(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 105210040 A
(43)申请公布日 2015.12.30

(21)申请号 201480024528.0

(22)申请日 2014.03.12

(30)优先权数据
61/800,123 2013.03.15 US

(85)PCT国际申请进入国家阶段日
2015.10.30

(86)PCT国际申请的申请数据
PCT/US2014/024775 2014.03.12

(87)PCT国际申请的公布数据
WO2014/151018 EN 2014.09.25

(71)申请人 索夫特机械公司
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 北京市磐华律师事务所 11336
代理人 董巍 谢栒

(51)Int.Cl.
G06F 9/46(2006.01)

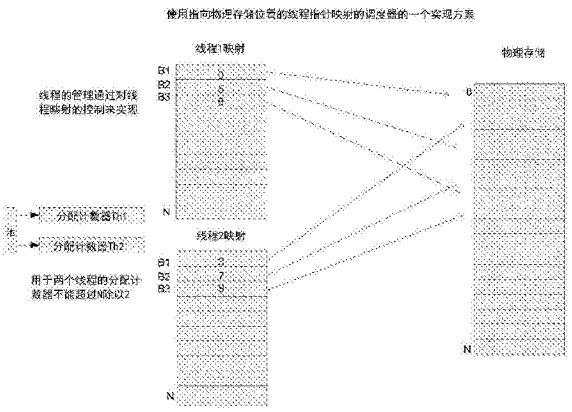
权利要求书2页 说明书14页 附图38页

(54)发明名称

用于执行分组成块的多线程指令的方法

(57)摘要

用于执行分组成块的多线程指令的方法。该方法包括使用全局前端接收进入的指令序列；将指令分组以形成指令块，其中所述指令块的指令与多个线程交错；调度所述指令块的指令以依照所述多个线程执行；以及跟踪对所述多个线程的执行以强制执行管线中的公正性。



CN 105210040 A

1. 一种用于执行分组成块的多线程指令的方法，包括：

使用全局前端接收进入的指令序列；

将所述指令分组以形成指令块，其中所述指令块的所述指令与多个线程交错；

调度所述指令块的所述指令以依照所述多个线程执行；以及

跟踪对所述多个线程的执行以强制执行管线中的公正性。

2. 根据权利要求 1 所述的方法，其中属于不同线程的块能够在调度器阵列中交错。

3. 根据权利要求 1 所述的方法，其中使用调度器线程指针来映射调度器阵列内交错的属于不同线程的块。

4. 根据权利要求 1 所述的方法，其中使用分配计数器来分配调度器阵列内的线程的块以实现公正策略。

5. 根据权利要求 1 所述的方法，其中使用动态的基于日历的分配来分配调度器阵列内的线程的块，以实现公正策略。

6. 根据权利要求 1 所述的方法，其中使用动态的基于日历的分配来分配调度器阵列内的线程的块，以实现线程分配的动态比例。

7. 根据权利要求 1 所述的方法，其中使用分配计数器来分配调度器阵列内的线程的块，以实现阻止一个线程阻塞另一个线程的进度。

8. 一种非暂时性计算机可读介质，具有计算机可读代码，所述代码当由计算机系统执行时，致使所述计算机系统实现用于执行分组成块的多线程指令的方法，包括：

使用全局前端接收进入的指令序列；

将所述指令分组以形成指令块，其中所述指令块的所述指令与多个线程交错；

调度所述指令块的所述指令以依照所述多个线程执行；以及

跟踪对所述多个线程的执行以强制执行管线中的公正性。

9. 根据权利要求 8 所述的计算机可读介质，其中属于不同线程的块能够在调度器阵列中交错。

10. 根据权利要求 8 所述的计算机可读介质，其中使用调度器线程指针来映射调度器阵列内交错的属于不同线程的块。

11. 根据权利要求 8 所述的计算机可读介质，其中使用分配计数器来分配调度器阵列内的线程的块，以实现公正策略。

12. 根据权利要求 8 所述的计算机可读介质，其中使用动态的基于日历的分配来分配调度器阵列内的线程的块，以实现公正策略。

13. 根据权利要求 8 所述的计算机可读介质，其中使用动态的基于日历的分配来分配调度器阵列内的线程的块，以实现线程分配的动态比例。

14. 根据权利要求 8 所述的计算机可读介质，其中使用分配计数器来分配调度器阵列内的线程的块，以实现阻止一个线程阻塞另一个线程的进度。

15. 一种计算机系统，具有耦连到存储器的处理器，所述存储器具有计算机可读代码，所述代码当由所述计算机系统执行时，致使所述计算机系统实现用于执行分组成块的多线程指令的方法，包括：

使用全局前端接收进入的指令序列；

将所述指令分组以形成指令块，其中所述指令块的所述指令与多个线程交错；

调度所述指令块的所述指令以依照所述多个线程执行；以及

跟踪对所述多个线程的执行以强制执行管线中的公正性。

16. 根据权利要求 15 所述的计算机系统，其中属于不同线程的块能够在调度器阵列中交错。

17. 根据权利要求 15 所述的计算机系统，其中使用调度器线程指针来映射调度器阵列内交错的属于不同线程的块。

18. 根据权利要求 15 所述的计算机系统，其中使用分配计数器来分配调度器阵列内的线程的块，以实现公正策略。

19. 根据权利要求 15 所述的计算机系统，其中使用动态的基于日历的分配来分配调度器阵列内的线程的块，以实现公正策略。

20. 一种用于实现微处理器中的减小尺寸的寄存器图数据结构的方法，包括：

使用全局前端接收进入的指令序列；

将所述指令分组以形成指令块；

使用多个多路复用器访问调度阵列的端口，以将所述指令块存储为一系列组块。

用于执行分组成块的多线程指令的方法

[0001] 本申请要求与在2013年3月15日提交的、穆罕默德A. 阿卜杜拉(Mohammad A. Abdallah)的题为“用于执行分组成块的多线程指令的方法”的、序列号为61/800,123的共同转让的共同未决的美国临时专利申请的权益,其全部内容通过引用并入本文。

[0002] 相关申请的交叉引用

[0003] 本申请与在2007年4月12日提交的、穆罕默德A. 阿卜杜拉的题为“用于处理指定并行独立操作的指令矩阵的装置和方法”的、序列号为2009/0113170的共同转让的共同未决的美国专利申请相关,其全部内容通过引用并入本文。

[0004] 本申请与在2007年11月14日提交的、穆罕默德A. 阿卜杜拉的题为“用于处理支持各种上下文切换模式和虚拟化方案的多线程体系架构中的复杂指令格式的装置和方法”的、序列号为2010/0161948的共同转让共同未决的美国专利申请相关,其全部内容通过引用并入本文。

技术领域

[0005] 本发明一般地涉及数字计算机系统,更具体地,涉及用于选择包括指令序列的指令的系统和方法。

背景技术

[0006] 需要处理器来处理相依赖的或完全独立的多个任务。这类处理器的内部状态通常包含可保存程序执行的每个特定瞬间的不同值的寄存器。在程序执行的每个瞬间,内部状态图像被称为处理器的体系架构状态。

[0007] 当切换代码执行以运行另一函数(例如,另一线程、过程或程序)时,需要保存机器/处理器的状态,这样新的函数可利用内部寄存器来建立其新的状态。一旦新功能结束,则可丢弃其状态,并且将恢复先前内容的状态且执行继续。这样的切换过程被称为上下文切换(context switch),并且通常包括数十个或数百个循环,特别是采用大量寄存器(例如64,128,256)和/或乱序执行的现代体系架构。

[0008] 在线程感知的硬件体系架构中,硬件支持多上下文状态用于有限数目的硬件支持的线程是正常的。在这种情况下,硬件复制所有体系架构状态元素用于每个所支持的线程。这消除了当执行新线程时对上下文切换的需求。然而,这仍存在多个缺点,也就是针对硬件中所支持的每个附加线程复制所有体系架构状态元素(即寄存器)的区域、电力和复杂度。此外,如果软件线程的数目超过所明确支持的硬件线程的数目,则仍然必须实施上下文切换。

[0009] 这将由于在要求大量线程的细粒度基础上对并行机制的需要而变得普遍。采用复制的上下文状态硬件存储的硬件线程感知体系架构对非线程软件代码没有帮助,且只能针对线程化的软件减少上下文切换的数目。然而,那些线程通常被构建用于粗粒并行机制,并导致繁重的软件开销用于初始化和同步化,留下细粒并行机制(诸如函数调用和循环并行执行)没有高效的线程初始化/自动生成。针对非明确地/容易地并行化/线程软件代码

使用现有技术中的编译器或用户并行化技术，这类所描述的开销伴随这类代码的自动并行化的困难。

发明内容

[0010] 在一个实施例中，本发明实现为用于执行分组成块的多线程指令的方法。该方法包括使用全局前端接收进入的指令序列；将指令分组以形成指令块，其中所述指令块的指令与多个线程交错；调度所述指令块的指令以依照所述多个线程执行；以及跟踪对所述多个线程的执行以强制执行管线中的公正性。

[0011] 前述是概要并且因此不可避免地包含简单化、一般化和细节的省略；因此，本领域技术人员将理解概要仅是示例性的，并且非意在以任何方式加以限制。本发明的其他方面、创造性特征和优势，如权利要求所唯一定义的，将在下文阐述的非限制的详细描述中变得显而易见。

附图说明

[0012] 以示例而非限制的方式将本发明示出在附图的图中，在附图中，类似的参考标记指代相似的元件。

[0013] 图 1 示出用于通过使用寄存器模板将指令分组到块中并跟踪指令之间的依赖关系的概略图。

[0014] 图 2 示出根据本发明一个实施例的寄存器图、源图以及指令图的概略图。

[0015] 图 3 示出根据本发明一个实施例的、阐明示例性寄存器模板以及来自寄存器模板的信息如何填充源图的示意图。

[0016] 图 4 示出阐明用于源图内的依赖关系广播的第一实施例的示意图。在该实施例中，每列包括一个指令块。

[0017] 图 5 示出阐明用于源图内的依赖关系广播的第二实施例的示意图。

[0018] 图 6 示出根据本发明一个实施例的、阐明针对从提交指针 (commit pointer) 开始分派以及广播相应的端口指派选择就绪块的示意图。

[0019] 图 7 示出根据本发明一个实施例的、实现图 6 中所描述的选择器阵列所使用的加法器树结构。

[0020] 图 8 更详细地示出选择器阵列加法器树的示例性逻辑。

[0021] 图 9 示出根据本发明一个实施例的、用于实现选择器阵列的加法器树的并行实现方案。

[0022] 图 10 示出根据本发明一个实施例的、阐明通过使用进位保留加法器可如何实现来自图 9 的加法器 X 的示例性示意图。

[0023] 图 11 示出根据本发明的用于针对从提交指针开始调度和使用选择器阵列加法器掩蔽就绪位的掩蔽实施例。

[0024] 图 12 示出根据本发明一个实施例的、寄存器模板如何填充寄存器图条目的概略图。

[0025] 图 13 示出根据本发明一个实施例的、用于减少的寄存器图印记 (footprint) 的第一实施例。

[0026] 图 14 示出根据本发明一个实施例的、用于减少的寄存器印记的第二实施例。

[0027] 图 15 示出根据本发明一个实施例的、快照之间的德尔塔 (delta) 的示例性格式。

[0028] 图 16 示出根据本发明一个实施例的、用于根据指令的块的分配创建寄存器模板快照的过程的示意图。

[0029] 图 17 示出根据本发明一个实施例的、用于根据指令的块的分配创建寄存器模板快照的过程的另一示意图。

[0030] 图 18 示出根据本发明一个实施例的、用于实现从先前寄存器模板创建后续寄存器模板的串行实现方案的硬件的概略图。

[0031] 图 19 示出根据本发明一个实施例的、用于实现从先前寄存器模板创建后续寄存器模板的并行实现方案的硬件的概略图。

[0032] 图 20 示出根据本发明一个实施例的、用于指令基于块的执行的硬件以及其如何与源图、指令图、寄存器模板以及寄存器图工作的概略图。

[0033] 图 21 示出根据本发明一个实施例的组块体系架构的示例。

[0034] 图 22 示出根据本发明一个实施例的、如何根据线程的块编号和线程 ID 对线程进行分配的描述。

[0035] 图 23 示出根据本发明一个实施例的、使用指向物理存储位置的线程指针映射以管理多线程执行的调度器的实现方案。

[0036] 图 24 示出根据本发明一个实施例的、使用基于线程的指针映射的调度器的另一实现方案。

[0037] 图 25 示出根据本发明一个实施例的、对线程的执行资源的动态的基于日历的分配的示意图。

[0038] 图 26 示出根据本发明一个实施例的双分派过程。

[0039] 图 27 示出根据本发明一个实施例的双分派暂时乘累加器。

[0040] 图 28 示出根据本发明一个实施例的双分派体系架构可视状态乘加器。

[0041] 图 29 示出根据本发明一个实施例的用于在经分组的执行单元过程上执行的指令块的取回和形成的概略图。

[0042] 图 30 示出根据本发明一个实施例的指令分组的示例性示意图。在图 30 的实施例中示出第三辅助操作的两个指令。

[0043] 图 31 示出根据本发明一个实施例的、块堆栈内的半块对如何映射到执行块单元。

[0044] 图 32 示出根据本发明一个实施例的、描绘中间块结果存储作为第一级别寄存器文件的示意图。

[0045] 图 33 示出根据本发明一个实施例的奇数 / 偶数端口调度器。

[0046] 图 34 示出图 33 的更详细版本，其中示出四个执行单元从调度器阵列接收结果并将输出写入到临时寄存器文件段。

[0047] 图 35 示出根据本发明一个实施例的描绘访客标志 (guest flag) 体系架构仿真的示意图。

[0048] 图 36 示出根据本发明一个实施例的、阐明机器的前端、调度器、执行单元和中央标志寄存器的示意图。

[0049] 图 37 示出如本发明的实施例所实现的中央标志寄存器仿真过程的示意图。

[0050] 图 38 示出仿真帐户设置中的中央标志寄存器行为的过程 3800 的步骤的流程图。

具体实施方式

[0051] 虽然结合一个实施例描述本发明，但是并非意在将本发明限制在本文中所阐述的具体形式。相反，意在覆盖如可合理地包括在如附随的权利要求所限定的本发明的范围内的这类替代物、修改和等同物。

[0052] 在下面的详细描述中阐述大量的具体细节，诸如具体的方法顺序、结构、元件和连接。然而，将理解，不必利用这些和其他具体细节来实践本发明的实施例。在其他情况下，省略或者不详细描述已知的结构、元件或连接，以避免对本描述产生不必要的遮蔽。

[0053] 说明书内对“一个实施例”或“实施例”的引用意在指示结合实施例所描述的特定的特征、结构或特性包括在本发明的至少一个实施例中。出现在说明书内各处的短语“在一个实施例中”未必指代同一个实施例、或与其他实施例互斥的独立的或可替代的实施例。此外，描述了一些实施例将展现出而其他实施例中没有的各种特征。类似地，描述了一些实施例要求而其他实施例不要求的各种要求。

[0054] 接下来详细描述的一些部分以对计算机存储器内的数据位的操作的过程、步骤、逻辑块、处理和其他符号表现的方式加以呈现。这些描述和表现是数据处理领域的技术人员使用以将其工作实质的最有效地传达给本领域其他技术人员的手段。本文中的过程、计算机执行步骤、逻辑块、处理等通常被设想为致使所希望的结果的自相一致序列的步骤或指令。步骤是要求对物理量的物理操纵的那些步骤。虽然不是必要的，但是通常这些量采用计算机可读存储介质的电或磁信号的形式，其能够被存储、转移、组合、比较和计算机系统中的其他操纵。主要因为一般使用的原因，有时已证明其指代如位、值、元素、符号、字符、术语、数字等是方便的。

[0055] 然而，应该记住，所有这些和相似的术语将与合适的物理量相关联，并且仅是适用于这些量的方便标签。除非特别陈述，否则如从下面的讨论中显而易见的，将理解贯穿本发明，利用诸如“处理”或“访问”或“写”或“存储”或“复制”等的讨论指代计算机系统或相似电子计算设备的动作和过程，所述计算机系统或相似电子计算操纵计算机系统寄存器和存储器以及其他计算机可读介质内的表示为物理（电子）量的数据并将其转换为类似地表示为计算机系统存储器或寄存器或其他这类信息存储、传输或显示设备内的物理量的其他数据。

[0056] 图 1 示出用于通过使用寄存器模板将指令分组到块中并跟踪指令之间的依赖关系的概略图。

[0057] 图 1 示出具有头部和主体的指令块。从一组指令创建块。块包括封装该组指令的实体。在本实施例的微处理器中，抽象级别上升到块而不是个别的指令。块被处理用于分派，而不是个别指令。每个块采用块编号加以标注。从而使机器的乱序管理工作简化。一个关键特征是找到管理大量将被处理的指令的方式，而不显著增加机器的管理开销。

[0058] 本发明的实施例通过实现指令块、寄存器模板和继承向量来达成该目标。在图 1 示出的块中，块的头部列出并封装了块的指令的所有源和目的地以及那些源来自于何处（例如来自哪些块）。头部包括更新寄存器模板的目的地。包括在头部中的源将与存储在寄存器模板中的块编号连接在一起。

[0059] 将被乱序处理的指令的数目决定了乱序机器的管理复杂度。乱序指令越多导致更大的复杂度。在处理器的乱序分派窗口中，源需要与先前指令的目的地相比较。

[0060] 如图 1 所示，寄存器模板具有针对每个寄存器从 R0 到 R63 的字段。块将其各自块编号写入到与块目的地相对应的寄存器模板字段中。每个块从寄存器模板读取表示其寄存器源的寄存器字段。当块撤回并将其目的地寄存器内容写入到寄存器文件中时，其编号将从寄存器模板中擦除。这意味着可从寄存器文件本身将那些寄存器读取为源。

[0061] 在所呈现的实施例中，每当分配块时每个机器循环更新寄存器模板。随着新模板更新生成，寄存器模板的先前快照被存储到阵列（例如图 2 中所示的寄存器图）中，每块一个。该信息被保留直到相应的块被撤回。这允许机器从误预测中恢复并非常快地清除（例如通过获得最后已知的依赖关系状态）。

[0062] 在一个实施例中，可以通过仅存储连续快照之间的 delta（快照之间的递增量）来压缩存储在寄存器图中的寄存器模板（从而节省存储空间）。以该方式，机器获得收缩的寄存器图。可以通过仅存储用于具有分支指令的块的模板来进一步获得压缩。

[0063] 如果除分支误预测之外需要恢复点，那么在分支恢复点首先获得恢复，然后可以无分配指令（但不执行它们）而重建状态直到机器到达恢复点之后的寻求地 (sought)。

[0064] 应该注意，在一个实施例中，如本文中所使用的术语“寄存器模板”与早先提交的共同转让的专利申请“通过使用由可分区引擎实例化的虚拟代码执行指令序列代码块”中所描述的术语“继承向量”同义，该专利申请由 Mohammad Abdallah 在 2012 年 3 月 23 日提交，序列号为 13428440，该专利申请的全部内容通过引用并入本文。

[0065] 图 2 示出根据本发明一个实施例的寄存器图、源图以及指令图的概略图。该图示出调度器体系架构（例如，具有源图、指令图、寄存器图等）的一个实施例。通过组合或分离上述结构的一个或多个来达成相同功能的调度器体系架构的其他实施例是可能的。

[0066] 图 2 用图表示出支持寄存器模板的操作和机器状态的保留的功能实体。图 2 的左手边示出寄存器图 T0 到 T4，箭头指示从一个寄存器模板 / 继承向量到下一个的信息的继承。寄存器图、源图和指令图各自包括用于存储与指令块相关的信息的数据结构。图 2 还示出具有头部的示例性指令块，以及指令块如何包括用于机器的寄存器的源和目的地。关于块所引用的寄存器的信息存储在寄存器图数据结构中。关于块所引用的源的信息存储在源图数据结构中。关于块所引用的指令自身的信息存储在指令图数据结构中。寄存器模板 / 继承向量本身包括存储块所引用的继承信息和依赖关系的数据结构。

[0067] 图 3 示出根据本发明一个实施例的、阐明示例性寄存器模板以及来自寄存器模板的信息如何填充源图的示意图。

[0068] 在本实施例中，应该注意，源图的目标是确定何时可以分派特定的块。当块被分派时，它将它的块编号广播到所有剩余的块。用于其他块的源的任何匹配（例如，比较）产生将被设置的就绪位（例如，某其他类型的指示符）。当所有就绪位被设置时（例如，与门），块就准备好被分派。块基于它们所依赖的其他块的而准备就绪被分派。

[0069] 当多个块准备好用于分派时，选择最老的块用于在较年轻的块之前分派。例如，在一个实施例中，可以使用第一查找电路来基于接近提交指针查找最老的块，并基于相对接近提交指针查找后续块（例如，作用于每个块的就绪位）。

[0070] 仍然参考图 3，在该示例中，检查在块 20 到达时创建的寄存器模板快照。如上所

述，寄存器模板具有用于从 R0 到 R63 的每个寄存器的字段。块将它们各自的块编号写入到与块目的地相对应的寄存器模板字段。每个块从寄存器模板读取表示其寄存器源的寄存器字段。第一数字是写入到寄存器的块，第二数字是该块的目的地编号。

[0071] 例如，当块 20 到达时，它读取寄存器模板的快照并在寄存器模板中查找它自身的寄存器源，以确定写入到它的每个源的最新的块并根据其目的地对先前的寄存器模板快照所做的更新来填充源图。随后的块将采用它们自身的目的地更新寄存器模板。这在图 3 的左下角示出，其中块 20 填充它的源：源 1、源 2、源 3、一直到源 8。

[0072] 图 4 示出阐明用于源图的依赖关系广播的第一实施例的示意图。在该实施例中，每列包括一个指令块。当块被分配时，它在它的源所依赖的所有那些块的列中做标记（例如通过写 0）。当任何其他块被分派时，它的编号被跨与该块相关的准确列广播。应该注意，写 1 是缺省值，指示对该块没有依赖关系。

[0073] 当块中的所有就绪位就绪时，该块非分派并且其编号被广播回所有剩余的块。该块编号与存储在其他块的源中的所有编号进行比较。如果存在匹配，则用于该源的就绪位被置位。例如，如果在源 1 上广播的块编号等于 11，那么用于块 20 的源 1 的就绪位将被置位。

[0074] 图 5 示出阐明用于源图内的依赖关系广播的第二实施例的示意图。该实施例由源来组织而不是由块来组织。这通过跨源图数据结构的源 S1 到 S8 来示出。以与上面图 4 所描述的类似的方式，在图 5 的实施例中，当块中的所有就绪位就绪时，该块被分派并且其编号被广播回所有剩余的块。该块编号与存储在其他块的源中的所有编号进行比较。如果存在匹配，则用于该源的就绪位被置位。例如，如果在源 1 上广播的块编号等于 11，那么用于块 20 的源 1 的就绪位将被置位。

[0075] 图 5 的实施例还示出如何仅在提交指针和分配指针之间的块上使能该比较。所有其他块无效。

[0076] 图 6 示出根据本发明一个实施例的、阐明针对从提交指针开始分派以及广播相应的端口分配选择就绪块的示意图。源图数据结构示出在图 6 的左侧。指令图数据结构示出在图 6 的右侧。选择器阵列示出在源图和指令图之间。在该实施例中，选择器阵列经由四个分派端口 P1 到 P4 每循环分派四个块。

[0077] 如上所述，块被选择用于从提交指针围绕分配指针分派（例如尝试首先荣幸分派最老的块）。选择器阵列被使用以找到从提交指针开始的前四个就绪块。期望的是分派最老的就绪块。在一个实施例中，可以使用加法器树结构来实现选择器阵列。这将在下面的图 7 中描述。

[0078] 图 6 还示出选择器阵列如何被耦连到穿过指令图中的条目的四个端口中的每一个。在该实施例中，端口随端口使能而耦连，使四个端口中的一个能够被激活，用于该指令图条目向下穿过分派端口到执行单元上。此外，如上所述，所分派的块被广播回源图。所选择的块的块编号被广播回。这在图 6 的更右侧示出。

[0079] 图 7 示出根据本发明一个实施例的、实现图 6 中所描述的选择器阵列所使用的加法器树结构。所描绘的加法器树结构实现选择器阵列的功能。加法器树挑选前四个就绪块并将它们装载到四个可用端口以用于分派（例如读端口 1 到读端口 4）。不使用仲裁。用来特别使能具体端口的实际逻辑在条目编号 1 中明确示出。为了清楚，未在其他条目中示出

逻辑。以该方式，图 7 示出如何实现对每个特定端口的直接选择以用于块分派的一个具体实施例。然而，应该注意，可替代地，可以实现使用优先编码器的实施例。

[0080] 图 8 更详细地示出选择器阵列加法器树的示例性逻辑。在图 8 的实施例中，示出用于范围超过位 (range exceed bit) 的逻辑。范围超过位确保不多于四个块将被选择用于分派，如果第五个块就绪而前四个也就绪，范围超过位将不允许第五个块被分派。应该注意，和数位为 S0 到 S3，均用来使能分派端口以及传播到串行实现中的下一加法器级。

[0081] 图 9 示出根据本发明一个实施例的、用于实现选择器阵列的加法器树的并行实现方案。并行实现方案不将总和从每个加法器转发到下一个。在并行实现方案中，每个加法器使用多输入加法实现方案（诸如多输入进位保留加法器树）来直接使用所有其必需的输入。例如，加法器“X”对所有先前的输入求和。为了执行更快计算次数（例如单个循环），该并行实现方法是理想的。

[0082] 图 10 示出根据本发明一个实施例的、阐明通过使用进位保留加法器可如何实现来自图 9 的加法器 X 的示例性示意图。图 10 示出可以在单个循环中对 32 个输入求和的结构。该结构使用 4:2 进位保留加法器组合而成。

[0083] 图 11 示出根据本发明的用于针对从提交指针开始调度和使用选择器阵列加法器掩蔽就绪位的掩蔽实施例。在该实现方案中，选择器阵列加法器尝试选择前四个就绪块，以从提交指针开始潜在地围绕分配指针分派。在该实现方案中，使用多输入并行加法器。此外，在该实现方案中利用这些循环缓冲器的源。

[0084] 图 11 示出如何采用两个掩码（单独地或分开地）将就绪位求与 (AND) 在一起以及如何将其应用到两个并行的加法器树。通过使用两个加法器树并比较四个的阈值来选择前四个。“X”标记表示“针对该加法器树从选择阵列中排除”，因此“X”值为 0。另一方面，“Y”标记表示“针对该加法器树确实包括在选择器阵列中”，因此“Y”值为 1。

[0085] 图 12 示出根据本发明一个实施例的、寄存器模板如何填充寄存器图条目的概略图。

[0086] 如上所述，通过寄存器模板填充寄存器图条目。寄存器图按序针对每个块存储寄存器模板的快照。当推测无效时（例如分支误预测），寄存器图具有无效推测点之前的最新有效快照。机器可以通过读取该寄存器图条目并将其加载在寄存器模板的底部来将其状态退回到最后的有效快照。寄存器的每个条目显示所有的寄存器继承状态。例如在图 12 的实施例中，如果用于块 F 的寄存器图无效，则机器状态可被退回到较早的最后有效的寄存器模板快照。

[0087] 图 13 示出根据本发明一个实施例的、用于减少的寄存器图印记的第一实施例。可通过仅存储那些包含分支指令的寄存器图模板快照来减少存储寄存器图条目所需要的存储器的数量。当发生例外时（例如推测无效、分支误预测等），可从分支指令中重建在例外之前发生的最后的有效快照。在从在例外之前直到例外的分支取回指令以建造最后的有效快照。指令被取回但未被执行。如图 13 中所示，仅那些包括分支指令的快照被保存在减少的寄存器图中。这大大减少了存储寄存器模板快照所需要的存储器的数量。

[0088] 图 14 示出根据本发明一个实施例的、用于减少的寄存器印记的第二实施例。可以通过仅存储快照的顺序子集（例如每四个快照一个）来减少存储寄存器图条目所需要的存储器的数量。可以使用比完整连续快照相对较小的存储器数量将连续快照之间改变存储为

从原始快照的“delta”。当发生例外时（例如推测无效、分支误预测等），可从在例外之前发生的原始快照中重建最后的有效快照。使用从在例外之前发生的原始快照的“delta”和连续快照来重建最后的有效快照。初始原始状态可以累积 delta 以到达所要求的快照的状态。

[0089] 图 15 示出根据本发明一个实施例的、快照之间的 delta 的示例性格式。图 15 示出原始快照和两个 delta。在一个 delta 中，仅 R5 和 R6 是被 B3 所更新的寄存器。其余的条目不变。在另一个 delta 中，仅 R1 和 R7 是被 B2 所更新的寄存器。其余的条目不变。

[0090] 图 16 示出根据本发明一个实施例的、用于根据指令的块的分配创建寄存器模板快照的过程的示意图。在该实施例中，图 16 的左侧示出两个解复用器，并且图 16 的上部是快照寄存器模板。图 16 示出用于从先前寄存器模板创建后续寄存器模板（例如串行实现方案）的示意图。

[0091] 该串行实现方案示出如何基于指令的块的分配创建寄存器模板快照。那些快照用来捕获最新的寄存器体系架构状态更新，其用于依赖关系跟踪（例如，如图 1 到 4 中所描述的）以及更新寄存器图用于处理误预测 / 例外（例如，如图 12 到 15 中所描述的）。

[0092] 解复用通过选择哪个传入源通过来工作。例如，寄存器 R2 将在第二输出处解复用到 1，而 R8 将在第七输出处解复用为 1 等等。

[0093] 图 17 示出根据本发明一个实施例的、用于根据指令的块的分配创建寄存器模板快照的过程的另一示意图。图 17 的实施例还示出从先前寄存器模板创建后续寄存器模板。图 17 的实施例还示出寄存器模板块继承的示例。该图示出如何从分配的块编号更新寄存器模板。例如，块 Bf 更新 R2、R8 和 R10。Bg 更新 R1 和 R9。虚线箭头指示从前一个快照继承值。该过程一直继续到块 Bi.。这样，例如，因为没有快照更新寄存器 R7，所以它的原始值 Bb 将一直传播下去。

[0094] 图 18 示出根据本发明一个实施例的、用于实现从先前寄存器模板创建后续寄存器模板的串行实现方案的硬件的概略图。解复用器用来控制两个输入多路复用器系列两个块编号中的哪个将被传播到下一级。其可以是来自先前级的块编号或者是当前块编号。

[0095] 图 19 示出根据本发明一个实施例的、用于实现从先前寄存器模板创建后续寄存器模板的并行实现方案的硬件的概略图。该并行实现方案使用特别编码的多路复用器控制以从先前寄存器模板创建后续寄存器模板。

[0096] 图 20 示出根据本发明一个实施例的、用于指令基于块的执行的硬件以及其如何与源图、指令图、寄存器模板以及寄存器图工作的概略图。

[0097] 在该实现方案中，分派器中的分配器调度器接收由机器的前端所取回的指令。这些指令以我们较早描述的方式穿过块队形。如较早所描述的，块产生寄存器模板并且这些寄存器模板用来填充寄存器图。源被从源图传递到寄存器文件层级，并且以上文所描述的方式广播回源图。指令图向执行单元传递指令。当指令所需要的源从寄存器文件层级来到时，指令被执行单元所执行。这些经执行的指令随后被转移出执行单元并回到寄存器文件层级。

[0098] 图 21 示出根据本发明一个实施例的组块体系架构的示例。组块的重要性在于它通过使用示出的四个多路复用器，减少了到从 4 到 1 的每个调度器条目的写端口的数目，而仍密集地包装所有条目而没有形成磁泡 (bubble)。

[0099] 组块的重要性可通过下面的示例（例如，注意每个循环中块的分配开始于上部位置，在这种情况下为 B0）看出。假定在循环 1 中，三个指令块被分配到调度器条目（例如，三个块将占用调度器中的前 3 个条目）。在下一循环（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创建磁泡，调度器阵列条目需被建造为支持四个写端口。在这功耗、时序、面积等方面是高代价的。上面的组块结构简化了所有的调度器阵列，其通过在分配到阵列之前使用多路复用结构使调度器阵列仅具有一个写端口。在上述示例中，循环 2 中的 B0 将被最后的复用器所选择，而循环 2 中的 B1 将被第一个复用器所选择（例如从左到右进行）。

[0100] 以该方式，每个针对条目的组块仅需要每条目一个写端口和每条目四个读端口。在成本上存在折衷，因为必须实现多路复用器，然而该成本超过因不必实现每条目四个写端口所节约的成本的许多倍，因为可能存在非常多的条目。

[0101] 图 21 还示出中间分配缓冲区。如果调度器阵列不能接受发送给它们的所有组块，则组块可被临时存储在中间分配缓冲区中。当调度器阵列具有空闲空间时，组块将被从中间分配缓冲区中转移到调度器阵列。

[0102] 图 22 示出根据本发明一个实施例的、如何根据线程的块编号和线程 ID 对线程进行分配的描述。块经由上文所描述的组块实现方案被分配到调度器阵列。每个线程块使用块编号在它们之间维持相继顺序。来自不同线程的块可交错（例如用于线程 Th1 的块和用于线程 Th2 的块在调度器阵列中交错）。以该方式，来自不同线程的块存在于调度器阵列内。

[0103] 图 23 示出根据本发明一个实施例的、使用指向物理存储位置的线程指针映射以管理多线程执行的调度器的实现方案。在该实施例中，线程的管理通过对线程映射的控制来实现。例如，此处图 23 示出线程 1 映射和线程 2 映射。映射跟踪个体线程的块的位置。映射 2 中的条目，映射中的物理存储位置条目被分配到属于该线程的块。在该实现方案中，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 除以 2（例如超过可用空间）。分配计数器具有可调整的阈值，以实现来自池中的全部条目的分配的公正性。分配计数器可以阻止一个线程使用所有的可用空间。

[0104] 图 24 示出根据本发明一个实施例的、使用基于线程的指针映射的调度器的另一实现方案。图 24 示出提交指针和分配指针之间的关系。如所示的，每个线程具有提交指针和分配指针，箭头示出用于线程 2 的真实指针可如何环绕分配块 B1 和 B2 的物理存储，但其不能分配块 B9，直到用于线程 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 的右侧示出块的分配和提交指针之间随着其围绕逆时针方向移动的关系。

[0105] 图 25 示出根据本发明一个实施例的、对线程的执行资源的动态的基于日历的分配的示意图。可基于每个线程的向前进度使用分配计数器动态控制公正性。如果两个线程均在进行大量向前进度，则两个分配计数器均被设置在相同的阈值（例如，9）。然而，如果一个线程向前进度迟缓，诸如遭遇 L2 高速缓存未命中或这类事件，则计数器的阈值的比例可被调整有利于仍在进行大量向前进度的线程。如果一个线程停止或暂停（例如在等待 OS 或 IO 响应的等待或自旋状态），则比例可被完全调整到另一线程，除了被保留用于暂停线程以发信号通知释放等待状态的单个返回条目。

[0106] 在一个实施例中，进程从50％ :50％的比例开始。一旦L2高速缓存未命中对块22的检测，管线的前端停止任何到管线的进一步取回或到线程2的块的调度器的分配。一旦线程2的块从调度器中退出，那些条目将可用于线程1分配，直到达到线程分配的新的动态比例的指针。例如，3个最近退出的线程2之外的块将被返回到池中用于分配到线程1而不是线程2，使线程1对线程2的比例为75％ :25％。

[0107] 应该注意，如果没有硬件机制（例如通过传递停止的线程2的块由线程1的块）绕过管线前端线程2的块，管线前端线程2的块的停止可能要求从管线前端清除那些块。

[0108] 图26示出根据本发明一个实施例的双分派过程。多分派通常包含多次分派块（其内具有多个指令），这样块内的不同指令可各自通过执行单元执行。一个示例将是地址计算指令的分派，其后跟随消耗结果数据的相继分派。另一个示例将是浮点操作，其中第一部分执行为定点操作，并且第二部分被执行以通过实施舍入(rounding)、标志生成／计算、指数调整等完成操作。块作为单个条目被原子地分配、提交和退出。

[0109] 多分派的主要益处在于它避免了将多个单独块分配到机器窗口中，从而使机器窗口实际上更大。较大的机器窗口意味着更多的机会用于优化和重新排序。

[0110] 看图26的左下角，描绘了指令块。该块不能在单个循环中被分派，因为加载地址计算和加载来自高速缓存／存储器的返回数据之间存在延迟。因此该块首先采用其保存为暂时状态的中间结果被分派（其结果被飞速传送到第二分派，对体系架构状态不可见）。第一分派发送两个组件1和2，其用于地址计算和LA的分派。第二分派发送组件3和4，其为根据加载来自高速缓存／存储器的返回数据的加载数据的执行部分。

[0111] 看图26的右下角，描绘了浮点乘累加操作。在硬件没有足够进入源带宽以在单个阶段中分派操作的情况下，使用双分配，如乘累加图所示的。第一分派是如所示的定点乘。第二分派是如所示的浮点加法舍入。当这些所分派的指令均执行时，它们有效地实施浮点乘／加。

[0112] 图27示出根据本发明一个实施例的双分派暂时乘累加器。如图27中所示，第一分派是整数32位乘，并且第二分派时整数累加。第一分派和第二分派之间传达的状态（乘的结果）是暂时的，并且在体系架构上不可见。在一个实现方案中，暂时存储可以保存一个乘法器以上的结果，并且可以标记它们以识别相对应的乘累加对，从而允许以任意方式（例如交错等）被分派的多个乘累加对的混合。

[0113] 注意，其他指令可以使用相同的硬件用于它们的实现（例如，浮点等）。

[0114] 图28示出根据本发明一个实施例的双分派体系架构可视状态乘加器。第一分派是单精度乘，并且第二分派时单精度加。在该实现方案中，第一分派和第二分派之间传达的状态信息（例如，乘的结果）在体系架构上可见，因为该存储是体系架构状态寄存器。

[0115] 图29示出根据本发明一个实施例的用于在经分组的执行单元过程上执行的指令块的取回和形成的概略图。本发明的实施例利用过程，凭借其指令被硬件或动态转换器／JIT取回并形成为块。块中的指令被组织，这样块中早期指令的结果为块中的后续指令提供源。这通过指令块中的虚线箭头示出。该性质使块能够高效地执行在执行块的堆栈执行单元上。即使指令并行执行，诸如如果它们共享相同的源（未在该图中明确示出），也可将指令分组。

[0116] 在硬件中形成块的一个替代性方案是在形成指令对、三重指令、四重指令等的软

件中形成它们（静态地或在运行时）。

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

[0118] 图 30 示出根据本发明一个实施例的指令分组的示例性示意图。在图 30 的实施例中示出第三辅助操作的两个指令。图 31 左侧的指令块包括上半块 /1 槽和下半块 /1 槽。从顶部向下的垂直箭头指示源进入到块中，而从底部向下的垂直箭头指示目的地回到存储器。从图 3 的左侧继续到右侧，示出可能的不同指令组合。在该实现方案中，每半块可接收三个源并且可传递两个目的地。OP1 和 OP2 是常规操作。辅助 OP 是辅助的操作，诸如逻辑、移位、移动、符号扩展、分支等。将块分为两半的益处在于允许使各自半个独立地分派在其自身上或基于依赖关系解析另外动态地合而为一块（用于端口利用或因为资源约束）的益处，这样具有执行次数的更好利用，同时使与一块相对应的 2 个半块允许机器将 2 个半块的复杂度抽象为像一块那样管理（即在分配和退出上）。

[0119] 图 31 示出根据本发明一个实施例的、块堆栈内的半块对如何映射到执行块单元。如在执行块中所示的，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，这样第一半块在槽 1 上执行，并且第二半块在槽 2 上执行。目标是如果每半块的指令组不依赖于另半块则允许两个半块独立地分派。从顶部进入执行块的成对的箭头是源的两个 32 位的字。离开执行块向下的成对的箭头是目的地的两个 32 位的字。从图 31 的左侧到右侧，示出能够被堆栈在执行块单元上的指令的不同示例性组合。

[0120] 图 31 的上部概述了半块对如何在整块上下文或任何半块上下文中执行。每个执行块具有两个槽 / 半块并且每个半块 / 执行槽执行单个、成对的或三个经分组的操作。存在四种类型的块执行类型。第一种是并行半块（其允许一旦其自身的源就绪则每半块独立地执行，但如果两个半块同时就绪则两个半块仍可作为一个块执行在一个执行单元上）。第二种是原子并行半块（其指的是可以并行执行的半块，因为两个半块之间不存在依赖关系但是它们被迫作为一个块一起执行，因为两个半块之间的资源共享使两个半块在每个执行块中可用的资源的约束下优先或需要一起原子地执行）。第三种类型是原子串行半块（其要求第一半块将数据转发到第二半块，通过采用或不采用内部存储的暂时转发）。第四种类型是顺序半块（如在双分派中的），其中第二半块依赖于第一半块并在比第一个靠后的循环上被分派，并且通过被堆栈用于依赖关系解析的外部存储转发数据，与双分派的情况类似。

[0121] 图 32 示出根据本发明一个实施例的、描绘中间块结果存储作为第一级别寄存器文件的示意图。寄存器的每个组表示指令的块（表示两个半块），在其中 32 位结果以及 64 位结果可通过使用两个 32 位寄存器支持一个 64 位寄存器来加以支持。每块存储假定虚拟块存储，这意味着来自不同块的两个半块可以写入同一个虚拟块存储。两个半块的组合结果的存储构成一个虚拟块存储。

[0122] 图 33 示出根据本发明一个实施例的奇数 / 偶数端口调度器。在该实现方案中，结果存储是非对称的。某些结果存储是每半块三个 64 位结果寄存器，而其他的是每半块一个 64 位结果寄存器，然而，可替代的实现方案可以使用每半块对称存储，并且此外还可以采用如图 32 中所描述的 64 位和 32 位分区。在这些实施例中，存储被每半块指派，而不是每块。该实现方案通过使用端口作为奇数或偶数减少了用于分派所需要的端口的数目。

[0123] 图 34 示出图 33 的更详细版本，其中示出四个执行单元从调度器阵列接收结果并将输出写入到临时寄存器文件段。端口在奇数和偶数间隔附上。调度阵列的左侧示出块编

号，右侧示出半块编号。

[0124] 每个核心具有到调度阵列的偶数和奇数端口，其中每个端口连接到奇数或偶数半块位置。在一个实现方案中，偶数端口和它们相对应的半块可以驻留在与奇数端口和其对应的半块不同的核心内。在另一个实现方案中，奇数和偶数端口将如该图中所示的跨多个不同的核心被分配。如 Mohammad Abdallah 在 2012 年 3 月 23 日提交的、题为“通过使用由可分区引擎所实例化的虚拟代码执行指令序列代码块”的、序列号为 13428440 的共同转让的专利申请中所描述的，核心可以是物理核心或虚拟核心，该专利申请的全部内容通过引用并入本文。

[0125] 在某些类型的块中，块的一半可独立于块的另一半加以分派。在其他类型的块中，块的两个半块均需要同时分派到相同的执行块单元。在另外其他类型的块中，块的两个半块需要被顺序分派（第二半块在第一半块之后）。

[0126] 图 35 示出根据本发明一个实施例的描绘访客标志体系架构仿真的示意图。图 35 的左侧示出具有五个标志的中央式标志寄存器。图 35 的右侧示出具有分布的标志寄存器的分布式标志体系架构，其中标志分布在寄存器自身之中。

[0127] 在体系架构仿真期间，需要分布式标志体系架构仿真中央式访客标志体系架构的行为。分布式标志体系架构还可通过使用多个独立的标志寄存器加以实现，独立的标志寄存器和与数据寄存器相关联的标志字段不同。例如，数据寄存器可实现为 R0 到 R15，而独立标志寄存器可实现为 F0 到 F3。在该情况下，那些标志寄存器不与数据寄存器直接相关联。

[0128] 图 36 示出根据本发明一个实施例的、阐明机器的前端、调度器、执行单元和中央标志寄存器的示意图。在该实现方案中，前端基于指令更新访客指令标志的方式将进入的指令分类。在一个实施例中，访客指令被分为 4 个本地指令类型，T1、T2、T3 和 T4。T1-T4 是指示每个访客指令类型更新哪些标志字段的指令类型。访客指令类型基于其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 示出如本发明的实施例所实现的中央标志寄存器仿真过程的示意图。图 37 中的作用物包括最新更新类型表、重命名表扩展、物理寄存器以及分布式标志寄存器。现在通过图 38 的流程图描述图 37。

[0130] 图 38 示出仿真访客设置中的中央标志寄存器行为的过程 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）基于进入的指令更新访客指令标志的方式将进入的指令进行分类。在一个实施例中，访客指令被分为四种标志体系架构类型，T1、T2、T3 和 T4。T1-T4 是指示每个访客指令类型更新哪些标志字段的指令类型。访客指令类型基于其类型更新不同的访客标志。例如，逻辑访客指令更新 T1 类型标志，移位访客指令更新 T2 类型标志，算术访客指令更新 T3 类型标志，并且特殊访客指令更新 T4 类型标志。应该注意，访客指令可以是体系架构指令表示，而本地的可以是机器内部执行的（例如微代码）。可替代地，访客指令可以是来自经仿真的体系架构（例如 x86、java、ARM 代码等）的指令。

[0132] 在步骤 3802 中，将那些指令类型更新其各自访客标志的顺序记录在最新更新类型表数据结构中。在一个实施例中，该动作由机器的前端实施。

[0133] 在步骤 3803 中，当那些指令类型到达调度器（分配 / 重命名级的有序部分）时，调度器指派与体系架构类型相对应的隐含的物理目的地，并在重命名 / 映射表数据结构中

记录该指派。

[0134] 并且在步骤 3804 中，当后续访客指令到达调度器中的分配 / 重命名级并且该指令想要读取访客标志字段时，(a) 机器确定哪些标志体系架构类型需要被访问以实施读取。(b) 如果发现所有需要的标志为同一最新更新标志类型（例如，如由最新更新类型表所确定的），则读取（映射到该最新标志类型的）相对应的物理寄存器以获得所需要的标志。(c) 如果发现所有需要的标志不是同一最新更新标志类型，则需要从映射到个体最新更新标志类型的相对应的物理寄存器中读取每个标志。

[0135] 以及在步骤 3805 中，从保留其最后被更新的如最新更新标志类型表所跟踪的最新值的物理寄存器中分别读取每个标志。

[0136] 应该注意，如果最新更新类型包括另一类型，那么所有子集类型需要映射到母集类型的相同物理寄存器。

[0137] 在退出时，目的地标志字段与克隆的中央式 / 访客标志体系架构寄存器合并。应该注意，由于本地体系架构利用分布式标志体系架构而不是单个寄存器中央式标志体系架构，所以实施克隆。

[0138] 更新某些标志类型的指令的示例：

[0139] CF、OF、SF、ZR—算术指令和加载 / 写标志指令

[0140] SF、ZF 和条件 CF—逻辑和移位

[0141] SF、ZF—移动 / 加载、EXTR、一些乘

[0142] ZF—POPCNT 和 STREX[P]

[0143] GE—SIMD 指令???

[0144] 读取某些标志的条件 / 预测的示例：

[0145] 0000EQ 相等 Z ＝＝ 1

[0146] 0001NE 不相等或无序 Z ＝＝ 0

[0147] 0010CS b 进位置位，大于或等于，或无序 C ＝＝ 1

[0148] 0011CC c 进位清零，小于 C ＝＝ 0

[0149] 0100MI 减，负，小于 N ＝＝ 1

[0150] 0101PL 加，正或零，大于或等于，无序 N ＝＝ 00110VS 溢出，无序 V ＝＝ 1

[0151] 0111VC 不溢出，非无序 V ＝＝ 0

[0152] 1000HI 无符号更高，大于，无序 C ＝＝ 1 并且 Z ＝＝ 0

[0153] 1001LS 无符号更新或相同，小于或等于，C ＝＝ 0 或 Z ＝＝ 1

[0154] 1010GE 有符号大于或等于，大于或等于 N ＝＝ V

[0155] 1011LT 有符号小于，小于，无序 N ！ ＝ V

[0156] 1100GT 有符号大于，大于 Z ＝＝ 0 并且 N ＝＝ V

[0157] 1101LE 有符号小于或等于，小于或等于，无序 Z ＝＝ 1 或 N ！ ＝ V

[0158] 1110 无 (AL)，总是（无条件），任何标志设置为任何值。

[0159] 以上已参照具体实施例以解释为目的对前述说明书进行了描述。然而，上面的示例性讨论并非意在穷尽或将本发明限制到所公开的精确形式。在上述教导下的许多修改和变形是可能的。实施例的选择和描述是为了最好地解释本发明的原理和其实际应用，从而使本领域其他技术人员能够最好地以如可能适合于预期实际使用的各种修改来利用本发

明和各种实施例。

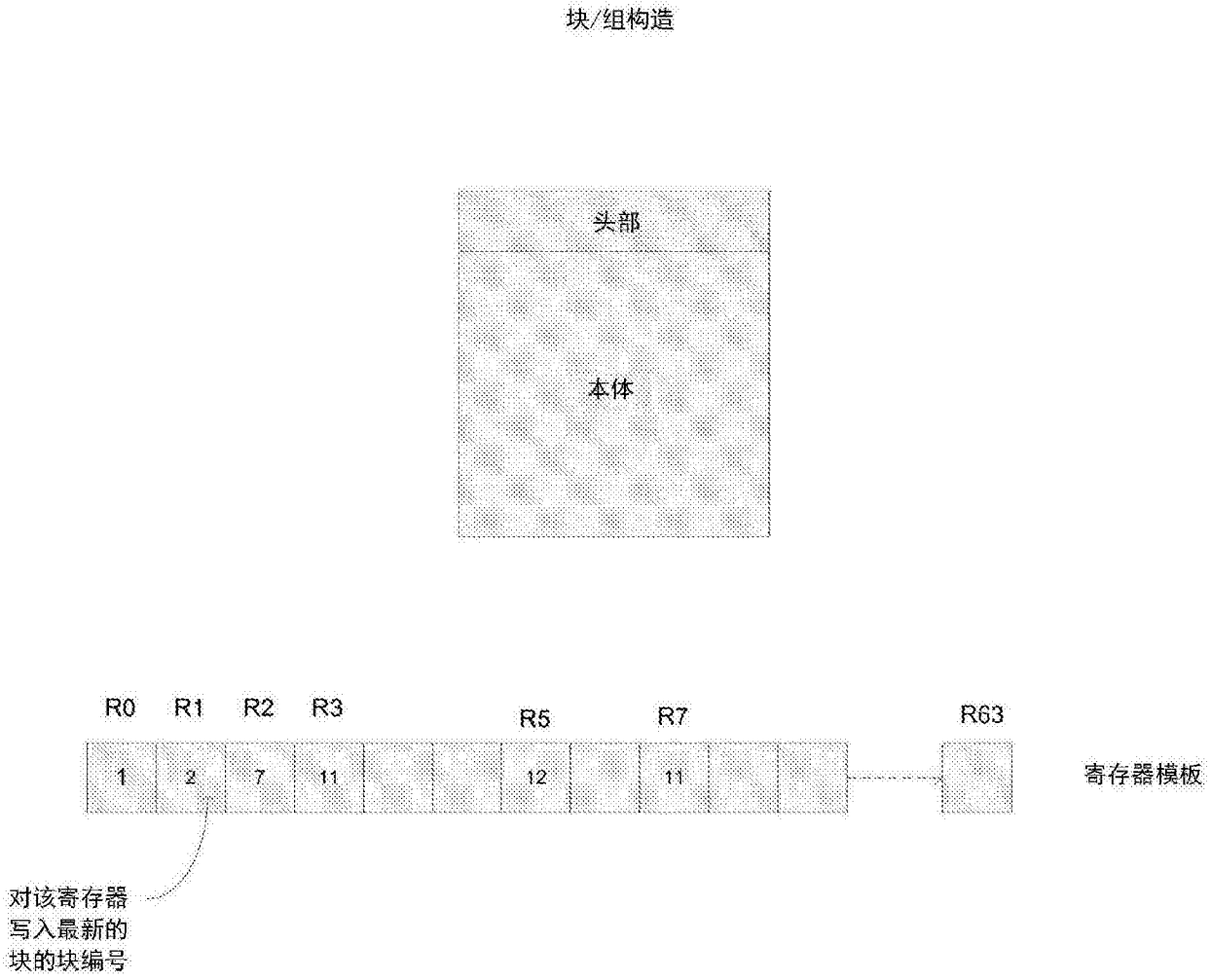

块/组构造
头部
本体
R0
R1
R2
R3
R5
R7
R63
1
2
7
11
12
11
寄存器模板
对该寄存器
写入最新的
块的块编号

图 1

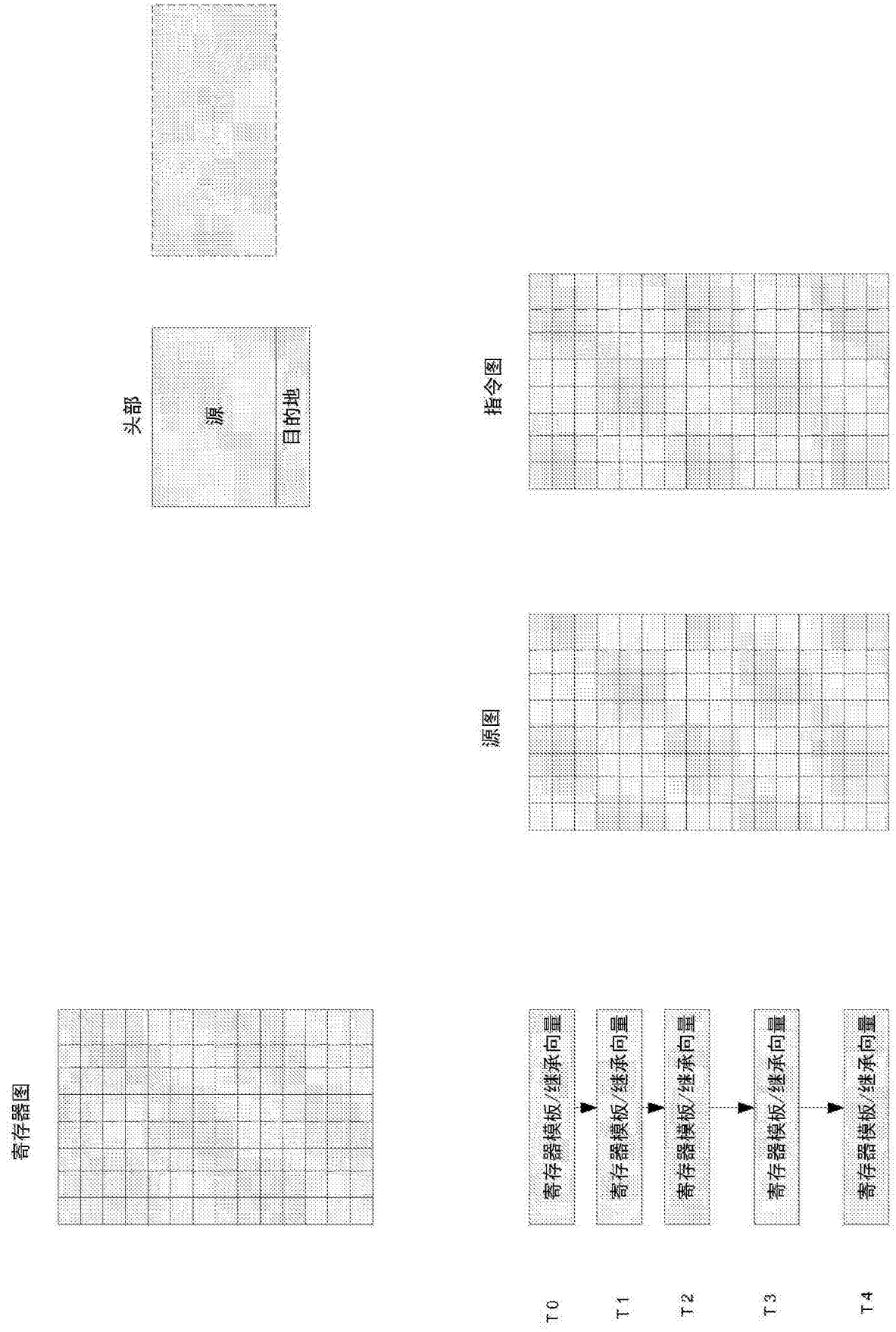

寄存器图
头部
源
目的地
指令图
源图
寄存器模板/继承向量
寄存器模板/继承向量
寄存器模板/继承向量
寄存器模板/继承向量
寄存器模板/继承向量
T0
T1
T2
T3
T4

图 2

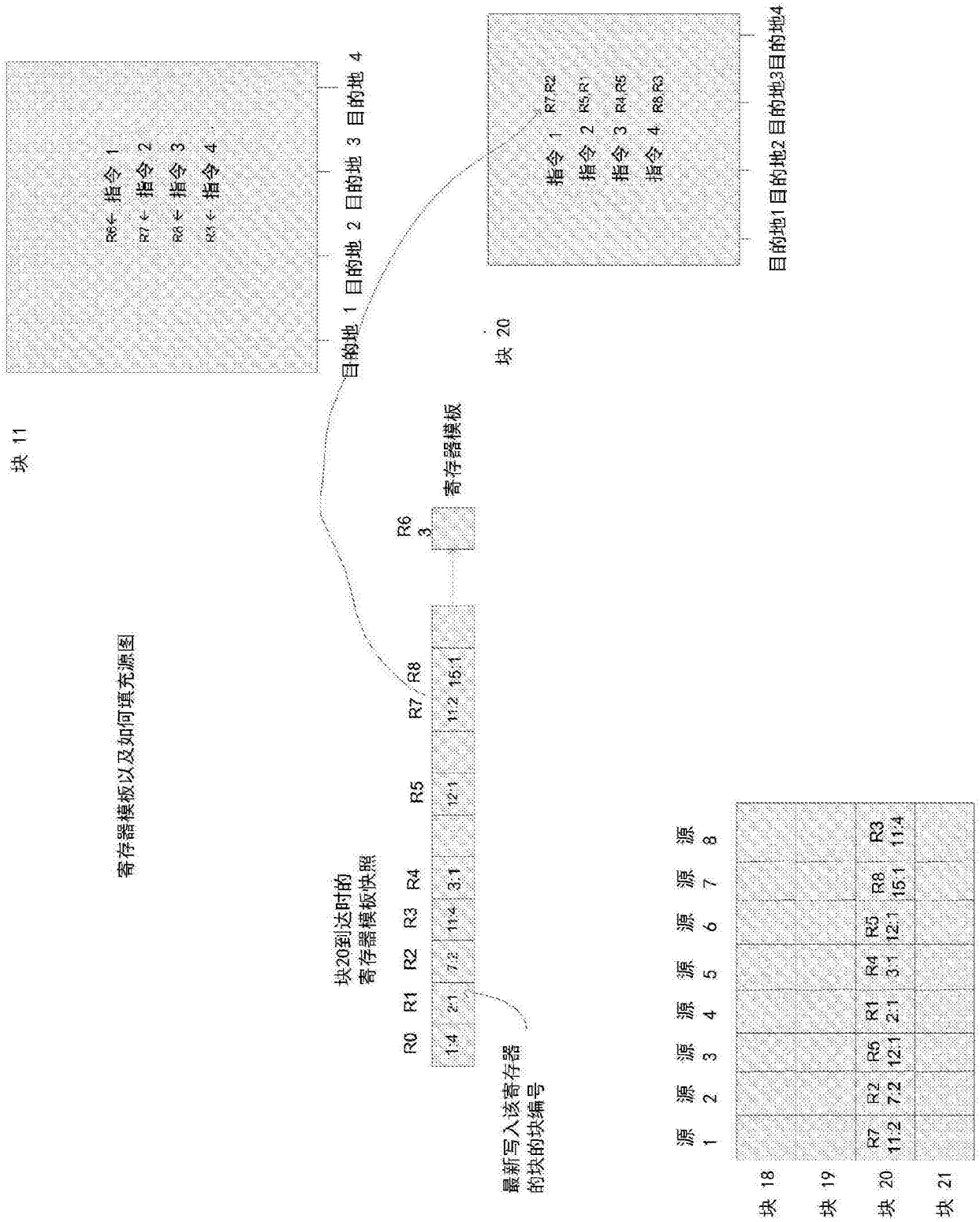

块 11
R6← 指令 1
R7← 指令 2
R8← 指令 3
R3← 指令 4
目的地 1 目的地 2 目的地 3 目的地 4
块 20
指令 1 R7,R2
指令 2 R5,R1
指令 3 R4,R5
指令 4 R8,R3
目的地1 目的地2 目的地3 目的地4
寄存器模板以及如何填充源图
块20到达时的
寄存器模板快照
R0 R1 R2 R3 R4 R5 R7 R8
1:4 2:1 7:2 11:4 3:1 12:1 11:2 15:1
R6
3
寄存器模板
最新写入该寄存器
的块的块编号
源 源 源 源 源 源 源 源
1 2 3 4 5 6 7 8
块 18
块 19
块 20
R7 R2 R5 R1 R4 R5 R8 R3
11:2 7:2 12:1 2:1 3:1 12:1 15:1 11:4
块 21

图 3

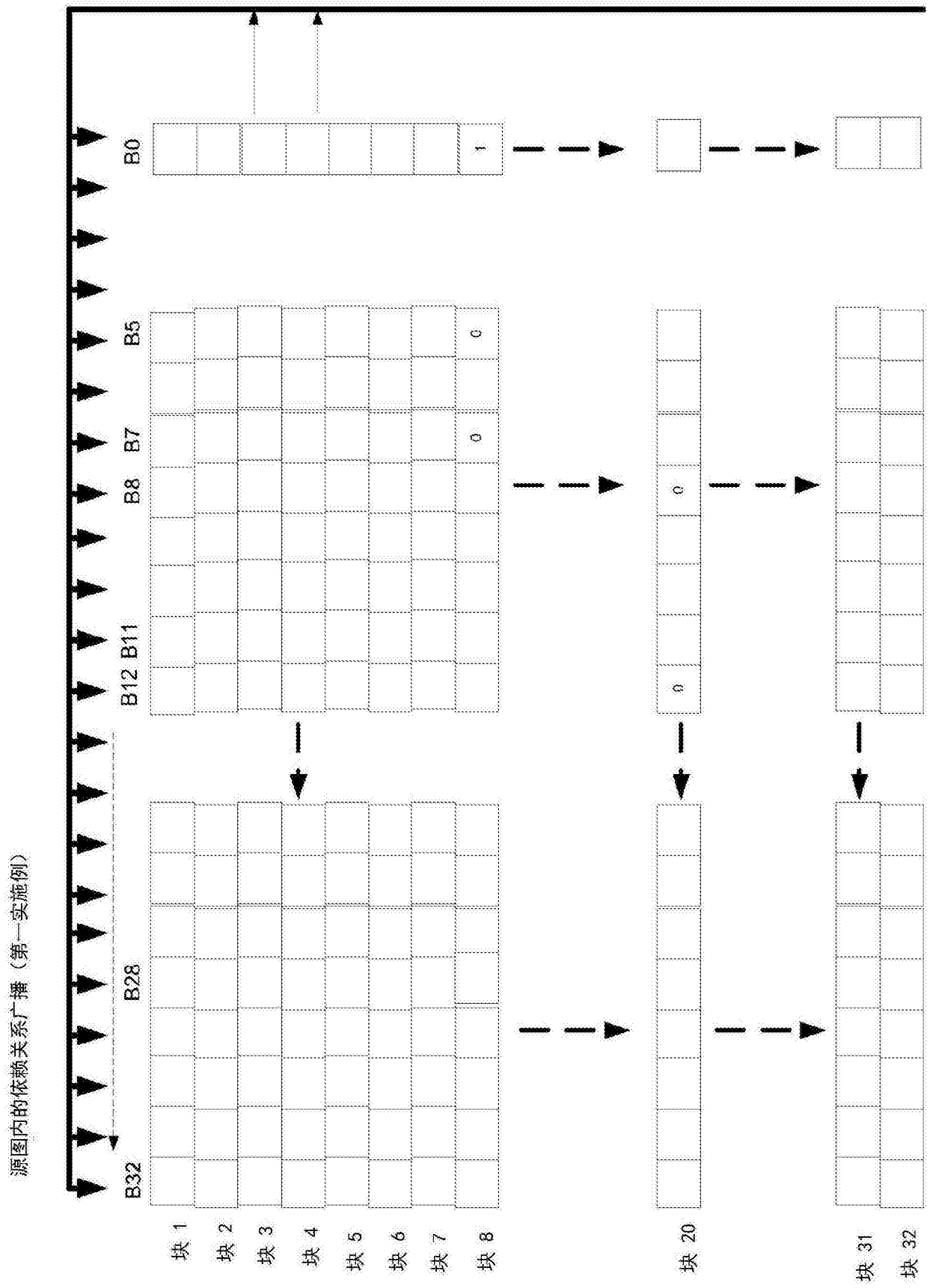

源图内的依赖关系广播（第一实施例）
B0
B5
B7
B8
B11
B12
B28
B32
1
0
0
0
0
块 1
块 2
块 3
块 4
块 5
块 6
块 7
块 8
块 20
块 31
块 32

图 4

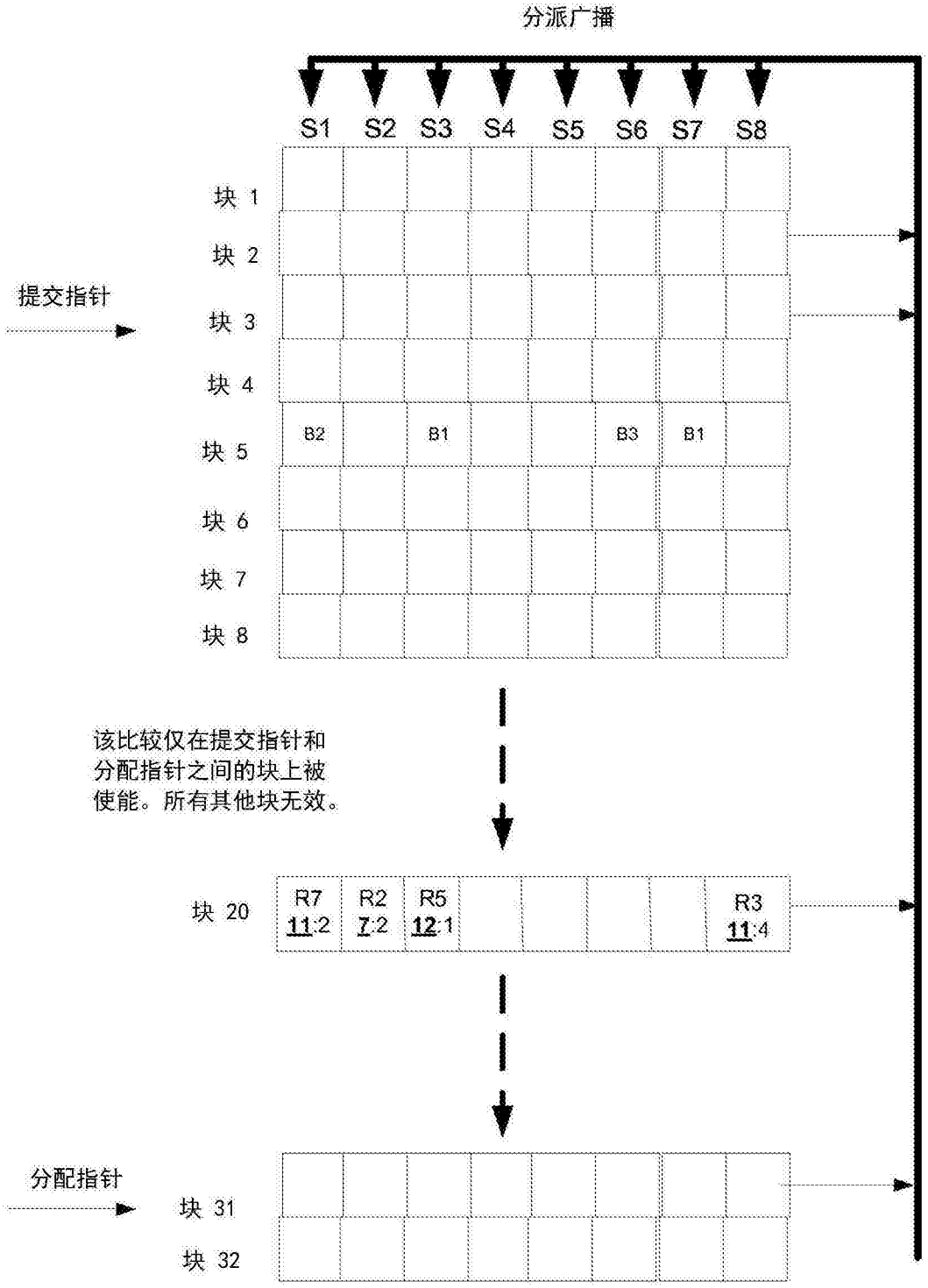

源图内的依赖关系广播（第二实施例）
分派广播
S1
S2
S3
S4
S5
S6
S7
S8
块 1
块 2
提交指针
块 3
块 4
块 5
块 6
块 7
块 8
B2
B1
B3
B1
该比较仅在提交指针和
分配指针之间的块上被
使能。所有其他块无效。
块 20
R7
11:2
R2
7:2
R5
12:1
R3
11:4
分配指针
块 31
块 32

图 5

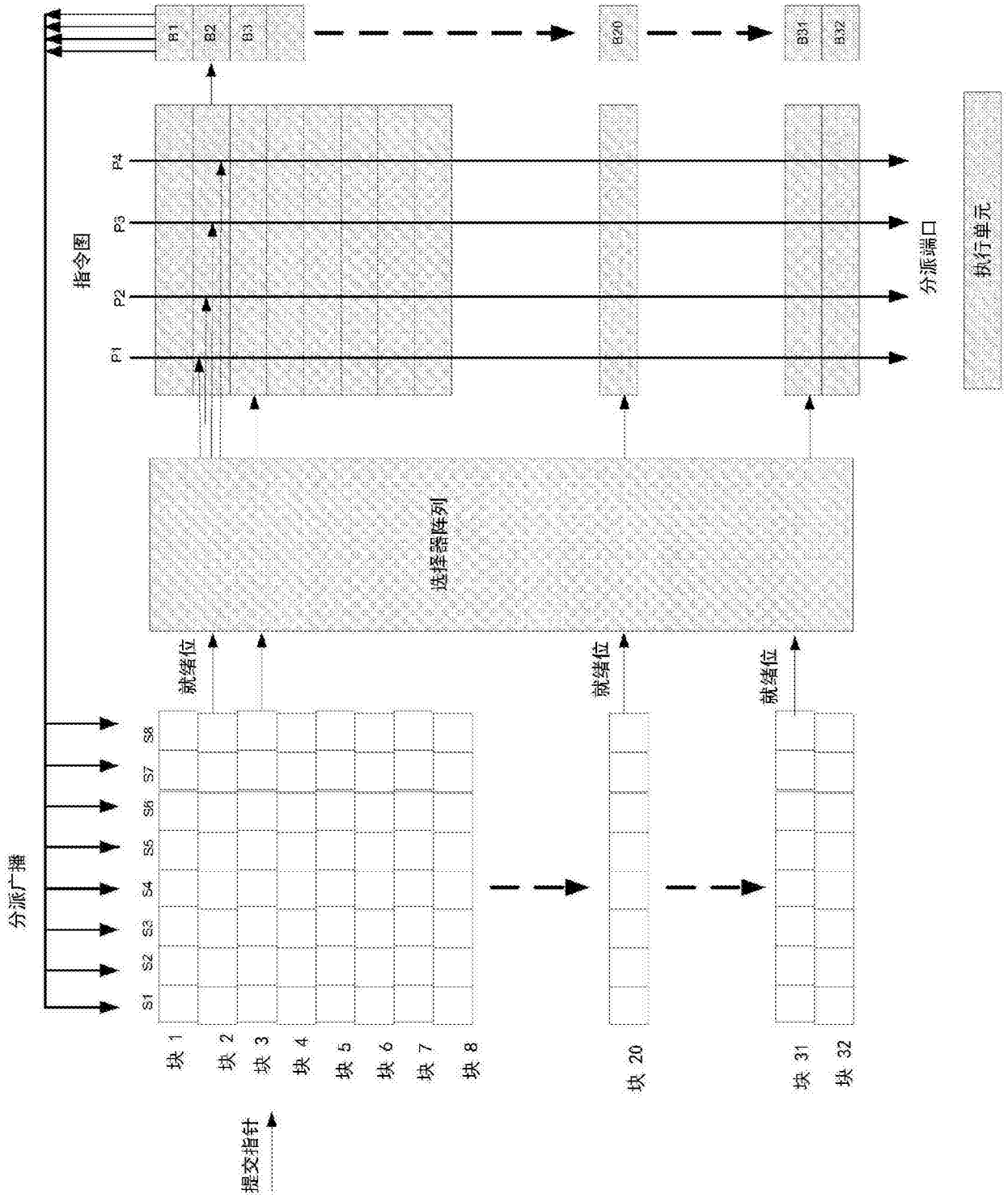

B1
B2
B3
B20
B31
B32
P4
P3
P2
P1
指令图
分派端口
执行单元
选择器阵列
就绪位
就绪位
就绪位
S8
S7
S6
S5
S4
S3
S2
S1
分派广播
块 1
块 2
块 3
块 4
块 5
块 6
块 7
块 8
块 20
块 31
块 32
提交指针

图 6

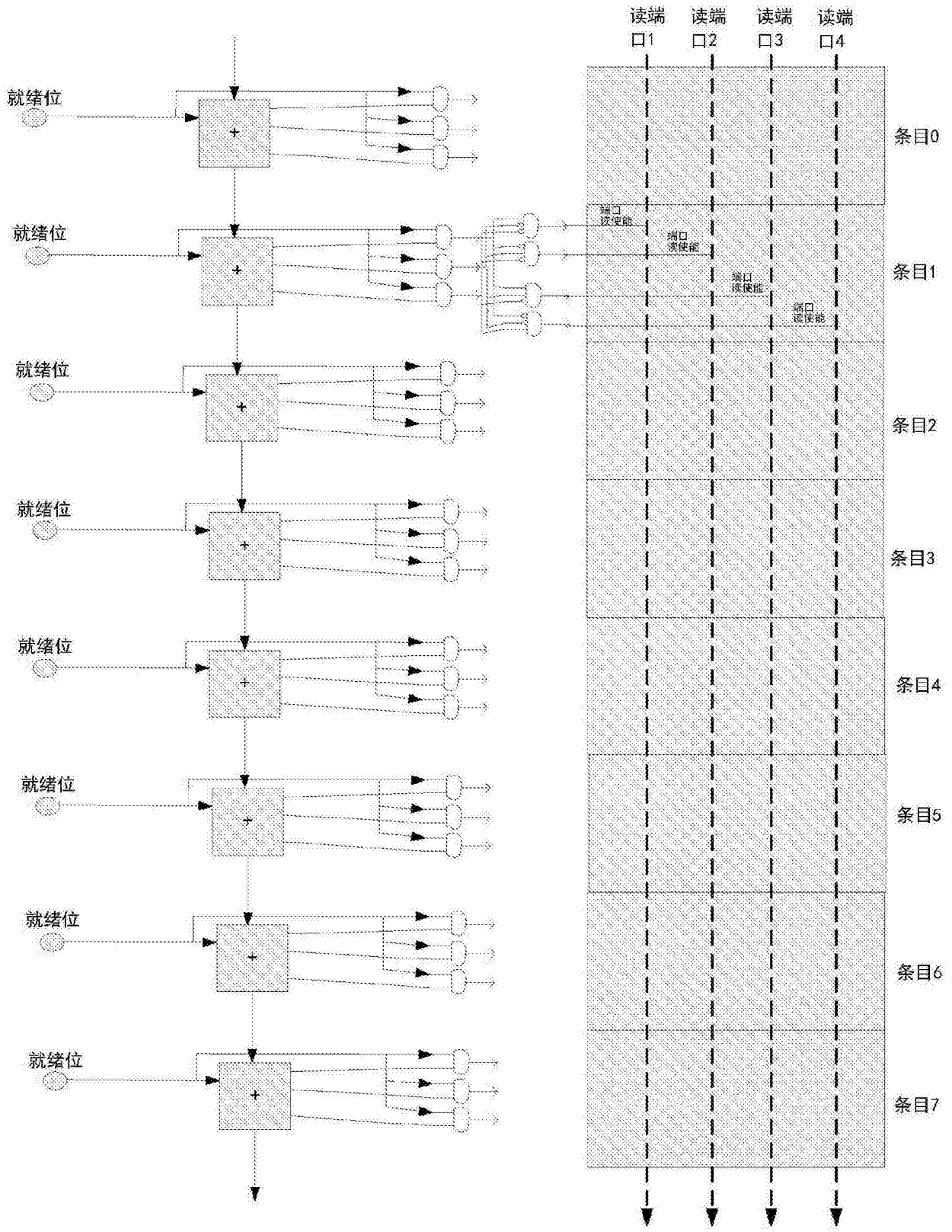

读端口1
读端口2
读端口3
读端口4
就绪位
就绪位
就绪位
就绪位
就绪位
就绪位
就绪位
就绪位
端口读使能
端口读使能
端口读使能
端口读使能
条目0
条目1
条目2
条目3
条目4
条目5
条目6
条目7

图 7

采用范围超过位对每个特定端口的直接选择

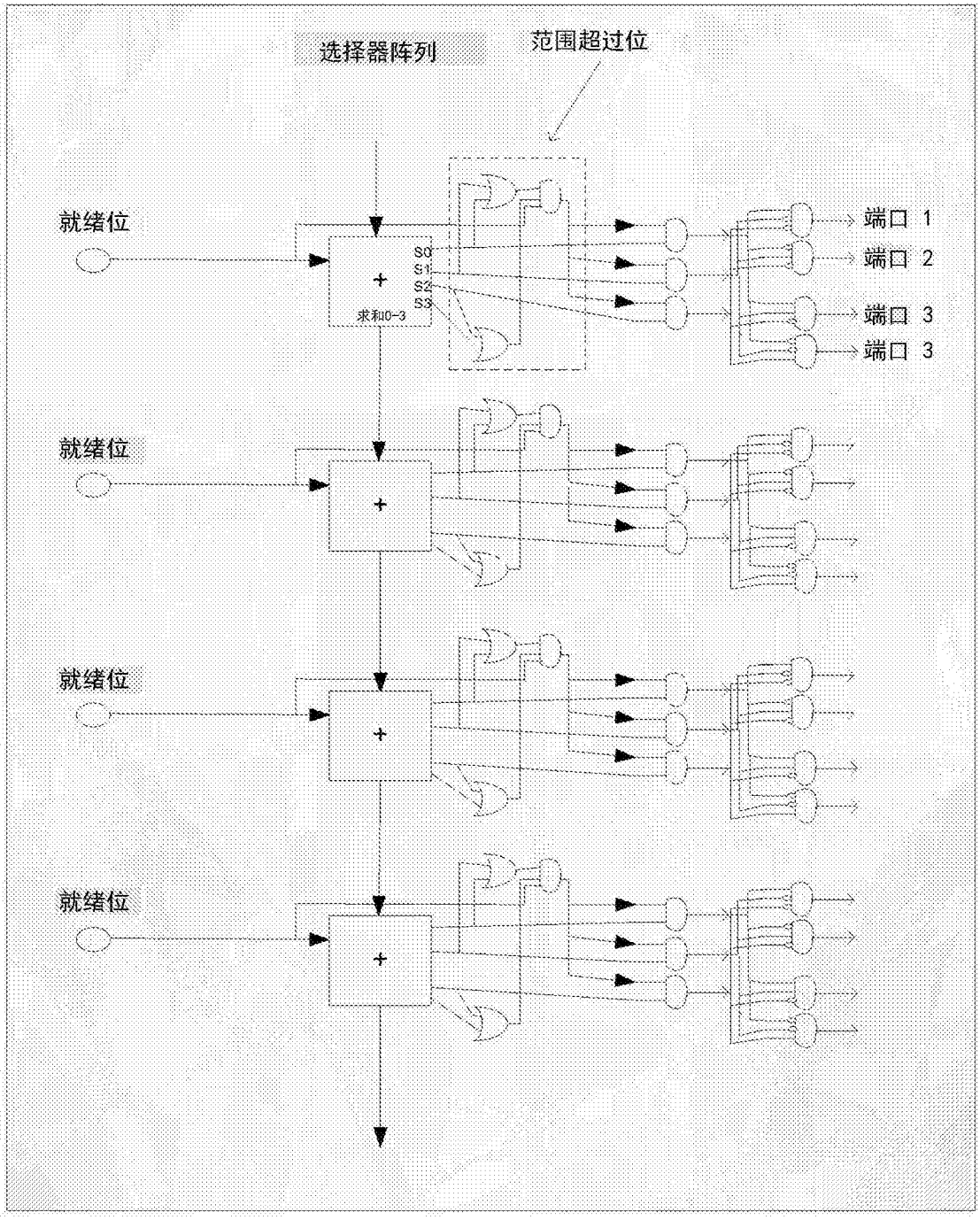


图 8

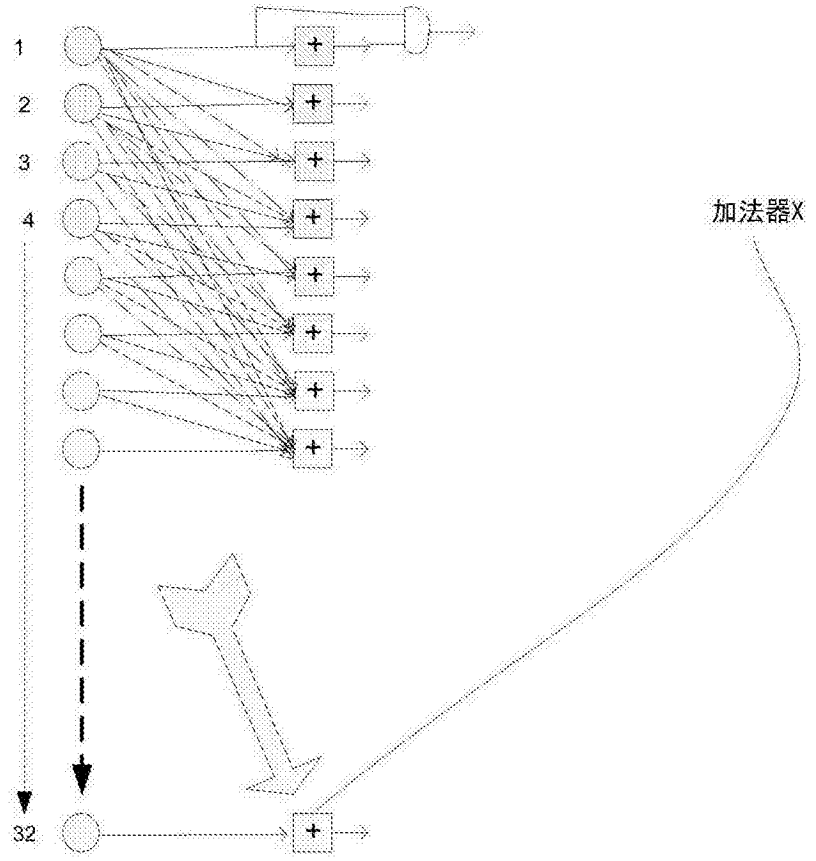


图 9

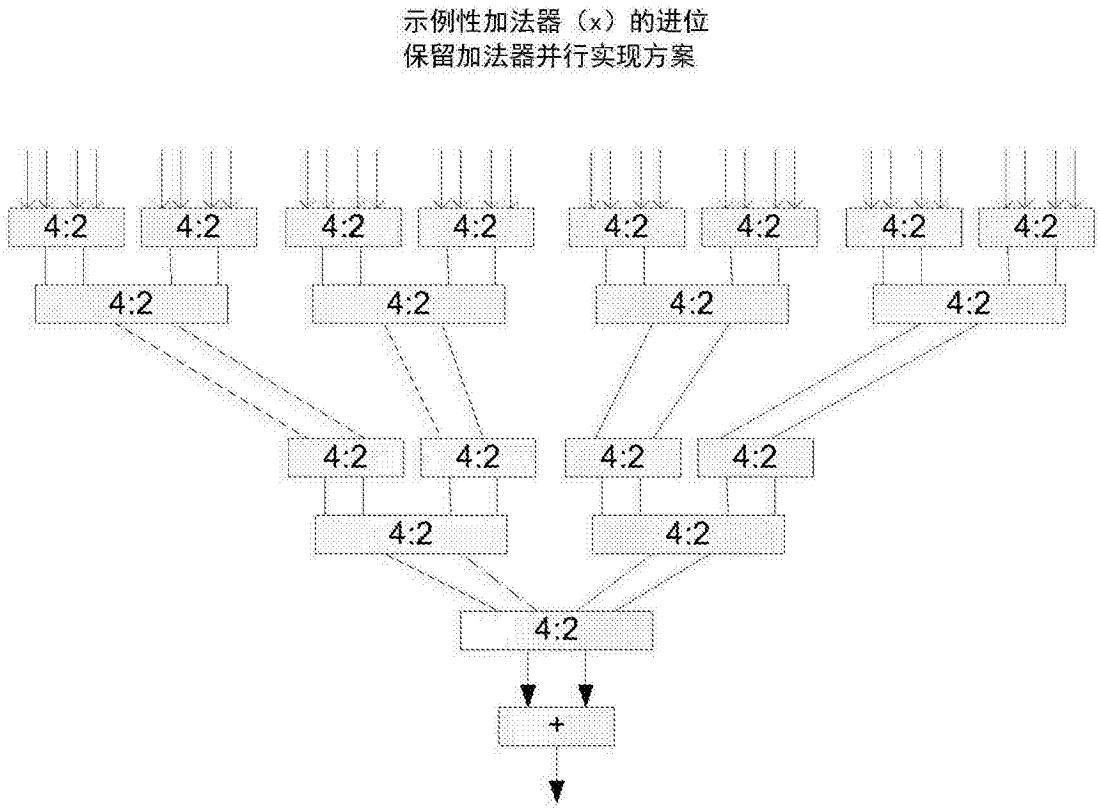


图 10

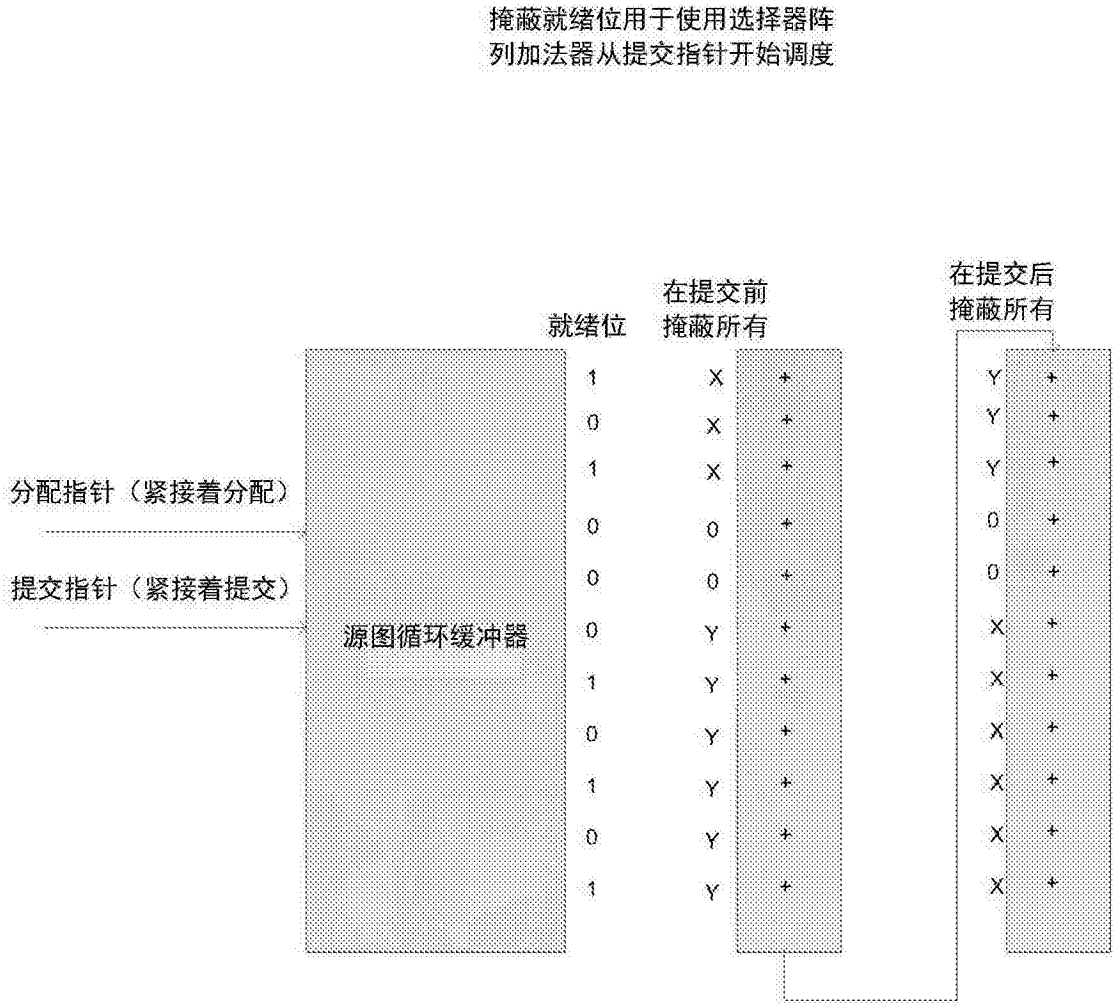

掩蔽就绪位用于使用选择器阵
列加法器从提交指针开始调度
就绪位
在提交前
掩蔽所有
在提交后
掩蔽所有
分配指针（紧接着分配）
提交指针（紧接着提交）
源图循环缓冲器
1 X + Y +
0 X + Y +
1 X + Y +
0 0 + 0 +
0 0 + 0 +
0 Y + X +
1 Y + X +
0 Y + X +
1 Y + X +
0 Y + X +
1 Y + X +

图 11

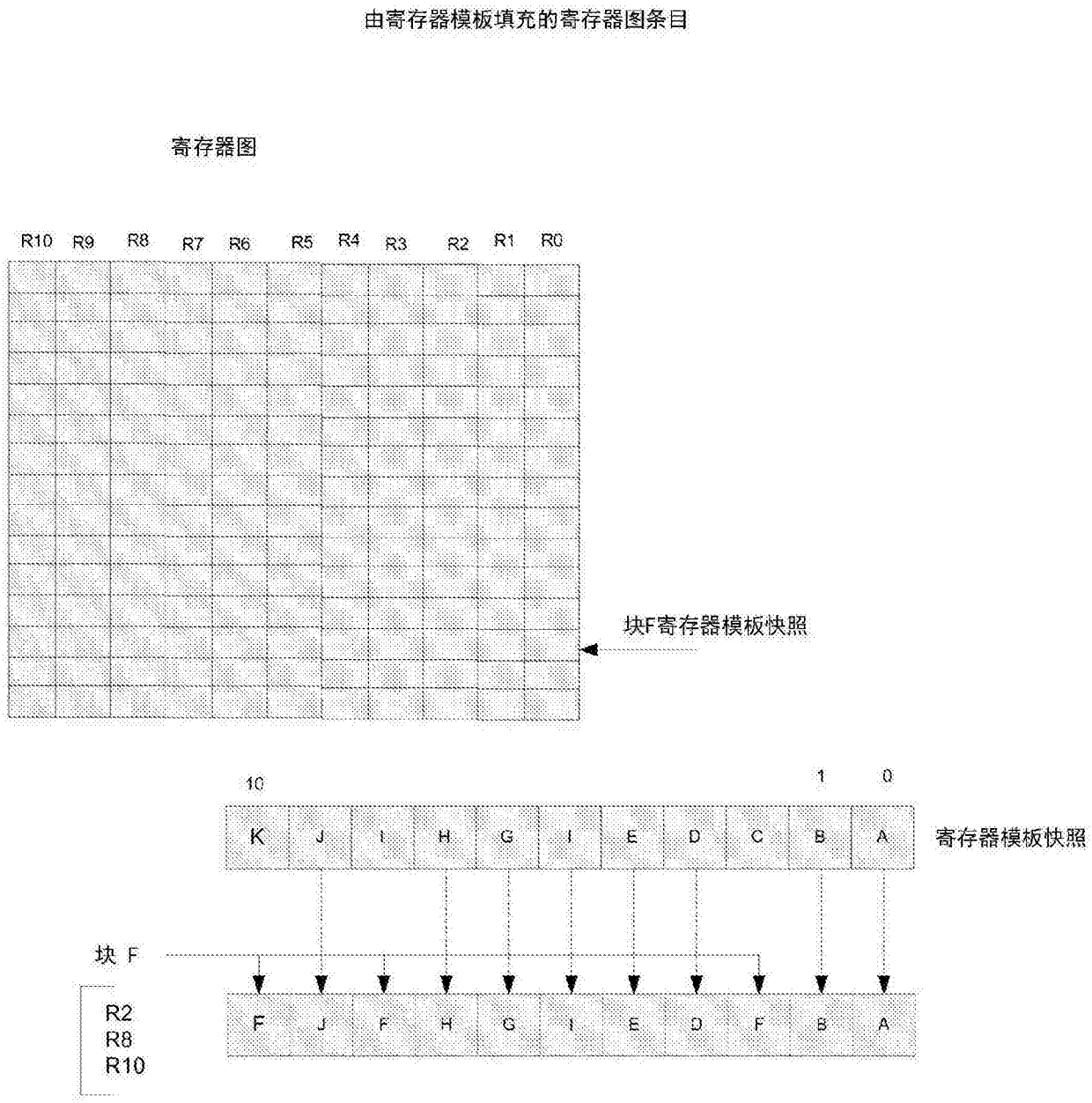

由寄存器模板填充的寄存器图条目
寄存器图
R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0
块F寄存器模板快照
10 1 0
K J I H G I E D C B A
寄存器模板快照
块 F
R2
R8
R10
F J F H G I E D F B A

图 12

减少的寄存器图第一实施例

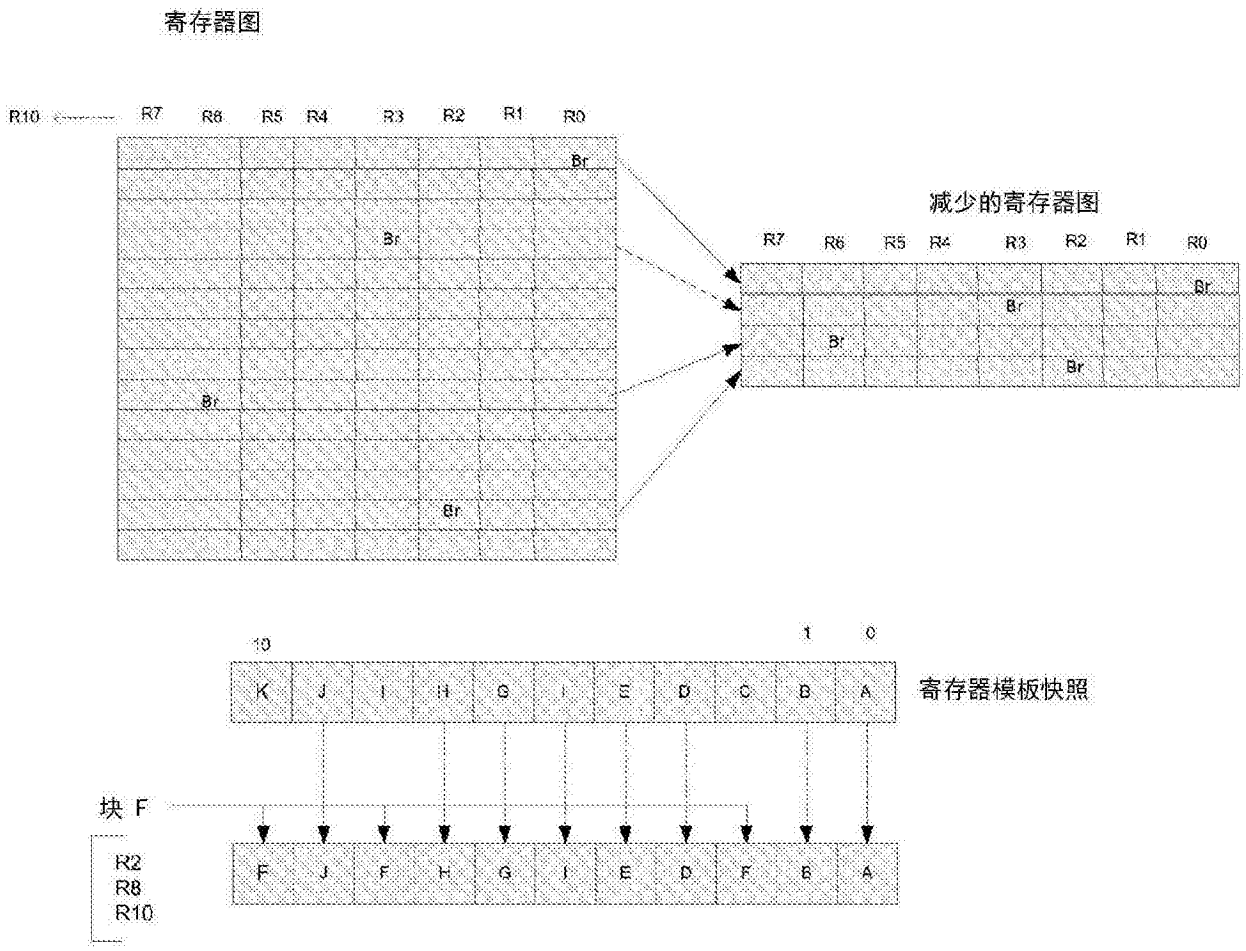


图 13

减少的寄存器图第二实施例

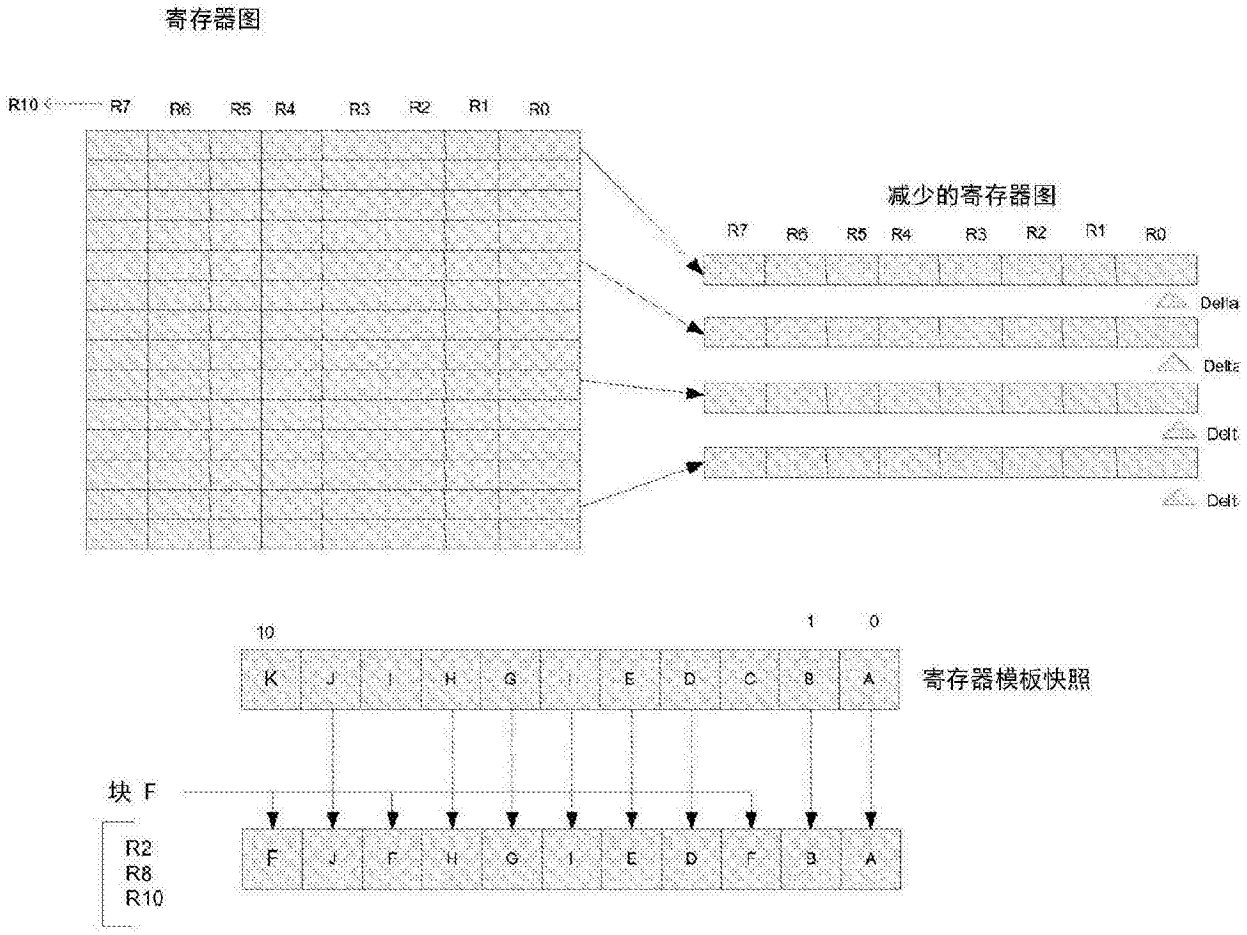


图 14

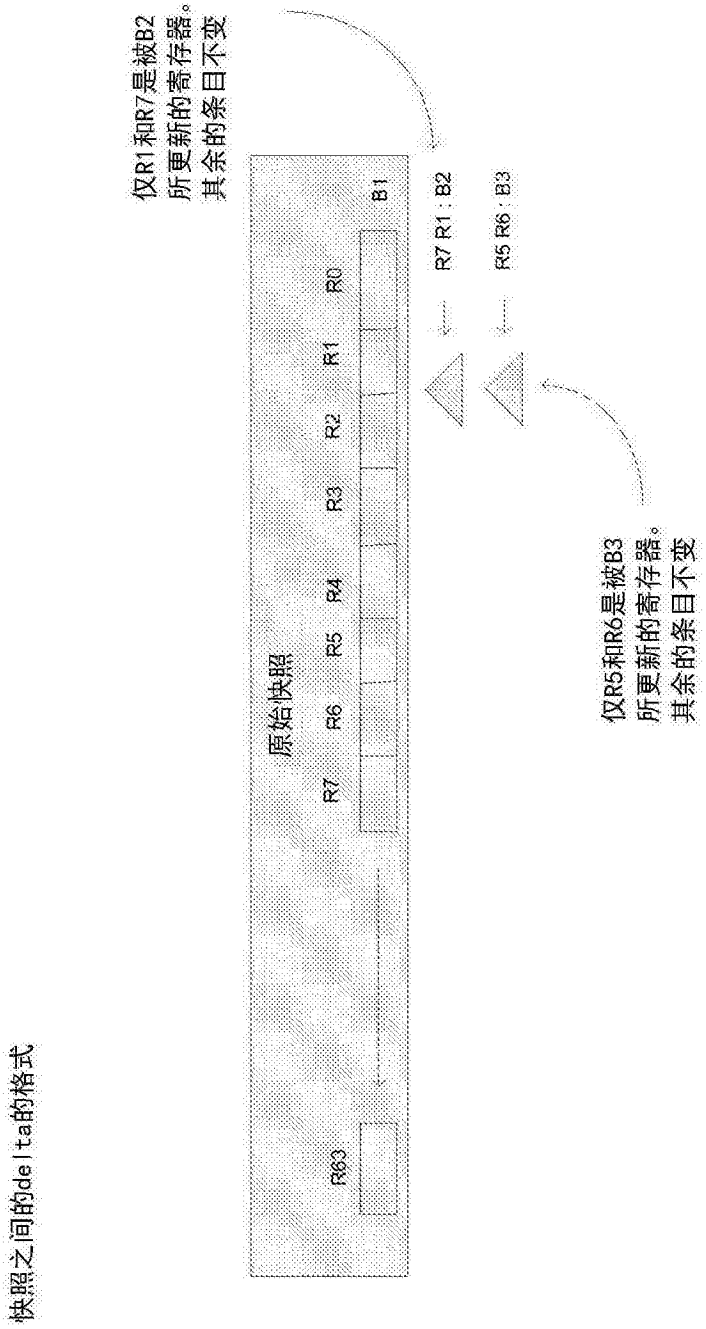

仅R1和R7是被B2
所更新的寄存器。
其余的条目不变
原始快照
R7
R6
R5
R4
R3
R2
R1
R0
B1
R63
R7 R1 : B2
R5 R6 : B3
仅R5和R6是被B3
所更新的寄存器。
其余的条目不变
快照之间的delta的格式

图 15

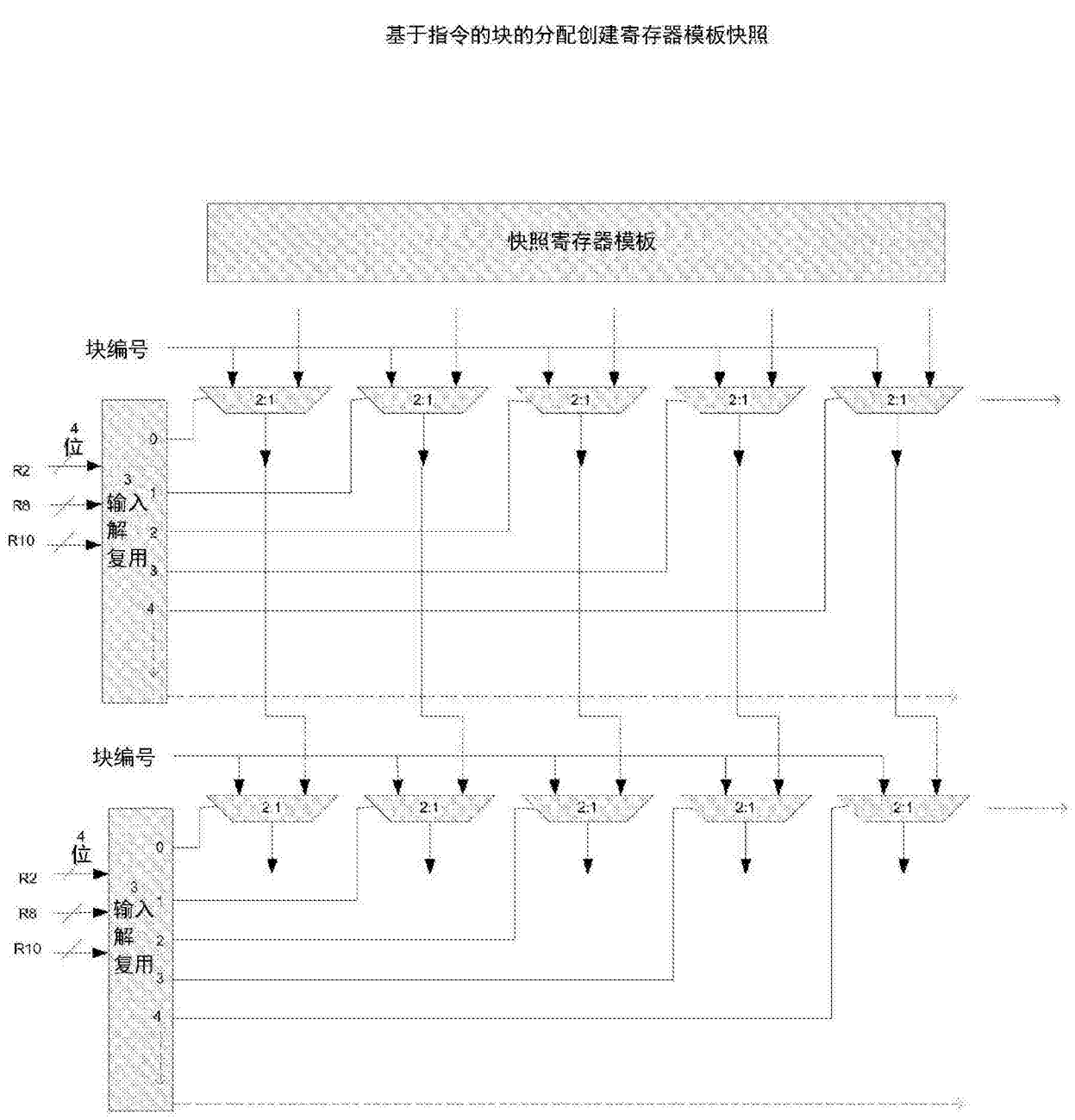

基于指令的块的分配创建寄存器模板快照
快照寄存器模板
块编号
2:1
4
位
R2
R8
R10
3
输入
解
复用
0
1
2
3
4
块编号
2:1
4
位
R2
R8
R10
3
输入
解
复用
0
1
2
3
4

图 16

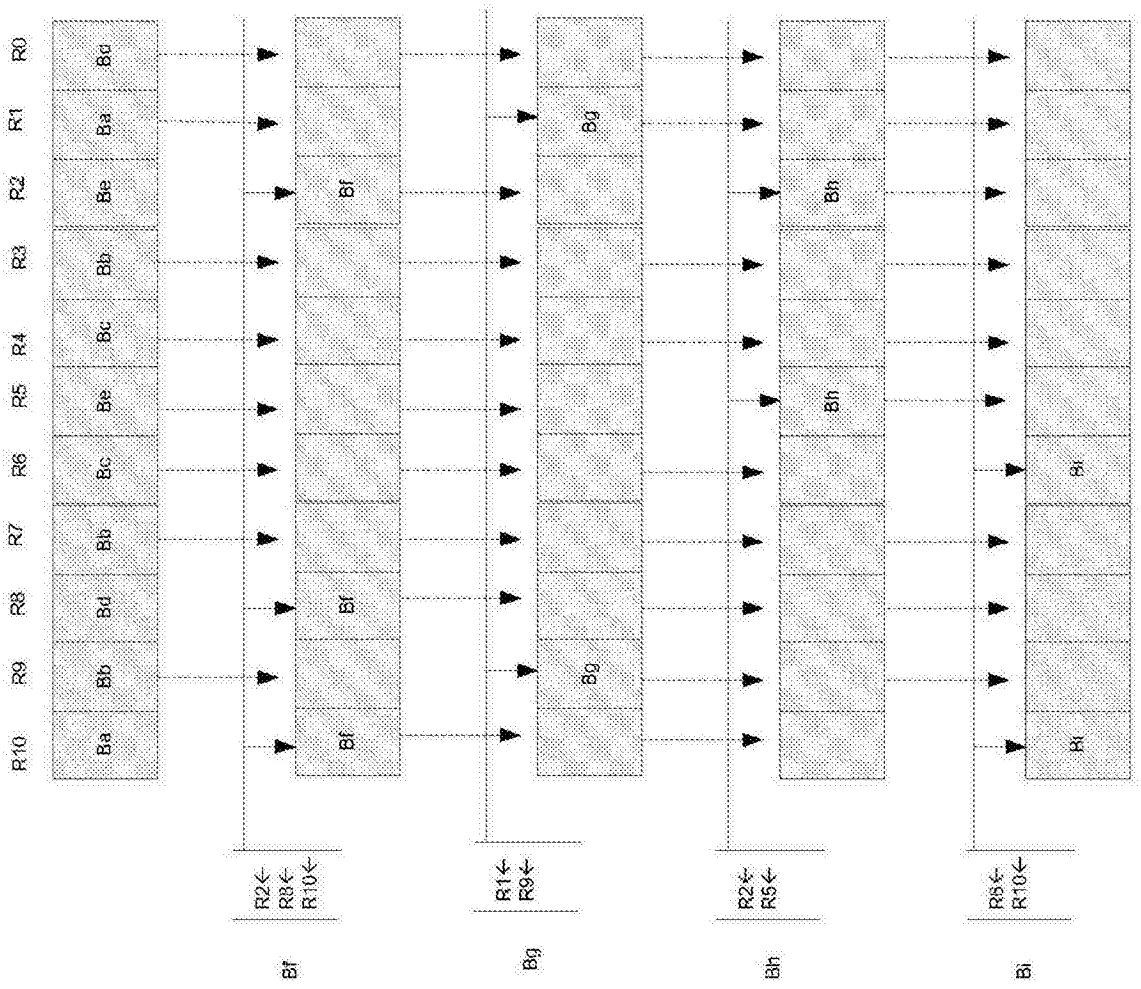

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
Bd
Ba
Be
Bb
Bc
Be
Bc
Bb
Bd
Bb
Ba
Bf
Bf
Bf
Bg
Bg
Bh
Bh
Bi
Bi
R2← R8← R10←
R1← R9←
R2← R5←
R6← R10←
Bf
Bg
Bh
Bi

图 17

串行实现

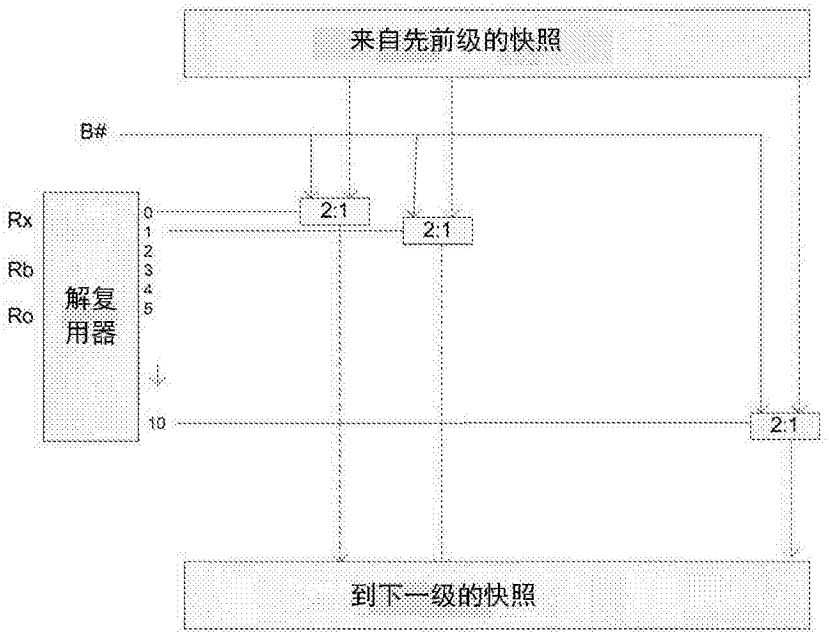


图 18

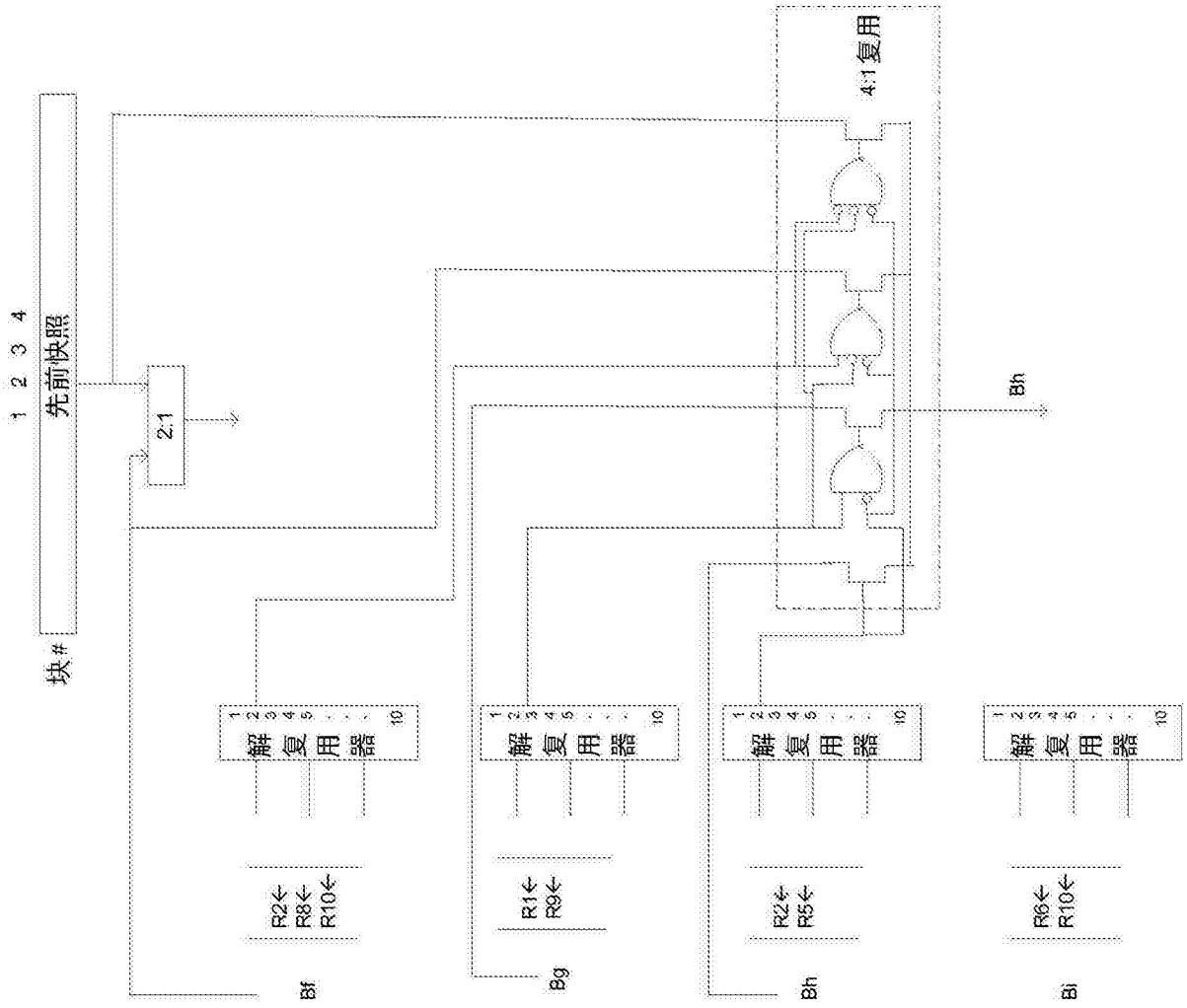


图 19

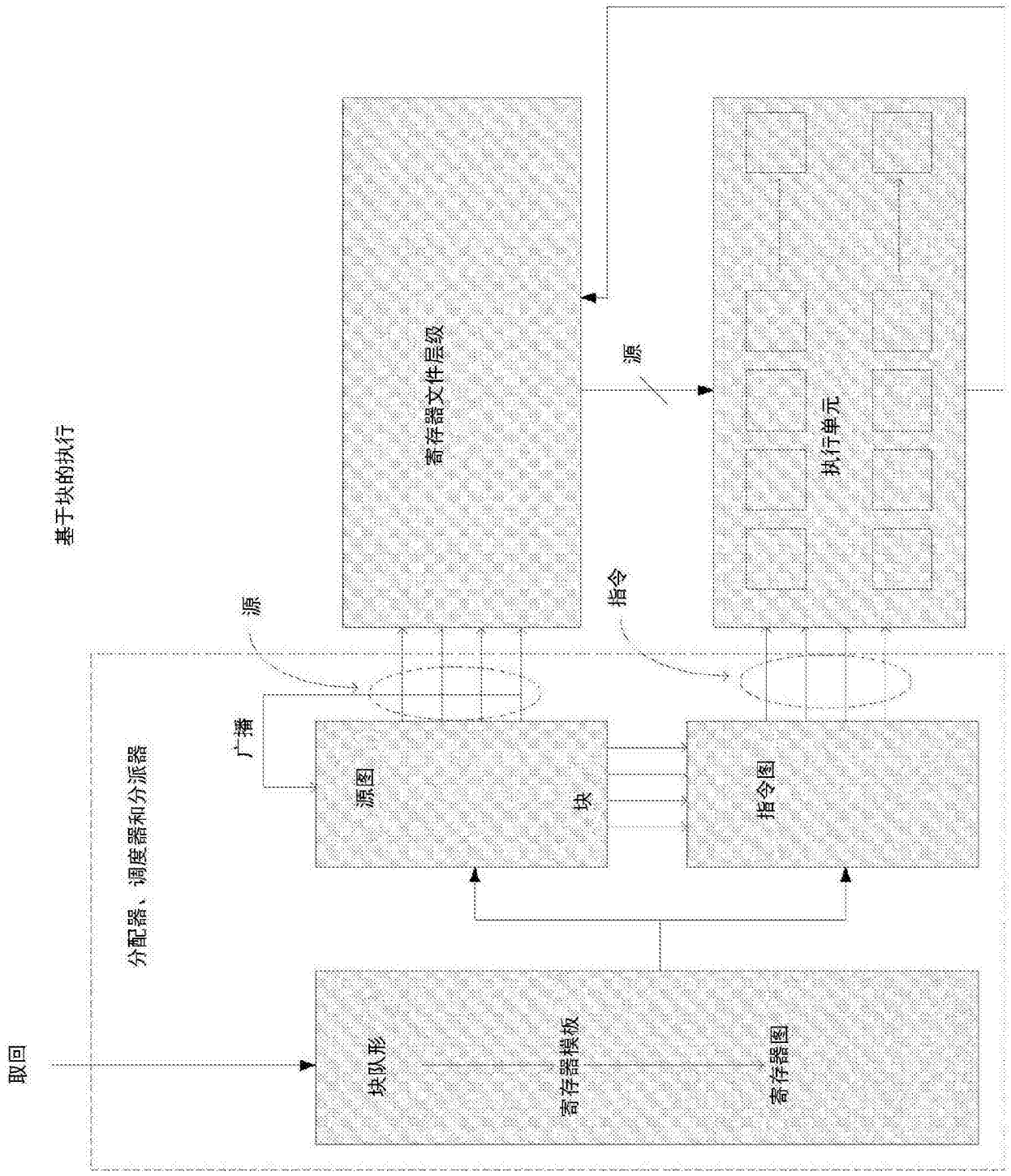

基于块的执行
取回
分配器、调度器和分派器
块队形
寄存器模板
寄存器图
源图
块
广播
源
指令图
指令
寄存器文件层级
源
执行单元

图 20

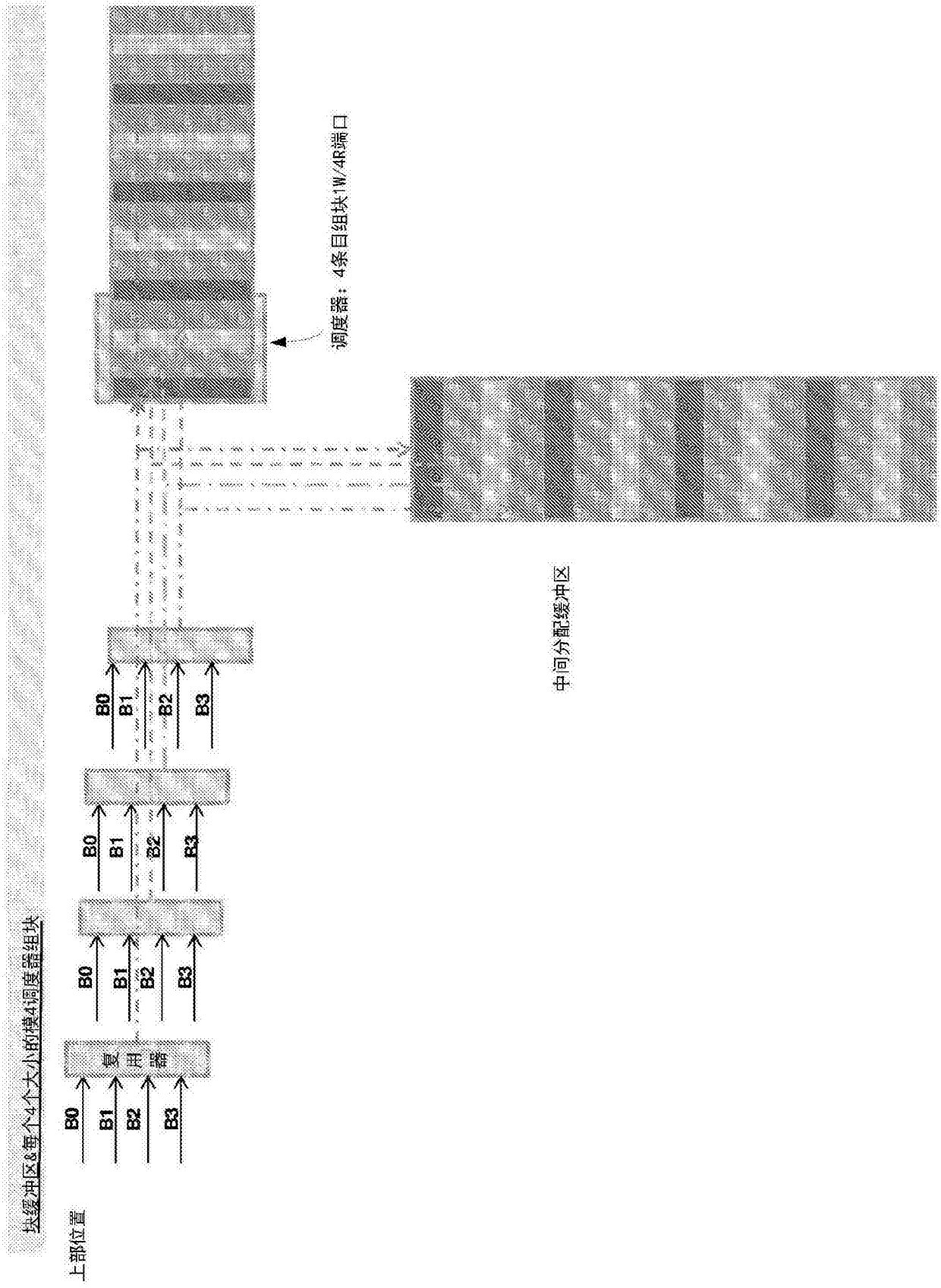

块缓冲区&每个4个大小的模4调度器组块
上部位置
B0
B1
B2
B3
复用器
调度器：4条目组块1W/4R端口
中间分配缓冲区

图 21

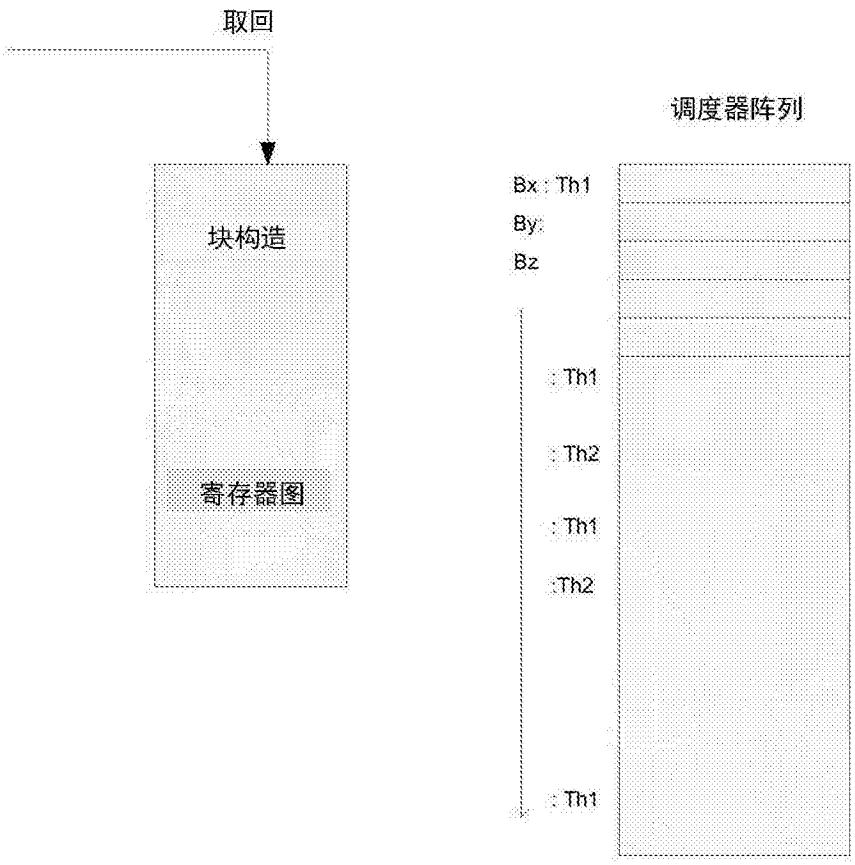

如何根据线程的块编号和线程ID对线程进行分配的描述
取回
块构造
寄存器图
调度器阵列
Bx : Th1
By
Bz
: Th1
: Th2
: Th1
:Th2
: Th1

图 22

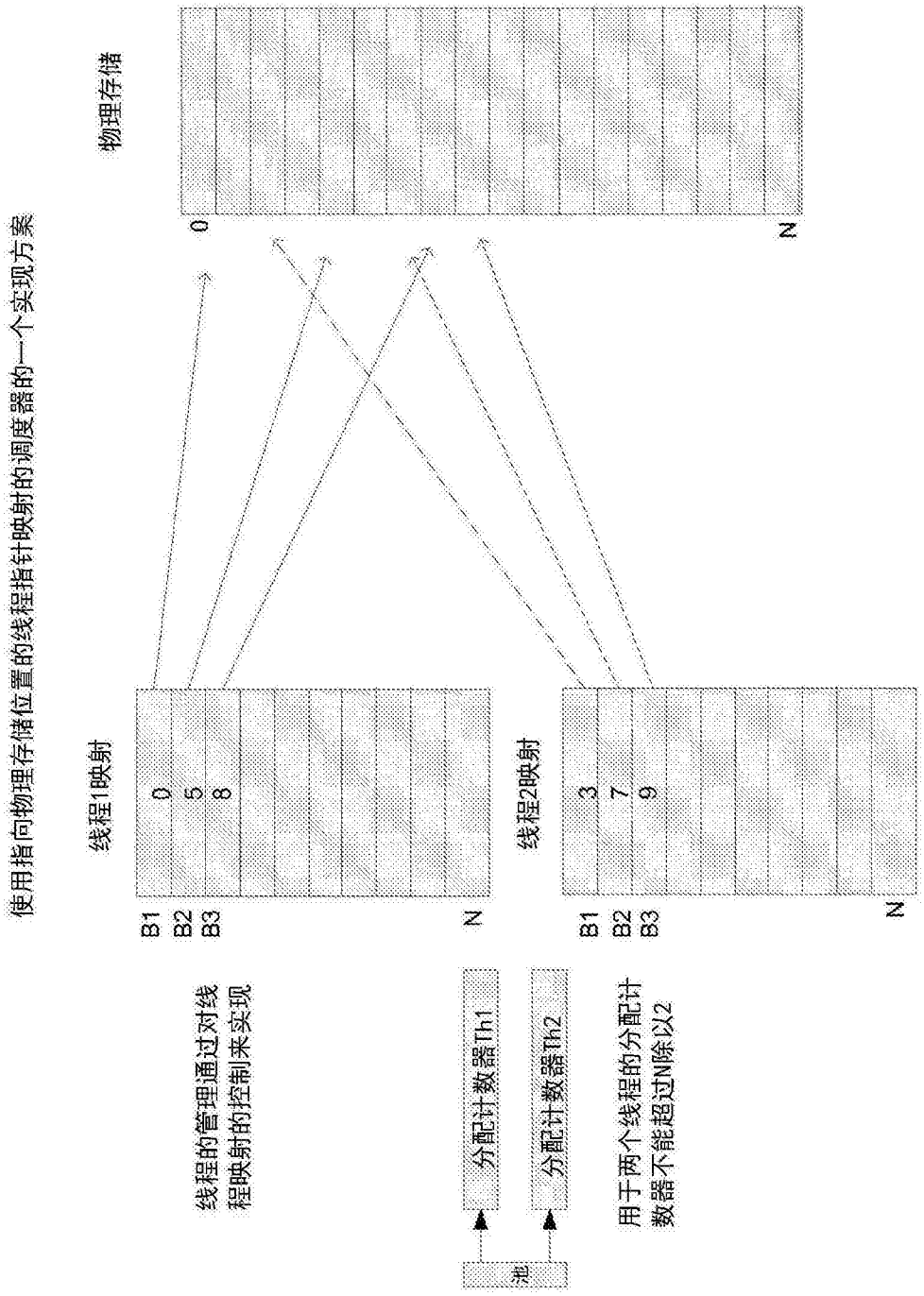

使用指向物理存储位置的线程指针映射的调度器的一个实现方案
物理存储
0
N
线程1映射
0
5
8
B1
B2
B3
N
线程2映射
3
7
9
B1
B2
B3
N
线程的管理通过对线程映射的控制来实现
分配计数器Th1
分配计数器Th2
池
用于两个线程的分配计数器不能超过N除以2

图 23

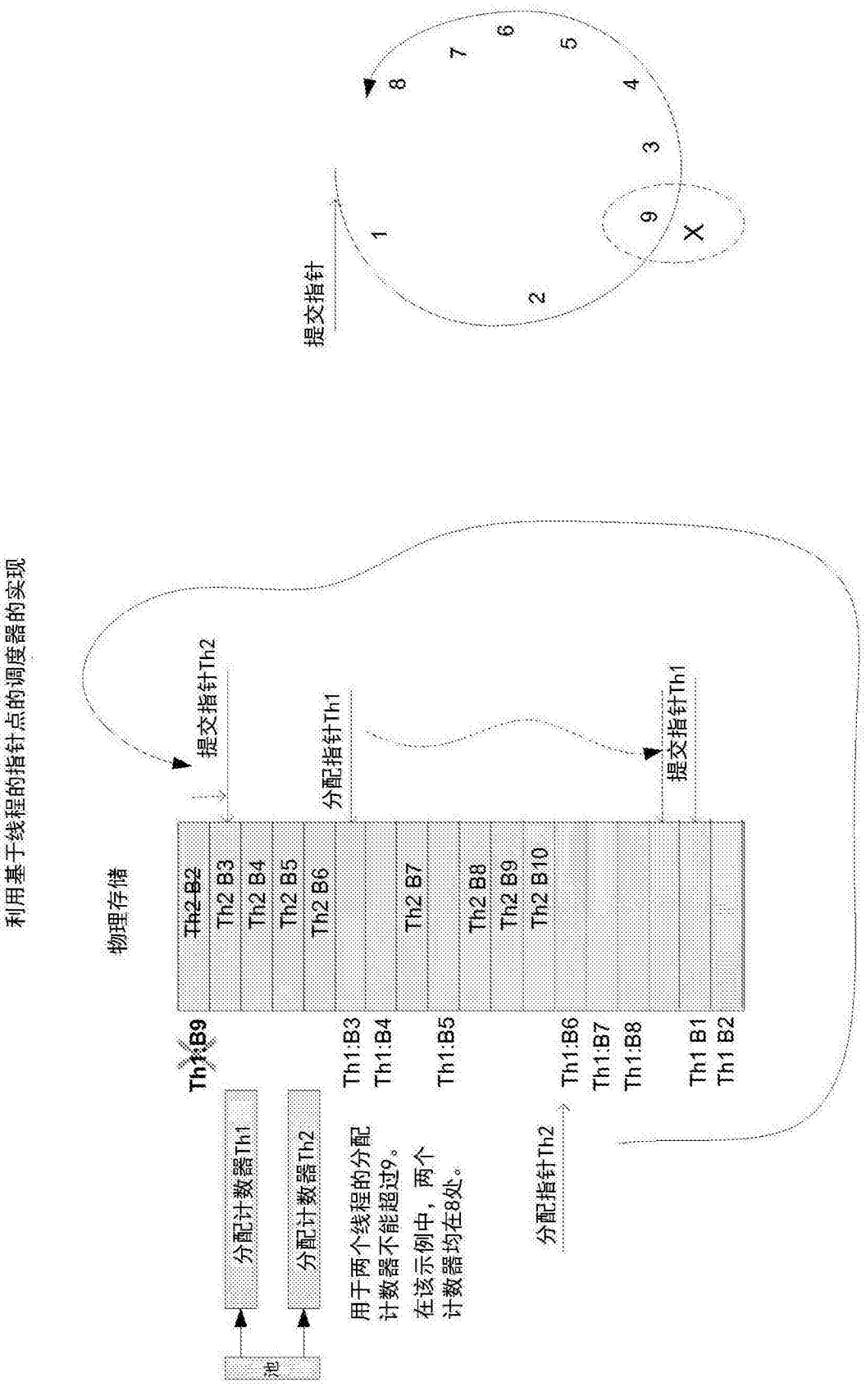

利用基于线程的指针点的调度器的实现
物理存储
池
分配计数器Th1
分配计数器Th2
Th1:B9
Th2 B2
Th2 B3
Th2 B4
Th2 B5
Th2 B6
Th2 B7
Th2 B8
Th2 B9
Th2 B10
Th1:B3
Th1:B4
Th1:B5
Th1:B6
Th1:B7
Th1:B8
Th1 B1
Th1 B2
提交指针Th2
分配指针Th1
提交指针Th1
分配指针Th2
用于两个线程的分配计数器不能超过9。
在该示例中，两个计数器均在8处。
提交指针
1
2
3
4
5
6
7
8
9
X

图 24

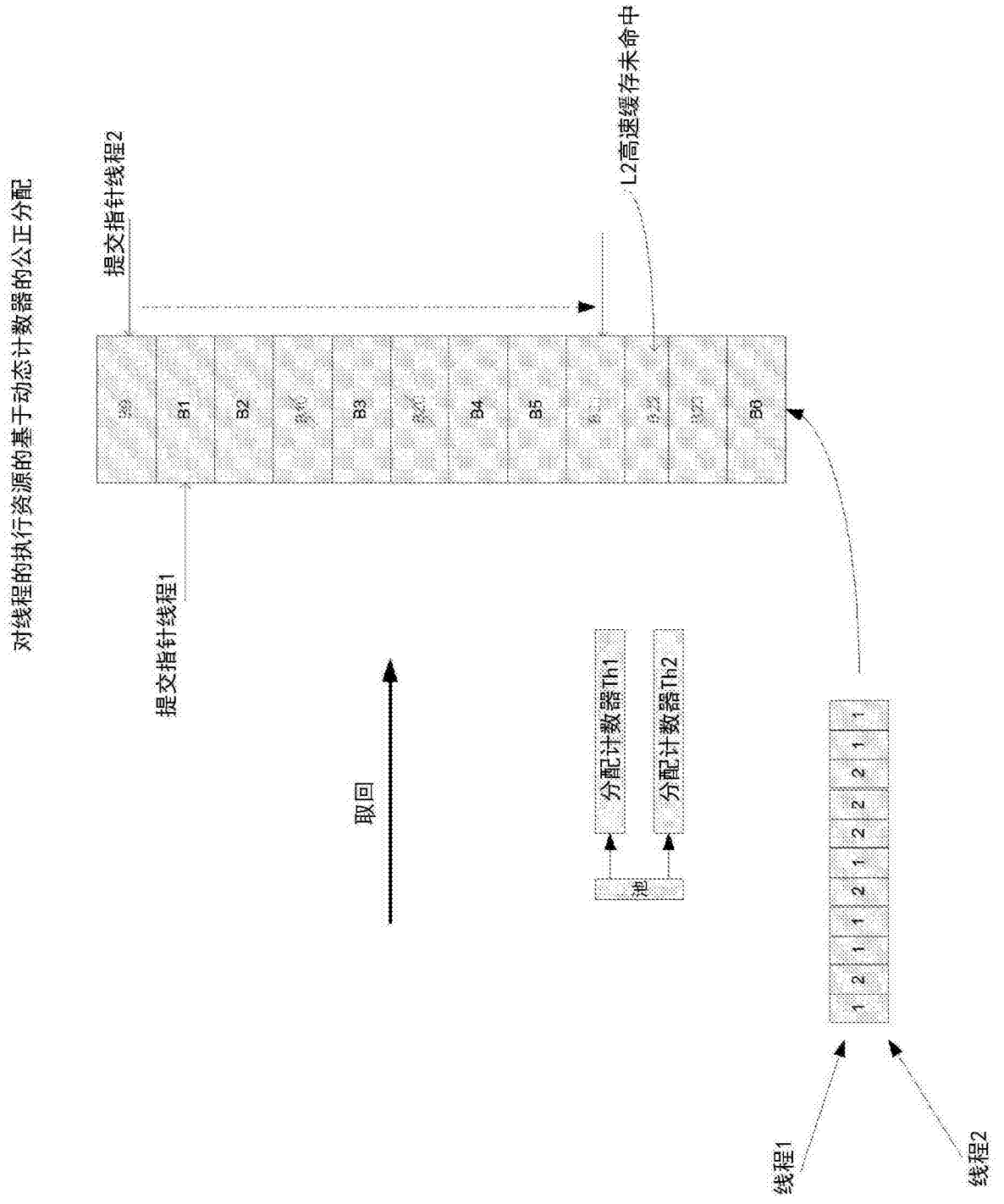

对线程的执行资源的基于动态计数器的公正分配
提交指针线程2
L2高速缓存未命中
B1
B2
B3
B4
B5
B8
提交指针线程1
取回
分配计数器Th1
分配计数器Th2
池
线程1
线程2

图 25

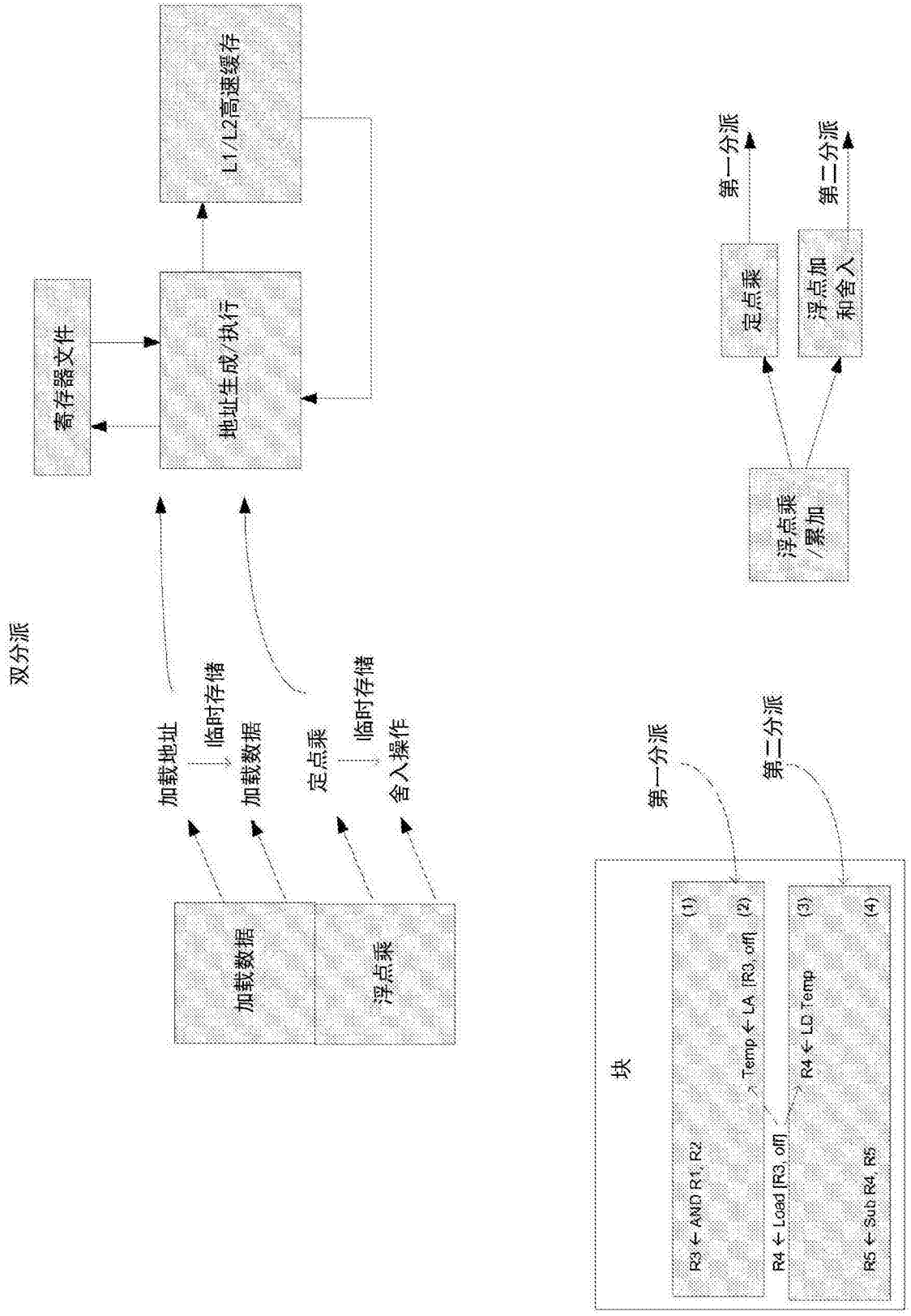

L1/L2高速缓存
寄存器文件
地址生成/执行
双分派
加载地址
临时存储
加载数据
定点乘
临时存储
舍入操作
加载数据
浮点乘
第一分派
定点乘
第二分派
浮点加
和舍入
浮点乘
/累加
第一分派
第二分派
块
(1)
(2)
(3)
(4)
R3 ← AND R1, R2
Temp ← LA, [R3, off]
R4 ← Load [R3, off]
R4 ← LD Temp
R5 ← Sub R4, R5

图 26

双分派暂时乘累加

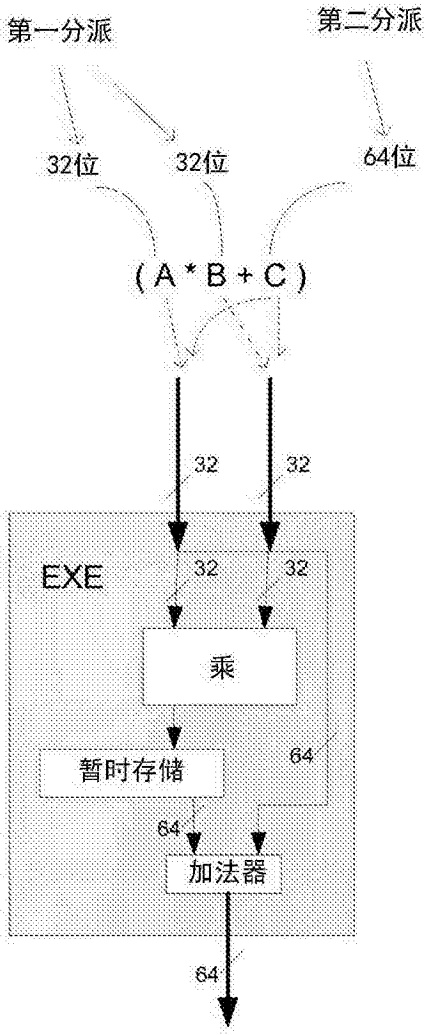


图 27

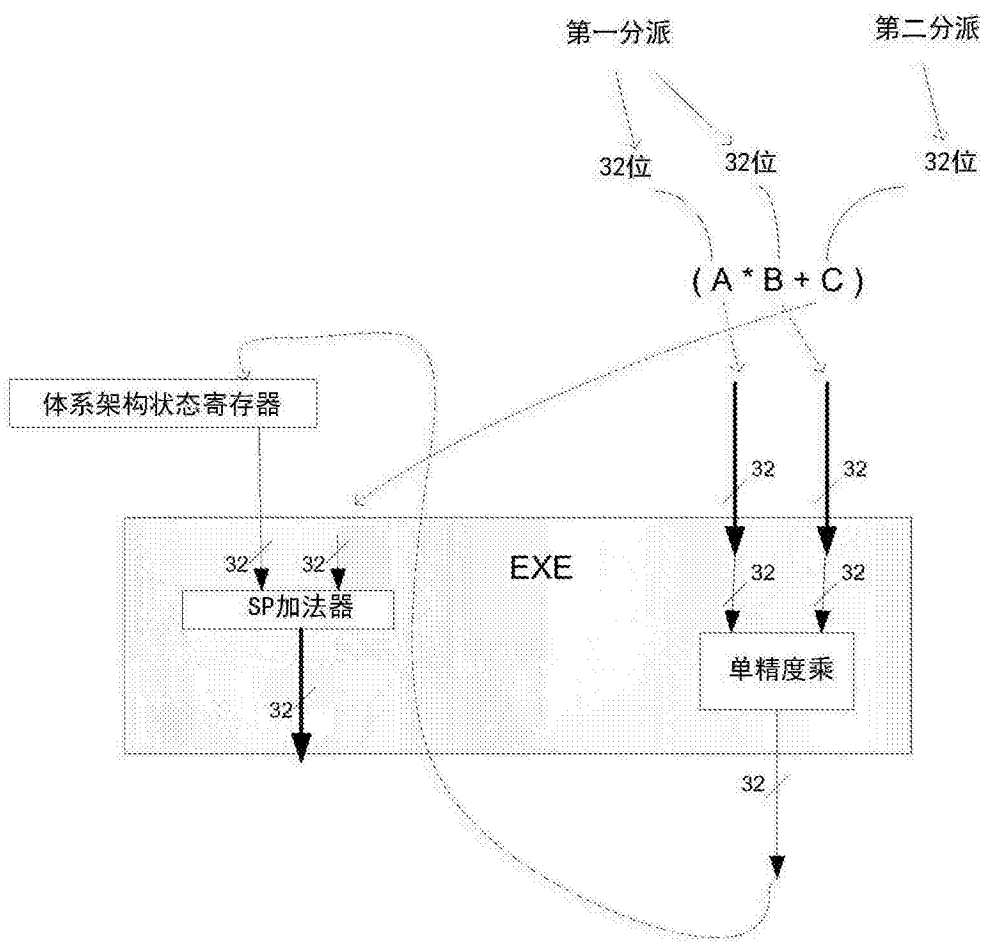

双分派体系架构上可见状态乘加
第一分派
第二分派
32位
32位
32位
(A * B + C)
体系架构状态寄存器
32
32
32
32
EXE
32
32
SP加法器
单精度乘
32
32

图 28

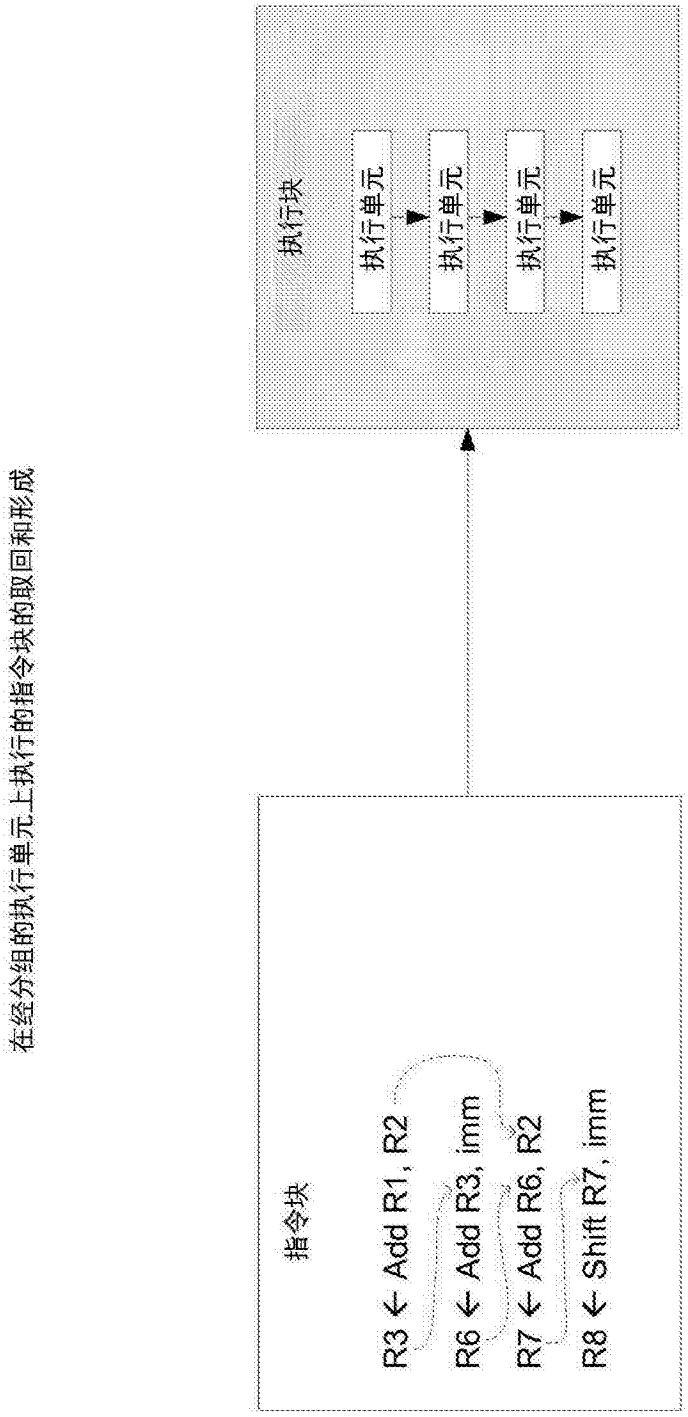

在经分组的执行单元上执行的指令块的取回和形成
执行块
执行单元
执行单元
执行单元
执行单元
指令块
R3 ← Add R1, R2
R6 ← Add R3, imm
R7 ← Add R6, R2
R8 ← Shift R7, imm

图 29

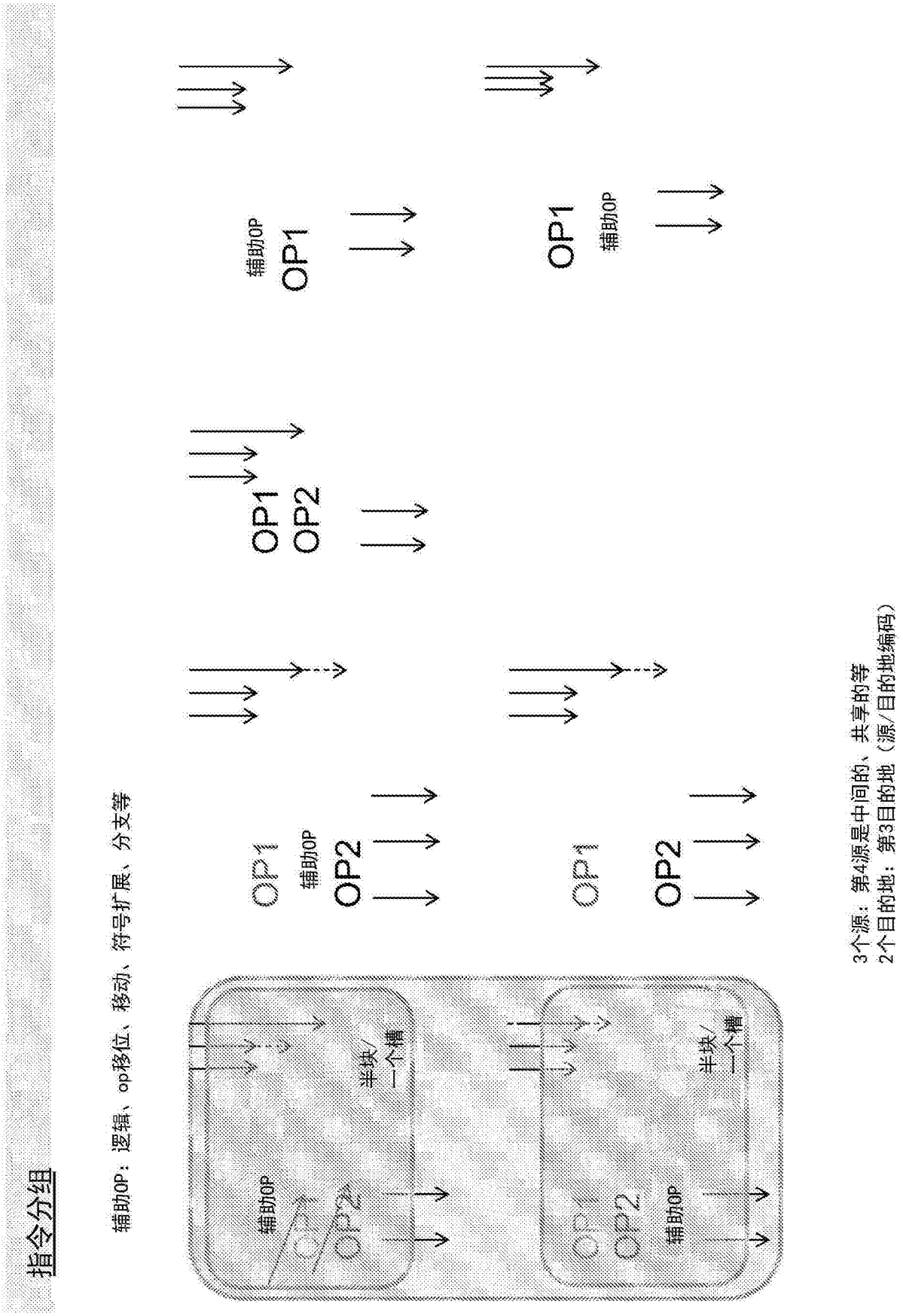

指令分组
辅助OP：逻辑、op移位、移动、符号扩展、分支等
辅助OP
OP1
OP2
半块/一个槽
3个源：第4源是中间的、共享的等
2个目的地：第3目的地（源/目的地编码）

图 30

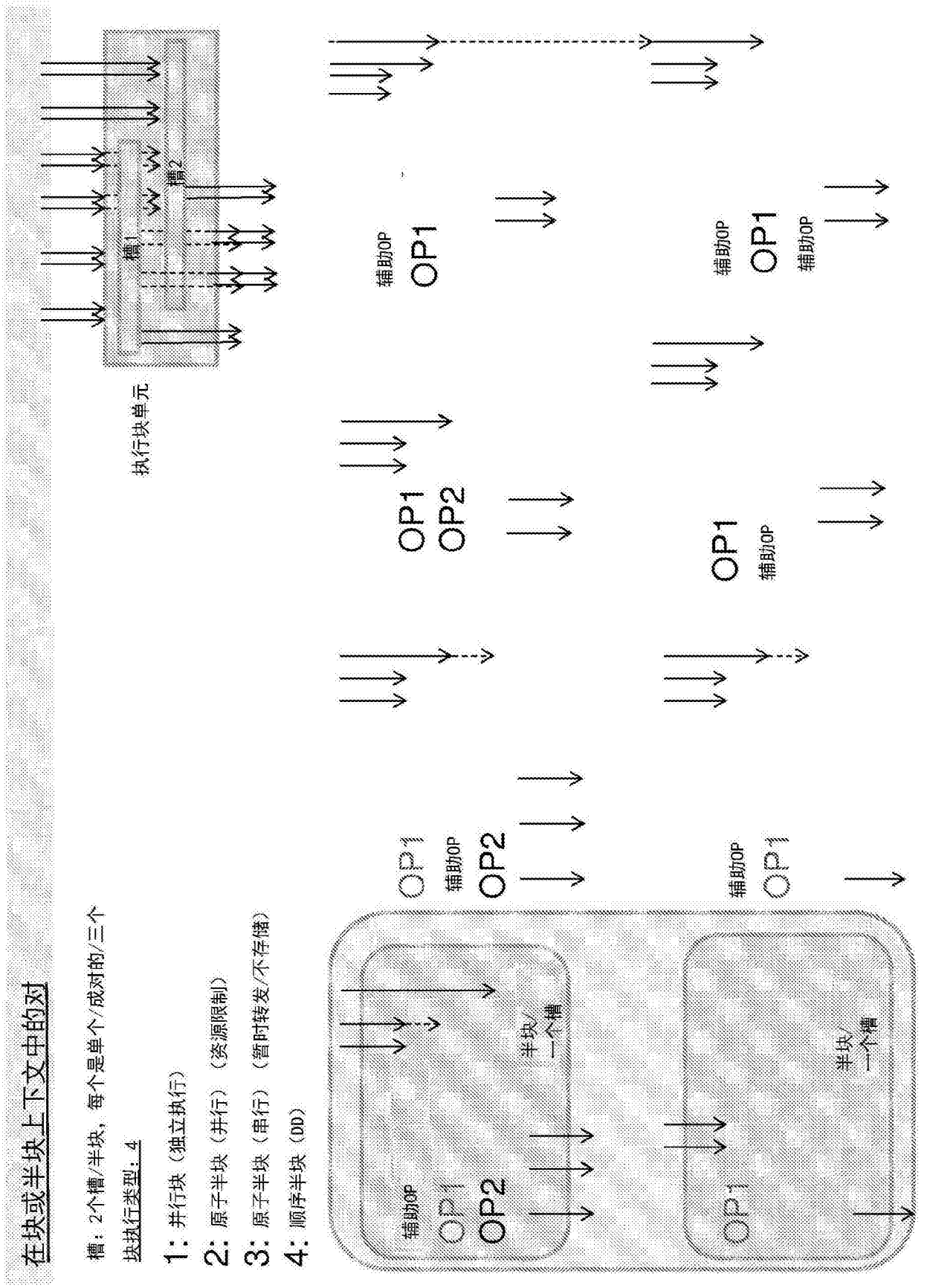

在块或半块上下文中的对
槽：2个槽/半块，每个是单个/成对的/三个
块执行类型：4
1: 并行块（独立执行）
2: 原子半块（并行）（资源限制）
3: 原子半块（串行）（暂时转发/不存储）
4: 顺序半块（DD）
执行块单元
槽1
槽2
辅助OP
OP1
OP2
半块/一个槽

图 31

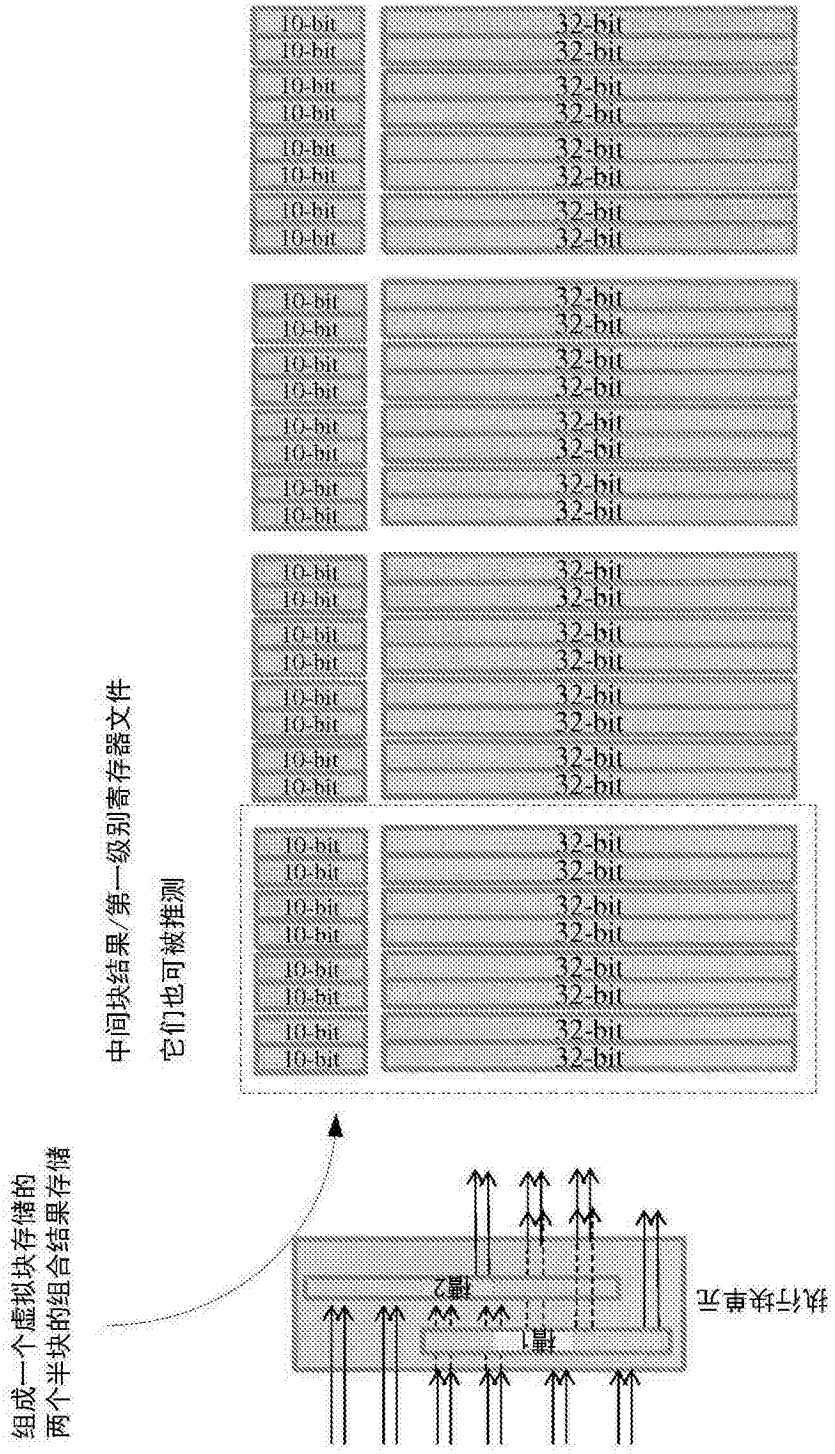

10-bit
32-bit
中间块结果/第一级别寄存器文件
它们也可被推测
组成一个虚拟块存储的
两个半块的组合结果存储
执行块单元
槽1
槽2

图 32

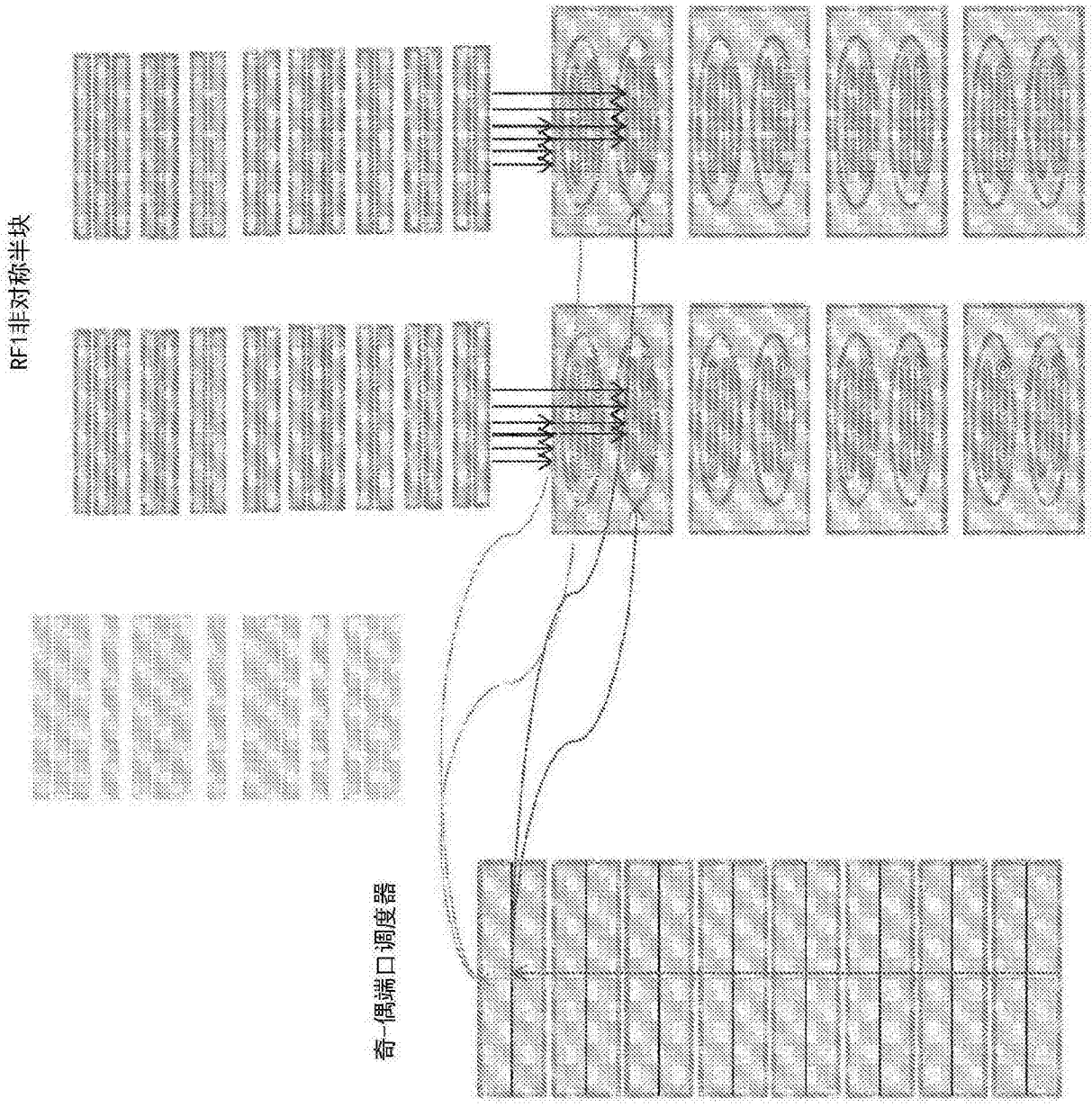


图 33

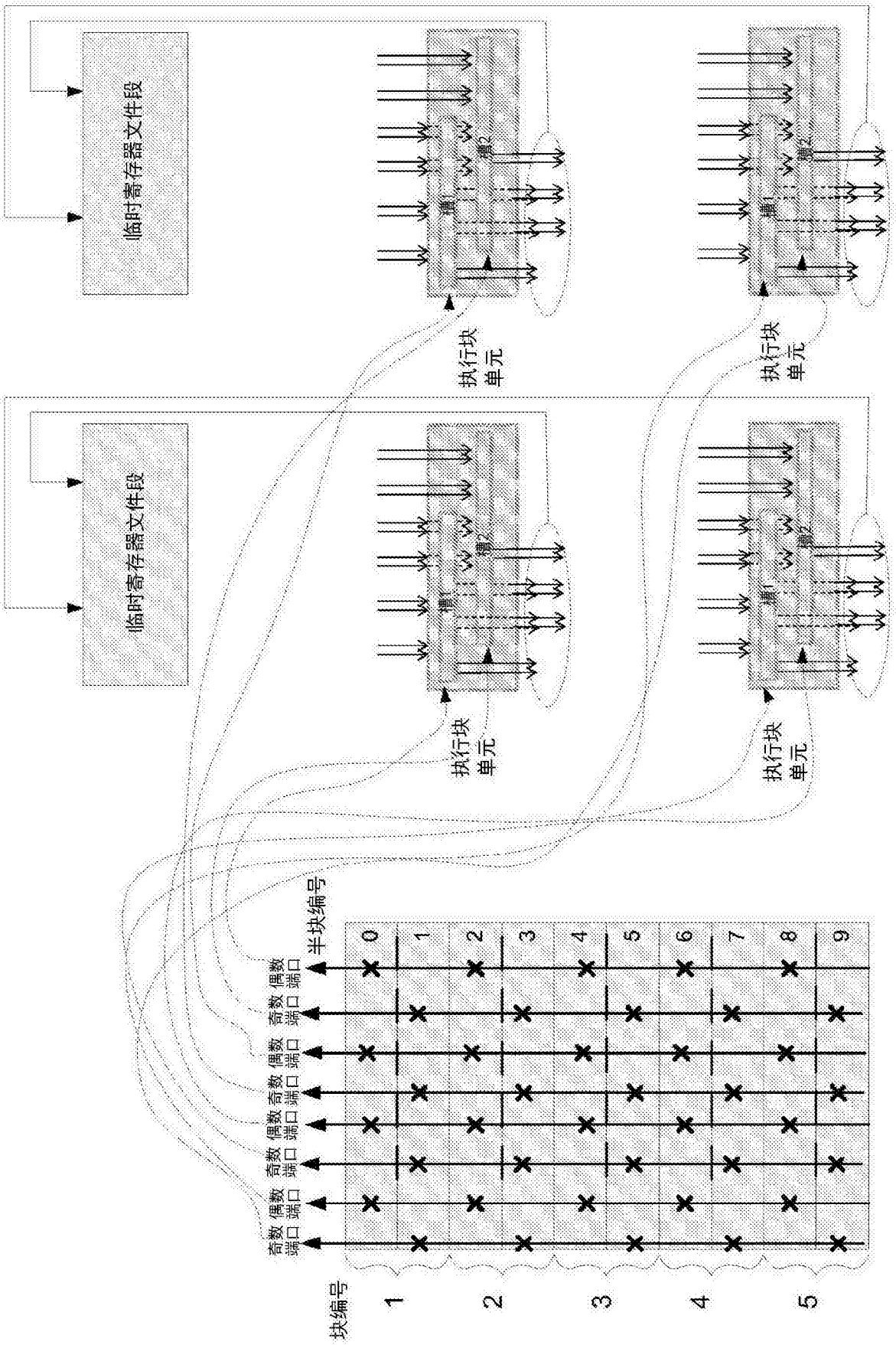


图 34

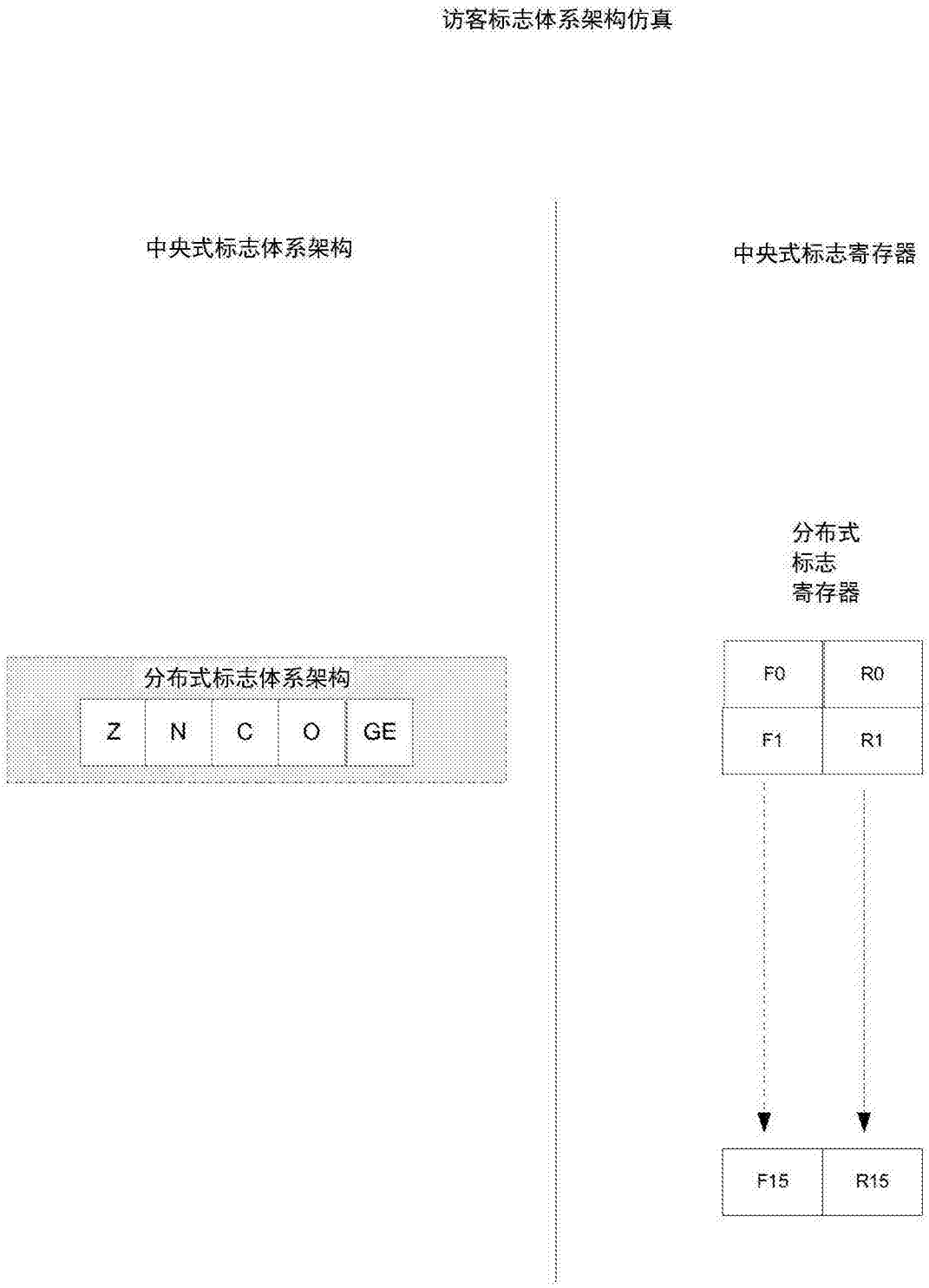

访客标志体系架构仿真
中央式标志体系架构
分布式标志体系架构
Z
N
C
O
GE
中央式标志寄存器
分布式
标志
寄存器
F0
R0
F1
R1
F15
R15

图 35

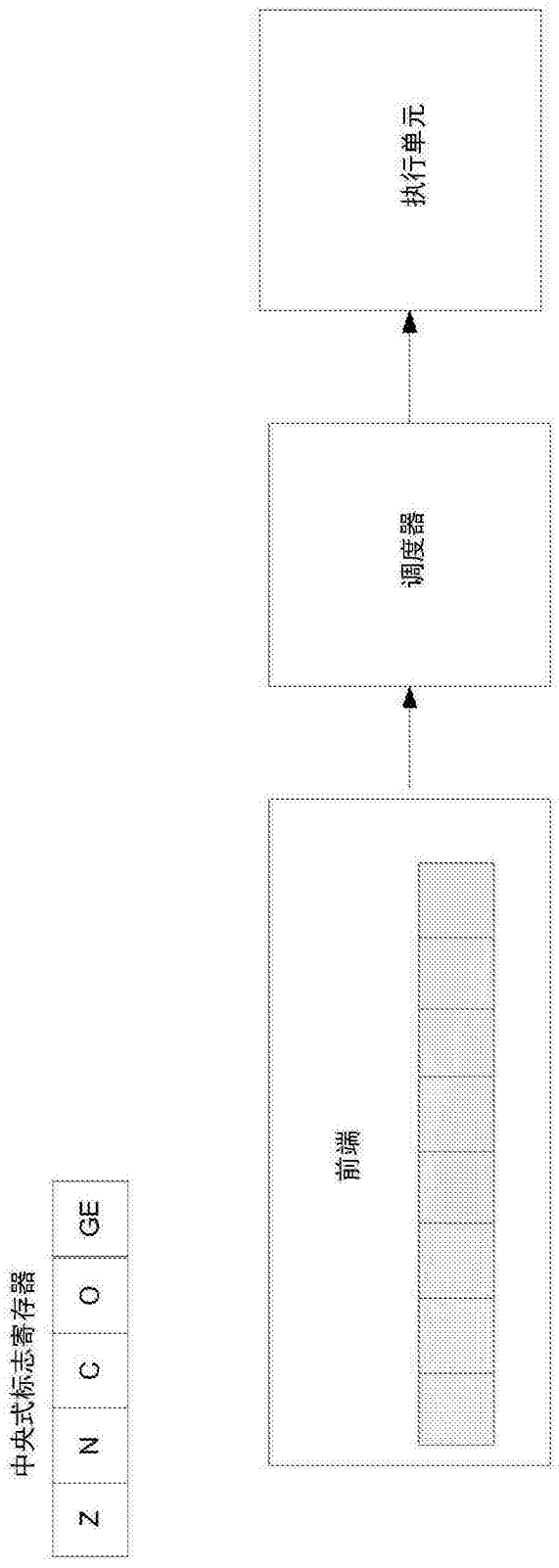


图 36

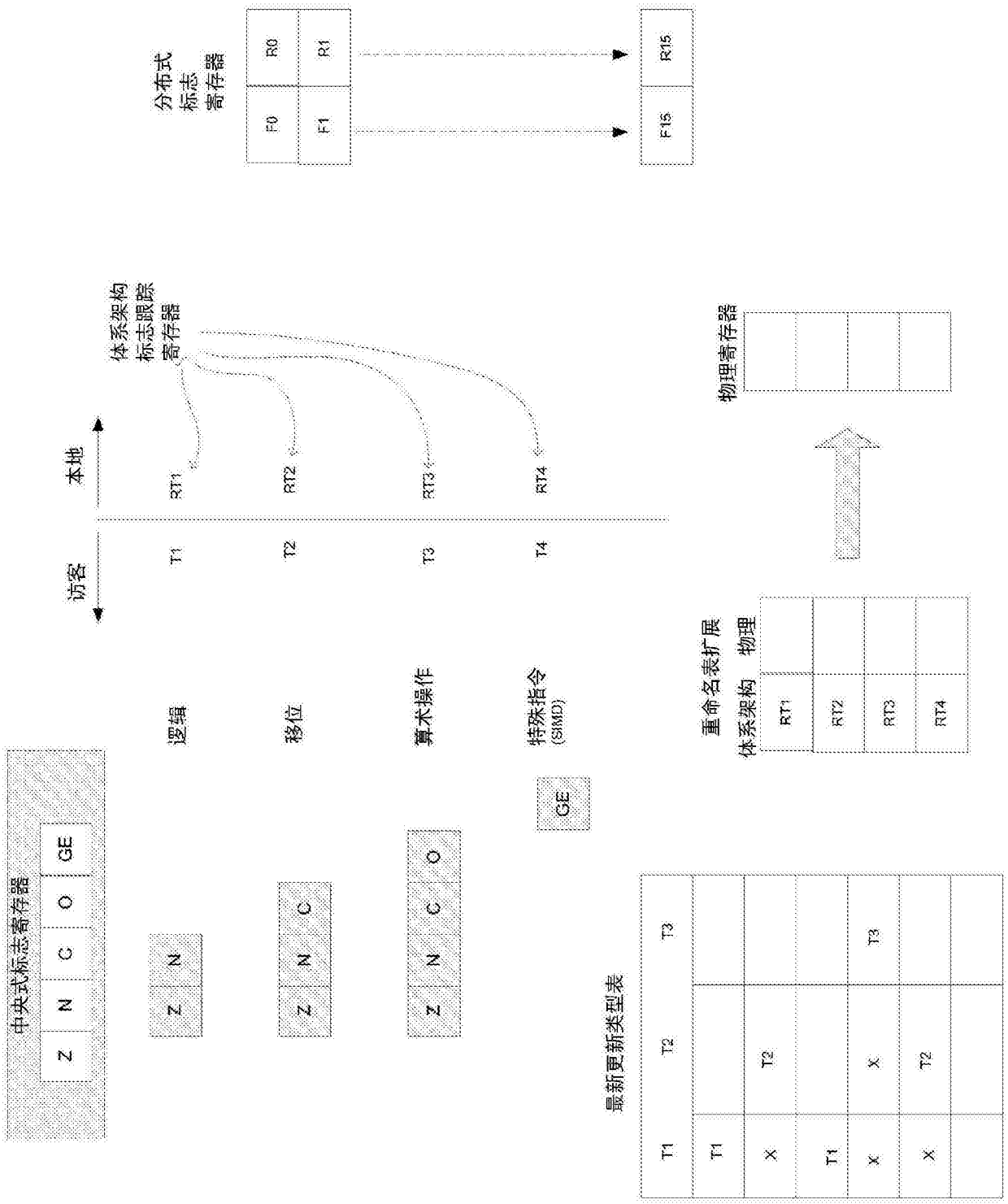

分布式
标志
寄存器
R0
R1
R15
F0
F1
F15
体系架构
标志跟踪
寄存器
本地
访客
RT1
RT2
RT3
RT4
T1
T2
T3
T4
逻辑
移位
算术操作
特殊指令
(SIMD)
GE
Z
N
C
O
物理寄存器
重命名表扩展
体系架构
物理
最新更新类型表
X

图 37

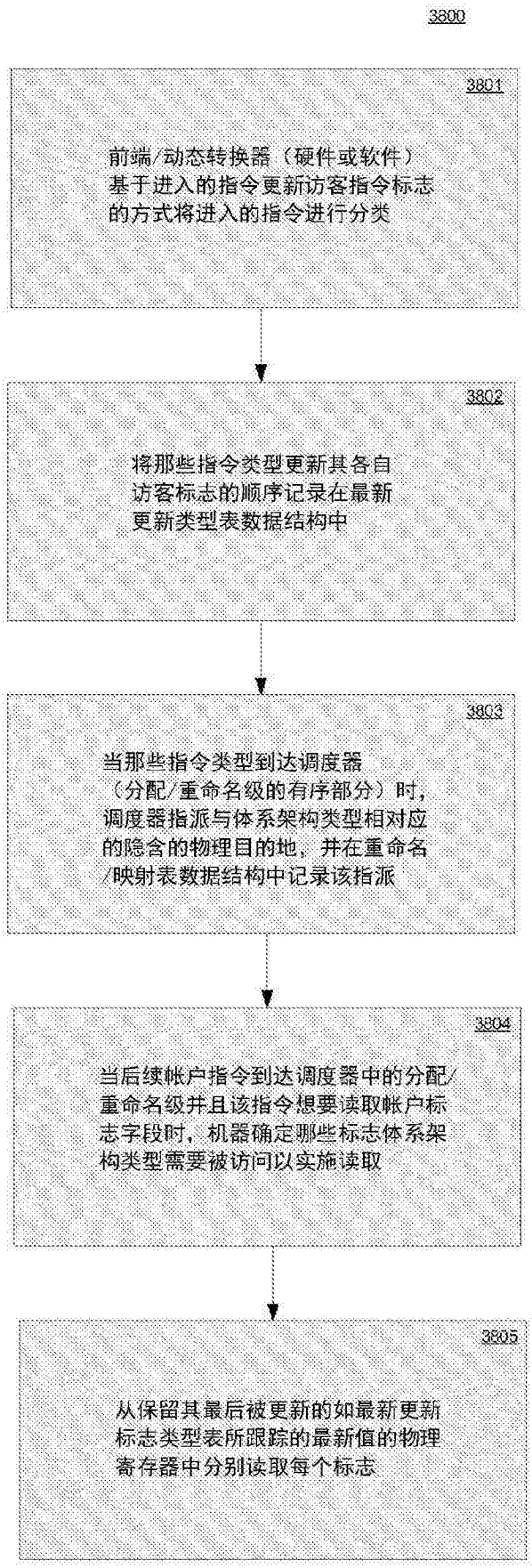

3800
3801
前端/动态转换器（硬件或软件）基于进入的指令更新访客指令标志的方式将进入的指令进行分类
3802
将那些指令类型更新其各自访客标志的顺序记录在最新更新类型表数据结构中
3803
当那些指令类型到达调度器（分配/重命名级的有序部分）时，调度器指派与体系架构类型相对应的隐含的物理目的地，并在重命名/映射表数据结构中记录该指派
3804
当后续帐户指令到达调度器中的分配/重命名级并且该指令想要读取帐户标志字段时，机器确定哪些标志体系架构类型需要被访问以实施读取
3805
从保留其最后被更新的如最新更新标志类型表所跟踪的最新值的物理寄存器中分别读取每个标志

图 38