(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 105247484 B
(45)授权公告日 2021.02.23

(21)申请号 201480024832.5
(22)申请日 2014.03.12
(65)同一申请的已公布的文献号
申请公布号 CN 105247484 A
(43)申请公布日 2016.01.13
(30)优先权数据
61/800,487 2013.03.15 US
(85)PCT国际申请进入国家阶段日
2015.11.02
(86)PCT国际申请的申请数据
PCT/US2014/024828 2014.03.12
(87)PCT国际申请的公布数据
WO2014/151043 EN 2014.09.25
(73)专利权人 英特尔公司
地址 美国加利福尼亚州
(72)发明人 穆罕默德·阿布达拉
(74)专利代理机构 上海专利商标事务所有限公司 31100
代理人 黄嵩泉
(51)Int.Cl.
G06F 9/455(2006.01)
G06F 9/46(2006.01)
(56)对比文件
CN 101151594 A,2008.03.26
WO 9750031 A1,1997.12.31
US 2012198209 A1,2012.08.02
CN 101151594 A,2008.03.26
审查员 杨华

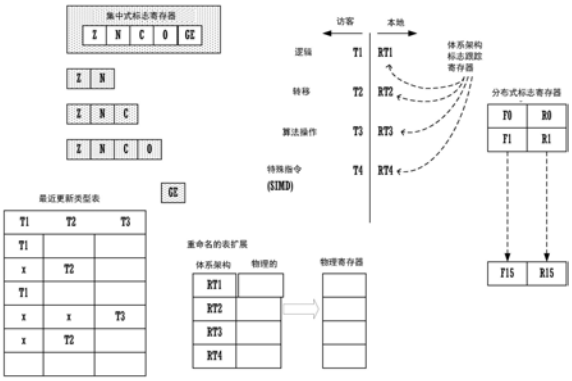
权利要求书3页 说明书14页 附图38页

(54)发明名称

利用本地分布式标志体系架构来仿真访客集中式标志体系架构的方法

(57)摘要

通过利用本地分布式标志体系架构仿真访客集中式标志体系架构的方法，该方法包括：利用全局前端接收进来的指令序列；将所述指令分组以形成指令块，其中每个所述指令块包括两个半块；调度所述指令块的所述指令，以按照调度器执行；以及利用分布式标志体系架构仿真集中式标志体系架构，以对访客指令执行进行仿真。



CN 105247484 B

1.通过利用本地分布式标志体系架构来仿真访客集中式标志体系架构的方法,包括:

利用机器的前端接收进来的指令的指令序列;

将所述指令分组以形成指令块,其中每个所述指令块包括两个半块,其中每个半块被独立地分派;

按照所述机器的调度器来调度指令块的指令以执行;以及

利用分布式标志体系架构来仿真集中式标志体系架构,以对访客指令执行进行仿真,其中所述分布式标志体系架构在多个独立的标志寄存器中分布和更新所述访客指令执行的多个标志,

其中利用分布式标志体系架构来仿真集中式标志体系架构包括:

所述机器的前端/动态转换器基于进来的指令更新访客指令标志的方式对所述进来的指令进行归类;

将指令类型更新它们各自的访客标志的顺序记录在最新更新的类型表数据结构中;

当指令类型到达调度器时,调度器分配与体系架构类型对应的隐含的物理目标并且将该分配记录到重命名/映射表数据结构中;

当后来的访客指令到达调度表中的分配/重命名阶段,并且该指令想要读取访客标志域时,机器判定哪些标志体系架构类型需要被访问以执行读取;以及

从保存最后更新的最新值的物理寄存器中分别读取每个标志,就像被最新更新的标志类型表跟踪一样。

2.根据权利要求1所述的方法,其中所述分布式标志体系架构仿真集中式访客标志体系架构的行为。

3.根据权利要求1所述的方法,其中通过利用多个独立的标志寄存器实现所述分布式标志体系架构。

4.根据权利要求1所述的方法,其中访客指令被归类到4个本地指令类型。

5.根据权利要求1所述的方法,其中访客指令被归类到4个本地指令类型并且访客指令类型基于它们的类型更新不同的访客指令标志。

6.根据权利要求1所述的方法,其中所述机器的前端/动态转换器基于进来的指令更新访客指令标志的方式对所述进来的指令进行归类。

7.具有计算机可读代码的非暂态计算机可读介质,当所述计算机可读代码由计算机系统执行时,使得计算机系统实施利用本地分布式标志体系架构来仿真访客集中式标志体系架构的方法,包括:

利用机器的前端接收进来的指令的指令序列;

将所述指令分组以形成指令块,其中每个所述指令块包括两个半块,其中每个半块被独立地分派;

按照所述机器的调度器来调度指令块的指令以执行;以及

利用分布式标志体系架构来仿真集中式标志体系架构,以对访客指令执行进行仿真,其中所述分布式标志体系架构在多个独立的标志寄存器中分布和更新所述访客指令执行的多个标志,

其中利用分布式标志体系架构来仿真集中式标志体系架构包括:

所述机器的前端/动态转换器基于进来的指令更新访客指令标志的方式对所述进来的

指令进行归类；

将指令类型更新它们各自的访客标志的顺序记录在最新更新的类型表数据结构中；

当指令类型到达调度器时，调度器分配与体系架构类型对应的隐含的物理目标并且将该分配记录到重命名/映射表数据结构中；

当后来的访客指令到达调度表中的分配/重命名阶段，并且该指令想要读取访客标志域时，机器判定哪些标志体系架构类型需要被访问以执行读取；以及

从保存最后更新的最新值的物理寄存器中分别读取每个标志，就像被最新更新的标志类型表跟踪一样。

8.根据权利要求7所述的计算机可读介质，其中所述分布式标志体系架构仿真集中式访客标志体系架构的行为。

9.根据权利要求7所述的计算机可读介质，其中通过利用多个独立的标志寄存器实现所述分布式标志体系架构。

10.根据权利要求7所述的计算机可读介质，其中访客指令被归类到4个本地指令类型。

11.根据权利要求7所述的计算机可读介质，其中访客指令被归类到4个本地指令类型并且访客指令类型基于它们的类型更新不同的访客指令标志。

12.根据权利要求7所述的计算机可读介质，其中所述机器的前端/动态转换器基于它们更新访客指令标志的方式对进来的指令进行归类。

13.具有耦合至存储器的处理器的计算机系统，所述存储器具有计算机可读代码，当所述计算机可读代码由计算机系统执行时，使得计算机系统实现利用本地分布式标志体系架构来仿真访客集中式标志体系架构的方法，包括：

利用机器的前端接收进来的指令的指令序列；

将所述指令分组以形成指令块，其中每个所述指令块包括两个半块，其中每个半块被独立地分派；

按照所述机器的调度器来调度所述指令块的指令以执行；以及

利用分布式标志体系架构来仿真集中式标志体系架构，以对访客指令执行进行仿真，其中所述分布式标志体系架构在多个独立的标志寄存器中分布和更新所述访客指令执行的多个标志，

其中利用分布式标志体系架构来仿真集中式标志体系架构包括：

所述机器的前端/动态转换器基于进来的指令更新访客指令标志的方式对所述进来的指令进行归类；

将指令类型更新它们各自的访客标志的顺序记录在最新更新的类型表数据结构中；

当指令类型到达调度器时，调度器分配与体系架构类型对应的隐含的物理目标并且将该分配记录到重命名/映射表数据结构中；

当后来的访客指令到达调度表中的分配/重命名阶段，并且该指令想要读取访客标志域时，机器判定哪些标志体系架构类型需要被访问以执行读取；以及

从保存最后更新的最新值的物理寄存器中分别读取每个标志，就像被最新更新的标志类型表跟踪一样。

14.根据权利要求13所述的计算机系统，其中所述分布式标志体系架构仿真集中式访客标志体系架构的行为。

15.根据权利要求13所述的计算机系统，其中通过利用多个独立的标志寄存器实现所述分布式标志体系架构。

16.根据权利要求13所述的计算机系统，其中访客指令被归类到4个本地指令类型。

17.根据权利要求13所述的计算机系统，其中访客指令被归类到4个本地指令类型并且访客指令类型基于它们的类型更新不同的访客指令标志。

18.根据权利要求13所述的计算机系统，其中所述机器的前端/动态转换器基于它们更新访客指令标志的方式对进来的指令进行归类。

19.一种计算机实现的系统，包括用于执行如权利要求1-6中的任一项所述的方法的装置。

利用本地分布式标志体系架构来仿真访客集中式标志体系架构的方法

[0001] 本申请要求共同未决的、共同转让的序列号:61/800,487、名称为“A METHOD FOR EMULATING A GUEST CENTRALIZED FLAG ARCHITECTURE BY USING A NATIVE DISTRIBUTED FLAG ARCHITECTURE”、由Mohammad A.Abdallah于2013年3月15日提交的美国临时申请的权益,其全部并入本申请。

[0002] 相关申请交叉引用

[0003] 本申请涉及共同未决的、共同转让的序列号:2009/0113170、名称为“APPARATUS AND METHOD FOR PROCESSING AN INSTRUCTION MATRIX SPECIFYING PARALLEL INDEPENDENT OPERATIONS”、申请人为Mohammad A.Abdallah于2007年4月12日提交的美国专利申请,其全部并入本申请。

[0004] 本申请涉及共同未决的、共同转让的序列号:2010/0161948、名称为“APPARATUS AND METHOD FOR PROCESSING COMPLEX INSTRUCTION FORMATS IN A MULTITHREADED ARCHITECTURE SUPPORTING VARIOUS CONTEXT SWITCH MODES AND VIRTUALIZATION SCHEMES”、申请人为Mohammad A.Abdallah于2007年11月14日提交的美国专利申请,其全部并入本申请。

技术领域

[0005] 本发明主要涉及数字计算机系统,尤其是涉及选择包括指令序列的指令的系统和方法。

背景技术

[0006] 处理器需要处理依赖的或者完全独立的多个任务。这种处理器的内部状态通常由在程序执行的每个特定实例中具有不同值的寄存器组成。在程序执行的每个实例中,内部状态图像被称为处理器的体系体系架构状态。

[0007] 当将代码执行切换到运行另一个函数(例如,另一个线程、进程或程序),则机器/处理器的状态必须进行保存,以便新的函数能够利用内部寄存器建立其新的状态。一旦新的函数终止,则可以丢弃它的状态,并且恢复以前的上下文状态且重新开始执行。这样的转换过程被称为上下文切换并且通常包括数十个或数百个循环,尤其是利用大量的寄存器(例如,64、128、256)和/或乱序执行的现在体系架构。

[0008] 在线程感知硬件体系架构中,对于硬件来说支持有限数量的硬件支持的线程的多个上下文状态是正常的。在这种情况下,对每个支持的线程,硬件复制所有的体系架构状态元素。这消除了执行新线程时上下文切换的需求。然而,这仍然有多个缺陷,即在硬件支持每个附加的线程的面积、功率以及复制所有体系架构状态元素(即寄存器)的复杂性。此外,如果软件线程的数目超过明确支持的硬件线程数目,则仍然必须执行上下文切换。

[0009] 由于在需要大量线程的细粒度的基础上需要并行化,这变得普遍。具有双倍的上下文状态硬件存储的硬件线程感知体系架构不利于非线程软件代码并且对线程化的软件

仅减少上下文切换的次数。然而，那些线程通常为了粗粒度并行化而建造，并且开始和同步导致了大量软件开销，远离细粒度并行化，例如函数调用和循环并行执行，没有有效的线程开始/自动产生。这种所描述的开销伴随着使用art编译器的状态的这样的代码的自动并行化或者用于不明确/易于并行化/线程化的软件代码的用户并行技术的困难。

发明内容

[0010] 在一个实施例中，本发明实现为通过利用本地分布式标志体系架构仿真访客集中式标志体系架构的方法。该方法包括：利用全局前端接收进来的指令序列；将所述指令分组以形成指令块，其中每个所述指令块包括两个半块；调度所述指令块的所述指令，以按照调度器执行；以及利用分布式标志体系架构仿真集中式标志体系架构，以对访客指令执行进行仿真。

[0011] 前述是个概要并且因此必然包含细节的简单化、归纳化和省略。因此，本领域技术人员要领会，该发明内容仅仅是说明性的并且不意图以任何方式限制。其他方面，创造性的特征以及本发明的优点，仅由权利要求限定，在下面详细解释的非限定的详细说明中变得显然。

附图说明

[0012] 本发明以举例的方式进行说明，并且不以限制的方式，随附的附图中的数字及其中相同的引用号涉及相同的元件。

[0013] 图1示出了利用寄存器模板将指令分组成块并且跟踪指令间的依赖关系的过程的总体示意图。

[0014] 图2示出了根据本发明的一个实施例的，寄存器视图、源视图以及指令视图的总体示意图。

[0015] 图3示出了根据本发明的一个实施例的、说明了示范性的寄存器模板以及通过寄存器模板中的信息如何对源视图进行填充的示意图。

[0016] 图4示出了说明源视图内依赖广播的第一实施例的示意图。在这个实施例中，每栏包括指令块。

[0017] 图5示出了说明源视图内依赖广播的第二实施例的示意图。

[0018] 图6示出了根据本发明的一个实施例的、说明了选择准备就绪的指令块用于从提交指针开始的分派以及广播对应的端口分配的示意图。

[0019] 图7示出了根据本发明的一个实施例的、用于实现图6中描述的选择器阵列的加法器树结构。

[0020] 图8示出了更加详细的选择器阵列加法器树的示范性逻辑。

[0021] 图9示出了根据本发明的一个实施例的、用于实现选择器阵列的加法器树的并行实现方式。

[0022] 图10示出了根据本发明的一个实施例的、说明如何利用进位保存加法器实现图9中的加法器X的示范性示意图。

[0023] 图11示出了根据本发明的一个实施例的、掩膜准备好的位用于从提交指针开始调度以及利用选择器阵列加法器的掩膜实施例。

[0024] 图12示出了根据本发明的一个实施例的、如何通过寄存器模板填充寄存器视图条目的总体示意图。

[0025] 图13示出了根据本发明的一个实施例的、减小的寄存器视图占用空间的第一实施例。

[0026] 图14示出了根据本发明的一个实施例的、减小的寄存器占用空间的第二实施例。

[0027] 图15示出了根据本发明的一个实施例的、快照间增量的示范性形式。

[0028] 图16示出了根据本发明的一个实施例的、根据指令块的分配创造寄存器模板快照的过程示意图。

[0029] 图17示出了根据本发明的一个实施例的、根据指令块的分配创造寄存器模板快照的过程的另一个示意图。

[0030] 图18示出了根据本发明的一个实施例的、实现由以前的寄存器模板创造后来的寄存器模板的串行实现方式的硬件的总体示意图。

[0031] 图19示出了根据本发明的一个实施例的、实现由以前的寄存器模板创造后来的寄存器模板的并行实现方式的硬件的总体示意图。

[0032] 图20示出了根据本发明的一个实施例的、用于基于块的指令执行的硬件以及它如何利用源视图、指令视图、寄存器模板以及寄存器视图进行工作的的总体示意图。

[0033] 图21示出了根据本发明的一个实施例的、成块体系架构的例子。

[0034] 图22示出了根据本发明的一个实施例的、如何根据它们的块号和线程ID分配线程的描述。

[0035] 图23示出了根据本发明的一个实施例的，为了管理多线程执行，利用指向物理存储位置的线程指针映射的调度器的实现方式。

[0036] 图24示出了根据本发明的一个实施例的、利用基于线程的指针映射的调度器的另一个实现方式。

[0037] 图25示出了根据本发明的一个实施例的、线程执行资源的动态的基于日历的分配的示意图。

[0038] 图26图解了根据本发明的一个实施例的双分派过程。

[0039] 图27图解了根据本发明的一个实施例的双分派暂态乘法累加。

[0040] 图28图解了根据本发明的一个实施例的体系架构可见状态的双分派的相乘相加。

[0041] 图29示出了根据本发明的一个实施例的、用于在分组的执行单元上执行的指令块的获取和形成过程的总体示意图。

[0042] 图30示出了根据本发明的一个实施例的、指令分组的示范性示意图。在图30的实施例中示出了具有第三辅助操作的两个指令。

[0043] 图31示出了根据本发明的一个实施例的、指令块堆中的半块对如何映射到执行块单元。

[0044] 图32示出了根据本发明的一个实施例的、描述了中间指令块结果存储作为第一级寄存器文件的示意图。

[0045] 图33示出了根据本发明的一个实施例的奇数/偶数端口调度器。

[0046] 图34示出了图33的更加详细的版本，其中示出了四个执行单元，其从调度器阵列接收结果并且将输出结果写入暂存寄存器文件部分。

[0047] 图35示出了根据本发明的一个实施例的、描述了访客标志结构仿真的示意图。

[0048] 图36示出了根据本发明的一个实施例的、说明了机器的前端、调度器以及执行单元以及集中式标志寄存器的示意图。

[0049] 图37示出了由本发明的实施例实现的集中式标志寄存器仿真过程的示意图。

[0050] 图38示出了仿真访客设置中的集中化的标志寄存器表现的过程3800的步骤流程图。

具体实施方式

[0051] 虽然结合一个实施例描述本发明，但并不意图将本发明限制为此处详述的具体形式。相反地，本发明意图覆盖根据附加的权利要求限定的、可以合理包含在本发明范围内的这样的替代、修改和等同。

[0052] 在下面的详细说明中，详细解释了大量的具体细节，例如具体的方法次序、结构、元件以及连接方式。应理解，无论如何，不需要利用这些以及其他具体细节实践本发明的实施例。在其他情况下，为了避免不必要地使本说明难以理解，公知的结构、元素或连接方式被省略或没有特别详细描述。

[0053] 说明书中涉及到的“一个实施例”或“实施例”意图表示连同实施例描述的特定的特征、结构或特点包括在本发明的至少一个实施例中。短语“在一个实施例中”在说明书中不同位置的表现并不一定全部涉及相同的实施例，也并不是与其他实施例互斥的单独的或可替代的实施例。而且，描述了在一些实施例中展示并且在其他实施例中未展示的多个特征。类似地，描述了一些实施例要求但其他实施例不要求的多个要求。

[0054] 详细说明的一些部分，其在下面跟随，以程序、步骤、逻辑块、过程以及对计算机内存中的数据位操作的其他符号表现形式展示。这些说明以及表现是数据处理领域技术人员用于将他们的工作内容更加有效地传递给本领域其他技术人员的手段。程序、计算机执行的步骤、逻辑块、过程等，在此处，并且一般被设想为步骤的有条理的顺序或者导向想要的结果的指令。步骤就是要求物理量的物理操作的那些。通常，尽管不是必须的，这些物理量采取计算机可读存储介质的电或磁信号的形式并且能够被存储、转移、组合、比较，并且否则在计算机系统中被操控。主要由于通用性，已经证实有时作为位、值、元件、符号、特点、术语、号码等等提及这些信号。

[0055] 然而，应该在思维中具有所有这些以及类似的术语都与合适的物理量相关联，并且仅仅是用于这些物理量的便签。除非特别说明，否则如下面的讨论很显然，应领会，遍及本发明，利用术语例如“处理”或“访问”或“写入”或“存储”或“复制”等的讨论涉及计算机系统或类似的电子计算设备的动作或过程，类似的电子计算设备将计算机系统的寄存器以及内存以及其他计算机可读介质中表现为物理(电子)量的数据操纵并转换成类似地表现为计算机系统内存或寄存器或其他这样的信息存储、传输或显示设备中的物理量的其他数据。

[0056] 图1示出了利用寄存器模板将指令分组成块并且跟踪指令间的依赖关系的过程的总体示意图。

[0057] 图1示出了具有块头和块正文的指令块。该指令块由一组指令创造。该指令块包括封装该组指令的实体。在微处理器的本实施例中，抽象度被提升到指令块而不是单个指令。

处理指令块用于分派而不是单个指令。每个指令块用块号作标记。机器的乱序管理工作因此被大大简化。一个关键特征是找到管理要处理的大量指令的方法，而不大量增加机器的管理开销。

[0058] 本发明的实施例通过执行指令块、寄存器模板和继承向量实现了这个目标。在图1示出的指令块中，指令块的标题列出并概述了指令块的指令的所有源和目标以及那些源来自何处(例如，来自哪个指令块)。该标题包括更新寄存器模板的目标。包括在标题中的源将由存储在寄存器模板中的块号联系在一起。

[0059] 乱序处理的指令数目决定了乱序机器的管理复杂程度。更多的乱序指令导致更大的复杂程度。源需要在处理器的乱序调度窗口中与先前指令的目标进行对比。

[0060] 如图1所示，寄存器模板具有针对每个寄存器的从R0到R63的字段。各指令块将它们各自的块号写入与指令块目标对应的寄存器模板字段。每个指令块从寄存器模板中读取表示它的寄存器源的寄存器字段。当指令块退隐(retire)并将它的目标寄存器内容写入寄存器文件时，它的块号从寄存器模板中擦除。这意味着那些寄存器可以作为来自寄存器文件本身的源读取。

[0061] 在本实施例中，无论何时分配指令块，寄存器模板都在机器的每个循环进行更新。因为产生了新的模板更新，寄存器模板的先前的快照被存储在阵列(例如，图2中示出的寄存器视图)中，每个指令块一个阵列。该信息被保留，直到对应的块退隐。这允许机器从误预测中恢复并且非常快速地刷新(例如，通过获得最后已知的依赖状态)。

[0062] 在一个实施例中，在寄存器视图中存储的寄存器模板可以通过仅存储连续快照间的增量(快照间增加的变化)而进行压缩(因此节约存储空间)。用这种方法，机器得到缩小的寄存器视图。进一步的压缩可以通过仅存储具有分支指令的指令块的模板而得到。

[0063] 如果需要恢复点而不是分支误预测，则可以首先在分支恢复点得到恢复，随后可以脱离分配指令(但不执行它们)而重建状态，直到该机器在恢复点之后完成寻找。

[0064] 应注意到，在一个实施例中，此处使用的术语“寄存器模板”与在早期提交的共同转让的、由Mohammad Abdallah于2012年3月23日提交的专利申请“EXECUTING INSTRUCTION SEQUENCE CODE BLOCKS BY USING VIRTUAL CORES INSTANTIATED BY PARTITIONABLE ENGINES”中描述的术语“继承向量”是同义词，其全部并入此处。

[0065] 图2示出了根据本发明的一个实施例的，寄存器视图、源视图以及指令视图的总体示意图。此图示出了调度器体系架构(例如，具有源视图、指令视图、寄存器视图等)的一个实施例。通过组合或分离一个或多个上述引用的结构实现相同功能的调度器体系架构的其他实现方式是可能的。

[0066] 图2图解了支持寄存器模板操作和机器状态保持的功能实体。图2的左侧示出了寄存器模板T0到T4，箭头表示从一个寄存器模板/继承向量到另一个寄存器模板/继承的信息的继承。寄存器视图、源视图以及指令视图的每个都包括用于存储关于指令块信息的数据结构。图2也示出了具有块头的示范性的指令块以及指令块如何包括机器的寄存器的源和目标。关于由指令块引用的寄存器的信息被存储在寄存器视图数据结构中。关于由指令块引用的源的信息被存储在源视图数据结构中。关于由指令块引用的指令自身的信息被存储在指令视图数据结构中。寄存器模板/继承向量自身包括存储了由指令块引用的依赖和继承信息的数据结构。

[0067] 图3示出了根据本发明的一个实施例的、说明了示范性的寄存器模板以及通过寄存器模板中的信息如何对源视图进行填充的示意图。

[0068] 在本实施例中，应注意到，源视图的目标是确定何时可以分派特定的指令块。当指令块被分派时，它向所有其余指令块广播它的块号。其他指令块(例如，比较)的源的任何匹配使得设置就绪位(例如，或者一些其他类型的指示器)。当设置了所有就绪位(例如，“与”门)，准备分派指令块。基于它们依赖的其他指令块的准备就绪分派指令块。

[0069] 当多个指令块准备好被分派，先于较新的指令块选择最旧的指令块进行分派。例如，在一个实施例中，找到第一电路可以用于基于与提交指针的接近度找到最旧的指令块并且基于与提交指针的相对接近度找到下一个指令块(例如，根据每个指令块的就绪位工作)。

[0070] 仍然参考图3，在这个例子中，在到达指令块20的检查时创造寄存器模板快照。如上所述，寄存器模板具有每个寄存器从R0到R63的字段。指令块将它们各自的块号写入对应于指令块目标的寄存器模板字段中。每个指令块从寄存器模板中读取表示它的寄存器源的寄存器字段。第一个数字是写入寄存器的指令块并且第二个数字是该指令块的目标数字。

[0071] 例如，当到达指令块20，它读取寄存器模板的快照并且在寄存器模板中查找它自己的寄存器源，以确定写入它的每个源的最新的指令块，并且根据其目标对之前的寄存器模板快照进行的更新来填充源视图，下一个指令块将用它们自己的目标更新寄存器模板。这在图3的底部左侧示出，在此指令块20填充它的源：源1、源2、源3，一直到源8。

[0072] 图4示出了说明了源视图内依赖广播的第一实施例的示意图。在此实施例中，每栏包括指令块。当指令块被分派时，它在它的源对块有依赖关系的所有那些块的栏中(例如，通过写0)进行掩膜。当任何其他指令块被分派时，它的块号跨与该指令块有关的确切的栏被广播。应注意到，写入1是默认值，表示那个指令块上没有依赖关系。

[0073] 当指令块中的所有就绪位准备就绪，该块被分派并且它的块号向所有其余的指令块广播。该块号与存储在其他指令块的源中的所有号码进行比较。如果存在匹配，则设置该源的就绪位。例如，如果在源1上广播的块号等于11，则设置指令块20的源1的就绪位。

[0074] 图5示出了说明了源视图内依赖广播的第二实施例的示意图。相对于由指令块组织，该实施例由源组织。这由贯穿源视图数据结构的源S1直到S8示出。以类似于上面图4描述的方式，在图5的实施例中，当指令块中的所有就绪位准备就绪，该指令块被分派并且它的块号向所有其余的指令块广播。该块号与存储在其他指令块的源中的所有号码进行比较。如果存在匹配，则设置该源的就绪位。例如，如果在源1上广播的块号等于11，则设置指令块20的源1的就绪位。

[0075] 图5的实施例还示出了该比较如何仅在提交指针和分配指针间的指令块上进行。所有其他指令块都无效。

[0076] 图6示出了根据本发明的一个实施例的、说明了选择准备就绪的指令块用于从提交指针开始的分派以及广播对应的端口分配的示意图。源视图数据结构在图6的左侧示出。指令视图数据结构在图6的右侧示出。选择器阵列在源视图和指令视图之间示出。在这个实施例中，选择器阵列通过四个分派端口P1到P4在每个循环中分派四个指令块。

[0077] 如上所述，用于从提交指针卷绕到分配指针选择指令块用于分派(例如，尝试首先分派较旧的指令块)。选择器阵列用于找到从提交指针开始的最早准备好的四个指令块。可

取的是分派最早准备好的指令块。在一个实施例中，选择器阵列可以通过利用加法树结构实现。这将在下面的图7中说明。

[0078] 图6还示出了选择器阵列如何耦合至穿过指令视图中的条目的四个端口的每一个。在这个实施例中，随着端口使能的端口耦合，使得四个端口中的一个被激活，并且针对该指令视图条目，向下穿过到达分派端口以及到达执行单元上。此外，如上所述，被分派的指令块通过源视图往回广播。用于分派的选择的指令块的块号往回广播(达到4个)。这在图6的更右侧示出。

[0079] 图7示出了根据本发明的一个实施例的、用于实现图6中描述的选择器阵列的加法器树结构。描述的加法器树实现了选择器阵列功能。加法器树选出最早的四个准备好的指令块并且将它们置于四个可用端口用于分派(例如，读取端口1直到端口4)。不使用仲裁。用于具体地使能具体端口的实际的逻辑在条目1中明确示出。为了清楚，该逻辑没有在其他条目中具体示出。用这种方式，图7示出了直接选择每个特定端口用于指令块分派是如何实现的一个具体实施例。应注意到，然而，可替代地，可以实现使用优先编码器的实施例。

[0080] 图8更加详细地示出了选择器阵列加法器树的示范性逻辑。在图8的实施例中，该逻辑示出了范围超过位。该范围超过位保证至多选择四个指令块用于分派。如果第五个指令块准备就绪，则如果最早的四个也准备就绪的话，范围超过位将不允许它被分派。应注意到，在顺序实现中，总数位S0到S3都用于使分派端口以及向下个加法器阶段的传递生效。

[0081] 图9示出了根据本发明的一个实施例的、用于实现选择器阵列的加法器树的并行实现方式。并行实现方式不会将总数从每个加法器转送到下一个。在并行实现方式中，每个加法器利用多输入加法实现方式直接使用它的必要的输入，例如，多输入进位保存加法器树。例如，加法器“X”总计所有之前的输入。为了执行更快的计算时间(例如，单循环)，这个并行实现方式是可取的。

[0082] 图10示出了根据本发明的一个实施例的、说明如何利用进位保存加法器实现图9中的加法器X的示范性示意图。图10示出了在单个循环中可以添加32个输入的结构。该结构利用4×2进位保存加法器组合在一起。

[0083] 图11示出了根据本发明的一个实施例的掩膜实施例，用于掩膜准备好的位用于从提交指针开始调度以及利用选择器阵列加法器。在这个实现方式中，选择器阵列加法器尝试选择最早的四个准备好的指令块，用于从提交指针潜在地卷绕到分配指针开始分派。在这个实现方式中，使用了多输入并行加法器。此外，在这个实现方式中利用了这些循环缓冲器的源。

[0084] 图11示出了准备好的位如何与两个掩膜的每个(个别地或单独地)一起进行“与”操作并且并行地用于两个加法器树。利用两个加法器树选择最早的4个指令块并且与其阈值进行比较。“X”标记表示“对于该加法器树从选择器阵列中排除”，因此“X”值为0。另一方面，“Y”标记表示“对于该加法器树包含在选择器阵列中”，因此“Y”值为1。

[0085] 图12示出了根据本发明的一个实施例的、如何通过寄存器模板填充寄存器视图条目的总体示意图。

[0086] 如上所述，寄存器视图条目由寄存器模板填充。寄存器视图顺序存储了每个指令块的寄存器模板的快照。当推断失效时(例如，分支误预测)，在无效推断点之前寄存器视图具有最新有效的快照。通过读取寄存器视图条目并且将其加载到寄存器模板的底层，机器

可以回滚它的状态到最后有效的快照。寄存器视图的每个条目示出所有寄存器继承状态。例如在图12的实施例中，如果指令块F的寄存器视图失效，机器状态可以回滚到较早的最后有效的寄存器模板快照。

[0087] 图13示出了根据本发明的一个实施例的、减小的寄存器视图占用空间的第一实施例。需要存储寄存器视图条目的内存数量可以通过仅存储包含分支指令的那些寄存器视图模板快照而减少。当出现例外(例如，推断失效、分支误预测等)时，可以根据发生在例外之前的分支指令重建最后有效的快照。为了建立最后有效的快照，从例外之前直到例外处的分支取指。指令被取出但不执行它们。如图13中所示，仅仅那些包括分支指令的快照被保存在减小的寄存器视图中。这大大减少了需要存储寄存器模板快照的内存数量。

[0088] 图14示出了根据本发明的一个实施例的、减小的寄存器视图占用空间的第二实施例。需要存储寄存器视图条目的内存数量可以通过仅存储快照(例如，每四个快照中的一个)的顺序子集而减少。利用比存储全部连续的快照相对较小数量的内存，可以将连续的快照间的变化存储为原始快照的“变量”。当出现例外(例如，推断失效、分支误预测等)时，可以根据发生在例外之前的原始快照重建最后有效的快照。发生在例外之前的原始快照的“变量”以及连续的快照被用于重建最后有效的快照。最初的起始状态可以累加变量以到达需要的快照状态。

[0089] 图15示出了根据本发明的一个实施例的、快照间增量的示范性形式。图15示出了原始快照和两个变量。在一个变量中，仅R5和R6是由B3更新的寄存器。其余条目没有变化。在另一个变量中，仅R1和R7是由B2更新的寄存器。其余条目没有变化。

[0090] 图16示出了根据本发明的一个实施例的、根据指令块的分配创造寄存器模板快照的过程示意图。在这个实施例中，图16的左侧示出了两个解复用器并且在图16的顶端是快照寄存器模板。图16示出了由以前的寄存器模板(例如，串行实现方式)创造后来的寄存器模板的示意图。

[0091] 这个串行实现方式示出了如何根据指令块的分配创造寄存器模板快照。那些快照用于捕获最新的寄存器体系架构状态更新，其用于依赖跟踪(例如，在图1中通过4描述的)以及更新寄存器视图以操纵误预测/例外(例如，图12中通过15描述的)。

[0092] 通过选择哪个到来的源通过而实施解复用。例如，寄存器R2将在第二输出上解复用到1，而R8将在第七输出上解复用到1等等。

[0093] 图17示出了根据本发明的一个实施例的、根据指令块的分配创造寄存器模板快照的过程的另一个示意图。图17的实施例还示出了由以前的寄存器模板创造后来的寄存器模板。图17的实施例还示出了寄存器模板块继承的例子。此图示出了寄存器模板如何根据分配的块号进行更新。例如，块Bf更新R2、R8和R10。Bg更新R1和R9。圆点箭头表示该值从以前的快照继承。这个过程一直进行，直到块Bi。因此，例如，由于没有快照更新寄存器R7，它的原始值Bb将一直传送下去。

[0094] 图18示出了根据本发明的一个实施例的、实现由以前的寄存器模板创造后来的寄存器模板的串行实现方式的硬件的总体示意图。解复用器用于控制两个输入多路复用器的串联，其两个块号将传送到下个阶段。它可以或者是来自以前阶段的块号，或者是当前块号。

[0095] 图19示出了根据本发明的一个实施例的、实现由以前的寄存器模板创造后来的寄

存器模板的并行实现方式的硬件的总体示意图。此并行实现方式利用专门的编码的多路复用器控制由以前的寄存器模板创造后来的寄存器模板。

[0096] 图20示出了根据本发明的一个实施例的、用于基于块的指令执行的硬件以及它如何利用源视图、指令视图、寄存器模板以及寄存器视图进行工作的的总体示意图。

[0097] 在这个实现方式中，分派器中的分配器、调度器接收机器的前端取来的指令。这些指令用我们早期描述的方法完成指令块形式。如早期描述的，指令块生成寄存器模板并且这些寄存器模板用于填充寄存器视图。从源视图中，源被转移到寄存器文件层，并且用上面描述的方法广播回源视图。指令视图将指令转移到执行单元。指令由执行单元执行，同时指令需要的源来自寄存器文件层。然后这些执行的指令被转移出执行单元并回到寄存器文件层。

[0098] 图21示出了根据本发明的一个实施例的、成块结构的例子。成块的重要性在于通过利用示出的四个多路复用器器将到每个调度器条目的写入端口数量从4减少到1，而仍然密集封装所有条目，没有形成气泡。

[0099] 成块的重要性可以通过下面的例子看到(注意到，每个循环中块的分配在顶端位置开始，此案中为B0)。假如在循环1中，三个指令块被分配给调度器条目(例如，三个指令块将占据调度器中的前3个条目)。在下个循环(例如，循环2)中，另两个指令块被分配。为了避免在调度器阵列条目中产生气泡，调度器阵列条目必须建立对四个写入端口的支持。这从能量消耗、时间、面积等来看是昂贵的。在分配给阵列之前使用多路复用器结构，上面的成块结构将所有调度器阵列简化成仅有一个写入端口。在上面的例子中，循环2中的B0将由最后的多路复用器选择，而循环2中的B1将由第一个多路复用器选择(例如，从左到右进行)。

[0100] 用这种方法，条目块的每个仅需要一个每条目写入端口并且四个每条目读取端口。由于必须实现多路复用器，有开销上的权衡，无论如何，由于可能存在很多条目，因此该开销通过不必实现四个每条目写入端口而被多次弥补。

[0101] 图21还示出了中间分配缓冲器。如果调度器阵列不能接受所有送到它们的块，则它们可以暂时存储在中间分配缓冲器中。当调度器阵列有空闲空间，该块将从中间分配缓冲器转移到调度器阵列。

[0102] 图22示出了根据本发明的一个实施例的、如何根据它们的块号和线程ID分配线程的描述。通过上面描述的成块实现方式，块被分配到调度器阵列。每个线程块利用块号在它们之中维持顺序的秩序。来自不同线程的块能够被交叉存取(例如，线程Th1的块和线程Th2的块在调度器阵列中交叉存取)。用这种方法，来自不同线程的块呈现在调度器阵列中。

[0103] 图23示出了根据本发明的一个实施例的，为了管理多线程执行，利用指向物理存储位置的线程指针映射的调度器的实现方式。在这个实施例中，线程的管理通过线程映射的控制实现。例如，此处的图23示出了线程1映射和线程2映射。该映射跟踪各个线程的块的位置。映射中的条目被分配到属于该线程的块。在这种实现方式中，每个线程具有分配计数器，其为两个线程计数。总的计数不能超过N/2(例如，超过可用空间)。为了在总的条目分配中实现公平，分配计数器具有可调节的阈值。分配计数器可以阻止一个线程利用所有可用空间。

[0104] 图24示出了根据本发明的一个实施例的、利用基于线程的指针映射的调度器的另一个实现方式。图24示出了提交指针和分配指针之间的关系。如所示，每个线程具有提交指

针和分配指针，箭头示出了线程2的真实指针如何卷绕物理存储分配块B1和B2，但它不能分配块B9，直到线程2的提交指针向下移动。这通过线程2的提交指针的位置和删除线示出。图24的右侧示出了当提交指针沿逆时针方向移动时，块的分配和提交指针之间的关系。

[0105] 图25示出了根据本发明的一个实施例的、线程执行资源的动态基于日历的分配的示意图。基于每个线程的向前进度，可以利用分配计数器动态控制公平。如果两个线程都有实质的向前进度，则两个分配计数器都设置为相同的阈值(例如，9)。然而，如果一个线程的向前进度缓慢，例如遭受了L2高速缓存缺失或这样的事件，则阈值计数器的比例可以根据仍然有实质向前进度的线程进行调节。如果一个线程被搁置或暂停(例如，处于等待或等待OS或IO回应的自旋状态)，该比例可以完全调节到另一个线程，除了为暂停的线程预留的单个返回条目以信号告知释放等待状态。

[0106] 在一个实施例中，过程从50％:50％的比例开始。根据在块22上的L2高速缓存缺失检测，管线前端推迟任何进一步的取出到管线中或分配到线程2块的调度器中。当线程2块从调度器中退出时，那些条目将对于线程1分配可用，直到在该点处完成线程分配的新的动态比例。例如，离开最近退出的线程2块的3个将返回池中用于分配，以用线程1而不是线程2，使得线程1与线程2的比例为75％:25％。

[0107] 应注意到，在管线前端的线程2块的推迟可以要求从管线前端冲刷那些块，如果没有硬件装置对其进行旁路(例如，通过线程1块经过被推迟的线程2块)。

[0108] 图26图解了根据本发明的一个实施例的双分派过程。多分派一般包括多次分派块(其中具有多个指令)，以便不同的指令块能够在每次经过执行单元时执行。一个例子是地址计算指令的分派由消耗结果数据的后续的分派跟随。另一个例子将是浮点操作，其中第一部分作为固定点操作执行并且执行第二部分，以通过执行取整、标志生成/计算、指数调节等来完成操作。块作为单个实体被自动分配、提交并且退回。

[0109] 多分派的主要益处在于它避免了将多个单独的块分配到机器窗口，因此，使得机器窗口有效增大。更大的机器窗口意味着更多的优化和重新排序的机会。

[0110] 现在看图26的左下部，描述了指令块。该块不能在单个循环中被分派，因为从高速缓存/内存中加载地址计算和加载返回数据之间存在延迟。该块被第一次分派，它的中间结果作为暂态被保留(它的结果在运行中被传递到第二分派并对于结构状态不可见)。第一分派发送在地址计算以及LA分派中使用的两个元件1和2。第二分派发送元件3和4，其是根据从高速缓存/内存中加载返回数据的加载数据的执行部分。

[0111] 现在看图26的右下部，描述了浮点相乘累加操作。在这种情况中，其中硬件没有足够的进入源带宽以在单个阶段分派操作，则使用双分派，如相乘累加图中示出的。如所示，第一分派是固定点相乘。如所示，第二分派是浮点加法取整。当这些被分派的指令都执行时，它们有效地执行了浮点相乘/累加。

[0112] 图27图解了根据本发明的一个实施例的双分派暂态相乘累加。如图27所示，第一分派是整数32位相乘，并且第二分派是整数累积相加。在第一分派和第二分派(相乘的结果)之间通信的状态是暂态并且体系架构上不可见。在一种实现方式中暂态存储可以保留多于一个乘数的结果并且能够给它们加标签，以识别对应的相乘累加对，因此，允许多个相乘累加对混合并以任意方式被分派(例如，交叉存取等)。

[0113] 注意到，其他指令可以将这个相同的硬件用于它们的实现方式(例如，浮点等)。

[0114] 图28图解了根据本发明的一个实施例的体系架构可见状态的双分派的相乘相加。第一分派是单精确的相乘，并且第二分派是单精确的相加。在此实现方式中，在第一分派和第二分派(例如，相乘的结果)之间通信的状态信息体系架构上可见，因为此存储是体系架构状态寄存器。

[0115] 图29示出了根据本发明的一个实施例的、用于在分组的执行单元过程上执行的指令块的获取和形成的总体示意图。本发明的实施例利用通过硬件或动态转换器/JIT获取指令并形成指令块的过程。指令块中的指令被组织，其结果是指令块中较早的指令提供指令块中后来指令的源。这在指令块中由圆点箭头示出。这个特性使得指令块在执行块的堆积的执行单元上有效执行。指令也可以被分组，即使它们可以并行执行，例如，如果它们分享相同的源(未在此图中明确示出)。

[0116] 在硬件中形成指令块的一个可替代方式是在软件中形成它们(静态地或在运行时)，在其中形成指令对、三个一组、四个一组等。

[0117] 指令分组功能的其他实现方式可以在共同转让的美国专利8,327,115中找到。

[0118] 图30示出了根据本发明的一个实施例的、指令分组的示范性示意图。在图30的实施例中示出了具有第三辅助操作的两个指令。图31的左侧指令块包括上半块/1槽和下半块/1槽。从顶端下来的垂直箭头表示进入块中的源，而从底端向下的垂直箭头表示回到内存的目标。从图3左侧开始向右侧行进，说明了可能的不同的指令组合。在这种实现方式中，每个半块可以接收三个源并且可以传递两个目标。OP1和OP2是正常操作。辅助的Op是辅助操作，例如逻辑、转移、移动、记号扩展、分支等。将指令块分成两个半块的好处是允许每个半块自己独立地分派或者否则的话基于(或者为了端口利用，或者由于资源约束)依赖方案动态的结合成一个块，因此，能够更好地利用执行时间，与此同时两个半块与一个块对应允许机器转移两个半块的复杂性，以像一个块一样管理(即分配和退回处)。

[0119] 图31示出了根据本发明的一个实施例的、指令块堆中的半块对如何映射到执行块单元。如在执行块上示出的，每个执行块具有两个槽—槽1和槽2。目标是将块映射到执行单元上，使得第一半块在槽1上执行并且第二半块在槽2上执行。目标是如果每个半块的指令组不依赖于另一个半块，则允许两个半块独立分派。从顶端开始进入执行块的成对的箭头是两个32位字的源。离开执行块向下的成对箭头是两个32位字的目标。图31从左到右，示出了指令的不同的示范性组合，其能够被堆叠到执行块单元上。

[0120] 图31的顶端概述了半块对如何在整块上下文中或任何半块的上下文中执行。每个s执行块具有两个槽/半块并且半块/执行槽的每一个执行或者单个、一对或三个一组的操作。有四种类型的块执行类型。第一个是并行半块(一旦每个半块自己的源准备好，其允许每个半块独立执行，但是如果两个半块同时准备好，两个半块仍能够作为一个块在一个执行单元上执行)。第二个是原子并行半块(其涉及可以并行执行的半块，因为两个半块间没有依赖关系，但它们被迫作为一个块一起执行，因为在两个半块之间共享的源使得两个半块首选地或必须地在每个执行块可用的源的约束下原子级一起执行)。第三个类型是原子串行半块s(其要求第一半块通过具有或没有中间存储的临时转送将数据转送给第二半块)。第四种类型是顺序半块(如在双分派中)，其中第二个半块依赖于第一个半块并且在比第一个靠后的循环中被分派，并且通过为依赖方案跟踪的外部存储转送数据，这类似于双分派情况。

[0121] 图32示出了根据本发明的一个实施例的、描述了中间指令块结果存储作为第一级寄存器文件的示意图。每组寄存器代表指令块(代表两个半块),其中32位结果以及64位结果都能够通过使用两个32位寄存器得到支持,以支持一个64位寄存器。每块存储假设了虚拟块存储,其意味着来自不同块的两个半块可以写入相同的虚拟块存储中。两个半块的组合的结果存储组成一个虚拟块存储。

[0122] 图33示出了根据本发明的一个实施例的奇数/偶数端口调度器。在这个实现方式中,结果存储是不对称的。一些结果存储是每半块三个64位结果寄存器,而其他是每半块一个64位结果寄存器,然而,可替代的实现方式可以利用每半块对称的存储,并且此外也可以如图32中描述的利用64位和32位分区。在这些实施例中,和按照每块分配相比存储按照每半块分配。这个实现方式通过使用它们如奇数或偶数减少了分派所需的端口数目。

[0123] 图34示出了图33的更加详细的版本,其中示出了四个执行单元,其从调度器阵列接收结果并且将输出写入临时寄存器文件部分。端口以偶数以及奇数间隔附接。调度阵列的左侧示出了块号并且右侧示出了半块号。

[0124] 每个内核具有进入调度阵列的偶数和奇数端口,其中每个端口与奇数或偶数的半块位置相连。在一种实现方式中,偶数端口和与它们对应的半块可以与奇数端口和与它们对应的半块处于不同的内核中。在另一种实现方式中,奇数和偶数端口将在此图示出的多个不同内核间分配。如名称为"EXECUTING INSTRUCTION SEQUENCE CODE BLOCKS BY USING VIRTUAL CORES INSTANTIATED BY PARTITIONABLE ENGINES"、由Mohammad Abdallah于2012年3月23日提交、序列号为13428440、其在此处全部并入本申请的之前更早提交的普通转让专利申请所描述的,内核可以是物理内核或虚拟内核。

[0125] 在块的某一类型中,块的一半可以独立于块的另一半被分派。在块的其他类型中,块的两个半块都需要同时被分派到相同的执行块单元。仍然在块的其他类型中,块的两个半块需要顺序分派(第二个半块在第一个半块之后)。

[0126] 图35示出了根据本发明的一个实施例的、描述了访客标志体系架构仿真的示意图。图35的左侧示出了具有五个标志的集中式标志寄存器。图35的右侧示出了具有分布式标志寄存器的分布式标志体系架构,其中标志在寄存器自身之中分配。

[0127] 在体系架构仿真期间,对于分布式标志体系架构,必须对集中式访客标志体系架构的行为进行仿真。分布式标志体系架构也可以通过利用与关联于数据寄存器的标志域相对的多个独立标志寄存器实现。例如,数据寄存器可以实现为R0到R15,而独立的标志寄存器可以实现为F0到F3。在这种情况中的那些标志寄存器不直接与数据寄存器关联。

[0128] 图36示出了根据本发明的一个实施例的、说明了机器的前端、调度器以及执行单元以及集中式标志寄存器的示意图。在此实现方式中,前端基于它们更新访客指令标志的方式对进来的指令进行归类。在一个实施例中,访客指令被归类成4本地指令类型T1、T2、T3和T4。T1-T4是表示每个访客指令类型更新哪些标志域的指令类型。访客指令类型基于它们的类型更新不同的访客指令标志。例如,逻辑访客指令更新T1本地指令。

[0129] 图37示出了由本发明的实施例实现的集中式标志寄存器仿真过程的示意图。图37中的参与者包括最新更新的类型表、重命名的表扩展、物理寄存器以及分布式标志寄存器。现在图37由图38的流程图描述。

[0130] 图38示出了在访客设置中仿真集中式标志寄存器表现的过程3800的步骤的流程

图。

[0131] 在步骤3801中，前端/动态转换器(硬件或软件)基于它们更新访客指令标志的方式对进来的指令进行归类。在一个实施例中，访客指令被归类到四个标志体系架构类型T1、T2、T3和T4中。T1-T4是表示每个访客指令类型更新哪些标志域的指令类型。访客指令类型基于它们的类型更新不同的访客指令标志。例如，逻辑访客指令更新T1类型标志，转移访客指令更新T2类型标志，算法访客指令更新T3类型标志以及特殊访客指令更新T4类型标志。应注意，访客指令可以是体系架构指令表示，而本地指令可以在机器内部执行(例如，微码)。替代地，访客指令可以是来自仿真体系架构(例如，x86、java、ARM码等)的指令。

[0132] 在步骤3802中，那些指令类型更新它们各自的访客标志的顺序记录在最新更新的类型表数据结构中。在一个实施例中，此动作由机器的前端执行。

[0133] 在步骤3803中，当那些指令类型到达调度器(按顺序的分配/重命名阶段部分)时，调度器分配与体系架构类型对应的隐含的物理目标并且将此分配记录到重命名/映射表数据结构中。

[0134] 并且在步骤3804中，当随后的访客指令到达调度器中的分配/重命名阶段，并且该指令想要读取访客标志域时，(a)机器判定哪些标志体系架构类型需要被访问以执行读取。(b)如果发现所有需要的标志都在相同的最近更新的标志类型中(例如，如由最近更新的类型表判定)，则读取对应的物理寄存器(例如，其映射到最近标志类型)以得到需要的标志。(c)如果发现所有需要的标志不在相同的最近更新的标志类型中，则需要从映射到个别最近更新的标志类型的对应的物理寄存器中读取每个标志。

[0135] 并且在步骤3805中，每个标志从保存最后更新的最新值的物理寄存器中分别读取，就像被最新更新的标志类型表跟踪一样。

[0136] 应注意到，如果最新更新类型包括另一个类型，则所有子集类型必须映射到超级集合类型的相同的物理寄存器中。

[0137] 在退回处，目标标志域和复制的集中式/访客标志体系架构寄存器结合在一起。应注意到，由于本地体系架构利用与单一寄存器集中式标志体系架构相对的分布式标志体系架构的事实而执行复制。

[0138] 更新某一标志类型的指令的例子：

[0139] CF,OF,SF,ZR-算法指令和加载/写入标志指令

[0140] SF,ZF和条件CF-逻辑和转移

[0141] SF,ZF-移动/加载,EXTR,一些乘法

[0142] ZF-POPCNT和STREX[P]

[0143] GE-SIMD指令???

[0144] 读取某些标志的条件/预测的例子：

[0145] 0000EQ Equal Z＝＝1

[0146] 0001NE Not equal,or Unordered Z＝＝0

[0147] 0010CS b Carry set,Greater than or equal,or Unordered C＝＝1

[0148] 0011CC c Carry clear,Less than C＝＝0

[0149] 0100MI Minus,negative,Less than N＝＝1

[0150] 0101PL Plus,Positive or zero,Greater than or equal to,Unordered N＝＝

0011OVS Overflow,Unordered V＝＝1

[0151] 0111VC No overflow,Not unordered V＝＝0

[0152] 1000HI Unsigned higher,Greater than,Unordered C＝＝1and Z＝＝0

[0153] 1001LS Unsigned lower or same,Less than or equal C＝＝0or Z＝＝1

[0154] 1010GE Signed greater than or equal,Greater than or equal N＝＝V

[0155] 1011LT Signed less than,Less than,Unordered N!＝V

[0156] 1100GT Signed greater than,Greater than Z＝＝0and N＝＝V

[0157] 1101LE Signed less than or equal,Less than or equal,Unordered Z＝＝1or N!＝V

[0158] 1110None(AL),Always(unconditional),Any flag set to any value

[0159] 前面的说明,为了解释的目的,已经参考具体实施例进行了描述。然而,上面说明性的讨论并不意图详尽或限制本发明为公开的精确形式。根据上面的教导,许多改进以及变化是可能的。选择并且描述实施例是为了更好地解释本发明的原理和它的实际应用,以当适于特定用途考虑时,因此使得本领域技术人员更好地利用本发明以及具有不同改进的多个实施例。

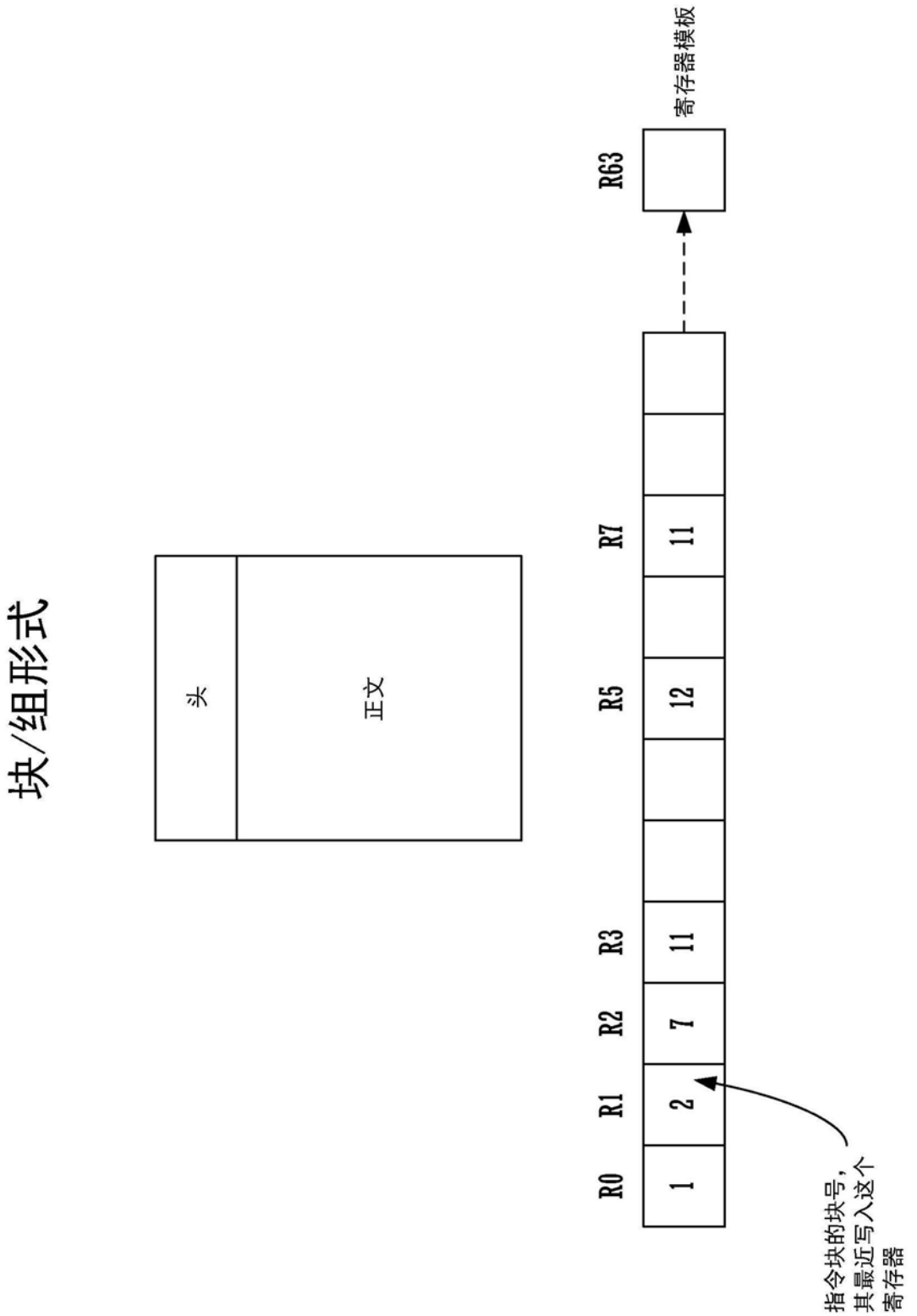

块/组形式
头
正文
寄存器模板
R63
R7
R5
R3
R2
R1
R0
11
12
11
7
2
1
指令块的块号，
其最近写入这个
寄存器

图1

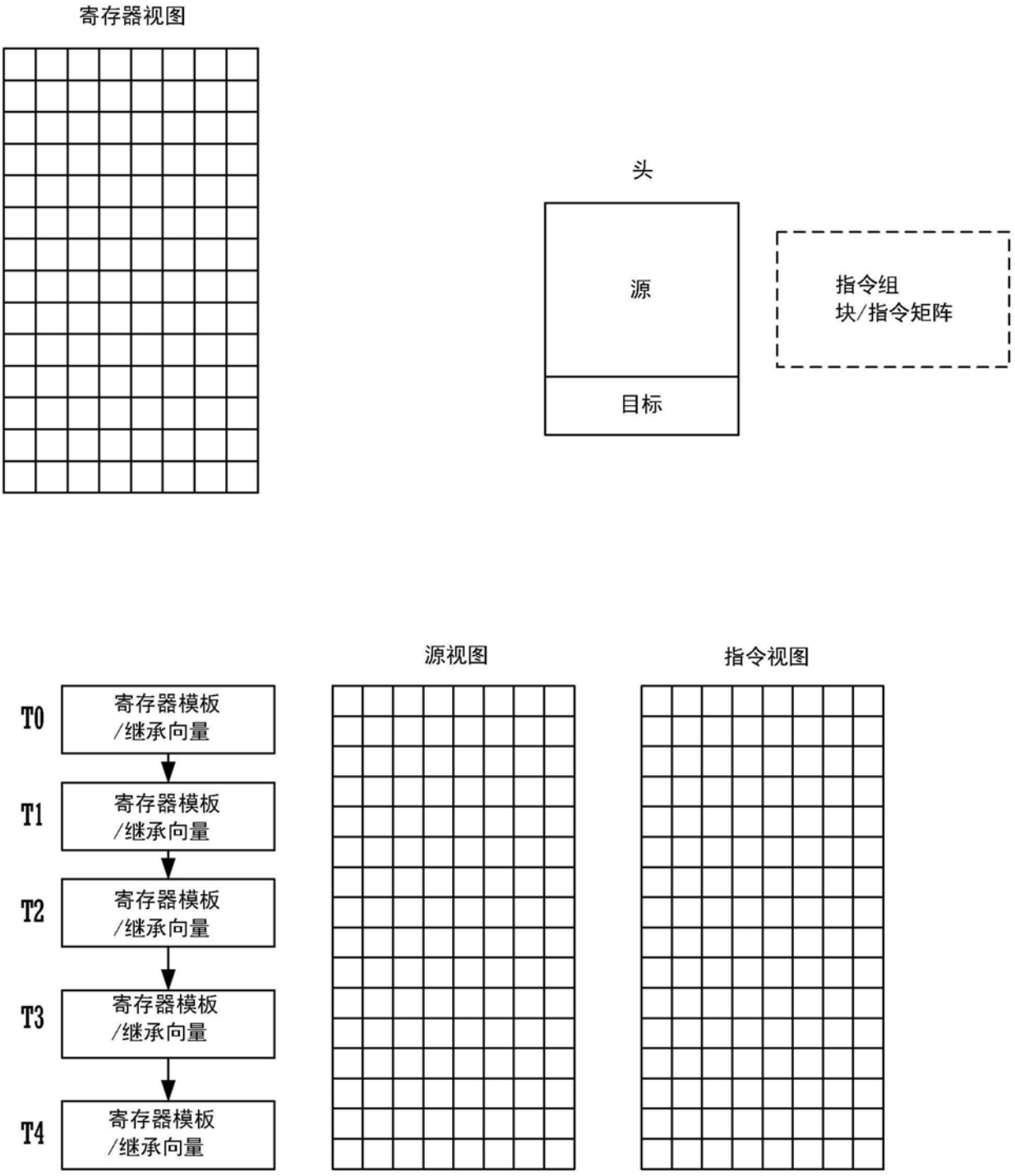

寄存器视图
头
源
目标
指令组
块/指令矩阵
源视图
指令视图
T0
寄存器模板
/继承向量
T1
寄存器模板
/继承向量
T2
寄存器模板
/继承向量
T3
寄存器模板
/继承向量
T4
寄存器模板
/继承向量

图2

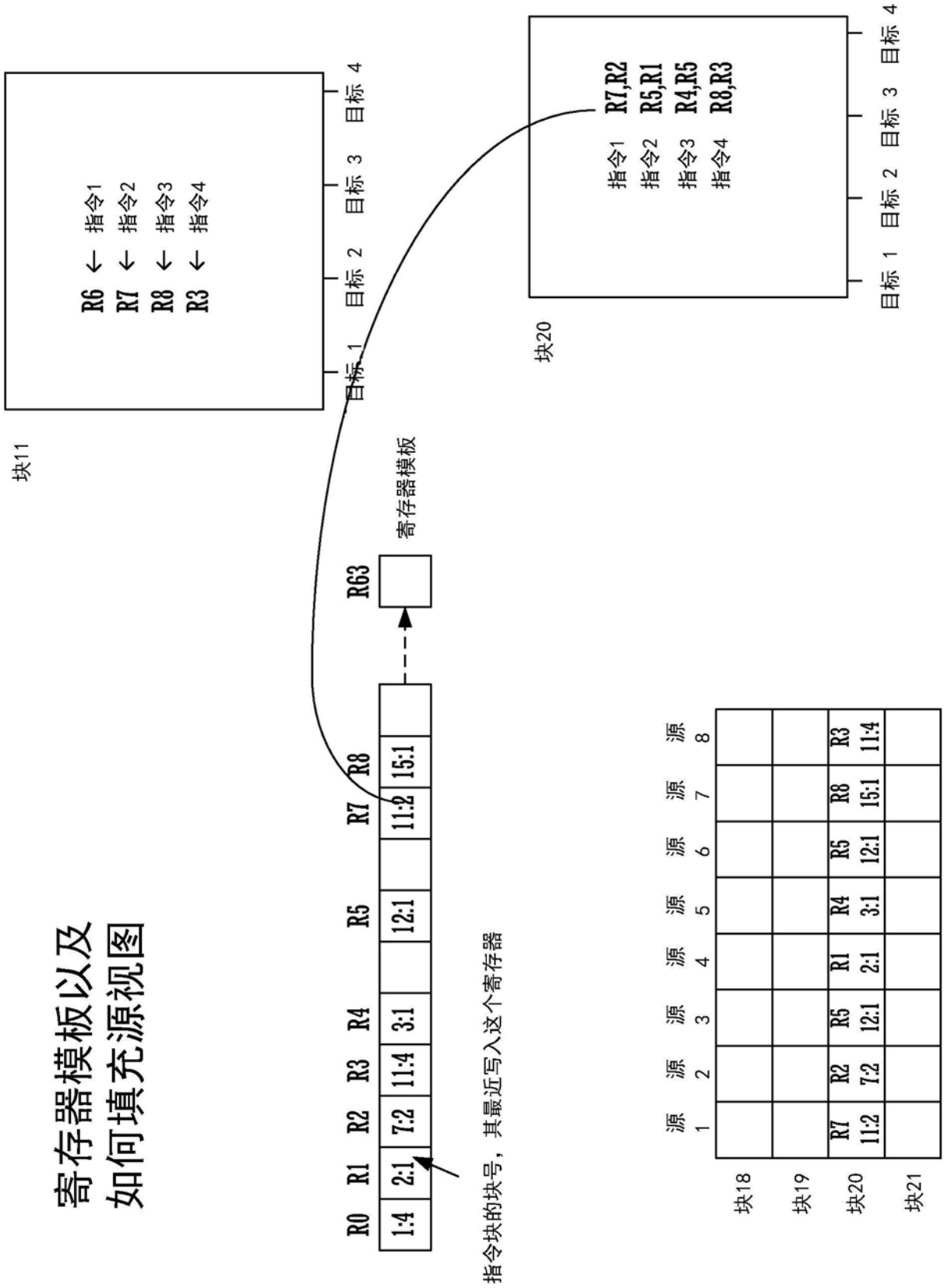

寄存器模板以及
如何填充源视图
块11
R6 ← 指令1
R7 ← 指令2
R8 ← 指令3
R3 ← 指令4
目标 1 目标 2 目标 3 目标 4
块20
指令1 R7,R2
指令2 R5,R1
指令3 R4,R5
指令4 R8,R3
目标 1 目标 2 目标 3 目标 4
寄存器模板
R0 R1 R2 R3 R4 R5 R7 R8 R63
1:4 2:1 7:2 11:4 3:1 12:1 11:2 15:1
指令块的块号，其最近写入这个寄存器
源 1 源 2 源 3 源 4 源 5 源 6 源 7 源 8
块18
块19
块20 R7 11:2 R2 7:2 R5 12:1 R1 2:1 R4 3:1 R5 12:1 R8 15:1 R3 11:4
块21

图3

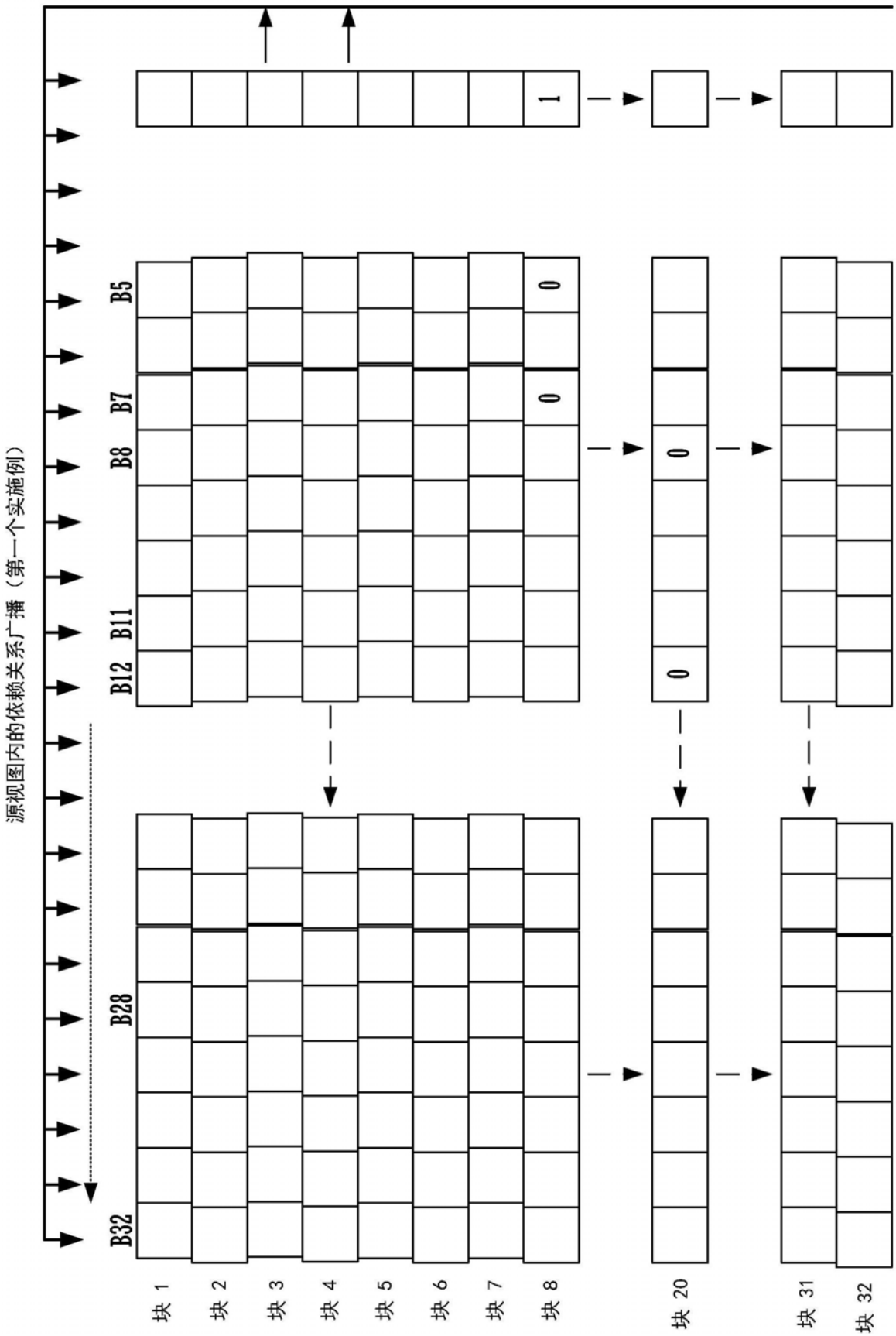

源视图内的依赖关系广播（第一个实施例）
B32
B28
B12
B11
B8
B7
B5
1
0
0
0
0
块 1
块 2
块 3
块 4
块 5
块 6
块 7
块 8
块 20
块 31
块 32

图4

源视图内的分派广播（第二个实施例）

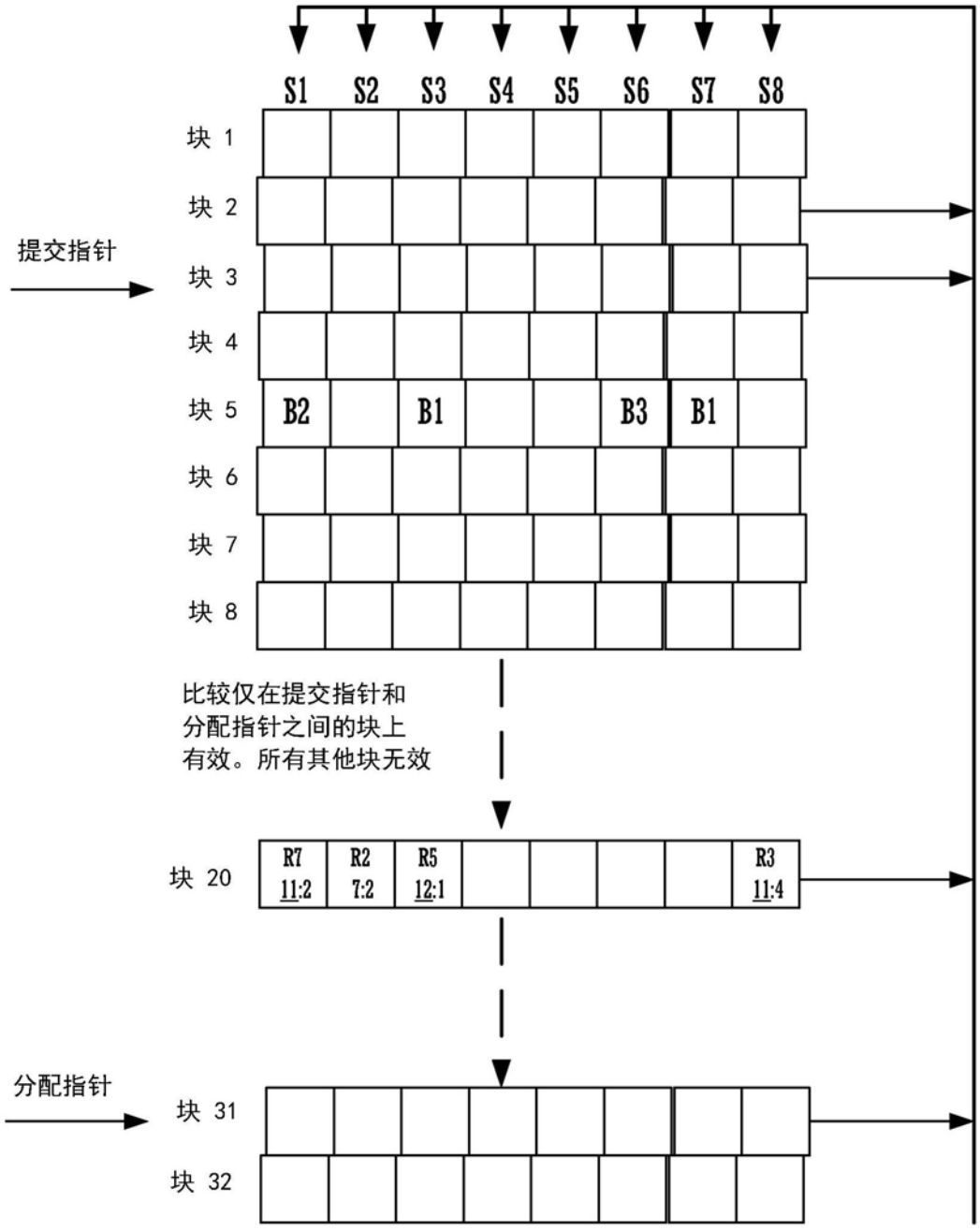


图5

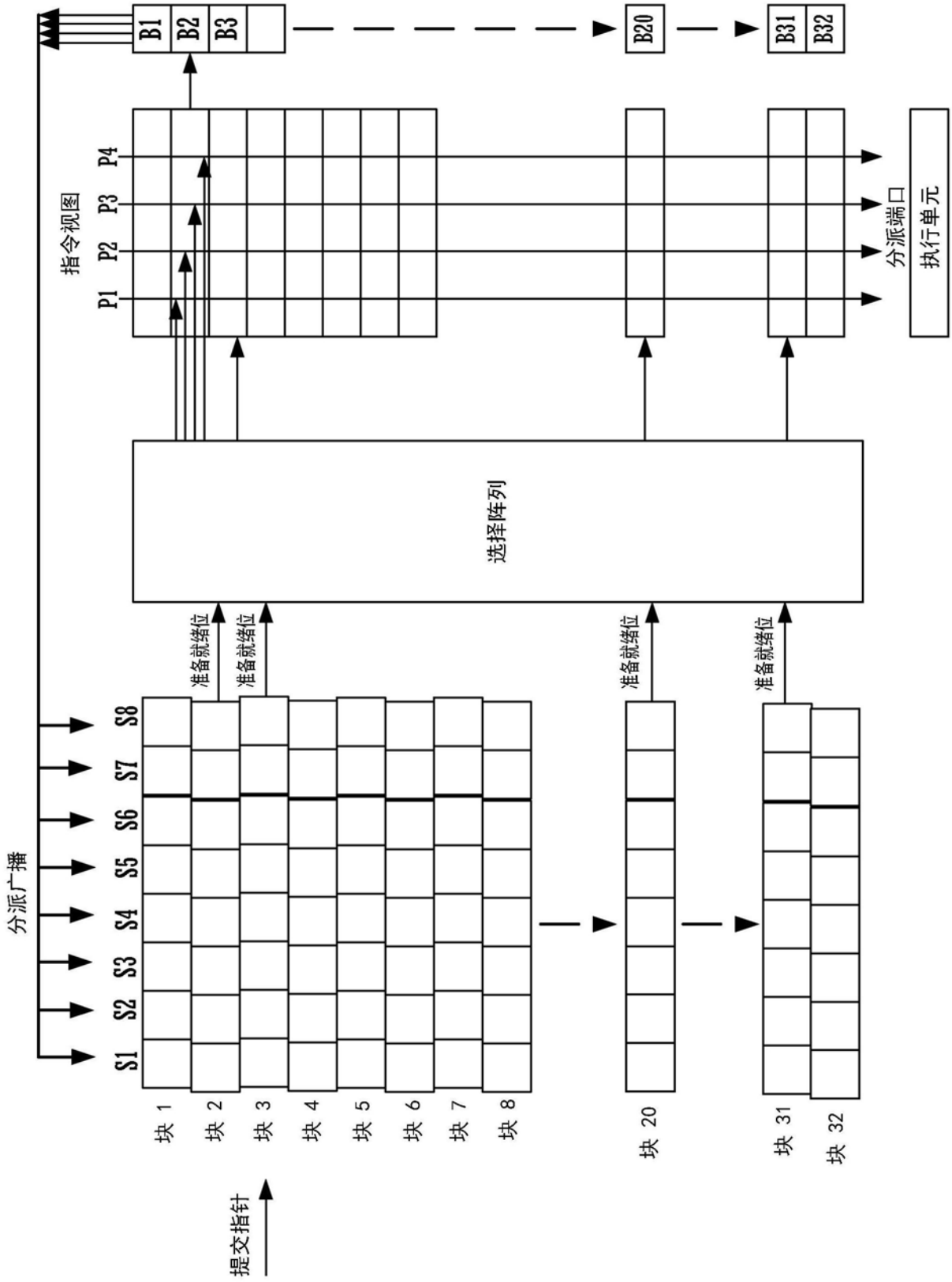

B1
B2
B3
B20
B31
B32
指令视图
P1
P2
P3
P4
分派端口
执行单元
选择阵列
准备就绪位
准备就绪位
准备就绪位
准备就绪位
分派广播
S1
S2
S3
S4
S5
S6
S7
S8
块 1
块 2
块 3
块 4
块 5
块 6
块 7
块 8
块 20
块 31
块 32
提交指针

图6

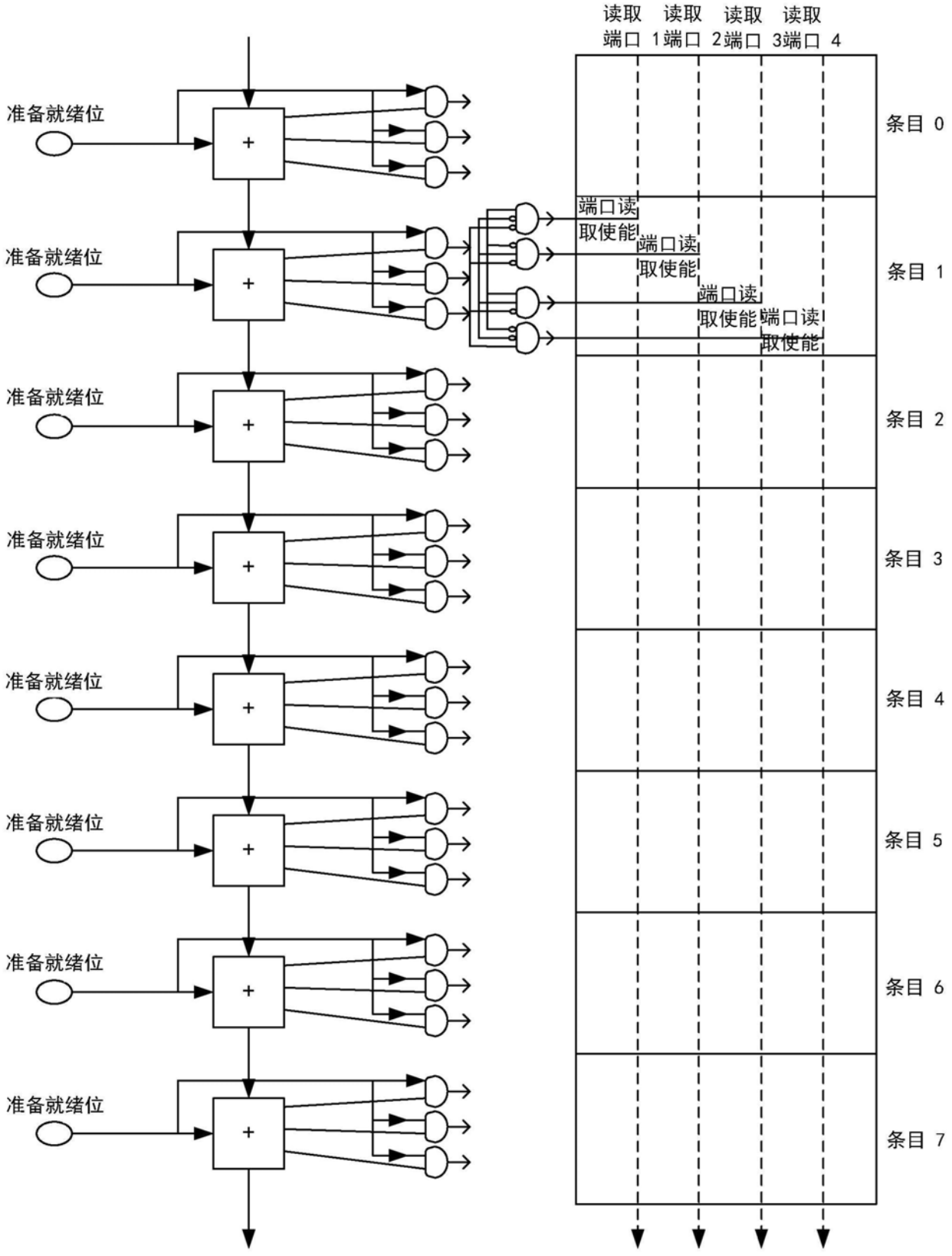

读取端口
读取1端口
读取2端口
读取3端口
4
准备就绪位
端口读取使能
条目 0
条目 1
条目 2
条目 3
条目 4
条目 5
条目 6
条目 7

图7

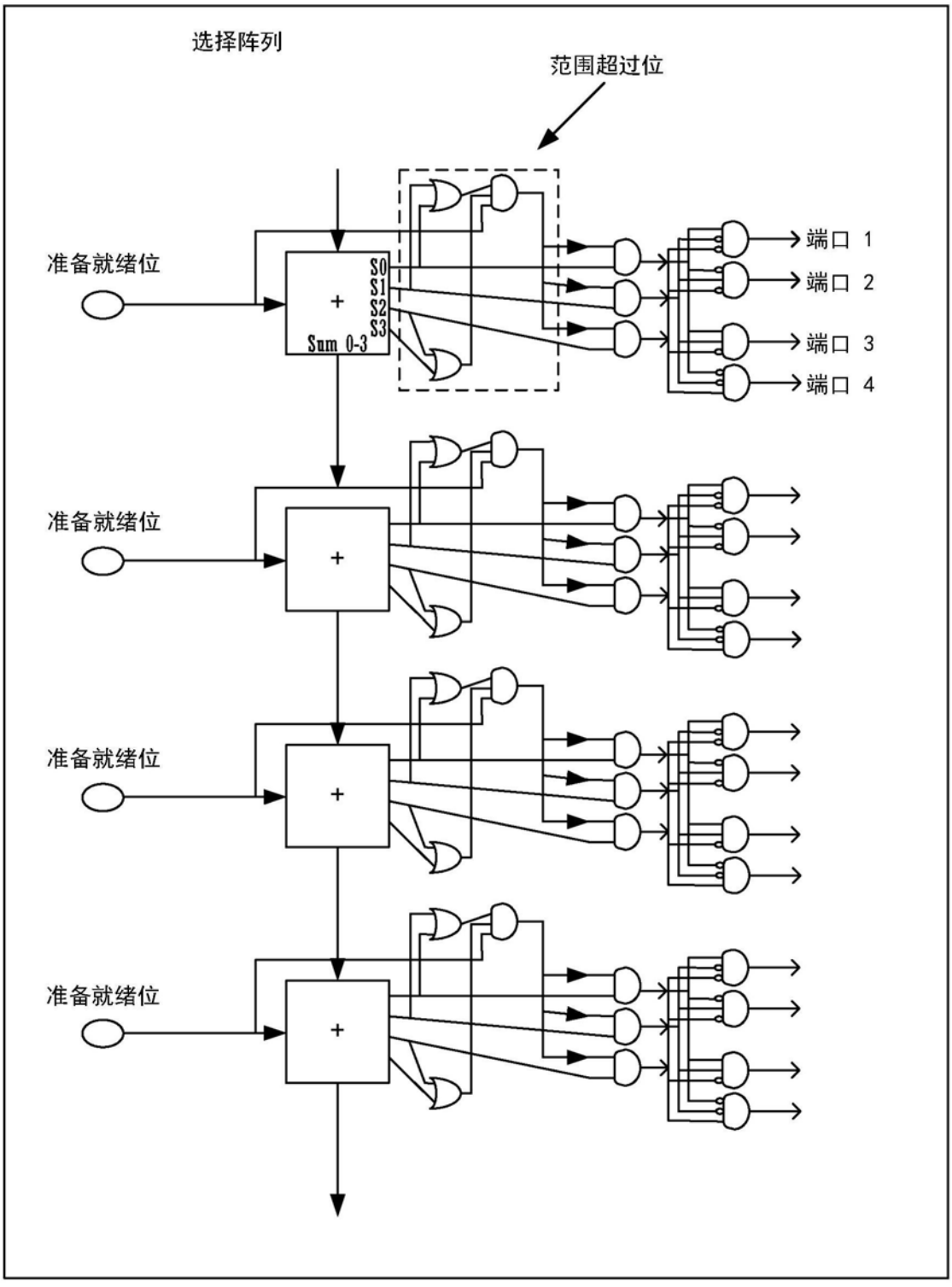

具有范围超过位的每个特定端口的直接选择
选择阵列
范围超过位
准备就绪位
S0
S1
S2
S3
Sum 0-3
端口 1
端口 2
端口 3
端口 4
准备就绪位
准备就绪位
准备就绪位

图8

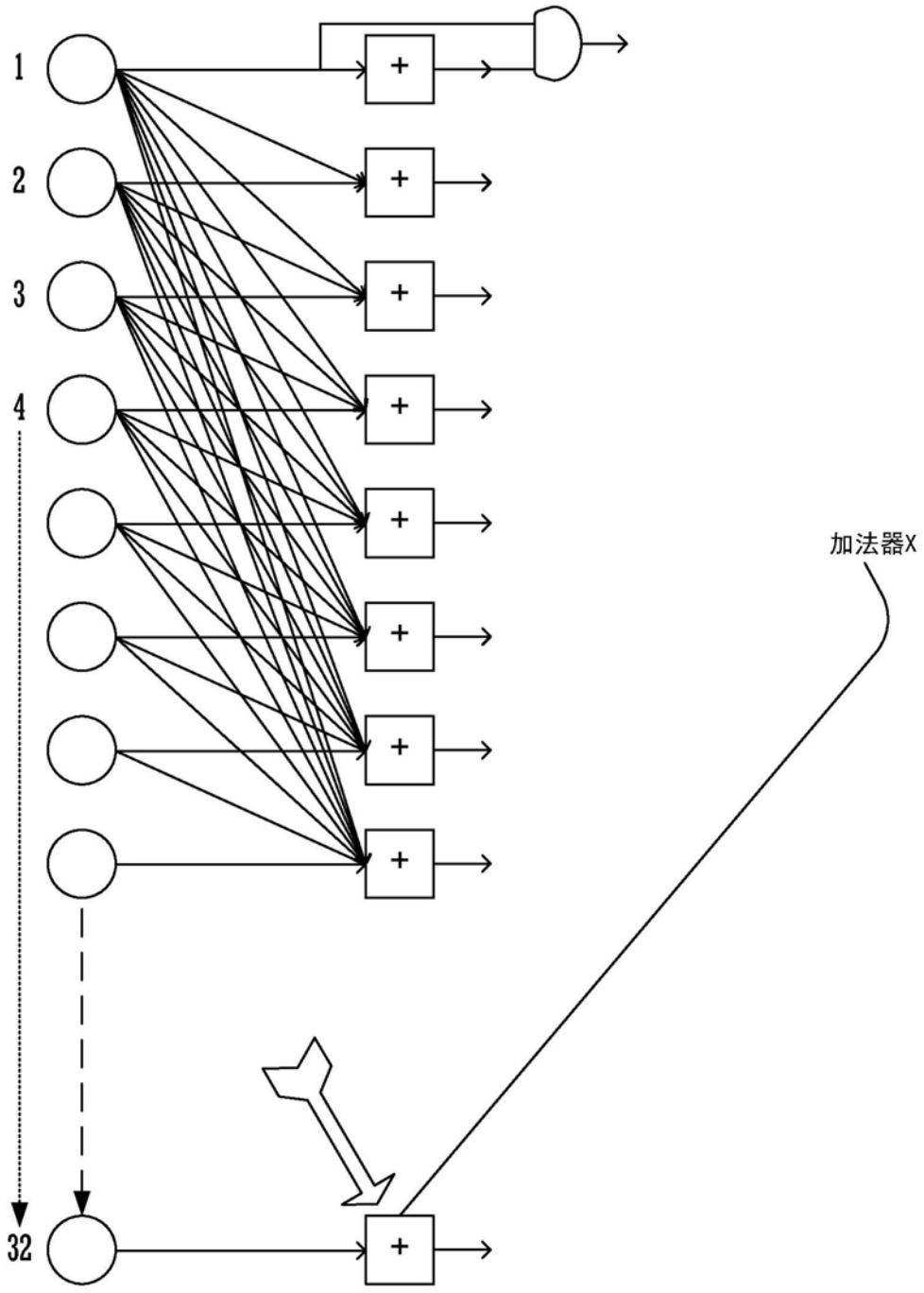


图9

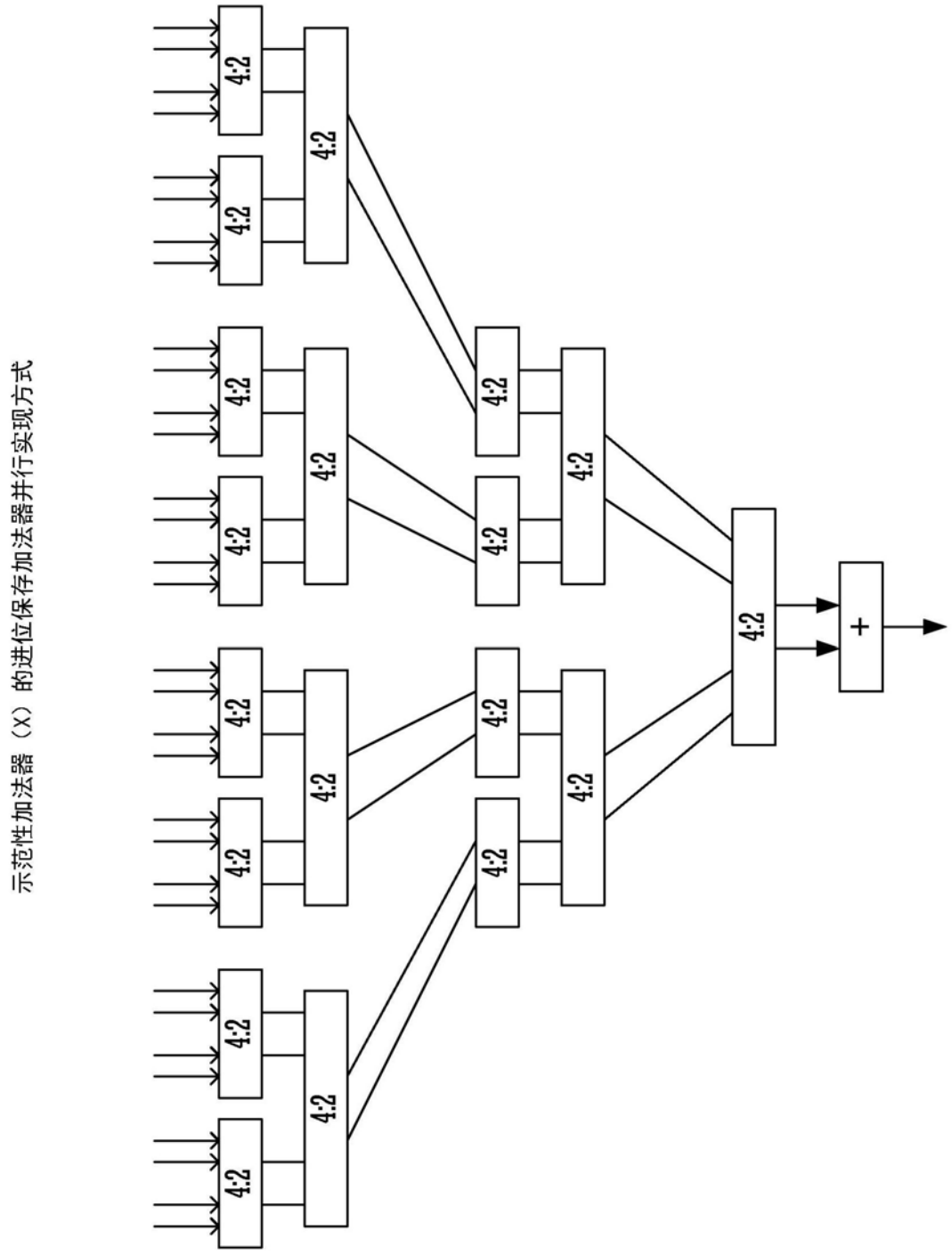


图10

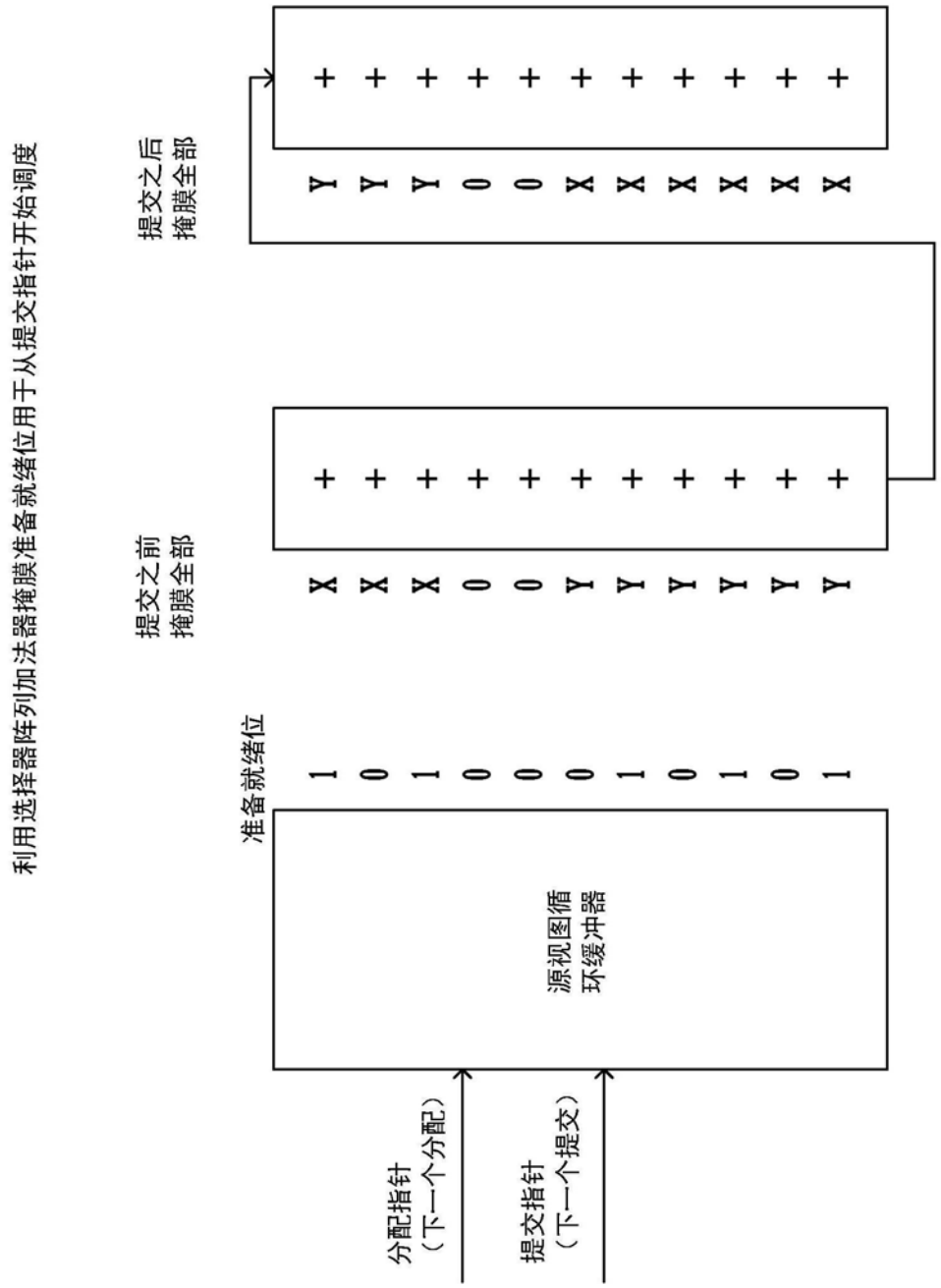

利用选择器阵列加法器掩膜准备就绪位用于从提交指针开始调度
提交之后
掩膜全部
提交之前
掩膜全部
准备就绪位
源视图循
环缓冲器
分配指针
（下一个分配）
提交指针
（下一个提交）

图11

由寄存器模板填充的寄存器视图条目

寄存器视图

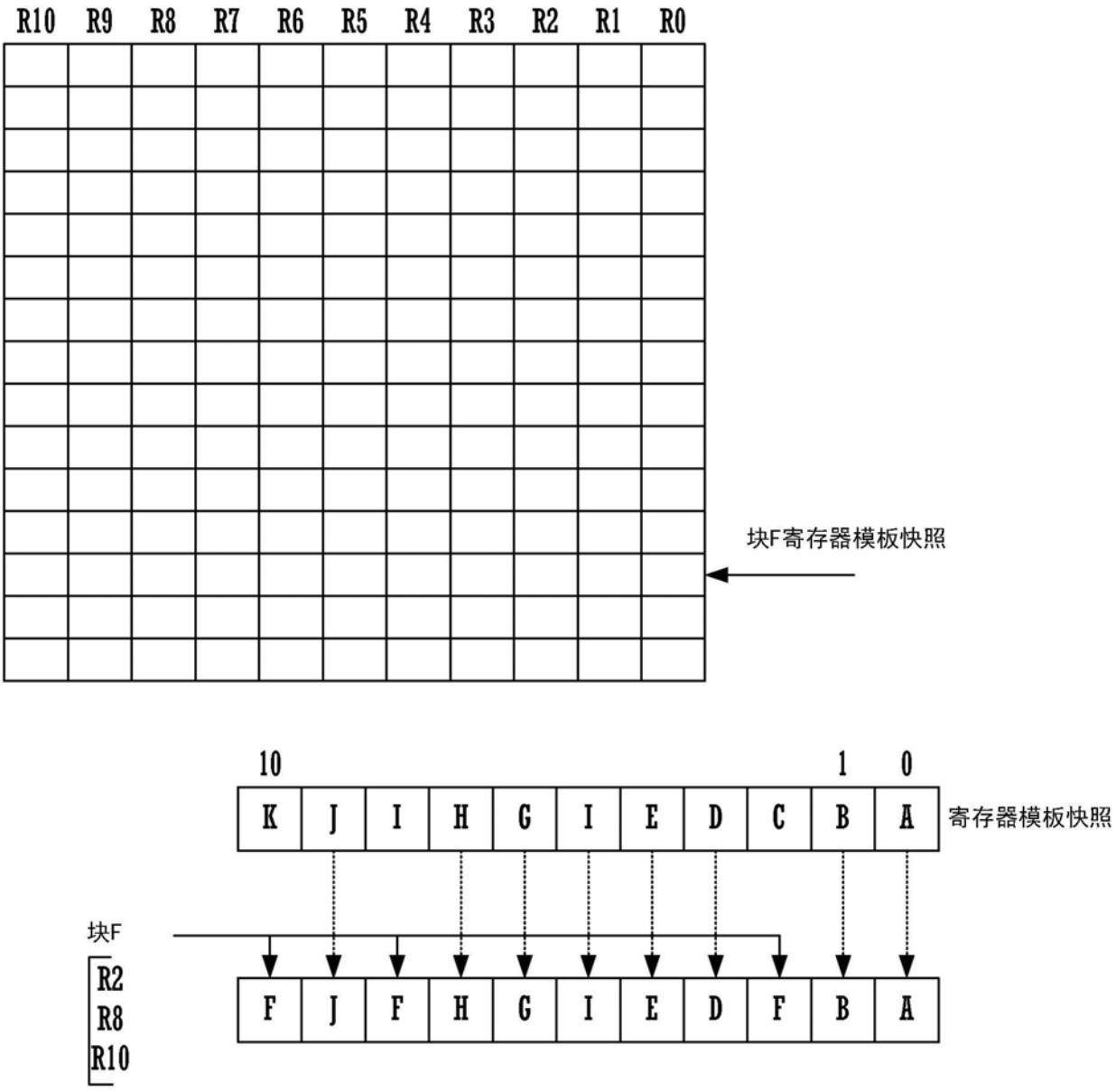


图12

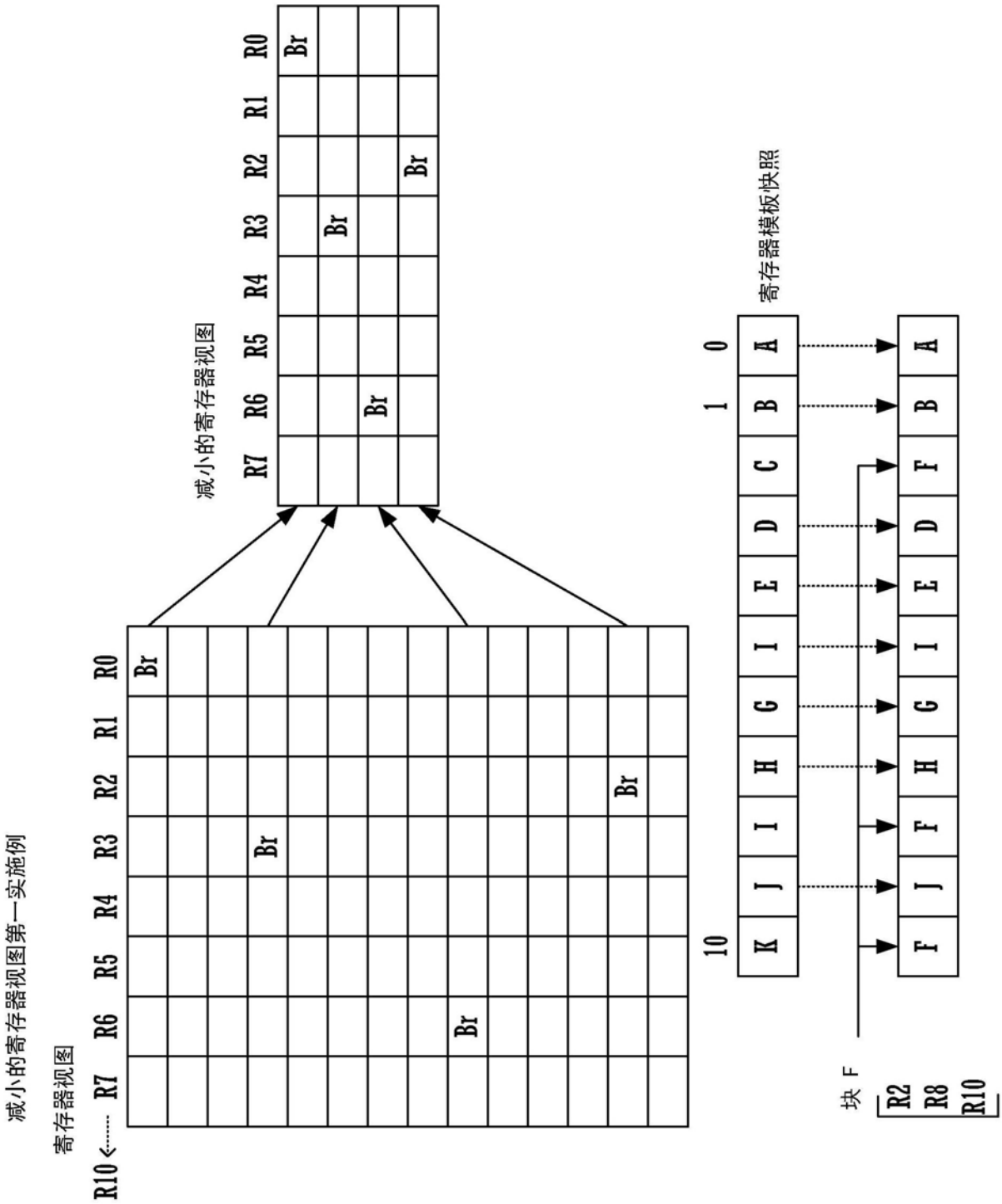

减小的寄存器视图第一实施例
寄存器视图
R10 ←…… R7 R6 R5 R4 R3 R2 R1 R0
减小的寄存器视图
R7 R6 R5 R4 R3 R2 R1 R0
寄存器模板快照
10 1 0
K J I H G I E D C B A
F J F H G I E D F B A
块 F
R2
R8
R10

图13

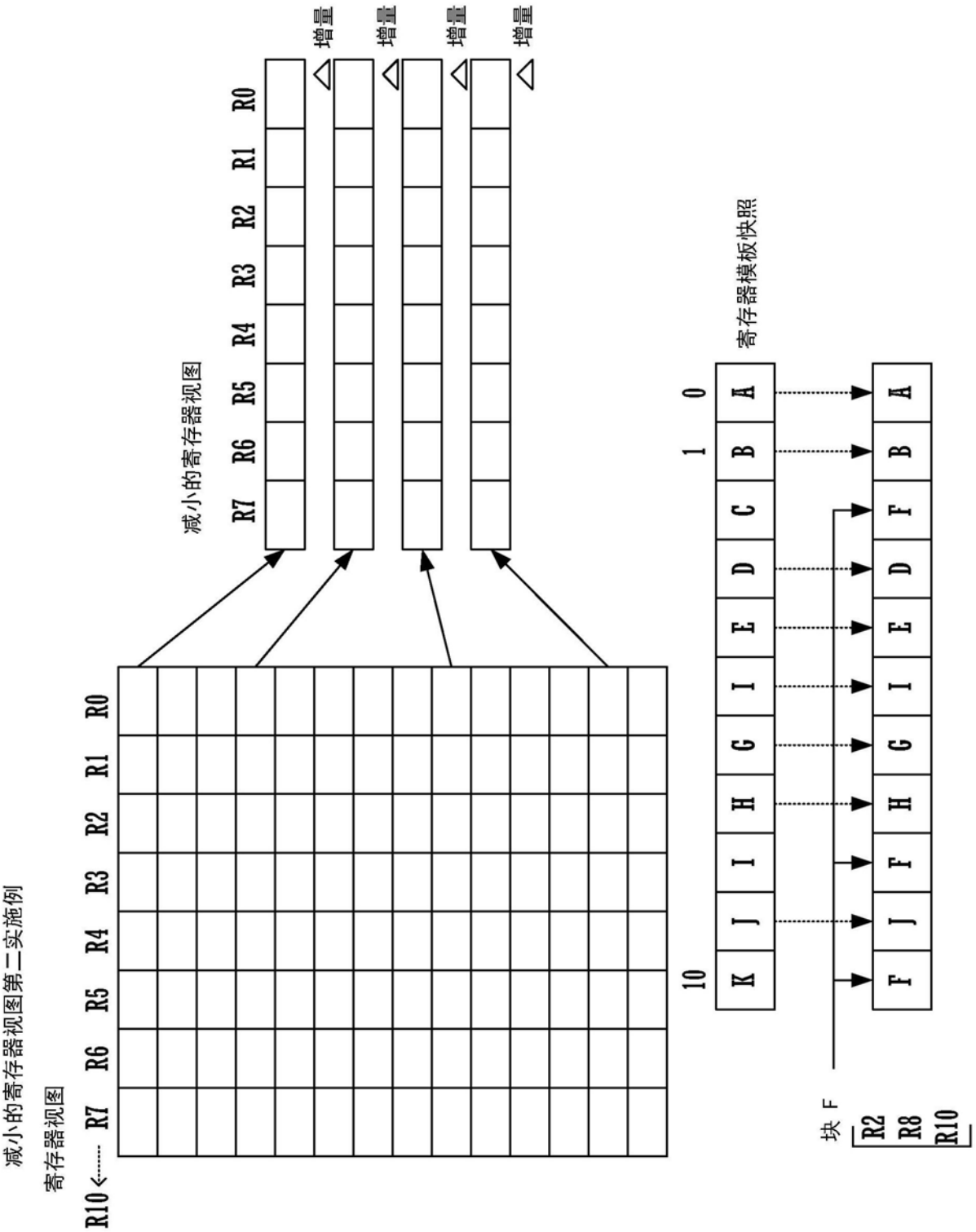

减小的寄存器视图第二实施例
寄存器视图
R10 ←…… R7 R6 R5 R4 R3 R2 R1 R0
减小的寄存器视图
R7 R6 R5 R4 R3 R2 R1 R0
增量
增量
增量
增量
寄存器模板快照
10 1 0
K J I H G I E D C B A
F J F H G I E D F B A
块 F
R2
R8
R10

图14

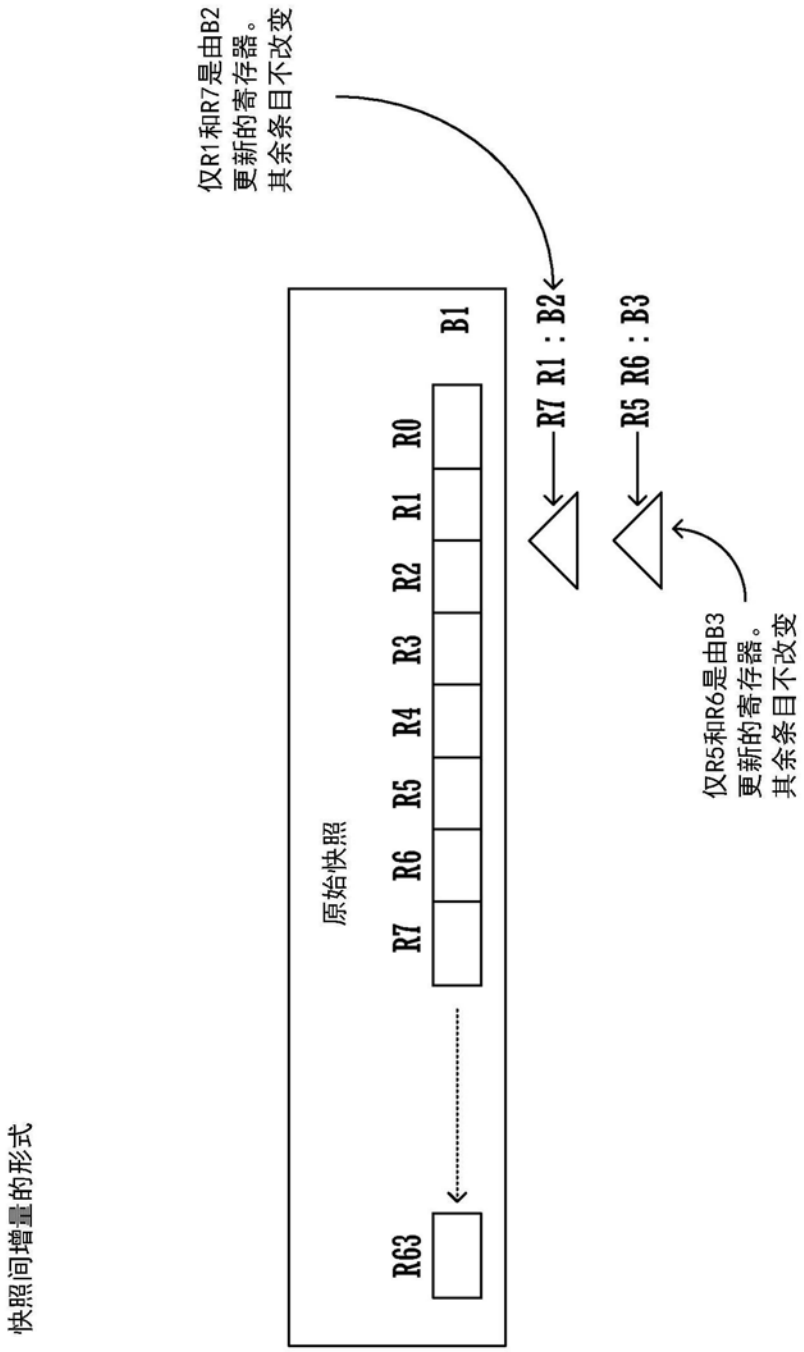

仅R1和R7是由B2
更新的寄存器。
其余条目不改变
R7 R1 : B2
R5 R6 : B3
仅R5和R6是由B3
更新的寄存器。
其余条目不改变
B1
R0
R1
R2
R3
R4
R5
R6
R7
原始快照
R63
快照间增量的形式

图15

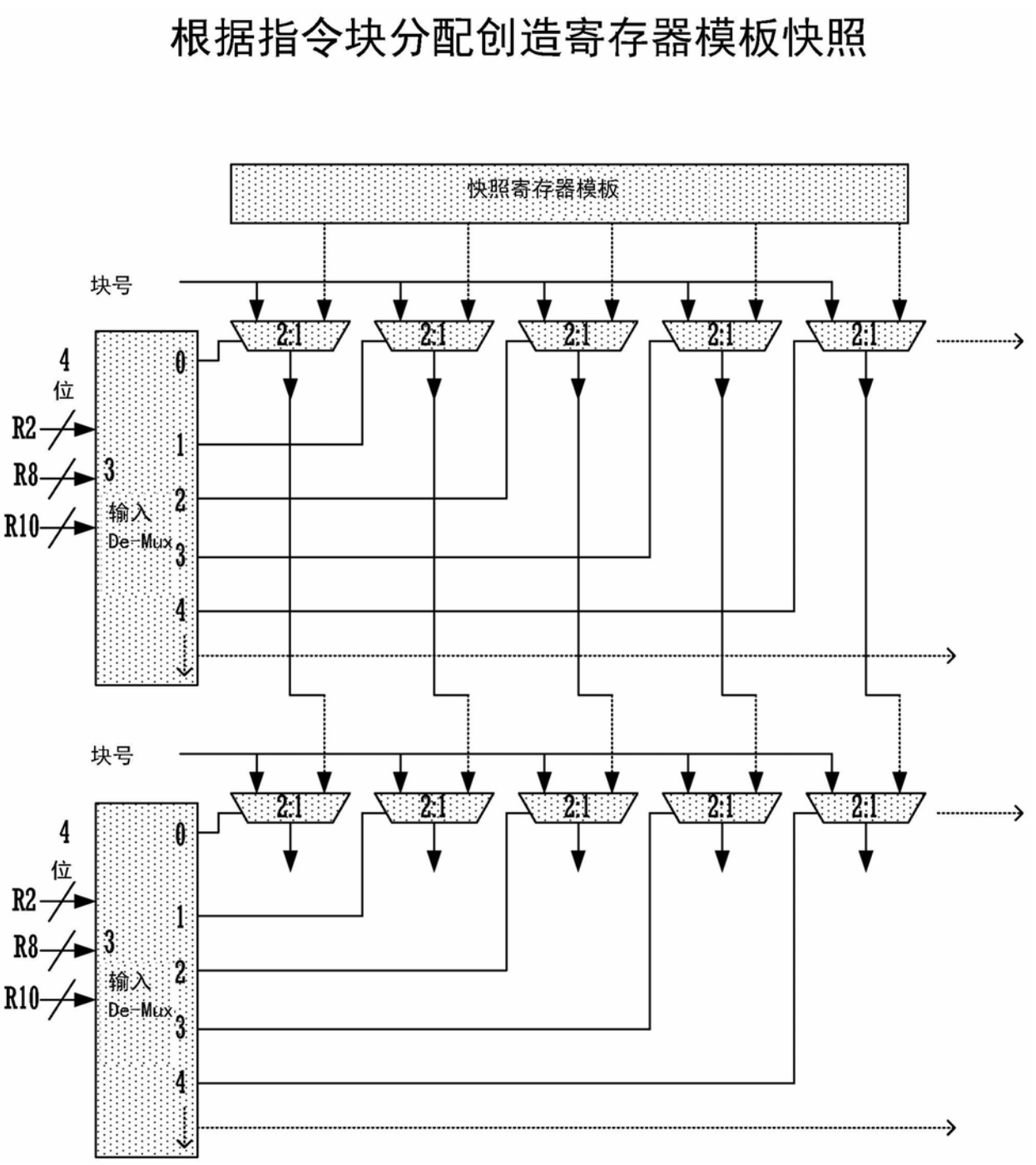

根据指令块分配创造寄存器模板快照
快照寄存器模板
块号
2:1
4
位
R2
R8
R10
3
输入
De-Mux
0
1
2
3
4
块号
2:1
4
位
R2
R8
R10
3
输入
De-Mux
0
1
2
3
4

图16

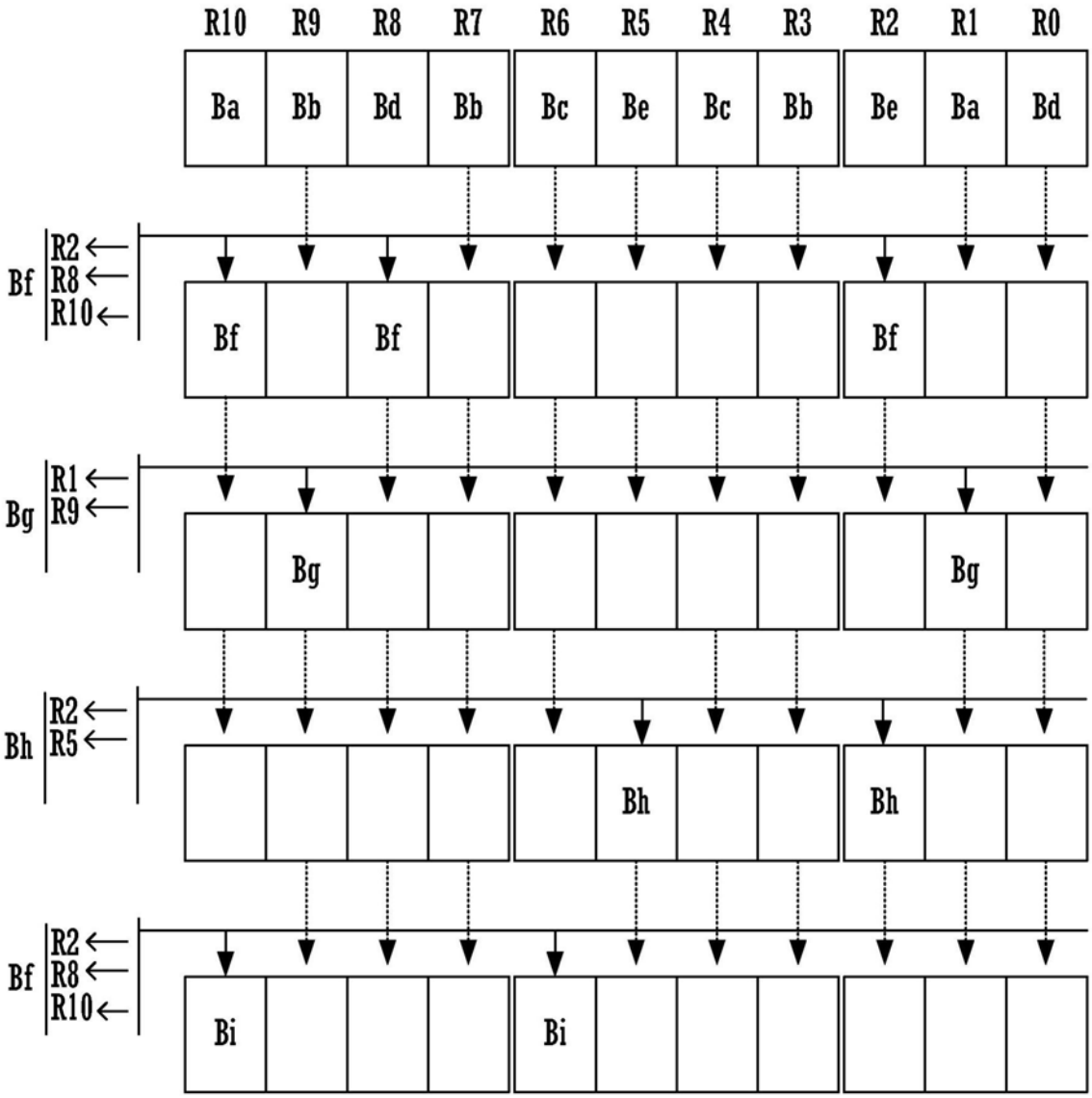

R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
Ba
Bb
Bd
Bb
Bc
Be
Bc
Bb
Be
Ba
Bd
Bf
R2
R8
R10
Bf
Bf
Bf
Bg
R1
R9
Bg
Bg
Bh
R2
R5
Bh
Bh
Bf
R2
R8
R10
Bi
Bi

图17

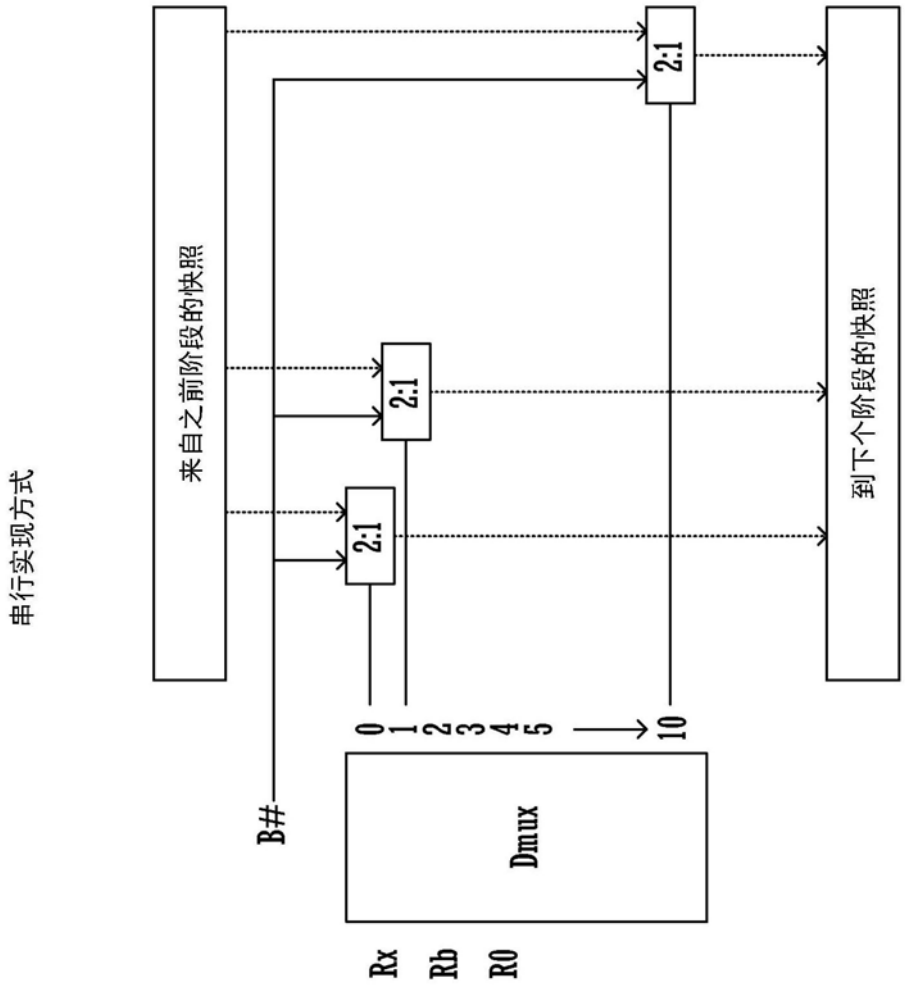


图18

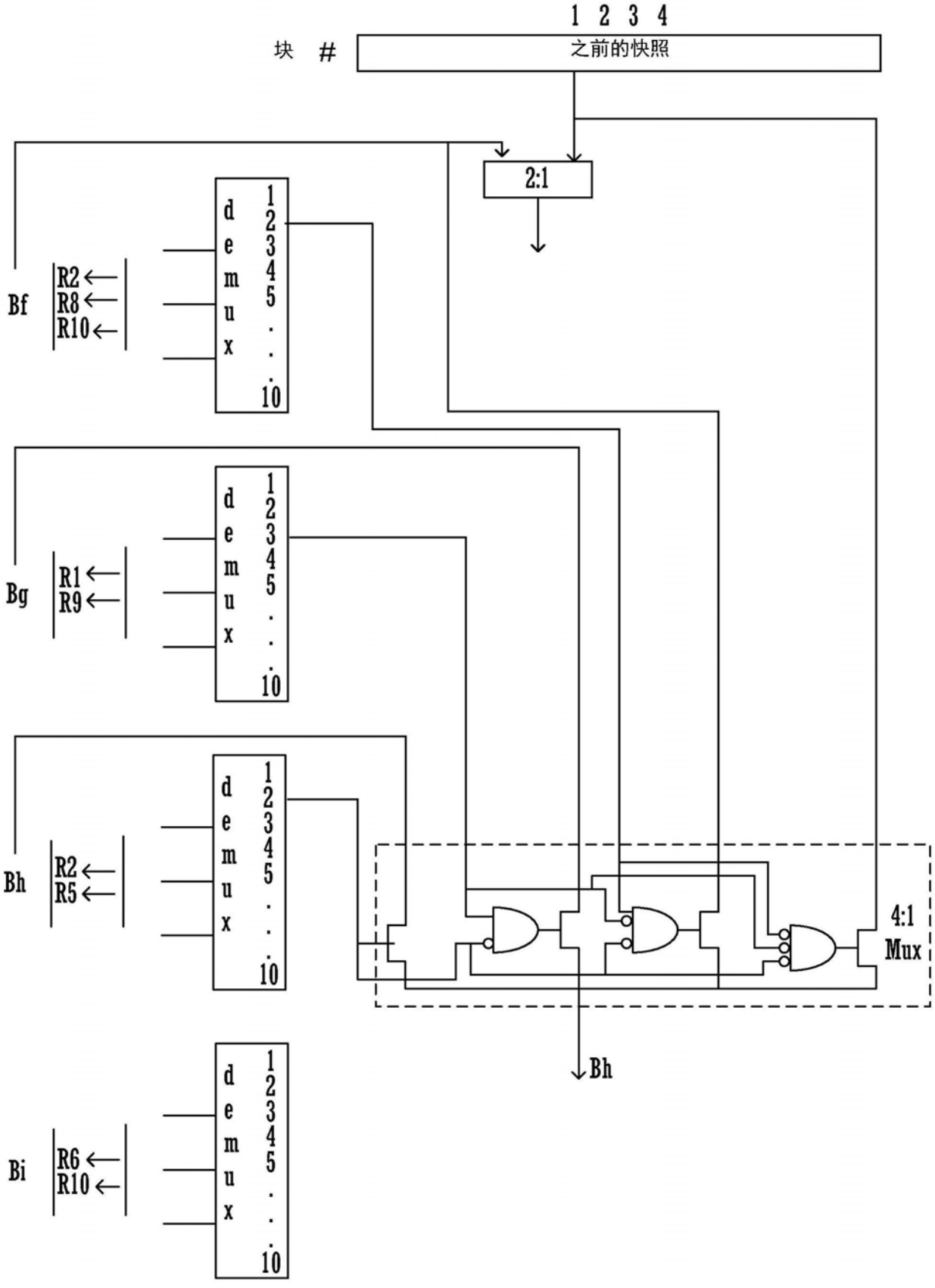

1 2 3 4
块 #
之前的快照
2:1
Bf
R2
R8
R10
Bg
R1
R9
Bh
R2
R5
Bi
R6
R10
demux
1 2 3 4 5 . . . 10
4:1
Mux
Bh

图19

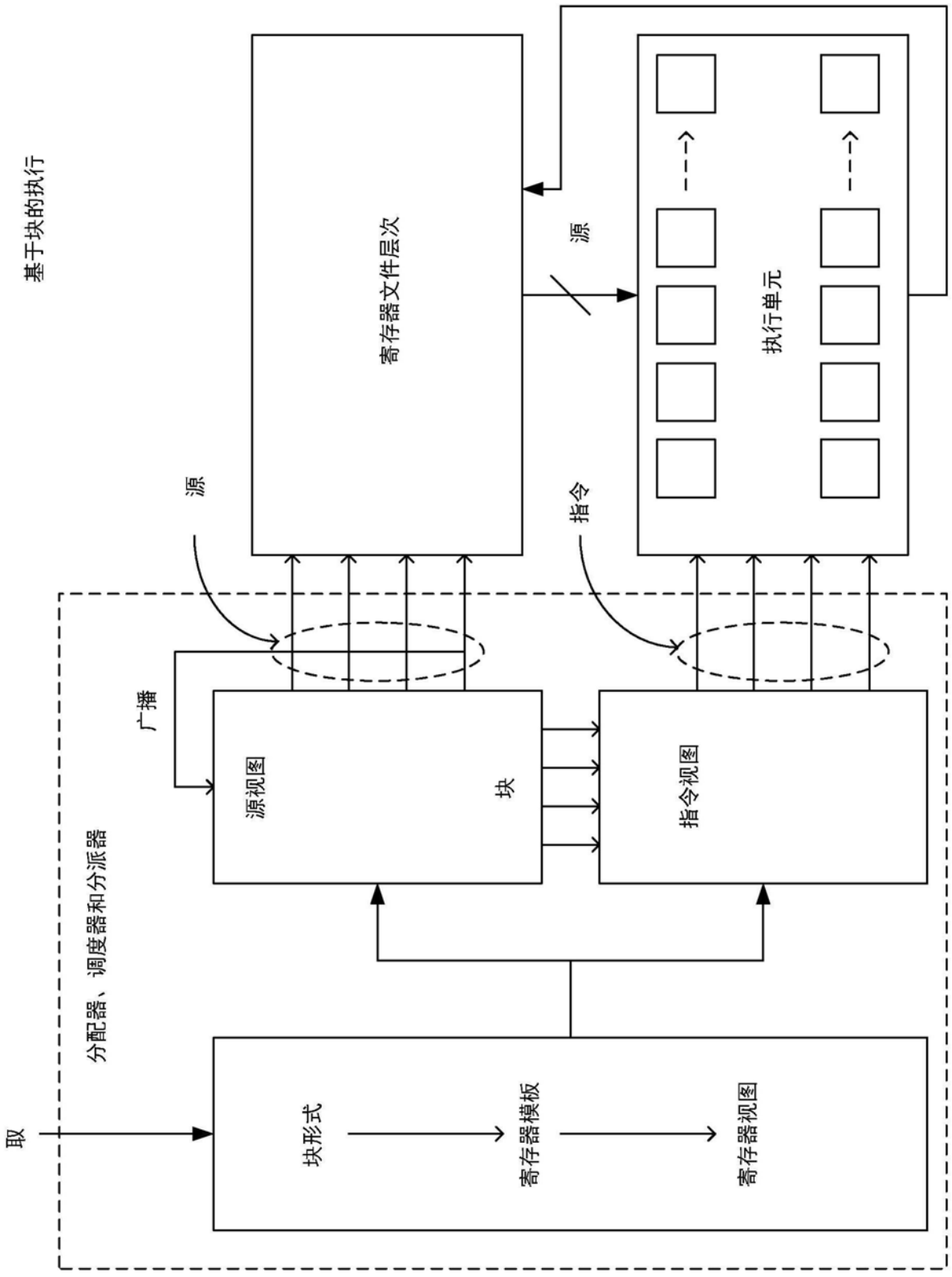

基于块的执行
寄存器文件层次
源
执行单元
源
指令
广播
源视图
块
指令视图
分配器、调度器和分派器
取
块形式
寄存器模板
寄存器视图

图20

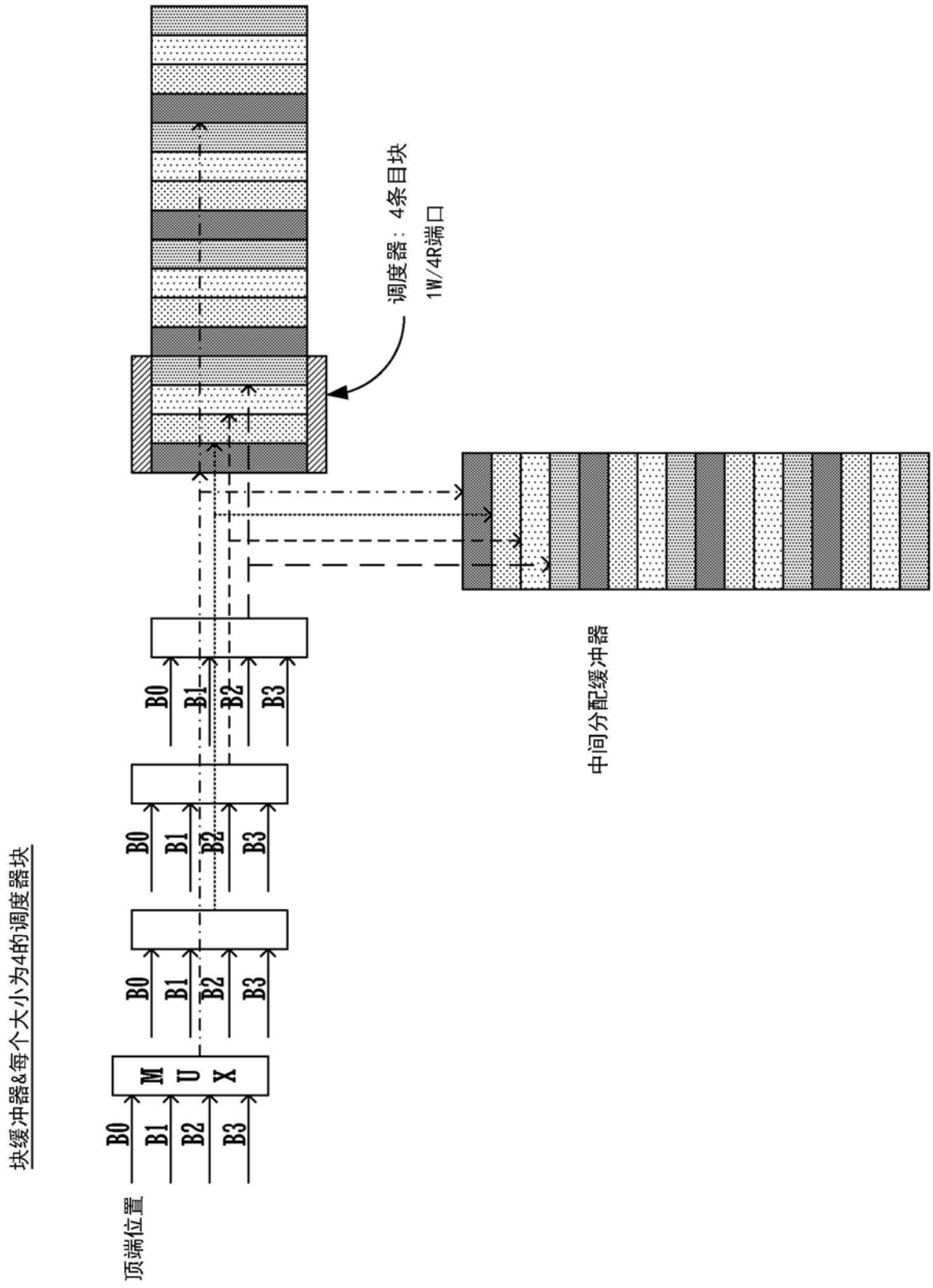

调度器：4条目块
1W/4R端口
中间分配缓冲器
块缓冲器&每个大小为4的调度器块
顶端位置
B0
B1
B2
B3
MUX

图21

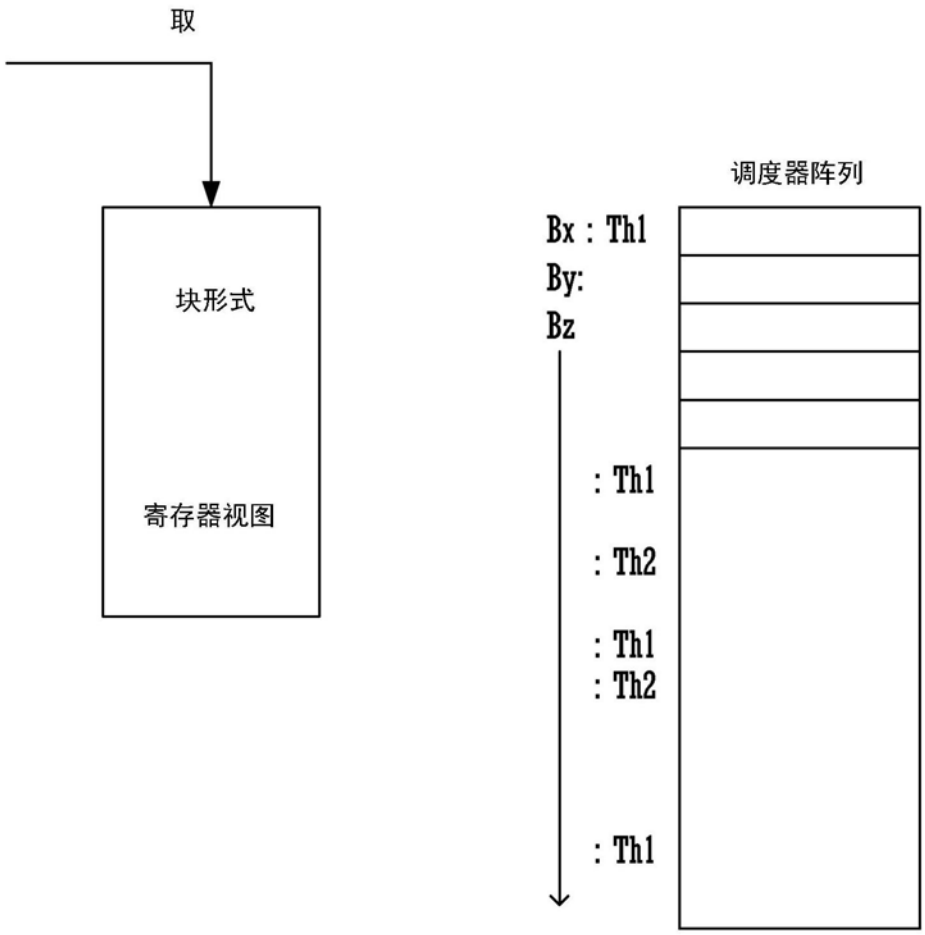

如何根据它们的块号和线程ID分配线程的描述
取
块形式
寄存器视图
调度器阵列
Bx : Th1
By:
Bz
: Th1
: Th2
: Th1
: Th2
: Th1

图22

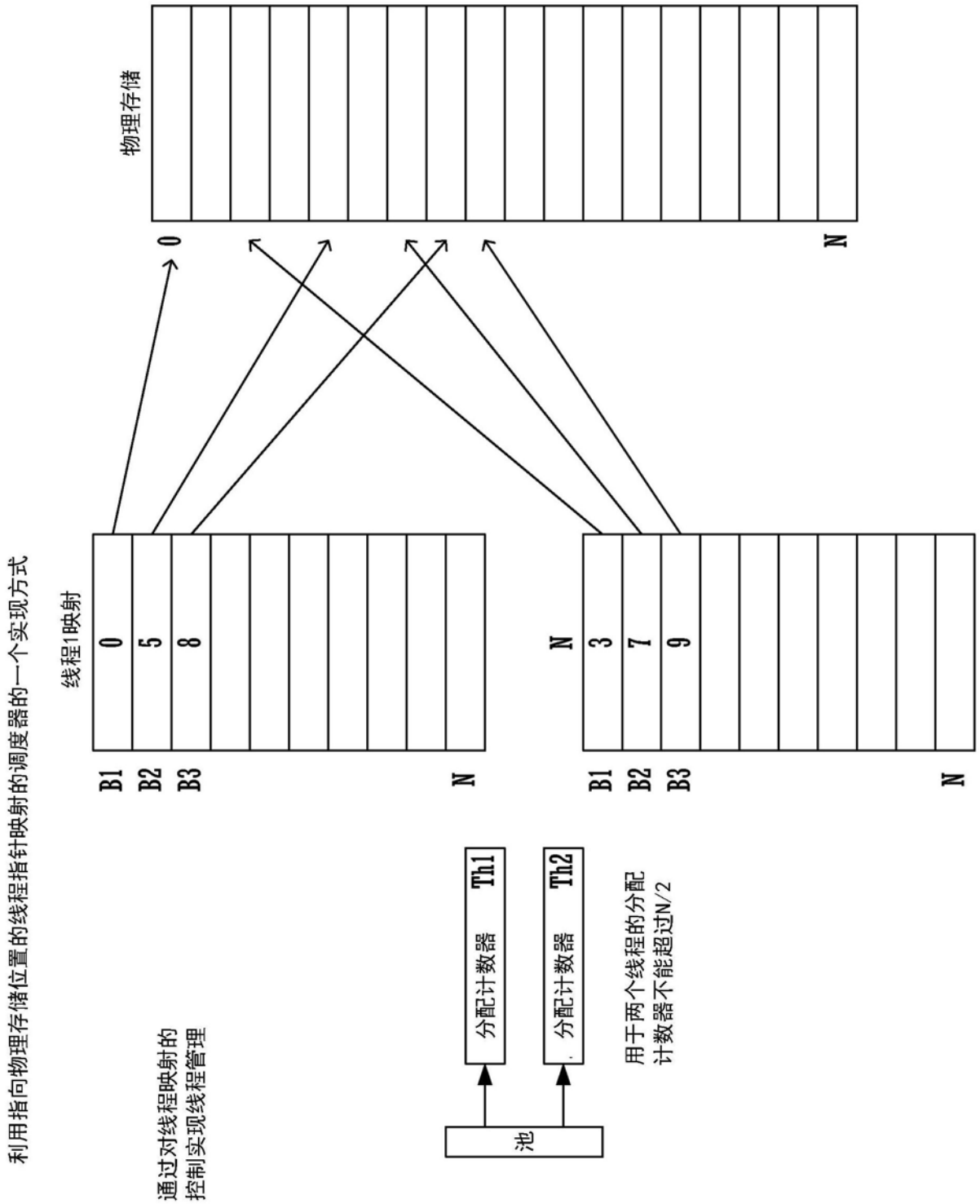

利用指向物理存储位置的线程指针映射的调度器的一个实现方式
物理存储
0
N
线程1映射
0
5
8
N
3
7
9
B1
B2
B3
N
B1
B2
B3
N
通过对线程映射的
控制实现线程管理
分配计数器 Th1
分配计数器 Th2
池
用于两个线程的分配
计数器不能超过N/2

图23

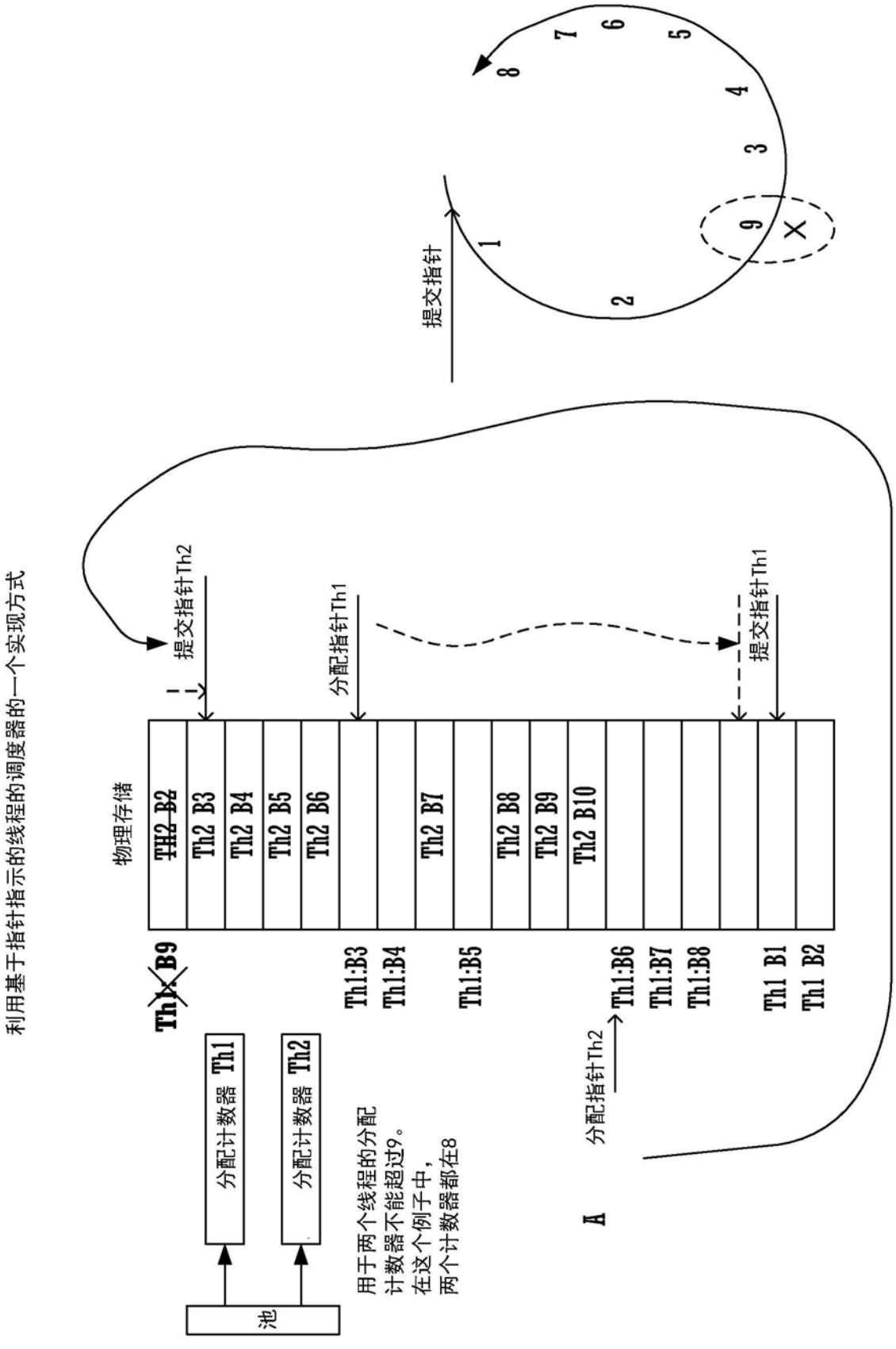

利用基于指针指示的线程的调度器的一个实现方式
物理存储
TH2 B2
Th2 B3
Th2 B4
Th2 B5
Th2 B6
Th2 B7
Th2 B8
Th2 B9
Th2 B10
Th1 B1
Th1 B2
Th1:B3
Th1:B4
Th1:B5
Th1:B6
Th1:B7
Th1:B8
提交指针Th2
分配指针Th1
提交指针Th1
分配指针Th2
提交指针
分配计数器 Th1
分配计数器 Th2
池
用于两个线程的分配
计数器不能超过9。
在这个例子中，
两个计数器都在8
A
1
2
3
4
5
6
7
8
9
X

图24

基于执行资源到线程的公平分配的动态计数器

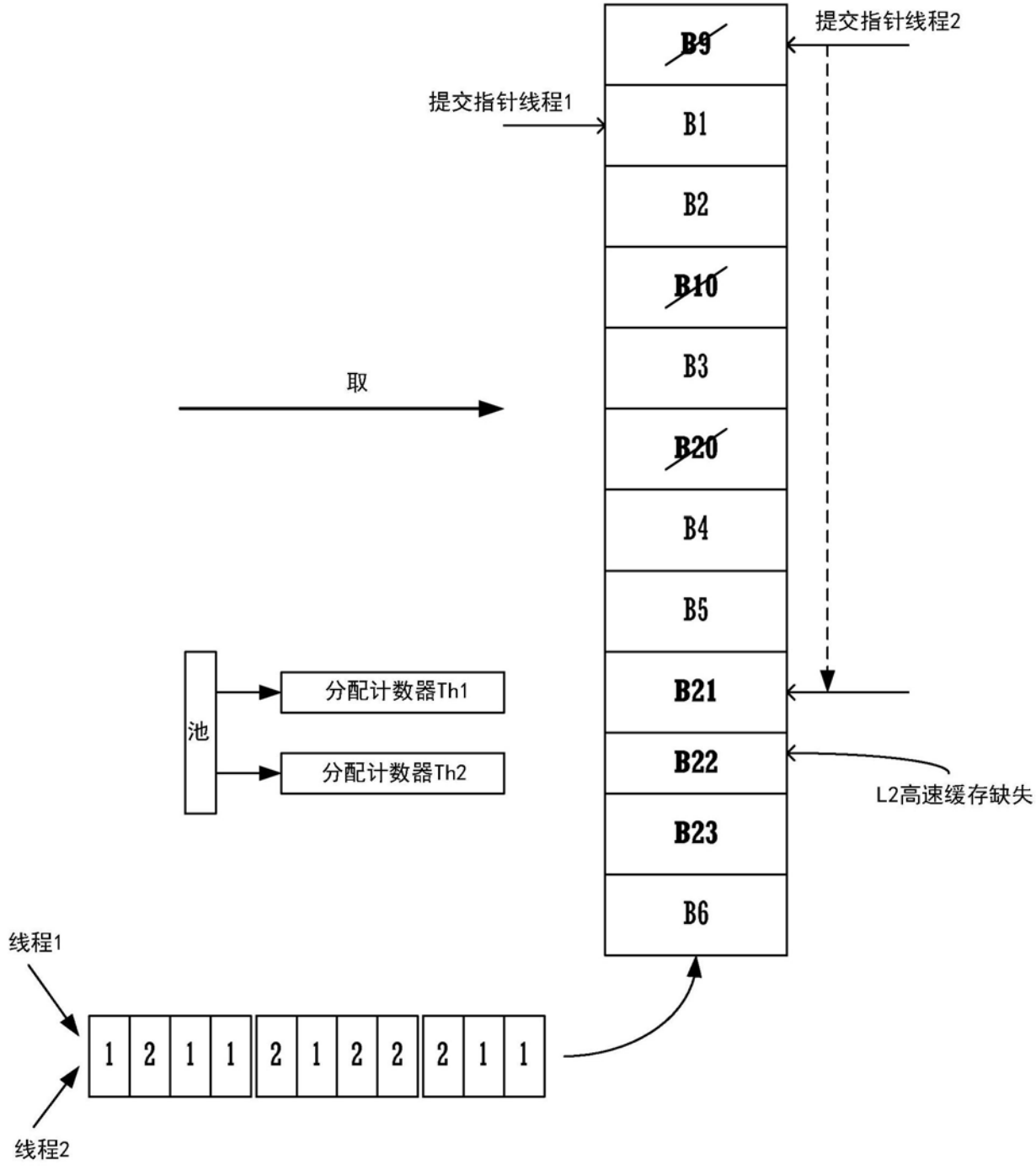


图25

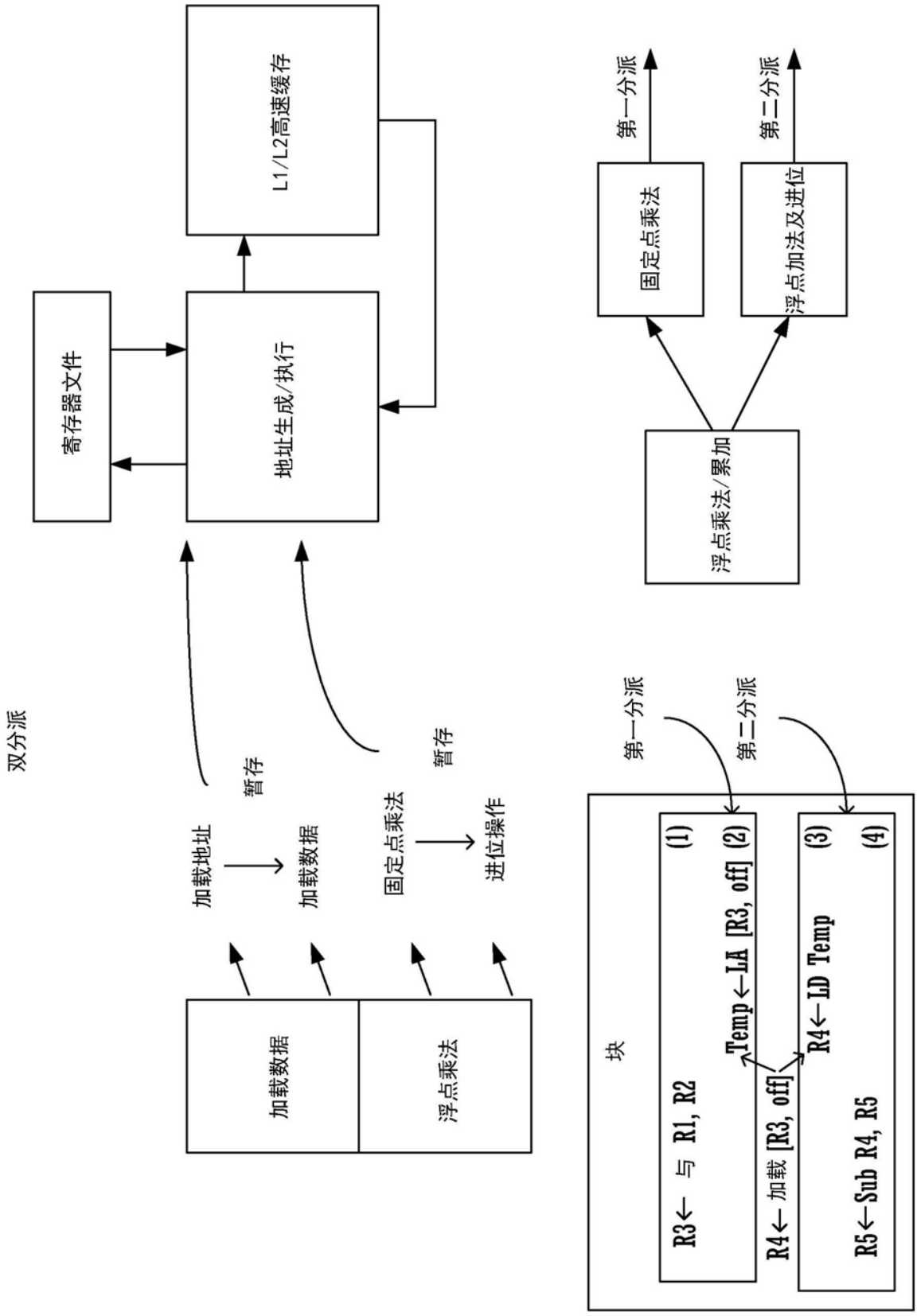

双分派
加载地址
暂存
加载数据
固定点乘法
暂存
进位操作
加载数据
浮点乘法
寄存器文件
地址生成/执行
L1/L2高速缓存
第一分派
第二分派
固定点乘法
浮点加法及进位
浮点乘法/累加
块
第一分派
第二分派
R3← 与 R1, R2
(1)
R4← 加载 [R3, off]
Temp←LA [R3, off]
(2)
R5←Sub R4, R5
R4←LD Temp
(3)
(4)

图26

双分派暂态相乘累加

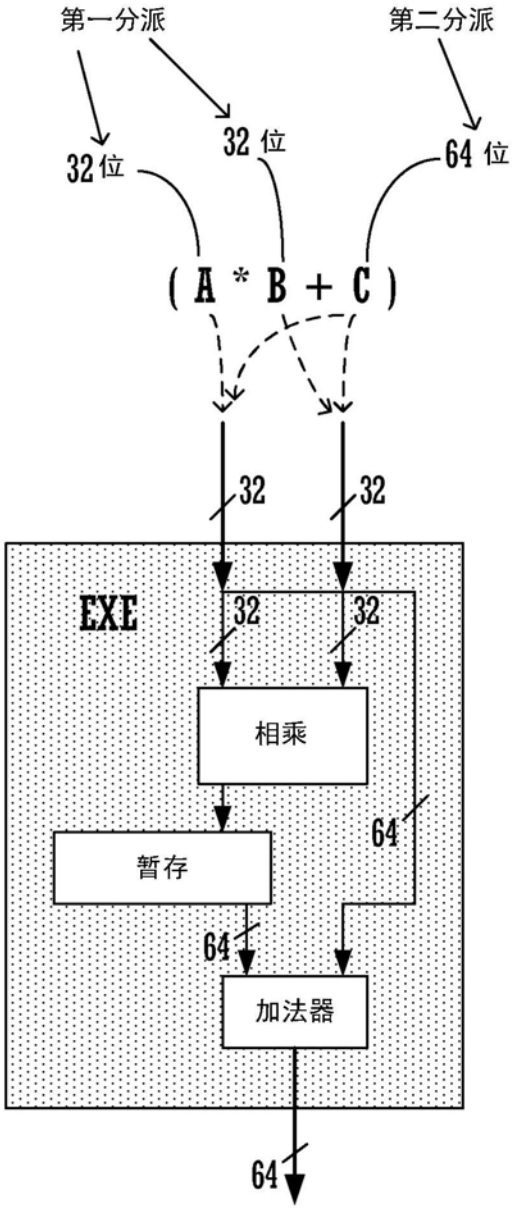


图27

体系架构上可见状态的双分派的相乘-相加

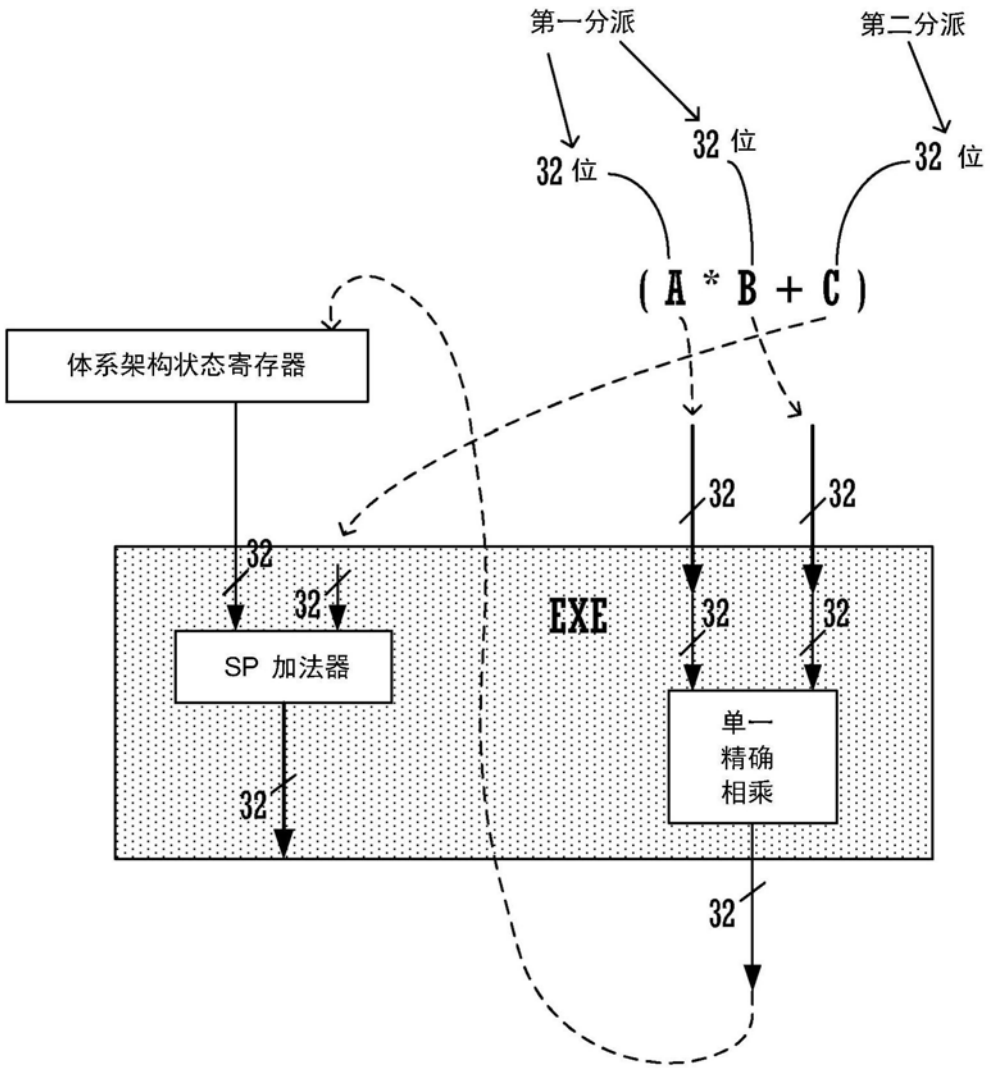


图28

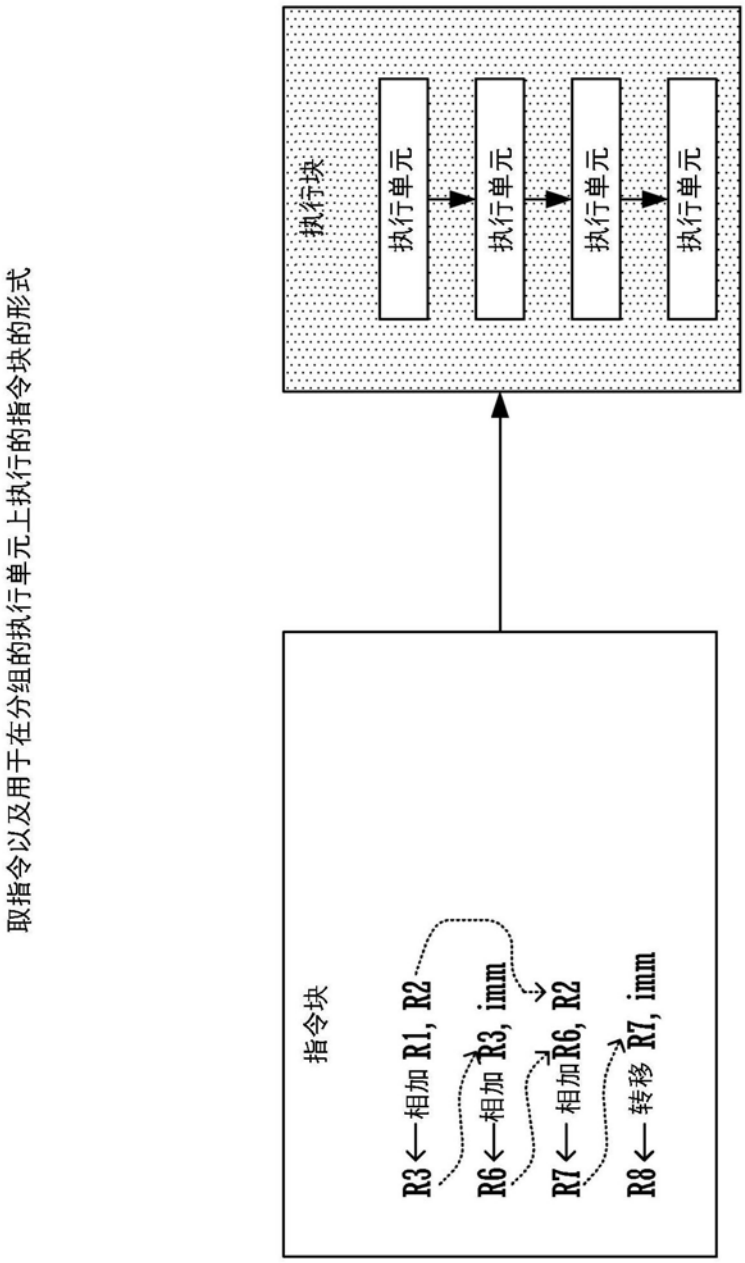

取指令以及用于在分组的执行单元上执行的指令块的形式
执行块
执行单元
执行单元
执行单元
执行单元
指令块
R3←相加 R1, R2
R6←相加 R3, imm
R7←相加R6, R2
R8←转移 R7, imm

图29

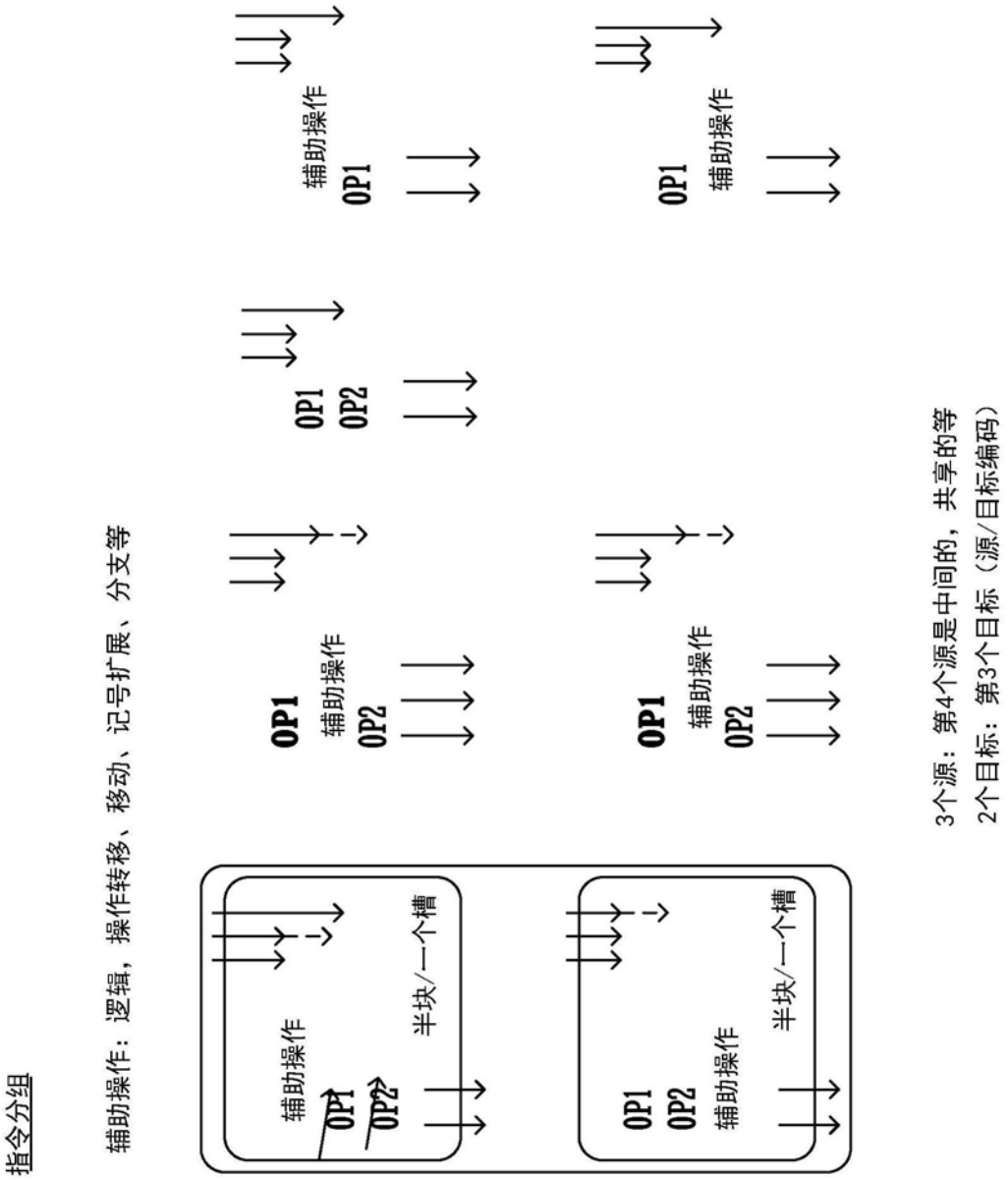

指令分组
辅助操作：逻辑，操作转移、移动、记号扩展、分支等
辅助操作
OP1
OP2
半块/一个槽
3个源：第4个源是中间的，共享的等
2个目标：第3个目标（源/目标编码）

图30

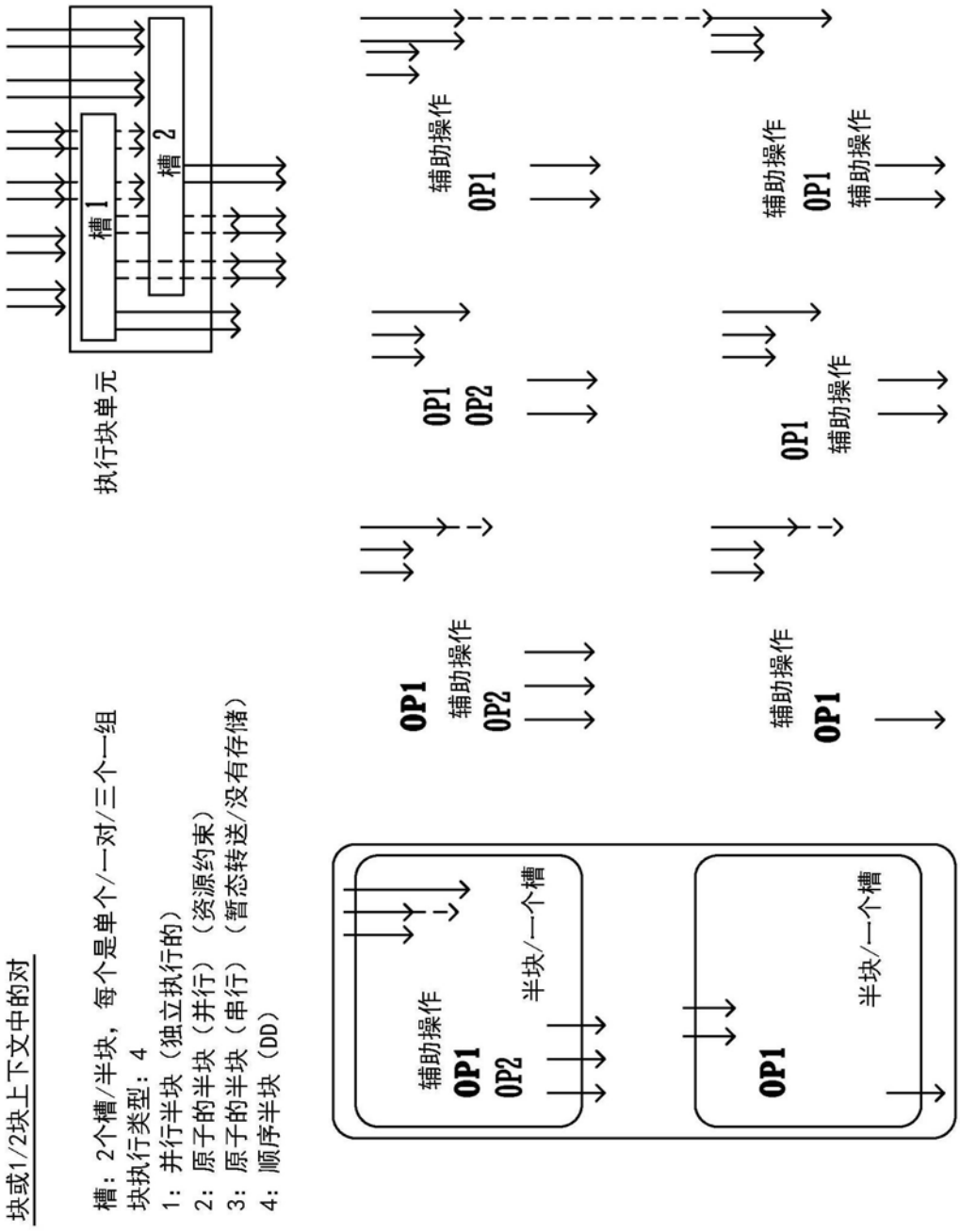

槽1
槽2
执行块单元
辅助操作
0P1
0P2
半块/一个槽
块或1/2块上下文中的对
槽：2个槽/半块，每个是单个/一对/三个一组
块执行类型：4
1：并行半块（独立执行的）
2：原子的半块（并行）（资源约束）
3：原子的半块（串行）（暂态转送/没有存储）
4：顺序半块（DD）

图31

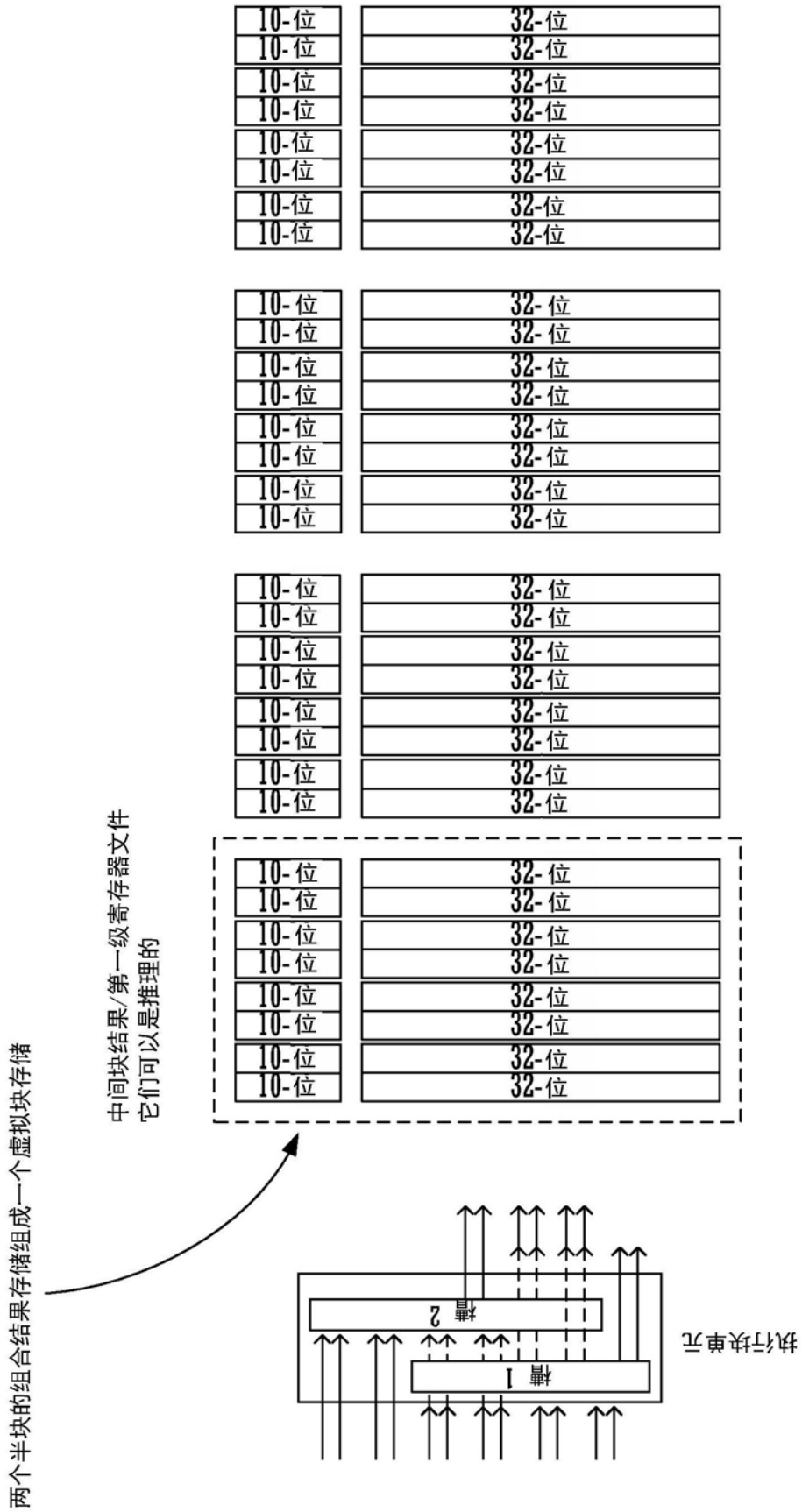

10-位
32-位
中间块结果/第一级寄存器文件
它们可以是推理的
两个半块的组合结果存储组成一个虚拟块存储
槽 2
槽 1
执行块单元

图32

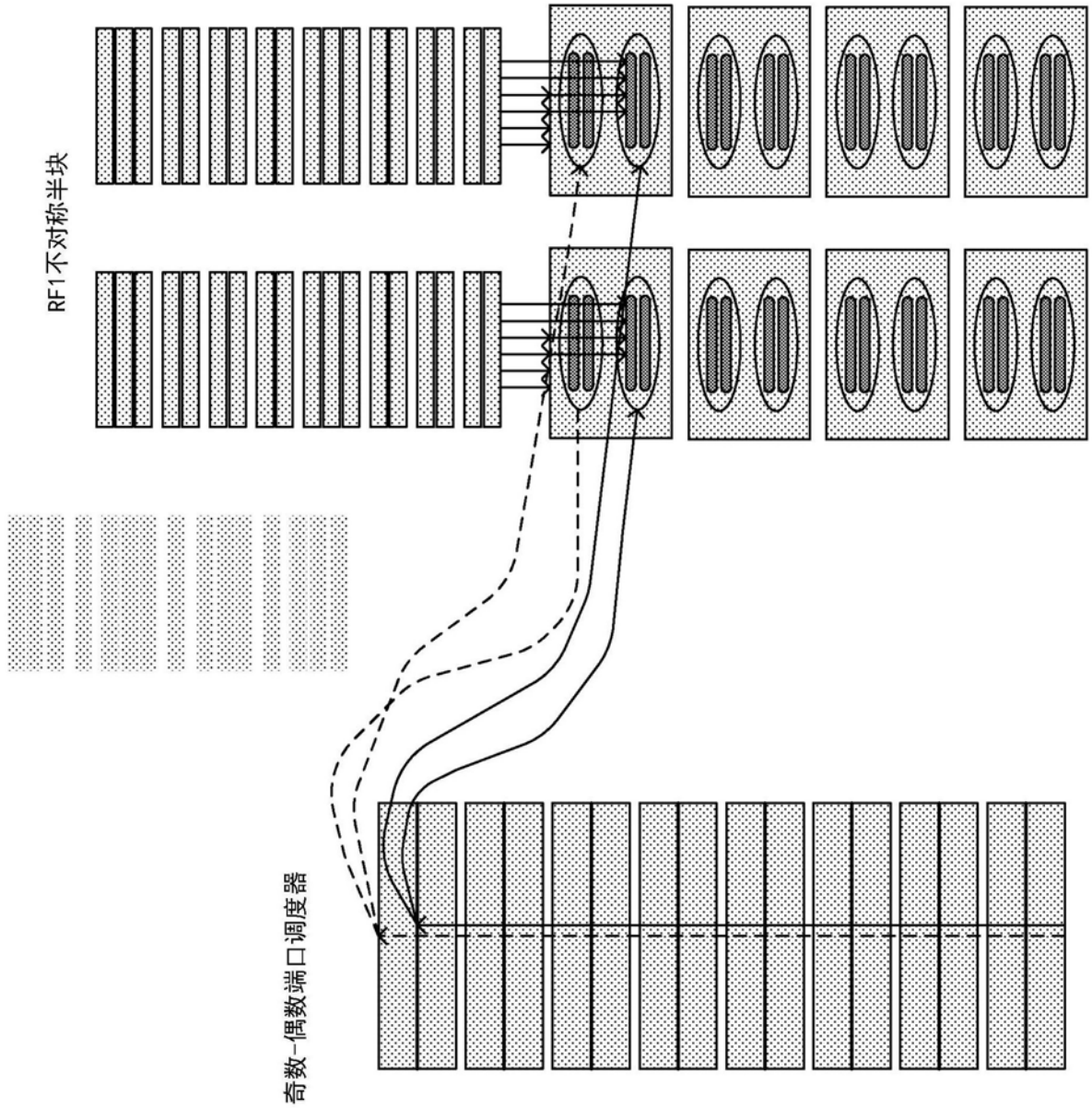


图33

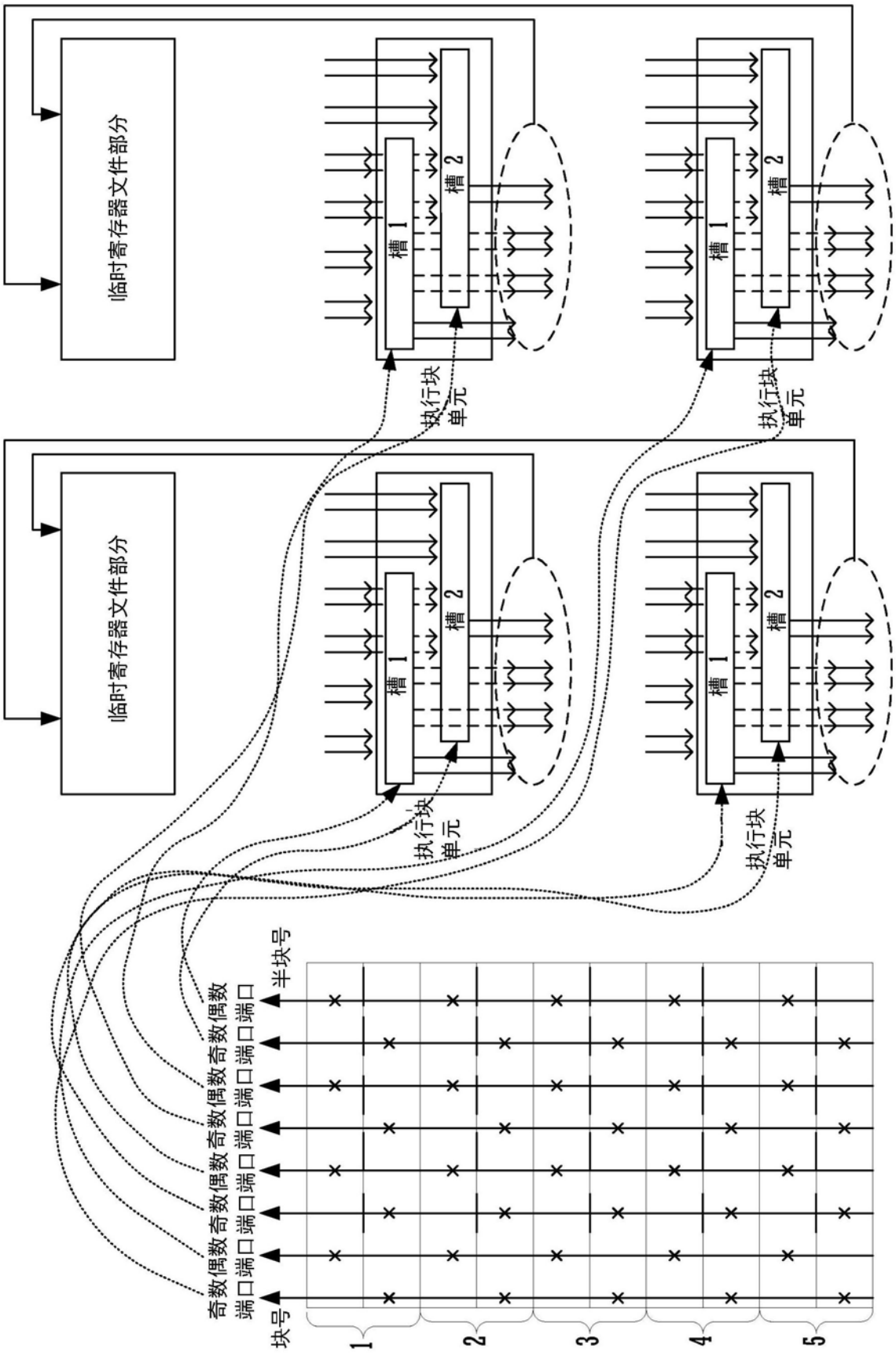

临时寄存器文件部分
临时寄存器文件部分
槽 1
槽 2
槽 1
槽 2
槽 1
槽 2
槽 1
槽 2
执行块单元
执行块单元
执行块单元
执行块单元
半块号
奇数偶数奇数偶数奇数偶数奇数偶数
端口端口端口端口端口端口端口端口
块号
1
2
3
4
5

图34

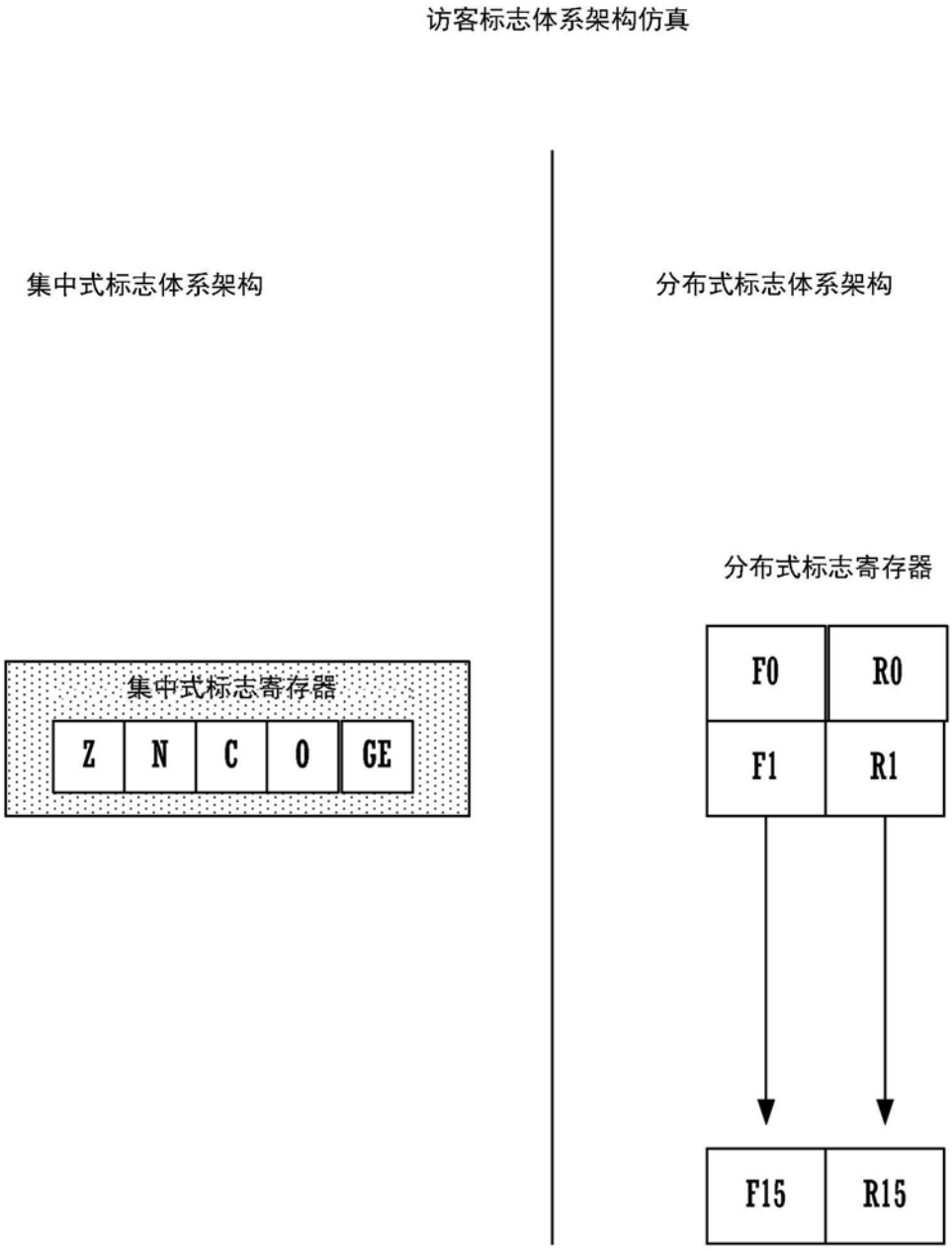

访客标志体系架构仿真
集中式标志体系架构
分布式标志体系架构
集中式标志寄存器
Z
N
C
O
GE
分布式标志寄存器
F0
R0
F1
R1
F15
R15

图35

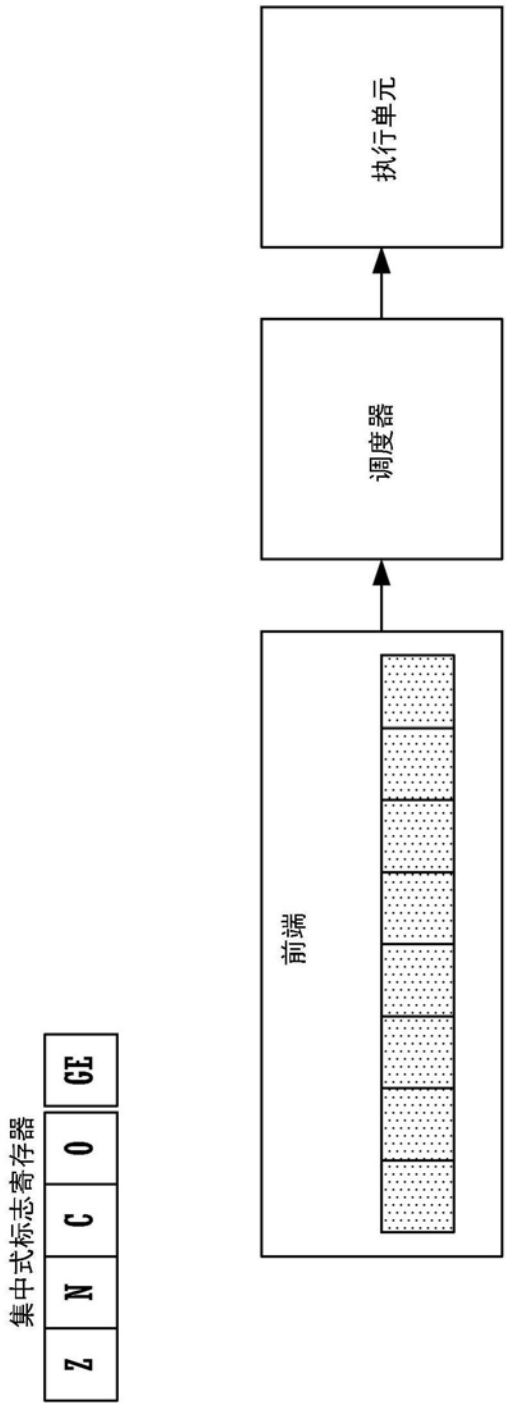


图36

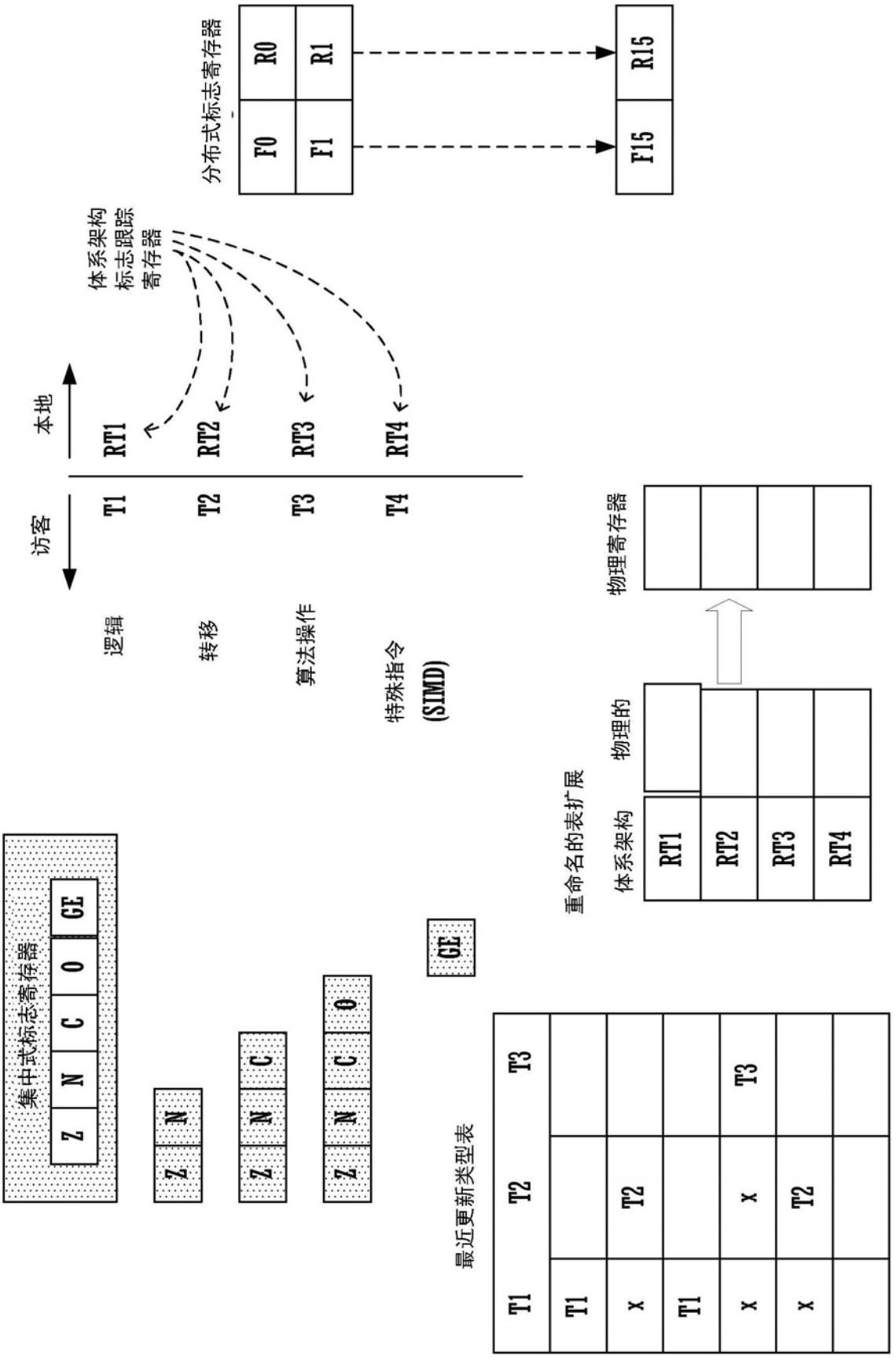

分布式标志寄存器
F0
R0
F1
R1
F15
R15
体系架构
标志跟踪
寄存器
本地
访客
T1
T2
T3
T4
RT1
RT2
RT3
RT4
逻辑
转移
算法操作
特殊指令
(SIMD)
物理寄存器
重命名的表扩展
体系架构
物理的
集中式标志寄存器
Z
N
C
O
GE
最近更新类型表
GE

图37

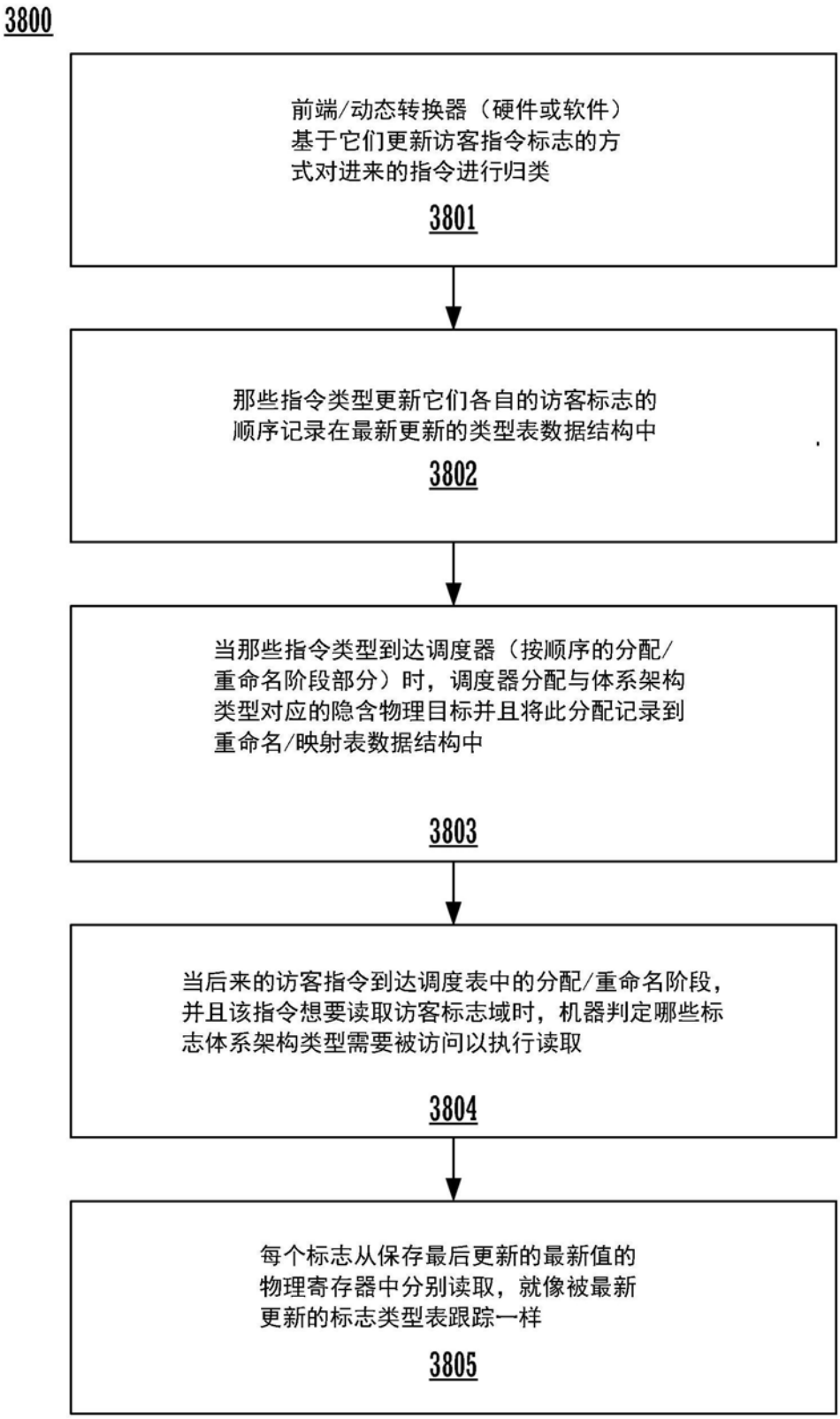

3800
前端/动态转换器（硬件或软件）基于它们更新访客指令标志的方式对进来的指令进行归类
3801
那些指令类型更新它们各自的访客标志的顺序记录在最新更新的类型表数据结构中
3802
当那些指令类型到达调度器（按顺序的分配/重命名阶段部分）时，调度器分配与体系架构类型对应的隐含物理目标并且将此分配记录到重命名/映射表数据结构中
3803
当后来的访客指令到达调度表中的分配/重命名阶段，并且该指令想要读取访客标志域时，机器判定哪些标志体系架构类型需要被访问以执行读取
3804
每个标志从保存最后更新的最新值的物理寄存器中分别读取，就像被最新更新的标志类型表跟踪一样
3805

图38