(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 108701024 A
(43)申请公布日 2018.10.23

(21)申请号 201780013704.4

(22)申请日 2017.02.27

(30)优先权数据
PCT/EP2016/000344 2016.02.27 EP
PCT/EP2016/000345 2016.02.27 EP

(85)PCT国际申请进入国家阶段日
2018.08.27

(86)PCT国际申请的申请数据
PCT/EP2017/054532 2017.02.27

(87)PCT国际申请的公布数据
WO2017/144728 EN 2017.08.31

(71)申请人 金辛格自动化有限责任公司
地址 德国拉施塔特

(72)发明人 K·金辛格

(74)专利代理机构 永新专利商标代理有限公司 72002
代理人 郭毅

(51)Int.Cl.
G06F 9/30(2006.01)
G06F 9/34(2006.01)
G06F 9/35(2006.01)

权利要求书2页 说明书4页 附图4页

(54)发明名称

用于在堆栈机器中分配虚拟寄存器堆栈的方法

(57)摘要

要解决的问题是寻求已知的指令集架构的替代方案,其提供相同或相似的效果或更更成本有效。解决方案:该问题通过在堆栈机器中分配处理单元的虚拟寄存器堆栈(10)的方法来解决,所述方法包括在所述堆栈机器的物理寄存器文件(17)中分配所述虚拟寄存器堆栈(10)的给定数目的最顶层元素(11),并且在所述堆栈机器的分层寄存器缓存(13)中分配所述虚拟寄存器堆栈(10)的后续元素。

CN 108701024 A

1.一种用于在堆栈机器中分配处理单元的虚拟寄存器堆栈(10)的方法,其包括:

在所述堆栈机器的物理寄存器文件(17)中分配所述虚拟寄存器堆栈(10)的给定数目的最顶层元素(11),并且在所述堆栈机器的分层寄存器缓存(13)中分配所述虚拟寄存器堆栈(10)的附加元素。

2.根据权利要求1所述的方法,其中,

所述寄存器文件(17)包括多个寄存器,每个寄存器能够由从零开始的寄存器号(14)唯一寻址,

所述虚拟寄存器堆栈(10)的每个元素能够通过从零开始的索引(15)唯一寻址,以及

为了分配所述最顶层的元素(11)中的一个元素,将能够用于对所述元素寻址的索引(15)映射到所述寄存器号(14)中的一个寄存器号(14)。

3.根据权利要求2所述的方法,其中,

通过和与所述寄存器文件(17)内的处理单元的寄存器的给定容量对应的位掩码(RB)进行逐位逻辑与、然后与所述寄存器在所述寄存器文件(17)中的偏移(RO)进行逐位逻辑或来映射所述索引(15)。

4.根据权利要求3所述的方法,其中,

如果寄存器的容量耗尽,则所述虚拟寄存器堆栈(10)的任何剩余元素溢出到所述分层寄存器缓存(13)。

5.根据权利要求3或4所述的方法,其中,所述处理单元是虚拟的,并且所述寄存器包含在所述寄存器文件(17)的由所述虚拟处理单元的专用寄存器(21)定义的段(12)中,所述专用寄存器(21)包括所述位掩码(RB)和所述偏移(RO)。

6.根据权利要求5所述的方法,其包括:

通过引用所述虚拟处理单元的进程指针(PP)管理所述专用寄存器(21),所述专用寄存器(21)优选地包括唯一任务标识符(TID)、进程标识符(PID)、指令指针(IP)、虚拟寄存器堆栈指针(SP)、数据堆栈指针(DP)和返回堆栈指针(BP)。

7.根据权利要求5或6所述的方法,其包括,

在激活时,分配所述寄存器文件(17)中的所述段(12)并加载所述虚拟处理单元,

在停用时,卸载所述虚拟处理单元并释放所述段(12)。

8.根据以上权利要求中任一项所述的方法,其中,将所述最顶层的元素(11)进一步备份到所述寄存器缓存(13)中。

9.根据权利要求8所述的方法,其中,

所述寄存器缓存(13)包括多个寄存器,每个寄存器能够由从零开始的地址(22)唯一寻址,并且为了分配后续元素中的一个元素,将能够用于对所述元素寻址的索引(15)映射到相应的地址(22)。

10.根据权利要求9所述的方法,其中,

通过与与所述寄存器缓存(13)的容量对应的位掩码(RB)进行逐位逻辑与、然后与所述寄存器缓存(13)内的偏移(RO)进行逐位逻辑或来映射所述索引(15)。

11.根据权利要求10所述的方法,其中,

如果所述寄存器缓冲(13)的容量耗尽,则所述虚拟寄存器堆栈(10)的任何剩余的最底部的元素(16)溢出到随机存取存储器。

12.一种堆栈机器，其适于执行根据权利要求1至11中任一项所述的方法的步骤。

13.一种计算机程序，其包括使根据权利要求12所述的堆栈机器执行根据权利要求1至11中任一项所述的方法的步骤的指令。

14.一种计算机可读的数据载体，在其上存储有根据权利要求13所述的计算机程序。

15.一种计算机实现的数据结构(17)，其应用在根据权利要求1至11中任一项所述的方法中，所述数据结构包括：

多个段(12、18、19、20)，每个段(12、18、19、20)包含虚拟处理单元的给定数目的寄存器并且由所述虚拟处理单元的专用寄存器(21)定义，所述专用寄存器(21)包括所述寄存器在所述结构(17)内的偏移(RO)。

16.根据权利要求15所述的数据结构(17)，其中，所述寄存器的数目是2的幂，所述虚拟处理单元的专用寄存器(21)还包括与所述寄存器的数目相对应的位掩码(RB)。

用于在堆栈机器中分配虚拟寄存器堆栈的方法

技术领域

[0001] 本发明涉及一种用于在堆栈机器中分配虚拟寄存器堆栈的方法以及相应的堆栈机器、计算机程序、数据载体和数据结构。

背景技术

[0002] 在计算机科学、计算机工程和编程语言实现中，堆栈机器是指一种处理器，其指令在下推堆栈上而不是在寄存器上操作。传统的堆栈机器通常具有无限大小的堆栈，可以用硅实现或者在标准寄存器机上通过软件模拟。由于大多数机器指令集架构(ISA：Instruction Set Architectures)仅仅定义有限数目的所谓的架构寄存器，因此传统堆栈机器由于低效的堆栈寄存器映射而遭受性能损失。诸如Forth、RPL和各种汇编语言的面向堆栈的编程语言在很大程度上依赖于这种堆栈机器实现。

[0003] 一种已知的基于硅的堆栈机器实现已经在Hand，Tom所著的“The Harris RTX 2000Microcontroller”(Journal of Forth Application and Research6.1(1990年)：第5-13页)中公开。对于在编译器设计的上下文中软件模拟堆栈机器的寄存器分配方法的讨论，参见SHANNON，Mark；BAILEY，Chris所著的“全局堆栈分配——用于堆栈机器的寄存器分配(在2006年Euroforth会议议程中)”。

发明内容

[0004] 所要求保护的发明在下文中以这样的方式公开，即可以理解其所处理的技术问题并且可以理解该解决方案。

[0005] 技术问题

[0006] 要解决的问题是寻求已知指令集架构的替代方案，其提供相同或相似的效果或更成本有效。

[0007] 问题的解决方案

[0008] 为了阐明解决方案的性质，参考独立权利要求的特征部分。

[0009] 发明的有利效果

[0010] 本发明引起高效的数据处理、高效的数据存储和增强的安全性。

附图说明

[0011] 图1示出处理单元的进程管理；

[0012] 图2示出寄存器文件；

[0013] 图3示出逐个元素增大的虚拟寄存器堆栈的简单示例；

[0014] 图4示出相同用例的更复杂的实例。

具体实施方式

[0015] 本发明引入处理器特定架构的抽象化并且命名为通用寄存器，这些寄存器是所有

已知的现有技术处理器的和ISA的关键设计元素。(这些通用寄存器在下文中将简称为“寄存器”并且与保存诸如堆栈指针之类的程序状态或对应于专用硬件元件的专用寄存器区分开。)为此,实施例创建了一种通用抽象类型的寄存器,该寄存器在虚拟寄存器空间中声明并且通过ISA操作码借助于虚拟地址(与现有技术的单独“寄存器名称”相对)来寻址。在这个意义上,本发明定义了一种堆栈机器,其包括完全可寻址的虚拟寄存器堆栈和ISA。

[0016] 如图1所示,在该堆栈机器上运行的每个进程至少暂时具有其自己的处理单元,该处理单元具有一组私有专用寄存器(21),其由硬件管理并且基本上对于应用软件而言是不可访问的。这些专用寄存器组(21)借助于进程指针(PP:Process Pointer)选择。特定地,本实施例的专用寄存器组尤其可以包括唯一任务标识符(TID:Task Identifier)、进程标识符(PID:Process Identifier)、指令指针(IP:Instruction Pointer)、寄存器偏移(RO:Register Offsets)、寄存器位掩码(RB:Register Bitmasks)、虚拟寄存器堆栈指针(SP:Stack Pointer)、数据堆栈指针(DP:Data Stack Pointer)和返回堆栈指针(BP:Return Stack Pointer)。在任何时刻,处理单元可以是活动的,即加载的,或者是非活动的,即卸载到存储器分层的任何不同层级。

[0017] 图2阐明机器的物理寄存器文件(17),其实际大小是可缩放的设计参数。如同从块示意图中可以看出的那样,寄存器文件(17)展示出由多个段组成的数据结构,所述多个段中的每一个包含特定的——在本示例中为“虚拟的”——处理单元的物理寄存器。对于任何这样的单元,关于图1提到的位掩码(RB)和偏移(RO)分别定义了在寄存器文件(17)内其寄存器的数目和位置。为此,每个寄存器可由从零开始的寄存器号(14)唯一寻址。作为一个示例,考虑包含编号为20、21、22和23的四个寄存器的段(12)。当这些寄存器分配在虚拟处理单元中时,后者的位掩码(RB)将设置为$00011_2=3_{10}$并且其偏移(RO)将设置为$10100_2=20_{10}$。

[0018] 图2的整体存储器组织方案也适用于存储器分层的更高层级,如图4中针对单个进程描绘的那样。

[0019] 图3例示了所述虚拟处理单元的虚拟寄存器堆栈(10)的连续增大,其包含子程序(subroutine)上下文、调用、返回和局部变量以及用作可寻址的操作数的中间计算结果。考虑到该单元的物理寄存器的数目,虚拟寄存器堆栈(10)的最顶层元素(11)——在这种情况下在任何给定时间最多四个这样的元素——被分配在寄存器文件(17-图2)的相应的段(12)中,而所有后续元素(23)都“溢出”,即分配在随机存取存储器中。所述虚拟寄存器堆栈上的导致其增大和缩小的推送和弹出操作由大多数ISA操作码、即软件隐式触发,并相应地由硬件执行。因此,寄存器文件段(12)与其所连接的随机存取存储器段(10)之间的任何数据传输都是根据延迟存储(lazy store)和加载策略自动执行的,这意味着寄存器溢出和填充是硬件自动的并且仅仅当必要时才执行,从而避免程序不需要或者尚未需要的数据值的冗余存储操作(延迟存储)和“虚加载(dummy load)”操作(延迟加载)。

[0020] 虚拟寄存器堆栈(10)的每个元素可通过从零开始的索引(15)唯一寻址。为了物理地分配该元素,该元素的相应的索引(15)通过与单元的位掩码(RB)进行逐位逻辑与(在本领域中称为“位屏蔽”的操作),然后与其偏移(RO)进行逐位逻辑或来映射。例如,根据这种机制,索引$7_{10}=00111_2$将被映射到物理寄存器$(00111_2 \text{AND } 00011_2) \text{OR } 10100_2=00011_2 \text{OR } 10100_2=10111_2=23_{10}$。因此,由该索引寻址的虚拟寄存器#7被分配在物理寄存器#23中。然而,应注意,贯穿进程的整个运行时间地,虚拟寄存器和物理寄存器之间绝不存在一对一的

对应关系，因为物理寄存器#23先前已用于存储虚拟寄存器#3，并且在超出图3范围的未来周期中可以很好地存储虚拟寄存器#11、#15、#19等而不背离本发明。另外还应注意，如果多个虚拟寄存器在物理寄存器中竞争分配，则具有最高索引的那个虚拟寄存器(11)获胜；剩余的溢出到随机存取存储器(23)。还要注意，在这种情况下，上面采用的可兼或或者交替产生的结果与异或XOR相同。

[0021] 该方法具有以下优点：寄存器文件(17)的相应段(12)被组织为类似于环形缓冲器，有时被称为圆形缓冲器、圆形队列或循环缓冲器。因此，当元素被推入虚拟寄存器堆栈(10)或从虚拟寄存器堆栈(10)弹出时，不需要使段(12)的内容移位或对段(12)的内容混洗(Shuffle)。此外，位于堆栈顶部附近的虚拟寄存器——其可能由机器的算术逻辑单元作为操作数访问——倾向于存储在易于访问的寄存器文件(17)中，而不是较慢且能效较低的缓存或甚至是片外存储器。因此，这里发明的堆栈机器展示出整体改进的能效和性能，这使其甚至适用于硬实时应用。在施加较少严格限制的用例中，本发明的实施例仍然允许降低的能量消耗，因为需要较小的寄存器文件来满足相同的运行时目标。

[0022] 图3的整体存储器组织和数据处理方案也适用于存储器分层的更高层级，如在图4中对于单个进程示出的那样。对于图3的虚拟寄存器堆栈(10)、随机存取存储器(23)和寄存器文件段(12)附加地，图4的增强场景还引入寄存器缓存分层的第一层级(13)，在其中分配虚拟寄存器堆栈(10)的附加元素，这些附加元素不能被段(12)容纳。虽然这种分层(13)可以包括各种类型的级联存储介质，如寄存器文件、片上或片外随机存取存储器或文件，但这些介质在下文中将统称为分层寄存器缓存(13)，术语“缓存”以广义使用。

[0023] 缓存分层(13)的每个层级还部分地用作所述缓存分层的包括“缓存层级0”在内的意味着寄存器文件(17)本身的所有较低层级的备份。在这方面，相关的存储器位置(24)构成两用阴影缓冲器，其在稳态进程运行时条件下用于延迟存储和加载操作，并且在进程中从其虚拟处理单元的卸载被重新用作备份存储介质保留从缓存分层的较低层级驱逐的虚拟寄存器，而在进程重新加载时，从所述备份存储介质恢复所述缓存分层的较低层级中的虚拟寄存器。需要注意，进程卸载和相关联的重新加载可以是部分的，这意味着并非寄存器缓存分层的所有较低层级需要被驱逐和恢复，从而允许在进程切换期间针对存储器使用权衡的运行时性能。

[0024] 还要注意，除给定的缓存分层方案的最高层级以外，每个较低缓存层级(包括“缓存层级0”，寄存器文件)可以保持零虚拟寄存器(11)。因此，任何进程可以但不必拥有寄存器文件的允许访问高性能特征的段(12)，如例如多个算术逻辑单元到虚拟寄存器堆栈和相关的ILP(Instruction Level Parallelism，指令层级并行)的并行多端口访问。根据本发明，进程因此可以精确地细调以满足它们各自的寄存器性能要求，甚至可能的是，为根本不具有寄存器文件的低端微控制器应用构建堆栈机器。

[0025] 在应用软件层级，仅仅本发明所有实施例共有的无限虚拟寄存器空间是可见的，而硬件中虚拟寄存器的分配细节可能因处理器类型而异，从特殊寄存器设置到设置，甚至在程序运行时期间，对应用软件层没有任何效果或影响。在这个意义上，应用软件可以以硬件抽象的形式直接编译到在编译器技术中称为LLVM(Low Level Virtual Machine；低级虚拟机)，从而严格地将纯算法软件域与其物理硬件实现和配置域解耦。

[0026] 所述LLVM方法具有以下优点：计算机软件和硬件之间的通用接口可以由虚拟指令

集架构(VISA)来定义，该虚拟指令集架构(VISA)对于由本发明定义的整个计算机类是通用的，以这样的方式，为所述类的任何实施例编写的程序原则上将在所述类的任何其他实施例上运行，不仅关于硬件制造商、性能类和应用类型(软件可移植性)而且关于时间(软件持久性)，作为硬件技术可以是新的，进一步开发的，改变的，改进的或放弃的，而不具有致使所述计算机类的早期技术等级编写的渲染软件过时的副作用(软件向后和向前兼容性)。

[0027] 工业适用性

[0028] 本发明尤其可以应用于整个半导体工业中。

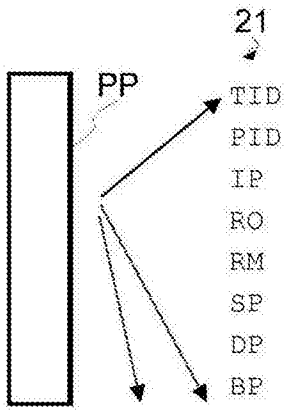

图1

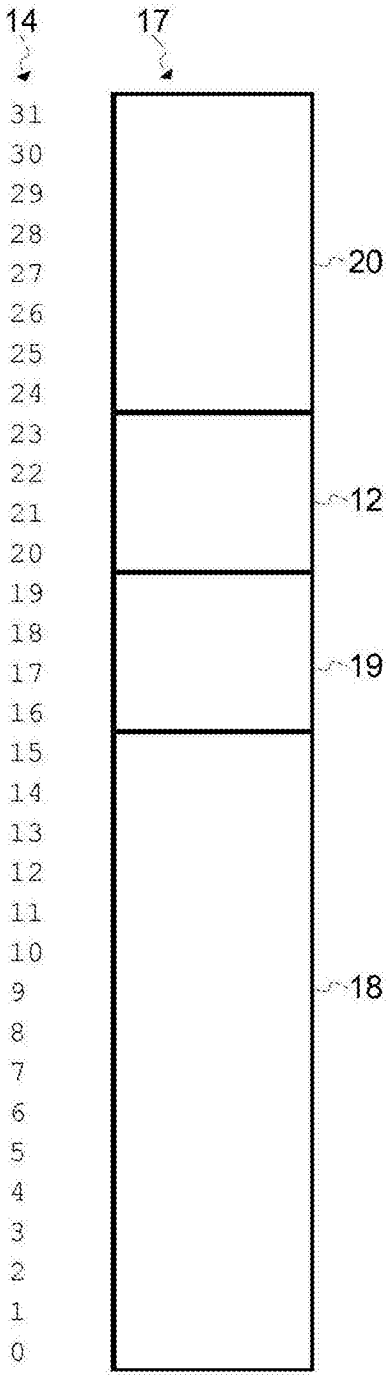
14
17
31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
20
12
19
18

图2

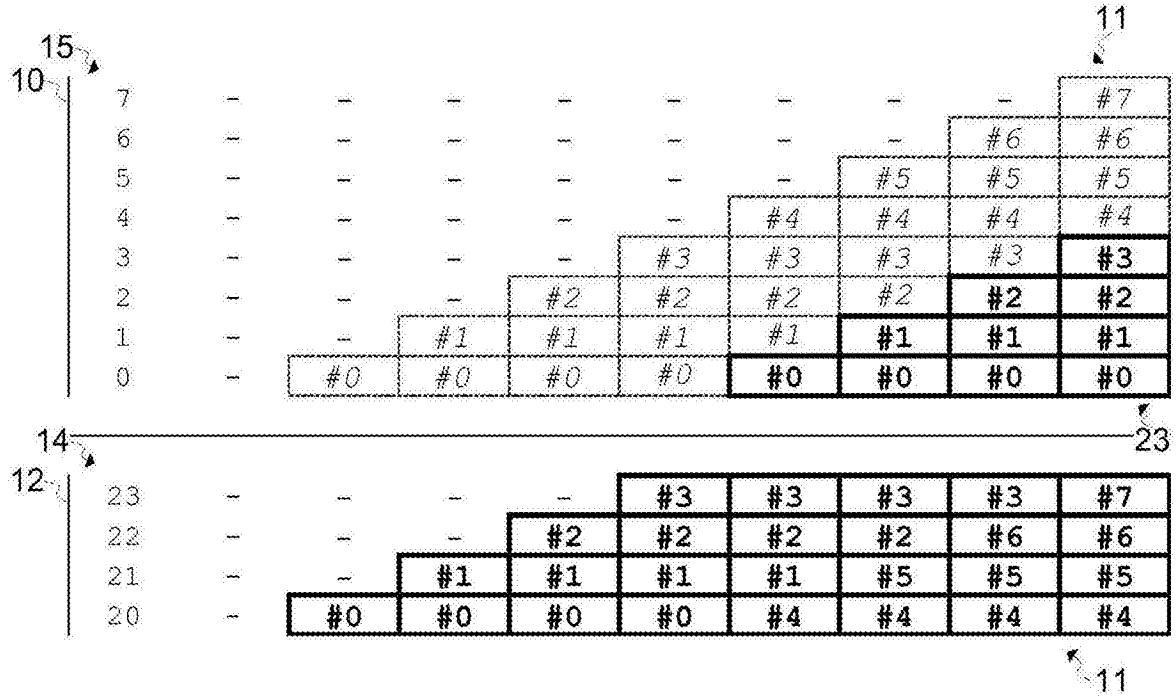
15
10
11
7 - - - - - - - - #7
6 - - - - - - - #6 #6
5 - - - - - - #5 #5 #5
4 - - - - - #4 #4 #4 #4
3 - - - - #3 #3 #3 #3 #3
2 - - - #2 #2 #2 #2 #2 #2
1 - - #1 #1 #1 #1 #1 #1 #1
0 - #0 #0 #0 #0 #0 #0 #0 #0
14
23
12
23 - - - - #3 #3 #3 #3 #7
22 - - - #2 #2 #2 #2 #6 #6
21 - - #1 #1 #1 #1 #5 #5 #5
20 - #0 #0 #0 #0 #4 #4 #4 #4
11

图3

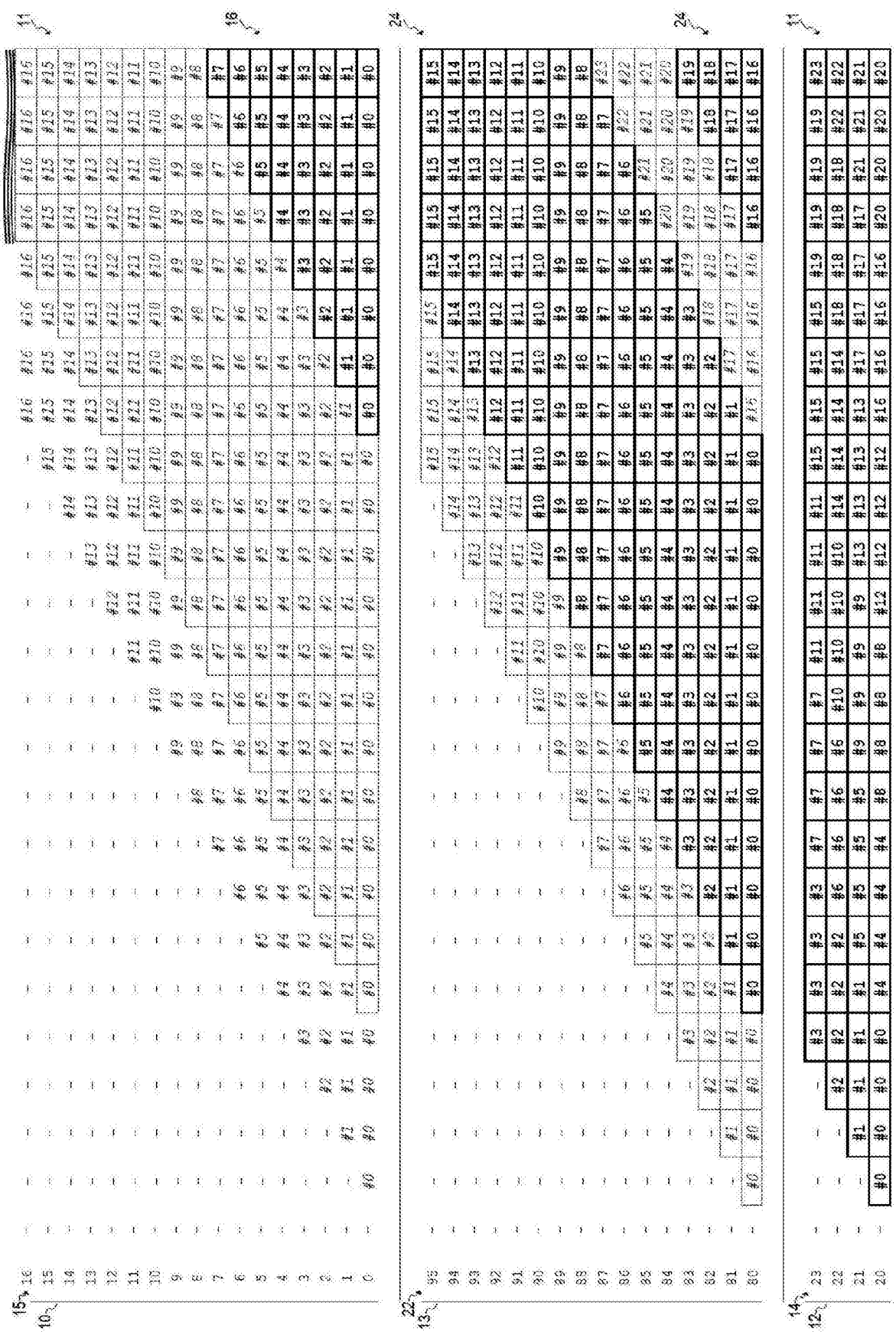

图4