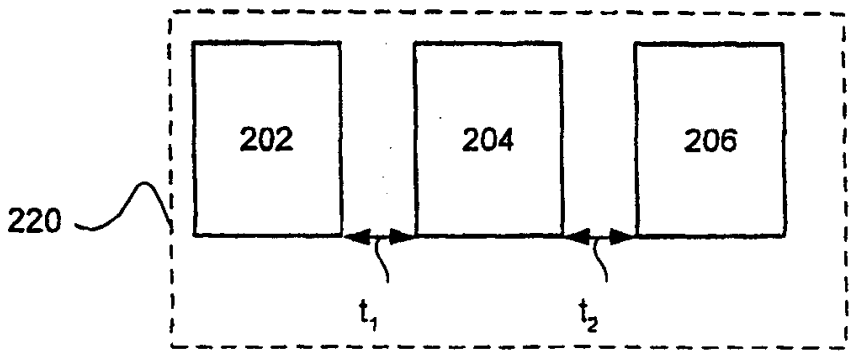

220
202
204
206
t_1
t_2

图 2C

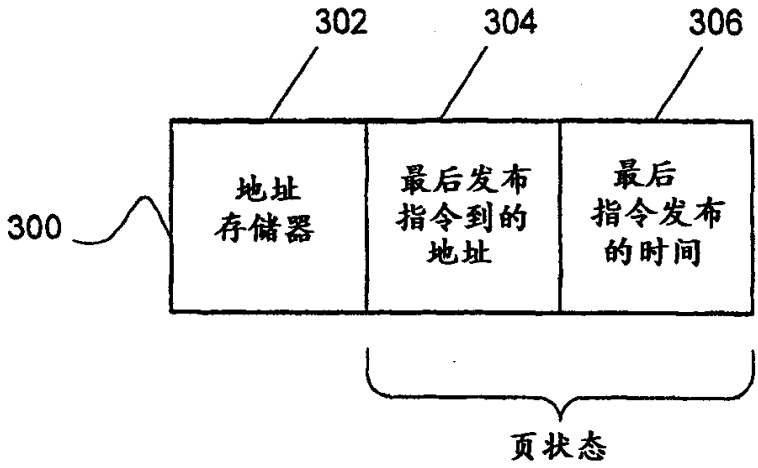

302
304
306
300
地址
存储器
最后发布
指令到的
地址
最后
指令发布
的时间
页状态

图 3A

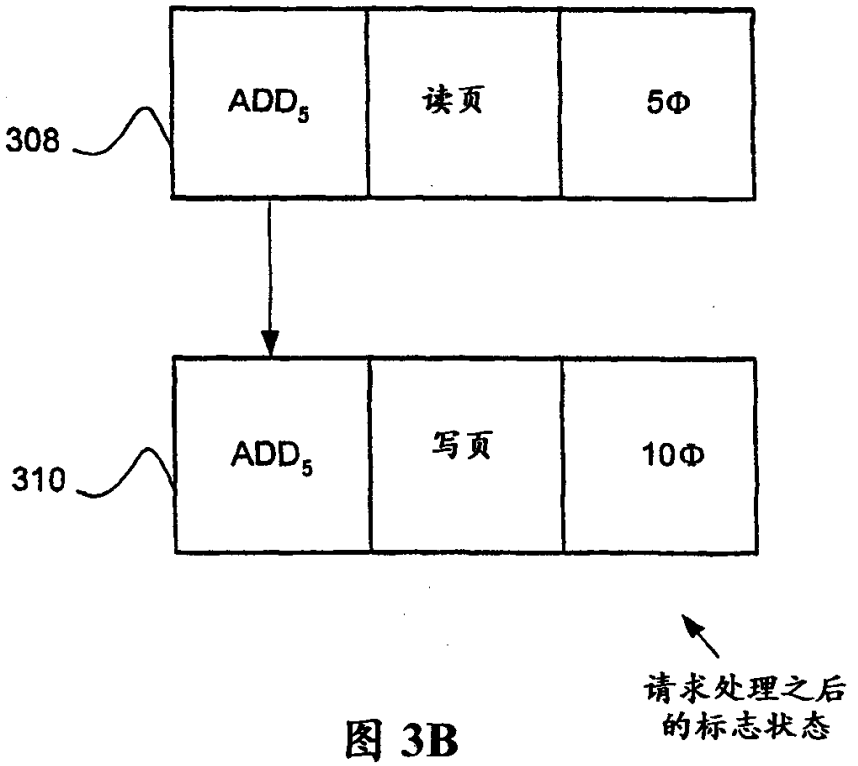

308
ADD5
读页
5Φ
310
ADD5
写页
10Φ
请求处理之后
的标志状态

图 3B

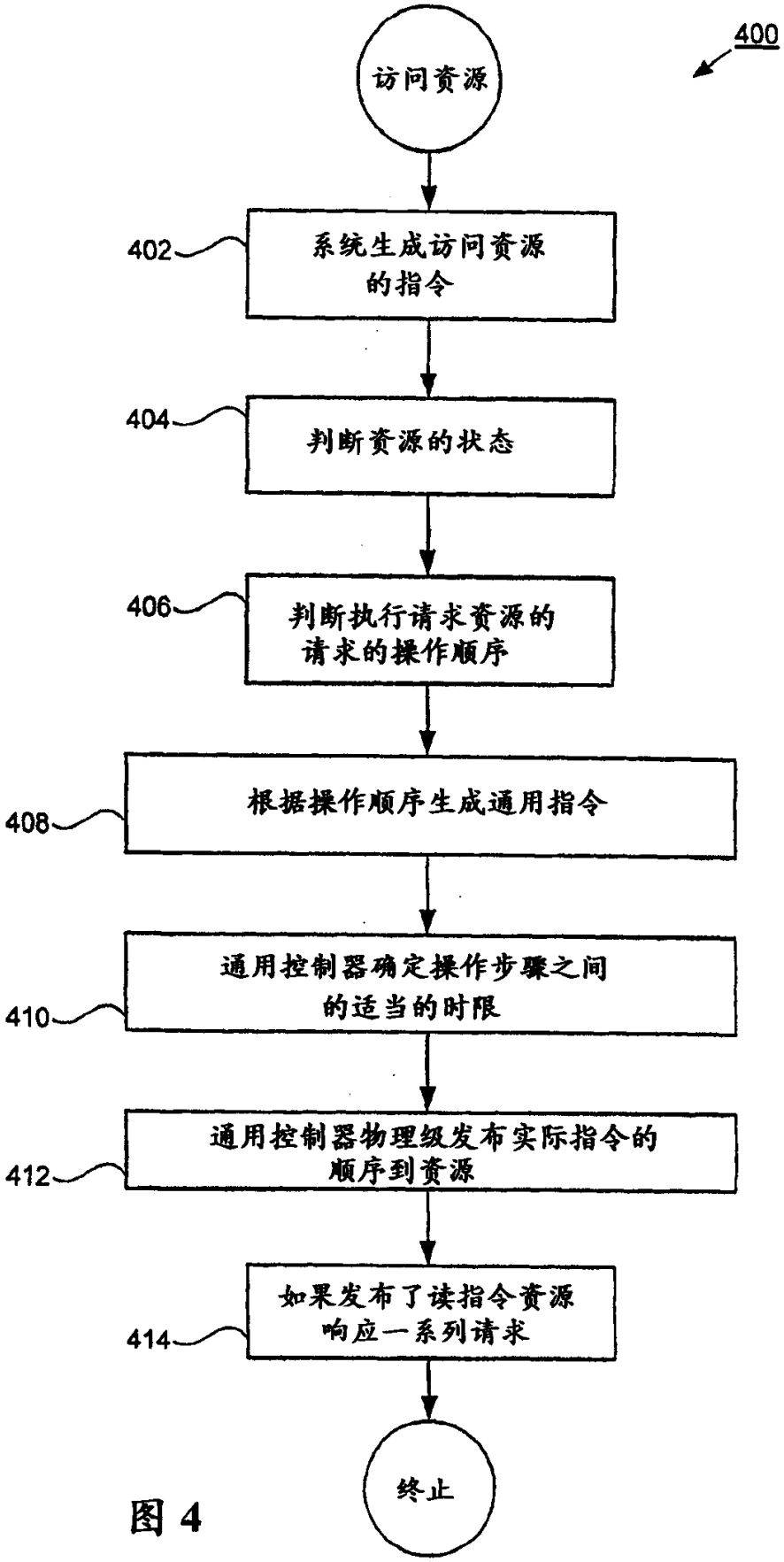

400
访问资源
402
系统生成访问资源的指令
404
判断资源的状态
406
判断执行请求资源的请求的操作顺序
408
根据操作顺序生成通用指令
410
通用控制器确定操作步骤之间的适当的时限
412
通用控制器物理级发布实际指令的顺序到资源
414
如果发布了读指令资源响应一系列请求
终止

图 4

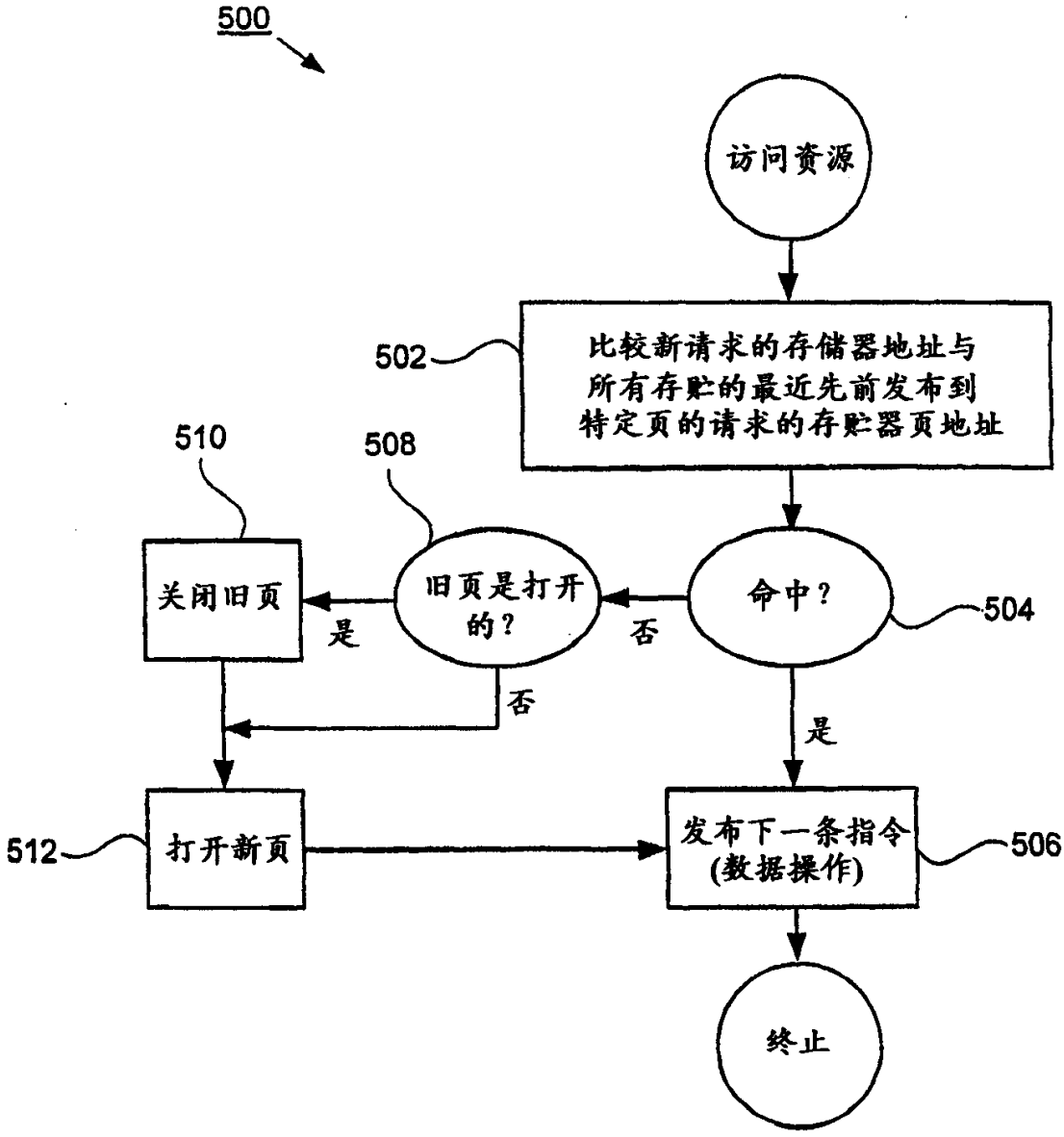

500
访问资源
502
比较新请求的存储器地址与
所有存贮的最近先前发布到
特定页的请求的存贮器页地址
510
508
关闭旧页
旧页是打开
的？
命中？
504
是
否
否
是
512
打开新页
发布下一条指令
(数据操作)
506
终止

图 5

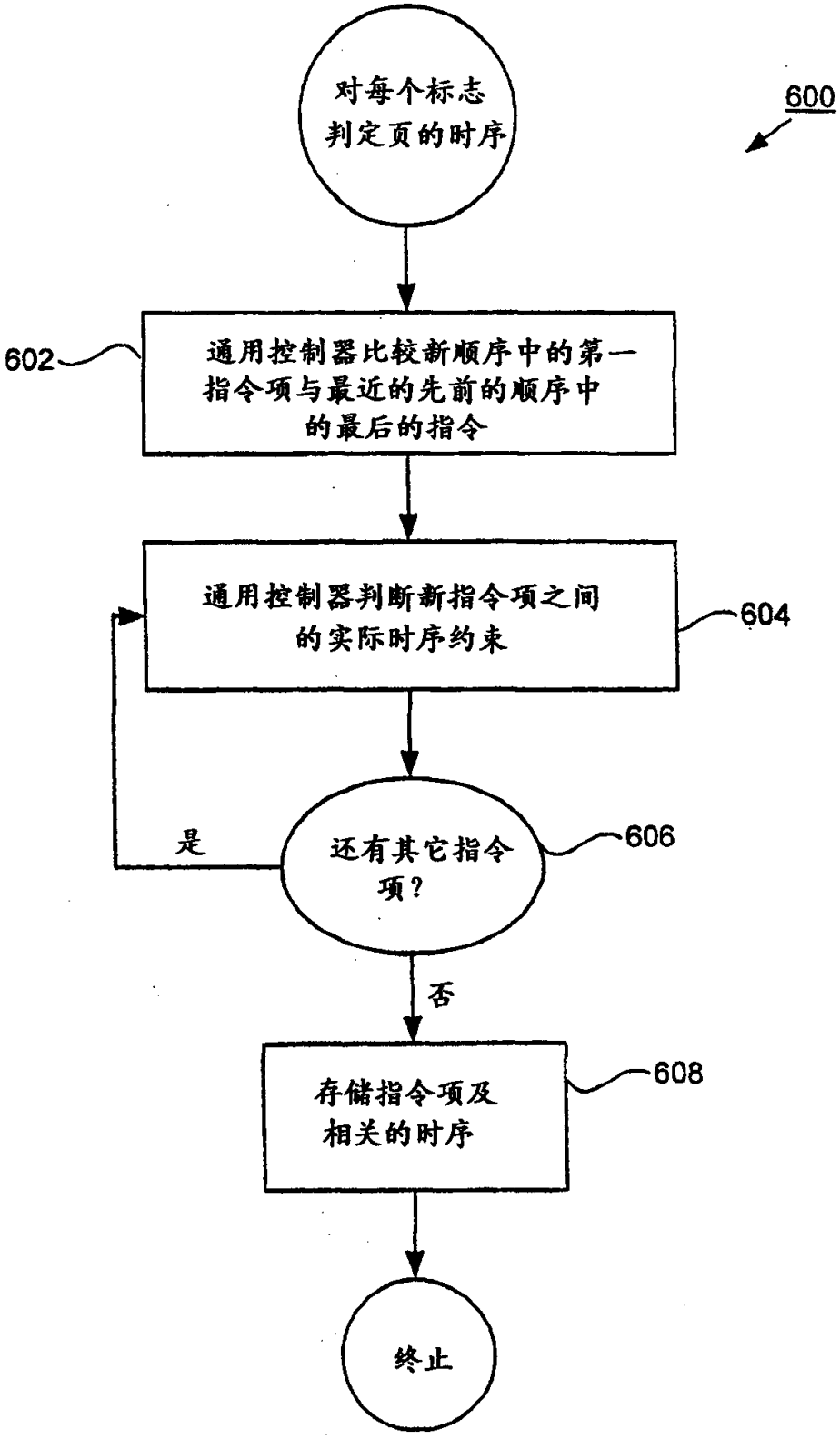

对每个标志判定页的时序
600
602
通用控制器比较新顺序中的第一指令项与最近的先前的顺序中的最后的指令
通用控制器判断新指令项之间的实际时序约束
604
还有其它指令项？
606
是
否
存储指令项及相关的时序
608
终止

图 6

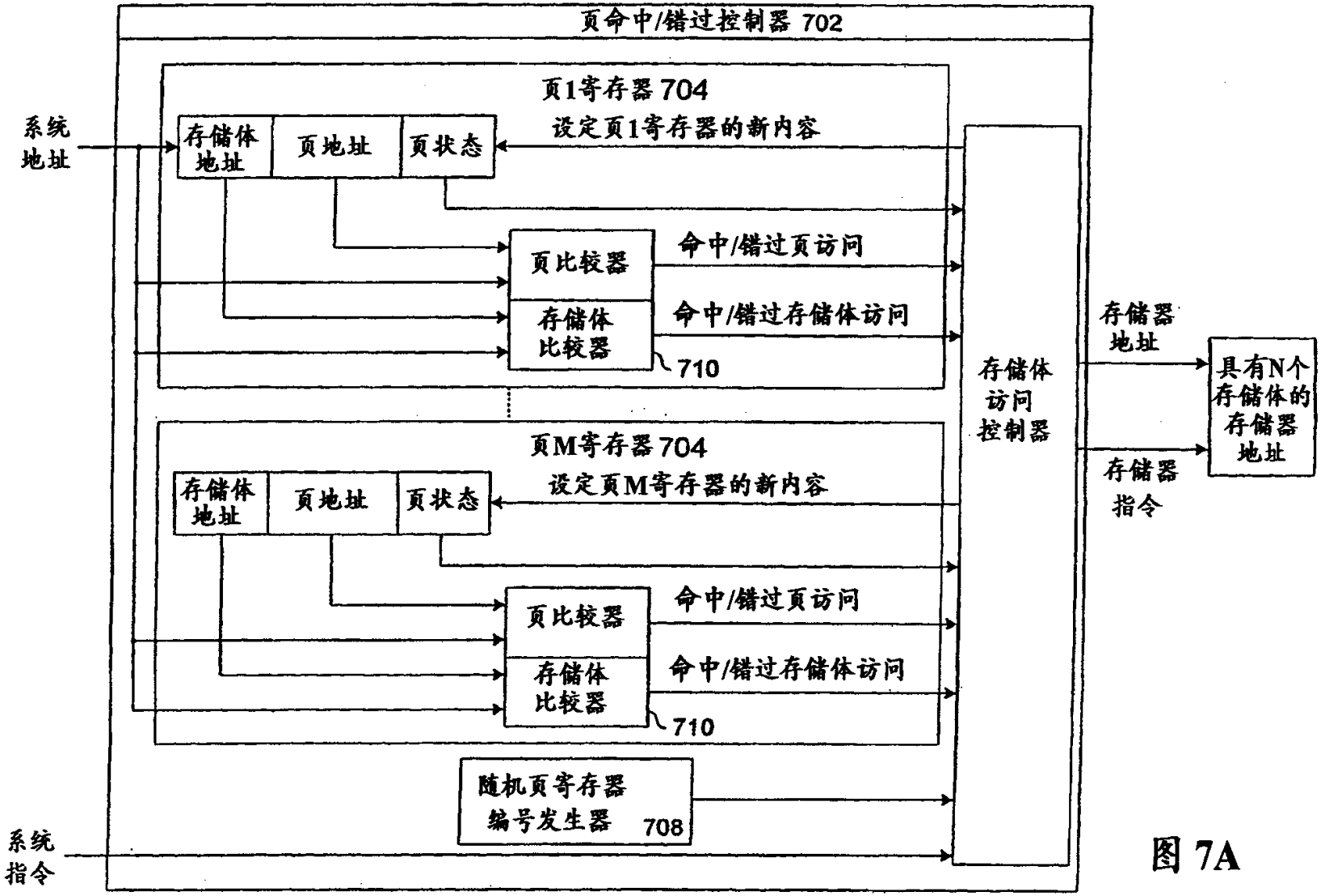

页命中/错过控制器 702
页1寄存器 704
系统地址
存储体地址
页地址
页状态
设定页1寄存器的新内容
页比较器
命中/错过页访问
存储体比较器
命中/错过存储体访问
710
页M寄存器 704
设定页M寄存器的新内容
存储体访问控制器
存储器地址
具有N个存储体的存储器地址
存储器指令
随机页寄存器编号发生器 708
系统指令

图 7A

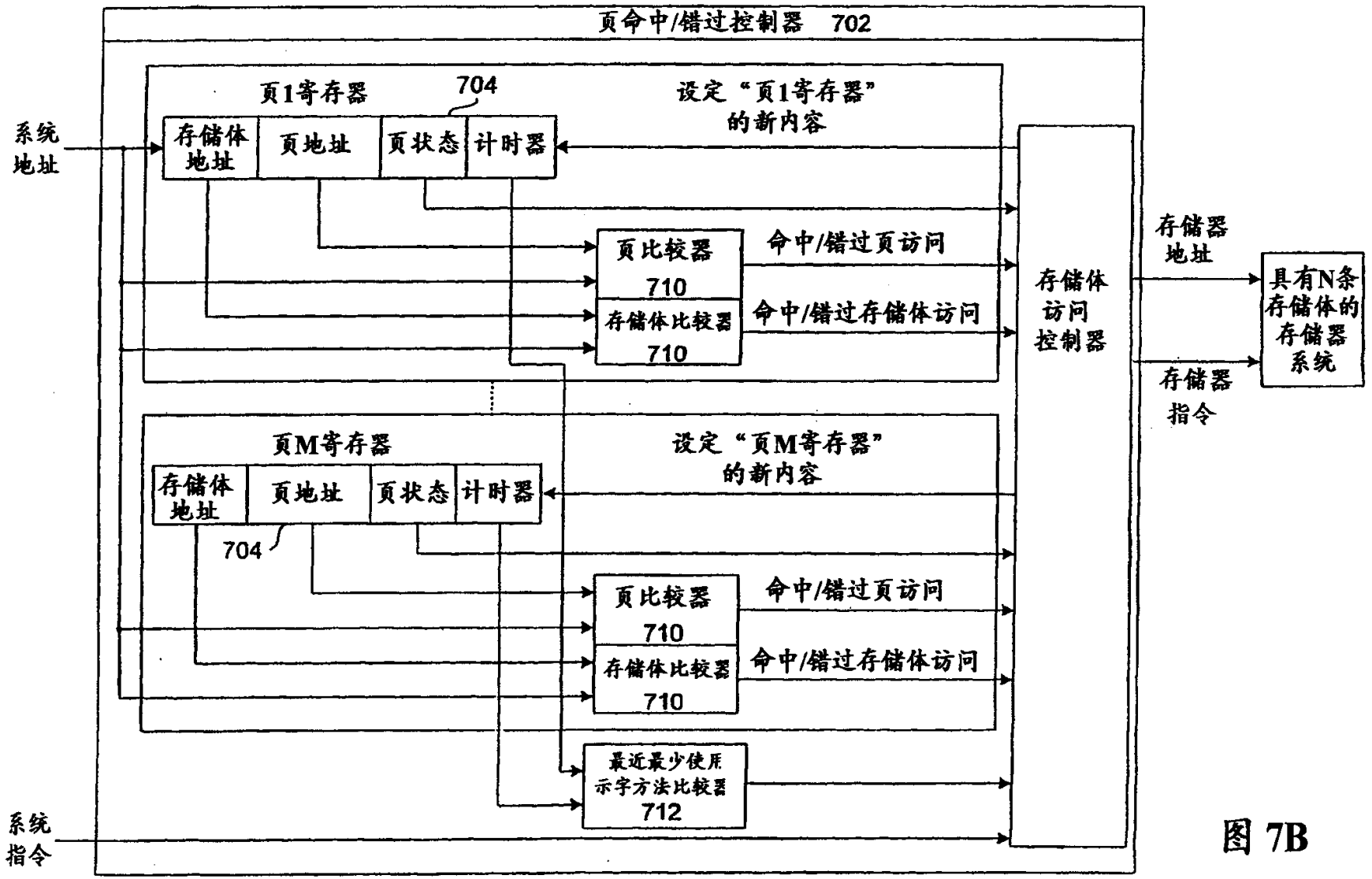

页命中/错过控制器 702
页1寄存器
704
设定“页1寄存器”的新内容
系统地址
存储体地址
页地址
页状态
计时器
页比较器
710
命中/错过页访问
存储体比较器
710
命中/错过存储体访问
存储体访问控制器
存储器地址
具有N条存储体的存储器系统
存储器指令
页M寄存器
设定“页M寄存器”的新内容
存储体地址
页地址
页状态
计时器
704
页比较器
710
命中/错过页访问
存储体比较器
710
命中/错过存储体访问
最近最少使用示字方法比较器
712
系统指令

图 7B

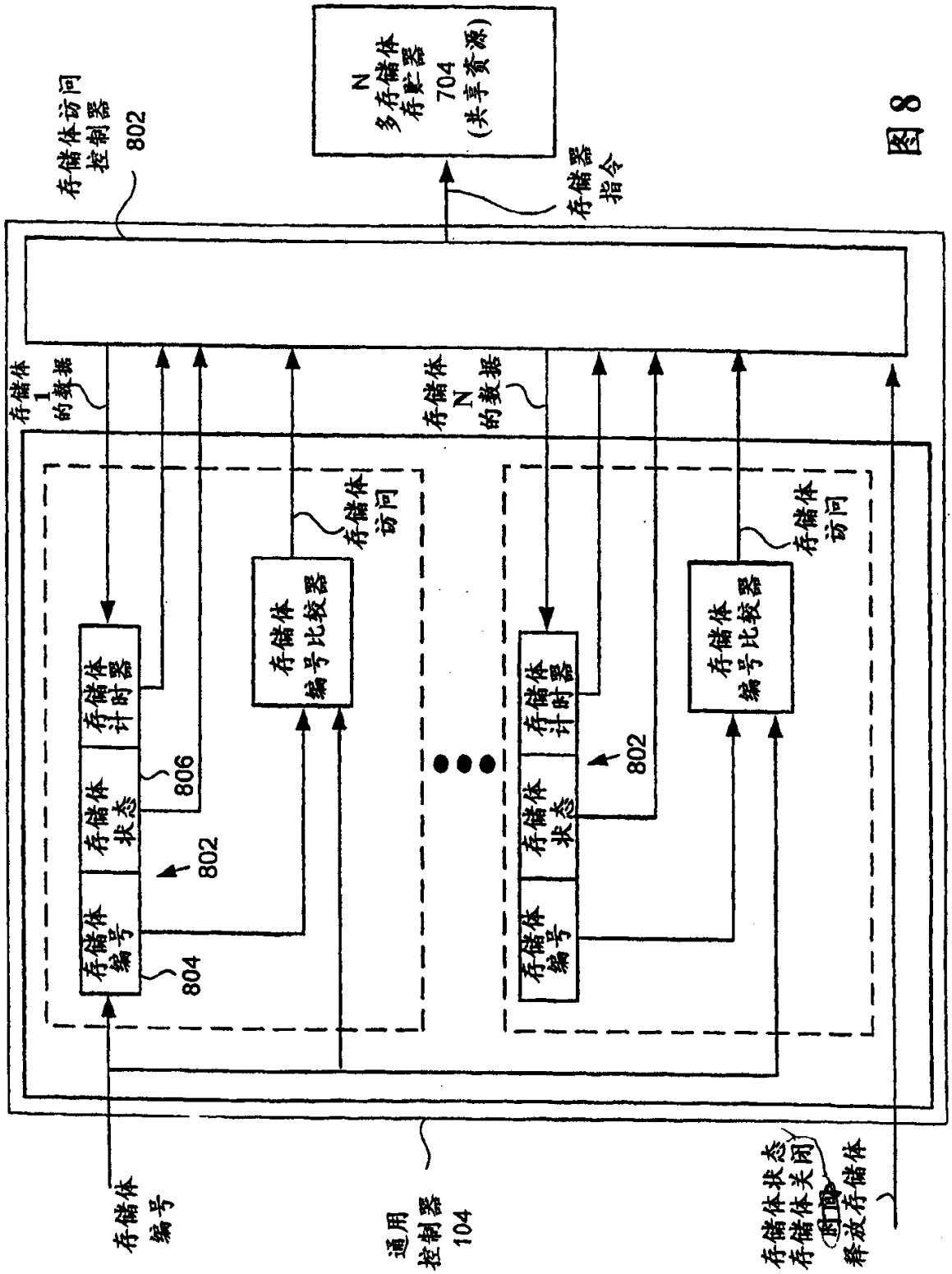

N
多存储体
存贮器
704
(共享资源)
存储器
指令
存储体访问
控制器
802
存储体
1
的数据
存储体
N
的数据
存储体
访问
存储体
计时器
存储体
状态
存储体
编号
存储体
编号比较器
806
802
804
存储体
编号
通用
控制器
104
存储体状态
存储体关闭
时间
释放存储体

图 8

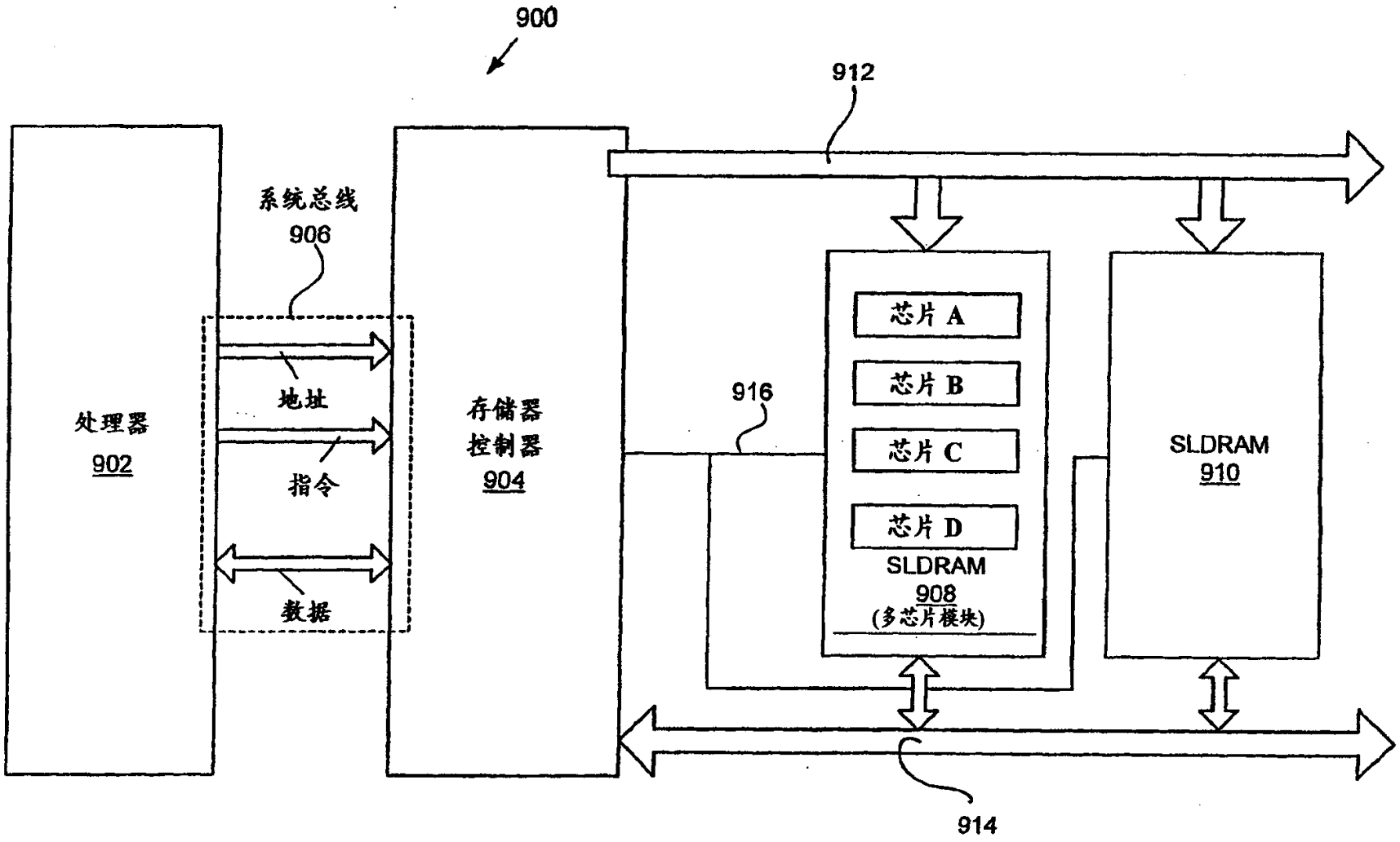

900
912
系统总线
906
处理器
902
地址
指令
数据
存储器
控制器
904
916
芯片 A
芯片 B
芯片 C
芯片 D
SLDRAM
908
(多芯片模块)
SLDRAM
910
914

图 9A

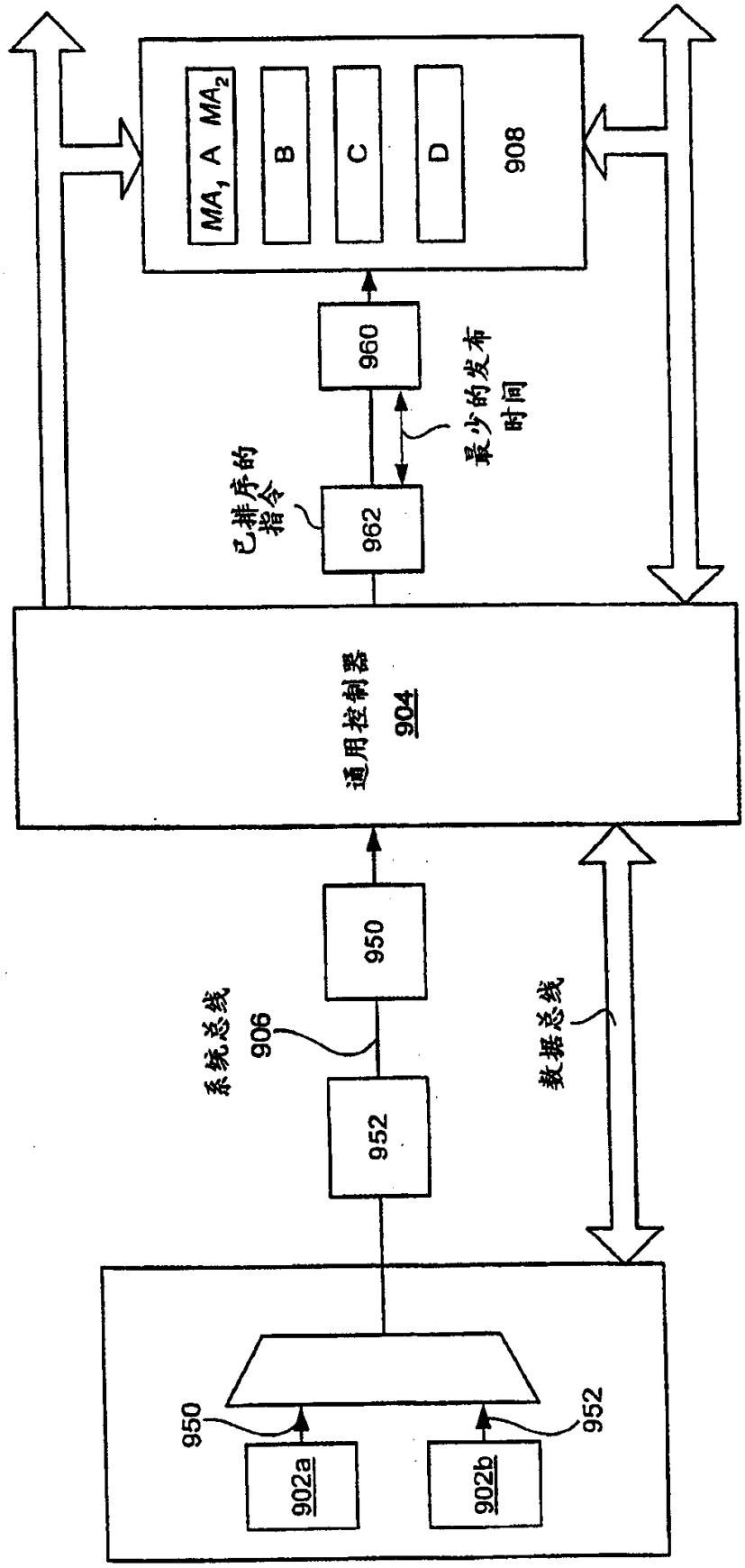

902a
902b
950
952
数据总线
系统总线
906
950
952
通用控制器
904
已排序的
指令
962
960
最少的发布
时间
MA1, A MA2
B
C
D
908

图 9B

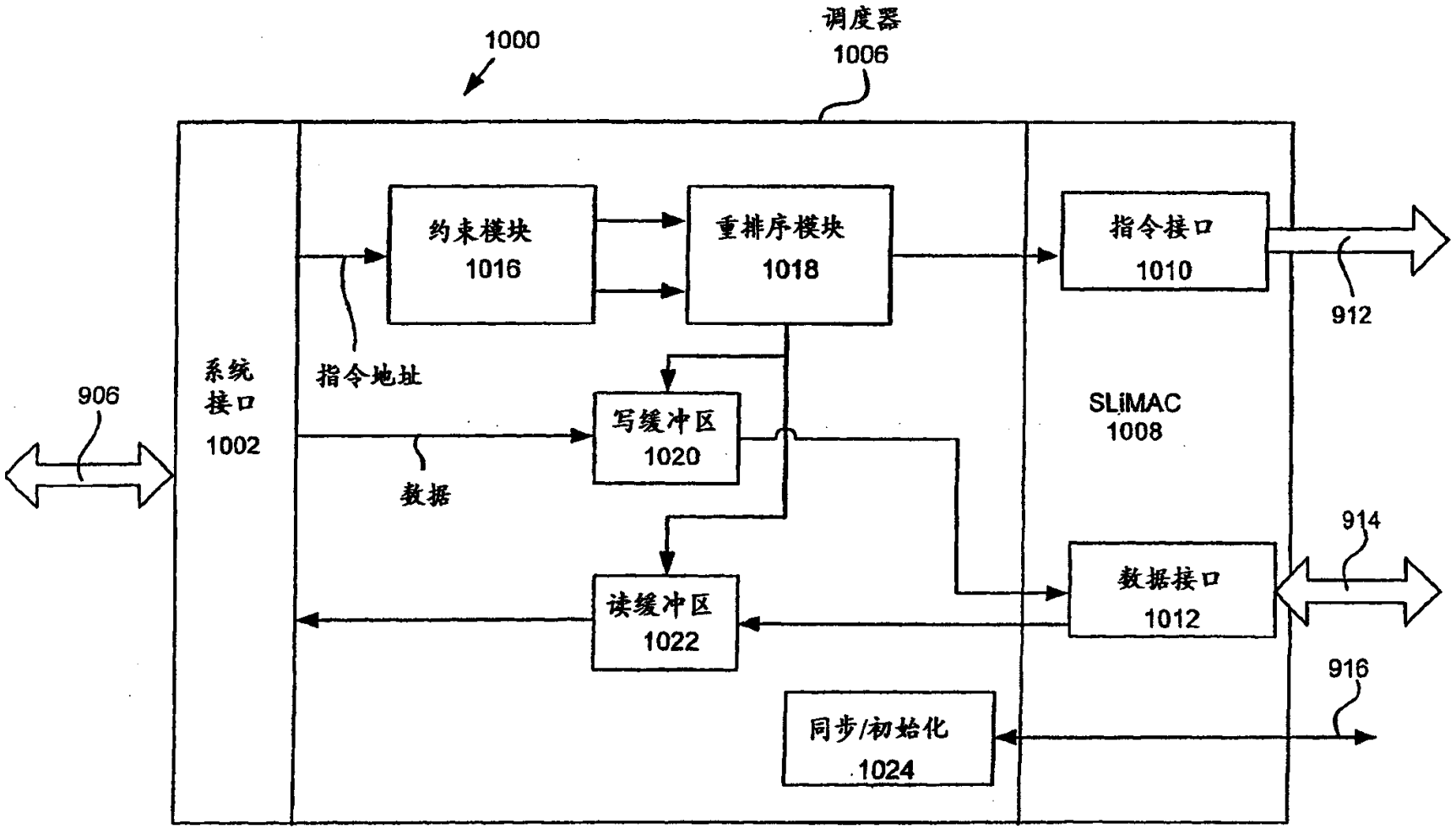

1000
调度器
1006
约束模块
1016
重排序模块
1018
指令接口
1010
912
906
系统
接口
1002
指令地址
写缓冲区
1020
SLiMAC
1008
数据
914
数据接口
1012
读缓冲区
1022
916
同步/初始化
1024

图 10

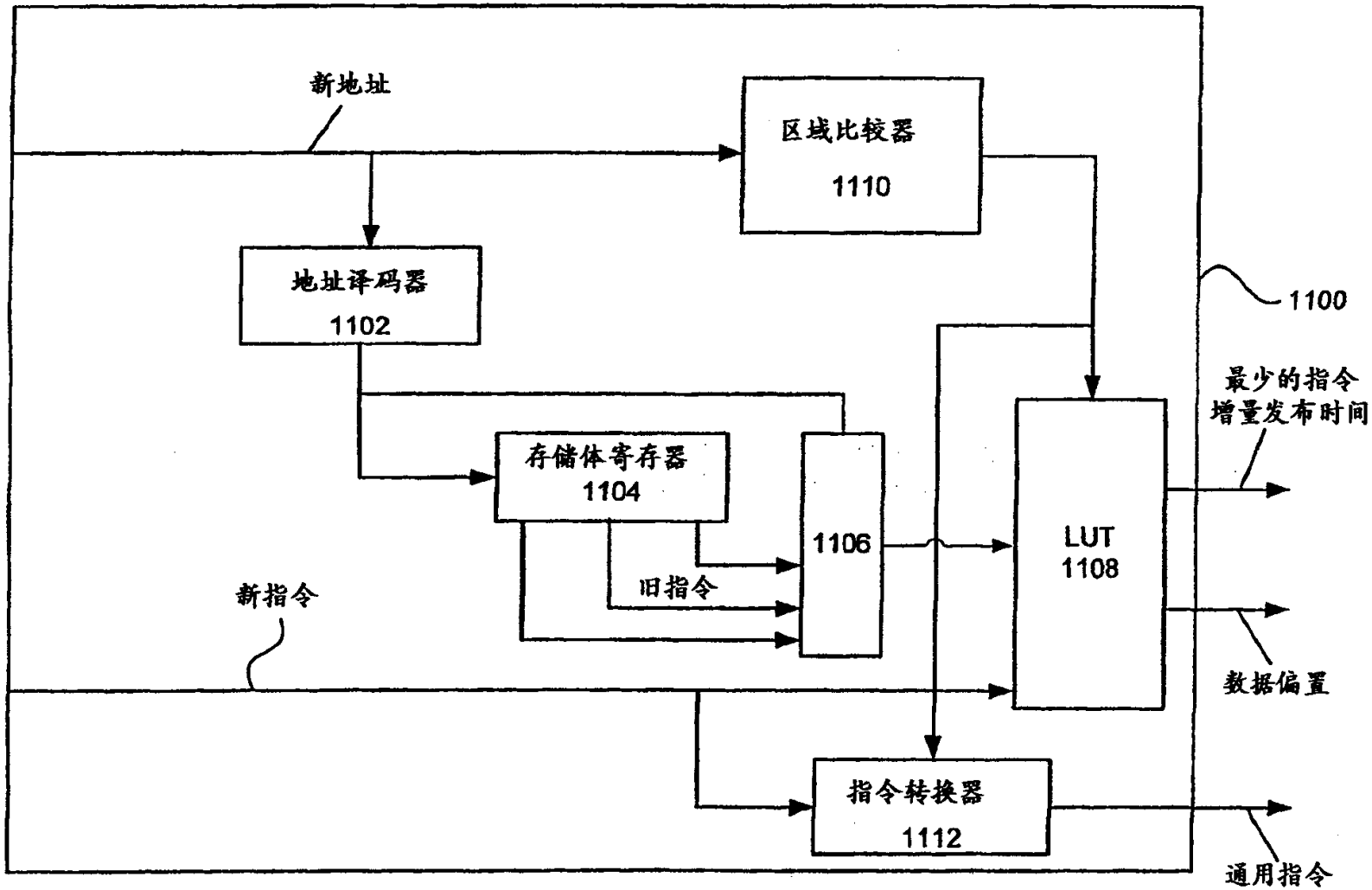

新地址
区域比较器
1110
地址译码器
1102
1100
最少的指令
增量发布时间
存储体寄存器
1104
1106
LUT
1108
旧指令
新指令
数据偏置
指令转换器
1112
通用指令

图 11

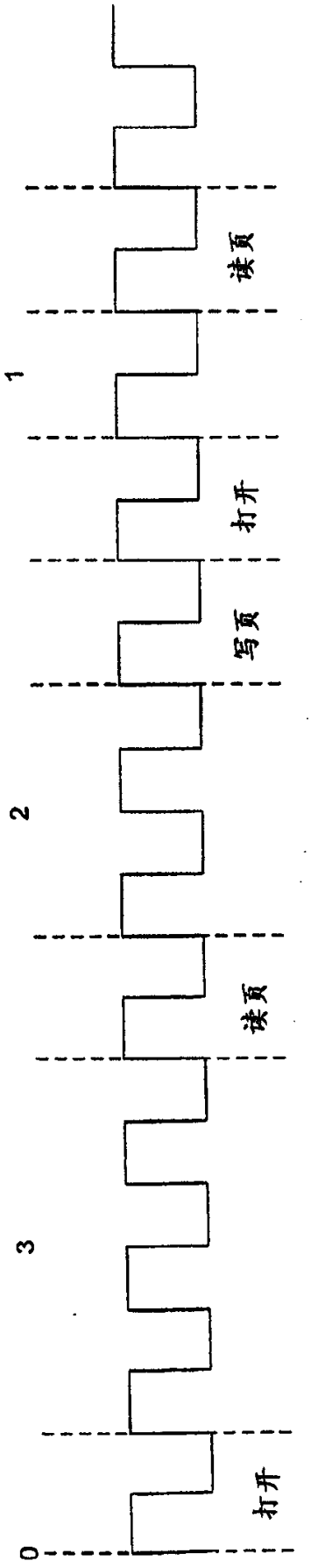


图 12

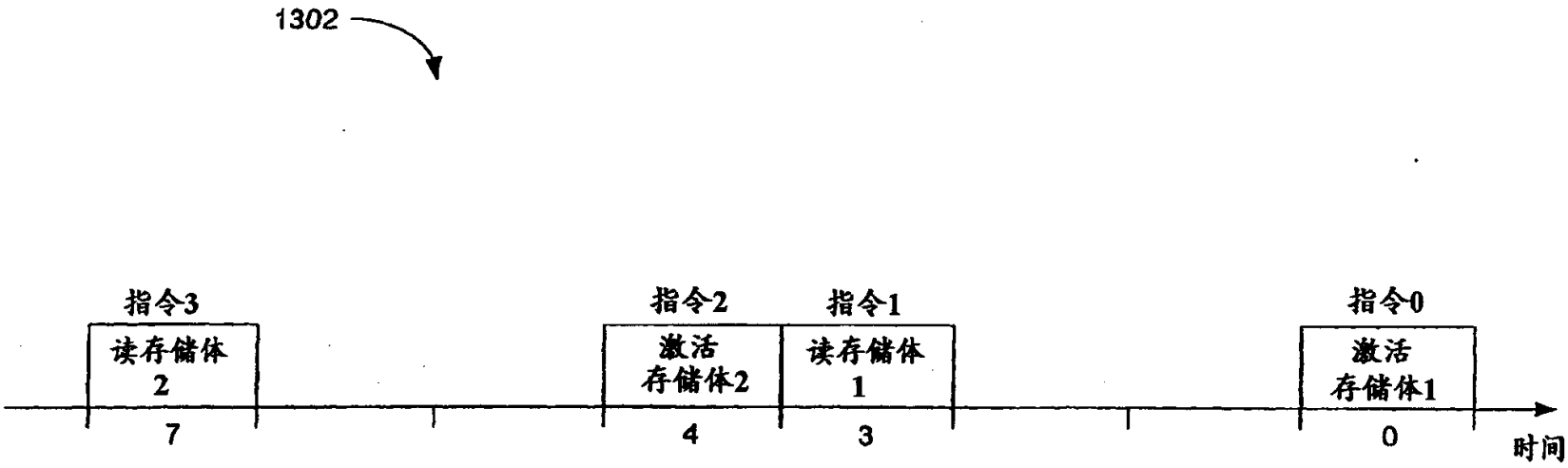


图 13A

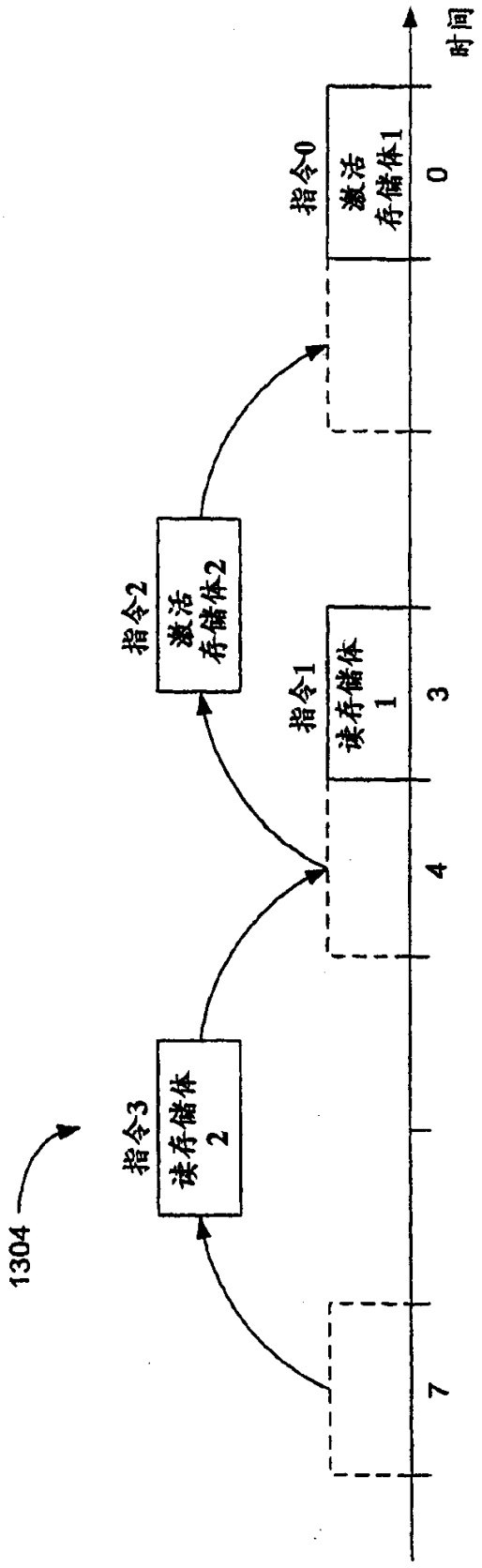


图 13B

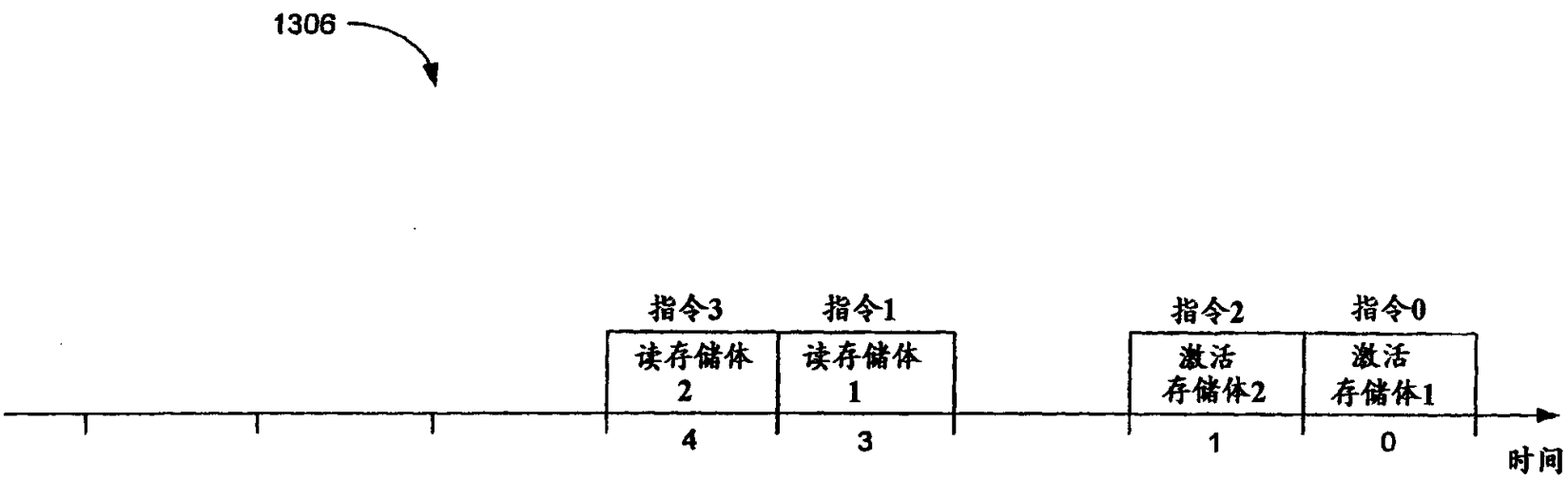


图 13C

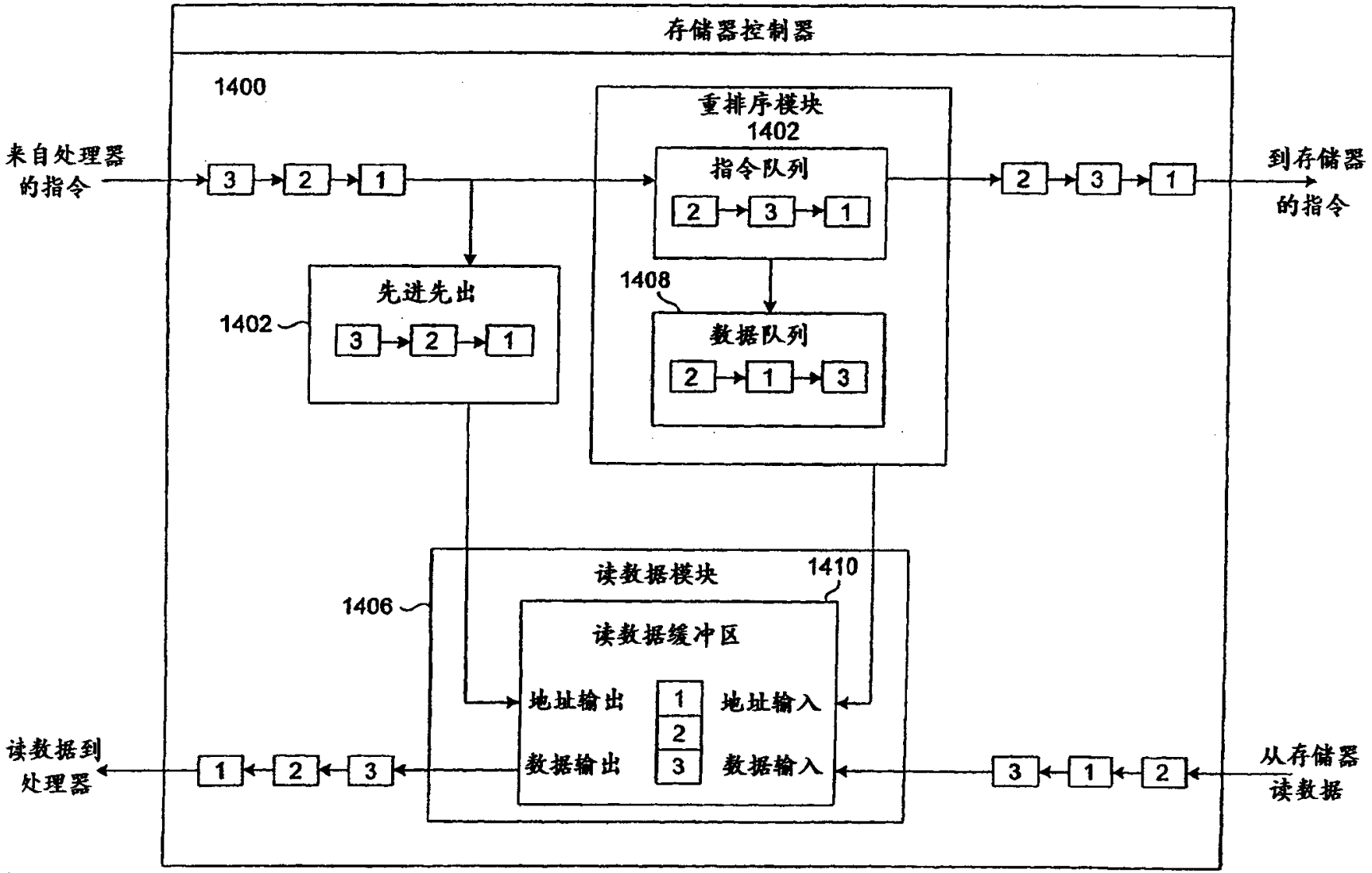

存储器控制器
1400
来自处理器的指令
3
2
1
重排序模块
1402
指令队列
2
3
1
2
3
1
到存储器的指令
先进先出
1402
3
2
1
1408
数据队列
2
1
3
读数据模块
1410
1406
读数据缓冲区
地址输出
1
2
3
地址输入
数据输出
数据输入
读数据到处理器
1
2
3
3
1
2
从存储器读数据

图 14

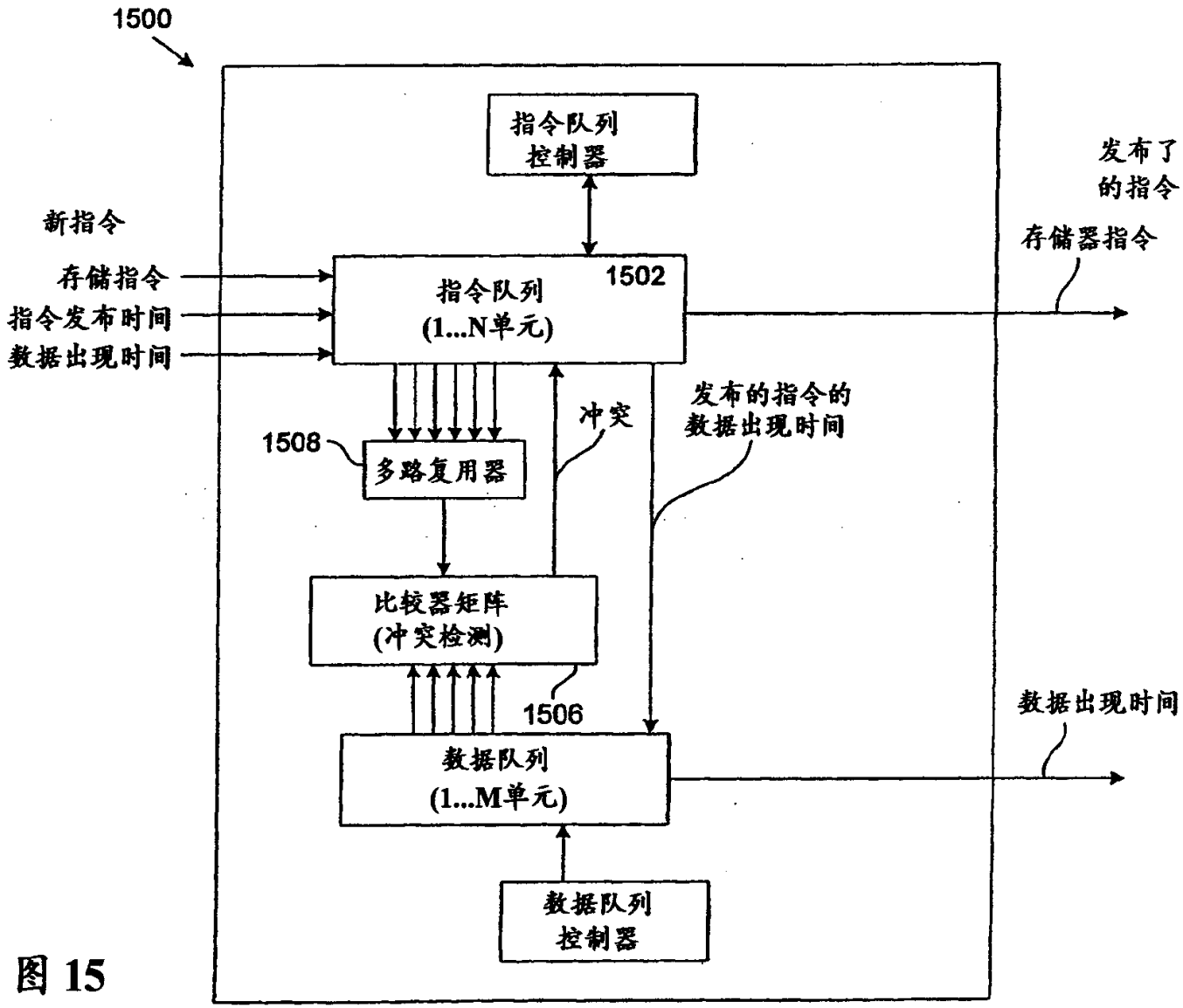

1500
指令队列
控制器
发布了
的指令
新指令
存储器指令
存储指令
指令发布时间
数据出现时间
指令队列
(1...N单元)
1502
冲突
发布的指令的
数据出现时间
1508
多路复用器
比较器矩阵
(冲突检测)
1506
数据出现时间
数据队列
(1...M单元)
数据队列
控制器

图 15

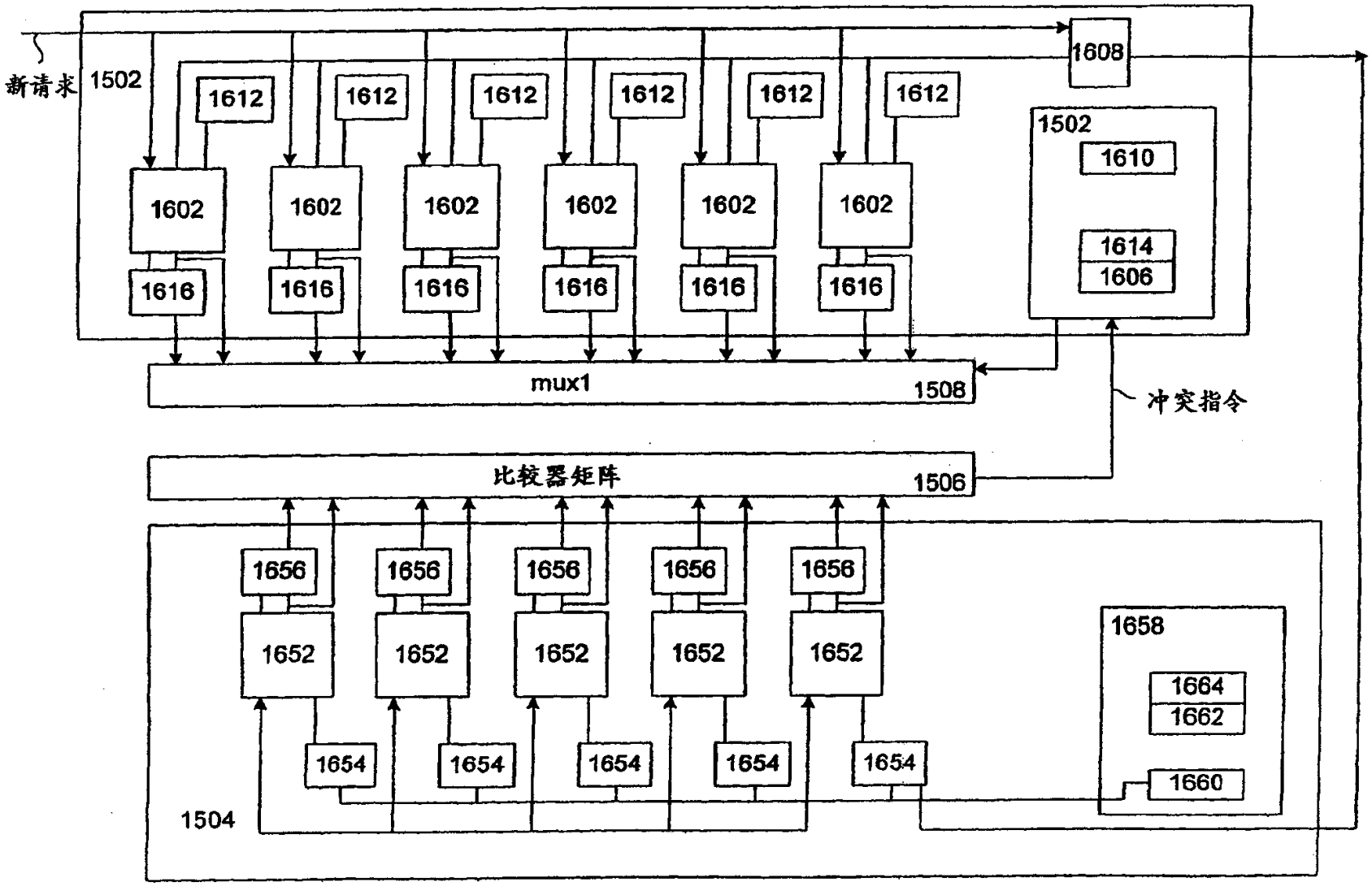

新请求
1502
1612
1602
1616
1608
1502
1610
1614
1606
mux1
1508
冲突指令
比较器矩阵
1506
1656
1652
1654
1658
1664
1662
1660
1504

图 16

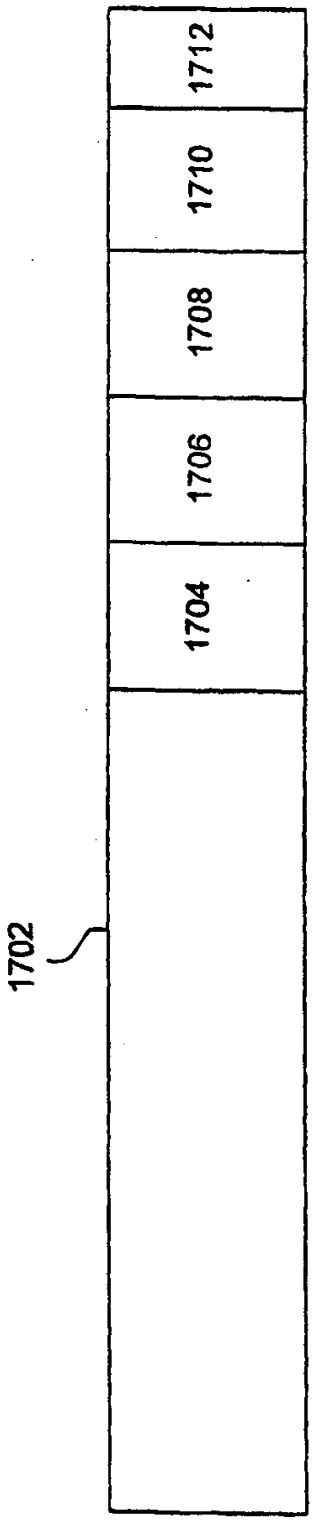


图 17

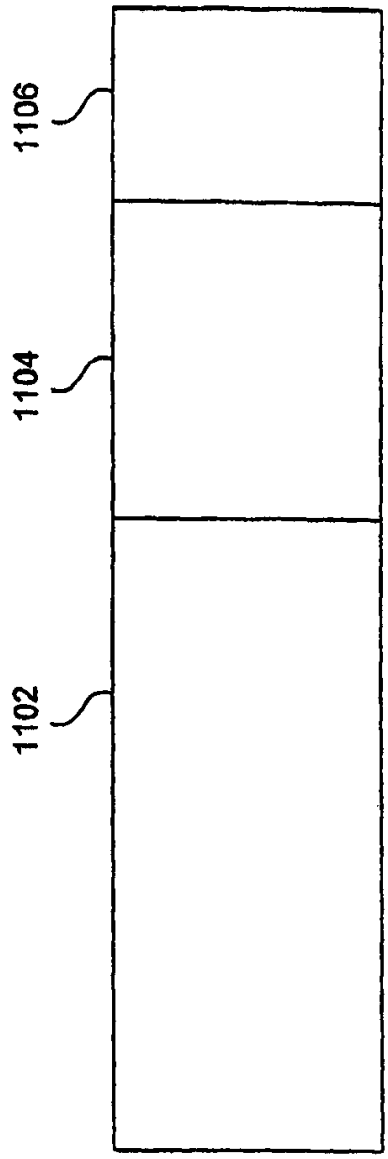


图 19

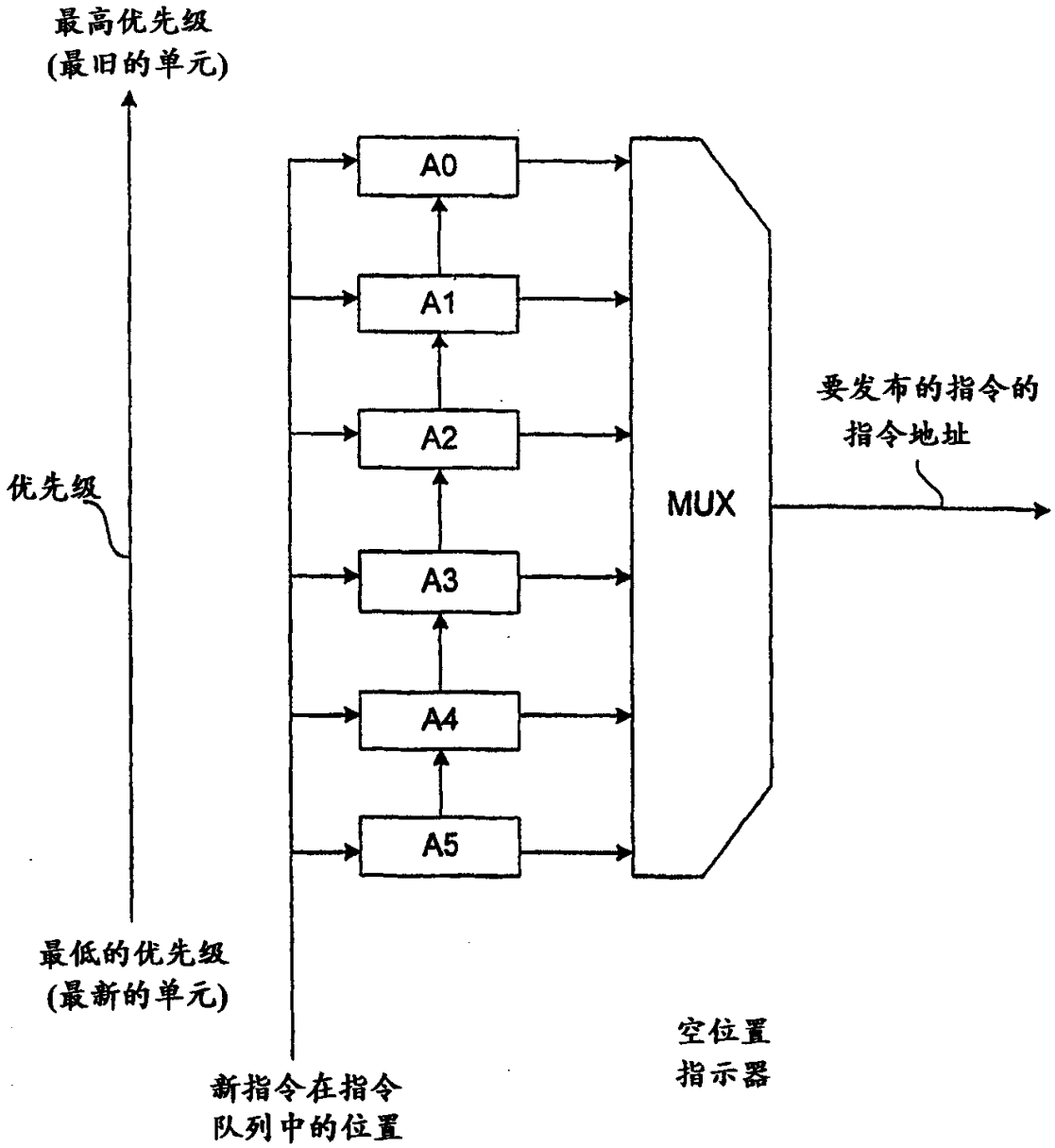

最高优先级
(最旧的单元)
A0
A1
A2
A3
A4
A5
MUX
要发布的指令的
指令地址
优先级
最低的优先级
(最新的单元)
空位置
指示器
新指令在指令
队列中的位置

图 18

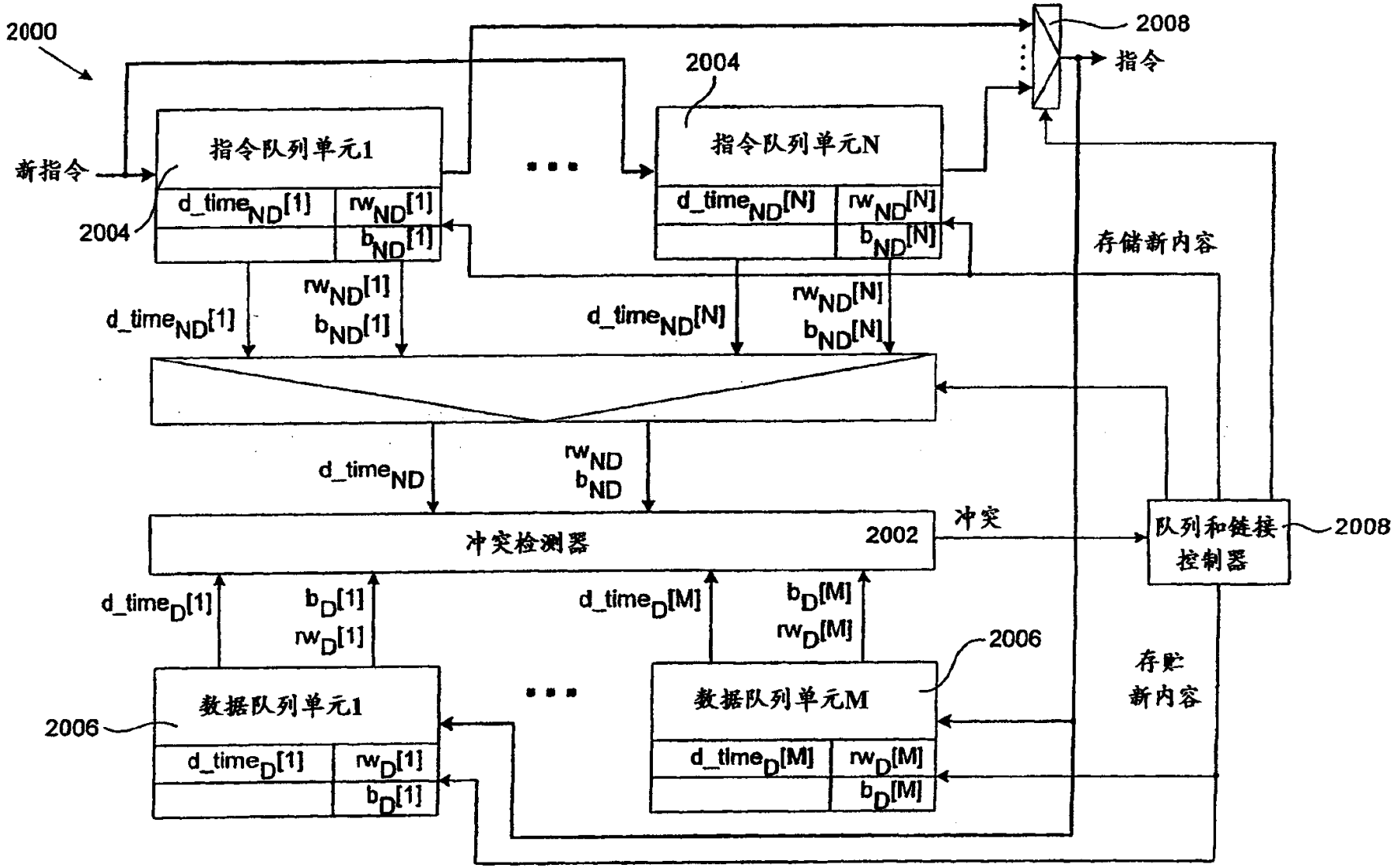

2000
2008
指令
2004
指令队列单元1
指令队列单元N
新指令
2004
d_time_ND[1] rw_ND[1] b_ND[1]
d_time_ND[N] rw_ND[N] b_ND[N]
存储新内容
d_time_ND[1]
rw_ND[1] b_ND[1]
d_time_ND[N]
rw_ND[N] b_ND[N]
d_time_ND
rw_ND b_ND
冲突检测器
2002
冲突
队列和链接控制器
2008
d_time_D[1]
b_D[1] rw_D[1]
d_time_D[M]
b_D[M] rw_D[M]
2006
存贮新内容
2006
数据队列单元1
数据队列单元M
d_time_D[1] rw_D[1] b_D[1]
d_time_D[M] rw_D[M] b_D[M]

图 20

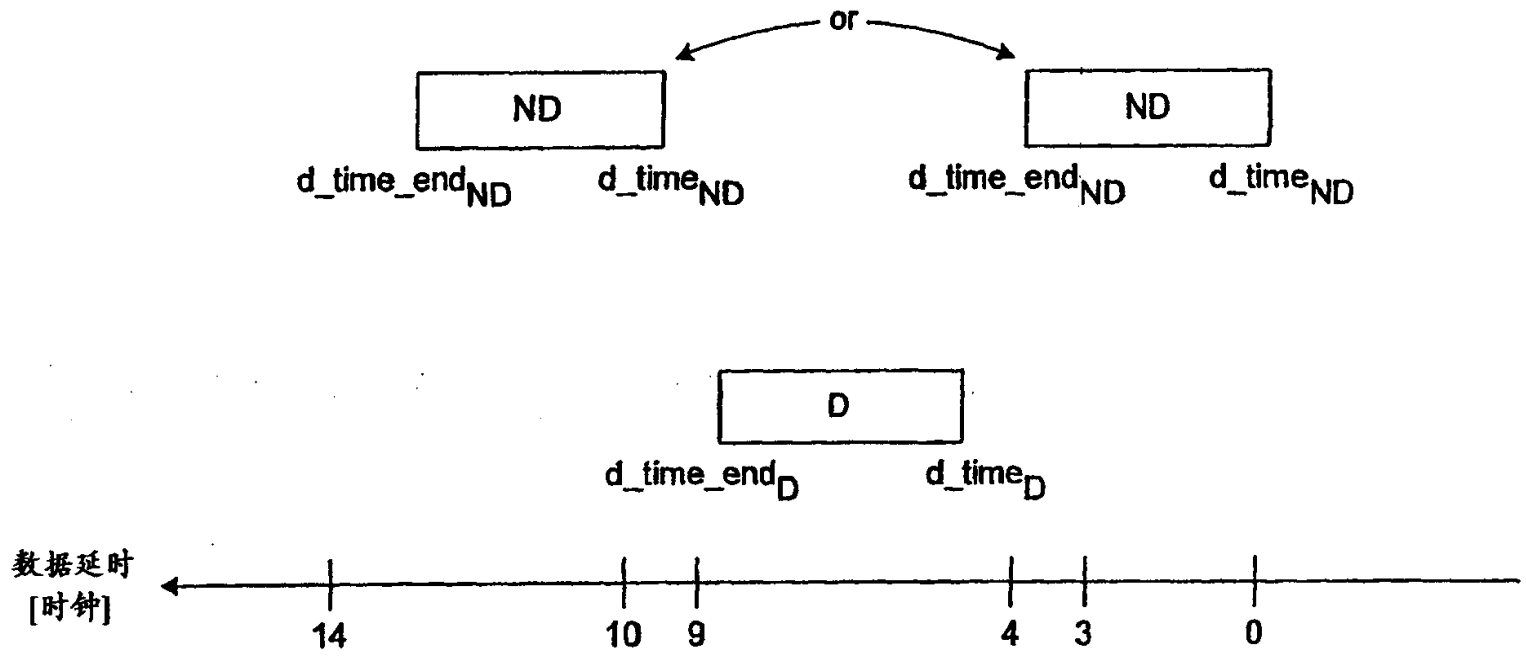

or
ND
ND
d_time_end_ND
d_time_ND
d_time_end_ND
d_time_ND
D
d_time_end_D
d_time_D
数据延时
[时钟]
14
10
9
4
3
0

图 21

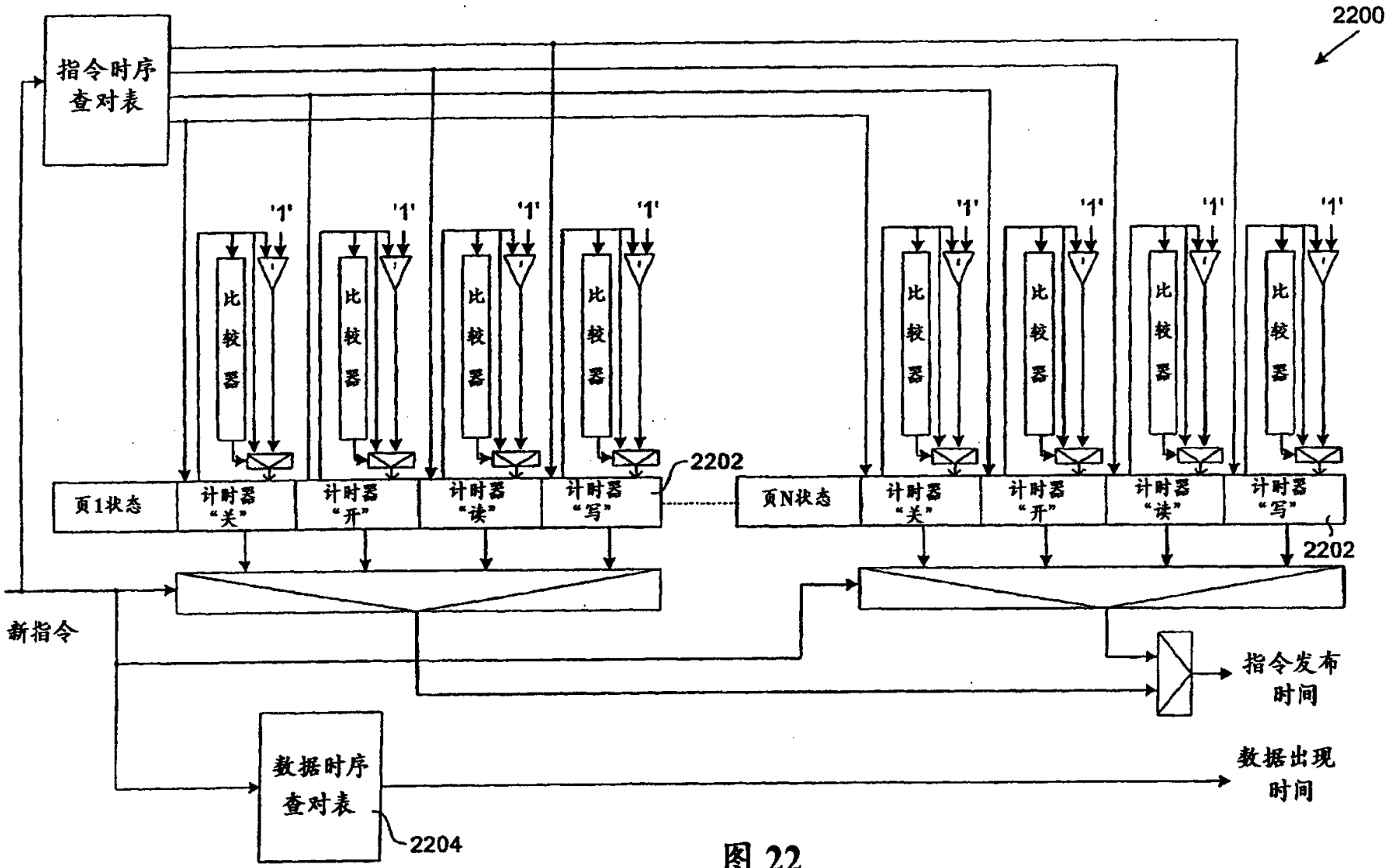

2200
指令时序查对表
'1'
比较器
2202
页1状态
计时器"关"
计时器"开"
计时器"读"
计时器"写"
页N状态
新指令
指令发布时间
数据时序查对表
2204
数据出现时间

图 22

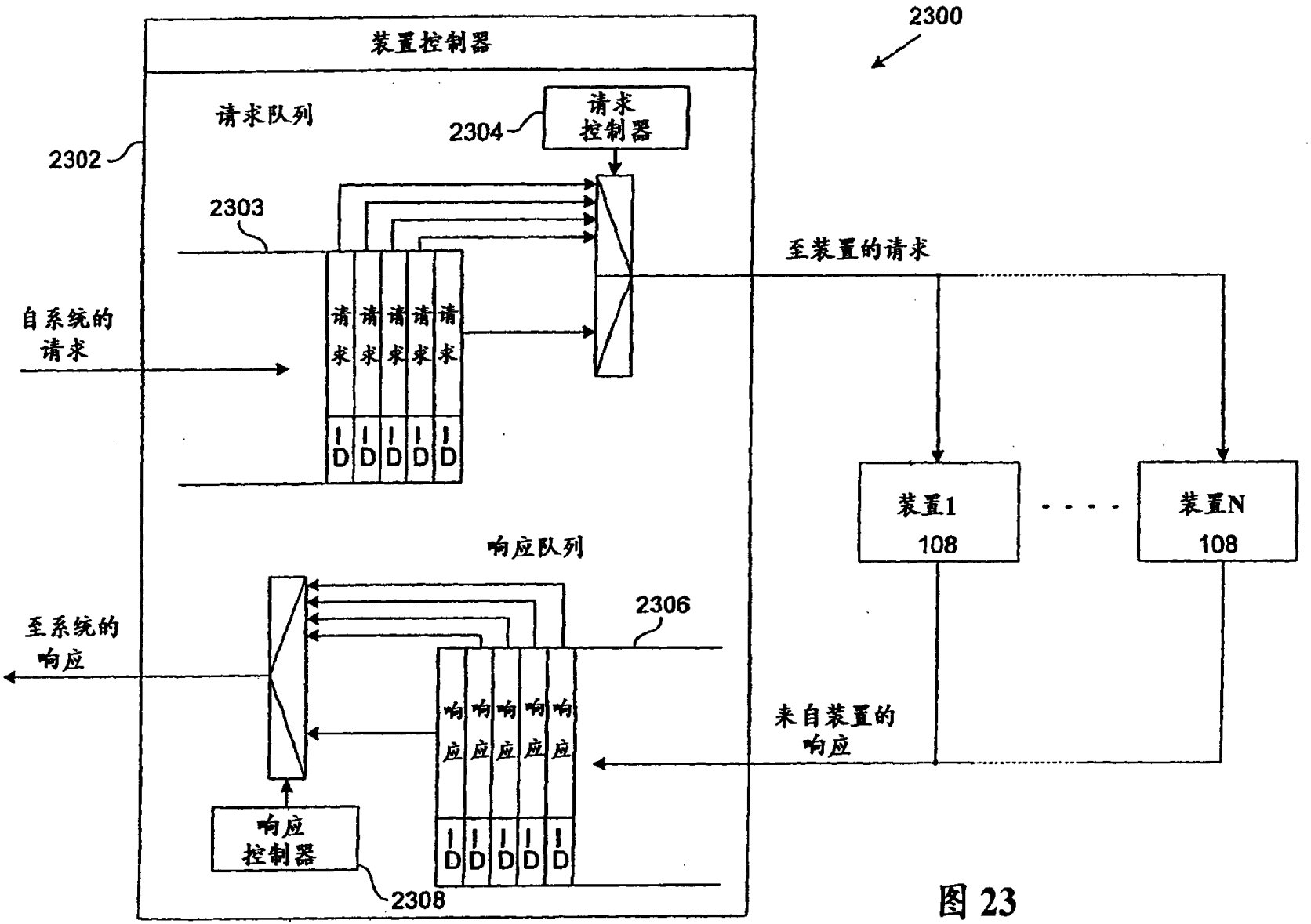

2300
装置控制器
请求队列
2304
请求
控制器
2302
2303
至装置的请求
自系统的
请求
请求
ID
装置1
108
装置N
108
响应队列
2306
至系统的
响应
来自装置的
响应
响应
控制器
2308

图 23

表 4

输入指令	区域		虚拟存储体 0	1	2	3	替换	LUT 输入	ΔT
打开 @ 1000	100	400					0		0
读页 @1000	100	400	100 打开				1	打开 读页 区域0	3ϕ
写页 @1001	100	400	100 读页				2	读页 写页 区域0	2ϕ
打开 @ 1001	10	200	100 写页				3	打开	0
读页 @ 1001	100	400	100 读页			200 打开	0	读页 写页 区域0	1

图 24