(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 103718157 B
(45)授权公告日 2017.05.24

(21)申请号 201180069925.6
(22)申请日 2011.12.12
(65)同一申请的已公布的文献号
申请公布号 CN 103718157 A
(43)申请公布日 2014.04.09
(30)优先权数据
13/078,901 2011.04.01 US
(85)PCT国际申请进入国家阶段日
2013.10.08
(86)PCT国际申请的申请数据
PCT/US2011/064487 2011.12.12
(87)PCT国际申请的公布数据
WO2012/134561 EN 2012.10.04
(73)专利权人 英特尔公司
地址 美国加利福尼亚州
(72)发明人 J·C·三额詹 B·托尔
R·C·凡伦天 M·B·吉尔卡尔
A·T·福塞斯 G·Z·克里斯沃斯
E·T·格罗科斯基
D·布拉德福德 L·K·吴
E·乌尔德-阿迈德-瓦尔
(74)专利代理机构 上海专利商标事务所有限公司 31100
代理人 何焜
(51)Int.Cl.
G06F 9/30(2006.01)
G06F 9/305(2006.01)
(56)对比文件
CN 101488084 A,2009.07.22,
JP 特开平5-274143 A,1993.10.22,
US 2010/0125720 A1,2010.05.20,
审查员 邢白灵

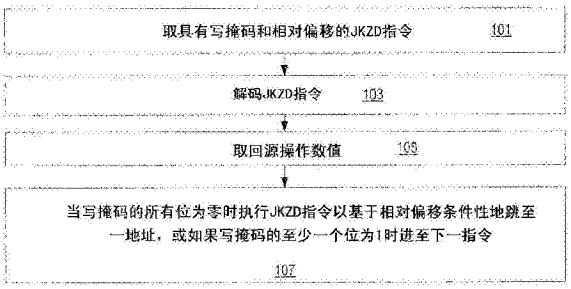
权利要求书2页 说明书26页 附图21页

(54)发明名称

使用掩码寄存器跳转的系统、装置和方法

(57)摘要

描述了在计算机处理器中执行跳转指令的系统、装置和方法的实施例。在一些实施例中，混合指令的执行使得当写掩码的所有位为零时条件性地跳转至目标指令的地址，其中目标指令的地址是使用所述指令的指令指针和相对偏移来计算的。



CN 103718157 B

1.一种在计算机处理器中执行如果写掩码为零则跳转至附近JKZD指令的方法，包括：

取出JKZD指令，其中所述JKZD指令包括写掩码操作数和相对偏移；

对所述取出的JKZD指令进行解码；以及

当所述写掩码操作数的所有位为零时执行所述取出的JKZD指令以条件性地跳转至目标指令的地址，其中所述目标指令的地址是使用所述JKZD指令的指令指针和所述相对偏移来计算的，其中所述写掩码操作数的每个位关联于作为控制流的实例的循环迭代，且所述写掩码操作数不是通用寄存器，并且所述写掩码操作数在每个数据元位置的基础上控制目的地中的数据元位置是否反映基操作和增量操作的结果。

2.如权利要求1所述的方法，其特征在于，所述写掩码操作数是16位寄存器。

3.如权利要求1所述的方法，其特征在于，所述相对偏移是8位立即值。

4.如权利要求1所述的方法，其特征在于，所述相对偏移是32位立即值。

5.如权利要求1所述的方法，其特征在于，所述JKZD指令的指令指针被存储在32位指令指针寄存器中。

6.如权利要求1所述的方法，其特征在于，所述JKZD指令的指令指针被存储在64位指令指针寄存器中。

7.如权利要求1所述的方法，其特征在于，所述执行还包括：

产生临时指令指针，所述临时指令指针是所述JKZD指令的指令指针加上所述相对偏移；

当所述临时指令指针不在包含JKZD指令的程序的代码段界限之外时，将所述临时指令指针设定为当前指令指针；以及

当所述临时指令指针在包含JKZD指令的程序的代码段界限之外时，产生出错，并且

所述写掩码操作数不是通用寄存器。

8.如权利要求7所述的方法，其特征在于，所述执行还包括：

当所述临时指令指针不在包含JKZD指令的程序的代码段界限之外时，当在将所述临时指令指针设定为所述目标指令的地址之前所述JKZD指令的操作数大小为16位时，对所述临时指令指针的前面两个字节清零。

9.一种在计算机处理器中执行如果写掩码不为零则跳转至附近JKNZD指令的方法，包括：

取出JKNZD指令，其中所述JKNZD指令包括写掩码操作数和相对偏移；

对所述取出的JKNZD指令进行解码；

当所述写掩码操作数的至少一个位不为零时执行所述取出的JKNZD指令以条件性地跳转至目标指令的地址，其中所述目标指令的地址是使用所述JKNZD指令的指令指针和所述相对偏移来计算的，其中所述写掩码操作数的每个位关联于作为控制流的实例的循环迭代，并且所述写掩码操作数在每个数据元位置的基础上控制目的地中的数据元位置是否反映基操作和增量操作的结果。

10.如权利要求9所述的方法，其特征在于，所述写掩码操作数是16位寄存器。

11.如权利要求9所述的方法，其特征在于，所述相对偏移是8位立即值。

12.如权利要求9所述的方法，其特征在于，所述相对偏移是32位立即值。

13.如权利要求9所述的方法，其特征在于，所述JKNZD指令的指令指针被存储在32位指

令指针寄存器中。

14.如权利要求9所述的方法，其特征在于，所述JKNZD指令的指令指针被存储在64位指令指针寄存器中。

15.如权利要求9所述的方法，其特征在于，所述执行还包括：

产生临时指令指针，所述临时指令指针是所述JKNZD指令的指令指针加上所述相对偏移；

当所述临时指令指针不在包含JKNZD指令的程序的代码段界限之外时，将所述临时指令指针设定为当前指令指针；以及

当所述临时指令指针在包含JKNZD指令的程序的代码段界限之外时，产生出错。

16.如权利要求15所述的方法，其特征在于，所述执行还包括：

当所述临时指令指针不在包含JKNZD指令的程序的代码段界限之外时，当在将所述临时指令指针设定为所述目标指令的地址之前所述指令的操作数大小为16位时，对所述临时指令指针的两个高字节清零。

17.一种使用掩码寄存器跳转的装置，包括：

多个写掩码寄存器，其中所述写掩码寄存器在每个数据元位置的基础上控制目的地中的数据元位置是否反映基操作和增量操作的结果；

硬件解码器，被配置为用于解码：

如果写掩码为零则跳转至附近JKZD指令，所述JKZD指令包括第一写掩码寄存器操作数和第一相对偏移，以及

如果写掩码不为零则跳转至附近JKNZD指令，其中所述JKNZD指令包括第二写掩码寄存器操作数和第二相对偏移；以及

执行逻辑，用于执行经解码的JKZD和JKNZD指令，其中对经解码的JKZD指令执行造成当所述第一写掩码寄存器操作数的所有位为零时条件性地跳转至第一目标指令的地址，所述第一目标指令的地址是使用所述JKZD指令的指令指针和所述第一相对偏移计算出的，而对经解码的JKNZD指令的执行造成当所述第二写掩码寄存器操作数的至少一个位不为零时条件性地跳转至第二目标指令的地址，所述第二目标指令的地址是使用所述JKNZD指令的指令指针和所述第二相对偏移计算出的，其中所述第一写掩码寄存器操作数的每个位关联于作为控制流信息的实例的循环迭代，且所述第一和第二写掩码寄存器操作数不是通用寄存器。

18.如权利要求17所述的装置，其特征在于，所述执行逻辑包括向量执行逻辑。

19.如权利要求17或18所述的装置，其特征在于，所述JKZD和JKNZD指令的所述第一和第二写掩码寄存器操作数是专用的16位寄存器。

20.如权利要求17或18所述的装置，其特征在于，所述JKZD和JKNZD指令的指令指针被存储在32位指令指针寄存器中。

使用掩码寄存器跳转的系统、装置和方法

发明领域

[0001] 本发明领域一般涉及计算机处理器体系结构，尤其涉及在被执行时导致特定结果的指令。

背景技术

[0002] 在程序执行期间编程者希望控制流改变的情形有很多次。历史上，已存在规定控制流改变的两种主要类型指令：分支和跳转。分支通常是相对于当前程序计数器的短改变的指示。跳转通常是指示程序计数器不直接关联于当前程序计数器的改变(例如跳转至一绝对存储器位置或使用动态或静态表来跳转)，并经常没有与当前程序计数器的距离限制。

附图说明

[0003] 在附图各图中通过示例而不是限制说明了本发明，其中相同标记指示相同元件，且其中：

[0004] 图1示出用于在处理器中执行JKZD指令的方法的实施例。

[0005] 图2示出在处理器中执行JKZD指令的另一实施例。

[0006] 图3示出用于在处理器中执行JKNZD指令的方法的实施例。

[0007] 图4示出在处理器中执行JKNZD指令的另一实施例。

[0008] 图5示出用于在处理器中执行JKOD指令的方法的实施例。

[0009] 图6示出在处理器中执行JKOD指令的另一实施例。

[0010] 图7示出用于在处理器中执行JKNOD指令的方法的实施例。

[0011] 图8示出在处理器中执行JKNOD指令的另一实施例。

[0012] 图9A是示出根据本发明实施例的一般向量友好指令格式及其类A指令模板的框图。

[0013] 图9B是根据本发明实施例示出一般向量友好指令格式及其类B指令模板的框图。

[0014] 图10A-C示出根据本发明实施例的示例性特定向量友好指令格式。

[0015] 图11是根据本发明一个实施例的寄存器架构的框图。

[0016] 图12A是根据本发明实施例的单CPU核连同它与芯片上互连网络以及其二级(L2)高速缓冲存储器的本地子集的连接的框图。

[0017] 图12B是根据本发明实施例的图12A中的CPU核的一部分的分解图。

[0018] 图13是示出根据本发明实施例的示例性无序架构的框图。

[0019] 图14是根据本发明的一个实施例的系统的框图。

[0020] 图15是根据本发明一个实施例的第二系统的框图。

[0021] 图16是根据本发明一个实施例的第三系统的框图。

[0022] 图17是根据本发明的实施例的SoC的框图。

[0023] 图18是根据本发明的实施例的具有集成的存储器控制器和图形的单核处理器和多核处理器的框图。

[0024] 图19是根据本发明实施例对照软件指令转换器的使用将源指令集中的二进制指令转换成目标指令集中的二进制指令的框图。

具体实施方式

[0025] 在以下描述中，陈述了多个具体细节。然而，应当理解，本发明的实施例可在没有这些具体细节的情况下实践。在其他实例中，未详细示出公知的电路、结构和技术以免混淆对本描述的理解。

[0026] 在说明书中对“一个实施例”、“实施例”、“示例实施例”等的引用表明所描述的实施例可包括特定特征、结构或特性，但不一定每个实施例均包括该特定特征、结构或特性。此外，这样的短语不一定是指同一个实施例。此外，当结合实施例描述特定特征、结构或特性时，认为本领域技术人员知道结合无论是否明显描述的其它实施例来实现这些特征、结构或特性。

[0027] 跳转指令

[0028] 下面详细示出若干跳转指令的若干实施例以及用来执行这些指令的系统、架构、指令格式等的实施例。这些跳转指令可用来基于指令所包含的写掩码(writemask)值条件性地改变程序的控制流序列。这些指令利用“写掩码”改变向量化代码的控制流，其中掩码的每个位关联于控制流信息的一个SIMD存档的实例——循环迭代。下面详细描述写掩码实施例的细节。

[0029] 下面的跳转指令的典型使用包括：具有动态收敛的循环的提早退出；迭代直到所有活动元素断开(例如运动估计菱形搜索和有限差算法)；当掩码为零时抑制假存储器出错；改善的聚集/分散指令的性能；以及对稀疏分布的预测代码节省工作(例如编译器无法承担存储器内的压缩/扩展)。

[0030] 基于写掩码的控制流的多数实例是下面任何一个：当写掩码全部为零时跳转或当掩码不是全部为零时跳转。示出示例性高级语言伪代码及其伪汇编对应物的表格如下所示。VCMPPS指令将源寄存器ZMM1和ZMM2的数据元作比较，且如果ZMM1的数据元小于ZMM2的相应数据元，则将它们作为“掩码”位存储在写掩码k1中。当然，VCMPPS不仅限于这种情况并可基于其它条件(例如等于、小于等于、无序的、不等于、不小于、不小于或等于或有序的)。

[0031]

伪代码	JNZ 方法

for(i=0; i<16; i++)
{
not_finished = TRUE;
while(not_finished)
{
a[i] = a[i] – b[i];
if(a[i] < b[i]) not_finished = FALSE;
}
}z
	loop_not_finished:
	VMOVAPS zmm1, a // 加载 a
	VMOVAPS zmm2, b // 加载 b
	VSUBPS zmm1, zmm1, zmm2
	// a[i] = a[i] – b[i]
	VCMPPS k1, zmm1, zmm2, LT
	// k1[i] = (a[i]<b[i])? 1 : 0
	KORTESTD k1, k1
	JNZ loop_not_finished

[0032] 表1

[0033] 对于这一序列的JNZ方法相对慢并在写掩码已产生之后需要两个指令两次跳出循环：

[0034] KORTEST k1,k1//(OR(k1,k1)==0x0)=>ZF

[0035] JNZ target_addr

[0036] KORTEST指令执行两个掩码的“OR”操作，如果结果为零，则将处于“条件代码”或状态寄存器中的零标志(例如FLAG或EFLAG)置位。JNZ(非零跳转)指令寻找该标志，并且如果零标志已被置位则跳转至目标地址。因此，有机会减少吞吐量以及(在未来)减少对该软件序列的等待时间。

[0037] JKZD-如果写掩码为零则跳转至附近

[0038] 要讨论的第一指令是如果写掩码为零则跳转至附近(JKZD)。该指令通过处理器的执行使源写掩码的值被检查以查看是否其所有的写掩码位都被置为“0”，如果是，则使处理器执行至目标指令的跳转，该目标指令至少部分地由目的地操作数和当前指令指针规定。如果所有写掩码位不为“0”(并因此不满足跳转条件)，则不执行任何跳转并继续执行JKZD指令之后的指令。

[0039] JKZD的目标指令的地址一般是通过包含在该指令中的相对偏移操作数(相对于EIP寄存器中的指令指针的当前值的带符号偏移)规定的。相对偏移(rel8、rel16或rel32)一般被规定为汇编代码中的标签，但在机器代码层面，它可被编码为带符号的8位或32位立即值(immediate value)，该8位或32位立即值被加至指令指针。典型地，指令编码对于-128至127的偏移最为有效。在一些实施例中，如果操作数大小(指令指针)为16位，则EIP寄存器的前两个字节不被使用(清零)以产生目标指令地址。在一些实施例中，在具有64位操作数大小的64位模式(RIP存储指令指针)下，短跳转的目标指令地址被定义为RIP=RIP+扩展至64位的8位偏移符号。在这种模式下，目标地址附近的跳转被定义为RIP=RIP+扩展至64位的32位偏移。

[0040] 该指令的一示例格式为“JKZD k1,rel8/32”，其中k1是写掩码操作数(例如类似于

之前详细描述的16位寄存器)而rel8/32是8或32位的立即值。在一些实施例中,写掩码具有不同大小(8位、32位等)。JKZD是指令的操作码。典型地,每个操作数被显式地定义在指令中。在其它实施例中,立即值是例如16位的不同大小。

[0041] 图1示出在处理器中执行JKZD指令的方法的实施例。在101获取包括写掩码和相对偏移的JKZD指令。

[0042] 在103对JKZD指令解码并在105取回例如写掩码之类的源操作数值。

[0043] 在107执行经解码的JKZD指令,这使得当写掩码的所有位为零时条件性跳转至从相对偏移和当前指令指针产生的地址处的指令,或如果写掩码的至少一个位为1则使JKZD指令之后的指令被取出、解码等。地址的产生可发生在该方法的解码、获取或执行阶段的任一阶段中。

[0044] 图2示出在处理器中执行JKZD指令的另一实施例。假设在该方法开始之前已执行了101-105中的一些并且这些步骤不再被示出以避免混淆接下来的细节。在201,作出写掩码中是否存在任何“1”值的判断。

[0045] 如果写掩码中存在“1”(并因此写掩码不为零),则在203不执行跳转并执行程序流中的连续指令。如果写掩码中没有“1”,则在205产生临时指令指针。在一些实施例中,该临时指令指针是当前指令指针加上符号扩展的相对偏移。例如,对于32位指令指针,临时指令指针的值为EIP加上符号扩展的相对偏移。该临时指令指针可被存储在寄存器中。

[0046] 在207作出操作数大小属性是否为16位的判断。例如,指令指针是16位、32位还是64位的值。如果操作数大小属性为16位,则在209临时指令指针的前两个字节被清零(置为零)。清零可以若干不同的方式发生,但在一些实施例中,将临时指令指针与具有两个最高有效字节为“0”且两个最低有效字节为“1”的立即数(例如该立即数是0x0000FFFF)作逻辑与操作。

[0047] 如果操作数大小不是16位,则在211作出该临时指令指针是否落在代码段界限内的判断。

[0048] 如果不是,则在213产生出错并且不执行跳转。对两个最高有效字节被清零的临时指令指针也作这样的判断。在指令不支持远跳转(跳转至其它代码段)的一些实施例中,当条件性跳转的目标在不同代码段中时,使用与针对JKZD指令进行测试的条件相反的条件,然后通过至其它代码段的无条件远跳转(JMP指令)访问该目标。在具有跳转限制的实施例中,如果程序想要跳转至代码的远区,则写掩码跳转的语义被否定以使后续(follow-through)代码作出进入特定代码的“远”跳转。例如,该条件可能是非法的:

[0049] JKZD FARLABEL;

[0050] 为了完成这种远跳转,可代替地使用下面两个指令:

[0051] JKNZD BEYOND;

[0052] JMP FARLABEL;

[0053] BEYOND:

[0054] 如果临时指令指针落在代码段界限内,则在213将指令指针被置为临时指令指针。例如,可将EIP值置为该临时指令指针。在215作出跳转。

[0055] 最后,在一些实施例中,该方法的前述一个或多个方面不被执行或以不同顺序执行。例如,如果处理器不具有16位操作数(指令指针),则该决策不会发生。

[0056] 表2示出表1的相同伪代码，但它利用JKNZD指令并省去了KORTESTD的需要。对于下列指令可产生相同的益处。

[0057]

伪代码	JNZ 方法

for(i=0; i<16; i++)
{
not_finished = TRUE;
while(not_finished)
{
a[i] = a[i] – b[i];
if(a[i] < b[i]) not_finished = FALSE;
}
}
	loop_not_finished:
	VMOVAPS zmm1, a // 加载 a
	VMOVAPS zmm2, b // 加载 b
	VSUBPS zmm1, zmm1, zmm2
	// a[i] = a[i] – b[i]
	VCMPPS k1, zmm1, zmm2, LT
	// k1[i] = (a[i]<b[i])? 1 : 0
	JKNZD k1, loop_not_finished

[0058] 表2

[0059] JKNZD-如果写掩码不为零则跳转至附近

[0060] 要讨论的第二指令是如果写掩码不为零则跳转至附近(JKNZD)。该指令通过处理器的执行使源写掩码的值被检查以查看是否其所有的写掩码位都被置为“0”，如果否，则使处理器执行至目标指令的跳转，该目标指令至少部分地由目的地操作数和当前指令指针规定。如果所有写掩码位为“0”(并因此不满足跳转条件)，则不执行任何跳转并继续执行JKNZD指令之后的指令。

[0061] JKNZD的目标指令的地址一般是通过包含在该指令中的相对偏移操作数(相对于EIP寄存器中的指令指针的当前值的带符号偏移)规定的。相对偏移(rel8、rel16或rel32)一般被规定为汇编代码中的标签，但在机器代码层面，它可被编码为带符号的8位或32位立即值，该8位或32位立即值被加至指令指针。典型地，指令编码对于-128至127的偏移最为有效。在一些实施例中，如果操作数大小(指令指针)为16位，则EIP寄存器的前两个位不被使用(清零)以产生目标指令地址。在一些实施例中，在具有64位操作数大小的64位模式(RIP存储指令指针)下，短跳转的目标指令地址被定义为RIP=RIP+扩展至64位的8位偏移符号。在这种模式下，跳转至附近(jump near)的目标地址被定义为RIP=RIP+扩展至64位的32位偏移。

[0062] 该指令的一示例格式为“JKNZD k1,rel8/32”，其中k1是写掩码操作数(例如类似于之前详细描述的16位寄存器)，而rel8/32是8或32位的立即值。在一些实施例中，写掩码具有不同大小(8位、32位等)。JKBZD是指令的操作码。典型地，每个操作数被显式地定义在指令中。在其它实施例中，立即值是例如16位的不同大小。

[0063] 图3示出用于在处理器中执行JKNZD指令的方法的实施例。在301取出包括写掩码和相对偏移的JKNZD指令。

[0064] 在303对JKNZD指令解码并在305取回例如写掩码之类的源操作数值。

[0065] 在307执行经解码的JKNZD指令，这使得当写掩码的所有位为零时条件性地跳转至从相对偏移和当前指令指针产生的地址处的指令，或如果写掩码的至少一个位为1则使JKNZD指令之后的指令被取出、解码等。地址的产生可发生在该方法的解码、取回或执行阶段的任一阶段中。

[0066] 图4示出在处理器中执行JKNZD指令的另一实施例。假设在该方法开始之前已执行了401-405中的一些，并且这些步骤不再被示出以避免混淆接下来的细节。在401，作出写掩码中是否存在任何“1”值的判断。

[0067] 如果写掩码中仅存在“0”(并因此写掩码为零)，则在403不执行跳转并执行程序流中的连续指令。如果写掩码中存在“1”，则在405产生临时指令指针。在一些实施例中，该临时指令指针是当前指令指针加上符号扩展的相对偏移。例如，对于32位指令指针，临时指令指针的值为EIP加上符号扩展的相对偏移。该临时指令指针可被存储在寄存器中。

[0068] 在407作出操作数大小属性是否为16位的判断。例如，指令指针是16位、32位还是64位的值。如果操作数大小属性为16位，则在409，临时指令指针的前两个字节被清零(置为零)。清零可以若干不同的方式发生，但在一些实施例中，将临时指令指针与具有两个最高有效字节为“0”且两个最低有效字节为“1”的立即数(例如该立即数是0x0000FFFF)作逻辑与操作。

[0069] 如果操作数大小不是16位，则在411作出该临时指令指针是否落在代码段界限内的判断。如果不是，则在413产生出错并且不执行跳转。对两个最高有效字节被清零的临时指令指针也可作这样的判断。在指令不支持远跳转(跳转至其它代码段)的一些实施例中，当条件性跳转的目标在不同代码段中时，使用与针对JKNZD指令所测试的该条件相反的条件，然后通过至其它代码段的无条件远跳转(JMP指令)访问该目标。例如，该条件可能是非法的：

[0070] JKNZD FARLABEL;

[0071] 为了完成这种远跳转，可代替地使用下面两个指令：

[0072] JKZD BEYOND;

[0073] JMP FARLABEL;

[0074] BEYOND:

[0075] 如果临时指令指针落在代码段界限内，则在413将指令指针置为该临时指令指针。例如，可将EIP值置为临时指令指针。在415作出跳转。

[0076] 最后，在一些实施例中，该方法的前述一个或多个方面不被执行或以不同顺序执行。例如，如果处理器不具有16位操作数(指令指针)，则该决策不会发生。

[0077] JKOD-如果写掩码全部为1则跳转至附近

[0078] 要讨论的第三指令是如果写掩码全部为1则跳转至附近(JKOD)。该指令通过处理器的执行使源写掩码的值被检查以查看是否其所有的写掩码位都被置为“1”，如果是，则使处理器执行至目标指令的跳转，该目标指令至少部分地由目的地操作数和当前指令指针规定。如果所有写掩码位不为“1”(并因此不满足跳转条件)，则不执行任何跳转并继续执行JKOD指令之后的指令。

[0079] JKOD的目标指令的地址一般是通过包含在该指令中的相对偏移操作数(相对于EIP寄存器中的指令指针的当前值的带符号偏移)规定的。相对偏移(rel8、rel16或rel32)

一般被规定为汇编代码中的标签，但在机器代码层面，它可被编码为带符号的8位或32位立即值，该8位或32位立即值被加至指令指针。典型地，指令编码对于-128至127的偏移最为有效。在一些实施例中，如果操作数大小(指令指针)为16位，则EIP寄存器的前两个位不被使用(清零)以产生目标指令地址。在一些实施例中，在具有64位操作数大小的64位模式(RIP存储指令指针)，短跳转的目标指令地址被定义为RIP=RIP+扩展至64位的8位偏移符号。在这种模式下，跳转至附近的目标地址被定义为RIP=RIP+扩展至64位的32位偏移。

[0080] 该指令的一示例格式为“JKOD k1,rel8/32”，其中k1是写掩码操作数(例如类似于之前详细描述的16位寄存器)而rel8/32是8或32位的立即值。在一些实施例中，写掩码具有不同大小(8位、32位等)。JKOD是指令的操作码。典型地，每个操作数被显式地定义在指令中。在其它实施例中，立即值是例如16位的不同大小。

[0081] 图5示出用于在处理器中执行JKOD指令的方法的实施例。在501取出包括写掩码和相对偏移的JKOD指令。

[0082] 在503对JKOD指令解码并在505取回例如写掩码之类的源操作数值。

[0083] 在507执行经解码的JKOD指令，这使得当写掩码的所有位为1时条件性跳转至从相对偏移和当前指令指针产生的地址处的指令，或如果写掩码的至少一个位为0则使JKOD指令之后的指令被取出、解码等。地址的产生可发生在该方法的解码、取回或执行阶段的任一阶段中。

[0084] 图6示出在处理器中执行JKOD指令的另一实施例。假设在该方法开始之前已执行了601-605中的一些并且这些步骤不再被示出以避免混淆接下来的细节。在601，作出写掩码中是否存在任何“0”值的判断。

[0085] 如果写掩码中存在“0”(并因此写掩码不全为1)，则在603不执行跳转并执行程序流中的连续指令。如果写掩码中没有“0”，则在605产生临时指令指针。在一些实施例中，该临时指令指针是当前指令指针加上符号扩展的相对偏移。例如，对于32位指令指针，临时指令指针的值为EIP加上符号扩展的相对偏移。该临时指令指针可被存储在寄存器中。

[0086] 在607作出操作数大小属性是否为16位的判断。例如，指令指针是16位、32位还是64位的值。如果操作数大小属性为16位，则在609临时指令指针的前两个字节被清零(置为零)。清零可以若干不同的方式发生，但在一些实施例中，将临时指令指针与两个最高有效字节为“0”且两个最低有效字节为“1”的立即数(例如该立即数是0x0000FFFF)作逻辑与操作。

[0087] 如果操作数大小不是16位，则在611作出该临时指令指针是否落在代码段界限内的判断。如果不是，则在613产生出错并且不执行跳转。对两个最高有效字节被清零的临时指令指针也作这样的判断。

[0088] 如果临时指令指针落在代码段界限内，则在613将指令指针被置为临时指令指针。例如，可将EIP值置为临时指令指针。在615作出跳转。

[0089] 最后，在一些实施例中，该方法的前述一个或多个方面不被执行或以不同顺序执行。例如，如果处理器不具有16位操作数(指令指针)，则该判决不会发生。

[0090] JKNOD-如果写掩码不全为1则跳转至附近

[0091] 最后要讨论的指令是如果写掩码不全为1则跳转至附近(JKNOD)。该指令通过处理器的执行使源写掩码的值被检查以查看是否其至少一个写掩码位被置为“0”，如果是，则使

处理器执行至目标指令的跳转，该目标指令至少部分地由目的地操作数和当前指令指针规定。如果没有一个写掩码位为“0”(并因此不满足跳转条件)，则不执行任何跳转并继续执行JKNOD指令之后的指令。

[0092] JKNOD的目标指令的地址一般是通过包含在该指令中的相对偏移操作数(相对于EIP寄存器中的指令指针的当前值的带符号偏移)规定的。相对偏移(rel8、rel16或rel32)一般被规定为汇编代码中的标签，但在机器代码层面，它可被编码为带符号的8位或32位立即值，该8位或32位立即值被加至指令指针。典型地，指令编码对于-128至127的偏移最为有效。在一些实施例中，如果操作数大小(指令指针)为16位，则EIP寄存器的前两个位不被使用(清零)以产生目标指令地址。在一些实施例中，在具有64位操作数大小的64位模式(RIP存储指令指针)下，短跳转的目标指令地址被定义为RIP=RIP+扩展至64位的8位偏移符号。在这种模式下，跳转至附近的目标地址被定义为RIP=RIP+扩展至64位的32位偏移。

[0093] 该指令的一示例格式为“JKNOD k1,rel8/32”，其中k1是写掩码操作数(例如类似于之前详细描述的16位寄存器)而rel8/32是8或32位的立即值。在一些实施例中，写掩码具有不同大小(8位、32位等)。JKNOD是指令的操作码。典型地，每个操作数被显式地定义在指令中。在其它实施例中，立即值是例如16位的不同大小。

[0094] 图7示出用于在处理器中执行JKNOD指令的方法的实施例。在701获取包括写掩码和相对偏移的JKNOD指令。

[0095] 在703对JKNOD指令解码并在305取回例如写掩码之类的源操作数值。

[0096] 在307执行经解码的JKNOD指令，这使得当写掩码的至少一个位为不为1时条件性地跳转至从相对偏移和当前指令指针产生的地址处的指令，或如果写掩码的所有位为1则使JKNZD指令之后的指令被取出、解码等。地址的产生可发生在该方法的解码、取回或执行阶段的任一阶段中。

[0097] 图8示出在处理器中执行JKNOD指令的另一实施例。假设在该方法开始之前已执行了701-705中的一些并且这些步骤不再被示出以避免混淆接下来的细节。在801，作出写掩码中是否存在任何“0”值的判断。

[0098] 如果写掩码中不存在“0”(并因此写掩码全为1)，则在803不执行跳转并执行程序流中的连续指令。如果写掩码中存在“0”，则在805产生临时指令指针。在一些实施例中，该临时指令指针是当前指令指针加上符号扩展的相对偏移。例如，对于32位指令指针，临时指令指针的值为EIP加上符号扩展的相对偏移。该临时指令指针可被存储在寄存器中。

[0099] 在807作出操作数大小属性是否为16位的判断。例如，指令指针是16位、32位还是64位的值。如果操作数大小属性为16位，则在809临时指令指针的前两个字节被清零(置为零)。清零可以若干不同的方式发生，但在一些实施例中，将临时指令指针与两个最高有效字节为“0”且两个最低有效字节为“1”的立即数(例如该立即数是0x0000FFFF)作逻辑与操作。

[0100] 如果操作数大小不是16位，则在811作出该临时指令指针是否落在代码段界限内的判断。如果不是，则在813产生出错并且不执行跳转。对两个最高有效字节被清零的临时指令指针也作这样的判断。

[0101] 如果临时指令指针落在代码段界限内，则在813将指令指针被置为临时指令指针。例如，可将EIP值置为临时指令指针。在815作出跳转。

[0102] 最后，在一些实施例中，该方法的前述一个或多个方面不被执行或以不同顺序执行。例如，如果处理器不具有16位操作数(指令指针)，则该决策不会发生。

[0103] 前面详细描述的指令体现的实施例可以下面具体描述的“一般向量友好指令格式”来体现。在其它实施例中，不利用这种格式并使用另一指令格式，然而，下面对写掩码寄存器、各种数据变换(混合(swizzle)、广播等)、寻址等的描述一般适用于前述指令的实施例的描述。另外，下面详细描述示例性系统、架构和流水线。前述指令的实施例可在这些系统、架构和流水线上执行，但不仅限于那些详细说明的内容。

[0104] 向量友好指令格式是适用于向量指令的指令格式(例如存在专门针对向量操作的某些字段)。尽管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，然而其它实施例仅使用通过向量友好指令格式支持的向量操作。

[0105] 示例性一般向量友好指令格式——图9A-9B

[0106] 图9A-9B是示出根据本发明实施例的一般向量友好指令格式及其指令模板的框图。图9A是示出根据本发明实施例的一般向量友好指令格式及其类A指令模板的框图；而图9B是示出根据本发明实施例的一般向量友好指令格式及其类B指令模板的框图。具体地说，对其定义类A和类B指令模板的一般向量友好指令格式900，这两个类A、B均包括无存储器访问905指令模板和存储器访问920指令模板。术语“一般”在向量友好指令格式的背景下表示不关系于任何特定指令集的指令格式。尽管将要描述按照向量友好指令格式的指令工作在源自寄存器(无存储器访问905指令模板)或寄存器/存储器(存储器访问920指令模板)的向量上的实施例，然而本发明的其它实施例也可仅支持这些中的一个。另外，尽管将要描述在向量指令格式中存在加载和存储指令的本发明实施例，然而作为替代或附加，其它实施例具有不同指令格式的指令，这些指令将向量移入和移出寄存器(例如从存储器移入寄存器，从寄存器移入存储器，在两寄存器之间移动)。此外，尽管将要描述支持两类指令模板的本发明实施例，然而其它实施例可仅支持这两类指令模板中的一个或两类以上的指令模板。

[0107] 尽管将要描述其中向量友好指令格式支持下列各项的本发明实施例：具有32位(4字节)的64字节向量操作数长度(或大小)或64位(8字节)数据元宽度(或大小)(并因此64字节向量由16个双字大小数据元或8个四字大小数据元构成)；具有16位(2字节)或8位(1字节)数据元宽度(或大小)的64字节向量操作数长度(或大小)；具有32位(4字节)、64位(8字节)、16位(2字节)或8位(1字节)数据元宽度(或大小)的32字节向量操作数长度(或大小)；以及具有32位(4字节)、64位(8字节)、16位(2字节)或8位(1字节)数据元宽度(或大小)的16字节向量操作数长度(或大小)；但其它实施例可支持具有更多、更少或不同的数据元宽度(例如128位(16字节)数据元宽度)的更多、更少和/或不同的向量操作数大小(例如956字节向量操作数)。

[0108] 图9A中的类A指令模板包括：1)在无存储器访问905指令模板中，如图所示地存在无存储器访问、完全舍入(full round)控制类操作910指令模板和无存储器访问数据转换类操作915指令模板；以及2)在存储器访问920指令模板内，如图所示地存在存储器访问、时间性的925指令模板和存储器访问、非时间性的930指令模板。图9B中的类B指令模板包括：在无存储器访问905指令模板中，如图所示地存在无存储器访问、写掩码控制、部分舍入(partial round)控制类型操作912指令模板和无存储器访问、写掩码控制、vsize类操作917指令模板；以及2)在存储器访问920指令模板中，图示为存在存储器访问、写掩码控制

927指令模板。

[0109] 格式

[0110] 一般向量友好指令格式900包括下文以图9A-9B所示的顺序列出的以下字段。

[0111] 格式字段940——该字段中的特定值(指令格式标识符值)唯一地标识向量友好指令格式,并因此指令在指令流中以向量友好指令格式出现。因此,格式字段940的内容将按照第一指令格式的指令的出现与按照其它指令格式的指令的出现区别开,由此允许将向量友好指令格式引入到具有其它指令格式的指令集中。如此,该字段是可选用的,因为不需要仅有一般向量友好指令格式的指令集。

[0112] 基操作字段942——其内容区分不同的基操作。如本文下面描述的,基操作字段942可包括操作码字段和或是操作码字段的一部分。

[0113] 寄存器索引字段944——其内容直接地或通过地址产生,规定了源操作数和目的地操作数的位置,如果它们在寄存器或存储器中的话。这些包括足够数目的位以从PxQ(例如32x1112)寄存器文件中选择N个寄存器。尽管在一个实施例中,N多达三个源寄存器和一个目的地寄存器,然而其它实施例可支持更多或更少的源寄存器和目的地寄存器(例如可支持多达两个源,其中这些源中的一个也充当目的地,可支持多达三个源,其中这些源中的一个也可充当目的地,可支持两个以上的源和一个目的地)。尽管在一个实施例中,P=32,然而其它实施例可支持更多或更少的寄存器(例如16个)。尽管在一个实施例中,Q=1112位,然而其它实施例可支持更多或更少的位(例如128、1024个位)。

[0114] 修饰符字段946——其内容将规定存储器访问的按照一般向量指令格式的指令的出现与不规定存储器访问的指令的出现区分开;也就是在无存储器访问905指令模板和存储器访问920指令模板之间进行区分。存储器访问操作读和/或写至存储器阶层(在一些情形下使用寄存器中的值规定源和/或目的地地址),而非存储器访问操作不那样(例如源和目的地是寄存器)。尽管在一个实施例中,该字段也在三种不同的方式之间作出选择以执行存储器地址计算,但其它实施例可支持更多、更少或不同的方式以执行存储器地址计算。

[0115] 增量操作字段950——其内容区分除了基操作外的多种不同操作中的哪一个将被执行。该字段是专门针对上下文的。在本发明的一个实施例中,该字段被分成类字段968、α字段952和β字段954。增量操作字段允许在单个指令中而不是2、3或4个指令中执行共同的多组操作。下面是使用增量字段950减少所需指令数的指令的一些例子(其命名将在下文中更详细地予以描述)

[0116]

优先指令序列	根据本发明一个实施例的指令序列
vaddps ymm0, ymm1, ymm2	vaddps zmm0, zmm1, zmm2
vpshufd ymm2, ymm2, 0x55 vaddps ymm0, ymm1, ymm2	vaddps zmm0, zmm1, zmm2 {bbbb}
vpmovsxbd ymm2, [rax] vcvtdq2ps ymm2, ymm2 vaddps ymm0, ymm1, ymm2	vaddps zmm0, zmm1, [rax]{sint8}
vpmovsxbd ymm3, [rax] vcvtdq2ps ymm3, ymm3 vaddps ymm4, ymm2, ymm3 vblendvps ymm1, ymm5, ymm1, ymm4	vaddps zmm1{k5}, zmm2, [rax]{sint8}
vmaskmovps ymm1, ymm7, [rbx] vbroadcastss ymm0, [rax] vaddps ymm2, ymm0, ymm1 vblendvps ymm2, ymm2, ymm1, ymm7	vmovaps zmm1 {k7}, [rbx] vaddps zmm2{k7}{z}, zmm1, [rax]{1toN}

[0117] 其中[rax]是用于地址产生的基础指针，而{}表示由数据操纵字段(这将在下文中更详细地描述)规定的转换操作。

[0118] 定标字段960——其内容允许对用于存储器地址产生的索引字段的内容进行定标(例如使用$2^{定标*}$索引+基地址的地址产生)。

[0119] 位移字段962A——其内容被用作存储器地址产生的一部分(例如使用$2^{定标*}$索引+基地址+位移的地址产生)。

[0120] 位移因子字段962B(注意位移字段962A直接并列在位移因子字段962B上表示使用一个或其它)——其内容被用作地址产生的一部分;它规定拟由存储器访问(N)的大小定标的位移因子——其中N是存储器访问的字节数(例如使用$2^{定标*}$索引+基地址+定标的位移的地址产生)冗余的低阶位被省略并因此位移因数字段的内容与存储器操作数总大小(N)相乘以产生在计算有效地址时使用的最终位移。N的值通过处理器硬件在运行时间基于完整操作码字段974(本文后面描述的)和数据操纵字段954C来确定，如本文后面描述的那样。位移字段962A和位移因子字段962B是可选用的，因为它们不用于无存储器访问905指令模板和/或不同的实施例可仅采用前面这两个字段中的一个或者一个也不采用。

[0121] 数据元宽度字段964——其内容辨别大量数据元宽度中的哪一个被使用(在一些实施例中针对所有指令，在其它实施例中仅针对一些指令)。该字段是可选用的，因为如果

利用操作码的一些方面仅支持一个数据元宽度和/或支持多个数据元宽度，则该字段是不需要的。

[0122] 写掩码字段970——其内容在每个数据元位置基础上控制该目的地向量操作数中的数据元位置是否反映基操作和增量操作的结果。类A指令模板支持融合-写掩码操作，而类B指令模板既支持融合-写掩码操作又支持归零-写掩码操作。当融合时，向量掩码允许目的地中的任何组的数据元受到保护而不会在任何操作的执行过程中更新(由基操作和增量操作规定)；在另外一个实施例中，在相应的掩码位具有0的情况下保留目的地的每个数据元的旧值。相反，当归零时，向量掩码允许目的地中的任何组的元素在任何操作执行过程中归零(由基操作和增量操作规定)；在一个实施例中，当相应的掩码位具有0值时将目的地的元素置为0。该功能的子集是控制正被执行的操作的向量长度(即被修改的元素——从第一个至最后一个——的跨距)的能力；然而，被修改的这些元素不一定是连续的。因此，写掩码字段970允许部分向量操作，包括加载、存储、算术、逻辑等。另外，该掩码操作可用于故障抑制(即通过对目的地的数据元位置进行掩码以防止接收到可能/将造成出错的任何操作的结果——例如，假设存储器中的向量横跨页边界并且是第一页而不是第二页将造成页出错，如果位于第一页上的向量的全部数据元通过写掩码被掩码，则可忽略页出错)。此外，写掩码允许包含某些类型的条件性语句的“向量化循环”。尽管在所描述的本发明实施例中，写掩码字段970的内容选择包含拟使用的写掩码的数个写掩码寄存器中的一个(并因此该写掩码字段970内容间接地标识拟被执行的掩码操作)，然而作为替代或附加，其它实施例允许掩码写字段970内容直接规定拟被执行的掩码操作。此外，当发生下列情形时，归零允许性能提升：1)寄存器重命名被用在其目的地操作数也不是源的指令上(也称非三重指令)，因为在寄存器重命名流水线阶段期间，目的地不再是隐含的源(没有任何来自当前目的地寄存器的数据元需要被复制至重命名的目的地寄存器或以某种方式连同操作地携带，因为不是操作结果的任何数据元(任何被掩码的数据元)将被归零)；以及2)在回写阶段期间，因为零正被写入。

[0123] 立即数字段972——其内容允许规定一立即数。该字段是可选的，因为它不出现在不支持立即数的一般向量友好格式的实现中而且也不出现在不使用立即数的指令中。

[0124] 指令模板类选择

[0125] 类字段968，其内容在不同的指令类之间有区别。参见图2A-B，该字段的内容在类A和类B指令之间选择。在图9A-B中，带圆角的方形被用来表示字段中存在特定值(例如在图9A-B中分别针对类字段968的类A968A和类B968B)。

[0126] 类A的无存储器访问指令模板

[0127] 在类A的无存储器访问905指令模板的情形下，α字段952被解释为RS字段952A，其内容辨别不同增量操作类型中的哪一个要被执行(例如舍入952A.1和数据转换952A.2分别是针对无存储器访问舍入类型操作910和无存储器访问数据转换类型操作915指令模板来规定的)，而β字段954区别哪个指定类型的操作要被执行。在图9中，圆角块被用来表示特定值存在(例如修饰符字段946中的无存储器访问946A；对于α字段952/rs字段952A的舍入952A.1和数据转换952A.2)。在无存储器访问905指令模板中，定标字段960、位移字段962A和位移定标字段962B是不存在的。

[0128] 无存储器访问指令模板——完全舍入控制类型操作

[0129] 在无存储器访问完全舍入控制类型操作910指令模板中，β字段954被解释成舍入控制字段954A，其内容提供静态舍入。尽管在本发明的所述实施例中，舍入控制字段954A包括抑制所有的浮点异常(SAE)字段956和舍入操作控制字段958，然而替代实施例可支持这些改变并将这些概念编码到同一字段中，或仅具有这些概念/字段中的一个或其它(例如可仅具有舍入操作控制字段958)。

[0130] SAE字段956——其内容区分是否禁用异常事件报告；当SAE字段956内容指示抑制被启用时，给定的指令不报告任何类型的浮点异常标志并且不唤起任何浮点异常处理程序。

[0131] 舍入操作控制字段958——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，向零舍入和向最近舍入)。因此，舍入操作控制字段958允许在每个指令的基础上改变舍入模式，并因此当需要时尤为有用。在处理器包括用于指定舍入模式的控制寄存器的本发明的一个实施例中，舍入操作控制字段950内容超越寄存器值(能够选择舍入模式而不必在该控制寄存器上执行保存-修改-恢复是有利的)。

[0132] 无存储器访问指令模板——数据转换类型操作

[0133] 在无存储器访问数据转换类型操作915指令模板中，β字段954被解释成数据转换字段954B，其内容区分多个数据转换中的哪一个要被执行(例如没有数据转换、混合、广播)。

[0134] 类A的存储器访问指令模板

[0135] 在类A的存储器访问920指令模板的情形下，α字段952被解释成逐出提示字段952B，其内容区分要使用多个逐出提示中的哪一个(在图9A中，针对存储器访问时间性925指令模板和存储器访问非时间性930指令模板分别规定时间性952B.1和非时间性952B.2)，而β字段954被解释成数据操纵字段954C，其内容区分众多数据操纵操作(也称原始操作)中的哪一个要被执行(例如没有操纵、广播、源的向上转换以及目的地的向下转换)。存储器访问920指令模板包括定标字段960、可选用的位移字段962A或位移定标字段962B。

[0136] 向量存储器指令通过转换支持执行来自存储器的向量装载和至存储器的向量存储。如同常规向量指令，向量存储器指令以逐个数据元的方式将数据转移出/转移至存储器，这些数据元实际上是通过被选择为写掩码的向量掩码的内容规定而转移的。在图9A中，使用圆角方形来表示字段中存在特定值(例如对于修饰符字段946的存储器访问946B、对于α字段952/逐出提示字段952B的时间性952B.1和非时间性952B.2)。

[0137] 存储器访问指令模板——时间性

[0138] 时间性数据是很快就可能再使用从而足以从高速缓存中获益的数据。然而，这是一个提示并且不同的处理器可能以不同方式实现，包括完全地忽略这种提示。

[0139] 存储器访问指令模板——非时间性

[0140] 非时间性数据是不大可能很快地再使用从而足以从一级高速缓冲存储器的高速缓存中获益的数据，并应当被给予逐出的优先级。然而，这是一个提示并且不同的处理器可能以不同方式实现，包括完全地忽略这种提示。

[0141] 类B的指令模板

[0142] 在类B的指令模板的情形下，α字段952被解释成写掩码控制(Z)字段952C，其内容区分受写掩码字段970控制的写掩码操作是否应当是融合的或归零的。

[0143] 类B的无存储器访问指令模板

[0144] 在类B的无存储器访问905指令模板的情形下，β字段954的一部分被解释为RL字段957A，其内容辨别不同增量操作类型中的哪一个要被执行(例如分别针对无存储器访问写掩码控制部分舍入控制类型操作912指令模板和无存储器访问写掩码控制VSIZE类型操作917指令模板规定舍入957A.1和向量长度(VSIZE)957A.2)，β字段中的其余部分辨别哪个指定类型的操作要被执行。在图9中，圆角块被用来表示特定值存在(例如修饰符字段946中的无存储器访问946A；对于RL字段957A的舍入957A.1和VSIZE957A.2)。在无存储器访问905指令模板中，定标字段960、位移字段962A和位移定标字段962B是不存在的。

[0145] 无存储器访问指令模板——写掩码控制，部分舍入控制类型操作

[0146] 在无存储器访问写掩码控制部分舍入控制类型操作910指令模板中，β字段954的其余部分被解释成舍入操作字段959A而异常事件报告被禁用(给定的指令不报告任何类型的浮点异常标志并且不唤起任何浮点异常处理程序)。

[0147] 舍入操作控制字段959A——就像舍入操作控制字段958——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、向零舍入和向最近舍入)。因此，舍入操作控制字段959A允许在每个指令的基础上改变舍入模式，并因此当需要时尤为有用。在处理器包括用于指定舍入模式的控制寄存器的本发明的一个实施例中，舍入运算控制字段950内容超越寄存器值(能够选择舍入模式而不必在该控制寄存器上执行保存-修改-恢复是有利的)。

[0148] 无存储器访问指令模板——写掩码控制VSIZE类型操作

[0149] 在无存储器访问写掩码控制VSIZE类型操作917指令模板中，β字段954的其余部分被解释成向量长度字段959B，其内容区分众多数据向量长度中的哪一个要被执行(例如在128、956或1112字节的基础上)。

[0150] 类B的存储器访问指令模板

[0151] 在类A的存储器访问920指令模板中，β字段954的一部分被解释成广播字段957B，其内容区分是否要执行广播类型数据操纵操作，而β字段954的其余部分解释向量长度字段959B。存储器访问920指令模板包括定标字段960、可选用的位移字段962A或位移定标字段962B。

[0152] 关于字段的附加注释

[0153] 关于一般向量友好指令格式900，全操作码字段974图示为包括格式字段940、基础操作字段942以及数据元宽度字段964。尽管示出了全操作码字段974包括所有这些字段的一个实施例，然而全操作码字段974在不支持所有这些字段的实施例中包括比所有这些字段更少的字段。全操作码字段974提供操作码。

[0154] 增量操作字段950、数据元宽度字段964以及写掩码字段970允许这些特征以一般向量友好指令格式在每个指令的基础上被规定。

[0155] 写掩码字段和数据元宽度字段的组合形成带类型的指令，这些指令允许基于不同的数据元宽度应用掩码。

[0156] 该指令格式需要相对少量的位，因为它基于其它字段的内容再使用不同的字段以实现不同的目的。例如，一个前景是修饰符字段的内容在图9A-B上的无存储器访问905指令模板和图9A-B上的存储器访问9250指令模板之间作出选择；而类字段968的内容在图9A的

指令模板910/915和图9B的912/917之间的那些无存储器访问905指令模板内作出选择；而类字段968的内容在图9A的指令模板925/930以及图9B的927之间的那些存储器访问920指令模板中作出选择。从另一前景看来，类字段968的内容在图9A和图9B的类A和类B指令模板之间作出选择；而修饰符字段的内容在图9A的指令模板905、920之间的那些类A指令模板中作出选择；同时修饰符字段的内容在图9B的指令模板905、920之间的那些类B指令模板中作出选择。在类字段的内容指示类A指令模板的情形下，修饰符字段946的内容选择α字段952的解释(在rs字段952A和EH字段952B之间)。在关联的方式中，修饰符字段946和类字段968的内容选择是将α解释为rs字段952A、EH字段952B还是写掩码控制(Z)字段952C。在类和修饰符字段指示类A无存储器访问操作的情形下，增量字段的β字段的解释基于rs字段内容而改变；而在类和修饰符字段指示类B无存储器访问操作的情形下，β字段的解释依赖于RL字段的内容。在类和修饰符字段指示类A存储器访问操作的情形下，增量字段的β字段的解释基于基础操作字段的内容而改变；而在类和修饰符字段指示类B存储器访问操作的情形下，增量字段的β字段的广播字段957B的解释基于基础操作字段的内容而改变。由此，基础操作字段、修饰符字段和增量操作字段的组合允许规定宽得多的许多种增量操作。

[0157] 在类A和类B中发现的各种指令模板在不同情况下是有利的。当为了性能原因需要归零写掩码操作或较小的向量长度时，类A是有用的。例如，归零当使用重命名时允许避免假依赖性，因为我们不再需要人工地与目的地融合；又如，当模拟具有向量掩码的较短向量大小时，向量长度控制缓解了存储-加载转发问题。当需要下列选项时，类B是有用的：1)在同时使用舍入模式控制时，允许浮点异常(例如当SAE字段的内容指示否)；2)能使用向上转换、混合、交换和/或向下转换；3)对图形数据类型的操作。例如，当处理不同格式的源时，向上转换、混合、交换、向下转换以及图形数据类型减少了所需的指令数；又如，允许异常的能力提供与定向舍入模式的完全IEEE相容性。

[0158] 示例性特定向量友好指令格式

[0159] 图10A-C示出根据本发明实施例的示例性特定向量友好指令格式。图10A-C示出特定向量友好指令格式1000，其特定的意义在于它规定字段的位置、大小、解释和顺序以及那些字段中的一些的值。可使用特定向量友好指令格式1000来扩展x86指令集，并由此这些字段中的一些与已有的x86指令集及其扩展(例如AVX)中使用的那些字段相似或相同。这种格式与带扩展的已有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、位移字段以及立即数字段保持一致。示出了来自图9的字段，来自图10A-C的字段映射入图9的字段。

[0160] 应当理解，尽管的本发明的实施例为了说明目是在一般向量友好指令格式900的背景下参照特定向量友好指令格式1000描述的，然而本发明不仅限于该特定向量友好指令格式1000，除非另有声明。例如，一般向量友好指令格式900考虑各字段的多种可能大小，而特定向量友好指令格式1000被图示为具有特定大小的字段。作为特定示例，尽管数据元宽度字段964被图示为特定向量友好指令格式1000中的一个位字段，但本发明不限于此(也就是说，一般向量友好指令格式900考虑数据元宽度字段964的其它大小)。

[0161] 格式-图10A-C

[0162] 一般向量友好指令格式900包括下面以图10A-C所示的顺序列出的字段。

[0163] EVEX前缀(字节0-3)

[0164] EVEX前缀1002——以四字节形式被编码。

[0165] 格式字段940(EVEX字节0,位[7:0])——第一字节(EVEX字节0)是格式字段940,并且其包含0x62(在本发明的一个实施例中用于区分向量友好指令格式的唯一值)。

[0166] 第二至第四字节(EVEX字节1-3)包括提供特定能力的数个位字段。

[0167] REX字段1005(EVEX字节1,位[7-5])——包括EVEX.R位字段(EVEX字节1,位[7]-R)、EVEX.X位字段(EVEX字节1、位[6]-X)以及957BEX字节1,位[5]-B)。EVEX.R、EVEX.X和EVEX.B位字段提供与相应VEX位字段相同的功能,并使用1的补码形式编码,即ZMM0被编码为1111B,ZMM15被编码为0000B。指令的其它字段如本领域内技术人员已知的那样对寄存器索引的较低三个位(rrr,xxx和bbb)进行编码,由此可通过加上EVEX.R、EVEX.X和EVEX.B来形成Rrrr、Xxxx和Bbbb。

[0168] REX'字段1010——这是REX'字段1010的第一部分并且是EVEX.R'位字段(EVEX字节1,位[4]-R'),它被用来对扩展的32寄存器组中的高16位或低16位进行编码。在本发明的一个实施例中,该位连同下面指出的其它位以位反转格式被存储从而(在公知的x8632位模式下)区别于BOUND指令,它们的真实操作码字节为62,但在MOD R/M字段(下面描述)中不接受MOD字段中的值11;本发明的替代实施例不以反转格式存储这个和下面的其它指示位。使用值1对较低的16个寄存器进行编码。换句话说,R'Rrrr是通过组合来自其它字段的EVEX.R'、EVEX.R和其它RRR来形成的。

[0169] 操作码映射字段1015(EVEX字节1,位[3:0]-mmmm)——其内容对隐含的前导操作码字节(0F、0F38或0F3)进行编码。

[0170] 数据元宽度字段964(EVEX字节2,位[7]-W)是通过符号EVEX.W表示的。EVEX.W被用来定义数据类型的粒度(大小)(32位数据元或者是64位数据元)。

[0171] EVEX.vvvv1020(EVEX字节2,位[6:3]-vvvv)——EVEX.vvvv的角色可包括下列内容:1)EVEX.vvvv对第一源寄存器操作数编码,其规定为反转(1的补码)形式并对于具有两个或更多个源操作数的指令来说是有效的;2)EVEX.vvvv对目的地寄存器操作数编码,其对于某些向量偏移规定为1的补码形式;或者3)EVEX.vvvv不对任何操作数编码,该字段被预留并应当包含1111b。因此,EVEX.vvvv字段1020对以反转(1的补码)形式存储的第一源寄存器说明符的4个低阶位进行编码。根据该指令,使用额外的不同EVEX位字段来将说明符的大小扩展至32个寄存器。

[0172] EVEX.U968类字段(EVEX字节2,位[2]-U)——如果EVEX.U=0,它表示类A或EXEX.U0;如果EVEX.U=1,它表示类B或EVEX.U1。

[0173] 前缀编码字段1025(EVEX字节2,位[1:0]-pp)提供基础操作字段的附加位。除了提供对EVEX前缀格式下的旧有SSE指令的支持外,这也有利于压缩SIMD前缀(而不是需要表达SIMD前缀的字节,EVEX前缀仅需要2个位)。在一个实施例中,为了支持在旧有格式和EVEX前缀格式两者下使用SIMD前缀(66H、F2H、F3H)的旧有SSE指令,这些旧有SIMD前缀被编码为SIMD前缀编码字段;并在被提供给解码器的PLA(由此PLA可无修饰地执行这些旧有指令的旧有格式和EVEX格式两者)之前在运行时间被扩展到旧有SIMD前缀中。尽管较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,然而某些实施例为了一致性以相似的方式扩展但允许通过这些旧有SIMD前缀规定不同的意义。替代实施例可重新设计PLA以支持2位SIMD前缀编码,并因此不需要扩展。

[0174] α字段952(EVEX字节3,位[7]-EH,也称EVEX.EH、EVEX.rs、EVEX.RL、EVEX.写掩码控制以及EVEX.N;也用α表述)——如之前描述的,该字段是文本特定的。下面给出附加的描述。

[0175] β字段954(EVEX字节3,位[6:4]-SSS,也称EVEX.s_{2-0}、EVEX.r_{2-0}、EVEX.rr1、EVEX.LL0、EVEX.LLB,也通过βββ表示)——如前所述,该字段是文本特定的。下面给出附加的描述。

[0176] REX'字段1010——这是REX'字段的剩余部分并且是EVEX.V'位字段(EVEX字节3,位[3]-V'),它被用来对扩展的32寄存器组中的高16位或低16位进行编码。该位以位反转的格式被存储。使用值1对较低的16个寄存器进行编码。换句话说,通过将EVEX.V'和EVEX.vvvv组合来形成V'VVVV。

[0177] 写掩码字段970(EVEX字节3,位[2:0]-kkk)——其内容如前所述地规定写掩码寄存器中的寄存器的索引。在本发明的一个实施例中,特定值EVEX.kkk=000具有特殊行为,其提示没有写掩码被用于该特定指令(这可通过多种方式实现,包括使用硬线连接至所有硬件或旁路掩码操作硬件的硬件的写掩码)。

[0178] 真实操作码字段1030(字节4)

[0179] 这也被称为操作码字节。在该字段中规定操作码的一部分。

[0180] MOD R/M字段1040(字节5)。

[0181] 修饰符字段946(MODR/M.MOD,位[7-6]-MOD字段1042)——如之前描述的,MOD字段1042的内容在存储器访问操作和无存储器访问操作之间作出区分。该字段将在下文中更详细地予以描述。

[0182] MODR/M.reg字段1044,位[5-3]——ModR/M.reg字段的角色可被概括成两种情况:ModR/M.reg或者对目的地寄存器操作数或者对源寄存器操作数进行编码,或者ModR/M.reg被视为操作码扩展并且不用来对任何指令操作数编码。

[0183] MODR/M.r/m字段1046,位[2-0]——ModR/M.r/m字段的角色可包括下列内容:ModR/M.r/m对引用存储器地址的指令操作数进行编码,或者ModR/M.r/m对目的地寄存器操作数或源寄存器操作数编码。

[0184] 定标、索引、基础(SIB)字节(字节6)

[0185] 定标字段960(SIB.SS,位[7-6])——如之前描述地,定标字段960的内容被用于存储器地址产生。该字段将在下文中更详细地予以描述。

[0186] SIB.xxx1054(位[5-3]和SIB.bbb1056(位[2-0]))——这些字段的内容之前已相关于寄存器索引Xxxx和Bbbb而提及。

[0187] 位移字节(字节7或字节7－10)

[0188] 位移字段962A(字节7-10)——当MOD字段1042包含10时,字节7－10是位移字段962A,并且其工作方式与旧有32位位移(disp32)相同并工作在字节粒度下。

[0189] 位移因子字段962B(字节7)——当MOD字段1042包含01时,字节7是位移因子字段962B。该字段的位置与旧有x86指令集8位位移(disp8)的位置相同,它工作在字节粒度下。由于disp8是扩展的符号,它只能在-128和128字节偏移之间寻址;就64字节高速缓冲存储器行而言,disp8使用8个位,这些位可被置为仅四个真实有用的值-128、-64、0和64;由于经常需要较大的范围,因此使用disp32;然而disp32需要四个字节。相比于disp8和disp32,位

移因子字段962B是disp8的重新解释；当使用位移因子字段962B时，通过位移因数字段的内容乘以存储器操作数访问的大小(N)来确定实际位移。这种类型的位移被称为disp8*N。这减小了平均指令长度(用于位移的单个字节但具有大得多的范围)。这种经压缩的位移基于这样一种假设，即有效位移是存储器地址的粒度的倍数并因此地址偏移的冗余低阶位不需要被编码。换句话说，位移因子字段962B代替旧有的x86指令集8位位移。因此，位移因数字段962B以与x86指令集8位位移相同的方式编码(因此ModRM/SIB编码规则没有什么变化)，唯一的例外是disp8被过载至disp8*N。换句话说，编码规则或编码长度没有任何变化，仅仅在通过硬件解释位移值时有变化(这需要通过存储器操作数的大小来对位移定标以获得逐字节的地址偏移)。

[0190] 立即数

[0191] 立即数字段972如前所述地操作。

[0192] 示例性寄存器架构——图11

[0193] 图11是根据本发明一个实施例的寄存器架构1100的框图。该寄存器架构的寄存器文件和寄存器被列出如下：

[0194] 向量寄存器文件1110——在所示实施例中，存在1112位宽的32个向量寄存器；这些寄存器被引用为zmm0-zmm31。低位16zmm寄存器的较低阶956个位覆盖在寄存器ymm0-16上。较低16zmm寄存器的较低阶128个位(ymm寄存器的较低阶128个位)覆盖在寄存器xmm0-15上。特定向量友好指令格式1000工作在这些覆盖的寄存器文件上，如下表所示。

[0195]

可调向量长度	类	操作	寄存器
不包括向量长度字段 959B 的指令模板	A (图 9A; U=0)	910, 915, 925, 930	Zmm 寄存器(向量长度为 64 字节)
	B (图 9B; U=1)	912	Zmm 寄存器(向量长度为 64 字节)
包括向量长度字段 959B 的指令模板	B (图 9B; U=1)	917, 927	zmm、ymm 或 xmm 寄存器(向量长度为 64 字节、32 字节或 16 字节)，这依赖于向量长度字段 959B

[0196] 换句话说，向量长度字段959B在最大长度和一个或多个其它较短长度之间作出选择，其中每个这样的较短长度是前一长度的一半；而不具有向量长度字段959B的指令模板

工作在最大向量长度上。此外，在一个实施例中，特定向量友好指令格式1000的类B指令模板工作在压缩的或标量的单/双精度浮点数据上以及压缩的或标量的整数型数据下。标量操作是对zmm/ymm/xmm寄存器中的最低阶数据元位置执行的操作；较高阶的数据元位置根据实施例保持与它们在指令之前的位置相同或者被归零。

[0197] 写掩码寄存器1115——在所示实施例中，存在8个写掩码寄存器(k0－k7)，每个的大小为64位。如之前描述的，在本发明的一个实施例中，向量掩码寄存器K0无法被用作写掩码；当通常指示k0的编码被用于写掩码时，它选择0xFFFF的硬接线写掩码，这有效地禁用对该指令的写掩码操作。

[0198] 多媒体扩展控制状态寄存器(MXCSR)1120——在所示实施例中，这个32位寄存器提供用于浮点运算的状态和控制位。

[0199] 通用寄存器1125——在所示实施例中，存在16个64位通用寄存器，这些通用寄存器连同已有的x86寻址模式一起使用以对存储器操作数进行寻址。这些寄存器是通过名称RAX、RBX、RCX、RDX、RBP、RSI、RDI、RSP以及R8-R15引用的。

[0200] 扩展的标志(EFLAGS)寄存器1130——在所示实施例中，这个32位寄存器被使用以记录许多指令的结果。

[0201] 浮点控制字(FCW)寄存器1135和浮点状态字(FSW)寄存器1140——在所示实施例中，这些寄存器由x87指令集扩展使用以在FCW情形下设置舍入模式、异常掩码和标志，并在FSW的情形下保持对异常的跟踪。

[0202] 标量浮点堆栈寄存器文件(x87堆栈)1145(在其上别名有MMX压缩的整型平寄存器文件1150)——在所示实施例中，x87堆栈是使用x87指令集扩展对32/64/80位浮点数据执行标量浮点运算的八元素堆栈；同时MMX寄存器用来对64位压缩的整数型数据执行操作以及对在MMX和XMM寄存器之间执行的某些操作保持操作数。

[0203] 段寄存器1155——在所示实施例中，存在六个16位寄存器，用来存储数据以供分段的地址产生使用。

[0204] RIP寄存器1165——在所示实施例中，这64位寄存器存储指令指针。

[0205] 本发明的替代实施例可使用更宽或更窄的寄存器。附加地，本发明的替代实施例可使用更多、更少或不同的寄存器文件和寄存器。

[0206] 示例性按顺序的处理器架构——图12A-12B

[0207] 图12A-B示出一示例性按顺序的处理器架构的框图。这些示例性实施例是围绕增加了宽向量处理器(VPU)的按顺序CPU核的多个实例设计的。核通过高宽带互连网络与某些固定的功能逻辑、存储器I/O接口和其它必要的I/O逻辑通信，这依赖于e14t应用。例如，该实施例作为自立CPU的实现典型地包括PCIe总线。

[0208] 图12A是根据本发明实施例的单CPU核连同它与芯片上互连网络1202以及其二级(L2)高速缓冲存储器1204的本地子集的连接的框图。指令解码器1200支持x86指令集，该指令集具有包括特定向量指令格式1000的扩展。尽管在本发明的一个实施例中(为了简化设计)，标量单元1208和向量单元1210使用不同的寄存器组(分别为标量寄存器1212和向量寄存器1214)并且在这两个寄存器之间传递的数据被写至存储器并随后从级1(L1)高速缓冲存储器1206写回，然而本发明的替代实施例可使用不同的方法(例如使用单个寄存器集或包括通信路径，该通信路径允许数据在两个寄存器文件之间转移而不需要被写入和读回)。

[0209] L1高速缓冲存储器1206允许对标量和向量单元中的高速缓冲存储器的低等待时间访问。连同向量友好指令格式中的加载操作码指令，这意味着L1高速缓冲存储器1206某些程度可被视为类似扩展的寄存器文件。这显著地改善了许多算法的性能，尤其是通过逐出提示字段952B。

[0210] L2高速缓冲存储器1204的本地子集是全局L2高速缓冲存储器的一部分，该全局L2高速缓冲存储器被分割成多个独立的本地子集，对每个CPU核有一个本地子集。每个CPU具有至L2高速缓冲存储器1204其本身的本地子集的直接访问路径。由CPU核读取的数据被存储在其L2高速缓冲存储器子集1204中并可被快速访问，与访问其自身的本地L2高速缓冲存储器子集的其它CPU并行。由CPU核写入的数据被存储在其本身的L2高速缓冲存储器子集1204中，并从其它子集转储清除，如果需要的话。环形网络确保对共享数据的一致性。

[0211] 图12B是根据本发明实施例的图12A中的CPU核的一部分的分解图。图12B包括L1高速缓冲存储器1204的L1数据高速缓冲器1206A部分，更详细地涉及向量单元1210和向量寄存器1214。具体地说，向量单元1210是16宽向量处理单元(VPU)(见16宽ALU1228)，它执行整型、单精度浮点以及双精度浮点指令。VPU通过混合单元1220支持对寄存器输入的混合、通过数值转换单元1222A-B支持数值转换，并通过复制单元1224支持对存储器输入的复制。写掩码寄存器1226允许预测作为结果的向量写。

[0212] 寄存器数据可以多种方式混合，例如以支持矩阵乘。来自存储器的数据可横跨VPU通道地复制。这在图形和非图形并行数据处理都是常见操作，其显著地提高了高速缓冲存储器效率。

[0213] 环形网络是双向的，以允许诸如CPU核的代理、L2高速缓冲存储器以及其它逻辑块在芯片内彼此通信。每个环形数据路径在每个方向上为1112位宽。

[0214] 示例性无序架构——图13

[0215] 图13是示出根据本发明实施例的示例性无序架构的框图。具体地说，图13示出一种公知的示例性无序架构，该架构已被修正以包含向量友好指令格式及其执行。在图13中，箭头指示两个或更多个单元之间的耦合，且箭头的方向指示那些单元之间的数据流的方向。图13包括耦合至执行引擎单元1310和存储器单元1315的前端单元1305；该执行引擎单元1310进一步耦合至存储器单元1315。

[0216] 前端单元1305包括级1(L1)分支预测单元1320，该L1分支预测单元1320耦合至级2(L2)分支预测单元1322。L1和L2分支预测单元1320、1322耦合至L1指令高速缓冲存储器单元1324。L1指令高速缓冲存储器单元1324被耦合至指令翻译后备缓存(TLB)1326，该TLB1326进一步耦合至指令取出和预解码单元1328。指令取出和预编码单元1328耦合至指令队列单元1330，该指令队列单元1330被进一步耦合至解码单元1332。解码单元1332包括复杂解码器单元1334以及三个简单解码器单元1336、1338和1340。解码单元1332包括微代码ROM单元1342。解码单元1332可如前所述地工作在解码级区段。L1指令高速缓冲存储器单元1324进一步耦合至存储器单元1315中的L2高速缓冲存储器单元1348。指令TLB单元1326进一步耦合至存储器单元1315中的第二级TLB单元1346。解码单元1332、微代码ROM单元1342以及环流检测器单元1344各自耦合至执行引擎单元1310中的重命名/分配器单元1356。

[0217] 执行引擎单元1310包括重命名/分配器单元1356，该重命名/分配器单元1356耦合

至隐退单元1374和统一调度单元1358。隐退单元1374进一步耦合至执行单元1360并包括重定序缓存单元1378。统一调度单元1358进一步耦合至物理寄存器文件单元1376，该物理寄存器文件单元1376耦合至执行单元1360。物理寄存器文件单元1376包括向量寄存器单元1377A、写掩码寄存器单元1377B以及标量寄存器单元1377C；这些寄存器单元可提供向量寄存器1110、向量掩码寄存器1115以及通用寄存器1125，并且物理寄存器文件单元1376可包括未示出的附加寄存器文件(例如在MMX压缩的整数型平寄存器文件1150上别名的标量浮点堆栈寄存器文件1145)。执行单元1360包括：三个混合的标量和向量单元1362、1364和1372；加载单元1366；存储地址单元1368；存储数据单元1370。加载单元1366、存储地址单元1368以及存储数据单元1370各自进一步耦合至该存储器单元1315中的数据TLB单元1352。

[0218] 存储器单元1315包括第二级TLB单元1346，该第二级TLB单元1346耦合至数据TLB单元1352。数据TLB单元1352耦合至L1数据高速缓冲存储器单元1354。L1数据高速缓冲存储器单元1354进一步耦合至L2高速缓冲存储器单元1348。在一些实施例中，L2高速缓冲存储器单元1348进一步耦合至L3以及存储器单元1315之内和/或之外的较高的高速缓冲存储器单元1350。

[0219] 作为示例，示例性无序架构可实现如下的进程流水线：1)指令取出和预解码单元1328执行取出和长度解码阶段；2)解码单元1332执行解码阶段；3)重命名/分配器单元1356执行分配阶段和重命名阶段；4)统一调度器1358执行调度阶段；5)物理寄存器文件单元1376、重定序缓冲器单元1378以及存储器单元1315执行寄存器读/存储器读阶段；执行单元1360作执行/数据转换阶段；6)存储器单元1315和重定序缓存单元1378执行写回/存储器写阶段；7)隐退单元1374执行ROB读阶段；8)各单元可牵涉到异常应对阶段9164；以及9)隐退单元1374和物理寄存器文件单元1376执行委托阶段。

[0220] 示例性单核和多核处理器——图18

[0221] 图18是根据本发明的实施例具有集成的存储器控制器和图形的单核处理器和多核处理器1800的框图。图18中的实线框示出具有单个核1802A的处理器1800、系统代理1810、一组的一个或多个总线控制器单元1816，而可选增加的虚线框示出具有多个核1802A-N的替代处理器1800、系统代理单元1810中的一组一个或多个集成存储器控制器单元1814以及集成图形逻辑1808。

[0222] 系统阶层包括核内的一个或多个高速缓冲存储器级、一组或一个或多个共享的高速缓冲存储器单元1806、耦合至一组集成存储器控制器单元1814的外部存储器(未示出)。这组共享高速缓冲存储器单元1806可包括一个或多个中级高速缓冲存储器，诸如级2(L2)、级3(L3)、级4(L4)或其他级的高速缓冲存储器、末级高速缓冲存储器(LLC)、和/或其组合。尽管在一个实施例中，基于环的互连单元1812将集成图形逻辑1808、一组共享的高速缓冲存储器单元1806以及系统代理单元1810互连，然而替代实施例可使用任何数量的公知技术来将这些单元互连。

[0223] 在一些实施例中，一个或多个核1802A-N能作多线程处理。系统代理1810包括协调和操作核1802A-N的那些组件。系统代理单元1810可包括例如功率控制单元(PCU)和显示单元。PCU可以是或包括调整核1802A-N和集成图形逻辑1808的功率状态所需的逻辑和组件。显示单元用于驱动一个或多个外部连接的显示器。

[0224] 核1802A-N就架构和/或指令集而言可以是同质的或异质的。例如，核1802A-N中的

一些可以是按顺序的(例如图12A和12B所示的那些),而另一些是无序的(例如图13所示的那些)。作为另一示例,核1802A-N中的两个或更多个能执行相同的指令集,而其它核只能执行该指令集的子集或不同的指令集。诸核中的至少一个能执行本文所述的向量友好指令格式。

[0225] 处理器可以是通用处理器,例如Core™ i3,i5,i7,2Duo和Quad,Xeon ™,或Itanium™处理器,它们可从加利福尼亚州圣克拉拉市的英特尔公司获得。替代地,处理器也可来自另一公司。处理器可以是专用处理器,例如网络或通信处理器、压缩引擎、图形处理器、协处理器、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1800可以是一个或多个衬底的一部分和/或使用诸如BiCMOS、CMOS或NMOS之类的任何数量的加工技术实现在一个或多个衬底上。

[0226] 示例性计算机系统和处理器——图14－17

[0227] 图14－16是适于包括处理器1800的示例性系统,而图17是可包括一个或多个核1802的示例性芯片上系统(SoC)。本领域已知的对于膝上型设备、台式机、手持PC、个人数字助理、工程工作站、服务器、网络设备、网络集线器、交换机、嵌入式处理器、数字信号处理器(DSP)、图形设备、视频游戏设备、机顶盒、微控制器、蜂窝电话、便携式媒体播放器、手持设备以及各种其他电子设备的其他系统设计和配置也是合适的。一般来说,能够纳入本文中所公开的处理器和/或其它执行逻辑的大量系统和电子设备一般都是合适的。

[0228] 现在参见图14,所示为根据本发明的一个实施例的系统1400的框图。系统1400可包括一个或多个处理器1410、1415,这些处理器1410、1415耦合至图形存储器控制器中枢(GMCH)1420。附加处理器1415的可选用性质用虚线表示在图14中。

[0229] 每个处理器1410、1415可以是处理器1800的某些版本。然而应当注意,集成的图形逻辑和集成的存储器控制单元存在于处理器1410、1415内是不大可能的。.

[0230] 图14示出GMCH1420可耦合至存储器1440,该存储器1440例如可以是动态随机存取存储器(DRAM)。对于至少一个实施例,DRAM可关联于非易失性高速缓冲存储器。

[0231] GMCH1420可以是芯片集或芯片集的一部分。GMCH1420可与处理器1410、1415通信并控制处理器1410、1415与存储器1440之间的相互作用。GMCH1420也可充当处理器1410、1415与系统1400的其它部件之间的加速总线接口。对于至少一个实施例,GMCH1420经由例如前侧总线(FSB)1495的多站总线与处理器1410、1415通信。

[0232] 此外,GMCH1420耦合至显示器1445(例如平板显示器)。GMCH1420可包括集成图形加速器。GMCH1420进一步耦合至输入/输出(I/O)控制器中枢(ICH)1450,ICH1450可用于将多种外设设备耦合至系统1400。例如在图14的实施例中示出外部图形设备1460,它可以是与另一外设设备1470一起耦合至ICH1450的分立图形设备。

[0233] 替代地,附加的或不同的处理器也可出现在系统1400中。例如,附加的处理器1415可包括与处理器1410相同的附加处理器、与处理器1410异质或不对称的附加处理器、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任何其它处理器。就品质度量谱而言,物理资源1410、1415之间可以有许多差别,包括架构、微架构、热、功耗特性等等。这些差别可有效地在处理元件1410、1415之间自我展示为非对称的和异质的。对于至少一个实施例,各处理元件1410、1415可留驻在同一管芯封装件内。

[0234] 现在参见图15,所示为根据本发明实施例的第二系统1500的框图。如图15所示,多

处理器系统1500是点对点互连系统，并包括经由点对点互连1550耦合的第一处理器1570和第二处理器1580。如图15所示，处理器1570、1580中的每一个可以是处理器1800的一些版本。

[0235] 替代地，处理器1570、1580中的一个或多个可以是处理器以外的元件，例如加速器或现场可编程门阵列。

[0236] 尽管仅示出两个处理器1570、1580，然而要理解本发明的范围不限于此。在其它实施例中，一个或多个附加处理元件可存在于给定处理器中。

[0237] 处理器1570可进一步包括集成存储器控制器中枢(IMC)1572和点对点(P-P)接口1576、1578。类似地，第二处理器1580可包括IMC1582和P-P接口1576、1588。处理器1570、1580可以使用点对点(PtP)接口电路1578、1588经由PtP接口1550来交换数据。如图15所示，IMC1572、1582将各处理器耦合至相应的存储器，即存储器1542和存储器1544，这些存储器可以是本地附连至相应处理器的主存储器的多个部分。

[0238] 处理器1570、1580均可使用点对点接口电路1576、1594、1586、1598经由单独的P-P接口1552、1554来与芯片组1590交换数据。芯片集1590还可经由高性能图形接口1539来与高性能图形电路1538交换数据。

[0239] 可在两个处理器外部的任一个处理器中包括共享的高速缓存(未示出)，并经由P-P互连与这些处理器连接，从而如果将处理器置于低功率模式时，可将任一个或两个处理器的本地高速缓存信息存储在共享的高速缓存中。

[0240] 芯片集1590可经由接口1596耦合至第一总线1516。在一个实施例中，第一总线1516可以是外围组件互连(PCI)总线，或诸如PCI Express总线或其它第三代I/O互连总线的总线，尽管本发明的范围不限于此。

[0241] 如图15所示，各种I/O设备1514可连同总线桥1518耦合至第一总线1516，该总线桥1518将第一总线1516耦合至第二总线1520。在一个实施例中，第二总线1520可以是低引脚数量(LPC)总线。多种设备可耦合至第二总线1520，这些设备在一个实施例中包括例如键盘/鼠标1522、通信设备1526以及诸如可包括代码1530的盘驱动或其它大容量存储设备的数据存储单元1528。此外，音频I/O1524可耦合至第二总线1520。注意，其它架构也是可能的。例如，不采用图15的点对点架构，系统可采用多站总线或其它这类架构。

[0242] 现在参见图16，所示为根据本发明实施例的第三系统1600的框图。图15和图16中的相同部件用相同附图标记表示，并从图16中省去图15中的某些方面以避免使图16的其它方面变得模糊。

[0243] 图16示出处理部件1570、1580可分别包括集成存储器和I/O控制逻辑(CL)1572、1582。对于至少一个实施例，CL1572、1582可包括存储器耦合器中枢逻辑(IMC)，例如前面结合图99和图15描述的。另外，CL1572、1582也可包括I/O控制逻辑。图16不仅示出耦合至CL1572、1582的存储器1542、1544，而且示出同样耦合至控制逻辑1572、1582的I/O设备1614。旧有I/O设备1615耦合至芯片集1590。

[0244] 现在参见图17，所示为根据本发明实施例的SoC1700的框图。同样的部件具有同样的附图标记。另外，虚线框是更先进的SoC的可选特征。在图17中，互连单元1702耦合至：应用处理器1710，其包括一组的一个或多个核1802A-N和共享的高速缓冲存储器单元1806；系统代理单元1810；总线控制器单元1816；集成存储器控制器单元1814；可包括集成图形逻辑

1808的一组或一个或多个媒体处理器1720；提供静态和/或视频相机功能的图像处理器1724；提供硬件音频加速的音频处理器1726以及提供视频编码/解码加速的视频处理器1728；静态随机存取存储器(SRAM)单元1730；直接存储器存取(DMA)单元1732；以及用于耦合至一个或多个外部显示器的显示单元1740。

[0245] 本文公开的机制的实施例可实现在硬件、软件、固件或这些实现手法的组合中。本发明的实施例可实现为在可编程系统上执行的计算机程序或程序代码，该可编程系统包括至少一个处理器、存储系统(包括易失性和非易失性存储器和/或存储元件)、至少一个输入设备以及至少一个输出设备。

[0246] 可将程序代码应用至输入数据以执行本文描述的功能并产生输出信息。输出信息可以已知形式被施加至一个或多个输出设备。为本申请的目的，处理系统包括任何具有下列特征的系统：其具有例如数字信号处理器(DSP)的处理器、微控制器、专用集成电路(ASIC)或微处理器。

[0247] 程序代码可实现在高级程序化语言或面向对象的编程语言中以与处理系统通信。程序也可实现为汇编语言或机器语言，如果需要的话。事实上，本文描述的机制不仅限于任何具体的编程语言的范围。在任一情形下，语言可以是编译语言或解释语言。

[0248] 至少一个实施例的一个或多个方面可以由存储在机器可读介质上的代表性指令来实现，该指令表示处理器中的各种逻辑，其在被机器读取时使得该机器生成执行本文描述的技术的逻辑。被称为“IP核”的这些表示可以被存储在有形的机器可读介质上，并被提供给多个客户或生产设施以加载到实际制造该逻辑或处理器的制造机器中。

[0249] 这些机器可读存储介质可包括但不限于由机器或设备制造或形成的物品的非临时的有形配置，其包括存储媒体，例如：硬盘；任何其它类型盘，包括软盘、光盘、压缩盘只读存储器(CD-ROM)、压缩盘可写(CD-RW)以及磁光盘；半导体器件，例如只读存储器(ROM)、诸如动态随机存取存储器(DRAM)和静态随机存取存储器(SRAM)的随机存取存储器(RAM)、可擦除可编程只读存储器(EPROM)、闪存、电可擦除可编程只读存储器(EEPROM)；磁卡或光卡；或适于存储电子指令的任何其它类型介质。

[0250] 因此，本发明的实施例也包括非临时的有形机器可读介质，该介质包含指令向量友好指令格式或包含设计数据，例如硬件描述语言(HDL)，它定义本文描述的结构、电路、装置、处理器和/或系统特征。这些实施例也被称为程序产品。

[0251] 在一些情形下，可使用指令转换器来将指令从源指令集转换至目标指令集。例如，指令转换器可翻译(例如使用静态二进制翻译、包括动态编译的动态二进制翻译)、变形、仿真或以其它方式将指令转换成一个或多个其它指令以供核处理。指令转换器可以在软件、硬件、固件、或其组合中实现。指令转换器可以在处理器上、在处理器之外或一部分在处理器上一部分在处理器外。

[0252] 图19是根据本发明实施例对照软件指令转换器的使用将源指令集中的二进制指令转换成目标指令集中的二进制指令的框图。在所示实施例中，指令转换器是软件指令转换器，尽管替代地该指令转换器可以软件、固件、硬件或其多种组合来实现。图19示出以高级语言1902撰写的程序，该程序可使用x86编译器1904编译以产生x86二进制代码1906，该x86二进制代码1906可由具有至少一个x86指令集核1916的处理器本地执行(假设被编译的一些指令是以向量友好指令格式出现的)。具有至少一个x86指令集核1916的处理器代表能

执行与具有至少一个x86指令集核的英特尔处理器基本相同的功能的任何处理器，这是通过可兼容地执行或处理下列内容来实现的：1)英特尔x86指令集核的指令集的本质部分，或2)面向运行在具有至少一个x86指令集核的英特尔处理器上的应用或其它软件的目标代码版本，以取得与具有至少一个x86指令集核的英特尔处理器基本相同的结果。x86编译器1904代表可工作以产生x86二进制代码1906(例如目标代码)的编译器，该二进制代码1906可通过或不通过附加的关联处理在具有至少一个x86指令集核1916的处理器上执行。类似地，图19示出以高级语言1902撰写的程序，该程序可使用替代的指令集编译器1908编译以产生替代的指令集二进制代码1910，该替代的指令集二进制代码1910可通过或不通过至少一个x86指令集核1914在本地处理(例如具有执行加利福尼亚州的桑尼维尔的MIPS Technologies的MIPS指令集和/或执行加利福尼亚州的桑尼维尔的ARM股份有限公司的ARM指令集的核的处理器)。指令转换器1912被用来将x86二进制代码1906转换成可由不具有x86指令集核1914处理器在本地执行的代码。该转换的代码不大可能与替代的指令集二进制代码1910相同，因为能够这样做的指令转换器难以制造；然而，转换的代码将完成一般操作并由来自替代指令集的指令构成。因此，指令转换器1912代表通过仿真、模拟或任何其它过程使不具有x86指令集处理器或核的处理器或其它电子器件执行x86二进制代码1906的软件、固件、硬件或其组合。

[0253] 本文披露的向量友好指令格式中的指令的某些操作可通过硬件组件执行并可以机器可执行指令体现，该机器可执行指令造成或至少导致以这些指令编程的电路或其它硬件组件执行一些操作。电路可包括通用或专用处理器、或逻辑电路，这里仅给出几个示例。操作也可任选地由硬件和软件的组合来执行。执行逻辑和/或处理器可包括响应于机器指令或从机器指令导出的一个或多个控制信号而存储指令指定的结果操作数的具体或特定电路或其他逻辑。例如，本文描述的指令的实施例可在图14－17的一个或多个系统中执行，并且向量友好指令格式下的指令的实施例可被存储在程序代码中以在系统中被执行。另外，这些附图的处理部件可利用本文详述的流水线和/或架构中的一种(例如按顺序架构和无序架构)。例如，无序架构的解码单元可对指令进行解码，将经解码的指令传至向量或标量单元等。

[0254] 以上描述旨在说明本发明的优选实施例。根据以上讨论还应该清楚，尤其是在快速增长且不容易预见进一步进步的这些技术领域，本领域技术人员可在装置和细节上修改本发明而不脱离落在所附权利要求及其等效技术方案的范围内的本发明原理。例如，方法的一个或多个操作可被合并或进一步拆分。

[0255] 替代实施例

[0256] 尽管已对在本地执行向量友好指令格式的实施例进行了描述，然而本发明的替代实施例可通过运行在执行不同指令集的处理器(例如执行加利福尼亚州的桑尼维尔的MIPS Technologies的MIPS指令集的处理器、执行加利福尼亚州的桑尼维尔的ARM股份有限公司的ARM指令集的处理器)上的仿真层执行向量友好指令格式。另外，尽管图中的流程图示出由本发明某些实施例执行的具体操作顺序，然而应当理解，这些顺序是示例性的(例如替代实施例可以不同顺序执行操作，合并某些操作，重叠某些操作，等等)。

[0257] 在以上描述中，为解释起见，阐明了众多具体细节以提供对本发明的实施例的透彻理解。然而，将对本领域技术人员明显的是，没有这些具体细节中的一些也可实践一个或

多个其他实施例。提供所描述的具体实施例不是为了限制本发明而是为了说明本发明的实施例。本发明的范围不是由前面提供的具体示例来确定的，而是仅由所附权利要求来确定的。

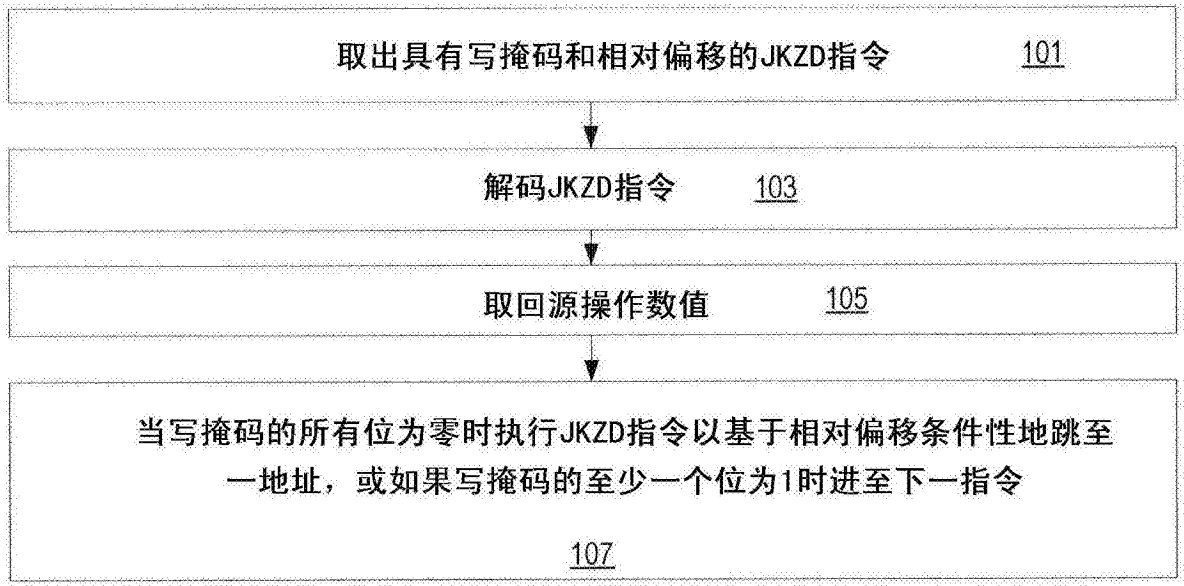

取出具有写掩码和相对偏移的JKZD指令
101
解码JKZD指令
103
取回源操作数值
105
当写掩码的所有位为零时执行JKZD指令以基于相对偏移条件性地跳至一地址，或如果写掩码的至少一个位为1时进至下一指令
107

图1

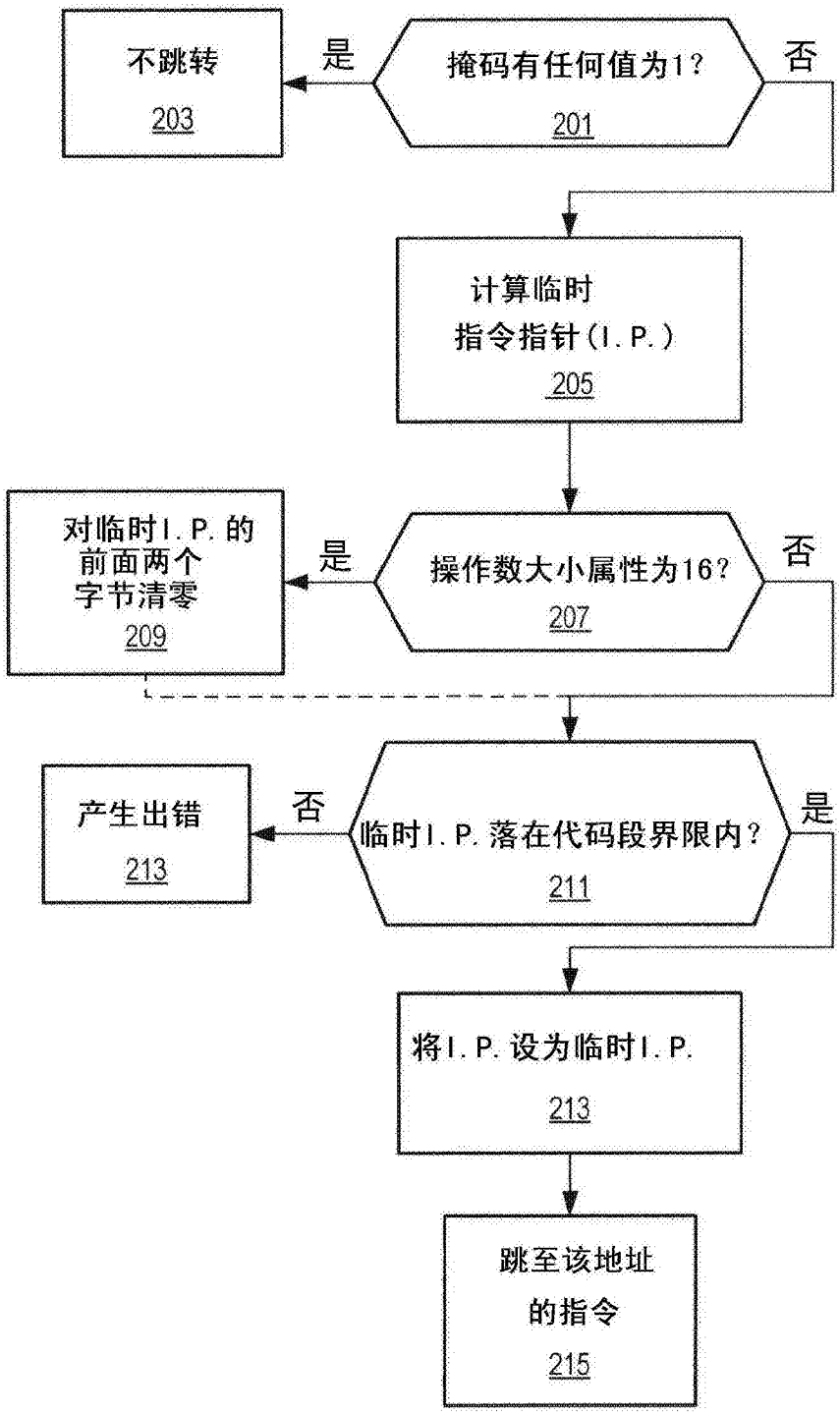

掩码有任何值为1？
201
是
不跳转
203
否
计算临时
指令指针(I.P.)
205
操作数大小属性为16？
207
是
对临时I.P.的
前面两个
字节清零
209
否
临时I.P.落在代码段界限内？
211
否
产生出错
213
是
将I.P.设为临时I.P.
213
跳至该地址
的指令
215

图2

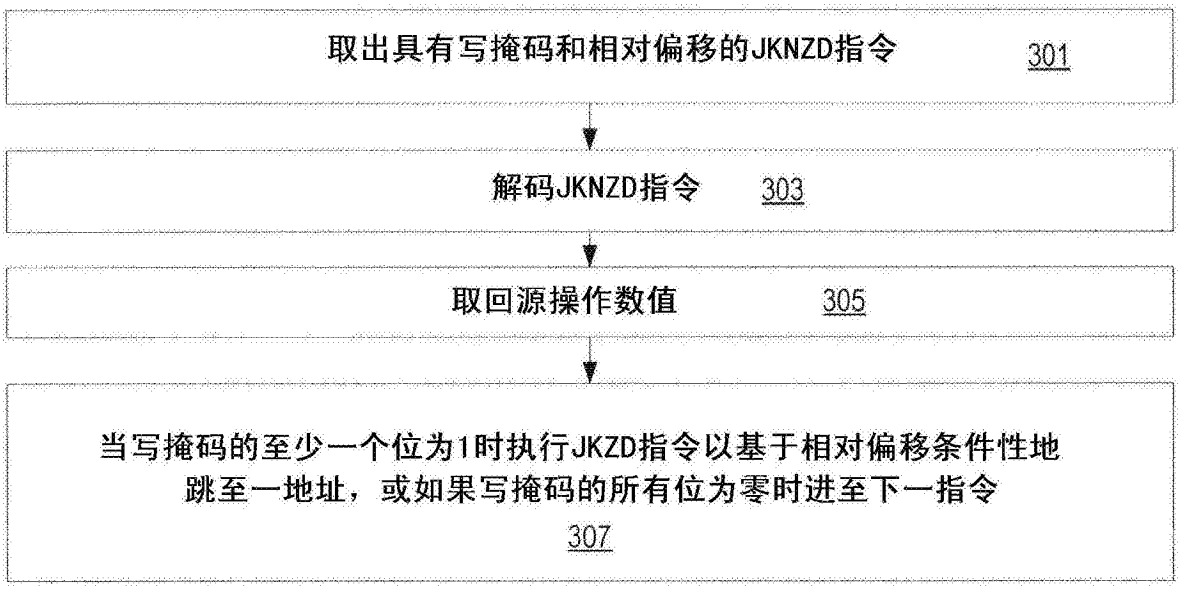

取出具有写掩码和相对偏移的JKNZD指令
301
解码JKNZD指令
303
取回源操作数值
305
当写掩码的至少一个位为1时执行JKZD指令以基于相对偏移条件性地跳至一地址，或如果写掩码的所有位为零时进至下一指令
307

图3

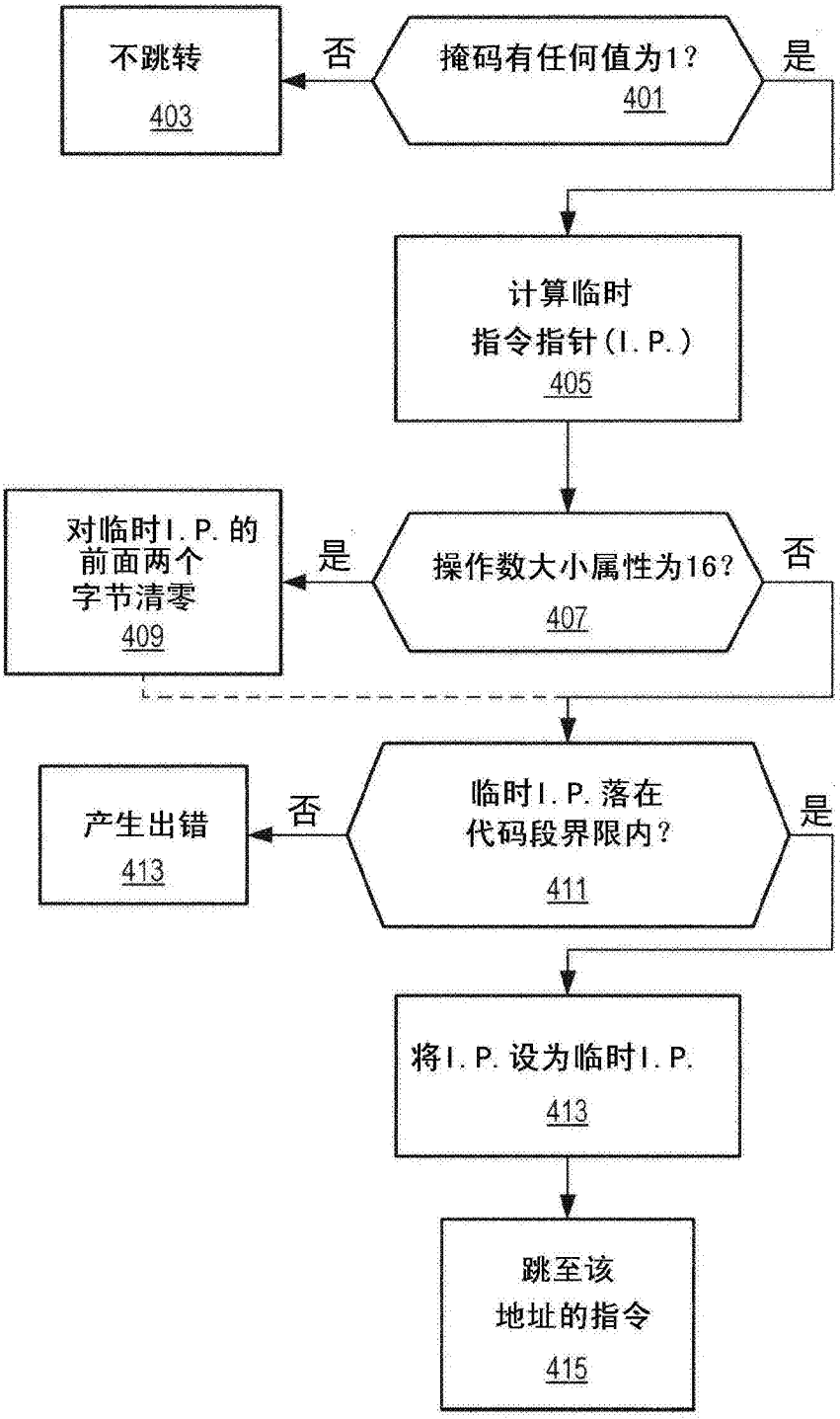

掩码有任何值为1？
401
否
不跳转
403
是
计算临时
指令指针(I.P.)
405
操作数大小属性为16？
407
是
对临时I.P.的
前面两个
字节清零
409
否
临时I.P.落在
代码段界限内？
411
否
产生出错
413
是
将I.P.设为临时I.P.
413
跳至该
地址的指令
415

图4

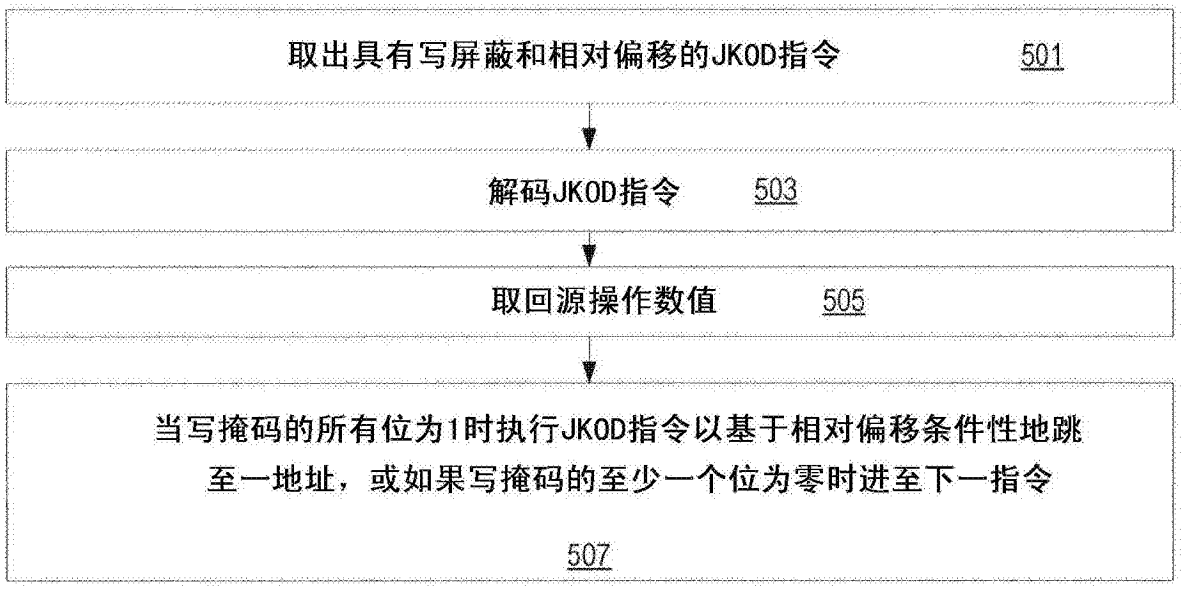

取出具有写屏蔽和相对偏移的JKOD指令
501
解码JKOD指令
503
取回源操作数值
505
当写掩码的所有位为1时执行JKOD指令以基于相对偏移条件性地跳至一地址，或如果写掩码的至少一个位为零时进至下一指令
507

图5

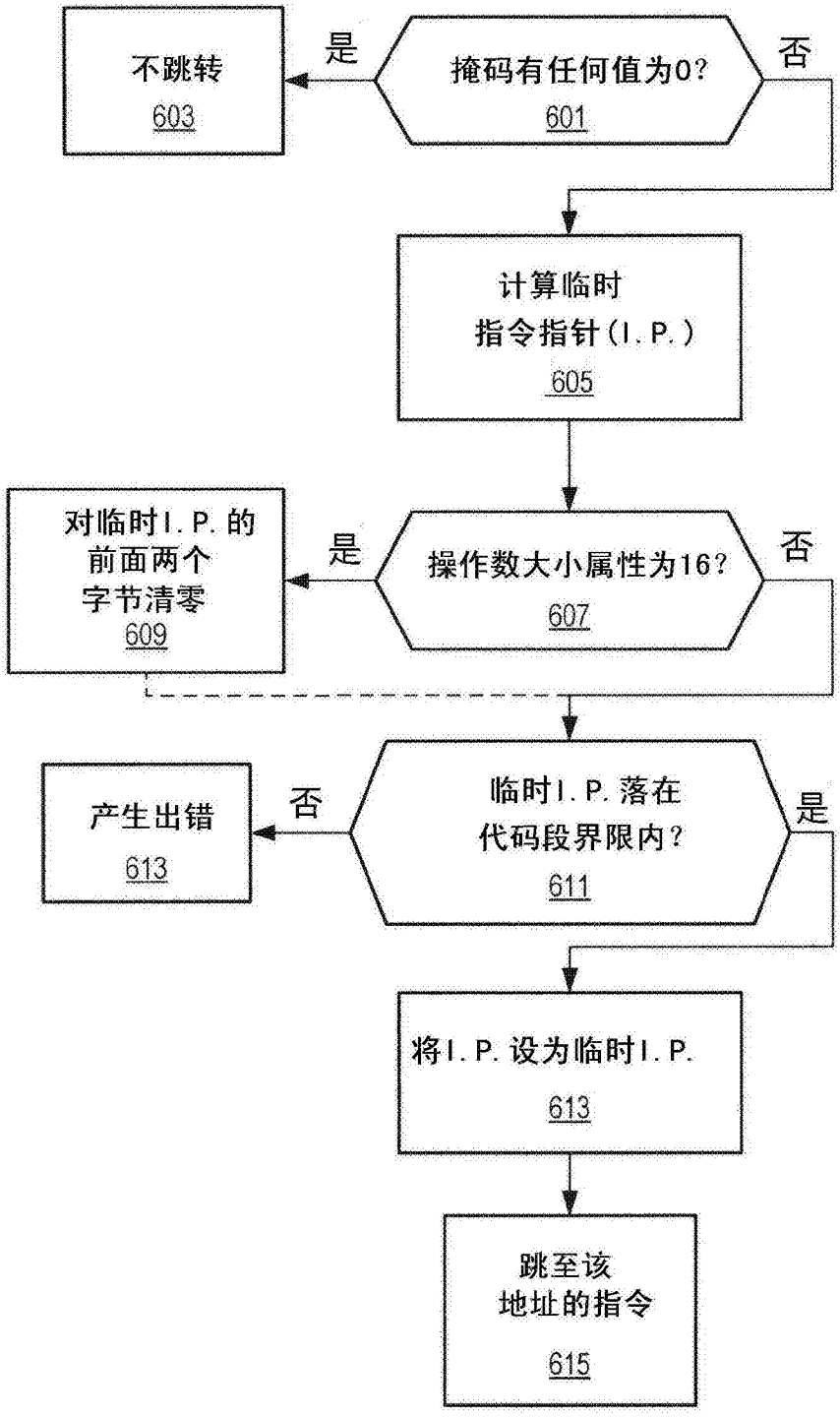

掩码有任何值为0?
601
是
不跳转
603
否
计算临时
指令指针(I.P.)
605
操作数大小属性为16?
607
是
对临时I.P.的
前面两个
字节清零
609
否
临时I.P.落在
代码段界限内?
611
是
否
产生出错
613
将I.P.设为临时I.P.
613
跳至该
地址的指令
615

图6

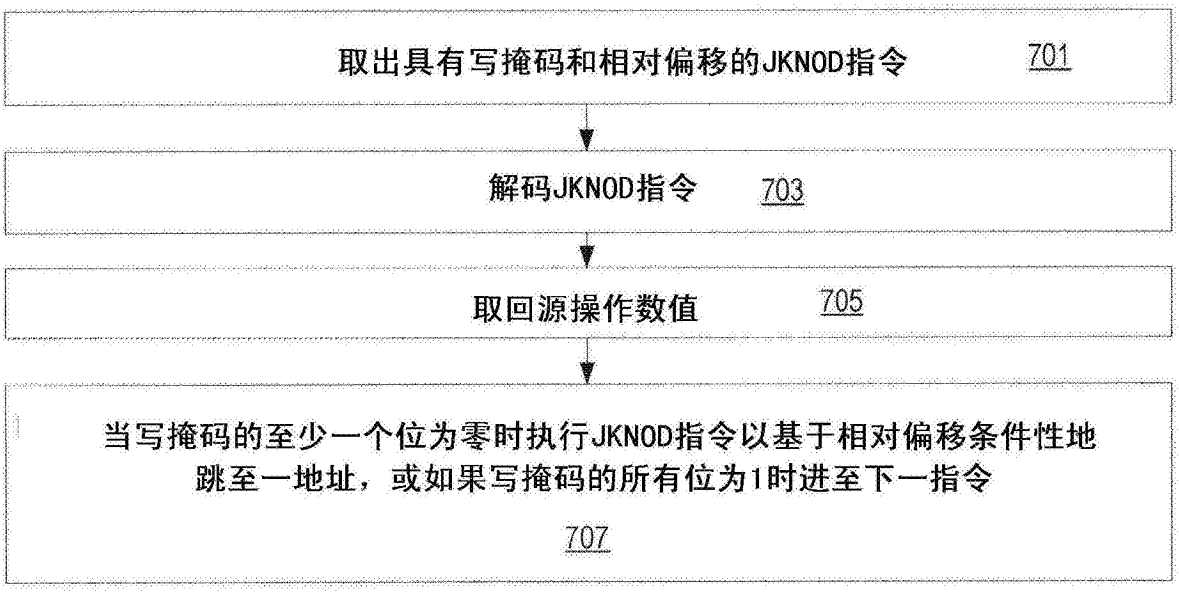

取出具有写掩码和相对偏移的JKNOD指令
701
解码JKNOD指令
703
取回源操作数值
705
当写掩码的至少一个位为零时执行JKNOD指令以基于相对偏移条件性地跳至一地址，或如果写掩码的所有位为1时进至下一指令
707

图7

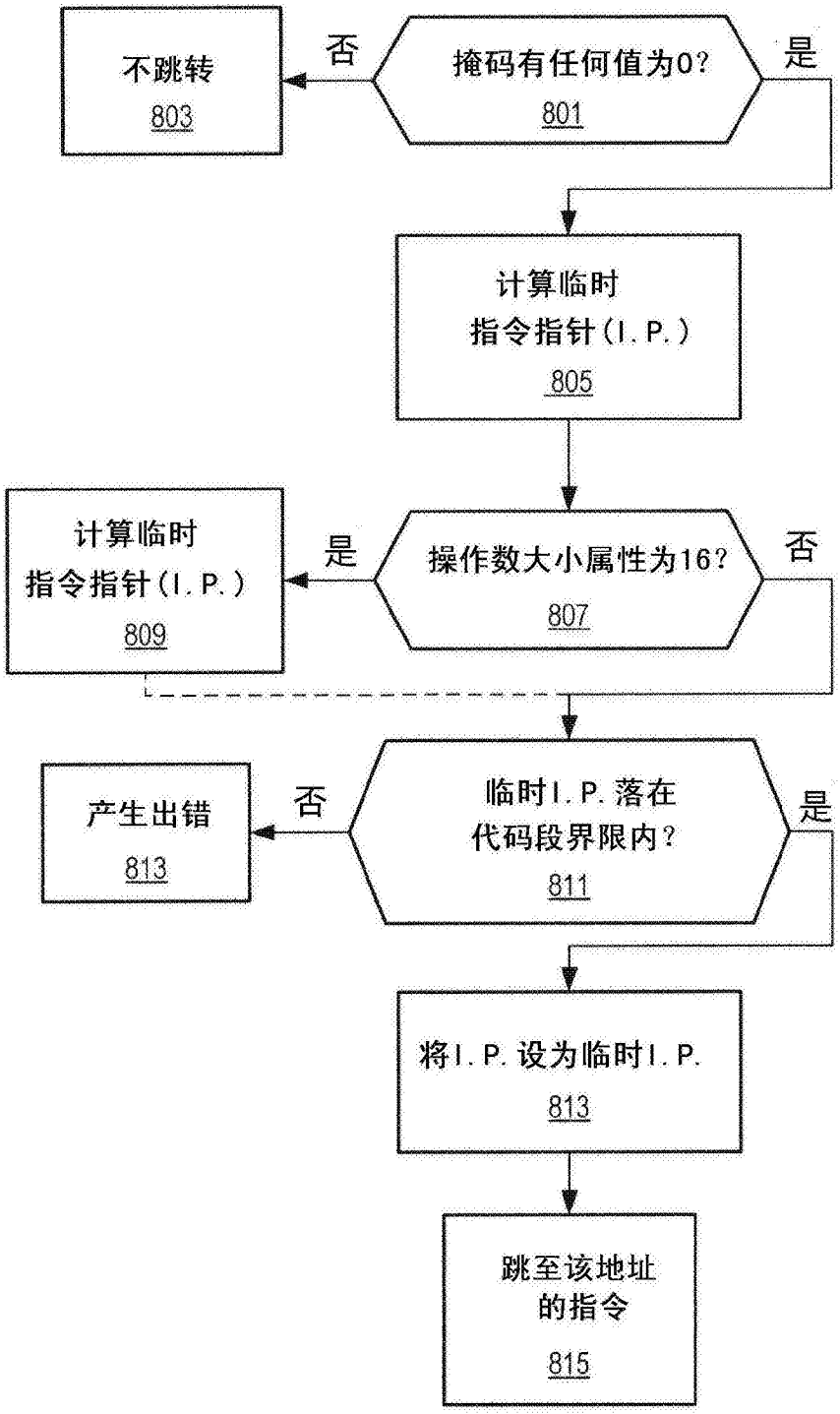

掩码有任何值为0?
801
否
不跳转
803
是
计算临时
指令指针(I.P.)
805
操作数大小属性为16?
807
是
计算临时
指令指针(I.P.)
809
否
临时I.P.落在
代码段界限内?
811
否
产生出错
813
是
将I.P.设为临时I.P.
813
跳至该地址
的指令
815

图8

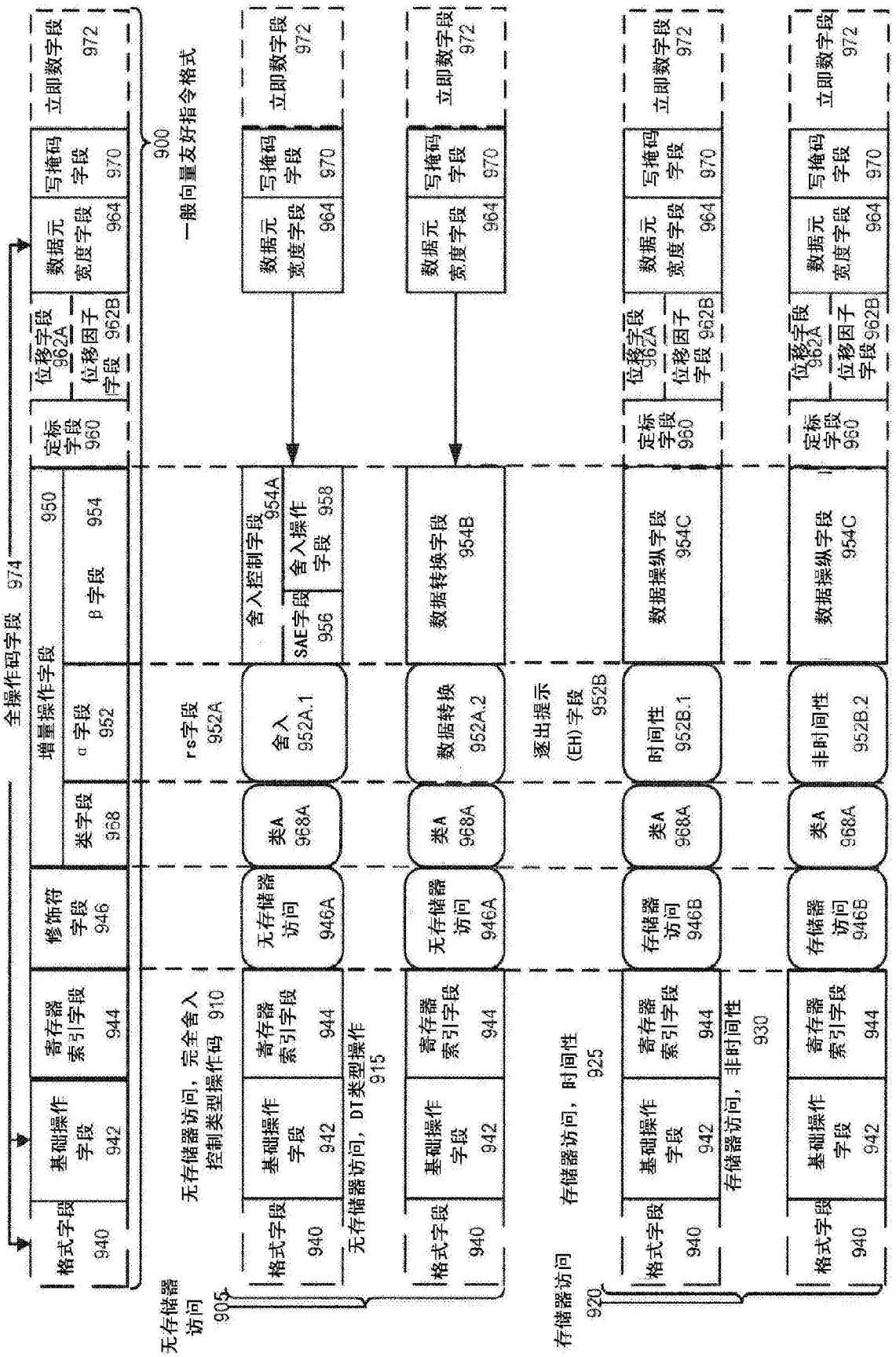

全操作码字段 974
900
一般向量友好指令格式
格式字段 940
基础操作字段 942
寄存器索引字段 944
修饰符字段 946
增量操作字段 950
定标字段 960
位移字段 962A
位移因子字段 962B
数据元宽度字段 964
写掩码字段 970
立即数字段 972
类字段 968
α 字段 952
β 字段 954
无存储器访问 905
无存储器访问，完全舍入控制类型操作码 910
rs字段 952A
舍入控制字段 954A
SAE字段 956
舍入操作字段 958
无存储器访问，DT类型操作 915
舍入 952A.1
数据转换 952A.2
数据转换字段 954B
无存储器访问 946A
类A 968A
存储器访问，时间性 925
逐出提示(EH)字段 952B
数据操纵字段 954C
存储器访问 920
存储器访问 946B
时间性 952B.1
存储器访问，非时间性 930
非时间性 952B.2

图9A

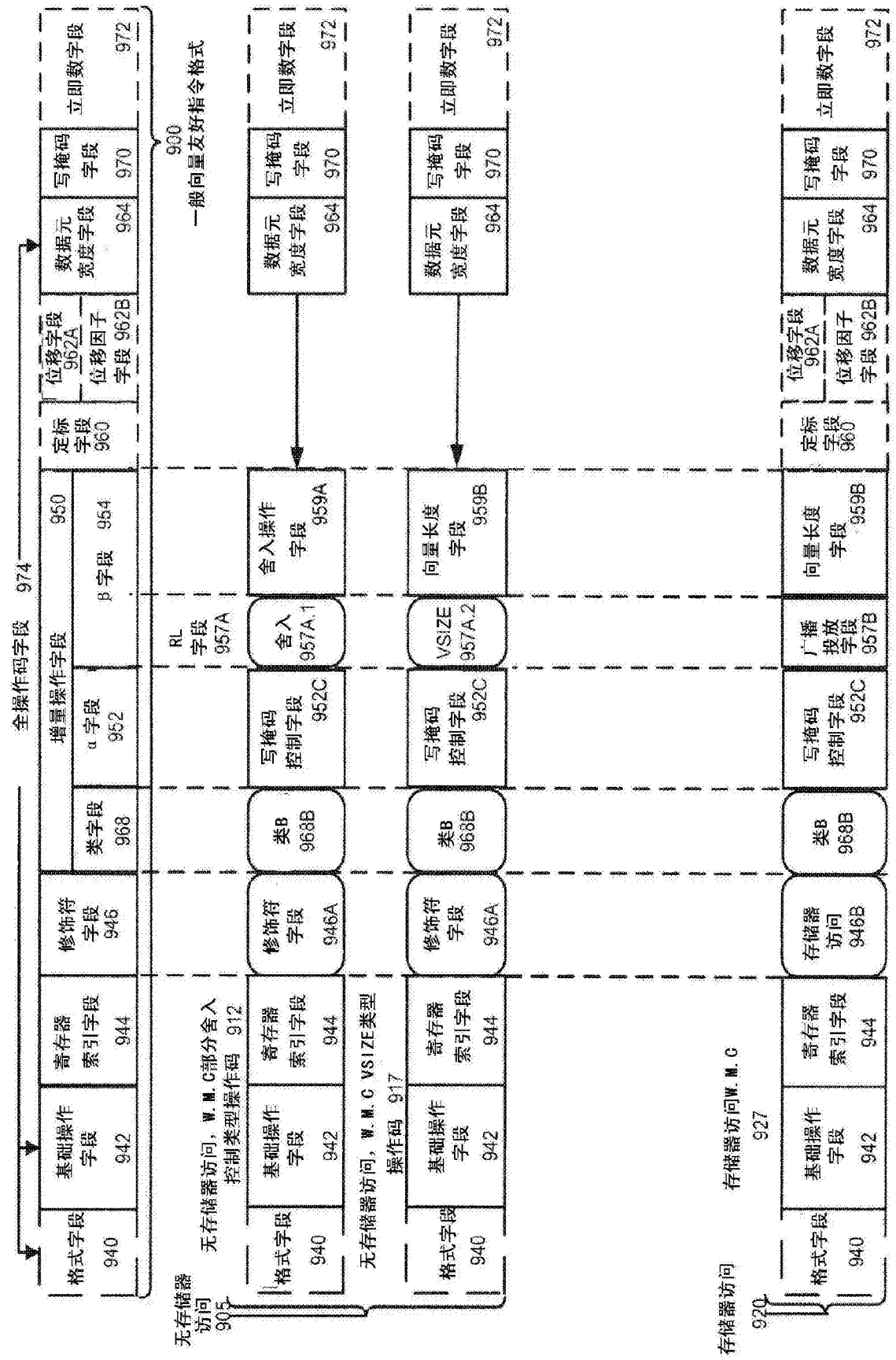

全操作码字段 974
900
一般向量友好指令格式
格式字段 940
基础操作字段 942
寄存器索引字段 944
修饰符字段 946
类字段 968
增量操作字段 950
α字段 952
β字段 954
定标字段 960
位移字段 962A
位移因子字段 962B
数据元宽度字段 964
写掩码字段 970
立即数字段 972
无存储器访问 905
无存储器访问，W.M.C部分舍入控制类型操作码 912
RL字段 957A
舍入操作字段 959A
写掩码控制字段 952C
舍入 957A.1
类B 968B
修饰符字段 946A
无存储器访问，W.M.C VSIZE类型操作码 917
VSIZE 957A.2
向量长度字段 959B
存储器访问 920
存储器访问W.M.C 927
存储器访问 946B
广播投放字段 957B

图9B

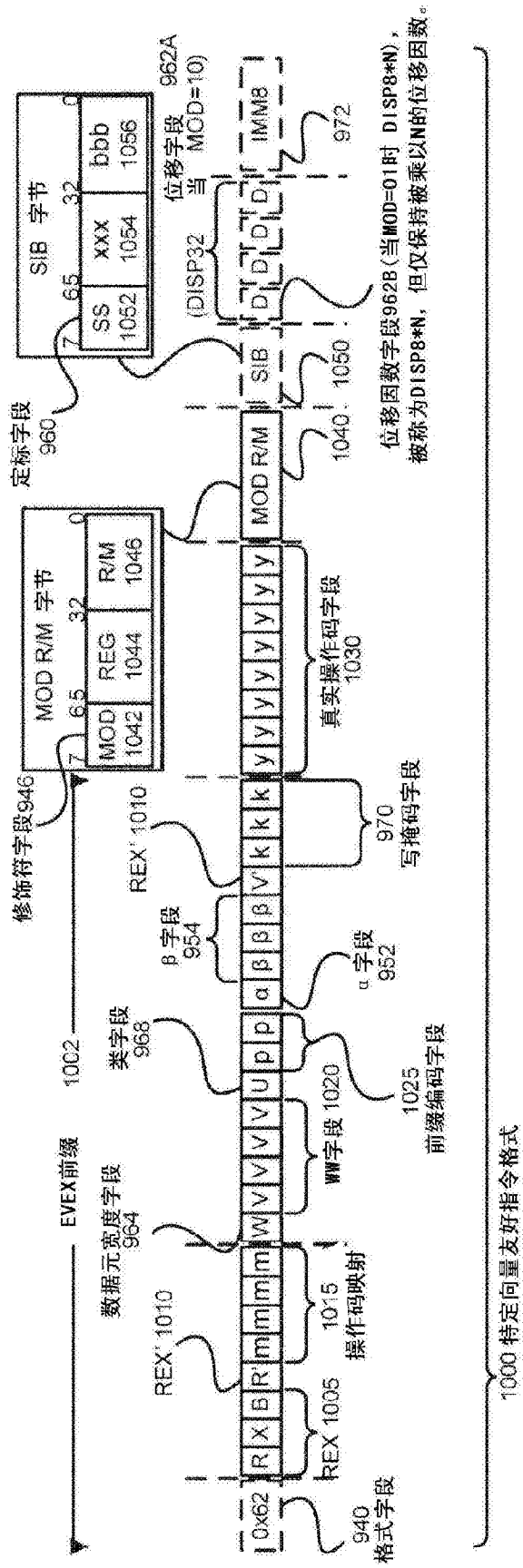

修饰符字段946
EVEX前缀
1000 特定向量友好指令格式
格式字段
940
0x62
REX 1005
REX' 1010
操作码映射
1015
数据元宽度字段
964
WW字段 1020
类字段
968
1025
前缀编码字段
α 字段
952
β 字段
954
REX' 1010
写掩码字段
970
真实操作码字段
1030
MOD R/M 字节
MOD
1042
REG
1044
R/M
1046
定标字段
960
SIB 字节
SS
1052
xxx
1054
bbb
1056
位移字段
当MOD=10)
962A
(DISP32
SIB
MOD R/M
1040
1050
IMM8
972
位移因数字段962B(当MOD=01时 DISP8*N),
被称为DISP8*N，但仅保持被乘以N的位移因数。

图10A

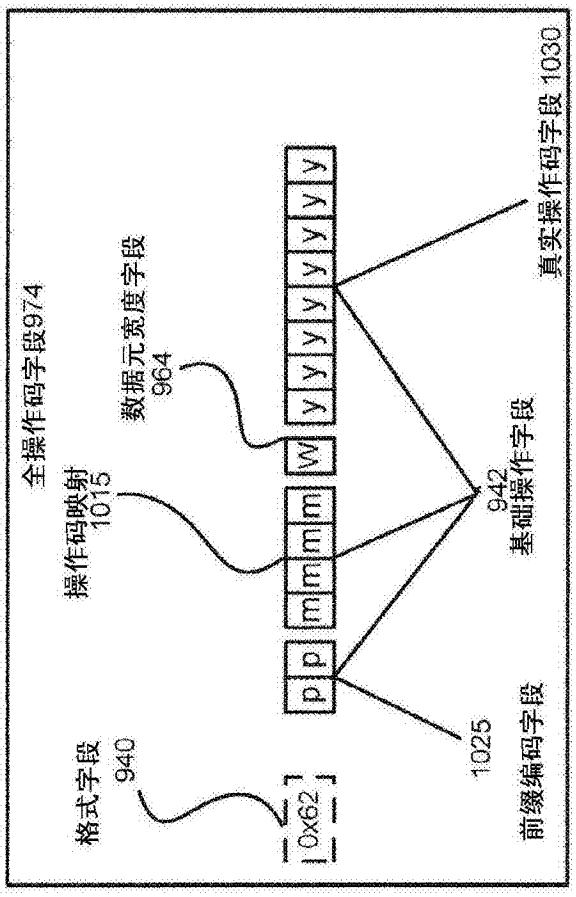

全操作码字段974
格式字段
940
0x62
操作码映射
1015
数据元宽度字段
964
1025
前缀编码字段
942
基础操作字段
真实操作码字段 1030

图10B

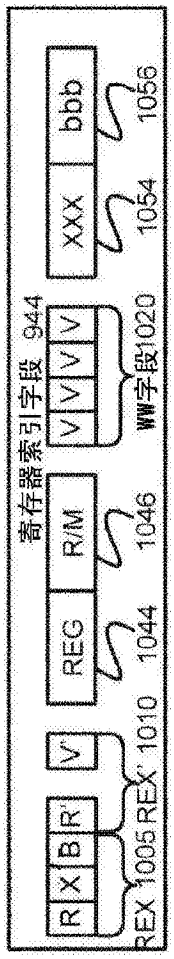

寄存器索引字段 944
R
X
B
R'
REX 1005
V'
REX' 1010
REG
1044
R/M
1046
V
V
V
V
WW字段1020
XXX
1054
bbb
1056

图10C

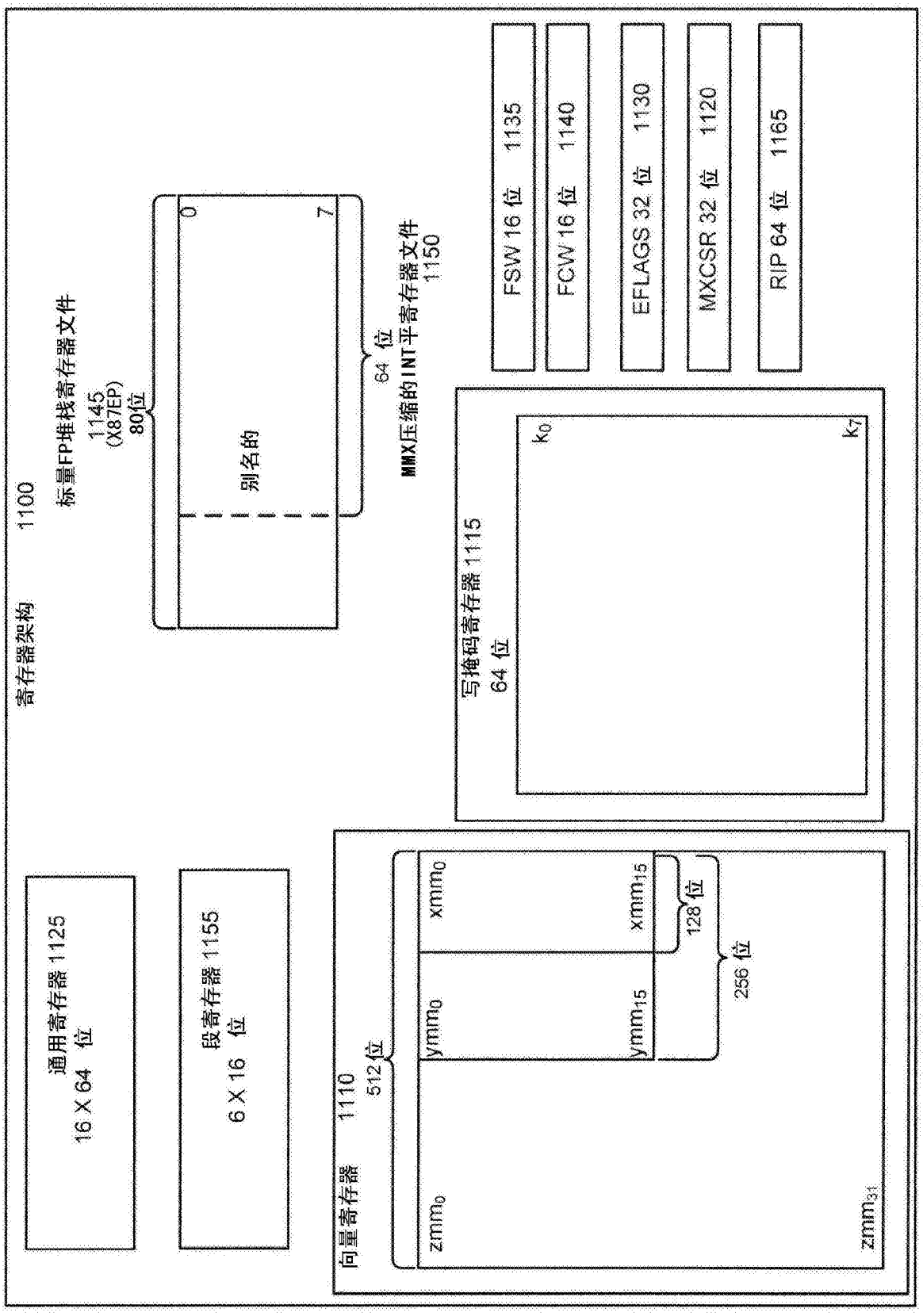

寄存器架构 1100
通用寄存器 1125
16 X 64 位
段寄存器 1155
6 X 16 位
向量寄存器 1110
512位
zmm0
ymm0
xmm0
ymm15
xmm15
128 位
256 位
zmm31
写掩码寄存器 1115
64 位
k0
k7
标量FP堆栈寄存器文件
1145
(X87EP)
80位
0
7
别名的
64 位
MMX压缩的INT平寄存器文件
1150
FSW 16 位 1135
FCW 16 位 1140
EFLAGS 32 位 1130
MXCSR 32 位 1120
RIP 64 位 1165

图11

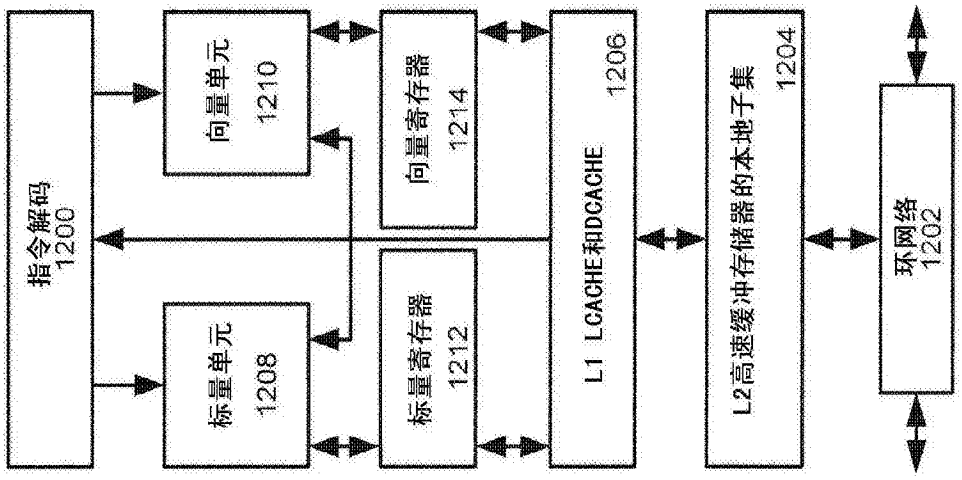

指令解码
1200
向量单元
1210
标量单元
1208
向量寄存器
1214
标量寄存器
1212
L1 LCACHE和IDCACHE
1206
L2高速缓冲存储器的本地子集
1204
环网络
1202

图12A

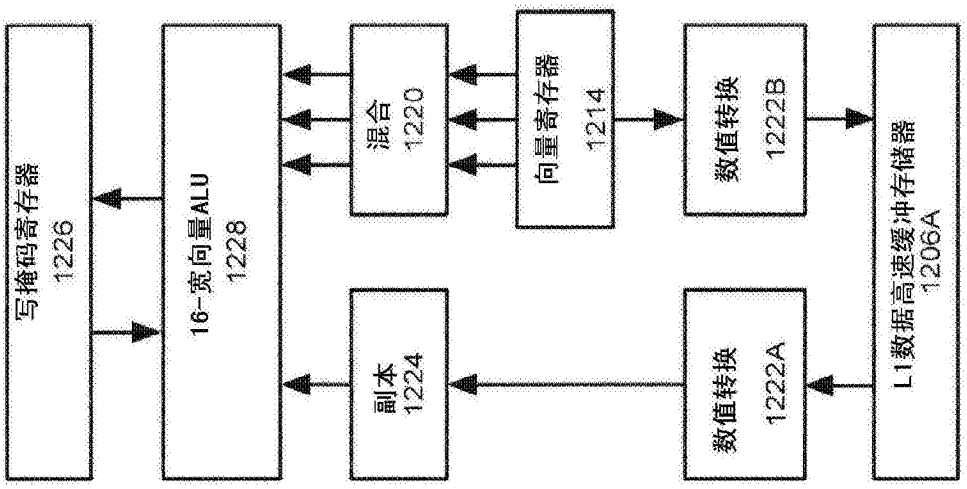

写掩码寄存器
1226
16-宽向量ALU
1228
混合
1220
向量寄存器
1214
数值转换
1222B
副本
1224
数值转换
1222A
L1数据高速缓冲存储器
1206A

图12B

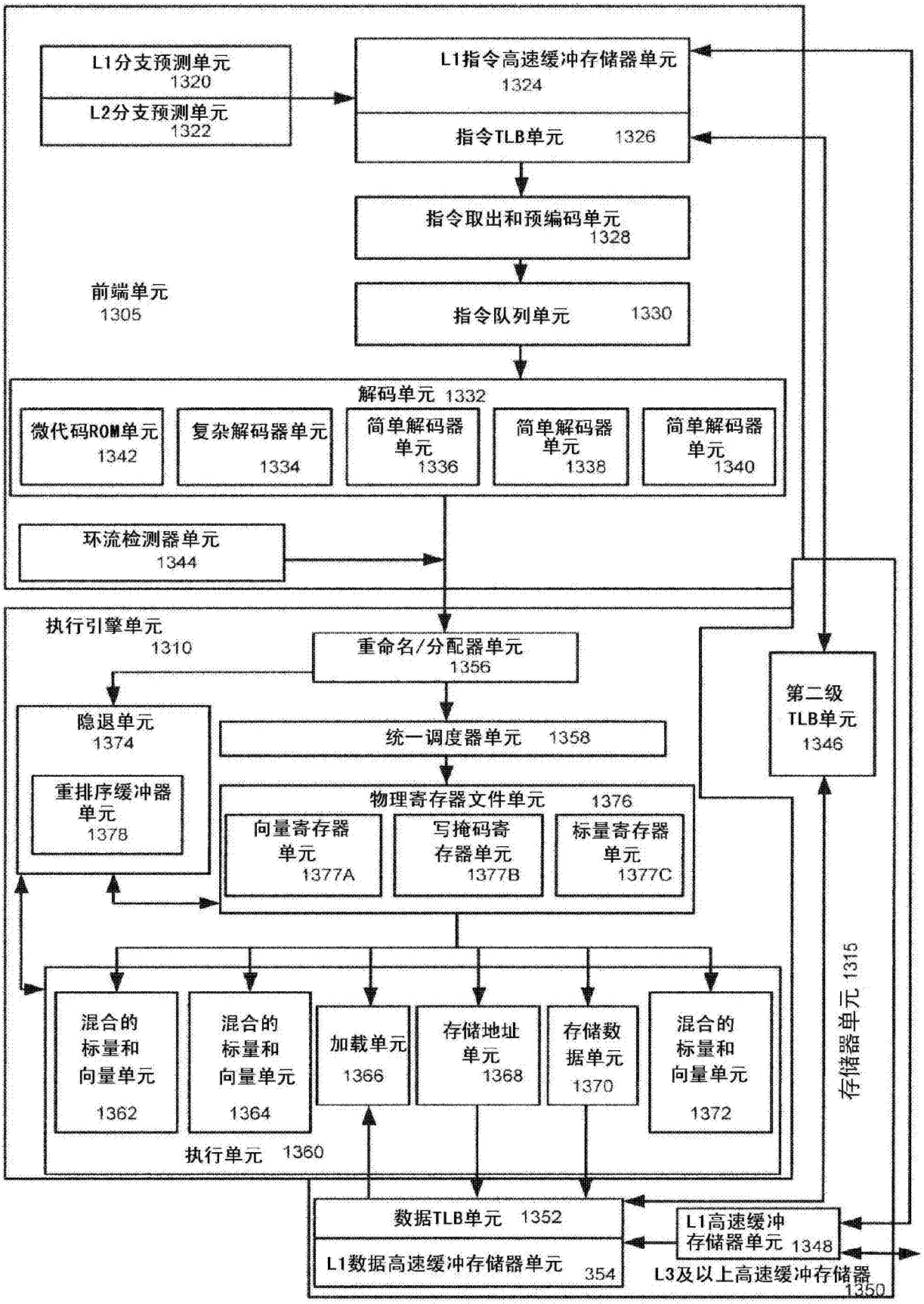

L1分支预测单元
1320
L2分支预测单元
1322
L1指令高速缓冲存储器单元
1324
指令TLB单元 1326
指令取出和预编码单元
1328
前端单元
1305
指令队列单元 1330
解码单元 1332
微代码ROM单元
1342
复杂解码器单元
1334
简单解码器单元
1336
简单解码器单元
1338
简单解码器单元
1340
环流检测器单元
1344
执行引擎单元
1310
重命名/分配器单元
1356
第二级TLB单元
1346
隐退单元
1374
统一调度器单元 1358
重排序缓冲器单元
1378
物理寄存器文件单元 1376
向量寄存器单元
1377A
写掩码寄存器单元
1377B
标量寄存器单元
1377C
混合的标量和向量单元
1362
混合的标量和向量单元
1364
加载单元
1366
存储地址单元
1368
存储数据单元
1370
混合的标量和向量单元
1372
存储器单元 1315
执行单元 1360
数据TLB单元 1352
L1数据高速缓冲存储器单元 354
L1高速缓冲存储器单元 1348
L3及以上高速缓冲存储器
1350

图13

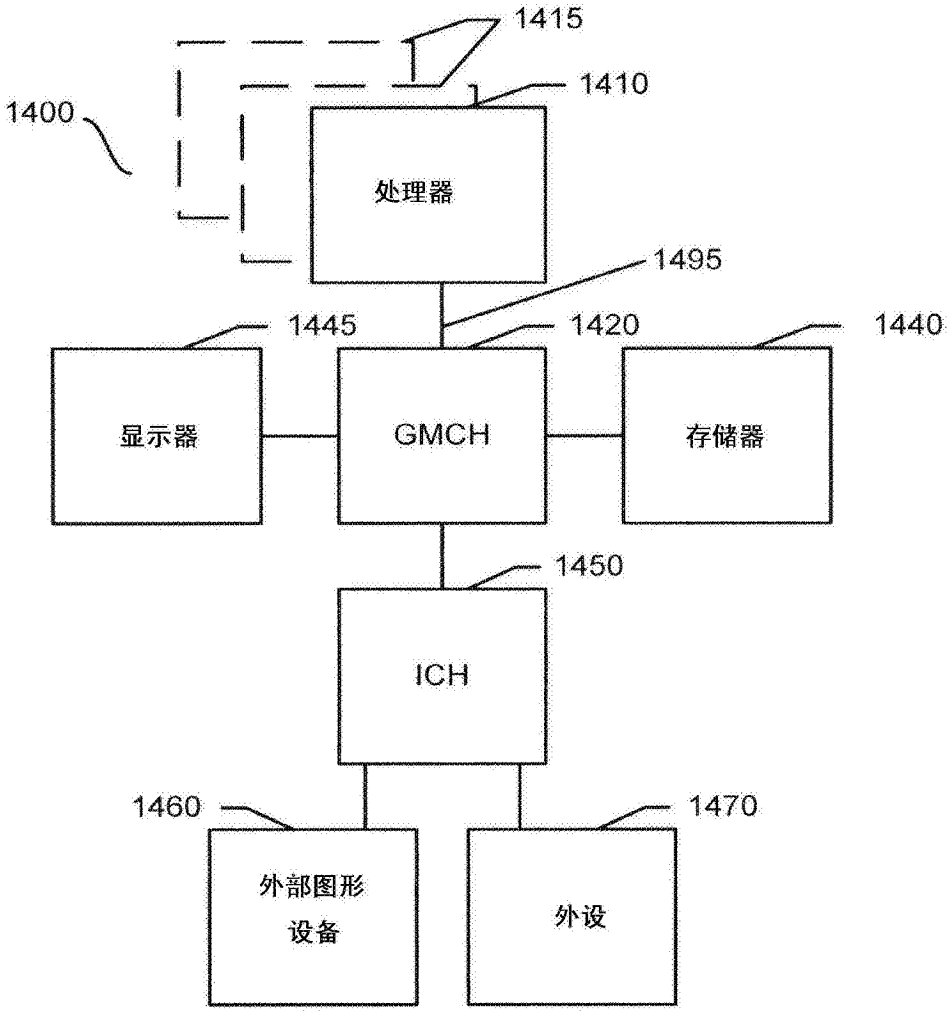

1415
1410
1400
处理器
1495
1445
1420
1440
显示器
GMCH
存储器
1450
ICH
1460
1470
外部图形
设备
外设

图14

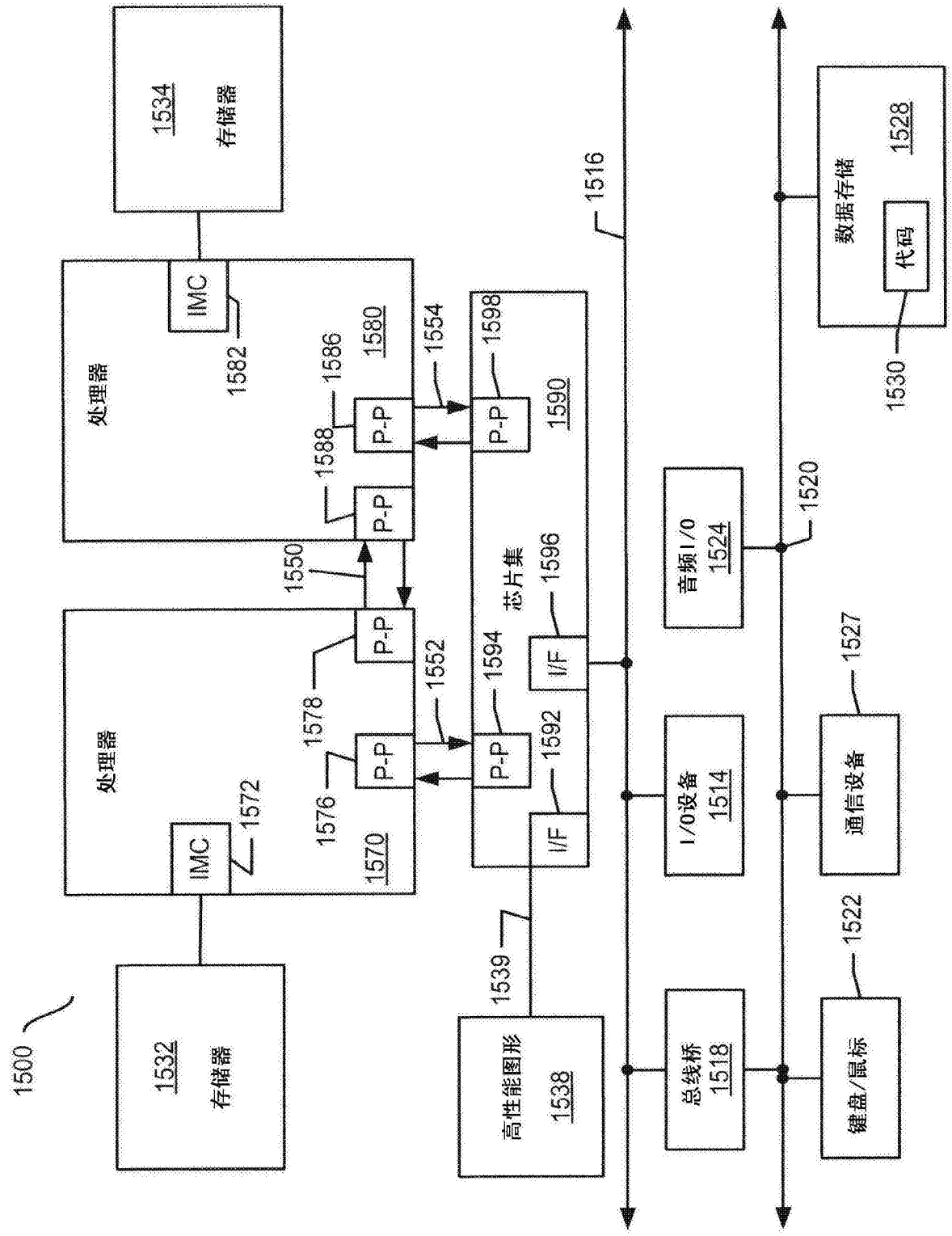

1500
1532
存储器
1534
存储器
处理器
IMC
1572
1582
处理器
IMC
1570
1580
1576
1578
1586
1588
1550
1552
1554
1594
1592
1598
1590
1596
芯片集
P-P
P-P
P-P
P-P
P-P
P-P
I/F
I/F
1539
高性能图形
1538
1516
总线桥
1518
I/O设备
1514
音频I/O
1524
1520
1527
通信设备
1522
键盘/鼠标
数据存储
1528
代码
1530

图15

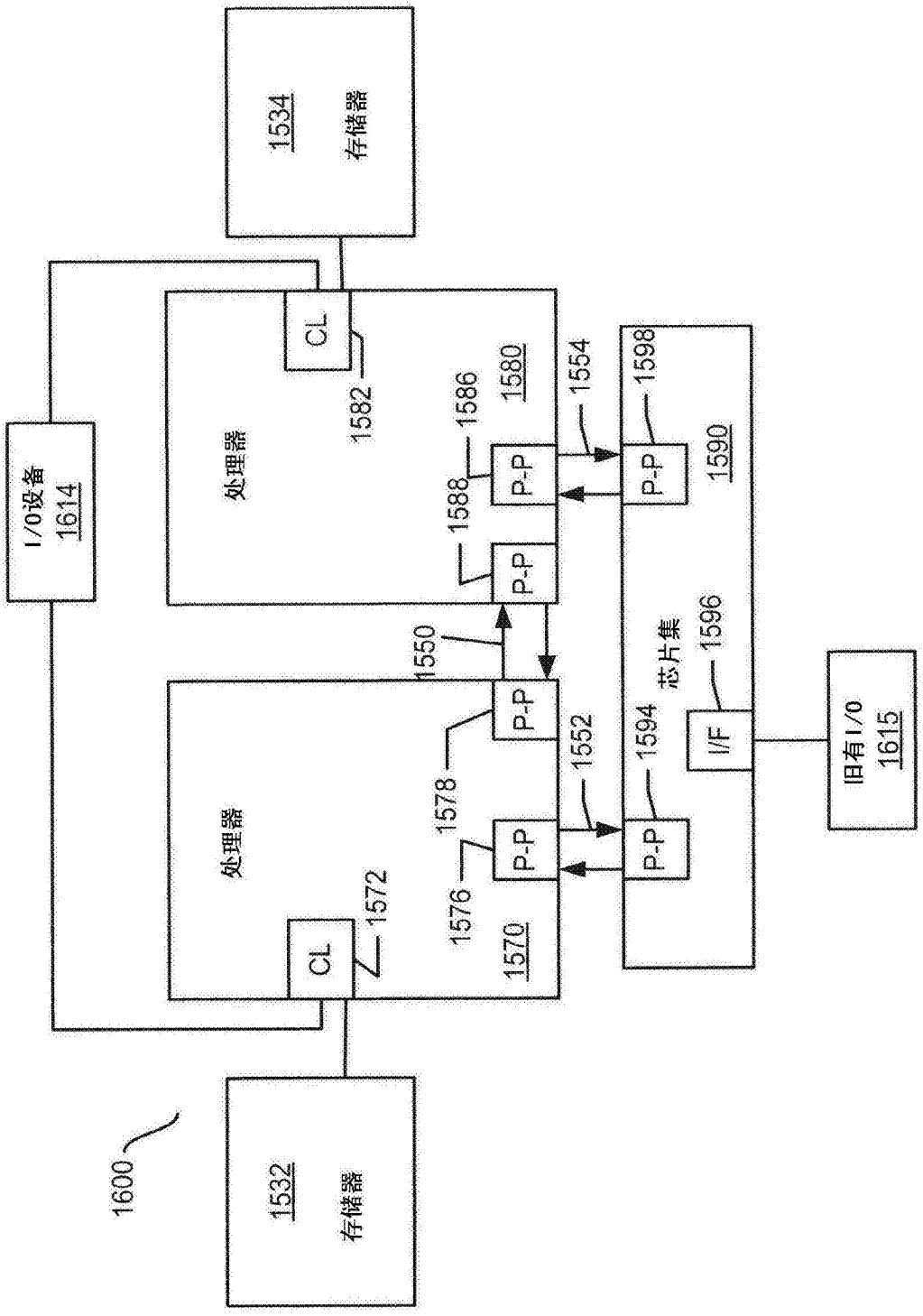

1534
存储器
CL
1582
处理器
1580
1586
1588
P-P
1554
1598
1590
I/O设备
1614
1550
1578
1552
1594
芯片集
1596
I/F
旧有I/O
1615
1576
1572
1570
1600
1532
存储器

图16

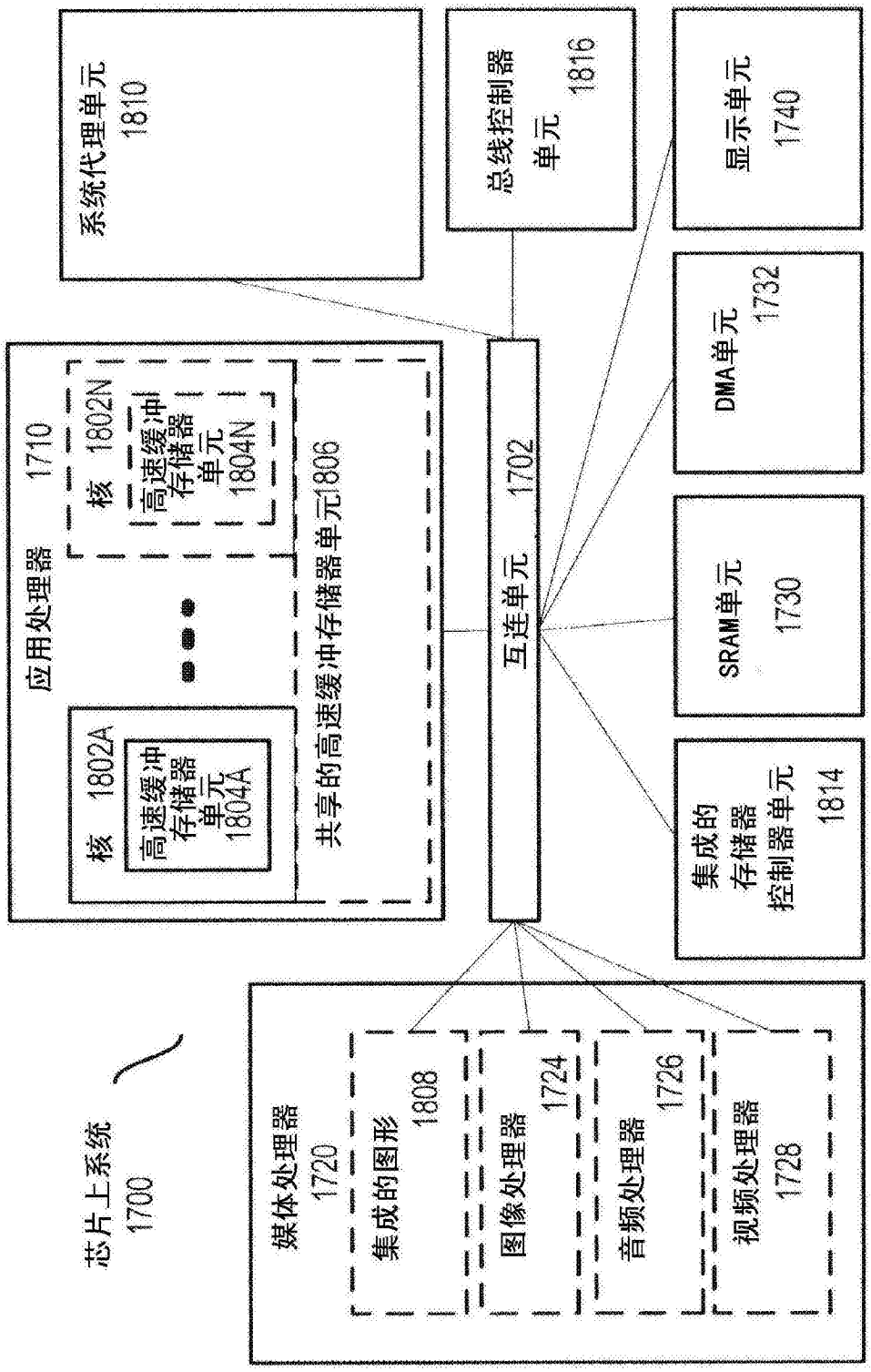

芯片上系统
1700
媒体处理器
1720
集成的图形
1808
图像处理器
1724
音频处理器
1726
视频处理器
1728
应用处理器
1710
核 1802A
高速缓冲存储器单元 1804A
核 1802N
高速缓冲存储器单元 1804N
共享的高速缓冲存储器单元1806
互连单元 1702
系统代理单元
1810
总线控制器单元
1816
显示单元
1740
DMA单元
1732
SRAM单元
1730
集成的存储器控制器单元
1814

图17

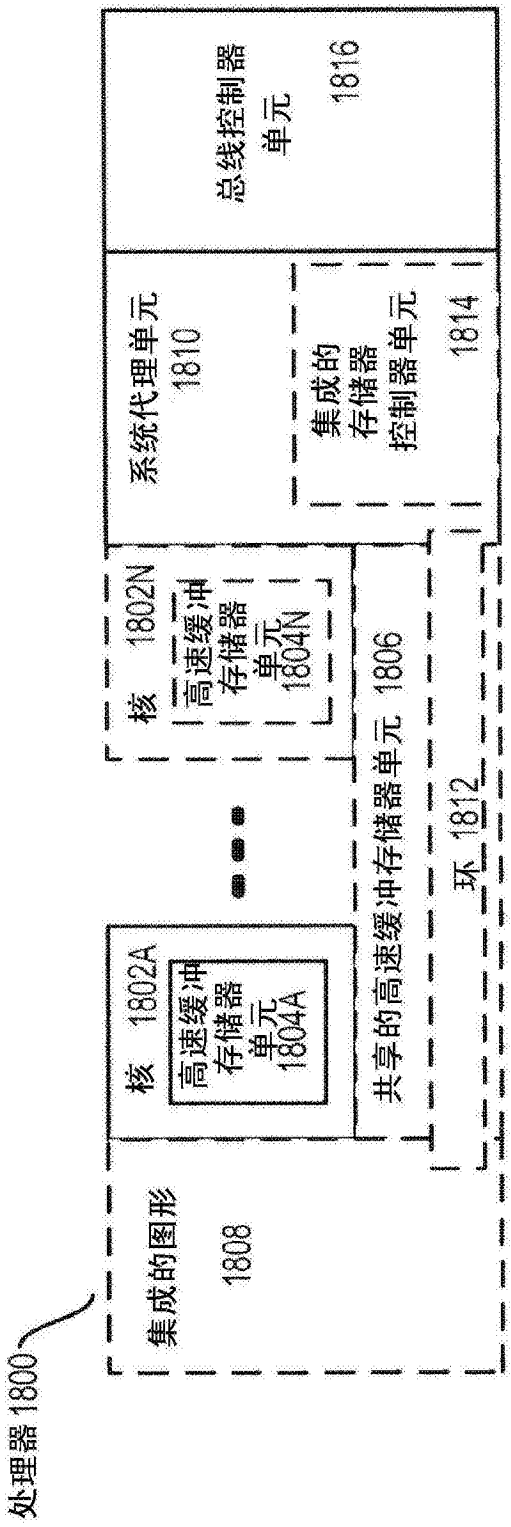

处理器 1800
集成的图形
1808
核 1802A
高速缓冲存储器单元 1804A
核 1802N
高速缓冲存储器单元 1804N
共享的高速缓冲存储器单元 1806
环 1812
系统代理单元 1810
集成的存储器控制器单元 1814
总线控制器单元 1816

图18

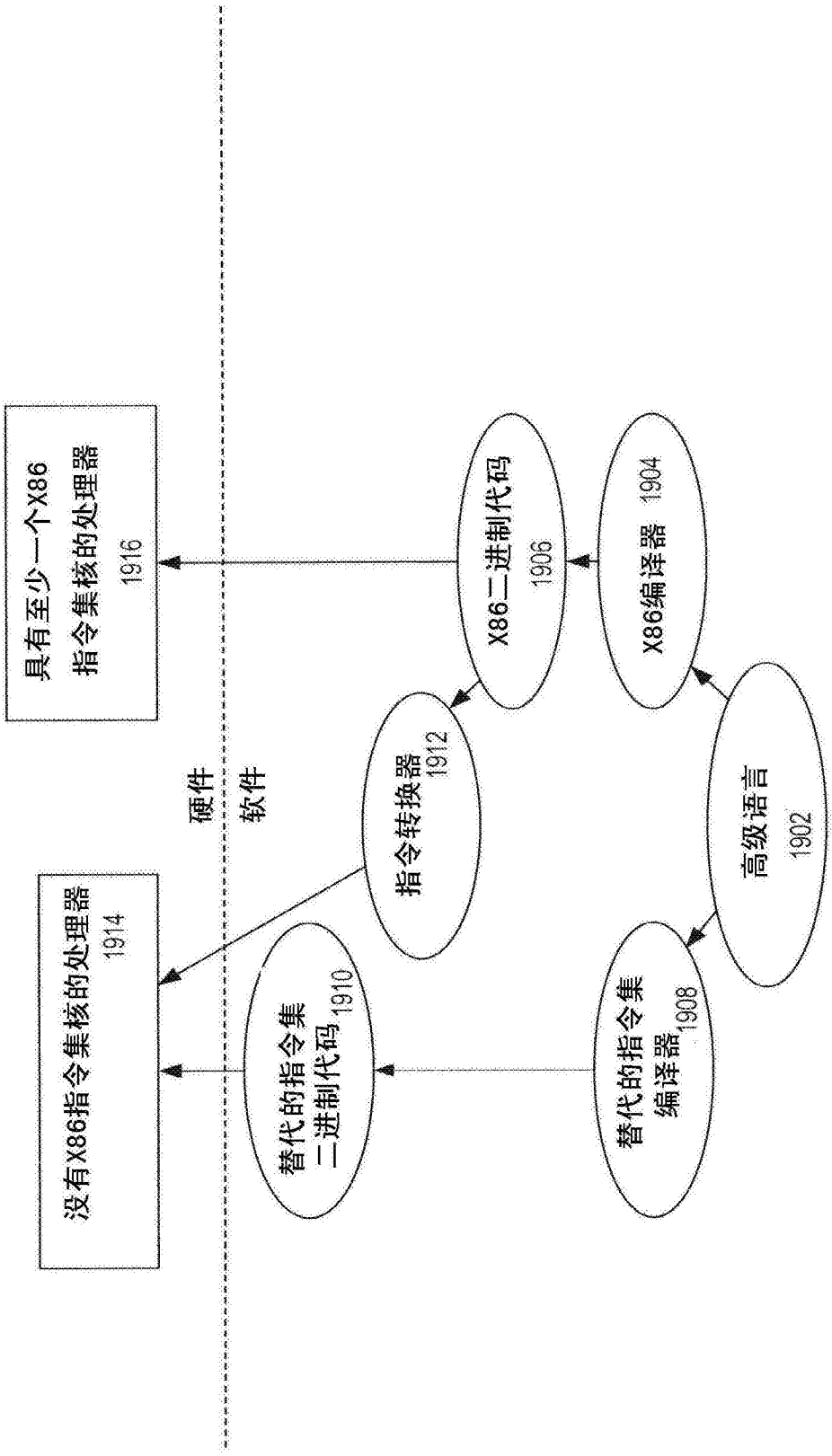

具有至少一个X86
指令集核的处理器
1916
硬件
软件
没有X86指令集核的处理器
1914
X86二进制代码
1906
X86编译器
1904
指令转换器
1912
高级语言
1902
替代的指令集
二进制代码
1910
替代的指令集
编译器
1908

图19