(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 104536723 A
(43)申请公布日 2015.04.22

(21)申请号 201410827036.7

(22)申请日 2008.06.27

(30)优先权数据
12/147,332 2008.06.26 US

(62)分案原申请数据
200880014972.9 2008.06.27

(71)申请人 拉塞尔·H·菲什
地址 美国得克萨斯州

(72)发明人 拉塞尔·H·菲什

(74)专利代理机构 北京英赛嘉华知识产权代理有限责任公司 11204
代理人 余朦 王艳春

(51)Int.Cl.
G06F 9/30(2006.01)

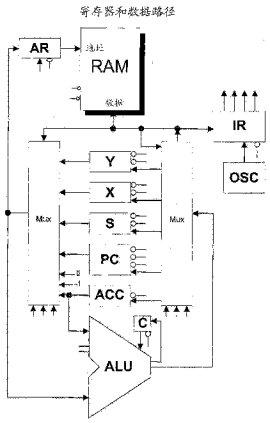
权利要求书1页 说明书24页 附图34页

(54)发明名称

线程优化的多处理器架构

(57)摘要

在一个方面，本发明包括一种系统，该系统包括：(a)位于单个芯片上的多个并行处理器；以及(b)计算机存储器，位于芯片上并且被处理器中的每个访问；处理器中的每个被操作为处理最小指令集，处理器中的每个包括专用于处理器中的至少三个特定寄存器中的每个的本地高速缓冲存储器。在另一方面，本发明包括一种系统，该系统包括：(a)位于单个芯片上的多个并行处理器；以及(b)计算机存储器，位于芯片上并且被处理器中的每个访问，处理器中的每个被操作为处理被优化用于线程级并行处理的指令集，每个处理器访问芯片上的计算机存储器的内部数据总线，内部数据总线的宽度是存储器的一行的宽度。



CN 104536723 A

1. 一种通过半导体制造工艺制备的线程优化多处理器，所述半导体制造工艺通过以下步骤执行：

在至少一个芯片上将少于 16 层的金属互连；

将至少一个处理器嵌入所述至少一个芯片中；以及

将所述至少一个芯片安装在双列直插内存模块上。

2. 一种通过半导体制造工艺制备的线程优化多处理器，所述半导体制造工艺通过以下步骤执行：

在至少一个芯片上将少于 8 层的金属互连；

将至少一个处理器嵌入所述至少一个芯片中；以及

将所述至少一个芯片安装在双列直插内存模块上。

3. 一种通过半导体制造工艺制备的线程优化多处理器，所述半导体制造工艺通过以下步骤执行：

在至少一个芯片上将少于 4 层的金属互连；

将至少一个处理器嵌入所述至少一个芯片中；以及

将所述至少一个芯片安装在双列直插内存模块上。

线程优化的多处理器架构

[0001] 本申请为题为“线程优化的多处理器架构”的中国专利申请的分案申请，该中国专利申请的申请号为200880014972.9，申请日为2008年6月27日。

技术领域

[0002] 本申请涉及线程优化的多处理器架构，更具体地涉及通过半导体制造工艺制备的线程优化多处理器。

背景技术

[0003] 可使用两种一般的方法来加快计算机速度：加快指令执行速度或以并行的方式执行更多的指令。由于指令执行速度已接近硅电子迁移率的极限，因而并行操作成为加快计算机速度的最佳可选方式。

[0004] 并行操作的早期尝试包括：

[0005] 1. 将取下一条指令与执行当前指令进行重叠。

[0006] 2. 指令流水线操作。指令流水线将每一条指令拆分为尽可能多的片段，然后尝试将连续的指令映射到并行的执行单元内。由于多步指令的无效性，很多软件程序不能提供足够的连续指令以使并行执行单元保持填满，并且在遇到分支结构、循环结构或判断结构时，要花费大量时间来重新填充执行单元，因而很难获得理论上的最高限度的改进。

[0007] 3. 单指令多数据或SIMD。该类技术是在因特尔SSE指令集中发现的，如在因特尔奔腾3或其它处理器中实现。在该技术中，在多个数据集上执行单一的指令。该技术只对例如视频图像绘制的特殊应用有帮助。

[0008] 4. 超立方体结构。该技术使用处理器的大的二维阵列（有时使用三维阵列）和本地存储器。支持处理器阵列所需的通信和互连固有地将它们限制在特殊应用中。

[0009] 流水线是由连续地完成指令片段的执行的多个连续的阶段组成的指令执行单元，所述执行例如取指令、译码、执行、存储等。多个流水线可并行地放置，以将程序指令连续地供给每条流水线，直到所有的流水线都在执行指令。然后从最初的流水线开始重复进行指令填充。当N条流水线都填充了指令并且都在执行时，其性能效果在理论上是单执行单元的执行速度的N倍。

[0010] 成功的流水线取决于如下方面：

[0011] 1. 指令的执行必须能够定义为几个连续的状态。

[0012] 2. 每条指令必须具有相同数目的状态。

[0013] 3. 每条指令的状态数目确定并行执行单元的最大数目。

[0014] 由于基于并行流水线的数目，流水线操作能够获得性能提高，且由于并行流水线的数目由每条指令的状态数目所确定，因此流水线鼓励复杂的多状态指令。

[0015] 过度流水线化的计算机很难获得接近并行执行单元期望的理论性能改进。

[0016] 该流水线损失的一些原因包括：

[0017] 1. 软件程序并不只由连续的指令构成。各种研究表明每8-10条指令就会发生执

行流的变化。改变程序流的任何分支都将会扰乱流水线。将流水线的扰乱降低到最小的尝试是很复杂的且是不完全的。

[0018] 2. 强迫所有的指令包含相同数目的状态常常导致执行满足最小公分母（即，最慢和最复杂）指令的需求的流水线。由于该流水线的原因，使得不管是否需要，所有的指令都被迫具有相同数目的状态。例如，逻辑操作（例如为 AND（与）或 OR（或））的执行比 ADD（加）的执行要快一个数量级，但是常常为两者分配了相同数量的执行时间。

[0019] 3. 流水线鼓励多状态复杂指令。可能需要两个状态的指令通常被延长以填充 20 个状态，就因为该流水线的深度是 20。（因特尔奔腾 4 使用 20 个状态的流水线。）

[0020] 4. 除了特殊状态的设计余量或冗余以外，每个流水线状态所需的时间还必须考虑到通过逻辑电路和相关晶体管的传播延迟。

[0021] 5. 由于仲裁逻辑内的晶体管的传播延迟，常常会降低流水线寄存器和其它资源存取的仲裁性能。

[0022] 6. 在附加的状态实际上会减慢执行速度而不是加快速度之前，指令可被拆分为的状态的数目具有上限。一些研究表明，Digital Equipment Corporation（数字装备公司）的 α 处理器的最后一代的流水线架构超出了该上限，而实际上比先前的较短流水线的处理器版本执行得更慢。

[0023] 拆分流水线

[0024] 重新分解 CPU 设计的观点之一是考虑将流水线执行单元拆分为多个（N 个）简化的处理器。（在这样的设计中，寄存器和一些其它逻辑可需要复制。）相对于上述流水线架构，N 个简化的处理器中的每个处理器都将具有如下的优点：

[0025] 1. 不会发生流水线停止。无分支预测的必要性。

[0026] 2. 指令能按所需时间执行，而不是全部都被分配给与最慢指令相同的执行时间。

[0027] 3. 可通过减少必要的执行状态来简化指令，从而减少流水线损失。

[0028] 4. 从流水线中去除的每个状态都能消除传播延迟，并能去除该状态所需的设计余量。

[0029] 5. 能消除寄存器仲裁。

[0030] 而且，相对于流水线化的 CPU，具有 N 个简化处理器的系统还具有如下优点：

[0031] 1. 消除最大流水线并行操作的限制。

[0032] 2. 不同于流水线化的处理器，可选择性地关闭多个单独处理器的电源，以在不使用这些处理器时降低功耗。

[0033] 当前并行操作方法的其它问题

[0034] 并行操作的很多实现屈服于阿姆达尔定律 (Amdahl's Law) 的限制。

[0035] 由于该问题的不可串行化部分，并行操作的加速会受到系统开销的限制。基本上，当并行操作的数量增加时，支持这种数量增加所需的通信会超过并行操作带来的益处。

[0036] 红线上的停止信号 (Stoplight Sitting at Redline)

[0037] 当前处理器的另一低效率是不能将计算功率调整到满足即时计算的需求。大多数计算机将其大部分时间花费在等待某些事发生。其等待 I/O、等待下一条指令、等待存储器存取、或有时等待人机接口。这种等待是计算功率的低效浪费。而且，花费在等待上的计算机时间常常导致功耗的增加和热量的产生。

[0038] 等待规则的例外是诸如引擎控制器、信号处理器和防火墙路由器等应用软件。由于问题集和解决方法集的预定性质，因此这些应用软件是用于并行操作加速的卓越候选者。使用 N 个乘法器可更快地解决需要 N 次独立乘法的乘积的问题。

[0039] 通用计算机的感知性能实际上已达到其峰点。通用计算机变得越来越忙，这是由于运行快速屏幕刷新的视频游戏、编译大的源文件或搜索数据库。在理想世界中，视频绘制将被分解为特殊用途、阴影、变换、绘制硬件。将程序分解为特殊用途硬件的一种方法是使用“线程”。

[0040] 线程是自含式的、很少与其它线程进行数据通信的独立程序。线程通常用于从低速实时活动收集数据，并提供收集结果。线程也可用于在显示器上绘制变化。线程在需要进一步与另一线程交互之前，可变换数千或数百万个状态。独立线程通过并行操作为提高性能提供了机会。

[0041] 许多软件编译器都支持线程的产生和管理，用于对软件设计过程进行分解。通过优选实施方式中的线程优化微处理器 (TOMI) 中实现的线程级并行操作技术，相同的分解将支持多 CPU 并行处理。

[0042] 线程级并行操作

[0043] 线程是用于在单 CPU 上分解软件程序的非常容易理解的技术。通过使用 TOMI 处理器，线程级并行操作能获得程序加速。

[0044] 与其它并行方法相比，TOMI 处理器的一个重要优点是，TOMI 处理器只需对当前软件编程技术做最小的改变，而不需要开发新算法。许多现存的程序需要重新编译，但是基本上不用重写。

[0045] 有效的 TOMI 计算机架构应该构建在大量简化处理器的周围。不同的架构可用于不同类型的计算问题。

[0046] 基本计算机操作

[0047] 对于通用计算机，最通用的操作按照频率减小的顺序排列是 ：加载和存储 ；排序 ；以及数学和逻辑。

[0048] 加载和存储

[0049] 加载 (LOAD) 和存储 (STORE) 的参数是源和目的地。加载和存储的能力是源和目的地的范围 (例如，4G 字节比 256 字节具有更有力的范围)。相对于当前源和目的地的位置对许多数据集来说是很重要的。加 1、减 1 是最有用的。

[0050] 增加从当前源和目的地的偏移量越来越不重要了。

[0051] 加载和存储也可受存储器分级体系所影响。从存储器加载可能是 CPU 可执行的最慢操作。

[0052] 排序

[0053] 分支和循环是基本的排序指令。根据测试改变指令顺序是计算机做出判定的方法。

[0054] 数学和逻辑

[0055] 数学和逻辑操作是三种操作中最少使用的。逻辑操作是 CPU 可执行的最快操作，并可仅仅需要单逻辑门延迟。由于高比特位依赖于低比特位操作的结果，因而数学操作更复杂。即使提前进位，32 位相加 (ADD) 也可需要至少 32 个门延迟。使用移位相加技术的相

乘 (MULTIPLY) 可等价于需要 32 个相加。

[0056] 指令大小的折衷

[0057] 完美指令集可由大到足以选择无限可能的源、目的地、操作和下一条指令的操作码组成。不幸地，该完美指令集的操作码将会无限宽，因而指令带宽将会是零。

[0058] 高指令带宽的计算机设计包括创建这样的具有操作码的指令集，其能够利用最少的操作码位有效地定义最通用的源、目的地、操作和下一条指令。

[0059] 宽的操作码导致高的指令总线带宽需求，且该得到的架构将会很快受限于冯·诺伊曼瓶颈，其中，计算机性能受到从存储器取指令的速度的限制。

[0060] 如果存储器总线是 64 位宽，就能在每个存储器周期取单条 64 位指令、两条 32 位指令、四条 16 位指令或八条 8 位指令。32 位指令的有用性最好为 16 位指令的两倍，这是因为 32 位指令将指令带宽减少为一半。

[0061] 指令集设计的主要目标是减少指令冗余。一般地，优化的高效指令集利用指令和数据的位置。最简单的指令优化实现已很久了。对于大多数计算机程序，最可能的下一条指令为存储器中按顺序的下一条指令。因此不用在每条指令中包含下一条指令的字段，而是大多数指令都假设下一条指令是当前指令 +1。创建零比特源和零比特目的地的架构是可能的。

[0062] 堆栈架构

[0063] 堆栈架构计算机也称为零操作数架构。堆栈架构根据后进先出堆栈的内容执行所有的操作。双操作数操作需要两个操作数都出现在堆栈上。当操作执行时，两个操作数都将从堆栈弹出 (POP)，执行该操作，然后将结果压入 (PUSH) 到堆栈中。由于源和目的地已隐含地位于堆栈中，因而堆栈架构计算机可具有非常短的操作码。

[0064] 大多数程序需要全局寄存器的内容，在需要该寄存器时，其在堆栈上可能并不总是可用的。将该事件的发生概率最小化的尝试包括为堆栈编索引，从而允许对不位于堆栈顶部的操作数进行存取。为堆栈编索引要么需要额外的操作码位从而导致更大的指令，要么需要额外的操作以将堆栈索引值放置在堆栈本身。某些时候，定义了一个或多个额外的堆栈。更好的但非最优化的解决方法是堆栈 / 寄存器架构的结合。

[0065] 堆栈架构操作也常常明显不符合最优化地造成冗余。例如，当堆栈在存储器内操作时，每次弹出和压入操作都可能造成浪费时间的存储器操作。而且，堆栈操作可消耗下一个操作中立即需要的操作数，从而需要进行操作数复制，并可能需要另一个存储器操作。以将一维数组中的所有元素乘以 15 的操作为例进行说明。

[0066] 在堆栈架构上，通过如下操作实现：

[0067] 1. 压入数组的起始地址

[0068] 2. 复制 (DUPLICATE) 地址（因此得到数组中将用于存储结果的地址。）

[0069] 3. 复制地址（因此得到数组中将用于进行读取的地址。）

[0070] 4. 间接压入 (PUSH INDIRECTLY)（将堆栈顶端所指的数组位置的内容压入）

[0071] 5. 压入 15

[0072] 6. 乘法 (MULTIPLY)（将第 3 步所读的数组内容进行 15 次乘法。）

[0073] 7. 交换 (SWAP)（从堆栈的顶部得到数组地址，以用于下一条指令。）

[0074] 8. 间接弹出 (POP INDIRECTLY)（弹出乘法结果并将其存储回数组）

[0075] 9. 递增 (INCREMENT)(指向下一个数组元素。)

[0076] 10. 转到步骤 2 直到数组完成。

[0077] 第 9 步的循环计数需要额外的参数,在某些架构中,该参数存储在另一堆栈中。

[0078] 在假定的寄存器 / 累加器架构中,该实施例实现如下 :

[0079] 1. 存储指向数组的起始位置的指针

[0080] 2. 读取指针 (将该指针指向的地址的内容读取到累加器中)

[0081] 3. 乘以 15

[0082] 4. 存储指针 (将结果存储到该指针指向的地址)

[0083] 5. 递增指针

[0084] 6. 转到第 2 步直到数组完成。

[0085] 比较上述实施例中用于堆栈架构的九个步骤和用于寄存器架构的五个步骤。而且,堆栈操作包括至少 3 次用于额外存储器存取的可能机会。假定的寄存器 / 累加器架构的循环控制能够简单地在寄存器中被处理。

[0086] 堆栈对于对表达式求值是很有用的,且因而用于大多数编译器中。堆栈对于嵌套操作 (例如为函数调用) 也是很有用的。大多数 C 编译器都用堆栈实现函数调用。然而,如果没有通用存储的补充,堆栈架构则需要大量的额外数据转移和处理。为了优化的目的,堆栈的压入和弹出操作还应该与数学和逻辑操作相分离。但是从上面的实施例可见,当重复加载和存储数据时,由于数组地址由间接压入和间接弹出而消耗,因此堆栈尤其效率低。

发明内容

[0087] 一方面,本发明包括一种系统,其包括 :(a) 位于单片芯片上的多个并行处理器 ;以及 (b) 计算机存储器,其位于芯片上并能够被每个处理器访问 ;其中,每个处理器都能够操作以处理最小 (de minimis) 指令集,并且,每个处理器都包括本地高速缓冲存储器,本地高速缓冲存储器用于处理器内的至少三个专用寄存器中的每个。

[0088] 在不同的实施方式中 :(1) 每个本地缓冲存储器的大小等于芯片上的随机存取存储器的一行 ;(2) 具有相关高速缓冲存储器的至少三个专用寄存器包括指令寄存器、源寄存器和目的寄存器 ;(3) 最小指令集包括七条基本指令 ;(4) 每个处理器都能够操作以处理单线程 ;(5) 累加器是用于除了递增指令外的每个指令的操作数 ;(6) 每条基本指令的目的地总是操作数寄存器 ;(7) 三个寄存器自动递增且三个寄存器自动递减 ;(8) 指令集不包括分支指令和跳转指令 ;(9) 每条指令的长度最多为 8 比特 ;(10) 单一的主处理器负责管理每个并行处理器。

[0089] 另一方面,本发明还包括一种系统,其包括 :(a) 位于单片芯片上的多个并行处理器 ;以及 (b) 计算机存储器,其位于芯片上,并能够被每个处理器访问 ;其中,每个处理器都能够操作以对指令集进行处理,指令集被优化用于线程级并行处理。

[0090] 在不同的实施方式中 :(1) 每个处理器都能够操作以处理最小指令集 ;(2) 每个处理器都包括本地高速缓冲存储器,高速缓冲存储器用于处理器中至少三个专用寄存器中的每个 ;(3) 每个本地高速缓冲存储器的大小等于芯片上的随机存取存储器的一行 ;(4) 至少三个专用寄存器包括指令寄存器、源寄存器和目的寄存器 ;(5) 最小指令集包括七条基本指令 ;(6) 每个处理器都能够操作以处理单线程 ;(7) 单一的主处理器负责管理每个并行处

理器；以及 (8) 最小指令集包括指令扩展的最小集合以优化处理器操作和促进软件编译器的效率。

[0091] 在另一实施方式中，本发明还包括一种线程级并行处理方法，其中，方法利用单一芯片上的主处理器、多个并行处理器以及计算机存储器实现，其中多个处理器中的每个都能够操作以处理最小指令集以及处理单线程，方法包括：(a) 为多个处理器的每个处理器中的三个专用寄存器中的每个寄存器分配本地高速缓冲存储器；(b) 将多个处理器中一个分配以处理单线程；(c) 由处理器处理每个分配的线程；(d) 由处理器处理来自每个线程的结果；(e) 在线程处理完成之后，对多个处理器的中的一个进行解分配；以及 (f) 最小指令集包括指令的最小集合以优化处理器管理。

[0092] 在不同的实施方式中，最小指令集包括七个基本指令，最小指令集中的指令的长度最大为 8 位。最小指令集还可包括除了七个基本指令以外的一组扩展指令，这组扩展指令优化 TOMI CPU 的内部操作、有助于优化正由 TOMI CPU 执行的软件程序指令的执行并且优化用于 TOMI CPU 的软件编译器的操作。具有多个 TOMI CPU 核的本发明实施方式还可包括用于管理多个 CPU 核的一组有限的处理器管理指令。

[0093] 在另一方面，本发明包括一种系统，该系统包括：(a) 多个并行处理器，安装在存储器模块上；(b) 外部存储器控制器；以及 (c) 通用中央处理单元，并行处理器中的每个被操作为处理被优化用于线程级并行处理的指令。

[0094] 在不同的实施方式中，(1) 并行处理器中的每个被操作为处理最小指令集；(2) 在存储器模式寄存器中分配的一个或多个位被操作为启用或禁用并行处理器中的一个或多个；(3) 存储器模块是双列直插内存模块；(4) 处理器中的每个被操作为处理单线程；(5) 多个线程通过共享的存储器共享数据；(6) 多个线程通过一个或多个共享的变量共享数据；(7) 存储器模块是 DRAM、SRAM 和 FLASH 存储器中的一个；(8) 并行处理器中的至少一个被当作主处理器，并行处理器中的其它被当作从属处理器；(9) 每个处理器具有时钟速度，除了主处理器的每个处理器被操作为将该处理器的时钟速度调整为优化性能或功率消耗；(10) 每个处理器被操作为被当作主处理器或从属处理器；(11) 主处理器请求从属处理器的处理，等待来自从属处理器的输出，并且组合输出；(12) 当从几个处理器中的每个接收到输出时，主处理器组合来自几个处理的输出；(13) 通过停止并行处理器中的一个或多个来提供低功率耗散；以及 (14) 并行处理器中的每个与程序计数器相关联并且被操作为通过将全 1 写入与并行处理器相关联的程序计数器中而被停止。

[0095] 在又一方面，本发明包括一种系统，该系统包括：多个并行处理器，被嵌入在动态随机存取存储器 (DRAM) 芯片中，多个并行处理器与外部存储器控制器和外部处理器通信，并行处理器中的每个被操作为处理被优化线程级并行处理的指令集。

[0096] 在其它不同的实施方式中，(1) 所述芯片用 DRAM 引脚封装；(2) 并行处理器被安装在双列直插内存模块上；(3) 系统作为 DRAM 工作，除了处理器通过 DRAM 模式寄存器启用；(4) 外部处理器被操作为将数据和指令从相关联的永久存储设备传输至 DRAM；(5) 永久存储设备是 FLASH 存储器；以及 (6) 外部存储器被操作为在并行处理器与外部设备之间提供输入 / 输出接口。

[0097] 在又一方面，本发明包括一种系统，该系统包括：(a) 位于单个芯片上的多个处理器；以及 (b) 计算机存储器，位于芯片上并且被处理器中的每个访问；处理器中的每个被操

作为处理最小指令集，处理器中的每个包括专用于处理器中的至少三个特定寄存器的本地高速缓冲存储器。

[0098] 在其它不同的实施方式中，(1) 本地高速缓存存储器中的每个的大小等于芯片上的随机存取存储器的一行；(2) 每个处理器访问芯片上的随机存取存储器的内部数据总线，内部数据总线的宽度为随机存取存储器的一行的宽度；(3) 内部数据总线的宽度为1024、2048、4096、8192、16328 或 32656 位；(4) 内部数据总线的宽度是 1024 位的整数倍；(5) 专用于处理器中的至少三个特定寄存器的本地高速缓冲存储器被操作为在一个存储器读或写周期内被填充或清洗；(6) 最小指令集基本上由七个基本指令组成；(7) 基本指令集包括 ADD、XOR、INC、AND、STOREACC、LOADACC 和 LOADI 指令；(8) 最小指令集中的每个指令的长度最大为 8 位；(9) 最小指令集包括多个指令集扩展以优化处理器上的指令序列的执行；而且所述指令扩展基本上由小于 20 个指令组成；(10) 最小指令集包括一组指令以选择地控制芯片上的多个处理器；(12) 每个处理器控制指令的长度最大为 8 位；(13) 通过被设计用于单块存储器设备的半导体制造工艺，用位于芯片上的计算机存储器，在芯片上制造多个处理器；(14) 半导体制造工艺使用小于 4 层的金属互连；(15) 半导体制造工艺使用小于 3 层的金属互连；(16) 将多个处理器集成到计算机存储器电路导致小于 30％的芯片尺寸的增加；(17) 将多个处理器集成到计算机存储器电路导致小于 20％的芯片尺寸的增加；(18) 将多个处理器集成到计算机存储器电路导致小于 10％的芯片尺寸的增加；(19) 将多个处理器集成到计算机存储器电路导致小于 5％的芯片尺寸的增加；(20) 小于 250,000 个晶体管用于建立芯片上的每个处理器；(21) 通过小于 4 层的金属互连的半导体制造工艺制造芯片；(22) 处理器中的每个被操作为处理单线程；(23) 累加器是除了递增指令以外的每个基本指令的操作数；(24) 用于每个基本指令的目的地总是操作数寄存器；(25) 三个寄存器自动递增，三个寄存器自动递减；(26) 每个基本指令只需要一个时钟周期即可完成；(27) 指令集包括无 BRANCH 指令和无 JUMP 指令；以及 (28) 单个主处理器负责管理并行处理器中的每个。

[0099] 在又一方面，本发明包括一种系统，该系统包括：(a) 位于单个芯片上的多个并行处理器；以及 (b) 计算机存储器，位于芯片上并且被处理器中的每个访问，处理器中的每个被操作为处理被优化用于线程级并行处理的指令集；每个处理器访问芯片上的计算机存储器的内部数据总线，内部数据总线的宽度不大于存储器的一行的宽度。

[0100] 在不同的实施方式中，(1) 处理器中的每个被操作为处理最小指令集；(2) 处理器中的每个包括专用于处理器中的至少三个特定寄存器中的每个的本地高速缓冲存储器；(3) 本地高速缓冲存储器中的每个的大小等于芯片上的计算机存储器的一行；(4) 至少三个特定寄存器包括指令寄存器、源寄存器和目的地寄存器；(5) 最小指令集基本上由七个基本指令组成；(6) 基本指令集包括 ADD、XOR、INC、AND、STOREACC、LOADACC 和 LOADI 指令；(7) 指令集中的每个指令的长度最大为 8 位；(8) 处理器中的每个被操作为处理单线程；(9) 单个主处理器负责管理并行处理器中的每个；(10) 最小指令集包括多个指令扩展以优化处理器上的指令序列的执行；而且指令扩展包括小于 20 个指令；(11) 每个指令扩展的长度最大为 8 位；(12) 最小指令集包括一组指令以选择地控制芯片上的多个处理器；(13) 每个处理器控制指令的长度最大为 8 位；以及 (14) 能够通过被设计用于单块存储器设备的半导体制造工艺，用位于芯片上的计算机存储器，在芯片上制造多个处理器。

[0101] 在又一方面，本发明包括一种使用单个芯片上的多个并行处理器、主处理器和计算机存储器的线程级并行处理方法，多个处理器中的每个被操作为处理最小指令集并且处理单线程，该方法包括：(a) 为多个处理器中的每个内的三个特定寄存器中的每个分配本地高速缓冲存储器；(b) 分配多个处理器中的一个以处理单线程；(c) 由处理器处理每个分配的线程；(d) 对处理器处理每个线程的结果进行处理；(e) 在线程被处理之后不再分配多个处理器中的一个。

[0102] 在不同的实施方式中，(1) 最小指令集基本上由七个基本指令组成；(2) 基本指令包括 ADD、XOR、INC、AND、STOREACC、LOADACC 和 LOADI 指令；(3) 最小指令集包括一组指令以选择地控制多个处理器；(4) 每个处理器控制指令的长度最大为 8 位；(5) 该方法进一步包括如下步骤：每个处理器通过存储器的内部数据总线访问计算机存储器，内部数据总线的宽度为芯片上的存储器的一行的宽度；以及 (6) 最小指令集中的每个指令的长度最大为 8 位。

[0103] 在又一方面，本发明包括一种系统，该系统包括：(a) 多个处理器，被嵌入在与用于存储器设备的电子工业标准设备封装和引脚布局兼容的存储器芯片内；以及 (b) 一个或多个处理器，可通过被传输至存储器芯片的存储器模式寄存器的信息激活，存储器芯片的功能与工业标准存储设备的操作兼容，除了所述处理器的一个或多个通过所述存储器模式寄存器被激活的情况。

[0104] 在本发明的另一方面，提供了一种通过半导体制造工艺制备的线程优化多处理器，所述半导体制造工艺通过以下步骤执行：在至少一个芯片上将少于 16 层的金属互连；将至少一个处理器嵌入所述至少一个芯片中；以及将所述至少一个芯片安装在双列直插内存模块上。

[0105] 在本发明的又一方面中，提供了一种通过半导体制造工艺制备的线程优化多处理器，所述半导体制造工艺通过以下步骤执行：在至少一个芯片上将少于 8 层的金属互连；将至少一个处理器嵌入所述至少一个芯片中；以及将所述至少一个芯片安装在双列直插内存模块上。

[0106] 在本发明的又一方面中，提供了一种通过半导体制造工艺制备的线程优化多处理器，所述半导体制造工艺通过以下步骤执行：在至少一个芯片上将少于 4 层的金属互连；将至少一个处理器嵌入所述至少一个芯片中；以及将所述至少一个芯片安装在双列直插内存模块上。

附图说明

[0107] 图 1 描述了一个实施方式的示例性 TOMI 架构；

[0108] 图 2 是示例性的基本指令集；

[0109] 图 2A 示出了操作中的向前分支；

[0110] 图 2B 图示了示例性 TOMI 指令集的指令映射；

[0111] 图 2C 图示了一组示例性的多 TOMI 处理器管理指令扩展；

[0112] 图 2D 图示了用于 TOMI 处理器的时钟编程电路；

[0113] 图 2E 示出了一组示例性指令扩展；

[0114] 图 3 图解说明了不同寻址方式中的有效地址；

[0115] 图 4 图解说明了从 4-32 比特的数据路径是如何简单地被创建的；

[0116] 图 5 描述了示例性本地高速缓冲存储器；

[0117] 图 6 描述了示例性高速缓冲存储器的管理状态；

[0118] 图 7A 示出了利用宽系统 RAM 总线的额外处理功能的一个实施方式；

[0119] 图 7B 示出了用于将由 TOMI 处理器访问的两个存储体的数据线路交织的示例性电路；

[0120] 图 8 描述了示例性的存储器映射；

[0121] 图 9 描述了示例性的处理可用性表；

[0122] 图 10 图解说明了处理器分配的三个组件；

[0123] 图 10A 图示了 DIMM 封装上的多个 TOMI 处理器的实施方式；

[0124] 图 10B 图示了在与通用 CPU 接口连接的 DIMM 封装上的多个 TOMI 处理器的实施方式；

[0125] 图 10C 描述了用于多个 TOMI 处理器实施方式的示例性 TOMI 处理器初始化；

[0126] 图 10D 图示了用于 TOMI 处理器初始化的存储器模式寄存器的使用；

[0127] 图 10E 描述了示例性 TOMI 处理器可用性状态图；

[0128] 图 10F 示出了示例性处理器间通信电路设计；

[0129] 图 10G 示出了识别 TOMI 处理器来执行工作的示例性硬件实现；

[0130] 图 10H 示出了示例性处理器仲裁图；

[0131] 图 11 描述了示例性的分解；

[0132] 图 12 描述了示例性的系统 RAM；

[0133] 图 13A 描述了 64 个 TOMI 处理器的单块阵列的示例性平面布置图；

[0134] 图 13B 描述了 TOMI 处理器的单块阵列的另一示例性平面布置图；

[0135] 图 14A 描述了 TOMI 外围控制器芯片 (TOMIPCC) 的示例性平面布置图；

[0136] 图 14B 描述了使用 TOMIPCC 的蜂窝式便携无线电话语言翻译器应用的示例性设计；

[0137] 图 14C 描述了使用 TOMIPCC 和多个 TOMI DIMMS 的以存储器为中心的数据库应用的示例性设计；

[0138] 图 15A 至 15D 示出了 32 位 TOMI 处理器的示例性实施方式的顶层示意图；

[0139] 图 15E 示出了图 15A-D 所示的示意图的信号说明。

具体实施方式

[0140] 本发明至少一个实施方式的 TOMI 架构优选地使用可像通用计算机一样操作的最小逻辑。最常用的操作是优先的。大多数操作是可见的、常规的，并可用于编译器优化。

[0141] 在一个实施方式中，如图 1 所示，TOMI 架构是对累加器架构、寄存器架构和堆栈架构的变体。在该实施方式中：

[0142] 1. 类似于累加器架构，除了递增指令之外，累加器总是操作数之一。

[0143] 2. 类似于寄存器架构，目的地总是操作数寄存器之一。

[0144] 3. 累加器和程序计数器也是寄存器空间，从而可在其上操作。

[0145] 4. 三个专用寄存器自动递增和自动递减，并对于创建堆栈和输入输出流也是有用

的。

[0146] 5. 所有的指令都是 8 比特长，简化并加快了解码。

[0147] 6. 无分支 (BRANCH) 和跳转 (JUMP) 指令。

[0148] 7. 如图 2 所示，只有七条基本指令允许操作员从 8 比特指令中选择 3 比特。

[0149] 优选实施方式的一些好处包括：

[0150] 1. 所有的操作都以逻辑允许的最大速度运行，而不是由流水线所需的平均性而压缩。逻辑操作是最快的。数学操作是次快的。需要存储器存取的操作是最慢的。

[0151] 2. 该架构按任何数据宽度而调整，仅受组件管脚数、加法器进位次数以及有效性所限制。

[0152] 3. 该架构接近执行通用计算机的所有操作所需的最小可能功能性。

[0153] 4. 该架构是非常透明的、有规则的，且大多数操作都可用于优化编译器。

[0154] 该架构设计得足够简单，从而可以在单一的单块芯片上被复制许多次。一个实施方式将 CPU 的多份拷贝和存储器一起嵌入在单块芯片上。简化的 32 位 CPU 可用少于 1,500 个门实现，其中大多数门都用于定义寄存器。通过利用与单一的因特尔奔腾 4 中使用的相同数目的晶体管，可实现优选实施方式中的将近 1,000 个 TOMI CPU。

[0155] TOMI CPU 中的简化指令集执行通用计算机的必要操作。处理器的指令集越小，处理器越能有效地运行。TOMI CPU 被设计为与现代处理器架构相比具有格外少的指令数。例如，与具有 286 条指令的因特尔奔腾处理器、具有 195 条指令的因特尔安腾处理器、具有 127 条指令的 StrongARM 处理器和具有多于 400 条指令的 IMB Cell 处理器相比，TOMI CPU 的一个实施方式具有 25 条指令。

[0156] 与最新一代的奔腾处理器所需的 30 个时钟周期形成对比，TOMI CPU 的基本指令集被简化和设计为在单一的系统时钟周期内执行。TOMI CPU 架构是“非流水线”架构。这种架构和单一的时钟周期指令执行显著地减少或消除在其它并行的处理或流水线架构中发生的延迟、依赖性和浪费的时钟周期。尽管基本指令只需要单一时钟周期来执行时，但是随着时钟速度增加（并且时钟周期时间减少），执行结果传播通过用于复杂数学指令的电路晶体管门（例如，ADD）所需的时间可达到单一时钟周期的极限。在这种情况下，允许在两个时钟周期内执行特定的指令可能是最佳的，从而不会减慢较快指令的执行。这取决于 CPU 设计中系统时钟速度、制造工艺和电路设计的优化。

[0157] TOMI 简化的指令集允许用少于 5000 个晶体管（不包括高速缓冲存储器）构造 32 位 TOMI CPU。通过附图 15A 至 15D 示出了单一的 32 位 TOMI CPU 的实施方式的顶层示意图，图 15E 示出了信号说明。即使使用高速缓冲存储器和相关的译码逻辑，与最新一代因特尔奔腾微处理器芯片需要 250,000,000 个晶体管相比，32 位 TOMI CPU 可使用 40,000 至 200,000 个晶体管（取决于 CPU 高速缓冲存储器的大小）构造。遗传的微处理器架构（例如因特尔奔腾、安腾、IBM Cell 和 StrongARM 等）需要大量且递增数量的晶体管来实现处理容量的递增。TOMI CPU 架构通过为每个 CPU 核使用格外少的晶体管而与这种工业进步相反。TOMI CPU 的较少的晶体管数量提供了许多的优点。

[0158] 由于 TOMI CPU 的紧凑的尺寸，因此可在同一个硅芯片上构造多个 CPU。这也允许将多个 CPU 和主存储器（例如 DRAM）构造在同意芯片上，而仅在 DRAM 芯片自身的制造成本之外使用极少的附加制造成本。因此，能够通过最小程度地增加 DRAM 芯片的尺寸和制造成

本，将多个 TOMI CPU 置于单一芯片上用于并行处理。例如，512MB DRAM 包含大约 7 亿个晶体管。64 个 TOMI CPU（假设单一的 TOMI CPU 需要 200,000 个晶体管）仅在任何 DRAM 设计上增加 1280 万个晶体管。对于 512MB DRAM，64 个 TOMI CPU 将增加小于 5％的芯片尺寸。

[0159] TOMI CPU 被设计为通过现有的便宜的批量存储器制造工艺（例如，用于 DRAM、SRAM 和 FLASH 存储器设备的制造工艺）制造。用于 TOMI CPU 的晶体管的数量少意味着 CPU 能够在小面积内被构造，并且能够容易地通过 2 层金属互连的便宜的半导体制造工艺而不是用于通过 8 或更多层的金属互连或其它逻辑处理制造大微处理器芯片（例如因特尔奔腾）的昂贵的制造工艺在硅中互连。现代的 DRAM 和其它商用存储器芯片使用具有较少层（例如，2 层）金属互连的较简单且成本较低的半导体制造工艺来获得较低的制造成本、较大的产品产量和较高的产品产值。用于商用存储器设备的半导体制造工艺的特征通常在于低电流泄漏设备工作；而用于构造现代微处理器的工艺致力于高速和高性能特性，而不是晶体管级别的低电流泄漏值。通过用于 DRAM 和其它存储器设备的相同制造工艺有效地实现的 TOMI CPU 的能力使 TOMI CPU 能够在嵌入在现有的 DRAM 芯片（或其它存储器芯片）内，并且具有低成本、高产量的芯片制造工艺的优点。这还提供了通过当前在工业使用中用于 DRAM 和其它存储器芯片的相同的封装和设备管脚布局（例如，符合存储器设备的 JEDEC 标准）、制造设施、测试固定设备和测试向量制造 TOMI CPU 的优点。在反面，将 DRAM 存储器嵌入到传统微处理器芯片将在相对的方向上工作，这是因为该微处理器芯片是使用具有 8 或更多层金属互连的昂贵且复杂的逻辑制造工艺制造的并且存储器电路受到由微处理器操作产生的大的电噪声和热量，这就会影响嵌入在处理器芯片中的存储器的类型、大小和功能。这个结果将产生较高的成本、较低的产量、较高的功率消耗、较小的存储器、最终产生较低性能的微处理器。

[0160] 优选实施方式的另一优点是 TOMI CPU 足够小（需要很小的功率），从而可在物理上位于 DRAM（或其它存储器）电路的附近并且允许 CPU 访问超宽的内部 DRAM 数据总线。在现代 DRAM 中，这种总线是 1024、4096 或 8192 位宽（或其整数倍），还通常对应于 DRAM 设计内的数据块中的一行数据的宽度。（通过比较，因特尔奔腾数据总线是 64 位，因特尔安腾总线是 128 位宽。）TOMI CPU 的内部高速缓冲存储器的大小被设计为与 DRAM 行的大小匹配，从而 CPU 高速缓冲存储器可在单一的 DRAM 存储器读或写周期内被填充（或清洗）。TOMI CPU 使用超宽内部 DRAM 数据总线作为 TOMI CPU 的数据总线。TOMI CPU 高速缓冲存储器可被设计为对用于有效布局和电路操作的 DRAM 行和 / 或列锁存电路的设计进行镜像，包括至 TOMI CPU 高速缓冲存储器的数据传输。

[0161] 优选实施方式的又一优点是由于晶体管的数量少并且因为 CPU 使用超宽内部 DRAM 数据总线访问存储器而不是不断驱动 I/O 电路来访问用于数据的片外存储器而由 TOMI CPU 产生的较低的电噪声。片内 CPU 高速缓冲存储器允许直接访问用于处理的数据而使访问片外存储器的需要最少。

[0162] 处理器架构的设计目标是使处理容量和速度最大，而使实现该处理速度所需的功率最小。TOMI CPU 架构是具有极小功率消耗的高速处理器。处理器的功率消耗与设计中使用的晶体管的数量直接相关。用于 TOMI CPU 的较少数量的晶体管将使 TOMI CPU 的功率消耗最小。简化且有效的指令集也允许 TOMI CPU 降低其功率消耗。此外，TOMI CPU 高速缓存和通过宽的内部 DRAM 数据总线访问片内存储器将不必不断驱动用于片外存储器访

问的 I/O 电路。以 1GHz 时钟速度工作的单一的 TOMI CPU 消耗大约 20 至 25 毫瓦的功率。与之相反，因特尔奔腾 4 处理器在 2.93GHz 时需要 130 瓦，因特尔安腾处理器在 1.6GHz 时需要 52 瓦，StrongARM 处理器在 200MHz 时需要 1 瓦，以及 IBM Cell 处理器在 3.2GHz 时需要 100 瓦。众所周知的是，处理器产生的热量直接与处理器所需的功率量有关。极小功率 TOMI CPU 架构排除了对可在当前的微处理架构中找到的风扇、大热沉和外部冷却机构的需要。同时，小功率 TOMI CPU 架构使得新的小功率电池和太阳能供电应用变得适用。

[0163] 指令集

[0164] 示例性指令集中的七条基本指令以及其位映射如图 2 所示。每条指令优选地由单一的 8 比特字组成。

[0165] 寻址方式

[0166] 图 3 图解说明了不同寻址方式的有效地址。

[0167] 寻址方式是：

[0168] 直接寻址 (Immediate)

[0169] 寄存器寻址 (Register)

[0170] 寄存器间接寻址 (Register Indirect)

[0171] 寄存器间接自动递增寻址 (Register Indirect Auto-increment)

[0172] 寄存器间接自动递减寻址 (Register Indirect Auto-decrement)

[0173] 特殊情况

[0174] 寄存器 0 和寄存器 1 都是程序计数器 (PC)。在一个实施方式中，用寄存器 0(PC) 作为操作数的所有操作都是有条件的，即累加器进位位 (C) 等于 1。如果 C = 1，则将 PC 的旧值交换到累加器 (ACC) 中。而将寄存器 1(PC) 作为操作数的所有操作都是无条件的。

[0175] 在可选的实施方式中，用寄存器 0(PC) 作为目的地的写操作的条件是进位位 (C) 等于 0。如果 C = 1，则不执行任何操作。如果 C = 0，则将累加器 (ACC) 中的值写入 PC 并且程序控制变为新的 PC 地址。用寄存器 1 作为目的地的写操作是没有条件的。累加器 (ACC) 中的值被写入 PC，程序控制变为新的 PC 地址。

[0176] 用寄存器 0 作为源的读操作加载 PC 的值加上 2。在该方式中，循环顶部的地址可被读取和存储，用于随后的使用。在大多数情况中，将循环地址推入堆栈 (S) 中。用寄存器 1 作为源的读操作加载由 PC 寻址的下一全字节所指向的值。在该方式中，可加载 32 位直接操作数。32 位直接操作数必须字对齐，但是 LOADACC 指令可位于直接在 32 位直接操作数之前的 4 字节字中的任意字节位置。在执行读之后，PC 讲递增使得其对 32 位直接操作数之后的第一个字对齐指令进行寻址。

[0177] 无分支

[0178] 分支和跳转操作通常是 CPU 设计者的难题，因为它们需要宝贵的操作码空间中的很多位。分支功能可如下创建：通过利用 LOADACC,xx 将期望的分支地址加载到 ACC 内，然后使用 STOREACC,PC 指令来实现分支。通过存储到寄存器 0，分支可以 C 的状态为条件。

[0179] 跳转

[0180] 跳转通过执行 INC,PC 而创建。执行将会需要两个周期，一个用于完成当前 PC 递增周期，一个用于完成 INC。通过将寄存器 0 递增，跳转可以 C 的状态为条件。

[0181] 相对分支

[0182] 相对分支 (relative branch) 的创建是通过将期望的偏移量加载到 ACC 中，然后执行 ADD,PC 指令而实现的。通过与寄存器 0 相加，相对分支可以 C 的状态为条件。

[0183] 向前分支

[0184] 由于循环所需的向后分支 (rearward branch) 的位置能够通过按照首先经过循环顶部的程序步骤保存 PC 而简单地获得，因而向前分支 (forward branch) 比向后分支更有用。

[0185] 比相对分支更有效的向前分支的创建通过将分支端点的最低有效位加载到 ACC 中，然后存储到 PC 中而实现。由于可根据使用寄存器 0 或寄存器 1，而有条件或无条件地存取 PC，因而向前分支也可通过将 PC 寄存器 (寄存器 0 或寄存器 1) 作为目的地操作数的选择而为有条件或无条件的。

[0186] 例如 ：

[0187] LOADI,#1C

[0188] STOREACC,PC

[0189] 如果 ACC 的最高有效位都为零，则仅将 6 个最低有效位传送到 PC 寄存器。如果当前 PC 寄存器的 6 个最低有效位小于将要加载的 ACC 值，则将寄存器中的最高有效位保持不变。如果当前 PC 寄存器中的 6 个最低有效位大于将要加载的 ACC 值，则将当前 PC 寄存器从第 7 位开始递增。

[0190] 这样有效地允许高达 31 条指令的向前分支。向前分支的方法应该在一切可能的情况下使用，这不仅因为相对于相对分支的 3 条指令来说，其只需要 2 条指令，而且因为其不需要经过加法器，而经过加法器是最慢的操作。图 2 显示了操作中的向前分支。

[0191] 循环

[0192] 循环的顶部可使用 LOADACC,PC 来保存。得到的指向循环结构顶部的指针则可存储在寄存器中，或压入到一个自动索引的寄存器中。在循环的底部，可用 LOADACC，EA 获得指针，并使用 STOREACC，PC 将该指针重新存储到 PC 中，从而产生向后循环。通过存储到寄存器 0，循环可以 C 的状态为条件，从而造成有条件的向后循环。

[0193] 自修改代码

[0194] 可利用 STOREACC，PC 写自修改代码。指令可被创建或取至 ACC，然后被存储到 PC 中作为下一条指令来执行。该技术可用于创建情况 (CASE) 结构。

[0195] 假设存储器中的跳转表数组由 N 个地址和跳转表 (JUMPTABLE) 的基地址组成。为了方便起见，跳转表可位于低位存储器内，因此其地址可用 LOADI 或在 LOADI 之后跟随一个或多个右移位、ADD、ACC 来创建。

[0196] 假设进入跳转表的索引位于 ACC 内，且跳转表的基地址位于命名为 JUMPTABLE 的通用寄存器中 ：

[0197] ADD,JUMPTABLE 将索引加入跳转表的基地址中

[0198] LOADACC,(JUMPTABLE) 加载具有索引的地址

[0199] STOREACC,PC 执行跳转

[0200] 如果将从 0000 开始的低位存储器分配给系统调用，那么，每个系统调用可执行如下，其中，SPECIAL_FUNCTION 是直接操作数 0-63 的名字 ：

[0201] LOADI,SPECIAL_FUNCTION 加载系统调用的编号

[0202] LOADACC,(ACC) 加载系统调用的地址

[0203] STOREACC,PC 跳转到函数

[0204] 右移位

[0205] 基本的架构不会预想右移位操作。当需要该操作时，优选实施方式的解决方法是，将通用寄存器之一指定作为“右移位寄存器”。STOREACC,RIGHTSHIFT 将 ACC 右移位后的单一位置存储到“右移位寄存器”中，其中，可用 LOADACC,RIGHTSHIFT 读取该值。

[0206] 架构的可缩放性

[0207] TOMI 架构优选地以 8 位指令为特征，但是数据宽度无需受到限制。图 4 图解说明了从 4 到 32 位的任何宽度的数据分支是如何创建的。创建更宽数据处理只需要将寄存器组、内部数据路径和 ALU 的宽度增加到期望的宽度。数据路径的上限只由加法器进位传播延迟和晶体管预算所限制。

[0208] 为了简便起见，优选的 TOMI 架构实现为冯·诺伊曼存储器配置，但是哈佛架构实现（具有独立的数据总线和指令总线）也是可能的。

[0209] 通用数学操作

[0210] 2 的补码运算能以几种方法实现。可将通用寄存器预配置为全“1”并命名为 ALLONES。假定操作数位于命名为 OPERAND 的寄存器中：

[0211] LOADACC,ALLONES

[0212] XOR,OPERAND

[0213] INC,OPERAND “2”的补码被留在 OPERAND 中

[0214] 通用编译器结构

[0215] 大多计算机程序都是由编译器生成的。因此，有用的计算机架构应该擅长于通用编译器结构。

[0216] C 编译器通常使堆栈维持将参数传递给函数调用。S、X 或 Y 寄存器可用作堆栈指针。函数调用使用例如 STOREACC,(X)+，将参数压入到担当堆栈的一个自动索引寄存器上。在进入函数之后，将参数弹出到通用寄存器中，以备使用。

[0217] 堆栈相对寻址

[0218] 有时，传递到函数调用的元素会多于适合于通用寄存器的元素。对于下面的实施例，假定堆栈压入操作使堆栈递减。如果将 S 用作堆栈寄存器，读取相对于堆栈顶部的第 N 项：

[0219] LOADI，N

[0220] STOREACC，X

[0221] LOADACC,S

[0222] ADD,X

[0223] LOADACC，(X)

[0224] 在数组中编索引

[0225] 进入数组函数之后，数组基地址则位于命名为 ARRAY 的通用寄存器中。读取数组中的第 N 个元素：

[0226] LOADI,N

[0227] STOREACC,X

[0228] LOADACC5ARRAY

[0229] ADD, X

[0230] LOADACC, (X)

[0231] 在 N 字元素数组中编索引

[0232] 有时，数组将为元素分配 N 个字的宽度。数组的基地址位于命名为 ARRAY 的通用寄存器中。在 5 字宽的数组中访问第 N 个元素的第一个字：

[0233]

```
LOADI, N
STOREACC, X          存储到临时寄存器中
ADD, ACC             乘以 2
ADD, ACC             再次乘以 2，等于 4
ADD, X               加 1，等于 5
LOADACC, ARRAY
ADD, X               加上数组的基地址
LOADACC, (X)
```

[0234] 指令集扩展

[0235] 本发明的另一实施方式包括如图 2 所示的七个基本指令的扩展。如图 2E 所示的指令集扩展有助于进一步优化 TOMI 处理器的内部操作、软件程序指令和用于 TOMI 处理器的软件编译器。

[0236] SAVELOOP—这个指令将程序计数器的当前值推入堆栈中。Saveloop 最有可能在循环结构的顶部处执行。在循环的底部处，所保存的程序计数器值将从堆栈被拷贝且存储到程序计数器中，实现向后跳转至循环的顶部。

[0237] SHIFTLOADBYTE—这个指令将 ACC 向左移位 8 位，读这个指令之后的 8 位字节，并且将其置于 ACC 的最不重要的 8 位内。在该方式中，可用指令序列加载长直接操作数。例如，加载 14 位的直接操作数：

[0238] LOADI, #14 \\ 加载 14 位操作数的最重要的 6 位

[0239] SHIFTLOADBYTE \\ 将这 6 位向左移位 8 个位置并且加载接下来的 8 位值

[0240] CONSTANT#E8 \\8 位直接操作数

[0241] 在 ACC 中得到的十六进制值是 14E8。

[0242] LOOP—这个指令将堆栈的顶部拷贝到程序计数器。Loop 最有可能在执行 Saveloop 之后的循环结构的底部处执行，以将程序计数器保存在 loop 的顶部。当 Loop 执行时，所保存的程序计数器将从堆栈被拷贝且保存至程序计数器，实现向后跳转至循环的底部。

[0243] LOOP_IF—这个指令将堆栈的顶部拷贝至程序计数器。这个指令基于 C 的值，执行有条件的循环。Loop_if 最有可能在执行 Saveloop 之后的循环结构的底部处执行，以将程序计数器保存在循环的顶部处。当 Loop_if 执行时，如果 C = 0，则所保存的程序计数器将从堆栈被拷贝且保存至程序计数器，实现向后跳转至循环的顶部。如果 C = 1，则程序计数

器递增以指向下一个连续的指令。

[0244] NOTACC—对 ACC 的每位进行补码。如果 ACC = 0,将 C 设为 1。否则,将 C 设为 0。

[0245] ROTATELEFT8—将 ACC 向左旋转 8 位。在每个旋转步骤处,移出 ACC 的 MSB 移入 ACC 的 LSB。

[0246] ORSTACK—对 ACC 和堆栈顶部的值执行逻辑或 (OR)。将结果置于 ACC 内。如果 ACC = 0,则将 C 设为 1。否则,将 C 设为 0。

[0247] ORSTACK+—对 ACC 和堆栈顶部的值上执行逻辑 OR。将值置于 ACC 内。在逻辑操作之后,递增堆栈指针 S。如果 ACC = 0,则将 C 设为 1。否则,将 C 设为 0。

[0248] RIGHTSHIFTACC—将 ACC 向右移位单一的比特。将 ACC 的 LSB 移入 C 内。

[0249] SETMSB—设置 ACC 的最重要的位。C 没有改变。这个指令在执行符号比较时使用。

[0250] 本地 TOMI 高速缓存

[0251] 高速缓冲存储器是在尺寸上小于主存储器、但比主存储器具有更快的存取速度的存储器。对于许多操作而言,减少的存取时间以及程序和数据存取的本地化使得高速缓存操作能够提高优选 TOMI 处理器的性能。从另一个方面看,通过增加 TOMI 处理器相对于主存储器的独立性,高速缓冲存储器提高了并行处理性能。高速缓存相对于主存储器的性能以及在需要将另一个主存储器从高速缓冲存储器加载或存储到高速缓冲存储器之前 TOMI 处理器能够执行的周期数目将确定由于 TOMI 处理器的并行性而产生的性能的增加量。

[0252] 由于 TOMI 处理器的并行性,TOMI 本地高速缓冲存储器提高了性能的增加。如图 5 所示,每个 TOMI 处理器优选地具有三个相关的本地高速缓冲存储器 :

[0253] Instruction(指令)—与 PC 相关

[0254] Source(源)—与 X 寄存器相关

[0255] Destination(目的地)—与 Y 寄存器相关

[0256] 由于高速缓冲存储器与特定的寄存器相关联,而不是“数据”或“指令”读取,因此高速缓冲存储器控制逻辑被简化,且高速缓冲存储器等待时间显著减少。这些高速缓冲存储器的最佳尺寸是由应用程序决定的。通常的实施方式中,每个高速缓冲存储器需要 1024 字节。换句话说,1024 条指令,以及 256 个 32 比特字的源和目的地。至少两个因素确定高速缓冲存储器的最佳尺寸。第一是在需要另一个高速缓冲存储器加载或存储操作之前,TOMI 处理器能够循环的状态数目。第二是高速缓冲存储器从主存储器加载或存储操作的成本,其中该成本相对于在主存储器操作过程中 TOMI 处理器执行周期的数目。

[0257] 在 RAM 中嵌入 TOMI 处理器

[0258] 在一个实施方式中,宽的总线将大的嵌入式存储器连接到高速缓冲存储器,因此到高速缓冲存储器的加载或存储操作能快速地发生。通过嵌入到 RAM 的 TOMI 处理器,整个高速缓冲存储器的加载或存储将由对 RAM 列的单存储器周期组成。在一个实施方式中,嵌入式存储器将响应 63 个 TOMI 处理器的请求,因此当另一个 TOMI 处理器高速缓冲存储器完成时,高速缓冲存储器对一个 TOMI 处理器加载或存储的响应时间可延长。

[0259] 如图 6 所示,在高速缓冲存储器可根据相关存储器寻址寄存器 X,Y,PC 的改变进行存储和加载。例如,PC 寄存器的总宽可以是 24 位。如果 PC 高速缓冲存储器是 1024 字节,PC 的低 10 位将定义 PC 高速缓冲存储器中的存取。当对 PC 写入以使其高 14 位有变化时,则将需要高速缓冲加载周期。与 PC 高速缓冲存储器相关的 TOMI CPU 将停止执行,直到高

速缓冲加载周期结束，且可从 PC 高速缓冲存储器读取指示指令。

[0260] 高速缓冲存储器双缓冲

[0261] 在预期到高速缓冲存储器的加载需求时，可加载次级高速缓冲存储器。两个高速缓冲存储器将为相同的，并根据 PC 高 14 位的内容可交替地选择和取消选择。在上述的实施例中，当 PC 的高 14 位变为与预缓存在次级高速缓冲存储器中的数据的高 14 位相匹配时，则可将次级高速缓冲存储器选为主高速缓冲存储器。之前的主高速缓冲存储器则将变成次级高速缓冲存储器。由于大多数计算机程序的存储器呈线性递增，本发明的一个实施方式的次级高速缓冲存储器将总是用来读取当前 PC 的高 14 位加 1 的、主存储器的高速缓冲的存储器内容中的内容。

[0262] 当将存储器数据转移到当前存储器外部时，增加次级高速缓冲存储器将会减少 TOMI 处理器等待从主存储器读取存储器数据的时间。次级高速缓冲存储器的增加几乎将 TOMI 处理的复杂性提高为两倍。对于最佳化系统，该双倍复杂性应该由相应地将 TOMI 处理器性能提高为两倍来抵消。否则，能用相同数目的晶体管实现没有次级高速缓冲存储器的两个简易的 TOMI 处理器。

[0263] 高速乘法、浮点操作、额外的功能性

[0264] 整数乘法和所有浮点操作需要许多执行周期，即使是专用硬件也是一样的。因此，应该将这些操作包含在其它处理器内，而不是包含在基本 TOMI 处理内。然而，可将简单的 16 位乘以 16 位的乘法器添加至 TOMI CPU（利用小于 1000 个晶体管）以将附加功能和多功能提供给 TOMI CPU 架构。

[0265] 数字信号处理 (DSP) 操作常常使用深度流水线化的乘法器，这些乘法器在每个周期产生一个结果，即使总的乘法需要许多周期。对于重复相同算法的单处理应用，这样的乘法器架构是最优化的，且可并合并为 TOMI 处理器的外围处理器，但是如果将其直接合并到 TOMI 处理的话，则可能会增加复杂性并降低总体性能。图 7A 显示了利用宽的系统 RAM 总线组织的额外处理功能的一个实施例。

[0266] 访问邻近的存储体

[0267] 存储器芯片中的存储器电路的物理布局设计通常被设计为使得存储器晶体管被布置在大的存储单元存储体内。存储体通常被组织为相等大小的矩形区域并且被置于芯片的两列或多列内。存储器单元在大存储体内的布局可用于加速存储器的读和 / 或写访问。

[0268] 在本发明的一个实施方式中，一个或多个 TOMI 处理器可被置于存储器芯片内的两列存储单元存储体之间。两个存储体的行数据线可被交叉使得利用如图 7B 所示的逻辑，TOMI 处理器可通过使选择 A 或选择 B 使能来访问存储体 A 或存储体 B。在这种方式中，通过存储器芯片内的特定 TOMI 处理器直接可寻址的存储器可被加倍。

[0269] TOMI 中断策略

[0270] 中断是处理器正常顺序操作之外的且造成处理器立即改变其操作顺序的事件。中断的实施例可能是外部装置操作的完成或某些硬件的错误条件。传统的处理器竭尽全力地快速停止正常顺序的操作，保存当前操作的状态，开始执行一些特殊的操作来处理由任何事件造成的中断，且当特殊操作完成时恢复先前的状态并继续顺序操作。中断处理质量的基本标准是响应时间。

[0271] 中断对传统处理器造成一些问题。其使执行时间不确定，浪费处理器周期存储和

恢复状态，使处理器设计变得复杂，并可引入减慢每个处理器操作的延时。

[0272] 立即中断响应对大多数处理器来说是没必要的，除了正在进行错误处理以及处理器直接与真实世界的活动进行交互。

[0273] 在多处理器 TOMI 系统的一个实施方式中，只有一个处理器处理基本的中断权能。所有其它处理器都非中断地运行，直到它们完成某些指定的工作并停止，或直到它们被协调处理器所停止。

[0274] 输入 / 输出 (I/O)

[0275] 在 TOMI 处理器环境的一个实施方式中，单一的处理器负责与外部世界的所有接口。

[0276] 直接存储器存取 (DMA) 控制

[0277] 在一个实施方式中，在 TOMI 处理器系统中通过 DMA 控制器立即响应外部世界。当被外部装置请求时，DMA 控制器将数据从外部装置传送到内部数据总线，用于写入系统 RAM。当被请求时，相同的控制器还将数据从系统 RAM 传送到外部装置。DMA 请求将具有内部总线存取的最高优先权。

[0278] 组织 TOMI 处理器阵列

[0279] 本发明优选实施方式的 TOMI 处理器设计为可被大量复制，且在单片芯片上与附加的处理功能性、非常宽的内部总线以及系统存储器像组合。该系统的示例性的存储器映射如图 8 所示。

[0280] 每个处理器的存储器映射的前 32 个位置 (十六进制的 1F) 用于该处理器的本地寄存器 (见图 3)。存储器映射的剩余位置可由所有处理器通过它们的高速缓冲存储器的寄存器进行寻址 (见图 6)。系统 RAM 的寻址范围只由与本地高速缓冲存储器相关的三个寄存器 PC，X，Y 的宽度限制。如果寄存器是 24 位宽，总的寻址范围将会是 4M 字节，但是无上限。

[0281] 在一个实施方式中，64 个 TOMI 处理器与存储器一起单片地实现。单一的主处理器负责管理其它 63 个处理器。当一个从属处理器空闲时，其就不计时，因此其消耗很少的功率或不消耗功率，并产生少量的热量或不产生热量。在初始化时，只有主处理器是操作的。主处理器开始取指令并执行指令，直到线程应该启动时。每个线程都已被预编译且加载到存储器中。为了启动线程，主处理器将线程分配到其中一个 TOMI CPU 中。

[0282] 处理器可用性

[0283] TOMI 处理器的工作可用性的协调优选地由图 9 示出的处理器可用性表来进行处理。协调 (主) 处理器优选地执行如下功能 :

[0284] 1. 将从属处理器的调用参数压入其堆栈中，所述参数包括但不限于线程的执行地址、源存储器和目的存储器。

[0285] 2. 启动从属处理器。

[0286] 3. 通过轮询或通过响应中断，来响应从属处理器线程完成事件。

[0287] 请求一个处理器

[0288] 协调处理器可从可用性表中请求处理器。返回可用标志 (available_flag) 设为“0”的最低处理器。然后协调处理器可将与可用处理器相关的可用标志设为“1”，从而启动从属处理器。如果无处理器可用，请求将会返回错误。可选地，可由协调处理器根据与请求

的将要执行的工作相关联的优先级分配处理器。根据优先权方案进行资源分配的技术是本领域公知的。图 10 图解说明了处理器分配的三个优选部分：协调处理器启动操作、从属处理器操作以及通过中断响应进行协调处理器结果处理。

[0289] 逐步启动从属处理器

[0290] 1. 协调处理器将用于线程运行的参数压入其自身的堆栈中。这些参数可包括：线程的起始地址，线程的源存储器、线程的目的存储器以及最后的参数计数 (parameter_count)。

[0291] 2. 协调处理器请求可用的处理器。

[0292] 3. 处理器分配逻辑返回设置了相关的可用标志并清除了相关的已完成标志 (done_flag) 的最低编号的从属处理器的号码，或者返回错误。

[0293] 4. 如果返回错误，协调处理器则重试请求直到从属处理器变得可用，或者执行某些特殊的操作来处理错误。

[0294] 5. 如果返回可用的处理器号码，协调处理器则清除指示的处理器的可用标志。该操作将堆栈参数的参数计数号压入所选处理器的堆栈中。将已完成标志清零。

[0295] 6. 从属处理器获取栈顶元素，并将其传送给从属处理器的程序计数器。

[0296] 7. 然后从属处理器将程序计数器指示的存储器列读取到指令高速缓冲存储器中。

[0297] 8. 从属处理器开始从指令高速缓冲存储器的起始处开始执行指令。第一条指令将可能从堆栈获取调用参数。

[0298] 9. 从属处理器执行指令高速缓冲存储器中的线程。当线程完成时，检查其相关的已完成标志的状态。如果设置了已完成标志，则等待直到已完成标志被清除，从而指示协调处理器已处理了任何先前结果。

[0299] 10. 如果与从属处理器相关的中断向量设置为 -1，则不通过设置已完成标志而创建中断。因此协调处理器可轮询已完成标志是否设置。

[0300] 当协调处理器检测到已设置了已完成标志时，其则可处理从属处理器的结果，并可重新指定从属处理器进行新的工作。当从属处理器的结果已经被处理时，相关的从属处理器则将清除相关的已完成标志。

[0301] 如果与从属处理器相关的中断向量不等于 -1，那么，设置相关的已完成标志将会造成协调处理器被中断，并且在中断向量地址开始执行中断处理程序。

[0302] 如果相关的可用标志也已设置，协调处理器则也可读取被压入到从属堆栈的返回参数。

[0303] 中断处理程序将处理从属处理器的结果，并且可能为重新指定从属处理器进行新的工作。当从属处理器的结果已被处理时，运行在协调处理器上的中断处理程序则将清除相关的已完成标志。

[0304] 11. 如果已完成标志被清除，从属处理器则设置其相关的已完成标志，并保存新的起始时间。从属处理器可继续工作或可返回可用状态。为了返回可用状态，从属处理器可将返回参数压入其堆栈中，然后进行堆栈计数并设置其可用标志。

[0305] 使用存储器模式寄存器管理 TOMI 处理器

[0306] 用于实现和管理多个 TOMI 处理器的一个技术是将 TOMI 处理器安装在如图 10A 所示的双列直插内存模块 (DIMM) 中。TOMI/DIMM 可被包含在由外部存储器控制器和通用 CPU

构成的系统（例如，个人计算机）中。图 10B 示出了这种构造。模式寄存器通常可在 DRAM、SRAM 和 FLASH 存储器内找到。模式寄存器是一组锁存器，这些锁存器可独立于存储器存取由外部存储器控制器写入。存储器模式寄存器中的位通常用于指定例如同步、刷新控制和输出脉冲长度的参数。

[0307] 能够在存储器模式寄存器内分配一个或多个位以启用或禁用 TOMI CPU。例如，当 TOMI CPU 被模式寄存器禁用时，包含 TOMI CPU 的存储器将作为普通的 DRAM、SRAM 或 FLASH 存储器起作用。当模式寄存器启用 TOMI CPU 初始化时，按照图 10C 所描述的顺序执行。在该实施方式中，单个处理器被确定为主处理器。这个处理器总是在 RESET 操作之后首先启动。在初始化结束时，主处理器以全速运行并且执行期望的应用程序。当 TOMI CPU 执行时，DRAM、SRAM 或 FLASH 存储器不能被存取。有时，存储器模式寄存器可由外部存储器控制器控制以暂停 TOMI CPU 的执行。当 TOMI CPU 暂停时，DRAM、SRAM 或 FLASH 的内容可被附接至通用 CPU 的外部存储器控制器读取。在该方式中，结果可被传递至通用 CPU，附加的数据或执行结果可被写入 DRAM、SRAM 或 FLASH 存储器。

[0308] 当通用 CPU 已经完成了 DRAM、SRAM 或 FLASH 存储器的任何读或写时，外部存储器控制器将模式寄存器位从 HALT 写成 RUN，TOMI CPU 从它们停止处继续执行。图 10D 示出了 DRAM、SRAM 或 FLASH 存储器的通常存储器模式寄存器以及这些寄存器如何被修改以控制 TOMI CPU。

[0309] 调整处理器时钟速度

[0310] 处理器时钟速度确定处理器功率耗散。TOMI 架构通过启用除了将要停止的一个处理器之外全部的处理器来实现低功率耗散。而且，除了主处理器之外的每个处理器的时钟速度都可被调整以利用如图 2D 所示的逻辑电路优化性能或功率消耗。

[0311] TOMI 处理器管理的另一实施方式

[0312] 某些计算机软件算法是递归的。换句话说，该算法的主要功能是调用其自身。被称为“分治法”的一类算法通常使用递归技术实现。分治法适用于数据的搜索和分类、以及某些数学函数。可用多个处理器（例如 TOMI 架构可用的那些处理器）并行执行这些算法。为了实现这些算法，一个 TOMI CPU 必须能够将工作传递给另一个 TOMI CPU 并且从那个 CPU 接收结果。TOMI 处理器的另一实施方式允许任一处理器为主处理器并且将任何其它可用的 TOMI 处理器作为从属处理器。在这个处理器管理的实施方式中支持启动和停止 TOMI 处理器、在处理器之间通信、和管理独立的和相关的线程。

[0313] 停止 TOMI CPU

[0314] TOMI CPU 可通过将全 1 写入其 PC 中来停止。当 TOMI CPU 被停止时，其时钟不再运转并且不再消耗功率。任何 TOMI CPU 可将全 1 写入其自身的 PC 内。

[0315] 启动 TOMI CPU

[0316] 当不是全 1 的其它值被写入 TOMI CPU 的 PC 内时，TOMI CPU 可开始执行。当主处理器通过如图 10D 所示的模式寄存器复位时，主处理器将 0 值写入其 PC 内。

[0317] 独立的处理器线程

[0318] 当多个线程在单个通用处理器上执行时，这些线程可能非常松散地连接，很少通信。某些线程可永久运行连续且永恒地连续传递结果而不是运行、返回结果和停止。这些线程的实施例可以是网络通信线程或读取鼠标设备的线程。鼠标线程连续运行，并将鼠标

位置和点击信息传递至可被轮询的共享存储器区域或者直接出现的回调程序中。

[0319] 这些独立的线程主要用于简化编程而不是加速执行。类似的线程可在例如 TOMI 的多处理器系统上执行。结果可被传递至共享的存储器区域。在某些情况下，可通过共享的变量完成通信。

[0320] 在 TOMI 架构中，共享的变量可比共享的存储器更有效，这是因为这种变量可避免在存储器 RAS 周期内将完整的行加载到 X_cache 或 Y_cache 的必要性。变量使用的实施例是指向用于监控网络流量的 TOMI CPU 的接收缓存器的输入指针。当接收到数据时，网络监控 CPU 递增该变量。数据消耗 CPU 不时地读取变量并且当足够的数据出现时执行操作以将存储行加载到 X_cache 或 Y_cache。然后，可从高速缓冲存储器读取接收到的网络数据直至由共享变量指示的值。

[0321] 相关的处理器线程

[0322] 某些算法（例如，被分类为分治法的那些算法）能够通过同时在几个处理器上执行算法片并且结合结果来实现并行。在这种设计中，单个主处理器将从几个从属处理器请求工作，然后在从属处理器并行执行这些工作时等待。在这种方式中，主处理器依赖于正由从属处理器完成的工作。

[0323] 当从属处理器的工作完成时，主处理器读取部分结果并且将它们组合成最终的结果。这个功能使 TOMI 架构能够有效地处理被称为“分治法”的一类算法。多个普通简单的分治法算法中的某些是搜索和分类。

[0324] 多处理器管理指令集扩展

[0325] 基本 TOMI 指令的一系列扩展允许独立和相关的线程管理。这些指令通过使用如图 2B 所示的可用的 NOOP 代码中的某些来实现。这些管理扩展指令在图 2C 中概述。

[0326] GETNEXTPROCESSOR—这个指令询问处理器可用性表并且将与下一个可用处理器相关联的数加载至 ACC。

[0327] SELECTPROCESSOR—这个指令将 ACC 写入处理器选择寄存器。处理器选择寄存器选择哪个处理器将由 TESTDONE 和 READSHAREDVARIABLE 估计。

[0328] STARTPROCESSOR—这个指令写入由处理器选择寄存器选择的处理器的 PC。这个指令最有可能在主处理器想要启动停止的从属处理器时执行。如果从属处理器的 PC 为全 1，则该从属处理器停止。通过将值写入从属处理器的 PC，主处理器使从属处理器开始在写入的 PC 的位置处运行程序。如果这个操作成功，则 ACC 包含正被写入所选处理器的 PC 值。如果这个操作失败，则 ACC 包含 -1。失败操作的最有可能的原因是所选的处理器不可用。

[0329] TESTDONE—这个指令测试由处理器选择寄存器选择的处理器的 PC，并且如果 PC ＝全 1 则将调用处理器的 C 位设为“1”。在这种方式中，可通过如下步骤创建对 PC 的循环测试：

[0330] LOADI, 处理器号

[0331] SELECTPROCESSOR

[0332] LOADACC, LOOPADDRESS

[0333] TOP TESTDONE

[0334] STOREACC, PC_COND// 循环至 TOP 直到所选处理器的 PC ＝全 1 而停止

[0335] TESTAVAILABLE—这个指令测试用于由处理器选择寄存器选择的处理器的处理器

分配表位中的“可用”位，并且如果所选的处理器可用则将调用处理器的 C 位设为“1”。在这种方式中，可通过如下步骤创建用于测试进一步工作可用性的循环：

[0336] LOADI, 处理器号

[0337] SELECTPROCESSOR

[0338] LOADACC, LOOPADDRESS

[0339] TOP TESTAVAILABLE

[0340] STOREACC, PC_COND// 循环至 TOP 直到所选的处理器可用

[0341] SETAVAILABLE—这个指令设置用于由处理器选择寄存器选择的处理器的处理器分配表中的“可用”位。这个指令最有可能由已经请求另一处理器做如图 10E 所示的工作的一个处理器执行。当工作处理器完成工作时，通过将其 PC 设为全 1 而停止。请求处理器周期性地为工作处理器进行 TESTDONE。当工作处理器已经完成工作时，请求处理器将通过共享的存储位置或者通过共享的变量读取结果。当结果已经被读取时，工作处理器可用以被再分配给另一任务，请求处理器将使用 SETAVAILABLE 使其它处理器可请求它做进一步的工作。

[0342] READSHAREDVARIABLE—这个指令读取由处理器选择寄存器选择的处理器的共享变量。这个指令最有可能由已经请求另一处理器工作的一个处理器执行。共享的变量可由任意处理器读取以确定分配的工作的进程。作为实施例，工作处理器可被分配以处理正从高速网络接收的数据。共享的变量指示已经被读取的数据的量并对其它处理器可用。每个处理器包含共享变量。共享变量可由任何其它处理器读取，但是共享变量只能由其自身的处理器写入。

[0343] STORESHAREDVARIABLE—这个指令将 ACC 的值写入执行该指令的处理器的共享变量中。共享变量可由任何其它处理器读取并且用于将状态和结果传输至其它处理器。

[0344] 使用数据就绪锁存器进行处理器间通信

[0345] 图 10F 图示了 TOMI 处理器间通信的一个可能的硬件实现。一个 TOMI 处理器可通过 SELECTPROCESSOR 命令建立其自身与另一 TOMI 处理器之间的连接。这个指令通过共享寄存器、READSHAREDVARIABLE 和 STORESHAREDVARIABLE 命令建立允许选择的和被选择的处理器交换数据的逻辑连接。

[0346] 图 10F 的上半部示出了处理器的、将数据发送至由就绪标志寄存器控制的另一处理器的逻辑电路。图 10F 的下半部示出了处理器的、从由就绪标志寄存器控制的另一处理器接收数据的逻辑电路。

[0347] 可将共享寄存器的状态读入进行选择或被选择的处理器的 C 位。在操作中，处理器将数据写入其共享的寄存器内，从而设置相关联的数据就绪标志。连接的处理器读取共享寄存器，直到 C 位指示相关联的数据就绪标志已经被设置。读操作清除就绪标志，从而处理器能够重复。

[0348] 仲裁处理器分配

[0349] 如上所述，TOMI 处理器能够将工作委托给其可用性已经通过 GETNEXTPROCESSOR 确定的另一 TOMI 处理器。

[0350] GETNEXTPROCESSOR 确定处理器是否可用。可用的处理器是当前没有执行工作、也没有保持还没有由 READSHAREDVARIABLE 获取的先前的工作的结果的处理器。

[0351] 图 10G 示出了识别可被委托工作的可用处理器的一个硬件实现。图 10H 示出了示例性处理器仲裁的事件。该过程如下所示：

[0352] 1. 请求处理器执行 GETNEXTPROCESSOR 指令，将“请求下一个可用的处理器”行推入仲裁逻辑。

[0353] 2. 仲裁逻辑生成与“下一个可用的处理器”行上的 TOMI CPU 对应的号。

[0354] 3. 请求处理器执行 SELECTPROCESSOR 指令，将该号存储至请求处理器的 PSP（处理器选择寄存器）。

[0355] 4. 然后，请求处理器执行 STARTPROCESSOR 指令，该指令写由处理器选择寄存器选择的处理器的 PC。如果该操作成功，则也将所选择的处理器的号保存至仲裁逻辑以指示所选择的处理器不再能用于被分配做工作。如果该操作失败，则原因可能是所选择的处理器不可用。请求处理器将执行另一 GETNEXTPROCESSOR 以找到另一可用的处理器。

[0356] 5. 当所选择的处理器执行 STORESHAREDVARIABLE 以使其结果可用时，通知仲裁逻辑所选择的处理器具有等待被读取的结果。

[0357] 6. 当通过将 -1 写入所选择的处理器的 PC 来停止该处理器时，通知仲裁逻辑所选择的处理器可用。

[0358] 7. 当通过 READSHAREDVARIABLE 获取所选择的处理器的结果时，通知仲裁逻辑已经读取所选择的处理器的结果。

[0359] 存储器锁定

[0360] TOMI 处理器通过其高速缓冲存储器对系统存储器进行读和写。一次对一个完全高速缓冲存储的列进行读或写。任何存储器可读取系统存储器的任何部分。单独的处理器可将存储器的列锁定为专用于排它地进行写操作。该锁定机制避免了处理器之间的存储器写冲突。

[0361] 建议的应用程序

[0362] 并行操作有效地加速了可分为独立的工作片段用于各处理器的应用程序。一个容易分解的应用程序是机器人视觉的图像处理。图像处理算法包括相关性、均衡、边缘识别以及其它操作。许多操作都是由矩阵处理执行。如图 11 所示，该算法常常很好地分解。

[0363] 在图 11 中，示例性的图像映射显示了 24 个处理器，其中每个处理器被分配以处理全部图像映射的一个矩形子集中的图像数据。

[0364] 图 12 示出了在一个实施方式中如何分配 TOMI 系统 RAM。一块系统 RAM 保存图像捕捉像素，且另一个块保存处理的结果。

[0365] 在操作中，协调处理器分配了 DMA 信道，以每隔一段固定的时间就将图像像素从外部源传送到内部系统 RAM。图像捕捉的速度通常是 60 个图像 / 秒。

[0366] 然后，协调处理器通过将待由 X 寄存器使用的图像映射的地址、待由 Y 寄存器使用的已处理图像的地址、2 的参数计数、以及图像处理算法的地址压入堆栈，而使从属处理器 1 可用。协调处理器随后类似地通过 25 使处理器 2 可用。处理器继续以并行的方式独立执行，直到图像处理算法完成。

[0367] 当算法完成时，每个处理器在处理器可用性表中设置其相关的已完成标志。该结果由协调处理器处理，该协调处理器对完成进行轮询或对“完成”事件上的中断做出响应。

[0368] 图 13A 和 13B 示出了用片内系统存储器实现的 64 个 TOMI 处理器的单块阵列的示

例性平面布置图。该平面布置图可随着电路设计、存储器电路的布局和整体芯片架构而变化。

[0369] TOMI 外围控制芯片 (TOMIPCC)

[0370] TOMI 架构的一个实施方式将一个或多个 TOMI CPU 嵌入到标准的 DRAM 芯片中。所形成的芯片封装在具有标准的 DRAM 引出线的标准 DRAM 封装。封装的零件可被安装在标准 DRAM DIMM(双列直插内存模块) 上。

[0371] 在操作中,这个实施方式以类似于标准 DRAM 的方式,除了嵌入的 TOMI CPU 是通过 DRAM 模式寄存器而启用。当 TOMI CPU 被启用并且工作时,其执行由外部处理器加载至 DRAM 的程序。通过共享 DRAM 将 TOMI CPU 的计算结果提供给外部处理器。

[0372] 在某些应用中,通过 PC 提供外部处理器。在另一应用中,可提供专用处理器以执行用于 TOMI DRAM 芯片的下列功能。图 14A 示出了这种处理器,TOMI 外围控制芯片 (TOMIPCC),的一个实施方式。该处理器的功能为 :

[0373] 1. 提供将数据和指令从相关联的永久存储设备传输至 TOMI 芯片 DRAM 中使用的机制。在许多系统中,永久存储设备可以是闪存 RAM。

[0374] 2. 在 TOMI 芯片与真实的设备 (例如,显示器和网络) 之间提供输入 / 输出接口。

[0375] 3. 执行协调 TOMI CPU 操作所必需的一小组操作系统功能。

[0376] 图 14B 示出了使用 TOMIPCC 的非常小的系统。蜂窝式便携无线电话语言翻译器的实施例仅由三个芯片构成 :闪存 RAM、TOMIPCC 和单个的 TOMI CPU/DRAM。在这个最小应用中,单个的 TOMI CPU/DRAM 通过由 D0-D7 指示的标准 8 位 DRAM I/O 通信。

[0377] 闪存 RAM 包含语音和语法的字典,该字典定义了口头语和用法以将口头语从一个翻译为另一个。

[0378] TOMIPCC 接收模拟口头语 (或其等同),将其转换为数字表示,并且将其呈现给 TOMI DRAM 用于解释和翻译。所产生的数字化的言语将从 TOMI DRAM 传递回 TOMIPCC,被转换为模拟语音表示,然后输出给蜂窝式便携无线电话用户。

[0379] 图 14C 示出了使用 TOMIPCC 的非常大的系统。这种系统的实施例是以存储器为中心的数据库应用 (或 MCDB)。MCDB 系统工作在高速存储器内的整个数据库上,而不是工作在非常慢的磁盘或存储设备中的存储器分页块。通过在与以存储器为中心的数据库位于相同片上的 TOMI CPU 上执行的所谓的分治算法的使用,TOMI 架构能够进行快速搜索和分类。

[0380] 这种系统可通过合并了多个 TOMI CPU 芯片的 TOMI DIMM(双列直插内存模块) 构造。标准 240 针 DIMM 的数据通路是 64 位。因此,以存储器为中心的数据库应用中的 TOMIPCC 将驱动由 D0-D63 指示的 64 位宽的数据库。

[0381] 可理解,仅示例性地通过实施例并参照附图描述了本发明,本发明不限于本文描述的特定实施方式。本领域的技术人员将认识到,在不背离本发明的范围或精神的情况下,可对本发明和本文描述的示例性实施方式进行改进和修改。

寄存器和数据路径

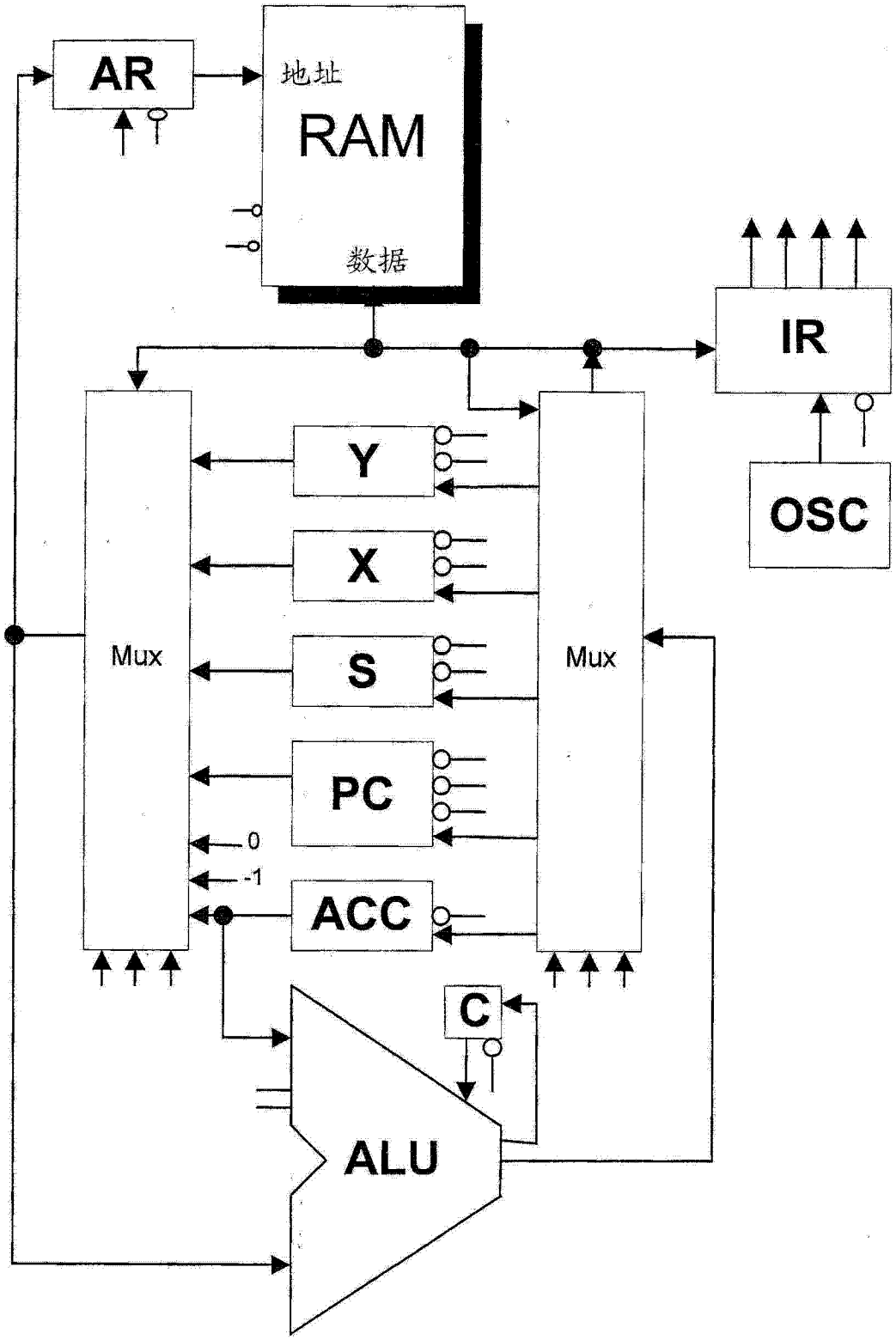


图 1

指令操作码映射

操作码	助记码	操作	进位
00b bbbbb	**LOADI, IMM**	IMM -> ACC	无变化
010 bbbbb	**LOADACC, R**	R -> ACC	无变化
011 bbbbb	**STOREACC, R**	ACC -> R	无变化
100 bbbbb	**ADD, R**	ACC + R -> R	C = 进位
101 bbbbb	**AND, R**	ACC AND R -> R	如果为0，C=1
110 bbbbb	**XOR, R**	ACC XOR R -> R	如果为0，C=1
111 bbbbb	**INC, R**	R + 1 -> R	如果为0，C=1

b-单个二进制位
IMM-6位立即操作数
R-寄存器

图 2

向前分支

	MSB		LSB
PC	XXXXXXXX	XXXXXXXX	XXWWWWWW
ACC	00000000	00000000	00zzzzzz

向前分支发生 ，zzzzzz → wwwwww
如果 wwwwww > zzzzzz,
那么 xxxxxxxx xxxxxxxx xx + 1→ xxxxxxxx xxxxxx xx

图 2A

指令映射

操作码位	765 (000)	765 (001)	765 (010)	765 (011)	765 (100)	765 (101)	765 (110)	765 (111)
43210	LOADI	LOADI	LOADACC	STOREACC	ADD	AND	XOR	INC
00000	短立即操作数	短立即操作数	SAVELOOP	如果C=0, ACC -> PC	如果C=0, ACC+PC->PC	LOOP_IF	NOOP	如果C=0, PC+1->PC
00001	短立即操作数	短立即操作数	SHIFTLOADBYTE	ACC -> PC	ACC+PC->PC	LOOP	SETMSB	PC+1->PC
00010	短立即操作数	短立即操作数	NOTACC	ROTATELEFT8	ACC+ACC->ACC 修改C	测试ACC 修改C	XOR ACC 修改C	INC ACC 修改C
00011	短立即操作数	短立即操作数						
00100	短立即操作数	短立即操作数	GETNEXT-PROCESSOR	SELECTNEXT-PROCESSOR	START-PROCESSOR	TESTDONE	TEST-AVAILABLE	SET-AVAILABLE
00101	短立即操作数	短立即操作数	S->ACC	ACC->(S)	ACC+(S)->ACC	NOOP	NOOP	NOOP
00110	短立即操作数	短立即操作数	S->ACC	ACC->(S)	(S)+ACC->ACC	(S)&ACC->ACC	(S)^ACC->ACC	(S)|ACC->ACC
00111	短立即操作数	短立即操作数	POP	NOOP	NOOP	NOOP	NOOP	NOOP
01000	短立即操作数	短立即操作数	RIGHTSHIFTACC	PUSH	(S)+ACC->ACC S+	(S)&ACC->ACC S+	(S)^ACC->ACC S+	(S)|ACC->ACC S+
01001	短立即操作数	短立即操作数						
01010	短立即操作数	短立即操作数	X->ACC	ACC->X	ACC+X->X	NOOP	NOOP	NOOP
01011	短立即操作数	短立即操作数	在由X指向的X缓存寄存器上操作					
01100	短立即操作数	短立即操作数	X先递减，在由X指向的X缓存寄存器上操作					
01101	短立即操作数	短立即操作数	在由X指向的X缓存寄存器上操作，X后递增					
01110	短立即操作数	短立即操作数	Y->ACC	ACC->Y	ACC+Y->Y	NOOP	NOOP	NOOP
01111	短立即操作数	短立即操作数	在由Y指向的Y缓存寄存器上操作					
10000	短立即操作数	短立即操作数	Y先递减，在由Y指向的Y缓存寄存器上操作					
10001	短立即操作数	短立即操作数	在由Y指向的Y缓存寄存器上操作，Y后递增					
10010	短立即操作数	短立即操作数	共享的变量					
10011	短立即操作数	短立即操作数	通用寄存器					
10100	短立即操作数	短立即操作数						
10101	短立即操作数	短立即操作数						
10110	短立即操作数	短立即操作数						
10111	短立即操作数	短立即操作数						
11000	短立即操作数	短立即操作数						
11001	短立即操作数	短立即操作数						
11010	短立即操作数	短立即操作数						
11011	短立即操作数	短立即操作数						
11100	短立即操作数	短立即操作数						
11101	短立即操作数	短立即操作数						
11110	短立即操作数	短立即操作数						
11111	短立即操作数	短立即操作数						

图 2B

多处理器管理指令集扩展

助记码	操作	进位结果
GETNEXTPROCESSOR	N -> ACC	无变化
SELECTPROCESSOR	ACC -> PSR	无变化
STARTPROCESSOR	ACC -> 处理器[PC]	成功，C=1
TESTDONE	测试处理器[PC]=-1	如果PC=-1,设C=1
TESTAVAILABLE	测试处理器[可用标记]	如果可用，设C=1
SETAVAILABLE	将可用标记设为1	无变化
READSHAREDVARIABLE	处理器[共享的变量]->ACC	无变化
STORESHAREDVARIABLE	ACC ->共享的变量	无变化

注：操作码互相依赖地执行并选自在图2B中说明的可用操作码

b-单个二进制位
IMM-6位立即操作数
R-寄存器
N-32位整数
PSR-处理器选择寄存器
处理器-由处理器选择寄存器指示的处理器

图 2C

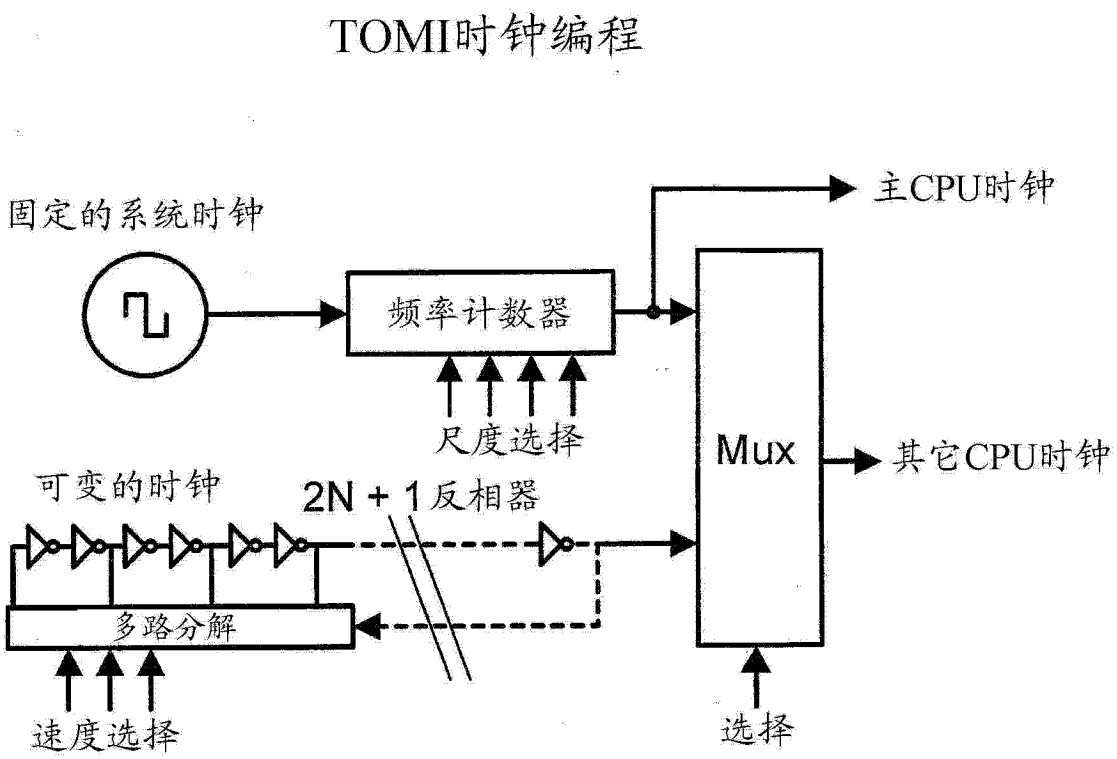

TOMI时钟编程
固定的系统时钟
主CPU时钟
频率计数器
尺度选择
Mux
其它CPU时钟
可变的时钟
2N + 1反相器
多路分解
速度选择
选择

图 2D

指令集扩展

助记码	操作	进位结果
SAVELOOP	PC -> -(S)	无变化
SHIFTLOADBYTE	将ACC向左移位8位, 下一个字节->ACC	无变化
LOOP	(S) -> PC	无变化
LOOP_IF	如果C=0, (S) ->PC	无变化
NOTACC	~ACC -> ACC	如果ACC=0，设C=1
ROTATELEFT8	将ACC向左旋转8位	无变化
ORSTACK	ACC | (S) -> ACC	如果ACC=0，设C=1
ORSTACK+	ACC | (S)+ -> ACC	如果ACC=0，设C=1
RIGHTSHIFTACC	将ACC向右移位	LSB -> C
SETMSB	2^31 -> ACC	无变化

注：操作码互相依赖地执行并选自在图2B中说明的可用操作码

b-单个二进制位
IMM- 6位立即操作数
R-寄存器
N-32位整数
PSR-处理器选择寄存器
处理器-由处理器选择寄存器指示的处理器

图 2E

有效地址

寄存器	名称	有效操作数	周期	具体操作
00000	PC0	PC+3用于读，否则PC	1	写的条件是C=0
00001	PC1	加载8位立即操作数读，否则PC	1	
00010	ACC	ACC	1	
00011				
00100	S	S	1	
00101	(S)	通过S寻址的数据	2	
00110	-(S)	(S)先递减	2	
00111	(S)+	(S)后递增	2	
01000	X	X	1	
01001	(X)	通过X寻址的数据	2	
01010	-(X)	(X)先递减	2	
01011	(X)+	(X)后递增	2	
01100	Y	Y	1	
01101	(Y)	通过Y寻址的数据	2	
01110	-(Y)	(Y)先递减	2	
01111	(Y)+	(Y)后递增	2	
10000	R10	寄存器10	1	
10001	R11	寄存器11	1	
10010	R12	寄存器12	1	
10011	R13	寄存器13	1	
10100	R14	寄存器14	1	
10101	R15	寄存器15	1	
10110	R16	寄存器16	1	
10110	R17	寄存器17	1	
10111	R18	寄存器18	1	
11000	R19	寄存器19	1	
11001	R1A	寄存器1A	1	
11010	R1B	寄存器1B	1	
11011	R1C	寄存器1C	1	
11100	R1D	寄存器1D	1	
11101	R1E	寄存器1E	1	
11111	R1F	寄存器1F	1	

图 3

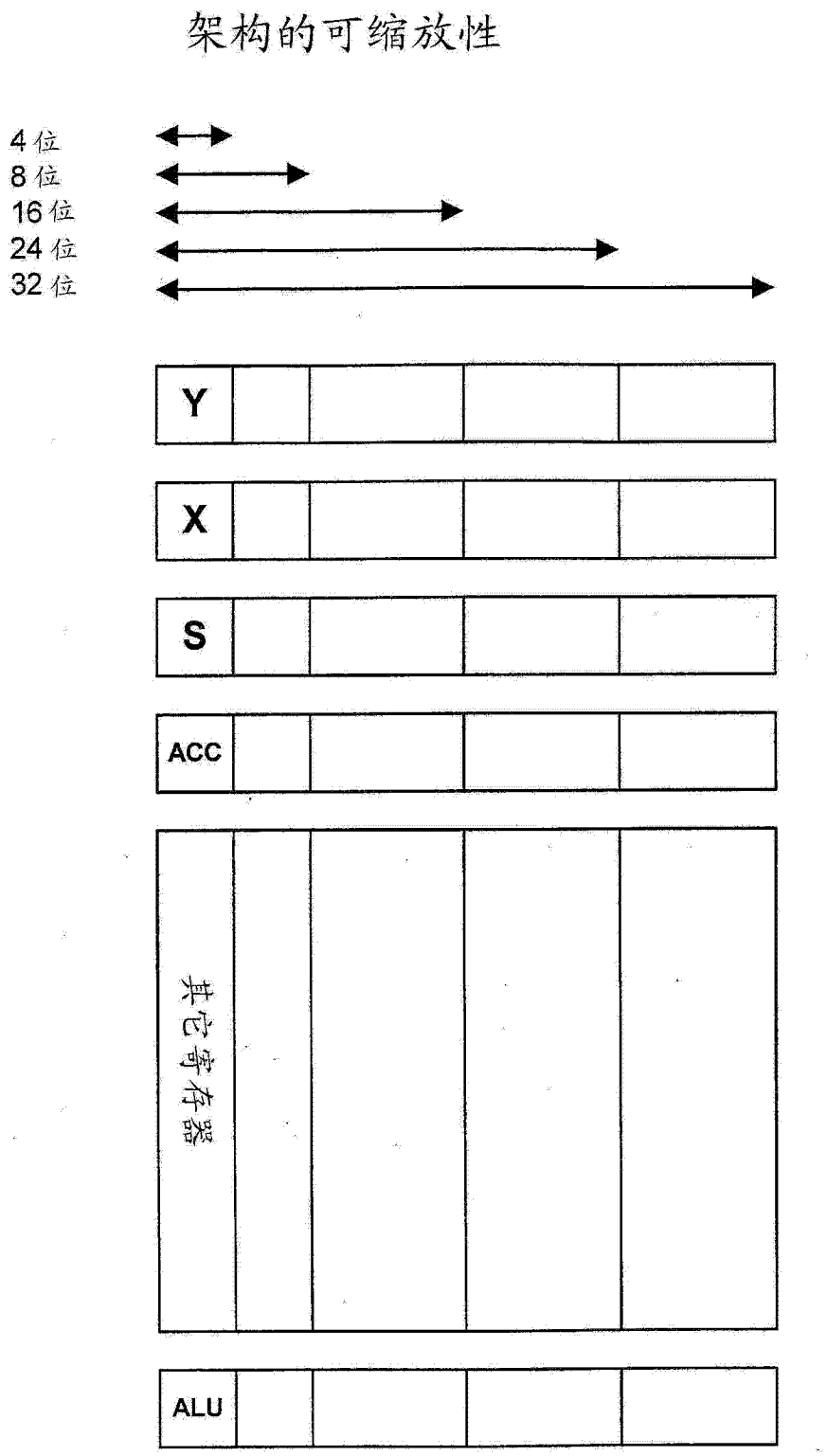


图 4

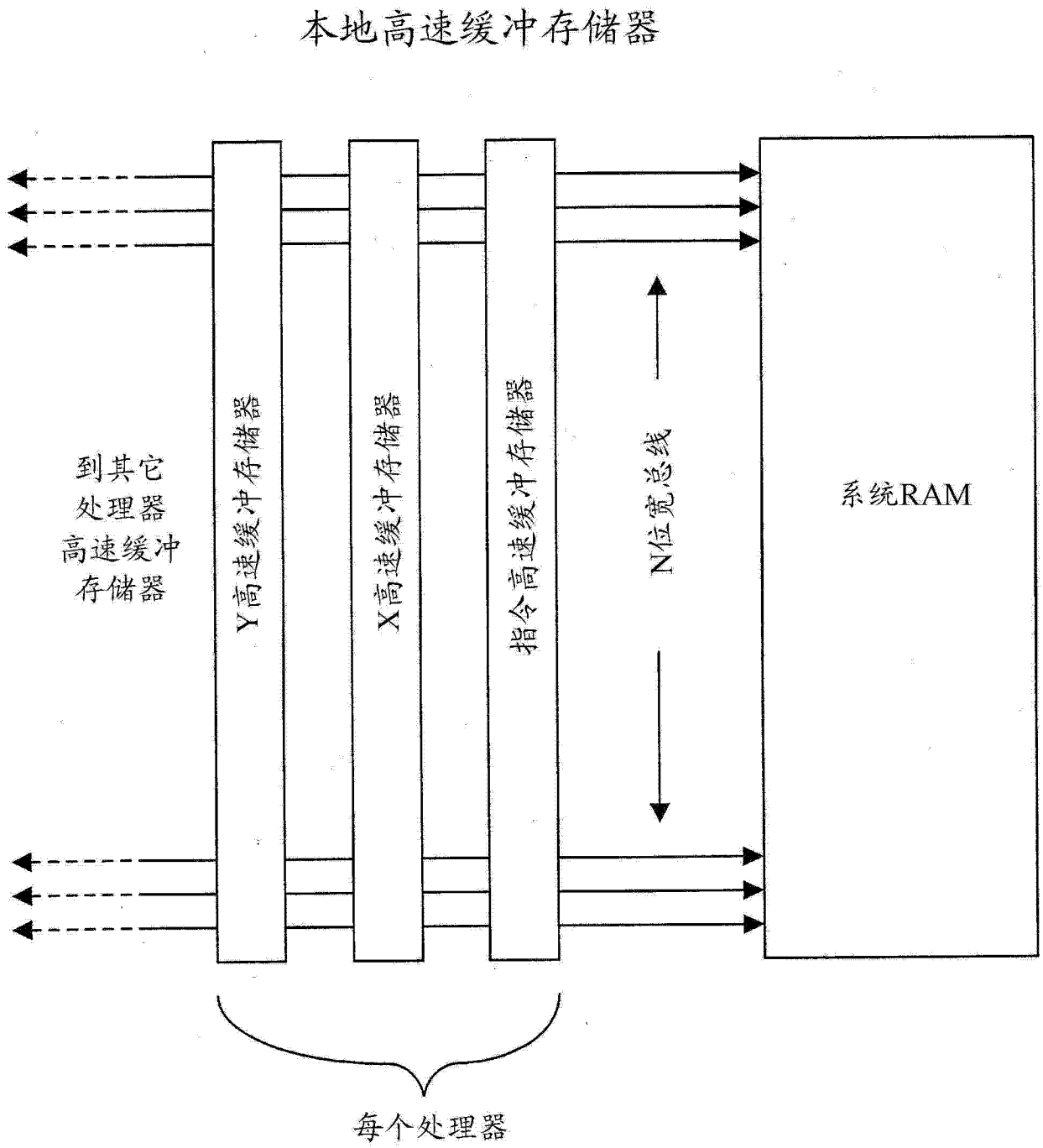

本地高速缓冲存储器
到其它
处理器
高速缓冲
存储器
Y高速缓冲存储器
X高速缓冲存储器
指令高速缓冲存储器
N位宽总线
系统RAM
每个处理器

图 5

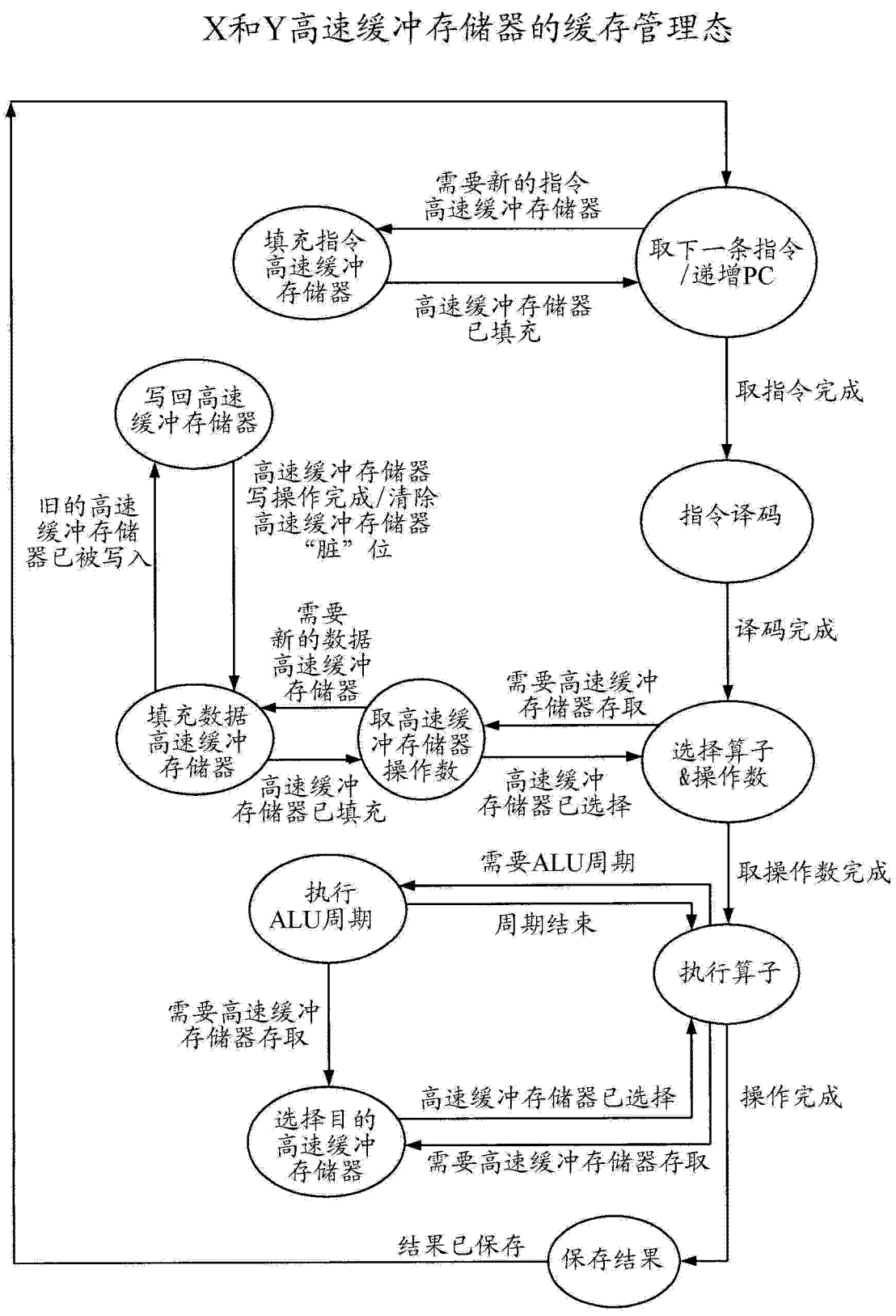

X和Y高速缓冲存储器的缓存管理态
需要新的指令
高速缓冲存储器
填充指令
高速缓冲
存储器
取下一条指令
/递增PC
高速缓冲存储器
已填充
写回高速
缓冲存储器
取指令完成
高速缓冲存储器
写操作完成/清除
高速缓冲存储器
“脏”位
旧的高速
缓冲存储
器已被写入
指令译码
译码完成
需要
新的数据
高速缓冲
存储器
需要高速缓冲
存储器存取
填充数据
高速缓冲
存储器
取高速缓
冲存储器
操作数
选择算子
&操作数
高速缓冲
存储器已填充
高速缓冲
存储器已选择
取操作数完成
需要ALU周期
执行
ALU周期
周期结束
执行算子
需要高速缓冲
存储器存取
高速缓冲存储器已选择
操作完成
选择目的
高速缓冲
存储器
需要高速缓冲存储器存取
结果已保存
保存结果

图 6

额外的功能性

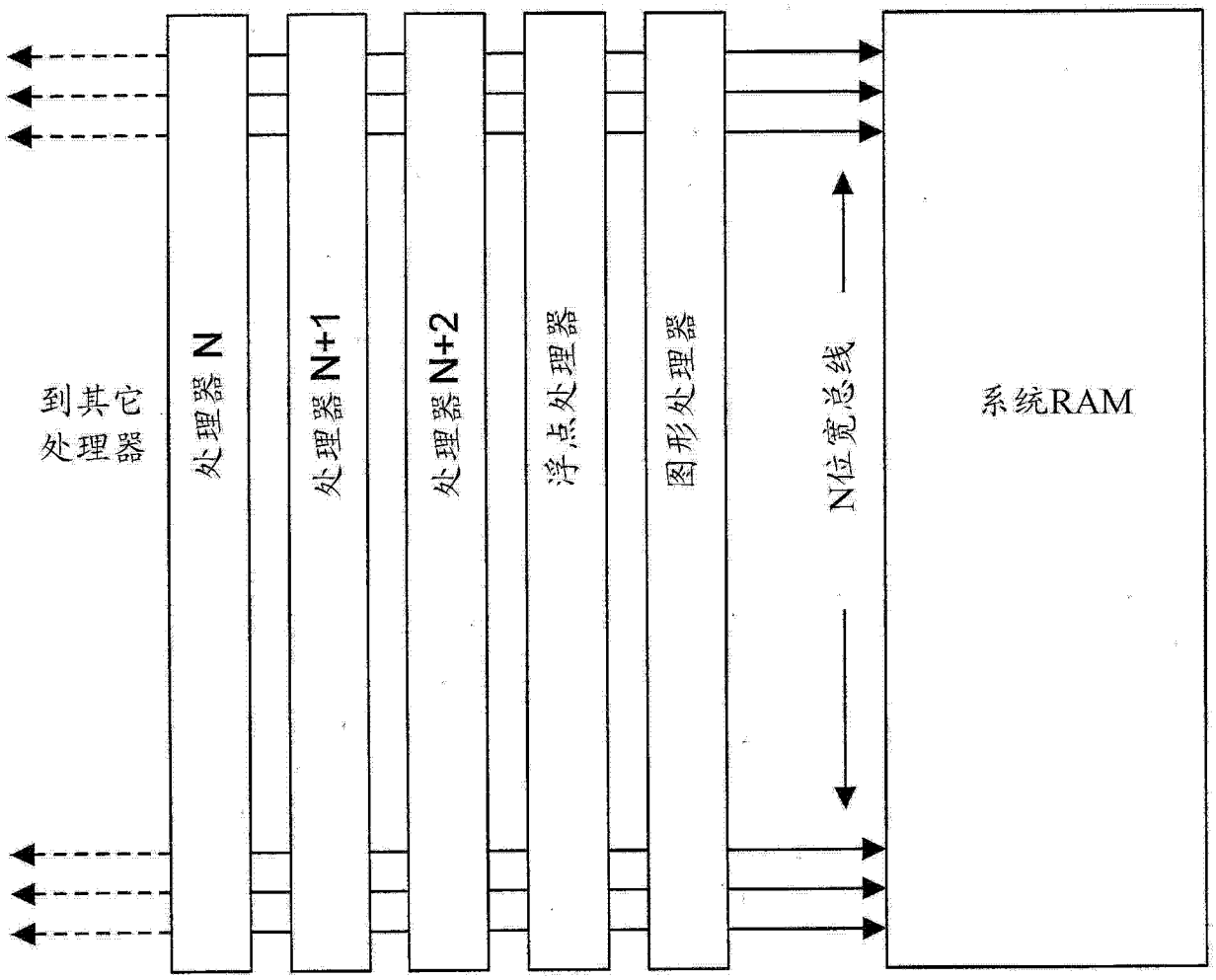


图 7A

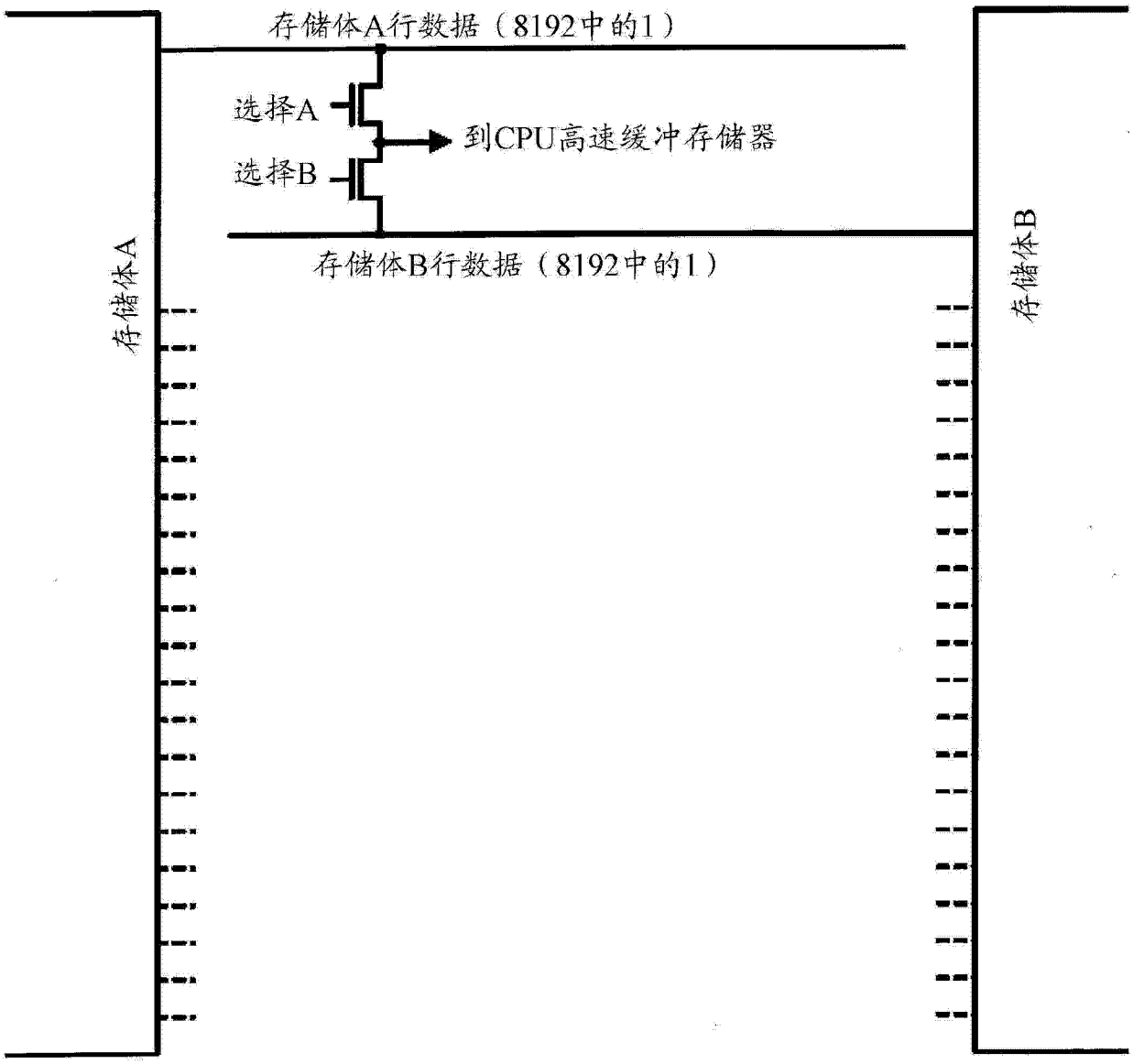

存储体A行数据（8192中的1）
选择A
选择B
到CPU高速缓冲存储器
存储体B行数据（8192中的1）
存储体A
存储体B
共享存储体

图 7B

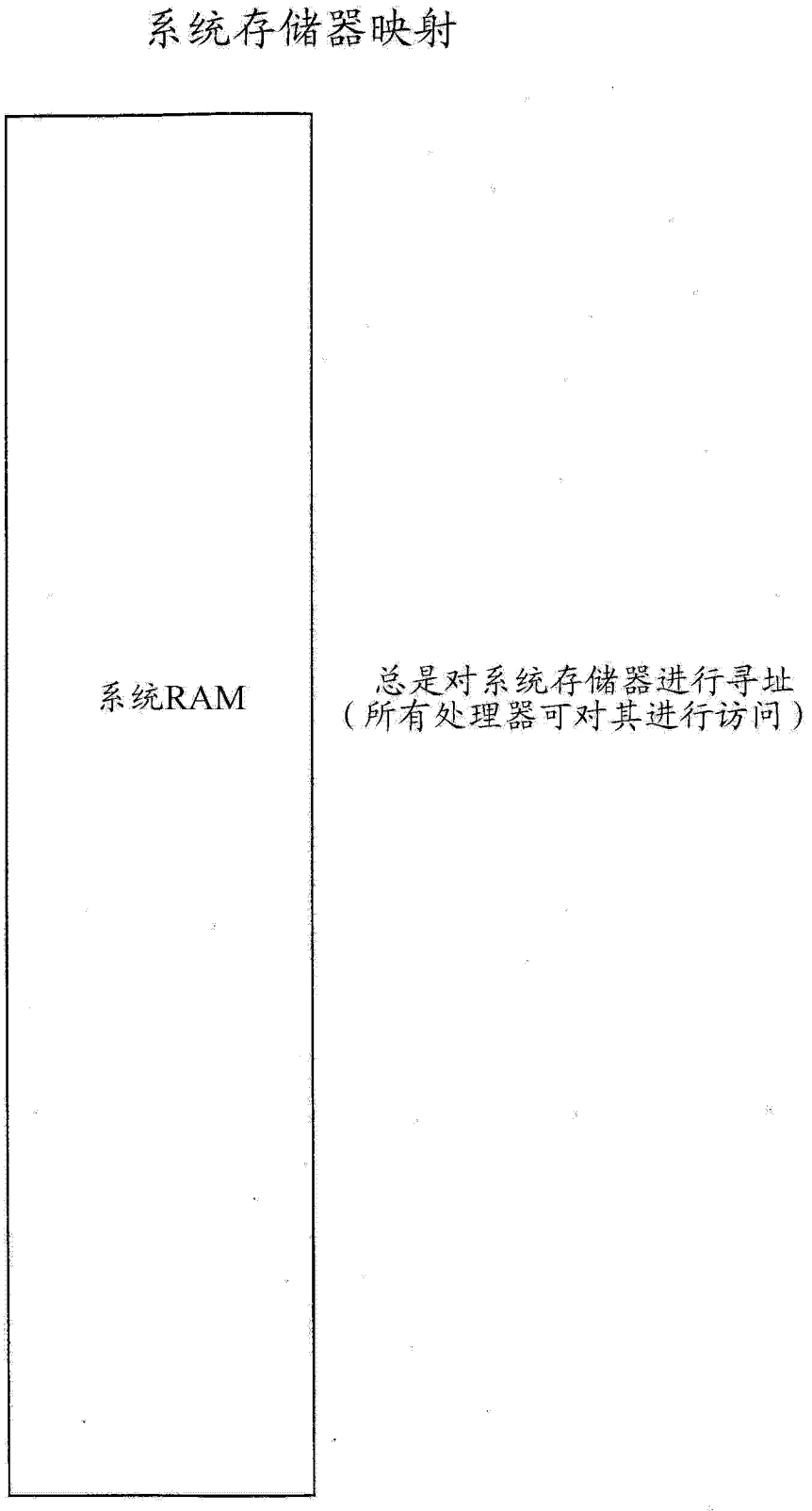


图 8

处理器可用性表

处理器	可用标志	已完成标志	中断向量	起始时间	错误标志
3F					
3E					
3D					
3C					
3B					
3A					
30					
2F					
2E					
					
08					
07					
06					
05					
04					
03					
02					
01					

图 9

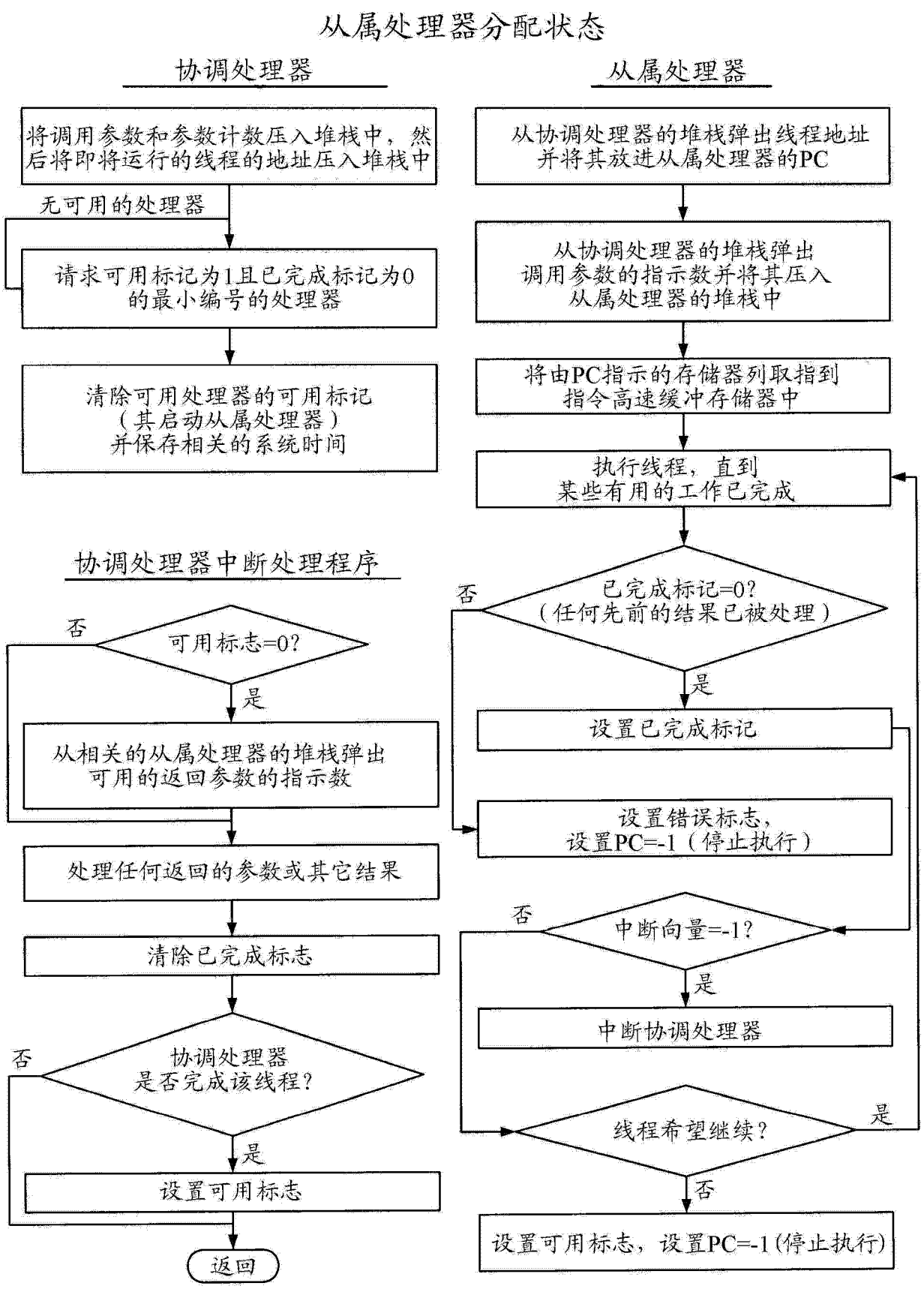

从属处理器分配状态
协调处理器
将调用参数和参数计数压入堆栈中，然后将即将运行的线程的地址压入堆栈中
无可用的处理器
请求可用标记为1且已完成标记为0的最小编号的处理器
清除可用处理器的可用标记（其启动从属处理器）并保存相关的系统时间
从属处理器
从协调处理器的堆栈弹出线程地址并将其放进从属处理器的PC
从协调处理器的堆栈弹出调用参数的指示数并将其压入从属处理器的堆栈中
将由PC指示的存储器列取指到指令高速缓冲存储器中
执行线程，直到某些有用的工作已完成
已完成标记=0?（任何先前的结果已被处理）
否
是
设置已完成标记
设置错误标志，设置PC=-1（停止执行）
中断向量=-1?
否
是
中断协调处理器
线程希望继续?
是
否
设置可用标志，设置PC=-1(停止执行)
协调处理器中断处理程序
可用标志=0?
否
是
从相关的从属处理器的堆栈弹出可用的返回参数的指示数
处理任何返回的参数或其它结果
清除已完成标志
协调处理器是否完成该线程?
否
是
设置可用标志
返回

图 10

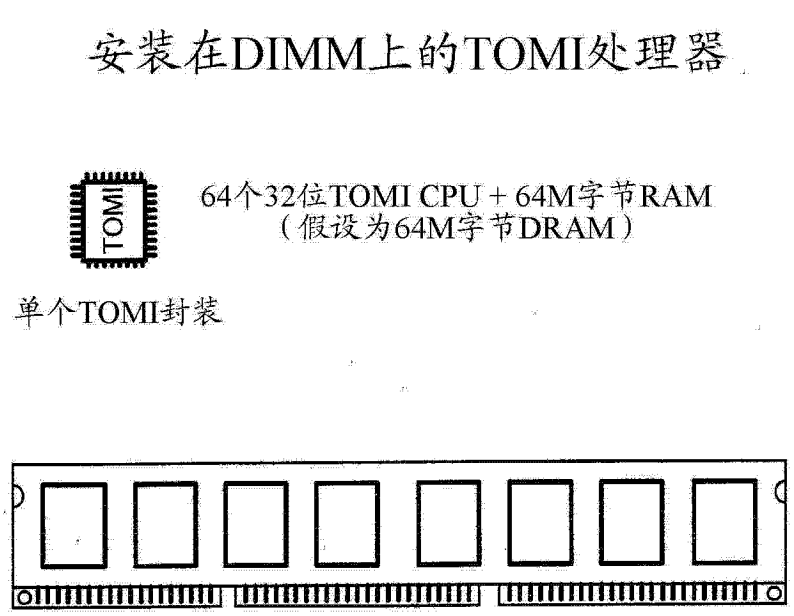

安装在DIMM上的TOMI处理器
TOMI
64个32位TOMI CPU + 64M字节RAM
（假设为64M字节DRAM）
单个TOMI封装
DIMM上的16个TOMI封装
1024个32位TOMI CPU + 1G字节RAM

图 10A

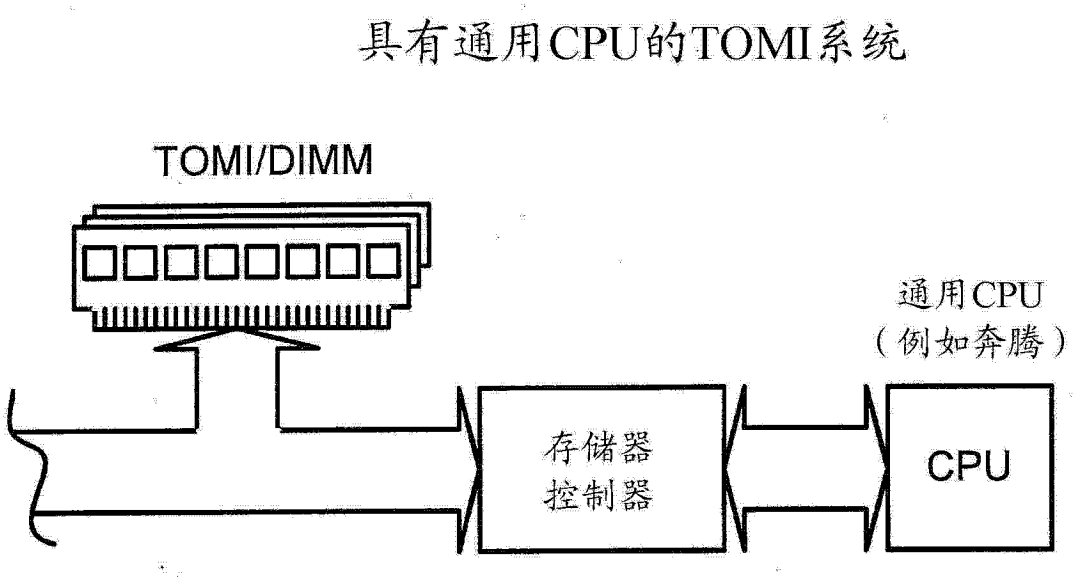

具有通用CPU的TOMI系统
TOMI/DIMM
通用CPU
（例如奔腾）
存储器
控制器
CPU

图 10B

TOMI处理器初始化

1.通过存储器模式寄存器启用TOMI CPU

2.复位配置为:
-除了主处理器以外的所有处理器停止,
-主处理器时钟设为低速,
-主处理器PC设为0,
-主处理器指令高速缓冲存储器加载包含上电自检（POST）的地址0和启动执行地址。

3.主处理器执行上电自检。

4.主处理器确定其最佳的时钟速度。

5.主处理器:
-单独启动每个从属处理器,
-以低速测试,
-停止处理器。

6. 主处理器:
-单独启动每个从属处理器,
-为每个从属处理器确定最佳的时钟速度,
-停止处理器。

7. 主处理器在启动执行地址处开始执行。

图 10C

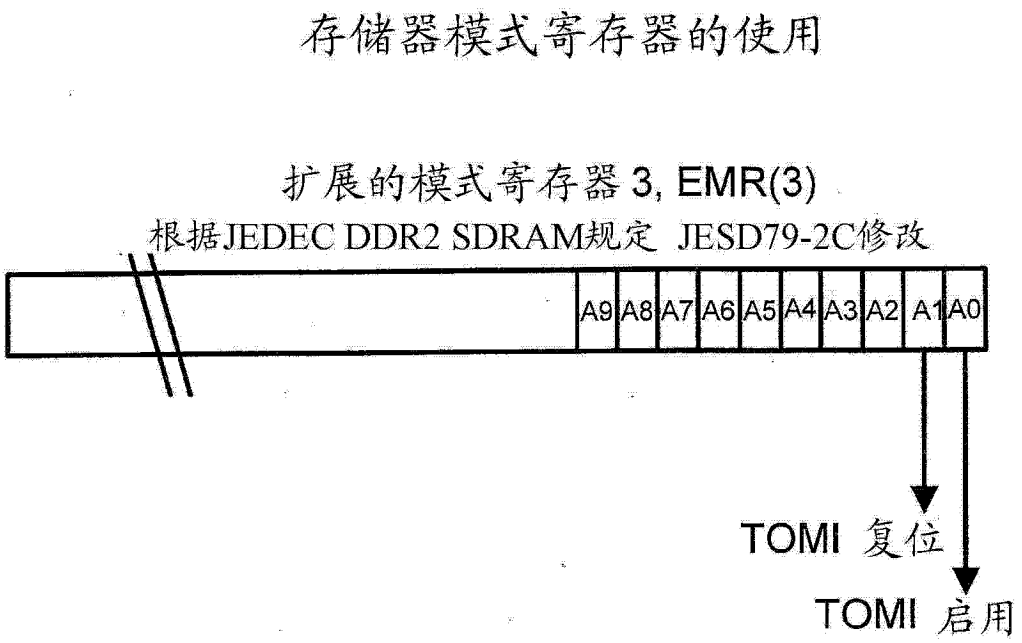

存储器模式寄存器的使用
扩展的模式寄存器 3, EMR(3)
根据JEDEC DDR2 SDRAM规定 JESD79-2C修改
A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
TOMI 复位
TOMI 启用

图 10D

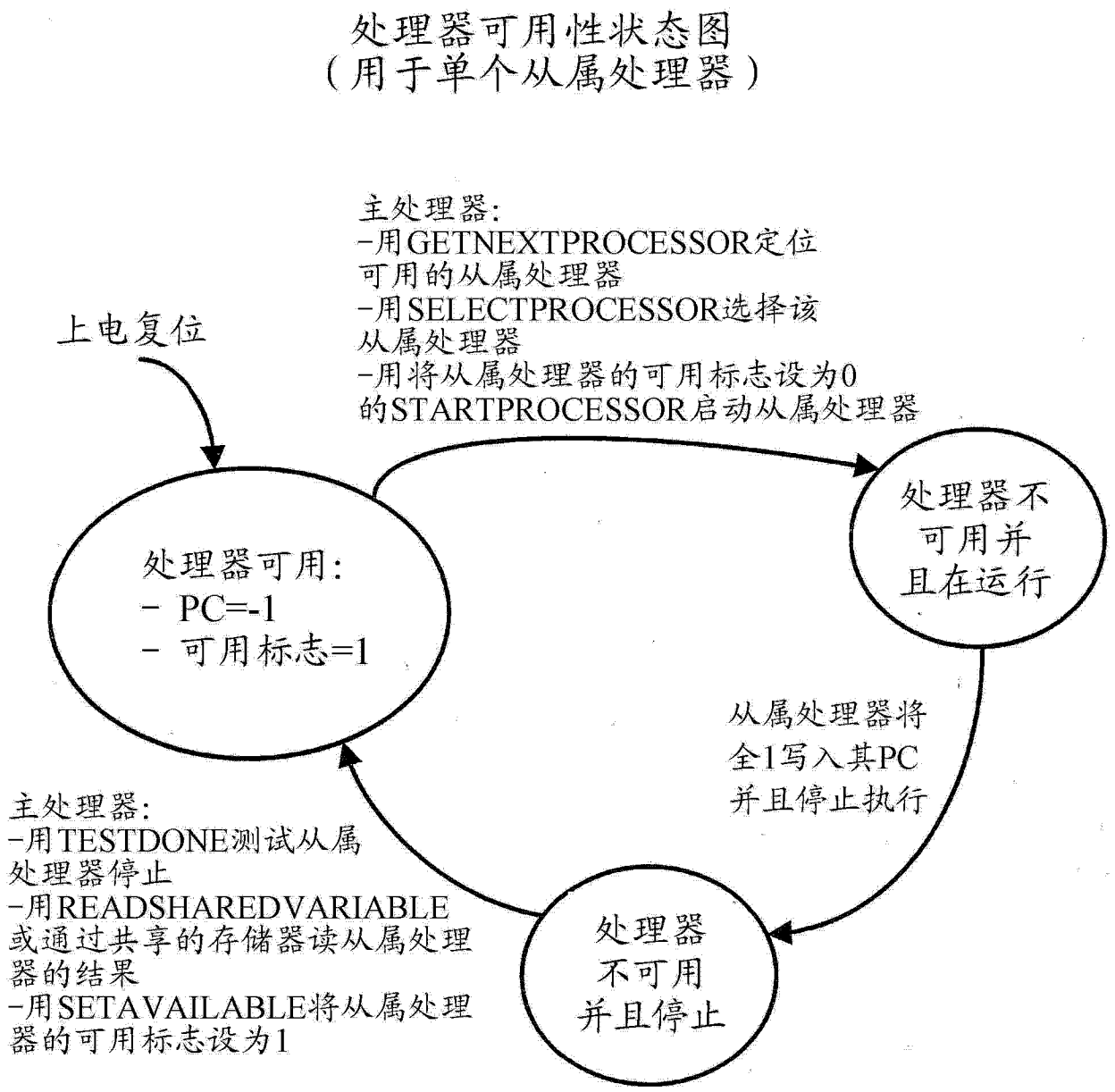

处理器可用性状态图
（用于单个从属处理器）
主处理器:
-用GETNEXTPROCESSOR定位
可用的从属处理器
-用SELECTPROCESSOR选择该
从属处理器
-用将从属处理器的可用标志设为0
的STARTPROCESSOR启动从属处理器
上电复位
处理器可用:
- PC=-1
- 可用标志=1
处理器不
可用并
且在运行
从属处理器将
全1写入其PC
并且停止执行
处理器
不可用
并且停止
主处理器:
-用TESTDONE测试从属
处理器停止
-用READSHAREDVARIABLE
或通过共享的存储器读从属处理
器的结果
-用SETAVAILABLE将从属处理
器的可用标志设为1

图 10E

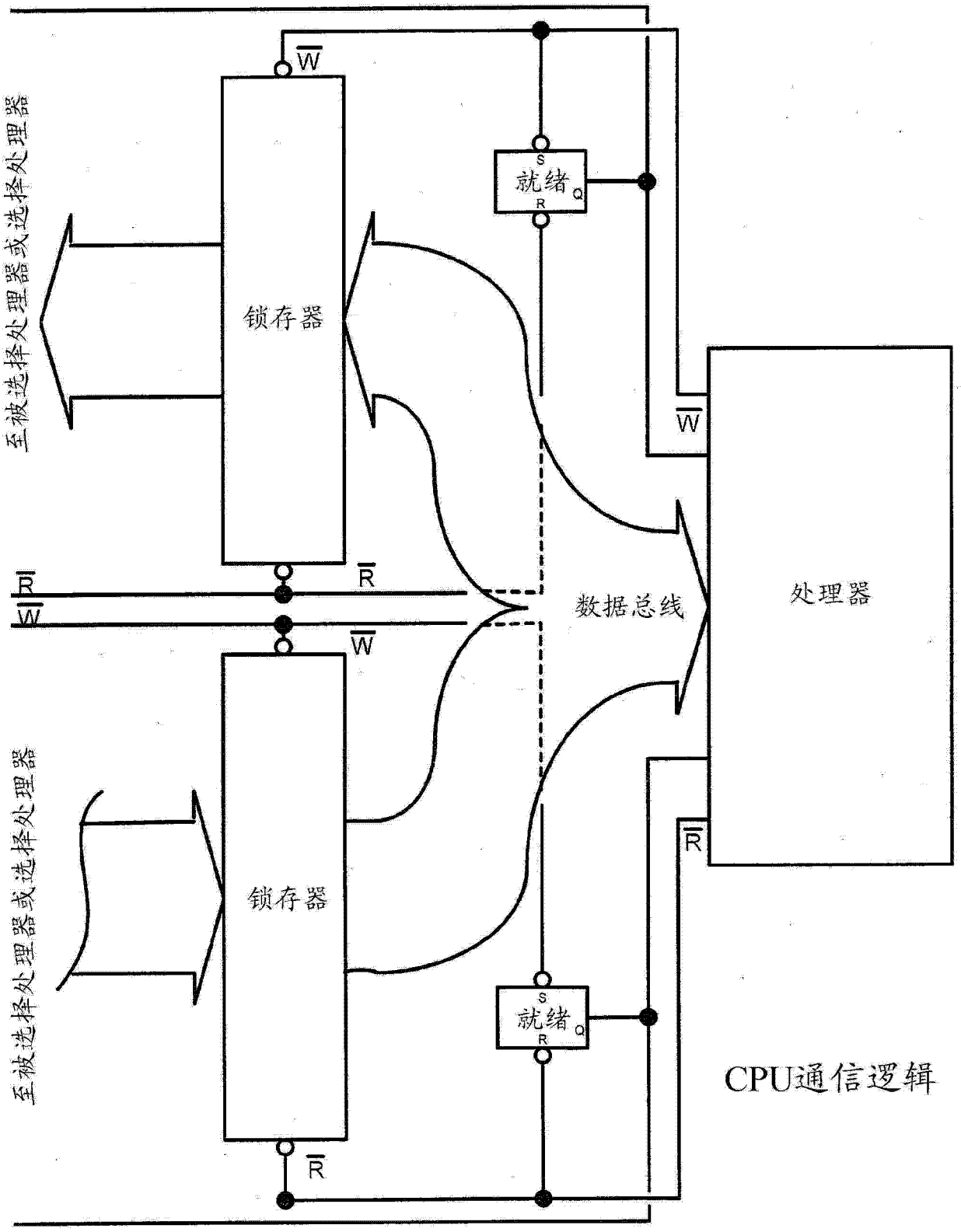

至被选择处理器或选择处理器
锁存器
S
就绪
R
Q
W
R
W
R
数据总线
处理器
至被选择处理器或选择处理器
锁存器
S
就绪
R
Q
R
CPU通信逻辑

图 10F

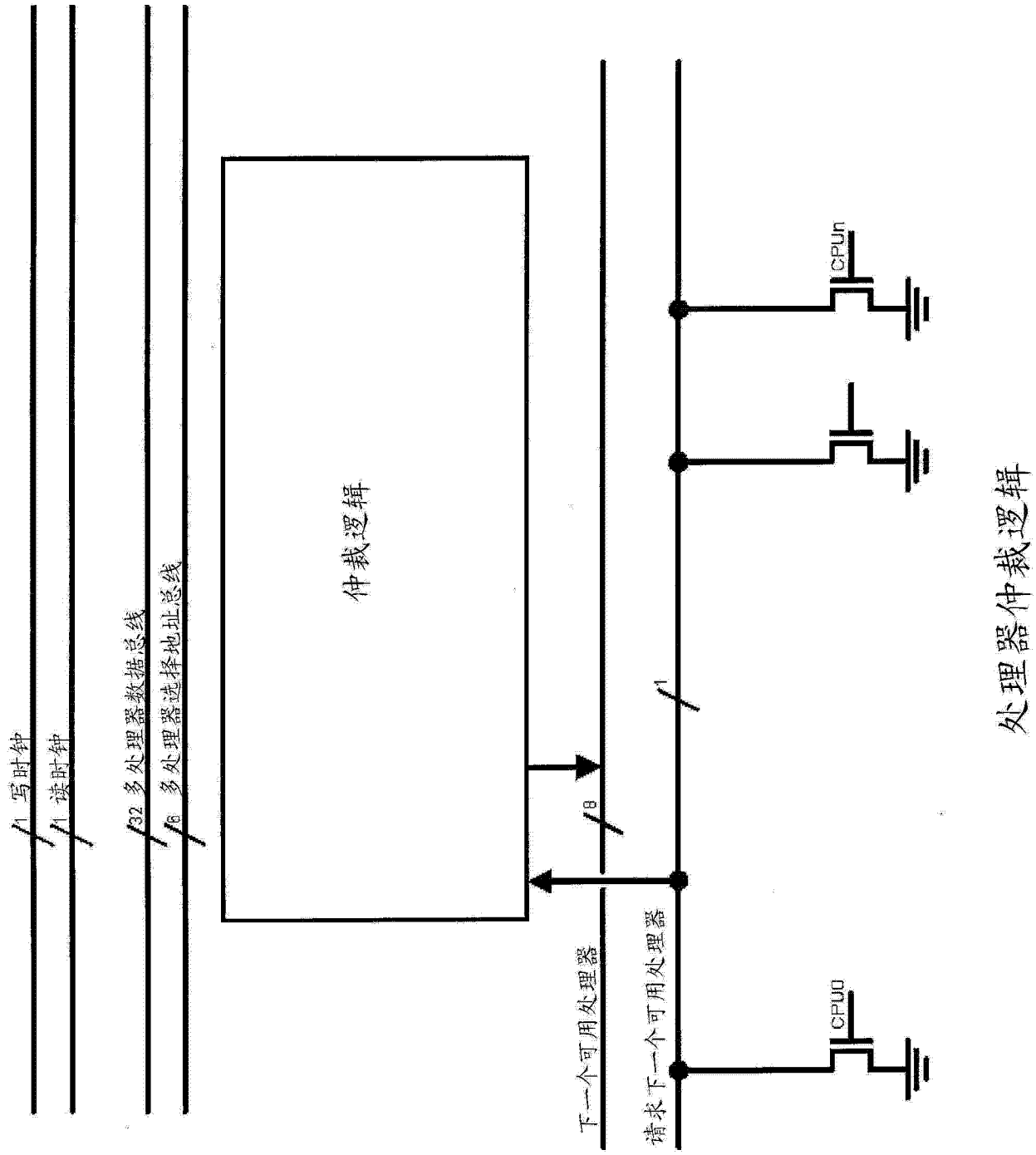

写时钟
读时钟
32 多处理器数据总线
6 多处理器选择地址总线
仲裁逻辑
8
下一个可用处理器
请求下一个可用处理器
CPU0
CPUn
处理器仲裁逻辑

图 10G

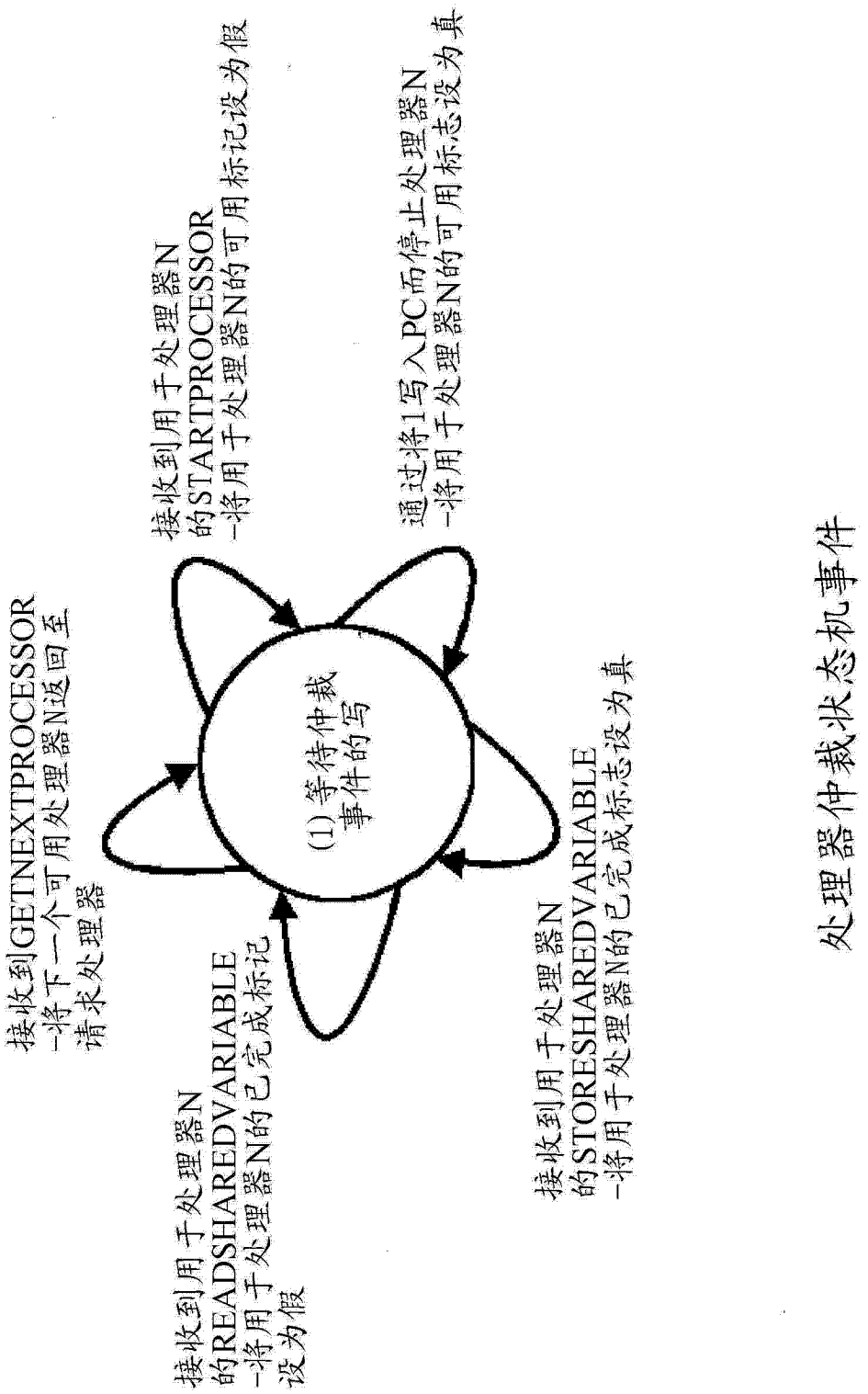

接收到GETNEXTPROCESSOR
-将下一个可用处理器N返回至
请求处理器
接收到用于处理器N
的STARTPROCESSOR
-将用于处理器N的可用标记设为假
通过将1写入PC而停止处理器N
-将用于处理器N的可用标志设为真
(1) 等待仲裁
事件的写
接收到用于处理器N
的READSHAREDVARIABLE
-将用于处理器N的已完成标记
设为假
接收到用于处理器N
的STORESHAREDVARIABLE
-将用于处理器N的已完成标志设为真
处理器仲裁状态机事件

图 10H

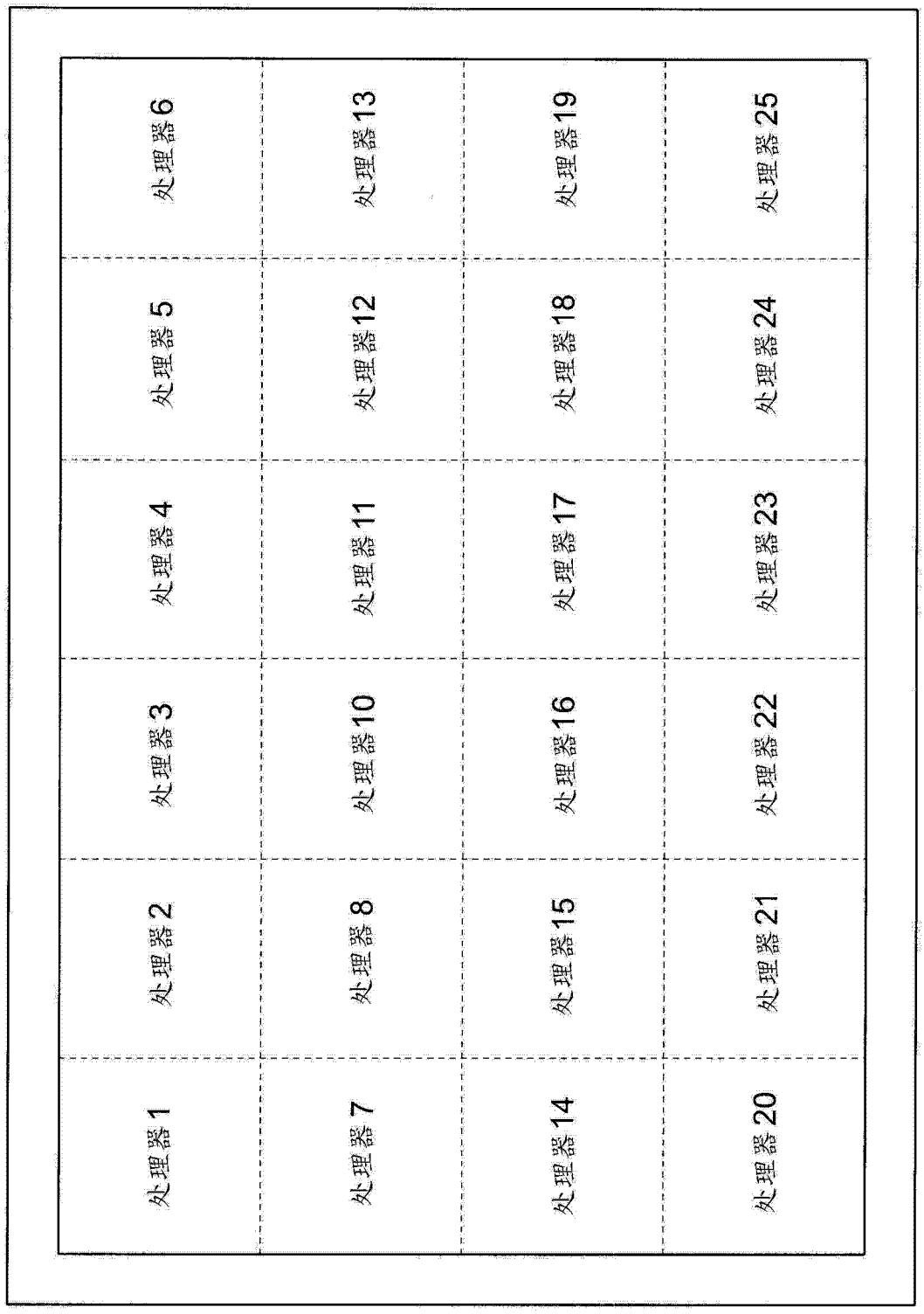

图像映射的分解计算
处理器 1
处理器 2
处理器 3
处理器 4
处理器 5
处理器 6
处理器 7
处理器 8
处理器 10
处理器 11
处理器 12
处理器 13
处理器 14
处理器 15
处理器 16
处理器 17
处理器 18
处理器 19
处理器 20
处理器 21
处理器 22
处理器 23
处理器 24
处理器 25

图 11


分解计算存储器映射
X(25)
处理器25
捕捉RAM
Y(25)
处理器25
结果RAM
X(24)
处理器24
捕捉RAM
Y(24)
处理器24
结果RAM
X(23)
处理器23
捕捉RAM
Y(23)
处理器23
结果RAM
X(22)
处理器22
捕捉RAM
Y(22)
处理器22
结果RAM
X(21)
处理器21
捕捉RAM
Y(21)
处理器21
结果RAM
X(20)
处理器20
捕捉RAM
Y(20)
处理器20
结果RAM
X(1)
处理器1
捕捉RAM
Y(1)
处理器1
捕捉RAM

图 12

64处理器系统的可能的平面布置图

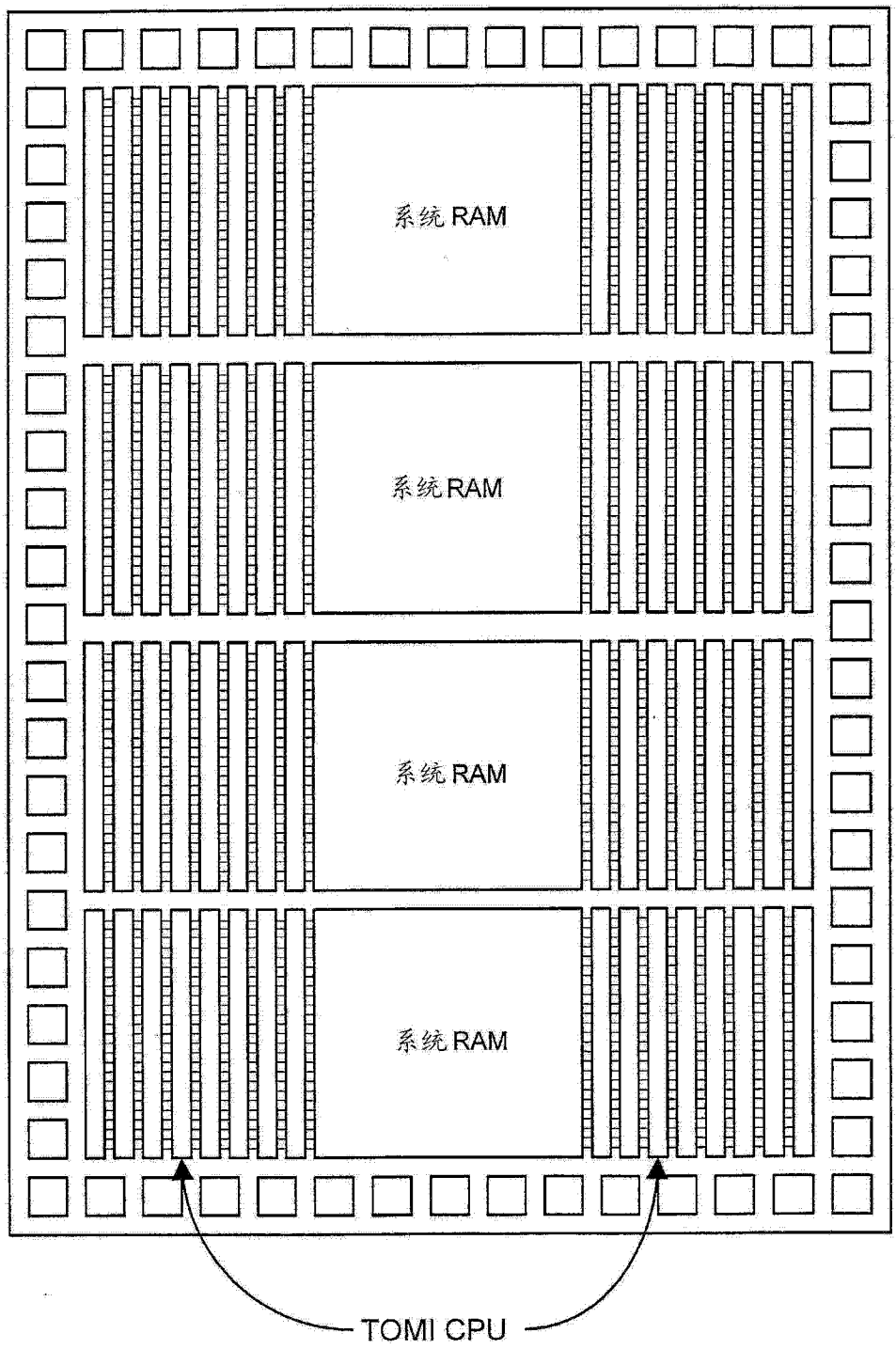


图 13A

另一可能的平面布置图

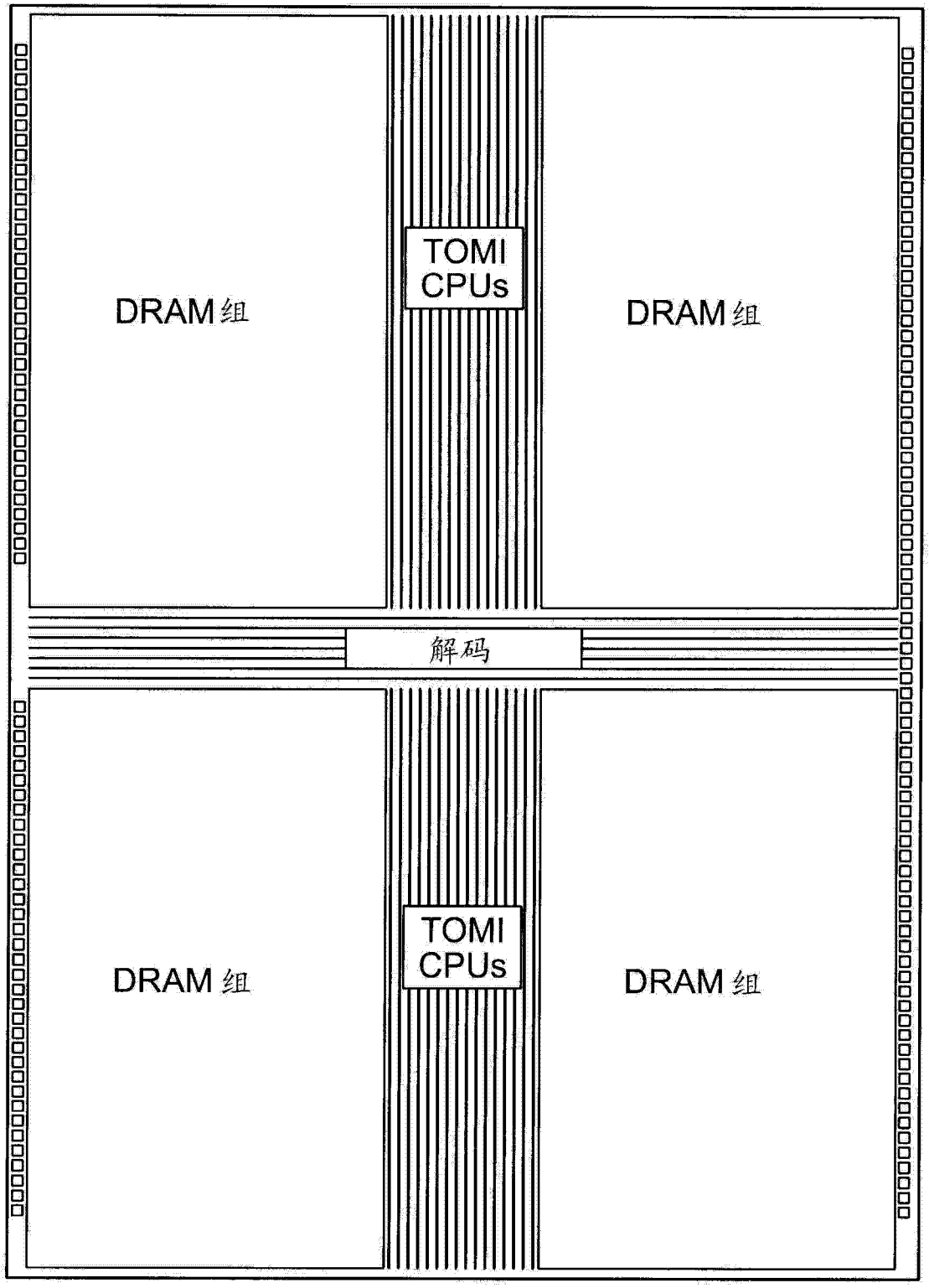
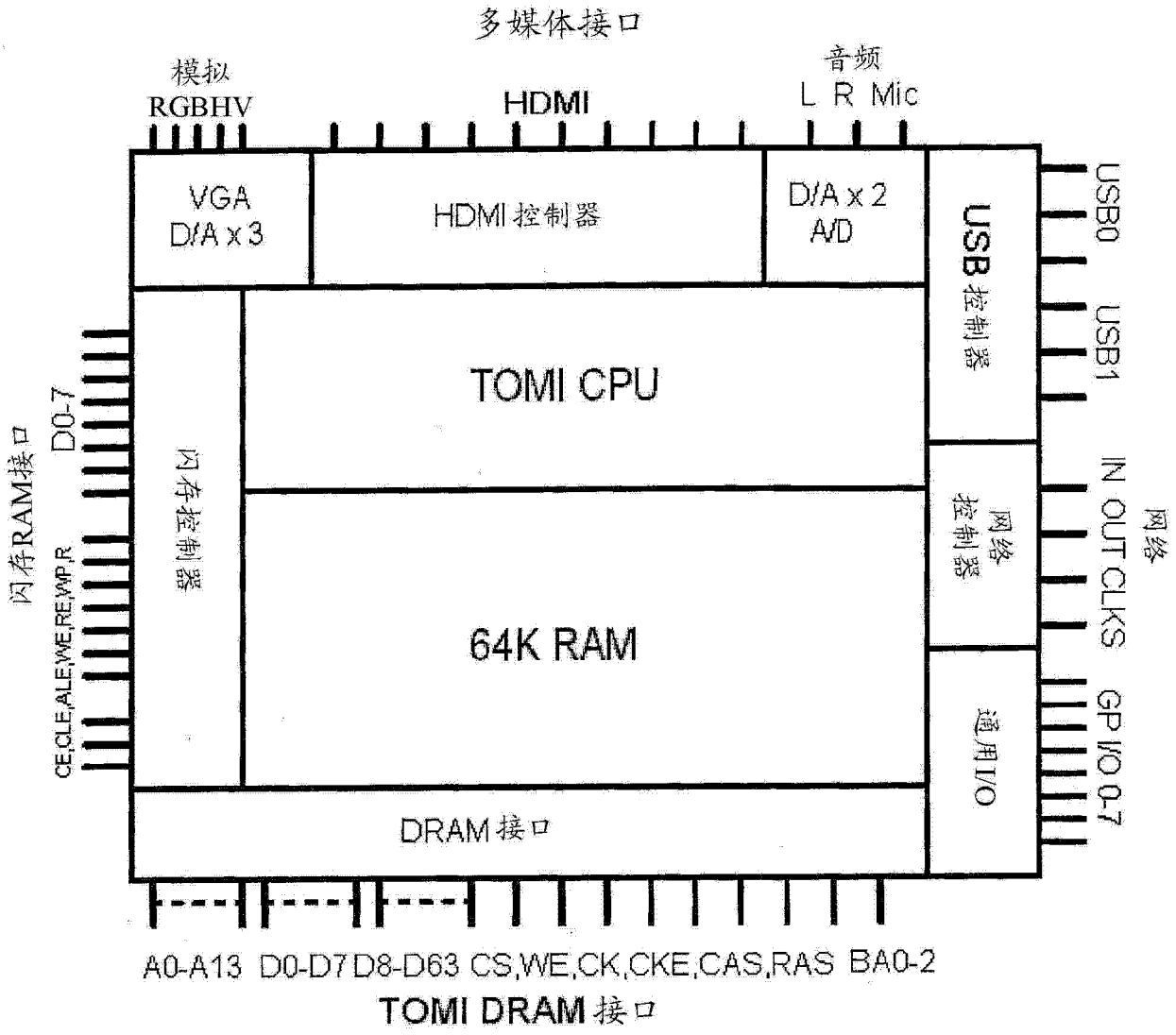
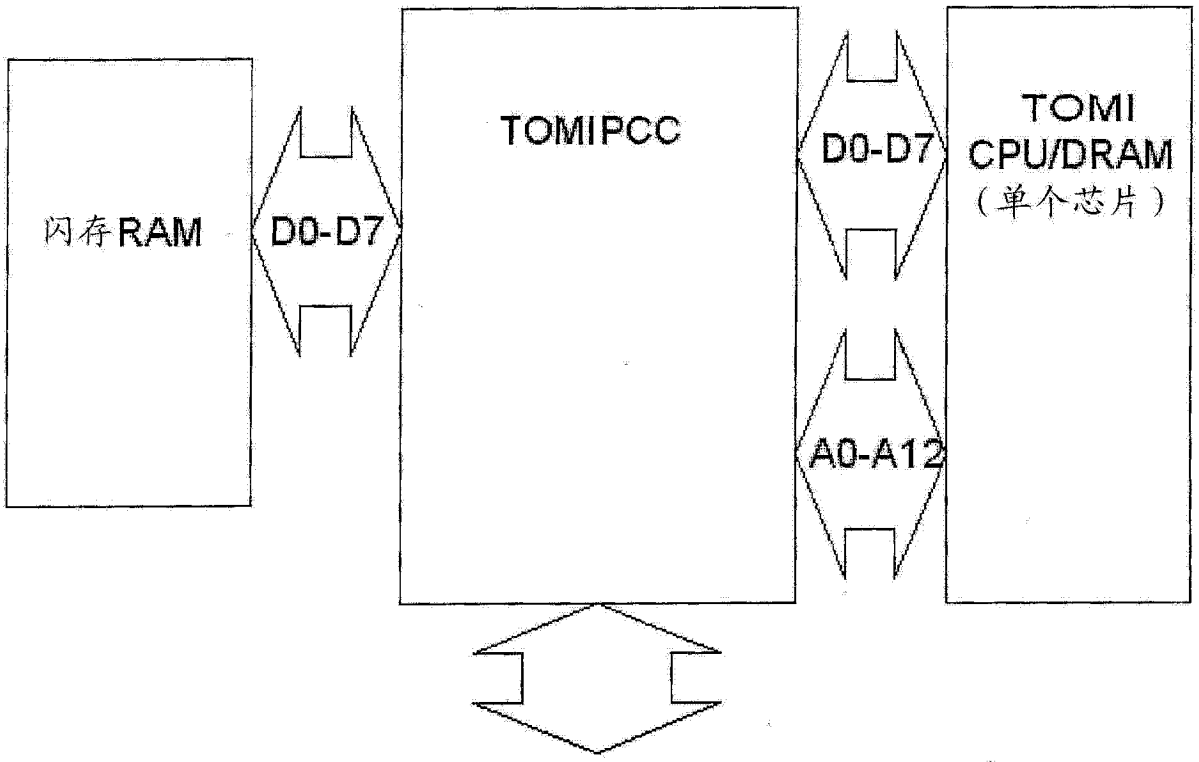


图 13B



TOMI外围控制器芯片

图 14A



蜂窝式便携无线电话语言翻译器

图 14B

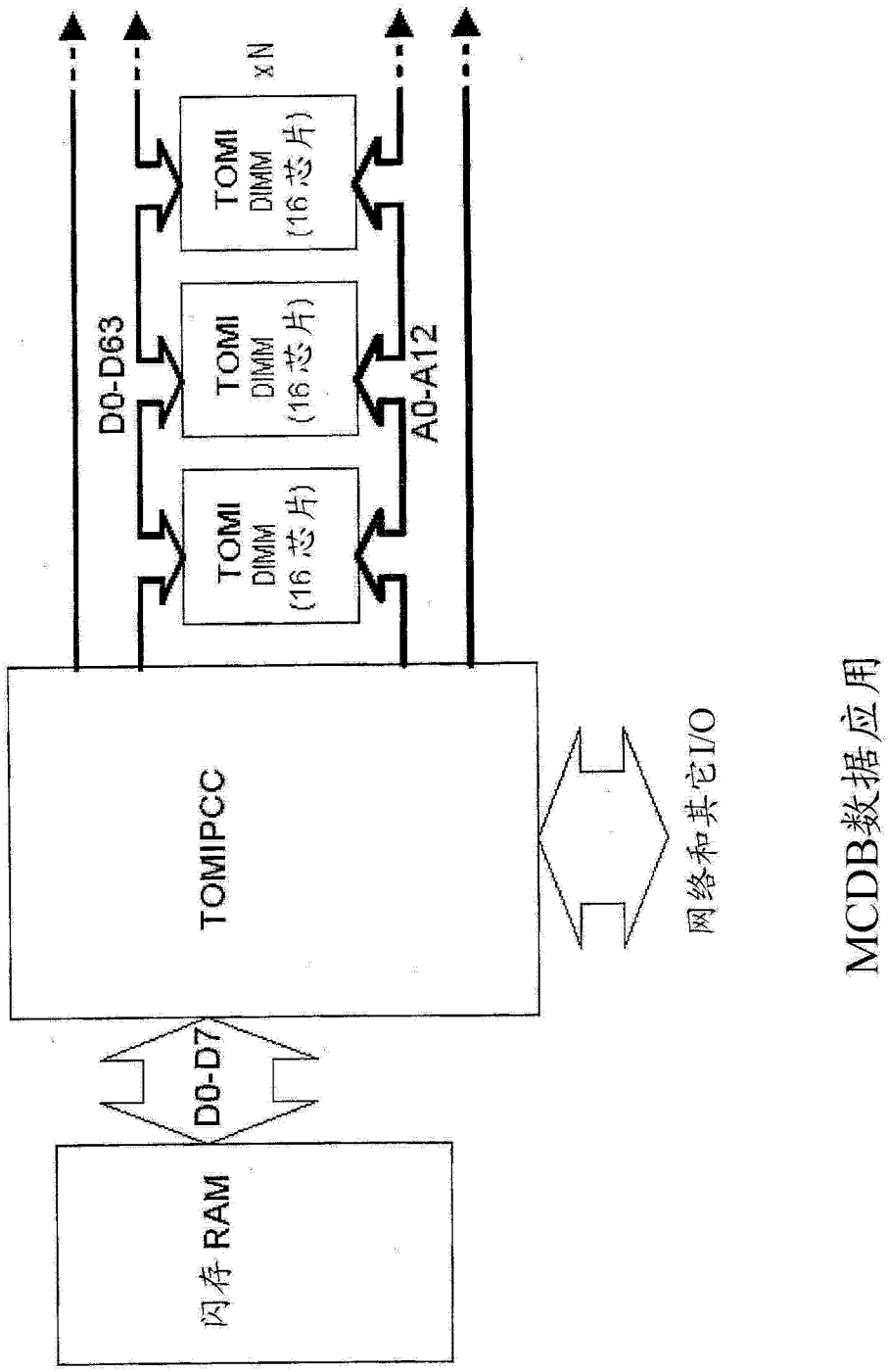

TOMI DIMM (16 芯片)
x N
D0-D63
A0-A12
TOMIPCC
网络和其它I/O
MCDB数据应用
D0-D7
闪存 RAM

图 14C

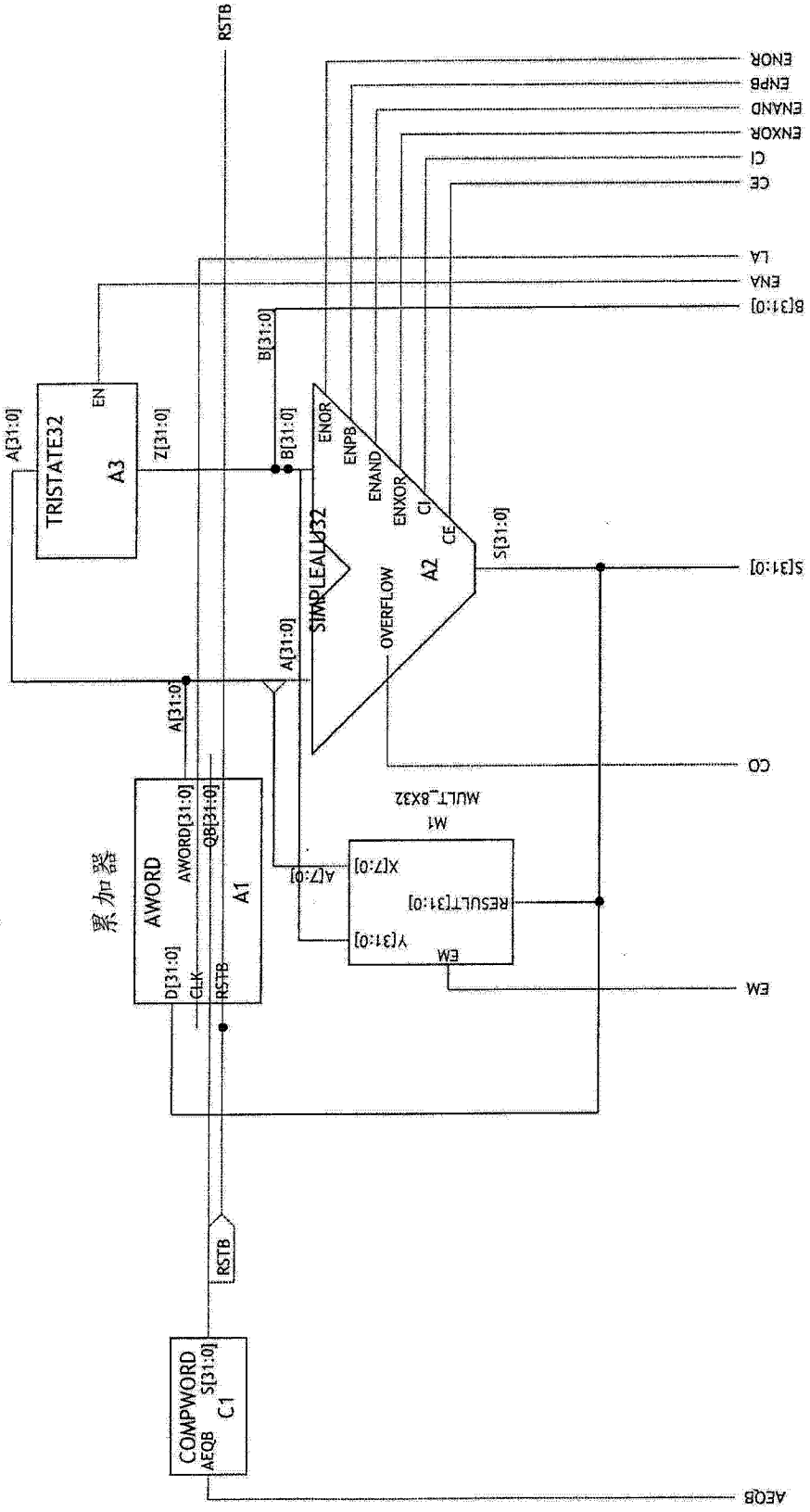

RSTB
ENOR
ENPB
ENAND
ENXOR
CI
CE
LA
ENA
B[31:0]
A[31:0]
TRISTATE32
EN
A3
Z[31:0]
B[31:0]
ENOR
ENPB
ENAND
ENXOR
CI
CE
SIMPLEALU32
A2
OVERFLOW
S[31:0]
S[31:0]
A[31:0]
A[31:0]
CO
累加器
AWORD
AWORD[31:0]
QB[31:0]
D[31:0]
CLK
RSTB
A1
MULT_8X32
M1
A[7:0]
X[7:0]
RESULT[31:0]
Y[31:0]
EM
EM
RSTB
COMPWORD
S[31:0]
AEQB
C1
AEQB

图 15A

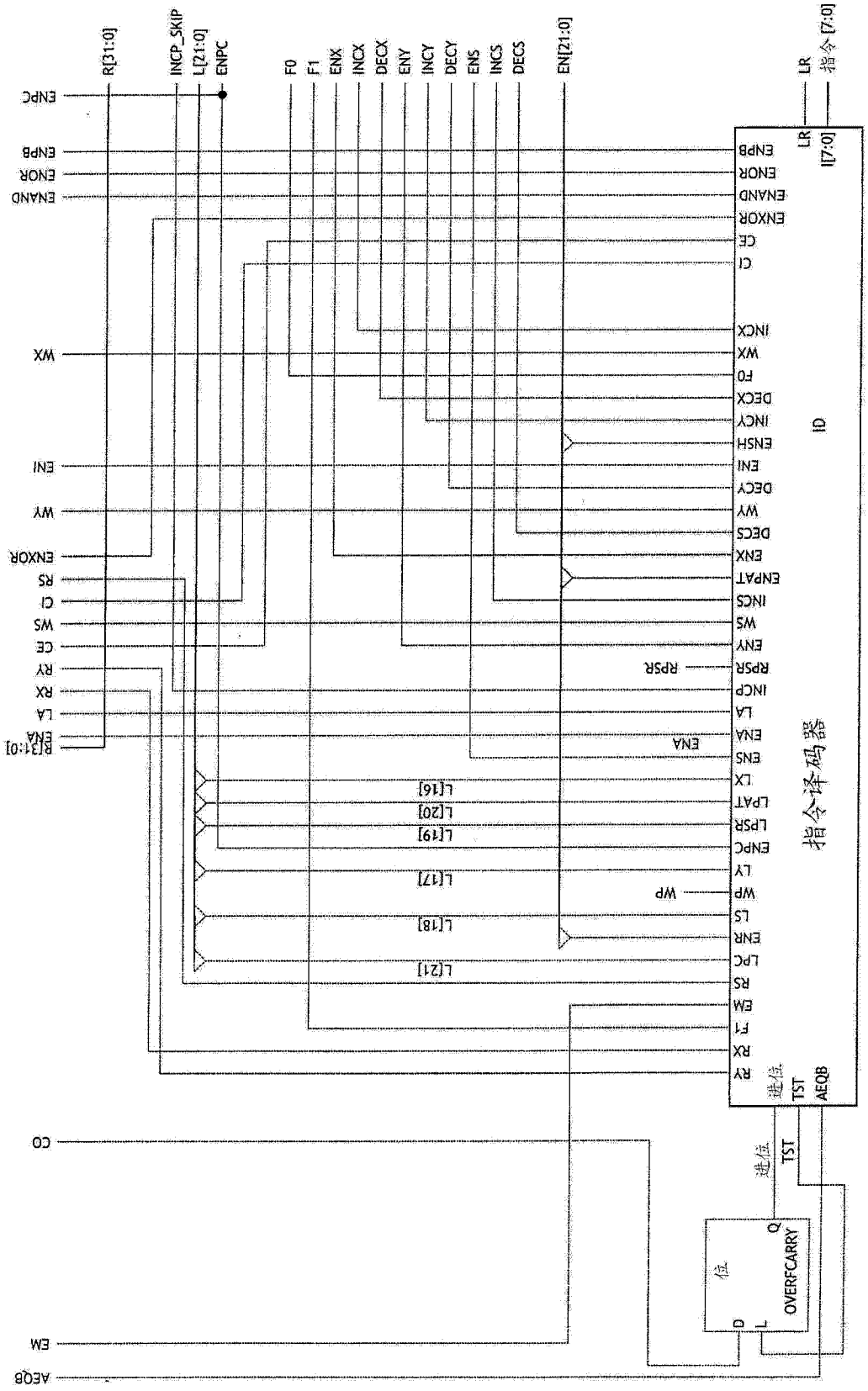

R[31:0]
INCP_SKIP
L[21:0]
ENPC
F0
F1
ENX
INCX
DECX
ENY
INCY
DECY
ENS
INCS
DECS
EN[21:0]
LR
指令[7:0]
ENPC
ENPB
ENOR
ENAND
WX
EN
WY
ENXOR
RS
CI
WS
CE
RY
RX
LA
ENA
R[31:0]
L[16]
L[20]
L[19]
L[17]
L[18]
L[21]
RPSR
WP
ID
指令译码器
进位
TST
AEQB
CO
EM
AEQB
OVERFCARRY
Q
D
L

图 15B

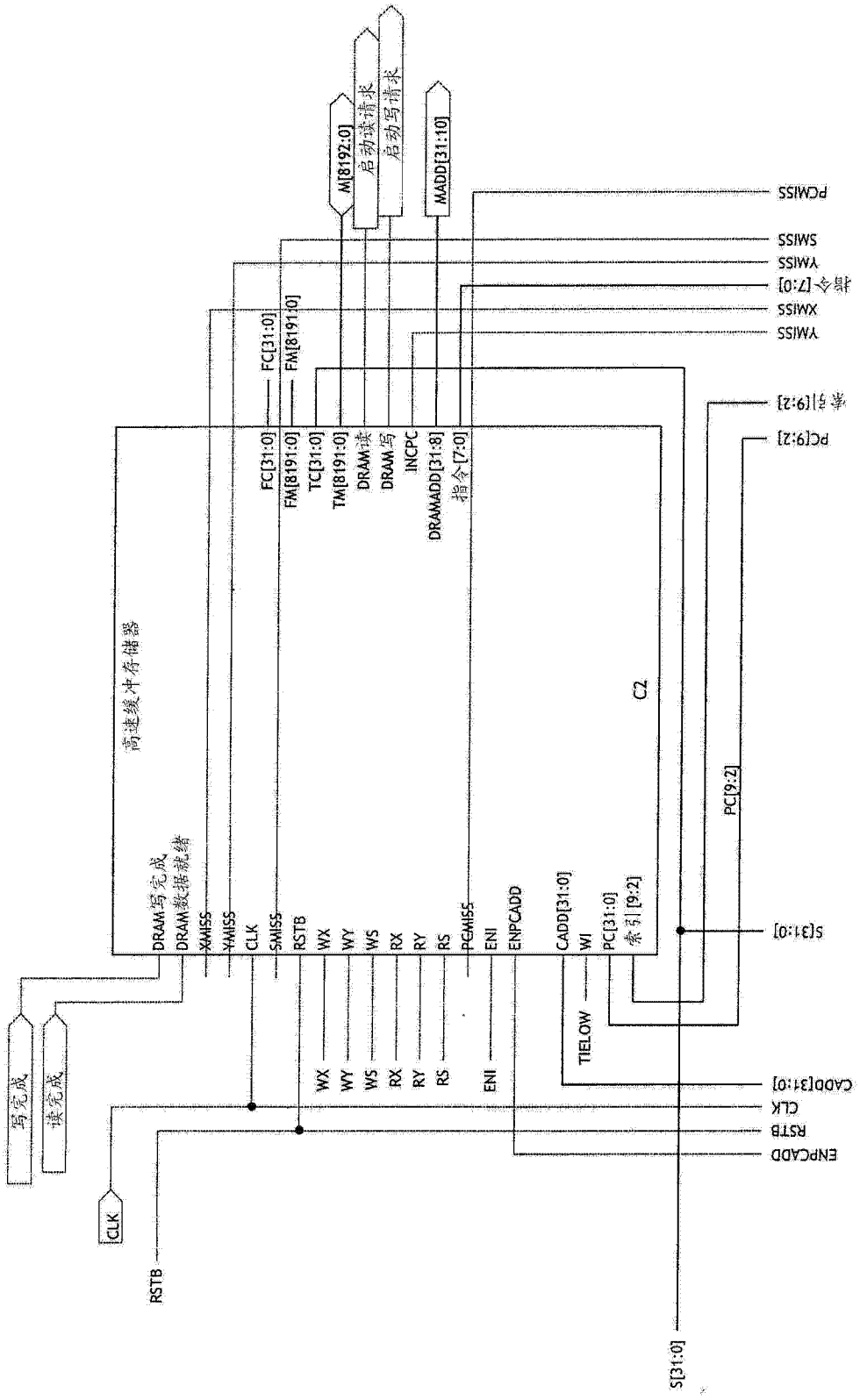

高速缓冲存储器
DRAM写完成
DRAM数据就绪
XMISS
YMISS
CLK
SMISS
RSTB
WX
WY
WS
RX
RY
RS
PCMISS
ENI
ENPCADD
CADD[31:0]
WI
PC[31:0]
索引[9:2]
C2
FC[31:0]
FM[8191:0]
TC[31:0]
TM[8191:0]
DRAM读
DRAM写
INCPC
DRAMADD[31:8]
指令[7:0]
M[8192:0]
启动读请求
启动写请求
MADD[31:10]
TIELOW
PC[9:2]
S[31:0]
写完成
读完成
CLK
RSTB
PCMISS
SMISS
YMISS
指令[7:0]
XMISS
YMISS
索引[9:2]
PC[9:2]
S[31:0]
CADD[31:0]
CLK
RSTB
ENPCADD

图 15C

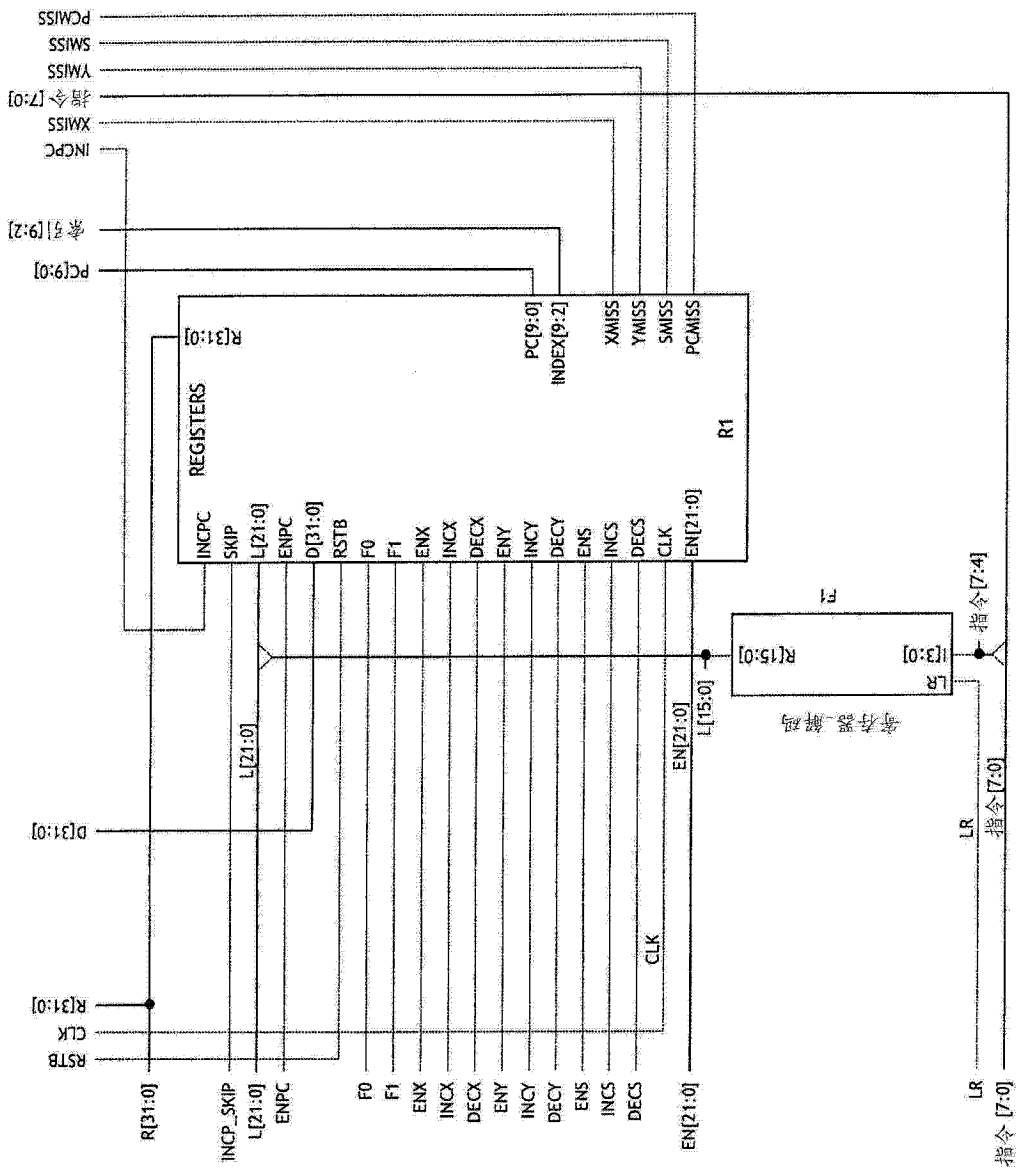

PCMISS
SMISS
YMISS
指令[7:0]
XMISS
INCPC
索引[9:2]
PC[9:0]
R[31:0]
REGISTERS
INCPC
SKIP
L[21:0]
ENPC
D[31:0]
RSTB
F0
F1
ENX
INCX
DECX
ENY
INCY
DECY
ENS
INCS
DECS
CLK
EN[21:0]
PC[9:0]
INDEX[9:2]
XMISS
YMISS
SMISS
PCMISS
R1
F1
R[15:0]
I[3:0]
LR
L[15:0]
寄存器_解码
指令[7:4]
D[31:0]
L[21:0]
EN[21:0]
LR
指令[7:0]
R[31:0]
CLK
RSTB
R[31:0]
INCP_SKIP
L[21:0]
ENPC
F0
F1
ENX
INCX
DECX
ENY
INCY
DECY
ENS
INCS
DECS
CLK
EN[21:0]
LR
指令 [7:0]

图 15D

信号名称	方向	来自/去往	说明
RSTB	输入	全局	复位Bar输入
CLK	输入	全局	系统时钟输入
M[8191:0]	I/O	存储器	宽存储器数据总线（8K比特）
MADD[31:10]	输出	存储器	高阶存储器位为每个地址寻址存储器中的8K组组
启动读请求	输出	存储器	请求从DRAM组读数据
启动写请求	输出	存储器	请求将数据写入DRAM组
写完成	输入	存储器	从存储器控制器反馈操作完成
读完成	输入	存储器	从存储器控制器反馈操作完成
待定：			
WP	输出	仲裁器	写由处理器选择寄存器所指的处理器
RPSR	输出	仲裁器	读处理器选择寄存器
PSR[5:0]	I/O	PSR	处理器选择数据总线
信号名称	来自块	去往块	
ENOR	指令_解码	ALU	在A和B输入之间实现OR功能，在S处得到结果
ENPB	指令_解码	ALU	从B到S实现通过B函数
ENAND	指令_解码	ALU	在A和B输入之间实现AND功能，在S处得到结果
ENXOR	指令_解码	ALU	在A和B输入之间实现XOR功能，在S处得到结果
CI	指令_解码	ALU	输入中的ALU进位用于加法、递增、递减指令
CE	指令_解码	ALU	进位使能用于加法，递增，递减的执行进位加法器单元
CO	ALU	累加器	对于使用进位的操作，从ALU输出溢出结果或进位输出
A[31:0]	累加器或高速缓冲存储器	ALU	用作第一个ALU输入的寄存器和高速缓冲存储器值
B[31:0]	多个位置	ALU	用作第二ALU输入的寄存器和高速缓冲存储器值
S[31:0]	ALU	多个位置	ALU输出总和或结果
EM	指令_解码	可选择的乘法器	启用乘法器输出
进位	OVERFCARRY	指令_解码	指示加果进位在最后测试“TST”指令之后发生
TST	指令_解码	OVERFCARRY	指仅为结果锁存至寄存器用于所后的分支指令
AEQB	比较	指令_解码	指令解码输入指示A与B之间的比较结果
WX	指令_解码	高速缓冲存储器	写X寄存器
WY	指令_解码	高速缓冲存储器	写Y寄存器
WS	指令_解码	高速缓冲存储器	写堆栈
RX	指令_解码	高速缓冲存储器	读X寄存器
RY	指令_解码	高速缓冲存储器	读Y寄存器
ENI	指令_解码	高速缓冲存储器	启用索引
ENA	指令_解码	TRISTATE32	允许累加器同时驱动两个ALU输入。
TIELOW TO WI	未使用	高速缓冲存储器	该信号允许指令高速缓冲存储器写回存储器，没有用。
PC[31:0]	寄存器	高速缓冲存储器	程序计数器。
索引[9:2]	寄存器	高速缓冲存储器	8比特中的32位组的索引值。
XMISS	寄存器	高速缓冲存储器	指示X高速缓冲存储器消掉和产生读或写存储器请求
YMISS	寄存器	高速缓冲存储器	指示Y高速缓冲存储器消掉和产生读或写存储器请求
SMISS	寄存器	高速缓冲存储器	指示堆栈高速缓冲存储器消掉和产生读或写存储器请求
PCMISS	寄存器	高速缓冲存储器	指示程序计数器高速缓冲存储器消掉和产生读或写存储器请求
INCPC	高速缓冲存储器	寄存器	用于成置PC花零存高速缓冲存储器消掉以解决的时候计数
INCP_SKIP	指令_解码	寄存器	用于递增PC防次以跳过指令
L[21:0]	指令_解码	零寄器	为所有零存器加载控制线
ENPC			启用程序计数器输出
F0	寄存器	高速缓冲存储器	强制0000H。可与F1一起使用以强制8000H。
F1	寄存器	高速缓冲存储器	强制0001H。可与F0一起使用以强制8000H。
ENX	寄存器	高速缓冲存储器	启用X寄存器输出
INCX	寄存器	高速缓冲存储器	X递增
DECX	寄存器	高速缓冲存储器	X递减
ENY	寄存器	高速缓冲存储器	启用Y寄存器输出
INCY	寄存器	高速缓冲存储器	Y递增
DECY	寄存器	高速缓冲存储器	Y递减
ENS	寄存器	高速缓冲存储器	启用堆栈指针
INCS	寄存器	高速缓冲存储器	堆栈指针递增
DECS	寄存器	高速缓冲存储器	堆栈指针递减
LR	指令_解码	寄存器_解码	加载程序寄存器
指令[7:0]	高速缓冲存储器	指令_解码	8位TOMI指令

图 15E