(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 113535236 A
(43)申请公布日 2021.10.22

(21)申请号 202011494673.9

(22)申请日 2020.12.17

(30)优先权数据
202041016867 2020.04.20 IN
16/914,338 2020.06.27 US

(71)申请人 英特尔公司
地址 美国加利福尼亚州

(72)发明人 苏拉布·古普塔
尼兰詹·桑德拉拉扬
拉加维德·纳塔拉扬
斯雷尼瓦斯·苏布拉莫尼

(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 陈蒙

(51)Int.Cl.
G06F 9/38(2006.01)

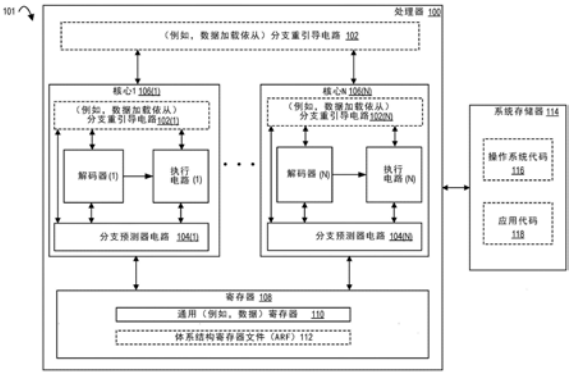
权利要求书3页 说明书34页 附图27页

(54)发明名称

基于指令集体系结构的和自动的加载跟踪的方法和装置

(57)摘要

本公开涉及基于指令集体系结构的和自动的加载跟踪的方法和装置。一种处理器包括：管线电路，其包括解码器来将指令解码成经解码的指令并且包括执行电路来执行经解码的指令；分支预测器电路，用于为分支指令生成预测路径；以及分支重引导电路，用于对于依从于来自加载指令的结果的分支指令，检查由管线电路接收的指令是否是加载指令，并且当由管线电路接收的指令是加载指令时，对于在利用解码器对分支指令的解码和利用执行电路对分支指令的执行之间对来自加载指令的结果的写回进行检查，并且当预测路径不同于基于来自加载指令的结果的路径时，在管线电路中将分支指令重引导到该路径并且使得对于基于来自加载指令的结果的路径执行分支指令。



CN 113535236 A

1.一种处理器,包括:

管线电路,其包括解码器来将指令解码成经解码的指令并且包括执行电路来执行经解码的指令;

分支预测器电路,用于为分支指令生成预测路径;以及

分支重引导电路,用于:对于依从于来自加载指令的结果的分支指令,检查由所述管线电路接收的指令是否是所述加载指令,并且当由所述管线电路接收的指令是所述加载指令时,对于在利用所述解码器对所述分支指令的解码和利用所述执行电路对所述分支指令的执行之间对来自所述加载指令的结果的写回进行检查,并且当所述预测路径不同于基于来自所述加载指令的结果的路径时,在所述管线电路中将所述分支指令重引导到所述路径并且使得对于基于来自所述加载指令的结果的所述路径执行所述分支指令。

2.如权利要求1所述的处理器,其中,所述分支重引导电路在与所述结果的最终存储目的地分离的加载值表格中对于所述结果的写回进行检查。

3.如权利要求2所述的处理器,其中,所述分支重引导电路在由所述管线电路接收的指令是所述加载指令时为所述加载值表格中用于所述加载指令的结果的条目指派索引值,并且使得所述索引值作为所述分支指令的字段被发送到所述管线电路中。

4.如权利要求3所述的处理器,其中,所述分支重引导电路在由所述管线电路接收的指令是所述加载指令时利用所述索引值来更新加载依从分支表格中用于所述分支指令的条目。

5.如权利要求3所述的处理器,其中,所述分支重引导电路为也依从于来自所述加载指令的结果的第二分支指令指派所述索引值。

6.如权利要求1所述的处理器,其中,所述分支指令的重引导发生在所述管线电路的分配阶段,该分配阶段指派所述执行电路来执行所述分支指令。

7.如权利要求6所述的处理器,其中,所述分支指令的重引导包括:从直到所述分配阶段为止的所述管线电路对于所述预测路径冲刷所述分支指令的数据。

8.如权利要求1至7中任一项所述的处理器,还包括:与包括所述管线电路的执行电路的执行阶段分离的电路,用于执行一个或多个操作以确定所述预测路径是否不同于基于来自所述加载指令的结果的路径。

9.一种方法,包括:

利用处理器的分支预测器电路为分支指令生成预测路径;

对于依从于来自加载指令的结果的分支指令,由所述处理器的分支重引导电路检查由所述处理器的管线电路接收的指令是否是所述加载指令,所述管线电路包括解码器来将指令解码成经解码的指令并且包括执行电路来执行经解码的指令;

当由所述管线电路接收的指令是所述加载指令时,由所述分支重引导电路对于在利用所述解码器对所述分支指令的解码和利用所述执行电路对所述分支指令的执行之间对来自所述加载指令的结果的写回进行检查;

当所述预测路径不同于基于来自所述加载指令的结果的路径时,由所述分支重引导电路在所述管线电路中将所述分支指令重引导到所述路径;并且

当所述预测路径不同于基于来自所述加载指令的结果的所述路径时,由所述执行电路对于基于来自所述加载指令的结果的所述路径执行所述分支指令。

10.如权利要求9所述的方法,其中,对于所述写回进行检查包括:在与所述结果的最终存储目的地分离的加载值表格中对于所述结果的写回进行检查。

11.如权利要求10所述的方法,还包括:

在由所述管线电路接收的指令是所述加载指令时,由所述分支重引导电路为所述加载值表格中用于所述加载指令的结果的条目指派索引值;并且

使得所述索引值作为所述分支指令的字段被发送到所述管线电路中。

12.如权利要求11所述的方法,还包括:在由所述管线电路接收的指令是所述加载指令时,由所述分支重引导电路利用所述索引值来更新加载依从分支表格中用于所述分支指令的条目。

13.如权利要求11所述的方法,还包括:由所述分支重引导电路为也依从于来自所述加载指令的结果的第二分支指令指派所述索引值。

14.如权利要求9所述的方法,其中,所述分支指令的重引导发生在所述管线电路的分配阶段,该分配阶段指派所述执行电路来执行所述分支指令。

15.如权利要求14所述的方法,其中,所述分支指令的重引导包括:从直到所述分配阶段为止的所述管线电路对于所述预测路径冲刷所述分支指令的数据。

16.如权利要求9至15中任一项所述的方法,还包括:利用与包括所述管线电路的执行电路的执行阶段分离的电路来执行一个或多个操作,以确定所述预测路径是否不同于基于来自所述加载指令的结果的路径。

17.一种系统,包括:

存储器,用于存储分支指令和加载指令;以及

处理器核心,与所述存储器耦合,所述处理器核心包括:

管线电路,其包括解码器来将指令解码成经解码的指令并且包括执行电路来执行经解码的指令,

分支预测器电路,用于为所述分支指令生成预测路径,以及

分支重引导电路,用于:对于依从于来自所述加载指令的结果的分支指令,检查由所述管线电路接收的指令是否是所述加载指令,并且当由所述管线电路接收的指令是所述加载指令时,对于在利用所述解码器对所述分支指令的解码和利用所述执行电路对所述分支指令的执行之间对来自所述加载指令的结果的写回进行检查,并且当所述预测路径不同于基于来自所述加载指令的结果的路径时,在所述管线电路中将所述分支指令重引导到所述路径并且使得对于基于来自所述加载指令的结果的所述路径执行所述分支指令。

18.如权利要求17所述的系统,其中,所述分支重引导电路在与所述结果的最终存储目的地分离的加载值表格中对于所述结果的写回进行检查。

19.如权利要求18所述的系统,其中,在由所述管线电路接收的指令是所述加载指令时,所述分支重引导电路为所述加载值表格中用于所述加载指令的结果的条目指派索引值,并且使得所述索引值作为所述分支指令的字段被发送到所述管线电路中。

20.如权利要求19所述的系统,其中,在由所述管线电路接收的指令是所述加载指令时,所述分支重引导电路利用所述索引值来更新加载依从分支表格中用于所述分支指令的条目。

21.如权利要求19所述的系统,其中,所述分支重引导电路为也依从于来自所述加载指

令的结果的第二分支指令指派所述索引值。

22.如权利要求17所述的系统，其中，所述分支指令的重引导发生在所述管线电路的分配阶段，该分配阶段指派所述执行电路来执行所述分支指令。

23.如权利要求22所述的系统，其中，所述分支指令的重引导包括：从直到所述分配阶段为止的所述管线电路对于所述预测路径冲刷所述分支指令的数据。

24.如权利要求17至23中任一项所述的系统，其中，所述处理器核心还包括：与包括所述管线电路的执行电路的执行阶段分离的电路，用于执行一个或多个操作以确定所述预测路径是否不同于基于来自所述加载指令的结果的路径。

基于指令集体系结构的和自动的加载跟踪的方法和装置

[0001] 相关申请的交叉引用

[0002] 本专利申请要求2020年4月20日递交的、标题为“用于数据依从片状分支的投机性推翻的基于ISA的和自动的加载跟踪(ISA-Based and Automatic Load Tracking for Opportunistic Override of Data-Dependent Flaky Branches)”的印度临时专利申请No.202041016867的权益,这里通过引用将该印度申请完全并入。

技术领域

[0003] 本公开概括而言涉及电子设备,更具体而言,本公开的实施例涉及用于推翻对于分支指令的预测的硬件,其中分支指令的结果依从于加载指令的结果。

背景技术

[0004] 处理器或者处理器的集合执行来自指令集的指令,例如指令集体系结构(instruction set architecture,ISA)。指令集是与编程有关的计算机体系结构的一部分,并且一般包括原生数据类型、指令、寄存器体系结构、寻址模式、存储器体系结构、中断和异常处理以及外部输入和输出(I/O)。应当注意这里的术语指令可以指宏指令,例如提供到处理器以便执行的指令,或者指微指令,例如由处理器的解码器对宏指令解码而产生的指令。

发明内容

[0005] 本公开实施例的一个方面提供了一种处理器,包括:管线电路(pipeline circuit),其包括解码器来将指令解码成经解码的指令并且包括执行电路来执行经解码的指令;分支预测器电路,用于为分支指令生成预测路径;以及分支重引导电路,用于:对于依从于来自加载指令的结果的分支指令,检查由所述管线电路接收的指令是否是所述加载指令,并且当由所述管线电路接收的指令是所述加载指令时,对于在利用所述解码器对所述分支指令的解码和利用所述执行电路对所述分支指令的执行之间对来自所述加载指令的结果的写回进行检查,并且当所述预测路径不同于基于来自所述加载指令的结果的路径时,在所述管线电路中将所述分支指令重引导到所述路径并且使得对于基于来自所述加载指令的结果的所述路径执行所述分支指令。

[0006] 本公开实施例的另一方面提供了一种方法,包括:利用处理器的分支预测器电路为分支指令生成预测路径;对于依从于来自加载指令的结果的分支指令,由所述处理器的分支重引导电路检查由所述处理器的管线电路接收的指令是否是所述加载指令,所述管线电路包括解码器来将指令解码成经解码的指令并且包括执行电路来执行经解码的指令;当由所述管线电路接收的指令是所述加载指令时,由所述分支重引导电路对于在利用所述解码器对所述分支指令的解码和利用所述执行电路对所述分支指令的执行之间对来自所述加载指令的结果的写回进行检查;当所述预测路径不同于基于来自所述加载指令的结果的路径时,由所述分支重引导电路在所述管线电路中将所述分支指令重引导到所述路径;并

且当所述预测路径不同于基于来自所述加载指令的结果的所述路径时，由所述执行电路对于基于来自所述加载指令的结果的所述路径执行所述分支指令。

[0007] 本公开实施例的又一方面提供了一种系统，包括：存储器，用于存储分支指令和加载指令；以及处理器核心，与所述存储器耦合，所述处理器核心包括：管线电路，其包括解码器来将指令解码成经解码的指令并且包括执行电路来执行经解码的指令，分支预测器电路，用于为所述分支指令生成预测路径，以及分支重引导电路，用于对于依从于来自所述加载指令的结果的分支指令，检查由所述管线电路接收的指令是否是所述加载指令，并且当由所述管线电路接收的指令是所述加载指令时，对于在利用所述解码器对所述分支指令的解码和利用所述执行电路对所述分支指令的执行之间对来自所述加载指令的结果的写回进行检查，并且当所述预测路径不同于基于来自所述加载指令的结果的路径时，在所述管线电路中将所述分支指令重引导到所述路径并且使得对于基于来自所述加载指令的结果的所述路径执行所述分支指令。

附图说明

[0008] 在附图中以示例而非限制方式图示了本公开，附图中相似的标记指示类似的要素并且其中：

[0009] 图1根据本公开的实施例图示了包括至少一个分支预测器电路和至少一个分支重引导电路的硬件处理器。

[0010] 图2根据本公开的实施例图示了包括管线式处理器中的分支预测器电路和分支重引导电路的计算机系统。

[0011] 图3根据本公开的实施例图示了加载依从分支表格条目的示例格式。

[0012] 图4根据本公开的实施例图示了一个依从分支指令的指令集体系结构(ISA)扩展的示例格式。

[0013] 图5根据本公开的实施例图示了两个依从分支指令的指令集体系结构(ISA)扩展的示例格式。

[0014] 图6根据本公开的实施例图示了为分支重引导填充加载依从分支表格(LDBT)的流程图。

[0015] 图7根据本公开的实施例图示了体系结构寄存器文件(ARF)扩展的示例格式。

[0016] 图8根据本公开的实施例图示了在分支重引导被使能时通过接收加载指令而触发的流程图。

[0017] 图9根据本公开的实施例图示了在分支重引导被使能时通过接收分支指令而触发的流程图。

[0018] 图10根据本公开的实施例图示了在分支重引导被使能时由对于加载指令的写回而触发的流程图。

[0019] 图11根据本公开的实施例图示了分支重引导的流程图。

[0020] 图12A是根据本公开的实施例图示出通用向量友好指令格式及其类别A指令模板的框图。

[0021] 图12B是根据本公开的实施例图示出通用向量友好指令格式及其类别B指令模板的框图。

[0022] 图13A是根据本公开的实施例图示出图12A和图12B中的通用向量友好指令格式的字段的框图。

[0023] 图13B是根据本公开的一个实施例图示出构成完整操作码字段的图13A中的特定向量友好指令格式的字段的框图。

[0024] 图13C是根据本公开的一个实施例图示出构成寄存器索引字段的图13A中的特定向量友好指令格式的字段的框图。

[0025] 图13D是根据本公开的一个实施例图示出构成增强操作字段1250的图13A中的特定向量友好指令格式的字段的框图。

[0026] 图14是根据本公开的一个实施例的寄存器体系结构的框图。

[0027] 图15A是根据本公开的实施例图示出示范性有序管线和示范性寄存器重命名、乱序发出/执行管线两者的框图。

[0028] 图15B是根据本公开的实施例图示出要被包括在处理器中的有序体系结构核心的示范性实施例和示范性寄存器重命名、乱序发出/执行体系结构核心两者的框图。

[0029] 图16A是根据本公开的实施例的单个处理器核心及其与片上互连网络以及与第2级(L2)缓存的其本地子集的连接的框图。

[0030] 图16B是根据本公开的实施例的图16A中的处理器核心的一部分的扩展视图。

[0031] 图17是根据本公开的实施例的可具有多于一个核心、可具有集成的存储器控制器并且可具有集成的图形的处理器的框图。

[0032] 图18是根据本公开的一个实施例的系统的框图。

[0033] 图19是根据本公开的实施例的更具体示范性系统的框图。

[0034] 图20示出了根据本公开的实施例的第二更具体示范性系统的框图。

[0035] 图21示出了根据本公开的实施例的片上系统(SoC)的框图。

[0036] 图22是根据本公开的实施例与使用软件指令转换器来将源指令集中的二进制指令转换成目标指令集中的二进制指令相对比的框图。

具体实施方式

[0037] 在接下来的描述中,阐述了许多具体细节。然而,要理解,没有这些具体细节也可实现本公开的实施例。在其他情况下,没有详细示出公知的电路、结构和技术,以免模糊对本说明书的理解。

[0038] 说明书中提及“一个实施例”、“一实施例”、“一示例实施例”等等表明描述的实施例可包括特定的特征、结构或特性,但可不一定每个实施例都包括该特定特征、结构或特性。另外,这种短语不一定指同一实施例。另外,当联系一实施例来描述特定的特征、结构或特性时,认为联系其他实施例(无论是否明确描述)来实现这种特征、结构或特性是在本领域技术人员的知识范围内的。

[0039] (例如,硬件)处理器(例如,具有一个或多个核心)可执行指令(例如,指令的线程)以在数据上操作,例如以执行算术、逻辑或者其他功能。例如,软件可请求操作并且硬件处理器(例如,其一个或多个核心)可响应于该请求而执行该操作。在某些实施例中,执行的操作(例如,线程)包括一个或多个分支操作(例如,分支指令)。

[0040] 在某些实施例中,分支操作(例如,指令)是无条件的(例如,每次该指令被执行时,

该分支就被采取)或者有条件的(例如,对于该分支采取的方向依从于条件),例如这样的情形:在条件分支(例如,条件跳转)之后要执行的指令直到该分支所依从的条件被决定(resolve)之前都不是确切已知的。这里,不是等到决定了条件为止,而是可以由处理器的分支预测器(例如,分支预测器电路)执行(例如,推测性执行)分支预测来预测分支是否将被采取,并且/或者(例如,如果预测要被采取),为该分支预测目标指令(例如,目标地址)。在一个实施例中,如果预测分支要被采取,则处理器为所采取的分支的方向(例如,路径)取得并且推测性地执行(一个或多个)指令,例如在预测的分支目标地址处找到的指令。在分支预测之后执行的指令在某些实施例中是推测性的,其中处理器尚未确定该预测是否正确。在某些实施例中,处理器在管线电路的后端(例如,在执行、引退和/或写回单元/电路中)决定分支指令。在一个实施例中,如果分支指令被处理器(例如,被后端)确定为不被采取,那么当前在管线电路中被采取的分支指令后面的所有指令(例如,以及其数据)被冲刷(例如,被丢弃)。某些分支指令被称为间接分支指令,例如这样的情形:分支目标(例如,针对该分支目标的指令指针)被存储在(例如,分支)寄存器中。

[0041] 在某些(例如,超标量)处理器(例如,具有处理器管线的增大的乱序(OoO)宽度和深度的那些)中,分支预测的准确性是使来自分支误预测的无效工作达到最低限度的关键。在某些实施例中,分支预测器(例如,分支预测器电路)从分支的过去行为学习来预测下一个(例如,传入的)分支。然而,在数据依从分支的情况下,高值的熵可从基于历史的分支预测器导致不良的预测准确性。

[0042] 这里的实施例利用了分支重引导(branch re-steer)来为下述分支减少误预测的惩罚:在这些分支中,分支的结果依从于(例如,仅一个)加载指令的结果。在一个实施例中,这种加载指令被称为馈送者(feeder)加载指令。这里的某些实施例对于管线式处理器(例如,核心)中的分支指令跟踪(一个或多个)(例如,馈送者)加载指令的进度。这里的某些实施例提供了对于指令集体系结构(ISA)的(一个或多个)扩展来传达依从分支的信息(例如,在被管线接收时,例如在解码时),并且如果加载操作的结果(例如,加载值)在分支进入管线的某个阶段(stage)(例如,管线的乱序部分)之前可用,则实际结果被用于计算分支预测。在某些实施例中,如果由分支预测器给出的原始预测(例如,预测的路径)不匹配这个基于加载值的预测(例如,基于加载值的路径),则管线(例如,管线的前端)被重引导到(例如,正确)路径,从而通过管线的早期重引导,或者避免了误预测,或者降低了误预测惩罚。这里的实施例对于数据依从片状分支(data-dependent flaky branch)的投机性重引导利用了基于ISA的和自动的加载跟踪。

[0043] 这里的某些实施例利用了一个或多个微体系结构选项,这些选项(i)包括ISA扩展和编译器提示,或者(ii)是完全在微体系结构中完成的。这里的某些实施例针对一种新颖的硬件,用于跟踪处理器的管线中的加载,以便例如基于实际(而不是预测的)加载值来推翻分支预测,其中这个早期推翻减少了当(例如,基线)分支预测器对于这种(例如,单个)加载数据依从分支提供错误预测时的分支误预测惩罚(例如,由于管线冲刷引起的)。

[0044] 某些实施例利用指令集(ISA)扩展向硬件传达程序行为,例如完全在硬件中检测数据依从分支,而不使用ISA扩展,或者对于生成加载值来获得基于数据值的预测的存储指令进行跟踪。这些实施例中的某些可在存储指令指针(instruction pointer,IP)与加载指令指针强相关时良好工作,然而存储-加载关系预测中的任何不准确性都可能导致不正确

的分支预测。另外，确定加载IP的正确动态实例来转发存储值也是一个挑战。为了避免这些问题，这里的某些实施例使用实际的加载值而不是预测的值来评估分支方向(例如，多条路径中的路径)，例如不使用对程序的编译器分析和/或不在执行加载指令和分支指令以便能够计算预测的过程中引入延迟。这里的某些实施例针对的是如下程序中的情况：其中，分支指令和其所依从于的加载指令之间的延迟已经存在于程序代码中(例如，在程序代码中不是紧接着的，或者是以其他方式充分地远离的)。在一个实施例中，编译器将此信息编码为加载指令和/或分支指令的一部分，分支指令于是可在加载准备就绪时使用该数据。投机性地，这里的某些实施例在管线中识别出加载已完成(例如，数据被接收在加载缓冲器中)并且可向其依从分支提供值。在某些实施例中(例如，在编译器没有这样标记它们的情况下)，处理器硬件(例如，分支重引导电路)在管线中识别加载已完成并且投机性地减少(一个或多个)管线冲刷的惩罚。这里的某些实施例不要求编译器支持和/或ISA变化(扩展)来作为解决方案的一个必要部分。这里的某些实施例可利用如本文所述的ISA变化。

[0045] 现在转到附图，图1根据本公开的实施例图示了包括至少一个分支预测器电路104(1)-104(N)和至少一个(例如，数据加载依从)分支重引导电路102,102(1)-102(N)的硬件处理器100。虽然描绘了多个分支预测器电路，对于核心106(1)-106(N)的分支预测可利用单个分支预测器电路。在一个实施例中，分支预测是分布式的，其中每个核心包括其自己的本地分支预测器104(1)-104(N)。每个本地分支预测器104(1)-104(N)可共享数据，例如关于处理器100执行的分支指令的历史。

[0046] 在一个实施例中，N是二以上的任何整数。硬件处理器100被描绘为耦合到系统存储器114，例如形成计算系统101。硬件处理器100的核心(例如，每个核心106(1)-106(N))可包括例如指令取得电路、解码器、执行电路或者引退电路(或者本文论述的其他单元或电路)的任何一者，作为该核心的管线电路。

[0047] 描绘的硬件处理器100包括寄存器108。作为例如对存储器114中的(例如，加载或存储)数据的访问的附加或替换，寄存器108可包括一个或多个通用(例如，数据)寄存器110来执行(例如，逻辑或算术)操作。寄存器108可包括一个或多个体系结构寄存器文件112。在一个实施例中，处理器100(例如，其分支预测器)将基于指令(例如，分支指令)的先前执行把分支历史数据(例如，情境数据)填充到一个或多个寄存器108中。在另一实施例中，分支历史可被保存到系统存储器114中。分支历史可包括分支指令的全局历史(例如，包括通过当前执行的程序代码到达分支指令的一系列分支所采取的路径的历史)，以及该分支指令的地址识别符(例如，与该分支指令相关联的指令指针值或程序计数器值)。在某些实施例中，全局分支(例如，路径和/或方向)历史包括方向信息，该方向信息指示出所决定的分支指令的方向有多经常被采取或不采取，以例如为分支指令的未来实例提供预测。在某些实施例中，全局分支(例如，路径和/或方向)历史包括路径信息，该路径信息指示出对于特定预测到达该分支指令的路径(例如，执行的指令)，例如，对于预定的预测，到达该分支指令所执行的指令的适当子集。

[0048] 系统存储器114可包括(例如，存储)以下软件中的一个或多个(例如，其任何组合)：操作系统(operating system，OS)代码116，或者应用代码118。

[0049] 注意这里的附图可不描绘所有数据通信连接。本领域普通技术人员将会明白，这是为了不模糊附图中的某些细节。注意附图中的双箭头可不要求双向通信，例如，其可指示

单向通信(例如,去往或来自该组件或装置)。在这里的某些实施例中可利用通信路径的任何或所有组合。在一个实施例中,处理器100具有单个核心。在某些实施例中,计算系统101和/或处理器100包括下文例如参考这里的任何附图论述的特征和/或组件中的一个或多个。

[0050] 作为一个示例,分支预测器改善管线式处理器的功能,例如通过生成对于(例如,有条件)分支指令的待执行实例(例如,对于代码中的"IF"操作)将采取多条路径中的哪条路径的预测。

[0051] 处理器(例如,微处理器)可采取对管线化的使用来增强性能。在管线式处理器的某些实施例内,用于执行指令的不同阶段的功能电路(例如,取得、解码、执行、引退等等)在多个指令上同时操作以实现某种程度的并行性,带来了相对于非管线式处理器的性能提高。在一个实施例中,指令取得单元(例如,电路)、指令解码器(例如,解码单元或解码电路)和指令执行单元(例如,电路)同时操作。在一个时钟周期期间,在某些实施例中,指令执行单元执行第一指令,同时指令解码器对第二指令解码,并且取得单元取得第三指令。在下一时钟周期期间,在某些实施例中,执行单元执行新解码的指令,同时指令解码器对新取得的指令解码,并且取得单元取得另外一个指令。这样,取得单元或解码器在处理新指令之前都不需要等待指令执行单元执行上一个指令。在某些实施例中(例如,在写回阶段期间),(一个或多个)被执行的指令(例如,在分支指令之后对于一条路径采取的指令)的结果在预测正确的情况下被保持(例如,被使得可见),并且在预测不正确的情况下被丢弃(例如,效果被回滚)。

[0052] 这里的实施例利用了分支重引导电路(例如,用于核心106(1)的分支重引导电路102(1)和用于核心106(N)的分支重引导电路102(N))来重引导不正确的预测,例如在分支指令被输入到核心的管线中之后、但在被发送到(例如,乱序)阶段之前的某时进行重引导。

[0053] 图2根据本公开的实施例图示了包括管线式处理器210中的分支预测器电路220和分支重引导电路222的计算机系统200。描绘的计算机系统200还包括网络装置201、输入/输出(I/O)电路203(例如,键盘)、显示器205和系统总线(例如,互连)207。

[0054] 在一个实施例中,图1中的处理器100的每个核心是处理器核心210的实例,其中N是任何正整数。在描绘的实施例中,每单个处理器(例如,每个核心210)包括分支预测器电路220和/或分支重引导电路222的实例。分支预测器电路220中可包括分支重引导电路222。分支预测器电路220可包括分支目标缓冲器(branch target buffer,BTB)、返回堆栈缓冲器(return stack buffer,RSB)、历史表格、寄存器和/或其他数据存储结构。这些之中的一个或多个可只被单个分支预测器使用,或者它们可被多个分支预测器中的任何分支预测器维护和/或使用。

[0055] 在某些实施例中,分支目标缓冲器存储(例如,在分支预测器阵列中存储)与多个分支指令(例如,已被执行了多次的一段代码的分支指令)的每一者相对应的预测目标指令。在某些实施例中,返回堆栈缓冲器将存储(例如,在最后进入数据是最先出来数据(last data in is the first data out,LIFO)的堆栈数据结构中存储)(例如,在堆栈上推送其返回地址的)任何CALL指令的返回地址。在某些实施例中,历史表格存储关于被处理器210执行的代码的多个分支的每一者的历史。在某些实施例中,寄存器包括存储分支全局路径历史的寄存器和/或存储分支全局方向历史的寄存器。

[0056] 在一个实施例中，处理器210是管线式处理器核心，该管线式处理器核心包括分支预测阶段220、取得阶段230、解码阶段240、分配阶段250、执行阶段260以及写回(例如，引退)阶段270。处理器核心210中示出的管线式阶段的每一者可包括不同级别的电路。或者，管线阶段可被细分为更大数目的阶段。另外，也可包括额外的管线阶段，例如预取阶段、指令指针生成(IP Gen)阶段，等等。

[0057] 在一个实施例中，管线式处理器210接收识别要被输入到处理器中的下一指令的指令指针(IP)(例如，经由连接(例如，线路)221接收)。例如，IP生成阶段可选择识别要被核心(例如，逻辑核心)取得和执行的程序序列中的下一指令的指令指针(例如，存储器地址)。在一个实施例中，管线式处理器210(例如，IP生成阶段)在每个时钟周期将最近取得的指令的存储器地址递增预定量X(例如，1)。

[0058] 然而，在异常的情况下，或者当分支指令被采取时，管线式处理器210(例如，IP生成阶段)可选择识别不是程序顺序上的下一循序指令的指令的指令指针。在某些实施例中，管线式处理器210(例如，分支预测阶段220)预测有条件分支指令是否要被采取，以例如减小分支惩罚。

[0059] 取得阶段230如图2中所示包括指令取得电路234，该指令取得电路234接受来自管线处理器中的上游(例如，来自分支预测阶段220)的指令指针并且从存储器202或指令缓存232取得相应指令。解码阶段240利用解码器242执行解码操作以将指令解码成经解码的指令(例如，解码成微操作)。分配阶段250如图2中所示利用分配电路252执行分配操作，例如执行寄存器重命名和调遣(例如，调度)。在一个实施例中，ISA支持一定数目的逻辑寄存器(例如，图2中的寄存器208的ARF或者图1中的ARF 112)，这些逻辑寄存器被映射到一定(例如，更大)数目的物理寄存器，例如ARF的每个寄存器指向寄存器208中的特定物理寄存器文件(physical register file，PRF)。在一个实施例中，寄存器的逻辑识别符(例如，“名称”)被映射到寄存器的物理识别符(例如，“名称”)，并且例如这些映射被更新。在一个实施例中，指令调遣包括预留指令将会使用的不同资源，包括执行资源、重排序缓冲器(reorder buffer，ROB)中的条目、发出队列、存储缓冲器274、加载缓冲器276，等等。在某些实施例中，如果资源不可用，则相应的指令被暂缓，直到某个其他指令释放要求的资源为止。执行阶段260如图2中所示利用执行电路262执行由经解码的指令指定的操作(例如，访问寄存器208、存储器202、存储缓冲器274和/或加载缓冲器)。写回阶段如图2中所示利用写回电路272引退执行的指令，例如，(一个或多个)执行的指令(例如，在有条件分支之后对于一个方向采取的指令)的结果在分支预测正确的情况下被保持(例如，被使得可见)并且在分支预测不正确的情况下被丢弃(例如，效果被回滚)。执行电路262作出的存储请求可使得数据被存储在存储缓冲器274中，例如，存储缓冲器274随后使得该数据被存储在其最终存储目的地中，例如(一个或多个)寄存器208、存储器202和/或数据缓存264内的目的地。执行电路262作出的加载请求可使得数据被加载到加载缓冲器276中，例如，加载缓冲器276随后使得该数据被加载在其所请求的存储目的地中，例如(一个或多个)寄存器208、存储器202和/或数据缓存264内的目的地。在替换实施例中，上文描述的管线式阶段也可包括额外的操作。

[0060] 分支预测器电路220(例如，分支预测器单元(Branch Predictor Unit，BPU))可向管线(例如，取得阶段230)提供与分支指令的预测采取方向相对应的指令。在某些实施例中，分支预测器电路220包括或者访问具有一个或多个条目的存储库，其中每个条目能够存

储识别分支指令的数据和识别预测方向(例如,采取的路径和/或未采取的路径)的相应数据。在某些实施例中,分支预测器电路220预测分支指令的分支目标。

[0061] 在一个实施例中,存储在分支预测器电路220中的分支指令是由编译器从要执行的代码预选择的。在某些实施例中,如图所示存储在图2的存储器202中的编译器代码204包括代码的序列,该代码的序列当被执行时将以高级别语言编写的程序的源代码转化成可执行的机器代码。在一个实施例中,编译器代码204还包括额外的分支预测器代码206,该分支预测器代码206为分支指令预测方向(例如,采取或不采取)和/或目标指令(例如,有可能被采取的分支指令的一部分(例如,预选择的分支指令))。分支预测器电路220(例如,其BTB)然后被更新以分支指令的方向预测和/或预测目标指令。

[0062] 描绘的核心(例如,其分支预测器220)包括对一个或多个寄存器(例如,寄存器208)的访问。在某些实施例中,寄存器208包括(一个或多个)通用寄存器和/或分支预测历史寄存器中的一个或多个。

[0063] 在某些实施例中,分支重引导电路222被包括来为其中分支的结果依于一个或多个被监视的加载指令的结果的分支减少误预测的惩罚。在一个实施例中,这种加载指令被称为馈送者加载指令。这里的某些实施例对于管线式处理器(例如,核心)中的分支指令跟踪(一个或多个)(例如,馈送者)加载指令的进度,例如如本文所述。在某些实施例中,分支重引导电路填充和/或更新加载依从分支表格224,以例如指示出分支指令和/或分支指令依从于其结果的加载指令。对这种指令的识别(例如,检测)和/或标记的示例在下文论述。

[0064] 在某些实施例中,分支重引导包括对以下成分中的一个或多个的使用:(1)检测加载(例如,指令)数据依从(例如,指令)分支的机制,(2)加载依从分支表格224,来存储所学习的馈送支配数据依从分支的结果的值的加载指令(例如,其IP),以及(3)加载值表格(load value table,LVT)226A(其可包括加载值表格226B中的全部加载值表格(LVT)或其子集的拷贝)。在一个实施例中,加载值表格226B位于解码器242和分配电路252之间。在另一实施例中,加载值表格226B位于执行随同被监视加载的写回的重引导检查的阶段内。在某些实施例中,加载指令(例如,如字段或者对于加载指令的其他指示所指示)在其准备好写回时(例如,当加载值被接收在加载缓冲器276中时)向加载值表格(LVT)(例如,226A和/或226B)提供加载值。在某些实施例中,如果加载值在分支进入某个阶段之前(例如,在管线的000部分之前,所述000部分例如但不限于预留站、重排序缓冲器等等)可用,则基于加载值进行重引导。在某些实施例中,从与最终存储目的地分离(并且例如与写回阶段中的加载缓冲器分离)的加载值表格读取来自加载指令的结果(例如,加载值)。

[0065] 在某些实施例中,分支重引导电路222利用要被监视来进行如本文所公开的可能重引导的分支指令来填充加载依从分支表格224中的各个条目。在一个实施例中,分支重引导电路222在加载依从分支表格224中添加识别要被监视的分支指令的条目,例如通过将该分支指令的指令指针(IP)包括到该条目中和/或将其馈送者加载指令的指令指针(IP)包括到该条目中。在一个实施例中,加载依从分支表格条目的格式是图3中的格式300。

[0066] 在某些实施例中,具有馈送者加载指令依从性的分支指令由编译器识别。在某些实施例中,具有馈送者加载指令依从性的分支指令是通过分析寄存器(例如,ARF)来识别的。

[0067] 作为一个示例,加载指令被输入到处理器210(例如,处理器210的管线)中(例如,

输入到分支预测器电路和/或指令取得阶段230中)并且(例如,该加载指令的IP)在加载依从分支表格224内被检查以确定其是否对应于被监视的分支指令。如果否,则在某些实施例中,处理器210的流程继续经过这些阶段。如果是,则在某些实施例中,分支重引导电路222将标记该加载指令来监视其结果(例如,结局)。在一个实施例中,分支重引导电路222在加载值表格(例如,226A和/或226B)中分配条目并且使得该加载指令的结果在接收到时(例如,在加载缓冲器276内接收到时)被存储到加载值表格的该条目中。在此示例中,分支指令被(例如,推测性地)输入到处理器210(例如,处理器210的管线)中(例如,输入到分支预测器电路中)并且(例如,该分支指令的IP)在加载依从分支表格224内被检查以确定其是否具有相应的被监视的加载指令。如果否,则在某些实施例中,处理器210的流程继续经过这些阶段(例如,根据推测性执行)。如果是,则在某些实施例中,分支重引导电路222将引起一个或多个检查(例如,经由连接(例如,线路)223进行)以查明加载指令的结果(例如,结局)现在是否准备就绪(例如,当在该分支指令的管线过程的(一个或多个)初始阶段期间没有准备就绪时)。在一个实施例中,将加载数据接收到加载缓冲器276中使得加载数据被(例如,从该加载指令的写回阶段270)发送到加载值表格中的相应条目(例如,基于加载IP到特定的加载值表格(LVT)索引的映射)。例如,加载值被从加载缓冲器276经由连接(例如,线路)225发送到LVT 226A和/或被从加载缓冲器276经由连接(例如,线路)227发送到LVT 226B。在一个实施例中,分支重引导电路222将引起对来自加载指令的结果的比较(例如,在将其监视的馈送者加载结果接收到加载值表格中时)(例如,在利用解码器242对分支指令解码和利用执行电路262执行分支指令之间),并且当预测的路径不同于基于来自加载指令的结果的(例如,实际)路径时,基于来自加载指令的结果将分支指令的执行重引导到该路径和/或为该路径引起分支指令的执行(例如,不推翻对分支指令的该实例的执行)。在某些实施例中,加载指令包括要被用于其结果的加载值表格(LVT)条目的识别符(例如,索引值),例如,该识别符被存储在加载依从分支表格224中的条目中。在某些实施例中,分支指令包括要被用于其馈送者加载的结果的加载值表格(LVT)条目的识别符(例如,索引值),例如,该识别符与分支指令一起被运送过管线(例如,向上运送直到对可能的重引导进行检查的最后一点(例如,阶段)为止)。在某些实施例中,在执行阶段260之前,在分配阶段250分配执行资源之前,在调度执行之前等等,检查可能的重引导(例如,一次或多次)。在某些实施例中,对于重引导的最终(例如,唯一)检查是在取得阶段230或者解码阶段240之后执行的(例如,但是在对分支指令的执行进行分配或调度之前)。以上只是示例,并且应当理解可以选择管线中的其他(一个或多个)点,例如执行阶段260或者写回阶段270之前的点。在某些实施例中,包括电路(例如,算术逻辑单元(arithmetic logic unit,ALU)228)来执行(一个或多个)操作以确定预测的路径何时不同于基于来自加载指令的结果的路径(例如,“大于”比较操作,“小于”比较操作,“等于”比较操作,等等),以例如避免使用执行阶段260资源。分支重引导电路222可引起对于不正确预测的路径的任何(例如,推测性)数据的冲刷,例如当预测的路径不同于基于来自加载指令的结果的路径时。分支重引导电路222可对于基于来自加载指令的结果的实际路径引起分支指令的执行(例如,经由将带有该结果的分支指令重引导回到管线的开始阶段(例如,前端)中)。图3根据本公开的实施例图示了加载依从分支表格条目的示例格式300。描绘的格式300包括一个或多个字段来存储(例如,由分支重引导电路存储):被监视的分支指令的IP 302(例如,用于重引导由馈送者加载指令引起的不正确

预测路径),(可选)这个分支指令的误预测的数目304,(可选)置信得分306(例如,其中304和306被用于预测的替换方案中),馈送者加载IP 308(以及例如比较信息,比如识别出加载值与(例如,恒定)值的比较的操作和值),以及加载值表格(LVT)索引310,来识别出加载指令的结果要被(例如,临时)存储在其中的相应加载值表格(LVT)的元素。例如,如果分支指令对应于“IF(a>100)”的伪代码并且馈送者加载指令将加载“a”的值(例如,从存储器加载),则比较字段可存储对“大于”比较的指示和“100”的值以引起“a”的值(例如,源发自加载值表格(LVT)表格226A或226B)与“100”的比较(例如,由ALU 228进行)以基于来自加载指令的结果“a”为分支指令确定实际路径。

[0068] 接下来,描述用于检测加载数据依从分支的示例方案。下面将参考以下伪代码示例(其中do_work可以是任何(一个或多个)操作):

```
a = MEM[x];              //L1:馈送给下面的三个“if”分支的加载
b = MEM[y];              //L2:馈送给下面的第三“if”分支的加载

if(a > 100)              //B1:直接依从于 L1 的加载值的分支
  {
    do_work();
  }
```

[0069]
```
if(a%2 == 0)             //B2:直接依从于 L1 的加载值的分支
  {
do_work();
  }

if( a == b )             //B3:直接依从于 L1&L2 的加载值的分支
  {
do_work();
  }
```

[0070] 此示例包括加载数据依从分支B1和B2,它们两者都使用L1的单个加载值(“a”)来计算结果,并且加载数据依从分支B3使用来自两个加载的数据来计算其结果。分支B3使用两个加载值来计算分支结果并且不被这里的某些实施例所支持以避免为单个分支指令跟踪多个加载的复杂性,虽然应当理解其他实施例根据这里的公开对于单个分支指令跟踪多个加载。

[0071] 如上所述,对加载数据依从分支的识别(例如,以及从而对加载依从分支表格224的填充)可经由(i)编译器和指令集体系结构(ISA)扩展,或者(ii)基于硬件的实现方式。

[0072] 经由编译器/ISA扩展的示例实现方式：

[0073] 在某些实施例中，加载依从分支检测由编译器(例如，利用数据流图)和本文论述的ISA扩展执行来使能在硬件中对加载数据依从分支的特殊处理。在一个实施例中，加载指令被标记并且到分支IP(例如，程序计数器(program counter，PC))的偏移量(或者分支IP本身)被作为扩展比特添加到加载指令(例如，包括一个或多个其他字段，例如包括但不限于下文在图12A-图13D中论述的那些)。

[0074] 这里的某些实施例使用(例如，并且提供)ISA扩展来指示(例如，存储)上文论述的L1->B1依从性和L1->B2依从性。在上述伪代码示例中，这里的某些实施例使用编译器(例如，数据流图)来检测(例如，确定)分支B1和B2直接依从于加载L1的值(例如，以及其他操作对象是即时值)。图4根据本公开的实施例图示了一个依从分支指令的指令集体系结构(ISA)扩展的示例格式400。描绘的格式400包括一个或多个字段来存储(例如，由编译器存储)：(可选)加载操作扩展402(例如，指示出分支指令只依从于一个加载)，分支IP 404，来识别出由加载指令馈送的分支指令，以及一个或多个操作406、408，以例如指示出要在加载数据的结果上执行的操作(例如，上述的[B1]if(a>100)示例中的“大于”)。第一操作指示406可以是“大于”或者小于并且第二操作指示408可以是“等于”。在上述示例中，B1使用加载值与“100”的直接比较并且B2使用模操作(%)(例如，来找出在被“2”除时的余数)。在某些实施例中，这个信息作为字段406(例如，以及408)被添加到ISA加载指令扩展以使能简单的(例如，经由图2中的ALU 228的)计算(例如，在前端中)来从加载值获得分支结果。

[0075] 来自格式400的ISA扩展的数据可被用于填充加载依从分支表格(load dependent branch table，LDBT)条目，例如，在从由编译器生成的输出(例如，机器代码)读取该指令时。

[0076] 在上述伪代码示例中，这里的某些实施例使用编译器(例如，数据流图)来检测(例如，确定)分支B3使用两个加载值来计算分支结果。图5根据本公开的实施例图示了两个依从分支指令的指令集体系结构(ISA)扩展的示例格式500。描绘的格式500包括一个或多个字段来存储(例如，由编译器存储)：(可选)加载操作扩展502(例如，指示出两个分支指令只依从于一个加载)，第一分支IP 504，来识别出由加载指令馈送的第一分支指令，一个或多个操作506、508，以例如指示出对于第一分支指令要在加载数据的结果上执行的操作(例如，上述的[B1]if(a>100)示例中的“大于”)，第二分支IP 510，来识别出由加载指令馈送的第二分支指令，一个或多个操作512、514，以例如指示出对于第二分支指令要在加载数据的结果上执行的操作(例如，上述的[B2]if(a%2==0)示例中的“模”和“等于”)。在某些实施例中，这个操作信息作为字段506和512(例如，以及508和/或514)被添加到ISA加载指令扩展以使能简单的(例如，经由图2中的ALU 228的)计算(例如，在前端中)来从加载值获得分支结果。

[0077] 来自格式500的ISA扩展的数据可被用于填充(一个或多个)加载依从分支表格(LDBT)条目，例如，在从由编译器生成的输出(例如，机器代码)读取该指令时。

[0078] 图6根据本公开的实施例图示了为分支重引导填充加载依从分支表格(LDBT)(例如，在检测到(比如在解码期间)馈送者加载指令时)的流程图600。流程600包括检测指令的加载类型602(例如，对加载指令和/或其IP的解码)，检查该加载指令是否包括用于分支重引导的ISA扩展604(例如，用于填充加载依从分支表格(LDBT)条目的ISA扩展)，并且如果

否,则结束608这个特定流程的任何进一步分支重引导动作,而如果是,则使得利用来自ISA扩展的数据填充加载依从分支表格的条目606,例如,加载IP(例如,在图3的字段308中),分支IP(例如,在图3的字段302中),比较(例如,操作)信息(例如,在图3的字段308中),等等,并且随后结束608这个特定流程的任何进一步分支重引导动作。这也可引起对加载值表格(LVT)条目的分配以及其相应的索引被插入到加载依从分支表格(LDBT)条目中(例如,在图3的字段310中)。

[0079] 上述示例说明了ISA扩展的示例设计以及在检测到(例如,解码)图2中的馈送者加载指令时采取的动作的流程图。在某些实施例中,ISA扩展(i)对在加载值上执行的操作解码以计算分支结果,并且(ii),如果加载指令馈送给多个分支,则ISA扩展将捕捉多个分支偏移量。为了使能这一点,可以使用不同的加载操作码扩展(LD OP Ext 1 402和LD OP Ext 2 502)。

[0080] 在某些实施例中,在此信息的一个或多个字段被作为扩展添加到加载指令的情况下,当该加载指令被管线接收(例如,解码)时,处理器(例如,分支重引导电路)将加载IP及其依从分支添加到LBT,例如,连同来自扩展比特的操作信息(例如,在图3的LBT条目308中称为CMP信息)。

[0081] 经由基于硬件的实现方式的示例实现方式:

[0082] 在某些实施例中,加载依从分支检测是在硬件中执行的(例如,不依赖于编译器或添加的ISA扩展)以使能在硬件中对加载数据依从分支的特殊处理。在某些实施例中,一种纯基于硬件的机制被用于检测加载依从分支(例如,上述伪代码示例中的L1->B1和L1->B2加载数据依从性)。在硬件中检测加载数据依从分支的一个实施例涉及通过影子体系结构寄存器文件(ARF)或者ARF条目的扩展来跟踪数据流。在一个实施例中,硬件(例如,分支重引导电路)检测仅被一个加载值支配的分支,例如,其中如果在跟踪期间发现多个加载在馈送(例如,寄存器)值,则硬件停止跟踪该依从链(例如,通过使(一个或多个)条目无效)。类似地,也可从依从链中排除多个寄存器操作(例如,只允许以即时操作对象执行的操作)。

[0083] 在一个实施例中,硬件(例如,分支重引导电路)从加载值计算分支结果,例如,其中在加载和分支指令之间在数据值上执行的操作被跟踪。在某些实施例中,一旦检测到加载依从分支,其馈送者加载IP和操作的序列就被记录在加载依从分支表格(LDBT)表格中。图7根据本公开的实施例图示了体系结构寄存器文件(ARF)扩展的示例格式700。描绘的格式700包括一个或多个字段来存储(例如,由分支重引导电路存储):寄存器的识别符702,在访问该寄存器的加载704(例如,该加载指令的IP),以及一个或多个操作706(0),706(1),...,706(N),例如来指示操作。在一个实施例中,一个或多个操作706(0)-706(N)指示出加载值何时可用,例如,ALU(228)要执行(一个或多个)什么操作来计算分支方向来推翻(例如,默认)分支预测。例如,一个或多个操作706(0)-706(N)是加载和分支之间的依从指令链。例如,如果加载通过OP0(R1+5->R2),OP1(R2*7->R3),Jump if R3>20的操作产生R1。

[0084] 如本文公开的(例如,经由LDBT和LVT)通过管线对加载(例如,以及其加载值表格(LVT)索引)的跟踪适用于上文描述的检测加载依从分支的两种类别的方法。图8-图11论述了可用于分支重引导的各种流程图。这些流程(或者本文描述的其他过程,或者其变体和/或组合)的一些或全部操作可在配置有可执行指令的一个或多个计算机系统的控制下执行并且被实现为总体在一个或多个处理器上执行的代码(例如,可执行指令、一个或多个计算

机程序或者一个或多个应用)、由硬件实现,或者由这些的组合实现。该代码可被存储在计算机可读存储介质上,例如,以包括可由一个或多个处理器执行的指令的计算机程序的形式。计算机可读存储介质是非暂态的。在一些实施例中,一个或多个(或者所有)操作是由其他附图的分支重引导电路执行的。

[0085] 图8根据本公开的实施例图示了在分支重引导被使能时通过接收(例如,取得)加载指令而触发的流程图800。描绘的流程800包括检测指令的加载类型802(例如,对加载指令和/或其IP的解码),检查该加载指令(例如,IP)是否存在于加载依从分支表格(LDBT)的条目中804,并且如果否,则结束808这个特定流程的任何进一步分支重引导动作,而如果是,则分配新的加载值表格(LVT)条目并且其使得该加载值表格(LVT)条目的识别符(例如,索引)被存储在LDBT表格中的相应条目内(例如,在图3中的字段310中)(例如,从而每个加载迭代使得新的LVT索引被用于相应的LDBT条目中)806,并且随后结束808这个特定流程的任何进一步分支重引导动作。

[0086] 图9根据本公开的实施例图示了在分支重引导被使能时通过接收(例如,取得)分支指令而触发的流程图900。描绘的流程900包括检测指令的分支类型902(例如,对分支指令和/或其IP的解码或者检测分支指令在处理器的管线中采取预测路径),检查该分支指令(例如,IP)是否存在于加载依从分支表格(LDBT)的条目中904(例如,图3中的字段302中),并且如果否,则结束916这个特定流程的任何进一步分支重引导动作,而如果是,则对于加载依从分支表格(LDBT)中的条目中的加载值表格(LVT)索引所识别的馈送者数据在加载值表格(LVT)中执行查找906,然后检查LVT条目是否被更新(例如,具有当前加载数据)908,并且如果否,则结束916这个特定流程的任何进一步分支重引导动作,而如果是,则基于来自加载指令的数据(例如,当前加载数据)执行分支路径910,然后检查基于该数据的分支路径是否匹配预测的分支路径912,并且如果否,则结束916这个特定流程的任何进一步分支重引导动作,而如果是,则发送(例如,向管线的取得阶段发送)重引导和/或冲刷指示以使得对于基于该数据(例如,当前加载数据)的路径执行(例如,正确的)分支指令(例如,并且取消对于基于预测的(例如,不正确)路径执行分支指令)914,并且随后结束916这个特定流程的任何进一步分支重引导动作(例如,通过将加载数据标记为现在是陈旧的)。

[0087] 图10根据本公开的实施例图示了在分支重引导被使能时由对于加载指令的写回而触发的流程图1000。描绘的流程1000包括检测到对于加载类型的指令的写回1002,检查该加载指令(例如,IP)是否存在于加载依从分支表格(LDBT)的条目中(例如,该加载IP的LDBT条目是否包括有效LVT索引)1004,并且如果否,则结束1010这个特定流程的任何进一步分支重引导动作,而如果是,将来自加载指令到LVT的写回的结果数据(以及,连同LVT索引)发送到LVT(或者多个LVT)1006,利用结果数据更新该LVT条目1008,并且随后结束1010这个特定流程的任何进一步分支重引导动作。在一个实施例中,LVT索引被指派给加载指令(例如,在图8中的806),并且之后,当加载在进行写回时,只有具有(例如,有效)LVT索引的(一个或多个)加载指令将把数据值发送到LVT(例如,在图10中的1006)。

[0088] 图11根据本公开的实施例图示了分支重引导的流程图1100。描绘的流程1100包括利用处理器的分支预测器电路为分支指令生成预测路径1102,由处理器的分支重引导电路对于依从于来自加载指令的结果的分支指令,检查由处理器的管线电路接收的指令是否是加载指令,该管线电路包括解码器来将指令解码成经解码的指令并且包括执行电路来执行

经解码的指令1104，当由管线电路接收的指令是加载指令时由分支重引导电路对于在利用解码器对分支指令的解码和利用执行电路对分支指令的执行之间对来自加载指令的结果的写回进行检查1106，当预测的路径不同于基于来自加载指令的结果的路径时由分支重引导电路在管线电路(例如，管线电路的前端)中将分支指令重引导到该路径1108，并且(可选地)当预测路径不同于基于来自加载指令的结果的实际路径时由执行电路基于来自加载指令的结果对于实际路径执行分支指令1110(例如，分支指令继续沿管线电路而下并且在管线电路的后端中的(一个或多个)执行电路中执行)。

[0089] 在某些实施例中，当指令进入管线(例如，管线的前端)时，其IP被对照LDBT中的加载IP来检查匹配，并且，如果存在匹配，则加载指令被标记以LVT索引。可以按循环赛的方式或者基于可用性来选择LVT索引。在某些实施例中，指派的LVT索引也被存储在LDBT中的相应条目中。在某些实施例中，当加载指令从存储器管线获得加载值(例如，并且进入加载缓冲器中)时，该值在LVT表格的LVT索引槽位处(例如，在前端)被更新。

[0090] 在某些实施例中，在分支指令取得时，在LBT中搜索分支IP，并且如果存在命中，则例如从表格读取LVT索引，并且随后在该LVT索引处对于该加载值查找LVT。在某些实施例中，如果加载指令已在LVT中写回了值，则例如基于加载值和分支比较条件的计算产生预期分支结果，并且当这个结果不同于分支预测器预测的时，进行重引导并且重引导前端(例如，图15B中的前端单元1530)。从而，在某些实施例中，重引导是用真实加载数据值来决定分支。这里的某些实施例不消除管线中的分支执行(例如，管线的OoO片段)，但由于这个高度准确性，使能了在前端的后来部分中的重引导。这增大了覆盖，因为即使当加载和分支在程序流中(例如，略微)更接近时或者加载写回(例如，略有)延迟时(例如由于乱序效应或者缓存错失)，重引导也是可能的。这个后来的重引导通过比分支执行更早地进行前端重引导来减少了误预测惩罚，例如，其中分支误预测惩罚(例如，时间长度)(例如，BP时间点重引导和/或分配时间点重引导)相对于前端重引导惩罚显著低于前者。这里的某些实施例聚焦于用基于加载值的预测(例如，已知其是更准确的，因为其是基于加载值的)来推翻(例如，默认)分支预测。这里的某些实施例不消除管线(例如，管线的后端)中的分支的执行，例如，因为其是正确性所必需的。例如，为了节省存储，在LDBT中可使用更少数目的标签比特来匹配分支IP或加载IP，这可导致混叠和虚假命中。这些实施例中的某些仍因为存在命中而执行对预测的推翻，例如，该推翻在LDBT中允许混叠的一些实现方式中可能不是正确的。

[0091] 在某些实施例中，在分支进入管线的某个阶段(例如，管线的OOO片段)之前，可以(例如，再次)查找LVT以查明加载值是否可用并且可为重引导计算分支结果。为了允许更快的查找，可以维持多个LVT拷贝(例如，一个在前端，另一个就在分配阶段之前)。在某些实施例中，LVT索引与分支指令一起被发送(例如，运送)过管线，例如，直到最终检查点(例如，上述示例中的分配阶段)为止。在某些实施例中，在分配阶段保持的LVT拷贝(例如，图2中的LVT 226B)是最低限度的，因为其只维护推翻结果。在某些实施例中，当相应的LVT条目在加载写回时被更新时，推翻结果可被计算并且被发送到这个LVT拷贝。作为一个设计选择，该计算(例如，由图2中的ALU 228进行的计算)可以在分支预测之时恰好适时进行或者一旦加载值可用就进行。在使用LVT拷贝的某些实施例中，结果将被从LDBT计算并发送到LVT拷贝，例如，为了避免计算电路(例如，图2中的ALU 228)的复制开销。

[0092] 下面详述其中可使用上述内容的示范性体系结构、系统等等。

[0093] 可考虑到以下示例来描述公开的技术的至少一些实施例：

[0094] 示例1.一种处理器，包括：

[0095] 管线电路，其包括解码器来将指令解码成经解码的指令并且包括执行电路来执行经解码的指令；

[0096] 分支预测器电路，用于为分支指令生成预测路径；以及

[0097] 分支重引导电路，用于对于依从于来自加载指令的结果的分支指令，检查由所述管线电路接收的指令是否是所述加载指令，并且当由所述管线电路接收的指令是所述加载指令时，对于在利用所述解码器对所述分支指令的解码和利用所述执行电路对所述分支指令的执行之间对来自所述加载指令的结果的写回进行检查，并且当所述预测路径不同于基于来自所述加载指令的结果的路径时，在所述管线电路中将所述分支指令重引导到所述路径并且使得对于基于来自所述加载指令的结果的所述路径执行所述分支指令。

[0098] 示例2.如示例1所述的处理器，其中，所述分支重引导电路在与所述结果的最终存储目的地分离的加载值表格中对于所述结果的写回进行检查。

[0099] 示例3.如示例2所述的处理器，其中，所述分支重引导电路在由所述管线电路接收的指令是所述加载指令时为所述加载值表格中用于所述加载指令的结果的条目指派索引值，并且使得所述索引值作为所述分支指令的字段被发送到所述管线电路中。

[0100] 示例4.如示例3所述的处理器，其中，所述分支重引导电路在由所述管线电路接收的指令是所述加载指令时利用所述索引值来更新加载依从分支表格中用于所述分支指令的条目。

[0101] 示例5.如示例3所述的处理器，其中，所述分支重引导电路为也依从于来自所述加载指令的结果的第二分支指令指派所述索引值。

[0102] 示例6.如示例1所述的处理器，其中，所述分支指令的重引导发生在所述管线电路的分配阶段，该分配阶段指派所述执行电路来执行所述分支指令。

[0103] 示例7.如示例6所述的处理器，其中，所述分支指令的重引导包括：从直到所述分配阶段为止的所述管线电路中对于所述预测路径冲刷所述分支指令的数据。

[0104] 示例8.如示例1所述的处理器，还包括：与包括所述管线电路的执行电路的执行阶段分离的电路，用于执行一个或多个操作以确定所述预测路径是否不同于基于来自所述加载指令的结果的路径。

[0105] 示例9.一种方法，包括：

[0106] 利用处理器的分支预测器电路为分支指令生成预测路径；

[0107] 对于依从于来自加载指令的结果的分支指令，由所述处理器的分支重引导电路检查由所述处理器的管线电路接收的指令是否是所述加载指令，所述管线电路包括解码器来将指令解码成经解码的指令并且包括执行电路来执行经解码的指令；

[0108] 当由所述管线电路接收的指令是所述加载指令时，由所述分支重引导电路对于在利用所述解码器对所述分支指令的解码和利用所述执行电路对所述分支指令的执行之间对来自所述加载指令的结果的写回进行检查；

[0109] 当所述预测路径不同于基于来自所述加载指令的结果的路径时，由所述分支重引导电路在所述管线电路中将所述分支指令重引导到所述路径；并且

[0110] 当所述预测路径不同于基于来自所述加载指令的结果的所述路径时，由所述执行

电路对于基于来自所述加载指令的结果的所述路径执行所述分支指令。

[0111] 示例10.如示例9所述的方法，其中，对于所述写回进行检查包括：在与所述结果的最终存储目的地分离的加载值表格中对于所述结果的写回进行检查。

[0112] 示例11.如示例10所述的方法，还包括：

[0113] 在由所述管线电路接收的指令是所述加载指令时，由所述分支重引导电路为所述加载值表格中用于所述加载指令的结果的条目指派索引值；并且

[0114] 使得所述索引值作为所述分支指令的字段被发送到所述管线电路中。

[0115] 示例12.如示例11所述的方法，还包括：在由所述管线电路接收的指令是所述加载指令时，由所述分支重引导电路利用所述索引值来更新加载依从分支表格中用于所述分支指令的条目。

[0116] 示例13.如示例11所述的方法，还包括：由所述分支重引导电路为也依从于来自所述加载指令的结果的第二分支指令指派所述索引值。

[0117] 示例14.如示例9所述的方法，其中，所述分支指令的重引导发生在所述管线电路的分配阶段，该分配阶段指派所述执行电路来执行所述分支指令。

[0118] 示例15.如示例14所述的方法，其中，所述分支指令的重引导包括：从直到所述分配阶段为止的所述管线电路对于所述预测路径冲刷所述分支指令的数据。

[0119] 示例16.如示例9所述的方法，还包括：利用与包括所述管线电路的执行电路的执行阶段分离的电路来执行一个或多个操作以确定所述预测路径是否不同于基于来自所述加载指令的结果的路径。

[0120] 示例17.一种存储代码的非暂态机器可读介质，所述代码当被机器执行时使得所述机器执行一种方法，该方法包括：

[0121] 利用处理器的分支预测器电路为分支指令生成预测路径；

[0122] 对于依从于来自加载指令的结果的分支指令，由所述处理器的分支重引导电路检查由所述处理器的管线电路接收的指令是否是所述加载指令，所述管线电路包括解码器来将指令解码成经解码的指令并且包括执行电路来执行经解码的指令；

[0123] 当由所述管线电路接收的指令是所述加载指令时，由所述分支重引导电路对于在利用所述解码器对所述分支指令的解码和利用所述执行电路对所述分支指令的执行之间对来自所述加载指令的结果的写回进行检查；

[0124] 当所述预测路径不同于基于来自所述加载指令的结果的路径时，由所述分支重引导电路在所述管线电路中将所述分支指令重引导到所述路径；并且

[0125] 当所述预测路径不同于基于来自所述加载指令的结果的所述路径时，由所述执行电路对于基于来自所述加载指令的结果的所述路径执行所述分支指令。

[0126] 示例18.如示例17所述的非暂态机器可读介质，其中，对于所述写回进行检查包括：在与所述结果的最终存储目的地分离的加载值表格中对于所述结果的写回进行检查。

[0127] 示例19.如示例18所述的非暂态机器可读介质，所述方法还包括：

[0128] 在由所述管线电路接收的指令是所述加载指令时，由所述分支重引导电路为所述加载值表格中用于所述加载指令的结果的条目指派索引值；并且

[0129] 使得所述索引值作为所述分支指令的字段被发送到所述管线电路中。

[0130] 示例20.如示例19所述的非暂态机器可读介质，所述方法还包括：在由所述管线电

路接收的指令是所述加载指令时，由所述分支重引导电路利用所述索引值来更新加载依从分支表格中用于所述分支指令的条目。

[0131] 示例21.如示例19所述的非暂态机器可读介质，所述方法还包括：由所述分支重引导电路为也依从于来自所述加载指令的结果的第二分支指令指派所述索引值。

[0132] 示例22.如示例17所述的非暂态机器可读介质，其中，所述分支指令的重引导发生在所述管线电路的分配阶段，该分配阶段指派所述执行电路来执行所述分支指令。

[0133] 示例23.如示例22所述的非暂态机器可读介质，其中，所述分支指令的重引导包括：从直到所述分配阶段为止的所述管线电路对于所述预测路径冲刷所述分支指令的数据。

[0134] 示例24.如示例17所述的非暂态机器可读介质，所述方法还包括：利用与包括所述管线电路的执行电路的执行阶段分离的电路来执行一个或多个操作以确定所述预测路径是否不同于基于来自所述加载指令的结果的路径。

[0135] 示例25.一种系统，包括：

[0136] 存储器，用于存储分支指令和加载指令；以及

[0137] 处理器核心，与所述存储器耦合，所述处理器核心包括：

[0138] 管线电路，其包括解码器来将指令解码成经解码的指令并且包括执行电路来执行经解码的指令，

[0139] 分支预测器电路，用于为所述分支指令生成预测路径，以及

[0140] 分支重引导电路，用于：对于依从于来自所述加载指令的结果的分支指令，检查由所述管线电路接收的指令是否是所述加载指令，并且当由所述管线电路接收的指令是所述加载指令时，对于在利用所述解码器对所述分支指令的解码和利用所述执行电路对所述分支指令的执行之间对来自所述加载指令的结果的写回进行检查，并且当所述预测路径不同于基于来自所述加载指令的结果的路径时，在所述管线电路中将所述分支指令重引导到所述路径并且使得对于基于来自所述加载指令的结果的所述路径执行所述分支指令。

[0141] 示例26.如示例25所述的系统，其中，所述分支重引导电路在与所述结果的最终存储目的地分离的加载值表格中对于所述结果的写回进行检查。

[0142] 示例27.如示例26所述的系统，其中，在由所述管线电路接收的指令是所述加载指令时，所述分支重引导电路为所述加载值表格中用于所述加载指令的结果的条目指派索引值，并且使得所述索引值作为所述分支指令的字段被发送到所述管线电路中。

[0143] 示例28.如示例27所述的系统，其中，在由所述管线电路接收的指令是所述加载指令时，所述分支重引导电路利用所述索引值来更新加载依从分支表格中用于所述分支指令的条目。

[0144] 示例29.如示例27所述的系统，其中，所述分支重引导电路为也依从于来自所述加载指令的结果的第二分支指令指派所述索引值。

[0145] 示例30.如示例25所述的系统，其中，所述分支指令的重引导发生在所述管线电路的分配阶段，该分配阶段指派所述执行电路来执行所述分支指令。

[0146] 示例31.如示例30所述的系统，其中，所述分支指令的重引导包括：从直到所述分配阶段为止的所述管线电路对于所述预测路径冲刷所述分支指令的数据。

[0147] 示例32.如示例25所述的系统，其中，所述处理器核心还包括：与包括所述管线电

路的执行电路的执行阶段分离的电路，用于执行一个或多个操作以确定所述预测路径是否不同于基于来自所述加载指令的结果的路径。

[0148] 在另外一个实施例中，一种装置包括存储代码的数据存储装置，所述代码当被硬件处理器执行时使得所述硬件处理器执行本文公开的任何方法。一种装置可如详细描述中所描述。一种方法可如详细描述中所描述。

[0149] 指令集可包括有条件分支指令。指令集可包括一个或多个指令格式。给定的指令格式可定义各种字段(例如，比特的数目、比特的位置)来指定要执行的操作(例如，操作码)和要在其上执行该操作的(一个或多个)操作对象和/或其他(一个或多个)数据字段(例如，掩码)，等等。一些指令格式通过指令模板(或子格式)的定义被进一步分解。例如，给定的指令格式的指令模板可被定义为具有该指令格式的字段的不同子集(包括的字段通常是按相同顺序的，但至少一些可具有不同的比特位置，因为包括的字段更少)和/或被定义为不同地解读给定的字段。从而，ISA的每个指令被利用给定的指令格式来表达(并且如果定义了的话，被以该指令格式的指令模板中的给定一个来表达)并且包括用于指定操作和操作对象的字段。例如，示范性ADD指令具有特定的操作码和指令格式，该指令格式包括操作码字段来指定该操作码并且包括操作对象字段来选择操作对象(源1/目标和源2)；并且此ADD指令在指令流中的出现在选择特定操作对象的操作对象字段中将具有特定内容。被称为高级向量扩展(Advanced Vector Extensions，AVX)(AVX1和AVX2)并且使用向量扩展(Vector Extensions，VEX)编码方案的一组SIMD扩展已被发布和/或发表(例如，参见2018年5月发布的Intel®64和IA-32体系结构软件开发者指南以及参见2018年5月发布的Intel®体系结构指令集扩展编程参考)。

[0150] 示范性指令格式

[0151] 本文描述的(一个或多个)指令的实施例可按不同的格式实现。此外，下文详述了示范性系统、体系结构和管线。(一个或多个)指令的实施例可被在这种系统、体系结构和管线上执行，但不限于详述的那些。

[0152] 通用向量友好指令格式

[0153] 向量友好指令格式是适合于向量指令的指令格式(例如，有某些特定于向量操作的字段)。虽然描述了其中通过向量友好指令格式支持向量和标量操作两者的实施例，但替换实施例只使用向量友好指令格式的向量操作。

[0154] 图12A-图12B是根据本公开的实施例图示出通用向量友好指令格式及其指令模板的框图。图12A是根据本公开的实施例图示出通用向量友好指令格式及其类别A指令模板的框图；而图12B是根据本公开的实施例图示出通用向量友好指令格式及其类别B指令模板的框图。具体而言，对于通用向量友好指令格式1200定义了类别A和类别B指令模板，这两个指令模板都包括无存储器访问1205指令模板和存储器访问1220指令模板。向量友好指令格式的上下文中的术语“通用”指的是该指令格式不被绑定到任何特定的指令集。

[0155] 虽然将描述其中向量友好指令格式支持以下所列项的本公开的实施例：64字节向量操作对象长度(或大小)，具有32比特(4字节)或64比特(8字节)数据元素宽度(或大小)(从而，64字节向量由16个双字大小元素或者8个四字大小元素构成)；64字节向量操作对象长度(或大小)，具有16比特(2字节)或8比特(1字节)数据元素宽度(或大小)；32字节向量操作对象长度(或大小)，具有32比特(4字节)、64比特(8字节)、16比特(2字节)或者8比特(1字

节)数据元素宽度(或大小);以及16字节向量操作对象长度(或大小),具有32比特(4字节)、64比特(8字节)、16比特(2字节)或者8比特(1字节)数据元素宽度(或大小);但替换实施例可支持具有更多、更少或不同的数据元素宽度(例如,128比特(16字节)数据元素宽度)的更多、更少和/或不同的向量操作对象大小(例如,256字节向量操作对象)。

[0156] 图12A中的类别A指令模板包括:1)在无存储器访问1205指令模板内,示出了无存储器访问、完全舍入控制型操作1210指令模板和无存储器访问、数据变换型操作1215指令模板;并且2)在存储器访问1220指令模板内,示出了存储器访问、暂态1225指令模板和存储器访问、非暂态1230指令模板。图12B中的类别B指令模板包括:1)在无存储器访问1205指令模板内,示出了无存储器访问、写入掩码控制、部分舍入控制型操作1212指令模板和无存储器访问、写入掩码控制、vsize型操作1217指令模板;并且2)在存储器访问1220指令模板内,示出了存储器访问、写入掩码控制1227指令模板。

[0157] 通用向量友好指令格式1200包括下面按图12A-图12B中所示的顺序列出的以下字段。

[0158] 格式字段1240-此字段中的特定值(指令格式识别符值)唯一地识别向量友好指令格式,从而识别采取向量友好指令格式的指令在指令流中的出现。这样,此字段是可选的,因为它对于只具有通用向量友好指令格式的指令集是不需要的。

[0159] 基本操作字段1242-其内容区分不同的基本操作。

[0160] 寄存器索引字段1244-其内容直接地或者通过地址生成指定源和目标操作对象的位置,无论它们在寄存器中还是在存储器中。这些包括充分数目的比特来从PxQ(例如,32x512、16x128、32x1024、64x1024)寄存器文件中选择N个寄存器。虽然在一个实施例中N可以是最多达三个源和一个目标寄存器,但替换实施例可支持更多或更少的源和目标寄存器(例如,可支持最多达两个源,其中这些源之一也充当目标,可支持最多达三个源,其中这些源之一也充当目标,可支持最多达两个源和一个目标)。

[0161] 修饰字段1246-其内容区分通用向量指令格式中的指定存储器访问的指令与那些不指定存储器访问的指令的出现;也就是说,区分无存储器访问1205指令模板和存储器访问1220指令模板。存储器访问操作读取和/或写入到存储器层次体系(在一些情况下利用寄存器中的值指定源和/或目标地址),而非存储器访问操作不读取和/或写入存储器层次体系(例如,源和目标是寄存器)。虽然在一个实施例中这个字段也在三个不同方式之间选择来执行存储器地址计算,但替换实施例可支持更多、更少或不同的方式来执行存储器地址计算。

[0162] 增强操作字段1250-其内容区分除了基本操作以外还要执行多种不同操作中的哪一种。此字段是依情境而定的。在本公开的一个实施例中,此字段被划分成类别字段1268、阿尔法字段1252和贝塔字段1254。增强操作字段1250允许了在单个指令而不是2、3或4个指令中执行共同操作群组。

[0163] 缩放比例字段1260-其内容允许了缩放索引字段的内容以进行存储器地址生成(例如,对于使用$2^{缩放比例}$*索引+基址的地址生成)。

[0164] 位移字段1262A-其内容被用作存储器地址生成的一部分(例如,对于使用$2^{缩放比例}$*索引+基址+位移的地址生成)。

[0165] 位移因子字段1262B(注意将位移字段1262A并列在位移因子字段1262B的正上方

表明一者或另一者被使用)-其内容被用作地址生成的一部分;其指定要被存储器访问的大小(N)缩放的位移因子-其中N是存储器访问中的字节的数目(例如,对于使用$2^{缩放比例}$*索引+基址+缩放的位移的地址生成)。冗余低阶比特被忽略,并且因此,位移因子字段的内容被乘以存储器操作对象总大小(N)以便生成要被用于计算有效地址的最终位移。N的值由处理器硬件在运行时基于完整操作码字段1274(本文中稍后描述)和数据操纵字段1254C来确定。位移字段1262A和位移因子字段1262B是可选的,因为它们不被用于无存储器访问1205指令模板,和/或不同的实施例可只实现两者中的一者或者两者都不实现。

[0166] 数据元素宽度字段1264-其内容区分若干个数据元素宽度中的哪一个将被使用(在一些实施例中是对于所有指令;在其他实施例中只对于指令中的一些)。此字段是可选的,因为如果只支持一个数据元素宽度和/或利用操作码的某个方面来支持数据元素宽度则不需要它。

[0167] 写入掩码字段1270-其内容基于每个数据元素位置控制目标向量操作对象中的该数据元素位置是否反映基本操作和增强操作的结果。类别A指令模板支持合并-写入掩蔽,而类别B指令模板支持合并-写入掩蔽和归零-写入掩蔽两者。当合并时,向量掩码允许了目标中的任何元素集合被保护免于任何操作(由基本操作和增强操作指定)的执行期间的更新;在其他的一个实施例中,保留目标的相应的掩码比特具有0的每个元素的旧值。与之不同,归零向量掩码允许了目标中的任何元素集合在任何操作(由基本操作和增强操作指定)的执行期间被归零;在一个实施例中,目标的元素在相应的掩码比特具有0值时被设置到0。此功能的子集是控制被执行的操作的向量长度(即,被修改的元素的跨度,从第一个到最后一个)的能力;然而,被修改的元素不是必须要连续。从而,写入掩码字段1270允许了部分向量操作,包括加载、存储、算术、逻辑等等。虽然描述了其中写入掩码字段1270的内容选择若干个写入掩码寄存器中包含要使用的写入掩码的那一个(并且从而写入掩码字段1270的内容间接识别要执行的该掩蔽)的本公开实施例,但替换实施例作为替代或附加允许掩码写入字段1270的内容直接指定要执行的掩蔽。

[0168] 即时字段1272-其内容允许对即时(immediate)的指定。此字段是可选的,因为在不支持即时的通用向量友好格式的实现方式中其不存在并且在不使用即时的指令中其不存在。

[0169] 类别字段1268-其内容区分指令的不同类别。参考图12A-图12B,此字段的内容在类别A和类别B指令之间进行选择。在图12A-图12B中,圆角方形用于指示特定的值存在于一字段中(例如,图12A-图12B中分别用于类别字段1268的类别A 1268A和类别B 1268B)。

[0170] 类别A的指令模板

[0171] 在类别A的非存储器访问1205指令模板的情况下,阿尔法字段1252被解读为RS字段1252A,其内容区分不同的增强操作类型中的哪一个要被执行(例如,对于无存储器访问舍入型操作1210和无存储器访问数据变换型操作1215指令模板分别指定舍入1252A.1和数据变换1252A.2),而贝塔字段1254区分指定类型的操作中的哪一个要被执行。在无存储器访问1205指令模板中,缩放比例字段1260、位移字段1262A和位移缩放比例字段1262B不存在。

[0172] 无存储器访问指令模板-完全舍入控制型操作

[0173] 在无存储器访问完全舍入控制型操作1210指令模板中,贝塔字段1254被解读为舍

入控制字段1254A,其(一个或多个)内容提供静态舍入。虽然在本公开的描述实施例中舍入控制字段1254A包括抑制所有浮点异常(suppress all floating point exceptions,SAE)字段1256和舍入操作控制字段1258,但替换实施例可支持可将这两个概念都编码到同一字段中或者可只具有这些概念/字段中的一者或另一者(例如,可只具有舍入操作控制字段1258)。

[0174] SAE字段1256-其内容区分是否禁用异常事件报告;当SAE字段1256的内容指示抑制被使能时,给定的指令不报告任何种类的浮点异常标志并且不引发任何浮点异常处理程序。

[0175] 舍入操作控制字段1258-其内容区分要执行一组舍入操作中的哪一个(例如,向上舍入、向下舍入、朝零舍入和朝最近舍入)。从而,舍入操作控制字段1258允许了基于每个指令改变舍入模式。在处理器包括用于指定舍入模式的控制寄存器的本公开的一个实施例中,舍入操作控制字段1250的内容推翻该寄存器值。

[0176] 无存储器访问指令模板-数据变换型操作

[0177] 在无存储器访问数据变换型操作1215指令模板中,贝塔字段1254被解读为数据变换字段1254B,其内容区分若干个数据变换中的哪一个要被执行(例如,无数据变换、调配(swizzle)、广播)。

[0178] 在类别A的存储器访问1220指令模板的情况下,阿尔法字段1252被解读为逐出提示字段1252B,其内容区分要使用逐出提示中的哪一个(在图12A中,对于存储器访问暂态1225指令模板和存储器访问非暂态1230指令模板分别指定暂态1252B.1和非暂态1252B.2),而贝塔字段1254被解读为数据操纵字段1254C,其内容区分若干个数据操纵操作(也称为基元)中的哪一个要被执行(例如,无操纵;广播;源的向上转换;以及目标的向下转换)。存储器访问1220指令模板包括缩放比例字段1260,并且可选地包括位移字段1262A或者位移缩放比例字段1262B。

[0179] 向量存储器指令执行从存储器的向量加载和向存储器的向量存储,带有转换支持。与常规向量指令一样,向量存储器指令以按数据元素的方式从/向存储器传送数据,其中被实际传送的元素由被选择为写入掩码的向量掩码的内容来规定。

[0180] 存储器访问指令模板-暂态

[0181] 暂态数据是可能很快就被再使用、快到足以受益于缓存的数据。然而,这是一个提示,并且不同的处理器可按不同的方式实现它,包括完全忽略该提示。

[0182] 存储器访问指令模板-非暂态

[0183] 非暂态数据是这样的数据:该数据不太可能快到足以受益于第1级缓存中的缓存地被再使用,并且应当被赋予逐出优先级。然而,这是一个提示,并且不同的处理器可按不同的方式实现它,包括完全忽略该提示。

[0184] 类别B的指令模板

[0185] 在类别B的指令模板的情况下,阿尔法字段1252被解读为写入掩码控制(Z)字段1252C,其内容区分由写入掩码字段1270控制的写入掩蔽应当是合并还是归零。

[0186] 在类别B的非存储器访问1205指令模板的情况下,贝塔字段1254的一部分被解读为RL字段1257A,其内容区分不同的增强操作类型中的哪一个要被执行(例如,对于无存储器访问、写入掩码控制、部分舍入控制型操作1212指令模板和无存储器访问、写入掩码控

制、VSIZE型操作1217指令模板分别指定舍入1257A.1和向量长度(VSIZE)1257A.2)，而贝塔字段1254的其余部分区分指定类型的操作中的哪一个要被执行。在无存储器访问1205指令模板中，缩放比例字段1260、位移字段1262A和位移缩放比例字段1262B不存在。

[0187] 在无存储器访问、写入掩码控制、部分舍入控制型操作1210指令模板中，贝塔字段1254的其余部分被解读为舍入操作字段1259A并且异常事件报告被禁用(给定的指令不报告任何种类的浮点异常标志并且不引发任何浮点异常处理程序)。

[0188] 舍入操作控制字段1259A-正如舍入操作控制字段1258一样，其内容区分要执行一组舍入操作中的哪一个(例如，向上舍入、向下舍入、朝零舍入和朝最近舍入)。从而，舍入操作控制字段1259A允许了基于每个指令改变舍入模式。在处理器包括用于指定舍入模式的控制寄存器的本公开的一个实施例中，舍入操作控制字段1250的内容推翻该寄存器值。

[0189] 在无存储器访问、写入掩码控制、VSIZE型操作1217指令模板中，贝塔字段1254的其余部分被解读为向量长度字段1259B，其内容区分要在若干个数据向量长度中的哪一个上执行(例如，128、256或512字节)。

[0190] 在类别B的存储器访问1220指令模板的情况下，贝塔字段1254的一部分被解读为广播字段1257B，其内容区分是否要执行广播型数据操纵操作，而贝塔字段1254的其余部分被解读为向量长度字段1259B。存储器访问1220指令模板包括缩放比例字段1260，并且可选地包括位移字段1262A或者位移缩放比例字段1262B。

[0191] 对于通用向量友好指令格式1200，完整操作码字段1274被示为包括格式字段1240、基本操作字段1242和数据元素宽度字段1264。虽然示出了其中完整操作码字段1274包括所有这些字段的一个实施例，但完整操作码字段1274在不支持所有这些字段的实施例中只包括这些字段中的一些。完整操作码字段1274提供操作代码(操作码)。

[0192] 增强操作字段1250、数据元素宽度字段1264和写入掩码字段1270允许了在通用向量友好指令格式中基于每个指令来指定这些特征。

[0193] 写入掩码字段和数据元素宽度字段的组合创建了类型化指令，因为它们允许基于不同的数据元素宽度来应用掩码。

[0194] 在类别A和类别B内找到的各种指令模板在不同的情形中是有益的。在本公开的一些实施例中，不同的处理器或处理器内的不同核心可只支持类别A、只支持类别B或者支持两个类别。例如，打算用于通用计算的高性能通用乱序核心可只支持类别B，打算主要用于图形和/或科学(吞吐量)计算的核心可只支持类别A，并且打算用于两者的核心可支持这两者(当然，具有来自两个类别的模板和指令的某种混合、但不具有来自两个类别的所有模板和指令的核心是在本公开的范围内的)。另外，单个处理器可包括多个核心，所有这些核心都支持相同类别或者其中不同的核心支持不同的类别。例如，在具有分开的图形和通用核心的处理器中，打算主要用于图形和/或科学计算的图形核心之一可只支持类别A，而通用核心中的一个或多个可以是只支持类别B的打算用于通用计算的具有乱序执行和寄存器重命名的高性能通用核心。不具有单独的图形核心的另一处理器可包括支持类别A和类别B两者的一个或多个通用有序或乱序核心。当然，在本公开的不同实施例中，来自一个类别的特征也可被实现在另一类别中。以高级别语言编写的程序将被置于(例如，被即时编译或静态编译到)多种不同的可执行形式中，包括：1)只具有由目标处理器支持的(一个或多个)类别的指令以便执行的形式；或者2)具有利用所有类别的指令的不同组合编写的替换例程并且

具有基于当前执行代码的处理器所支持的指令来选择要执行的例程的控制流程代码的形式。

[0195] 示范性特定向量友好指令格式

[0196] 图13A是根据本公开的实施例图示出示范性特定向量友好指令格式的框图。图13A示出了在如下意义上特定的特定向量友好指令格式1300:其指定字段的位置、大小、解读和顺序,以及这些字段中的一些的值。特定向量友好指令格式1300可被用于扩展x86指令集,从而字段中的一些与现有的x86指令集及其扩展(例如,AVX)中使用的那些相似或相同。此格式与带有扩展的现有x86指令集的前缀编码字段、真实操作码字节字段、MOD R/M字段、SIB字段、位移字段和即时字段保持一致。图示出了来自图13A的字段所映射到的来自图12A和图12B的字段。

[0197] 应当理解,虽然出于说明目的在通用向量友好指令格式1200的情境中参考特定向量友好指令格式1300描述了本公开的实施例,但除非有声明,否则本公开不限于特定向量友好指令格式1300。例如,通用向量友好指令格式1200对于各种字段设想了多种可能的大小,而特定向量友好指令格式1300被示为具有特定大小的字段。作为具体示例,虽然数据元素宽度字段1264在特定向量友好指令格式1300中被示为一比特字段,但本公开不限于此(也就是说,通用向量友好指令格式1200设想了数据元素宽度字段1264的其他大小)。

[0198] 通用向量友好指令格式1200包括按图13A中所示的顺序的下面列出的以下字段。

[0199] EVEX前缀(字节0-3)1302-被编码为四字节形式。

[0200] 格式字段1240(EVEX字节0,比特[7:0])-第一字节(EVEX字节0)是格式字段1240并且其包含0x62(在本公开的一个实施例中用于区分向量友好指令格式的唯一值)。

[0201] 第二至第四字节(EVEX字节1-3)包括提供特定能力的若干个比特字段。

[0202] REX字段1305(EVEX字节1,比特[7-5])-由EVEX.R比特字段(EVEX字节1,比特[7]-R)、EVEX.X比特字段(EVEX字节1,比特[6]-X)和1257BEX字节1,比特[5]-B)构成。EVEX.R、EVEX.X和EVEX.B比特字段提供与相应的VEX比特字段相同的功能,并且被利用反码形式来编码,即ZMM0被编码为1111B,ZMM15被编码为0000B。指令的其他字段如本领域中已知的那样对寄存器索引的较低三个比特编码(rrr、xxx和bbb),从而Rrrr、Xxxx和Bbbb可通过添加EVEX.R、EVEX.X和EVEX.B来形成。

[0203] REX'字段1210-这是REX'字段1210的第一部分并且是用于对扩展32寄存器集合的高16或低16编码的EVEX.R'比特字段(EVEX字节1,比特[4]-R')。在本公开的一个实施例中,此比特以及如下所示的其他比特被以比特反转格式来存储以与BOUND指令相区分(在公知的x86 32比特模式中),BOUND指令的真实操作码字节是62,但不在MOD R/M字段(下文描述)中接受MOD字段中的11的值;本公开的替换实施例不以反转格式存储这个比特和下面指示的其他比特。值1被用于对低16寄存器编码。换言之,R'Rrrr是通过组合EVEX.R'、EVEX.R和来自其他字段的其他RRR形成的。

[0204] 操作码映射字段1315(EVEX字节1,比特[3:0]-mmmm)-其内容编码了暗示的主导操作码字节(0F、0F 38或0F 3)。

[0205] 数据元素宽度字段1264(EVEX字节2,比特[7]-W)-由符号EVEX.W表示。EVEX.W被用于定义数据类型的粒度(大小)(32比特数据元素或64比特数据元素)。

[0206] EVEX.vvvv 1320(EVEX字节2,比特[6:3]-vvvv)-EVEX.vvvv的作用可包括以下的:

1)EVEX.vvvv编码了以反转(反码)形式指定的第一源寄存器操作对象,并且对于具有2个或更多个源操作对象的指令是有效的;2)EVEX.vvvv编码了对于某些向量移位以反码形式指定的目标寄存器操作对象;或者3)EVEX.vvvv不编码任何操作对象,该字段被保留并且应当包含1111b。从而,EVEX.vvvv字段1320编码了以反转(反码)形式存储的第一源寄存器指定符的4个低阶比特。取决于指令,一额外的不同EVEX比特字段被用于将指定符大小扩展到32寄存器。

[0207] EVEX.U 1268类别字段(EVEX字节2,比特[2]-U)-如果EVEX.U＝0,则其指示类别A或EVEX.U0;如果EVEX.U＝1,则其指示类别B或者EVEX.U1。

[0208] 前缀编码字段1325(EVEX字节2,比特[1:0]-pp)-为基本操作字段提供额外比特。除了对于采取EVEX前缀格式的传统SSE指令提供支持以外,这还具有使SIMD前缀紧缩的益处(EVEX前缀只要求2个比特,而不是要求一字节来表达SIMD前缀)。在一个实施例中,为了支持采取传统格式和采取EVEX前缀格式两者的使用SIMD前缀(66H、F2H、F3H)的传统SSE指令,这些传统SIMD前缀被编码到SIMD前缀编码字段中;并且在运行时被扩展成传统SIMD前缀,然后才被提供到解码器的PLA(因此PLA可执行这些传统指令的传统和EVEX格式两者,而无需修改)。虽然更新的指令可直接使用EVEX前缀编码字段的内容作为操作码扩展,但某些实施例为了一致性以类似的方式扩展,但允许这些传统SIMD前缀指定不同的含义。替换实施例可重设计PLA来支持2比特SIMD前缀编码,从而不要求扩展。

[0209] 阿尔法字段1252(EVEX字节3,比特[7]-EH;也称为EVEX.EH、EVEX.rs、EVEX.RL、EVEX.写入掩码控制以及EVEX.N;也用α来图示)-如前所述,此字段是依情境而定的。

[0210] 贝塔字段1254(EVEX字节3,比特[6:4]-SSS;也称为EVEX.s_{2-0}、EVEX.r_{2-0}、EVEX.rr1、EVEX.LL0、EVEX.LLB;也用βββ来图示)-如前所述,此字段是依情境而定的。

[0211] REX'字段1210-这是REX'字段的剩余部分并且是可用于对扩展32寄存器集合的高16或低16编码的EVEX.V'比特字段(EVEX字节3,比特[3]-V')。此比特被以比特反转格式来存储。值1被用于对低16寄存器编码。换言之,V'VVVV是通过组合EVEX.V'、EVEX.vvvv形成的。

[0212] 写入掩码字段1270(EVEX字节3,比特[2:0]-kkk)-其内容指定如前所述的写入掩码寄存器中的寄存器的索引。在本公开的一个实施例中,特定值EVEX.kkk＝000具有暗示对于特定指令没有使用写入掩码的特殊行为(这可通过多种方式来实现,包括使用被硬连线到全一的写入掩码或者绕过掩蔽硬件的硬件)。

[0213] 真实操作码字段1330(字节4)也被称为操作码字节。操作码的一部分在此字段中指定。

[0214] MOD R/M字段1340(字节5)包括MOD字段1342、Reg字段1344和R/M字段1346。如前所述,MOD字段1342的内容区分存储器访问和非存储器访问操作。Reg字段1344的作用可被总结成两个情形:编码目标寄存器操作对象或者源寄存器操作对象,或者被作为操作码扩展来对待并且不被用于编码任何指令操作对象。R/M字段1346的作用可包括以下的:编码引用存储器地址的指令操作对象,或者编码目标寄存器操作对象或源寄存器操作对象。

[0215] 缩放比例、索引、基数(Scale,Index,Base,SIB)字节(字节6)-如前所述,缩放比例字段1250的内容被用于存储器地址生成。SIB.xxx1354和SIB.bbb 1356-先前已对于寄存器索引Xxxx和Bbbb提及了这些字段的内容。

[0216] 位移字段1262A(字节7-10)-当MOD字段1342包含10时,字节7-10是位移字段1262A,并且其工作方式与传统32比特位移(disp32)相同并且在字节粒度上工作。

[0217] 位移因子字段1262B(字节7)-当MOD字段1342包含01时,字节7是位移因子字段1262B。此字段的位置与传统x86指令集8比特位移(disp8)的相同,其在字节粒度上工作。由于disp8被符号扩展,所以其只能在-128和127字节偏移量之间寻址;就64字节缓存线而言,disp8使用8个比特,这8个比特可被设置到仅四个真正有用的值-128、-64、0和64;由于经常需要更大的范围,所以使用disp32;然而,disp32要求4个字节。与disp8和disp32不同,位移因子字段1262B是对disp8的重解读;当使用位移因子字段1262B时,实际位移由位移因子字段的内容乘以存储器操作对象访问的大小(N)来确定。这种类型的位移被称为disp8*N。这减小了平均指令长度(单个字节被用于位移,但具有大得多的范围)。这种压缩的位移是基于如下假设的:有效位移是存储器访问的粒度的倍数,并且因此,地址偏移量的冗余低阶比特不需要被编码。换言之,位移因子字段1262B代替了传统x86指令集8比特位移。从而,位移因子字段1262B被按与x86指令集8比特位移相同的方式编码(因此在ModRM/SIB编码规则中没有变化),唯一例外是disp8被超载到disp8*N。换言之,在编码规则或编码长度中没有变化,而只在硬件对位移值的解读中有变化(硬件需要按存储器操作对象的大小来缩放位移以获得按字节地址偏移量)。即时字段1272如前所述那样操作。

[0218] 完整操作码字段

[0219] 图13B是根据本公开的一个实施例图示出构成完整操作码字段1274的特定向量友好指令格式1300的字段的框图。具体而言,完整操作码字段1274包括格式字段1240、基本操作字段1242和数据元素宽度(W)字段1264。基本操作字段1242包括前缀编码字段1325、操作码映射字段1315和真实操作码字段1330。

[0220] 寄存器索引字段

[0221] 图13C是根据本公开的一个实施例图示出构成寄存器索引字段1244的特定向量友好指令格式1300的字段的框图。具体而言,寄存器索引字段1244包括REX字段1305、REX'字段1310、MODR/M.reg字段1344、MODR/M.r/m字段1346、VVVV字段1320、xxx字段1354和bbb字段1356。

[0222] 增强操作字段

[0223] 图13D是根据本公开的一个实施例图示出构成增强操作字段1250的特定向量友好指令格式1300的字段的框图。当类别(U)字段1268包含0时,其表示EVEX.U0(类别A 1268A);当其包含1时,其表示EVEX.U1(类别B 1268B)。当U=0并且MOD字段1342包含11时(表示无存储器访问操作),阿尔法字段1252(EVEX字节3,比特[7]-EH)被解读为rs字段1252A。当rs字段1252A包含1时(舍入1252A.1),贝塔字段1254(EVEX字节3,比特[6:4]-SSS)被解读为舍入控制字段1254A。舍入控制字段1254A包括一比特SAE字段1256和两比特舍入操作字段1258。当rs字段1252A包含0时(数据变换1252A.2),贝塔字段1254(EVEX字节3,比特[6:4]-SSS)被解读为三比特数据变换字段1254B。当U=0并且MOD字段1342包含00、01或10时(表示存储器访问操作),阿尔法字段1252(EVEX字节3,比特[7]-EH)被解读为逐出提示(eviction hint,EH)字段1252B并且贝塔字段1254(EVEX字节3,比特[6:4]-SSS)被解读为三比特数据操纵字段1254C。

[0224] 当U=1时,阿尔法字段1252(EVEX字节3,比特[7]-EH)被解读为写入掩码控制(Z)

字段1252C。当U＝1并且MOD字段1342包含11时(表示无存储器访问操作)，贝塔字段1254的一部分(EVEX字节3，比特[4]-S_0)被解读为RL字段1257A；当其包含1(舍入1257A.1)时，贝塔字段1254的其余部分(EVEX字节3，比特[6-5]-S_{2-1})被解读为舍入操作字段1259A，而当RL字段1257A包含0(VSIZE 1257.A2)时，贝塔字段1254的其余部分(EVEX字节3，比特[6-5]-S_{2-1})被解读为向量长度字段1259B(EVEX字节3，比特[6-5]-L_{1-0})。当U＝1并且MOD字段1342包含00、01或10时(表示存储器访问操作)，贝塔字段1254(EVEX字节3，比特[6:4]-SSS)被解读为向量长度字段1259B(EVEX字节3，比特[6-5]-L_{1-0})和广播字段1257B(EVEX字节3，比特[4]-B)。

[0225] 示范性寄存器体系结构

[0226] 图14是根据本公开的一个实施例的寄存器体系结构1400的框图。在图示的实施例中，存在32个512比特宽的向量寄存器1410；这些寄存器被称为zmm0至zmm31。低16zmm寄存器的低阶256比特被覆盖在寄存器ymm0-16上。低16zmm寄存器的低阶128比特(ymm寄存器的低阶128比特)被覆盖在寄存器xmm0-15上。特定向量友好指令格式1300如以下表格中所示在这些覆盖的寄存器文件上操作。

[0227]

可调整向量长度	类别	操作	寄存器
不包括向量长度字段 1259B 的指令模板	A（图 12A；U=0）	1210, 1215, 1225, 1230	zmm 寄存器（向量长度是 64 字节）
	B（图 12B；U=1）	1212	zmm 寄存器（向量长度是 64 字节）
包括向量长度字段 1259B 的指令模板	B（图 12B；U=1）	1217, 1227	zmm、ymm 或 xmm 寄存器（向量长度是 64 字节、32 字节或 16 字节），取决于向量长度字段 1259B

[0228] 换言之，向量长度字段1259B在最大长度和一个或多个其他更短长度之间做出选择，其中每个这种更短长度是前一长度的一半；并且没有向量长度字段1259B的指令模板在最大向量长度上操作。另外，在一个实施例中，特定向量友好指令格式1300的类别B指令模板在紧缩或标量单/双精度浮点数据和紧缩或标量整数数据上操作。标量操作是在zmm/ymm/xmm寄存器中的最低阶数据元素位置上执行的操作；更高阶数据元素位置或者被保持与其在该指令之前相同，或者被归零，这取决于实施例。

[0229] 写入掩码寄存器1415-在图示的实施例中，有8个写入掩码寄存器(k0至k7)，每个的大小是64比特。在替换实施例中，写入掩码寄存器1415的大小是16比特。如前所述，在本公开的一个实施例中，向量掩码寄存器k0可被用作写入掩码；当通常将会指示k0的编码被用于写入掩码时，其选择硬连线的写入掩码0xFFFF，实际上对于该指令禁用了写入掩蔽。

[0230] 通用寄存器1425-在图示的实施例中，有十六个64比特通用寄存器，它们与现有的x86寻址模式一起被用于寻址存储器操作对象。这些寄存器被用名称RAX、RBX、RCX、RDX、

RBP、RSI、RDI、RSP以及R8至R15来引用。

[0231] 标量浮点堆栈寄存器文件(x87堆栈)1445,其上化名了MMX紧缩整数平坦寄存器文件1450-在图示的实施例中,x87堆栈是用于利用x87指令集扩展在32/64/80比特浮点数据上执行标量浮点操作的八元素堆栈;而MMX寄存器被用于在64比特紧缩整数数据上执行操作,以及为在MMX和XMM寄存器之间执行的一些操作保持操作对象。

[0232] 本公开的替换实施例可使用更宽或更窄的寄存器。此外,本公开的替换实施例可使用更多、更少或不同的寄存器文件和寄存器。

[0233] 示范性核心体系结构、处理器和计算机体系结构

[0234] 处理器核心可按不同的方式、为了不同的目的、在不同的处理器中实现。例如,这种核心的实现方式可包括:1)打算用于通用计算的通用有序核心;2)打算用于通用计算的高性能通用乱序核心;3)主要打算用于图形和/或科学(吞吐量)计算的专用核心。不同处理器的实现方式可包括:1)包括打算用于通用计算的一个或多个通用有序核心和/或打算用于通用计算的一个或多个通用乱序核心的CPU;以及2)包括主要打算用于图形和/或科学(吞吐量)的一个或多个专用核心的协处理器。这样的不同处理器导致不同的计算机系统体系结构,这些体系结构可包括:1)协处理器在与CPU分开的芯片上;2)协处理器在与CPU相同的封装中、分开的晶粒上;3)协处理器与CPU在同一晶粒上(在此情况下,这种协处理器有时被称为专用逻辑,例如集成图形和/或科学(吞吐量)逻辑,或者被称为专用核心);以及4)片上系统,其可在同一晶粒上包括描述的CPU(有时称为(一个或多个)应用核心或者(一个或多个)应用处理器)、上述的协处理器以及额外的功能。接下来描述示范性核心体系结构,然后是对示范性处理器和计算机体系结构的描述。

[0235] 示范性核心体系结构

[0236] 有序和乱序核心框图

[0237] 图15A是根据本公开的实施例图示出示范性有序管线和示范性寄存器重命名、乱序发出/执行管线两者的框图。图15B是根据本公开的实施例图示出要被包括在处理器中的有序体系结构核心的示范性实施例和示范性寄存器重命名、乱序发出/执行体系结构核心两者的框图。图15A-15B中的实线框图示了有序管线和有序核心,而虚线框的可选添加图示了寄存器重命名、乱序发出/执行管线和核心。考虑到有序方面是乱序方面的子集,将描述乱序方面。

[0238] 在图15A中,处理器管线1500包括取得阶段1502、长度解码阶段1504、解码阶段1506、分配阶段1508、重命名阶段1510、调度(也称为调遣或发出)阶段1512、寄存器读取/存储器读取阶段1514、执行阶段1516、写回/存储器写入阶段1518、异常处理阶段1522和提交阶段1524。

[0239] 图15B示出了处理器核心1590包括耦合到执行引擎单元1550的前端单元1530,并且两者都耦合到存储器单元1570。核心1590可以是精简指令集计算(reduced instruction set computing,RISC)核心、复杂指令集计算(complex instruction set computing,CISC)核心、超长指令字(very long instruction word,VLIW)核心或者混合或替换核心类型。作为另外一个选项,核心1590可以是专用核心,例如网络或通信核心、压缩引擎、协处理器核心、通用计算图形处理单元(general purpose computing graphics processing unit,GPGPU)核心、图形核心,等等。

[0240] 前端单元1530包括分支预测单元1532，其耦合到指令缓存单元1534，指令缓存单元1534耦合到指令转化后备缓冲器(translation lookaside buffer，TLB)1536，该TLB 1536耦合到指令取得单元1538，该指令取得单元1538耦合到解码单元1540。解码单元1540(或者解码器或解码器单元)可对指令(例如，宏指令)解码，并且生成一个或多个微操作、微代码入口点、微指令、其他指令或其他控制信号作为输出，这些微操作、微代码入口点、微指令、其他指令或其他控制信号是从原始指令解码来的，或者以其他方式反映原始指令，或者是从原始指令得出的。解码单元1540可利用各种不同的机制来实现。适当机制的示例包括但不限于查找表、硬件实现、可编程逻辑阵列(programmable logic array，PLA)、微代码只读存储器(read only memory，ROM)，等等。在一个实施例中，核心1590包括微代码ROM或其他介质，其为某些宏指令存储微代码(例如，在解码单元1540中或者以其他方式在前端单元1530内)。解码单元1540耦合到执行引擎单元1550中的重命名/分配器单元1552。

[0241] 执行引擎单元1550包括耦合到引退单元1554和一组一个或多个调度器单元1556的重命名/分配器单元1552。(一个或多个)调度器单元1556表示任何数目的不同调度器，包括预留站、中央指令窗口等等。(一个或多个)调度器单元1556耦合到(一个或多个)物理寄存器文件单元1558。物理寄存器文件单元1558的每一者表示一个或多个物理寄存器文件，这些物理寄存器文件中的不同物理寄存器文件存储一个或多个不同的数据类型，例如标量整数、标量浮点、紧缩整数、紧缩浮点、向量整数、向量浮点、状态(例如，作为要执行的下一指令的地址的指令指针)，等等。在一个实施例中，物理寄存器文件单元1558包括向量寄存器单元、写入掩码寄存器单元和标量寄存器单元。这些寄存器单元可提供体系结构式向量寄存器、向量掩码寄存器和通用寄存器。(一个或多个)物理寄存器文件单元1558与引退单元1554重叠以例示出可用来实现寄存器重命名和乱序执行的各种方式(例如，利用(一个或多个)重排序缓冲器和(一个或多个)引退寄存器文件；利用(一个或多个)未来文件、(一个或多个)历史缓冲器和(一个或多个)引退寄存器文件；利用寄存器图谱和寄存器的池；等等)。引退单元1554和(一个或多个)物理寄存器文件单元1558耦合到(一个或多个)执行集群1560。(一个或多个)执行集群1560包括一组一个或多个执行单元1562和一组一个或多个存储器访问单元1564。执行单元1562可在各种类型的数据(例如，标量浮点、紧缩整数、紧缩浮点、向量整数、向量浮点)上执行各种操作(例如，移位、加法、减法、乘法)。虽然一些实施例可包括专用于特定功能或功能集合的若干个执行单元，但其他实施例可只包括一个执行单元或者全都执行所有功能的多个执行单元。(一个或多个)调度器单元1556、(一个或多个)物理寄存器文件单元1558和(一个或多个)执行集群1560被示为可能是多个，因为某些实施例为某些类型的数据/操作创建单独的管线(例如，标量整数管线、标量浮点/紧缩整数/紧缩浮点/向量整数/向量浮点管线和/或存储器访问管线，它们各自具有其自己的调度器单元、物理寄存器文件单元和/或执行集群-并且在单独的存储器访问管线的情况下，实现了某些实施例，其中只有此管线的执行集群具有(一个或多个)存储器访问单元1564)。还应当理解，在使用分开管线的情况下，这些管线中的一个或多个可以是乱序发出/执行，其余的是有序的。

[0242] 存储器访问单元1564的集合耦合到存储器单元1570，存储器单元1570包括数据TLB单元1572，数据TLB单元1572耦合到数据缓存单元1574，数据缓存单元1574耦合到第2级(L2)缓存单元1576。在一个示范性实施例中，存储器访问单元1564可包括加载单元、存储地

址单元和存储数据单元,其中每一者耦合到存储器单元1570中的数据TLB单元1572。指令缓存单元1534进一步耦合到存储器单元1570中的第2级(L2)缓存单元1576。L2缓存单元1576耦合到一个或多个其他级别的缓存并且最终耦合到主存储器。

[0243] 作为示例,示范性寄存器重命名、乱序发出/执行核心体系结构可实现管线1500如下:1)指令取得1538执行取得和长度解码阶段1502和1504;2)解码单元1540执行解码阶段1506;3)重命名/分配器单元1552执行分配阶段1508和重命名阶段1510;4)(一个或多个)调度器单元1556执行调度阶段1512;5)(一个或多个)物理寄存器文件单元1558和存储器单元1570执行寄存器读取/存储器读取阶段1514;执行集群1560执行执行阶段1516;6)存储器单元1570和(一个或多个)物理寄存器文件单元1558执行写回/存储器写入阶段1518;7)在异常处理阶段1522中可涉及各种单元;并且8)引退单元1554和(一个或多个)物理寄存器文件单元1558执行提交阶段1524。

[0244] 核心1590可支持一个或多个指令集(例如,x86指令集(带有已随着更新版本添加的一些扩展);加州森尼维耳市的MIPS技术公司的MIPS指令集;加州森尼维耳市的ARM控股公司的ARM指令集(带有可选的额外扩展,例如NEON)),包括本文描述的(一个或多个)指令。在一个实施例中,核心1590包括逻辑来支持紧缩数据指令集扩展(例如,AVX1、AVX2),从而允许了被许多多媒体应用使用的操作被利用紧缩数据来执行。

[0245] 应当理解,核心可支持多线程处理(执行操作或线程的两个或更多个并行集合),并且可按多种方式来支持多线程处理,包括时间切片式多线程处理、同时多线程处理(其中单个物理核心为该物理核心在同时进行多线程处理的每个线程提供逻辑核心),或者这些的组合(例如,时间切片式取得和解码,然后是同时多线程处理,例如像Intel® Hyper-threading技术中那样)。

[0246] 虽然寄存器重命名是在乱序执行的情境中描述的,但应当理解寄存器重命名可用于有序体系结构中。虽然处理器的图示实施例还包括分开的指令和数据缓存单元1534/1574和共享的L2缓存单元1576,但替换实施例可对于指令和数据两者具有单个内部缓存,例如第1级(L1)内部缓存,或者多级别的内部缓存。在一些实施例中,系统可包括内部缓存和在核心和/或处理器外部的外部缓存的组合。或者,所有缓存都可在核心和/或处理器外部。

[0247] 具体示范性有序核心体系结构

[0248] 图16A-图16B图示出更具体的示范性有序核心体系结构的框图,该核心将是芯片中的若干个逻辑块(包括相同类型和/或不同类型的其他核心)之一。逻辑块通过高带宽互连网络(例如,环状网络)与某些固定功能逻辑、存储器I/O接口和其他必要I/O逻辑通信,这取决于应用。

[0249] 图16A是根据本公开的实施例的单个处理器核心及其与片上互连网络1602以及与第2级(L2)缓存1604的其本地子集的连接的框图。在一个实施例中,指令解码单元1600支持具有紧缩数据指令集扩展的x86指令集。L1缓存1606允许低时延访问以将存储器缓存到标量和向量单元中。虽然在一个实施例中(为了简化设计),标量单元1608和向量单元1610使用分开的寄存器集合(分别是标量寄存器1612和向量寄存器1614)并且在它们之间传送的数据被写入到存储器,然后被从第1级(L1)缓存1606读回,但本公开的替换实施例可使用不同的方案(例如,使用单个寄存器集合或者包括允许数据在两个寄存器文件之间传送而不

被写入和读回的通信路径)。

[0250] L2缓存的本地子集1604是全局L2缓存的一部分,全局L2缓存被划分成单独的本地子集,每个处理器核心有一个。每个处理器核心具有到其自己的L2缓存的本地子集1604的直接访问路径。处理器核心读取的数据被存储在其L2缓存子集1604中并且可被迅速访问,与其他处理器核心访问其自己的本地L2缓存子集并行。处理器核心写入的数据被存储在其自己的L2缓存子集1604中并且在必要时被从其他子集冲刷出。环状网络确保了共享数据的一致性。环状网络是双向的,以允许诸如处理器核心、L2缓存和其他逻辑块之类的代理在芯片内与彼此通信。每个环状数据路径在每个方向上是1012比特宽的。

[0251] 图16B是根据本公开的实施例的图16A中的处理器核心的一部分的扩展视图。图16B包括L1缓存1604的L1数据缓存1606A部分,以及关于向量单元1610和向量寄存器1614的更多细节。具体而言,向量单元1610是16宽向量处理单元(vector processing unit,VPU)(参见16宽ALU 1628),其执行整数、单精度浮点和双精度浮点指令中的一个或多个。VPU支持利用调配单元1620调配寄存器输入,利用数值转换单元1622A-B进行的数值转换,以及利用复制单元1624对存储器输入的复制。写入掩码寄存器1626允许断言结果向量写入。

[0252] 图17是根据本公开的实施例的可具有多于一个核心、可具有集成的存储器控制器并且可具有集成的图形的处理器1700的框图。图17中的实线框图示了具有单个核心1702A、系统代理1700和一组一个或多个总线控制器单元1716的处理器1700,而虚线框的可选添加图示了具有多个核心1702A-N、系统代理单元1700中的一组一个或多个集成存储器控制单元1714和专用逻辑1708的替换处理器1700。

[0253] 从而,处理器1700的不同实现方式可包括:1)其中专用逻辑1708是集成图形和/或科学(吞吐量)逻辑(其可包括一个或多个核心)并且核心1702A-N是一个或多个通用核心(例如,通用有序核心、通用乱序核心或者两者的组合)的CPU;2)其中核心1702A-N是主要打算用于图形和/或科学(吞吐量)的大量的专用核心的协处理器;以及3)其中核心1702A-N是大量的通用有序核心的协处理器。从而,处理器1700可以是通用处理器、协处理器或专用处理器,例如网络或通信处理器、压缩引擎、图形处理器、GPGPU(通用图形处理单元)、高吞吐量集成众核(many integrated core,MIC)协处理器(包括30或更多个核心)、嵌入式处理器,等等。处理器可实现在一个或多个芯片上。处理器1700可以是一个或多个基片的一部分和/或可利用若干个工艺技术中的任何一者实现在一个或多个基片上,这些技术例如是BiCMOS、CMOS或NMOS。

[0254] 存储器层次体系可包括核心内的一级或多级缓存、一组或一个或多个共享缓存单元1706以及耦合到该组集成存储器控制器单元1714的外部存储器(未示出)。该组共享缓存单元1706可包括一个或多个中间级别缓存,例如第2级(L2)、第3级(L3)、第4级(4)或其他级别的缓存,最后一级缓存(last level cache,LLC),和/或这些的组合。虽然在一个实施例中基于环的互连单元1712互连集成图形逻辑1708、该组共享缓存单元1706和系统代理单元1710/(一个或多个)集成存储器控制器单元1714,但替换实施例也可使用任何数目的公知技术来互连这种单元。在一个实施例中,在一个或多个缓存单元1706和核心1702A-N之间维持一致性。

[0255] 在一些实施例中,核心1702A-N中的一个或多个能够进行多线程处理。系统代理1710包括协调和操作核心1702A-N的那些组件。系统代理单元1710可包括例如功率控制单

元(power control unit,PCU)和显示单元。PCU可以是或者可以包括调控核心1702A-N和集成图形逻辑1708的功率状态所需要的逻辑和组件。显示单元用于驱动一个或多个外部连接的显示器。

[0256] 核心1702A-N就体系结构指令集而言可以是同质的或者异质的;也就是说,核心1702A-N中的两个或更多个可能够执行同一指令集,而其他的可只能够执行该指令集的子集或者不同的指令集。

[0257] 示范性计算机体系结构

[0258] 图18-图21是示范性计算机体系结构的框图。本领域中已知的用于膝上型计算机、桌面型计算机、手持PC、个人数字助理、工程工作站、服务器、网络装置、网络集线器、交换机、嵌入式处理器、数字信号处理器(digital signal processor,DSP)、图形装置、视频游戏装置、机顶盒、微控制器、蜂窝电话、便携式媒体播放器、手持装置和各种其他电子装置的其他系统设计和配置也是适当的。总之,能够包含本文公开的处理器和/或其他执行逻辑的许多种系统或电子装置一般是适当的。

[0259] 现在参考图18,其中示出了根据本公开的一个实施例的系统1800的框图。系统1800可包括一个或多个处理器1810、1815,它们耦合到控制器中枢1820。在一个实施例中,控制器中枢1820包括图形存储器控制器中枢(graphics memory controller hub,GMCH)1890和输入/输出中枢(Input/Output Hub,IOH)1850(它们可在分开的芯片上);GMCH 1890包括与存储器1840和协处理器1845耦合的存储器和图形控制器;IOH1850将输入/输出(I/O)装置1860耦合到GMCH 1890。或者,存储器和图形控制器的一者或两者被集成在处理器内(如本文所述),存储器1840和协处理器1845直接耦合到处理器1810,并且控制器中枢1820与IOH1850在单个芯片中。存储器1840例如可包括分支预测器代码1840A,以存储当被执行时使得处理器执行本公开的任何方法的代码。

[0260] 额外的处理器1815的可选性在图18中用虚线表示。每个处理器1810、1815可包括本文描述的处理核心中的一个或多个并且可以是处理器1700的某个版本。

[0261] 存储器1840可例如是动态随机访问存储器(dynamic random access memory,DRAM)、相变存储器(phase change memory,PCM)或者两者的组合。对于至少一个实施例,控制器中枢1820经由多点分支总线(例如前端总线(frontside bus,FSB))、点到点接口(例如Quickpath Interconnect(QPI))或者类似的连接1895与(一个或多个)处理器1810、1815通信。

[0262] 在一个实施例中,协处理器1845是专用处理器,例如高吞吐量MIC处理器、网络或通信处理器、压缩引擎、图形处理器、GPGPU、嵌入式处理器,等等。在一个实施例中,控制器中枢1820可包括集成的图形加速器。

[0263] 在物理资源1810、1815之间,就包括体系结构特性、微体系结构特性、热特性、功率消耗特性等等在内的价值度量的范围而言可以有多种差异。

[0264] 在一个实施例中,处理器1810执行控制一般类型的数据处理操作的指令。嵌入在这些指令内的可以是协处理器指令。处理器1810将这些协处理器指令识别为应当由附接的协处理器1845执行的类型。因此,处理器1810在协处理器总线或其他互连上向协处理器1845发出这些协处理器指令(或者表示协处理器指令的控制信号)。(一个或多个)协处理器1845接受和执行接收到的协处理器指令。

[0265] 现在参考图19,其中示出了根据本公开的实施例的第一更具体示范性系统1900的框图。如图19中所示,多处理器系统1900是点到点互连系统,并且包括经由点到点互连1950耦合的第一处理器1970和第二处理器1980。处理器1970和1980的每一者可以是处理器1700的某个版本。在本公开的一个实施例中,处理器1970和1980分别是处理器1810和1815,而协处理器1938是协处理器1845。在另一实施例中,处理器1970和1980分别是处理器1810和协处理器1845。

[0266] 处理器1970和1980被示为分别包括集成存储器控制器(integrated memory controller,IMC)单元1972和1982。处理器1970还包括点到点(P-P)接口1976和1978作为其总线控制器单元的一部分;类似地,第二处理器1980包括P-P接口1986和1988。处理器1970、1980可利用P-P接口电路1978、1988经由点到点(P-P)接口1950交换信息。如图19中所示,IMC 1972和1982将处理器耦合到各自的存储器,即存储器1932和存储器1934,存储器1932和存储器1934可以是在本地附接到各个处理器的主存储器的一部分。

[0267] 处理器1970、1980可各自利用点到点接口电路1976、1994、1986、1998经由个体P-P接口1952、1954与芯片集1990交换信息。芯片集1990可以可选地经由高性能接口1939与协处理器1938交换信息。在一个实施例中,协处理器1938是专用处理器,例如高吞吐量MIC处理器、网络或通信处理器、压缩引擎、图形处理器、GPGPU、嵌入式处理器,等等。

[0268] 共享缓存(未示出)可被包括在任一处理器中,或者在两个处理器之外,但经由P-P互连与处理器连接,从而使得任一个或两个处理器的本地缓存信息在处理器被置于低功率模式中的情况下可被存储在该共享缓存中。

[0269] 芯片集1990可经由接口1996耦合到第一总线1916。在一个实施例中,第一总线1916可以是外围组件互连(Peripheral Component Interconnect,PCI)总线,或者诸如快速PCI总线或另一种第三代I/O互连总线之类的总线,虽然本公开的范围不限于此。

[0270] 如图19中所示,各种I/O装置1914可耦合到第一总线1916,以及将第一总线1916耦合到第二总线1920的总线桥1918。在一个实施例中,一个或多个额外的处理器1915,例如协处理器、高吞吐量MIC处理器、GPGPU、加速器(例如,图形加速器或数字信号处理(DSP)单元)、现场可编程门阵列或者任何其他处理器,耦合到第一总线1916。在一个实施例中,第二总线1920可以是低引脚数(low pin count,LPC)总线。各种装置可耦合到第二总线1920,例如包括键盘和/或鼠标1922、通信装置1927和存储单元1928,例如盘驱动器或者其他大容量存储装置,它们在一个实施例中可包括指令/代码和数据1930。另外,音频I/O 1924可耦合到第二总线1920。注意其他体系结构是可能的。例如,取代图19的点到点体系结构,系统可实现多点分支总线或者其他这种体系结构。

[0271] 现在参考图20,其中示出了根据本公开的实施例的第二更具体示范性系统2000的框图。图19和图20中的相似元素带有相似的标号,并且图19的某些方面被从图20中省略以避免模糊图20的其他方面。

[0272] 图20图示出处理器1970、1980可分别包括集成存储器和I/O控制逻辑(“CL”)1972和1982。从而,CL 1972、1982包括集成存储器控制器单元并且包括I/O控制逻辑。图20图示出不仅存储器1932、1934耦合到CL 1972、1982,而且I/O装置2014也耦合到控制逻辑1972、1982。传统I/O装置2015耦合到芯片集1990。

[0273] 现在参考图21,其中示出了根据本公开的实施例的SoC 2100的框图。图17中的相

似元素带有相似的标号。另外，虚线框是更先进SoC上的可选特征。在图21中，(一个或多个)互连单元2102耦合到：应用处理器2110，其包括一组一个或多个核心202A-N，和(一个或多个)共享缓存单元1706；系统代理单元1710；(一个或多个)总线控制器单元1716；(一个或多个)集成存储器控制器单元1714；一组或一个或多个协处理器2120，其可包括集成图形逻辑、图像处理器、音频处理器和视频处理器；静态随机访问存储器(static random access memory，SRAM)单元2130；直接存储器访问(direct memory access，DMA)单元2132；以及显示单元2140，用于耦合到一个或多个外部显示器。在一个实施例中，(一个或多个)协处理器2120包括专用处理器，例如网络或通信处理器、压缩引擎、GPGPU、高吞吐量MIC处理器、嵌入式处理器，等等。

[0274] 本文公开的(例如，机制的)实施例可以用硬件、软件、固件或者这种实现方案的组合来实现。本公开的实施例可实现为在包括至少一个处理器、存储系统(包括易失性和非易失性存储器和/或存储元件)、至少一个输入装置和至少一个输出装置的可编程系统上执行的计算机程序或程序代码。

[0275] 程序代码，例如图19中所示的代码1930，可被应用到输入指令以执行本文描述的功能并且生成输出信息。输出信息可按已知的方式被应用到一个或多个输出装置。对于本申请而言，处理系统包括任何具有处理器的系统，例如；数字信号处理器(digital signal processor，DSP)、微控制器、专用集成电路(application specific integrated circuit，ASIC)或者微处理器。

[0276] 程序代码可以用高级过程式或面向对象的编程语言实现来与处理系统通信。如果希望，程序代码也可以用汇编或机器语言来实现。实际上，本文描述的机制在范围上不限于任何特定的编程语言。在任何情况下，该语言可以是经编译或者解译的语言。

[0277] 至少一个实施例的一个或多个方面可由存储在机器可读介质上的表示处理器内的各种逻辑的代表性指令实现，这些代表性指令当被机器读取时使得该机器制造逻辑来执行本文描述的技术。这种被称为“IP核”的表示可被存储在有形机器可读介质上并且被提供到各种客户或制造设施以加载到实际制作该逻辑或处理器的制造机器中。

[0278] 这种机器可读存储介质可包括但不限于由机器或装置制造或形成的物品的非暂态有形布置，包括存储介质，例如硬盘，任何其他类型的盘，包括软盘、光盘、致密盘只读存储器(compact disk read-only memory，CD-ROM)、可改写致密盘(compact disk rewritable，CD-RW)和磁光盘，半导体装置，例如只读存储器(read-only memory，ROM)，随机访问存储器(random access memory，RAM)，例如动态随机访问存储器(dynamic random access memory，DRAM)，静态随机访问存储器(static random access memory，SRAM)，可擦除可编程只读存储器(erasable programmable read-only memory，EEPROM)，闪速存储器，电可擦除可编程只读存储器(electrically erasable programmable read-only memory，EEPROM)，相变存储器(phase change memory，PCM)，磁卡或光卡，或者适合用于存储电子指令的任何其他类型的介质。

[0279] 因此，本公开的实施例还包括非暂态有形机器可读介质，其包含指令或者包含定义本文描述的结构、电路、装置、处理器和/或系统特征的设计数据，例如硬件描述语言(Hardware Description Language，HDL)。这种实施例也可被称为程序产品。

[0280] 仿真(包括二进制转化、代码变形等等)

[0281] 在一些情况下，指令转换器可用于将指令从源指令集转换到目标指令集。例如，指令转换器可将指令转化(例如，利用静态二进制转化、包括动态编译的动态二进制转化)、变形、仿真或以其他方式转换到要被核心处理的一个或多个其他指令。指令转换器可以用软件、硬件、固件或者其组合来实现。指令转换器可以在处理器上、在处理器外或者一部分在处理器上一部分在处理器外。

[0282] 图22是根据本公开的实施例与使用软件指令转换器来将源指令集中的二进制指令转换成目标指令集中的二进制指令相对比的框图。在图示的实施例中，指令转换器是软件指令转换器，虽然可替换地，指令转换器可以用软件、固件、硬件或者其各种组合来实现。图22示出了高级别语言2202的程序可被利用x86编译器2204来编译以生成x86二进制代码2206，x86二进制代码2206可由具有至少一个x86指令集核心2216的处理器原生执行。具有至少一个x86指令集核心2216的处理器表示任何这样的处理器：这种处理器可通过兼容地执行或以其他方式处理(1)Intel®x86指令集核心的指令集的实质部分或者(2)针对在具有至少一个x86指令集核心的Intel®处理器上运行的应用或其他软件的目标代码版本来执行与具有至少一个x86指令集核心的Intel®处理器基本上相同的功能，以便实现与具有至少一个x86指令集核心的Intel®处理器基本上相同的结果。x86编译器2204表示可操作来生成x86二进制代码2206(例如，目标代码)的编译器，x86二进制代码2206在带有或不带有额外的链接处理的情况下可被在具有至少一个x86指令集核心的处理器2216上执行。类似地，图22示出了高级别语言2202的程序可被利用替换指令集编译器2208来编译以生成替换指令集二进制代码2210，替换指令集二进制代码2210可由没有至少一个x86指令集核心的处理器2214(例如，具有执行加州森尼维耳市的MIPS技术公司的MIPS指令集和/或执行加州森尼维耳市的ARM控股公司的ARM指令集的核心的处理器)原生执行。指令转换器2212用于将x86二进制代码2206转换成可由没有x86指令集核心的处理器2214原生执行的代码。这个转换后的代码不太可能与替换指令集二进制代码2210相同，因为能够做到这一点的指令转换器是难以制作的；然而，转换后的代码将实现一般操作并且由来自替换指令集的指令构成。从而，指令转换器2212表示通过仿真、模拟或任何其他过程允许不具有x86指令集处理器或核心的处理器或其他电子装置执行x86二进制代码2206的软件、固件、硬件或者其组合。

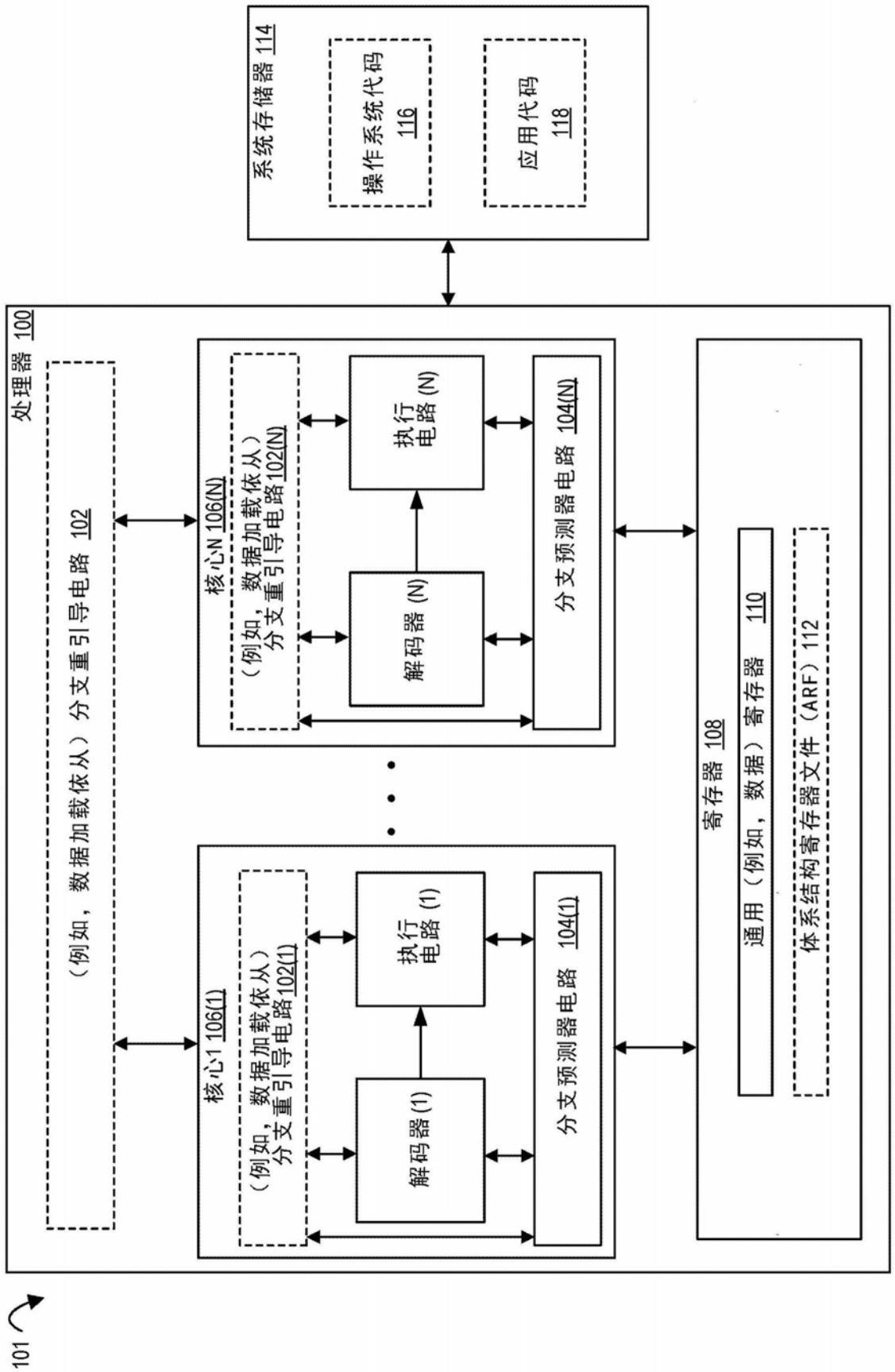

系统存储器 114
操作系统代码 116
应用代码 118
处理器 100
（例如，数据加载依从）分支重引导电路 102
核心N 106(N)
（例如，数据加载依从）分支重引导电路102(N)
执行电路 (N)
解码器 (N)
分支预测器电路 104(N)
核心1 106(1)
（例如，数据加载依从）分支重引导电路102(1)
执行电路 (1)
解码器 (1)
分支预测器电路 104(1)
寄存器 108
通用（例如，数据）寄存器 110
体系结构寄存器文件（ARF）112
101

图1

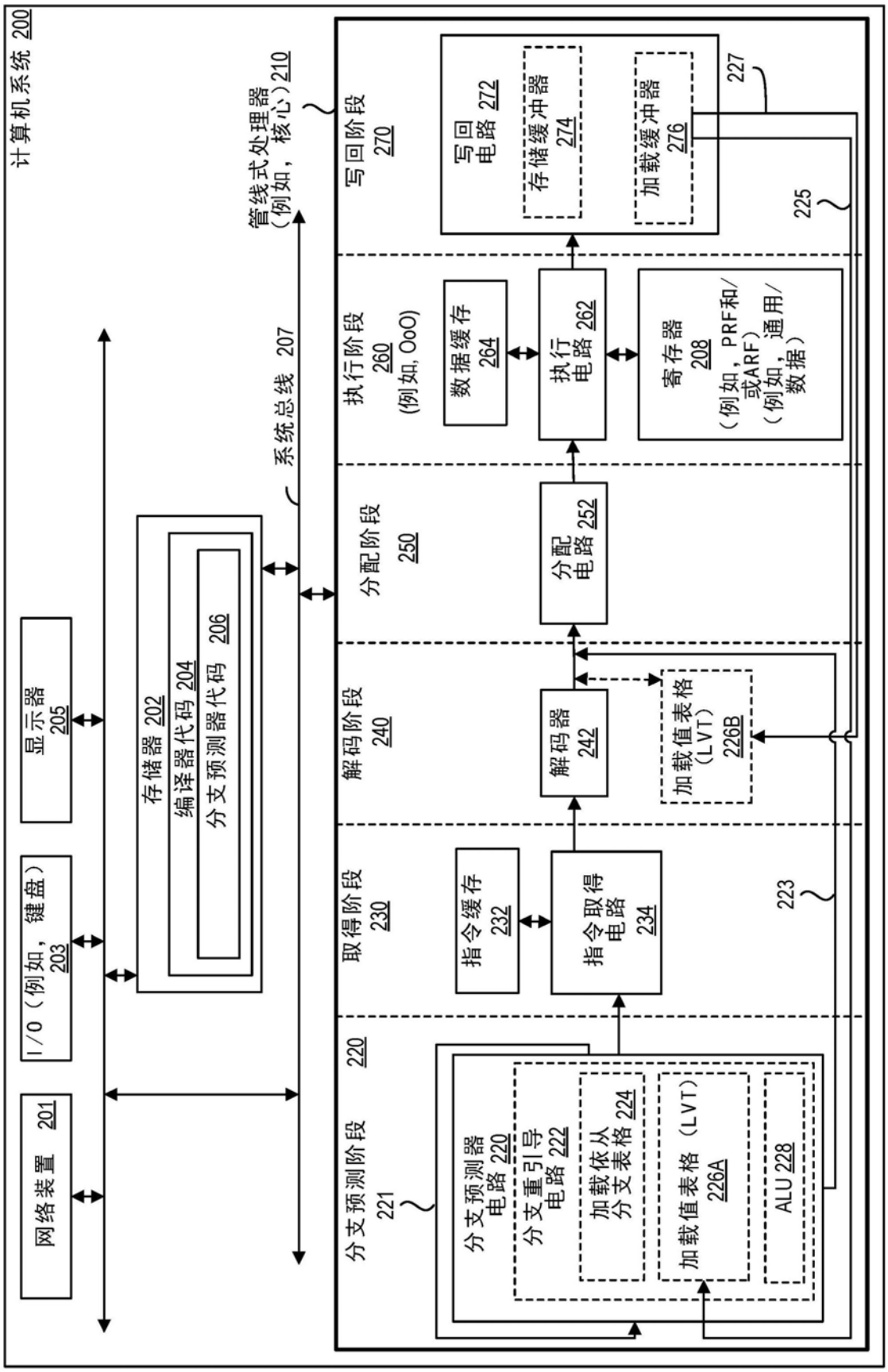

计算机系统 200
网络装置 201
I/O（例如，键盘）203
显示器 205
存储器 202
编译器代码 204
分支预测器代码 206
系统总线 207
管线式处理器（例如，核心）210
分支预测阶段 220
221
分支预测器电路 220
分支重引导电路 222
加载依从分支表格 224
加载值表格（LVT）226A
ALU 228
取得阶段 230
指令缓存 232
指令取得电路 234
解码阶段 240
解码器 242
加载值表格（LVT）226B
分配阶段 250
分配电路 252
执行阶段 260（例如，OoO）
数据缓存 264
执行电路 262
寄存器 208（例如，PRF和/或ARF）（例如，通用/数据）
写回阶段 270
写回电路 272
存储缓冲器 274
加载缓冲器 276
223
225
227

图2


加载依从分支表格条目格式
300
分支IP
302
误预测的数目
304
置信得分
306
加载IP，比较信息
308
LVT索引
310

图3

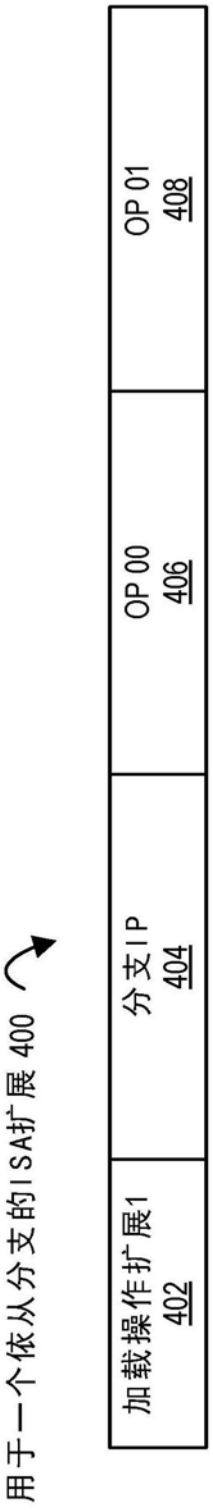


图4

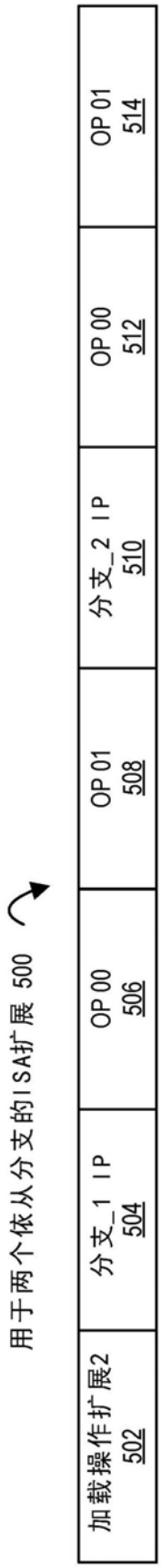

加载操作扩展2
502
分支_1 IP
504
OP 00
506
OP 01
508
分支_2 IP
510
OP 00
512
OP 01
514
用于两个依从分支的ISA扩展 500

图5

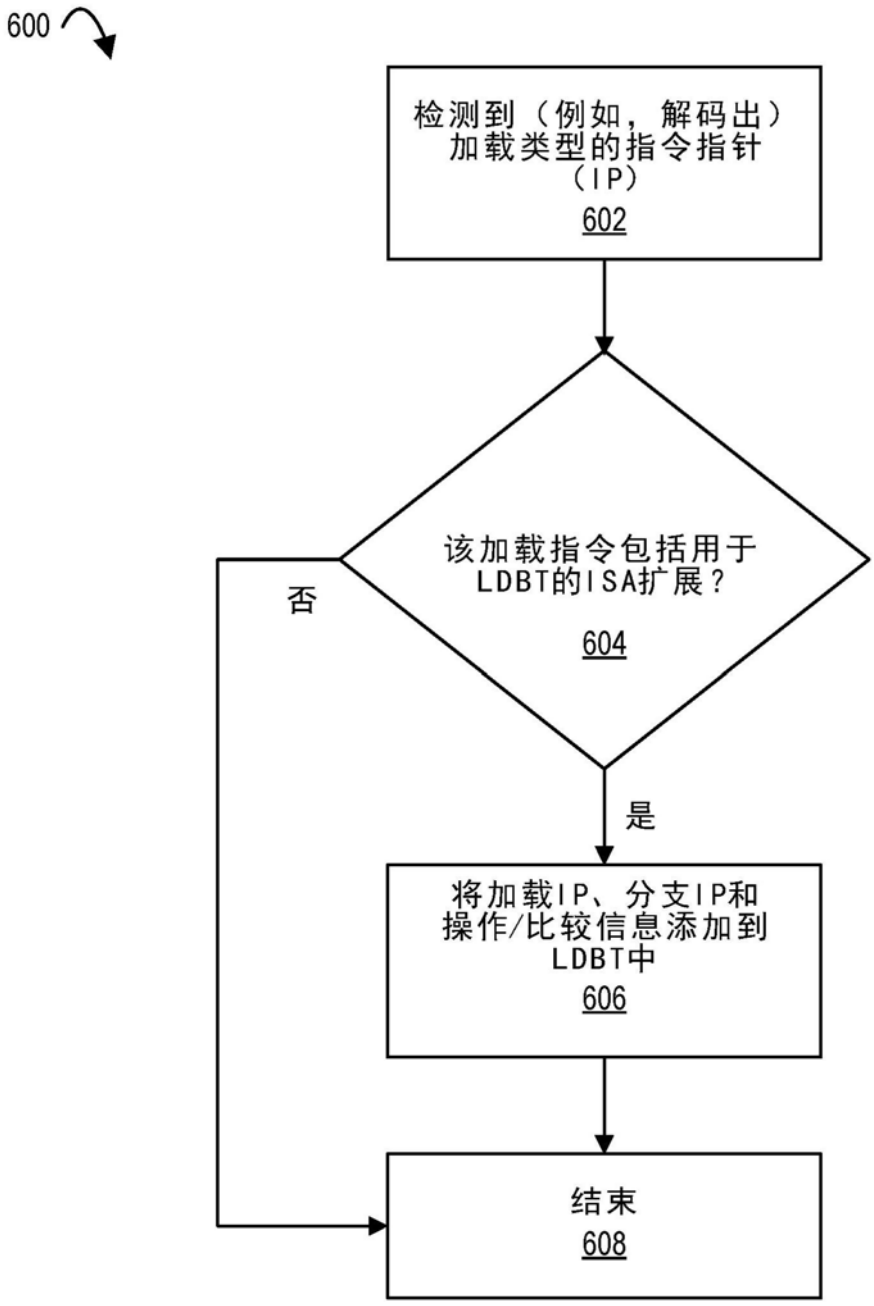

600
检测到（例如，解码出）加载类型的指令指针（IP）
602
该加载指令包括用于LDBT的ISA扩展？
604
否
是
将加载IP、分支IP和操作/比较信息添加到LDBT中
606
结束
608

图6

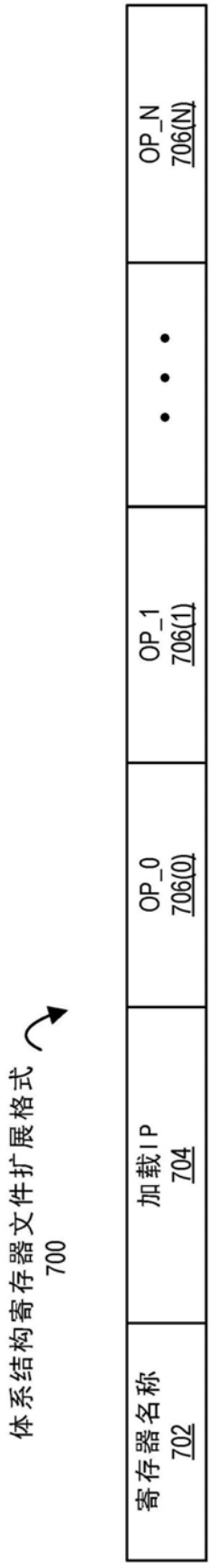

体系结构寄存器文件扩展格式
700
寄存器名称
702
加载IP
704
OP_0
706(0)
OP_1
706(1)
• • •
OP_N
706(N)

图7

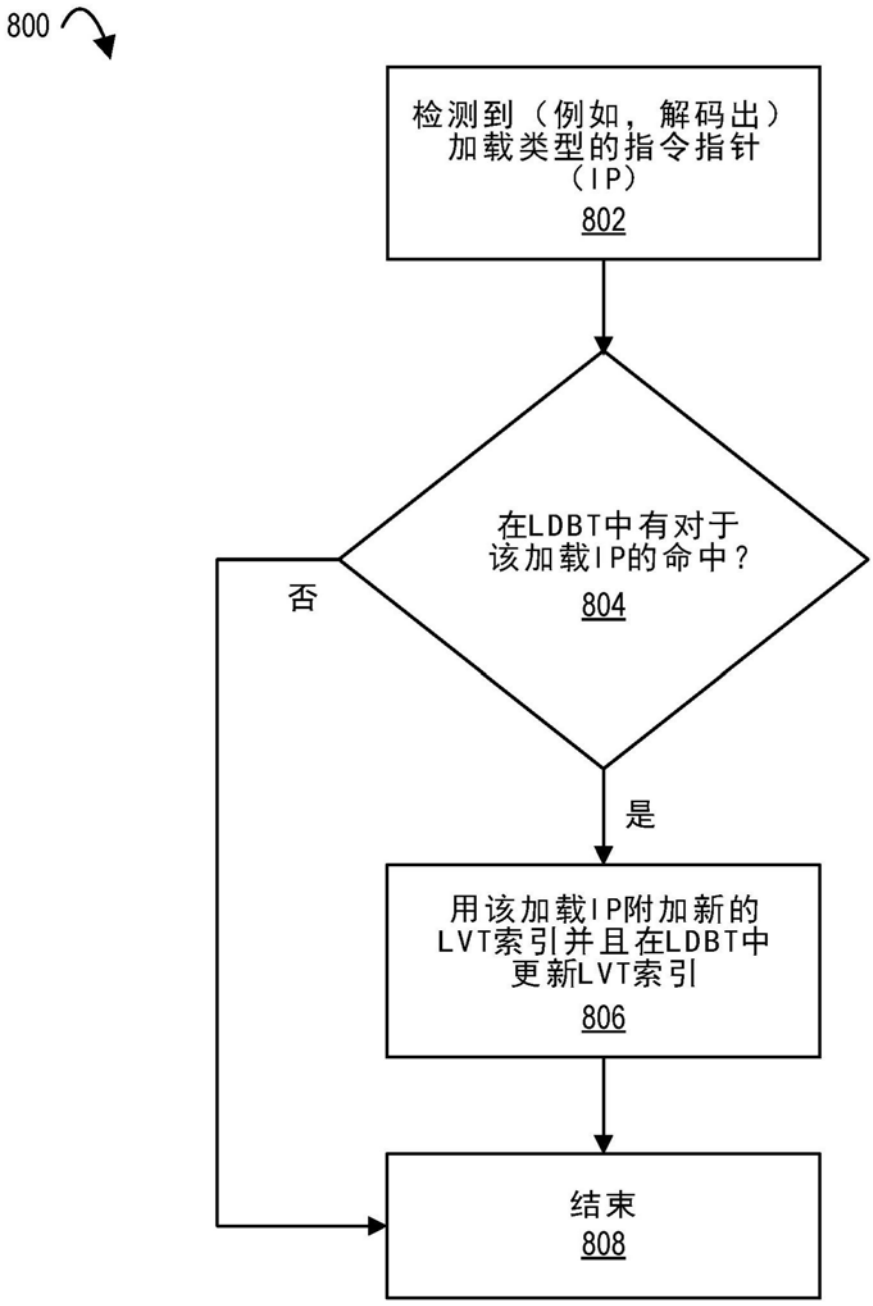

800
检测到（例如，解码出）加载类型的指令指针（IP）
802
在LDBT中有对于该加载IP的命中？
804
否
是
用该加载IP附加新的LVT索引并且在LDBT中更新LVT索引
806
结束
808

图8

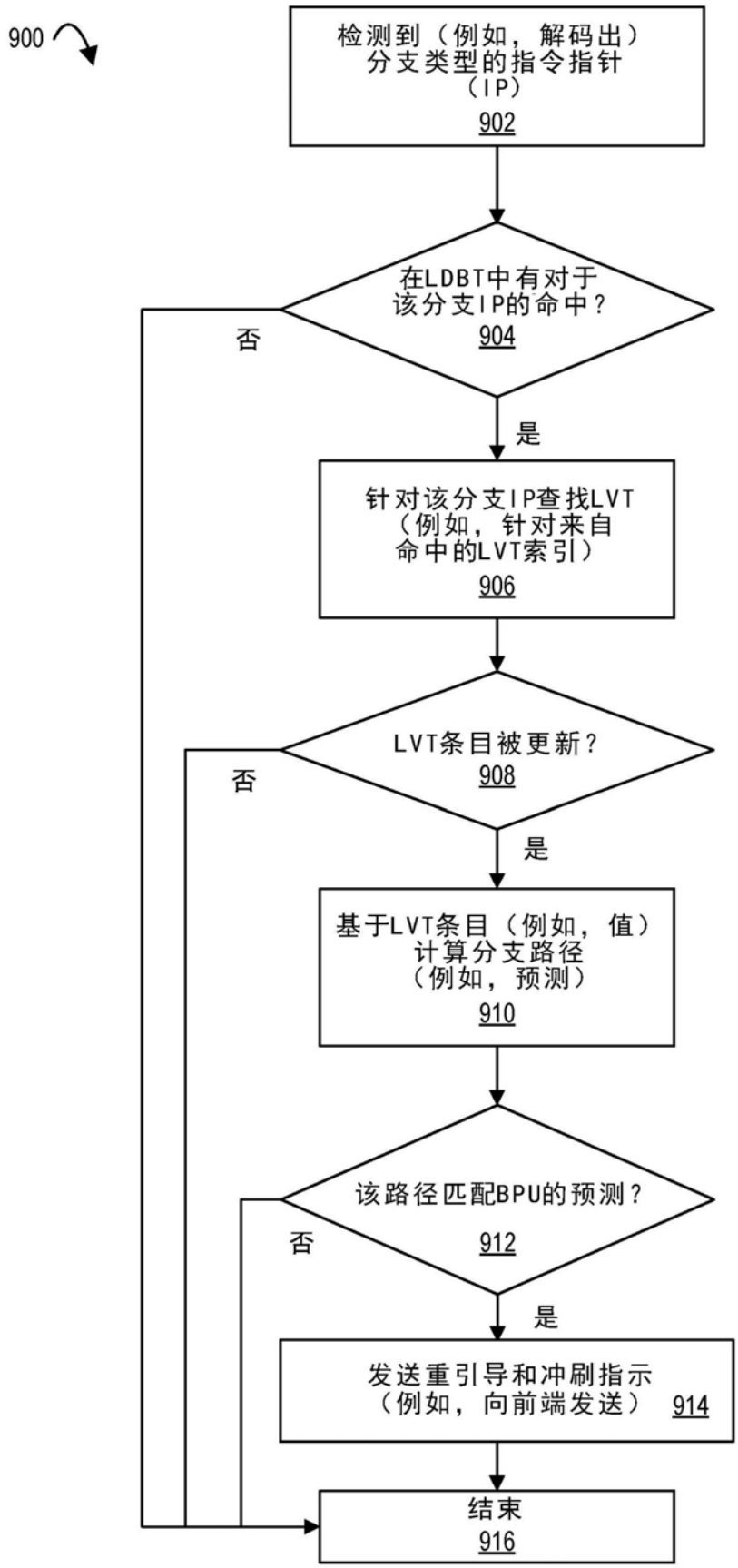

900
检测到（例如，解码出）分支类型的指令指针（IP）
902
在LDBT中有对于该分支IP的命中？
904
否
是
针对该分支IP查找LVT（例如，针对来自命中的LVT索引）
906
LVT条目被更新？
908
否
是
基于LVT条目（例如，值）计算分支路径（例如，预测）
910
该路径匹配BPU的预测？
912
否
是
发送重引导和冲刷指示（例如，向前端发送）
914
结束
916

图9

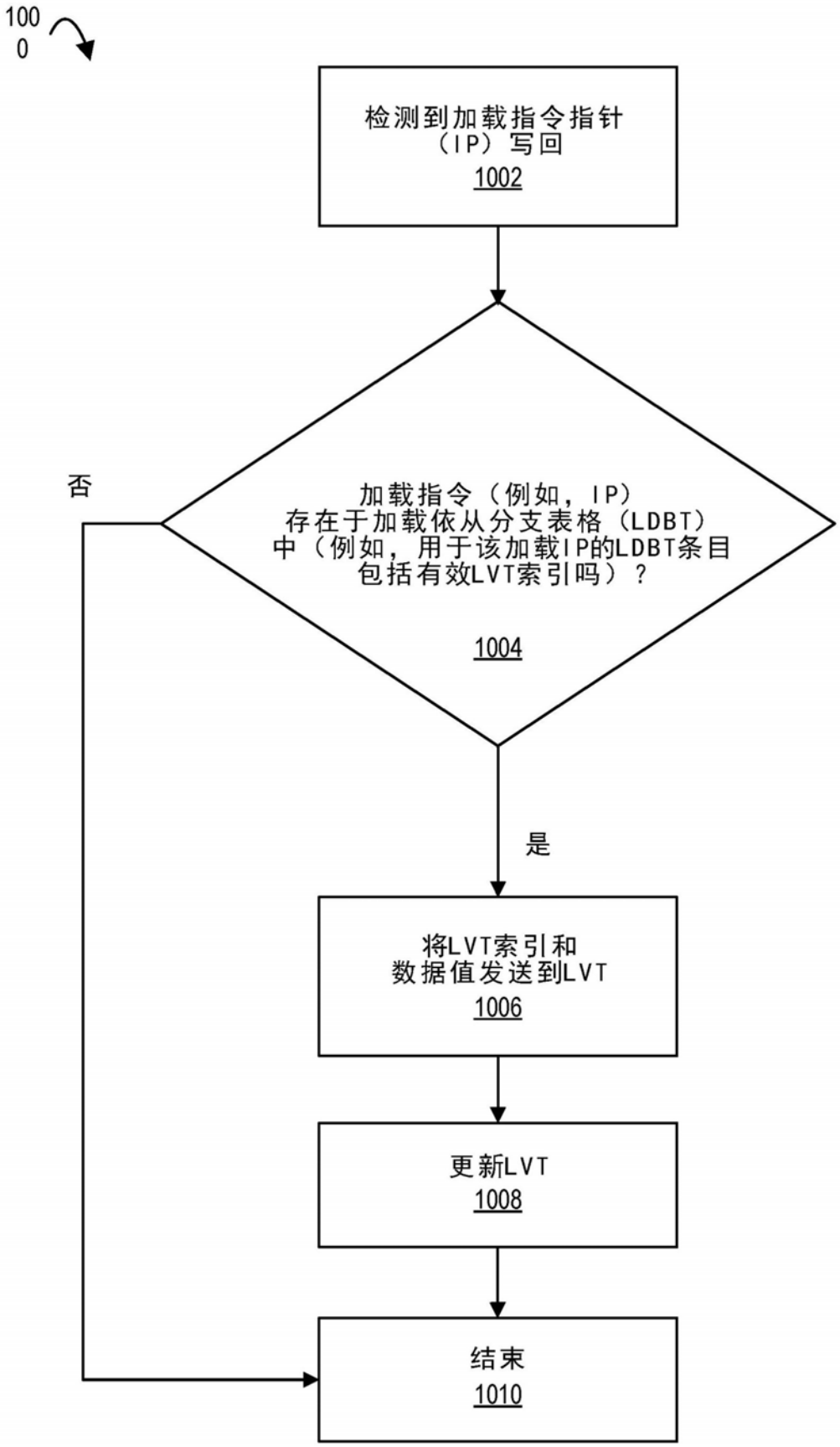

1000
检测到加载指令指针（IP）写回
1002
加载指令（例如，IP）存在于加载依从分支表格（LDBT）中（例如，用于该加载IP的LDBT条目包括有效LVT索引吗）？
1004
否
是
将LVT索引和数据值发送到LVT
1006
更新LVT
1008
结束
1010

图10

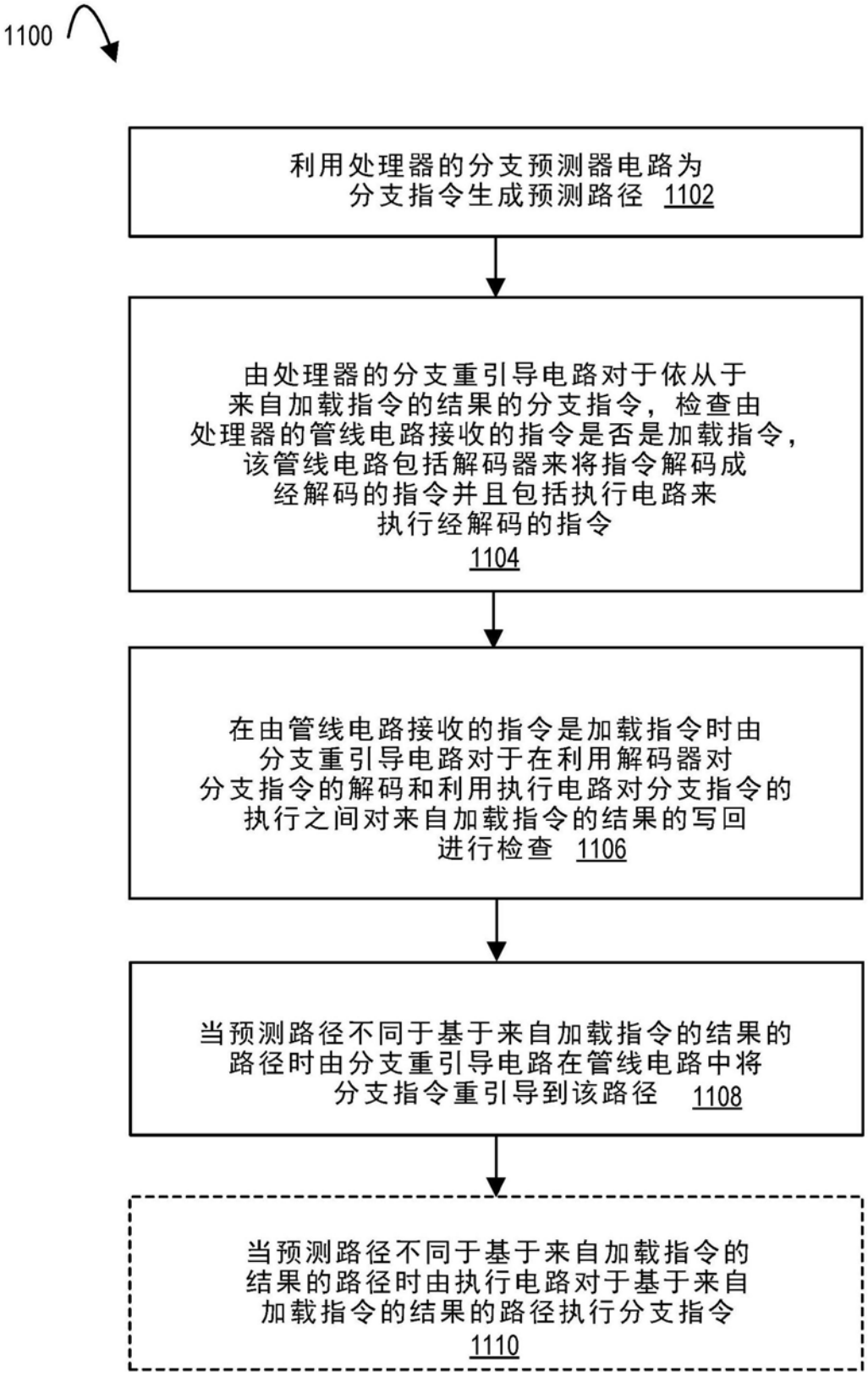

1100
利用处理器的分支预测器电路为分支指令生成预测路径 1102
由处理器的分支重引导电路对于依从于来自加载指令的结果的分支指令，检查由处理器的管线电路接收的指令是否是加载指令，该管线电路包括解码器来将指令解码成经解码的指令并且包括执行电路来执行经解码的指令 1104
在由管线电路接收的指令是加载指令时由分支重引导电路对于在利用解码器对分支指令的解码和利用执行电路对分支指令的执行之间对来自加载指令的结果的写回进行检查 1106
当预测路径不同于基于来自加载指令的结果的路径时由分支重引导电路在管线电路中将分支指令重引导到该路径 1108
当预测路径不同于基于来自加载指令的结果的路径时由执行电路对于基于来自加载指令的结果的路径执行分支指令 1110

图11

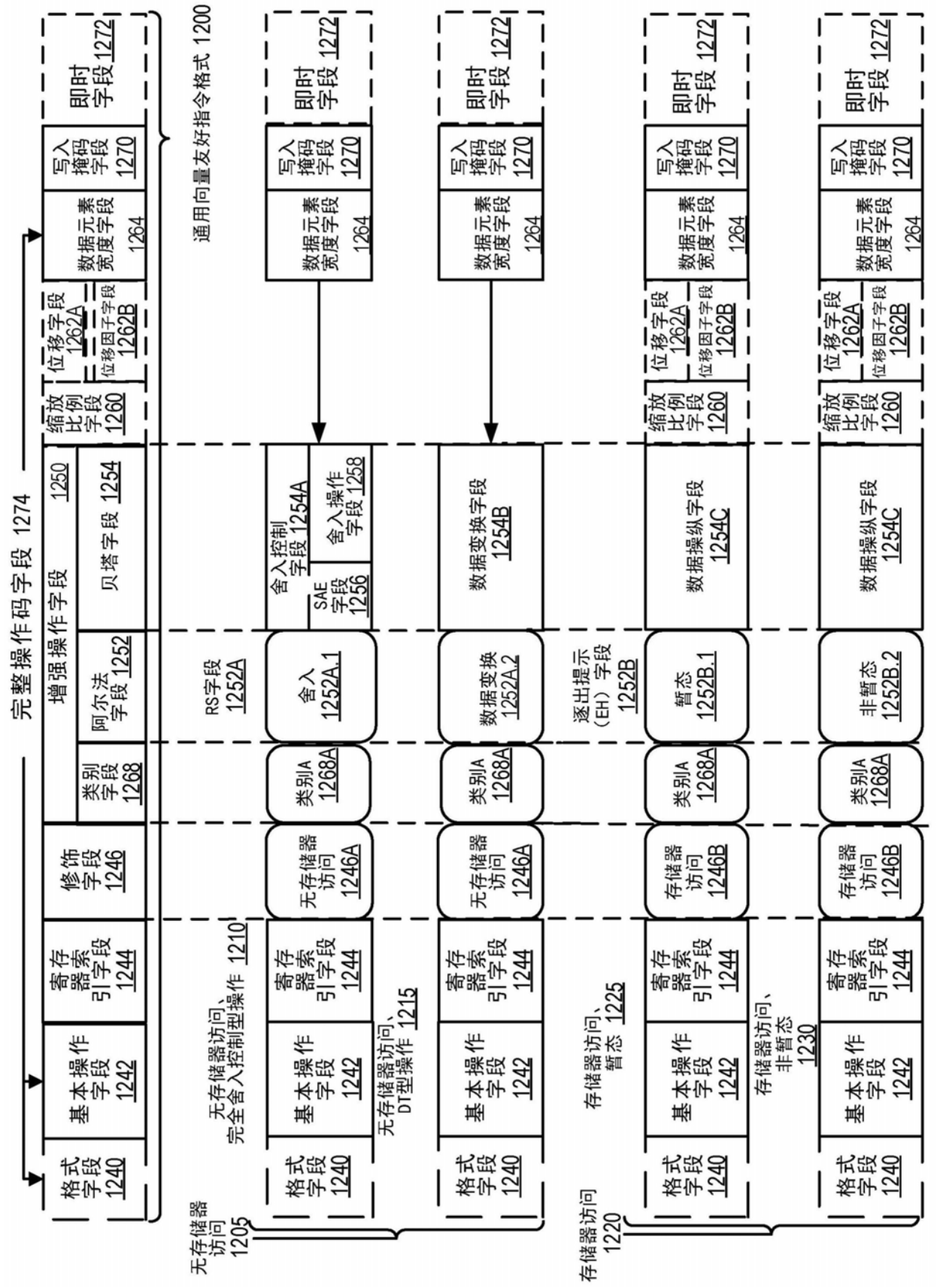

通用向量友好指令格式 1200
完整操作码字段 1274
格式字段 1240
基本操作字段 1242
寄存器索引字段 1244
修饰字段 1246
类别字段 1268
增强操作字段 1250
阿尔法字段 1252
贝塔字段 1254
缩放比例字段 1260
位移字段 1262A
位移因子字段 1262B
数据元素宽度字段 1264
写入掩码字段 1270
即时字段 1272
无存储器访问 1205
无存储器访问、完全舍入控制型操作 1210
RS字段 1252A
舍入 1252A.1
舍入控制字段 1254A
SAE字段 1256
舍入操作字段 1258
无存储器访问、DT型操作 1215
无存储器访问 1246A
类别A 1268A
数据变换 1252A.2
数据变换字段 1254B
存储器访问 1220
存储器访问、暂态 1225
逐出提示（EH）字段 1252B
暂态 1252B.1
数据操纵字段 1254C
存储器访问 1246B
存储器访问、非暂态 1230
非暂态 1252B.2

图12A

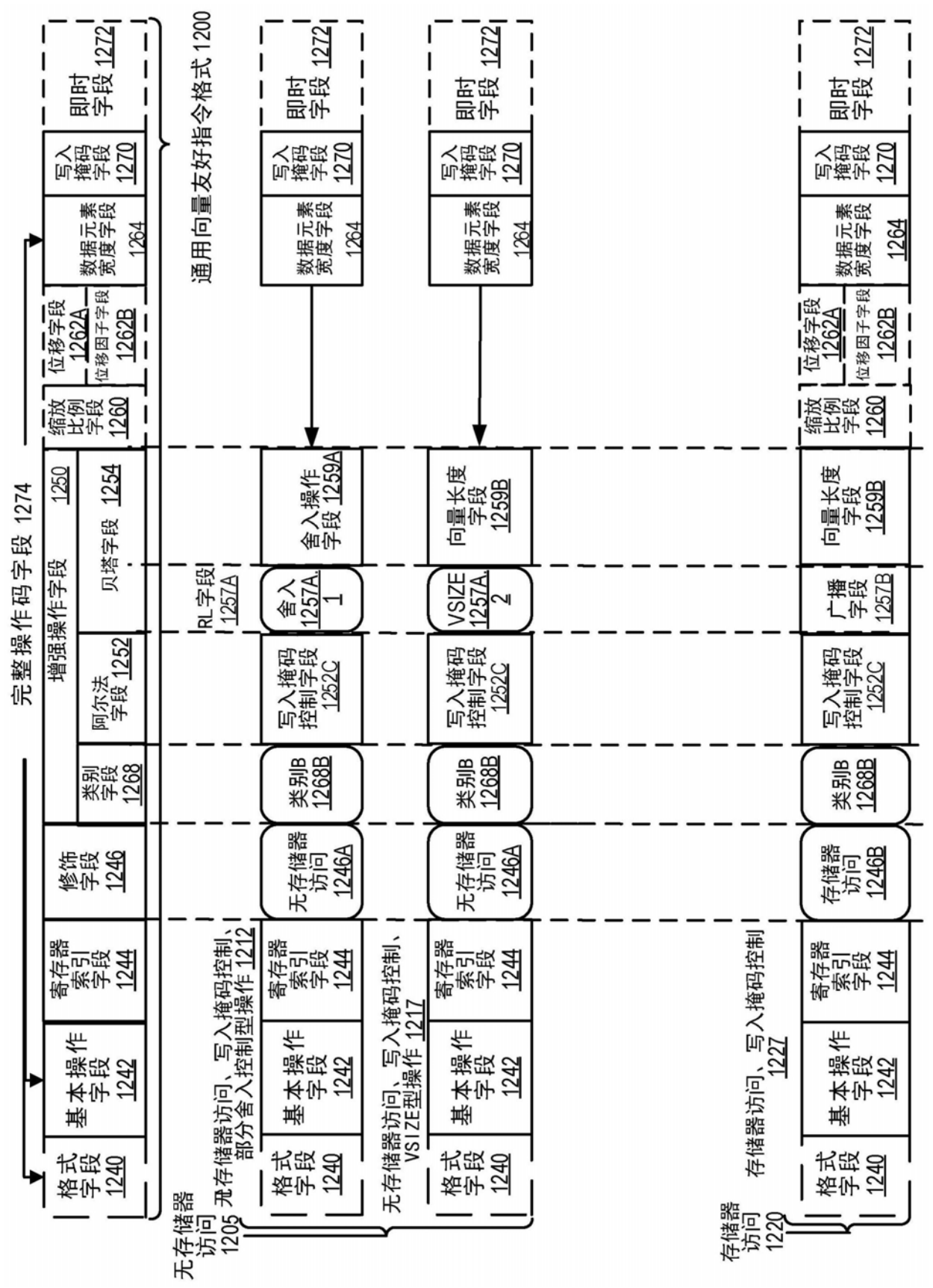

通用向量友好指令格式 1200
完整操作码字段 1274
格式字段 1240
基本操作字段 1242
寄存器索引字段 1244
修饰字段 1246
类别字段 1268
增强操作字段 1250
阿尔法字段 1252
贝塔字段 1254
缩放比例字段 1260
位移字段 1262A
位移因子字段 1262B
数据元素宽度字段 1264
写入掩码字段 1270
即时字段 1272
RL字段 1257A
舍入操作字段 1259A
向量长度字段 1259B
舍入 1257A.1
VSIZE 1257A.2
广播字段 1257B
写入掩码控制字段 1252C
无存储器访问 1205
无存储器访问、写入掩码控制、部分舍入控制型操作 1212
无存储器访问 1246A
类别B 1268B
无存储器访问、写入掩码控制、VSIZE型操作 1217
存储器访问 1220
存储器访问、写入掩码控制 1227
存储器访问 1246B

图12B

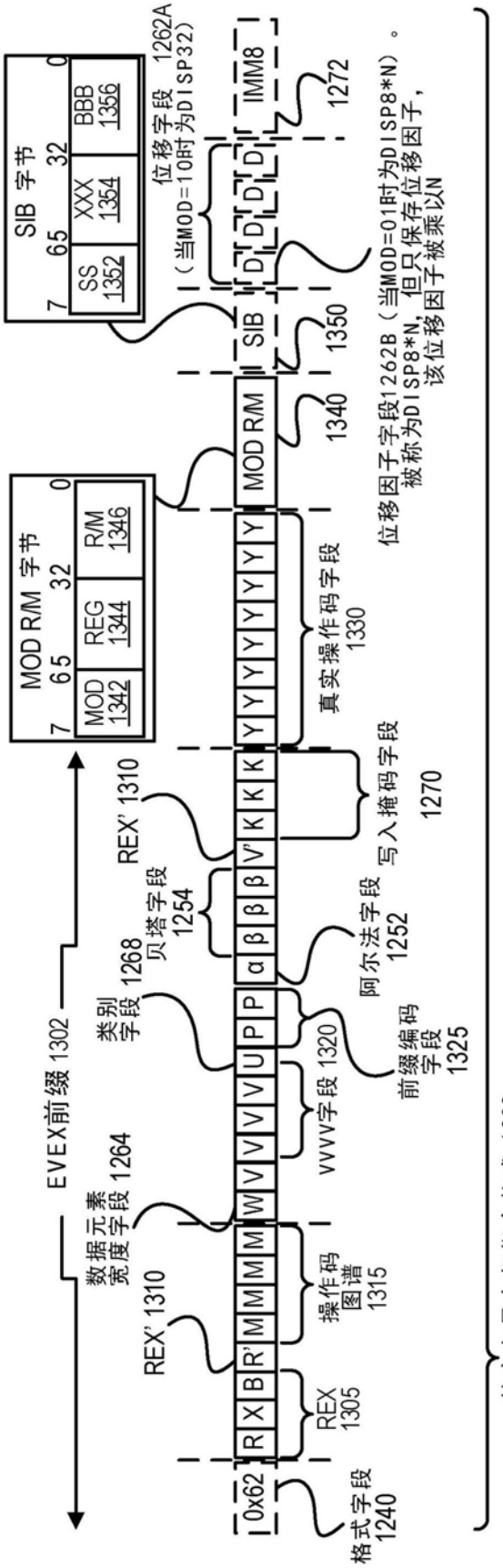

SIB 字节
SS 1352
XXX 1354
BBB 1356
位移字段 1262A
（当MOD=10时为DISP32）
MOD R/M 字节
MOD 1342
REG 1344
R/M 1346
EVEX前缀 1302
数据元素宽度字段 1264
类别字段 1268
贝塔字段 1254
REX' 1310
阿尔法字段 1252
写入掩码字段 1270
真实操作码字段 1330
MOD R/M 1340
SIB 1350
IMM8 1272
位移因子字段1262B（当MOD=01时为DISP8*N）。被称为DISP8*N，但只保存位移因子，该位移因子被乘以N
REX 1305
操作码图谱 1315
VVVV字段 1320
前缀编码字段 1325
格式字段 1240
0x62
特定向量友好指令格式 1300

图13A

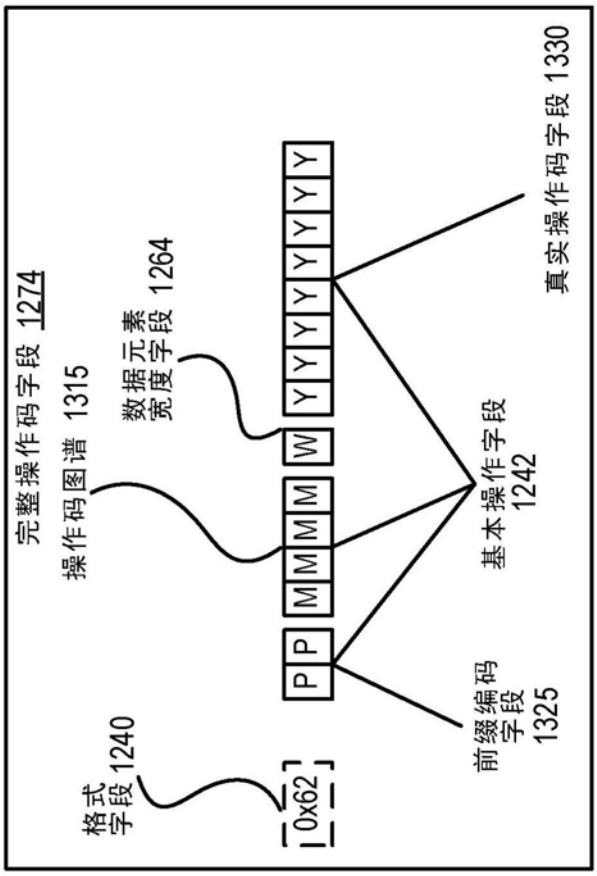

完整操作码字段 1274
操作码图谱 1315
数据元素宽度字段 1264
格式字段 1240
0x62
P P M M M W Y Y Y Y Y Y Y
前缀编码字段 1325
基本操作字段 1242
真实操作码字段 1330

图13B

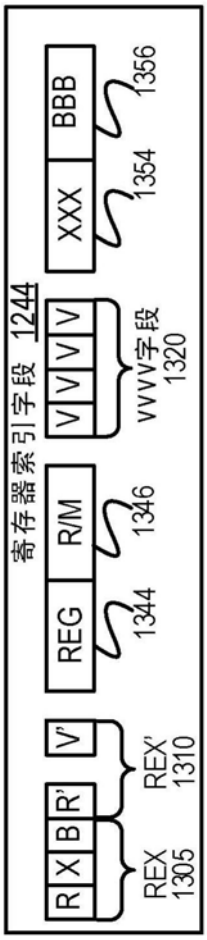

寄存器索引字段 1244
R
X
B
R'
REX
1305
V'
REX'
1310
REG
1344
R/M
1346
V
V
V
VVVV字段
1320
XXX
1354
BBB
1356

图13C

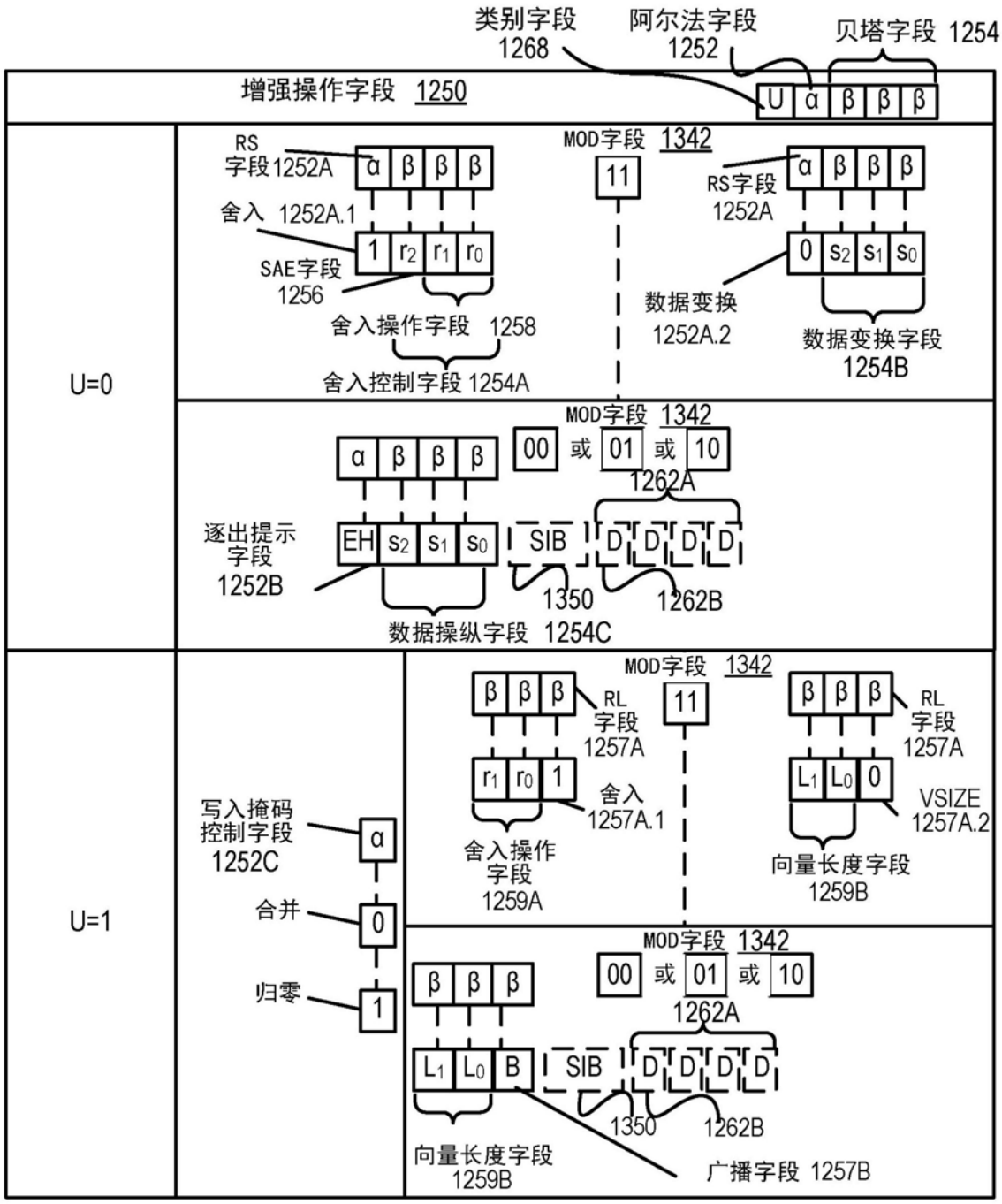

类别字段 1268
阿尔法字段 1252
贝塔字段 1254
增强操作字段 1250
U α β β β
U=0
RS 字段1252A
舍入 1252A.1
SAE字段 1256
舍入操作字段 1258
舍入控制字段 1254A
MOD字段 1342
11
RS字段 1252A
数据变换 1252A.2
数据变换字段 1254B
00 或 01 或 10
1262A
逐出提示 字段 1252B
EH s2 s1 s0
SIB
D D D D
1350
1262B
数据操纵字段 1254C
U=1
写入掩码 控制字段 1252C
合并
归零
RL 字段 1257A
舍入 1257A.1
舍入操作 字段 1259A
VSIZE 1257A.2
向量长度字段 1259B
L1 L0 B
广播字段 1257B

图13D

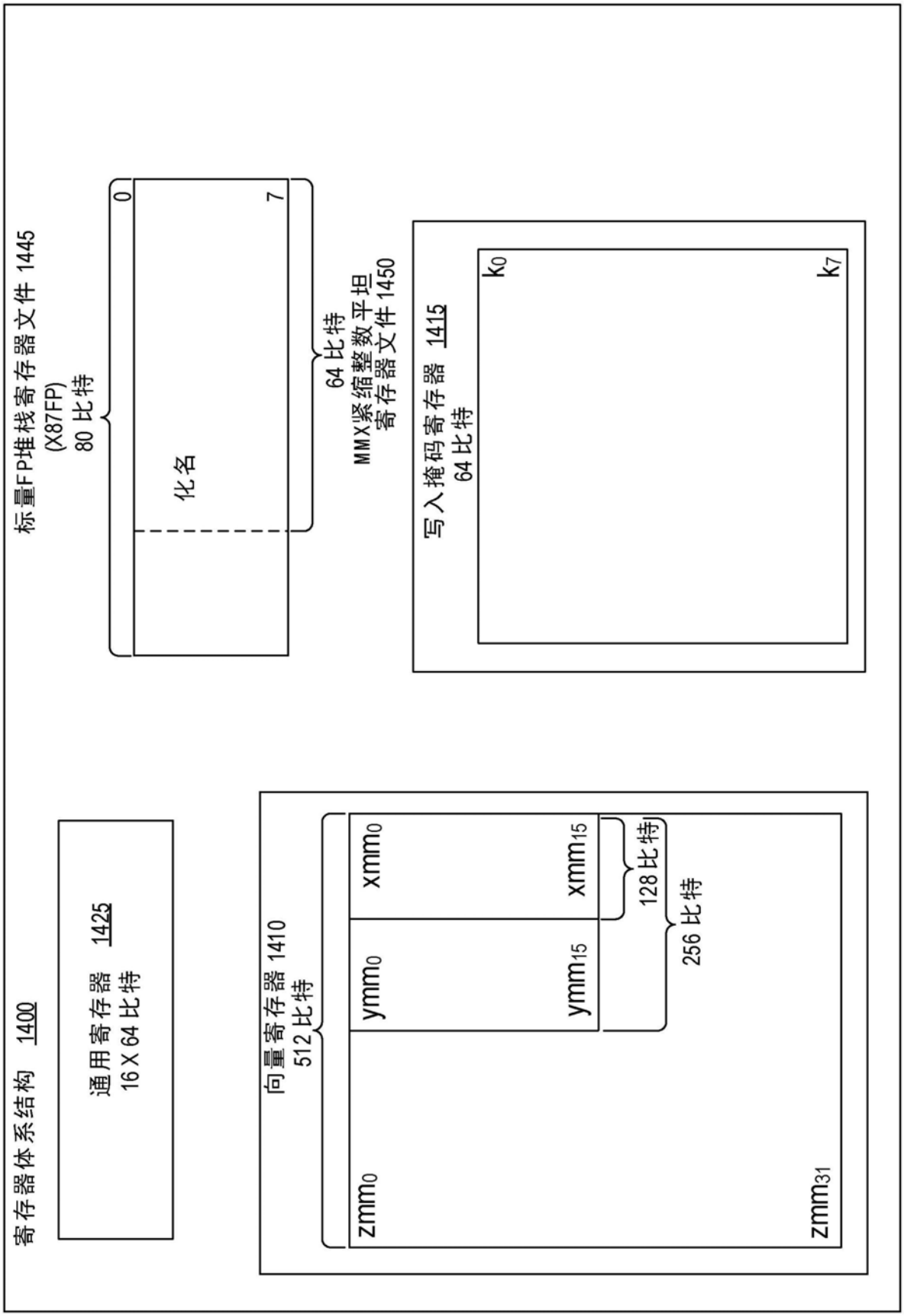

寄存器体系结构 1400
通用寄存器 1425
16 X 64 比特
标量FP堆栈寄存器文件 1445
(X87FP)
80 比特
0
7
化名
64 比特
MMX紧缩整数平坦
寄存器文件 1450
写入掩码寄存器 1415
64 比特
k0
k7
向量寄存器 1410
512 比特
zmm0
ymm0
xmm0
ymm15
xmm15
128 比特
256 比特
zmm31

图14

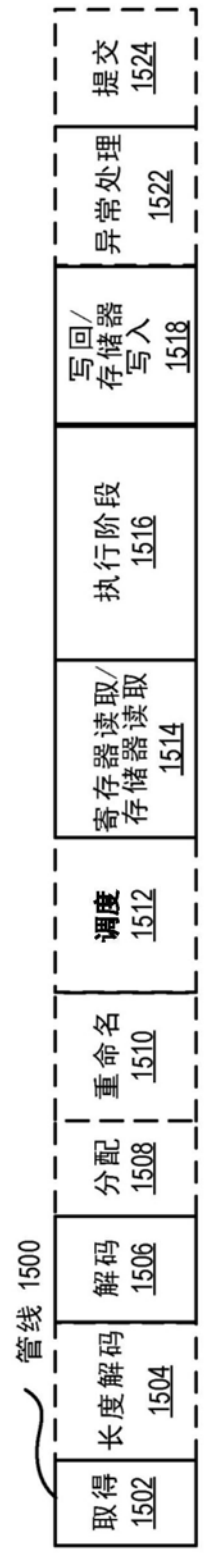

管线 1500
取得 1502
长度解码 1504
解码 1506
分配 1508
重命名 1510
调度 1512
寄存器读取/存储器读取 1514
执行阶段 1516
写回/存储器写入 1518
异常处理 1522
提交 1524

图15A

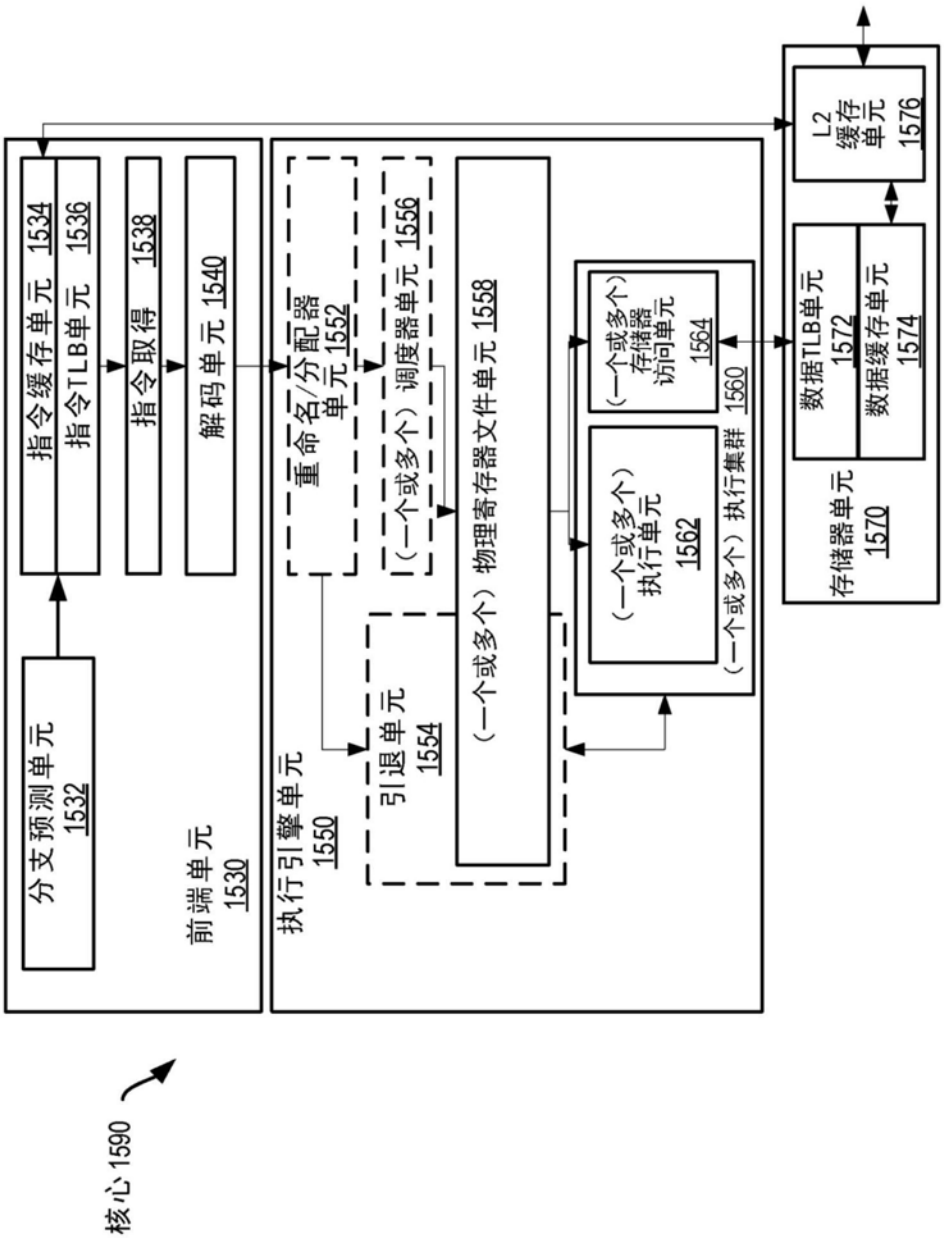

核心 1590
前端单元 1530
分支预测单元 1532
指令缓存单元 1534
指令TLB单元 1536
指令取得 1538
解码单元 1540
执行引擎单元 1550
重命名/分配器单元 1552
引退单元 1554
(一个或多个)调度器单元 1556
(一个或多个)物理寄存器文件单元 1558
(一个或多个)执行集群 1560
(一个或多个)执行单元 1562
(一个或多个)存储器访问单元 1564
存储器单元 1570
数据TLB单元 1572
数据缓存单元 1574
L2缓存单元 1576

图15B

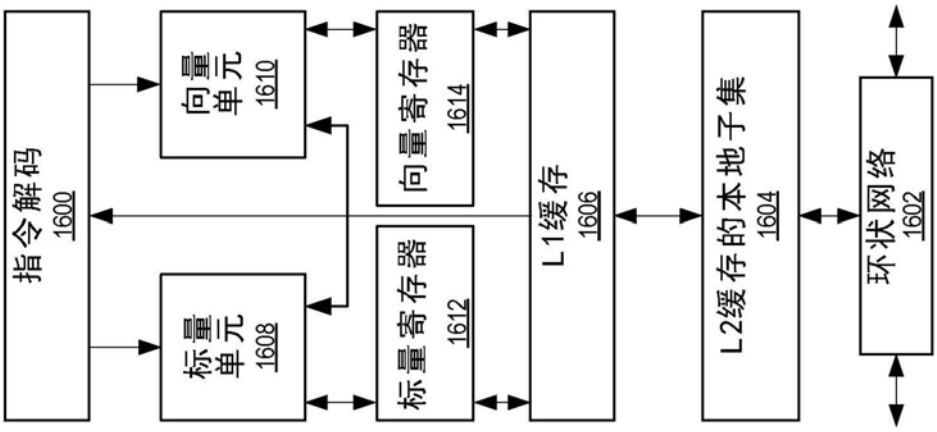

指令解码
1600
向量
单元
1610
标量
单元
1608
向量寄存器
1614
标量寄存器
1612
L1缓存
1606
L2缓存的本地子集
1604
环状网络
1602

图16A

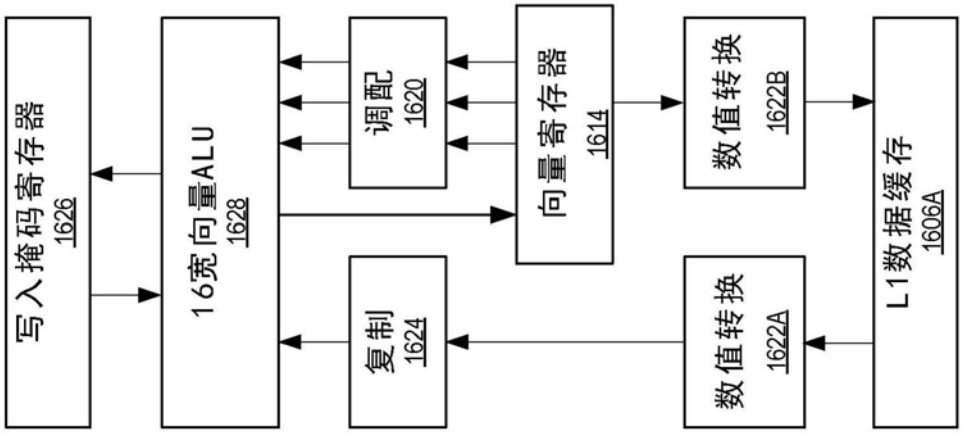

写入掩码寄存器
1626
16宽向量ALU
1628
调配
1620
复制
1624
向量寄存器
1614
数值转换
1622B
数值转换
1622A
L1数据缓存
1606A

图16B

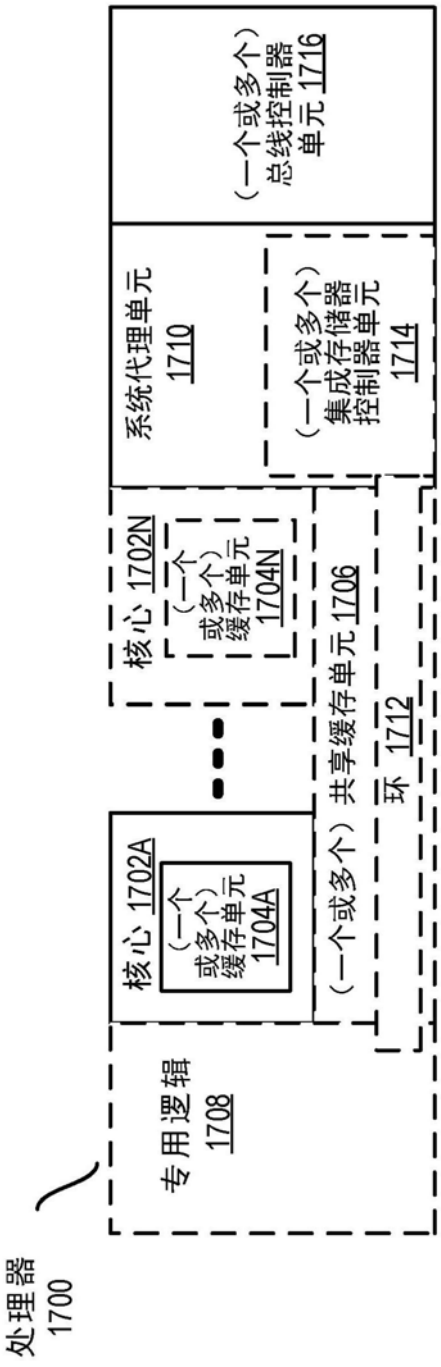

处理器
1700
专用逻辑
1708
核心 1702A
（一个或多个）缓存单元 1704A
核心 1702N
（一个或多个）缓存单元 1704N
（一个或多个）共享缓存单元 1706
环 1712
系统代理单元 1710
（一个或多个）集成存储器控制器单元 1714
（一个或多个）总线控制器单元 1716

图17

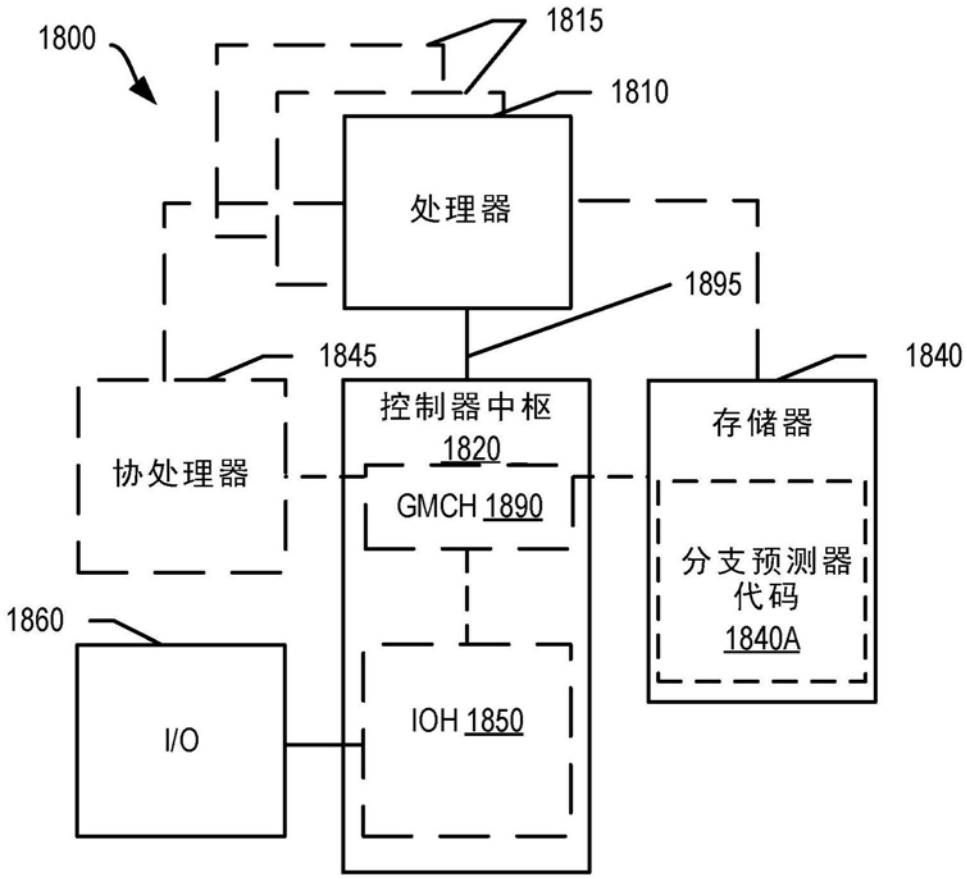

1800
1815
1810
处理器
1895
1845
1840
控制器中枢
1820
协处理器
GMCH 1890
存储器
分支预测器
代码
1840A
1860
I/O
IOH 1850

图18

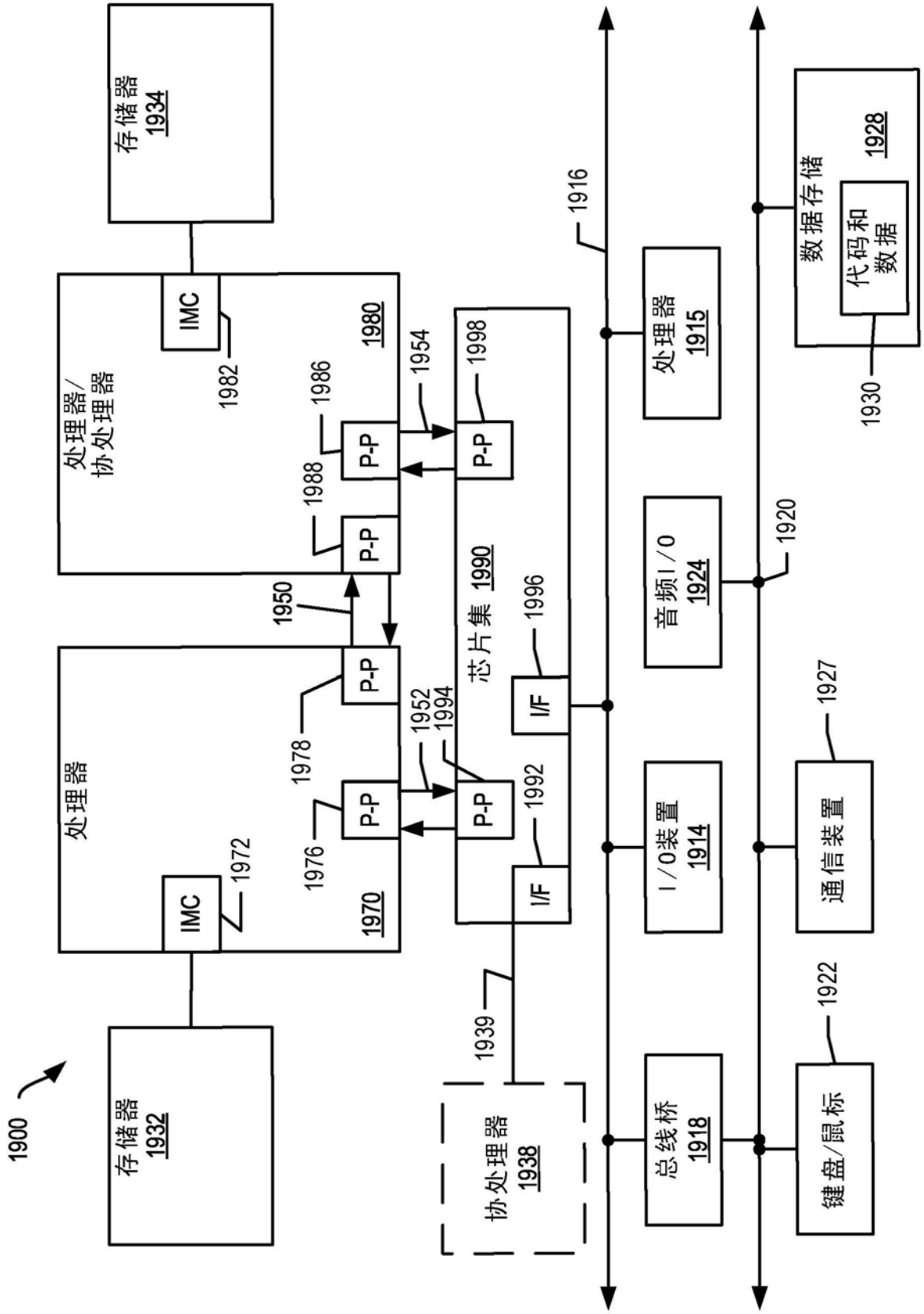

1900
存储器 1932
存储器 1934
处理器 1970
处理器/协处理器 1980
IMC 1972
IMC 1982
P-P 1976
P-P 1978
P-P 1988
P-P 1986
P-P 1952
P-P 1994
P-P 1954
P-P 1998
1950
芯片集 1990
I/F 1992
I/F 1996
1939
协处理器 1938
1916
处理器 1915
音频I/O 1924
1920
I/O装置 1914
通信装置 1927
总线桥 1918
键盘/鼠标 1922
数据存储 1928
代码和数据 1930

图19

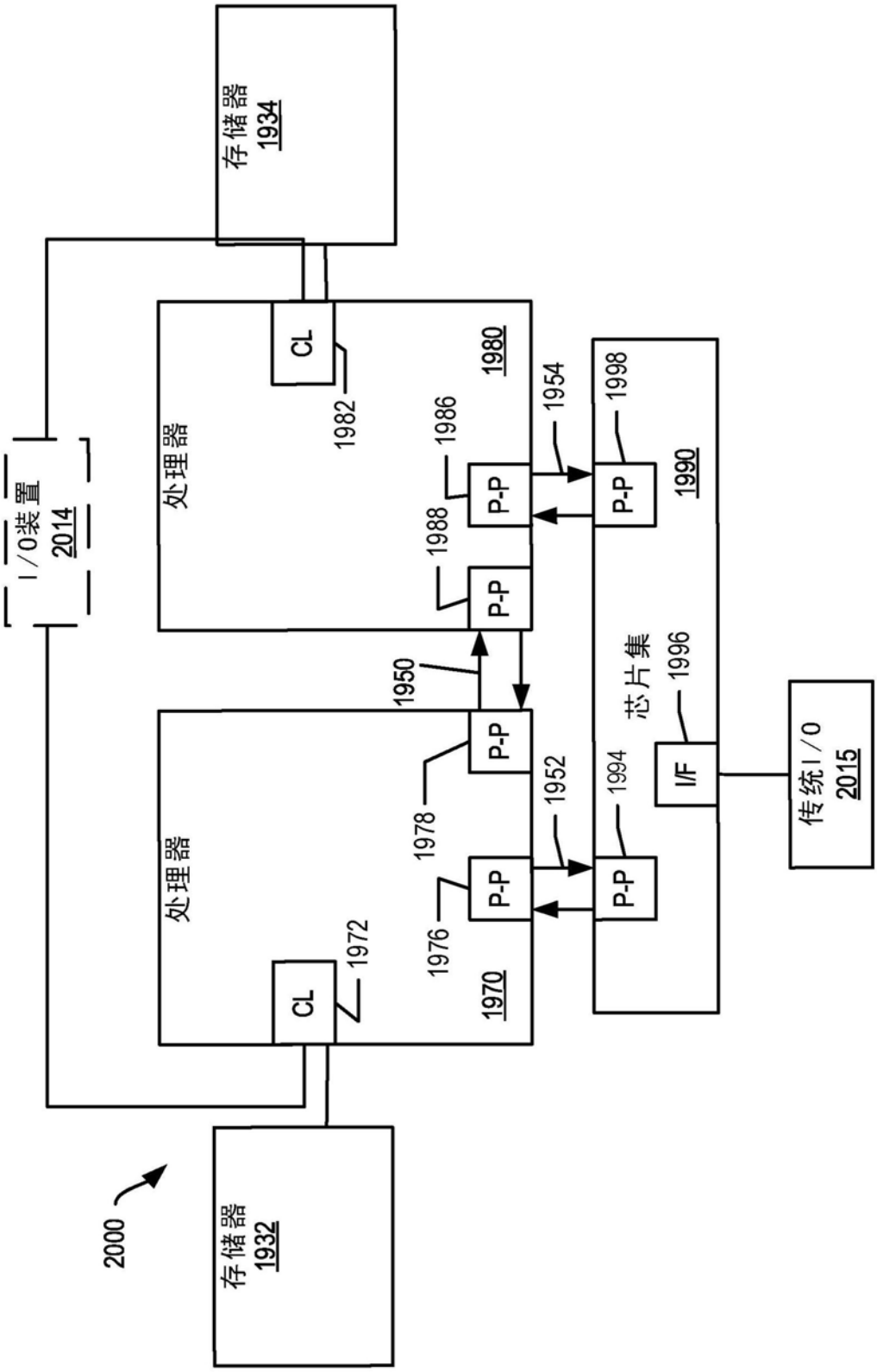

存储器
1934
CL
1982
处理器
1980
1986
1954
1998
P-P
P-P
1988
P-P
1990
I/O装置
2014
1950
芯片集
1996
P-P
1978
1952
1994
I/F
传统I/O
2015
处理器
P-P
1976
1970
CL
1972
2000
存储器
1932

图20

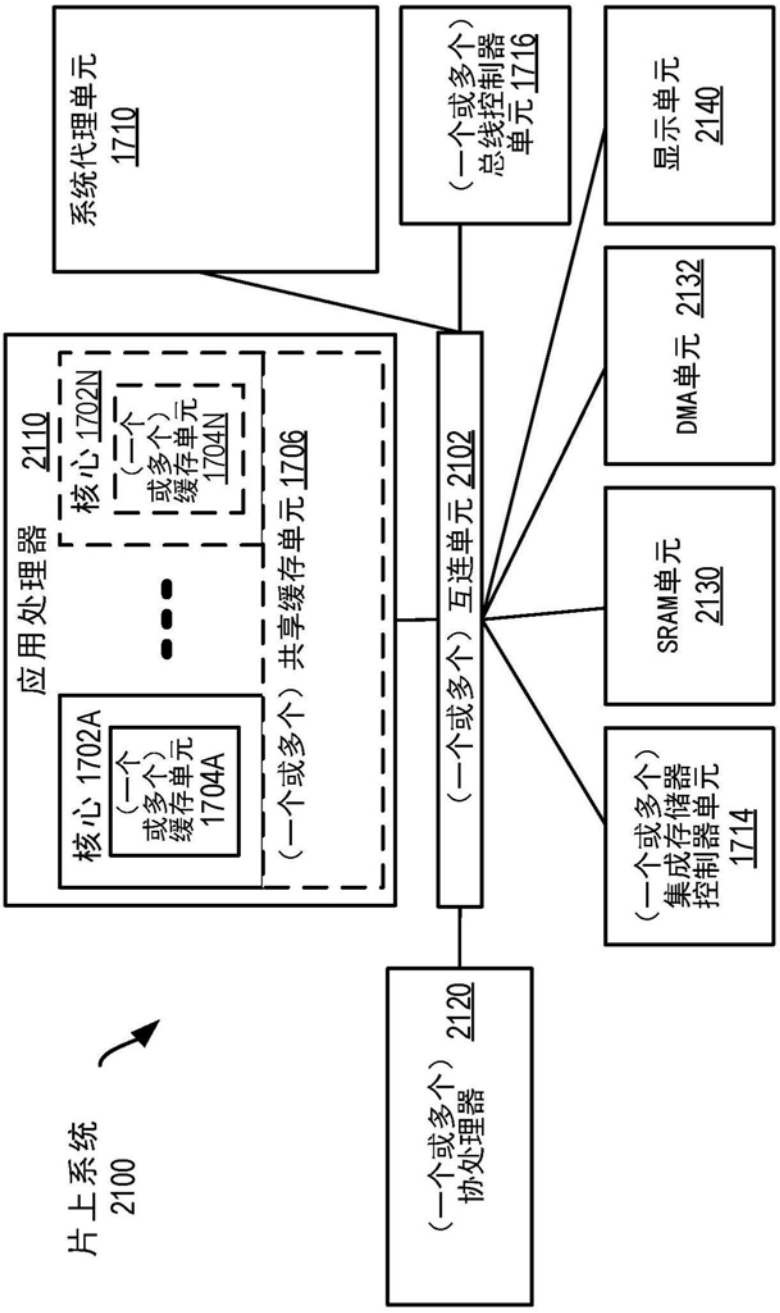

片上系统
2100
应用处理器 2110
核心 1702A
（一个或多个）缓存单元 1704A
核心 1702N
（一个或多个）缓存单元 1704N
（一个或多个）共享缓存单元 1706
（一个或多个）互连单元 2102
系统代理单元 1710
（一个或多个）总线控制器单元 1716
显示单元 2140
DMA单元 2132
SRAM单元 2130
（一个或多个）集成存储器控制器单元 1714
（一个或多个）协处理器 2120

图21

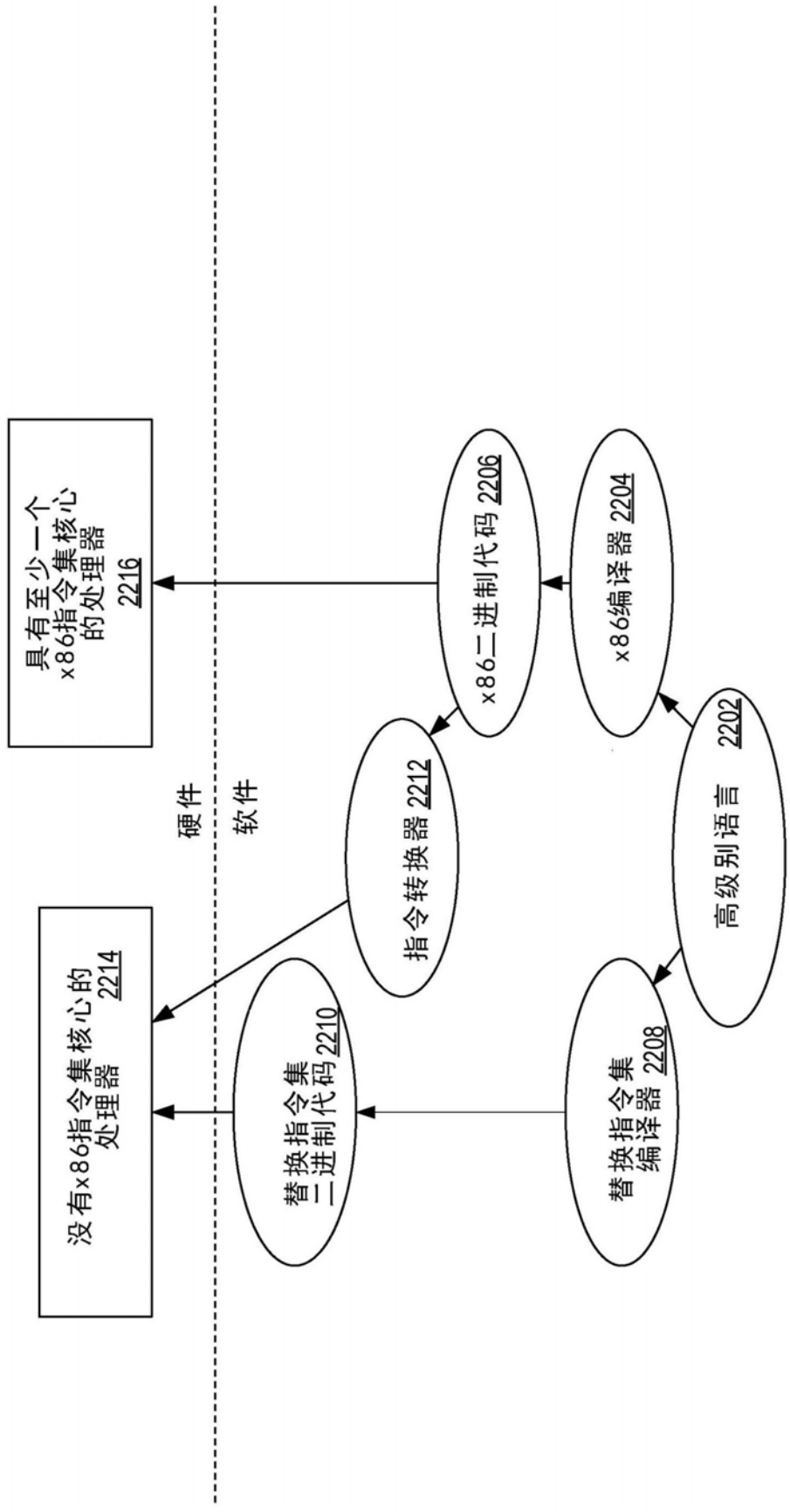

具有至少一个x86指令集核心的处理器 2216
x86二进制代码 2206
x86编译器 2204
指令转换器 2212
高级别语言 2202
硬件
软件
没有x86指令集核心的处理器 2214
替换指令集二进制代码 2210
替换指令集编译器 2208

图22