[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/38

[12] 发明专利说明书

[21] ZL 专利号 98122422.9

[43] 授权公告日 2003 年 5 月 28 日

[11] 授权公告号 CN 1109968C

[22] 申请日 1998.11.18 [21] 申请号 98122422.9

[30] 优先权

[32] 1998. 1.20 [33] US [31] 008792

[71] 专利权人 国际商业机器公司

地址 美国纽约

[72] 发明人 小托马斯·J·赫勒

威廉·托德·保蒂

审查员 杨 蕊

[74] 专利代理机构 中国国际贸易促进委员会专利商标事务所

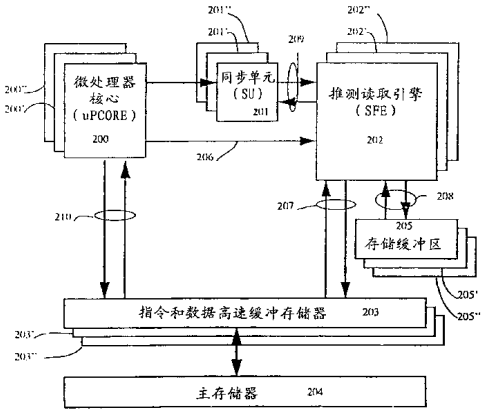
代理人 吴丽丽

权利要求书 5 页 说明书 19 页 附图 8 页

[54] 发明名称 带同步流水线的微处理器的改进方法

[57] 摘要

一种通过具有多流水线同步的寄存器管理系统来改善微处理器计算机系统乱序支持的系统和方法，用于在具有第一处理部件和第二处理部件的计算机系统中处理顺序指令流，每一个所述处理部件具有由它自己的通用寄存器和控制寄存器所决定的状态。 在所述第一处理部件处理所述顺序指令流的任何时刻，存在所述第二处理部件来开始继续对同一指令流进行处理是有好处的，然后第一处理部件和第二处理部件对连续的指令流进行处理而且有可能执行同一条指令。



ISSN1008-4274

1．一种在具有第一和第二处理部件的计算机系统中处理顺序指令流的方法，每一个所述处理部件具有由自己的通用寄存器和控制寄存器决定的状态，其特征在于，包括以下各步：

将一条所述顺序指令流初始化指令分配给所述处理部件中的所述第一个，

使用所述处理部件中的所述第一个来继续对所述顺序指令流进行处理，并将计算机系统的结构状态的任何改变发送到所述第二处理部件，并且

当在所述第一处理部件处理所述顺序指令流的任何时候，由第二处理部件接着继续处理所述同一顺序指令流对计算机系统来说能够带来效益时，

所述第二处理部件恢复被发送的状态，并接着继续通过用所述第二处理部件处理所述顺序指令流对同一顺序指令流进行处理，

所述第二处理部件然后在所述第二处理部件处理所述顺序指令流期间将计算机系统的结构状态的任何改变发送到所述第一处理部件；

由此，所述第一和第二处理部件在所述顺序指令流的处理期间可以执行同一条指令，但是只允许所述处理部件中的一个来改变由所述第一处理部件和第二处理部件联合状态决定的所述计算机系统的整个结构状态。

2．根据权利要求1的方法，其特征在于，所述第一处理部件包括作为多处理器运行的多个第一处理部件。

3．根据权利要求1的方法，其特征在于，所述联合状态由所述第一处理部件决定。

4．根据权利要求1的方法，其特征在于，所述第一处理部件和第二处理部件的联合状态由所述第一处理部件和一组一个或多个第一处理部件以及至少一个第二处理部件作为多处理器来决定。

5. 根据权利要求 4 的方法，其特征在于，为每一个所述第二处理部件提供一个同步单元。

6. 根据权利要求 4 的方法，其特征在于，为每一个所述第二处理部件提供一个同步单元并且该同步单元决定何时所述第二处理部件开始处理正在被第一处理部件处理的同一条指令。

7. 根据权利要求 4 的方法，其特征在于，为每一个所述第二处理部件提供一个同步单元并且该同步单元决定所述第二处理部件何时开始处理正在被所述第一处理部件处理的指令流的下一条指令或是同一条指令。

8. 根据权利要求 4 的方法，其特征在于，为每一个所述第二处理部件提供一个同步单元并且该同步单元决定所述第二处理部件何时开始处理正在被所述第一处理部件处理的指令流的下一条指令或是同一条指令，并决定所述第二处理部件对指令的处理应该何时被停止或是忽略。

9. 根据权利要求 8 的方法，其特征在于，所述第二处理部件对指令的处理应该何时被停止或是忽略的决定基于从第一处理部件和第二处理部件提供给同步单元的输入而计算得到的对整个计算机系统的效益判断来进行。

10. 根据权利要求 9 的方法，其特征在于，所述提供给同步单元的输入包括当前判断的信息或是存储在系统中的信息。

11. 根据权利要求 10 的方法，其特征在于，所述输入存储在同步单元中的指令计数器中。

12. 根据权利要求 6 的方法，其特征在于，当所述第一处理部件对指令的处理出现停滞时，所述同步单元决定所述第二处理部件何时开始处理由第一处理部件正在处理的同一条指令。

13. 根据权利要求 6 的方法，其特征在于，当在指令处理部件进行处理的过程中出现在设计第二处理部件时没有处理的操作时，所述同步单元决定何时将第二处理部件的状态和结构状态进行重新同步。

14. 根据权利要求 6 的方法，其特征在于，当判断出在处理所述

指令流中第二处理部件没有为计算机系统提供任何效益时，同步单元决定何时将第二处理部件的状态与结构状态重新同步。

15. 根据权利要求 6 的方法，其特征在于，当所述第一处理部件对指令的处理过程出现停滞时，所述同步单元决定何时与哪一个第一处理部件一起第二处理部件开始处理正在被第一处理部件处理的同一条指令。

16. 根据权利要求 6 的方法，其特征在于，当在指令处理部件进行处理的过程中出现在设计第二处理部件时没有处理的操作时，所述同步单元决定何时与哪一个第一处理部件一起重新将第二处理部件状态与结构状态同步。

17. 根据权利要求 6 的方法，其特征在于，当判断出在处理所述指令流中第二处理部件没有为计算机系统提供任何效益时，同步单元决定何时与哪一个第一处理部件一起重新将第二处理部件状态与结构状态同步。

18. 根据权利要求 6 的方法，其特征在于，所述第二处理部件的结果存储到它的私有通用寄存器或存储缓冲区。

19. 根据权利要求 1 的方法，其特征在于，为所述第二处理部件提供一个同步单元，该同步单元决定所述第二处理部件何时开始对正在被所述第一处理部件处理的处理流中的下一条指令或是同一条指令进行处理，并决定所述第二处理部件对指令的处理何时应该被停止或是忽略。

20. 根据权利要求 1 的方法，其特征在于，所述第二处理部件在处理所述指令流时将结果存入它的私有通用寄存器或与所述第二处理部件连接的私有存储缓冲区，并且所述第一处理部件从由所述第一处理部件和第二处理部件共享的数据和指令高速缓冲存储器中读取数据以进行读取。

21. 根据权利要求 20 的方法，其特征在于，所述第二处理部件被用来处理由所述第一处理部件处理的指令流中的某些相同的指令。

22. 根据权利要求 21 的方法，其特征在于，所述第二处理部件是

一种乱序处理器。

23. 根据权利要求 15 的方法，其特征在于，在停滞之后，所述第一处理部件和第二处理部件被重新同步，并且在重新同步期间第二处理部件在重新同步之前为第一处理部件将指令流预处理的所有的结果和局部结果清除。

24. 根据权利要求 16 的方法，其特征在于，在重新同步期间，第二处理部件在重新同步之前为第一处理部件将指令流预处理的所有的结果和局部结果清除。

25. 根据权利要求 17 的方法，其特征在于，在重新同步期间，第二处理部件在重新同步之前为第一处理部件将指令流预处理的所有的结果和局部结果清除。

26. 一种用于在具有第一处理部件和第二处理部件并且每一个所述处理部件具有它自己的由其通用寄存器和控制寄存器决定的状态的计算机系统中处理顺序指令流的方法，其特征在于，包括以下各步：

将一条所述顺序指令流初始化指令分配给所述处理部件中的所述第一个，

使用所述处理部件中的所述第一个来继续对所述顺序指令流进行处理，并将第二处理部件所要求的计算机系统的结构状态的任何改变发送到所述第二处理部件，并且累积在将来的某时候为所述第二处理部件结构状态所使用的所述发送的改变，并且当在所述第一处理部件处理所述顺序指令流的任何时候，由第二处理部件接着继续处理所述同一顺序指令流对计算机系统来说能够带来效益时，

所述第二处理部件恢复以前从所述第一处理部件发送的所述累积的结构状态，并接着继续通过用所述第二处理部件处理所述顺序指令流对同一顺序指令流进行处理，

然后，所述第二处理元件在处理所述顺序指令期间将第一处理部件所要求的计算机系统的结构状态的任何改变发送到所述第一处理部件，以累积在将来的某时候为所述第一处理部件结构状态所使用的所述改变，

由此，所述第一和第二处理部件在所述顺序指令流的处理期间可以执行同一条指令，但是只允许所述处理部件中的一个来改变由所述第一处理部件和第二处理部件联合决定的所述计算机系统的整个结构状态。

27. 根据权利要求 26 的方法，其特征在于，在所述第二处理部件处理所述顺序指令流的任何时候，由所述第一处理部件接着继续处理同一指令流对计算机系统来说能够带来效益时，所述第一处理部件恢复由第二处理部件发送的所述累积的结构状态，并接着继续通过所述第一处理部件处理所述顺序指令流来对同一顺序指令流进行处理，所述第一处理部件包括起多处理器作用的多个第一处理部件。

带同步流水线的微处理器的改进方法

技术领域

本发明涉及计算机系统，特别是一种通过流水线同步来提高系统性能的具有协处理器的微处理器计算机系统的方法。

技术背景

由于大量的重要工作量受到有限的高速缓冲存储器影响，现在的微处理器的性能受到了严重的限制。有限的高速缓冲存储器影响包括所有引起性能降低，而在微处理器的一级高速缓冲存储器扩展到无限大时就可以消除的因素。在很多时候，微处理器停下来等待存储在外部存储器中的操作数所花的时间几乎等于执行指令所用的时间。在数据库操作和事务更新操作时尤其如此。

现在许多微处理器设计采取减少有限的高速缓冲存储器带来的性能恶化的方法。增大高速缓冲存储器容量、多级高速缓冲存储器、高速多芯片模块、失序执行以及指令预取等手段被广泛运用并被认为是最成功的。操作数预取也被成功地用于某些采用常规乱序处理和不采用常规乱序处理的工作当中。然而，操作数预取对于数据库操作和事务操作并不是特别有效。增加高速缓冲存储器可以减小有限高速缓冲存储器影响，但是这方面的进一步改进受到了不断增加的电路规模或芯片数目等带来的成本性能的限制。当前的乱序执行技术较大地减小了有限高速缓冲存储器影响，但是它们也降低了处理器的时钟频率而且增加了设计的复杂性。因而需要提供一种能充分减少实现从前认为可行的乱序执行设计的复杂性的改进的微处理器设计。

术语

CPI 指每条指令的机器周期数。

SFE 是本发明提供的推测读取引擎。

uPCore 表示一种在周期时间、设计复杂性和无限一级高速缓冲存

储器的 CPI 之间进行折衷平衡的微处理器设计。

发明内容

本发明提供了设计微处理器计算机系统的一种方法，特别是用于具有利用流水线同步提高系统性能的协处理器的微处理器计算机系统的一种方法。我们改进了对乱序执行的支持，并提供了在计算机系统，特别是具有微处理器和协处理器的计算机系统中使用大容量高速缓冲存储器和多级高速缓冲存储器的能力，其中协处理器提供推测引擎来减少了有限高速缓冲存储器带来的性能恶化，从而提高了系统的性能。

改进的最佳实施例通过采用多级流水线同步的寄存器管理提供了更好的微处理器支持。这些改进是通过提供一种具有与基本上顺序处理（在允许处理如指令预取和在需要时同时完成加载等超标量技术时）的核心处理器一起工作的若干执行部件的推测读取引擎（SFE）、一种乱序执行的方法、一种使一组微处理器同步的方法以及一种允许产生对 SFE 和微处理器核心（uPCore）所共享的层次存储器的推测存储器参考的寄存器管理方法来实现的。

uPCore 和 SFE 都被看作处理部件。在具有第一处理部件和第二处理部件的计算机系统中系统对顺序指令流进行处理，这两种处理部件分别具有由它们自己的通用寄存器和控制寄存器的设置所决定的状态。在处理过程的任何时候，若由第二处理部件来接着继续进行同一指令流的后续处理是很有用的，则第一处理部件和第二处理部件可以对指令流进行并且有可能执行同一条指令，但是只允许所述处理部件中的一个（在最佳实施例中是 uPCore）改变由第一处理部件和第二处理部件的状态共同决定的所述计算机系统整个结构的状态。

在最佳实施例中，第二处理部件将具有比第一处理部件更多的流水步，用以允许进行乱序执行，从而减少有限高速缓冲存储器带来的性能恶化，提高系统的性能。第二处理部件的处理以及结果的存储都不改变此最佳实施例中计算机系统的结构状态。结果存储在其通用寄存器或私有存储缓冲区中。在遇到无效的操作、停滞或计算出对于将协处理器作为乱序协处理器(SFE)处理具有特定效益时要进行两个处理

部件状态的重新同步。

SFE 与 uPCore 相互接口，因此，本发明尤其适用于通过 SFE 和位于同一硅片上的 uPCore 来实现。也可以采用多个硅片的实现方式，并且包括在本发明的当前实施例中。uPCore 具有常规的结构并保持着最佳实施例中组合系统的结构状态。而在推广的镜像对应版本中保持结构状态的工作由另一处理部件来完成或是由两者共同完成。在此最佳实施例中，SFE 进行的操作不会直接地改变 uPCore 的结构状态。SFE 用来产生将uPCore要用到的指令和操作数预先填充到组合系统的高速缓冲存储器中的存储参考。这些改进扩展了由较早的寄存器重命名方案（如在美国专利 4,901,233（此后称为 Liptay）以及美国专利 4,574,349 所提出的）所可能达到的系统性能。

这些以及其它的一些改进在下面的详细描述中作出了说明。为了通过由国际商用机器公司首先提出的广泛地实现的以前的设计的优点以及特征来更好地理解本发明，请参考下面的说明以及附图。

附图说明

图 1A 和图 1B 提供了与 IBM 的美国专利 Liptay 4,901,233 所提供的同样的描述；来说明了 IBM 大型机以及 Intel 的 Pentium、Digital 的 Alpha、Sun 的 UltraSparc 之类的微处理器所广泛使用的以前成果的限制。

图 2 以图解的形式对最佳实施例进行了概述。

图 3 举例说明了 SFE 的细节和 SFE 与存储缓冲区、高速缓冲存储器、uPCore 之间的接口。还举例说明了 SFE 通过 SFE 和 uPCore 所共享的高速缓冲存储器来发送指令和取操作数的优选硬件。

图 4 更详细地说明了 uPCore 和 SFE 之间的同步单元。

图 5 更详细地说明了对 Liptay 重命名发明的改进以允许 uPCore 和 SFE 之间同步的寄存器。

图 6 说明了用于 uPCore 的优选硬件。

图 7 更详细地以数据流图的形式说明了推测引擎和微处理器 uPCore 的相互作用，举例说明了为提高性能所采取的方法。

（注意：为了举例说明方便，附图可能被分成若干部分，按照约定，在使用多页的时候我们将附图的上部放在第一页，后面的各页接着说明其后的部分。）

具体实施方式

在对本最佳实施例进行详细考虑之前，可能有必要通过举例来说明由 IBM 首先提出并在美国专利 Liptay 4,901,233 中说明的乱序微处理器设计的典型的现有技术。图 1 说明了这样一种现有技术乱序微处理器设计的典型技术，最初是由美国专利 4,901,233 进行说明的，它教导使用了寄存器管理系统（RMS）。RMS 允许使用比结构中所命名的寄存器更多的物理寄存器作为通用用途和程序分支之后的精确恢复。使用寄存器管理系统本质上是为了实现乱序执行。乱序执行能够显著地减少有限高速缓冲存储器影响带来的性能恶化这一点是公认的。美国专利 4,901,233 所说明的最佳实施例包括了对采用以前技术的顺序处理器设计的基本流水线的改进。要求这些改进将 RMS 集成到整个系统中并且导致形成更长的指令流水线，如更多的流水步或是每一流水步包含比顺序设计指令流水线更多的逻辑。美国专利 4,901,233 的最佳实施例实现了对先前的顺序设计在无限 L1 高速缓冲存储器 CPI 和有限高速缓冲存储器 CPI 方面的改进。本发明并不排除使用乱序技术来改进无限 L1 高速缓冲存储器 CPI，但也允许限制它们的使用来获得设计复杂性和主指令执行流水线的乱序支持两方面更好的平衡。本发明重点在于使用乱序技术来减少有限 L1 高速缓冲存储器 CPI 的增加而不增加 uPCore 流水线长度和流水线中每一流水步的长度。由于在数据库和事务工作量方面，改进的周期时间比美国专利 4,901,233 提供的在无限 L1 高速缓冲存储器 CPI 方面的小规模改进提供了更好的性能，因此总体的结果是得到了比美国专利 4,901,233 更好的系统性能。此外，本发明通过在 uPCore 作为顺序设计实现时，将寄存器管理系统从主指令处理流水线中独立出来，使得在涉及乱序指令处理的所有问题的设计复杂性有较大的降低。由于这一点，我将说明美国专利 4,901,233 所实现的图 1A 和 1B。

Liptay 发明是一种用于特定数目可寻址（逻辑）寄存器阵列（如通用寄存器）例如 n 个通用寄存器结构设计要求的计算机系统的寄存器管理系统。就像所说明的那样，Liptay 设计中的许多部件将在我的系统中用到。一个具有 m 寄存器的寄存器阵列（RA），其中 m 大于 n，被用来实现 n 个通用寄存器的功能。作为一个举例说明的实施例，根据已知的具有 16 个通用寄存器的 IBM System/320 结构的系统在美国专利 4,901,233 中作出了说明，并且直到今天本系统仍然在当前的 S/390 机器中采用。RA 提供了动态分配特定的 RA 位置来完成结构寄存器的功能。当特定的寄存器分配的功能完成之后，RA 中的位置被释放，并在合适的时候作为同一 GPR 或其它的结构 GPR 重新分配。

本寄存器管理系统并不依赖于整个计算机结构，而是可以在其它不同的环境中实现，就像在现在的微处理器设计中使用一样。因此，不论图 1A 和图 1B 中所说明的计算机系统是大型机的处理器还是微处理器，都相应地具有一个与高速缓冲存储系统 14 相连的主存储器 12。高速缓冲存储系统 14 可以以任何方式组织，在本例中是由与主存储器 12 相连的分别处理指令操作和数据操作的指令高速缓冲存储器 16 和数据高速缓冲存储器 18 所构成的。虽然图 1A 和图 1B 中并没有说明，能提供在层次排列和这种存储器设计中存储器速度和容量的优势的多级高速缓冲存储结构的层次存储设计仍然包括在本发明中，就像图 2 所举例说明的那样。

如图 1A 和图 1B 所示，指令从指令高速缓冲存储器 16 通过指令缓冲单元 20 传到指令寄存器单元 22。为了方便说明，指令寄存器单元拥有不止一个独立的指令寄存器，而是可以有 2、3 或者 4 个这样的寄存器。

作为执行单元的通用寄存器可以按照所执行的功能类别来分别设计，如算术单元和逻辑单元，标量单元和矢量单元，定点单元和浮点单元等等。由于通用执行单元的任意排列用到通用寄存器（GPR），本发明同样适用于计算机中通用执行单元的不同数目、功能的各种排列和设计。

为了方便说明，Liptay 系统与分别标为 24 和 26 的通用执行单元(GPE)1 和 2 一起说明。通用执行单元 24 的输出与存储缓冲区单元 28 相连，而存储缓冲单元的输出与数据高速缓冲存储器 18 相连。通用执行单元 24 实际上可以为单个的执行单元，也可以为组合执行单元，如实施例中所说明的，单元 24 产生送往存储缓冲区 28 的结果，在存储缓冲区 28 中这些结果被一直保持到指令执行完毕，并且之后可能存储到内存中。根据本发明，通用单元 26 的输出与通用寄存器组（RA）30 相连。通用单元 26 对指令进行操作，产生需要在寄存器中可使用而不是立即存储的结果。

一个指令堆栈或是指令队列 31 被用来从指令寄存器单元 22 接收指令，并直接将它们适应地送至 GPE 24 或 26。不同类型的多个执行单元可以与一个单独的 RA 和寄存器管理系统一起使用。RA 30 包括 32 个可动态指定的实际寄存器（物理寄存器）来完成结构所认可的 16 个 GPR 的功能。

RA 30 由寄存器管理系统(RMS)32 通过控制总线 34 控制，并向其提供状态信息。RMS 32 被连接到其它的几个系统上，用来接收或提供不同类型的状态信息。一个中断控制部件 36 与指令寄存器 22、RMS 32 以及 RA 30 相连来进行正确的中断处理并保留所需要的状态信息。

RMS 32 与指令寄存器单元 22 和 GPE 24、26 相连来从指令的执行中得到后续的指令以及为输入操作数、输出操作数进行寄存器的分配。

图 1A 和 1B 中的计算机具有一个与指令寄存器单元 22 相连并从指令寄存器单元 22 接收指令的指令队列，该指令队列输出到指令地址计算部件(I-ACE)52。I-ACE 52 还与 RA 30 相连并直接从 RA 30 接收输入，I-ACE 52 的输出与指令高速缓冲存储器 16 相连。指令队列 50 与 RMS 32 相连来提供状态信息。

图 1A 和 1B 中的计算机具有一与指令寄存器单元 22 相连并从指令寄存器单元 22 接收输出的地址队列 60，该地址队列 60 的输出作为输入连接到数据地址计算部件(D-ACE)62。D-ACE 62 的其它输入从 RA 30 获得，D-ACE 62 与 RMS 32 相连来提供状态信息。

D-ACE 62 的输出与地址读取队列 64 相连，地址读取队列 64 具有一个作为输入与数据高速缓冲存储器相连的输出和一个作为输入与地址存储队列 66 相连的输出。地址存储队列具有一个输出连接到数据高速缓冲存储器，并且还与 RMS 32 相连来提供状态信息。

计算机具有一个与 RMS 32 相连来提供状态信息的浮点算术单元 70。就像将要解释的那样，有一点很重要，RMS 32 可以与跟 RA 30 不关联的单元和寄存器一起工作。例如，一个 RMS 可以与多个寄存器阵列一起工作。更具体地说，一个 RMS 可以控制两个分别与多个相同或不同类型的指令执行单元相连的 RA。

浮点单元（FPU）70 的输入由浮点指令队列 72 和浮点数据寄存器单元 74 提供。浮点指令队列的输入从 I-REG 22 得到。浮点数据寄存器单元的输入从数据高速缓冲存储器 18 和 FPU 70 得到。浮点单元 70 的输出与存储缓冲区单元 76 相连，存储缓冲区单元 76 的输出作为数据高速缓冲存储器 18 的输入与其相连。

现在更详细地考虑本发明，可以看出，将说明的系统对图 2 所说明的具有大容量高速缓冲存储器和多级高速缓冲存储器的系统很有效。本发明提高了已有高速缓冲存储器的性能，推测预取机制提高了每一级高速缓冲存储器的命中率。如果某些情况下，通过 SFE 的硅片规模使得片内高速缓冲存储器得到增加，总体性能方面的效益应该可以通过所能够获得的来衡量。在某些情况下，这一比较不一定必须正确，例如 L1 高速缓冲存储器的情况，对 L1 高速缓冲存储器来说通常是周期时间的约束是关键的，而不是面积约束。初步的结果表明，使用大小大约为片内二级高速缓冲存储器的 1/4 到 1/2 的 SFE 大约能将性能提高 15%到 20%。

图 2 的最佳实施例

如图 2 所说明的最佳实施例所示，部件之间的互连是通过不同的接口实现的，如 uPCore （200）和同步单元（SU 201）、SFE（202）以及指令和数据高速缓冲存储器（203）之间的接口。高速缓冲存储器内存系统可以用任何合适的方法来组织，但在本例中是与层次存储器中

主存储器 204 相连的指令和数据的组合高速缓冲存储器 203 一起说明的，层次主存储器通过提供按层叠排列的多级高速缓冲存储器（如 203′、203″）来获得存储器规模和速度的优势，这一存储器设计包含于本发明中。分开的指令高速缓冲存储器和数据高速缓冲存储器同样也包含于本发明之中。

任何数目的 uPCore 200...200′...200″ 可以与任何数目的 SFE 202...202′...202″ 一起使用。一个 SFE 可以在任何给定的时刻与一个单独的 uPCore 相关连，也可以在同步功能完成之后关联到另外的 uPCore。每个 SFE 与一存储缓冲区和一 SU 相连。例如 201′、202′以及 205′一起使用来提供所要求的 SFE 功能。一个 uPCore 可以同时与任何数目的 SFE 相关连。最佳实施例中具有一个 SFE 和若干个 uPCore。

然而，在进行最佳实施例的详细硬件说明之前，在 uPCore 以其它的方式工作时，图 2 也可以实现另外一种上位的实施例。上位的图 2 也实现了此处说明的功能，但是其结构控制在以镜像方式实现相同功能的 200...200′...200″ 和 202...202′...202″ 之间交替轮换。

所以最佳实施例是可替换的上位的实施例的一个特殊最佳实施例，在上位的实施例中，第一个传统的处理部件 200，200′，200″ 和第二个处理部件 202，202′，202″ 一起工作，交替地控制机器的结构状态。在我们的图 2 所示的最佳实施例中，第一处理部件控制结构状态，顺序处理顺序指令流中的大部分指令。所以一般地，在计算机中处理顺序指令流的方法包括一个第一处理部件和一个第二处理部件，每一个处理部件具有由它自己的通用寄存器和控制寄存器的设置所决定的状态，开始时所述顺序指令流的初始化指令分配给所述处理部件中的第一个，如 200。所述顺序指令流的处理通过使用所述处理部件中的第一个来继续进行，所述处理部件中的第一个将任何结构状态的改变迅速转送所述处理部件中的第二个。然而，在所述第一个处理部件（如 uPCore 200）处理所述顺序指令流的任何时候，若第二处理部件（如 SFE 202）来开始继续对同一指令流进行处理是十分有用的，则计算机

系统的第二处理部件恢复已经转送的状态。通过由第二处理部件对同一顺序指令流进行处理来继续同一顺序指令流的处理。

然后第二处理部件将第一处理部件所要求的计算机系统结构状态的任何改变发送给第一处理部件。

不论是轮流控制的可选择实施例还是我们的最佳实施例，第一处理器和第二处理器可以处理同一条指令，但是所述处理部件中只有一个能够改变由第一处理部件和第二处理部件的组合决定的所述计算机系统的全部结构状态。在优选实施例中，这一组合是由第一处理部件所决定的。虽然在另外的实施例中，系统的结构状态可以由第二处理部件的状态完全决定或部分决定，但本最佳实施例中，第二处理 SFE 的工作并不改变系统的结构状态。uPCore 流水线顺序处理大多数顺序指令，同时 SFE 对指令进行处理来填充 uPCore 和 SFE 共享的高速缓冲存储器并尽可能与 uPCore 重新同步，uPCore 在 SFE 处理指令和 SFE 的结果存储到独立的私有存储缓冲区时控制结构状态，此时最佳实施例中的有限高速缓冲存储器的影响就得到减小。

在替换的控制实施例中，而不是真的优选实施例中，结构状态的控制不断来回切换。

在上述方法中，每一个所述处理部件仍然具有由它自己的通用寄存器和控制寄存器的设置所决定的状态，并且每个第一处理部件和第二处理部件在所述顺序指令流的处理过程中有可能执行同一条指令，但只允许所述处理部件中的一个来改变由第一处理部件和第二处理部件的组合确定的所述计算机系统的全部结构状态，但是，结构状态的控制可以由第一处理部件交给第二处理部件也可以由第二处理部件交回给第一处理部件。这一过程仍然通过使用所述的处理部件的第一个处理部件从将第二处理部件所要求的计算机系统中的任何改变发送给所述第二处理部件时开始，并将这些发送的改变积累起来用于所述第二处理部件将来的结构状态，来对所述顺序指令流进行处理。从而，当所述第一处理部件处理所述顺序指令流的任何时候，若由所述第二处理部件来接着继续处理同一指令流是很有用的，则第二处理部件恢复

累积的先前从所述第一处理部件发送来的结构状态，并通过处理所述顺序指令流来继续处理同一指令流。当第二处理部件控制所述顺序指令流的处理时，它将第一处理部件所要求的计算机系统的结构状态的任何改变发送给第一处理部件，以供将这些改变积累起来形成将来要用到的结构状态。然后控制可以再次更换，当所述第二处理部件处理所述顺序指令流的任何时候，若由所述第一处理部件来接着继续处理同一指令流是很有用的，则第一处理部件恢复累积的先前从所述第二处理部件发送来的结构状态，并处理所述顺序指令流来继续处理同一指令流。

现在第一处理部件和第二处理部件可以作为多处理器工作。同样，如 200，200′ 和 200″ 所说明的，第一处理器可以包括作为多个处理器与单个 SFE 或是多个 SFE 工作的若干第一处理部件。然而，多个 SFE 不能与单个的 uPCore 一起使用。那就是说，一个多处理器可以与一组一个或多个第一处理部件和至少一个第二处理部件的组合一起工作。在最佳实施例中，同步的能力以一个同步单元 SU 201，201′，201″ 的形式提供给每一个第二处理部件。该 SU 决定了第二处理部件 SFE202，202′ 和 202″ 何时开始处理正在被第一处理部件 uPCore 处理的同一条指令。所以，为每一个 SFE 提供了一个同步单元，SU 决定了 SFE 何时开始处理同一条指令或正在由 uPCore 处理的指令流中的另一条指令。SU 决定了 SFE 处理部件对指令的处理何时应该被停止或是被忽略。这一决定是由计算出来的整个计算机系统的利益决策和第一处理部件和第二处理部件提供给同步单元的输入共同作出的。该输入可以在当前提供给同步单元，或从系统存储的信息得到，如图 4 中所示，其中计数器 407 和 408 提供这一信息。

如果第一处理部件在处理指令时有一个停滞决定，如在图 7 的 709 中，同步单元将决定第二处理单元何时开始处理它所处理的同一条指令。当处理指令时出现了第二处理部件所不能处理的操作时，如得不到合法的操作数（707），在最佳实施例中，同步单元将通过 SFE 和 uPCore 状态的重新同步来决定何时将第二处理部件的状态与计算机系

统的结构状态同步。如果检测到第二处理部件在处理指令流时不能给计算机系统带来任何好处（特定的效益判断 208），则同步单元决定何时将第二处理部件的状态与计算机系统的结构状态重新同步。图 7，707，708，709 中所说明的判断不但决定何时存在由同步单元实现的重新同步，而且决定用哪一个处理部件来重新同步状态。处理器对指令进行预处理，SFE 将它的结果存储到它的私有 GPR 或存储缓冲区 205，205′ 或 205″ 中。这一存储过程并不影响其它处理部件的结构状态，SFE 可以处理下一条指令或是处理流中正在由所述第一处理部件处理的同一条指令，SU 可以决定所述第二处理部件何时应该被停止或取消，这些独立的同步允许 SFE 提高处理器处理指令流中大多数指令的性能。第一处理部件从被所述第一处理部件和第二处理部件共享的用于读取的数据和指令高速缓冲存储器中取出数据。

最佳实施例的方法允许 SFE 能作为乱序处理器用于对所述顺序指令流进行预处理来为第一处理部件填充高速缓冲存储器。在重新同步期间，当所述第二处理部件对指令的处理应该停止或取消时，第二处理部件把重新同步之前为第一处理部件进行的指令流预处理的所有结果和局部结果清除。

因而可以理解，在最佳实施例中，除了私有存储缓冲区 205 之外，SFE、同步单元和两个 uPCore（代表一组 uPCore）也用于上面所讨论和图 7 中所说明的方法中。可能的状态为：运行(A)，清除(B)， SFE 与 uPCore 200 的重新同步(C)，以及 SFE 与 uPCore 200′ 的重新同步(D)。SFE 的初始状态为(C)。在状态(C)中，SFE 接收最近从 uPCore 200 中退出的指令地址以准备在该地址开始乱序执行。同步单元 201 通过每一个 uPCore 与 SFE 运行来继续监控 SU 与 uPCore 之间的接口以指出 uPCore 由于一次高速缓冲存储器未命中而停止。uPCore 运行并通过接口 210 不断地查询高速缓冲存储器存储器和主存储器。指令和操作数通过接口从指令和数据高速缓冲存储器 203 返回到 uPCore。

从重新同步到运行(状态 A)的状态转换发生在 SFE 的寄存器管理系统将与 uPCore 相关的 SRAL 的内容装入到 SFE 的 DRAL 中的时候。在进

入 SFE 运行状态时，SFE 在最近通过接口 206 从 uPCore 200 中接收到的指令地址开始指令的读取和执行。SFE 的 GPR 状态表示了同一指令地址指向的指令退出时 uPCore 的状态。当 SFE 为运行状态时，通过接口 206 接收的 GPR 结果将继续写入通用寄存器阵列，但寄存器管理系统会将它们关联到一个同步寄存器分配表中。它们将只会在同步事件之后被 SFE 中执行的指令所使用。这样，SFE 保持了它所关联的每一个 uPCore 的 GPR 状态的独立的映象，它可以在以后对其进行访问。同时，SFE 的 RMS 只使用 SFE 的执行结果来更新用于 SFE 的指令流执行的 GPR 的映象。

在 SFE 进入运行状态之后，uPCore 继续按照它自己从高速缓冲存储器 203 中读取的速度执行，高速缓冲存储器 203 中包括由 uPCore 常规处理部件在使用之前预先提供指令和操作数给高速缓冲存储器 203 的推测引擎 SFE 处理部件存储参考，而 SFE 将开始执行乱序指令。在最佳实施例中，uPCore 可以专门地设计为一个顺序处理器，或者经过顺序处理最优化的处理器，或能够处理有 95%的指令不能从预测中获益的指令流中的指令的处理器。在一级高速缓冲存储器未命中时会出现一次流水线停滞。由于 SFE 的乱序执行能力，SFE 可以跳过引起停滞的指令继续工作。在 SFE 运行的时候，它产生了通过接口 207 送到指令和数据高速缓冲存储器和通过接口 208 送到存储缓冲区的读取参考。如果高速缓冲存储器和存储缓冲区中都没有合适的数据时，就出现了高速缓冲存储器未命中。如果在存储缓冲区中没有相应的入口，指令和操作数就通过接口 207 返回到 SFE；而如果有相应的入口就通过接口 208 返回。SFE 存储参考没有送到指令和数据高速缓冲存储器而是送到了存储缓冲区。这样，SFE 存储指令的结果可以被 SFE 中后面执行的指令所用到而不必改变 uPCore 和高速缓冲存储器的结构状态。所有 SFE 的存储都保存在存储缓冲区中。

同步单元通过接口 209 监测 SFE 的活动。如果 SFE 的执行超出了支持范围的指令或是遇到了设计时没有处理的或者是非法的中断或异常，这些都会在接口 209 指出，同步单元然后将 SFE 转入图 7 中的清

除状态（B）。同步单元同样监测 uPCore 的指令解码过程和 SFE 的指令退出过程。如果没有其它的有效操作 707，或者检测出 SFE 没有提供推测预取获益 708，SFE 就被假设落在 uPCore 的执行后面太远了，这样也同样要进入清除状态（B）。如果当前与该 SFE 相关联的 uPCore 仍然停滞在决策点（709），那么会暂缓进入清除状态，SFE 仍然处理运行状态。SFE 收益可以用于决定 SFE 何时进入清除状态的其它说明包括在本发明中。

一旦 SFE 进入了清除状态（B），它将一直保持着一状态直到所有指令、指令的部分以及部分结果已经从 SFE 的数据通路和控制结构中清除。在此期间，没有任何请求被送到指令或数据高速缓冲存储器。当这些已经完成 706 时，SFE 离开清除状态，能够进入 C 和 D 两个状态之一。SFE 即可以与 uPCore 200 也可以与 uPCore 200′ 进行重新同步。由 SFE 决定的在这两个操作之间的选择 704 可以基于一系列因素，这些因素都包括在本发明之中。最佳实施例中使用了一个简单的指示来指出哪一个 uPCore 是最后与 SFE 同步的，在哪种情况下 SFE 将使用其它的 uPCore 来进行同步。使用其它的算法有可能出现同一个 uPCore 被通过决策点 704 来选择多次。一旦重新同步完成，状态就会再一次改回为运行状态，又重新开始另一个周期。

推测引擎（SFE）

SFE 使用常规的乱序处理并另外采取了特定的称为超标量技术的功能或技术来产生推测的操作数和指令预取。这些技术包括寄存器重命名，指令重排序、完成计分等等。SFE 具有很多种不同的可能的实现方法。理想设计的标准将包括与当前乱序设计大大不同的周期时间和面积约束。图 3 说明了 SFE 和它与系统其它部件的接口的细节。这一过于简化的流程用来突出该具有通用寄存器阵列和指令处理流水线的寄存器管理系统的新颖的相互作用。这一点与图 1A 和图 1B 相似，但是存在着重要的区别。首先，存在着构成 GRP 和 SFE 之间接口 206 的一部分的附加接口 306。第二是当前发明的 RMS 301 被改成包括同步寄存器配表（SRAL）的使用。第三是对存储器层次的存储被送到存储缓

冲区 205 而不是像美国专利 4,901,233 即 Liptay 中说明的指令和数据高速缓冲存储器。SFE 数据流程继续通过图 1A 和图 1B 从美国专利 4,901,233 即 Liptay 如图 3 所示的那样到达存储缓冲区 205。

接口 302、303、304 和 305 包括接口 209 的一部分，分别传送同步地址、清除指示、重新同步指示和指令解码指示。同步地址由 SFE 在重新与一 uPCore 的结构状态同步之后用作指令读取和执行的起点。清除 SFE 指示将使得 SFE 抛弃所有指令结果和局部结果，并清除 SFE 存储缓冲区中的内容。重新同步指示由 SFE 用来决定应该与哪个 uPCore 进行同步，并决定重新同步应该何时结束。SFE 使用指令完成接口来指示 SU 指令已经解码成功。SU 使用这一信息决定 SFE 是否提供了推测读取效益。SFE 通过接口 307 向指令和数据高速缓冲存储器发送指令和操作数读取请求，并通过接口 308 向存储缓冲区发送读取请求。通过接口 307 发送的推测读取由 SFE 在 uPCore 在停滞之后恢复执行时将进行同样的读取请求时预先实现。由于要得到的目标已经被访问过并装入到最近等级的高速缓冲存储器中，从而 uPCore 缩短了这些读取请求的响应时间。

由于 SFE 独立于 uPCore 的结构状态，乱序指令处理的实现与大多数结构无关。这改进了进度并减少了整个设计的周期时间的影响。与 SFE 关联的实现现在可以与 uPCore 完全分开。SFE 可以以要满足不同的大指令集的需要的 uPCore 所无法采用的方式来进行优化来产生推测读取。SFE 并不需要执行任何不常用的指令、异常处理操作或恢复算法。遇到这些不常用的操作时，SFE 将停止指令流的执行并通知同步单元。如果不常用的操作继续存在，uPCore 最终将离开停滞状态，以更简单的顺序设计的方法对它进行处理。

必须对 SFE 设计进行优化来解码并迅速发布大量的指令而不必只对无限 CPI 是正确的。SFE 可以被设计为比先前的设计具有更长的指令流水线而不必考虑无限高速缓冲存储器性能影响。包括 SFE 和 uPCore 的整个系统中无限一级高速缓冲存储器的性能，只依赖于 uPCore 流水线而不是 SFE。

在本发明的设计中，操作数预取不需要由 uPCore 实现，如果需要，SFE 系统的使用从 uPCore 中消除了这一特征和它的相关的复杂性。有几种情况中操作数预取可能需要保留在 uPCore 中，这是与当前发明相一致的。

对 RMS 的创新改动的细节在根据本发明的最佳实施例的图 5 中进行了说明，SFE 为与该 SFE 相关的每一个 uPCore 维护一个同步寄存器分配数组（SRAL）。本发明的寄存器管理系统，包括 SRAL 使用的扩展，不依赖于整个计算机结构并且可以在不同的环境中实现。因此，不用限制本发明的范围，根据具有 16 个通用寄存器（GPR）的 IBM 系统 390 结构对图 3 中根据本发明说明的 SFE 进行了说明。GPR 寄存器阵列（RA）与 RMS 一起提供动态的特定寄存器分配和特定的 RA 状态分配来完成结构寄存器的功能。当一个特定的寄存器的功能完成时，RA 中的状态被释放，并且可以作为同一 GPR 或其它 GPR 在适当的时候重新分配。

RA 包括 48 个可动态分配的实际（物理）寄存器来完成本发明的该实施例中认可的 16 个 GPR 的功能。一个解码寄存器分配表（DRAL）在指令解码时被用来将 GPR 分配转化为 RA 分配。当每一个指令解码时，在 DRAL 中查找 GPR 的参考来决定将 RA 的什么状态分配给 GPR，并在分配新的 RA 状态来接受结果时，DRAL 被更新来反映这些分配。这样，DRAL 指向使用 GPR 的每一条指令来找到分配给最近的指令来参考该 GPR 的 RA 状态。

备份寄存器分配表允许分别对一、二或者三个条件分支进行处理而不必等待。它的结构与 DRAL 相似，并以每周期将 DRAL 的整个内容复制到 BRAL 的方式与 DRAL 相连或反之亦然。这些传输由逻辑单元 505 进行控制。当遇到一个条件分支的时候使用 BRAL 将 DRAL 的内容保存以免所猜测的分支不正确。

一个数组控制表（ACL）与 RA 和 SFE 的部分相连来接受信息和发送控制信息。逻辑单元 505 控制 ACL 并调整 ACL、DRAL 以及 BRAL 的操作。对于每一个支持 GPR 的 RA 都存在一个 ACL 寄存器来记住与该 RA

有关的状态信息。对于数组中的每一个状态都有对应的一个入口。

寄存器管理系统所附加的SRAL对于SFE的功能和本发明来说都是关键的。SRAL具有与DRAL相同的结构，并以每周期将SRAL的整个内容复制到DRAL的方式与DRAL相连。

与SFE关联的每一个uPCore都被提供了一个SRAL。当uPCore产生GPR和CR的更新时，它们的更新被通过接口206发送到SFE。结果可以延迟一个周期，从而将uPCore的周期时间影响减到最小。GPR的更新被写进RA并更新与源uPCore相关联的SRAL来指向RA的位置。由于当前实施例中的uPCore通常以顺序执行设计的方式运行，接口206中的GPR的更新反映了退出的指令的GPR的更新，从而随时可以写入SRAL当前所指示的同一个RA。在重新同步操作期间，SRAL必须与16个新的RA入口一起提供以保证可以容纳从uPCore送来的连续的更新。在当前的实施例中，由于同步操作通常在释放所有RA入口而不仅是与SRAL相关联的RA入口的SFE清除操作之后，所以这一点不成问题。复制在SRAL中的SFE中的uPCore的GPR状态通常有最少1个周期的延迟。当SFE需要与uPCore同步时，将SRAL的内容简单地移到DRAL中就可以完成这一任务。这一操作与Liptay中分支预测失败时使用BRAL来恢复微处理器的状态相似。

本发明的SRAL的功能与Liptay中的BRAL有着显著的不同。首先，SRAL中写入的是其它指令处理流水线所发送的GPR更新。第二，引起SRAL中内容转移到DRAL中的原因与Liptay中引起BRAL中内容转移到DRAL中的原因不同。在Liptay中是由分支预测失败所引起的。在本发明中，没有产生预取效益的指示将引起这一转移。因而，可以理解在SRAL的功能方面美国专利4,901,233以及其商业实施例是与本发明完全不同的。BRAL不能用于这一目的，在本发明中它用于和Liptay中所介绍的相同的功能，在检测到分支预测错误之后恢复处理器的状态。第三点重要的区别是，当SRAL中的内容转移到DRAL时，SRAL中的每一入口被马上改变，来指向16个新的RA状态。在Liptay中，在进行未决定的分支的解码时，BRAL将直接从DRAL中装入。

可以使用不止一个的 SRAL 来允许 SFE 与更多的 uPCore 同步。两个或更多的 uPCore 可以使用同一个 SFE 来提供预取效益，但不能同时使用 SFE。每一个附加的 SRAL 必须由相关的 uPCore 的 GPR 结果总线和存储缓冲区来协同工作以实现同步。

uPCore

在当前的最佳实施例中 uPCore 设计是一个常规的微处理器（较为适合的是当前的超标量设计，如 Motorola 和 IBM 推出的 PowerPC 601，但也可以为更老一些的设计如 Intel 286）。大家都知道，在计算机设计领域中系统可以拥有不止一个通用执行单元。例如，通用单元可以按照所完成的功能类型来设计。虽然图中只说明了 uPCore 中的两个这样的通用执行单元，但任何数目通用执行单元的使用都包括在本发明中。除了图 6 中所说明的外，本发明中的 uPCore 部分不要求对传统微处理器设计进行任何改动。图 6 说明了最近退出的指令的地址是怎样锁存在 604 并通过接口 604′ 送到 SFE 的。通用执行单元 601 和 602 的 GPR 结果总线也锁存在 603 并通过接口 603′ 送到 SFE。图 6 中说明的 uPCore 是一顺序设计，但使用任何乱序设计，如其它当前商业用途的微处理器，也同样包括在本发明之中。

同步单元

同步单元(SU 201)包括了所有要求的逻辑功能来控制 uPCore 和 SFE 的交互。SU 由一状态机和相关的输入寄存器 404、405 和 406 组成。状态机的输出包括到控制清除功能和输入到寄存器管理系统的 SFE 的接口 209。到 RMS 的线负责在同步操作时控制将 SRAL 装入到 DRAL。同步单元包括用来检测 SFE 是否为整个系统带来了预取效益的逻辑功能。当前实施例使用两个指令计数器 408 和 407 来实现这一功能。第一个计数器在 uPCore 每退出一条指令时增量。第二个计数器在 SFE 每解码一条指令时增量。这两个计数器在重新同步操作期间被重新设置为 0。重新同步之后，将对这两个计数器进行比较来决定 SFE 是否有机会产生对 uPCore 由帮助的推测读取参考。如果 SFE 不能在 uPCore 执行前足够长的时间将指令解码，那么将不可能带来效益。两个计数器的比

较提供了一个不太精密但是足够的可能产生效益的指示，并可以作为图 7 中特定效益决定点 708 的输入。当前实施例采用阈值 10 来实现这一功能。当 SFE 解码的数目没有比 uPCore 退出的数目至少大 1 0 时，同步单元将指示没有效益。

同步单元同样保留一个指示来指出与 SFE 当前相关联的是哪一个 uPCore。每一个 SFE 具有一个独立的同步单元但是每一个 SFE 可以与任何数目的 uPCore 相关联。当前实施例中一个 SFE 与两个 uPCore 相关联。

CP 和 SFE 交互的其它扩展

CP 和 SFE 的交互还可以有其它的扩展。在一个例子中将包括使用 SE 来更新由 SE 和 CP 共享的分支预测表。SE 同样可以向 CP 提供关于可能的指令异常和其它可能使得 CP 避免流水线中断的条件的指示。响应 SFE 读取请求所读取的指令和操作数可以直接发送到 uPCore。从而在推测请求为精确的时候，数据将离 uPCore 的通用执行单元和指令解码逻辑更近。在一些情况下，这将很大程度上减少有限高速缓冲存储器的影响。

在对本发明的最佳实施例进行了说明之后，大家可以理解，有经验的技术人员，现在或是将来都可能在本发明的范围内进行各种不同的实现或改进。

进行不同实现的人们将会对所说明的性能分析感到满意。该性能分析说明了乱序（或连续）执行在减小有限一级高速缓冲存储器 CPI 时比在减小无限一级高速缓冲存储器 CPI 时提供了更大的效益。当前的技术趋势表明有限高速缓冲存储器影响正在迅速增长，从而有限一级高速缓冲存储器 CPI 的效益比无限一级高速缓冲存储器 CPI 的效益更重要。

本发明所详细说明的关于支持核心微处理器的推测读取引擎(SFE)，以及在允许对 SFE 和微处理器核心(uPCore)所共享的层次存储器的推测内存参考时它们与维持一致操作的结构状态的核心微处理器之间的交互的规定，对那些想使用乱序执行来获得对先前技术设计的

显著简化或对没有采取乱序执行的先前技术设计进行显著的改进的人们将会是十分有帮助的。理想地讲，本发明允许为了寻求更好的系统性能而使用乱序执行的更优化的设计均衡。本发明允许微处理器优化成更高的频率、更低的复杂性以及更小的一级高速缓冲存储器 CPI 而不必对主流水线增加太多乱序执行复杂度，与有些当前设计中采用的不断增加的深度步长相反。

同时，协处理器可以使用乱序执行技术来在寻求减少微处理器和协处理器的有限高速缓冲存储器影响方面取得更大的范围。协处理其中乱序执行的复杂度可以通过协处理器不需要支持全部结构指令或与指令执行相关的全部异常和中断这一事实来减小。以下的权利要求应该解释为包括更多的改进并保持最先公开的发明的适当保护。

图1A

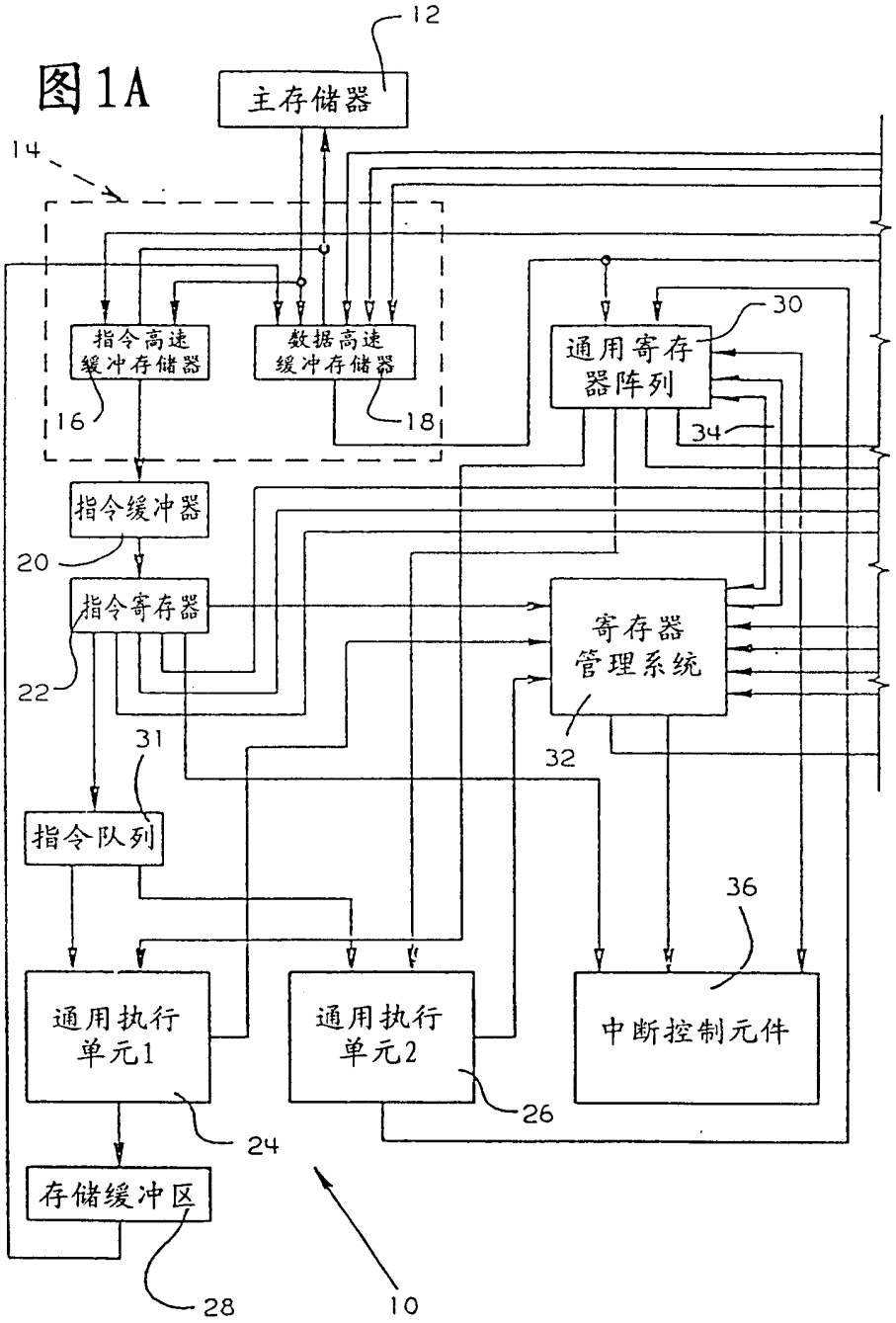

12
主存储器
14
指令高速缓冲存储器
数据高速缓冲存储器
16
18
30
通用寄存器阵列
34
指令缓冲器
20
指令寄存器
22
寄存器管理系统
32
31
指令队列
36
通用执行单元1
通用执行单元2
中断控制元件
26
24
存储缓冲区
28
10

图1B

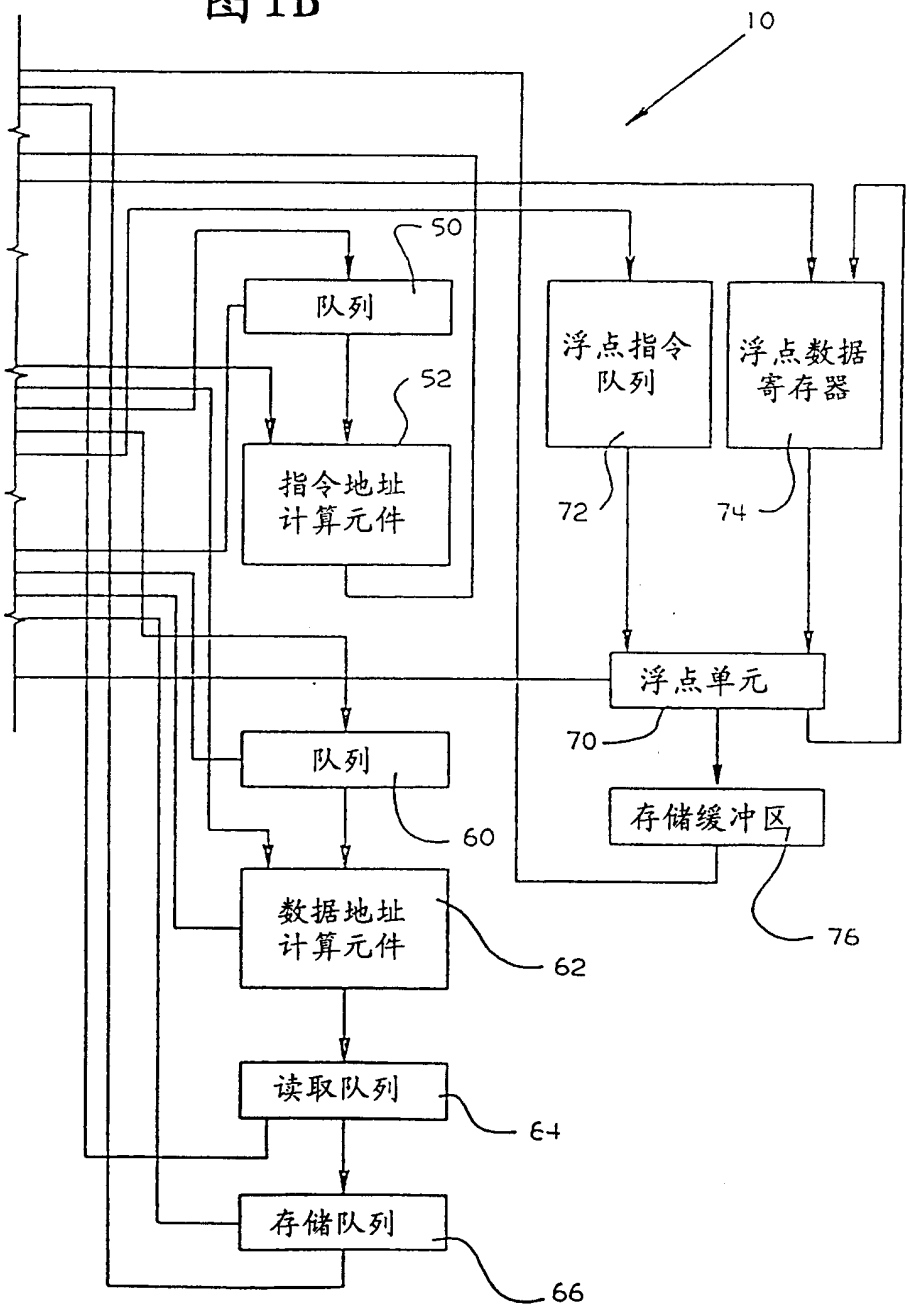

10
50
队列
52
指令地址
计算元件
浮点指令
队列
浮点数据
寄存器
72
74
浮点单元
70
队列
60
存储缓冲区
76
数据地址
计算元件
62
读取队列
64
存储队列
66

图 2

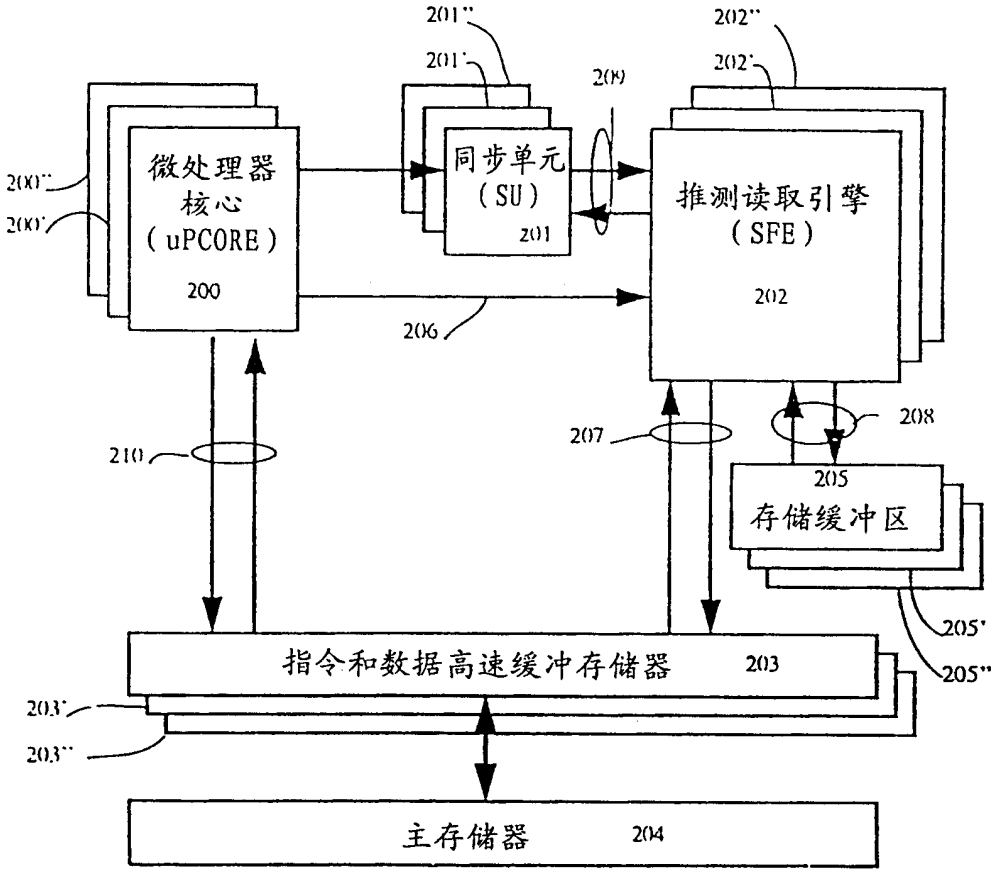

微处理器
核心
(uPCORE)
200
同步单元
(SU)
201
推测读取引擎
(SFE)
202
存储缓冲区
205
指令和数据高速缓冲存储器 203
主存储器 204
206
207
208
210

图 3

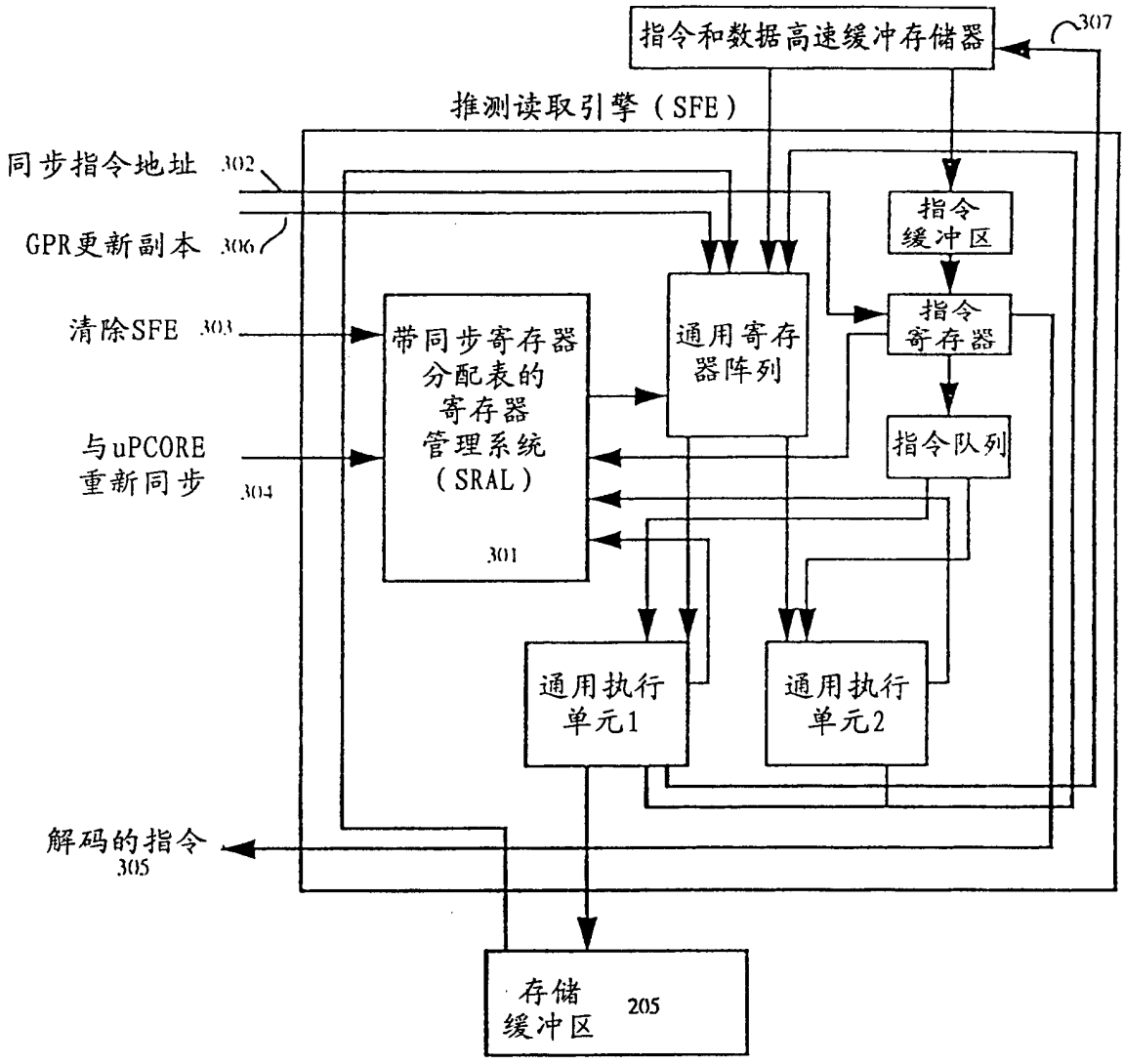

指令和数据高速缓冲存储器
307
推测读取引擎（SFE）
同步指令地址 302
GPR更新副本 306
清除SFE 303
与uPCORE
重新同步 304
带同步寄存器
分配表的
寄存器
管理系统
（SRAL）
301
通用寄存
器阵列
指令
缓冲区
指令
寄存器
指令队列
通用执行
单元1
通用执行
单元2
解码的指令
305
存储
缓冲区
205

图4

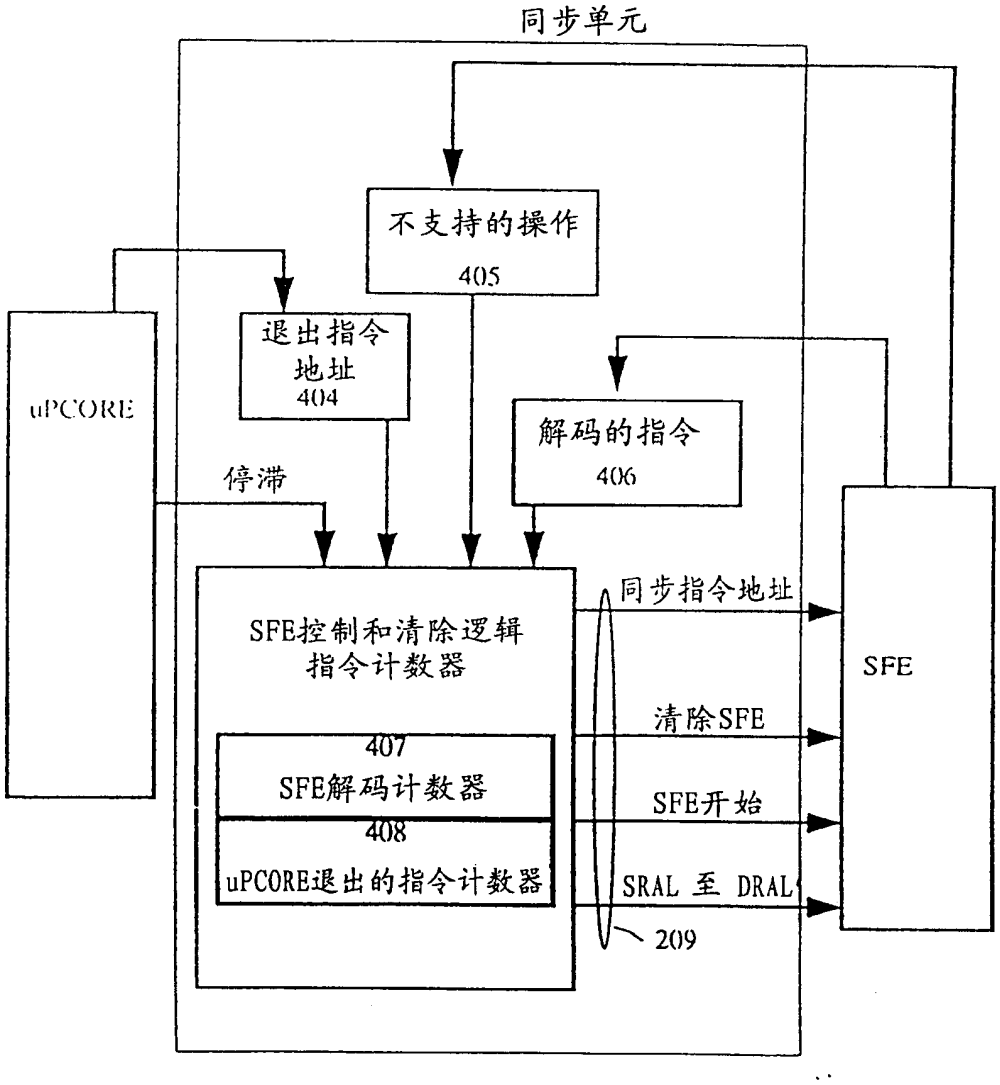

同步单元
不支持的操作
405
退出指令
地址
404
解码的指令
406
uPCORE
停滞
SFE控制和清除逻辑
指令计数器
407
SFE解码计数器
408
uPCORE退出的指令计数器
同步指令地址
清除SFE
SFE开始
SRAL 至 DRAL
209
SFE

图5

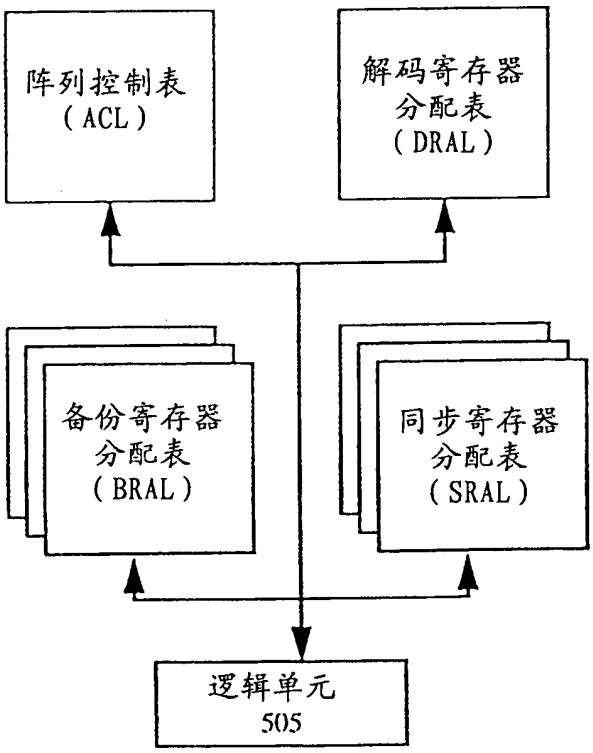

阵列控制表
(ACL)
解码寄存器
分配表
(DRAL)
备份寄存器
分配表
(BRAL)
同步寄存器
分配表
(SRAL)
逻辑单元
505

图6

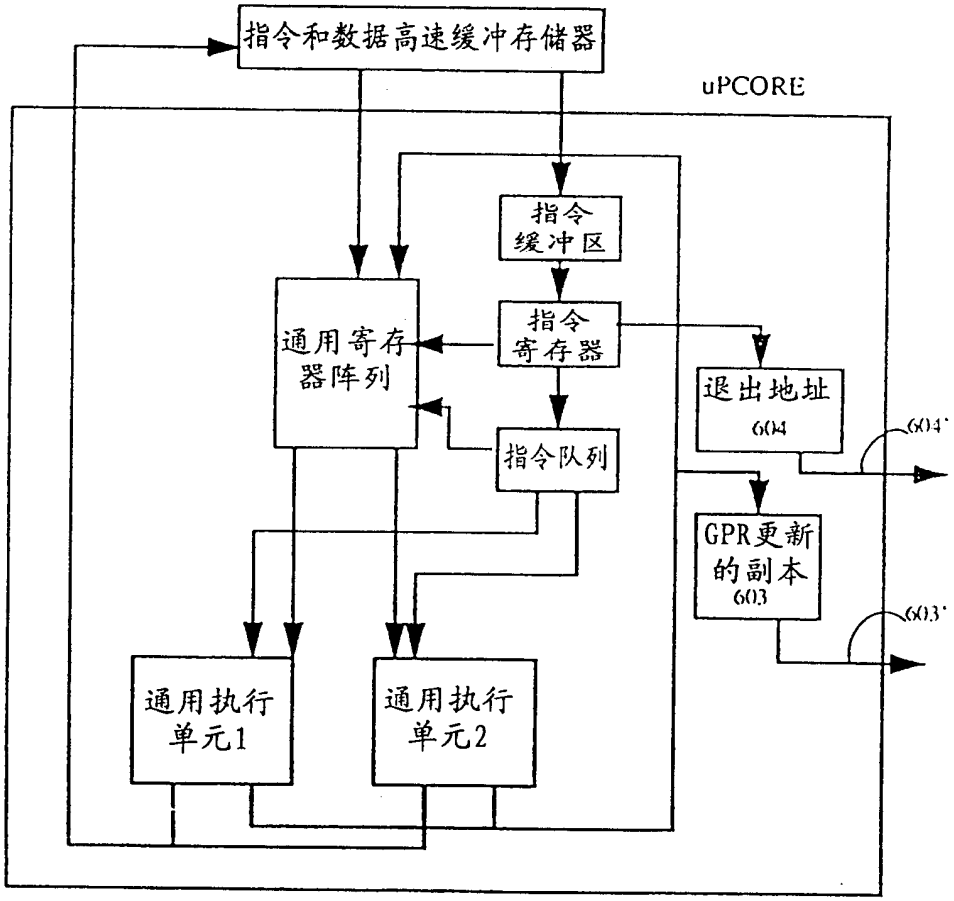
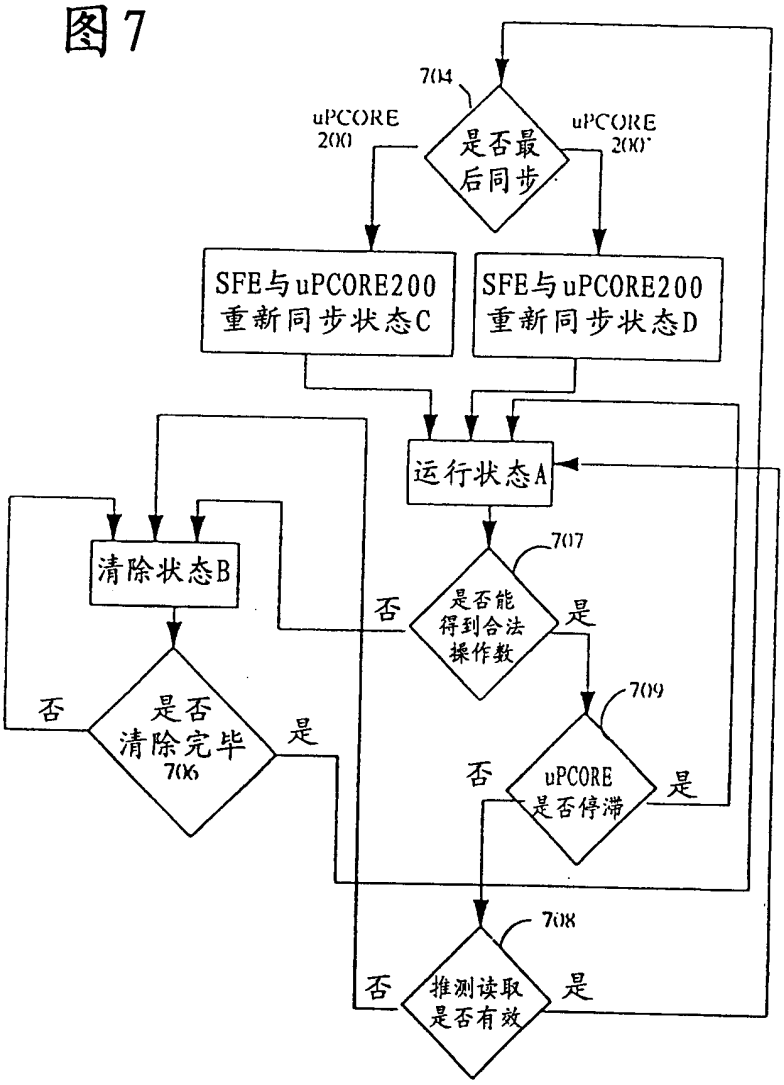

指令和数据高速缓冲存储器
uPCORE
指令
缓冲区
通用寄存
器阵列
指令
寄存器
退出地址
604
604'
指令队列
GPR更新
的副本
603
603'
通用执行
单元1
通用执行
单元2

图7


704
uPCORE 200
是否最后同步
uPCORE 200'
SFE与uPCORE200重新同步状态C
SFE与uPCORE200重新同步状态D
运行状态A
清除状态B
707
否
是否能得到合法操作数
是
709
否
是否清除完毕 706
是
否
uPCORE 是否停滞
是
708
否
推测读取是否有效
是