(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 102880447 A
(43)申请公布日 2013.01.16

(21)申请号 201210164802.7
(22)申请日 2004.08.26
(30)优先权数据
60/499,180 2003.08.28 US
60/502,358 2003.09.12 US
60/502,359 2003.09.12 US
10/684,348 2003.10.10 US
(62)分案原申请数据
200480024800.1 2004.08.26
(71)申请人 美普思科技有限公司
地址 美国加利福尼亚州
(72)发明人 凯文·基塞尔
(74)专利代理机构 永新专利商标代理有限公司 72002
代理人 舒雄文 蹇炜
(51)Int.Cl.
G06F 9/30(2006.01)
G06F 9/38(2006.01)
G06F 9/48(2006.01)

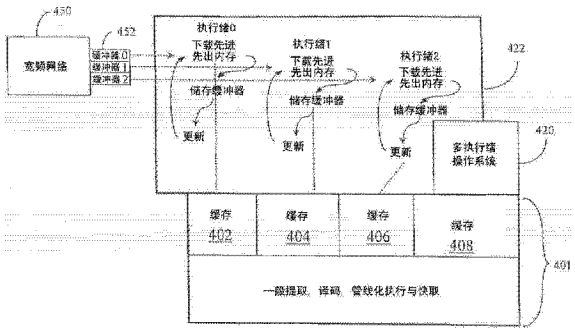
权利要求书 4 页 说明书 23 页 附图 32 页

(54)发明名称

一种在处理器中挂起和释放执行过程中计算线程的整体机制

(57)摘要

一种在能支持和执行多个程序线程的处理器中的处理机制，包括用于调度一个程序线程的参数和该程序线程中的一个指令，该指令能够存取该参数。当该参数等于第一数值时，且当一个程序线程发出该指令时，该指令根据编码在该参数中的一个或多个条件而重新调度该程序线程。



CN 102880447 A

1. 在一个能够支持与执行多程序线程的处理器中，一种由一个线程重新调度执行或释放该线程本身的设备，包括：

(a)用于发出一个指令的装置，该指令存取一个数据储存装置中的一部分记录，该部份记录编码了与决定该线程是否被重新调度的一个或多个条件相关的一个或多个参数；以及

(b)用于根据在该部分记录中的该一个或多个参数来依照该条件重新调度该线程或释放该线程的装置。

2. 如权利要求 1 所述的设备，其中该记录被置于一个通用寄存器(GPR)中。

3. 如权利要求 1 所述的设备，其中这些参数中的一个参数与被释放的而不是被重新调度的线程相关。

4. 如权利要求 3 所述的设备，其中与该被释放的线程相关的该一个参数为一零值。

5. 如权利要求 3 所述的设备，其中这些参数中的一个参数与被重新排队等待调度的线程相关。

6. 如权利要求 5 所述的设备，其中该一个参数的值为一任意奇数值。

7. 如权利要求 5 所述的设备，其中该一个参数的值为负 1 的二进制补码值。

8. 如权利要求 1 所述的设备，其中这些参数中的一个参数与将执行机会让与其它线程直到一个特定条件被满足的该线程相关。

9. 如权利要求 8 所述的设备，其中该一个参数被编码于该记录中的位向量或是一个或多个值字段之一中。

10. 如权利要求 3 所述的设备，其中，在该线程发出该指令并被重新调度的情形下，当该一个或多个条件被满足时，该线程的执行会在该线程指令流中该线程所发出的该指令之后的位置继续。

11. 如权利要求 1 所述的设备，其中这些参数中的一个参数与被释放的而不是被重新调度的线程相关，并且这些参数中的另一个参数与被重新排队等待调度的线程相关。

12. 如权利要求 1 所述的设备，其中这些参数中的一个参数与被释放的而不是被重新调度的线程相关，并且这些参数中的另一个参数与将执行机会让与其它线程直到一个特定条件被满足相关。

13. 如权利要求 1 所述的设备，其中这些参数中的一个参数与被重新排队等待重新调度的线程相关，并且这些参数中的另一个参数与将执行机会让与其它线程直到一个特定条件被满足相关。

14. 如权利要求 1 所述的设备，其中这些参数中的一个参数与被释放的而不是被重新调度的线程相关，这些参数中的另一个参数与被重新排队等待调度的线程相关，并且这些参数中的再另一个参数与将执行机会让与其它线程直到一个特定条件被满足相关。

15. 如权利要求 1 所述的设备，其中该指令为一个 YIELD 指令。

16. 在一个能够支持与执行多程序线程的处理器中，一种由一个线程重新调度执行或释放该线程本身的方法，包括：

(a)发出一个指令，该指令存取一个数据储存装置中的一部分记录，该部份记录编码了与决定该线程是否被重新调度的一个或多个条件相关的一个或多个参数；以及

(b)根据在该部分记录中的该一个或多个参数来依照该条件重新调度该线程或释放该

线程。

17. 如权利要求 16 所述的方法，其中该记录被置于一个通用寄存器（GPR）中。

18. 如权利要求 16 所述的方法，其中这些参数中的一个参数与被释放的而不是被重新调度的线程相关。

19. 如权利要求 18 所述的方法，其中与该被释放的线程相关的该一个参数为一零值。

20. 如权利要求 16 所述的方法，其中这些参数中的一个参数与被重新排队等待调度的线程相关。

21. 如权利要求 16 所述的方法，其中该一个参数为一任意奇数值。

22. 如权利要求 21 所述的方法，其中该一个参数为负 1 的二进制补码值。

23. 如权利要求 16 所述的方法，其中这些参数中的一个参数与将执行机会让与其它线程直到一个特定条件被满足为止的该线程相关。

24. 如权利要求 23 所述的方法，其中该一个参数被编码于该记录中的位向量或者一个或多个值字段之一中。

25. 如权利要求 16 所述的方法，其中，在该线程发出该指令并被重新调度的情形下，当该一个或多个条件被满足时，该线程的执行会在该线程指令流中该线程所发出的该指令之后的位置继续。

26. 如权利要求 16 所述的方法，其中这些参数中的一个参数与被释放的而不是被重新调度的线程相关，并且这些参数中的另一个参数与被重新排队等待调度的线程相关。

27. 如权利要求 16 所述的方法，其中这些参数中的一个参数与被释放的而不是被重新调度的线程相关，并且这些参数中的另一个参数与将执行机会让与其它线程直到一个特定条件被满足相关。

28. 如权利要求 16 所述的方法，其中这些参数中的一个参数与被重新排队等待重新调度的线程相关，并且这些参数中的另一个参数与将执行机会让与其它线程直到一个特定条件被满足相关。

29. 如权利要求 16 所述的方法，其中这些参数中的一个参数与被释放的而不是被重新调度的线程相关，这些参数中的另一个参数与被重新排队等待调度的线程相关，并且这些参数中的再另一个参数与将执行机会让与其它线程直到一个特定条件被满足相关。

30. 如权利要求 16 所述的方法，其中该指令为一个 YIELD 指令。

31. 一种支持与执行多个软件实体的数字处理器，包括：

一个数据储存装置中的一部分记录，该部分记录编码了与一个或多个条件相关的一个或多个参数，该一个或多个条件决定了当一个线程将执行机会让与其它线程时该线程是否被重新调度或释放。

32. 如权利要求 31 所述的数字处理器，其中该部分记录被置于一个通用寄存器（GPR）中。

33. 如权利要求 31 所述的数字处理器，其中这些参数中的一个参数与被释放的而不是被重新调度的线程相关。

34. 如权利要求 33 所述的数字处理器，其中与该被释放的线程相关的该一个参数为一零值。

35. 如权利要求 31 所述的数字处理器，其中这些参数中的一个参数与被重新排队等待

调度的线程相关。

36. 如权利要求 35 所述的数字处理器，其中该一个参数的值为一任意奇数值。

37. 如权利要求 35 所述的数字处理器，其中该一个参数的值为负 1 的二进制补码值。

38. 如权利要求 31 所述的数字处理器，其中这些参数中的一个参数与将执行机会让与其它线程直到一个特定条件被满足的线程相关。

39. 如权利要求 38 所述的数字处理器，其中该一个参数被编码于该记录中的位向量或是一个或多个值字段之一中。

40. 如权利要求 31 所述的数字处理器，其中这些参数中的一个参数与被释放的而不是被重新调度的线程相关，并且这些参数中的另一个参数与一个被重新排队等待调度的线程相关。

41. 如权利要求 31 所述的数字处理器，其中这些参数中的一个参数与被释放的而不是被重新调度的线程相关，并且这些参数中的另一个参数与将执行机会让与其它线程直到一个特定条件被满足相关。

42. 如权利要求 31 所述的数字处理器，其中这些参数中的一个参数与被重新排队等待重新调度的线程相关，并且这些参数中的另一个参数与将执行机会让与其它线程直到一个特定条件被满足相关。

43. 如权利要求 31 所述的数字处理器，其中这些参数中的一个参数与被释放的而不是被重新调度的线程相关，这些参数中的另一个参数与被重新排队等待调度的线程相关，并且这些参数中的再另一个参数与将执行机会让与其它线程直到一个特定条件被满足相关。

44. 在一个能够支持多个程序线程的处理器中，一种方法，包含 ：

执行一个指令，该指令存取一个数据储存装置中的一部分记录，该部份记录编码了与决定该线程是否被重新调度的一个或多个条件相关的一个或多个参数，其中该指令包含在一个程序线程中 ；以及

当在该部分记录中的该一个或多个参数等于第一数值时，响应于该指令而释放该程序线程，当该参数等于第二数值时，响应于该指令而重新调度该程序线程。

45. 如权利要求 44 所述的方法，其中该第一数值为零。

46. 如权利要求 44 所述的方法，还包括 ：

当该参数等于第三数值时，响应于该指令而挂起该程序线程的执行，其中该第三数值不等于该第一数值。

47. 如权利要求 46 所述的方法，其中该第三数值表示，执行该程序线程所需具备的条件并不满足。

48. 如权利要求 47 所述的方法，其中该条件被编码于该参数中作为位向量或值字段。

49. 如权利要求 46 所述的方法，其中该第二数值不相等于该第一数值和第三数值。

50. 如权利要求 49 所述的方法，其中该第二数值为负 1。

51. 如权利要求 49 所述的方法，其中该第二数值为一个奇数值。

52. 在一个能够支持多个程序线程的处理器中，一种方法，包括 ：

执行一个指令，该指令存取一个数据储存装置中的一部分记录，该部份记录编码了与决定该线程是否被重新调度的一个或多个条件相关的一个或多个参数，其中该指令包含于一个程序线程中 ；

当该参数等于第一数值时，响应于该指令而挂起该程序线程的执行，当该参数等于第二数值时，响应于该指令而重新调度该程序线程。

53. 如权利要求 52 所述的方法，其中该第二数值不等于该第一数值。

一种在处理器中挂起和释放执行过程中计算线程的整体机制

[0001] 本申请是申请号为200480024800.1，申请日为2004年8月26日，发明名称为“一种在处理器中挂起和释放执行过程中计算线程的整体机制”的中国发明专利申请的分案申请。

[0002] 与本发明互相参照的相关申请

[0003] 本发明要求以下申请的优先权：

[0004] (1)美国暂时申请案No.60/499,180，申请日2003年8月28日，其标题为“多线程应用的特别扩充(Multithreading Application SpecificExtension)”(代理人标号P3865，发明人凯文基赛尔(Kevin D.Kissell)，快邮编号EV 315085819US)，

[0005] (2)美国暂时申请案No.60/502,358，申请日2003年9月12日，其标题为“在一处理器架构上多线程应用的特别扩充(Multithreading Application Specific Extension to a Processor Architecture)”(代理人标号0188.02US，发明人凯文基赛尔(Kevin D.Kissell)，快邮编号ER 456368993US)，和

[0006] (3)美国暂时申请案No.60/502,359，申请日2003年9月12日，其标题为“在一处理器架构上多线程应用的特别扩充(Multithreading Application Specific Extension to a Processor Architecture)”(代理人标号0188.03US，发明人凯文基赛尔(Kevin D.Kissell)，快邮编号ER 456369013US)，其中提到的各申请案的全部内容皆为本发明所参照的参考资料。

[0007] 本发明也与申请中的美国非暂时申请案No.(尚未收到)相关，申请日2003年10月10日，其标题为“确定多线程处理器上执行的程序的服务质量的机制(Mechanisms for Assuring Quality of Service for Programs Executing on a Multithreaded Processor)”(代理人标号3865.01，发明人凯文基赛尔(Kevin D.Kissell)，快邮编号EL 988990749US)，这里提到的申请案的全部内容皆为本发明所参照的参考资料。

技术领域

[0008] 本发明属于数字处理器的领域(例如，微处理器、数字信号处理器、微控制器等等)，特别是有关于，涉及在单个处理器中管理多个线程的执行的装置与方法。

背景技术

[0009] 在数字计算的领域，计算能力的发展历史显示了在各方面都有持续的进步。持续的进步一直在发生，例如处理器的装置密度与线路互连的技术，可用于改善运算速度、容错能力、使用更高速的时脉信号或者更多其它改进。另一个可改善整体计算能力的研究领域为并行处理，其不仅包括使用多个分开的处理器执行并行操作。

[0010] 并行处理的概念包括将任务分散至多个分开的处理器，但是也包括多个程序同时在一个处理器上执行的方案。此方案一般被称为多线程。

[0011] 接下来将介绍多线程的概念：随着处理器操作频率逐渐加快，要隐藏在计算机系

统的操作中固有的延迟(latency)也变的越来越困难。一个高级处理器在一个特定应用中其高速数据缓存中丢失了百分之一的指令，如果它对于片外 RAM 有 50 个周期的延迟的话，则可能导致大概百分之五十的时间停顿。如果当该处理器因为丢失的高速缓存指令而停顿时，属于另一个不同应用程序的指令可以被执行的话，该处理器的性能可以因此而改善，并且一部份或者全部的跟内存有关的延迟也可有效的被消除。举例来说，图 1A 显示了因为高速缓存丢失而停顿的单个指令流 101。支持该指令运作的机器仅可在一个时间内执行单个线程或任务。相反的，图 1B 显示了在指令流 101 停顿时指令流 102 可被执行。在这种情况当中，该支持机器可以同时支持两个线程，也因此更有效的使用该机器所拥有的资源。

[0012] 更一般的说，各个单独的计算机指令都具有特定的语法，使得不同种类的指令需要不同的资源去执行期望的运算。整数负载没有充分使用到整个浮点运算单元的逻辑或寄存器，任何除了寄存器移位之外的运算皆需要使用加载 / 储存单元的资源。没有一个单一指令使用到全部处理器的资源，而且当为了追求更高性能的设计而因此加入了更多的管线级与并行功能单元后，会进而降低平均被所有指令使用而全部消耗的处理器资源的比例。

[0013] 多线程的发展一大部分源自于，如果一个顺序程序基本上不能有效率地完全使用处理器的全部资源，该处理器就应该能够将这些资源中的一部分在属于程序执行的多个线程当中进行分配。这种方式并不一定导致任何特定程序可被更快速的执行，事实上，一些多线程方案实质上降低了单一线程程序执行的性能，然而却可使一个并行指令流的集合在更短的时间内和 / 或使用更少的处理器数目来运行。这个概念可用图 2A 与图 2B 来说明，其中各自显示了单一线程处理器 210 与双线程处理器 250。处理器 210 支持单一线程 212，表示为使用加载 / 储存单元 214。如果当存取高速缓存 216 时发生一个丢失，那么处理器 210 便会停顿(根据图 1A 所描述)直到重新获得该丢失数据。在这个过程当中，乘法器 / 除法器单元 218 始终处于闲置状态而且没有被有效使用。然而，处理器 250 支持两个线程；即 212 与 262。因此，若是线程 212 发生停顿，处理器 250 仍然可以同时执行线程 262 与乘法器 / 除法器单元 218，因而更有效地利用了所有的资源(根据图 1B 所描述)。

[0014] 在单一处理器上具有多线程可获得更佳的多任务处理能力的好处。然而，捆绑多个程序线程在关键事件上可以降低事件反应时间，而且线程级的并行处理在原理上可在单一应用程序中被充分利用。

[0015] 已经提出了各种多线程处理方式。其中之一为指令交错多线程(interleaved multithreading)，也就是分时复用(TDM)方案，对于每个发出的指令从一个线程切换至另一个线程。该方案在调度上有一定程度的“公平性”，但是为了静态分配多个发起槽(issue slot)至多个线程，通常会限制单一程序线程的性能。动态交错的方式可以改良这个问题，但是实现这个方式比较复杂。

[0016] 另一个多线程的方案是块交错多线程(blocked multithreading)，其从一个单一程序线程持续地发出连续多个指令，直到某个特定的阻塞事件发生，例如高速缓存丢失或者重新设定，举例来说，导致该线程被挂起而另一个线程被激活。因为块交错多线程变换线程的频率较小，所以其实现方式可以是比较简单的。另一方面，阻塞的动作在调度线程的过程中是比较不具有“公平性”。单个线程可以独占整个处理器很长一段时间，如果它非常地幸运能在高速缓存中找到它需要的所有数据。一种混合调度方案结合了块交错多线程与指令交错多线程的特定，也常被使用与研究。

[0017] 仍然有另一种多线程的形式为同时多线程(simultaneous multithreading),是一种在超标量处理器上实现的方案。在同步多线程中,来自不同线程的多个指令可以被同时发出。例如,一个超标量精简指令集计算机(RISC),每周期发出总共两个指令,以及一个同步多线程的超标量管线,每周期从两个线程中任一个发出总计两个指令。那些单一程序线程所依附或停顿的周期会导致该处理器不被充分利用,因此在同步多线程中这些周期可被另一线程的发出指令所填补。

[0018] 同步多线程也因此成为一个非常有用的技术,用以解决并恢复在超标量管线中所浪费的效率。但也有争议地被认为是最复杂的多线程系统所使用的方法,因为在一周期内激活多于一个线程,会使得内存存取保护装置的实现更加地复杂等等。另外一个值得注意的一点,对于一定的工作量,中央处理器(CPU)操作的管线化越完善,越会降低其多线程实现的潜在获得的效率。

[0019] 多线程与多重处理实质上是非常相关的。事实上,一般可认为其中的差异只有一点不同:也就是多重处理器只共享内存和/或线路连线,而多线程处理器除了共享内存和/或线路连线之外,还共享指令提取与发出逻辑,和其它可能的处理器资源。在单个多线程处理器当中,各个线程互相竞争发起槽与其它资源,从而限制了并行处理能力。某些多线程程序与架构模型假设了新线程会被分配到不同的处理器,使得该程序可被有效的并行处理。

[0020] 在本申请案提出申请时,应该已有许多的多线程解决方案,用以解决当前领域许多不同的问题。其中之一即为实时线程的改善方案。一般来说,实时多媒体算法通常被执行于专用处理器/数字信号处理器(DSP),用以保证服务质量(QoS)与响应时间,且并不包括将线程混合并共享于多线程方案当中,因为实时软件并无法容易地获得保证该软件可被适时地执行。

[0021] 在这方面,非常清楚的必须要有一个方案与机制,允许一个或多个实时线程或虚拟处理器能够在特定的指令间的间隔保证获得在多线程处理器中特定比例的指令发起槽,因而计算带宽与响应时间能被清楚的定义。如果这样的机制能被使用,则具有严格 QoS 要求的线程即可被包含在此多线程的混用中。另外,这种系统中的实时线程(如 DSP 相关的线程),可以或多或少地被避免于遇到中断而因搬动重要资源而变化执行的时间。这种技术可以在特殊的情况下使用具有 DSP 加强功能的 RISC 处理器与内核,以取代一般在消费性多媒体应用上使用分离的 RISC 与 DSP 内核。

[0022] 在本申请案提出申请时,当前技术中多线程方案的另一个问题为在处理器中产出与消灭激活的线程。为了支持相对细粒度多线程(fine-grained multithreading),期望以可能的最小系统开销产生和消灭程序执行过程的并行线程,并且至少在一般的情况下不会干扰到必须的操作系统功能。在这个方面,清楚需要的是一些指令如 FORK (线程产生)与 JOIN (线程终止)。另一个存在于多线程处理器中的问题为,调度原则使得一个线程持续运行直到被另一些资源阻塞,然而一个没有被任何资源阻塞的线程仍然需要使得该处理器切换至其它的线程。所以在这个方面可以清楚的知道,还有需要 PAUSE 或 YIELD 指令。

发明内容

[0023] 本发明的一个基本目的是提供一种适用于细粒度多线程的健壮系统,其中可以利用最小的系统开销来产生和消灭线程。

[0024] 根据本发明的一方面，在一个能够支持并执行多程序线程的处理器中，提供了一种由一个线程重新调度执行或释放该线程本身的方法，其包括：(a)发出一个指令，其能存取一个数据储存装置中的一部分记录，该部分记录编码了与决定该线程是否会被重新调度的一个或多个条件相关的一个或多个参数；和(b)遵循该条件，根据在该部分记录中的一个或多个参数重新调度或释放该线程。在一个较佳实施例中，该记录被置于一个通用寄存器(GPR)中。另外，在一个较佳实施例中，这些参数中的一个参数与该被释放的而不是被重新调度的线程相关。在一些较佳实施例中，与该被释放的线程相关的该一个参数的值为零。

[0025] 在一些实施该方法的实施例当中，这些参数中的一个参数与被重新排队等待调度的线程相关。在一些实施例当中，该参数为任意的奇数值。在一些实施例当中，该参数为负1的二进制补码值。在一些实施例中，这些参数中的一个参数与将执行机会让与其他线程直到一个特定条件被满足的实施例相关。另外，在另一些实施例中，该一个条件被编码于该记录中的一个位向量或是一个或多个位字段中。

[0026] 在许多实施该方法的实施例当中，在该线程发出该指令并且被重新调度的情形下，当该一个或多个条件被满足时，该线程的执行在线程指令流中该线程发出的该指令之后的位置继续。在一些实施例当中，这些参数中的一个参数与被释放而不是被重新调度的线程相关，并且这些参数中的另一个参数与被重新排队等待调度的线程相关。在其它的一些实施例当中，这些参数中的一个参数与该被释放而不是被重新调度的线程相关，并且这些参数中的另一个参数与将执行机会让与其它线程直到一个特定条件被满足的线程相关。在其它的一些实施例当中，这些参数中的一个参数与被重新排队等待重新调度的线程相关，并且这些参数中的另一个参数与将执行机会让与其它线程直到一个特定的条件被满足的线程相关。另外，在其它的一些实施例当中，这些参数中的一个参数与该被释放而不是被重新调度的线程相关，而这些参数中的另一个参数与被重新排队等待调度的线程相关，并且这些参数中的再另一个参数与将执行机会让与其它线程直到一个特定的条件被满足的线程相关。

[0027] 根据本发明的另一个方面，提供了一种能支持与执行多个软件实体的数字处理器，包括一个数据储存装置中的一部分记录，该部分记录编码了与一个或多个条件相关的一个或多个参数，该一个或多个条件决定了当一个线程将执行机会让与其它线程时该线程是否会被重新调度或释放。

[0028] 在该处理器的一些实施例当中，该部分记录被置于一个通用寄存器(GPR)中。在一些其它较佳实施例，这些参数中的一个参数与被释放而不是被重新调度的线程相关。在另一些较佳实施例中，该与被释放的线程相关的参数的值为零。

[0029] 在该处理器的其他一些实施例当中，这些参数中的一个参数与被重新排队等待调度的线程相关。在一些实施例当中，该一个参数为任意的奇数值。在另一些实施例当中，该参数为负1的二进制补码值。在另一些实施例中，这些参数中的一个参数与将执行机会让与其它线程直到一个特定条件被满足的线程相关。另外，在一些情况下，该一个参数可以被编码于该记录中的一个位向量或是一个或多个位字段中。

[0030] 在该处理器的其他一些实施例当中，这些参数中的一个参数与该被释放而不是被重新调度的线程相关，并且这些参数中的另一个参数与被重新排队等待调度的线程相关。在其它的一些实施例当中，这些参数中的一个参数与被释放而不是被重新调度的线程相

关，并且这些参数中的另一个参数与将执行机会让与其它线程直到一个特定条件被满足的线程相关。在其它的一些实施例当中，该参数当中的一个参数与被重新排队等待重新调度的线程相关，并且这些参数中的另一个参数与将执行机会让与其它线程直到一个特定条件被满足的线程相关。

[0031] 在其它的一些实施例当中，这些参数中的一个参数与被释放而不是被重新调度的线程相关，而该参数当中的另一个参数与被重新排队等待调度的线程相关，并且这些参数中的再另一个参数与将执行机会让与其它线程直到一个特定条件被满足的线程相关。

[0032] 根据本发明的另一方面，提供在一个能够支持与执行多程序线程的处理器中，一种由一个线程重新调度执行或释放该线程本身的设备，包括：(a)用于发出一个指令的装置，该指令存取一个数据储存装置中的一部分记录，该部份记录编码了与决定该线程是否被重新调度的一个或多个条件相关的一个或多个参数；以及(b)用于根据在该部分记录中的该一个或多个参数来依照该条件重新调度该线程或释放该线程的装置。

[0033] 在该处理系统的一些较佳实施例当中，该记录被置于一个通用寄存器(GPR)中。在一些其它的较佳实施例，这些参数中的一个参数与被释放而不是被重新调度的线程相关。在一些实施例中，与被释放的线程相关的该一个参数的值为零。在一些其它的实施例当中，这些参数中的一个参数与被重新排队等待调度的线程相关。在一些实施例当中，该用来重新调度的参数为任意的奇数值。在其它的一些实施例当中，该用来重新调度的参数为负 1 的二进制补码值。

[0034] 在该系统的一些实施例当中，这些参数中的一个参数与将执行机会让与其它线程直到一个特定条件被满足的线程相关。另外，在一些实施例中，该一个参数被编码于该记录中的一个位向量或是一个或多个位字段中。在该系统的许多实施例当中，在一个线程发出该指令并且被有条件地重新调度的情形下，当该一个或多个条件被满足时，该线程的执行在该线程指令流中该指令之后的位置继续。

[0035] 在该处理系统的一些实施例当中，该参数当中的一个参数与被释放而不是被重新调度的线程相关，并且这些参数中的另一个参数与被重新排队等待调度的线程相关。在其它的一些实施例当中，该参数当中的一个参数与被释放而不是被重新调度的线程相关，并且这些参数中的另一个参数与将执行机会让与其它线程直到一个特定条件被满足的线程相关。

[0036] 在其它的一些实施例当中，该参数当中的一个参数与被重新排队等待重新调度的线程相关，并且这些参数中的另一个参数与将执行机会让与其它线程直到一个特定条件被满足相关。另外，在其它的一些实施例当中，该参数当中的一个参数与被释放而不是被重新调度的线程相关，而该参数当中的另一个参数与被重新排队等待调度的线程相关，并且这些参数中的再另一个参数与将执行机会让与其它线程直到一个特定条件被满足的线程相关。

[0037] 在该方法的一些实施例当中，该指令为一个 YIELD 指令。还有，在该处理系统的一些实施例当中，该指令为一个 YIELD 指令。

[0038] 根据本发明的另一方面，在一个能够支持多程序线程的处理器当中，提供了一种方法，包括：执行一个指令，该指令存取一个数据储存装置中的一部分记录，该部份记录编码了与决定该线程是否被重新调度的一个或多个条件相关的一个或多个参数，其中该指令

包含在一个程序线程中；当在该部分记录中的该一个或多个参数等于第一数值时，则响应于该指令释放该程序线程；当该参数等于第二数值时，响应于该指令而重新调度该程序线程。

[0039] 在该方法的一些实施例当中，该第一数值为零。在该方法的另一些实施例当中，该方法还包括一个步骤：当该参数等于第三数值时，根据该指令挂起该程序线程的执行，其中该第三数值不等于该第一数值。在该方法的一些实施例当中，该第三数值表示，执行该程序线程所需要具备的条件并不满足。

[0040] 在该方法的一些其它实施例当中，该条件以一个位向量或值字段的形式被编码在该参数之中。在一些其它的实施例当中，该第二数值不等于该第一数值与第三数值。在其他一些实施例当中，该第二数值为负 1。在另外一些实施例当中，该第二数值为奇数值。

[0041] 根据本发明的另一个方面，在一个能够支持多程序线程的处理器当中，提供了一个方法，包括：执行一条指令，该指令存取一个数据储存装置中的一部分记录，该部份记录编码了与决定该线程是否被重新调度的一个或多个条件相关的一个或多个参数，其中该指令包含于一个程序线程中；当该参数等于第一数值时，则根据该指令挂起该程序线程的执行。在该方法的一些其它实施例当中，该方法还包含一个步骤：当该参数等于第二数值时，根据该指令而重新调度该程序线程，其中该第二数值不等于该第一数值。

[0042] 本发明的实施例将在下文中更加详细的描述，在这些实施例中将第一次提供一种用于细粒度多线程的真正强壮的系统，使得产生与消灭线程所用的系统开销最小化。

附图说明

[0043] 图 1A 是一个示意图，显示单一指令流由于高速缓存丢失而停顿的情况；

[0044] 图 1B 是一个示意图，显示当如图 1A 的指令流被停顿时一个指令流仍然能被执行；

[0045] 图 2A 是一个示意图，显示单一线程处理器；

[0046] 图 2B 是一个示意图，显示双线程处理器 250；

[0047] 图 3 是一个示意图，描述了根据本发明的一个实施例中，一个处理器支持第一与第二 VPE；

[0048] 图 4 是一个示意图，描述了根据本发明的一个实施例中，一个处理器能够支持单一 VPE，该 VPE 能进一步支持三个线程；

[0049] 图 5 显示了根据本发明的一个实施例中，一个 FORK 指令的格式；

[0050] 图 6 显示了根据本发明的一个实施例中，一 YIELD 指令的格式；

[0051] 图 7 是一个表格，显示了一个用于 GPR rs 的十六位的限定掩码；

[0052] 图 8 显示了根据本发明的一个实施例中，一 MFTR 指令的格式；

[0053] 图 9 是一个表格，说明了根据本发明的一个实施例中，一个 MFTR 指令的字段；

[0054] 图 10 显示了根据本发明的一个实施例中，一 MTTR 指令的格式；

[0055] 图 11 是一个表格，说明了根据本发明的一个实施例中，一个 MTTR 指令 u 和 sel 位；

[0056] 图 12 显示了根据本发明的一个实施例中，一个 EMT 指令的格式；

[0057] 图 13 显示了根据本发明的一个实施例中，一 DMT 指令的格式；

[0058] 图 14 显示了根据本发明的一个实施例中，一个 ECONF 指令的格式；

[0059] 图 15 是根据本发明的一个实施例中，一个系统协处理器特权资源的说明表格；

[0060] 图 16 是根据本发明的一个实施例中，一个 ThreadControl 寄存器的架构；

[0061] 图 17 是根据本发明的一个实施例中，一个 ThreadControl 寄存器架构中各字段的说明表格；

[0062] 图 18 是根据本发明的一个实施例中，一个 ThreadStatus 寄存器的架构；

[0063] 图 19 是根据本发明的一个实施例中，一个 ThreadStatus 寄存器架构中各字段的说明表格；

[0064] 图 20 是根据本发明的一个实施例中，一个 ThreadContext 寄存器的架构；

[0065] 图 21 是根据本发明的一个实施例中，一个 ThreadConfig 寄存器的架构；

[0066] 图 22 是根据本发明的一个实施例中，一个 ThreadConfig 寄存器架构中各字段的说明表格；

[0067] 图 23 是根据本发明的一个实施例中，一个 ThreadSchedule 寄存器的架构；

[0068] 图 24 是根据本发明的一个实施例中，一个 VPESchedule 寄存器的架构；

[0069] 图 25 是根据本发明的一个实施例中，一个 Config4 寄存器的架构；

[0070] 图 26 是根据本发明的一个实施例中，一个 Config4 寄存器架构中各字段的说明表格；

[0071] 图 27 是一个表格，定义了线程异常所需的 Cause 寄存器的异常代码值；

[0072] 图 28 是一个表格，定义了 ITC 指示符；

[0073] 图 29 是一个表格，定义了 Config3 寄存器架构中的各字段；

[0074] 图 30 是一个表格，描述了每个 VPE 上下文的 VPE 禁止位；

[0075] 图 31 是一个表格，描述了 ITC 储存的运作方式；

[0076] 图 32 是一个示意图，描述了根据本发明的一个实施例中的 YIELD 功能的操作；

[0077] 图 33 是一个示意图，描述了根据本发明的一个实施例的一个计算机操作系统；

[0078] 图 34 是一个示意图，描述了根据本发明的一个实施例中，在一个处理器中使用 VPE 和在一个 VPE 中使用线程来实施调度。

具体实施方式

[0079] 根据本发明的一个较佳实施例，一处理器架构包括一指令集，而该指令集包含多个特征、多个功能与多个指令，而能够在一兼容处理器上产生多线程的运算。本发明并不止限于任何特定的处理器架构与指令集，而是可以大致归类为众所皆知而参照的 MIPS 架构，指令集与处理器技术（总言之，为 MIPS 技术）。并且加上本发明所详细描述的实施例也可归类为 MIPS 技术。更多有关 MIPS 技术的信息（包括以下所参照到的文件）可以从 MIPS 科技公司（MIPS ttechnology, Inc.）（位于 Mountain View, California）和其网站 www.mips.com（该公司网站）获得。

[0080] 所提到的“处理器”与“数字处理器”其意义包含任何可程序化的装置（举例来说，微处理器、微控制器、数字信号处理器、中央处理单元、处理器内核等等），包含在硬件方面（例如，专用硅芯片、现场可程序门阵列（FPGA）等等），在软件方面（例如，硬件描述语言、C 语言、C ＋语言等等）或任何其组成（或其组合）。

[0081] 术语“线程”与“程序线程”在本文中代表相同的意义。

[0082] 概要描述

[0083] 在本发明的实施例中，“线程上下文”是一处理器状态的集合，用于描述在一处理器上的一指令流执行的状态。所说的状态通常反映在处理器寄存器的内容中。举例来说，在一个与工业规格 MIPS32 和 / 或 MIPS64 指令集架构兼容的处理器（MIPS 处理器）中，线程上下文为由通用寄存器（GPRs），高低（Hi/Lo）乘法结果寄存器，有程序计数器（PC）功能和一些相关的特权系统控制状态的寄存器组成。系统控制状态在 MIPS 处理器中通常称作第零协处理器（CP0）的部分保留，并且一大部分是被系统控制寄存器与翻译后援缓存器（Translation Lookaside Buffer，TLB）所保存（如果使用了 TLB）。相反的，“处理器上下文”是一个更大的处理器状态集合，包含至少一个线程上下文。再参照之前提到的 MIPS 处理器为例，一个处理器上下文包含至少一个线程上下文（如前述），也就是 CP0 和必须的系统状态，用以描述已知的 MIPS32 与 MIPS64 专属资源或特权资源架构（PRA）。（简单的说，PRA 是一组有关一个指令集架构操作时所依据的环境与能力参数的集合。该 PRA 提供了操作系统所必须的机制用以管理处理器的资源，例如，虚拟内存、高速缓存、异常运算与使用者上下文）。

[0084] 根据本发明的一个实施例，关于一指令集架构与 PRA 的一个特殊应用扩充的多线程（multithreading application-specific extensions，Multithreading ASE）允许在一处理器中包含两个不同，但并不互相排斥的多线程性能。首先，一个单一处理器可以有一定数目的处理器上下文，而其中每一个都透过共享某些处理器的资源且支持一个指令集架构而作为独立的处理单元操作。这些独立的处理单元在这里被称为虚拟处理单元（VPE）。对软件而言，具有 N 个 VPE 的处理器被看成是有 N 个路径且对称的多处理器（SMP）。这允许已有的具 SMP 功能的操作系统可以管理 VPE 集合，也就是透明地共享处理器的执行单元。

[0085] 图 3 用一个单一处理器 301 描述了相关的性能，其支持了一个第一 VPE（VPE0），VPE0 包含第零寄存器状态 302 与第零系统协处理器状态 304。处理器 301 也支持第二 VPE（VPE1），其包含第一寄存器状态 306 与第一系统协处理器状态 308。VPE0 与 VPE1 共享处理器 301 的这些部分，包括取指、译码、管线化执行和高速缓存 310。与 SMP 兼容的操作系统 320 被执行在该处理器 301 上，并支持 VPE0 与 VPE1。如图所示，软件进程 A 322 与进程 C 326 分别被执行于 VPE0 与 VPE1 上，如同他们被执行于两个不同的处理器上。进程 B 324 处于队列状态，而可以在 VPE0 或 VPE1 的任一个上执行。

[0086] 多线程 ASE 所允许的第二个能力为，每个处理器或 VPE 皆可以在基本架构中所需的单一线程上下文之外，再含有某些数目的线程上下文。多线程 VPEs 需要特别的操作系统支持，并且在其支持下提供一个简易、细粒度多线程程序模型，其中线程可以被产生与消灭，使得在一般的情况下不会干扰操作系统，并且系统服务线程可以响应外部条件（例如，事件等等）安排调度，而没有中断的延迟。

[0087] 图 4 描述了这第二个能力且使用了处理器 401 来支持单一 VPE，其包含寄存器状态 402，404 与 406（支持三个线程 422）以及系统协处理器状态 408。与图 3 不同的是，在这个例子中三个线程是在单一应用地址空间中，且在单一 VPE 上共享 CP0 资源（以及硬件资源）。另外也描述了一个专门多线程操作系统 420。在此范例中，多线程 VPE 正在处理来自一个宽带网络 450 的数据包，而此数据包的下载分布于整组的先进先出缓冲器（FIFO）452（在多线

程 VPE 的输入 / 输出内存空间中每个 FIFO 皆有不同的地址)。控制应用程序产生了足够多的线程,与使用的 FIFO 数目相同,且将每一线程应用于在读取 FIFO 的紧凑循环中。

[0088] 一个线程上下文可以是四种状态之一。其可以是空闲(free),激活(activated),停止(halted)或连线(wired)。一个空闲的线程上下文不具有有效的内容,且不能被调度为可发出指令。一个激活的线程上下文可以根据实施的规则来调度,而从程序计数器提取与发出指令。一个停止的线程上下文可以具有有效的内容,但是不能够提取与发出指令。一个连线的线程上下文可以被指定作为映像寄存器使用,也就是说其被保留成专用于异常处理程序,以避免在该异常处理程序中储存与恢复存储器上下文所产生的开销。一个空闲的线程上下文不可以是激活,停止或连线的。只有激活的线程上下文可以被调度。只有空闲的线程上下文可以被分配而产生新的线程。

[0089] 为了允许协同线程的细粒度同步,一个供线程内部之间沟通(ITC)的记忆空间在虚拟内存中被产生,并且有空(empty)/ 满(full)位语法用来允许线程在加载或储存时被阻塞,直到数据被其它线程产生或消耗。

[0090] 线程产生 / 消灭与同步特性在一般情况下不会干预操作系统,但是他们所操控的资源可通过操作系统将之虚拟化。这允许了多线程程序可以利用更多的虚拟线程执行,其数目多于在一个 VPE 上的线程上下文数目,而使得线程的迁移能平衡多处理器系统的负荷。

[0091] 再仔细的从执行过程的某个点来看,一个线程与一个特定 VPE 上的一个特定线程上下文捆绑在一起。VPE 的线程上下文组合的索引在该点的发生时间提供了一个唯一的标识符。但是上下文的切换与迁移能够使单一相继的线程的执行具有一连串不同的线程索引,例如是在一连串不同的 VPE 上。

[0092] 在一个特定的处理器重置调整状态中执行线程上下文、TLB 项目和其它资源与同一处理器上多个 VPE 的动态绑定。每一个 VPE 输入其重置向量,如同它就是一个独立的处理器。

[0093] 多线程的执行和异常模型

[0094] 多线程 ASE 并没有强加任何特殊的实现方式或调度模型用于并行线程与 VPE 的执行。调度方式可以是循环式的,任意粒度的时间切割或同时的。然而没有一个实现方式允许一个阻塞的线程独占任何共享的处理器资源,然后使得硬件运行陷入死锁。

[0095] 在一 MIPS 处理器中,多个线程执行在一个单一 VPE 上,并皆共享一样的系统协处理器(CP0),一样的 TLB 和一样的虚拟地址空间。每一个线程有一个独立的内核 / 监督者 / 使用者状态,用于内存访问与指令译码。当一个异常发生时,除了执行该异常的线程之外,所有的线程都被停止或挂起,直到状态字符串的 EXL 与 ERL 位被清除。或者,在 EJTAG 调试异常的情形下,退出该调试状态。该状态字符串被置于 CP0 中的状态寄存器中。有关该 EXL 与 ERL 位还有 EJTAG 调试异常的详细资料可以从下列的两个出版物获得,可以从 MIPS 科技公司取得该出版物,并且其全部内容在各种情形下可列为本文的参考文件 :MIPS32™ Architecture for Programmers Volumn III:The MIPS32™ Privileged Resource Architecture,Rev.2.00,MIPS 科技公司(2003) 和 MIPS64™ Architecture for Programmers Volumn III :The MIPS64™Privileged Resource Architecture,Rev.2.00,MIPS 科技公司(2003)。

[0096] 因为执行一个指令流而引起的同步异常的异常处理程序，例如 TLB 的丢失与浮点的异常，都由用于执行该指令流的该线程来执行。当一个未被屏蔽的异步异常，例如一个中断，被提升至一个 VPE 时，它的实现与执行了该异常处理程序的那个线程相关。

[0097] 甚至当使用影像寄存器组来执行例外处理程序时，每一个异常都与一个线程上下文有关。这个相关的线程上下文是被异常处理程序执行的 RDPGPR 与 WRPGPR 指令所要处理的目标。有关 RDPGPR 与 WRPGPR 指令（用来访问影像寄存器）的详细描述可以从以下两个出版物中获得，可以从 MIPS 科技公司取得该出版物，并且其全部内容在各种情形下可列为本文的参考文件：MIPS32™ Architecture for Programmers Volumn III:The MIPS32™ Instruction Set,Rev.2.00,MIPS 科技公司（2003）和 MIPS64™Architecture for Programmers Volumn III:The MIPS64™ Instruction Set,Rev.2.00,MIPS 科技公司（2003）。

[0098] 该多线程 ASE 包含了两个异常状况。第一个是线程未取得的状况，其中一个线程分配要求不能被满足。第二个为线程下溢状况，其中一个线程的终止与释放使得没有线程被分配于一个 VPE 上。这两种异常状况都被映射至一个新的单一线程异常。当该异常发生时，他们可以根据 CP0 寄存器的位设置而被区分。

[0099] 指令

[0100] 在一个较佳的实施例中，多线程 ASE 包含七个指令。FORK 与 YIELD 指令控制线程分配，释放和调度，并且可以在被执行和使能的全部执行模式中取得。MFTR 与 MTTR 指令是系统协处理器（Cop0）指令，可用于特权系统软件来管理线程状态。一个新的 EMT 指令与一个新的 DMT 指令是特权的 Cop0 指令，其用来激活与禁能一个 VPE 的多线程操作。最后，一个新的 ECONF 指令是特权的 Cop0 指令，用于退出一个特殊处理器配置状态并重新初始化该处理器。

[0101] FORK- 分配与调度一个新的线程

[0102] FORK 指令可以驱使一个空闲线程上下文被分配与激活。它的格式 500 如图 5 所示。FORK 指令从由字段 502（rs）与 504（rt）标识的 GPR（通用寄存器）取得两个操作数值。GPR rs 的内容是用来对新线程开始提取与执行的地址。GPR rt 的内容是一个值，用来传送至新线程的 GPR。目的 GPR 由 CP0 的 ThreadConfig 寄存器的 ForkTarget 字段的值确定，其说明位于图 21 中，并于稍后会加以描述。新线程的内核 / 监督者 / 使用者状态被设定于 FORK 处理的线程中。如果没有空闲线程上下文给该 FORK 指令使用，产生关于该 FORK 指令的一个线程异常。

[0103] YIELD- 重调度与有条件的释放一个线程

[0104] YIELD 指令使得当前的线程被重新调度。它的格式 600 如图 6 所示，并且图 32 中的流程图 3200 描述了根据本发明的一个实施例的系统操作，来说明 YIELD 指令的功能。

[0105] YIELD 指令例如从字段 602（rs）中指定的 GPR 得到一个单一操作数值。在一较佳实施例中使用了一个 GPR，但是在其它的实施例中，该操作数值可以在实质上任何可由系统访问的数据储存装置（例如，非 GPR 寄存器，内存等等）中被储存或取得。在一实施例中，GPR rs 的内容可以被视为是一个描述符，描述了一个发出的线程应该被重新调度的情况。如果该 GPR rs 的内容是零（即该操作数的值为零），如图 32 的步骤 3202 所示，则该线程并不会被重调度，而是会如同步骤 3204 所示被释放（即，终止或永久的停止更进一步的执行），

并且与其相关的线程上下文的储存器(即上述提到的用于保存状态的寄存器)变成空闲,从而可以被其它线程发出的接下来的FORK指令进行分配。如果该GPR rs的最低有效位被设定(即,rs0=1),则如图32的步骤3206所示该线程马上被重新调度,并且如果没有其它可执行的线程抢先的话,就继续执行该线程。在该实施例中,该GPR rs的内容被视为15位的限定符掩码,如图7中表格700的描述(即,用于编码各种条件的位向量)。

[0106] 请参照表格700,寄存器rs的位15至位10表示提供给处理器的硬件中断信号,位9至位8表示处理器中产生的软件中断信号,位7至位6表示MIPS架构中最基本地关联加载(Load Linked)和条件存储(Store Conditional)同步原语的操作,还有位5至位2表示提供给处理器的外部非中断信号。

[0107] 如果GPR rs的内容值是一偶数(即,位0未被设定),并且GPR rs的限定符掩码中的任何其他位皆被设定(步骤3208),则该线程被挂起,直到满足至少一个对应条件。如果当此情形发生,该线程就被重新调度(步骤3210),并且从YIELD之后的指令重新开始执行。这个过程的使能并不会受CP0.Status.iMn中断屏蔽位的影响,因此总共有被位15至10和位5至2(如图7所示)编码的十个外部条件(例如,事件等等)以及被位9至6(如图7所示)编码的四个软件条件,在目前的实施例中被用来响应外部信号以使能独立的线程,而不需要处理器执行异常处理。在这个特定例子中,有六个硬件中断和四个非中断信号,再加上两个软件中断和两个非中断信号,最后再加上一个专用于重调度功能的信号(即rs0),总共对应于十五个条件。(该CP0.Status.iMn中断屏蔽位是在CP0 Status寄存器中的一个八位的集合,其可选择性地屏蔽MIPS处理器的八个基本的中断输入。如果一个IM位被设定,则其它相关的中断输入就不会引起处理器的异常事件。)

[0108] 在EIC的中断模式中,位IP2至IP7对有最高优先权的中断进行编码,而不只是表示了一个有正交指示的向量。当处理器使用EIC的中断模式时,在一个YIELD指令中与位IP2至IP7相关联的GPR rs的位因此不再能够被用于针对一个特定外部事件去重新使能一个线程。在EIC的中断模式中,只有与系统相关连的外部事件指示符(例如,在本实施例中,是GPR rs的位5至2)可以被用作YIELD的限定符。EIC的中断模式与位IP2至IP7已被更进一步地描述于下列的出版物,上文中已经指出并引用了该出版物的整个内容:MIPS32™ Architecture for Programmers Volumn III:The MIPS32™ Privileged Resource Architecture,与MIPS64™ Architecture for Programmers Volumn III:The MIPS64™ Privileged Resource Architecture。

[0109] 如果执行YIELD的结果是对于处理器或VPE上最近分配的线程的释放,则关于该YIELD指令产生一个线程异常,具有在CP0的ThreadStatus寄存器中的下溢指示(如图18所示且在稍后会加以说明)。

[0110] 上述实施例使用了YIELD指令的GPR rs中所包含的操作数作为线程调度的参数。在该例子中,这个参数被看作一个15位的正交指示的向量(参照图7,位1与15被保留,所以在此较佳实施例中只有十五个条件被编码)。此实施例也将此参数当成是一个指定的值(即,用于决定是否一个给定的线程应该被释放,参考图32的步骤3202)。然而,这样一个参数的特性可以被改变,以适合各种不同指令的实施例。例如,不是依靠最低有效位(即rs0)来决定是否一个线程可被立即重新调度,而是使用该参数本身的值(例如,二进制补码形式的负一{-1})来决定一个线程是不是应该被立即重新调度(即,用于调度的重新排队)。

[0111] 在该指令的其他实施例中，可以将这样的线程调度参数看作包含一个或多个多位值的字段，以使得一个线程可以确定，它将关于一个大的时间名空间(例如，32 位或更大)中的一个单一事件而产生。在这样的实施例中，至少与该目标事件有关的位可以被当前 YIELD 指令访问。当然，如一特定实施例所期望的，更多的位字段可被传送至该指令(与更多的事件相关联)。

[0112] 该 YIELD 指令的其他实施例，在由该指令访问的一个线程调度参数中可以包含前述位向量与值字段的组合，或其它具体应用的改进和提高，(例如)以满足特定实现的需要。YIELD 指令的可选实施例可以用任何已知方法访问如前所述的一个线程调度的参数，例如，从一个 GPR (如图 6 所示)，从任意其它的数据存储装置(包含内存)以及作为该指令本身中的立即值。

[0113] MFTR- 从线程寄存器移动

[0114] MFTR 指令是一个特权(Cop0)指令，可允许一个操作系统执行一个线程来访问另一个不同的线程上下文。其格式 800 被描述于图 8。

[0115] 要被访问的线程上下文由 CP0 的 ThreadControl 寄存器的 AlternateThread 字段的值确定，该字段如图 16 所示并且会在稍后描述。在选定的线程上下文中，要被读取的寄存器由字段 802 标定的 rt 操作数寄存器的值，和分别位于该 MFTR 指令的字段 804 与 806 中的 u 与 sel 位来确定，并且依据图 9 所示的表格 900 进行说明。产生的值被写入由字段 808 标定的目的寄存器 rd。

[0116] MTTR- 向线程寄存器移动

[0117] MTTR 指令与 MFTR 指令相反。它是一个特权 Cop0 指令，其将当前线程的线程上下文中的寄存器值复制到另一个线程上下文的寄存器中。其格式 1000 如图 10 所示。

[0118] 要被访问的线程上下文由 CP0 的 ThreadControl 寄存器的 AlternateThread 字段的值确定，该字段如图 16 所示并且会在稍后描述。在选定的线程上下文中，要被写入的寄存器由字段 1002 标定的 rd 操作数寄存器中的值，结合分别在该 MTTR 指令的字段 1004 与 1006 中提供的 u 与 sel 位来确定，并且依据在图 11 中所显示的表格 1100 进行解释(其编码相似于 MFTR)。由字段 1008 标定的寄存器 rt 中的值被复制到选定的寄存器。

[0119] EMT- 使能多线程

[0120] EMT 指令为特权 Cop0 指令，其通过设定 CP0 的 ThreadControl 寄存器的 TE 位来使能多个线程的并行执行，该寄存器如图 16 所示并且会在稍后描述。该指令的格式 1200 显示于图 12。包含在该 EMT 执行之前的 TE(Thread Enabled)位值的该 ThreadControl 寄存器的值会被传回寄存器 rt。

[0121] DMT- 禁能多线程

[0122] DMT 指令为特权 Cop0 指令，其通过清除 CP0 的 ThreadControl 寄存器的 TE 位来禁止多线程的并行执行，该寄存器如图 16 所示并且会在稍后描述。该指令的格式 1300 显示于图 13。

[0123] 除了发出该 DMT 指令的线程之外，所有的线程都被禁止进一步的指令提取与执行。这与所有线程暂停状态是无关的。包含该 DMT 执行之前的 TE (Thread Enabled)位值的该 ThreadControl 寄存器的值会被传回寄存器 rt。

[0124] ECONF- 结束处理器的配置

[0125] ECONF 指令为一特权 Cop0 指令，其通知 VPE 配置的结束，并使能多 VPE 的执行。该指令的格式 1400 显示于图 14。

[0126] 当一个 ECONF 指令被执行时，Config3 寄存器的 VPC 位(稍后描述)即被清除，而该寄存器的 MVP 位的当前值也变成只读，并且处理器的所有 VPE，包含正在执行 ECNOF 的这个 VPE，都产生一个 Reset 异常。

[0127] 特权资源(Privileged Resource)

[0128] 图 15 的表格 1500 列出了系统协处理器的与多线程 ASE 相关的特权资源。除了特别的说明之外，不管是新的还是修改过的第零协处理器(CP0)的如下所述的寄存器都是可访问的(即，写入和读出)，就像传统的第零协处理器(即，MIPS 处理器)的系统控制寄存器一样。

[0129] 新的特权资源

[0130] (A) ThreadControl 寄存器(CP0 寄存器号码 7，选择号码 1)

[0131] 该 ThreadControl 寄存器是在每个 VPE 中作为系统协处理器的一个部分。其结构 1600 显示于图 16。该 ThreadControl 寄存器的字段可根据图 17 的表格 1700 来设定。

[0132] (B) ThreadStatus 寄存器(CP0 寄存器号码 12，选择号码 4)

[0133] 该 ThreadStatus 寄存器存在于每个线程上下文中。每一个线程皆有其 ThreadStatus 的拷贝，并且特权程序代码可以通过 MFTR 与 MTTR 指令访问其它线程的 ThreadStatus。其结构 1800 显示于图 18。该 ThreadStatus 寄存器的字段可根据图 19 的表格 1900 来设定。

[0134] 在一个激活线程的 Halted 位写入一个 1，会使得该激活线程停止提取指令，并且将内部重启动程序计数器(PC)设定到下一个发出的指令。在一个激活线程的 Halted 位写入一个 0，使得被调度的该线程从内部重启动程序计数器(PC)地址提取和执行指令。只要在未被激活的线程的 Activated 位或是 Halted 位中有任一为 1，则该线程就可避免被一个 FORK 指令分配和激活。

[0135] (C) ThreadContext 寄存器(CP0 寄存器号码 4，选择号码 1)

[0136] 该 ThreadContext 寄存器 2000 存在于每个线程上下文中，并且如图 20 所示其长度与处理器的 GPR 是相同的。这纯粹是一个软件可读 / 写的寄存器，可被操作系统用作特定线程储存的指针，例如一个线程上下文保存的区域。

[0137] (D) ThreadConfig 寄存器(CP0 寄存器号码 6，选择号码 1)

[0138] 该 ThreadConfig 寄存器存在于每个处理器或 VPE 中。其结构 2100 显示于图 21 中。该 ThreadConfig 寄存器的字段被定义在图 22 的表格 2200 中。

[0139] ThreadConfig 寄存器的 WiredThread 字段允许在一个 VPE 上可获得的线程上下文的集合在影像寄存器集合与并行执行线程之间被分割。线程上下文的索引若是小于该 WireThread 的值，则可从影像寄存器获得该线程上下文。

[0140] (E) ThreadSchedule 寄存器(CP0 寄存器号码 6，选择号码 2)

[0141] ThreadSchedule 寄存器是选择性的，但是当被实现时，最好被实现于每个线程中。其结构 2300 显示于图 23 中。

[0142] 调度向量(Schedule Vector)(如图所示，在一较佳实施例中其宽度为 32 位)为对于相关线程进行调度所要求的发出带宽的描述。在此实施例中，每一位皆代表该处理器或

VPE 的发出带宽的 1/32，并且每一位的位置代表了在有 32 个时段的调度循环中的一个明确的时段。

[0143] 如果在一个线程的 ThreadSchedule 寄存器中设定了一位，那么该线程即被保证了在相关的处理器或 VPE 上每 32 个可能的连续发出中，可获得一个对应的发出时段。在 ThreadSchedule 寄存器的一位上写入一个 1，当相同处理器或 VPE 上的其它线程已经具有一样的 ThreadSchedule 位设定时，则将产生线程异常。虽然在这里，ThreadSchedule 寄存器的优选宽度是 32 位，但是可以预料的到，在其它的实施例中该宽度可以改变(即，增加或减少)。

[0144] (F)VPESchedule 寄存器(CP0 寄存器号码 6，选择号码 3)

[0145] VPESchedule 寄存器是可选择的，并且优选的存在于每个 VPE 中。它只有当 Config3 寄存器的 MVP 位被设定时才能被写入(请参考图 29)。其格式 2400 显示于图 24。

[0146] 调度向量(如图所示，在一较佳实施例中其宽度为 32 位)为对于相关 VPE 进行调度所要求的发出带宽的描述。在此实施例中，每一位皆代表一个多 VPE 处理器的发出总带宽的 1/32，并且每一位的位置代表了在有 32 个时段的调度循环中的一个明确的时段。

[0147] 如果 VPE 的 VPESchedule 寄存器中的一个位被设定，那么该线程即被保证了在相关处理器上每 32 个可能的连续发出中，可获得一个对应的发出时段。在 VPE 的 VPESchedule 寄存器的一位上写入一个 1，当其它 VPE 已经具有一样的 VPESchedule 位设定时，则将产生线程异常。

[0148] 依据处理器目前默认的线程调度原则(例如，循环法等等)，只要发出时段未被任何线程特别的排定，其仍然可被自由的分配给任何可执行的 VPE/ 线程。

[0149] VPESchedule 寄存器与 ThreadSchedule 寄存器创造了一个发出带宽分配的结构。VPESchedule 寄存器的设定指定了对于 VPE 的带宽，其为在一个处理器或内核上全部可取得带宽的一定比例，而 ThreadSchedule 寄存器指定了对于线程的带宽，其为在一个包含线程的 VPE 中可取得全部带宽的一定比例。

[0150] 虽然在这里，VPESchedule 寄存器的优选宽度是 32 位，但是可以预期，在其它的实施例中该宽度可以改变(即，增加或减小)。

[0151] (G)Config4 寄存器(CP0 寄存器号码 16，选择号码 4)

[0152] 寄存器 Config4 存在于每一个处理器中。其包含了对于动态多 VPE 处理器配置所必须配置信息。如果处理器并不处于 VPE 配置状态(即，Config3 寄存器的 VMC 位被设定)，则除了 M (连续)字段之外的所有字段的值都会成为与实施方式有关且是不可预测其结果的。其结构 2500 描述于图 25。Config4 寄存器的字段的定义如图 26 的表格 2600 中所示。在某些实现方式或实施例中，Config3 寄存器的 VMC 位可以是一个被事先保留 / 不指定的位。

[0153] 对于目前存在的特权资源架构的修改

[0154] 该多线程 ASE 对于当前 MIPS32 与 MIPS64PRA 的一些单元做了变更。

[0155] (A)Status 寄存器

[0156] Status 寄存器中的 CU 位对于多线程配置具有一些额外的意义。设定 CU 位的动作也就是要求将一个协处理器上下文与和该 CU 位关联的线程绑定。如果一个协处理器上下文是可用的，则将它与该线程绑定在一起，以使该线程所发出的指令能传送到该协处理器，

并且该CU位会保留写入该位的1。如果没有一个协处理器上下文是可用的，则该CU位便会读回0。写入一个0去设定该CU位，会使得任何相关联的协处理器被释放。

[0157] (B)Cause寄存器

[0158] 如图27所示，线程异常需要有一个新的Cause寄存器的异常代码值。

[0159] (C)EntryLo寄存器

[0160] 如图28所示，一个事先被保留的高速缓存标志变成ITC指示符。

[0161] (D)Config3寄存器

[0162] 如图29的表格2900中所示，定义了新的Config3寄存器的字段，用来表示多线程ASE与多个线程上下文是否可用。

[0163] (E)Ebase

[0164] 如图30所示，Ebase寄存器的一个事先被保留的位30变成每个VPE上下文中的一个VPE的禁止位。

[0165] (F)SRSCtl

[0166] 先前预设定的HSS字段现在变成了ThreadConfig寄存器的WiredThread字段的一个功能。

[0167] 未使用FORK指令的线程分配与初始化

[0168] 在一较佳实施例中，一个操作系统“手动”产生一个线程的过程如下：

[0169] 1.执行一个DMT，用以停止其它线程的执行或是可能的FORK指令执行。

[0170] 2.通过将连续的值设定在ThreadControl寄存器的AlternateThread字段，并用MFTR指令来读取ThreadStatus寄存器，来识别一个可获得的线程上下文。一个空闲的线程在其ThreadStatus寄存器中不会有Halted或Activated位被设定。

[0171] 3.设定选定线程的ThreadStatus寄存器的Halted位，用以避免其被其它线程配置。

[0172] 4.执行一个EMT指令去重新使能多线程。

[0173] 5.使用MTTR指令并使其u字段设定为1，来复制任何需要的GPR至选定的线程上下文。

[0174] 6.使用MTTR指令并使其u和sel字段设定为0且rt字段设定为14(EPC)，从而写入所需的开始执行地址至该线程的内部重启动地址寄存器中。

[0175] 7.使用MTTR指令将0和1分别写入选定的ThreadStatus寄存器的Halted位和Activated位。

[0176] 然后该新分配的线程可以被调度。如果在该过程中设定了EXL或ERL，由于他们隐含了禁止多线程的执行，则执行DMT，设定新线程的Halted位和执行EMT的这些步骤可以被省略。

[0177] 未使用YIELD指令的线程的终止和释放

[0178] 在本发明的一较佳实施例中，一个操作系统用来终止当前线程的过程如下：

[0179] 1.如果操作系统不支持关于线程下溢状态的线程异常，则使用MFTR指令来扫描ThreadStatus寄存器的设定，以检验在处理器上有另一个可运行的线程，相反的如果没有，就向程序发出错误信号。

[0180] 2.写入任何重要的GPR寄存器的值至内存。

[0181] 3. 在 Status/ThreadStatus 寄存器中，设定内核模式（Kernel mode）。

[0182] 4. 当目前的线程维持在一个特权状态时，清除 EXL/ERL 来允许其他线程被调度。

[0183] 5. 使用一个标准的 MTC0 指令来写入 0 值至 ThreadStatus 寄存器的 Halted 与 Activated 位。

[0184] 正常的过程是一个线程按照这种方式终止自己。在一个特权模式当中，一个线程也可以使用 MTTR 指令来终止另一个线程，只不过会有额外的问题产生，这时操作系统需要决定应该释放哪个线程上下文，以及在哪点上该线程的运算状态是稳定的。

[0185] 线程间通讯的储存（Inter-Thread Communication Storage）

[0186] 线程间通讯（ITC）的储存是一个可选择的功能，其可以替代用于细粒度多线程的关联载入 / 条件存储同步方法。因为通过载入与储存的动作来操作，所以这个 ITC 存储在指令集架构中是不可见的，但是在特权资源架构中，它是可见的，并且需要有效的微架构的支持。

[0187] 参照虚拟内存页面，其包含有被标示为 ITC 储存的 TLB 项目，可以被归为是一个有特定属性的储存。每一个页面映射至一组 1-128 个 64 位的存储位置，其中每一个存储位置都有一个与其相关的 Empty/Full 位，并且可以使用标准的载入和储存指令，以四个方法之一来访问该存储位置。该访问模式被编码在所产生的虚拟地址的最低有效（和未翻译）位，如图 31 的表格 3100 所示。

[0188] 因此每一个储存位置可以用 C 语言的结构来描述 ：

[0189]

```
struct{
    unit64 ef_sync_location;
    unit64 force_ef_location;
    unit64 bypass_location;
    unit64 ef_state;
} ITC_location;
```

[0190] 其中，全部四个位置都参考潜在存储空间的相同 64 位。当每次访问实施同样的 Empty/Full 协议时，该储存的参考可以具有小于 64 位的访问类型（例如，LW，LH，LB）。

[0191] Empty 与 Full 位不相同，因此不相互耦合的多项目数据缓冲器，如 FIFO，可以被映射至 ITC 储存空间。

[0192] 可以通过向和从通用储存器复制 {bypass_location，ef_state} 对的方式来保存和恢复 ITC 的存储。严格的说，当 64 位的 bypass_location 必须被保留时，只有 ef_state 的最低有效位需要被操控。在多项目的数据缓冲器中，每一个位置必须被读取直到 Empty 位，从而通过拷贝读出该缓冲器的内容。

[0193] 每 4K 页面的位置数目与每一个 VPE 的 ITC 页面的数目都是 VPE 或处理器可以设定的参数。

[0194] ITC 储存的“物理地址空间”可以是全局的，跨越一个多处理器系统中的所有 VPE 和处理器，这样一个线程便可以从正在执行该线程的一个 VPE 同步到另一个不同 VPE 的一个位置上。全局的 ITC 储存地址可以从每一个 VPE 的 EBase 寄存器的 CPUNum 字段上取得。

该 CPUNum 的 10 个位对应于 ITC 储存地址的 10 个有效位。一般为了单处理器的应用所设计的处理器或内核不需要输出一个物理接口至 ITC 储存，并且可以将其作为一个处理器内部的资源。

[0195] 多 VPE 处理器

[0196] 一个内核或处理器可以实现多个共享资源的 VPE，如共享功能单元。每一个 VPE 都可以看到自己的在 MIPS32 或 MIPS64 指令中的具体实施和特权资源架构。每一个都可以看到自己的寄存器堆或线程上下文阵列，并且也可看到自己的 CP0 系统协处理器和自己的 TLB 状态。对于在具有 2-CPU 高速缓存相干的 SMP 多处理器的软件而言，在同一个处理器上的两个 VPE 是无法区分的。

[0197] 一个处理器上的每个 VPE 都可以在 CP0 在 Ebase 寄存器的 CPUNum 字段中看到一个不同的值。

[0198] 处理器架构上的资源，如线程上下文，TLB 储存和协处理器，可以在硬件式的配置下与 VPE 绑定，或者可以在一个支持必须的配置能力的处理器中被动态地配置。

[0199] 重置与虚拟处理器配置

[0200] 为了能够反向兼容 MIPS32 与 MIPS64 PRA，在重置时，一个可配置的多线程 / 多 VPE 处理器必须具有完全的默认线程 /VPE 配置。一般的情形下都是如此，但是对于一个有单一线程上下文的单一 VPE 却不一定必须如此。Config3 寄存器的 MVP 位可以在重置时被取得，用来决定动态的 VPE 配置是否是可能的。如果这项能力被忽略，例如在传统的软件中，该处理器就会按照默认配置中的每个具体设定来操作。

[0201] 如果该 MVP 位被设定，则寄存器 Config3 的 VPC（虚拟处理器配置）位就可以通过软件设定。这可以使得处理器进入一个设定状态，该设定状态下，Config4 寄存器的内容可以被读出，用以决定可使用的 VPE 上下文，线程上下文，TLB 项目和协处理器的数目，并且可使某些正常情况下只读的 Config 寄存器的“预设”字段变成可写入。可以将一些限制施加在配置状态指令流上，例如它们可以被禁止使用高速缓存的或是 TLB 映射的内存地址。

[0202] 在配置状态中，可配置的 VPE 的全部数目被编码于 Config4 寄存器的 PVPE 字段中。通过将每个 VPE 的索引写入 EBase 寄存器的 CPUNum 字段中，可以选择每一个 VPE。对于被选择的 VPE，下列的寄存器字段都可通过写入而被设定。

[0203] · Config1.MMU_Size

[0204] · Config1.FP

[0205] · Config1.MX

[0206] · Config1.C2

[0207] · Config3.NThreads

[0208] · Config3.NITC_Pages

[0209] · Config3.NITC_PLocs

[0210] · Config3.MVP

[0211] · VPESchedule

[0212] 并不是所有上述所提到的设定参数都需要是可配置的。举例来说，即使每一个 VPE 的 ITC 页面是可配置的，每一个页面的 ITC 位置的数目可以是固定的，或者两个参数都可以被固定，对于每个 VPE，FPU 可以被预先分配或是硬连线的，等等。

[0213] 协处理器作为分离的不同单元被分配给 VPE。协处理器可被多线程化的程度应该经由协处理器特定的控制和状态寄存器被表示和控制。

[0214] 通过清除 EBase 寄存器的 VPI 禁止位，使能一个 VPE，用于配置之后的执行。

[0215] 通过发出一个 ECONF 指令可以退出该配置状态。这个指令使得所有未被禁止的 VPE 取得一个重置异常，并且开始同时执行。如果 Config3 寄存器的 MVP 位在配置期间被清除，而且被一个 ECONF 指令保持为零，则该 VPC 位就不能再被设定，而且该处理器配置就会被有效的冻结，直到下一个处理器重置。如果 MVP 仍然被设定，则再次设定该 VPC 位可以使一个操作系统再次进入配置模式。可是，如果该处理器的一个运行中的 VPE 重新进入配置模式，可能会有不可预测的结果。

[0216] 对于多线程处理器的服务质量调度

[0217] 到目前为止该说明书描述了一种 MIPS 兼容系统的具体应用扩展，用于实现多线程。如前面所述，所描述的 MIPS 实现只是用于列举描述，而并不用于限制本发明所包含的范围。如同之前所描述的功能与机制可以应用于 MIPS 之外的系统。

[0218] 在实时和近乎实时线程的多线程中的特殊服务的问题在背景段落被提出，该问题在之前有关于 ThreadSchedule 寄存器(图 23)和 VPESchedule 寄存器(图 24)的说明中已经简单地涉及。本说明书的以下部分将更详细地处理该问题；也更清楚地说明用于具体处理线程级服务质量(QoS)的特定扩充。

[0219] 背景

[0220] 一般用于传输多媒体数据的网络设计都会牵涉到服务质量(QoS)的概念，用来描述需要使用不同策略来处理在网络中不同的数据流。以语音的传输为例，相对的对于带宽的要求不高，但是却不能忍受几十毫秒的延迟。在宽带的多媒体网络中，QoS 协议可以保证在时间为关键要素的传输中，能取得任何特别的处理与优先权，这是在时间上能适时传输的必须保证。

[0221] 影响在单一芯片上 RISC 与 DSP 相组合的程序执行的其中一个主要问题是，在一个组合的多任务的环境中，要去保证 DSP 程序代码的严格实时执行是非常困难的。从而该 DSP 应用可被视为，在处理器带宽中需要一个 QoS 条件。

[0222] 多线程与 QoS

[0223] 有许多种方式可以对来自多线程的指令发出进行调度。交错式的调度器可以在每个周期改变线程，而块交错式的调度器可以在当一个高速缓存丢失或其它严重的停顿发生时改变线程。以上详细描述的多线程 ASE，提供了一个架构给多线程处理器，用于避免对于一个特定线程调度的机制或策略任何依赖。然而，调度策略可能对于 Qos 为各种线程的执行所提供保证的内容有重大的影响。

[0224] 一个具有 DSP 扩充功能的 RISC 在 Qos 能够保证实时 DSP 程序代码能够被执行的情形下会变的更有用处。在该处理器上实现多线程，使得在一个独立的线程上执行 DSP 程序代码，甚至也有可能是在一个独立的虚拟处理器上执行 DSP 程序代码，使得 DSP 线程的硬件调度能够被可编程地确定来保证 QoS，从而理所当然的消除了具有 DSP 加强功能的 RISC 的一个主要障碍。

[0225] QoS 线程调度算法

[0226] 服务质量线程的调度可以宽松的被定义为一组调度机制和策略，其允许程序员或

系统设计者对于一段特定的程序代码的执行时间可以作出确信的和可预测的陈述。一般来说，这些陈述的形式为“这段程序代码将执行不多于 Nmax 且不少于 Nmin 个周期”。在许多的情形下，只有 Nmax 数字在实际的执行中被考虑，但是在某些应用中，程序代码的运行超前于调度也会造成问题，所以 Nmin 也应该被考量。如果 Nmax 数字与 Nmin 数字的差距能更小的话，整个系统的行为也更能精确的被预测。

[0227] 简单的优先权方案

[0228] 一种简单的模型被提出来，用于在多线程发出调度时提供一定程度的 QoS，其为简单地将最高优先权分配给一个指定实时线程，因此当该线程可执行时，总是选择它来发出指令。这种方式可以提供一个 Nmin 的最小值，似乎也可提供该指定线程的 Nmax 的可能的最小值，但是仍然会有一些不太好的后果。

[0229] 首先，在该方案中只有一个线程可以有 QoS 的保证。该算法暗含了在一个不同于该指定实时线程的线程中，任意程序代码的 Nmax 会变成实质上不受约束。其次，当该特定线程内的一段程序代码的 Nmin 数被最小化的时候，则异常就必须被包含在模型当中。如果该指定线程产生该异常，则该 Nmax 的值就会变的更复杂，并且在某种情形下是不可能确定的。如果该异常是被该指定线程以外的线程所产生的，则该指定线程中的程序代码的 Nmax 就会受到严格约束，但是该处理器的中断响应时间变得不受约束。

[0230] 这种简单的优先权方案也许在某些情形下是有用的，并且在硬件的实现上也有实际的优点，但是它们仍然没有提供一个通用的 QoS 调度的解决方案。

[0231] 基于保留的方案

[0232] 另一个功能更强大且独特的线程调度模型是基于保留发出时段。在这种方案中，硬件调度机制允许一个或多个线程可被分配得到 M 个连续发出时段中的 N 个。在一个没有中断的环境中，对于一个实时代码段，该方案并没有提供向优先权方案所提供的低 Nmin 值，可是却拥有了其它的优点。

[0233] · 多于一个的线程可以被保证有 QoS。

[0234] · 即使当中断是与具有最高优先权的线程之外的其他线程绑定的，该中断延迟也可以是受约束的。这样可以使实时程序代码区段有较低的 Nmax。

[0235] 基于保留方案的调度的一种简单形式是，将每第 N 个发出时段分配给一个实时线程。在 1 与 2 之间并没有 N 的中间值，这说明在一个多线程环境中的实时线程可以取得最多 50% 的处理器的发出时段。当一个实时任务需要使用多于 50% 的嵌入式处理器的带宽时，十分需要一种方案，其允许更灵活地分配发出带宽。

[0236] 具有 QoS 的混合线程调度

[0237] 上述的多线程系统是侧重中立的调度策略，但是还可以被扩充，以允许形成一种混合的线程调度模型。在这种模型中，实时线程可以被给予一定比例的线程发出时段的固定调度，并且用与实现方式相关的默认调度方案来分配剩余的时段。

[0238] 绑定线程至发出时段

[0239] 处理器中的指令是被快速地顺序发出。在一个多线程的环境当中，在多数的线程当中，一个线程可以通过在一个给定的时段数目中所占用的时段数目的比例来计算出所使用的带宽。相反的，本发明认识到，可以任意的声明一确定数目的时段，并限制该处理器为某特定的线程保留该固定数目中的一定数目的时段。从而可以指定带宽中的一个固定部分

来保证一个实时线程。

[0240] 很清楚地，可以将时段按比例地分配给多于一个实时线程，并且该方案进行操作的粒度受到发出时段的该固定数目的约束，所述比例就是以该固定数目为基础得到的。举例来说，如果选择 32 个时段，则任意一个特定的线程可以被保证具有带宽的 1/32 至 32/32。

[0241] 也许用于将固定发出带宽分配给线程的最具一般性的模型是将每个线程与一对整数 {N,D} 关联，这对整数表示分配给该线程的发出时段比例的分子与分母，例如是 1/2 或 4/5。如果所允许的整数范围足够大的话，这样可以允许对于线程优先权分配的几乎任意细粒度的调整，但是如此做的话还是会有一些实质上的缺点。其中一个问题是，要使用一个硬件逻辑将一个很大的配对集合 {{N0,D0},{N1,D1},…{Nn,Dn}} 转换成一个发出调度并不是一件简单的事，并且多于 100% 的时段被分配这种错误情况无法非常容易的被检测出来。另一个问题就是，这种方案允许在相当长的一段时间上，一个线程被分配 N/D 比例的发出时段，但是它并不一定允许任意声明关于哪个发出时段被分配给一个较短子程序代码片段的线程。

[0242] 因此，在本发明的一较佳实施例中，不使用整数对，而是用一个位向量与每一个需要有实时带宽 QoS 的线程相关联，该位向量表示要被分配给该线程的调度时段。在该较佳实施例中，该向量也就是前述 ThreadSchedule 寄存器(图 23)的内容，可以被系统软件所看到。虽然该 ThreadSchedule 寄存器具有 32 位宽的调度“掩码”，但是在其他实施例中该屏蔽当中可以具有更长或更短的位宽度。一个具有 32 位宽度的线程调度掩码可以允许一个线程被分配从 1/32 至 32/32 的该处理器的发出带宽，并且也可进一步对于特定的发出线程给予特定的发出带宽模式。对于一个 32 位的掩码，值 0xaaaaaaaa 将每第二个时段分配给该线程。值 0x0000ffff 也可以将 50% 的发出带宽分配给该线程，但是以 16 个连续时段的块方式进行分配。将值 0xeeeeeeee 分配给线程 X 并且将值 0x01010101 分配给线程 Y，从而给予线程 X 每四个周期中的三个(32 个中的 24 个)和给予线程 Y 每八个周期中的一个(32 个中的 4 个)，并且将剩下的每一组 32 个周期中的 4 个，被其它可能确定性较低的硬件算法分配给其它线程。更进一步的说，线程 X 将具有每四个周期中的三个，并且该线程 Y 在两组连续指令之间具有不超过八个周期。

[0243] 在该实施例中的调度冲突可以被很简单地检测出，因为没有一位将被设置在多于一个线程的 ThreadSchedule 寄存器中。也就是说，如果为一个线程设定了一个特定的位，那么对于被分配了发出掩码的所有其他线程，该位必须是零值。因此，如果有任何的冲突，都可以被轻易的检测出。

[0244] 实时线程的发出逻辑相对的简单直接：每一个发出机会皆关联至一个 32 模数的索引，该索引会被传送给所有就绪的线程，这些就绪线程中的至多一个会被分配该关联的发出时段。如果得到该时段，则该相关联的线程就会发出它的下一个指令。如果没有任何线程拥有该时段，则该处理器会选择一个可执行的非实时线程。

[0245] ThreadSchedule 寄存器的硬件实现如果使用少于 32 位的话，可以减少每个线程的储存与逻辑的大小，但却会同时降低调度的灵活性。原则上，该寄存器可以扩充至 64 位，或甚至被实现(在 MIPS 处理器的情况下)为一连串的寄存器，增加了在 MIPS32 CP0 寄存器空间中的选择值，从而提供更长的调度向量。

[0246] 使线程免除中断服务

[0247] 如前所述，中断服务可以使得执行该异常程序的线程在执行时间上具有很大的可变性。因此，期望使需要严格 QoS 保证的线程能够免除中断服务。这里提出了一个较佳的实施例，对于每一个线程利用一个单一位，该位对于操作系统是可以看见的，用来使任何异步异常延迟，直到一个非免除的线程被调度(即，ThreadStatus 寄存器的 IXMT 位，请参考图 18 与图 19)。这样会增加中断的延迟，但是通过 ThreadSchedule 寄存器的值的选择，可以将该中断延迟限制在受约束和可控的程度下。如果中断处理程序只是执行在那些没有被分配给可免除的实时 QoS 线程的发出时段中，自然地该中断服务对于该实时程序代码的执行时间就没有任何优先的影响。

[0248] 线程与虚拟处理单元的发出时段分配

[0249] 以上详细描述的多线程 ASE 描述了一种线程资源的有层次的分配，其中一些数目的 VPE(虚拟处理单元)各自具有一些数目的线程。每一个 VPE 具有 CP0 的硬件实现和特权资源架构(当配置在一 MIPS 处理器上时)，所以运行在其中一个 VPE 上的操作系统软件(OS)不可能直接知道和控制被其它 VPE 所要求的发出时段。因此每一 VPE 的发出时段名称空间被关联到该 VPE，这就形成了一个发出时段分配的层次结构。

[0250] 图 34 为调度电路 3400 的方框示意图，其描述了这种线程资源的层次分配。处理器调度器 3402 (即，主处理器的全部调度逻辑)经由“时段选择”信号 3403 传递一个发出时段号码至该主处理器内的全部 VPE 中的全部 VPESchedule 寄存器。信号 3403 对应于 VPESchedule 寄存器内一个位的位置(在本较佳实施例中，即为三十二个位置中的一个)。通过使该位的位置在每个发出时段出现时移至一个增加的位置，并且当到达了最高有效位位置时(即，在此较佳实施例中是第 31 位)再重置到最低有效位位置(即，第 0 位)，调度器 3402 重复地循环信号 3403。

[0251] 再参照图 34，以此图为例，位位置 1 (即，时段 1)经由信号 3403 传递至该主处理器的全部 VPESchedule 寄存器，即寄存器 3414 与 3416。在任一个 VPESchedule 寄存器中，如果其对应位为“设定”(即，该位为逻辑 1)，该寄存器就用一个“VPE 发出要求”信号来通知处理器调度器。作为响应，调度器就会用一个“VPE 发出允许”信号来允许该 VPE 使用目前的发出时段。再参照图 34，(VPE0 中的)VPESchedule 寄存器 3414 的位位置 1 被设定，因此发出了一个 VPE 发出要求信号 3415 至处理器调度器 3402，然后该处理器调度器 3402 也会响应一个 VPE 发出允许信号 3405。

[0252] 当一个 VPE 被授予一个发出时，他在 VPE 层次上采用相类似的逻辑。再参照图 34，VPE 调度器 3412 (即 VPE0 3406 的调度逻辑)响应于信号 3405，而经由时段选择信号 3413 传递一个发出时段号码给该 VPE 内的全部 ThreadSchedule 寄存器。这些 ThreadSchedule 寄存器每一个都关联至由该相关 VPE 所支持的线程。信号 3413 对应于 ThreadSchedule 寄存器中的一个位位置(在本实施例中，可以是三十二个位中的一个)。通过使位位置在每个发出时段出现时移至一个增加的位置，并且当到达了最高有效位位置时(即，在此较佳实施例中是第 31 位)再重置到最低有效位位置(即，第 0 位)，调度器 3412 重复地循环信号 3403。该时段号码独立于在 VPESchedule 层次所使用的时段号码。

[0253] 请参照图 34 并以其为例，位位置 0 (即，“时段 0”)经由信号 3413 传递至在该目标 VPE 内的全部 ThreadSchedule 寄存器，也就是寄存器 3418 与 3420。任一个线程的 ThreadSchedule 寄存器的该选定位置的位已被设定的话，该线程通知 VPE 调度器，从而被

允许使用目前的发出时段。参照图 34,(线程 0 的)ThreadSchedule 寄存器 3418 的位位置 0 被设定,因此将线程发出要求信号 3419 发送至该 VPE 调度器 3412,而该 VPE 调度器也响应了一个线程发出允许信号 3417 (从而允许线程 0 可以使用目前的发出时段)。在一些周期当中,如果 VPESchedule 寄存器中没有设定与指定的时段对应的位,或是 ThreadSchedule 寄存器中没有设定与指定的时段对应的位,则该处理器或 VPE 调度器就会根据某种其它默认调度算法来分配下一个发出时段。

[0254] 根据之前所述,在一个较佳实施例当中,每一个 VPE,例如图 34 的 VPE0 (3406)和 VPE1 (3404),都具有一个 VPESchedule 寄存器(其格式显示于图 24),用来允许以该寄存器内容的长度为模的特定时段,可以被确定地分配给该 VPE。图 34 的 VPESchedule 寄存器为 VPE0 的寄存器 3414 和 VPE1 的寄存器 3416。那些没有被分配给任何 VPE 的发出时段,通过特定实现方式的分配策略进行分配。

[0255] 另外根据上文的描述,被分配给在一个 VPE 之内的线程的时段是从给予该 VPE 的时段中分配的。举一个具体例子,如果一个处理器有两个 VPE,如图 34 所示,其中一个 VPE 的 VPESchedule 寄存器具有 0xaaaaaaaa 值,而另一个 VPE 的 VPESchedule 寄存器具有 0x55555555 值,则发出时段就会被交替分配给这两个 VPE。如果这两个 VPE 之一中的一个线程的 ThreadSchedule 寄存器包含 0x55555555 值,则该线程会取得包含该线程的 VPE 的每两个发出时段中的一个,或者是说整个处理器的每四个发出时段中的一个。

[0256] 因此,该每一个 VPE 相关的 VPESchedule 寄存器的值决定了每个 VPE 会得到哪些处理时段。特定线程被分配给每一个 VPE,例如是 VPE0 中所示的线程 0 与线程 1。其它没有显示的线程也类似地被分配给 VPE1。每个线程都有一个关联的 ThreadSchedule 寄存器,例如线程 0 的寄存器 3418,线程 1 的寄存器 3420。ThreadSchedule 寄存器的值决定了一个 VPE 中每个线程的处理时段的分配。

[0257] 调度器 3402 与 3412 可以用简单的组合逻辑来实现,以执行上述的功能,根据本发明的公开内容,建构这些调度器并不需要复杂的实验就能够由本领域技术人员来实现。例如,调度器的构成也可以使用传统的方法,如通过组合逻辑,可程序逻辑,软件等等,用以得到所描述的功能。

[0258] 图 33 描述了一个通用形式的计算机系统 3300,根据本发明的各种实施例可以在该计算机系统上实施。该系统包含了具有必须的译码和执行逻辑的一个处理器 3302 (本领域一般技术人员对此应该很清楚),用以支持一个或多个上述指令(即,FORK,YIELD,MFTR,MTTR,EMT,DMT 和 ECONF)。在一个较佳实施例当中,内核 3302 还包含如图 34 所示的调度电路 3400,并且代表上述的“主处理器”。系统 3300 还包含 :系统接口控制器 3304,可以与该处理器双向通信 ;RAM 3316 和 ROM3314,可被系统接口控制器进行存取 ;三个 I/O 装置 3306,3308 和 3310,通过总线 3312 与系统接口控制器通信。通过这里对装置和程序代码应用的详细描述,系统 3300 可以作为一个多线程系统进行操作。本领域技术人员应该很清楚,图 33 中所示的一般形式可以有很多替代形式。举例来说,总线 3312 可以有许多的形式来实现,并且在某些实施例当中可以是一种芯片上的总线。同样的,I/O 装置的数目也只是为了描述方便,实质上是可以在不同的系统上做任意的变更。另外,虽然在图中只有装置 3306 发出了一个中断要求,很明显地其它的装置也可以发出中断要求。

[0259] 更进一步的改善

[0260] 到目前为止，所描述的实施例中，32 位的 ThreadSchedule 寄存器和 VPESchedule 寄存器并不允许精确地分配奇数比例的发出带宽。如果一个程序员期望精确地分配所有发出时段的三分之一给一个指定的线程，他只能近似到 10/32 或 11/32。在一个实施例中，一个具有可程序的掩码或长度的寄存器，允许程序员去指定 ThreadSchedule 寄存器和 / 或 VPESchedule 寄存器中的位的子集，在重新开始这次序列之前被发出逻辑使用。在所提出的例子当中，该程序员设定了只有 30 个位是有效的，并且将 VPESchedule 寄存器和 / 或 ThreadSchedule 寄存器适当地编程为具有值 0x24924924。

[0261] 本文所描述的多线程 ASE 当然可以实现在硬件中，例如，在中央处理单元（CPU），微处理器，数字信号处理器，处理器内核，系统整合芯片（SOC）或其它任何可编程器件内，或与上述各个器件连接。另外，该多线程 ASE 也可以实现在软件之中（例如，计算机可读程序代码，程序代码，任何形式的指令和 / 或数据，如源语言，目标语言或机器语言），该软件设置在计算机可使用的（例如，可读的）的介质中，该介质用来储存该软件。该软件实现了这里描述的装置和过程的功能，制造，建模，仿真，描述和 / 或测试。举例来说，这些可以通过使用以下工具实现 :通用编程语言（比如 C 语言，C++ 语言），GDSII 数据库，硬件描述语言（HDL），其包含 Verilog HDL、VHDL、AHDL （Altera HDL）等等，或者其它可利用的程序，数据库和 / 或电路（即原理图）设计工具。这些软件可以被置于任何已知的计算机可使用的介质，包含半导体，磁盘，光盘（例如 CD-ROM, DVD-ROM 等等），或是作为任何计算机可使用（例如，可读取）的传输介质（例如，载波，包括数字、光学的任何其它介质，或基于模拟的介质）中所容纳的计算机数据信号。因此，该软件可以在包括互联网络与内联网络的通信网络中传输。

[0262] 一个由软件实施的多线程 ASE 可以被包含在一个半导体的知识产权内核内，如一个处理器内核（例如，以 HDL 实现），并且可以在集成电路的生产过程中被转变成硬件。另外，这里描述的一个多线程 ASE 也可以作为硬件与软件的组合实现。

[0263] 对于本领域技术人员来说，很明显地可以在不超出本发明所揭示的精神与范围的情况下，对本发明所揭示的实施例加以润饰与修改。举例来说，之前所描述的实施例大多是使用 MIPS 处理器，架构和技术，作为具体例子。本发明具有各种实施例，可以被用于更广的范围，而不限于这些具体例子。更进一步而言，一个本领域技术人员可以找到方法对本发明所描述的功能性做些微的改变，而这仍然是在本发明的精神与范围内。在描述 QoS 时，ThreadSchedule 寄存器与 VPESchedule 寄存器的内容不限于所描述的长度，而且可以在本发明的精神和范围内作出修改。

[0264] 因此，实质上只能依照所附权利要求的范围来限定本发明的范围。

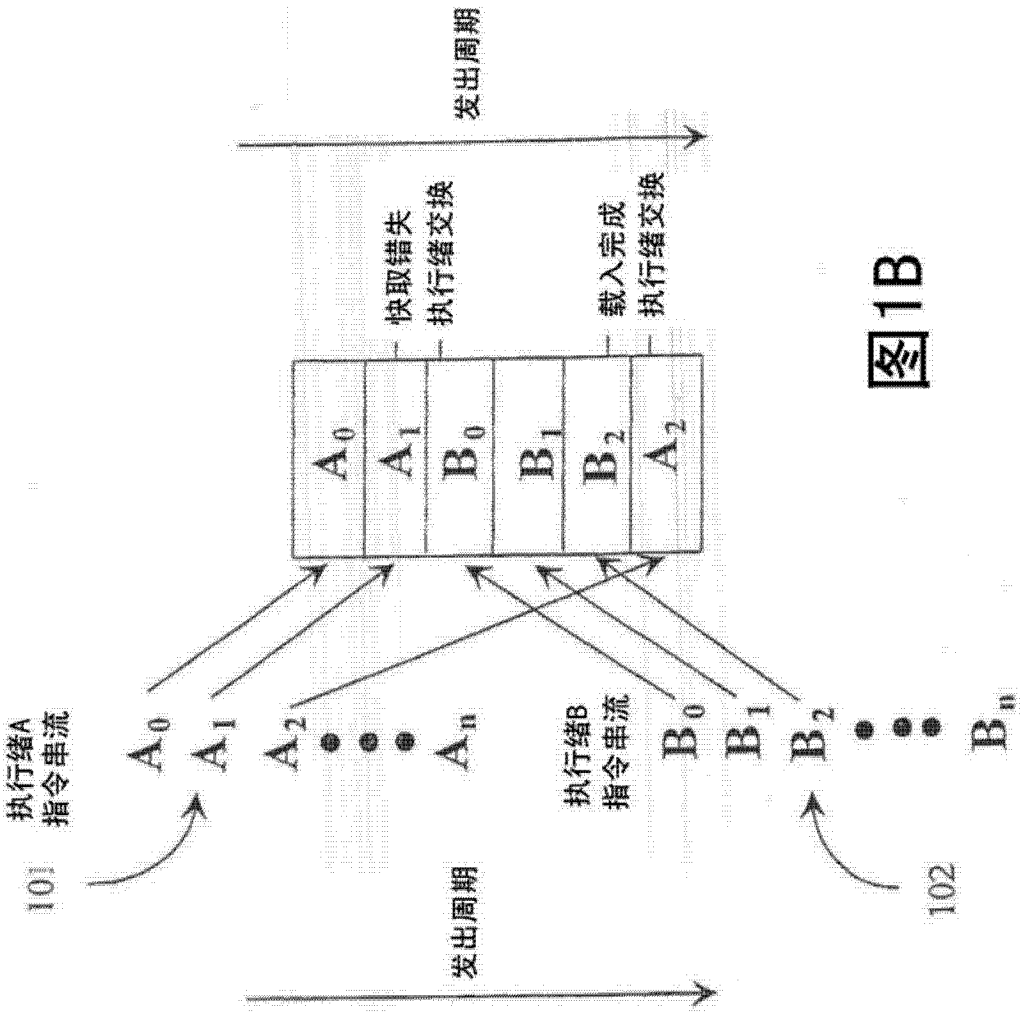

执行绪A
指令串流
101
A_0 A_1 A_2 ● ● ● A_n
发出周期
快取错失
执行绪交换
载入完成
执行绪交换
A_0 A_1 B_0 B_1 B_2 A_2
执行绪B
指令串流
B_0 B_1 B_2 ● ● ● B_n
102
发出周期

图1B

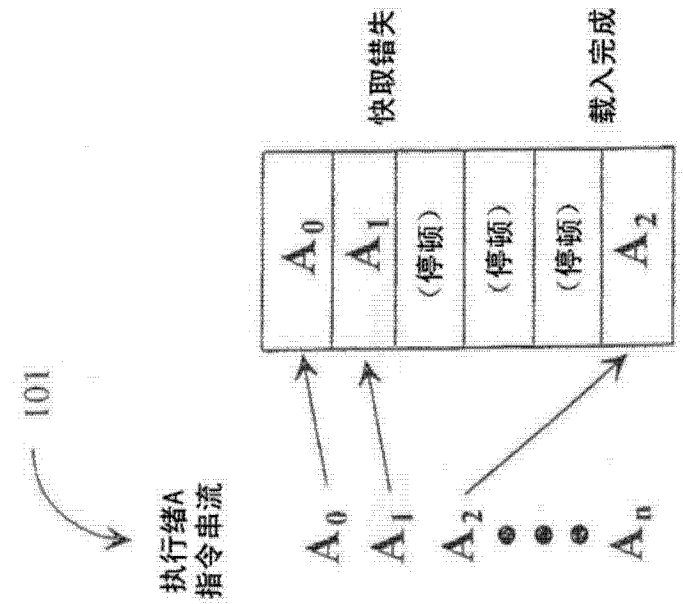

101
执行绪A
指令串流
A_0 A_1 A_2 ● ● ● A_n
快取错失
载入完成
A_0 A_1 (停顿) (停顿) (停顿) A_2

图1A

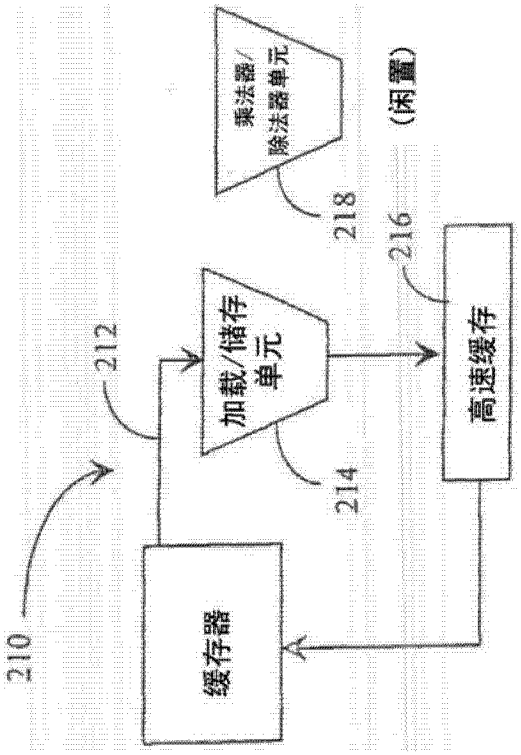

210
缓存器
212
加载/储存单元
214
乘法器/除法器单元
218
(闲置)
216
高速缓存

图 2A

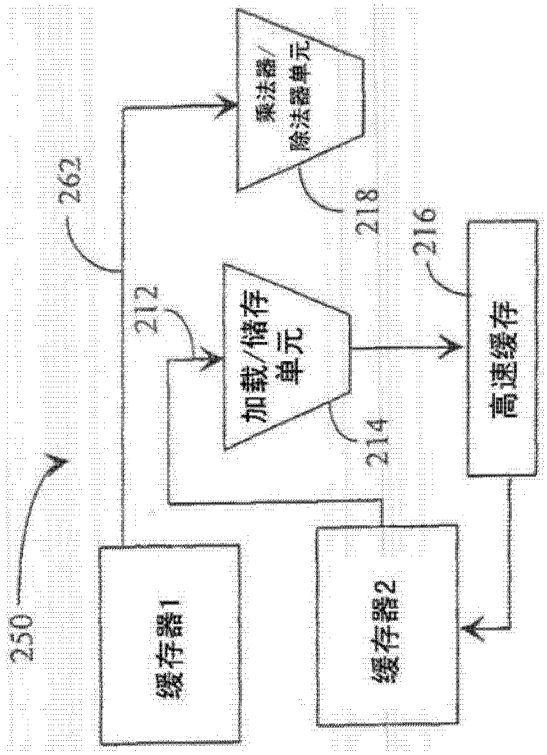

250
缓存器1
262
212
加载/储存单元
214
乘法器/除法器单元
218
缓存器2
216
高速缓存

图 2B

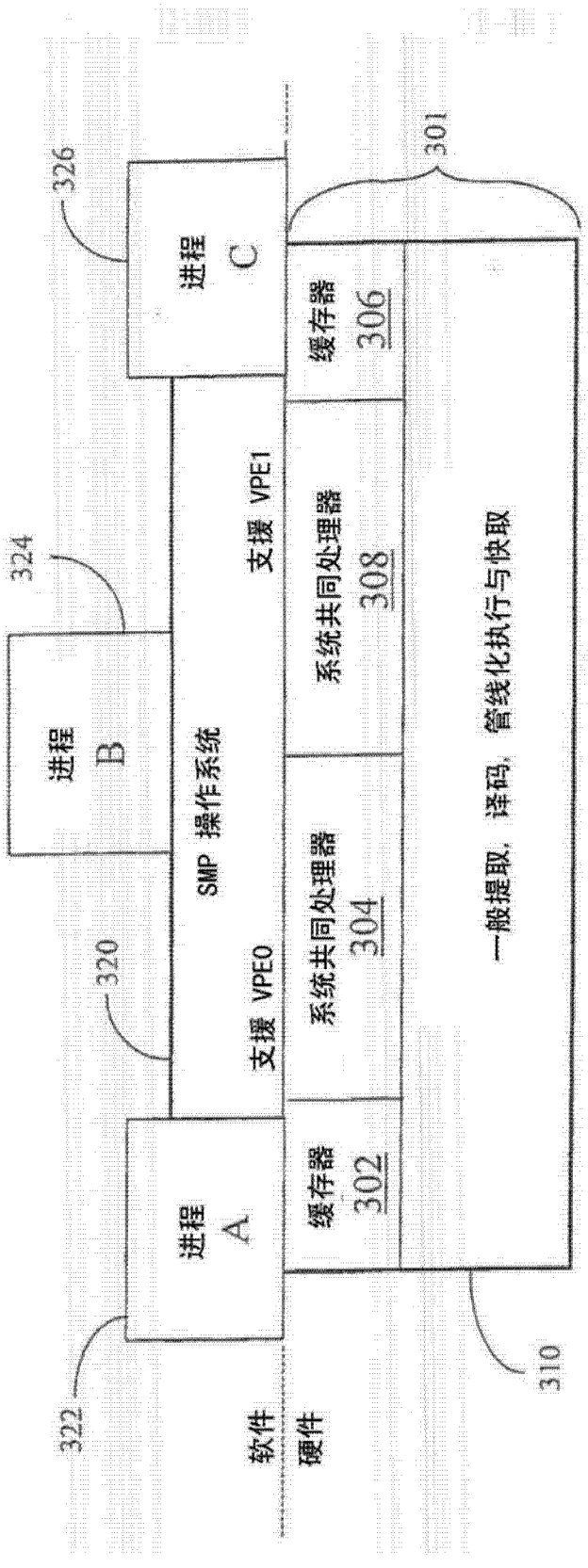

进程 A
进程 B
进程 C
SMP 操作系统
支援 VPE0
支援 VPE1
缓存器 302
系统共同处理器 304
缓存器 306
系统共同处理器 308
一般提取，译码，管线化执行与快取
301
310
320
322
324
326
软件
硬件

图 3

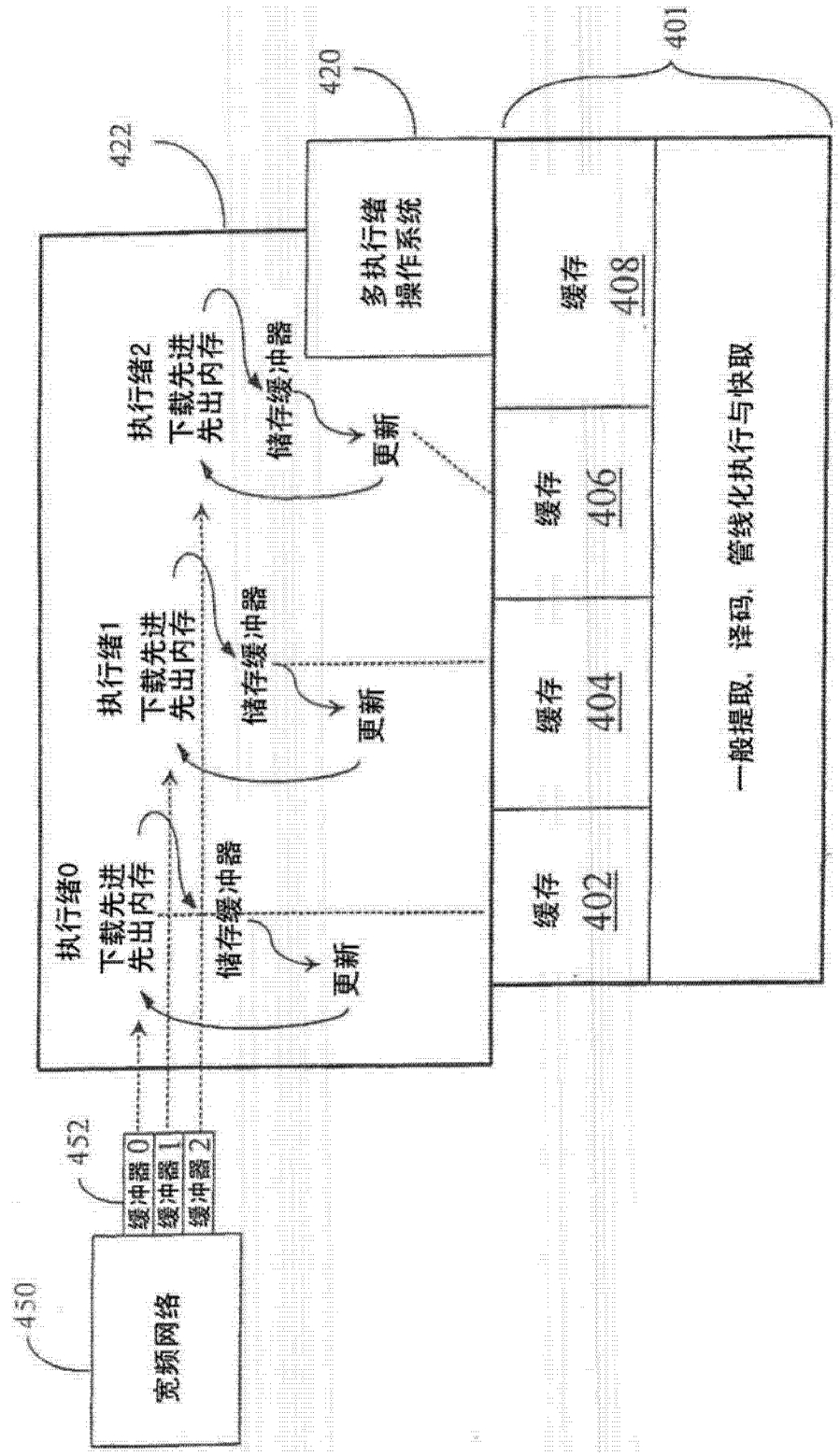

401
420
422
多执行绪操作系统
缓存 408
缓存 406
缓存 404
缓存 402
一般提取，译码，管线化执行与快取
执行绪2 下载先进先出内存
储存缓冲器
更新
执行绪1 下载先进先出内存
储存缓冲器
更新
执行绪0 下载先进先出内存
储存缓冲器
更新
452
缓冲器 0
缓冲器 1
缓冲器 2
450
宽频网络

图 4

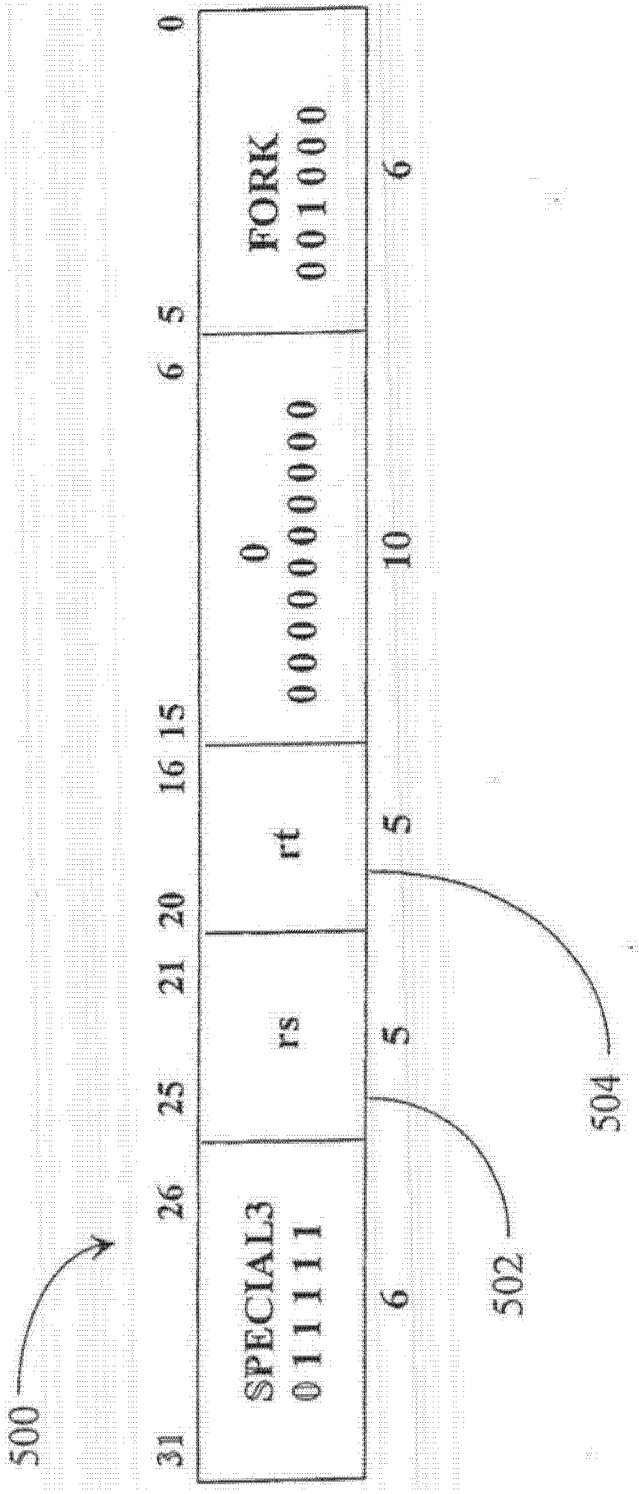

0
5
6
15
16
20
21
25
26
31
FORK
001000
0
0000000000
rt
rs
SPECIAL3
0111111
6
10
5
5
6
500
502
504

图 5

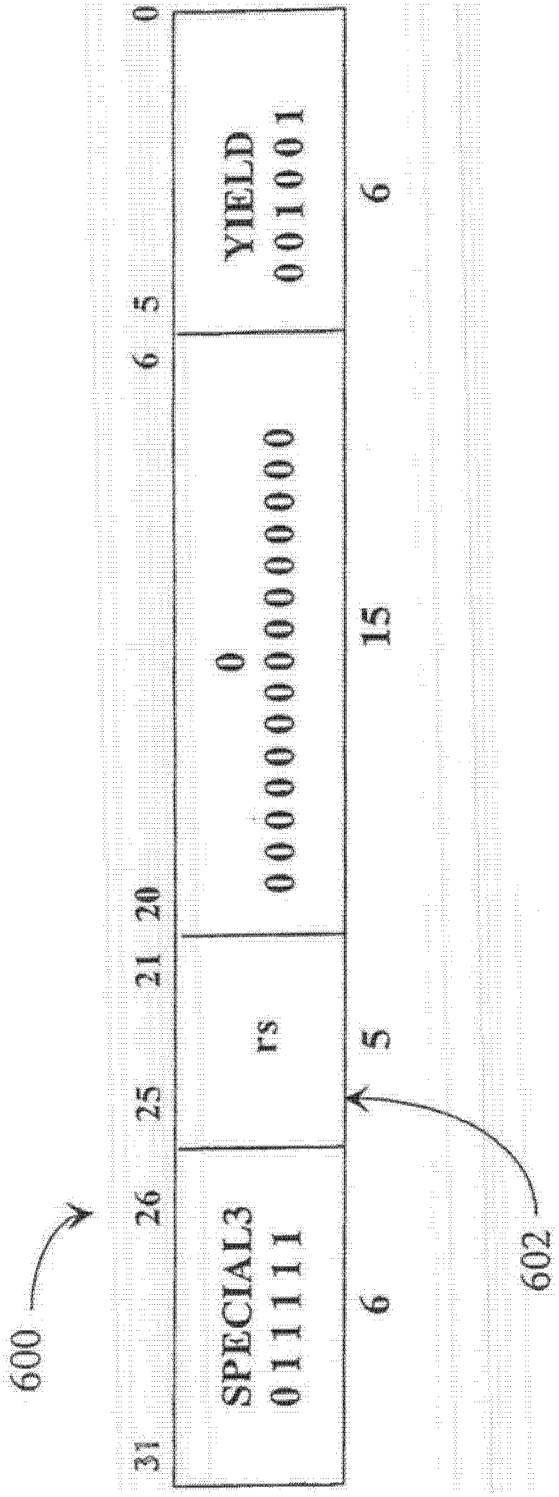
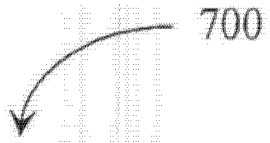

0
6
5
YIELD
001001
6
20
0
00000000000000
15
21
rs
5
25
26
SPECIAL3
011111
6
31
600
602

图 6



一般用途缓存器（GPR）rs之内容	执行绪排程致能状况
MSB:16	保留. 在非零的状态下无法预测其行为
15	IP7 中断设定
14	IP6 中断设定
13	IP5 中断设定
12	IP4 中断设定
11	IP3 中断设定
10	IP2 中断设定
9	IP1 中断设定
8	IP0 中断设定
7	下载连结（Load-Linked）状态清除
6	下载连结（Load-Linked）状态设定
5	系统相关外部事件3
4	系统相关外部事件2
3	系统相关外部事件1
2	系统相关外部事件0
1	保留. 在非零的状态下无法预测其行为
0	于下一周期致能

图 7

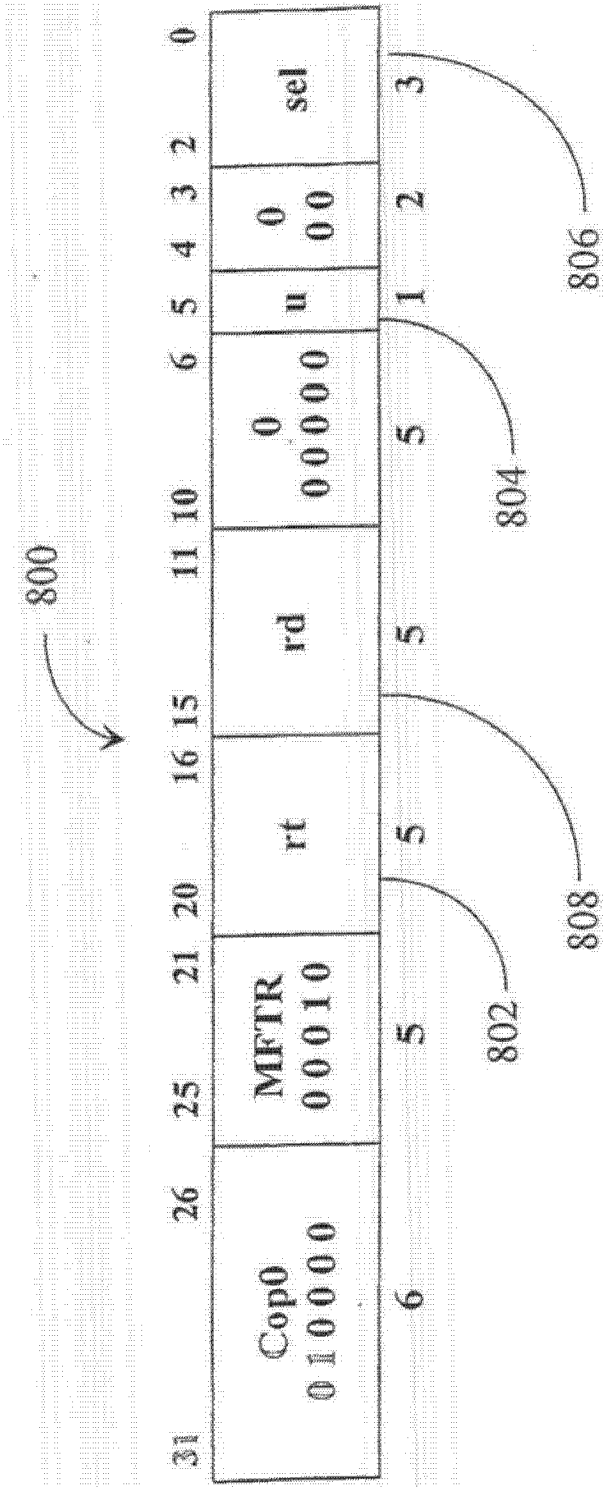

800
0
2
3
4
5
6
10
11
15
16
20
21
25
26
31
sel
0
00
u
0
00000
rd
rt
MFTR
00010
Cop0
010000
3
2
1
5
5
5
5
6
806
804
808
802

图 8

u 值	sel 值	选定缓存器
0	n	共同处理器 0 缓存器号码 rt，sel=sel
1	0	GPR[rt]
1	1	假如rt=0，Lo缓存器 假如rt=1，Hi缓存器 假如rt=2，ACX缓存器 其它rt的值，保留，不可预测
1	2	FPR[rt]
1	3	FPCR[rt]
1	4	Cop2 Data[rt]
1	5	Cop2 Control [rt]
1	>5	保留，不可预测

图 9

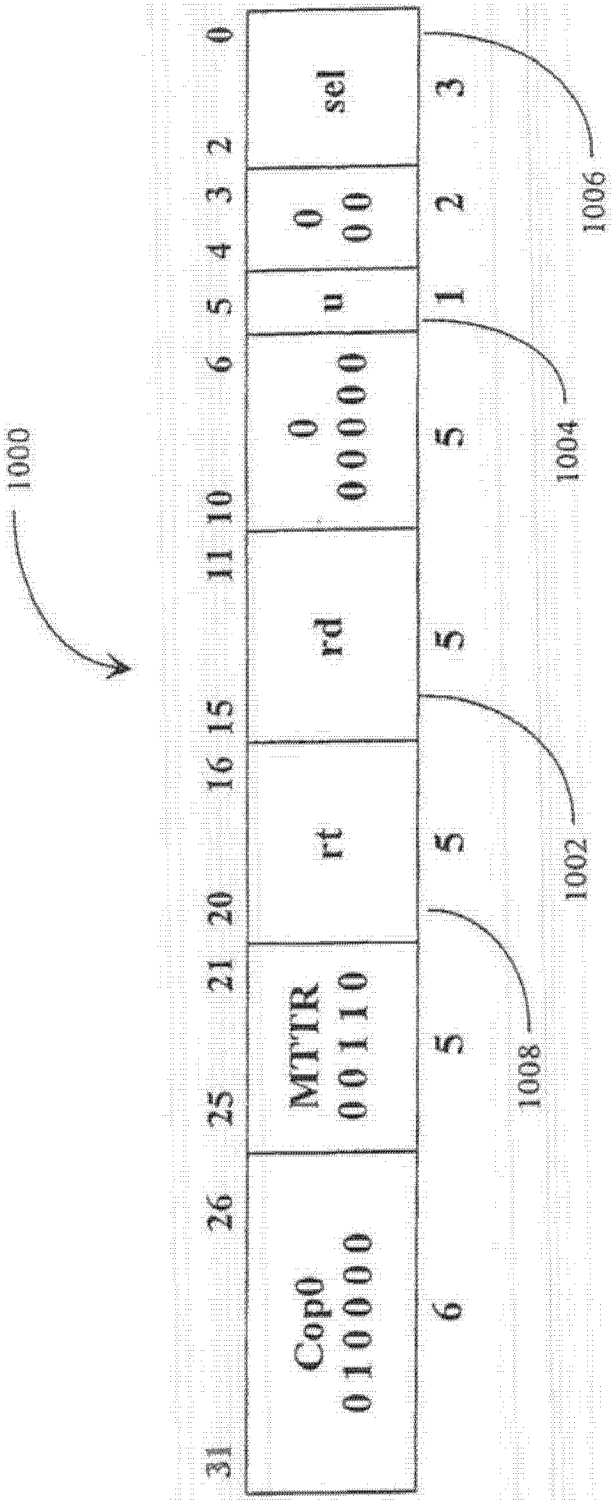
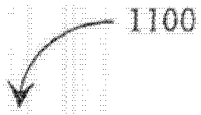

1000
0
2
3
4
5
6
10
11
15
16
20
21
25
26
31
sel
0
00
u
0
00000
rd
rt
MTTR
00110
Cop0
010000
3
2
1
5
5
5
5
6
1006
1004
1002
1008

图 10



u 值	sel 值	选定缓存器
0	n	共同处理器 0 缓存器号码 rd，sel=sel
1	0	GPR[rd]
1	1	假如rd=0，Lo缓存器 假如rd=1，Hi缓存器 假如rd=2，ACX缓存器 其它rd的值，保留，不可预测
1	2	FPR[rd]
1	3	FPCR[rd]
1	4	Cop2 Data[rd]
1	5	Cop2 Control [rd]
1	>5	保留，不可预测

图 11

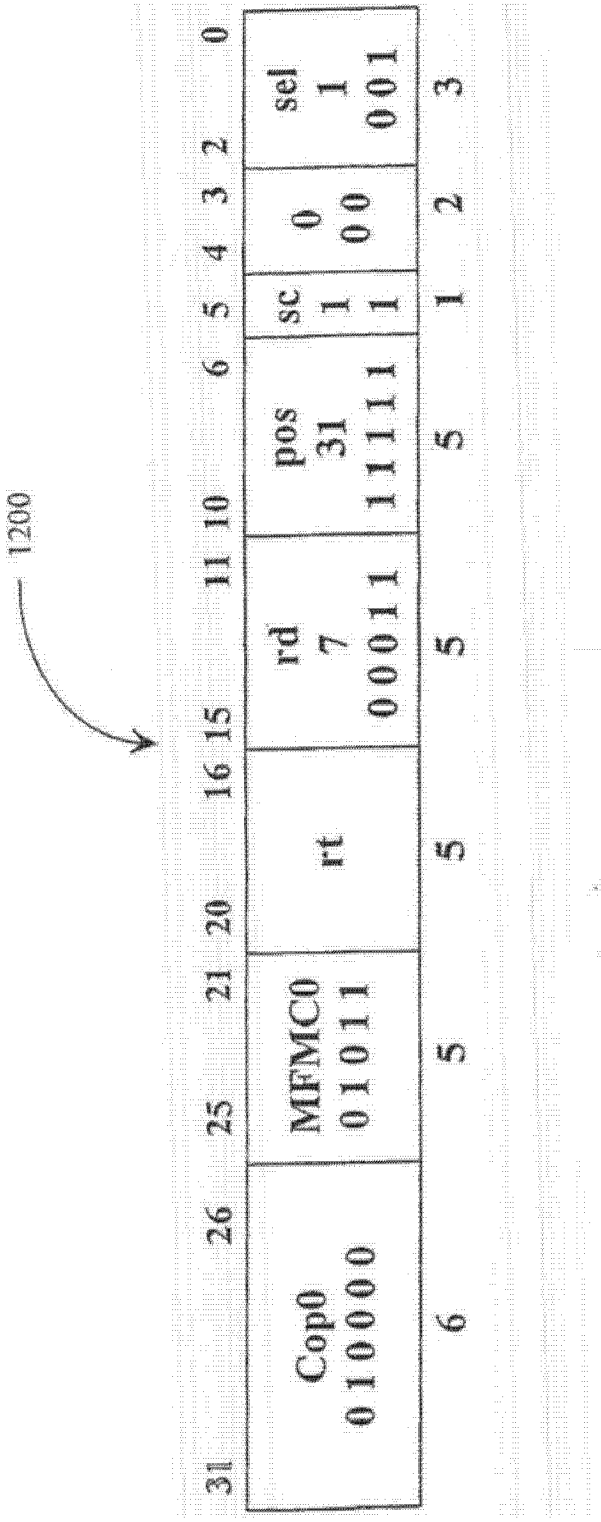

1200
31 26 25 21 20 16 15 11 10 6 5 4 3 2 0
Cop0
010000
MFMC0
01011
rt
rd
7
00011
pos
31
11111
sc
1
1
0
00
sel
1
001
6 5 5 5 5 1 2 3

图 12

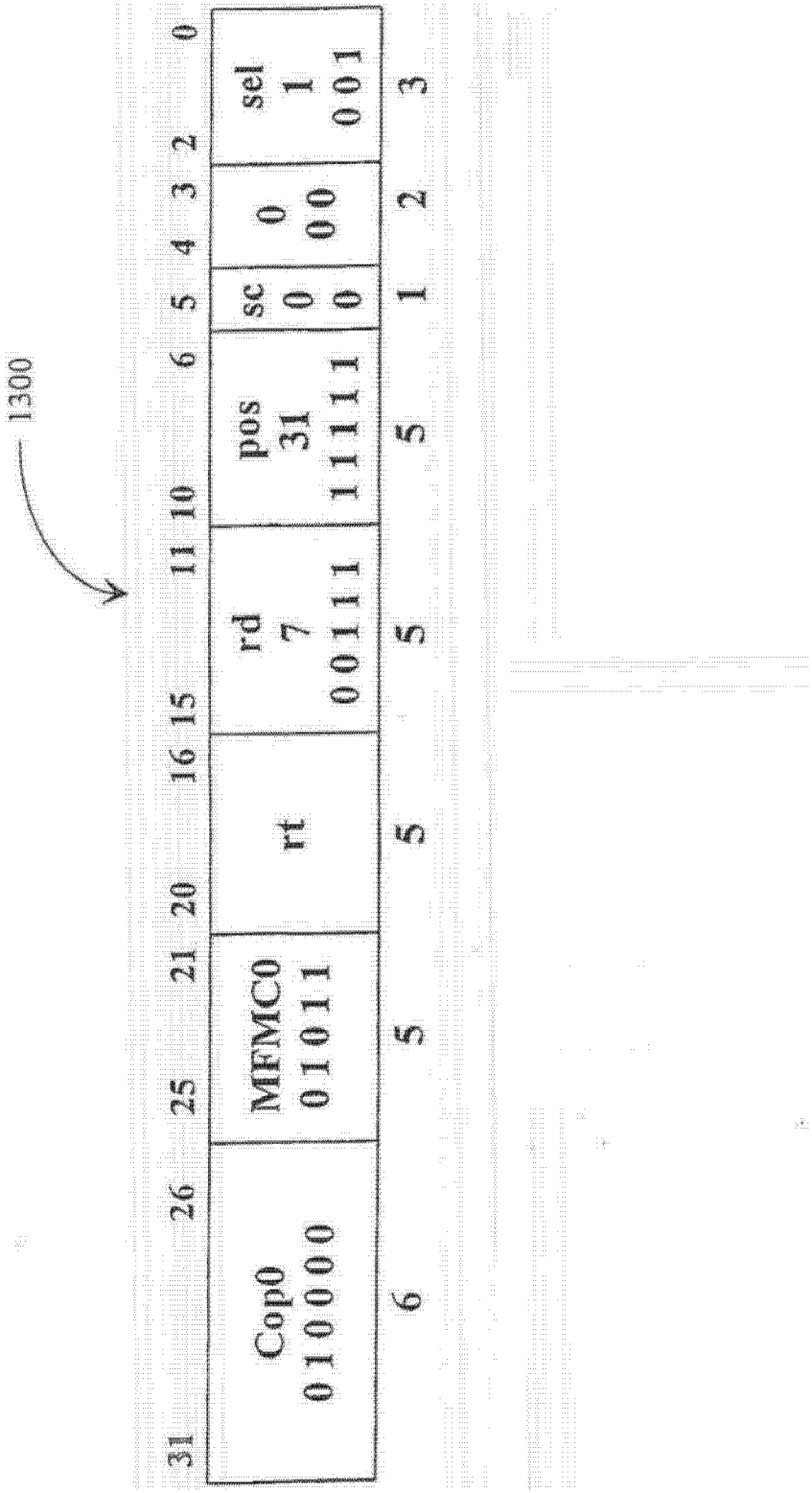

1300
31 26 25 21 20 16 15 11 10 6 5 4 3 2 0
Cop0
010000
6
MFMC0
01011
5
rt
5
rd
7
00111
5
pos
31
11111
5
sc
0
0
1
0
00
2
sel
1
001
3

图 13

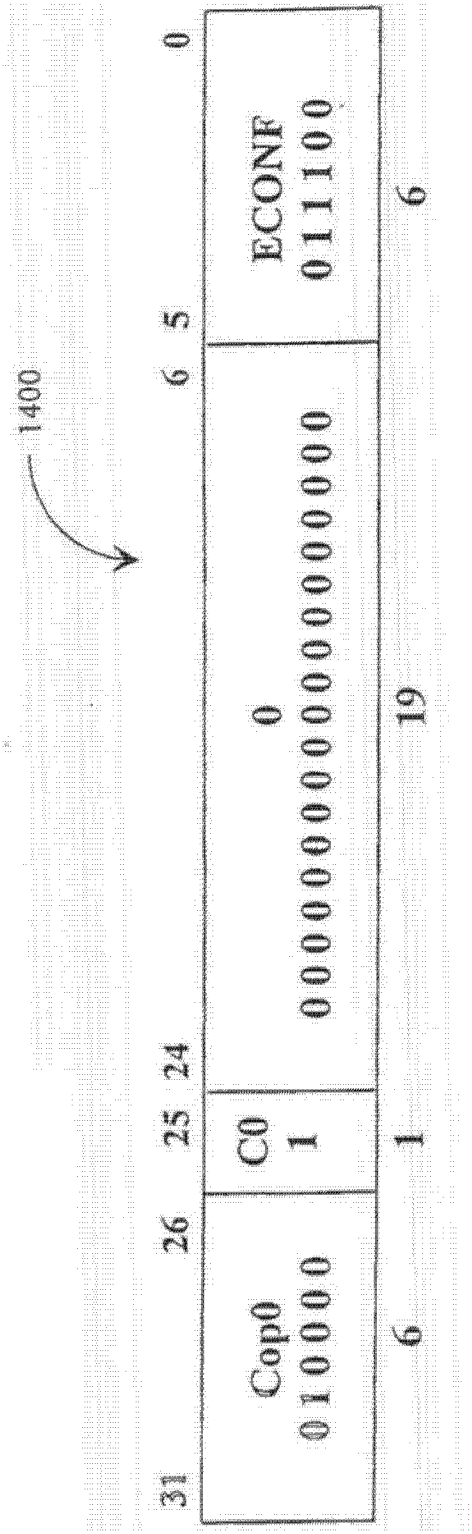

1400
0
5
24
25
26
31
6
19
1
6
ECONF
0111100
0
0000000000000000000000
C0
1
Cop0
010000

图 14

缓存器名称	新的或是变更的	CP0缓存器号码	缓存器选择号码	说明
ThreadContext	New	4	1	用于操作系统（OS）对于每一执行绪的读取/写入的储存
ThreadConfig	New	6	1	每一VPE缓存器包含相对的非挥发性执行绪设定数据
ThreadSchedule	New	6	2	选择性的执行绪缓存器用于在VPE中指定发出的频宽
VPESchedule	New	6	3	选择性的执行绪缓存器用于在处理器中指定发出的频宽
ThreadControl	New	7	1	每一VPE缓存器包含相对的挥发性执行绪设定数据
ThreadStatus	New	12	4	每一执行绪状态的信息，包含Status 与EntryHi 缓存器的执行绪指定的复制版本
Config4	New	16	4	对于每一处理器设定信息而有关于执行绪与 VPE相关的资源
Status	Modified	12	0	对于CU3-CU1语意的附加要素
Cause	Modified	13	0	新的Cause码
EntryLo	Modified	2,3	0	先前保留的快取属性所指定的
Config3	Modified	16	3	附加的字段用于描述执行绪相关资源的设定
EBase	Modified	15	1	先前保留的位用于禁止有与CPUNum值相关的VPE
SRSCtl	Modified	12	2	先前使用硬件连结的字段而改用软件与该 ThreadConfig缓存器的功能

1500

图 15

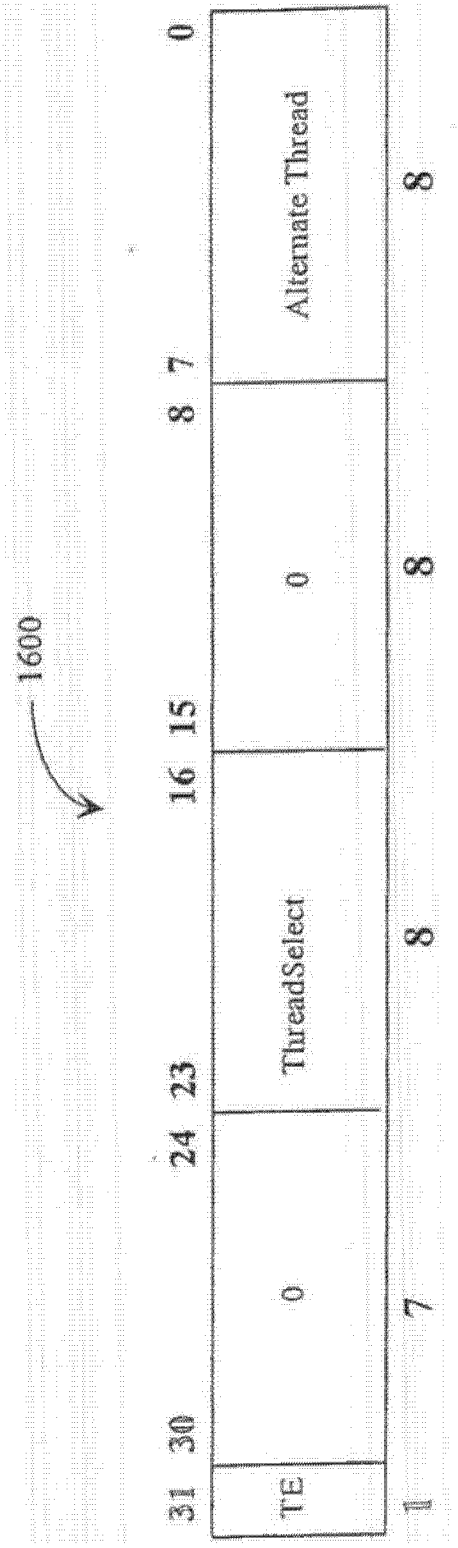
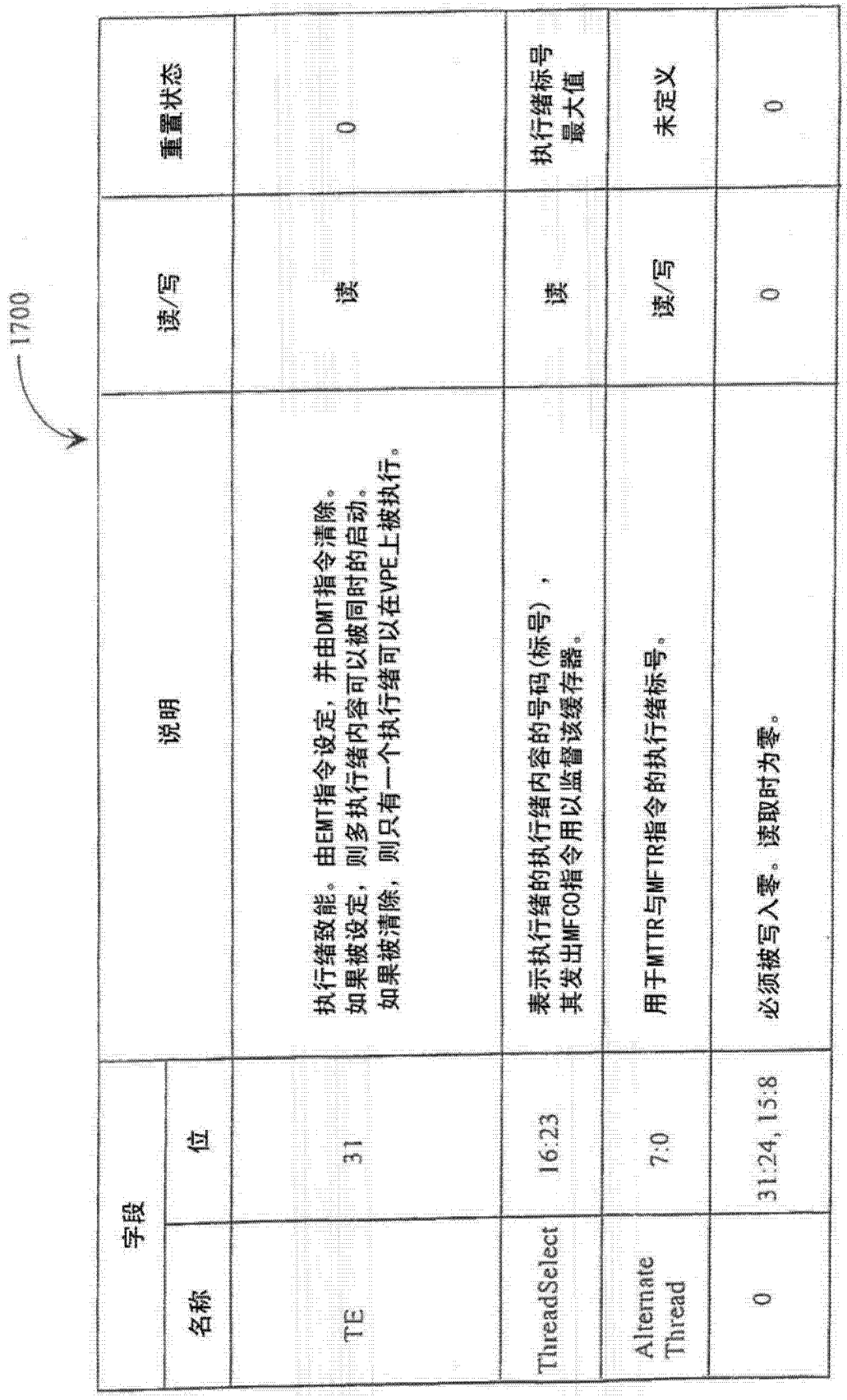

1600
0
8
16
24
30
31
7
15
23
TE
0
ThreadSelect
0
Alternate Thread
8
8
8
7
1

图 16



字段		说明	读/写	重置状态
名称	位			
TE	31	执行绪致能。由EMT指令设定，并由DMT指令清除。 如果被设定，则多执行绪内容可以被同时的启动。 如果被清除，则只有一个执行绪可以在VPE上被执行。	读	0
ThreadSelect	16:23	表示执行绪的执行绪内容的号码(标号)， 其发出MFC0指令用以监督该缓存器。	读	执行绪标号 最大值
Alternate Thread	7:0	用于MTTR与MFTR指令的执行绪标号。	读/写	未定义
0	31:24, 15:8	必须被写入零。读取时为零。	0	0

图 17

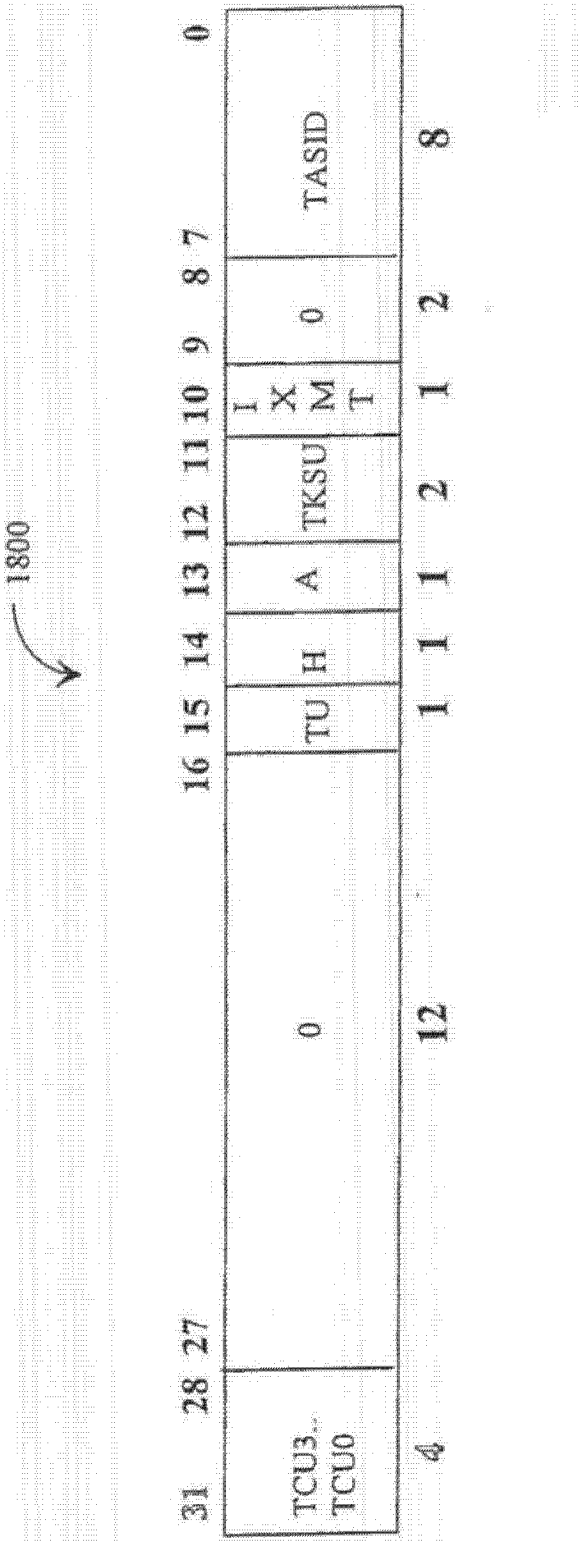
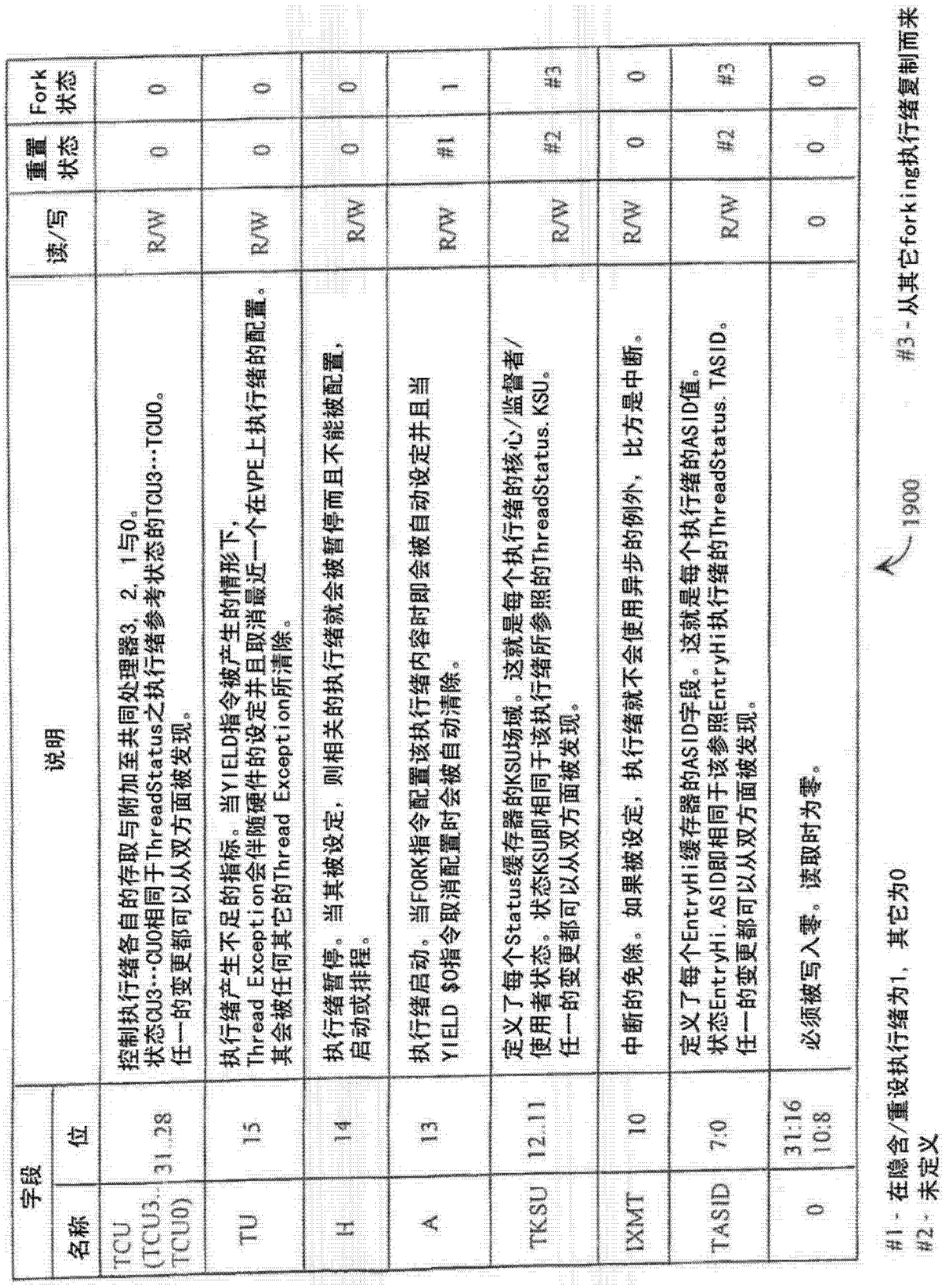

1800
0
7
8
9
10
11
12
13
14
15
16
27
28
31
TASID
0
I
X
M
T
TKSU
A
H
TU
0
TCU3..
TCU0
8
2
1
2
1
1
1
12
4

图 18



字段 名称	字段 位	说明	读/写	重置状态	Fork状态
TCU (TCU3…TCU0)	31…28	控制执行绪各自的存取与附加至共同处理器3，2，1与0。 状态CU3…CU0相同于ThreadStatus之执行绪参考状态的TCU3…TCU0。 任一的变更都可以从双方面被发现。	R/W	0	0
TU	15	执行绪产生不足的指标。当YIELD指令被产生的情形下， Thread Exception会伴随硬件的设定并且取消最近一个在VPE上执行绪的配置。 其会被任何其它的Thread Exception所清除。	R/W	0	0
H	14	执行绪暂停。当其被设定，则相关的执行绪就会被暂停而且不能被配置， 启动或排程。	R/W	0	0
A	13	执行绪启动。当FORK指令配置该执行绪内容时即会被自动设定并且当 YIELD $0指令取消配置时会被自动清除。	R/W	#1	1
TKSU	12..11	定义了每个Status缓存器的KSU场域。这就是每个执行绪的核心/监督者/ 使用者状态。状态KSU即相同于该执行绪所参照的ThreadStatus.KSU。 任一的变更都可以从双方面被发现。	R/W	#2	#3
IXMT	10	中断的免除。如果被设定，执行绪就不会使用异步的例外，比方是中断。	R/W	0	0
TASID	7:0	定义了每个EntryHi缓存器的ASID字段。这就是每个执行绪的ASID值。 状态EntryHi.ASID即相同于该参照EntryHi执行绪的ThreadStatus.TASID。 任一的变更都可以从双方面被发现。	R/W	#2	#3
0	31:16 10:8	必须被写入零。读取时为零。	0	0	0

1900

#1 - 在隐含/重设执行绪为1，其它为0
#2 - 未定义
#3 - 从其它forking执行绪复制而来

图 19

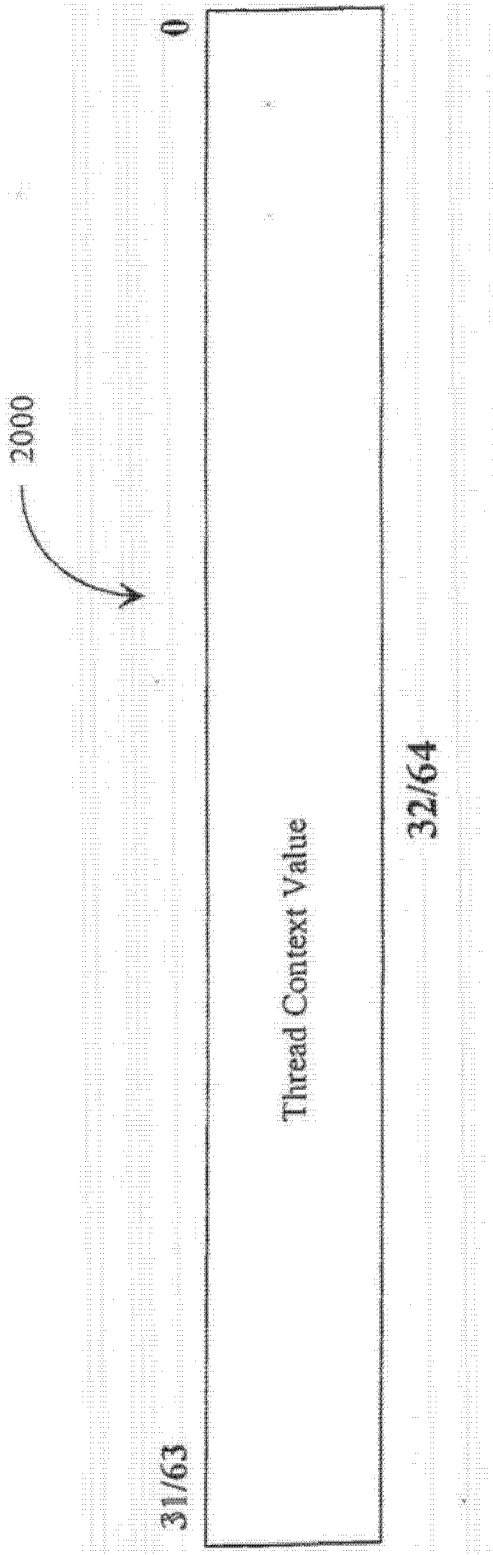


图 20

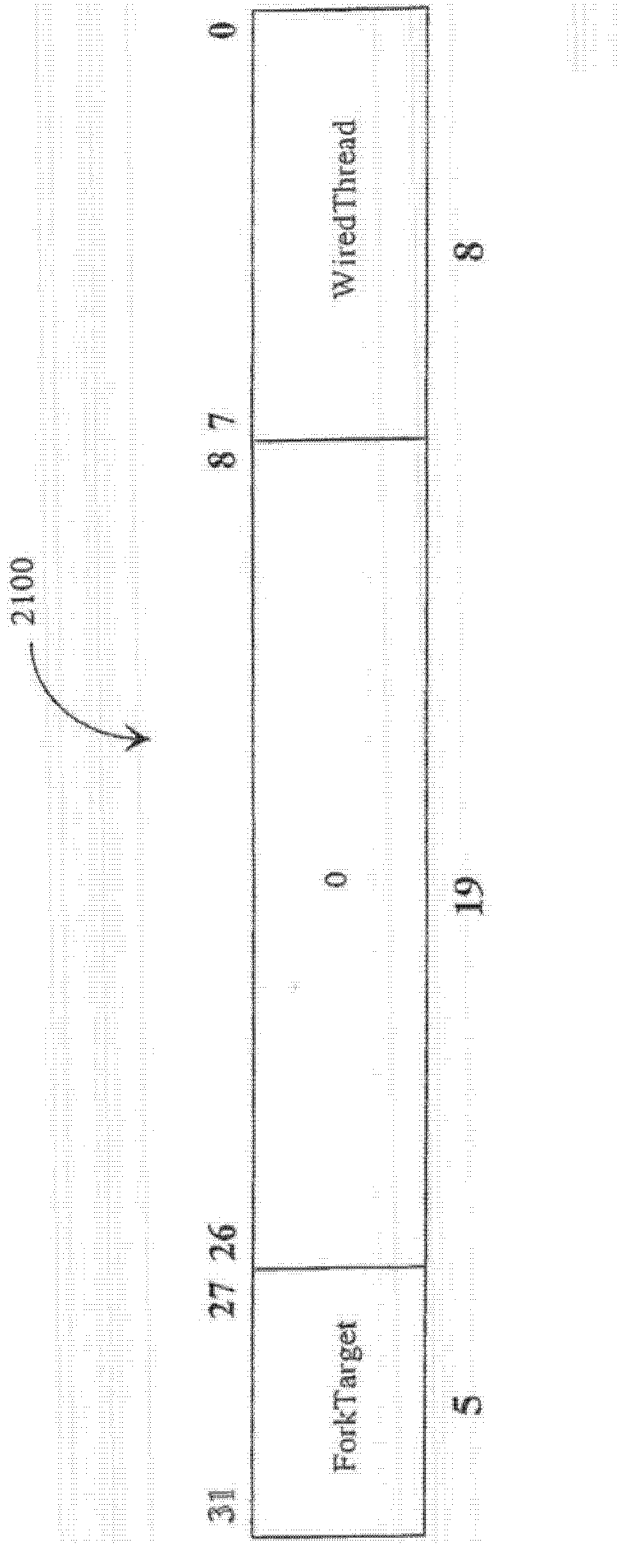


图 21

2200

字段		说明	读/写	重置状态
名称	位			
ForkTarget	31..27	在一个FORK指令中其GPR号码被使用为缓存器rt所指向的目的值。	R or R/W	预先设定或未定义
WiredThread	7:0	在动态配置的情况下，可取得的最低编号的执行绪的标号。	R/W	Undefined
0	26..8	必须被写入零。读取时为零。	0	0

图 22

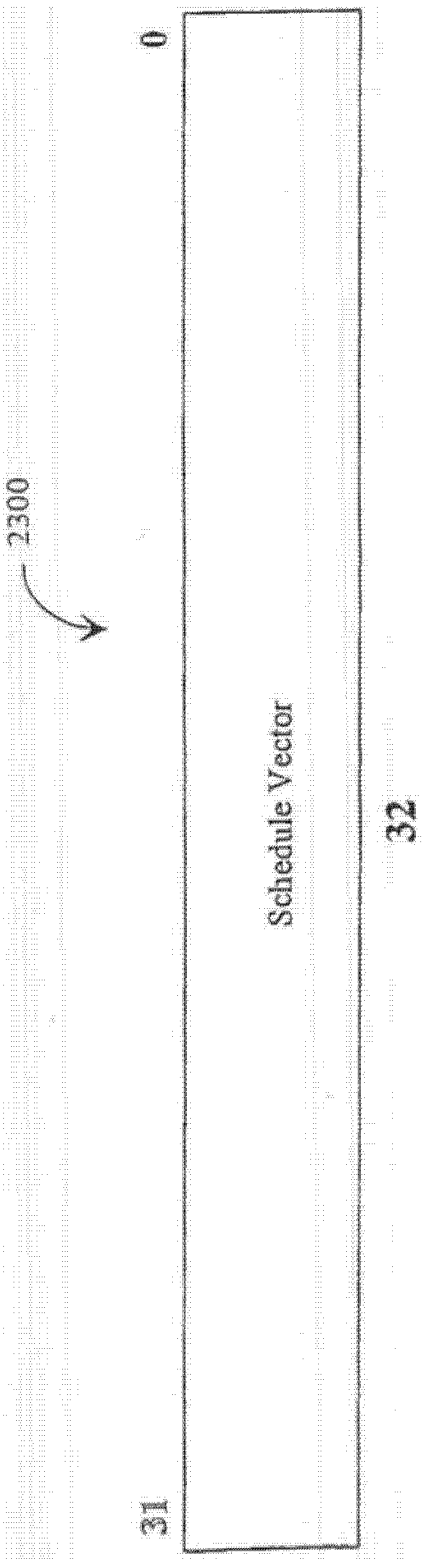


图 23

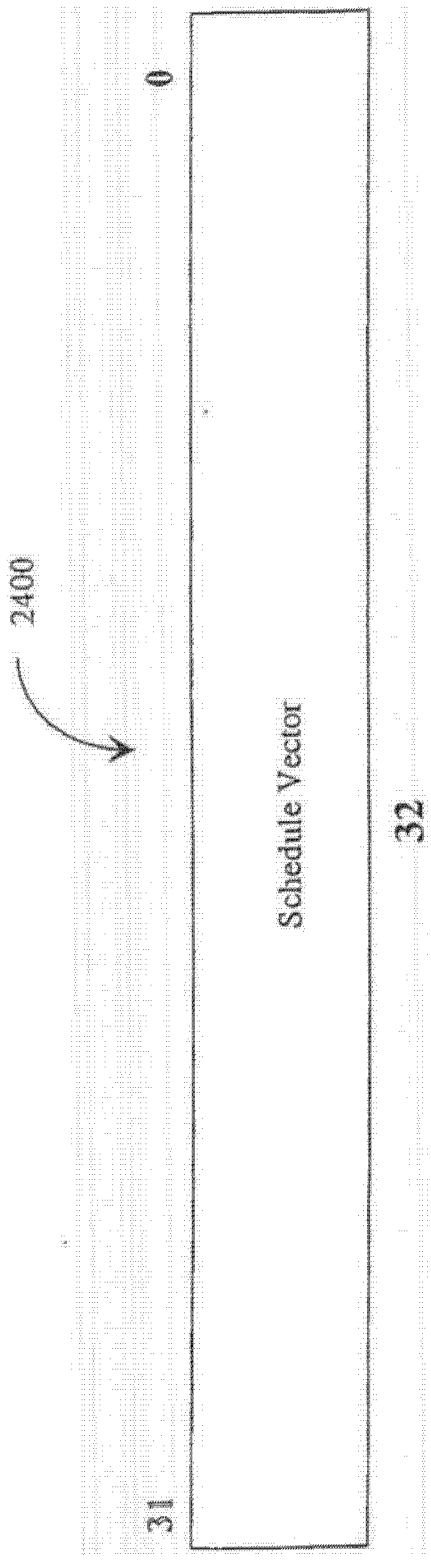


图 24

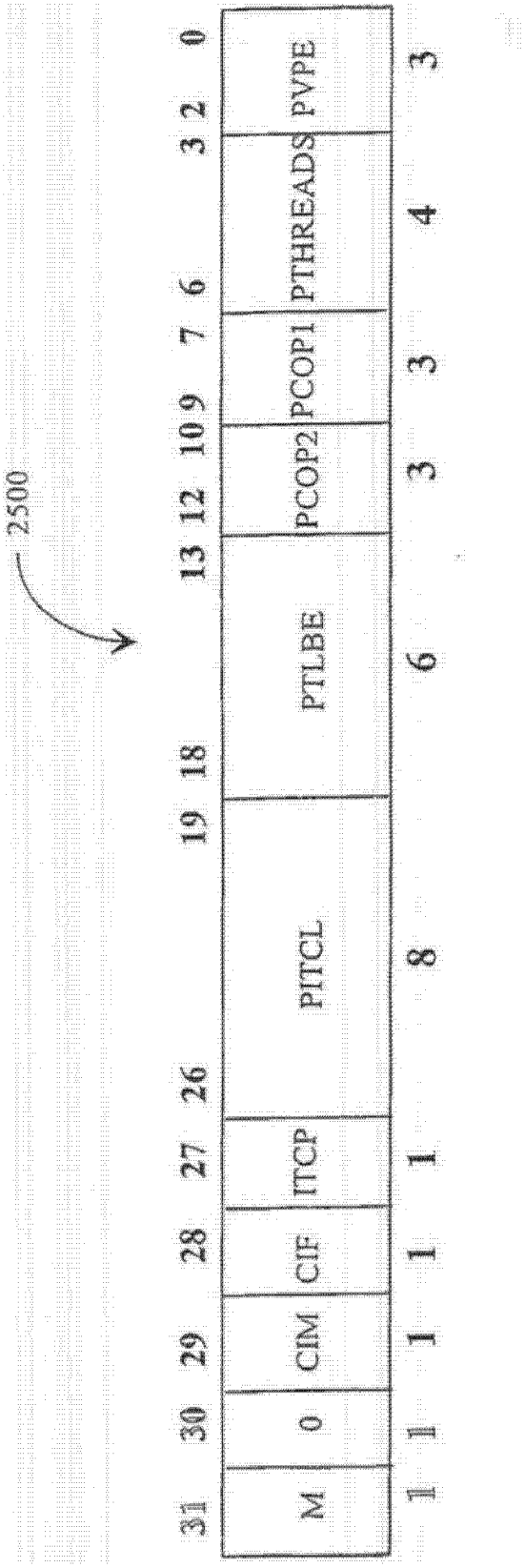
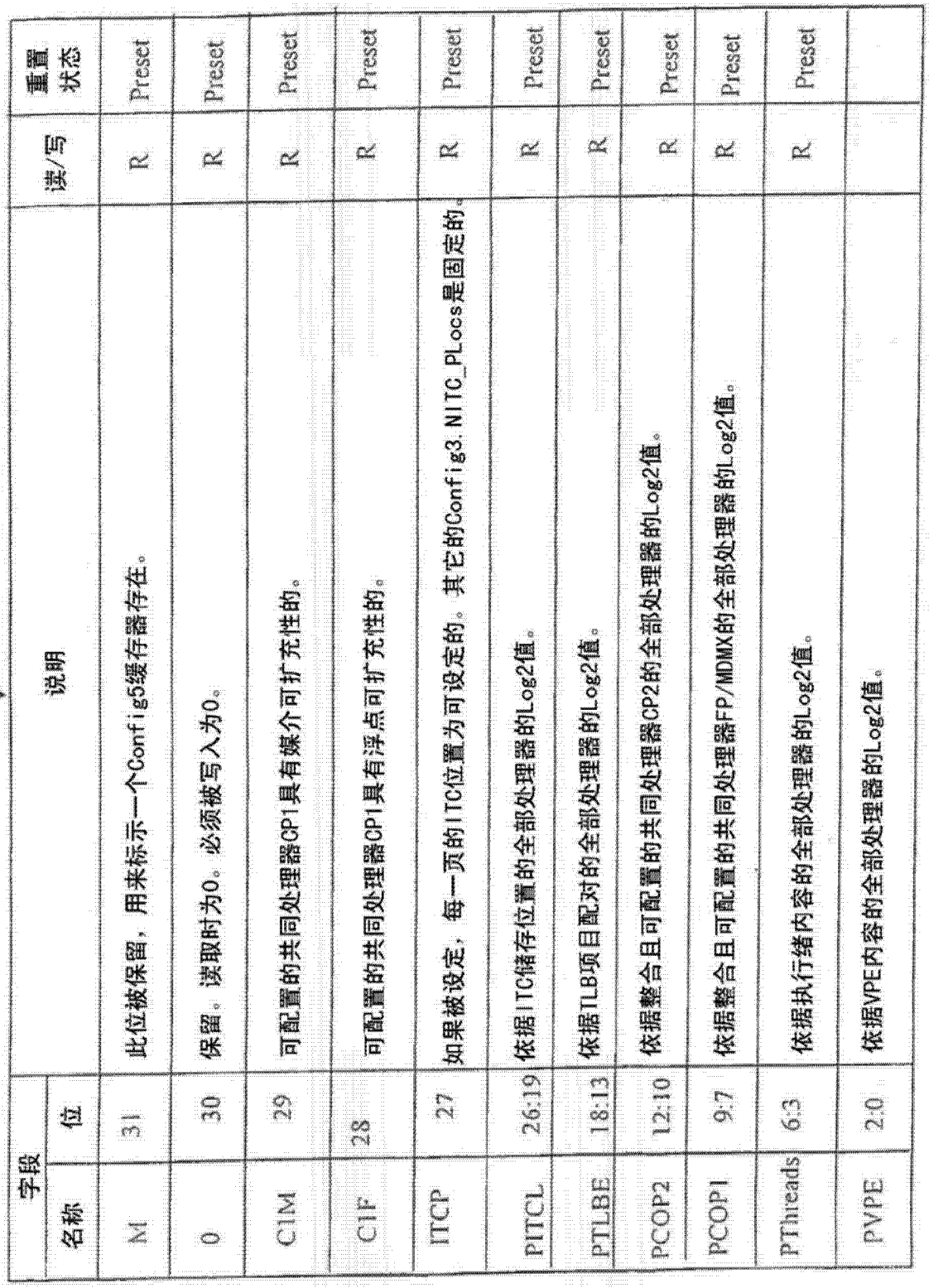

2500
M
0
CIM
CIF
ITCP
PITCL
PTLBE
PCOP2
PCOP1
PTHREADS
PVPE
31
30
29
28
27
26
19
18
13
12
10
9
7
6
3
2
0
1
1
1
1
1
8
6
3
3
4
3

图 25



2600

字段		说明	读/写	重置状态
名称	位			
M	31	此位被保留，用来标示一个Config5缓存器存在。	R	Preset
0	30	保留。读取时为0。必须被写入为0。	R	Preset
CIM	29	可配置的共同处理器CP1具有媒介可扩充性的。	R	Preset
CIF	28	可配置的共同处理器CP1具有浮点可扩充性的。	R	Preset
ITCP	27	如果被设定，每一页的ITC位置为可设定的。其它的Config3.NITC_PLocs是固定的。	R	Preset
PITCL	26:19	依据ITC储存位置的全部处理器的Log2值。	R	Preset
PTLBE	18:13	依据TLB项目配对的全部处理器的Log2值。	R	Preset
PCOP2	12:10	依据整合且可配置的共同处理器CP2的全部处理器的Log2值。	R	Preset
PCOP1	9:7	依据整合且可配置的共同处理器FP/MDMX的全部处理器的Log2值。	R	Preset
PThreads	6:3	依据执行绪内容的全部处理器的Log2值。	R	Preset
PVPE	2:0	依据VPE内容的全部处理器的Log2值。	R	

图 26

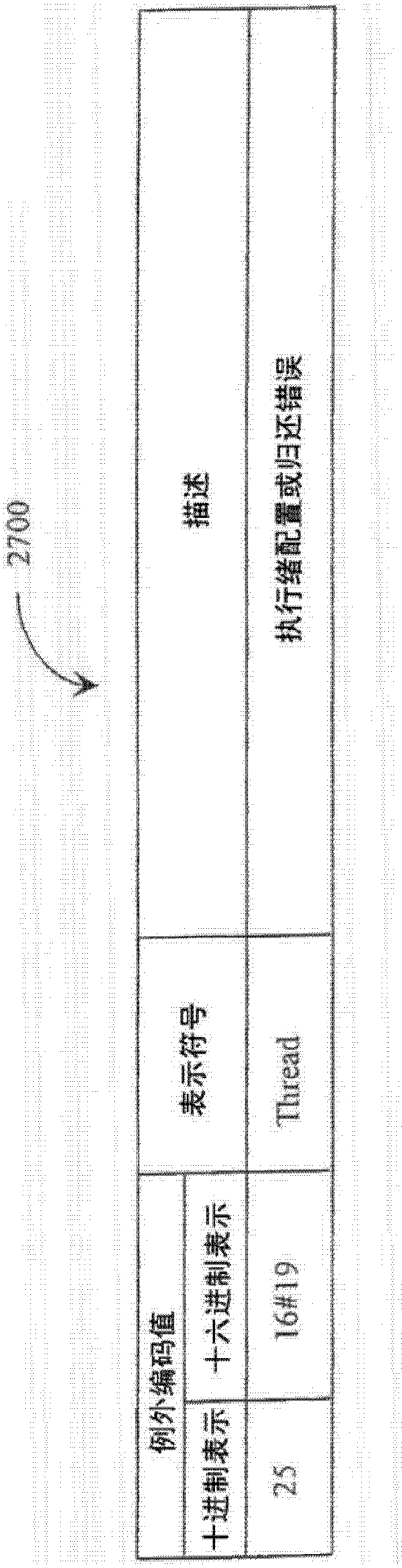

2700
例外编码值
十进制表示
十六进制表示
表示符号
描述
25
16#19
Thread
执行绪配置或归还错误

图 27

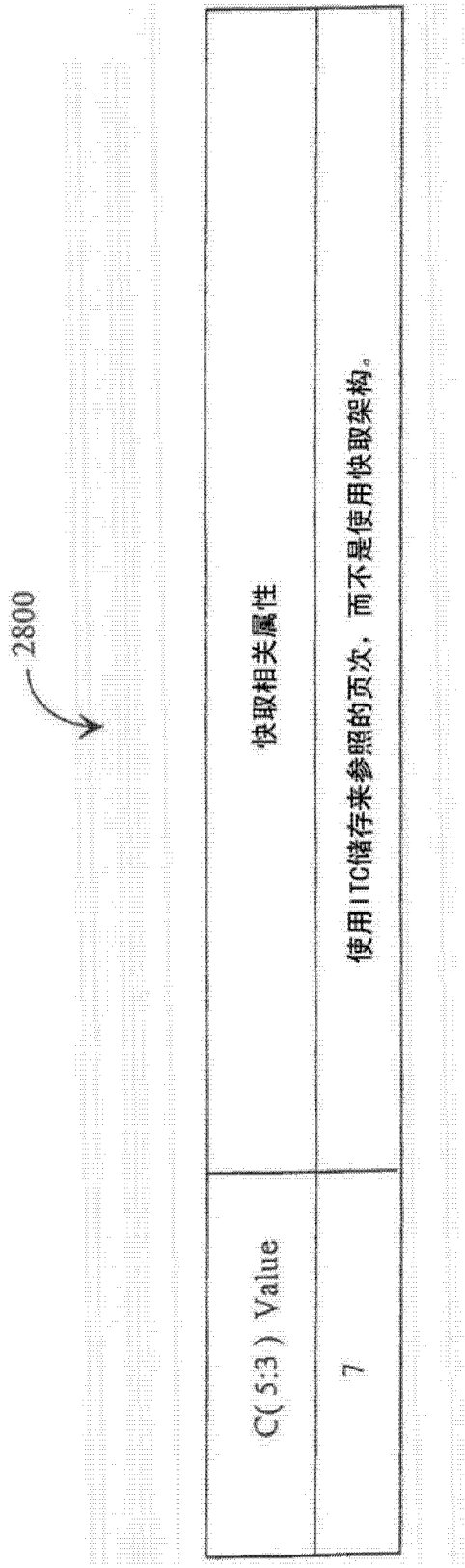
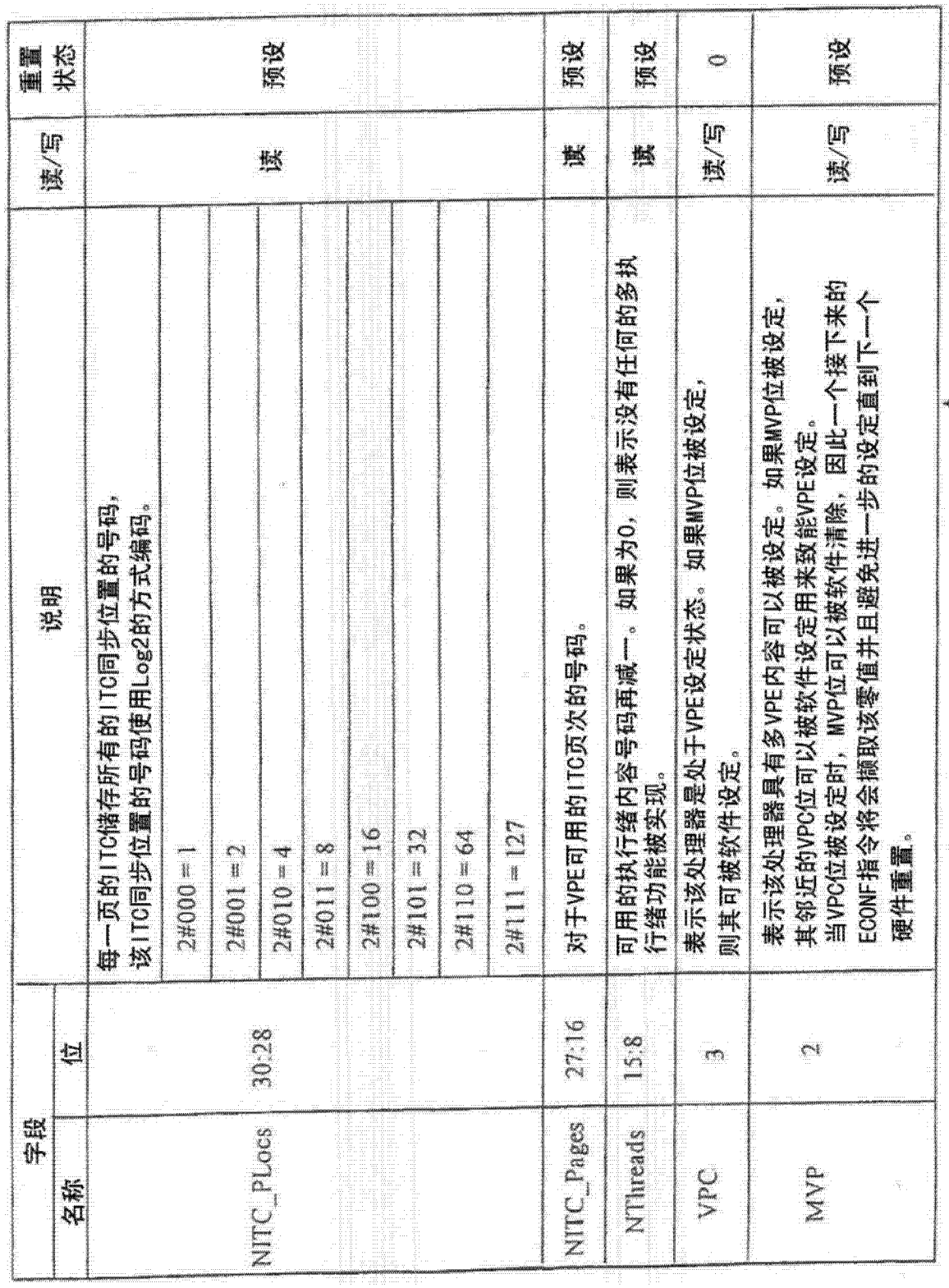

2800
C(5:3) Value
快取相关属性
7
使用ITC储存来参照的页次，而不是使用快取架构。

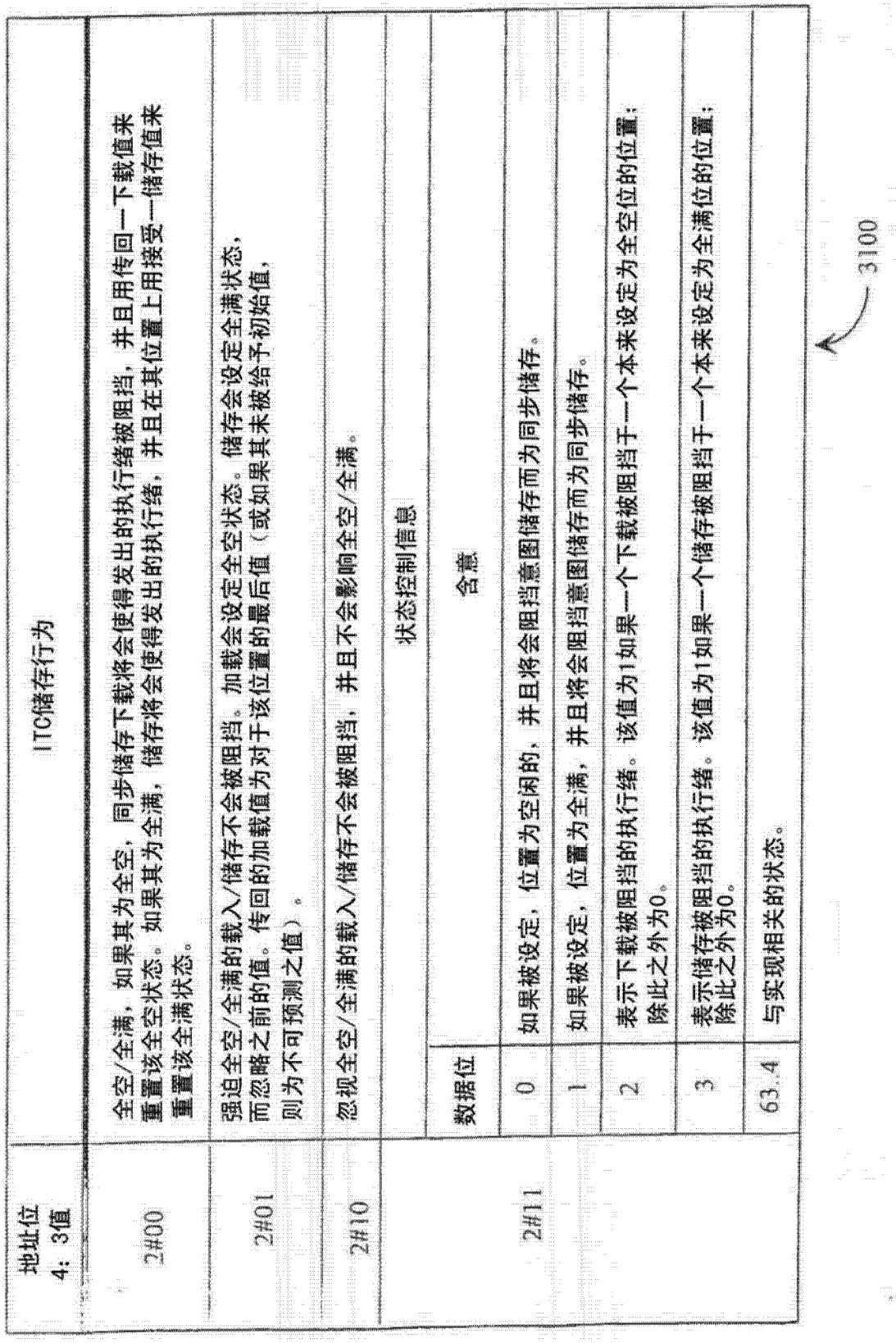
图 28



字段 名称	字段 位	说明	读/写	重置状态
NITC_PLocs	30:28	每一页的ITC储存所有的ITC同步位置的号码，该ITC同步位置的号码使用Log2的方式编码。 2#000 = 1 2#001 = 2 2#010 = 4 2#011 = 8 2#100 = 16 2#101 = 32 2#110 = 64 2#111 = 127	读	预设
NITC_Pages	27:16	对于VPE可用的ITC页次的号码。	读	预设
NThreads	15:8	可用的执行绪内容号码再减一。如果为0，则表示没有任何的多执行绪功能被实现。	读	预设
VPC	3	表示该处理器是处于VPE设定状态。如果MVP位被设定，则其可被软件设定。	读/写	0
MVP	2	表示该处理器具有多VPE内容可以被设定。如果MVP位被设定，其邻近的VPC位可以被软件设定用来致能VPE设定。 当VPC位被设定时，MVP位可以被软件清除，因此一个接下来的ECONF指令将会撷取该零值并且避免进一步的设定直到下一个硬件重置。	读/写	预设

2900

图 29



地址位 4：3值	ITC储存行为		
2#00	全空/全满，如果其为全空，同步储存下载将会使得发出的执行绪被阻挡，并且用传回一下载值来重置该全空状态。如果其为全满，储存将会使得发出的执行绪，并且在其位置上用接受一储存值来重置该全满状态。		
2#01	强迫全空/全满的载入/储存不会被阻挡。加载会设定全空状态。储存会设定全满状态，而忽略之前的值。传回的加载值为对于该位置的最后值（或如果其未被给予初始值，则为不可预测之值）。		
2#10	忽视全空/全满的载入/储存不会被阻挡，并且不会影响全空/全满。		
2#11	状态控制信息		
	数据位	含意	
	0	如果被设定，位置为空闲的，并且将会阻挡意图储存而为同步储存。	
	1	如果被设定，位置为全满，并且将会阻挡意图储存而为同步储存。	
	2	表示下载被阻挡的执行绪。该值为1如果一个下载被阻挡于一个本来设定为全空位的位置；除此之外为0。	
	3	表示储存被阻挡的执行绪。该值为1如果一个储存被阻挡于一个本来设定为全满位的位置；除此之外为0。	
	63..4	与实现相关的状态。	

3100

图 31

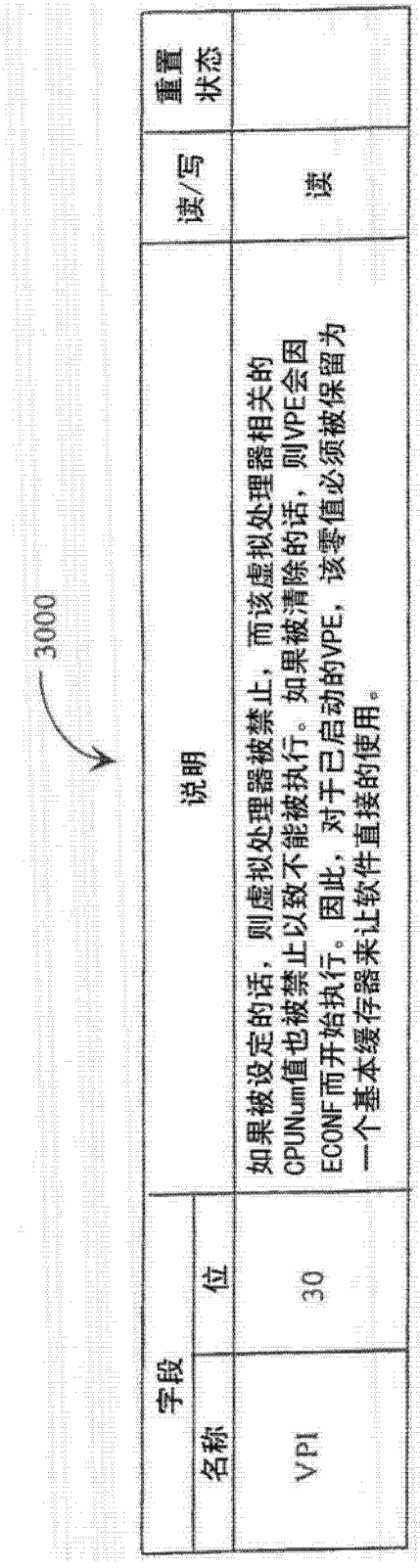

3000
字段
名称
位
VPI
30
说明
如果被设定的话，则虚拟处理器被禁止，而该虚拟处理器相关的CPUNum值也被禁止以致不能被执行。如果被清除的话，则VPE会因ECONF而开始执行。因此，对于已启动的VPE，该零值必须被保留为一个基本缓存器来让软件直接的使用。
读/写
读
重置状态

图 30

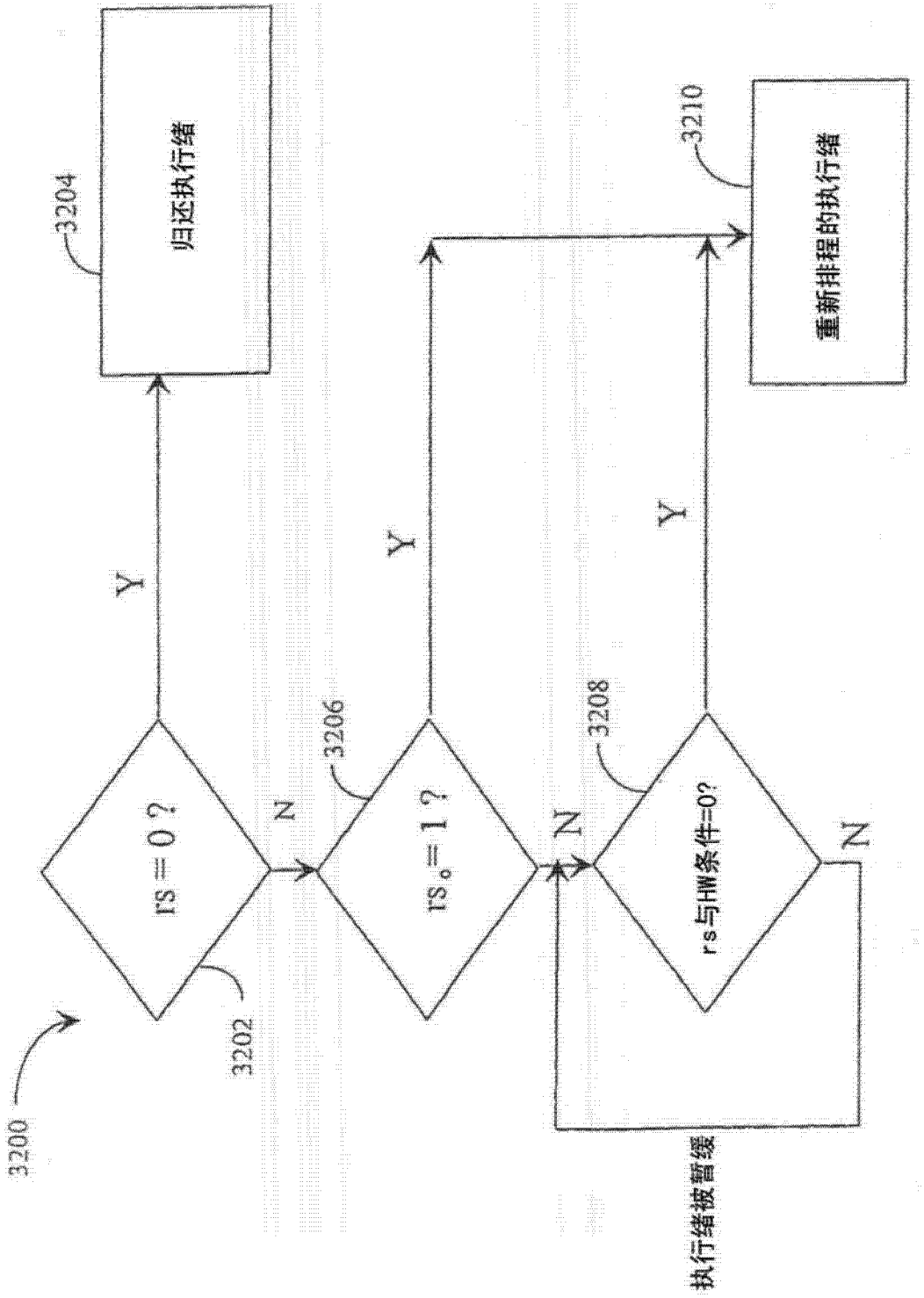

3200
3202
rs = 0 ?
Y
3204
归还执行绪
N
3206
rs。= 1 ?
Y
3210
重新排程的执行绪
N
3208
rs与HW条件=0?
Y
N
执行绪被暂缓

图 32

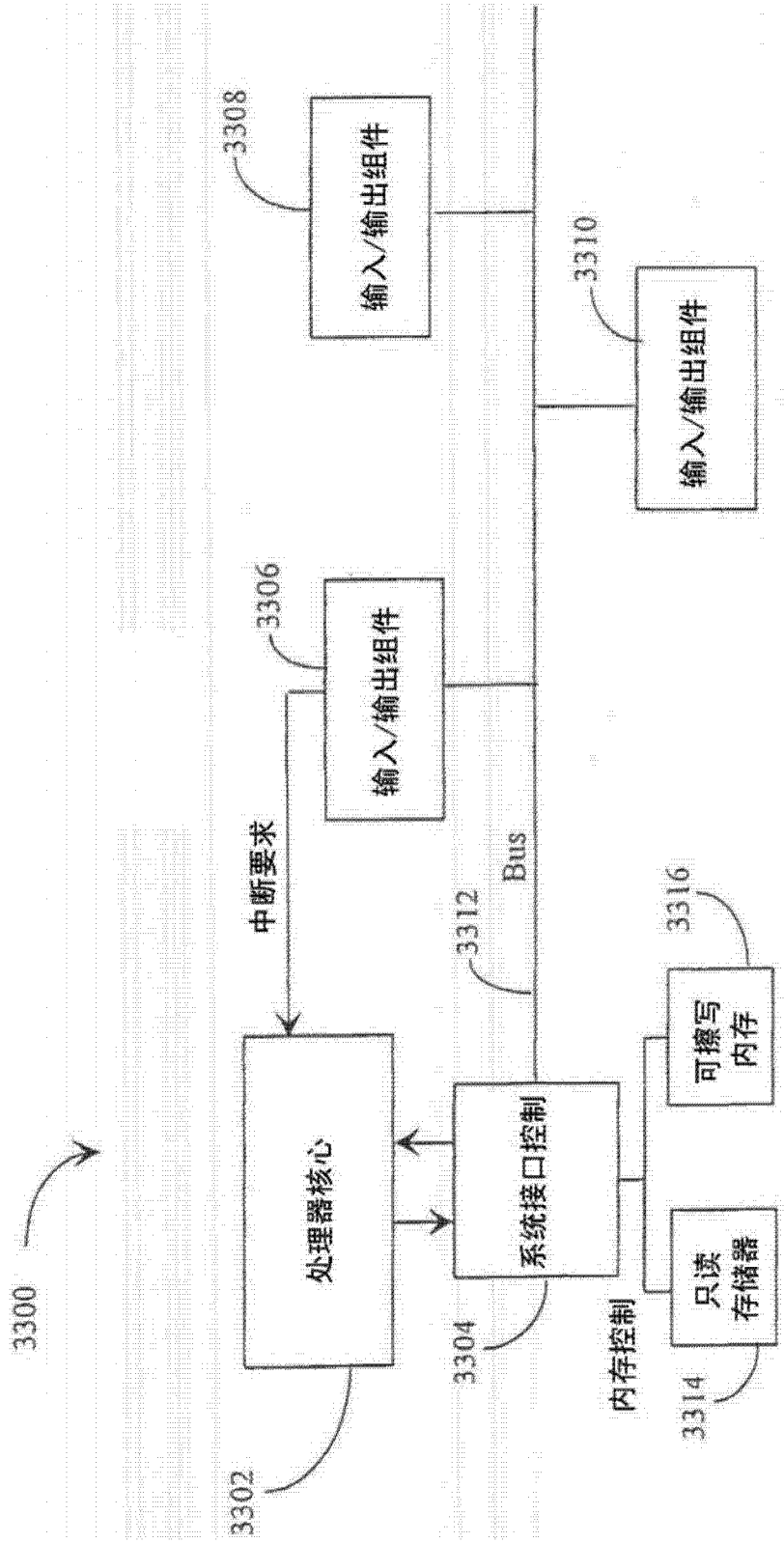

3300
处理器核心
3302
系统接口控制
3304
Bus
3312
中断要求
输入/输出组件
3306
输入/输出组件
3308
输入/输出组件
3310
内存控制
只读存储器
3314
可擦写内存
3316

图 33

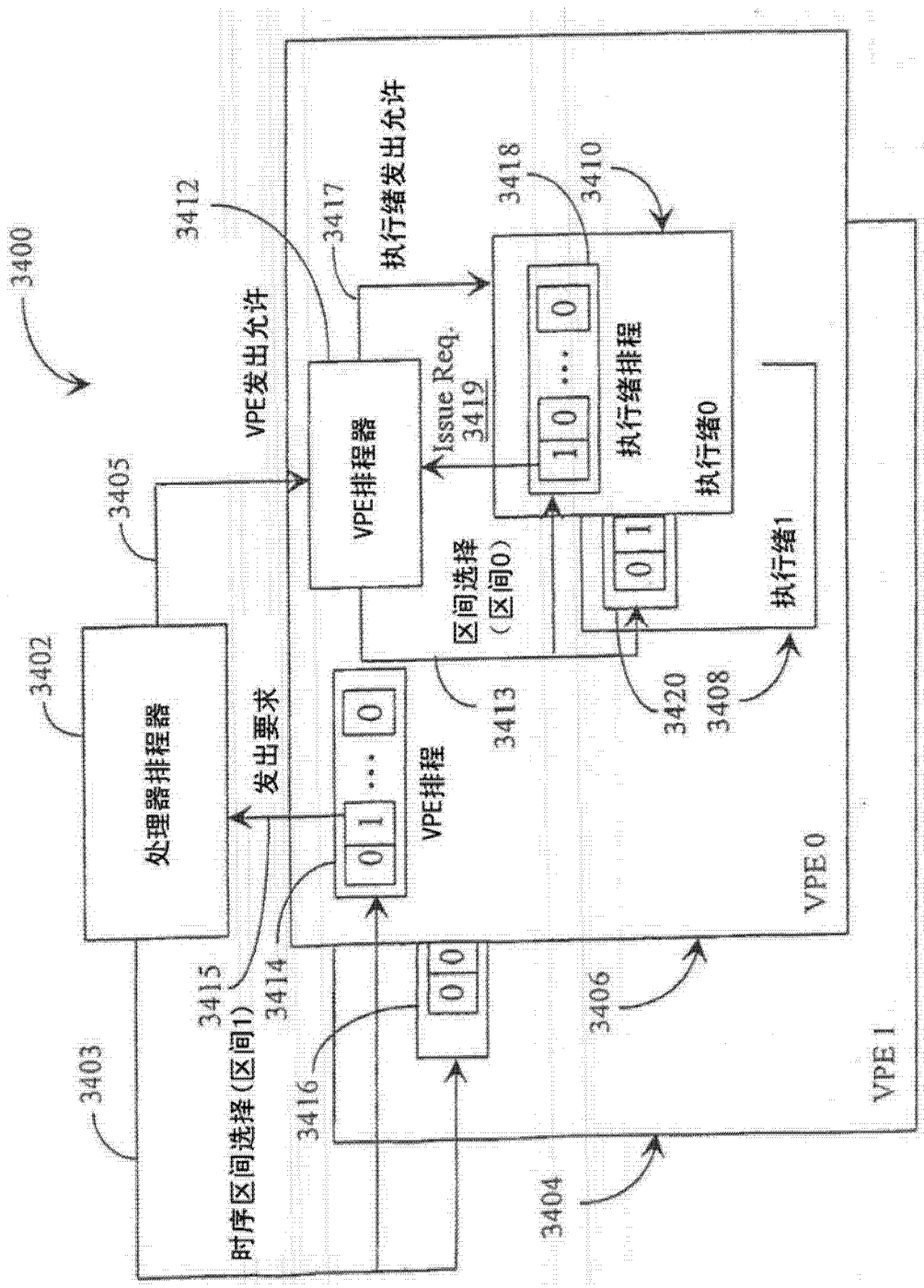

3400
3412
3417
执行绪发出允许
3418
3410
VPE发出允许
3405
VPE排程器
Issue Req.
3419
执行绪排程
执行绪0
执行绪1
区间选择
（区间0）
3402
处理器排程器
发出要求
3413
3420
3408
VPE排程
VPE 0
3415
3414
时序区间选择（区间1）
3416
3406
3403
VPE 1
3404

图 34