[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 12/10 (2006.01)

[12] 发明专利说明书

专利号 ZL 98813652.X

[45] 授权公告日 2008 年 6 月 4 日

[11] 授权公告号 CN 100392622C

[22] 申请日 1998.12.11 [21] 申请号 98813652.X

[30] 优先权

[32] 1997.12.16 [33] US [31] 08/991,734

[86] 国际申请 PCT/US1998/026409 1998.12.11

[87] 国际公布 WO1999/031594 英 1999.6.24

[85] 进入国家阶段日期 2000.8.16

[73] 专利权人 英特尔公司

地址 美国加利福尼亚州

[72] 发明人 H·阿克卡利

[56] 参考文献

US5420990A 1995.5.30

US5613083A 1997.3.18

审查员 韩燕_1

[74] 专利代理机构 中国专利代理(香港)有限公司

代理人 吴立明 王忠忠

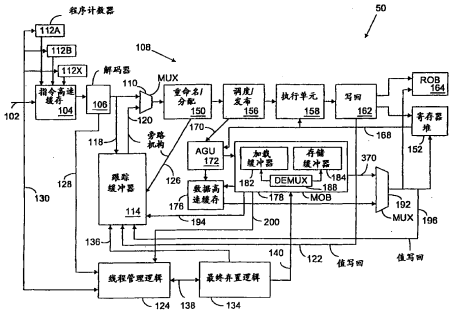
权利要求书 3 页 说明书 37 页 附图 25 页

[54] 发明名称

具有存储器顺序缓冲器的处理器

[57] 摘要

本发明的处理器，包含一个包含多个加载缓冲器和多个存储缓冲器的存储器顺序缓冲器 MOB，每个加载缓冲器保存一组线程之一的多个加载指令，每个存储缓冲器保存线程之一的多个存储指令，而线程中的至少一些之间有时存在从属关系，其中，在每个加载缓冲器中以程序顺序保存加载指令，在每个存储缓冲器中以程序顺序保存存储指令，MOB 对加载和存储指令排序，保持线程中的加载指令与存储指令之间的数据一致性，MOB 包括匹配确定逻辑和再执行触发逻辑；匹配确定逻辑确定加载和存储指令中的某些指令的地址之间是否匹配；当在选定的条件下加载和存储指令的地址之间存在匹配时，再执行确定逻辑控制对所述加载和存储指令中选定的指令的再次执行，无需再次执行包含被再次执行的加载和存储指令的全部线程。



1. 一种处理器，包含:

一个包含多个加载缓冲器和多个存储缓冲器的存储器顺序缓冲器，其中，每个加载缓冲器保存一组线程之一的多个加载指令，每个存储缓冲器保存所述线程之一的多个存储指令，而所述线程中的至少一些线程之间有时存在从属关系，其中，在每个加载缓冲器中以程序顺序保存加载指令，并且在每个存储缓冲器中以程序顺序保存存储指令，并且其中，所述存储器顺序缓冲器 对加载和存储指令排序，以保持所述线程中的加载指令与存储指令之间的数据一致性，而其中所述存储器顺序缓冲器包括匹配确定逻辑和再执行触发逻辑;

其中，所述匹配确定逻辑确定加载和存储指令中的某些指令的地址之间是否存在匹配; 以及

其中，当在选定的条件下所述加载和存储指令的地址之间存在匹配时，再执行确定逻辑控制对所述加载和存储指令中选定的指令的再次执行，而无需再次执行包含被再次执行的加载和存储指令的全部线程。

2. 权利要求 1 的处理器，进一步包含控制线程的动态生成的线程管理逻辑，其中，线程管理逻辑被耦合到所述存储器顺序缓冲器，使得由线程管理逻辑生成的线程的加载和存储指令中的至少一些指令被供给所述存储器顺序缓冲器。

3. 权利要求 1 的处理器，进一步包含提供程序顺序和弃置顺序的标志的线程管理逻辑，其中，所述线程管理逻辑被耦合到所述存储器顺序缓冲器，使得由线程管理逻辑生成的线程的加载和存储指令中的至少一些指令被供给存储器顺序缓冲器。

4. 权利要求 3 的处理器，其中，线程在最终弃置时按弃置顺序从存储器顺序缓冲器弃置。

5. 权利要求 3 的处理器，其中，在确定所有指令已经在没有推测错误的情况下被执行或者已经是被复位线程的一部分之后，线程在最终弃置时按弃置顺序从所述存储器顺序缓冲器 弃置。

6. 权利要求 1 的处理器，其中，存储缓冲器包含一个字段，用于保存要存储到存储器的数据。

7. 权利要求 1 的处理器，其中，加载缓冲器包含状态字段，它包含关于加载指令中的相应一个以前已经接收存储器中的数据还是存储缓冲器中的数据的标志。

8. 权利要求 1 的处理器，其中，加载缓冲器包含状态字段，它包含加载指令中的相应一个以前从其接收过数据的存储缓冲器项的存储缓冲器标识号 。

9. 权利要求 1 的处理器，其中，加载缓冲器包含状态字段，它包含加载指令以前从其接收过数据的存储指令的线程标识号。

10. 一种处理器，包含:

一个用于并行地执行至少一部分线程的执行管道，所述执行管道包含一个对加载和存储指令排序的存储器顺序缓冲器，其中，所述存储器顺序缓冲器包含加载缓冲器和存储缓冲器，而加载缓冲器中不同的加载缓冲器保存来自所述线程中不同的线程的加载指令，存储缓冲器中不同的存储缓冲器则保存来自所述线程中不同的线程的存储指令，其中所述线程中的至少一些线程之间有时存在从属关系，并且其中在加载缓冲器中以程序顺序保存加载指令，在存储缓冲器中以程序顺序保存存储指令;

检测电路，检测与线程从属关系中的至少一些线程从属关系和加载缓冲器中的加载指令相关的推测错误; 以及

触发逻辑，用于把所述加载指令中的至少一些加载指令标识为从属于所述推测错误至少其中之一，并用于触发对所标识的加载指令中的至少一些加载指令的重新执行; 以及

在执行管道外部的跟踪缓冲器，用于在执行之后将加载和存储指令中至少一部分指令的备份保存在所述存储器顺序缓冲器中，一直到最终弃置，其中跟踪缓冲器被耦合到所述存储器顺序缓冲器。

11. 权利要求 10 的处理器，其中，触发逻辑包含再执行触发电路，用于触发至少一些加载指令的再执行，其中，再执行触发逻辑电路被耦合到存储器顺序缓冲器，且是用以比较加载指令的地址和存储指令的地址的匹配确定逻辑的一部分。

12. 一种处理器，包含:

一个存储器顺序缓冲器，它包括:

各自用于保存一组线程之一的加载指令的多个加载缓冲器，其中在加载缓冲器中以程序顺序保存加载指令;

各自用于保存所述线程之一的存储指令的多个存储缓冲器，其中，在存储缓冲器中以程序顺序保存存储指令，而所述线程中的至少一些线程之间有时存在从属关系;

用于将要执行的加载指令的其中一条加载指令的地址与存储缓冲器中的至少一条存储指令的地址进行比较的比较电路; 和

用于确定加载指令是从存储器读取数据还是从存储缓冲器之一中读取数据的数据路径控制逻辑。

13. 一种处理器，包含:

一个存储器顺序缓冲器，它包括:

各自用于保存一组线程之一的加载指令的多个加载缓冲器，其中在所述加载缓冲器中以程序顺序保存加载指令;

各自用于保存所述线程之一的存储指令的多个存储缓冲器，其中在所述存储缓冲器中以程序顺序保存存储指令，而所述线程中的至少一些线程之间有时存在从属关系;

用于将要执行的存储指令的其中一条存储指令的地址和至少一个状态位与至少一条加载指令的地址和至少一个状态位进行比较的比较电路; 和

用于根据该比较确定是否重新执行一条或多条加载指令的检测逻辑。

具有存储器顺序缓冲器的处理器

技术领域

本发明涉及处理器，更具体来说，涉及具有存储器顺序缓冲器的处理器。

背景技术

当前的超标量处理器，如微处理器，执行诸如转移预测和无序执行的技术，以增强效能。具有无序执行管道的处理器执行某些指令时，执行的次序不同于指令被提取和解码的次序。对于没有从属对象(dependencies)的指令来说，指令可以无序地执行。无序执行由于防止了执行单元仅仅因为程序指令的顺序就闲置，因而提高了处理器的效能。指令结果是在执行之后重新排序的。

通过限制指令解码要顺序进行，简化了处理数据从属对象的任务。处理器然后就可以确定数据如何通过寄存器从一条指令流到后继指令。为确保程序的正确性，寄存器要被重命名，指令要在备用站(reservation stations)中一直等到它们的输入操作数的生成，这时指令才被发往适当的功能单元去执行。寄存器重命名器(renamer)、备用站以及有关机构，将具有从属对象的指令连接在一起，使得从属指令不在其所从属的指令之前执行。相应地，这种处理器受到顺序提取和解码的限制。

当来自指令高速缓存的指令缺失，或者某个转移被误预测(mis-predicted)时，处理器只得等待，要么等到从更高层的高速缓存或存储器提取了指令块，要么等到误预测的转移被还原，错误路径的执行被复位。这种过程的结果是，在指令高速缓存缺失和误预测转移之前和之后的独立指令不能被并行地执行—尽管这样做是正确的。

已经采用了存储器顺序缓冲器以排序加载和存储。处理器中需要有改进的机制让处理器从推测错误中恢复。

发明内容

根据本发明的一个方面，一种处理器，包含:

一个包含多个加载缓冲器和多个存储缓冲器的存储器顺序缓冲器(MOB)，其中，每个加载缓冲器保存一组线程之一的多个加载指令，每个存储缓冲器保存所述线程之一的多个存储指令，而所述线程中的至少一些线程之间有时存在从属关系，其中，在每个加载缓冲器中以程序顺序保存加载指令，并且在每个存储缓冲器中以程序顺序保存存储指令，并且其中，所述 MOB 对加载和存储指令排序，以保持所述线程中的加载指令与存储指令之间的数据一致性，而其中所述 MOB 包括匹配确定逻辑和再执行触发逻辑;

其中，所述匹配确定逻辑确定加载和存储指令中的某些指令的地址之间是否存在匹配; 以及

其中，当在选定的条件下所述加载和存储指令的地址之间存在匹配时，再执行确定逻辑控制对所述加载和存储指令中选定的指令的再次执行，而无需再次执行包含被再次执行的加载和存储指令的全部线程。

根据本发明的另一个方面，一种处理器，包含:

一个用于并行地执行至少一部分线程的执行管道，所述执行管道包含一个对加载和存储指令排序的存储器顺序缓冲器(MOB)，其中，所述存储器顺序缓冲器包含加载缓冲器和存储缓冲器，而加载缓冲器中不同的加载缓冲器保存来自所述线程中不同的线程的加载指令，存储缓冲器中不同的存储缓冲器则保存来自所述线程中不同的线程的存储指令，其中所述线程中的至少一些线程之间有时存在从属关系，并且其中在加载缓冲器中以程序顺序保存加载指令，在存储缓冲器中以程序顺序保存存储指令;

检测电路，检测与线程从属关系中的至少一些线程从属关系和加载缓冲器中的加载指令相关的推测错误; 以及

触发逻辑，用于把所述加载指令中的至少一些加载指令标识为从属于所述推测错误至少其中之一，并用于触发对所标识的加载指令中的至少一些加载指令的重新执行; 以及

在执行管道外部的跟踪缓冲器，用于在执行之后将加载和存储指令中至少一部分指令的备份保存在所述 MOB 中，一直到最终弃置，

其中跟踪缓冲器被耦合到所述 MOB。

根据本发明的又一个方面，一种处理器，包含:

一个存储器顺序缓冲器，它包括:

各自用于保存一组线程之一的加载指令的多个加载缓冲器，其中在加载缓冲器中以程序顺序保存加载指令;

各自用于保存所述线程之一的存储指令的多个存储缓冲器，其中，在存储缓冲器中以程序顺序保存存储指令，而所述线程中的至少一些线程之间有时存在从属关系;

用于将要执行的加载指令的其中一条加载指令的地址与存储缓冲器中的至少一条存储指令的地址进行比较的比较电路; 和

用于确定加载指令是从存储器读取数据还是从存储缓冲器之一中读取数据的数据路径控制逻辑。

根据本发明的再一个方面，一种处理器，包含:

一个存储器顺序缓冲器，它包括:

各自用于保存一组线程之一的加载指令的多个加载缓冲器，其中在所述加载缓冲器中以程序顺序保存加载指令;

各自用于保存所述线程之一的存储指令的多个存储缓冲器，其中在所述存储缓冲器中以程序顺序保存存储指令，而所述线程中的至少一些线程之间有时存在从属关系;

用于将要执行的存储指令的其中一条存储指令的地址和至少一个状态位与至少一条加载指令的地址和至少一个状态位进行比较的比较电路; 和

用于根据该比较确定是否重新执行一条或多条加载指令的检测逻辑。

在本发明的一个实施例中，处理器包括一个包含加载缓冲器和存储缓冲器的存储器顺序缓冲器(MOB)，其中，MOB 对加载和存储指令排序，以保持不同线程中的加载与存储指令之间的数据一致性（data coherency），其中至少有一个线程从属于至少另一个线程。在本发明的另一个实施例中，处理器包括一个执行管道，用于并行地执行至少一部分线程，其中至少有一个线程从属于至少另一个线程，执行管道包含一个对加载和存储指令排序的存储器顺序缓冲器。处理器也包含检测电路，用于检测与加载缓冲器中的加载指令相关

联的推测错误。

附图说明

从下面展开的详细说明和本发明实施例的各附图中将能更全面地了解本发明，不过，这些说明和附图不应被用来将本发明限定于所说明的特定实施例，相反，它们只是为了方便对本发明解释和理解。

图 1 是一个处理器实施例中某些部件的高级框图表示。

图 2 是按照本发明一个实施例的处理器的框图。

图 3 是按照本发明另一个实施例的处理器的框图。

图 4 是一个二线程的例子的流程图。

图 5 是另一个二线程的例子的流程图。

图 6 是一个四线程的例子的流程图。

图 7 是表示图 6 的线程的重叠执行的示意图。

图 8 是表示按照本发明一个实施例的各个跟踪缓冲器的框图。

图 9 表示一个指示在两个时刻的程序和弃置顺序的阵列。

图 10 是图 8 的一个跟踪缓冲器实施例中的某些部件的框图表示。

图 11 是图 8 的另一个跟踪缓冲器实施例中的某些部件的框图表示。

图 12 是图 10 的跟踪缓冲器的指令队列阵列的一个实施例的一部分的图形表示。

图 13 是图 10 的跟踪缓冲器的数据与从属对象阵列的一个实施例的一部分的图形表示。

图 14 是表示修改量寄存器(modifier registers)的一个实施例和在创建图 10 的阵列的从属字段(dependency field)中所使用的一种修改后的寄存器(modified register)。

图 15 在创建图 13 的阵列的从属字段中所使用的一种逻辑“或”门。

图 16 是表示用于创建图 13 的阵列的从属字段的操作的一个实施例的流程图。

图 17 是按照本发明实施例的、某特定寄存器和跟踪缓冲器中上有从属对象的位置的图形表示。

图 18 是图 10 的跟踪缓冲器的输出寄存器堆的一个实施例的一部分的图形表示。

图 19 是图 10 的跟踪缓冲器的输入寄存器堆的一个实施例的一部分的图形表示。

图 20 是按照本发明实施例的、与图 18 的输出寄存器堆和图 19 的输入寄存器堆联合使用的比较器和再执行触发逻辑(replay triggering logic)的框图。

图 21 是表示输出寄存器堆的内容可以被使用的程序位置的流程图。

图 22 是表示按照本发明实施例、位于图 2 的存储器顺序缓冲器(MOB)内的各个 MOB 的框图。

图 23 是图 22 的 MOD 的存储缓冲器(store buffer)的一个实施例的局部图示。

图 24 是图 22 的 MOD 的加载缓冲器(load buffer)的一个实施例的局部图示。

图 25 表示一个比较加载和存储指令的地址的比较器。

图 26 表示一个比较加载和存储指令的地址的比较器。

图 27 是按照本发明实施例的 MOB 控制电路和存储缓冲器的框图表示。

图 28 是按照本发明实施例的 MOB 控制电路和加载缓冲器的框图表示。

图 29 是一个六线程的例子的流程图。

图 30 是一个树，表示在时刻 t1 时图 29 的线程中的关系。

图 31 是一个树，表示在假定线程 T4 在线程 T1 弃置之前复位的情况下，在时刻 t2 时图 29 的线程中的关系。

图 32 是一个树，表示在假定线程 T1 在线程 T4 复位之前弃置的情况下，在时刻 t2 时图 29 的线程中的关系。

图 33 是一个树，表示在时刻 t3 时图 29 的线程中的关系。

图 34 是表示一个五线程的例子的框图。

图 35 是一个树，表示在时刻 t1 时图 34 的线程中的关系。

图 36 是一个树，表示在时刻 t2 时图 34 的线程中的关系。

图 37 是按照替代图 2 的实施例的另一个实施例的处理器的框图

表示。

图 38 是一个包括图 2 的处理器的计算机系统。

具体实施方式

A. 线程的创建和管道 108 概述

B. 关于跟踪缓冲器 114 的详细说明

1. 跟踪缓冲器 114A

a. 指令队列阵列 202A

b. DAD 阵列 206A 和从属生成电路 212A

c. 输出寄存器堆 210A 和输入寄存器堆 208A

2. 跟踪缓冲器 114'

C. 一个再执行顺序算法

D. 二级或最终弃置

E. 存储系统

1. 存储缓冲器和加载缓冲器

2. 加载和存储地址的比较

a. 加载指令的执行

b. 存储指令的执行

c. 复位

3. 存储指令的再执行

4. 多个加载指令的再执行

5. 加载和存储指令的最终弃置

F. 关于线程管理逻辑和最终弃置逻辑的其它信息

G. 一个没有多线程操作（multithreading）的实施例

H. 其它信息和实施例

图 1 表示处理器 10 的某些部件。处理器 10 包括执行管道 12 和跟踪缓冲器 14，后者位于执行管道 12 之外。执行管道 12 可包括存储器顺序缓冲器。导线 18 上的指令被提供到执行管道 12 去执行。指令也通过导线 22 被提供到跟踪缓冲器 14。在执行管道 12 中，可以推测性地执行指令。推测的例子包括数据推测（data speculation）和从属性推测（dependency speculation）。可以广泛使用各种推测。处理器 10 包括有检测推测错误（误推测）和从错误推测恢复的机制（包括跟踪缓冲器 14）。

当检测到误推测时，就将误推测指令从跟踪缓冲器 14 通过导线 24 提供到执行管道 12，并在执行管道 12 中再执行。如果某指令“被再执行”，则该指令和从属于该指令的所有指令要被重新执行，不过未必同时执行。如果某指令“被完全再执行”，则该指令和按程序顺序来说位于该指令之后的所有指令要被重新执行。程序顺序是指令在顺序处理器中执行时应有的顺序。指令可以完全地按程序顺序通过导线 18，也可以不按程序顺序。处理器 10 可以是一个有序或无序处理器。从属指令的重新执行，会产生依赖于正在被再执行的从属指令的指令。通过控制触发再执行的事件，能控制指令重新执行的次数。一般来说，执行一词的含义包括原始执行和重新执行。至少将一部分指令的结果通过导线 26 提供到跟踪缓冲器。最终弃置逻辑 34 在确定指令的原始执行正确或重新执行正确时，就弃置跟踪缓冲器 14 中的这种指令。

执行管道 12 可以是各种各样的执行管道中的任何一种，可以是更大型管道的一部分。执行管道 12 可以用于各种各样的处理器。图 2 中所示的例子，表示具有执行管道 108 的处理器 50 的各种部件，图 3 中所示的例子，表示具有执行管道 308 的处理器 100。在图 2 的本发明一个实施例中，执行管道 108 包括寄存器重命名。在另一个实施例中，执行管道不包括寄存器重命名。处理器可以并发地处理多个线程（如图 2 中的处理器 50 那样），或者不同时地处理多个线程（如图 3 中的处理器 100 那样）。首先将讨论处理器 50。

本说明书中提及的“一个实施例”或“实施例”，指的是结合该实施例所述的特定特点、结构或特征，被本发明的至少一个实施例涵盖。本说明书中各处出现的“在一个实施例中”，不一定都指同一个实施例。

A. 线程的创建和管道 108 概述

通过导线 102 将指令提供到指令高速缓存（I-高速缓存）104。图中显示的解码器 106 从 I-高速缓存接受指令，但是解码器也可以在指令到达 I-高速缓存之前就解码指令。视所选择的环境和具体实现的不同，“指令”一词可以包括宏操作（macro-op）、微操作（uops）或者某个其它形式的指令。有各种各样的指令集可以使用，它们包括—但不限于—精简指令集计算（RISC）或复杂指令集计算（CISC）

指令。此外，解码器 106 也可以将 CISC 指令译解成 RISC 指令。I-高速缓存 104 中的指令，通过 MUX 110 提供到管道 108，通过导线 118 提供到跟踪缓冲器 114。

跟踪（trace）是一个指令集。线程包括跟踪和有关信号，诸如寄存器值和程序计数器值。

线程管理逻辑 124 由 I-高速缓存 104 中的程序或过程创建不同的线程，方法是通过导线 130 向程序计数器 112A、112B、...112X 提供起始计数（其中 X 代表程序计数器的个数）。例如，X 可以是 4，或者大于 4 或小于 4。线程管理逻辑 124 能通过停止相关的程序计数器来结束线程。线程管理逻辑 124 可以使程序计数器再开始另一个线程。不同线程的各部分是并发地从 I-高速缓存 104 读取的。

为了确定在程序或过程中的何处创建线程，线程管理逻辑 124 可通过导线 128 从解码器 106 读取指令。线程可以包括由程序员或编译程序插入的、明确地划定了线程的起始和终结界限的指令。另外，线程管理逻辑 124 也可以分析程序或过程的指令，把向 I-高速缓存 124 提供的程序或过程分解成不同的线程。例如，转移、循环、后向转移、返回、跳转、过程调用和函数调用都是分割线程的好位置。线程管理逻辑 124 会考虑潜在线程的长度、使用多少变量、连续的线程之间公用的变量的个数、以及涉及从何处开始线程的其它因素。线程管理逻辑 124 在确定线程的边界时，会考虑程序顺序。程序顺序是线程和线程内部的指令在顺序处理器中执行时应有的顺序。线程内部的指令可以无序地执行（与程序顺序相反）。线程可以由管道 108 作基本上独立的处理。线程管理逻辑 124 可以包括一个包含历史表在内的预测机构，以避免作出次于最优的选择。例如，线程管理逻辑 124 可能在创建一个线程后确定该线程原来实际上并非程序顺序的一部分。在这种情况下，如果再次遇到同样的程序代码，就可以用该预测机构来决定是否还要创建同样的线程。线程管理逻辑 124 可以将线程的动态创建与对编译程序或程序员明确的指令提示的使用组合起来使用，来确定在指令中的什么位置创建线程。

动态创建线程是由并非特地为多线程操作编写或编译的程序来创建线程，其中线程中至少有一个从属于线程的另一个。程序可能起源于一个包含执行管道 108 和线程管理逻辑 124 的芯片。动态创

建线程、执行线程、检测并更正执行中的推测错误，统称为动态多线程操作。

图 4 表示一个包含一条条件后向转移指令的线程 T1。按照程序顺序，线程 T2 是在该条件后向转移指令之后执行的。按照时间顺序，当线程 T1 第一次到达该条件后向转移指令时，线程 T2 就被推测性地开始执行。因此，线程 T1 和 T2 有的部分是并行执行的。如果线程 T2 牵涉到误推测，则线程 T2 受影响的指令就要再执行。

线程管理逻辑 124 可通过导线 130 监测程序计数器的计数值。监测计数值的目的是确定某线程何时要结束。例如，当不满足条件转移的条件时，要是允许线程 T1 的程序计数器继续的话，它就会前进到线程 T2 的第一条指令。所以，当条件不满足时，线程管理逻辑 124 就停止线程 T1 的程序计数器。

图 5 表示一个包含一条函数调用指令的线程 T1。按照程序顺序，当到达该调用指令时，程序计数器就跳转到该函数的位置并执行到一条返回指令，此时，程序计数器返回到该调用之后的指令。按照程序顺序，线程 T2 是从该返回之后的指令开始执行的。按照时间顺序，当线程 T1 第一次到达该调用时，线程 T2 就被推测性地开始执行。如果线程 T2 牵涉到误推测，则线程 T2 受影响的指令就要再执行。线程 T1 在其程序计数器到达线程 T2 的第一条指令时结束。图 5 中的 Load MX 和 Store MX 指令将在下文讨论。

图 6 表示的线程 T1、T2、T3 和 T4 是一个程序段落的一部分。不同的程序计数器产生了线程 T1、T2、T3 和 T4。线程 T1 包含的指令，先是达到点 A（函数调用指令），然后从点 B 到点 C（条件后向转移指令）、到点 D、再到点 C（该循环可能要重复若干次）。线程 T2 的起始指令，按照程序顺序来说，紧接着在点 A 调用的函数的返回指令。线程 T3 的起始指令，按照程序顺序来说，紧接着点 C 的条件后向转移，该线程继续到点 E、到点 F、到点 G、到点 H、到点 I，点 I 是一条返回指令，返回到线程 T2 紧接着点 A 后开始的指令。线程 T4 的起始指令，按照程序顺序来说，紧接着点 E 的条件后向转移。

如图 7 中所示，线程 T1、T2、T3 和 T4 有的部分是并行地取指令、解码和执行的。线程的取指令、解码和执行之所以无序，是因为不遵循程序顺序。按照时间顺序，线程 T2、T3 和 T4 分别是紧接

着位于点 A、C 和 E 的指令执行的。纵向的短划线表示一种父子关系。线程 T2、T3 和 T4 是依赖寄存器和/或存储器位置中的数据推测性地被执行的，执行之前并不肯定这些数据是正确的。处理器 100 具有检测误推测并使误推测的指令再执行的机构。事实上线程 T4 不是程序顺序的一部分。可以执行线程 T4，直到线程管理逻辑 124 确定线程 T4 不是程序顺序的一部分。此时，可以将线程 T4 复位，将处理器 100 中在处理线程 T4 时占用的资源释放，然后分配给另一个线程。按照程序顺序，线程 T1、T2 和 T3 要以下列次序执行：首先是线程 T1、其次是线程 T3，最后是线程 T2。

参看图 2，来自 MUX 100 的指令被重命名/分配单元 150 接收，后者提供寄存器堆 152 中被重命名的物理寄存器的物理寄存器标识（PRID）。该 RPID 通过旁路导线 126 被提供给跟踪缓冲器 114。分配涉及到将寄存器分配给指令并分配调度/发布单元 156 的备用站的条目。一旦备用站中的特定指令的操作数准备就绪，该指令就被发往执行单元 158 的一个执行单元（例如整数、浮点），或者发往一个存储器执行管道—后者包括地址生成单元（AGU）172、存储器顺序缓冲器（MOB）178 和数据高速缓存 176。根据指令，可以通过导线 168 从寄存器堆提供操作数。按照本发明的一个实施例，可以这样连接线程内的从属指令，使得它们不按顺序执行。然而，不同线程中会同时有从属指令被无序地取指令、解码和执行。某些线程的执行可能是推测性的。

为了提高性能，将备用站和有关机构设计得使指令的发送具有低延迟和高带宽。对延迟和带宽的要求限制了能在备用站中等待的指令的个数。通过在管道 108 外设置跟踪缓冲器 114，就能有大量的指令可供执行/再执行，而又不会显著降低管道 108 的通量。通过管道操作，能减少执行管道 108 与跟踪缓冲器 114 之间延迟的影响。

执行的结果和相关信息，从写回单元 162 通过导线 122（就寄存器来说）以及通过 MUX 192 和导线 196，被写到跟踪缓冲器 114。结果和相关信息也可以被写到寄存器堆 152 和相关的重定序寄存器（ROB—re-order register）164。就管道 108 而言，一旦某指令的结果和相关信息被写到寄存器堆 152 和 ROB 164，该指令就依次地被弃置。这种弃置被称为第一级或初始弃置。在第一级弃置时或第一

级弃置之前，要释放调度/发布单元 156 中该被弃置指令使用的资源，包括备用站、寄存器堆 152 和 ROB 164。然而，有关该指令的所有需要的详细信息要保留在跟踪缓冲器 114 和 MOB 178 中，一直到最终弃置—下文将作说明。

如果按照程序顺序，在较晚的线程中使用的数据是在较早的线程中产生的，那么在较晚的线程与较早的线程之间就存在着从属性。数据可能是在较早的线程中通过一条存储器或非存储器指令产生的。例如，如果较晚的线程中的某加载指令（load instruction）与较早的线程中的某存储指令（store instruction）有相同的地址，则较晚的线程从属于较早的线程。如果较晚的线程中的某指令使用一个在较早的线程中修改过的寄存器，则较晚的线程也从属于较早的线程。同样，如果按照程序顺序，较晚的指令使用由较早的指令产生的数据，则较晚的指令从属于较早的指令。“从属性”一词也用在“从属性推测”这个短语中。从属性推测的例子是，推测某加载指令与某较早的存储指令之间没有从属性。地址匹配（address matching）是检查从属性推测错误的技术的一个例子。数据推测的例子是，推测某寄存器中的数据是正确数据。寄存器匹配（register matching）是检查数据推测错误技术的一个例子。

B. 关于跟踪缓冲器 114 的详细说明

参看图 8，跟踪存储器 114 包括跟踪缓冲器 114A、114B、114C、…、114Y，其中 Y 代表跟踪缓冲器的个数。例如，如果 Y=4（即 Y=D），则有 4 个跟踪缓冲器。如果 Y 小于 3，则跟踪缓冲器 114 就不是包含图 8 中所示的全部跟踪缓冲器。Y 可以等于 X，也可以与 X 不同（X 是程序计数器个数）。跟踪缓冲器 114 可以是划分成各个跟踪缓冲器的一个单一存储器，或者是物理上分离的跟踪缓冲器，或者是这二者的组合。

参看图 9，在一个实施例中，线程管理逻辑 124 包括一个由线程标识（ID）组成的规定了程序顺序（也是弃置顺序）的阵列 198。在该例中，每个跟踪缓冲器都有一个独有的线程标识符或者向线程标识符的一对一的映射。例如，跟踪缓冲器 114A 被赋予线程标识符 1，跟踪缓冲器 114B 被赋予线程标识符 2，等等。线程标识符可以是硬连线的或者是程序设置的。在一个实施例中，每个程序计数器都与

一个特定的线程标识符和跟踪缓冲器相关联。（另外，也可以没有这种限制关系）。

图 9 表示的是线程在时刻 t1 和时刻 t2 的弃置顺序的一个例子。该例中，只有四个跟踪缓冲器和四个线程标识符。相关的线程号在括号中表示。根据具体的实现，括号中的线程号实际上并不包含在阵列 198 中。在时刻 t1，程序和弃置顺序是线程 T1、T3、T2 和 T4—如同图 6 的例子中的一样。在时刻 t1 与 t2 之间，确定线程 T4 不在程序顺序中。因此，将线程 T4 复位，在跟踪缓冲器 114D 中为线程 T5（图中未予示出）腾出空间。将线程 T5 与线程标识符 4 关联。线程 T1 弃置，在跟踪缓冲器 114A 中为线程 T6 腾出空间。将线程 T6 与线程标识符 1 关联。在时刻 t2，程序和弃置顺序是线程 T3、T2、T5 和 T6（如果线程 T1 是在线程 T4 复位之前弃置的，则线程 T5 和 T6 就应当有不同的线程标识符，但程序和弃置顺序则不改变）。根据所使用的算法，有可能线程 T2 一开始在阵列 198 中位于线程 T3 之前，但是程序和弃置顺序应当更正过来—如时刻 t1 时的阵列 198 一样。

前文说过，线程的程序顺序是线程在顺序处理器上执行时的顺序。指令的程序顺序是指令在顺序处理器上执行时的顺序。线程管理逻辑 124 不必在一开始就为线程确定真正的程序顺序。然而，线程管理逻辑 124 终归要确定真正的程序顺序。

参看图 8，跟踪缓冲器 114A、114B、…、114Y 通过与导线 118 相连的导线 118A、118B、…118Y 接收指令。在导线 118A、118B、…118Y 与导线 118 之间可以有多路分解电路。或者，也可以用选通信号来控制激活哪个跟踪缓冲器。再者，也可以有足够的并行导线来处理并行的事务处理。跟踪缓冲器 114A、114B、…、114Y 通过与导线 120 相连的导线 120A、1208B、…120Y，向管道 108 提供指令和用于再执行的相关信息。注意到可以有多个指令从跟踪缓冲器 114 并行地通过导线 120 和 MUX 110 去重新执行。与此同时，可以有多条指令首次从解码器 106 通过 MUX 110。有一个线程标识符和一个指令标识符伴随每个指令通过管道。伴随指令的还可以有一个再执行计数值。就加载荷存储指令来说，伴随指令的也可以有一个加载缓冲器标识符（LBID）和一个存储缓冲器标识符（SBID）。在一个实施

例中，LBID 和 SBID 伴随每个指令，虽然当指令不是加载（load）或存储（store）时 LBID 和 SBID 的值可能没有意义。如下文所述的那样，一个 PRID 或值也可以伴随一条被重新执行的指令。

跟踪缓冲器 114A、114B、…、114Y 通过与导线 126 相连的旁路导线 126A、126B、…、126Y 从重命名/分配单元 150 接收 PRID、LBID 和 SBID 值。跟踪缓冲器 114A、114B、…、114Y 通过与导线 122 相连的导线 122A、122B、…、122Y，以及通过与导线 196 相连的导线 196A、196B、…、196Y，接收写回结果信息以及相关信号。再执行信号是通过与导线 194 相连的导线 194A、194B、…、194Y 提供的。在导线 120、126、122、194 和 196 中可以使用多路转换和/或选通电路和/或相当数量的并行导线。各跟踪缓冲器可以完全相同，也可以有所不同。

在图 10 中的跟踪缓冲器 114A，表示的是跟踪缓冲器的第一个实施例。在图 11 中的跟踪缓冲器 114A'，表示的是跟踪缓冲器的第二个实施例。跟踪缓冲器的其它实施例可能包括跟踪缓冲器 114A 和 114A'的变化形式，也可能包括有相当差异的结构。

1. 跟踪缓冲器 114A

参看图 10，跟踪缓冲器 114A 包括一个指令队列阵列 202A、一个数据与从属对象（DAD）阵列 206A、一个输入寄存器堆 208A、一个输出寄存器堆 210A、从属对象生成电路 212A 和控制电路 224A。“阵列”一词广义上包括在多个方向上的信息，没有对特定形式的限制。

a. 指令队列阵列 202A

以下结合图 12，说明按照本发明一个实施例的指令队列阵列 202A 的结构及其与其它部件的交互作用。指令队列阵列 202A 接收从 I-高速缓存提取的、作为某特定线程的一部分的指令。线程内部的指令是按照顺序提取并写入指令队列阵列 202A 的。作为另一个线程的一部分的指令，要写入不同跟踪缓冲器的指令队列，或是在不同的时间写入指令队列阵列 202A。指令队列阵列 202A 包含对应每个指令标识符的各种信息字段。不同实施例可能包含有些不同的字段和不同的行数。在指令队列阵列 202A 这个实施例中，没有考虑程序计数器值，但在其它实施例中可能会考虑。指令队列阵列 202A 和附图中表

示的所有其它部件，包含未予示出的各种字段、信号和结构。这些字段、信号和结构之所以未予示出，是因为它们是随具体实现的不同而不同的，本领域的熟练人员理解，它们会使本说明书复杂化，会影响阐明本发明的实质。

指令要在跟踪缓冲器 114A 中一直等待到它们被最终弃置或丢弃（原因例如是，确定该线程不是程序的顺序执行的一部分）。如果指令队列阵列 202A 内容已满，而在跟踪指令集中还有尚未执行的指令，则跟踪缓冲器 114 或重命名/分配单元 150 就不接收这种指令—直到有指令从指令队列阵列 202A 弃置并有一行被释放。通过移动首部和尾部指针，可以对系统 100 中各种阵列的条目进行分配和释放。

对指令队列阵列 202A 的说明，结合了以下各行代码:

```
I0:   mul R1,R2→R1
I1:   mul R3,R4→R2
I2:   add R1,R2→R1
I3:   add 10,R1→R4
I4:   store R2→Mx
I5:   store R1→My
```

这些代码是某线程内的前 6 条指令。显然，按照程序顺序，某个不是跟踪缓冲器 114A 的跟踪缓冲器，要早于跟踪缓冲器 114A。

“操作码”字段含有特定指令相关的操作码。“目的地”、“源 1”和“源 2”字段，标识指令的目的地、源 1 和源 2。“源 1 的索引”字段标识包含源的跟踪缓冲器 114A 内的指令项。例如，指令标识符 0 的目的地被用作指令标识符 2 的源 1。所以将 0 放置在指令标识符 2 的“源 1 的索引”字段。指令标识符 2 的目的地被用作指令标识符 3 的源 2。所以将 2 放置在指令标识符 3 的“源 2 的索引”字段。X 表示无所谓。

“有效 1”和“有效 2”字段是数个位，当某指令标识符的对应源操作数以前已经被跟踪缓冲器 114 中的线程外部的指令产生时，数位被设置成第一值（例如逻辑 0），当某指令标识符的对应源操作数以前已经被线程内部的指令产生时，数位被设置成第二值（例如逻辑 1）。指令标识符 0 的源 1（R1）是在指令队列阵列 202A 中的跟踪指令集的外部产生的。相应地，指令标识符 0 的有效 1 是逻辑 0。

指令标识符 3 的源 2 来自指令标识符 2 的目的地。相应地，指令标识符 3 的有效 2 是逻辑 1。

指令 I3 要将 R1 加上常数“10”。该常数可以与指令存储在一起，可以存储在某个专用寄存器（未予示出）中，存储在源 1 字段中，或是通过其它某种机构存储。图 2 中显示，指令标识符 3 的源 1 字段中是 X（无所谓）。或者也可在源 1 字段中放置某种标志符。

存储缓冲器标识符（SBID）字段保存一个与存储缓冲器中的存储（store）指令相关联的 SBID，这将在下文作说明。加载缓冲器标识符（LBID）字段保存一个与加载缓冲器中的加载（load）指令相关联的 LBID 项，这将在下文作说明。SBID 和 LBID 的值由重命名/分配单元 150 分配，并通过旁路导线 126 被写到指令队列阵列。指令队列阵列 202A 中可以包含一个线程标识号字段，但是并不需要有线程标识号字段，因为它是隐含的。

b. DAD 阵列 206A 和从属生成电路 212A

参看图 13，一个 DAD 阵列 206A 实施例中包括与指令队列阵列 202A 的各指令标识符项形成一一对应关系的各“指令标识符”项（行）。指令队列阵列 202A 与 DAD 阵列 206A 确实也可以是同一个阵列的不同部分。然而，在有些实施例中，有不同的读取端口与指令队列阵列 202A 和 DAD 阵列 206A 相关联。

DAD 阵列 206A 包含一个“值或 PRID”字段，它含有某指令生成的值或者寄存器堆 152 中的 PRID。值是通过写回单元 162 和写回总线 122 和 196 从执行单元被写回到跟踪缓冲器 114A 的。一个占两位的“状态”字段，指示“值或 PRID”字段中含有的是一个“值”还是“PRID”。在一个实施例中，“值或 PRID”字段中有可能既不含“值”也不含“PRID”。“再执行计数”字段能唯一地标识一次指令发送，每当具有相同指令标识符的指令在管道 108 中被再执行时，它就递增。按照一个实施例，指令有可能在管道 108 中被同时再执行一次以上。在这种情况下，按照一个实施例，只将最高“再执行计数”的相关信息写回到 DAD 阵列 206A。

“从属字段”中的各位对应每个逻辑寄存器。为简明起见，图 13 中只表示了 4 个逻辑寄存器（R1、R2、R3 和 R4）。然而，这个数字可以更大。在该例中，当从属字段项被设置为 1 时，指示在跟踪指

令集和指令项的输入值之间存在一个数据从属链和如果没有从属为零。从属字段项确定的是，如果收到输入值（例如检测到值误推测时），跟踪指令集中的哪个指令需要得到执行。

随着指令被提取、解码并写入跟踪缓冲器 114A，从属位被顺序计算，并写入 DAD 阵列 206A。从属位可以在确定是否再执行某指令之前就生成。图 13 中的从属位对应于在上文的 B.1.a 一节中引用的 6 条指令 I0-I5。

从属字段可以通过机械方法（mechanical approach）来创建。在描述这种方法之前，将在更直觉的知识层次解释这种创建。

i. 直觉知识层次

指令 I0 的结果只依赖寄存器 R1 和 R2。所以，在指令标识符 0（该行保存指令 I0 的相关信息）的 R1 和 R2 列中放置 1，R3 和 R4 列中依然是 0。

指令 I1 的结果只依赖寄存器 R3 和 R4。所以，在指令标识符 1 的 R1 和 R2 列中放置 0，在 R3 和 R4 列中放置 1。

指令 I2 的结果直接依赖分别在指令 I0 和 I1 中生成的寄存器 R1 和 R2 的内容。在指令 I0 中，R1 依赖在跟踪指令集开始时的 R1 和 R2 的值。在指令 I2 中，R2 依赖在跟踪指令集开始时的 R3 和 R4 的值。所以，指令 I2 间接地依赖在跟踪指令集开始时的 R1-R4 的值，于是在指令标识符 2 的 R1-R4 各列中放置 1。

指令 I3 的结果直接依赖在指令 I2 中生成的寄存器 R1 的内容。所以，指令 I3 间接地依赖在跟踪指令集开始时的 R1-R4 的值—因为 I2 依赖这些值，于是在指令标识符 3 的 R1-R4 各列中放置 1。

指令 I4 的结果直接依赖在指令 I1 中生成的寄存器 R2 的内容。R2 则依赖在跟踪指令集开始时的 R3 和 R4 的值。所以在指令标识符 4 的 R1 和 R2 列中放置 0，在 R3 和 R4 列中放置 1。

指令 I5 的结果直接依赖在指令 I2 中生成的寄存器 R1 的内容，而 I2 则依赖在跟踪指令集开始时的 R1-R4 的值。所以在指令标识符 4 的 R1-R4 各列中放置 1。

ii. 一种机械方法

以下是可用于按照本发明一个实施例生成从属字段的寄存器和算法。参看图 14，从属对象生成电路

(dependency generation circuitry) 212A 含有对应每个逻辑寄存器的临时寄存器 230、232、234 和 236，外加寄存器 240。临时寄存器 230、232、234 和 236 含有逻辑寄存器 R1、R2、R3 和 R4 的修改量(modifiers)。修改寄存器(modified register)240 含有的各个位，指示哪个逻辑寄存器要被某跟踪指令集(trace)中的指令修改。每当有新指令写入跟踪缓冲器时，寄存器 230、232、234、236 和 240 就被更新。寄存器之间的边界有点任意。例如，它们可以都在一个组合寄存器中。

为各逻辑寄存器提供一个跟踪缓冲器地址寄存器，它指向跟踪缓冲器114A修改该逻辑寄存器的最后指令。各修改位(modified bits)及最后修改量地址(last modifier addresses)，被用于计算下一条要写入跟踪缓冲器 114A 的指令的从属位。

注意，在本文中，修改寄存器只是指将某个值写入寄存器。这并不一定指寄存器的内容由于指令的结果而改变。例如，如果将 R1 和 R2 的内容相乘(如指令 I0 中的那样)并将结果写入寄存器 R1，R1 的内容未必会由于指令 I0 的结果而改变。例如，如果在指令之前 R1 的内容为“0”或 R2 为“1”时，则指令之后的 R1 的内容就不会改变。

图 16 中的流程图 250 代表一个算法，对指令的每个源操作数(例如源 1 和源 2)执行该算法，能创建 DAD 阵列 206A 的从属字段。在步骤 252 中，判断寄存器 240 中的相关位是否置位。如步骤 254 所示，如果寄存器 240 中的该位没有置位，则将该寄存器所关联的从属字段中的该位设置为逻辑 1。如步骤 258 所述，如果寄存器 240 中的该位置位，就用从该相关寄存器的对应索引寄存器(230、232、234 或 236)创建的索引(index)来读取源从属字段。然后，如步骤 262 所述，用逻辑“或”操作将这些源从属位与当前指令从属位合并。这种逻辑“或”操作由图 15 中的“或”门(OR gate)244 表示(其中多个位在输入端出表示)。在执行图 16 的算法时，修改寄存器和所涉及的修改量，是恰好在某指令被执行之前就存在的。

就 I0 而言，在指令 I0 之前，寄存器 240 中对应 R1、R2、R3 和 R4 的都是逻辑 0，寄存器 230、232、234 和 236 的值为 X(无所谓)。执行步骤 252 时，寄存器 240 中对应 R1 和 R2 的修改位每个都是 0。

所以，执行步骤 254 后，DAD 阵列 206A 的指令标识 0 所在行中对应 R1 和 R2 的从属字段位每个都被设置为 1。寄存器 R3 和 R4 涉及不到，在指令标识 0 所在行中依旧是 0。指令 I0 修改寄存器 R1。所以将 0 放置在寄存器 230 中，表示指令 I0 是最近一次修改寄存器 R1 的指令。寄存器 232、234 和 236 的值依旧为 X(无所谓)。将寄存器 240 的 R1 位设置为 1，表示 R1 被跟踪指令集内部的指令修改过。

指令 I1 的从属字段的生成，类似于 I0 的从属字段的生成。修改寄存器 240 的 R1 逻辑寄存器列依旧被设置为 1。在修改寄存器 240 的 R2 列中放置逻辑 1。寄存器 232 中的 1 代表指令 I1。

就指令 I2 而言，根据步骤 252，在指令 I2 之前，寄存器 240 中对应 R1 和 R2 的修改位各为逻辑 1(即置位)。执行步骤 258，在指令 I2 之前，将 R1 的修改量寄存器(230)和 R2 的修改量寄存器(232)用作索引。寄存器 230 含有表示指令 I0 的 0。DAD 阵列 206A 的指令标识 0 中的指令 I0 的从属字段是 0011。寄存器 232 含有表示指令 I1 的 1。指令标识 1 中的指令 I1 的从属字段是 1100。根据步骤 262，0011 和 1100 的逻辑“或”是 1111。所以，将 1111 放置到 DAD 阵列 206A 中对应的指令标识 2 的从属字段中。R1 被指令 I2 修改。不过，寄存器 240 中已经有了表示寄存器 R1 的 1。将 2 放入寄存器 230 中，表示指令 I2 是最近一次修改寄存器 R1 的指令。

指令 I3 的从属字段的生成，类似于 I2 的从属字段的生成。将逻辑 1 添加到修改寄存器 240 的 R4 列，将代表指令 I3 的 3 放入寄存器 236 中。逻辑“或”的结果是 1111。

就指令 I4 而言，根据步骤 252，在指令 I4 之前，寄存器 240 中对应 R2 的修改位被设置为 1。执行步骤 258，在指令 I4 之前，用 R2 的修改量寄存器(232)用作作为索引。寄存器 232 含有表示指令 I1 的 1。DAD 阵列 206A 的指令标识 1 中的指令 I1 的从属字段是 1100。根据步骤 262，1100(指令标识 1 的源)与 0000(没有源 2)的逻辑“或”是 1100。所以，将 1100 放置到 DAD 阵列 206A 中对应的指令标识 4 这一行的从属字段中。

指令 I5 的从属字段的生成，类似于 I4 的从属字段的生成。指令 I5 和 I6 修改外部存储器位置，不会引起寄存器 230、232、234、236 或 240 的变化。

从属信息可以被调度/发布单元 156 使用，要不然调度/发布单元 156 可只推导其自己的从属信息。

在再执行中，可以有不同的方法从跟踪缓冲器 114A 发出一系列或一串指令。一种方法是顺序地读跟踪缓冲器，提取从属位被置位的那些指令，并把发送它们，供再执行。然而，零可能有在管道中产生冒泡(bubbles)的作用。另一种方法是，在发送供执行/再执行的指令之前，通过打包逻辑(packing logic)去除冒泡。参看图 17，另一种方法要使用另外某种硬件，包括对应每个逻辑寄存器的阵列 268。阵列 268 包含从属于寄存器 R1 的各指令的指令标识值。指令 268 中的值起着指向指令队列阵列 202A 中全部指令标识项的指针的作用。这就便于对指令缓冲器的快速读取。每次读取一个指令块(可能有 2 条或 4 条指令)。跟踪缓冲器 114A 可能是多端口的(multi-ported)，有四个解码器，它将从寄存器阵列获得的这些索引每一个都传入各解码器，于是就能在一个周期读出指令 I0、I2、I3 和 I5。可以在再执行开始之前、创建从属字段的时候对寄存器 R1 阵列进行汇编。间接的水平（level of indirection）利用了高带宽再执行。

c. 输出寄存器堆 210A 和输入寄存器堆 208B

跟踪缓冲器 114A 包括检测某些推测错误的检测电路。按照本发明的一个实施例，每个跟踪缓冲器都有一个输出寄存器堆，用于保存相关线程的寄存器环境(register context)，一个输出入寄存器堆，用于接收按程序顺序的前一个线程的寄存器环境。寄存器环境是逻辑寄存器的内容或状态。输出寄存器堆的内容经常被更新，也许每次寄存器中有变化时就更新。输入寄存器堆的内容只有在某个比较之后才被更新—下文将作说明。

图 18 和 19 表示的是(跟踪缓冲器 114A 中的)输出寄存器堆 210A 和(跟踪缓冲器 114B 中的)输入寄存器堆 208B 的实施例—尽管可以使用其它实施例。输出寄存器堆 210A 和输入寄存器堆 208B 中包括一个“值或 PRID”字段和一个状态字段。状态字段指示在“值或 PRID”字段保存的是一个有效值还是一个有效 PRID。在一个实施例中，要么有有效值，要么有有效 PRID。在另一个实施例中，可能二者都没有，在这种情况下，依赖输入寄存器堆的指令会等待一个。

注意，在上面说明的例子中，指令 I0 要使用寄存器 R1 和 R2，

它们哪一个都不曾是包含指令 I0 的线程内某个指令的目的地。然而，却要能从输入寄存器堆 208B 得到寄存器 R1 和 R2 对应的值或 PRID。

参看图 20，比较器 280B 将当前线程的(跟踪缓冲器 114B 中的)输入寄存器堆 208B 的内容，与按程序顺序的前一个线程的(跟踪缓冲器 114A 中的)输出寄存器堆 210A 进行比较。比较可以在前一个线程的执行结束时进行，或者在前一个线程的原始执行期间进行。比较也在前一个线程的弃置(retirement)结束时进行。在一个实施例中，仅在前一个线程的弃置结束时进行比较。

有各种事件会触发由比较器 280B 进行的比较。进行比较是为了检测推测错误。如果输入和与输出寄存器堆之间存在差别，则表明前一个线程的输出寄存器的一个或多个值已经变化。作为响应，输入寄存器堆 208B 被更新，再执行触发逻辑 284B 促使受影响的指令用变化后的寄存器值再执行。从属字段可能会被再执行触发逻辑 284B 使用。不能保证变化后的值就是最终正确值(即在纯粹顺序处理器中本当产生的寄存器值)。指令可能还需要再次再执行，可能要再执行多次。

在一个实施例中，线程的检测电路包括一个输出寄存器堆、一个输入寄存器堆、一个比较器和相关的控制电路，用于检测在包含该输入寄存器堆的跟踪缓冲器中存放的指令中的某些推测错误。在一个实施例中，检测电路可包含有某些不同的电路。

举例来说，参看图 21，线程 T2 是个当前线程，与跟踪缓冲器 114B 关联。线程 T1 是线程 T2 的前一个线程，与跟踪缓冲器 114A 关联。线程 T1 包含一个函数调用、函数和从函数调用的返回指令。将函数调用时输出寄存器堆 210A 的现有内容复制到输入寄存器堆 208B。根据输入寄存器堆 208B 中的寄存器环境，对线程 T2 的指令进行推测性的执行。在到达返回指令时，由比较器 280B 对输入寄存器堆 208B 的内容与输出寄存器堆 210A 的内容进行比较。如果有差别，就更新输入寄存器堆 208B 并将线程 T2 中受影响的指令再执行。这种比较也可以在中间的一个或多个时刻进行。这可以防止由于更加均匀分布的指令再执行而产生的瓶颈问题，但是，如果—例如—输出寄存器堆的内容在该功能期间不止一次变化时，又会导致更多的再执行。如果输出寄存器堆总是发生变化，则最好有一个接收输出寄存器堆

210A 内容的中间缓冲器。然后可以对中间缓冲器与输入寄存器堆 208B 的内容进行比较。

如图 8 和 10 中所示，寄存器环境是通过导线 216 在输出寄存器堆和输入寄存器堆之间传递的。导线 216 将每个输入寄存器堆与可能存放前一个线程的跟踪指令集的每个跟踪缓冲器的输出寄存器堆连接。如果能保证程序顺序总是遵循特定的跟踪缓冲器顺序，则导线 216 的布局就可以相当简单。输出和输入寄存器堆可以由图 10 和 11 中所示的控制电路 224A 来控制。

因为输出和输入寄存器堆将提供一个值或者一个 PRID，所以，输入寄存器堆接收内容的时间与能用该输入寄存器堆中某个寄存器作为源操作数来执行指令的时间之间的延迟可能非常短。如果某个值得不到，则可以将到达寄存器堆 152 的 PRID 用于管道 108 中的执行。

预计，随着正确的源操作数从各种线程的寄存器堆出现，许多指令都要被再执行若干次。不过也可预计，对于许多程序来说，有大量的指令要么根本不需要再执行，要么再执行相对较少的次数，这大大提高了单位时间得到正确执行的指令的数量，减少了运行程序所需的总体时间。

2. 跟踪缓冲器 114'

参看图 11，跟踪缓冲器 114'类似于(图 10 中的)跟踪缓冲器 114。不过，在跟踪缓冲器 114'中，从属字段是在决定了要再执行某指令之后、在从属生成与解码电路 218A 中生成的。尽管这会导致再执行时的某些初始延迟，但是如果对用于再执行的指令的发布以及对从属对象的确定是以管道方式进行的，则一旦过程开始后，就很少有另外的延迟。

在一个实施例中，从属生成与解码电路 218A 仅用一个字段来存放从属信息。(在图 13 中，有 4 个字段。)该同一个字段可以被重复使用。例如，在从属于寄存器 R1 的指令的再执行期间，可用该字段来列举从属于寄存器 R1 的指令。在从属于寄存器 R2 的指令的再执行期间，可用该相同字段来列举从属于寄存器 R2 的指令，如此等等。从属生成与解码电路 218A 可以只包含一个修改量字段和一个索引寄存器。(在图 14 中有 4 个。)或者，从属生成与解码电路 218A 也可

以包含多个变址字段和寄存器。从属生成与解码电路 218A 一次只能确定一些指令的从属对象(dependencies)。

数据阵列 214A 包括对应每个指令标识项的“值或 PRID”字段、状态位字段和再执行计数字段(如图 10 和 13 中的 DAD 阵列 206A 一样)。或者也可以将数据阵列 214A 的内容放入从属生成与解码电路 218A 中，不再需要数据阵列 214A。让数据阵列 214A 和从属生成与解码电路 218A 之间保持独立是有益的，理由有二。首先，它们可以包括不同读端口。其次，在一个实施例中，从属生成与解码电路 218A 中没有指令队列阵列 202A 和数据阵列 214A 中的那么多的行数。换言之，在一个实施例中，从属生成与解码电路 218A 重复使用各行，就像它能重复使用从属字段一样。所以，有许多可能。

正如以下将要更详细地说明的那样，MOD 178 通过导线 194 发出加载指令何时要被再执行的信号。可以生成一个具有一个从属字段(如图 13 中 R1 的存储字段一样)的阵列，列出从属于该待再执行的加载指令的指令。然而，对于加载指令来说，该从属指令列表以该加载指令作为开始，而不是像寄存器的情况中那样，以跟踪指令集中的第一条指令作为开始。加载指令的从属字段可以放在(图 11 中的)从属生成与解码电路 218A 中。(当然，其它跟踪指令集的加载指令要从其它跟踪缓冲器取出后被再执行。)在一个实施例中，从属生成与解码电路 218A 被用作加载指令和寄存器这二者的从属字段。同一个字段可用于二者。在另一个实施例中，寄存器的从属字段在 DAD 阵列 206A 中，加载指令的从属字段在从属生成与解码电路 218A 中。

在另外一个实施例中，加载指令是被完全再执行的(即该加载指令之后的所有指令都被重新执行)，这样就不需要从属字段。

C. 一个再执行顺序算法

当再执行触发逻辑(诸如再执行触发逻辑 284B)确定某个源操作数(或其它输入值)被误预测了时，它就触发相应的跟踪缓冲器(诸如跟踪缓冲器 114B)发布要在管道 108 中再执行的那些直接或间接从属于该被误预测源操作数的指令。可以由跟踪缓冲器中 DAD 阵列的从属字段或者通过如图 11 中所示的另一个阵列来确定直接或间接从属的指令。

所确定的指令被按指令在跟踪缓冲器中的顺序(就是按程序顺序)

从跟踪缓冲器中调度出来去执行。例如，对指令标识 0 项中的指令的调度，在指令标识 0 项中的指令之前或者同时。然而，可以在调度/发布单元 156 的控制下，对这些指令进行无序的执行—就像在任何无序处理器中一样。将控制位附在从跟踪缓冲器发出的指令上，向重命名/分配单元 150 指出是否要(1)进行寄存器重命名，(2)放弃在重命名/分配单元 150 中进行重命名别名表查找，而是使用相应的跟踪缓冲器中的 PRID，或者(3)彻底放弃重命名，取 DAD 阵列中的值，把它当作指令中的常数操作数来使用。

正如结合图 12 所作的解释一样，“有效 1”和“有效 2”字段内的位，在指令的相应的源操作数已经由跟踪缓冲器 114A 中的线程外部的指令(例如指令的目的地)产生时，被设置成第一值(例如逻辑 0)；在指令的源操作数已经由线程内部的指令产生时，被设置成第二值(例如逻辑 1)。

(1)有效位 1。如果指令队列阵列 202A 中的该有效位被设置成逻辑 1，则用源操作数的索引来读取 DAD 阵列 206A 中的相应的值或 PRID。如果 DAD 阵列的状态字段的值位和 PRID 位都不是有效的，则表明源操作数寄存器尚未被重命名。在这种情况下，通过导线 120 和 MUX 110 将指令与具有逻辑 0 值的值与 PRID 状态位一起发出，让重命名/分配单元 150 像通常所作的那样去进行别名表查找(寄存器重命名)。如果 PRID 或值是有效的，它就被与指令一起通过导线 120 和 MUX 110 传送到重命名/分配单元 150，后者对其作出的响应是不进行重命名步骤。

(2)有效位 0。如果某源操作数的对应有效位被设置成逻辑 0，则该输入操作数来自跟踪的外部。源寄存器名被用来访问输入寄存器堆 208A。将输入寄存器堆 208A 中的值或 PRID 连同指令一起传送到重命名/分配单元 150，后者对其作出的响应是不进行重命名步骤。

无论有效位是 0 还是 1，对于每个被调度的指令来说，DAD 阵列 206A 中的值与 PRID 状态字段位都要被复位或保持为逻辑 0。这样能达到两个目的。其一，它能保证将允许在重命名步骤中 PRID 被复制到该项之前被调度的后面的从属指令被按照重命名别名表来重命名，以避免使用跟踪缓冲器 114A 中的过时的 PRID。其二，它也保证指令在最后的执行实例被写回之前一直不弃置，因而使指令仅当在

所有数据误预测都已经被更正时才弃置。

D. 二级或最终弃置

一条指令从跟踪缓冲器 114 中弃置(退出)的时机是，前面所有线程的所有指令都已经弃置并且属于该指令的所有再执行事件都已经被服务。换言之，指令是在能确保该指令已经被用正确的源操作数执行过的时候被最终弃置的。线程是按顺序被弃置的。例如，线程 X 中的指令一直要等到前面所有线程都已经被弃置时(即前面所有线程的指令都已经被弃置时)才能被弃置。线程内部的指令是按顺序弃置的，不过，都作好了弃置准备的指令是可以同时被弃置的。

最终弃置是由最终弃置逻辑 134 控制的。在本发明一个实施例中，最终弃置包括(1)向顺序寄存器堆提交结果；(2)服务中断、例外和/或转移误预测；(3)释放跟踪缓冲器和 MOB 178 的各资源项；(4)通知 MOB 将存储指令作弃置标记并将它们发往存储器。释放各资源项可能涉及移动首部指针。如下文所述，MOB 178 中的存储指令，一直要等到能确定相关数据被复制到数据高速缓存 176 或其它存储器时，才被释放。关于 MOB 178 中的加载和存储指令的最终弃置，在下文作说明。

E. 存储器系统

图 22 表示，图 2 的 MOB 178 的一个实施例包括 MOD 178A、178B、…、178Y，其中 Y 代表 MOB 的个数，与跟踪缓冲器 114 的个数相配。MOD 178A、178B、…、178Y 保存分别保存跟踪缓冲器 114A、114B、…、114Y 中跟踪指令集的加载和存储指令的副本。加载指令被保存在加载缓冲器 182A、182B、…、182Y 中。存储指令被保存在存储缓冲器 184A、184B、…、184Y 中。导线 194 代表各种向 MOB 178 或从 MOB 178 传输信号的导线。再执行导线 194 从 MOB 178 向跟踪缓冲器 114 提供信号，提醒跟踪缓冲器 114 有加载指令要被再执行。控制电路 302 执行各种控制功能。

1. 存储缓冲器和加载缓冲器

图 23 表示的存储缓冲器 184A 的一个实施例，也代表存储缓冲器 184B、…、184Y。可以使用其它各种实施例。存储缓冲器 184A 包含各行的存储缓冲器项的各种字段。每项由一个存储缓冲器标识符(SBID)来标识。重命名/分配单元 150 向每个首次被提取和执行、但

没有开通再执行的存储指令分配一个 SBID 项。该存储指令具有该相同的 SBID 值，一直到最终弃置。例如，在图 23 中，项 SBID 0 被分配给指令 store 0。项 SBID 1 被分配给指令 store 1，如此等等。图 23 中表示了一个 LBID 字段，它存放下文将作说明的“存储 LBID”值。在一个实施例中，当(图 12 中)指令队列阵列 202A 的某项含有存储指令时，指令队列阵列 202A 的 SBID 字段就存放标识含有该存储指令的存储缓冲器 184A 中该项的 SBID，LBID 字段则存放该存储指令的存储 LBID（如果有的话）。SBID 和存储 LBID 通过管道 108 伴随有存储指令。在那个实施例中，LBID 字段可以不包括在存储缓冲器 184A。

指令标识符字段存放指令队列阵列 202A 中的存储指令的指令标识符。线程缓冲器标识符字段在存储缓冲器 184A 和跟踪缓冲器 114A 二者中都是隐含的。操作码字段存放存储指令的操作码。存储地址字段存放存储指令的目标地址。在所示实施例中，该地址是由 AGU 172 生成的。SB 地址有效字段包含一位，指示该存储地址是否是个有效地址。数据字段存放要被存储的数据。数据有效字段包含一位，指示该数据是否有效。可以使用独立的地址和数据有效位，这是因为有效地址的到达时间可能与有效数据的不同。地址和数据二者的到达都早于存储指令的执行。按照一个实施例，数据被作为一部分包含在指令中。弃置字段包含一位，当最终弃置逻辑 134 指出该存储指令应当弃置时，该位置位；当从存储器收到向存储器的存储已经完成的确认时，该位复位。加载和存储指令的弃置，将在下文讨论。再执行计数字段包含一个再执行计数值（它类似于图 13 中 DAD 阵列 206A 的再执行计数字段）。按照一个实施例，存储指令每次只能再执行一次，因而没有再执行计数字段。

图 24 表示的加载缓冲器 182A 的一个实施例，也代表加载缓冲器 182B、…、182Y。可以使用其它各种实施例。加载缓冲器 184A 包含各行的加载缓冲器项的各种字段。每项由一个加载缓冲器标识符(LBID)来标识。重命名/分配单元 150 向每个首次被提取和执行、但没有开通再执行的加载指令分配一个 LBID 项。该加载指令具有该相同的 LBID 值，一直到最终弃置。例如，在图 24 中，项 LBID 0 被分配给指令 load 0。项 LBID 1 被分配给指令 load 1，如此等等。(LBID

项号和 SBID 字段可以被称作 MOB ID)。图 24 中表示了一个 SBID 字段，它存放下文将作说明的“加载 SBID”值。在一个实施例中，当(图 12 中)指令队列阵列 202A 的某项含有加载指令时，指令队列阵列 202A 的 LBID 字段就存放标识含有该加载指令的加载缓冲器 184A 中该项的 LBID，SBID 字段则存放该加载指令的加载 LBID（如果有的话）。LBID 和 SBID 伴随加载指令通过管道 108。在该实施例中，SBID 字段也可能不包含在加载缓冲器 182A 中。

指令标识符字段存放指令队列阵列 202A 中的加载指令的指令标识符。线程缓冲器标识符字段在加载缓冲器 182A 和跟踪缓冲器 114A 二者中都是隐含的。操作码字段存放加载指令的操作码。加载地址字段存放加载指令从其加载的地址。项有效字段包含一位，指示该项被一个有效的加载指令占据。在所示实施例中，没有包含地址有效字段，这是因为该地址已经被 AGU 172 生成。PRID 字段存放来自重命名/分配单元 150 的、指示加载指令在寄存器堆 152 中的目的地的 PRID。SB 命中（SB Hit）、线程标识符合再执行计数字段（如果有的话），可以被看作是状态字段的一部分，下文将结合存储指令的执行来加以说明。

当存储和加载指令被首次从重命名/分配单元 150 接收时，就在存储缓冲器 184 和加载缓冲器 182 中分配存储和加载指令的各项，并在寄存器堆 150 和 ROB 164 中分配用于接收被加载值的寄存器的各项。这些项不会受到第一级弃置，而是像跟踪缓冲器 114 中的各项一样，一直被分配，直到最终弃置。相应地，在再执行时不重新分配各项。如果某存储或加载缓冲器满，来自 I-高速缓存 104 的存储或加载指令就不会穿过重命名/分配单元 150，直到有某项被释放。然而，来自跟踪缓冲器的正被重新执行的加载或存储指令将穿过重命名/分配单元 150。

2. 加载和存储地址的比较

参看图 5，按照程序顺序，线程 T1 中的 store MX 的执行时间早于 load MX 在线程 T2 中执行的时间。然而，由于是并行执行，所以，按照时间顺序，store MX 可能在 load MX 之前或之后执行。如果按照程序顺序，store MX 在 load MX 之前执行，那么对 load MX 的推测性执行，相对于 store MX 的顺序将是正确的。如果按程序顺序先

于 store MX 的所有指令都已经被弃置，则 load MX 肯定将从存储器位置 MX 加载正确值。正确值是如果线程是由顺序处理器运行时会被加载的值。如果按程序顺序先于 store MX 的指令并非全部都已经被弃置，则总是存在着 store MX 的数据不正确的机会。

相反，如果按时间顺序，store MX 是在 load MX 之后执行的，则 load MX 的推测性执行将不是按照相对于 store MX 的正确顺序，不能确定 load MX 将加载正确的值。要是正确的值正好就在存储器位置MX中(或者存储缓冲器项的数据字段存放着store MX,直到store MX 最终被弃置），那也纯属偶然。为了确保执行最终正确，MOD 178 包含了各种保证线程之间的存储器数据的一致性的机制。

a. 加载指令的执行

在加载指令被执行之前，要将其地址与存储指令的地址进行比较，以确定哪个存储指令是最邻近的在先匹配存储指令（CEMSI）。“匹配”指具有与加载指令相同的地址。“在先”指该 CEMSI 按程序顺序来说比加载指令更早。“最邻近”指在该 CEMSI 与要执行的加载指令之间没有其它匹配的存储指令。如果只有在先匹配存储指令，它就是CEMSI。

如果有 CEMSI，加载指令就从 CEMSI 的数据字段读取其数据。如果没有 CEMSI，加载指令就从存储器—诸如数据高速缓存 176（一种二级高速缓存）或主存储器—取得其数据。将来自存储缓冲器 184 或存储器的数据传过 MUX 192 并写入跟踪缓冲器 114 中由线程标识符和指令标识符指定的项。也可以将该数据写入寄存器堆 152 中由 PRID 指定的寄存器。也可以根据具体的高速缓存规则—例如写回（write back）、写过（write through）等等，将数据存储在数据高速缓存 176 中。MUX 192 是个旁路（bypass），因为它能旁路存储器—诸如数据高速缓存 176（一种二级高速缓存）或主存储器。

在一个实施例中，有不同的比较器与每一个存储缓冲器 184 的每项关联，用于将待执行的加载指令的地址与存储指令的地址进行比较。图 25 中的比较器 320 是一个例子，它接收加载指令地址和存储缓冲器 184A 中 SBID 1 项的存储地址。导线 322 以及其它比较器的输出导线与 MOB 控制电路 302 相连。

加载 SBID 指向对待执行的加载指令而言是最邻近的在先存储指

令（CESI）的 SBID。该 CESI 位于与该加载指令具有相同的线程标识符的存储缓冲器中。如果有 CEMSI，则它要么是该 CESI，要么按程序顺序来说比该 CESI 更早。重命名/分配单元 150 跟踪程序中存储和加载指令的顺序并提供 SBID 和 LBID 值。它们可以通过导线 126 被写到跟踪缓冲器 114。按照一个实施例，如果某加载指令没有对应的 CESI，则没有与该指令相关联的加载 SBID。这在跟踪指令集中的第一条存储器指令是加载指令时发生。可以采用各种技术来处理这种情形，包括用重命名/分配单元 150 发送某些信号来指示没有有效的加载 SBID。下文所说明的阵列回绕位（array wrap around bit）可以用于这个用途。

考察按以下程序顺序的存储和加载指令:

store 0

store 1

load 0

store 2

load 1

store 3

store 4

load 2。

LBID 字段中的存储 LBID 值，在存储缓冲器 184A 中表示。SBID 字段中的加载 SBID 值，在加载缓冲器 182A 中表示。例如，LBID 项 1 的 SBID 字段中的 2，表示位于存储缓冲器 184A 中的项 SBID 2 处的存储指令，存放着对应 LBID 项 1 中的加载指令的 CESI。指令 store 0、store 1、store 2、load 0 晚于或早于 load 1。指令 store 3、store 4、load 2 早于或晚于 load 1。

控制电路 302 可以用各种方法来确定哪条存储指令是 CEMSI。现在结合图 27 举例讨论一些方法，图中的存储缓冲器 184A、184B、184C 和 184D 是 MOD 178 中仅有的存储缓冲器，分别与线程 A、B、C 和 D 相关联。假设程序顺序是线程 A、线程 B、线程 C、线程 D。本例中，待执行的加载指令在加载缓冲器 182C 中。有一个 CESI，它在存储缓冲器 184B 中。

导线 342、344、346 和 348 是各种比较器的输出导线。导线 362、

364、366 和 368 提供使比较器能进行比较的控制信号。在不同的实施例中，控制电路 302 使能（1）每个存储缓冲器中所有项的比较器；（2）仅仅是某个存储缓冲器中的那些比较器—该存储缓冲器具有的线程标识符与加载指令的线程标识符相同，或者按程序顺序来说早于加载指令的线程标识符；或者（3）仅仅是与按程序顺序来说早于加载指令的各项相关联的那些比较器。

匹配确定逻辑 356 确定存储指令中哪条是 CEMSI。在图 27 中，存储缓冲器 184C 上部的 store MX 指令是 CEMSI。假设该 store MX 不在存储缓冲器 184C 中的话，则 CEMSI 就应该是存储缓冲器 184B 中的 store MX 指令。在比较器和匹配确定逻辑 356 确定是否有 CEMSI 的同时，可以在数据高速缓存 176（或其它存储器）中进行查找，为如果没有 CEMSI 的情况作好准备。匹配确定逻辑 356 包含数据路径控制逻辑 390，它在导线 370 上提供控制信号，控制 MUX 192 从存储器还是存储缓冲器传送数据。

按照一种方法，有两种由 MOB 控制电路 302 作出的优先权决定。一种是决定存储缓冲器内存储指令的优先权。另一种是决定存储缓冲器的优先权。决定的先后顺序可以随便。在对存储缓冲器内的优先权的决定中可以使用一种进位链结构。例如，在一个实施例中，对于除具有与加载指令的相同的线程标识符的存储缓冲器以外的每个存储缓冲器，确定哪条匹配的存储指令是按程序顺序来说最年轻的，如果有的话。对于具有与加载指令的相同的线程标识符的存储缓冲器，如果有的话，确定哪条指令按程序顺序来说最邻近（包括等于）CESI。然后，确定那些存储缓冲器中的哪一个含有一个其线程标识符按程序顺序来说最接近该加载指令的线程标识符的匹配指令。

存储缓冲器 184 可以是各有一个首部和尾部的环形阵列。一开始，具有较大 SBID 值的存储指令是较年轻的。然而，随着存储项的释放和分配，尾部最终将回绕，使得首部指向一个比尾部指向的更高的 SBID 项。在一个实施例中，一个回绕位在尾部从最高变成最低 SBID 值时反转，并被提供给最邻近的匹配确定逻辑 356。

b. 存储指令的执行

当存储指令被执行时，要将其地址与加载指令的地址进行比较，

以确定（如果有的话）按程序顺序来说较晚的（出自相同或较年轻的线程的）加载指令哪些具有与存储指令地址相同的地址。存储 SBID 所指向的最邻近的在后加载指令（CLLI）指定了可以考虑的最早加载指令。

在一个实施例中，有不同的比较器与每一个存储缓冲器 182 的每项关联，用于进行这些比较。其中一个比较器就是图 26 中所示的比较器 324。只是作为例子，将比较器 324 与加载缓冲器 182A 的 LBID 1 项关联。比较器 324 在一个输入端接收一个存储指令的地址，在另一个输入端接收加载缓冲器 182A 中 SBID 1 项的加载地址。输出导线 326 上的信号表示地址是否相同。导线 326 以及其它比较器的输出导线与 MOB 控制电路 302 相连。如下文所述，比较器（诸如比较器 324）也可以比较存储指令的状态位与加载缓冲器中的状态位。

图 28 与图 27 类似。然而在图 28 中，加载缓冲器 182A-182D 中的加载地址是与待执行的存储指令的地址比较的，匹配确定逻辑 356 确定是否要再执行加载指令。在一个实施例中，匹配确定逻辑包括再执行触发逻辑 394，后者在导线 194 上提供信号，向跟踪缓冲器指出哪些加载指令要再执行。在一个实施例中，匹配确定逻辑 356 考察加载指令与自 CLLI 开始的存储指令的匹配。可以采用不同的算法。线程管理逻辑 124 指出按程序顺序来说晚于正在执行的存储指令的线程标识符的那些线程标识符。在一个实施例中，将所有比较器都被使能。在另一个实施例中，仅使能加载缓冲器中具有按程序顺序来说等于或晚于加载指令的线程标识符的线程标识符的导线。在另一个实施例中，仅使能与 CLLI 和较晚的指令相关联的加载缓冲器中的导线。要考察的线程，可以在对加载缓冲器内哪些加载指令按程序顺序来说晚于存储指令的确定之前、之后或期间确定。

按照一个实施例，检测加载指令的执行中的某些推测错误的检测电路，包括与加载缓冲器关联的比较器、匹配确定逻辑 356 的部分、以及相关的控制电路。在其它实施例中，检测电路可包含有些不同的电路。并不要求检测推测错误的检测电路位于执行管道中。不同的匹配确定逻辑都能与数据路径控制逻辑和再执行触发逻辑结合起来使用。

i. 有地址匹配的各种情况

在确定是否再执行时，要考察有地址匹配的那些更年轻指令的状态字段（SB 命中、SBID、线程标识符、再执行计数（若使用的话）。状态字段指出加载指令是从存储器（例如数据高速缓存 176）还是从存储缓冲器的数据字段取得其数据的。例如，如果数据来自存储器，SB 命中字段就含有 0，如果数据来自存储缓冲器，就含有 1。SBID 和线程标识符字段存放数据来自的存储指令的 SBID 和线程标识符。存储指令的线程标识符未必是有地址匹配的加载指令的线程标识符。加载指令的线程标识符在加载缓冲器中是隐含的。再执行计数字段（若使用的话）指示涉及到哪个再执行。（如果 SB 命中是 0，则 SBID、线程标识符及再执行计数诸字段中的数据没有意义）

如果 SB 命中=0（来自存储器的以前数据），就在导线 194 上从加载缓冲器向由加载指令的线程标识符所标识的跟踪缓冲器发出再执行事件信号，于是从跟踪缓冲器再执行该加载指令及所有从属指令。在导线 194 上发送指令标识符和线程标识符，以指示哪条指令被再执行。

如果 SB 命中=1（来自存储缓冲器的以前数据），则 SBID 字段、线程标识符字段和再执行计数字段（如果使用的话）中的值控制是否触发一次再执行。第一种情况是，特定加载指令的状态字段的线程标识符等于存储指令的线程标识符，特定加载指令的状态字段中的 SBID 与存储指令的 SBID 匹配。在第一种情况下，如果存储指令的再执行计数大于状态字段中的再执行计数，就再执行加载指令。如果没有加载计数（因为存储指令每一次只能被再执行一次），就再执行加载指令。

第二种情况是，状态字段中的线程标识符等于存储指令的线程标识符，但是状态字段中的 SBID 与存储指令的 SBID 不匹配。在第二种情况下，如果状态字段中的 SBID 小于存储指令的 SBID，就再执行加载指令，但是，如果状态字段中的 SBID 大于存储指令的 SBID，就不再执行。

第三种情况是，状态字段和存储指令的线程标识符不匹配。预计这是一种不常发生的情况。为简单起见，按照一个实施例，加载指令被再执行（即使它可能违反程序顺序）。这可能是一个假的再执行。加载指令被再执行时，将接收正确的存储数据。可以采用其

它方法，但是它们复杂得多，不值得用于这种不常发生的情况。

ii. 没有地址匹配的各种情况

如果地址不匹配，则不触发再执行—除非是下述不常见的情况。如果 SB 命中=1，状态字段的线程标识符与存储指令的线程标识符匹配，并且状态字段的 SBID 与存储指令的 SBID 匹配。在这种情况下，就有再执行，被再执行的加载指令从新的一项或存储器中接收其数据。

c. 复位

当确定某线程不符合程序顺序时，线程要被复位。然而，其它线程的加载指令可能已经从该线程中的存储指令的相关数据字段取过数据。线程管理逻辑 124 向控制电路 302 发送控制信号。在一个实施例中，当某线程被复位时，要将复位线程的线程标识符与每一个加载缓冲器（除了该复位线程对应的加载缓冲器）中的每一个加载指令进行比较。如果状态字段中的线程标识符与复位线程的线程标识符匹配时，就触发对加载指令的再执行。被再执行的加载指令取自适当的跟踪缓冲器。

3. 存储指令的再执行

如上所述，加载指令是根据存储指令的执行情况而被再执行的。在一个实施例中，存储指令是根据跟踪缓冲器中寄存器的比较结果表明某个寄存器值已经改变而被再执行的。例如，参看图 12 和 13，跟踪缓冲器 114A 的指令标识符 4 和 5 是存储指令，图中显示它们依赖于寄存器 R1-R4。

4. 多个加载指令的再执行

某加载缓冲器中有一个以上的加载指令的状态字段与某个存储指令匹配是可能的。为了避免复杂的逻辑，一种方法是让控制电路 302 检测何时有多重加载地址匹配并促使位于跟踪指令集中最早的加载指令之后的所有指令被重新执行。

5. 加载和存储指令的最终弃置

当某个加载或存储指令要被最终弃置时，最终弃置逻辑 134 向跟踪缓冲器 114 和 MOB 184 提供信号，指出有指令要被最终弃置。（由指令标识符和线程标识符所确定的）跟踪缓冲器中的一项于是被释放。就加载指令而言，（由指令标识符和 LBID 所确定的）加载

缓冲器中的一项被释放。就加载指令而言，最终弃置是彻底的（complete）。就存储指令而言，在释放之前，必须将数据字段中的数据提交到存储器。存储缓冲器中该项的释放以及由此导致的最终弃置，要一直等到收到存储已经完成的确认时才发生。或者，也可以在确认之前使该项最终弃置，但是该项的重新分配则要一直等到收到确认时才能发生。导线 200 上的信号能向线程管理逻辑 124 指出存储指令的弃置何时完成、何时可以开始下一个线程。

“SB 弃置”指示某指令已经被弃置。当最终弃置逻辑 134 指出某指令应当弃置时，SB 弃置字段中的一位就被置位（asserted）。一旦 SB 弃置字段被置位，就将相关的指令写入存储器。一旦 MOB 184A 知道该指令已经被写到存储器，SB 弃置字段马上就被复位（deasserted），该指令就被释放。

加载缓冲器 182A 和存储缓冲器 184A 可以是具有首部和尾部的队列。当有指令被释放时，首部就被移动。在加载缓冲器 184A 以及跟踪缓冲器 114 中，弃置和释放可以同时发生。最终弃置逻辑 134 通过导线 136 和 140 提供信号。Demux 188 选择是否加载缓冲器 182 或存储缓冲器 184 的其中之一将接收弃置信号。Demux 188 是可选的，可由加载缓冲器 182 和存储缓冲器 184 中的使能端口替代。

F. 关于线程管理逻辑和最终弃置逻辑的其它信息

在一个实施例中，线程管理逻辑 124 用一个树结构来追踪线程顺序。按照该树结构，程序顺序（也是弃置顺序）从顶端流向底部，右边的节点按程序顺序来说早于左边的节点。根是程序顺序中的第一个。树是个抽象概念，而树结构则是实现该树的电路系统。

线程在后向转移或函数调用之后的指令位置开始。就是说，线程在假定没有采取后向转移或没有调用函数时的下一条指令的位置开始（如图 4 和 5 中的线程 T2 所示）。这样做时，从某个线程（节点）的角度来看，线程的子节点的程序顺序与线程被启动（创建）的顺序相反。例如在图 6 中，按照时间顺序，线程 T2 的执行在线程 T3 的执行之前开始，但是按照程序顺序，线程 T3 则先于线程 T2 出现。

在一个实施例中，有三种事件可导致将线程从树中去除：（1）位于树的根部的线程在该线程被弃置时被去除。当根部的线程被弃

置时，按程序顺序的下一个线程（节点）变成树根，并相应地重新分配各节点。（2）将按程序顺序来说的处于最后的线程从树中去除，腾出空间，让给要添加到树中的按程序顺序来说位置更高的线程。就此而言，树起着后进先出（LIFO）堆栈的作用。（3）当发现某线程的父线程的程序计数器超出起始计数值至结束计数值的范围时，该线程会被复位，由此而从树中去除。在有子线程是在后向转移的位置被创建（例如图 6 和 29 中的线程 T4）的情况下，起始计数值是该后向转移的目标，结束计数值是在该后向转移指令位置的程序计数器值。在函数调用后面开始的进程也可能由于没有从该函数的返回而被复位—尽管这种情况的发生非常罕见。处理没有从函数返回的这个可能的一种方法是，不管这种可能，而是让系统在该线程变成按程序顺序来说最低的线程时最终将其从树种去除—像在事件（2）中的那样。当线程被从树中去除时，分配给该线程的资源（诸如跟踪缓冲器、存储缓冲器、加载缓冲器）就被释放。

事件（1）和（3）在图 29 中表示，该图包括图 6 的诸例线程，还增加了线程 T5 和 T6。线程 T5 紧接在 J 点的后向转移指令之后开始，线程 T6 紧接在 K 点的后向函数调用转移指令之后开始。假设只有四个跟踪缓冲器。图 30 表示在时刻 t1 时的树结构。线程 T2 被添加到树中的时间早于线程 T3 被添加.到树中的时间。线程 T4 在线程 T3 被添加到树中之后被添加到树中。线程 T2 和 T3 是线程 T1 的子线程。线程 T4 是线程 T3 的子线程。按照由顶到底和自右至左的规则，程序顺序和弃置顺序是线程 T1、T3、T4 和 T2。图 31 表示假定线程 T4 在线程 T1 弃置之前被复位的情况下在时刻 t2 时的树结构。程序和弃置顺序是 T1、T3、T2 和 T5。图 32 表示假定线程 T1 在线程 T4 被复位之前弃置的情况下在时刻 t2 时的树结构。程序和弃置顺序是 T3、T4、T2 和 T5。图 33 表示在线程 T1 弃置和线程 T4 复位后的时刻 t3 时的树结构。程序和弃置顺序是 T3、T2、T5 和 T6。

图 34 所示的事件（2）包含嵌套的函数。按照时间顺序，线程被创建（启动）的顺序是 T1、T2、T3、T4 和 T5。然而，程序顺序是 T1、T5、T4、T3 和 T2。本例中只有四个跟踪缓冲器。所以，并非五个线程都是同时存在的。图 35 表示线程 T5 被启动前的时刻 t1 的树结构。程序和弃置顺序是 T1、T4、T3 和 T2。线程 5 还不是树结构的

一部分。图 36 表示是在线程 T5 已经启动后的时刻 t2 时的树结构。线程 2 是按程序顺序来说最低的，它被从树结构中去除，为线程 T5 腾出空间。被从树中去除的线程可以在以后某个时间重新启动。或者，另一个线程可以执行被从树中去除的线程的部分或全部指令。在一个实施例中，就复位而言，线程可寻求联接以后的下一个线程而不是被复位的线程。或者，该线程也可以一直继续，直到由于其它原因被结束。阵列 198 的函数可以在树的节点中执行。

子线程的线程标识符在树结构中是按照程序顺序适当定位的。（尽管由线程管理逻辑 124 所决定的程序顺序可能会改变。）当某线程联接或对应按程序顺序来说树中的下一个线程的程序计数时，该线程就结束。如果该线程只有一个子线程，它就是按程序顺序来说的下一个线程。例如在图 33 中，线程 T2 是按程序顺序来说的下一个线程。

最终弃置逻辑 134 从树结构获取信息来组装阵列 198 或者直接从树结构的电路获取信息。在线程管理逻辑 124 的树结构和其它逻辑与最终弃置逻辑 134 的逻辑之间可以有解码电路。阵列 198 可以不要。

总之，树结构提供具有至少以下意义的信息：（1）树规定了弃置顺序；（2）树规定了程序顺序，为例如上述的 MOB 178 所使用；（3）树通过指出另一个线程的起始指令而规定了线程的终结点；（4）通过指示那些资源可用和那些资源被释放，树可用于线程资源的分配。

G. 一个没有多线程操作的实施例

图 3 表示一个包含管道 308 的处理器 100。处理器 100 与处理器 50 类似。然而，跟踪缓冲器 300 是唯一的跟踪缓冲器，MOB 310 是唯一的 MOB。处理器 50 不是为处理多个线程而设计的。因此，线程管理逻辑对处理器 100 来说并不需要。例如跟踪缓冲器 300 可以类似于跟踪缓冲器 114A，只是不需要特定于多线程的部件。例如，导线 26 和输出寄存器堆 210 就不需要。可以用各种电路来检测推测错误，包括众所周知的电路。例如 MOB 310 可以类似于 MOB 178A，只是不需要特定于多线程的部件。例如，加载缓冲器中就不需要线程标识符字段。可以对处理器 100 的其它部件，针对它们在处理器 50

中的配置作一定修改，去除与多线程操作有关的特征功能。跟踪缓冲器 300 和 MOB 310 可以用于推测和从推测错误的恢复。跟踪缓冲器便于在管道外部存放大量指令，用于在最终弃置之前进行可能的再执行。

处理器 50 可用于非多线程程序。在这种情况下，线程管理逻辑 124 可以按程序顺序一直保存同一个线程标识符。或者也可以将线程管理逻辑 124 关闭。在非多线程的情形中，只使用跟踪缓冲器 114 的其中之一和 MOB 178 的其中之一。或者也可以将各跟踪缓冲器组合成更大的跟踪缓冲器，将各 MOB 组合成更大的 MOB。

H. 其它信息和实施例

参看图 37，处理器 400 是个包含多管道单元 402 的多处理器(MP)芯片。多管道单元 400 不同于图 2 的共享资源管道 108 的地方是，一个完整的管道(例如每个管道的单独的重命名/分配单元)是与多管道单元 402 的每个管道 0、1、...W 包含在一起的。(W 可对于或大于或小于 X。)否则的话，处理器 400 就会基本上与处理器 50 相同，或者与处理器 50 非常不同。其它处理器可包含多管道单元 402 的一些特点以及管道 108 的一些特点。

本文提及的每一种处理器，都能包含在各种计算机系统的一部分中。参看图 38，举例来说，处理器 50 可以是计算机系统 430 的一部分。相同 430 也可以包含第二个处理器 434。处理器 50 内可包含一个芯片级二级(L2)高速缓存。处理器 50 可通过处理器总线 442 与存储器控制器 440 通信。存储器控制器 440 可通过总线 452 和 454 与主存储器 446 和外围设备 448 通信。

可以在不使用寄存器重命名的处理器中使用一个类似于管道 108 或 308(见图 2 和 3)的管道。在这种情况下，可以修改寄存器重命名所涉及的部件(例如重命名/分配单元 150)，删除与重命名有关的特征功能。

本文所描述和表示的电路和元件只是示例性的。它们可以用其它各种电路和元件替代。此外，在尺寸、延迟等方面可以有各种设计上的平衡。例如，如果执行路径中的(例如保留站、寄存器堆、MOB 中的)缓冲器太大，就只好降低最大操作时钟频率。本文所示的各部件可以根据各种技术来设计和构造。

在两个所示结构之间可能有中间结构(正如缓冲器)或信号。有些导线可能不是向图示中那样是连续的，而是被中间结构隔断的。附图中方框的边界是为了解释的方便。实际的设备不必包含带这种边界的部件。图示各部件的相对大小并不表示实际的相对大小。箭头表示某些实施例中的某种数据流，但没有表示每一个信号，诸如数据请求。上文中如果描述了一个逻辑高信号，该信号也可用逻辑低信号代替，反之亦然。

图示处理器中的各部件可全部位于同一个处理器芯片上。或者，例如跟踪缓冲器也可以位于执行管道以外的不同芯片上。

“连接”、“耦合”等相关的用语，并不限于直接连接或直接耦合，而是包括间接连接和间接耦合。“响应”等相关用语，只一个信号或事件在某种程度上受到另一个信号或事件的影响，这种影响未必是完全或直接的。如果本说明书述及“可以”、“能”或“最好要”包括某部件时，该特定不是必定要包括部件。

MOB 可用数据匹配而不是地址匹配来检测误推测。

阅读本说明书的本领域的熟练人员将知道，在本发明的范围内可以由以上说明和附图中得出各种变通例。所以，界定本发明的是以下各权利要求—包括其补充。

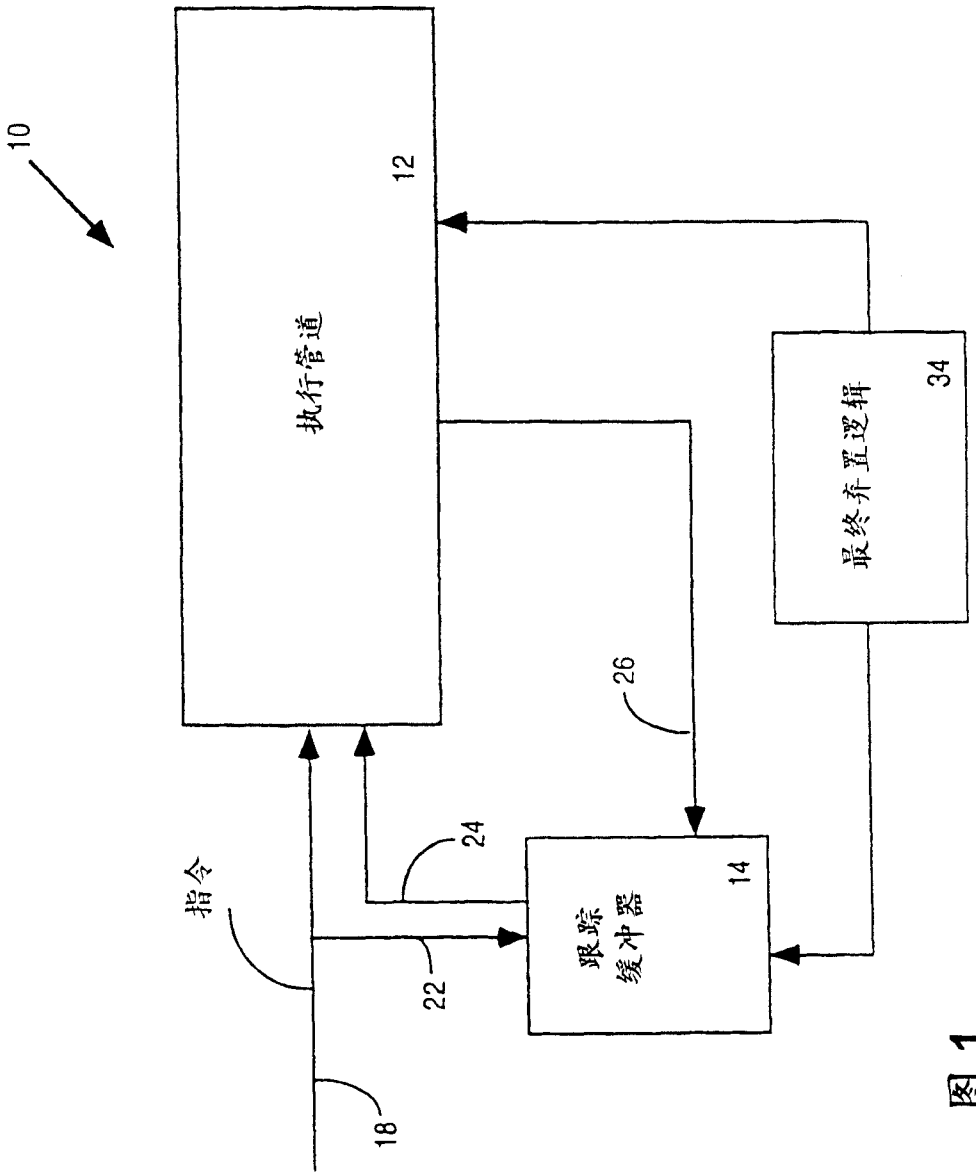


图 1

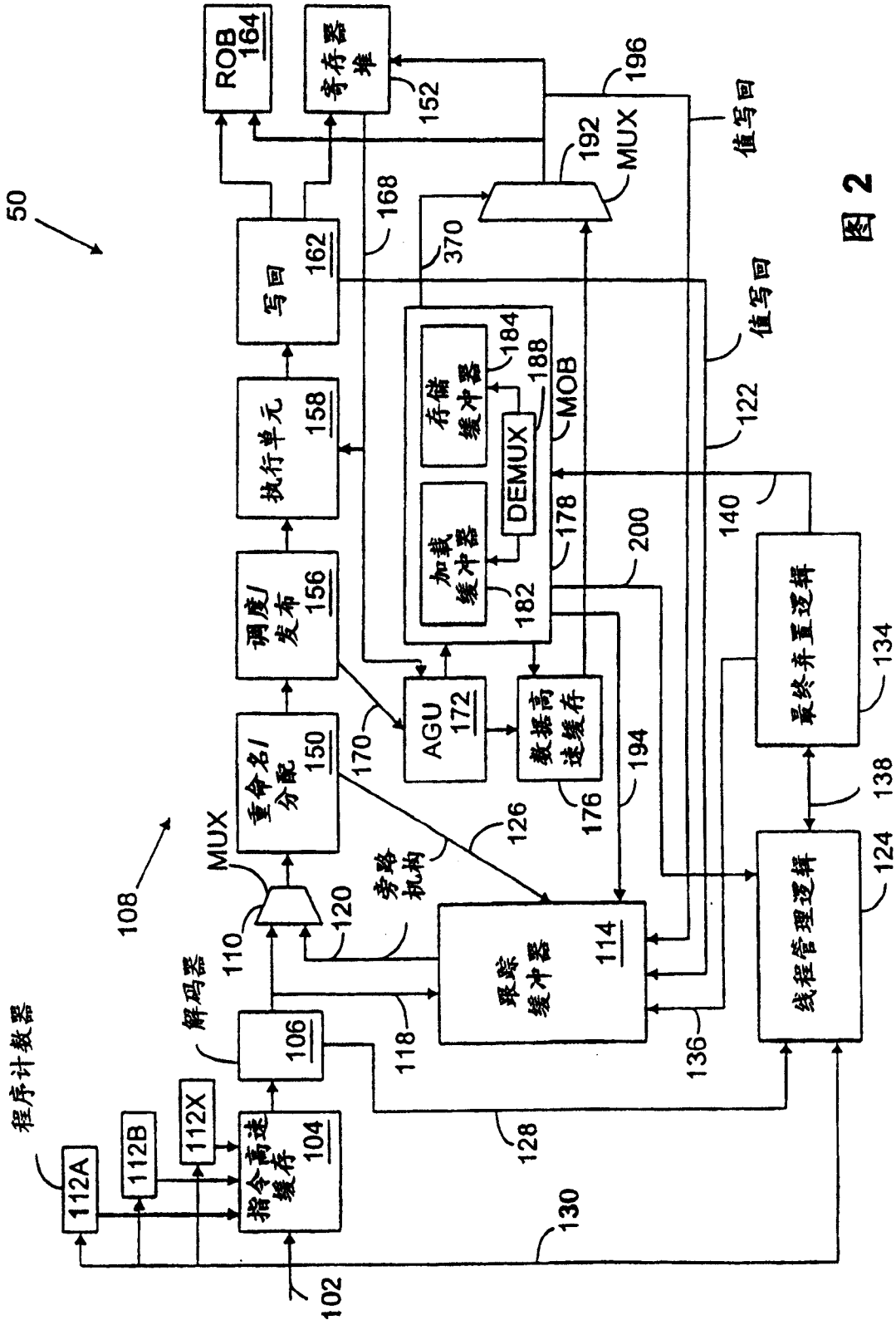

50
程序计数器
112A
112B
112X
指令高速缓存 104
102
解码器
106
108
MUX
110
重命名/分配 150
调度/发布 156
执行单元 158
写回 162
ROB 164
寄存器堆 152
168
MUX 192
196
值写回
370
120
旁路机构
126
AGU 172
170
加载缓冲器 182
存储缓冲器 184
DEMUX
MOB 188
178
200
数据高速缓存 176
194
122
140
值写回
跟踪缓冲器 114
118
128
130
136
线程管理逻辑 124
最终弃置逻辑 134
138

图 2

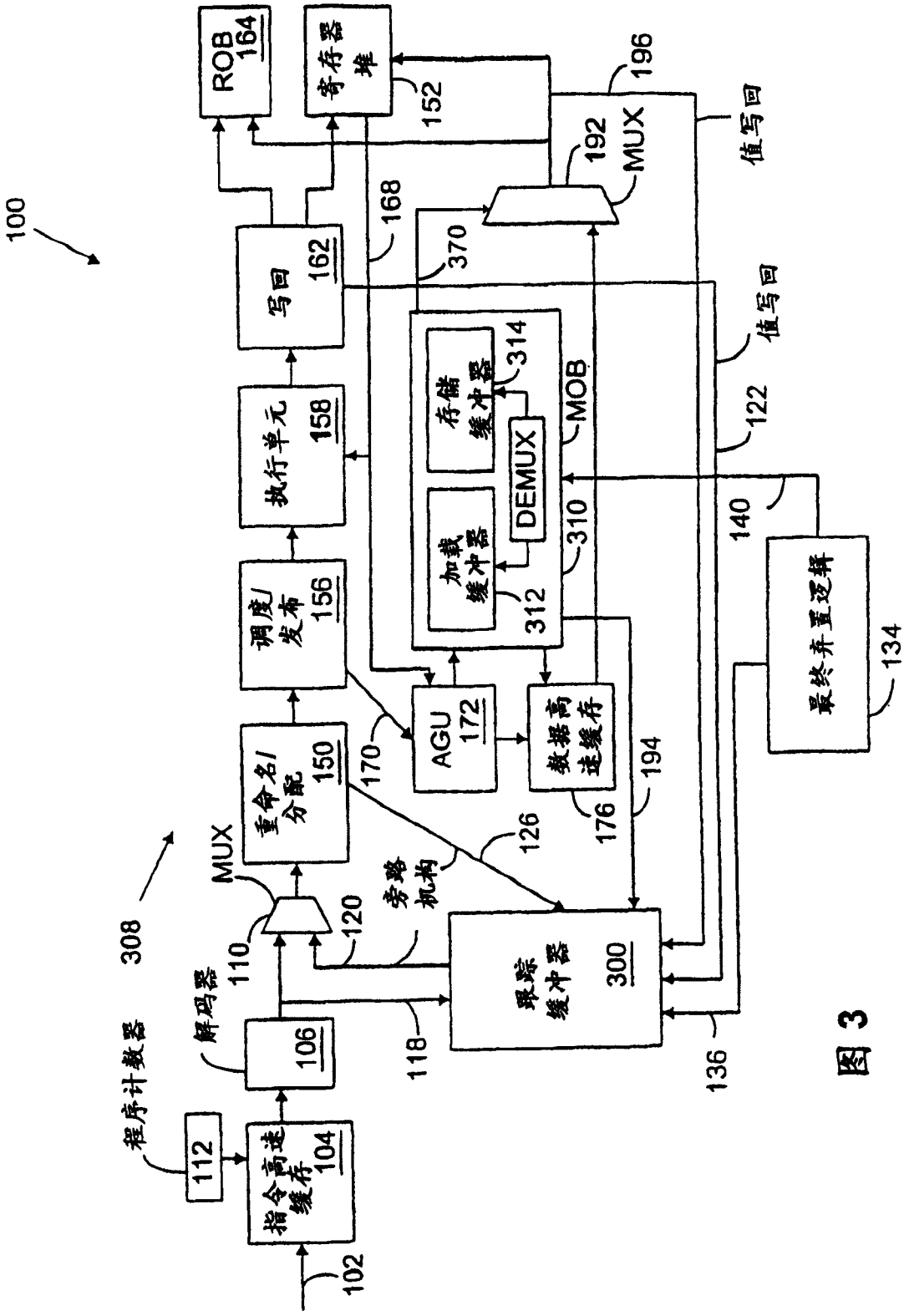

100
ROB 164
寄存器堆
152
196
MUX
192
值写回
写回 162
168
370
执行单元 158
存储缓冲器
314
MOB
DEMUX
加载缓冲器
312
310
122
140
调度/发布 156
AGU 172
数据高速缓存
194
176
最终弃置逻辑
134
重命名/分配 150
170
126
MUX
110
120
旁路机构
308
解码器
106
118
跟踪缓冲器 300
136
程序计数器
112
指令高速缓存 104
102

图 3

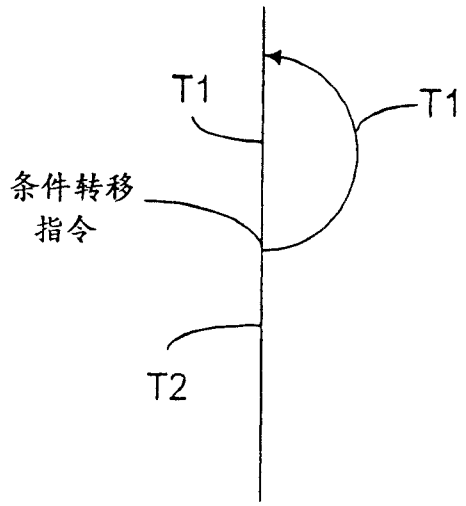


图 4

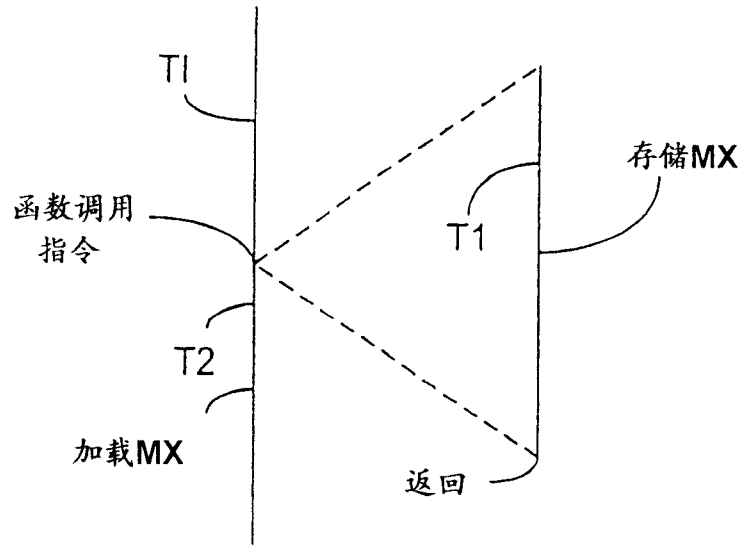


图 5

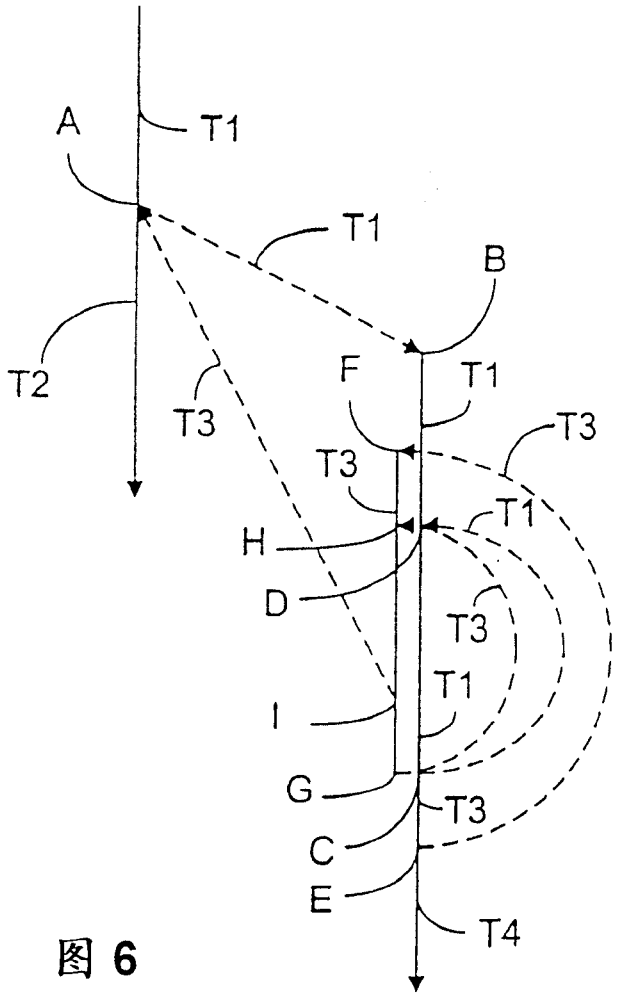


图 6

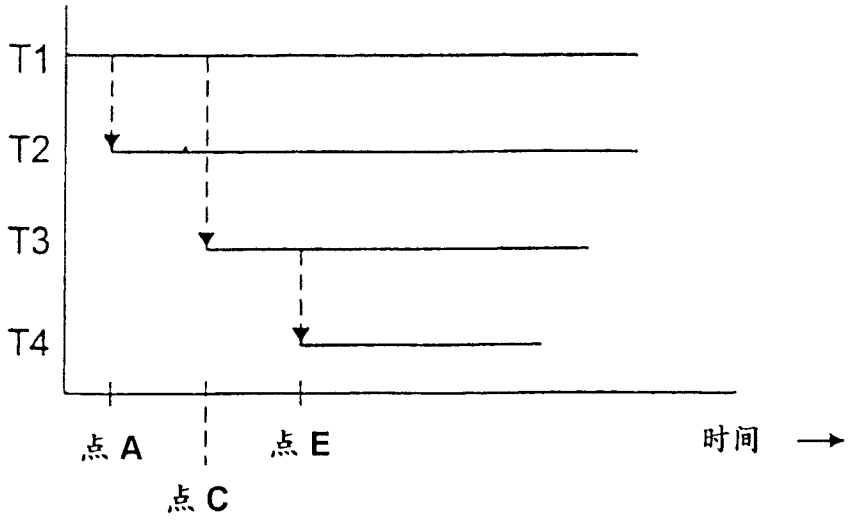


图 7

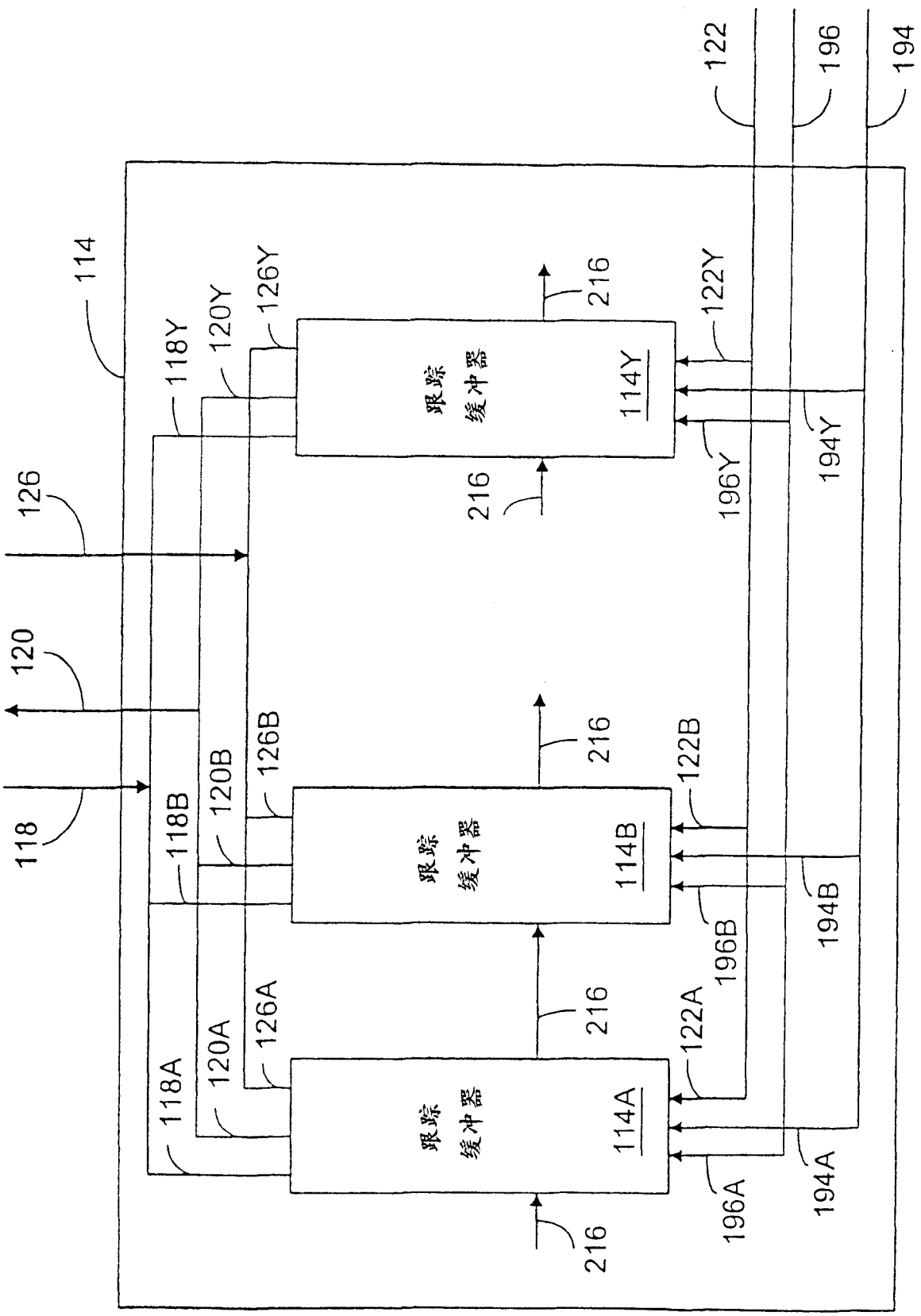

114
118
120
126
118A
120A
126A
118B
120B
126B
118Y
120Y
126Y
跟踪
缓冲器
114A
114B
114Y
216
122
196
194
122A
122B
122Y
196A
196B
196Y
194A
194B
194Y

图 8

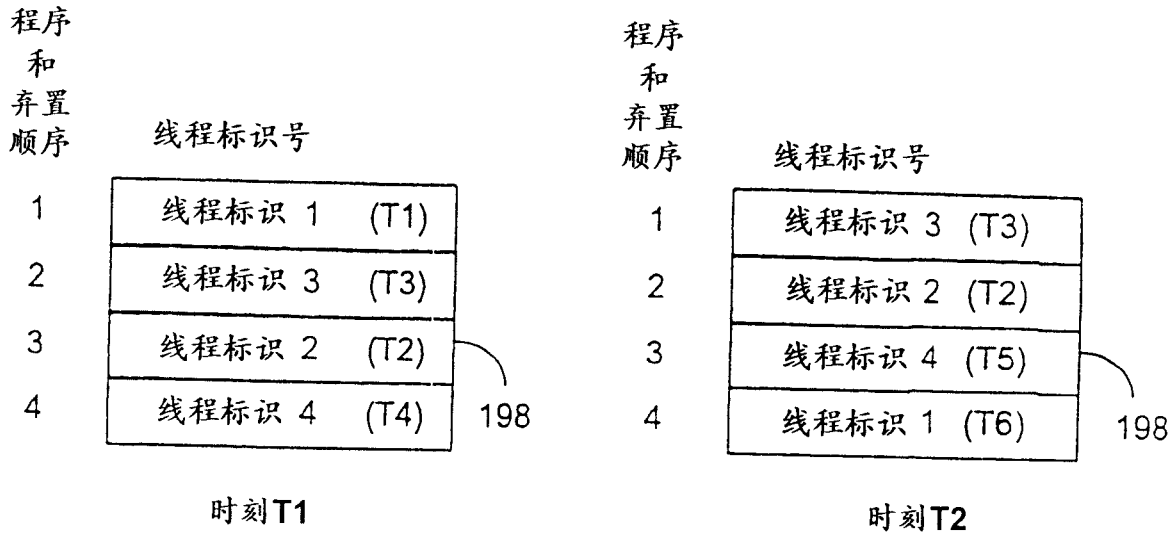

程序和弃置顺序
线程标识号
1
2
3
4
线程标识 1 (T1)
线程标识 3 (T3)
线程标识 2 (T2)
线程标识 4 (T4)
198
时刻T1
程序和弃置顺序
线程标识号
1
2
3
4
线程标识 3 (T3)
线程标识 2 (T2)
线程标识 4 (T5)
线程标识 1 (T6)
198
时刻T2

图 9

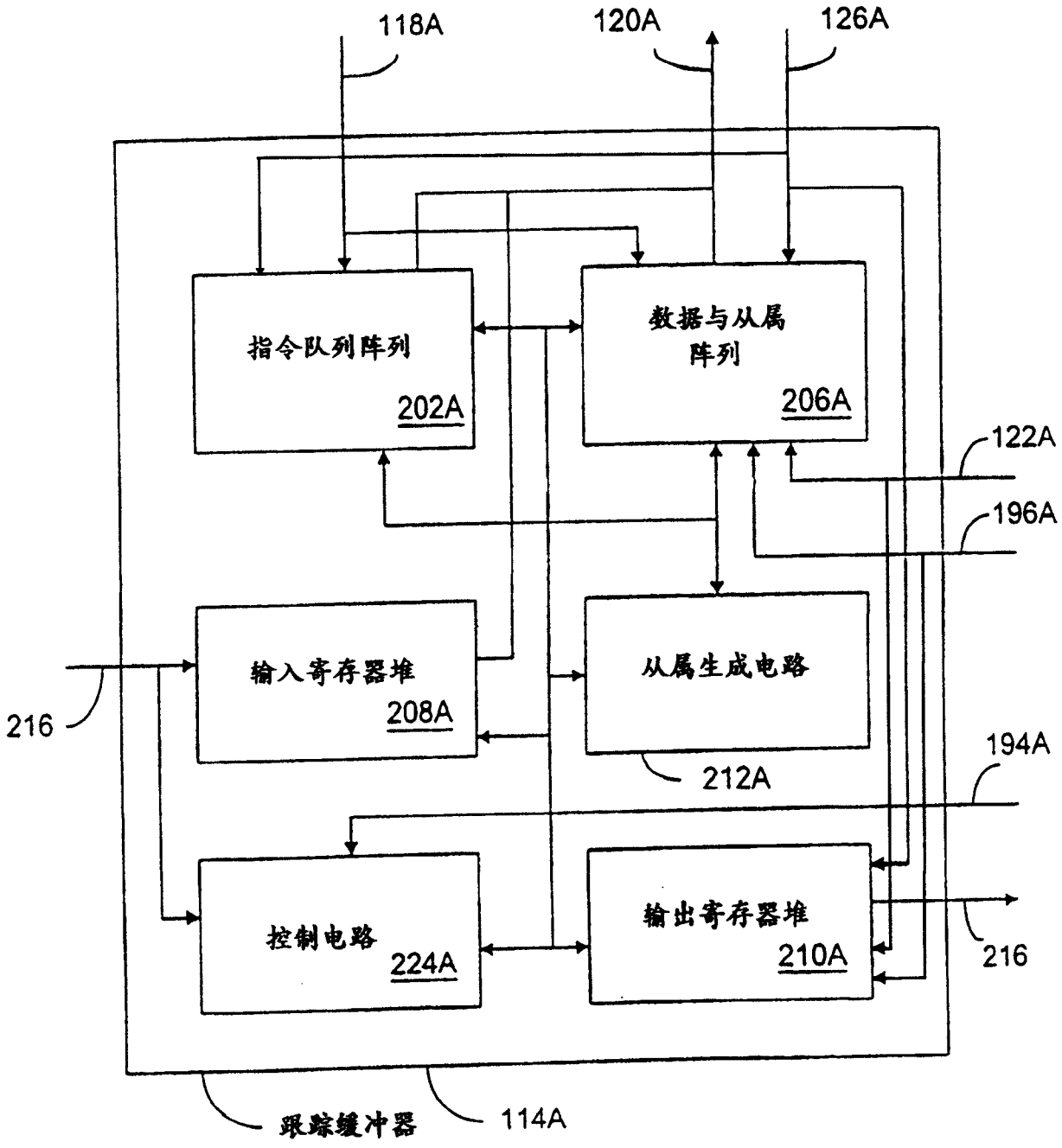

118A
120A
126A
指令队列阵列
202A
数据与从属
阵列
206A
122A
196A
输入寄存器堆
208A
从属生成电路
212A
216
194A
控制电路
224A
输出寄存器堆
210A
216
跟踪缓冲器
114A

图 10

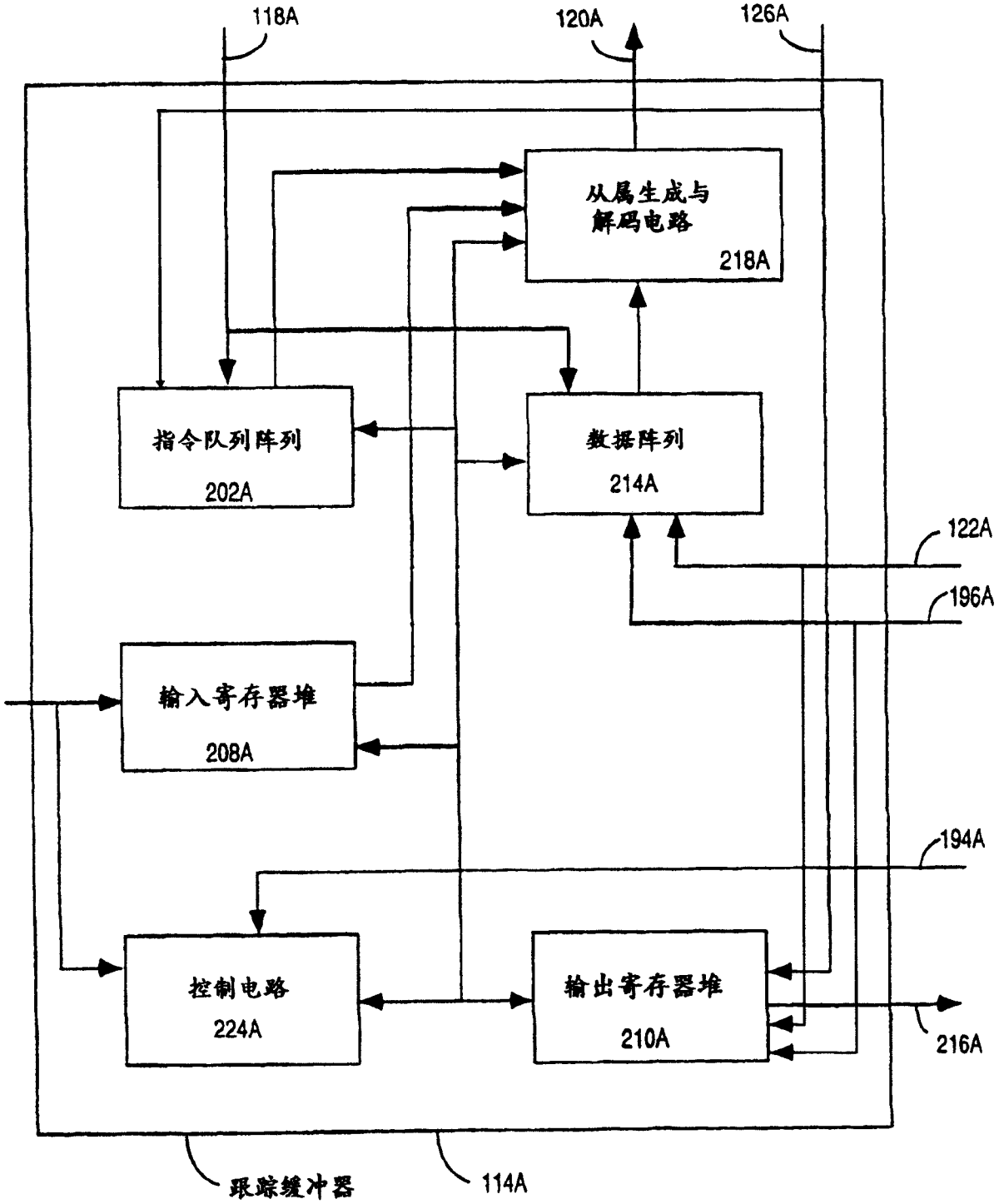

118A
120A
126A
从属生成与
解码电路
218A
指令队列阵列
202A
数据阵列
214A
122A
196A
输入寄存器堆
208A
194A
控制电路
224A
输出寄存器堆
210A
216A
跟踪缓冲器
114A

图 11

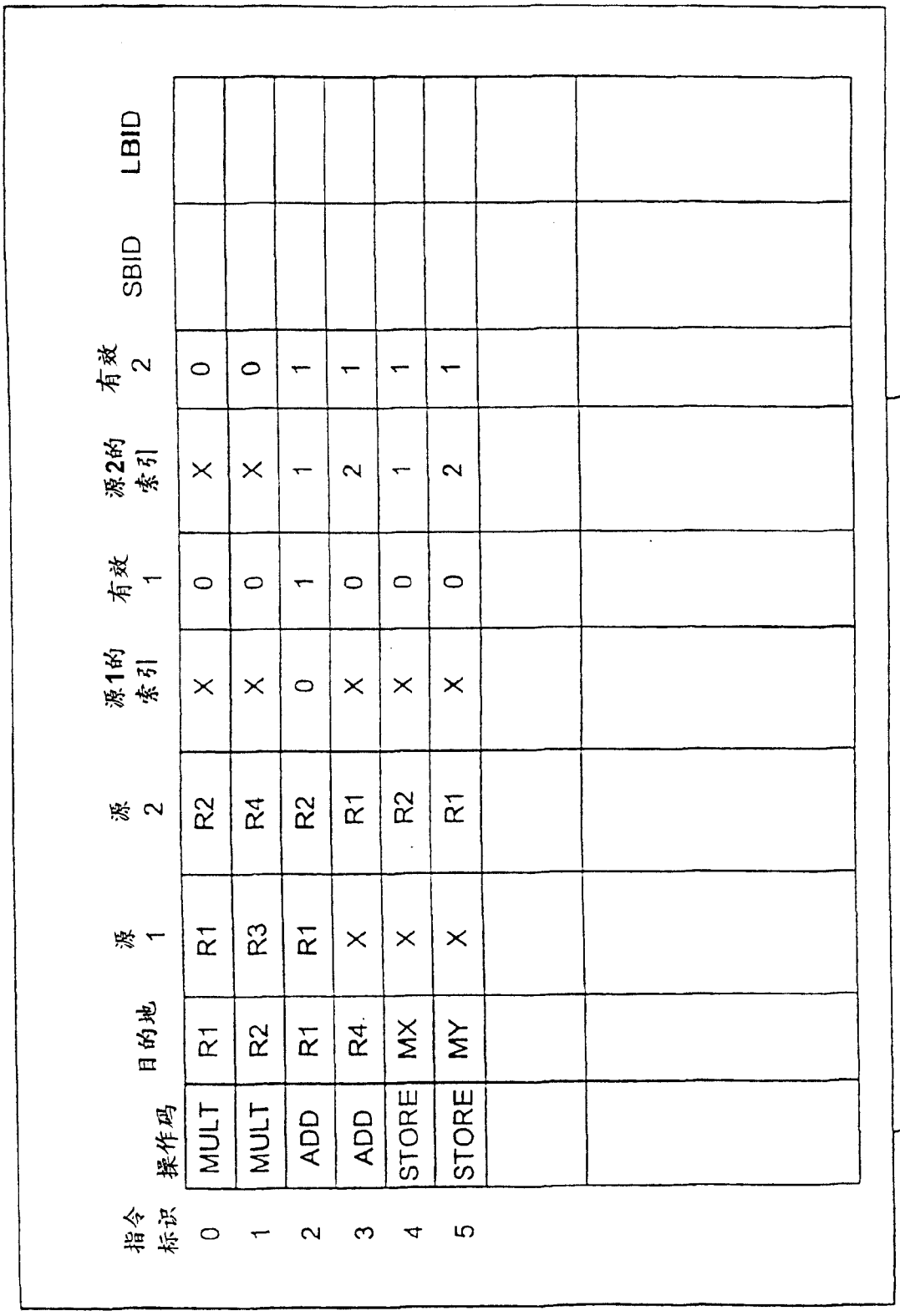

指令标识
操作码
目的地
源1
源2
源1的索引
有效1
源2的索引
有效2
SBID
LBID
0 MULT R1 R1 R2 X 0 X 0
1 MULT R2 R3 R4 X 0 X 0
2 ADD R1 R1 R2 0 1 1 1
3 ADD R4 X R1 X 0 2 1
4 STORE MX X R2 X 0 1 1
5 STORE MY X R1 X 0 2 1
指令队列阵列
202A

图 12

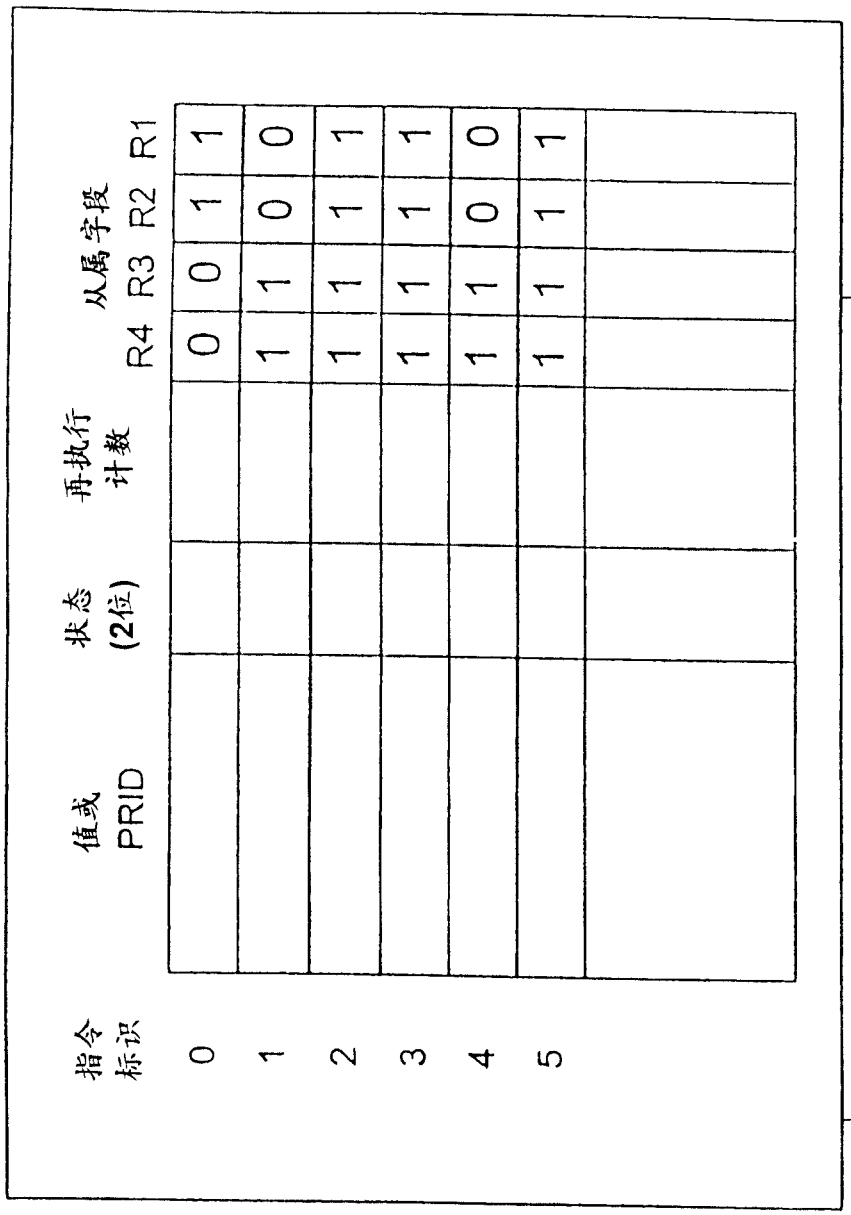

指令标识
值或PRID
状态(2位)
再执行计数
从属字段
R4 R3 R2 R1
0 0 0 1 1
1 1 1 0 0
2 1 1 1 1
3 1 1 1 1
4 1 1 0 0
5 1 1 1 1
数据与从属阵列
206A

图 13

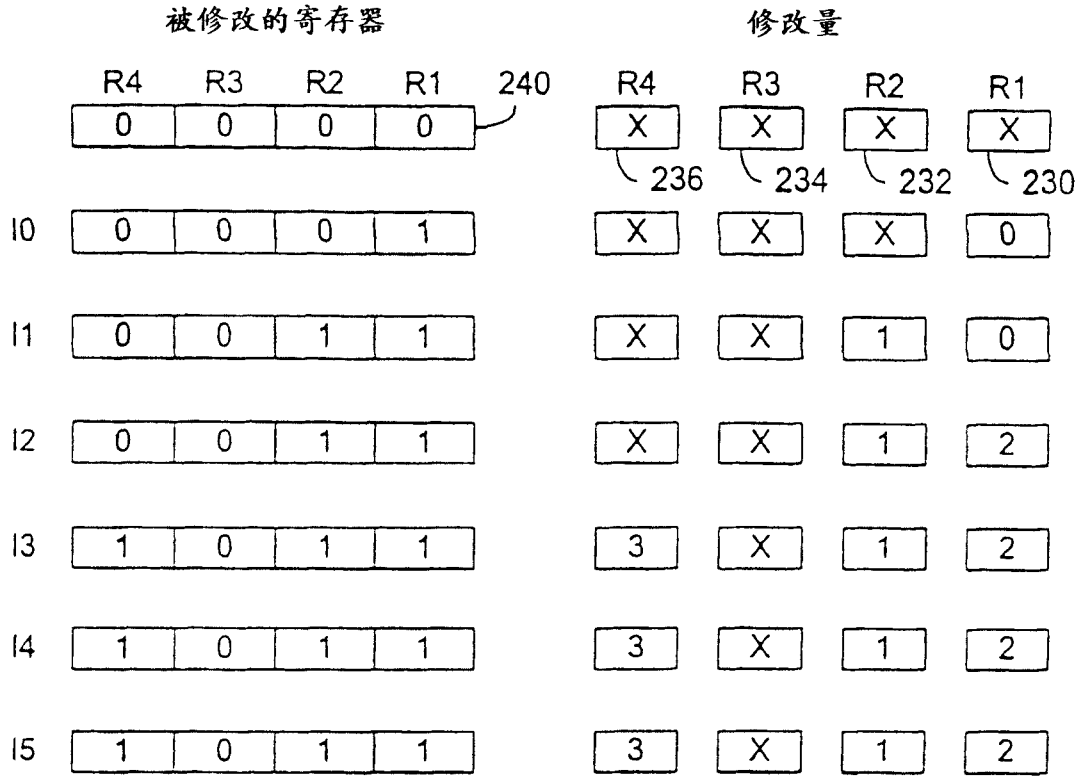

被修改的寄存器
修改量
R4 R3 R2 R1 240
R4 R3 R2 R1
0 0 0 0
X X X X
236 234 232 230
I0 0 0 0 1 X X X 0
I1 0 0 1 1 X X 1 0
I2 0 0 1 1 X X 1 2
I3 1 0 1 1 3 X 1 2
I4 1 0 1 1 3 X 1 2
I5 1 0 1 1 3 X 1 2

图 14

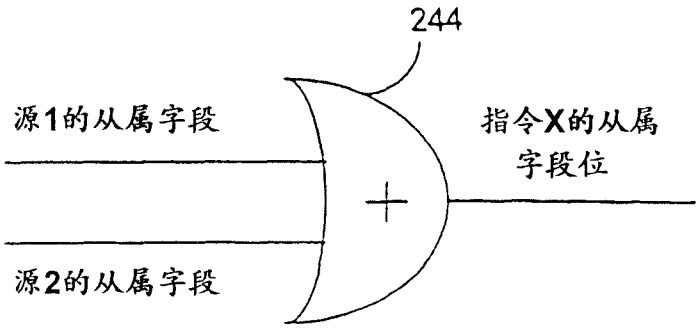

244
源1的从属字段
指令X的从属
字段位
源2的从属字段

图 15

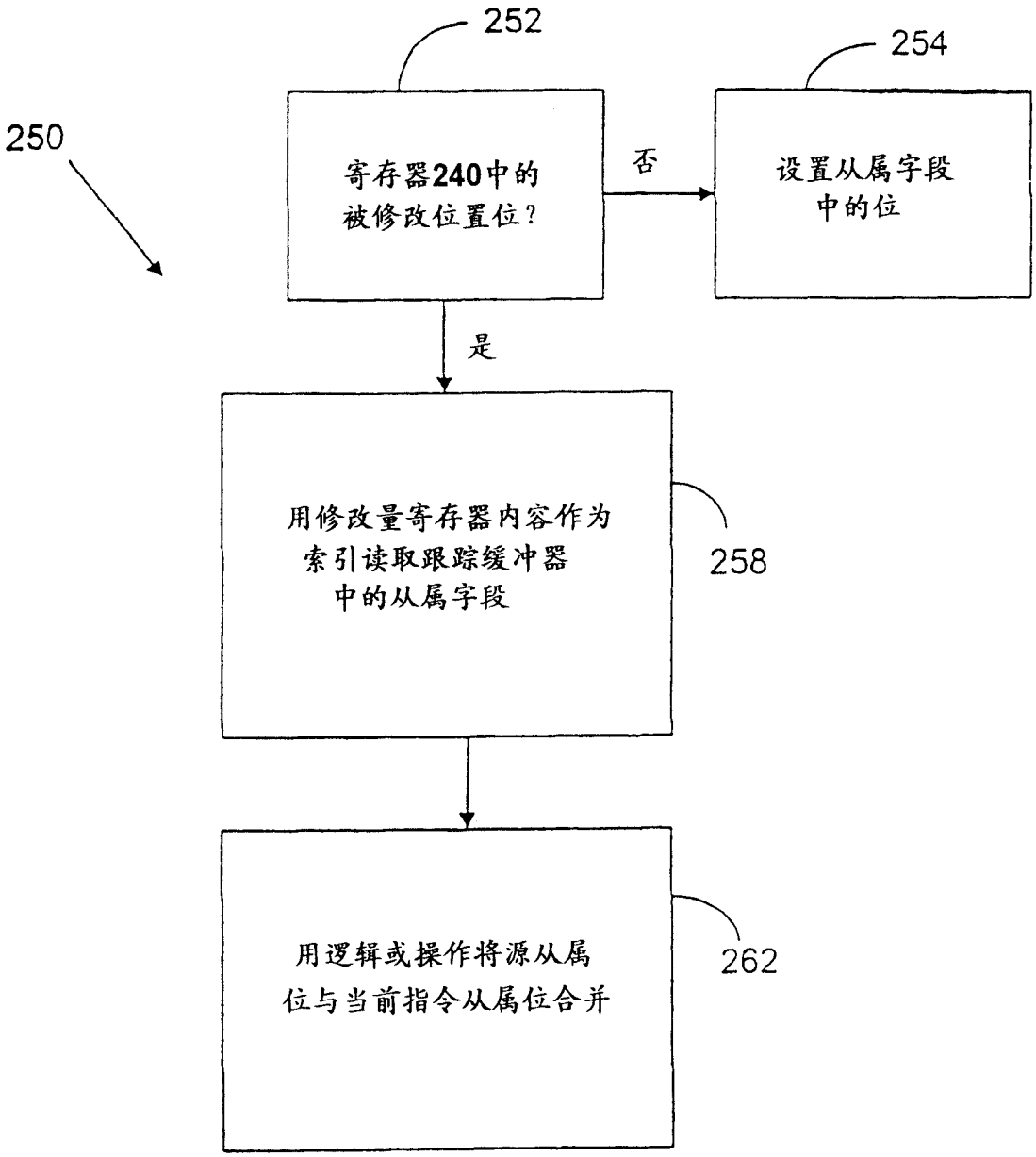

250
252
寄存器240中的
被修改位置位？
否
254
设置从属字段
中的位
是
用修改量寄存器内容作为
索引读取跟踪缓冲器
中的从属字段
258
用逻辑或操作将源从属
位与当前指令从属位合并
262

图 16

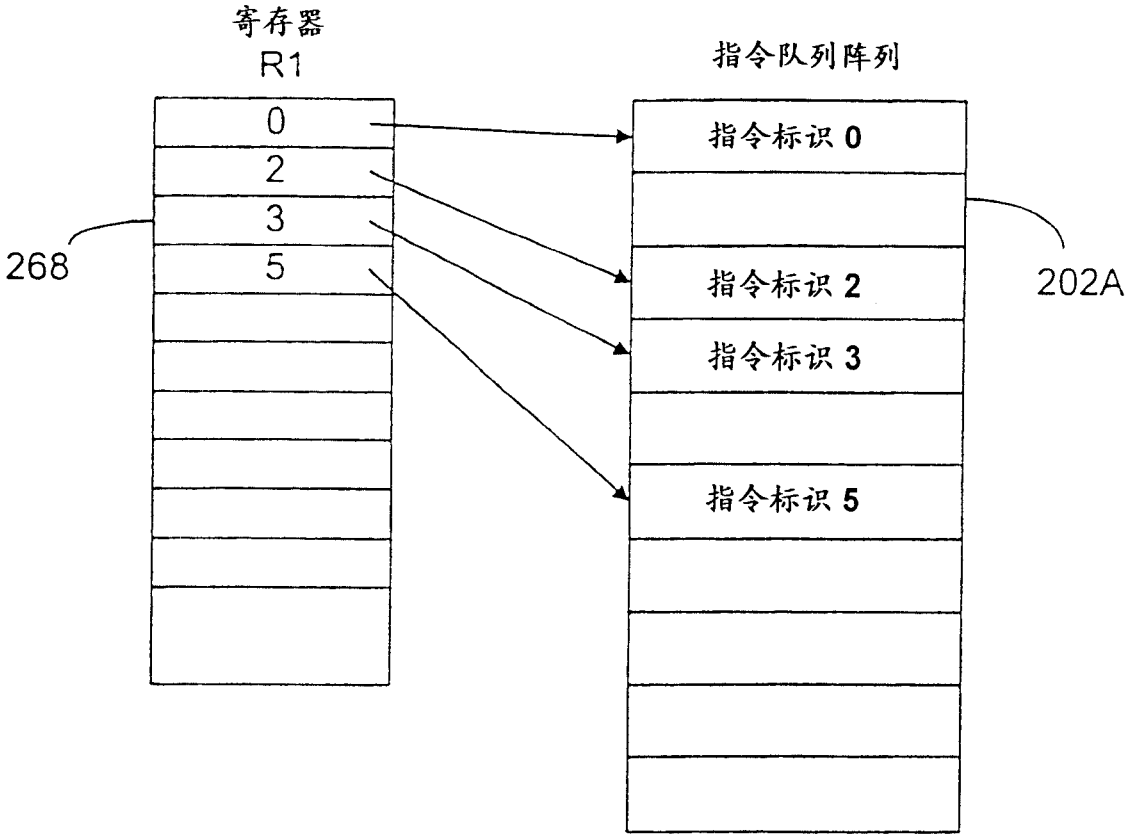


图 17

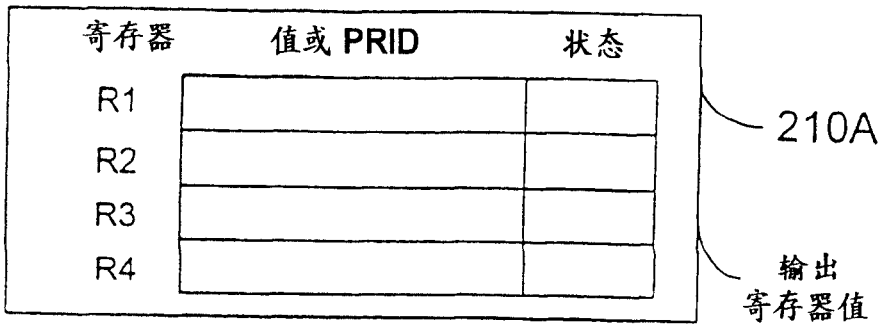

寄存器
值或 PRID
状态
R1
R2
R3
R4
210A
输出
寄存器值

图 18

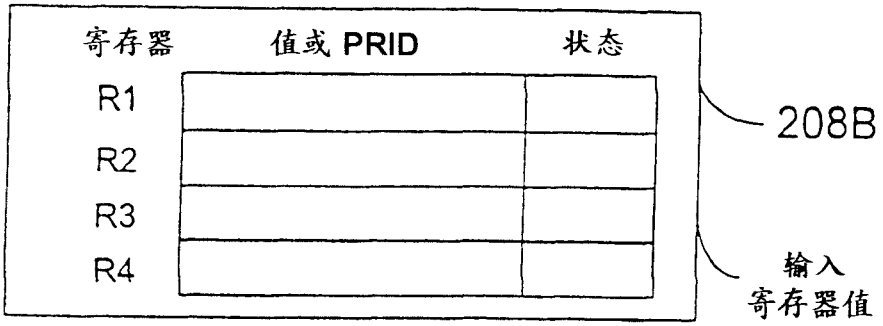

寄存器
值或 PRID
状态
R1
R2
R3
R4
208B
输入
寄存器值

图 19

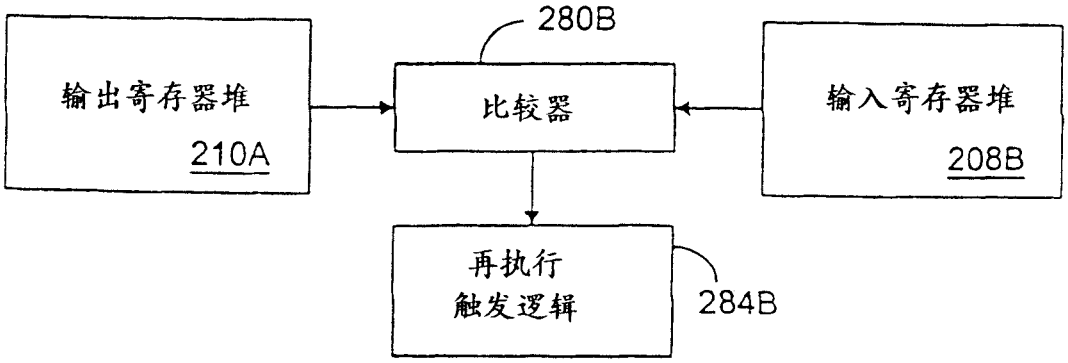

280B
输出寄存器堆
210A
比较器
输入寄存器堆
208B
再执行
触发逻辑
284B

图 20

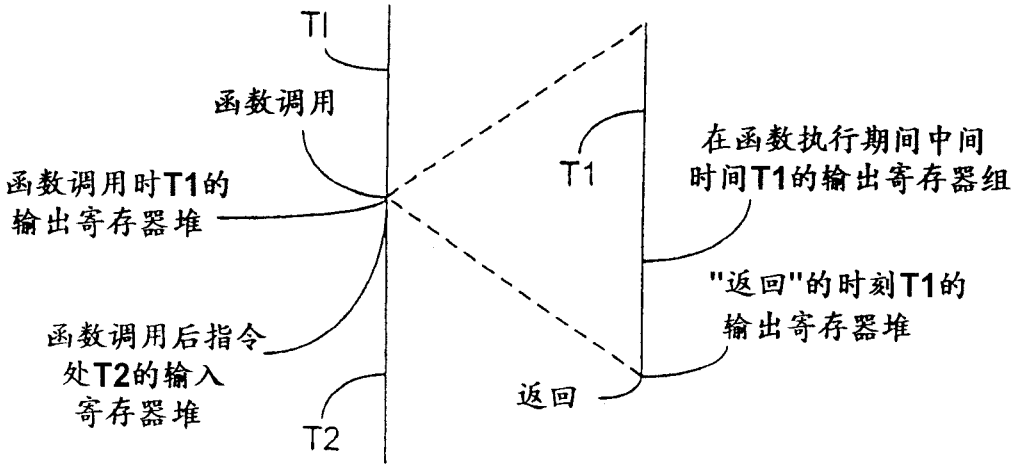

TI
函数调用
函数调用时T1的
输出寄存器堆
函数调用后指令
处T2的输入
寄存器堆
T2
T1
在函数执行期间中间
时间T1的输出寄存器组
"返回"的时刻T1的
输出寄存器堆
返回

图 21

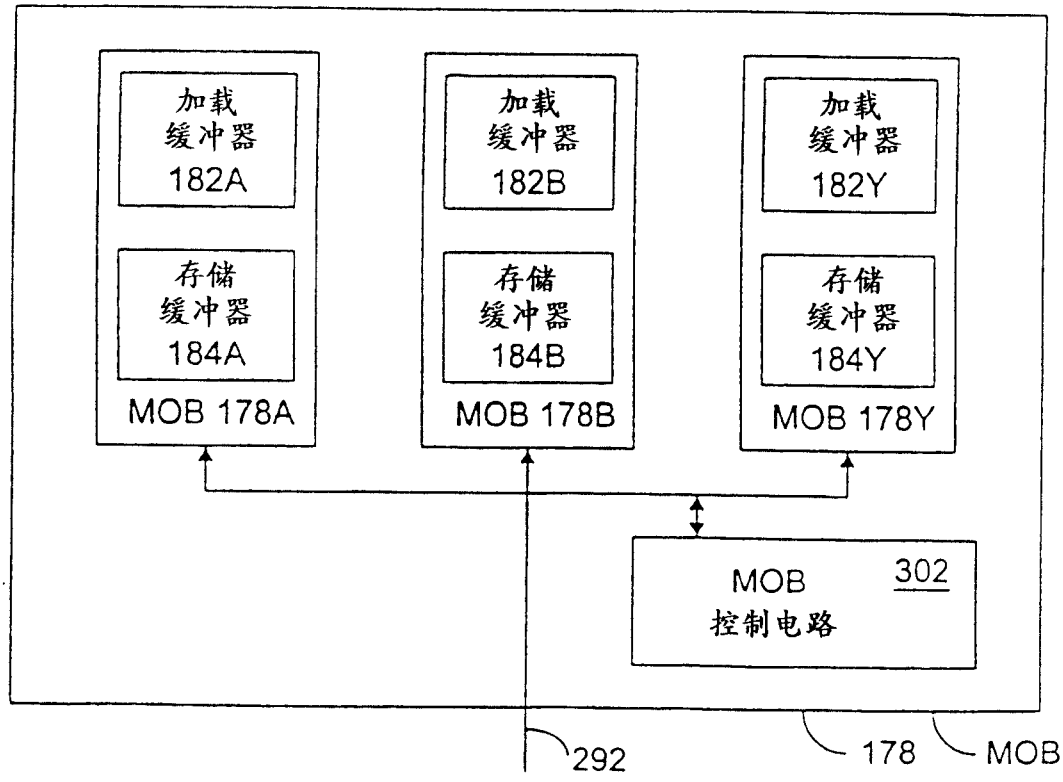

加载
缓冲器
182A
存储
缓冲器
184A
MOB 178A
加载
缓冲器
182B
存储
缓冲器
184B
MOB 178B
加载
缓冲器
182Y
存储
缓冲器
184Y
MOB 178Y
MOB
控制电路
302
292
178
MOB

图 22

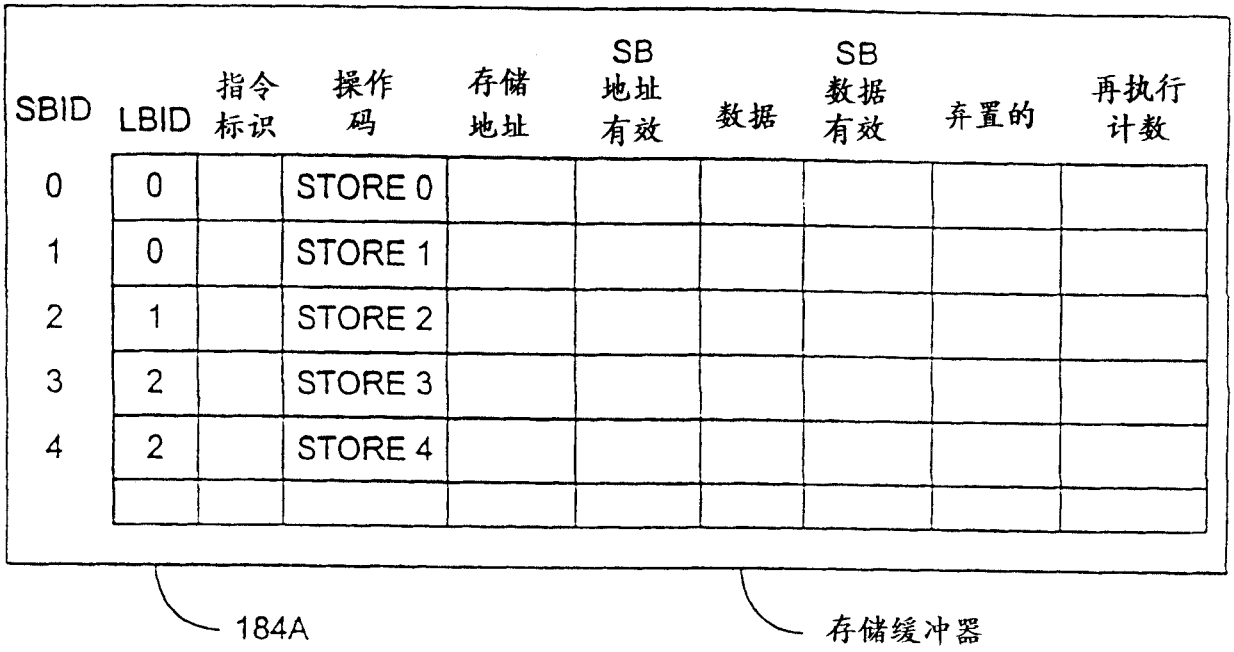

SBID
LBID
指令标识
操作码
存储地址
SB地址有效
数据
SB数据有效
弃置的
再执行计数
0 0 STORE 0
1 0 STORE 1
2 1 STORE 2
3 2 STORE 3
4 2 STORE 4
184A
存储缓冲器

图 23

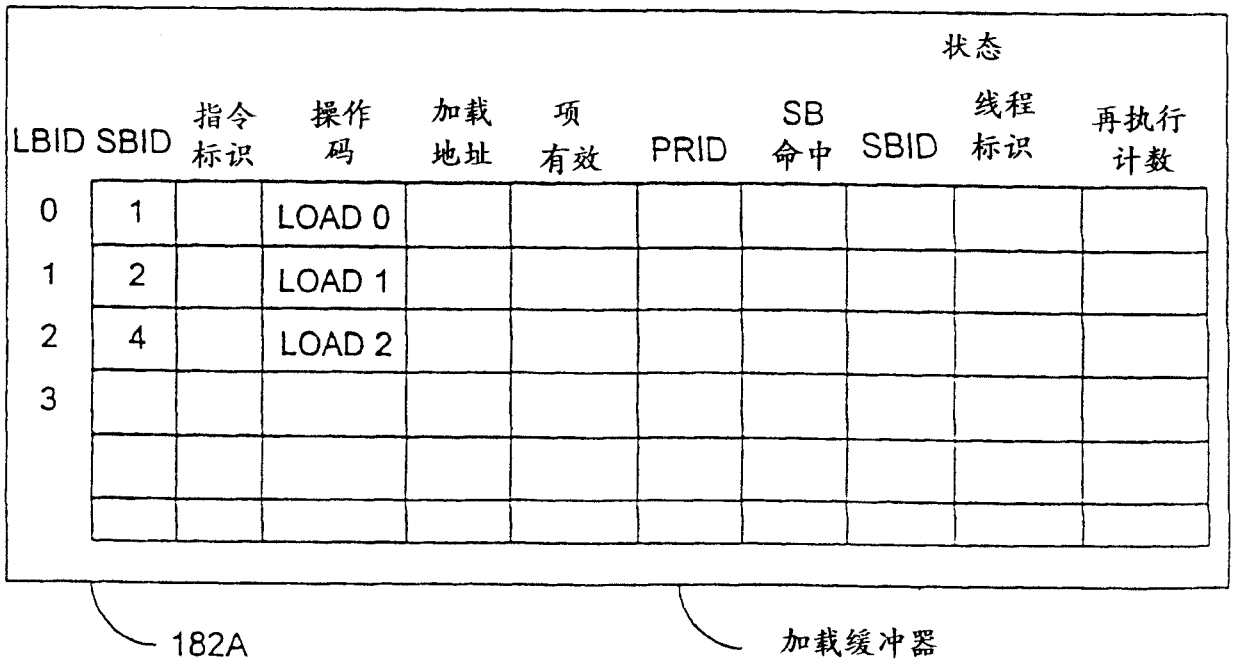

LBID
SBID
指令标识
操作码
加载地址
项有效
PRID
SB命中
状态
SBID
线程标识
再执行计数
0 1 LOAD 0
1 2 LOAD 1
2 4 LOAD 2
3
182A
加载缓冲器

图 24

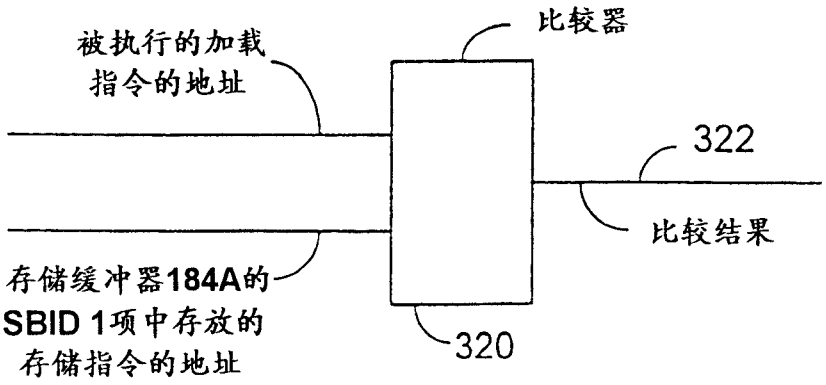

比较器
被执行的加载
指令的地址
322
比较结果
存储缓冲器184A的
SBID 1项中存放的
存储指令的地址
320

图 25

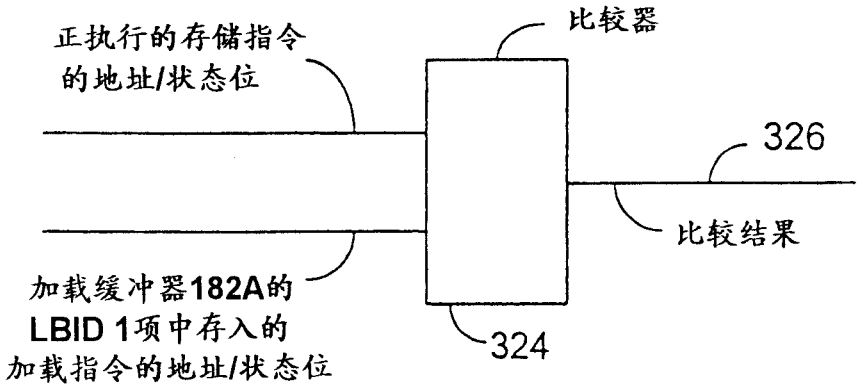

比较器
正执行的存储指令
的地址/状态位
326
比较结果
加载缓冲器182A的
LBID 1项中存入的
加载指令的地址/状态位
324

图 26

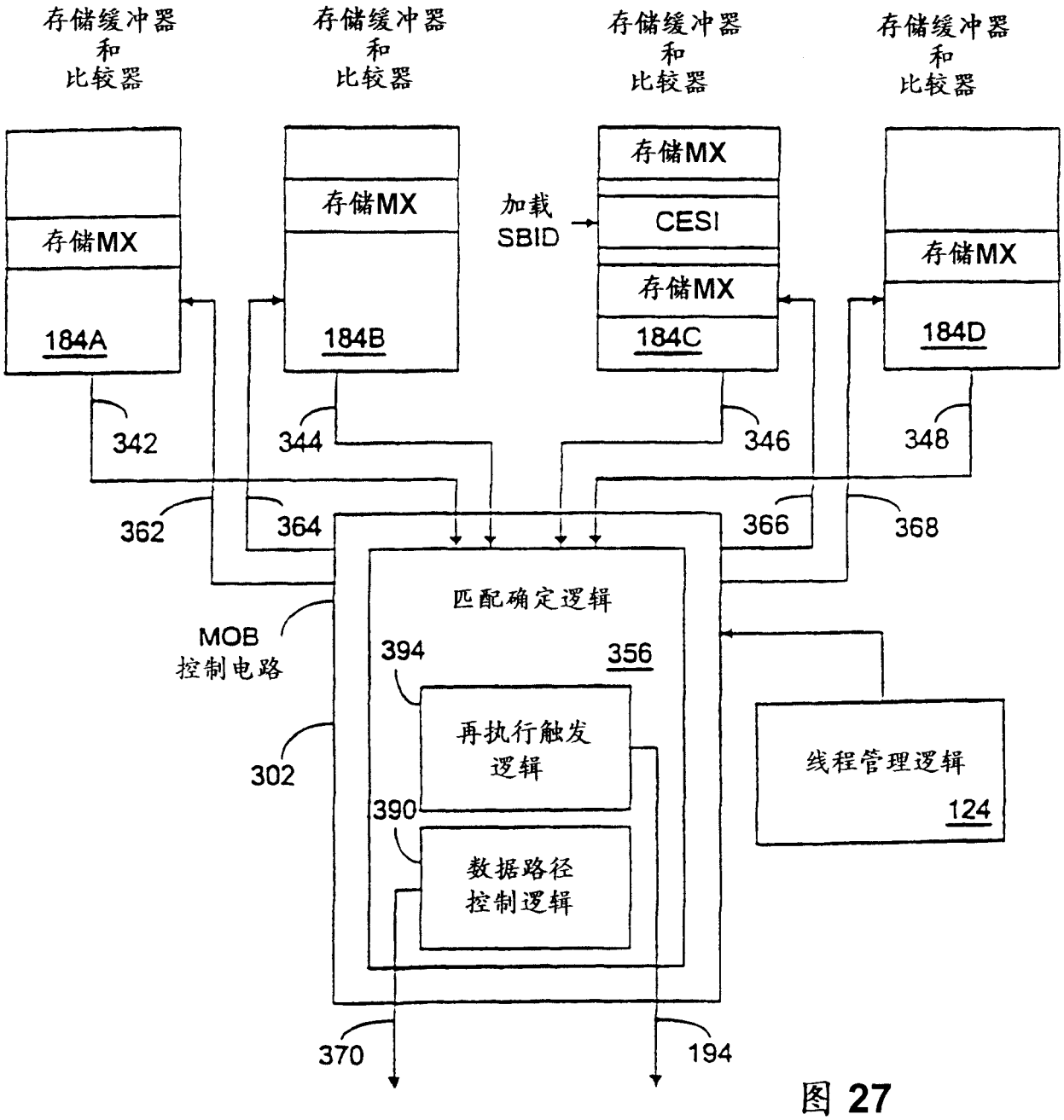

存储缓冲器
和
比较器
存储缓冲器
和
比较器
存储缓冲器
和
比较器
存储缓冲器
和
比较器
存储MX
184A
存储MX
184B
加载
SBID
存储MX
CESI
存储MX
184C
存储MX
184D
342
344
346
348
362
364
366
368
匹配确定逻辑
MOB
控制电路
394
356
再执行触发
逻辑
302
390
数据路径
控制逻辑
线程管理逻辑
124
370
194

图 27

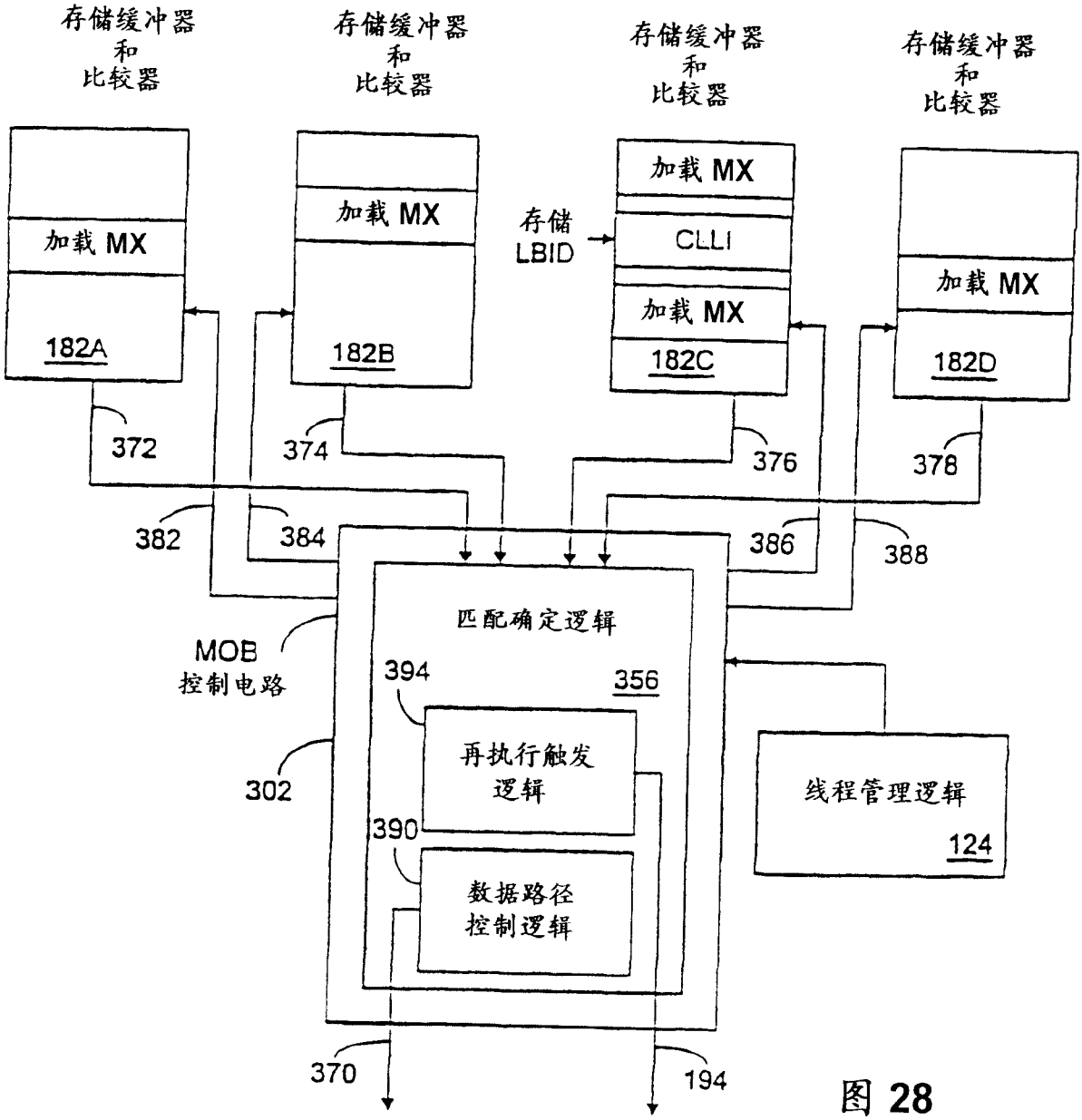

存储缓冲器
和
比较器
存储缓冲器
和
比较器
存储缓冲器
和
比较器
存储缓冲器
和
比较器
加载 MX
182A
加载 MX
182B
存储
LBID
加载 MX
CLLI
加载 MX
182C
加载 MX
182D
372
374
376
378
382
384
386
388
匹配确定逻辑
MOB
控制电路
394
356
再执行触发
逻辑
302
390
数据路径
控制逻辑
线程管理逻辑
124
370
194

图 28

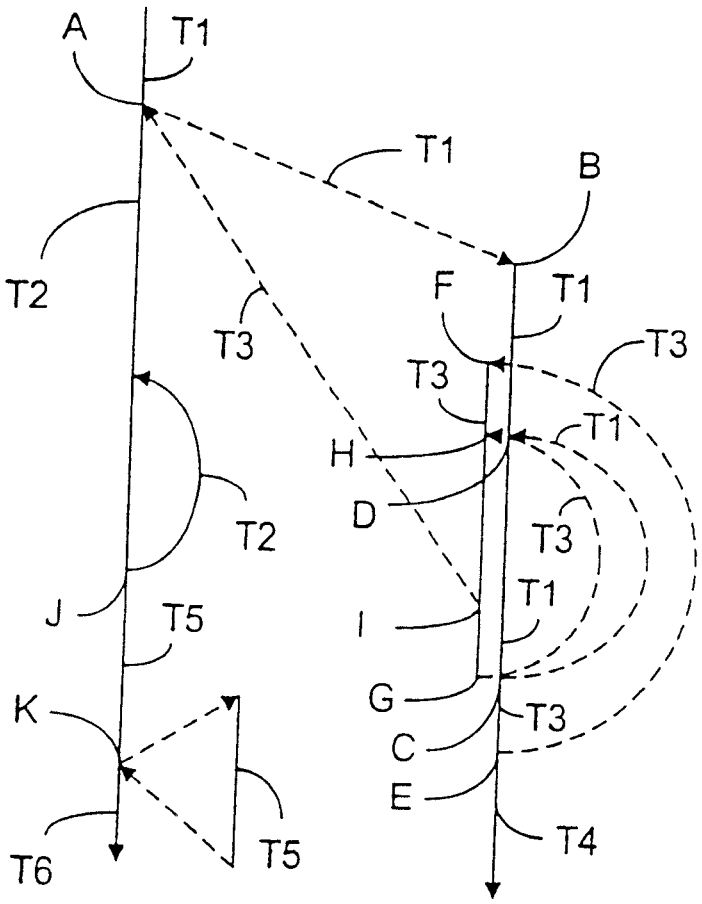


图 29

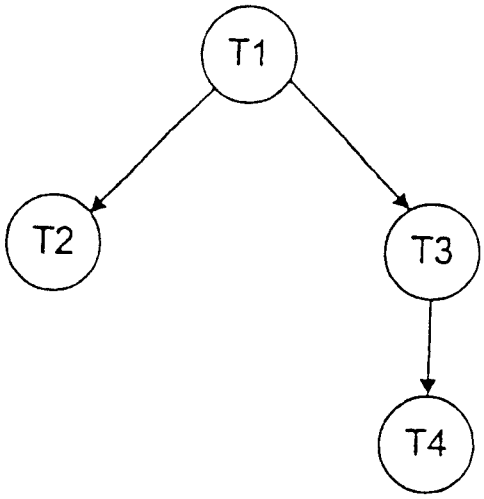
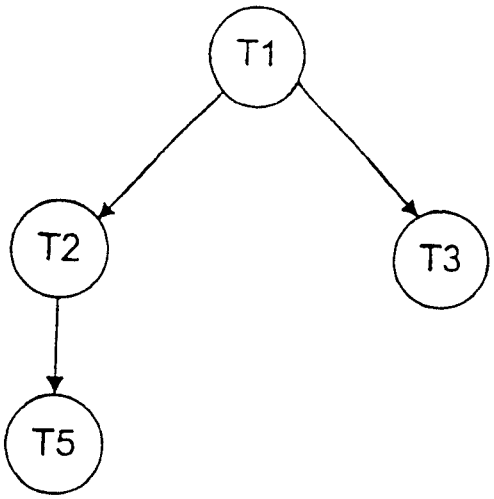


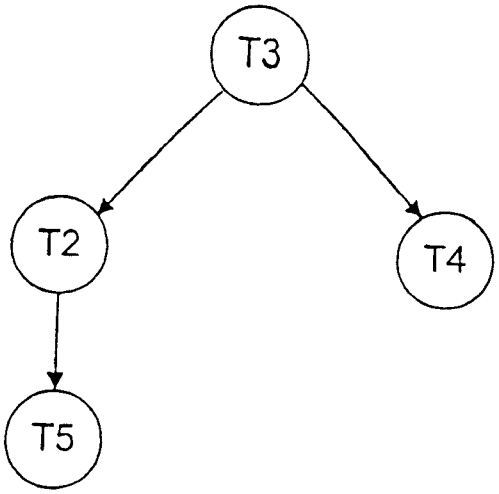
图 30

时刻t1时的树



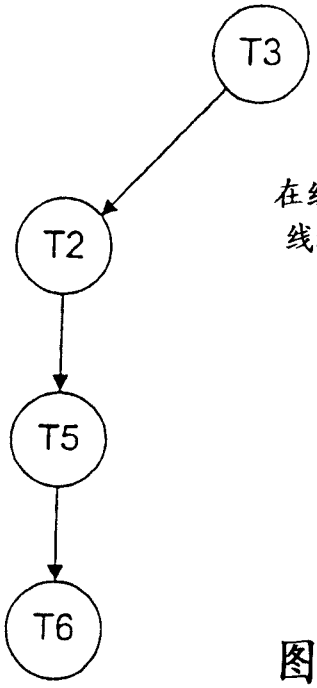
假定线程T4在线程T1弃置之前复原在时刻t2时的树

图 31



假定线程T1在线程T4复原之前弃置在时刻t2时的树

图 32



在线程T1已经弃置并且线程T4被复原之后的时刻t3的树

图 33

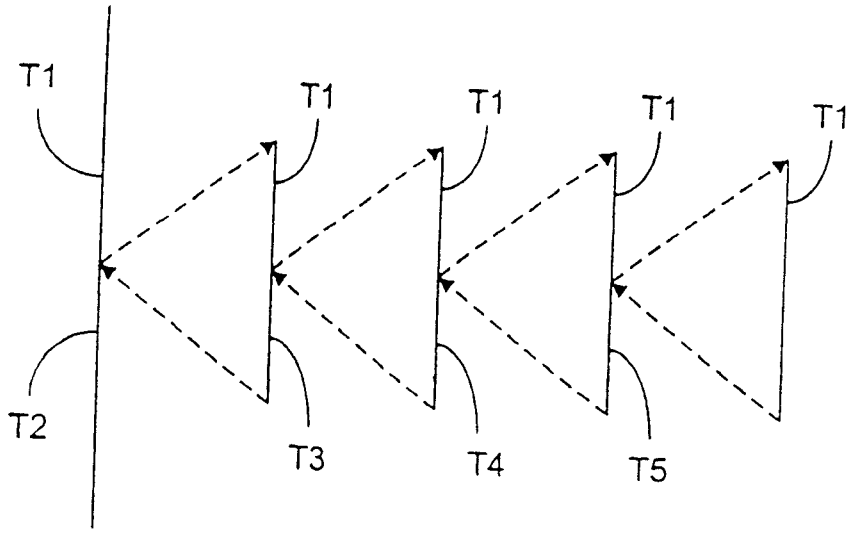
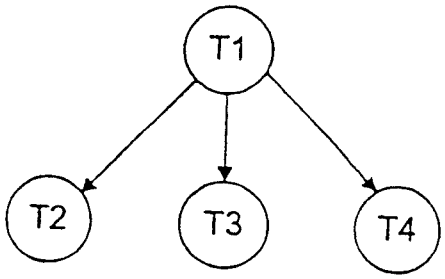
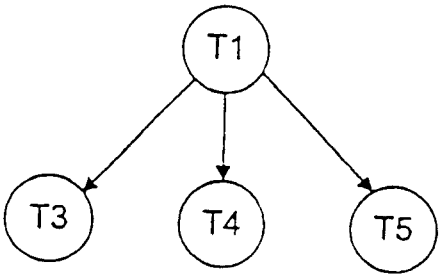


图 34



时刻t1时的树

图 35



时刻t2时的树

图 36

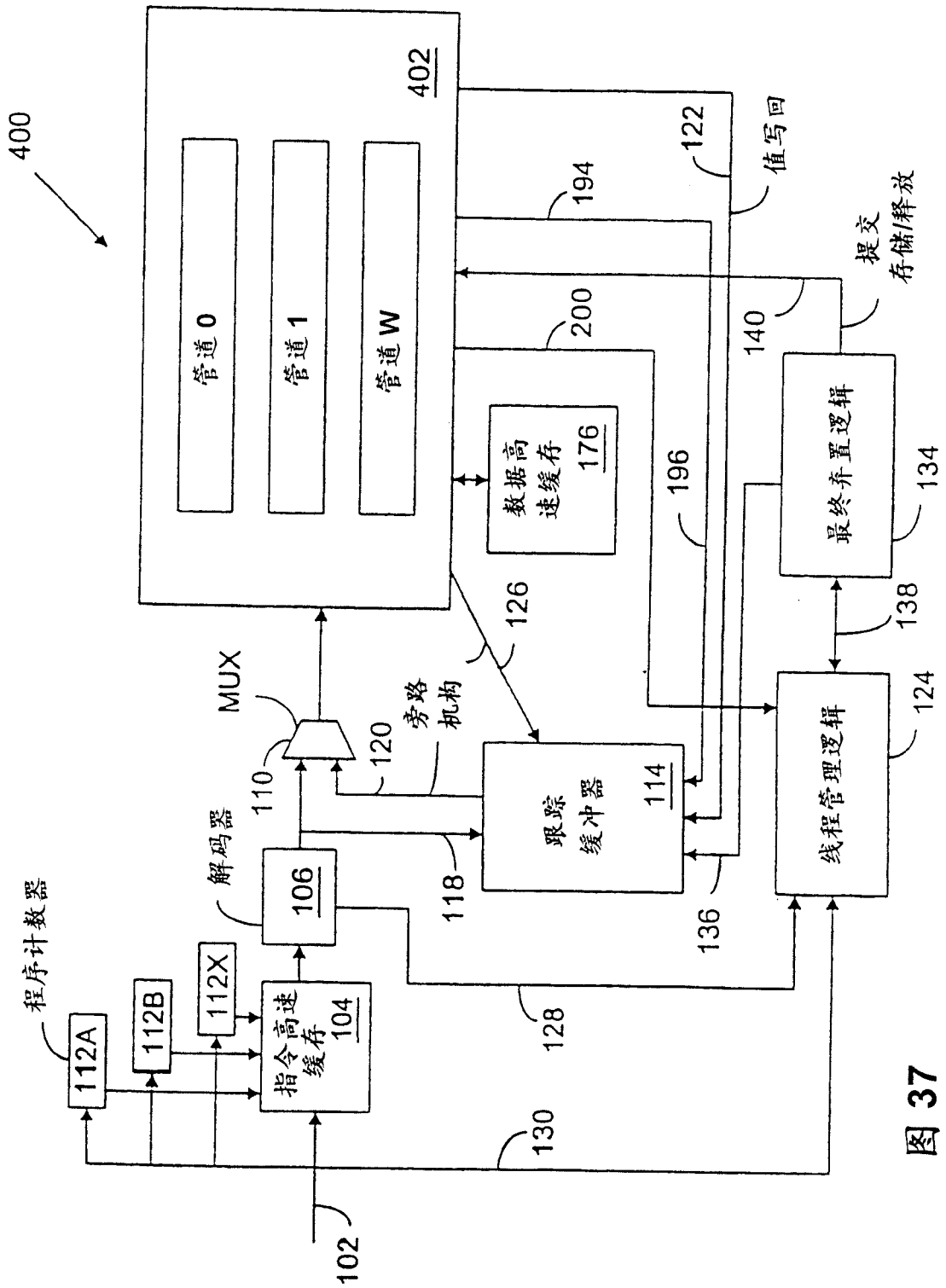

400
402
管道 0
管道 1
管道 W
194
122
值写回
200
140
提交
存储/释放
数据高
速缓存
176
196
最终弃置逻辑
134
126
MUX
110
120
旁路
机构
跟踪
缓冲器
114
118
106
解码器
112X
112B
112A
程序计数器
指令高速
缓存
104
136
138
124
线程管理逻辑
128
130
102

图 37

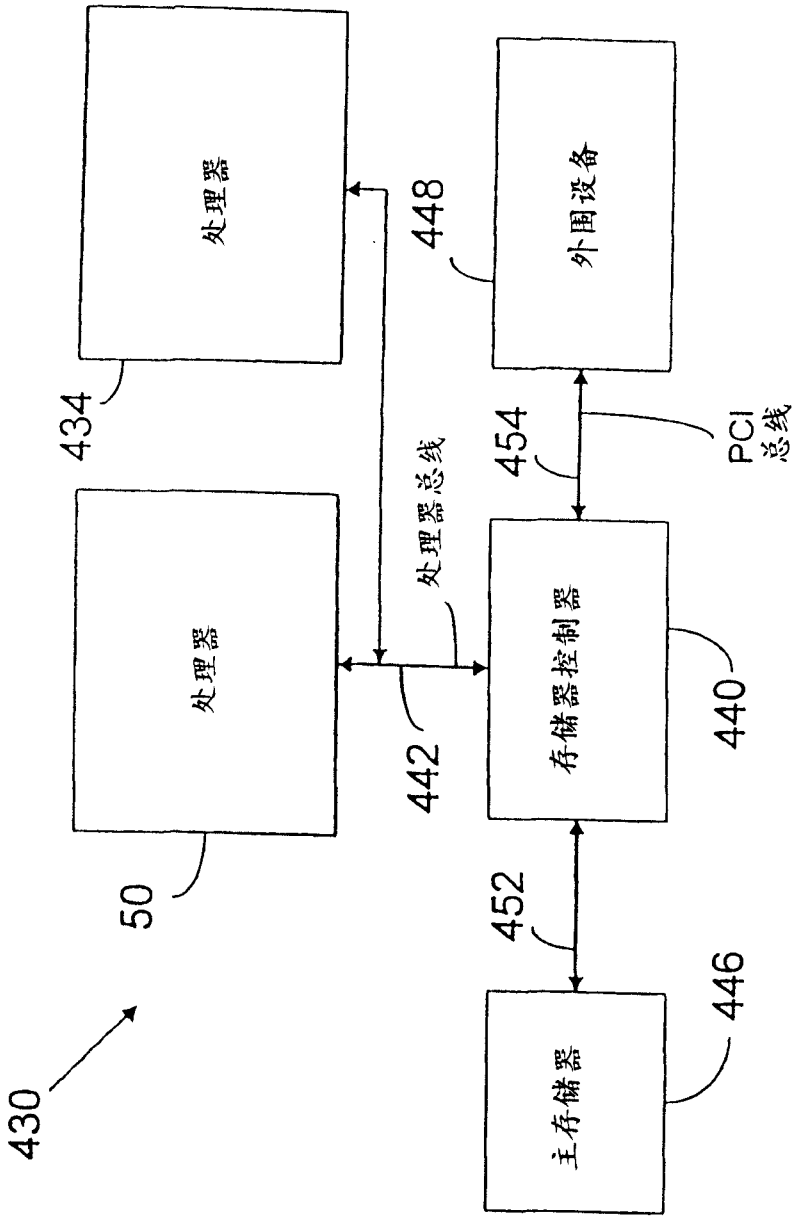

处理器
处理器
处理器总线
存储器控制器
外围设备
PCI
总线
主存储器
430
434
50
442
440
448
454
452
446

图 38