[19] 中华人民共和国国家知识产权局

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

[12] 发明专利说明书

专利号 ZL 200610002572.9

[45] 授权公告日 2008 年 4 月 16 日

[11] 授权公告号 CN 100382035C

[22] 申请日 2000.10.12
[21] 申请号 200610002572.9
分案原申请号 00818830.0
[30] 优先权
[32] 1999.12.9 [33] US [31] 09/458570
[73] 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 D·波格斯
A·默钱特 R·科塔 R·苏
[56] 参考文献
WO98/43193A2 1998.10.1
WO99/21083A1 1999.4.29
US5854922A 1998.12.29
审查员 刘 慧

[74] 专利代理机构 中国专利代理(香港)有限公司
代理人 陈景峻

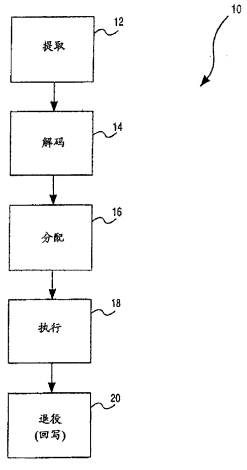
权利要求书 5 页 说明书 39 页 附图 23 页

[54] 发明名称

进入和退出多线程处理器中多线程的方法和装置

[57] 摘要

一种方法包括维护状态机以提供多位输出，多位输出的每一位都表示多线程处理器所执行的多线程中的有关线程的相应状态。 检测第一线程的状态，对其作出响应，多线程处理器中的功能单元根据状态机的多位输出进行配置。



1.一种方法，包括以下步骤：

维护状态机以表示多线程处理器中执行的多线程的有关线程的相应状态；

检测所述多线程处理器中第一线程的状态变化；以及

当所述第一线程的状态变化是从活动状态到非活动状态时，根据所述多线程处理器中第一线程的状态变化，改变功能单元的分区方案，以便为所述多线程处理器中的第二线程提供服务，但不为第一线程提供服务。

2.如权利要求1所述的方法，其特征在于所述状态表示所述有关线程是活动的或是非活动的。

3.如权利要求1所述的方法，其特征在于包括对所述功能单元进行分区，以便在所述第一线程的所述状态变化包括从非活动状态向活动状态转换时，为所述多线程处理器中的所述第一线程和第二线程提供服务。

4.如权利要求1所述的方法，其特征在于所述检测所述第一线程中的所述状态变化包括检测所述第一线程的事件的发生。

5.如权利要求4所述的方法，其特征在于包括：对所述第一线程的所述事件的发生作出响应而断言第一信号，并在所述第一信号的所述断言期间对所述状态机进行评估。

6.如权利要求5所述的方法，其特征在于：在对所述第一信号取消断言时，根据所述状态机的所述多位输出来配置所述多线程处理器中的所述功能单元。

7.如权利要求1所述的方法，其特征在于所述检测所述第一线程中的所述状态变化包括检测所述第一线程的睡眠事件的发生，其中，所述第一线程的所述睡眠事件的发生使所述第一线程从活动状态转换为睡眠状态。

8.如权利要求 7 所述的方法，其特征在于包括：对所述睡眠事件的发生的所述检测进行响应，设置禁止寄存器以禁止不是所述第一线程的睡眠状态的中断事件的事件。

9.如权利要求 1 所述的方法，其特征在于包括对所述第一线程的所述多线程处理器中的状态进行保存和取消分配。

10.如权利要求 9 所述的方法，其特征在于所述对所述第一线程的所述多线程处理器中的状态进行保存和取消分配包括把所述第一线程的所述状态记录在存储器资源中。

11.如权利要求 1 所述的方法，其特征在于包括使所述多线程处理器的寄存器文件中的寄存器可以为所述多线程处理器中的第二线程所用。

12.如权利要求 1 所述的方法，其特征在于所述功能单元包含包括存储器顺序缓冲器、存储缓冲器、翻译后援缓冲器、重新排序缓冲器、寄存器别名表及自由列表管理器在内的组中的任一个。

13.如权利要求 1 所述的方法，其特征在于包括把电子篱笆指令在最接近所述多线程处理器的前端的位置插入所述第一线程的指令流，所述电子篱笆指令定义所述指令流中的事件边界，其中假定所有存储器访问都已从所述处理器中排出。

14.如权利要求 1 所述的方法，其特征在于包括恢复所述多线程处理器中的状态。

15.如权利要求 1 所述的方法，其特征在于所述检测所述第一线程中的所述状态变化包括检测所述第一线程的中断事件的发生，其中所述第一线程的中断事件的发生使所述第一线程从睡眠状态转换为活动状态。

16.如权利要求 15 所述的方法，其特征在于包括：检测不构成中断事件的所述第一线程的第三事件，并将所述第三事件记录在与所述第一线程有关的未决寄存器中。

17.一种装置，包括：

状态机，用于提供多线程处理器中执行的多线程的有关线程的相应状态的表示，并且检测所述多线程处理器中第一线程的状态变化；以及

配置逻辑装置，当所述第一线程的状态变化包括从活动状态到非活动状态并且第二线程处于活动状态时，改变功能单元的分区方案，以便为所述多线程处理器中的第二线程提供服务，但不为第一线程提供服务。

18.如权利要求 17 所述的装置，其特征在于所述状态表示所述有关线程是活动的或是非活动的。

19.如权利要求 18 所述的装置，其特征在于所述配置逻辑装置对所述功能单元进行分区，以便在所述第一线程的所述状态变化包括从非活动状态向活动状态的转换并且第二线程处于活动状态时，为所述多线程处理器中的所述第一线程和所述第二线程提供服务。

20.如权利要求 17 所述的装置，其特征在于所述状态机通过检测所述第一线程的事件的发生来检测所述第一线程的所述状态变化。

21.如权利要求 20 所述的装置，其特征在于包括事件检测器，所述事件检测器对所述第一线程的所述事件的发生作出响应而断言清除信号，其中在所述第一信号的所述断言期间对所述状态机进行评估。

22.如权利要求 21 所述的装置，其特征在于：在所述清除信号的取消断言时，所述配置逻辑装置根据所述状态机的所述多位输出对所述多线程处理器中的所述功能单元进行配置。

23.如权利要求 17 所述的装置，其特征在于：用于检测所述第一线程中的所述状态变化的所述状态机检测所述第一线程的睡眠事件的发生，其中所述第一线程的睡眠事件的发生使所述第一线程从活动状态转换为睡眠状态。

24.如权利要求 23 所述的装置，其特征在于包括微码定序器，所述微码定序器对所述睡眠事件的发生的检测进行响应，发出微指令来

设置禁止寄存器，以禁止不是所述第一线程的所述睡眠状态的中断事件的事件。

25.如权利要求 17 所述的装置，其特征在于所述配置逻辑装置对所述第一线程的有关功能单元中的状态进行保存、取消分配及恢复。

26.如权利要求 25 所述的装置，其特征在于与所述功能单元有关的所述配置逻辑装置把所述第一线程的状态信息记录在存储器资源中，以便对状态进行保存和取消分配，并将所述第一线程的状态信息从所述存储器资源中恢复到功能单元中，以便恢复状态。

27.如权利要求 25 所述的装置，其特征在于：与所述功能单元有关的所述配置逻辑装置，在所述第一线程存在时，使分配给所述第一线程的所述多线程处理器的寄存器文件中的寄存器可以为所述多线程处理器中的第二线程所用，并且在所述第二线程存在时，使分配给所述第二线程的所述多线程处理器的所述寄存器文件中的寄存器可以为所述多线程处理器中的所述第一线程所用。

28.如权利要求 17 所述的装置，其特征在于：所述功能单元包含包括存储顺序缓冲器、存储缓冲器、翻译后援缓冲器、重新排序缓冲器、寄存器别名表及自由列表管理器在内的组中的任一个。

29.如权利要求 17 所述的装置，其特征在于包括微码定序器，所述微码定序器把电子篱笆指令在最接近所述多线程处理器的前端的位置引入所述第一线程的指令流，所述电子篱笆指令定义所述指令流中的事件边界，以确保所有存储器存取从所述处理器中排出。

30.如权利要求 17 所述的装置，其特征在于所述配置所述功能单元包括恢复所述多线程处理器中的状态。

31.如权利要求 21 所述的装置，其特征在于：所述事件检测器通过检测所述第一线程的中断事件的发生，来检测所述第一线程的所述状态变化，其中所述第一线程的中断事件的发生使所述第一线程从睡眠状态转换为活动状态。

32.如权利要求 21 所述的装置，其特征在于所述事件检测器检测

不构成中断事件的所述第一线程的第三事件，并将所述第三事件记录在与所述第一线程有关的未决寄存器中。

33.一种装置，包括：

第一装置，用于表示多线程处理器中执行的多线程的有关线程的相应状态，并且检测所述多线程处理器中第一线程的状态变化；以及

第二装置，用于当所述第一线程的状态变化包括从活动状态到非活动状态并且第二线程处于活动状态时，改变功能单元的分区方案，以便为所述多线程处理器中的第二线程提供服务，但不为第一线程提供服务。

34.一种系统，包括：

一个多线程处理器，所述多线程处理器包括：

状态机，用于提供多线程处理器中执行的多线程的有关线程的相应状态的表示，并且检测所述多线程处理器中第一线程的状态变化；以及

配置逻辑装置，当所述第一线程的状态变化包括从活动状态到非活动状态并且第二线程处于活动状态时，改变功能单元的分区方案，以便为所述多线程处理器中的第二线程提供服务，但不为第一线程提供服务；

存储所述多线程的存储器；

将所述多线程送至所述多线程处理器的总线；以及

与所述总线相连的网络接口装置。

进入和退出多线程处理器中多线程的方法和装置

技术领域

一般来说，本发明涉及多线程处理器领域，更具体地说，涉及进入和退出多线程（MT）处理器中多线程的方法及装置。

背景技术

近来，多线程（MT）处理器设计被认为是增强处理器性能的一个极有吸引力的选择。其中，处理器中的多线程技术提供了更为有效利用各种处理器资源的可能性，尤其是更为有效利用处理器中执行逻辑装置的可能性。具体地说，通过将多线程馈送给处理器的执行逻辑装置，时钟周期可以用来为另一个线程提供服务，所述时钟周期不然因在特定线程处理过程中的停止或其它延时而为空闲。特定线程处理过程中的停止可能由处理器流水线中许多事件所引起。例如，对于包含在线程中的指令的高速缓存未命中或转移误预测(即较长等待时间操作)，通常导致对相关线程停止的处理。较长等待时间操作对执行逻辑装置效率的消极影响由于近来执行逻辑装置吞吐量的增加而加剧，其中，执行逻辑装置吞吐量的增加已经超过了存储器存取和检索速率的发展。

鉴于诸如 Windows NT®和 Unix 操作系统的许多流行操作系统对这类多线程应用所提供的支持，多线程计算机应用也日益普及。在多媒体领域中，多线程计算机应用尤为有效。

根据相关处理器中所采用的线程交错存取或交换方案，多线程处理器大致上可以分为两类（即精设计或粗设计）。精多线程设计支持处理器中的多活动线程，并且通常逐个周期地交错两个不同的线程。在出现诸如高速缓存未命中的某个长等待时间事件时，粗多线

程设计通常交错不同线程的指令。Eickemayer, R.、Johnson, R 等人的“商业应用环境的多线程单处理器的评估”一文（The 23rd Annual International Symposium on Computer Architecture pp.203-212, May 1996.）中讨论了粗多线程设计。多线程计算机体系结构：技术现状概要一书中 Laudon, J、Gupta, A 的“多环境处理器设计中体系结构和实现折中”（edited by R.A. Iannuci et al., pp.167-200, Kluwer Academic Publisher, Norwell, Massachusetts, 1994）进一步说明精设计和粗设计之间的差别。Laudon 还建议了一种交错方案，将精设计的逐个周期交换与粗设计的全流水线互锁结合（或封锁方案）。为此，Laudon 建议了“补偿”指令，使特定线程（或环境）对于一定数量的周期不可用。在出现诸如高速缓存未命中的预定事件时可发出这种“补偿”指令。这样，通过简单地使其中一个线程不可用，Laudon 即避免了必须执行实际的线程交换。

在无序的推测性执行处理器体系结构环境下，处理器的多线程体系结构提出了许多其它挑战。更具体地说，当考虑多线程时，可能导致指令流的流程中意外变化的事件处理（例如转移指令、异常或中断）是复杂的。在实现多线程间资源共享的处理器中（即：存在有限或无复制的功能单元用于该处理器支持的每个线程），有关特定线程的事件发生的处理的复杂之处在于：处理这类事件时必须考虑到其它线程。

在多线程处理器中实现资源共享的情况下，还希望尝试增加利用对该多线程处理器中所服务的线程状态中的变化作出响应的共享资源。

发明内容

根据本发明，提供了一种方法，该方法包括维护状态机以提供多位输出,多位输出的每一位均表示多线程处理器中执行的多线程的相关线程的相应状态。检测多线程处理器中第一线程的状态变化。

多线程处理器中的功能单元根据状态机的多位输出而被配置。

根据本发明的一种方法，包括以下步骤:

维护状态机以提供多位输出，所述多位输出的每一位都表示多线程处理器所执行的多线程中的有关线程的相应状态，其中所述多位输出的每一位都表示所述有关线程的所述状态是活动的或是非活动的;

检测所述多线程处理器中第一线程的状态的变化；以及

根据所述状态机的所述多位输出来配置所述多线程处理器中的功能单元。

根据本发明的一种装置，包括:

状态机，用于提供多位输出，所述多位输出的每一位都表示多线程处理器所执行的多线程中的有关线程的相应状态，其中所述多位输出的每一位都表示所述有关线程的所述状态是活动的或是非活动的，并且检测所述多线程处理器中第一线程的状态的变化；以及

配置逻辑装置，用于根据所述状态机的所述多位输出来配置所述多线程处理器中的功能单元。

根据本发明的另一种装置，包括:

第一装置，用于提供多位输出，所述多位输出的每一位都表示多线程处理器所执行的多线程中的有关线程的相应状态，并且检测所述多线程处理器中第一线程的状态的变化，其中所述多位输出的每一位都表示所述有关线程的所述状态是活动的或是非活动的；以及

第二装置，用于根据所述状态机的所述多位输出来配置所述多线程处理器中的功能单元。

通过附图及下面的详细描述，本发明的其它特点将会明显。

附图说明

通过示例来说明本发明，但本发明并不限于所述附图的表示，

附图中相同标号表示相同的单元，附图中：

图 1 是方框图，说明具有多线程支持的处理器流水线的一个实施例。

图 2 是方框图，说明通用多线程微处理器形式的处理器的示例性实施例。

图 3 是方框图，说明示例性多线程微处理器的所选组件，具体描述各个功能单元，它们提供经逻辑装置分区以适应多线程的缓存（或存储）功能。

图 4 是方框图，说明根据一个实施例的无序群集器。

图 5 是寄存器别名表和寄存器文件的图示，并被用于一个实施例中。

图 6A 是方框图，详细说明根据一个实施例的重新排序缓冲器，它被逻辑装置分区以为多线程处理器中的多线程提供服务。

图 6B 是按照一个实施例的未决事件寄存器和事件禁止寄存器的图示。

图 7A 是流程图，说明根据一个实施例的、处理多线程处理器中的事件的一种方法。

图 7B 是流程图，说明根据一个实施例的、处理多线程处理器中的“虚拟彻底清除（virtual nuke）”事件的一种方法。

图 8 是根据一个实施例的在多线程处理器中实现的多个可由事件检测器检测的示例性事件的图示。

图 9 和 10 是方框图，分别说明诸如图 6A 所示的示例性重新排序缓冲器中重新排序表的示例性内容。

图 11A 是流程图，说明根据一个示例性实施例的、在至少支持第一和第二线程的多线程处理器中执行清除（或彻底清除）操作的一种方法。

图 11B 是方框图，说明根据一个示例性实施例的、用来根据活动线程状态机的输出来配置功能单元的配置逻辑装置。

图 12 是时序图，说明根据一个实施例断言彻底清除信号。

图 13 是流程图，说明根据一个实施例的、在多线程处理器中提供独占访问事件处理程序的一种方法。

图 14 是状态图，说明根据一个实施例的、在多线程处理器中实现的独占访问状态机的操作。

图 15 是状态图，说明根据一个实施例的可由多线程处理器中实现的活动线程状态机占用的状态。

图 16A 是流程图，说明根据一个实施例的、在检测到多线程处理器中活动线程的睡眠事件时退出该活动线程的一种方法。

图 16B 是根据一个实施例的、在退出线程时储存状态和释放寄存器的图示。

图 17 是流程图，说明根据一个实施例的、在检测到非活动线程的中断事件时将该线程从非活动状态转换为活动状态的一种方法。

图 18 是流程图，说明根据一个实施例的、对多线程处理器中至少一个功能单元的时钟信号的允许和禁止进行管理的一种方法。

图 19A 是方框图，说明根据一个实施例的用于允许和禁止多线程处理器中的时钟信号的时钟控制逻辑装置。

图 19B 是示意图，说明图 19A 所示时钟控制逻辑装置的一个实施例。

具体实施方式

现说明用于进入和退出多线程处理器中多线程的方法和装置。为便于说明，在以下描述中给出了大量具体细节，以便透彻理解本发明。然而，本领域的技术人员将清楚，没有这些具体细节也可以实施本发明。

为便于说明，术语“事件”将包括处理器内部或外部的任何事件，所述事件引起处理器中指令流（宏指令或微指令）服务的改变或中断。因此，术语“事件”将包括但不限于可在处理器内部或外

部产生的转移指令进程、异常以及中断。

为了便于说明，术语“处理器”应表示能执行指令序列（例如宏指令或微指令）的任何机器，且应包括但不限于通用微处理器、专用微处理器、图形控制器、音频控制器、多媒体控制器、微控制器或网络控制器。此外，术语“处理器”应还表示复杂指令集计算机（CISC）、精简指令集计算机（RISC）或超长指令字（VLIW）处理器。

另外，术语“清除点”应包括通过流程标记符在指令流中（包括微指令或宏指令流）提供的任何指令，或可以处理事件的指令流中某位置的其它指令。

术语“指令”应包括但不限于宏指令或微指令。

对主要以硬件或软件实现的本发明某些示例性实施例进行说明。然而，本领域的技术人员将知道，可以容易地以硬件、软件或硬件和软件的组合来实现许多特性。

实现本发明实施例的软件（例如微指令和宏指令）可以完全或至少部分地驻留在处理器可存取的主存储器中和/或处理器中（例如在高速缓存或微码定序器）。例如，事件处理程序和状态机可用微码定序器发送的微码来实现。

软件还可以经网络接口装置进行传送或接收。

为了便于说明，术语“机器可读媒体”应包括能储存机器执行的指令序列或对其进行编码的任何媒体，其中机器执行的指令序列使机器执行本发明方法的任何一种方法。因此，术语“机器可读媒体”应包括但不限于固态存储器、光盘和磁盘、以及载波信号。

处理器流水线

图 1 是一个高级方框图，说明处理器流水线 10 的一个实施例。流水线 10 包括多个管道级，以提取管道级 12 开始，在该级检索指令（例如宏指令）并将其馈送到流水线 10。例如，可以从结合在处理器中或与处理器紧密关联的高速缓冲存储器中检索宏指令，或者

经处理器总线从外部主存储器中检索宏指令。从提取管道级 12 开始，宏指令被传送到解码管道级 14，宏指令在该级被翻译为适合在处理器中执行的微指令（又称作“微码”）。然后，微指令被下传给分配管道级 16，在该级根据可用性和需求将处理器资源分配给各个微指令。微指令则在退役管道级 20 退役或“回写”（例如提交给体系结构状态）之前在执行级 18 被执行。

微处理器体系结构

图 2 是方框图，说明通用微处理器形式的处理器 30 的示例性实施例。下面将处理器 30 作为多线程（MT）处理器进行说明，因此它能处理多指令线程（或环境）。但是，以下在说明中提供的多个示教并不是特定用于多线程处理器的，它们也可以应用于单线程处理器中。在一个示例性实施例中，处理器 30 可以包括 Intel 体系结构(IA)微处理器，它能执行 Intel 体系结构指令集。这种 Intel 体系结构微处理器的一个示例是 Intel 公司（Santa Clara, California）制造的 Pentium Pro®微处理器或 Pentium III®微处理器。

在一个实施例中，处理器 30 包括有序前端和无序后端。有序前端包括总线接口单元 32，用作处理器 30 和可以使用处理器 30 的计算机系统中的其它组件（例如主存储器）之间的导管。为此，总线接口单元 32 将处理器 30 与处理器总线（未示出）连接，其中，可以经处理器总线在处理器 30 接收或者从处理器 30 传送数据和控制信息。总线接口单元 32 包括前侧总线（FSB: Front Side Bus）逻辑装置 34，控制通过处理器总线进行的通信。总线接口单元 32 还包括总线队列 36，对通过处理器总线进行的通信提供缓冲功能。所示总线接口单元 32 从存储器执行单元 42 接收总线请求 38 并向存储器执行单元 42 发送窥探或总线返回值，其中存储器执行单元 42 提供处理器 30 中的本地存储器能力。存储器执行单元 42 包括统一数据和指令高速缓存 44、数据翻译后援缓冲器（TLB）46 以及存储器排序缓冲器 48。存储器执行单元 42 从微指令翻译引擎 54 接收指令提取

请求 50 并向微指令翻译引擎 54 传送原始指令 52（即编码宏指令），其中微指令翻译引擎 54 将接收的宏指令翻译为相应的微指令集。

微指令翻译引擎 54 实际作为跟踪高速缓存“未命中处理程序”，用于在跟踪高速缓存未命中的情况下将微指令传送给跟踪高速缓存 62。为此，在跟踪高速缓存未命中的情况下，微指令翻译引擎 54 用来提供提取和解码管道级 12 和 14。所示微指令翻译引擎 54 包括下一指令指针（NIP）100、指令翻译后援缓冲器（TLB）102、转移预测器 104、指令流缓冲器 106、指令预解码器 108、指令导引逻辑装置 110、指令解码器 112 以及转移地址计算器 114。下一指令指针 100、TLB 102、转移预测器 104 以及指令流缓冲器 106 共同组成转移预测单元（BPU）99。指令解码器 112 和转移地址计算器 114 共同组成指令翻译（IX）单元 113。

下一指令指针 100 向统一高速缓存 44 发出下一指令请求。在示例性实施例中，处理器 30 包括能处理两个线程的多线程微处理器，下一指令指针 100 可以包括复用器（MUX）（未示出），复用器在与包含在从下一指令指针 100 发出的下一指令请求中的第一或第二线程有关的指令指针之间进行选择。在一个实施例中，下一指令指针 100 逐个周期地（“乒乓”）交错第一和第二线程的下一指令请求，其中假定两个线程的指令已被请求，并且用于两个线程的指令流缓冲器 106 资源还没有耗尽。根据初始请求地址是处于 32 字节还是 64 字节定位线的上半部分，下一指令指针请求可以适合于 16、32 或 64 字节。下一指令指针 100 可以通过转移预测器 104、转移地址计算器 114 或跟踪高速缓存 62 来重新定向，跟踪高速缓存未命中请求为最高优先级重新定向请求。

当下一指令指针 100 向统一高速缓存 44 进行指令请求时，生成两位的“请求标识符”，与该指令请求关联，并用作相关指令请求的“标记符”。当响应指令请求而返回数据时，统一高速缓存 44 将以下标记符或标识符与数据一起返回：

1.下一指令指针 100 提供的“请求标识符”;

2.标识返回块的三位“块标识符”; 以及

3.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，后者执行地址查找操作，并将物理地址传送到统一高速缓存 44。统一高速缓存 44 将对应的宏指令传送到指令流缓冲器 106。每下一指令请求也直接从下一指令指针 100 传送到指令流缓冲器 106，以便允许指令流缓冲器 106 来标识从统一高速缓存 44 接收的宏指令所属的线程。然后，来自第一和第二线程的宏指令从指令流缓冲器 106 发送到指令预解码器 108，后者执行关于接收指令流的（宏指令的）多个长度计算和字节标记操作。具体地说，指令预解码器 108 生成一系列字节标记矢量，它们也用来区分传送到指令导引逻辑装置 110 的指令流中的宏指令。

随后，指令导引逻辑装置 110 利用字节标记矢量来将离散宏指令导引到指令解码器 112，用于解码。宏指令还从指令导引逻辑装置 110 传送到转移地址计算器 114，用于转移地址计算。然后，微指令从指令解码器 112 传送到跟踪传送引擎 60。

在解码过程中，将流程标记符与由宏指令翻译成的各微指令相关联。流程标记符表示有关微指令的特征，并且可以例如表示有关微指令为代表宏指令的微码序列中的第一或最后微指令。流程标记符包括“宏指令开始”（BOM）和“宏指令结束”（EOM）流程标记符。根据本发明，解码器 112 还可以对微指令进行解码，以便使共享资源（多处理器）（SHRMP）流程标记符和同步（SYNC）流程标记符与其相关联。具体地说，共享资源流程标记符将微指令标识为特定线程中的位置，在这个位置，该线程可以被中断（例如被重新启动或暂停），其负面结果比线程中其它位置上的要小。在本发明的一个示例性实施例中，解码器 112 被构造为用共享资源流程标记符以及较长微码序列中的断续点来标记包含父宏指令的结束或开始的微指

令。同步流程标记符将微指令标识为特定线程中的位置，在这个位置，所述线程可以例如对另一个线程中的同步指令作出响应而与其同步。为了便于说明，术语“同步”应表示至少一个线程中的至少第一点的标识，其中处理器状态可以相对于该线程和/或至少另一个线程来进行调整，相对于该线程中或另一个线程中的第二点，对处理器造成的中断要少。

在本发明的一个示例性实施例中，解码器 112 被构造为对微指令进行标记，所述微指令位于被选宏指令边界，其中并存于同一个处理器中的线程间共享的状态可以由一个线程改变，而不会给其它线程的执行带来负面影响。

解码的指令（即微指令）从微指令翻译引擎 54 发送到跟踪传送引擎 60。跟踪传送引擎 60 包括跟踪高速缓存 62、跟踪转移预测器（BTB）64、微码序列 66 以及微码（uop）队列 68。跟踪传送引擎 60 用作微指令高速缓存，并且是下游执行单元 70 的主要微指令源。通过在处理器流水线中提供微指令高速缓存功能，跟踪传送引擎 60，具体地说是跟踪高速缓存 62，使由微指令翻译引擎 54 所进行的翻译工作可以被操纵，以便提供增加的微指令带宽。在一个示例性实施例中，跟踪高速缓存 62 可以包括一个 256 的组，8 路成组相联存储器。在本示例性实施例中，术语“跟踪”可以表示储存在跟踪高速缓存 62 项目中的微指令序列，每个项目包括对包含跟踪的以前和进行中的微指令的指针。这样，跟踪高速缓存 62 便于高性能定序：在当前存取结束之前，就知道为获取随后的微指令而要存取的下一个项目的地址。在一个实施例中，跟踪可以看作是指令“块”，这些指令“块”通过跟踪头来相互区分，并且在遇到间接转移时结束，或者在到达许多给出的阈值条件的其中之一时，诸如单个跟踪所包含的条件转移数量或包含跟踪的微指令的最大总数，结束指令“块”。跟踪高速缓存转移预测器 64 提供与跟踪高速缓存 62 中的跟踪有关的局部转移预测。跟踪高速缓存 62 和微码定序器 66 将微指令提供

给微码队列 68，然后，再从其中将微指令馈送到无序执行群集器。所示微码定序器 66 还包括以微码实施的多个事件处理程序 67，对出现诸如异常或中断的事件进行响应，实现处理器 30 中的多个操作。事件处理程序 67，稍后将被详细说明，由包含在处理器 30 后端的寄存器更名器 74 中的事件检测器 188 进行调用。

处理器 30 可以被看作具有有序前端和无序后端，其中，有序前端包括总线接口单元 32、存储器执行单元 42、微指令翻译引擎 54 及跟踪传送引擎 60，无序后端将在下面被详细说明。

从微码队列 68 发送的微指令被接收到无序群集器 71 中，其中无序群集器 71 包括调度器 72、寄存器更名器 74、分配器 76、重新排序缓冲器 78 及重放队列 80。调度器 72 包括一组预定站，并用于调度和发送微指令，由执行单元 70 执行。寄存器更名器 74 相对于隐含整数和浮点寄存器（可以用来代替八个通用寄存器中的任何一个或八个浮点寄存器中的任何一个，其中处理器 30 执行 Intel 体系结构指令集）来执行寄存器重命名功能。分配器 76 用来根据可用性和需求把执行单元 70 和群集器 71 的资源分配给微指令。在没有足够资源用来处理微指令的情况下，分配器 76 负责断言停止信号 82，它通过跟踪传送引擎 60 传送到微指令翻译引擎 54，如图中 58 所示。源字段由寄存器更名器 74 进行调整的微指令按严格的程序顺序放置在重新排序缓冲器 78 中。当重新排序缓冲器 78 中的微指令已经完成执行并准备退役时，则从重新排序缓冲器中将这些微指令移去并以有序的方式（即按照原程序顺序）检索这些微指令。重放队列 80 把要重放的微指令传送给执行单元 70。

所示执行单元 70 包括浮点执行引擎 84、整数执行引擎 86 以及 0 级数据高速缓存 88。在处理器 30 执行 Intel 体系结构指令集的一个示例性实施例中，浮点执行引擎 84 还可以执行 MMX® 指令和流 SIMD（单指令、多数据）扩展（SSE）。

多线程实现

在图 2 所示处理器 30 的示例性实施例中，可能存在支持多线程功能的资源的有限复制，因此，需要在线程间实现某种程度的资源共享。将会知道，所采用的资源共享方案取决于处理器能同时处理的线程数量。由于处理器中的功能单元通常提供一些缓冲（或存储）功能和传送功能，所以资源共享的问题可以看作是包括（1）存储，以及（2）处理/传送带宽共享组件。例如，在支持同时处理两个线程的处理器中，各种功能单元中的缓冲器资源可以在两个线程之间被静态或逻辑装置分区。同样，由用于两个功能单元之间信息传送的通路所提供的带宽必须在两个线程之间进行划分和分配。由于这些资源共享问题可能会出现在处理器流水线中的多个位置，所以，可以根据特定位置的指示和特征，在这些不同位置采用不同的资源共享方案。将会知道，考虑到不同的功能和操作特性，不同的资源共享方案可以适合于不同的位置。

图 3 是方框图，说明图 2 所示处理器 30 的一个实施例的所选组件，并说明了各种功能单元，它们提供经逻辑装置分区以适应两个线程（即线程 0 和线程 1）的缓冲能力。通过将缓冲资源中的第一预定项目集分配给第一线程，以及将缓冲资源中的第二预定项目集分配给第二线程，可以实现用于功能单元的缓存（或存储）和处理装置的两个线程的逻辑装置分区。然而，在其它实施例中，还能动态共享缓冲。具体地说，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第一线程有关，第二对读和写指针则与第二线程有关。第一读和写指针集可限于缓冲资源中第一预定项目数量，而第二读和写指针集可限于相同缓冲资源中第二预定项目数量。在所述实施例中，所示指令流缓冲器 106、跟踪高速缓存 62 以及指令队列 103 均提供在第一和第二线程之间逻辑装置分区的存储能力。

无序群集器（71）

图 4 是方框图，进一步详细说明无序群集器 71 的一个实施例。群集器 71 提供处理器 30 中的预定站、寄存器重命名、重放以及退

役功能。群集器 71 从跟踪传送引擎 60 接收微指令，将资源分配给这些微指令，重命名各微指令的源和目标寄存器，调度微指令以便发送到适当的执行单元 70，处理由于数据推测而重放的微指令，最后使微指令退役（即，使微指令处于持久体系结构状态）。

群集器 71 接收的微指令同时被传送到寄存器别名表 120 和分配及自由列表管理逻辑装置 122。寄存器别名表 120 负责将逻辑装置寄存器名翻译成由调度器 72 和执行单元 70 使用的物理寄存器地址。更具体地说，参照图 5，寄存器别名表 120 重命名物理寄存器文件 124 中维护的整数、浮点及段寄存器。所示寄存器文件 124 包括 126 个物理寄存器，为八（8）个体系结构寄存器的别名。在所述实施例中，所示寄存器别名表 120 包括前端表 126 和后端表 128，供处理器 30 相应前端及后端使用。寄存器别名表 120 中的每个项目均与体系结构寄存器有关，或者被看作是体系结构寄存器，并且包括指针 130，指向储存了属于相关体系结构寄存器的数据的寄存器文件 124 中的位置。这样，指定较少数量体系结构寄存器的传统微处理器体系结构所产生的问题可以得到解决。

分配及自由列表管理逻辑装置 122 负责群集器 71 中的资源分配和状态恢复。逻辑装置 122 将下列资源分配给每个微指令:

1. 序列号，它被提供给各微指令，以便当微指令在群集器 71 中被处理时，跟踪线程中微指令的逻辑装置顺序。属于各微指令的序列号与微指令的状态信息一起储存在重新排序缓冲器 162 的表 180（下面在图 10 中示出）中。

2. 自由列表管理项目，它被提供给各微指令，以便允许在状态恢复操作时跟踪和恢复微指令的历史记录。

3. 重新排序缓冲器（ROB）项目，它按序列号来索引。

4. 物理寄存器文件 124 项目（称作“大理石”），微指令可以在其中储存有用结果。

5. 加载缓冲器（未示出）项目。

6. 停止缓冲器（未示出）项目。

7. 指令队列项目（例如，对存储器指令队列或通用指令地址队列，下面将进行说明）。

在逻辑装置 122 无法获取所接收的微指令序列的必要资源时，逻辑装置 122 将请求跟踪传送引擎 60 停止传送微指令，直到有足够资源可用为止。通过断言图 2 所示的停止信号 82 来传送该请求。

对于向微指令分配寄存器文件 124 中的项目，图 5 示出垃圾堆阵列（trash heap array）132，用来维护寄存器文件 124 中没有分配给体系结构寄存器的项目记录（即：它们在寄存器别名表 120 中没有指针）。逻辑装置 122 访问垃圾堆阵列 132，以便标识寄存器文件 124 中可分配给所接收的微指令的项目。逻辑装置 122 还负责回收寄存器文件 124 中成为可用的项目。

逻辑装置 122 还维护自由列表管理器（FLM）134，以便能跟踪体系结构寄存器。具体地说，当微指令被分配到的寄存器别名表 120 时，自由列表管理器 134 保存寄存器别名表 120 的变化的历史记录。自由列表管理器 134 提供“展开”寄存器别名表 120 的能力，以便在有误预测或事件的情况下指向非推测性状态。自由列表管理器 134 还使寄存器文件 124 的项目中的数据存储“老化”，以便保证所有状态信息均是最近的。最后，在退役时，物理寄存器标识符从自由列表管理器 134 传送到垃圾堆阵列 132，以便分配给其它微指令。

指令队列单元 136 以顺序程序顺序将微指令传送到调度器和记分板单元（SSU）138，并保持和发送执行单元 70 所需的微指令信息。指令队列单元 136 可以包括两个不同的结构，即指令队列（IQ）140 和指令地址队列（IAQ）142。指令地址队列 142 是小型结构，被设计成按需要把关键信息（例如微指令源、目的地以及等待时间）馈送到单元 138。指令地址队列 142 还可以包括：存储器指令地址队列（MIAQ）和通用指令地址队列（GIAQ），其中存储器指令地址队列（MIAQ）对存储器操作的信息进行排队，而通用指令地址队列

（GIAQ）对非存储器操作的信息进行排队。指令队列 140 储存不太关键信息，诸如微指令的操作码和直接数据。当读取相关微指令并将其写入调度器和记分板单元 138 时，从指令队列单元 136 解除微指令的分配。

调度器和记分板单元 138 负责通过确定各微指令源可能准备就绪的时间以及适当的执行单元可用于发送的时间，来调度用于执行的微指令。图 4 所示的单元 138 包括寄存器文件记分板 144、存储调度器 146、矩阵调度器 148、慢微指令调度器 150 以及浮点调度器 152。

单元 138 通过检查寄存器文件记分板 144 中所保持的信息，来确定源寄存器准备就绪的时间。为此，在一个实施例中，寄存器文件记分板 144 具有 256 位，跟踪对应于寄存器文件 124 中各寄存器的数据资源的可用性。例如，在向相关项目分配数据或向单元 138 进行写入操作时，可清除寄存器文件 124 中特定项目的记分板位。

存储调度器 146 缓冲存储器类微指令，检查资源可用性，然后再对存储器类微指令进行调度。矩阵调度器 148 包括两个紧密结合的算术逻辑装置单元（ALU）调度器，允许对相关背对背微指令的调度。浮点调度器 152 缓冲浮点微指令并对其进行调度，而慢微指令调度器 150 则对未由上述调度器处理的微指令进行调度。

所示检验器、重放及退役单元（CRU）160 包括重新排序缓冲器 162、检验器 164、分级队列 166 及退役控制电路 168。单元 160 具有三种主要功能，也就是检验功能、重放功能及退役功能。具体地说，检验器和重放功能包含重新执行未正确执行的微指令。退役功能包括将体系结构有序状态提交给处理器 30。更具体地说，检验器 164 用来保证每个微指令都正确地执行了正确的数据。在没有用正确的数据执行微指令时（例如由于误预测转移），则重放有关微指令来用正确的数据执行。

重新排序缓冲器 162 负责通过按程序顺序使微指令退役，来将体系结构状态提交给处理器 30。由退役控制电路 168 产生的退役指

针 182 表示重新排序缓冲器 162 中在退役的项目。当退役指针 182 通过项目中的微指令时，则释放自由列表管理器 134 中的对应项目，并且相关寄存器文件项目现在可被回收并传送到垃圾堆阵列 132。所示退役控制电路 168 实现活动线程状态机 171，其目的和功能将在下面被说明。退役控制电路 168 控制向寄存器文件 124 中的对应体系结构状态提交保持在重新排序缓冲器 162 中的推测性结果。

重新排序缓冲器 162 还负责处理内部及外部事件，下面将进行详细说明。重新排序缓冲器 162 检测到事件发生时，则断言“彻底清除”信号 170。彻底清除信号 170 具有从当前正在转换的处理器流水线中冲洗（flushing）所有微指令的作用。重新排序 162 还为跟踪传送引擎 60 提供一个地址，从该地址开始对微指令定序，以便为事件提供服务（即：从该地址开始发送以微码实施的事件处理程序 67）。

重新排序缓冲器（162）

图 6A 是方框图，详细说明重新排序缓冲器 162 的一个示例性实施例，它经逻辑装置分区来为多线程处理器 30 中的多线程提供服务。具体地说，所示重新排序缓冲器 162 包括重新排序表 180，当处理器 30 以多线程模式运行时，可经逻辑装置分区来容纳第一和第二线程的项目。以单线程模式运行时，整个表 180 可以用来为单线程提供服务。在一个实施例中，表 180 包括单式存储结构，以多线程模式运行时，该单式存储结构由两（2）个退役指针 182 和 183 引用，其中，退役指针 182 和 183 限于表 180 中预定的和不同的项目集。同样，以单线程模式运行时，表 180 由单个退役指针 182 引用。表 180 包括与寄存器文件 124 的各项目对应的项目，并储存序列号及故障信息形式的状态信息、逻辑装置目标地址以及寄存器文件 124 中各微指令数据项目的有效位。表 180 中的项目均按照组成各微指令的唯一标识符的序列号进行索引。按照序列号，表 180 中的项目以顺序且有序的方式进行分配和解除分配。除了其它流程标记符之外，所示表 180 还储存各微指令的共享资源流程标记符 184 和同步流程

标记符 186。

重新排序缓冲器 162 包括事件检测器 188，该事件检测器 188 被连接以便接收中断矢量形式的中断请求，并存取由退役指针 182 和 183 引用的表 180 中的项目。所示事件检测器 188 还输出彻底清除信号 170 和清除信号 172。

假定特定线程（例如线程 0）的特定微指令未受转移误预测、异常或中断，当退役指针 182 或 183 加 1，以便寻址到相关项目时，则储存在表 180 项目中的特定指令的信息将被退役到体系结构状态。在这种情况下，形成退役控制电路 168 的一部分的指令指针计算器 190 将宏指令或微指令指针加 1，以便（1）指向寄存器文件 124 的对应项目中指定的转移目标地址，或者（2）如果没有发生转移，则指向下一个宏指令或微指令。

如果出现了转移误预测，信息则通过故障信息字段传送到退役控制电路 168 和事件检测器 188。由通过故障信息所表示的转移误预测来看，处理器 30 可能已经提取至少一些不正确的指令，这些指令已经渗入处理器流水线。由于表 180 中的项目是按顺序分配的，所以误预测转移微指令之后的所有项目都是受到误预测转移指令流程影响的微指令。对在故障信息中登记误预测转移的微指令的试退役作出响应，事件检测器 188 断言清除信号 172，该清除信号 172 清除处理器的整个无序后端的所有状态，从而刷新误预测微指令之后的指令所产生的所有状态的无序后端。清除信号 172 的断言也阻塞随后提取的微指令的发出，其中所述提取的微指令可位于处理器 30 的有序前端中。

在退役控制电路 168 中，当通过退役微指令的故障信息通知误预测转移时，IP 计算器 190 确保对指令指针 179 和/或 181 进行更新，以表示正确的指令指针值。根据是否会发生转移，IP 计算器 190 采用来自与表 180 的相关项目对应的寄存器文件项目中的结果数据，来更新指令指针 179 和/或 181，或者当没有发生转移时，将指令指

针 179 和 181 加 1。

事件检测器 188 还包括多个寄存器 200，用于维护有关对多线程中的每个线程所检测的事件的信息。寄存器 200 包括事件信息寄存器 202、未决事件寄存器 204、事件禁止寄存器 206、展开寄存器 208 以及管脚状态寄存器 210。寄存器 202-210 中的每一个都能储存有关对特定线程产生的事件的信息。因此，多线程的事件信息可以由寄存器 200 进行维护。

图 6B 是第一线程（例如 T0）的示例性未决事件寄存器 204 和示例性事件禁止寄存器 206 的示意图。

对多线程处理器 30 中支持的每个线程均提供未决事件寄存器 204 和事件禁止寄存器 206。对每个线程提供不同的寄存器 204 和 206，或者，对单个物理寄存器进行逻辑装置分区以支持多线程。

示例性未决事件寄存器 204 包含一位或其它数据项，用于由事件检测器 188 登记的每个事件类型（例如以下参照图 8 所述的事件）。这些事件可以组成处理器 30 内部产生的内部事件或处理器 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）。在所述实施例中，每个线程的未决事件寄存器 204 不包含回写事件的位，因为这样的事件不是线程特定的，并因此未在未决事件寄存器中“排队”。为此，事件检测器 188 可以包括回写检测逻辑装置 205，在检测到回写事件时断言回写信号。未决事件寄存器 204 中各线程的位由事件检测器 188 进行设置，其中，事件检测器 188 触发一个锁存器，在未决事件寄存器 204 中设置适当位。在一个示例性实施例中，未决事件寄存器 204 中与预定事件有关的设置位提供未决的相关类型事件的表示，下面将进行说明。

各线程的事件禁止寄存器 206 同样包含一位或其它数据结构，用于由事件检测器 188 识别的每个事件类型，对该位进行设置或复位（即清除），以便记录关于特定线程为中断事件的事件。事件禁止寄存器 206 中的相应位由控制寄存器写操作进行设置，该操作使用

修改处理器 30 中非更名状态的特殊微指令。事件禁止寄存器 206 中的位同样可以采用控制寄存器写操作来进行复位（或清除）。

示例性处理器还可具有某些模式，在这些模式中，可以对事件禁止寄存器 206 中的位进行设置，以便禁止相应模式中的选择事件。

特定线程的每个未决事件寄存器 204 和事件禁止寄存器 206 中维护的特定事件类型的位被输出到“与”门 209，当寄存器 204 和 206 的内容表示相关事件类型未决或未被禁止时，“与”门 209 再输出各事件类型的事件检测信号 211。例如，在事件类型未被禁止的情况下，当事件在未决事件寄存器 204 中登记时，立即就该事件进行信号通知，由相关事件类型的事件检测信号 211 的断言所检测。另一方面，如果事件禁止寄存器 206 的内容禁止了该事件类型，该事件发生将会被记录在未决事件寄存器 204 中，但是如果清除了事件禁止寄存器 206 中的适当位，则只是断言事件检测信号 211，而该事件仍在寄存器 204 中记录为未决。这样，某个事件可以记录在未决事件寄存器 204 中，但只是在取消对特定线程的事件的禁止的稍后，可能才就相关事件发生的事件检测信号 211 进行信号通知。

每个线程的每个事件类型的事件检测信号 211 被馈送到事件处理逻辑装置（事件优先化和选择逻辑装置）和时钟控制逻辑装置，这将在下面进行说明。

一旦完成特定事件的处理，该特定事件的事件处理程序负责清除特定线程的未决事件寄存器 204 中的适当位。在另一个实施例中，未决事件寄存器可以通过硬件进行清除。

多线程处理器环境中的事件发生和事件处理

多线程处理器 30 中的事件可以从多种源来进行检测并由信号通知。例如，处理器 30 的有序前端可以就事件发出信号通知，执行单元 70 同样可以就事件发出信号通知。事件可以包括中断和异常。中断是处理器 30 外部产生的事件，并可经通用总线（未示出）由设备向处理器 30 发起。中断可能会使控制流程指向微码事件处理程序 67。

异常可以大致分为故障、陷阱及帮助等。异常是通常在处理器 30 中产生的事件。

事件直接传送给重新排序缓冲器 162 中的事件检测器 188，对其作出响应，事件检测器 188 执行与产生事件的线程有关的多个操作。在高层，事件检测器 188 对事件的检测作出响应，挂起线程的微指令的退役，将适当的故障信息写入表 180，断言彻底清除信号 170，调用事件处理程序 67 以便处理事件，确定重新开始地址，然后再重新开始提取微指令。事件可以采用中断请求（或中断矢量）的形式直接传送给事件检测器 188，或者通过记录在退役的第一或第二线程的指令的重新排序表 180 中的故障信息进行传送。

彻底清除信号 170 的断言具有清除多线程处理器 30 的有序前端和无序后端的状态的作用。具体地说，对彻底清除信号 170 的断言作出响应，许多功能单元，但不一定是所有单元，均清除状态和微指令。存储器顺序缓冲器 48 和总线接口单元 32 的某些部分没有进行清除（例如，退役但没有提交储存、总线窥探等）。彻底清除信号 170 的断言还停止前端进行的指令提取，并且还停止微指令向微码队列 68 中的定序。当该操作可以在单线程多处理器中或者在执行单线程的多处理器中顺利执行时，其中多线程是现存的并在多线程处理器 30 中被处理，当处理与单线程有关的事件发生时，其它线程的存在不能忽略。因此，本发明提出一种方法和装置，用于处理多线程处理器中的事件，当出现单线程的事件时，它了解多线程处理器 30 中多线程的处理和存在。

图 7A 是流程图，说明根据本发明的一个示例性实施例的一种方法 220，用于处理多线程处理器 30 中的事件发生。方法 220 在块 222 以第一线程的第一事件的事件检测器 188 进行检测开始。图 8 是可通过块 222 中事件检测器 188 检测的多个示例性事件 224 的图示。图 8 所表示的事件已经根据对事件 224 的响应的特征进行大致分组。第一组事件包括 RESET（复位）事件 226 和 MACHINE CHECK（机

器检查）事件 228，在进行检测时，第一组事件立即由事件检测器 188 按照下面描述的方式向多线程处理器 30 中的多线程发出信号通知，并使所有线程同时进入相同的事件处理程序 67。第二组事件包括 FAULT（故障）事件 230、ASSIST（帮助）事件 232、DOUBLE FAULT（双重故障）事件 234、SHUTDOWN（关机）事件 236 以及 SMC（自修改代码）事件 238，在就事件发出信号通知的特定线程的微指令的退役时，所述事件中的每一个被报告。具体地说，事件检测器 188 在故障信息表示故障状况的微指令退役时，将检测到第二组的某个事件。第二组某个事件的检测由事件检测器 188 仅向产生相关事件的线程发出信号通知。

第三组事件包括 INIT（短复位）事件 240、INTR（局部中断）事件 242、NMI（不可屏蔽中断）事件 244、DATA BREAKPOINT（数据断点）事件 246、TRACE MESSAGE(跟踪消息)事件 248 以及 A20M（地址回绕）事件 250。在具有接受中断或接受陷阱流程标记符的微指令退役时，报告第三组的事件。第三组事件的检测由事件检测器 188 仅向产生相关事件的线程发出信号通知。

第四组事件包括 SMI（系统管理中断）事件 250、STOP CLOCK（停止时钟）事件 252 及 PREQ（探查请求）事件 254。当多线程的其中任何一个使具有适当中断流程标记符的微指令退役时，就第四组事件向存在于多线程处理器 30 中的所有线程发出信号通知，并且报告第四组事件。在对第四组的任何一个事件作出响应的多线程之间没有实现同步。

根据一个示例性实施例，第五组事件特定于多线程处理器体系结构，并在所述实施例中实现，以针对特定于多线程处理器环境的多个考虑事项。第五组事件包括 VIRTUAL NUKE（虚拟彻底清除）事件 260、SYNCHRONIZATION（同步）事件 262 以及 SLEEP（睡眠）事件 264。

VIRTUAL NUKE 事件 260 是在以下时间相对于第二线程所登记

的事件:(1)多线程处理器 30 中的第一线程具有未决事件时(例如上述任何一个事件为未决),(2)第二线程不具有未决事件时(除事件 260 之外),以及(3)具有共享资源流程标记符 184 或同步流程标记符 186 的微指令由重新排序缓冲器 162 退役时。VIRTUAL NUKE 事件 260 具有调用虚拟彻底清除事件处理程序的作用,其中虚拟彻底清除事件处理程序在具有流程标记符 184 或 186 的退役微指令之后的微指令处重新开始执行第二线程。

当需要特定线程(例如第一线程)来修改多线程处理器 30 中的共享状态或资源时,SYNCHRONIZATION 事件 262 由微码发出通知信号。为此,微码定序器 66 将同步微指令插入第一线程的流程中,并且为了避免死锁情况,采用共享资源流程标记符 184 和同步流程标记符 186 来标记"同步微指令"。只在第一线程的同步微指令退役时,以及在具有相关联的同步流程标记符 186 的第二线程的微指令退役时,才检测(或登记)SYNCHRONIZATION 事件 262。SYNCHRONIZATION 事件 262 具有调用同步事件处理程序的作用,其中同步事件处理程序在储存于微码临时寄存器的指令指针处重新开始执行第一线程。有关 SYNCHRONIZATION 事件 262 的处理将在下面进行详细说明。第二线程执行虚拟 NUKE 260。

SLEEP 事件 264 是使相关线程从活动状态转换到非活动(或睡眠)状态的事件。非活动线程则可以通过适当的 BREAK 事件再从非活动状态转换到活动状态。将线程再转换到活动状态的 BREAK 事件的性质与将线程转换到非活动状态的 SLEEP 事件 264 相关。下面详细说明线程进入和退出活动状态的情况。

图 9 是方框图,说明重新排序缓冲器 162 中重新排序表 180 的示例性内容,为了说明本发明的一个示例性实施例中的事件和清除点(又称作"彻底清除点")检测,下面将会详细说明重新排序缓冲器 162 中重新排序表的内容。事件检测器 188 在块 222 对上述任何一个事件的检测可能会引起对事件 266 的响应,其中事件 266 是从

多线程处理器 30 中的内部源或从处理器 30 外的外部源传送到事件检测器 188 的。这样的事件 266 的传送的一个示例可以是中断矢量。另一方面，事件发生可以通过退役从而由退役指针 182 标识的特定线程（例如线程 1）的微指令的故障信息 268 传送给事件检测器 188。要指出，对于外部事件，每个线程有一（1）个信号（例如相应地信号 266 和 267）。对于内部事件，含有线程的重新排序缓冲器 162 项目指示故障通过其位置相关的线程（例如 T0 对 T1）。在检测到事件时，事件检测器 188 将有关特定事件的事件信息（例如事件类型、事件源等）储存在事件信息寄存器 202 中，并且还在未决事件寄存器 204 中登记相关线程的未决事件。如上所述，在相关线程的未决事件寄存器 204 中登记未决事件包括在寄存器 204 中设置与特定事件有关的位。还要指出，如果事件没有被相关线程的事件禁止寄存器 206 中的位设置所禁止，则通过断言适当的事件检测信号 211，可以有效地检测事件，在某些情况下，微指令包含适当的流程标记符。

现在回到图 7A 所示的流程图，在块 222 中检测到第一线程的第一事件之后，事件检测器 188 在块 270 停止第一线程的退役，并断言“预彻底清除”信号 169。断言预彻底清除信号 169 以避免死锁情况，在死锁情况下，第一线程支配指令流水线来禁止第二线程。具体地说，如果第二线程被禁止访问指令流水线，则不可能出现有关第二线程的条件，其中要求所述条件开始多线程彻底清除操作。因此，预彻底清除信号 169 被传送给处理器的前端，具体地说是传送给存储器执行单元 42，以便使构成检测到事件的第一线程的微指令的处理器流水线饥饿。仅仅是举例，通过禁止由存储器执行单元 42 或前端其它组件执行的指令预提取及自修改代码（SMC）操作，可以执行使处理器流水线饥饿的行为。总之，通过停止第一线程的微指令退役，和/或通过暂停或大量减少利用第一线程向处理器流水线中馈送微指令，给予第二线程在处理器中的优先权，并减少死锁情况的可能性。

在判定框 272 作出以下判定，第二线程在多线程处理器 30 中是否为活动，并因此由重新排序缓冲器 162 退役。如果第二线程不是活动的，则方法 220 直接进行到块 274，在该块中执行称作“彻底清除操作”的第一类型清除操作。可以参考退役控制电路 168 维护的活动线程状态机 171 来执行特定线程是活动的还是非活动的判定。彻底清除操作以断言彻底清除信号 170 开始，其中，如上所述，彻底清除信号 170 具有对多线程处理器 30 的有序前端和无序后端种的状态进行清除的作用。由于只有第一线程是活动的，所以不需要考虑彻底清除操作对目前可能存在于多线程处理器 30 中的任何其它线程的影响。

另一方面，如果在判定框 272 确定第二线程在多线程处理器 30 中是活动的，则方法 220 继续执行一系列操作，这些操作组成第二线程清除点（或彻底清除点）的检测，其中，在该清除点可以进行彻底清除操作，而对第二线程的负面影响降低。在检测清除点之后执行的彻底清除操作与块 274 中执行的操作相同，并因此清除多线程处理器 30 的状态（即：第一和第二线程的状态）。状态的清除包括在本说明书其它部分描述的微指令“排出”操作。在本申请中公开的一个示例性实施例中，在检测清除点之后执行的彻底清除操作不区分多线程处理器 30 中对第一线程所保持的状态和对第二线程所保持的状态。在候选实施例中，在检测清除点之后执行的彻底清除操作可能只清除单线程（即检测到事件的线程）的状态，其中，资源共享的重要程度出现在多线程处理器 30 中，并且这样的共享资源可以动态地被分区及取消分区，以便为多线程提供服务，单线程状态的清除特别复杂。但是，该候选实施例可能要求更为复杂的硬件。

在判定框 272 的肯定确定之后，在判定框 278 还要对第二线程是否已经遇到事件作出判定。这样的事件可包括上述除 VIRTUAL NUKE 事件 260 之外的任何事件。这个确定也是由事件检测器 188 对第二线程的事件信号 266 或故障信息信号 269 作出响应来作出。

与第二线程所遇到的任何事件有关的信息储存在第二线程专用的事件信息寄存器 202 的部分中，并且该事件发生在未决事件寄存器 204 中进行登记。

如果第二线程单独遇到事件，则所述方法直接进行到块 280，在该块中执行多线程彻底清除操作，以便清除多线程处理器 30 的状态。另一方面，如果第二线程没有遇到事件，则在判定框 282 作出判定：第一线程所遇到的第一事件是否要求修改共享状态或共享资源，以便处理第一事件。例如，在第一事件包含如上所述的 SYNCHRONIZATION 事件 262，这表示第一线程要求访问共享状态资源。可以通过使具有相关的共享资源和同步流程标记符 184 和 186 的第一线程同步微指令退役，来标识 SYNCHRONIZATION 事件 262。图 10 是方框图，与图 9 所示类似，说明重新排序表 180 的示例性内容。所示分配给第一线程（例如线程 0）的表 180 的部分包括由退役指针 182 引用的同步微指令。所示同步微指令还具有相关的共享资源流程标记符 184 和同步流程标记符 186。所示同步微指令的退役将作为 SYNCHRONIZATION 事件 262 的发生而由事件检测器 188 来登记。

如果确定第一线程（例如线程 0）的第一事件不修改共享状态或资源，则方法 220 进入判定框 284，在该框中作出判定：第二线程（例如线程 1）是否在使具有相关的共享资源流程标记符 184 的微指令退役。参照图 9，所示线程 1 的退役指针 182 引用具有共享资源流程标记符 184 和同步流程标记符 186 的微指令。在这种情况下，在判定框 284 中给出的条件已经完成，因此，方法 220 进行到块 280，在该块中执行多线程彻底清除操作。另一方面，如果第二线程（例如线程 1）的退役指针 182 没有引用具有共享资源流程标记符 184 或同步流程标记符 186 的微指令，则所述方法进行到块 286，在该块中，通过增加退役指针 182 来继续第二线程的退役。方法 220 从块 286 返回判定框 278，在该框中再次作出判定：第二线程是否遇到事件。

如果在判定框 282 确定第一线程（例如线程 0）的第一事件的处理要求修改共享状态资源，则方法 220 进入判定框 288，在该框中作出判定：第二线程（例如线程 1）是否在使具有相关的同步流程标记符 186 的微指令退役。如果是的话，则在块 280 执行多线程彻底清除操作。如果不是，则在块 286 继续使第二线程的微指令退役，直到第二线程遇到事件或第二线程的退役指针 182 指向具有相关的同步流程标记符 186 的微指令。

在块 280 开始了彻底清除操作之后，在块 290 中，以微码实现并从微码定序器 66 定序的适当事件处理程序 67 继续处理相关事件。

虚拟彻底清除事件

如上所述，VIRTUAL NUKE 事件 260 以与其它事件略有不同的方式被处理。为此，图 7B 是说明根据一个示例性实施例的检测和处理 VIRTUAL NUKE 事件 260 的方法 291 的流程图。方法 291 假定当前第二线程没有事件为未决（即：记录在第二线程的未决寄存器中）。

方法 291 在块 292 以第一线程的第一事件的事件检测器 188 进行的检测开始。这样的事件可以是前面参照图 8 所述的任何一个事件。

在块 293，事件检测器 188 停止第一线程的退役。在块 294，事件检测器 188 检测到具有共享资源流程标记符 184 或同步流程标记符的微指令的退役。在块 295，“虚拟彻底清除”处理程序从微码定序器 66 中被调用。在块 296 中，“虚拟彻底清除”事件处理程序在前面于块 294 中退役的微指令之后的微指令处重新开始执行第二线程。方法 291 则在块 297 结束。

彻底清除操作

图 11A 是流程图，说明根据一个示例性实施例的一种方法 300，用于执行至少支持第一和第二线程的多线程处理器中的清除（或彻底清除）操作。方法 300 在块 302 以事件检测器 188 对事件的发生和检测作出响应而进行的彻底清除信号 170 的断言开始。彻底清除

信号 170 被传送给多线程处理器 30 中的多个功能单元，并且，对彻底清除信号 170 进行的断言和取消断言定义了一个窗口，在该窗口中，执行准备状态清除和功能单元的配置的活动。图 12 是时序图，说明与时钟信号 304 的上升沿产生同步的彻底清除信号 170 的断言。

在块 303，对活动线程状态机进行评估。

在块 306 中，第一、第二线程的序列号和最后的微指令信号被传送给分配及自由列表管理逻辑装置 122 和 TBIT，其中，最后的微指令信号表示事件产生的微指令是否退役，TBIT 是跟踪转移预测单元（TBPU）（又是 TDE 60 的一部分）中的结构，用于跟踪处理器 30 的有序前端中的宏指令和微指令指针信息。TBIT 利用该信息来锁存有关事件的信息（例如微指令和宏指令的指令指针）。

在块 308 中，事件检测器 188 建立第一和第二线程中每一个的事件矢量并将其传送给微码定序器 66。其中每个事件矢量包括信息，该信息标识（1）物理重新排序缓冲器位置（即标识彻底清除点时各退役指针 182 的值），它在彻底清除点（或清除点）被定位时在退役，（2）事件处理程序标识符，它在构成事件处理程序 67 以处理检测事件的微码被定位的情况下标识微码定序器 66 中的位置，以及（3）线程标识符，用于标识第一或第二线程，以及（4）线程优先级位，它确定事件处理程序 67 相对于其它线程调用的事件处理程序的优先级。

在块 310 中，分配及自由列表管理逻辑装置 122 采用在块 306 传送的序列号来将影像寄存器别名表（影像 RAT）进行到检测到彻底清除点的点；并且在块 312，主寄存器别名表 120 的状态从影像寄存器别名表中被恢复。

在块 314，分配及自由列表管理逻辑装置 122 从自由列表管理器 134 恢复寄存器号（或“大理石”），并将所恢复的寄存器号指配给垃圾堆阵列 132，其中寄存器号又可以被分配。当所有适当的寄存器号都已从自由列表管理器 134 中恢复时，分配及自由列表管理逻辑装

置 122 还断言“恢复的”信号（未示出）。彻底清除信号 170 保持在断言状态中，直到从分配及自由列表管理逻辑装置 122 接收到该“恢复的”信号。

在块 316 中，第一和第二线程的所有“前辈（senior）”存储（即已经退役但还未更新存储器的存储）均采用存储提交逻辑装置（未示出）从存储顺序缓冲器中排出。

在块 320 中，事件检测器 188 在时钟信号 304 的上升沿取消断言彻底清除信号 170，如图 12 所示。要指出，彻底清除信号 170 保持在断言状态至少时钟信号 304 的三个时钟周期。但是，如果在断言彻底清除信号 170 之后来自分配及自由列表管理逻辑装置 122 的“恢复”信号还没有在时钟信号 304 最开始的两个时钟周期内被断言，事件检测器 188 则把彻底清除信号 170 的断言延长超出所述三个时钟周期。在一个实施例中，彻底清除信号 170 可以保持足够长的时间（例如三个时钟周期）以完成上述块 303、306 及 308。可以要求彻底清除信号 170 保持额外的周期，以完成块 310、312、314 及 316。为此，存储顺序缓冲器断言“存储缓冲器排出”信号，以便延长彻底清除信号的断言。

在块 322，多线程处理器 30 中的微码定序器 66 和其它功能单元检查活动线程状态机 171 保持的“活动位”，以便确定第一和第二线程它们每个在事件发生之后是处于活动状态还是非活动状态。更具体地说，活动线程状态机 171 对存在于多线程处理器 30 中的各线程保持相应的位指示，所述位指示表示相关线程是处于活动状态还是处于非活动（睡眠）状态。由事件检测器 188 检测且事件检测器 188 对其作出响应而断言彻底清除信号 170 的事件可以包括 SLEEP 事件 264 或 BREAK 事件，将第一或第二线程在活动状态和非活动状态之间进行转换。如图 12 中的 324 所示，在断言彻底清除信号 170 期间对活动线程状态机 171 进行评估，因此，在取消断言彻底清除信号 170 时，“活动位”的状态看作是有效的。

在判定框 326 中，每个检查活动线程状态机 171 的活动位的功能单元作出判定：第一和第二线程是否都是活动的。如果根据活动位的状态确定两个线程都是活动的，则方法 300 进行到块 328，在该块中，将每个功能单元配置为支持第一和第二活动线程并为它们提供服务。例如，通过激活限于存储器阵列中的项目的特定集（或范围）的第二指针或第二组指针，可以对各种功能单元中提供的存储和缓存能力进行逻辑装置分区。此外，如果两个线程都是活动的，则可以激活某个 MT 特定的支持。例如，与微码定序器有关的线程选择逻辑装置可以根据活动线程状态机 171 的输出，以“乒乓”方式对来自第一线程（例如 T0）、来自第二线程（例如 T1）或来自第一和第二线程（例如 T0 和 T1）的线程进行排序。此外，可以根据活动线程状态机的位输出来执行局部时钟选通。在另一个实施例中，处理器中任意数量的状态机均可以根据活动线程状态机的输出来修改其行为或改变状态。在块 330，微码定序器 66 继续对第一和第二线程的微指令进行排序。

另一方面，如果在判定框 326 确定第一和第二线程中只有一个是活动的，或两个线程都是非活动的，则在块 332 将每个功能单元配置为仅支持单个活动线程并为其提供服务，并且可以对某个 MT 特定的支持进行去激活。在没有线程为活动的情况下，将功能单元作为缺省设置配置为支持单个活动线程。在功能单元以前被配置（例如逻辑装置分区）为支持多线程的情况下，可以禁止用来支持其它线程的指针，并且由剩余指针所引用的数据阵列中的项目集可以扩展为包括以前由禁止的指针所引用的项目。这样，可以知道，以前分配给其它线程的数据项目可以接着为单个活动线程所用。通过在其它线程为非活动状态时使较多的资源可以为单个活动线程所用，相对于多线程处理器 30 中还支持其它线程的情况，单个剩余线程的性能可以得到增强。

在块 334，微码定序器 66 忽略一个或多个非活动线程的事件矢

量，并仅对可能的活动线程的微指令进行排序。在没有任何线程是活动的情况下，微码定序器 66 忽略所有线程的事件矢量。

通过提供在彻底清除信号 170 的取消断言（发送信号通知彻底清除操作结束）时能由各种功能单元检查的活动线程状态机 171 保持的活动位，可以提供一个方便集中的指示，根据该指示，在彻底清除操作完成之后，各种功能单元可以配置为支持多线程处理器 30 中正确数量的活动线程。

图 11B 是方框图，说明示例性配置逻辑装置 329，它与功能单元 331 相关联，用来将功能单元 331 配置为支持多线程处理器中的一个或多个活动线程。功能单元 331 可以是上述任何一个功能单元，或者是本领域技术人员知道的包含在处理器中的任何功能单元。所示功能单元 331 包含由配置逻辑装置 329 进行配置的存储和逻辑装置组件。例如，存储组件可以包含寄存器的集合。当多线程是活动的时候（即当处理器以 MT 模式运行时），所述寄存器中的每一个可被分配给所述线程中特定一个线程的存储微指令或数据。因此，图 11B 所示存储组件经逻辑装置分区以支持第一和第二线程（例如 T0 和 T1）。当然，存储组件可以被分区以支持任意数量的活动线程。

所示逻辑装置组件包括 MT 逻辑装置，特别支持处理器中的多线程操作（即 MT 模式）。

所示配置逻辑装置 329 保持指针值 333，指针值 333 被输出到功能单元 331 的存储组件中。在一个示例性实施例中，这些指针值 333 用来对存储组件进行逻辑装置分区。例如，可以为每个活动线程生成单独的一对读和写指针值。每个线程的指针值的上、下限由配置逻辑装置 329 根据活动线程的数量来确定。例如，如果别的线程变为非活动的，可以由特定线程的一组指针值表示的寄存器的范围则可以增加，以覆盖以前分配给另一个线程的寄存器。

配置逻辑装置 329 还包括 MT 支持允许指示 335，MT 支持允许指示 335 被输出给功能单元的逻辑装置组件以允许或禁止功能逻辑

装置 331 的 MT 支持逻辑装置。

由活动线程状态机 174 输出的活动位 327 向配置逻辑装置提供输入，并由配置逻辑装置 329 用来生成值 333 的适当点及提供适当的 MT 支持允许输出。

事件处理程序进行的独占访问

某些事件处理程序（例如用于处理分页和同步事件的事件处理程序）要求独占访问多线程处理器 30，以便利用共享资源并修改共享状态。因此，微码定序器 66 实现独占访问状态机 69，为需要独占访问的第一和第二线程的这些事件处理程序提供独占访问。在多线程处理器 30 中一个以上的线程是活动时只可引用独占访问状态机 69。与被提供有独占访问的事件处理程序有关联的流程标记符被插入到线程的流程中，以便标记包括事件处理程序的独占码的结束。一旦对所有线程完成独占访问，微码定序器 66 就恢复正常发出微指令。

图 13 是流程图，说明根据一个示例性实施例的一种方法 400，用于为多线程处理器 30 中的事件处理程序 67 提供独占访问。方法 400 在块 402 以相应第一和第二线程的第一和第二事件矢量的微码定序器 66 从事件检测器 188 进行接收开始。如上所述，每个第一和第二事件矢量将标识相应的事件处理程序 67。

在判定框 403，对于是否超过一（1）个以上的线程是活动的作出判定。该确定由微指令定序器参考活动线程状态机 171 来进行。如果不是，则方法 400 进行到块 434。如果是，则方法 400 进入判定框 404。

在判定框 404，微码定序器 66 作出确定：第一或第二事件处理程序 67 是否要求独占访问共享资源或修改共享状态。如果是，则在块 406 中，微码定序器 66 实现独占访问状态机 69，以便为第一和第二事件处理程序 67 中的每一个提供独占访问。图 14 是状态图，说明根据示例性实施例的独占访问状态机 69 的操作。所示状态机 69

包括五种状态。在第一状态 408，由微码定序器 66 发出第一和第二线程的微码。在对要求独占访问事件处理程序的事件作出响应而发生彻底清除操作 410 时，状态机 69 转换到第二状态 412，其中发出与第一线程的事件有关的第一事件处理程序 67（即微指令）。在构成第一事件处理程序 67 的所有微指令的排序之后，以及在这样的微指令指示的所有操作完成之后，微码定序器 66 则在 414 发出停止微指令（例如具有相关的停止流程标记符的微指令），以便将状态机 69 从第二状态 412 转换到第三状态 416，其中停止发出第一线程微指令。在 418，从重新排序缓冲器 162 中使在 414 发出的停止微指令退役，从而将状态机 69 从第三状态 416 转换到第四状态 420，其中微码定序器 66 发出与第二线程的事件有关的第二事件处理程序 67。在构成第二事件处理程序 67 的所有微指令的排序之后，以及在这样的微指令指示的所有操作完成之后，微码定序器 66 则在 422 发出另一个停止微指令，以便将状态机 69 从第四状态转换到第五状态 424，其中停止第二事件处理程序 67。在 426，从重新排序缓冲器 162 使在 422 发出的停止微指令退役，从而将状态机 69 从第五状态 424 返回第一状态 408。

在块 432，假定第一和第二线程都是活动的，则恢复这两个线程的微指令的正常排序和发出。

另一方面，如果在判定框 404 确定第一和第二事件处理程序都没有要求独占访问处理器 30 的共享资源或状态，则该方法进行到块 434，在该块中，微码定序器 66 以非独占的交错方式对构成第一和第二事件处理程序 67 的微码进行排序。

活动线程状态机（171）

图 15 是状态图 500，说明根据一个示例性实施例的可由活动线程状态机 171 占用的各种状态，并说明根据一个示例性实施例的转换事件，它们可以使活动线程状态机 171 在各种状态之间进行转换。

所示活动线程状态机 171 处在四种状态其中之一，即单线程 0

（ST0）状态 502、单线程 1（ST1）状态 504、多线程（MT）状态 506 及零线程（ZT）状态 508。活动线程状态机 171 为每个线程保持单一活动位，当该位被设置时，它将有关线程标识为活动，而当该位被复位时，它将有关线程标识为非活动或睡眠。

四种状态 502-508 之间的转换由事件对来触发，其中事件对中的每个事件均与第一或第二线程有关。在状态图 500 中，多个事件类型被表示为用于各种状态之间的转换。具体地说，SLEEP 事件是使线程变为非活动的一种事件。BREAK 事件是一种事件，当针对特定线程出现时，它使该线程从非活动状态转换到活动状态。特定事件是否够格为 BREAK 事件将取决于使线程变为非活动状态的 SLEEP 事件。具体地说，只有某些事件会使由于特定 SLEEP 事件而曾为非活动的线程变为活动的。NUKE 事件是在针对特定线程出现时会导致执行彻底清除操作的任何事件，如上所述。以上参照图 8 所述的所有事件均可能包含彻底清除事件。最后，在状态图 500 中还说明了关于特定线程的“无事件”发生，作为可能与关于另一个线程的事件发生共同出现以使状态转换的条件。

在一个实施例中，如果对于特定线程发出了 SLEEP 事件信号通知，并且该线程的 BREAK 事件为未决，则立即对该 BREAK 事件提供服务（例如，该线程不会进入睡眠状态且稍后唤醒以为 BREAK 事件提供服务）。反之也成立：可以对特定线程发出 BREAK 事件的信号通知，并且 SLEEP 事件为未决，随后则对 BREAK 事件提供服务。

在由事件检测器 188 断言彻底清除信号 170 时，对活动线程状态机 171 进行评估，如图 12 中的 324 所示。在取消断言彻底清除信号 170 之后，多线程处理器 30 中的所有功能单元均根据活动线程状态机 171 保持的活动位进行配置。具体地说，检验器、重放和退役单元（CRU）160 将基于活动位生成的信号传送给所有受到影响的功能单元，以便向这些功能单元指明有多少线程现存于多线程处理器中以及其中哪些线程是活动的。在断言彻底清除信号 170 之后，功

能单元的配置（例如分区或取消分区）通常在时钟信号 304 的一个时钟周期内完成。

线程退出和进入

本发明提出一种示例性机制，由此多线程处理器 30 中的线程可以进入和退出（例如变为活动或非活动），其中这种进入和退出出现在均匀序列（uniform sequence）中，而不管运行线程的数量，以及在多线程处理器 30 中没有其它线程是活动的或者是在运行的时候，给各种功能单元的时钟信号可以适度地停止。

如以上参照状态图 500 所述，对当前非活动线程的 BREAK 事件的检测作出响应而发生线程进入（或激活）。特定非活动线程的 BREAK 事件定义取决于相关线程处于非活动的原因。对当前活动线程的 SLEEP 事件作出响应而发生线程退出。SLEEP 事件的示例包括执行包含在活动线程中的暂停（HLT）指令、检测 SHUTDOWN 或 ERROR_SHUTDOWN（错误_关闭）条件或有关该活动线程的“等待 SIPI”（启动处理器间中断）条件。

图 16A 是流程图，说明根据本发明一个示例性实施例的一种方法 600，用于在检测到活动线程的 SLEEP 事件时退出该活动线程。方法 600 从块 602 开始，在该块中，保存活动线程的所有被要求状态，并且以前已分配给该活动线程的微指令的寄存器文件 124 中的所有寄存器项目均被取消分配。仅仅是举例，在寄存器文件 124 的 128 个寄存器项目中，以前分配给活动线程的微指令的 28 个项目被取消分配。活动线程的取消分配的寄存器的内容保存在“暂存区”中，其中可以包括与多线程处理器 30 中的控制寄存器总线相连接的寄存器阵列或随机存取存储器（RAM）。

寄存器文件 124 中寄存器项目的取消分配可以由取消分配微码序列来进行，其中取消分配微码序列是由微码定序器 66 对活动线程的 STOPCLK、HALT（HLT）或 SHUTDOWH 事件的检测作出响应而发出的。取消分配微码序列用来删除（或使无效）自由列表管理

器 134 中寄存器文件项目的记录，并创建（或使有效）垃圾堆阵列 132 中寄存器文件项目的记录。换句话说，取消分配寄存器文件项目的记录由取消分配微码序列从自由列表管理器 134 传送到垃圾堆阵列 132。

图 16B 是可能在块 602 中执行的操作的示例性实施例的图示。例如，所示以前分配给第一线程（例如 T0）的寄存器文件 124 中第一组寄存器的内容被传送到暂存区。保存状态中可能执行的附加操作包括向暂存区储存现有线程的体系结构寄存器的内容，以及在退出第一线程时向暂存区储存分配给该第一线程的微码临时寄存器的内容。在退出线程时空出的寄存器则可用于重新分配给另一个线程（例如 T1）。

在重新进入特定线程（例如 T0）时，将会知道，分配给该线程的寄存器的内容可以从暂存区中进行恢复，如图 16B 中的虚线所示。

在块 604 中，退出线程的线程特定“电子篱笆微指令”被插入到该退出线程的微指令流程中，以便从存储顺序缓冲器 48、各种高速缓存以及处理器总线中排出与该线程有关的任何剩余未决存储器存取。直到所有这些块完成，该操作才退役。

由于这些执行单元 20 相对快地执行微指令，所以添加到执行单元输入的所有新微指令均通过对 SLEEP 事件的检测作出响应的彻底清除信号的断言来进行清除。如上所述，彻底清除信号 170 保持足够的一段时间（例如三个时钟周期），以便允许在断言彻底清除信号 170 之前进入执行单元 70 的微指令从执行单元 70 出现。由于这些微指令从执行单元 70 出现，所以它们被清除并且回写被取消。

在块 606，通过微指令把保持在事件检测器 188 中的展开寄存器 208 设置为表示现存的线程处于非活动（或睡眠）状态，其中所述微指令由微码定序器 66 所生成，它回写一个设置展开寄存器的状态的值。

在块 608，通过由微码定序器 66 发出的控制寄存器写入微指令，

将现存线程的事件禁止寄存器 206 设置为禁止现存线程的非中断事件。现存线程的事件禁止寄存器的设置，作为控制寄存器微指令被指示，取决于被服务的睡眠事件的类型。如上所述，根据触发向非活动级转换的 SLEEP 事件，只有某些事件取得关于非活动线程的中断事件的资格。参照非活动线程的事件禁止寄存器 206 的状态，确定事件是否取得特定非活动线程的中断事件的资格。

在块 612，采用特殊微指令来对现存线程的睡眠事件发出信号通知，其中所述特殊指令把睡眠事件编码放置在所述特殊微指令的回写故障信息字段中。

图 17 是流程图，说明根据一个示例性实施例的一种方法 700，用于在检测到非活动线程的 BREAK 事件时使非活动线程进入活动状态。方法 700 在 702 以检测事件的事件发生开始，其中该事件可能取得关于非活动线程的 BREAK 事件的资格或者可能不。在判定框 703，由相关事件的事件检测逻辑装置 185 作出确定，以便确定所述事件是否取得非活动线程的 BREAK 事件的资格。为此，事件检测逻辑装置 185 检查事件检测器 188 的寄存器 200 中的事件禁止寄存器 206。如果相关事件类型未被表示为关于非活动线程的禁止 BREAK 事件，则方法 700 进行到块 704，在该块中，根据需要接通时钟，正常地就所述事件发出信号通知（等待其它线程的可彻底清除点），并且关于任何事件，对处理程序进行调用。事件处理程序检查线程睡眠状态，如果设置，则继续，在块 706 恢复微码状态。事件处理程序 67 通过访问展开寄存器 208 来确认线程的非活动状态。

更具体地说，事件处理程序 67 通过恢复所有保存的寄存器状态、禁止寄存器状态及指令指针信息，进行到恢复进入线程的微码状态。

在块 706 中的微码状态恢复之后，方法 700 进行到块 708，在该块中，对进入线程恢复体系结构状态。在块 710，进入线程的事件禁止寄存器 206 由微码定序器 66 发出的适当微指令来进行复位或清除。在块 712，事件处理程序 67 继续为 BREAK 事件提供服务。此

时，在多线程处理器 30 中执行构成事件处理程序 67 的微码，以响应于事件发生而执行一系列操作。在块 716，则在处理器 30 中对进入线程再次恢复指令提取操作。然后，方法 700 在块 718 结束。

时钟控制逻辑装置

为了减少多线程处理器 30 中的功耗和散热，需要在某些条件下停止或挂起处理器 30 中的至少一些时钟信号。图 18 是流程图，说明根据一个示例性实施例的一种方法 800，用于停止或挂起诸如上述示例性处理器 30 的多线程处理器中被选时钟信号。为了说明起见，对处理器中时钟信号的挂起或停止进行引用，以包括挂起或停止处理器 30 中的一个或多个时钟信号的许多技术。例如，可挂起处理器 30 中的锁相环（PLL），禁止沿时钟中心（clock spine）的核心时钟信号的分配，或者选通或禁止经时钟中心向处理器中各功能单元分配时钟信号。一个实施例设想后一种情况，其中逐个功能单元地挂起或停止向处理器 30 的功能单元提供内部时钟信号。因此，内部时钟信号可以提供给某些功能单元，而关于其它功能单元进行选通。在美国专利 No. 5655127 中，描述了单线程微处理器环境下的这种布置。

在一个实施例中，图 18 所示方法 800 可以由时钟控制逻辑装置 35 来执行，其中时钟控制逻辑装置 35 结合在处理器 30 的总线接口单元 32 中。在其它实施例中，时钟控制逻辑装置 35 当然可以放置在处理器 30 以外的地方。图 19A 和 19B 分别是方框图和示意图，进一步详细说明示例性时钟控制逻辑装置 35。

首先来看图 19A，所示时钟控制逻辑装置 35 接收三个主要输入，即（1）经活动线程状态机 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 输出的窥探控制信号 822，其中总线接口单元 32 检测总线上可窥探的访问，并断言信号 882。时钟控制逻辑装置 35 采用这些输入来生成停止时钟信号 826，后者又抑制或禁

止处理器 30 中某些功能单元的时钟。

图 19B 是示意图，说明示例组合逻辑装置，它采用输入 211、820 以及 822 来输出停止时钟信号 826。具体地说，事件检测器信号 211 向“或”门 822 提供输入，“或”门 822 又将输入提供给另一个“或”门 824。活动位 820 和窥探控制信号 822 还将输入提供给“或非”门 824，后者对所述输入进行“或”运算，以便输出停止时钟信号 826。

现在具体地来看图 18，方法 800 在判定框 802 以确定多线程处理器 30 中是否有任何线程（例如第一和第二线程）是活动的为开始。该确定通过向图 19B 所示“或”门 824 输出活动位 820 来反映。虽然示例性实施例说明关于两个线程来作确定，但易于知道，可以关于多线程处理器所支持的任意数量的线程来作确定。

在判定框 802 的否定确定之后，方法 800 进入判定框 804，在该框中确定：对于多线程处理器所支持的任何线程，是否有任何未被禁止的事件为未决。在示例性实施例中，这还包括确定：对于第一或第二线程，是否有任何事件为未决。该确定通过向图 19B 所示“或”门 822 中输入事件检测信号 211 来表示。

在判定框 804 的否定判定之后，在判定框 806 作另一个确定：是否有任何窥探（例如总线窥探、SNC 窥探或其它窥探）正在由处理器总线进行处理。在本发明的示例性实施例中，该确定通过向“或”门 824 输入窥探控制信号 822 来实现。

在判定框 806 的否定确定之后，方法 800 进行到块 808，在该块中，停止或挂起给被选功能单元的内部时钟信号。具体地说，没有挂起或停止给总线未决逻辑装置和总线访问逻辑装置的时钟信号，因为这允许总线接口单元 32 来检测系统总线上发起的 BREAK 事件或窥探（例如管脚事件），以及对这样的 BREAK 事件作出响应而重新开始给功能单元的时钟。给功能单元的内部时钟信号的抑制通过断言停止时钟信号 826 来实现，其中停止时钟信号 826 具有对给预定功能单元的时钟信号进行选通的功能。

在完成块 808 之后，方法 800 返回到判定框 802。之后，可以连续循环判定框 802、804 和 806 中的确定。

在判定框 802、804 以及 806 其中任何一个肯定确定之后，方法 800 转移到块 810，在该块中，如果给某些功能单元的时钟信号已被选通，则再次激活这些内部时钟信号。另一方面，如果时钟信号已经是活动的，则这些时钟信号保持在活动状态。

在对中断事件进行响应而执行块 810 的情况下，（例如，在判定框 804 的肯定判定之后），在断言彻底清除信号时，微处理器中的功能单元可以根据活动线程的数量以上述方式活动地进行分区。例如，在具有两个或更多线程的多线程处理器 30 中，所述线程中的某些线程可能是非活动的，在这种情况，功能单元将不会进行分区来适应非活动线程。

在完成了块 810 时，方法 800 再次返回到判定框 802，并开始判定框 802、804 及 806 表示的判定的另一次重复。

已经说明了进入和退出多线程处理器中的多线程的方法和装置。虽然是参照特定示例性实施例对本发明进行说明，但显然，在不脱离本发明更宽范围和精神的情况下，可以对这些实施例进行各种修改和改变。因此，说明书和附图是说明性的而不是限制性的。

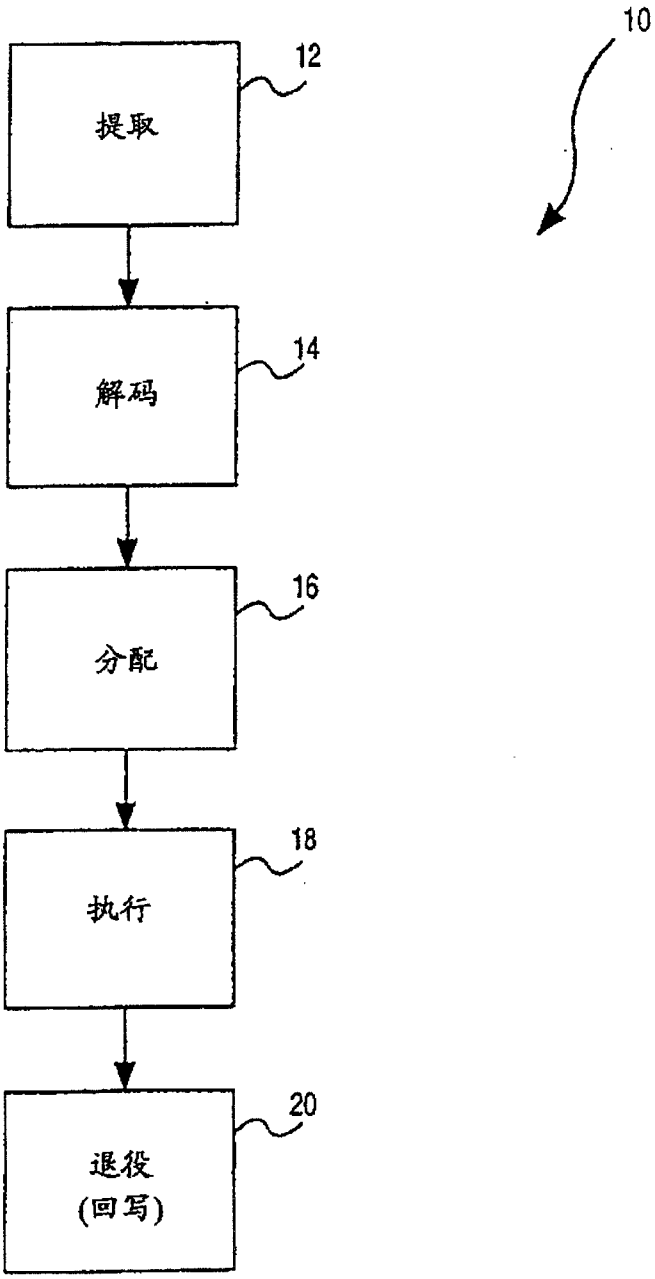


图 1

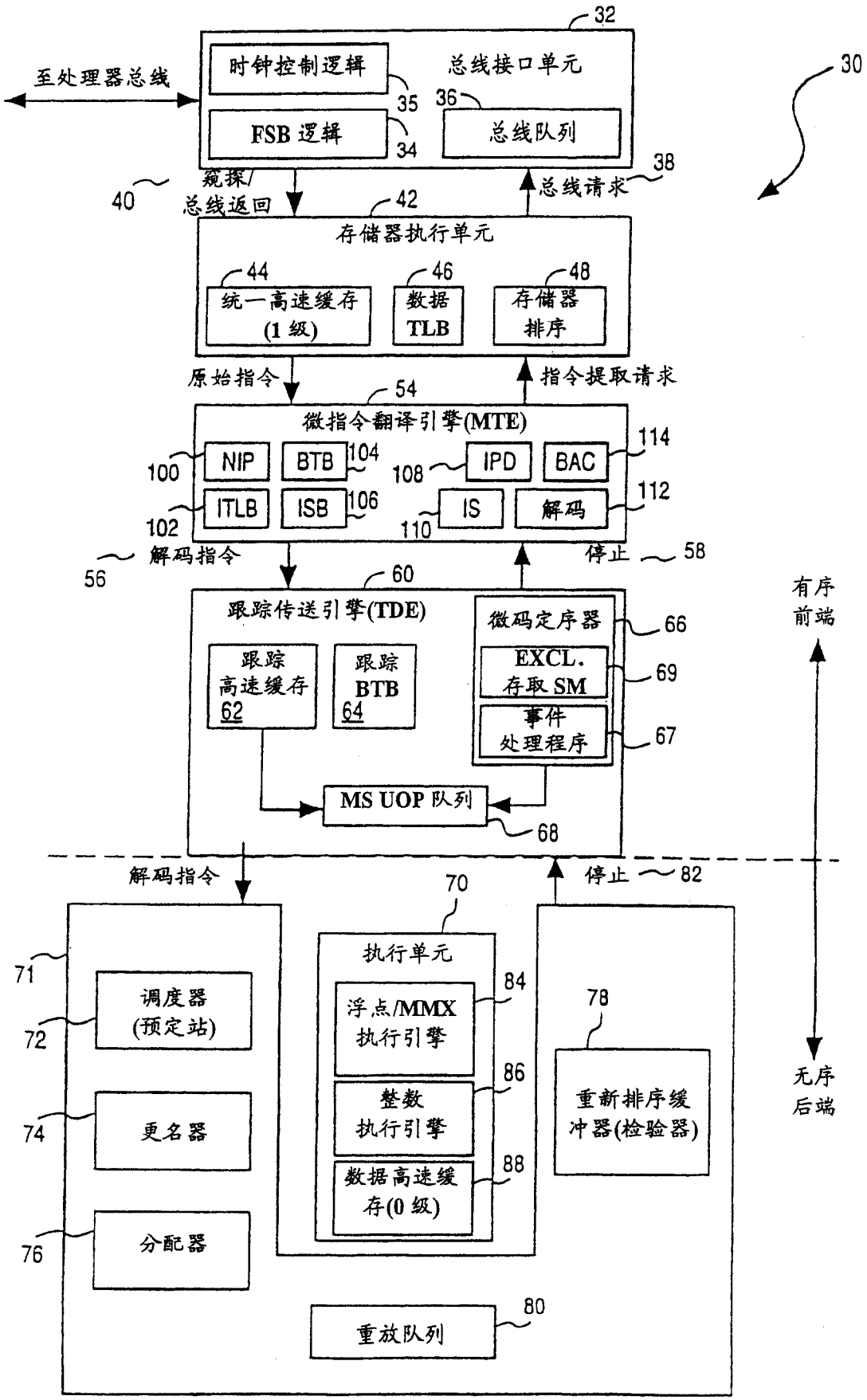

32
至处理器总线
时钟控制逻辑
总线接口单元
35
36
FSB 逻辑
34
总线队列
30
38
40
窥探/
总线返回
总线请求
42
存储器执行单元
44
46
48
统一高速缓存
(1 级)
数据
TLB
存储器
排序
原始指令
指令提取请求
54
微指令翻译引擎(MTE)
100
NIP
BTB
104
108
IPD
BAC
114
102
ITLB
ISB
106
110
IS
解码
112
解码指令
56
停止
58
60
跟踪传送引擎(TDE)
微码定序器
66
跟踪
高速缓存
62
跟踪
BTB
64
EXCL.
存取 SM
69
事件
处理程序
67
MS UOP 队列
68
有序
前端
解码指令
停止
82
70
执行单元
71
调度器
(预定站)
84
浮点/MMX
执行引擎
78
72
86
整数
执行引擎
重新排序缓
冲器(检验器)
74
更名器
88
数据高速缓
存(0 级)
76
分配器
80
重放队列
无序
后端

图 2

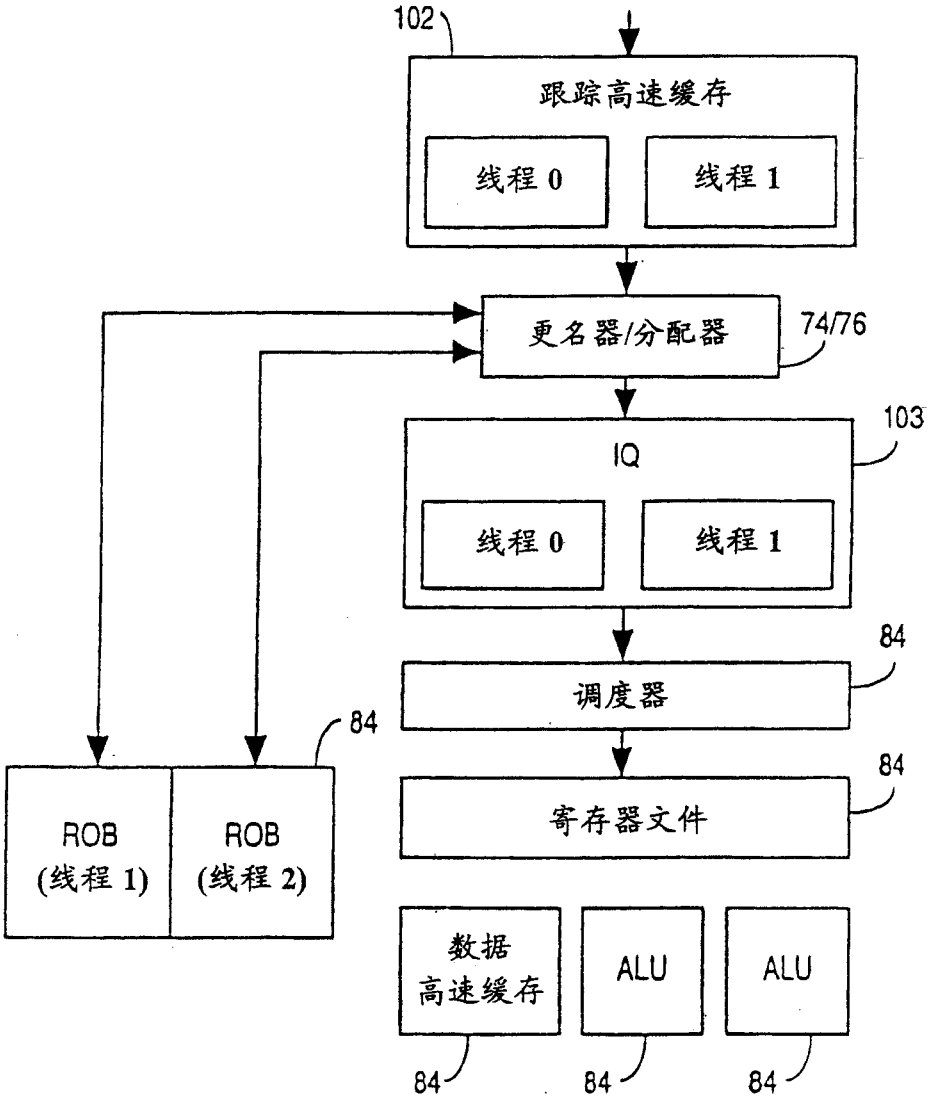

102
跟踪高速缓存
线程 0
线程 1
更名器/分配器
74/76
103
IQ
线程 0
线程 1
84
调度器
84
寄存器文件
84
ROB
(线程 1)
ROB
(线程 2)
数据
高速缓存
ALU
ALU
84
84
84

图 3

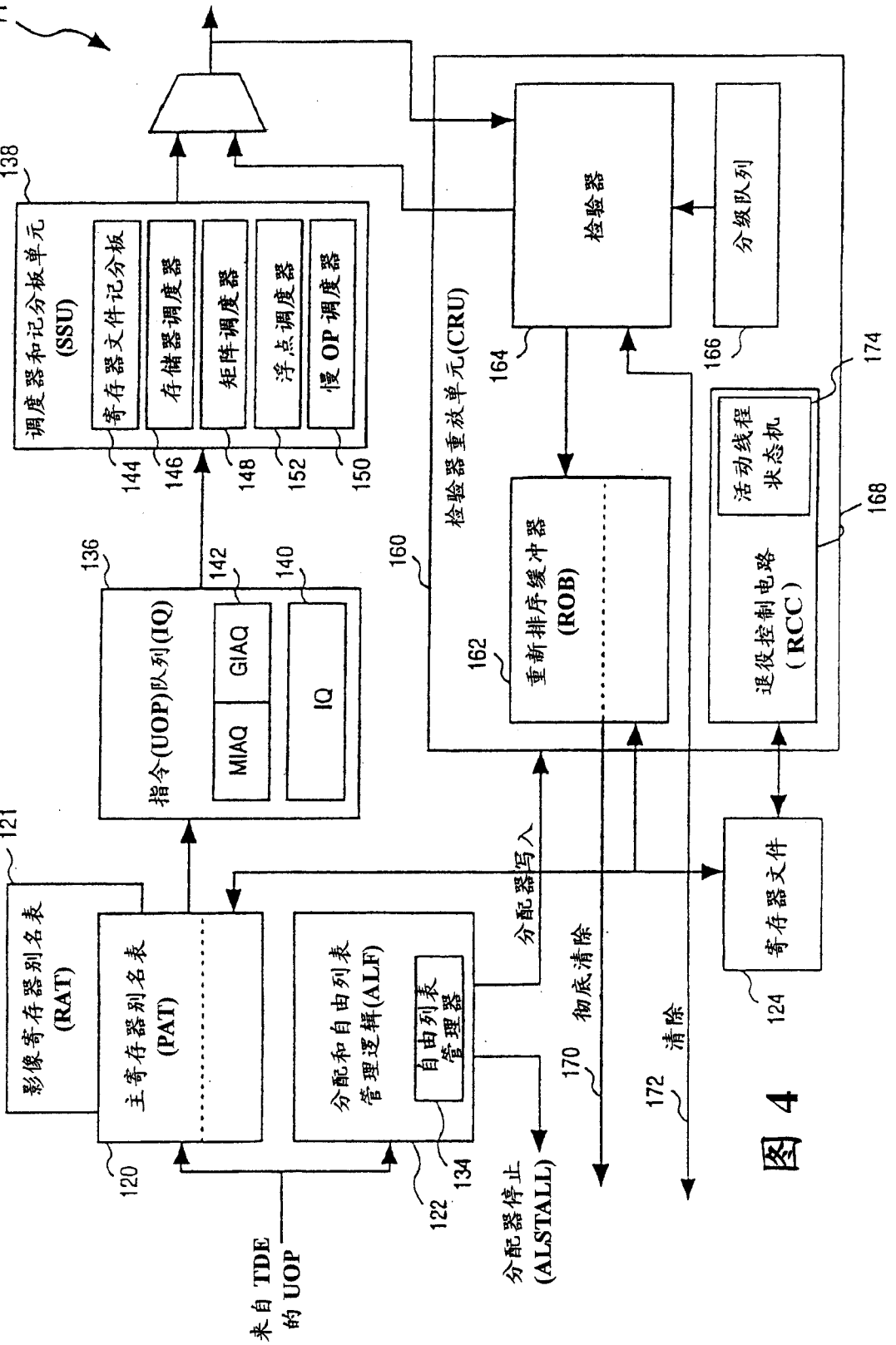

71
138
调度器和记分板单元
(SSU)
寄存器文件记分板
存储器调度器
矩阵调度器
浮点调度器
慢 OP 调度器
144
146
148
152
150
检验器重放单元(CRU)
160
检验器
164
分级队列
166
174
活动线程
状态机
168
退役控制电路
(RCC)
162
重新排序缓冲器
(ROB)
136
指令(UOP)队列(IQ)
142
140
GIAQ
MIAQ
IQ
121
影像寄存器别名表
(RAT)
主寄存器别名表
(PAT)
120
分配和自由列表
管理逻辑(ALF)
自由列表
管理器
122
134
寄存器文件
124
分配器写入
物质清除
170
清除
172
来自 TDE
的 UOP
分配器停止
(ALSTALL)

图 4

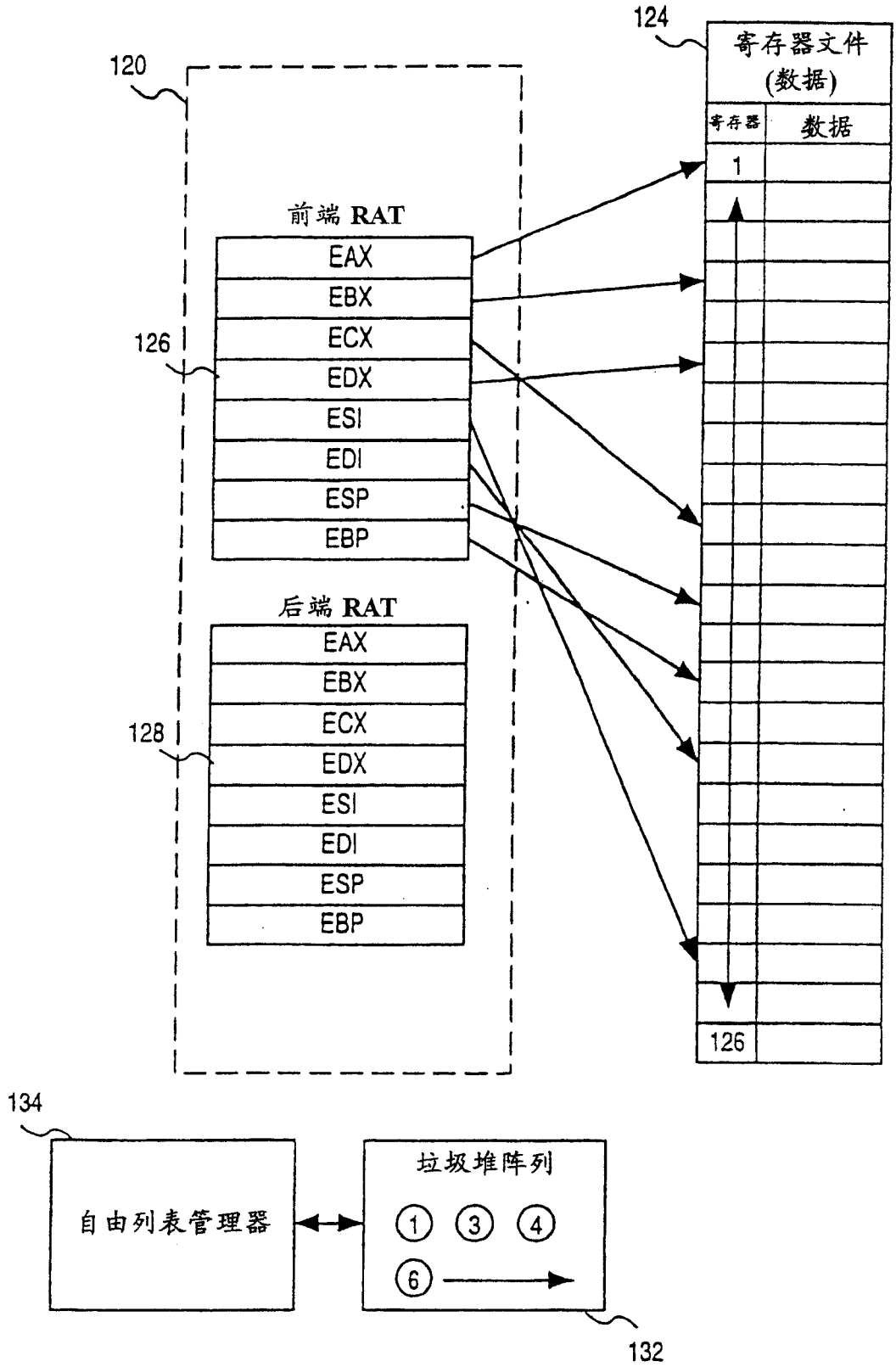
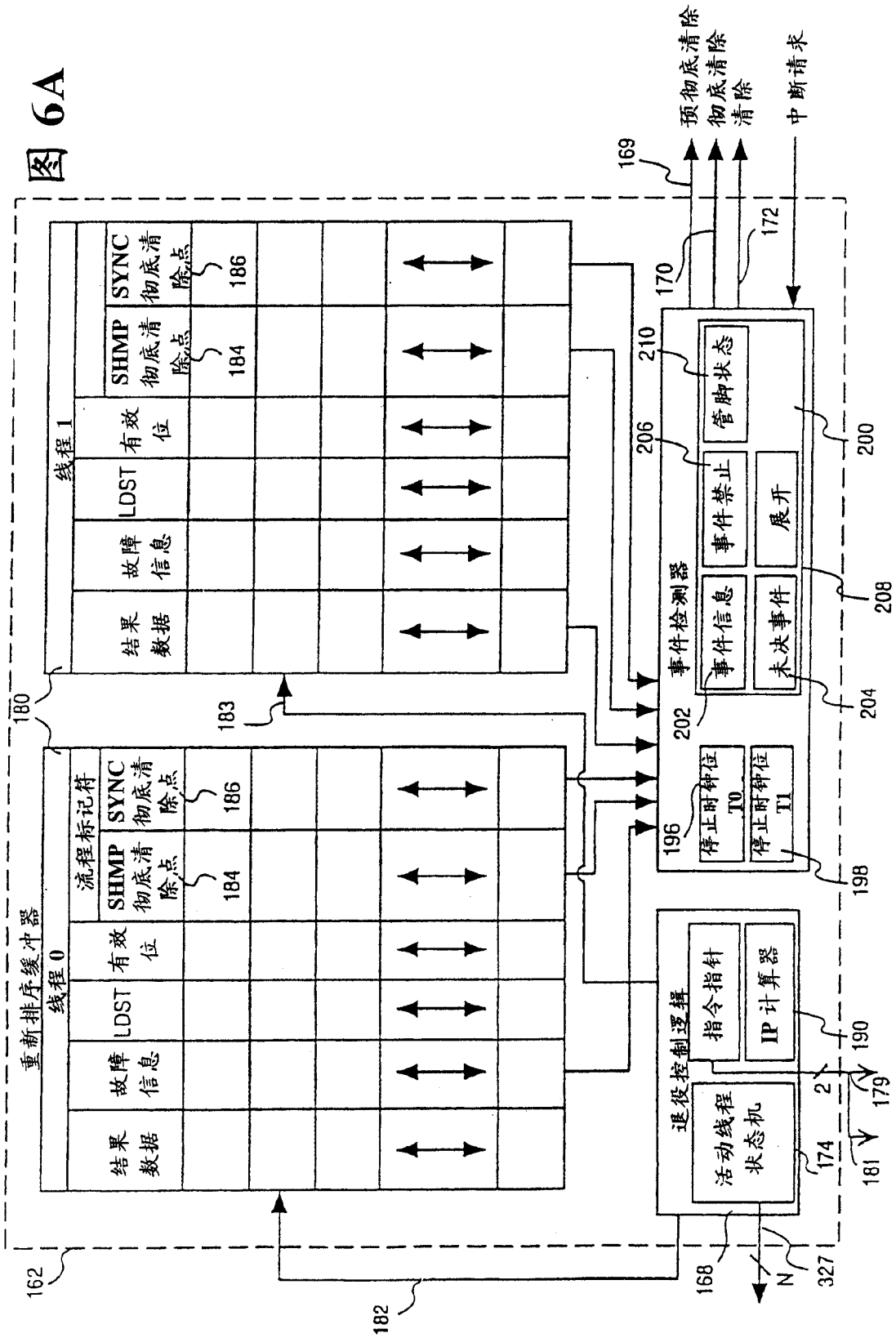

120
124
寄存器文件
(数据)
寄存器
数据
1
前端 RAT
EAX
EBX
ECX
EDX
ESI
EDI
ESP
EBP
126
后端 RAT
EAX
EBX
ECX
EDX
ESI
EDI
ESP
EBP
128
126
134
自由列表管理器
垃圾堆阵列
1
3
4
6
132

图 5

图 6A


重新排序缓冲器
线程 0
线程 1
结果数据
故障信息
LDST
有效位
SHMP 物底清除点
流程标记符
SYNC 物底清除点
184
186
180
183
162
182
168
N
327
169
预物底清除
物底清除
清除
中断请求
172
170
210
206
200
202
事件检测器
事件信息
事件禁止
管脚状态
未决事件
展开
208
204
196
停止时钟位 T0
停止时钟位 T1
198
退役控制逻辑
活动线程状态机
指令指针
IP 计算器
174
190
179
181
2

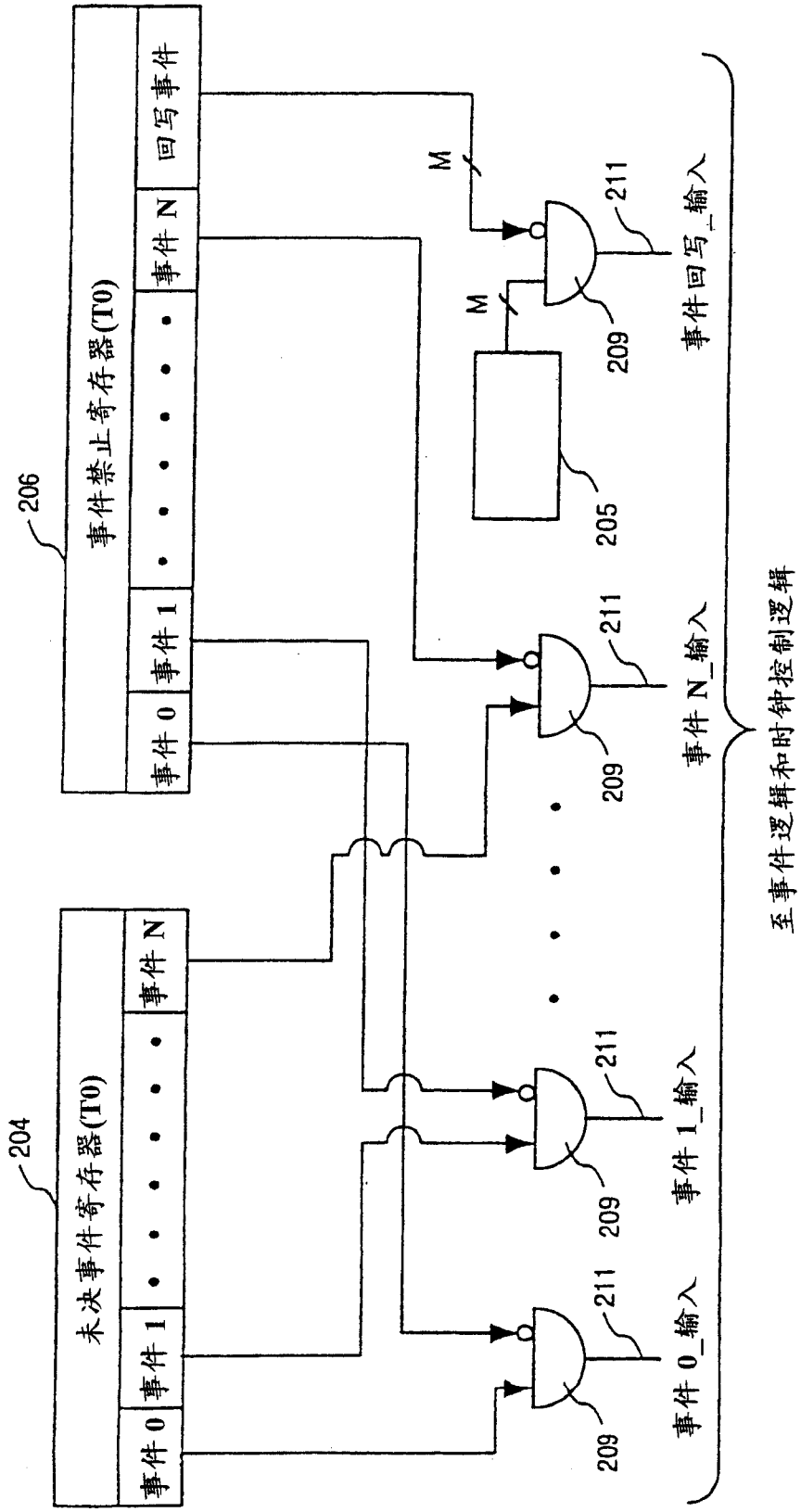

未决事件寄存器(T0)
事件 0
事件 1
事件 N
204
事件禁止寄存器(T0)
事件 0
事件 1
事件 N
回写事件
206
M
M
205
209
211
事件 0_输入
事件 1_输入
事件 N_输入
事件回写_输入
至事件逻辑和时钟控制逻辑

图 6B

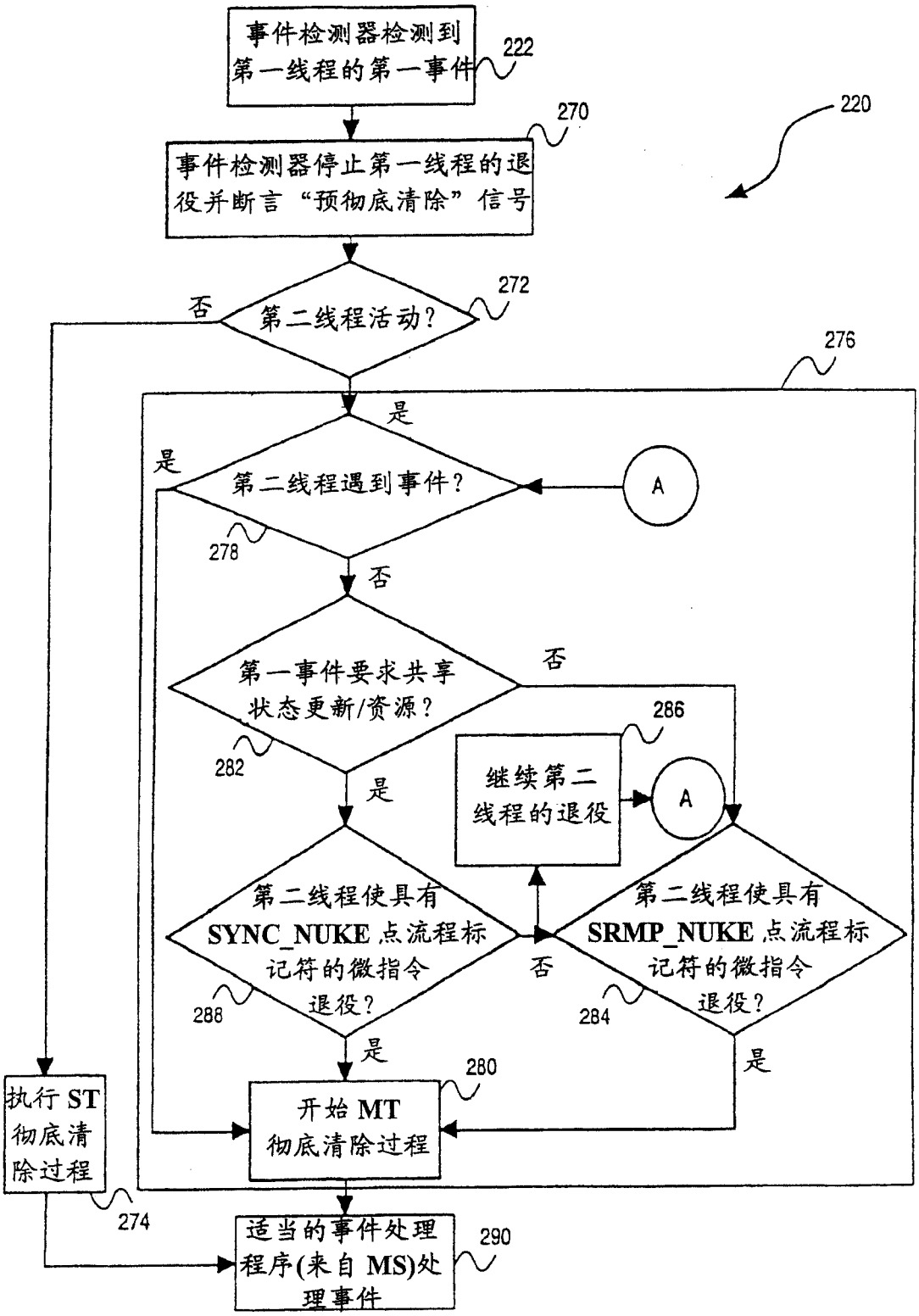

事件检测器检测到第一线程的第一事件
222
220
270
事件检测器停止第一线程的退役并断言“预彻底清除”信号
否
第二线程活动?
272
276
是
是
第二线程遇到事件?
A
278
否
第一事件要求共享状态更新/资源?
否
286
282
继续第二线程的退役
A
是
第二线程使具有SYNC_NUKE点流程标记符的微指令退役?
否
第二线程使具有SRMP_NUKE点流程标记符的微指令退役?
288
284
是
是
280
执行ST彻底清除过程
开始MT彻底清除过程
274
适当的事件处理程序(来自MS)处理事件
290

图 7A

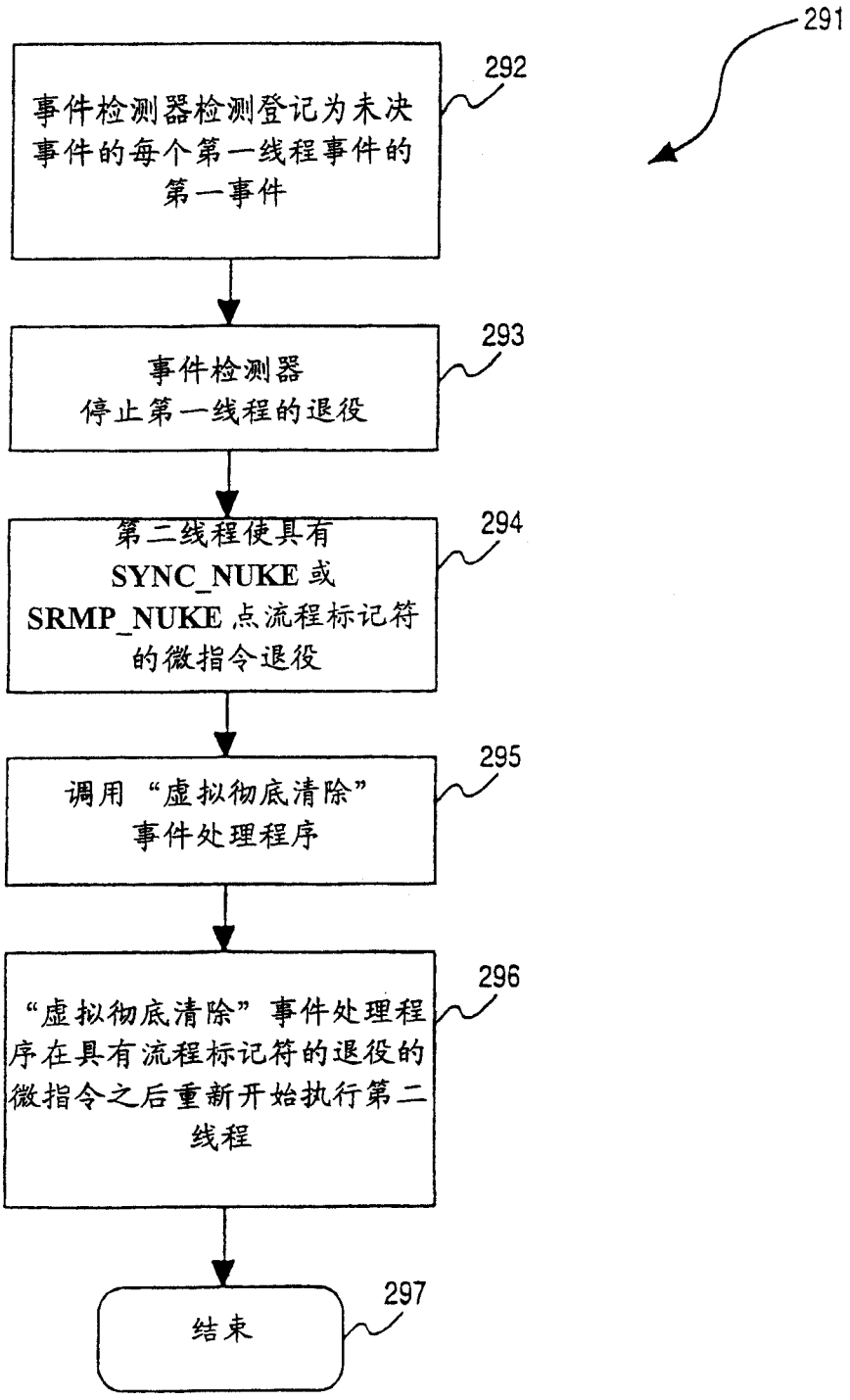

291
292
事件检测器检测登记为未决事件的每个第一线程事件的第一事件
293
事件检测器停止第一线程的退役
294
第二线程使具有SYNC_NUKE或SRMP_NUKE点流程标记符的微指令退役
295
调用“虚拟彻底清除”事件处理程序
296
“虚拟彻底清除”事件处理程序在具有流程标记符的退役的微指令之后重新开始执行第二线程
297
结束

图 7B

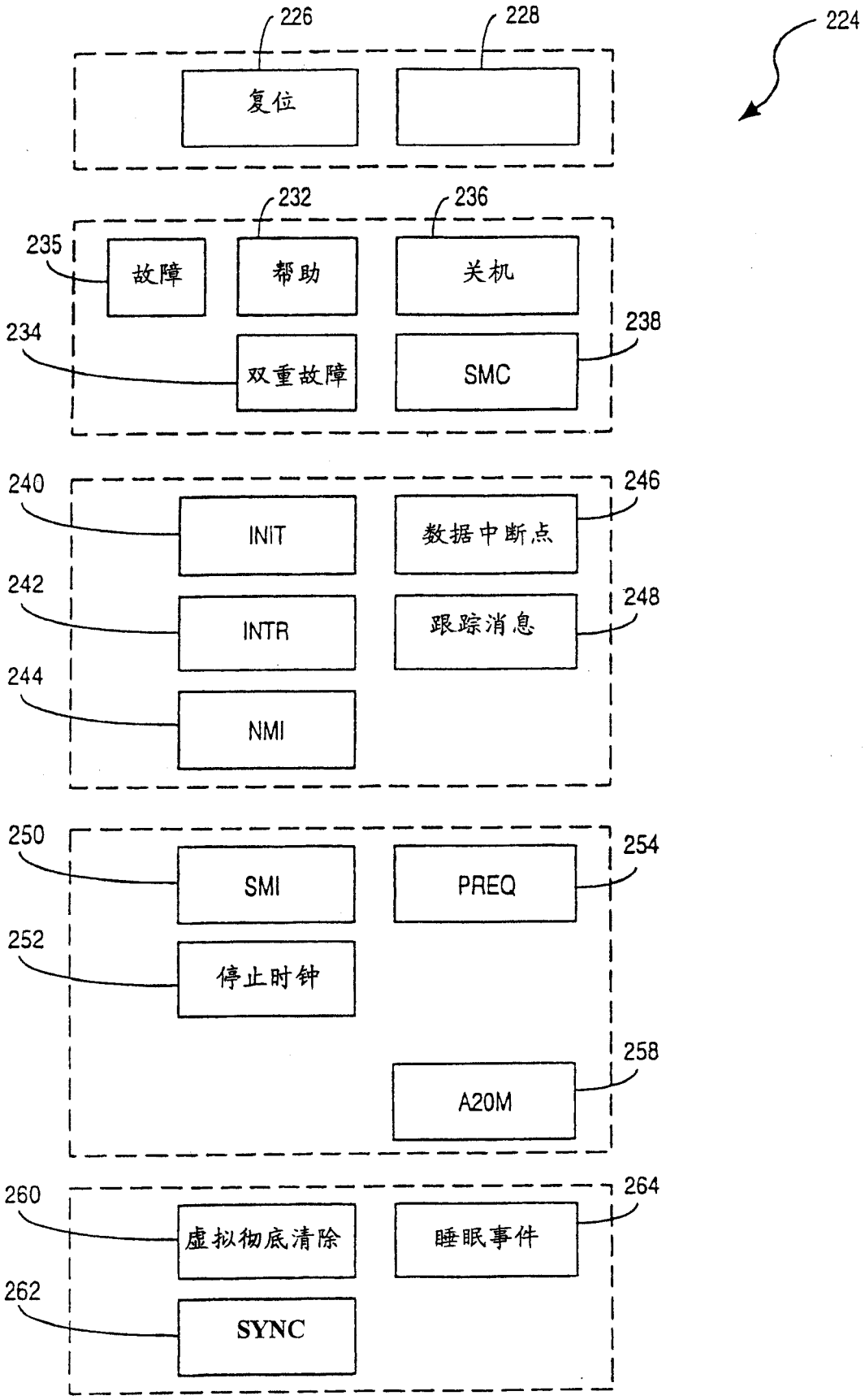

224
226
228
复位
232
236
235
故障
帮助
关机
238
234
双重故障
SMC
240
246
INIT
数据中断点
242
248
INTR
跟踪消息
244
NMI
250
254
SMI
PREQ
252
停止时钟
258
A20M
260
264
虚拟彻底清除
睡眠事件
262
SYNC

图 8

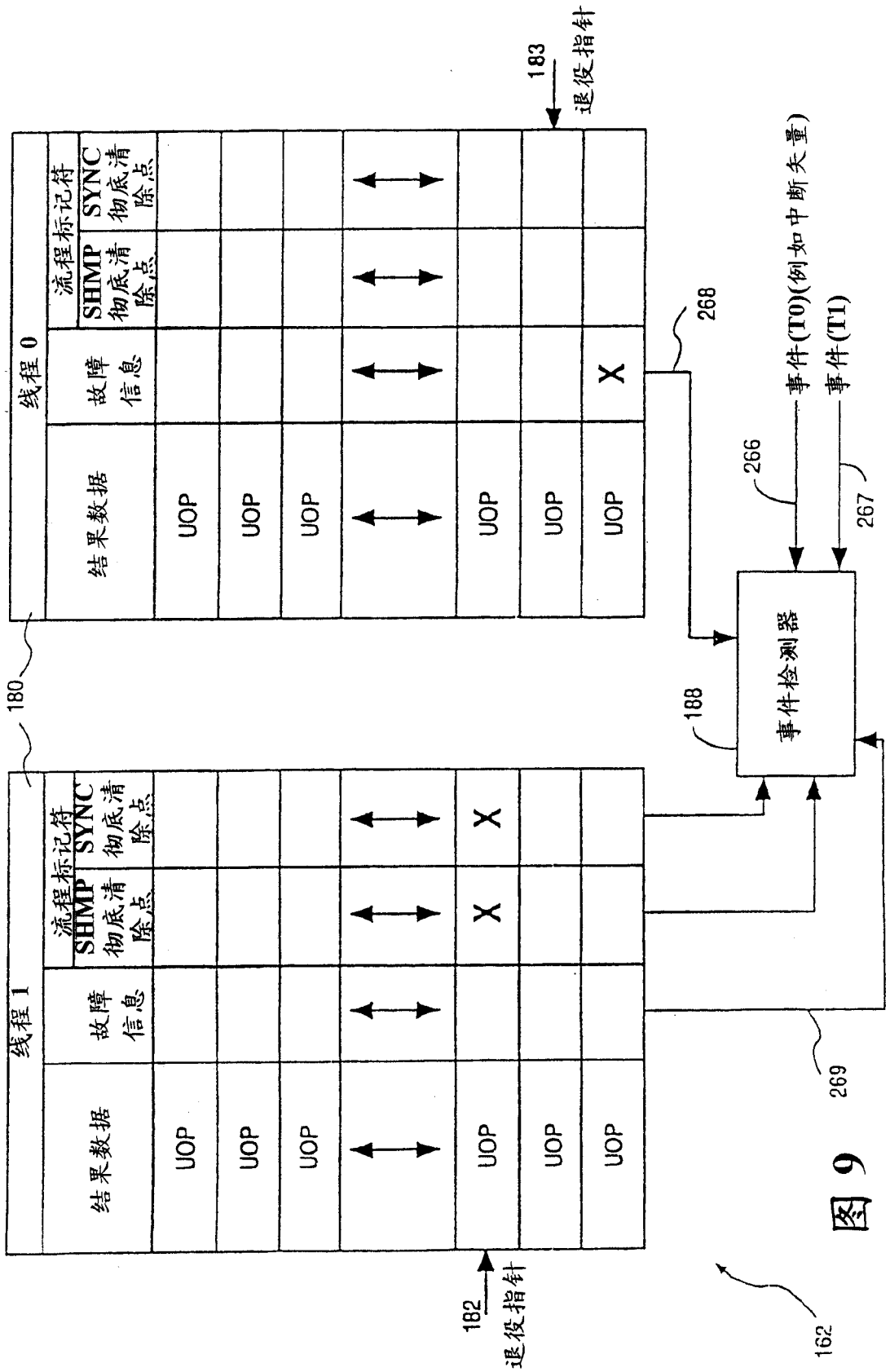

线程 0
结果数据
故障信息
流程标记符
SHMP 物底清除点
SYNC 物底清除点
UOP
X
180
183
退役指针
268
266
267
事件(T0)(例如中断矢量)
事件(T1)
188
事件检测器
269
线程 1
182
退役指针
162

图 9

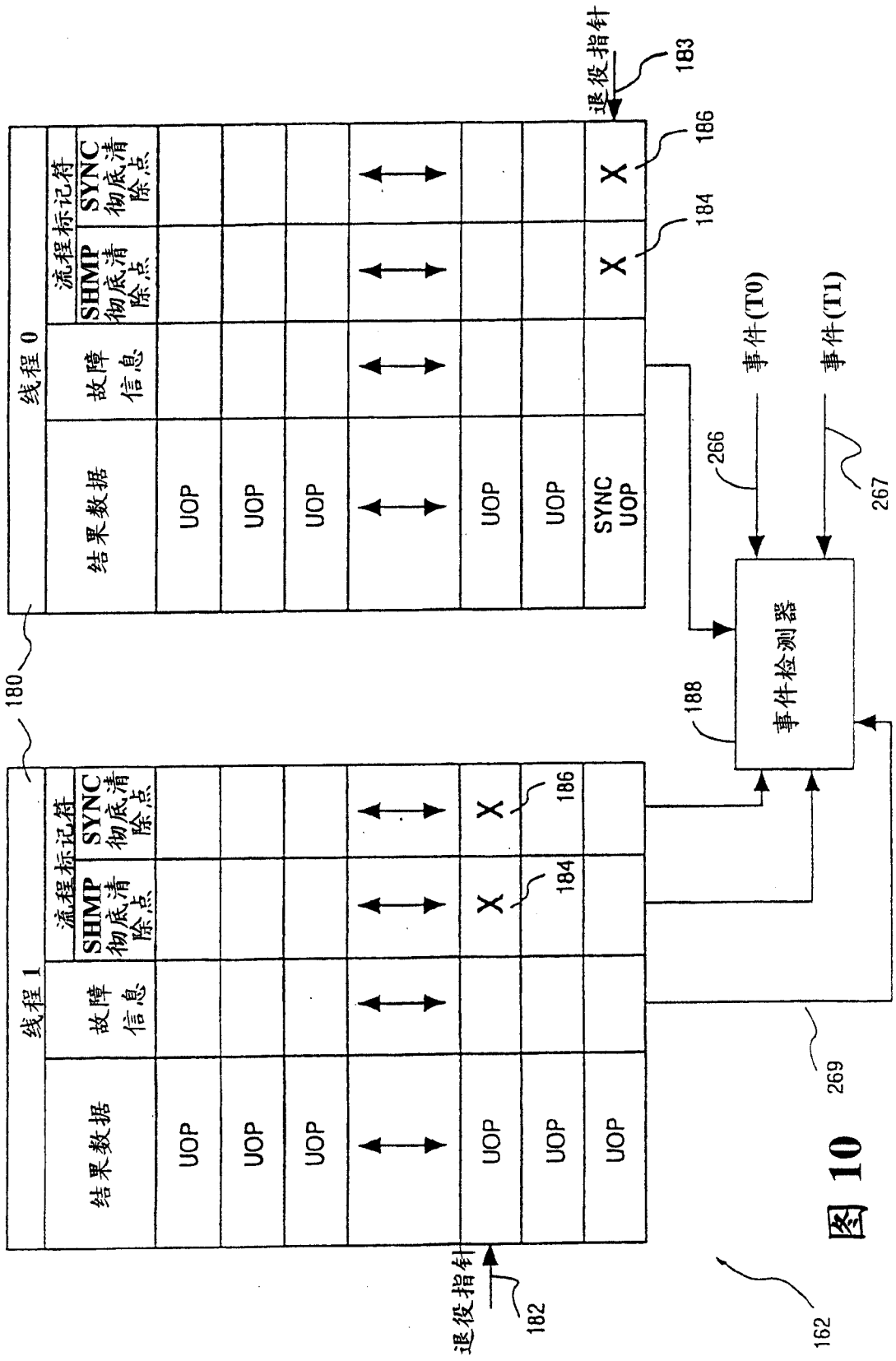

线程 0
结果数据
故障信息
流程标记符
SHMP 彻底清除点
SYNC 彻底清除点
UOP
SYNC UOP
X
退役指针
183
186
184
180
188
事件检测器
事件(T0)
事件(T1)
266
267
269
162
线程 1
退役指针
182

图 10

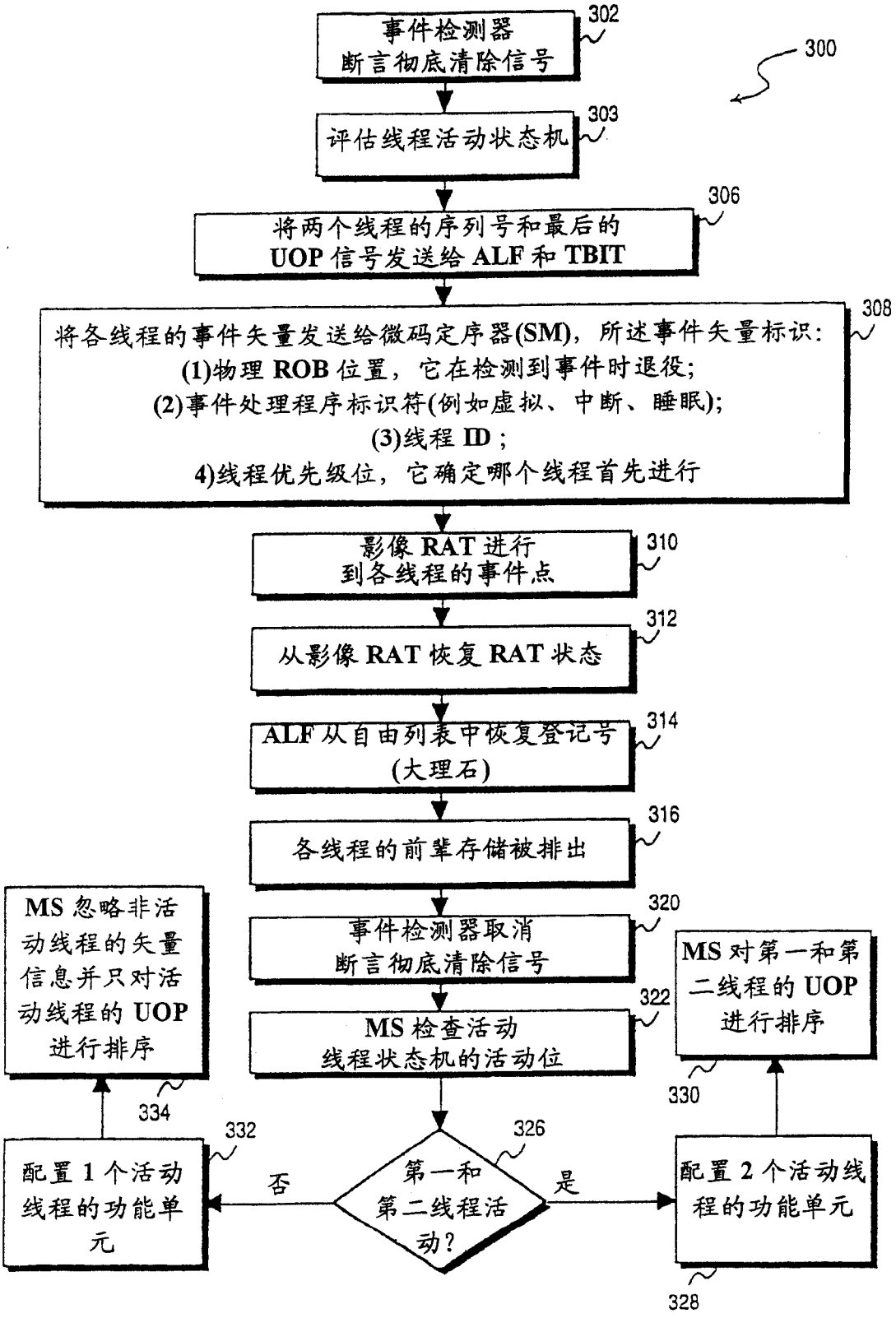

300
事件检测器
断言彻底清除信号
302
评估线程活动状态机
303
将两个线程的序列号和最后的
UOP 信号发送给 ALF 和 TBIT
306
将各线程的事件矢量发送给微码定序器(SM)，所述事件矢量标识：
(1)物理 ROB 位置，它在检测到事件时退役；
(2)事件处理程序标识符(例如虚拟、中断、睡眠)；
(3)线程 ID；
4)线程优先级位，它确定哪个线程首先进行
308
影像 RAT 进行
到各线程的事件点
310
从影像 RAT 恢复 RAT 状态
312
ALF 从自由列表中恢复登记号
(大理石)
314
各线程的前辈存储被排出
316
事件检测器取消
断言彻底清除信号
320
MS 检查活动
线程状态机的活动位
322
第一和
第二线程活
动?
326
否
是
MS 忽略非活
动线程的矢量
信息并只对活
动线程的 UOP
进行排序
334
配置 1 个活动
线程的功能单
元
332
MS 对第一和第
二线程的 UOP
进行排序
330
配置 2 个活动线
程的功能单元
328

图 11A

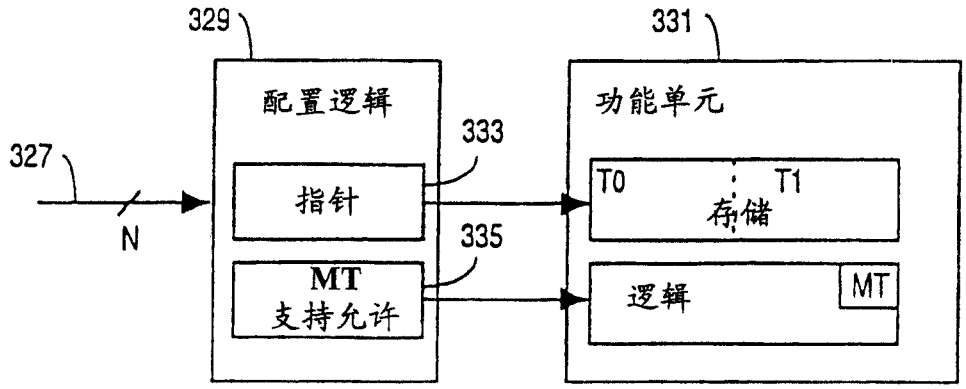


图 11B

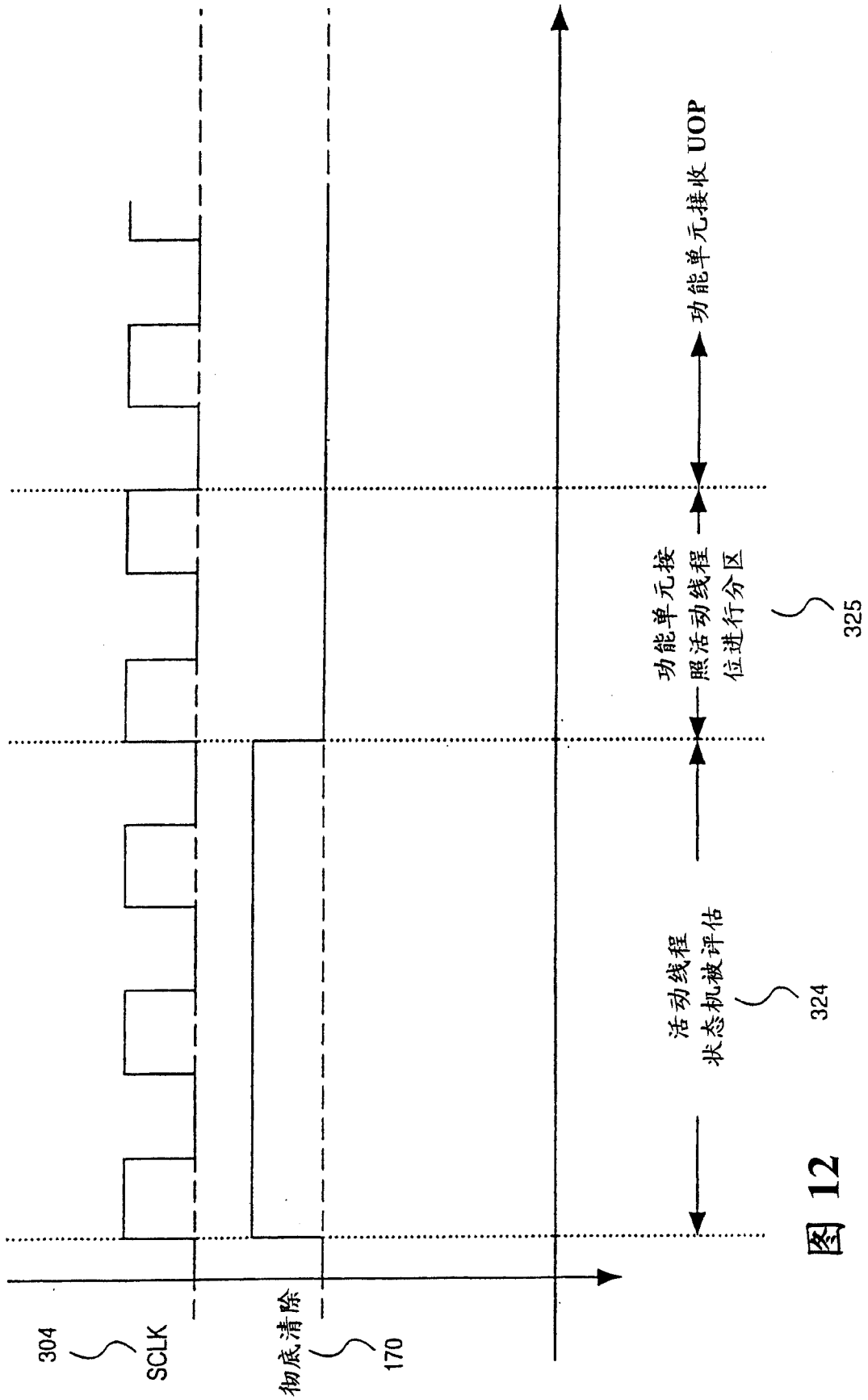

304
SCLK
彻底清除
170
活动线程
状态机被评估
324
功能单元按
照活动线程
位进行分区
325
功能单元接收 UOP

图 12

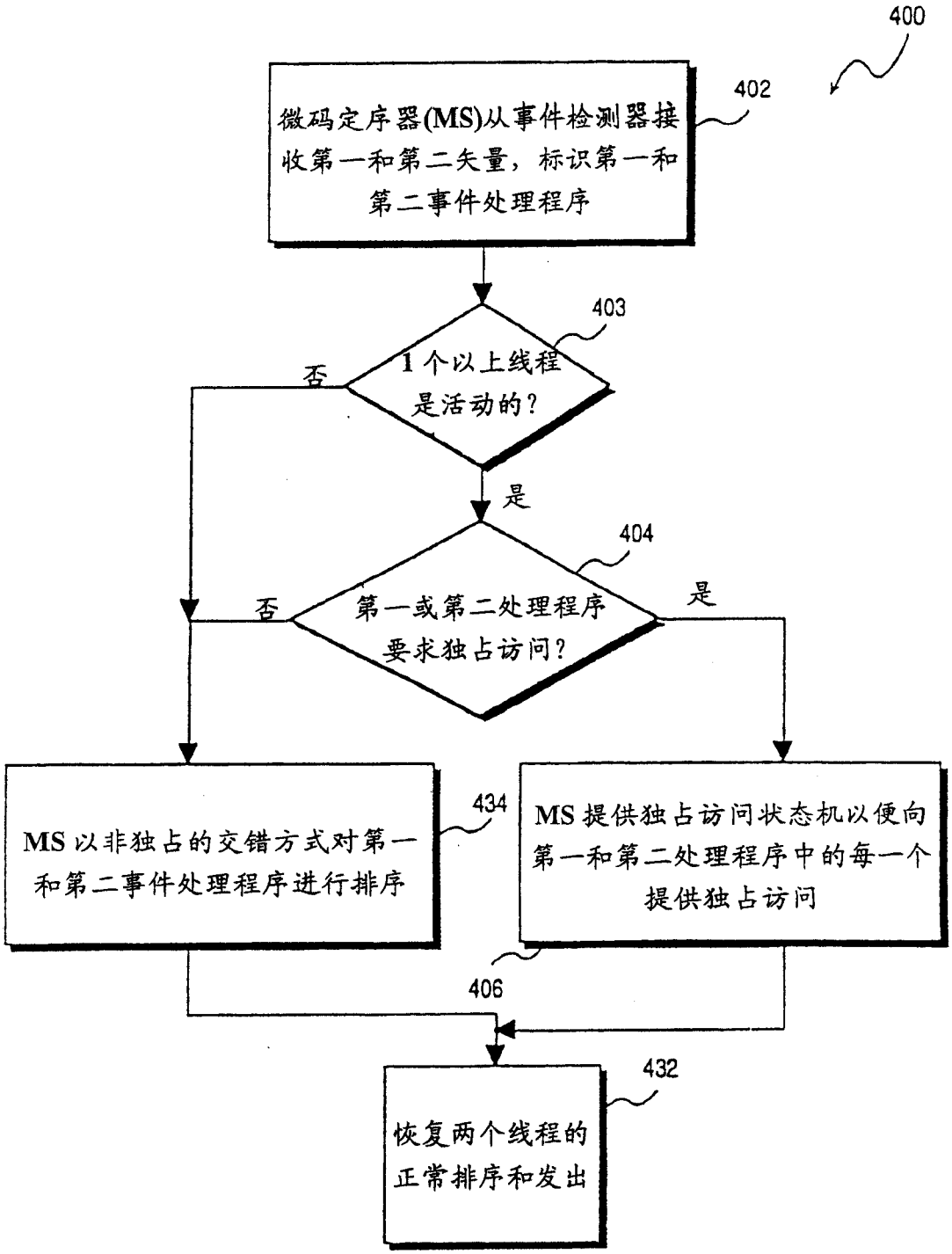

400
402
微码定序器(MS)从事件检测器接收第一和第二矢量，标识第一和第二事件处理程序
403
1个以上线程是活动的？
否
是
404
第一或第二处理程序要求独占访问？
否
是
434
MS以非独占的交错方式对第一和第二事件处理程序进行排序
MS提供独占访问状态机以便向第一和第二处理程序中的每一个提供独占访问
406
432
恢复两个线程的正常排序和发出

图 13

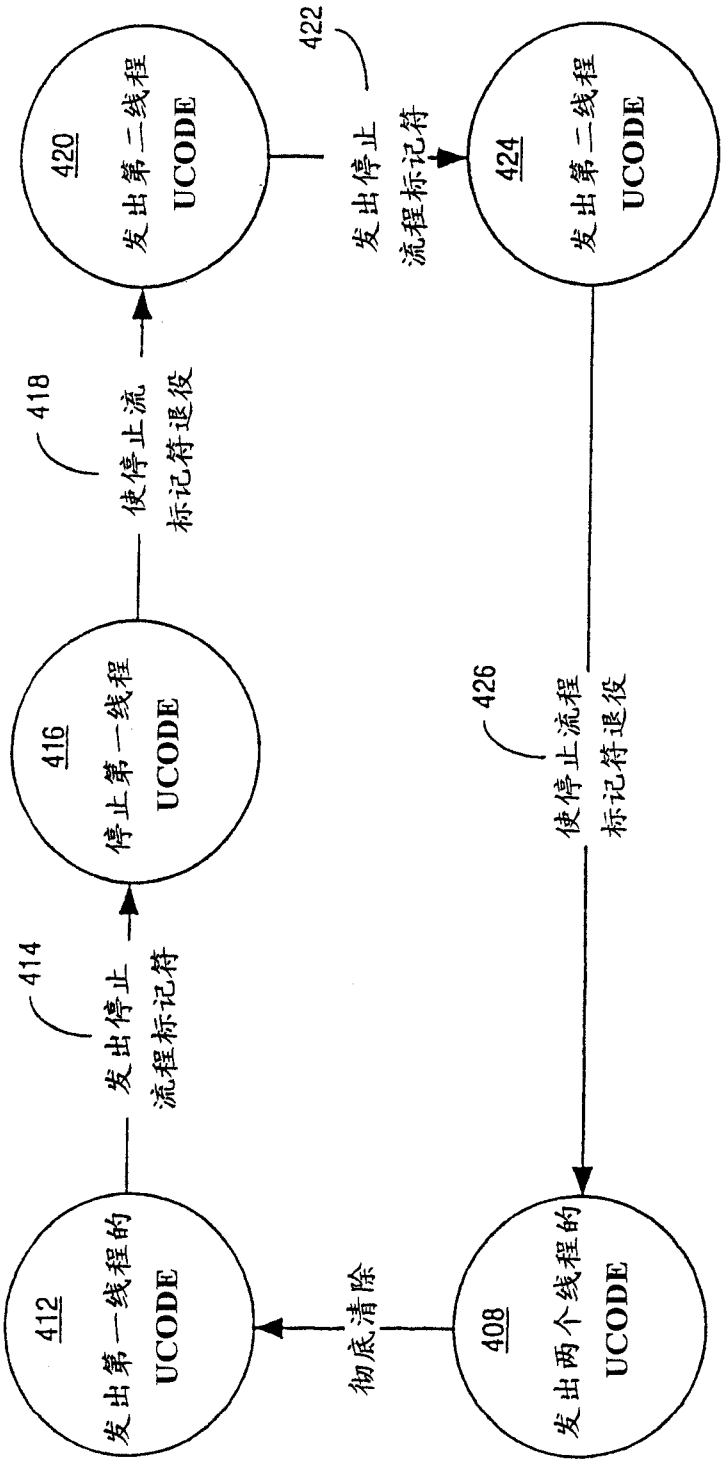

408
发出两个线程的
UCODE
彻底清除
412
发出第一线程的
UCODE
414
发出停止
流程标记符
416
停止第一线程
UCODE
418
使停止流
标记符退役
420
发出第二线程
UCODE
422
发出停止
流程标记符
424
发出第二线程
UCODE
426
使停止流程
标记符退役

图 14

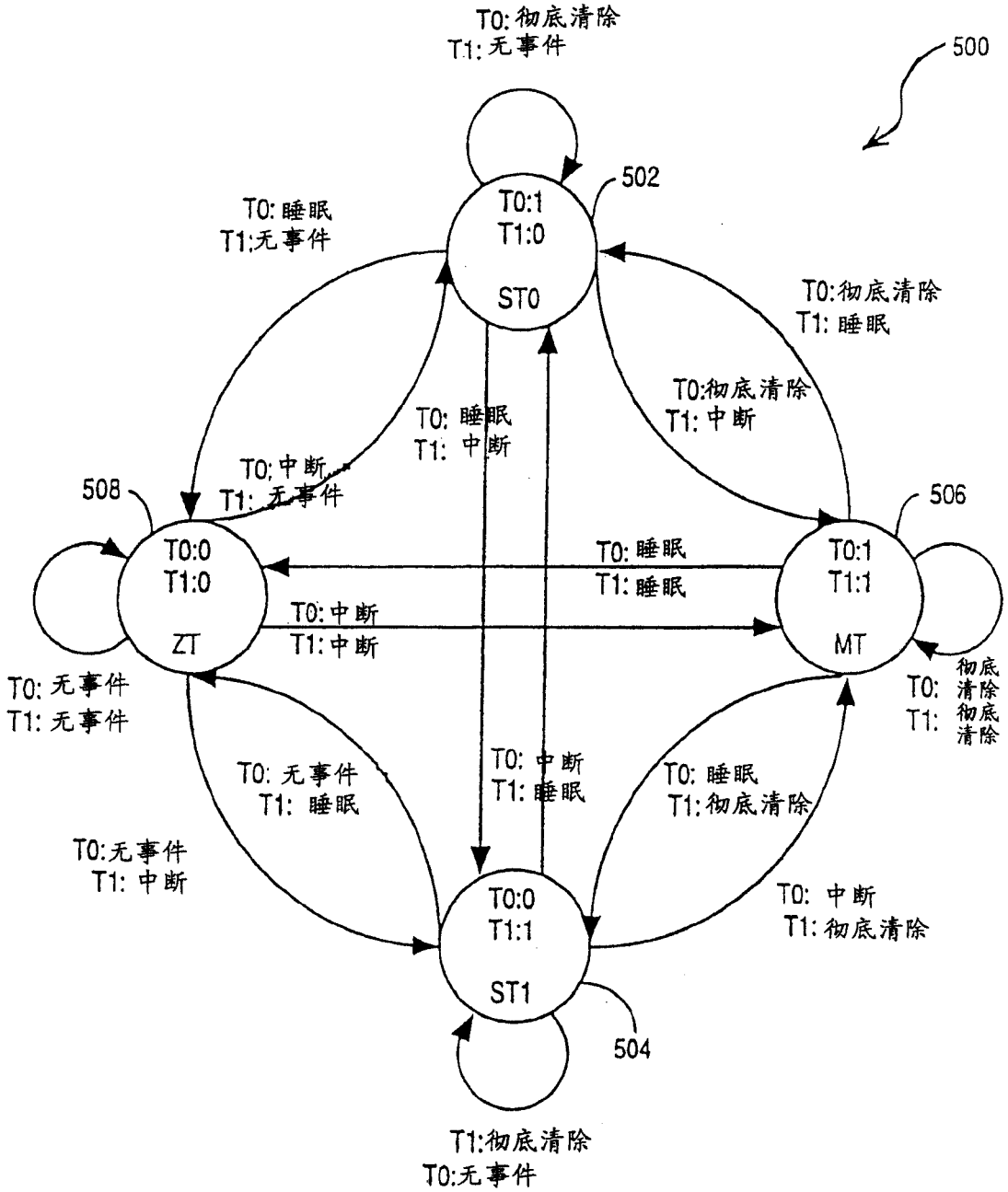

T0:彻底清除
T1:无事件
500
502
T0:1
T1:0
ST0
T0:睡眠
T1:无事件
T0:彻底清除
T1:睡眠
T0:彻底清除
T1:中断
T0:睡眠
T1:中断
T0:中断
T1:无事件
508
506
T0:0
T1:0
ZT
T0:睡眠
T1:睡眠
T0:1
T1:1
MT
T0:中断
T1:中断
T0:无事件
T1:无事件
彻底清除
T0:
T1:
彻底清除
T0:无事件
T1:睡眠
T0:中断
T1:睡眠
T0:睡眠
T1:彻底清除
T0:无事件
T1:中断
T0:0
T1:1
ST1
T0:中断
T1:彻底清除
504
T1:彻底清除
T0:无事件

图 15

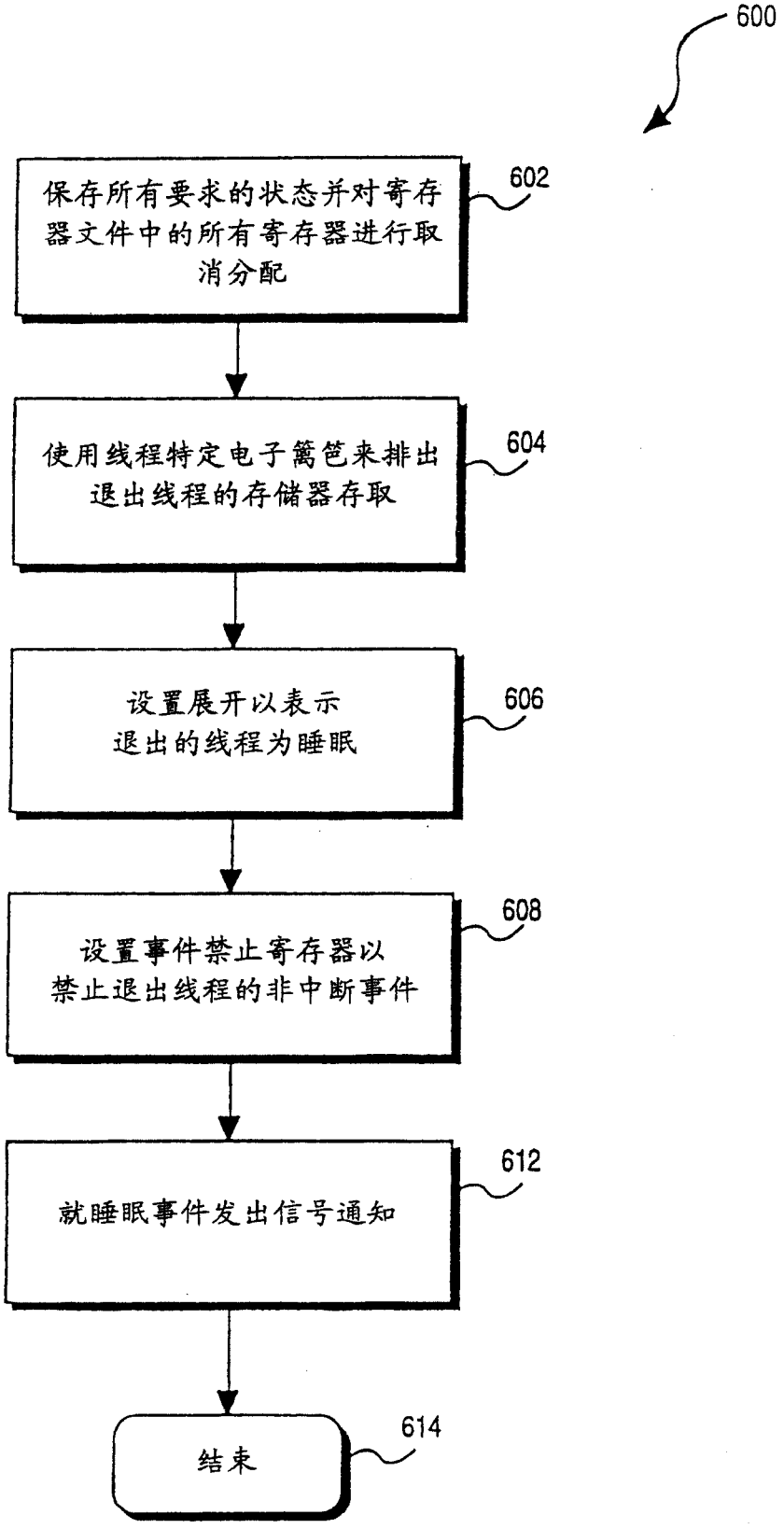

600
保存所有要求的状态并对寄存器文件中的所有寄存器进行取消分配
602
使用线程特定电子篱笆来排出退出线程的存储器存取
604
设置展开以表示退出的线程为睡眠
606
设置事件禁止寄存器以禁止退出线程的非中断事件
608
就睡眠事件发出信号通知
612
结束
614

图 16A

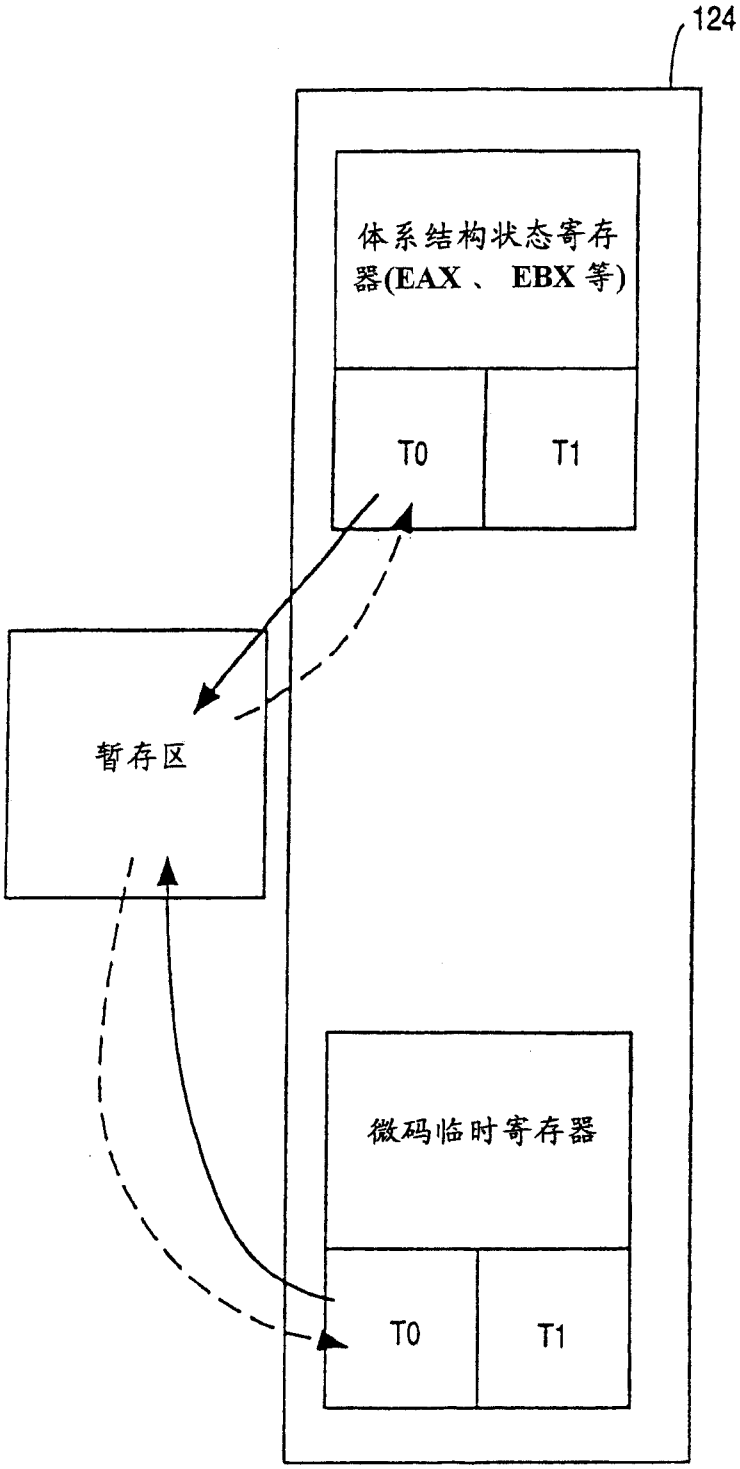


图 16B

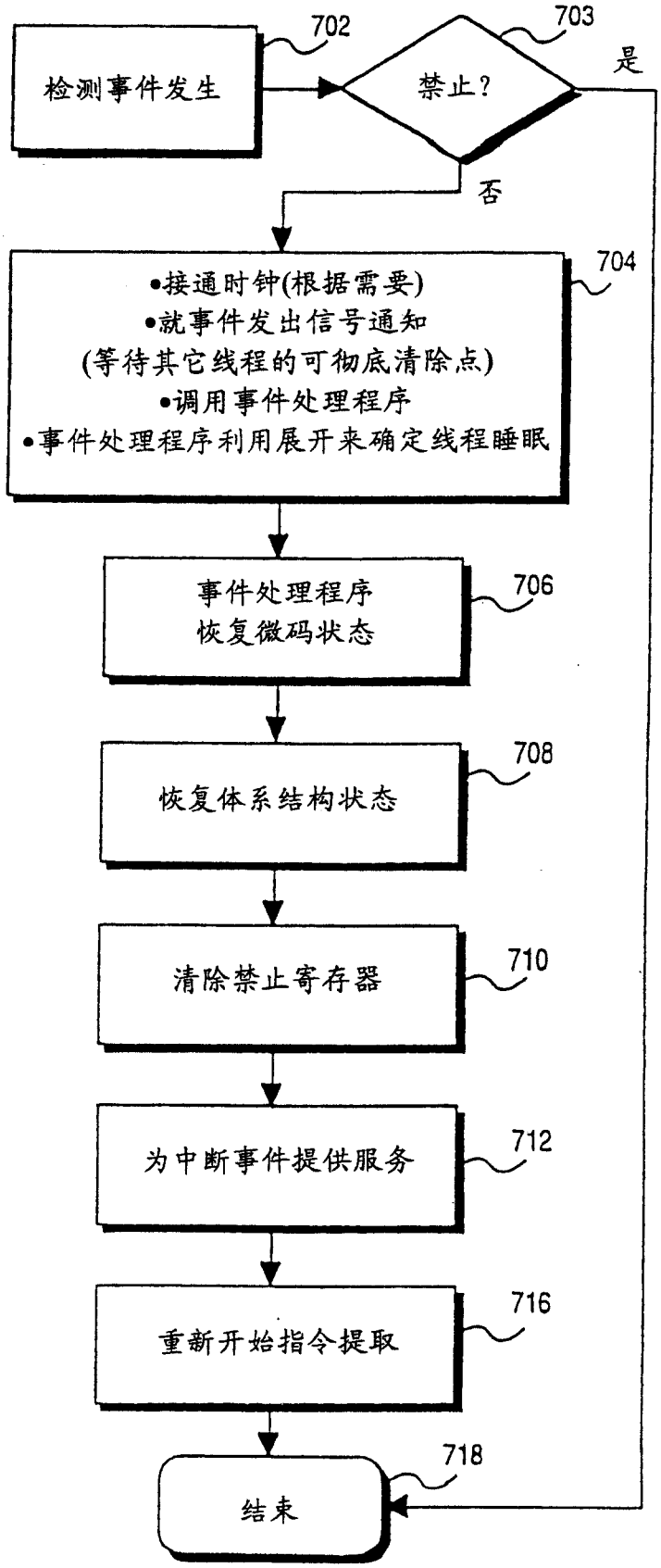

检测事件发生
702
禁止?
703
是
否
•接通时钟(根据需要)
•就事件发出信号通知
(等待其它线程的可彻底清除点)
•调用事件处理程序
•事件处理程序利用展开来确定线程睡眠
704
事件处理程序
恢复微码状态
706
恢复体系结构状态
708
清除禁止寄存器
710
为中断事件提供服务
712
重新开始指令提取
716
结束
718

图 17

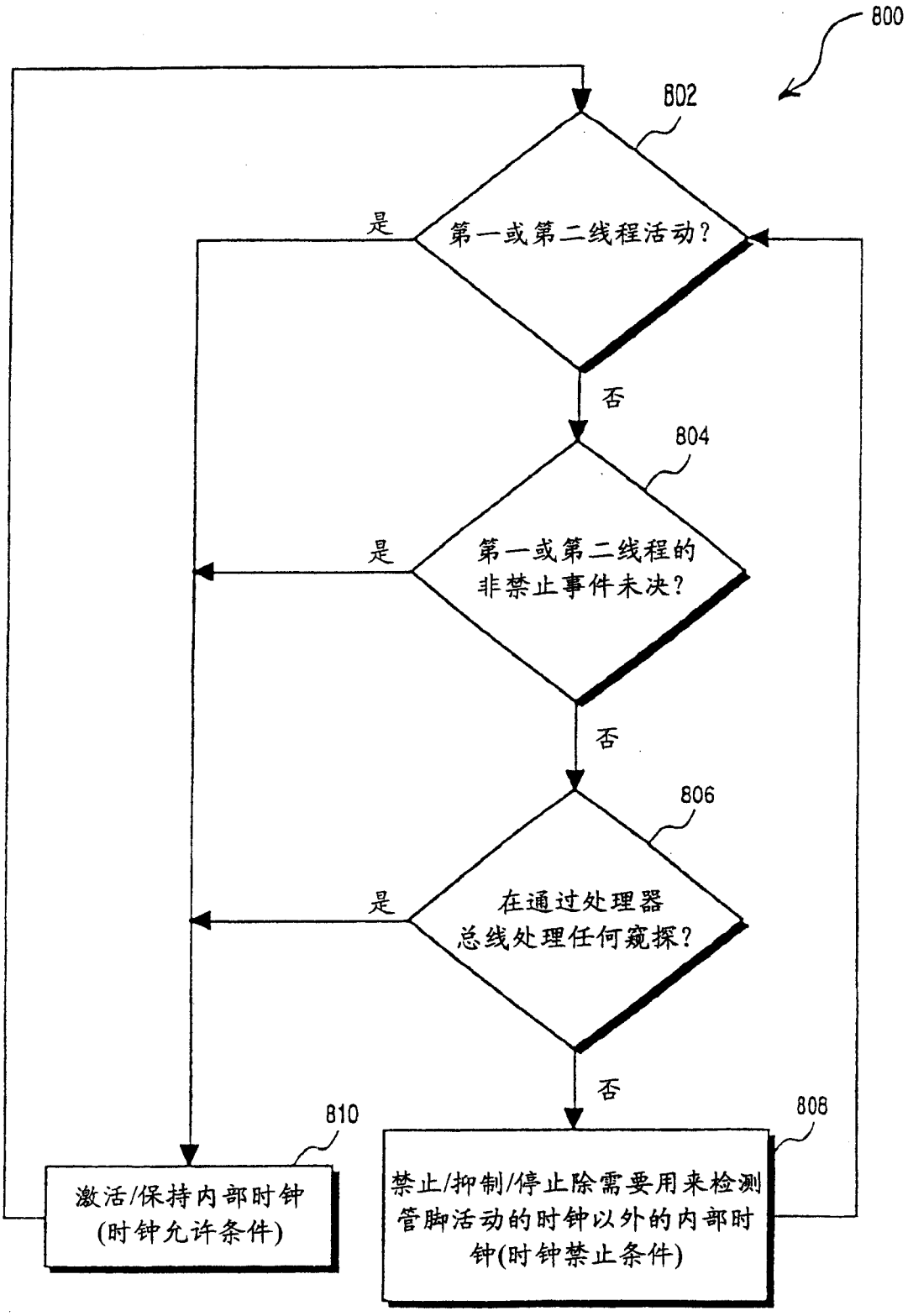
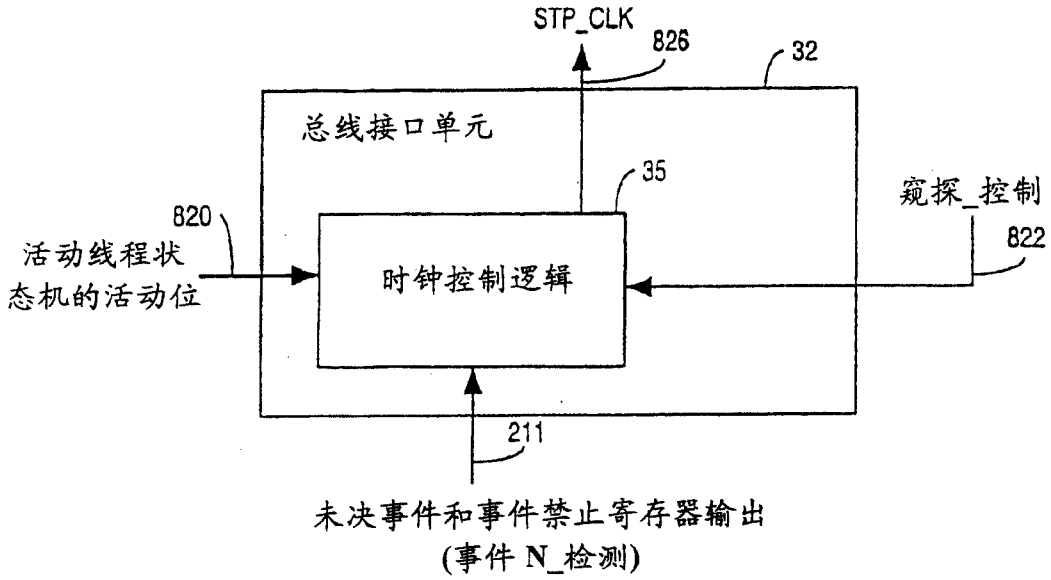

800
802
第一或第二线程活动?
是
否
804
第一或第二线程的
非禁止事件未决?
是
否
806
在通过处理器
总线处理任何窥探?
是
否
808
禁止/抑制/停止除需要用来检测
管脚活动的时钟以外的内部时
钟(时钟禁止条件)
810
激活/保持内部时钟
(时钟允许条件)

图 18

图 19A


STP_CLK
826
32
总线接口单元
35
820
活动线程状
态机的活动位
时钟控制逻辑
窥探_控制
822
211
未决事件和事件禁止寄存器输出
(事件 N_检测)

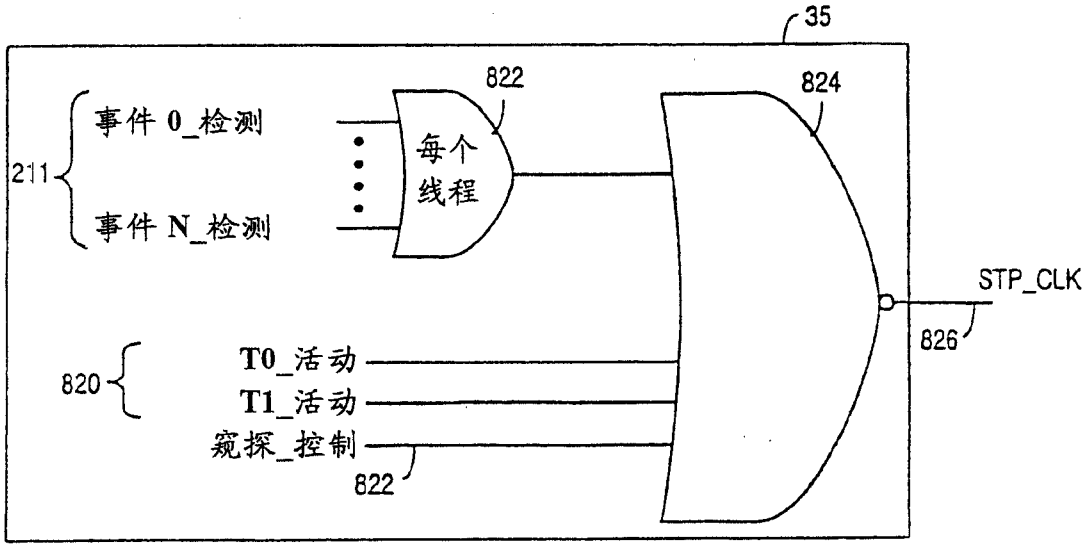

35
822
824
事件 0_检测
211
事件 N_检测
每个
线程
STP_CLK
826
820
T0_活动
T1_活动
窥探_控制
822

图 19B