

[19] 中华人民共和国国家知识产权局

[51] Int. Cl⁷
G06F 9/46
G06F 1/32



[12] 发 明 专 利 说 明 书

专利号 ZL 00818879.3

[45] 授权公告日 2005 年 10 月 12 日 [11] 授权公告号 CN 1222878C

[22] 申请日 2000.10.11 [21] 申请号 00818879.3
[30] 优先权
[32] 1999.12.9 [33] US [31] 09/458,589
[86] 国际申请 PCT/US2000/028213 2000.10.11
[87] 国际公布 WO2001/041529 英 2001.6.14
[85] 进入国家阶段日期 2002.8.8
[71] 专利权人 英特尔公司
地址 美国加利福尼亚州
[72] 发明人 D·罗杰斯 B·托尔 A·伍德
审查员 赵晓春

[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 勇

权利要求书 4 页 说明书 35 页 附图 23 页

[54] 发明名称 在一个多线程处理器内停用时钟信号的方法和装置

[57] 摘要

一种方法，包含：根据在一个多线程处理器内支持的多个线程中的每一个，保持一个待决事件的一个指示。还保持用于多个线程中每一个的一个活动或者不活动状态的一个指示。检测一个时钟停用条件。可以由缺少相对于多个线程中每一个的待决事件和用于多个线程中每一个的不活动状态来指示这个时钟停用条件。如果一个时钟信号被启用了，则响应于该时钟禁止条件的检测，相对于在该多线程处理器内的至少一个功能单元，停用该时钟信号。

I S S N 1 0 0 8 - 4 2 7 4

1. 一种用于对多线程处理器中时钟信号进行管理的方法, 包含:
相对于在一个多线程处理器内支持的多个线程中的每一个, 保持一个待决事件的一个指示;

- 5 为在该多线程处理器内支持的多个线程中的每一个, 保持一个活动或者不活动状态的一个指示;

检测由相对于多个线程中每一个的非待决事件指示、和用于这多个线程中每一个的不活动状态的指示所指示的一个时钟停用条件; 以及

- 10 如果一个时钟信号被启用了, 则响应于该时钟停用条件的检测, 相对于在该多线程处理器内的至少一个功能单元, 停用该时钟信号。

2. 如权利要求 1 所述的方法, 包含: 检测由相对于在所述多线程处理器内支持的多个线程中的至少一个线程的一个待决事件指示、或者用于在所述多线程处理器内支持的多个线程中的至少一个线程的一个活动状态的指示来指示的一个时钟启用条件, 以及, 如果一个时钟信号被停用了, 则响应于该时钟启用条件的检测, 相对于在该多线程处理器内的至少一个功能单元, 启用该时钟信号。

3. 如权利要求 1 所述的方法, 包含: 提供一个总线访问操作的一个指示, 并且仅仅在没有所述总线访问操作的指示的情况下, 检测时钟停用条件。

4. 如权利要求 3 所述的方法, 其特征在于: 所述总线访问操作包含一个探听操作, 而且其中当该总线访问操作在一个预定的探听期间是活动的时, 提供该总线访问操作的指示。

5. 如权利要求 2 所述的方法, 包含: 为由所述多线程处理器支持的多个线程中的至少一个第一线程保持一个事件禁止指示, 其中该事件禁止指示标识相对于第一线程的至少一个事件, 该事件不构成触发第一线程在不活动和活动状态之间的一次转换的一个事件。

6. 如权利要求 5 所述的方法, 其特征在于: 时钟停用条件和时钟启用条件的检测受到相对于第一线程的事件禁止指示的支配, 而且如果由用于第一个线程的事件禁止指示指示一个待决事件被禁止了, 则检测到时钟停用条件或者没有检测到一个时钟启用条件。

7. 如权利要求 1 所述的方法, 其特征在于: 维护相对于多个线程

中每一个的线程动作指示包含：保持一个活动线程状态机，其为支持的多个线程中的每一个输出一个信号以指示一个相应的线程是在活动状态还是在非活动状态中。

5 8. 如权利要求 1 所述的方法，其特征在于：相对于所述至少一个功能单元停用时钟信号包含选通该时钟信号。

9. 如权利要求 1 所述的方法，其特征在于：相对于所述至少一个功能单元停用该时钟信号包含：在一个处理器时钟中心上禁止一个处理器时钟信号的传送。

10 10. 如权利要求 1 所述的方法，其特征在于：相对于所述至少一个功能单元停用该时钟信号包含：停用一个产生一个处理器时钟信号的锁相回路。

11. 如权利要求 1 所述的方法，其特征在于：停用时钟信号包含：保持提供一个时钟信号到该多线程处理器的总线插脚，以便当该时钟信号被停用时启用一个总线访问操作或者事件的检测。

15 12. 一种用于对多线程处理器中时钟信号进行管理的装置，包含：

一个待决事件指示器，其相对于在一个多线程处理器内支持的多个线程中的每一个，保持一个待决事件的一个指示；

20 一个活动线程指示器，其为在多线程处理器内支持的多个线程中的每一个，保持一个活动或者不活动状态的一个指示；以及

时钟控制模块，检测由相对于多个线程中每一个的非待决事件的指示和相对于多个线程中每一个的一个不活动状态指示的一个时钟停用条件，并且响应于该时钟停用条件的检测，相对于在该多线程处理器内的至少一个功能单元停用一个时钟信号。

25 13. 如权利要求 12 所述的装置，其特征在于：该时钟控制模块检测由相对于在多线程处理器内支持的多个线程中的至少一个线程的一个待决事件指示所指示的一个时钟启用条件，并且如果该时钟信号被停用了，则响应于该时钟启用条件的检测，相对于在该多线程处理器内的至少一个功能单元启用该时钟信号。

30 14. 如权利要求 12 所述的装置，其特征在于：该时钟控制模块检测由用于在多线程处理器内支持的多个线程中至少一个线程的一个活动状态指示所指示的一个时钟启用条件，并且如果该时钟信号被停

用了，则响应于该时钟启用条件的检测，相对于在该多线程处理器内的至少一个功能单元启用该时钟信号。

15 15. 如权利要求 12 所述的装置，包含提供了一个总线访问操作的一个指示的总线探听模块，而且其特征在于：该时钟控制模块仅仅在没有来自于该总线探听模块的一个总线访问操作的一个指示的情况下，检测该时钟停用条件。

16. 如权利要求 15 所述的装置，其特征在于：该总线访问操作包含一个探听操作，而且，当该总线访问操作活动持续一个预定探听周期时，该总线探听模块提供该总线访问操作的指示。

10 17. 如权利要求 13 所述的装置，包含：一个事件禁止指示器，为由该多线程处理器支持的多个线程中的至少一个第一线程保持一个事件禁止指示，该事件禁止指示标识有关该第一个线程的至少一个事件，该事件不构成触发第一个线程在不活动和活动状态之间的一次转换的一个事件。

15 18. 如权利要求 17 所述的装置，其特征在于：时钟控制模块检测该时钟停用条件和时钟启用条件受到有关第一个线程的事件禁止指示的支配，如果由用于第一个线程的事件禁止指示指示一个待决事件被禁止了，则该时钟控制模块检测到了该时钟停用条件，或者没有检测到一个时钟启用条件。

20 19. 如权利要求 12 所述的装置，包含：一个活动线程状态机，其为支持的多个线程中的每一个输出一个信号，来指示一个相应的线程是在活动状态还是在非活动状态中。

20. 如权利要求 12 所述的装置，其特征在于：相对于至少一个功能单元，该时钟控制模块通过选通该时钟信号来停用该时钟信号。

25 21. 如权利要求 12 所述的装置，其特征在于：相对于至少一个功能单元，该时钟控制模块通过在一个处理器时钟中心上禁止一个处理器时钟信号的传送来停用该时钟信号。

30 22. 如权利要求 12 所述的装置，其特征在于：相对于至少一个功能单元，该时钟控制模块通过停用一个产生一个处理器时钟信号的锁相回路来停用该时钟信号。

23. 如权利要求 12 所述的装置，其特征在于：该时钟控制模块相对于至少一个功能单元停用该时钟信号，但是保持提供一个时钟信

号到多线程处理器的总线插脚，以便当该时钟信号被停用时，启用一个总线访问操作或者事件的检测。

24. 一种用于对多线程处理器中时钟信号进行管理的装置，包含：

5 第一装置，用于相对于在一个多线程处理器内支持的多个线程中的每一个，保持一个待决事件的一个指示；

 第二装置，用于为在该多线程处理器内支持的多个线程中的每一个，保持一个活动或者不活动状态的一个指示；以及

10 第三装置，用于检测由相对于多个线程中每一个的非待决事件的指示和相对于多个线程中每一个的一个不活动状态所指示的一个时钟停用条件，以及响应于该时钟停用条件的检测，相对于在该多线程处理器内的至少一个功能单元停用一个时钟信号。

在一个多线程处理器内停用时钟信号的方法和装置

发明领域

- 5 本发明通常涉及多线程处理器领域，更具体地涉及一种用于在一个多线程（MT）处理器内停用一个时钟信号的方法和装置。

发明背景技术

- 多线程（MT）处理器设计近来已经被认为是一个越来越吸引人的、用于增加处理器性能的选择。在一个处理器内的多线程，特别提供了
10 用于更有效地使用各种处理器资源、和尤其用于更有效地使用在一个处理器内的执行逻辑的可能性。具体来说，通过传送多个线程到一个处理器的执行逻辑，否则将由于在一个特定线程处理过程中的一个停止或者其它延迟而成为空闲的时钟周期可以被利用以服务于一个进一步的线程。在一个特定线程处理过程中的一个停止可以由在一个处理器流水线内的许多事故引起。例如，对于包含在一个线程内的一条指令来说，一个高速缓存未中或者一个分支预测错误（即，一个长的延迟操作）通常会导致相关线程处理过程的停止。长延迟操作在执行逻辑效率上的负面影响由近来在执行逻辑吞吐量中的增加而被加重，其中该增加在存储器访问和检索速率中有超前的提高。

- 20 鉴于由许多流行的操作系统，诸如 Windows NT®和 UNIX 操作系统、对这种多线程应用的支持，多线程计算机应用也变得越来越普遍。多线程计算机应用尤其在多媒体领域是有效率的。

- 依据在相关处理器内使用的线程交叉或切换方案，多线程处理器大致可以被分为两个类别（即，精细或者粗糙设计）。精细多线程设计
25 在一个处理器内支持多个活动的线程，而且通常在一个周期接一个周期的基础上交叉两个不同的线程。粗糙多线程设计一般在发生某些长延迟事件、诸如一个高速缓存未中时交叉不同线程的指令。一个粗糙多线程设计在 1996 年 5 月、第 23 期 Annual International Symposium on Computer Architecture 的 203 - 212 页上、由
30 Eickemayer, R.、Johnson, R. 等人所著的“Evaluation of Multithreaded Uniprocessors for Commercial Application Environments”中进行了讨论。在由 Massachusetts Norwell 的

Kluwer Academic 出版社于 1994 年出版的、由 R. A. Iannucci 等人编辑的 Multithreaded Computer Architectures: A Summary of the State of the Art 的 167 - 200 页上、由 Laudon, I. Gupta, A 所著的 “Architectural and Implementation Tradeoffs in the Design of Multiple Context Processors” 中, 进一步讨论了在精细和粗糙设计之间的区别。Laudon 进一步提出了一种把一个精细设计中的一个周期接一个周期的切换和一个粗糙设计中的完全流水线互锁(或者分块方案)组合在一起的交叉方案。为此, Laudon 提出了一条使一个特定线程(或者环境)不可用持续一个特定周期数目的“后退”指令。

5 可以依据预定事件、诸如一个高速缓存未中的发生, 发布这样的一条“后退”指令。这样, Laudon 通过简单地使一个线程不可用而避免了执行一个实际的线程切换。

一个用于一个处理器的多线程结构在一个无序的推测性执行处理器体系结构的环境中给出了许多进一步的难题。更具体地说, 当考虑多线程时, 可能导致在一个指令流的流动中一个意外变化的事件(例如, 转移指令、例外或者中断)的处理是复杂的。在一个实现了在多个线程之间共享资源(即有有限的、或者没有功能单元的复制用于由该处理器支持的每一个线程)的处理器中, 关于一个特定线程的事件发生的处理是复杂的, 这是因为在处理这种事件中必须考虑进一步的线程。

15 20

在一个多线程处理器内实现了资源共享的地方, 响应于在该多线程处理器内正被服务的线程状态中的变化, 试图增加该共享资源的使用率是更令人期望的。

发明概述

25 根据本发明, 提供了一种方法, 包含: 相对于多线程处理器内支持的多个线程中的每一个, 保持一个待决事件的一个指示。为在该多线程处理器内支持的多个线程中的每一个保持一个活动或者不活动状态的一个指示。检测由相对于多个线程中每一个的非待决事件指示和用于每一个线程的不活动状态所指示的一个时钟停用条件。如果一个时钟信号被启用了, 则响应于该时钟停用条件的检测, 相对于在该多线程处理器内的至少一个功能单元, 停用该时钟信号。

30

通过附图及其后的详细说明, 本发明的这些及其它特征和优点将

会是显然的。

附图简要说明

本发明通过举例进行说明而且不局限于附图中的图表，在附图中类似的附图标记表示类似的单元，其中：

5 图 1 是一个框图，说明了一个具有多线程支持的处理器中的一条流水线的的一个实施例。

图 2 是一个框图，其以一个通用多线程微处理器的形式说明了一个处理器的一个示范性实施例。

10 图 3 是一个框图，说明了一个示范多线程微处理器中的所选组件，并且具体描述了各种提供一个缓存（或者存储）性能的功能单元，其中该缓存被逻辑分区以容纳多个线程。

图 4 是依据一个实施例、说明了一个无序群集的一个框图。

图 5 是一个寄存器别名表和一个寄存器堆的图解表示，并且在一个实施例内使用。

15 图 6A 是一个依据一个实施例、说明了有关一个重新排序缓存的详情的框图，其中该缓存被逻辑分区以服务在一个多线程处理器内的多个线程。

图 6B 是依据一个实施例的一个待决事件寄存器和一个事件禁止寄存器的一个图解表示。

20 图 7A 是一个流程图，其依据一个实施例说明了一种在一个多线程处理器内处理一个事件的方法。

图 7B 是一个流程图，其依据一个实施例说明了一种在一个多线程处理器内处理一个“虚拟清空”事件的方法。

25 图 8 是可以由依据一个实施例、在一个多线程处理器内实现的一个事件检测器检测到的多个示例事件的一个图解表示。

图 9 和 10 中的框图分别显示了在一个诸如如图 6A 所示的示例重新排序缓存中的一个重新排序表的示例内容。

30 图 11A 是一个流程图，其依据一个示例实施例说明了一种在一个支持至少第一和第二线程的多线程处理器内执行一个清除（或者清空）操作的方法。

图 11B 是一个框图，其依据一个示例实施例说明了配置逻辑，其中该逻辑根据一个活动线程状态机的输出进行操作以配置一个功能单

元。

图 12 是一个时序图，其依据一个实施例说明了一个清空信号的断定。

5 图 13 是一个流程图，其依据一个实施例说明了一种在一个多线程处理器内提供对一个事件处理器的排它访问的方法。

图 14 是一个状态图，其依据一个实施例描述了在一个多线程处理器内实现的一个排它访问状态机的操作。

图 15 是一个状态图，其依据一个实施例说明了可以由在一个多线程处理器内实现的一个活动线程状态机占据的状态。

10 图 16A 是一个流程图，依据一个实施例说明了一种在一个多线程处理器内检测到用于一个活动线程的一个睡眠事件时，退出该活动线程的方法。

图 16B 是一个依据一个实施例、当退出一个线程时存储寄存器的状态和解定位的图解表示。

15 图 17 是一个流程图，依据一个实施例说明了一种在检测到用于一个不活动线程的一个中断事件时、把该线程从一个不活动状态转换到一个活动状态的方法。

20 图 18 被一个流程图，其依据一个实施例说明了一种管理向在一个多线程处理器内的至少一个功能单元启用和停用一个时钟信号的方法。

图 19A 是一个框图，依据一个实施例说明了用于在一个多线程处理器内启用和停用一个时钟信号的时钟控制逻辑。

图 19B 是一个示意图，显示了如图 19A 所示的时钟控制逻辑的一个实施例。

25 详细说明

这里描述了一种用于在一个多线程处理器内管理一个时钟信号的方法和装置。在下面的描述中，为了说明起见，阐述了许多具体的细节以便提供对本发明的一个彻底了解。然而，对本领域技术人员来说，显然可以实践本发明而不用这些细节。

30 为了本说明书起见，使用的术语“事件”应当包含在一个处理器内部或外部的任何事件，其导致对在一个处理器内的一个指令流（宏或者微指令）服务的改变或者中断。据此，采用的术语“事件”应当

包含，但不局限于，可以在该处理器内部或外部产生的转移指令处理、例外和中断。

为了本说明书起见，术语“处理器”应当用来涉及能够执行一个指令（例如，宏或者微指令）序列的任何机器，而且应当用来包含，
5 但不局限于，通用微处理器、专用微处理器、图形控制器、音频控制器、多媒体控制器、微型控制器或者网络控制器。此外，采用的术语“处理器”应当尤其涉及复杂指令集计算机（Complex Instruction Set Computers, CISC）、精简指令集计算机（Reduced Instruction Set Computers, RISC）、或者超长指令字（Very Long Instruction
10 World, VLIW）处理器。

此外，采用的术语“清除点”应当包含：通过一个流标记或者其他指令在一个指令流（包含一个微指令或者宏指令流）中提供的、在该指令流中一个可以处理一个事件的位置处的任何指令。

采用的术语“指令”应当包含，但不局限于，一条宏指令或者一
15 条微指令。

本发明的某些示例实施例被描述为主要以硬件或者软件实现。尽管如此，但是那些本领域技术人员还可以理解：许多特征可以容易地以硬件、软件或者硬件和软件的一个组合来实现。用于实现本发明实施例的软件（例如，任何微指令以及宏指令）可以完全或者至少部分
20 地驻留在一个可由一个处理器访问及/或在处理器本身内的主存储器（例如，在一个高速缓存或者一个微码定序器内）中。例如，事件处理器和状态机可以以从一个微码定序器发送的微码形式实现。

软件可以进一步经由网络接口设备传送或者接收。

为了这个说明书起见，采用的术语“机器可读介质”应当包含任
25 何能够存储或者编码一个指令序列用于由该机器执行并且导致该机器执行本发明中的任何一种方法的介质。采用的术语“机器可读介质”因此应当包含，但不局限于，固态存储器、光和磁盘、以及载波信号。

处理器流水线

图1是说明了处理器流水线10的一个实施例的一个高级框图。流
30 水线10包含多个流水线阶段，其从一个检索指令（例如，宏指令）并且把该指令送入流水线10的取指令流水线阶段12开始。例如，一条宏指令可以从与该处理器集成或者与该处理器紧密相关的一个高速缓

冲存储器中检索得到，或者可以经由一条处理器总线从一个外部主存储器检索得到。宏指令从取指令阶段 12 传送到一个解码流水线阶段 14，其中宏指令被翻译为适于在该处理器内执行的微指令（也称为“微码”）。然后该微指令向下传送到一个分配流水线阶段 16，在那儿依据可用性和需要把处理器资源分配给各种微指令。然后该微指令在被收回之前在一个执行阶段 18 中被执行，或者在一个收回流水线阶段 20 中被“写回”（例如，提交到一个体系结构状态）。

微处理器体系结构

图 2 是一个框图，其以一个通用多线程微处理器的形式，说明了一个处理器 30 的一个示例实施例。如下所述，处理器 30 是一个多线程（MT）处理器，而且因此能够处理多个指令线程（或者环境）。然而，在该说明书中下面提供的许多示教不专门针对一个多线程处理器，并且可能在一个单线程处理器中找到应用。在一个示例实施例中，处理器 30 可以包含一个能够执行 Intel 结构指令集的 Intel 结构（IA）微处理器。这样一个 Intel 结构微处理器的示例是由 California, Santa Clara 的 Intel 公司制造的 Pentium Pro®微处理器或者 Pentium III®微处理器。

在一个实施例中，处理器 30 包含一个有序前端和一个无序后端。该有序前端包含一个总线接口单元 32，其起到在处理器 30 和一个其中可以使用该处理器 30 的计算机系统中的其它组件（例如，主存储器）之间的流水线作用。为此，总线接口单元 32 把处理器 30 连接到一条处理器总线（没有显示），经由该处理器总线可以在处理器 30 处接收数据和控制信息并且从处理器 30 传送数据和控制信息。总线接口单元 32 包含控制通过该处理器总线进行通信的前侧总线（FSB）逻辑 34。总线接口单元 32 进一步包含一个总线队列 36，其对于通过该处理器总线的通信提供一个缓冲功能。总线接口单元 32 被显示为从一个在处理器 30 内提供一个局部存储器性能的存储器执行单元 42 中接收总线请求 38，以及发送探测或者总线返回到该存储器执行单元 42。该存储器执行单元 42 包含一个统一的数据与指令高速缓存 44、一个数据翻译后援缓存（TLB）46、以及存储器排序缓存 48。存储器执行单元 42 从一个微指令翻译引擎 54 接收指令取出请求，并且传送原始指令（即，编码的宏指令）到该微指令翻译引擎 54，其中该微指令翻译引擎 54 把接收到的宏

指令翻译成为一组相应的微指令。

因为如果发生一个轨迹高速缓存未命中时，该微指令翻译引擎 54 进行操作以传送微指令到一个轨迹高速缓存 62，所以该微指令翻译引擎 54 事实上操作作为一个轨迹高速缓存“未命中处理器”。为此，如果发生一个轨迹高速缓存未命中，则微指令翻译引擎 54 就起到提供取指令和解码流水线阶段 12 和 14 的作用。显示的微指令翻译引擎 54 包含：一个下一指令指针 (NIP) 100、一个指令翻译后援缓冲器 (TLB) 102、一个转移预测器 104、一个指令流缓存 106、一个指令预解码器 108、指令控制逻辑 110、一个指令解码器 112、和一个转移地址计算器 114。下一指令指针 100、TLB 102、转移预测器 104 以及指令流缓存 106 一起构成了一个转移预测单元 (BPU)。指令解码器 112 和转移地址计算器 114 一起组成了一个指令翻译 (IX) 单元 113。

下一指令指针 100 发布下一个指令请求到统一的高速缓存 44。在其中处理器 30 包含一个能够处理两个线程的多线程微处理器的示例实施例中，下一指令指针 100 可以包含一个多路复用器 (MUX) (没有显示)，其在与第一个或者第二个线程相关的指令指针之间进行选择用于包含在由此发布的下一个指令请求内。在一个实施例中，假定用于两个线程的指令都已经被请求了，而且用于两个线程的指令流缓存 106 资源都还没有被用尽，则该下一指令指针 100 将在一个周期接一个周期 (“往复转换”) 的基础上，交叉用于第一和第二线程的下一个指令请求。取决于初始请求地址是否在一个 32 字节或者 64 字节定位行的上半部，下一指令指针请求可以是 16、32 或者 64 字节。下一指令指针 100 可以由转移预测器 104、转移地址计算器 114 或者轨迹高速缓存 62 重定向，其中一个轨迹高速缓存未命中请求是最高优先级的重定向请求。

当下一指令指针 100 做出一个到统一高速缓存 44 的指令请求时，它产生一个两位的“请求标识符”，该标识符与该指令请求相关联并且起到一个用于该相关指令请求的“标记”作用。当响应于一个指令请求返回数据时，该统一高速缓存 44 和数据一起返回下列标识或者标识符：

1. 由下一指令指针 100 提供的“请求标识符”；
2. 一个标识返回块的三位“块标识符”；以及

3. 一个标识返回数据所属线程的“线程标识符”。

下一指令请求从下一指令指针100传送到指令TLB 102, 该TLB 102执行一个地址查找操作, 并且传送一个物理地址到统一的高速缓存44。统一的高速缓存44传送一个相应的宏指令到指令流缓存106。每一个下一指令请求也都从下一指令指针100直接传送到该指令流缓存106, 以便允许该指令流缓存106标识从该统一高速缓存44接收到的一个宏指令属于哪个线程。来自于第一和第二线程的宏指令然后从该指令流缓存106发布到指令预解码器108, 指令预解码器108对于一个接收的指令流(宏指令)执行多个长度计算和字节标记操作。具体地说, 该指令预解码器108产生一系列字节标记向量, 其尤其服务于把在该指令流内、要传送到指令控制逻辑110的宏指令划分开。

指令控制逻辑110然后为了解码, 利用该字节标记向量来引导离散的宏指令到指令解码器112中。为了转移地址计算起见, 宏指令也从该指令控制逻辑110传送到转移地址计算器114。微指令然后从该指令解码器112传送到轨迹传送引擎60。

在解码期间, 流标记与一个宏指令被翻译成的每一个微指令相关联。一个流标记指示相关微指令的一个特征, 而且可以, 例如, 指示相关的微指令是在表示一个宏指令的一个微码序列中的第一个或者最后一个微指令。流标记包含一个“宏指令开始”(BOM)和一个“宏指令末端”(EOM)流标记。根据本发明, 解码器112可以进一步解码该微指令, 以具有与此相关的共享资源(多处理器)(SHRMP)流标记和同步(SYNC)流标记。具体来说, 一个共享资源流标记把一个微指令标识作为在一个特定线程内的位置, 在该位置处, 和在该线程中的别处相比, 该线程可以被中断(例如, 重新启动或者暂停)而具有较少的负面后果。在本发明的一个示例实施例中, 解码器12被构造为用一个共享资源流标记来标记组成一个原始宏指令的结尾或者开始的微指令、以及在较长微码序列中的间断点。一个同步流标记把一个微指令标识作为在一个特定线程内的一个位置, 在该位置处, 响应于例如在另一个线程内的一个同步指令, 该线程可以与另一个线程同步。为了本说明书起见, 采用的术语“同步”应当涉及在至少一个线程中标识至少一个第一点, 在该点处, 可以根据那个线程及/或至少一个进一步线程, 相对于在那个线程或者在另一个线程中的一个第二点, 减小

或者降低对该处理器的破坏来修改处理器状态。

在本发明的一个示例实施例中，解码器 112 被构造为标记位于选定宏指令边界处的微指令，其中在同一个处理器中共存的线程之间共享的状态能够由一个线程进行改变而不会不利地影响其它线程的执行。

被解码的指令（即，微指令）从微指令翻译引擎 54 中发送到一个轨迹传送引擎 60。该轨迹传送引擎 60 包含一个轨迹高速缓存 62、一个轨迹转移预测器（BTB）64、一个微码定序器 66 和一个微码（uop）队列 68。轨迹传送引擎 60 起到一个微指令高速缓存的作用，并且是用于一个下游执行单元 70 的主要微指令来源。通过在处理器流水线内提供一个微指令高速缓存功能，该轨迹传送引擎 60，具体来说该轨迹高速缓存 62，允许调整由该微指令翻译引擎 54 执行的翻译工作以提供增加了的微指令带宽。在一个示例实施例中，该轨迹高速缓存 62 可以包含一个 256 组、8 路组的相关存储器。在本示例实施例中，术语“轨迹”可以涉及存储在该轨迹高速缓存 62 的入口内的一序列微指令，每一个入口包含指向组成该轨迹的、在前和正在进行的微指令的指针。这样，因为在当前访问完成之前，为了获得一个后续微指令而要被下一个访问的入口地址是已知的，所以该轨迹高速缓存 62 有助于高性能的定序。在一个实施例中，轨迹可以被看作由轨迹头彼此区别的指令“块”，并且当遇到一个间接转移或者到达多个给出的阈值条件中的一个时被终止，其中这些阈值条件诸如在单个轨迹中可以容纳的条件转移数目、或者可以组成一个轨迹的最大微指令总数。

轨迹高速缓存转移预测器 64 提供了关于在该轨迹高速缓存 62 内的轨迹本地转移预测。该轨迹高速缓存 62 和微码定序器 66 提供微指令到微码队列 68，然后微指令从该微码队列 68 送到一个无序执行群集。显示的微码定序器 66 进一步包含多个以微码形式实现的事件处理器 67，这些事件处理器 67 响应于一个诸如例外或者中断的事件的发生而在处理器 30 内实现多个操作。如将在下面进一步仔细描述的那样，事件处理器 67 由包含在处理器 30 后端中的一个寄存器重命名器 74 内的一个事件检测器 188 激活。

处理器 30 可以被看作具有一个包含了总线接口单元 32、存储器执行单元 42、微指令翻译引擎 54 和轨迹传送引擎 60 的有序前端，以及

一个将在下面进行详细描述无序后端。

从微码队列 68 发送的宏指令被接收到一个无序群集 71 中，该无序群集 71 包含一个调度器 72、一个寄存器重命名器 74、一个分配器 76、一个重新排序缓存 78 以及一个重放队列 80。该调度器 72 包含一
5 组保留位置，并且操作来调度和发送用于由执行单元 70 执行的微指令。寄存器重命名器 74 相对于隐藏的整数和浮点数寄存器执行一个寄存器重命名功能（在一个处理器 30 执行 Intel 结构指令集的地方，可以被使用来代替任意八个通用寄存器或者任意八个浮点寄存器）。分配器 76 依据可用性和需要进行操作，以把执行单元 70 和群集 71 的资
10 源分配到微指令。如果没有足够的资源可以用来处理一个微指令的话，则分配器 76 负责声明一个停止信号 82，该信号通过轨迹传送引擎 60 传送到微指令翻译引擎 54，如在 58 所示的那样。已经让它们的源字段由寄存器重命名器 74 调整的微指令以严格的程序次序放置到重新排序缓存 78 中。当在该重新排序缓存 78 内的微指令已经完成执行并且准备收回时，它们然后从一个重新排序缓存中删除并且以一种有序方式（即，依据一个原有程序次序）被检索。重放队列 80 传送要被重放的微指令到执行单元 70。

显示的执行单元 70 包含一个浮点数执行引擎 84、一个整数执行引擎 86、和一个 0 级数据高速缓存 88。在一个处理器 30 执行 Intel 结
20 构指令集的示例实施例中，该浮点数执行引擎 84 可以进一步执行 MMX 指令和流 SIMD（单指令，多数据）扩展（SSE）。

多线程实现

在图 2 中所示的处理器 30 的示例实施例中，可以有有限的资源复制来支持一个多线程性能，并且因此必须在线程当中实现某种程度的
25 资源共享。将要理解：使用的资源共享方案取决于该处理器能够同时处理的线程数目。因为在一个处理器内的功能单元一般提供某些缓存（或者存储）功能和传送功能，所以资源共享的发布可以被看作包含（1）存储和（2）处理/传送带宽共享组件。例如，在一个支持同时处理两个线程的处理器中，在各个功能单元内的缓存资源可以被静态或者逻辑地在两个线程之间进行划分。类似地，由一个路径提供的、
30 用于在两个功能单元之间传送信息的带宽必须在两个线程之间进行划分和分配。因为可以在一个处理器流水线内的多个位置处产生这些资

源共享发布，根据该特定位置的指令和特征，在这些不同位置处可以使用不同的资源共享方案。将要理解：鉴于变化的功能和操作特性，不同的资源共享方案可能适于不同的位置。

图 3 是一个框图，说明了用于在图 2 中说明的处理器 30 的一个实施例的选定组件，并且描述了提供一个缓存性能的不同功能单元，该缓存性被逻辑划分以容纳两个线程（即线程 0 和线程 1）。通过分配在一个缓存资源内的一个第一预定组入口到一个第一线程以及分配在该缓存资源内的一个第二预定组入口到一个第二线程，可以实现一个功能单元的缓存（或者存储）和处理功能用于两个线程的逻辑划分。然而，在替换实施例中，缓存还能够被动态共享。具体来说，这可以通过提供两对读和写指针来实现，其中第一对读和写指针与一个第一线程相关联，而第二对读和写指针与一个第二线程相关联。第一组读和写指针可以局限于在一个缓存资源内的一个第一预定数目入口，而第二组读和写指针可以局限于在同一个缓存资源内的一个第二预定数目入口内。在说明的实施例中，显示的指令流缓存 106、轨迹高速缓存 62、和一个指令队列 103 的每一个都提供了一个在第一和第二线程之间进行逻辑划分的存储容量。

无序群集 (71)

图 4 是一个框图，说明了无序群集 71 的一个实施例的更多细节。群集 71 在处理器 30 内提供了保留位置、寄存器重命名、重放和收回功能。群集 71 从轨迹传送引擎 60 接收微指令、分配资源到这些微指令、为每一个微指令重命名源和目的寄存器、调度微指令用于发送到合适的执行单元 70、处理由于数据推测而被重放的微指令、以及最后最后收回微指令（即，确认该微指令到一个永久体系结构状态）。

在群集 71 处接收的微指令同时被传送到一个寄存器别名表格 120 和分配及释放列表管理逻辑 122。寄存器别名表格 120 负责把逻辑寄存器名翻译到由调度器 72 和执行单元 70 使用的物理寄存器地址。更具体地说，参见图 5，该寄存器别名表格 120 重命名保持在一个物理寄存器堆 124 内的整数、浮点数和段寄存器。显示的寄存器堆 124 包含 126 个被别名到八个（8）体系结构寄存器的物理寄存器。在说明的实施例中，该显示的寄存器别名表格 120 包含一个前端表格 126 和一个后端表格 128 用于由相应的处理器 30 前端和后端使用。在该寄存器别名表

格 120 内的每一个入口都与一个体系结构寄存器相关联，或者被看作一个体系结构寄存器，而且包含一个指向在该寄存器堆 124 内的一个位置的指针 130，在该位置处存储属于相关体系结构寄存器的数据。以这种方式，可以解决由一个传统微处理器体系结构提供的、指定一个
5 相对小数量的体系结构寄存器的问题。

分配和释放列表管理逻辑 122 负责在群集 71 内的资源分配和状态恢复。逻辑 122 分配下列资源到每一个微指令：

1. 一个顺序号，其给予每一个微指令以当在该群集 71 内处理该微指令时，在一个线程内跟踪它的逻辑次序。属于每一个微指令的顺序
10 号和用于该微指令的状态信息被一起存储在该重新排序缓存 162 内的一个表格 180（显示在下面图 10 中）内。

2. 一个释放列表管理入口，其给予每一个微指令，以在一个状态恢复操作的情况下允许微指令的历史记录被跟踪和恢复。

3. 一个重新排序缓存（ROB）入口，由该顺序号索引。

15 4. 一个物理寄存器堆 124 入口（被称为“弹子”），在其内该微指令可以存储有用的结果。

5. 一个载入缓存（没有显示）入口。

6. 一个停止缓存（没有显示）入口。

20 7. 一个指令队列入入口（例如，到一个存储器指令队列或者一个通用指令地址队列的入口，将在下面进行描述）。

如果逻辑 122 不能获得用于一个接收的微指令序列的必要资源的话，该逻辑 122 将要求轨迹传送引擎 60 停止微指令的传送直到有足够的资源变得可用为止。通过声明在图 2 中说明的停止信号 82 来传递这个请求。

25 考虑把在该寄存器堆 124 内的一个入口分配到一个微指令，图 5 显示了一个垃圾堆数组 132，其在该寄存器堆 124 内保持没有被分配给体系结构寄存器的入口记录（即，它们没有在该寄存器别名表格 120 内被指向）。逻辑 122 访问该垃圾堆数组 132 以标识在该寄存器堆 124 内、可以用来分配到一个接收的微指令的入口。逻辑 122 也负责重新
30 要求在该寄存器堆 124 内变得可用的入口。

逻辑 122 进一步保持一个释放列表管理器（FLM）134 以允许跟踪该体系结构寄存器。具体来说，当微指令被分配到此，该释放列表

管理器 134 保持一个对该寄存器别名表格 120 的改变的历史记录。该释放列表管理器 134 提供这样的性能，以在给定一个错误预测或者一个事件时，“展开”该寄存器别名表格 120 以指向一个非推测状态。该释放列表管理器 134 也“更新”在该寄存器堆 124 入口中的数据储存以 5 确保所有的状态信息是当前的。最后，在收回处，物理寄存器标识符从该释放列表管理器 134 传送到该垃圾堆数组 132 用于分配给一个进一步的微指令。

一个指令队列单元 136 以顺序程序次序传送微指令到一个调度器和记分板单元 (SSU) 138，并且保持和发送由该执行单元 70 需要的微指令信息。指令队列单元 136 可以包含两个明确的结构，即一个指令队列 (IQ) 140 和一个指令地址队列 (IAQ) 142。指令地址队列 142 是设计成当需要时传送关键信息（例如，微指令源、目的和延迟）到单元 138 的小型结构。指令地址队列 142 此外可以包含一个排队信息用于存储操作的存储器指令地址队列 (MIAQ) 和一个排队信息用于非 10 存储操作的通用指令地址排队 (GIAQ)。指令队列 140 存储较不关键的信息，诸如用于微指令的操作码和即时数据。当相关的微指令被读取和写入到调度器和记分板单元 138 中时，从该指令队列单元 136 中解分配该微指令。

调度器和记分板单元 138 通过确定每一个微指令源可以就绪的时间，以及何时合适的执行单元可用于发送，负责调度执行的微指令。如图 4 所示的单元 138 包含一个寄存器堆记分板 144、一个存储器调度器 146、一个矩阵调度器 148、一个慢微指令调度器 150 和一个浮点数调度器 152。 20

单元 138 通过检查保持在寄存器堆记分板 144 内的信息来确定何时该源寄存器是就绪的。为此，在一个实施例中，该寄存器堆记分板 144 具有 256 位，其跟踪对应于在该寄存器堆 124 内的每一个寄存器的数据资源可用性。例如，当分配数据到相关入口或者有一个到该单元 138 的写操作时，可以清除用于在该寄存器堆 124 内的一个特定入口的记分板位。

存储器调度器 146 缓存存储器类的微指令、检查资源可用性、然后调度存储器类的微指令。矩阵调度器 148 包含两个紧密绑定的算术逻辑单元 (ALU) 调度器，其允许调度相关的背靠背微指令。浮点数调 30

度器 152 缓存和调度浮点数微指令，而慢微指令调度器 150 调度不被上述调度器处理的微指令。

显示的一个检验器、重放和收回单元 (CRU) 160 包含一个重新排序缓存 162、一个检验器 164、一个分段队列 166 和一个收回控制电路 168。单元 160 具有三个主要功能，即一个检验功能、一个重放功能和一个收回功能。具体地说，检验器和重放功能包含重新执行已经被不正确执行的微指令。该收回功能包含确认体系结构有序状态到处理器 30。更具体地说，检验器 164 进行操作以确保每一个微指令已经正确地执行了正确的数据。如果该微指令没有用正确数据进行执行（例如，由于一个错误预测的转移），则相关的微指令被重放以用正确数据执行。

重新排序缓存 162 通过以程序次序收回微指令来负责确认体系结构状态到该处理器 30。由一个收回控制电路 168 产生的一个收回指针 182 指示在该重新排序缓存 162 内正被收回的一个入口。当该收回指针 182 移动通过在一个入口内的一个微指令时，则释放在该释放列表管理器 134 内的相应入口，而且现在可以回收相关的寄存器堆入口并且把它们传送到垃圾堆数组 132。显示的收回控制电路 168 也实现了一个活动线程状态机 171，它的目的和功能将在下面进行说明。收回控制电路 168 控制把在该重新排序缓存 162 中保持的推测性结果确认到在该寄存器堆 124 内的相应体系结构状态。

重新排序缓存 162 也负责处理内部和外部事件，如将在下面进一步详细描述的那样。当由该重新排序缓存 162 检测到一个事件发生时，就声明一个“清空”信号 170。清空信号 170 具有从该处理器流水线中清空所有当前在传送中的微指令的效果。重新排序缓存 162 也向该轨迹传送引擎 60 提供一个地址，从该地址开始排序微指令以服务该事件（即，从那个地址发送以微码体现的一个事件处理器 67）。

重新排序缓存 (162)

图 6A 是说明了有关重新排序缓存 162 的一个示例实施例的更多细节的一个框图，该缓存被逻辑划分以服务在该多线程处理器 30 内的多个线程。具体地说，显示的重新排序缓存 162 包含一个重新排序表格 180，当该处理器 30 以一个多线程模式进行操作时，该表格可以被逻辑划分以容纳用于第一和第二线程的入口。当以一个单线程模式操作

时，整个表格 180 可以被使用来服务该单个线程。在一个实施例中，表格 180 包含一个统一的存储器结构，当该存储器结构以多线程模式操作时，由两个 (2) 局限于在该表格 180 内的预定和明确入口组的收回指针 182 和 183 引用。类似地，当以一个单线程模式操作时，表格 180 由单个收回指针 182 引用。表格 180 包含对应于寄存器堆 124 中的每一个入口的一个入口，并且存储一个顺序号和以错误信息形式的状态信息、一个逻辑目的地址、以及一个用于在寄存器堆 124 内的每一个微指令数据入口的有效位。在表格 180 内的入口每一个都由构成用于每一个微指令的一个唯一标识符的顺序号进行索引。根据该顺序号，在该表格 180 内的入口以一种顺序和有序方式进行分配和解分配。除了其它流标记之外，显示的表格此外还存储一个用于每一个微指令的共享资源流标记 184 和一个同步流标记 186。

重新排序缓存 162 包含一个事件检测器 188，其被连接以接收以中断向量形式的中断请求以及访问在表格 180 内、由收回指针 182 和 183 引用的入口。显示的事件检测器 188 此外还输出清空信号 170 和清除信号 172。

假定用于一个特定线程（例如，线程 0）的一个特定微指令经历非转移错误预测、例外或者中断的话，则当该收回指针 182 或者 183 被递增以寻址相关入口时，保存在表格 180 内的入口中、用于该特定指令的信息将被收回到体系结构状态。在这种情况下，一个形成为收回控制电路 168 一部分的指令指针计算器 190 递增该宏或者微指令指针以指向 (1) 在寄存器堆 124 内的相应入口内指定的一个转移目标地址或者指向 (2) 如果不使用一个转移的话，下一个宏或者微指令。

如果已经发生了一个转移错误预测，则信息通过错误信息域传送到收回控制电路 168 和事件检测器 188。鉴于通过错误信息指示的转移错误预测，处理器 30 也许已经取出了至少某些不正确的指令，而且这些指令已经遍及该处理器流水线。因为在表格 180 内的入口以连续次序分配，所以在该错误预测转移微指令之后的入口是由该错误预测转移指令流沾染了的微指令。响应于在错误信息内为其记录了一个错误预测转移的一个微指令的试图收回，事件检测器 188 声明清除信号 172，其清除处理器整个无序后端的所有状态，并且因此刷新由在一个错误预测微指令之后的指令产生的无序后端所有状态。清除信号 172

的声明同时也阻塞随后取出的、可能位于处理器 30 的有序前端内的微指令的发布。

在该收回控制电路 168 内，依据通过一个收回微指令的错误信息通知的一个错误预测转移，IP 计算器 190 确保指令指针 179 及 / 或 181 被更新以表示正确的指令指针值。基于该转移是否将被使用，IP 计算器 190 用来自对应于表格 180 相关入口的寄存器堆入口的结果数据来更新指令指针 179 及 / 或 181，或者当不使用转移时递增指令指针 179 和 181。

事件检测器 188 还包含多个寄存器 200 用于保持关于为多个线程中的每一个检测的事件的信息。寄存器 200 包含一个事件信息寄存器 202、一个待决事件寄存器 204、一个事件禁止寄存器 206、和展开寄存器 208 以及一个管脚状态寄存器 210。每一个寄存器 202 - 210 都能够存储关于为一个特定线程产生的一个事件的信息。因此，用于多个线程的事件信息可以由寄存器 200 保持。

图 6B 是用于一个第一线程（例如，T0）的一个示例待决事件寄存器 204 和一个示例事件禁止寄存器 206 的一个示意图。

待决事件和事件禁止寄存器 204 和 206 为在多线程处理器 30 内支持的每一个线程提供。可以为每一个线程提供明确的寄存器 204 和 206，或者单个物理寄存器可以被逻辑划分以支持多个线程。

示例待决事件寄存器 204 包含一位，或者其它数据项，用于每一种由该事件检测器 188 登记的事件类型（例如，如下参考图 8 所述的事件）。这些事件可以构成在该处理器 30 内部产生的内部事件，或者在该处理器 30 外部产生的外部事件（例如，从处理器总线接收的管脚事件）。在说明的实施例，用于每一个线程的待决事件寄存器 204 不包含用于写回事件的一位，因为这样的事件不是线程特有的而且因此不在该待决事件寄存器中“排队”。为此，事件检测器 188 可以包含写回检测逻辑 205，其在检测到一个写回事件上声明一个写回信号。通过事件检测器 188 来设置在用于每一个线程的待决事件寄存器 204 内的位，其中该检测器 188 触发一个设置在该待决事件寄存器 204 内的适当位的锁存器。在一个示例实施例中，如将在下面描述的那样，在该待决事件寄存器 204 内与一个预定事件相关联的一组位提供了一个相关类型的事件正被待决的指示。

类似地，用于每一个线程的事件禁止寄存器 206 包含一位，或者其它数据结构，用于由该事件检测器 188 辨别的每一种事件类型，这个位被设置或者重置（即，清除）以相对于该特定线程把一个事件记录为一个中断事件。在一个事件禁止寄存器 206 内的相应位由一个控制寄存器写操作设置，该控制寄存器写操作使用一个特定微指令来修改在处理器 30 内的非重命名状态。在一个事件禁止寄存器 206 内的一位可以类似地使用一个控制寄存器写操作重置（或者清除）。

一个示例处理器还可以具有某些模式，在这些模式中可以设置在该事件禁止寄存器 206 中的位来禁止在相应的模式内选择事件。

用于保持在用于一个特定线程的每一个待决事件和事件禁止寄存器 204 和 206 内的一种特定事件类型的位被输出到一个与（AND）门 209，当寄存器 204 和 206 的内容指示相关的事件类型被待决而不是被禁止时，该与（AND）门 209 反过来输出一个用于每一种事件类型的事件已检测信号 211。例如，在一种事件类型不被禁止的地方，依据在待决事件寄存器 204 内一个事件的登记，通过声明用于相关事件类型的事件已检测信号 211，该事件马上被发信号通知被检测了。在另一方面，如果事件类型被事件禁止寄存器 206 的内容禁止，则事件发生将被记录在待决事件寄存器 204 内，但是仅仅如果在该事件禁止寄存器 206 内的适当位被清除了同时该事件仍然在该寄存器 204 内被记录为待决的话，才声明该事件已检测信号 211。因此，一个事件可以被记录在待决事件寄存器 204 内，但是用于相关事件发生的事件已检测信号 211 仅仅可以在以后的某一时间、当用于特定线程的事件禁止被删除了时，被发信号通知。

用于每一种用于每一个线程的事件类型的事件已检测信号 211 被送到事件处理逻辑（事件优先化和选择逻辑）和时钟控制逻辑，如将在下面进一步描述那样。

一旦事件处理已经完成时，用于一个特定事件的一个事件处理器负责清除在用于一个特定线程的待决事件寄存器 204 内的适当位。在一个替换实施例中，该待决事件寄存器可以由硬件清除。

在一个多线程处理器环境内的事件发生和事件处理

在多线程处理器 30 内的事件可以从若干个源检测到以及发信号通知。例如，处理器 30 的有序前端可以发信号通知一个事件，而且执行

单元 70 可以同样地发信号通知一个事件。事件可以包含中断和例外。中断是在处理器 30 外部产生、并且可以经由一个公共总线（未显示）从一个设备发起到处理器 30 的事件。中断可以导致控制流被导向到一个微码事件处理器 67。除了别的以外，例外可以泛泛地分类为错误、陷阱和帮助。例外是一般在处理器 30 内产生的事件。

事件被直接传递到在重新排序缓存 162 内的事件检测器 188，响应于此，事件检测器 188 执行多个有关为其产生事件或者相对于其产生事件的线程的操作。在一个高层处来看，响应于一个事件的检测，事件检测器 188 暂停用于该线程的微指令收回、写适当的错误信息到表格 180 里、声明清空信号 170、激活一个事件处理器 67 来处理该事件、确定一个重新启动地址、然后重新启动微指令的取出。事件可以以一个中断请求（或者中断扇区）的形式被直接传递到事件检测器 188，或者通过记录在重新排序表格 180 内、用于正被收回的第一或者第二线程的一个指令的错误信息被传递到事件检测器 188。

清空信号（nuke signal）170 的声明具有清除多线程处理器 30 有序前端和无序后端的状态的效果。具体地说，响应于清空信号 170 的声明，许多功能单元，然而并非必须是全部，被清除了状态和微指令。存储器有序缓存 48 和总线接口单元 32 的某些部分没有被清除（例如，收回但不确认存储、总线探听、等）。清空信号 170 的声明进一步停止前端的指令取出以及停止把微指令排序到微码队列 68 中。虽然这个操作能够不受损害地在一个单线程多处理器、或者一个执行单个线程的多处理器内执行，在多个线程共存而且在一个多线程处理器 30 内处理的地方，当解决有关一个线程的事件发生时，不能忽略其它线程的存在。因此，本发明提出一种用于在一个多线程处理器内处理一个事件的方法和装置，当发生用于单个线程的一个事件时，该方法和装置考虑到在该多线程处理器内多个线程的处理和存在。

图 7A 是一个流程图，其依据本发明的示例实施例说明了一种在一个多线程处理器 30 内处理一个事件发生的方法 220。方法 220 开始于块 222，在那儿由事件检测器 188 检测用于第一线程的一个第一事件。图 8 是多个可以由在块 222 处的事件检测器 188 检测的示例事件 224 的一个图解表示。在图 8 中表示的事件已经依据对事件 224 响应的特征，泛泛地进行了分组。第一组事件包含 RESET 事件 226 和 MACHINE

CHECK 事件 228, 它们由事件检测器 188 以如下所述的方式发信号通知到在一个多线程处理器 30 内的多个线程, 一旦检测到这些事件就导致所有线程同时到达同一个事件处理器 67。第二组事件包含 FAULT 事件 230、ASSIST 事件 232、DOUBLE FAULT 事件 234、SHUTDOWN 事件 236 和 SMC (自我修改代码) 事件 238, 这些事件的每一个在发信号通知该事件的一个特定线程的微指令收回上被报告。具体地说, 事件检测器 188 将依据错误信息为其指示一个错误状态的一个微指令的收回而检测第二组中的一个事件。第二组中的一个事件的检测仅仅由事件检测器 188 发信号通知到产生相关事件的线程。

10 第三组事件包含一个 INIT (简短重置) 事件 240、一个 INTR (本地中断) 事件 242、一个 NMI (非屏蔽中断) 事件 244、一个 DATA BREAKPOINT 事件 246、一个 TRACE MESSAGE 事件 248 和一个 A20M (地址绕回) 事件 250。在具有一个接收中断或者接收陷阱流标记的一个微指令的收回上报告第三组事件。第三组中的事件检测仅仅由事件检测器 188 发信号通知到产生相关事件的线程。

15 第四组事件包含一个 SMI (系统管理中断) 事件 250、一个 STOP CLOCK 事件 252、和一个 PREQ (探测请求) 事件 254。第四组中的事件被发信号通知到在该多线程处理器 30 内现存的所有线程, 而且当多个线程的任何一个收回一个具有一个适当中断流标记的微指令时报告该事件。响应于第四组中的任意事件, 没有多个线程之间实现同步。

依据一个示例实施例, 第五组事件, 是一个多线程处理器结构特有的并且在所描述的实施例内实现以解决为一个多线程处理器环境所特有的问题。第五组事件包含一个 VIRTUAL NUKE 事件 260、一个 SYNCHRONIZATION 事件 262 和一个 SLEEP 事件 264。

25 VIRTUAL NUKE 事件 260 是一个当: (1) 在该多线程处理器 30 内的第一线程具有一个待决事件 (例如, 任何上面所述的事件被待决)、(2) 第二线程没有待决事件 (不同于事件 260)、和 (3) 具有一个共享资源流标记 184 或者一个同步流标记 186 的微指令被重新排序缓存 162 收回时, 相对于一个第二线程被登记的事件。VIRTUAL NUKE 事件 260 具有激活一个虚拟清空事件处理器的效果, 该虚拟清空事件处理器在具有流标记 184 或者 186 的收回微指令之后微指令处重新启动第二线程的执行。

当一个特定线程（例如，一个第一线程）被要求在多线程处理器 30 内修改一个共享状态或者资源时，由微码发信号通知该 SYNCHRONIZATION 事件 262。为此，该微码定序器 66 插入一个同步微指令到用于第一个线程的流中，而且为了避免一个死锁情况，用一个共享资源流标记 184 和一个同步流标记 186 一起标记该“同步微指令”。仅仅依据用于第一线程的同步微指令的收回，以及依据用于第二线程、具有一个同步流标记 186 与此相关的一个微指令的收回，检测（或者登记）SYNCHRONIZATION 事件 262。SYNCHRONIZATION 事件 262 具有激活一个同步事件处理器的效果，该同步事件处理器在保存在一个微码暂存寄存器中的一个指令指针处重新启动第一个线程的执行。在下面提供了有关一个 SYNCHRONIZATION 事件 262 的处理的更多细节。第二个线程执行虚拟的 NUKE 260。

SLEEP 事件 264 是一个导致一个相关线程从一个活动状态转换到一个不活动（或者睡眠）状态的事件。不活动线程则可以由一个适当的 BREAK 事件再次从不活动转换到活动状态。转换该线程回到活动状态的 BREAK 事件的特性取决于转换该线程到不活动状态的 SLEEP 事件 264。线程进入一个活动状态和退出一个活动状态在下面详细描述。

图 9 是一个显示了在重新排序缓存 162 内的重新排序表格 180 的示例内容的框图，其应当在下面进行描述为了说明在本发明的一个示例实施例内的事件和清除点（还称为“清空点”）检测。响应于从在该多线程处理器 30 内的一个内源或者从在处理器 30 外的一个外源传送到事件检测器 188 的一个事件 266，在块 222 处可以发生事件检测器 188 对上述任一事件的检测。这样一个事件 266 通信的一个示例可以是一个中断向量。做为选择，通过用于正被收回并且因此由收回指针 182 标识的一个特定线程（例如，线程 1）的一个微指令的错误信息 268，一个事件发生可以传送到事件检测器 188。应当注意：对于外部事件，每一个线程有一个（1）信号（例如，分别如信号 266 和 267）。对于内部事件，重新排序缓存 162 入口包含通过它的位置指示该错误属于哪个线程的线程（例如，T0 对比 T1）。依据一个事件的检测，事件检测器 188 在事件信息寄存器 202 内存储涉及该特定事件的事件信息（例如事件类型、事件源，等），并且此外在待决事件寄存器 204 中登记用于相关线程的一个待决事件。如上所述，在待决事件寄存器

204 内登记用于相关线程的一个待决事件包含在寄存器 204 内设置与该特定事件相关联的一个位。此外要注意到：如果事件没有被在事件禁止寄存器 206 内设置的、用于相关线程的一个位禁止，而且在一个微指令包含一个适当的流标记的情况下，通过声明一个适当的事件已
5 检测信号 211，可以有效地检测到该事件。

现在返回到图 7A 中显示的流程图，在块 222 处检测用于第一个线程的第一个事件之后，事件检测器 188 在块 270 处停止收回第一个线程并且声明一个“预清空”信号 169。声明预清空信号 169 以避免一种死锁情况，在该死锁情况中第一线程把第二线程排除在外控制指令流水线。具体地说，第二线程被拒绝访问指令流水线，而相对于第二线程、开始一个多线程清空操作所要求的状态不能发生。预清空信号 169
10 因此被传送到处理器的前端，具体地说传送到存储器执行单元 42，以填充构成其中检测到该事件的第一线程的微指令处理器流水线。处理器流水线的填充可以，仅仅举例来说，通过禁止由存储器执行单元 42 或者前端的其它组件执行的指令预取和自我修改代码（SMC）操作，来
15 执行。概括地说，通过停止第一线程中的微指令收回，及/或通过停止或者实质上减小传送有关第一线程的微指令到处理器流水线中，第二线程在该处理器中被给予了优先权而且一种死锁情况的可能性被降低了。

20 在判定框 272 处，做出关于第二线程在多线程处理器 30 内是否是活动的判定，并且据此由该重新排序缓存 162 收回第二线程。如果第二线程不是活动的，则方法 220 直接进行到块 274，在那儿执行称为一个“清空操作”的一个第一类型清除操作。可以参考由收回控制电路 168 保持的活动线程状态机 171 来执行关于一个特定线程是否活动的
25 判定。清空操作从声明清空信号 170 开始，如上所述，清空信号 170 的声明具有清除多线程处理器 30 有序前端和无序后端的状态的效果。由于仅仅第一个线程是活动的，所以不需要考虑该清空操作在任何其它可能在多线程处理器 30 内给出和现存的线程上的影响。

在另一方面，如果在判定框 272 处确定第二线程在多线程处理器
30 30 内是活动的，则方法 220 继续执行一系列构成检测用于第二线程的一个清除点（或者清空点）的操作，在该清除点处，可以执行一个对该第二线程具有减少的负影响的清空操作。在一个清除点检测之后执

行的清空操作是和块 274 处执行的操作相同的操作，并且因此清除了多线程处理器 30 的状态（即，用于第一和第二线程的状态）。状态清除包含微指令“排出”操作，其在该说明书的别处进行描述。在本应用中公开的一个示例实施例中，在检测一个清除点之后执行的清空操作在多线程处理器 30 内、不在为第一线程保持的状态和为第二线程保持的状态之间进行区分。在一个替换实施例中，在一个清除点检测之后执行的清空操作可能仅仅清除单个线程（即，检测到事件的线程）的状态，在一个多线程处理器 30 内发生一个有效程度的资源共享以及这样的共享资源被动态地划分和未划分以服务多个线程的地方，为单个线程清除状态尤其是复杂的。然而，这个替换实施例可能需要越来越复杂的硬件。

在判定框 272 处的肯定判定之后，在判定框 278 处做出关于第二线程是否已经遇到一个事件的一个进一步判定。这样一个事件可以包含除 VIRTUAL NUKE 事件 260 之外任何上述的事件。这个判定再次由事件检测器 188 响应于事件信号 266 或者用于第二线程的一个错误信息信号 269 做出。关于由第二线程遇到的任何事件的消息被保存在专用于第二线程的部分事件信息寄存器 202 中，而且在待决事件寄存器 204 内登记该事件发生。

如果第二线程已经独立地遇到一个事件，则该方法直接进行到块 280，在那儿执行一个多线程的清空操作以清除多线程处理器 30 的状态。做为选择，如果第二线程没有遇到一个事件，则在判定框 282 处做出第一个线程遇到的第一个事件是否要求一个共享状态、或者共享资源被修改以处理第一个事件的判定。举例来说，在第一个事件包含如上所述的一个 SYNCHRONIZATION 事件 262 的地方，这个指示第一个线程必需访问一个共享状态资源。可以通过收回一个用于第一线程、具有共享资源和同步流标记 184 和 186 与此相关的同步微指令来标识 SYNCHRONIZATION 事件 262。图 10 是一个类似于在图 9 中所示的框图，其显示了用于重新排序表格 180 的示例内容。显示的、分配给第一线程（例如，线程 0）的部分表格 180 包含一个由收回指针 182 引用的同步微指令。显示的同步微指令此外还具有与此相关的一个共享资源流标记 184 和一个同步流标记 186。说明的同步微指令的收回将由事件检测器 188 登记为一个 SYNCHRONIZATION 事件 262 的发生。

如果用于第一线程（例如，线程 0）的第一事件被确定不修改一个共享的状态或者资源，该方法 220 继续到判定框 284，在那儿做出关于第二线程（例如，线程 1）是否正收回一个具有一个与此相关的共享资源流标记 184 的微指令的判定。参见图 9，显示的、用于线程 1 的收回指针 182 引用一个具有一个共享资源流标记 184 和一个同步流标记 186 的微指令。在这个情况中，出现于判定框 284 处的状态将已经被满足了，而且方法 220 因此继续到块 280，在那儿执行多线程的清空操作。做为选择，如果用于第二线程（例如，线程 1）的收回指针 182 没有引用一个具有一个共享资源流标记 184 或者一个同步流标记 186 的微指令，方法继续到块 286，在那儿通过前进收回指针 182 来继续第二线程的收回。方法 220 从块 286 返回到判定框 278，在那儿再次做出第二线程是否已经遇到一个事件的一个判定。

如果在判定框 282 处，确定用于第一线程（例如，线程 0）的第一事件处理需要修改一个共享的状态或者资源，该方法 220 继续到判定框 288，在那儿做出第二线程（例如，线程 1）是否正收回一个具有一个与此相关的同步流标记 186 的微指令的判定。如果是这样的话，则在块 280 处执行多线程清空操作。如果不是的话，继续在块 286 收回用于第二线程的微指令直到遇到用于第二线程的一个事件或者用于第二线程的收回指针 182 索引到一个具有与此相关的一个同步流标记 186 的微指令为止。

在块 280 处清空操作开始之后，在块 290 处，在微码形式实现并且从该微码定序器 66 中定序的一个适当事件处理器 67 继续处理相关的事件。

Virtual Nuke 事件

如上所述，VIRTUAL NUKE 事件 260 以一种稍微不同于其它事件的方式进行处理。为此，图 7B 是一个流程图，说明一种依据一个示例实施例，检测和处理一个 VIRTUAL NUKE 事件 260 的方法 291。方法 291 假定当前没有用于第二线程的事件被待决（即，记录在一个用于第二线程的寄存器中）。

方法 291 开始于块 292，在那儿由事件检测器 188 检测用于第一线程的一个第一事件。这样一个事件能够是上述参考图 8 讨论的任何事件。

在块 293 处，事件检测器 188 停止收回第一个线程。在块 294 处，事件检测器 188 检测收回一个具有一个共享资源流标记 184 或者一个同步流标记的微指令。在块 295 处，从微码定序器 66 中激活一个“虚拟清空 (virtual nuke)”处理器。在块 296 处，该“虚拟清空”事件处理器在上面块 294 处收回的微指令之后的一个微指令处重新启动第二线程的执行。方法 291 然后在块 297 结束。

Nuke 操作

图 11A 是一个流程图，说明了一种依据一个示例实施例，在一个支持至少第一和第二线程的多线程处理器内执行一个清除（或者清空）操作的方法 300。方法 300 在块 302 处、用由事件检测器 188 响应于一个事件的发生和检测声明清空信号 170 开始。清空信号 170 被传送到在多线程处理器 30 内的多个功能单元，而且它的声明和解声明定义了这样一个窗口，在该窗口内执行为功能单元的状态和配置清除而准备的动作。图 12 是一个时序图，显示了与一个时钟信号 304 的上升沿同步发生的清空信号 170 的声明。

在块 303 处，估算活动线程状态机。

在块 306 处，指示在其上发生事件的微指令是否被收回、用于第一和第二线程的序号和最近的微指令信号，被传送到分配和释放列表管理逻辑 122 和 TBIT，其中 TBIT 是在轨迹转移预测单元 (TBPU)（其反过来是 TDE 60 的一部分）中的一个结构，用于跟踪在处理器 30 有序前端内的宏指令和微指令指针信息。TBIT 使用这个信息来锁存涉及该事件的信息（例如，微指令和宏指令指令指针）。

在块 308 处，事件检测器 188 构造和传送用于第一和第二线程的每一个的一个事件向量到微码定序器 66。每一个事件向量在其中包含信息标识（1）当定位清空点（或者清除点）时正被收回的物理重新排序缓冲区单元（即，当标识清空点时每一个收回指针 182 的值）、（2）标识在微码定序器 66 内的一个位置的一个事件处理器标识符，其中定位构成一个事件处理器 67 来处理检测到的事件的微码、（3）标识第一或者第二线程的一个线程标识符、以及（4）一个确定该事件处理器 67 相对于为其它线程激活的事件处理器的优先级的线程优先级位。

在块 310 处，分配和释放列表管理逻辑 122 使用在块 306 处传送的序号把一个阴影寄存器别名表格（阴影 RAT）前进到一个检测到清

空点的点处，而且在块 312 处，从阴影寄存器别名表格中恢复主要寄存器别名表格 120 的状态。

在块 314 处，分配和释放列表管理逻辑 122 从释放列表管理器 134 中恢复寄存器号码（或者“弹子”），并且分配该恢复的寄存器号码到垃圾堆数组 132，从该垃圾堆数组 132 中可以再次分配该寄存器号码。当所有的适当寄存器号码已经从释放列表管理器 134 中恢复时，分配和释放列表管理逻辑 122 此外还声明一个“已恢复”信号（没有显示）。清空信号 170 保持在一个声明状态直到从分配和释放列表管理逻辑 122 中接收到这个“已恢复”信号为止。

在块 316 处，用于第一和第二线程的所有“老的”存储器（即，已经收回但是还没有更新的存储器）使用存储确认逻辑（没有显示）从该存储器有序缓存中排出。

在块 320 处，如图 12 所示，事件检测器 188 然后在时钟信号 304 的一个上升沿上解声明该清空信号 170。注意到：清空信号 170 被保持在一个声明状态中至少三个时钟信号 304 时钟周期。然而，如果来自分配和释放列表管理逻辑 122 的“已恢复”信号没有在接着清空信号 170 声明之后的头两个时钟信号 304 时钟周期内被声明，事件检测器 188 将扩展清空信号 170 的声明超过说明的三个时钟周期。在一个实施例中，清空信号 170 可以被保持足够长（例如，三个时钟周期）以允许完成上述讨论的块 303、306 和 308 的完成。清空信号 170 可以被要求保持额外的周期以允许完成块 310、312、314 和 316。为此，存储器有序缓存声明一个“存储器缓存排出”信号以扩展清空信号声明的扩展。

在块 322 处，在多线程处理器 30 内的微码定序器 66 及其它功能单元检查由活动线程状态机 171 保持的“活动位”以确定在该事件发生之后，第一和第二线程中每一个是否都在一个活动状态还是在一个不活动状态中。更具体地说，活动线程状态机 171 保持用于在多线程处理器 30 内现存的每一个线程的一个相应位指示，其指示相关的线程是在一个活动状态还是在不活动（睡眠）状态中。由事件检测器 188 检测以及事件检测器 188 响应于此声明清空信号 170 的事件，可以包含一个 SLEEP 事件 264 或者一个 BREAK 事件，这些事件在活动和不活动状态之间转换第一或者第二线程。如在图 12 中的 324 处，在声明清

空信号 170 期间估算活动线程状态机 171, 而且依据清空信号 170 的解声明, 该“活动位”的状态因此被认为是有效的。

在判定框 326 处, 检查活动线程状态机 171 的活动位的每一个功能单元做出第一和第二线程是否活动的一个判定。如果基于活动位的状态, 确定两个线程都是活动的, 则方法 300 继续到块 328, 在那儿每一个功能单元都被配置为支援和服务第一和第二活动线程。举例来说, 通过激活一个第二指针、或者第二组指针来对在不同功能单元内提供的存储和缓存性能进行逻辑上的划分, 这些第二指针局限于在一个存储阵列内一个特定组 (或者范围) 的入口。此外, 如果两个线程都是活动的, 则可以激活某些 MT 特有的支持。举例来说, 与微码定序器相关联的线程选择逻辑可以基于活动线程状态机 171 的输出, 以一种“乒乓”方式定序来自第一线程 (例如, T0)、来自第二线程 (例如, T1) 或者来自第一和第二线程 (例如, T0 和 T1) 的线程。此外, 可以基于活动线程状态机的位输出执行本地时钟选通。在一个文件地址寄存器实施例 15 中, 在一个处理器内任意数量的状态机, 可以基于活动线程状态机的输出修改它们的行为, 或者改变状态。在块 330 处, 微码定序器 66 然后继续定序用于第一和第二线程的微指令。

做为选择, 如果在判定框 326 处确定第一和第二线程的仅仅一个是活动的, 或者两个线程都是不活动的, 则在块 332 处, 每一个功能单元都被配置为支持和服务仅仅一个活动线程而且可以无效某些 MT 特有的支持。在没有线程是活动的地方, 作为一个缺省设置, 功能单元被配置为支持单个活动线程。在其中一个功能单元先前被配置 (例如, 逻辑划分) 为支持多个线程的情况中, 可以停用被使用来支持进一步线程的指针, 而且由剩余指针引用的、在一个数据阵列内的入口组可以被扩展以包含先前由停用的指针引用的入口。以这种方式, 可以理解: 可以使先前分配给其他线程的数据入口然后用于由单个活动的线程使用。通过当进一步线程是不活动时让更多的资源可用于该单一活动的线程, 相对于当其他线程还在多线程处理器 30 内支持时该单个剩余线程的性能, 可提高该剩余线程的性能。

在块 334 处, 微码定序器 66 忽略用于一个不活动线程、或者多个不活动线程的事件矢量, 而且仅仅对一个可能的活动线程定序微指令。在没有线程是活动的地方, 微码定序器 66 忽略用于所有线程的事

件矢量。

通过提供由该活动线程状态机 171 保持的活动位，提供了一个适当和集中的指示，依据于此各个功能单元可以被配置为在一个清空操作完成之后在多线程处理器 30 内支持一个正确数量的活动线程，其中
5 活动线程状态机 171 能够由各个功能单元依据清空信号 170 解声明(发信号通知一个清空操作的结束)进行检查。

图 11B 是一个显示了示例配置逻辑 329 的框图，其与一个功能单元 331 相关联，并且操作来配置功能单元 331 以在多线程处理器内支持一个或多个活动线程。功能单元 331 可以是上面所述功能单元、或者任何将由一个所属技术领域专业人员认为要被包含在一个处理器内的任何功能单元中的任何一个。显示的功能单元 331 具有由配置逻辑
10 329 配置的存储与逻辑组件。举例来说，存储组件可以包含多个寄存器。当多个线程是活动时(即，当一个处理器正以一个 MT 模式操作时)，这些寄存器中的每一个都可以被分配以存储用于这些线程特定一个的微指令或者数据。因此，如图 11B 所示的存储组件逻辑被划分为支持
15 第一和第二线程(例如，T0 和 T1)。当然，该存储组件能够被划分以支持任意数量的活动线程。

显示的逻辑组件包含 MT 逻辑，该逻辑特别地用于在该处理器内支持多线程操作(即，一个 MT)。

显示的配置逻辑 329 保持指针值 333，其被输出到功能单元 331 的存储组件。在一个示例实施例中，这些指针值 333 被使用来逻辑划分该存储组件。例如，能够为每一个活动线程产生一个单独的读和写指针值对。取决于活动线程的数目，由配置逻辑 329 确定用于每一个线程的指针值的上下界限。例如，如果其他线程变得不活动的话，由
20 一组用于一个特定线程的指针值指示的寄存器范围可以被增加以覆盖先前分配给另一个线程的寄存器。

配置逻辑 329 还包含 MT 支持启用指示 335，其被输出到功能单元的逻辑组件以启用或者停用功能逻辑 331 的 MT 支持逻辑。

由活动线程状态机 174 输出的活动位 327，提供输入到配置逻辑，并且由配置逻辑 329 使用来产生值 333 的适当点以及提供适当的 MT 支持启用输出。
30

一个事件处理器的排它访问

某些事件处理器（例如，用于处理分页和同步事件的那些）需要排它访问该多线程处理器 30 来使用共享资源以及修改共享状态。因此，微码定序器 66 实现了一个排它访问状态机 69，其在用于第一和第二线程的事件处理器需要这样的排它访问的地方，反过来排它访问这些事件处理器。仅仅当多于一个线程在多线程处理器 30 内活动时才可以引用该排它访问状态机 69。与具有排它访问的一个事件处理器相关联的一个流标记被插入到用于该线程的流中，来标记包含该事件处理器的排它代码的结尾。一旦为所有线程完成了排它访问，微码定序器 66 继续微指令的正常发布。

图 13 是一个流程图，其说明了一种依据示例实施例，在一个多线程处理器 30 内提供对一个事件处理器 67 的排它访问的方法 400。方法 400 在块 402 处、由微码定序器 66 从事件检测器 188 接收用于相应的第一和第二线程的第一和第二事件向量开始。如上所述，每一个第一和第二事件向量将标识一个相应的事件处理器 67。

在判定框 403 处，做出关于是否有多于一个（1）线程在活动的判定。这个判定由微码定序器参考活动线程状态机 171 做出。如果不是的话，方法 400 继续到块 434。如果是这样的话，方法 400 继续到判定框 404。

在判定框 404 处，微码定序器 66 做出关于第一或者第二事件处理器 67 是否需要排它访问一个共享资源、或者修改一个共享状态的判定。如果是这样的话，在块 406 处微码定序器 66 实现状态机 69，其按序提供对第一和第二事件处理器 67 每一个的排它访问。图 14 是依据示例实施例，描述了该排它访问状态机 69 的操作的一个状态图。显示的状态机 69 包含五个状态。在第一状态 408 中，用于第一和第二线程的微码都由微码定序器 66 发布。响应于需要一个排它访问事件处理器的一个事件发生一个清空操作 410 时，状态机 69 转换到第二状态 412，其中发布与用于第一线程的一个事件相关联的第一个事件处理器 67（即，微指令）。在所有构成第一个事件处理器 67 的微指令定序之后，以及在由这样的微指令指示的所有操作完成之后，微码定序器 66 然后在 414 发布一个停止微指令（例如具有一个相关的停止流标记的微指令），以把该状态机 69 从第二状态 412 转换到一个第三状态 416，在该状态中停止发布第一线程的微指令。在 418 处，在 414 发布的停

止微指令从重新排序缓存 162 收回,以由此把状态机 69 从第三状态 416 转换到第四状态 420, 其中微码定序器 66 发布与用于第二线程的一个事件相关联的第二事件处理器 67. 在所有构成第二事件处理器 67 的微指令定序之后, 以及在由这样的微指令指示的所有操作完成之后, 微码定序器 66 然后在 422 发布一个进一步的停止微指令, 以把该状态机 69 从第四状态转换到一个第五状态 424, 在该状态中停止发布第二事件处理器 67 的微指令。在 426 处, 在 422 发布的停止微指令从重新排序缓存 162 收回, 由此把状态机 69 从第五状态 424 转换回到第一状态 408。

10 在块 432 处, 假定两个线程都是活动的, 则再继续用于第一和第二线程的微指令的正常定序和发布。

做为选择, 如果判定框 404 确定第一或者第二事件处理器都不需要排它访问处理器 30 的共享资源或者状态, 则方法继续到块 434, 其中微码定序器 66 以一种非排它、交叉的方式定序构成第一和第二事件处理器 67 的微码。

15 活动线程状态机 (171)

图 15 是一个状态图 500, 其依据一个示例实施例说明了可以由活动线程状态机 171 占据的状态, 还依据一个示例实施例说明了可以导致活动线程状态机 171 在各个状态之间进行转换的转换事件。

20 显示的活动线程状态机 171 驻留在四个状态中的一个中, 这四个状态即单个线程 0 (ST0) 状态 502、单个线程 1 (ST1) 状态 504、多个线程 (MT) 状态 506、以及零线程 (ZT) 状态 508。活动线程状态机 171 为每一线程保持单个活动位, 当该位被设置时, 标识相关的线程正是活动的; 以及当被重置时, 指示相关的线程正是不活动或者在睡眠中。

25 在四个状态 502 - 508 之间的转换由事件对触发, 一个事件对的每一个事件都与第一或者第二线程有关。在状态图 500 中, 指出了多个有助于在状态之间进行转换的事件类型。具体地说, SLEEP 事件是一个导致线程变得不活动的事件。BREAK 事件是一个当为一个特定线程发生时, 导致该线程从一个不活动状态转换到一个活动状态的事件。一个特定事件是否有资格作为一个 BREAK 事件可以取决于导致该线程变得不活动的 SLEEP 事件。具体地说, 一旦不活动是一个特定 SLEEP 事

件的结果，则仅仅某些事件将导致一个线程变得活动。一个 NUXE 事件是任何当为特定线程发生时，导致执行一个如上所述的清空操作的事件。上面参考图 8 讨论的所有事件都可能包含清空事件。最后，还在状态图 500 内说明了相对于一个特定线程发生的一个“没有事件”，
5 它作为一个可以与相对于一个进一步的线程发生的事件组合存在以导致一个状态转换的条件。

在一个实施例中，如果为一个特定线程发信号通知一个 SLEEP 事件，且用于那个线程的 BREAK 事件是待决的，则马上服务于该 BREAK 事件(例如，线程不会进入睡眠并且被稍后唤醒来服务该 BREAK 事件)。
10 相反情况也可以是真的，因为可以为一个特定线程发信号通知一个 BREAK 事件，而且一个 SLEEP 事件是待决的，然后就服务于该 BREAK 事件。

依据由事件检测器 188 对清空信号 170 的声明，估算活动线程状态机 171，如在图 12 中的 324 处指示的那样。在解声明该清空信号 170
15 之后，基于由活动线程状态机保持的活动位配置在多线程处理器 30 内的所有功能单元。具体地说，检验器、重放和收回单元 (CRU) 160 传送基于该活动位产生的一个信号到所有受影响的功能单元，以向功能单元指示在该多线程处理器内有多少线程是现存的，以及这些线程中的哪一个是活动的。在声明清空信号 170 之后，一般在一个时钟信号
20 304 的时钟周期内完成功能单元的配置(例如划分或者取消划分)。

线程退出和入口

本发明提出了一种示例机制，借此在一个多线程处理器 30 内的线程可以进入和退出(例如，变得活动或者不活动)，其中这样的进入和退出以一种统一的次序发生而不管正在运行的线程数目，而且当在
25 多线程处理器 30 内没有更多的线程是活动的或者正在运行时，可以很好地停止到各个功能单元的时钟信号。

如上参考状态图 500 所述，响应于用于一个当前不活动线程的 BREAK 事件的检测，发生线程进入(或者激活)。取决于一个特定线程变得不活动的理由，定义用于该相关不活动线程的 BREAK 事件。响应
30 于用于一个当前活动线程的 SLEEP 事件，发生线程退出。SLEEP 事件的示例包含执行包含在一个活动线程内的一个停止 (HLT) 指令、一个 SHUTDOWN 或者一个 ERROR_SHUTDOWN 条件的检测、或者相对于该活动

线程的一个“等待 SIPI”（起动处理器间中断）条件。

图 16A 是一个流程图，依据本发明的示例实施例说明了一种在检测到用于一个活动线程的一个 SLEEP 事件时，退出该活动线程的方法 600。方法 600 在块 602 开始，在那儿保存用于该活动线程所有需要的状态，并且解除分配先前已经分配给用于该活动线程的微指令、在寄存器堆 124 内的所有寄存器入口。举例来说，对于在寄存器堆 124 内的仅仅 128 个寄存器入口，解分配先前分配给该活动线程的微指令的 28 个入口。被解分配的、用于该活动线程的寄存器内容被保存在一个“便笺本”中，其可以包含与在多线程处理器 30 内的一条控制寄存器总线相连的一个寄存器阵列或者随机存取存储器（RAM）。

可以由微码定序器 66 响应于用于活动线程的 STOPCLK、HALT（HLT）或者 SHUTDOWN 事件的检测、发布的一个解除分配微码序列，来执行在寄存器堆 124 内的寄存器入口的解分配。解分配微码序列进行操作以在释放列表管理器 134 内删除（或者无效）用于该寄存器堆入口的记录，并且在垃圾堆数组 132 内创建（或者确认）用于该寄存器堆入口的记录。换句话说，通过该解分配微码序列，用于该解分配寄存器堆入口的记录从释放列出管理器 134 传送到垃圾堆数组 132。

图 16B 是可以在块 602 执行的操作的一个示例实施例的图解表示。例如，在寄存器堆 124 内第一组寄存器内容的传送，这就是说先前分配给第一线程（例如，T0）的内容被显示为传输到该便笺本。在保存状态中可以执行的额外操作包含：存储用于一个退出线程的体系结构寄存器内容到便笺本；以及在退出这个第一线程时，存储分配给第一线程的微码暂存寄存器的内容到便笺本。在一个线程退出上空出的寄存器然后可以用于再分配到另一个线程（例如，T1）。

当重新进入一个特定线程时，应当理解：分配给这个线程的寄存器内容可以从该便笺本中恢复，如在图 16B 中的虚线指示的那样。

在块 604 处，一个用于该退出线程的线程特有“围栏微指令”被插入到用于该退出线程的微指令流中，来从该存储器有序缓存 48、各个高速缓存和处理器总线中排出任何剩余的、与该线程相关联的待决存储器访问。这个操作不会收回直到所有这些块完成为止。

当这些执行单元 20 相对快速地执行微指令时，利用响应于 SLEEP 事件检测声明的清空信号，清除所有添加到执行单元输入端的所有新

的微指令。如上所述，清空信号 170 被保持足够长的一段时间（例如，三个时钟周期）以便允许在声明该清空信号 170 之前进入该执行单元 70 的微指令从中排出。当这些微指令从执行单元 70 中排出时，它们被清除并且取消写回。

- 5 在块 606 处，通过由微码定序器 66 产生的一个微指令写回一个设置该展开寄存器状态的值，设置保持在事件检测器 188 内的展开寄存器 208，以指示该退出线程是在一个不活动（或者一个睡眠）状态中。

- 在块 608 处，通过让控制寄存器写入由微码定序器 66 发布的微指令，设置用于该退出线程的事件禁止寄存器 206 来禁止用于该退出线程的非中断事件。由控制寄存器微指令指示的、用于该退出线程的事件禁止寄存器的设置，取决于正被服务的睡眠事件类型。如上讨论所述，取决于触发到该不活动状态转换的 SLEEP 事件，相对于该不活动线程，仅仅某些事件有资格作为中断事件。具体参考用于该不活动线程的事件禁止寄存器 206，做出关于一个事件是否有资格作为用于一个
10 特定不活动线程的中断事件的判定。
15

 在块 612 处，使用一个特定微指令发信号通知用于该退出线程的睡眠事件，该微指令在该特定微指令的写回错误信息字段中放置一个睡眠事件编码。

- 图 17 是一个流程图，依据一个示例实施例说明了一种在检测到用于一个不活动线程的一个 BREAK 事件时，把该线程从一个不活动状态
20 转换到一个活动状态的方法 700。方法 700 在 702 处，为一个事件检测一个事件发生开始，其中相对于一个不活动线程，该事件可能有资格作为一个 BREAK 事件或者不能作为一个 BREAK 事件。在判定框 703 处，由一个事件检测逻辑 185 为相关事件做出确定该事件是否有资格
25 作为一个用于该不活动线程的一个 BREAK 事件的判定。为此，事件检测逻辑 185 检查在事件检测器 188 中的寄存器 200 内的事件禁止寄存器 206。如果相对于该不活动线程，相关的事件类型没有被指出是一个禁止的 BREAK 事件，则方法 700 继续到块 704，其中根据需要打开时钟，正常地发信号通知该事件（在另一个线程上等待一个可清空点），
30 并且如同用于任何事件一样激活处理器。事件处理器检查该线程睡眠状态，如果设置了睡眠状态，则继续在 706 处恢复微码状态。事件处理器 67 通过访问展开寄存器 208 来确认线程的不活动状态。

更具体地说，通过恢复所有保存的寄存器状态、禁止寄存器状态、和指令指针信息，事件处理器 67 着手恢复用于该进入线程的微码状态。

在块 706 处恢复微码状态之后，方法继续块 708，其中恢复用于该进入线程的体系结构状态。在块 710 处，通过从微码定序器 66 发布的一个适当的微指令，重置或者清除用于该进入线程的事件禁止寄存器 206。在块 712 处，事件处理器 67 继续服务 BREAK 事件。在这一点上，在多线程处理器 30 内执行构成该事件处理器 67 的微码，以响应于该事件发生执行一系列操作。在块 716 处，在处理器 30 内为该进入线程重新继续执行指令取出操作。方法 700 然后在块 718 处终止。

时钟控制逻辑

为了减少在该多线程处理器 30 内的功耗和热耗散，在一定条件下停止、或者暂停至少某些在处理器 30 内的时钟信号是符合需要的。图 18 是一个流程图，其依据一个示例实施例说明了一种在一个诸如上面所述的示例处理器 30 的多线程处理器内，停止、或者暂停选择的时钟信号的方法 800。为了本说明书起见，使用的对在处理器内的时钟信号的暂停或者停止应当拥有许多在处理器内暂停或者停止一个时钟信号、或者多个信号的技术。例如，能够暂停在处理器 30 内的一个锁相回路（PLL）、能够禁止沿着一个时钟中心的一个核心时钟信号分布、或者能够选通或者相反阻止经由该时钟中心分布一个时钟信号到在该处理器内的单独功能单元。一个实施例设想后面的情况，其中在一个功能单元接着一个功能单元的基础上，暂停、或者停止供应一个内部时钟信号到在处理器 30 内的功能单元。因此，当相对于其他功能单元被选通时，该内部时钟信号可以是提供给某些功能单元。这样一个方案在美国专利 5,655,127 中的单线程微处理器环境内描述了。

在一个实施例中，在图 18 中说明的方法 800 可以由集成在处理器 30 的总线接口单元 32 内的时钟控制逻辑 35 执行。在替换实施例中，时钟控制逻辑 35 当然可以位于远离处理器 30 的别处。图 19A 和 19B 是分别说明有关示例时钟控制逻辑 35 的更多细节的框图和示意图。

首先参见图 19A，显示的时钟控制逻辑 35 接收三个主要输入，即（1）经由活动线程状态机 174 输出的活动位 820（例如，T0_ACTIVE 和 T1_ACTIVE）；（2）由事件检测器 188 输出的事件检测信号 211；

以及(3)由总线接口单元 32 输出的一个探听控制信号 822,其在总线上检测一个可探听的访问并且声明信号 882。时钟控制逻辑 35 使用这些输入产生一个停止时钟信号 826,该信号 826 反过来抑制或者禁止在处理器 30 内某些功能单元的时钟脉冲。

5 图 19B 是一个说明示例组合逻辑的示意图,其利用输入 211、820 和 822 输出停止时钟信号 826。具体地说,事件检测器信号 211 提供输入到一个或(OR)门 822,其反过来提供输入到一个进一步的或(OR)门 824。活动位 820 和探听控制信号 822 还提供输入到 NOR 门 824,该 NOR 门或运算这些输入以输出停止时钟信号 826。

10 具体地参见图 18,方法 800 在判定框 802 开始,在判定框 802 处判定是否有任何线程(例如,一个第一和一个第二线程)在多线程处理器 30 内是活动的。这个判定由在图 19B 中活动位 820 到或(OR)门 824 的输出反映。虽然示例实施例说明相对于两个线程可以满足该判定,可以容易地理解:相对于在一个多线程处理器内支持的任意数量
15 线程做出这个判定。

在判定框 802 处的一个否定判定之后,方法 800 继续判定框 804,其中做出关于没有被禁止的任何事件是否为在该多线程处理器内支持的任何线程待决的判定。再次,在示例实施例中,这包含确定任何事件是否为第一或者第二线程待决。这个判定由图 19B 中显示的,事件
20 检测信号 211 到或(OR)门 822 的输入来表示。

在判定框 804 处的一个否定判定之后,在判定框 806 处做出任何探听(例如,总线探听、SNC 探听或者其他探听)是否正由处理器总线处理的一个进一步的判定。在本发明的示例实施例中,这个判定由输入该探听控制信号 822 到或(OR)门 824 实现。

25 在判定框 806 处的一个否定判定之后,方法 800 继续到块 808,其中到选定功能单元的内部时钟信号被停止或者抑制了。具体地说,到总线待决逻辑和总线访问逻辑的时钟信号没有被暂停或者停止,因为这允许该总线接口单元 32 检测在系统总线上起源的 BREAK 事件或者探听(例如,管脚事件)并且响应于这样的 BREAK 事件再启动该时钟
30 脉冲到功能单元。通过声明停止时钟信号 826 可以实现内部时钟信号到功能单元的抑制,其具有选通时钟信号到预定功能单元的效果。

在块 808 完成之后,方法 800 返回到判定框 802。在判定框 802、

804 和 806 处的判定之后，可以通过一个连续的基础进行循环。

在判定框 802、804 和 806 中任何一个的肯定判定之后，方法 800 转移到块 810，其中，如果已经选通了到某些功能单元的时钟信号的话，则这些内部时钟信号再次被激活。做为选择，如果时钟信号已经是活动的，则把这些时钟信号保持在一个活动状态。

在响应于一个中断事件，执行块 810 的地方（例如，在判定框 804 的一个肯定判定之后），在微处理器内的功能单元可以以上面所述的方式，基于活动线程的数目，在清空信号的声明处被主动地划分。例如，在一个具有两个或更多线程的多线程处理器 30 中，一些线程可以是不活动的，而在这样情况下将不会划分该功能单元以容纳不活动的线程。

当完成块 810 时，方法 800 再次返回到判定框 802，并且开始由判定框 802、804 和 806 表示的判定的另一次迭代。

因此，已经描述了用于在一个多线程处理器内管理一个时钟信号的方法和装置。虽然已经参考具体的示例实施例加以描述了，但是显然可以对这些实施例进行各种修改和改变而不背离本发明宽阔的范围和精神。因此，说明书和附图只作为一个示例说明，而不是起限制作用。

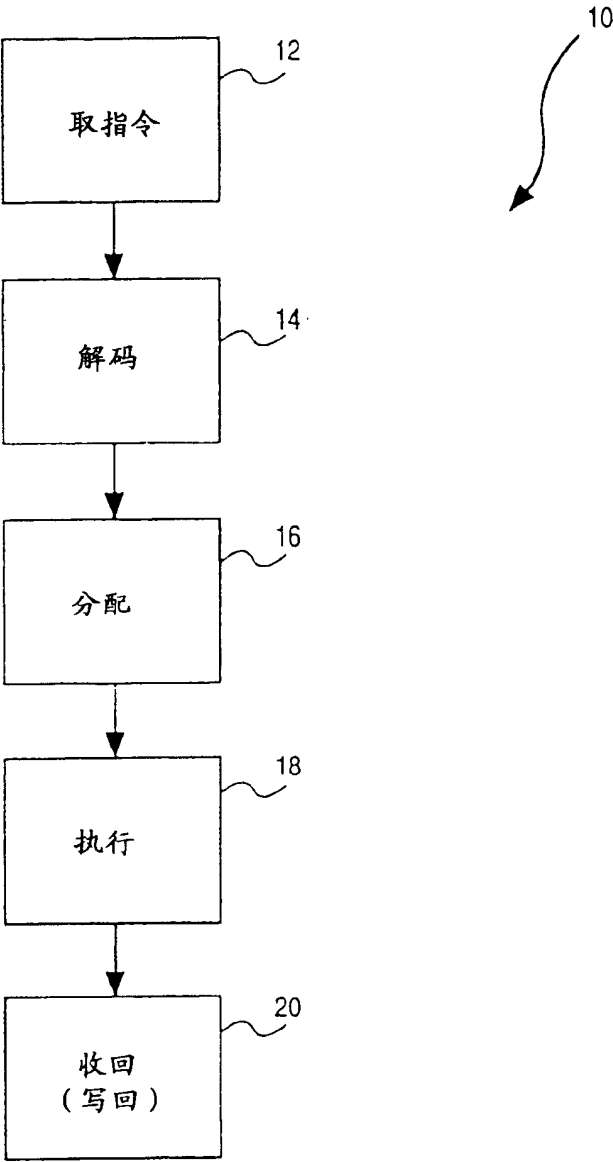


图 1

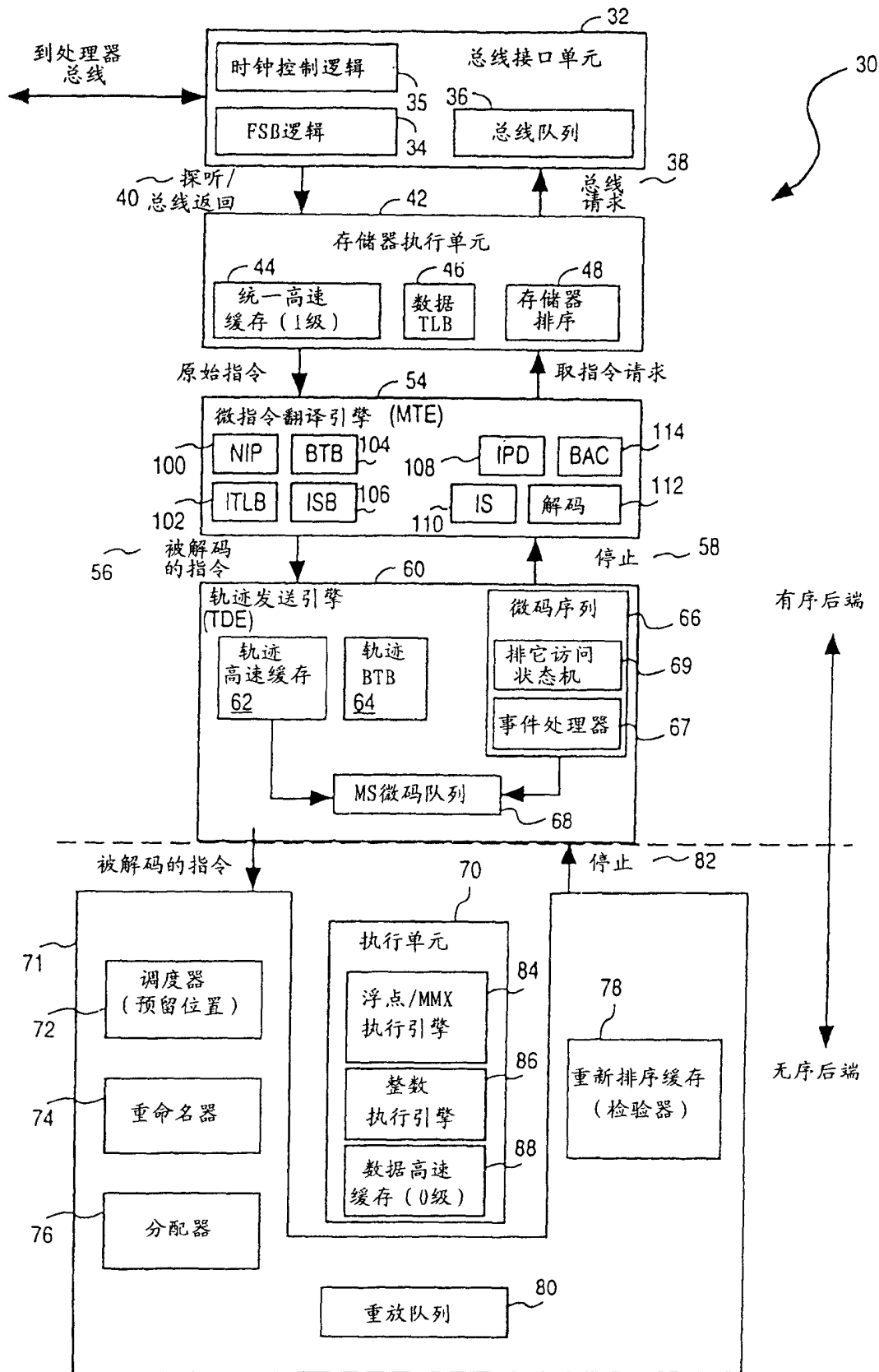


图 2

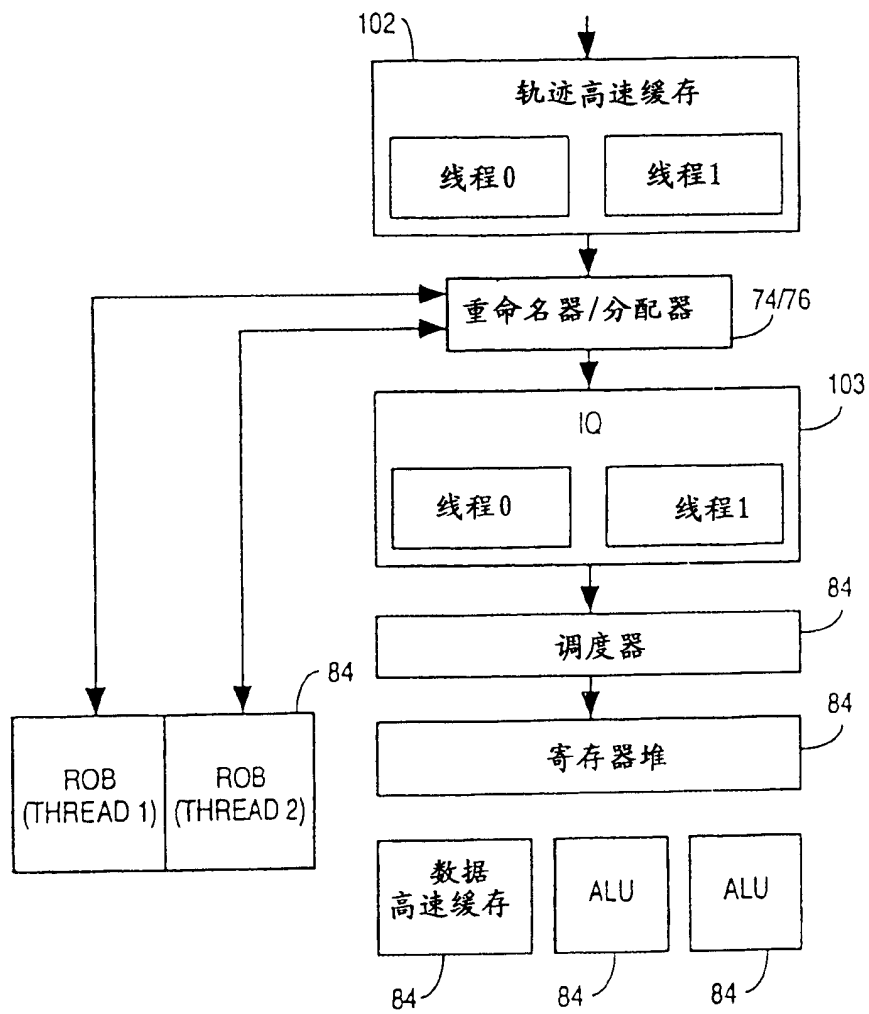
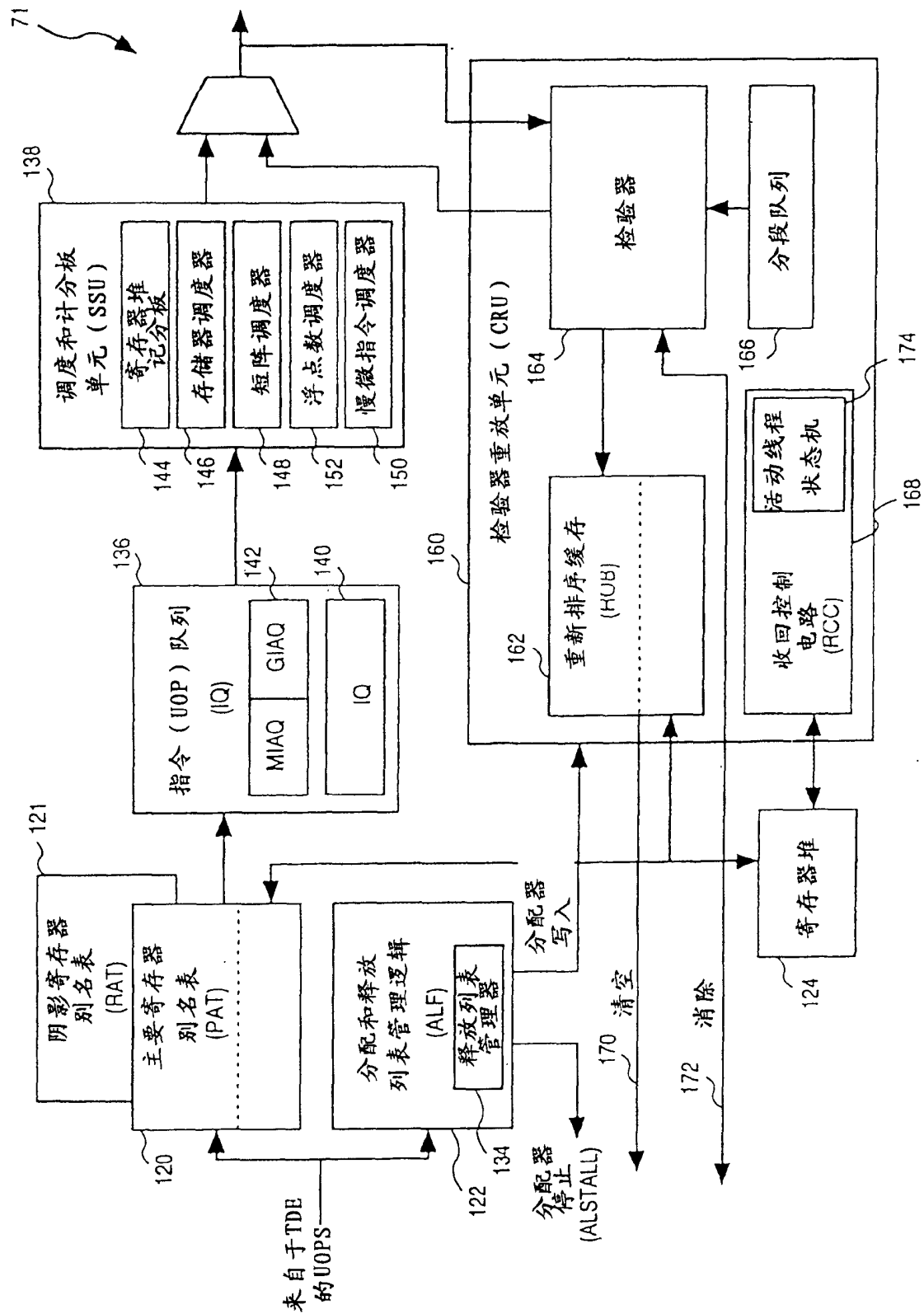


图 3

4

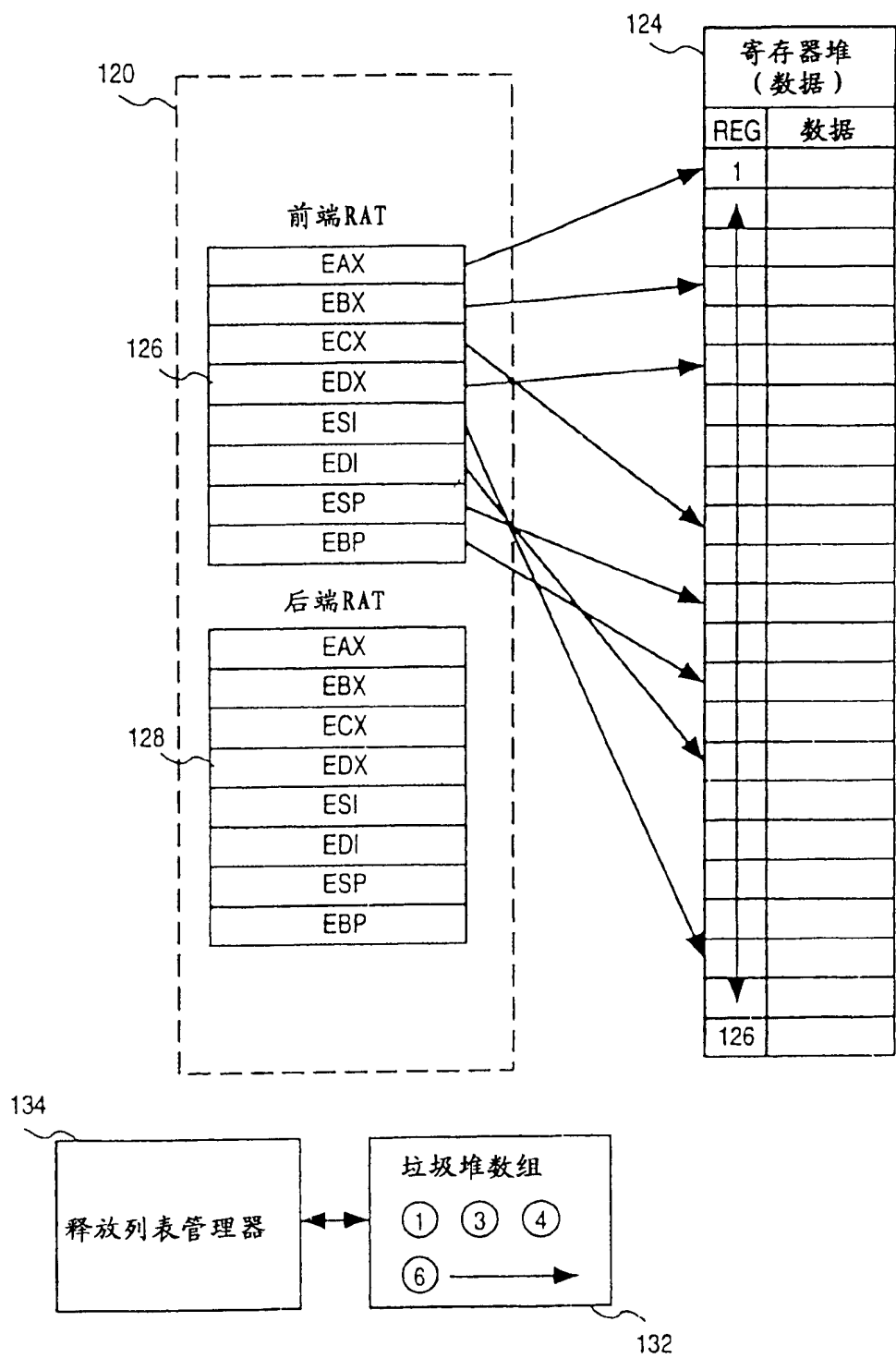
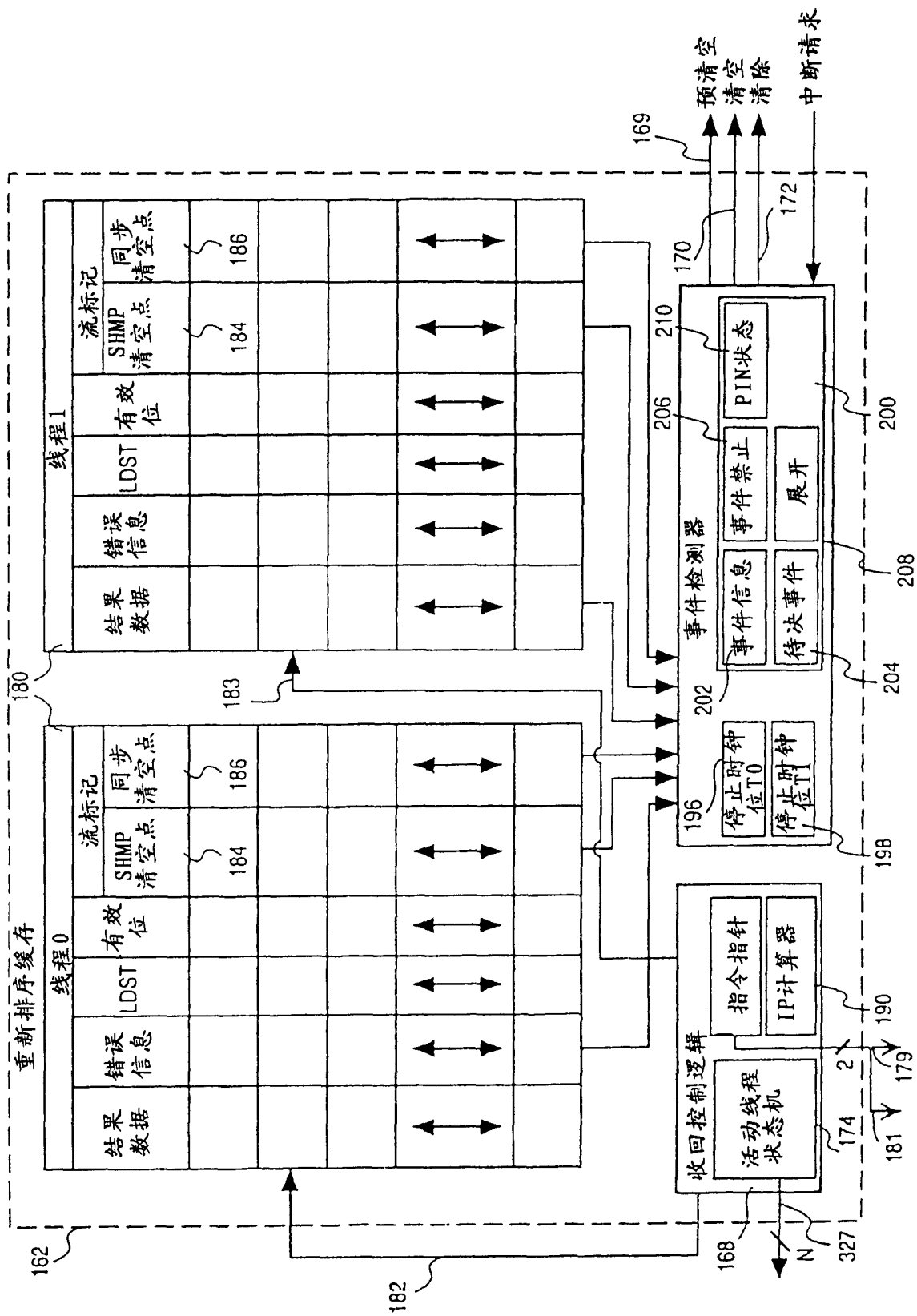



图 5



9

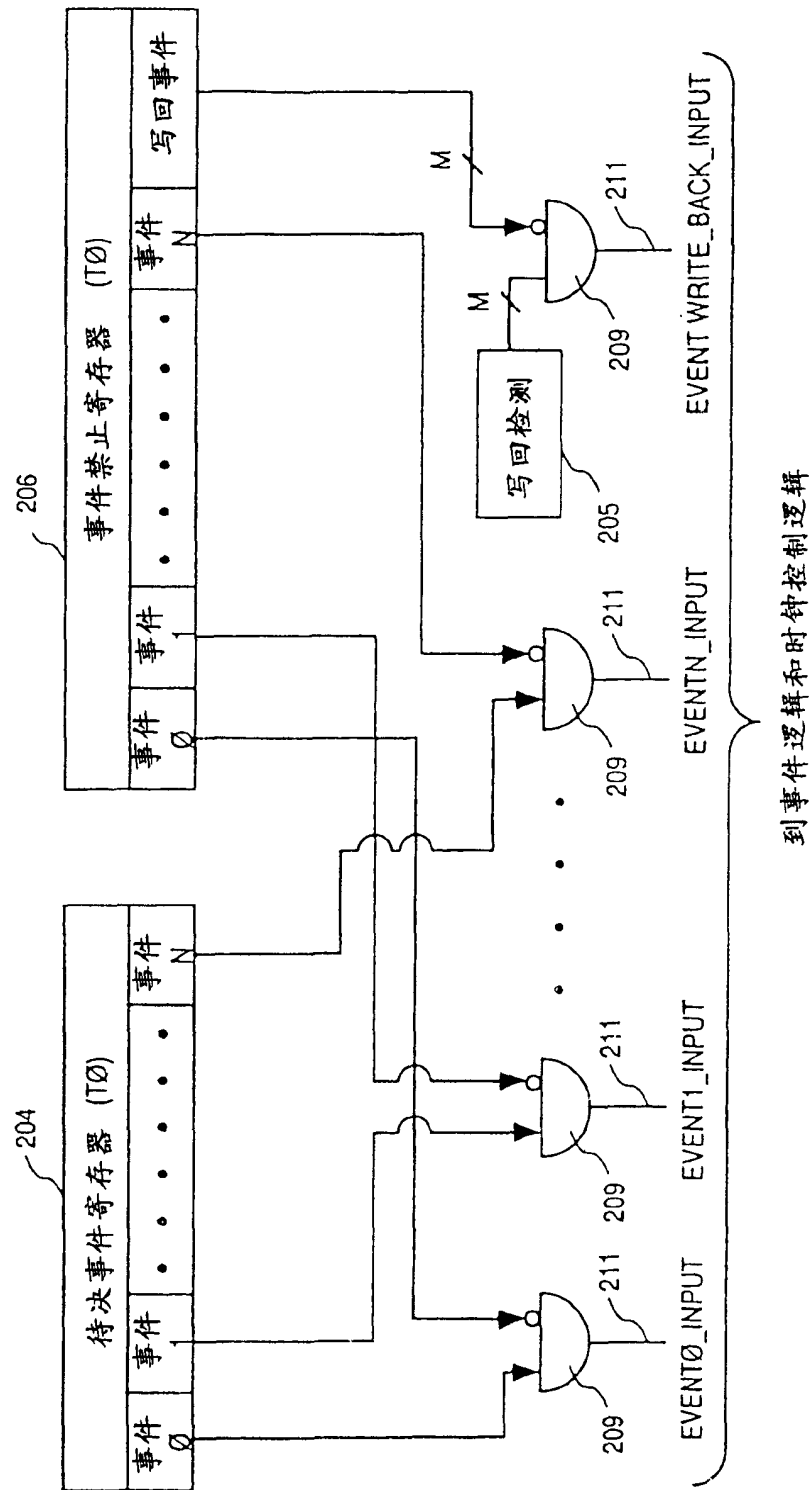


图 6B

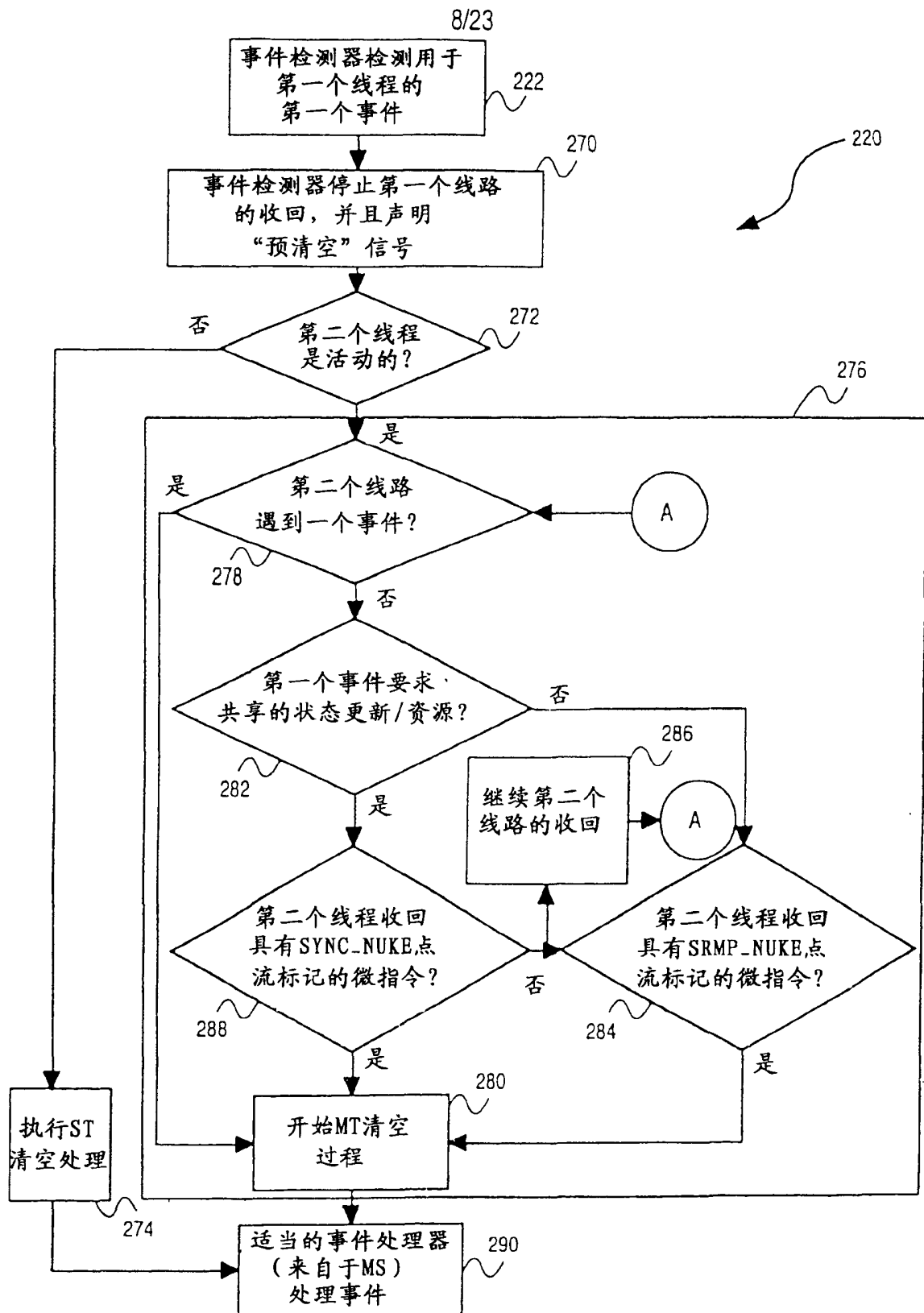


图 7A

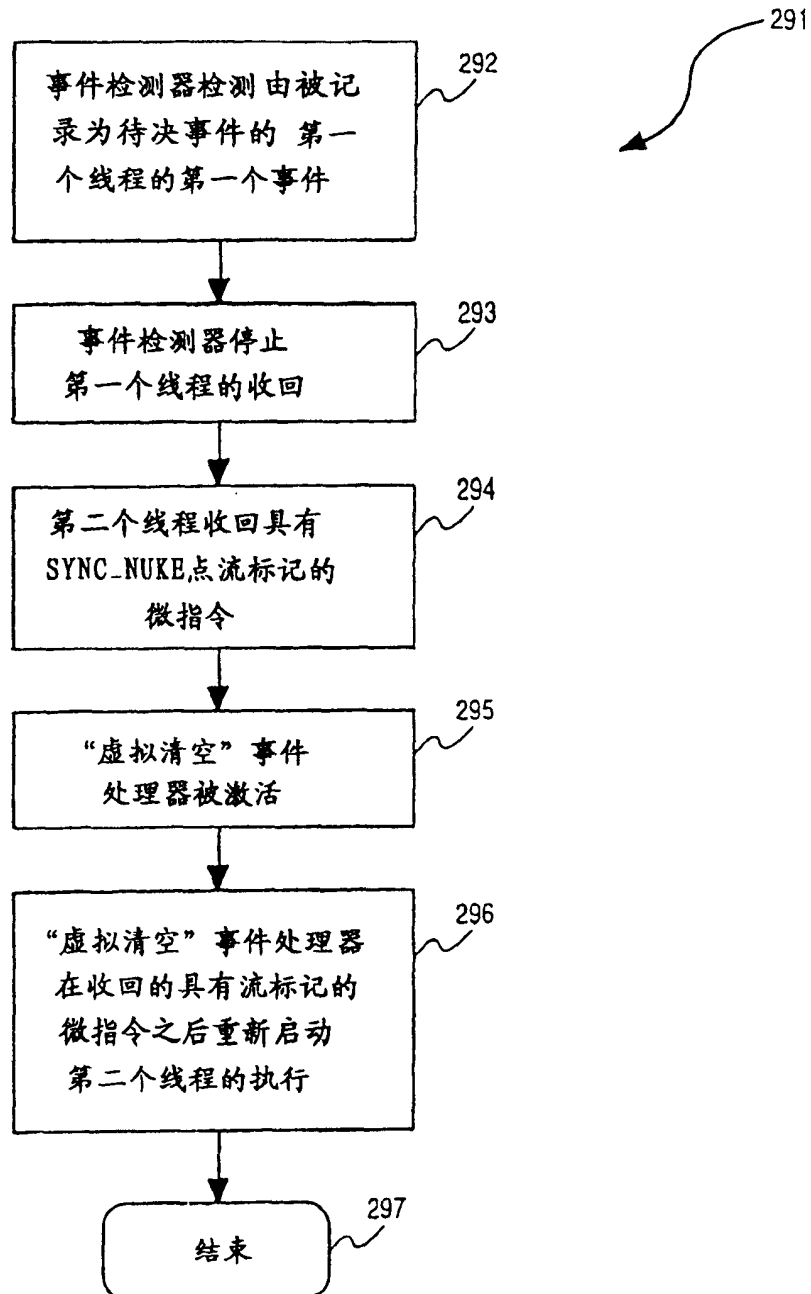


图 7B

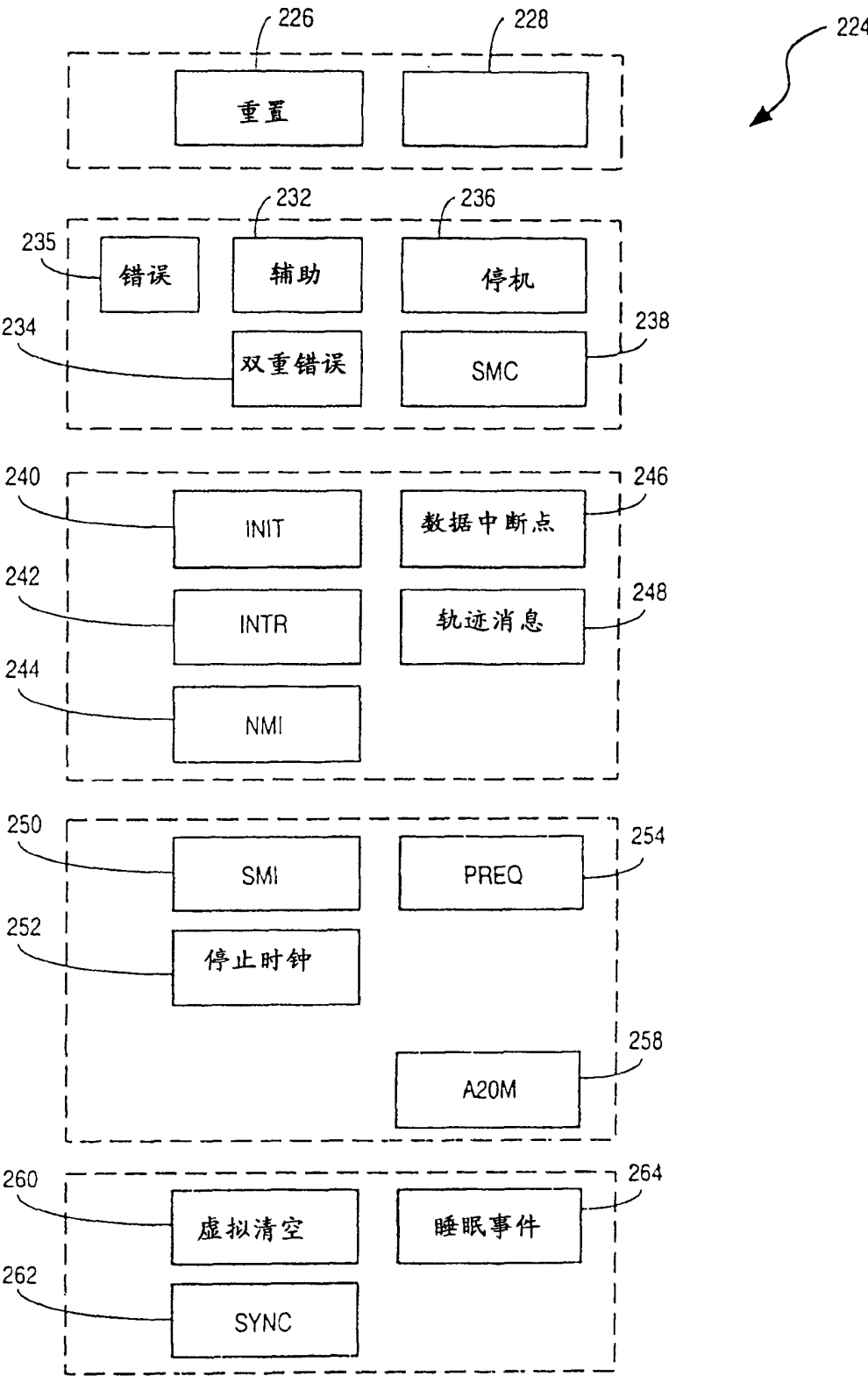


图 8

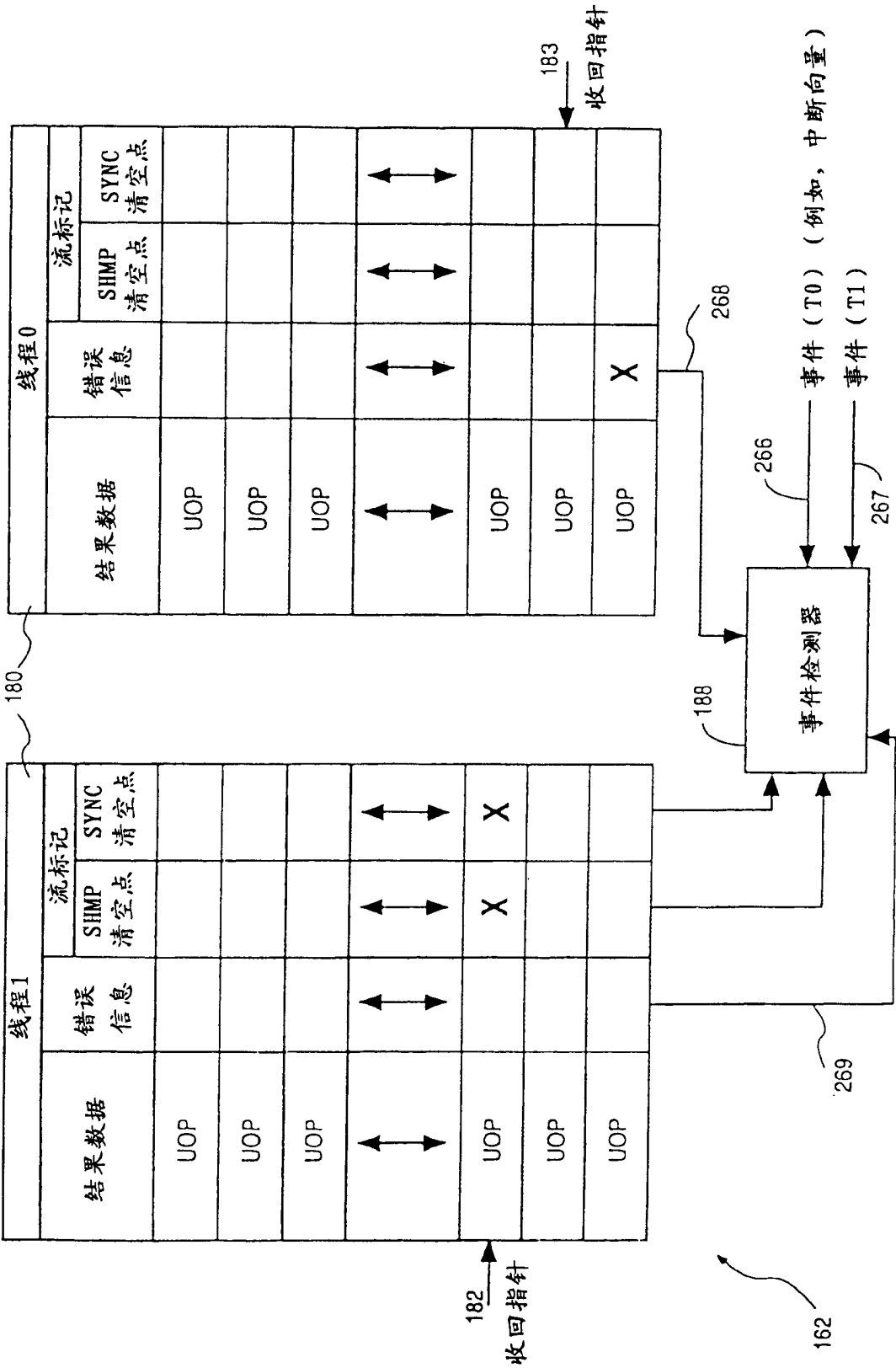


图 9

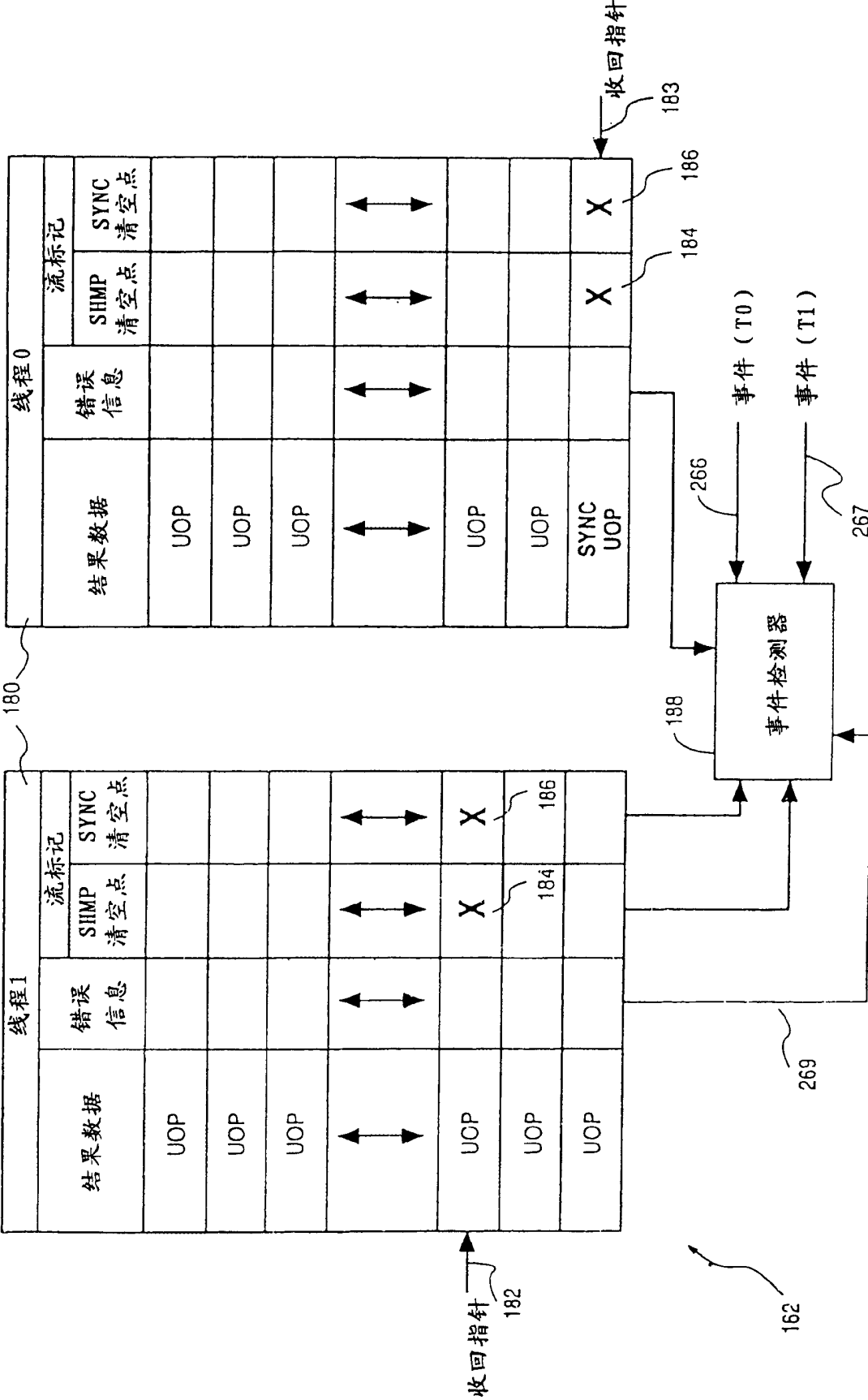


图 10

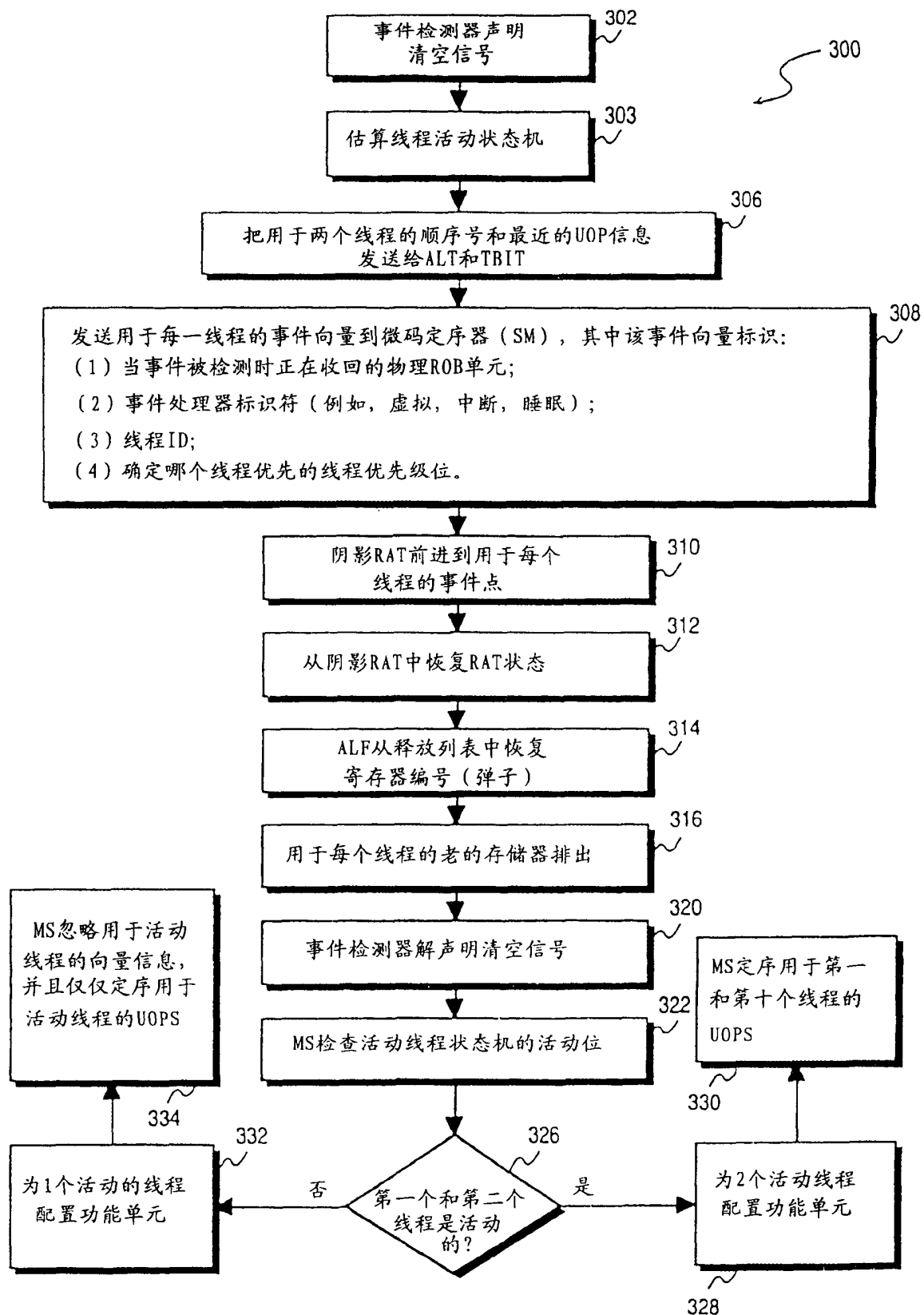


图 11A

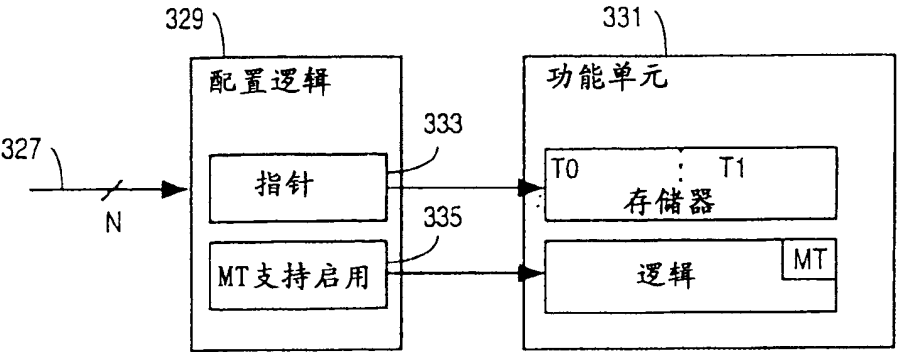


图 11B

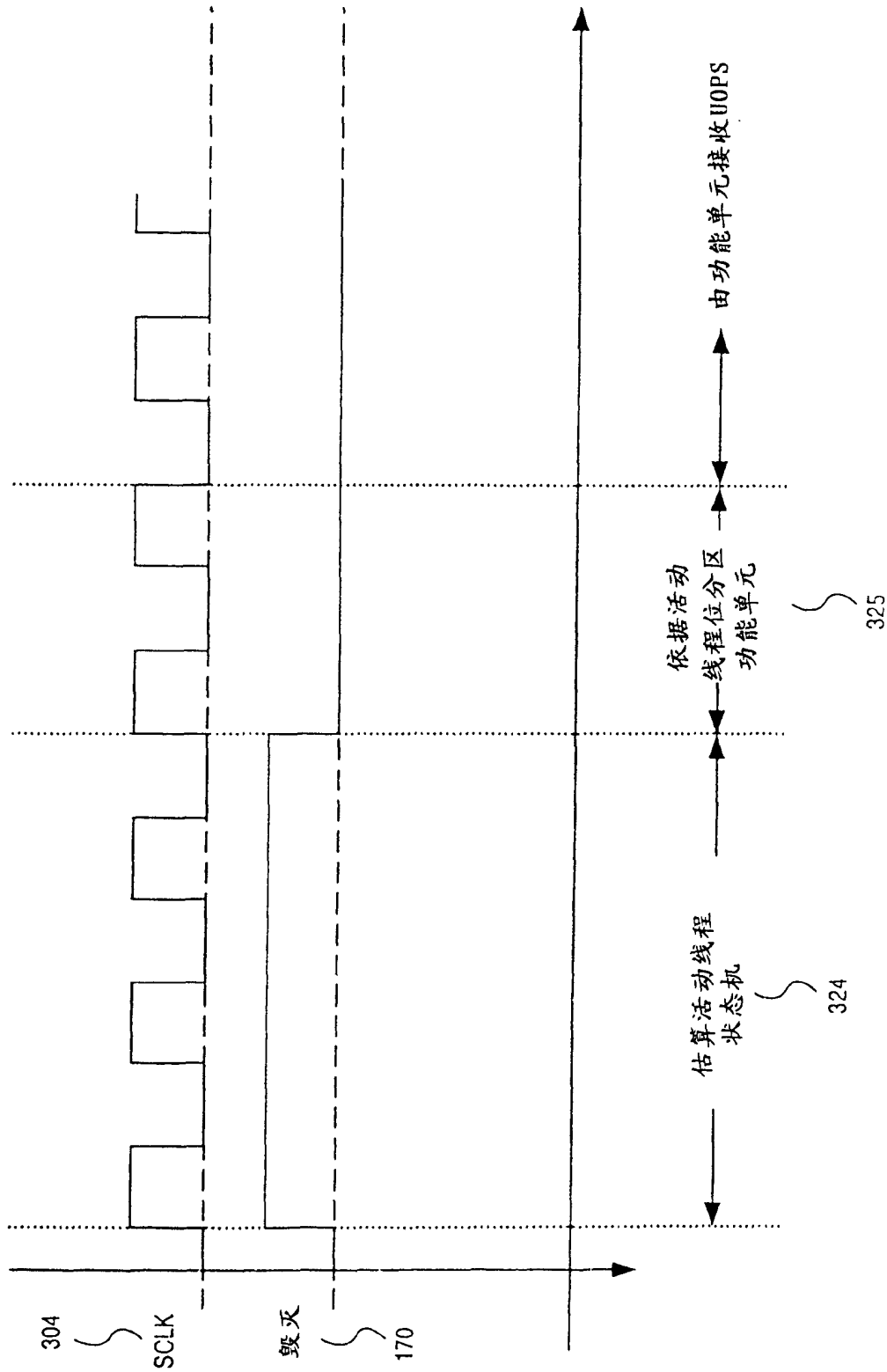


图 12

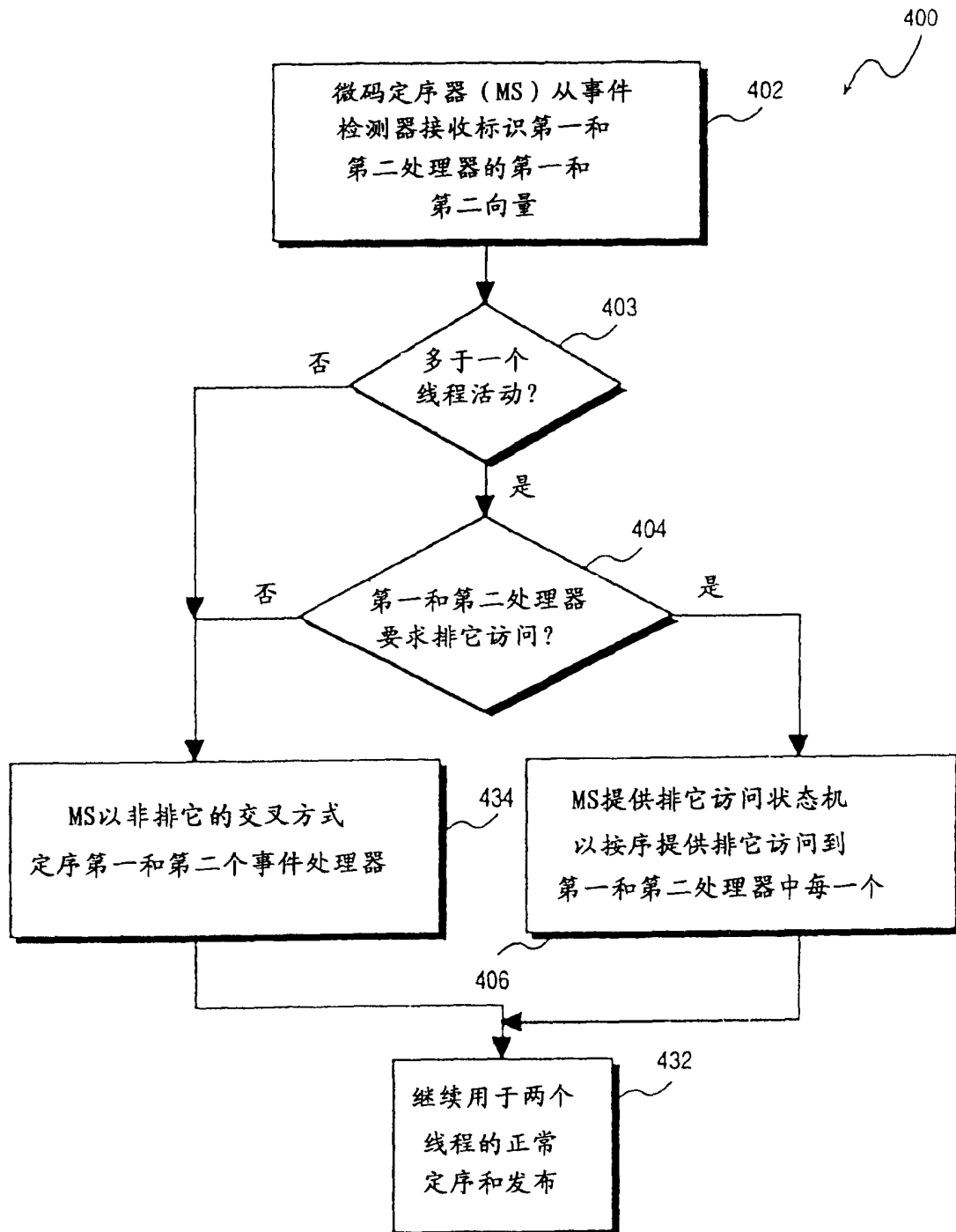


图 13

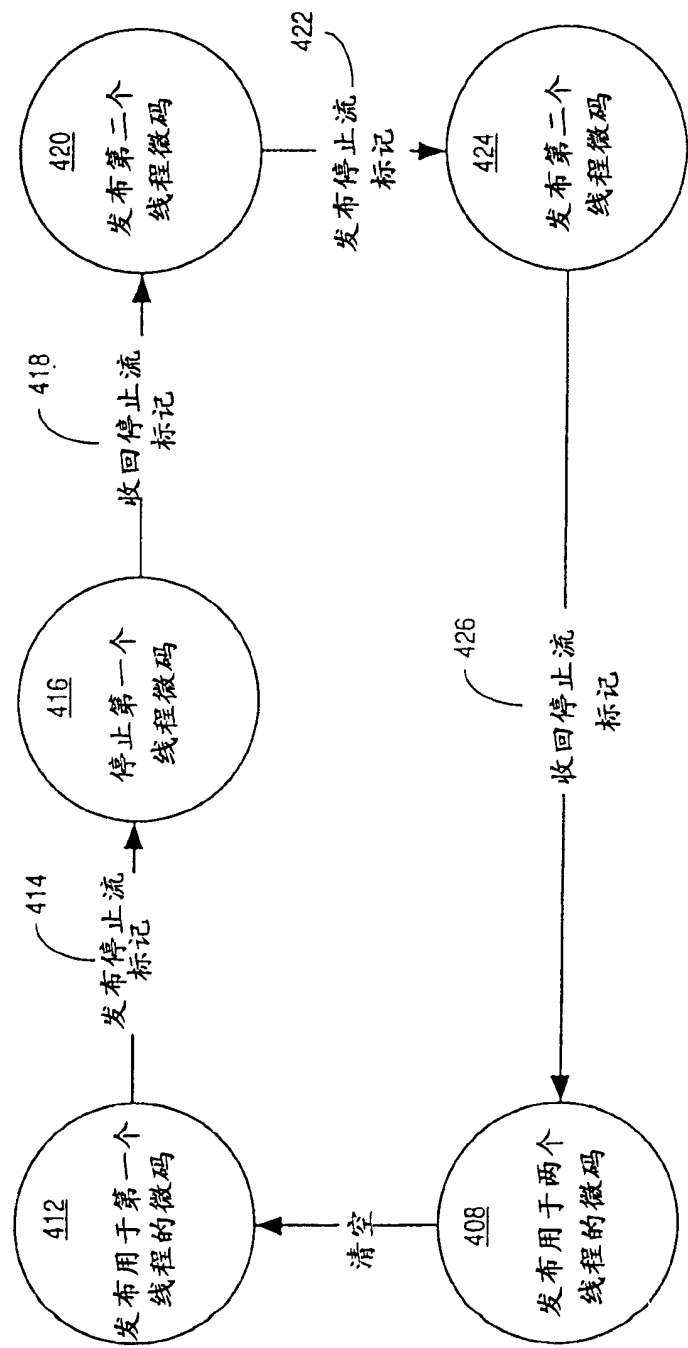


图 14

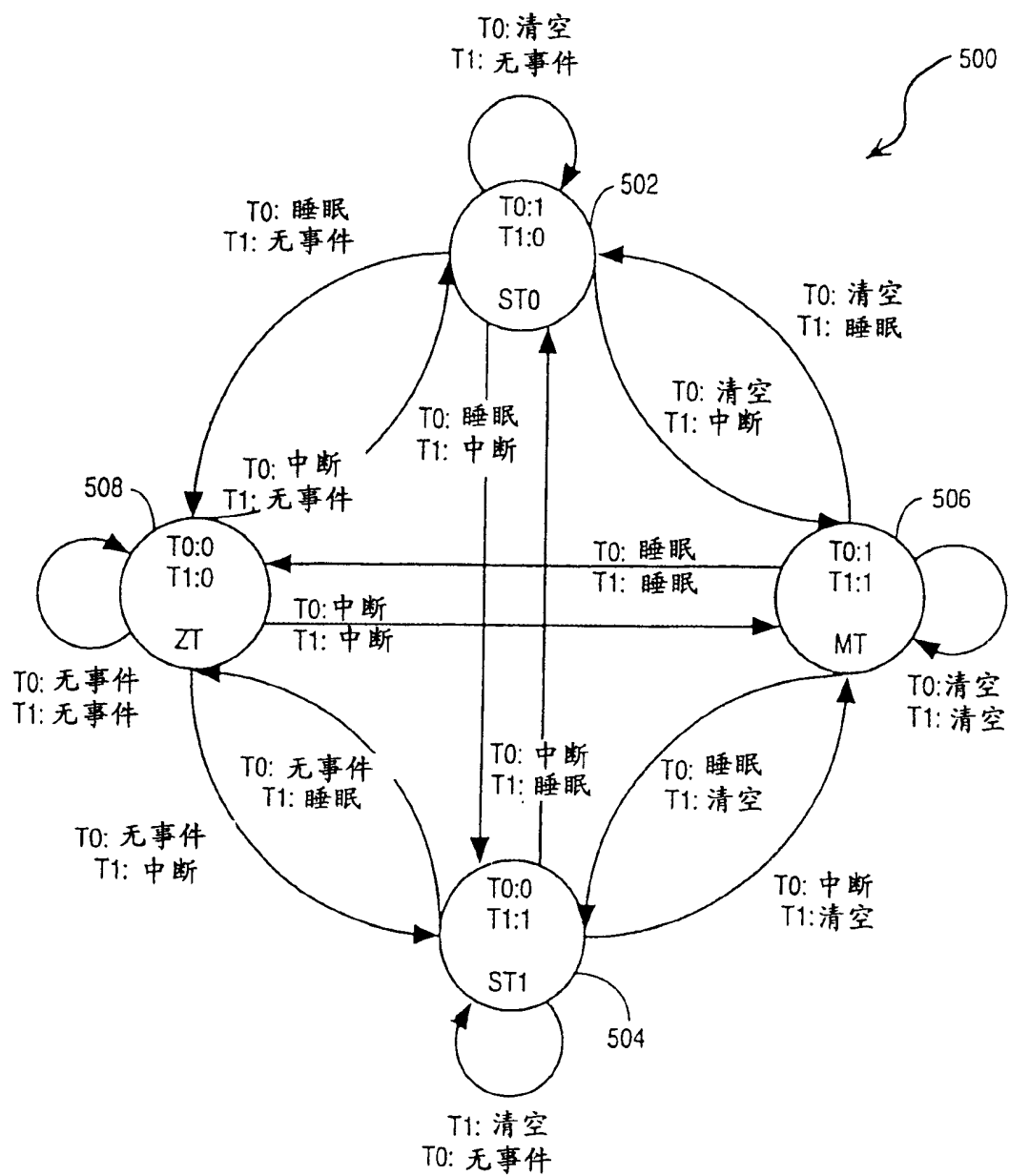


图 15

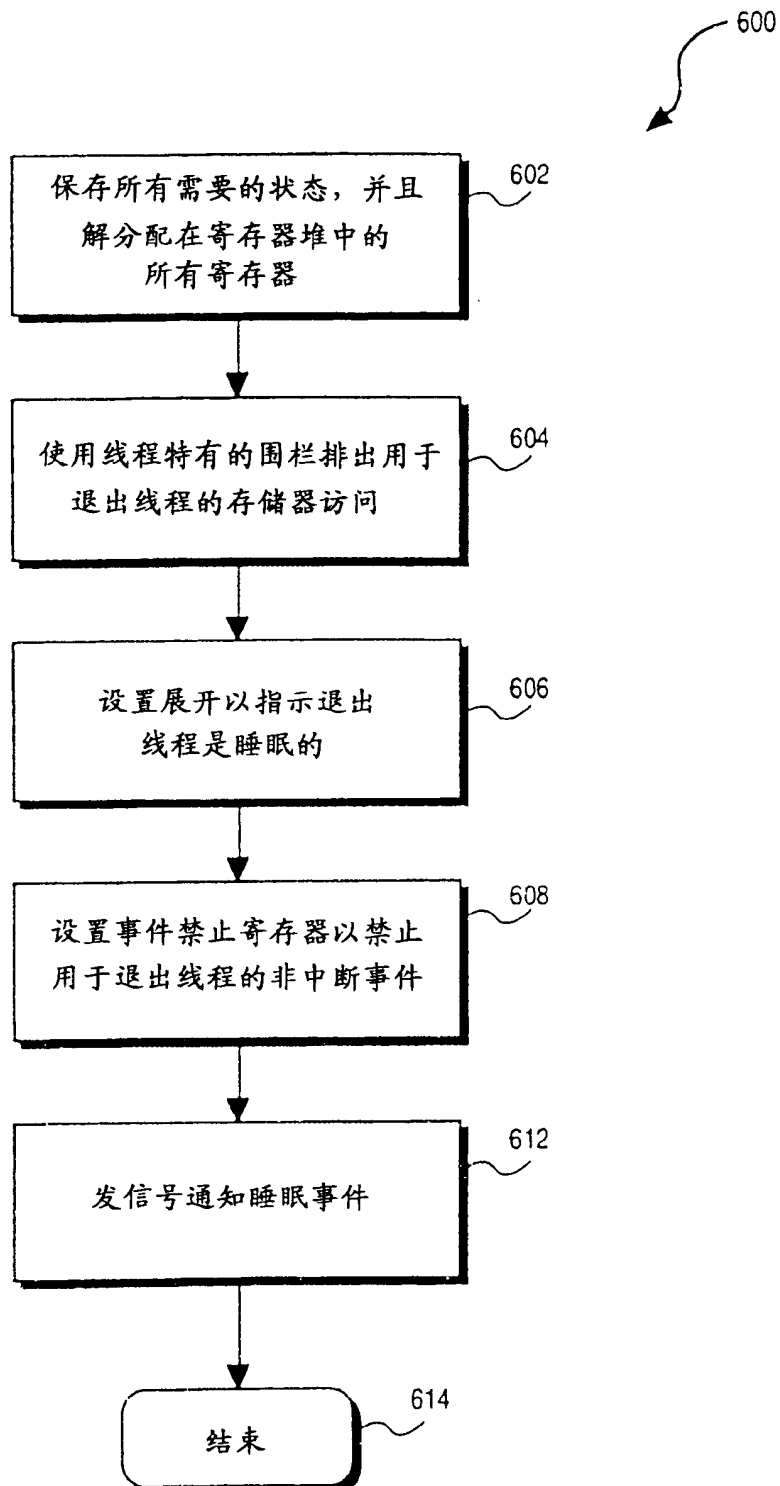


图 16A

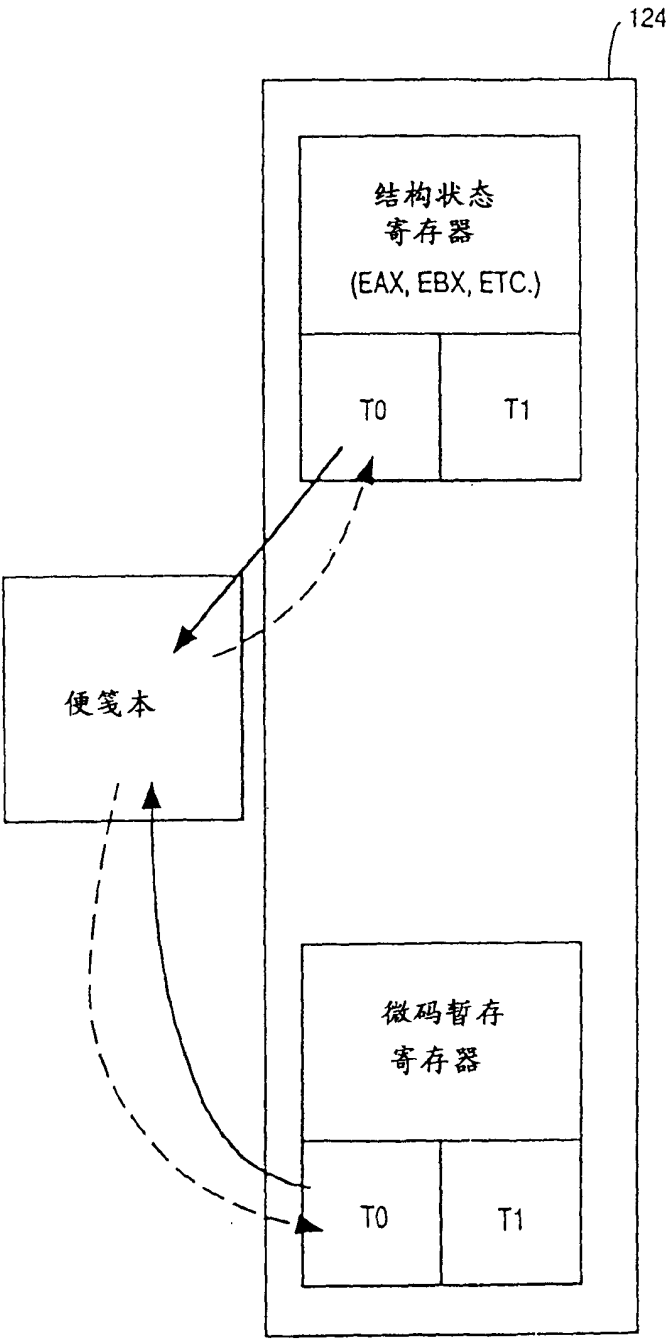


图 16B

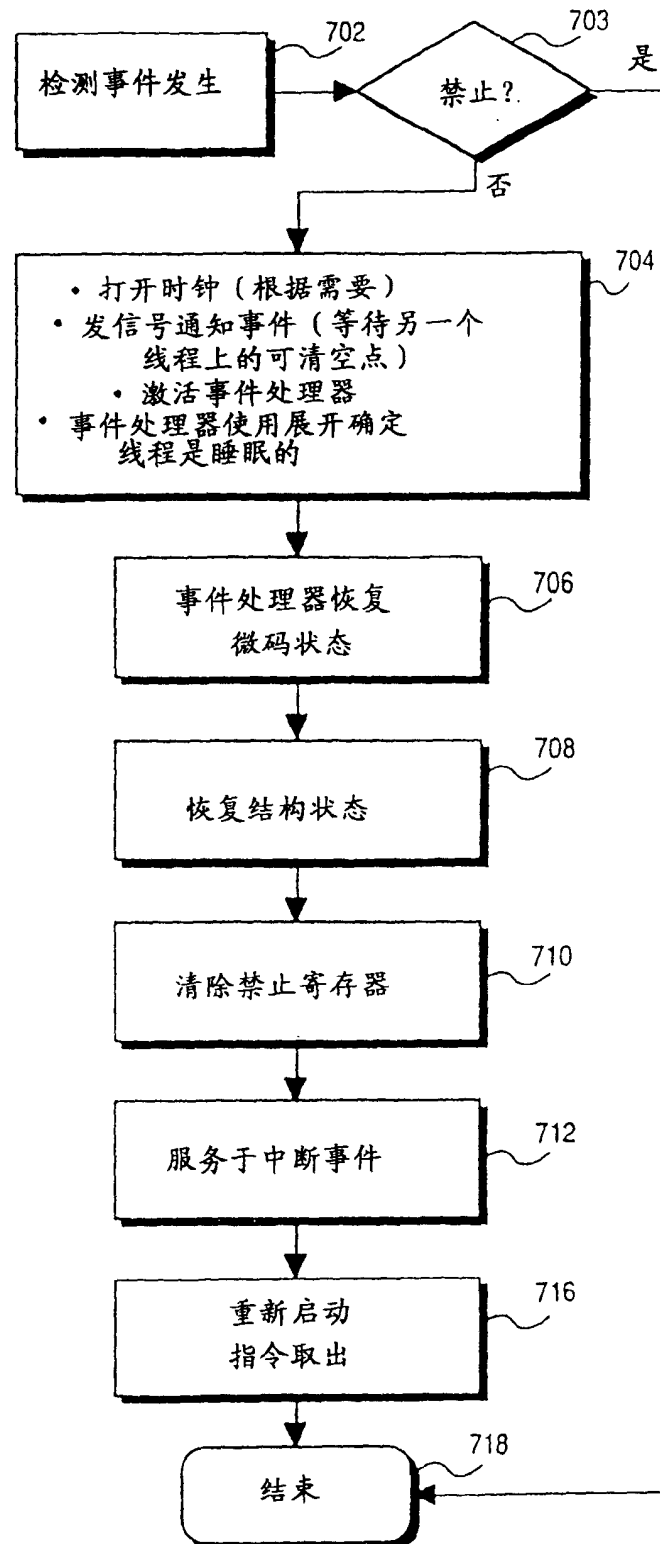


图 17

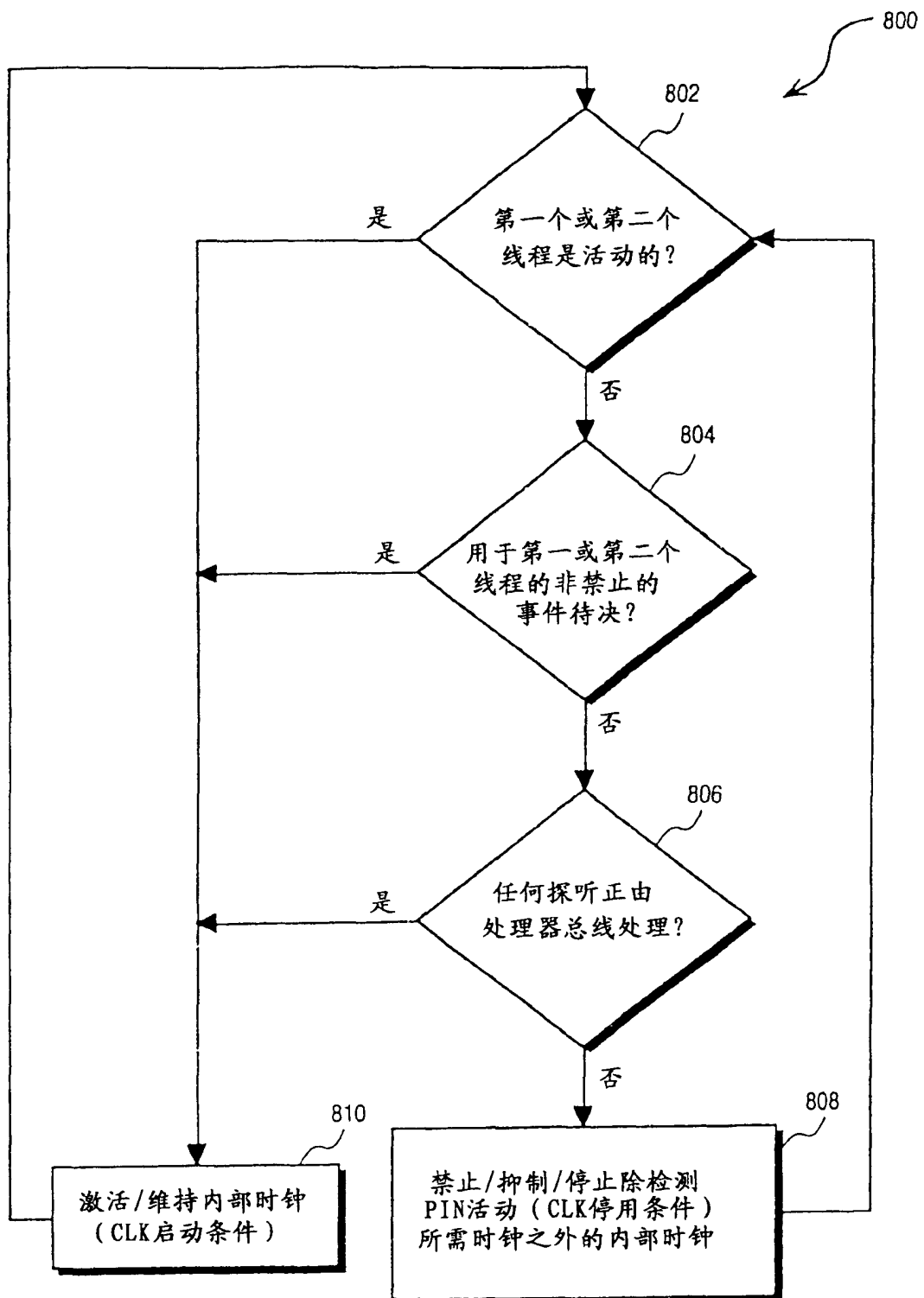


图 18

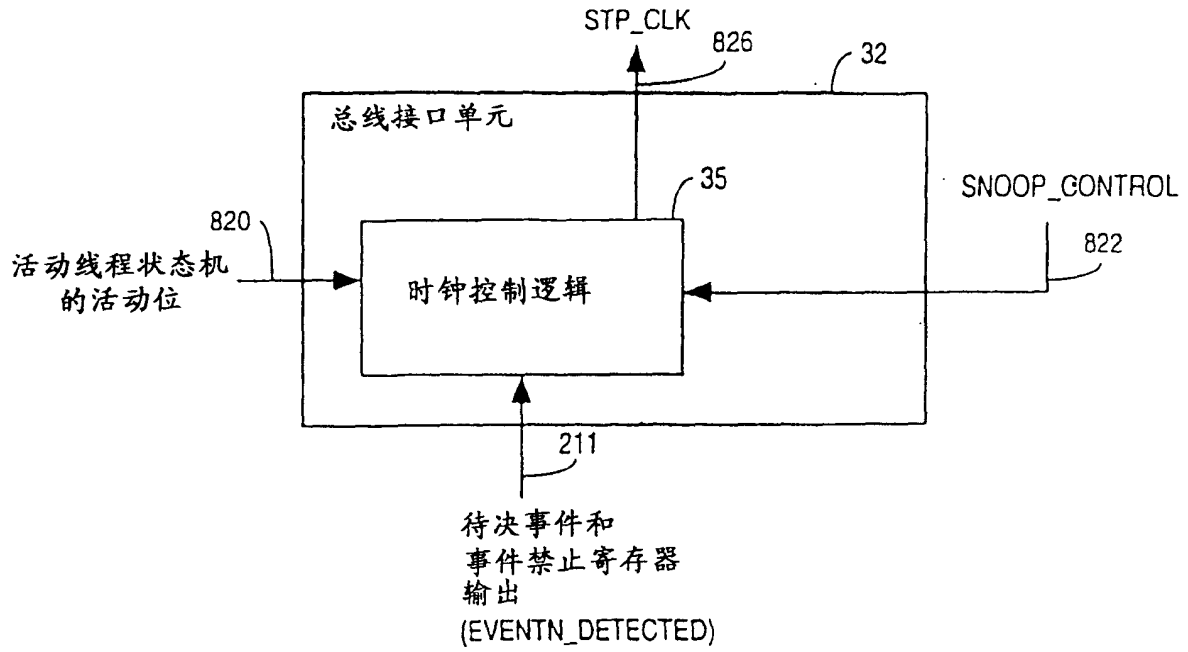


图 19A

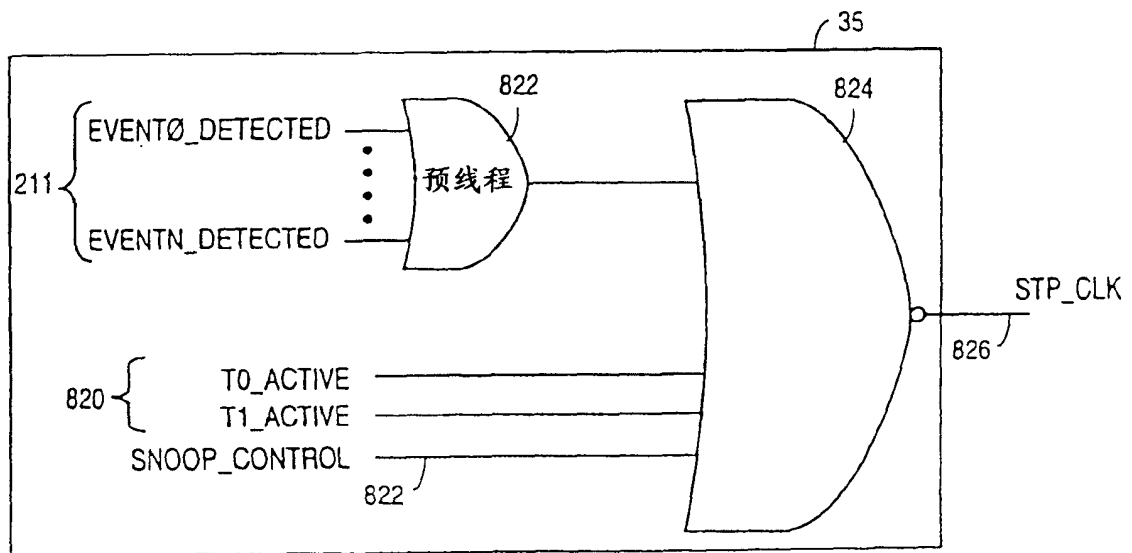


图 19B