(19)国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 115904649 A
(43)申请公布日 2023.04.04

(21)申请号 202211136935.3

(22)申请日 2022.09.19

(30)优先权数据
63/250,697 2021.09.30 US
17/561,452 2021.12.23 US

(71)申请人 英特尔公司
地址 美国加利福尼亚州

(72)发明人 G·奈格 A·玛里克 R·桑卡兰 H·沙菲 V·尚伯格 V·桑吉潘 J·布兰德特

(74)专利代理机构 上海专利商标事务所有限公司 31100
专利代理师 任曼怡 黄嵩泉

(51)Int.Cl.
G06F 9/48(2006.01)

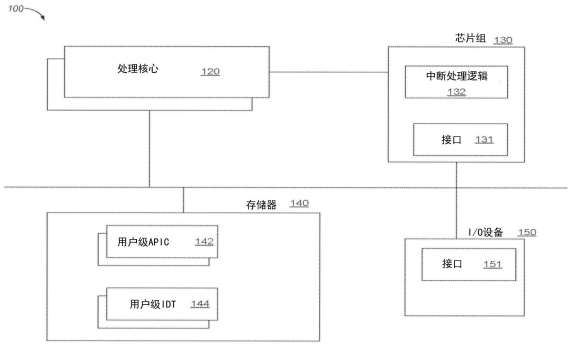
权利要求书2页 说明书39页 附图27页

(54)发明名称

用户级处理器间中断

(57)摘要

本申请公开了用户级处理器间中断。描述了用于用户级处理器间中断的处理器、方法和系统。在实施例中，处理系统包括存储器和处理核心。存储器用于存储与正在由处理系统执行的第一应用相关联的中断控制数据结构。处理核心包括指令解码器，该指令解码器用于对第一指令解码，该第一指令由第二应用调用以向第一应用发送处理器间中断；并且处理核心响应于经解码的指令而用于：确定处理器间中断的标识符与同第一应用相关联的通知中断向量相匹配，在中断控制数据结构中设置与处理器间中断的标识符相对应的待决中断标志，以及调用用于由中断控制数据结构标识的处理器间中断的中断处理程序。



CN 115904649 A

1.一种用于用户级处理器间中断的处理系统,包括:

存储器,所述存储器用于存储与正在由所述处理系统执行的第一应用相关联的中断控制数据结构;以及

处理核心,所述处理核心包括用于对第一指令解码的指令解码器,所述第一指令由第二应用调用以向所述第一应用发送处理器间中断,所述处理核心响应于经解码的指令而进行以下步骤:

确定所述处理器间中断的标识符与同所述第一应用相关联的通知中断向量相匹配,

在所述中断控制数据结构中设置与所述处理器间中断的标识符相对应的待决中断标志,以及

调用用于由所述中断控制数据结构标识的所述处理器间中断的中断处理程序。

2.如权利要求1所述的处理系统,其中,所述处理核心进一步用于:

响应于设置所述待决中断标志而触发指示所述处理器间中断的待决性的微体系结构事件。

3.如权利要求2所述的处理系统,其中,所述处理核心进一步用于:

响应于检测到所述微体系结构事件而调用所述中断处理程序。

4.如权利要求1所述的处理系统,其中,所述中断控制数据结构包括含多个待决中断比特的比特映射,其中,每个比特在所述比特映射内的位置与中断标识符相对应。

5.如权利要求1所述的处理系统,进一步包括处理逻辑,所述处理逻辑用于:

响应于将所述处理器间中断标识为用户级中断而标识与所述第一应用相关联的发布中断描述符的地址;

在所述发布中断描述符数据结构中设置与所述处理器间中断的标识符相对应的标志;以及

传送具有由所述发布中断描述符标识的中断编号的通知中断。

6.如权利要求1所述的处理系统,其中,调用所述中断处理程序进一步包括:

将指令指针的当前值存储在栈上;以及

将所述中断处理程序的地址加载到所述指令指针中。

7.如权利要求1所述的处理系统,其中,所述处理系统是片上系统SoC。

8.如权利要求1所述的处理系统,其中,调用所述中断处理程序进一步包括:

选择由所述中断控制数据结构标识的待决中断之中优先级最高的待决中断。

9.如权利要求1所述的处理系统,其中,所述通知中断向量由与所述第一应用相关联的发布中断描述符数据结构的预先定义的字段标识。

10.一种用于用户级处理器间中断的方法,包括:

由正在由处理器执行的第一应用调用第一指令,以向第二应用发送处理器间中断;

由所述处理器的指令解码器进行解码;

由所述处理器响应于经解码的指令而确定所述处理器间中断的标识符与同正在由所述处理器执行的第二应用相关联的通知中断向量相匹配;

在与所述第二应用相关联的中断控制数据结构中设置与所述处理器间中断的标识符相对应的待决中断标志;以及

调用用于由所述中断控制数据结构标识的所述处理器间中断的中断处理程序。

11.如权利要求10所述的方法,进一步包括:

响应于设置所述待决中断标志而触发指示所述处理器间中断的待决性的微体系结构事件。

12.如权利要求11所述的方法,进一步包括:

响应于检测到所述微体系结构事件而调用所述中断处理程序。

13.如权利要求10所述的方法,其中,调用所述用户级中断处理程序进一步包括:

将指令指针的当前值存储在栈上;以及

将所述中断处理程序的地址加载到所述指令指针中。

14.如权利要求10所述的方法,其中,所述中断控制数据结构包括含多个待决中断比特的比特映射,其中,每个比特在所述比特映射内的位置与中断标识符相对应。

15.如权利要求10所述的方法,进一步包括:

响应于将所述处理器间中断标识为用户级中断而标识与所述第二应用相关联的发布中断描述符的地址;

在所述发布中断描述符数据结构中设置与传入中断的标识符相对应的标志;以及

传送具有由所述发布中断描述符标识的中断编号的通知中断。

16.如权利要求10所述的方法,其中,调用所述中断处理程序进一步包括:

选择由所述中断控制数据结构标识的待决中断之中优先级最高的待决中断。

17.如权利要求10所述的方法,其中,所述通知中断向量由与所述第二应用相关联的发布中断描述符数据结构的预先定义的字段标识。

18.一种计算机可读非暂态存储介质,包括可执行指令,所述可执行指令在由用于用户级处理器间中断的处理系统执行时使得所述处理器系统进行以下步骤:

由处理器的指令解码器对由正在由处理器执行的第一应用调用的第一指令解码,所述第一指令用于向第二指令发送处理器间中断;

由所述处理器响应于经解码的指令而确定所述处理器间中断的标识符与同正在由处理器执行的第二应用相关联的通知中断向量相匹配;

在与所述第二应用相关联的中断控制数据结构中设置与所述处理器间中断的标识符相对应的待决中断标志;以及

调用用于由所述中断控制数据结构标识的所述处理器间中断的中断处理程序。

19.如权利要求18所述的计算机可读非暂态存储介质,其中,调用所述中断处理程序进一步包括:

选择由所述中断控制数据结构标识的待决中断之中优先级最高的待决中断。

20.如权利要求18所述的计算机可读非暂态存储介质,其中,所述通知中断向量由与所述第二应用相关联的发布中断描述符数据结构的预先定义的字段标识。

用户级处理器间中断

技术领域

[0001] 本发明的领域总体上涉及信息处理,并且更具体但非限制性地涉及处理器体系结构。

背景技术

[0002] 在计算机和其他信息处理系统中,中央处理单元(central processing unit,CPU)外部的输入/输出(Input/output,I/O)可受以内核特权级别操作的软件模块(驱动器)管理。来自I/O设备的通知可按中断的形式被递送至对应的驱动器。典型地,中断首先被递送至操作系统(operating system,OS)内核,该操作系统内核随后可将控制传递至适当的驱动器。

附图说明

[0003] 将参考附图来描述根据本公开的各实施例,其中:

[0004] 图1描绘根据本公开的一个或多个方面的示例处理系统的高级组件图。

[0005] 图2示意性地图示根据本公开的一个或多个方面的示例用户级中断控制器数据结构、以及与用户级线程相关联的用户级中断处理程序地址数据结构的示例。

[0006] 图3示意性地图示根据本公开的一个或多个方面、由平台硬件发布用户级中断的示例方法所采用的示例数据结构。

[0007] 图4描绘根据本公开的一个或多个方面用于由平台硬件发布用户级中断的示例方法的流程图。

[0008] 图5A-图5B描绘根据本公开的一个或多个方面用于将中断递送至用户级应用的示例方法的流程图。

[0009] 图6-图10图示根据实施例的指令的示例。

[0010] 图11图示示例系统的实施例。

[0011] 图12图示可具有多于一个的核心、可具有集成存储器控制器、并且可具有集成图形器件的处理器的实施例的框图。

[0012] 图13(A)是图示根据本发明的实施例的示例有序管线以及示例寄存器重命名、乱序发出/执行管线两者的框图。

[0013] 图13(B)是图示根据本发明的实施例的要包括在处理器中的有序体系结构核心的示例实施例和示例寄存器重命名、乱序发出/执行体系结构核心两者的框图。

[0014] 图14图示(一个或多个)执行单元电路的实施例,诸如图13(B)的(一个或多个)执行单元电路。

[0015] 图15是根据一些实施例的寄存器体系结构的框图。

[0016] 图16图示指令格式的实施例。

[0017] 图17图示寻址字段的实施例。

[0018] 图18图示第一前缀的实施例。

[0019] 图19(A)-图19(D)图示如何使用第一前缀1601(A)的R、X和B字段的实施例。

[0020] 图20(A)-图20(B)图示第二前缀的实施例。

[0021] 图21图示第三前缀的实施例。

[0022] 图22图示根据本发明的实施例的对照使用软件指令转换器将源指令集中的二进制指令转换成目标指令集中的二进制指令的框图。

具体实施方式

[0023] 本文中描述的是用于将处理器间中断递送至用户级应用的处理系统和相关方法。

[0024] 在常见实现方式中,用户级应用可经由在内核模式下执行的对应驱动器与I/O设备交互。在说明性示例中,应用可执行系统调用,这引起到内核模式的转变。OS内核通过将控制传递至设备驱动器来处理系统调用。驱动器随后可与I/O设备交互,以执行由应用请求的操作。当I/O设备完成操作时,其可经由中断来通知应用,该中断可由OS内核和设备驱动器处置。后者随后可(例如,通过调用先前由应用注册的回调处理程序)来通知应用。

[0025] 由此,应用与I/O设备的交互可引起多个特权级别转变,包括来自应用的对驱动器的系统调用、在I/O设备操作已经被发起之后从内核模式回到应用的转变、由于中断而到内核级别的转变、以及回到应用以调用应用回调处理程序的转变。那些转变可引起用户级应用与I/O设备之间每一次交互时附加的等待时间。在某些情形中,此类等待时间可能超过I/O设备的操作等待时间。由此,减少或消除由特权级别转变引起的软件等待时间可能显著改善与I/O设备进行交互的各种用户级应用的操作方面。

[0026] 根据本公开的一个或多个方面,处理系统(例如,单核心或多核心处理器)可包括被设计成用于支持新的事件(在本文中被称为“用户级中断”)的某个处理逻辑。不同于普通中断,用户级中断在不进行特权级别转变的情况下被递送至对应的用户级应用,并且仅可在此类应用正在被执行时被递送。

[0027] 用户级中断事件可调用CPU控制流修改机制,该机制在本文中被称为“用户级中断递送”。在某些实现方式中,用户级中断递送可基于处理器状态(例如,当某个用户级应用正在被执行时)并通过某些存储器数据结构的状态被触发,这些存储器数据结构的状态可由处理器和操作系统协作地管理。这些和其他软件可配置的机制和数据结构可使处理器能够在不将控制流重定向至内核级软件的情况下将某些I/O中断直接递送至用户级应用,如以下在本文中更详细地描述。

[0028] 在某些实现方式中,存储器数据结构(其通过与高级编程中断控制器(advanced programmed interrupt controller,APIC)类比而在本文中被称为“用户级APIC”)可由处理器和操作系统协作地管理。可针对正在由处理器执行的用户级应用的每个线程创建用户级APIC的单独实例。用户级APIC可包括比特映射,该比特映射包括多个比特标志。每个比特标志可指示由与比特标志在比特映射中的位置相对应的向量标识的用户级中断的状态。在说明性示例中,处理器可设置比特来指示由与比特标志在比特映射中的位置相对应的向量标识的用户级中断当前是待决的。

[0029] 操作系统可进一步维护另一存储器数据结构,该数据结构在本文中被称为用户级中断处理程序地址数据结构。可以针对正在由处理器执行的用户级应用的每个线程创建用户级中断处理程序地址数据结构的单独的实例。在某些实现方式中,用户级中断处理程序

地址数据结构可由表来表示，该表包括用户级中断处理程序的多个地址、通过中断向量进行索引(其通过与中断描述符表(Interrupt Descriptor Table，IDT)类比而被称为“用户级IDT”)。替代地，单个用户级中断处理程序地址可被存储在处理器寄存器中，以标识将负责所有的中断向量的用户级中断处理程序。在后一种场景中，用户级中断向量可被推送在中断处理程序的栈上。在以下说明书和权利要求书中，“用户级中断处理程序地址数据结构”应当要么指代以上引用的包括用户级中断处理程序的多个地址的表，要么指代以上引用的单个用户级中断处理程序地址，其可被存储在一个或多个处理器寄存器中或者被存储在系统存储器中。

[0030] 响应于在对应的用户级应用正在被执行的同时接收到用户级中断的通知，处理器可通过调用由用户级中断处理程序地址数据结构标识的用户级中断处理程序来向应用进行通知。当用户级中断处理程序完成执行时，控制流可被返回至在用户级中断被递送时被执行的软件模块。由此，用户级中断可由用户级应用在不引起特权级别转变的情况下递送和处理，如以下在本文中更详细地描述。

[0031] 在某些实现方式中，处理器可例如基于中断向量编号对用户级中断排定优先级。在说明性示例中，响应于接收到用户级中断的通知，处理器可设置和与当前正在由处理器执行的用户级应用相关联的用户级APIC中的用户级中断向量相对应的比特。如以下在本文中更详细地描述，针对由用户级APIC标识的一个或多个待决用户级中断之中具有最高优先级的待决用户级中断，处理器随后可调用由与用户级应用相关联的用户级中断处理程序地址数据结构标识的用户级中断处理程序。

[0032] 以下通过示例的方式而非通过限制的方式在本文中更详细地描述的以上引用的方法和系统的各个方面。

[0033] 在以下描述中，阐明了众多特定细节，诸如处理器和系统配置的特定类型、特定的硬件结构、特定的体系结构和微体系结构细节、特定的寄存器配置、特定的指令类型、特定的系统组件、特定的测量/高度、特定的处理器管线阶段和操作的示例，以提供对本公开的透彻理解。然而，对本领域普通技术人员将显而易见的是，不一定要采用这些特定细节来实施本文中公开的方法。在其他实例中，未详细描述公知的组件或方法，以避免不必要地使本公开模糊，公知的组件或方法诸如，特定或替代的处理器体系结构、用于所描述算法的特定逻辑电路/代码、特定的固件代码、特定的互连操作、特定的逻辑配置、特定的制造技术和材料、特定的编译器实现方式、代码中算法的特定表达、特定的掉电和功率门控技术/逻辑以及计算机系统的其他特定的操作细节。

[0034] 虽然参照处理器描述了以下示例，但其他实现方式适用于其他类型的集成电路和逻辑器件。本文中描述的示例的类似技术和教导可应用于可受益于更高的管线吞吐量和改善的性能的其他类型的电路或半导体器件。本文中所描述的示例的教导可适用于执行数据操纵的任何处理器或机器。然而，本公开不限于执行512比特、256比特、128比特、64比特、32比特、或16比特数据操作的处理器或机器，并且可应用于在其中执行数据的操纵或管理的任何处理器和机器。

[0035] 对本公开进行说明的示例和所附附图不应当以限制性意义来解释，因为它们仅仅旨在提供本文中所描述的实施例的示例，而并非对本文中所描述的实施例的所有可能实现方式进行穷举。虽然下述的示例在执行单元和逻辑电路上下文中描述指令处置和分配，但

本文中所描述的系统和方法的其他实现方式也可通过存储在机器可读有形介质上的数据或指令来完成,这些数据或指令当由机器执行时使得该机器执行与本文中所描述的至少一个实施例一致的功能。在某些实现方式中,与本文中所描述的实施例相关联的功能以机器可执行指令来具体化。这些指令可用来使以这些指令编程的通用处理器或专用处理器执行本文中所描述的方法。本文中所描述的实现方式可以作为计算机程序产品或软件来提供,该计算机程序产品或软件可以包括在其上存储有指令的机器或计算机可读介质,这些指令可以用于对计算机(或其他电子设备)编程以执行根据本文中所描述的实施例的一个或多个操作。替代地,本文中所描述的系统和方法的操作可由包含用于执行这些操作的固定功能逻辑的专用硬件组件来执行,或由经编程的计算机组件以及固定功能硬件组件的任何组合来执行。

[0036] 用于对逻辑进行编程以执行本文中所描述的方法的指令可被存储在系统中的存储器(诸如,DRAM、缓存、闪存、或其他存储装置)内。此外,指令可以经由网络或借助于其他计算机可读介质被分发。因此,机器可读介质可包括用于以机器(例如,计算机)可读形式存储或传输信息的任何机制,但不限于:软盘、光盘、致密盘只读存储器(Compact Disc,Read-Only Memory,CD-ROM)、以及磁光盘、只读存储器(ROM)、随机存取存储器(RAM)、可擦除可编程只读存储器(Read-Only Memory,EPROM)、电可擦除可编程只读存储器(Electrically Erasable Programmable Read-Only Memory,EEPROM)、磁卡或光卡、闪存、或在通过因特网经由电、光、声、或其他形式的传播信号(例如,载波、红外信号、数字信号等)传输信息时使用的有形机器可读存储。因此,计算机可读介质包括适用于以机器(例如,计算机)可读形式存储或传输电子指令或信息的任何类型的有形机器可读介质。

[0037] 在本文中,“处理器”指能够执行对算术、逻辑或I/O操作进行编码的指令的设备。在一个说明性示例中,处理器可遵循冯诺依曼体系结构模型,并且可包括算术逻辑单元(arithmetic logic unit,ALU)、控制单元以及多个寄存器。在进一步的方面中,处理器可包括一个或多个处理核心,并且因此,处理器可以是典型地能够处理单个指令管线的单核心处理器,或者可以是可以同时处理多个指令管线的多核心处理器。在另一方面,处理器可被实现为单个集成电路、两个或更多个集成电路,或者可以是多芯片模块(例如,其中各个微处理器管芯被包括在单个集成电路封装中,并且因此,这些微处理器管芯共享单个插槽)的组件。

[0038] 对“一个实施例”、“实施例”、“示例实施例”、“各实施例”等的引用指示所描述的(一个或多个)实施例可包括特定的特征、结构或特性,但是多于一个实施例可包括该特定的特征、结构或特性,并且并非每一个实施例都一定包括该特定的特征、结构或特性。一些实施例可以具有针对其他实施例所描述的特征中的一些或全部,或完全不具有这些特征。而且,此类短语不一定是指同一实施例。当结合实施例来描述特定的特征、结构或特性时,认为结合无论是否明确地描述的其他实施例来实施此类特征、结构或特性在本领域技术人员的知识范围之内。

[0039] 如在本说明书和权利要求书中所使用,除非以其他方式指定,否则对用于描述要素的序数词“第一”、“第二”、“第三”等的使用仅仅指示正在引用的要素的特定实例或类似要素的不同实例,并且不旨在暗示如此描述的这些要素在时间上、空间上、按等级或按任何其他方式必须按照特定的序列。另外,如在实施例的描述中所使用,在多个项之间的“/”字

符可意指实施例可包括第一项和/或第二项(和/或任何其他附加项),或者可使用、利用和/或根据第一项和/或第二项(和/或任何其他附加项)来实现。

[0040] 此外,术语“比特”、“标志”、“字段”、“条目”、“指示符”等可用于描述无论是以硬件还是以软件实现的、任何类型或内容的寄存器中的存储位置、表、数据库或其他数据结构,并且这些术语不旨在将实施例限于任何特定类型的存储位置或任何特定存储位置内的比特或其他元素的数量。例如,术语“比特”可用于指代寄存器内的比特位置和/或被存储在或要被存储在该比特位置中的数据。术语“清除”可用于指示将逻辑值0存储在存储位置中或以其他方式引起逻辑值0被存储在存储位置中,并且术语“清除的”可指在存储或引起已经发生之后的状态。术语“置位”可用于指示将逻辑值1、全1或某个其他指定值存储在存储位置中或以其他方式引起逻辑值1、全1或某个其他指定值被存储在存储位置中,并且术语“置位的”可指在存储或引起存储已经发生之后的状态。然而,这些术语并非旨在将本发明的实施例限于任何特定的逻辑约定,因为任何逻辑约定都可在本发明的实施例内使用。

[0041] 在本说明书中,可使用新的和/或现有的(例如,英特尔®64和IA-32体系结构)名称和/或缩写来提供简洁性,但这不旨在限制实施例。例如,名称或缩写可被用于指代寄存器(或其他存储位置)和/或被存储在寄存器中、要被存储器在寄存器中、和/或从寄存器(或其他存储位置)读取的一个值/多个值中的任一者或这两者。

[0042] 现在参考图1,所示出的是根据本公开的一个或多个方面的示例处理系统100的框图。如图1中所示,处理系统100可包括一个或多个处理核心120,该一个或多个处理核心120可经由共享互连耦合至系统存储器140。

[0043] 处理系统100可进一步包括芯片组130,该芯片组支持用于处理核心120和/或处理系统100的其他组件的存储器操作、输入/输出操作、配置、控制、内部或外部接口、连接、或通信功能、和/或其他类似功能。芯片组130的各个元件可一起被编组在单个芯片上,跨多个芯片分散,和/或部分地、总体地、冗余地或根据分布式方式被集成到包括处理器核心120的一个或多个处理器中。

[0044] 在某些实现方式中,芯片组130可包括中断处理逻辑132,该中断处理逻辑132实现根据本公开的一个或多个方面的用户级中断发布,如以下在本文中更详细地描述。替代地,中断处理逻辑132可驻留在处理系统100的其他组件中。本文中所描述的系统和方法的各实现方式可使用不变的或经修改的、当前用于普通中断处理的芯片组元件来执行用户级中断递送。

[0045] 系统存储器140可包括一个或多个介质,诸如数据和/或程序代码之类的信息可被存储在该一个或多个介质上,该一个或多个介质诸如静态或动态随机存取存储器、基于半导体的只读存储器或闪存、磁盘或光盘存储器、或可由处理核心120读取的任何其他类型的介质。

[0046] 设备150可包括可发起中断请求的任何类型的I/O设备、外围设备或其他设备,诸如键盘、鼠标、轨迹球、指点设备、监视器、打印机、介质卡、网络接口、信息存储设备等。设备150可被具体化在分立的组件中,或者可与其他设备一起集成。在某些实现方式中,设备150可代表多功能I/O、外围设备或其他设备中的功能。

[0047] 处理核心120、芯片组130、系统存储器140、以及设备150可直接地彼此耦合或者通过一个或多个并行的、顺序的、管线化的、异步的、同步的、有线的、无线的和/或其他总线或

点到点连接或通信手段间接地彼此耦合。在图1的说明性示例中，芯片组130包括接口131，接口131用于通过任何此类连接或其他通信手段从设备150接收信号、消息和/或事务(诸如中断请求)或者将信号、消息和/或事务传送至设备150和/或系统100中的任何其他代理或组件。类似地，设备150包括用于向芯片组130和/或处理系统100的任何其他代理或组件传送和/或接收信号、消息和/或事务的接口151。在某些实现方式中，处理系统100还可包括未在图1中示出的各种其他组件。

[0048] 如以下在本文中更详细地讨论，在某些实现方式中，存储器140可被用于存储与正在由处理系统100执行的多个用户级线程相关联的多个用户级APIC数据结构142和多个用户级中断处理程序地址数据结构144。替代地，用户级APIC数据结构142和/或用户级中断处理程序地址数据结构144可被存储在处理核心120的寄存器中。与给定用户线程相关联的用户级APIC数据结构142和/或用户级中断处理程序地址数据结构144的基址可被存储在被操作系统用来加载关于上下文切换的状态的状态或上下文保存(例如，XSAVE)区域中。在某些实现方式中，处理器可在上下文切换时将XSAVE区域中存储的用户级APIC数据结构142和/或用户级中断处理程序地址数据结构144的基址加载到某些处理器寄存器中。

[0049] 普通中断通过其各自的独特中断类型或中断编号(通常被称为“中断向量”)来彼此区分。用户级中断可与普通中断共享向量空间(使得每一个中断向量要么标识普通中断要么标识用户级中断)，或者可向用户级中断分配专用的用户级中断向量空间。将向量空间分开允许通过每个用户级应用对用户级中断进行独立的优先级排定，由此促进用户线程跨处理核心的动态迁移。

[0050] 如由图2示意性地图示，处理器可被配置成用于使用与正在由处理器执行的多个用户级线程相关联的多个用户级APIC数据结构142来跟踪待决用户级中断。每个用户级APIC数据结构142可包括含多个比特标志210的比特映射。比特映射内的每个比特标志的位置可与标识要由与用户级APIC数据结构相关联的用户级应用处理的用户级中断类型的中断编号相对应。响应于接收到一个或多个用户级中断的通知(例如，采用发布的中断描述符中设置的一个或多个比特的形式，如以下在本文中更详细地描述)，处理器可设置和与当前正在由处理器执行的用户级应用相关联的用户级APIC数据结构中的用户级中断向量相对应的一个或多个位。用户线程的用户级APIC数据结构的基址可被存储在该线程的XSAVE区域中。

[0051] 在某些实现方式中，处理器可维护针对用户级中断的软件控制的掩蔽比特。该掩蔽比特的某个值可阻止处理器递送任何用户级中断，直到该比特值被改变。在说明性示例中，处理器可设置掩蔽比特，以作为以下在本文中更详细地描述的用户级中断递送过程的部分而对用户级中断进行掩蔽。相反，用户级中断处理程序(或将流控制从用户级中断处理程序返回的指令)可清除该比特，以允许用户级中断递送。掩蔽比特可驻留在现有的处理器寄存器中，或者驻留在新定义的处理器寄存器中。处理器可支持用于允许对单独地或作为比特驻留在其中的寄存器的部分而对该比特进行读取和写入的新的指令。

[0052] 在某些实现方式中，处理器可例如基于中断向量编号对尚未被递送的用户级中断排定优先级。在说明性示例中，优先级可与按升序的中断向量编号相关联，使得向最低的向量编号给予最低的优先级，向次最低向量编号给予第二优先级，并且向最高向量编号给予最高优先级。处理器可选择用于递送由用户级APIC比特标识的一个或多个待决用户级中断

之中具有最高优先级的待决用户级中断。

[0053] 除了跟踪待决用户级中断之外，在某些实现方式中，处理器还可跟踪已经被递送但仍由软件进行服务的中断。在此种情况下，处理器可仅在用户级中断的向量编号超过当前正在被服务的中断的最高向量编号的情况下才递送该用户级中断。为了促进此种用户级中断递送模式，处理器可支持软件可通过其指示用户级中断服务的完成的机制(例如，由新的指令和/或新的寄存器实现)。另外，处理器可支持软件可通过其来限制要被递送的用户级中断的类型的机制(例如，通过指示应当被递送的最低向量)。

[0054] 处理器随后可通过在与用户级应用相关联的用户级中断处理程序地址数据结构中查找用户级中断处理程序地址来标识与所选择的优先级最高的中断相关联的中断处理程序。用户线程的用户级中断处理程序地址数据结构的基址可被存储在该线程的XSAVE区域中。在某些实现方式中，用户级中断处理程序地址数据结构可由表144来表示，该表144包括用户级中断处理程序220的多个地址、通过中断向量进行索引，如由图2示意性所图示。替代地，用户级中断处理程序地址数据结构可包括负责所有中断向量的单个用户级中断处理程序的地址。

[0055] 处理器随后可调用所标识的用户级中断处理程序。在说明性示例中，处理器随后可将指令指针(Instruction Pointer，IP)的当前值存储在栈上，并且将所标识的用户级中断处理程序地址加载到IP中。当调用所标识的用户级中断处理程序时，处理器还可为用户级中断设置掩蔽比特，以对用户级中断进行掩蔽；包含该掩蔽比特的处理器寄存器可以在处理器早先存储在栈上的那些处理器寄存器之中。用户级中断处理程序可将CPU状态的一个或多个分量(例如，处理器寄存器)的值保存在存储器中或保存在栈上。用户级中断处理程序可包括被设计成用于处理待决中断(例如，将某些数据从与I/O设备相关联的存储器缓冲区复制到由应用分配的存储器缓冲器中，或者反之亦然)的多个指令。用户级中断处理程序可通过恢复所保存的CPU状态、执行返回(例如，RET)指令(该返回指令使IP加载有被存储在栈顶的地址)来完成，由此将控制流传递至在用户级中断被递送之前已经被执行的指令之后的指令。在某些实现方式中，从用户级中断处理程序返回流控制可通过使用现有的返回指令来执行。替代地，可向指令集添加新的指令，以用于实现从用户级中断处理程序的控制流返回。此类新的指令可清除用于用户级中断的掩蔽比特以对用户级中断解除掩蔽；替代地，该指令可从栈加载包含该掩蔽比特的处理器寄存器。在各说明性示例中，可采用调用所标识的中断处理程序的其他实施例。

[0056] 在某些实现方式中，当来自I/O设备的通知通过平台硬件(例如，芯片组130或处理器非核心)被传送至CPU时，此类通知可明确地被标记为用户级中断。替代地，通知可通过具有普通中断向量的普通中断消息来实现。

[0057] 处理器可基于向量编号或某种其他属性来确定哪些中断应当被视为用户级中断。在说明性示例中，操作系统可维护具有与多个普通中断向量相对应的多个条目的表。中断映射表的每个条目可包括指示处理器是否应当将对应的中断转换为用户级中断的比特标志。在某些实现方式中，中断映射表的每个条目可进一步包括应当被用于对应的中断递送的用户级中断向量。替代地，当响应于接收到普通中断消息而递送用户级中断时，处理器可保持普通中断的向量。在某些实现方式中，中断请求可被标识为内核中断或用户级中断，从而允许其通过单独的中断映射表被独立地处理。在另一实现方式中，中断请求可通过共同

的中断映射表来处理，其中被用于映射该中断请求的条目被编程为用于指定该中断请求是作为内核中断还是作为用户级中断来处理。

[0058] 在某些实现方式中，平台硬件(例如，芯片组130或处理器非核心)可通过将与来自I/O设备的中断有关的信息记录在与关联于传入的中断向量的用户级应用相关联的存储器数据结构中来处理该中断。如图3示意性地图示，对于可接收用户级中断的每个用户线程302，操作系统可创建并维护存储器数据结构310，该存储器数据结构310在本文中被称为“发布中断描述符”。发布中断描述符310可包括比特映射，该比特映射包括多个比特标志314。每个比特标志314可指示由与比特标志在比特映射中的位置相对应的向量标识的用户级中断的状态。在说明性示例中，平台硬件可设置与待决用户级中断向量相对应的比特。

[0059] 如以下在本文中所描述，发布中断描述符310可进一步包括通知中断向量316，该通知中断向量316标识平台硬件可用来向处理器通知待决用户级中断的普通中断。发布中断描述符310可进一步包括可由软件和/或处理系统的其他代理使用的一个或多个控制比特318。

[0060] 操作系统可进一步创建和维护包括比特映射322的中断映射表320，该比特映射322包括多个比特标志324。每个比特标志324可指示处理器是否应当将来自由该比特标志在比特映射322内位置标识的中断源的中断是否应当被转换为用户级中断。中断映射表322可进一步包括发布中断描述符的标识符(例如，基址)328的列表326。每个发布中断描述符可与用户线程相关联，该用户线程与由该发布中断描述符在列表326内的位置标识的普通中断的源相关联。

[0061] 在某些实现方式中，中断映射表320可进一步包括应当被用于对应的用户级中断递送的中断向量329的列表327。替代地，当响应于接收到普通中断消息而递送用户级中断时，处理器可保持普通中断的向量。

[0062] 响应于接收到来自I/O设备的中断，平台硬件可在中断映射表320中查找中断源标识符(例如，由设备编号表示)。如果与中断源相关联的比特标志324指示中断应当作为用户级中断来处理，则平台硬件可从中断映射表320取回与中断源相关联的发布中断描述符310的基址328。平台硬件随后可在发布中断描述符中设置比特标志314来指示对应的用户级中断是待决的。比特标志314随后将被处理器清除，以确认发布中断的接收。在设置比特标志314之后，平台硬件可向处理器传送具有由发布中断描述符310的通知中断向量字段316标识的向量的通知中断。

[0063] 响应于接收到中断，处理器可将中断向量与当前正在由处理器执行的用户线程的通知中断向量进行比较。在说明性示例中，当前用户线程的通知中断向量可由发布中断描述符310标识。替代地，当前用户线程的通知中断向量可由控制寄存器标识，该控制寄存器是由操作系统可编程的。响应于确定了传入的中断并非通知中断，处理器可使用共同中断递送机制(例如，将控制流传递至通过IDT查找标识的中断处理程序)来递送传入的中断。替代地，响应于确定该传入的中断向量与通知中断向量相匹配，处理器可处理发布用户级中断。

[0064] 发布用户级中断的处理可包括由处理器读取与当前用户线程相关联的发布中断描述符310的比特标志314。处理器可在与当前用户线程相关联的用户级APIC数据结构142中、在发布中断描述符310的已设置的比特标志314的位置处设置比特标志210。处理器随后

可触发微体系结构事件以指示待决用户级中断的存在。响应于检测到微体系结构事件，处理器可选择优先级最高的用户级中断来递送，通过查找用户级中断处理程序地址数据结构来标识用于所选择的用户级中断的中断处理程序，并且调用所标识的中断处理程序，如以下在本文中参考图5B更详细地描述。

[0065] 图4描绘根据本公开的一个或多个方面的用于由平台硬件发布用户级中断的示例方法的流程图。方法400可由计算机系统执行，该计算机系统可包括硬件(例如，电路、专用逻辑和/或可编程逻辑)、软件(例如，在计算机系统上可执行以执行硬件仿真的指令)或其组合。方法400和/或其功能、例程、子例程或操作中的每一者可由一个或多个物理处理器和/或执行该方法的计算机程序的其他组件执行。在一个示例中，如由图4所图示，方法400可由以下在本文中所描述并如由图6-图12所图示的处理系统执行。

[0066] 参考图4，在框410处，实现方法的平台硬件可接收来自I/O设备的中断。

[0067] 在框420处，平台硬件可在中断映射表中查找中断源标识符(例如，由设备编号表示)，该中断映射表由操作系统管理、用于向平台硬件指示哪些中断应当被视为用户级中断，如以上在本文中更详细地描述。

[0068] 响应于在框430处确定了与中断源相关联的比特标志指示中断应当作为用户级中断来处理，则处理可在框440处继续；否则，方法可分支到框470，以执行共同硬件中断递送机制。

[0069] 在框440处，平台硬件可从中断映射表取回与中断源相关联的发布中断描述符的基址，如以上在本文中更详细地描述。

[0070] 在框450处，平台硬件可在发布中断描述符中设置比特标志来指示对应的用户级中断是待决的。

[0071] 在框460处，如以上更详细地描述，平台硬件可向处理器传送具有由发布中断描述符的通知中断向量字段标识的向量的通知中断，并且方法可以终止。

[0072] 图5A-图5B描绘根据本公开的一个或多个方面的用于将中断递送至用户级应用的示例方法的流程图。具体而言，方法500A图示对通知中断的处理，并且方法500B图示用户级中断递送。方法500A-500B可由计算机系统执行，该计算机系统可包括硬件(例如，电路、专用逻辑和/或可编程逻辑)、软件(例如，在计算机系统上可执行以执行硬件仿真的指令)或其组合。方法500A-500B和/或其功能、例程、子例程或操作中的每一者可由一个或多个物理处理器和/或执行该方法的计算机程序的其他组件执行。在一个示例中，如由图5A-图5B所图示，方法500A-500B可由以下在本文中所描述并如由图6-图12所图示的处理系统执行。

[0073] 参考图5A，在框510处，实现方法的处理器可接收来自平台的中断。

[0074] 响应于在框520处确定了传入的中断向量与当前用户线程的通知中断向量相匹配，处理可在框530处继续；否则，方法可分支到框550以执行共同中断递送机制。

[0075] 在框530处，处理器可在与当前用户线程相关联的用户级APIC数据结构中、在与当前用户线程相关联的发布中断描述符的已设置的比特标志的位置处设置比特标志。

[0076] 在框540处，处理器可基于用户级APIC的状态触发微体系结构事件来指示待决用户级中断的存在，以便触发方法500B来进行待决用户级中断的递送。

[0077] 方法500B图示用户级中断递送。该方法可由方法500A引起的、用于指示待决用户级中断的存在的微体系结构事件来触发。由此，方法500B可在微体系结构事件被断言之后

的某个时间点被调用,这取决于若干种条件,包括例如,用户级中断是否被掩蔽以及当前特权级别是否允许用户级中断处理。

[0078] 在框560处,处理器可检测指示待决用户级中断的存在的微体系结构事件。

[0079] 在框570处,处理器可选择用于递送由用户级APIC比特标识的一个或多个待决用户级中断之中具有最高优先级的待决用户级中断,如以上在本文中更详细地描述。

[0080] 在框580处,处理器可通过在与用户级应用相关联的用户级中断处理程序地址数据结构中查找中断处理程序地址来标识与所选择的优先级最高的中断相关联的中断处理程序,如以上在本文中更详细地描述。

[0081] 在框590处,如以上在本文中更详细地描述,处理器可调用所标识的用户级中断处理程序,并且方法可终止。

[0082] 本文中详述的示例涉及被称作用户中断的体系结构特征。此种特征将用户中断定义为事件。用户中断在不改变分段状态的情况下被递送至在无特权/用户模式下(例如,在当前特权级别(current privilege level,CPL)为3的64比特模式下)操作的软件。不同的用户中断通过用户中断向量(例如,64比特向量)来区分,作为用户中断递送的部分,该用户中断向量被推送在栈上。在一些示例中,用户中断返回(user-interrupt return,UIRET)指令的执行使用户中断递送反转。

[0083] 在一些示例中,用户中断体系结构通过管理程序管理的状态来配置。此种状态包括多个型号(或机器)特定寄存器(model specific register或machine specific register,MSR)。在预期使用中,操作系统(operating system,OS)将在OS管理的线程之间切换时更新这些MSR的内容。

[0084] MSR中的一个MSR引用被称作用户发布中断描述符(user posted-interrupt descriptor,UPID)的数据结构。针对OS管理的线程的用户中断可以在与该线程相关联的UPID中发布。此类用户中断将在被称作用户中断通知的普通中断(也在UPID中标识)的接收之后被递送。

[0085] 系统软件可以定义用于发布用户中断和用于发送用户中断通知的操作。另外,在一些示例中,指令SENDUIPI允许应用软件发送处理器间用户中断(inter-processor user interrupt,用户IPI)。SENDUIPI的执行在UPID中发布用户中断并发送用户中断通知。

[0086] 枚举和启用

[0087] 在一些示例中,通过设置控制寄存器(例如,CR4)中的比特(例如,比特25(UINTR))来启用用户中断。设置CR4.UINTR启用用户中断递送、用户中断通知标识、以及用户中断指令。这可能不会通过读取MSR(read MSR,RDMSR)、写入MSR(write MSR,WRMSR)或上下文保存(XSAVE)特征集合影响用户中断MSR的可访问性。

[0088] 在一些示例中,通过CPUID.(EAX=7,ECX=0):EDX[5]来枚举对用户中断的处理器支持。如果该位被设置,则软件可将CR4.UINTR设置为1并且可以使用RDMSR和WRMSR来访问用户中断MSR。

[0089] 在一些示例中,用户中断特征是XSAVE管理的。

[0090] 用户中断状态和用户中断MSR

[0091] 在一些示例中,用户中断体系结构定义以下状态。该状态中的一些可以经由RDMSR和WRMSR指令(诸如通过用户中断MSR)访问,并且一些可以使用本文中所描述的指令来访

问。

[0092] 用户中断状态

[0093] 在实施例中,以下是用户中断状态的要素(此处与它们如何被访问无关地枚举):

• UIRR:用户中断请求寄存器。该值针对64个用户中断向量中的每一个包括一个位。如果UIRR[i]=1,则具有向量i的用户中断请求服务。注释UIRRV被用于指代UIRR中被设置的最高有效比特的位置;如果UIRR=0,UIRRV=0。

• UIF:用户中断标志。如果UIF=0,则用户中断递送受阻;如果UIF=1,则用户中断可被递送。用户中断递送清除UIF,并且新的UIRET指令设置UIF。

• UIHANDLER:用户中断处理程序。这是用户中断处理程序的线性地址。用户中断递送将该地址加载到RIP中。

• UISTACKADJUST:用户中断栈调整。该值控制用户中断递送之前对栈指针(RSP)的调整。其可以计及OS应用二进制接口(application binary interface,ABI)的“红区(red zone)”,或者被配置成用于用替代的栈指针来加载RSP。如果比特0为1,则用户中断递送用UISTACKADJUST加载RSP;否则,其从RSP减去UISTACKADJUST。以任一方式,用户中断递送随后将RSP与16字节边界对齐。

• UINV:用户中断通知向量。这是被视为用户中断通知的那些普通中断的向量。当逻辑处理器接收到用户中断通知时,该逻辑处理器处理通过UPIDADDR引用的用户发布中断描述符(UPID)中的用户中断。

• UPIDADDR:用户发布中断描述符地址。这是逻辑处理器在接收到具有向量UINV的普通中断时查阅的UPID的线性地址。

• UITTADDR:用户中断目标表地址。这是逻辑处理器在软件调用SENDUIPI指令时查阅的用户中断目标表(user-interrupt target table,UITT)的线性地址。

• UITTSZ:用户中断目标表大小。该值是UITT中的有效条目的最高索引。

[0094] 用户中断MSR

[0095] 在实施例中,所标识的状态要素中的一些可以作为用户中断MSR、使用RDMSR和WRMSR指令来访问:

• IA32_UINTR_RR MSR(MSR地址985H)。该MSR是到UIRR的接口(64比特)。

• IA32_UINTR_HANDLER MSR(MSR地址986H)。该MSR是到UIHANDLER地址的接口。

• IA32_UINTR_STACKADJUST MSR(MSR地址987H)。该MSR是到UISTACKADJUST值的接口。

• IA32_UINTR_MISC MSR(MSR地址988H)。该MSR是到UITTSZ和UINV值的接口。在一些示例中,MSR具有以下格式:比特31:0为UITTSZ;比特39:32为UINV;并且比特63:40被保留。因为该MSR将与UIF共享XSAVE区域的8字节部分,因此MSR的比特63被保留。

• IA32_UINTR_PD MSR(MSR地址989H)。该MSR是到UPIDADDR地址的接口。

• IA32_UINTR_TT MSR(MSR地址98AH)。该MSR是到UITTADDR地址的接口(另外,比特0启用SENDUIPI)。

[0096] 用户中断的评估和递送

[0097] 在实施例中,处理器基于UIRR来确定是否存在要递送的用户中断。一旦逻辑处理器已经识别到待决用户中断,该逻辑处理器将通过引起与软件执行异步的控制流改变在后

续指令边界上递送该待决用户中断。

[0098] 用户中断识别

[0099] 每当UIRR不等于0时,就存在待决的用户中断。

[0100] 对UIRR进行修改的任何指令或操作均更新逻辑处理器对待决用户中断的识别。以下指令和操作可能需要这样做:

- 对IA32_UINTR_RR MSR的WRMSR
- 用户中断状态分量的XRSTORS。
- 用户中断递送。
- 用户中断通知处理。
- VMX转变,其加载IA32_UINTR_RR MSR。

[0101] 这些指令或操作中的每一者在其以UIRR不等于0完成的情况下均引起对待决用户中断的识别;如果其以UIRR=0完成则没有待决用户中断被识别。一旦被识别,则待决用户中断可被递送至软件。

[0102] 用户中断递送

[0103] 在一些示例中,如果CR4.UINTR=1且用户中断已经被识别,则该用户中断将在以下条件全部成立时所在的指令边界处被递送:(1)UIF=1;(2)不存在通过MOV SS或通过POP SS造成的阻塞;(3)CPL=3;(4)IA32_EFER.LMA=CS.L=1(逻辑处理器处于64比特模式);以及(5)软件并非正在受保护容器(例如,飞地)内执行。

[0104] 用户中断递送具有仅低于普通中断的优先级的优先级。它将逻辑处理器从使用TPAUSE和UMWAIT指令进入的状态唤醒;它不会唤醒处于关机状态或处于等待SIPI状态的逻辑处理器。

[0105] 用户中断递送不改变CPL(其以CPL=3整体地发生)。以下伪代码提供用户中断递送的示例:

```
IF UIHANDLER is not canonical in current paging mode
THEN#GP(0);
FI;
tempRSP:=RSP;
IF UISTACKADJUST[0]=1
THEN RSP:=UISTACKADJUST;
ELSE RSP:=RSP-UISTACKADJUST;
FI;
RSP:=RSP&~FH;                    //迫使栈成为16字节对齐的
Push tempRSP;
Push RFLAGS;
Push RIP;
Push UIRRV;//64比特推送;较高的58比特被推送为0
IF shadow stack is enabled for CPL=3
THEN ShadowStackPush RIP;
FI;
```

```
IF end-branch is enabled for CPL＝3
THEN IA32_U_CET.TRACKER:＝WAIT_FOR_ENDBRANCH;
FI;
UIRR[Vector]:＝0;
IF UIRR＝0
THEN cease recognition of any pending user interrupt;
FI;
UIF:＝0;
RFLAGS.TF:＝0;
RFLAGS.RF:＝0;
RIP:＝UIHANDLER;
```

[0106] 如果UISTACKADJUST[0]＝0,则用户中断递送将RSP递减UISTACKADJUST;否则它用UISTACKADJUST加载RSP。在任一情况下,用户级递送通过清除RSP[3:0]将RSP与16字节边界对齐。

[0107] 在事务性执行期间发生的用户中断递送引起事务性执行中止以及到非事务性执行的转变。如果是由于普通中断,事务性中止会按原样加载EAX。用户中断递送在事务性中止被处理之后发生。

[0108] 由用户中断递送执行的栈访问可能引发错误(页错误、或由于正则性违反造成的栈错误)。RSP被恢复至RSP在此类错误被递送(栈顶上方的存储器位置可能已经被写入)之前的原始值。如果此类错误产生使用EXT比特的错误代码,则该比特将被清除为0。

[0109] 如果发生此类错误,则UIRR不被更新并且UIF不被清除,并且结果是,逻辑处理器继续识别用户中断是待决的并且用户中断将在该错误被处置之后正常地重新发生。

[0110] 如果控制流实施技术(control-flow enforcement technology,CET)的影子栈特征针对CPL＝3被启用,则用户中断递送将返回指令指针推送到影子栈上。如果CET的间接分支跟踪特征被启用,则用户中断递送将间接分支跟踪器转变至WAIT_FOR_ENDBRANCH(等待_结束分支)状态;ENDBR64指令被预期作为用户中断处理程序的第一指令。

[0111] 用户中断递送可以通过以下各项中的一项或多项来跟踪:体系结构最后分支记录(Architectural Last Branch Record,LBR)、处理器追踪(Processor Trace,PT)和性能监视。对于PT和LBR两者,用户中断递送以与普通中断递送相同的方式被记录。由此,对于LBR,用户中断落入OTHER_BRANCH(其他_分支)类别中,这暗示IA32_LBR_CTL.OTHER_BRANCH[比特22]被设置以记录用户中断递送,并且IA32_LBR_x_INFO.BR_TYPE字段将指示针对任何所记录的用户中断的OTHER_BRANCH。对于英特尔PT,控制流追踪通过设置IA32_RTIT_CTL.BranchEn[比特13]来启用。

[0112] 用户中断递送也将递增性能计数器,对于该性能计数器,对BR_INST_RETIRED.FAR_BRANCH计数被启用。一些实现方式可具有用于对用户中断计数的专用事件。

[0113] 用户中断通知标识和处理

[0114] 用户中断发布是这样的过程:通过该过程,平台代理(或在CPU上操作的软件)将用户中断记录在存储器中的用户发布中断描述符(UPID)中。平台代理(或软件)可将普通中断(被称作用户中断通知)发送至逻辑处理器,用户中断的目标正在该逻辑处理器上操作。

[0115] UPID可具有如表格1中所示的格式：

比特位置	名称	描述
0	未完成通知	如果该比特被设置，则存在针对 PIR 中的一个或多个用户中断的未完成的通知。
1	抑制通知	如果该比特被设置，则代理（包括 SENDUIPI）在该描述符中发布用户中断时不应当发送通知。
15:2	保留	用户中断通知处理忽略这些比特；针对 SENDUIPI，必须为零。
23:16	通知向量	由 SENDUIPI 使用。
31:24	保留	用户中断通知处理忽略这些比特；针对 SENDUIPI，必须为零。
63:32	通知目的地	目标物理 APIC ID –由 SENDUIPI 使用。在 xAPIC 模式下，比特 47:0 是 8 比特 APIC ID。在 x2APIC 模式下，整个字段形成 32 比特 APIC ID。
127:64	发布中断请求（Posted-interrupt requests，PIR）	针对每个用户中断向量存在一个比特。如果对应的比特为 1，则存在对向量的用户中断请求。

表格1

[0116] 注释PIR(发布中断请求)是指UPID中的64个发布中断请求。

[0117] 如果普通中断在CR4.UINTR＝IA32_EFER.LMA＝1时到达，则逻辑处理器确定中断是否为用户中断通知。该过程被称作用户中断通知标识。

[0118] 一旦逻辑处理器已经标识了用户中断通知，其就将UPID的PIR中的用户中断复制到UIRR中。该过程被称作用户中断通知处理。

[0119] 在用户中断通知标识期间或在用户中断通知处理期间或在那些操作之间(当它们接连地发生时),逻辑处理器不是可中断的。

[0120] 用户中断通知标识

[0121] 如果CR4.UINTR＝IA32_EFER.LMA＝1,逻辑处理器在其接收到普通中断时执行用户中断通知标识。以下方法提供当CR4.UINTR＝IA32_EFER.LMA＝1时处理器对普通中断的响应的示例:

• 本地APIC被确认;这向处理器核心提供中断向量V。

• 如果V＝UINV,则逻辑处理器继续下一动作。否则,具有向量V的中断通过IDT正常地被递送;不应用该方法的其余部分并且不发生用户中断通知。

• 处理器将零写入本地APIC中的EOI寄存器;这从本地APIC解除带有向量V＝UINV的中断。

[0122] 用户中断通知标识涉及本地APIC的确认,并且由此仅在普通中断不被掩蔽时发生。

[0123] 如果用户中断通知标识完成第三动作,则逻辑处理器随后执行用户中断通知处理。

[0124] 在事务性执行期间发生的普通中断引起事务性执行中止以及到非事务性执行的转变。这在用户中断通知标识之前发生。

[0125] 在软件在受保护容器(例如,飞地)内执行的同时发生的普通中断引起异步飞地退出(asynchronous enclave exit,AEX)。该AEX在用户中断通知标识之前发生。

[0126] 用户中断通知处理

[0127] 一旦逻辑处理器已经标识用户中断通知,则其使用IA32_UINTR_PD MSR中的线性地址处的UPID来执行用户中断通知处理。

[0128] 以下方法提供用户中断通知处理的示例:

• 逻辑处理器清除UPID中的未完成通知比特(比特0)。这原子地被完成,以便使描述符的其余部分不被修改(例如,利用锁定的AND(与)操作)。

• 逻辑处理器将PIR(例如,UPID的比特127:64)读取到临时寄存器中,并且向PIR写入全0。这原子地被完成,以便确保被清除的每个比特在临时寄存器中被设置(例如,利用锁定的XCHG操作)。

• 如果临时寄存器中的任何比特均被设置,则逻辑处理器在UIRR中设置与临时寄存器中被设置的比特相对应的每个比特(例如,利用OR(或)操作),并且识别待决的用户中断(如果其尚未这样做)。

[0129] 在一些示例中,逻辑处理器以不可中断的方式执行以上动作。前两个动作可被组合成单个原子性步骤。如果第三动作引起用户中断的识别,则处理器可在随后的指令边界上递送该中断。

[0130] 在一些示例中,虽然用户中断通知处理可在任何特权级别下发生,但前两个动作中的所有存储器访问均以管理程序特权被执行。

[0131] 前两个动作各自使用线性地址来访问UPID,并且因此可能引发错误(页错误或由于正则性违反造成的通用保护错误)。如果此类错误产生使用EXT比特的错误代码,则该比特将被设置为1。

[0132] 如果发生此类错误,则对由早先用户中断通知标识执行的体系结构状态的更新保持被提交并且并非未完成的;如果此类错误在第二动作处发生(如果该第二动作并非与第一动作原子性地被执行),则对由第一动作执行的体系结构状态的任何更新也保持被提交。(例如,通过确保不发生页错误并且IA32_UINTR_PD MSR中的线性地址相对于使用中的分页模式是正则的)建议系统软件防止此类错误。

[0133] 用户中断通知处理之前的用户中断通知标识可能由于普通中断、虚拟中断或由VM进入注入的中断而发生。以下各项指定针对用户中断通知处理的这些情况中的每种情况逻辑处理器的活动状态:

• 如果用户中断通知标识是由于普通中断或虚拟中断并且逻辑处理器在该中断前已处于HLT状态,则其在用户中断通知处理之后返回至HLT状态。

• 如果用户中断通知标识是与由VM进入注入的中断并且VMCS的宾客机状态区域中的活动状态字段指示HLT状态,则逻辑处理器在用户中断通知处理之后进入HLT状态。

• 在其他情况下,在用户中断通知处理之后,逻辑处理器处于活跃状态。

[0134] 指令

[0135] 在实施例中,用户中断体系结构包括用于控制流转移和对新状态的访问的新指令。UIRET是用于实现从用户中断处理程序的返回的一种新指令。其他新指令允许由用户代码对UIF的访问。用户IPI还使用新指令SENDUIPI。这些指令的示例在附图中。

[0136] 用户IPIS

[0137] 处理器支持处理器间用户中断(interprocessor user interrupt,用户IPI)通过(由系统软件配置的)用户中断目标表和(由应用软件执行的)SENDUIPI指令的发送。

[0138] 用户中断目标表(user-interrupt target table,UITT)是由16字节条目组成的数据结构。每个UITT条目(UITT entry,UITTE)具有以下格式:

• 比特0:V,有效比特。

• 比特7:1被保留并且固定为0。

• 比特15:8:UV,用户中断向量(在范围0-63中,因此比特15:14固定为0)。

• 比特63:16被保留。

• 比特127:64:UPIDADDR,UPID的线性地址(64字节对齐的,因此比特69:64固定为0)。

[0139] 在一些实施例中,UITT位于线性地址UITTADDR处;UITTSZ是UITT中的有效条目的最高索引(由此,UITT中的条目数量为UITTSZ+1)。

[0140] 现有指令支持

[0141] 在一些示例中,某些指令支持用户中断体系结构。RDMSR和WRMSR指令支持对用户中断MSR的访问。体系结构还由XSAVE特征集合来支持。

[0142] 由RDMSR和WRMSR提供的支持

[0143] RDMSR和WRMSR支持针对用户中断MSR的正常读取和写入操作。即使CR4.UINTR=0,这些操作也被支持。以下项目标识特定于这些MSR的点:

• IA32_UINTR_RR MSR(MSR地址985H)。该MSR保存UIRR的当前值。在对该MSR的WRMSR之后,如果在该MSR中某个比特被设置并且仅在该比特被设置的情况下,逻辑处理器识别待决用户中断。

• IA32_UINTR_HANDLER MSR(MSR地址986H)。该MSR保存UIHANDLER的当前值。这是必须相对于处理器支持的最大线性地址宽度正则的线性地址。对该MSR的WRMSR在其源操作对象不满足该要求的情况下导致通用保护错误(#GP)。

• IA32_UINTR_STACKADJUST MSR(MSR地址987H)。该MSR保存UISTACKADJUST的当前值。该值包括相对于处理器支持的最大线性地址宽度建立的线性地址。如果不是,则对该MSR的WRMSR导致通用保护错误(#GP)。该MSR的比特0对应于UISTACKADJUST[0],其控制用户中断递送如何更新栈指针。WRMSR可将其设置为0或1。

• IA32_UINTR_MISC MSR(MSR地址988H)。该MSR的比特31:0保存UITTSZ的当前值,而比特39:32保存UINV的当前值。该MSR的比特63:40被保留。如果WRMSR将那些比特中的任何比特置位(如果EDX[31:8]不等于000000H),则WRMSR导致#GP。因为该MSR与UIF共享XSAVE区域的8字节部分,因此MSR的比特63将不被使用并且将被保留。

• IA32_UINTR_PD MSR(MSR地址989H)。该MSR保存UPIDADDR的当前值。这是相对于处理器支持的最大线性地址宽度正则的线性地址。如果不是,则对该MSR的WRMSR导致通用保护错误(#GP)。该MSR的比特5:0被保留。如果WRMSR将那些比特中的任何比特置位(如果EAX[5:0]不等于000000b),则WRMSR导致#GP。

• IA32_UINTR_TT MSR(MSR地址98AH)。该MSR的比特63:4保存UITTADDR的当前值。这是相对于处理器支持的最大线性地址宽度正则的线性地址。如果不是,则对该MSR的WRMSR导致通用保护错误(#GP)。该MSR的比特3:1被保留。如果WRMSR将那些比特中的任何比特置位(如果EAX[3:1]不等于000b),则WRMSR导致#GP。该MSR的比特0确定SENDUIPI指令是否被启用。WRMSR可将其设置为0或1。

[0144] 由XSAVE特征集合提供的支持

[0145] 上文所标识的状态可特定于OS管理的用户线程,并且系统软件随后在改变用户线程时需要改变该状态的值。通过将对此种状态的支持添加到XSAVE特征集合来促进此种上下文管理。该章节描述此种支持。

[0146] XSAVE特征集合支持状态分量的保存和恢复,这些状态分量中的每一者是分立的处理器寄存器集合(或寄存器的部分)。每个此类状态分量与XSAVE支持的特征相对应。XSAVE特征集合使用状态分量比特映射来组织XSAVE支持的特征的状态分量。状态分量比特映射包括64比特;此类比特映射中的每个比特与单个状态分量相对应。一些状态分量是管理程序状态分量。XSAVE特征利用仅XSAVES和XRSTORS指令来支持管理程序状态分量。

[0147] 用户中断状态分量

[0148] XSAVE特征集合将利用管理程序用户中断状态分量来管理用户中断寄存器。为用户中断状态分量分派状态分量比特映射中的比特14;此种规范将指代那个具有注释“UINTR”的位置。系统软件使得处理器能够通过设置IA32_XSS.UINTR来管理用户中断状态分量。(这暗示了XSETBV将不允许XCR0.UINTR被置位。)

[0149] 用户中断状态分量包括具有以下布局的存储器中的48个字节:

• 字节7:0用于UIHANDLER(IA32_UINTR_HANDLER MSR)。

• 字节15:8用于UISTACKADJUST(IA32_UINTR_STACKADJUST MSR)。

• 字节23:16用于UITTSZ和UINV(来自IA32_UINTR_MISC MSR)并且用于UIF,被组织如下:

o字节19:16用于UITTSZ(IA32_UINTR_MISC MSR的比特31:0)。

o字节20用于UINV(IA32_UINTR_MISC MSR的比特39:32)。

o字节22:21(2字节)和字节23的比特6:0被保留。(如果它们未来被定义,则它们可被用于IA32_UINTR_MISC MSR情况下的比特62:40。)

o字节23的比特7用于UIF。由于字节23的比特7用于UIF(其并非IA32_UINTR_MISC MSR的部分),因此从字节23:16读取值的软件应当在尝试将该64比特值写入IA32_UINTR_MISC MSR之前清除该64比特值的比特63。

• 字节31:24用于UPIDADDR(IA32_UINTR_PD MSR)。

• 字节39:32用于UIRR(IA32_UINTR_RR MSR)。

• 字节47:40用于UITTADDR(IA32_UINTR_TT MSR,包括比特0、有效比特)。如果所有的用户中断寄存器均为0,则用户中断状态分量处于其初始状态。

[0150] 管理程序状态分量的某些部分可被标识为主启用状态(master-enable state)。XSAVES和XRSTORS以专门方式该处理该状态。UINV是用户中断状态分量的主启用状态。

[0151] XSAVE相关枚举

[0152] XSAVE特征集合包括用于枚举与每个XSAVE支持的状态分量有关的细节的体系结构。以下项目提供用户中断状态分量的XSAVE特定枚举的细节:

• CPUID.(EAX=0DH,ECX=1):EBX枚举包含当前通过XCR0|IA32_XSS启用的所有状态的XSAVE区域的按字节的大小。当IA32_XSS.UINTR[bit 14]=1时,该值将包括用户中断状态分量所要求的48字节。

• CPUID.(EAX=0DH,ECX=1):ECX.UINTR[bit 14]被枚举为1,指示用户中断状态分量是管理程序状态分量并且IA32_XSS.UINTR可以被设置为1。

• CPUID.(EAX=0DH,ECX=14):EAX被枚举为48(30H),用户中断状态分量的按字节的大小。

• CPUID.(EAX=0DH,ECX=14):EBX被枚举为0(这是针对任何管理程序状态分量的情况)。

• CPUID.(EAX=0DH,ECX=14):ECX[0]被枚举为1,指示用户中断状态分量是管理程序状态分量。

• CPUID.(EAX=0DH,ECX=14):ECX[1]被枚举为0,指示用户中断状态分量不需要在64字节边界上对齐。

• CPUID.(EAX=0DH,ECX=14):ECX[31:2]被保留并被枚举为0。

• CPUID.(EAX=0DH,ECX=14):EDX被保留并被枚举为0。

[0153] XSAVES

[0154] 由XSAVES对用户中断状态分量的管理遵循XSAVE特征集合的体系结构。以下项目标识特定于保存用户中断状态分量的点:

• XSAVES将用户中断寄存器写入用户中断状态分量。

• XSAVES将0存储到被标识为被保留的比特和字节。

• 被保存用于UIHANDLER、UPIDADDR和UITTADRR的值相对于通过CPUID.I枚举的最大线性地址宽度是正则的。

• 在保存用户中断状态分量之后,XSAVES清除UINV。

[0155] XRSTORS

[0156] 由XRSTORS对用户中断状态分量的管理遵循XSAVE特征集合的体系结构。以下项目标识特定于恢复用户中断状态分量的点：

• 在恢复用户中断状态分量之前，XRSTORS验证UINV为0。如果不是，则在加载用户中断状态分量的任何部分之前，XRSTORS导致通用保护错误(#GP)。(UINV为IA32_UINTR_MISC[39:32]；XRSTORS不检查该MSR的其余部分的内容。)

• 如果由XRSTORS使用的指令掩码和XSAVE区域指示用户中断状态分量应当从XSAVE区域被加载，则XRSTORS使用所标识的格式从XSAVE区域读取用户中断寄存器。在以下情况中的任何情况下，值读取均导致通用保护错误(#GP)：

o如果被标识为被保留的比特和字节中的任何比特和字节不为0；

o如果要被加载到UIHANDLER、UISTACKADJUST、UPIDADDR或UITTADDR中的任一者的值相对于通过CPUID枚举的最大线性地址宽度不是正则的；或者

o如果要被加载到UPIDADDR或UITTADDR的值使该寄存器中被保留的比特中的任何比特置位(保留比特为UPIDADDR的比特5:0和UITTADDR的比特3:1；针对SENDUIPI，UITTADDR的比特0是有效比特)。

• 如果在加载用户中断状态分量的任何部分之后XRSTORS引起错误或VM退出，则XRSTORS在递送错误或VM退出之前清除UINV。(用户中断状态的其他要素(包括IA32_UINTR_MISC MSR的其他部分)可保持由XRSTORS加载的值。)

• 在加载用户中断状态分量的XRSTORS的不出错的执行之后，如果UIRR的新值中的某个比特被置位并且仅在该比特被置位的情况下，逻辑处理器识别待决用户中断。

[0157] 虚拟机扩展(VIRTUAL MACHINE EXTENSION，VMX)支持

[0158] VMX体系结构支持指令集及其系统体系结构的虚拟化。某些扩展被用于支持用户中断的虚拟化。该章节描述这些扩展。

[0159] VMCS改变

[0160] VM退出控制被定义为所谓的清除UINV。向该控制分派位置27。VM进入控制被定义为所谓的加载UINV。向该控制分派位置19。

[0161] 宾客机UINV是与UINV相对应的、宾客机状态区域中新的16比特字段(编码要被确定)。用户宾客机UINV的VMCS字段编码是00000814H。

[0162] 宾客机UINV字段仅存在于支持清除UINV”VM退出控制或“加载UINV”VM进入控制的1设置的处理器上。

[0163] 普通中断的处理

[0164] 在VMX非根操作之外，具有CR4.UINTR＝IA32_EFER.LMA＝1的逻辑处理器通过执行用户中断通知标识、并且如果成功则执行用户中断处理来对普通中断进行响应。

[0165] 在VMX非根操作中，普通中断的处理取决于“外部中断退出”VM执行控制的设置：

• 如果该控制为0，则正常地发生用户中断通知标识、并且如果成功则正常地发生用户中断通知处理。

• 如果该控制为1，则逻辑处理器不执行用户中断通知标识(或用户中断通知处理)。替代地，应用先前存在的行为：发生VM退出(除非中断引起针对中断虚拟化的发布中断处理)。

[0166] 虚拟中断的处理

[0167] 如果“虚拟中断递送”VM执行控制为1,则处于VMX非根操作中的逻辑处理器可向宾客机软件递送虚拟中断。这通过使用虚拟中断的向量以从IDT选择描述符并使用该描述符来递送中断来完成。

[0168] 如果CR4.UINTR＝IA32_EFER.LMA＝1,则虚拟中断的递送被修改。具体而言,逻辑处理器首先执行某种形式的用户中断通知标识:

•代替于确认本地APIC,逻辑处理器执行虚拟中断递送的初始步骤:

V:＝RVI;

VISR[V]:＝1;

SVI:＝V;

VPPR:＝V&F0H;

VIRR[V]:＝0;

IF any bit is set in VIRR

THEN RVI:＝highest index of bit set in VIRR ELSE RVI:＝0

FI;

cease recognition of any pending virtual interrupt;(停止对任何待决虚拟中断的识别;)

(RVI、SVI、VIRR、VISR和VPPR由针对虚拟中断的体系结构来定义。)

•如果V＝UINV,则逻辑处理器继续下一步骤。否则,具有向量V的虚拟中断通过IDT正常地被递送;不应用该方法的其余部分并且不发生用户中断通知处理。

•代替于向本地APIC中的EOI寄存器写入0,逻辑处理器执行EOI虚拟化的初始步骤:

VISR[V]:＝0;

IF any bit is set in VISR

THEN SVI:＝highest index of bit set in VISR ELSE SVI:＝0;

FI;

perform PPR virtualization;(执行PPR虚拟化;)

[0169] 不同于由对EOI的宾客机写入引起的EOI虚拟化(如针对虚拟中断递送所定义),逻辑处理器不会作为此种经修改形式的用户中断通知标识的部分而对EOI退出比特映射进行检查,并且对应的VM对出不会发生。此种经修改形式的用户中断通知标识仅在虚拟中断不被掩蔽时(例如,如果RFLAGS.IF＝1)发生。

[0170] 如果此种经修改形式的用户中断通知标识完成最后动作,则逻辑处理器随后执行如上文所指定的用户中断通知处理。

[0171] 逻辑处理器在此种经修改形式的用户中断通知标识期间或在此种经修改形式的用户中断通知标识与任何后续的用户中断通知处理之间是不可中断的。

[0172] 在事务性执行期间发生的虚拟中断引起事务性执行中止以及到非事务性执行的转变。这在此种经修改形式的用户中断通知标识之前发生。

[0173] 在软件在受保护容器(例如,飞地)内执行的同时发生的虚拟中断引起异步飞地退出(asynchronous enclave exit,AEX)。此类AEX在此种经修改形式的用户中断通知标识之

前发生。

[0174] 关联至新操作的VM退出事件

[0175] 用户中断体系结构引入用户中断递送和用户中断通知处理。

[0176] 这些操作使用以下线性地址来访问存储器:用户中断递送向栈写入;用户中断通知处理在IA32_UINTR_PD MSR中的线性地址处读取和写入UPID。此类存储器访问可能引发错误(#GP、#PF等),这些错误可能引起VM退出(取决于VMCS中异常比特映射的配置)。另外,VMX非根操作中的存储器访问可能引发APIC访问VM退出、EPT违反、EPT误配置、页修改日志已满VM退出、以及SPP引起的VM退出。

[0177] 一般而言,此类VM退出正常地被处理。以下项目呈现特殊情况:

• 在用户中断递送期间发生的APIC访问VM退出、EPT违反、页修改日志已满VM退出、或SPP引起的VM退出将使退出资格的比特16设置为1,指示VM退出是“与指令执行异步的”。

• 在用户中断通知处理期间发生的任何VM退出(包括那些由于错误而导致的退出)将使IDT向量化信息字段设置成指示VM退出关联至具有向量UINV的中断(设置为值8000000xyH,其中xy=UINV)。如果在用户中断通知处理之后逻辑处理器已经进入HLT状态,则VM退出将“HLT”保存到VMCS的宾客机状态区域的活动状态字段中。

[0178] 对用户中断MSR的访问

[0179] MSR比特映射不影响作为用户中断识别、用户中断递送、用户中断通知标识、或用户中断通知处理的部分的逻辑处理器的读取或写入用户中断MSR的能力。MSR比特映射仅控制RDMSR和WRMSR指令的操作。

[0180] SENDUIPI的操作

[0181] SENDUIPI的操作以以下动作(在某些条件下被执行)结束:

```
IF local APIC is in x2APIC mode
THEN send ordinary IPI with vector tempUPID.NV to 32-bit physical APIC ID
tempUPID.NDST;
ELSE send ordinary IPI with vector tempUPID.NV to 8-bit physical APIC ID
tempUPID.NDST[15:8];
FI;
```

[0182] 在VMX非根操作之外,逻辑处理器将通过向本地APIC的中断命令寄存器(interrupt-command register,ICR)写入来发送该IPI。在VMX非根操作中,行为取决于“使用TPR影子”和“使APIC访问虚拟化”VM执行控制的设置:

• 如果“使用TPR影子”VM执行控制为0,则行为不被修改:如上文所指定,逻辑处理器通过向APIC的ICR写入来发送指定的IPI。

• 如果“使用TPR影子”VM执行控制为1且“使APIC访问虚拟化”VM执行控制为0,则逻辑处理器通过执行以下各项使x2APIC模式IPI的发送虚拟化:

o将64比特值Z写入虚拟APIC页(VICR)上的偏移300H,其中Z[7:0]=tempUPID.NV(8比特虚拟向量),Z[63:32]=tempUPID.NDST(32比特虚拟APIC ID)且Z[31:8]=000000H

(指示物理寻址的固定模式IPI)。

o以退出资格300H引起APIC写入VM退出。

APIC写入VM退出是陷阱型的:保存在VMCS的宾客机状态区域中的CS:RIP的值引用SENDUIPI之后的指令。APIC写入VM退出的基本退出原因是“APIC写入”(56)。退出资格是导致VM退出的写入访问的页偏移——在该情况下为300H。

• 如果“使用TPR影子”VM执行控制和“使APIC访问虚拟化”VM执行控制均为1,则逻辑处理器通过执行以下步骤使xAPIC模式IPI的发送虚拟化:

o将32比特值X写入虚拟APIC页(VICR_HI)上的偏移310H,其中X[31:24]=tempUPID.NDST[15:8](8比特虚拟APIC ID)且X[23:0]=000000H。

o将32比特值Y写入虚拟APIC页(VICR_LO)上的偏移300H,其中Y[7:0]=tempUPID.NV(8比特虚拟向量)且Y[31:8]=000000H(指示物理寻址的固定模式IPI)。

o以退出资格300H引起APIC写入VM退出(参见上文)。

[0183] 对VM进入的改变

[0184] 对宾客机状态区域的检查

[0185] 如果“加载UINV”VM进入控制为1,则VM进入确保宾客机UINV字段的比特15:8为0。如果该检查失败,则VM进入失败。此类失败被视为在加载宾客机状态期间或之后发生的所有VM进入失败。

[0186] 加载MSR

[0187] VM进入可从VM进入MSR加载区域加载MSR。如果VM进入加载用户中断MSR中的任何用户中断MSR,则其以与WRMSR的方式一致的方式来这样做。

[0188] 事件注入

[0189] VM进入的现有行为是使得:如果VM进入中断信息字段具有形式8000000xyH的值,则VM进入注入具有向量V=xyH的中断。这通过使用V从IDT选择描述符并使用该描述符来递送中断来完成。

[0190] 如果VMCS的宾客机状态区域的CR4字段中的比特25(UINTR)被设置为1,并且“IA-32e模式宾客机”VM进入控制为1,则VM进入在其正在注入中断的情况下被修改。具体而言,逻辑处理器首先执行某种形式的用户中断通知标识:

• 此种确认本地APIC的动作被省略。

• 如果UINV=V(其中,V是正在被注入的中断的向量),则逻辑处理器继续下一动作。否则,具有向量V的中断通过IDT正常地被递送;不应用该方法的其余部分并且不发生用户中断通知处理。

• 此种向本地APIC中的EOI寄存器写入0的动作被省略。

因为VM进入仅在中断在宾客机中未被掩蔽时才允许中断注入(例如,当RFLAGS被加载有设置比特9IF的值时),此种经修改形式的用户中断通知标识仅在虚拟中断未被掩蔽时发生。

[0191] 如果用户中断通知标识完成第二动作,则逻辑处理器随后执行用户中断通知处理。

[0192] 逻辑处理器在此种经修改形式的用户中断通知标识期间或在此种经修改形式的用户中断通知标识与任何后续的用户中断通知处理之间是不可中断的。

[0193] 只要在VMCS的宾客机状态区域的CR4字段中UINTR被设置为1并且“IA-32e模式宾客机”VM进入控制为1,VM进入事件注入的这种改变就会发生;“外部中断退出”和“虚拟中断递送”VM执行控制的设置不影响此更改。

[0194] VM进入之后的用户中断识别

[0195] VM进入在其以UIRR不等于0完成的情况下引起对待决用户中断的识别;如果其以UIRR＝0完成则没有待决用户中断被识别。

[0196] 对VM退出的改变

[0197] 记录VM退出信息

[0198] 在用户中断递送期间发生的APIC访问VM退出、EPT违反或页修改日志已满VM退出将退出资格的比特16设置为1,指示VM退出是“与指令执行异步的”。

[0199] 在用户中断通知处理期间发生的VM退出使IDT向量化信息字段设置成指示VM退出关联至具有向量UINV的中断(设置为值8000000xyH,其中xy＝UINV)。

[0200] 保存宾客机状态

[0201] 如果处理器支持用户中断,则每一次VM退出将UINV保存到VMCS中的宾客机UINV字段中(该字段的比特15:8被清除)。

[0202] 保存MSR

[0203] VM退出将MSR保存到VM退出MSR存储区域中。如果VM退出保存用户中断MSR中的任何用户中断MSR,则其以与RDMSR的方式一致的方式来这样做。

[0204] 加载主机状态

[0205] 如果“清除UINV”VM退出控制为1,则VM退出清除UINV。

[0206] 加载MSR

[0207] VM退出可从VM退出MSR加载区域加载MSR。如果VM退出加载用户中断MSR中的任何用户中断MSR,则其以与WRMSR的方式一致的方式来这样做。

[0208] VM退出之后的用户中断识别

[0209] VM退出在其以UIRR不等于0完成的情况下引起对待决用户中断的识别;如果其以UIRR＝0完成则没有待决用户中断被识别。

[0210] 实施例可包括新的VM退出控制“清除UINV”(例如,在比特位置27处)。一些支持此控制的1设置的处理器通过设置IA32_VMX_EXIT_CTLS MSR(索引483H)和IA32_VMX_TRUE_EXIT_CTLS MSR(索引48FH)中的每一者的比特59来枚举该支持。

[0211] 实施例可包括新的VM进入控制“加载UINV”(例如,在比特位置19处)。一些支持此控制的1设置的处理器通过设置IA32_VMX_ENTRY_CTLS MSR(索引484H)和IA32_VMX_TRUE_ENTRY_CTLS MSR(索引490H)中的每一者的比特51来枚举该支持。

[0212] 实施例可包括新的控制比特(例如,CR4[25],如CR4中的CR4.UINTR)。在VMX操作中支持该比特的1设置的处理器可以(例如,通过设置IA32_VMX_CR4_FIXED1 MSR(索引489H)中的比特25)枚举该支持。

示例实施例

[0213] 在实施例中,处理系统包括存储器和处理核心。存储器用于存储与正在由处理系统执行的第一应用相关联的中断控制数据结构。处理核心包括指令解码器,该指令解码器用于对第一指令解码,该第一指令由第二应用调用以向第一应用发送处理器间中断;并且

处理核心响应于经解码的指令而用于:确定处理器间中断的标识符与同第一应用相关联的通知中断向量相匹配,在中断控制数据结构中设置与处理器间中断的标识符相对应的待决中断标志,以及调用用于由中断控制数据结构标识的处理器间中断的中断处理程序。

[0214] 任何此类实施例可包括以下方面中的任何方面或以下方面的任何组合。处理器核心还用于响应于设置待决中断标志而触发指示处理器间中断的待决性的微体系结构事件。处理核心还用于响应于检测到该微体系结构事件而调用中断处理程序中断控制数据结构包括含多个待决中断比特的比特映射,其中,每个比特在比特映射内的位置与中断标识符相对应。处理系统还包括处理逻辑,该处理逻辑用于:响应于将处理器间中断标识为用户级中断而标识与第一应用相关联的发布中断描述符的地址;在发布中断描述符数据结构中设置与处理器间中断的标识符相对应的标志;以及传送具有由发布中断描述符标识的中断编号的通知中断。调用中断处理程序还包括:将指令指针的当前值存储在栈上;以及将中断处理程序的地址加载到该指令指针中。处理系统被实现为片上系统(System-on-Chip,SoC)。调用中断处理程序还包括:选择由中断控制数据结构标识的待决中断之中优先级最高的待决中断。通知中断向量由与第一应用相关联的发布中断描述符数据结构的预先定义的字段标识。

[0215] 在实施例中,一种方法包括:由正在由处理器执行的第一应用调用第一指令,以向第二应用发送处理器间中断;由处理器的指令解码器进行解码;由处理器响应于经解码的指令而确定处理器间中断的标识符与同正在由处理器执行的第二应用相关联的通知中断向量相匹配;在与第二应用相关联的中断控制数据结构中设置与处理器间中断的标识符相对应的待决中断标志;以及调用用于由中断控制数据结构标识的处理器间中断的中断处理程序。

[0216] 任何此类实施例可包括以下方面中的任何方面或以下方面的任何组合。方法还包括响应于设置所述待决中断标志而触发指示所述处理器间中断的待决性的微体系结构事件。方法还包括响应于检测到所述微体系结构事件而调用所述中断处理程序。调用用户级中断处理程序还包括:将指令指针的当前值存储在栈上;以及将中断处理程序的地址加载到该指令指针中。中断控制数据结构包括含多个待决中断比特的比特映射,其中,每个比特在比特映射内的位置与中断标识符相对应。方法还包括:响应于将处理器间中断标识为用户级中断而标识与第二应用相关联的发布中断描述符的地址;在发布中断描述符数据结构中设置与传入中断的标识符相对应的标志;以及传送具有由发布中断描述符标识的中断编号的通知中断。调用中断处理程序还包括:选择由中断控制数据结构标识的待决中断之中优先级最高的待决中断。通知中断向量由与第二应用相关联的发布中断描述符数据结构的预先定义的字段标识。

[0217] 在实施例中,一种设备可包括用于执行本文中公开的任何功能的装置。在实施例中,一种装置可以包括数据存储设备,该数据存储设备存储代码,该代码当由硬件处理器执行时使该硬件处理器执行本文中公开的任何方法。在实施例中,一种装置可如在具体实施方式中所描述。在实施例中,一种方法可如在具体实施方式中所描述。在实施例中,一种非暂态机器可读介质可以存储指令,这些指令在由机器执行时使该机器执行包括本文中公开的任何方法的方法。实施例可以包括本说明书中描述的任何细节、特征等或细节、特征等的组合。

示例计算机体系结构

[0218] 下面详细描述示例计算机体系结构。本领域中已知的用于膝上型电脑、桌面型电脑、手持PC、个人数字助理、工程工作站、服务器、网络设备、网络集线器、交换机、嵌入式处理器、数字信号处理器(digital signal processor,DSP)、图形设备、视频游戏设备、机顶盒、微控制器、蜂窝电话、便携式媒体播放器、手持设备、以及大量各种其他电子设备的其他系统设计和配置也是适当的。总之,能够包含本文公开的处理器和/或其他执行逻辑的各种系统或电子设备一般都是适当的。

[0219] 图11图示了示例系统的实施例。多处理器系统1100是一种点对点互连系统,并包括多个处理器,其中包括经由点到点互连1150耦合的第一处理器1170和第二处理器1180。在一些实施例中,第一处理器1170和第二处理器1180是同构的。在一些实施例中,第一处理器1170和第二处理器1180是异构的。

[0220] 处理器1170和1180被示为分别包括集成存储器控制器(integrated memory controller,IMC)单元电路1172和1182。处理器1170还包括作为其互连控制器单元的部分的点到点(point-to-point,P-P)接口1176和1178;类似地,第二处理器1180包括P-P接口1186和1188。处理器1170、1180可以利用点到点(P-P)接口电路1178、1188经由P-P互连1150交换信息。IMC 1172和1182将处理器1170、1180耦合到各自的存储器,即存储器1132和存储器1134,这些存储器可以是在本地附接到各个处理器的主存储器的一部分。

[0221] 处理器1170、1180可以各自利用点到点接口电路1176、1194、1186、1198,经由各个P-P互连1152、1154来与芯片组1190交换信息。芯片组1190可以可选地经由高性能接口1192与协处理器1138交换信息。在一些实施例中,协处理器1138是专用处理器,例如高吞吐量MIC处理器、网络或通信处理器、压缩引擎、图形处理器、GPGPU、嵌入式处理器,等等。

[0222] 共享缓存(未示出)可以被包括在任一处理器1170、1180中,或者在两个处理器之外但经由P-P互连与这些处理器相连接,从而使得:如果一处理器被置于低功率模式,任一个或两个处理器的本地缓存信息也可以被存储在该共享缓存中。

[0223] 芯片组1190可以经由接口1196耦合到第一互连1116。在一些实施例中,第一互连1116可以是外围组件互连(Peripheral Component Interconnect,PCI)互连,或者是诸如PCI快速(PCI Express)互连或者另一I/O互连之类的互连。在一些实施例中,这些互连中的一个耦合到功率控制单元(power control unit,PCU)1117,PCU 1117可以包括电路、软件和/或固件来执行关于处理器1170、1180和/或协处理器1138的功率管理操作。PCU 1117向电压调节器提供控制信息,以使得电压调节器生成适当的调节电压。PCU 1117还提供控制信息以控制所生成的操作电压。在各实施例中,PCU 1117可以包括各种功率管理逻辑单元(电路)来执行基于硬件的功率管理。这种功率管理可以是完全由处理器控制的(例如,由各种处理器硬件控制,并且可以由工作负载和/或功率约束、热约束或其他处理器约束所触发),和/或功率管理可以响应于外部源而被执行(例如平台或功率管理源或系统软件)。

[0224] PCU 1117被图示为作为与处理器1170和/或处理器1180分开的逻辑而存在。在其他情况下,PCU 1117可以在处理器1170或1180的核心(未示出)中给定的一个或多个核心上执行。在一些情况下,PCU 1117可以被实现为微控制器(专用的或通用的)或者其他控制逻辑,其被配置为执行其自己的专用功率管理代码(有时称为P代码)。在另外的其他实施例中,PCU 1117要执行的功率管理操作可以被实现在处理器外部,例如借由单独的功率管理

集成电路(power management integrated circuit,PMIC)或者在处理器外部的另一组件来实现。在另外的其他实施例中,PCU 1117要执行的功率管理操作可被实现在BIOS或其他系统软件内。

[0225] 各种I/O设备1114以及互连(总线)桥1118可以耦合到第一互连1116,互连(总线)桥1118将第一互连1116耦合到第二互连1120。在一些实施例中,一个或多个额外的处理器1115被耦合到第一互连1116,例如协处理器、高吞吐量MIC处理器、GPGPU、加速器(诸如例如图形加速器或数字信号处理(digital signal processing,DSP)单元)、现场可编程门阵列(field programmable gate array,FPGA)、或者任何其他处理器。在一些实施例中,第二互连1120可以是低引脚数(low pin count,LPC)互连。各种设备可以耦合到第二互连1120,这些设备例如包括键盘和/或鼠标1122、通信设备1127、以及存储单元电路1128。存储单元电路1128可以是盘驱动器或者其他大容量存储设备,其在一些实施例中可包括指令/代码和数据1130。另外,音频I/O 1124可以耦合到第二互连1120。注意,除了上述的点到点体系结构以外其他体系结构也是可能的。例如,诸如多处理器系统1100之类的系统可以实现多点分支(multi-drop)互连或者其他这种体系结构,而不是点到点体系结构。

示例核心体系结构、处理器和计算机体系结构

[0226] 处理器核心可以按不同的方式、为了不同的目的、在不同的处理器中实现。例如,这些核心的实现方式可以包括:1)通用有序核心,针对通用计算目的;2)高性能通用乱序核心,针对通用计算目的;3)专用核心,主要针对图形和/或科学(吞吐量)计算目的。不同处理器的实现方式可以包括:1)CPU,包括针对通用计算目的的一个或多个通用有序核心和/或针对通用计算目的的一个或多个通用乱序核心;以及2)协处理器,包括主要针对图形和/或科学(吞吐量)目的的一个或多个专用核心。这些不同处理器导致不同的计算机系统体系结构,这些体系结构可以包括:1)协处理器与CPU在分开的芯片上;2)协处理器与CPU在同一封装中分开的管芯上;3)协处理器与CPU在同一管芯上(在此情况下,这种协处理器有时被称为专用逻辑,例如集成图形和/或科学(吞吐量)逻辑,或者被称为专用核心);以及4)片上系统,其可以在与所描述的CPU(有时称为(一个或多个)应用核心或者(一个或多个)应用处理器)同一管芯上包括上述的协处理器以及另外的功能。接下来描述示例核心体系结构,然后是对示例处理器和计算机体系结构的描述。

[0227] 图12图示了示例处理器1200的实施例的框图,该处理器1200可以具有多于一个的核心,可以具有集成存储器控制器,并且可以具有集成图形器件。实线框图示的处理器1200具有单个核心1202(A)、系统代理1210和一组一个或多个互连控制器单元电路1216,而可选地添加的虚线框将替代性处理器1200图示为具有多个核心1202(A)-(N)、系统代理单元电路1210中的一组一个或多个集成存储器控制单元电路1214、专用逻辑1208以及一组一个或多个互连控制器单元电路1216。注意,处理器1200可以是图11的处理器1170或1180或者协处理器1138或1115之一。

[0228] 从而,处理器1200的不同实现方式可以包括:1)CPU,其中专用逻辑1208是集成图形和/或科学(吞吐量)逻辑(可以包括一个或多个核心,未示出),核心1202(A)-(N)是一个或多个通用核心(例如,通用有序核心、通用乱序核心、或这两者的组合);2)协处理器,其中核心1202(A)-(N)是主要针对图形和/或科学(吞吐量)目的的大量专用核心;以及3)协处理器,其中核心1202(A)-(N)是大量的通用有序核心。从而,处理器1200可以是通用处理器、协

处理器或者专用处理器，例如网络或通信处理器、压缩引擎、图形处理器、GPGPU(通用图形处理单元电路)、高吞吐量集成众核(many integrated core，MIC)协处理器(包括30个或更多个核心)、嵌入式处理器，等等。该处理器可被实现在一个或多个芯片上。处理器1200可以是一个或多个衬底的一部分和/或可以使用多种工艺技术中的任何技术来实现在一个或多个衬底上，这些工艺技术例如BiCMOS、CMOS、或者NMOS。

[0229] 存储器层次体系包括核心1202(A)-(N)内的一级或多级缓存单元电路1204(A)-(N)、一组一个或多个共享缓存单元电路1206、以及耦合到该组集成存储器控制器单元电路1214的外部存储器(未示出)。该组一个或多个共享缓存单元电路1206可以包括一个或多个中间级别缓存，例如第2级(L2)、第3级(L3)、第4级(4)或者其他级别的缓存，例如最后一级缓存(last level cache，LLC)，和/或这些的组合。虽然在一些实施例中基于环的互连网络电路1212对专用逻辑1208(例如，集成图形逻辑)、该组共享缓存单元电路1206和系统代理单元电路1210进行互连，但替代性实施例使用任何数目的公知技术来对这些单元进行互连。在一些实施例中，在共享缓存单元电路1206中的一个或多个电路与核心1202(A)-(N)之间维持一致性。

[0230] 在一些实施例中，核心1202(A)-(N)中的一个或多个具有多线程能力。系统代理单元电路1210包括对核心1202(A)-(N)进行协调和操作的那些组件。系统代理单元电路1210可以包括例如功率控制单元(power control unit，PCU)电路和/或显示单元电路(未示出)。PCU可以是(或者可以包括)对核心1202(A)-(N)和/或专用逻辑1208(例如，集成图形逻辑)的功率状态进行调节所需的逻辑和组件。显示单元电路用于驱动一个或多个在外部连接的显示器。

[0231] 核心1202(A)-(N)就体系结构指令集而言可以是同构的或者异构的；也就是说，核心1202(A)-(N)中的两个或更多个核心可能能够执行同一指令集，而其他核心可能能够只执行该指令集的子集或者能够执行一不同的ISA。

示例核心体系结构

有序和乱序核心框图

[0232] 图13(A)的框图图示了根据本发明的实施例的示例有序管线和示例寄存器重命名、乱序发出/执行管线两者。图13(B)的框图图示了根据本发明的实施例，要被包括在处理器中的有序体系结构核心的示例实施例和示例寄存器重命名、乱序发出/执行体系结构核心两者。图13(A)-(B)中的实线框图示了有序管线和有序核心，而可选地添加的虚线框图示了寄存器重命名、乱序发出/执行管线和核心。考虑到有序方面是乱序方面的子集，将描述乱序方面。

[0233] 在图13(A)中，处理器管线1300包括取得阶段1302、可选的长度解码阶段1304、解码阶段1306、可选的分配阶段1308、可选的重命名阶段1310、调度(也称为调遣或发出)阶段1312、可选的寄存器读取/存储器读取阶段1314、执行阶段1316、写回/存储器写入阶段1318、可选的异常处置阶段1322、以及可选的提交阶段1324。在这些处理器管线阶段的每一者中可以执行一个或多个操作。例如，在取得阶段1302期间，从指令存储器取得一个或多个指令，在解码阶段1306期间，可以对取得的一个或多个指令进行解码，可以生成使用转发寄存器端口的地址(例如，加载存储单元(load store unit，LSU)地址)，并且可以执行分支转发(例如，立即数偏移或者链接寄存器(link register，LR))。在一种实施例中，解码阶段

1306和寄存器读取/存储器读取阶段1314可以被组合到一个管线阶段中。在一种实施例中，在执行阶段1316期间，可以执行经解码的指令，可以执行到高级微控制器总线(Advanced Microcontroller Bus，AMB)接口的LSU地址/数据管线化，可以执行乘法和加法操作，可以执行具有分支结果的算术操作，等等。

[0234] 作为示例，示例寄存器重命名、乱序发出/执行核心体系结构可以通过以下方式实现管线1300：1)指令取得1338执行取得和长度解码阶段1302和1304；2)解码单元电路1340执行解码阶段1306；3)重命名/分配器单元电路1352执行分配阶段1308和重命名阶段1310；4)(一个或多个)调度器单元电路1356执行调度阶段1312；5)(一个或多个)物理寄存器堆单元电路1358和存储器单元电路1370执行寄存器读取/存储器读取阶段1314；执行集群1360执行执行阶段1316；6)存储器单元电路1370和(一个或多个)物理寄存器堆单元电路1358执行写回/存储器写入阶段1318；7)在异常处置阶段1322中可能涉及各种单元(单元电路)；并且8)引退单元电路1354和(一个或多个)物理寄存器堆单元电路1358执行提交阶段1324。

[0235] 图13(B)示出了处理器核心1390包括耦合到执行引擎单元电路1350的前端单元电路1330，并且两者都耦合到存储器单元电路1370。核心1390可以是精简指令集计算(reduced instruction set computing，RISC)核心、复杂指令集计算(complex instruction set computing，CISC)核心、超长指令字(very long instruction word，VLIW)核心、或者混合或替代性核心类型。作为另外一个选项，核心1390可以是专用核心，例如网络或通信核心、压缩引擎、协处理器核心、通用计算图形处理单元(general purpose computing graphics processing unit，GPGPU)核心、图形核心，等等。

[0236] 前端单元电路1330可以包括分支预测单元电路1332，该分支预测单元电路耦合到指令缓存单元电路1334，该指令缓存单元电路耦合到指令转译后备缓冲器(translation lookaside buffer，TLB)1336，该指令TLB耦合到指令取得单元电路1338，该指令取得单元电路耦合到解码单元电路1340。在一种实施例中，指令缓存单元电路1334被包括在存储器单元电路1370中，而不是前端单元电路1330中。解码单元电路1340(或解码器)可以对指令解码，并且生成一个或多个微操作、微代码入口点、微指令、其他指令或其他控制信号作为输出，这些微操作、微代码入口点、微指令、其他指令或其他控制信号是从原始指令解码来的，或者以其他方式反映原始指令，或者是从原始指令得出的。解码单元电路1340还可以包括地址生成单元电路(address generation unit，AGU，未示出)。在一种实施例中，AGU使用经转发的寄存器端口来生成LSU地址，并且可以进一步执行分支转发(例如，立即数偏移分支转发，LR寄存器分支转发，等等)。可以利用各种不同的机制来实现解码单元电路1340。适当机制的示例包括但不限于查找表、硬件实现方式、可编程逻辑阵列(programmable logic array，PLA)、微代码只读存储器(read only memory，ROM)，等等。在一种实施例中，核心1390包括微代码ROM(未示出)或其他介质，其存储用于某些宏指令的微代码(例如，在解码单元电路1340中或者以其他方式在前端单元电路1330内)。在一种实施例中，解码单元电路1340包括微操作(micro-op)或操作缓存(未示出)，以保存/缓存在处理器管线1300的解码或其他阶段期间生成的经解码的操作、微标记或微操作。解码单元电路1340可以耦合到执行引擎单元电路1350中的重命名/分配器单元电路1352。

[0237] 执行引擎电路1350包括重命名/分配器单元电路1352，其耦合到引退单元电路1354和一组一个或多个调度器电路1356。调度器电路1356代表任意数目个不同调度器，包

括预留站、中央指令窗口，等等。在一些实施例中，(一个或多个)调度器电路1356可以包括算术逻辑单元(arithmetic logic unit，ALU)调度器/调度电路、ALU队列、地址生成单元(address generation unit，AGU)调度器/调度电路、AGU队列，等等。(一个或多个)调度器电路1356耦合到(一个或多个)物理寄存器堆电路1358。(一个或多个)物理寄存器堆电路1358的每一者代表一个或多个物理寄存器堆，这些物理寄存器堆中的不同物理寄存器堆存储一个或多个不同的数据类型，例如标量整数、标量浮点、紧缩整数、紧缩浮点、向量整数、向量浮点、状态(例如，指令指针，即要执行的下一指令的地址)，等等。在一种实施例中，(一个或多个)物理寄存器堆单元电路1358包括向量寄存器单元电路、写入掩码寄存器单元电路、以及标量寄存器单元电路。这些寄存器单元可以提供体系结构式向量寄存器、向量掩码寄存器、通用寄存器，等等。(一个或多个)物理寄存器单元堆电路1358被引退单元电路1354(也称为引退队列)重叠，以展示可以用来实现寄存器重命名和乱序执行的各种方式(例如，利用(一个或多个)重排序缓冲器(reorder buffer，ROB)和(一个或多个)引退寄存器堆；利用(一个或多个)未来的堆、(一个或多个)历史缓冲器、以及(一个或多个)引退寄存器堆；利用寄存器图谱和寄存器的池；等等)。引退单元电路1354和(一个或多个)物理寄存器堆电路1358耦合到(一个或多个)执行集群1360。(一个或多个)执行集群1360包括一组一个或多个执行单元电路1362和一组一个或多个存储器访问电路1364。执行单元电路1362可以对各种类型的数据(例如，标量浮点、紧缩整数、紧缩浮点、向量整数、向量浮点)执行各种算术、逻辑、浮点或其他类型的操作(例如，移位、加法、减法、乘法)。虽然一些实施例可以包括专用于特定功能或功能集合的若干个执行单元或执行单元电路，但其他实施例可以只包括一个执行单元电路或者全部执行所有功能的多个执行单元/执行单元电路。(一个或多个)调度器电路1356、(一个或多个)物理寄存器堆单元电路1358以及(一个或多个)执行集群1360被示为可能是多个，因为某些实施例为某些类型的数据/操作创建单独的管线(例如，标量整数管线、标量浮点/紧缩整数/紧缩浮点/向量整数/向量浮点管线和/或存储器访问管线，它们各自具有其自己的调度器电路、(一个或多个)物理寄存器堆单元电路和/或执行集群——并且在单独的存储器访问管线的情况下，所实现的某些实施例中只有该管线的执行集群具有(一个或多个)存储器访问单元电路1364)。还应当理解，在使用分开的管线的情况下，这些管线中的一个或多个可以是乱序发出/执行，而其余是有序的。

[0238] 在一些实施例中，执行引擎单元电路1350可以执行到高级微控制器总线(AMB)接口(未示出)的加载存储单元(LSU)地址/数据管线化，以及地址阶段和写回、数据阶段加载、存储、以及分支。

[0239] 一组存储器访问电路1364耦合到存储器单元电路1370，该存储器单元电路包括数据TLB单元电路1372，该数据TLB电路耦合到数据缓存电路1374，该数据缓存电路耦合到第2级(L2)缓存电路1376。在一个示例实施例中，存储器访问单元电路1364可以包括加载单元电路、存储地址单元电路、以及存储数据单元电路，它们中的每一者耦合到存储器单元电路1370中的数据TLB电路1372。指令缓存电路1334进一步耦合到存储器单元电路1370中的第2级(L2)缓存单元电路1376。在一种实施例中，指令缓存1334和数据缓存1374被组合成L2缓存单元电路1376、第3级(L3)缓存单元电路(未示出)和/或主存储器中的单个指令和数据缓存(未示出)。L2缓存单元电路1376耦合到一个或多个其他级别的缓存并且最终耦合到主存储器。

[0240] 核心1390可以支持一个或多个指令集(例如,x86指令集(带有已随着更新版本添加的一些扩展);MIPS指令集;ARM指令集(带有可选的额外扩展,例如NEON)),该指令集包括本文描述的(一个或多个)指令。在一种实施例中,核心1390包括支持紧缩数据指令集扩展(例如,AVX1、AVX2)的逻辑,从而允许利用紧缩数据来执行被许多多媒体应用使用的操作。

(一个或多个)示例执行单元电路

[0241] 图14图示了(一个或多个)执行单元电路的实施例,例如图13(B)的(一个或多个)执行单元电路1362。如图所示,(一个或多个)执行单元电路1362可以包括一个或多个ALU电路1401、向量/SIMD单元电路1403、加载/存储单元电路1405、和/或分支/跳转单元电路1407。ALU电路1401执行整数算术和/或布尔操作。向量/SIMD单元电路1403对紧缩数据(例如SIMD/向量寄存器)执行向量/SIMD操作。加载/存储单元电路1405执行加载和存储指令,以将数据从存储器加载到寄存器中,或者从寄存器存储到存储器。加载/存储单元电路1405也可以生成地址。分支/跳转单元电路1407取决于指令而引起到某个存储器地址的分支或跳转。浮点单元(floating-point unit,FPU)电路1409执行浮点算术。(一个或多个)执行单元电路1362的宽度取决于实施例而有所不同,并且可以在从16比特到1024比特的范围中。在一些实施例中,两个或更多个更小的执行单元被逻辑地组合以形成更大的执行单元(例如,两个128比特执行单元被逻辑组合以形成256比特执行单元)。

示例寄存器体系结构

[0242] 图15是根据一些实施例的寄存器体系结构1500的框图。如图所示,存在向量/SIMD寄存器1510,其宽度从128比特到1024比特不等。在一些实施例中,向量/SIMD寄存器1510在物理上是512比特的,并且取决于映射,只有一些低位比特被使用。例如,在一些实施例中,向量/SIMD寄存器1510是512比特的ZMM寄存器:低位256比特被用于YMM寄存器,并且低位128比特被用于XMM寄存器。因此,存在寄存器的覆盖。在一些实施例中,向量长度字段在最大长度和一个或多个其他更短的长度之间作出选择,其中每个这种更短长度是前一长度的一半长度。标量操作是对ZMM/YMM/XMM寄存器中的最低阶数据元素位置执行的操作;更高阶数据元素位置或者被保持为与其在该指令之前相同,或者被归零,这取决于实施例。

[0243] 在一些实施例中,寄存器体系结构1500包括写入掩码/谓词(predicate)寄存器1515。例如,在一些实施例中,有8个写入掩码/谓词寄存器(有时称为k0至k7),它们各自的大小是16比特、32比特、64比特或128比特。写入掩码/谓词寄存器1515可以允许合并(例如,允许目的地中的任何元素集合在任何操作的执行期间被保护免于更新)和/或归零(例如,归零向量掩码允许目的地中的任何元素集合在任何操作的执行期间被归零)。在一些实施例中,给定的写入掩码/谓词寄存器1515中的每个数据元素位置对应于目的地的数据元素位置。在其他实施例中,写入掩码/谓词寄存器1515是可缩放的,并且由针对给定向量元素的设定数目个使能比特组成(例如,给每个64比特向量元素8个使能比特)。

[0244] 寄存器体系结构1500包括多个通用寄存器1525。这些寄存器可以是16比特、32比特、64比特等等,并能够被用于标量操作。在一些实施例中,这些寄存器用名称RAX、RBX、RCX、RDX、RBP、RSI、RDI、RSP以及R8至R15来称呼。

[0245] 在一些实施例中,寄存器体系结构1500包括标量浮点寄存器1545,它被用于使用x87指令集扩展对32/64/80比特浮点数据进行标量浮点操作,或者作为MMX寄存器,来对64比特紧缩整数数据执行操作,以及针对在MMX和XMM寄存器之间执行的一些操作保存操作对

象。

[0246] 一个或多个标志寄存器1540(例如,EFLAGS,RFLAGS,等等)存储状态和控制信息,用于算术、比较和系统操作。例如,一个或多个标志寄存器1540可以存储条件代码信息,例如进位、奇偶性、辅助进位、零、符号、以及溢出。在一些实施例中,一个或多个标志寄存器1540被称为程序状态和控制寄存器。

[0247] 段寄存器1520包含用于访问存储器的段点。在一些实施例中,这些寄存器由名称CS、DS、SS、ES、FS和GS来称呼。

[0248] 机器特定寄存器(machine-specific register,MSR)1535对处理器性能进行控制和报告。大多数MSR 1535处置与系统有关的功能,并且是应用程序不可访问的。机器检查寄存器1560由用于对硬件错误进行检测和报告的控制、状态和错误报告MSR组成。

[0249] 一个或多个指令指针寄存器1530存储指令指针值。(一个或多个)控制寄存器1555(例如,CR0-CR4)确定处理器(例如,处理器1170、1180、1138、1118和/或1200)的操作模式和当前在执行的任务的特性。调试寄存器1550控制并允许监视处理器或核心的调试操作。

[0250] 存储器管理寄存器1565指定用于保护模式存储器管理中的数据结构的位置。这些寄存器可以包括GDTR、IDTR、任务寄存器、以及LDTR寄存器。

[0251] 本发明的替代性实施例可以使用更宽或更窄的寄存器。此外,本发明的替代性实施例可以使用更多、更少或不同的寄存器堆和寄存器。

指令集

[0252] 指令集体系结构(instruction set architecture,ISA)可以包括一个或多个指令格式。给定的指令格式可以定义各种字段(例如,比特的数目、比特的位置)来指定要执行的操作(例如,操作码)和要被执行该操作的(一个或多个)操作对象和/或其他(一个或多个)数据字段(例如,掩码),等等。一些指令格式通过指令模板(或子格式)的定义被进一步分解。例如,给定的指令格式的指令模板可以被定义为具有该指令格式的字段的不同子集(所包括的字段通常是按相同顺序的,但至少一些具有不同的比特位置,因为所包括的字段更少)和/或被定义为具有以不同方式解读的给定字段。从而,ISA的每个指令是利用给定的指令格式来表达的(并且如果定义了的话,则以该指令格式的指令模板中的一个给定指令模板来表达),并且包括用于指定操作和操作对象的字段。例如,示例ADD指令具有特定的操作码和指令格式,该指令格式包括操作码字段来指定该操作码,并且包括操作对象字段来选择操作对象(源1/目的地和源2);并且这个ADD指令在指令流中的出现将在选择特定操作对象的操作对象字段中具有特定内容。

示例指令格式

[0253] 可以按不同的格式来实现本文描述的(一个或多个)指令的实施例。此外,下文详述了示例系统、体系结构和管线。(一个或多个)指令的实施例可以在这些系统、体系结构和管线上被执行,但不限于详述的那些。

[0254] 图16图示了指令格式的实施例。如图所示,指令可以包括多个成分,这些成分包括但不限于用于以下项的一个或多个字段:一个或多个前缀1601、操作码1603、寻址信息1605(例如,寄存器标识符、存储器寻址信息,等等)、位移值1607、和/或立即数1609。注意,一些指令利用了该格式的一些或所有字段,而其他指令可能只使用操作码1603的字段。在一些实施例中,图示的顺序是这些字段要被编码的顺序,然而应当明白,在其他实施例中,这些

字段可以被按另外的顺序来编码、被组合，等等。

[0255] (一个或多个)前缀字段1601在被使用时对指令进行修改。在一些实施例中，一个或多个前缀被用于重复字符串指令(例如，0xF0、0xF2、0xF3，等等)、提供节超控(section override)(例如，0x2E、0x36、0x3E、0x26、0x64、0x65、0x2E、0x3E，等等)、执行总线锁定操作、和/或改变操作对象(例如，0x66)和地址大小(例如，0x67)。某些指令要求强制性的前缀(例如，0x66、0xF2、0xF3，等等)。这些前缀中的某些可以被认为是“传统”前缀。其他前缀(本文详述了其一个或多个示例)指示出和/或提供了进一步的能力，例如指定特定的寄存器等等。这些其他前缀通常跟随在“传统”前缀之后。

[0256] 操作码字段1603被用于至少部分地定义在指令的解码时要执行的操作。在一些实施例中，在操作码字段1603中编码的主操作码的长度为1个、2个或3个字节。在其他实施例中，主操作码可以是其他的长度。额外的3比特操作码字段有时被编码在另一个字段中。

[0257] 寻址字段1605被用于对指令的一个或多个操作对象进行寻址，例如存储器中的位置或者一个或多个寄存器。图17图示了寻址字段1605的实施例。在这个图示中，示出了可选的MOD R/M字节1702和可选的缩放、索引、基址(Scale,Index,Base,SIB)字节1704。MOD R/M字节1702和SIB字节1704被用来编码指令的最多两个操作对象，每个操作对象是直接寄存器或有效存储器地址。注意，这些字段中的每一者是可选的，即，不是所有的指令都包括这些字段中的一个或多个。MOD R/M字节1702包括MOD字段1742、寄存器字段1744、以及R/M字段1746。

[0258] MOD字段1742的内容区分存储器访问和非存储器访问模式。在一些实施例中，当MOD字段1742具有值b11时，利用寄存器直接寻址模式，否则使用寄存器间接寻址。

[0259] 寄存器字段1744可以编码目的地寄存器操作对象或者源寄存器操作对象，或者也可以编码操作码扩展而不被用于编码任何指令操作对象。寄存器索引字段1744的内容直接指定或者通过地址生成来指定源或目的地操作对象的位置(在寄存器中或者在存储器中)。在一些实施例中，寄存器字段1744得到来自前缀(例如，前缀1601)的额外比特的补充，以允许更大的寻址。

[0260] R/M字段1746可以用来编码引用了存储器地址的指令操作对象，或者可以用来编码目的地寄存器操作对象或源寄存器操作对象。注意，在一些实施例中，R/M字段1746可以被与MOD字段1742相组合以规定寻址模式。

[0261] SIB字节1704包括缩放字段1752、索引字段1754、以及基址字段1756，以用于地址的生成。缩放字段1752指示缩放因子。索引字段1754指定要使用的索引寄存器。在一些实施例中，索引字段1754得到来自前缀(例如，前缀1601)的额外比特的补充，以允许更大的寻址。基址字段1756指定了要使用的基址寄存器。在一些实施例中，基址字段1756得到来自前缀(例如，前缀1601)的额外比特的补充，以允许更大的寻址。在实践中，缩放字段1752的内容允许缩放索引字段1754的内容以进行存储器地址生成(例如，对于使用$2^{缩放}$*索引+基址的地址生成)。

[0262] 一些寻址形式利用位移值来生成存储器地址。例如，可以根据$2^{缩放}$*索引+基址+位移、索引*缩放+位移、r/m+位移、指令指针(RIP/EIP)+位移、寄存器+位移等等来生成存储器地址。该位移可以是1字节、2字节、4字节等等的值。在一些实施例中，位移字段1607提供这个值。此外，在一些实施例中，位移因子的使用被编码在寻址字段1605的MOD字段中，它指示

出压缩位移方案，对于该方案，位移值通过disp8与基于向量长度确定的缩放因数N、b比特的值、以及指令的输入元素大小相结合地相乘来计算。位移值被存储在位移字段1607中。

[0263] 在一些实施例中，立即数字段1609为指令指定立即数。立即数可以被编码为1字节值、2字节值、4字节值等等。

[0264] 图18图示了第一前缀1601(A)的实施例。在一些实施例中，第一前缀1601(A)是REX前缀的实施例。使用这个前缀的指令可以指定通用寄存器、64比特紧缩数据寄存器(例如，单指令多数据(SIMD)寄存器或向量寄存器)、和/或控制寄存器和调试寄存器(例如，CR8-CR15和DR8-DR15)。

[0265] 使用第一前缀1601(A)的指令可以使用3比特字段指定最多三个寄存器，这取决于格式：1)使用MOD R/M字节1702的reg字段1744和R/M字段1746；2)使用MOD R/M字节1702与SIB字节1704，包括使用reg字段1744以及基址字段1756和索引字段1754；或者3)使用操作码的寄存器字段。

[0266] 在第一前缀1601(A)中，比特位置7:4被设置为0100。比特位置3(W)可以被用于确定操作对象大小，但不能单独确定操作对象宽度。因此，当W＝0时，操作对象大小由代码段描述符(code segment descriptor，CS.D)决定，而当W＝1时，操作对象大小为64比特。

[0267] 注意，添加另一个比特允许对16(2^4)个寄存器进行寻址，而单独的MOD R/M reg字段1744和MOD R/M的R/M字段1746各自只能寻址8个寄存器。

[0268] 在第一前缀1601(A)中，比特位置2(R)可以是MOD R/M的reg字段1744的扩展，并且当该字段编码了通用寄存器、64比特紧缩数据寄存器(例如，SSE寄存器)或者控制或调试寄存器时，可被用来修改MOD R/M的reg字段1744。当MOD R/M字节1702指定其他寄存器或定义扩展操作码时，R被忽略。

[0269] 比特位置1(X)X比特可以修改SIB字节索引字段1754。

[0270] 比特位置B(B)B可以修改MOD R/M的R/M字段1746或SIB字节基址字段1756中的基址；或者它可以修改用于访问通用寄存器(例如，通用寄存器1525)的操作码寄存器字段。

[0271] 图19(A)-(D)图示了如何使用第一前缀1601(A)的R、X和B字段的实施例。图19(A)图示了当SIB字节1704不被用于存储器寻址时，来自第一前缀1601(A)的R和B被用来扩展MOD R/M字节1702的reg字段1744和R/M字段1746。图19(B)图示了当不使用SIB字节17 04时，来自第一前缀1601(A)的R和B被用来扩展MOD R/M字节1702的reg字段1744和R/M字段1746(寄存器-寄存器寻址)。图19(C)图示了当SIB字节17 04被用于存储器寻址时，来自第一前缀1601(A)的R、X和B被用于扩展MOD R/M字节1702的reg字段1744以及索引字段1754和基址字段1756。图19(D)图示了当寄存器被编码在操作码1603中时，来自第一前缀1601(A)的B被用来扩展MOD R/M字节1702的reg字段1744。

[0272] 图20(A)-(B)图示了第二前缀1601(B)的实施例。在一些实施例中，第二前缀1601(B)是VEX前缀的实施例。第二前缀1601(B)编码允许指令具有多于两个操作对象，并且允许SIMD向量寄存器(例如，向量/SIMD寄存器1510)长于64比特(例如，128比特和256比特)。第二前缀1601(B)的使用提供了三操作对象(或更多)的语法。例如，先前的两操作对象指令执行诸如A＝A+B之类的操作，这覆写了源操作对象。对第二前缀1601(B)的使用使得操作对象能够执行非破坏性操作，比如A＝B+C。

[0273] 在一些实施例中，第二前缀1601(B)有两种形式——两字节形式和三字节形式。两

字节的第二前缀1601(B)主要被用于128比特、标量和一些256比特指令;而三字节的第二前缀1601(B)提供了3字节操作码指令和第一前缀1601(A)的紧凑替换。

[0274] 图20(A)图示了第二前缀1601(B)的两字节形式的实施例。在一个示例中,格式字段2001(字节0 2003)包含值C5H。在一个示例中,字节1 2005在比特[7]中包括"R"值。这个值是第一前缀1601(A)的同一值的补码。比特[2]被用来规定向量的长度(L)(其中0的值是标量或者128比特向量,而1的值是256比特向量)。比特[1:0]提供等同于一些传统前缀的操作码扩展性(例如,00=无前缀,01=66H,10=F3H,并且11=F2H)。被示为vvvv的比特[6:3]可以被用于:1)编码第一源寄存器操作对象,该操作对象以反转(反码)形式指定,并且对于具有2个或更多个源操作对象的指令有效;2)编码目的地寄存器操作对象,该操作对象以反码形式指定,用于某些向量移位;或者3)不编码任何操作对象,该字段被保留并且应当包含某个值,例如1111b。

[0275] 使用这个前缀的指令可以使用MOD R/M的R/M字段1746,以编码引用存储器地址的指令操作对象,或者编码目的地寄存器操作对象或源寄存器操作对象。

[0276] 使用这个前缀的指令可以使用MOD R/M的reg字段1744,以编码目的地寄存器操作对象或者源寄存器操作对象,被作为操作码扩展来对待而不被用于编码任何指令操作对象。

[0277] 对于支持四个操作对象的指令语法,vvvv、MOD R/M的R/M字段1746和MOD R/M的reg字段1744编码了四个操作对象中的三个。然后立即数1609的比特[7:4]被用来编码第三源寄存器操作对象。

[0278] 图20(B)图示了第二前缀1601(B)的三字节形式的实施例。在一种示例中,格式字段2011(字节0 2013)包含值C4H。字节1 2015在比特[7:5]中包括"R"、"X"和"B",它们是第一前缀1601(A)的这些值的补码。字节1 2015的比特[4:0](示为mmmmm)包括根据需要来对一个或多个隐含的前导操作码字节进行编码的内容。例如,00001意味着0FH前导操作码,00010意味着0F38H前导操作码,00011意味着0F3AH前导操作码,等等。

[0279] 字节2 2017的比特[7]的使用与第一前缀1601(A)的W类似,包括帮助确定可提升的操作对象大小。比特[2]被用来规定向量的长度(L)(其中0的值是标量或者128比特向量,而1的值是256比特向量)。比特[1:0]提供等同于一些传统前缀的操作码扩展性(例如,00=无前缀,01=66H,10=F3H,并且11=F2H)。被示为vvvv的比特[6:3]可以被用于:1)编码第一源寄存器操作对象,该操作对象以反转(反码)形式指定,对于具有2个或更多个源操作对象的指令有效;2)编码目的地寄存器操作对象,该操作对象以反码形式指定,用于某些向量移位;或者3)不编码任何操作对象,该字段被保留并且应当包含某个值,例如1111b。

[0280] 使用这个前缀的指令可以使用MOD R/M的R/M字段1746,以编码引用存储器地址的指令操作对象,或者编码目的地寄存器操作对象或源寄存器操作对象。

[0281] 使用这个前缀的指令可以使用MOD R/M的reg字段1744,以编码目的地寄存器操作对象或者源寄存器操作对象,或者被作为操作码扩展来对待而不被用于编码任何指令操作对象。

[0282] 对于支持四个操作对象的指令语法,vvvv、MOD R/M的R/M字段1746和MOD R/M的reg字段1744编码了四个操作对象中的三个。然后立即数1609的比特[7:4]被用来编码第三源寄存器操作对象。

[0283] 图21图示了第三前缀1601(C)的实施例。在一些实施例中,第三前缀1601(C)是EVEX前缀的实施例。第三前缀1601(C)是四字节前缀。

[0284] 第三前缀1601(C)能够在64比特模式中编码32个向量寄存器(例如,128比特、256比特和512比特寄存器)。在一些实施例中,利用写入掩码/操作掩码(参见对于先前图中的寄存器的论述,例如图15)或谓词的指令利用这个前缀。操作掩码寄存器允许条件处理或选择控制。操作掩码指令——其源/目的地操作对象是操作掩码寄存器并且将操作掩码寄存器的内容视为单个值——是使用第二前缀1601(B)而被编码的。

[0285] 第三前缀1601(C)可以编码特定于指令类别的功能(例如,具有“加载+操作”语义的紧缩指令可以支持嵌入式广播功能,具有舍入语义的浮点指令可以支持静态舍入功能,具有非舍入算术语义的浮点指令可以支持“抑制所有异常”功能,等等)。

[0286] 第三前缀1601(C)的第一字节是格式字段2111,它在一种示例中具有62H的值。随后的字节被称为有效载荷字节2115-2119,并且共同形成P[23:0]的24比特值,以(本文中详述的)一个或多个字段的形式提供特定的能力。

[0287] 在一些实施例中,有效载荷字节2119的P[1:0]与低位的两个mm比特相同。在一些实施例中,P[3:2]被保留。比特P[4](R')在与P[7]和MOD R/M的reg字段1744相组合时允许访问高16向量寄存器集合。当不需要SIB类型寻址时,P[6]也可以提供对高16向量寄存器的访问。P[7:5]由R、X和B构成,它们是针对向量寄存器、通用寄存器、存储器寻址的操作对象指定符修饰符比特,并且当与MOD R/M寄存器字段1744和MOD R/M的R/M字段1746相组合时,允许访问超出低8个寄存器的下一组8个寄存器。P[9:8]提供等同于一些传统前缀的操作码扩展性(例如,00=无前缀,01=66H,10=F3H,并且11=F2H)。P[10]在一些实施例中是固定值1。被示为vvvv的P[14:11]可以被用于:1)编码第一源寄存器操作对象,该操作对象以反转(反码)形式指定,对于具有2个或更多个源操作对象的指令有效;2)编码目的地寄存器操作对象,该操作对象以反码形式指定,用于某些向量移位;或者3)不编码任何操作对象,该字段被保留并且应当包含某个值,例如1111b。

[0288] P[15]类似于第一前缀1601(A)和第二前缀1611(B)的W,并且可以作为操作码扩展比特或操作对象大小提升。

[0289] P[18:16]指定操作掩码(写入掩码)寄存器(例如,写入掩码/谓词寄存器1515)中的寄存器的索引。在本发明的一种实施例中,特定值aaa=000具有特殊行为,暗示着没有操作掩码被用于这个特定的指令(这可以通过多种方式实现,包括使用硬连线到全一的操作掩码或者绕过掩蔽硬件的硬件)。当合并时,向量掩码允许目的地中的任何元素集合在(由基本操作和增强操作指定的)任何操作的执行期间被保护免于更新;在另一实施例中,保留目的地的每个元素的旧值(如果相应的掩码比特具有0值)。与之不同,当归零时,向量掩码允许目的地中的任何元素集合在(由基本操作和增强操作指定的)任何操作的执行期间被归零;在一种实施例中,目的地的元素在相应掩码比特具有0值时被设置到0。这个功能的子集是对于正被执行的操作的向量长度(即,被修改的元素的跨度,从第一个到最后一个)进行控制的能力;然而,被修改的元素不一定要连续。从而,操作掩码字段允许了部分向量操作,包括加载、存储、算术、逻辑,等等。虽然在描述的本发明的实施例中,操作掩码字段的内容选择若干个操作掩码寄存器中包含要使用的操作掩码的那一个(从而操作掩码字段的内容间接标识了要执行的掩蔽),但替代地或附加地,替代性实施例允许掩码写入字段的内容

直接指定要执行的掩蔽。

[0290] P[19]可以与P[14:11]相组合，来以非破坏性源语法编码第二源向量寄存器，该语法可以利用P[19]来访问高16个向量寄存器。P[20]编码多种功能，这些功能在不同类别的指令中有所不同，并且可以影响向量长度/舍入控制指定符字段(P[22:21])的含义。P[23]指示出对合并-写入掩蔽的支持(例如，当被设置为0时)或者对归零和合并-写入掩蔽的支持(例如，当被设置为1时)。

[0291] 下面的表格详述了在使用第三前缀1601(C)的指令中对寄存器的编码的示例实施例。

	4	3	[2:0]	寄存器类型	通常用途
REG	R'	R	MOD R/M reg	GPR，向量	目的地或源
VVVV	V'	vvvv		GPR，向量	第 2 源或目的地
RM	X	B	MOD R/M R/M	GPR，向量	第 1 源或目的地
基址	0	B	MOD R/M R/M	GPR	存储器寻址
索引	0	X	SIB.索引	GPR	存储器寻址
VIDX	V'	X	SIB.索引	向量	VSIB 存储器寻址

表格2:64比特模式中的32寄存器支持

	[2:0]	寄存器类型	通常用途
REG	MOD R/M reg	GPR，向量	目的地或源
VVVV	vvvv	GPR，向量	第 2 源或目的地
RM	MOD R/M R/M	GPR，向量	第 1 源或目的地
基址	MOD R/M R/M	GPR	存储器寻址
索引	SIB.索引	GPR	存储器寻址
VIDX	SIB.索引	向量	VSIB 存储器寻址

表格3:在32比特模式中编码寄存器指定符

	[2:0]	寄存器类型	通常用途
REG	MOD R/M reg	k0-k7	源
VVVV	vvvv	k0-k7	第 2 源
RM	MOD R/M R/M	k0-k7	第 1 源
{k1]	aaa	k0[1]-k7	操作掩码

表格4:操作掩码寄存器指定符编码

[0292] 程序代码可被应用到输入指令以执行本文描述的功能并且生成输出信息。输出信息可以按已知的方式被应用到一个或多个输出设备。对于本申请而言,处理系统包括任何具有处理器的系统,处理器例如是数字信号处理器(digital signal processor,DSP)、微控制器、专用集成电路(application specific integrated circuit,ASIC)、或者微处理器。

[0293] 可以用面向过程或面向对象的高级编程语言来实现程序代码以与处理系统进行通信。如果希望,也可以用汇编或机器语言来实现程序代码。实际上,本文描述的机制在范围上不限于任何特定的编程语言。在任何情况下,该语言可以是编译语言或者解释语言。

[0294] 本文公开的机制的实施例可以用硬件、软件、固件或者这些实现途径的组合来实现。本发明的实施例可以被实现为计算机程序或程序代码,在包括至少一个处理器、存储系统(包括易失性和非易失性存储器和/或存储元件)、至少一个输入设备和至少一个输出设备的可编程系统上执行。

[0295] 至少一个实施例的一个或多个方面可以由被存储在机器可读介质上的代表性指令来实现,这些指令代表处理器内的各种逻辑,这些指令当被机器读取时,使得该机器制作用于执行本文所述技术的逻辑。这些表现形式被称为“IP核”,可以被存储在有形机器可读介质上并被提供给各种客户或制造设施,以加载到实际上制作该逻辑或处理器的制作机器中。

[0296] 这些机器可读存储介质可包括——但不限于——由机器或设备制造或形成的物品的非暂态有形布置形式,包括存储介质,例如:硬盘,任何其他类型的盘(包括软盘、光盘、致密盘只读存储器(compact disk read-only memory,CD-ROM)、可重写致密盘(compact disk rewritable,CD-RW)、以及磁光盘),半导体器件(例如,只读存储器(read-only memory,ROM),诸如动态随机访问存储器(dynamic random access memory,DRAM)、静态随机访问存储器(static random access memory,SRAM)之类的随机访问存储器(random access memory,RAM),可擦除可编程只读存储器(erasable programmable read-only memory,EPROM),闪速存储器,电可擦除可编程只读存储器(electrically erasable

programmable read-only memory,EEPROM),相变存储器(phase change memory,PCM)),磁卡或光卡,或者适合用于存储电子指令的任何其他类型的介质。

[0297] 因此,本发明的实施例还包括非暂态有形机器可读介质,这些介质包含指令或者包含定义本文描述的结构、电路、装置、处理器和/或系统特征的设计数据,例如硬件描述语言(Hardware Description Language,HDL)。这种实施例也可以被称为程序产品。

仿真(包括二进制转译、代码变形,等等)

[0298] 在一些情况下,指令转换器可以被用于将指令从源指令集转换到目标指令集。例如,指令转换器可以将指令转译(例如,利用静态二进制转译、包括动态编译的动态二进制转译)、变形、仿真或者以其他方式转换到要被核心处理的一个或多个其他指令。可以用软件、硬件、固件或者其组合来实现指令转换器。指令转换器可以在处理器上、在处理器外、或者一部分在处理器上而一部分在处理器外。

[0299] 图22图示根据本发明的实施例的对照使用软件指令转换器将源指令集中的二进制指令转换成目标指令集中的二进制指令的框图。在图示的实施例中,指令转换器是软件指令转换器,但可替代地,可以用软件、固件、硬件或者其各种组合来实现指令转换器。图22示出了高级语言2202的程序可以被利用第一ISA编译器2204编译以生成第一ISA二进制代码2206,该代码可以由具有至少一个第一ISA指令集核心的处理器2216原生执行。具有至少一个第一ISA指令集核心的处理器2216代表任何这样的处理器:这种处理器能够通过兼容地执行或以其他方式处理(1)第一ISA指令集核心的指令集的实质部分或者(2)以在具有至少一个第一ISA指令集核心的Intel®处理器上运行为目标的应用或其他软件的目标代码版本,来执行与具有至少一个第一ISA指令集核心的Intel处理器基本上相同的功能,以便实现与具有至少一个第一ISA指令集核心的处理器基本上相同的结果。第一ISA编译器2204代表可操作来生成第一ISA二进制代码2206(例如,目标代码)的编译器,该第一ISA二进制代码在进行或不进行额外的链接处理的情况下能够在具有至少一个第一ISA指令集核心的处理器2216上被执行。类似地,图22示出了高级语言2202的程序可以被利用替代性指令集编译器2208来编译以生成替代性指令集二进制代码2210,该代码可以由没有第一ISA核心的处理器2214原生执行。指令转换器2212被用来将第一ISA二进制代码2206转换成可以由没有第一ISA指令集核心的处理器2214原生执行的代码。这种转换后的代码可能不会与替代性指令集二进制代码2210相同,这是因为能够进行这一点的指令转换器难以制作;然而,转换后的代码将实现总的操作,并由来自该替代性指令集的指令构成。从而,指令转换器2212代表软件、固件、硬件或它们的组合,其通过仿真、模拟或任何其他过程,允许不具有第一ISA指令集处理器或核心的处理器或其他电子设备执行第一ISA二进制代码2206。

[0300] 对“一个实施例”、“实施例”、“示例实施例”等的引用指示所描述的实施例可包括特定的特征、结构或特性,但是每个实施例可以不一定包括该特定的特征、结构或特性。而且,此类短语不一定是指同一实施例。此外,当结合实施例描述特定的特征、结构或特性时,认为结合无论是否被明确描述的其他实施例而影响此类特征、结构或特性是在本领域技术人员的知识范围之内的。

[0301] 此外,在上文描述的各实施例中,除非另外专门指出,否则,诸如短语“A、B或C中的至少一个”之类的分隔语言旨在被理解为意味着A、B、或C、或其任何组合(例如,A、B、和/或C)。由此,分隔语言不旨在也不应当被理解为暗示给定的实施例要求A中的至少一个、B中的

至少一个或C中的至少一个各自都存在。

[0302] 因此,说明书和附图应被认为是说明性而非限制性意义的。然而,将显而易见的是,可对这些实现方式作出各种修改和改变,而不背离如权利要求中所述的本公开的更宽泛的精神和范围。

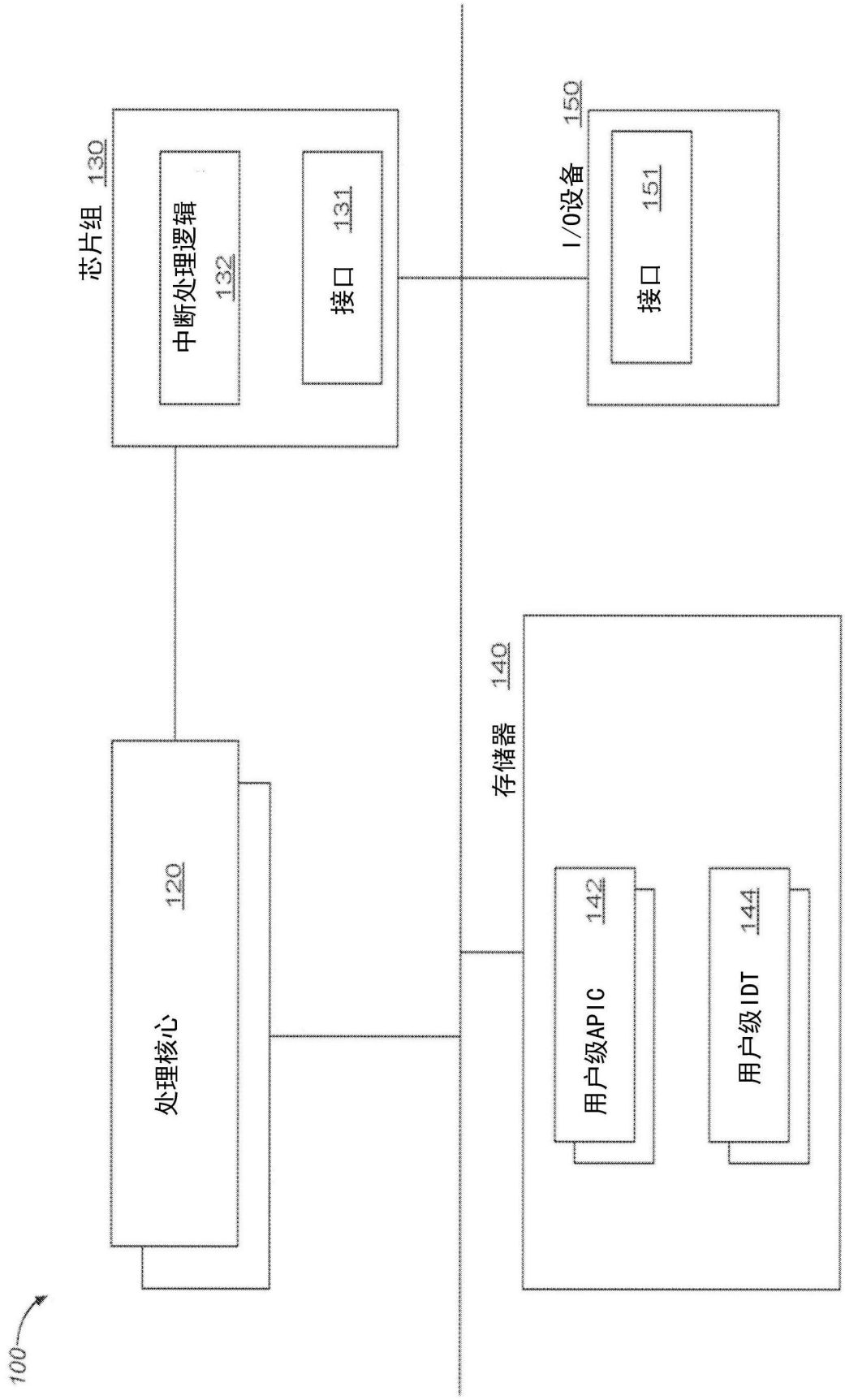

芯片组 130
中断处理逻辑 132
接口 131
I/O设备 150
接口 151
处理核心 120
存储器 140
用户级APIC 142
用户级IDT 144
100

图1

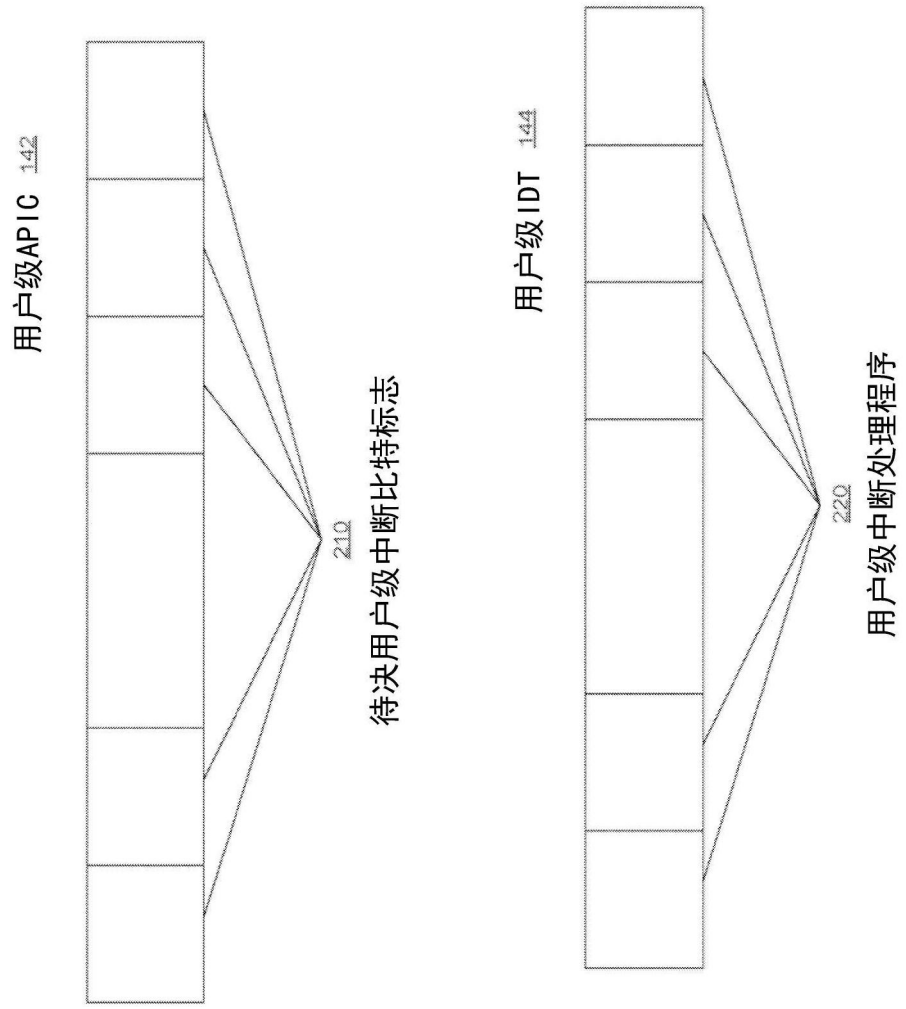


图2

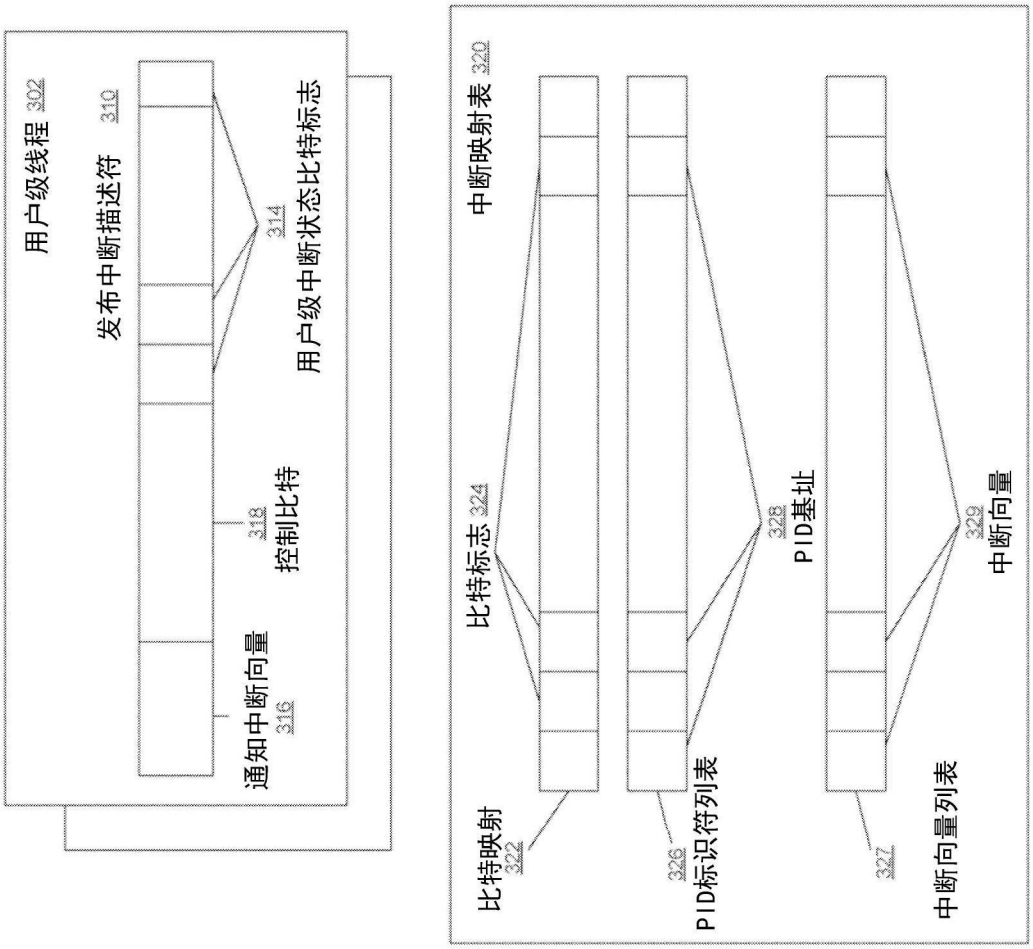

用户级线程 302
发布中断描述符 310
314
用户级中断状态比特标志
控制比特 318
通知中断向量 316
中断映射表 320
比特标志 324
比特映射 322
PID标识符列表 326
PID基址 328
中断向量 329
中断向量列表 327

图3

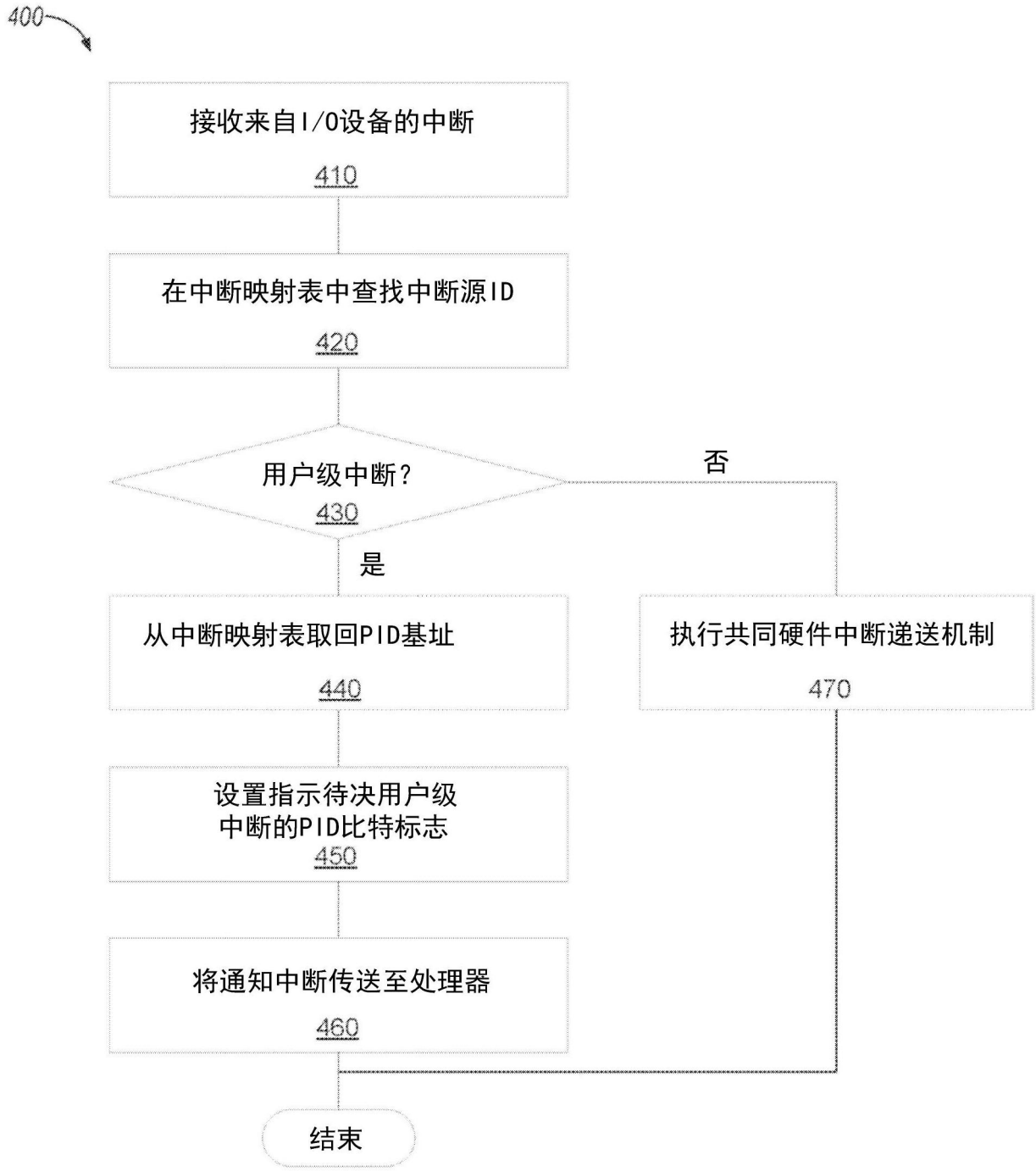

400
接收来自I/O设备的中断
410
在中断映射表中查找中断源ID
420
用户级中断?
430
否
是
从中断映射表取回PID基址
440
执行共同硬件中断递送机制
470
设置指示待决用户级
中断的PID比特标志
450
将通知中断传送至处理器
460
结束

图4

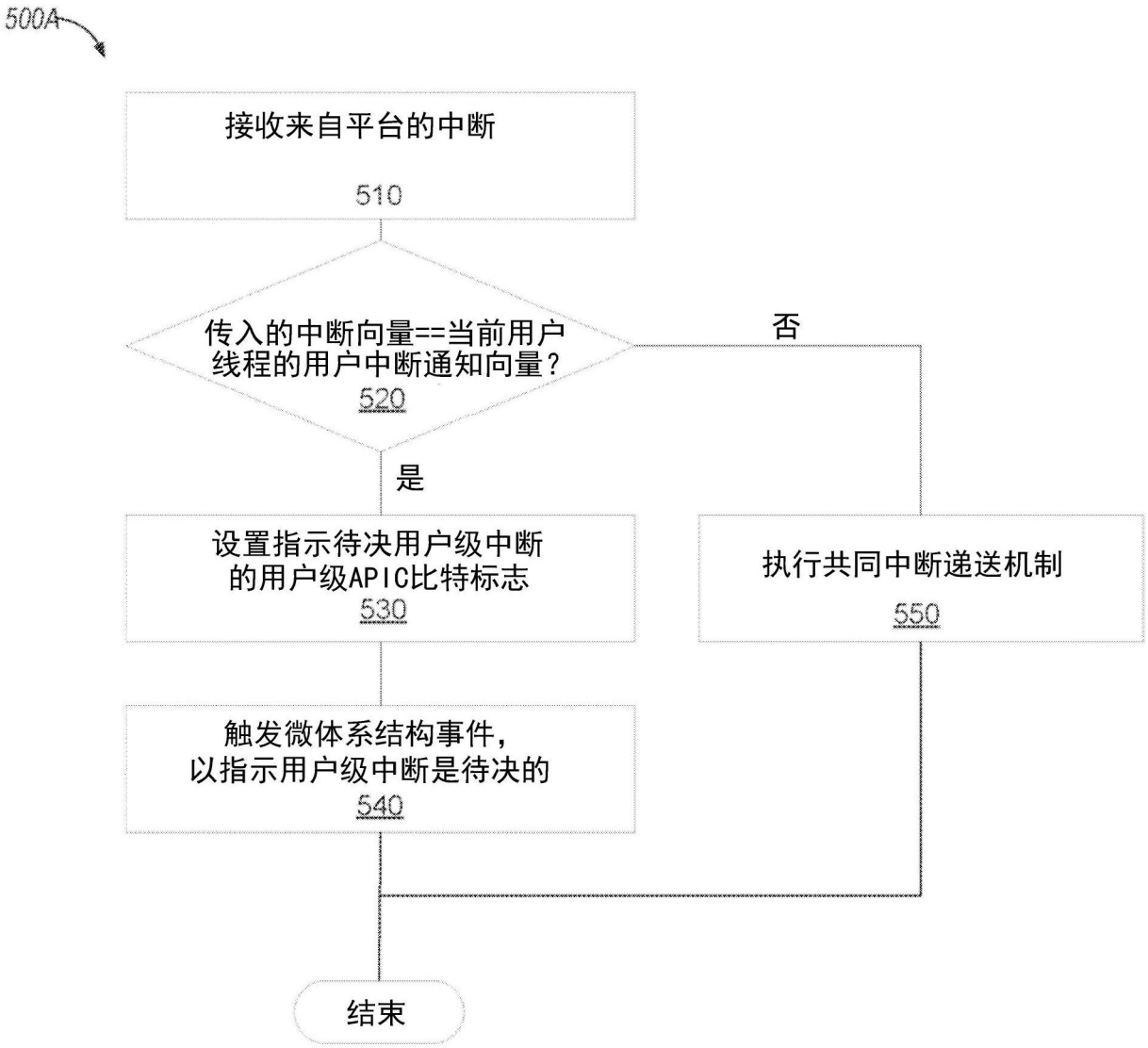

500A
接收来自平台的中断
510
传入的中断向量==当前用户线程的用户中断通知向量？
520
否
是
设置指示待决用户级中断的用户级APIC比特标志
530
执行共同中断递送机制
550
触发微体系结构事件，以指示用户级中断是待决的
540
结束

图5A


500B
检测指示用户级中断待决的微体系结构事件
560
选择优先级最高的用户级中断以供进行递送
570
查找用户级IDT，以标识用于所选择的中断的中断处理程序
580
调用所选择的中断处理程序
590
结束

图5B

CLUI-清除用户中断标志

操作码/指令	Op/En	64/32比特模式支持	CPUID特征标志	描述
F3 0F 01 EE CLUI	ZO	V/V	UINTR	清除用户中断标志；当用户中断标志被清除时，用户中断受阻。

指令操作对象编码

Op/En	元组	操作对象1	操作对象2	操作对象3	操作对象4
ZO	不适用	不适用	不适用	不适用	不适用

描述

CLUI清除用户中断标志（UIF）。其效果立即发生：用户中断无法在CLUI之后的指令边界上被递送。

CLUI在事务性区域内的执行引起事务性中止；该中止如果是由于CLI的执行引起的则其按原样加载EAX。

操作

```
UIF := 0;
```

受影响的标志

无。

受保护模式异常

#UD 在受保护模式下，CLUI指令不被识别。

真实地址模式异常

#UD 在真实地址模式下，CLUI指令不被识别。

虚拟-8086模式异常

#UD 在虚拟-8086模式下，CLUI指令不被识别。

兼容性模式异常

#UD 在兼容性模式下，CLUI指令不被识别。

64比特模式异常

#UD 如果锁定前缀被使用。
如果在飞地内部被执行。
如果CR4.UINTR=0.
如果CPUID.07H.0H:EDX.UINTR[比特5]=0。

图6

SENDUIPI-发送用户处理器间中断

操作码/指令	Op/En	64/32比特模式支持	CPUID特征标志	描述
F3 0F C7 /6 SENDUIPI reg	A	V/V	UINTR	发送处理器间用户中断。

指令操作对象编码

Op/En	元组	操作对象1	操作对象2	操作对象3	操作对象4
A	不适用	ModRM:reg (r, w)	不适用	不适用	不适用

描述

SENDUIPI指令取得单个寄存器操作对象。该操作对象始终具有64比特；
操作对象大小超控（例如，前缀66）被忽略。

虽然SENDUIPI可在任何特权级别下被执行，但指令的所有存储器访问以管理程序特权被执行。

SENDUIPI指令的虚拟化（具体而言，通知中断的发送的虚拟化）在章节11.9.2.5中讨论。

操作部分引用值UITTADDR和UITTSZ。这些值在章节11.3.1中被定义。它还包括对用户发布中断描述符(UPID)的操作。UPID的格式在章节11.5中被定义。

操作

```
IF reg > UITTSZ;
    THEN #GP(0);
FI;
read tempUITTE from 16 bytes at UITTADDR+ (reg « 4);
IF tempUITTE.V = 0 or tempUITTE sets any reserved bit (see Section 11.7.1)
    THEN #GP(0);
FI;
read tempUPID from 16 bytes at tempUITTE.UPIDADDR;// 在锁定之下
IF tempUPID sets any reserved bits or bits that must be zero （参见表11-1）
    THEN #GP(0); // 释放锁
FI;
tempUPID.PIR[tempUITTE.UV] := 1;
IF tempUPID.SN = tempUPID.ON = 0
    THEN
        tempUPID.ON := 1;
        sendNotify := 1;
    ELSE sendNotify := 0;
FI;
write tempUPID to 16 bytes at tempUITTE.UPIDADDR;// 释放锁
IF sendNotify = 1
    THEN
        IF local APIC is in x2APIC mode
            THEN send ordinary IPI with vector tempUPID.NV
                to 32-bit physical APIC ID tempUPID.NDST;
            ELSE send ordinary IPI with vector tempUPID.NV
                to 8-bit physical APIC ID tempUPID.NDST[15:8];
        FI;
FI;
```

受影响的标志

无。

受保护模式异常

#UD	在受保护模式下，SENDUIPI指令不被识别。

真实地址模式异常

#UD	在真实地址模式下，SENDUIPI指令不被识别。

虚拟-8086模式异常

#UD	在虚拟-8086模式下，SENDUIPI指令不被识别。

兼容性模式异常

#UD	在兼容性模式下，SENDUIPI指令不被识别。

64比特模式异常

#UD	如果锁定前缀被使用。 如果在飞地内部被执行。 如果CR4.UINTR=0. 如果CPUID.07H.0H:EDX.UINTR[比特5]=0。
#PF	如果发生页错误。
#GP	如果寄存器操作对象的值超过UITTSZ。 如果所选择的UITTE无效或者对任何保留比特进行设置。 如果所选择的UPID对任何保留比特进行设置。 如果存在使用相对于当前分页模式不是正则的线性地址访问存储器的尝试。

图7

STUI-设置用户中断标志

操作码/指令	Op/En	64/32比特模式支持	CPUID特征标志	描述
F3 0F 01 EF STUI	ZO	V/V	UINTR	设置用户中断标志。

指令操作对象编码

Op/En	元组	操作对象1	操作对象2	操作对象3	操作对象4
ZO	不适用	不适用	不适用	不适用	不适用

描述

STUI设置用户中断标志（UIF）。其影响立即发生；用户中断可在STUI之后的指令边界上被递送。
（这与STI相反，STI的影响被延迟了一个指令）。
STUI在事务性区域内的执行引起事务性中止；该中止如果是由于STI的执行引起的则其按原样加载EAX。

操作

```
UIF := 1;
```

受影响的标志

无。

受保护模式异常

#UD	在受保护模式下，STUI指令不被识别。

真实地址模式异常

#UD	在真实地址模式下，STUI指令不被识别。

虚拟-8086模式异常

#UD	在虚拟-8086模式下，STUI指令不被识别。

兼容性模式异常

#UD	在兼容性模式下，STUI指令不被识别。

64比特模式异常

#UD	如果锁定前缀被使用。 如果在飞地内部被执行。 如果CR4.UINTR=0. 如果CPUID.07H.0H:EDX.UINTR[比特5]=0。

图8

TESTUI-确定用户中断标志

操作码/指令	Op/En	64/32比特模式支持	CPUID特征标志	描述
F3 0F 01 ED TESTUI	ZO	V/V	UINTR	将UIF的当前值复制到EFLAGS.CF。

指令操作对象编码

Op/En	元组	操作对象1	操作对象2	操作对象3	操作对象4
ZO	不适用	不适用	不适用	不适用	不适用

TESTUI将用户中断标志（UIF）的当前值复制到EFLAGS.CF。该指令可以被执行而不论CPL如何。

TESTUI可在事务性区域内正常地被执行。

操作

```
CF := UIF;
ZF := AF := OF := PF := SF := 0;
```

受影响的标志

ZF、OF、AF、PF、SF标志被清除并且CF标志为用户中断标志的值。

受保护模式异常

#UD	在受保护模式下，TESTUI指令不被识别。

真实地址模式异常

#UD	在真实地址模式下，TESTUI指令不被识别。

虚拟-8086模式异常

#UD	在虚拟-8086模式下，TESTUI指令不被识别。

兼容性模式异常

#UD	在兼容性模式下，TESTUI指令不被识别。

64比特模式异常

#UD	如果锁定前缀被使用。 如果在飞地内部被执行。 如果CR4.UINTR=0. 如果CPUID.07H.0H:EDX.UINTR[比特5]=0。

图9

UIRET-用户中断返回

操作码/指令	Op/En	64/32比特模式支持	CPUID特征标志	描述
F3 0F 01 EC UIRET	ZO	V/V	UINTR	从处置用户中断返回。

指令操作对象编码

Op/En	元组	操作对象1	操作对象2	操作对象3	操作对象4
ZO	不适用	不适用	不适用	不适用	不适用

描述

UIRET从对用户中断的处置返回。其可以被执行而不论CPL如何。

UIRET在事务性区域内的执行引起事务性中止；该中止如果是由于IRET的执行引起的则其按原样加载EAX。

UIRET可以通过体系结构最后分支记录（LBR）、英特尔处理器追踪（英特尔PT）和性能监视来跟踪。对于英特尔PT和LBR两者，UIRET以与IRET精确地相同的方式被记录。由此，对于LBR，UIRET落入OTHER_BRANCH类别中，这暗示IA32_LBR_CTL.OTHER_BRANCH[比特22]必须被设置以记录用户中断递送，并且IA32_LBR_x_INFO.BR_TYPE字段将指示针对任何所记录的用户中断的OTHER_BRANCH。对于英特尔PT，控制流追踪必须通过设置IA32_RTIT_CTL.BranchEn[比特 13]被启用。

UIRET也将递增性能计数器，对于该性能计数器，对BR_INST_RETIRED.FAR_BRANCH计数被启用。

操作

```
Pop tempRIP;
Pop tempRFLAGS; //参见下文，针对这如何被用于加载RFLAGS
Pop tempRSP;
IF tempRIP is not canonical in current paging mode
    THEN #GP(0);
FI;
IF shadow stack is enabled for CPL = 3
    THEN
        PopShadowStack SSRIP;
        IF SSRIP ≠ tempRIP
            THEN #CP (FAR-RET/IRET);
        FI;
FI;
RIP := tempRIP;
//RFLAGS中的更新，仅CF、PF、AF、ZF、SF、TF、DF、OF、NT、RF、AC和ID
RFLAGS := (RFLAGS & ~254DD5H)|(tempRFLAGS & 254DD5H);
RSP := tempRSP;
UIF := 1;
Clear any cache-line monitoring established by MONITOR or UMONITOR;
```

受影响的标志

参见操作部分。

受保护模式异常

#UD	在受保护模式下，UIRET指令不被识别。

真实地址模式异常

#UD	在受保护模式下，UIRET指令不被识别。

虚拟-8086模式异常

#UD	在虚拟-8086模式下，UIRET指令不被识别。

兼容性模式异常

#UD	在兼容性模式下，UIRET指令指令不被识别。

64比特模式异常

#GP(0)	如果返回指令指针是非正则的。
#SS(0)	如果使值出栈的尝试引起非正则的地址被引用。
#PF(fault-code)	如果发生页错误。
#AC(0)	如果对齐检查被启用，并且非对齐存储器引用被作出而当前特权级别为3。
#CP	如果来自栈的返回指令指针和来自影子栈的返回指令指针不匹配。
#UD	如果锁定前缀被使用。 如果在飞地内部被执行。 如果CR4.UINTR=0. 如果CPUID.07H.0H:EDX.UINTR[比特5]=0。

图10

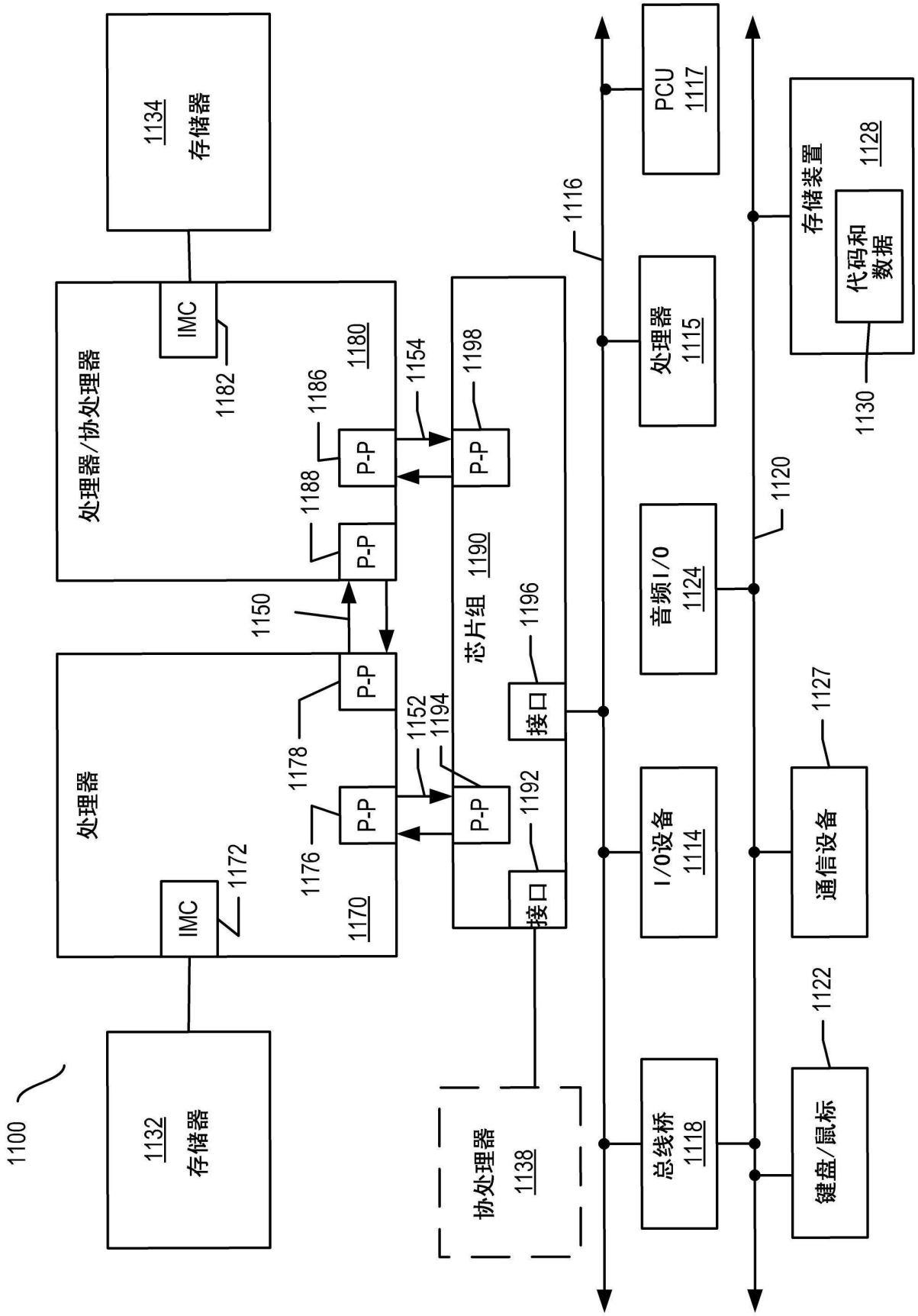

1100
1132
存储器
1134
存储器
处理器
处理器/协处理器
IMC
1172
IMC
1182
1170
1180
P-P
P-P
P-P
P-P
1176
1178
1186
1188
1150
1152
1154
1194
芯片组 1190
1196
接口
1192
接口
1198
协处理器
1138
总线桥
1118
I/O设备
1114
音频I/O
1124
处理器
1115
PCU
1117
1116
1120
1122
键盘/鼠标
1127
通信设备
存储装置
1128
代码和
数据
1130

图11

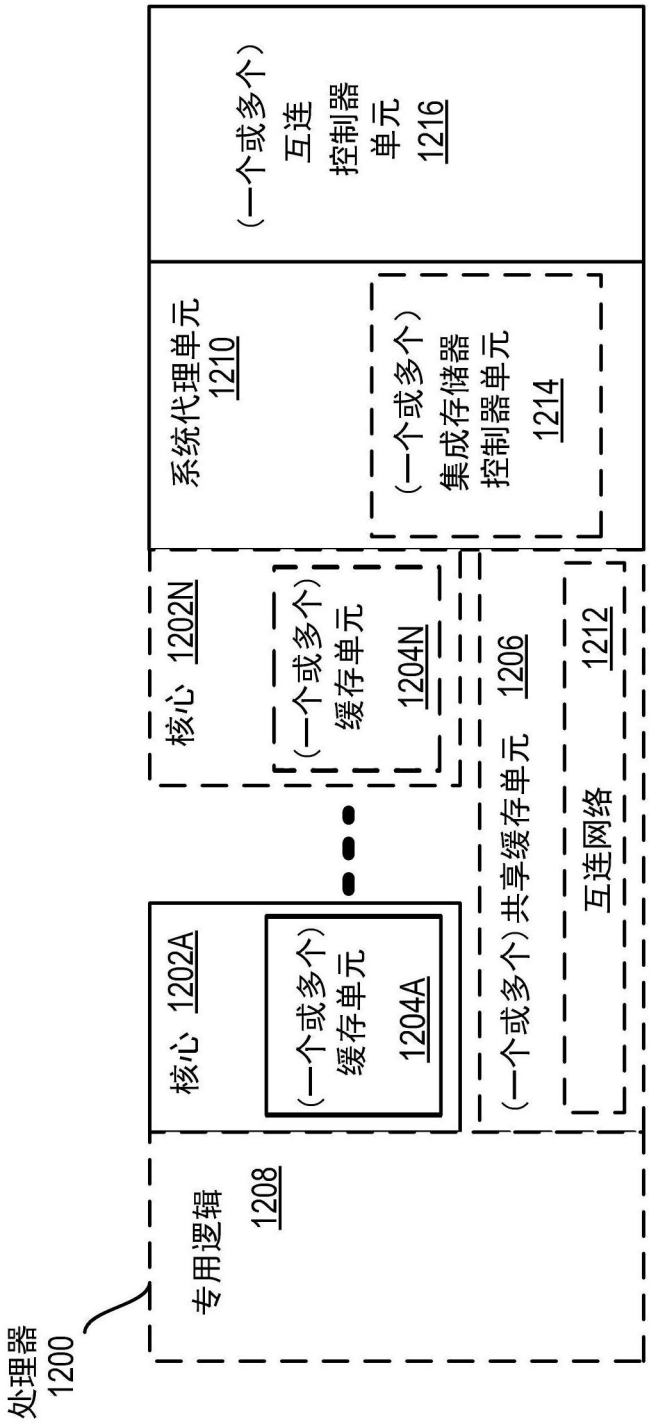

处理器
1200
专用逻辑
1208
核心 1202A
(一个或多个)
缓存单元
1204A
核心 1202N
(一个或多个)
缓存单元
1204N
(一个或多个)共享缓存单元 1206
互连网络 1212
系统代理单元
1210
(一个或多个)
集成存储器
控制器单元
1214
(一个或多个)
互连
控制器
单元
1216

图12


1300 管线
取得 1302
长度解码 1304
解码 1306
分配 1308
重命名 1310
调度 1312
寄存器读取/存储器读取 1314
执行阶段 1316
写回/存储器写入 1318
异常处置 1322
提交 1324

图13(A)

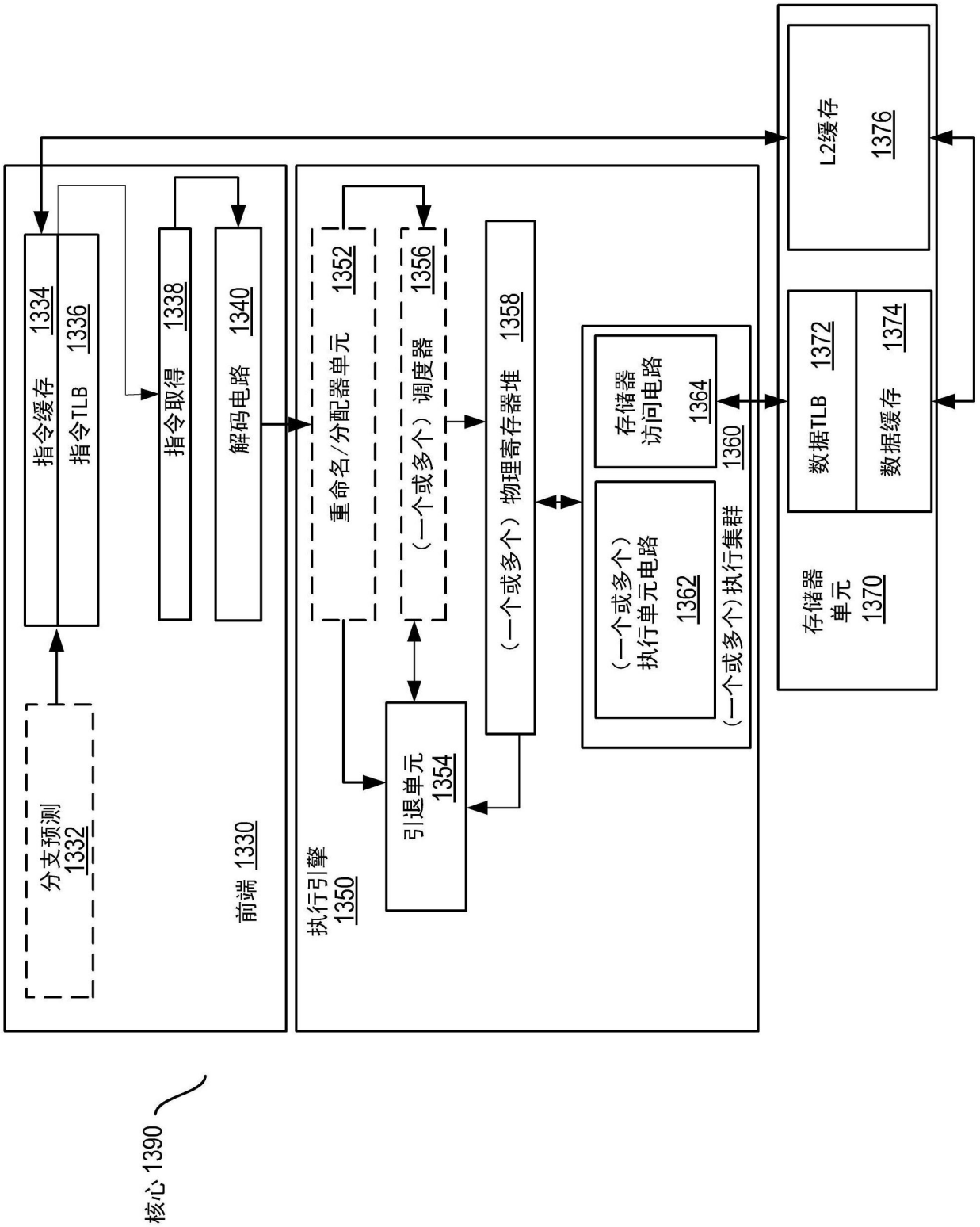

核心 1390
前端 1330
分支预测 1332
指令缓存 1334
指令TLB 1336
指令取得 1338
解码电路 1340
执行引擎 1350
重命名/分配器单元 1352
引退单元 1354
（一个或多个）调度器 1356
（一个或多个）物理寄存器堆 1358
（一个或多个）执行集群 1360
（一个或多个）执行单元电路 1362
存储器访问电路 1364
存储器单元 1370
数据TLB 1372
数据缓存 1374
L2缓存 1376

图13(B)

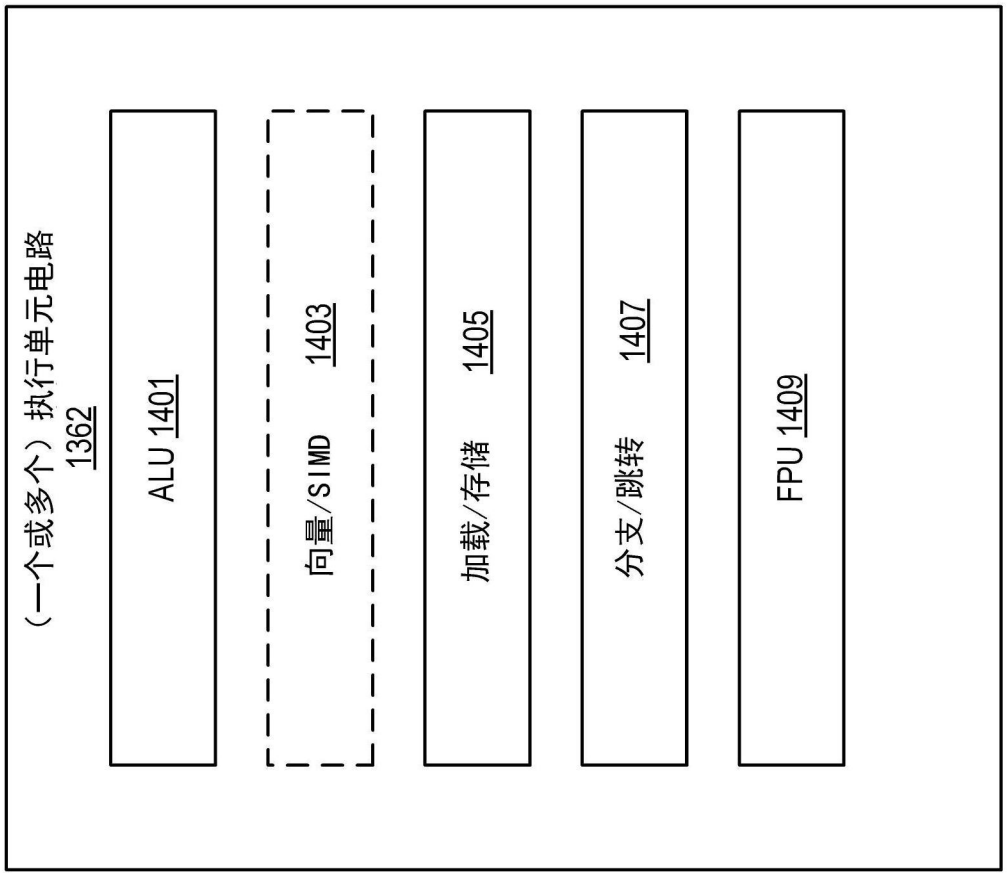

（一个或多个）执行单元电路
1362
ALU 1401
向量/SIMD 1403
加载/存储 1405
分支/跳转 1407
FPU 1409

图14

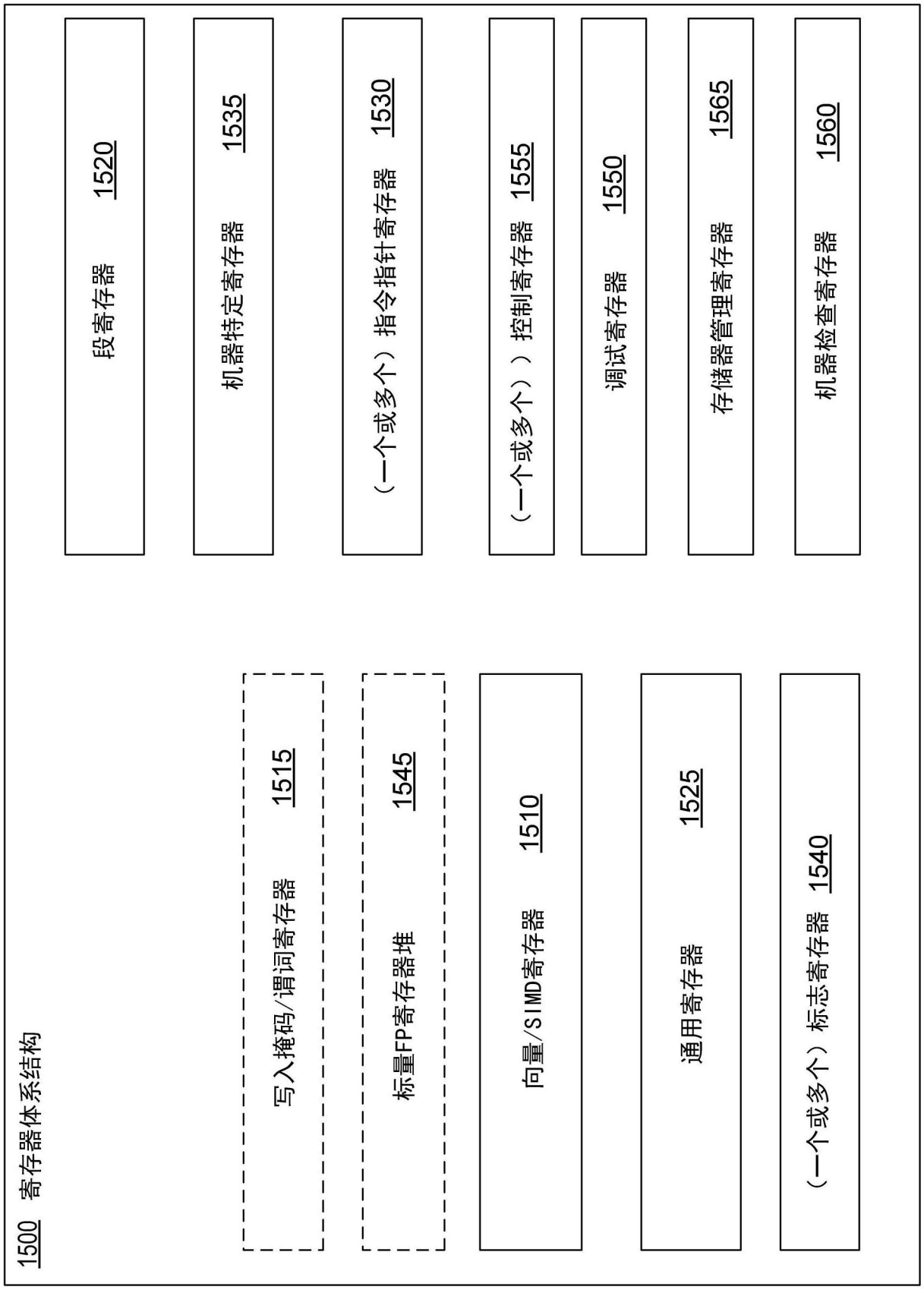

1500 寄存器体系结构
段寄存器 1520
机器特定寄存器 1535
（一个或多个）指令指针寄存器 1530
（一个或多个））控制寄存器 1555
调试寄存器 1550
存储器管理寄存器 1565
机器检查寄存器 1560
写入掩码/谓词寄存器 1515
标量FP寄存器堆 1545
向量/SIMD寄存器 1510
通用寄存器 1525
（一个或多个）标志寄存器 1540

图15

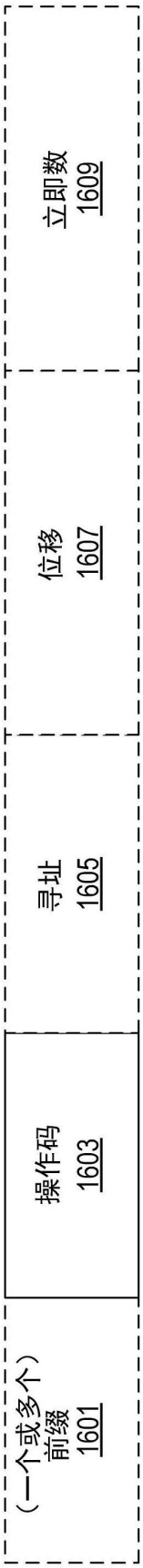


图16

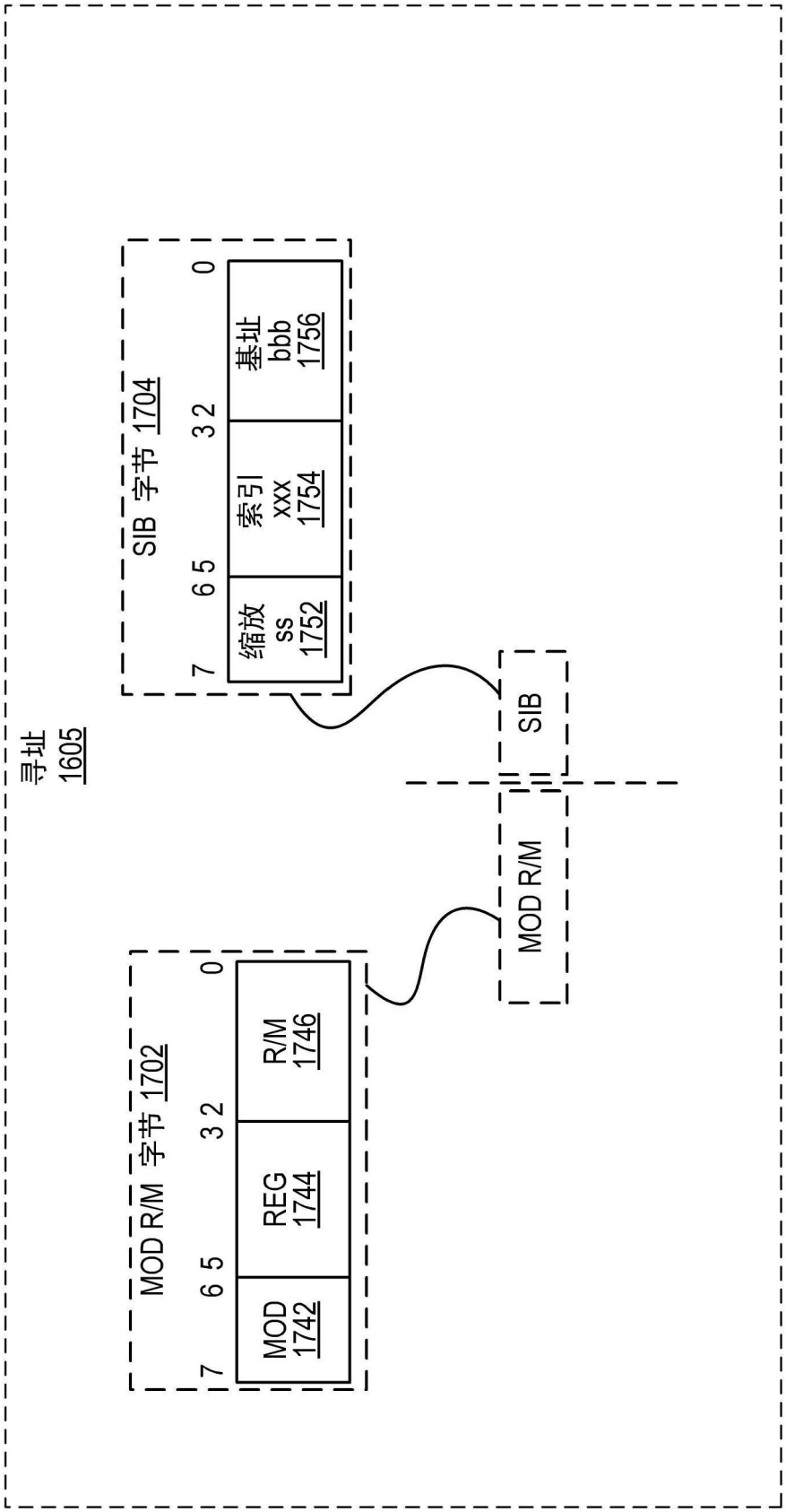

寻址
1605
SIB 字节 1704
7 6 5 3 2 0
缩放
ss
1752
索引
xxx
1754
基址
bbb
1756
SIB
MOD R/M
MOD R/M 字节 1702
7 6 5 3 2 0
MOD
1742
REG
1744
R/M
1746

图17

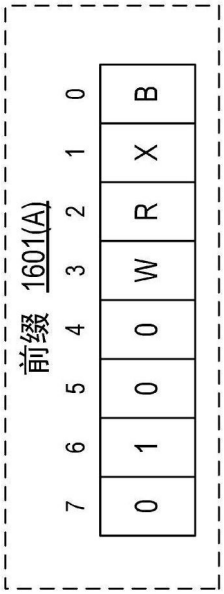

前缀 1601(A)
7 6 5 4 3 2 1 0
0 1 0 0 W R X B

图18

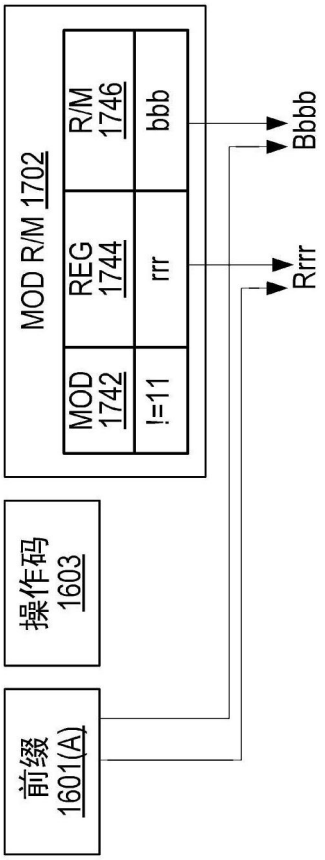

MOD R/M 1702
MOD 1742
!=11
REG 1744
rrr
R/M 1746
bbb
Bbbb
Rrrr
操作码 1603
前缀 1601(A)

图19(A)

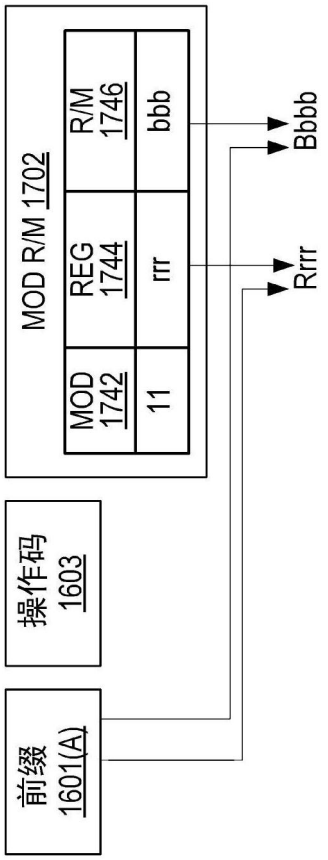

MOD R/M 1702
MOD 1742
11
REG 1744
rrr
R/M 1746
bbb
Bbbb
Rrrr
操作码 1603
前缀 1601(A)

图19(B)

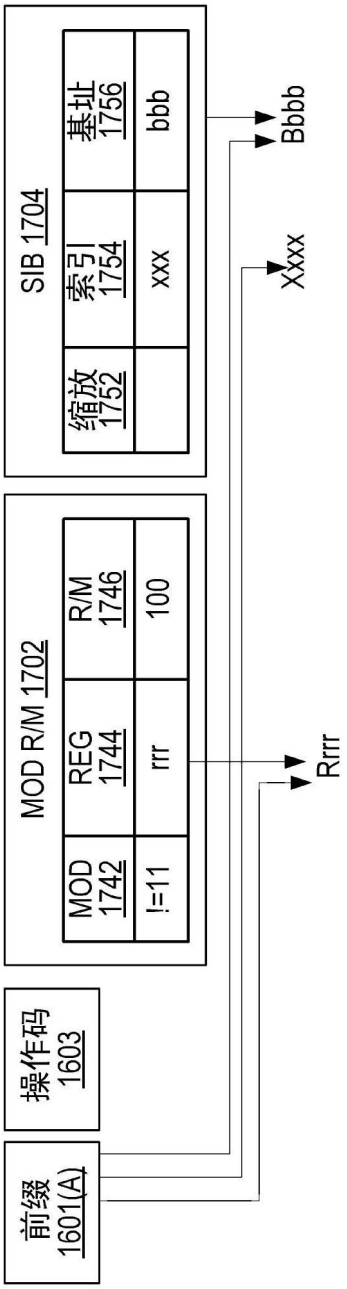

前缀 1601(A)
操作码 1603
MOD R/M 1702
MOD 1742
!=11
REG 1744
rrr
R/M 1746
100
SIB 1704
缩放 1752
索引 1754
xxx
基址 1756
bbb
Rrrr
Xxxx
Bbbb

图19(C)

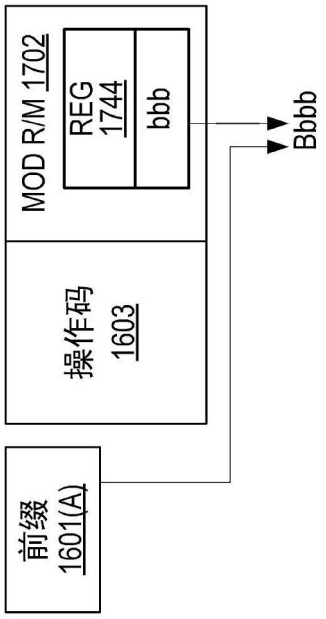


图19(D)

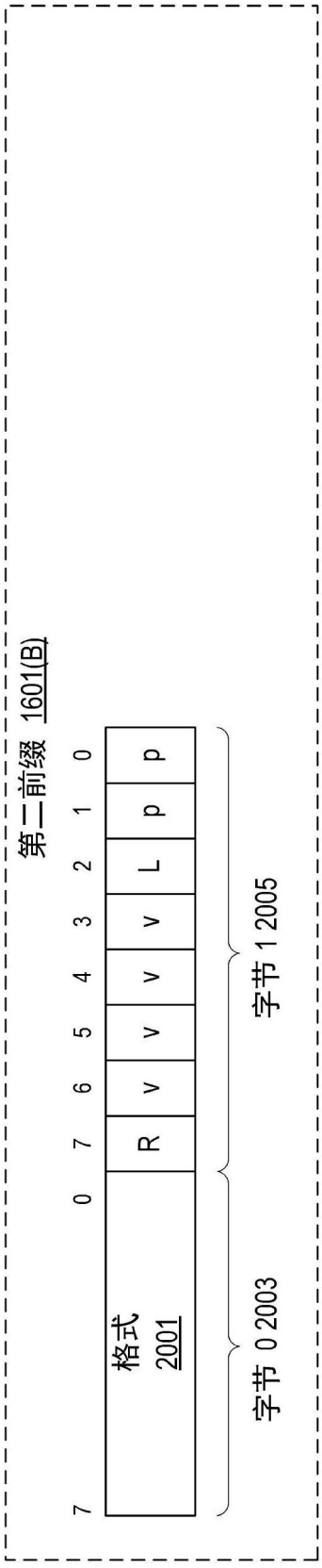


图20(A)

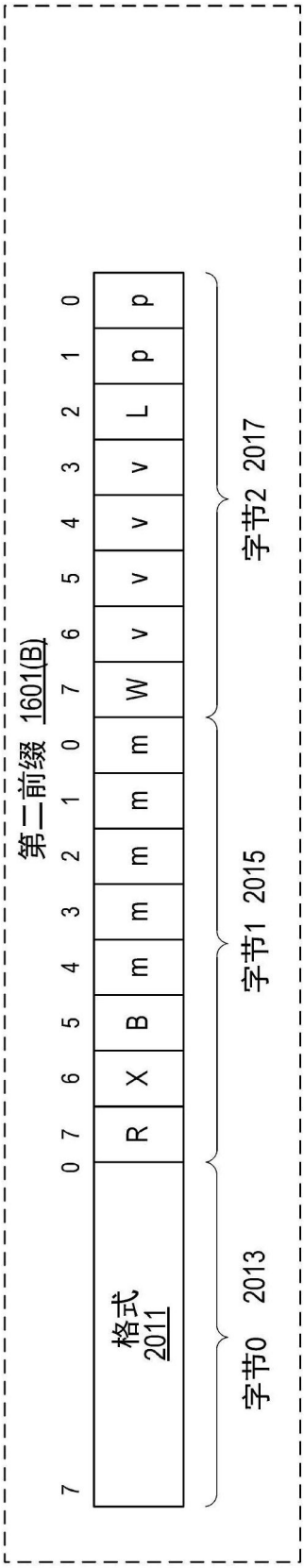

第二前缀 1601(B)
格式 2011
R
X
B
m
m
m
m
m
m
m
W
v
v
v
v
v
L
p
p
字节0 2013
字节1 2015
字节2 2017

图20(B)

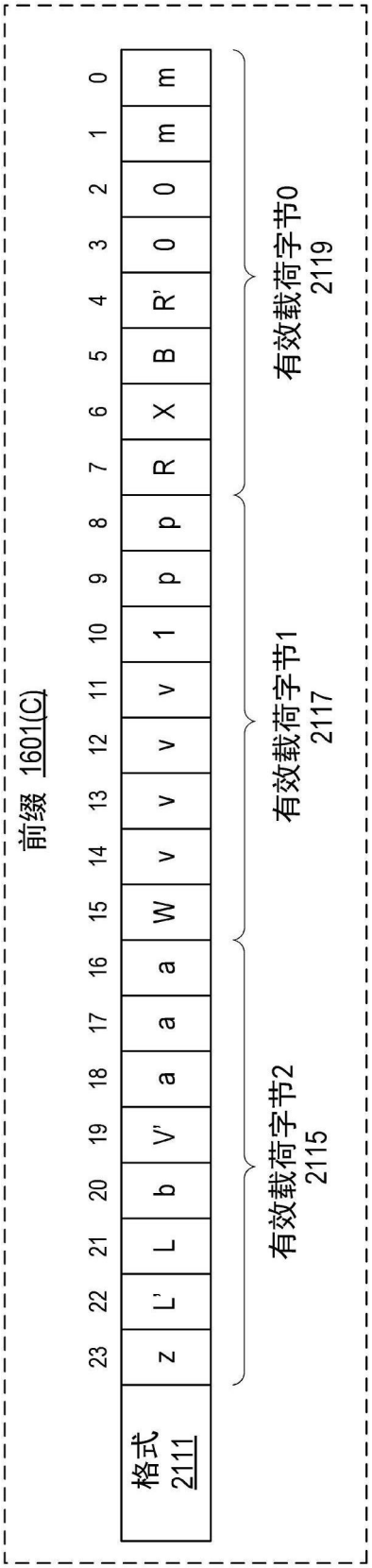

前缀 1601(C)
格式 2111
23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
z L' L b V' a a a W v v v v 1 p p R X B R' 0 0 m m
有效载荷字节2 2115
有效载荷字节1 2117
有效载荷字节0 2119

图21

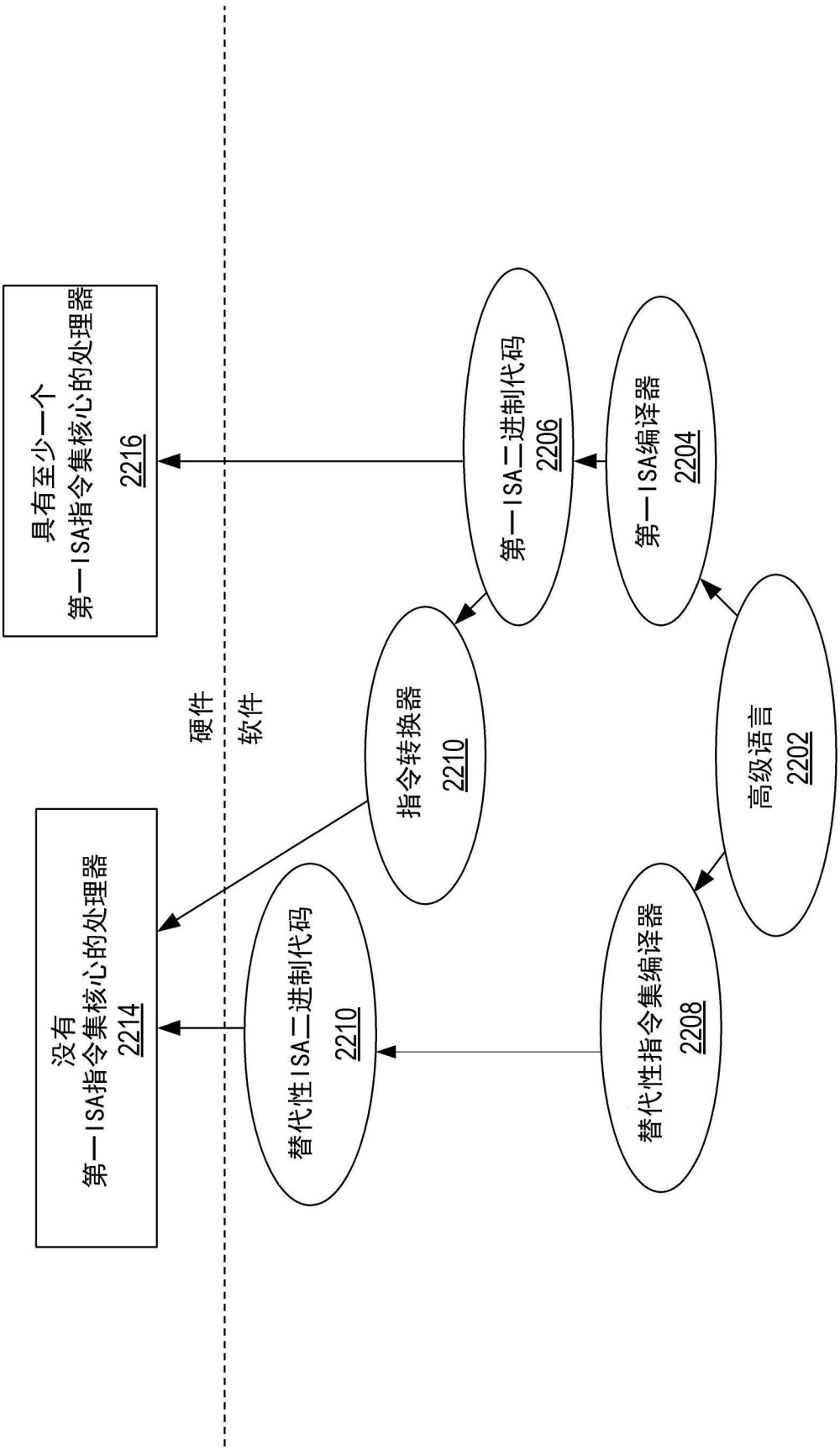

具有至少一个
第一ISA指令集核心的处理器
2216
第一ISA二进制代码
2206
第一ISA编译器
2204
指令转换器
2210
高级语言
2202
硬件
软件
没有
第一ISA指令集核心的处理器
2214
替代性ISA二进制代码
2210
替代性指令集编译器
2208

图22