(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 104704503 A
(43)申请公布日 2015.06.10

(21)申请号 201380050004.4
(22)申请日 2013.08.07
(30)优先权数据
1217531.1 2012.10.01 GB
13/680,352 2012.11.19 US
(85)PCT国际申请进入国家阶段日
2015.03.25
(86)PCT国际申请的申请数据
PCT/GB2013/052107 2013.08.07
(87)PCT国际申请的公布数据
WO2014/053803 EN 2014.04.10

(71)申请人 ARM 有限公司
地址 英国剑桥
(72)发明人 托马斯·克里斯托弗·乔洛卡特
理查德·罗伊·格里森思怀特
西蒙·约翰·克拉斯克
(74)专利代理机构 北京东方亿思知识产权代理有限责任公司 11258
代理人 李晓冬

(51)Int.Cl.
G06F 21/52(2006.01)

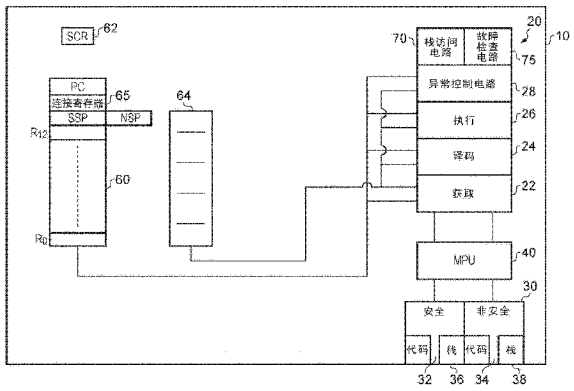
权利要求书3页 说明书17页 附图17页

(54)发明名称

在安全域与次安全域之间进行转换时保护安全数据和程序代码免受非安全访问的数据处理装置和方法

(57)摘要

数据处理装置和方法被提供用于处理数据。数据处理装置包括：处理电路，用于响应于程序代码而执行数据处理操作；以及数据存储设备，用于存储数据，数据存储设备包括多个区域，其包括安全区域和次安全区域。安全区域被配置为存储当在安全域中操作时可被处理电路访问而当在次安全域中操作时不可被处理电路访问的敏感数据。数据存储设备包括多个栈，其包括在安全区域中的安全栈。处理电路包括栈访问电路，其被配置为响应于需要从安全域转换到次安全域的事件而将预定处理状态存储到安全栈。特别地，如果事件是第一事件类型，则由栈访问电路存储的预定处理状态至少包括被存储在安全栈上的预定相对位置处的返回地址。相反，如果事件是第二事件类型，则由栈访问电路存储的预定处理状态至少包括被存储在预定相对位置处的第一值，其中该第一值不是程序代码的有效地址。处理电路还包括故障检查电路，其被配置为在接收到从次安全域到安全域的第一事件类型返回时，如果存储在预定相对位置中的数据是第一值，则标识第一故障情况。该方法提供了对抗试图使用错误返回方法（例如，从异常执行函数调用返回或从函数调用执行异常返回）的来自次安全域的攻击的保护。



CN 104704503 A

1. 一种数据处理装置，所述数据处理装置包括：

处理电路，所述处理电路被配置为响应于程序代码而执行数据处理操作；

数据存储设备，所述数据存储设备被配置为存储数据，所述数据存储设备包括多个区域，所述区域包括安全区域和次安全区域，其中所述安全区域被配置为存储当在安全域中操作时可被所述处理电路访问而当在次安全域中操作时不可被所述处理电路访问的敏感数据；

所述数据存储设备包括多个栈，所述栈包括在所述安全区域中的安全栈；

所述处理电路包括栈访问电路，所述栈访问电路被配置为响应于需要从所述安全域转换到所述次安全域的事件而将预定处理状态存储到所述安全栈；

如果所述事件是第一事件类型，则由所述栈访问电路存储的所述预定处理状态至少包括被存储在所述安全栈上的预定相对位置处的返回地址；

如果所述事件是第二事件类型，则由所述栈访问电路存储的所述预定处理状态至少包括被存储在所述预定相对位置处的第一值，其中所述第一值不是程序代码的有效地址；并且

所述处理电路还包括故障检查电路，所述故障检查电路被配置为在接收到从所述次安全域到所述安全域的第一事件类型返回时，如果存储在所述预定相对位置的数据是所述第一值，则标识第一故障情况。

2. 如权利要求 1 所述的数据处理装置，其中所述第一事件类型是函数调用，并且所述第二事件类型是异常。

3. 如权利要求 1 和 2 中的任一项所述的数据处理装置，其中：

如果所述事件是所述第一事件类型，则所述处理电路被配置为向所述次安全域中被用于处理所述第一事件类型的程序代码提供虚拟返回地址，所述虚拟返回地址不是程序代码的有效地址；并且

当完成所述次安全域中被用于处理所述第一事件类型的程序代码时，所述虚拟返回地址导致所述返回地址从所述安全栈中被获取。

4. 如权利要求 2 所述的数据处理装置，其中：

如果所述事件是异常，则由所述栈访问电路存储的所述预定处理状态包括预定签名；并且

所述故障检查电路被配置为当接收到从所述次安全域到所述安全域的异常返回时，如果所述预定签名不在由所述栈访问电路存储的所述预定处理状态中，则标识第二故障情况。

5. 如权利要求 4 所述的数据处理装置，其中所述预定签名形成被存储在所述预定相对位置处的所述第一值，使得所述第一和第二故障情况能够使用所述预定签名来检测。

6. 如权利要求 4 和 5 中的任一项所述的数据处理装置，其中：

如果所述事件是函数调用，则所述处理电路被配置为向所述次安全域中被所述函数调用标识的程序代码提供虚拟函数调用返回地址，所述虚拟函数调用返回地址是从不是程序代码的有效地址的地址范围中选择的；并且

所述预定签名具有与所述虚拟函数调用返回地址不同的值。

7. 如权利要求 4、5 和 6 中的任一项所述的数据处理装置，其中：

如果所述事件是异常，则所述处理电路被配置为向所述次安全域中被用于处理所述异常的程序代码提供虚拟异常返回地址，所述虚拟异常返回地址是从不是程序代码的有效地址的地址范围中选择的；并且

所述预定签名具有与所述虚拟异常返回地址不同的值。

8. 如权利要求 4 到 8 中的任一项所述的数据处理装置，其中所述处理电路响应于所述第二故障情况而在所述安全域内执行故障处理程序代码。

9. 如权利要求 4 到 8 中的任一项所述的数据处理装置，其中如果所述第二故障情况被标识，则栈指针值不被调整，使得所述异常返回的重放也将导致所述第二故障情况被标识。

10. 如上述权利要求中的任一项所述的数据处理装置，其中所述预定相对位置是所述栈访问电路使用的栈帧内的预定位置。

11. 如权利要求 10 所述的数据处理装置，其中所述预定位置是所述栈帧的底部位置。

12. 如上述权利要求中的任一项所述的数据处理装置，其中所述处理电路响应于所述第一故障情况而在所述安全域内执行故障处理程序代码。

13. 如上述权利要求中的任一项所述的数据处理装置，其中如果所述第一故障情况被标识，则栈指针值不被调整，使得所述第一事件类型返回的重放也将地址所述第一故障情况被标识。

14. 如上述权利要求中的任一项所述的数据处理装置，其中：

在所述安全栈的初始化时，第二值被存储在所述安全栈上的所述预定相对位置处，所述第二值与所述第一值不同，并且也不是程序代码的有效地址；并且

所述故障检查电路被配置为当接收到从所述次安全域到所述安全域的返回时，如果存储在所述预定相对位置中的数据是所述第二值，则标识第三故障情况。

15. 如上述权利要求中的任一项所述的数据处理装置，还包括：

多个寄存器；并且

如果所述事件是异常，则由所述栈访问电路存储的所述预定处理状态包括至少所述寄存器的子集的内容。

16. 如上述权利要求中的任一项所述的数据处理装置，其中：

所述多个栈还包括在所述次安全区域中的次安全栈；并且

所述处理电路被配置为基于后台处理正在其中被执行的域来确定将数据存储到哪个栈或从哪个栈加载数据。

17. 一种在数据处理装置上处理数据的方法，所述数据处理装置包括：处理电路，所述处理电路用于响应于程序代码而执行数据处理操作；以及数据存储设备，所述数据存储设备用于存储数据，所述数据存储设备包括多个区域，所述区域包括安全区域和次安全区域，所述安全区域被配置为存储当在安全域中操作时可被所述处理电路访问而当在次安全域中操作时不可被所述处理电路访问的敏感数据；所述数据存储设备包括多个栈，所述栈包括在所述安全区域中的安全栈；并且所述方法包括：

响应于需要从所述安全域转换到所述次安全域的事件而将预定处理状态存储到所述安全栈；

如果所述事件是第一事件类型，则至少将被存储在所述安全栈上的预定相对位置处的返回地址存储为所述预定处理状态；

如果所述事件是第二事件类型，则至少将被存储在所述预定相对位置处的第一值存储为所述预定处理状态，其中所述第一值不是程序代码的有效地址；并且

在接收到从所述次安全域到所述安全域的第一事件类型返回时，如果存储在所述预定相对位置的数据是所述第一值，则标识第一故障情况。

18. 一种数据处理装置，所述数据处理装置包括：

数据处理装置，所述数据处理装置用于响应于程序代码而执行数据处理操作；

数据存储装置，所述数据存储装置用于存储数据，所述数据存储装置包括多个区域，所述区域包括安全区域和次安全区域，所述安全区域被配置为存储当在安全域中操作时可被所述处理电路访问而当在次安全域中操作时不可被所述处理电路访问的敏感数据；

所述数据存储装置包括多个栈，所述栈包括在所述安全区域中的安全栈；

所述处理装置包括栈访问装置，所述栈访问装置用于响应于需要从所述安全域转换到所述次安全域的事件而将预定处理状态存储到所述安全栈；

如果所述事件是第一事件类型，则由所述栈访问装置存储的所述预定处理状态至少包括被存储在所述安全栈上的预定相对位置处的返回地址；

如果所述事件是第二事件类型，则由所述栈访问装置存储的所述预定处理状态至少包括被存储在所述预定相对位置处的第一值，其中所述第一值不是程序代码的有效地址；并且

所述处理装置还包括故障检查装置，所述故障检查装置用于响应于接收到从所述次安全域到所述安全域的第一事件类型返回，如果存储在所述预定相对位置的数据是所述第一值，则标识第一故障情况。

在安全域与次安全域之间进行转换时保护安全数据和程序代码免受非安全访问的数据处理装置和方法

技术领域

[0001] 本发明涉及数据处理领域，并且特别涉及敏感数据和代码的处理。

背景技术

[0002] 很多数据处理系统及架构提供了隔离并保护敏感数据和代码部分不被无权的人或进程访问的方法。虽然能够提供安全是重要的，但是在与该保护相关联的性能和电路面积中存在开销。

[0003] 在诸如微控制器之类的小型系统中，将这些开销保持较低是非常重要的，从而可能需要在安全等级与性能之间做出折衷。

[0004] 英国剑桥的ARM®用其Trustzone结构提供了保持数据和代码安全的一个方法，其中存在安全状态和非安全状态（也被称为安全和非安全域），并且异常指令被用于在状态之间进行转换，该异常处理器保护安全侧的安全。虽然该方法提供了很高的安全度，但是需要以软件异常处理器形式的相当大的软件干预来改变安全状态，这既降低了系统的性能，又增加了为安全软件开发外部应用程序接口API所需的工作量，因为所有通话必须通过异常处理器来代理。类似地，虽然在安全域中发生但需要在非安全域中处理的异常也需要通过安全异常处理器来代理，这允许安全状态在控制转到非安全异常处理器之前得到保护。

[0005] US7966466和US2008/0250216公开了一种供选择的安全系统，其中数据存储设备具有安全侧和非安全侧，并且在该数据存储设备中代码当前正被执行的位置确定了处理器正在其中操作的域，从而确定了允许访问的数据。

[0006] 包含安全状态的许多系统架构只允许用于在状态之间进行转换（通常通过使用异常）的一种机制，因为这简化了保护安全状态中的数据和程序代码安全的任务。但是，为了提高速度和效率，也可以提供用于在安全和次安全状态之间进行转换的多种机制。例如，由Microchip科技公司开发的CodeGuard系统允许异常和直接函数调用作为用于在安全性状态之间进行转换的机制。但是，使用用于在状态之间进行转换的多种机制增加了对恶意攻击的脆弱性，例如，允许次安全状态中的软件试图通过具有与用于从安全状态转换到次安全状态中的原机制不同的类型的返回机制返回安全状态（例如，通过使用函数调用返回来从异常返回，或使用异常返回来从函数调用返回）。

发明内容

[0007] 从第一方面来看，本发明提供了一种数据处理装置，所述数据处理装置包括：处理电路，所述处理电路被配置为响应于程序代码而执行数据处理操作；数据存储设备，所述数据存储设备被配置为存储数据，所述数据存储设备包括多个区域，所述区域包括安全区域和次安全区域，其中所述安全区域被配置为存储当在安全域中操作时可被所述处理电路访问而当在次安全域中操作时不可被所述处理电路访问的敏感数据；所述数据存储设备包括

多个栈，所述栈包括在所述安全区域中的安全栈；所述处理电路包括栈访问电路，所述栈访问电路被配置为响应于需要从所述安全域转换到所述次安全域的事件而将预定处理状态存储到所述安全栈；如果所述事件是第一事件类型，则由所述栈访问电路存储的所述预定处理状态至少包括被存储在所述安全栈上的预定相对位置处的返回地址；如果所述事件是第二事件类型，则由所述栈访问电路存储的所述预定处理状态至少包括被存储在所述预定相对位置处的第一值，其中所述第一值不是程序代码的有效地址；并且所述处理电路还包括故障检查电路，所述故障检查电路被配置为在接收到从所述次安全域到所述安全域的第一事件类型返回时，如果存储在所述预定相对位置的数据是所述第一值，则标识第一故障情况。

[0008] 根据本发明，允许多个事件类型导致在安全域和次安全域之间的转换。如果事件需要从安全域转换到次安全域，则栈访问电路被安排为在向次安全域的转换发生之前将某预定处理状态存储在安全栈上。什么被存储在安全栈上作为该预定处理状态将取决于事件类型，并且被选择来保证如果次安全程序代码试图使用与不同事件类型相关联的返回来返回到安全域，则这将被检测到。

[0009] 特别地，如果导致从安全域转换到次安全域的事件是第一事件类型，则栈访问电路存储在安全栈上的预定处理状态至少包括返回地址，并且该返回地址被存储在安全栈上的预定相对位置处（例如，在栈访问电路使用的栈帧的底部条目处来存储预定处理状态，或在相对于该底部条目的某预定偏移处来存储预定处理状态）。

[0010] 类似地，如果事件是第二事件类型，则在该事件中由栈访问电路存储的预定处理状态至少包括被存储在该预定相对位置处的第一值。该第一值是不是程序代码的有效地址的值。

[0011] 如果随后第一事件类型返回从次安全域到安全域被接收，则故障检查电路将从预定相对位置取回数据。由于第一事件类型返回正在被使用，则应该是该预定相对位置存储先前由栈访问电路存储的返回地址的情况。但是，如果相反它存储第一值，则这指示次安全程序代码正在试图通过第一事件类型返回来返回而不管原来从安全域到次安全域的转换是由于第二事件类型的事实的情况，并且在这种情况下，故障检查电路将标识第一故障情况，从而标记该活动。因此，该机制防止恶意用户使用很多种攻击，包括在给定先前导致从安全域转换到次安全域中的时间类型的情况下，试图通过使用与应该使用不同的事件类型返回到安全域中来间接执行到安全代码的任意分支。

[0012] 第一事件类型和第二事件类型可采用各种形式。但是，在一个实施例中，第一事件类型时函数返回，并且第二事件类型是异常。因此，上述机制将保证试图紧接从安全域异常分支到次安全域中而执行函数调用返回到安全域中的恶意攻击将被阻止。

[0013] 在一个实施例中，如果事件是所述第一事件类型，则处理电路被配置为向所述次安全域中被用于处理所述第一事件类型的程序代码提供虚拟（dummy）返回地址，所述虚拟返回地址不是程序代码的有效地址。然后，当完成次安全域中被用于处理所述第一事件类型的程序代码时，虚拟返回地址使所述返回地址（即，实际返回地址）从安全栈中被获取。特别地，当看到虚拟返回地址时，栈访问电路被安排为从安全栈加载预定相对位置处保留的数据，因此故障检查电路可检查该数据是指定所期望的返回地址，还是上面提到的指示故障情况的第一值。

[0014] 在一个实施例中，虚拟返回地址还可被用于保证当事件导致从安全域转换到次安全域时，随后的返回（使用该虚拟返回地址）将导致转换回到安全域，从而允许故障检查电路执行所需检查。

[0015] 在一个实施例中，如果事件是异常，则由栈访问电路存储的预定处理状态包括预定签名，并且故障检查电路被配置为当接收到从次安全域到安全域的异常返回时，如果预定签名不在由栈访问电路存储的预定处理状态中，则标识第二故障情况。

[0016] 通过这种方法，如果异常返回是紧接着通过函数调用来发生的从安全域到次安全域中的转换而从次安全域发布的，则已被栈访问电路存储在安全栈上的预定处理状态将不包括该预定签名，并且相应地这将指示使用不正确返回的意图攻击。

[0017] 预定签名可被置于安全栈内的任意适当位置，并采用任意想要的形式。但是，在一个特别有效的实施例中，预定签名形成被存储在预定相对位置处的上述第一值。因此，该单个预定签名使得第一和第二故障情况均能够通过故障检查电路来检测。

[0018] 在一个实施例中，如果事件是函数返回，则处理电路被配置为向所述次安全域中被所述函数调用标识的程序代码提供虚拟函数调用返回地址，该虚拟函数调用返回地址是从不是程序代码的有效地址的地址范围中选择的，并且所述预定签名具有与所述虚拟函数调用返回地址不同的值。通过保证预定签名具有与所述虚拟函数调用返回地址不同的值，这简化了处理电路的实现，因为要被执行的所需动作可只根据正在被分析的值来确定，而与该值是从函数调用返回产生的还是从栈中读取的无关。

[0019] 或者或此外，如果事件是异常，则处理电路被配置为向所述次安全域中被用于处理所述异常的程序代码提供虚拟异常返回地址，该虚拟异常返回地址是从不是程序代码的有效地址的地址范围中选择的；并且所述预定签名具有与所述虚拟异常返回地址不同的值。同样，通过使预定签名不同于虚拟异常返回地址可获得实现效率。

[0020] 在一个实施例中，预定相对位置是栈访问电路使用的栈帧内的预定位置。虽然理论上该位置可以是栈帧内的任意位置，但在一个实施例中，该预定位置是栈帧的底部位置，即，由栈指针指向的位置。

[0021] 一旦第一故障情况被引发，则存在很多方法可以处理第一故障情况。但是，在一个实施例中，处理电路响应于所述第一故障情况而在所述安全域内执行故障处理程序代码。通过保证故障处理程序代码在安全域内被执行，这防止了次安全域中的攻击者重新获取控制，从而相应地防止了重试攻击。

[0022] 作为用于阻止攻击的任意重试的替代机制，数据处理装置可被安排使得如果所述第一故障情况被标识，则栈指针值不被调整，使得第一事件类型返回的重放也将导致所述第一故障情况被标识。因此，通过不调整栈指针，攻击者不能仅消费安全栈的最后位置并然后重试攻击以便在安全栈内的不同位置处进入。

[0023] 在一个实施例中，上述两种机制均可被用于提供对抗来自次安全域的攻击的重试的鲁棒性。虽然以上机制是关于第一故障情况的处理来描述的，但是其也可被等同应用于第二故障情况的处理。

[0024] 在一个实施例中，可引起来自次安全域的潜在攻击的另一情境随着安全栈的初始化而出现。在该时刻，安全栈将是空的。如果在该时刻，正在被执行的程序代码在次安全域中，则程序代码可能试图返回到安全域中。该返回应被阻止，因为不存在从安全域到次安全

域的原转换来合理地从其返回。在一个实施例中，数据处理装置通过保证在安全栈的初始化时，第二值被存储在安全栈上的所述预定相对位置处来提供保护不受这种情景危害，所述第二值与所述第一值不同，并且也不是程序代码的有效地址。然后，故障检查电路被配置为当接收到从次安全域到安全域的返回时，如果存储在所述预定相对位置中的数据是所述第二值，则标识第三故障情况。根据该实施例，从次安全域的返回将导致故障检查电路加载被存储在安全栈上的预定相对位置处的数据，并且在该情境中，该数据将标识上述第二值，从而导致第三故障情况被检测。第三故障情况的处理可与上述针对第一和第二故障情况相同的方式发生，从而保证攻击的任意重试也被阻止。

[0025] 在一个实施例中，第三故障情况与第一故障情况相同，并且相应地，相同的故障处理代码被用于这两个故障情况。因此，在这种实施例中，故障检查电路不必区分第一和第二值。

[0026] 在一个实施例中，数据处理装置还包括多个寄存器，并且如果事件是异常，则由栈访问电路存储的预定处理状态还包括至少所述寄存器的子集的内容。在一个特定实施例中，其内容被栈访问电路存储的寄存器的数量将取决于该异常要被安全域还是次安全域来处理。

[0027] 在一个实施例中，多个栈还包括在所述次安全区域中的次安全栈，并且处理电路被配置为基于后台处理正在其中被执行的域来确定将数据存储到哪个栈或从哪个栈加载数据。在本申请中，术语“后台处理”被用于指示被函数调用或异常打断的处理。至少对于异常而言，较高优先级异常可打断已在被处理的过程中的较低优先级异常。因此，如果异常处理程序自身被更高优先级异常（这里称为抢占异常）打断，则被打断的异常处理程序自身对于抢占异常可变为“后台处理”。

[0028] 从第二方面来看，本申请提供了一种在数据处理装置上处理数据的方法，所述数据处理装置包括：处理电路，所述处理电路用于响应于程序代码而执行数据处理操作；以及数据存储设备，所述数据存储设备用于存储数据，所述数据存储设备包括多个区域，所述区域包括安全区域和次安全区域，所述安全区域被配置为存储当在安全域中操作时可被所述处理电路访问而当在次安全域中操作时不可被所述处理电路访问的敏感数据；所述数据存储设备包括多个栈，所述栈包括在所述安全区域中的安全栈；并且所述方法包括：响应于需要从所述安全域转换到所述次安全域的事件而将预定处理状态存储到所述安全栈；如果所述事件是第一事件类型，则至少将被存储在所述安全栈上的预定相对位置处的返回地址存储为所述预定处理状态；如果所述事件是第二事件类型，则至少将被存储在所述预定相对位置处的第一值存储为所述预定处理状态，其中所述第一值不是程序代码的有效地址；并且在接收到从所述次安全域到所述安全域的第一事件类型返回时，如果存储在所述预定相对位置的数据是所述第一值，则标识第一故障情况。

[0029] 从第三方面来看，本申请提供了一种数据处理装置，所述数据处理装置包括：数据处理装置，所述数据处理装置用于响应于程序代码而执行数据处理操作；数据存储装置，所述数据存储装置用于存储数据，所述数据存储装置包括多个区域，所述区域包括安全区域和次安全区域，所述安全区域被配置为存储当在安全域中操作时可被所述处理电路访问而当在次安全域中操作时不可被所述处理电路访问的敏感数据；所述数据存储装置包括多个栈，所述栈包括在所述安全区域中的安全栈；所述处理装置包括栈访问装置，所述栈访问装

置用于响应于需要从所述安全域转换到所述次安全域的事件而将预定处理状态存储到所述安全栈；如果所述事件是第一事件类型，则由所述栈访问装置存储的所述预定处理状态至少包括被存储在所述安全栈上的预定相对位置处的返回地址；如果所述事件是第二事件类型，则由所述栈访问装置存储的所述预定处理状态至少包括被存储在所述预定相对位置处的第一值，其中所述第一值不是程序代码的有效地址；并且所述处理装置还包括故障检查装置，所述故障检查装置用于响应于接收到从所述次安全域到所述安全域的第一事件类型返回，如果存储在所述预定相对位置的数据是所述第一值，则标识第一故障情况。

附图说明

[0030] 本发明将参考如附图所示的本发明的实施例，仅通过示例的方式来进一步地说明，其中：

[0031] 图 1 根据一个实施例，示出了具有带有安全区域和非安全区域的数据存储设备的数据处理装置；

[0032] 图 2 根据一个实施例，示意性地示出了存储器地址空间；

[0033] 图 3A 和 3B 示出了所述实施例的技术意欲提供防护的两个不同攻击情境；

[0034] 图 4A 根据一个实施例，示意性地示出了在接收到函数调用或异常时，预定处理状态在安全栈上的存储，而图 4B 示出了对于各种情境，该预定处理状态的形式；

[0035] 图 5 是示出了在一个实施例中，当接收到函数调用时处理电路的操作的流程图；

[0036] 图 6 是示出了在一个实施例中，当接收到异常时处理电路的操作的流程图；

[0037] 图 7 是示出了在一个实施例中，当处理分支操作时处理电路的操作的流程图，包括分支操作是函数调用返回或异常返回的情况；

[0038] 图 8A 和 8B 根据一个实施例，示意性地示出了当安全栈被初始化时，存储在安全栈上的预定签名可如何被用于防止从次安全域分支到安全代码中的无权尝试；

[0039] 图 9 示出了当处理末尾连锁异常的链时执行状态保存和恢复操作的示例；

[0040] 图 10 示出了使用异常返回值的一部分来指示用于控制状态保存操作的状态信息的示例；

[0041] 图 11 是更详细地示出了执行被调用者寄存器的状态保存的第一示例的流程图；

[0042] 图 12A 是示出了在进入最初异常时设置异常返回地址的示例的流程图；

[0043] 图 12B 是示出了在进入末尾连锁异常时设置异常返回地址的示例的流程图；

[0044] 图 13 是示出了在从异常返回时所执行的状态恢复的示例的流程图；

[0045] 图 14 是更详细地示出了异常返回处理的流程图；

[0046] 图 15 和 16 示出了使用图 10 所示的状态信息可避免不必要的状态保存和恢复操作的示例；

[0047] 图 17 示出了较高优先级异常抢占较低优先级异常的示例；并且

[0048] 图 18A 和 18B 示出了当在执行状态保存的过程中发生抢占时，可如何执行状态保存的示例。

具体实施方式

[0049] 图 1 示出了可以是例如微控制器的数据处理装置 10。其包括用于处理指令的处理

电路 20，以及用于存储由处理电路 20 进行处理的数据并且用于存储处理电路 20 执行的程序代码的数据存储设备 30。

[0050] 数据存储设备 30 具有不同安全性的两个区域：安全区域 32 和非安全区域 34（这里也称为次安全区域）。存储在安全区域 32 中的数据在存储在非安全区域 34 中的代码执行时不能被其访问。

[0051] 数据处理装置 10 还具有存储器保护单元 MPU 40，其控制对安全区域 32 和非安全区域 34 的访问。虽然该控制可通过存储器保护单元来执行，但在其他实施例中，其可由处理装置内的电路以更分布式的形式完成，该电路监测当前正在被执行的代码所存储的区域，并基于此来控制对存储器的不同区域的访问。

[0052] 在该实施例中，处理电路 20 正在执行的域的安全性可根据当前正在被执行的代码所存储的区域来确定。因此，存储在安全数据存储设备 32 中的安全程序代码在安全域中被执行，并且使用安全栈 36 来存储数据值。类似地，存储在非安全数据存储设备 34 中的非安全代码在非安全域中被执行，并且使用非安全栈 38 来在执行期间存储数据值。用于确定当前操作域的其他技术也是可以的，例如，基于随着控制流改变指令来标识处理器应该在哪个域操作的目标域值。

[0053] 处理电路 20 具有用于获取要被执行的指令的获取电路 22。其还具有用于对这些指令进行译码的译码电路 24，以及用于执行这些指令的执行电路 26。要被执行的指令由获取电路 22 通过存储器保护单元 MPU 40 从数据存储设备 30 来获取。指令和数据通过 MPU 40 取回，MPU 40 控制对安全区域和非安全区域的访问并将安全数据与非安全侧隔离。

[0054] 在该实施例中，存在寄存器组 60，其具有在数据处理过程中使用的通用寄存器。这些通用寄存器具有程序计数器 PC，其指示哪个指令是下一个将被执行的指令；以及栈指针 SP，其指示下一个数据访问应在栈中的哪个点做出。在该实施例中，因为在安全侧存在栈，并且在非安全侧存在栈，因此存在安全栈指针 SSP 和非安全栈指针 NSP，但是在任意一个时刻，这些指针中只有一个是正在被执行的程序直接可见的。应该注意的是，在一些实施例中，对于每个栈可存在多个栈指针，但是仍然在任意一个时刻只有一个将是可见的。在寄存器组 60 中还有用于存储正被处理电路 20 处理的数据值的通用寄存器。在该实施例中，这些寄存器被标记为 R_0 到 R_{12}。

[0055] 寄存器组 60 还包括连接寄存器 (link register)65，当异常被获取或函数被调用时，其可被用于存储返回值。返回值允许系统确定返回是异常返回还是函数返回，并且确定当从异常或函数返回时需要什么处理。不同种类的返回值可被使用，包括：函数返回地址，其指示随着函数的完成而要被处理的程序指令的地址；指示函数返回的虚拟函数返回值，对于该函数返回，实际函数地址已被存储到安全栈以向次安全域隐藏实际函数地址；以及异常返回 (EXC_RETURN) 值，其指示异常返回，并且可包括信息（例如，后台处理被处理的域的安全等级的指示），该信息可使处理器确定如何处理该异常返回，例如，当恢复状态时访问哪个栈以及需要加载多少个寄存器。返回值的不同形式稍后将进行说明。

[0056] 图 1 还示出了附加寄存器组 64，其具有附加专用寄存器，例如，浮点寄存器。在一个实施例中，可在安全配置寄存器 (SCR)62 中设置值来标识附加寄存器组 64 中是否有寄存器能够存储安全数据，并且如果是，则这些寄存器将被认为形成一组寄存器（与寄存器组 60 的寄存器一起）的一部分，当控制从安全域到次安全域进行转换（反之亦然）时，该组寄

存器需要被管理。

[0057] 存在很多可引起从安全域到次安全域转换（反之亦然）的机制。根据所述实施例，用于在安全性域之间进行转换的一个允许的机制是函数调用机制，其中函数调用可被发布以使得当前软件程序的执行暂停，以便使能向该函数调用所标识的另一软件程序转换，该另一软件程序将在安全域还是次安全域中被执行，取决于该另一软件程序在数据存储设备30中被存储在哪里。一旦该另一软件程序已被执行，则执行函数调用返回以返回到被暂停的原软件程序的执行。虽然一些函数调用将标识在与当前软件程序相同的域中将被执行的目标软件程序，但在其他实例中，目标软件程序可需要在与当前软件程序正在执行的域不同的域中执行。

[0058] 根据所述实施例，可被用于在安全性域之间进行转换的另一机制是异常机制。根据该机制，当异常发生时，当前软件程序的异常将被暂停，并且替代的是，执行将分支到用于处理异常的异常处理程序，所使用的异常处理程序取决于发生的异常的类型。一旦异常处理程序已被执行，则异常返回将被用于返回到由于发生异常而被暂停的原软件程序。

[0059] 异常控制电路28被提供以控制异常的获取，并且其中这些异常导致从更安全到次安全域的转换，可存储敏感数据的一组寄存器将在获取异常之前被清除，以避免存储在这些寄存器中的数据可被次安全侧获得。存储在这些寄存器中的一些或全部中的状态将在栈访问电路70的控制下被存储在安全栈上，使得在从异常返回时，该状态能够被恢复。为了异常处理，用于控制将寄存器内容存储到适当的栈的栈访问电路70可被认为形成异常控制电路28的一部分。但是，更一般地，还存在与在需要从安全域到次安全域转换的函数调用发生时将预定处理状态存储到安全栈相关联的一些栈访问电路70，因此在图1中，栈访问电路70被标识为与异常控制电路28分离。

[0060] 此外，如图1所示，故障检查电路75被提供用于检测在次安全域中的软件试图通过返回机制返回到安全域的情况，该返回机制具有与被用于从安全域转换到次安全域中的原机制不同的类型（例如，通过使用函数调用返回来从异常中返回，或使用异常返回来从函数调用中返回）。虽然故障检查电路75的一部分可被认为被包含在异常控制电路28中（即，负责检查异常返回的部分），但是故障检查电路75的其他部分可被分布在处理电路的硬件中的其他地方。例如，在一个实施例中，负责检查函数调用返回的部分可与获取电路22相关联。故障检查电路75的操作稍后将进行更详细地探讨。

[0061] 虽然在图1中数据存储设备30包括单个安全区域和单个次安全区域，但是这里所述的技术同样适用于包括两个以上不同安全性区域的不同实现。

[0062] 此外，虽然图1将异常控制电路28和栈访问电路70展示为与处理电路20的其他部分（例如，执行级26）分离，但是实际上，异常控制电路28和栈访问电路70可至少部分重新使用处理电路20的一些元件来控制异常处理和栈操作。

[0063] 图2示意性地示出了可被用于一个实施例的存储器地址空间。存储器地址空间100将被分区以形成一个或多个安全区域和一个或多个次安全区域。为了简洁起见，在图2中假设存在单个安全区域和单个次安全区域。安全地址空间的一部分102将被留出用于存储将在安全域中被执行的程序代码，而安全地址空间的另一部分104将被分配给安全栈36。安全地址空间的剩余部分106将被用于各种目的，例如，作为存储器堆，作为在执行过程中分配的自由空间等。次安全地址空间也将以类似的方式进行分区，从而提供用于存储

将在次安全域中被执行的程序代码的部分 110，被分配给次安全栈 38 的部分 112，以及剩余部分 114。

[0064] 根据所述实施例，存储器地址空间还包括预留区域 120，在该区域中的任意地址都不是程序代码的有效地址。如以下将进行的更详细的探讨那样，这些预留地址中的一些被用于向所述实施例提供想要的功能。

[0065] 为了提高速度和效率，上述实施例提供了用于在安全与次安全域之间进行转换的两种机制，即，函数调用机制和异常机制。但是，多域转换机制的使用增加了对恶意攻击的脆弱性，例如，允许次安全域中的软件试图通过返回机制返回到安全域，该返回机制具有与被用于从安全域转换到次安全域中的原机制不同的类型。两个特定攻击情境被示于图 3A 和图 3B 中。如图 3A 所示，在点 145 处，假设一些安全代码正在执行，并且在点 150 处，对存在于次安全域中的函数进行函数调用。然后该函数在点 155 处被执行，但是在点 160 处，次安全软件试图通过使用异常返回机制来任意分支到安全代码中。

[0066] 类似地，在图 3B 中，在点 175 处，假设一些安全代码正在执行，并且在点 180 处，发生异常（例如，中断），导致分支到次安全域中的异常处理程序。该异常处理程序在点 185 处被执行，但随后在点 190 处，次安全域中的软件试图做出向安全域的函数返回。

[0067] 如果系统的安全性要被维护，则以上两个攻击情境均需要被防止，因为如果这样的情境被允许发生，则这将允许次安全软件试图做出进入到安全代码中的任意分支，这可提供用于获得对安全数据的访问的机制。

[0068] 为了免受这样的攻击，当函数调用或异常发生，导致从安全域到次安全域的转换时，栈访问电路 70 被安排为将预定处理状态存储到安全栈的栈帧上，如将参考图 4A 和 4B 进行更详细地探讨那样。在图 4A 中，假设安全程序 FOO 当前正在安全域中执行，然后在 FOO 完成之前发生函数调用或异常。在函数调用或异常发生之前，通常 FOO 会使用安全栈 200 上分配的栈帧 205，以便存储 FOO 所使用的临时数据。当函数调用或异常发生时，则分离栈帧 210 将被分配给栈访问电路 70，然后栈访问电路将在分支到由函数调用所标识的所需软件程序，或分支到处理异常所需的异常处理程序之前在栈帧 210 中存储预定处理状态。

[0069] 首先考虑异常发生的情况，然后如果异常处理程序要在安全域内执行，则由栈访问电路存储在栈帧 210 中的预定处理状态包括图 4B 中标识的寄存器内容 215。这些寄存器在这里被称为“调用者”寄存器，并且这些寄存器是栈访问电路 70（或异常控制电路 28）将一直负责把其状态保存到栈上，而不管异常处理程序将要在哪个域执行的寄存器。默认情况下，然后异常处理程序将负责寄存器的剩余部分（这里称为“被调用者”寄存器）的状态。特别地，然后异常处理程序将在异常处理程序的主体中重新使用被调用者寄存器之前，将被调用者寄存器的状态保存到与异常处理程序正在其中执行的域相关联的栈上。此外，一旦异常处理程序完成，则异常处理程序将在发布异常返回之前，负责恢复这些被调用者寄存器的状态（通常通过将其从栈中复制回相关的寄存器中）。

[0070] 但是，根据所述实施例，在异常将需要从安全域转换到次安全域，并且先前的后台处理在安全域中的情况下，栈访问电路 70 额外负责在使处理电路转换到执行异常处理程序之前，将被调用者寄存器的状态保存在栈帧 210 中。因此，如图 4B 所示，在该情况下，存储在栈帧 210 中的预定处理状态采用形式 220。应该认识到，图 4B 中特定标识的调用者和被调用者寄存器只是调用者和被调用者寄存器可被如何分区的示例，而到底哪些寄存器被

认为是调用者寄存器或被调用者寄存器将基于实现方式而不同。

[0071] 同样如图 4B 所示，在异常处理程序在次安全域，从而需要从安全到次安全域的转换的情况下，栈访问电路也被安排为在栈帧 210 中的预定相对位置处存储预定签名 222，在该实施例中，示出该预定相对位置是栈帧中的底部位置。该预定签名可采用各种形式，但在一个实施例中被选择为具有不对应于程序代码的有效地址的值。在一个实施例中，预定签名被选择为具有存储器地址空间的预留部分中的地址值 120 中的一个。在一个特定实施例中，预定签名具有值 0xF0A5125A。

[0072] 通常，当函数调用发生时，正常情况下栈访问电路 70 将不在栈帧 210 中保存任何预定处理状态。因此，如果函数调用标识仍在安全域中的目标代码，则没有预定处理状态被存储。但是，如果函数调用标识在次安全域中的目标代码，则栈访问电路 70 被安排为在栈帧 210 中存储返回地址 225，一旦函数调用已完成，该返回地址 225 将被需要以便继续执行 FOO。在需要转换到次安全域的异常发生的情况下，该返回地址 225 被存储在与预定签名 222 被存储的预定相对位置相同的预定相对位置，即，栈帧内的底部位置。

[0073] 取代安全栈内存储的实际返回地址 225，次安全域内的目标代码被提供有实际上不是程序代码的有效地址的虚拟函数调用返回地址。在一个实施例中，该虚拟函数调用返回地址从预留地址 120 中的一个中选择，并且进一步被选择为具有与上述预定签名 222 不同的值。

[0074] 当函数调用发生时，软件仍然可执行寄存器状态的状态保存，即使根据图 4B 所示的栈帧，状态保存不是硬件所需要的。将寄存器分为调用者和被调用者寄存器对这样的函数调用是有用的，因为其既允许在函数调用前被执行的软件（调用者软件），又允许在函数调用后被执行的软件（被调用者软件）影响哪些寄存器经受状态保存。例如，如果调用者软件知道其没用过某些调用者寄存器，或已经结束使用一些调用者寄存器中的数据，则这些寄存器中的值在函数调用后将不必被维护，因此这些寄存器不必经受状态保存。类似地，如果被调用者软件将不会使用某些被调用者寄存器，则针对这些寄存器被调用者软件不必执行状态保存。因此，允许调用者和被调用者软件影响用于状态保存的寄存器的选择通过降低适当时候的状态保存的数量使得性能提升。相比之下，如果只有调用者软件或只有被调用者软件负责状态保存，则一些寄存器可能被不必要地保存，以防其他软件需要该寄存器被保存。

[0075] 图 5 是根据一个实施例，示出了当函数调用发生时由处理电路执行的步骤的流程图。在步骤 300，确定当前执行的代码是否正在安全域中执行。如果不是，则过程直接进行到步骤 310，其中连接寄存器 (LR) 被设置为等于返回地址。此后，过程进行到步骤 325，其中执行分支到新函数（即，被标识为函数调用的目标的程序代码）。由于返回地址已在连接寄存器内被设置，当新函数完成时，正确的返回地址可被指定在函数调用返回中。

[0076] 如果在步骤 300 确定当前执行的代码正在安全域中执行，则在步骤 305，确定作为函数调用的目的地的新函数是否要在次安全域中执行。确定这一点存在很多方法，但在一个实施例中，这仅仅通过确定函数调用的目标地址是与安全存储器区域还是次安全存储器区域相关联的地址来获得。如果其与次安全存储器区域相关联，则表明函数调用地址的目的地在次安全域中。如果函数调用的目的地不在次安全域中，则过程再次进行到步骤 310。

[0077] 但是，如果函数调用的目的地在次安全域中，则表明需要从安全域转换到次安全

域，并在此处过程进行到步骤 315，其中连接寄存器被设置为如上所述的虚拟函数调用返回地址。此外，在步骤 320，实际返回地址被栈访问电路 70 推到安全栈上。其后，过程进行到步骤 325，其中执行分支到作为函数调用的目标的新函数。

[0078] 不仅是图 5 所示的硬件操作，而且在函数调用之前执行的软件也可执行来自寄存器的数据的状态保存，并且在函数调用之后执行的软件也可执行来自被调用者寄存器的数据的状态保存。

[0079] 图 6 是示出了当异常（异常的一个示例是中断）发生时处理电路的操作的流程图。在步骤 400，连接寄存器被设置为选定的虚拟异常返回地址，在一个实施例中，存在多个不同的虚拟异常返回地址可被选择。在一个实施例中，每个可能的虚拟异常返回地址将会是不是程序代码的有效地址的地址。步骤 400 的更多细节稍后将进行说明。其后，在步骤 405，调用者保存寄存器被推到后台处理域（即，后台代码正在其中执行的域）的栈上。

[0080] 其后，在步骤 408，确定后台处理是否在安全域中，并且如果不是，则过程直接进行到步骤 432。如将在下面说明的那样，后台处理是否在安全域中可基于位于连接寄存器 65 中的异常返回值来确定。如果后台处理在安全域中，则在步骤 410，确定当前执行的代码是否正在安全域中执行，并且如果不是，则过程再次直接进行到步骤 432。但是如果在步骤 410 确定当前执行的代码（可以是后台处理或者可以是先前的末尾连锁异常）正在安全域中执行，则在步骤 412 确定目的地是否在次安全域中，即，处理异常所需的异常处理代码是否将在次安全域中被执行。如果不是，则过程再次直接进行到步骤 432。但是，如果目的地在次安全域中，则表明在后台处理在安全域的情况下，将发生从安全域到次安全域的转换。因此，过程进行到步骤 420，其中被调用者保存寄存器被推到后台域的栈上，并且上述预定签名被写到栈帧的底部。与后台处理相关联的栈可根据位于连接寄存器 65 中的异常返回值来确定。稍后将提供这一步骤的更多细节。

[0081] 在步骤 430，异常控制电路 28 保证所有寄存器被清除。在一个示例中，所有寄存器可在步骤 430 被清除。在替换实施例中，寄存器也可以在其被推到栈上时被清除，因此调用者寄存器可在步骤 405 期间被清除，而被调用者寄存器可在步骤 420 或 430 期间被清除。

[0082] 接着步骤 430，或者如果在步骤 408、410 和 412 检查的条件中的任一项不发生，则过程随后进行到步骤 432。如果异常入口是到末尾连锁异常，则连接寄存器 65 用新的异常返回值进行更新。这一步骤将在下面针对图 12B 进行更详细地探讨。

[0083] 在步骤 435，处理分支到异常处理程序。

[0084] 然后，异常处理程序可执行被调用者寄存器的状态保存。在一个实施例中，异常处理程序可一直执行被调用者寄存器的状态保存，即使在被调用者寄存器中的数据在步骤 420 已经被推入栈并被清除的情况下（在这种情况下，硬件会将来自被调用者寄存器的数据存储到安全栈，而软件会将来自被调用者寄存器的被清除的值存储到次安全栈）。或者，异常处理程序能够检测被调用者寄存器的状态保存是否已被执行，并且如果是，则可省去被调用者寄存器的状态保存。

[0085] 图 7 是示出了分支操作如何进行处理的流程图。该分支操作可包括正常分支处理操作、函数调用返回和异常返回。虽然图 7 示出了分支操作，但在替换实施例中，类似的机制可与可导致指令流中的改变的任意指令（例如，具有作为目的地寄存器的程序计数器的加载指令）结合使用。

[0086] 在步骤500，确定分支操作所指定的目标地址是否在预留地址范围120内，在该特定示例中是目标地址是否大于或等于0xF0000000的情况。如果不是，则表明是正常分支活动，并且过程进行到步骤505，其中正常分支处理被执行。因为分支操作的处理将被本领域的技术人员很好地理解，因此这里不提供正常分支处理的进一步的细节。

[0087] 如果在步骤500确定目标地址在预留地址范围120内，则在步骤510，确定目标地址是否与虚拟函数调用返回地址一致。在一个特定实施例中，虚拟函数调用返回地址是0xF7FFFFFF。如果目标地址确实对应于虚拟函数调用返回地址，则在步骤515，安全栈内的栈帧210的上述预定相对位置被访问，以便从安全栈中读取返回地址。其后，过程返回到步骤500，其中，假设函数调用返回已被正确用于从先前的函数调用返回，在步骤515读取的返回地址将是程序代码的真实地址，并且相应地在步骤505，过程将分支到正常分支处理。但是，相反如果函数调用返回已被错误地用作从异常返回的机制，则当栈帧220在步骤515被访问时，预定签名222将作为实际返回地址被取回。当该地址随后在步骤500被分析时，将确定其在预留范围内，但在步骤510，将确定该地址不是虚拟函数调用返回地址。然后过程将进行到步骤520，其中还将确定该地址不是有效异常返回地址中的一个，并且相应地，过程将分支到步骤525，其中故障将被创建以标识函数调用返回已被错误使用。

[0088] 现在考虑异常返回被指定的情况，则如果该异常返回与导致从安全域到次安全域的转换的先前异常相关联，则这会将若干不同有效异常返回地址中的一个标识为目标地址，这些可能的异常返回地址中的每一个都在预留范围120内，并且与预定签名和虚拟函数调用返回地址都不同。相应地，过程将从步骤500进行到步骤510，并从步骤510到步骤520，其中跟随“是”的路径将随后到达步骤530。异常返回处理步骤530的细节稍后将进行说明。在具有低于或等于当前异常的优先级而高于后台处理的优先级的新的异常在该当前异常返回的时刻正等待执行的情况下，则该新的较低优先级的异常将被作为末尾连锁异常（即，该新的异常将紧接在当前异常完成后而在返回到先于正被获取的当前异常发生的后台处理之前被立即处理），并且如图7所示，过程将分支回图6中的步骤408。

[0089] 一旦步骤530的异常返回处理已被执行，并且没有进一步的末尾连锁异常等待处理，则过程进行到步骤535，其中确定异常返回处理步骤530是否已经指示预定签名被预期在存储于栈帧210内的处理状态中。如先前所探讨的那样，这将是如果异常导致从安全域到次安全域的转换，则后台代码在安全域中被执行的情况。如果没有预定签名是预期的，则过程仅进行到步骤540，其中相关寄存器值从栈帧中出栈。这一过程稍后将针对图13进行更详细地说明。

[0090] 但是，如果在步骤535，确定预定签名是预期的，则过程进行到步骤545，其中栈帧210的底部被读取，并且取回的值与预定签名对比。在步骤550，确定是否存在匹配，并且如果是，则过程进行到步骤540。但是，如果不存在匹配，则过程进行到步骤555，其中故障被创建，表明异常返回已被错误地用于试图从函数调用返回。特别地，如果异常返回正被错误地用于试图从函数调用返回，则在步骤545读取安全栈将导致返回地址225被取回，该返回地址225将不匹配预定签名，并将相应地导致故障在步骤555被创建。

[0091] 存在很多方法来处理在步骤525或步骤555被创建的故障。在一个实施例中，处理电路响应于任一故障情况而在安全域内执行适当的故障处理程序代码。通过保证故障处理程序代码在安全域内被执行，这防止了次安全域中的攻击者重新获得控制，并相应地防

止了重试攻击。

[0092] 作为用于阻碍任意攻击重试的替换机制，如果任一故障情况被识别，则数据处理装置可被安排使得栈指针值不被调整，使得返回的重演也将导致同样的故障情况被识别。因此，通过不调整栈指针，攻击者不能仅消费安全栈的最后位置，然后重试攻击以便在安全栈内的不同位置处进入。在一个实施例中，以上两种机制可被用于提供对抗来自次安全域的攻击的重试的鲁棒性。

[0093] 在一个实施例中，可引起来自次安全域的潜在攻击的另一情境随着安全栈的初始化而出现。在该时刻，安全栈将是空的。如果在该时刻，正在被执行的程序代码在次安全域中，则程序代码可能试图返回到安全域中。该返回应被阻止，因为不存在从安全域到次安全域的原转换来合理地从其返回。图 8A 和 8B 示出了可被用于阻止这样的返回成功的机制。如图 8A 所示，当安全栈 600 被初始化时，预定签名值 602 被存储在安全栈上（在一个实施例中，当栈被初始化时，该值由软件来存储）。该预定签名值通常将不同于先前参考图 4B 所述的预定签名 222，但是仍将是不对应于程序代码的有效地址的值。在一个特定实施例中，预定签名 602 具有值 0xF05AEDA5。

[0094] 如图 8B 所示，如果在点 605 处，软件正在次安全域中执行，然后在点 610 处试图返回到安全域中，则故障检查电路将在点 615 处从安全栈中读取预定签名值 602，并将确定其不对应于程序代码的有效地址，并将相应地创建故障。因此，从次安全域中返回将被阻止。

[0095] 特别地，如果在点 610 处的返回是函数调用返回，则在点 615 生成的故障与在图 7 中的步骤 525 创建的故障相同，并且相同的故障处理代码可被用于响应该故障。因此，在该实施例中，故障检查电路不需要在预定签名 602 与上述预定签名 222 之间进行区分。相反，如果在点 620 处的返回是异常返回，则预定签名值 602 与预定签名值 222 不同的事实将意味着图 7 的步骤 550 将检测不到匹配，并且相应地在步骤 555，故障将被引发。

[0096] 如针对图 7 的步骤 530 所述的那样，当从一个异常返回时，第二异常可能正等待处理，并且可能在返回到在第一异常之前被执行的后台处理之前被处理。这被称为末尾连锁。图 9 示出了接着第一异常，第二和第三异常在转回后台处理之前被末尾连锁的示例。

[0097] 如图 4B 的栈帧 220 所示，如果后台处理在安全域中，并且存在从安全处理到次安全状态中的异常的转换，则调用者保存寄存器和被调用者保存寄存器均被异常控制电路 28 推入安全栈。但是，通常在安全域中被处理的异常将预期只有调用者保存寄存器会被保存到栈（如图 4B 的栈帧 215 所示），而被调用者保存寄存器的保存留给异常处理程序。因此，如图 9 所示，当进入第一异常而在点 700 处执行被调用者寄存器的附加状态保存时，然后当在点 705 处转到安全域中的第二异常时，被调用者保存寄存器可从栈中弹出以恢复栈帧 215，其被预期末尾连锁第二异常在安全域中进行处理。

[0098] 但是，如果要在次安全域中被处理的末尾连锁第三异常发生，则在点 710 处，被调用者寄存器需要再次被推入栈，因为从安全域到次安全域的转换意味着被调用者寄存器中的数据需要对次安全处理隐藏。最后，当第三异常完成，并且没有进一步的末尾连锁异常时，则在图 9 的点 715 处，调用者和被调用者保存寄存器均从栈中被弹出。

[0099] 图 9 示出了当在末尾连锁异常之间进行切换时所执行的若干状态保存操作和状态恢复操作。这些操作花费时间，因此延迟了到来异常的处理，从而降低了处理性能。因此，避免这些操作中的一些是有用的。本技术认识到当从次安全异常切换到末尾连锁的安全异

常时，被调用者寄存器不必要从安全栈中弹出。将被调用者寄存器留在栈上是可接受的，因为这不影响异常的处理，并且所有被调用者保存值被允许从异常将在其中被处理的安全域中访问。因此，被调用者寄存器可响应于导致从安全域转换到次安全域的第一异常而被保存到栈。当处理从次安全异常切换到安全末尾连锁异常时，被调用者寄存器的恢复可被省去。对于导致从安全域转换到次安全域的进一步异常，通过硬件的被调用者保存寄存器的附加状态保存可被省去。

[0100] 图 10 示出了在进入异常时可被存储在连接寄存器中的异常返回值的示例。在一个实施例中，连接寄存器 65 不需要将异常返回值保留异常处理程序被执行的整个时间，相反其可在开始时被设置为该值，然后立即移动到栈中。当异常完成时，软件分支到异常返回值所标识的地址，CPU 发现该地址是特定事件，因为其不是有效指令地址。在一个实施例中，上述虚拟函数调用返回地址在函数调用发生时也被存储在连接寄存器中，并以类似的方式进行处理。

[0101] 异常返回值包括用于确定在进入导致从安全域转换到次安全域的异常时是否需要由硬件保存被调用者保存寄存器。异常返回值具有若干不同的可能值，每个可能值对应于地址空间的预留区域 120，并且与虚拟函数返回地址以及上述预定签名 222 和 602 不同。异常返回地址包括状态保存状态值字段 720 和安全性字段 725。

[0102] 状态保存状态值字段 720 存储状态保存状态值 SCRS，其指示对于接下来导致从安全域转换到次安全域的异常，是否需要被调用者保存寄存器的附加状态保存。在图 10 的示例中，值 0 指示附加状态保存可被跳过，而值 1 指示需要附加状态保存，但是也可使用其他状态值的映射。

[0103] 安全性字段 725 存储安全性域值 S，其指示在末尾连锁异常的当前链中的初始异常之前正被执行的后台处理是在次安全域还是安全域中。当进入末尾连锁异常的链中的每个异常时，异常返回值将在图 6 的步骤 432 中被设置为新的值。当设置异常返回值时，状态保存状态值字段 720 的值可被改变以影响后面要被执行的状态保存处理。以这种方式，信息可从一个异常传递到另一个异常以指示对于异常之间的下一转换是否需要附加状态保存。这将参考图 11-16 进行解释。

[0104] 图 11 更详细地示出了用于将被调用者保存寄存器推入安全栈 36 并将预定签名写入安全栈 36 的图 6 的步骤 420。该附加状态保存基于异常返回值的状态保存状态值字段 720 的值。在步骤 750，处理器确定异常返回值的状态保存状态值字段 720 是否具有值 0。如果是，则跳过将被调用者保存寄存器推入栈，并且该方法继续图 6 的步骤 430。在这种情况下，被调用者保存寄存器的状态保存不是必要的，因为来自这些寄存器的数据将已经响应于先前的异常而被保存到栈。

[0105] 另一方面，如果在步骤 750，状态保存状态值不具有值 0，则该方法进行到步骤 755，并且来自被调用者保存寄存器的数据被推入安全栈。被调用者保存寄存器也被清除，使得其值对于次安全域中的后续处理是不可访问的。此外，在步骤 760，预定签名 222 被写到栈的底部。然后该方法也进行到图 6 的步骤 430。因此，根据图 11，附加状态保存是否被执行取决于状态保存状态字段的值。

[0106] 图 12A 示出了针对当执行后台处理时接收到的最初异常（如果存在末尾连锁异常的链，则其是第一个异常），在图 6 的步骤 400 设置异常返回寄存器的示例。在步骤 770，确

定处理器正在操作后台处理的当前域。如果当前域是安全域，则在步骤 775，安全性域字段 725 被设置为具有值 1 以指示安全域。如果当前域是次安全域，则在步骤 780，安全性域字段 725 被设置为值 0。不管哪个域是当前域，在步骤 790，状态保存状态值字段 720 被初始化为值 1，其指示在下一个从次安全域到安全域的转换处可能需要附加状态保存。然后，在步骤 795，异常返回值被写入连接寄存器 65，其中状态保存状态值字段 720 和安全性字段 725 被设置为在先前步骤中确定的值。然后该方法进行回到图 6 的步骤 405 以继续异常进入处理。

[0107] 图 12B 更详细地示出了用于设置末尾连锁异常的异常返回值的步骤 432 的示例。在步骤 800，该方法确定正被进入的异常是否是末尾连锁异常。如果不是，则该方法进行到图 6 的步骤 435 以分支到异常处理程序，而不执行图 12B 的剩余步骤。但是，如果该异常是末尾连锁异常，则在步骤 802，确定安全域值 S（来自图 10 中被用于终止链中的先前异常的 EXC_RETURN 值）是否是 1。如果不是，则 S = 0，并且过程进行到步骤 804 以将状态保存状态值设为 1。如果后台处理是次安全的（如由 S = 0 所指示的），则没有附加状态保存将被执行（图 6 的步骤 408 将导致附加状态保存步骤被省去）。虽然在这种情况下，状态保存状态值字段 720 将不会影响状态保存，但为了防止次安全域中的处理从状态保存状态值字段 720 获得信息，最安全的是在步骤 804 将该字段设置为默认值 1。

[0108] 另一方面，如果在步骤 802，安全性域值 S 等于 1，则过程进行到步骤 805，其中确定新的末尾连锁异常是否要在安全域中被处理。如果不是，则再次在步骤 804 将状态保存状态值字段 720 设为 1。

[0109] 如果新的异常要在安全域中被处理，则在步骤 806，确定当前处理是否在安全域中。如果不是，则在步骤 808，状态保存状态值字段 720 被设为 0，而如果当前处理是安全的，则在步骤 810，状态保存状态值字段 720 被设置为与先前异常相同的值。在步骤 812，新的异常返回值被写到连接寄存器 65，其中状态保存状态值字段 720 根据步骤 804、808 或 810 来设置。然后该方法进行到图 6 的步骤 435 以分支到异常处理程序。

[0110] 图 13 更详细地示出了用于使寄存器出栈的图 7 的步骤 540。基于异常处理的过去历史，该栈可包括只有调用者寄存器需要被恢复的栈帧 215 或调用者和被调用者保存寄存器都需要被恢复的栈帧 220。图 13 示出了处理器可以如何确定哪个栈帧将出现在栈上。在步骤 820，确定当前的异常返回是否正从次安全异常切换到安全后台处理。例如，安全域字段 725 可被用于确定后台处理是安全还是次安全的。如果处理不是正从次安全异常切换到安全后台处理，则在步骤 822，该过程确定当前异常返回是否正从次安全异常切换到安全后台处理，并且状态保存状态值字段 720 的值是否为 0。如果步骤 820 和 822 中确定的条件均不满足，则在步骤 824，只有调用者寄存器将其值从栈中恢复。但是，如果在步骤 820 和 822 中测试的条件中有一个满足，则在步骤 824 将数据恢复到调用者寄存器之前，在步骤 826，被调用者保存值被恢复到被调用者寄存器。因此，该系统可确定哪些寄存器需要将其寄存器状态恢复。

[0111] 图 14 更详细地示出了图 7 的异常返回处理步骤 530。在步骤 830，确定是否存在应为末尾连锁的待处理异常。如果该异常的优先级低于或等于刚结束的异常的优先级，而高于后台处理的优先级，则该异常应为末尾连锁的。如果存在末尾连锁异常要被处理，则该方法返回到图 6 的步骤 408 以处理进入末尾连锁异常。

[0112] 如果不存在末尾连锁异常，则该方法进行到步骤840和850，其中确定：(a)异常返回正从次安全异常切换到安全后台处理；还是(b)异常返回正从安全异常切换到安全后台处理；并且状态保存状态值字段是否具有值0。如果这些条件中有一个满足，则在步骤860，确定预定签名222是预期的。如果这些条件都不满足，则在步骤870确定预定签名不是预期的。然后该方法进行到图7的步骤535，其中基于预定签名222是否是预期的来执行处理以检测异常返回是否已被不适当地使用。注意，步骤840和850检查与图13的步骤820和822相同的两个条件，因为如图4B的栈帧220所示，当被调用保存寄存器值出现在栈上时，则预定签名也将是预期的。

[0113] 图15和16示出了图9所示的不必要的栈保存和恢复操作示例可如何使用状态保存状态值来避免。在图15中，当在点900处发生次安全异常时，后台处理在安全域中。如图12A所示，安全性域值S被设置为1以指示后台处理在安全域中，并且状态保存状态值SCRS也被设为1。当第一异常在点910完成时，存在优先级等于或低于第一异常而高于后台处理的待处理的第二异常。该异常是末尾连锁的，使得其在回到后台处理之前被处理。在该点处，被调用者保存寄存器的出栈被跳过。当切换到第二异常时，状态保存状态值SCRS被设为0，如图12B的步骤808所示，因为后台处理在安全域中(S＝1)，因此目的地是安全的而当前过程是次安全的。一旦第二异常在点920处完成处理，则末尾连锁的第三异常等待处理，并且要在次安全域中进行处理。由于状态保存状态值SCRS等于0，则根据图11的步骤750，附加状态保存可被省去，因为被调用者保存寄存器状态已被保存到栈。因此，第三异常可被更快处理，因为不存在与附加栈操作相关联的延迟。最后，在点930，处理返回到后台，并且根据图13的方法确定步骤820的条件是满足的，因此被调用保存寄存器状态和调用保存寄存器状态均从栈中弹出并恢复到相应的寄存器。

[0114] 图16示出了附加栈保存操作可被避免的另一示例。在这种情况下，当执行安全后台处理时，在点1000处发生要在安全域中进行处理的最初异常。因为目的地不是次安全域，因此根据图6的步骤412，被调用者保存寄存器的保存将被省去。在点1000，只有调用者保存寄存器状态被硬件推入栈。状态保存状态值被设为1，如图12A所示。在点1005，次安全第二异常是接着第一异常的末尾连锁异常。因为状态保存状态值SCRS等于1，则在点1005，附加状态保存根据图11的步骤750和755来执行，因此被调用者保存寄存器状态被推入栈。因为S＝1并且目的地是次安全的，则在图12B的步骤804，对于接下来的异常，状态保存状态值被设为1。在点1010发生从次安全第二异常到安全第三异常的末尾连锁转换，并且被调用者保存寄存器的出栈被跳过。在该点处，状态保存状态值SCRS根据图12B的步骤808而被设为0，因为S＝1，所以目的地是安全的而当前过程是次安全的。这意味着在点1015接着的末尾连锁异常处，被调用者保存寄存器的入栈可被跳过，因为状态保存状态值的值为0。如图16所示，即使最初异常不是导致从安全域切换到次安全域的转换，在接下来从安全域到次安全域的切换时执行的附加状态保存也可被执行一次，并在后面的转换时将不再重复。

[0115] 从图15和16可以看出，用于如图14所示设置异常返回值的机制意味着当异常正在次安全域中处理时，状态保存状态值SCRS将一直具有值1，同时，当处理安全异常时，状态保存状态值具有可变值0或1来指示在下一个安全到次安全的转换处是否需要附加状态保存。这是有用的，因为在次安全域中时将该值设置为固定值1防止了次安全域中的代码

能够从状态保存状态值中获得任意信息，状态保存状态值可允许信息被推测异常处理的过去历史。这提高了安全性。

[0116] 此外，如果次安全代码可篡改状态保存状态值，则这可影响在安全域中时的后续状态保存操作，这可导致安全性破坏。为了避免这一情况，处理电路在从次安全异常切换到安全异常时（例如，在图 16 的点 1010 处）可检测状态保存状态值仍具有在进入次安全域中的异常时被设置的固定值，并且如果该值已被改变，则触发错误。或者，没有错误可被触发，但是当返回安全域时，状态保存状态值可被重置为其在安全域中应具有的值，覆盖由次安全代码对状态保存状态值字段的任意修改。

[0117] 图 17 示出了异常的抢占的示例。每个异常可与优先级值相关联。如果当具有较低优先级的异常正被执行时发生具有较高优先级的异常，则高优先级异常可抢占较低优先级异常，并且在较低优先级异常未完成的情况下进行处理。虽然图 17 示出了优先级值的较大数值指示较高优先级的示例，但是也可用较低优先级值表示较高优先级，例如，用优先级值 0 指示高于优先级值 1 的优先级异常。

[0118] 如图 17 所示，当抢占发生时，被抢占的异常可被视为接下来的异常的后台处理。例如，在图 17 的点 1100，后台处理被打断以处理具有优先级值 1 的第一异常。在点 1105，发生了具有优先级值 4 的第二异常。由于其是比当前优先级等级更高的优先级，因此第二异常抢占第一异常，并且现在第一异常是后台处理。当在点 1110 从第二异常返回时，将执行状态恢复以恢复在第二异常发生的时刻由后台处理（即，第一异常）所使用的状态。当在点 1115 第一异常完成时，执行进一步的状态恢复以恢复由原后台用于处理的值。因此，在先前的实施例中，“后台处理”自身可以是根据被更高优先级异常抢占的异常的异常处理程序来执行的处理。

[0119] 图 18A 和 18B 示出了状态保存状态值在抢占期间有用处的示例。如果后台处理是安全的，并且发生次安全异常，则如上所述，被调用和调用保存寄存器均将被推入栈，以防止由次安全异常处理器对被调用寄存器状态的访问。但是，当该状态保存操作正被执行时，也可发生具有更高优先级的另一异常。在这种情况下，较高优先级异常将抢占原来的次安全异常，因此次安全异常将不被处理。如果抢占异常要在安全域中处理，则将不必将被调用者保存寄存器推入栈。

[0120] 图 18A 示出了当调用者保存寄存器正被推入栈时发生抢占异常的示例。在这种情况下，因为抢占异常是安全的，所以不必执行被调用者寄存器的附加状态保存，因此一旦调用者寄存器的入栈完成，则安全异常的处理就可开始。这避免了由将被调用者寄存器状态推入栈所导致的延迟。如果随后发生末尾连锁的次安全异常，则图如 18A 所示，被调用者寄存器状态可在此刻被推入栈。

[0121] 另一方面，图 18B 示出了当被调用者寄存器正被推入栈准备处理原来的次安全异常时，在安全域中发生抢占异常的示例。在这种情况下，当被调用者寄存器的状态保存完成时，继续推入被调用者寄存器并切换到安全异常可能更有效。这是因为停止被调用者寄存器的状态保存以及倒转已经完成的被调用者寄存器的入栈操作可比仅完成状态保存花费更长的时间。因此，如果状态保存被完成，则将存在较少延迟。如果被调用者寄存器的保存完成，则状态保存状态值 SCRS 可被设为 0 以指示附加状态保存已经发生，使得当发生进一步的次安全末尾连锁异常时，其可被跳过。因此状态保存状态值还使在处理抢占时的性能

提升。

[0122] 本申请的主题与一般指定共同待决的申请号为 13/368419 的美国申请和申请号为 1217531.1 的英国专利申请中所探讨的主题相关，这两个文档的全部内容被通过引用结合于此。

[0123] 虽然这里描述了特定实施例，但是应该认识到，本发明不限于此，并且在本发明的范围内可对其作出很多修改和添加。例如，在不脱离本发明的范围的情况下，可作出以下独立权利要求的特征与从属权利要求的特征的各种结合。

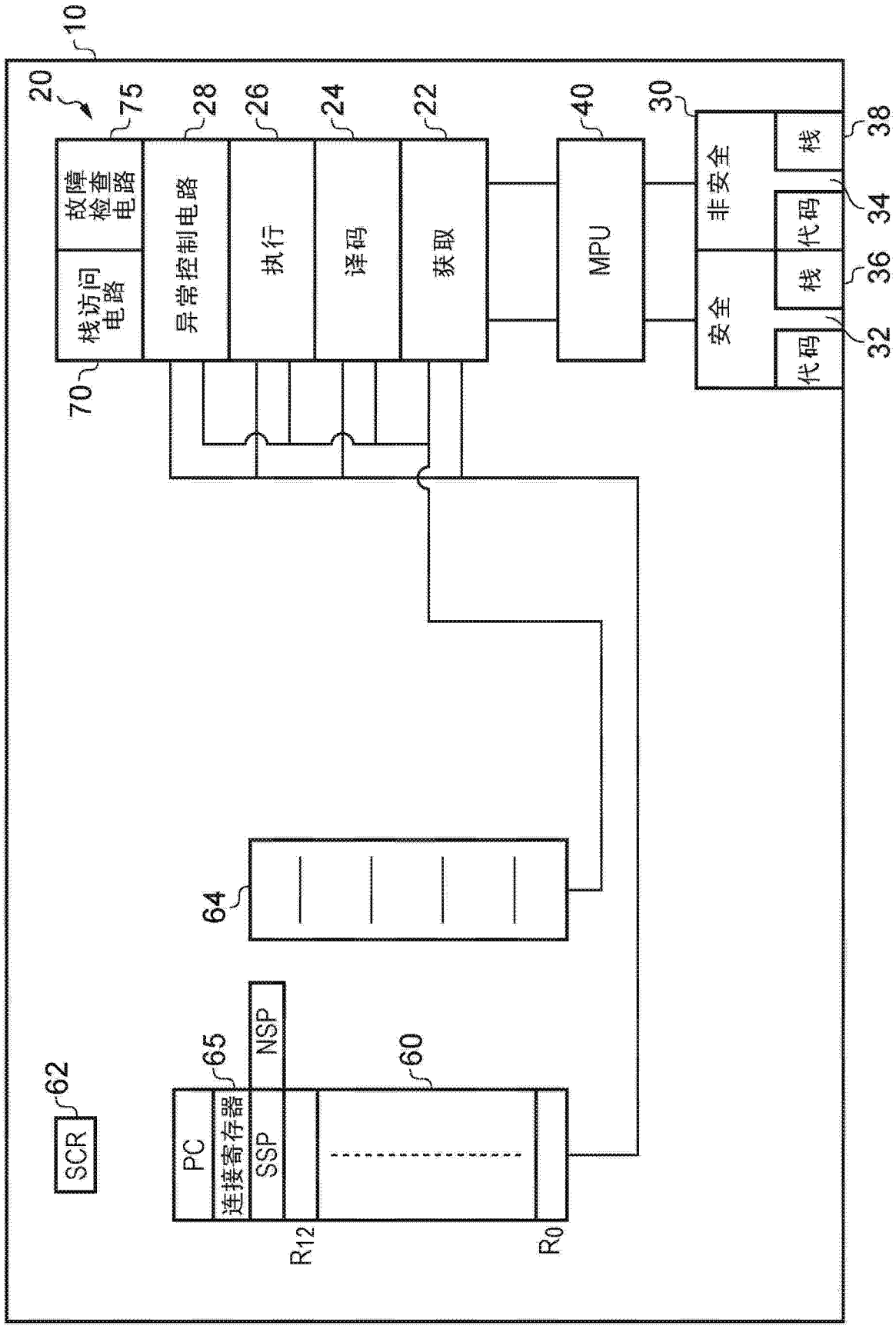

10
20
75
28
26
24
22
40
30
故障检查电路
栈访问电路
异常控制电路
执行
译码
获取
MPU
非安全
安全
栈
代码
栈
代码
38
34
36
32
70
64
62
SCR
65
NSP
60
PC
连接寄存器
SSP
R12
R0

图 1

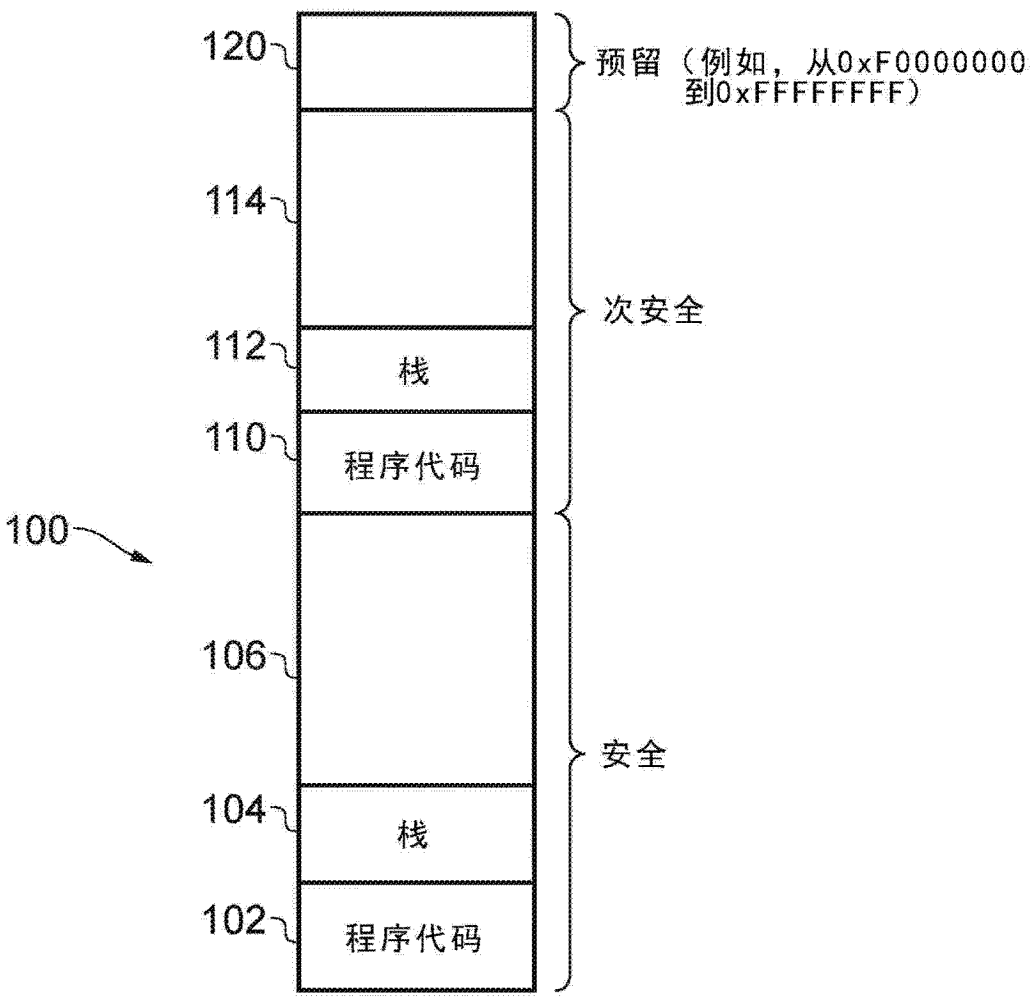

存储器地址空间
120
预留（例如，从0xF0000000
到0xFFFFFFFF）
114
次安全
112
栈
110
程序代码
100
106
安全
104
栈
102
程序代码

图 2

攻击情境

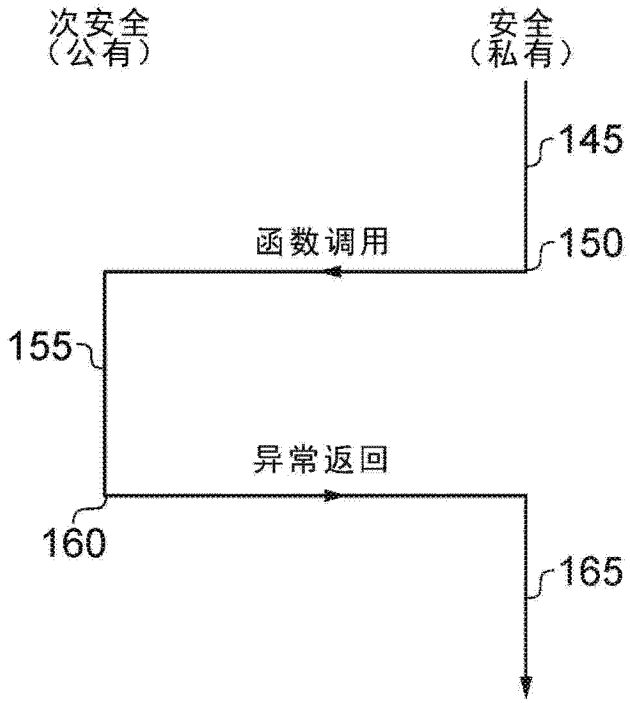

次安全
（公有）
安全
（私有）
145
函数调用
150
155
异常返回
160
165

图 3A

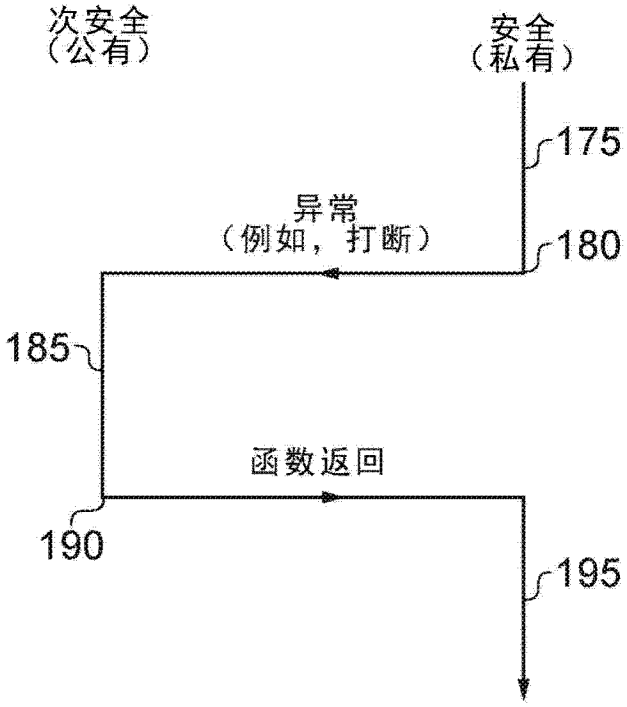
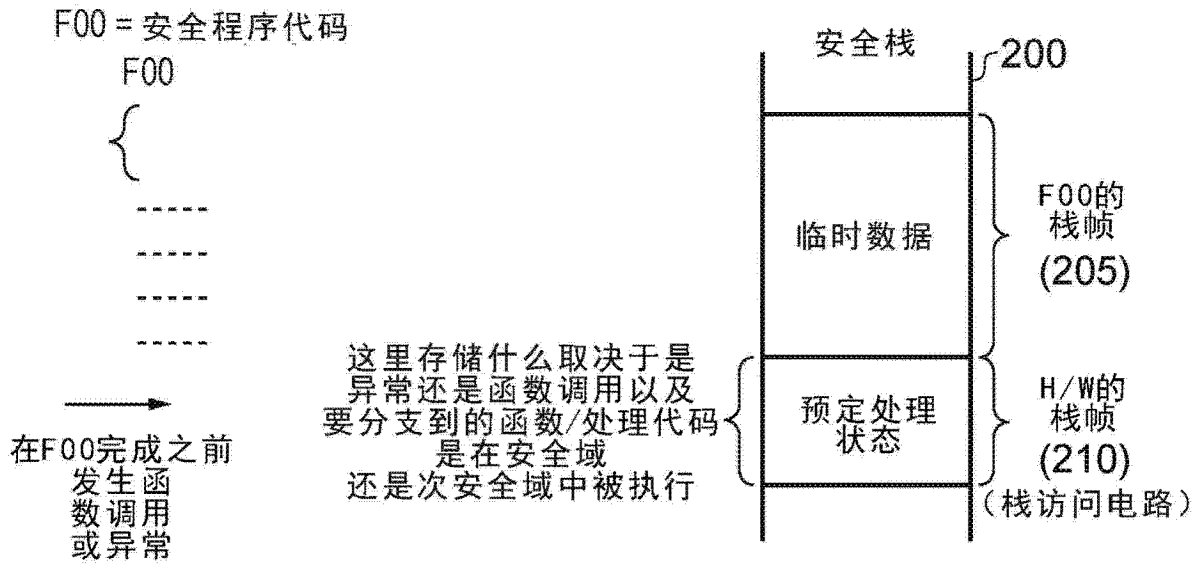

次安全
（公有）
安全
（私有）
175
异常
（例如，打断）
180
185
函数返回
190
195

图 3B


F00 = 安全程序代码
F00
{

在F00完成之前
发生函
数调用
或异常
这里存储什么取决于是
异常还是函数调用以及
要分支到的函数/处理代码
是在安全域
还是次安全域中被执行
安全栈
200
临时数据
预定处理
状态
F00的
栈帧
(205)
H/W的
栈帧
(210)
(栈访问电路)

图 4A

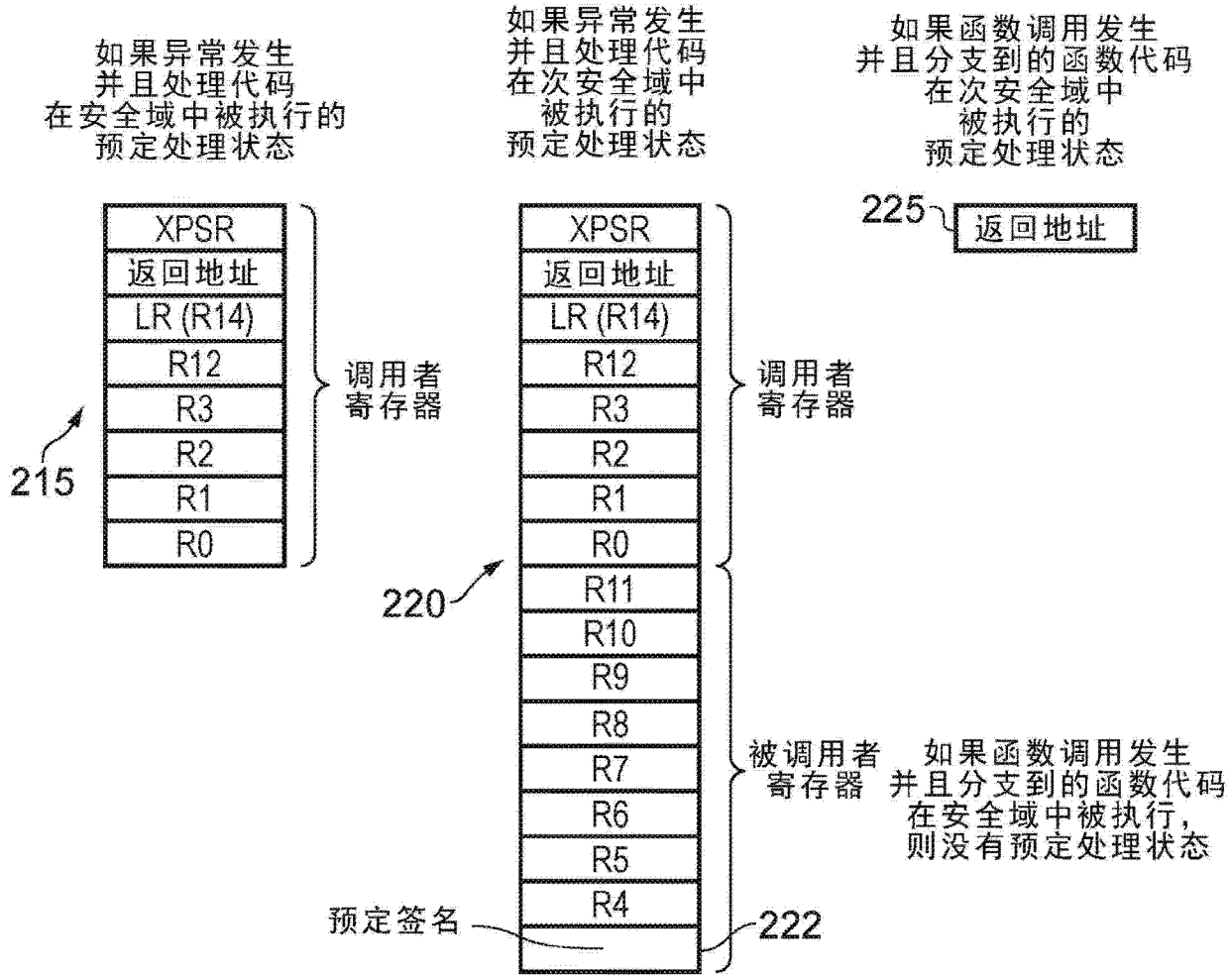

如果异常发生
并且处理代码
在安全域中被执行的
预定处理状态
如果异常发生
并且处理代码
在次安全域中
被执行的
预定处理状态
如果函数调用发生
并且分支到的函数代码
在次安全域中
被执行的
预定处理状态
225
返回地址
XPSR
返回地址
LR (R14)
R12
R3
R2
R1
R0
调用者
寄存器
215
220
R11
R10
R9
R8
R7
R6
R5
R4
被调用者
寄存器
预定签名
222
如果函数调用发生
并且分支到的函数代码
在安全域中被执行,
则没有预定处理状态

图 4B

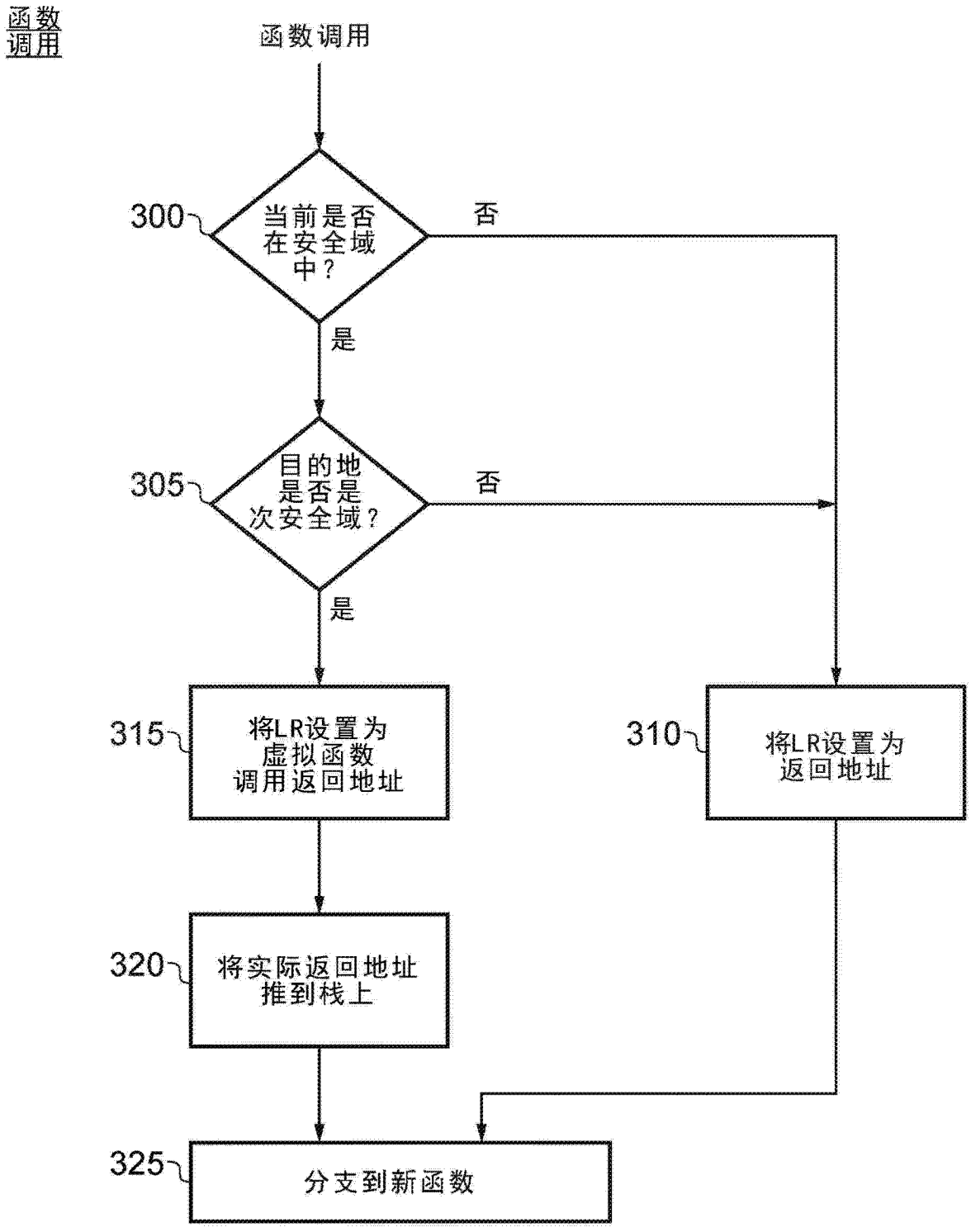

函数调用
函数调用
300
当前是否在安全域中？
否
是
305
目的地是否是次安全域？
否
是
315
将LR设置为虚拟函数调用返回地址
310
将LR设置为返回地址
320
将实际返回地址推到栈上
325
分支到新函数

图 5

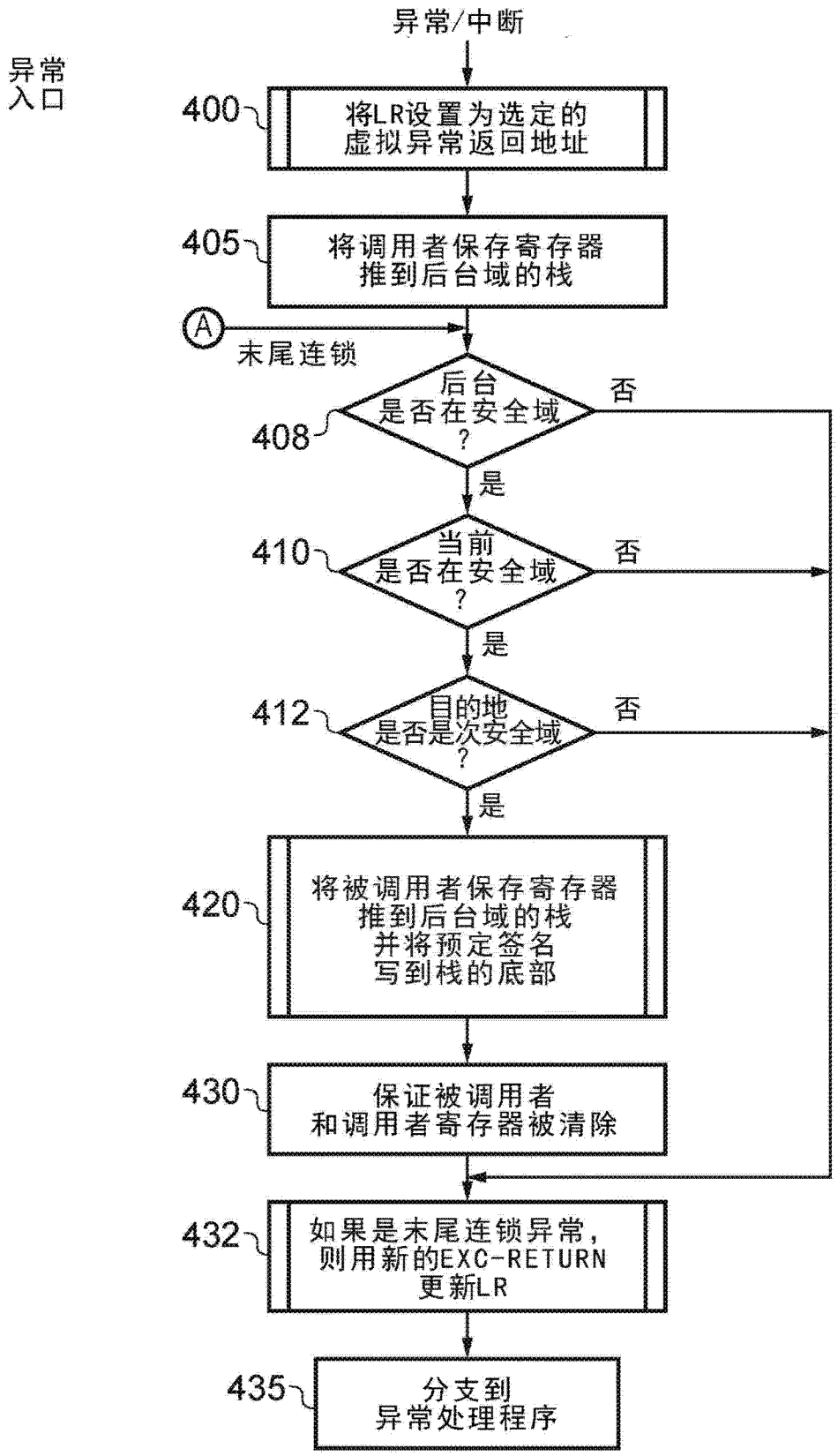

异常/中断
异常入口
400
将LR设置为选定的虚拟异常返回地址
405
将调用者保存寄存器推到后台域的栈
A
末尾连锁
后台是否在安全域？
408
否
是
当前是否在安全域？
410
否
是
目的地是否是次安全域？
412
否
是
420
将被调用者保存寄存器推到后台域的栈并将预定签名写到栈的底部
430
保证被调用者和调用者寄存器被清除
432
如果是末尾连锁异常，则用新的EXC-RETURN更新LR
435
分支到异常处理程序

图 6

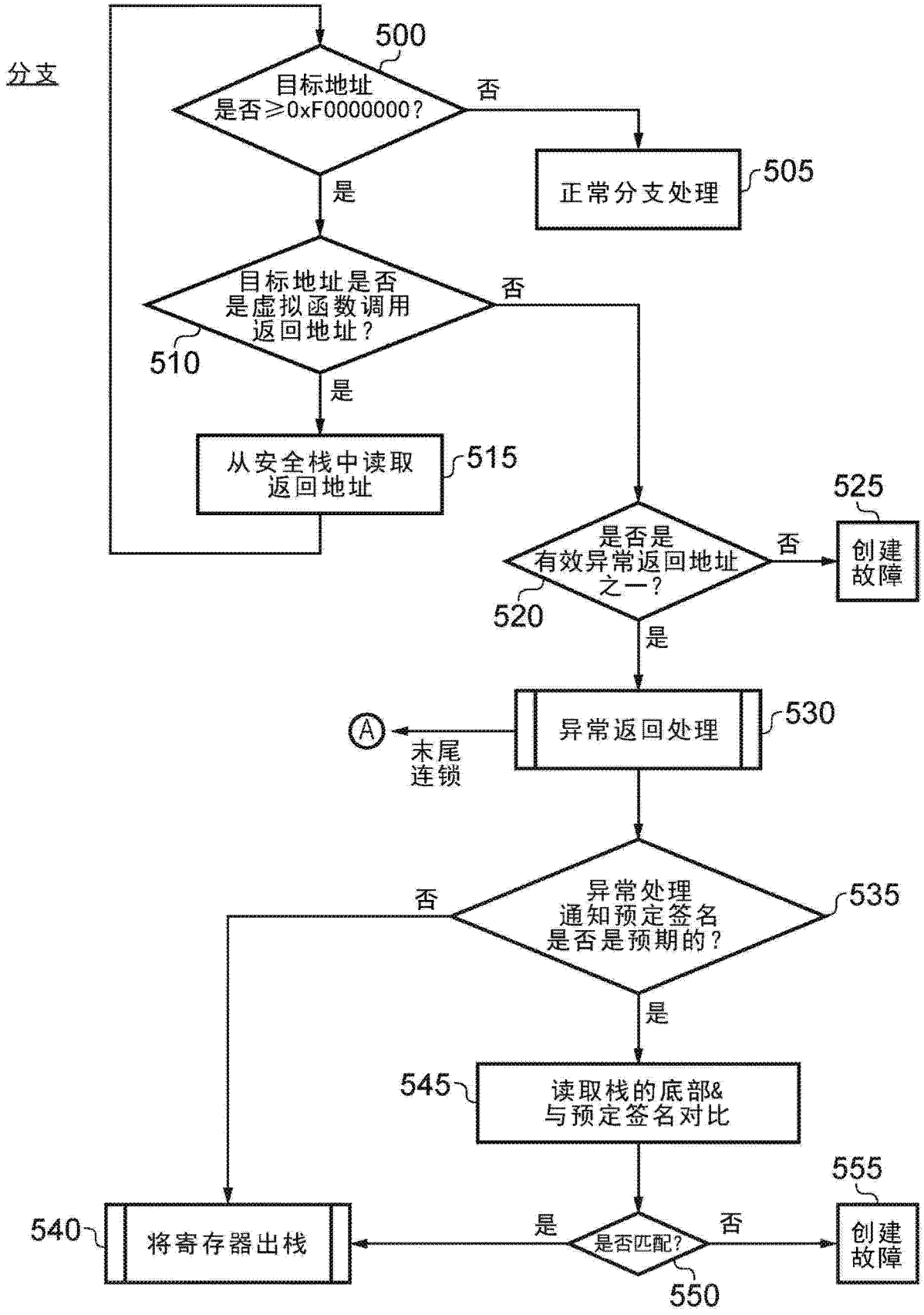

分支
500
目标地址
是否≥0xF0000000?
否
是
505
正常分支处理
目标地址是否
是虚拟函数调用
返回地址?
510
否
是
从安全栈中读取
返回地址
515
525
是否是
有效异常返回地址
之一?
520
否
创建
故障
是
异常返回处理
530
A
末尾
连锁
异常处理
通知预定签名
是否是预期的?
535
否
是
545
读取栈的底部&
与预定签名对比
555
540
将寄存器出栈
是
是否匹配?
否
创建
故障
550

图 7

初始化安全栈

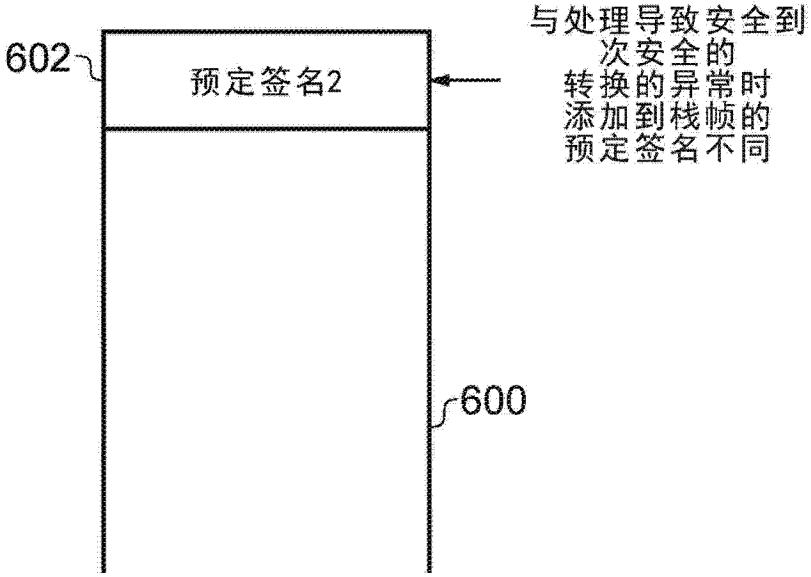


图 8A

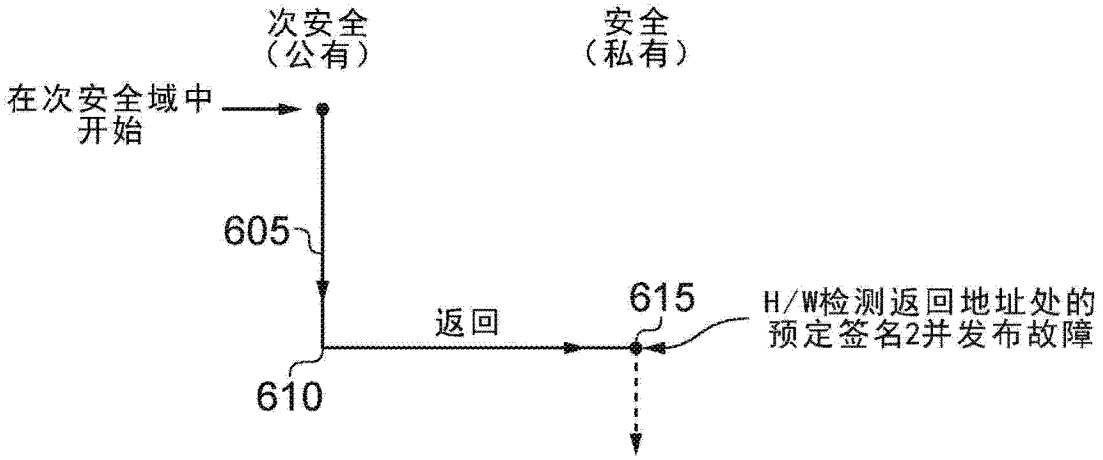


图 8B

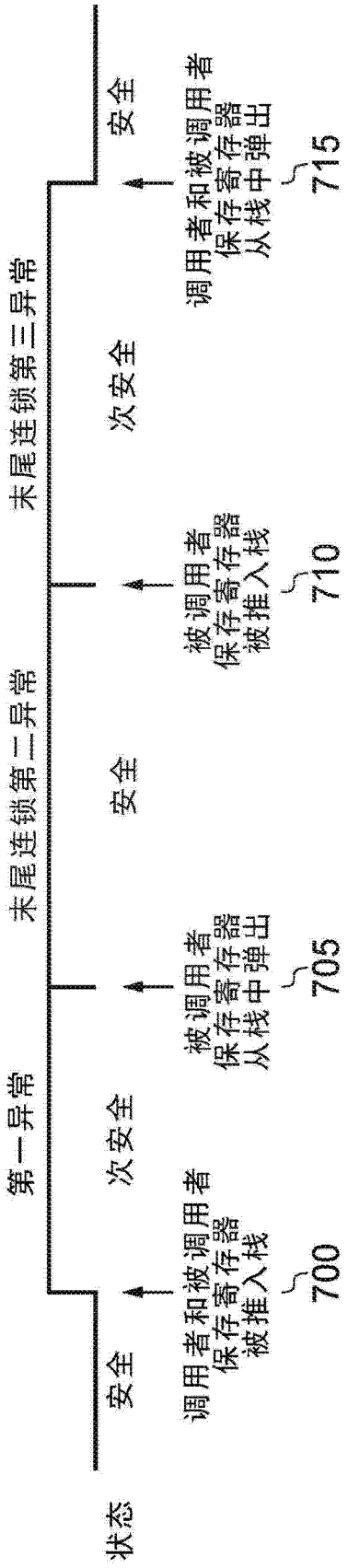

状态
第一异常
未尾连锁第二异常
未尾连锁第三异常
安全
次安全
安全
次安全
安全
调用者和被调用者
保存寄存器
被推入栈
700
被调用者
保存寄存器
从栈中弹出
705
被调用者
保存寄存器
被推入栈
710
调用者和被调用者
保存寄存器
从栈中弹出
715

图 9

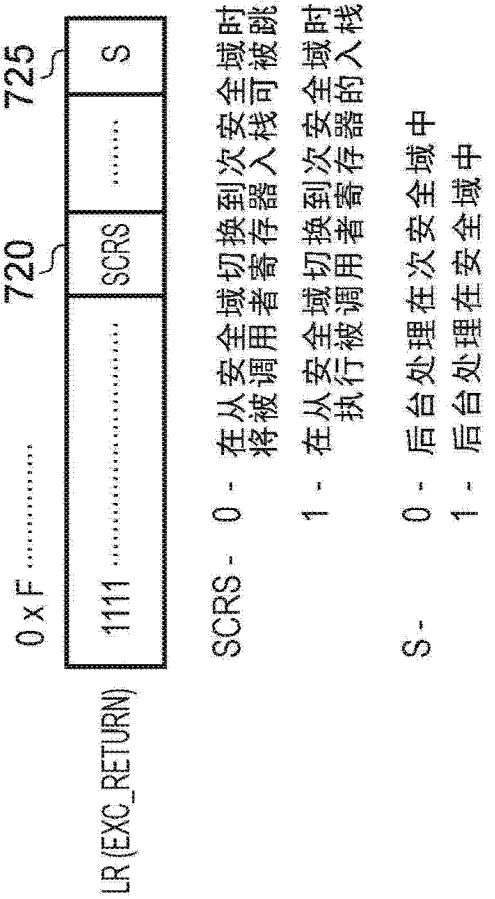

0xF .....
1111 .....
SCRS .....
S
720
725
LR (EXC_RETURN)
SCRS - 0 - 在从安全域切换到次安全域时，将被调用者寄存器入栈可被跳过
1 - 在从安全域切换到次安全域时，执行被调用者寄存器的入栈
S - 0 - 后台处理在次安全域中
1 - 后台处理在安全域中

图 10

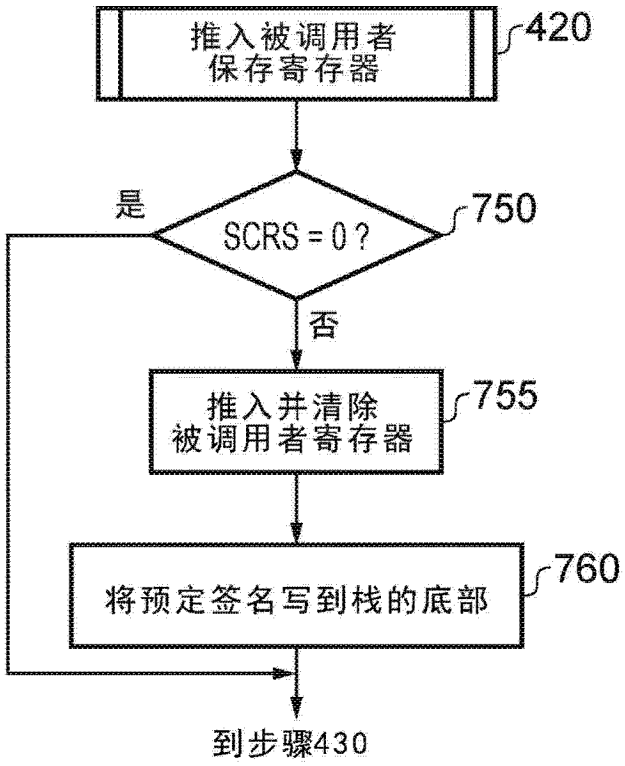

推入被调用者
保存寄存器
420
是
SCRS = 0 ?
750
否
推入并清除
被调用者寄存器
755
将预定签名写到栈的底部
760
到步骤430

图 11

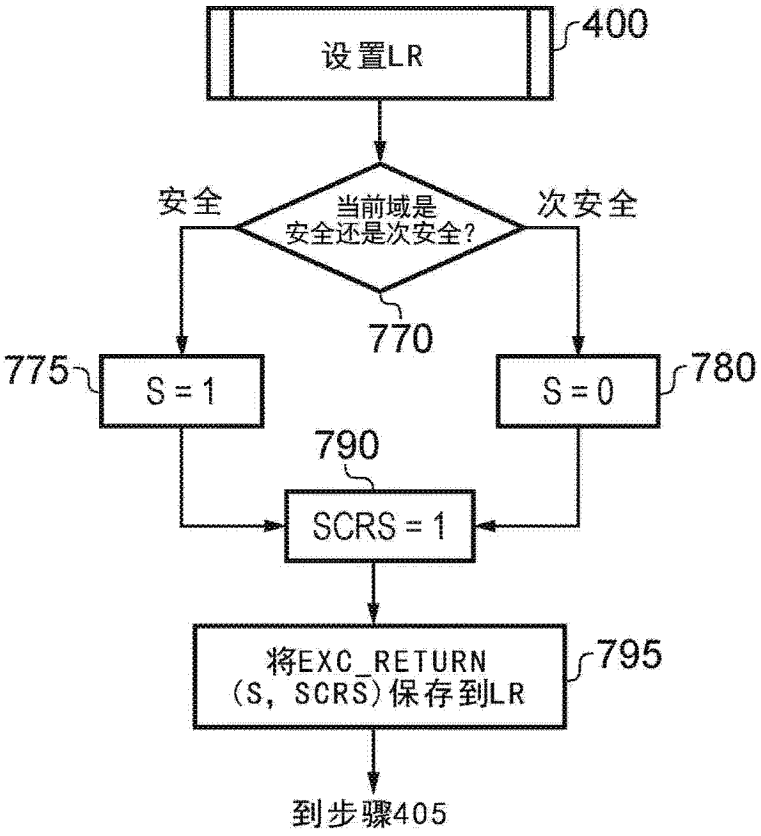

设置LR
400
安全
当前域是
安全还是次安全?
次安全
770
775
S = 1
S = 0
780
790
SCRS = 1
将EXC_RETURN
(S, SCRS)保存到LR
795
到步骤405

图 12A

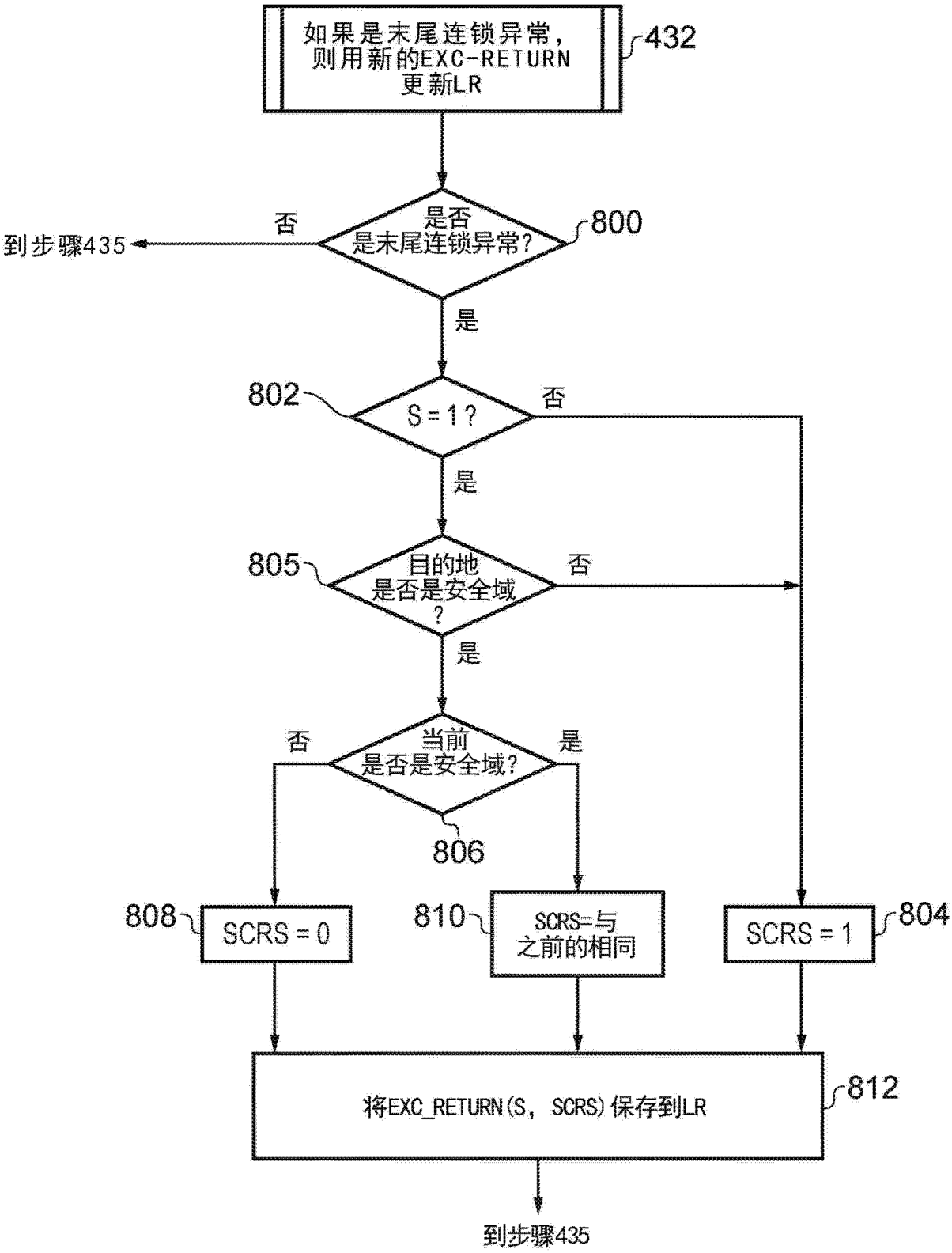

如果是末尾连锁异常，则用新的EXC-RETURN更新LR
432
是否是末尾连锁异常?
800
否
到步骤435
是
802
S = 1?
否
是
805
目的地是否是安全域?
否
是
当前是否是安全域?
否
是
806
808
SCRS = 0
810
SCRS=与之前的相同
SCRS = 1
804
将EXC_RETURN(S, SCRS)保存到LR
812
到步骤435

图 12B

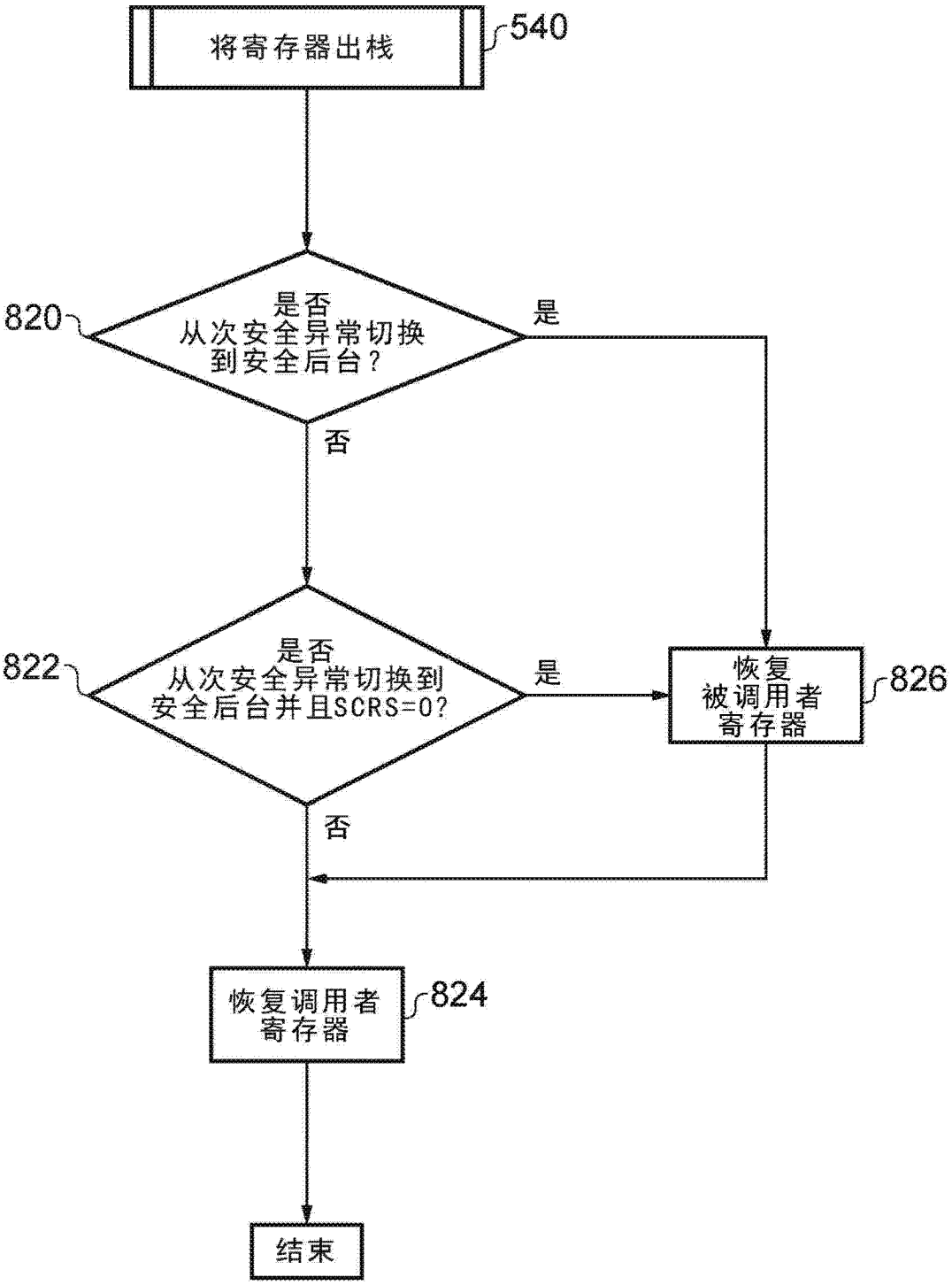

将寄存器出栈
540
是否
从次安全异常切换
到安全后台？
820
是
否
是否
从次安全异常切换到
安全后台并且SCRS=0？
822
是
恢复
被调用者
寄存器
826
否
恢复调用者
寄存器
824
结束

图 13

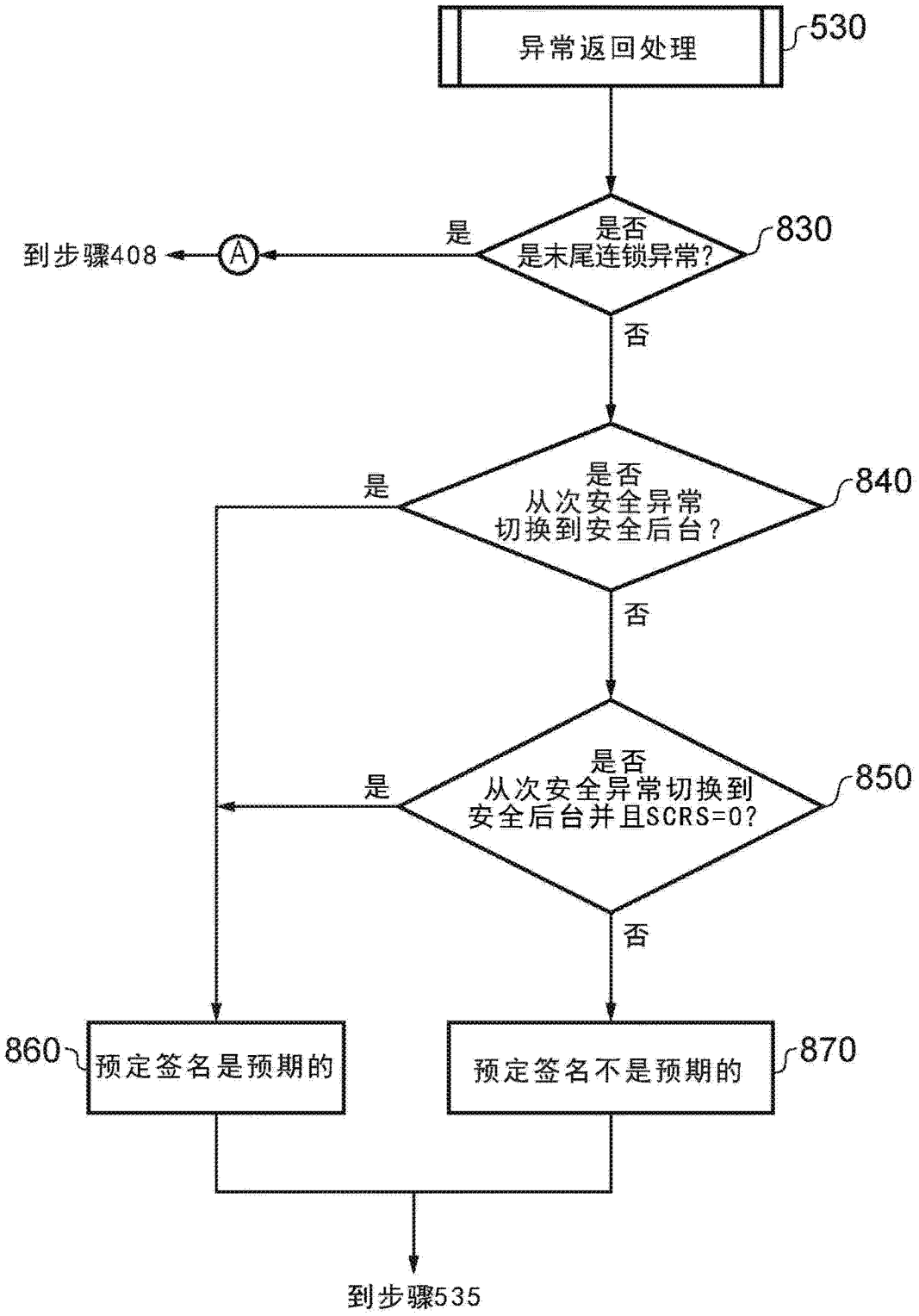

异常返回处理
530
是否
是末尾连锁异常?
830
是
到步骤408
A
否
是否
从次安全异常
切换到安全后台?
840
是
否
是否
从次安全异常切换到
安全后台并且SCRS=0?
850
是
否
860
预定签名是预期的
预定签名不是预期的
870
到步骤535

图 14

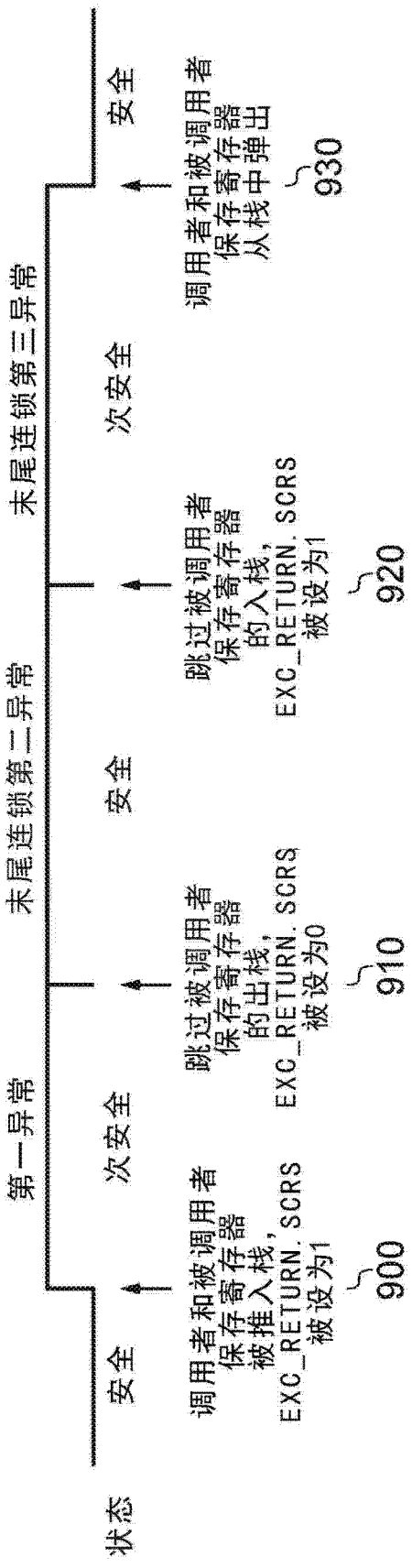

状态
安全
次安全
安全
次安全
安全
第一异常
末尾连锁第二异常
末尾连锁第三异常
调用者和被调用者保存寄存器被推入栈，EXC_RETURN.SCRS被设为1
900
跳过被调用者保存寄存器的出栈，EXC_RETURN.SCRS被设为0
910
跳过被调用者保存寄存器的入栈，EXC_RETURN.SCRS被设为1
920
调用者和被调用者保存寄存器从栈中弹出
930

图 15

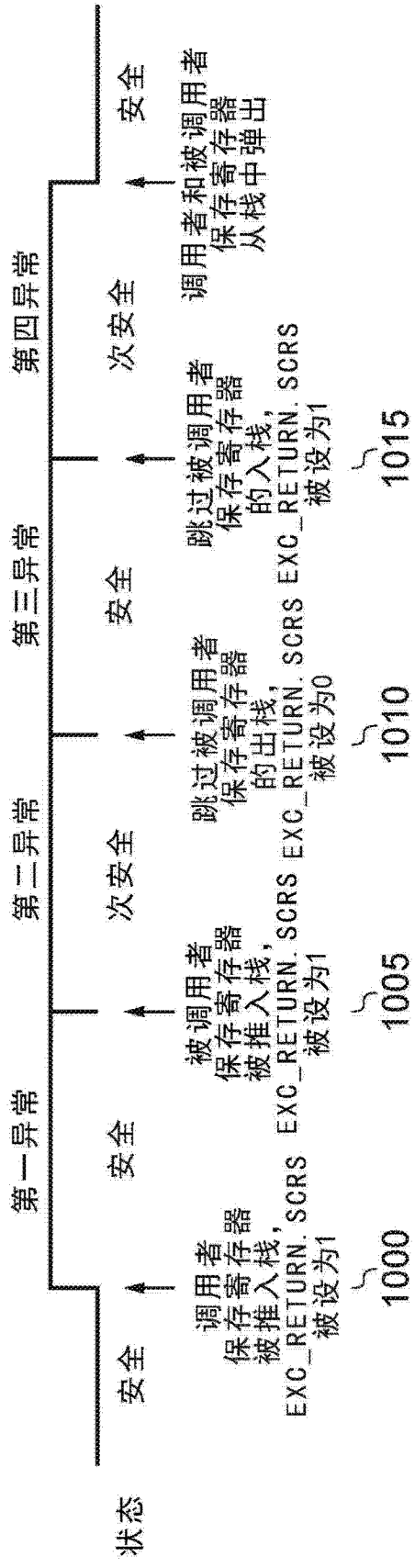

状态
安全
安全
次安全
安全
次安全
安全
第一异常
第二异常
第三异常
第四异常
调用者保存寄存器被推入栈，EXC_RETURN.SCRS被设为1
1000
被调用者保存寄存器被推入栈，EXC_RETURN.SCRS被设为1
1005
跳过被调用者保存寄存器的出栈，EXC_RETURN.SCRS被设为0
1010
跳过被调用者保存寄存器的入栈，EXC_RETURN.SCRS被设为1
1015
调用者和被调用者保存寄存器从栈中弹出

图 16

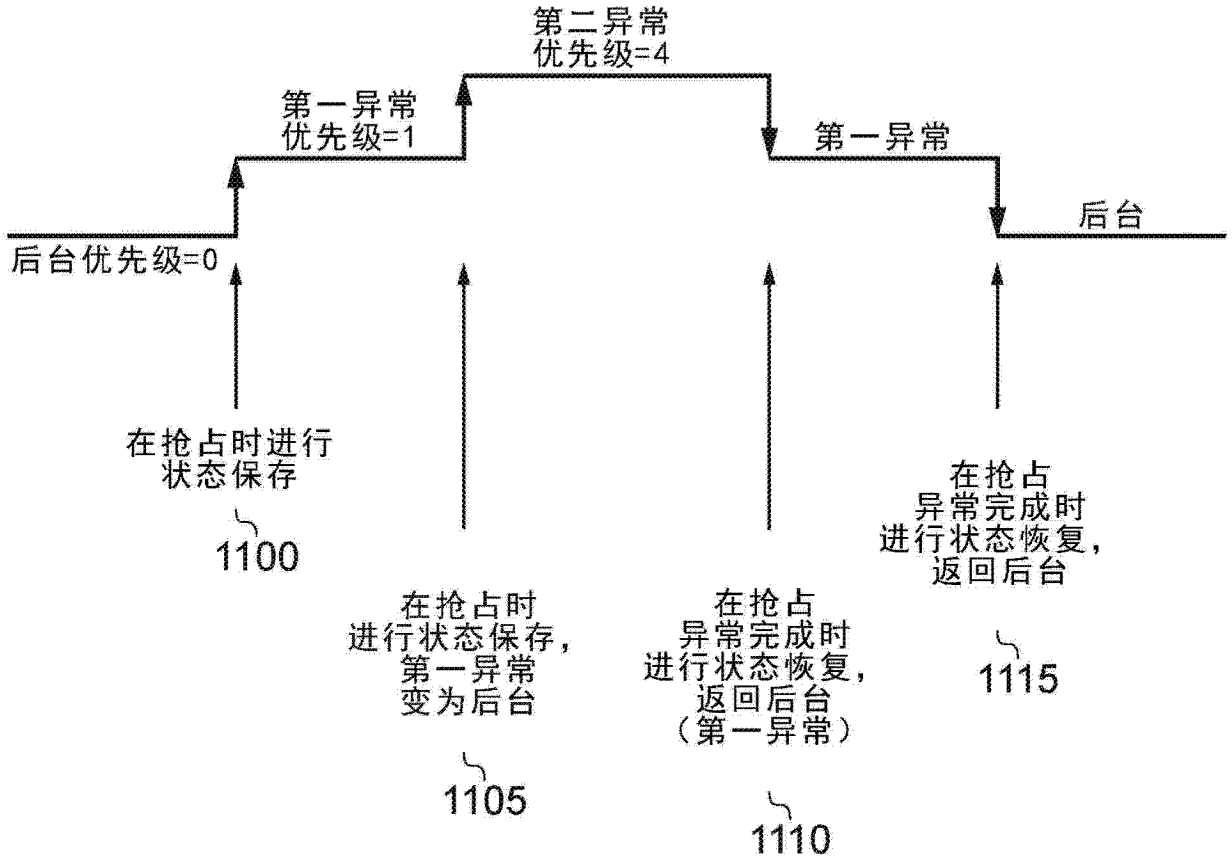

第二异常
优先级=4
第一异常
优先级=1
第一异常
后台
后台优先级=0
在抢占时进行
状态保存
1100
在抢占时
进行状态保存，
第一异常
变为后台
1105
在抢占
异常完成时
进行状态恢复，
返回后台
（第一异常）
1110
在抢占
异常完成时
进行状态恢复，
返回后台
1115

图 17

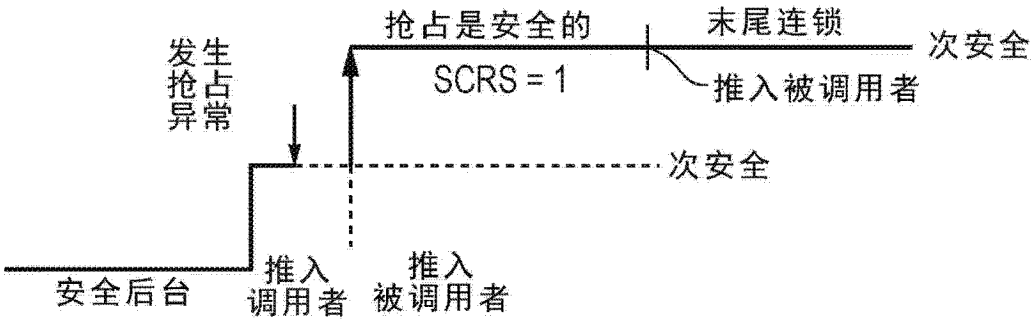

抢占是安全的
末尾连锁
次安全
发生
抢占
异常
SCRS = 1
推入被调用者
次安全
安全后台
推入
调用者
推入
被调用者

图 18A

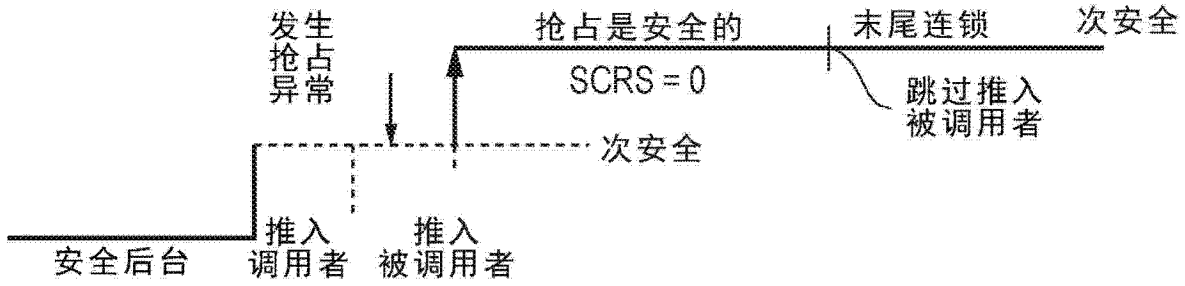

发生
抢占
异常
抢占是安全的
末尾连锁
次安全
SCRS = 0
跳过推入
被调用者
次安全
推入
调用者
推入
被调用者
安全后台

图 18B