(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 108885660 B
(45)授权公告日 2022.04.05

(21)申请号 201680081293.8
(22)申请日 2016.12.12
(65)同一申请的已公布的文献号
申请公布号 CN 108885660 A
(43)申请公布日 2018.11.23
(30)优先权数据
62/268,639 2015.12.17 US
62/270,187 2015.12.21 US
15/168,689 2016.05.31 US
(85)PCT国际申请进入国家阶段日
2018.08.08
(86)PCT国际申请的申请数据
PCT/US2016/066188 2016.12.12
(87)PCT国际申请的公布数据
WO2017/106101 EN 2017.06.22
(73)专利权人 查尔斯斯塔克德雷珀实验室有限公司
地址 美国马萨诸塞州
(72)发明人 A·德翁 E·博林
(74)专利代理机构 北京市正见永申律师事务所 11497
代理人 黄小临
(51)Int.Cl.
G06F 21/52(2006.01)
G06F 21/62(2006.01)
G06F 12/0875(2006.01)
G06F 9/30(2006.01)
G06F 9/38(2006.01)
(56)对比文件
US 2009164766 A1,2009.06.25
US 2008222397 A1,2008.09.11
CN 104794067 A,2015.07.22
CN 102160033 A,2011.08.17
CN 101558388 A,2009.10.14
审查员 刘燕

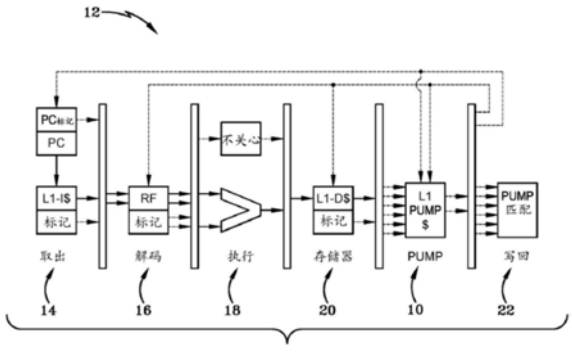
权利要求书3页 说明书129页 附图95页

(54)发明名称

用于元数据处理的技术

(57)摘要

描述了可以用来编码在处理器上运行的代码的任意数目的安全性策略的元数据处理的技术。元数据可以添加到系统中的每个字且可以使用与数据流并行工作以推行任意策略集合的元数据处理单元。在一个方面中,元数据可以特性化为无限制的和软件可编程的以可应用于各种元数据处理策略。技术和策略具有各种用途,例如包括安全、安全性和同步。另外,描述了关于在基于RISC-V架构的实施例中的元数据处理的方面和技术。



CN 108885660 B

1.一种处理当前指令的方法,包括:

对于元数据处理,接收当前指令;和

在与包括当前指令的代码执行域相隔离的元数据处理域中执行用于当前指令的元数据处理,当前指令参考具有元数据处理中使用的第一元数据标记的第一存储器位置,所述用于当前指令的元数据处理包括:

执行处理以从存储器提取第一元数据标记;

在从存储器接收用于第一存储器位置的第一元数据标记之前,确定第一存储器位置的第一元数据标记的预测值;

使用第一存储器位置的第一元数据标记的预测值,确定用于当前指令的结果操作数的第一结果元数据标记;和

从存储器接收第一元数据标记;

确定第一元数据标记是否匹配于第一元数据标记的预测值;和

响应于确定第一元数据标记匹配于第一元数据标记的预测值,使用第一结果元数据标记作为用于结果操作数的最终结果元数据标记。

2.如权利要求1所述的方法,其中,所述用于当前指令的元数据处理进一步包括:

根据当前指令和用于当前指令的输入元数据标记的集合来确定当前指令的第一规则,其中,所述第一规则包括第一存储器位置的第一元数据标记的预测值且包括第一结果元数据标记,所述第一规则包括在用于元数据处理域中的元数据处理的规则高速缓存中;和

响应于确定第一元数据标记不匹配第一元数据标记的预测值,执行用于当前指令的元数据处理域中的规则高速缓存未命中处理。

3.如权利要求2所述的方法,其中,所述用于当前指令的元数据处理域中的规则高速缓存未命中处理包括:

确定是否允许代码执行域中的当前指令的执行;

响应于确定允许代码执行域中的当前指令的执行,生成用于当前指令的新规则,其中,所述新规则根据当前指令、输入元数据标记的集合和第一元数据标记来生成;和

将新规则插入到用于元数据处理域中的元数据处理的规则高速缓存中。

4.如权利要求2所述的方法,其中,输入元数据标记的集合包括用于以下任意的元数据标记:程序计数器、当前指令和当前指令的输入操作数。

5.如权利要求1所述的方法,其中,所述结果操作数是存储执行当前指令的结果的目的地存储器位置或者目的地寄存器。

6.如权利要求2所述的方法,其中,根据包括第一级和第二级的多个级来处理指令,第一级在第二级之前发生,在所述第一级中确定第一存储器位置的第一元数据标记的预测值,所述第二级包括执行所述确定第一元数据标记是否匹配于第一元数据标记的预测值,且所述第二级还包括执行响应于确定第一元数据标记不匹配第一元数据标记的预测值而执行用于当前指令的元数据处理域中的所述规则高速缓存未命中处理。

7.如权利要求2所述的方法,其中,所述规则高速缓存可配置为根据预测选择器模式以预测模式或者正常处理模式操作,所述规则高速缓存配置为当执行所述用于当前指令的元数据处理时以预测模式操作。

8.如权利要求7所述的方法,其中,当规则高速缓存配置为以所述预测模式操作时,所

述规则高速缓存根据第一规则生成第一输出，所述第一输出包括作为第一级的输出的用于下一指令的程序计数器的元数据标记、用于当前指令的结果操作数的第一结果元数据标记和第一元数据标记的预测值，当规则高速缓存配置为以所述正常处理模式操作时，所述规则高速缓存根据不同于第一规则的第二规则生成第二输出，所述第二输出不包括第一元数据标记的预测值，且所述第二输出包括用于当前指令的结果操作数和用于下一指令的程序计数器的元数据标记。

9.如权利要求7所述的方法，其中，所述规则高速缓存当以预测模式操作时使用第一策略的规则的第一版本，且当以正常处理模式操作时使用第一策略的规则的第二版本，且其中，所述第一规则包括在规则的第一版本中且第二规则包括在规则的第二版本中。

10.一种系统，包括：

流水线处理器，包括多个流水线级，所述多个流水线级包括存储器级和写回级；

用于在存储器级的完成之前操作存储器级的集成的用于元数据处理的可编程单元，其中，用于元数据处理的可编程单元执行参考具有在元数据处理中使用的第一元数据标记的第一存储器位置的用于当前指令的元数据处理，用于元数据处理的可编程单元接收包括用于当前指令的第一元数据标记的第一输入，用于元数据处理的可编程单元生成作为到写回级的输入提供的第一输出，所述第一输出包括第一存储器位置的第一元数据标记的预测值和用于当前指令的结果操作数的第一结果元数据标记，所述第一结果元数据标记由用于元数据处理的可编程单元根据用于第一存储器位置的第一元数据标记的预测值来确定；和

写回级的硬件组件，确定用于第一存储器位置的第一元数据标记是否匹配于第一元数据标记的预测值，和当第一元数据标记匹配于第一元数据标记的预测值时使用第一结果元数据标记作为用于结果操作数的最终结果元数据标记。

11.如权利要求10所述的系统，其中，所述用于元数据处理的可编程单元是与存储器级同时操作且进一步以预测模式操作并确定第一存储器位置的第一元数据标记的预测值的第一用于元数据处理的可编程单元，且其中，所述系统包括以正常非预测模式操作且不确定第一存储器位置的第一元数据标记的任何预测值的第二用于元数据处理的可编程单元，所述第二用于元数据处理的可编程单元集成为所述存储器级和写回级之间的另一级。

12.如权利要求11所述的系统，其中，所述第一用于元数据处理的可编程单元使用第一策略的规则的第一版本用于当以预测模式操作时使用，所述第二用于元数据处理的可编程单元使用第一策略的规则的第二版本用于当以正常非预测模式操作时使用，所述第一用于元数据处理的可编程单元根据来自第一版本的第一规则来确定第一输出，所述第二用于元数据处理的可编程单元根据来自第二版本的第二规则来确定第二输出，所述第二输出包括用于第一存储器位置的第二结果元数据标记且所述第二输出作为到写回级的输入提供，且其中，当第一元数据标记不匹配预测值时，所述写回级的硬件组件另外使用第二结果元数据标记作为用于结果操作数的最终结果元数据标记。

13.一种包括在其上存储的代码的非瞬时计算机可读介质，所述代码当执行时执行处理当前指令的方法，所述方法包括：

对于元数据处理接收当前指令；和

在与包括当前指令的代码执行域相隔离的元数据处理域中执行用于当前指令的元数据处理，当前指令参考具有元数据处理中使用的第一元数据标记的第一存储器位置，所述

用于当前指令的元数据处理包括：

执行处理以从存储器提取第一元数据标记；

在从存储器接收用于第一存储器位置的第一元数据标记之前，确定第一存储器位置的第一元数据标记的预测值；

使用第一存储器位置的第一元数据标记的预测值，确定用于当前指令的结果操作数的第一结果元数据标记；和

从存储器接收第一元数据标记；

确定第一元数据标记是否匹配于第一元数据标记的预测值；和

响应于确定第一元数据标记匹配于第一元数据标记的预测值，使用第一结果元数据标记作为用于结果操作数的最终结果元数据标记。

用于元数据处理的技术

[0001] 相关申请的交叉引用

[0002] 本申请要求于2016年5月31日提交的美国申请No.15/168,689的优先权,该美国申请No.15/168,689要求于2015年12月17日提交的美国临时申请序号62/268,639,“SOFTWARE DEFINED METADATA PROCESSING”和于2015年12月21日提交的美国临时申请序号62/270,187,“SOFTWARE DEFINED METADATA PROCESSING”的优先权,将其全部通过引用包括于此。

背景技术

[0003] 本申请总的来说关于数据处理,且更具体地,涉及用于元数据处理的可编程单元。

[0004] 当今的计算机系统非常难以保护。例如,现有的处理器架构允许违反较高级抽象(abstraction)的比如缓存溢出、指针伪造等各种行为。关闭编程语言和硬件之间的间隙可能留给软件,而推行严密抽象的成本通常认为过高。

[0005] 一些近来的努力已经表明在执行期间传播(propagate)元数据以推行(enforce)在安全性违规(violation)和恶意攻击发生时抓住它们的策略(policy)的价值。这些策略可以以软件推行,但是典型地导致高的不必要的开销,比如在性能和/或成本方面,这阻碍它们的部署,或者另外鼓励了提供较少保护的粗糙近似。用于固定策略的硬件支持可以将开销减小到可接受的水平且防止大部分不期望的代码违规,比如可以由恶意代码或者恶意软件攻击执行的。例如,Intel近来公布了用于边界检查和隔离的硬件。虽然这缓解了许多当今的攻击,完全安全的系统将需要多于存储器安全性和隔离。攻击快速地演变以利用任何剩余形式的易损性。

发明内容

[0006] 因此,存在可以快速地适用于该不断变化的场景的灵活的安全性架构。将期望这种架构提供用于以最小开销的软件定义的元数据处理的支持。期望这种架构可扩展以广泛地支持和推行任意数目和类型的策略而不用对分配给元数据的位的数目设置可见的、硬的边界。可以在执行期间传播元数据以推行策略和抓住例如通过恶意代码或者恶意软件攻击的对这些策略的违反。

[0007] 根据在这里描述的技术的一个方面,是处理指令的方法,包括:对于元数据处理,接收具有关联的元数据标记的当前指令,所述元数据处理在与包括当前指令的代码执行域相隔离的元数据处理域中执行;在元数据处理域中和根据元数据标记和当前指令,确定规则(rule)是否存在于当前指令的规则高速缓存中,所述规则高速缓存包括关于由所述元数据处理使用以定义允许的操作的元数据的规则;和响应于确定没有规则存在于用于当前指令的规则高速缓存中,在元数据处理域中执行规则高速缓存未命中(miss)处理,包括:确定是否允许当前指令的执行;响应于确定允许在代码执行域中执行当前指令,生成用于当前指令的新规则;写入到寄存器;和响应于写入到寄存器,将新规则插入到规则高速缓存中。用于选择当前指令的规则的第一元数据可以存储在由元数据处理使用的多个控制状态寄存器的第一部分中,且其中,多个控制状态寄存器的第一部分可以用于将用于当前指令的

多个元数据标记传递到元数据处理域，其中，所述多个元数据标记可以用作元数据处理域中的数据。寄存器可以是由元数据处理使用的多个控制状态寄存器的第一控制状态寄存器，且其中，多个控制状态寄存器的第一部分可以用于将多个元数据标记从元数据处理域传递到规则高速缓存。多个元数据标记可以用于当前指令。新规则可以响应于将另一元数据标记写入到第一控制状态寄存器而插入到规则高速缓存中，其中，可以在当前指令的结果上放置该另一元数据标记，且该结果可以是任意目的地寄存器或者存储器位置。多个控制状态寄存器可以包括以下的任何一个或多个：包括所有其他生成的元数据标记从其推导出的初始元数据标记的自举(bootstrap)标记控制状态寄存器；指定默认元数据标记的默认标记控制状态寄存器；指定用于标记分类为公共和不可信的指令和数据的公共不可信元数据标记的公共不可信控制状态寄存器；包括写入到包括关于操作组(opgroup)的信息和用于不同操作码(opcode)的维护(care)信息的表的数据的操作组值控制状态寄存器；指定操作组值控制状态寄存器的数据写入到的表中的位置的操作组地址控制状态寄存器；和其中到泵清空(pumpflush)控制状态寄存器的写入触发规则高速缓存的清空(flush)的泵清空控制状态寄存器。多个控制状态寄存器可以包括指示元数据处理的当前模式的标记模式控制状态寄存器。标记模式控制状态寄存器可以指示何时脱离元数据处理由此不由元数据处理推行一个或多个定义的策略的规则。标记模式控制状态寄存器可以设置为定义的一组允许状态之一以指示元数据处理的当前模式。允许状态可以包括以下的任意一个：关闭状态，由此元数据处理在所有结果上写入默认标记的状态，和指示当以一个或多个指定特权级别在代码域中执行指令时参启用了元数据处理且元数据处理可操作的状态。规则高速缓存未命中处理可以在脱离元数据处理的定义的一组允许状态的第一个中执行。这些允许状态可以包括指示仅当以用户特权级别在代码域中指令执行时启用元数据处理的第一状态；指示仅当以用户或者管理程序(supervisor)特权级别在代码域中指令执行时启用元数据处理的第二状态；指示仅当以用户、管理程序或者系统管理程序(hypervisor)特权级别在代码域中指令执行时启用元数据处理的第三状态；和指示当以用户、管理程序、系统管理程序或者机器特权级别在代码域中指令执行时启用元数据处理的第四状态。可以根据标记模式控制状态寄存器的当前标记模式结合在代码域中执行的代码的当前特权级别来确定启用或者脱离元数据处理，其中，当脱离元数据处理时可以不推行一个或多个定义的策略的规则，且其中，当启用元数据处理时可以推行规则。表可以包括将指令集的操作码映射到相应的操作组的信息和位向量信息。操作组可以指示由元数据处理域类似地处理的关联的操作码的组。位矢量信息可以指示是否与处理操作码关联地使用相对于元数据处理域的特定输入和输出。可以使用小于可允许操作码位的最大数目的操作码位的第一部分来索引该表，且最大数目可以指示指令集的操作码的位数的上限。多个控制状态寄存器的第一部分可以包括扩展的操作码控制状态寄存器，该扩展的操作码控制状态寄存器包括用于当前指令的附加操作码位(如果有的话)，其中，当前指令可以包括在具有可变长度操作码的指令集中，且其中，指令集的每个操作码可以可选地包括附加操作码位，且扩展的操作码控制状态寄存器包括用于当前指令的附加操作码位(如果有的话)。对于使用该表映射的每个操作码，存在与所述每个操作码对应的结果位向量，该结果位向量可以指示以用于元数据处理的所述每个操作码使用扩展的操作码控制状态寄存器的附加操作码位的什么部分(如果有的话)。当前指令可以是与单个元数据标记相关联的存储器的单个字中存储的多个指令之

一，且所述单个元数据标记可以与单个字中包括的多个指令相关联。多个控制状态寄存器可以包括指示单个字中存储的多个指令中的哪个是当前指令的子指令控制状态寄存器。单个元数据标记可以是指向包括用于单个字中的多个指令中的每一个的不同元数据标记的第一存储器位置的第一指针。至少用于多个指令的第一指令的第一存储器位置中存储的第一元数据标记可以包括指向包括用于第一指令的元数据标记信息的第二存储器位置的第二指针。用于第一指令的元数据标记信息可以包括复杂结构。该复杂结构可以包括至少一个标量数据字段和指向第三存储器位置的至少一个指针字段。

[0008] 根据在这里的技术的另一方面，是在其上包括代码的非瞬时计算机可读介质，该代码当执行时实现处理指令的方法，包括：对于元数据处理，接收具有关联的元数据标记的当前指令，所述元数据处理在与包括当前指令的代码执行域相隔离的元数据处理域中执行；在元数据处理域中和根据元数据标记和当前指令，确定规则是否存在于当前指令的规则高速缓存中，所述规则高速缓存包括关于由所述元数据处理使用以定义允许的操作的元数据的规则；和响应于确定没有规则存在于用于当前指令的规则高速缓存中，在元数据处理域中执行规则高速缓存未命中处理，包括：确定是否允许当前指令的执行；响应于确定允许在代码执行域中执行当前指令，生成用于当前指令的新规则；写入到寄存器；和响应于写入到寄存器，将新规则插入到规则高速缓存中。

[0009] 根据在这里的技术的另一方面，是系统，包括：处理器；和存储器，包括在其上存储的代码，该代码当由处理器执行时实现处理指令的方法，包括：对于元数据处理，接收具有关联的元数据标记的当前指令，所述元数据处理在与包括当前指令的代码执行域相隔离的元数据处理域中执行；在元数据处理域中和根据元数据标记和当前指令，确定规则是否存在于当前指令的规则高速缓存中，所述规则高速缓存包括关于由所述元数据处理使用以定义允许的操作的元数据的规则；和响应于确定没有规则存在于用于当前指令的规则高速缓存中，在元数据处理域中执行规则高速缓存未命中处理，包括：确定是否允许当前指令的执行；响应于确定允许在代码执行域中执行当前指令，生成用于当前指令的新规则；写入到寄存器；和响应于写入到寄存器，将新规则插入到规则高速缓存中。处理器可以是精简指令集计算架构中的流水线处理器。

[0010] 根据在这里的技术的另一方面，是处理指令的方法，包括：接收用于在与包括当前指令的代码执行域相隔离的元数据处理域中执行的元数据处理的当前指令；和通过与用于当前指令的元数据相关的元数据处理域，根据一个或多个策略的集合来确定是否允许当前指令的执行，其中，当前指令访问第一例程的堆栈帧的第一位置，其中，当前指令和堆栈帧的位置具有关联的元数据标记，且一个或多个策略的集合包括提供堆栈保护和防止对包括第一例程的堆栈帧的存储位置的堆栈存储位置的非法(improper)访问的堆栈保护策略。堆栈保护策略可以包括访问第一例程的堆栈帧的第一位置的当前指令的元数据处理中使用的第一规则。如果第一位置具有指示其是第一例程的堆栈位置的元数据且当前指令包括在第一例程中，则第一规则可以允许当前指令的执行。当前指令可以由第一例程的特定起用(invocation)实例使用，且其中，堆栈保护策略可以包括在当前指令的元数据处理中使用的第一规则。如果当前指令包括在第一例程中且也由第一例程的特定起用实例使用，则第一规则可以允许当前指令的执行。第一规则可以包括检查与程序计数器相关联和指示任意授权和能力(capability)的元数据以确定是否允许第一例程的特定起用实例的当前指令

的执行。堆栈保护策略可以提供其中单个堆栈帧中的不同对象具有不同颜色的元数据标记的对象级别保护和用于其中单个堆栈帧的多个子对象中的每一个具有不同元数据标记的包括多个子对象的分层对象的分层对象保护中的任意一个。该方法可以包括创建用于新例程调用的新堆栈帧;和根据严格对象初始化或者懒惰对象着色来标记或者着色新堆栈帧的存储器位置,其中,严格对象初始化包括执行初始化处理,该初始化处理执行触发在将信息存储到新堆栈帧之前最初标记新堆栈帧的每个存储器位置的一个或多个规则的元数据处理的一个或多个指令,且其中,懒惰对象着色结合响应于将数据存储到特定存储器位置的指令而触发的规则的元数据处理来标记新堆栈帧的特定存储器位置。该一个或多个策略可以包括规则的集合,用于推行保证仅当在特定调用之后做出时到特定返回位置的返回是有效的动态控制流量完整性策略。第一位置可以包括对包括返回指令的调用例程的调用指令转移控制,且第二位置可以包括第二指令,其中所述第二位置可以指示作为执行调用例程的返回指令的结果,控制被转移到的返回目标位置。该方法可以包括以第一代码标记来标记包括调用指令的第一位置;以第二代码标记来标记指示返回目标位置的第二位置;执行用于以第一代码标记来标记的调用指令的集合的第一规则的元数据处理,其中,用于以第一代码标记来标记的调用指令的第一规则的元数据处理包括以指示返回地址寄存器包括用于第二位置的有效返回地址的有效返回地址标记来标记返回地址寄存器,其中,调用指令的执行更新关于返回地址寄存器的标记以指示返回到第二位置的能力;执行用于调用例程的返回指令的集合的第二规则的元数据处理,如果以有效返回地址能力标记来标记返回地址寄存器,则该调用例程允许返回指令的执行以将控制转移到返回地址寄存器中存储的返回地址,其中,第二规则将返回地址寄存器的有效返回地址性能标记传播到用于在返回指令的运行时间执行之后的下一指令的程序计数器标记;和执行用于在返回指令的运行时间执行之后的用于第二指令的集合的第三规则的元数据处理,其中,如果第二指令具有等于第二代码标记的代码标记且如果程序计数器标记是有效返回地址能力标记,则该第三规则的元数据处理允许第二指令的执行,其中,第三规则清除用于在第二指令的运行时间执行之后的下一指令的程序计数器标记。

[0011] 根据在这里的技术的另一方面,是处理指令的方法,包括:接收用于在与包括当前指令的代码执行域隔离的元数据处理域中执行的元数据处理的当前指令;和通过与用于当前指令的元数据相关的元数据处理域,根据一个或多个策略的集合来确定是否允许当前指令的执行,其中,该一个或多个策略包括推行从完全序列的第一指令到完全序列的最后指令的以特定次序的指令的完全序列的执行的规则的集合。该方法可以包括将第一共享物理页映射到第一进程(process,处理)的第一虚拟地址空间中;和将第一共享物理页映射到用于第二进程的第二虚拟地址空间中,所述第一共享物理页包括多个存储器位置,其中,多个存储器位置中的每一个与元数据处理域中的规则处理关联地使用的多个全局元数据标记之一相关联。多个全局元数据标记可以指示由至少包括第一进程和第二进程的多个进程共享的元数据标记的集合,且其中,可以对于第一进程和第二进程两者由元数据处理域推行相同策略。由元数据处理域推行相同策略可以使用元数据来允许第一进程执行操作,该操作在另外的情况(otherwise)就不被用于第二进程的相同策略所允许,且其中,程序计数器可以具有关联的程序计数器标记,且关联的程序计数器标记的不同值可以由相同策略的规则使用以允许第一进程执行所述操作,该操作在另外的情况就不被用于第二进程的相同策

略所允许。该方法可以包括由应用(application)的分配例程来执行第一处理以使用应用的当前颜色生成用于应用的下一颜色，其中，应用的当前颜色指示用于应用的应用专用(application-specific)颜色序列的当前状态，下一颜色指示用于应用的应用专用颜色序列的下一状态，且当前颜色存储在第一原子单元(atom)上的第一元数据标记中。第一处理可以包括执行第一组一个或多个指令，其中，第一组一个或多个指令触发由元数据处理域使用一个或多个规则的元数据处理，其中，由元数据处理域使用一个或多个规则的元数据处理使用当前颜色生成下一颜色，且通过在第一原子单元的第一元数据标记中存储下一颜色来更新应用的应用专用颜色序列的当前状态。第一组一个或多个指令可以包括在应用的分配例程中，且第一原子单元可以是任意的寄存器和存储器位置。应用专用颜色序列可以是可用于由应用使用的不同颜色的无限制(unbounded)序列，且下一颜色可以存储为由应用使用的一个或多个存储器位置中的每一个的标记值，其中，一个或多个存储器位置可以由分配例程分配。规则的集合可以包括第一规则和第二规则，且其中，指令的完全序列可以包括第一指令和第二指令，且其中，第二指令可以在第一指令之后立即执行。该方法可以包括执行用于第一指令的第一规则的元数据处理，其中，第一规则的元数据处理包括将用于第一指令的运行时间执行之后的下一指令的程序计数器的程序计数器标记设置为特殊标记值;和执行用于第二指令的第二规则的元数据处理，其中，第二规则的元数据处理包括保证仅当用于第二指令的程序计数器的程序计数器标记等于特殊标记时允许第二指令的执行。

[0012] 根据本发明的另一方面，是包括在其上存储的代码的非瞬时计算机可读介质，所述代码当执行时实现处理指令的方法，包括:接收用于在与包括当前指令的代码执行域相隔离的元数据处理域中执行的元数据处理的当前指令;和通过与用于当前指令的元数据相关的元数据处理域，根据一个或多个策略的集合来确定是否允许当前指令的执行，其中，当前指令访问第一例程的堆栈帧的第一位置，其中，当前指令和堆栈帧的位置具有关联的元数据标记，且一个或多个策略的集合包括提供堆栈保护和防止对包括第一例程的堆栈帧的存储位置的堆栈存储位置的非法访问的堆栈保护策略。

[0013] 根据在这里的技术的另一方面，是一种系统，包括:处理器;和包括在其上存储的代码的存储器，所述代码当由处理器执行时实现处理指令的方法，包括:接收用于在与包括当前指令的代码执行域相隔离的元数据处理域中执行的元数据处理的当前指令;和通过与用于当前指令的元数据相关的元数据处理域，根据一个或多个策略的集合来确定是否允许当前指令的执行，其中，当前指令访问第一例程的堆栈帧的第一位置，其中，当前指令和堆栈帧的位置具有关联的元数据标记，且一个或多个策略的集合包括提供堆栈保护和防止对包括第一例程的堆栈帧的存储位置的堆栈存储位置的非法访问的堆栈保护策略。

[0014] 根据在这里的技术的另一方面，是包括在其上存储的代码的非瞬时计算机可读介质，所述代码当执行时实现处理指令的方法，包括:接收用于在与包括当前指令的代码执行域相隔离的元数据处理域中执行的元数据处理的当前指令;和通过与用于当前指令的元数据相关的元数据处理域，根据一个或多个策略的集合来确定是否允许当前指令的执行，其中，该一个或多个策略包括推行从完全序列的第一指令到完全序列的最后指令的以特定次序的指令的完全序列的执行的规则的集合。

[0015] 根据在这里的技术的另一方面，是一种系统，包括:处理器;和包括在其上存储的

代码的存储器，所述代码当由处理器执行时实现处理指令的方法，包括：接收用于在与包括当前指令的代码执行域相隔离的元数据处理域中执行的元数据处理的当前指令；和通过与用于当前指令的元数据相关的元数据处理域，根据一个或多个策略的集合来确定是否允许当前指令的执行，其中，该一个或多个策略包括推行从完全序列的第一指令到完全序列的最后指令的以特定次序的指令的完全序列的执行的规则的集合。

[0016] 根据在这里的技术的另一方面，是生成和使用元数据标记的方法，包括：在与代码执行域相隔离的元数据处理域中使用的多个指定寄存器的第一指定寄存器中存储自举标记；和执行第一处理以从自举标记推导出一个或多个附加元数据标记，其中，所述第一处理包括执行触发元数据处理域中的一个或多个规则的元数据处理的代码执行域中的一个或多个指令。自举标记可以用作从其推导出由元数据处理域使用的所有其他元数据标记的初始种子标记。自举标记可以硬布线或者存储在只读存储器的一部分中。存储和第一处理可以包括在通过当引导(boot)包括元数据处理域和代码执行域的系统时执行自举程序的第一代码部分而执行的处理中。该方法可以包括从第一指定寄存器中存储的自举标记推导出默认标记；在多个指定寄存器的第二指定寄存器中存储默认标记；和执行触发从第二指定寄存器写入默认标记作为用于由代码执行域使用的多个存储器位置中的每一个的元数据标记的元数据处理域中的规则的元数据处理的指令序列。第一处理可以包括生成从自举标记推导出的元数据标记的初始集合，其中，初始集合的每一个元数据标记可以通过执行触发元数据处理域中的规则高速缓存未命中处理的代码执行域中的当前指令而生成，由此没有规则存在于当前指令的规则高速缓存中，规则高速缓存包括关于由元数据处理域使用以定义允许操作的元数据的规则。规则高速缓存未命中处理可以包括由元数据处理域中执行的规则高速缓存未命中处理器(handler,处理程序)来计算当前指令的新规则，其中，该新规则包括元数据标记的初始集合的结果元数据标记。初始集合的每个元数据标记可以是可以进一步用于推导出其他元数据标记的标记发生器。一个或多个指定指令的第一集合的执行可以触发规则和生成每个元数据标记的元数据处理域中的规则高速缓存未命中处理，该每个元数据标记指示为用于生成一个或多个其他元数据标记的序列的标记发生器，且其中，一个或多个指定指令的第二集合的执行可以触发规则和生成每个元数据标记的元数据处理域中的规则高速缓存未命中处理，该每个元数据标记指示为不能用于进一步生成附加元数据标记的非生成标记。自举程序可以进一步包括触发在元数据处理域中的处理的规则的指令，该规则写入关于指定代码部分的一个或多个指令的一个或多个特殊元数据代码标记以向标记的一个或多个指令提供扩展的特权、能力或者授权。指定代码部分可以包括一个或多个内核程序(kernel)代码和加载器(loader)代码。一个或多个特殊元数据代码标记从元数据标记的初始集合的第一元数据标记推导出，其中，第一元数据标记是特殊指令标记发生器。元数据标记的初始集合可以包括以下的任何一个或多个：作为用于生成用于标记指令的一个或多个代码标记的序列的标记发生器的初始指令元数据标记；作为用于生成一个或多个其他存储分配(malloc)标记发生器的序列的标记发生器的初始存储分配元数据标记，其中，一个或多个其他存储分配标记发生器中的每一个用于生成用于不同应用的一个或多个其他元数据标记的序列，其关于着色任意分配的存储器单元(memory cell)和指向由该不同应用使用的存储器单元的指针；作为用于生成一个或多个其他控制流完整性标记发生器的序列的标记发生器的起始控制流完整性标记，其中，一个或多个其他控制流

完整性标记发生器中的每一个用于生成用于不同应用的一个或多个其他元数据标记的序列，其关于标记该不同应用的控制转移目标；和作为用于生成一个或多个其他污点标记发生器的序列的标记发生器的初始污点标记，其中，一个或多个其他污点标记发生器中的每一个用于生成用于不同应用的一个或多个其他元数据污点标记的序列，其关于用元数据污点标记来标记由该不同应用使用的的数据项，其中元数据污点标记基于产生了或者修改了这些数据项的代码。可以通过执行触发元数据处理域中的规则的其他处理的指令来生成元数据标记的序列。其他处理可以包括使用序列中的当前元数据标记来生成序列中的下一元数据标记，其中，当前元数据标记指示序列的当前状态且存储为与原子单元相关联的元数据标记，其中，原子单元是任意的寄存器或者存储器位置；和通过保存下一元数据标记为与原子单元相关联的元数据标记而更新序列的当前状态。

[0017] 根据在这里的技术的另一方面，是获得用于应用的控制流信息的方法，包括：执行加载用于由处理器的执行的应用的加载器，其中，所述执行加载器包括执行包括触发元数据处理域中的一个或多个规则的第一集合的元数据处理的一个或多个指令的第一代码部分，其中，所述一个或多个规则的第一集合的元数据处理包括收集和存储用于应用的控制流信息作为对元数据处理域可访问和对代码执行域不可访问的应用元数据；和执行代码执行域中的应用的指令，其中，所述执行所述应用的指令触发使用控制流信息的至少一部分以确定是否允许应用中的控制从第一源位置转移到第一目标位置的控制流策略的规则的第二集合的元数据处理。第一目标位置可以具有允许将控制转移到第一目标位置的一个或多个可允许源位置的集合。收集和存储用于应用的控制流信息为应用元数据可以进一步包括执行其他处理的元数据处理域。其他处理可以包括以标识一个或多个可允许源位置的集合的第一元数据来标记第一目标位置，其中，第一元数据存储为应用元数据的控制流信息的一部分。应用的第一指令可以将控制从第一源位置转移到第一目标位置，且第一指令可以触发使用第一元数据以通过确定第一源位置是否包括在允许将控制转移到第一目标位置的一个或多个可允许源位置的集合中而确定是否允许第一指令的执行的控制流策略的一个或多个规则的元数据处理。其他处理还可以包括以唯一源元数据标记来标记集合的每个可允许源位置。每个可允许源位置的每个唯一源元数据标记可以包括在应用的源元数据标记的第一序列中，其中，第一序列可以是从控制流发生器标记生成的源元数据标记的唯一序列。控制流发生器标记可以从由初始自举标记推导出的初始控制流发生器标记生成。初始控制流发生器标记可以用于生成多个附加控制流发生器标记，且其中，每一个附加控制流发生器标记可以用于生成用于不同应用的唯一源元数据标记的序列。

[0018] 根据在这里的技术的另一方面，是包括在其上存储的代码的非瞬时计算机可读介质，所述代码当执行时实现生成和使用元数据标记的方法，包括：在与代码执行域相隔离的元数据处理域中使用的多个指定寄存器的第一指定寄存器中存储自举标记；和执行第一处理以从自举标记推导出一个或多个附加元数据标记，其中，所述第一处理包括执行触发元数据处理域中的一个或多个规则的元数据处理的代码执行域中的一个或多个指令。

[0019] 根据在这里的技术的另一方面，是一种系统，包括：处理器；和包括在其上存储的代码的存储器，所述代码当执行时实现生成和使用元数据标记的方法，包括：在与代码执行域相隔离的元数据处理域中使用的多个指定寄存器的第一指定寄存器中存储自举标记；和执行第一处理以从自举标记推导出一个或多个附加元数据标记，其中，所述第一处理包括

执行触发元数据处理域中的一个或多个规则的元数据处理的代码执行域中的一个或多个指令。

[0020] 根据在这里的技术的另一方面，是包括在其上存储的代码的非瞬时计算机可读介质，所述代码当执行时，执行获得用于应用的控制流信息的方法，包括：执行加载用于由处理器的执行的应用的加载器，其中，所述执行加载器包括执行包括触发元数据处理域中的一个或多个规则的第一集合的元数据处理的一个或多个指令的第一代码部分，其中，所述一个或多个规则的第一集合的元数据处理包括收集和存储用于应用的控制流信息为对元数据处理域可访问和对代码执行域不可访问的应用元数据；和执行代码执行域中的应用的指令，其中，所述执行所述应用的指令触发使用控制流信息的至少一部分以确定是否允许应用中的控制从第一源位置转移到第一目标位置的控制流策略的规则的第二集合的元数据处理。

[0021] 根据在这里的技术的另一方面，是一种系统，包括：处理器；和包括在其上存储的代码的存储器，所述代码当执行时，实现获得用于应用的控制流信息的方法，包括：执行加载用于由处理器执行的应用的加载器，其中，所述执行加载器包括执行包括触发元数据处理域中的一个或多个规则的第一集合的元数据处理的一个或多个指令的第一代码部分，其中，所述一个或多个规则的第一集合的元数据处理包括收集和存储用于应用的控制流信息为对元数据处理域可访问和对代码执行域不可访问的应用元数据；和执行代码执行域中的应用的指令，其中，所述执行所述应用的指令触发使用控制流信息的至少一部分以确定是否允许应用中的控制从第一源位置转移到第一目标位置的控制流策略的规则的第二集合的元数据处理。

[0022] 根据在这里的技术的另一方面，是用于在标记和未标记数据源之间执行处理器调解(processor-mediated)数据转移的方法，包括：在处理器上执行从未标记数据源加载第一数据的第一指令，所述未标记数据源包括不具有关联的元数据标记的存储器位置；由第一硬件以指示第一数据不可信并且来自公开数据源的第一元数据标记来标记第一数据，其中，具有第一元数据标记的第一数据存储在第一缓存中；和在处理器上执行触发使用第一组一个或多个规则的元数据处理的第一代码，其中，使用第一组一个或多个规则的元数据处理来执行重新标记，所述重新标记重新标记第一数据以具有第二元数据标记，指示第一数据是可信的。第二元数据标记可以另外指示第一数据来自公共源。具有第二元数据标记的第一数据可以存储在作为包括每个具有关联的元数据标记的存储器位置的标记的数据源的存储器中。存储器可以是包括来自一个或多个可信数据源的数据的可信存储器。元数据处理可以在与包括第一代码的代码执行域相隔离的元数据处理域中执行。第一组一个或多个规则可以是关于由元数据处理使用以定义允许操作的元数据的规则。第一代码可以包括一个或多个指令，且该一个或多个指令中的每一个可以具有指示所述每个指令具有起用重新标记第一数据以具有第二元数据标记的一个或多个规则的授权的特殊指令标记。具有第一元数据标记的第一数据可以加密，且该方法可以包括：通过在处理器上执行一个或多个指令来解密具有第一元数据标记的第一数据并生成具有第一元数据标记的第一数据的解密形式；和通过在处理器上执行一个或多个附加指令来执行验证(validation)处理，所述验证处理使用数字签名以保证第一数据的解密形式有效，其中，所述重新标记在第一数据的成功验证处理之后执行。具有第二元数据标记的第一数据可以以解密形式存储在标记

存储器的第一存储器位置，且该方法包括加密第一数据以产生以加密形式的第一数据并根据第一数据生成数字签名，其中，所述加密和所述生成通过在处理器上执行附加代码来执行；和在处理器上执行将第一数据的加密形式从标记存储器的第一存储器位置存储到未标记存储器的目的地位置的第二指令，其中，第一数据的加密形式存储在目的地位置中而没有关联的元数据标记，且其中，在目的地位置中存储第一数据的加密形式之前由第二硬件除去第二元数据标记。在第一时间点，第一数据可以存储在未标记存储器部分的第一位置中，且在第二时间点，具有指示第一数据不可信并且来自公开数据源的第一元数据标记的第一数据可以存储在标记存储器部分的第二位置中。未标记存储器部分和所述标记存储器部分可以包括在由同一存储器控制器服务的同一存储器中，且其中，第二元数据处理规则可以仅允许处理器执行将具有指示数据是公共的关联的元数据标记的数据写入到未标记存储器部分的操作，且其中，可以仅允许来自外部未标记源的对未标记数据操作的直接存储器操作来访问同一存储器的未标记存储器部分。第二元数据处理规则的至少一部分可以进一步仅允许处理器执行将具有指示数据是公共的且另外不可信的关联的元数据标记的数据写入到未标记存储器部分的操作。未标记数据源可以连接到仅包括未标记数据源的第一互连结构(interconnect fabric)，其中，具有第二元数据标记的第一数据可以存储在连接到仅包括标记数据源的第二互连结构的存储器的位置中。第二处理器可以连接到第一互连结构且可以使用来自未标记数据源的未标记数据来执行其他指令。可以执行其他指令而不执行元数据处理和不使用关于元数据的规则来推行可允许操作，其中，由所述第二处理器执行所述其他指令可以包括执行包括以下任意的一个或多个操作：从第一互连结构的未标记数据源读取数据，和将数据写入到第一互连结构的未标记数据源。

[0023] 根据在这里的技术的另一方面，是一种系统，包括：处理器；和一个或多个标记存储器，其中，一个或多个标记存储器的每个存储器位置具有关联的元数据标记；包括第一未标记存储器的一个或多个未标记存储器，其中，一个或多个未标记存储器的存储器位置不具有关联的元数据标记；包括关于在执行元数据处理以定义关于指令的允许操作时使用的元数据的规则的规则高速缓存，其中，在由处理器执行当前指令之前，执行使用规则高速缓存的一个或多个规则的元数据处理以确定是否允许当前指令的执行；第一指令，当由处理器执行时，将第一数据从第一未标记存储器加载到由处理器使用的数据高速缓存中，其中，数据高速缓存中存储的第一数据具有关联的第一元数据标记；第二指令，当由处理器执行时，将第二数据从数据高速缓存存储到第一未标记存储器，其中，数据高速缓存中存储的第二数据具有关联的第二元数据标记；第一硬件部件，将未标记数据转换为由处理器在系统中使用的标记数据，其中，响应于第一指令的执行，第一硬件部件从第一未标记存储器接收没有任何关联的元数据标记的第一数据，并输出具有关联的第一元数据标记的第一数据；和第二硬件部件，将标记数据转换为未标记数据，其中，响应于第二指令的执行，第二硬件部件接收具有关联的第二元数据标记的第二数据并输出没有任何关联的元数据标记的第二数据。可以加密没有任何关联的元数据标记的第一数据，且第一硬件部件可以将第一数据转换为解密形式，可以使用数字签名执行第一数据的验证处理，且在成功的验证处理时，可以标记第一数据为具有指示第一数据被可信的关联的第一元数据标记。具有第二关联的元数据标记的第二数据可以以解密形式，且第二硬件部件可以将第二数据转换为加密形式并根据第二数据生成数字签名。第一硬件部件可以标记第一数据为具有指示第一数据被可

信且还标识第一数据来自公共源的关联的第一元数据标记。一个或多个密码密钥集合可以是以硬件编码的和存储在存储器中的任何方式。一个或多个密码密钥集合可以由第一硬件部件用于执行解密和验证处理，且可以由第二硬件部件用于执行加密和创建数字签名。第一数据可以标识由第一硬件部件使用以解密第一数据的密码密钥集合中特定的一个，且其中，第二数据的关联的第二元数据标记可以标识由第二硬件部件使用以加密和签名第二数据的密码密钥集合中特定的一个。

[0024] 根据在这里的技术的另一方面，是处理当前指令的方法，包括：对于元数据处理接收当前指令；和在与包括当前指令的代码执行域相隔离的元数据处理域中执行用于当前指令的元数据处理，所述当前指令参考具有在元数据处理中使用的第一元数据标记的第一存储器位置，所述用于当前指令的元数据处理包括：执行处理以从存储器提取第一元数据标记；在从存储器接收用于第一存储器位置的第一元数据标记之前，确定第一存储器位置的第一元数据标记的预测值；使用第一存储器位置的第一元数据标记的预测值，确定当前指令的结果操作数的第一结果元数据标记；和从存储器接收第一元数据标记；确定第一元数据标记是否匹配第一元数据标记的预测值；和响应于确定第一元数据标记匹配第一元数据标记的预测值，使用第一结果元数据标记作为结果操作数的最终结果元数据标记。用于当前指令的元数据处理可以包括根据当前指令和用于当前指令的输入元数据标记的集合来确定当前指令的第一规则，其中，所述第一规则包括第一存储器位置的第一元数据标记的预测值且包括第一结果元数据标记，所述第一规则包括在用于元数据处理域中的元数据处理的规则高速缓存中；和响应于确定第一元数据标记不匹配第一元数据标记的预测值，执行用于当前指令的元数据处理域中的规则高速缓存未命中处理。用于当前指令的元数据处理域中的规则高速缓存未命中处理可以包括确定是否允许代码执行域中的当前指令的执行；响应于确定允许代码执行域中的当前指令的执行，生成当前指令的新规则，其中，根据当前指令、输入元数据标记的集合和第一元数据标记生成所述新规则；和将新规则插入到用于元数据处理域中的元数据处理的规则高速缓存中。其他输入元数据标记的集合可以包括用于当前指令的多个其他元数据标记，其中所述其他元数据输入标记的集合可以包括用于以下任意的元数据标记：程序计数器、当前指令和当前指令的输入操作数。结果操作数可以是存储执行当前指令的结果的目的存储器位置或者目的地寄存器。可以根据包括第一级和第二级的多个级来处理指令，其中，第一级可以在第二级之前发生。可以在第一级中确定第一存储器位置的第一元数据标记的预测值，且第二级可以包括执行所述确定第一元数据标记是否匹配第一元数据标记的预测值，且第二级还可以包括响应于确定第一元数据标记不匹配第一元数据标记的预测值而执行所述用于当前指令的元数据处理域中的规则高速缓存未命中处理。规则高速缓存可配置为根据预测选择模式以预测模式或者正常处理模式操作。规则高速缓存可配置为当执行用于当前指令的所述元数据处理时以预测模式操作。当规则高速缓存配置为以所述预测模式操作时，规则高速缓存可以根据第一规则生成第一输出。第一输出可以包括用于下一指令的程序计数器的元数据标记，用于当前指令的结果操作数的第一结果元数据标记，和第一元数据标记的预测值作为第一级的输出。当规则高速缓存配置为以所述正常处理模式操作时，规则高速缓存可以根据与第一规则不同的第二规则生成第二输出，其中，第二输出可以不包括第一元数据标记的预测值，且第二输出可以包括用于当前指令的结果操作数和用于下一指令的程序计数器的元数据标记。规则高速缓

存可以在以预测模式操作时使用第一策略的规则的第一版本和否则在以正常处理模式操作时可以使用第一策略的规则的第二版本，且其中，第一规则可以包括在规则的第一版本中且第二规则可以包括在规则的第二版本中。

[0025] 根据在这里的技术的另一方面，是一种系统，包括：包括多个流水线级的流水线处理器，所述多个级包括存储器级和写回级；用于在存储器级的完成之前操作存储器级的集成的元数据处理的可编程单元(PUMP)，其中，PUMP执行参考具有在元数据处理中使用的第一元数据标记的第一存储器位置的用于当前指令的元数据处理，其中，PUMP接收包括用于当前指令的第一元数据标记的第一输入，再其中，PUMP生成作为到写回级的输入提供的第一输出，该第一输出包括第一存储器位置的第一元数据标记的预测值和用于当前指令的结果操作数的第一结果元数据标记，其中，第一结果元数据标记由PUMP根据用于第一存储器位置的第一元数据标记的预测值确定；且所述写回级的硬件部件确定用于第一存储器位置的第一元数据标记是否匹配第一元数据标记的预测值，并且当第一元数据标记匹配第一元数据标记的预测值时使用第一结果元数据标记作为用于结果操作数的最终结果元数据标记。PUMP可以是与存储器级同时操作且进一步以预测模式操作，且可以确定第一存储器位置的第一元数据标记的预测值的第一PUMP，且其中，该系统可以包括以正常、非预测模式操作，且可以不确定第一存储器位置的第一元数据标记的任何预测值的第二PUMP。第二PUMP可以集成为存储器级和写回级之间的另一级。第一PUMP可以使用第一策略的规则的第一版本用于当以预测模式操作时使用，且第二PUMP可以使用第一策略的规则的第二版本用于当以正常、非预测模式操作时使用。第一PUMP可以根据来自第一版本的第一规则确定第一输出，且第二PUMP可以根据来自第二版本的第二规则确定第二输出。第二输出可以包括用于第一存储器位置的第二结果元数据标记且所述第二输出可以作为到写回级的输入而提供。写回级的硬件部件可以在第一元数据标记不匹配预测值时另外使用第二结果元数据标记作为用于结果操作数的最后结果元数据标记。

[0026] 根据在这里的技术的另一方面，是包括在其上存储的代码的非瞬时计算机可读介质，所述代码当执行时实现在标记和未标记数据源之间的处理器调解数据转移的方法，包括：在处理器上执行从未标记数据源加载第一数据的第一指令，所述未标记数据源包括不具有关联的元数据标记的存储器位置；由第一硬件以指示第一数据不可信和来自公开数据源的第一元数据标记来标记第一数据，其中，具有第一元数据标记的第一数据存储在第一缓存中；和在处理器上执行触发使用第一组一个或多个规则的元数据处理的第一代码，其中，使用第一组一个或多个规则的元数据处理执行重新标记，所述重新标记重新标记第一数据以具有第二元数据标记，指示第一数据是可信的。

[0027] 根据在这里的技术的另一方面，是包括在其上存储的代码的非瞬时计算机可读介质，所述代码当执行时实现处理当前指令的方法，包括：对于元数据处理接收当前指令；和在与包括当前指令的代码执行域相隔离的元数据处理域中执行用于当前指令的元数据处理，所述当前指令参考具有在元数据处理中使用的第一元数据标记的第一存储器位置，所述用于当前指令的元数据处理包括：执行处理以从存储器提取第一元数据标记；在从存储器接收用于第一存储器位置的第一元数据标记之前，确定第一存储器位置的第一元数据标记的预测值；使用第一存储器位置的第一元数据标记的预测值，确定当前指令的结果操作数的第一结果元数据标记；和从存储器接收第一元数据标记；确定第一元数据标记是否匹

配第一元数据标记的预测值;和响应于确定第一元数据标记匹配第一元数据标记的预测值,使用第一结果元数据标记作为结果操作数的最终结果元数据标记。

附图说明

[0028] 在这里的技术的特征和优点将从以下结合附图进行的其示例性实施例的详细说明变得更明显,在附图中:

[0029] 图1是示出了集成为处理器流水线中的流水线级的PUMP高速缓存的实例的示意图;

[0030] 图2是示出了PUMP评估框架的示意图;

[0031] 图3A是示出了具有使用图2示出的评估框架的简单实现的单个运行时间策略的性能结果的图;

[0032] 图3B是示出了具有简单实现的单个能量策略的性能结果的图;

[0033] 图4A是示出了具有64b标记的简单实现的合成策略运行时间开销的一系列条形图,其中,合成策略同时推行以下策略(i)空间和时间存储器安全,(ii)污点跟踪,(iii)控制流完整性和(iv)代码和数据分离;

[0034] 图4B是示出了具有64b标记的简单实现的合成策略能量开销的一系列条形图;

[0035] 图4C是示出了与基线相比具有简单实现的功率上限的一系列条形图;

[0036] 图5A是没有操作组优化和具有操作组优化的PUMP规则的数目的比较条形图;

[0037] 图5B是示出了基于PUMP性能的不同操作组优化的未命中比率的影响的一系列图;

[0038] 图6A是用于具有复合策略的gcc基准点的每个DRAM转移的唯一标记的分布的图,示出了大多数的字具有相同标记;

[0039] 图6B是示出了主存储器标记压缩的图;

[0040] 图7A是示出了16b L2标记和12b LI标记之间的转化的示意性图;

[0041] 图7B是示出了12b L1标记和16b L2标记之间的转化的示意性图;

[0042] 图8A是示出了L1标记长度关于L1 PUMP清空(flush)(log10)的影响的示意性图;

[0043] 图8B是示出了L1标记长度关于L1 PUMP未命中比率的影响的示意性图;

[0044] 图9A是示出了不同策略的未命中比率的一系列条形图;

[0045] 图9B是示出四个示例性微架构优化的高速缓存命中比率的线图;

[0046] 图9C是示出未命中服务性能的线图;

[0047] 图9D是示出基于性能的未命中处理器命中率的线图;

[0048] 图9E是示出用于复合策略的优化的影响的一系列条形图;

[0049] 图10A是示出了优化实现的运行时间开销的一系列图;

[0050] 图10B是示出了优化实现的能量开销的一系列条形图;

[0051] 图10C是示出了与基线相比的优化实现的绝对功率的一系列条形图;

[0052] 图11A是示出不同代表性基准点的标记位长和UCP-高速缓存($)性能的运行时间开销影响的一系列阴影图;

[0053] 图11B是示出不同代表性基准点的标记位长和UCP-$性能的能量开销影响的一系列阴影图;

[0054] 图12A是示出了关于代表性的基准点的优化的运行时间影响的一系列图,其中A:

简单;B:A+操作分组;C:B+DRAM压缩;D:C+(10b LI,14b,L2)短标记;E:D+(2048-UCP;512-CTAG));

[0055] 图12B是示出了关于代表性的基准点的优化的能量影响的一系列图,其中A:简单的;B:A+操作分组;C:B+DRAM压缩;D:C+(10b LI,14b,L2)短标记;E:D+(2048-UCP;512-CTAG));

[0056] 图13A是示出了在代表性基准点的组成中的运行时间策略影响的一系列图;

[0057] 图13B是示出了在组成中的能源策略影响的一系列图;

[0058] 图14是提供研究的策略的概况的标为“表1”的第一表;

[0059] 图15是提供标记方案的分类法的概况的标为“表2”的第二表;

[0060] 图16是提供对于基线和简单PUMP扩展处理器的存储器资源估计的概况的标为“表3”的第三表;

[0061] 图17是提供实验中使用的PUMP参数范围的概况的标为“表4”的第四表;

[0062] 图18是提供PUMP优化处理器的存储器资源估计的概况的标为“表5”的第五表;

[0063] 图19是提供污点跟踪错误处理器的概况的标为“算法1”的第一算法;

[0064] 图20是提供N策略未命中处理器的概况的标为“算法2”的第二算法;

[0065] 图21是提供具有HW支持的N策略未命中处理器的概括的标为“算法3”的第三算法;

[0066] 图22是PUMP规则高速缓存数据流和微架构的示意图;

[0067] 图23是PUMP微架构的示意图;

[0068] 图24是与图1类似的示出了集成为处理器流水线中的流水线级的示例性PUMP高速缓存及其操作组转化、UCP和CTAG高速缓存的示意图;

[0069] 图25是在根据在这里的技术的实施例中的控制状态寄存器(CSR)的实例;

[0070] 图26是在根据在这里的技术的实施例中的标记模式的实例;

[0071] 图27是图示在根据在这里的技术的实施例中的具有单独处理器的单独的元数据处理子系统/域的实例;

[0072] 图28图示在根据在这里的技术的实施例中的PUMP输入和输出;

[0073] 图29图示在根据在这里的技术的实施例中的关于操作组表的输入和输出;

[0074] 图30图示在根据在这里的技术的实施例中由PUMP执行的处理;

[0075] 图31和图32提供关于在根据在这里的技术的实施例中的PUMP输入和输出的控制和选择的附加细节;

[0076] 图33是图示在根据在这里的技术的实施例中的6级处理流水流水线的实例;

[0077] 图34到图38是图示在实施例中的子指令和关联技术的实例;

[0078] 图39到图42是图示实施例中的字节级标记和关联技术的实例;

[0079] 图43是图示在根据在这里的技术的实施例中的可变长度操作码的实例;

[0080] 图44是图示在根据在这里的技术的实施例中的操作码映射表的实例;

[0081] 图45是图示在根据在这里的技术的实施例中的共享页的实例;

[0082] 图46是图示在根据在这里的技术的实施例中的控制点转移的实例;

[0083] 图47是图示在根据在这里的技术的实施例中的调用堆栈的实例;

[0084] 图48到图49是图示在根据在这里的技术的实施例中的存储器位置标记或者着色的实例;

[0085] 图50是图示在根据在这里的技术的实施例中的setjmp和longjmp的实例；

[0086] 图51、图52和图53是在根据在这里的技术的实施例中的不同运行时间行为和关联的防止动作和用于实现防止动作的机制的表；

[0087] 图54、图55和图56是图示在根据在这里的技术的实施例中可以执行以学习或者确定策略规则的处理的实例；

[0088] 图57、图58、图59和图60是图示关于数据的外部版本和内部标记版本之间的转换的实施例中的组件的实例；

[0089] 图61、图62和图63是图示在根据在这里的技术的实施例中的执行标记预测的方面的实例；

[0090] 图64到图65图示具有实施例中的分配的存储器的在这里的着色存储器位置技术的使用；

[0091] 图66到图67图示在根据在这里的技术的实施例中的提供硬件规则支持的不同组件；

[0092] 图68到图70是图示PUMP返回值的实施例中在这里的技术的使用的实例；

[0093] 图71是图示具有指令序列的实施例中的在这里的技术的使用的实例；

[0094] 图72是在根据在这里的技术的实施例中可以关于引导(boot)系统执行的处理步骤的流程图；

[0095] 图73是在根据在这里的技术的实施例中关于标记生成的树标记分级的实例；

[0096] 图74、图75、图76和图77是图示在根据在这里的技术的实施例中关于I/O PUMP的方面和特征的实例；

[0097] 图78、图79、图80、图81和图82是图示在根据在这里的技术的实施例中关于存储和确定标记值使用的分级的实例；和

[0098] 图83和图84是图示在根据在这里的技术的实施例中的控制流完整性和关联处理的实例。

具体实施方式

[0099] 以下段落描述将元数据标记UI系统的主存储器、高速缓存和寄存器中的每个字不可分割地关联的用于元数据处理的可编程单元(PUMP)的各种实施例和方面。为支持无限制的元数据，标记足够大以对存储器中的数据结构是间接的。关于每个指令，输入的标记用于确定是否允许操作，且如果允许操作则计算结果的标记。在一些实施例中，以软件定义标记检查和传播规则。但是，为最小化性能影响，这些规则高速缓存在硬件结构中，即PUMP规则高速缓存，其与处理器的运算逻辑单元(ALU)部分并行操作。在一些实施例中，比如可以使用软件和/或硬件实现的未命中处理器可以用于基于当前生效的策略来服务高速缓存未命中。

[0100] 在使用四个不同策略的组成的至少一个实施例中，可以以不同方式强调PUMP来度量(参见图14)PUMP的性能影响和图示安全性属性的范围，例如，(1)使用标记区分来自存储器中的数据的代码和提供相对简单代码注入攻击的保护的非可执行数据和非可写代码(NXD+NWC)策略；(2)检测堆分配存储器中的所有空间和时间侵害的存储器安全策略，以实际上无限的(260)颜色的数目(“污点记号”)扩展；(3)限制仅到程序的控制流图中的允许的

边缘的间接控制转移的控制流完整性(CFI)策略,防止面向返回的编程风格攻击(推行细粒度CFI,不是对攻击潜在地脆弱的粗粒度的近似);和(4)细粒度的污点跟踪策略(一般化),其中每个字可以潜在地同时由多个源(库和IO流)污染。

[0101] 前述是可以在根据在这里的技术的实施例中使用的公知策略的实例。对于这种其保护性能已经在文献中建立的公知策略,在这里的技术可以用于推行这种策略同时还减小使用PUMP推行它们的性能影响。除了NXD+NWC之外,这些策略中的每一个需要区分实质上无限数目的唯一项;相反,具有有限数目的元数据位的解决方案最好也仅可以支持粗略简化的近似。

[0102] 如在这里的其他地方所示和所述的,根据在这里的技术的一个实施例可以利用PUMP的简单的直接实现,其使用指针大小(64b或者字节)的标记到64b字由此至少将系统中的所有存储器的大小和能量使用翻倍。规则高速缓存在此之上添加面积和能量。对于该特定实施例,测量到190%(参见图16)的面积开销和大约220%,的几何平均(geomean)能量开销。此外,在一些应用上运行时间开销可以在300%以上。这种高开销可能阻碍采用,如果它们是可以做到的最好的话。

[0103] 但是,如以下更详细地描述的,大多数的策略展现对于标记和在其上定义的规则两者的空间和时间局部性。因此,根据在这里的技术的实施例可以通过在一组类似的(或甚至相同的)指令上定义唯一规则,减小强制性未命中和增加规则高速缓存的有效容量而显著地减小唯一规则的数目。可以通过利用标记中的空间局部性减小离片(off-chip)存储器业务量。可以通过使用少量位表示在某一时间使用中的指针大小的标记的子集来最小化片上面积和能量开销。可以通过提供用于高速缓存部件策略的硬件支持来减小复合策略未命中处理器的运行时间成本。因此,根据在这里的技术的实施例可以包括这种优化以由此允许PUMP实现较低开销而不牺牲其丰富的策略模型。

[0104] 根据在这里的技术的实施例可以用可用于编码任意数目的安全性策略的元数据来增强存储器字和内部处理器状态,所述安全性策略可以隔离地或者同时地推行。根据在这里的技术的实施例可以通过将与数据流并行工作以推行任意策略集合的元数据处理单元(PUMP)添加到“传统的”处理器(例如,RISC-CPU、GPU、向量处理器等)来实现前述;本公开的技术具体地使得元数据无限制和软件可编程,以使得在这里的技术可以被采用和应用于各种各样的元数据处理策略。例如,PUMP可以集成为传统的(RISC)处理器的新/分开的流水线级,或者可以集成为与“主机”处理器并行地工作的分立硬件。对于前一情况,可能有指令级模拟器、精致策略、实现优化和资源估计,和特性化设计的大范围模拟。

[0105] 尝试以精细(即,指令)粒度级别推行策略的现有解决方案不能推行任意策略集合。通常,仅可以在指令级别推行少量的固定策略。在较高粒度级别(即线程)推行策略不能防止某些类别的面向返回的编程攻击,因此使得该类型的推行在其有用性上有限。相反,根据在这里的技术的实施例允许可以单独地或者同时地在指令级别推行的无限数目的策略的表达(唯一限制是大小地址空间,因为关于可以指向任何任意数据结构的地址指针来表示元数据)。

[0106] 应当注意,在以下段落中描述的各个图图示在这里描述的技术的各个方面的各种实例、方法及其他示例实施例。将理解在这种图中,图示的元素边界(例如,框、框的组或者其他形状)通常表示边界的一个实例。本领域技术人员将理解在某些实例中,一个元素可以

设计为多个元素或者多个元素可以设计为一个元素。在某些实例中，示为另一元素的内部组件的元素可以实现为外部组件，反之亦然。此外，元素可以不按比例描绘。

[0107] 参考图1，用于元数据处理的可编程单元(PUMP)10集成到现有的精简指令集计算或者计算机(RISC)处理器12中，其具有依序实现和适于能耗敏感(energy-conscious)应用的5-级流水线，其实际上以PUMP 10的附加转换为6-级流水线。第一级是读取级14，第二级是解码级16，第三级是执行级18，第四级是存储器级20，且第五级是写回级22。PUMP 10介入存储器级20和写回级22之间。

[0108] 各种实施例可以使用作为提供策略推行和元数据传播的机制的电子逻辑来实现PUMP 10。PUMP 10的实施例可以由以下表征：(i)关于在四个不同策略及其组合下的基准点的标准集合的PUMP 10的简单实现的运行时间、能量、功率上限和面积影响的经验评估；(ii)微架构的优化的集合；和(iii)来自这些优化的增益的量度，示出了在10％以下的典型的运行时间开销，10％的功率上限影响，和典型地通过使用片上存储器结构的110％附加面积在60％以下的能量开销。

[0109] 在计算中，基准设置可以表征为通常通过对其运行多个标准测试和试验而运行计算机程序、程序的集合或者其他操作的动作，以评估对象的相对性能。在这里使用的术语“基准点(benchmark)”指的是基准设置程序本身。贯穿本申请和附图使用的基准点程序的类型是GemsFDTD、astar、bwaves、bzip2、cactusADM、calculix、deall、gamess、gcc、gobmk、gromacs、h264ref、hmmer、Ibm、leslie3d、libquantum、mcf、mile、namd、omnetpp、perlbench、sjeng、specrand、sphinx3、wrf、zeusmp和mean。例如，参见图10A、图10B和图10C。

[0110] 如在此使用的，“逻辑”包括但不限于执行一个或多个功能或者一个或多个动作，和/或导致来自另一逻辑、方法和/或系统的功能或者动作的硬件、固件、软件和/或每个的组合。例如，基于期望应用或者需要，逻辑可以包括软件控制微处理器、类似处理器(例如，微处理器)的离散逻辑，专用集成电路(ASIC)、编程逻辑器件、包括指令的存储器装置、具有存储器的电气设备等。逻辑可以包括一个或多个门，门的组合或者其他电路组件。逻辑还可以完全具体实现为软件。在描述多个逻辑时，可以将多个逻辑包括在一个物理逻辑中。类似地，在描述单个逻辑时，可以在多个物理逻辑之间分布单个逻辑。

[0111] 在根据在这里的技术的至少一个实施例中，PUMP 10可以表征为对现有的RISC处理器12的扩展。以下段落提供构成可以在根据在这里的技术的实施例中使用的PUMP的10硬件接口层、基本微架构改变和伴随底层软件的ISA(指令集架构)级别扩展的更多细节。

[0112] 在根据在这里的技术的实施例中，充实了PUMP的系统中的每个字可以与指针-大小标记相关联。这些标记在硬件级是未解释的。在软件级别，如策略所定义的，标记可以表示无限制大小和复杂性的元数据。仅需要较少位的元数据的较简单策略可以直接在标记中存储元数据；如果需要更多位，则间接(indirection)用于存储元数据为存储器中的数据结构，其中该结构的地址用作标记。值得注意，这些指针-大小标记是本公开的一个示例性方面且不被考虑为限制。基本可访问存储器字是以标记不可分割地扩展，使得包括存储器、高速缓存和寄存器的所有值隙(value slot)适当地更宽。还标记程序计数器(PC)。该软件定义元数据的概念(notion)及其作为指针-大小标记的表示扩展了先前的标记方法，其中仅较少位用于标记和/或它们是硬布线到固定的解释。标记方案的一些示例性分类法在图15

中再现的表2中表示。

[0113] 元数据标记不可由用户程序寻址。而是，元数据标记由关于如以下详述的根据规则高速缓存未命中调用的策略处理器来寻址。通过PUMP 10规则实现对标记的所有更新。

[0114] 除无限制的元数据之外，根据在这里的技术的PUMP 10的实施例的另一特征是用于关于元数据的单循环公共情况计算的硬件支持。关于形式操作码的规则来定义这些计算：(PC,CI,OP1,OP2,MR)＝>(PC_{new},R)，其应该读为：“如果当前操作码是操作码，关于程序计数器的当前标记是PC，关于当前指令的标记是CI，关于其输入操作数(如果有的话)的标记是OP1和OP2，且关于存储器位置的标记(在加载/存储的情况下)是MR，则在下一机器状态中关于程序计数器的标记应该是PC_{new}，且关于指令结果的标记(目的地寄存器或者存储器位置，如果有的话)应该是R”。该规则格式允许从多达五个输入标记计算两个输出标记，显著地比在先前工作中考虑的更灵活，先前工作典型地从多达两个输入计算一个输出(参见图15中的表2)。超出仅跟踪数据标记(OP1、OP2、MR、R)的先前解决方案，本公开提供可以用于跟踪和推行代码块的出处(provenance)、完整性和使用的当前指令标记(CI)；以及可以用于记录包括隐含信息流的执行历史、环境授权和“控制状态”的PC标记。CFI策略利用用于记录间接跳转(jump)的源的PC标记和用于标识跳转目标的CI标记，NXD+NWC平掌控(leverge)CI以推行该数据不可执行，且污点跟踪使用CI以基于生产其的代码来污染(taint)数据。

[0115] 为解析公共情况中的单个周期中的规则，根据在这里的技术的实施例可以使用最近使用的规则的硬件高速缓存。取决于指令和策略，可以不使用给定规则中的一个或多个输入隙(input slot)。为避免以用于未使用的隙的所有可能值的规则破坏高速缓存，规则高速缓存查找逻辑参考包括用于每个输入隙-操作码对的“不关心”(参见图1)位的位向量，其确定在规则高速缓存查找表(lookup)中是否实际上使用了相应标记。为高效地处理这些“不关心”输入，在向PUMP 10呈现输入之前将它们掩盖。该不关心位矢量由作为未命中处理器安装的一部分的特权指令来设置。

[0116] 图1总地图示根据在这里的技术的一个实施例，其具有包括PUMP 10硬件的修改的5-级处理器12流水线。添加规则高速缓存查找表作为附加级并独立地旁路标记和数据，以使得PUMP 10级不在处理器流水线中创建附加的停顿(stall)。

[0117] 放置PUMP 10作为单独的级(在存储器级20和写回级22之间)针对将关于从存储器读取(加载)，或者要在存储器中重写(存储)的字的标记作为到PUMP 10的输入提供的需要而提出的。因为允许规则取决于正在写入的存储器位置的现有标记，写入操作成为读取-修改-写入操作。类似读取规则在存储器级20期间读取现有标记，在PUMP 10级中检查读取规则，且在也可以被称为写回级22的提交(commit)级期间执行写入。如任何高速缓存方案那样，多级高速缓存可以用于PUMP 10。如以下更详细地描述的，根据在这里的技术的实施例可以利用二级高速缓存。对多级高速缓存的扩展是对本领域技术人员显而易见的。

[0118] 在一个非限制实例中，当在写回级22中的规则高速缓存中发生最后级未命中时，如下处理：(i)仅用于该目的的处理器寄存器的(新)集合中保存当前操作码和标记，和(ii)控制转移到策略未命中处理器(以下更加详细地描述)，其(iii)决定是否允许操作和如果允许操作则生成适当的规则。当未命中处理器返回时，硬件(iv)将该规则安装到PUMP 10规则高速缓存中，且(v)重新发布故障指令。为提供特权未命中处理器和系统软件和用户代码

中的其余之间的隔离，未命中处理器操作模式被添加到处理器，由关于规则高速缓存未命中设置和当未命中处理器返回时复位的在处理器状态中的位来控制。为避免关于每个规则高速缓存未命中保存和恢复寄存器的需要，可以以仅可用于未命中处理器的16个附加寄存器来扩展整数(integer)寄存器文件。另外，规则输入和输出在未命中处理器模式时表现为寄存器(对比的，寄存器窗口)，允许未命中处理器(但是不允许别的)将标记操纵为普通值。再次，这些都是写回级22的非限制实例。

[0119] 添加新未命中-处理器-返回指令以结束将规则安装到PUMP 10规则高速缓存中和返回到用户代码。在该特定的非限制实例中，该指令可以仅当在未命中-处理器模式下时发布。当在未命中-处理器模式时，忽略规则高速缓存且代替地PUMP 10应用单个硬布线的规则：由未命中处理器接触的所有指令和数据必须以预定义的MISSHANDLER(未命中处理器)标记来标记，且给所有指令结果以相同标记。以这种方式，PUMP 10架构防止用户代码破坏由策略提供的保护。替代地，PUMP可以用于推行关于未命中-处理器访问的灵活规则。标记不是可分的、可寻址的或者可由用户代码替代的；元数据数据结构和未命中处理器代码不能由用户代码接触；且用户代码不能直接将规则插入到规则高速缓存中。

[0120] 参考图19，算法1图示对于污点跟踪策略的未命中处理器的操作。为最小化不同标记(且因此规则)的数目，未命中处理器通过“规范化(canonicalizing)”其建造的任何新数据结构来使用用于逻辑等效元数据的单个标记。

[0121] 不是强迫用户选择单个策略，而是同时推行多个策略且之后添加新的策略。对这些“无限制”标记的示例性优点是它们可以同时推行任意数目的策略。这可以通过让标记是指向来自几个组成策略的标记的元组(tuple)的指针来实现。例如，为组合NXD+NWC策略与污点跟踪策略，每个标记可以是指向元组(s,t)的指针，其中s是NXD+NWC标记(DATA(数据)或者CODE(代码))且t是污点标记(到污点的集合的指针)。规则高速缓存查找表是类似的，但是当发生未命中时，分开地评估两个组成策略：仅当两个策略都允许操作时才允许操作，且产生的标记是来自两个组成策略的结果对。但是，在其他实施例中，可以表示要怎样组合策略(不是简单地是所有组成部分之间的AND(与))。

[0122] 参考图20，算法2示出对于任何N个策略的复合未命中处理器的通用行为。取决于元组中的标记怎样关联，这可能导致标记且由此规则的数目的大的增加。为了表明同时支持多个策略和度量其关于工作集大小的效果的能力，通过实验实现复合策略(“复合(composite)”)，且复合策略包括上面描述的所有四个策略。复合策略表示以下进一步详细描述的支持的策略工作负荷的种类。如在图4A和图20中看到的，复合策略同时推行以下策略(i)空间和时间存储器安全，(ii)污点跟踪，(iii)控制流完整性和(iv)代码和数据分离。

[0123] 大多数策略在操作码上分派以选择适当的逻辑。类似NXD+NWC的某些策略将仅检查是否允许操作。其他的可以查阅数据结构(例如，CFI策略查阅允许的间接调用的图和返回id)。存储器安全检查地址颜色(即，指针颜色)和存储器区域颜色之间的相等性。污点跟踪通过组合输入标记(Alg 1)计算新鲜的结果标记。必须访问大的数据结构(CFI)或者跨越大的集合(污点跟踪，复合)规范化的策略可以做出将在片上高速缓存中未命中的许多存储器访问并去到DRAM。在跨越全部基准点的平均上，NXD+NWC的服务未命中需要30个循环，存储器安全需要60个循环，CFI需要85个循环，污点跟踪需要500个循环，且复合需要800个循环。

[0124] 如果策略未命中处理器确定不允许操作，则其调用适当的安全性故障处理器。该故障处理器做什么取决于运行时间系统和策略；典型地，它将关闭违规处理，但是在有些情况下它可以代替地返回适当的“安全值”。对于以UNIX-风格操作系统的增加的部署，每个进程(process)应用假定的策略，允许每个进程得到不同的策略集合。每个进程应用的叙述是非限制的而是示例性的，且本领域技术人员认可这一点。它还允许我们将标记、规则和未命中处理支持放置到进程的地址空间中，避免OS-级别上下文切换的需要。长期来讲，可能的PUMP策略也可以用于保护OS。

[0125] 以下使用图1示出的128b字(64b有效载荷和64b标记)和修改的流水线处理器12，详述用于测量运行时间、能量、面积和功率开销的评估方法，且将其应用在PUMP硬件和软件的简单实现上。首先描述和测量多个PUMP实现是有用的，即使优化实现是开销(相对于基线处理器)最终期望的版本。描述两者，因为其详述在变成更多复杂的版本之前的关键机制的基本版本。

[0126] 为估计PUMP的物理资源影响，主要地聚焦于存储器成本，因为存储器是在简单的RISC处理器和在PUMP硬件扩展中的主要的面积和能量消费者。对于L1存储器(参见图1)考虑32nm低工作功率(LOP)处理，且对于L2存储器考虑低备用功率(LSTP)，且32nm低工作功率(LOP)处理使用CACTI 6.5用于建模主存储器和处理器片上存储器的面积、访问时间、每个访问的能量和静态(泄漏)功率。

[0127] 基线处理器(没有-PUMP)具有用于数据和指令的单独的64KB L1高速缓存和统一的512KB L2高速缓存。使用延迟优化的L1高速缓存和能量优化的L2高速缓存。所有高速缓存使用写回纪律。基线L1高速缓存具有大约880ps的等待时间；假定它可以在一个循环返回结果且将其时钟设置为1ns，给定可与现代的嵌入式和蜂窝电话处理器相比的1GHz周期目标。在图16中的表3中呈现用于该处理器的参数。

[0128] PUMP规则高速缓存10硬件实现的一个实施例可以包括两个部分：以标记在级14、16、20扩展所有架构状态，和将PUMP规则高速缓存添加到处理器12。以64b标记扩展片上存储器中的每64b字增加它们的面积和每个访问的能量并恶化它们的访问等待时间。这对于L2高速缓存是潜在地可容忍的，L2高速缓存已经具有多周期访问等待时间且不是每个周期都使用。但是添加额外周期的等待时间给访问L1高速缓存(参见图1)可能导致流水线中的停顿。为避免此，在该简单实现中，L1高速缓存的有效容量减小到在基线设计中的一半且然后添加标记。这给出对L1高速缓存的相同的单周期访问，但是可能由于增加的未命中而恶化性能。

[0129] 在根据在这里的技术的实施例中，与传统的高速缓存地址密钥(小于地址宽度)相比，PUMP规则高速缓存10利用长匹配密钥(5个指针-大小标记加指令操作码，或者328b)，并返回128b结果。在一个实施例中，可以使用完全关联的L1规则高速缓存，但是将导致高能量和延迟(参见图16中的表3)。作为替代，根据在这里的技术的实施例可以利用以四个散列(hash，哈希)函数激发的多散列高速缓存方案，如图22所示。L1规则高速缓存设计用于在单个周期中产生结果，在第二周期中检查假命中，当L2规则高速缓存设计用于低能量时，给出多周期访问等待时间。再次，图16中的表3示出了简单实现中使用的的1024-项L1和4096-项L2规则高速缓存的参数。当这些高速缓存达到容量时，使用简单先入先出(FIFO)替换策略，其看来对当前工作负荷实际上工作良好(这里FIFO在LRU的6％之内)。

[0130] 参考图2,PUMP的性能影响的估计标识ISA、PUMP和地址-轨迹模拟器的组合。gem5模拟器24生成关于64位Alpha基线ISA的SPEC CPU2006程序的指令轨迹(省略在其上gem5失败的xalancbmk和tonto)。每个程序模拟以上列出的四个策略中的每一个和用于IB指令的预热时段的复合策略且然后评估下一500M指令。在gem5模拟器24中,每个基准点在基线处理器上运行而没有标记或者策略。产生的指令轨迹26然后运行通过执行每个指令的元数据计算的PUMP模拟器28。该“调相的”模拟策略对于失败-停止策略是精确的,在失败-停止策略中PUMP的结果不能导致程序的控制流偏离其基线执行。当地址-轨迹模拟可能对于高度流水线的和非依序(out-of-order)的处理器是不精确的时,它们对于简单的依序(in-order)5和6-级流水线是相当精确的。关于基线配置,之后是地址模拟器32中的定制地址-轨迹模拟和记账的gem5指令模拟和地址轨迹生成30在gem5的周期-精确模拟的1.2%之内。

[0131] PUMP模拟器28包括实现每个策略的未命中-处理器代码(以C语言写的),且取决于策略给初始存储器分配元数据标记。PUMP模拟器28允许在PUMP 10规则高速缓存中捕获访问模式和估计关联的运行时间和能量成本,说明需要长的等待周期来访问L2规则高速缓存。因为具有未命中处理器代码的PUMP模拟器28还在处理器上运行,用于关于gem5的未命中处理器的单独的模拟捕获其动态行为。因为未命中-处理器代码潜在地影响数据和指令高速缓存,创建合并的地址轨迹,其包括来自用户和未命中-处理器代码两者的适当地交织的存储器访问,该未命中-处理器代码最终地址-轨迹模拟以估计存储器系统的性能影响。

[0132] 在以下段落中,与非-PUMP基线相比提供单PUMP的评估。

[0133] 作为评估的一点,应当注意在基线处理器之上的PUMP 10的总的面积开销是190%(参见图16中的表3)。该面积开销的主要部分(110%)来自PUMP 10规则高速缓存。统一的L2高速缓存贡献大部分的剩余面积开销。L1 D/I高速缓存保持大致相同,因为它们的有效容量被减半。该高存储器面积开销大致为三倍静态功率,贡献24%的能量开销。

[0134] 评估的另一点涉及运行时间开销。对于关于大多数基准点的所有单个策略,即使该简单实现的平均运行时间开销也仅是10%(参见图3A和图3B;读箱型图:条是中值,箱型覆盖以上和以下各一个四分位数(中间50%的情况),点表示每个单独的数据点,须线指示除了非正常值(多于1.5×各个四分位数)之外的全范围),其中主要的开销来自将标记位转移到处理器和从处理器转移需要的附加DRAM业务量。对于存储器安全策略(图3A和图3B),存在展现高未命中处理器开销的几个基准点,由于关于新分配的存储器块的推行未命中,推动它们的总和开销多达40-50%。对于复合策略运行时间(在附图中标为“CPI”或者“CPI开销”),五个基准点遭受未命中处理器中的非常高的开销(参见图4A),其中最坏情况在GemsFTDT中接近780%,且几何平均达到50%。对于在图4B中示出的复合策略能量(在附图中标为“EPI”或者“EPI开销”),三个基准点(即,GemsFTDT、astar、omnetpp)遭受未命中处理器中的非常高的开销,其中最坏情况在GemsFTDT中接近于1600%,在astar中接近于600%,且在omnetpp中接近于520%。

[0135] 两个因素贡献于该开销:(1)解析最后级别规则高速缓存未命中需要的大量周期(因为必须查询每个组成未命中处理器),和(2)扩展工作集合大小和增加规则高速缓存未命中率的规则的数目的激增。在最坏情况下,唯一复合标记的数目可以是每个组成策略中的唯一标记的积。但是,总的规则在最大单个策略,存储器安全之上增大3x-5x的因数。

[0136] 评估的另一点是能量开销。由于更宽的字移动更多位,和由于未命中处理器代码

执行更多指令，两者都贡献能量开销，影响单个和复合策略两者(图3B和图4B)。CFI和存储器安全策略且因此复合策略访问通常需要耗费能量的DRAM访问的大的数据结构。最坏情况的能量开销对于单个策略接近400％，且对于复合策略是大约1600％，其中几何平均开销大约是220％。

[0137] 对于许多平台设计，最坏情况功率，或者相等地，每个周期的能量是限制器。该功率上限可以由平台可以从电池汲取的最大电流或者在移动装置或者具有环境冷却的有线装置中的最大维持操作温度来驱动。图4C示出了简单实现以在基线和单PUMP实现两者中驱动最大功率的1bm将最大功率上限提升76％。注意到，该功率上限增加低于最坏情况能量开销，部分地因为某些基准点多于它们耗费的额外能量地减慢，且部分地因为具有高能量开销的基准点是在基线设计中每个周期消耗最少绝对能量的基准点。典型地这些能量高效程序的数据工作集合适配片上高速缓存，这样它们很少付出DRAM访问的高成本。

[0138] 包括上面描述的前述实现的实施例关于大多数基准点实现合理的性能，关于它们中的某些的复合策略的运行时间开销和关于所有策略和基准点的能量和功率开销看来不可接受地高。为解决这些开销，可以引入一系列目标微架构优化且包括在根据在这里的技术的实施例中。在图17的表4中，关于总成本对于与PUMP组件相关联的架构参数的影响来检查这些优化。具有相同规则的操作码的分组用于增加PUMP规则高速缓存的有效容量，标记压缩用于减小DRAM转移的延迟和能量，短标记用于减小片上存储器中的面积和能量，和统一组件策略(UCP)和组成标记(CTAG)高速缓存用于减小未命中处理器中的开销。

[0139] 现在将描述的是可以在根据在这里的技术的实施例中使用的“操作组”。在实际策略中，对于几个操作码定义类似规则是普遍的。例如，在污点跟踪策略中，用于Add(加)和Sub(减)指令的规则相同(参见图19中的算法1)。但是，在简单实现中，这些规则占用规则高速缓存中的单独的项(entry)。基于该观察，将指令操作码(“操作码”)以相同规则分组为“操作组”，减小需要的规则的数目。哪些操作码可以被分组在一起取决于策略；因此在执行级18(图1)扩展“不关心”SRAM以在规则高速缓存查找之前将操作码转译为操作组。对于复合策略，300以上的Alpha操作码减小为14个操作组，且规则总数减小因数1.1×-6×。平均是1.5×(图5A测量跨越所有SPEC基准点的效果)。这有效地增加了硅面积中给定投资的规则高速缓存容量。操作组还减小强制性未命中的数目，因为关于组中的单个指令的未命中而安装应用于组中的每个指令操作码的规则。图5B概述对于具有和没有操作分组的复合策略的不同L1规则高速缓存大小的跨越所有SPEC基准点的未命中率。图5B示出了由操作分组减小的未命中率的范围和平均两者。特别地，在操作组优化之后的1024-项规则高速缓存具有比没有它的4096-项规则高速缓存更低的未命中率。较低的未命中率自然地减小了在未命中处理器中花费的时间和能量(参见图12A和图12B)，且较小的规则高速缓存直接减小面积和能量。

[0140] 根据在这里的技术的实施例可以利用现在将描述的主存储器标记压缩。使用关于64b字的64b标记加倍离片存储器业务量且因此近似地加倍关联的能量。虽然典型地，标记展现空间局部性-许多相邻字具有相同标记。例如，图6A绘出具有复合策略的gcc基准点的每个DRAM转移的唯一标记的分布，示出了大多数字具有相同标记：平均起来8字高速缓存线的每个DRAM转移仅有大约1.14个唯一标记。利用该空间标记局部性来压缩必须转移到离片存储器和从离片存储器转移的标记位。因为在高速缓存线中转移数据，高速缓存线用作该

压缩的基础。在主存储器中分配每个高速缓存线128B以保持寻址简单。

[0141] 但是，如图6B所示，不是直接存储128b标记的字，而是存储八个64b字(有效载荷)，后面是八个4b索引，且然后多达八个60b标记。索引标识哪个60b标记与关联的字一起。标记被缩减到60b以容纳索引，但是这并不牺牲标记作为指针的使用：假定字节寻址和16B(两个64b字)对准的元数据结构，64b指针的低4b可以填充为零。结果，在转移索引的4B之后，所有剩余的是需要在高速缓存线中转移唯一7.5B标记。例如，如果由高速缓存线中的所有字使用相同标记，则在第一读取中存在64B+4B＝68B的转移，然后第二读取中是8B，总共76B代替128B。该4b索引可以是直接索引或者特殊值。定义特殊索引值以表示默认标记，以使得在该情况下不需要转移任何标记。通过以该方式压缩标记，每个DRAM转移的平均能量开销从110％减小到15％。

[0142] 例如，由于其在减小离片存储器能量方面的简单性和有效性的组合，可以在根据在这里的技术的实施例中使用以上呈现的压缩方案。本领域技术人员清楚地认可用于细粒度存储器标记的附加的替换巧妙方案存在-包括多级标记页表、变量-粒度TLB-类结构和范围高速缓存-且这些也可以用于减小在根据在这里的技术的实施例中的DRAM占用空间(footprint)。

[0143] 现在将描述的是怎样在根据在这里的技术的实施例中执行标记转译。再次参考图1，简单PUMP规则高速缓存很大(添加110％面积)，因为每个高速缓存规则是456b宽。支持PUMP 10还需要以64b标记扩展基线片上存储器(RF和L1/L2高速缓存)。这里使用用于每个64b字的完全64b(或者60b)标记导致重的面积和能量开销。但是，64KB L1-D$仅保存8192字且由此至多保存8192个唯一标记。与64KB L1-I$一起，在L1存储器子系统中可能有至多16384个唯一标记；这些可以仅以14b标记表示，减小系统中的延迟、面积、能量和功率。高速缓存(L1、L2)存在以利用时间局部性，且该观察提出可以利用局部性以减小面积和能量。如果标记位减小到14b，则PUMP规则高速缓存匹配密钥从328b减小到78b。

[0144] 为获得前述节省优点而不损失完全指针大小标记的灵活性，不同宽度标记可以用于不同片上存储器子系统和按照需要在这些之间转译。例如，一个可以在L1存储器中使用12b标记和在L2存储器中使用16b标记。图7A详述可以在L1和L2存储器子系统之间执行的标记转译。将字从L2高速缓存34移动到L1高速缓存36需要将其16b标记转译为相应的12b标记，如果需要创建新的关联。对于L2-标记到L1-标记转译的简单SRAM 38具有指示是否存在用于L2标记的L1映射的额外位。图7B详述使用L1标记作为地址的以SRAM 39查找表执行的L1标记40到L2标记42(关于写回或者L2查找表)的转译。类似的转译在60b主存储器标记和16b L2标记之间发生。

[0145] 当长标记不在长短转译表中时，分配新的短标记，潜在地收回不再使用中的先前分配的短标记。存在丰富的设计空间来研究确定何时收回短标记，包括无用单元收集和标记使用计数。为简单起见，短标记被顺序地分配和当短标记空间耗尽时在给定级别以上(指令，数据和PUMP)清空(flush)所有高速缓存，避免跟踪何时特定短标记可用于收回的需要。可以以使得高速缓存便宜地清空的适当的技术设计高速缓存。例如，在根据在这里的技术的实施例中，可以以轻量群组清除(lightweight gang clear)明确设计所有高速缓存，如本领域已知的，和例如在K.Mai，R.Ho，E.Alon，D.Liu，Y.Kim，D.Patil和M.Horowitz，Architecture and Circuit Techniques for a 1.1GHz 16-kb Reconfigurable Memory

in 0.18um-CMOS.IEEE J.Solid-State Circuits,40(1):261-275,2005年1月中描述的,将其通过引用包括在这里。

[0146] 与表3(在图16中再现)相比,其中每个L1规则高速缓存访问成本51pJ,提供在这里的技术用于以8b L1标记减小到10pJ或者以16b L1标记减小到18pJ,其中能量随着这些点之间的标记长度线性地缩放。能量对L1指令和数据高速缓存的影响很小。类似地,对于16b L2标记,L2 PUMP访问成本120pJ,从以64b标记的173pJ下降。瘦L1标记也允许我们恢复L1高速缓存的容量。对于12b标记,全容量(76KB,有效的64KB)高速缓存将满足单循环定时要求,减小了简单实现由减小的L1高速缓存容量导致的性能代价。结果,L1标记长度研究限于12位或者更少。虽然即使短标记也减小能量,但是它们也增加清空的频率。

[0147] 图8A和图8B示出清空怎样随增加L1标记长度而减少,以及对L1规则高速缓存未命中率的影响。

[0148] 现在将描述的是可以关于未命中处理器加速使用的各种技术。根据在这里的技术的实施例可以将四个策略组合为单个复合策略。参考图20,在算法2中,N-策略未命中处理器的每个调用必须拆开标记的元组,且复合策略需要的规则增加规则高速缓存未命中率,这在图9A中标识。即使污点跟踪和CFI策略单独地具有低未命中率,来自存储器安全策略的较高未命中率也驱动复合策略的未命中率高。单独策略的较低未命中率提示它们的结果可以是可缓存的,即使当复合规则不是可缓存的。

[0149] 关于比如图23中图示的PUMP微架构的各个方面,可以使用硬件结构以优化复合策略未命中处理。根据在这里的技术的实施例可以利用统一组件策略(UCP;参见图21中的算法3)高速缓存(UCP$),其中高速缓存最近的组件策略结果。在这种实施例中,修改用于合成策略的一般未命中处理器以当解析组件策略(例如,参见图21的算法3,这种在线3)时在该高速缓存中执行查找。当该高速缓存对于组件策略未命中时,以软件执行其策略计算(且在该高速缓存中插入结果)。

[0150] 也在图24中图示,可以以与常规PUMP规则高速缓存相同的硬件体系结构实现UCP高速缓存,具有附加的策略标识符字段。FIFO替换策略可以用于该高速缓存,但是可以通过使用比如用于组件策略的重新计算成本的量度来优先化空间而实现最好的结果。对于适度容量,该高速缓存滤出大多数策略重新计算(图9B;存储器安全的低命中率由与新存储器分配相关联的强制性未命中驱动)。结果,未命中处理器周期的平均数对于最有挑战的基准点减小因数5(图9E)。当在L2 PUMP中存在未命中时,对于每个策略可以在UCP高速缓存中命中,因为需要的复合规则可能是少量组件策略规则的积。对于GemsFDTD,三个或更多组件策略在大约96%的时间命中。

[0151] 也包括在图23和图24中,可以添加高速缓存以将结果标记的元组转译为其规范的复合结果标记。前述高速缓存可以被称为组成标记(CTAG)高速缓存(CTAG$),因为其对于几个组件策略规则返回结果标记的相同元组是共同的,所以其是有效的(图9D)。例如,在多数情况下,PCtag将是相同的,即使结果标记不同。此外,许多不同规则输入可能导致相同输出。例如,在污点跟踪中,执行集合联合,且许多不同联合将具有相同结果;例如,(蓝色,{A,B,C})是{A}∪{B,C}和{A,B}∪{B,C}(污点跟踪)两者的结果写入到蓝色隙(blue slot)(存储器安全)中的复合答案。FIFO替换策略用于该高速缓存。CTAG高速缓存将平均未命中处理器周期减小另一因数2(参见图9E)。

[0152] 合起来，2048-项UCP高速缓存和512-项CTAG高速缓存将在每个L2规则高速缓存未命中上花费的平均时间从800周期减小到80周期。

[0153] 根据在这里的技术的实施例还可以通过预取在包括规则的一个或多个高速缓存中存储的一个或多个规则来改进性能。因此，另外可以以近期可能需要的预先计算的规则减小强制性未命中率。示例性实例具有用于存储器安全规程的高价值。例如，当分配新存储器标记时，将需要用于该标记的新规则(初始化(1)，添加偏移到指针和移动(3)，标量加载(1)，标量存储(2))。因此，所有这些规则可以一次性添加到UCP高速缓存。对于单一策略存储器安全情况，规则可以直接添加到规则高速缓存中。这将存储器安全未命中处理器调用的数目减小2x。

[0154] 关于总的评估和参考图11A，架构参数单调地影响特定成本，提供能量、延迟和面积的折衷，但不定义单个成本标准内的最小值。存在阈值效应，即一旦标记位足够小，则L1 D/I高速缓存可以恢复到基线的容量，以使得采用基线作为上限以用于L1标记长度，但是在该点以外，减小标记长度会减小能量，对性能影响小。

[0155] 图11B示出减小标记长度是大多数基准点程序(例如Ieslie3d，mcf)的主要能量效应，其中几个程序示出了由增加UCP高速缓存容量(例如，GemsFDTD，gcc)带来的相等或者较大益处。忽略其他成本关心，为减小能量，选择大的未命中处理器高速缓存和少的标记位。运行时间开销(参见图11A)也以较大的未命中处理器高速缓存最小化，但是受益于多于更少的标记位(例如GemsFDTD，gcc)。这些益处的大小跨越基准点和策略而变化。跨越所有基准点，超出10b L1标记的益处对于SPEC CPU2006基准点是小的，所以10b用作能量和延迟之间的折中，并当变得接近于关注的架构参数的空间内的最低能量级时，使用2048-项UCP高速缓存和512-项CTAG高速缓存以减小面积开销。

[0156] 图12A和图12B示出应用优化对运行时间和能量开销的总体影响。每个优化对于某些基准点是主要的(例如，对于astar是操作组，对于lbm是DRAM标记压缩，对于h264ref是短标记，对于GemsFDTD是未命中处理器加速)，且某些基准点看起来受益于所有优化(例如gcc)，其中每个优化连续地除去一个瓶颈和暴露下一个。来自基准点的不同行为遵循如下详述的它们的基线特性。

[0157] 具有较低局部性的应用具有由于高主存储器业务量而由DRAM驱动的基线能量和性能。这种基准点(例如，lbm)中的开销趋于DRAM开销，所以DRAM开销的减小直接影响运行时间和能量开销。具有更多局部性的应用在基线配置方面更快，消耗更少能量，且受DRAM开销影响更少。结果，这些基准点由减小的L1容量和L1 D/I和规则高速缓存中的标记能量更重地影响。DRAM优化对这些应用具有较少效果，但是使用短标记对能量具有大的效果且除去了L1 D/I高速缓存容量代价(penalty)(例如h264ref)。

[0158] 具有重的动态存储器分配的基准点由于强制性未命中而具有较高的L2规则高速缓存未命中率，因为新创建的标记必须安装在高速缓存中。这以简单实现驱动了几个基准点(GemsFDTD，omnetpp)的高开销。如在这里描述的未命中处理器优化减小这种未命中的公共情况成本，且操作组优化减小容量未命中率。对于简单实现，GemsFDTD每200个指令取得L2规则高速缓存未命中，且用800个周期以服务每个未命中，驱动它的780％运行时间开销的大部分(参见图4A)。通过优化，GemsFDTD基准点每400个指令服务L2规则高速缓存未命中，且每个未命中平均仅用140个周期，将其运行时间开销减小到大约85％(参见图10A)。

[0159] 总的来说,这些优化对于除了GemsFDTD和omnetpp之外的所有基准点使得运行时间开销到10%以下(参见图10A),这在存储器分配上是高的。该平均能量开销接近60%,仅具有4个基准点超过80%(参见图10B)。

[0160] 为了图示,可以使用以以不同方式应对的PUMP的四个不同策略的组成(参见在图14的表1)来测量PUMP的性能影响,并图示安全性属性的范围:(1)使用标记区分存储器中的代码与数据并提供对于简单代码注入攻击的保护的不可执行数据和非可写代码(NXD+NWC)策略;(2)检测堆分配存储器中的所有空间和时间违规的存储器安全策略,以实际上无限的(260)颜色的数目("污点记号")扩展;(3)限制仅到程序的控制流图中的允许的边缘的间接控制转移的控制流完整性(CFI)策略,防止面向返回的编程风格攻击(推行细粒度CFI,不是对攻击潜在地脆弱的粗晶粒的近似);和(4)细粒度的污点跟踪策略(一般化),其中每个字可以潜在地同时由多个源(库和10个流)污染。如在这里其他地方注意到的,这些是其保护性能已经在文献中建立的已知策略,且在这里的描述可以聚焦于测量和减小使用PUMP推行它们的性能影响。除了NXD+NWC之外,这些策略中的每一个区分实质上无限数目的唯一项;相反,具有有限数目的元数据位的解决方案最好也仅可以支持粗略简化的近似。也如以上注意到的,PUMP的简单、直接实现可能是昂贵的。例如,将指针大小(64b)标记添加到64b字至少加倍系统中的所有存储器的大小和能量使用。规则高速缓存添加面积和在其上的能量。对于该简单实现,测量的面积开销大约是190%,且geomean能量开销是大约220%;此外,运行时间开销关于某些应用是失望的(在300%以上)。这种高开销将阻碍采用,如果它们是能够做到的最好的话。

[0161] 比如在这里描述的微架构优化可以包括在根据在这里的技术的实施例中以将关于功率上限的影响减小到10%(参见图10C),提出优化的PUMP将对平台的操作包络线有很少影响。DRAM压缩将lbm的能量开销减小到20%;因为它也减慢9%,其功率需要仅增加10%。

[0162] 与简单设计的190%相比(例如,参见图16的表3),优化的设计的面积开销是大约110%(例如,参见图18的表5)。短标记显著地减小L1和L2高速缓存(现在在基线以上仅添加5%)和规则高速缓存(仅添加26%)的面积。相反地,优化设计花费某些面积以减小运行时间和能量开销。UCP和CTAG高速缓存添加33%面积开销,同时短标记的转译存储器(L1和L2两者)添加另外的46%。虽然这些附加的硬件结构添加面积,它们提供能量的净减小,因为不经常访问它们,且UCP和CTAG高速缓存也实质上减小未命中处理器周期。

[0163] 如在这里描述的模型和优化的一个目标是使得其对于实施例添加同时推行的附加策略是相对简单的。由于未命中处理器运行时间的大的增加,关于简单PUMP设计的复合策略导致对于几个基准点的多于递增的成本,但是这些由未命中处理器优化得以减少。

[0164] 图13A(对于CPI开销)和图13B(对于EPI开销)通过首先示出每个单个策略的开销,然后示出添加策略到存储器安全的复合体(最复杂的单个策略),来图示递增的策略添加怎样影响运行时间开销。与添加较高开销策略相反地,该渐进方式使得什么开销简单地来自添加任何策略更清楚。为了得到缩放超出这里的四个策略的感受,CFI策略(返回和计算的跳转/调用)和污点跟踪策略(代码污染和I/O污染)每个被分为两部分。示出附加的策略的运行时间开销递增地跟踪第一复杂策略(存储器安全)以上,对非离群值(non-outliers)没有可感知的运行时间影响(最坏情况非离群值从9%上升到10%开销)和在两个离群值中的

较大的增加(20-40％)，因为主要由于增加的未命中处理器解析复杂性而添加每个新种类的策略。能量遵循类似的趋势，对非离群值策略有适度的影响(geomean从60％上升到70％)，这对于除了GemsFDTD之外的所有情况都是这样。

[0165] 在图15再现的表2中标识有关工作的概要。

[0166] 按照根据在这里的技术的策略编程模型，PUMP策略与操纵这些标记的规则的集合一起包括标记值的集合以实现某些期望的标记传播和推行机制。规则以两个形式：系统的软件层(符号规则)或者硬件层(具体规则)。

[0167] 例如，为图示PUMP的操作，考虑用于在程序执行期间限制返回点的简单示例策略。该策略的动机来自于已知为面向返回编程(ROP)的一类攻击，其中攻击者识别出受到攻击的程序的二进制可执行中的“小配件(gadget)”的集合，并通过构造每个包括指向某个小配件的返回地址的堆栈帧的适当序列来使用这些"小配件"的集合来组成复杂的恶意行为；然后利用缓存溢出或者其他脆弱性来以期望序列重写堆栈的顶部，导致依序执行片段(snippet)。限制ROP攻击的一个简单方式是将返回指令的目标限制到明确定义的返回点。这是通过以元数据标记目标标记作为有效返回点的指令来使用PUMP。每次执行返回指令，PC上的元数据标记设置为检查指示刚刚发生返回。在下一指令，检查PC标记，验证关于当前指令的标记是目标，且如果关于当前指令的标记不是目标则以信号通知安全性违规。通过使得元数据更丰富，可以精确地控制哪些返回指令可以返回到哪些返回点。通过使得其更丰富，可以实现完全CFI检查。

[0168] 从PUMP 10的策略设计者和软件部分的观点，可以使用以微小域-专用语言所写的符号规则来紧凑地描述策略。示例性符号规则及其程序语言例如在标题为“PROGRAMMING THE PUMP,Hardware-Assisted Micro-Policies for Security(编程PUMP，硬件辅助的安全性微策略)”的部分中描述。

[0169] 符号规则可以紧凑地编码多种多样的元数据跟踪机制。但是，在硬件级，规则需要调节为有效率的解释的表示以避免减慢初级计算。为此，可以引入较低级规则格式，称为具体规则。直观地，给定策略的每个符号规则可以扩展为具体规则的等效集合。但是，因为单个符号规则总的来说可能生成无限制数目的具体规则，所以缓慢地执行该工作，当系统执行时按照需要生成具体规则。

[0170] 对于具有元数据标记(例如，比ROP更丰富的)的策略，从符号到具体规则的转译遵循相同的总的路线，但是细节变得有点复杂。例如，污点跟踪策略取得要作为指向存储器数据结构的指针的标记，每个描述任意地大小的污点集合(表示可能贡献于给定条数据的数据源或者系统组件)。加载操作组的符号规则叙述加载值上的污点应该是在指令本身上的污点、负载的目标地址和在该地址的存储器的并集。符号规则及其程序语言通过从先前标识的标题为“PROGRAMMING THE PUMP,Hardware-Assisted Micro-Policies for Security”的论文引用而包括于此且可用于公开检查。

[0171] 为减小不同标记的数目(且因此，对规则高速缓存的压力)，可以以规范形式内部存储元数据结构，且因为标记不变，完全地利用共享(例如，给予集合元素规范次序以使得这些集合可以紧凑地表示，共享公共前缀子集)。当不再需要时，可以收回(例如，由无用单元收集)这些结构。

[0172] 实施例可以利用复合策略。可以通过令标记为指向来自几个组件策略的标记的元

组的指针来同时推行多个正交策略。(一般来说,多个策略可以不正交)。例如,为构成具有污点跟踪策略的第一返回操作组(ROP)策略,令每个标记为指向元组(r;t)的表示的指针,其中r是ROP-标记(代码位置标识符)且t是污点标记(到污点的集合的指针)。高速缓存查找处理也是完全一样的,但是当发生未命中时未命中处理器提取元组的组件且分派到评估符号规则的两个集合的例程。仅当两个策略都具有应用的规则时允许操作;在该情况下,产生的标记是到包括来自两个子策略的结果的对的指针。

[0173] 关于策略系统和保护,策略系统作为每个用户进程内的存储器的分隔区域存在。策略系统例如可以包括用于未命中处理器的代码、策略规则和表示策略的元数据标记的数据结构。在该进程中放置策略系统最小地侵入现有的Unix处理模型,且便于策略系统和用户代码之间的轻量切换。策略系统使用接下来描述的机制与用户代码隔离。

[0174] 清楚地,如果攻击者可以重写元数据标记或者改变它们的解释,则由PUMP提供的保护将是无用的。在这里描述的技术设计用于防止这种攻击。内核程序、加载器和(对于某些策略)编译器是可信的。具体来说,编译器被依赖以向字分配初始标记,且当需要时,向策略系统传递规则。加载器将保存由编译器提供的标记,且例如使用加密签名保护从编译器到加载器的路径免于篡改(tamper)。

[0175] 根据在这里的技术的实施例可以使用对于每个进程建立初始存储器图像的标准Unix风格内核程序。(可以使用微策略以消除这些假定中的一些,进一步减小TCB的大小)。进一步假定在这些实施例中,正确地实现规则高速缓存未命中处理软件。这很小,因此是用于形式验证的好目标。一个关切是防止在进程中运行的用户代码推翻由进程的策略提供的保护。用户代码应该不能够(i)直接地操纵标记--所有标记改变应该根据当前生效的一个或多个策略执行;(ii)操纵由未命中处理器使用的数据结构和代码;(iii)在硬件规则高速缓存中直接插入规则。

[0176] 关于寻址,为防止由用户代码对标记的直接操纵,附加到每个64b字的标记本身不是分开地可寻址的。具体来说,不可以指定仅与标记或者标记的一部分对应的地址以读取或者写入它。所有用户可访问指令在作为原子单元的(数据,标记)对上操作—对值部分操作的标准ALU和对标记部分操作的PUMP。

[0177] 关于在根据在这里的技术的实施例中的未命中处理器架构,可以仅关于对PUMP高速缓存的未命中来激活策略系统。为提供策略系统和用户代码之间的隔离,未命中处理器操作模式被添加到处理器。整数寄存器文件以仅可用于未命中处理器的16个附加寄存器来扩展,以避免保存和恢复寄存器。注意到,16个附加寄存器的使用是说明性的且实际上可能需要扩展整数寄存器文件到更少/更多寄存器。故障指令的PC,规则输入(操作组和标记)和规则输出当在未命中处理器模式下时作为寄存器出现。添加未命中处理器返回指令,其结束将具体规则安装到高速缓存中并返回到用户代码。

[0178] 在根据在这里的技术的实施例中,当处理器12处于未命中处理器模式时脱离PUMP 10的正常行为。代替地,应用单个硬布线规则:由未命中处理器接触的所有指令和数据必须以与由任何策略使用的标记不同的预定义的未命中处理器标记来标记。这保证相同地址空间中的未命中处理器码和数据与用户代码之间的隔离。用户代码不能接触或者执行策略系统数据或者代码,且未命中处理器不能无意地接触用户数据和代码。未命中处理器返回指令可以仅在未命中处理器模式下发布,防止用户代码将任何规则插入到PUMP中。

[0179] 当先前工作已经使用更聪明的方案来紧凑地表示或者近似安全和安全性策略时，这通常是关于想要策略的折衷，且它可以平衡复杂性和紧凑性。如在这里描述的，可以包括以很少或者没有附加运行时间开销而更完全地和更自然地捕获安全性策略的需要的更丰富的元数据。不是施加对元数据表示和策略复杂性的固定限制，而是PUMP 10提供性能上的适度退化。这允许策略在需要时使用更多数据而不影响通常情况的性能和大小。它进一步允许策略的增加的精细化和性能调节，因为即使复杂策略也可以容易地表示和执行。

[0180] 通过安装用于基于元数据值的策略推行的证据，本公开定义用于软件定义的元数据处理的架构并标识加速器以除去大部分运行时间开销。在这里引入和描述的架构没有关于元数据位的数目或者与四个微架构优化一起同时支持的策略的数目的限制(即，没有任何限制)，该四个微架构优化(操作组、标记压缩、标记转译和未命中处理器加速)实现可与专用硬件元数据传播解决方案相比的性能。软件定义的元数据策略模型及其加速将可应用于除这里图示的那些以外的大范围的策略，包括声音信息-流控制、细粒度访问控制、完整性、同步、竞态检测(race detection)、调试(debug)、应用专用策略和动态代码的受控生成和执行。

[0181] 在这里描述的各种方面和实施例的某些非限制优点提供(i)编程模型和支持用于紧凑地和精确地描述由该架构支持的策略的接口模型；(ii)策略编码和使用四个不同类别的充分研究的策略的组成的具体实例；和(iii)这些策略的需要、复杂性和性能的量化。

[0182] 如在这里描述的实施例的编程模型可以编码许多其他策略。信息-流控制比这里的简单的污点跟踪模型更丰富，但是可以以RIFLE-风格二进制转译或者通过与来自编译器的某些支持一起使用PC标记来跟踪隐含流。微策略可以支持轻量访问控制和划分(compartmentalization)。标记可以用于区分不可伪造的资源。唯一的生成的令牌可以作为用于密封和签署数据的密钥，其然后可以用于强抽象-保证数据仅由授权的代码组件创建和解构。微策略规则可以推行数据不变性质，比如不变性和线性。微策略可以支持并行性，如用于同步原语的带外元数据，比如用于数据或者未来的完全/空位，或者如检测关于锁定的竞态条件的状态。系统设计师可以向现有的代码应用专用微策略而不用核查或者重写每一行。

[0183] 如在这里描述的设计的PUMP 10提供灵活性和性能的吸引人的组合，在以当规则复杂性增长时最得体的性能下降支持更丰富和复合策略的同时，以在多数情况下可与专用机制相比较的单个策略性能来支持不同集合的低级、细粒度安全性策略。另外，由PUMP提供的机制可以用于保护它自己的软件结构。根据在这里的技术的实施例可以通过使用PUMP 10实现“划分”微策略和使用其保护未命中-处理器代码而代替特殊未命中处理器操作模式。最终，如在这里描述的，可以组合策略的正交集合，其中由每个策略提供的保护与其他的完全独立。但是策略通常交互：例如，信息-流策略可能需要在由存储器安全策略分配的新鲜区域放置标记。策略组成需要关于表达和有效率的硬件支持两者方面的分析。

[0184] 现在将描述的是图示在根据在这里的技术的实施例中的存储器安全策略的实现的另一实例，其标识堆-分配存储器中的所有时间和空间违规。在至少一个实施例中，对于每个新分配，可以执行处理以构成新鲜颜色-id的，c，并写入c作为关于新创建的存储器块(例如，比如经由memset)中的每个存储器位置的标记。到新块的指针也标记为c。之后，当执行处理以解参考指针时，处理可以包括检查指针的标记与关于指针参考或者指向的存储器

单元的标记相同。当释放块时，关于块的全部单元的标记可以修改为表示闲置存储器的常数F。堆最初可以标记为F。特殊标记⊥可以用于非指针。因此，通常，实施例可以对于颜色c或者⊥的存储器位置写入标记t。

[0185] 因为存储器单元可以包括指针，总的来说存储器中的每个字可以与两个标记相关联。在这种实施例中，关于每个存储器单元的标记是到对(c,t)的指针，其中c是分配该单元的存储器块的id且t是关于该单元中存储的字的标记。一个实施例可以使用基于用于就符号规则而言指定策略的在这里其他地方描述的规则功能的域专用语言。与检查每个存储器访问有效(即，访问的单元在由该指针指向的块内)一起，用于加载和存储的规则负责封装和解封这些对：

[0186] 加载(load)：$(-,-,c_1,-,(c_2,t_2))$

[0187] 如果$c_1=c_2$，则$\rightarrow(-,t_2)$

[0188] 存储(store)：$(-,-,t_1,c_2,(c_3,t_3))$

[0189] 如果$c_2=c_3$，则$\rightarrow(-,(c_3,t_1))$

[0190] 在前述及其他规则中执行的检查揭示符号规则有效的条件(例如，在存储规则中$c_2=c_3$以上)。符号“-”指示规则中的不关心字段。

[0191] 地址运算操作保存指针标记：

[0192] add：$(-,-,c,\perp,-)\rightarrow(-,c)$

[0193] 为维持关于指针的标记仅可以来源于分配的不变性质，从抓取创建数据的操作(例如，加载常数)将其标记设置为⊥。

[0194] 在实现存储器安全策略的实施例中，比如malloc和释放的操作因此可以使用标记的指令和短暂(ephemeral)规则修改为标记存储器区域(例如，一旦它们被使用就可以从高速缓存删除)。关于malloc，处理可以经由短暂规则生成用于到新区域的指针的新鲜标记。例如，用于移动的规则可以是比如以下的短暂规则：

[0195] 移动(move)：$(-,t_{malloc},t,-,-)\rightarrow^{1}(-,t_{newtag})$

[0196] 具有上标1的箭头(例如，$\rightarrow^{1}$)可以指示短暂规则。新标记的指针然后可以用于使用特殊存储规则写入零到分配区域中的每个工作，特殊存储规则为：

[0197] store：$(-,t_{mallocinit},t_1,c_2,F)\rightarrow(-,(c_2,t_1))$

[0198] 之后，返回标记指针。在之后的时间点，释放可以使用修改的存储指令以重新标记区域为未分配的：

[0199] store：$(-,t_{freeinit},t_1,c_2,(c_3,t_4))\rightarrow(-,F)$

[0200] 之后，该区域返回到释放列表。

[0201] 在这种使用存储器安全策略的实施例中，操作组可以用于描述如下的规则集合：

[0202] (1)nop,cbranch,ubranch,ijump,return：$(-,-,-,-,-)\rightarrow(-,-)$

[0203] (2)ar2sld：$(-,-,\perp,\perp,-)\rightarrow(-,\perp)$

[0204] (3)ar2sld：$(-,-,c,\perp,-)\rightarrow(-,c)$

[0205] (4)ar2sld：$(-,-,\perp,c,-)\rightarrow(-,c)$

[0206] (5)ar2sld：$(-,-,c,c,-)\rightarrow(-,\perp)$

[0207] (6)ar1sld：$(-,-,t,-,-)\rightarrow(-,t)$

[0208] (7)ar1ld,dcall,icall,flags：$(-,-,-,-,-)\rightarrow(-,\perp)$

[0209] (8) load:如果c1＝c2,则$(-,-,c_1,-,(c_2,t_2))\rightarrow(-,t_2)$

[0210] (9) store:如果$c_2=c_3\wedge c_i\notin\{t_{mallocinit}, t_{freeinit})$,则$(-,-,t_1,c_2,(c_3,t_3))\rightarrow(-,(c_3,t_1))$

[0211] (10) store:$(-,t_{mallocinit},t_1,c_2,F)\rightarrow(\rightarrow,(c_2,t_1))$

[0212] (11) store:$(-,t_{freeinit},t_1,c_2,(c_3,t_4))\rightarrow(\rightarrow,F)$

[0213] (12) move:$(-,t_{malloc},t,-,-,)\rightarrow^{1}(-,t_{newtag})$

[0214] (13) move: $(-,\ \overline{t_{malloc}},\ t,\ -,\ -,)\rightarrow(-,\ t)$

[0215] 用于策略说明的以上使用的符号规则可以使用变量来写,允许几个符号规则描述不同值的无限制全域之上的策略。但是,规则高速缓存中存储的具体规则参考特定的具体标记值。例如,如果23和24是有效的存储器块颜色,则实施例可以使用具有用于c＝23和c＝24的PUMP规则高速缓存中的符号规则(3)以上的具体实例的具体规则。例如,假定实施例编码⊥为0,且标明不关心字段为0,则用于以上符号规则(3)的具体规则是:

[0216] ar2sld:(0,0,23,0,0)→(0,23)

[0217] ar2sld:(0,0,24,0,0)→(0,24)

[0218] 与在这里其他地方的讨论一致,在至少一个实施例中,未命中处理器可以获得从符号规则编译的具体输入标记和执行代码以产生关联的具体输出标记,以将规则插入到PUMP规则高速缓存中。当符号规则识别处违规时,控制转移到错误处理器,且没有新的具体规则插入到PUMP规则高速缓存中。

[0219] 现在将描述的是基于RISC-V架构的根据在这里的技术的实施例进一步以元数据标记和PUMP扩展以支持按照在这里的讨论的软件定义的元数据处理(SDMP)。RISC-V可以特性化为精简指令集计算(RISC)指令集架构(ISA)的开源实现。在这种实施例中,元数据标记位于每个字的指令和数据两者上。在RISC-V架构中,字是64位。RISC-V架构提供不同字长变化-具有64位的字长的RV64和具有32位的字长的RV32。寄存器和用户地址空间的宽度或者大小可以随字长而变。标记大小或者宽度可以独立于字长或者宽度,但是在实施例中可能典型地相同。如现有技术中已知的,RISC-V架构具有32位指令且因此使用64位字长支持和操作的实施例可以在单个标记的字中存储2个指令。关于与扩展用于与元数据标记、PUMP和SDMP一起使用的RISC-V架构相关的不同技术和特征的使用,在这里其他地方讨论RISC-V架构的前述及其他方面。

[0220] 例如,在"The RISC-V Instruction Set Manual Vol.1,User-Level ISA,Version 2.0",2014年5月6日,Waterman,Andrew等,(也称为"RISC-V user level ISA")中描述包括用户级指令的RISC-V架构,上述文件通过引用包括于此,且例如在RISCV.ORG网站,且通过加州大学伯克利作为技术报告UCB/EECS-2014-54公开可得到。RISC-V架构还包括特权架构,该特权架构包括特权指令和运行操作系统、附加外部装置等需要的附加功能性,例如如在"The RISC-V Instruction Set Manual Volume II:Privileged Architecture,Version 1.7",2015年5月9日,也称为"RISC-V privileged ISA")中描述的,该文件通过引用包括于此,且例如在RISCV.ORG网站且通过加州大学伯克利作为技术报告UCB/EECS-2015-49公开可用得到。

[0221] RISC-V架构的实施例可以具有如下的四个RISC-V特权级别:用于用户/应用(U)特

权级别的级别0,用于管理程序(S)特权级别的级别1,用于系统管理程序(H)特权级别的级别2,和用于机器(M)特权级别的级别3。在前述中,RISC-V特权级别可以从最高到最低从0到3分级,其中级别0指示最高或者最大特权级别,且级别3指示最低或者最小特权级别。这种特权级别可以用于提供不同组件之间的保护,且对执行当前特权级别或者模式不允许的操作的代码的执行的尝试将导致产生例外,比如到下层执行环境中的陷阱。机器级别具有最高特权,且是RISC-V硬件平台的唯一强制性特权级别。以机器模式(M-模式)运行的代码是固有地可信的,因为它具有对机器实现的低级访问。用户模式(U-模式)和管理程序模式(S-模式)意在分别用于现有的应用和操作系统使用,而系统管理程序模式(H-模式)意在支持虚拟机器监控程序。每个特权级别具有具有可选扩展和变型的特权ISA扩展的核心集合。应当注意,RISC-V架构的实现必须至少支持M-模式,且大多数实现至少支持U-模式和M-模式。可以添加S-模式以提供管理程序级别操作系统的代码与M-模式中执行其他更多特权代码之间的隔离。用户或者应用代码可以典型地在U-模式下执行直到发生陷阱(例如,管理程序调用、页错误)或者中断,迫使控制转移到在支持的较高特权模式或者级别(例如,H、S或者M模式)之一运行的陷阱处理器。然后执行陷阱处理器的代码,且控制然后可以返回到导致陷阱的初始用户代码或者应用。这种用户代码或者应用的执行可以在触发陷阱处理器调用的U-模式中的初始陷阱指令处或者其之后恢复。RISC-V实现中支持的模式的不同组合可以仅包括:单个M模式,两个模式M和U,三个模式M、S和U,或者所有四个模式M、H、S、U。在在这里描述的至少一个实施例中,可以支持所有4个前述特权级别。至少,根据在这里的技术的实施例可以支持M和U模式。

[0222] RISC-V架构具有可以由一个或多个关联的特权级别原子地读取和修改的控制状态寄存器(CSR)。总的来说,CSR可以以四个特权级别的第一个和高于第一个的四个特权级别的任何其他访问。例如,假定在U-模式(级别3)下执行程序且发生比如规则高速缓存未命中的陷阱,由此控制转移到以较高特权或者模式(例如,0-2的任意级别)运行的陷阱处理器,比如规则高速缓存未命中处理器代码。在陷阱的发生时,信息可以位于在M模式下执行的陷阱处理器可访问的CSR中,例如,该CSR要不然就对以较低特权级别执行的任何其他代码不可访问(例如,对在H、S或者U模式下的代码不可访问)。在至少一个实施例中,规则高速缓存未命中处理器可以以PUMP保护级别以上的特权级别运行(例如,可以在H-模式、S-模式或者M-模式下运行)。在这种实施例中,如在这里其他地方描述的,标记定义和策略可以在规则高速缓存未命中处理器级别是跨操作系统(例如,每个虚拟机)全局的,由此可以跨所有执行代码应用相同标记定义和策略。在至少一个实施例中,可以支持每个应用或者处理策略,其中全局地安装这种策略且可以标记PC(标识当前指令的程序计数器)和/或代码以区分进程或者应用专用规则。在虚拟机(VM)不共享存储器的实施例中,可以以每个VN的基础定义策略。

[0223] 按照在这里其他地方的讨论,PUMP可以特性化为用于SDMP的规则高速缓存。可能有在关于指令和指令输入的标记和用于操作结果的标记的集合之间的映射。标记处理是与指令的正常操作独立和并行的。在至少一个实施例中,PUMP并行于普通RISC-V操作运行,提供用于操作结果的标记。因为PUMP是高速缓存,第一次PUMP接收特定指令就发生规则高速缓存未命中,且因此特定的PUMP输入(例如,强制性)的相应集合或者当PUMP不能在高速缓存中保持规则时(例如,超过高速缓存的容量因此从规则高速缓存中逐出规则,或者可能冲

突)。规则高速缓存未命中导致然后由未命中处理器系统(例如,规则高速缓存未命中处理器)的代码处理的未命中陷阱。输入可以通过PUMP CSR传递到未命中处理器,且也可以通过CSR提供规则插入回到PUMP。这在以下更详细地讨论。在这里其他地方讨论第一实施例,其中存在5个PUMP输入标记。作为变型,实施例可以包括不同数目的标记及其他PUMP输入。PUMP标记输入的特定数目可以随指令集和操作数而变。例如,基于RISC-V架构在一个实施例中可以包括以下作为PUMP输入:

[0224] 1.Opgrp-指示包括当前指令的特定操作组。通常,操作组是一组指令的抽象且在这里的其他地方讨论。

[0225] 2.PCtag-关于PC的标记

[0226] 3.CItag-关于指令的标记

[0227] 4.OP1tag-关于到指令的RS 1输入的标记

[0228] 5.OP2tag-关于到指令的RS2输入的标记(或者当CSR指令时关于CSR的标记)

[0229] 6.OP3tag-关于到指令的RS3输入的标记

[0230] 7.Mtag-关于到指令的存储器输入或者指令的存储器目标的标记

[0231] 8.funct12(funct7)-如在这里其他地方描述的在某些指令中发生的扩展的操作码位。

[0232] 9.subinstr-当存在在字中封装的多个指令时,该输入标识在字中的哪个指令是由PUMP操作的当前指令。

[0233] 基于RISC-V架构在一个实施例中可以包括以下作为PUMP输出:

[0234] 1.Rtag-关于结果的标记:目的地寄存器、存储器或者CSR

[0235] 2.newPCtag-关于在该操作之后PC的标记(例如,有时在这里称为PCnew标记)。

[0236] 信息例如可以从在陷阱发生时在U-模式中执行的用户代码经由CSR传递到在M模式中执行的陷阱处理器,比如规则高速缓存未命中处理器。以类似的方式,当恢复U-模式下的程序执行时信息可以经由CSR在M模式下的陷阱处理器之间传递,其中CSR中的信息可以位于在U-模式下可访问的相应的寄存器中。以该方式,可能有在一个特权级别的CSR和在其他特权级别的寄存器之间的映射。例如,在根据在这里的技术的实施例中,可以定义CSR对M模式处理器和PUMP可访问,其中在陷阱发生时特定的指令操作数标记写入到CSR以将标记作为输入传递到PUMP和规则高速缓存未命中处理器。以类似的方式,CSR可以用于将信息从陷阱处理器和/或PUMP(在高于U-模式的特权级别操作)传递到在U-模式中执行的其他代码,比如当在规则高速缓存未命中之后恢复程序执行时(例如,其中当在用于当前指令的PUMP规则高速缓存中未找到匹配规则时发生规则高速缓存未命中)。例如,CSR可以用于输出或者传送用于PCnew和RD的PUMP输出标记。另外,可以定义CSR,其中可以响应于到特定CSR的写入发生不同动作。例如,规则高速缓存未命中处理器代码可以通过写入到特定的CSR而将新规则写入/插入到PUMP的规则高速缓存中。定义的特定的CSR可以随实施例而变。

[0237] 参考图25,示出了在根据在这里的技术的一个实施例中可以定义和使用的CSR的实例。表900包括具有十六进制的CSR地址的第一列902,特权的第二列904,指示CSR名称的第三列906和具有CSR的描述的第四列908。表900的每行可以标识不同定义的CSR的信息。900中的不同的CSR也在这里其他地方关于可以包含在实施例中的附加特征更详细地描述。

[0238] 行901a-c标识具有用于由PUMP标记代码和/或指令的特殊标记值的CSR。在至少一

个实施例中，由项901a定义的sboottag CSR可以包括系统中使用的第一初始或者开始标记值。前述开始标记值可以被称为自举标记值。在一个方面中，自举标记值可以特性化为可以从其衍生出所有其他标记或者所有其他标记基于的“种子”。因此，自举标记可以在一个实施例用作用于生成所有其他标记的开始点。以类似于操作系统中的自举代码的起始位置的初始加载的方式，硬件可以用于将CSR 901a初始化为用作自举标记的特定的预定义标记值。一旦已经读取自举标记作为引导根据在这里的技术的系统的一部分，可以清除sboottag CSR。例如，操作系统代码的特权部分可以包括调用使用自举标记值执行初始标记传播的规则的指令。在这里其他地方进一步描述自举标记的使用和标记生成及传播。行901b标识包含用于标记来自如在这里其他地方描述的公开不可信源的数据的标记值的CSR。对于行901c，标识包含可以用作当标记数据和/或指令时的默认标记值的默认标记值的CSR。

[0239] 行901d和e分别指示用于写入到操作组/关心表的地址和数据(例如，在这里其他地方也称为包括操作组和操作码的关心/不关心位的映射或者转译表)。写入到由行901e指示的CSR触发到操作组/关心表的写入。行901f标识为了清空PUMP规则高速缓存可以写入的CSR。行901g-901m标识将用于当前指令的标记输入提供到PUMP和规则高速缓存未命中处理器的CSR。行901j-m每个指示用于导致规则高速缓存未命中的正在处理的当前指令的操作数的不同操作数标记，由此指令可以包括多达4个这种操作数(4个操作数中的3个是寄存器(CSR 901j-l)且第四个操作数是具有在由行901m指示的CSR中存储的标记的存储器位置)。行901n标识当当前指令的操作码使用如在这里其他地方描述的扩展的funcl2字段时保持扩展操作码位的CSR。行901o标识指示字中的哪个子指令是正在参考的当前指令的CSR。如在这里其他地方讨论的，单个标记的字可以是64位且每个指令可以是32位，由此两个指令可以包括在单个标记的字中。由行901o指示的CSR标识两个指令中的哪个正在由PUMP处理。行901p-q分别标识新PC(例如，下一指令的新PC标记)和RD(目的地寄存器，当前指令的结果的地址)包括PUMP输出标记的CSR。写入到由901q指示的CSR导致规则(例如，匹配触发PUMP规则高速缓存未命中的当前指令)写入到PUMP规则高速缓存中。行901r标识PUMP操作的标记模式。在这里其他地方更加详细地描述标记模式。

[0240] 在至少一个实施例中，可以通过写入到由901e指示的CSR sopgrpvalue来填充用于存储操作组和关心/不关心位的一个或多个表(例如操作组/关心表)，其中前述CSR 901e的内容写入到由901d指示的sopgrpaddr CSR中存储的地址。规则可以响应于写入到由项901q定义的srtag CSR而写入或者安装到PUMP规则高速缓存中。规则写入是经由PUMP CSR(例如，基于PUMP CSR输入901g-o)指定与操作码(或更具体地，用于操作码的操作组)匹配的标记值和用于当前指令的标记值作为到PUMP的输入的规则。

[0241] 为允许关于CSR操作的标记和标记保护，数据流允许CSR标记输入到PUMP和从PUMP输出。根据RISC-V架构，存在分别从CSR读取和写入到CSR的读和写指令。关于具有PUMP的CSR指令，到PUMP的R2tag输入是当前CSR标记。CSR读/写指令(例如，csrrc、csrrci、csrrs、csrrsi、csrrw、csrrwi)写入两个输出：(1)RD和(2)由指令参考的CSR。在该情况下，PUMP输出R标记(或者目的地的RD标记)指定由PUMP输出和将CSRtag直接复制到寄存器目的地标记的CSR标记。

[0242] RDtag←CSRtag

[0243] CSRtag←Rtag

[0244] 关于由列904指示的特权，由行901r的定义CSR mtagmode通过在机器或者M-模式级别的代码执行可访问以用于读/写。由行901a-q定义的其余CSR通过至少在管理程序或者S-模式级别的代码执行可访问以用于读/写。因此，对于各种CSR在列904指示的特权表示执行代码的最小RISC-V特权级别以用于代码访问特定的CSR。实施例可以对于实施例中使用的CSR分配与实例900中图示的不同的RISC-V特权级别。

[0245] 根据在这里的技术的实施例可以定义影响由PUMP执行的标记传播的多标记模式。当前标记模式由在如行901r所定义的CSR mtagmode中存储的当前时间点的值来标识。在至少一个实施例中，可以与RISC-V定义特权(例如，上面描述的M、H、S和U模式)组合地使用标记模式以定义关于PUMP使用的CSR保护模型。

[0246] 为了允许规则高速缓存未命中处理器设置可配置，可以使用进一步扩展RISC-V特权的保护模型。不是完全由特权级别定义PUMP CSR访问，而是可以进一步相对于与RISC-V特权级别结合的当前标记模式来定义CSR访问。因此，在根据在这里的技术的至少一个实施例中，是否允许执行代码访问CSR可以取决于CSR的最小RISC-V特权级别，当前标记模式和执行代码的当前RISC-V特权级别。以下更详细地讨论标记模式。

[0247] 参考图26，示出了可以用于根据在这里的技术的实施例的标记模式的实例。表910包括以下列-912mtagmode位编码，914操作和916标记结果。表910的每一行表示用于不同可能标记模式的信息。当标记模式如由911a表示的是000时，PUMP关闭且不使用且不生成任何标记结果。当标记模式是010时，PUMP写入关于全部结果的默认标记(例如，用于目的地或者结果寄存器或者存储器位置的Rtag)。

[0248] 关于行911c-f，表示是可以对于在不同RISC-V特权级别的代码执行指定用于启用或者脱离PUMP的不同标记模式。当启用PUMP时，PUMP可以特性化为有效、使能的，且当执行代码时提供保护由此在代码执行期间推行其策略的规则。相反地，当脱离PUMP时，PUMP可以特性化为无效、禁用的且当执行代码时不提供保护由此在代码执行期间不推行其策略的规则。当脱离PUMP时，可以使用一个或多个默认标记传播规则传播标记，而不是基于具有匹配当前指令的那些标记值的标记值的规则的评估来传播标记。启用或者脱离PUMP可以随特定的假定可信级别和归因于在不同RISC-V特权级别执行的代码的期望保护级别而变。

[0249] 关于标记模式911c-f，除了由901r表示的mtagmode CSR之外，实例900的全部PUMP CSR仅当脱离PUMP时可访问。也就是，除了由901r表示的mtagmode CSR之外，实例900的PUMP CSR仅对在当前RISC-V操作特权执行的代码或者比由标记模式表示的最高层级PUMP特权更特权的模式可访问(例如，由911c表示的最高层级特权是U模式，由911d表示的最高层级特权是S模式，由911e表示的最高层级特权是H模式，且由911f表示的最高层级特权是M模式)。

[0250] 当如由911c表示的标记模式是100时，PUMP脱离且当RISC-V特权级别表示比U-模式更高或更提升的特权级别时不操作。因此，标记模式911c指示PUMP及其提供保护的规则仅当在U-模式执行代码时启用和推行，由此指示在高于U-模式的特权级别(例如，在S、M或者H模式)执行的代码是可信的。当如由911c表示标记模式是100且执行代码的RISC-V保护级别是S、M或者H模式时，PUMP脱离且其CSR可对仅在S、M或者H模式中执行的代码可访问(例如，CSR对以U-模式执行的代码不可访问)。

[0251] 当如由911d表示的标记模式是101时，PUMP脱离且当RISC-V特权级别表示比S-模

式更高或更提升的特权级别时不操作。因此，标记模式911d指示PUMP及其提供保护的规则仅当在S-模式和U-模式执行代码时启用和推行，由此指示在高于S-模式的特权级别(例如，在M或者H模式)执行的代码是可信的。当如由911d表示标记模式是101且执行代码的RISC-V保护级别是M或者H模式时，PUMP脱离且其CSR可对仅在M或者H模式中执行的代码可访问(例如，CSR对以S或者U模式执行的代码不可访问)。

[0252] 当如由911e表示的标记模式是110时，PUMP脱离且当RISC-V特权级别表示比H-模式更高或更提升的特权级别时不操作。因此，标记模式911e指示PUMP及其提供保护的规则仅当在H-模式、S-模式和U-模式执行代码时启用和推行，由此指示在高于H-模式的特权级别(例如，在M模式)执行的代码是可信的。当如由911e表示标记系统模式是110且执行代码的RISC-V保护级别是M模式时，PUMP脱离且其CSR可对仅在M模式中执行的代码可访问(例如，CSR对以U、H或者S模式执行的代码不可访问)。

[0253] 当如由911f表示标记模式是111时，总是启用PUMP且对于M、H、S和U的全部RISC-V特权级别操作。因此，标记911f指示当在M-模式、H-模式、S-模式和U-模式的任意执行代码时启用和推行PUMP及其提供保护的规则，由此指示没有代码是固有地可信的。通过如由911f表示的标记模式＝111，从不脱离PUMP且其CSR对任何执行代码都不可访问。

[0254] 关于由行911c-f表示的标记模式，当执行代码的当前RISC-V特权级别高于由标记模式表示的最高参与PUMP级别时，可以脱离PUMP，且可以使用一个或多个默认标记传播规则来传播标记。

[0255] 当标记模式具有如由行911a表示的编码000时(指示PUMP关闭)或者当标记模式具有如由行911b表示的编码010时(指示写入默认模式)，表900的全部CSR可以仅由以M模式执行的代码可访问。

[0256] 因此，在根据在这里的技术的至少一个实施例，是否允许执行代码访问CSR可以取决于CSR的最小RISC-V特权级别(比如在表900的列904中指定的)、当前标记模式和执行代码的当前RISC-V特权级别。例如，在不考虑标记模式的RISC-V架构中，由于对于所有这种CSR的由904表示的最小特权级别，不允许在U-模式执行的代码访问900中定义的的任意CSR。但是，不考虑标记模式的情况下，允许以至少H模式的特权执行的代码访问除了901r之外的900的全部CSR，且允许以M模式执行的代码访问900的全部CSR。现在考虑根据904的最小RISC-V特权和标记模式确定900的CSR的CSR访问。例如，考虑在H-级别执行的代码部分A。当如由911c表示的标记模式是100时或者当如由911d表示的标记模式是101时，允许代码部分A访问(表900的)CSR 901a-q。但是，可以不允许在S模式执行的代码部分B访问CSR 901a-q，因为代码部分B不具有由用于这种CSR的904中的定义的CSR特权级别指定的最小特权级别。因此，例如，在使用表900中定义的CSR在H级别执行的一个实施例中，代码部分A可以是高速缓存未命中处理器。如第二实例，假定对于CSR 901a-q定义的最小RISC-V特权是SRW(S模式表示为访问这种CSR的最小特权级别)。当如在911c标记模式是100时和当如在911d标记模式是101时，允许在H模式执行的代码部分A访问CSR 901a-q，且当如在911c标记模式是100时允许在S模式执行的代码部分B访问CSR 901a-q。因此，代码部分A或者B可以是高速缓存未命中处理器的代码。

[0257] 在至少一个实施例中，当PUMP关闭时，比如在引导处理的适当部分期间，911a的关闭标记模式可以是当前标记模式。当初始化存储器位置具有相同默认标记(例如，如由CSR

901c表示的)时,911b的默认标记模式可以是当前标记模式。通常,虽然在RISC-V架构中指定4个特权模式,实施例可以替代地使用不同数目的特权模式,其中第一特权级别表示用户模式或者无特权模式,且第二特权级别表示执行的提升的或者特权模式(例如,类似于基于UNIX的操作系统中的核心模式)。在这种实施例中,可以启用PUMP且当在用户或者无特权模式中执行代码时推行策略规则,且当在第二提升的特权模式中执行代码时可以脱离PUMP(例如,PUMP保护关闭或不推行规则)。以该方式,当执行比如未命中处理器的可信或者提升的特权代码以在PUMP规则高速缓存中存储新规则时,实施例可以脱离PUMP。

[0258] 如上所述,实施例可以使用默认传播规则以确定例如,当脱离PUMP时和/或当规则指定不关心PUMP输出新PC标记和R标记时,PUMP输出新PC标记和R标记(例如,这种不关心值可以由当前指令的特定操作码的关心矢量指示)。在一个实施例中,以下可以表示在使用的默认传播规则中具体表现的逻辑。

[0259] • newpctag是用于默认传播的PCtag

[0260] • Rtag是用于CSR读和写操作的CSR;

[0261] • RDtag是分配的RS2tag(CSRtag)

[0262] -允许各标记与各数据值一起交换

[0263] -RDtag←RS2tag←初始CSRtag

[0264] -CSRtag←Rtag←初始RS1tag

[0265] • Rtag是用于CSRR?I、CSRRS、CSRRC的RS2tag(CSRtag)

[0266] -保持CSRtag不变

[0267] -RDtag←RS2tag←初始CSRtag

[0268] -CSRtag←Rtag←初始RS2tag←初始CSRtag

[0269] • Rtag是用于JAL和JALR指令的PCtag(其用于返回地址)

[0270] • Rtag是用于AUIPC指令的PCtag。在RISC-V中,AUIPC(添加上层邻接(upper immediate)到PC)指令用于建造PC相对地址和使用U类型格式。AUIPC形成与20位U-邻接的32位偏移,最低12位以零填充,添加该偏移到PC,然后将结果置于寄存器rd中。

[0271] • Rtag是用于LUI指令的CItag。在RISC-V中,LUI(加载上层邻接)指令用于建造32位常数和使用U类型格式。LUI将U邻接值置于目的地寄存器RD的顶部20位,最低12位以零填充。

[0272] • Rtag是用于非存储器非CSR、非JAL(R)/AUIPC/LUI操作的RS1tag,

[0273] • Rtag是用于存储器写操作的RS2tag

[0274] • Rtag是用于存储器加载操作的Mtag

[0275] 在基于RISC-V架构的在这里的技术的至少一个实施例中,可以对于规则高速缓存未命中发生定义新PUMP未命中陷阱。PUMP未命中陷阱可以具有比虚拟存储器故障或者非法指令更低的优先级。

[0276] 在使用RISC-V架构处理根据在这里的技术的至少一个实施例中,可以维护数据和元数据之间的严格分隔和隔离,其中在标记元数据处理和普通指令处理之间存在分隔和隔离。因此,可以维持元数据规则处理和普通或者典型程序指令执行之间的分开的执行域。可以执行用于与执行代码的指令和数据相关联的标记使用PUMP执行的元数据处理。陷阱中的PUMP规则高速缓存未命中结果导致控制转移到高速缓存未命中处理器,规则高速缓存未命

中处理器生成或者提取匹配当前指令的规则并在PUMP规则高速缓存中存储该规则。信息可以使用CSR在以上提到的执行域之间通信。当从执行程序的指令执行域切换到元数据规则处理域时(比如当经由规则高速缓存未命中陷阱触发规则高速缓存未命中处理器时),可以使用CSR提供标记及与指令(导致该陷阱)相关的其他信息作为到PUMP以及未命中处理器的输入。以类似的方式,当控制从元数据规则处理域转移执行程序的指令执行域时(比如当在处理规则高速缓存未命中陷阱之后从规则高速缓存未命中处理器返回时),可以使用CSR来传递PUMP输出,其中CSR的内容然后存储在指令执行域中的相应的映射寄存器中。按照在这里的讨论,不映射到规则的指令(例如,没有该指令的匹配规则位于高速缓存中,且高速缓存未命中处理器确定对于当前指令不存在这种匹配规则)指示不允许执行该规则由此触发陷阱或者其他事件。例如,处理器可以停止当前程序代码的执行。

[0277] 以该方式,即使相同RISC-V处理器和存储器可以用于两个域中,可能存在前述域和关联的数据路径之间的严格分隔。使用在这里的技术,不允许执行代码的指令读取或者写入元数据标记或者规则。包括标记指令和数据的全部元数据转换可以通过PUMP进行。类似地,可以仅由元数据子系统的规则高速缓存未命中处理器执行到PUMP高速缓存中的规则插入。关于由元数据子系统或者处理系统执行的处理,执行代码的元数据标记被置于PUMP CSR中,且成为输入到且由元数据系统在上面操作的“数据”(例如,指针到元数据存储器空间中)。元数据子系统经由PUMP输入CSR读取PUMP输入以用于根据规则的处理。如果允许指令经由规则进行,则PUMP写入标记结果(例如,比如对于PC新和R标记)到定义的PUMP输出CSR。可以响应于到特定CSR(例如,比如901q中的srtag CSR)的写入触发到规则高速缓存中的规则插入。以该方式,通过PUMP中的规则进行全部标记更新且由元数据子系统控制。仅元数据子系统可以经由在发生规则高速缓存未命中时调用的高速缓存未命中处理器将规则插入到PUMP高速缓存中。另外,在使用RISC-V架构在这里描述的至少一个实施例中,可以维持元数据处理和一般指令处理之间的前述分隔而不增加超出“RISC-V用户级ISA”和“RISC-V特权ISA”中的那些指令的任何新指令。按照在这里其他地方的讨论,根据在这里的技术的实施例可以维持数据和元数据之间的严格分隔和隔离,由此在基于标记的元数据处理和一般指令处理之间存在分隔。在至少一个实施例中,可以通过使得分开的物理元数据处理子系统具有分开的处理器和分开的存储器来维持这种分隔。因此,当执行程序的处理指令时可以使用第一处理器和第一存储器,且第二处理器和第二存储器可以包括在元数据处理子系统中,用于比如当执行规则高速缓存未命中处理器的代码时,执行元数据处理来使用。

[0278] 参考图27,示出了可以在根据在这里的技术的实施例中包括的组件的实例1000。实例1000包括关于用于执行程序的一般处理使用的第一子系统或者处理器1002和元数据处理子系统或者处理器1004。第一子系统1002可以特性化为关于一般程序执行使用的程序执行子系统。子系统1002是包括关于执行程序代码和使用其中这种代码和数据包括如在这里其他地方描述的由元数据处理子系统1004使用的标记的数据使用的组件。子系统1002包括存储器1008a,指令或者I-存储1008b,ALU(算术及逻辑单元)1008d和程序计数器(PC)1008e。应当注意,PUMP 1003可以关于子系统1002中代码的执行使用,但是可以被认为是元数据处理子系统1004的一部分。可以标记子系统1002中的全部代码和数据,比如一般地由与数据1002b相关联的标记1002a图示的,其中1002a和1002b可以存储在存储器1008a中。类似地,元素1001a表示关于PC 1008e的指令的标记,1001b表示指令1008b的标记,1001c表示

存储器位置1008a的标记，且1001d表示寄存器1008c的标记。

[0279] 元数据处理子系统1004是包括使用当前指令的标记和作为到PUMP1003的输入提供的关联数据的关于元数据规则处理使用的组件的处理器(也称为元数据处理器)。PUMP 1003可以如在这里其他地方描述的且包括规则高速缓存。例如，在至少一个实施例中，PUMP 1003可以包括图22中图示的组件。PUMP 1003的组件的更多具体图示和实例，用于PUMP输入和输出的关联的PUMP CSR和可以在根据在这里的技术的至少一个实施例中包括的关联逻辑以下和在这里其他地方更加详细地描述。子系统1004是用于元数据处理的单独的处理器且包括类似于子系统1002中的那些组件。子系统1004包括存储器1006a，I-存储1006b，寄存器文件1006b和ALU 1006d。存储器1006a可以包括关于元数据规则处理使用的元数据结构。例如，存储器1006a可以包括由作为指针的标记指向的结构或者数据。指针标记和由指针标记指向的结构/数据的实例在这里其他地方描述，比如关于CFI策略描述。I-存储1006b和存储器1006a可以包括指令或者代码，比如执行元数据处理的未命中处理器。元数据处理器1004不需要访问1002的其他组件，比如关于程序执行使用的数据存储器1008a，因为元数据处理器1004仅执行元数据处理(例如，基于标记和规则)。子系统1004包括它自己的组件，比如单独的存储器1006a，且不需要存储子系统1002中的元数据处理代码和数据。而是，任何信息，比如可以由PUMP 1003使用的当前指令的标记，被作为输入提供给元数据处理子系统1004(例如，PUMP输入1007)。

[0280] 实例1000图示具有单独的元数据处理子系统1004的替换实施例，而不是在如在这里其他地方描述的对一般程序执行使用的相同子系统上执行元数据处理。例如，不是启用单独的元数据处理器或者子系统1004，而是实施例可以仅包括PUMP 1003和子系统1002。在具有单个处理器的这种实施例中，CSR可以如在这里描述的使用以在元数据处理和执行用户程序的普通处理模式之间传递信息以由此提供隔离和分隔。在具有单个处理器而不是单独的元数据处理器的这种实施例中，未命中处理器的代码可以用使其受保护的方式来存储在该单个存储器中。例如，没有单独的元数据处理器或者子系统，可以使用如在这里其他地方描述的标记来保护未命中处理器的代码以限制访问，未命中处理器的代码可以映射到不可由用户代码等访问的存储器的部分。

[0281] 现在将描述的是关于PUMP I/O(输入/输出)的更多细节。应当注意，以下描述的PUMP I/O应用于可以使用与一般代码执行相同的处理器或者子系统的PUMP的实施例以及可以使用单独的处理器或者子系统的实施例，比如在实例1000中的。此外，以下描述的PUMP I/O可以与基于RISC-V架构的实施例一起使用，且可以通用化以用于其他处理器架构。

[0282] 参考图28，示出了在根据在这里的技术的实施例中概述PUMP I/O的实例1010。如在这里其他地方，比如关于图1和图24描述的，PUMP在级5和6中操作。关于一般PUMP验证使用PUMP输入(例如，验证是否使用策略规则允许当前指令)以在用于当前指令的PUMP的规则高速缓存中找到匹配规则，如果有的话。一般PUMP验证可以对于每个指令发生，比如在级5的部分，如在这里其他地方以6级流水线描述的。另外，可以关于控制到规则高速缓存中的规则插入来使用PUMP输入，比如可以在6级流水线的级6中发生。与一般PUMP验证相关联的PUMP I/O由从顶部(输入1012)到底部(输出1014)在垂直方向上的输入和输出在实例1010中表示。与控制到PUMP规则高速缓存中的规则插入相关联的PUMP I/O由从左(输入1016)到右(输出1018)在水平方向上的输入和输出在实例1010中表示。另外，元素1012表示也关于

规则插入使用的附加输入，如其他地方更详细地描述的。

[0283] 首先，考虑与一般PUMP验证处理相关联的PUMP I/O。PUMP输入1012可以包括标记，比如PC标记、CI标记、指令操作数标记(例如，OP1标记、OP2标记或者CSR标记(用于RISC-V中的基于CSR的指令)、OP3标记、M标记(用于存储器指令的存储器位置标记。注意到。Mtag在这里也可以被称为用于存储器指令的MR标记)、操作码信息(例如，由Opgrp输入、用于扩展操作码的用于RISC-V的funct12(funct7)输入表示的操作组，提供指示符的subinstr输入(该指示符的指令是包括多个指令的指令字中的当前指令，比如在实例200和220中)和关心输入位。Opgrp可以是用于当前指令的操作组，其中Opgrp可以是在前级(例如，级3或者级4)的输出，如在这里其他地方描述的。Funct 12(funct 7)PUMP输入可以是使用指令字(例如，实例400)的附加位用于那些RISC-V操作码的附加操作码位，如果有的话。PUMP输出1014可以包括Rtag(例如，用于指令结果寄存器或者目的地存储器位置的标记)，PC新标记(表示置于用于下一指令的PC上的传播标记)，和表示是否存在导致级6中的未命中处理器器的陷阱的PUMP规则高速缓存未命中的指示符1014a。

[0284] 关心位1012a可以表示哪个PUMP输入1012和哪个PUMP输出1014是对于特定指令关心/不关心(例如，忽略)的。关于PUMP输入的关心位可以包括用于funct12的关心位和用于funct7的第二关心位。如在这里其他地方描述的，前述关心位都表示当前指令的特定操作码是否包括用于RISC-V指令的扩展的12操作码位部分的任何位(例如，实例400的404a)。如果funct12和funct7关心位都是“不关心”，则掩蔽扩展的12操作码位部分的全部12位。如果funct7指示“关心”，则掩蔽扩展的12操作码位部分的全部底部5位。如果funct12指示“关心”，则不掩蔽扩展的12操作码位部分。

[0285] 现在考虑与控制到PUMP规则高速缓存中的规则插入相关联的PUMP I/O。PUMP输入1016可以结合关于PUMP高速缓存规则插入的输入1012来使用。PUMP输入1016可以包括Op 1数据(从元数据处理器或者子系统的输出)，指令(来自级6)和标记模式(从元数据处理器或者子系统的输出)和特权(表示RISC-V特权的priv)。标记模式和1016的priv输入由元数据处理器或者子系统使用以确定在元数据处理器中执行的比如未命中处理器或者其他代码的代码是否具有提供各种输入到元数据处理器(例如，比如输入1012)的访问的以下与在这里其他部分描述的CSR的足够特权。Rdata1018是用于级6中的到元数据处理器或者子系统的输入(例如，高速缓存未命中处理器处理输入)。应当注意，Op 1数据、R数据及实例1010的其他项目在以下段落和图中更加详细地描述。

[0286] 因此一般地，在实例1010中，元素1012表示从执行用户代码的处理器(例如，非元数据处理器或者比如1002的子系统)到PUMP和元数据处理器的输入，元素1014表示由元数据处理器生成的输出，元素1016表示到输入给PUMP的由元数据处理器生成的输出，且元素1018表示到元数据处理器的输入。

[0287] 参考图29，示出了在根据在这里的技术的实施例中概述关于操作组/关心表(例如，实例420的元素422)的I/O的实例1020。如在这里其他地方描述的，操作组/关心表可以用于每一指令以查找和输出用于当前指令的操作码的操作组和关心位。I/O的该第一流动在1020中通过从顶部(输入1022)到底部(输出1024)的垂直方向上的输入和输出图示。如在这里其他地方描述的，输入1022可以是用于到操作码/关心表中的索引的操作码或者其一部分(例如，比如关于实例420中的操作码部分的实例描述的)。输入1022可以来自级3。输出

1024可以是用于特定操作码的操作组(opgrp)和关心位。输出1024是到级5的输入(例如,如1012中包括的PUMP输入oprgrp和关心中的两个)。

[0288] I/O的第二流动在1020中通过从左(输入1026)到右(输出1028)的水平方向上的输入和输出图示。I/O的第二流动中1020中图示了关于控制作为到元数据处理器或者级6的输入的PUMP输出Rdata 1028的选择执行的处理。输入1026如上关于1016所述。输出1028如上关于1018所述。

[0289] 参考图30,示出了理论上表示由在根据在这里的技术的实施例中的PUMP执行的处理的实例1030。实例1030包括对应于用于上面关于实例1010中的水平PUMP I/O流动(例如,元素1012、1016和1018)描述的规则插入的PUMP控制的PUMP控制1031。实例1030包括掩蔽1032、散列1034、规则高速缓存查找1036和输出标记选择1038,其对应于对于如上关于实例1010中的垂直PUMP I/O流动(例如,元素1012和1014)所述的每一指令执行的一般PUMP验证路径I/O流动。掩蔽1032表示将1012的关心位施加到1012的掩蔽的未使用的PUMP输入。散列1034表示在由1036表示的规则高速缓存查找期间使用的散列的计算。关于图22图示和描述了在一个实施例中可以用于实现由1032、1034和1036表示的逻辑的组件。输出标记选择1038表示基于关心矢量位(输入1012中包括的关心)和htagmode CSR(表示当前标记模式)的如1014中包括的PUMP输出Rtag和PC新标记的选择。

[0290] 参考图31,示出了表示可以用于实现在根据在这里的技术的实施例中的PUMP的输出标记选择1038的逻辑的组件的实例1040。实例1040包括多路复用器(MUX)1043a-b。一般地,MUX 1043a可以用于选择PC新标记1043的最终标记值作为PUMP的输出(例如,1014的PC新标记),且MUX 1043b可以用于选择R标记1047的最终标记值作为PUMP的输出(例如,1014的R标记)。元素1042表示用作MUX 1043a的选择器的输入。输入1042用于选择1041a或者1041b作为PC新标记1043。输入1042可以包括与启用(engaged)(表示是否启用PUMP的布尔)的逻辑与(&&)的PC新标记关心位(例如,来自1012的关心位)。元素1043表示用作MUX 1043b的选择器的输入。输入1043用于选择由1045a-1045b表示的输入之一作为R标记1047。输入1043可以包括与启用逻辑与(&&)的Rtag关心位(例如,形成1012的关心位)。因此,一般地PUMP输入1012中包括的关心位标识哪个PUMP输入不关心(被掩蔽)和哪个PUMP输出(Rtag和PC新标记)不关心(被掩蔽)。此外,当脱离PUMP时输出1043和1047被当做“不关心”值,因为处理器在比指定为PUMP操作的阈值的当前标记模式更高的特权级别运行。

[0291] 元素1049表示布尔启用怎样被确定为当前RISC-V特权和当前标记模式的函数。元素1049包括使用现有技术中已知的标准符号的逻辑表达式,由此“A＝＝B”表示是否逻辑测试在A与B之间相等,“A&&B”表示A和B的逻辑与操作,且“A||B”表示在A与B之间的逻辑或操作。

[0292] 元素1041a和1045a表示作为从规则高速缓存查找1036的输出的到1043a的输入。PC标记1041b是PUMP输入1012中包括的PC标记。其他输入1041b一般表示可能选择作为由PUMP输出的最终R标记1047的多个其他输入。例如,在一个实施例中,其他输入1041b可以包括M标记、PC标记、CI标记、OP1标记、OP2标记、OP3标记和取决于指令的可能的其他标记。特定的R标记输出1047可以随特定的RISC-V指令/操作码而变。

[0293] 以下可以概述在一个实施例中作为PUMP输出值生成的R标记1047和PC新标记1043的特定值。应当注意,以下指示用于不同RISC-V指令的特定的R标记输出值。因此,作为最终

PUMP R标记值输出的特定R标记值可以随关于后续元数据处理利用这种PUMP输出的指令而变。

[0294] 1.当对于PC新标记,输出关心位关闭时PCtag不改变

[0295] 2.Rtag是用于CSRRW操作的Opltag

[0296] 3.Rtag是用于CSRR?I、CSRRS、CSRRC操作的Op2tag(CSRtag)

[0297] 4.Rtag是用于JAL和JALR指令的PCtag

[0298] 5.Rtag是用于AUIPC指令的PCtag

[0299] 6.Rtag是用于LUI指令的CItag

[0300] 7.Rtag是用于当输出关心位关闭(指示Rtag的关心)时非存储器、非CSR、非JAL(R)/AUIPC/LUI操作的Opltag

[0301] 8.Rtag是当输出关心位关闭时用于存储器写操作的Op2tag

[0302] 9.Rtag是当输出关心位关闭时用于存储器加载操作的Mtag

[0303] 参考图32,示出了在根据在这里的技术的实施例中可以用于控制PUMP I/O的组件的实例1050。一般地,再次参考实例1030,1050的组件可以包括逻辑上在1032顶部的另一层(例如,与图22的组件对接)。元素M1-M14表示用于到其的各种输入的选择的多路复用器。元素1052一般表示来自用于当前指令的1012的输入操作码、PC标记、CI标记、Op1标记、Op2标记、Op3标记和M标记。元素1056一般是指用于存储多路复用器M1-M7的所选的输出的寄存器的行。在一个实施例中,基于RISC-V架构,行1056中的每个框可以是寄存器,且具体来说,CSR包括如在这里其他地方描述的特定值(例如,表示可以用于一个实施例中的CSR的实例900)。

[0304] 应当注意,实例1050的元素1052不包括1012的所有输入。例如,为了简单在实例1050中不图示1012的funct12(funct7)和subinstr输入。但是,本领域技术人员理解来自1012的输入funct12(funct7)和subinstr也可以包括在1052中。通常,输入1052可以适于到可以用于实施例中的元数据规则处理的特定输入。

[0305] 当PUMP执行用于一般PUMP验证的指令的处理时(例如,验证是否使用策略规则允许当前指令),可以简单地通过输入1052作为输出1054。在该情况下的输出1054流过到PUMP,作为输入,比如到用于元数据处理的到图22的组件的输入(或更一般地通过到元数据处理器或者子系统)。通过一般PUMP验证,PUMP然后可以产生输出1014(例如,如果在规则高速缓存找到用于当前指令的匹配规则,则Rtag和PC新标记,且否则生成高速缓存未命中1014a)。

[0306] 在规则高速缓存未命中的发生时,作为第一步骤,用于当前指令的来自1052的当前值加载到1056的寄存器G1-G7中。因此,G1-G7包括导致了规则高速缓存未命中的用于当前指令的操作码和标记值的快照,且现在可以通过高速缓存未命中处理器读出这种处理需要的G1-G7的一个或多个期望值,对于后续处理使用这种值。

[0307] 因此,在第二步骤中,高速缓存未命中处理器执行,读取来自G1-G7的值作为输入,并生成用于当前指令的新规则。多路复用器M16可以用于控制来自G1-G7的各种可能输入的选择,其中来自M10的所选的输出表示为用于由高速缓存未命中处理器处理的R数据1053(例如,其可以在与当执行程序代码时相同的处理器上执行或者否则可以在比如在实例1000中的单独的元数据处理器上执行)。给定导致规则高速缓存未命中的用于当前指令的

输入G1-G7,高速缓存未命中处理器执行处理以确定要插入到高速缓存中的新规则。高速缓存未命中处理器生成用于刚刚确定的新规则的输出R标记和PC新标记,将Rtag写入到Rtag CSR G8,且将PC新标记写入到PC新CSR G9。在实例1050中,Op1数据1051表示由元数据处理器生成的输出,比如用于新规则的输出Rtag和PC新标记,其中这些输出然后如描述的存储在CSR G8和G9中。

[0308] 此时,CSR G1-G9中的值是用于刚刚由高速缓存未命中处理器生成的新规则的标记值,且可以被插入/写入到规则高速缓存作为第三步骤中的新规则。在使用具有RISC-V架构的在这里的技术的至少一个实施例中,写入到由G8表示的R标记CSR触发新规则(例如,CSR G1-G9的内容)到规则高速缓存的写入。关于规则插入,提供CSR G1-G7作为输出1052,且提供CSR G8和G9作为输出1055,给PUMP用于存储到规则高速缓存中。更具体地,在一个实施例中,输出1052和1055可以提供给图22的组件用于规则插入。

[0309] 在简单情况下,实施例可以通过如刚刚描述的将用于新规则的CSR G1-G9的内容写入到PUMP规则高速缓存(例如,经由输出1052和1055)而插入一个新规则以满足当前规则未命中。在这种实施例中,不需要多路复用器M1-M7,因为由执行高速缓存未命中处理器的元数据规则处理器输出的Op1数据1051仅生成用于新规则的R标记和PC新标记。但是,实施例也可以允许规则预取或者多个规则插入到规则高速缓存中。例如,在规则高速缓存未命中的发生时,高速缓存未命中处理器可以确定要写入/插入到规则高速缓存中的多个规则而不是仅用于当前指令的单个新规则。在该情况下,Op1数据1051可以包括用于操作码、PCtag、CItag、Op1tag、Op2tag、Op3tag和Mtag的附加新值(写入到CSR G1-G7)以及用于Rtag和PC新标记的新值(如写入到CSR G8和G9)。在这种情况下,多路复用器M1-M7可以用于从Op1数据1051选择前述新值分别作为用于CSR G1-G7的输入。

[0310] 一般地,Op1数据1051表示从元数据处理器到PUMP的输出,且R数据1053表示从PUMP到元数据处理器的输出。此外,元素1052表示从执行用户代码的处理器到PUMP的输入(例如,作为一般指令处理的一部分),其中用于元素1054的值等于当执行一般PUMP验证(例如,验证是否使用策略规则允许当前指令)时在1052中的那些值。

[0311] 参考图33,示出了图示与在具有含有分支预测的RISC-V架构的根据在这里的技术的一个实施例中的6级处理器流水线结合的PUMP处理级的实例1060。实例1060图解6级流水线,具有:包括取出要执行的下一指令(例如,在I高速缓存1063a中存储取出的指令)和分支预测的级1,表示解码指令级的级2,包括从寄存器获得值(例如,寄存器读取)和当前指令的分支解析的级3,包括指令执行的级4(例如,执行快速ALU操作和启动比如浮点(FP)、整数乘法和除法的多级操作),包括接收对多级操作的响应和请求存储器操作数的级5,和包括确认指令(例如,存储到目的地和在数据高速缓存1063b的结果,如由1069表示的),和处理例外、陷阱和中断的级6。也在实例1060中示出PUMP处理级。元素1062指示可以在级3中执行opgrp/关心表查找,提供其输出1062a作为在级4中到PUMP散列1064的输入。到PUMP散列1064的其他输入包括Mtag 1061(例如,作为当前指令的操作数的存储器位置的标记)及其他标记值1062b,由此输入1061和1062a-b用于确定表示PUMP规则高速缓存1066中的高速缓存地址或者位置的输出1064a。指令操作数、PC、当前指令等的其他标记值1062b的实例在这里其他地方描述且可以关于确定当前指令的规则高速缓存1066中的位置(例如,图22)使用。元素1068表示基于来自级5的PUMP处理的输出1066a的高速缓存规则未命中检测。输出

1066a可以包括关于是否存在当前指令的规则高速缓存未命中的指示符。如果1066a报告潜在命中，则1068确定命中是真实命中或者虚假命中，将虚假命中转为未命中。元素1066b表示在没有规则高速缓存未命中且存在匹配当前指令的高速缓存中的规则的情况下到级6的PUMP输出。输出1066b可以包括PC新标记和R标记。应当注意，可以改变实例1060的PUMP级。例如，操作组/关心查找1062可以在级4而不是级3执行，且PUMP规则高速缓存位置的确定和查找两者都在级5中完成(例如，取决于特定的PUMP规则高速缓存实现)。

[0312] 关于非存储器操作，不需要Mtag作为到PUMP级的输入且PUMP可以在没有该输入的情况下继续执行处理。在存储器操作指令的情况下，PUMP停止直到已经从存储器提取了Mtag。替代地，实施例可以执行如在这里其他地方描述的Mtag预测。按照在这里其他地方的讨论，需要提供PC新标记回到级1，比如关于图1图示和描述的。只要指令确认，PC新标记就是下一指令的适当的PC标记。如果不确认当前指令(例如，没有规则高速缓存命中)，则由规则高速缓存未命中处理器确定PC新标记(如同传回到级1)。当陷阱处理器开始或者执行上下文切换(例如，PC恢复)时，标记来自于所保存的PC。

[0313] 如在这里描述的，实施例可以将单个标记与每个字关联。在至少一个实施例中，与每个标记相关联的字长可以是64位。标记的字的内容例如可以包括指令或者数据。在这种实施例中，单个指令的大小。但是，实施例也可以支持是除了64位之外的不同大小的指令。例如，实施例可以基于RISC-V架构，且如在这里其他地方更具体地描述的，是基于建立的精简指令集计算(RISC)原理的开源指令集架构(ISA)。使用RISC-V架构的实施例可以包括多个不同大小的指令，例如。32位指令以及64位指令。在这种情况下，根据在这里的技术的实施例可以将单个标记与单个64位字关联，其中单个字因此可以包括一个64位指令或者两个32位指令。

[0314] 参考图34，示出了可以与在根据在这里的技术的实施例中的指令相关联的标记的实例200。元素201图示以上提到的其中单个标记202a与单个指令204a相关联的情况。在至少一个实施例中，202a和204a中的每一个的大小可以是64位字。元素203图示也在以上提到的替代，其中单个标记202b与两个指令204b和204c相关联。在至少一个实施例中，202b的大小可以是64位字，且指令204b和204c每个可以是与标记202b相关联的同一64位指令字205中包括的32位指令。通常，应当注意取决于实施例中使用的指令大小，在单个标记的指令字中可能有多于2个指令。如果如元素203所示，标记的粒度不匹配指令的粒度，则多个指令与单个标记相关联。在有些情况下，同一标记202b可以用于指令204b和204c中的每一个。但是，在有些情况下，同一标记202b可以不用于指令204b、204c中的每一个。在以下段落中，与单个标记的字相关联的单个指令字中包括的比如204b和204c的多个指令中的每一个也可以被称为子指令。

[0315] 因此现在将描述可以关于同一指令字中的多个子指令用于实施例中的技术，由此可以关于多个子指令中的每一个使用不同标记。

[0316] 参考图35，示出了可以用于根据在这里的技术的实施例的子指令和标记的实例。实例220包括单个64位子指令字205，且包括两个32位子指令204b和204c。标记202b可以是如上在实例200中所述的关于子指令字205的标记。在根据在这里的技术的至少一个实施例中，子指令字205的标记202b可以是到另一存储器位置222的指针221，其包括一对标记，该对包括用于指令字205的子指令204b-c中的每一个的标记。在该实例220中，该对标记包括

表示用于子指令1 204b的第一标记tag1的222a，且还包括表示用于子指令2 204c的第二标记tag2的222b。在至少一个实施例中，该对222的每个标记222a-222b可以是非指针标记(例如，标量)，可以是包括由PUMP使用以用于如在这里描述的处理的信息的到又一存储器位置的指针标记，或者要不然可以是包括一个或多个非指针字段的更复杂结构和/或一个或多个非指针字段。例如，标记1 222a可以是用于子指令1 204a的指针标记，且tag2 222b可以是用于子指令2 204b的指针标记。如在220所示，元素223a表示包括由PUMP使用以用于处理子指令1 204b的信息而指向或者标识另一存储器位置224a的tag1 222a，且元素223b表示包括由PUMP使用以用于处理子指令2 204c的信息而指向或者标识另一存储器位置224b的tag2222b。应当注意，取决于实施例和子指令，224a和224b中的每一个可以是非指针，可以是到存储器位置的又一指针，或者可以是包括一个或多个指针和一个或多个非指针的某些组合的复杂结构。

[0317] 在同一子指令字205内具有多个子指令的实施例中，可以将附加输入提供给PUMP，指示在时间点正在执行子指令字205中包括的哪个子指令。例如，在指令字205中存在2个子指令204b-c时，到PUMP的附加输入可以是0或者1，分别指示在特定时间点正在执行子指令1 204b或者子指令2204c。在基于RISC-V架构的按照在这里其他地方的讨论的至少一个实施例中，CSR(比如在这里其他地方描述的ssubinstr CSR)可以被定义哪个记录或者存储到PUMP的附加输入(表示正在执行哪个子指令)。在至少一个实施例中，PUMP通常可以从数据路径接收前述附加输入(例如，从代码执行域)而不使用CSR。但是，规则未命中时，前述附加输入可以记录在CSR中，以使得其中执行规则未命中处理器的元数据处理域可以获得前述附加输入(例如，用于前述附加输入的CSR值提供给关于规则插入的PUMP)。

[0318] 为进一步图示，实施例可以包括提供用于程序中的两个位置之间的控制转移的子指令。这种子指令的实例可以是提供跳转、分支、返回或更一般地用于从代码的源位置到代码中的目标位置(例如，信宿或者目的地)的转移控制的那些子指令。关于在这里其他地方描述的CFI或者控制流动完整性，可以期望具有CFI策略的PUMP实现规则以限制或者控制位置之间的转移仅到由程序支持的那些。例如，考虑从具有标记T1的代码中的源位置到具有标记T2的代码中的目标位置做出控制转移的情况。由PUMP在推行CFI策略时使用的信息可以是允许将控制转移到T2的有效源位置的列表。在CFI策略的实施例中，两个规则可以用于提供当从源位置到目标位置转移控制时两个指令或者操作码的两个校验。考虑如图36的实例230中所示的转移或者调用的伪代码表示。在实例230中，可以进行调用将控制231a从foo例程231中的源位置转移到例程bar 233中的目标位置。特别地，控制可以从具有标记T1的源位置X1 232转移231a到具有标记T2的目标位置X2 234。目标位置X2可以是例程条(bar)的代码233a的主体内的第一指令。CFI策略的规则可以用于以从232到234的转移被允许或者有效来校验。在至少一个实施例中，可以每次执行校验使用CFI策略的2个规则以保证控制从232到234的转移有效。在源位置X1的指令是控制从其转移到目标的分支点或者源点。在源(例如，在源位置X1 232执行指令之前)，第一规则可以用于标记或者设置PC的标记以表示源位置。例如，第一规则可以标记或者设置PC的标记为地址X1以表示源位置。随后，在目标位置X2 234执行指令之前，第二规则可以用于校验源位置X1是否是允许控制从其转移到目标位置X2的有效源位置。

[0319] 在至少一个实施例中，可以通过确定标识源位置232(例如，表示源位置地址X1)的

PC的标记的标记(如由第一规则设置)是否标识控制可以从其转移到目标位置234的有效源位置,而执行第二规则的校验。在这种实施例中,第二规则可以提供有表示允许控制转移到目标位置234的全部有效源位置的定义列表。在至少一个实施例中,定义列表例如可以通过它们的地址,比如以上提到的X1来标识有效源位置。

[0320] 参考图37,示出了图示在根据在这里的技术的实施例中的可以关于源和目标位置的子指令使用的标记的实例240。实例240包括表示指定用于如上所述的单个指令字的2个子指令204b-c的单个标记202b的元素203。关于指令字的标记202b可以指向表示分别用于两个子指令204b-c的两个标记242a-b的标记对242。两个标记242a-b中的每一个通常可以是到由PUMP规则使用以用于CFI验证的信息的指针,该信息根据与两个标记242a-b中的每一个相关联的特定的子指令与源或者目标位置相关。

[0321] 实例240图示两个子指令204b-c是目标位置的一个实施例中的结构。子指令标记242a指向243a包括源id字段245a和允许源设置字段245b的结构245的位置。源id字段245a在子指令204b不是源位置的情况,比如这里子指令204b是目标位置的情况下可以为空。源设置字段245b可以是到包括标识允许转移控制到包括子指令204b的特定目标位置的一个或多个有效源位置的列表结构247的位置的指针。在至少一个实施例中,列表结构247可以包括表示有效源位置的大小或者数目的第一元素。因此,“n”的大小247a(n是大于0的整数)表示由列表247中的元素247b-n表示的源位置的数目。元素247b-n中的每一个可以标识可以转移控制到包括子指令204b的目标位置的不同有效源位置。在至少一个实施例中,允许源247b-n中的每一个可以是标量或者非指针,即,例如,是有效源位置之一的地址。

[0322] 在实例240中,对于子指令2 204c使用的元素243b、246和248分别类似于对于子指令1 204b使用的元素243a、245和247。通常,在这种使用240的结构的实施例中,可以向不存在的任何项目分配空或者零值。如果指令字205包括既不是源又不是目的地位置的一对子指令204b-c,则标记202b可以为空(例如,或者要不然标识非指针或者不指向结构242的其他指针)。如果子指令204b-c之一既不是转移的源又不是目标位置,则其在242中的关联标记为空。例如,如果子指令204b既不是源又不是目标位置但是子指令204是目标,则242a可以为空且242b可以如实例240中所示。如果子指令204b-c不是源位置,则其源id为空(例如,因为实例240中的204b-204c是目标位置,245a和236a两者都为空)。如果子指令204b-c不是目标位置,其允许的源设置字段指针为空。例如,如果子指令204b标识源位置而不是目标位置,则源id 245a将标识源位置指令的地址且245b将为空。

[0323] 为进一步图示,参考图38的另一实例250,其使用比如实例240中描述的结构,差异在于第一子指令251a既不是源又不是目标位置,且第二子指令251b是可以从任意3个有效源位置转移控制的目标位置。元素251a-b可以表示具有标记251的单个标记的字中包括的两个32位子指令。标记251可以是包括结构252的标识存储器中的位置的指针1228,具有用于子指令251a-b的一对标记。元素252可以类似于实例240的242。元素252a可以是用于子指令251a的标记指针,且元素252b可以是用于子指令251b的标记指针。因为子指令251a既不是源又不是目标位置,所以252a如由零表示的为空。因为子指令251b是目标位置,252b是到结构254的指针1238。元素254可以类似于实例240的246。元素254a是源id字段(类似246a)且元素254b是包括到允许源设置结构256(类似实例240的248)的指针(地址1248)的允许源设置字段(类似246b)。因为子指令251b仅是目标位置而并非源,254a源id为空。元素256可

以类似于实例240的248。元素256a可以是表示有效源位置的数目的大小字段(类似248a)。元素256b-d可以表示有效源id,其例如可以是有效源位置指令的地址。在该实例中,256a指示存在具有分别在项256b-d中存储的地址50bc、5078、5100的3个有效源位置。关于前述,应当注意通常指令可以是目标和源两者,以使得是目标不意味着源id总是为空。例如,如果指令是目标和源两者,则源id将不为空,且指令的标记将包括可允许/允许源的列表。

[0324] 应当注意,比如在项256b-d中的源位置的地址,且更一般地在允许源集合的任何允许源(例如,实例240的248的任意248b-n)中的源位置的地址可以是字节级地址粒度。

[0325] 以类似于刚刚对于单个标记的字中包括的多个指令(也称为子指令)描述的方式,实施例可以允许访问小于数据的单个标记的字的数据部分。例如,实施例可以包括以字节级访问数据的指令,且可以期望提供字节级标记以使得每个字节可以以类似于对于单个标记的字中包括的多个子指令中的每一个提供不同标记的方式具有它自己的关联标记。在以下实例中,描述提供字节级标记,其中64位字中包括的8字节中的每一个可以具有它自己的关联标记。但是,更一般地,在这里的技术可以用于提供用于单个标记的字中包括的任意数目的多个数据项的子字标记。在此情况下,与标记的数据字相关联的标记可以是到标识用于标记的数据字的字节的字节级标记的结构的指针。

[0326] 参考图39,示出了可以用于根据在这里的技术的实施例的字节级标记的实例260。元素262表示与标记的64位字265相关联的标记262a,其中字265包括表示为B1-B8的8个字节。标记262a可以是指向261结构266的存储器位置的指针,结构266包括用于数据字265的字节B1-B8中的每一个的标记。结构266可以包括作为指示结构中的剩余项的数目的大小字段的第一字段265a。结构中的每个后续项可以包括标记值且表示具有特定标记值的字265的一个或多个字节。在该实例中,大小265a是8,其中265的字节B1-B8中的每一个具有不同标记值。元素266a-h分别表示字265的B1-B8的标记值。

[0327] 参考图40,示出了可以用于根据在这里的技术的实施例的字节级标记的第二实例267。元素262表示与标记的64位字265相关联的标记262a,其中字265包括表示为B1-B8的8个字节。标记262a可以是指向268a结构268b的存储器位置的指针,结构268b包括用于数据字265的字节B1-B8中的每一个的标记。结构268b可以包括作为指示结构中的剩余项的数目的大小字段的第一字段265b。因此,265b类似于图39的265a。结构268b中的每个后续项可以包括标记值且表示具有特定标记值的字265的一个或多个字节。在该实例中,大小265b是7,表示7个后续项266a-226f和268c。如关于图39的实例260描述元素266a-f。元素268c指示标记7是用于字节B7和B8两者的标记。因此,结构268b包括的项比图39的结构266少一个项,因为在实例267中,字节B7和B8两者具有相同的标记值,即标记7。以该方式,由数据字的标记(例如,262a)指向的结构(例如268b)可以具有按照需要取决于特定字节级标记的变化数目。

[0328] 应当注意,数据访问粒度的特定级别可以随在实施例中的特定架构和指令集而变。前述可以用于提供在允许字节级数据访问的实施例中的字节级标记。作为变型,实施例可以支持在不同粒度级别的数据访问,且在这里的技术可以容易地扩展至粒度的任何子字标记级别。

[0329] 类似地,实例260和267图示可以用于保持字节级或者其他子字数据标记的数据结构的一个实例。作为变型,实施例可以使用树或者其他分级结构以指定用于单个标记的数

据字的字节的字节级标记。表示字节级标记的树或者其他分级结构可以类似在这里描述的用于存储字级别标记的分级结构，例如，关于分别在这里其他地方描述的图78-81的元素100、120、130和140。

[0330] 为进一步图示，实施例可以使用树结构表示字节级标记，如在图41的实例270中。在实例270中，元素262可以表示与包括字节B1-B8的标记的字265相关联的标记262a。标记262a可以是到树结构的指针或者地址，表示B1-B8 265的字节级标记。例如，标记262a可以指向树结构的根节点272的位置。在该实例中的树结构可以包括级别1的根节点272，在级别2的节点274a-b，在级别3的节点276a-d和在级别4的节点278a-h。树的每个节点可以与一个或多个字节的字节范围相关联。树的叶子可以表示字节B1-B8的字节级标记。因此树的非叶子节点不指定标记值而是指示需要查询在一个或多个下层的一个或多个后继节点以确定用于与非叶子节点相关联的字节范围的字节级标记。叶子节点可以表示用于多个字节265的范围的同类的或者相同的标记值。每个非叶子节点可以包括到非叶子节点的左子节点的左指针和到非叶子节点的右子节点的右指针。父节点的每一个子节点可以表示与父节点相关联的字节范围的划分。

[0331] 实例270图示其中没有同类的字节级标记且265的字节B1-B8中的每一个具有不同标记值的树结构。以按照在这里其他地方的讨论的方式(例如，对于分别图78-81的元素100、120、130和140)，如果子树以其根作为表示用于与第一节点相关联的字节范围的同类标记值的第一节点，则实施例可以从子树省略后继节点。例如，为进一步图示，参考图42。在实例280中，元素262可以表示与包括如上所述的字节B1-B8的标记的字265相关联的标记262a。标记262a可以是到树结构的指针或者地址，表示B1-B8 265的字节级标记。在该实例280中，字节B1-B8中的每一个具有相同标记T1，且因此树结构仅需要包括根节点281。因为字节B1-B8的字节级标记可以随时间修改或者改变，因此可以更新由标记262a指向的树结构或者其他结构以反映这种字节级标记修改。

[0332] 在提供相同数据字265内的字节级标记，或更一般地子字标记的实施例中，可以将附加输入提供给PUMP，指示正在参考哪一个或多个字节级标记(与字265中包括的哪一个或多个字节对应)。例如，对于其中在单个标记的数据字265中存在8个字节B1-B8的字节级标记，PUMP的附加输入可以是8位的位掩码，其中8位中的每一位与字节B1-B8中的不同的一个相关联，且表示是否使用用于字265的特定字节的字节级标记。作为变型，实施例可以通过指定字节范围，比如起始字节和长度或者大小(例如，通过指定起始字符B4和表示大小或者长度5的字节B4-B8)来表示一个或多个字节。在基于RISC-V架构的按照在这里其他地方的讨论的至少一个实施例中，可以定义CSR，其哪个记录或者存储附加输入表示要由PUMP使用用于一个或多个字节B1-B8的哪一个或多个字节级标记。附加输入例如可以是标识由PUMP使用的特定字节级标记的位掩码或者其他适当的表示。在至少一个实施例中，PUMP可以正常接收表示哪一个或多个字节要用作从数据路径(例如，从代码执行域)的输入的前述附加输入而不使用CSR。但是，在规则未命中时，前述附加输入可以记录在CSR中，以使得其中执行规则未命中处理器的元数据处理域可以获得前述附加输入(例如，用于前述附加输入的CSR值提供给关于规则插入的PUMP)。

[0333] 如在这里其他地方讨论的，可以以类似的方式以策略级处理许多指令。例如，加和减指令操作码或者操作码可以典型地相同地处理它们的元数据，由此通过考虑到PUMP的相

同标记输入和由PUMP传播的相同标记输出，两个操作码可以对于特定策略在规则级类似地表现。在这种情况下，加和减操作码可以在的单个操作组或者“操作组”中分组在一起，以使得同一组规则可以用于该特定操作组中的全部操作码。操作码怎样分组在一起是取决于策略的且可以随策略而变。在一个实施例中，可以使用转译或者映射表，其关于每个策略级别将特定的操作码映射到且关联的操作组。换句话说，可以对于每个策略(或者具有相同操作码到操作组映射的多个策略的指定组)创建不同映射表，因为映射可以每个策略改变。

[0334] 对于特定的操作码，转译或者映射表可以确定如上所述的操作组且也可以确定用于特定的操作码的附加信息。这种附加信息可以包括如在这里其他地方讨论的关心/不关心位向量，其可以指示哪些PUMP输入和PUMP输出(例如，输入标记和传播的输出标记)分别实际上用作用于规则处理的输入和作为用于特定的操作码的规则处理的相关输出传播。在一实施例中可以相对于任何PUMP输入和输出确定这些不关心位向量。在一个实施例中，该不关心位向量可以指示哪些输入标记和输出标记是相关的，且也可以指示哪些特定操作码位实际上用于特定操作码。这在以下相对于RISC-V架构和指令格式更详细地描述，但是也可以更一般地关于不同架构的其他适当的指令格式使用。包括用于特定操作码的操作组和关心/不关心位的前述转译或者映射表(例如，以下讨论的实例420的元素422)在这里其他地方也可以被称为操作组/关心表。

[0335] RISC-V具有每个使用用于操作码的不同指令集位的多个不同指令格式。参考图43的实例400，示出了在使用RISC-V架构的指令的实施例中可以在用于不同操作码的不同位编码中包括的指令的位。通常，RISC-V架构包括多个指令格式，其中指令的不同位可以用作操作码编码的一部分。通过32位指令，总共多达22位可以用于表示操作码的编码。元素404表示可以用于表示取决于指令格式的用于特定操作码的位编码的RISC-V架构中的指令的部分。元素404包括可以用于编码特定操作码的位404a-404c的3个字段。元素404a指示7位的第一操作码字段，操作码A。元素404b指示3位的第二操作码字段，funct3。元素404a指示12位的第三操作码字段，funct12。取决于指令(例如，比如系统调用的)，操作码编码可以包括多达由404a-c表示的全部22位。更具体地，在RISC-V中，可以仅使用404c的7位，仅使用404b和404c的10位(排除404a)或者使用404a-c的全部22位来编码操作码。作为进一步变型，RISC-V架构的指令可以具有使用如由402表示的字段的操作码编码。元素402包括如上所述的位404b和404c的两个字段。另外，不在操作码编码中使用funct12 404a的全部12位，指令可以仅使用由funct7 402a表示的12位中的7位。因此，作为又一可能性，操作码可以具有如由元素402所示的使用字段402、404b和404c的编码。

[0336] 在图44中图示了实例420，其图示可以用于根据在这里的技术的实施例的映射或者转译表。如上所述可以提供操作码421作为到操作码映射表422中的输入或者索引以查找或者确定操作码421的映射的输出424。映射的输出424可以包括用于特定操作码421的操作组和用于PUMP输入和输出的关心/不关心位向量。在基于RISC架构和指令格式的实施例中，操作码可以潜在地具有多达22位编码。但是，由于需要容纳22位操作码的大量项(例如，表可以包括指示其关联的操作组和关心/不关心位向量信息的用于每个操作码的项，导致用于22位操作码的几百万项)，使用这种大的22位操作码作为到表中的索引是不合理的。为在这种实施例中减小表422的大小，可以仅使用22位操作码字段的一部分来对表422进行索引。例如，在至少一个实施例中，操作码421输入可以是10位操作码，如在实例400中由元素

404b和404c表示的。因此，可以使用操作码的操作码位404b和404c来索引表422以确定操作码的操作组和关联的关心/不关心位向量。

[0337] 在这种实施例中，指令的funct 12 404a的剩余12个操作码位可以作为到PUMP的输入提供，其中404a的适当部分对于特定操作码掩蔽。关于funct12 404a的哪些特定的位应该/不应该对于特定操作码掩蔽的信息可以包括在从查找操作码的映射表422输出的关心/不关心位向量信息中。在基于RISC-V架构的至少一个实施例中，关心/不关心位向量信息相对于用于操作码的funct 12 404a的12个操作码位可以指示以下之一：

[0338] 1.因为不使用404a的位，所以可以掩蔽全部12位；

[0339] 2.使用如由402a表示的12位中的7位，其中掩蔽404a的最低5位(例如，位20-25)；或者

[0340] 3.使用404a的全部12位且因此没有404a的位的掩蔽。

[0341] 此外，在这种实施例中，funct12 404a的12个操作码位可以记录或者存储在CSR中，比如在这里其他地方描述的sfunct12 CSR，作为关于执行到PUMP中的规则插入的PUMP输入提供。在至少一个实施例中，PUMP可以正常从数据路径(例如，从代码执行域)接收前述操作码位而不使用CSR。但是，在规则未命中时，前述可以记录在CSR中，以使得其中执行规则未命中处理器的元数据处理域可以获得前述作为输入(例如，提供CSR值作为规则插入情况到PUMP的输入)。

[0342] 在根据在这里的技术的至少一个实施例中，多个用户处理可以使用其中物理页映射到用户处理地址空间的虚拟存储器环境执行。可以使用在这里的技术以允许在多个用户进程之间共享存储器的物理页，其中相同组的一个或多个物理页可以同时映射到多个用户进程的地址空间。在至少一个实施例中，由对于其可允许共享的这些进程使用的标记可以特性化为全局的，即跨各个用户进程地址空间具有相同值和含义或者解释。

[0343] 参考图45，示出了图示在根据在这里的技术的实施例中在进程之间的物理页共享的实例430。实例430包括具有地址空间434的进程P1和具有地址空间436的进程P2。元素434可以表示范围0到MAX的虚拟存储器处理地址空间，其中MAX表示由P1使用的最大虚拟存储器地址，且0表示由P1使用的最小虚拟地址。如现有技术中已知的，存储器432的物理页可以映射到比如434的虚拟地址空间中，其中可以使用这种映射的物理页的映射的虚拟地址由P1访问映射的物理页的内容。例如，物理页A 432a可以映射到Pi的虚拟地址空间的子范围X1中。进程P1例如可以通过参考子范围X1中的特定虚拟地址而从页面A 432a中的位置读取数据项或者指令。

[0344] 类似地，存储器432的物理页可以映射到虚拟地址空间436中，在虚拟地址空间436中，映射的物理页的内容可以使用这种映射的物理页的映射的虚拟地址由P2访问。例如，物理页A 432a可以映射到P2的虚拟地址空间的子范围X2中。进程P2例如可以通过参考子范围X2中的特定虚拟地址而从页面A 432a中的位置读取数据项或者指令。

[0345] 标记431可以表示关于页面A432的存储器位置的标记，其中这种标记可以由PUMP关于如在这里描述的规则处理使用。因为页面A 432经由如图所示的映射由P1和P2共享，相同组的标记431也由PUMP关于P1和P2两者的执行指令使用。在这种实施例中，标记431可以特性化为由P1和P2两者共享的全局标记。另外，在至少一个实施例中，由多个进程P1和P2共享的全局标记431以比如使用相同规则和策略的类似的方式解释。例如，具有值100的第一

标记可以与432a中的第一存储器位置相关联。第一标记可以表示值，该值表示关于确定对于特定执行指令是否可允许执行参考第一存储器位置或者其内容的操作的策略的规则使用的第一存储器位置的着色。第一标记可以由规则关于P1和P2两者的指令执行来解释为相同颜色。例如，100的标记值需要由规则关于P1和P2两者解释为相同颜色。此外，策略和规则的相同集合或者实例可以由PUMP用于P1和P2两者。

[0346] 在这种使用如上所述关于共享存储器的全局标记的实施例中，可以期望也允许进一步区分或者允许基于每个进程的不同访问、授权或操作。例如，假定页面A 432a包括由P1和P2两者共享的数据。但是，即使全局标记用于标记共享页A 432a，也可以期望允许基于每个进程的相对于432a的共享数据的不同操作或者访问。例如，进程P1可以具有到页面432a的写入访问且进程P2可以具有到页面432a的只读访问。但是432a可以是以全局标记标记的共享存储器页面。在这种具有关于共享页的全局标记的实施例中，可以关于P1和P2使用相同策略和规则集合，其中可以使用关于PC的不同标记值来区分每个进程的不同读和写访问性能。例如，进程P1可以包括执行到432a中的存储器位置的写入的第一指令，且当前PC标记具有值X。访问策略的规则可以执行以下逻辑：

[0347] 如果PCtag＝X，则允许写

[0348] 如果PCtag＝Y，则允许只读

[0349] 在这种情况下，PC标记具有值X，其由规则解释以允许进程P1的写入访问，且因此允许P1执行第一指令。进程P2可以执行也执行到432a中的存储器位置的写入的第二指令，且当前PC标记具有值Y。在这种情况下，PC标记具有值Y，其由规则解释以不允许进程P2的写入访问而是允许只读访问，且因此不允许P2执行第二指令。

[0350] 因此，在至少一个实施例中，PC标记可以用于编码可以每个进程不同的特权、访问或者授权，由此特定的允许的特权、访问或者授权可以由不同的PC标记值表示。

[0351] 实施例可以以任何适当的方式指定要用于每个进程的特定PC标记值。例如，特权代码可以作为最初指定要用于特定进程的PC标记值的操作系统启动或者初始化的一部分执行。作为变型，实施例可以执行映射操作作为将共享页A 432a映射到进程地址空间中的一部分。当执行映射时由操作系统应用的规则可以传播或者产生特定PC标记，作为表示基于特定进程的期望访问、特权或者授权的输出。

[0352] 以该方式，相同规则集合可以与具有全局标记的共享页一起使用，其中规则基于PC标记编码用于访问、授权或者特权中的差异的逻辑。应当注意，PC标记也可以是到存储器位置的指针，由此指针标记以如在这里关于其他标记描述的的方式指向包括用于不同策略的不同标记值的结构。以该方式，相同组的PC标记值可以用于表示可以随策略而变的进程的不同性能。例如，如上以P1所述的PC标记值X具有如上以策略共享区的存储器安全策略或者数据访问策略所述的第一用途。相同PC标记值X可以具有由第二不同策略的规则，比如控制流动完整性(CFI)给予的第二用途和含义。

[0353] 在这里描述的CFI策略的方面可以关于基于可允许调用、跳转、返回点等的静态定义限制控制转移来使用。但是，可以包括在CFI策略中的附加方面或者维度涉及动态或者运行时间调用信息的推行，由此进一步改善可以做出返回的控制转移的条件。为进一步图示，参考图46的实例500，其包括例程foo 502，bar 504和baz 506。例程Foo 502可以包括在地址X1调用例程bar的调用指令，导致控制到bar 504的运行时间转移501a。例程bar 504然后

包括将对例程bar的控制返回到501b地址X2的返回指令。因此,X2表示在X1调用到例程bar之后的例程foo中的指令的返回点地址或者位置。例程Foo 502可以包括在地址Y1调用例程baz 506的第二调用指令,导致控制到baz 506的运行时间转移501c。例程baz 506然后包括将对例程bar的控制返回到501d地址Y2的返回指令。因此,Y2表示在Y1调用到例程baz之后的例程foo中的指令的返回点地址或者位置。

[0354] 静态CFI策略例如可以基于反映动态运行时间控制流动方面的当前运行时间堆栈或者调用链,允许任何两个转移点之间的全部潜在控制流动,而不进一步限制控制流动或者转移。例如,如果foo 502可以调用bar 504,如在500中所示,在foo中在X1的bar的调用之后,存在从bar回到指令的地址X2的静态地允许的控制流动。但是,如果还未调用foo或者foo迄今为止仅发起了到应该在bar调用之前返回的某个东西的另一调用,应该不可能执行返回到X2的返回链路。作为如实例500中所示的运行时间执行的另一实例,对于通过501a到bar 504的调用,应该不可能通过501d在Y2返回到Foo 502。

[0355] 现在将描述的可以关于扩展CFI策略的规则以推行控制返回流动路径控制的动态CFI返回策略使用的技术。对于动态CFI返回策略保证返回到比如X2的特定返回位置仅当在特定调用或者起用,比如在X1到bar的调用之后做出时有效,动态CFI返回策略当做出调用时比如可以在一个或多个标记中存储信息以排除无效返回。如现有技术中已知的,当比如使用RISC-V指令集的JAL(跳转和链接)指令做出调用时,在返回地址寄存器RA中保存返回地址。RISC-V指令集也包括作为返回指令的实例的JALR(跳转和链接寄存器)指令。在一个方面中,来自JAL的RA寄存器中保存的返回地址可以特性化为返回到该点的“能力”。在至少一个实施例中,可以以使得规则将适当的标记能力推动到所得的返回地址上的标记来标记JAL指令。例如,通过作为返回地址寄存器的RA,规则可以在RA寄存器上放置标记,指示RA寄存器包括有效或者适当的返回地址,且在之后的点,RA寄存器的地址可以用作控制可以转移到的返回点。换句话说,关于RA寄存器的标记给出要用作载入PC中以执行控制的返回转移的返回地址的RA中的地址的许可。当以RA的地址加载PC时,RA标记也可以通过CFI策略的规则存储为PC标记。

[0356] 为进一步图示可以用于限制关于返回的控制流动的技术,实施例可以每个返回点(例如,X2、Y2)以动态CFI标记,比如期望-A来编码标记。此外,每个JAL指令(或者调用指令)编码标记使得对于JAL指令评估的规则以适当的动态CFI返回到A标记来标记RA寄存器中的返回地址(其中由JAL计算返回地址)。对于每个返回,比如使用以动态CFI返回到A标记来标记的RA寄存器的每个JALR指令,PUMP规则将标记(动态CFI返回到A标记)传播到PC上,如可以关于其他静态CFI策略规则执行的那样。CFI策略的规则可以实现检查用于返回指令的RA寄存器以的逻辑。如果用于返回的RA寄存器没有用动态CFI返回到A标记来标记,则知道RA寄存器不包括允许以JALR指令使用的有效返回地址。在返回点(例如,X2和Y2),规则可以实现逻辑,由此当遇到期望-A代码标记时(例如,作为在X2的关于指令的标记),检查该PC以动态CFI返回到A来标记,且从PC清除动态CFI返回到A标记。

[0357] 作为以上的结果,就防止了代码返回到任何返回地址。此外,如果返回地址复制到另一位置,比如另一寄存器,则该规则可以防止复制的值保留返回授权能力。这防止了代码制造可以用于执行用于同一调用的多个返回的寄存器中的返回地址的副本。作为以上的另一结果,如果堆栈上的有效返回地址(适当地标记的)以新地址(不适当地标记的)重写且然

后做出尝试以返回到新地址，则防止该返回。

[0358] 实施例也可以包括用以防止或者进一步限制多于一次地使用动态CFI返回到A标记的规则的能力。作为第一实现，实施例可以使用限制可以在哪里写入或者复制返回地址的规则(如存储在以动态CFI返回到A标记来标记的RA寄存器中)。例如，实施例可以使用仅允许适当地标记的RA寄存器的返回地址以适当地代码标记的函数(function)代码将返回地址写入到堆栈的规则。作为第二替代实现，实施例可以包括使用PC状态(例如，PC标记)和原子单元存储器操作以使得返回地址线性(例如，跟随或者在调用之后发生)的规则。例如，执行调用设置PC标记以表示有效返回地址。规则可以仅允许如果PC标记设置为有效返回地址的返回。可以使用当写入返回地址到存储器时将PC标记设置为无返回地址的附加规则。可以使用当复制返回地址到目标寄存器时，可以设置PC标记为无返回地址，且目标寄存器不被标记为有效返回地址的规则。可以使用当使用来自RA寄存器的返回地址执行算术操作时，结果不被标记为有效返回地址的规则。可以使用仅允许以具有非返回地址(例如，其中PC标记设置为有效返回地址)的原子交换操作恢复来自存储器的返回地址的规则。

[0359] 实施例可以进一步定义规则以提供堆栈保护策略。在一个方面中，堆栈保护策略可以被部分地示为比如存储器安全的一个或多个其他策略的扩展，在存储器安全中，规则可以使用用于策略推行的指令和数据两者的标记。应该注意在后续讨论和在这里的其他地方，比如例程和过程的术语可以可互换地使用且更一般地参考代码的可调用单元，其当起用时导致在调用堆栈上创建新堆栈帧。也可以用于代码的可调用单元的其他名称可以包括函数、子例程、子程序、方法等。

[0360] 参考图47，示出了实例520，其图示在根据在这里的技术的实施例中的用于运行时间起用的帧的调用堆栈。在520中，假定例程foo 502执行对G1的调用，该对G1的调用又调用G2。因此，在执行中的点，执行例程foo且例程foo做出对例程G1的第一调用且G1做出对例程G2的调用。元素522可以表示用于例程foo的第一调用堆栈帧。元素524可以表示用于例程G1的第二调用堆栈帧。元素526可以表示用于例程G2的第三调用堆栈帧。

[0361] 用于运行时间调用实例或者起用的堆栈帧(比如522、524、526)中存储的信息例如可以包括返回地址，由用于寄存器的调用实例使用的数据，变量或者数据项等。元素522a和524a可以表示分别包括在foo的帧522和G1的帧524中的返回地址。比如可以由恶意代码执行的一个普通攻击可以修改堆栈520上存储的比如522a和524a的返回地址。使用比如在这里其他地方描述的用于动态CFI返回策略的技术(例如，关于实例500中的图46描述的)可以防止不适当的或者无效的返回，比如使用来自已经不适当地修改的堆栈位置的返回地址。但是，可以进一步期望也推行提供堆栈保护和防止堆栈存储位置，比如返回地址的不适当修改的附加规则。因此，这种用于堆栈帧保护策略的附加规则可以防止522a或者524a的修改，而不是允许522a的不适当的修改且然后停止使用不适当地修改的返回地址的返回。

[0362] 如以下更详细地描述，可以提供不同级别的堆栈保护。在一个方面中，堆栈保护可以基于静态过程确定(也称为在这里其他地方描述的静态授权保护模型)或者可以基于过程以及特定过程的起用实例两者确定(也称为在这里其他地方描述的实例授权保护模型)。通过静态授权保护模型，堆栈保护策略的规则可以基于创建帧的特定过程或者例程提供堆栈保护。例如，代替仅包括如在520中用于foo的单个实例的单个帧的堆栈，可能存在在一个时间点包括在当前调用链中的foo的多个起用实例，且因此有用于例程foo的堆栈中的多个

调用堆栈帧(例如,比如基于对foo的循环调用)。基于静态例程或者过程,foo的任何实例也能够修改或者访问用于foo的实例的任何调用堆栈帧中的信息。例如,foo实例1可以具有调用堆栈帧1,且foo实例2可以具有调用堆栈帧2。基于用于堆栈保护的静态程序或者过程,foo实例1的代码能够访问堆栈帧1和2,且foo实例2的代码也能够访问堆栈帧1和2。在这种实施例中,用于同一过程或者例程foo的全部实例的调用堆栈帧可以以相同标记着色。例如,foo实例1的帧1和foo实例2的帧2两者可以以标记T1着色,以使得存储器安全策略的规则将允许跨同一例程或者过程的不同实例的以上提到的堆栈帧访问。

[0363] 作为堆栈保护的进一步精细粒度,实施例可以使用基于静态例程或者过程以及例程或者过程的特定运行时间实例(例如,实例授权保护模型)来进一步限制堆栈的访问的堆栈保护策略的规则。例如,如上所述,foo实例1可以具有调用堆栈帧1,且foo实例2可以具有调用堆栈帧2。基于静态例程或者过程以及用于堆栈保护的起用实例,foo实例1的代码能够访问堆栈帧1而不是堆栈帧2,且foo实例2的代码能够访问堆栈帧2而不是堆栈帧1。在这种实施例中,过程或者例程的每个起用实例的调用堆栈帧可以以不同标记着色。例如,foo实例1的帧1可以以标记T1着色,且foo实例2的帧2可以以标记T2着色,以使得存储器安全策略的规则将基于每个特定启用和例程或者过程来允许以上提到的堆栈帧访问。

[0364] 实施例可以进一步提供用于单个过程调用实例的堆栈的不同区域或者部分的精细粒度级别,比如通过每个以不同颜色着色堆栈帧中的不同对象或者数据项(也称为在这里其他地方描述的对象保护模型)。如在这里其他地方描述的,堆栈帧可以包括在例程或者过程的特定起用中使用的数据项或者对象的存储,其中每个这种数据项或者对象可以以不同颜色标记。例如,参考图48,示出了图示具有由例程或者过程foo分配的存储的数据项540和堆栈帧531中的关联的标记的存储器的实例530。元素540表示具有在例程foo中分配的存储的变量540a-540c,且元素531表示用于调用堆栈中的例程foo的该特定起用实例的调用堆栈帧。元素531包括用于变量阵列540a的存储器区域532,用于变量线(line)540b的存储器区域534和用于变量密码540c的存储器区域536。另外,帧531包括用于存储的返回地址的存储器区域538。不同区域532、534、536和538中的每一个可以以如由533表示的不同标记来标记或者着色。区域532中的每个字可以以Red1标记。区域534中的每个字可以以Red2标记。区域536中的每个字可以以Red3标记。区域538中的每个字可以以Red4标记。

[0365] 作为进一步的变型,实施例可以定义用于代码集合(例如,例程、过程等)的不同信任区域或者边界,并提供不同保护级别。例如,不是所有调用的例程可以具有相同级别的信任。例如,开发者可以具有他/她已经写入的第一例程集合,且具有由第一代码集合执行的操作不包括任何恶意代码的高信任级别。但是,第一例程集合可能做出调用到由第三方或者从因特网获得的库中。该库可能是不可信的。因此,实施例可以基于不同代码主体和由每个使用的特定数据项来改变保护级别。例如,参考图49的实例550,假定可信用户代码中的例程foo调用库中的例程有害程序(routine evil),且作为参数将到区域534的指针(到数据项线540b的指针)传递到有害程序。在这种情况下,代替以不同颜色着色或者标记531的每个区域,区域532、536和538可以全部以相同颜色,比如Red5着色,且区域534可以以不同颜色,比如Red6标记。这可以用于进一步保证由例程有害程序访问的存储器区域534以与531的其他区域不同的颜色标记,来作为存储器安全级别,因为有害程序被认为是不可信代码。另外,传递到有害程序的到区域534的指针可以以与区域534相同的颜色Red6着色或者

标记。以该方式，存储器安全策略规则可以把由有害程序使用的对存储器的访问限制于那些以Red6标记的。

[0366] 特定例程、库或者主体或者代码是否具有特定信任级别可以基于使用一个或多个标准和输入的分析确定。例如，基于运行时间分析和库的代码的使用，可以确定信任级别。例如，如果库做出到又一未知或者不可信外部或者第三方库的调用，则信任级别可能相对低。代码主体的信任级别可以在从其获得代码的源或者位置使用。例如，来自从因特网获得的库的代码的使用可以考虑为不可信。相反地，由不起用任何不可信代码的特定开发者开发的代码可以具有高信任级别。

[0367] 以下更加详细地描述堆栈帧和堆栈保护的前述及其他方面。

[0368] 关于堆栈帧且再次参考实例530，编译器可以通过提供添加整数(帧的大小)到现有堆栈指针来创建新堆栈指针。旧的堆栈指针可以推到堆栈上(到帧中)且然后通过从堆栈将其读回来恢复。到堆栈指针的添加可以表示包括比如上面在用于数据项540a-c的531中描述的许多独立对象的帧的总大小。堆栈需要用于这3个数据项540a-c的空间，且编译器能够确定数据项540a-c需要的总空间。在标准使用中，编译器通过计算它们的偏离堆栈指针(或者从堆栈指针创建的帧指针)的地址来访问分别用于这些数据项540a-c的存储532、534和536。因此，在实施例中的编译器、运行时间和调用惯例可以通过进行单指针运算来创建和使用到堆栈调用帧的不同区域的指针。

[0369] 静态授权保护模型指示在属于静态代码块的对象上的授权，这些对象比如创建帧的例程或者过程。因此，如在这里其他地方讨论的，创建帧的过程foo具有创建到在该帧中的东西的指针的授权。在最简单的情况下，相同授权将允许foo访问它创建的任意帧，即使这些帧在堆栈上较早或者较晚。静态授权指的是可以在加载时间预先分配标记(例如，用于存储器单元的颜色，着色的指针，创建着色的点的代码标记(例如，也称为指令标记或者关于指令的标记))。实例授权保护基于堆栈上函数起用的深度来提供授权。对象保护指示在堆栈上分配的对象的级别，而不只是堆栈帧的保护。因此，对象保护允许检测和防止从帧内的一个对象(例如，阵列，缓存)到同一帧上的另一对象的溢出，这是使用具有静态授权保护模型或者实例保护模型的简单堆栈帧粒度PUMP规则不能实现的。对象保护可以应用于静态授权保护模型和实例保护模型两者。作为对象保护的变型，实施例也可以采用用于分层对象的分层对象保护，比如包括多个不同数据项子对象，比如整数和阵列的结构。在具有其中第一对象包括一个或多个子对象中的每一个的一个或多个级别的分层对象的至少一个实施例中，可以对于第一对象生成第一标记且然后可以基于第一标记生成附加子对象标记。每个子对象标记可以用于标记不同子对象。子对象标记可以是表示分级中的子对象的特定位置的值。例如，可以生成标记T1用于以包括如子对象2和3的2个阵列的结构使用。用于2个阵列中的每个的不同子对象标记可以从T1生成且用于标记2个阵列子对象。

[0370] 现在将描述的是在根据在这里的技术的实施例中可以关于不同堆栈操作的堆栈存储器执行的处理。在启动时，堆栈存储器可以使用空堆栈帧标记来标明或者标记全部存储器单元。按照在这里的其他讨论和技术，可以通过起用PUMP规则来执行这种标记。应当注意，堆栈存储器单元对空堆栈帧标记的初始标记可以不对于整个堆栈一次性执行，而是可以在扩展堆栈的内核程序页故障处理器中递增地执行。

[0371] 关于比如由编译器分配新堆栈帧，可以对于新分配的帧创建新帧标记。到新帧的

指针可以以新帧标记来标记。例如，实施例可以标记创建新帧指针的指令(例如，比如执行指针运算的ADD(加)指令(通过加到堆栈指针))，其中关于指令的标记触发策略规则以创建新帧标记。使用规则和标记传播，可以对于堆栈指针创建特殊标记，且特殊标记可以用于标记该堆栈指针。随后，对于每个帧指针，可以从堆栈指针特殊标记导出唯一的帧指针标记，且可以以其唯一的帧指针标记来标记帧指针。在这种实施例中，帧指针标记可以从堆栈指针的标记的副本(例如，加或者0)创建。

[0372] 当比如对于例程或者过程的新起用分配新堆栈帧时，新分配的堆栈帧的存储器单元例如可以使用称为严格对象初始化的第一技术或者称为懒惰对象着色的第二技术来标记或者着色。

[0373] 通过严格对象初始化的第一技术，新分配的帧的空堆栈帧单元全部初始着色或者标记为想要的一个或多个颜色，比如基于帧的静态对象。这种初始着色可以作为在随后使用帧例如存储用于关联的起用的信息之前新分配的帧的初始化处理的一部分执行。实施例可以添加代码，该代码触发规则执行空堆栈帧单元到想要的一个或多个颜色的着色或者标记，比如基于帧的静态对象。关于指令的代码标记可以用于授权和定义关联的存储器单元着色。可以基于帧存储器单元颜色，比如根据存储器安全策略规则，来允许或不允许帧的着色的存储器单元的后续存储或者读取(例如，对于以颜色CI标记的存储器单元，规则允许存储器操作使用具有也是相同颜色CI的指针访问着色的存储器单元内容，但是如果指针是不同颜色C2，则可以不允许存储器操作)。另外，关于指令的代码标记可以提供执行过程内的存储器操作的授权。

[0374] 通过懒惰对象着色的第二技术，没有如严格对象初始化技术那样的全部堆栈对象的初始着色。而是通过懒惰对象着色，到标记为空堆栈帧的堆栈存储器位置的存储导致触发允许存储且还基于写入者来改变存储器位置的颜色的规则。到标记为空堆栈帧的堆栈存储器位置的读取是未初始化的存储器读取，且可以取决于策略允许/不允许未初始化的存储器读取而允许/不允许。通过懒惰对象着色，不执行在创建时起用规则完全地初始标记帧的全部存储器单元的初始代码块。而是，通过关于存储操作起用的规则来标记存储器单元。

[0375] 在至少一个实施例中，使用严格对象初始化或者懒惰对象着色可以取决于期望的保护级别和不能维持的脆弱性的发生。

[0376] 直接访问来自堆栈/帧指针的数据的例程或者过程内的代码是标记为允许其这样做的代码。关于懒惰对象着色，到存储器单元的存储导致如上所述的基于写入者着色存储器单元。例如，通过回头参考实例530，具有帧531的例程foo的存储指令可以将值写入到阵列532中的存储器位置。根据事实上当前的堆栈保护策略，为了存储指令写入到用于foo的调用帧的阵列532中的位置，可能需要存储指令具有标记Red1。可以触发策略的第一规则以执行该用于存储指令的校验。因此，实施例可以使得编译器生成触发第一规则以Red1标记存储指令的代码序列。(例如，作为前述的变型，关于存储器单元的标记，比如Red1，可以与关于指令存储或者其他指令的标记相关但是不与其相同。例如，“Red1code”CI标记可以指示具有该标记的指令可以访问Red1标记的存储器单元且可以创建Red1标记的存储器单元)。当存储指令是当前指令时，可以触发前述第一规则，校验指令标记以保证其是Red1。作为输出，规则可以以Red1标记来标记阵列532中的存储器位置。

[0377] 创建到特定对象的指针的过程内的代码被标记为污点或者设置用于该对象的指

针。该指针可以用于后续指令中过程自己的使用和/或可以作为变元(argument)传递到另一过程。

[0378] 将寄存器值存储到帧或者从帧恢复寄存器值可以基于帧授权。存储这些寄存器值的堆栈帧的一个或多个存储器位置可以当做堆栈帧中的唯一对象。指令标记提供用于这种标记的存储和加载指令的授权。通过懒惰对象着色,以存储数据到存储器单元的授权标记的存储指令也提供基于写入者来标记存储器单元的授权(例如,包括存储指令的过程)。

[0379] 在堆栈上传递的过程变元可以以允许调用者和被调用者两者访问的标记标明。注意到,可以特殊地标记返回地址(例如,比如关于图46的在这里其他地方描述的动态CFI返回策略)。因此,如果返回地址存储在堆栈上(例如,比如关于嵌套或者递归调用),由于关于返回地址的标记将不允许存储重写堆栈上的返回地址。当堆栈导出的指针关于到另一过程的调用传递到另一帧时,使用该指针执行的存储器访问导致触发存储器安全策略的规则,如在这里其他地方描述的。可以基于特定存储器位置的标记来标记创建到存储器位置的指针的指令。指令标记可以指示访问存储器位置的授权。指令可以触发标记指针以表示访问存储器位置的授权的规则。例如,规则可以向指针分配与该指令相同的标记或者基于该指令标记的变型。因此,在一个方面中,创建指针的指令也创建通过指针访问存储器位置的能力和通过作为变元传递到被调用过程的指针来共享该能力。应当注意,通过懒惰对象着色,指针将需要具有一个标记,其提供对空堆栈帧单元进行标记的授权,这对于堆存储器安全指针可能是不允许的单元。

[0380] 关于导致从堆栈除去帧的返回或者其他操作(例如,比如由于被调用例程的完成),标记的代码可以清除该帧。关于这种代码的标记提供授权,以把与该帧相关联的任何帧对象标记改变为空堆栈帧单元标记。

[0381] 在根据在这里的技术的计算机系统的实施例中执行的程序代码可以包括执行例外处理的代码。如现有技术中已知的,例外处理是响应于表示需要由例外处理器执行的特殊处理的异常或者例外状况的发生的例外而执行的处理。因此,当在程序中的第一点发生例外时,可以中断正常程序执行流以使得控制转移到例外处理器。在控制转移到该处理器之前,执行的当前状态可以保存在预定位置。如果在已经由该处理器处理例外之后可以恢复程序执行,则可以恢复程序的执行(例如,控制然后可以转移回到程序中的第一点之后)。例如,除以零操作可能导致可持续的例外,其中程序执行可以在由该处理器处理例外之后恢复。关于实现例外处理器,实施例可以使用比如setjump和longjump的库例程。例如,setjump和longjump可以分别是标准C库例程,setjmp和longjmp定义如下:

[0382] int setjmp(jmp_buf env)

[0383] 其中setjmp建立局部jmp_buf缓存且将其初始化用于跳转。setjmp在由env变元指定的环境缓存中保存程序的调用环境以用于之后由longjmp使用。如果返回来自于直接起用,则setjmp返回0。如果返回来自于到longjmp的调用,则setjmp返回非零值。

[0384] void longjmp(jmp_buf env,int value)

[0385] 其中longjmp恢复通过程序的相同起用中的setjmp例程的起用保存的环境缓存env的上下文。从嵌套的信号处理器起用longjmp是未定义的。由value指定的值从longjmp传递到setjmp。在完成longjmp之后,程序执行继续,就好像刚刚返回setjmp的相应起用那样。如果传递到longjmp的值是0,setjmp将好像其已经返回1那样表现;否则,它将好像其已

经返回value那样表现。

[0386] 因此，setjmp可以用于保存程序的当前状态。程序的状态例如取决于存储器的内容(即代码、全局、堆和堆栈)和其寄存器的内容。寄存器的内容包括堆栈指针、帧指针和程序计数器。setjmp保存程序的当前状态，以使得longmp可以恢复程序状态且因此将程序执行的状态返回到当调用setjmp时程序执行的原来状态。换句话说，longjmp()不返回。而是，当起用longjmp时，执行返回或者恢复到由先前保存的程序状态表示的特定点(如由setjmp保存的)。因此，longjmp()可以用于将控制从信号处理器转移回到程序中的保存的执行点而不使用标准调用或者返回惯例。

[0387] 例如，参考图50。在实例560中，主例程562可以调用第一例程563，且第一563例程可以调用第二例程564。如图所示，主例程562可以包括在调用第一例程之前在点X1对setjmp的调用。在点X1调用第一时间setjmp，它返回零且然后调用第一例程。在执行longjmp之后，setjmp返回1。第二例程564包括在点X2对longjmp的调用，则导致控制转移回到在其中调用setjmp的位置X1的主例程。setjmp现在被再次调用且返回1，所以不调用第一例程且控制进行到NEXT(下一个)。

[0388] 关于堆栈保护策略，可以期望在恢复到先前由settmp保存的点X1的执行之前清除堆栈。例如，基于以上调用链主-第一-第二，3个堆栈帧可能存在于调用堆栈中，且可以执行处理以清除与longjmp调用和setjmp调用之间的调用链中的起用相关联的堆栈存储器。具体来说，longjmp的代码可以包括对于在该实例中的第一例程563和第二例程564清除堆栈帧的代码。现在将描述的是可以关于根据堆栈保护策略执行这种堆栈清除使用的技术。

[0389] 关于当执行将程序状态保存到堆栈存储器的setjmp时的堆栈保护策略，实施例可以以区分的标记部分来标记当前堆栈指针存储器单元，以使得关于后续longjmp，规则可以校验堆栈自从setjmp以来没有改变。数据可以保存到表示当前程序状态的setjmp数据结构，即jmpbuf。保存的数据可以包括堆栈指针、程序计数器、第一指针(标记为是允许指向以区分的标记部分来标记的存储器位置的指针(例如，指向当前堆栈指针存储器单元))，和第二指针(标记为提供执行longjmp处理的授权的longjmp-清除-授权-指针)。在至少一个实施例中，longjmp-清除-授权-指针可以仅提供清除与可以从该过程递归地调用的过程集合中的帧相关联的标记的授权。

[0390] 关于当执行longjmp时的堆栈保护策略，代码可以校验当前堆栈指针表示比设置的跳转结构(例如，setjmp数据结构，jmpbuf)的保存的堆栈指针更深的堆栈位置。可以触发规则检查包括保存的堆栈指针(如由setjmp保存的)设置的跳转结构的存储器单元具有与(设置的跳转结构的)标记的第一指针兼容的标记。可以执行代码清除当前堆栈指针和保存的堆栈指针(如先前通过在设置跳转结构中设置跳转而保存的)之间的全部堆栈存储器位置。这种代码可以使用提供堆栈清除授权的以上提到的标记为longjmp-清除-授权-指针的第二指针(例如，用于指向清除的堆栈位置的第二指针)执行清除。可以由执行清除的代码触发规则，其中规则校验第二指针被标记为longjmp-清除-授权-指针。唯一地标记longjmp中的指令，以使得起用的规则允许唯一地标记的指令使用标记为longjmp-清除-授权-指针的指针。不在longjmp中的其他代码不能使用标记为longjmp-清除-授权-指针的指针(例如，另一代码不标记为允许longjmp-清除-授权-指针的使用)。

[0391] 在至少一个实施例中，可以通过使得编译器生成起用规则以执行期望指令标记

和/或存储器位置标记的指令序列来执行指令的标记。例如,对于堆栈存储器位置标记,编译器可以生成具有触发规则初始化或者复位堆栈位置的标记的存储指令的指令序列。对于标记指令,编译器可以生成具有触发规则以标记指令的存储指令的指令序列,其中用于指令的标记可以基于与由指令访问的标记的存储器位置相关联的颜色。关于从具有关联的堆栈帧的调用返回,可以添加代码从堆栈清除帧。当采用严格对象初始化且响应于调用创建新帧时,可以添加代码适当地标记或者着色新帧的对象。

[0392] 现在将参考图51到图53描述的是可以例如由恶意代码对堆栈做出的不同的未授权或者不想要的修改(“堆栈攻击”指的是通过堆栈修改做出的攻击)或者由非恶意代码做出的不想要的堆栈修改(例如意外重写或者缓存溢出)的实例。

[0393] 图51到图52图示可以采用以防止关于由比如第三方代码的代码模块(例如,起用的库例程)做出的堆栈修改的堆栈攻击且可以特性化为任意攻击者模型的动作。因此,例如,可以作为包括执行未授权的或者不想要的堆栈修改的代码的调用的第三方库例程的结果发生570和575中的情况。另外,堆栈修改也可以由进一步由调用的库例程的代码起用的又一例程做出。570和575的每行包括3列信息。对于每行572a-h,列1标识防止表示不期望的运行时间执行行为的项目,列2标识可以采用以避免列1的不期望的行为的防止动作,且列3标识可以用于实现或者推行列2的防止动作的一个或多个机制。通常,在列3中,列出替代机制,且每个可以取决于特定系统独立地和分开地实现。例如,现有的系统可以使用分开的进程作为第一机制,同时第二系统可以替代地使用能力且第二系统可以替代地使用特定的堆栈位置的着色或者标记。

[0394] 为进一步图示和按照在这里其他地方的讨论,比如当做出调用时执行的编程逻辑(prolog)代码的代码将返回地址和寄存器写入到堆栈。编程逻辑代码可以起用规则,这些规则以特殊标记来标记堆栈位置以限制什么代码可以修改或者一般地访问堆栈位置。例如,编程逻辑代码可以执行存储器写入/存储以在堆栈帧的存储器单元中存储返回地址、寄存器等。这种编程逻辑代码的写入/存储指令可以起用以特殊标记STACK FRAME TAG(堆栈帧标记)来标记堆栈帧中的存储器单元以将这些存储器位置标为特殊的和限制什么代码可以修改存储器单元的规则。编程逻辑代码的写入/存储指令也可以以PROLOG STACK TAG(编程逻辑堆栈标记)来标记,以限制可以执行该标记的指令。以下是由编程逻辑代码的写入/存储指令起用的规则推行的逻辑的实例,其以特殊标记STACK FRAME TAG来标记堆栈帧的存储器单元以将存储器位置标为特殊的和限制什么代码可以修改存储器单元。

[0395] 如果(CI＝PROLOG STACK TAG)AND(这是存储器写入操作),则输出或者Rtag＝STACK FRAME TAG

[0396] 在前述规则逻辑中,输出标记指的是放置在堆栈位置上的标记。

[0397] 以类似的方式,可以允许比如通过执行返回起用的收尾代码清除堆栈或者其一部分。收尾代码可以以EPILOG STACK TAG(收尾堆栈标记)(例如CI标记)的特殊标记来标记,且可以通过以特殊标记STACK FRAME TAG标记的指针的访问给予授权。收尾代码可以使用以STACK FRAME TAG(堆栈帧标记)特殊标记的指针,使用写入/存储操作执行前述堆栈清除。为进一步限制执行堆栈清除,可以如上所述标记收尾代码。在这种实施例中,写入/存储指令可以起用实现以下逻辑以推行策略的规则,在该策略中,可以仅使用特殊标记的指针(以STACK FRAME TAG标记的)由收尾代码执行堆栈清除,该规则为:

[0398] 如果(CI＝EPILOG STACK TAG)AND(存储器写操作)AND

[0399] (Mtag＝STACK FRAME TAG),则输出或者Rtag＝默认标记

[0400] 意在从堆栈恢复返回地址和寄存器的代码可以给予读取堆栈的这些特殊标记的存储器单元的授权。这种授权例如可以通过以下任意给出:标记代码(CI标记)以表示允许代码访问堆栈的特别标记的存储器单元,标记PC以指示代码具有授权,或者标记由代码使用的指针,其中指针指向特殊标记的存储器单元且关于指针的标记表示访问授权。例如,读取/加载指令可以给予读取以STACK FRAME TAG标记的堆栈存储器单元的授权。在一个实施例中,读取/加载指令可以通过使用特殊标记的指针(以STACK FRAME TAG标记的)从堆栈存储器位置读取来仅允许给予读取/加载指令授权。使用特殊标记的指针(以STACK FRAME POINTER标记的)仅允许读取/加载指令从特殊标记的堆栈存储器位置(以STACK FRAME TAG标记的)读取的规则逻辑可以是:

[0401] 如果(存储器读取操作)AND(R2tag＝STACK FRAME POINTER)AND(Mtag＝STACK FRAME TAG)则Rtag＝DEFAULT TAG

[0402] 作为前述的变型,可以通过以特殊标记STACK FRAME TAG标记由读取/加载指令使用的指针来给予读取/加载指令授权。

[0403] 仅使用特殊标记的指令(以STACK FRAME INSTRUCTION标记的)允许读取/加载指令从堆栈存储器位置读取的规则逻辑可以是:

[0404] 如果(存储器读取操作)AND(CItag＝STACK FRAME INSTRUCTION)

[0405] AND(Mtag＝STACK FRAME TAG),则Rtag＝DEFAULT TAG

[0406] 以下与在其他地方更详细地描述机制的实例。

[0407] 元素572a标识从不返回到调用例程(调用者)的被调用例程(被调用者)的不期望的运行时间行为。为防止该行为,采取的动作可以是具有与每个调用相关联的超时,其中可以允许最大时间量以完成起用的例程。在经过最大时间量之后,终止起用的例程的运行时间执行。实现超时的机制可以包括使得第三方代码的起用的例程由推行超时的单独线程做出,或者使用时间或者指令限制调用来直接限制被调用例程的时间量。

[0408] 元素572b标识资源耗尽的不期望的运行时间行为,其中被调用例程可以耗尽比如存储器的可用资源。为防止该行为,采取的动作可以是限制使得对被调用例程可用的资源。实现超时的机制可以包括使得第三方代码的起用的例程由推行最大资源限制的单独线程做出,或者使用特殊指令限制调用来直接限制被调用例程的资源量。

[0409] 元素572c标识比如通过做出到又一例程的期望调用而实行不期望的授权的起用例程的不期望的运行时间行为。为防止该行为,采取的动作可以是限制被调用例程的授权到最小化可允许的特权。实现此的机制可以包括以授权和控制被调用者或者被调用例程的能力来标记PC,和限制对被调用例程可访问的文件系统或者其他资源的部分,和限制起用的例程可以做出的可允许系统调用。

[0410] 元素572d标识被调用例程读取由随后由被调用例程调用的其他例程在寄存器中留下的项目的不期望的运行时间行为(例如,mycode调用库中的P1,且P1进一步调用例程有害程序且P1可以读取由有害程序在寄存器中留下的数据)。为防止该行为,可以采取的动作是清除非输入和非返回寄存器。实现此的机制可以包括执行明确的寄存器清除,着色包括非返回和非输入寄存器的堆栈的部分,以使得它们不能由被调用例程读取,且使得单独进

程来起用被调用例程。

[0411] 元素572e标识被调用例程读取由随后由被调用例程调用的其他例程在堆栈上留下的项目的不期望的运行时间行为(例如,mycode调用库中的P1,且P1进一步调用例程有害程序且P1可以读取由有害程序在堆栈上留下的数据)。为防止该行为,可以采取的动作是使得被调用的堆栈不可访问(例如,由比如有害程序的进一步起用的其他例程使用的堆栈区域对比如P1的首次调用的例程不可访问)。实现此的机制可以包括使用单独堆栈(例如,用于首次调用的例程P1和进一步起用的例程有害程序),能力(例如,标记PC或者使用允许访问特定堆栈区域特殊标记的指针,以限制允许读取堆栈区域的代码的能力或者授权),着色(例如,标记堆栈的数据区域以限制什么代码可以访问),和使得单独处理起用被调用例程。

[0412] 元素572f标识在堆栈前缀的项目之上所写的被调用例程的不期望的运行时间行为(例如,重写标识返回地址的返回地址区域)。堆栈前缀可以是包括返回到某些在前调用者所需的信息的堆栈的区域。为防止该行为,采取的动作使得堆栈前缀对被调用例程不可访问或者不可写。实现此的机制可以包括使得被调用例程和启起用被调用例程的用户代码使用单独堆栈,使用能力(例如,通过特殊标记的代码或者通过PC标记或者特殊标记的指针提供授权的代码来允许访问),使用着色(例如,以特殊标记来标记堆栈前缀的数据项,以使得不允许访问被调用例程),且使得单独进程来起用被调用例程。

[0413] 元素572g标识读取堆栈前缀中的数据的被调用例程的不期望的运行时间行为。为防止该行为,采取的动作是使用类似于以572f描述的机制的机制使得堆栈前缀对被调用例程不可访问。

[0414] 元素572h标识被调用例程通过重写到返回地址的指针而重定向堆栈前缀中的控制流的不期望的运行时间行为,其中指针存储在堆栈前缀中。为防止该行为,可以采取的动作是保护堆栈前缀中存储的返回地址。在一个方面中,元素572h标识572h的特定实例,且因此572h的机制类似于572f的机制。实现此的机制可以包括使得被调用例程和起用被调用例程的用户代码使用单独堆栈,使用能力(例如,通过特殊标记的代码或者通过PC标记或者由访问授权标记的特殊标记的返回指针提供授权的代码来允许访问),使用着色(例如,以特殊标记来标记包括返回地址的堆栈前缀的存储器位置,以使得不允许访问被调用例程),且使得单独进程来起用被调用例程。

[0415] 图53图示可以采用以防止关于任意输入攻击者模型的堆栈攻击的动作。

[0416] 元素581a标识在执行例程的当前帧中的不想要的项目之上执行代码写入的不期望的运行时间行为。为防止该行为,采取的动作可以是维持对象完整性。实现此的机制可以包括使用按对象的能力(例如,通过提供有特殊标记的代码或者通过PC标记或者由访问授权标记的特殊标记的返回指针提供授权的代码来允许访问),或者按对象的着色(例如,标记对象的存储器位置)。

[0417] 元素581b标识读取执行例程的当前帧中的项目的不期望的运行时间行为。为防止该行为,采取的动作可以是维持对象完整性。实现此的机制可以包括使用按对象的能力(例如,通过提供有特殊标记的代码或者通过PC标记或者由访问授权标记的特殊标记的返回指针提供授权的代码来允许访问),或者按对象的着色(例如,以对象特定标记来标记对象的存储器位置)。

[0418] 元素581c标识在(例如,起用执行代码的其他例程的)先前帧中的不想要的项目之

上写入的执行代码(具有当前帧)的不期望的运行时间行为。为防止该行为,采取的动作可以是隔离或者分隔堆栈帧。实现此的机制可以包括使用按帧的能力(例如,通过提供有特殊标记的代码或者通过PC标记或者由访问授权标记的特殊标记的返回指针提供授权的代码来允许访问),或者按帧的着色(例如,以帧特定标记来标记帧的存储器位置)。

[0419] 元素581d标识读取在(例如,起用执行代码的其他例程的)先前帧中的项目的执行代码(具有当前帧)的不期望的运行时间行为。为防止该行为,采取的动作可以是隔离或者分隔堆栈帧。实现此的机制可以包括使用按帧的能力或者按帧的着色,如以元素581c描述的。

[0420] 元素581e标识读取由当前执行代码起用的另一例程留下的堆栈的项目的执行代码(具有当前帧)的不期望的运行时间行为。防止动作是使得起用的例程的被调用堆栈对当前执行代码不可访问。实现此的机制可以包括使用单独进程、单独堆栈、和以类似于如关于572g描述的方式的能力和着色。

[0421] 元素581f标识修改返回指针(例如,包括起用执行代码的例程中的返回地址的堆栈中的位置)的执行代码(具有当前帧)的不期望的运行时间行为。防止动作是保护返回指针或者包括返回地址的堆栈中的位置。实现此的机制可以包括以类似于如关于572g描述的方式使用能力和着色。

[0422] 根据在这里的技术的实施例可以使用PUMP规则元数据处理系统作为另一混合系统的一部分以学习和验证新的规则集合。例如,PUMP规则元数据处理系统可以用于学习(例如,通过记录(logging))允许的控制流,且因此确定用于执行程序的规则和允许的有效的控制转移。规则和允许的有效的控制转移然后可以用作对于执行的程序推行的CFI策略的规则和有效的控制转移的集合。

[0423] 为进一步图示用于程序的CFI策略的学习规则和控制转移,可以执行第一训练或者学习阶段。在该第一阶段中,以标记的所有控制点(例如,分支或者转移源和目标)和CFI策略的训练版本来执行程序,其中没有用于控制转移指令的规则。因此,每次存在控制转移,比如分支或者跳转指令,存在PUMP规则高速缓存未命中,导致到PUMP规则元数据系统的高速缓存未命中处理器的控制转移。高速缓存未命中处理器可以执行处理以记录关于控制转移的信息。记录的信息例如可以包括转移的源位置和转移的目标位置。其他信息例如还可以包括从其做出转移的调用过程或者例程(例如,且包括源位置)和控制转移到的被被调过程或者例程(例如,且包括目标位置)。更具体地,在学习或者训练阶段中,第一次发生特定控制转移时,高速缓存未命中处理器计算从源到目标的特定控制转移的学习的规则集合的新规则。从同一源到同一目标的后续运行时间控制转移使用该计算的规则。以该方式,如果推定程序无编程缺陷(bug-free)且在程序运行期间实行非攻击程序(非恶意代码)和所有控制路径,在程序执行的结尾如由学习的规则集合指示的控制转移的记录的集合表示为对于该特定程序的所有有效或者可允许的控制转移。因此,学习的规则集合可以表示用于程序的CFI策略的初始或者第一规则集合。

[0424] 可以执行处理以验证表示用于该程序的CFI策略的学习的规则集合。该验证可以包括保证这些规则都不允许无效的控制转移。学习的规则集合的验证可以以任何适当的方式执行。例如,实施例可以运行验证每个规则的分析工具。该工具例如可以检查二进制或者对象代码,符号表和初始源代码等,以验证每个规则对应于允许的转移。为进一步图示,验

证可以检查具有标记的所有控制点(例如,分支或者转移源和目标)的二进制代码。以该方式,标记的二进制或者源代码表示所有潜在源和目标位置的有效集合,由此提供实际上可以在运行时间控制转移中使用的潜在源和目标的有效集合。记录的任何运行时间控制转移应该仅从源到目标发生,其中源和目标中的每一个包括在有效集合中。例如,标记的二进制或者源代码可以包括位置A1、A2、A3和A4。任何记录的控制转移应该包括作为A1、A2、A3或者A4的源,和作为A1、A2、A3或者A4的目标。如果由第一规则表示的记录的运行时间控制转移是从A1到B7,则因为B7不应该是控制转移的目标,可以使第一规则无效(例如,B7不包括在按照A1、A2、A3和A4标记的静态地确定的可能控制点的集合中)。在一个方面中,学习的规则集合可以特性化为可以经由作为验证处理的结果的规则排除而进一步减少的候选规则集合。

[0425] 用于已经验证的程序的CFI策略的初始或者学习的规则集合的所有规则然后可以用作对于该程序推行的CFI策略中包括的验证的规则集合。

[0426] 参考图54,示出了概述刚刚描述为用于学习、验证和使用策略规则的可以由根据在这里的技术的实施例执行的处理的实例。在602中,事实上最初可以不用CFI策略规则执行程序,以使得每个新控制转移导致规则高速缓存未命中并触发高速缓存未命中处理器生成关于在运行时间遇到的控制转移的新规则。新规则可以标识从源和目标的控制转移且可以包括在学习的规则的第一集合604中。在程序执行的结尾,学习的规则的第一集合604包括用于在运行时间发生的每个不同控制转移的规则。然后可以在606的处理中验证学习的规则的第一集合以保证每个规则表示有效的控制转移。606的处理可以使用如上所述的用于自动化规则验证的工具且还可以包括其他处理。例如,606的验证处理可以包括向用户呈现已经由工具验证的规则以用于控制转移有效的进一步确认。验证规则的第二集合608可以作为规则验证处理606的结果来生成。随后,验证规则608的第二集合可以由PUMP系统用作当在610中在第二时间点执行程序时推行的CFI策略。

[0427] 因此,对于先前602中的第一程序执行可以用于确定用于程序的有效控制转移的集合。但是,假定该单一程序执行实行所有控制路径可能是不合理的,由此608中标识为有效的控制转移可以表示小于所有可能有效的控制转移。在该情况下,可以如上所述使用CFI策略规则的验证的集合关于610执行处理。在运行时间期间,如果遇到导致规则高速缓存未命中的控制转移(例如,指示未预见的控制转移不具有608中的规则),则可以在运行时间执行附加校验,例如,以验证比如上面描述的控制转移(例如,使用以二进制代码标记或者在源程序中注释的可能控制点的集合)。如果控制转移被确定为无效,则可能触发故障或者例外。

[0428] 作为替代,如果遇到导致规则高速缓存未命中的控制转移由此表示不期望的运行时间控制转移,则高速缓存未命中处理器可以记录该不期望的转移规则以用于之后的验证,且还允许该不期望的控制转移以事实上附加的或者不同的策略继续进行。例如,对于未验证的控制转移,可以考虑转移为不可信的,所以可以修改策略以反映由于未验证的控制转移的不可信性质导致的较高级别保护。例如,该不期望的转移可以将控制转移到库例程。该库例程可以使用反映比事实上在不期望转移之前的更高级别保护和更少信任的策略来执行。对于验证的控制转移,第一堆栈保护策略可以事实上在不期望的控制转移之前的第一时间点,且第二堆栈保护策略事实上在不期望的控制转移之后。第一堆栈保护策略可以

推行静态过程授权。第一保护策略可以不包括以对象保护模型的如在这里其他地方描述的在对象级别的任何着色。在不期望的控制转移之后，根据具有严格对象着色的在这里其他地方描述的对象保护模型，事实上第二堆栈保护策略可以提供用于堆栈保护。因此，一旦遇到不期望的控制转移，执行的代码可以利用提供更紧密精细级别的粒度的堆栈保护的更限制性的第二堆栈保护策略。另外，一旦发生不期望的控制转移，可以以降低的优先级别继续程序执行。

[0429] 参考图55和图56，示出了可以使用验证规则的集合，比如用于上面描述的程序的CFI策略的规则在根据在这里的技术的实施例中执行的处理步骤的流程图620、630。流程图620描述关于不具有执行程序的CFI策略中的规则的不期望的控制转移执行的处理步骤的第一集合。流程图631描述关于不具有执行程序的CFI策略中的规则的不期望的控制转移执行的处理步骤的第二集合。

[0430] 参考流程图620，在步骤622，可以使用验证的规则的集合执行程序。在程序执行期间的步骤624，执行运行时间控制转移。在步骤626，确定是否存在规则高速缓存未命中，由此指示转移是不期望的。具体来说，如果存在在用于运行时间控制转移的验证规则的第二集合中的规则，则控制转移是期望的，其中步骤626评估为“否”，且处理以步骤628继续，在步骤628中执行控制转移且程序继续执行。

[0431] 如果步骤626估计“是”(例如，高速缓存未命中指示不期望的控制转移)，则处理以步骤632继续，在步骤632中，对于不期望的控制转移执行运行时间验证处理。具体来说，未命中处理器可以执行尝试验证不期望的转移的处理。规则验证处理的实例可以包括确定运行时间源和目标位置是否包括在如上所述可以使用标记的二进制代码、初始源程序和符号表等确定的潜在控制转移点的集合中。在步骤634，对确定不期望的控制转移的步骤632的验证处理是否有效进行确定。如果步骤634评估为“是”，则处理以步骤636继续，在步骤636，新规则被添加到用作用于程序的CFI策略的第二集合且程序的处理继续。如果步骤634评估为“否”，则例如可以通过导致陷阱来终止程序执行。

[0432] 参考流程图631，步骤622、624、626和628如上关于流程图620所述。如果步骤626评估为“是”，控制进行到步骤639，在步骤639可以记录不期望的控制转移(例如，记录的不期望的控制转移的候选规则)以用于之后验证。在步骤639，即使控制转移是不期望的，也允许程序继续执行。但是，在步骤639，程序执行例如使用比如以上提到的一个或多个限制性策略、减小的执行优先级等继续。

[0433] 比如上面描述的的上述处理可以关于比如污点跟踪的其他策略类似地执行。例如，对于污点跟踪，通过使得高速缓存未命中处理器“记录”每个高速缓存未命中，可以执行第一学习或者训练阶段以经由程序执行学习策略的规则。如在这里描述的，污点跟踪可以包括基于产生或者访问其的代码标记数据(例如，比如使用如在这里其他地方描述的CI。)基于代码或者源的污点数据的一个原因是确定适当地包括程序且程序不执行不需要的或者不适当的数据访问。例如，规则可以用于保证由JPEG解码器污染的数据从不流入密码数据库中，或者信用卡数据、社会保险号码或者其他个人信息仅由特定集合的一个或多个受限应用访问。通过确定污点跟踪策略，可以对于学习或者训练阶段执行处理，而没有在测试数据上运行的污点跟踪规则，其在第一次它看到数据的特定流动(例如，程序的哪些例程访问什么用户，什么用户输入写入到什么数据库等)就导致高速缓存处理器未命中并记录该

规则。以类似于如上对于CFI策略所述的方式，在第一学习阶段的测试运行的结尾，具有一组学习的规则的集合要在操作期间应用以保护程序。学习的规则集合的验证处理也可以使用如上对于CFI学习的规则集合所述的工具或者其他适当的方式执行。这种用于污点跟踪的验证处理可以包括保证每个数据流或者访问是适当的。

[0434] 此外，以类似于如关于流程图620和631描述的方式，验证的规则集合可以用于PUMP系统，其中高速缓存未命中处理程序处理用于不具有验证集合中的相应规则的任何数据访问的处理。类似于流程图620的处理，高速缓存未命中处理器然后也可以执行运行时间验证处理(例如，类似于步骤632)以确定用于数据访问或者数据流的候选规则是否有效，和允许程序执行继续(例如，类似于步骤634，636)或者不继续(例如，类似于步骤638)。替代地，类似于流程图631的处理，高速缓存未命中处理器可以记录可以离线验证(例如，不在运行时间期间)的不期望的数据访问或者数据流的候选规则，并例如使用更限制性的策略、减小的优先级等继续程序执行(例如，类似于步骤639)。

[0435] 以上实例描述通常的二进制学习处理。根据在这里的技术的实施例可以进一步支持做出关于是否允许事件(例如，控制转移或者数据访问)的决定时统计数据的使用。在至少一个实施例中，计数器可以添加到每个规则以对程序执行期间每个规则的使用次数计数。当从PUMP高速缓存逐出(evict)规则时，处理可以将累积的规则使用添加到可以用于提供关于规则使用的附加统计数据的全局、软件计数中。该计数还可以用于允许某些事情发生有限次数。例如，关于跟踪从源到目标的数据流的污点跟踪规则，可以对于源和目标之间的不期望的数据流动允许有限阈值量的数据(例如，由特定程序从特定数据库读取的数据量X)。一旦已经转移阈值量，则在源和目标之间不转移附加数据直到已经成功地验证相应的候选规则为止。通过具有阈值量的限制使用情况，PUMP系统(例如，未命中处理器)可以允许缺乏规则的指令发生某些有限次数。如应用于阈值的聚合或者计数可以以不同的方式进行。例如，考虑不期望的控制转移。非聚合的情况，高速缓存未命中处理器可以不允许没有验证规则的相同的不期望的控制转移发生多于5次。聚合的情况，比如跨程序的所有不期望的控制转移，可以允许程序做出最大100次的不期望的控制转移。这可能例如对于其对于不期望的控制转移或者不期望的数据访问的单个实例的发生是可接受的情况是有用的。例如，可以允许检查来自特定源的数据的单个查询。但是，如果对数据源(例如，特定数据库)执行阈值数目以上的查询，则程序应该被暂停(flagged)或者停止。

[0436] 更一般的统计数据情况可以用于学习正常行为的范围。例如，可以在学习阶段中执行程序以确定策略的不同规则的相对使用(例如，每个规则的使用比率)。例如，可以记录对于运行时间控制转移起用的每个规则的相对使用。理想地，可以使用许多不同数据集合对于程序执行这种执行以学习什么可以考虑为平均或者正常的程序行为。规则学习和验证然后可以导致用于验证的控制转移的规则集合(如上所述)和另外指示每个验证规则的相对使用的比率。验证规则和关联的使用比率两者可以在后续处理期间用作推行策略规则。在当推行策略时的后续程序执行期间，PUMP系统可以校验是否当前规则使用超出了期望比率。实施例例如可以包括用于规则的范围或者最大期望使用，其中起用大于最大值的规则的控制转移可以暂停。例如，起用大于最大值的特定控制转移规则的程序可以暂停以用于进一步检查或者分析。使用该机制，可以类似于监视网络行为以生成防火墙规则的方式来监视程序运行时间行为。统计学习算法可以用于捕获规则使用，和类似主存储器业务量和

高速缓存未命中率的可能的其他标准运行时间特性，以相对于攻击行为学习正常情况。在应用如上所述的有限使用的阈值的实施例中，如果程序展现异常的或者以其他方式可以考虑为不可信的其他运行时间行为，则使用限制可以大大地减小或者以其他方式设置为零。替代地，如果程序展现正常运行时间行为或者以其他方式可以考虑为可信的，与不可信方案相比使用限制可以设置地更高或者增加。

[0437] 以上技术可以用于确定策略的有效规则的集合，比如在CFI策略的规则中反映的有效控制转移，而不使得编译器输出任何附加信息。因此，根据在这里的技术的实施例可以具有每个策略的两个版本-一个用于学习阶段且另一用于后续推行。学习阶段可以用作自动化诊断模式以发现可允许的数据访问或者用于污点跟踪的流动，发现用于CFI策略的控制转移，等等。

[0438] 现在将描述的是可以使用RISC-V处理器在根据在这里的技术的实施例中使用的架构的实例。另外，以下描述可以关于在由处理器使用的未标记和标记的数据源之间执行基于处理器的调解(mediate)数据转移使用的技术。提供这种技术用于标记可能带入到系统中用于由处理器使用的外部不可信数据，以及从在系统内使用的标记数据除去标记以生成用于在系统外使用的未标记数据。

[0439] 参考图57，示出了在根据在这里的技术的实施例可以使用以在标记和未标记的数据之间调解的组件的实例。实例700包括RISC-V CPU 702、PUMP704、L1数据高速缓存706、L1指令高速缓存708、用于标记的数据转移的在系统内内部使用的互连结构710、引导ROM 712a、DRAM控制器(ctrl)712b和存储标记的数据的外部DRAM 712c。还包括作为硬件组件的添加标记714a和验证丢弃标记714b，用于从用于由处理器702使用的未标记存储器716转移外部未标记数据进来和将未标记数据转移出去到未标记存储器716的互连结构(interconnect fabric)715。应当注意，除未标记存储器716之外的外部未标记数据的其他源701可以连接到未标记结构715。例如，元素701可以包括闪存存储器中存储的未标记数据，可从网络访问的未标记数据等。DRAM控制器(ctrl)712b是用于从DRAM 712c读取数据和将数据写入到其的控制器。引导ROM 712a可以包括当引导系统时使用的引导代码。

[0440] 实例700图示单独的标记结构710和未标记结构，具有用于在这两个之间移动数据的处理器702。添加标记714a把未标记数据作为输入，且以指示数据是公开的(可以在在这里描述的系统外使用)和不可信的(因为源可以是未知的或者要不然不是来自已知的可信源)标记来标记其。在至少一个实施例中，未标记存储器716的未标记数据可以由714a接收。从716接收的未标记数据可以被加密，由此添加标记714a简单地添加不可信标记到所接收的加密数据。所接收的数据可以使用非对称密码，比如使用公钥-私钥对的公钥密码加密，或者现有技术中已知的其他适当的加密技术。所接收的数据可以以加密形式存储。如现有技术中已知的，对于所有者的公钥-私钥对，私钥仅对拥有者已知但是公钥是公开的且由其他人使用。第三方可以使用拥有者的公钥来加密发送给拥有者的信息。拥有者然后可以使用其私钥(不与任何其他人共享的)来解密接收到的加密信息。以类似的方式，拥有者可以使用他的私钥来加密信息，其中加密信息发送给第三方，该第三方使用拥有者的公钥来解密该加密信息。

[0441] 验证丢弃标记714b可以接收标记的加密数据并除去标记，由此导致未标记的加密数据输出到未标记存储器716。这种存储在存储器716中的未标记的加密数据例如可以用于

不使用标记和如使用在这里描述的PUMP执行的关联的元数据规则处理的另一系统和处理器上。

[0442] 在至少一个实施例中，在714a接收的未标记数据可以加密，如上所述，且签名以提供数据的完整性。此外，签名可以用于验证所接收的数据项以保证认证(authentication)和数据完整性(例如，从由签名的初始发送者发送之后还未修改，保证数据由签名数据的发送者发送)。例如，拥有者可以散列消息以产生散列值或者“提要”，且然后以拥有者的私钥加密提要以产生数字签名。拥有者可以发送消息和签名到第三方。第三方可以使用签名验证所接收的数据。首先，第三方可以使用拥有者的公钥来解密消息。可以通过计算解密消息的散列或者提要，以拥有者/签名者的公钥来解密签名以获得期望的提要或者散列，并比较计算的提要与解密的期望提要或者散列来验证签名。匹配的提要确认消息从其由拥有者签名后还未被修改。

[0443] 在操作中，比如加载指令的指令可以参考未标记存储器716中存储的数据，该数据然后转移到数据高速缓存706中以用于指令执行。对于这种加载指令，数据可以从716转移到715上用于由714a处理，714a输出标记数据(标记为不可信和公开的)。由714a输出的标记数据存储在L1数据高速缓存706中用于处理。以类似的方式，存储指令可以将来自数据高速缓存706数据存储到未标记存储器716中的位置。对于这种存储指令，数据可以在710上从706转移到输出未标记数据的验证丢弃标记714b。由71ba输出的未标记数据然后在715上发送以用于在716中存储。

[0444] 代码可以处理器702上执行以将未标记数据从716输入到系统中以例如用于存储在DRAM 712c上。以下可以表示输入未标记数据的代码的逻辑：

[0445] 1.由添加标记714a输出的标记数据可以存储在不可信缓存中(标记为公开的不可信的)。

[0446] 2.解密不可信缓存中存储的标记数据并存储在解码缓存中。因此解码缓存包括标记为公开的不可信的解密数据。

[0447] 3.执行验证处理以保证解码缓存包括有效的不受干扰的(uncompromised)数据。这种验证处理可以使用如在这里其他地方描述的和现有技术中已知的数字签名。

[0448] 4.如果解码缓存包括验证数据，则可以执行可信代码的第二部分以将标记为公开的不可信的解码缓存的数据转换为标记为可信的数据。可信的代码部分可以包括当执行时起用规则以将解码缓存的数据重新标记为可信的公开的一个或多个指令。现在标记为可信的公开的重新标记的数据可以存储在位于外部DRAM 712c中的可信缓存中。

[0449] 可信代码可以包括以特殊指令标记来标记的存储器指令，给予其授权以当执行时起用重新标记基准存储器位置的规则。例如，可信代码可以包括特殊标记的存储指令，其在具有公开的可信的新标记的目的地存储器位置(可信缓存)中存储标记为公开的不可信的数据(不可信缓存)。前述存储的可信指令例如可以由加载器特殊地标记。

[0450] 以下可以表示将数据从公开的不可信的重新标记为公开的可信的信任代码的逻辑：

[0451] for i＝1 to N

[0452] temp＝*untrusted buffer[i];

[0453] trusted buffer[i]＝temp;

[0454] 其中N是不可信缓存的长度且temp是用于执行的重新标记的临时缓存。第一指令，temp＝*untrusted buffer[i]，可以导致将不可信缓存的第一元素从未标记存储器716加载到temp中的加载指令。第二指令，trusted buffer[i]＝temp可以是将在temp中标记为公开的不可信的数据存储到具有公开的可信的新标记的可信缓存[i]的存储指令。因此，第二指令是如上所述特别标记以具有执行从不可信到可信的数据重新标记的授权的指令。

[0455] 以类似的方式，当712c的标记数据正在输出或者存储在未标记存储器716(或者任何未标记的存储器源701)中时，由加密数据项和生成签名的处理器702执行代码，其中加密的数据项和签名可以发送到714b，其中，在715上传输以存储在716中之前除去标记。

[0456] 作为对实例700的变型，存储器716和712c可以是统一的且互连结构710和715可以是统一的。在这种实施例中，可以限制允许未标记的存储器源701访问的地址范围。例如，对图58的实例720做出参考。实例720包括类似于如在700中标号的那些的组件，差异在于去除组件714a-b、715和716且存储器712c包括表示用于存储不可信的公开的标记数据的存储器712c的区域的部分U 722。比如不可信的DMA和I/O子系统的未标记的存储器源701可以限于使用存储器722的底部16或者256MB。在一个实施例中，U 722中存储的数据可以不明确地标记，而是具有在该有限范围中的地址的U中存储的所有数据可以隐含地标记并当做公开的和不可信的。作为变型，实施例可以以指示不可信的公开数据的永久标记预先标记部分U 722，且前述关联的永久元数据标记不能修改。规则可以防止处理器将其他数据存储到区域U 722中。例如由包括在701中的DMA执行的不可信的DMA操作可以限制为写入到区域U 722中。

[0457] 需要运行未移植(unported)的I/O处理代码的实施例可以在组件的不可信侧在专用I/O处理器上执行。例如，对图59的实例730做出参考。实例730包括类似于如在700中编号的那些的组件，差异在于添加了组件732、732a和732b。元素732是在没有PUMP和元数据规则处理的情况下运行的附加的RISC-V处理器。元素732a表示用于第二处理器732的数据高速缓存，且元素732b表示用于第二处理器732的指令高速缓存。数据高速缓存732可以连接到未标记的互连结构715。

[0458] 如在这里其他地方更详细地描述的，单独的I/O PUMP可以用作调解于未标记的数据源(例如，701、716)和由处理器702使用的标记存储器712c之间的另一替代。

[0459] 参考图60，示出了可以在关于在这里的技术在这里使用以调解于未标记的数据源(例如，701、716)和由处理器702使用的标记存储器712c之间的系统中包括的组件的另一实施例。实例740包括类似于实例700的组件，差异在于组件714a-b被除去和以内向部件(intern)742和外向部件(extern)744替代。在本实施例中，内向部件742和外向部件744可以是执行上述处理的硬件组件。具体来说，内向部件742可以包括处理接收的未标记数据并输出标记为可信的公开的验证数据项的硬件。可信的公开的标记数据项可以传递到结构710上以用于存储在由处理器702关于执行指令使用的数据高速缓存706中。内向部件742可以包括执行未标记的加密数据的验证处理，和假定成功验证进一步将接收的未标记数据标记为可信的公开的之硬件。外向部件744可以包括处理标记的未加密数据和输出签名的加密数据项的硬件。如果签名的加密数据项将要用于不执行如在这里描述的元数据规则处理的另一处理器上，则外向部件可以在加密之前除去标记。

[0460] 在最简单的情况下，内向部件742和外向部件744的硬件可以主管单个公钥-私钥

集合,其中也使用单个密钥集合执行签名和加密。密钥集合可以以由742和744使用的硬件来编码。在进一步的变型中,内向部件742和外向部件744的硬件可以主管多个公钥-私钥集合,其中也使用单个密钥集合之一(每个集合包括不同的公钥-私钥对)执行签名和加密。多个密钥集合可以以由742和744使用的硬件来编码。清除包括有进入的未标记数据的数据告诉内向部件单元742使用哪个密钥集合。因此,内向部件742可以执行包括多个密钥集合的硬件数据存储(例如,联合存储器)中的查找以选择期望的密钥集合。多个密钥集合中的每一个可以与不同标记相关联,所以由清除数据指示的特定密钥集合也指示标记数据将包括的特定标记。以该方式,由742输出的标记数据项的标记表示数据项是公开的和可信的,且还表示数据项是使用多个密钥集合中的特定的一个加密/解密的。在具有多个密钥集合的实施例中,外向部件744可以检查标记以确定关于加密和签名数据项使用多个密钥集合中的哪个特定的一个。因此,内向部件单元742处理提供隔离的硬件组件,其验证接收的未标记数据和执行标记,由此避免使得比如以上提到的可信代码部分的一部分代码具有标记数据的能力的需要。

[0461] 回去参考图1和图24,在级5中到PUMP 10的输入包括如在这里其他地方描述的标记。对于包括作为指令的操作数的存储器位置的指令,获得存储器输入和关联的标记,MR标记(有时在这里也称为Mtag)可能导致额外流水线停止(stall),由此在级5的PUMP 10不能进行直到它使得所有其输入包括MR标记为止。不是等待一提取从存储器读取的实际的MR标记值,而是根据在这里的技术可以执行处理以确定期望或者预测的MR标记,其然后可以用于确定R标记,即用于指令的结果的标记值(例如,目的地寄存器或者存储器位置,如果有的话)。在这种实施例中,可以在级6,即写回或者确认级(例如,参见图1的元素22和在图24中作为最后级6的确认级)做出最后校验,以确定预测的MR标记是否与从用于指令的操作数的存储器提取的动作MR标记相匹配。预测的MR标记的前述选择和使用以确定用于具有作为操作数的存储器位置的指令的Rtag,可以被称为Rtag预测加速器优化。

[0462] 参考图61,示出了图示用于Rtag预测加速器优化的根据在这里的技术的实施例的组件的实例800。实例800包括与如在这里其他地方描述(例如,图1和图24)的在级5的PUMP 10对应的PUMP 802,其具有用于执行Rtag预测加速器优化的附加特征。PUMP 802包括MR标记804a作为输入以及如在这里其他地方描述的其他PUMP输入804。PUMP 802还包括另一输入,预测选择符模式804b,其表示PUMP 802以正常处理模式运行(其中不执行MR标记预测处理的非预测模式)或者要不然以预测模式运行(其中执行MR标记预测处理)。在至少一个实施例中,预测模式选择符804b可以是0,表示其中没有确定预测的MR标记值的用于PUMP的正常处理模式,或者是1,表示其中确定预测的MR标记值的用于PUMP的预测模式。当预测模式选择符是1时,PUMP 802可以以预测模式执行,其中可以掩蔽或者忽略MR标记804a输入且PUMP 802产生预测的MR标记805c作为输出。当预测模式选择符是0时,PUMP 802可以以比如在这里其他地方描述的正常处理模式执行,其中MR标记804a是到PUMP 802的输入且不生成输出805c。

[0463] 如在实例800中所示的,级5中的PUMP 802的附加输出包括R标记805a和PC新标记805b。当使用预测的MR标记时,可以确定用于预测的MR标记的规则,其中规则指定用于R标记的关联标记。当以预测模式操作时,预测的MR标记805c是到流水线的级6 808的附加输入。元素808可以表示如在这里其他地方描述的确认或者写回级(例如,图1和图24)。因此,

元素808a通常可以表示如在这里其他地方描述的除805a-c之外的其他级6输入。

[0464] 在级6 808中，当PUMP 802以预测模式操作时可以执行附加处理808b。元素808b指示可以在级6 808中执行校验，该校验比较预测的MR标记与从用于指令的操作数的存储器获得的动作MR标记。换句话说，808b通过确定预测的MR标记是否匹配从存储器获得的MR标记来评估PUMP 802是否正确地预测了MR标记值。如果预测的MR标记不匹配从存储器获得的MR标记，则不正确的规则被触发且由PUMP 802用于以不正确的预测的MR标记来确定R标记805a。现在必须选择正确的规则(根据实际的MR标记)且用于确定修订的R标记。因此，如果预测的MR标记不匹配MR标记，则确定规则高速缓存未命中且执行高速缓存未命中处理。按照在这里其他地方的描述，高速缓存未命中处理可以包括使用MR标记选择和评估正确规则的处理。

[0465] 加载/读取和存储/写入指令是可以包括存储器位置作为受益于预测的MR标记的使用的操作数的实施例中的指令的实例。到PUMP的其他输入804包括除MR标记804a之外的其他或者剩余输入标记的集合。例如，如关于图23所示的一个实施例可以具有5个输入标记-PC标记、CI标记、OP1标记、OP2标记和MR标记-和2个输出标记-PC新和R标记。因此，(除MR标记之外)剩余输入标记的集合包括以下4个标记，PC标记、CI标记、OP1标记、OP2标记。确定预测的MR标记或者指令可以包括确定具有匹配指令的4个标记(例如，PC标记，CI标记，OP1标记，OP2标记)的标记值的一个或多个规则的集合。在有些情况下，仅单个规则可以包括4个输入标记的匹配标记值。在该情况下，单个匹配规则也对可以用作预测的MR标记805c的MR标记而指定值。另外，可以使用4个输入标记和预测的MR标记来评估规则以进一步确定R标记805a。

[0466] 例如，考虑具有典型加载和存储操作的存储器安全策略。加载操作可以使用指针从源存储器位置加载数据，其中第一规则指示关于源存储器位置的标记或者颜色应该匹配指针的标记或者颜色。存储操作可以使用指针存储数据到目标存储器位置，其中第二规则指示关于目标存储器位置的标记或者颜色应该匹配指针的标记或者颜色。对于加载指令，第一规则可以是具有匹配加载指令的4个输入标记，即PC标记、CI标记、OP1标记和OP2标记的标记值的唯一规则。第一规则的MR标记可以用作预测的MR标记805c。另外，可以使用4个输入标记和预测的MR标记的集合来确定第一规则的R标记。以类似的方式，对于存储指令，第二规则可以是具有匹配存储指令的4个输入标记，即PC标记、CI标记、OP1标记和OP2标记的标记值的唯一规则。第二规则的MR标记可以用作预测的MR标记805c。另外，可以使用4个输入标记和预测的MR标记的集合来确定第二规则的R标记。

[0467] 在其他实例中，具有匹配指令的输入标记，即PC标记、CI标记、OP1标记和OP2标记的标记的策略的规则集合可以包括多个匹配规则，其中每个匹配规则标识可以用作预测的MR标记805c的不同的可允许或者候选MR标记。实施例可以使用任何适当的技术来选择多个可允许MR标记之一用作预测的MR标记。例如，实施例可以选择最普遍或者可能发生的可允许MR标记的集合的MR标记。最可能发生的MR标记可以基于先前的观察或者规则制订(profiling)。作为替代，实施例可以设置预测的MR标记为先前或者最近的MR标记。在最坏情况下，如果预测的MR标记不匹配曾经接收的实际的MR标记，则可以执行如在这里描述的高速缓存未命中处理，以与指令的其他输入标记一起使用实际的MR标记来确定正确的规则。

[0468] 在至少一个实施例中，可以创建当PUMP以预测模式操作时使用的用于存储器操作的规则的类别。规则的列表可以被称为“预测存储器标记”规则的类别。对于“预测存储器标记”规则，MR标记804a不用作到PUMP 802的输入，且因此不关于由PUMP执行的各种查找来使用。例如，用于“预测存储器标记”规则的关心/不关心位向量可以将MR标记作为不关心来处理。另外，“预测存储器标记”规则可以省略作为输入的MR标记而是指定预测的MR标记为输出。如上所述，如果存在匹配PC标记、CI标记、OP1标记和OP2标记的特定输入标记集合的多个匹配的正常规则，则与匹配规则集合对应的单个“预测存储器标记”规则可以指定预测的MR标记作为输出，该输出是最普遍或者最期望的MR标记。在一个实施例中，与匹配规则集合对应的单个“预测存储器标记”规则可以将由PUMP 802接收到的最后或者先前MR标记指定为预测的MR标记。

[0469] 策略逻辑可以决定是否插入或者使用“预测存储器标记”规则。实施例可以保持每个策略的2个版本，其中第一版本包括用于当以预测模式操作时使用的策略“预测存储器标记”规则，且第二版本包括用于当以正常处理模式或者非预测模式操作时使用的正常或者非预测策略规则。如果当使用“预测存储器标记”规则时在级6的808b执行的校验对于给定指令失败，则高速缓存未命中处理可以使用正常规则集合(例如，上述第二版本的规则)执行处理以确定指令的匹配规则。

[0470] 在使用RISC-V处理器和架构的实施例中，预测模式选择符804b可以具有相应的PUMP CSR。在这里其他地方更详细地描述了在使用RISC-V架构的实施例中的CSR的使用。

[0471] 参考图62，示出了可以在根据在这里的技术的实施例中执行的处理步骤的流程图。流程图840概述如上关于实例800所述的处理。如上所述，实例800中图示的PUMP 802表示在级5的PUMP提供到处理器流水线的级6的输入。在至少一个实施例中，流程图840的步骤842、844、846、848和852可以表示在PUMP内具体表现的如上所述在级5中执行的处理步骤和使用的特定策略规则，且步骤854、856和858可以在如上所述的级6中执行。

[0472] 在步骤842，做出关于预测模式是否开启/启用由此表示PUMP使用“预测存储器标记”规则以预测模式操作的确定。如果步骤842评估为“否”，则控制进行到步骤846，在步骤846，PUMP使用正常规则以正常或者非预测模式操作。如果步骤842评估为“是”，则控制进行到步骤844，在步骤844，做出关于当前指令是否是存储器输入操作指令的确定。如果步骤842评估为“否”，则控制进行到步骤846。如果步骤844评估为“是”，则控制进行到步骤848，在步骤848，PUMP使用“预测存储器标记”规则以预测模式操作。在步骤848，可以确定用于指令的匹配“预测存储器标记”规则。在步骤852，可以使用来自步骤848的匹配″预测存储器标记″规则来确定当前指令的R标记。在步骤854，做出关于是否预测的MR标记匹配实际MR标记的确定。如果步骤854评估为“否”，则控制进行到步骤856以通过起用规则未命中处理器来执行规则高速缓存未命中处理。如果步骤856评估为“是”，则控制进行到步骤858，在步骤858，如以包括预测的MR标记的规则确定的R标记被用作R标记PUMP输出。

[0473] 作为实例800的变型，参考图63，其图示包括以正常非预测模式运行的PUMP 802以及以预测模式运行的第二PUMP 822的实施例的组件。在该实例中，以预测模式运行的PUMP 822也可以被称为MR标记预测PUMP，其中预测模式选择符822b总是开(例如，1)。类似地，对于PUMP 802，预测模式选择符804b也可以是关(例如，0)。MR标记预测PUMP 822可以仅使用″预测存储器标记″规则，且PUMP 802可以仅使用策略规则的正常或者非预测版本。在这种实

施例中,PUMP 802和822可以在级5中并行操作。元素828可以表示与MR标记预测PUMP 822相关联的级5和6处理和组件。元素829可以表示与以正常模式操作的PUMP 802相关联的级5和6处理和组件。在829,PUMP 802输出如关于实例800的,差异在于预测的MR标记805c不再由PUMP 802输出。另外,级6 808不执行校验808b。元素828可以包括以类似于实例800的方式执行处理的组件,差异在于MR标记预测PUMP 822仅使用如上所述的"预测存储器标记"规则。

[0474] 对级6(808)进行修改以从MR标记预测PUMP 822取得PUMP输出Rtag 805a和PC新标记805b,并从PUMP 802取得输出Rtag 805d和PC新标记805e。另外,在级6中,基于预测的MR标记是否匹配实际MR标记(例如,如由808a表示的),在Rtags 805a和805d之间做出选择,且还在PC新标记805c和805e之间做出选择。如果存在预测的MR标记和实际的MR标记之间的匹配(例如,808a评估为1或者真),则使用来自预测的PUMP 822的标记(例如,Rtag 805a和PC新标记805b),并丢弃来自非预测的PUMP 802的标记(例如,Rtag 805d和PCnew标记805e)。如果存在预测的MR标记和实际的MR标记之间的不匹配(例如,808a评估为0或者假),则丢弃来自预测的PUMP 822的标记805a-c并使用来自非预测的PUMP 802的标记805d-e。非预测PUMP 802比预测PUMP 822的输出805a-c晚一周期地提供其输出输出805d-e,所以当需要关于PCnew标记和MR标记来自级5的PUMP输出作为到级6的输入用于处理时,这引入停止到级6中,等待前述级6输入。非预测PUMP 802当其被选择时也可能经历PUMP规则高速缓存未命中,在这样的情况下,就像如本公开内其他地方描述的典型规则高速缓存未命中那样处理。

[0475] 参考级6 808,元素850和852表示多路复用器。元素808a可以表示用于基于预测的MR标记是否匹配MR标记的逻辑结果从850和852中的每一个选择输入的选择符。如果前述两个标记值匹配,Rtag 805a被选为到850中的输入,其提供为表示在级6的在最后Rtag输出的所选的Rtag 850a。否则如果前述两个标记值不匹配,Rtag 805d被选为到850中的输入,其提供为所选的Rtag 850a。另外,如果前述两个标记值匹配,则PCnew标记805b被选为到852中的输入,其提供为表示级6中的在最后PCnew标记输出的所选的PCnew标记852a。否则如果前述标记值不匹配,则PCnew标记805e被选为到852中的输入,其提供为所选的PCnew标记852a。

[0476] 现在将描述的是使用可以在根据在这里的技术的实施例中使用的着色分配存储器的技术。

[0477] 比如以C编程语言代码的用户程序的用户程序可以包括对关于存储器分配和去分配使用的例程的调用。例如,malloc和free是C标准库中的例程且可以链接到用户程序的可执行中。因此,malloc和free作为用户进程地址空间中的例程与可以起用malloc和free的其他用户代码一起执行。起用malloc用于动态存储器分配以分配通过执行代码使用的存储器块。在至少一个实施例中,malloc可以具有在起用时指定的输入,表示要分配的存储器块的大小,由此malloc返回指向分配的存储器块的指针。程序使用由malloc返回的指针来访问分配的存储器块。在至少一个实施例中,起用free以释放或者去分配先前以malloc分配的存储器。当不再需要使用malloc分配的存储器块时,指针(如由malloc返回的)可以作为输入变元传递到free,由此free去分配存储器(定位为由指针表示的地址)以使得其可以用于其他目的。在根据在这里的技术的实施例中的在处理器上执行的用户代码可以执行这种对malloc和free或者类似地执行存储器分配和去分配的其他例程或功能的调用。执行动态

存储器分配的比如malloc和free的例程可以利用关于分配的存储器的存储器管理元数据。在以下段落中，这种用于存储器管理的元数据可以被称为malloc元数据且是独特的，且附加于包括标记及由指针标记指向的其他元数据的在这里描述的基于标记的元数据(例如，其中对执行用户代码不可访问的且由比如关于实例1000和在这里其他地方描述的元数据处理器或者子系统处理的基于标记的元数据)。malloc元数据例如可以包括关于分配的存储器块的信息，比如分配的存储器块的大小，和指向用于随后分配的存储器块的malloc元数据部分的指针。

[0478] 参考图64，示出了图示比如关于malloc的存储器分配的实例。在实例1100中，程序可以执行对malloc的第一调用以分配所请求的大小的第一存储器块。响应于此，malloc可以分配所请求的大小的存储器块1102b并返回表示存储器块1102b的开始地址的指针P1。用户程序然后可以使用指针P1或者基于从P1的偏移的另一地址来存储数据到分配的存储器块1102b和从分配的存储器块1102b读取数据。另外，为了动态存储器管理的目的，malloc也可以对于每个的存储器块分配用于它自己的malloc元数据的存储区1102a。元素1102a表示用于存储分配的存储器块1102b的malloc元数据的由malloc分配和使用的存储器部分。以类似的方式，用户程序可以随后执行对malloc的第二调用以分配第二存储器块。元素1104a表示响应于该第二调用由malloc分配的存储器部分，其中1104a用于存储malloc元数据。元素1104b表示分配的第二存储器块，其中P2是返回到用户程序以访问第二存储器块的指针。以类似的方式，用户程序可以随后执行对malloc的第三调用以分配第三存储器块。元素1106a表示响应于该第三调用由malloc分配的存储器部分，其中1106a用于存储malloc元数据。元素1106b表示分配的第三存储器块，其中P2是返回到用户程序以访问第三存储器块的指针。

[0479] 在执行代码不再需要分配的存储器块，比如1102b之后，代码可以进行对free的调用以释放存储器块1102b，以使得存储器块1102b去分配且可以用于其他目的。当做出这种对free的调用时可以返回指针P1。以类似的方式，当不再需要存储器块1104b-c时，可以做出对free的调用分别指定指针P2和P3。

[0480] 通过由malloc返回给执行用户代码的比如P1的指针，用户代码可以无意或者有意地访问malloc元数据，因为保存malloc元数据的存储器部分1102a的地址映射到执行代码的地址空间。例如，用户代码可以给另一指针P4分配存储器部分1102a中的地址(例如，P4＝P1-2)且然后读取或者写入到由指针P4标识的存储器位置。因此，用户代码例如可以重写1102a中存储的malloc元数据且读取1102a中存储的malloc元数据。以该方式，在由P4标识的地址执行到存储器位置的写入可能损坏malloc元数据部分1102a。更一般地，前述可以由关于malloc元数据部分1102a、1104a和1106a中的任意的用户代码执行。

[0481] 关于对free的调用，用户代码可以指定与先前使用malloc分配的分配的存储器块的开始地址对应的指针。例如，用户代码可以执行对free的调用，指定前述指针P4为变元而不是P1、P2或者P3。例如假定malloc关于对malloc的调用对于每个malloc元数据部分1102a-c分配X字节块(例如，X是非零整数)。例程free可以在从第一地址(P4-X)到第二地址(P4-1)的存储器位置分别表示跨越比如1102a的malloc元数据部分的开始和结束地址的假定下来执行处理。在该情况下，由free执行的处理可能使用损坏的malloc元数据部分1102a，例如导致不期望的运行时间性能和/或动态存储器管理错误。

[0482] 实施例可以使用在这里描述的技术保护malloc元数据部分1102a、1104a和1106a以避免由比如用户代码的其他执行代码执行的重写造成的损坏。这种技术可以包括以特定颜色或者标记来标记代码和/或数据和推行规则，以仅允许比如在这里其他地方描述的期望的访问和操作。

[0483] 参考图65，在至少一个实施例中，由malloc和free使用的存储器部分可以以由如在这里描述的元数据处理使用的第一标记来着色或者标记，且由用户代码使用的其他存储器部分(如由malloc分配的)可以以由如在这里描述的元数据处理使用的第二不同标记来着色或者标记。在实例1100中，由malloc和free使用的数据部分(包括malloc元数据)可以着色或者标记红色，且用户数据部分(由malloc分配用于由用户代码使用的存储器块)可以着色或者标记蓝色。实施例可以使得至少一个标记或者颜色排他地保留用于着色或者标记由malloc和free使用的存储器位置。在该实例中，红色是保留用于标记由malloc和free使用的存储器位置的颜色。如在这里其他地方描述的，实施例也可以保留用于执行用户代码的一个或多个颜色或者标记。在至少一个实施例中，分配用于由用户程序使用的所有存储器可以以相同颜色标记。作为变型，实施例可以对于每个对malloc的调用使用不同标记，且因此对于分配的每个单独的存储器块使用不同颜色。在该实例1110中，为说明的简单起见，仅单个颜色蓝色用于标记由malloc用于用户程序分配的全部存储器块。

[0484] 元素1111可以表示指定用于相应的存储器位置1113的标记。元素1112a、1114a和1116a分别表示用于malloc元数据部分1102a、1104a和1106a的标记。元素1112b、1114b和1116b分别表示用于用户的存储器块1102b、1104b和1106b的标记，它们是经由如上所述对malloc做出的调用通过用户代码由malloc分配给用户的。

[0485] 元素1112a、1114a和1116a表示分别在1102a、1104a和1106a中的每个存储器位置标记为红色。元素1112b、1114b和1116b表示分别在1102b、1104b和1106b中的每个存储器位置标记为蓝色。

[0486] 通常，关于触发以由1111表示的标记来着色1113的存储器块的规则，实施例可以使用指令标记、着色指针或者前述的组合，且还推行存储器安全策略，由此仅malloc和free能够访问malloc元数据区域1102a、1104a和1106a，且用户代码不能访问。

[0487] 在第一实施例中，可以比如以特殊指令标记(例如，CI标记)由加载器来标记(例如，指令标记)malloc和free的代码。malloc和free两者可以以相同的唯一或者特殊指令标记(例如，以tmem的相同CI标记来标记的malloc和free代码)来标记或者可以每个以他们自己的唯一或者特殊指令标记(例如，以tmalloc标记的malloc代码和以tfree标记的free代码)来标记。malloc的代码可以包括当执行时触发执行比如实例1110中的着色的规则的存储指令。free的代码可以包括当执行时比如通过以表示空存储器的F标记来重新标记块或者malloc元数据部分的每个存储器单元，而触发重新初始化或者去分配malloc元数据部分(例如，1102a、1104a和1106a)或者先前进行了存储器分配的存储器块(例如，1102b、1104b和1106b)的规则的存储指令。此外，在第一实施例中，存储器安全策略可以包括通过特定指令，比如加载和存储指令的执行来触发的规则，由此规则仅允许以以上提到的一个或多个特定指令标记来标记的指令：1)访问malloc元数据部分1102a、1104a和1106a和2)执行如在实例1110中的存储器块着色。这种规则通常可以校验CI标记以保证着色或者访问1102a、1104a和1106a中的任意的存储器单元的每个指令具有表示malloc或者free的特殊指令标

记。

[0488] 在第二实施例中,不使用特殊指令标记,实施例可以使用着色的指针,其具有通过比如加载和存储指令的特定指令的执行触发的存储器安全策略的规则。加载器可以标记用颜色红色参考malloc元数据部分1102a、1104a和1106a的malloc和free的指针。malloc的代码可以包括当执行时触发执行比如实例1110中的着色的规则的存储指令。free的代码可以包括当执行时比如通过以表示空存储器的F标记来重新标记存储器单元,而触发重新初始化或者去分配malloc元数据部分(例如,1102a、1104a和1106a)或者先前进行了存储器分配的存储器块(例如,1102b、1104b和1106b)的规则的存储指令。存储器安全策略可以包括通过比如加载和存储指令的特定指令的执行来触发的规则,由此规则仅允许以使用红色着色的指针参考存储器单元的指令来访问malloc元数据部分1102a、1104a和1106a。这种规则通常可以校验MR标记以保证访问1102a、1104a和1106a中的任意的存储器单元的存储器指令使用具有与存储器单元的第二颜色匹配的第一颜色的指针。

[0489] 在第三实施例中,如上所述特殊指令标记和着色的指针两者可以组合地使用。以下是在这种第三实施例中可以使用的指令和规则的实例。按照在这里的其他讨论,以下实例基于以下来使用规则:到元数据处理的5个输入标记,PC(程序计数器)、CI(当前指令)、OP1(当前指令的操作数1)、OP2(当前指令的操作数2)、MR(在当前指令中参考的存储器位置,如果有的话)和两个传播或者生成的标记,PCnew(用于下一指令的下一PC的新PC标记)和R(用于当前指令的结果的标记;用于标记当前指令的结果存储到其中的目的地寄存器或者存储器位置)。另外,“-”表示标记的不关心。在这种实施例中,加载器可以用特殊标记tmalloc来标记malloc的指令且用特殊标记tfiree来标记free的指令。可以使用以下提到的触发的规则创建着色的指针。

[0490] 关于malloc,通过执行malloc的代码部分而触发的元数据规则处理可以经由作为malloc的代码部分中的存储指令的结果起用的第一规则,生成用于到比如1102b的新分配的存储器块的指针的标记。例如,malloc C代码可以是“P1=next free”,其中next free(接下来释放)是到1113中的下一空闲存储器位置的指针,且存储指令可以是“move R1,R2”,其中寄存器R1是包括地址next free的源寄存器且寄存器R2是作为指针P1的目的地寄存器。寄存器R1可以是OP1(具有OP1标记)且寄存器R2可以是结果或者目的地寄存器(具有作为激发的规则的结果传播或者生成的R标记)。malloc的代码部分可以包括也以表示malloc码中包括指令的特殊标记tmalloc来标记的指令,比如前述移动指令。在至少一个实施例中,加载器可以用特殊代码标记tmalloc来标记malloc的指令。第一规则可以用标记蓝色来标记到分配的存储器块1102b的指针P1。作为以上malloc中的移动指令的结果触发的第一规则可以是:

[0491] mv rule1A:(-,t malloc,blue-predecessor,-,-)→(-,blue)

[0492] 以上规则仅当CI标记是tmalloc且因此用于malloc中标记的移动指令时激发。假定由malloc使用的指针是P1,以上mv rule1A以标记或者颜色蓝色来标记寄存器R2中存储的标记P1,以表示其是到蓝色存储器位置(例如,以蓝色标记来标记的存储器位置)的指针。

[0493] 以蓝色标记的指针P1然后可以用于malloc的另一第二存储指令,以将0或者某些其他初始值写到分配的存储器块1102b中的每个字。例如“*P1=0”可以包括在导致“Store R3,(R2)”的malloc C代码中,其中R3是包括零(0)的源寄存器操作数OP1,且R2是包括地址

P1的OP2寄存器。在该存储指令中,“(R2)”是操作数MR且还表示作为存储指令的目标或者目的地的存储器位置。另外,malloc中的以上存储指令也可以标记为tmalloc且可以导致触发如下的第二特殊存储规则:

[0494] store 2A:(-,t malloc,-,blue,F)→(-,blue)

[0495] 在返回标记指针P1到起用malloc的用户代码之前。

[0496] 以上store rule2A(存储规则2A)仅当CI标记是tmalloc,R2(表示P1)中的指针或者地址标记为蓝色时,和当由P1指向的存储器位置MR具有F标记时激发。前述存储器位置*P1假定在着色存储器位置为蓝色之前以“F”标记。在该实例中,F表示空闲存储器位置。产生的用于存储器位置的MR标记表示存储器位置的蓝色标记。

[0497] 因此,malloc可以包括导致触发用于正被分配的存储器块的每个存储器位置的以上提到的第二规则的代码。

[0498] malloc也可以包括触发用于初始化malloc元数据部分1102a、1104a和1106a的以下描述的附加规则(例如,类似于以上的移动(mv)规则1A和存储规则2A)的代码。例如,malloc C代码可以是“(P1-2)=MD区域”,其中MD区域是到malloc元数据区域1102a中的指针且移动指令可以是“移动R7,R8”,其中寄存器R7是包括地址“P1-2”的源寄存器且寄存器R8是作为指针MD区域的目的地寄存器。由以上移动指令触发的规则可以是:

[0499] mv rule 1B:(-,t malloc md,-,-,-)→(-,red)

[0500] 以标记MD区域指针为红色。

[0501] malloc也可以包括触发以下提到的存储规则2B(类似于以上的存储规则2A)以标记malloc元数据部分的每个存储器位置以使得存储分别用于malloc元数据部分1102a、1104a和1106a的标记1112a、1114a、1116a的代码。例如假定size(大小)是表示进行malloc的存储器块1102b的大小的整数,且“*(P1-2)=size”包括在导致“storeR6,(R7)”的malloc C代码中,其中R6是包括大小值的源寄存器操作数OP1,且R7是包括地址P1的OP2寄存器。在该存储指令中,“(R7-2)”是操作数MR且还表示作为存储指令的目标或者目的地的MR 1102a中的存储器位置。存储规则2B可以是:

[0502] store 2B:(-,t malloc md,-,red,F)→(-,red)

[0503] store 2D:(-,t malloc md,-,red,red)→(-,red)

[0504] 其如果存储指令标记为tmalloc,如果包括地址P1的R7寄存器标记为红色且如果MR操作数标记为F则执行存储。应当注意,实施例也可以包括作为以上提到的存储2B规则的变型的以上提到的存储规则2D(存储2D),由此存储2D规则可以在期望更新元数据值的情况下使用。

[0505] 在之后时间点,free可以包括导致触发以下提到的存储规则3以比如当释放或者去分配蓝色着色的块1102b时重新标记先前进行了存储器分配的存储器块(例如,分配用于用户代码使用的存储器块)的存储器位置的比如“*P=0”的代码。加载器可以以tfree着色或者标记free的指令。例程free可以包括导致“Store R4,(R1)”的C代码语句“*p=0”,其中R4是包括零的源寄存器操作数OP1,R1是包括要初始化的存储器位置的地址的OP2寄存器,且“(R1)”表示具有包括到存储器位置的地址的R1的存储器操作数MR。存储规则3可以是:

[0506] store rule 3:(-,t free,-,blue,blue)→(-,F)

[0507] 因此,free可以包括导致触发用于正被去分配的存储器块的每个存储器位置的以

上提到的第三规则的代码,其中先前使用用于由用户代码(例如,用于存储除了malloc元数据之外的数据)使用的malloc分配了存储器块。存储规则3校验以保证CI标记＝t free且存储器位置和到其的指针两者都具有相同颜色,蓝色。

[0508] 应当注意,“蓝色”的MR标记通常可以是先前由malloc使用以着色分配的用户存储器块的任何颜色。

[0509] free的代码也可以包括触发以下关于重新标记比如1112a的malloc元数据部分的每个存储器位置而描述的移动(mv)规则1C和存储规则4的代码。free的代码可以包括触发类似于以上的移动(mv)规则1B的以下提到的移动(mv)规则1C的代码。移动(mv)规则1C可以是:

[0510] mv rule1C:(-,t free,-,-,-)→(-,red)

[0511] 以标记由free关于使用存储规则4重新标记而使用的红色指针。

[0512] 可以触发以下存储规则4(类似于以上存储规则3)以重新标记分别用于元数据部分1102a、1104a和1106a的比如retag 1112a、1114a、1116a的malloc元数据部分的每个存储器位置。存储规则4可以是:

[0513] store rule 4:(-,t free,-,red,red)→(-,F)

[0514] 如果存储指令标记tfree,且如果MR操作数使用标记为红色的指针,则其执行存储。存储器位置以“F”标记以显示表示其为空闲的。

[0515] 在第四实施例中,PC标记可以用于提供malloc和free以向malloc和free提供从malloc元数据部分1102a、1104a和1106a读取数据和向其写入数据的足够特权、访问或者授权,且还排除其他代码访问前述元数据部分。PC标记例如关于实例430在这里其他地方描述,实例430使用不同PC标记值基于每个进程提供不同特权、访问或者授权。以类似的方式,特殊或者唯一PC标记值可以用于向malloc和free提供授权以相对于malloc元数据部分1102a、1104a和1106a执行加载和存储操作。为进一步图示,malloc可以包括以tmalloc(例如当执行指令时CI标记＝tmalloc)标记的指令。malloc也可以包括当执行时触发规则的应用的代码,该规则传播或者产生特定PC标记作为表示访问malloc元数据部分1102a、1104a和1106a的特权或者授权的输出。malloc可以包括第一指令INS1,比如:

[0516] Add 0,R2

[0517] 其中R2是malloc元数据部分中的地址,比如区域1102a中的地址P6,且(R2)表示具有着色红色的1102a中的地址P6的存储器位置。前述指令INS1当执行时,可以导致生成具有比如X1的标记值的PCnew,其中X1表示访问1102a需要的特权。在该情况下,对于以上第一指令INS1触发的规则可以是:

[0518] add:(-,tmalloc,--,red,-)→(X1,red)

[0519] 而以颜色红色着色R2,且还设置PC为X1以表示到具有在R2中存储的地址(例如,地址P6)的存储器位置的读/写访问。随后,malloc可以包括第二指令INS2,“store R3,(R2)”以将来自寄存器R3(例如,OP1)的值存储到具有地址P6(P6存储在R2中)的存储器位置中。对于以上第二指令INS2触发的规则可以是:

[0520] store:(XI,tmalloc,-,red,red)→(PCdefault,red)

[0521] 其中PCnew清除或者复位为PCdefault,其是不表示访问malloc元数据部分1102a的特权的默认PC标记。因此,在该特定实例中,第一ADD指令触发给malloc许可对1102a的

读/写访问的特权或者授权的规则。在执行malloc的以上第二指令执行写入之后，传播的PC标记从用于对1102a的读/写访问的malloc除去特权或者授权。作为变型，实施例可以包括具有包括触发规则的指令的编程逻辑的malloc的版本，该规则通过生成X1的PCnew标记来许可malloc对1102a的读/写访问(例如，编程逻辑包括触发以上提到的规则的ADD指令INS1)。在返回之前的malloc的结尾，可以执行收尾(epoligue)，其包括当执行时触发通过生成PCdefault的PCnew标记来除去malloc对1102a的读/写访问的规则(例如，收尾包括触发以上提到的规则的存储指令INS2)。

[0522] 以类似的方式，free可以包括起用规则以生成或者传播PCnew标记值以向free提供对1102a的访问的指令。应用的规则可以基于特定进程来传播或者产生特定PC标记作为表示期望访问、特权或者授权的输出，由此可以由不同PC标记值表示特定允许的特权、访问或者授权。

[0523] 应当注意，前述图示用于所有进行malloc的存储器块的单个颜色蓝色和用于所有malloc元数据部分的单个颜色红色。更一般地，如在这里其他地方描述的，可以向malloc提供生成如着色堆存储器的不同部分可能需要的无限制数目的新颜色的授权。如在这里其他地方讨论的，例如，malloc可以被给予和初始预定一个或多个颜色或者标记的集合，且可以随后从初始预定的集合生成需要的标记。例如，malloc的初始预定集合可以包括黄色或者Y和红色或者R。对于执行的进程，malloc可以对于每个对malloc的调用生成基于新鲜的Y的标记(例如，Y1、Y2、Y3、...)以分配由用户代码使用的新存储器块(例如，除了用于malloc元数据存储之外的)。因此，不同的基于Y的标记可以用于着色每个进行malloc的存储器块1102b、1104b和1106b(例如，以Y1着色1102b，以Y2着色1104b，以Y3着色1106b)。Malloc可以生成用于对于对malloc的每个调用创建的每个不同malloc元数据部分的基于新鲜的R的标记(例如，R1、R2、R3、...)。因此，基于R的标记可以用于每个以不同的基于R的标记来着色malloc元数据部分1102a、1104a、1106a(例如，以R1着色1102a，以R2着色1104a，以R3着色1106a)。由malloc使用的当前或者最后的基于R的标记和当前或者最后的基于Y的标记可以经由当执行malloc指令时触发的规则而存储为状态信息。例如，malloc可以包括触发将最后的基于Y的标记，Y9存储为第一存储器位置的标记的规则的指令。Y9可以生成为Rtag。后续指令可以再次参考以保存的最后的基于Y的标记，Y9标记的相同第一存储器位置，其中，后续指令触发以下规则：1)基于最后的基于Y的标记Y9生成新标记Y10，和2)将标记Y10保存为关于第一存储器位置的标记。Y10可以生成为Rtag。由后续指令触发的规则可以指示例如确定Rtag为MRtag+1，其中MRtag是用于后续指令的Y9。

[0524] 现在将描述的是可以用作关于使用硬件加速的未命中处理的元数据处理的优化的技术。通常，在这里的实施例中使用的某些策略可能导致频繁的规则高速缓存未命中，且用于这种策略的高速缓存未命中处理器可能占用很多周期来运行。在某些策略中，各种规则输入之间的关系可能就逻辑地确定结果或者输出而言相当简单，且可以因此是硬布线的和以专用硬件来快速解决。

[0525] 结果，使用硬件(HW)规则高速缓存未命中处理器实现的这种策略可以比不使用这种硬件加速的其他策略以短得多的时间量解决。在这种实施例中，比如用于一个或多个所选的策略的高速缓存未命中处理器的策略组件可以以专用硬件实现。因此，根据在这里的技术的实施例可以单独地，或者与使用软件规则高速缓存未命中处理器的软件定义的策略

组件结合地使用这种硬件支持的策略。

[0526] 作为一个实例,考虑使用存储器安全着色的存储器安全策略。关于比如在这里其他地方描述的存储器安全策略,可以着色存储器单元和指针,由此关于加载和存储操作两者起用的规则可以仅允许其中指针颜色匹配存储器单元的颜色的存储器参考。例如,对于加载指令触发的规则可以用于推行其中(例如,其中寄存器是比如OP1的操作数的寄存器标记的)指针颜色等于存储器-单元颜色(例如,比如Mtag的存储器位置标记)的策略。存储器安全策略可以通过以简单地捕获该许多颜色的相等颜色关系的许多不同具体规则来填充PUMP规则高速缓存,对容量造成问题,增加容量未命中率。在如在这里描述的没有预加载规则高速缓存的一些实施例中,需要强制规则高速缓存未命中以插入这些规则中的每一个。因为存储器安全策略规则通常可以关于执行用户代码触发,可以使用HW规则高速缓存未命中处理器而不是软件规则高速缓存未命中处理器来支持存储器安全策略规则。

[0527] 在这种实施例中,HW规则高速缓存未命中处理器可以生成或者计算在规则高速缓存未命中发生时插入到高速缓存中的新规则。例如,用于存储器安全的未命中处理器可以使用如HW规则高速缓存未命中处理器的硬件实现,其对于加载指令,比较OP1tag与Mtag,如果OP1tag等于Mtag,则HW规则高速缓存未命中处理器可以用分配Mtag的Rtag来生成新规则。例如,如果指针PTR是红色且由PTR指向的存储器单元是红色,则允许起用规则的指令且产生的标记Rtag应该是红色的。为生成前述作为要插入规则高速缓存中的新规则,HW高速缓存未命中处理器可以首先比较OP1tag与Mtag。如果它们不相等,存在规则违规且不允许该指令(例如,使得处理器停止执行)。如果HW规则高速缓存未命中处理器确定OP1tag等于Mtag,则HW规则高速缓存未命中处理器可以生成包括Opcode＝load,OP1tag＝红色,Mtag＝红色且Rtag＝红色的新规则作为硬件的输出(规则的所有其他标记输入和输出可以是不关心的),其中生成的规则然后可以被插入到规则高速缓存中。

[0528] 参考图66,示出了图示在根据在这里的技术的实施例中的硬件实现的高速缓存未命中处理器的实例。实例1300包括1301,其图示输入到PUMP规则高速缓存1302(例如,图22)的输入1302a以执行查找以确定匹配输入1302a的规则是否在高速缓存中。如果是,则基于高速缓存中存储的规则确定输出1302b。按照在这里其他地方的讨论,输入1302a可以包括操作码和输入标记-PCtag、CItag、OP1tag、OP2tag、Mtag。输出1302b可以包括比如PCnew标记和Rtag的规则的输出标记。关于以软件实现高速缓存未命中处理器的根据在这里的技术的实施例,在规则高速缓存未命中的发生时,可以起用软件高速缓存未命中处理器,由此未命中处理器的代码执行和计算用于导致当前规则高速缓存未命中的输入1302a的新规则。高速缓存未命中处理器首先确定输入是否与可允许规则一致(例如,对于存储器安全负载规则,OP1tag等于Mtag),且如果是,则计算用于特定输入1302a的输出(例如,确定Rtag为mtag)由此生成用于输入1302a的规则。新规则(基于输入1302a和计算的未命中处理器的输出的组合)插入到规则高速缓存中。按照在这里其他地方的讨论,新规则可以包括操作码,输入标记-PCtag、CItag、OP1tag、OP2tag、Mtag和输出标记-PCnewtag、Rtag。

[0529] 元素1303图示可以在根据在这里的技术的实施例中使用的HW规则高速缓存未命中处理器1304而不是软件规则高速缓存未命中处理器。在这种实施例中,HW规则高速缓存未命中处理器1304可以使用例如包括门级逻辑及其他硬件组件的专用硬件实现。在这种实施例中,HW未命中处理器1304可以取得与PUMP规则高速缓存1302相同的输入1302a并可以

使用其硬件生成将输出PUMP规则高速缓存的相同输出1302b。随后，可以通过如上所述组合操作码、输入标记和输出标记形成新规则。新规则然后可以存储在PUMP规则高速缓存中(例如图22)。

[0530] 在至少一个实施例中，用于存储器安全策略的HW规则高速缓存未命中处理器可以如上所述以硬件实现(例如，使用门级逻辑)，以将规则加载到高速缓存中，可以将存储器-单元标记简单地从Mtag复制到Rtag且立即执行PUMP规则插入。注意在该简单情况下，不需要去参考存储器和执行存储器中的任何数据结构操作。

[0531] 另外，在至少一个实施例中，存储器安全可以将存储器单元的标记实现为一对标记：(1)存储器-单元颜色标记，(2)关于存储器单元中的指针的指针-颜色标记。存储器-安全加速可以包括专用高速缓存以执行Mtag和OP2tag到关于存储的新Rtag中的组合，和执行从Mtag对提取指针-标记以放置到用于加载的Rtag上。这些高速缓存的未命中可以使用更简单的专用软件处理器。虽然对于比如存储器安全的单个(非合成)策略描述了前述，相同通用技术可以应用于关于UCP的复合策略的组件。

[0532] 实施例也可以对于比如那些期望共同参考的规则的有限公共子集，使用HW规则高速缓存未命中处理器来执行硬件加速。例如，在存储器安全中，用于运算期间的加载/存储和传播的规则是最标准和程式化的。存在其他的非公共规则，用于最初着色的存储器区域和在空的情况收回存储器区域。这种非公共规则可能导致使用如在这里描述的典型的规则未命中处理器而不是以硬件支持实现。

[0533] 在至少一个实施例中，HW规则高速缓存未命中处理器可以将映射功能直接实现为门级逻辑。例如，这种门级逻辑可以对于规则将输入标记映射到输出标记，比如对于存储器安全策略的存储指令规则将Mtag映射映射到Rtag。作为另一实例，用于CFI(控制流完整性)策略的HW规则高速缓存未命中处理器可以使用门级逻辑以使得控制流目标或者目的地的标记是到允许调用者的集合(例如，允许控制转移到标记的特定控制流目标或者目的地的源位置或者地址)的指针，允许CFI HW规则高速缓存未命中处理器从该集合读取以得到匹配。作为又一实例，堆栈保护策略可以以允许硬件从一个导出另一个的方式(例如，它们可以仅相差几位，且即使这些标记是已通过一起分配堆栈-帧-代码标记指针和堆栈-帧-存储器-单元标记指针的指针也可以安排为这样)来编码堆栈-帧-代码标记和关联的堆栈-帧-存储器-单元标记；因此，推行堆栈保护策略的HW规则高速缓存未命中处理器将能够确定用于创建的标记(在从堆栈指针创建的情况下)，或者关于这种代码内的存储器参考的需要(在读取或者写入的情况下)。

[0534] 作为使用HW规则高速缓存未命中处理器计算或者确定后来插入到PUMP规则高速缓存中的新规则的变型，实施例可以实际上硬布线策略的一个或多个规则的逻辑，其中这些规则以硬件完全地实现和推行且因此不存储在PUMP规则高速缓存中。例如，不使用用于策略的HW规则高速缓存未命中处理器和PUMP规则高速缓存，实施例可以使用硬件来推行和编码策略的规则(例如，以比如门级逻辑和电路的硬件实现的策略的规则)。在这种使用PUMP规则高速缓存和HW指定规则两者的实施例中，可以执行PUMP规则高速缓存以及HW指定规则两者的规则查找。在该情况下，可以起用未命中处理器(例如，HW规则高速缓存未命中处理器或者软件未命中处理器)以响应于未找到用于PUMP规则高速缓存或者HW指定规则中的特定输入的规则而确定/计算新规则。

[0535] 复合策略呈现附加的挑战和机会。按照在这里其他地方的讨论，复合策略包括对于指令同时推行的多个策略。挑战是复合策略需要解析几个不同策略组件。机会是用于复合策略的解析的整个序列可以以数据高速缓存、UCP高速缓存(复合策略中的每个策略组件一个UCP高速缓存)和CTAG高速缓存，对于复合策略的所有不同策略组件使用HW规则高速缓存未命中处理器来进行硬件支持。从先前经验，共同挑战是新分配的存储器(例如，使用malloc)，由此新存储器颜色标记，在何处导致强制的规则高速缓存未命中。在这些情况下，存储器安全策略组件需要新规则，但是其他组件可能已经在UCP高速缓存中有它们的规则。通过经由用于最高级复合策略和用于存储器安全颜色匹配的HW规则高速缓存未命中处理器的硬件加速，可以用以硬件运行且查询高速缓存(例如，数据高速缓存、UCP高速缓存和CTAG高速缓存)的小的有限状态机来执行存储器规则解析，而不是运行基于软件的未命中处理器代码需要几百到几千周期来解析这些规则。

[0536] 在至少一个实施例中，UCP高速缓存可以由组件策略分解且全部并行解析以产生标记结果的复合集合以然后反馈到CTAG高速缓存中。如果可以通过它们的UCP高速缓存或者通过简单的硬件规则，比如用于存储器安全的硬件规则来解析所有策略，则用于查找UCP高速缓存的总时间将是单个策略的时间而不是与策略的数目成正比。如果组件策略的数目固定且与提供的硬件匹配，这完美地工作。尽管如此，轻微变型简单地跨固定的可用的UCP高速缓存数目来分布组件策略，以使得顺序UCP高速缓存解析的数目仅是组件标记的数目与物理的UCP高速缓存的比率。

[0537] 参考图67，示出了实例1310，其图示关于可以在根据在这里的技术的实施例中使用的复合策略的HW规则高速缓存未命中处理器的使用。在该特定实例中，3个策略包括该复合策略，由此对于相同指令同时推行所有3个策略，虽然更一般地复合策略可以包括任意数目的策略且不限于3个。元素1314a-c是用于包括复合策略的3个策略的HW规则高速缓存未命中处理器。输入1312可以提供给HW规则高速缓存未命中处理器1314a-c中的每一个，其分别确定用于特定策略的规则输出1316a-c(例如，HW规则高速缓存未命中1314a确定包括用于策略A的Rtag和PCnew标记的输出1316a；HW规则高速缓存未命中处理器1314b确定包括用于策略B的Rtag和PCnew标记的输出1316b)。随后，输出1316a-c可以组合为表示用于3个策略的复合Rtag和PCnew标记的单个复合结果1318。组合输出1316a-c以确定复合结果1318也可以使用硬件或者软件实现。新规则可以与复合结果1318(例如，Rtag和PCnew标记的复合值)一起插入到高速缓存中，其中新规则包括用于触发规则高速缓存未命中处理的特定指令的输入1312(例如，操作码和输入标记)。

[0538] 另外，虽然在实例1310中未示出，实施例可以与在根据在这里的技术的实施例中的HW规则高速缓存未命中处理器1314a-c结合地使用UCP高速缓存和CTAG高速缓存。如在这里其他地方描述的(例如，关于图21、图23和图24)，策略A、B和C中的每一个可以具有它自己的UCP高速缓存，用于高速缓存最近的策略结果标记的结果(例如，用于策略A的UCP高速缓存存储由未命中处理器1314a基于指令的操作码和输入标记的组合最近计算的结果标记-PCnewtag和Rtag结果)。如在这里其他地方描述的(例如，关于图21、图23和图24)，CTAG高速缓存可以存储如可以从比如策略A、B和C的多个复合策略输出的，用于各个Rtag值的特定组合的Rtag的复合结果。CTAG高速缓存也可以存储如可以从比如策略A、B和C的多个复合策略输出的，用于各个pcnew标记值的特定组合的PCnewtag的复合结果。因此，从输出1316a-c生

成复合结果1318的硬件可以使用来自CTAG高速缓存的信息来确定复合结果1318。另外，HW规则高速缓存未命中处理器1314a-c也可以对于策略A、B和C将来自UCP高速缓存的作为输入信息。

[0539] 作为如在实例1310中具有用于复合策略的全部3个策略的HW规则高速缓存未命中处理器的替代，实施例可以选择性地选择实现用于包括复合策略的一个或多个，但是少于所有这种策略的HW规则高速缓存未命中处理器。在这种实施例中，规则高速缓存未命中处理器的一部分可以以硬件实现且复合策略的规则高速缓存未命中处理器的剩余部分可以以软件实现，如在这里其他地方描述的。

[0540] 应当注意，如在这里描述的某些策略例如可以关于存储器安全策略分配新标记。在至少一个实施例中，可以给用于可以分配新标记的比如存储器安全的策略的HW规则高速缓存未命中处理器提供新标记值的基于FIFO的高速缓存(例如，可以用作生成的新分配的标记值的标记的高速缓存。如果分配的标记是表示地址的指针，则高速缓存包括地址或者指针而不是标记值)，基于HW的处理器可以使用该基于FIFO的高速缓存。以该方式，HW规则高速缓存未命中处理器可以通过从基于FIFO的高速缓存读取顶部项来简单地执行分配。周期性地，可以在元数据处理域中执行软件处理器而以可用于分配的新标记来重新填充基于FIFO的高速缓存。

[0541] 在这里描述其中在元数据处理域和用户代码或者执行域的“正常”代码处理之间存在完全的和严格的隔离的实施例。作为变型，实施例可以采取更宽松的方法和扩展前述严格隔离模型，其仍然不允许由用户代码或者执行域对元数据处理域的信息的修改或者写入，但是可以允许信息/值由元数据域返回到用户代码或者执行域。

[0542] 现在将描述的是可以在至少一个实施例中包括的技术，其可以利用前述更宽松的方法，由此元数据处理域返回可以由在正常代码处理或者执行域中执行的代码使用或者参考的值(例如，元数据处理返回作为到正常或者用户代码执行域的输入的值)。例如，如在这里其他地方描述的，实施例可以使用malloc和free例程，其中这些例程使得它们的代码以指令标记来标记，向它们提供需要的独特能力，以使得malloc和free的代码当执行时，触发允许malloc和free执行访问malloc元数据、生成新颜色标记、以这种新颜色标记来标记用户数据区域等的它们的处理的能力的规则。前述情况提供了唯一地分配给malloc和free的这种特权或者能力，而排除其他代码，比如用户代码的例外。现在考虑这种实施例，其中malloc和free执行它们的处理且使用代码标记来由加载器特殊地标记malloc和free代码，其采用唯一地标识这种代码属于具有特殊执行特权的malloc和free的一个或多个特殊代码标记。在这种实施例中，可以是以下情况：用户代码做出对free的调用，例如提供已经损坏或者以其他方式不指向现在去分配的先前分配的存储区域的开始的指针PTR1。可以由free推定PTR1是指向先前由malloc分配的用户数据区域的第一位置。例如，Free可以假定到用户数据区域的预定结构和存储器堆的关联的存储器位置，比如关于图64和图65描述的，其中malloc元数据存储在相对于分配的用户数据区域的预定位置中。

[0543] 现在将描述的是可以在实施例中使用以使得PUMP返回值到代码执行域的技术。

[0544] 参考图68的实例1200，示出了如关于图65的实例1110描述的元素1111和1113，进一步以以下讨论的指针PTR1和PTR2注释。假定用户代码以PTR1起用free，意在去分配存储器块1102b。P1可以表示free期望的指针或者地址。但是，在该实例中的PTR1可以表示通常

表示除了P1之外的不同地址的损坏的或者不正确的地址(例如,PTR1可以标识存储器1102b中的位置,或者可以表示甚至不指向堆中的地址)。虽然PTR1已经损坏或者以其他方式不指向正确的存储器位置P1,free可以使用PTR1执行处理,以使用相对于PTR1的相对寻址来访问malloc元数据,其中malloc元数据假定为以其预定义的结构、格式或者布局存在。例如,由free使用的malloc元数据区域可以假定为位于紧接在分配的用户数据部分之前,如在图64和图65中。在这种情况下,free的代码可以基于预定布局确定它在处理中使用以去分配特定的存储器块的malloc元数据是位于PTR1之前的特定偏移OFF1。例如,参考图68,free可以假定PTR1＝P1,其中PTR1可以由用户代码关于对free的调用来提供。Free可以基于预定义的数据布局使用如上所述的相对寻址,针对该预定义的数据布局,要去分配的存储器块1102b的相应malloc元数据1102a应该开始于具有地址PTR2＝PTR1-OFF1的存储器位置。在该实例中,PTR1不等于P1,且PTR1实际上指向分配的存储器块1102b中的某处,以使得地址计算PTR2＝PTR1-OFF1也在用户分配的存储器块1102b中(PTR2表示由free使用的关联的malloc元数据的期望的开始)。

[0545] 在这种由用户代码在free起用时提供的PTR1不指向期望位置P1且由此PTR2表示由free使用的malloc元数据的推测的开始的情况下,free的代码可能使用这种数据作为其malloc元数据而不正确地访问存储在存储器块1102b中的数据,导致违规、中断或者陷阱(例如,可能由于由PUMP检测到的规则违规,或者free的执行期间的其他代码执行错误状态)。因此,用户处理空间或者域中执行的代码的执行,在如使用来自用户代码的PTR1对free的调用中起用的例程free的执行期间,可能由于前述违规而中止。不是使得例程free导致用户代码的前述中止,可以期望允许free的代码查询PUMP,或更一般地,元数据处理返回值。返回的值例如可以是表示与PTR2相关联的颜色(如将由free的代码使用以访问malloc元数据)是否实际上指向有效或者期望malloc元数据区域的布尔值。使用这种返回PUMP或者元数据处理值允许free基于与在地址PTR2的存储器位置相关联的颜色是否表示有效的malloc元数据颜色,比如红色,来执行不同条件处理。如果PTR2标识如经由PTR2的颜色确定的无效的malloc元数据区域,则例程free可以执行某些恢复或者其他动作。这种动作可以与使得用户代码由于规则违规、陷阱、中断或者其他执行错误而中止相比,是更期望的。

[0546] 在使用RISC-V指令集以实现返回元数据处理值的至少一个实施例中,新指令gmd(get-metadata-info(得到元数据信息))可以添加到RISC-V指令集,比如:

[0547] gmd R1,R2,R3

[0548] 其中

[0549] R1包括由PUMP或者元数据处理返回的结果值;

[0550] R2包括以具有地址PTR2的存储器位置的颜色标记的地址PTR2;和

[0551] R3以如对于有效的malloc元数据区域期望的有效颜色来标记。

[0552] 因此,R2和R3可以是作为输入或者源操作数的寄存器,且R1可以是包括结果或者输出的寄存器。在该特定实例中,R3tag可以是红色,表示有效的malloc元数据区域,和R2tag的颜色可以是蓝色。由新指令起用的规则可以输出一返回值作为布尔值,在该实例中表示是否R2tag＝R3tag,其中前述布尔值结果可以是用户执行代码的地址空间中包括的对free可访问的寄存器R1中存储的由元数据规则处理输出的返回值(例如,PUMP输出)。应当

注意,R1可以用Rtag标记来标记,作为按照在这里其他地方讨论的结果标记。

[0553] 以下描述可以使用具有如上所述的PTR1、PTR2和OFF1的类C伪代码描述而由free的代码执行的逻辑处理:

```
free (char *PTR1)
    PTR2 = PTR1-OFF1;   /** PTR
    if (IS_RED (PTR2)) then
```

[0554]

```
        PTR2 指向有效着色的 malloc 元数据区域。执行处理以去分配。
    else
        PTR2 不指向有效地着色的 malloc 元数据区域。执行恢复处理。
```

[0555] 在以上逻辑处理中,可以校验IS_RED以查看PTR2是否是颜色RED(红色)。

[0556] 由以上提到的else块的代码执行的恢复处理例如可以尝试通过从PTR2向后或者向前搜索来定位有效的malloc元数据区域的开始。以上提到的else块的代码可以以更多定义的期望方式,比如通过表示无效地着色的指针PTR2的运行时间错误消息/条件来允许用户代码的终止。

[0557] 新指令Get metadata info R1、R2、R3可以包括在例如作为以C所写的free例程的汇编和链接代码的结果生成的指令中以执行以上提到的逻辑处理。实施例可能想要控制或者限制可以允许什么特定代码部分以执行该新指令。PUMP规则可以用于调解(mediate)或者限制何时允许该新指令由什么例程执行。例如,可以允许free或者malloc的代码执行新Get metadata info指令而不是用户代码。其中一些在这里描述的任何适当的技术可以用于向例程free提供执行返回PUMP值的新指令需要的特权或者授权。例如,free的代码可以以表示允许free执行新指令的特殊指令标记来标记。例如,加载器可以以特殊标记NI来标记free代码中出现的新指令。规则可以用于将什么代码可以被允许起用新指令调解或者限制为具有NI的指令标记(CI标记)的那些。

[0558] 参考图69,示出了图示在根据在这里的技术的实施例中的元数据规则处理的输入和输出的实例1210。元素1212通常可以表示如在这里描述的元数据处理。到元数据处理的输入1212a例如可以包括如在这里描述的各种标记和操作码信息。由元数据处理1212生成的输出1214可以包括如在这里其他地方描述的Rtag 1214a和PCtag 1214b。另外,元数据处理可以生成作为返回值1214c的新输出。返回值1214c可以放置于寄存器,比如具有新指令的以上表示的R1中,其是对用户处理空间/代码执行可访问的寄存器的集合。按照在这里其他地方的描述,1214a和1214b表示分别置于结果(例如,结果寄存器或者存储器位置)和PC上的标记,由此1214a-b对用户处理空间/代码执行不可访问。应当注意,元数据处理是否返回返回值1214c可以是关于特定指令或者操作码有条件的。例如,如在这里其他地方描述的,可以使用多路复用器基于操作码如关于图27-图33描述的来过滤元数据处理输出以启用/禁用输出返回值1214c。在该实例中的值1214c表示当操作码是新指令的操作码时是否R2tag=R3tag的逻辑结果。否则,如果操作码不表示新指令操作码,则可以由元数据处理有条件地返回默认值作为返回值1214c。

[0559] 参考图70,示出了实例1220,其图示当由元数据处理返回值到用户执行域时,比如当执行如上所述的free的代码中包括的新指令时,在根据在这里的技术的实施例中可以执

行的组件和处理。为说明的简单，实例1220图示仅采用有用于该新返回值的目的地或者结果寄存器R1和关联结果寄存器的元数据处理的逻辑和组件。元素1222a通常可以表示如在这里其他地方描述的用于元数据处理的PUMP输入(例如，比如在该实例中的R2tag和R3tag的标记，操作码)。PUMP 1222可以包括用于新指令的规则，其校验代码标记是否是NI，并输出表示是否R2tag＝R3tag的逻辑结果(例如，在该实例中，OP1表示第一输入源操作数R2且OP2表示第二输入源操作数R3)。该规则导致输出前述逻辑结果1221a。元素1225可以表示多路复用器，具有用作用于多路复用器1225的选择符1225a的操作码。当当前指令的操作码表示新指令Get metadata info的特定操作码时，1225a导致选择1221a以作为返回值1214c输出。否则，如果操作码不是新指令的操作码，则1225a导致选择默认返回值1222a作为返回值1214c。返回值1214c是目的地寄存器，RD，1228中存储的PUMP输出(例如，1214c存储在D1 1228b中，表示寄存器RD中存储的内容对用户处理地址空间中执行的代码可访问)。因为RD 1228是结果寄存器，所以规则也可以导致以Rtag标记RD(例如，Rtag存储在标记部分T1 1228a中，其中T1是RD寄存器的标记字)。在至少一个实施例中，Rtag可以是表示RD包括新指令的输出的特殊标记SPEC1。基于如在这里其他地方描述的符号逻辑，其中到规则的标记输入是(PCtag、CItag、OP1tag、OP2tag、MR标记)且规则输出是(PCtag，Rtag)，与第三输出，NEWOUT一起表示新返回值1214c，规则可以表示为：

[0560] gmd:(-,NI,t1,t2,-)→(-,SPEC1,NEWOUT)

[0561] 其中如果t1＝t2则NEWOUT＝1且否则NEWOUT＝0。

[0562] 更一般地，新指令的前述使用可以用于根据在这里的技术的实施例中以返回作为可以由特殊标记(例如，以NI标识)的代码使用的任何适当的和所需的值的值，以表示可经由一个或多个起用的元数据处理规则允许的新指令的发生。

[0563] 替代实施例可以避免添加新指令。这可以通过在该情况下代码-标记现有指令以控制该行为和设置关心位以选择值输出来完成。另一替代可以添加值-输出-关心-位，其也是PUMP的输出，以使得规则可以确定值输出应该流到RD值结果的情况。该第二情况允许操作码当不标记时正常地表现，且仅当给定适当的代码标记时展现该特殊行为。

[0564] 现在将描述的是可以用于保证以从序列的第一指令到最后指令的指定次序作为单个单元或者完全序列原子地执行特定指令序列的技术。另外，这种技术保证除了到序列的第一指令之外没有到指令序列中的控制转移和除了经由序列的最后指定指令之外没有到序列之外的转移或者退出。例如，考虑图71的简单指令序列。

[0565] 在实例1400中，示出了2个指令1402和1404的序列。第一指令1402将内容从存储器位置(其中存储器位置的地址存储在R2中)读取或者加载到R1。第二指令将零(0)写入或者存储到相同存储器位置(存储器位置具有R2中存储的地址)。可以提供这种指令序列用于保证从存储器位置(具有R2中指定的地址)读取的值仅使用一次，由此在从存储器位置读取值之后立即从存储器位置擦除或者清零旧的值。因此存储器位置的清零由序列的第二指令1404执行，且需要作为从存储器位置读取值的第一指令1402之后序列中的下一指令来执行。

[0566] 在使用RISC架构的根据在这里的技术的至少一个实施例中，规则可以用于推行前述数据项的线性和1400的指令序列的原子性。在这种实施例中，可以更新PC标记(PC新标记)以传递序列中的下一期望指令的序列的状态。在至少一个实施例中，一个解决方案是以

CI标记来标记指令1402,表示其作为线性读指令。另外,表示具有R2中存储的地址的存储器位置的(R2)可以归类和标记为具有唯一元数据id X1的线性变量(例如,从全部其他线性变量中,X1唯一地标识该线性变量)。作为第一指令1402的结果可以触发第一规则。第一规则可以指示仅允许标记为线性读取的指令来从线性变量读取。另外,产生的PCnew标记可以是清除-线性-变量-X1-下一个以表示执行的下一指令需要清除线性变量X1。可以作为执行第二指令1404的结果触发第二规则,其中以表示用于初始化或者清除存储器位置的特殊值的特殊EMPTY(空)标记来标记操作数的值,零(写入到存储器位置)。另外,需要存储器位置是表示来自紧接着之前的指令1402的特定标记的线性变量的线性变量X1。如果1402之后的第二指令做了将EMPTY值写入线性变量X1之外的任何事,则导致陷阱。因此,第二规则顺序地推行期望的和1400中的指令序列的原子性。

[0567] 更具体地,假定第一指令1402是将来自具有R2中存储的地址的存储器位置的内容载入R1中的加载指令。加载指令可以如下:

[0568] load R1,(R2)

[0569] 另外,假定第二指令1404是将零(0)移动到具有在R2中存储的地址的存储器位置中的移动指令。移动指令可以如下:

[0570] move 0,(R2)

[0571] 遵循在这里其他地方提到的惯例,规则可以定义为操作码,输入标记-PCtag、CItag、OP1tag、OP2tag、Mtag和输出标记-PCnewtag、Rtag。基于前述规则惯例,对于第一加载指令,OP1是R1,OP2是R2,且(R2)是标记为Mtag的存储器位置。由第一加载指令触发的第一规则可以是:

[0572] load:(-,linear read,-,-,linear variable X1)→(clear-linear-variable-X1-next,-)

[0573] 基于前述规则惯例,对于第二存储指令,OP1是0,OP2是R2,且(R2)是标记为Mtag的存储器位置。由第二移动指令触发的第二规则可以是:

[0574] move:(clear-linear-variable-X1-next,-,EMPTY,-,linear variable X1)→(default tag,-)

[0575] 该实例示出标记和规则可以怎样用于保证特定指令序列的不可划分性。本领域技术人员可以容易地看到该通用技术可以应用于其中期望推行数据可以仅以作为特定指令序列的一部分的特定方式访问的许多其他场景。本领域技术人员也可以容易地看到可以对于其中需要指令序列的严格推行的任何情况采用该技术。如上所述,该通用技术涉及用一特殊标记来标记来自序列中的指令N的新PC(例如,PCnew标记),该特殊标记被校验作为由粘连的序列中的下一期望指令N+1触发的规则中的PC标记。

[0576] 现在将描述的是作为引导或者启动基于RISC-V架构的在根据在这里的技术的实施例中的系统的一部分执行的技术。以下段落可以涉及在这里其他地方描述的各种CSR,比如关于其中可以关于元数据处理域使用这种CSR的实例900。

[0577] 如在这里其他地方描述的,自举标记可以是硬布线的或者特定ROM位置中存储的值。作为引导系统的一部分,可以执行自举代码的分段,其通常执行包括初始化不同CSR、存储器等的初始化。作为初始化的一部分,这种处理也以从初始自举标记导出的默认标记值来最初地标记存储器位置。在至少一个实施例中,可以以用作系统中的所有其他标记从其

导出的初始“种子”标记的特殊自举标记来初始化比如boottag CSR(例如,如在实例900中的sboottag CSR)的CSR。比如加载器的不同代码实体可以使它们的指令被特殊地标记(例如,设置为特殊指令标记的CI标记),以由此指定加载器为具有执行不具有特殊指令标记的其他代码不允许进行的任务的特定特权或者授权。可以使用由加载器的代码触发的规则来推行前述,该规则检查CI标记以保证其是特殊标记以使得触发的规则执行期望的标记操作。因此,例如,用于标记加载器的指令的特殊CI标记可以作为通过执行作为启动处理的一部分的代码所触发的特殊规则的结果从自举标记生成或者导出。通常,一旦标记了代码或者存储的指令的某些部分,可以通过这种标记的代码的执行来触发规则以生成更多期望的标记且还将这种生成的标记置于代码和数据上。以下更加详细地描述前述及其他方面。

[0578] 在系统的引导或者启动时,比如在标记模式CSR中存储的标记模式(例如,实例900的901r)可以最初是关闭的(例如,实例910的911a)。可以执行自举ROM程序,首先直接将默认标记CSR(例如,实例900的901c)设置为特殊默认标记值。随后,自举程序可以将标记模式CSR设置为由此元数据处理域在所有结果上写入如在默认标记CSR中存储的默认标记的模式。换句话说,当在defaulttag标记模式中(例如,实例900的911b),PUMP输出Rtag总是默认标记值。

[0579] 随后,在存储器位置已经初始化和以默认标记来标记之后,可以执行处理以生成将用于进一步生成或者导出所有其他后续标记的标记的初始集合(例如,该初始集合可以进一步用于以无限制的方式导出标记的一个或多个其他生成)。这种处理可以包括执行触发规则以生成标记的初始集合的指令序列或者代码段。在该情况下,标记模式可以设置为在代码段的执行期间启用PUMP的适当的标记模式级别。例如,参考实例910,如果在系统管理程序(hypervisor)模式下执行引导代码,则标记模式可以设置为如由911e表示的x110或者如由911f表示的x111,以在代码段的执行期间启用PUMP,由此作为代码段指令的结果触发和推行规则。

[0580] 应当注意,在执行以上提到的代码段之前,可以执行处理以证实或者验证代码段。例如,在至少一个实施例中,以上提到的代码段可以以加密形式存储,其中在执行之前,解密和证实或者验证代码段(例如,比如使用数字签名)以保证代码段还未被篡改或者修改。

[0581] 为进一步图示,自举程序可以包括在当启用PUMP时执行以由此生成标记的初始集合的的以上提到代码段4指令中:

[0582] 1.R1←read boottag CSR

[0583] 2.Add R2←R1+1

[0584] 3.Add R3←R2+1

[0585] 4.Add R4←R3+1

[0586] 在以上指令1中,R1是通用寄存器。指令1读取boottag CSR,将boottag CSR中存储的值和boottag CSR中存储的标记两者转移到R1上。boottag CSR设置为保存处理器复位期间的特定标记或者由特权模式设置为包括其标记的CSR的写入。从boottag CSR的读取也可以清除boottag CSR,以使得在引导期间的该初始提取之后其不可用于提取。

[0587] 在形成以上形式“Add Rn←Ry+1”的指令2-4的每一个ADD指令中,其中Rn表示存储Add的结果的目标或者结果寄存器,且其中,Ry也表示寄存器源操作数。前述代码段的指令2可以触发第二规则,该第二规则从第一标记生成第二标记且将第二标记置于由R2指向的存

储器位置上。前述代码段的指令3可以触发第三规则，该第三规则进一步从第二标记生成第三标记且将第三标记置于由R3指向的存储器位置上。前述代码段的指令4可以触发第四规则，该第四规则进一步从第三标记生成第四标记且将第四标记置于指向R4的存储器位置上。以该方式，前述代码段可以用于生成存储为寄存器上的标记值的4个标记的初始集合。前述通用技术可以进一步以类似的方式扩展以生成任何期望数目的标记的初始集合。

[0588] 通常，在至少一个实施例中生成标记的初始集合时，初始集合中的标记的特定数目可以是预定义的数目。作为当执行指令时触发的不同唯一规则的结果可以生成每一个特殊标记。比如以上代码段中的每个指令可能导致高速缓存未命中，且由此导致高速缓存未命中处理器的执行以计算作为用于特定指令的规则输出的一部分的Rtag，其中Rtag是初始集合的标记之一。以类似于以上代码段的指令的方式，可以在不同时间点执行不同代码序列以进一步使用初始集合的标记之一来生成其他标记。因此，初始集合中的每个标记可以表示用于进一步生成标记的另一序列的标记发生器。在前述实例中，ADD指令可以用于生成可以用于生成标记的另一整体序列的下一标记发生器。如以下讨论的，初始集合的标记发生器(其本身是用作生成另一序列的开始点的进一步标记发生器)可以不同于常规或者非生成标记，非生成标记不能进一步用作生成标记的另一序列的发生器。因此，比如ADD的特定指令可以用于触发规则和未命中处理以生成标记发生器的集合或者序列。这可以与比如MOVE的另一指令形成对照，其可以触发规则和未命中处理以生成序列中的非生成标记。关于比如malloc的代码，ADD指令可以类似地用来生成新应用标记颜色发生器，该新应用标记颜色发生器用于生成用于第一应用的不同颜色的序列(例如，新应用标记颜色发生器可以是用于生成用于特定应用的不同颜色的序列RED-APP1、BLUE-APP1、GREEN-APP1等的APP1)。标记的ADD指令然后可以用于获得特定应用指定序列中的下一标记，比如RED-APP1-gen、BLUE-APP1-gen或者GREEN-APP1-gen之一。然后标记的MOVE指令可以用于分别从RED-APP1-gen、BLUE-APP1-gen或者GREEN-APP1-gen生成实际颜色RED-APP1、BLUE-APP1或者GREAN-APP1(其中RED-APP1、BLUE-APP1、GREAN-APP1不能用于进一步生成附加标记序列)。

[0589] 当启用PUMP时执行的自举程序的代码段也可以包括附加代码，该附加代码当执行时触发规则而以任何期望特殊指令标记来标记内核程序代码/指令和另外标记其他代码模块或者实体，以使这种特殊标记的代码能够具有期望特权或者性能。例如，代码段可以包括触发规则而以扩展特权或者授权到这种代码以执行特权标记操作的特殊指令标记来标记加载器代码，例程malloc和free的代码的指令。特殊代码标记可以使用触发规则以生成进一步期望的标记且还以生成的标记适当地标记附加代码和/或数据的预定代码序列/指令集合，以类似于如上所述的方式从标记的初始集合生成。

[0590] 在至少一个实施例中，可以关于以上提到的代码段的部分采用附加的措施或者技术。例如，用于生成标记的初始集合的以上提到的4个指令可以使用“粘连”策略的规则包括在第一指令序列中，以推行比如在这里其他地方(例如，实例1400)描述的连续性和原子性。

[0591] 在已经执行以上提到的代码段以生成标记的初始集合和进一步特殊地标记内核程序代码和任何其他期望指令之后，控制可以转移到附加引导代码。在基于RISC-V架构的至少一个实施例中，附加引导代码可以在系统管理程序特权级别执行。这种附加引导代码例如可以包括触发加载规则的初始集合到PUMP中的指令。一旦已经完成引导，则如由标记模式CSR表示的PUMP标记模式可以设置为适当的级别，以关于比如在用户特权级别执行的

用户代码来启用PUMP(例如,设置如在实例910的911c中的标记模式以表示启用PUMP和以U(用户)模式或者仅特权级别的操作)。

[0592] 参考图72,示出了可以在根据在这里的技术的实施例中执行的处理步骤的流程图。流程图1600概述上述处理。在步骤1602,标记模式设置为关,其中标记模式CSR表示如在这里其他地方关于实例910的911a描述的PUMP关闭状态。在步骤1604,boottag CSR初始化为特殊自举标记。在步骤1606,开始自举程序的执行。在步骤1608,自举程序可以将defaulttag CSR设置为默认标记。在步骤1610,标记模式CSR可以被修改为把默认标记写到所有结果上的模式(例如,当在该标记模式时每个Rtag=默认标记)。在步骤1612,可以执行指令,触发规则以初始化存储器位置和以默认标记来标记存储器位置。在步骤1614,标记模式CSR可以改变为在步骤1616中的后续代码段的执行期间启用PUMP的模式。在步骤1616,以启用的PUMP执行后续代码段。该代码段包括触发规则以生成标记的初始集合,清除boottag CSR,标记内核程序代码和以特殊代码标记来标记附加代码部分的指令,该特殊代码标记依照要求向这种标记的代码提供扩展的能力、授权和特权。在步骤1618,控制可以转移到执行的附加引导代码。当完成引导处理时,系统现在已经准备好以启用的PUMP和用于执行用户代码的操作来执行用户代码。

[0593] 现在将更加详细地描述如何从自举标记生成标记。以自举标记开始的标记生成处理也可以被称为标记树或者生命树。更一般地,标记生成进程形成如图73的实例1620中所示的分级结构。

[0594] 实例1620图示boottag 1621作为标记生成进程的根。元素1621a-d可以表示比如如上所述生成的标记的初始集合。在该实例中,标记1621a-d的初始集合可以包括用于进一步生成无限制数目的特殊指令标记的序列1622的初始OS特定殊指令标记1621a,该序列1622然后可以将1623应用于不同代码部分或者模块1624的标记指令。从初始OS特殊指令标记1621a,可以对于要标记的不同模块生成附加标记1622。例如,可以对于将1623a应用于malloc代码的malloc来生成第一OS特殊指令标记1 1622a,由此以特殊指令标记1 1622a来标记1624a malloc的指令。以该方式,可以以标识malloc为标记发生器的特殊指令标记来标记malloc代码(例如,表示malloc代码具有进一步生成其他新标记和进一步使用新生成的标记来标记其他存储器单元的特权)。

[0595] 在关于malloc的该实例中,1621b可以是用于进一步生成malloc标记发生器应用标记1626的初始malloc标记,每个用户应用生成一个,因为malloc的实例包括在每个用户应用中。我们想要给予每个用户应用中的每个这种malloc实例以生成如包括在1625中的不同着色标记的特权。

[0596] 通常,实例1620图示用于特殊指令标记1621a、Malloc 1621b、CFI 1621c和Taint(污点)162Id的标记1621a-d的初始集合。因此,第一行中的标记的垂直显示中的标记1621a-d中的每一个(除了boottag 1621之外)表示用于生成无限制的标记序列的不同初始标记。例如,该值1621a用于进一步导出或者生成无限制数目的特殊指令标记1622。该值1621b用于进一步导出或者生成无限制数目的值1626。1626的每个实例可以进一步用作用于每个应用的标记的另一无限制的序列的发生器。例如,1626a表示用于进一步生成用于单个应用app1的不同颜色的另一无限制的序列1629的发生器值。以类似的方式,1626的每个不同发生器值可以用于进一步生成用于每个应用的无限制数目的颜色。

[0597] 值1621c可以用作进一步生成无限制数目的值1627时的发生器。元素1627类似于1626之处在于,对于特定应用或者app N的CFI标记发生器n的每个发生表示进一步生成另一无限制的序列的特权或者能力。例如,1627a表示用于进一步产生用于单个应用app1的不同颜色的另一无限制的序列1630的发生器值。以类似的方式,1627的每个不同发生器值可以用于进一步生成用于每个应用的无限制数目的颜色。

[0598] 值1621d可以用作进一步生成无限制数目的值1628时的发生器。元素1628类似于1626和1627之处在于,对于特定应用或者app N的CFI标记发生器n的每个发生表示进一步生成另一无限制的序列的特权或者能力。例如,1628a表示用于进一步产生用于单个应用app1的不同颜色的另一无限制的序列1631的发生器值。以类似的方式,1628的每个不同发生器值可以用于进一步生成用于每个应用的无限制数目的颜色。

[0599] 如图所示,分别来源于1621c-d的CFI和Taint的序列或者子树类似于来源于1621b的Malloc子树。在实例1620中,nxtTag或者TInxtTag用于表示生成的无限制序列中的下一元素,且getTag用于从序列成员提取下一标记。通常,getTag可以用于表示提取标记以使用,其本身不是标记发生器。如果可用的标记将要被给予特定代码部分使用,我们不想要也给予该代码部分生成标记的能力。例如,想要给予每个应用用于该应用的Malloc标记发生器(例如App1ColorTagX),但是不想给予应用生成用于其他应用的Malloc标记发生器的能力。所以,getTag将类型从发生器改变为实例。nxtTag和TInxtTag之间的区别在于nxtTag在没有“标记的指令”的情况下可用,而TInxtTag是仅由适当地标记的指令可用的。

[0600] Malloc应用标记序列1626允许操作系统或者加载器生成用于每个应用的颜色标记发生器。例如,元素1626a表示用于生成应用颜色序列1629的标记的应用专用颜色标记发生器值。在应用内,AppYColorTag序列1629允许malloc生成每个颜色的授权。该颜色授权可以用于:着色分配的存储器的单元,着色用于分配的指针和该颜色的空闲单元(例如,当起用free时)。比如以malloc和free的颜色的使用在这里其他地方描述了。

[0601] 以该方式,不同标记可以保留用于不同用途。从如上所述的最初标记的内核程序指令,可以执行内核程序代码,进一步以不同能力或者授权来标记其他代码部分。例如,操作系统的内核程序代码可以以比如许可加载器进一步标记其他代码和数据、生成附加标记发生器等的能力的特定特权来进一步标记其他代码实体,比如加载器。当加载包括malloc的用户程序时,加载器可以进一步以一个或多个特定指令标记来标记malloc代码,表示其为给予其进一步生成用于着色不同存储器区域的其他标记的能力的malloc码。置于加载器的代码上的特定指令标记因此向加载器提供特权的一个集合。malloc码上的第二不同指令标记的放置向malloc代码提供另一不同特权集合。通常,当执行序列的标记生成时,序列中的当前标记保存为关于生成序列中的下一标记参考和使用的状态信息。如在这里描述的,这种关于序列中的当前标记的状态信息可以保存和用于元数据处理域中。可以作为规则处理和高速缓存未命中处理的结果来保存和恢复当前标记,或更一般地元数据处理状态信息。序列中的当前标记,比如分配用于特定应用的最后颜色可以保存为序列的当前状态,作为关于指定存储器位置的标记。当需要分配应用的新的下一颜色时,malloc的的代码可以触发提取应用的最后分配的颜色和使用最后分配的颜色来确定应用专用颜色序列中的下一颜色的规则。通常,生成标记的唯一序列可以包括执行触发规则的指令,其执行以下:

[0602] 1.在原子单元的标记部分中(例如,寄存器,存储器位置)中存储/保存序列状态;

[0603] 2.执行触发使用保存/存储的序列状态来生成序列的下一标记的规则的指令;和

[0604] 3.在原子单元的标记部分中存储/保存序列的下一标记(从2生成),其中下一标记现在是更新的序列的当前状态。

[0605] 回头参考实例1620,加载器可以使用malloc对于每个应用分配1626的malloc标记发生器应用标记中的特定的一个。加载器例如可以执行触发规则的代码,该规则生成比如1626a的下一malloc标记发生器标记,且然后经由标记存储器位置将该标记存储作为状态信息。随后,关于由应用对malloc的首次调用,可以执行malloc的代码,触发然后提取保存的malloc标记发生器标记,使用保存的标记生成应用的第一颜色,且然后更新保存的状态信息以将第一颜色存储作为对于应用生成的最后或者最近颜色的规则。对于由应用对malloc的第二调用,可执行malloc的代码,触发然后提取先前保存的第一颜色,使用保存的第一颜色来产生应用的第二颜色,且然后更新保存的状态信息以现在将第二颜色存储作为对于应用生成的最后或者最近颜色的规则。以类似的方式,对malloc的其他后续调用可以触发其他规则,以基于用于应用的保存的状态信息(例如,最近分配的颜色)分配附加颜色。

[0606] 现在将描述的是可以在根据在这里的技术的实施例中包括的直接存储器访问(DMA)架构的方面。通常,在以下段落中描述的是利用I/O PUMP以调解(mediate)从源发布的DMA,,以访问使用标记数据的第二互连结构的存储器中存储的数据,该源比如连接到使用未标记数据的第一互连结构的不可信装置。

[0607] 参考图74,示出了可以在根据在这里的技术的实施例中包括的组件的实例。实例1500包括与实例700的组件和在这里其他地方描述的其他组件(例如,图57-图60)类似地编号的组件。另外,实例1500还包括I/O PUMP 1502和附加动作者,可以发出DMA请求以访问存储器712c中存储的数据的DMA请求源或者启动器1504a-c。实例1500包括连接到未标记结构715的以太网DMA装置A 1504a、以太网DMA装置B 1504b和UART(通用异步接收器/发射器)或者串行通信装置1504c。读取或者写入数据的DMA请求可以来源于装置1504a-c之一。该请求发送到I/O PUMP 1502,其执行处理以确定是否允许DMA请求,且如果是,则允许请求进行。因此,I/O PUMP 1502可以特性化为调解从未标记结构715之上接收的DMA请求,由此一般假定是连接到发出这种DMA请求的715的装置可能是不可信的。

[0608] 在至少一个实施例中,I/O PUMP 1502可以是如在这里描述的PUMP的示例(例如,图22),差异是推行的规则是控制到存储器1712c中的DMA访问的DMA策略的规则。I/O PUMP 1502的前述使用符合保证包括存储器操作的所有指令由规则调解的通用架构。如果允许自治的DMA装置1504a-c直接、无调解地访问存储器,则DMA装置1504a-c可以破坏推行规则的不变性和安全特性。因此,为允许DMA,根据在这里的技术的实施例也可以对到存储器712c中的DMA访问推行规则。类似于推行用于处理器指令的规则的PUMP,I/O PUMP 1502推行用于从DMA装置,比如1504a-c的存储器加载和存储的规则。通常,I/O PUMP调解所有加载和存储。在RISC-V架构的在这里描述的至少一个实施例中,I/O PUMP以类似于如在这里其他地方关于RISC-V架构中使用的PUMP描述的方式使用CSR并执行规则高速缓存未命中处理。I/O PUMP 1502具有类似于PUMP的CSR的集合,但是经由存储器映射的地址来访问它们。对比如关于实例1520在以下段落中描述的I/O PUMP CSR的访问也可以是使用规则保护的标记。当尝试定位I/O PUMP中的规则时遇到的规则高速缓存未命中触发要由处理器,RISC-V CPU 702服务的中断。I/O PUMP使用与处理器702相同的规则解析过程,但是存在单个DMA策略,

其仅包括用于DMA加载和存储以访问存储器712c中的数据的规则。I/O PUMP原子地写入存储器712c中(例如,作为单个原子操作写入标记和值)。但是,在一些实施例中,从读取Mtag到写入Mtag的完整过程(例如,执行标记校验或者验证和写入的处理)可以不是原子地以标准存储。

[0609] I/O PUMP 1502是用于SDMP的规则高速缓存。I/O PUMP提供DMA操作中涉及的标记的集合和操作结果之间的映射。在至少一个实施例中I/O PUMP独立于处理器702运行。因为I/O PUMP 1502是高速缓存,当其之前从未看到输入的集合(强制的)时或者当其不能保持到规则上(容量,或者可能冲突)时其将取得未命中。这以类似于如在这里对于PUMP描述的规则高速缓存未命中的方式导致相对于I/O PUMP的规则高速缓存未命中。相对于I/O PUMP规则高速缓存1502的未命中产生一个中断,该中断由规则高速缓存未命中处理器系统以软件处理,其与服务于处理机702未命中陷阱的是同一个。在相对于I/O PUMP 1502的规则未命中时,输入经由以下描述的I/O PUMP CSR(例如,实例1520)传递到未命中处理器(比如在元数据处理域中的处理器702的代码上执行),且通过CSR提供规则插入回到I/O PUMP。I/O PUMP未命中导致禁用I/O PUMP直到由处理器702服务为止。在至少一个实施例中,I/O PUMP的禁用的状态意味着由I/O PUMP调解的所有DMA转移被停止,直到I/O PUMP未命中被服务为止。

[0610] 按照以PUMP在这里其他地方的讨论,I/O PUMP输入包括操作组(opgrp),用于DMA指令及其操作数的标记(例如,PCtag、CI标记、OP1标记、OP2标记、Mtag(在这里有时也称为MRtag))。I/O PUMP输出可以包括如在这里描述的Rtag和PCnew标记(用于下一指令的PC的标记)。关于I/O PUMP,这种输入和输出可以具有如以下在一个实施例中描述的进一步的含义和值。

[0611] 之后是在一个实施例中的I/O PUMP输入:

[0612] 1.Opgrp-存在当前两个:加载和存储

[0613] 2.PCtag-DMA IO装置的状态(类似于用于代码的PCtag)

[0614] 3.CItag-标识DMA IO装置的标记(类似于关于代码的指定区域的指令标记)

[0615] 4.OP1tag-假定总是“公开的,不可信的”(不在IO PUMP高速缓存中物理地表示,而是用于规则)

[0616] 5.OP2tag-与OP1tag相同

[0617] 6.Mtag-关于对DMA操作的存储器输入的标记

[0618] 7.byteenable-正在读取/写入哪些字节

[0619] 之后是在一个实施例中的I/O PUMP输出:

[0620] 8.Rtag-用于存储的存储器结果上的标记

[0621] 10.PCnew标记-在此操作之后的DMA I/O装置的状态

[0622] 采用I/O PUMP,可能没有可编程操作组映射表(例如,实例420)。而是,由I/O PUMP使用以查找规则的操作组可以是表示用于DMA加载和DMA存储操作的单个操作组的固定操作码。在至少一个实施例中,没有用于I/O PUMP的关心掩蔽。

[0623] 当存在关于如在这里比如在图22中描述的PUMP的规则高速缓存未命中时,可以期望处理器702将在其相应的规则已经插入到PUMP规则高速缓存中之后自动地重新发出导致未命中的指令。结果,规则插入简单地将规则置于PUMP高速缓存中,并期望重新发出指令以

得到标记的结果。但是，以DMA操作的行为与前述不同。不期望DMA操作被中断且需要重试操作。为了支持这些DMA操作，可以对于I/O PUMP不同地处理规则插入。具体来说，一旦I/O PUMP已经由于未命中而故障，则处理可以保持未决的DMA操作并等待处理器702(例如，执行规则未命中处理以计算输出用于新规则的标记Rtag和PC新标记)提供用于规则的未命中输出标记(假定这将被允许)。当提供输出时，除触发到IO PUMP中的规则写入之外，这些输出还被转发到DMA流水线(例如，以下关于实例1540描述的)，就好像它们来自I/O PUMP那样，所以操作可以继续而不强迫操作被重新发出到I/O PUMP。可以通过提供用于更新的PCtag标记，即PCnew标记，的指定的禁止-DMA-装置标记来处理规则违规，其将表示不允许该操作且不允许来自特定DMA装置1504a-c的进一步的DMA操作，直到其PCtag复位为止。通常，发出DMA操作或者请求的特定DMA装置，比如1504a-c之一的装置标记可以是唯一地标识发出DMA装置(例如，DMA请求的源)的CI的特定值，和表示DMA装置的当前状态的PC标记。在至少一个实施例中，PC标记可以在一时间点设置为特定值，禁止来自由CI标记标识的特定DMA装置的DMA请求的进一步处理。

[0624] 参考图75，示出了在根据在这里的技术的实施例中可以由I/O PUMP使用的CSR的表。表1520包括地址列1524(表示CSR的存储器映射的地址)、名称列1526和描述列1528。表1520的每一行对应于由I/O PUMP使用的定义的CSR之一。行1522a指示CSR事务处理(transaction)ID具有地址0x00。到事务处理ID CSR的写入递增存储的当前事务处理ID(例如，用于预取)，且从事务处理ID CSR的读取返回事务处理ID CSR中存储的当前事务处理ID。行1522b指示CSR opgrp具有地址0x01。opgrp CSR包括当前DMA指令的操作组，且在规则未命中时作为到规则未命中处理器的输入使用。行1522c指示CSR byteenable具有地址0x02。byteenable CSR指示DMA操作作用于在字中的哪个字节，并且和在规则未命中时作为到规则未命中处理器的输入使用。按照在这里的其他讨论，这允许策略提供字节级别保护；触发的规则可以校验以保证允许DMA请求数据的字节由发出请求的特定DMA装置访问，比如通过特殊地标记对不同DMA装置可访问的存储器部分而。行1522d指示CSR pctag具有地址0x03。pctag CSR包括当前DMA指令的PC标记，且在规则未命中时作为到规则未命中处理器的输入使用。行1522e指示CSR citag具有地址0x04。citag CSR包括当前DMA指令的CI标记，且在规则未命中时作为到规则未命中处理器的输入使用。行1522f指示CSR mtag具有地址0x07。mtag CSR包括当前DMA指令的M标记，且在规则未命中时作为到规则未命中处理器的输入使用。行1522g指示CSR newpctag具有地址0x08。newpctag CSR包括在当前DMA指令的完成之后置于PC上的PC新标记(例如PUMP和高速缓存未命中处理的输出)。行1522h指示CSR rtag具有地址0x09。rtag CSR包括置于当前DMA指令的存储器结果上的标记(例如PUMP和高速缓存未命中处理的输出)。行1522i指示CSR确认具有地址0x0A。到确认CSR的写入导致写入到确认CSR的值与当前事务处理(ID)(如存储在事务处理ID CSR中)之间的比较。如果前述两个匹配，则该匹配触发规则到I/O PUMP的写入。规则写入包括用于当前DMA指令的操作码和标记输入和输出(如由未命中处理确定的)。行1522j指示CSR状态具有地址0x0E。状态CSR包括表示I/O PUMP的状态的值。例如在如在这里描述的一个实施例中，状态CSR可以表示使能或者禁止I/O PUMP。可以在如在这里其他地方描述的PUMP I/O规则高速缓存未命中的情况下禁止。行1522k指示CSR清空具有地址0x0F。清空CSR当写入到时触发I/O PUMP的清空(例如，清空或者清除来自I/O PUMP高速缓存的规则)。

[0625] 在至少一个实施例中，如果状态CSR的位0是1，则意味着禁止I/O PUMP，且或者如果位0具有值0，则意味着禁止I/O PUMP。PUMP I/O未命中禁止PUMP。状态CSR的位1指示PUMP是否故障和等待服务(例如，位1＝1意味着I/O PUMP故障/高速缓存未命中和等待服务)。状态CSR的位2指示当前是否正在由规则高速缓存未命中处理器解决I/O PUMP规则未命中，且如果事务处理ID匹配，则将插入的结果直接提供给未决的未命中操作。状态CSR的所有前述位由确认操作(成功的或者不成功的)复位(例如，位0＝使能，位1＝无故障，位2＝无未决未命中)。也可以执行到状态CSR的写入以复位前述位，例如，像起动时需要的那样初始地使能I/O PUMP。用于不成功的写入的状态CSR的复位允许DMA装置重试操作和重新触发故障。

[0626] 处理器702对I/O PUMP CSR的加载/存储存储器操作应该以iopump CI标记来标记。策略规则应该就位以限制对具有iopump CI标记的指令的操作。单独的I/O PUMP CSR不具有标记。

[0627] 未标记或者不可信结构715上的每个装置1504a-c可以配置有它自己的标记，该标记当处理器执行对装置的加载或者存储时呈现为装置标记(例如，参见1534b，其中装置标记存储在以下描述的装置寄存器文件中，且当特定装置执行DMA加载或者存储时指定为CI标记)。这允许细粒度的控制，经由该细粒度的控制，代码和授权可以直接存取哪些装置。对于对装置的所有加载和存储呈现相同标记，且标记不基于加载和存储操作而改变。与每个装置1504a-c相关联并标识每个装置1504a-c的特定装置标记可以存储在装置寄存器文件中。指定用于装置1504a-c的特定装置标记只可以通过修改装置寄存器文件而改变。装置寄存器文件可以对于每个装置1504a-c表示唯一目标装置ID(用于标识在未标记或者不可信结构715上的装置)和用于唯一目标装置ID的目标装置专用标记。在至少一个实施例中，装置寄存器文件本身可以作为关于具有它自己的装置标记的不可信结构715上的装置被访问。为自举装置寄存器文件的使用，装置标识寄存器文件自己的标记(存储在装置寄存器文件中)可以在使能PUMP之前在起动期间写入到该文件。例如，装置标识寄存器文件自己的标记可以在PUMP关闭时作为引导处理的一部分写入到文件(例如，由实例910的911a表示的标记模式)。指令的CI标记可以标识执行加载或者存储指令的DMA目标装置的目标ID，其中CI标记可以用于由这种加载和存储操作触发的规则以限制(例如，允许或不允许)指定DMA装置的特定加载或者存储操作。另外，如果特定DMA装置执行不允许的加载和/或存储操作，则可以修改与该特定DMA装置相关联的状态为禁止，以使得忽略进一步的请求(例如，DMA加载和存储)。

[0628] 如上所述，发出或者是DMA请求或者指令的源的DMA装置可以具有由DMA装置的PCtag指示的关联状态。具体来说，唯一PCtag可以用于表示对于从(由CI标记标识的)DMA装置允许的DMA操作的禁止状态。被禁止的发出者(initiators)使得它们的DMA请求在以下描述的DMA或者Trustbridge流水线的开始时(例如，实例1530和1540)被拒绝。

[0629] 应当注意，实施例可以具有调解所有DMA业务的单个I/O PUMP，每个DMA引擎的I/O PUMP，或者对于多个DMA引擎调解DMA业务的多个I/O PUMP。在实例1510中图示用于单个DMA引擎(例如，单个存储器712c)的单个I/O PUMP。如在实例1500中的单个I/O PUMP的使用可以成为瓶颈，且因此实施例可以选择以多个I/O PUMP调解I/O业务。在这种存在多个I/O PUMP的实施例中，每个可以独立地使能或者禁止，以使得即使可能禁止多个I/O PUMP中的一个或多个的第一部分(由于I/O PUMP未命中)，剩余的多个I/O PUMP的第二部分可以被使

能和继续服务于DMA请求。

[0630] 在至少一个实施例中，可以允许作为DMA操作的发出者或者源的不同DMA装置中的每个仅访问存储器712c的指定部分。可经由DMA访问的存储器712c的不同部分的每个可以以不同标记来标记。例如，装置1504a可以访问存储器712c的地址的第一范围，且装置1504b可以访问存储器712c的地址的不同的第二范围。对应第一范围的712c的存储器位置可以以第一标记来标记，且对应第二范围的712c的存储器位置可以以第二标记来标记。以该方式，规则可以用于推行或者限制装置1504a对第一范围中的存储器位置的访问，和推行或者限制装置1504b对第二范围中的存储器位置的访问。作为变型，不同标记可以与允许的访问的类型相关联(例如，只读，只写，读写)。以类似的方式，在具有访问相同存储器712c的多个DMA引擎的实施例中，可对每一个DMA引擎可排他地访问的单个存储器712c的不同部分可以被唯一地标记，由此规则推行或者限制每个DMA引擎对其存储器位置的指定地址范围的访问。

[0631] 参考图76，示出了图示在根据在这里的技术的实施例中可信结构1532(例如，对应于标记的互连结构710)和不可信结构1536(例如，对应于未标记的互连结构715)之间的数据流的实例。元素1534通常表示关于1532和1536之间的DMA调解由I/O PUMP 1534a执行的处理。元素1534可以表示作为DMA调解的一部分执行以验证和服务DMA操作的可信桥接器(trust bridge)或者DMA流水线1534c。元素1538a可以表示来自不可信结构1536的输出信道(例如，比如到实例1500中的DMA装置1504a-c)。元素1538b可以表示到不可信结构1536的输入信道(例如，从装置1504a-c之一)。通常，I/O PUMP 1534a将需要在DMA读和写操作期间发出读取请求以验证在目标存储器上的标记允许请求的DMA访问。I/O PUMP将需要缓存请求(如以下在处理级之间在实例1540中描述的)和执行标记的通信操作的主控制。

[0632] 元素1537表示作为输入提供以加载(或者提取)如在实例1520中描述的I/O PUMP CSR的值。另外，用于不同DMA装置发出者的装置状态信息可以存储在不可信结构装置寄存器文件1534b中，其包括用于DMA装置发出者(例如，比如不可信结构715上的1504a-c)的PCtag(例如，比如是否禁止来自该DMA装置的请求的DMA装置的状态)和CItag(例如，DMA装置唯一标识符)。用于执行DMA加载或者存储的特定DMA装置的装置寄存器文件1534b中的项可以提供用于当前DMA加载或者存储的CI标记和PCtag值。元素1535a可以表示用于不可信结构1536上的装置的通道，以做出1534的调解的DMA处理请求。元素1535b可以表示用于将1534的调解的DMA请求的结果返回到不可信结构1536的通道。

[0633] 元素1531a-b表示用于将DMA请求从不可信结构1536(经由1534的DMA调解处理)转发到可信结构1532的通道。具体来说，通道1531a是用于将初始标记读取(未验证的)DMA请求转发到可信结构1532的通道，且通道1531b是用于转发具有更新的标记的数据的最终写入的第二通道。给出以下关于实例1540描述的DMA或者trustbridge流水线的进一步讨论，两个通道的使用可以变得更明显。元素1531c表示经由DMA调解处理1534从可信结构1531c到不可信结构的通道。

[0634] 在一个实施例中，元素1534可以表示如在图77的实例1540中所示的DMA处理流水线。实例1540表示用于服务如由来自不可信或者未标记结构(例如，实例1500中的1506和实例1530中的1536)的DMA装置1504a-c做出的DMA操作的4级处理流水线。元素1542、1544、1546和1548可以表示作为DMA请求的结果触发的规则。元素1545表示比如关于其他图(例

如,实例1500的1502)描述的I/O PUMP。元素1543表示DMW处理流水线的级。在第一级1541a,从不可信结构接收到DMA请求,且经由第二存储器取出级1541b中的规则1542做出未验证请求以从存储器712c获得所请求的DMA数据及其关联的标记。用于DMA请求数据从存储器取出的标记信息作为输入提供到第三验证级1541c,其中对于对应于当前DMA请求的规则在I/O PUMP高速缓存1545中执行查找。如果在I/O PUMP中未找到规则,则当在处理器702中执行规则未命中处理器以计算DMA请求的输出Rtag和PCnew标记或者要不然确定不允许当前DMA请求(由此触发故障或者陷阱)时,可以在级1541c中停止和禁止I/O PUMP处理。假定用于当前DMA请求的规则位于I/O PUMP中,则确定允许执行DMA请求。如果DMA请求是写入请求,则在级4 1541d中,DMA请求的写入数据与其标记信息一起被写回到存储器712c。对于DMA写入操作,一旦已经完成写入,则可以将响应1548a提供给不可信结构(且然后提供给发出DMA请求的DMA装置)。对于DMA读取操作,可以将响应1546a返回到不可信结构(且然后到发出DMA请求的DMA装置),其中响应包括在级2 1541b中取出的请求数据。

[0635] 元素1542可以表示当在级2 1541b中执行存储器取出时传递来自不可信结构的请求和传递关于对I/O PUMP 1545(在级3 1541c中)的I/O请求的信息(来自级1 1541a)的规则。元素1544可以表示从可信结构收集标记响应,格式化到I/O PUMP的实际规则输入,和将信息从级1541b传播到写回级1541d以与I/O PUMP的输出合并的规则。

[0636] 作为前述实施例的变型,回头参考实例1500。在至少一个实施例中,不是如上所述将规则存储在I/O PUMP高速缓存中,I/O PUMP可以实现为硬布线的I/O PUMP,其中可以使用比如布线以实现I/O PUMP加载和存储DMA规则的固定集合的逻辑门的专用硬件来实现规则。

[0637] 作为进一步的变型,I/O PUMP可以在又一实施例中替代地定义为如关于实例1500描述的可编程的高速缓存,差别在于作为规则高速缓存的I/O PUMP具有有限容量且以全部存储在I/O PUMP高速缓存中的规则的固定集合填充。在该后一实施例中,I/O PUMP可以以全部DMA规则的完整集合填充,以使得从不存在该I/O PUMP的规则高速缓存未命中。因此,从不需要服务I/O PUMP规则高速缓存未命中。

[0638] 现在将描述的是可以关于初始化、设置或者复位比如可以与存储器位置相关联的标记使用的技术。按照在这里其他地方的描述,关于这种技术使用的标记可以表示非指针标记(其中非指针标记是用于关联的存储器位置的实际标记值)或者指针标记(其中指针标记是包括一个或多个实际标记值的另一存储器位置的指针或者地址)。例如,可以关于复合标记使用与存储器位置相关联的指针标记,其中指针标识包括比如用于并行实现的多个复合策略的多个标记值的存储器中的地址。如在这里其他地方描述的,可以并行支持的示例复合策略包括在这里其他地方描述的存储器安全策略和控制流完整性(CFI)策略。

[0639] 例如,关于存储器安全和堆栈策略执行的处理可以包括将与存储器位置相关联的大量标记设置或者初始化为特定值。例如,当分配比如可以与特定颜色相关联的存储器的区域时,与区域中的存储器位置相关联的每个标记需要被初始化以具有特定颜色值。作为另一实例,当比如当释放存储器区域时收回存储器的区域时,释放或者未分配的区域的全部存储器位置可以初始化为表示作为释放或者未分配的存储器位置的特定标记值。

[0640] 当要标记的存储器区域的大小增加时,执行以初始化或者复位区域中的全部存储器位置的标记的处理可能消耗不可接受的时间量且变得特别地不可接受。因此,以下段落

中描述的是提供用于高效地初始化或者设置存储器位置的标记(例如,标记)的技术。在至少一个实施例中,例如可以关于分配存储器的区域或者释放存储器的区域来执行标记初始化或者设置。在这里描述的这种技术可扩展以用于大存储器区域。虽然以下关于存储器位置的标记说明了这种技术,更一般地,这种技术可以关于初始化、设置或者复位每个与数据项或者实体相关联的值使用。

[0641] 在至少一个实施例中,标记和存储器的区域的关联的存储器位置可以以分级结构或布置表示,其中分级的叶子表示存储器位置的标记。为了说明,以下讨论参考树作为分级结构。但是,通常,任何适当的分级结构可以用于表示与存储器位置的区域相关联的地址空间。

[0642] 在极端的情况下,在一个实施例中,树或者分级结构的叶子可以表示存储器中的单独的且保存标记。但是,如果整个的子树以相同标记值同质地标记,在这里的技术可以简单地在树中的那个特定节点和关联级别存储标记值而不进一步表示子树的任何下代节点。在该情况下,节点的标记值指定特定区域的多个存储器位置(例如,比如顺序或者连续存储器地址的范围)的标记值。以该方式,如果存在大的同质标记的区域,则可以在存储标记值时节省存储。在没有同质的标记值的最坏情况的场景中(例如,没有具有连续地址的两个存储器位置具有相同标记值),则树的叶子每个都表示单个存储器位置,比如区域中的单个字的标记值。

[0643] 提供这种比如如以下段落描述的树的分级结构,可以执行处理以提供简单地重写树中的一个节点而重新标记或者初始化2的幂(power-of-two)存储器区域。对于非2的幂区域,可以执行处理以将区域分区为2的幂区域的最小集合(例如,在最小集合中至多2*log2(区域大小)这种区域)。当需要特定字或者存储器位置的标记(例如,读取关联的存储器位置的标记)时,可以执行处理以使用树确定标记。在以下描述的至少一个实施例中,高速缓存存储器的分层可以用于树的不同级别。可以由与具有相对于期望存储器位置的高速缓存命中的树中的最高级别相关联的高速缓存来提供标记值(例如,执行对于期望存储器位置的地址的标记值的高速缓存查找)。关于执行以写入或者修改与存储器位置相关联的标记值的处理,处理可以包括执行单个写入以标明子树,或者多个写入(例如,2*log2(区域大小)log写入)。这种多个写入例如可以响应于修改或者设置在这种修改或者设置标记值之前同质地标记的第一存储器区域中包括的第一存储器位置的标记而执行。在该情况下,设置或者修改标记值使得第一存储器区域不再被同质地标记,且因此更新表示第一存储器区域的标记值的分层结构以进一步分解表示第一存储器区域的标记值的子树。

[0644] 参考图78,示出了在根据在这里的技术的实施例中可以用于表示与存储器的区域对应的地址空间的标记值的分层结构的实例。实例100为了说明的简单而图示树为用于表示包括8个存储器位置的存储器区域的分层结构。通常,在这里的技术可以用于使用具有任意数目的级别、在每个级别的任何适当的数目的节点、每个父节点的任何适当的数目的子节点等的任何适当的分层来表示任何地址空间或者存储器区域的标记值。

[0645] 实例100图示表示包含地具有地址0到7的8个存储器位置的标记值的二叉树。取决于如果有的话,8个存储器位置的哪些使用结构100的相同子树来同质地标记,在该实例中的树可以包括多达4个级别的节点。在该实例中,8个存储器位置的整个存储器区域可以重复地分区为2的幂的较小的存储器区域,其中每个这种较小的存储器区域的分区对应于树

中的不同级别的节点。例如，级别1包括与分区为每个由在级别2 106的节点(节点B1和B2)表示的两个较小的区域的整个地址空间0到7对应的节点A1。级别2 106包括与地址0-3相关联的节点B1和与地址4-7相关联的节点B2。

[0646] 在级别2 106的节点B1和B2中的每一个可以进一步分区为每个由在级别3 108的节点表示的两个较小的区域。节点B1及其关联地址范围0-3被分区为由节点C1和C2表示的两个区域，其中C1与地址范围0-1相关联且C2与地址范围2-3相关联。类似地，节点B2及其关联地址范围4-7被分区为由节点C3和C4表示的两个区域，其中C3与地址范围4-5相关联且C4与地址范围6-7相关联。

[0647] 在级别3 108中的节点C1-C4的每一个可以进一步分区为每个由在级别4 110的节点表示的两个较小的区域。在该实例中，在级别4的节点每个表示单个字或者存储器位置的标记值。节点C1及其关联地址范围0-1被分区为由节点D1和D2表示的两个区域，其中D1与地址0相关联且D2与地址1相关联。节点C2及其关联地址范围2-3被分区为由节点D3和D4表示的两个区域，其中D3与地址2相关联且D4与地址3相关联。节点C3及其关联地址范围4-5被分区为由节点D5和D6表示的两个区域，其中D5与地址4相关联且D6与地址6相关联。节点C4及其关联地址范围6-7被分区为由节点D7和D8表示的两个区域，其中D7与地址6相关联且D8与地址7相关联。

[0648] 所有节点A1、B1-B2、C1-C4和D1-D8可以表示在8个存储器位置的区域的标记值的分层表示中存在的最大数目的可能节点。但是，如以下更详细地描述的，表示存储器位置0-7中存储的标记值的树中包括的特定节点可以取决于特定标记值以及在各种时间点表示的同质和非同质标记区域而改变。分层的级别可以从对应于根或者级别1 104节点的最高级别到对应于最底级别4 110的最低级别分级。

[0649] 关于通过第一实例在这里的技术，参考图79的120。在该第一实例中，假定与在分层120的特定级别的节点相关联的所有存储器位置具有相同标记值T1，由此表示同质地标记的存储器位置的子树，在特定级别的节点具有所有这种存储器位置的标记值，且不需要查阅子树中的进一步下代节点以确定任何同质地标记的存储器位置的标记值。例如，如果比如关于以相同标记初始化存储器的区域，所有存储器位置0-7包括相同标记值T1，则存储器位置0-7的标记值T1可以在节点A1存储(例如，如由节点A1指示的“标记＝T1”表示的)。在至少一个实施例中，不需要进一步存储树的其他节点的附加标记值，因为地址0-7的整个区域的标记值由单个节点A1表示。在该情况下，元素122表示可以包括在在具有地址0-7的存储器位置存储的标记值的分层表示中的单个节点，和可以从分层表示省略剩余节点B1、B2、C1-C4和D1-D8。

[0650] 在第二实例中，参考图80的130。在该第二实例中，假定存储器位置0-3具有相同的第一标记值T1，且存储器位置4-7具有相同的第二标记值T2(第一和第二标记值不同)。在该情况下，节点A1可以包括表示节点A1不指定存储器位置0-7的同质标记且存储器位置0-7的标记值由在分层的一个或多个下级的节点指定的指示符(例如，由节点A1的“TAG VALUE(标记值)＝NO TAG VALUE(无标记值)”指示表示)。在分层的级别2，第一标记值T1可以在节点B1存储(如由节点B1的“标记值＝T1”指示表示的)，且第二标记值T2可以在节点B2存储(如由节点B2的″标记值＝T2″指示表示的)。B1是根的子树(B1、C1、C2、D1-D4)表示同质地标记的存储器位置0-3的集合。B2是根的子树(B2、C3、C4、D5-D8)表示同质地标记的存储器位置

4-7的另一集合。在至少一个实施例中，不需要进一步存储在级别3和4的树的其他节点(例如，级别3的节点C1-C4和级别4的节点D1-D8)的附加标记值，因为地址0-7的整个区域的标记值由在级别2的节点B1和B2表示。在该情况下，元素132表示可以包括在在具有地址0-7的存储器位置存储的标记值的分层表示中的单个节点，和可以从分层表示省略剩余节点B1、B2、C1-C4和D1-D8。

[0651] 在第一时间点，标记分层可以是如同用只有单个节点122的实例120来描述的，因为所有标记具有相同标记值。在后续第二时间点，地址0-3的标记值可以被修改为相同第一标记值T1，且地址4-7可以被修改为相同第二标记值T2。作为前述标记修改的结果，实例130中的如上所述的两个添加节点B1和B2可以添加到分层。假定现在在后续第三时间点，地址0-3的标记值保持与在实例130中相同。但是，可以如以下关于图81描述的修改地址4-7的标记值，由此附加节点C3-C4和D5-D6被添加到标记分层。

[0652] 在第三实例中，参考图81的140。在该第三实例中，假定存储器位置0-3具有如上所述的相同的第一标记值T1(其中第一标记值T1可以存储在节点B1，且B1是根的子树(B1、C1、C2、D1-D4)表示同质地标记的存储器位置0-3的集合)。进一步，假定存储器位置4-5每个包括不同标记值，其中存储器位置4具有标记值T3且存储器位置5具有标记值T4，且存储器位置6-7是同质地标记的且包括相同标记值T5。在该情况下，按照以上描述，节点A1、B2和C3每个可以包括特定节点不指定标记值由此在分层的一个或多个下级的节点指定与节点A1、B2和C3相关联的特定存储器位置的标记值的指示符(例如，标记＝无标记)。例如，对应于存储器位置4-5的节点C3可以包括节点C3不指定标记值由此在分层中的一个或多个下级的节点指定存储器位置4-5的标记值的指示符。在级别4的节点D5可以指定存储器位置4的标记值T3(例如，节点D5的标记＝T3指示符)，且在级别4的节点D6可以指定存储器位置5的标记值T5(例如，节点D6的标记＝T4指示符)对应于存储器位置6-7的节点C4可以指定标记值T5(例如，节点C4的标记＝T5指示符)，对存储器位置6-7共同的同质的标记值，且指示不需要进一步在节点D7和D8中存储附加标记值(例如，不需要进一步检查C4的下代节点D7、D8)。在该情况下，元素142表示可以包括在在具有地址0-7的存储器位置存储的标记值的分层表示中的节点，可以从分层表示省略剩余节点C1-C2、D1-D4和D7-D8。

[0653] 120、130和140的前述说明可以表示用于在不同时间点的地址或者存储器位置0-7的存储器区域的标记值的分层表示，因为与存储器位置相关联的标记值可以随时间改变。以类似于如上所述添加节点到分层的方式，当现有节点的子树被修改为全部具有相同标记值，可以按照需要从分层除去节点(例如，如果节点的后代全部具有相同标记值，则可以从分层除去全部下代节点，且该节点可以用作子树的唯一节点以表示该节点及其后代的单个同质标记值)。

[0654] 在至少一个实施例中，当在树中的级别的第一节点指定与第一节点相关联的一个或多个存储器位置的值时，不需要进一步表示第一节点的下代节点(例如，不需要进一步表示超出第一节点的子树的节点)。为进一步图示，参考图79的120中以上提到的第一实例，其中仅需要节点A1的标记值表示存储器区域0-7的单个同质标记值。为进一步说明，参考以上提到的图81的第三实例140，其中实施例可以不进一步表示C1-C2、D1-D4和D7-D8。以该方式，使用这种存储器位置和关联的标记值的分层表示可以关于存储器位置的标记值节省存储。换句话说，在至少一个实施例中，不是总是分配和存储每一个存储器位置的单独的标记

值，而是可以分配存储，其中分层中的单个标记值表示具有顺序或者连续地址的多个存储器位置的同质标记值。参考120中提到的第一实例，不是分配用于每个包括相同标记值的存储器位置0-7的8个标记值的存储，而是可以分配存储器用于存储节点A1的单个标记值。

[0655] 在最坏情况场景中，假定在具有地址0-7的存储器区域中没有同质地标记的存储器位置，则图78的节点的整个分层结构用于表示在地址0-7存储的标记值。例如，分层的每一个叶子可以表示存储器中的不同字。因此，分层的底部级别4 110可以表示地址空间0-7的标记值。

[0656] 回头参考图78，假定存在用于表示存储器位置0-7的地址的8位地址空间。在至少一个实施例中，整个8位地址空间可以分区为每个包括8个存储器位置的不同存储器区域，其中每个不同存储器区域可以具有由标记值分层的不同实例表示的标记值。因此，对于刚刚描述的地址0-7的存储器区域，最高或者顶部5位全部＝0且地址0-7可以以剩余的较低3位表示。最高或者顶部5位＝0因此可以用于指示地址0-7的存储器区域。在这种实施例中，8个存储器位置的每个存储器区域可以具有表示特定存储器区域的标记值的比如图78中图示的分开的标记值分层。在该实例中，表示8个地址或者存储器位置的不同范围的每个不同存储器区域可以通过检查存储器位置的8位地址的顶部5位来区分。

[0657] 在根据在这里的技术的至少一个实施例中，可以使用一系列标记高速缓存存储器，其中标记高速缓存的数目可以对应于表示标记值的节点的分层中的级别的数目。继续上面讨论的实例和回头参考图78的100，8个存储器位置的存储器区域的标记分层的每个实例具有4个级别。在这种情况下，实施例可以使用如图82的示例150中说明的4个标记高速缓存存储器152、154、156和158以存储存储器位置的标记。通常，4个标记高速缓存存储器152、154、156和158中的每一个与标记值分层中的不同级别相关联，且可以存储关于标记值分层的关联的不同级别中的每个节点的信息。例如，标记级别高速缓存152可以包括每个存储器区域(在如上所述的该特定实例中是8个存储器位置的存储器区域)的标记值分层的级别1 104节点或者根的信息。标记级别高速缓存154可以包括每个存储器区域的标记值分层的级别2 106节点的信息。标记级别高速缓存156可以包括每个存储器区域的标记值分层的级别3 108节点的信息。标记级别高速缓存158可以包括每个存储器区域的标记值分层的级别4 108节点的信息。作为在该实例中的级别4 110的分层中的最低或者最底级别可以对应于可以在数据高速缓存中存储的存储器位置的高速缓存行(line)(例如，表示为比如由图1的元素20表示的L1-D$)。实施例可以具有标记高速缓存的级别4 158且另外具有可以在数据高速缓存的高速缓存行中分开地存储的元数据标记。节点的每个高速缓存152、154、156和158具有主存储器中的关联表示。

[0658] 关于具有8位地址空间的在这里描述的示例实施例，存储器位置的地址的顶部或者最高5位152a可以由级别1高速缓存152使用以查找高速缓存152是否包括存储器位置的地址的任何级别1节点。存储器位置的地址的顶部或者最高6位154a可以由级别2高速缓存154使用以查找高速缓存154是否包括存储器位置的地址的任何级别2节点。存储器位置的地址的顶部或者最高7位156a可以由级别3高速缓存156使用以查找高速缓存156是否包括存储器位置的地址的任何级别3节点。存储器位置的地址的8位154a可以由级别4高速缓存158使用以查找高速缓存158是否包括存储器位置的地址的任何级别4节点。

[0659] 对于特定地址，与除了最低级别之外的级别相关联的每个高速缓存可以返回：

[0660] 1).特定地址的标记值(表示这是用于多个地址的在此级别的同质标记值)；

[0661] 2).高速缓存不指定特定地址的标记值且需要查询在分层中的较低级别的高速缓存以获得特定地址的标记值的指示符(该特定级别不指定用于特定地址的同质标记值)；或者

[0662] 3).空或者第二指示符，表示没有包括与特定地址对应的节点或者标记信息的此特定级别高速缓存中的高速缓存位置。第二指示符还表示在较低的非最低级别高速缓存，没有高速缓存包括该地址的节点或者标记信息。这在以下更详细地讨论。

[0663] 按照以上讨论，在以上的项2)返回的指示符可以是与比如在实例120、130和140中说明的节点相关联的“无标记”指示符。例如，参考图80的说明130，假定执行处理以确定存储器位置5的标记。在该情况下，级别1高速缓存152可以返回指示存储器位置5的标记值由其他较低级别高速缓存154、156或者158之一指定的无标记指示符。级别2高速缓存154可以返回说明以上返回的高速缓存项1)的存储器位置5的标记值T2。为说明其中返回第二指示符的以上返回的项3)，考虑级别3高速缓存156。级别3高速缓存156可以不包括与存储器位置5对应的任何节点信息(例如，没有高速缓存位置包括与存储器位置5查找相关联的节点或者标记信息)，且因此可以返回上述3)中的第二指示符，指示在该级别3高速缓存中没有存储器位置5的节点信息。在这种实施例中，处理通常可以利用从最高标记级别高速缓存返回的标记值。例如，关于参考实例130的存储器位置5，级别2高速缓存154是返回存储器位置5的标记值的标记高速缓存的最高级别。

[0664] 对于L1(级别1)数据高速缓存中存储的存储器位置的内容，高速缓存的信息可以包括当前标记值以及其中定义标记值的标记高速缓存分层中的级别。再次参考使用图80的分层130的以上存储器位置5的实例，如果存储器位置5的内容也存储在数据高速缓存中，则数据高速缓存可以包括标记值T2以及当前标记值T2由其节点信息存储在级别2高速缓存154中的级别2节点(例如B2)来定义的信息。因此，实例150说明其中可以存储标记值的标记高速缓存分层中的4个标记高速缓存，且实施例可以另外与标记高速缓存分层中存储的任何标记信息分开的标记的数据高速缓存(例如，L1数据高速缓存)。

[0665] 在根据在这里的技术的实施例中，处理可以由PUMP执行以解析或者确定具有特定地址的特定存储器位置的标记值。当执行处理以获得特定存储器位置的标记值和内容时，可能存在数据高速缓存命中，由此存储器位置内容及其标记存储在数据高速缓存中。在发生存储器位置的数据高速缓存命中时，可以执行处理以查询定义该存储器位置的标记值的标记高速缓存分层的存储的级别，以保证从标记高速缓存级别获得的第一高速缓存的标记值匹配于存储在数据高速缓存中的存储器位置的第二标记值。如果两个不匹配，这指示存储在数据高速缓存中的第二高速缓存的标记值是陈旧的、过时的和已经修改的。在该情况下，如果存储器位置的数据高速缓存中存储的第二高速缓存的标记值和从存储器位置的标记高速缓存获得的第一标记值不匹配，则可以执行包括更新存储器位置的数据高速缓存中存储的第二高速缓存的标记值(例如，以匹配标记高速缓存分层中存储的)的处理。在至少一个实施例中，对于其数据和其标记在数据高速缓存中高速缓存的存储器位置，可以对于包括其中定义标记的标记高速缓存分层的级别的存储器位置的标记在数据高速缓存中跟踪信息。高速缓存标记分层中的级别的前述存储可以是优化，由此存储的级别可以用于容易地从标记分层访问标记值(例如，不是必须查询全部标记级别高速缓存，或者以其他方式

搜索现有节点的分层,比如在从分层的根或者顶部向下向着叶子节点的搜索中)。因此,在发生数据高速缓存命中,且存储器位置的数据高速缓存中存储的标记值不匹配存储器位置的标记分层中存储的标记值时,处理可以包括关于在标记分层中哪里定义存储器位置的标记,来更新数据高速缓存中存储的标记值,并附加地更新数据高速缓存中存储的分层级别信息。随后,可以重新启动由PUMP执行的用于解析或者确定存储器位置的标记值的处理。

[0666] 在发生存储器位置的数据高速缓存未命中时(例如,在数据高速缓存中未找到存储器位置内容和标记时),可以执行处理以并行执行对标记高速缓存的各级别中标记值的标记高速缓存查找(例如,除了最底标记高速缓存级别之外)。例如,可以通过查询分别用于标记高速缓存的级别1、2、3和4的4个高速缓存152、154、156和158 158来并行地执行存储器位置的标记值的查找。如上所述,由标记高速缓存152、154、156和158的最高级别返回的标记值用作存储器位置的标记值。另外,应当注意在适当地表示的标记值分层中,仅152、154、156和158中的单独一个可以返回存储器位置的标记值。在至少一个实施例中,可以索引高速缓存152、154、156和158以允许并行访问。

[0667] 实施例也可以不执行相对于全部4个标记高速缓存152、154、156和158的特定存储器位置的标记的并行查找或者搜索。作为变型,实施例可以从根节点级别(级别1)向下向着叶子节点(例如,级别4)遍历分层的标记高速缓存。对于在级别N的标记高速缓存未命中,可以遍历树或者分层,插入节点到用于特定存储器访问的标记高速缓存的不同级别中。关于用于标记高速缓存的并行搜索的级别高速缓存未命中,实施例可以选择仅当存在标记时插入节点到级别高速缓存中。只要某个级别高速缓存提供标记,就不需要全部其他级别高速缓存具有无标记(NO TAG)项。

[0668] 如在这里其他地方讨论的,可以修改存储器位置的标记。响应于修改存储器位置的标记,可以执行处理以相应地更新指定存储器位置的标记值的分层。这种更新可以包括使不再同质的分层的任何一个或多个级别无效。另外,可以执行处理以相应地更新高速缓存的级别,比如在图82的实例150中说明的。

[0669] 当执行操作以设置或者初始化存储器位置的标记时,这种处理可以包括用于执行期望操作的有效性的PUMP校验。例如,考虑以新标记重新标记存储器区域中的全部存储器位置的情况。处理可以包括获得区域中的所有存储器位置的当前标记值和经由PUMP处理来检查重新标记的有效性的校验。该处理可以包括如果允许则去除区域中的存储器位置的标记,且然后如果允许则更新区域中的存储器位置的标记值。按照以上的讨论,更新、修改或者设置存储器区域的标记可以包括相应地修改分层和关联的节点以反映存储器区域中的不同存储器位置的标记值(例如,分解在修改之前同质和在修改之后非同质的区域的部分,由此可能有添加的附加子节点或者下代节点以反映一个或多个标记值修改)。

[0670] 在至少一个实施例中,存储器区域的标记值的分层表示可以是树。例如,树可以是其中每个节点具有0、1或者2个子节点的二叉树。作为变型,分层表示可以是树而不是二叉树。例如,可以允许树中的每个节点具有任何适当数目的子节点直到指定的最大值。分层表示可以包括任何适当数目的级别,在每个级别的节点,每个父节点的子节点等。如现有技术中已知的,存在分层表示的各种参数之间的折衷,比如深度或级别的数目和在每个节点的节点/每个父节点的子节点的数目。例如,在每个级别的节点的数目越大,级别的数目越少,且因此当确定存储器位置的标记值时要查询的时间/级别的量越短。但是,在这种情况下,

需要执行更多写入以清除区域。

[0671] 现在将描述的是关于在根据在这里的技术的实施例中的CFI策略，比如通过作为加载器代码的结果触发的规则可以执行的技术。为使用访问标记信息的元数据处理规则来推行CFI策略，关于可允许控制流的信息需要被传递到元数据处理域。为此，根据在这里的技术的实施例可以使用以下段落中描述的方法。通常，进行从分支源到目标或者目的地的控制转移。关于可允许控制流，对于特定控制流目标或者目的地，可以标识源的集合，其被允许转移控制到特定控制流目标或者目的地。每个可能的控制流目标的源的集合可以传递到元数据处理域，比如存储的元数据标记信息，其然后可以在用户代码的运行时间执行期间关于CFI策略推行由CFI策略的规则来使用(例如，在代码执行域或者非元数据处理域中的代码执行)。

[0672] 执行的处理可以包括唯一地标记每个源且然后以允许转移控制到此特定目标的可允许源的集合(例如，源的地址)来标记每个目标。例如，参考图83。在实例1700中，元素1701可以表示在代码执行或者非元数据处理域中执行的应用的指令的代码部分。元素1702a和1704a-c表示代码部分中的指令的位置。元素1702a表示控制流目标A。元素1704a-c表示允许转移控制到目标A 1702a的控制流源。这种从1704a-c中的每个的控制转移由JMP(跳转)A指令表示。元素1706表示允许转移控制到目标A的可允许源的集合。D7表示指令1704a的唯一源标记。C3表示指令1704b的唯一源标记。E8表示指令1704c的唯一源标记。如由1710所示，JMP(跳转)指令1704a-c分别标记为D7、C3和E8。如也由1710说明的，指令1704a-c也分别存储在地址1020、1028和1034。目标位置A具有地址800。在该情况下，允许转移控制到目标A的可允许源的集合，或者源指令的地址可以是由1706表示的集合{1020、1028、1034}。因此，集合1706是需要传递到元数据处理域的可允许控制流信息的实例，在元数据处理域中这种可允许控制流信息存储为用于由CFI策略的规则使用的标记元数据。在根据在这里的技术的至少一个实施例中，加载器的代码可以激发执行收集元数据处理域需要的控制流信息的处理以推行用于包括代码部分1701的应用的CFI策略的规则。可以关于加载应用(例如，加载应用的可执行代码)来执行加载器代码，由此加载器代码当执行以加载应用时，触发执行所需处理以收集控制流信息的规则(如随后由元数据处理使用以在应用的执行期间推行CFI策略)。

[0673] 在按照在这里其他地方的描述的至少一个实施例中，内核程序代码的执行可以触发以使能加载器的标记的指令的特殊指令标记来标记加载器的代码的规则，该加载器的标记的指令当执行时触发生成用于标记源(例如，生成源标记D7、C3和E8)的源标记的序列(序列的每个标记是唯一的)的规则。例如，参考包括由作为执行加载器的代码的结果激发的规则执行的逻辑处理的图84。使用类C伪代码描述在1720中描述逻辑处理，其中这种处理可以对于比如A 1702a的每个控制流目标执行。在步骤1721，源集合初始化为空集。在步骤1722，对于允许转移控制到目标的每个源，可以执行步骤1723、1724和1725。在步骤1723，向t分配新分配的CFI源标记。在步骤1724，以在步骤1723生成的新分配的标记t来标记(转移控制到目标的指令的)源位置。在步骤1725，更新源集合以也包括标记t。在一个方面中，步骤1725的操作可以特性化为形成用于目标的可允许源的集合的并集，其中对于如对于每个源执行的在1722开始的循环处理的每个迭代在1725中执行该并集操作。步骤1726以源集合来标记或者标明目标。

[0674] 元素1723可以是触发生成或者分配新CFI源标记的规则的加载器代码中包括的以下指令：

[0675] ADD R1←R1+R1

[0676] 其中ADD指令(例如,比如RISC-V指令集中的ADDI)已经以标明该指令为可允许标记发生器指令的CFI分配标记的特殊CI标记来由内核程序代码标记。在至少一个实施例中,源标记的不同序列可以关于CFI策略对于每个应用由加载器生成(例如,在实例1620中,加载器可以使用CFI标记1630的不同序列来作为用于应用的CFI源标记的唯一序列,其中CFI标记的序列可以从1627的CFI标记发生器App-n标记中的特定的一个生成)。CFI分配标记是置于以上的加载器ADD指令上的CI标记,表示允许ADD指令分配或者生成应用专用CFI序列中的下一标记。CFI分配标记可以是如关于实例1620描述的1624的特殊指令类型之一。以上的ADD指令指示R1上的标记保存CFI序列的状态,其中状态可以是先前生成的序列的最后标记。以上ADD指令的执行触发生成CFI序列中的下一新标记和更新R1上的标记以现在是新生成的标记的规则。使用如在这里其他地方描述的规则惯例,以下ADD规则可以表示由以上ADD指令触发的规则。

[0677] ADD:(-,CFI-alloc-tag,t1,t1,-)→(-,t1next)

[0678] 其保证ADDI指令的CI标记是CFI-alloc-tag(CFI分配标记)。在该ADD规则中,t1表示用于生成序列中的下一标记t1next的序列中的先前标记(保存为应用的CFI标记序列的当前状态),其中t1next然后存储为用于RD的标记(目的地或者结果寄存器)。CFI序列中的前述标记t1next可以用作置于源点上的唯一CFI源标记。

[0679] 元素1724可以是用于触发以唯一CFI源标记来标记源位置的规则的比如以下ST(存储)指令的加载器代码的指令：

[0680] ST R1→R3

[0681] 其中R3是到正被标记的用户程序代码中的控制流源位置的指针(例如,实例1700中的1704a),且R1上的标记是要置于源位置上的唯一CFI源标记。以上ST指令也可以以表示ST指令包括在触发以下规则ST的加载器代码中的比如CI-LDR的特殊CI标记来标记：

[0682] ST:(-,CI-LDR,t1,-,codetag)→(-,t1)

[0683] 其中CI标记＝CI-LDR,t1是当前存储为R1上的标记的CFI源标记,且codetag(代码标记)是在地址R3的源位置上的指令标记(例如,保证源位置当前标记为代码)。结果,以唯一CFI源标记t1来标记目的地(R3)。

[0684] 元素1725可以是用于触发规则的比如以下ADD指令的加载器代码的指令,该规则将源的地址(例如,当前由R3指向,其中R3包括源的地址)添加到表示可以转移控制到目标的可允许源位置的CFI源标记的累积集合：

[0685] ADD R2←R2+R3

[0686] 其中R2上的标记指向表示可允许源位置的累积集合的存储器位置。以上ADD指令可以以特殊CFI UNION(并集)指令标记来标记,表示该ADD指令执行CFI源的并集操作且该并集存储为R2上的标记。可以作为以上ADD指令的结果来激发用于ADD的以下规则：

[0687] ADD:(-,CFI UNION,tset,tsrc,-)→(-,tunion)

[0688] 其校验以保证CI标记是CFI UNION,tset是目标集合,且tsrc是源标记。它产生表示添加tsrc到tset的新CFI集合tunion。

[0689] 元素1726可以是用于触发规则的比如以下ST指令的加载器代码的指令，该规则以可以转移控制到目标的可允许源位置的并集或者累积列表来标记目标：

[0690] ST R2→R17

[0691] R17可以是包括目标位置的地址的寄存器，且R2如上所述可以是以可允许源位置的当前累积集合并集标记的寄存器(例如，R2上的标记表示其地址包括在R17中的目标位置的可允许源位置的集合)。以上ST指令可以以表示指令为允许标记控制转移目标位置的特殊的指令的特殊指令标记CFI MARK TARGET(CFI标明目标)标记(例如，加载器代码的STORE(存储)指令1726可以已经由内核程序代码以类似于关于触发执行1720的处理的规则的加载代码指令的其他代码标记的方式来标记)。作为用于1726的以上STORE指令的结果可以触发以下ST规则：

[0692] ST:(-,CFI MARK TARGET,tset,-,codetag)→(-,tset)

[0693] 其当CI标记是CFI MARK TARGET且目标(由R17指向，其中R17包括目标地址)以指示指令的codetag标记时触发，且将tset注释置于目标上。

[0694] 可以定义用于以源、目标和可允许源位置的集合使用的不同标记结构或者布局在这里其他地方描述，以及任何其他适当的结构定义(例如，参见描述用标记的源和目标位置使用的标记布局的实例240、250、260、267、270和280，其可以更一般地以任何指令以及关于每个标记的字的多个指令来使用)。

[0695] 因此，如在实例1720中的上述处理步骤可以通过适当地标记加载器的代码以使得当执行这种加载器代码时，激发规则使得由在根据在这里的技术的实施例中的元数据处理域执行实例1720的步骤来执行。应当注意，前述指令序列和作为指令的结果激发的规则仅是可以用于使用在这里的技术的实施例的指令和规则的一个实例。例如，实施例可以包括触发执行如上所述的处理的规则(例如，元素1725)的加载器代码中的ADD之外的不同指令。

[0696] 在前述描述中，为了简洁、清楚和了解而使用某些术语。不应超出现有技术的需要而暗示不必要的限制，因为这种术语用于描述性目的而意在被广泛地解释。此外，本公开的优选实施例的描述和说明是举例且本公开不限于示出或者描述的确切细节。

[0697] 在这里描述的技术的各种方面可以通过执行在任何一个或多个不同形式的计算机可读介质上存储的代码执行。计算机可读介质可以包括可以是可拆卸或者不可拆卸的不同形式的易失性(例如，RAM)和非易失性(例如，ROM、闪存存储器、磁盘或者光盘或者磁带)存储设备。

[0698] 虽然已经关于具体示出和描述的各种实施例公开了本发明，关于其的它们的修改和改进将对本领域技术人员容易地变得明显。因此，本发明的精神和保护范围应该仅由以下权利要求限制。

[0699] 编程PUMP

[0700] 用于安全性的硬件辅助的微策略

[0701] 摘要

[0702] 各种各样的安全性策略可以格式化为在ISA级别关于元数据的规则且以可编程硬件高效地推行。我们精心制成基于用于元数据处理(PUMP)架构的可编程单元的用于这种策略的编程模型，且支持关于与主计算一起的未解释元数据的灵活的规则评估。我们通过在四个特定域-空间和时间存储器安全、污点跟踪、控制流完整性和原语键入中实现不同复杂

性的安全和安全性策略的不同集合来说明模型的通用性。单独地和组合地,我们对于简单的RISC ISA特性化这些策略的性能。大多数策略的平均运行时间开销仅是8%。这示出PUMP模型可以实现具有专用硬件的性能的软件推行的灵活性和可适配性。

[0703] 1.引言

[0704] 对于攻击者太容易破坏程序的意图。设计用于对它们执行的操作的想要的高级语义不可知的现代处理器在该形势中是复杂的-当晶体管昂贵时的技术时代的传统和主要设计目标是运行时间性能。通过越来越委以关键任务的计算机系统,系统安全性最终成为关键设计目标。同时,与甚至普通的片上系统晶片相比处理器现在很小,使得以安全性增强硬件来扩充它们是可行的和便宜的。对于充分地保护它们管理的数据的隐私和完整性的明天的计算机,我们必须以按照现代威胁和硬件成本的安全性机制来重新架构整个计算堆栈。

[0705] 安全性文献提供可以减小由于恶意和错误代码的脆弱性的大范围的运行时间策略。这些策略通常将高级语言抽象(这是数值数组,这是代码指针,...)或者用户级别安全性不变性(该字符串来自网络)编码到关于程序的数据和代码的元数据注释中。通过当计算进行时传播该元数据和在适当的点动态地检查违规来推行高级语义或者策略。我们称这些低级细粒度的推行机制为微策略(或者非正式地仅“策略”)。

[0706] 微策略的软件实现可以定义任意元数据和在其上的任意的强大的计算。软件实现促进新策略的快速部署,但是就运行时间和能量成本而言它可能是不可行的昂贵(1.5x-10x)[42],导致不利的安全性性能折衷。可以以具有低开销[41,?]的硬件支持简单微策略;但是,定制为支持单个策略的硬件可能花数年来部署且很慢地采用。现今的动态网络攻击情况要求支持对演进的威胁的快速现场响应的机制。

[0707] 更大灵活性的希望促进了很多近来的努力使得策略推行硬件更可编程[18,45,19,13](参见§5)。这里,我们考虑称为PUMP[7]的设计,“Programmable Unit for Metadata Processing”,其允许就关于任意元数据的指令粒度的计算而言定义各种各样的低级运行时间策略。在硬件级别,数据的每个字与字大小的元数据标记相关联。这些标记不由硬件解释;它们可以以软件映射到它们附于的数据的比如类型、起源、分类级别或者可值得信任性的信息的表示。因为标记足够大以表示指针,它们可以指向任意大小和复杂性的数据结构,包括元数据的元组,允许并行推行多个正交策略。标记程序计数器以支持跟踪程序的控制状态的历史;标记程序代码以支持关于代码起源、控制流和区分(compartmentalization)的策略。以规则高速缓存扩充处理器核心,该规则高速缓存允许与指令执行同步的高性能规则解析和用于当查找该高速缓存中未命中时到处理代码的策略的快速上下文切换的特殊操作模式。这允许PUMP以软件的可表达性和可适配性和硬件的性能来促进各种各样的低级策略的推行。

[0708] 我们在该论文中的目的是示出类PUMP标记和规则处理两者相对于真实威胁是有用的,且以规则的形式的写入策略是易处理的。我们通过详细说明可以怎样编程PUMP以支持低级安全性和安全策略的不同集合来进行。我们呈现四族策略(所有都是在文献中熟悉的)的具体实现和估计:(i)基本类型,推行类型安全的弱形式;(ii)空间和时间存储器安全,捕获用于堆分配数据的出界(bounds)和释放后使用(use-after-free)的错误,(iii)控制流完整性(CFI)[2],防止代码重新使用攻击;(iv)污点跟踪,其中污点可以表示可能有助于给定数据的数据源或者组件。大部分这些策略超出当前系统可以以软件高效地支持的。

最终,我们示出怎样可以同时应用这些策略。因为这些策略已经在现有文献中很好地研究,我们的主要焦点不在于它们提供的安全性保证,而是在于探索它们可以怎样表示为规则和以PUMP推行。我们使用指令轨迹模拟以估算当PUMP附于简单的依次RISC处理器(阿尔法[1])时这些策略跨SPEC CPU2006基准点套件的运行时间影响。我们示出PUMP可以以各种各样的复杂性支持策略并量化性能影响。该范围说明当威胁演进时改进策略的能力和该演化可以怎样影响性能。

[0709] 该论文是[7]的扩展的、丰富的和重设侧重的版本,[7]是今年夏天稍晚在研讨会呈现的短论文。先前论文侧重于PUMP到RISC处理器中的直接硬件集成,建立关于大多数基准点的合理性能,和标识用于改进的区域。在当前工作中我们避开在[7]中很好地解释的微架构考虑,代替地侧重于编程模型和策略本身的更加具体的说明和评估。我们也解释PUMP软件服务怎样保护其自身防止滥用。我们报告的性能关于[7]改进是由于:(i)操作组的使用(§2),(ii)未命中成本的更精确估算(§3)和(iii)通过仅在需要时使用指针标记来减少DRAM访问(§4)。

[0710] 总之,该工作的主要贡献是(i)编程模型和支持用于简洁地和精确地描述由该架构支持的策略的界面模型(§2和§3);(ii)策略编码和使用四个不同类别的良好研究的策略的组成的详细实例;和(iii)这些策略的需要、复杂性和性能的量化(§4)。在§5和§6中,我们讨论有关的和未来的工作。几个附加材料可以匿名形式在http://git.io/8K7IKA获得。这些包括:具有研究的策略的完整定义,我们的实验的源代码和[7]的匿名版本的附录。

[0711] 2.策略编程模型

[0712] PUMP策略由标记值的集合与操作这些标记的规则的集合一起组成以实现某些期望的跟踪和推行机制。取决于我们谈论系统的软件层(符号规则)还是硬件层(具体规则),规则以两个形式出现。

[0713] 实例。为图示PUMP的操作,让我们考虑用于在程序执行期间限制返回点的简单示例策略。该策略的动机来自于已知为面向返回编程(ROP)[39]的一类攻击,其中攻击者识别出受到攻击的程序的二进制可执行码中的“小配件(gadget)”的集合,并通过构造堆栈帧的适当序列来使用这些"小配件"的集合来装配复杂的恶意行为,其每个包括指向某些小配件的返回地址;然后利用缓存溢出或者其他脆弱性来以期望序列重写堆栈的顶部,导致这些片段(snippets)依序执行。

[0714] 限制ROP攻击的一个简单方式是将返回指令的目标限制到明确定义的返回点。我们可以使用PUMP通过以元数据标记目标来标记作为有效返回点的指令来这样做。每次我们执行返回指令,我们将PC上的元数据标记设置为已验(check)以指示刚刚发生返回。在下一指令,我们注意到PC标记是已验,证明当前指令上的标记是目标,且如果不是的话,则通知安全性违规。我们将在该部分中之后看到,通过使得元数据更丰富,我们可以精确地控制哪些返回指令可以返回到哪些返回点。通过使得其更丰富,我们可以实现成熟的CFI校验[2](参见§4.3)。

[0715] 符号规则。从PUMP的策略设计者和软件部分的观点,可以使用以微小域-专用语言所写的符号规则紧凑地描述策略。每个符号规则具有形式:

[0716] opgroup:(PC,CI,OP1,OP2,MR)→(PC',R')if guard?

[0717] 其叙述规则与指令操作码的集合(opgroup)相匹配,一起的还有关于程序计数器

(PC)的元数据标记、当前指令(CI)、来自寄存器文件的多至两个操作数(OP1,OP2)和由指令(MR)引用的存储器位置,如果有的话。如果所有相关标记表达匹配且防护?断定(predicate)保持,则应用规则。在该情况下,右手侧确定如何更新关于PC(PC')和关于操作结果(R')的标记。我们使用操作组(opgroup)代替操作码,因为在大多数策略中,将存在具有相同规则的许多操作码。我们写入"-"以指示忽略的输入/输出字段("通配符")。当防护?条件仅是真时,我们省略它。

[0718] 对于仅概略的简单ROP策略,我们将操作码拆分为两个操作组-返回(仅包括单个操作码)和$\overline{return}$(所有其余);可能的标记值是已验(check)、目标和⊥。PC将总是以已验或者⊥标记,且每个指令将标记目标或者⊥。(指令标记由可信加载器提供;参见§3。)符号规则是:

[0719] return:(⊥,-,-,-,-)→(check,-) (1)

[0720] $\overline{return}: (\text{check}, \text{target}, -, -, -) \rightarrow (\perp, -)$ (2)

[0721] $\overline{return}: (\perp, -, -, -, -) \rightarrow (\perp, -)$ (3)

[0722] return:(check,target,-,-,-)→(check,-) (4)

[0723] 规则1叙述,当当前操作是返回(且PC没有已经标记已验)时,我们将关于PC的标记改变为已验。当我们以PC标记为已验运行指令时(规则2),我们检查指令标记CI是目标;如果是,则我们允许操作并清除关于PC的标记。如果当前操作不是返回且PC标记是⊥,我们简单地继续(规则3)。规则4处理返回的有效目标本身是返回的特殊情况。如果不应用规则,则不允许操作(例如,不允许配置PC=check(已验和CI=⊥)。我们假定符号规则不重叠。

[0724] 接下来,让我们考虑该策略的更精确的变型,其中我们保证不仅每个返回达到某个有效返回目标,而且它以代码点为目标,从该代码点它可能实际上已经被调用。该策略假定编译器具有返回点的完全知识,且可以对于每一个来分析它可能潜在地返回到哪个调用地点。使用该信息,我们可能将唯一标记附于每个返回和每个潜在返回目标。在遇到返回时,PUMP将关于指令的标记(而不是通用标记已验)复制到PC上(规则1'和4')。关于下一步骤,它检查实际返回点在期望的返回点当中-即,允许从PC到CI的返回(规则2'和4')。

[0725] return:(⊥,ci,-,-,-)→(ci,-) (1')

[0726] $\overline{return}: (\text{pc}, \text{ci}, -, -, -) \rightarrow (\perp, -) \text{ if } (\text{pc}, \text{ci}) \in \chi$ (2')

[0727] $\overline{return}: (\perp, -, -. -, -) \rightarrow (\perp, -)$ (3')

[0728] return:(pc,ci,-,-,-)→(ci,-)if(pc,ci)∈x (4')

[0729] 在这些规则中,我们使用x(由编译器提供的代码位置标识符对的集合)表示经由代码中的返回的允许的间接控制流。如这里所示,描述符号规则中的标记的表达不限于恒定值:我们可以写紧凑地描述大的标记集合的更通用的表达。

[0730] 具体规则。符号规则可以紧凑地编码多种多样的元数据跟踪机制。但是,在硬件级别,我们需要调整的规则表示,用于有效率的解释以避免减慢主要计算。为此,我们引入称为具体规则的低级规则格式。直观地,给定策略的每个符号规则可以扩展为具体规则的等效集合。但是,因为单个符号规则可以总的来说生成无限制数目的具体规则,我们缓慢地执行该工作,当系统执行时按照需要生成具体规则。

[0731] PUMP硬件包括可以并行于处理器的ALU操作查询的具体规则的高速缓存。当发出指令时，规则高速缓存执行来自当前机器状态的标记(当前PC标记，关于当前指令的操作数的标记等)相对于高速缓存中的所有具体规则的关联匹配。如果找到匹配，则高速缓存返回用于PC的新标记和用于指令的结果的标记。否则，处理器归错于规则未命中处理器-查询策略的符号规则和确定是否应该允许进行出故障的机器状态的软件例程；如果是，则它生成适当的具体规则，在高速缓存中安装它，和重新启动出故障指令。否则，它起用适当的安全性故障处理器。用于具体规则的通用格式是：

[0732] opgroup:(PC,CI,OP1,OP2,MR)＝>(PC',R')

[0733] 其中输入和输出字段是固定标记。注意到不需要符号规则格式中的“防护(guard)?”字段，因为未命中处理器在添加任何具体规则到高速缓存中之前会校验相应条件。

[0734] 一个方便的编码手段大大地减小具体规则的数目。我们注意到对于所有符号规则很普遍地对于给定操作组标明特定输入/输出为“通配符”。例如，在我们的ROP策略中，用于返回的规则和$\overline{return}$操作组不需要关于OP1、OP2和MR输入匹配且不需要产生R'结果。为避免生成用于未使用的输入字段的所有可能的值的具体规则，我们定义包括用于每个操作组和输入字段的不关心位的位向量，其确定相应标记是否实际上用于规则高速缓存查找。类似地，不关心矢量标明未使用的输出，对于其返回默认标记(以下我们对于此使用⊥)。

[0735] 例如，因为对于ROP策略，$\overline{return}$操作组具有用于OP1、OP2、MR和R1'的不关心位集合，规则2'仅导致两个具体规则

[0736] $\overline{retu}:(t1,t2,\ \perp,\ \perp,\ \perp)=>(\perp,\ \perp)$

[0737] $\overline{retu}:(t1,t3,\ \perp,\ \perp,\ \perp)=>(\perp,\ \perp)$

[0738] 如果编译器知道标记为t1的返回指令是代码中仅有的返回且它仅能返回到标记为t2和t3的返回目标。“不关心”位置掩蔽为⊥。另一方面，符号规则3'对应于四个具体规则：

[0739] $\overline{return}:(\perp,\ \perp,\ \perp,\ \perp,\ \perp)=>(\perp,\ \perp)$

[0740] $\overline{return}:(\perp,t1,\ \perp,\ \perp,\ \perp)=>(\perp,\ \perp)$

[0741] $\overline{return}:(\perp,t2,\ \perp,\ \perp,\ \perp)=>(\perp,\ \perp)$

[0742] $\overline{return}:(\perp,t3,\ \perp,\ \perp,\ \perp)=>(\perp,\ \perp)$

[0743] 因为CI不是用于$\overline{return}$的“不关心”位置(而规则3'不标明CI为通配符，规则2'不，且两个规则都关于相同操作码)，对于它可以取的每个可能值我们得到不同具体规则-⊥加上所有标识符(在该实例中，仅t1，t2和t3)。

[0744] 从操作码到操作组和不关心矢量的映射是可编程的。ROP策略仅使用两个操作组(返回和$\overline{return}$)，但是其他策略可能需要更多；例如，原语类型策略(§4.1)使用十个。

[0745] 结构化的标记。对于具有比ROP更丰富的元数据标记的策略，从符号到具体规则的转译遵循相同的总路线，但是细节变得有点更复杂。例如，污点跟踪策略(§4.4)把标记作为

指向存储器数据结构的指针，每个描述任意地大小的污点集合(表示可能有助于给定条数据的数据源或者系统组件)。加载操作组的符号规则叙述加载值上的污点应该是关于指令本身的污点、负载的目标地址和在该地址的存储器的联并集：

[0746] load:(-,ci,op,-,mr)→(-,ci∪op1∪mr)

[0747] 假如，在某时刻，(i)要执行的下一指令是Id r0 r1且其标记是tci，寄存器r0包括标记tp的指针p，且在地址p的存储器包括标记$t_{\emptyset}$的值；(ii)t_{Ci}指向表示集合{TA,TB}的数据结构(即，污点id的阵列)(iii)tp指向{TC,TD}的表示；且(iv)$t_{\emptyset}$指向空集。此外，假如我们之前从未遇到污点{TA,TB,TC,TD}-即，当前在存储器中不存在表示我们应该使用以污染加载结果的集合的数据结构。在该情况下，规则高速缓存查找将未命中且执行将出故障到规则未命中处理器中，这将生成适当的具体规则且在高速缓存中安装它，可能逐出另一规则以腾出空间。这需要分配新存储器(即，在地址t_{new})和初始化其以表示{TA,TB,TC,TD}。

[0748] 生成的具体规则然后将是：

[0749] $$load:(\perp, tci, tp, \perp, t_{\emptyset})=>(\perp, t_{new})$$

[0750] 重新启动指令之后，下一高速缓存查找将继续，且r1中的加载的值将标记为t_{new}。

[0751] 为减小不同标记的数目(且因此，对规则高速缓存的压力)，可以以规范形式内部存储元数据结构，且因为标记不变，完全地利用共享(例如，给予集合元素规范次序以使得集合可以紧凑地表示共享公共前缀子集)。当不再需要时，可以收回(例如，由无用单元收集)这些结构。

[0752] 复合策略。更进一步地，我们可以通过使得标记是到来自几个组件策略的标记的元组的指针，而同时推行多个正交策略。(总的来说，多个策略可能不是正交的；我们在§6中返回到该点。)例如，为以我们刚刚描述的污点跟踪策略来构成第一ROP策略，我们将使得每个标记是到元组(r,t)的表示的指针，其中r是ROP-标记(代码位置标识符或者⊥)，且t是污点标记(到污点的集合的指针)。高速缓存查找处理也是完全一样的，但是当发生未命中时未命中处理器提取元组的组成部分且分派到评估符号规则的两个集合的例程。仅当两个策略都具有应用的规则时允许操作；在该情况下，产生的标记是到包括来自两个子策略的结果的对的指针。

[0753] 指令修饰符和短暂规则。某些策略(例如存储器安全)需要动态地生成的新鲜标记。实现该效果的一个方式是使用关于比如移动的指令的标记作为修饰符以将对新鲜标记的请求传递到策略管理系统。

[0754] $$Move:(-, t_{policygen}, -, -, -)\xrightarrow{1}(-, t_{newtag})$$

[0755] 这叙述以$t_{policygen}$标记的移动指令被解释为生成新鲜标记的请求。该结果t_{newtag}是与指定策略相关联的唯一标记。指令上的标记$t_{policygen}$也用作该服务请求的授权或者能力；没有该标记，不可能做出调用；可信加载器保证仅以该标记注释特殊指定的代码区域(例如，在§4.2中的存储器安全策略中的malloc例程)。“1”指示短暂规则，其结果不是持续地存储在硬件规则高速缓存中(因为它随每个起用改变)。

[0756] 用于初始化标记的代码也可能需要替换“稳态”规则。例如，在存储器安全策略中，malloc将需要初始化关于新分配的存储器区域的标记。标准规则是指针仅可以写入到适当地标记为匹配该指针的存储器区域中。但是当将新造的标记写到新区域中的每个字上时必

须允许malloc替换该规则。我们通过给予存储操作以特殊修饰符标记(仅用于malloc)来这样做：

[0757] store:(-,$t_{mallocinit}$,t1,c2,F)→(-,(c2,t1))

[0758] 3.策略系统和保护

[0759] 策略系统作为每个用户处理内的存储器的分开区域存在。它包括用于未命中处理器的代码、策略规则和表示策略的元数据标记的数据结构。在进程中放置策略系统最小地破坏现有的Unix处理模型，且促进策略系统和用户代码之间的轻量切换。策略系统使用接下来描述的机制与用户代码隔离。

[0760] 元数据威胁模型。清楚地，如果攻击者可以重写元数据标记或者改变它们的解释，则由PUMP提供的保护将是无用的。我们的系统设计用于防止这种攻击。我们信任内核程序、加载器和(对于某些策略)编译器。具体来说，我们取决于编译器向字分配初始标记，且当需要时，向策略系统传递规则。我们假定加载器将保存由编译器提供的标记，且例如使用加密签名保护从编译器到加载器的路径免于篡改。我们假定将建立用于每个进程的初始存储器图像的标准Unix-类型内核程序。(可以使用微策略以消除一些假定，进一步减小TCB的大小-参见§6)。我们进一步假定正确地实现规则高速缓存未命中处理软件。这很小，因此是用于形式验证的好目标。最近的工作[8]表明类似于PUMP的编程模型的可实行性。

[0761] 我们的主要关切是防止在进程中运行的用户代码破坏由进程的策略提供的保护。用户代码不应该能够(i)直接操纵标记-所有标记改变应该根据当前生效的策略规则来执行；(ii)操纵由未命中处理器使用的数据结构和代码；(iii)在硬件规则高速缓存中直接插入规则。

[0762] 寻址。为防止由用户代码的标记的直接操纵，附加到每个64b字的标记本身不是分开地可寻址的。具体来说，不可以指定仅与标记或者标记的一部分对应的地址以读取或者写入它。所有用户可访问指令在作为原子单元的(数据，标记)对上操作-标准ALU在值部分上操作，和PUMP在标记部分上操作。

[0763] 未命中处理器架构。策略系统仅关于对PUMP高速缓存的未命中激活。为提供策略系统和用户代码之间的隔离，我们将未命中处理器操作模式添加到处理器；我们也以仅可用于未命中处理器的16个附加寄存器来扩展整数寄存器文件，以避免保存和恢复寄存器。故障指令的PC，规则输入(操作组和标记)和规则输出当在未命中处理器模式下时表现为寄存器。我们还添加未命中处理器返回指令，其结束安装具体规则到高速缓存中且返回到用户代码。

[0764] 当处理器处于未命中处理器模式时脱离PUMP的正常行为。代替地，应用单个硬布线规则：由未命中处理器接触的所有指令和数据必须以与由任何策略使用的标记不同的预定义的未命中处理器标记来标记。这保证相同地址空间中的未命中处理器码和数据与用户代码之间的隔离。用户代码不能接触或者执行策略系统数据或者代码，且未命中处理器不能无意地接触用户数据和代码。未命中处理器返回指令可以仅在未命中处理器模式下发布，防止用户代码将任何规则插入到PUMP中。

[0765] 4.策略和实验

[0766] 在该部分中，我们示出如何使用PUMP实现推行安全性不变性的不同集合的四族策略。对于每个族，我们首先概述威胁模型。我们然后描述缓解其的策略和相应规则。使用来

自公开的脆弱性套件[10]的实例,我们示出每个策略将怎样得到典型利用。最重要地,我们描述每个策略置于系统上的负载。我们通过与来自文献的类似策略比较来结束。

[0767] 为评估策略负载,我们使用来自SPEC CPU2006[25]基准点套件的28C,C++和FORTRAN应用,并以gem5模拟环境[9]对于64位Alpha ISA[1]模拟它们(我们排除在其上gem5失败的tonto和xalancbmk基准点)。gem5模拟不直接建模PUMP;而是,其产生我们通过运行分开的PUMP模拟器的指令踪迹。该调节的模拟对于这里描述的策略足够了,因为它们对计算的影响仅是当策略违规发生时中止执行。我们模拟4096项未命中处理器前(pre-miss-handler)的规则高速缓存。

[0768]

	无		类型		存储器安全					CFI					污点跟踪						组合	
	①无未命中处理器	②不重要的处理器	③insn/addr/其它	④i/a/o + retadd	⑤沙盒	⑥1色	⑦8色	⑧32色	⑨定点保护(2^{64}色)	⑩IID [2]	⑪2ID [2]	⑫CCFIR [51]	⑬PUMP JOP	⑭PUMP ROP	⑮PUMP CFI (⑬⑭)	⑯1污点	⑰每个流	⑱每个库	⑲每份文件	⑳每个函数	㉑每个流和库 (⑰⑱)	㉒最少成分 (②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑)	㉓全部成分 (⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑)
标记使用	-	-	p		p	pp1	op2	mr	r	pc	ci	p	p	pc	pc	ci	pp1	op2	mr	r	all	
操作组	-	-	10	11	7					2	3	3	2			7					13	
符号规则	-	-	15	16	13					2	3	4	4			8					49	53
初始标记	1	-	3	4	34	6	6	6	6	2	4	6	351	1.7K	2.1K	2	3	34	371	1.2K	36	16	2.2K
动态分配的标记	0	-	0	0	2	2	18	241	102K	0	0	0	0	0	0	0	1	76	381	2.3K	77	4	252K
最终标记	1	-	3	4	36	8	26	247	102K	2	4	6	351	1.7K	2.1K	2	4	110	752	2.5K	113	20	255K
具体规则	1	-	17	19	14	18	93	578	386K	5	11	21	426	1.8K	2.3K	10	14	1.1K	6.5K	47K	1.2K	79	1.1M
元数据结构	-	-	-	-	2	2	2	2	2	-	-	-	-	-	-	1	2	5	13	50	5	4	5
元数据空间	-	-	0	0	2	2	24	418	51K	0	0	0	2K	26K	28K	3	9	510	13K	573K	517	76	652K
策略-依赖指令	0	10	116	136	64					24	44	64	682			810					1240	1800
(动态)	0	10	25	26	20	20	32	36	38	16	26	31	117	120	120	56	174	430	1076	4110	430	211	710
运行时间开销(%)	8	8	8	8	8	8	8	8	13	8	8	8	8	8	8	8	8	8	9	314	8	8	38

[0769] 图表1:策略和特性(对25SPEC CPU2006基准点平均)

[0770] 在§2中描述的抽象编程模型不在软件级别对唯一标记的数目、具体规则的数目或者用于表示元数据的数据结构的大小施加限制。为理解PUMP实际上怎样执行,必须考虑很多问题。给定策略、应用和数据集实际上生成多少唯一元数据标记?用O个操作组和T个标记,理论上程序可能需要O·T5个具体规则,但是典型情况是什么?元数据标记的总数和元数据表示的大小怎样影响性能?对标记和规则使用有多少局部性?具体规则解析成本多高,且规则高速缓存未命中怎样影响性能?当标记、规则、元数据大小或者规则解析时间增加时性能恶化不严重?

[0771] 为开始理解这些效果,我们对于每个策略测量除运行时间开销之外的特性的数目-参见图表1。标记使用(Tage usage)示出哪些标记不由策略中的任何规则使用。操作组(Opgroups)是捕获策略需要的操作组的最小数目;我们使用的操作组越少,我们对于具体规则得到越大压缩,且因此有效的PUMP容量越大。符号规则(Symbolic rules)是我们写入以表示策略的符号规则的数目。初始标记(Initial tags)是在执行开始之前的初始存储器图像中的标记的数目。在执行期间将动态地分配更多标记(dyn.alloc.tags)。此外,比如污点跟踪的策略将创建标记以表示污点的集合的并集,且复合策略将形成单独的策略标记的元组。最终标记(Final tags)标识在十亿个指令模拟周期的结尾存在的标记的数目;这给出策略复杂性的某些感受且可以用于推断标记创建的速率。具体规则(Concrete rules),模拟周期期间生成的唯一具体规则的数目,将符号规则解析为具体规则需要的推行未命中的数目特性化,且实际上,特性化强制性未命中率。元数据结构(Metadata struct),由每个标记指向的数据结构的字中的平均大小,说明具有无限制的元数据的值。元数据空间(Metadata space),保持元数据标记指向的有关策略的信息的所有数据结构所需的字的数目,特性化超出标记本身的存储器开销。策略-依赖指令(Policy-depend instrs)是将符号

规则解析为具体规则的代码所需的指令的总数;这在理解策略的复杂性上是有用的。策略-依赖指令(动态的)(Policy-depend instrs(dynamic))是执行以从符号规则解析为具体规则的策略-依赖的指令的平均数目;这指示每个策略的未命中处理器的运行时间复杂性。策略-依赖的部分的影响取决于规则的复杂性,元数据数据结构,元数据数据结构的局部性和分配新结果标记的需要。未命中处理器的策略-独立部分仅需要几十个指令(参见图表1中的列(B))。运行时间开销是运行策略的应用的壁钟运行时间与不具有PUMP的基线Alpha相比的比率。即使没有使用策略,只是为了添加用于标记和PUMP的硬件结构也存在某些运行时间开销。具体来说,标记增强的处理器上的L1高速缓存是无PUMP的基线Alpha有效容量的一半,以在容纳更大的标记的字宽度的同时实现相同循环时间。这导致标记增强的处理器的较高L1未命中率。该开销在第一列(Ⓐ)捕获,其中所有标记是默认的,存在单个规则,且实际上从不起用未命中处理器。

[0772] 图表1中的平均数目是出于紧凑性的简化。基准点展现效果的范围。这些如图表3-图表6所示,其中我们使用框图来示出在SPEC CPU2006基准点集合中跨应用的特性分布。图表6示处超过Ⓐ的运行时间开销。

[0773] 我们仅测量运行时间性能,而搁置某些其他非平凡(nontrivial)成本。具体来说,在自然实现中,添加字大小的标记到高速缓存和存储器的每个字施加最小2x面积开销。添加PUMP高速缓存和较大的存储器的影响,这可能转化为4x能量开销。我们对于仔细的优化可以减小这些数目到大约30%面积和50%能量或者可能甚至更低是乐观的。我们正工作来表明该要求。

[0774] 4.1原语类型

[0775] 威胁模型。数据误解释是欺骗处理器执行不想要的操作的通常方式。这里我们考虑低级类型混淆的形式,其中代表对手运行的代码可能尝试使用任何数据值作为指针或者执行字作为指令,我们强制不能执行数据且不能在运行时间创建或者修改代码(还参见§4.3)。

[0776] 策略和规则。在策略Ⓒ中我们使用标记来分开指令(标记的insn)、地址(addr)和所有其他数据(其他)。不能创建或者修改指令,且仅可以执行指令。仅地址可以用于存储器访问指令。另一类型的标记用作用于不是指令或者地址的字的总受器(catch-all)。以下规则验证nop(例如)在其执行之前实际上标记insn:

[0777] nop:(-,insn,-,-,-)→(-,-) (5)

[0778] 允许地址运算-例如,当到add的变元之一是地址,结果是地址时:

[0779] add:(-,insn,addr,other,-)→(-,addr) (6)

[0780] 我们也强制加载和存储指令只间接引用(dereference)指针,但是不读取或者写入指令:

[0781] load:(-,insn,addr,-,t)→(-,t)if t/=insn (7)

[0782] store:(-,insn,t,addr,-)→(-,t)if t/=insn (8)

[0783] 为帮助防止其中返回地址被重写的攻击(例如,经由堆栈粉碎),我们考虑添加用于返回地址的第四标记(retaddr)的扩展的策略(Ⓓ)。我们使用其标记调用的返回地址(规则9)。Alpha ISA中的调用在reg26中放置返回地址,同时返回转移控制到该寄存器中的

地址(地址寄存器关于进一步调用溢出到堆栈)。规则10校验当执行返回指令时reg26中的值是键入的retaddr。

[0784] call:(-,insn,addr,-,-)→(-,retaddr) (9)

[0785] return:(-,insn,retaddr,-,-)→(-,-) (10)

[0786] 仪器化的编译器可以推断这些类型标记且将它们应用于二进制的初始存储器图像-所有生成的指令得到标记的insn,到堆栈分配的存储器的指针得到标记的addr,且其他每个得到标记的其他;新地址类型字经由动态存储器分配产生。但是,因为我们当前不具有这种编译器,我们使用不同方法推导这些标记以用于我们的模拟和分析。首先,我们标记二进制可执行的insn中的所有指令。为推导应该标记地址的字,我们使用执行轨迹的事后分析,保持跟踪何时和从哪里加载每个寄存器且它是否之后用作到加载或者存储的指针操作数。其他每个标记为其他。该获得最初标记的方法允许我们测量关于SPEC基准点归类(typing)策略的运行时间影响。但是,该设置不允许我们做出关于是否我们的归类策略将足够宽容以接受所有基准标记而不产生不必要的警报的任何主张。这由归类需要的紧凑编译器集成引起,且对于我们以下呈现的其他策略不发生。

[0787] 保护示范。我们使用CWE-843(类型混淆)[30]的实例,其中程序员类型掷出(cast)一个整数到函数指针且之后起用该函数。这转化为加载标记为寄存器以外的立即值到寄存器中,且在之后的点,跳到由该寄存器指向的地址。使用策略Ⓒ我们能够抓住故障指令,因为该策略允许仅到标记地址的值的间接跳转。

[0788] 特性。策略Ⓒ和Ⓓ不创建新标记。Ⓒ可以用仅生成17个具体规则的15个符号规则来编码,同时Ⓓ需要16个符号规则和19个具体规则。因为规则的总数很小,我们仅看到可忽略的运行时间开销(与无未命中处理器策略Ⓐ相比小于0.01%)。因此,PUMP提供简单的硬布线类型标记的性能,而不将策略烧固到硬件中。

[0789] 有关工作。计算机架构中标记的首先使用之一是区分机器中的字的类型[34,23]。符号象征学LISP机[31]分配它们的36b原语字当中的用于标记的2-8b以区分包括指令、指针的几个味、整数、浮点和未初始化值的原语类型的集合;伯克来SPUR[43]用于6b对象类型标记。

[0790] 4.2空间和时间存储器安全

[0791] 威胁模型。下一组策略对准堆分配的数据的存储器安全,防止攻击者利用编程错误,比如超出对象的边界的引用(空间违规),已经释放区域之后经由指针的引用,或者释放无效指针(时间违规)。这包括典型的基于堆的攻击,比如堆粉碎和指针伪造。我们这里研究的该策略仅防护堆分配的数据,对于其对malloc和free的调用告诉我们如何建立和拆卸存储器区域;我们不处理堆栈分配或者解开(unboxed)的结构。假定某些编译器支持(参见[32]),这些可以原则上也被处理。

[0792] 策略和规则。直观地,对于每个新分配,我们构造新鲜的块id,称为c(用于“颜色”),且将c写为关于新创建的存储器块中的每个存储器位置的标记(a la memset)。到新块的指针也标记c。之后,当我们间接引用指针时,我们校验其标记与关于其指向的存储器单元的标记相同。当释放块时,关于块的全部单元的标记可以改变为表示空存储器的常数F。

[0793] 我们使用用于非指针的附加的标记⊥,且写入用于作为颜色c或者⊥的标记的t。我们注意到一个附加细节-存储器单元可以包括指针。所以存储器中的字必须与两个标记相关联。我们通过使得关于每个存储器单元的标记是到对(c,t)的指针来处理它,其中c是其中分配该单元的存储器块的id且t是关于单元中存储的字的标记。与检查每个存储器访问有效(即,访问的单元在由该指针指向的块内)一起,用于加载和存储的规则负责封装和解封这些对:

[0794] load:(-,-,c1,-,(c2,t2))→(-,t2)if c1=c2 (11)

[0795] $$\text{store}:(-,-,t1,c2,(c3,t3)) \quad (12)$$
$$\rightarrow(-,(c3,t1))\ \text{if}\ c2=c3$$

[0796] 地址运算操作保存指针标记:

[0797] add:(-,-,c,⊥,-)→(-,c) (13)

[0798] 为维持关于指针的标记仅可以来源于分配的不变性质,从头创建数据的操作(类似加载常数)将其标记设置为⊥。

[0799] 我们扩充malloc和free以使用在§2的结尾描述的指令修饰符和短暂规则来标记存储器区域。在malloc中,我们经由短暂规则生成用于到新区域的指针的新鲜标记。我们然后使用新标记的指针,以使用特殊存储规则将零写入分配的区域中的每个字

[0800] store:(-,tmallocinit,t1,c2,F)→(-,(c2,t1)) (14)

[0801] 在返回标记的指针之前。相反地,free使用修改的存储指令以重新标记区域为未分配的。

[0802] store:(-,tfreeinit,t1,c2,(c3,t4))→(-,F) (15)

[0803] 在将存储器区域返回到释放列表之前。

[0804] 我们实现该策略的几个变型,说明不同的性能/安全性折衷。在第一个(Ⓔ)中,我们分配单个颜色给由给定源模块分配的所有存储器区域。该沙盒策略提供进程内的每个模块隔离,类似于基于软件的错误隔离[46]。在下一变型中,我们使用不同的多个颜色来标记由对malloc的连续调用返回的区域-仅从单个颜色(Ⓕ)-这提供空间和时间存储器安全的最弱形式,仅区分分配的与未分配的存储器-对8个(Ⓖ和32个(Ⓗ)颜色。增加颜色的数目减小由于颜色的重新使用而发生的混叠效应。最终,我们使用用于颜色的整个64位标记空间,实现精确的完全存储器安全策略(Ⓘ)。

[0805] 保护示范。我们使用来自Juliet套件[10]的两个攻击。第一个是CWE-416的情况(释放之后使用)[28],其中使用尝试从标记为F的存储器位置加载的策略(Ⓘ)来抓住应用。第二个是CWE-122的情况(基于堆的缓冲器溢出)[27],其中分配缓存且之后写入超出它的边界(使用strcpy),重写有效区域。使用Ⓘ,PUMP停止尝试在标记为F的存储器位置放置字符(character)的指令。

[0806] 特性。沙盒(Ⓔ)和具有少数颜色的策略(ⒻⒼ和Ⓗ)仅分配几个标记和创建少数规则(对于32颜色情况小于600)。这些不添加运行时间开销-规则全部在高速缓存中适配。完全存储器安全(Ⓘ)更昂贵:它每个存储器分配分配一个标记,对于存储器分配新具体规

则必须添加到高速缓存。这需要经由未命中处理器的更多旅程且意味着在某些基准点中，具体规则的集合比高速缓存更大。

[0807] 尽管如此，规则局部性很高(参见图7)，且平均运行时间开销仅是13％。我们看到对于GemsFDTD的大约130％的最大开销。

[0808] 有关工作。Clause等[[16]首先示范使用元数据污染的空间和时间存储器保护。Deng等[[19,20]以硬件标记管理支持该污染。硬边界[21]是将边界信息置于阴影空间中以维持监视和无监视代码之间的数据结构布局兼容性的对空间存储器安全的方法。硬边界的运行时间开销是10-20％。看门狗[32]是硬边界的后续措施，其另外通过生成每个分配的唯一标识符来防止时间违规；它具有24％的平均运行时间开销。软边界[33]是软件方法，其类似硬边界，提供用于C的空间存储器安全，但是以增加的运行时间开销(关于SPEC和Olden基准点的67％)为代价。Baggy Bounds[3]也仅针对空间违规且实现关于SPEC2000的60％的运行时间开销。

[0809] 4.3控制流完整性

[0810] 威胁模型。该组策略对准代码重新使用攻击。我们做出攻击者可能既不执行数据也不注入或者修改代码的标准假定[2]。(我们可以使用来自§4.1的基本类型策略以推行该假定，如我们以我们构造的策略在§4.5中做的。)代替地，攻击者尝试将现有代码片段(小配件)链在一起以引起恶意行为。

[0811] 策略和规则。所有代码重新使用攻击的公共要素是引入在初始二进制中不存在的控制流。我们实现相对于程序的控制流向图验证每个间接控制流(计算的跳转)的CFI策略的族。因为固定代码，不需要动态地校验直接跳转[2]。首先，我们实现[2,51]的粗粒度的CFI策略(Ⓙ Ⓚ和Ⓛ)。Ⓙ标记所有间接调用、间接跳转和返回指令及其具有单个标记{f}的潜在目标。在执行之外间接控制流的源的指令时，我们将该标记转移到PC：

[0812] indir:(-,{f},-,-,-)→({f},-) (16)

[0813] 所有其他指令标记Ø。无论何时PC标记{f}，当前指令必须具有相同标记：

[0814] $\overline{\text{indir}} : (\text{pc}, \text{ci}, -, -, -) \rightarrow (\emptyset, -) \text{ if } \text{pc} \subseteq \text{ci}$ (17)

[0815] 策略Ⓚ使用更多标记(Ø,{r},{c}和{r,c}以分开地跟踪来源于从来源于间接调用和跳转(其标记包括c)的那些的返回(其标记包括r)的控制流。策略Ⓛ以用于返回到特权代码(其标记包括p)中的两个附加标记({p}和{p,c})来扩展Ⓚ，允许关键代码片段的附加保护[51]。

[0816] 如Goktas[22]等的攻击示出，这些松散的CFI策略不足以针对复杂的代码重新使用攻击提供保护。我们还实现细粒度的CFI策略的集合，Goktas等将其描述为“理想CFI”。我们首先引入两个正交策略：精确地跟踪间接跳转及调用和它们的目标之间的关联的PUMP JOP(Ⓜ)；和对于返回做同样事情的PUMP ROP(Ⓝ)，如在§2(规则1’-4’)中呈现的。我们最终将这两个策略合并到PUMP CFI(Ⓞ)中-精确地跟踪和验证所有间接控制流的单个策略。在所有这些策略中，编译器或者链接器假定为相应地计算间接控制流和标记指令的声音过近似。

[0817] 保护示范。我们相对于由对“无害”函数的单个调用组成的特别设计的程序测试这

些策略。代码还包括从不调用的“坏”函数，模仿不是执行路径的一部分但是可能被利用以导致不想要的行为的休眠小配件。为模拟面向返回的攻击，无害函数中的成行装配以坏函数的地址重写堆栈指针，将执行欺骗为返回到坏函数中。策略Ⓝ通过通知坏的返回不在有效控制流的集合中来检测该模拟攻击。

[0818] 特性。以上CFI策略中的每个可以仅以2-4个符号规则非常紧凑地编码。更简单的策略(Ⓙ，Ⓚ和Ⓛ)也需要非常小数目的标记和具体规则。如图1所示，(Ⓛ)最大，使用6个常数标记且不需要多于21个具体规则。通过这种小的工作集合大小，这些策略不导致空策略以上的可观察到的运行时间开销。将更强的CFI策略(Ⓜ，Ⓝ，Ⓞ)应用于SPEC基准点产生多达用于这些策略的几千个具体规则，其完全适配到4096个项中，预未命中处理器PUMP高速缓存。因此，我们得到理想CFI的增加保护而没有附加的运行时间开销。完整CFI(Ⓞ)策略需要平均28K字以存储用于应用的控制流图(对于该模拟，我们从由gem5生成的指令轨迹提取它;实际上，它将占用比示出的更多空间，包括在我们的模拟中从不实践的允许的控制流路径)。

[0819] 有关工作。CFI[2]提供相对于普遍的代码重新使用攻击的有吸引力的防卫，但是通常考虑它太昂贵。近来的工作[51]已经示范了低开销CFI方案，其使用“跳板”提供分支目标校验和随机化为相对于成功构造的小配件的进一步防卫。但是，该工作仅锁定允许的调用和返回目标，类似于来自§2的规则1-4中的单个目标实例，不是如策略Ⓝ，Ⓜ和Ⓞ做出的具有指定目标的指定返回点，使得其对攻击脆弱[22];且它未解决过程内CFI，如我们的Ⓜ和Ⓞ做的。

[0820] 4.4污点跟踪

[0821] 威胁模型。该策略解决攻击者将变形数据输入到不进行输入清洁的程序的情况，引起不想要的或者恶意的行为(例如，SQL或者OS命令注入)

[0822] 策略和规则。污点跟踪通过检测何时不可信数据可以流入敏感操作而缓解这些威胁。PUMP促进细粒度污染跟踪，用无限数目的源，每个源分开的污点和每条数据上的多个污点，允许每个标记是到源id的集合的指针。关于值的污点是关于用于计算它的这些值的污点的并集。典型污点传播规则包括：

[0823] $\mathrm{add}:(-,ci,op1,op2,-)\rightarrow(-,ci\cup op1\cup op2)$ (18)

[0824] $\mathrm{load}:(-,ci,op1,-,mr)\rightarrow(-,ci\cup op1\cup mr)$ (19)

[0825] $\mathrm{store}:-,ci,op1,op2,-)\rightarrow(-,ci\cup op1\cup op2)$ (20)

[0826] 我们学习的所有策略使用相同的符号规则集合，不同仅在于初始污点的数目和源。我们以两个不同方式引入污点:通过输入源(Ⓟ和Ⓠ)和通过代码区域(Ⓡ，Ⓢ和Ⓣ)。在Ⓟ中，我们使用用于所有输入源的单个污点(即，用于SPEC程序的标准I/O流和输入文件)。这类似于大多数的先前工作[41]，其中单个位的污点t简单地指示来自不可信源的任何数据是否已经用于计算标记为t的值。策略Ⓠ通过向每个输入流分配唯一污点id来扩展Ⓟ;没有流的数目的限制。通过程序代码的污染针对不可信库和有编程缺陷的组件进行保护。我们通过对于(i)每个库(Ⓡ)，(ii)每个包括的报头文件(Ⓢ)，或者(iii)代码(Ⓣ)

中的每个函数使用唯一污点来改变粒度。这些策略需要编译器以相关的污点标识符标记指令。最终,我们组合Ⓠ和Ⓡ以形成策略Ⓤ。

[0827] 保护示范。我们考虑CWE-78(OS命令注入)[29]的情况,其中仅允许用户参数化传递到system系统调用的Is命令的变元。恶意用户与任意命令一起添加从命令终结符字符开始的参数串。这转化为标记为“不可信”的后清洁的数据,以作为变元传递到execve系统调用。使用策略Ⓤ,PUMP在其看到它即将组合不可信与系统调用污点时停止执行。

[0828] 特性。所有这些策略使用就7个操作组而言定义的相同的8个符号规则的集合。前两个(Ⓟ和Ⓠ)使用输入流作为污点源。对于Ⓠ,跨所有SPEC程序,我们平均仅看到2个源,且我们而已需要10和14个具体规则。因此,这些策略不导致值得注意的运行时间开销。对于策略Ⓡ-Ⓤ,我们看到更大的工作集合。

[0829] 通过功能实验(Ⓣ)的污点故意地推动机制到极端,提供比实际上可能有用的更细粒度的标记。它的大数目的污点导致比PUMP高速缓存一次可以保存的更大量级的规则。此外,标记处理开销变大(4110个指令)。这些因素导致314%的平均运行时间开销。这示出PUMP机制在复杂策略下收紧但是仍然可以支持它们。每个文件的污点(Ⓢ)也比可能有用的更细粒度,且由于更小的规则集合和更紧的未命中处理器解析,其实现在9%的低运行时间开销。

[0830] 我们分配污点到整个库的策略Ⓡ和Ⓤ表示更合理的使用。这里,平均运行时间开销保持与无未命中处理器情况没有区别。这示出PUMP能够表示和支持实质上不具有附加运行时间开销的更丰富的模型(与使用1b-或者4b-污点的先前工作相比)。此外,跨这些各种污点情况,最终标记仅是2-3x初始和动态分配的标记;这示出当我们创建非单元素标记集合时,我们看不到接近于理论上最坏情况幂集效应的情况。

[0831] 有关工作。已经使用污点跟踪解决的脆弱性包括格式串攻击[48,17,41,18,12],跨站点脚本编写[48,18,12],存储器利用[48,17,41,14,18,36,12],代码注入[48,17,18 12]和其他[49,18]。大多数现有工作聚焦于软件技术,其中程序是仪器化的。典型地,除其他障碍(例如,动态二进制转化中的处理竞态条件[15])外,这些引入显著的运行时间开销(通常在2x以上,某些多达20x)。

[0832] 类似Dl FT[41],Minos[17],和SI FT[35]的硬件方法使用单个污点位。Raksha-关于核心[18]和专用协同处理器[26]变型两者-支持使用4位标记的多达四个并行策略。相反地,我们允许与可能具有不同级别的可可信性的多个不可信源对应的任意污点集合。在§5讨论更灵活的标记方案。

[0833] 4.5复合策略

[0834] 这些策略中的每一个是可能有用的,但是如果必须一次仅选择单个策略推行将是不可取的-例如,在保护免于缓存溢出或者命令注入脆弱性之间做出选择。代替地,典型地想要来自组成多个策略的保护。事实上,某些我们的单独策略需要相互保护以防卫完全威胁(CFI取决于用于保证不能执行数据和不能创建或者修改代码的类型保护)。

[0835] 组合可能潜在地增加标记的数目以及创建的规则的数目,由此显著地恶化性能。为了特性化组合效果,我们实现两个复合策略。首先,我们实现基于四个保护类别中的每一

个的最简单实例的完全最小的一个:3原语类型(Ⓒ),简单存储器安全(Ⓕ),CCFIR(Ⓛ)和单个位输入-污点(Ⓟ)。第二,我们实现作为4原语类型(Ⓓ),完全空间和时间存储器安全(Ⓘ),PUMP CFI(Ⓞ)和每个流输入污点和每个库代码污点的组合(Ⓤ)的组合的更完整和强大的保护。

[0836] 特性。简单复合策略Ⓥ适于高速缓存中且具有与组成的策略相同的性能。对于更大的复合策略Ⓦ,解析所有策略的需要实大大增加未命中处理器中的规则解析所需的指令的数目,将其从38提升到710。该最终标记的增加仅是2.5x,提出存在来自复合标记的某些乘积集合作用,但是不接近最坏情况场景。此外,由于标记的更大集合和附加操作组两者,具体规则仅增加大约3x。更大的具体规则集合(现在远大于PUMP高速缓存容量)和增加的未命中处理器成本的组合导致38%的平均开销,其中最坏情况开销表现为高达280%(GemsFDTD)。这示出PUMP可以以性能影响的代价处理从复合导致的大的规则集合。对于许多应用,开销保持适度,但是对于某些它变得不合理地大。这与通过函数实验的污点(Ⓣ)一起,指向附加软件和微架构优化以减小未命中处理器服务时间的需要,以实现关于比如作为我们进行中的工作的侧重点的丰富复合体的合理的性能。

[0837] 4.6讨论

[0838] 规则的总数不完全地捕获规则的局部性且因此不完全捕获有效工作集合大小。图表7示出了在用于gcc基准点的十亿指令模拟内的每一个百万指令序列内使用的唯一规则的数目的累积分布函数(CDF)。这示出完全存储器安全(Ⓘ)具有非常紧凑的工作集合(大部分小于3000);因为它具有任何非复合策略的最大数目的具体规则,这是重要的。该局部性帮助解释为什么尽管有大得多的规则集合,性能开销也保持很低。完整CFI(Ⓞ)具有更大的工作集合,但是完整规则集合完全地适配4096项高速缓存,所以没有逐出(eviction)。

[0839] 当先前工作已经使用聪明的方案来紧凑地表示或者近似安全和安全性策略(例如,[42])时,这通常是关于想要的策略的折衷,且它可以平衡复杂性和紧凑性。我们示出了可以包括以很少或者没有附加运行时间开销而更完全地和更自然地捕获安全性策略的需要的更丰富的元数据。不是施加元数据表示和策略复杂性的固定边界,PUMP 10提供性能上的适度退化。这允许策略在需要时使用更多数据而不影响普通情况性能和大小。它进一步允许策略的增加的精细化和性能调节,因为即使复杂策略也可以容易地表示和执行。

[0840] 4.7其他微策略

[0841] 我们相信我们的编程模型可以编码许多其他策略。信息流控制(例如,[6,37,40,24,8])比这里的简单污点跟踪模型更丰富,但是可以以RIFLE风格二进制转化[44]或者通过使用具有来自编译器的某些支持的PC标记来支持跟踪隐含流。微策略可以支持轻量级的访问控制和分区[47]。标记可以用于区分不可伪造的资源[50]。唯一的生成的令牌可以作为用于密封和签署数据的密钥,其然后可以用于强抽象-保证数据仅由授权的代码组件创建和解构。微策略规则可以推行数据不变性质,比如不变性和线性。微策略可以支持如用于比如用于数据或者未来的满/空位(例如[5])的同步原语的带外元数据或者如检测关于锁定的竞态条件的状态(例如,[38,52])的并行性。系统设计师可以向现有的代码应用专用微策略而不用核查或者重写每条线。

[0842] 5.有关工作

[0843] 与我们的示例策略有关的工作已经在§4覆盖。这里,我们讨论通常有关硬件标记校验和传播的工作。通过以下提到的几个例外,大部分在前工作使用具有硬布线的或者高度限制的策略的小的标记位集合(参见图表2)。支持附加到每个字的单个污点位的污点硬件的第一波具有硬布线的污点传播逻辑。后来的系统添加处理多个独立的污点标记(例如,[18]),多个位标记(例如,[45])和更灵活的策略(例如,[19])的能力。支持一次性多于一个策略的唯一设计Raksha支持至多四个污点跟踪策略[18]。

[0844] 最接近我们的在前系统是Aries[11],FlexiTaint[45],基于登录的架构(LBA)[13],和Harmoni[20],其全部提出由软件处理器反向的可编程规则高速缓存。仅FlexiTaint和LBA详述使用可编程规则高速缓存的特定示例安全性策略。在除了LBA之外的所有情况下,规则高速缓存基于用于操作的两个操作数的两个输入并产生单个输出,同时PUMP潜在地得到多达五个输入并产生两个输出:图表1概述这些标记源和目的地怎样用于我们的安全性策略。LBA潜在地得到多个输入,但是它不以硬件处理元数据的产生。使得其快速的LBA中的某些创新(例如,一般传播跟踪到包括关于污点组合放弃的一元遗传跟踪的限制)具体地放弃我们的解决方案提供的通用性。即使对于这些限制的策略,与我们的对于大多数的单个策略的8%的平均开销相比,LBA也具有～50%的运行时间开销。我们这里示出的策略比由FlexiTaint支持的更丰富,这是由于额外的标记输入和输出和更丰富的标记元数据两者。

[0845]

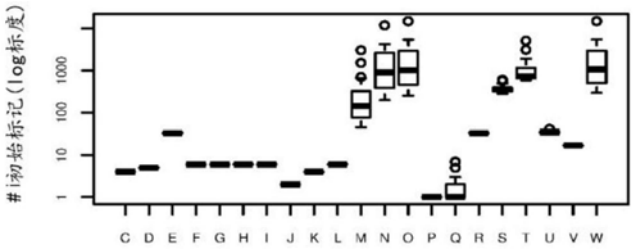
参考文献	HW/SW	标记 大小	标记 IO	安全性策略
[14, 17, 35]	H	1b	2/1	硬件污点
[41]	H+S	1b	2/1	硬件污点
[12]	H+S	1b	2/1	有限的可编程性
[18, 26]	H+S	4b	2/1	四1b策略; 有限的sw可编程性
[45]	H+S	4b	2/1	有限的可编程性
[4]	H+S	1-8b	1/-	不传播, 几乎全部被存储
[19]	H	var	2/1	FPGA 可重构的
[20]	H+S	var	2/1	有限的可编程性, 添加表
[13]	H+S	64b	5/-	在分开的、增大的核上的sw
PUMP	H+S	64b	5/2	细粒度的、sw定义的策略

[0846] 图表2硬件标记方案

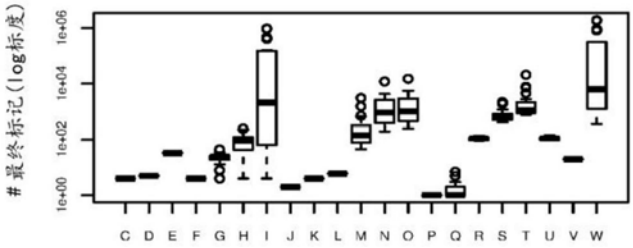
[0847] 6.未来工作

[0848] PUMP设计提供灵活性和性能的吸引人的组合,在以当规则复杂性增长时最和缓的性能下降支持更丰富和合成策略的同时,以在多数情况下可与专用机制相比较的单个策略性能支持不同集合的低级、细粒度安全性策略。为更全面地理解该设计空间,多个问题将需要进一步的研究。首先,一旦我们具有运行硬件实现,我们将需要集成PUMP硬件和低级软件与主机操作系统和软件工具链(例如,编译器,链接器和加载器)。第二,我们想知道由PUMP提供的机制是否可以用于保护它自己的软件结构。我们相信我们可以通过使用PUMP实现“分区”微策略和使用其保护未命中-处理器代码而代替特定的未命中处理器操作模式。最终,我们在这里看到易于组合策略的正交集合,其中由每个提供的保护完全独立于其他的。

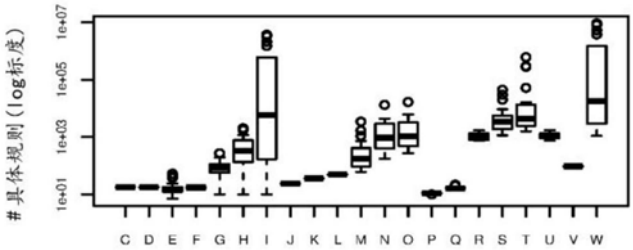
但是策略通常交互:例如,信息-流策略可能需要在由存储器安全策略分配的新鲜区域放置标记。策略组合需要表达和有效率的硬件支持两者的更多研究。

[0849] 

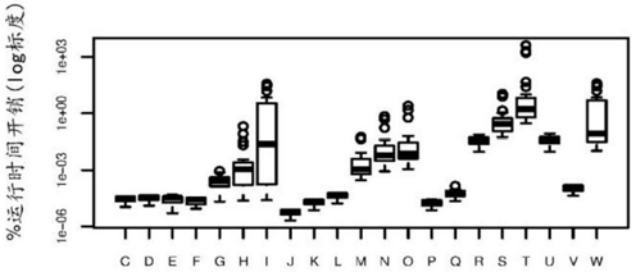
[0850] 图表3初始标记的分布

[0851] 

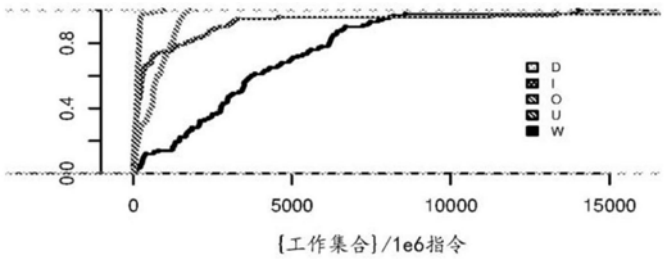
[0852] 图表4最终标记的分布

[0853] 

[0854] 图表5具体规则的分布

[0855] 

[0856] 图表6运行时间开销的分布(高于策略A)

[0857] 

[0858] 图表7用于gcc的工作集合大小的CDF

[0859] 7.参考文献

[0860] [1]Alpha Architecture Handbook.Digital Equipment Corporation,1992.

[0861] [2]M.Abadi,M.Budiu,U.Erlingsson,and J.Ligatti.Control-flow

integrity.In Proc.ACM CCS,pages 340-353,2005.

[0862] [3]P.Akritidis,M.Costa,M.Castro,and S.Hand.Baggy bounds checking:an efficient and backwards-compatible defense against out-of-bounds errors.In Proc.USENIX Security,pages 51-66,2009.

[0863] [4]D.Arora,S.Ravi,A.Raghunathan,and N.K.Jha.Architectural support for run-time validation of program data properties.IEEE Trans.VLSI Sys.,15(5):546-559,May 2007.

[0864] [5]Arvind,R.S.Nikhil,and K.K.Pingali.I-structures:Data structures for parallel computing.In Proc.Wkshp on Graph Reduction(Springer-Verlag LNCS 279),Sept.1986.

[0865] [6]T.H.Austin and C.Flanagan.Efficient purely-dynamic information flow analysis.In Workshop on Programming Languages and Analysis for Security (PLAS),PLAS,pages 1 13-124.ACM,2009.

[0866] [7](authors removed for anonymity).PUMP-A Programmable Unit for Metadata Processing,2014.To appear.

[0867] [8]A.Azevedo de Amorim,N.Collins,A.DeHon,D.Demange,C.Hrij:cu,D.Pichardie,B.C.Pierce,R.Pollack,and A.Tolmach.A verified information-flow architecture.In POPL,pages 165-178.ACM,Jan.2014.

[0868] [9]N.Binkert,B.Beckmann,G.Black,S.K.Reinhardt,A.Saidi,A.Basu,J.Hestness,D.R.Hower,T.Krishna,S.Sardashti,R.Sen,K.Sewell,M.Shoaib,N.Vaish,M.D.Hill,and D.A.Wood.The gem5 simulator.SIGARCH Comput.Archil News,39(2):1-7,Aug.2011.

[0869] [10]T.Boland and P.E.Black.Juliet 1.1 C/C++and Java test suite.Computer,pages 88-90,2012.

[0870] [11]J.Brown and T.F.Knight,Jr.A minimally trusted computing base for dynamically ensuring secure information flow.Technical Report 5,MIT CSAIL,November2001.Aries Memo No.15.

[0871] [12]H.Chen,X.Wu,L.Yuan,B.Zang,P.-c.Yew,and F.T.Chong.From Speculation to Security:Practical and Efficient Information Flow Tracking Using Speculative Hardware.In

[0872] Proc.ISCA,pages 401-412,2008.

[0873] [13]S.Chen,M.Kozuch,T.Strigkos,B.Falsafi,P.B.Gibbons,T.C.Mowry,V.Ramachandran,O.Ruwase,M.P.Ryan,and E.Vlachos.Flexible Hardware Acceleration for Instruction-Grain Program Monitoring.In Proc.ISCA,pages 377-388,2008.

[0874] [14]S.Chen,J.Xu,N.Nakka,Z.Kalbarczyk,and R.Iyer.Defeating memory corruption attacks via pointer taintedness detection.In Proc.IEEE DSN,pages 378-387,2005.

[0875] [15]J.Chung,M.Dalton,H.Kannan,and C.Kozyrakis.Thread-safe dynamic binary translation using transactional memory.In HPCA,pages 279-289.IEEE,

2008.

[0876] [16]J.A.Clause,I.Doudalis,A.Orso,and M.Prvulovic.Effective memory protection using dynamic tainting.In Proc.ASE,pages 284-292.ACM,2007.

[0877] [17]J.R.Crandall and F.T.Chong.Minos:Control data attack prevention orthogonal to memory model.In Proc.IEEE MICRO,pages 221-232,2004.

[0878] [18]M.Dalton,H.Kannan,and C.Kozyrakis.Raksha:a flexible information flow architecture for software security.In Proc.ISCA,pages 482-493,2007.

[0879] [19]D.Y.Deng,D.Lo,G.Malysa,S.Schneider,and G.E.Suh.Flexible and Efficient Instruction-Grained Run-Time Monitoring Using On-Chip Reconfigurable Fabric.In Proc.IEEE MICRO,pages 137-148,2010.

[0880] [20]D.Y.Deng and G.E.Suh.High-performance parallel accelerator for flexible and efficient run-time monitoring.In Proc.IEEE DSN,pages 1-12,2012.

[0881] [21]J.Devietti,C.Blundell,M.M.K.Martin,and S.Zdancewic.HardBound: Architectural support for spatial safety of the C programming language.In S.J.Eggers and J.R.Lams,editors,ASPLOS,pages 103-1 14.ACM,2008.

[0882] [22]E.Goktas,E.Athanasopoulos,H.Bos,and G.Portokalidis.Out Of Control:Overcoming Control-Flow Integrity.In Proc.IEEE S&P,2014.

[0883] [23]C.J.Haley,S.M.Luera,M.D.Schanken,and W.B.Geer.Final evaluation report unisys a series mcp/as release 3.7.Technical Report CSC-EPL-871003, Library No.S-228,515,National Computer Security Center,Fort Meade,MD,August 5 1987.

[0884] [24]D.Hedin and A.Sabelfeld.Information-flow security for a core of JavaScript.In 25th IEEE Computer Security Foundations Symposium(CSF),CSF, pages 3-18.IEEE,2012.

[0885] [25]J.L.Henning.SPEC CPU2006 benchmark descriptions.SIGARCH Comput.Archil News,34(4):1-17,Sept.2006.

[0886] [26]H.Kannan,M.Dalton,and C.Kozyrakis.Decoupling Dynamic Information Flow Tracking with a Dedicated Coprocessor.In Proc.IEEE DSN,pages 105-1 14, 2009.

[0887] [27]MITRE Corp.CWE-122:Heap-based buffer overflow.

[0888] [28]MITRE Corp.CWE-416:Use after free.

[0889] [29]MITRE Corp.CWE-78:Improper neutralization of special elements used in an OS command(OS command injection).

[0890] [30]MITRE Corp.CWE-843:Access of resource using incompatible type (type confusion).

[0891] [31]D.A.Moon.Architecture of the Symbolics 3600.In Proc.ISCA,pages 76-83,Los Alamitos,CA,USA,1985.IEEE Computer Society.

[0892] [32]S.Nagarakatte,M.M.K.Martin,and S.Zdancewic.Hardware-Enforced Comprehensive Memory Safety.IEEE Micro,33(3):38-47,May-June 2013.

[0893] [33]S.Nagarakatte,J.Zhao,M.M.K.Martin,and S.Zdancewic.SoftBound: highly compatible and complete spatial memory safety for C.In Proc.PLDI, pages245-258.ACM,2009.

[0894] [34]E.I.Organick.Computer System Organization:The B5700/B6700 Series.Academic Press,1973.

[0895] [35]M.Ozsoy,D.Ponomarev,N.B.Abu-Ghazaleh,and T.Suit SIFT:a low-overhead dynamic information flow tracking architecture for SMT processors.In Conf.Computing Frontiers,page 37,2011.

[0896] [36]F.Qin,C.Wang,Z.Li,H.Kim,Y.Zhou,and Y.Wu.LIFT:A low-overhead practical information flow tracking system for detecting security attacks.In Proc.IEEE MICRO,pages 135-148,2006.

[0897] [37]A.Russo and A.Sabelfeld.Dynamic vs.static flow-sensitive security analysis.In Proc.CSF,pages 186-199,2010.

[0898] [38]S.Savage,M.Burrows,G.Nelson,P.Sobalvarro,and T.Anderson.Eraser:A dynamic race detector for multi-threaded programs.ACM Trans.Comp.Sys.,15(4), 1997.

[0899] [39]H.Shacham.The Geometry of Innocent Flesh on the Bone:Return-into-libc without Function Calls(on the x86).In Proc.ACM CCS,pages 552-561, Oct.2007.

[0900] [40]D.Stefan,A.Russo,J.C.Mitchell,and D.Mazieres.Flexible dynamic information flow control in Haskell.In 4th Symposium on Haskell,pages 95-106.ACM,2011.

[0901] [41]G.E.Suh,J.W.Lee,D.Zhang,and S.Devadas.Secure Program Execution via Dynamic Information Flow Tracking.In Proc.ASPLOS,pages 85-96,2004.

[0902] [42]L.Szekeres,M.Payer,T.Wei,and D.Song.SoK:Eternal war in memory.In Proc.IEEE S&P,pages 48-62,2013.

[0903] [43]G.S.Taylor,P.N.Hilfinger,J.R.Larus,D.A.Patterson,and B.G.Zorn.Evaluation of the SPUR lisp architecture.In Proc.ISCA,pages 444-452, 1986.

[0904] [44]N.Vachharajani,M.J.Bridges,J.Chang,R.Rangan,G.Ottoni,J.A.Blome, G.A.Reis,M.Vachharajani,and D.I.August.RIFLE:An architectural framework for user-centric information-flow security.In Proc.IEEE MICRO,2004.

[0905] [45]G.Venkataramani,I.Doudalis,Y.Solihin,and M.Prvulovic.FlexiTaint:A programmable accelerator for dynamic taint propagation.In Proc.HPCA,pages 173-184,Feb.2008.

[0906] [46]R.Wahbe,S.Lucco,T.E.Anderson,and S.L.Graham.Efficient software-based fault isolation.In SOSP,pages 203-216,1993.

[0907] [47]E.Witchel,J.Cates,and K.Asanovi c.ondrian memory protection.In Proc.ASPLOS,pages 304-316,New York,NY,USA,2002.ACM.

[0908] [48]W.Xu,S.Bhatkar,and R.Sekar.Taint-enhanced policy enforcement:a practical approach to defeat a wide range of attacks.In Proc.USENIX Security,Berkeley,CA,USA,2006.

[0909] [49]H.Yin,D.X.Song,M.Egele,C.Kruegel,and E.Kirda.Panorama:capturing system-wide information flow for malware detection and analysis.In Proc.CCS,pages 1 16-127.ACM,2007.

[0910] [50]N.Zeldovich,H.Kannan,M.Dalton,and C.Kozyrakis.Hardware enforcement of application security policies using tagged memory.In Proceedings of the 8th USENIX conference on Operating systems design and implementation,OSDI,pages 225-240.USENIX Association,2008.

[0911] [51]C.Zhang,T.Wei,Z.Chen,L.Duan,L.Szekeres,S.McCamant,D.Song,and W.Zou.Practical Control Flow Integrity&Randomization for Binary Executables.In Proc.IEEE S&P,2013.

[0912] [52]P.Zhou,R.Teodorescu,and Y.Zhou.HARD:Hardware-assisted lockset-based race recording.In Proc.HPCA,2007.

[0913] 8.符号规则

[0914] 8.1原语类型

[0915] nop,ubranch:

[0916] (-,insn,-,-,-)→(-,-) (1)

[0917] ar2s1d:(-,insn,other,other,-)→(-,other) (2)

[0918] ar2s1d:(-,insn,addr,other,-)→(-,addr) (3)

[0919] ar2s1d:(-,insn,other,addr,-)→(-,addr) (4)

[0920] ar2s1d:(-,insn,addr,addr,-)→(-,other) (5)

[0921] ar1s1d:(-,insn,other,--)→(-,other) (6)

[0922] ar1s1d:(-,insn,addr,-,-)→(-,addr) (7)

[0923] ar1d,flags:

[0924] (-,insn,-,-,-)→(-,other) (8)

[0925] cbranch:(-,insn,other,other,-)→(-,-) (9)

[0926] ijump,return:

[0927] (-,insn,addr,-,-)→(-,-) (10)

[0928] dcall,icall:

[0929] (-insn,addr,-,-)→(-,addr) (11)

[0930] load:(-,insn,addr,-,t)→(-,t)if t≠insn (12)

[0931] store:(-,insn,t,addr,-)→(-,t)if t≠insn (13)

[0932] move:(-,insn,other,-,-)→(-,other) (14)

[0933] move:(-,insn,addr,-,-)→(-,addr) (15)

[0934] 用于校验返回地址的替代规则：

[0935] ijump:(-,insn,addr,-,-)→(-,-) (10)

[0936] return:(-,insn,retaddr,-,-)→(-,-) (10)

[0937] dcall,icall:

[0938] (-,insn,addr,-,-)→(-,retaddr) (11)

[0939] 8.2存储器安全

[0940] 具有N＝2^{64}-k的N-着色用于完全存储器安全。我们将颜色写为c,且使用它们标记到堆的指针。我们假定不同于颜色的特殊标记⊥,且其用于标记不是到堆的指针的全部数据。用于寄存器的标记是颜色或者⊥(写入的t)。用于存储器的标记是颜色的对和颜色或者⊥(写入的(c1,t2))或者F(未分配的)的对,堆最初全部标记F。最终关于指令的标记从以下集合得到:

{tmalloc, tmallocinit, tfreeinit, tsomething else }

nop, cbranch, ubranch, ijump, return :

(-, -, -, -, -)→(-, -) (1)

ar2s1d : (-, -, ⊥, ⊥, -)→(-, ⊥) (2)

ar2s1d : (-, -, c, ⊥, -)→(-, c) (3)

ar2s1d : (-, -, ⊥, c, -)→(-, c) (4)

ar2s1d : (-, -, c, c, -)→(-, ⊥) (5)

ar1s1d : (-, -, t, -, -)→(-, t) (6)

[0941] ar1d, dcall, icall, flags :

(-, -, -, -, -)→(-, ⊥) (7)

load : (-, -, c1, -, (c2, t2))→(-, t2) if c1=c2 (8)

store : (-, ci, t1, c2, (c3, t3)) (9)

→ (-, (c3, t1)) if c2=c3∧ci/∈{$t_{mallocinit}$, $t_{freeinit}$}

store: (-, $t_{mallocinit}$, t1 , c2, F)→(-, (c2, t1)) (10)

store: (-, $t_{freeinit}$, t1 , c2, (c3, t4))→(-, F) (11)

move: (-, t_{malloc}, t,-,-) $\xrightarrow{1}$(-, tnewtag) (12)

move: (-, $\overline{\text{tmalloc}}$, t, -, -)→(-, t) (13)

```
primitive_malloc=malloc;
malloc (int size) {
  void *p=primitive_malloc(size);  // orig ptr
  void *tp;  // 标记 ptr
  void *tmp;  // 标记ptr 到各个字
  asm: malloc move r1=p, r2=tp // 分配新鲜标记
  tmp=tp;
  for (int i=0;  i<size;  i++) {
    // 设置新区域中的字上的区域标记asm mallocinit
    store r1=0, r2=tmp
    tmp++;
  }
  return (tp);
}
primitive_free=free;
free (void *p) {
  size =size(p);  // 指针区域大小
  void *tmp=base(p);  // 指针区域的基数
  for (int i=0; i<size; i++) {
    // 设置释放区域中的字上的区域标记
    asm freeinit store r1=0, r2=tmp
    tmp++;
    }
  return;
}
```

[0943] 8.3CFI

[0944] 8.3.1CFI-1ID[2]

[0945] 我们使用写为集合∅和{f}的2个标记。标记{f}.用于标记所有间接控制流,以及所有其潜在目的地。标记∅用于每个其他的。

[0946] return,ijump,icall:

[0947] (-,{f},-,-,-)→({f},-) (1)

[0948] $\overline{\text{return,ijump,icall}}$:

$$(pc,\ ci,\ -,\ -,\ -)\rightarrow(\varnothing,\ -)\ \text{if}\ pc\sqsubseteq ci \quad (2)$$

[0949] 8.3.2CFI-2ID[2]

[0950] 在该策略中,r用于标明返回及其潜在目标,且c用于间接调用和跳转及其潜在目标。因为这两个情况可能重叠,我们使用写为以下集合的4个标记:ø,{r},{c}和{r,c}.

[0951] return : $(pc,\ ci,\ -,\ -,\ -)$ (1)

$$\rightarrow(\{r\},\ -)\ \text{if}\ r\in ci,\ pc\sqsubseteq ci$$

[0952] ijump, icall:

$$(pc,\ ci,\ -,\ -,\ -)\rightarrow(\{c\},\ -)\ \text{if}\ c\in ci,\ pc\sqsubseteq ci \quad (2)$$

[0953] $\overline{\text{return,ijump,icall}}$:

$$(pc,\ ci,\ -,\ -,\ -)\rightarrow(\varnothing,\ -)\ \text{if}\ pc\sqsubseteq ci \quad (3)$$

[0954] 8.3.3CCFIR[51]

[0955] r是返回id,c是调用id,p是返回-到-特权-代码-id。假定写为以下集合的6个标记:ø,{r},{p},{c},{r,c}和{p,c}。

[0956] return : $(pc,\ ci,\ -,\ -,\ -)$ (1)

$$\rightarrow(\{r\},\ -)\ \text{if}\ r\in ci,\ pc\sqsubseteq ci$$

[0957] return : $(pc,\ ci,\ -,\ -,\ -)$ (2)

$$\rightarrow(\{p\},\ -)\ \text{if}\ p\in ci,\ pc\sqsubseteq ci$$

[0958] ijump, icall:

$$(pc,\ ci,\ -,\ -,\ -)\rightarrow(\{c\},\ -)\ \text{if}\ c\in ci,\ pc\sqsubseteq ci \quad (3)$$

[0959] $\overline{\text{return,ijump,icall}}$:

$$(pc,\ ci,\ -,\ -,\ -)\rightarrow(\varnothing,\ -)\ \text{if}\ pc\sqsubseteq ci \quad (4)$$

[0960] 8.3.4CFI-ROP

[0961] 我们假定允许的控制流图x,包括成对的返回ID和可能的目的地ID。我们写入ID作为以下的ci或者pc。标记是有效ID或者⊥。

[0962] return:$(\perp,ci,-,-,-)\rightarrow(ci,-)$ (1')

[0963] $\overline{\text{return}}$: $(pc,\ ci,\ -,\ -,\ -)\rightarrow(\perp,\ -)\ \text{if}\ (pc,\ ci)\in\chi$ (2')

[0964] $\overline{\text{return}}$: $(\perp,\ -,\ -,\ -,\ -)\rightarrow(\perp,\ -)$ (3')

[0965] return:$(pc,ci,-,-,-)\rightarrow(ci,-)\text{if}(pc,ci)\in x$ (4')

[0966] 8.3.5CFI-JOP

[0967] 假定允许的控制流图,x。

[0968] ijump,icall:

[0969] $(\perp,ci,-,-,-)\rightarrow(ci,-)$ (1)

[0970] ijump,icall:

[0971] (pc,ci,-,-,-)→(ci,-)if(pc,ci)∈x (2)

[0972] $\overline{\text{ijump,icall}}$:

$$(\perp, -, -, -, -) \rightarrow (\perp, -) \qquad (3)$$

[0973] $\overline{\text{ijump,icall}}$:

$$(pc, ci, -, -, -) \rightarrow (\perp, -) \text{ if } (pc, ci) \in \chi \qquad (4)$$

[0974] 8.3.6完全-CFI

[0975] 我们假定允许的控制流图x。

[0976] return,ijump,icall:

[0977] (⊥,ci,-,-,-)→(ci,-) (1)

[0978] return,ijump,icall:

[0979] (pc,ci,-,-,-)→(ci,-)if(pc,ci)∈x (2)

[0980] $\overline{\text{return,ijump,icall}}$:

$$(\perp, -, -, -, -) \rightarrow (\perp, -) \qquad (3)$$

[0981] $\overline{\text{return,ijump,icall}}$:

$$(pc, ci, -, -, -) \rightarrow (\perp, -) \text{ if } (pc, ci) \in \chi \qquad (4)$$

[0982] 8.4污点跟踪

[0983] nop,cbranch,ubranch,ijump,return:

[0984] (-,-,-,-,-)→(-,-) (1)

[0985] ar2s1d:(-,ci,op1,op2,-)→(-,ci∪op1∪op2) (2)

[0986] ar1s1d:(-,ci,op1,-,-)→(-,ci∪op1) (3)

[0987] ar1d,dcall,icall,flags:

[0988] (-,ci,-,-,-)→(-,ci) (4)

[0989] load:(-,ci,op1,-,mr)→(-,ci∪op1∪mr) (5)

[0990] store:(-,ci,op1,op2,-)→(-,ci∪op1∪op2) (6)

[0991] $\text{move} : (-, t_{taint}, -, -, -) \xrightarrow{1} (-, t_{newtag})$ (7)

[0992] move:(-,ci≠t_{taint},op1,-,-)→(-,ci∪op1) (8)

[0993] 8.5子字操作

[0994] 我们在我们的实验中使用的以上规则不说明子字操作。为适当地支持子字操作，我们将需要将加载和存储操作组分解为用于字操作的两个操作组(wload和wstore)和两个操作组字节操作(bload和bstore)。

[0995] 用于明确地谈论加载或者存储的全部策略的规则将需要改变(简单类型,存储器安全和污点跟踪)。这里是简单类型策略(的无retaddr变型)将怎样改变(w操作组对应于先前规则)：

[0996] wload:(-,insn,addr,-,other)→(-,other) (1)

[0997] wload:(-,insn,addr,-,addr)→(-,addr) (2)

[0998] wstore:(-,insn,other,addr,-)→(-,other) (3)

[0999] wstore:(-,insn,addr,addr,-)→(-,addr) (4)

[1000] bload:(-,insn,addr,-,other)→(-,other) (5)

[1001] bload:(-,insn,addr,-,addr)→(-,other) (6)

[1002] bstore:(-,insn,other,addr,-)→(-,other) (7)

[1003] bstore:(-,insn,addr,addr,-)→(-,other) (8)

[1004] 这里是用于存储器安全的b规则：

[1005] $\text{bload: } (-,\ -,\ c1,\ -,\ (c2,\ c_3^{\perp})) \rightarrow (-,\ \perp) \text{ if } c1=c2 \quad (1)$

[1006] $\text{bstore: } (-,\ ci,\ c_1^{\perp},\ c2,\ (c3,\ c_4^{\perp})) \quad (2)$

$\rightarrow (-,\ (c3,\ \perp)) \text{ if } c2=c3 \wedge ci \notin \{t_{mallocinit},\ t_{freeinit}\}$

[1007] 这里是用于污点跟踪的bstore规则：

[1008] $\text{bstore: } (-,\ ci,\ op1,\ op2,\ mr) \quad (1)$

$\rightarrow (-,\ ci \cup op1 \cup op2 \cup mr)$

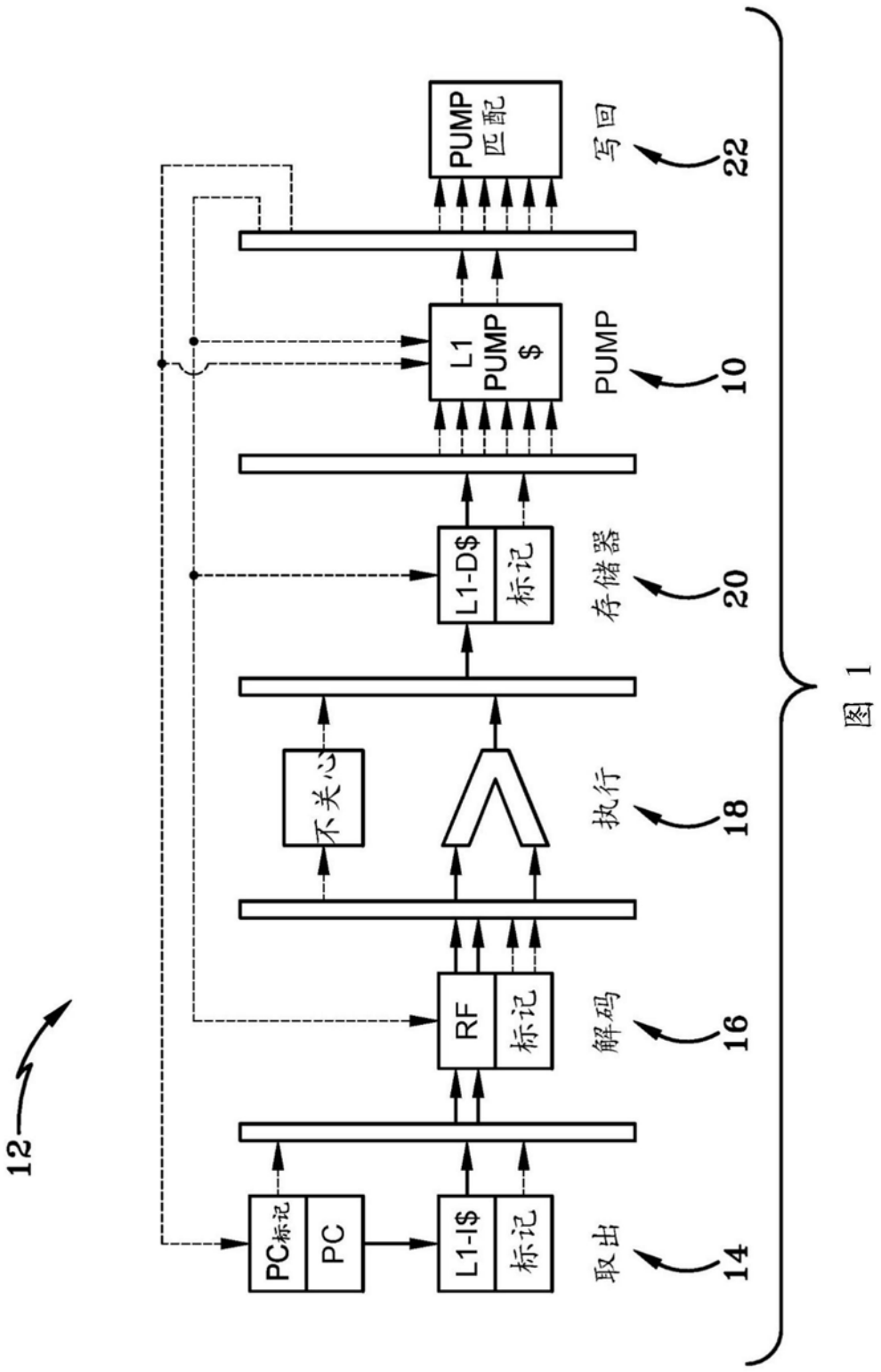

12
PC标记
PC
L1-I$
标记
取出
14
RF
标记
解码
16
不关心
执行
18
L1-D$
标记
存储器
20
L1
PUMP
$
PUMP
10
PUMP
匹配
写回
22
图 1

图1

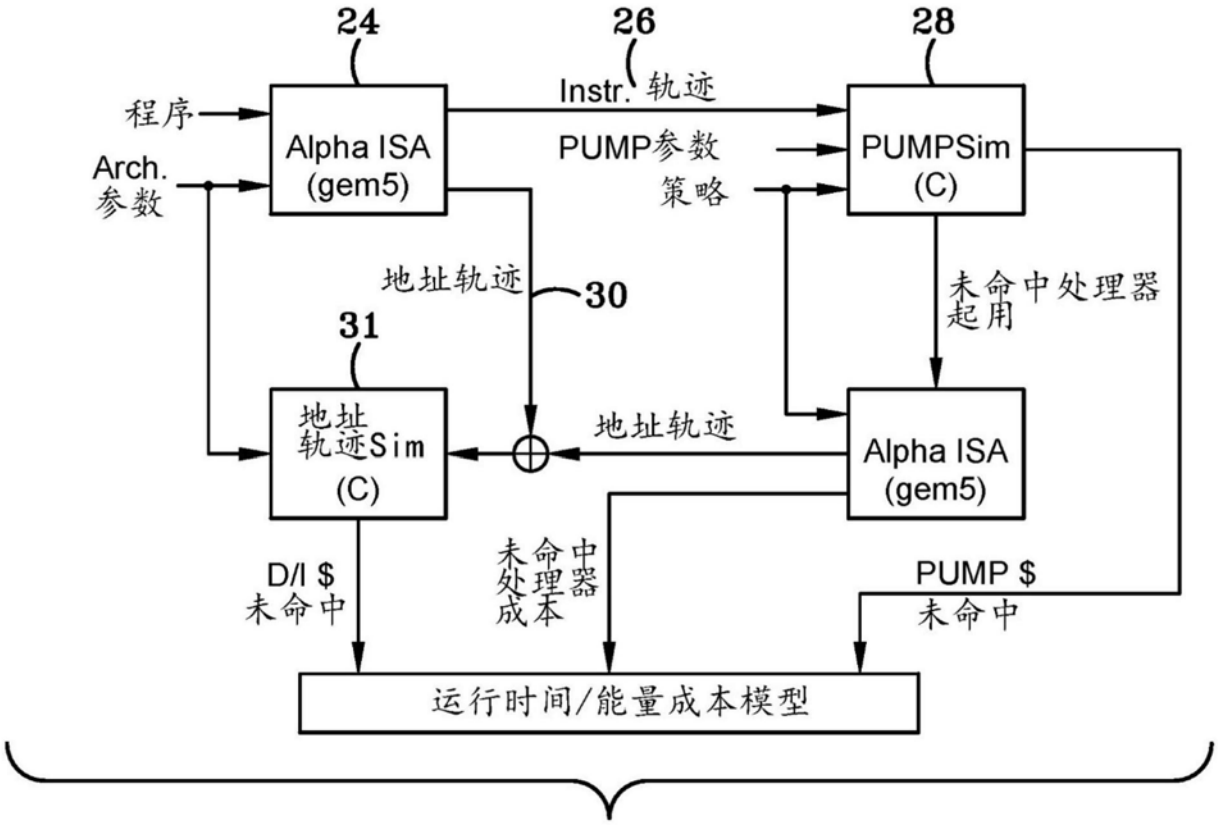

24
26
28
程序
Alpha ISA
(gem5)
Instr. 轨迹
PUMP参数
策略
PUMPSim
(C)
Arch.
参数
地址轨迹
30
未命中处理器
起用
31
地址
轨迹Sim
(C)
地址轨迹
Alpha ISA
(gem5)
D/I $
未命中
未命中
处理器
成本
PUMP $
未命中
运行时间/能量成本模型

图 2

图2

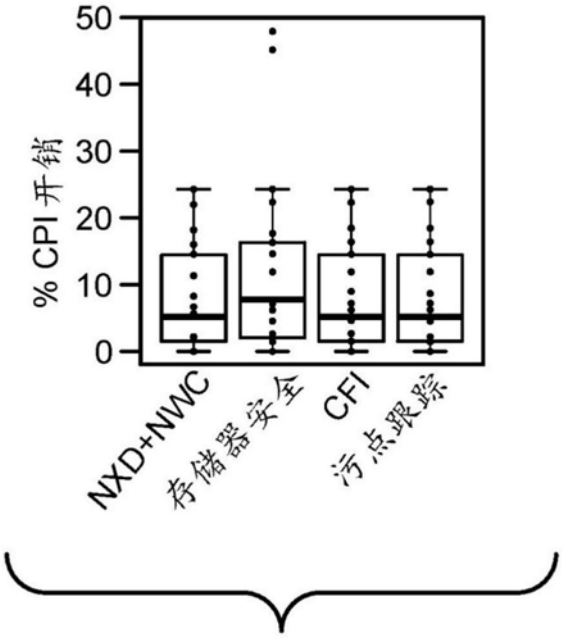

50
40
30
20
10
0
% CPI 开销
NXD+NWC
存储器安全
CFI
污点跟踪

图 3A

图3A

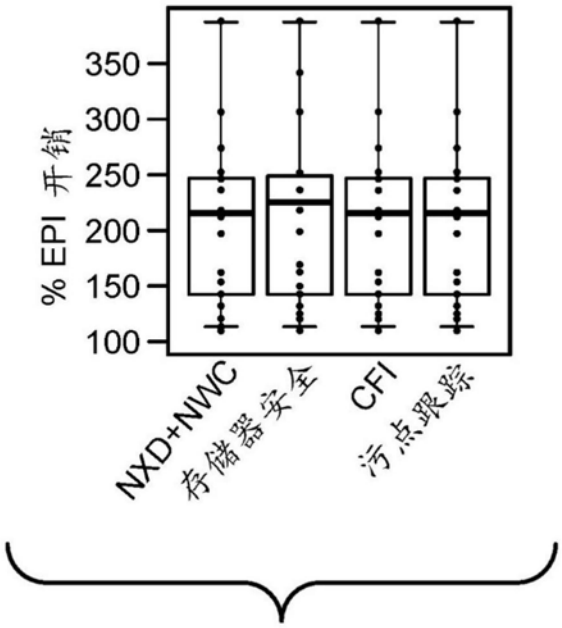


图 3B

图3B

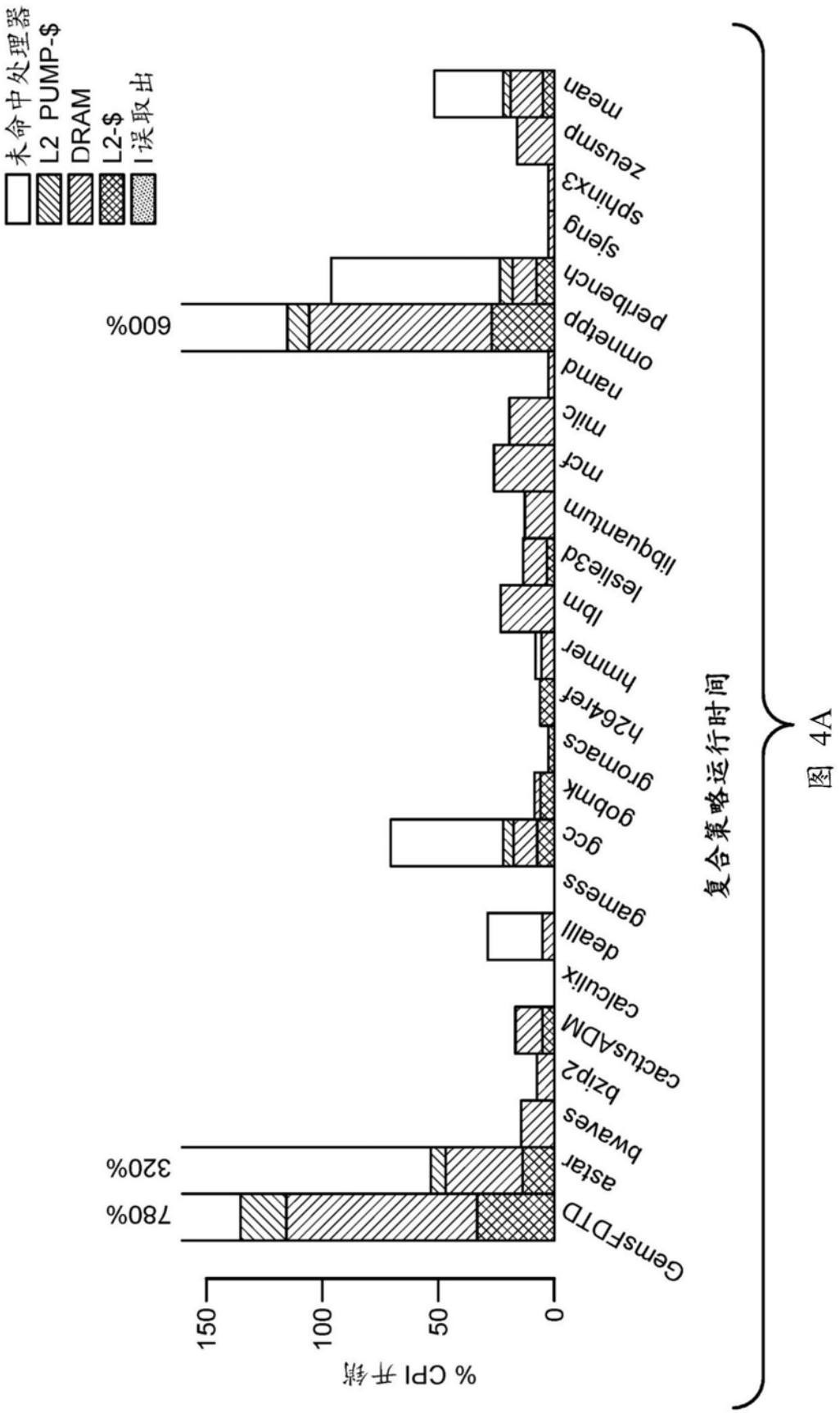

未命中处理器
L2 PUMP-$
DRAM
L2-$
I误取出
mean
zeusmp
sphinx3
sjeng
perlbench
omnetpp
namd
milc
mcf
libquantum
leslie3d
lbm
hmmer
h264ref
gromacs
gobmk
gcc
gamess
dealII
calculix
cactusADM
bzip2
bwaves
astar
GemsFDTD
600%
780%
320%
150
100
50
0
% CPI 开销
复合策略运行时间

图 4A

图4A

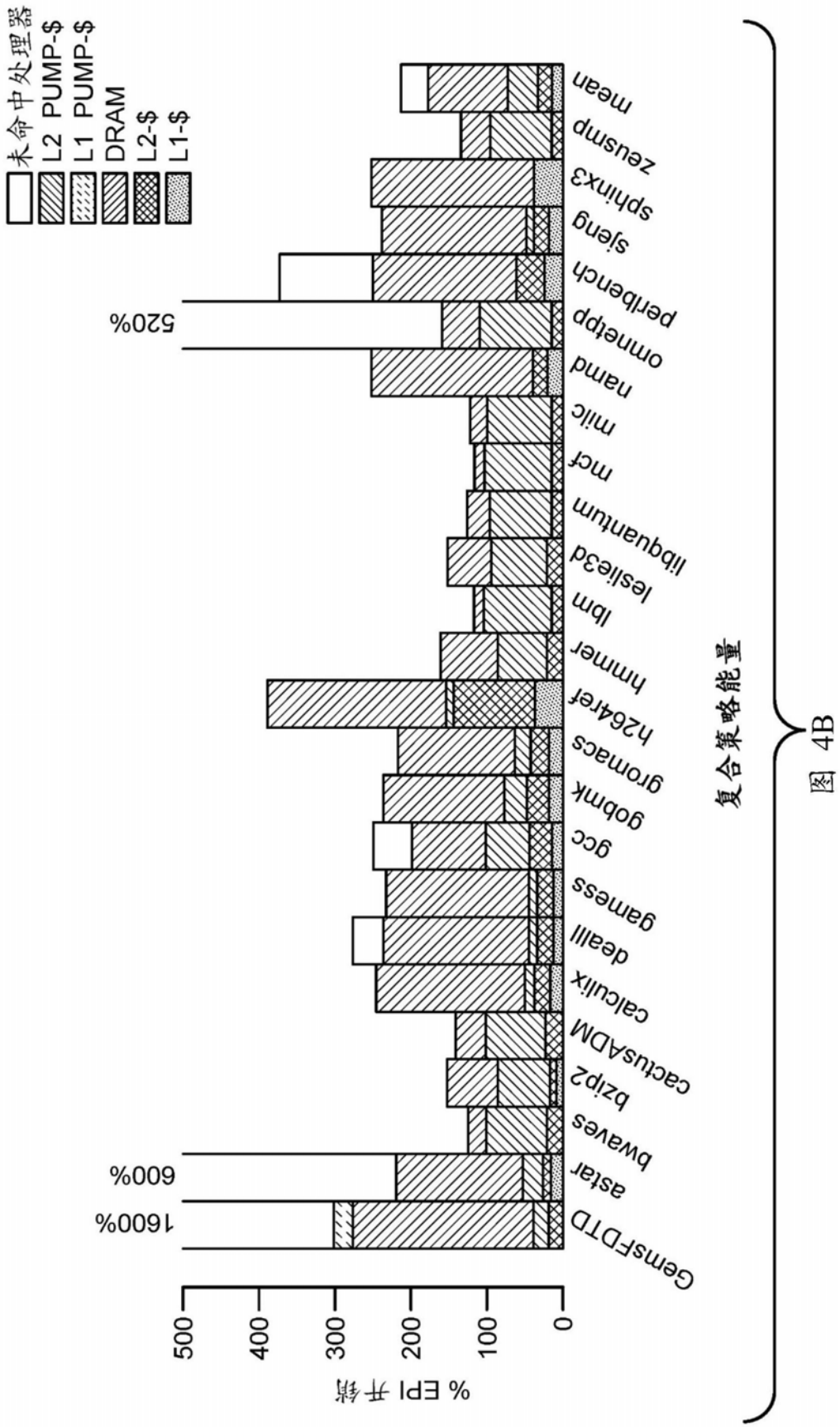

未命中处理器
L2 PUMP-$
L1 PUMP-$
DRAM
L2-$
L1-$
% EPI 开销
500
400
300
200
100
0
1600%
600%
520%
GemsFDTD
astar
bwaves
bzip2
cactusADM
calculix
dealII
gamess
gcc
gobmk
gromacs
h264ref
hmmer
lbm
leslie3d
libquantum
mcf
milc
namd
omnetpp
perlbench
sjeng
sphinx3
zeusmp
mean
复合策略能量

图 4B

图4B

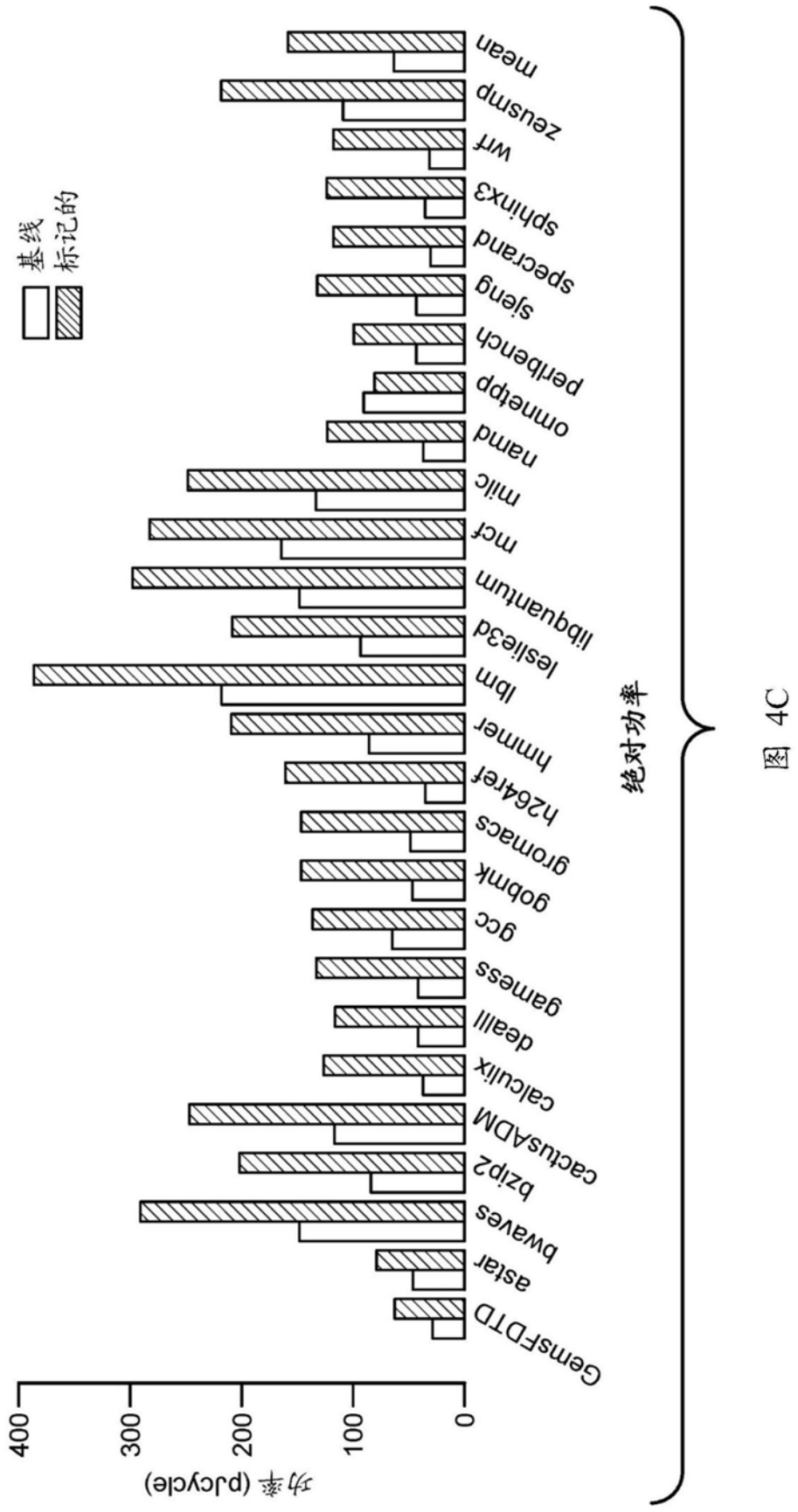

基线
标记的
GemsFDTD
astar
bwaves
bzip2
cactusADM
calculix
dealII
gamess
gcc
gobmk
gromacs
h264ref
hmmer
lbm
leslie3d
libquantum
mcf
milc
namd
omnetpp
perlbench
sjeng
specrand
sphinx3
wrf
zeusmp
mean
绝对功率
功率 (pJ/cycle)
0
100
200
300
400

图 4C

图4C

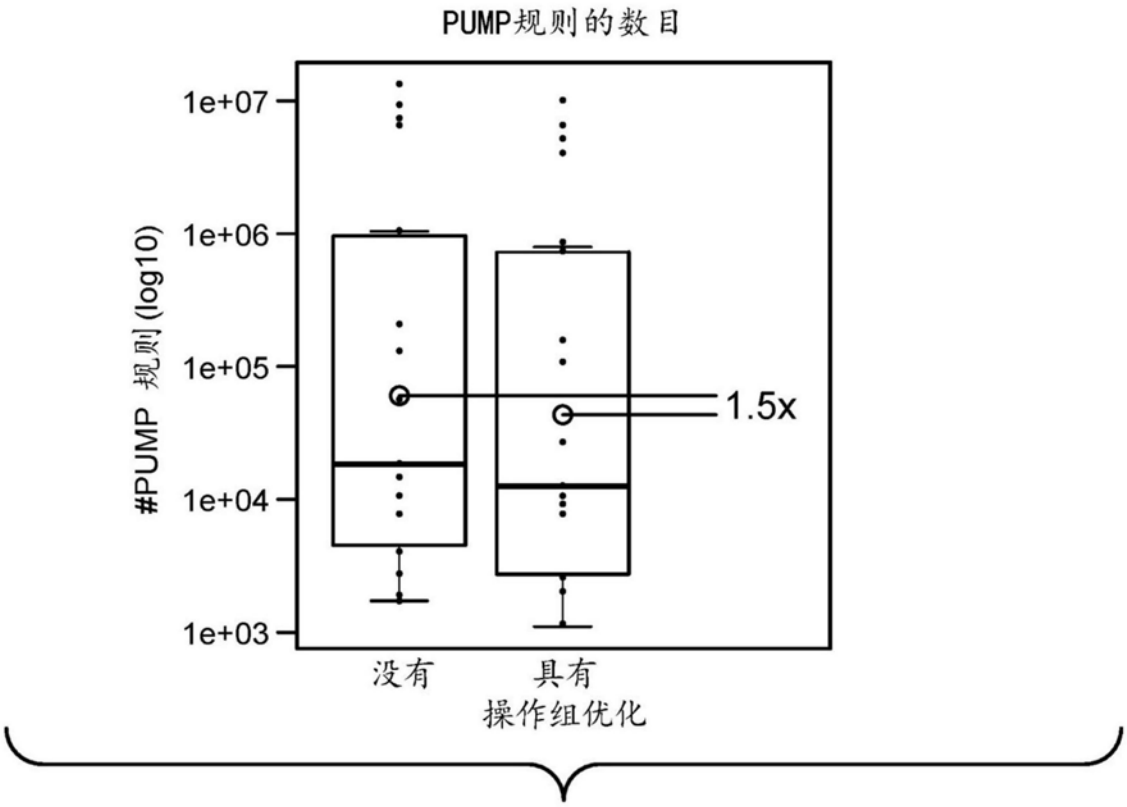

PUMP规则的数目
#PUMP 规则(log10)
1e+07
1e+06
1e+05
1e+04
1e+03
1.5x
没有
具有
操作组优化

图 5A

图5A

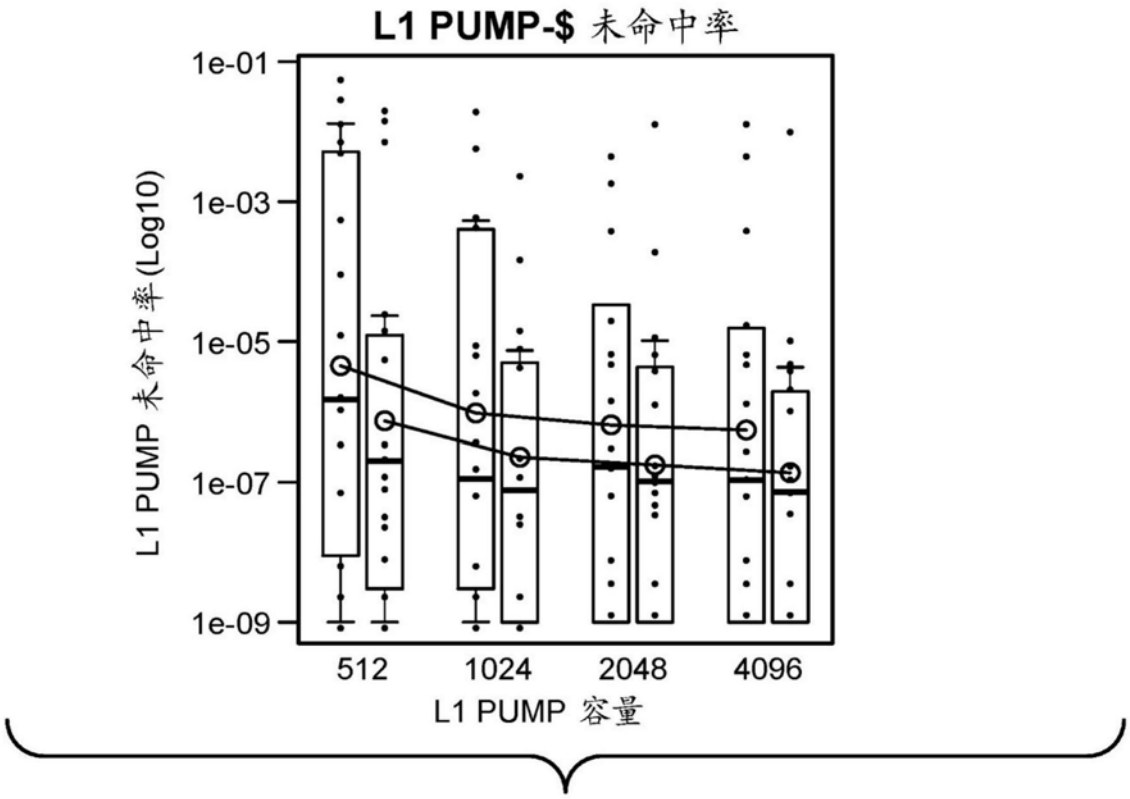


图 5B

图5B

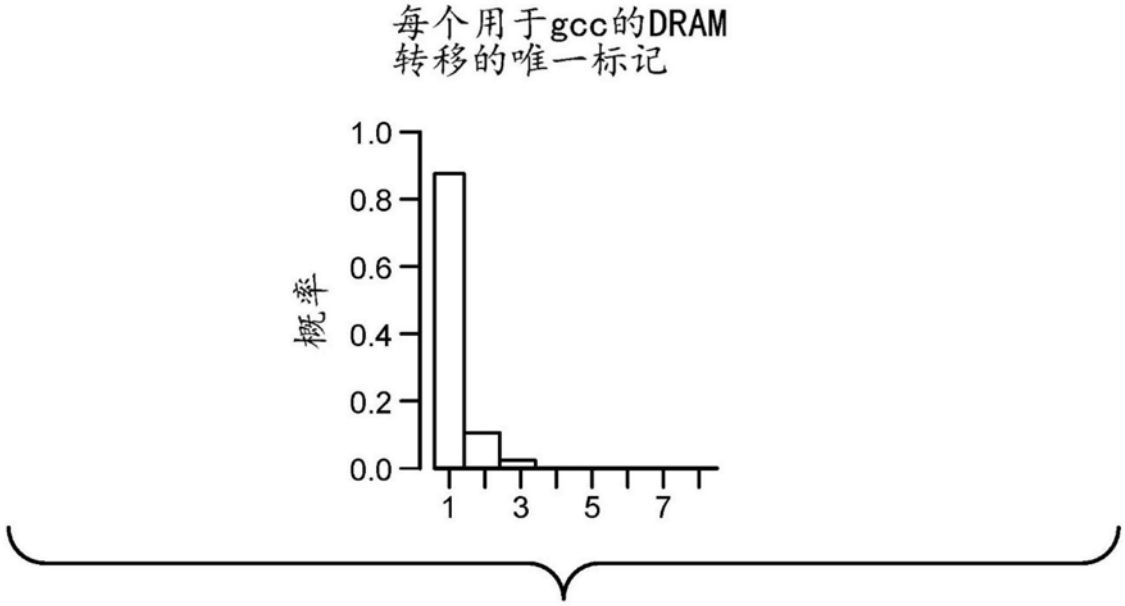


图 6A

图6A

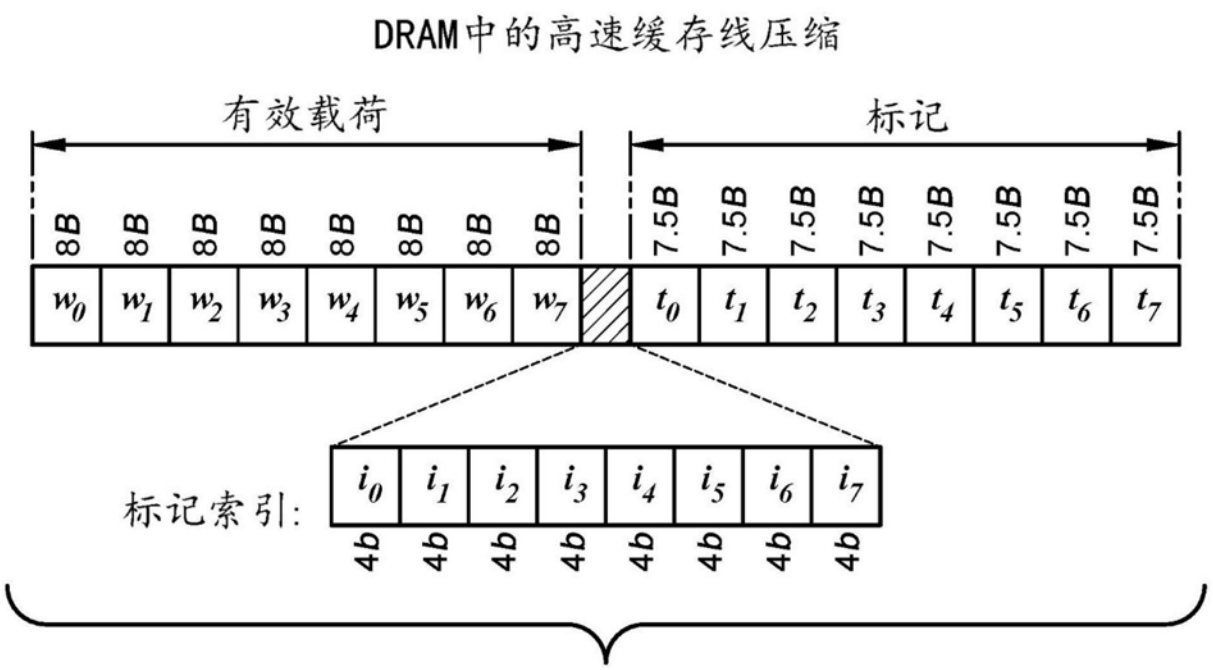


图 6B

图6B

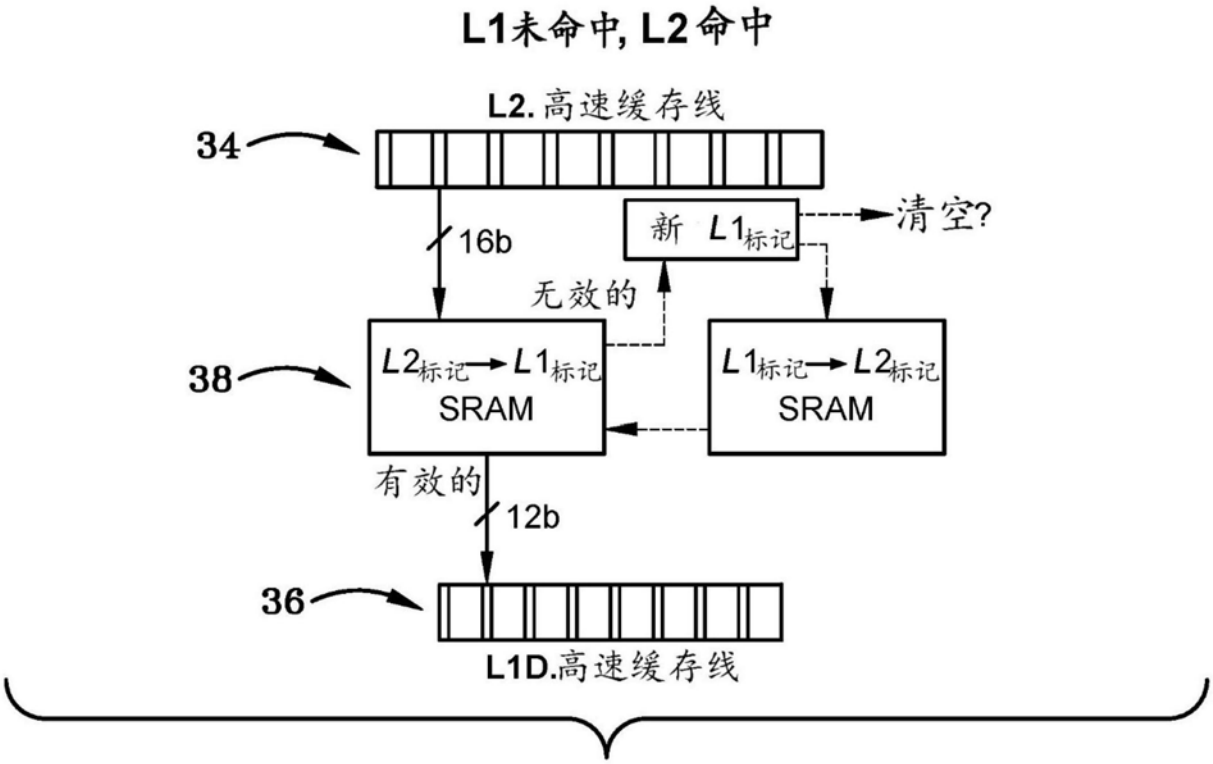


图 7A

图7A

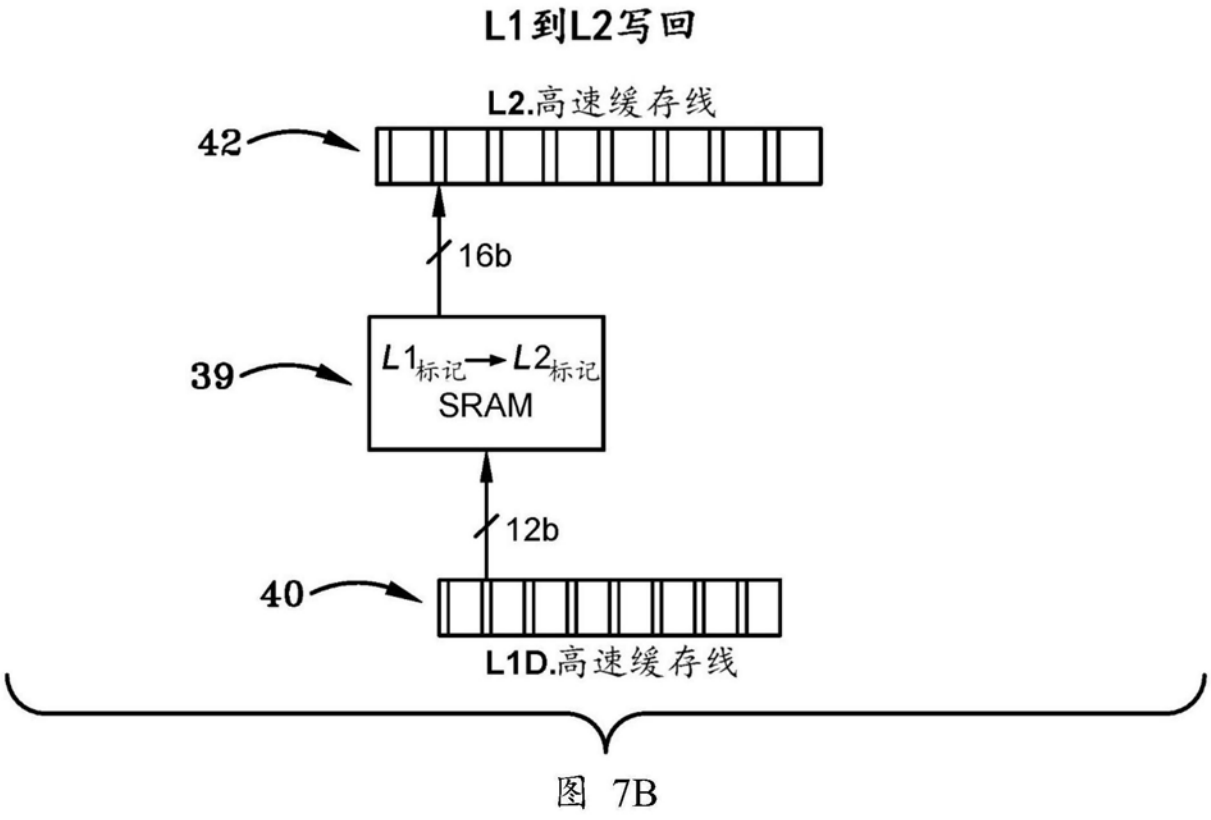


图 7B

图7B

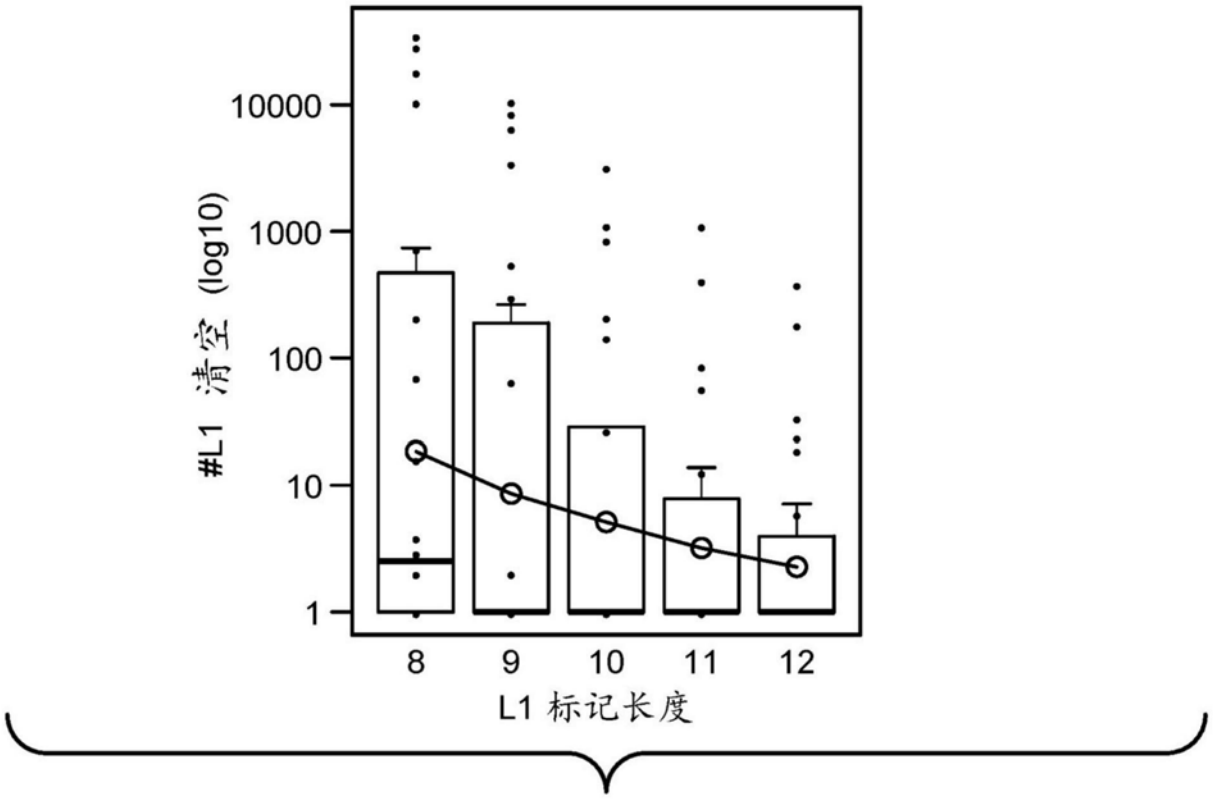


图 8A

图8A

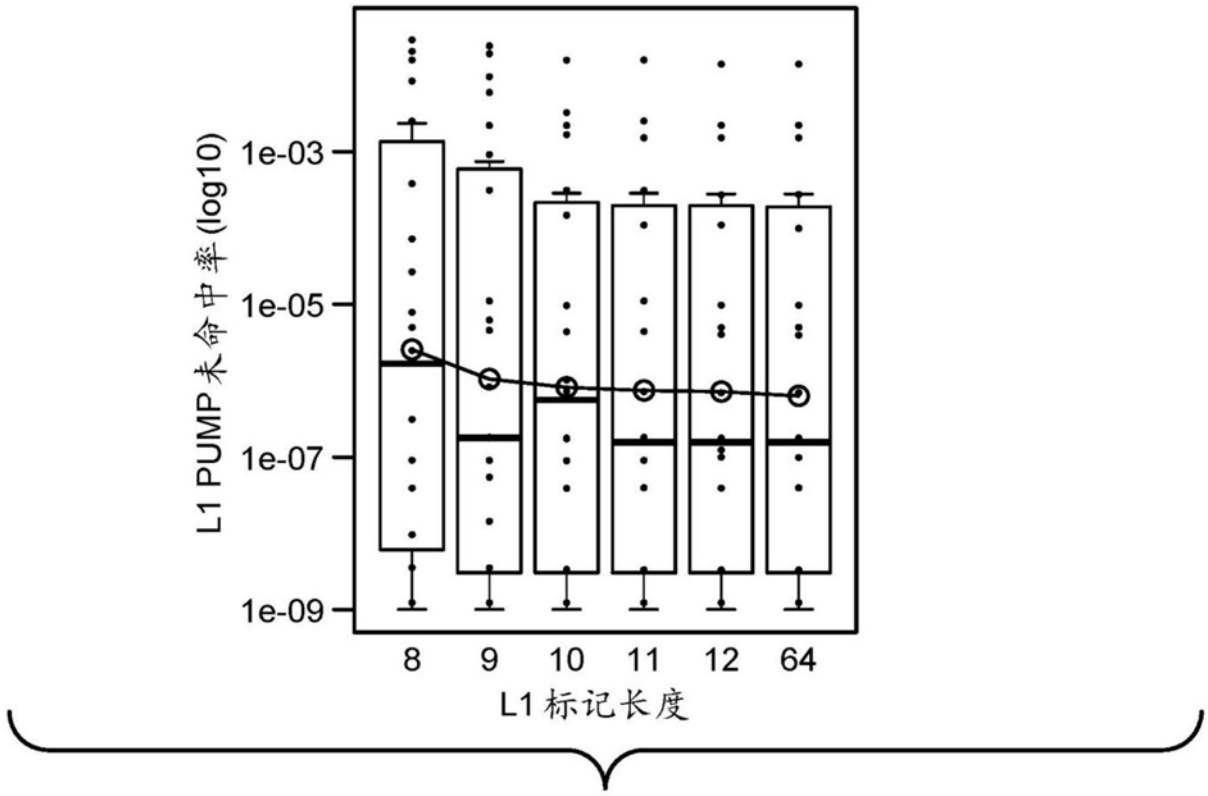


图 8B

图8B

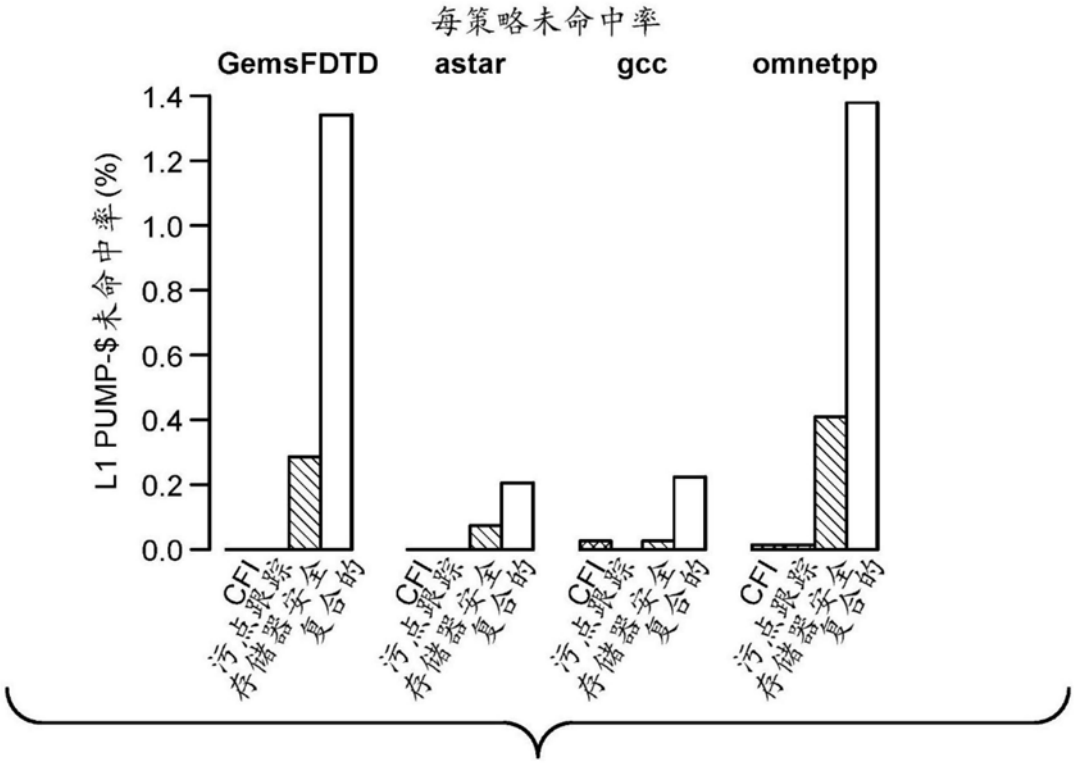

每策略未命中率
GemsFDTD
astar
gcc
omnetpp
L1 PUMP-$未命中率(%)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
CFI
污点跟踪
存储器安全的
复合的

图 9A

图9A

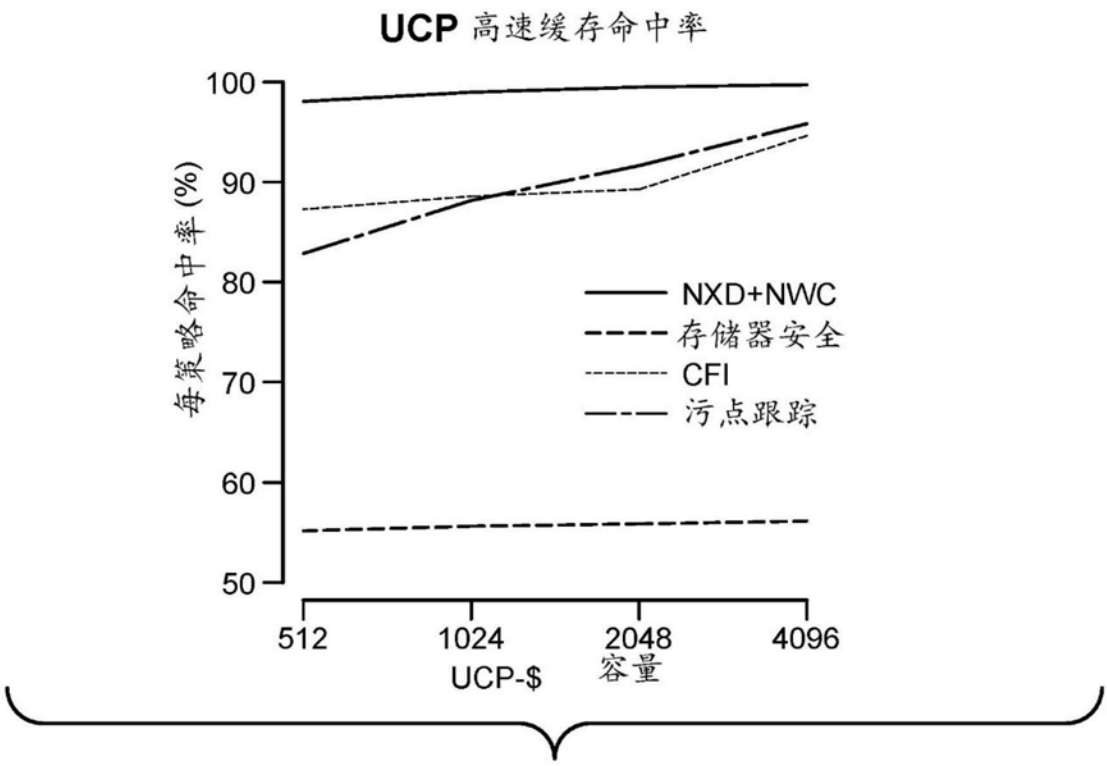

UCP 高速缓存命中率
每策略命中率 (%)
100
90
80
70
60
50
512
1024
2048
4096
UCP-$
容量
NXD+NWC
存储器安全
CFI
污点跟踪

图 9B

图9B

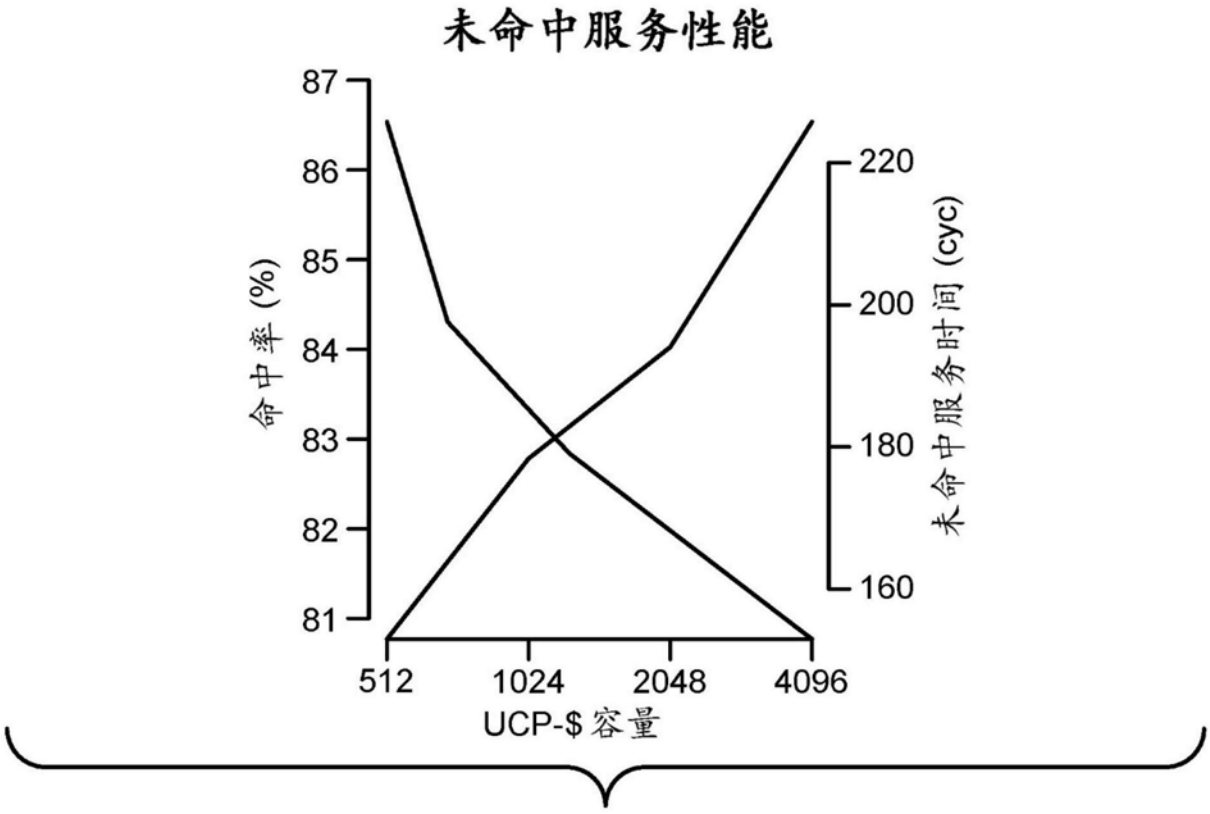

未命中服务性能
87
86
85
84
83
82
81
命中率 (%)
220
200
180
160
未命中服务时间 (cyc)
512
1024
2048
4096
UCP-$容量

图 9C

图9C

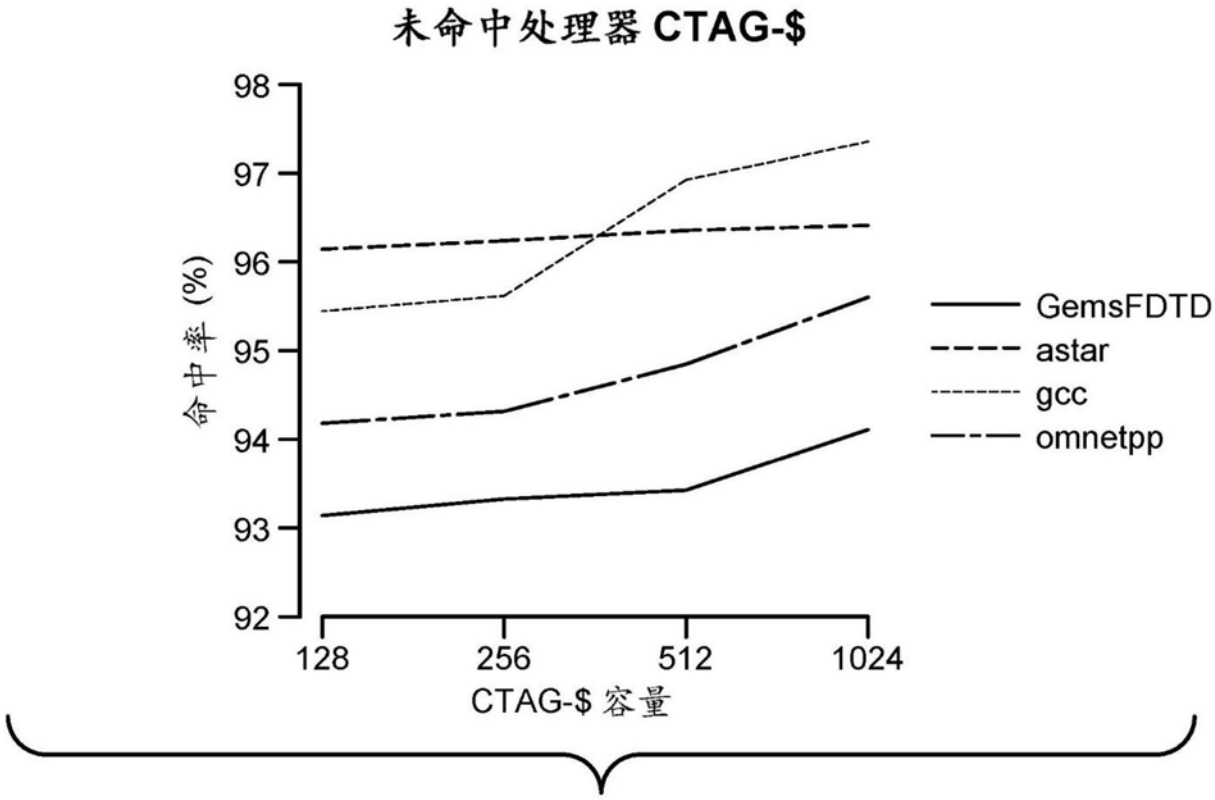

未命中处理器 CTAG-$
98
97
96
95
94
93
92
命中率 (%)
128
256
512
1024
CTAG-$ 容量
GemsFDTD
astar
gcc
omnetpp

图 9D

图9D

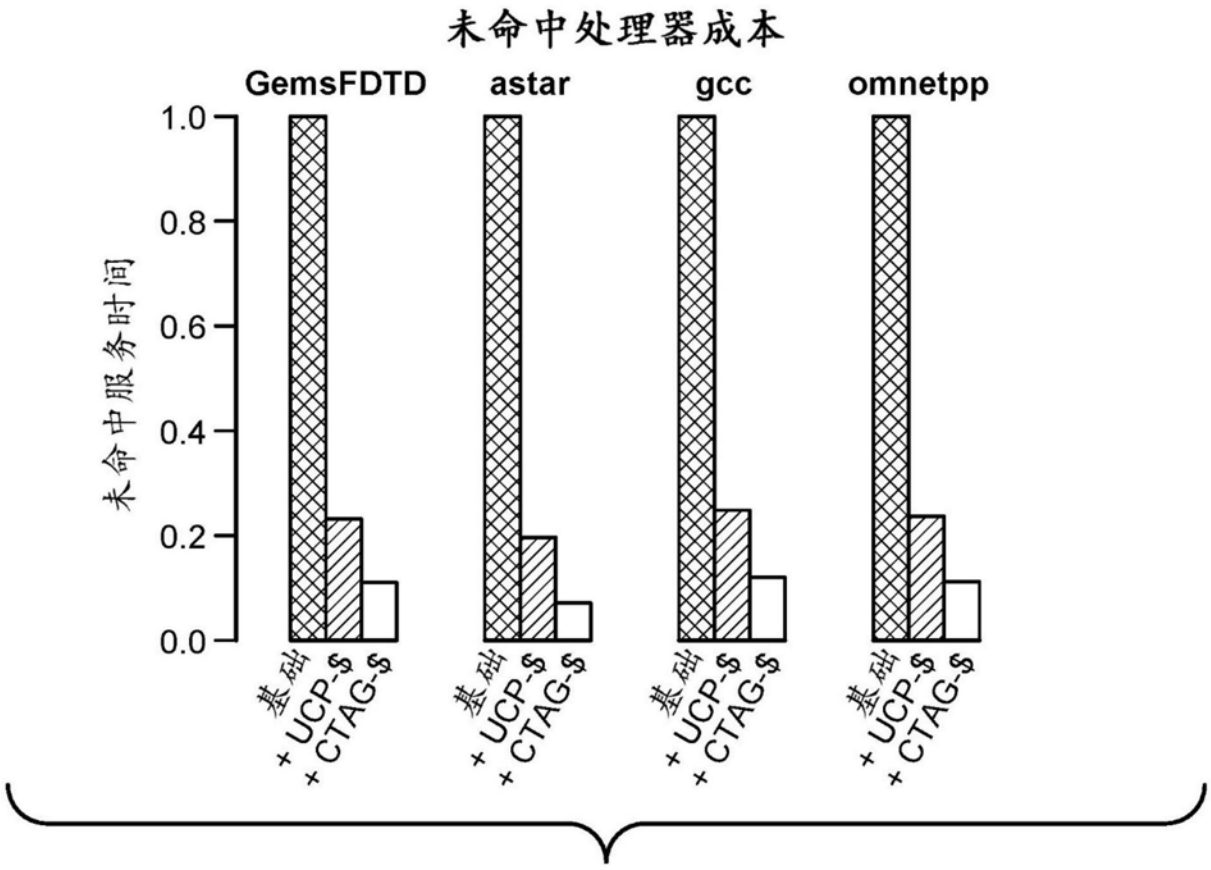

未命中处理器成本
GemsFDTD
astar
gcc
omnetpp
未命中服务时间
1.0
0.8
0.6
0.4
0.2
0.0
基础
+ UCP-$
+ CTAG-$

图 9E

图9E

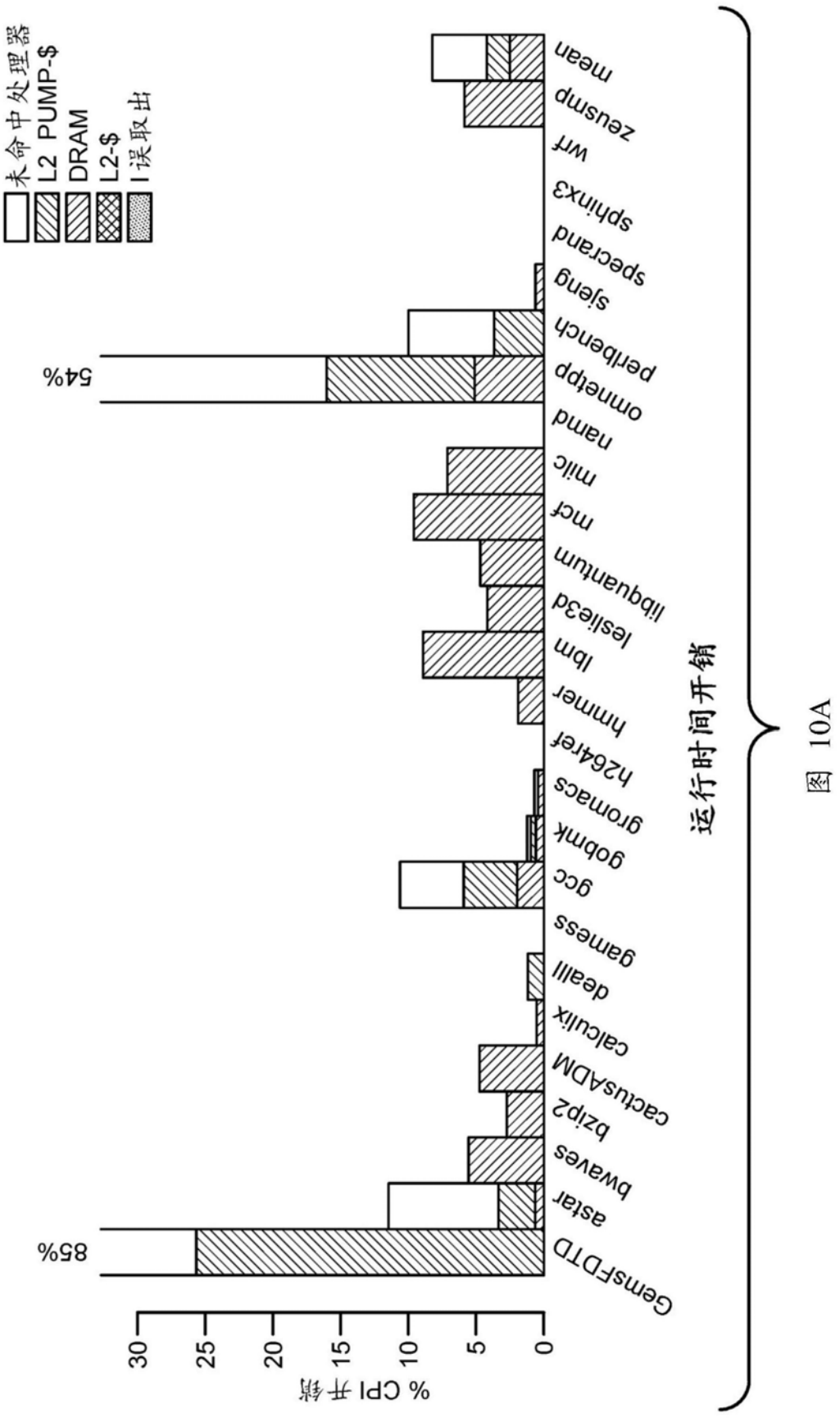

未命中处理器
L2 PUMP-$
DRAM
L2-$
I误取出
% CPI 开销
30
25
20
15
10
5
0
85%
54%
GemsFDTD
astar
bwaves
bzip2
cactusADM
calculix
dealII
gamess
gcc
gobmk
gromacs
h264ref
hmmer
lbm
leslie3d
libquantum
mcf
milc
namd
omnetpp
perlbench
sjeng
specrand
sphinx3
wrf
zeusmp
mean
运行时间开销

图 10A

图10A

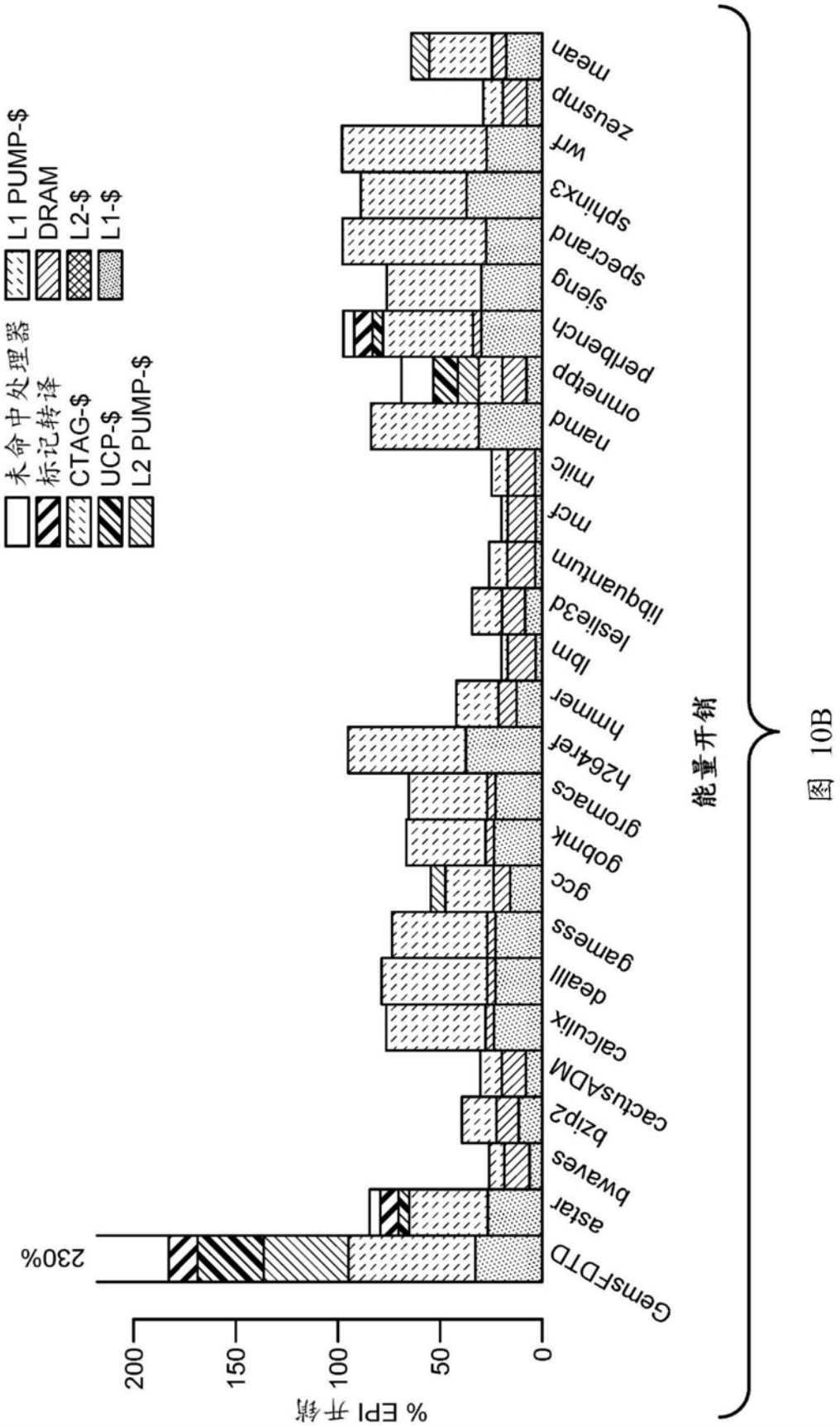

未命中处理器
标记转译
CTAG-$
UCP-$
L2 PUMP-$
L1 PUMP-$
DRAM
L2-$
L1-$
% EPI 开销
200
150
100
50
0
230%
GemsFDTD
astar
bwaves
bzip2
cactusADM
calculix
dealII
gamess
gcc
gobmk
gromacs
h264ref
hmmer
lbm
leslie3d
libquantum
mcf
milc
namd
omnetpp
perlbench
sjeng
specrand
sphinx3
wrf
zeusmp
mean
能量开销

图 10B

图10B

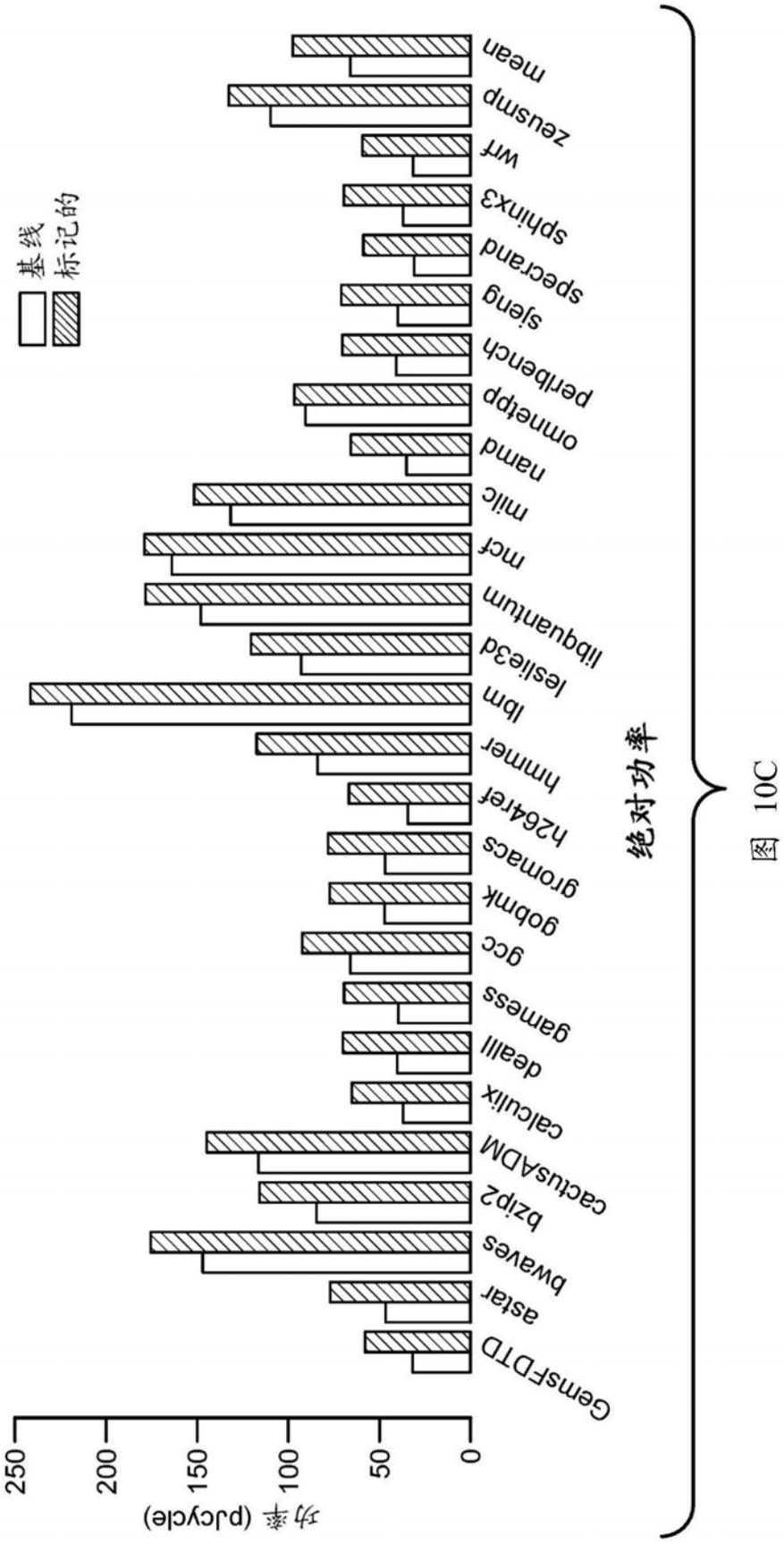

绝对功率
基线
标记的
功率 (pJ/cycle)
250
200
150
100
50
0
GemsFDTD
astar
bwaves
bzip2
cactusADM
calculix
dealII
gamess
gcc
gobmk
gromacs
h264ref
hmmer
lbm
leslie3d
libquantum
mcf
milc
namd
omnetpp
perlbench
sjeng
specrand
sphinx3
wrf
zeusmp
mean

图 10C

图10C

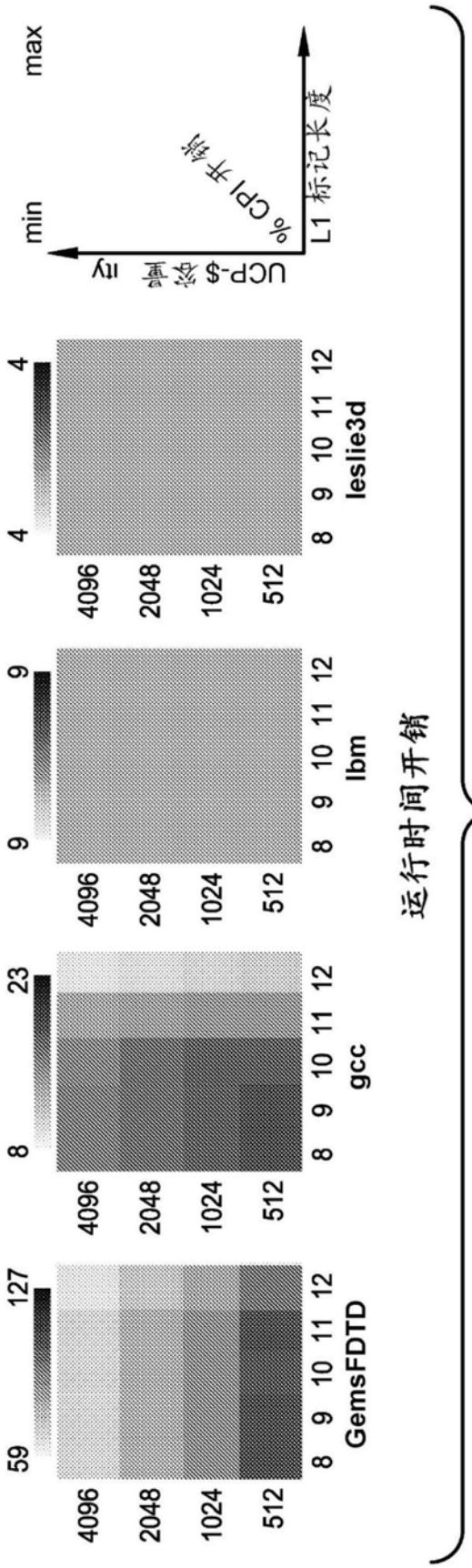

max
min
L1 标记长度
%CPI 未命中
UCP-$ 命中率
GemsFDTD
gcc
lbm
leslie3d
4096
2048
1024
512
8 9 10 11 12
运行时间开销

图 11A

图11A

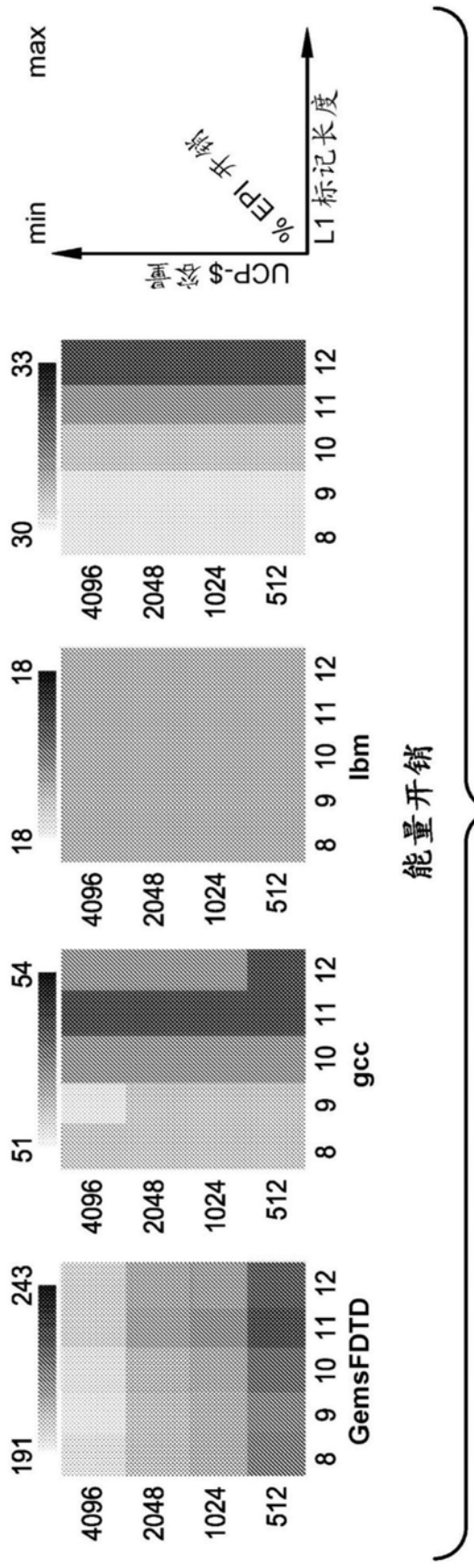

max
min
L1 标记长度
% EPI 开销
UCP-$ 的性能
GemsFDTD
gcc
lbm
191
243
51
54
18
18
30
33
8 9 10 11 12
4096
2048
1024
512
能量开销

图 11B

图11B

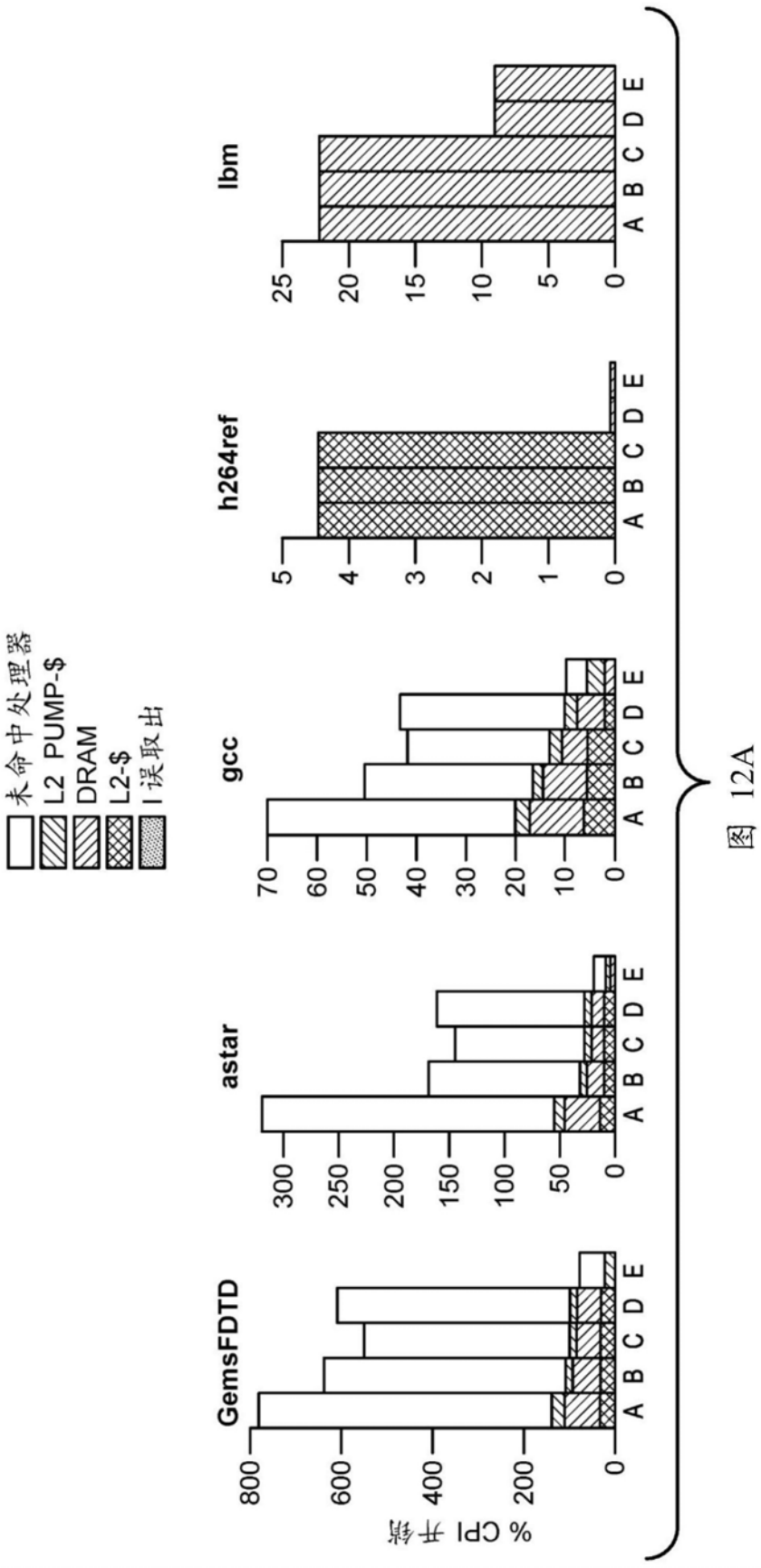

% CPI 开销
GemsFDTD
astar
gcc
h264ref
lbm
A B C D E
未命中处理器
L2 PUMP-$
DRAM
L2-$
I 误取出

图 12A

图12A

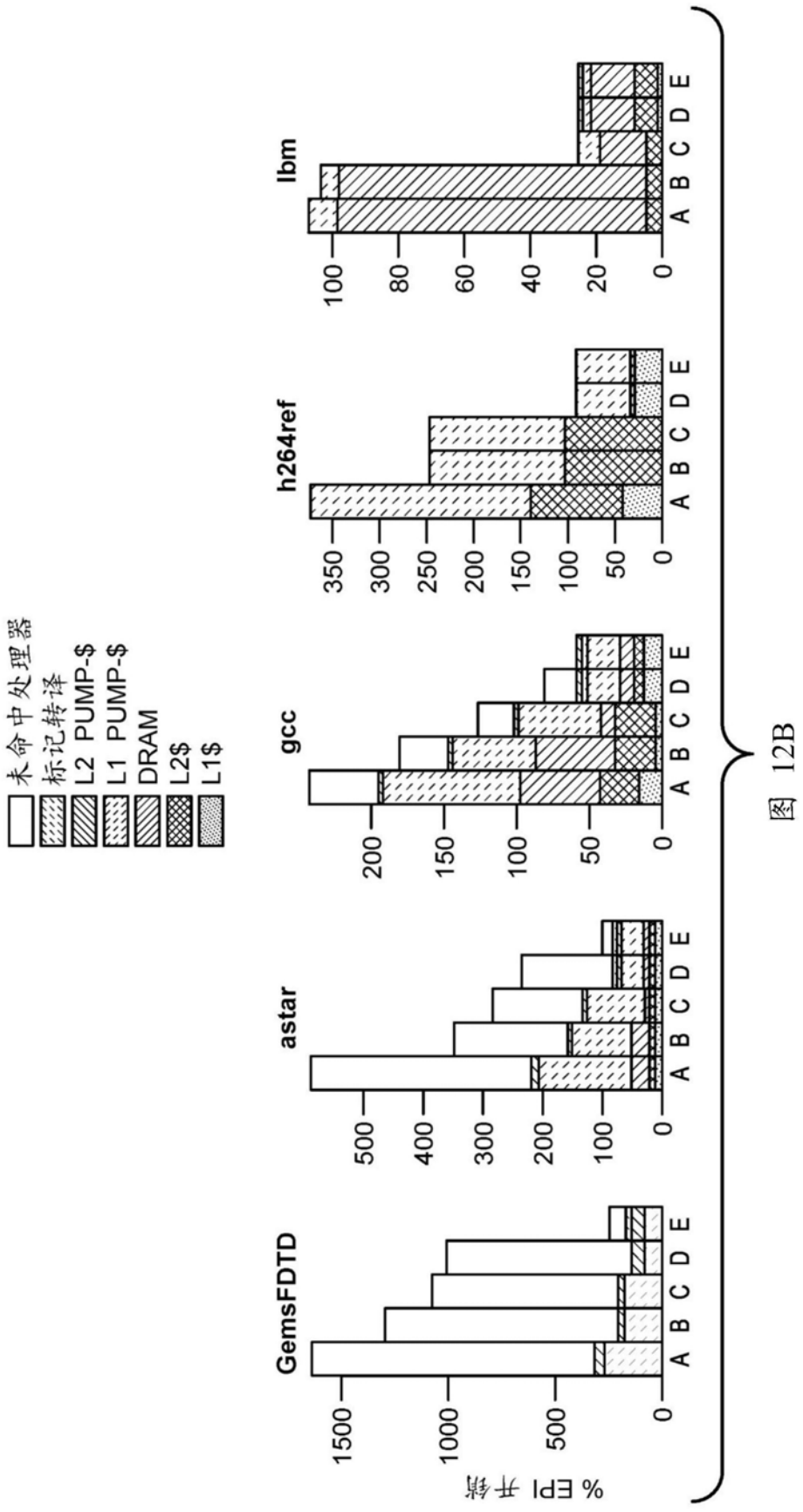

未命中处理器
标记转译
L2 PUMP-$
L1 PUMP-$
DRAM
L2$
L1$
% EPI 开销
GemsFDTD
astar
gcc
h264ref
lbm
A B C D E

图 12B

图12B

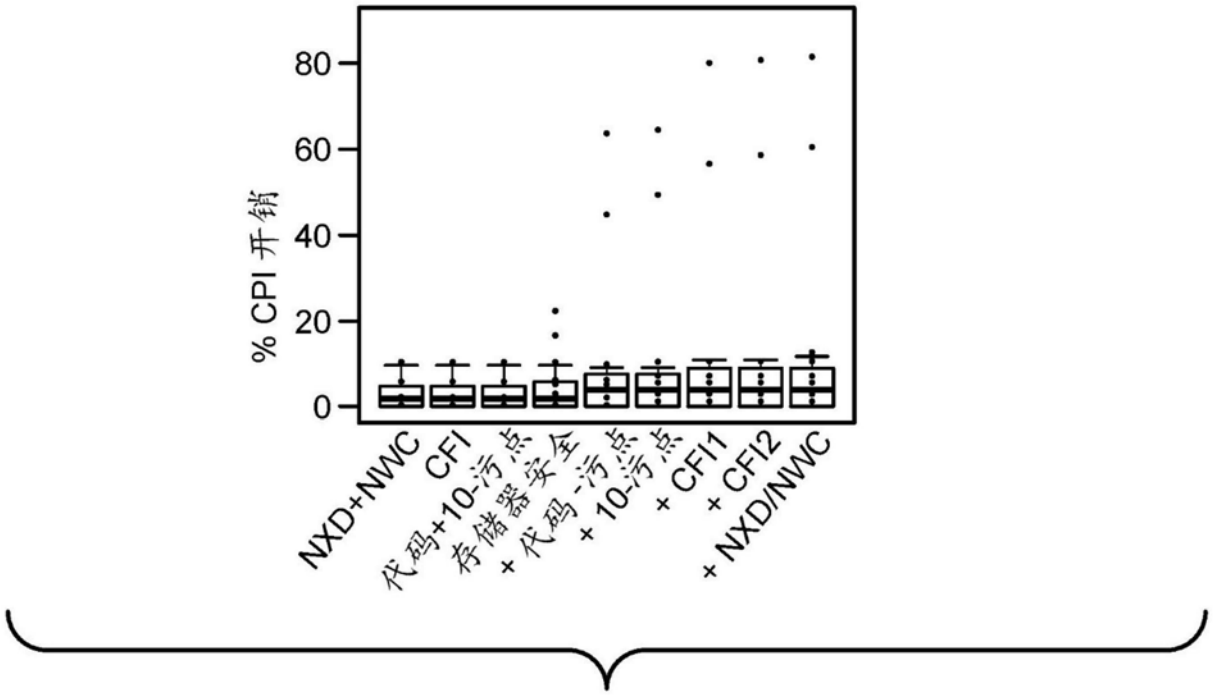


图 13A

图13A

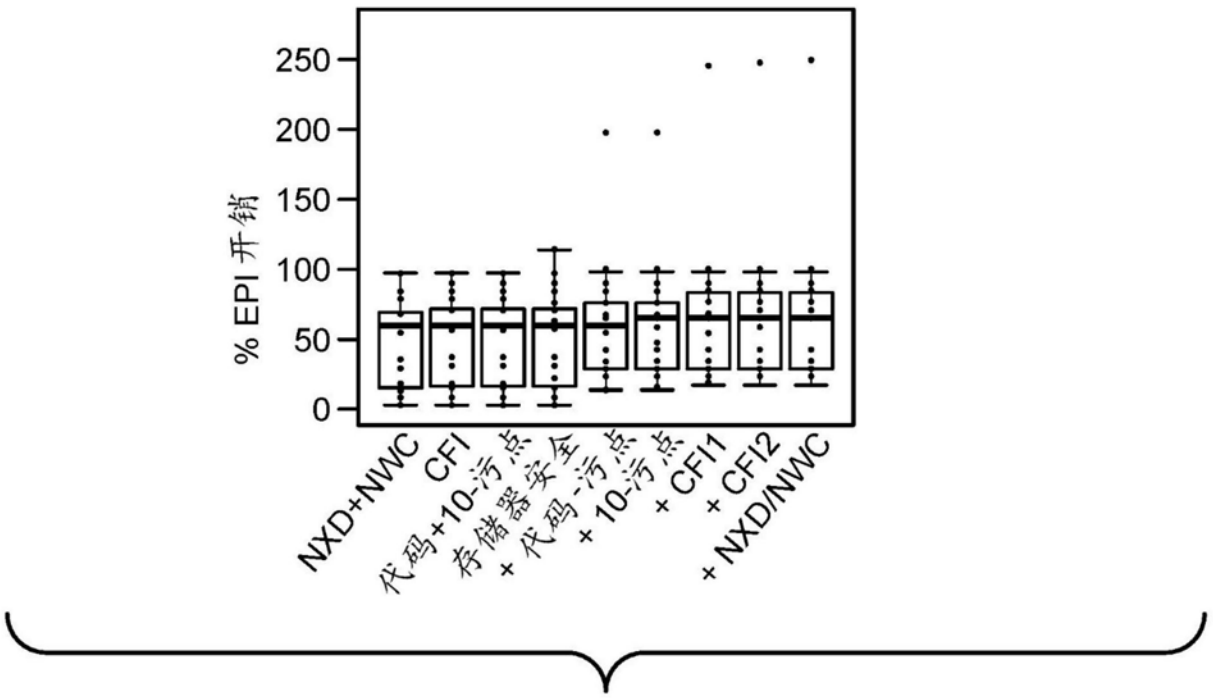


图 13B

图13B

表1-调查的策略的概述

策略	威胁	元数据	最大唯一标记	标记校验（允许？）	标记传播规则	参考
NXD+NWC	代码注入	关于存储器位置的DATA或者CODE	2	*CI*=CODE; *MR*/ =关于写入的CODE	无	
存储器安全	关于堆的空间/时间存储器安全性违规	关于指针的颜色；区域颜色+关于存储器位置的有效载荷颜色	(#mallocs)^2	指针颜色==参考的区域颜色	*R* ← 关于mov/add/sub的OPi *R* ← 关于负载的有效载荷(MR)	[17]
CFI	控制流劫持（JOP/ROP/代码重新使用）	关于每个间接控制流源和目标的唯一id	#sources+#targets	(*PC*,*CI*) ∈ 程序的控制流路径	*PC* ← 关于间接跳转的CI（包括调用，返回）	[4]
污点跟踪	不可信代码，来自IO的低完整性数据	关于指令和IO的源-污点-id，关于字的输入污点-id的集合	$2^{(\#code+IO\ ids)}$	用户定义的校验	*R* ← *CI U OP1 U OP2 U MR*	[47]
复合	所有以上的	具有4个要素的结构	以上的积	所有以上的（具有N=4的Alg.2）		

图 14

图14

表2-标记方案的分类

标记位	传播?	输出 允许?	R(结果)	PC	输入 PC	CI	OP1	OP2	MR	使用
2	X	软	X	X	X	X	X	X	✓	存储器保护
字	X	有限的可编程性	X	X	X	X	X	X	✓	存储器卫生，堆栈，隔离
32	X	有限的可编程性	X	X	X	X	X	X	✓	伪造的数据，隔离
2	X	固定的	固定的	X	X	X	X	X	✓	细粒度的同步
1	✓	固定的	X	X	X	X	✓	X	X	能力
2-8	✓	固定的	固定的	X	X	X	✓	✓	X	类型
128	✓	固定的	复制	X	X	X	✓	X	✓	存储器安全
0	✓	软件定义的		X	仅传播一					不变的校验
1	✓	固定的	固定的	X	X	X	✓	✓	✓	污点
4	✓	有限的可编程性		X	X	X	✓	✓	X	污点，介入，故障隔离
10	✓	有限的可编程性	固定的	X	X	X	✓	✓	✓	污点，隔离
未指定的	✓	软件定义的		X	X	X	✓	✓	✓	灵活的污点
32	✓	软件定义的		X	X	X	✓	✓	✓	可编程的，污点，存储器校验，参考计数
0-64	✓	软件定义的		✓	✓	✓	✓	✓	✓	信息流，类型
无限的	✓	软件定义的		✓	✓	✓	✓	✓	✓	完全可编程的，指针大小的标记

传播=标记以数据传播（✓）或者是存储器地址的性质（X）
允许？=用于允许/不允许操作的逻辑
R=策略影响结果的标记
PC=关于程序计数器的可编程标记
CI=是校验或者传播的当前指令标记
OP1，OP2，MR=是关于校验或者传播的这些输入的标记

图 15

图15

表3-用于基线和在32nm节点的简单PUMP-扩展的处理器的存储器资源估计

单元	设计	组织	面积 (mm^2)	访问能量 读/写 (pJ)	静态功率 (pJ/cyc)	等待时间 (ps)	Cyc
寄存器文件 (Int 和 FP)	基线	64b, 2R1W, {32 整数, 32 浮点 }	0.002	0.3/0.5	0.08	264	1
	扩展的64b标记	128b, 2R1W, {48 整数 , 32浮点 }	0.007	1.0/1.4	0.23	295	1
L1-$ (I 和 D)	基线	64KB, 4-向 , 64B/线	0.236	17/11	14.4	880	1
	扩展的64b标记	64KB, 4-向 , 128B/线 (有效32KB, 64B/线)	0.244	19/14	14.5	880	1
L2-$ (统一的)	基线	512KB, 8-向 , 64B/线	1.207	393/481	0.111	4000	5
	扩展的64b标记	1MB, 8-向 , 128B/线 (有效512KB, 64B/线)	2.350	758/1223	0.214	4930	5
TLB (I 和 D)	任一	1KB, 2-向集合关联	0.04	3.6/4.5	2	800	1
DRAM	基线	1GB, 每个转移访问64B线	-	15,000	-	-	100
	扩展的64b标记	1GB, 访问128B线 (有效的，每个转移64B线)	-	31,000	-	-	130
L1 PUMP-$	64b 标记	完全关联1024项，328b匹配，128b除外 (不使用的；仅为了参考示出)	1.500	750/900	-	3000	4
		多哈希1024项，328b匹配，128b除外	0.683	51/62	32	500	1
L2 PUMP-$	64b标记	多哈希4096项，328b匹配，128b除外	0.994	173/444	0.085	3300	4

总基线面积 **1.485mm²**

总64b标记的面积 **4.318mm²** (+ 基线以上+190%)

图 16

图16

表4-PUMP参数范围表

单元	参数	范围	最终的
L1 PUMP-$	容量	512-4096	1024
	标记位	8-12	10
L2 PUMP-$	容量	1024-8192	4096
	标记位	13-16	14
UCP-$	容量	512-4096	2048
CTAG-$	容量	128-1024	512

图17

表5-用于在32nm节点的PUMP优化的处理器的存储器资源估计

单元	设计	组织	面积 (mm^2)	访问能量 读/写 (pJ)	静态功率 (pJ/cyc)	等待时间 (ps)	Cyc
寄存器文件	扩展的10b	74b, 2R1W, {48 整数, 32 浮点 }	0.005	0.4/0.5	0.13	360	1
L1高速缓存	10b-标记	74KB, 4-向 , 74B/线 (前端64KB, 64B/线)	0.272	19/13	16.4	975	1
L2高速缓存	14b-标记	592KB, 8-向 , 78B/线 (前端512KB, 64B/线)	1.247	343/640	0.133	4600	5
TLB	-	1KB, 2-向集合关联	0.040	3.6/4.5	2	800	1
DRAM	64b-标记	1GB, 访问128B线 (移动76B)	-	17,500	-	-	112
L1 PUMP高速缓存	10b L1 标记	多哈希1024项， 58b匹配， 20b除外	0.095	15/43.2	2.22	520	1
L2 PUMP高速缓存	14b L2 标记	多哈希4096项， 78b匹配， 28b除外	0.287	99.4/267	0.032	2800	3
完全→ L2-标记	64b → 14b	8K项， 64b匹配， 14b除外， 多哈希	0.432	166/436	0.052	3400	4
L2-标记→ 完全	14b → 64b	16K x 64 SRAM	0.216	55.5/31.5	0.027	1700	2
L2-标记→ L1-标记	14b → 10b	16K x 11 SRAM	0.038	8.8/4.7	0.004	1420	2
L1-标记→ L2-标记	10b → 14b	1K x 14 SRAM	0.004	0.8/1	0.0004	780	1
UCP 高速缓存	64b标记	多哈希2048项， 328b匹配， 128b除外	0.377	196/479	0.035	2730	3
CTAG高速缓存	64b标记	多哈希512项	0.107	56/139	0.009	1700	2

总面积 **3.120mm^2** (+ 基线以上+110%)

图 18

图18

算法1 污点跟踪未命中处理器（部分）

1: 开关 (op)
2: 情况 加，减，或：
3: $PC_{new} \leftarrow PC$
4: $R \leftarrow$ 规范化 $(CI \cup OP1 \cup OP2)$
5: (... 对于省略的其他指令的情况 ...)
6: 默认：到误差处理器的陷阱

图 19

图19

算法2 N-策略未命中处理器

1: **for** i=1 to N **do**
2: $M_i \leftarrow \{op, PC[I], CI[I], OP1[I], OP2[I], MR[I]\}$
3: $\{pc_i, res_i\} \leftarrow$ 策略$_i$ (M_i)
4: $PC_{new} \leftarrow$ 规范化 $([pc_1, pc_2, ..., pc_N])$
5: $R \leftarrow$ 规范化 $([res_1, res_2, ..., res_N])$

图 20

图20

算法3 具有HW支持的N-策略未命中处理器

```
1:  for  i=1 to N do
2:       M_i ← {op, PC[I], CI[I], OP1[I], OP2[I], MR[I]}
3:       {hit, pc_i , res_i} ← UCP-$(i, M_i)
4:       if !hit then
5:            {pc_i , res_i} ← policy_i (M_i)
6:            insert {i, pc_i , res_i} into UCP-$
7:  pc[1...N] ← {pc_1 , pc_2 , ..., pc_N}
8:  {hit, PC_new} ← CTAG-$ (pc[1...N])
9:  if !hit then
10:      PC_new ← canonicalize (pc[1...N])
11:      insert {pc[1...N], PC_new} into CTAG-$
12: res[1...N] ← {res_1 , res_2 , ..., res_N}
13: {hit, R} ← CTAG-$ (res[1...N])
14: if !hit then
15:      R ← canonicalize (res[1...N])
16:      insert {res[1...N], R} into CTAG-$
```

图 21

图21

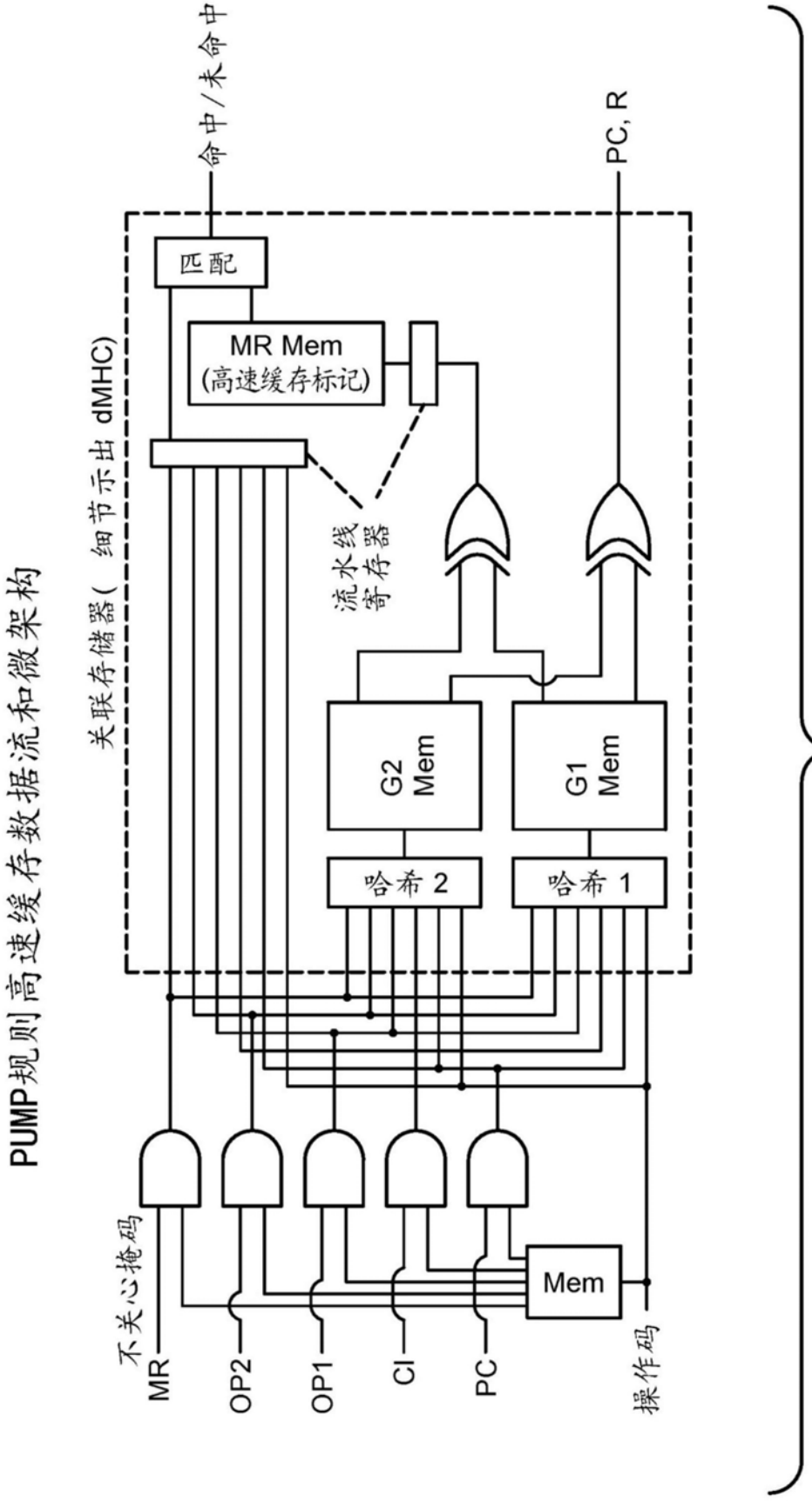

PUMP规则高速缓存数据流和微架构
关联存储器(细节示出 dMHC)
命中/未命中
PC, R
匹配
MR Mem
(高速缓存标记)
流水线
寄存器
G2
Mem
G1
Mem
哈希 2
哈希 1
Mem
不关心掩码
MR
OP2
OP1
CI
PC
操作码

图 22

图22

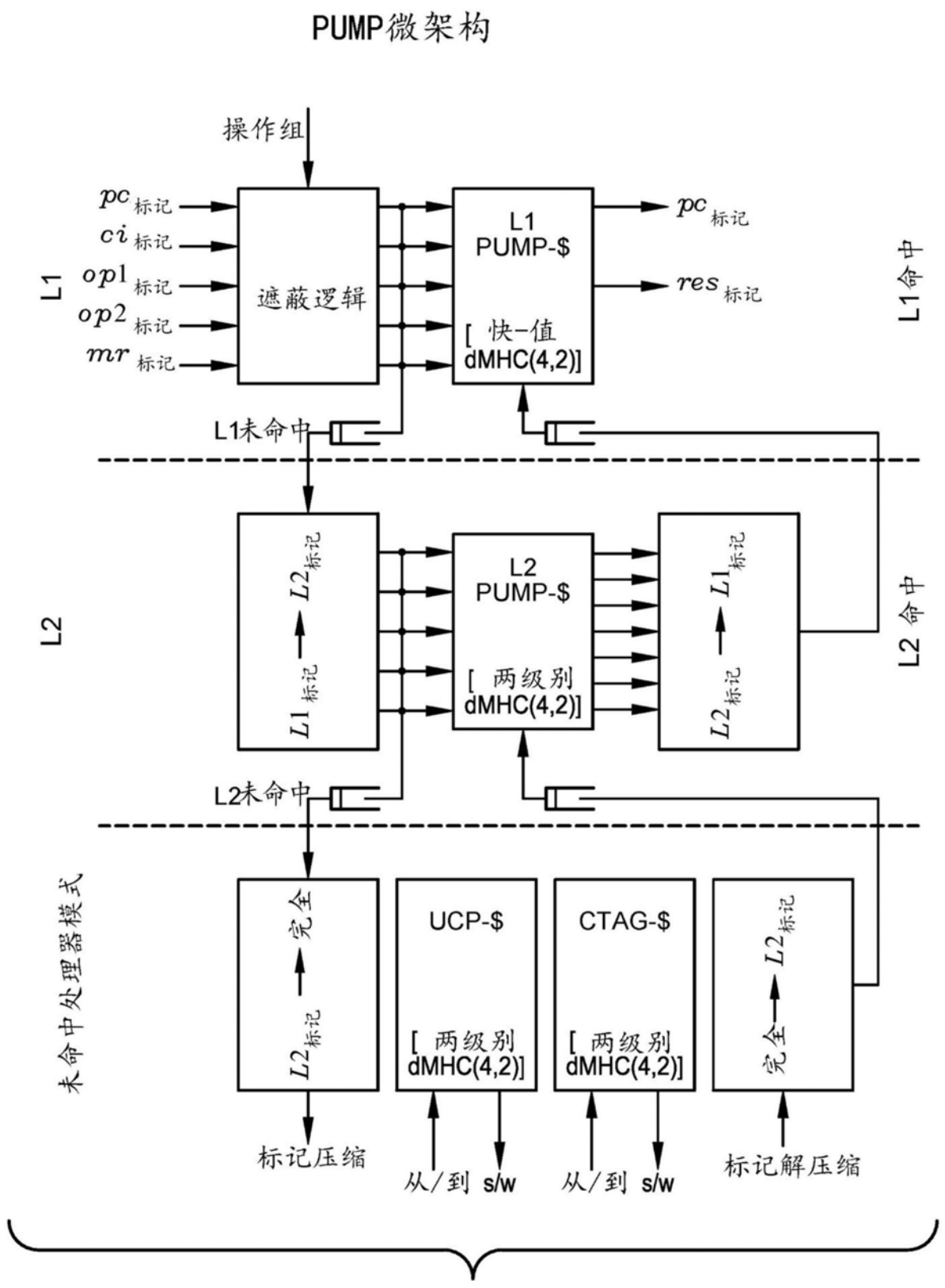

PUMP微架构
操作组
pc 标记
ci 标记
op1 标记
op2 标记
mr 标记
L1
遮蔽逻辑
L1
PUMP-$
[快-值
dMHC(4,2)]
pc 标记
res 标记
L1命中
L1未命中
L2
L2 PUMP-$
[两级别
dMHC(4,2)]
L2 命中
L2未命中
未命中处理器模式
UCP-$
[两级别
dMHC(4,2)]
CTAG-$
[两级别
dMHC(4,2)]
标记压缩
从/到 s/w
从/到 s/w
标记解压缩

图 23

图23

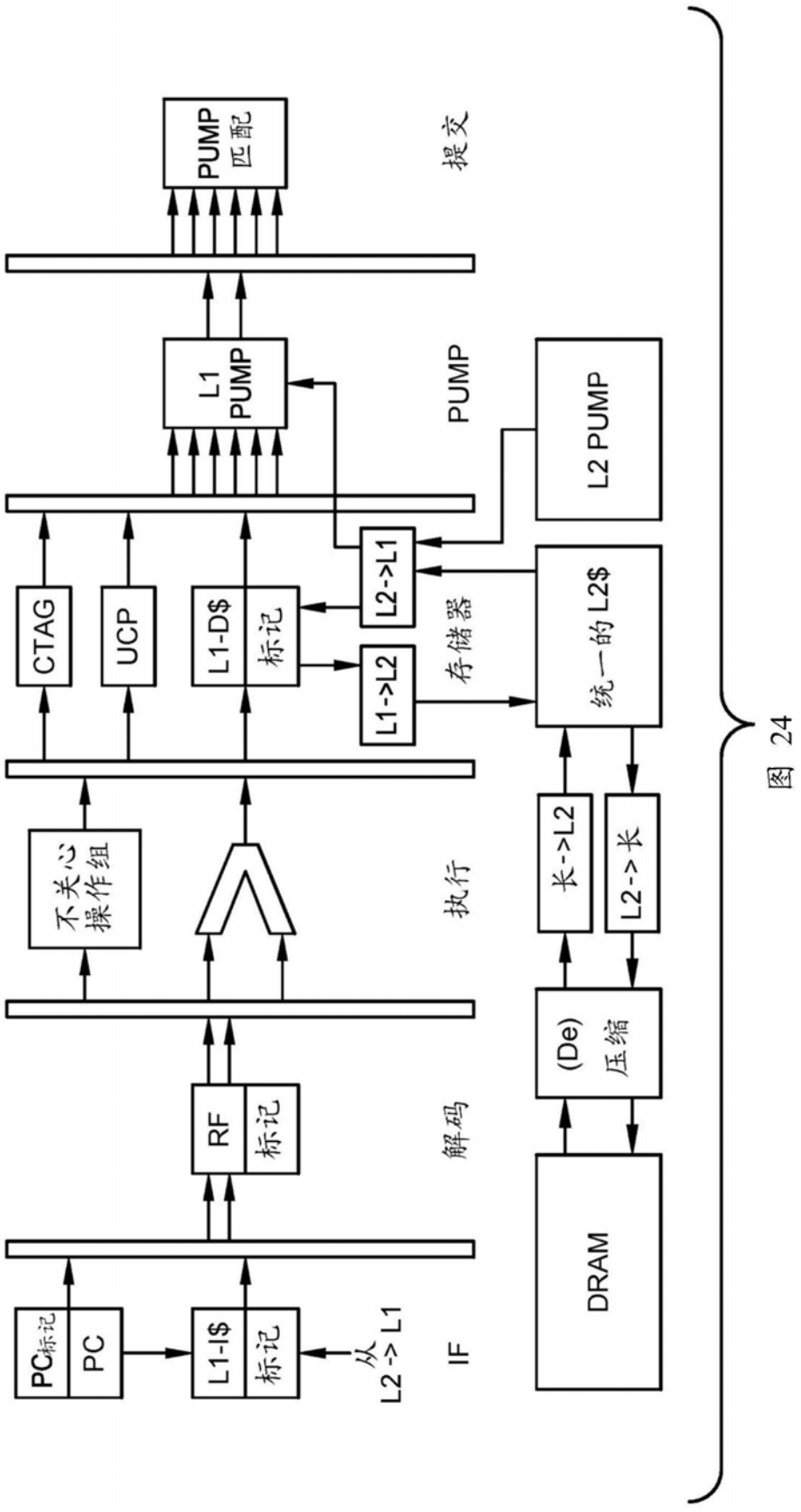

PUMP
匹配
提交
L1
PUMP
PUMP
L2 PUMP
CTAG
UCP
L1-D$
标记
L2->L1
L1->L2
存储器
统一的 L2$
不关心
操作组
执行
长->L2
L2-> 长
RF
标记
(De)
压缩
解码
PC标记
PC
L1-I$
标记
从
L2 -> L1
IF
DRAM

图 24

图24

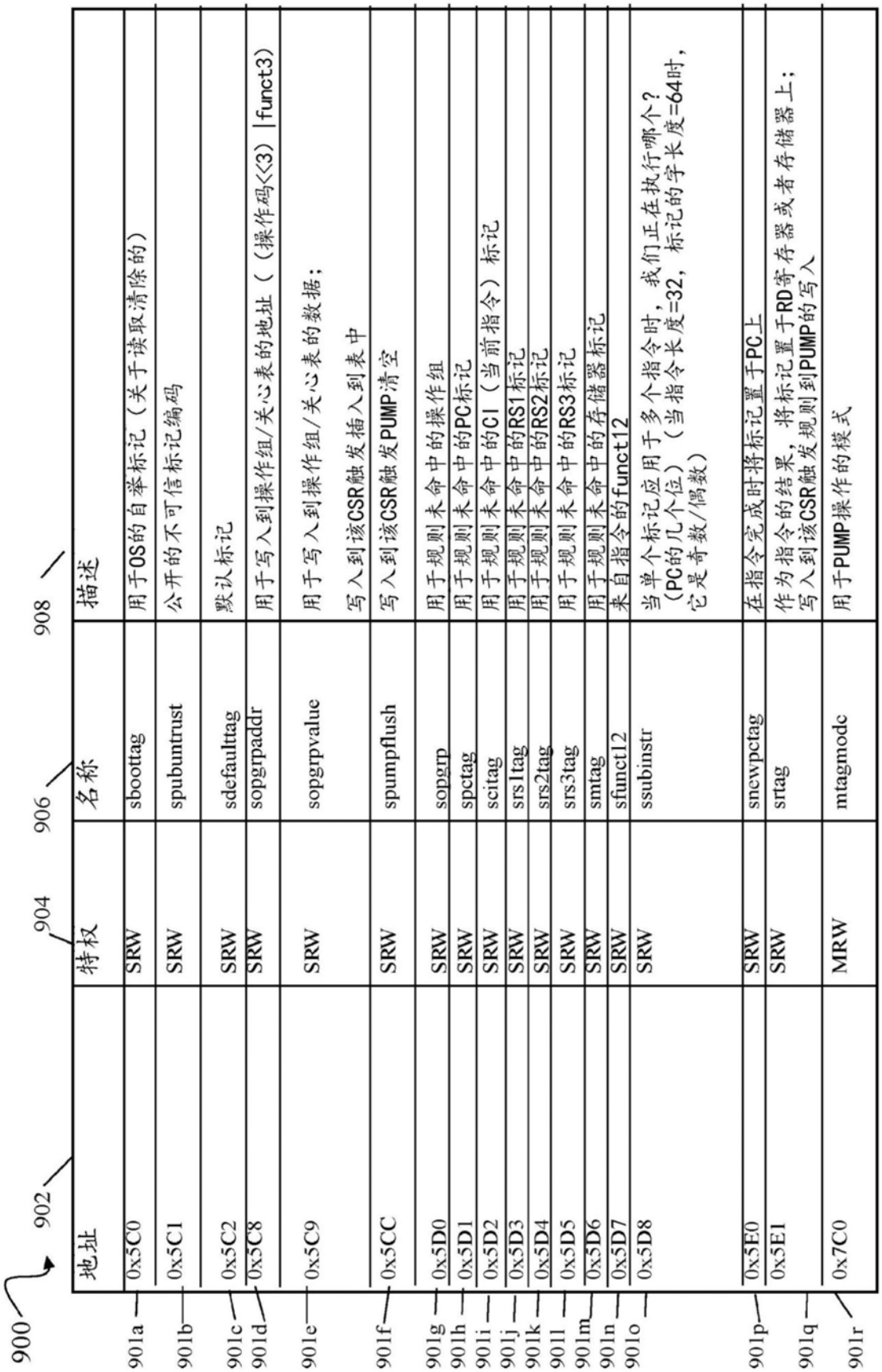

900
902
904
906
908
地址
特权
名称
描述
901a 0x5C0 SRW sboottag 用于OS的自举标记（关于读取清除的）
901b 0x5C1 SRW spubuntrust 公开的不可信标记编码
901c 0x5C2 SRW sdefaulttag 默认标记
901d 0x5C8 SRW sopgrpaddr 用于写入到操作组/关心表的地址（（操作码<<3）|funct3）
901e 0x5C9 SRW sopgrpvalue 用于写入到操作组/关心表的数据；
写入到该CSR触发插入到表中
901f 0x5CC SRW spumpflush 写入到该CSR触发PUMP清空
901g 0x5D0 SRW sopgrp 用于规则未命中的操作组
901h 0x5D1 SRW spctag 用于规则未命中的PC标记
901i 0x5D2 SRW scitag 用于规则未命中的CI（当前指令）标记
901j 0x5D3 SRW srs1tag 用于规则未命中的RS1标记
901k 0x5D4 SRW srs2tag 用于规则未命中的RS2标记
901l 0x5D5 SRW srs3tag 用于规则未命中的RS3标记
901m 0x5D6 SRW smtag 用于规则未命中的存储器标记
901n 0x5D7 SRW sfunct12 来自指令的funct12
901o 0x5D8 SRW ssubinstr 当单个标记应用于多个指令时，我们正在执行哪个？
（PC的几个位）（当指令长度=32，标记的字长度=64时，
它是奇数/偶数）
901p 0x5E0 SRW snewpctag 在指令完成时将标记置于PC上
901q 0x5E1 SRW srtag 作为指令的结果，将标记置于RD寄存器或者存储器上；
写入到该CSR触发规则到PUMP的写入
901r 0x7C0 MRW mtagmodc 用于PUMP操作的模式

图25

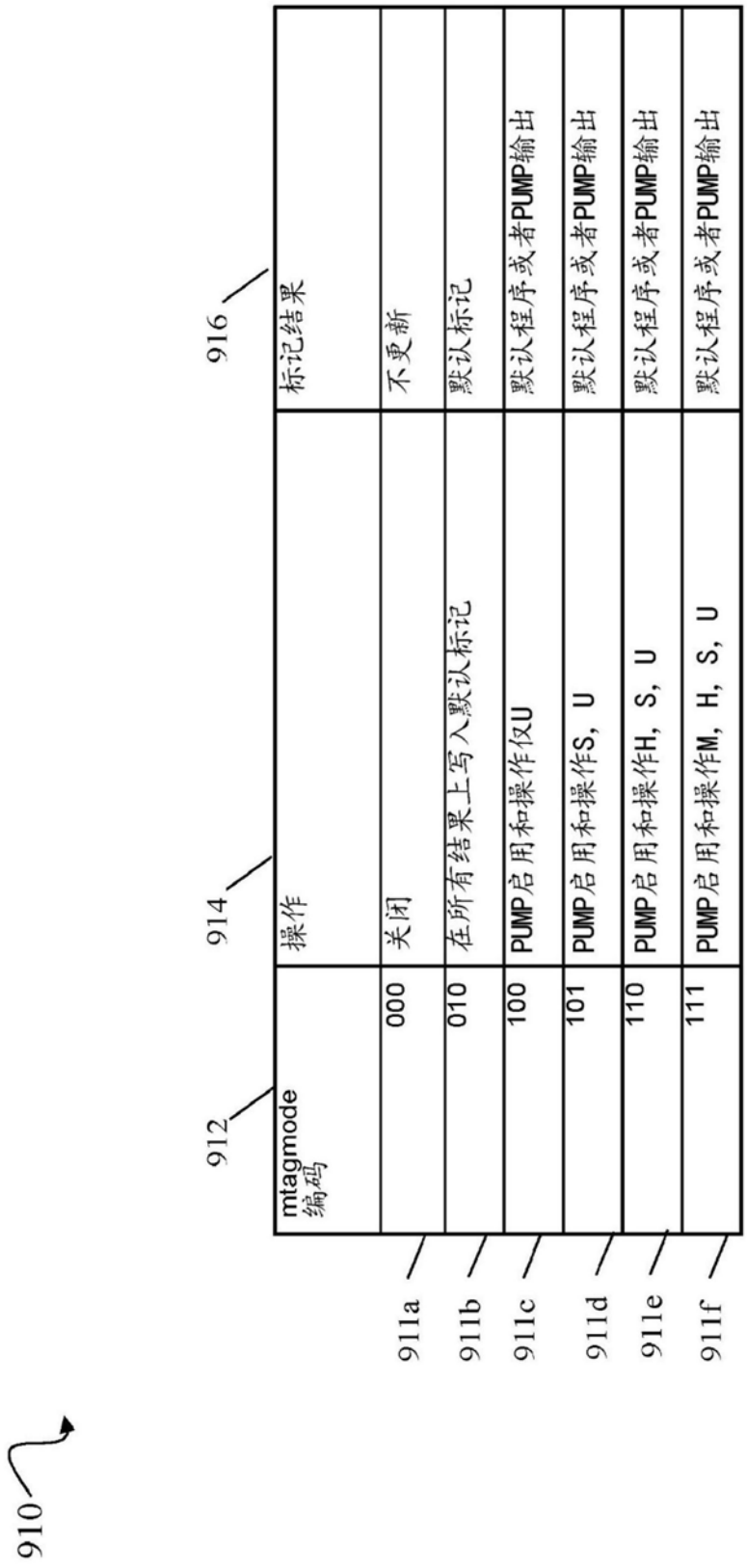

910
912
914
916
mtagmode 编码
操作
标记结果
000
关闭
不更新
010
在所有结果上写入默认标记
默认标记
100
PUMP启用和操作仅U
默认程序或者PUMP输出
101
PUMP启用和操作S，U
默认程序或者PUMP输出
110
PUMP启用和操作H，S，U
默认程序或者PUMP输出
111
PUMP启用和操作M，H，S，U
默认程序或者PUMP输出
911a
911b
911c
911d
911e
911f

图26

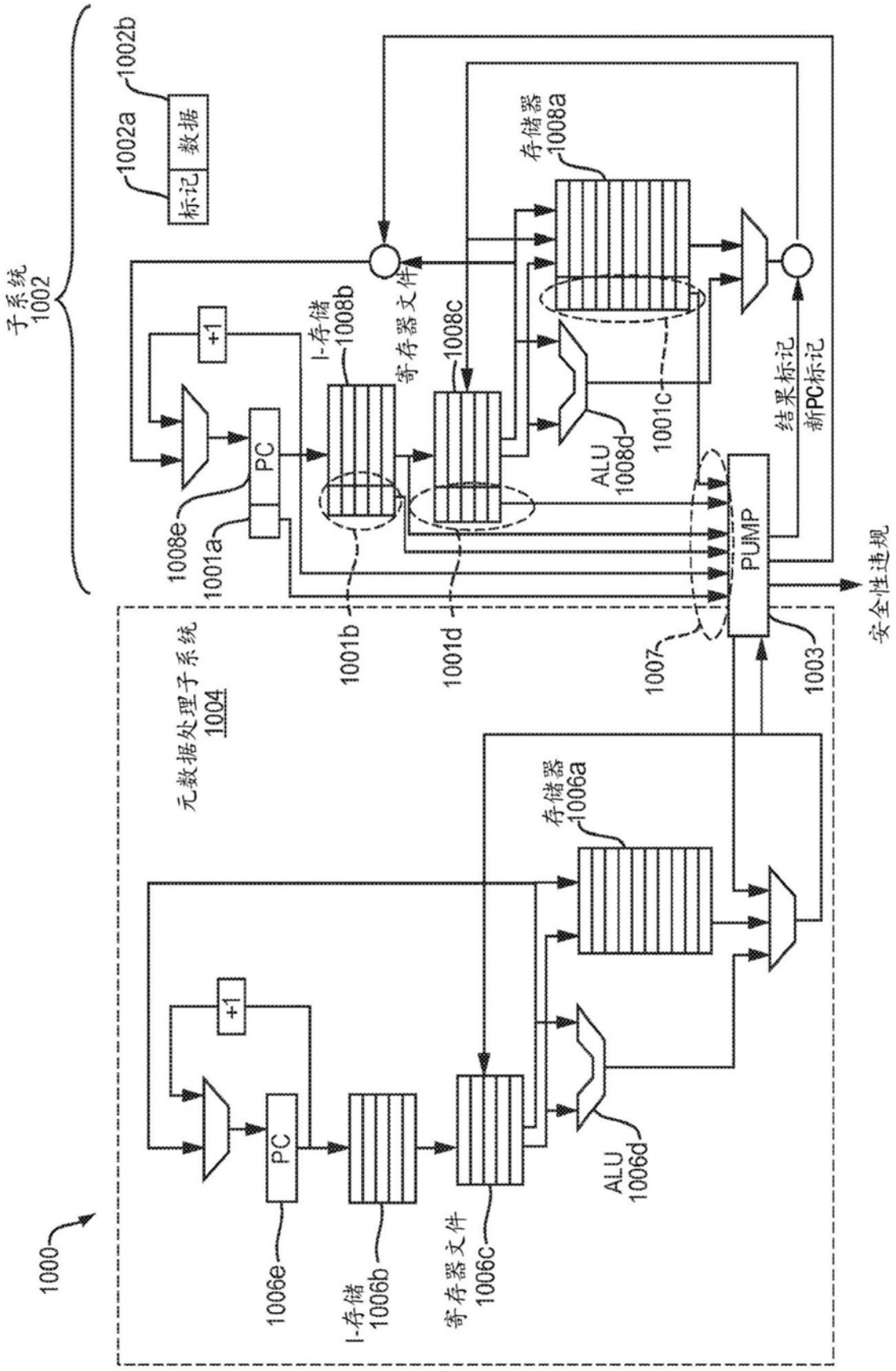

子系统
1002
1002b
1002a
标记
数据
存储器
1008a
寄存器文件
1008c
I-存储
1008b
+1
PC
1008e
1001a
ALU
1008d
1001c
结果标记
新PC标记
PUMP
安全性违规
1001b
1001d
1007
1003
元数据处理子系统
1004
1000
存储器
1006a
+1
PC
1006e
I-存储
1006b
寄存器文件
1006c
ALU
1006d

图27

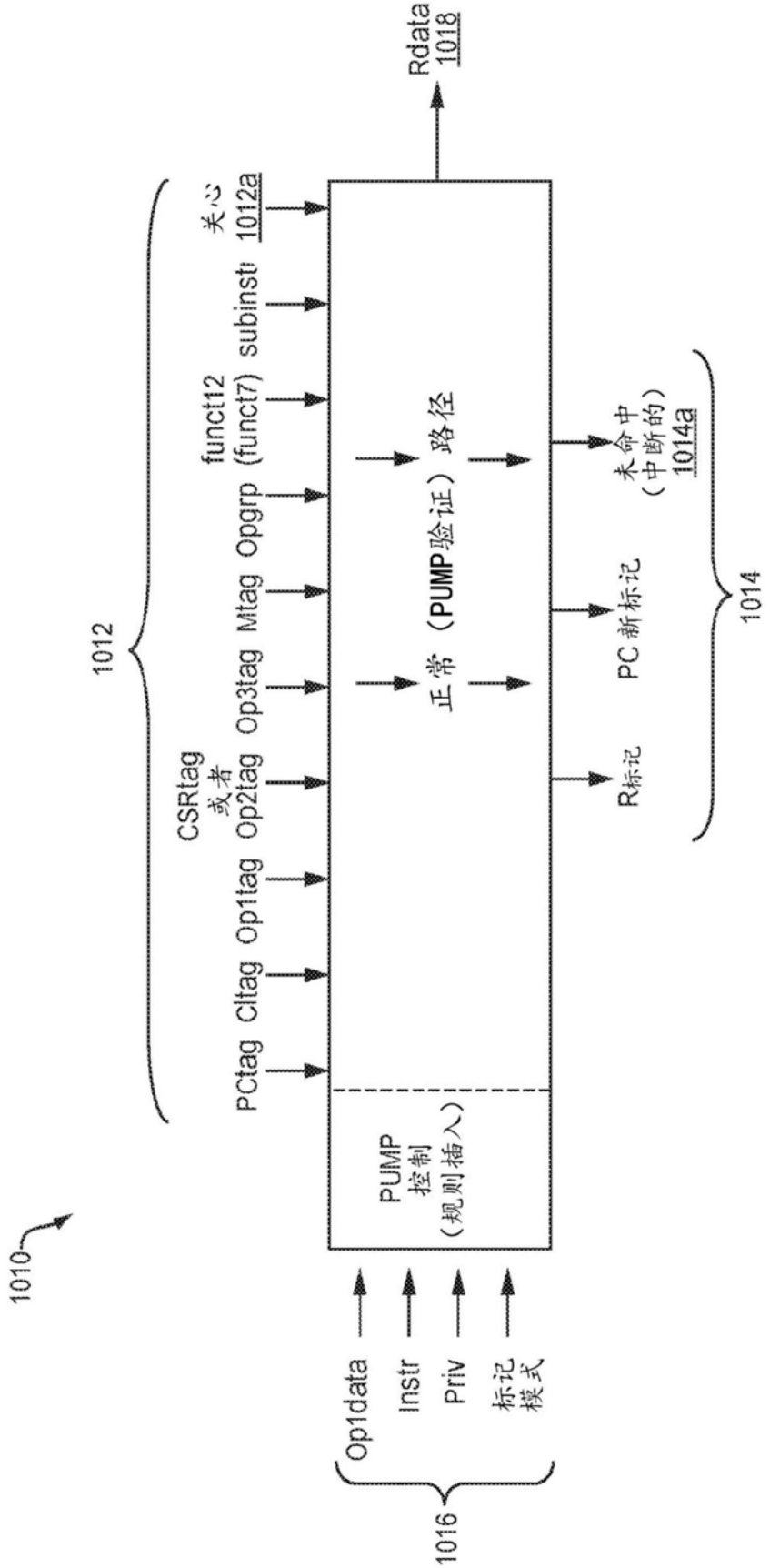

1010
1012
PCtag
Cltag
Op1tag
CSRtag
或者
Op2tag
Op3tag
Mtag
Opgrp
funct12
(funct7)
subinstr
关心
1012a
PUMP
控制
(规则插入)
正常
(PUMP验证)
路径
Rdata
1018
R标记
PC新标记
未命中
(中断的)
1014a
1014
Op1data
Instr
Priv
标记
模式
1016

图28

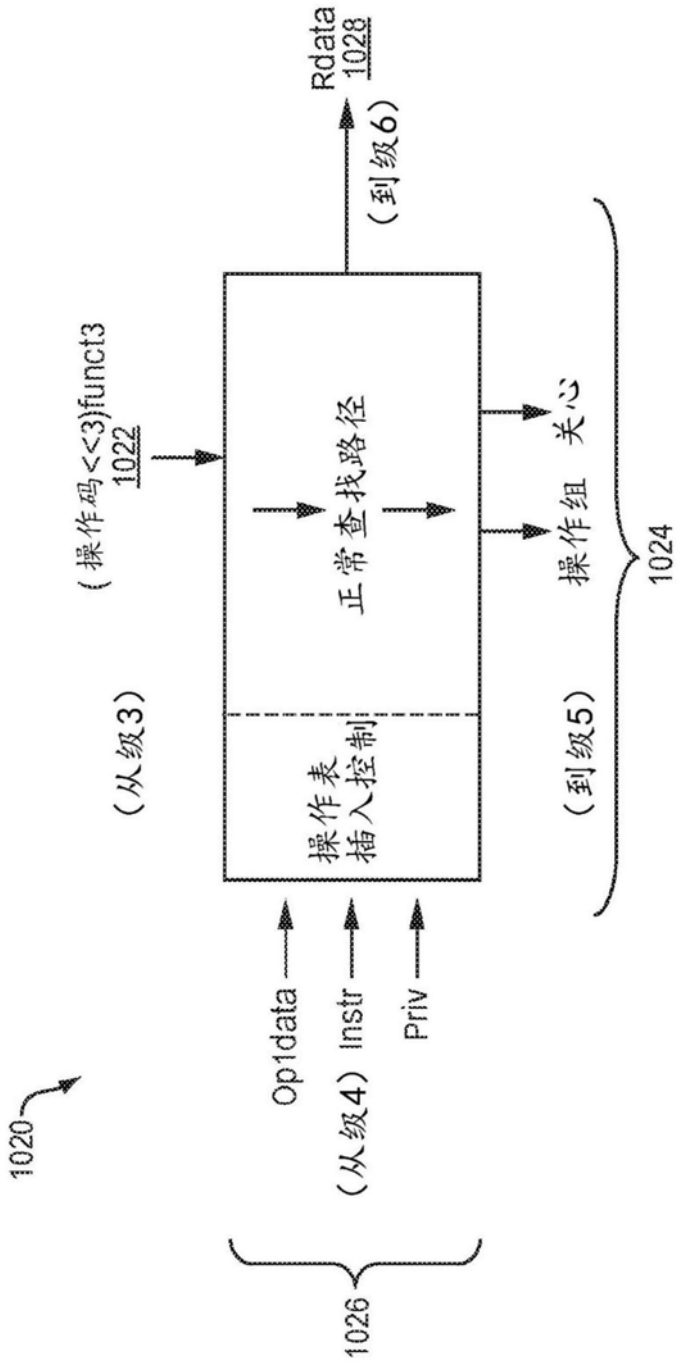

Rdata
1028
（到级6）
（操作码<<3)funct3
1022
正常查找路径
关心
操作组
1024
（从级3）
操作表
插入控制
（到级5）
1020
Op1data
（从级4）Instr
Priv
1026

图29

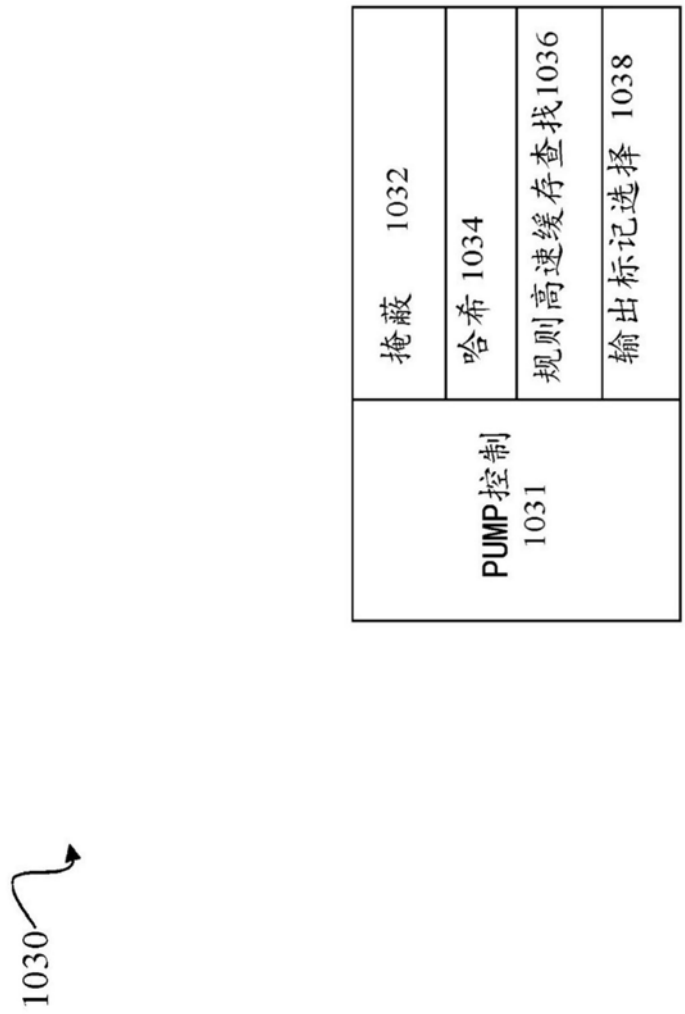


图30

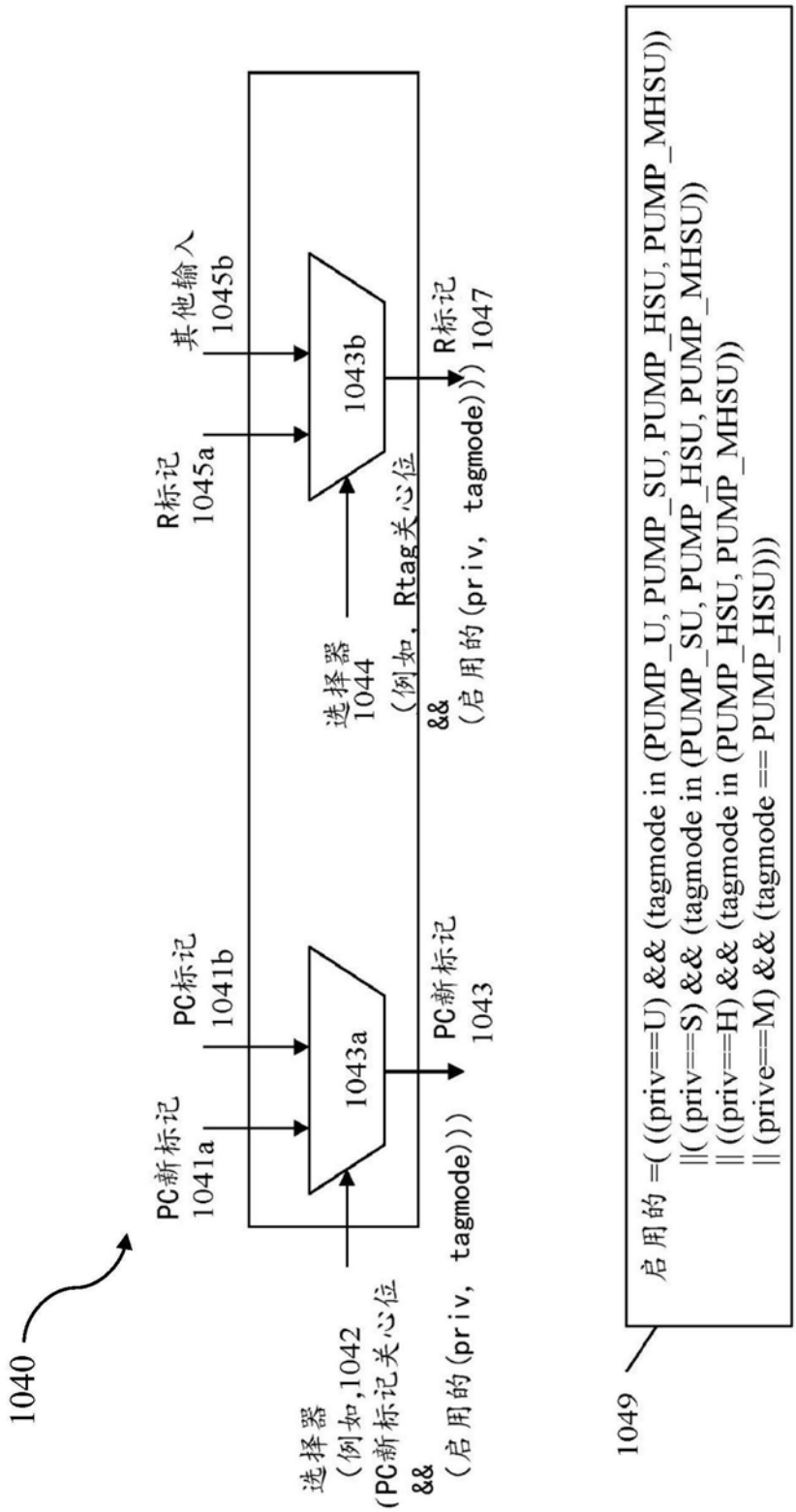

1040
PC新标记
1041a
PC标记
1041b
1043a
PC新标记
1043
选择器
(例如,1042
(PC新标记关心位
&&
(启用的(priv, tagmode)))
R标记
1045a
其他输入
1045b
1043b
选择器
1044
(例如, Rtag关心位
&&
(启用的(priv, tagmode)))
R标记
1047
1049
启用的 =(((priv==U) && (tagmode in (PUMP_U, PUMP_SU, PUMP_HSU, PUMP_MHSU))
||((priv==S) && (tagmode in (PUMP_SU, PUMP_HSU, PUMP_MHSU))
|| ((priv==H) && (tagmode in (PUMP_HSU, PUMP_MHSU))
|| (prive==M) && (tagmode == PUMP_HSU)))

图31

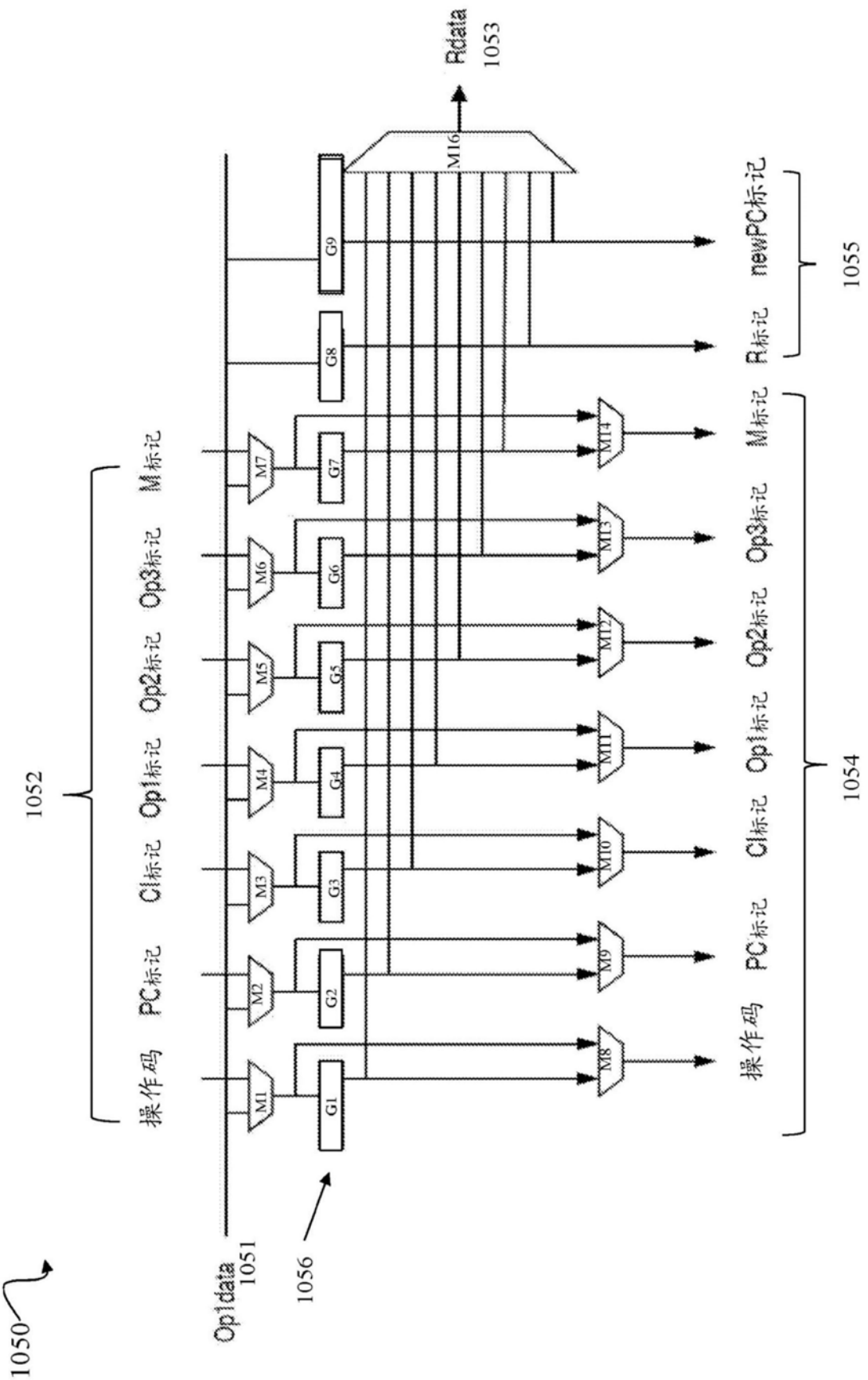

1050
1051
Op1data
1056
1052
操作码
PC标记
CI标记
Op1标记
Op2标记
Op3标记
M标记
M1
M2
M3
M4
M5
M6
M7
G1
G2
G3
G4
G5
G6
G7
G8
G9
M16
Rdata
1053
M8
M9
M10
M11
M12
M13
M14
操作码
PC标记
CI标记
Op1标记
Op2标记
Op3标记
M标记
R标记
newPC标记
1054
1055

图32

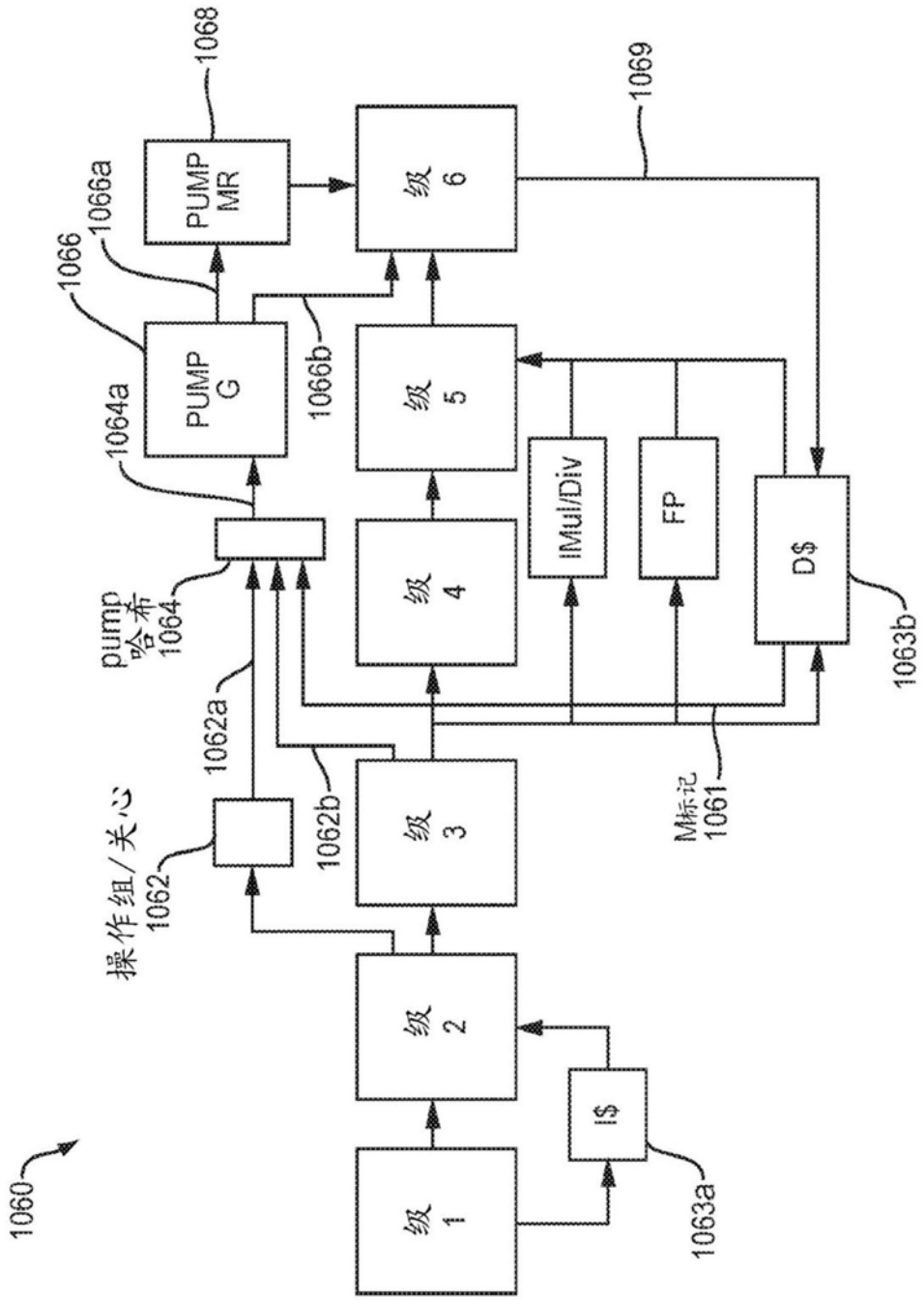

1060
1068
PUMP MR
1066a
1066
PUMP G
1064a
pump 哈希 1064
操作组/关心 1062
1062a
1062b
1066b
级 1
级 2
级 3
级 4
级 5
级 6
1069
I$
1063a
IMul/Div
FP
D$
1063b
M标记 1061

图33

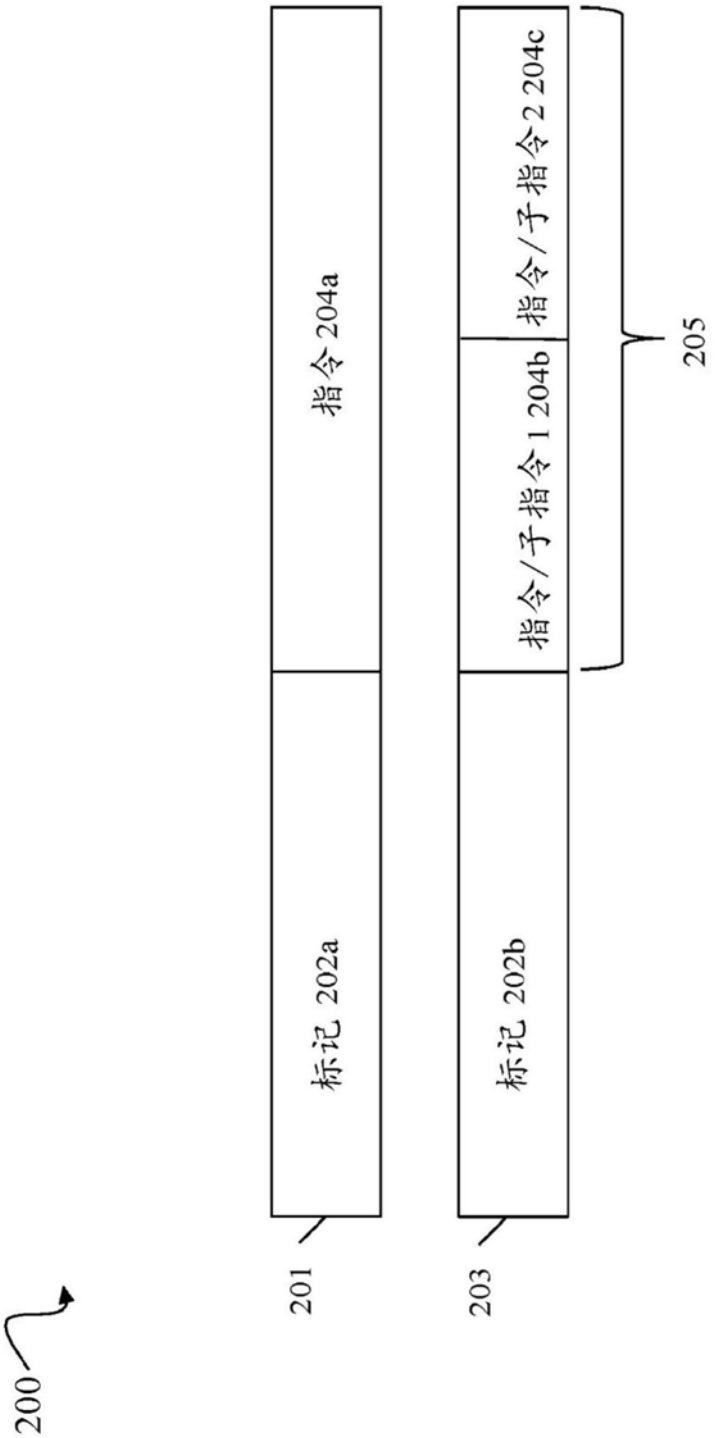

200
201
203
标记 202a
指令 204a
标记 202b
指令/子指令1 204b
指令/子指令2 204c
205

图34

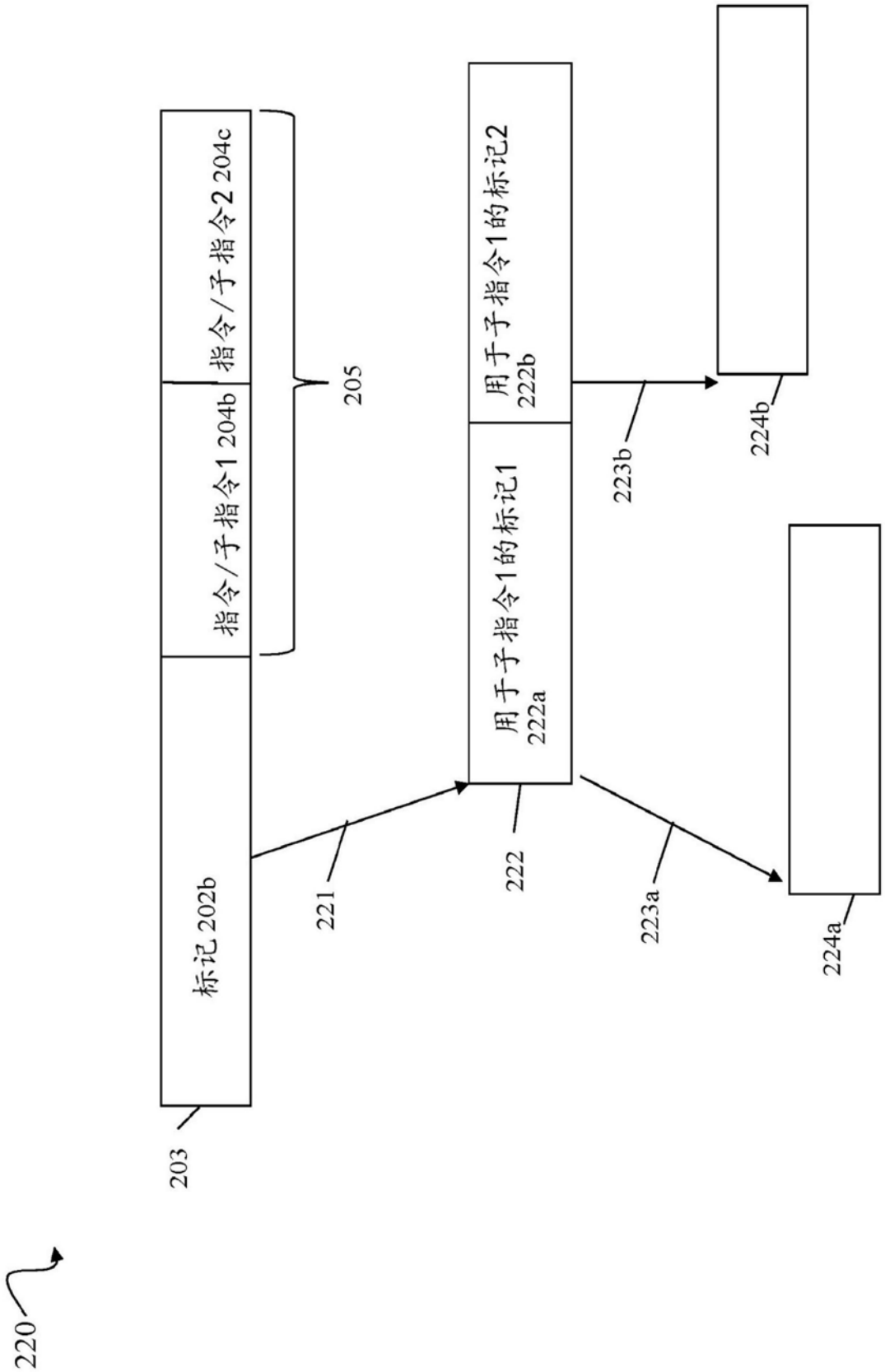

220
203
标记 202b
指令/子指令1 204b
指令/子指令2 204c
205
221
222
用于子指令1的标记1
222a
用于子指令1的标记2
222b
223a
223b
224a
224b

图35

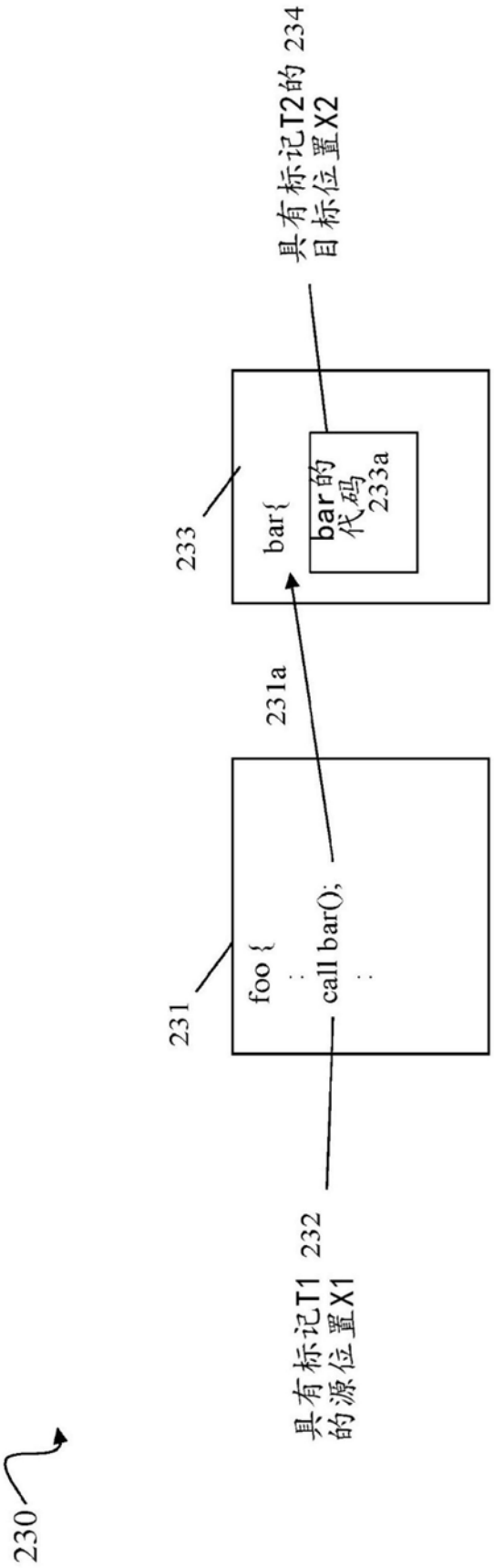

230
具有标记T1 232
的源位置X1
231
foo {
:
call bar();
:
231a
233
bar{
bar的
代码
233a
具有标记T2的 234
目标位置X2

图36

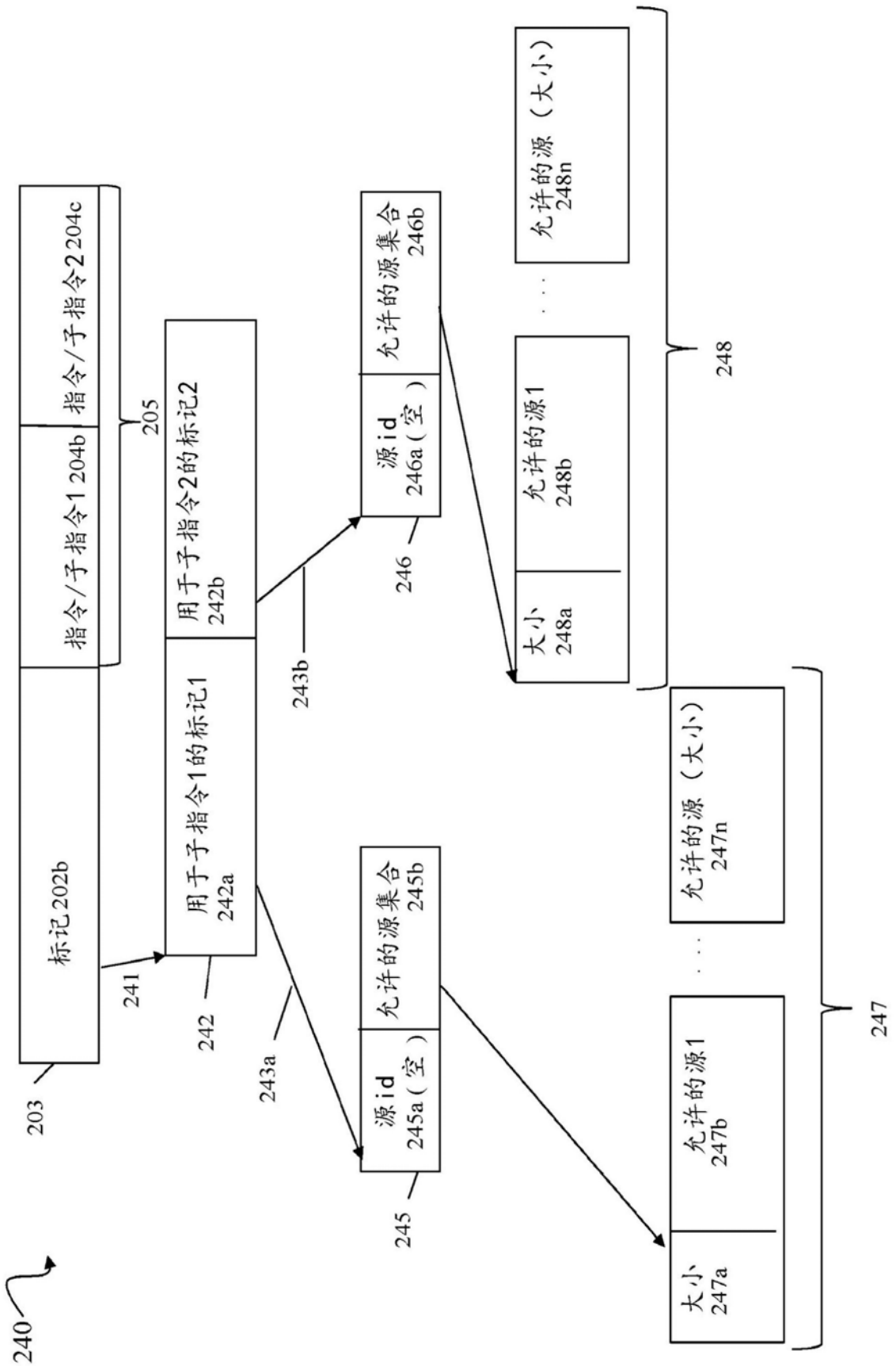

240
203
标记202b
指令/子指令1 204b
指令/子指令2 204c
205
241
242
用于子指令1的标记1
242a
用于子指令2的标记2
242b
243a
243b
245
源id
245a（空）
允许的源集合
245b
246
源id
246a（空）
允许的源集合
246b
247
大小
247a
允许的源1
247b
允许的源（大小）
247n
248
大小
248a
允许的源1
248b
允许的源（大小）
248n

图37

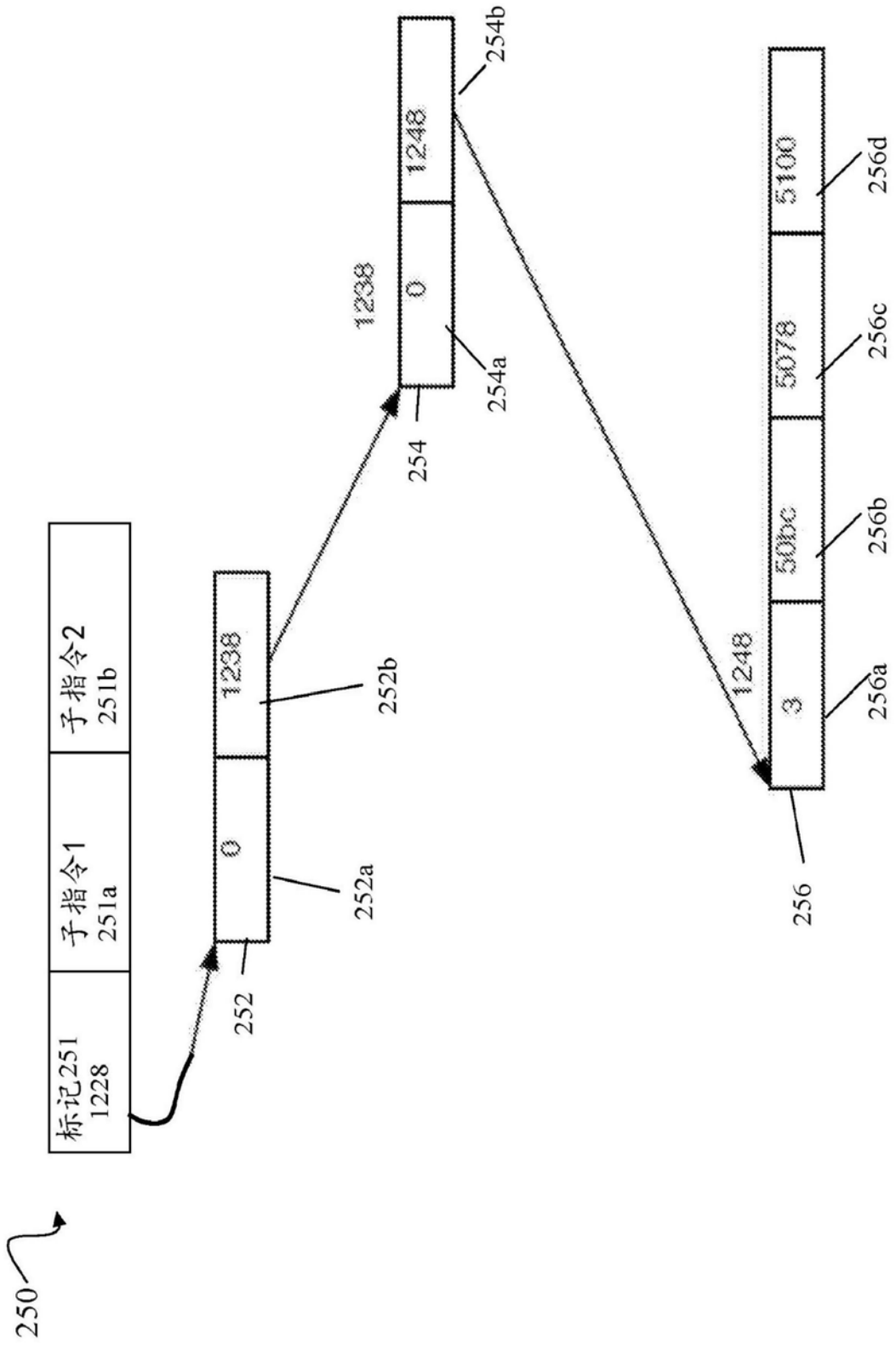

250
标记251
1228
子指令1
251a
子指令2
251b
252
0
1238
252a
252b
254
0
1238
1248
254a
254b
1248
3
50bc
5078
5100
256
256a
256b
256c
256d

图38

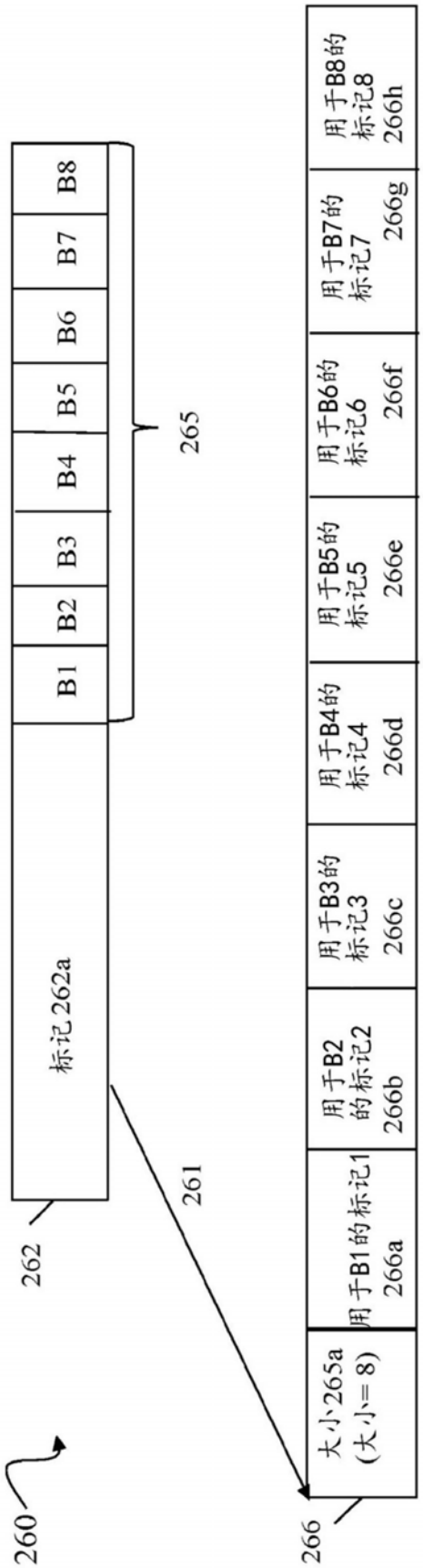

260
262
标记 262a
B1
B2
B3
B4
B5
B6
B7
B8
265
261
266
大小265a
(大小= 8)
用于B1的标记1
266a
用于B2
的标记2
266b
用于B3的
标记3
266c
用于B4的
标记4
266d
用于B5的
标记5
266e
用于B6的
标记6
266f
用于B7的
标记7
266g
用于B8的
标记8
266h

图39

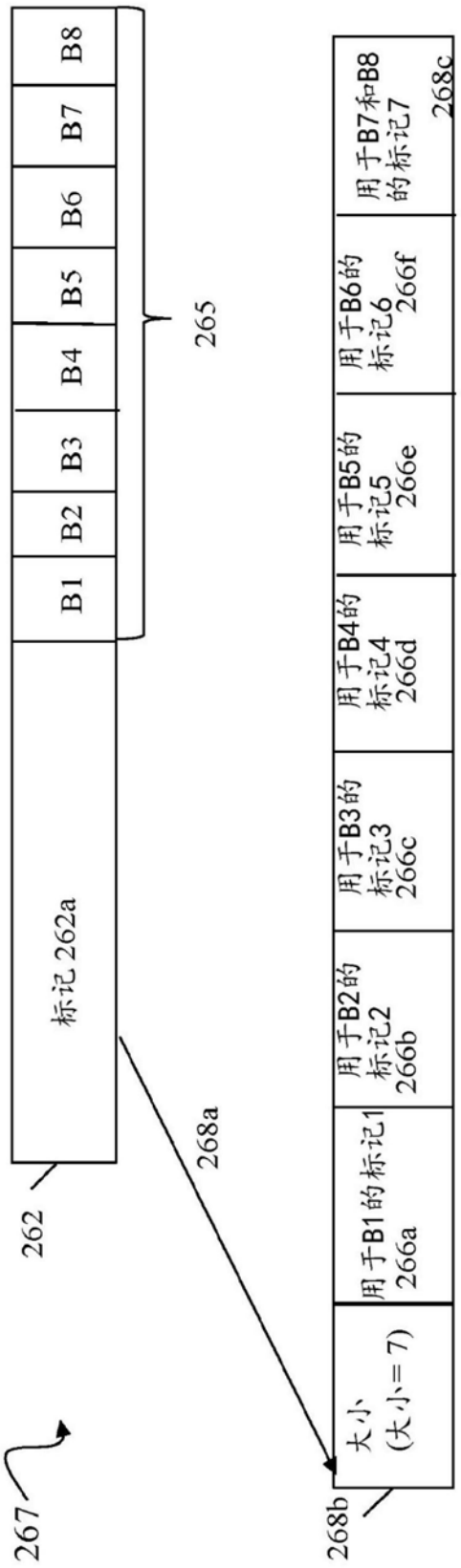

B8
B7
B6
B5
B4
B3
B2
B1
265
标记 262a
262
268a
267
268b
大小
(大小= 7)
用于B1的标记1
266a
用于B2的
标记2
266b
用于B3的
标记3
266c
用于B4的
标记4
266d
用于B5的
标记5
266e
用于B6的
标记6
266f
用于B7和B8
的标记7
268c

图40

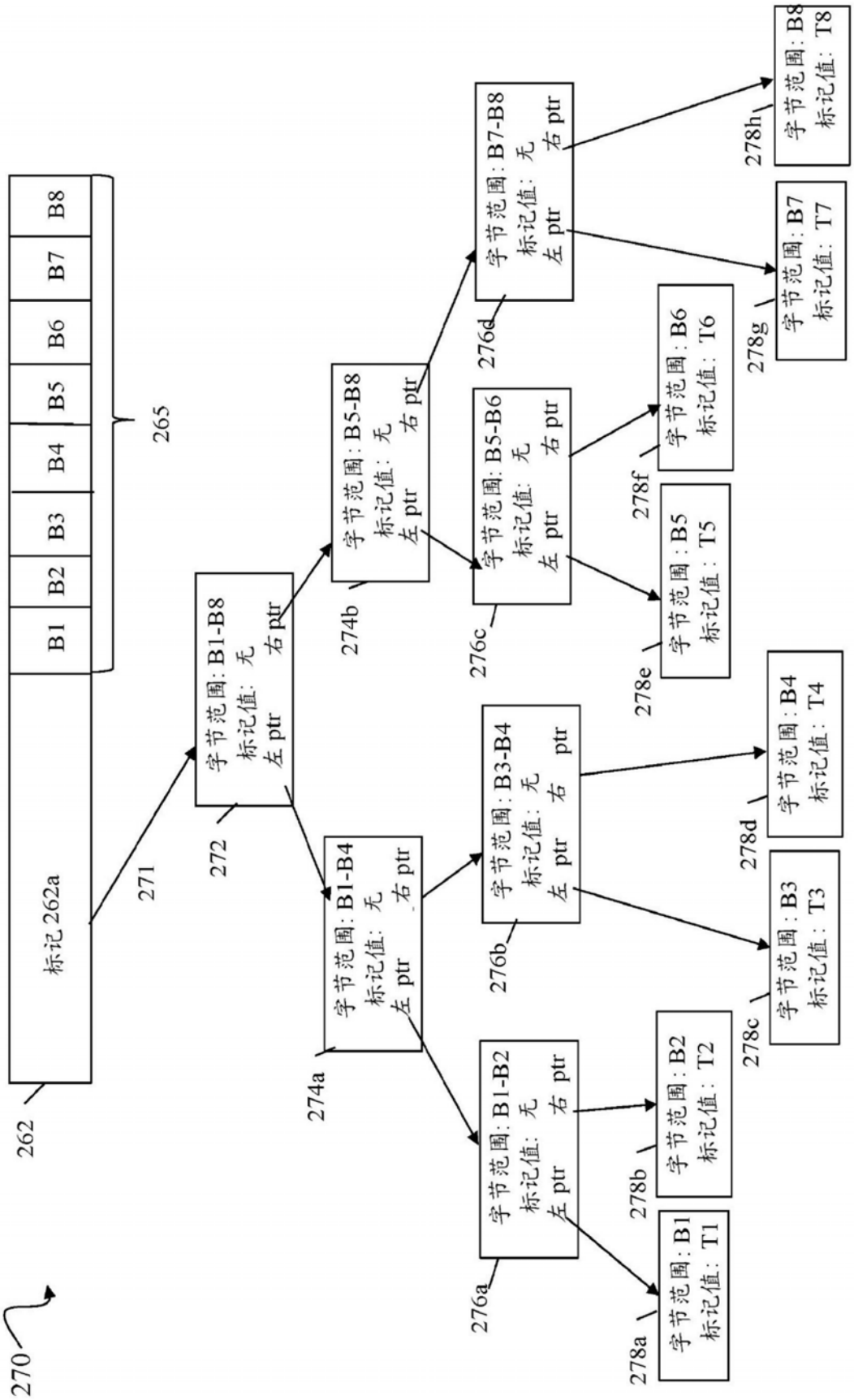

270
262
标记 262a
B1
B2
B3
B4
B5
B6
B7
B8
265
271
272
字节范围: B1-B8
标记值: 无
左 ptr 右 ptr
274a
字节范围: B1-B4
标记值: 无
左 ptr 右 ptr
274b
字节范围: B5-B8
标记值: 无
左 ptr 右 ptr
276a
字节范围: B1-B2
标记值: 无
左 ptr 右 ptr
276b
字节范围: B3-B4
标记值: 无
左 ptr 右 ptr
276c
字节范围: B5-B6
标记值: 无
左 ptr 右 ptr
276d
字节范围: B7-B8
标记值: 无
左 ptr 右 ptr
278a
字节范围: B1
标记值: T1
278b
字节范围: B2
标记值: T2
278c
字节范围: B3
标记值: T3
278d
字节范围: B4
标记值: T4
278e
字节范围: B5
标记值: T5
278f
字节范围: B6
标记值: T6
278g
字节范围: B7
标记值: T7
278h
字节范围: B8
标记值: T8

图41

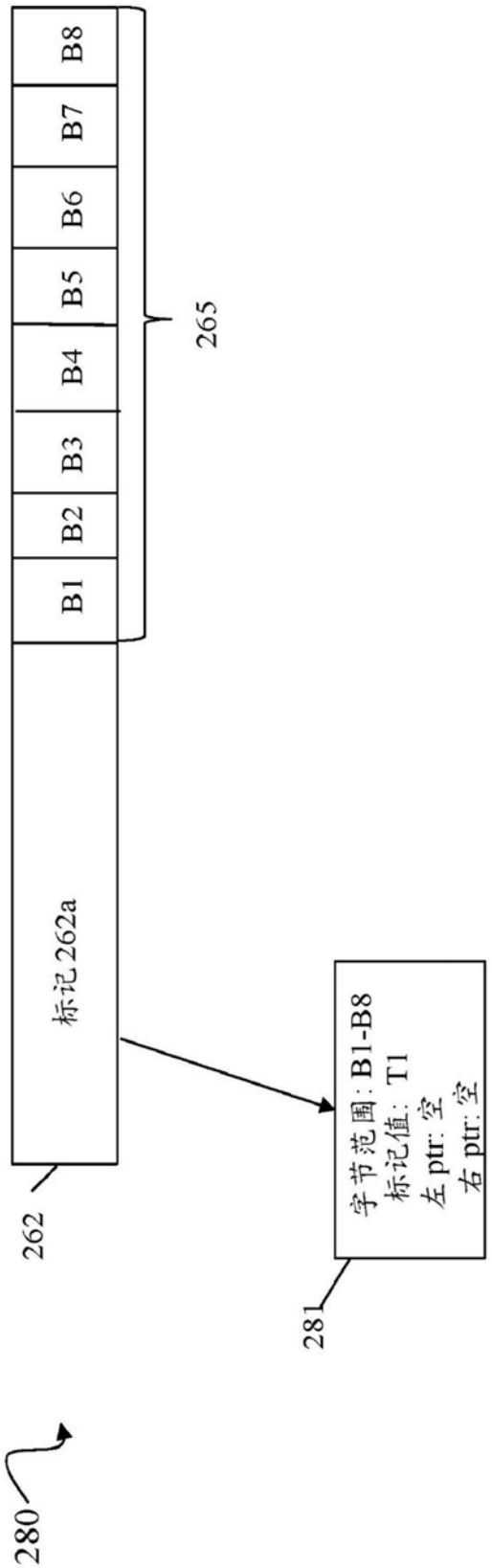

B8
B7
B6
B5
B4
B3
B2
B1
265
标记 262a
262
字节范围: B1-B8
标记值: T1
左 ptr: 空
右 ptr: 空
281
280

图42

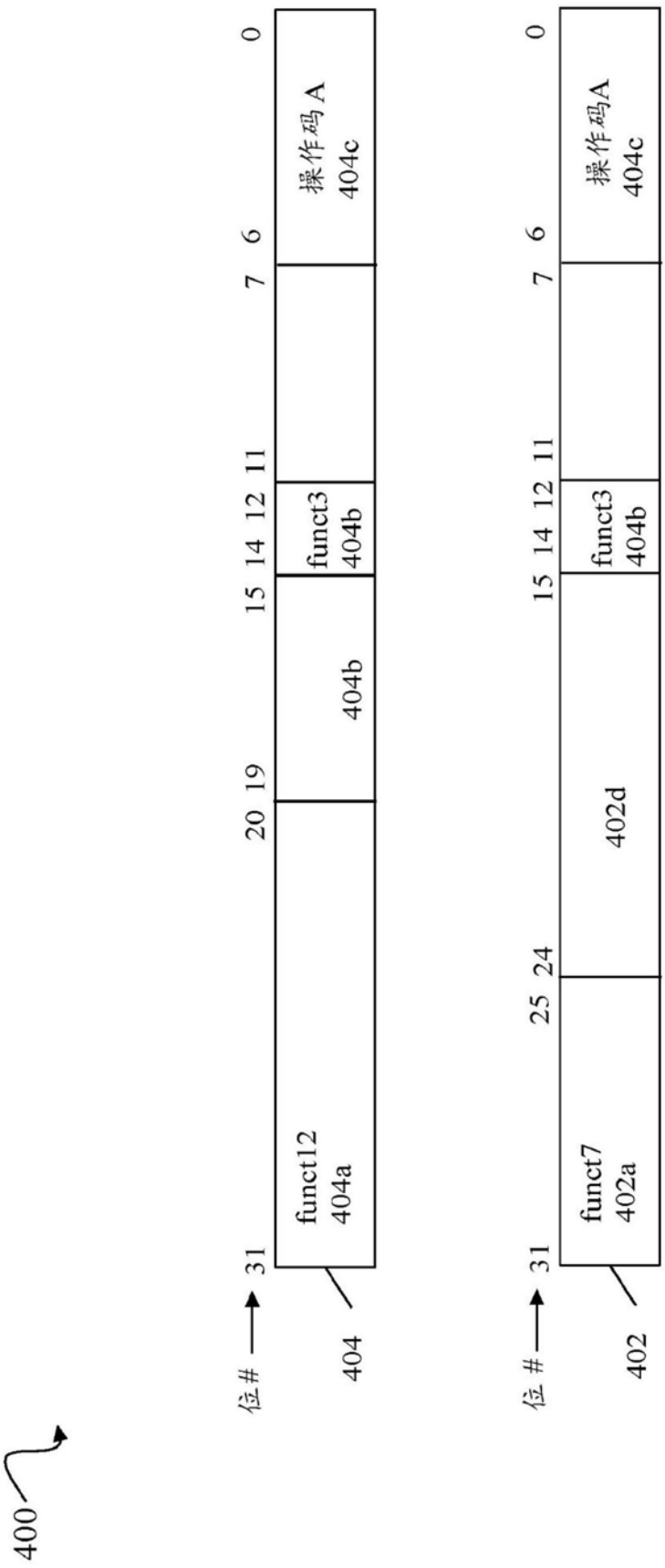

400
位 #
31
20 19
15 14 12 11
7 6
0
funct12
404a
404b
funct3
404b
操作码 A
404c
404
位 #
31
25 24
15 14 12 11
7 6
0
funct7
402a
402d
funct3
404b
操作码A
404c
402

图43

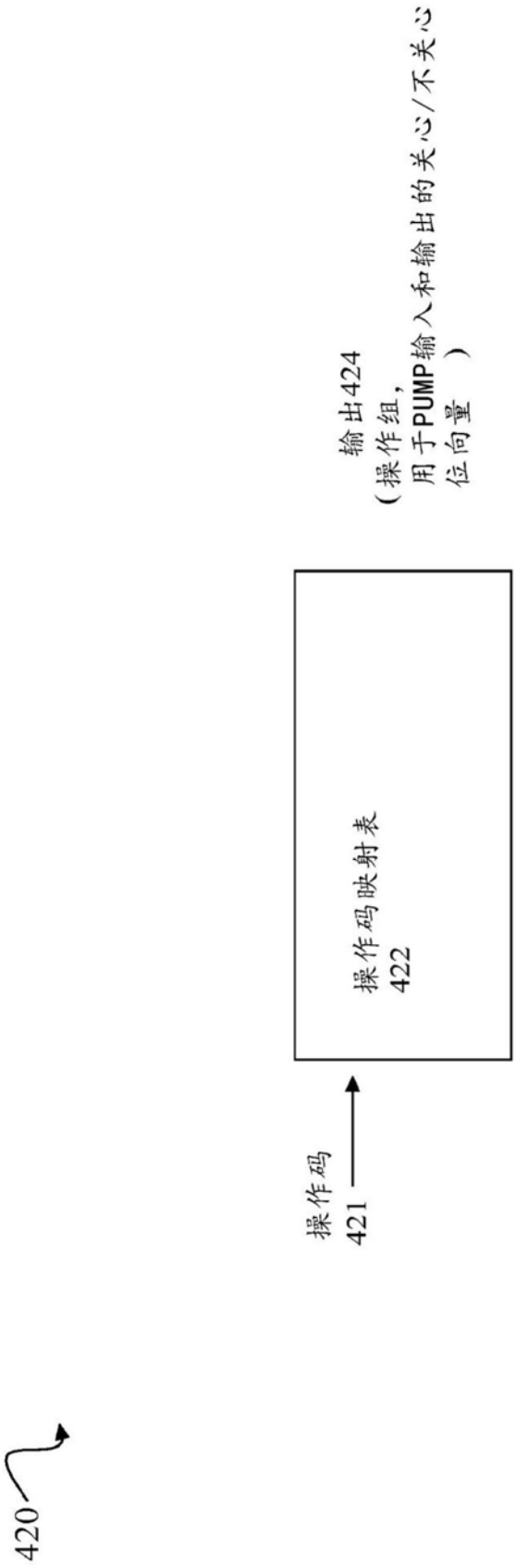


图44

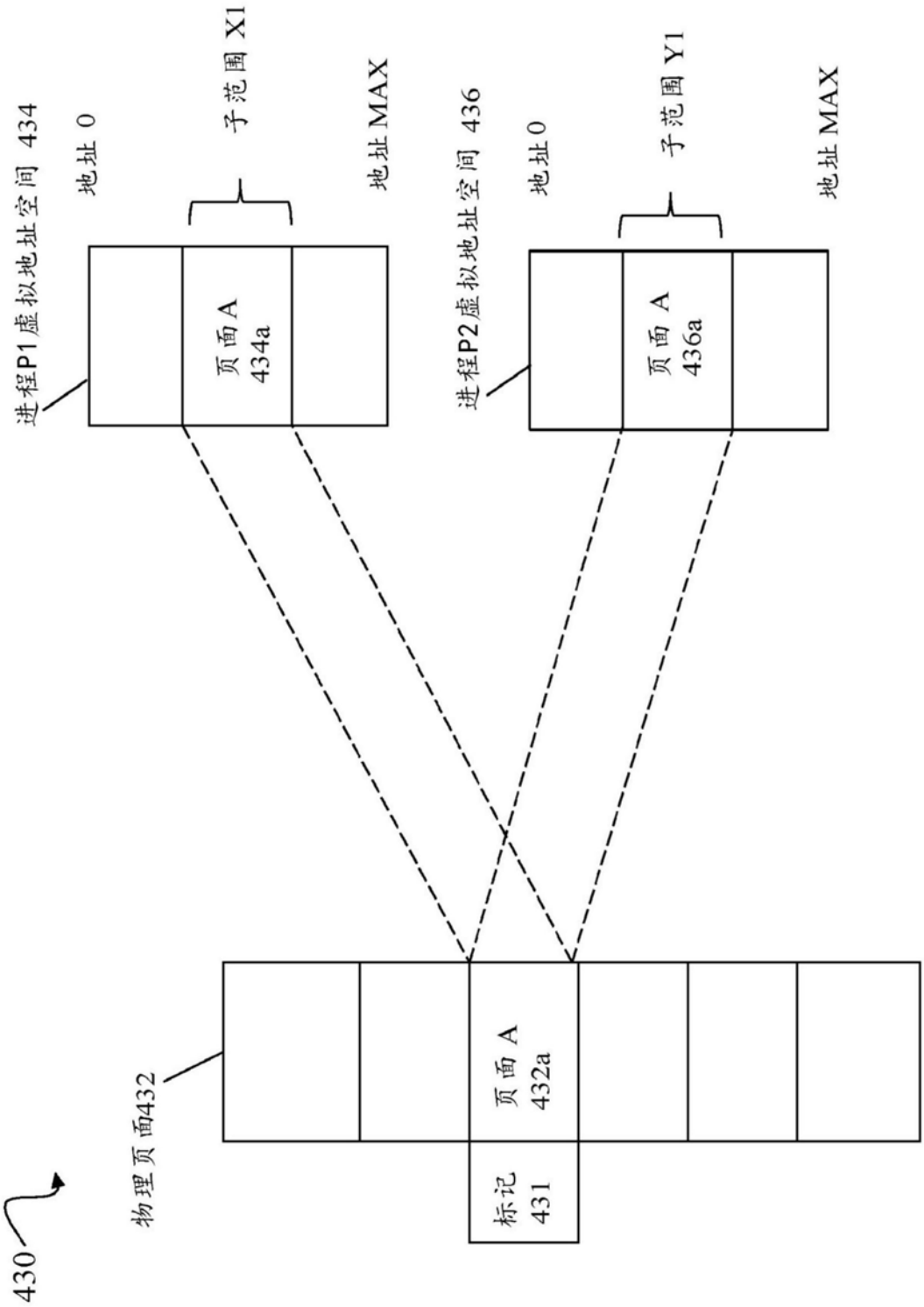

进程P1虚拟地址空间 434
地址 0
子范围 X1
地址 MAX
页面 A
434a
进程P2虚拟地址空间 436
地址 0
子范围 Y1
地址 MAX
页面 A
436a
物理页面432
页面 A
432a
标记
431
430

图45

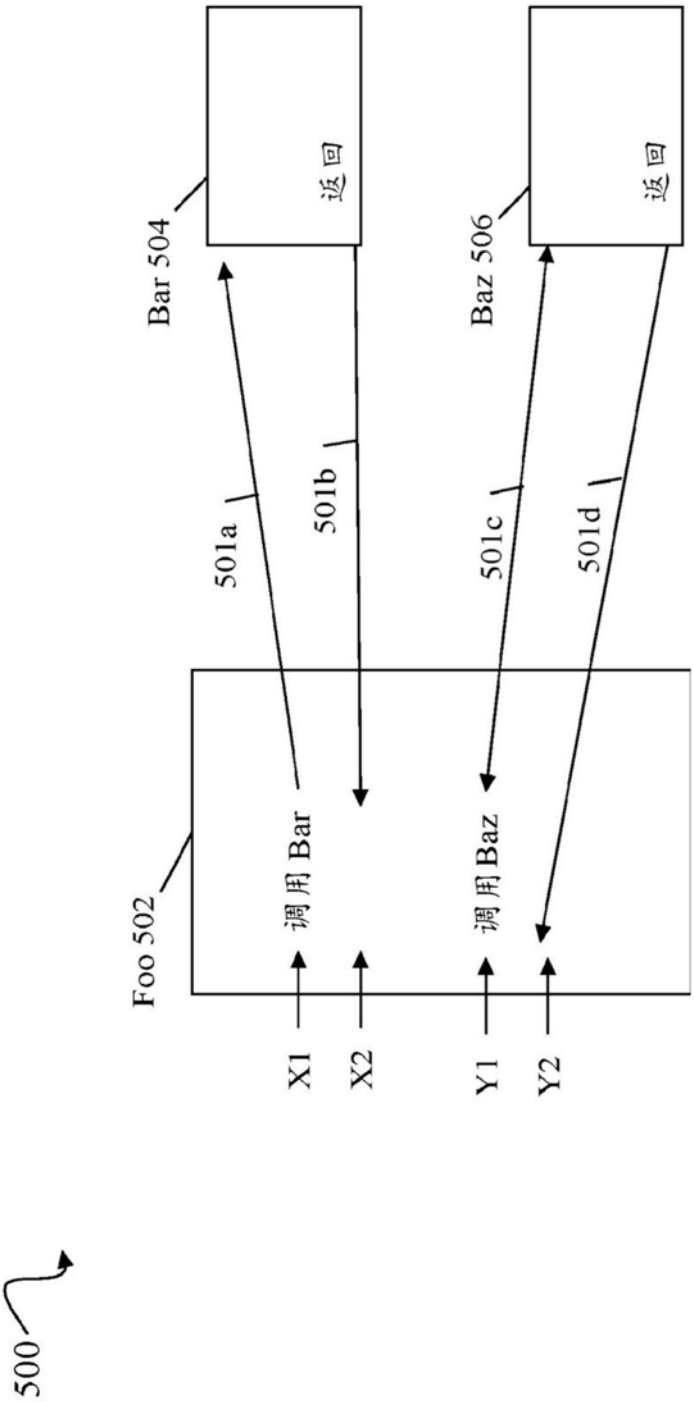

Bar 504
返回
Baz 506
返回
501a
501b
501c
501d
Foo 502
调用 Bar
调用 Baz
X1
X2
Y1
Y2
500

图46

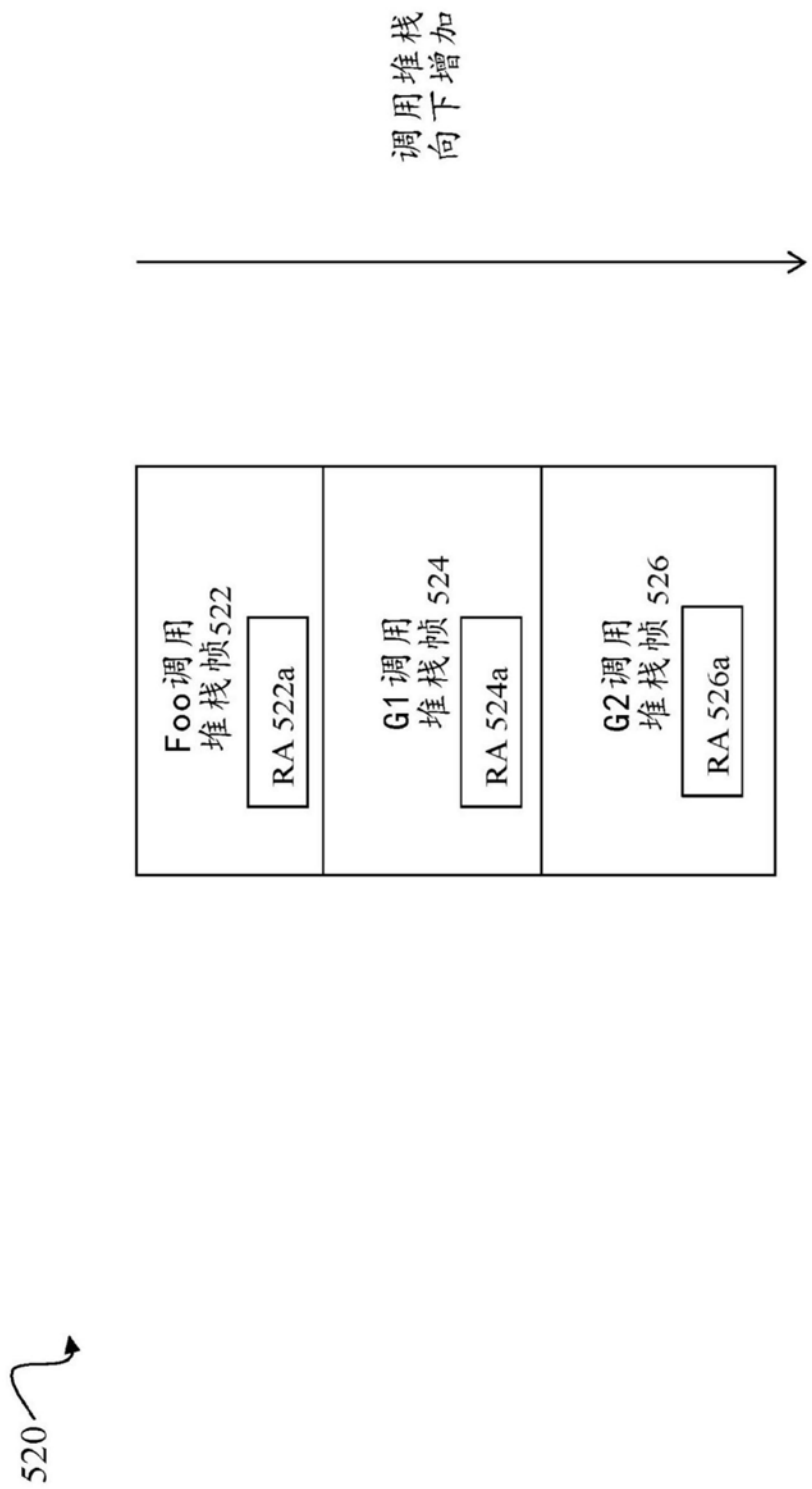

调用堆栈
向下增加
Foo调用
堆栈帧522
RA 522a
G1调用
堆栈帧 524
RA 524a
G2调用
堆栈帧 526
RA 526a
520

图47

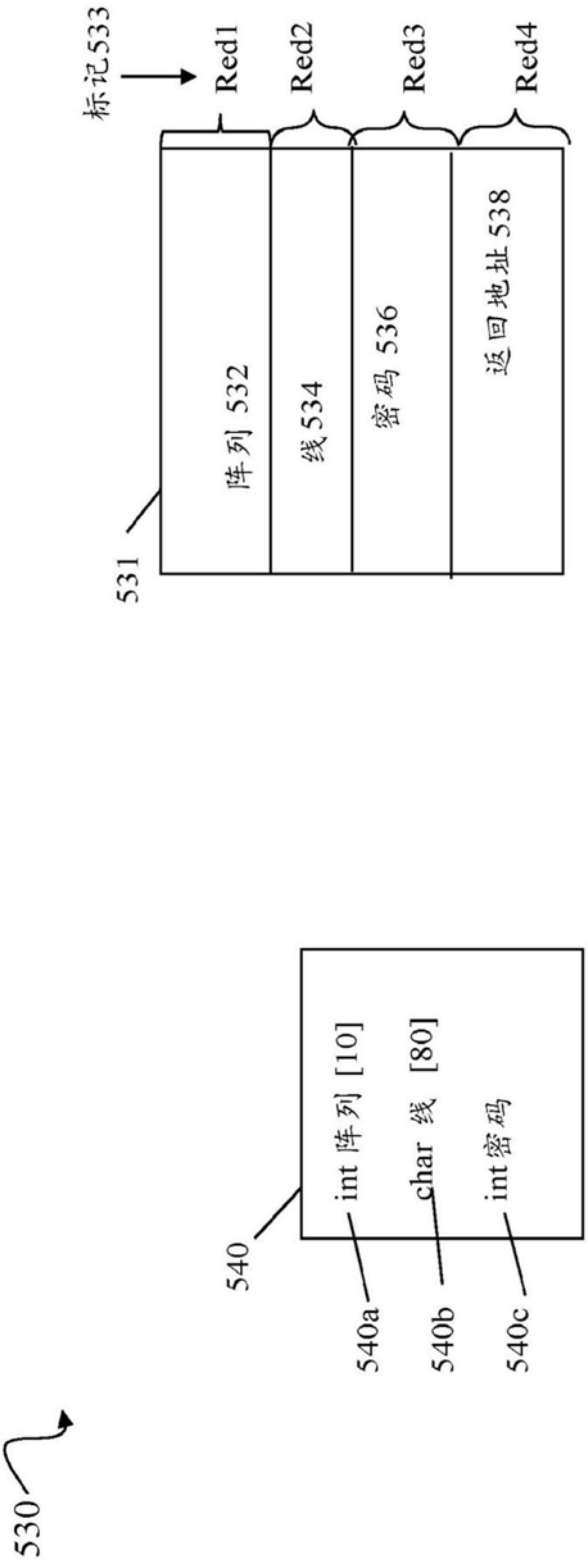

标记533
Red1
Red2
Red3
Red4
阵列 532
线534
密码 536
返回地址 538
531
int 阵列 [10]
char 线 [80]
int 密码
540
540a
540b
540c
530

图48

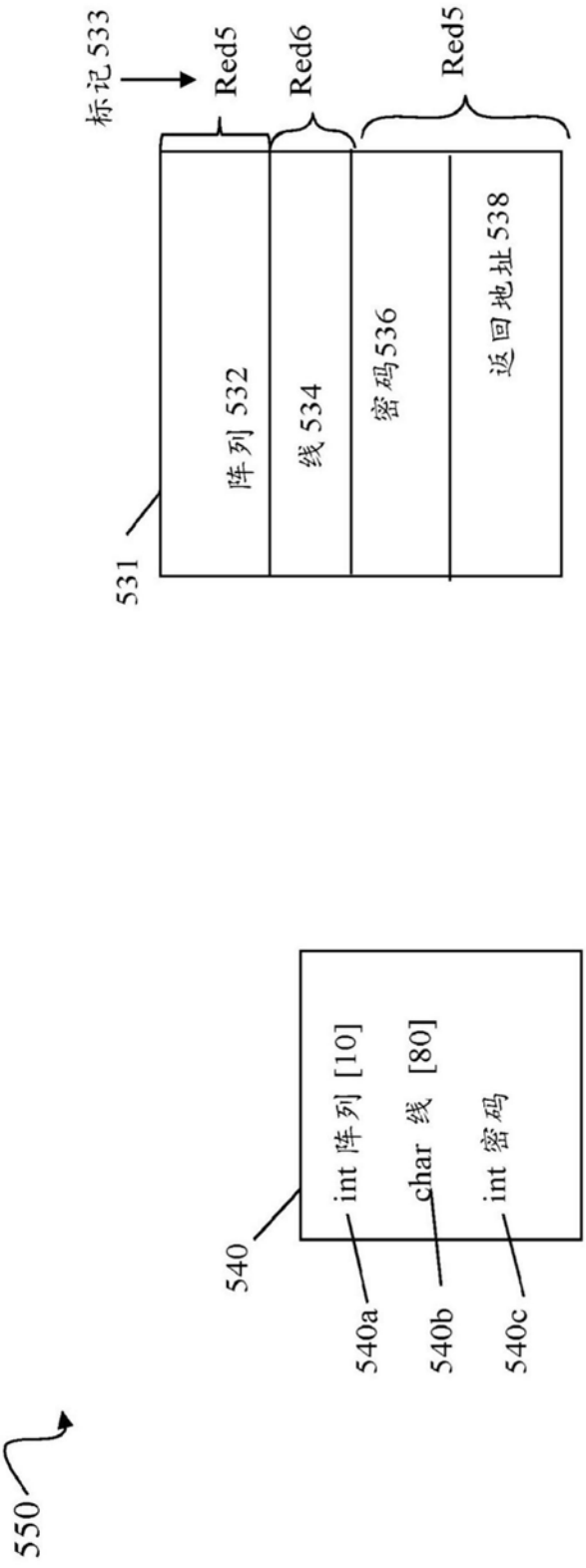

标记533
Red5
Red6
Red5
阵列 532
线 534
密码536
返回地址 538
531
550
540
int 阵列 [10]
char 线 [80]
int 密码
540a
540b
540c

图49

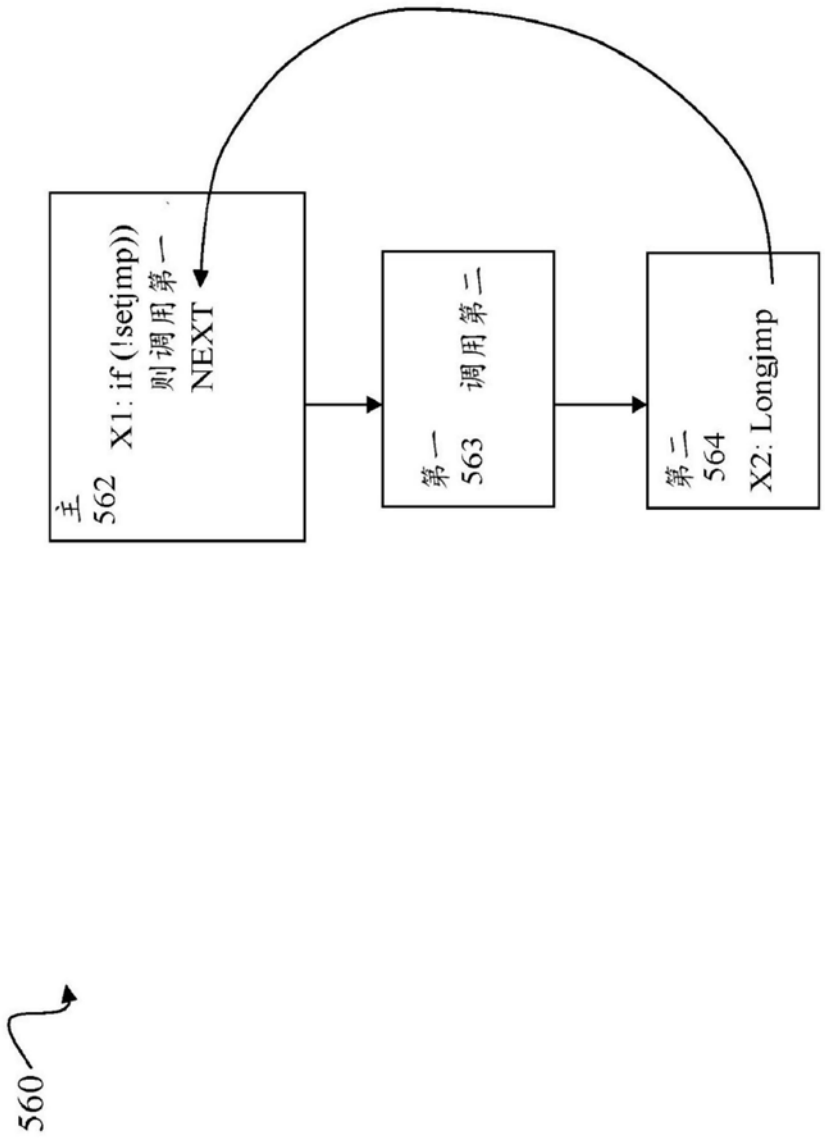


图50

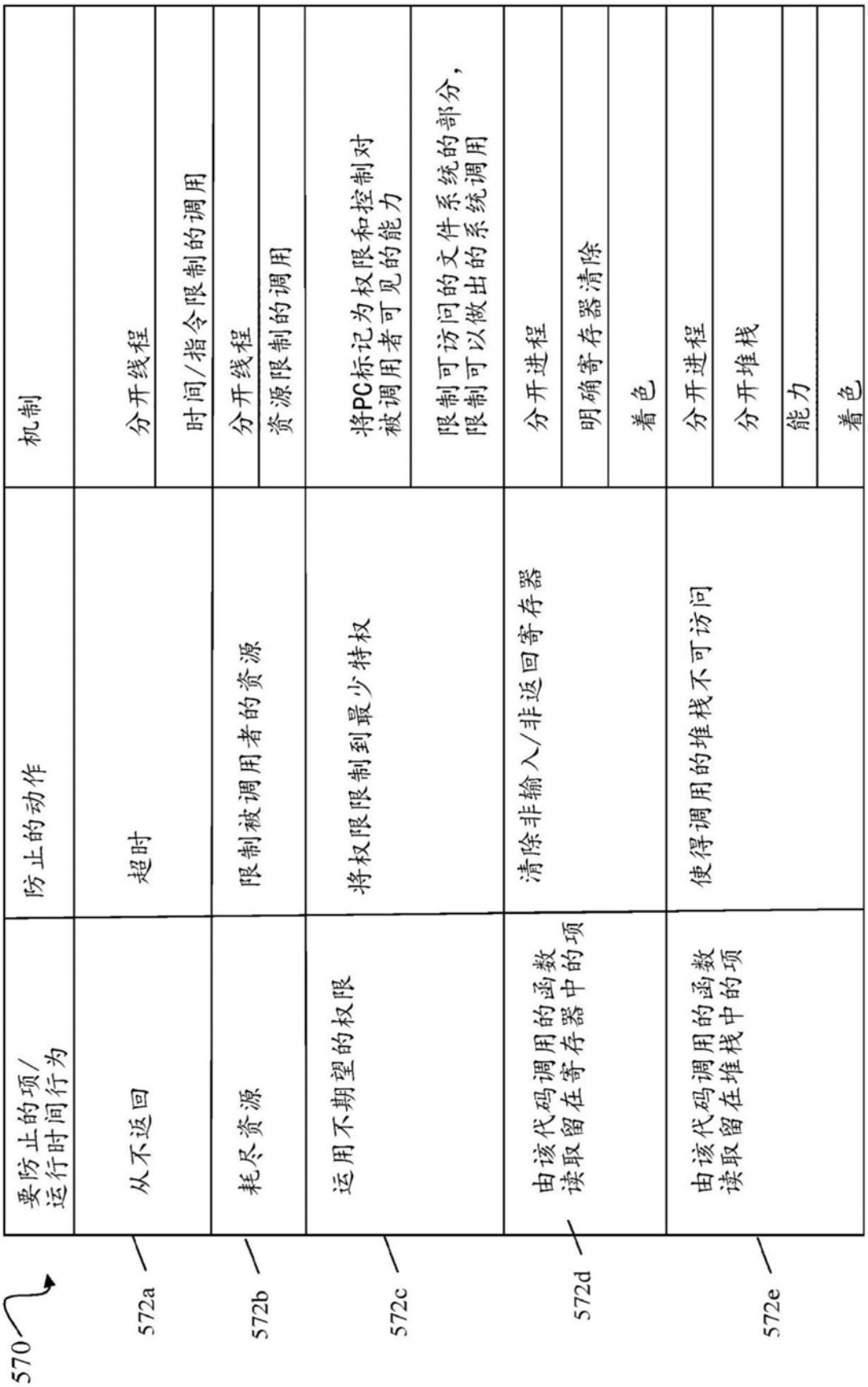

570
572a
572b
572c
572d
572e
要防止的项/运行时间行为
防止的动作
机制
从不返回
超时
分开线程
时间/指令限制的调用
耗尽资源
限制被调用者的资源
分开线程
资源限制的调用
运用不期望的权限
将权限限制到最少特权
将PC标记为权限和控制对被调用者可见的能力
限制可访问的文件系统的部分，限制可以做出的系统调用
由该代码调用的函数读取留在寄存器中的项
清除非输入/非返回寄存器
分开进程
明确寄存器清除
着色
由该代码调用的函数读取留在堆栈中的项
使得调用的堆栈不可访问
分开进程
分开堆栈
能力
着色

图51

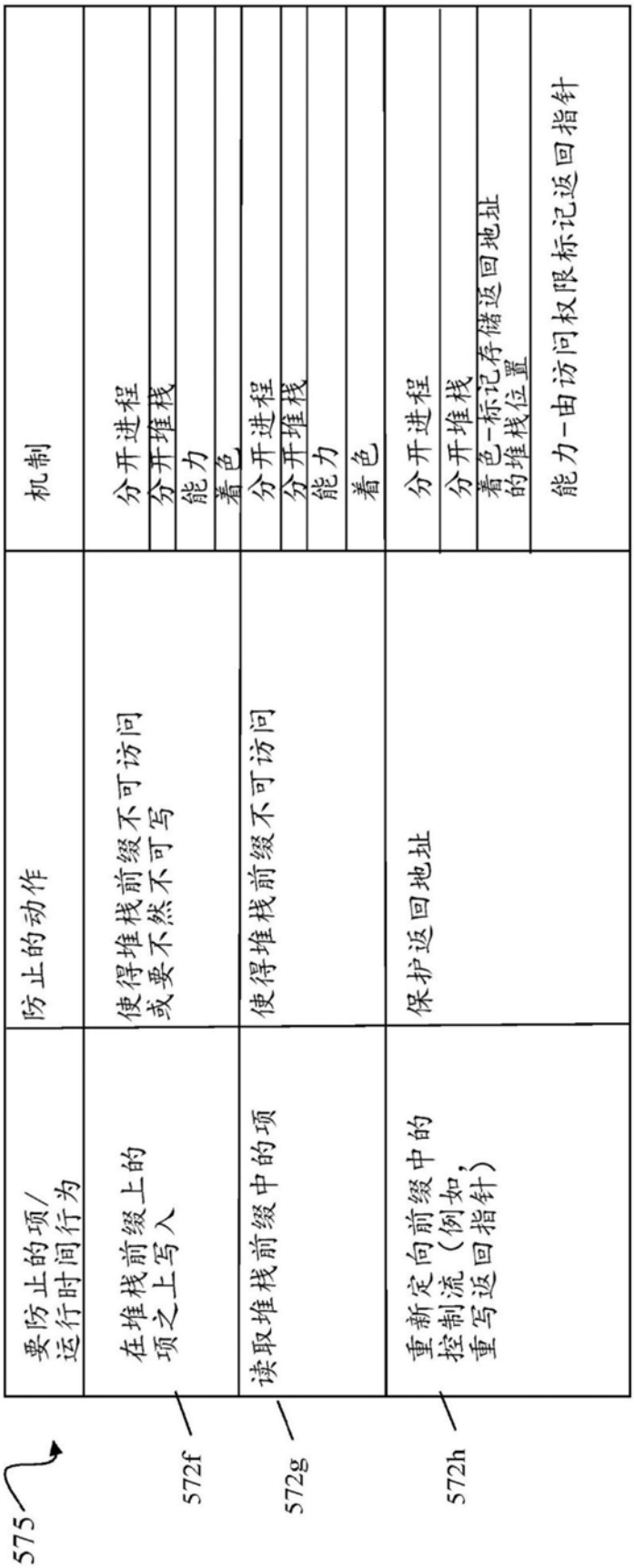

575
572f
572g
572h
要防止的项/运行时间行为
防止的动作
机制
在堆栈前缀上的项之上写入
使得堆栈前缀不可访问或要不然不可写
分开进程
分开堆栈
能力
着色
读取堆栈前缀中的项
使得堆栈前缀不可访问
分开进程
分开堆栈
能力
着色
重新定向前缀中的控制流（例如，重写返回指针）
保护返回地址
分开进程
分开堆栈
着色-标记存储返回地址的堆栈位置
能力-由访问权限标记返回指针

图52

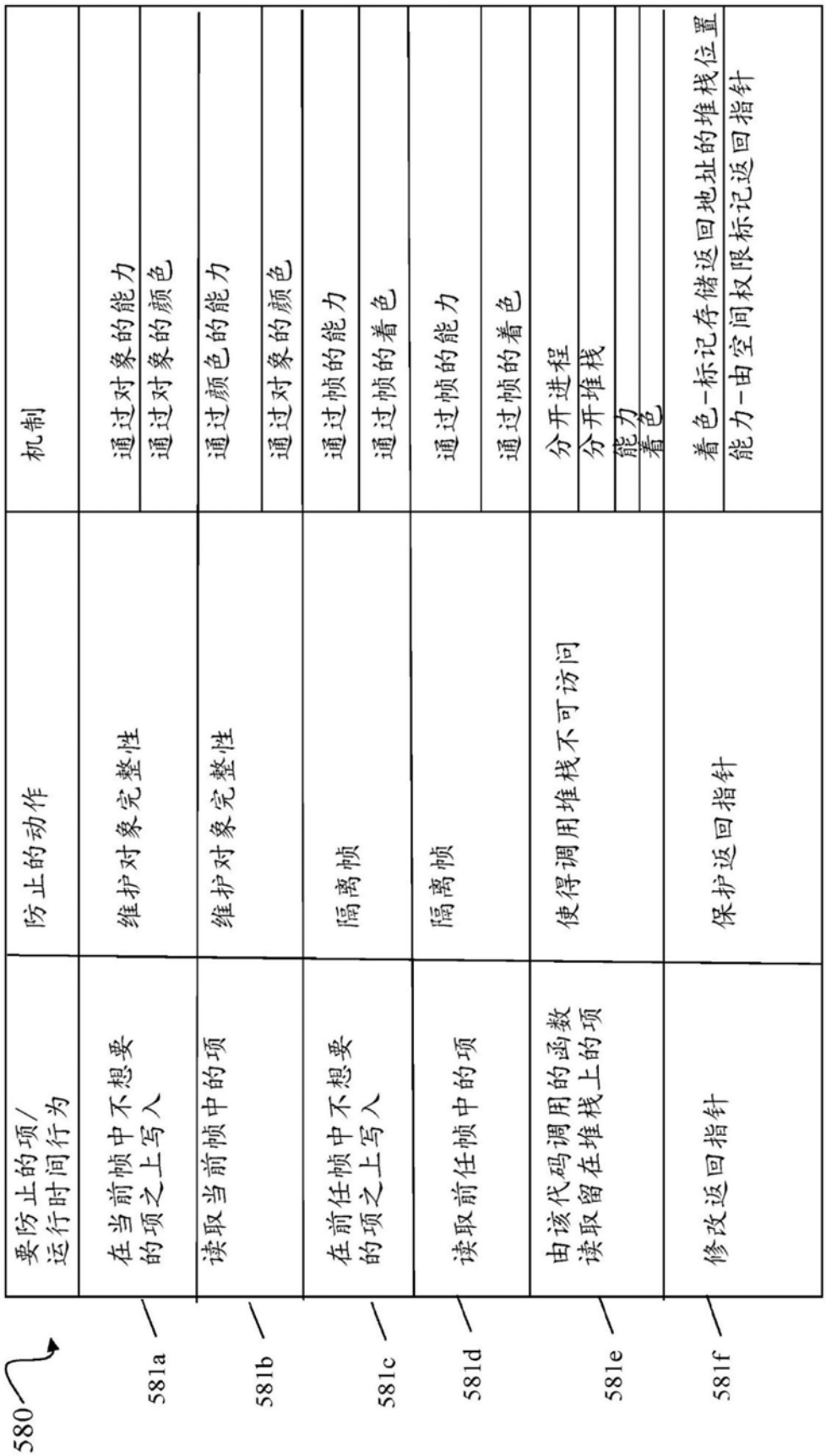

580
581a
581b
581c
581d
581e
581f
要防止的项/运行时间行为
防止的动作
机制
在当前帧中不想要的项之上写入
维护对象完整性
通过对象的能力
通过对象的颜色
读取当前帧中的项
维护对象完整性
通过颜色的能力
通过对象的颜色
在前任帧中不想要的项之上写入
隔离帧
通过帧的能力
通过帧的着色
读取前任帧中的项
隔离帧
通过帧的能力
通过帧的着色
由该代码调用的函数读取留在堆栈上的项
使得调用堆栈不可访问
分开进程
分开堆栈
能力
着色
修改返回指针
保护返回指针
着色-标记存储返回地址的堆栈位置
能力-由空间权限标记返回指针

图53

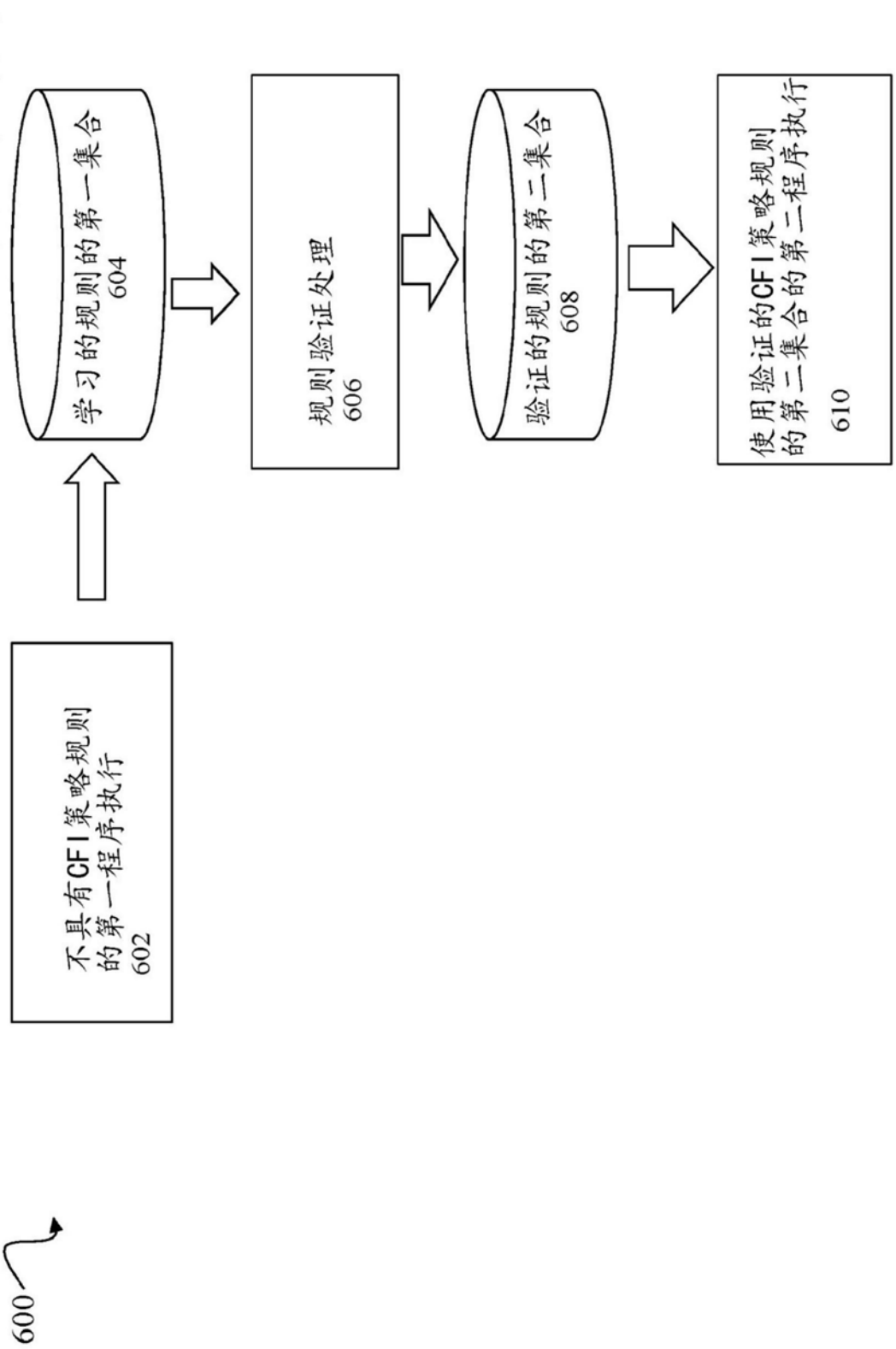

以无CFI策略规则运行程序
不具有CFI策略规则的第一程序执行
602
学习的规则的第一集合
604
规则验证处理
606
验证的规则的第二集合
608
使用验证的CFI策略规则的第二集合的第二程序执行
610
600

图54

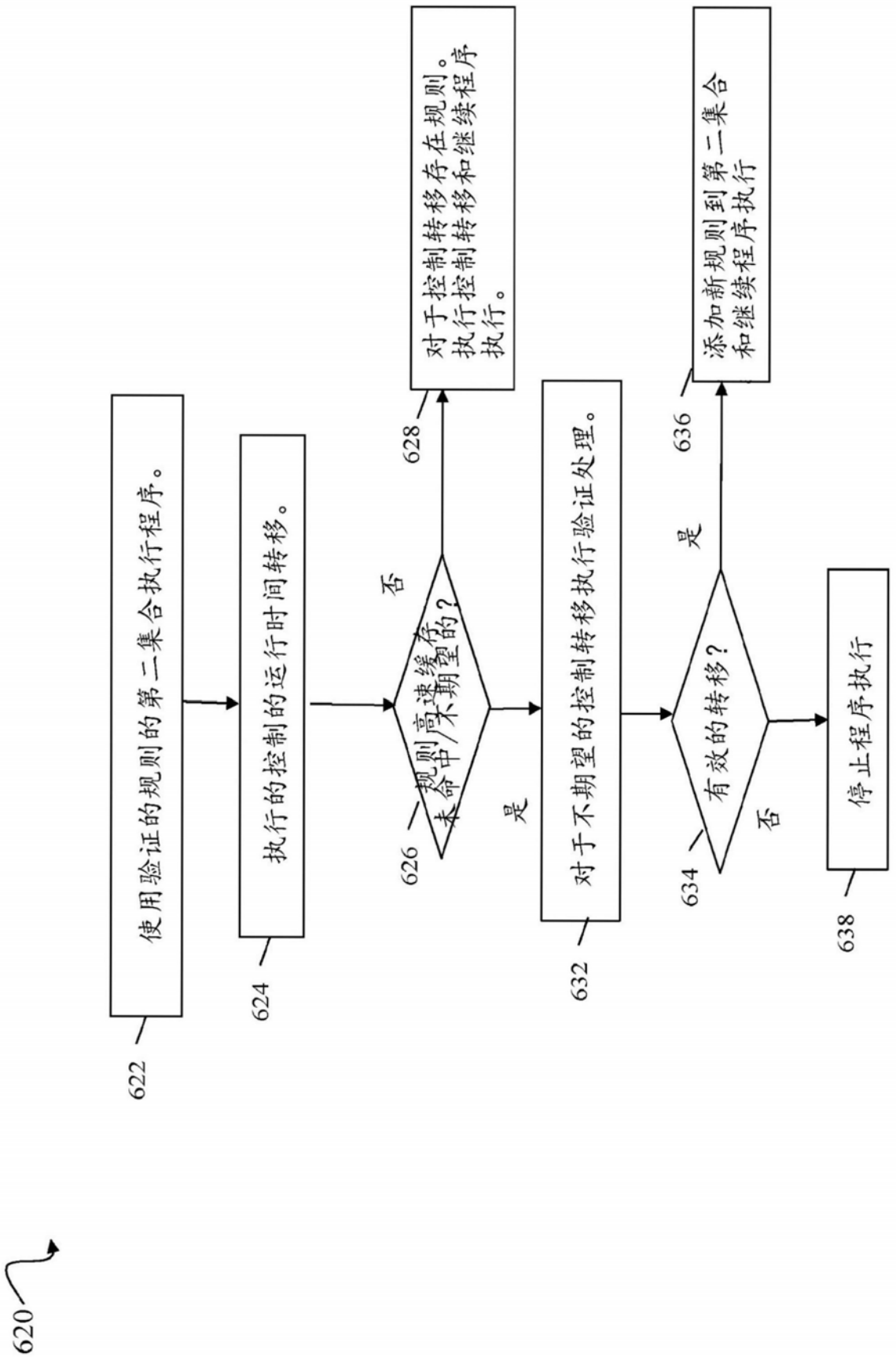

620
使用验证的规则的第二集合执行程序。
622
执行的控制的运行时间转移。
624
规则高速缓存未命中/不期望的?
626
否
是
对于控制转移存在规则。执行控制转移和继续程序执行。
628
对于不期望的控制转移执行验证处理。
632
有效的转移?
634
否
是
添加新规则到第二集合和继续程序执行
636
停止程序执行
638

图55

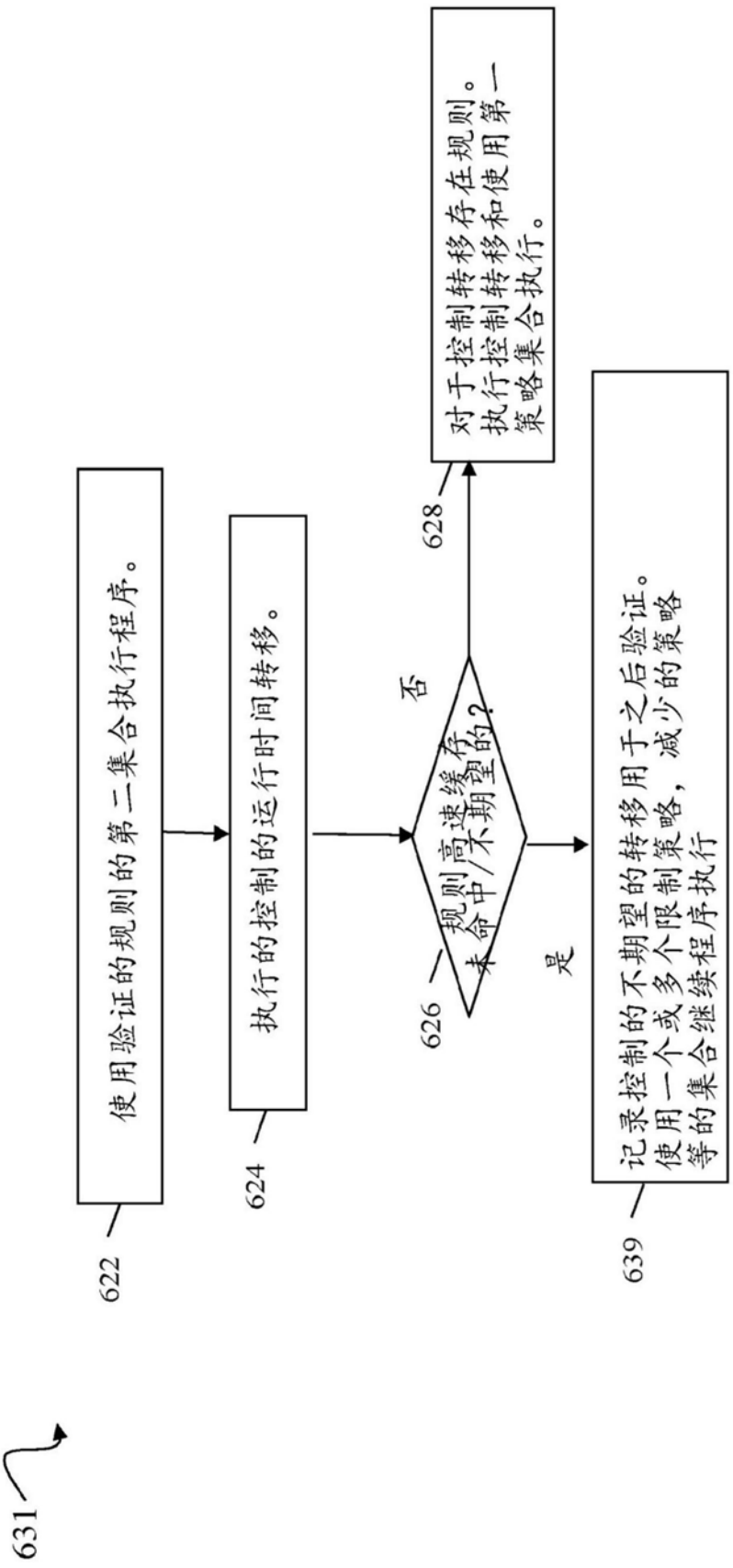

631
622
使用验证的规则的第二集合执行程序。
624
执行的控制的运行时间转移。
626
规则高速缓存未命中/不期望的?
否
628
对于控制转移存在规则。执行控制转移和使用第一策略集合执行。
是
639
记录控制的不期望的转移用于之后验证。使用一个或多个限制策略，减少的策略等的集合继续程序执行

图56

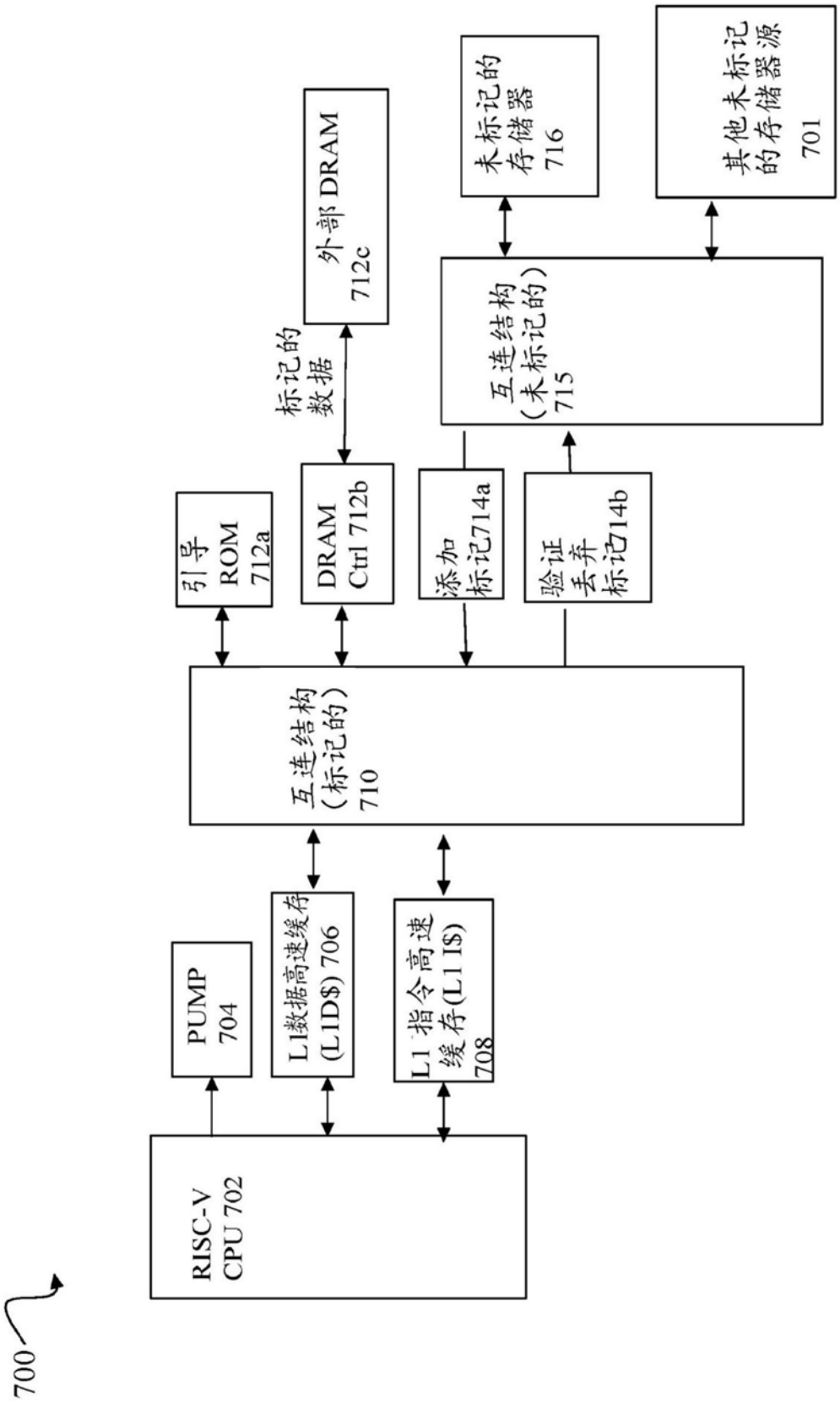

700
RISC-V CPU 702
PUMP 704
L1数据高速缓存 (L1D$) 706
L1 指令高速缓存(L1 I$) 708
互连结构 (标记的) 710
引导 ROM 712a
DRAM Ctrl 712b
外部 DRAM 712c
标记的数据
添加标记714a
验证丢弃标记14b
互连结构 (未标记的) 715
未标记的存储器 716
其他未标记的存储器源 701

图57

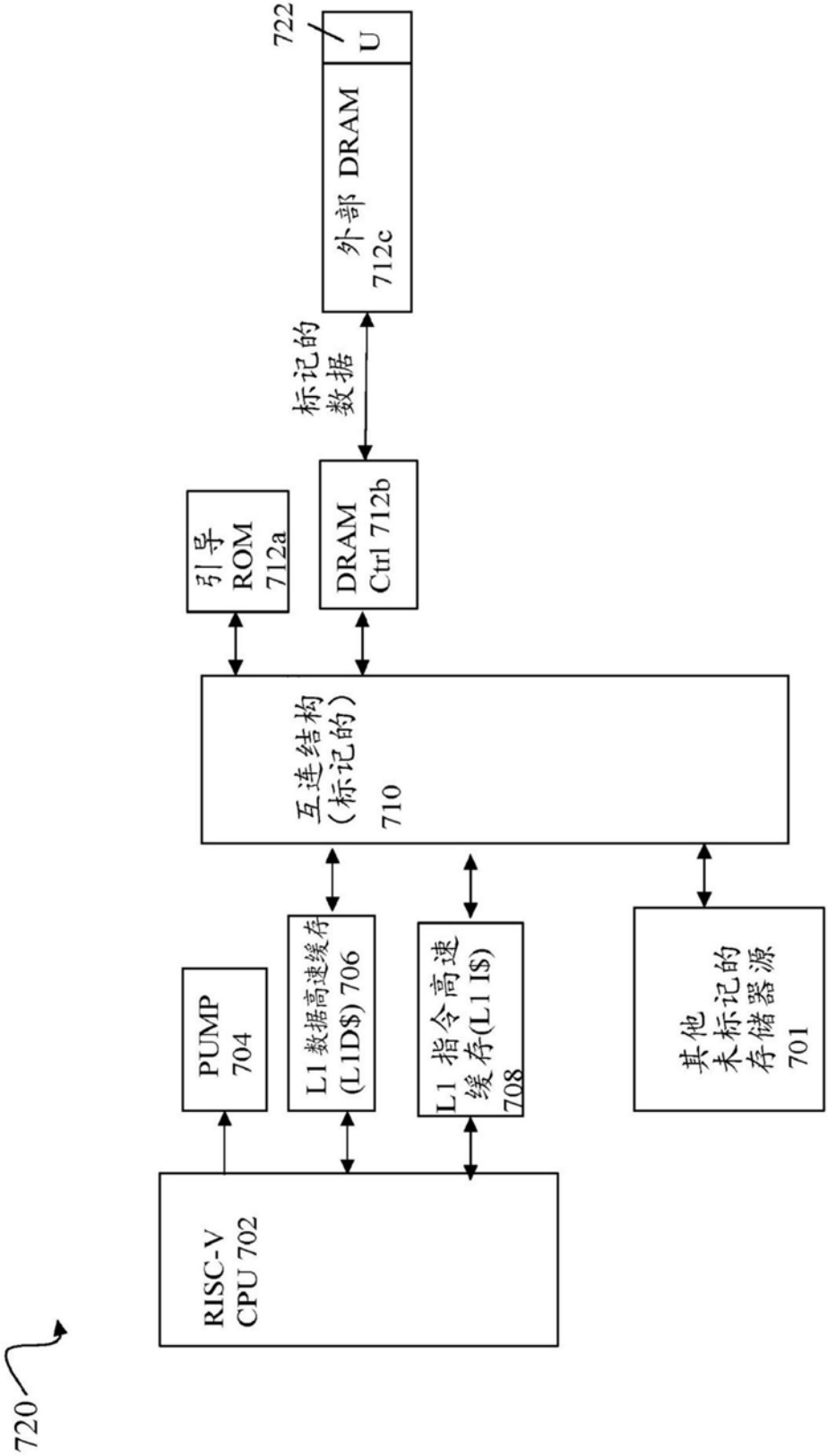

720
RISC-V CPU 702
PUMP 704
L1 数据高速缓存 (L1D$) 706
L1 指令高速缓存(L1 I$) 708
互连结构 (标记的) 710
引导 ROM 712a
DRAM Ctrl 712b
标记的数据
外部 DRAM 712c
722
U
其他未标记的存储器源 701

图58

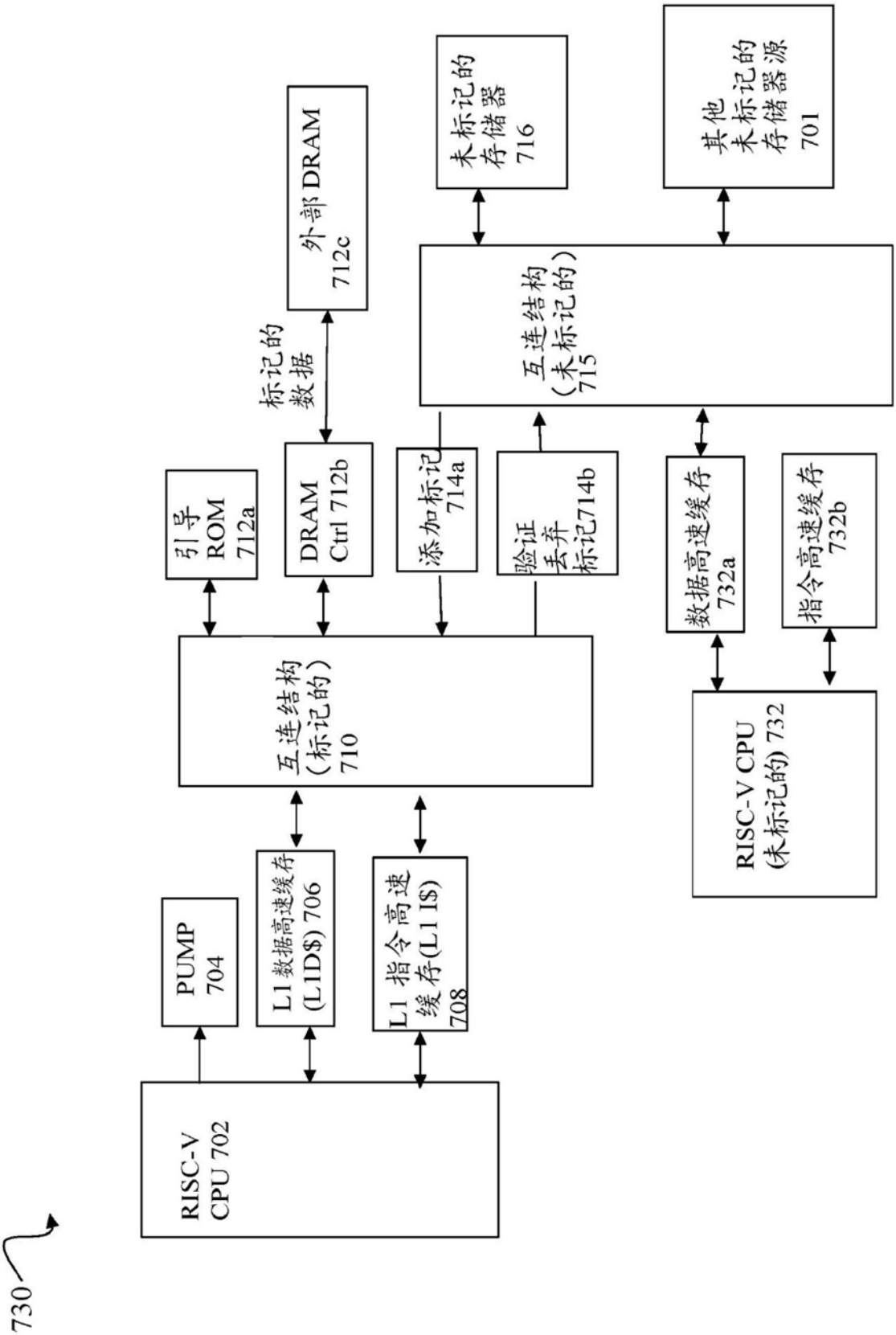

730
RISC-V CPU 702
PUMP 704
L1 数据高速缓存 (L1D$) 706
L1 指令高速缓存(L1 I$) 708
互连结构 (标记的) 710
引导 ROM 712a
DRAM Ctrl 712b
外部 DRAM 712c
标记的数据
添加标记 714a
验证丢弃标记714b
互连结构 (未标记的) 715
未标记的存储器 716
其他未标记的存储器源 701
数据高速缓存 732a
指令高速缓存 732b
RISC-V CPU (未标记的) 732

图59

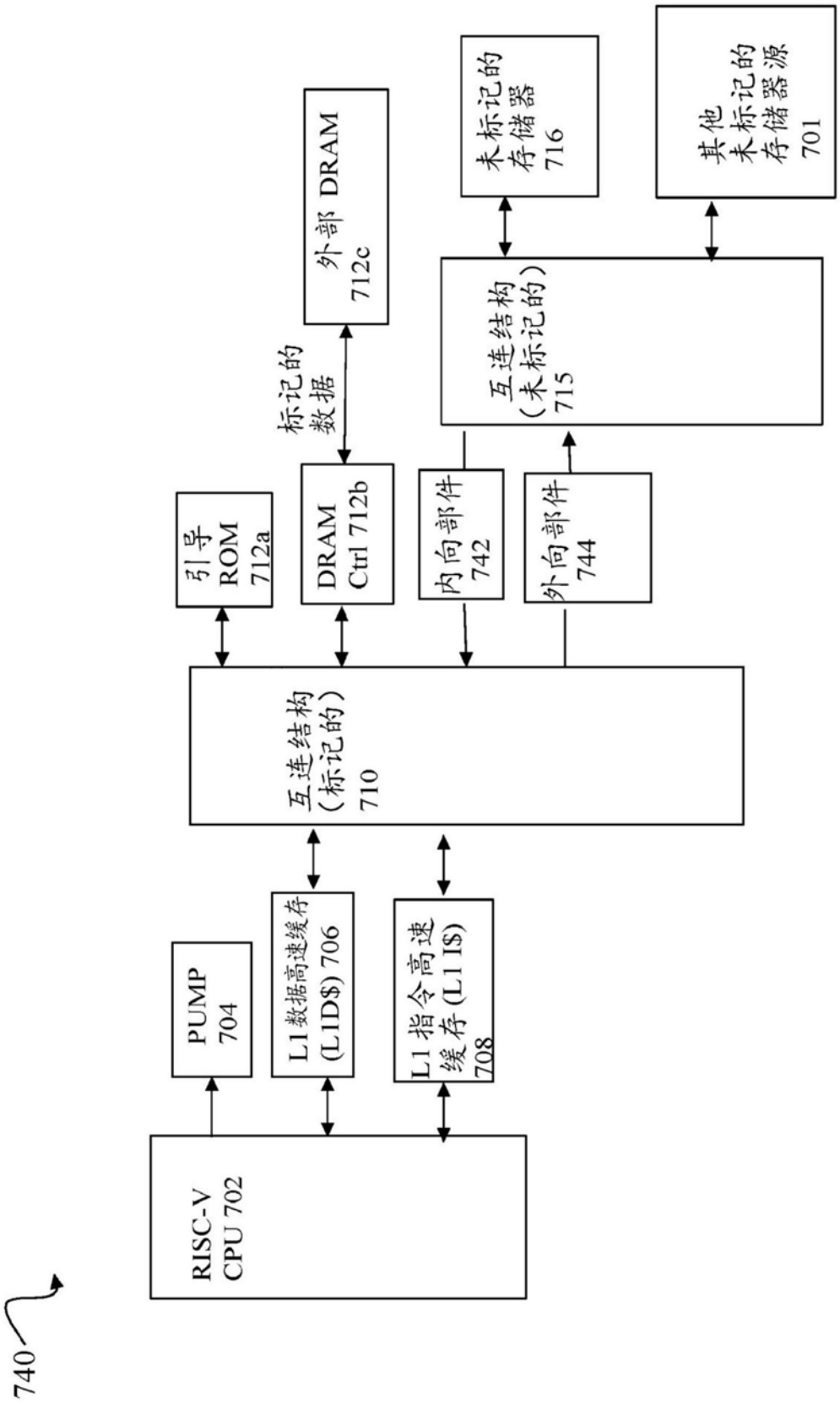

740
RISC-V CPU 702
PUMP 704
L1数据高速缓存 (L1D$) 706
L1指令高速缓存 (L1 I$) 708
互连结构 (标记的) 710
引导 ROM 712a
DRAM Ctrl 712b
外部 DRAM 712c
标记的数据
内向部件 742
外向部件 744
互连结构 (未标记的) 715
未标记的存储器 716
其他未标记的存储器源 701

图60

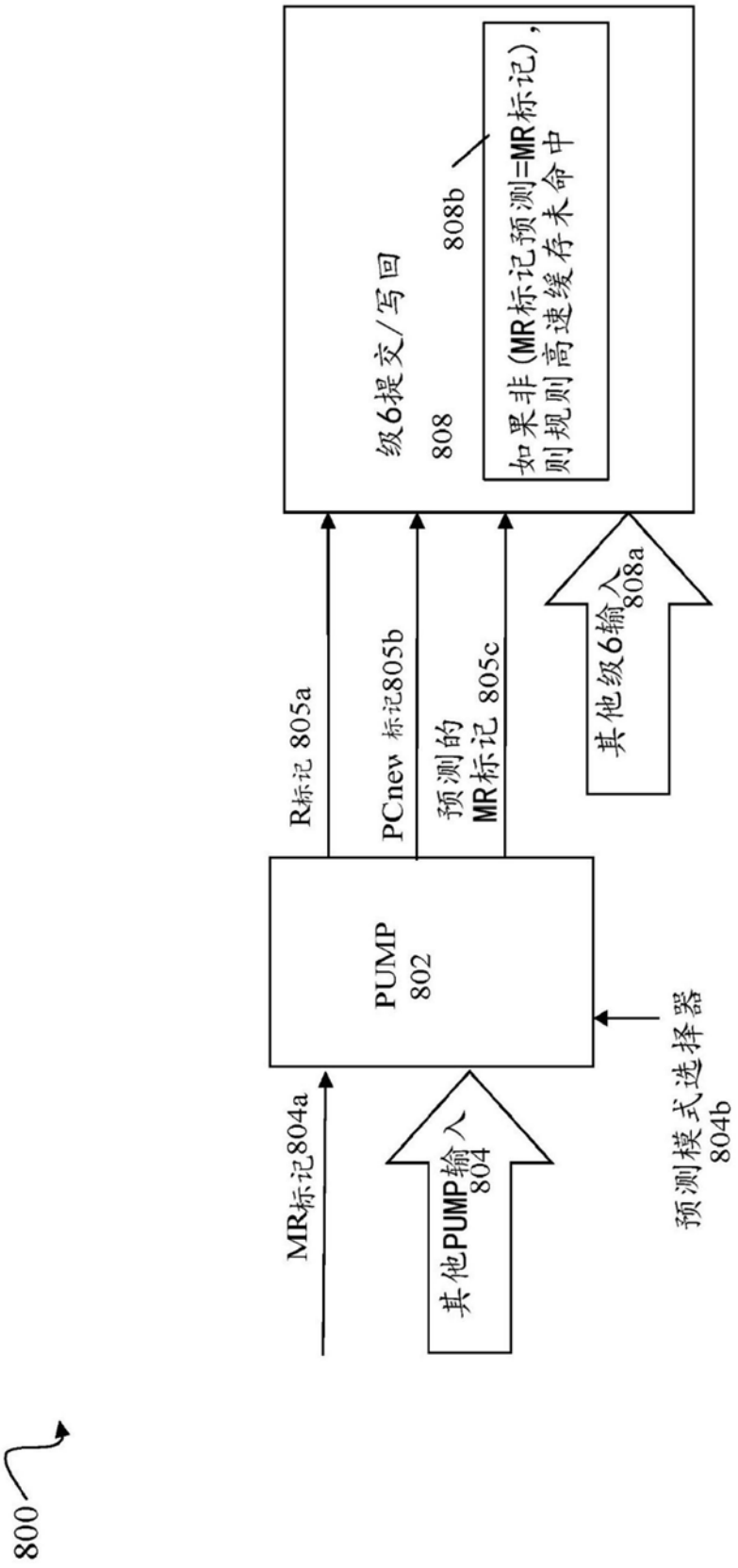

800
MR标记804a
其他PUMP输入
804
PUMP
802
预测模式选择器
804b
R标记 805a
PCnew 标记805b
预测的
MR标记 805c
其他级6输入
808a
级6提交/写回
808
808b
如果非(MR标记预测=MR标记),
则规则高速缓存未命中

图61

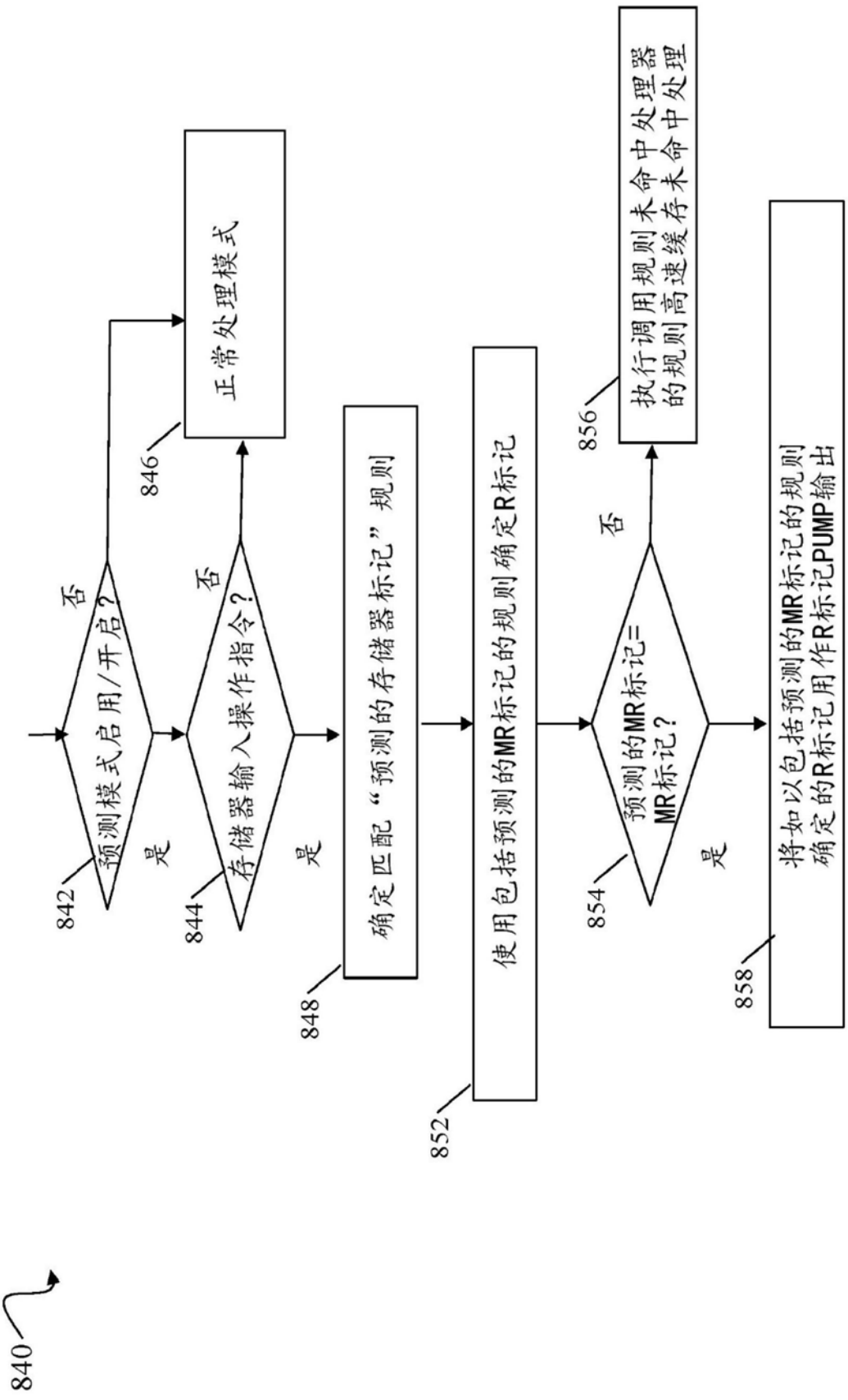

840
842
预测模式启用/开启?
否
是
844
存储器输入操作指令?
否
是
846
正常处理模式
848
确定匹配“预测的存储器标记”规则
852
使用包括预测的MR标记的规则确定R标记
854
预测的MR标记=MR标记?
否
是
856
执行调用规则未命中处理器的规则高速缓存未命中处理
858
将如以包括预测的MR标记的规则确定的R标记用作R标记PUMP输出

图62

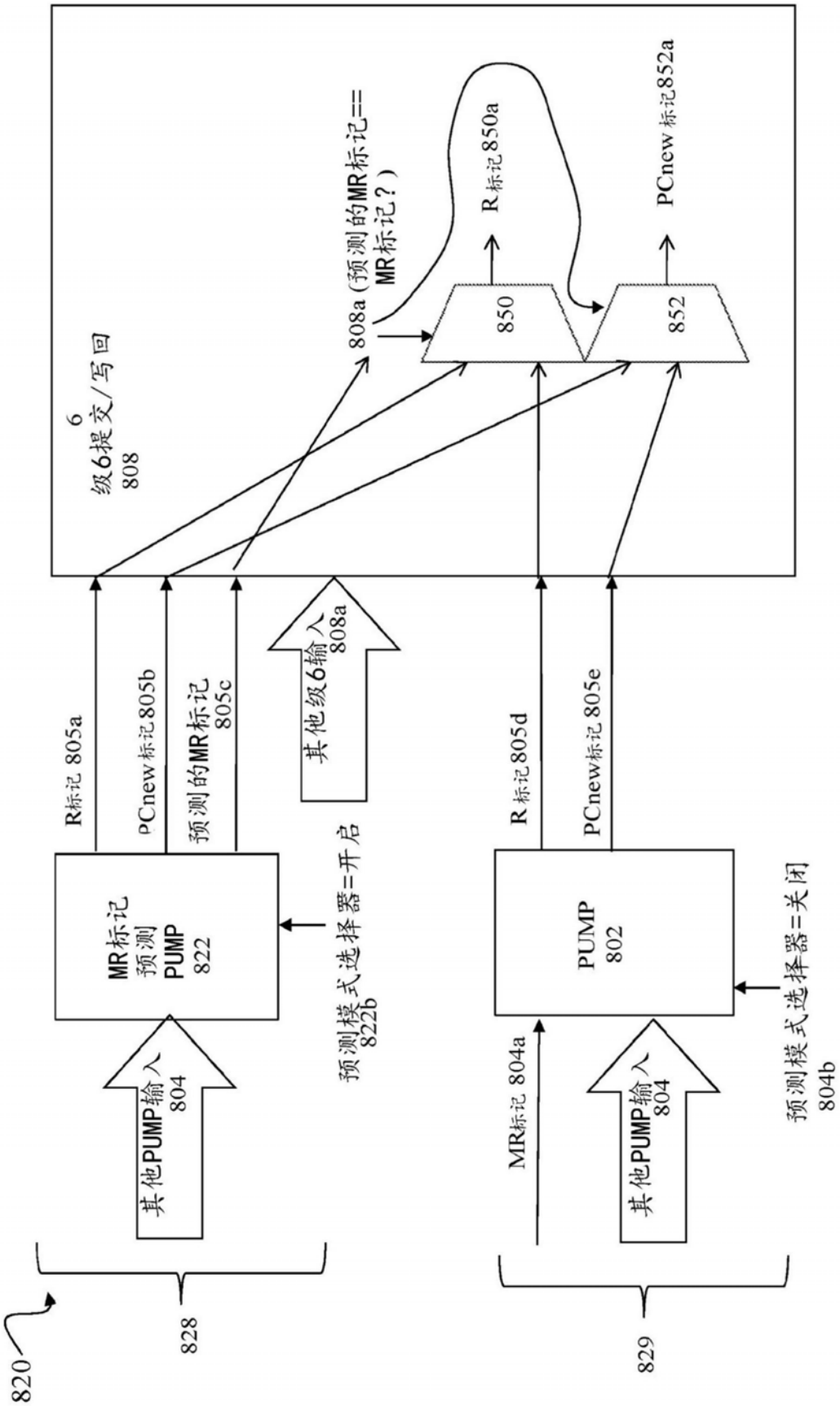

820
828
829
MR标记
预测
PUMP
822
其他PUMP输入
804
预测模式选择器=开启
822b
R标记 805a
PCnew标记805b
预测的MR标记
805c
其他级6输入
808a
6
级6提交/写回
808
808a (预测的MR标记==
MR标记？)
850
R标记850a
852
PCnew标记852a
PUMP
802
R标记805d
PCnew标记 805e
MR标记 804a
其他PUMP输入
804
预测模式选择器=关闭
804b

图63

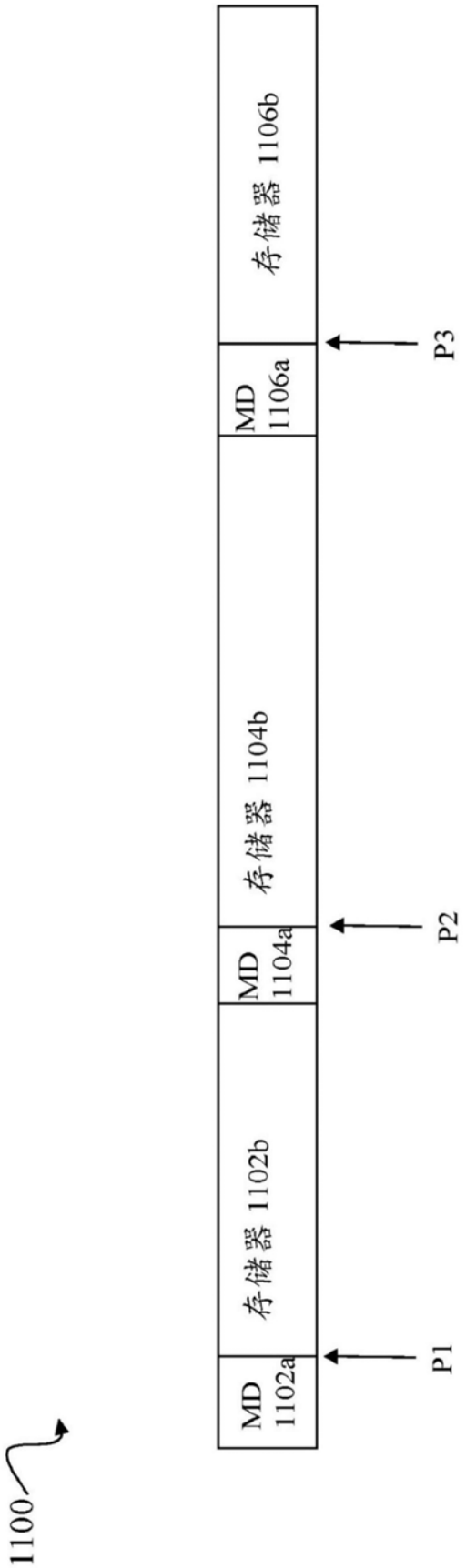

1100
MD 1102a
存储器 1102b
MD 1104a
存储器 1104b
MD 1106a
存储器 1106b
P1
P2
P3

图64

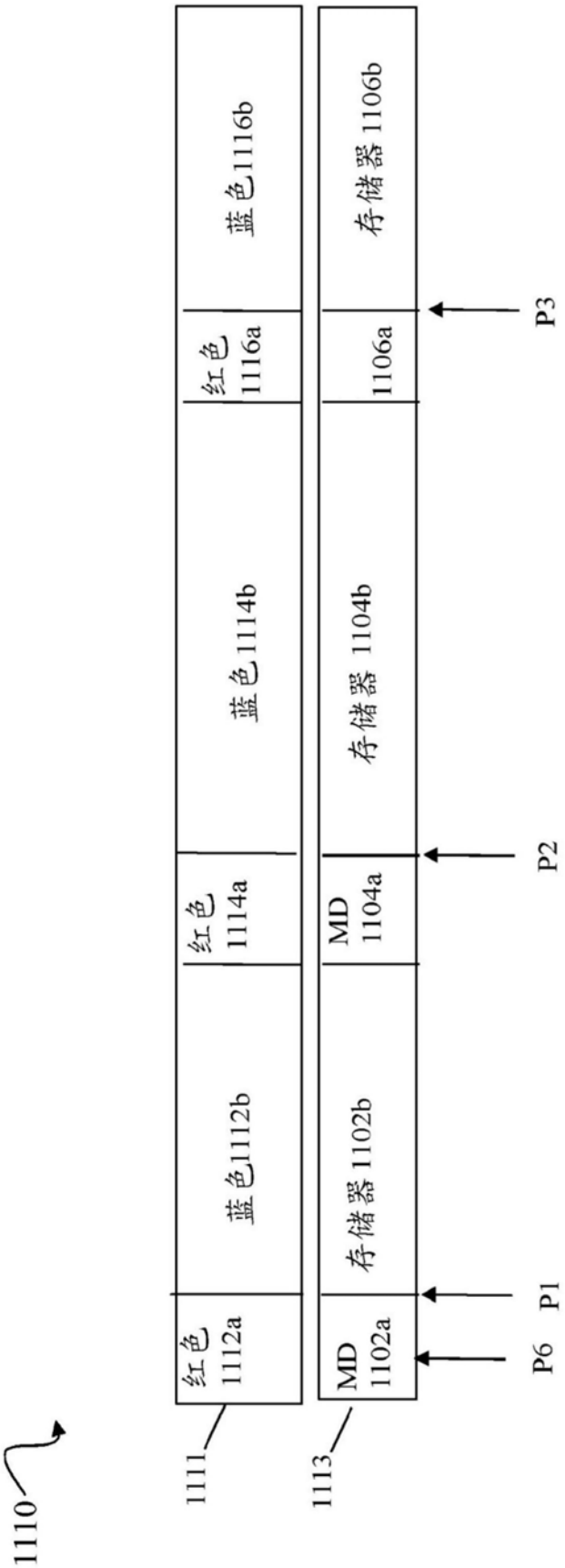

1110
1111
1113
红色 1112a
蓝色1112b
红色 1114a
蓝色1114b
红色 1116a
蓝色1116b
MD 1102a
存储器1102b
MD 1104a
存储器 1104b
1106a
存储器 1106b
P6
P1
P2
P3

图65

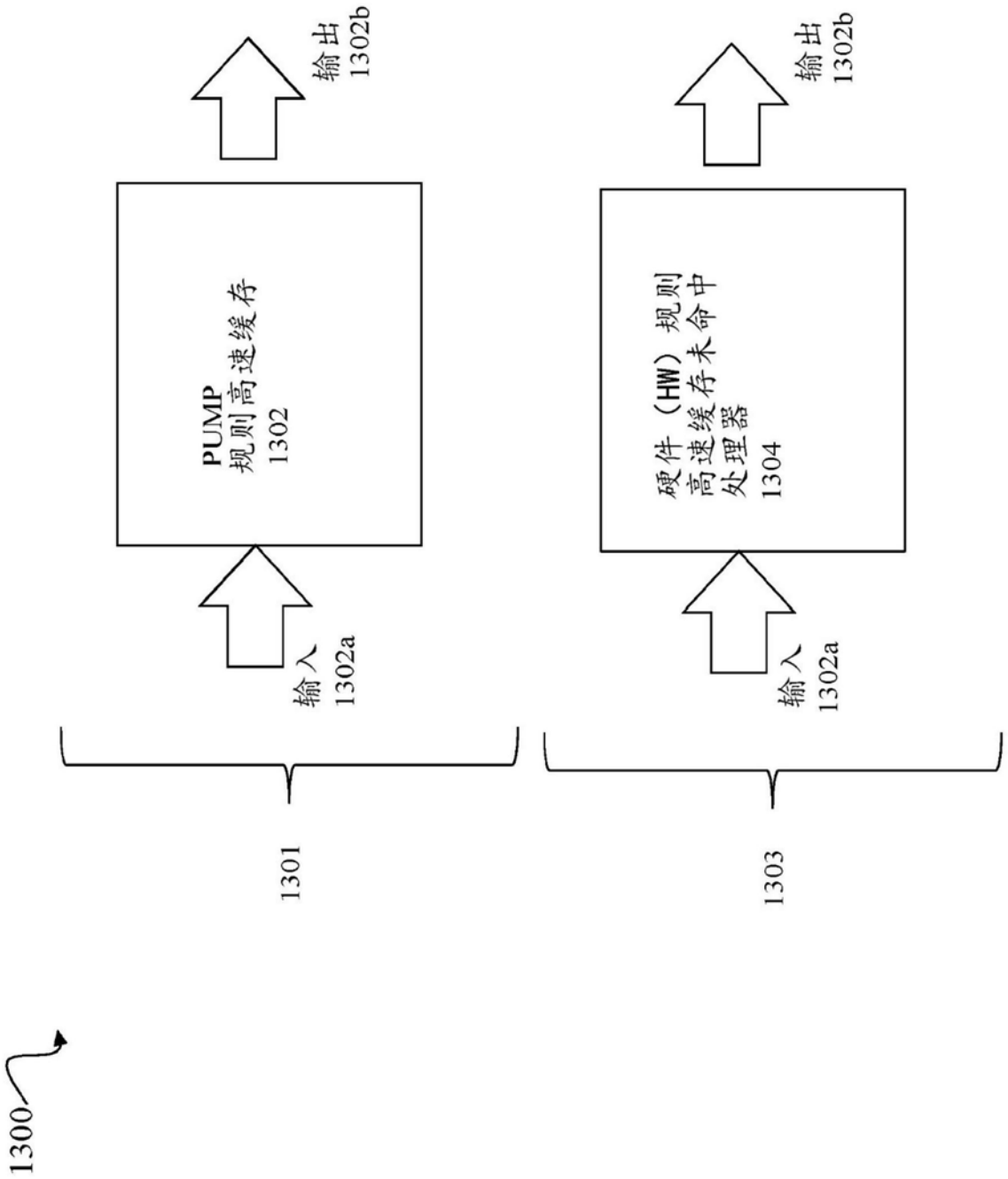

1300
1301
1303
输入
1302a
输出
1302b
PUMP
规则高速缓存
1302
硬件（HW）规则
高速缓存未命中
处理器
1304

图66

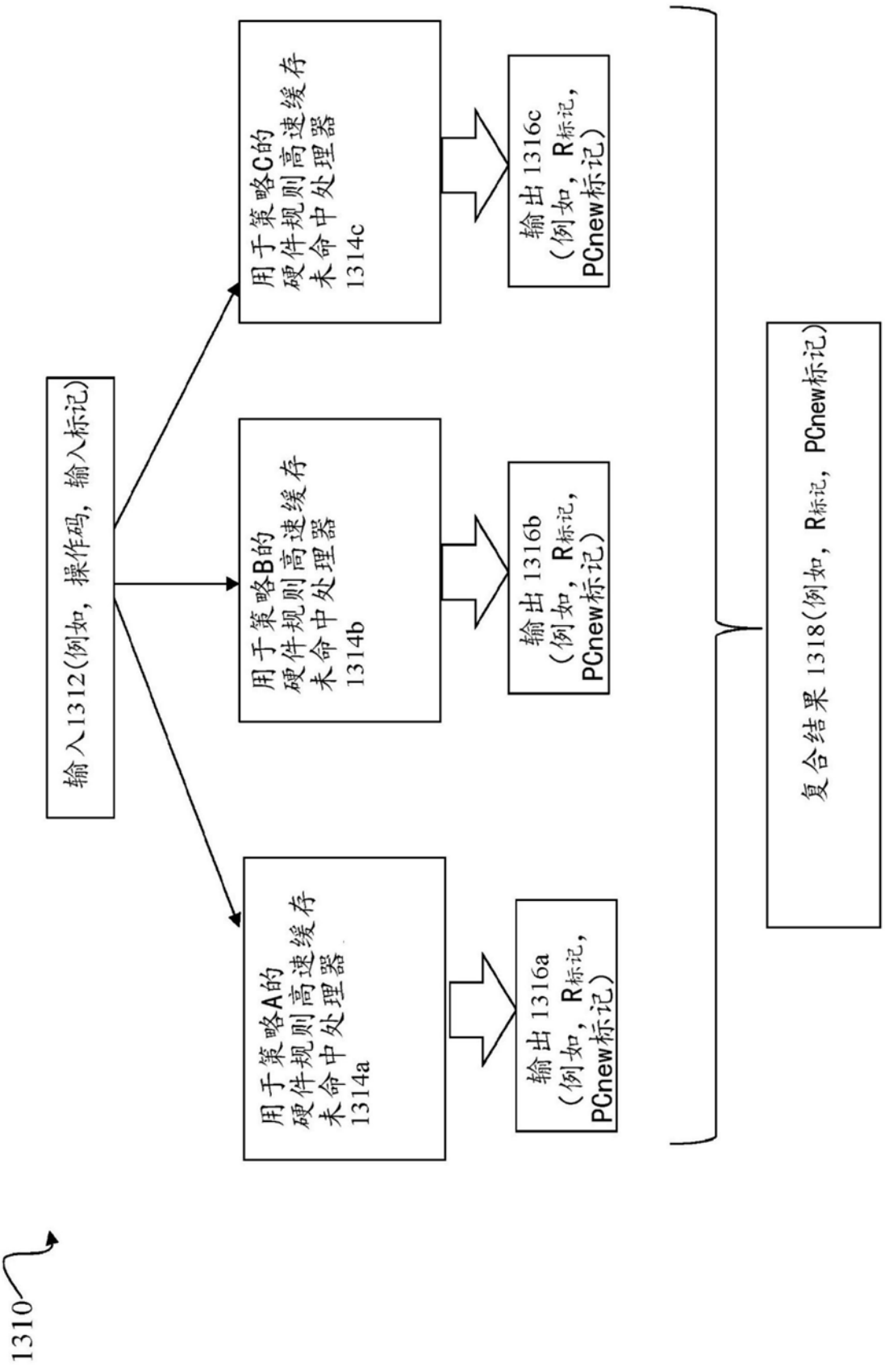

1310
输入1312(例如，操作码，输入标记)
用于策略A的硬件规则高速缓存未命中处理器1314a
用于策略B的硬件规则高速缓存未命中处理器1314b
用于策略C的硬件规则高速缓存未命中处理器1314c
输出1316a(例如，R标记，PCnew标记)
输出1316b(例如，R标记，PCnew标记)
输出1316c(例如，R标记，PCnew标记)
复合结果1318(例如，R标记，PCnew标记)

图67

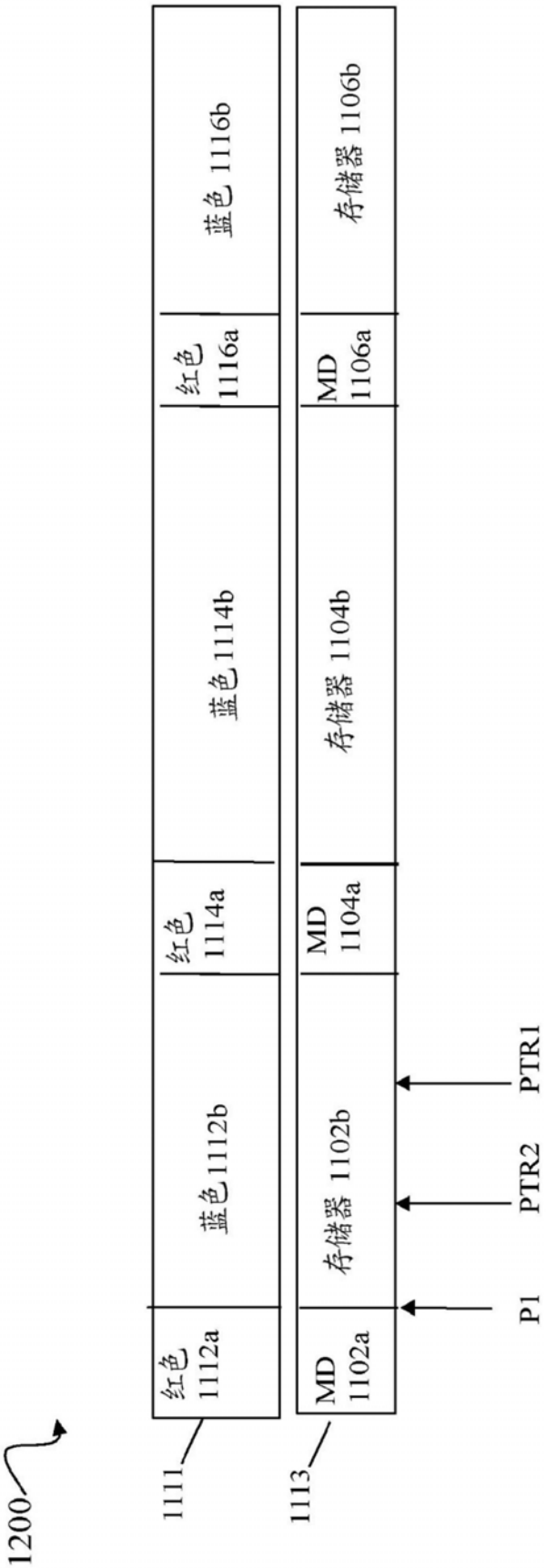

1200
1111
1113
红色 1112a
蓝色 1112b
红色 1114a
蓝色 1114b
红色 1116a
蓝色 1116b
MD 1102a
存储器 1102b
MD 1104a
存储器 1104b
MD 1106a
存储器 1106b
P1
PTR2
PTR1

图68

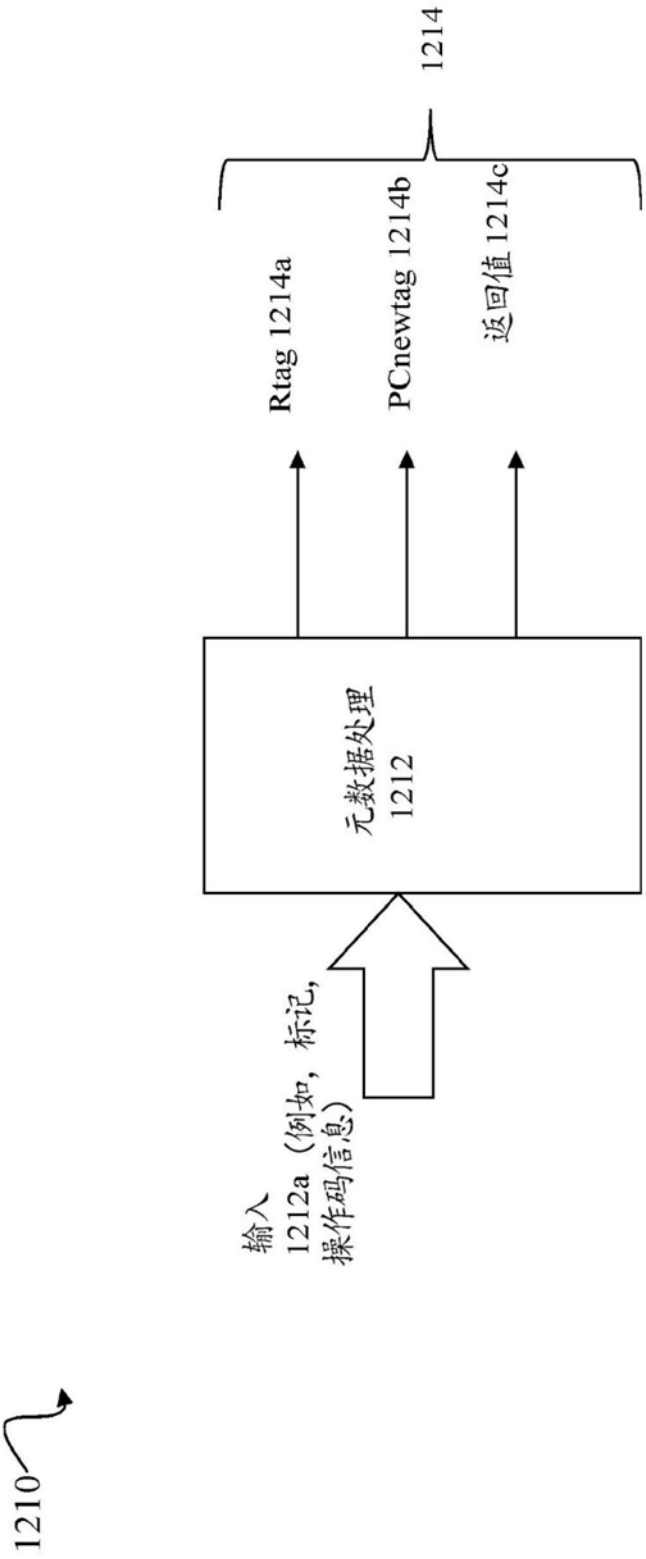

1214
Rtag 1214a
PCnewtag 1214b
返回值 1214c
元数据处理
1212
输入
1212a（例如，标记，
操作码信息）
1210

图69

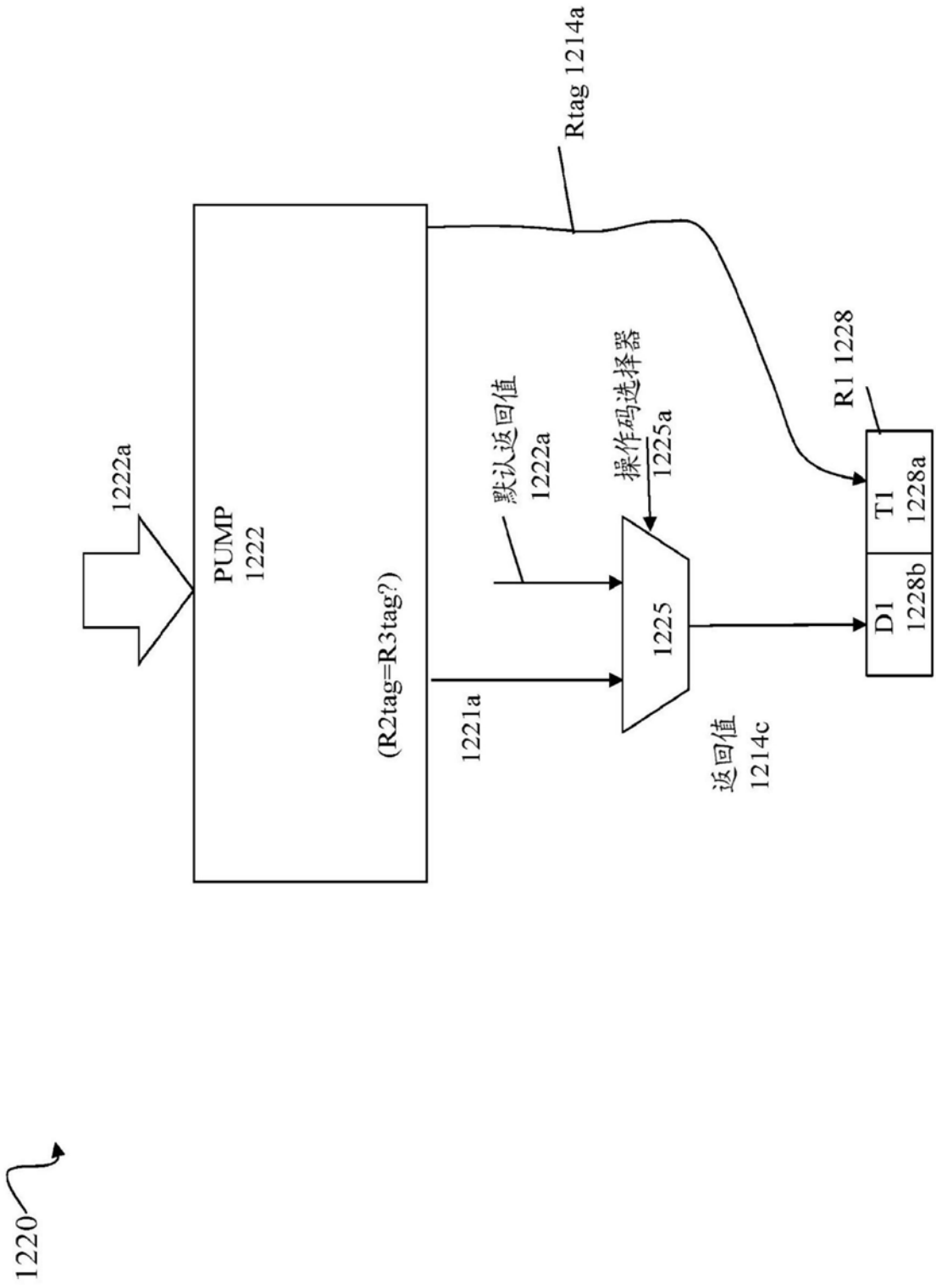

Rtag 1214a
R1 1228
T1
1228a
D1
1228b
1222a
PUMP
1222
(R2tag=R3tag?)
默认返回值
1222a
操作码选择器
1225a
1225
1221a
返回值
1214c
1220

图70

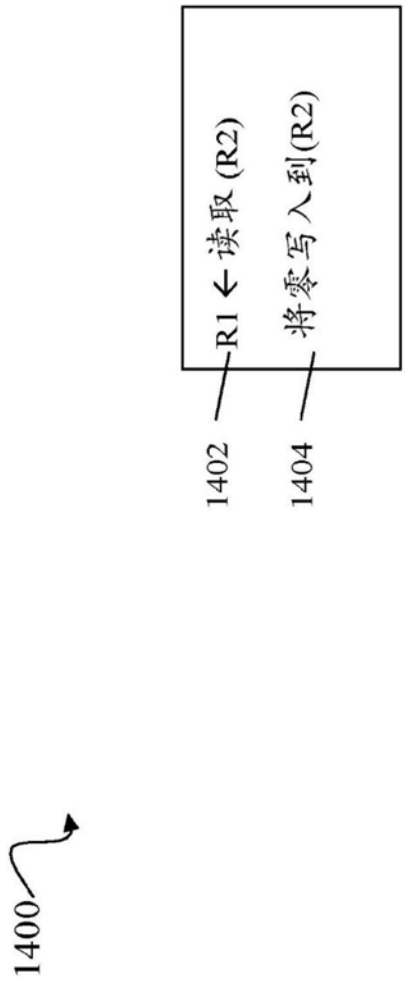


图71

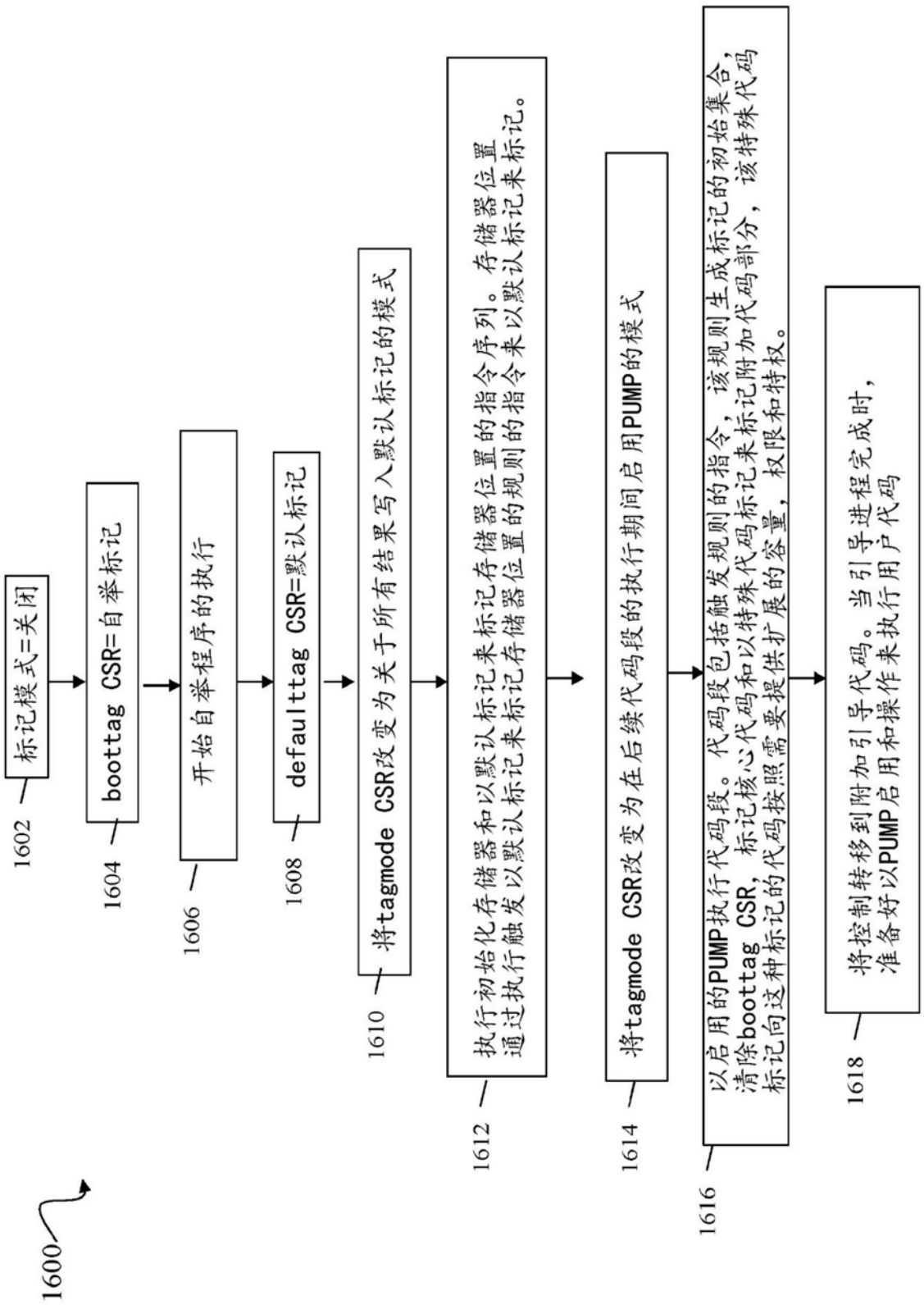

1600
1602
标记模式=关闭
1604
boottag CSR=自举标记
1606
开始自举程序的执行
1608
defaulttag CSR=默认标记
1610
将tagmode CSR改变为关于所有结果写入默认标记的模式
1612
执行初始化存储器和以默认标记来标记存储器位置的指令序列。存储器位置通过执行触发以默认标记来标记存储器位置的规则的指令来以默认标记来标记。
1614
将tagmode CSR改变为在后续代码段的执行期间启用PUMP的模式
1616
以启用的PUMP执行代码段。代码段包括触发规则的指令，该规则生成标记的初始集合，清除boottag CSR，标记核心代码和以特殊代码标记来标记附加代码部分，该特殊代码标记向这种标记的代码按照需要提供扩展的容量，权限和特权。
1618
将控制转移到附加引导代码。当引导进程完成时，准备好以PUMP启用和操作来执行用户代码

图72

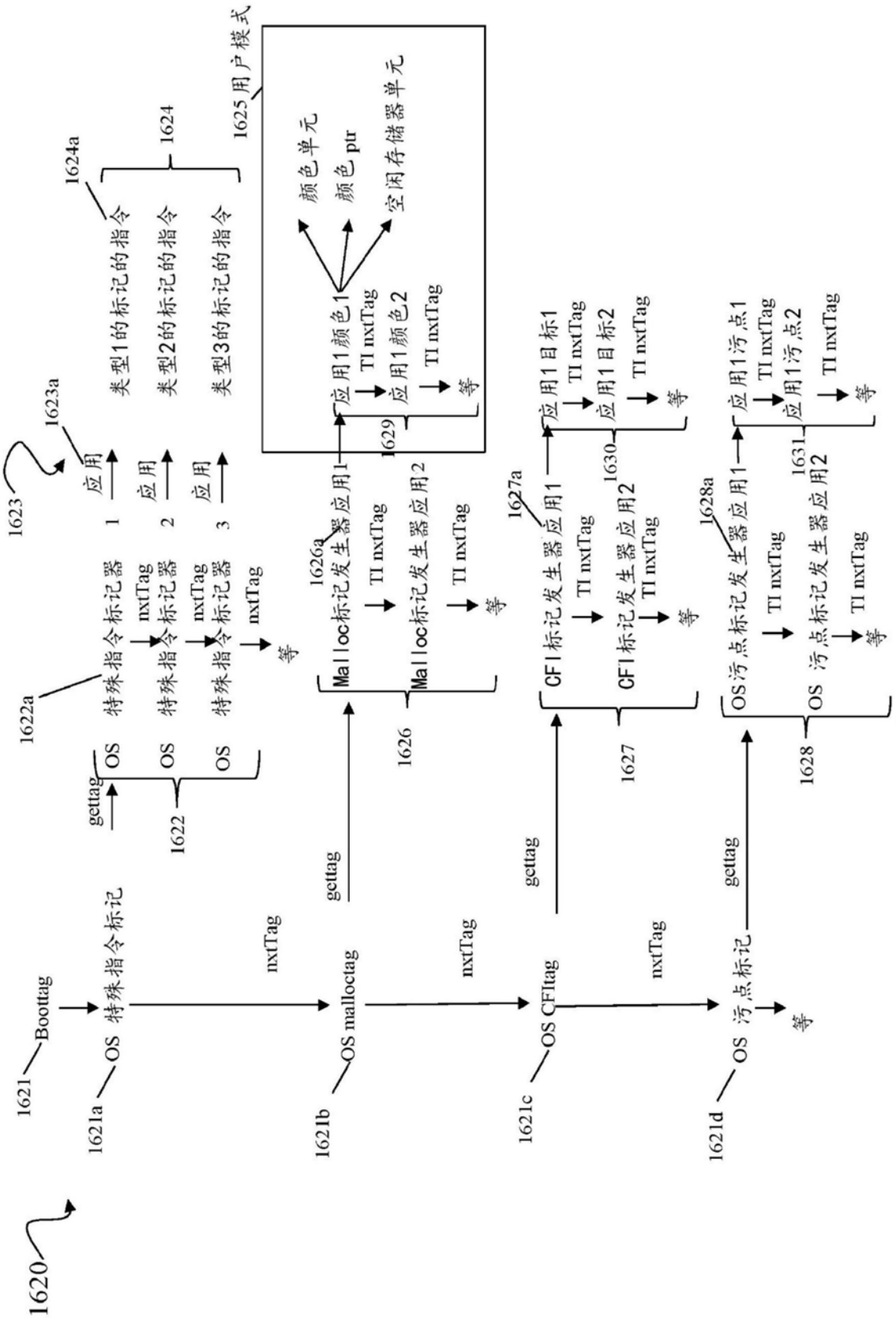

1620
1621
Boottag
1621a
OS 特殊指令标记
1621b
OS malloctag
1621c
OS CFItag
1621d
OS 污点标记
nxtTag
gettag
1622
1622a
OS 特殊指令标记器 nxtTag
OS 特殊指令标记器 nxtTag
OS 特殊指令标记器 nxtTag
等
1623
1623a
应用 1
应用 2
应用 3
1624
1624a
类型1的标记的指令
类型2的标记的指令
类型3的标记的指令
1625 用户模式
1626
1626a
Malloc标记发生器应用1
Malloc标记发生器应用2
TI nxtTag
等
1627
1627a
CFI标记发生器应用1
CFI标记发生器应用2
1628
1628a
OS污点标记发生器应用1
OS 污点标记发生器应用2
1629
应用1颜色1
应用1颜色2
颜色单元
颜色 ptr
空闲存储器单元
1630
应用1目标1
应用1目标2
1631
应用1污点1
应用1污点2

图73

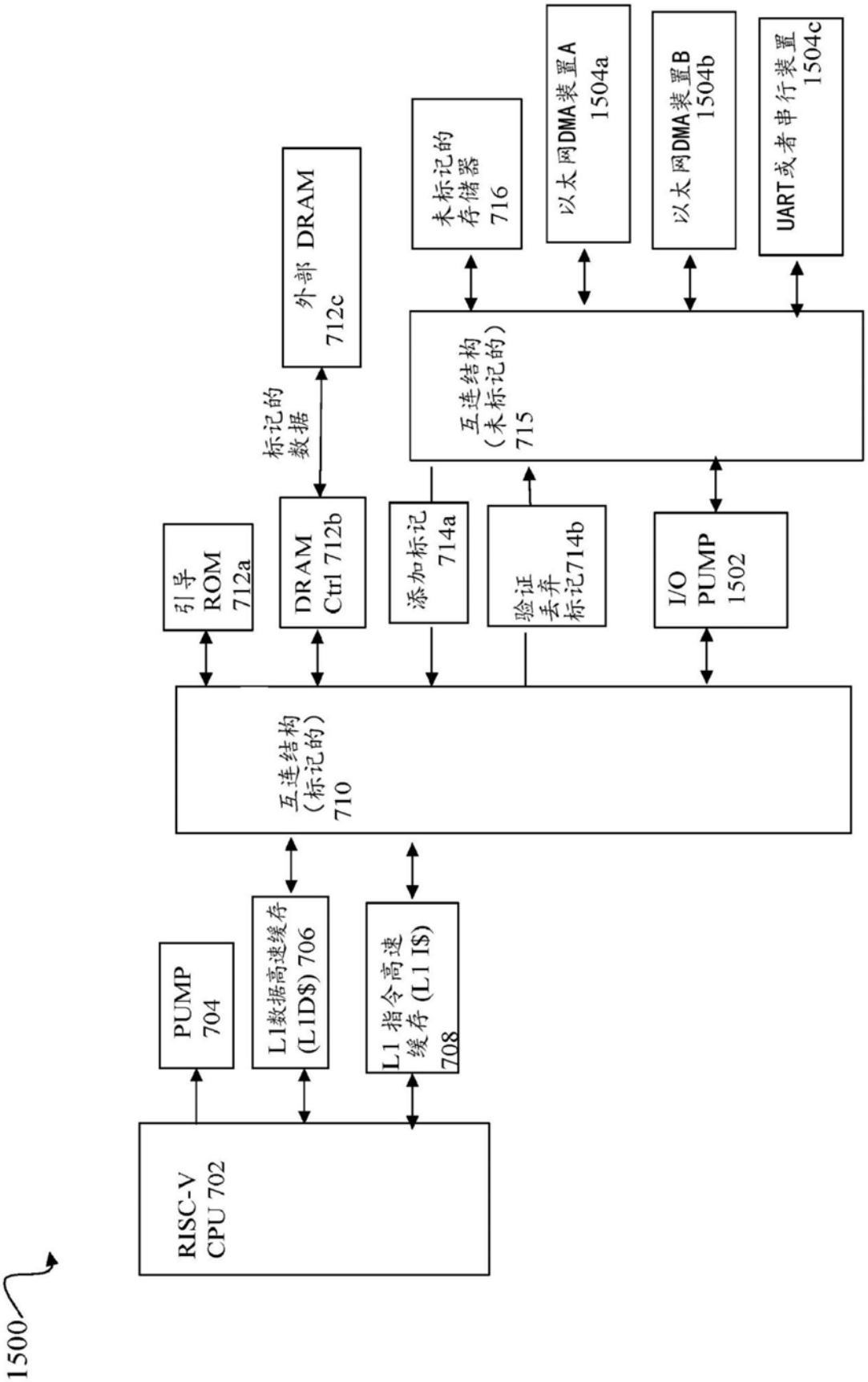

1500
RISC-V CPU 702
PUMP 704
L1数据高速缓存 (L1D$) 706
L1 指令高速缓存 (L1 I$) 708
互连结构 (标记的) 710
引导 ROM 712a
DRAM Ctrl 712b
添加标记 714a
验证丢弃标记714b
I/O PUMP 1502
互连结构 (未标记的) 715
标记的数据
外部 DRAM 712c
未标记的存储器 716
以太网DMA装置A 1504a
以太网DMA装置B 1504b
UART或者串行装置 1504c

图74

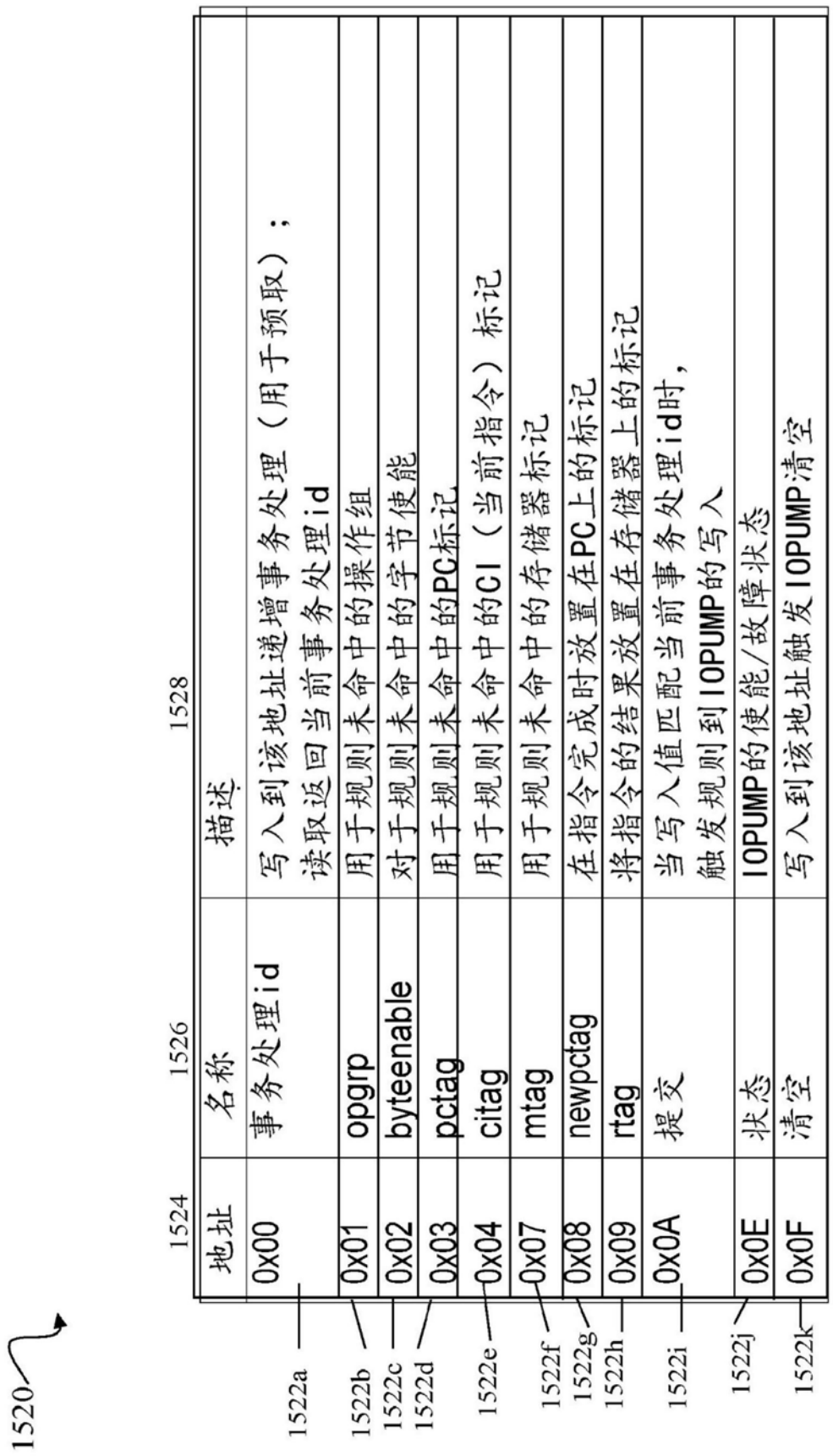

1520
1524
1526
1528
地址
名称
描述
0x00
事务处理id
写入到该地址递增事务处理（用于预取）；
读取返回当前事务处理id
0x01
opgrp
用于规则未命中的操作组
0x02
byteenable
对于规则未命中的字节使能
0x03
pctag
用于规则未命中的PC标记
0x04
citag
用于规则未命中的CI（当前指令）标记
0x07
mtag
用于规则未命中的存储器标记
0x08
newpctag
在指令完成时放置在PC上的标记
0x09
rtag
将指令的结果放置在存储器上的标记
0x0A
提交
当写入值匹配当前事务处理id时，
触发规则到IOPUMP的写入
0x0E
状态
IOPUMP的使能/故障状态
0x0F
清空
写入到该地址触发IOPUMP清空
1522a
1522b
1522c
1522d
1522e
1522f
1522g
1522h
1522i
1522j
1522k

图75

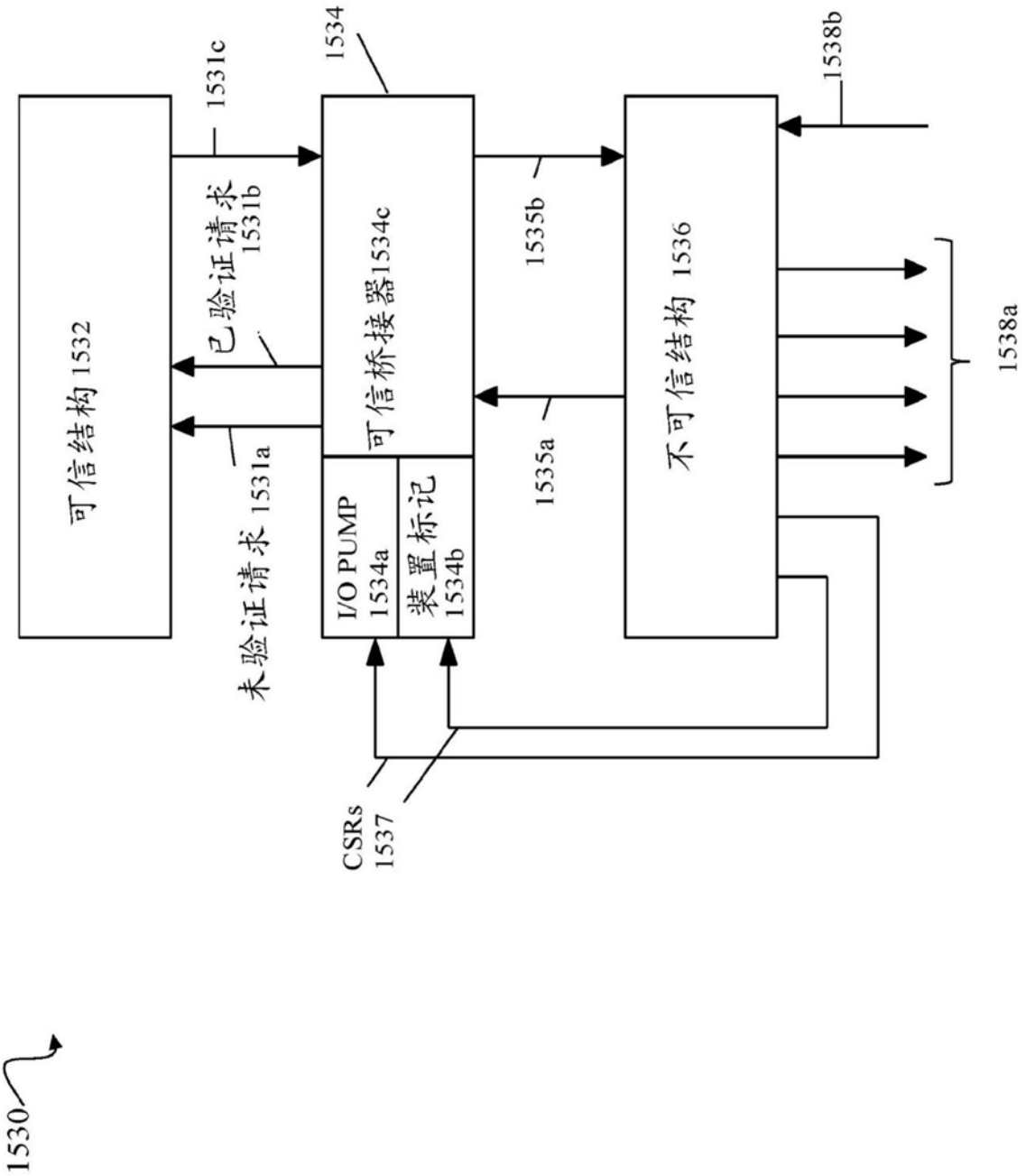

1530
可信结构 1532
1531c
已验证请求 1531b
未验证请求 1531a
1534
可信桥接器1534c
I/O PUMP 1534a
装置标记 1534b
CSRs 1537
1535b
1535a
不可信结构 1536
1538b
1538a

图76

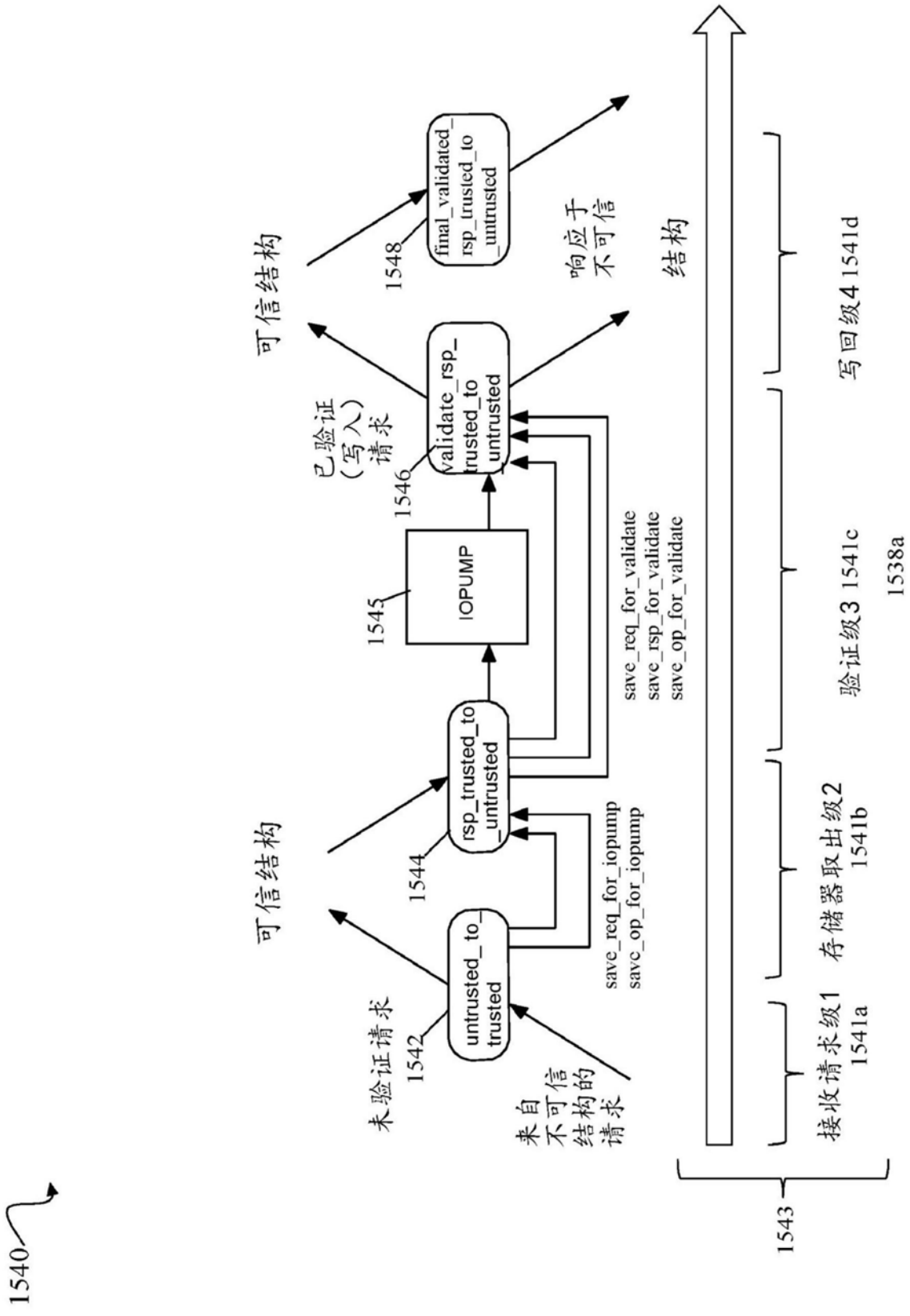

1540
1543
接收请求级1
1541a
存储器取出级2
1541b
验证级3 1541c
写回级4 1541d
1538a
结构
未验证请求
1542
untrusted_to_trusted
来自不可信结构的请求
可信结构
save_req_for_iopump
save_op_for_iopump
1544
rsp_trusted_to_untrusted
1545
IOPUMP
save_req_for_validate
save_rsp_for_validate
save_op_for_validate
已验证（写入）请求
1546
validate_rsp_trusted_to_untrusted
响应于不可信
可信结构
1548
final_validated_rsp_trusted_to_untrusted

图77

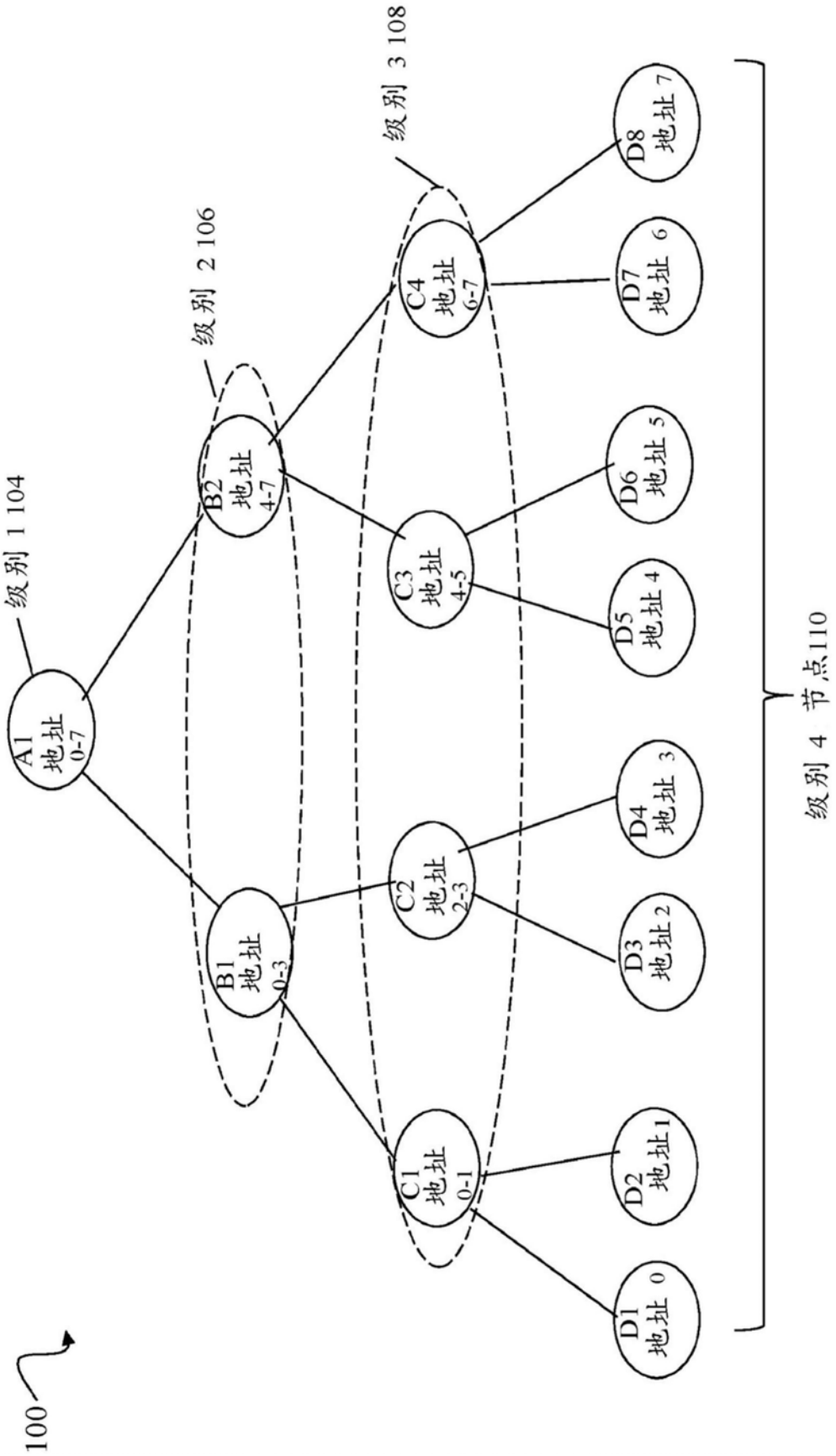

100
级别 1 104
级别 2 106
级别 3 108
级别 4 节点110
A1 地址 0-7
B1 地址 0-3
B2 地址 4-7
C1 地址 0-1
C2 地址 2-3
C3 地址 4-5
C4 地址 6-7
D1 地址 0
D2 地址1
D3 地址 2
D4 地址 3
D5 地址 4
D6 地址 5
D7 地址 6
D8 地址 7

图78

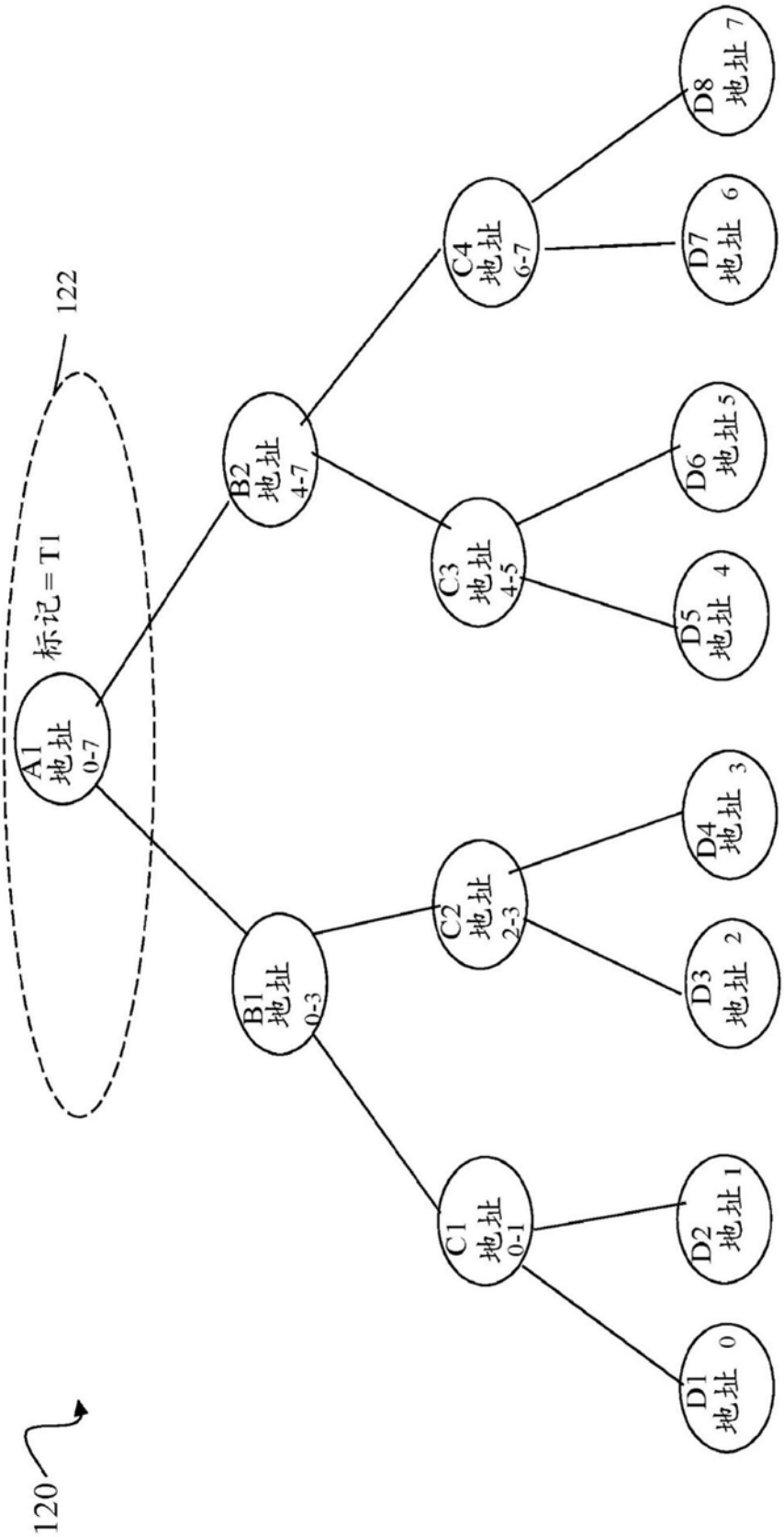

120
122
标记=T1
A1 地址 0-7
B1 地址 0-3
B2 地址 4-7
C1 地址 0-1
C2 地址 2-3
C3 地址 4-5
C4 地址 6-7
D1 地址 0
D2 地址 1
D3 地址 2
D4 地址 3
D5 地址 4
D6 地址 5
D7 地址 6
D8 地址 7

图79

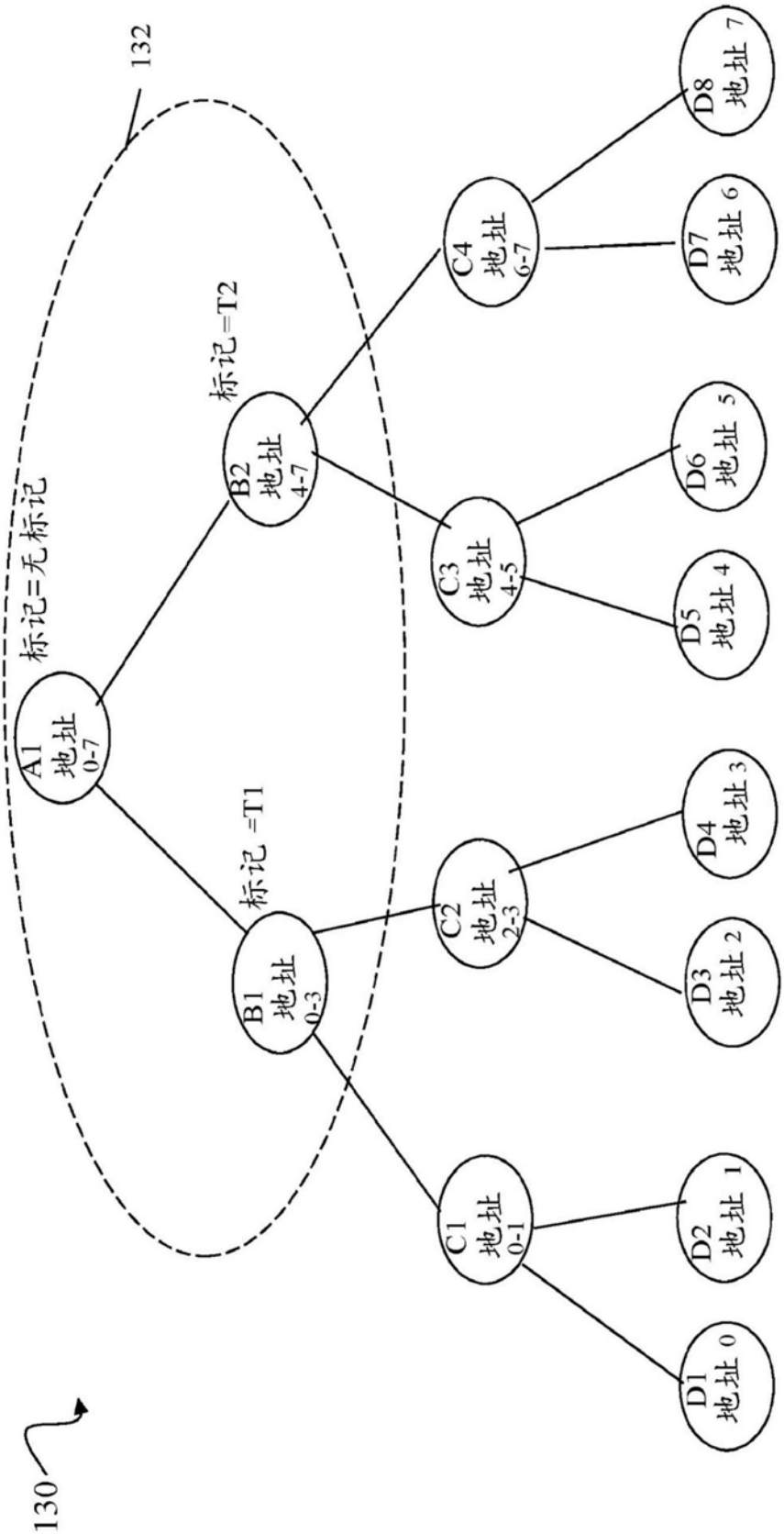

130
132
标记=无标记
A1
地址
0-7
标记 =T1
B1
地址
0-3
标记=T2
B2
地址
4-7
C1
地址
0-1
C2
地址
2-3
C3
地址
4-5
C4
地址
6-7
D1
地址 0
D2
地址 1
D3
地址 2
D4
地址 3
D5
地址 4
D6
地址 5
D7
地址 6
D8
地址 7

图80

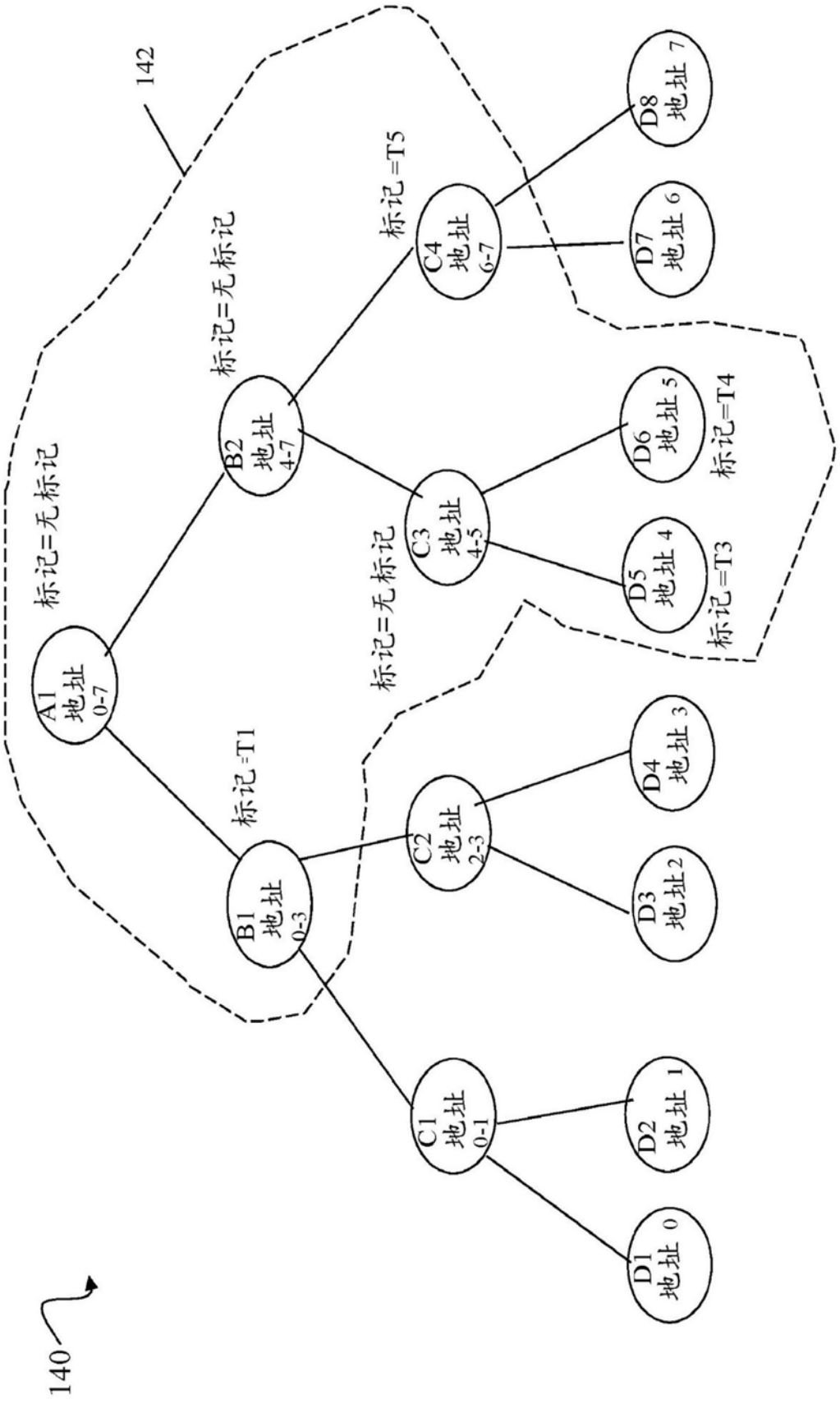

140
142
A1
地址
0-7
标记=无标记
B1
地址
0-3
标记=T1
B2
地址
4-7
标记=无标记
C1
地址
0-1
C2
地址
2-3
C3
地址
4-5
标记=无标记
C4
地址
6-7
标记=T5
D1
地址 0
D2
地址 1
D3
地址2
D4
地址 3
D5
地址 4
标记=T3
D6
地址 5
标记=T4
D7
地址 6
D8
地址 7

图81

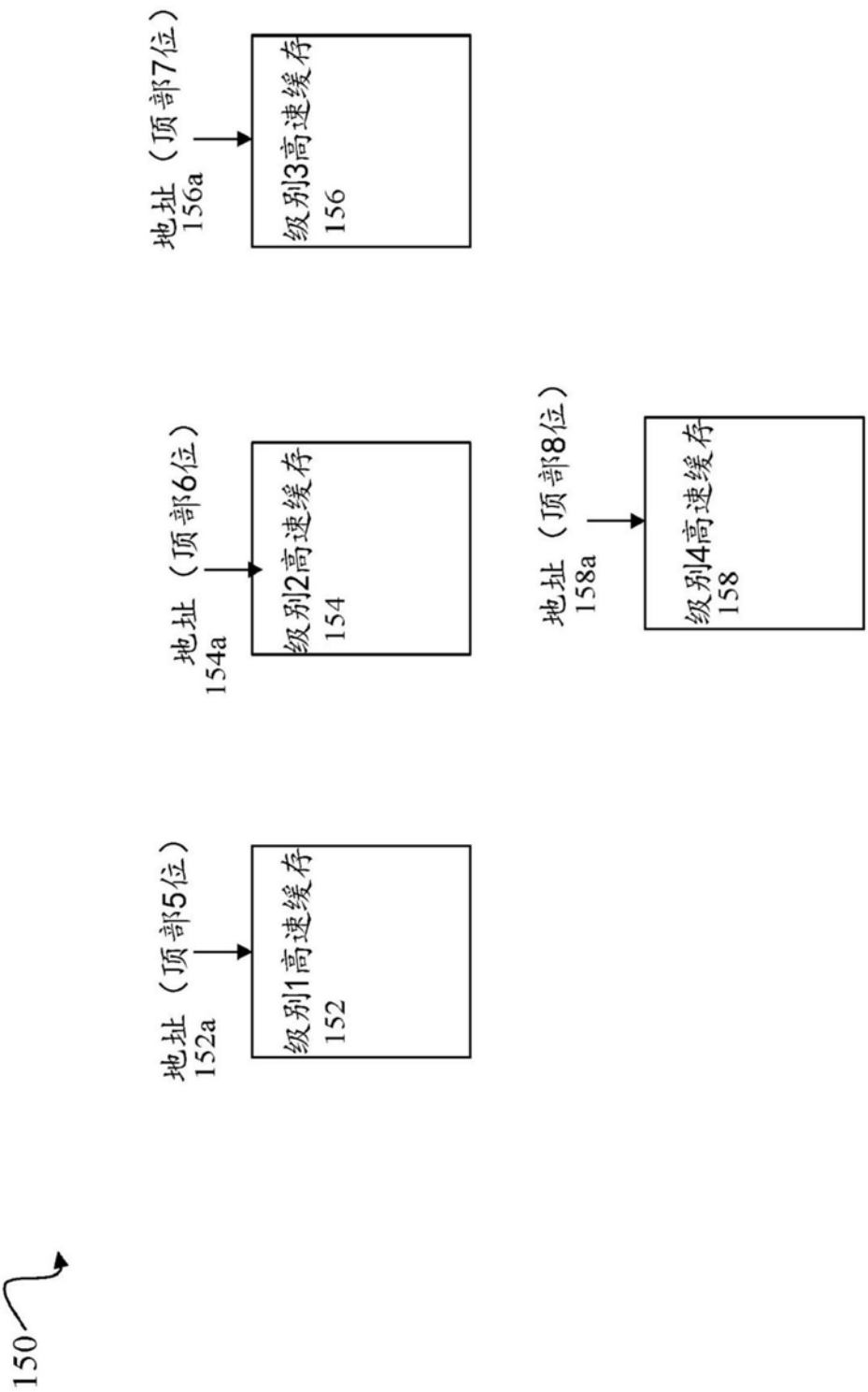

150
地址（顶部5位）
152a
级别1高速缓存
152
地址（顶部6位）
154a
级别2高速缓存
154
地址（顶部7位）
156a
级别3高速缓存
156
地址（顶部8位）
158a
级别4高速缓存
158

图82

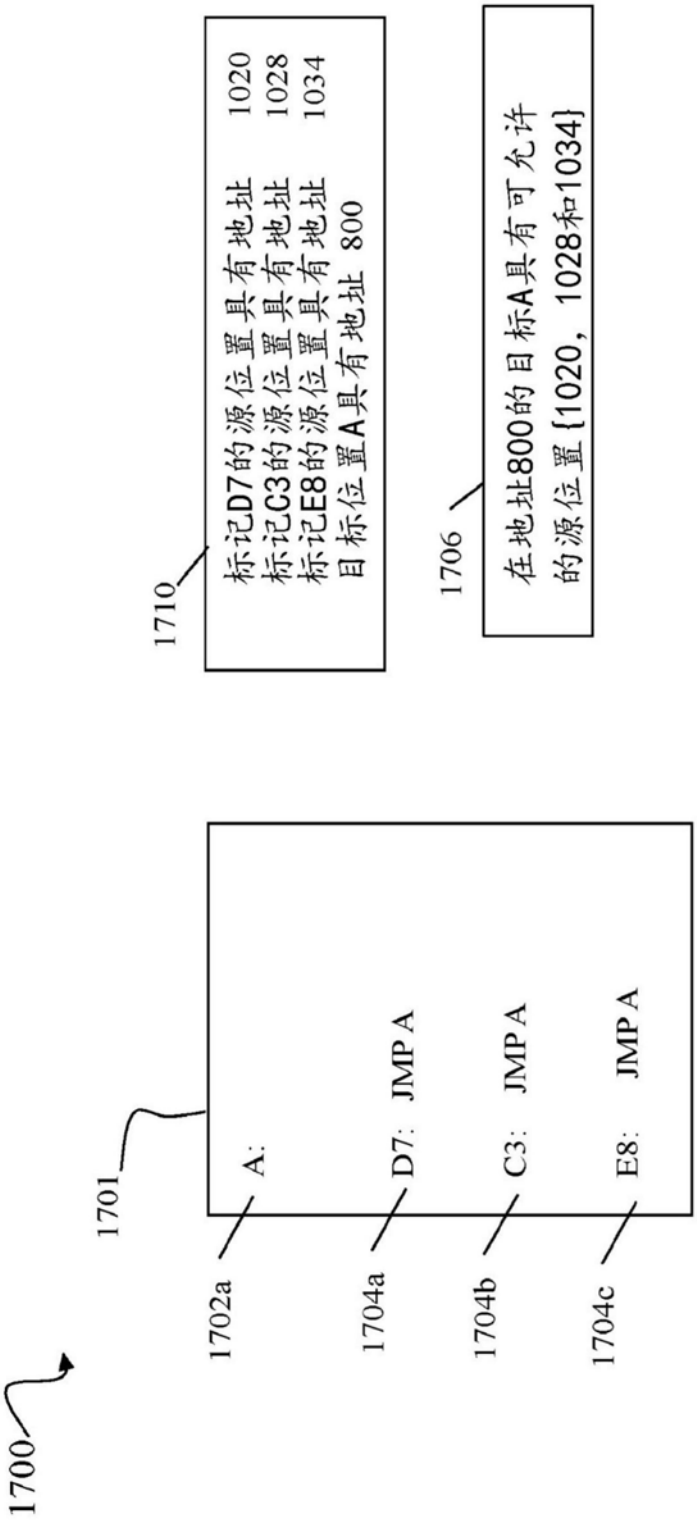

1700
1701
A:
D7: JMP A
C3: JMP A
E8: JMP A
1702a
1704a
1704b
1704c
1710
标记D7的源位置具有地址 1020
标记C3的源位置具有地址 1028
标记E8的源位置具有地址 1034
目标位置A具有地址 800
1706
在地址800的目标A具有可允许
的源位置{1020，1028和1034}

图83

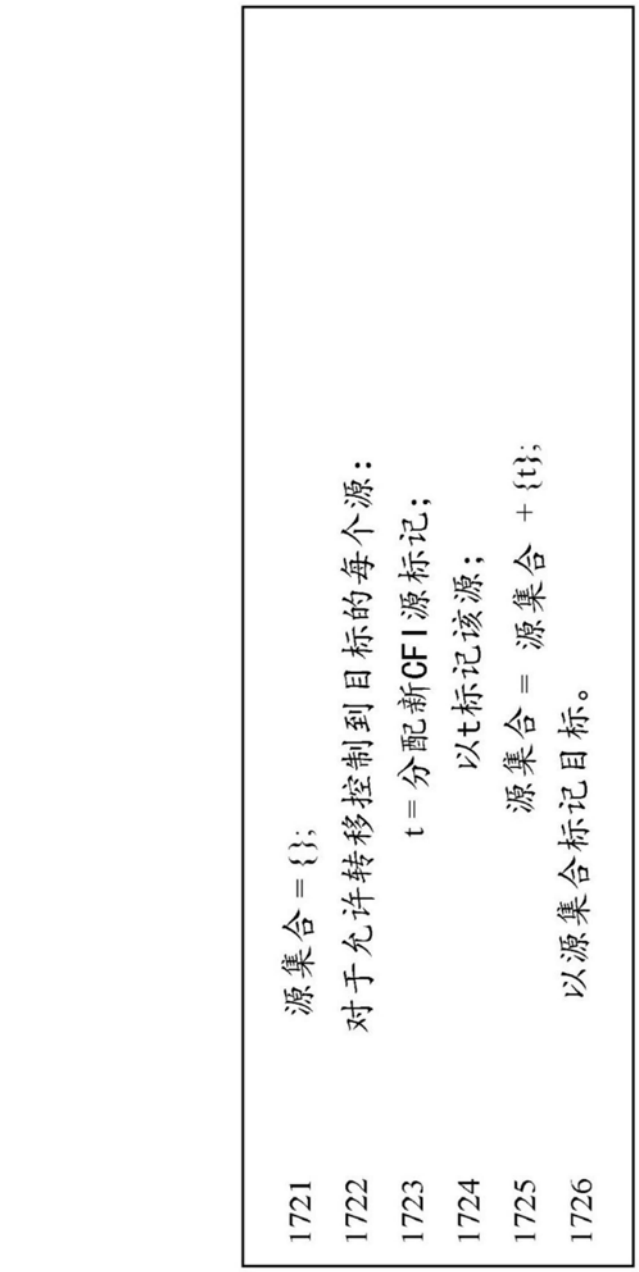

1721 源集合 = {};
1722 对于允许转移控制到目标的每个源：
1723 t = 分配新CFI源标记；
1724 以t标记该源；
1725 源集合 = 源集合 + {t};
1726 以源集合标记目标。
1720

图84