(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 111433740 B
(45)授权公告日 2024.07.30

(21)申请号 201880078157.2
(22)申请日 2018.12.14
(65)同一申请的已公布的文献号
申请公布号 CN 111433740 A
(43)申请公布日 2020.07.17
(30)优先权数据
62/613,280 2018.01.03 US
16/208,701 2018.12.04 US
(85)PCT国际申请进入国家阶段日
2020.06.03
(86)PCT国际申请的申请数据
PCT/GB2018/053636 2018.12.14
(87)PCT国际申请的公布数据
WO2019/135063 EN 2019.07.11
(73)专利权人 ARM有限公司
地址 英国剑桥
(72)发明人 理查德·罗伊·格里森思怀特
贾科莫·加布雷利
马修·詹姆斯·霍斯内尔
(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
专利代理师 李丽
(51)Int.Cl.
G06F 9/30(2006.01)
G06F 12/08(2016.01)
G06F 12/14(2006.01)
G06F 9/38(2006.01)
(56)对比文件
US 6484230 B1,2002.11.19
审查员 王莉媛

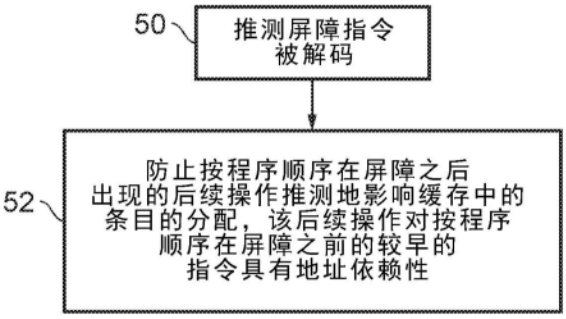
权利要求书3页 说明书20页 附图5页

(54)发明名称

推测屏障指令

(57)摘要

一种装置包括用于执行数据处理的处理电路和用于对指令进行解码来控制处理电路执行数据处理的指令解码电路。该指令解码电路响应于推测屏障指令,而控制处理电路来防止按程序顺序出现在推测屏障指令之后的后续操作推测地影响缓存中的条目的分配,该后续操作对按程序顺序在推测屏障指令之前的较早的指令具有地址依赖性。这提供了针对推测缓存定时边通道攻击的保护。



CN 111433740 B

1.一种数据处理装置,包括:

处理电路,用于执行数据处理;

指令解码电路,用于对指令进行解码,来控制所述处理电路执行所述数据处理;并且

其中:

所述指令解码电路响应于推测屏障指令,而控制所述处理电路来防止按程序顺序出现在所述推测屏障指令之后的后续操作推测地影响缓存中的条目的分配,所述后续操作具有对按所述程序顺序在所述推测屏障指令之前的较早的指令的地址依赖性;以及

在所述推测屏障指令完成之前:

如果加载、存储、数据或指令预加载RW2按程序顺序出现在所述推测屏障指令之后,并且所述加载、存储、数据或指令预加载RW2具有对条件选择指令的结果的地址依赖性,其中:

对于所述条件选择指令的输入寄存器之一,所述条件选择指令具有对已推测地执行的加载R1的寄存器数据依赖性,并且

对于所述条件选择指令的另一输入寄存器,所述条件选择指令不具有对R1的寄存器依赖性,并且

所述条件选择指令的条件是使得如果未按架构执行R1则选择不依赖于R1的输入,

那么,所述处理电路被配置为防止RW2以如下方式影响所述缓存中的条目的分配,该方式能够用于通过评估所述缓存中的哪些条目已被分配或逐出,来确定来自R1的推测地加载的值的值的任何部分。

2.一种数据处理装置,包括:

处理电路,用于执行数据处理;

指令解码电路,用于对指令进行解码,来控制所述处理电路执行所述数据处理;并且

其中:

所述指令解码电路响应于推测屏障指令,而控制所述处理电路来防止按程序顺序出现在所述推测屏障指令之后的后续操作推测地影响缓存中的条目的分配,所述后续操作具有对按所述程序顺序在所述推测屏障指令之前的较早的指令的地址依赖性;以及

在所述推测屏障指令完成之前:

如果加载、存储、数据或指令预加载RW2按程序顺序出现在所述推测屏障指令之后,并且所述加载、存储、数据或指令预加载RW2具有对条件移动指令的结果的地址依赖性,其中:

对于所述条件移动指令的输入寄存器,所述条件移动指令不具有对已推测地执行的加载R1的寄存器依赖性,并且

所述条件移动指令的条件是使得如果未按架构执行R1则该条件通过,

那么,所述处理电路被配置为防止RW2以如下方式影响所述缓存中的条目的分配,该方式能够用于通过评估所述缓存中哪些条目已被分配或逐出,来确定来自R1的推测地加载的值的值的任何部分。

3.如权利要求1或2所述的装置,其中,所述缓存是以下中的一项:

数据缓存;

指令缓存;以及

分支预测缓存。

4.一种计算机可读存储介质,其上存储有计算机程序,所述计算机程序当被执行时控

制主机处理装置来提供用于执行待执行的目标程序代码的指令的指令执行环境,所述计算机程序包括:

指令解码程序逻辑,用于对所述目标程序代码的指令进行解码,来控制处理程序逻辑执行数据处理;其中:

所述指令解码程序逻辑响应于推测屏障指令,而防止按程序顺序出现在所述推测屏障指令之后的后续操作推测地影响缓存中的条目的分配,所述后续操作具有对按所述程序顺序在所述推测屏障指令之前的较早的指令的地址依赖性;以及

其中,在所述推测屏障指令完成之前:

如果加载、存储、数据或指令预加载RW2按程序顺序出现在所述推测屏障指令之后,并且所述加载、存储、数据或指令预加载RW2具有对条件选择指令的结果的地址依赖性,其中:

对于所述条件选择指令的输入寄存器之一,所述条件选择指令具有对已推测地执行的加载R1的寄存器数据依赖性,并且

对于所述条件选择指令的另一输入寄存器,所述条件选择指令不具有对R1的寄存器依赖性,并且

所述条件选择指令的条件是使得如果未按架构执行R1则选择不依赖于R1的输入,

那么,所述处理程序逻辑被配置为防止RW2以如下方式影响所述缓存中的条目的分配,该方式能够用于通过评估所述缓存中的哪些条目已被分配或逐出,来确定来自R1的推测地加载的值的值的任何部分。

5.一种计算机可读存储介质,其上存储有计算机程序,所述计算机程序当被执行时控制主机处理装置来提供用于执行待执行的目标程序代码的指令的指令执行环境,所述计算机程序包括:

指令解码程序逻辑,用于对所述目标程序代码的指令进行解码,来控制处理程序逻辑执行数据处理;其中:

所述指令解码程序逻辑响应于推测屏障指令,而防止按程序顺序出现在所述推测屏障指令之后的后续操作推测地影响缓存中的条目的分配,所述后续操作具有对按所述程序顺序在所述推测屏障指令之前的较早的指令的地址依赖性;以及

在所述推测屏障指令完成之前:

如果加载、存储、数据或指令预加载RW2按程序顺序出现在所述推测屏障指令之后,并且所述加载、存储、数据或指令预加载RW2具有对条件移动指令的结果的地址依赖性,其中:

对于所述条件移动指令的输入寄存器,所述条件移动指令不具有对已推测地执行的加载R1的寄存器依赖性,并且

所述条件移动指令的条件是使得如果未按架构执行R1则该条件通过,

那么,所述处理程序逻辑被配置为防止RW2以如下方式影响所述缓存中的条目的分配,该方式能够用于通过评估所述缓存中哪些条目已被分配或逐出,来确定来自R1的推测地加载的值的值的任何部分。

6.一种数据处理方法,包括:

对推测屏障指令进行解码;并且

响应于对所述推测屏障指令的解码,而控制处理电路来防止按程序顺序出现在所述推测屏障指令之后的后续操作推测地影响缓存中的条目的分配,所述后续操作具有对按所述

程序顺序在所述推测屏障指令之前的较早的指令的地址依赖性;以及

在所述推测屏障指令完成之前:

如果加载、存储、数据或指令预加载RW2按程序顺序出现在所述推测屏障指令之后,并且所述加载、存储、数据或指令预加载RW2具有对条件选择指令的结果的地址依赖性,其中:

对于所述条件选择指令的输入寄存器之一,所述条件选择指令具有对已推测地执行的加载R1的寄存器数据依赖性,并且

对于所述条件选择指令的另一输入寄存器,所述条件选择指令不具有对R1的寄存器依赖性,并且

所述条件选择指令的条件是使得如果未按架构执行R1则选择不依赖于R1的输入,

那么,所述方法包括:防止RW2以如下方式影响所述缓存中的条目的分配,该方式能够用于通过评估所述缓存中的哪些条目已被分配或逐出,来确定来自R1的推测地加载的值的值的任何部分。

7.一种数据处理方法,包括:

对推测屏障指令进行解码;并且

响应于对所述推测屏障指令的解码,而控制处理电路来防止按程序顺序出现在所述推测屏障指令之后的后续操作推测地影响缓存中的条目的分配,所述后续操作具有对按所述程序顺序在所述推测屏障指令之前的较早的指令的地址依赖性;以及

在所述推测屏障指令完成之前:

如果加载、存储、数据或指令预加载RW2按程序顺序出现在所述推测屏障指令之后,并且所述加载、存储、数据或指令预加载RW2具有对条件移动指令的结果的地址依赖性,其中:

对于所述条件移动指令的输入寄存器,所述条件移动指令不具有对已推测地执行的加载R1的寄存器依赖性,并且

所述条件移动指令的条件是使得如果未按架构执行R1则该条件通过,

那么,所述方法包括:防止RW2以如下方式影响所述缓存中的条目的分配,该方式能够用于通过评估所述缓存中哪些条目已被分配或逐出,来确定来自R1的推测地加载的值的值的任何部分。

推测屏障指令

技术领域

[0001] 本技术涉及数据处理的领域。

背景技术

[0002] 数据处理装置可以支持指令的推测执行,其中在知道指令的输入操作数是否正确或者是否完全需要执行指令之前执行指令。例如,处理装置可以具有用于预测分支指令的结果的分支预测器,使得可以在知道分支的实际结果应当是什么之前推测地获取、解码和执行后续指令。而且,一些系统可以支持加载推测,其中在实际上从存储器返回真实值之前预测从存储器加载的值,以允许更快地处理后续指令。其他形式的推测也是可能的。

发明内容

[0003] 至少一些示例提供了一种装置,包括:用于执行数据处理的处理电路;指令解码电路,用于对指令进行解码,以控制处理电路执行数据处理;并且其中:指令解码电路响应于推测屏障指令而控制处理电路,以防止按程序顺序在推测屏障指令之后出现的后续操作推测地影响缓存中的条目的分配,该后续操作对按程序顺序在推测屏障指令之前的较早的指令具有地址依赖性。

[0004] 至少一些示例提供了一种计算机程序,用于控制主机处理装置以提供用于执行目标程序代码的指令的指令执行环境,该计算机程序包括:指令解码程序逻辑,用于对目标程序代码的指令进行解码,以控制处理程序逻辑执行数据处理;其中:指令解码程序逻辑响应于推测屏障指令,而防止按程序顺序在推测屏障指令之后出现的后续操作推测地影响缓存中的条目的分配,该后续操作对按程序顺序在推测屏障指令之前的较早的指令具有地址依赖性。

[0005] 至少一些示例提供了一种数据处理方法,包括:对推测屏障指令进行解码;以及响应于对推测屏障指令的解码,而控制处理电路以防止按程序顺序在推测屏障指令之后出现的后续操作推测地影响缓存中的条目的分配,该后续操作对按程序顺序在推测屏障指令之前的较早的指令具有地址依赖性。

[0006] 至少一些示例提供了一种装置,包括:用于执行数据处理的处理电路;指令解码电路,用于对指令进行解码以控制处理电路执行数据处理,其中响应于条件分支指令,指令解码电路被配置为控制处理电路来选择第一结果和第二结果中的一个,在第一结果中条件分支指令之后的下一指令是第一指令,而在第二结果中所述下一指令是第二指令;以及分支预测电路,用于预测应当为条件分支指令选择第一结果还是第二结果;其中:响应于条件分支指令的单向推测变体,分支预测电路和指令解码电路中的至少一个被配置为控制处理电路:当分支预测电路为条件分支指令的单向推测变体预测第一结果时,针对第一指令施加对推测执行的限制;并且当分支预测电路为条件分支指令的单向推测变体预测第二结果时,省略针对第二指令施加对推测执行的限制。

[0007] 至少一些示例提供了一种数据处理方法,包括:响应于条件分支指令,选择第一结

果和第二结果中的一个,在第一结果中条件分支指令之后的下一指令在是第一指令,而在第二结果中所述下一指令是第二指令;预测应当为条件分支指令选择第一结果还是第二结果;并且当条件分支指令是条件分支指令的单向推测变体时:当为条件分支指令的单向推测变体预测第一结果时,向第一指令施加对推测执行的限制;并且当为条件分支指令的单向推测变体预测第二结果时,省略向第二指令施加对推测执行的限制。

附图说明

[0008] 本技术的其他方面、特征和优点将通过以下结合附图阅读的示例的描述而变得清晰可见。

[0009] 图1示意性地示出了支持推测执行指令的数据处理装置的示例;

[0010] 图2示意性地示出了根据基于推测的基于缓存的定时边通道的潜在攻击的示例;

[0011] 图3示出了处理推测屏障指令的方法;

[0012] 图4示出处理存储推测屏障指令的方法;

[0013] 图5示出了处理条件分支指令的单向推测变体的方法;以及

[0014] 图6示出了可以使用的模拟器示例。

具体实施方式

[0015] 数据处理装置可以具有用于确保存储器中的一些数据不能被在处理电路上执行的某些处理访问的机制。例如,基于特权的机制和/或存储器保护属性可以用于控制对存储器的某些区域的访问。近来,已经认识到,在使用推测执行和缓存的系统中,恶意者有可能通过利用如下性质来获得来自他们无权访问的存储器的区域中的信息:即使在推测执行的指令的任何架构影响已经在错误推测之后被逆转之后,推测执行的指令的影响也可能存留在数据缓存中。这样的攻击可能会训练分支预测器或其他推测机制,以欺骗更多特权代码推测地执行指令的序列,这些指令被设计为使得特权代码访问依赖于敏感信息的存储器地址的模式,以使得无法访问敏感信息的特权较低的代码能够使用缓存定时边通道来探查特权较高的代码已将哪些地址分配给缓存或从缓存中清除,来给出一些信息,这可以允许推断出敏感信息。这样的攻击可以称为推测边通道攻击。

[0016] 如下面更详细地讨论的,数据处理装置的指令解码电路可以支持推测屏障指令。响应于推测屏障指令,指令解码电路可以控制处理电路来防止后续操作推测地影响缓存中的条目的分配,该后续操作按程序顺序出现在推测屏障指令之后并且对按程序顺序在推测屏障指令之前的较早的指令具有地址依赖性。这提供了相对于上述攻击类型的保护。例如,生成被允许访问敏感信息的代码的编译器的程序员可以在较早的加载指令和稍后的加载指令之间包括推测屏障指令,该较早的加载指令的地址是从由不受信的代码提供的值得出的,并且稍后的加载指令的地址是基于由较早的加载指令的加载的值计算得出的。包括屏障可以防止稍后的加载指令以如下方式推测地影响缓存中的条目的分配,使得缓存定时通道可用于确定有关不受信的代码不可访问的敏感信息的信息。因此,通过提供对推测屏障指令的支持,可以提高安全性。

[0017] 在推测屏障指令之后的后续操作可以是加载指令或存储指令之一,针对该后续操作防止缓存中的条目的分配的推测影响。较早的指令(该加载或存储指令对其具有地址依

赖性)可以是加载指令。

[0018] 在一个示例中,直到推测屏障指令完成:

[0019] • 如果对条件选择指令的结果具有地址依赖性的加载、存储、数据或指令预加载RW2按程序顺序出现在推测屏障指令之后,其中:

[0020] o对于条件选择指令的输入寄存器之一,条件选择指令具有对已推测地执行的加载R1的寄存器数据依赖性,并且

[0021] o对于条件选择指令的另一输入寄存器,条件选择指令不具有对R1的寄存器依赖性,并且

[0022] o条件选择指令的条件是使得如果未按架构执行R1则选择不依赖于R1的输入,

[0023] 那么处理电路被配置为防止RW2以如下方式影响缓存中的条目的分配,该方式可用于通过评估缓存中的哪些条目已被分配或逐出,来确定来自R1的推测地加载的值的值的任何部分。这样可以防止攻击者能够通过分析后续操作RW2对缓存的影响来得出有关由加载R1推测地加载的值的任何信息。

[0024] 在另一示例中,直到推测屏障指令完成:

[0025] • 如果对条件移动指令的结果具有地址依赖性的加载、存储、数据或指令预加载RW2按程序顺序出现在推测屏障指令之后时,其中:

[0026] o对于条件移动指令的输入寄存器,条件移动指令不具有对已经推测地执行的加载R1的寄存器依赖性,并且

[0027] o条件移动指令的条件是使得如果未按架构执行R1则该条件通过,

[0028] 那么处理电路被配置为防止RW2以如下方式影响缓存中的条目的分配,该方式可用于通过评估缓存中哪些条目已被分配或逐出,来确定来自R1的推测地加载的值的值的任何部分。同样,这防止了攻击者基于对后续操作RW2对缓存分配的影响的分析来利用对加载R1的错误推测来获取有关由R1加载的值的信息。

[0029] 缓存可以是数据缓存、和指令缓存、以及分支预测缓存中的任何一者,针对该缓存防止推测屏障指令之后的后续操作对条目的分配的推测影响。缓存的其他示例可以包括:值预测器缓存、加载/存储别名(aliasing)预测器缓存、以及其他预测结构。

[0030] 处理电路可以以不同的方式防止推测屏障指令之后的后续操作推测地影响缓存中的条目的分配。在一个示例中,这可以通过至少在推测屏障指令完成之前防止后续操作的推测执行来防止。一旦已经解析了程序顺序中的任何在前指令,就可以认为推测屏障指令已完成。在一些示例中,一旦推测屏障指令已经完成,就可以允许后续操作的推测执行(假设推测基于与遵循推测屏障指令的操作相关联的预测)。

[0031] 在其他示例中,处理电路可以防止后续操作推测地影响缓存中的条目的分配,同时允许对后续操作的推测执行。对缓存中的条目分配的影响可以是为先前不具有分配了该缓存的条目的给定地址分配新条目,或者逐出与给定地址相关联的先前缓存的条目使得该给定地址不再被缓存。

[0032] 例如,不是防止推测执行本身,而是可以防止推测执行对缓存的影响。例如,如果在推测屏障指令之后的推测地执行的操作是基于推测屏障指令之前的指令的结果的推测预测来执行的并且推测屏障指令尚未完成,则一些实现方式可以选择完全防止该推测地执行的操作缓存数据。

[0033] 可替代地,在推测屏障指令完成以前,由后续操作推测加载的任何数据都可被保留在推测缓冲器中并且不被分配到主缓存中,直到推测已经解析并且推测屏障指令已经完成为止。例如,推测缓冲器可以是可以作为与主缓存的任何缓存方式共存的附加缓存方式来进行查找的缓冲器,但是推测缓冲器的内容可以在特权级别的某些转变时(例如,在向特权较低的状态转变时)被丢弃,以使得特权较低的代码不能在尚未正确地解析推测地加载的数据时对其进行访问。这种途径可以防止主缓存区域被推测地加载的值污染(并防止由于那些推测地加载的值的分配而导致主缓存中的其他值被逐出),因此如果随后有到特权较低的状态的转变,则特权较低的代码将无法使用缓存定时边通道来推断有关缓存的值的任何信息。

[0034] 在一个示例中,响应于推测屏障指令,指令解码电路系统可以控制处理电路系统,来使用以下各项中的至少一项来防止后续指令被推测地执行:用于较早的指令的数据值预测,当较早的指令是除条件分支指令之外的指令时针对较早的指令的条件代码预测,以及在较早的指令是取决于指示向量值的哪些数据元素是要处理的活动元素的谓词(predicate)信息的向量指令的情况下,对该谓词信息的预测。通过防止后续指令能够推测地使用针对推测屏障指令之前的指令所做的数据值预测、条件代码预测或谓词预测的结果,可以减少攻击者可利用的途径。

[0035] 在一个示例中,处理电路可以准许对跟在推测屏障指令之后的控制指令流的推测控制,基于与推测屏障指令之前的在架构上尚未解析的较早的指令相关联的预测。因此,仍然可以准许在推测屏障指令之前和之后对流推测进行控制。准许推测控制流是可能的,因为抑制响应于跟在屏障之后的推测地执行的指令而导致的对缓存的影响可能足以缓解上述攻击。通过允许对屏障前后的控制流进行推测控制,可以提高性能。类似地,在一些示例中,可以准许对跟在推测屏障指令之后的条件指令的推测执行,除非它们在按程序顺序在时间上在推测屏障指令之前的较早的指令尚未在架构上得到解析时,使用对该较早的指令的数据值预测或条件代码预测的结果。

[0036] 在一些数据处理装置中,另一推测形式可以是对加载或存储指令的地址进行推测。加载或存储指令的地址可以取决于基于来自寄存器的值的地址计算,可以通过较早的指令来计算该来自寄存器的值。当存储指令后面跟着加载指令,而该加载指令的地址还未知时,则可能存在加载指令的地址实际上可能与较早的存储指令的地址相同的风险,尽管与加载的地址与较早的存储的地址不同的情况相比可能这种可能性较小。由于等待存储值所依赖的较早的指令的结果,可能会延迟存储指令的执行。直到执行加载指令,才可能知道加载指令是否确实访问与存储指令相同的地址。如果处理电路在发出要执行的加载指令之前等待存储指令完成,则在加载地址实际上与存储地址不同的情况下,这可能会导致不必要的延迟。

[0037] 因此,通过推测地提早发出加载指令(绕过较早的存储),然后在实际上发现加载指令的地址与存储指令的地址不同的情况下,由于加载不会不必要地延迟等待来自较早的存储指令的值,因此提高了性能。但是,在实际上发现加载地址与较早的存储地址相同的情况下,这种推测执行可能导致加载错误的值,因为加载指令可能加载在存储指令被执行前存储在目标地址中的过期值,而不是由存储指令的执行产生的值。

[0038] 在一些情形下,攻击者可能尝试利用加载指令对较早的存储指令的这种推测绕过

来试图获得对敏感信息的访问。可以提供存储推测屏障指令以减轻这种攻击。响应于存储推测屏障指令,指令解码电路可以防止按程序顺序在该存储推测屏障指令之前的较早的存储指令通过对按程序顺序在该存储推测屏障指令之后并且指定与该较早的存储指令相同的地址的后续加载指令的推测执行而被绕过。存储推测屏障指令可以具有与上述推测屏障指令不同的指令编码。因此,对存储推测屏障指令的支持提供了更高的安全性。

[0039] 可以提供存储推测屏障指令的不同变体。对于存储推测屏障指令的第一变体,后续加载指令可以是指定与较早的存储指令相同的虚拟地址的指令。对于第二变体,后续加载指令可以是指定与较早的存储指令相同的物理地址的指令。提供第二变体例如对于防止在进入或离开操作系统时使用缓存定时边通道机制来利用推测加载是有用的。

[0040] 响应于存储推测屏障指令,指令解码电路还可以防止按程序顺序在该存储推测屏障指令之前的较早的加载指令推测地加载从按程序顺序在该存储推测屏障指令之后并且指定与该较早的加载指令相同的地址的后续存储指令得出的数据。这防止如下攻击,这些攻击可能尝试利用相反的场景,即在假定存储将具有不同的地址的情况下,跟在推测加载指令之后的存储在较早的加载之前被执行,但后来发现实际上存储到与较早的加载相同的地址,在这种情况下,对存储的推测执行可能会导致较早的加载加载了错误的值。存储推测屏障指令可以避免攻击者能够使用缓存定时边通道来探测由这种不正确的推测引起的缓存分配的变化,从而提高了安全性。

[0041] 可能引起推测执行的一种类型的指令可以是条件分支指令,响应于该条件分支指令,指令解码器可以控制处理电路来选择第一结果或第二结果,在第一结果中第一指令被选择为条件分支之后的下一指令,而在第二结果中第二指令被选择为条件分支之后的下一指令。例如,第一和第二结果中的一个可以针对条件分支指令的被采用的结果,而另一个可以是未采用的结果。可以提供分支预测电路,以在知道实际结果之前预测应为条件分支指令选择第一结果还是第二结果。然后可以基于分支预测来推测地执行后续指令。

[0042] 在一个示例中,可以提供条件分支指令的单向推测变体,响应于该变体,分支预测电路和指令解码电路中的至少一个可以控制处理电路来在分支预测电路预测了条件分支指令的单向推测变体的第一结果时对第一指令施加推测执行的限制,而在分支预测电路预测了条件分支指令的单向推测变体的第二结果的情况下,可以省略对第二条指令施加推测执行的限制。

[0043] 因此,当分支预测器预测条件分支的一个结果时,与它预测条件分支的另一结果时相比,推测执行可能更受限。这对于防止上述类型的基于推测的缓存定时边通道攻击可能是有用的。例如,如果条件分支检查从潜在不受信的代码传递给它的值是否在允许范围内,或检查由潜在不受信的代码提供的密码是否正确,则可以设计被采用和未采用结果之一,以继续处理潜在的敏感信息,而另一结果可能没有访问该敏感信息的风险。因此,通常程序员或编译器可能会知道条件分支中的哪个结果比另一结构更有受到攻击的风险。通过提供条件分支指令的单向推测变体,在此之后,对一个预测的分支结果的推测执行比对另一预测的分支结果的推测执行受到的限制更少,这可以允许处理器在预测第二结果时通过允许推测执行来提高性能,而在预测第一结果时通过限制推测执行来增大安全性。

[0044] 在为预测条件分支预测第一结果的情况下,可能有多种方式来限制推测执行。在一个示例中,可以完全防止对跟在条件分支之后的给定指令的推测执行(至少直到预测被

解析为正确的为止)。可替代地,对推测的限制仍然可以允许后续指令被推测地执行,但是可能涉及施加于跟在分支之后的后续指令的缓存限制(同样,至少直到预测被解析为正确的为止)。例如,可以响应于在针对条件分支的第一结果的预测之后推测地执行的指令而禁止缓存,或者可以启用缓存,但是将其缓存到与主缓存分离的单独的推测缓冲器中,如上述示例中所示。

[0045] 一般而言,可以多种方式将条件分支指令的单向推测变体与条件分支指令的传统双向推测变体(对于该双向推测变体,无论分支预测器预测第一结果还是第二结果,推测执行都不受限制)区分开来。在一个示例中,单向推测变体可以具有不同的指令编码(例如,不同的指令操作码,或者该指令编码内的区分不同变体的字段)。可替代地,条件分支指令的单向和双向推测变体可以具有相同的编码,但是存储在数据处理装置的控制寄存器中的控制参数可以指定当遇到这种条件分支指令时将使用哪个变体。

[0046] 也可能有单向推测变体本身的许多替代变体。例如,对于第一变体,第一结果可以包括被采用的结果,而第二结果可以包括未采用的结果。对于第二变体,第一结果和第二结果可能是相反的,因此第一结果可以包括未采用的结果,而第二结果可以包括第二结果。同样,可以通过不同的指令编码或控制寄存器中的参数来区分这些变体。因此,通过提供不同的变体,其中可以在第一变体的情况下针对被采用的结果或者在第二变体的情况下针对未采用的结果施加推测执行的限制,为程序员或编译器提供了灵活性来根据被采用的结果还是未采用的结果更容易受到推测缓存定时边通道的攻击而选择使用哪个变体。

[0047] 可以有多种方式,其中处理装置的微架构可以确保:当针对条件分支指令的单向推测变体预测第一结果时,施加推测执行的限制,而当预测第二结果时不施加。

[0048] 在一个示例中,分支预测电路可以基于分支预测状态信息进行其预测,并且可以把条件分支指令的单向推测变体排除在分支预测状态信息的训练之外,其中该分支预测状态信息是基于先前的条件分支指令的结果而训练的。通过把单向变体排除在分支预测状态的训练之外,可以防止分支预测器学习单向变体的实际结果,从而使单向变体的预测可能偏向于第二结果,对于该第二结果,推测受到的限制较少,例如通过针对单向变体不包括任何存储的分支预测状态,或通过针对单向变体包括默认预测第二结果的分支预测状态的条目,而无需基于实际分支结果进行训练。

[0049] 例如,在一些分支预测器中,对于没有存储的分支预测状态的任何指令,预测可以默认为未采用的结果。在这种情况下,对于单向推测条件分支指令的第一变体(采用第一结果而未采用第二结果),有可能不为该单向预测分支分配任何分支预测状态信息,因为分支预测器可以默认为没有存储的分支预测状态的任何指令预测第二结果(未采用),以确保不可能基于第一结果来推测地执行指令。相反,对于第二变体(对该第二变体来说,要对其限制推测的第一结果是未采用的结果),可以在分支预测电路中分配条目,将第二(被采用)结果指定为默认预测,即使单向分支的实际分支结果是第一结果它也将继续预测第二结果(因为没有基于单向分支的实际结果的训练),以确保可以没有基于第一结果的推测。

[0050] 限制对微架构中的条件分支指令的单向推测变体的第一结果的推测的另一种方式可以是:分支预测器仍可以以与不存在单向推测控制但然后分支预测期基于其来输出预测的条件分支类似的方式基于单向预测的实际结果来训练其分支预测状态信息,处理管线可以控制是否允许推测,如果允许推测,是否应当限制缓存。因此,通过这种微架构途径,分

支预测器可以基于类似于其他双向推测的条件分支的单向推测变体来训练其分支预测状态,但是可以基于针对单向推测变体预测哪个结果来应用对于是否进行推测的过滤。

[0051] 应当理解,这两种微架构途径都在条件分支指令的单向推测变体的架构定义之内,就该条件分支指令的单向推测变体而言,当预测第一结果时施加对推测的限制,而在预测第二结果时省略对推测的限制。

[0052] 另外,在一些示例中,指令解码器还可以支持条件分支指令的推测受限的变体,针对该变体,将推测执行的限制施加到跟在分支之后的下一指令,而与分支预测电路选择第一结果还是第二结果无关。这对于程序流的两个分支都被认为存在潜在的基于推测的缓存定时边通道攻击风险的情况可能有用,因此对于该特定分支,管线不会对针对该指令做出的预测进行推测地操作,可替代地如果允许推测执行,则无论推测是基于第一结果还是第二结果,都可以限制这种状态执行指令的缓存的影响,直到与条件分支指令相关联的条件得到解析为止。

[0053] 因此,上述技术有助于提供针对潜在的边通道攻击的鲁棒性。

[0054] 图1示意性地示出了具有包括多个管线级的处理管线的数据处理装置2的示例。管线包括用于预测分支指令的结果的分支预测器4。提取级6基于分支预测器4做出的预测生成一系列提取地址。提取级6从指令缓存器8中提取由提取地址标识的指令。解码级10对所提取的指令进行解码,以生成用于控制管线的后续级的控制信息。重命名级12执行寄存器重命名,以将由指令标识的架构寄存器指定符映射到标识硬件中提供的寄存器14的物理寄存器指定符。寄存器重命名对于支持无序执行是有用的,因为这可以通过将指令映射到硬件寄存器文件中的不同物理寄存器来消除指定同一架构寄存器的指令之间的危险,以增加指令被以不同于它们的程序顺序的顺序执行的可能性,按照程序顺序它们被从缓存8提取,这可以通过允许在较早的指令等待操作数变得可用的同时执行在后的指令来提高性能。在分支预测错误的情况下,将架构寄存器映射到不同物理寄存器的能力还可以促进架构状态的回滚。发布级16将等待执行的指令排队,直到用于处理这些指令所需的操作数在寄存器14中可用为止。执行级18执行指令以实现相应的处理操作。回写级20将执行的指令的结果回写到寄存器14。

[0055] 执行级18可以包括多个执行单元,例如,分支单元21,用于评估分支指令是否已经被正确预测;ALU(算术逻辑单元)22,用于执行算术或逻辑运算的;浮点单元24,用于使用浮点操作数执行运算;向量处理单元25,用于处理向量运算,在向量运算中响应于单个指令而处理多个独立的数据元素;以及加载/存储单元26,用于执行将数据从存储器系统加载到寄存器14的加载操作或将数据从寄存器14存储到存储器系统的存储操作。在该示例中,存储器系统包括一级指令缓存8、一级数据缓存30、在数据和指令之间共享的二级缓存32、以及主存储器34,但是将理解,这仅仅是可能的存储器层次结构的一个示例并且其他实现方式可以具有另外的缓存级别或不同的布置。可以使用用于控制地址转换和/或存储器保护的存储器管理单元(MMU)35来控制对存储器的访问。加载/存储单元26可以使用MMU 35的转换后备缓冲器(TLB)36来将由管线生成的虚拟地址映射到标识存储器系统内的位置的物理地址。应当理解,图1所示的管线仅仅是一个示例,其他示例可以具有管线级或执行单元的不同集合。例如,按序处理器可能没有重命名级12。

[0056] 分支预测器4是推测机制的一个示例,该推测机制可以由数据处理装置用来基于

对条件分支指令的分支结果的预测和/或对间接分支指令的目标地址的预测、在知道是否确实需要数据处理操作之前推测地执行数据处理操作。还可能存在与执行单元18相关联的推测控制电路40,用于基于与该指令相关联的信息的预测(不同于预测除)来控制执行级推测地执行指令。

[0057] 例如,条件指令可以根据存储在寄存器14中的条件状态码42的值,控制执行级18执行条件处理操作。一些条件设置指令可基于指令结果使条件状态代码42更新。例如,由ALU 22处理的算术指令可以使条件代码42被更新以指示结果的属性,比如:算术运算的结果是否为零;是否为负的,或者运算产生了有符号的溢出还是无符号的溢出。然后,后续条件指令可以测试条件状态代码42的当前值是否满足某个测试条件。从架构的角度来看,如果代码满足测试条件,则可以执行相关联的处理操作(例如算术或逻辑运算),而如果条件状态代码42不满足测试条件,则该条件操作可以不执行该指令,而可以将该指令视为没有任何架构影响的无操作指令。然而,在微架构中,推测控制电路40可以在知道实际条件代码之前,基于对条件状态代码42的预测来推测地执行与条件指令相关联的处理操作,以避免等待较早的指令完成,该较早的指令可能会更改条件代码。如果发现预测不正确,则可以丢弃推测地执行的指令的结果,并且可以将程序流退回到最后的正确执行点。

[0058] 可以由推测控制电路40执行的另一种形式的推测可以是与由向量处理单元25执行的向量引入相关联的谓词值44的预测。向量指令,也称为SIMD(单指令多数据)指令,可以对存储在同一寄存器内的多个数据元素进行操作。例如,向量加法指令可以触发向量处理单元执行多个加法运算,这些加法运算中的每一个在两个向量寄存器的对应位置处添加相应的一对数据元素,以产生要写入到结果向量寄存器的对应结果元素。这允许响应于一个指令而执行多个独立的加法运算。向量指令可用于通过允许响应于指令(包括要由向量处理单元25执行的向量指令)的向量化循环的单次迭代而处理标量循环的多次迭代,来更快地处理处理指令的标量循环。

[0059] 在指令的向量化序列内,可能希望包括条件功能,以使得如果向量的一个元素不满足某些条件,则不执行本应对元素执行的后续运算,而在该同一向量内的其他元素认可被处理,如果它们满足要求的条件的话。同样,在对标量循环进行向量化时,标量循环的迭代次数可能不会映射到向量中提供的元素的数量的精确倍数,在这种情况下,可能会出现循环尾迭代,其中向量的某些元素可能不需要被处理,因为没有足够的标量迭代来完全填充上一个向量循环迭代中的向量。因此,定义谓词值44可能是有用的,该谓词值指定向量的哪些元素是活动元素。结果向量的不活动元素可被清除为零,或者可以保留执行在该指令之前存储在目的地寄存器那些部分中的先前值。

[0060] 因此,在确定对应向量指令的结果之前,可能需要知道谓词值44。谓词值44可以由较早的指令来设置,例如,等待其他指令的结果的条件指令。等待谓词被实际计算可能会延迟向量指令。如果可以对谓词的值进行预测(例如,基于执行相同指令的先前实例,或者基于所有元素均处于活动状态的默认假设),则可以推测地执行向量指令以在预测正确的情况下提高性能。如果后来发现对谓词的预测不正确,则可以将处理倒退至较早的执行点,来丢弃任何不正确地推测的指令的结果。因此,另一种形式的推测控制可以是基于谓词值44的预测来推测地执行向量指令。

[0061] 另一种形式的推测可以是对由加载/存储单元执行的加载或存储指令的地址。例

如,在加载指令跟在较早的存储指令之后或者存储指令跟在较早的加载指令之后的情况下,则可以假设它们实际上将访问不同的数据值并且因此是独立的,从而可以在第一指令之前推测地执行第二指令,以在地址确实不同的情况下提高性能。但是,如果发现推测是不正确的,而这对指令中的第二指令实际上最终访问了与第一指令相同的地址,则推测可能是不正确的,这可能导致指令之一提供了错误的结果。如果检测到错误推测,则可以将处理倒退到较早的执行点。

[0062] 攻击者可能会利用这种推测机制来获取对不应该允许攻击者访问的敏感信息的访问。处理装置可以使用基于特权的机制进行操作,其中MMU35可以定义访问许可,该访问许可将访问限制为以给定特权级别或更高级别执行的代码对存储器地址空间的特定区域的访问。控制非特权代码的攻击者可能试图利用缓存定时边通道来获取对有关攻击者无法访问的存储器的特权区域中的敏感信息的信息的访问权限。

[0063] 缓存定时边通道的基本原理是:可以通过测量访问先前在缓存中的条目所花费的时间通过测量访问已分配条目的时间来确定进入到缓存的分配的模式,尤其是哪些缓存集已用于分配。然后可以使用它来确定哪些地址被分配到缓存中。

[0064] 基于推测的缓存定时边通道的新颖之处在于它们对推测存储器读取的使用。推测存储器读取是高级微处理器的典型代表,是实现非常高性能的整体功能的一部分。通过对架构上未解析的分支(或程序流中的其他更改)之外的可缓存位置执行推测存储器读取,此外,这些读取的结果本身可用于形成其他推测存储器读取的地址。这些推测读取导致向缓存中分配条目,这些条目的地址指示第一推测读取的值。如果不受信的代码能够以这种引起对否则将在不受信的代码处不可访问的位置的第一推测读取的方式控制推测,则这将成为可利用的边通道。但是该不受信的代码可测量缓存内的第二推测分配的影响。

[0065] 对于任何形式的监管软件,不受信软件通常会将要用作偏移量的数据值传递到将由受信软件访问的数组或类似结构中。例如,应用程序(不受信的)可以基于文件描述符ID询问有关打开的文件的信息。当然,监管软件将在偏移量被使用之前检查该偏移量是否在合适的范围内,因此可以使用以下形式编写用于此范例的软件:

[0066]
```
1  struct array {
2    unsigned   long    length;
3    unsigned   char    data[];
4  };
5  struct array *arr =...;
6  unsigned long untrusted_offset_from_user =...;
7  if (untrusted_offset_from_user < arr->length) {
8    unsigned   char    value;
9    value =arr->data[untrusted_offset_from_user];
10    ...
11   }
```

[0067] 在现代微处理器中,处理器实现方式通常可以在执行与untrusted_offset_from_user范围检查(由上面的代码中的第7行暗示)相关联的分支之前,推测地执行数据访问(由上面的代码中的第9行暗示)以建立值。以监管级别(例如OS内核或管理程序)运行此代码的处理器可以从该监管级别可访问的普通存储器中的任何位置推测地加载,该位置是由不受信的软件传递的untrusted_offset_from_user的超出范围的值确定的。从架构上讲,这不是问题,就像如果推测不正确那么加载的值将被硬件丢弃一样。

[0068] 然而,先进的处理器可以使用推测地加载的值进行进一步推测。基于推测的缓存定时边通道利用了这种进一步的推测。例如,先前的示例可被扩展为以下形式的:

[0069]
```
1 struct array {
2   unsigned long length;
3   unsigned char data[];
4 };
5 struct array *arr1 = ...; /* small array */
6 struct array *arr2 = ...; /*array of size 0x400 */
7 unsigned long untrusted_offset_from_user =...;
8       if (untrusted_offset_from_user < arr1->length) {
9     unsigned char value;
10    value =arr1->data[untrusted_offset_from_user];
11    unsigned long index2 =((value&1)*0x100)+0x200;
12    if (index2 < arr2->length) {
13      unsigned char value2 = arr2->data[index2];
14    }
15 }
```

[0070] 在此示例中,然后将使用从arri->data计算出的地址结合untrusted_offset_from_user从存储器中加载的value(值)(第10行)用作进一步的存储器访问的基础(第13行)。因此,value2的推测加载来自从为value推测地加载的数据得出的地址。如果处理器对value2的推测加载导致到缓存的分配,则可以使用标准缓存定时边通道来推断该加载的部分地址。由于该地址取决于value中的数据,因此可以使用边通道推断value的部分数据。通过将这种途径应用于value的不同比特(在多次推测执行中),可以确定value的整个数据。

[0071] 图2示出了通过图示说明这种类型的攻击的图。在图2的示例中,变量x对应于上述的untrusted_offset_from_user,它是从以较低特权级别EL0运行的不受信的代码获得的。变量y对应于以上示例中的value,它是由以较高特权级别EL1运行的代码加载的,EL1被允许访问EL0无法访问但攻击者希望获得访问权限的秘密信息。将变量x与指示数组1的大小的大小参数进行比较,如果不受信的参数大于数组大小,则条件分支将跳转到后续加载指令LD(分别对应于上面示例中的第10行和第13行的加载)。然而,假定条件分支将确定未采用的结果,即使随后确定不受信的变量x超出范围,也可以推测地执行这些加载。如果攻击者已经以使#a+x映射到秘密的地址的方式选择了“x”,则这可能允许加载到越界地址#a+x以加载攻击者应当无法访问的秘密信息。然后,第二加载可以在基于秘密的一部分选择的地址处加载来自第二数组(数组2)的值。此第二加载可以导致缓存分配发生变化,然后可以由以较低特权级别(EL0)运行的特权较低的代码来利用该变化,该代码可以使用缓存定时分析来探查第二数组的哪个特定地址被缓存,从而推断出有关秘密的信息。

[0072] 因此,不受信的软件可以通过为untrusted_offset_from_user(x)提供超出范围的数量来访问监管软件可以访问的任何位置,因此,不受信的软件可以使用此途径来恢复可被监管软件访问的任何存储器的值

[0073] 现代处理器具有多种不同类型的缓存,包括指令缓存、数据缓存和分支预测缓存。如果这些缓存中的条目的分配由已基于不受信的输入加载的某些数据的任何部分的值确定,则原则上可以激发此边通道。

[0074] 作为该机制的概括,应当理解,底层的硬件技术意味着可以推测地执行经过分支的代码,因此可以推测地执行在分支之后访问存储器的任何序列。在这种推测中,在一个推测地记载的值然后被用来构造也可以推测地执行的第二加载或间接分支的地址的情况下,该第二加载或间接分支可以以如下方式留下由第一推测加载加载的值的指示,该方式可由否则将无法读取该值的代码使用对缓存的定时分析来读取。这种概括意味着,一般生成的许多代码序列会将信息泄漏到可以由其他特权较低的软件读取的缓存分配模式中。此问题的最严重形式是本节前面所述的形式,其中特权较低的软件能够选择以这种方式泄漏什么

值。

[0075] 已使用在Linux内核中包含的eBPF字节码解释器或JIT引擎、使用在内核空间中运行的代码在多个处理器上演示了此边通道。以这种方式运行的代码保存例程,以对推测地加载的数据执行必要的移位和取消引用(dereferencing)。这种机制的使用避免了在内核空间中搜索可直接利用的合适例程的需求。

[0076] 应当理解,这是利用推测的一种示例方式。代码的分析已经表明,存在少数地方,其中使用不受信的偏移量加载的值本身被用于形成地址,到可以使用此机制来取回有意义的信息量的程度。

[0077] 处理器推测经过未解析的分支是很常见的,因此,可能会在不按顺序执行操作的缓存处理器中观察到这种行为。对于一些按顺序执行其操作的处理器,推测执行可能不足以允许使用此途径在缓存中引起必要的分配。

[0078] 对于泄漏的值被特权较低的软件确定的情形,实际的软件缓解措施是确保从机密得出的地址(即在上面的示例中将用于加载value2的地址)仅在得出秘密的访问是非推测地执行的访问时才指示秘密(value中的数据)。

[0079] 这可以在一些实现方式中通过基于用于确定分支的结果的条件(即,在前面的示例中,用于清理untrusted_offset_from_user)使用条件选择或条件移动指令来实现。在这不起作用的实现方式中,可以使用新的[定义如下]屏障(该指令是对可以使用条件选择/条件移动的实现方式的NOP)。因此,条件选择/条件移动和新的屏障的结合足以解决这个问题。本节稍后将描述新屏障的细节。

[0080] 对于允许利用此边通道的序列而言,存在于特权代码中一般是不寻常的。但是,由较低特权级别提供的字节码的编译是将此类序列注入到特权软件中的途径。编译此类字节码的即时编译器将这些机制用作其编译的序列的一部分,是尤其重要的。在提供这种类型的代码注入机制(例如eBPF)可行的系统中,也可能会将其禁用。

[0081] 可能会遇到此问题的另一领域是,其中在单个异常级别内存在由软件强制执行的特权边界,就像Java脚本解释器或Java运行时中可能发生的那样。例如,在解释器中,特权软件强制执行的一个关键元素涉及对本示例中看到的不受信的值进行清理的过程,因此有可能给出此机制的示例。类似地,由Java字节码的运行时编译生成的序列可能需要在其生成的序列中并入权变措施(work-around)。

[0082] 在插入此屏障不切实际的情况下,一种替代的插入方案是可以插入DSB SYS和ISB的组合以防止推测,但这可能会比使用条件选择/条件移动和CSDB屏障具有更大的性能影响。

[0083] 新屏障的第一示例:

[0084] CSDB是新的条件推测屏障。

[0085] 在屏障完成之前:

[0086] 1)如果按程序顺序在屏障之后出现的任何加载、存储、数据或指令预加载RW2,其对条件选择指令的结果具有地址依赖性,其中:

[0087] i.对于条件选择指令的输入寄存器之一,条件选择指令具有对已推测地执行的加载R1的寄存器数据依赖性,并且

[0088] ii.对于条件选择指令的另一输入寄存器,条件选择指令不具有对R1的寄存器依

赖性,并且

[0089] iii.条件选择指令的条件是使得如果未按架构执行R1则选择不依赖于R1的输入,

[0090] 那么RW2的推测执行以如下方式不会影响缓存中的条目的分配,该方式可用于通过评估缓存中的哪些条目已被分配或逐出,来确定来自R1的推测加载数据值的值的任何部分。

[0091] 2)如果按程序顺序在屏障之后出现的任何间接分支(B2),其目标地址对条件选择指令的结果具有寄存器依赖性,其中:

[0092] i.对于条件选择指令的输入寄存器之一,条件选择指令具有对已推测地执行的加载R1的寄存器数据依赖性,并且

[0093] ii.对于条件选择指令的另一输入寄存器,条件选择指令不具有对R1的寄存器依赖性,并且

[0094] iii.条件选择指令的条件是使得如果未按架构执行R1则选择不依赖于R1的输入,

[0095] 那么B2的推测执行以如下方式不影响缓存中的条目的分配,该方式可用于通过评估缓存中的哪些条目已被分配或逐出,来确定来自R1的推测加载数据值的值的任何部分。

[0096] 不能推测地执行屏障,但是一旦知道它不是推测的,则屏障可以完成。

[0097] 新屏障的第二示例:

[0098] CSDB是新的条件推测屏障。

[0099] 在屏障完成之前:

[0100] 1.对于按程序顺序出现在推测屏障指令之后任何加载、存储、数据或指令预加载RW2,其对条件移动指令的结果具有地址依赖性,其中

[0101] i.对于条件移动指令的输入寄存器,条件移动指令不具有对已经推测地执行的加载R1的寄存器依赖性,并且

[0102] ii.条件移动指令的条件是使得如果未按架构执行R1则该条件通过,

[0103] 则RW2的推测执行以如下方式不影响缓存中的条目的分配,该方式可用于通过评估缓存中哪些条目已被分配或逐出,来确定来自R1的推测加载数据值的值的任何部分。

[0104] 2.对于按程序顺序在屏障之后出现的任何间接分支(B2),其目标地址对条件移动指令的结果具有寄存器依赖性,其中:

[0105] i.对于条件移动指令的输入寄存器,条件移动指令不具有对已经推测地执行的加载R1的寄存器依赖性,并且

[0106] ii.条件移动指令的条件是使得如果未按架构执行R1该条件通过,

[0107] 则B2的推测执行以如下方式不影响缓存中的条目的分配,该方式可用于通过评估缓存中的哪些条目以被分配或逐出,来确定来自R1的推测加载数据值的值的任何部分。

[0108] 不能推测地执行屏障,但是一旦知道它不是推测的,则屏障可以完成。

[0109] 屏障的使用

[0110] 这些示例说明如何将屏障用于在处理器上执行的汇编代码中。

[0111] 以前面示出的示例为例:

```
struct array {
  unsigned long length;
  unsigned char data[];
};
struct array *arr1 = ...; /* small array */
struct array *arr2 = ...; /* array of size 0x400 */
unsigned long untrusted_offset_from_user = ...;
  if (untrusted_offset_from_user < arr1->length) {
    unsigned char value;
    value = arr1->data[untrusted_offset_from_user];
    unsigned long index2 = ((value&1)*0x100)+0x200;
    if (index2 < arr2->length) {
      unsigned char value2 = arr2->data[index2];
    }
}
```

[0112]

[0113] 在第一示例中,可以将其编译为以下形式的汇编代码:

```
  LDR X1, [X2] ; X2 is a pointer to arr1->length
  CMP X0, X1 ; X0 holds untrusted_offset_from_user
  BGE out_of_range
  LDRB W4, [X5,X1] ; X5 holds arr1->data base
  AND X4, X4, #1
  LSL X4, X4, #8
  ADD X4, X4, #0x200
  CMP X4, X6 ; X6 holds arr2->length
  BGE out_of_range
  LDRB X7, [X8, X4] ; X8 holds arr2->data base
out_of_range
```

[0114]

[0115] 在此情况下,可以通过将该代码更改如下来缓解边通道:

```
  LDR X1, [X2] ; X2 is a pointer to arr1->length
  CMP X0, X1 ; X0 holds untrusted_offset_from_user
  BGE out_of_range
  LDRB W4, [X5,X1] ; X5 holds arr1->data base
  CSEL X4, XZR, X4, GE
  CSDB ; this is the new barrier
  AND X4, X4, #1
  LSL X4, X4, #8
  ADD X4, X4, #0x200
  CMP X4, X6  ; X6 holds arr2->length
  BGE out_of_range
  LDRB X7, [X8, X4] ; X8 holds arr2->data base
out_of_range
```

[0116]

[0117] 在第二示例中,等效代码如下:

[0118] 原始代码:

```
  LDR R1, [R2] ; R2 is a pointer to arr1->length
  CMP R0, R1 ; R0 holds untrusted_offset_from_user
  BGE out_of_range
  LDRB R4, [R5,R1] ; R5 holds arr1->data base
  AND R4, R4, #1
  LSL R4, R4, #8
  ADD R4, R4, #0x200
  CMP R4, R6  ; R6 holds arr2->length
  BGE out_of_range
  LDRB R7, [R8, R4]; R8 holds arr2->data base
out_of_range
```

[0119]

[0120] 添加缓解措施的代码:

```
LDR R1, [R2] ; R2 is a pointer to arr1->length
CMP R0, R1 ; R0 holds untrusted_offset_from_user
BGE out_of_range
```

[0121]

[0122]
```
LDRB R4, [R5,R1] ; R5 holds arr1->data base
MOVGE R4, #0
CSDB
AND R4, R4, #1
LSL R4, R4, #8
ADD R4, R4, #0x200
CMP R4, R6  ; R6 holds arr2->length
BGE out_of_range
LDRB R7, [R8, R4]; R8 holds arr2->data base
out_of_range
```

[0123] 为了防止在数据缓存、指令缓存或分支预测缓存中创建此边通道,可以在以下情况下使用此缓解方法:

[0124] • 根据从不受信的偏移量读取的值确定数据地址

[0125] • 从不受信的偏移量中读取的值确定间接分支目的地

[0126] • 从不受信的偏移量中读取的值确定分支决策

[0127] 当应用到涉及使用不受信的值的特定代码序列时,此缓解措施将防止该代码序列可用于利用此边通道访问任何数据。

[0128] 对于某些但不是全部的实现方式,在设备存储器中映射特别重要的秘密(例如加密密钥)将防止其被分配到缓存中。在操作系统中可行的情况下,以这种方式映射此类数据可以用作这些实现方式的附加保护措施,尽管会明显提高性能成本。

[0129] 因此,如图3所示,可以提供推测屏障指令的处理方法。在步骤50,对推测屏障指令进行解码。作为响应,在步骤52,指令解码器10控制管线的其余级,来防止按程序顺序在条件屏障指令CSDB之后出现的后续操作推测地影响缓存(例如,此缓存可以是数据缓存30、指令缓存8、分支预测器4内的分支预测缓存、或其他缓存,例如,值预测器缓存或加载/存储别名预测器)中的条目分配,其中该后续操作对按程序顺序在屏障之前的较早的指令具有地址依赖性。可以通过完全防止后续指令的推测执行,或者通过允许推测执行但防止其更新缓存或将任何缓存的条目分配到缓存的单独推测区域中,来防止对缓存分配的推测影响,例如,如果特权级别降低,则可以将该推测区域擦除(当推测被正确地解析时,可将数据从推测区域传输到主区域中)。跟在屏障之后的后续指令对缓存的推测影响的限制可能会持续到屏障完成,即直到与在屏障之前的较早的指令相关联的任何推测都得到解析为止。

[0130] 下面描述推测屏障指令CSDB的另一示例。屏障的语义是:按程序顺序出现在CSDB之后的任何指令(除分支指令外)都不能使用以下任何结果来推测地执行:

[0131] 任何指令的数据值预测,或

[0132] 针对除条件分支指令外的任何指令对条件代码42的预测,或

[0133] 针对在架构上尚未解析的按程序顺序在CDSB之前出现的向量指令对向量预测状态44的预测。

[0134] 为了定义CSDB,条件代码42和向量谓词值44不被视为数据值。

[0135] 此定义准许:控制CSDB之前和之后的流推测,以及CSDB之后的条件数据处理指令的推测执行,除非它们使用数据值的结果或按程序顺序在CSDB之前出现的、尚未在架构上解析的指令的条件代码预测。上述代码示例也可以与该屏障指令示例一起使用。

[0136] 推测的另一示例是关于加载指令和存储指令(在程序顺序内以任一顺序出现)是否将访问相同地址的推测。在许多现代的高性能处理器中,进行了性能优化,从而对地址的加载将推测地绕过较早的存储,该较早的存储的目标地址尚不为硬件所知,但实际上与加

载的地址相同。发生这种情况时,加载将推测地读取该地址处的数据的较早的值,而不是由该存储写入的值。该推测地加载的值然后可用于随后的推测存储器访问,这将导致分配到缓存中,并且这些分配的定时可用作对被选作地址的数据值的观察边通道。原则上,在高级无序处理器中,在以下形式的任何代码序列中:

```
STR X1, [X2]
...
LDR X3, [X4] ; X4 contains the same address as X2
<arbitrary data processing of X3>
LDR X5, [X6, X3]
```

[0137]

[0138] 然后可以使用X3的值来推测地执行该序列中的第二加载,该值是在X3中从第一加载返回的推测值得出的。该推测地加载的值可以取自在第一地址处保存的值,该第一地址在程序的执行中比重写该值的STR更早。通过第二加载的推测执行生成的任何缓存分配都将揭示有关推测地加载到X3中的该较早的数据的一些信息。攻击者可以用此来规避存储正在覆写某些较早的数据以防止发现该值的情况。这种推测绕过途径通过一系列推测加载得以扩展,因此在这种情况下

```
STR X1, [X2]
...
LDR X3, [X4] ; X4 contains the same address as X2
<arbitrary data processing of X3>
LDR X5, [X6, X3]
<arbitrary data processing of X5>
LDR X7, [X8, X5]
```

[0139]

[0140] 然后可以使用X3的值推测地执行此序列中的第二和第三加载,该X3的值取自第一地址处保存的值,该第一地址在程序的执行中比重写该值的STR更早。由第三加载的推测执行生成的任何缓存分配都将揭示有关X5中的数据的一些信息。在这种情况下,如果攻击者控制了指向X2和X4的地址处保存的先前值,那么它能影响后续的推测,从而允许通过第二推测加载选择数据,并通过检查由第三加载引起的缓存分配来揭示所选数据。

[0141] 在存储和第一加载都位于相同的虚拟和物理地址的情况下,这种推测重新排序只能在单个异常级别内发生。

[0142] 在存储和第一加载是到不同的虚拟地址但到相同的物理地址的情况下,可能在不同异常级别的代码之间发生推测重新排序,从而在此情况下:

```
STR X1, [X2]
...
ERET ; exception return to a lower level
...
LDR X3, [X4] ; X4 contains a different virtual address as X2, but the same
physical address
<arbitrary data processing of X3>
LDR X5, [X6, X3]
```

[0143]

[0144] 使用X3作为偏移量推测地加载到缓存中的位置可以指示先前的数据值,该数据值位于X2和X4所指向的物理地址处。

[0145] 在现代高性能处理器中,如果存储的地址就可用性而言相对于加载的地址的可用性被延迟,例如,由于在存储地址的生成过程中的缓存未命中,则展示存储和对同一地址的后续加载的重新排序并且由这样的加载进行的较旧数据的推测读取就相对简单了。

[0146] 在存储和加载使用相同的寄存器来传送地址的情况下,处理器通常不会推测地在较早的存储之前对相同的地址执行加载。但是,在一些微架构特定的情况下,原则上对某些实现方式是可能的。这种重新排序的确切条件通常是处理器正在处理的先前存储器访问的

延迟的复杂函数。

[0147] 这种机制的具体关注点将是存储和第一加载在栈上的访问(使用栈指针或其他具有相同地址的寄存器),因为这是代码中相对常见的模式。原则上,这可以提供如下机制,通过该机制,早先存在于栈上但已被覆写的值将控制处理器的后续推测。对于特权栈,早先存在于栈上的值实际上可能在特权较低的执行控制之下。

[0148] 在以下序列中:

[0149]
```
STR X1, [SP]
...
LDR X3, [SP]
<arbitrary data processing of X3>
LDR X5, [X6, X3]

<arbitrary data processing of X5>
LDR X7, [X8, X5]
```

[0150] 然后这可以为已经确定了在存储(可能是由于系统调用请求处理某些数据)之前被保存在栈上的值的特权级别较低的代码提供控制通道,来使用第二加载将数据的推测加载定向到处理器的特权更高的地址空间地址中的任何地方。然后出于以下事实使得第二加载的结果是可观察的:它被用来形成第三加载的地址,该第三加载引起缓存分配。可以以与应用于所有这些边通道的方式相同的方式、通过经典的缓存定时分析来检测该缓存分配的存在。原则上,这可以允许使用定时边通道通过特权较低的代码读取任意特权数据。

[0151] 类似地,可以使用函数指针来重用栈,从而允许在特权更高的地址空间中推测地运行任意代码的选择,如本示例所示:

[0152]
```
STR X1, [SP]
...
LDR X3, [SP]
...
BLR X3.
```

[0153] 原则上,这将允许选择推测工具来揭示有趣的数据。

[0154] 至少在一些实现方式中可能表现出的该行为的另一种形式是:无序处理器可以使加载从指令流中的稍后存储推测地返回数据,如在该序列中可见:

[0155]
```
...
LDR X3, [X4]
<arbitrary data processing of X3>
LDR X5, [X6, X3]
...
STR X1, [X2] ; X2 contains the same address as X4
```

[0156] 在发生这种情况时,通过第二加载在缓存中进行的分配可能会导致通过使用缓存定时边通道来观察稍后存储的值。

[0157] 在一些实现方式上演示了概念的简单证明,其中存储使其地址相对于到同一地址的稍后加载有所延迟,导致上述类型的稍后推测存储器访问。那些推测存储器访问导致在缓存中的分配,该分配可以使用定时边通道来揭示通过确定保存在正在向其存储或从中加载的存储器位置中的较早的值而选择的数据的值。使用定制代码证明了这一概念。

[0158] 这种形式的绕过的更一般的情况,特别是在存储地址在加载地址之前或与之同时可用(通常在访问栈时发生)的情况下,尚未得到证明,并且对于用户代码而言,保证必要的复杂条件来延迟先前的存储器访问来引起处理器进行必要的重新排序以泄漏此类数据,将

是非常困难的。但是,不可能排除这种机制可能被用作低带宽通道来从特权更高的存储器中读取数据。

[0159] 尚未证明通过加载观察稍后存储的机制,但是据信至少在一些实现方式上是可行的。

[0160] 可以提供两种存储推测屏障指令来缓解这种攻击:SSBB和PSSBB。使用SSBB屏障确保SSBB之前的使用虚拟地址的任何存储不会被SSBB之后对相同虚拟地址的加载的任何推测执行绕过。SSBB屏障还确保SSBB之前的到特定虚拟地址的任何加载不会从SSBB之后的存储中推测地加载。在异常级别内的软件管理特权的情况下,此屏障可用于防止使用此机制来利用推测加载。PSSBB屏障的使用可确保PSSBB之前的使用特定物理地址的任何存储不会被PSSBB之后的到同一物理地址的加载的任何推测执行绕过。PSSBB屏障还确保PSSBB之前的到特定物理地址的任何加载不会从PSSBB之后的存储中推测地加载。

[0161] 图4示出了存储推测屏障指令的处理方法,该指令可以是上述的SSBB或PSSBB指令。在步骤60,对这种存储推测屏障指令进行解码。在步骤62,指令解码器10控制处理管线,以防止按程序顺序在存储推测屏障指令之前的较早的存储指令被后续加载指令的推测执行绕过,该后续加载指令按程序顺序跟在存储推测屏障指令之后并且指定与较早的存储指令相同的地址。对于SSBB指令,“相同的地址”是指相同的虚拟地址,而对于PSSBB指令,“相同的地址”是指相同的物理地址。

[0162] 另外,在步骤64,响应于存储推测屏障指令,指令解码器控制管线的后续级,来防止按程序顺序在存储推测屏障指令之前的较早的加载指令推测地加载由后续存储指令生成的数据,该后续存储指令按程序顺序跟在存储推测屏障指令之后并且指定相同的地址(同样,取决于是SSBB还是PSSBB变体被解码,“相同的地址”可以是虚拟地址或物理地址)。因此,此屏障使得编译器或程序员能够提供针对推测边通道攻击的保护。

[0163] 推测的另一示例是分支预测器4对条件分支指令的结果(被采用或未采用)的预测,然后可以根据该预测引起对后续指令的推测执行。通常在高性能CPU微架构设计中推测分支。为了以高精度进行推测,生成了预测表,该预测表基于该分支的先前执行来预测是否采用该分支。

[0164] 提出了提供单向分支,即,仅在一个方向上(不管是被采用还是未采用,而不是两者)进行推测的分支。除了允许在两个方向(被采用和未采用)进行推测的双向分支之外,在指令集架构中还可以支持这种单向分支。

[0165] 考虑下面的高级代码示例:

[0166]
```
<pseudo-code>
if a < b {
// sensitive code, do not speculate into here.
}
// non-sensitive code
</pseudo-code>
```

[0167] 这通常将转换为要在处理管线上执行的指令序列,例如:

```
<psuedo-asm>
cmp a, b
b.ge nonsensitive_code
sensitive_code:
; something sensitive here.
nonsensitive_code:
</pseudo-asm>
```

[0168]

[0169] 如果通常(a<b)成立,则最终预测器将以高置信预测a<b,并且可推测应执行敏感代码。这可能导致诸如上述的边通道定时攻击。

[0170] 但是,如果我们提供以下代码:

```
<psuedo-asm>
cmp a, b
bpt.ge nonsensitive_code
sensitive_code:
; something sensitive here.
nonsensitive_code:
</pseudo-asm>
```

[0171]

[0172] 在“bpt”是“branch-only-predict-taken(仅预测被采用的分支)”的汇编助记符的情况下,则微架构只能在它对分支将被采用具有高置信时才使用分支预测器进行推测。如果它对被采用的预测没有足够的置信,或者对未采用的预测具有高置信,则预测器可以暂停执行,直到分支被解析为止。这将防止推测到敏感代码区域。bpn(branch-only-predict-not-taken(仅预测未采用的分支))的正交途径可以守护在另一方向上的预测。

[0173] 微架构有许多选择来处理这些分支。它们可以使用正常的分支预测器并继续构建历史记录,但是只有在对允许的预测方向有信心的情况下才使用该历史记录,或者可以将这些分支完全排除在预测之外。也可以支持第三种形式的分支bnv(branch-never-predict(永不预测分支)),这将永不进行预测。

[0174] 因此,图5示出了条件分支指令的这种单向推测变体的处理方法。在步骤100,处理管线遇到条件分支指令的单向推测变体。可以以不同方式识别单向推测变体。在一些情况下,提取级6可能能够直接识别条件分支指令的单向推测变体,例如通过部分解码指令或访问可能由预解码器添加到指令中的预解码信息,该预解码器可以在指令被分配到指令缓存8中时对其进行分析。在另一示例中,当指令解码器10对分支指令进行解码时,如果分支被识别为条件分支指令的单向推测变体,则可以将其用信号发回到分支预测器4,然后分支预测器4可以更新分支预测状态的对应条目,即给定指令地址处的指令已经被检测为单向推测变体的条件分支指令。这可以允许提取级6在同一指令的后续提取中从所存储的分支预测状态识别出该指令是单向推测变体,即使它本身不具有解码指令的能力。

[0175] 响应于单向推测变体,在步骤102,分支预测器4确定预测哪个分支结果。如果结果是第一结果(可以是被采用的结果或未采用的结果),则在步骤104,分支预测器4预测跟在分支之后要执行的下一指令应是第一指令。如果第一结果是被采用的结果,则第一指令是分支指令的分支目标地址处的指令,而如果第一结果是未采用的结果,则第一指令是从条件分支起依次向后的下一指令。在步骤106,向第一指令施加限制推测执行。例如,可以防止第一指令被推测地执行,或者可以被允许推测地执行,但是基于推测执行对缓存结构的更新受到限制,以防止攻击者能够从缓存定时边通道攻击中得出信息。

[0176] 另一方面,如果在步骤102预测到第二结果(第二结果是与第一结果相反的结果),则在步骤108,下一指令被预测为第二指令(同样,如果第二结果未被采用则是下一顺序指令,或者如果第二结果被采用则是分支目标地址处的指令)。在步骤110,对于第二指令反而省略了限制推测(将在步骤106对第一结果应用),因此第二指令的推测执行可以不受限制地进行。

[0177] 因此,利用此处描述的单向分支,这允许在预测第二结果时继续进行推测,以提高性能,但在预测第一结果的情况下限制推测以防止上述形式的攻击。一些实现方式可以仅支持单向分支的单个变体,例如上述bpt或bpn(对于bpt,图5中的第一结果是未采用的结果,第二结果是被采用的结果,而对于bpn,第一结果是被采用的结果,第二结果是未采用的结果)。其他实现方式可以同时支持bpt和bpn。这可以允许程序员或编译器根据在被采用的结果或未采用的结果之后执行的代码是否最易受到丢失敏感信息的影响而选择适当的变体。同样,一些系统也可以支持无推测变体,其中无论预测结果是第一结果还是第二结果,都始终施加推测的限制。

[0178] 以下条款列出了更多示例:

[0179] 1.一种装置,包括:处理电路,用于执行数据处理;指令解码电路,用于对指令进行解码,来控制处理电路执行数据处理;其中:指令解码电路对条件推测屏障指令作出响应。

[0180] 2.一种计算机程序,用于控制主机处理装置以提供用于执行目标程序代码的指令的指令执行环境,该计算机程序包括:指令解码程序逻辑,用于对目标程序代码的指令进行解码,来控制处理程序逻辑以执行数据处理,其中:指令解码程序逻辑对条件推测屏障指令作出响应。

[0181] 3.一种数据处理方法,包括:解码条件推测屏障指令;并且响应于对条件推测屏障指令的解码,而控制处理电路。

[0182] 图6示出了可以使用的模拟器实现方式。虽然较早描述的实施例在用于操作支持有关技术的特定处理硬件的装置和方法方面实现了本发明,但是也可以提供根据本文描述的实施例的指令执行环境,该指令执行环境是通过使用计算机程序实现的。只要此类计算机程序提供了基于软件的硬件架构实现方式,它们通常就称为模拟器。模拟器计算机程序的种类繁多,包括模拟器、虚拟机、模型、和二进制转换器,包括动态二进制转换器。通常,模拟器实现方式可以在主机处理器200上运行,可选地在主机操作系统210上运行,以支持模拟器程序220。在一些布置中,在硬件与所提供的指令执行环境之间和/或在同一主机处理器上提供的多个不同的指令执行环境之间可能存在多层仿真。从历史上看,一直需要功能强大的处理器来提供以合理的速度执行的模拟器实现方式,但是在某些情形下(例如出于兼容性或重用原因而希望运行另一个处理器的本机代码时)这种途径可能是合乎情理的。例如,模拟器实现方式可以为指令执行环境提供主处理器硬件不支持的附加功能,或者提供通常与不同硬件架构相关联的指令执行环境。在1990年冬季USENIX大会的Robert Bedichek的“Some Efficient Architecture Simulation Techniques(一些有效的架构模拟技术)”(第53-63页)中给出了对模拟的概述。

[0183] 就先前已经参考特定硬件构造或特征描述实施例的程度而言,在模拟实施例中,可以由合适的软件构造或特征提供等效的功能。例如,特定电路可以在模拟实施例中实现为计算机程序逻辑。类似地,可以在模拟实施例中将诸如寄存器或缓存之类的存储器硬件

实现为软件数据结构。在主机硬件(例如,主机处理器200)上存在前述实施例中提及的一个或多个硬件元件的布置中,在合适的情况下,一些模拟实施例可以利用主机硬件。

[0184] 模拟器程序220可被存储在计算机可读存储介质(可以是非暂态介质)上,并且向目标代码230(可以包括应用、操作系统和管理程序)提供程序接口(指令执行环境),该程序接口与由模拟器程序220建模的硬件架构的应用程序接口相同。因此,可以使用模拟器程序220从指令执行环境内执行目标代码230的程序指令,包括上述的推测屏障指令、存储推测屏障指令、和条件分支指令的单向推测形式,以使得实际上不具有上述装置2的硬件特征的主机计算机200可以仿真这些特征。模拟器程序220可以包括指令解码程序逻辑225,用于对目标代码230的指令进行解码并将它们映射到使用来自主机硬件200支持的本机指令集的一个或多个指令执行的对应功能。

[0185] 在本申请中,词语“被配置为”用于表示装置的元件具有能够实现所定义的操作的配置。在本上下文中,“配置”是指硬件或软件的互连方式或布置。例如,该装置可以具有提供定义的操作的专用硬件,或者可以对处理器或其他处理设备进行编程以执行该功能。“被配置为”并不意味着需要为了提供所定义的操作而以任何方式改变装置元件。

[0186] 尽管这里已经参考附图详细描述了本发明的说明性实施例,但是应当理解,本发明不限于那些精确的实施例,并且本领域的技术人员可以在其中实践各种改变和修改,而不脱离由所附权利要求限定的本发明的范围和精神。

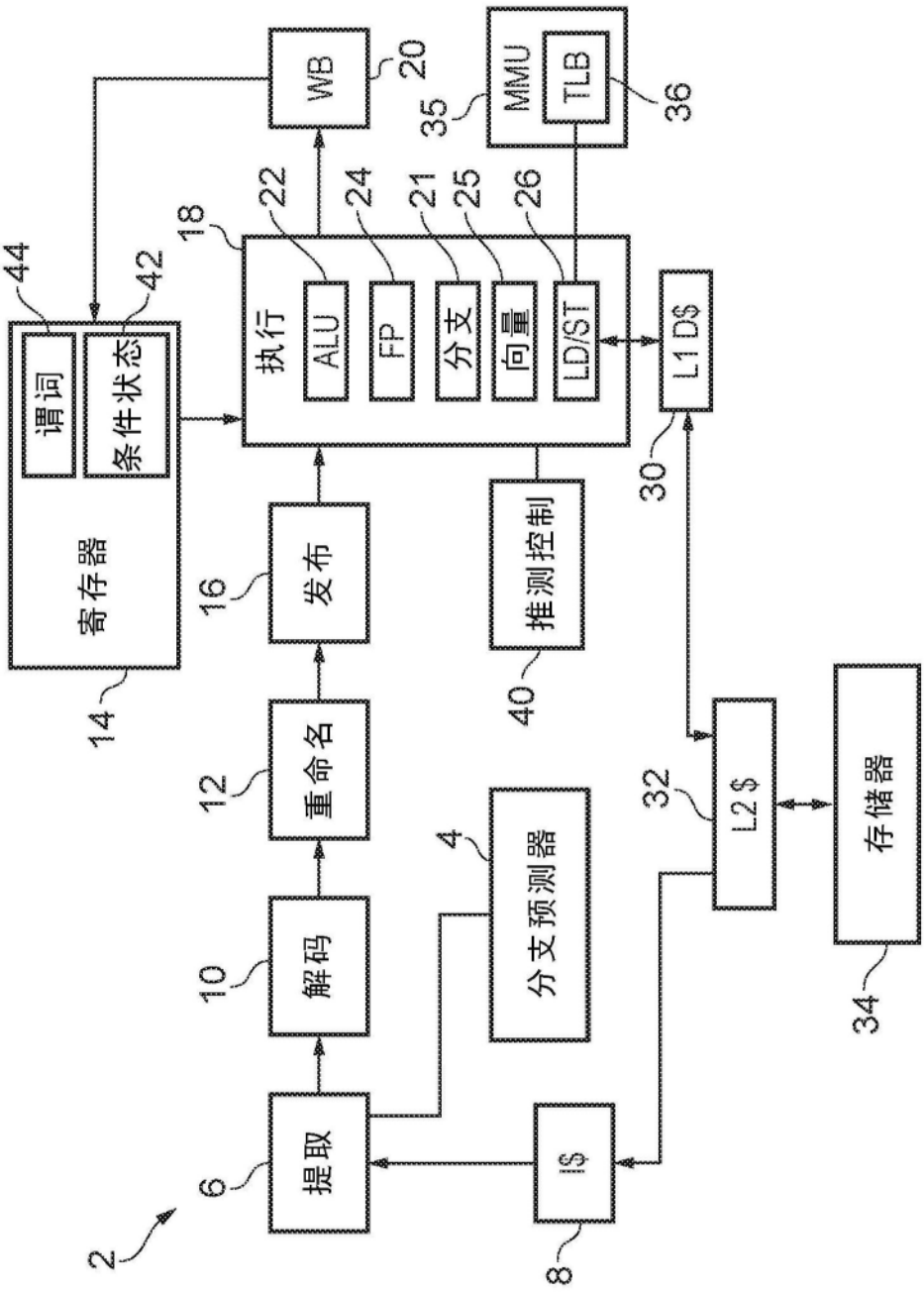

WB
20
MMU
TLB
35
36
22
24
21
25
26
18
执行
ALU
FP
分支
向量
LD/ST
L1 D$
44
42
谓词
条件状态
寄存器
14
16
发布
推测控制
40
30
12
重命名
4
分支预测器
32
L2 $
存储器
34
10
解码
6
提取
I$
8
2

图1

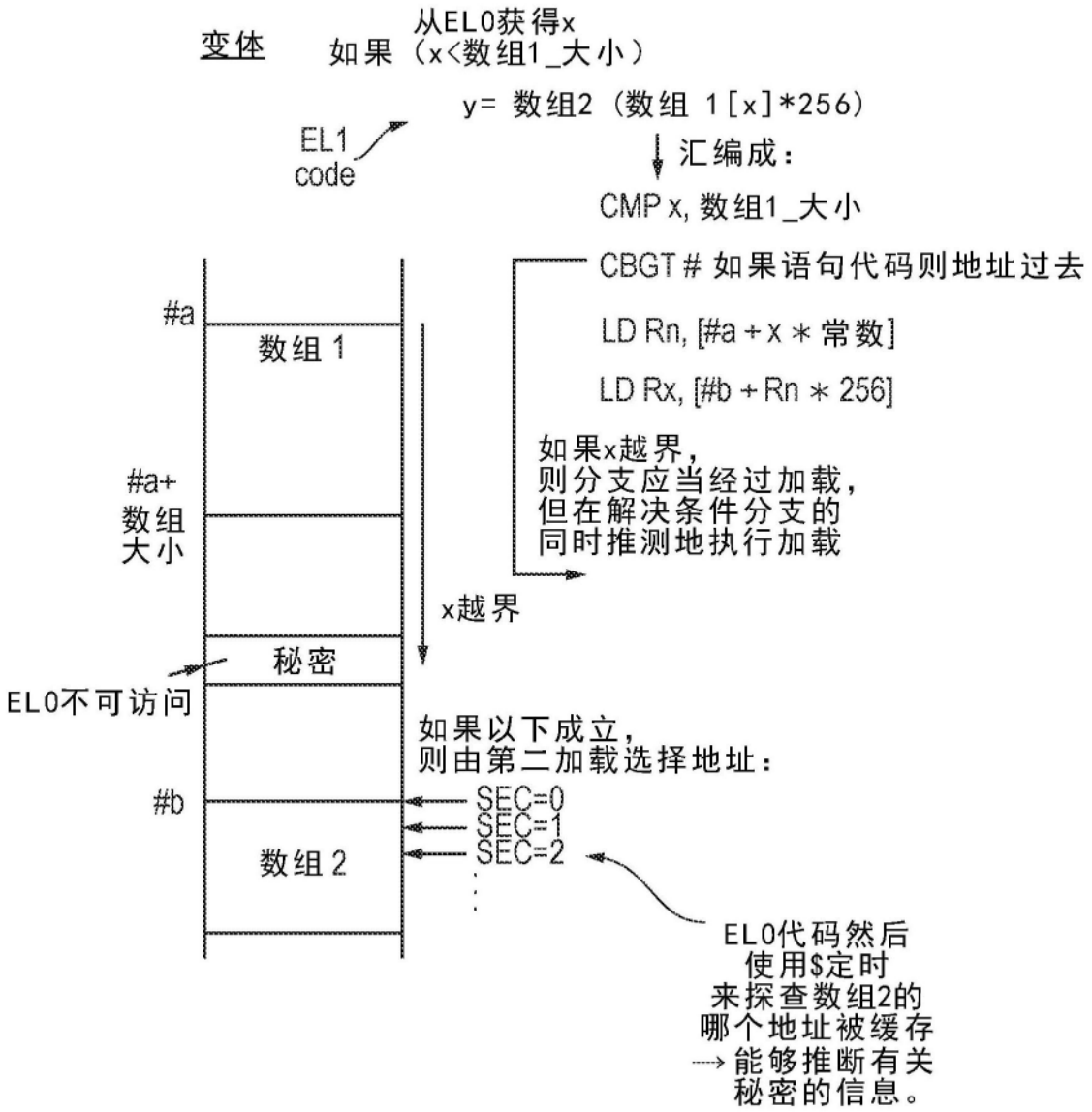

变体
从EL0获得x
如果（x<数组1_大小）
y= 数组2（数组 1[x]*256）
EL1
code
汇编成：
CMP x, 数组1_大小
CBGT # 如果语句代码则地址过去
LD Rn, [#a + x * 常数]
LD Rx, [#b + Rn * 256]
如果x越界，
则分支应当经过加载，
但在解决条件分支的
同时推测地执行加载
#a
数组 1
#a+
数组
大小
x越界
秘密
EL0不可访问
如果以下成立，
则由第二加载选择地址：
#b
SEC=0
SEC=1
SEC=2
数组 2
EL0代码然后
使用$定时
来探查数组2的
哪个地址被缓存
→能够推断有关
秘密的信息。

图2

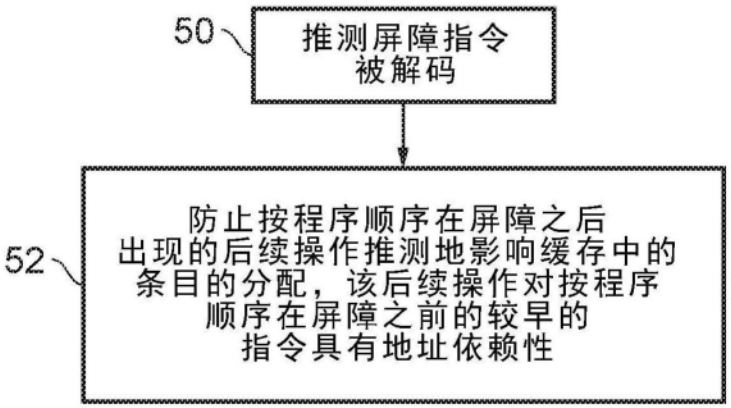

50
推测屏障指令
被解码
52
防止按程序顺序在屏障之后
出现的后续操作推测地影响缓存中的
条目的分配，该后续操作对按程序
顺序在屏障之前的较早的
指令具有地址依赖性

图3

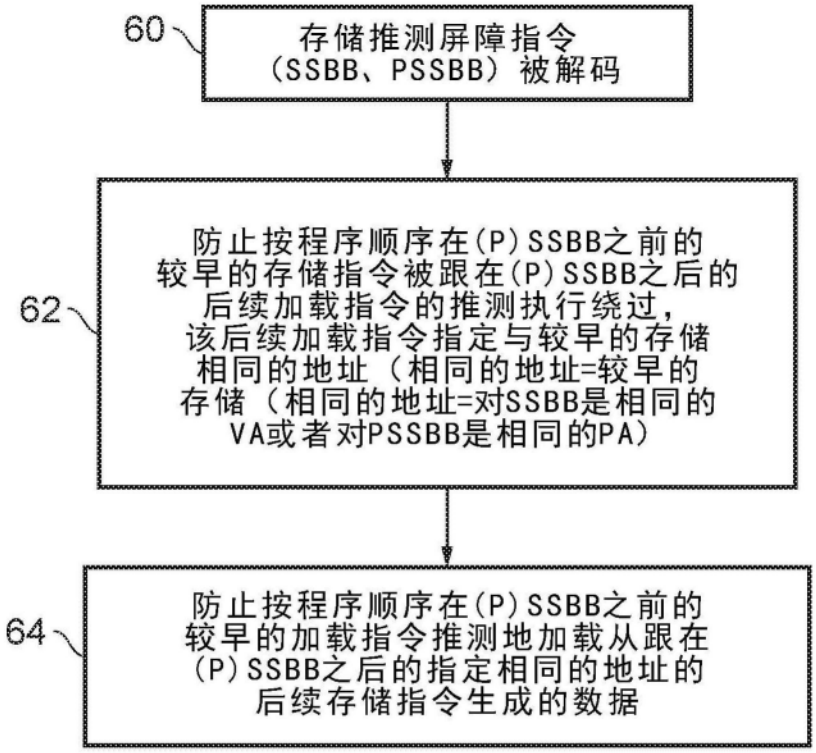

60
存储推测屏障指令
（SSBB、PSSBB）被解码
62
防止按程序顺序在(P)SSBB之前的
较早的存储指令被跟在(P)SSBB之后的
后续加载指令的推测执行绕过，
该后续加载指令指定与较早的存储
相同的地址（相同的地址=较早的
存储（相同的地址=对SSBB是相同的
VA或者对PSSBB是相同的PA）
64
防止按程序顺序在(P)SSBB之前的
较早的加载指令推测地加载从跟在
(P)SSBB之后的指定相同的地址的
后续存储指令生成的数据

图4

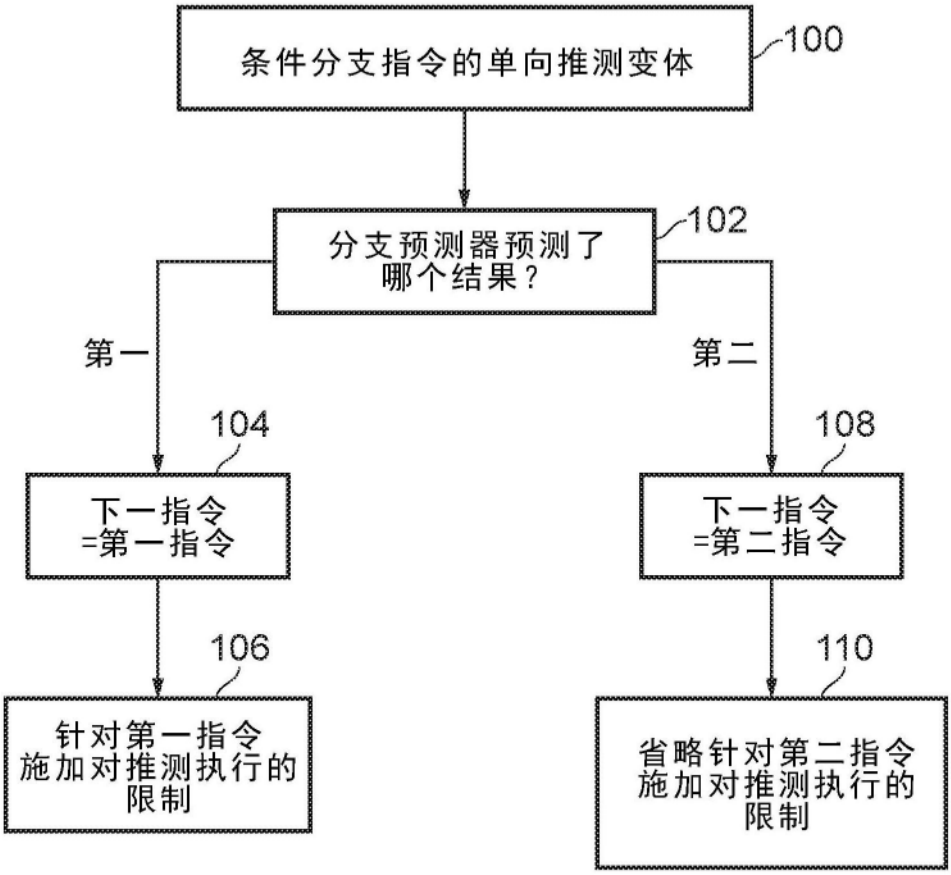

条件分支指令的单向推测变体
100
分支预测器预测了哪个结果？
102
第一
第二
104
下一指令=第一指令
108
下一指令=第二指令
106
针对第一指令施加对推测执行的限制
110
省略针对第二指令施加对推测执行的限制

图5

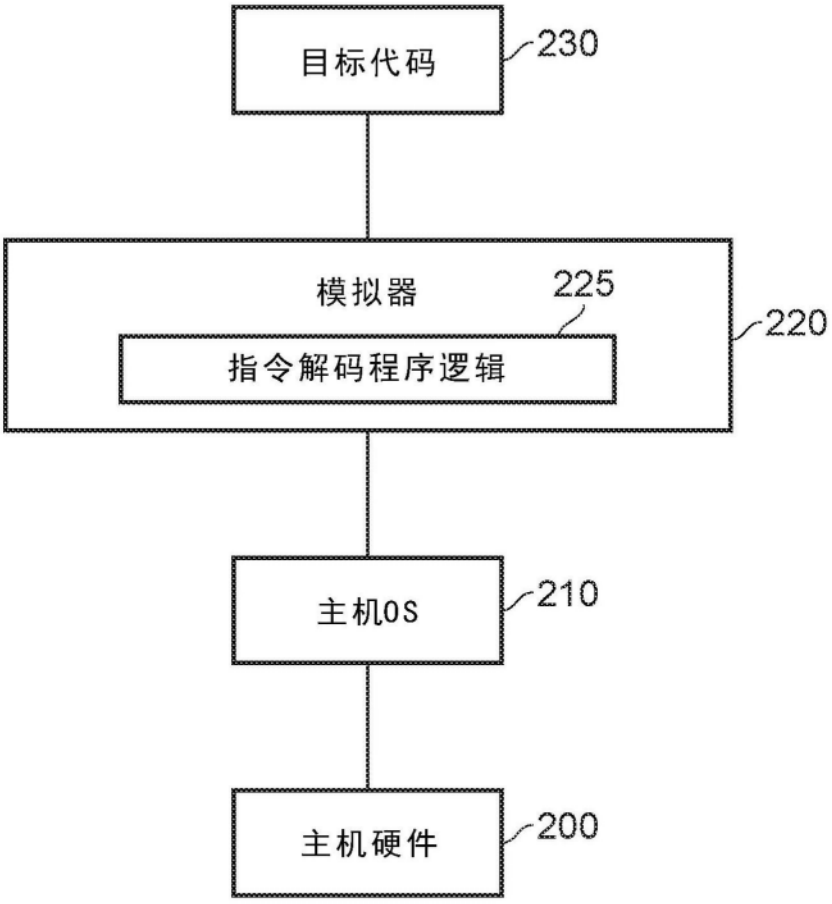


图6