[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 11/16

[12] 发明专利申请公开说明书

[21] 申请号 03821646.9

[43] 公开日 2005 年 10 月 12 日

[11] 公开号 CN 1682195A

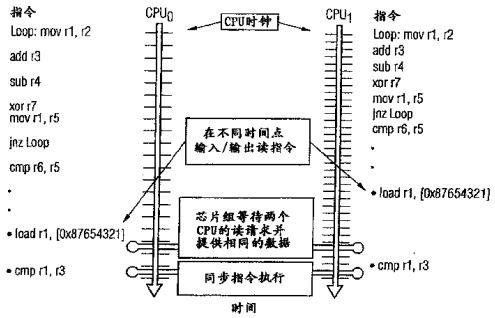
[22] 申请日 2003.8.1 [21] 申请号 03821646.9
[30] 优先权
[32] 2002. 9.12 [33] EP [31] 02020602.5
[32] 2002.12.12 [33] EP [31] 02027847.9
[86] 国际申请 PCT/EP2003/008559 2003.8.1
[87] 国际公布 WO2004/034260 德 2004.4.22
[85] 进入国家阶段日期 2005.3.11
[71] 申请人 西门子公司
地址 德国慕尼黑
[72] 发明人 P·佩勒斯卡 D·施纳贝尔
A·韦伯
[74] 专利代理机构 中国专利代理(香港)有限公司
代理人 程天正 张志醒

权利要求书 5 页 说明书 14 页 附图 3 页

[54] 发明名称 同步或异步定时的处理单元的同步方法和电路装置

[57] 摘要

为冗余系统设立多倍同样构造的、以 Lockstep 工作方式的处理器板。 实现 Lockstep 系统的基本前提是，所有包含在板子中的组件、也即 CPU、芯片组、主存储器等具有决定性的特性。 在此，决定性的特性意味着，当这些组件在相同的时间收到相同的激励时，它们在故障状态下将在相同的时间点提供相同的结果。 另外，决定性的特性以采用时钟同步的接口为前提。 在系统中，异步接口在许多情况下将导致某种时间不准确性，由此不能保持系统的时钟同步的总特性。 但为了能执行 Lockstep 工作，本发明规定了一种不同于已知软件解决方案的、用硬件实现的同步相同或不同的冗余处理单元（PRO_0，PRO_1）的方法，所述的处理单元处理相同的指令序列并同步或异步地被定时，据此，向所述处理单元（PRO_0，PRO_1）外部起作用的事务由被分配给所述处理单元（PRO_0，PRO_1）的模块（EQ_0，EQ_1）用来同步所述的处理单元（PRO_0，PRO_1），其方式是，所述的处理单元分别通过所述分配的模块被延迟，直到所有处理单元的指令执行已获得当前的事务。



ISSN1008-4274

1. 用于同步相同或不同的冗余处理单元（PRO_0, PRO_1）的方法，所述的处理单元处理相同的指令序列并同步或异步地被定时，据此，向所述处理单元（PRO_0, PRO_1）外部起作用的事务由被分配给所述处理单元（PRO_0, PRO_1）的模块（EQ_0, EQ_1）用来同步所述的处理单元（PRO_0, PRO_1），其方式是，所述的处理单元分别通过所述分配的模块被置为等候状态，直到所有处理单元的指令执行已获得当前的事务。

2. 如权利要求 1 所述的方法，其特征在于：通过所述的模块（EQ_0, EQ_1）经连接（L_0, L_1）传输表征所述事务的参数来同步所述的处理单元（PRO_0, PRO_1）。

3. 如权利要求 2 所述的方法，其特征在于：通过以下方式来执行读事务，即由分配给处理单元的模块将处理单元置为等候状态直到需要读取的数据到达，并且把读事务的一个或多个参数发送给与事务目标（I/O_0, I/O_1, MEM_0, MEM_1, REG_0, REG_1, CMEM）最直接相连的模块，其中，由与事务目标最直接相连的模块接收其它模块的一个或多个参数和一个或多个本地产生的参数，并进行比较，在一致的情况下执行所述的读事务和把读取的数据发送给所有的模块，然后由所有的模块把读取的数据传送给所分配的处理单元和促使指令执行的继续。

4. 如权利要求 3 所述的方法，其特征在于：通过以下方式执行数据比较以校验数据的完整性，即有规则地或根据请求从主存储器（MEM0, MEM1）读取数据区并比较其参数，其中通过所选择的或通过所有的模块来进行该比较。

5. 如权利要求 2 所述的方法，其特征在于：通过以下方式来执行写事务，即由分配给处理单元的模块将处理单元置为等候状态直到写过程结束，并且把写事务的一个或多个参数发送给与事务目标（I/O_0, I/O_1, MEM_0, MEM_1, REG_0, REG_1, CMEM）最直接相连的模块，其中，由与事务目标最直接相连的模块接收其它模块的一个或多个参数和一个或多个本地产生的参数，并进行比较，在一致的情况下执行所述的写事务和把完成的写过程回复给所有的模块，然后由所有的模块促使所分配的处理单元继续指令的执行。

6. 如权利要求 2 所述的方法，其特征在于：通过临时存储外部事件来处理外部事件，其中在处理单元的特殊工作模式中调用该存储的外

部事件以通过处理单元的至少各一个执行单元来处理，而且所述的处理单元响应于满足由指令所预给定或固定给定的条件而进入该工作模式，并通过所述的模块（EQ_0, EQ_1）延迟指令执行的继续，直到所有的处理单元已结束该特殊工作模式。

7. 如权利要求 6 所述的方法，其特征在于：如果所述处理单元的比较器单元（K）测得计数单元（CIC）与寄存器单元（MIR）一致，则执行到所述特殊工作模式的切换，其中所述寄存器单元（MIR）的内容可通过指令预给定，并且对于所有的处理单元（PRO_0, PRO_1）都是相同的，所述的计数单元（CIC）包含有从最后切换到所述特殊工作模式以来由执行单元所执行的指令数量。

8. 如权利要求 6 或 7 所述的方法，其特征在于：由输送给所述处理单元的外部事件触发一个随事件矢量的读事务开始的事件处理例程，通过以下方式来执行读事务，即由分配给处理单元的模块将处理单元置为等候状态直到需要读取的数据到达，并且把读事务的一个或多个参数发送给与事务目标（I/O_0, I/O_1, MEM_0, MEM_1, REG_0, REG_1, CMEM）最直接相连的模块，其中，由与事务目标最直接相连的模块接收其它模块的一个或多个参数和一个或多个本地产生的参数，并进行比较，在一致的情况下执行所述的读事务和把读取的数据发送给所有的模块，然后由所有的模块把读取的数据传送给所分配的处理单元和促使指令执行的继续。

9. 如权利要求 1 或 2 所述的方法，其特征在于：通过以下方式进行直接的存储器访问以便把数据从存储器传输到输入/输出模块（I/O_0, I/O_1），即通过利用寄存器中的录入项把由处理单元产生的指令传输给输入/输出模块来启动所述直接的存储器访问。

10. 如权利要求 1 或 2 所述的方法，其特征在于：通过以下方式进行直接的存储器访问以便把数据从输入/输出模块（I/O_0, I/O_1）传输到存储器中：

－在第一步骤中把一个由输入/输出模块产生的描述符存放到存储器中，并由处理单元用轮询方法读出，

－在第二步骤中通过处理单元读取所述模块（EQ_0, EQ_1）之一内的寄存器，以促使不再允许输入/输出模块在存储器内进行写事务，

－在第三步骤中通过所述的模块（EQ_0, EQ_1）把最后由输入/输出模

块发送的写事务写入到所有处理单元的存储器中，

－在第四步骤中由所有处理单元读取存储器内的一个存储位置，该存储位置的值指示了直接存储器访问的结束，以及

－在第五步骤中重新读取所述的寄存器，或者读或写另一个寄存器，以便通过 I/O 单元再次对主存储器实现写访问。

11. 如权利要求 1 或 2 所述的方法，其特征在于：通过以下方式进行直接的存储器访问以便在输入/输出模块（I/O_0, I/O_1）和存储器之间进行数据传输：

－在第一步骤中通过处理单元读取所述模块（EQ_0, EQ_1）之一内的寄存器，以促使不再允许输入/输出模块在存储器内进行读事务，

－在第二步骤中把一个由处理单元产生的描述符存放到存储器中，并且该描述符可由一个或多个输入/输出模块（I/O_0, I/O_1）用轮询方法读出，

－在第三步骤中重新读取所述的寄存器，或者读或写另一个寄存器，以便通过 I/O 单元再次对主存储器实现读访问，以及

－在第四步骤中由一个或多个输入/输出模块（I/O_0, I/O_1）读取存储器内的一个存储位置，该存储位置的值指示了直接存储器访问的开始。

12. 如权利要求 3－7 之一所述的方法，其特征在于：当确定其它模块的参数以及一个或多个本地产生的参数有偏差时，由与事务目标最直接相连的模块引入故障处理。

13. 如权利要求 12 所述的方法，其特征在于：由所述的故障处理停止所述需执行的事务，并启动一个例程来检测故障单元、隔离该故障单元和/或重建同步。

14. 如权利要求 12 所述的方法，其特征在于：在存在 N 个处理单元的情况下，故障处理采取从 N 取 N-M(M<N) 的多数判定，并去活有偏差的处理单元。

15. 如权利要求 2－14 之一所述的方法，其特征在于：通过以下方式对各个处理单元进行故障识别，即对于从所述的一个或多个参数在处理单元的模块中最早地变得可用时开始的任一事务，放弃没有到达的或在预定的时间过去之后才到达的参数，其中利用没有到达的或在预定的时间过去之后才到达的参数来为处理单元启动故障处理。

16. 如权利要求 1－15 之一所述的方法，其特征在于：由所述的模块（EQ_0, EQ_1）采用以下的事务来同步所述的处理单元（PRO_0, PRO_1）：

－与被分配给相应处理单元（PRO_0, PRO_1）的本地存储器（MEM_0, MEM_1）有关的不可临时存储的存储器事务，和/或

－向输入/输出模块（I/O_0, I/O_1）的输入/输出事务，和/或

－向外部寄存器（REG_0, REG_1）的存储器映射的输入/输出事务，和/或

－与处理单元（PRO_0, PRO_1）的共用存储器（CMEM）有关的不可临时存储的存储器事务。

17. 如权利要求 1－16 之一所述的方法，其特征在于：由所述的模块（EQ_0, EQ_1）通过连接（L_0, L_1）传输以下事务参数以便同步处理单元（PRO_0, PRO_1）：

－输入/输出地址，和/或

－存储器地址，和/或

－需转移的数据，和/或

－事务的类型，和/或

－根据输入/输出地址和/或存储器地址和/或需转移的数据和/或事务的类型求出的标记。

18. 用于同步冗余数据处理系统的被同步或异步地定时的处理单元（PRO_0, PRO_1）的装置，具有以下部分：

－至少两个处理单元（PRO_0, PRO_1），用于处理相同的指令序列，

－分别专门地分配给所述处理单元的外围设备（MEM_0, MEM_1），用于存储和/或交换数据，

－能由所有处理单元共用的外围设备（I/O_0, I/O_1, REG_0, REG_1, CMEM），用于存储和/或交换数据，

－被分配给所述处理单元的模块（EQ_0, EQ_1），其中所述的模块（EQ_0, EQ_1）具有用于监视事务的工具、用于触发所分配的处理单元的等候状态直到所有的处理单元获得当前的事务的工具、以及用于将事务参数传输给其它模块的工具（L_0, L_1）。

19. 如权利要求 18 所述的装置，其特征在于：每个处理单元具有以下部分：

－至少一个执行单元（EU），

－至少一个计数器单元（CIC），用于计数从最后切换到特殊工作模式以来所述执行单元所执行的指令，

－至少一个寄存器单元（MIR），其内容可通过指令预给定或固定地给定，

－至少一个比较器单元（K），用于响应于所述计数器单元（CIC）与寄存器单元（MIR）的一致性而把执行单元（EU）切换到特殊工作模式，其中在所述的特殊工作模式下被临时存储的、且需要被输送给处理器模块和影响处理器模块的外部事件通过所述的处理器模块进行调用。

20. 如权利要求 18 或 19 所述的装置，其特征在于：所述的模块（EQ_0, EQ_1）具有借助于以下事务同步处理单元的工具：

－与被分配给相应处理单元（PRO_0, PRO_1）的本地存储器（MEM_0, MEM_1）有关的不可临时存储的存储器事务，和/或

－向输入/输出模块（I/O_0, I/O_1）的输入/输出事务，和/或

－向外部寄存器（REG_0, REG_1）的存储器映射的输入/输出事务，和/或

－与处理单元（PRO_0, PRO_1）的共用存储器（CMEM）有关的不可临时存储的存储器事务。

21. 如权利要求 18－20 之一所述的装置，其特征在于：所述的模块具有求出代表事务的以下参数的工具：

－输入/输出地址，和/或

－存储器地址，和/或

－需转移的数据，和/或

－事务的类型，和/或

－根据输入/输出地址和/或存储器地址和/或需转移的数据和/或事务的类型求出的标记。

同步或异步定时的处理单元的同步方法和电路装置

在电信系统、数据中心和其它可用性高的系统中，在许多情形下会采用多达几百个所谓的处理器板来安排所需要的计算功率。这种处理器板典型地由处理器或 CPU(中央处理单元)、芯片组、主存储器和外围模块组成。

一个常规处理器板每年出现硬件故障的概率处于一位数的百分比范围内。由于有大量的处理器板被综合到一个系统中，所以任意一个硬件组件都会产生与年周期有关的高故障概率，其中如果不采取合适的预防措施，这种单个的故障可能引起整个系统的故障。

尤其在电信系统处，越来越多地也在数据中心处，要求有高度的系统可用性。这譬如用百分比来表达，或规定每年最多允许的故障时间。典型的要求例如是可用性大于 99.999%，或不可用性最多为每年几分钟。由于在硬件故障时更换处理器板或重建业务需要几十分钟到几小时的时间，因此必须针对系统面上的硬件故障采取相应的预防措施，以便能满足所述的系统可用性要求。

为遵守这种高系统可用性的要求，已知的解决方案是设立冗余的系统组件。已知的方法可以分为两大类：基于软件的方法和基于硬件的方法。

在基于软件的方法中典型地采用一种媒件。但这种基于软件的解决方案被证明是欠缺灵活性的，因为在这种系统中只能采用已经为该特殊的冗余方案所开发的（应用）软件。这大大地限制了可使用的（应用）软件的范围。此外，为软件冗余原理开发应用软件实际上是极其费事的，其中这种开发另外会带来复杂的测试方法。

基于硬件的方法的基本原理在于，把冗余包封到硬件平面上，使得这对于软件而言是透明的。由硬件自身管理的冗余的主要优点是，应用软件不会受到冗余原理的损害，因此在大多数情况下能够采用任何一个软件。

对于能容忍硬件故障的系统－其冗余对软件是透明的－来说，在实践中经常采取的原理是所谓的 Lockstep（步锁）原理。Lockstep 的意思是，相同结构的硬件、例如两个板，是同样地时钟同步地运行的。通

过硬件机制来确保冗余的硬件在给定的时间点上经受相同的输入激励，并由此必定产生相同的结果。比较冗余组件的结果，在有偏差的情况下确定存在故障并引入合适的措施（向操作人员告警，部分或全部地进行安全性关断，系统重启）。

实现 Loctstep 系统的基本前提条件是，所有包含在板子中的组件、也即 CPU、芯片组、主存储器等具有时钟决定性的特性。在此，时钟决定性的特性意味着，当这些组件在相同的时间点收到相同的激励时，它们在无故障状态下将在相同的时间点提供相同的结果。另外，时钟决定性的特性以采用时钟同步的接口为前提。在系统中，异步接口在许多情况下将导致某种时间不准确性，由此不能保持系统的时钟同步的总特性。

但对芯片组和 CPU 而言，异步接口恰好在提高功能性方面提供了技术上的优点，由此不可能按照 Lockstep 方法实现时钟同步的工作方式。另外，现代的 CPU 越来越多地采用不能实现时钟同步工作方式的机制。这例如是内部的、对外为不可见的校正措施，例如可能导致指令处理产生稍微延迟的、在访问高速缓存器时对内部可纠正错误的校正，或者指令的推测性执行。另一例子是将来越来越多地实现 CPU 内部的、无时钟的执行单元，这些执行单元在速度和损耗功率方面能取得巨大的优点，但阻止了 CPU 的时钟同步的或决定性的工作。

US 专利文献 5226152 公开过一种用于冗余处理器的功能性的 Lockstep 装置，其中所有的处理器被连接到一个逻辑装置上，由该逻辑装置同步处理器对外围设备的访问，并实现冗余处理器的功能性 Lockstep 工作。

然而，就文章开头所说的处理器板而言，利用这种具有中央逻辑装置的装置会产生巨大的缺点，即：除了所述的处理器板外，还需要分别为某个数量的处理器板设立一个逻辑板，然后由该逻辑板控制外围设备访问的同步。但这种逻辑板那里必须得到监视，因而可能导致复杂的监视机制。

换句话说，虽然 US5226152 的装置看起来适合于为具有多个处理器的单板系统设立功能性 Lockstep，但这种装置并不适合于本文开头所说的那种具有多个相同处理器板的系统。

因此本发明的任务在于提供一种方法，用于保障 Lockstep 方法的

优点和考虑技术的发展。

该任务通过如权利要求 1 的特征所述的、同步或异步定时的处理单元的同步方法来解决，以及通过如权利要求 13 的特征所述的、同步或异步定时的处理单元的同步电路装置来解决。

优选的实施方案由从属权利要求给出。

根据本发明规定了一种用于同步相同或不同的冗余处理单元 PRO_0, PRO_1 的方法，所述的处理单元处理相同的指令序列并同步或异步地被定时，据此，向所述处理单元 PRO_0, PRO_1 外部起作用的事务被分配给所述处理单元 PRO_0, PRO_1 的模块 EQ_0, EQ_1 用来同步所述的处理单元 PRO_0, PRO_1，其方式是，所述的处理单元分别通过所述分配的模块被延迟和由此平衡，直到所有处理单元的指令执行已获得当前的事务。

在此可以采用以下的事务来进行同步：

- 与被分配给相应处理单元 PRO_0, PRO_1 的本地存储器 MEM_0, MEM_1 有关的不可临时存储的存储器事务，和/或

- 向输入/输出模块 I/O_0, I/O_1 的输入/输出事务，和/或

- 向外部寄存器 REG_0, REG_1 的存储器映射的输入/输出事务，和/或

- 与处理单元 PRO_0, PRO_1 的共用存储器 CMEM 有关的不可临时存储的存储器事务。

在此可以通过以下方式来执行读事务，即由分配给处理单元的模块将处理单元置为等候状态直到到达需要读取的数据，并且把读事务的一个或多个参数发送给与事务目标 I/O_0, I/O_1, MEM_0, MEM_1, REG_0, REG_1, CMEM 最直接相连的模块，其中，由与事务目标最直接相连的模块接收其它模块的一个或多个参数和一个或多个本地产生的参数，并进行比较，在一致的情况下执行所述的读事务和把读取的数据发送给所有的模块，然后由所有的模块把读取的数据传送给所分配的处理单元和允许指令执行的继续。

在此可以通过以下方式来执行写事务，即由分配给处理单元的模块将处理单元置为等候状态直到写过程结束，并且把写事务的参数发送给与事务目标 I/O_0, I/O_1, MEM_0, MEM_1, REG_0, REG_1, CMEM 最直接相连的模块，其中，由与事务目标最直接相连的模块接收其它模块的一个或多个参数和一个或多个本地产生的参数，并进行比较，在一致的情况下执行所述的写事务和把产生的写过程回复给所有的模块，然后由所有的模块

允许所分配的处理单元继续指令的执行。

优选地，如果外部事件的处理通过从一个存储位置或一个寄存器读取一个值、例如中断矢量而被引入，并同时确保所有处理单元在指令执行的相同位置处获得外部事件，则可以结合本发明的同步方法根据事务来处理外部事件、例如中断。引入事件处理的读事务如上文所述那样来进行，譬如借助于一个中断确认周期。

同步外部事件的合适方法在欧洲专利申请 02020602 中讲述过，并规定临时存储外部事件，其中在处理单元的特殊工作模式中调用该存储的外部事件以通过处理单元的至少各一个执行单元来处理，而且所述的处理单元响应于满足由指令所预给定或固定给定的条件而进入该工作模式，并通过所述的模块 EQ_0, EQ_1 延迟指令执行的继续，直到所有的处理单元已结束该特殊工作模式。

如果所述处理单元的比较器单元 K 测得计数单元 CIC 与寄存器单元 MIR 一致，则例如执行到所述特殊工作模式的切换，其中所述寄存器单元 MIR 的内容可通过指令预给定，并且对于所有的处理单元 PRO_0, PRO_1 都是相同的，所述的计数单元 CIC 包含有从最后切换到所述特殊工作模式以来由执行单元所执行的指令数量。

当与事务目标最直接相连的模块确定其它模块的参数以及一个或多个本地产生的参数有偏差时，则可以引入故障处理。在此，故障处理可以停止所述需执行的事务，并启动一个例程用于诊断、隔离故障和/或必要时重建同步。在存在 N（例如 N=3）个处理单元的情况下，可以采取从 N 取 N-1 的多数判定，或一般采取从 N 取 N-M 的多数判定，并去活有偏差的处理单元。

另外可以通过以下方式对各个处理单元进行故障识别，即对于从所述的一个或多个参数在处理单元的模块中最早地变得可用时开始的任一事务，放弃没有到达的或在预定的时间过去之后才到达的参数，其中利用没有到达的或在预定的时间过去之后才到达的参数来为处理单元启动故障处理。

本发明另外还规定了一种用于同步被同步或异步地定时的处理单元 PRO_0, PRO_1 的装置，具有以下部分：

- 至少两个处理单元 PRO_0, PRO_1，用于处理相同的指令序列，
- 分别专门地分配给所述处理单元的外围设备 MEM_0, MEM_1，用于存

储和/或交换数据，

－能由所有处理单元共用的外围设备

I/O_0, I/O_1, REG_0, REG_1, CMEM，用于存储和/或交换数据，

－被分配给所述处理单元的模块 EQ_0, EQ_1，其中所述的模块 EQ_0, EQ_1 具有用于监视事务的工具、用于停止所分配的处理单元直到所有的处理单元获得当前的事务的工具、以及用于将事务参数传输给其它模块的工具 L_0, L_1。

在此，所述的模块 EQ_0, EQ_1 具有尤其借助于以下事务同步处理单元的工具：

－与被分配给相应处理单元 PRO_0, PRO_1 的本地存储器 MEM_0, MEM_1 有关的不可临时存储的存储器事务，和/或

－向输入/输出模块 I/O_0, I/O_1 的输入/输出事务，和/或

－向外部寄存器 REG_0, REG_1 的存储器映射的输入/输出事务，和/或

－与处理单元 PRO_0, PRO_1 的共用存储器 CMEM 有关的不可临时存储的存储器事务。

在此，所述的模块优选地具有求出代表事务的以下参数的工具：

－输入/输出地址，和/或

－存储器地址，和/或

－需转移的数据，和/或

－事务的类型，和/或

－根据输入/输出地址和/或存储器地址和/或需转移的数据和/或事务的类型求出的标记。

为处理诸如中断等外部事件，处理单元优选地具有以下部分：

－至少一个执行单元 EU，

－至少一个计数器单元 CIC，用于计数从最后切换到特殊工作模式以来所述执行单元所执行的指令，

－至少一个寄存器单元 MIR，其内容可通过指令预给定或固定地给定，

－至少一个比较器单元 K，用于响应于所述计数器单元 CIC 与寄存器单元 MIR 的一致性而把执行单元 EU 切换到特殊工作模式，其中在所述的特殊工作模式下被临时存储的、且需被输送给处理器模块和影响处理器模块的外部事件通过所述的处理器模块进行调用。

在此，调用临时存储的外部事件可以优选地借助于软件、固件、微代码或硬件来实现。

本发明的主要优点在于：在容忍硬件故障的平台上能够采用任意新的或已有的软件，其中在该平台中可以采用支持本发明的处理单元，而对 CPU 的时钟同步的决定性工作方式没有要求。

其它优点是：

－例如由 CPU、北桥和本地存储器构成的相互成冗余的处理单元不必恒相地耦合而工作。

－CPU 不必相同，这尤其允许在一个冗余系统中同时采用不同的 CPU 级，并且以不同的时钟频率工作。

－CPU 在猜测性地执行指令方面可以有不同的特性。

－例如由于在数据讹误地出现α粒子后进行的校正，相同 CPU 的不同 CPU 内部执行时间仅导致在稍微不同的时间点上获得同步事件。

由于将来 CPU 的时间不准确性，在确保时钟同步的决定性工作方式时的上述问题只导致在时间上不正确相关的指令执行。由于 CPU 在常规的应用中必须对外部事件作出反应，譬如对外围设备所产生的中断或对由设备已写入到主存储器中的数据作出反应，所以必须确保 CPU 在指令执行的相同位置获知这些事件，因为否则的话，对这些事件的分析可能导致冗余 CPU 的不同程序运行。

本发明负责在指令执行的相同位置给冗余 CPU 提供外部的、对程序运行来说重要的事件，例如中断或由外部设备产生的数据，由此能够仿真所述的 Lockstep 工作方式。

另外还比较冗余 CPU 的在指令执行的相同位置被提供的输出事件，并由此认可这些事件。与通过软件方法根据处理器外围设备实现同步和数据分配的已知方法相反，本发明是通过硬件来实现的。在此有一个决定性的优点：性能影响要比软件方法低很多倍。所述的方法另外对于应用软件和操作系统软件是完全透明的，也即已有的应用软件和操作系统软件可以不用修改就继续被使用。

下面结合三个附图来详细讲述本发明的实施例。

图 1 简要地示出了具有所属外围设备的两个处理单元和事务的同步。

图 2 简要地示出了借助于利用两个模块的外围设备事务被同步的

两个处理单元。

图 3 简要地示出了具有进一步细节的优选处理单元的结构。

图 4 示出了两个不同地被定时的处理单元的指令处理时间图及其按照本发明的同步。

在图 1 中简要地示出了两个处理单元 PRO_0、PRO_1，它们的在外部起作用的事务被同步。示例地示出了针对以下组件的事务：本地存储器 MEM_0、MEM_1，寄存器 REG_0、REG_1，以及输入/输出模块或 I/O 模块 I/O_0、I/O_1。在此，第一处理单元 PRO_0 被分配了第一组件 MEM_0、REG_0 和 I/O_0，而第二处理单元 PRO_1 被分配了第二组件 MEM_1、REG_1 和 I/O_1。如相应的虚线连接所示，处理单元能访问其它各个处理单元的寄存器 REG_n 和 I/O 模块 I/O_n，而只有被分配的处理单元 PRO_k 才能访问本地存储器 MEM_k。

另外还示例地示出了处理单元共同访问的组件，此处为共用的存储器 CMEM，其中，与寄存器和 I/O 模块不同的是，该共用存储器没有被分配给处理单元中的任何一个。

图 2 再次示出了两个处理单元，并示例性地示出了图 1 的 I/O 模块以及寄存器。它们不是常规地直接通过相应的接口或接口模块相连接的，而是借助了均衡器模块 EQ_0、EQ_1。处理单元 PRO_0 的所有访问通过均衡器 EQ_0 接收、处理和相应地继续传送，同样，处理单元 PRO_0 也通过均衡器 EQ_0 提供所有的外部数据和事件。类似地，处理单元 PRO_1 被分配了一个相等的均衡器 EQ_1。

均衡器 EQ_0、EQ_1 交换信息，并且为此优选地具有快速和直接的连接 L_0、L_1。如图所示，该连接可以被逻辑地和/或物理地划分为第一连接 L_0: EQ_0->EQ_1 和第二连接 L_1: EQ_1->EQ_0。

如图 2 的虚线所示，根据本发明也可以连接其它的单元，包括各一个处理单元 PRO、一个均衡器 EQ 和外围设备 REG、I/O，以便构成相应多倍冗余的系统。通过加入其它的这种单元，将产生 3 倍冗余的系统，其中已经能够通过“从 3 取 2”多重判定来进行错误处理。

图 3 最后示出了本发明结合常规处理器/外围设备结构的详细实现，其特征在于，主处理器 CPU 通过北桥（Northbridge）接口单元 NB 与南桥（Southbridge）接口单元 SB 相耦合，其中所述的北桥譬如还包括与本地存储器 MEM_0 的接口，而所述的南桥譬如包括中断控制器和其它的 I/O 功能。

如图 3 示例地所示，处理单元 PRO_0 可以由 CPU、北桥和本地存储器构成。在特别优选的方案中，除了常规的单元（为简便起见只示出了它们当中的一个高速缓存器和一个执行单元 EU）外，CPU 还可以包含一个寄存器 MIR、一个计数器 CIC 和一个比较器 K，它们被用来只在指令执行当中的某些位置处把诸如中断和异常等外部事件传送给执行单元，并保证一个否则不间断的指令序列处理过程。

本发明的均衡器模块 EQ_0 优选地被布置在北桥和南桥之间，因为北桥和南桥之间的接口具有所有必要的信号线路以使均衡器能够停止指令序列的处理，直到处理单元 PRO_0 与相邻的处理单元（未示出）达到同步。只示出了连接 L_0、L_1 用于连接均衡器 EQ_0 和相邻处理单元的均衡器。

图 3 所示的逻辑分组并没有必要对应于各个组件的实际物理分组。例如北桥可以被集成到 CPU 中，或者均衡器可以被集成到北桥或南桥中，或与北桥一起被集成到 CPU 中。

图 4 用时间流程图示出了两个处理单元的指令执行的同步。在图 4 所示的实施例中，相同的指令序列通过两个 CPU、即 CPU_0 和 CPU_1 来处理，其中 CPU_0 利用比 CPU_1 更小的时钟速率工作。为了使 CPU_1 在比 CPU_0 更早的时间点上得到每个指令，前提是在开始时、也即在处理 mov r1, r2 时所有的寄存器和分配给 CPU 的存储器都已被同步。

只要 CPU 没有例如借助于 I/O 模块与外界进行交互作用或访问共用存储器，则非同步的指令处理是可以容忍的。然而对于这种事务，例如在图 4 的实施例中读出 I/O 寄存器 0x87654321，这种事务对于两个 CPU 而言必须是同时进行的，尤其是利用同样的结果同步地进行的。这借助于下文所述的均衡器来实现。均衡器在该事务点上同时负责重建 CPU 的同步。

按照 Lockstep 工作方式，本发明的方法在下面被称为仿真 Lockstep。仿真 Lockstep 的实现包括至少两个处理单元 PRO_0 和 PRO_1，其可以由 CPU、存储器以及存储控制器（标准芯片组的北桥）构成。这些处理单元构造相同，但可以具有不同的 CPU 或一个 CPU 的不同级，并且在相同的状态下、也即在相同的存储器-和 CPU-寄存器内容下被启动。根据本发明不需要通过共同或同步的时钟进行耦合。

在机器指令执行的范围内，由 CPU 初始化存储器周期，例如写周

期、读周期和必要时的 I/O 周期。为了必要时利用 CPU 之间的数据交换来同步 CPU，满足以下条件的所有周期都是合适的：

(a)它们是指令决定性的，也即它们被相同地由所有的 CPU 在同样的程序位置和以同样的顺序发出，以及

(b)它们由 CPU 总是向外部发出，也即它们总是可以在处理器外部被看见和提取；处理器内部的高速缓存器周期例如是不合适的。

下面的存储器周期例如满足该边缘条件：

－在自身的存储器 MEM_0、MEM_1 中不能临时存储的或不能高速缓存的存储器周期，

－I/O 周期，

－存储器映射到例如外部寄存器 REG_0、REG_1 上的 I/O 周期，

－不能临时存储和不能高速缓存到外部共用存储器 CMEM 的存储器周期。

不同的外部寄存器，例如定时器，计数器和/或中断逻辑，以及对外界的 I/O 单元、例如以太网控制器或 SCSI 控制器，通常与 CPU 存在通信。在 CPU 和 I/O 单元之间，通过异步或同步的接口为每个 CPU 接入各一个均衡器，由其实现仿真 Lockstep 工作。在均衡器 EQ_0 和 EQ_1 之间需要异步或同步的点对点连接 L_0、L_1，以便能交换数据、地址或标记。在传输故障的情况下，在异步的接口上可以安排重复地传输。

对 I/O 单元或寄存器的读或写访问是作为存储器映射的 I/O 或直接的 I/O 来实现的。I/O 单元全部是可见的，并可以通过分开的存储器地址达到。相反，所述的寄存器在主－主配置或主－从配置中可以被切换。在主－主配置中，被分别分配的处理单元的寄存器进行读或写动作。这种工作方式的前提是，寄存器在处理单元的访问时具有相同的状态，以便保证这些单元的并行工作方式。

在主－从配置中，由所有的单元专门地读取主单元的寄存器，而且主单元的寄存器只被主单元写。例如，为读取所有单元的当前时间，使用主单元的日时间（ToD）计数器来确保在读取 ToD 计数器时给所有的单元提供精确相同的时间，也即只有被分配给处理单元的寄存器动作。在其它单元上发生的诸如中断等事件则必须传输给主单元。对该寄存器的写访问必须在所有单元上进行，或者被寄存到屏蔽寄存器的主存储器内，以便在故障情况下能用新的主单元以正确的数据继续工作。这可以

借助于软件或硬件来控制。

下面详细讲述各个事务和借助于该事务进行的同步过程。

读事务

处理单元 PRO 的 CPU 的读指令从 I/O 单元读取数据。这种读指令在图 4 中被示出，其例如是指令 load r1, [0x87654321]。该指令由所有的 CPU 在指令执行的相同位置产生，并指向某个 I/O 单元（例如 I/O_0）或主寄存器。但读指令的时间点在多个 CPU 中可以是不同的。在图 4 中 CPU_0 比 CPU_1 晚得到所述的读指令。

由 CPU 产生的 I/O 地址或存储器地址以及事务的属性（例如存储器读或 I/O 读或数据长度）、或者从地址和属性产生的标记由直接连接在 CPU 上的均衡器发送给所有其它的均衡器。只有当连接在被寻址的 I/O 源上的均衡器识别出已经由所有的 CPU 产生了读请求时，才执行本来的读访问。在主－从配置的情况下，被读取的数据被分配给所有的均衡器，然后由这些均衡器通过把数据传递给 CPU 来终接各个被连接的 CPU 的读指令。数据可以在不同的时间点进入 CPU，但并不会因此影响进一步的程序执行。

如果在一个均衡器中的 I/O 地址或标记有偏差，则要么不执行读访问并产生一个故障中断，例如向 CPU 产生一个不可遮蔽的中断 NMI，要么在配置有 3 个 CPU 的情况下采取多数判定，例如“从 3 取 2”。故障单元被隔离开和被诊断。

为了识别各个单元的故障，在时间上监视读访问，也就是说所有 CPU 的读指令都必须按照某个预定的时间产生。如果超过各指令之间的这种时间间隔，则产生超时，并分离和诊断出故障的单元。

读访问按照其出现的顺序被处理。不设立检修。

写事务

写指令把数据写入 I/O 单元或存储单元。该指令由所有 CPU 在指令执行的相同位置产生，并例如指向某个 I/O 单元，例如 I/O_0。但写指令的时间点在多个 CPU 中可以是不同的。

例如由 CPU 产生的 I/O 地址、数据和属性或由此算出的标记从直接相连的均衡器被发送给所有其它均衡器。只有当由所有 CPU 已产生了写请求并已被均衡器认可时，才执行本来的写访问。

如果在一个均衡器中的 I/O 地址、数据和/或属性或标记有偏差，

则要么不执行写访问并产生故障中断，例如向 CPU 产生一个不可遮蔽的中断 NMI，要么在配置有 3 个 CPU 的情况下采取例如“从 3 取 2”的多数判定。分离并诊断故障单元。

为了识别各个单元的故障，在时间上监视写访问，也就是说所有 CPU 的写指令都必须按照某个预定的时间产生。如果超过各指令之间的这种时间间隔，则产生超时，并分离和诊断出故障的单元。

写访问按照其出现的顺序被处理。不设立检修。但可以由 CPU 产生多个写周期（所谓的 Posted Writes）。为了处理这种多重写事务，可以设立相应规格的先进先出存储器（未示出）。

中断

影响程序过程的外部事件不是直接输入给 CPU，而是由合适构造的硬件首先缓存。在此，该硬件可以是 CPU 外的模块的组成部分，或者是 CPU 自身的组成部分。CPU 包含一个计数器 CIC(完成的指令计数器)，用于对 CPU 已完整执行的指令或机器指令进行计数。CPU 另外还包含一个寄存器 MIR(最大指令寄存器)，该寄存器由一个支持仿真 Lockstep 工作的软件（ELSO）来写。

另外，CPU 还具有一个比较器 K，其将被执行过的指令数量、也即计数器 CIC 与寄存器 MIR 进行比较，并在相等的情况下例如产生一个中断请求，由其根据寄存器 MIR 所预定的指令数量来中断指令执行，并将 CPU 切换到另一工作模式。在该工作模式下，例如执行合适的微代码，或者转移到一个中断服务例程，或者按每个硬件信号指示该同步点的到达。于是在该工作模式下，为冗余的 CPU 如此地提供外部事件，使得在离开该工作模式之后所有的 CPU 都能同样地分析该事件，并由此按照顺序执行同样的指令。

例如，在达到由寄存器 MIR 所给定的机器指令数量之后，CPU 转移到一个中断服务例程，在该例程中询问通过所述 CPU 的硬件保持的中断信号的状态，使得有时在稍晚的时间点上提出该询问的冗余 CPU 收到相同的消息。该询问例如是对中断寄存器的读访问。该读访问象上文那样被处理，由此确保所有 CPU 读取相同的中断矢量，并引入相同的动作。

在离开该特殊的工作模式之前将计数器 CIC 复位。接着跳回到因达到寄存器 MIR 所给定的计数值 CIC 而发生中断的程序位置。此后 CPU 再执行由寄存器 MIR 所给定数量的机器指令，并在计数器 CIC 达到寄

存器值 MIR 时切换模式，由此能接受外部事件。

例如，一个支持仿真 Lockstep 工作的软件 ELSO 能把寄存器 MIR 置为值 10.000。于是，以 5GHz 时钟频率工作并平均每个时钟执行一个机器指令(时钟长度:1/200ps)的 CPU 将在指令执行 2μs 之后被中断，并实现与外部事件的同步。

直接的存储器访问 DMA

在 DMA 事务（直接存储器访问）中，一个 I/O 单元能够直接对主存储器进行读和写访问。I/O 单元和 CPU 的访问的时间关系没有被给出。如果 CPU 在 DMA 转移期间访问相同的存储区，则处理单元可能丧失其准同步的工作方式，因为处理单元的主存储器在访问时间点上没有必要再相同。

因此对于 DMA 事务必须保证，将一个在所有的 CPU 上在指令执行的相同位置到达的通知发送给 CPU。为此在以下示出了多个解决方案。

- 例如可以按如下方式来产生该通知，即在 DMA 转移结束后由 I/O 单元产生一个中断，由该中断告诉 CPU 该转移已结束和再次释放被转移的存储区。作为中断的结果，读出源、也即 I/O 单元的中断状态。通过两个单元的 I/O 总线（例如 PCI 总线）的这种读取，迫使事务产生串行化，使得由 I/O 单元产生的数据按顺序被确保出现在所有处理单元的主存储器中。

- 在另一改进方案中，可以在把处理单元的 CPU 产生的指令传送给 I/O 单元时通过 CPU 在寄存器中产生一个录入项，由此触发 DMA 转移。作为替代方案，由 CPU 和 I/O 单元同时使用的脚本或列表可以作为本地存储器出现在 I/O 单元中。于是，CPU 的可能的访问就象一个存储器映射的读或写指令来进行，并且确保了所有的 CPU 利用同样的数据工作。

在另一方向上，如果由一个或多个 I/O 单元产生的 CPU 指令描述符应该出现在处理单元 PRO 的主存储器中，而且由 CPU 利用轮询方法读出，那么 CPU 就读取一个所谓的 I/O 锁定寄存器。据此，均衡器至少再也不把 I/O 单元的写事务发送到处理单元 PRO 的本地主存储器中，而且最后由 I/O 单元发送的写事务通过均衡器被写入到所有处理单元的本地主存储器中。这常被称为“刷新”（冲洗）。由此，就 I/O 单元产生的写事务而言，确保了所有处理单元的主存储器内的相同内

容。然后读取所有 CPU 的主存储器内的存储位置，其值例如指示了 I/O 指令的结束。然后，重新读或写 I/O 锁定寄存器，或者读或写一个 I/O 自由寄存器，以便通过 I/O 单元再次实现对主存储器的写访问。

－在第二改进方案中，当由一个或多个 CPU 产生的 I/O 单元指令描述符应出现在 PRO 的主存储器中、并且由 CPU 按轮询方法读出时，可以采用以下方法：由 CPU 读取一个所谓的 I/O 锁定存储器。然后，均衡器至少不再向处理单元的主存储器发送 I/O 单元的读事务。然后，所有 CPU 的主存储器的存储位置被写入以下值，该值表现为 I/O 指令的触发器。此后，重新读或写所述的 I/O 锁定寄存器，或者读或写一个 I/O 自由寄存器，以便通过 I/O 单元再次实现对主存储器的读访问。

数据比较

由 I/O 单元从主存储器读取的所有数据都通过所有的均衡器从所连接的处理单元的主存储器中完整地、或作为标记读出，且被发送给与所请求的 I/O 单元相连接的均衡器，再由该均衡器进行比较。作为替代方案，其它的均衡器同样可以进行比较。在相同的情况下，数据被继续传送给 I/O 单元。如果识别为不同，则必要时采取一种多数判定、例如“从 3 取 2”，并分离和诊断故障单元。

由处理单元的 CPU 产生的所有数据被完整地或作为标记被发送给与目标 I/O 单元相连的均衡器，并由该均衡器进行比较。作为替代方案，同样可以由其它的单元进行比较。在相同的情况下，把数据继续传递给 I/O 单元。如果出现不同，则必要时采取一种多数判定、例如“从 3 取 2”，并分离和诊断故障单元。

所有通过处理单元的 CPU 产生的读请求（例如用读命令、地址和属性来表征）完全地、或作为标记被发送给与源相连的均衡器，并由该均衡器进行比较。作为替代方案，同样可以由其它的单元进行比较。在相同的情况下，执行读事务，并把读出的数据发送给所有的均衡器。如果出现不同，则必要时采取一种多数判定、例如“从 3 取 2”，并分离和诊断故障单元。

在仿真 Lockstep 中，CPU 的读和写事务将并不就其本地主存储器 MEM 而被比较，因为它们可能是完全不同的，例如因为 CPU 的不同的猜测性访问或因为不同的高速缓存器特性等缘故。为了校验不同处理单元 PRO 的存储区内容的等同性，必须通过例如一个例行软件在一个时间点

触发校验，在该时间点可以确保存储器内容在无故障状态下是一致的，并且在校验的时段内保持一致。存储器校验本身可以通过软件来实现，也就是说由软件/CPU 读取例如一个存储区，求取校验和，并比较由不同处理单元求出的校验和。但存储器校验也可以通过硬件以如下方式来实现，即例如由布置在均衡器中的装置来读取所连接的处理单元的存储器，并求出校验和和相互比较。

具有共享存储器的多处理器结构

仿真 Lockstep 工作也适合于同步多个处理单元对一个共同的存储器（共享存储器）CMEM 的存储器访问和适合于执行上述的数据比较，前提是，事务必须满足文章开头所说的边缘条件；因此例如非高速缓存的存储器事务。

因此在一种改进方案中可以定义由多个处理器单元(具有本地存储器）的多处理器配置，它们全部能够访问一个共同的存储器 CMEM。在此，出于冗余的原因和为了故障识别，每个处理器单元被加倍，也就是说，一个处理器单元由两个相同的处理单元 PRO(未示出)组成，后者按上述的方式并行地执行所有的任务，并且还在访问共同的存储器时同步，并在此执行数据比较。

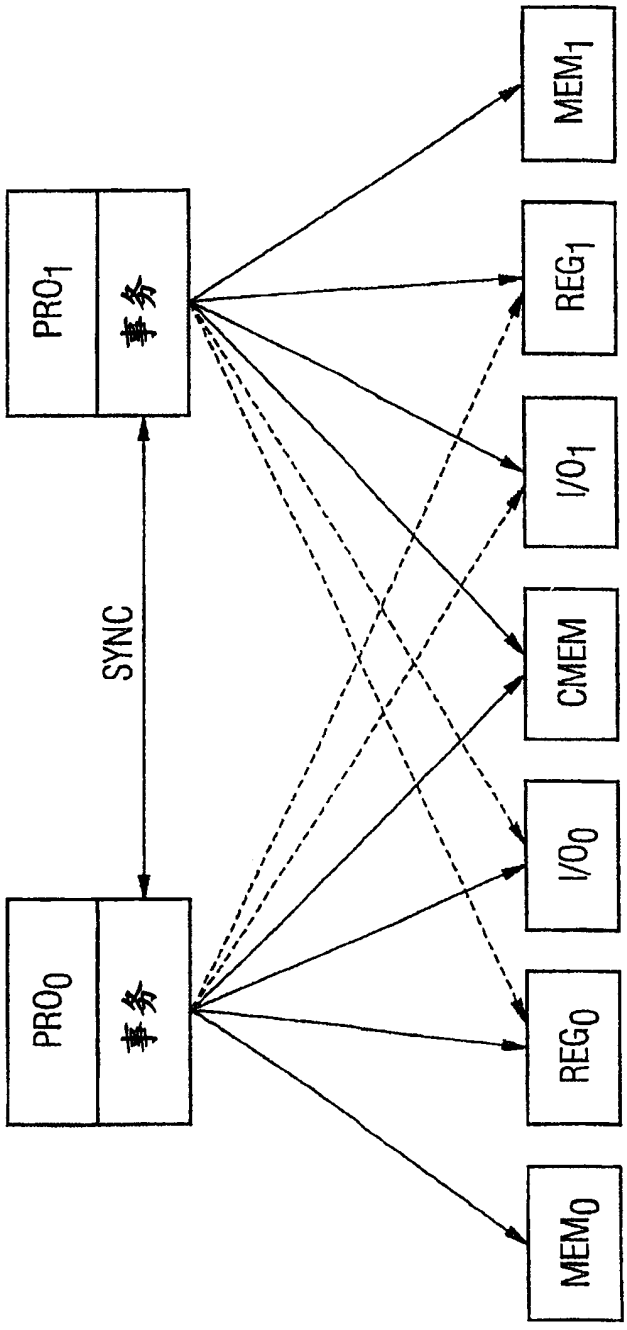


图 1

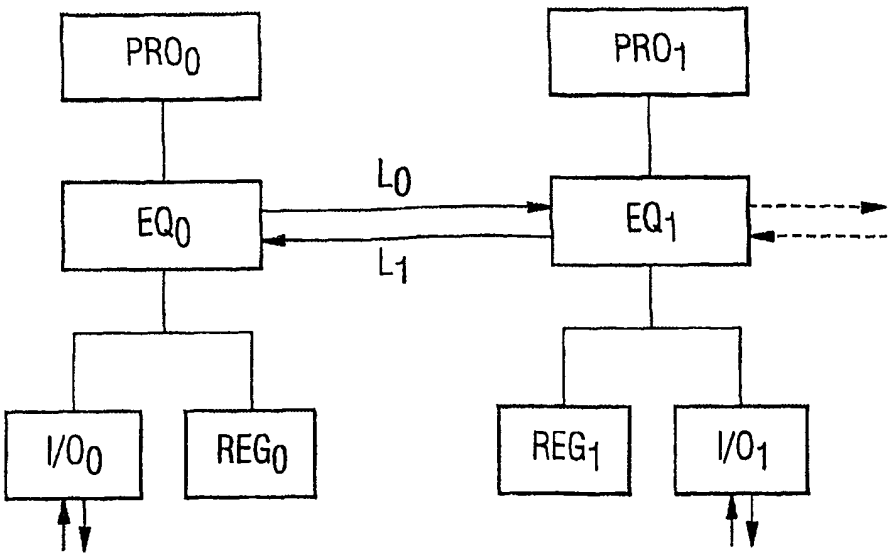

PRO0
PRO1
L0
EQ0
EQ1
L1
I/O0
REG0
REG1
I/O1

图 2

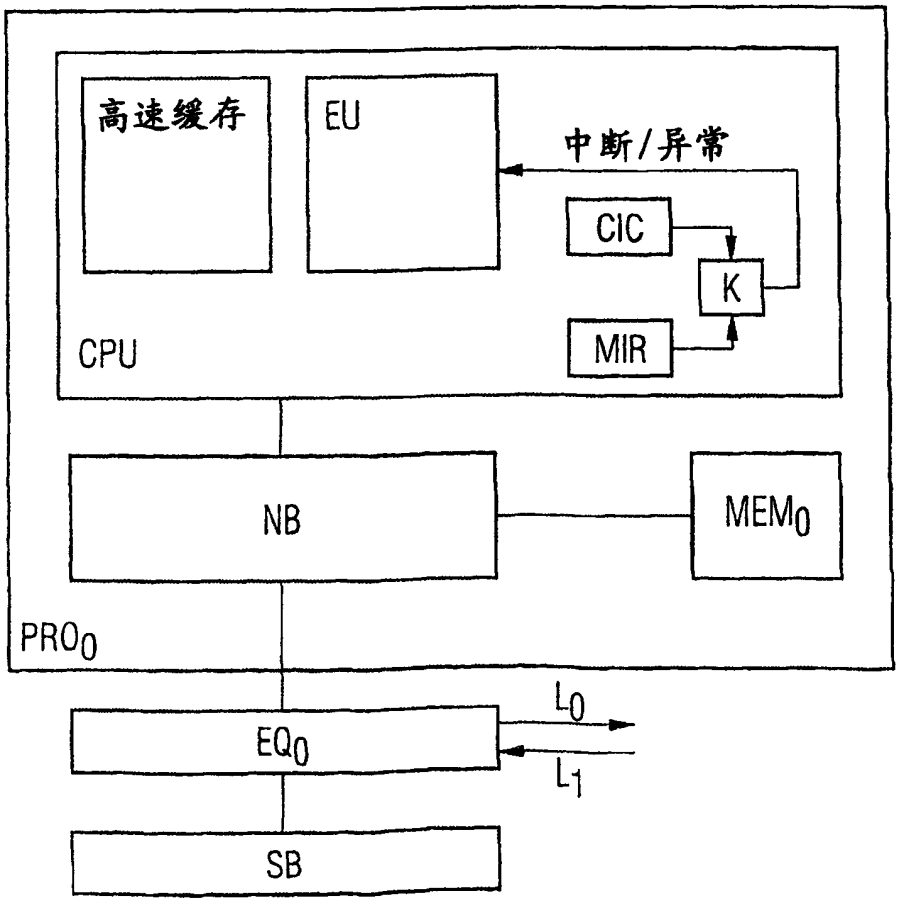

高速缓存
EU
中断/异常
CIC
K
MIR
CPU
NB
MEM0
PRO0
L0
EQ0
L1
SB

图 3

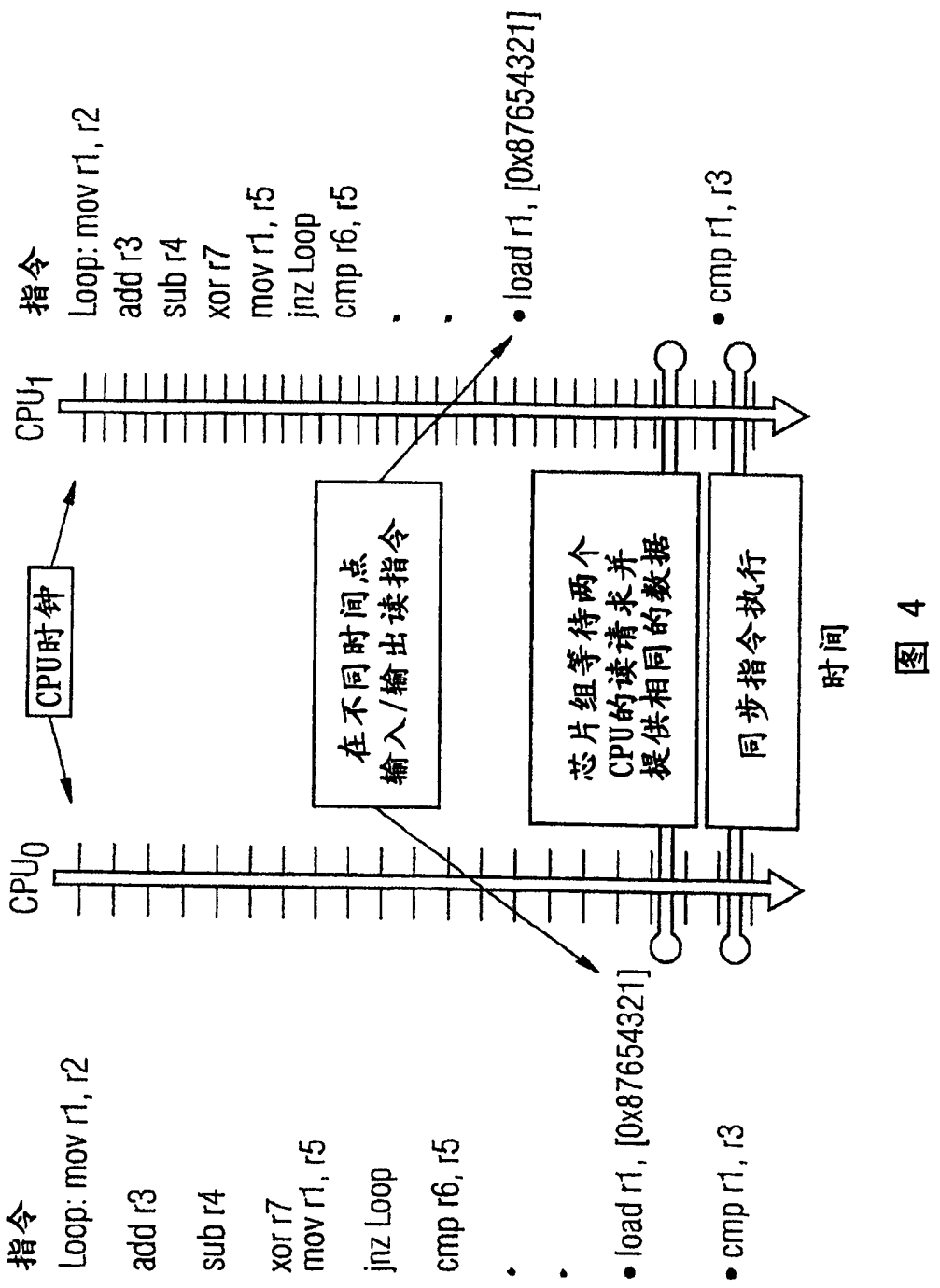

指令
Loop: mov r1, r2
add r3
sub r4
xor r7
mov r1, r5
jnz Loop
cmp r6, r5
.
.
• load r1, [0x87654321]
• cmp r1, r3
CPU0
CPU时钟
CPU1
指令
Loop: mov r1, r2
add r3
sub r4
xor r7
mov r1, r5
jnz Loop
cmp r6, r5
.
.
• load r1, [0x87654321]
• cmp r1, r3
在不同时间点
输入/输出读指令
芯片组等待两个
CPU的读请求并
提供相同的数据
同步指令执行
时间

图 4