



## [12] 发明专利说明书

专利号 ZL 03816690.9

[45] 授权公告日 2009 年 9 月 30 日

[11] 授权公告号 CN 100545863C

[22] 申请日 2003.4.23 [21] 申请号 03816690.9

[30] 优先权

[32] 2002.6.13 [33] US [31] 10/170,604

[86] 国际申请 PCT/US2003/012693 2003.4.23

[87] 国际公布 WO2003/107272 英 2003.12.24

[85] 进入国家阶段日期 2005.1.13

[73] 专利权人 模拟设备公司

地址 美国马萨诸塞州

[72] 发明人 宁 可 马尔卡·霍夫曼 加比·伊

[56] 参考文献

US5052046A 1991.9.24

审查员 汪 宁

[74] 专利代理机构 中原信达知识产权代理有限责任公司

代理人 钟 强 谷惠敏

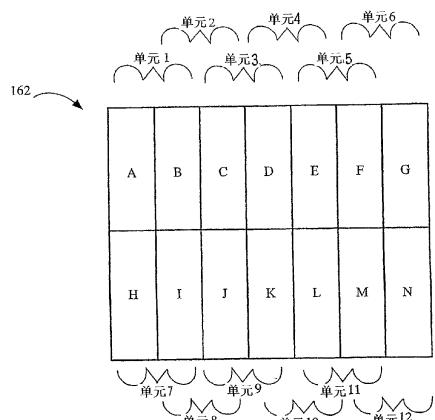
权利要求书 3 页 说明书 16 页 附图 6 页

## [54] 发明名称

用于使用图像条和循环寻址排列进行图像数据处理的方法和装置

## [57] 摘要

一种相对高速循环存储设备，其结合其他的过程，提高图像处理效率。为达到此目的，一种处理存储在初始存储器内的图像数据的方法和装置将图像逻辑分成多个相邻的条。第一多条存储在具有循环寻址布置的工作存储器内，其中该工作存储器快于初始存储器速度并且有多个顺序的地址位置。该第一多条是相邻的并具有起始地址。此外，该第一多条以相邻方式存储在工作存储器内，并通过工作存储器相对于起始地址进行处理。



1. 一种处理存储在初始存储器内的图像数据的方法，该方法包括：

将图像逻辑地划分成多个相邻的图像条；

将相邻图像条组从初始存储器传输到比该初始存储器快的工作存储器，该工作存储器具有在循环寻址布置中的多个顺序的地址位置，由此使得该相邻图像条组从起始地址相邻地存储在工作存储器内；以及

通过工作存储器根据起始地址处理该相邻的图像条组。

2. 如权利要求 1 所述的方法，其中将相邻的图像条组从初始存储器传输到工作存储器的步骤包括：将该图像条组中的不同图像条的相邻部分存储在工作存储器相邻的地址位置中。

3. 如权利要求 1 所述的方法，其中工作存储器有第一存储位置和最新存储位置，该第一存储位置逻辑上与该最新存储位置相邻。

4. 如权利要求 1 所述的方法，其中对于该相邻图像条组的处理使用了相对于基地址指针的偏移计算。

5. 如权利要求 1 的所述方法，进一步包括：当处理图像条组时，在工作存储器内存存储附加的图像条，该附加的图像条与该图像条组相邻。

6. 如权利要求 5 所述的方法，进一步包括：将来自该图像条组的至少一个图像条与该附加的图像条一起处理。

7. 如权利要求 1 所述的方法，其中工作存储器被逻辑地划分成两个或更多个组，所述组与图像条相对应，从而使得每个组是这样的存储器块，该存储器块被配置为具有与图像数据条相同的地址位置总数，并且其中传输该相邻的图像条组的步骤包括：基于工作存储器中的块数，将图像条内的行逻辑地移位预选的次数。

8. 一种用于处理存储在初始存储器内的图像数据的装置，该数据图像被逻辑划分成多个相邻的图像条，该装置包括：

    用于以循环方式访问工作存储器的地址管理器，工作存储器快于初始存储器并有多个顺序的地址位置；

    用来将相邻的图像条组从初始存储器传输到工作存储器的数据传输设备，从而使得该相邻的图像条组从起始地址相邻地存储在工作存储器内；以及

    处理器，用于通过工作存储器根据起始地址处理该相邻的图像条组。

9. 如权利要求 8 所述的装置，其中通过将该图像条组中的不同图像条的相邻部分存储在工作存储器的相邻地址位置中，从而该数据传输设备将该相邻的图像条组从初始存储器传输到工作存储器。

10. 如权利要求 8 所述的装置，其中工作存储器具有第一存储位置和最新存储位置，该第一存储位置逻辑上与该最新存储位置相邻。

11. 如权利要求 8 所述的装置，其中处理器通过使用相对于基地

址指针的偏移计算来对相邻图像组进行处理。

12. 如权利要求 8 所述的装置，其中当处理该图像条组时，该数据传输设备进一步在工作存储器中存储附加的图像条，该附加的图像条与该图像条组相邻。

13. 如权利要求 12 所述的装置，其中处理器进一步将来自该图像条组的至少一个图像条与该附加的图像条一起处理。

14. 如权利要求 8 所述的装置，其中该工作存储器被逻辑地划分成两个或更多个组，所述组与图像条相对应，从而使得每个组是这样的存储器块，该存储器块被配置为具有与图像数据条相同的地址位置总数，并且其中通过基于工作存储器中的块数逻辑地将图像条内的行移位预选的次数，从而数据传输设备进一步对相邻的图像条组进行传输。

用于使用图像条和循环寻址排列进行图像数据处理的方法和装置

## 技术领域

本发明涉及一种存储器管理，并且特别地，本发明涉及提高图形处理系统内的存储器效率。

## 背景技术

在现代计算机系统众多的优点当中，其之一就是其制造逼真的图表显示的能力。因此，计算机系统已成为多种图形领域中广泛应用的工具，如摄影和图形设计。然而，为此目的而增强的计算机应用具有提高了设计复杂程度并且相应地也提高了制造图像软件复杂程度的效果。

现有技术通过计算机系统增加更多处理能力来响应于该需求。例如，许多计算机系统有专用图形处理器，该专用图形处理器被专门设计用于减少主处理器上的计算负荷。因此，许多这种计算机具有超过其他计算机元件（如主存储器（即 RAM）或高速缓冲存储器）速度的处理速度。实际上，许多传统的个人计算机具有其独自超过主存储器和高速缓冲存储器速度的主处理器。相应地，尽管处理器的速度提高了，但主存储器和高速缓冲存储器会在计算机系统内制造处理瓶颈，不合需要地阻碍图像光栅化过程。

## 发明内容

根据本发明一方面，一种相对高速循环存储设备，其结合其他的过程，提高图像处理效率。为达到此目的，一种处理存储在初始存储器内的图像数据的方法和装置将图像逻辑分成多个相邻的条。第一多条存储在具有循环寻址布置的工作存储器内，其中该工作存储器快于初始存储器速度并且有多个顺序的地址位置。该第一多条是相邻的，

并具有起始地址。此外，该第一多条以相邻方式存储在工作存储器内，并通过工作存储器相对于起始地址进行处理。

在一些实施例中，当不同条的相邻部分（在多条内）存储在工作存储器的相邻地址位置时，该第一多条被认为是以相邻方式存储的。为维持循环性，工作存储器可具有第一存储位置和最新存储位置，其中第一存储位置逻辑上与最新存储位置相邻。而且，在处理过程中，可计算从起始地址开始的偏移量。

在其他一些实施例中，当该第一多条被处理时新的条被存储在工作存储器内。该新的条与第一多条相邻。因此，来自该第一多条的至少一个条与新的条一起处理。工作存储器可划分为给定数量的条区域，并且每个条中的图像数据包括多个图像数据行。在这些实施例中，根据条区域的给定数量，可通过移位用于存储在工作存储器内的多个行来存储条。

根据本发明的其他方面，一种用于处理存储在被逻辑地分成多个相邻条的初始存储器内的图像数据的装置包括用于以循环方式访问工作存储器的地址管理器。工作存储器快于初始存储器并有多个顺序的地址位置。该装置还包括数据传输设备，其将第一多条存储于工作存储器，其中该第一多条是相邻的并有起始地址。该第一多条以相邻方式存储在工作存储器内。该装置还包括处理器，用于通过工作存储器相对于起始地址处理第一多条。

本发明的说明性实施例实现为计算机程序产品，该产品具有在其上带有计算机可读程序代码的计算机可用介质。根据传统的过程，计算机系统可以读取和使用该计算机可读编码。

#### 附图说明

通过下面结合附图进行更详细的说明，将会更加全面地理解本发

明的上述内容和优势，在附图中：

图 1 示意性地示出了能实现说明性实施例的示例计算机系统。

图 2 示意性地示出了根据本发明说明性实施例可进行处理的图形图像帧。

图 3 显示可用来处理图像的两个存储设备

图 4A 显示图 3 中所示的第二存储器（高速缓冲存储器）配置的附加细节。

图 4B 是图 4A 中所示的第二存储器另一个逻辑视图。

图 5 显示图 2 中所示的处理图像的说明性方法。

图 6 显示图 3、4A 和 4B 所示的跨越通过第二存储器的说明性方法。

### 具体实施方式

在本发明的说明性实施例中，计算机系统内的图像数据从较慢存储器向较快、循环配置的存储器传输以进行处理。下面将讨论示例性实施例的细节。

图 1 是在其上能实现说明性实施例的示例计算机系统的示意图。图 1 中的示例计算机系统 100 仅用于说明的目的，不应被认为是对本发明的限制。尽管下面的说明会参考常用来描述特定计算机系统的术语，但所描述的概念同样可应用于其他的计算机系统，包括具有与图 1 所示结构不相似构造的系统。

计算机 100 包括：具有传统微处理器的中央处理单元 (CPU) 105，用于暂时存储信息的主存储器 110（例如随机访问存储器，本领域通常被叫做“RAM”），和用于永久存储只读信息的只读存储器 (ROM) 115。此外，计算机 100 还具有用于控制系统 RAM110 的存储器控制器 120，和用于控制至系统 RAM110 直接存储器访问的直接存储器访问 (DMA) 控制器 160。计算机 100 还包括用于联接多种内部元件的主总线 130。

可由众所周知的非易失性存储介质提供大容量存储，例如磁盘 142，数字通用磁盘（在本领域通常叫做“DVD”，图中未显示），CD-ROM147，和硬盘 152。而且，数据和软件可通过诸如磁盘 142 和 CD-ROM147 的可移除介质或通过网络连接与计算机系统 100 进行交换。

至计算机 100 的用户输入可由若干设备提供。例如，键盘 156 和鼠标 157 可通过键盘和鼠标控制器 155 连接至总线 130。对于本领域的技术人员来说，其他输入设备如数码相机、笔和/或写字板以及语音输入的麦克风，可通过总线 130 和合适的控制器连接至计算机 100，这是显而易见的。

计算机系统 100 优选地由诸如 WINDOWS NT 操作系统（由 Microsoft Corp., of Redmond, Washington 发行）的操作系统软件控制和协调。在其他计算机系统控制功能中，操作系统控制系统资源的分配，并执行系统任务，如过程调度、存储器管理、联网以及 I/O 业务。除此之外，说明性实施例（其与存储器管理相关）可实现为为操作系统的一部分或作为独立的程序。

图 2 是根据本发明的说明性实施例可进行处理的图形图像 162 的帧示意图。图形图像 162 可以是从诸如数码相机或视频输入的任何已知源产生的数字化图像。例如，图像 162 可要求一些处理，如符合传统的 MPEG 标准的压缩操作的应用。

为达到上述目的，图形图像 162 被逻辑分成多个相邻的数据条（以下简称“条”），这些条在逻辑上被视为形成多个相邻的数据单元（以下简称“单元”）。每个条由像素的二维阵列组成，该阵列构成图像 162 的其各自部分。在图 2 所示的例子中，图像 162 被分成七条的两行。该每条的行形成六个单元。这些条被标识成条 A-N，而这些

单元被标识成单元 1-12。一个单元由两个相邻的条组成。例如，条 A 和 B 形成单元 1。下表表示图 2 中单元和其相应的条：

单元	条
1	A,B
2	B,C
3	C,D
4	D,E
5	E,F
6	F,G
7	H,I
8	I,J
9	J,K
10	K,L
11	L,M
12	M,N

单元形成于每行中，从而每个中间的单元（即每行中既不是第一个单元也不是最后一个单元）与前一个和其后的单元重叠。例如，单元 2 由条 B 和条 C 形成，而单元 3 由条 C 和条 D 形成。因此，单元 2 和单元 3 共享条 C。同样，单元 1 和单元 2 共享条 B。如下面所进行的更详细的描述，该重叠通过在计算机 100 内部存储器系统内允许流水线操作，促进了并行处理。对本领域的技术人员来说，条的逻辑分配的优势，如提高处理速度，是显而易见的。

不过应当注意到，图 2 中所示的逻辑条和单元仅作为示例，并非建议只有该数量的条和单元才足够。在一些实施例中，例如，单元可以由三个条或一个条组成。相应地，对于特定的条和单元的大小/数量的阐述并不限制本发明各种实施例的范围。

图 2 所示的图像 162 的条和单元说明性地在图像 162 在快速存储

器内被处理之前逻辑地形成。图 3 显示两个可用来处理这种逻辑划分的图像 162 的存储设备。在高级，图像 162 最初被存储在初始存储设备中（初始存储器 164），然后以逐条方式传输至工作存储设备（工作存储器 166）。工作存储器 166 具有快于初始存储器 164 的速度的速度。当图像 162 在工作存储器 166 中时，CPU 处理该图像 162。

更为具体地，参考图 3，初始存储器 164 可以是外部 SDRAM（同步动态随机访问存储器），其储存全部图像 162。DMA 控制器将图像 162 的条传输至工作存储器 166，在本例中，该工作存储器 166 是内部高速缓冲存储器。下面参照图 5 和图 6 对传输和处理图像 162 的方法作更详细的说明。不过应当注意到，尽管已经讨论了 SDRAM 和高速缓冲存储器，但也可以使用能达到此相对速度的其他类型的存储器。相应地，在许多实施例中，任何将数据传输给较快存储器用于处理的较慢存储器应是足够的。

图 4A 显示图 3 中所示的第二存储器（高速缓冲存储器）配置的附加细节。工作存储器 166 配置为具有地址位置（也叫做“存储位置”）集编号，该每个地址位置存储用于图像 162 内一个像素的数据。例如，图 4A 所示的工作存储器 166 具有用于 96 个像素数据的地址位置。每个地址位置的大小适当，以用来存储从初始存储器 164 接收到的未处理的图像数据。相应地，每个地址位置应有足够的空间存储色彩（如红、绿和蓝）、透明度、深度以及其他给定像素所需的数据。

在处理过程中，每个地址位置可被连续地跨越。例如，如果要读取的第一个地址位置为 0，则要读取的第二个地址位置为 1，并且其后读取的地址位置为 2 等等…。在替代的实施例中，地址位置不是被连续处理。但是，在上述任一个实施例中，选择一个基地址（下面将进行讨论，通常叫做“起始地址”），并且处理是相应于该基地址进行。更具体地，处理是以相对于该基地址的偏移来进行。例如，如果基地址是地址位置 4，然后到处理地址位置 6-8，则逻辑规定处理距离

基地址的地址位置偏移 2-4。

根据说明性实施例，工作存储器 166 配置为循环存储器。相应地，当地址位置 95 被读取、跳过或另外被跨越之后，处理回到地址位置 0。而且，工作存储器 166 被逻辑分成两个或更多的对应于图像 162 中的条的组。更具体地，如图 4A 所示的工作存储器 166 被划分成三个相等大小的块。这些块以下简称为第一块 166A，第二块 166B 和第三块 166C。每个存储块配置为具有与图像数据条的地址位置相同总数量的地址位置。在说明性实施例中，每个存储块因此被配置成具有与图像 162 的条完全相同的二维大小。

图 4B 是图 4A 所示的工作存储器 166 的另一个逻辑视图。具体地说，对于指针或其他存储器跨越装置，工作存储器 166 出现作为地址位置的一维阵列。在所示的例子中，当跨越地址位置 95 之后，就读取/跳过或跨越地址位置 0。然而相邻的图像数据存储在该地址位置内，如图 4A 所示。

按照与本发明其他要素类似的方式，图 4A 和图 4B 所示的地址位置的总数量仅作示例。可相应地使用不同数量的地址位置和分区，并可使用不同大小的存储器。对于特定大小的讨论并不限制本发明的范围。

图 5 是图 2 所示的处理图像 162 的方法。该方法从步骤 500 开始，其中，图像 162 被逻辑划分成多个条。例如，可按图 2 所示划分图像 162。根据图像 162 的大小，该图像 162 可被划分成一行或更多行的条，如上所述，每行形成多个重叠单元。对于在图像 162 内的每行执行剩余步骤。

具体地，然后将相邻条的第一多集从初始存储器 164（即较慢存储器，如 2 级存储器）传输至工作存储器 166（即较快存储器，如 1

级存储器)。在说明性实施例中, 条 A 和条 B (即单元 1) 可首先装载进工作存储器 166 的第一和第二块 166A 和 166B。如图所示, 这些条被连续地装载进工作存储器 166。更具体地, 在两个相邻条之间的相邻像素被存储在工作存储器 166 内相邻的地址位置中。在所示的例子中, 该两个相邻的条在相同的行内有相同的像素。

为实现此目的, DMA 控制器被配置成 2D-DMA。除此之外, 它说明性地执行两个操作: 即 1) “装载” 操作, 用于将单元 (即条集) 装载到 FIFO (先入先出) 管上; 以及 2) “存储” 操作, 用于将图像数据阵列从 FIFO 管作为像素二维阵列 (即作为条或单元) 存储在工作存储器 166 中。注意: 如上所述, 每个条是形成图像 162 一部分的像素数据的二维阵列。相应地, 这些操作确保当被存储到初始存储器 164 内时维持该条数据格式。

在说明性实施例中, 用于两个所述 DMA 操作的软件语义如下:

```
// The DMA_LOAD semantics:
movtofifo (short *in, int xc, int xs, int ye, int ys)
{
    while (yc) {
        for (x=0;x<xc;x++) {
            *fifo_write++ = *in;
            in += xs;
        }
        in += ys;
        yc--;
    }
}
// The DMA_STORE semantics;
movfromfifo (short *out, int xc, int xs, int yc, int ys)
{
    while (yc) {
        for (x=0;x<xc;x++) {
            *out = *fifo_read++;
            out += xs;
        }
        out += ys;
        yc--;
    }
}
```

```

    }

//Parameters:
xs - the x stride
xc - the x count
ys - the y stride
yc - the y count
sz - the element size 8bit, 16bit, 32bit

```

当第一条集装载之后，然后在步骤 504 设置一对指针。更具体地，该系统配置成具有指向给定单元第一地址位置（在工作存储器 166 内）的基指针 和指向相对于该基指针的偏移的地址位置（也在工作存储器 166 内）的读取指针。最初，该两个指针都指向相同的地址位置。例如，该指针最初设置成指向条 A 内的第一地址位置。如图 6 所示（如下所述），每当处理一个新的单元，基指针就递增地址位置集编号，而当每个单元被跨越后读取指针在该单元内连续递增一次（用于每个地址位置）。

相应地，该方法进行到步骤 506，其中，根据如下参考图 6 所述的方法，处理当前的条集（即，当前被处理的单元）。然后在步骤 508 确定被处理的行是否有附加的条用于处理。如果该行的附加条不需要处理，则过程结束（针对该行）。

相反，如果附加的条需要处理，则过程继续到步骤 510，其中，下一个条集存储在工作存储器 166 内。在图 2 所示的例子中，单个数据条（如条 C）存储在工作存储器 166 的下一个随后块，在此情形下，该下一个随后块就是工作存储器 166 的第三块 166C。过程循环回到步骤 504，指针被重新设置。为达到此目的，基指针和读取指针设置成指向下一个单元的第一地址位置（如条 B 的第一存储位置）。相应地，处理下一个单元（由条 B 和条 C 组成），因此允许重新使用数据（即条 B）。迭代此方法，直到整行被处理。如果图像 162 有附加行，则对下一行重复该方法。

在说明性实施例中，图 5 所示方法的多个步骤可以不同的顺序或基本同时进行。例如，处理当前条集时（步骤 506），CPU105 可确定是否要处理附加的条（步骤 508），并且如果是，它将这些条存储在工作存储器 166 的下一个块中（步骤 510）。相应地，一些实施例将这些步骤组合进一个基本并行的操作中。

图 6 显示了步骤 506 所引用的处理当前条集的方法。该方法从步骤 600 开始，其中，基指针和读取指针被定位。该步骤可以隐含在过程中，因而它在多种实施例中不是必需的步骤。该过程然后继续到步骤 602，其中，读取地址位置集用于处理。为达到此目的，读取指针递增，以从基指针读取地址位置集编号。在本文讨论的示例中，读取指针递增八次，以读取在其当前单元内的第一八个地址位置中的每一个。相应地，该步骤允许读取具有单元数据的工作存储器 166 的两个相邻块的一部分。

当地址位置集被读取后，然后读取指针跳过下一个随后的地址位置集（步骤 604）。在所示的例子中，读取指针跳过随后的四个地址位置。该步骤允许该方法跳过没有单元数据的工作存储器 166 的那一个块。然后在步骤 606 确定读取指针是否指向与由基指针所指向的地址相同的地址(即基地址)。如果是，则该方法结束，因为整个单元已经存储在工作存储器 166 内。相反，如果读取指针指向不同的地址位置，则过程循环回到步骤 602，其中，读取下一个地址位置集编号。相应地，迭代此方法直到整个单元存储在工作存储器 166 内。如上所述，在读取地址位置后，CPU105 会执行一些处理功能，如压缩图像数据。

相应地，图 5 和图 6 中所描述的方法允许将像素数据的二维阵列存储在工作存储器 166 内。从基指针读取/存储过程开始和配置工作存储器作为循环存储器，使得该方法能有效地完成。

下面例子说明图 4A 所示的工作存储器 166 内的所述方法的九个迭代。更具体地，该例子从图像处理的九个迭代来显示工作存储器 166 的内容。图像 162 像素可按行顺序地寻址。换句话说，每行有 64 个可顺序寻址的像素位置。该例子的参数如下：

图像大小：64x64

条大小：4x8

条数：每行 16 条，8 个条行，总共 128 条

单元大小：8x8(2 条)

循环寻址大小：3\*4\*8

在下面的例子中，图像 162 内的每个像素以下列格式标识：“a（行，列）”。因此，如标记“a（0，1）”表示用于位于图像 162 的 0 行和列 1 中的像素的像素数据。此外，工作存储器 166 第一列内的每个存储位置的地址编号在紧挨工作存储器 166 第一列的左端列出。当然，参照图 4A，可确定所有在示例工作存储器 166 内的地址。最后，每个迭代的基地址大写并加下划线，而接收图像数据的存储部分用加粗。

模拟输出：

迭代（1），基地址=0。

0000:	<b>a[ 0, 0]</b>	a[ 0, 1]	a[ 0, 2]	a[ 0, 3]	a[ 0, 4]	a[ 0, 5]	a[ 0, 6]	a[ 0, 7]	N/A	N/A	N/A	N/A
0012:	<b>a[ 1, 0]</b>	a[ 1, 1]	a[ 1, 2]	a[ 1, 3]	a[ 1, 4]	a[ 1, 5]	a[ 1, 6]	a[ 1, 7]	N/A	N/A	N/A	N/A
0024:	<b>a[ 2, 0]</b>	a[ 2, 1]	a[ 2, 2]	a[ 2, 3]	a[ 2, 4]	a[ 2, 5]	a[ 2, 6]	a[ 2, 7]	N/A	N/A	N/A	N/A
0036:	<b>a[ 3, 0]</b>	a[ 3, 1]	a[ 3, 2]	a[ 3, 3]	a[ 3, 4]	a[ 3, 5]	a[ 3, 6]	a[ 3, 7]	N/A	N/A	N/A	N/A
0048:	<b>a[ 4, 0]</b>	a[ 4, 1]	a[ 4, 2]	a[ 4, 3]	a[ 4, 4]	a[ 4, 5]	a[ 4, 6]	a[ 4, 7]	N/A	N/A	N/A	N/A
0060:	<b>a[ 5, 0]</b>	a[ 5, 1]	a[ 5, 2]	a[ 5, 3]	a[ 5, 4]	a[ 5, 5]	a[ 5, 6]	a[ 5, 7]	N/A	N/A	N/A	N/A
0072:	<b>a[ 6, 0]</b>	a[ 6, 1]	a[ 6, 2]	a[ 6, 3]	a[ 6, 4]	a[ 6, 5]	a[ 6, 6]	a[ 6, 7]	N/A	N/A	N/A	N/A
0084:	<b>a[ 7, 0]</b>	a[ 7, 1]	a[ 7, 2]	a[ 7, 3]	a[ 7, 4]	a[ 7, 5]	a[ 7, 6]	a[ 7, 7]	N/A	N/A	N/A	N/A

迭代（2），基地址=4。

0000:	a[ 0, 0]	a[ 0, 1]	a[ 0, 2]	a[ 0, 3]	A[ 0, 4]	a[ 0, 5]	a[ 0, 6]	a[ 0, 7]	a[ 0, 8]	a[ 0, 9]	a[ 0,10]	a[ 0,11]
0012:	a[ 1, 0]	a[ 1, 1]	a[ 1, 2]	a[ 1, 3]	a[ 1, 4]	a[ 1, 5]	a[ 1, 6]	a[ 1, 7]	a[ 1, 8]	a[ 1, 9]	a[ 1,10]	a[ 1,11]
0024:	a[ 2, 0]	a[ 2, 1]	a[ 2, 2]	a[ 2, 3]	a[ 2, 4]	a[ 2, 5]	a[ 2, 6]	a[ 2, 7]	a[ 2, 8]	a[ 2, 9]	a[ 2,10]	a[ 2,11]
0036:	a[ 3, 0]	a[ 3, 1]	a[ 3, 2]	a[ 3, 3]	a[ 3, 4]	a[ 3, 5]	a[ 3, 6]	a[ 3, 7]	a[ 3, 8]	a[ 3, 9]	a[ 3,10]	a[ 3,11]
0048:	a[ 4, 0]	a[ 4, 1]	a[ 4, 2]	a[ 4, 3]	a[ 4, 4]	a[ 4, 5]	a[ 4, 6]	a[ 4, 7]	a[ 4, 8]	a[ 4, 9]	a[ 4,10]	a[ 4,11]
0060:	a[ 5, 0]	a[ 5, 1]	a[ 5, 2]	a[ 5, 3]	a[ 5, 4]	a[ 5, 5]	a[ 5, 6]	a[ 5, 7]	a[ 5, 8]	a[ 5, 9]	a[ 5,10]	a[ 5,11]
0072:	a[ 6, 0]	a[ 6, 1]	a[ 6, 2]	a[ 6, 3]	a[ 6, 4]	a[ 6, 5]	a[ 6, 6]	a[ 6, 7]	a[ 6, 8]	a[ 6, 9]	a[ 6,10]	a[ 6,11]
0084:	a[ 7, 0]	a[ 7, 1]	a[ 7, 2]	a[ 7, 3]	a[ 7, 4]	a[ 7, 5]	a[ 7, 6]	a[ 7, 7]	a[ 7, 8]	a[ 7, 9]	a[ 7,10]	a[ 7,11]

迭代 (3) , 基地址=8。

0000:	a[ 7,12]	a[ 7,13]	a[ 7,14]	a[ 7,15]	a[ 0, 4]	a[ 0, 5]	a[ 0, 6]	a[ 0, 7]	A[ 0, 8]	a[ 0, 9]	a[ 0,10]	a[ 0,11]
0012:	a[ 0,12]	a[ 0,13]	a[ 0,14]	a[ 0,15]	a[ 1, 4]	a[ 1, 5]	a[ 1, 6]	a[ 1, 7]	a[ 1, 8]	a[ 1, 9]	a[ 1,10]	a[ 1,11]
0024:	a[ 1,12]	a[ 1,13]	a[ 1,14]	a[ 1,15]	a[ 2, 4]	a[ 2, 5]	a[ 2, 6]	a[ 2, 7]	a[ 2, 8]	a[ 2, 9]	a[ 2,10]	a[ 2,11]
0036:	a[ 2,12]	a[ 2,13]	a[ 2,14]	a[ 2,15]	a[ 3, 4]	a[ 3, 5]	a[ 3, 6]	a[ 3, 7]	a[ 3, 8]	a[ 3, 9]	a[ 3,10]	a[ 3,11]
0048:	a[ 3,12]	a[ 3,13]	a[ 3,14]	a[ 3,15]	a[ 4, 4]	a[ 4, 5]	a[ 4, 6]	a[ 4, 7]	a[ 4, 8]	a[ 4, 9]	a[ 4,10]	a[ 4,11]
0060:	a[ 4,12]	a[ 4,13]	a[ 4,14]	a[ 4,15]	a[ 5, 4]	a[ 5, 5]	a[ 5, 6]	a[ 5, 7]	a[ 5, 8]	a[ 5, 9]	a[ 5,10]	a[ 5,11]
0072:	a[ 5,12]	a[ 5,13]	a[ 5,14]	a[ 5,15]	a[ 6, 4]	a[ 6, 5]	a[ 6, 6]	a[ 6, 7]	a[ 6, 8]	a[ 6, 9]	a[ 6,10]	a[ 6,11]
0084:	a[ 6,12]	a[ 6,13]	a[ 6,14]	a[ 6,15]	a[ 7, 4]	a[ 7, 5]	a[ 7, 6]	a[ 7, 7]	a[ 7, 8]	a[ 7, 9]	a[ 7,10]	a[ 7,11]

迭代 (4) , 基地址=12。

0000:	a[ 7,12]	a[ 7,13]	a[ 7,14]	a[ 7,15]	a[ 7,16]	a[ 7,17]	a[ 7,18]	a[ 7,19]	a[ 0, 8]	a[ 0, 9]	a[ 0,10]	a[ 0,11]
0012:	A[ 0,12]	a[ 0,13]	a[ 0,14]	a[ 0,15]	a[ 0,16]	a[ 0,17]	a[ 0,18]	a[ 0,19]	a[ 1, 8]	a[ 1, 9]	a[ 1,10]	a[ 1,11]
0024:	a[ 1,12]	a[ 1,13]	a[ 1,14]	a[ 1,15]	a[ 1,16]	a[ 1,17]	a[ 1,18]	a[ 1,19]	a[ 2, 8]	a[ 2, 9]	a[ 2,10]	a[ 2,11]
0036:	a[ 2,12]	a[ 2,13]	a[ 2,14]	a[ 2,15]	a[ 2,16]	a[ 2,17]	a[ 2,18]	a[ 2,19]	a[ 3, 8]	a[ 3, 9]	a[ 3,10]	a[ 3,11]
0048:	a[ 3,12]	a[ 3,13]	a[ 3,14]	a[ 3,15]	a[ 3,16]	a[ 3,17]	a[ 3,18]	a[ 3,19]	a[ 4, 8]	a[ 4, 9]	a[ 4,10]	a[ 4,11]
0060:	a[ 4,12]	a[ 4,13]	a[ 4,14]	a[ 4,15]	a[ 4,16]	a[ 4,17]	a[ 4,18]	a[ 4,19]	a[ 5, 8]	a[ 5, 9]	a[ 5,10]	a[ 5,11]
0072:	a[ 5,12]	a[ 5,13]	a[ 5,14]	a[ 5,15]	a[ 5,16]	a[ 5,17]	a[ 5,18]	a[ 5,19]	a[ 6, 8]	a[ 6, 9]	a[ 6,10]	a[ 6,11]
0084:	a[ 6,12]	a[ 6,13]	a[ 6,14]	a[ 6,15]	a[ 6,16]	a[ 6,17]	a[ 6,18]	a[ 6,19]	a[ 7, 8]	a[ 7, 9]	a[ 7,10]	a[ 7,11]

迭代 (5) , 基地址=16。

0000:	a[ 7,12]	a[ 7,13]	a[ 7,14]	a[ 7,15]	a[ 7,16]	a[ 7,17]	a[ 7,18]	a[ 7,19]	a[ 7,20]	a[ 7,21]	a[ 7,22]	a[ 7,23]
0012:	a[ 0,12]	a[ 0,13]	a[ 0,14]	a[ 0,15]	A[ 0,16]	a[ 0,17]	a[ 0,18]	a[ 0,19]	a[ 0,20]	a[ 0,21]	a[ 0,22]	a[ 0,23]
0024:	a[ 1,12]	a[ 1,13]	a[ 1,14]	a[ 1,15]	a[ 1,16]	a[ 1,17]	a[ 1,18]	a[ 1,19]	a[ 1,20]	a[ 1,21]	a[ 1,22]	a[ 1,23]
0036:	a[ 2,12]	a[ 2,13]	a[ 2,14]	a[ 2,15]	a[ 2,16]	a[ 2,17]	a[ 2,18]	a[ 2,19]	a[ 2,20]	a[ 2,21]	a[ 2,22]	a[ 2,23]
0048:	a[ 3,12]	a[ 3,13]	a[ 3,14]	a[ 3,15]	a[ 3,16]	a[ 3,17]	a[ 3,18]	a[ 3,19]	a[ 3,20]	a[ 3,21]	a[ 3,22]	a[ 3,23]
0060:	a[ 4,12]	a[ 4,13]	a[ 4,14]	a[ 4,15]	a[ 4,16]	a[ 4,17]	a[ 4,18]	a[ 4,19]	a[ 4,20]	a[ 4,21]	a[ 4,22]	a[ 4,23]
0072:	a[ 5,12]	a[ 5,13]	a[ 5,14]	a[ 5,15]	a[ 5,16]	a[ 5,17]	a[ 5,18]	a[ 5,19]	a[ 5,20]	a[ 5,21]	a[ 5,22]	a[ 5,23]
0084:	a[ 6,12]	a[ 6,13]	a[ 6,14]	a[ 6,15]	a[ 6,16]	a[ 6,17]	a[ 6,18]	a[ 6,19]	a[ 6,20]	a[ 6,21]	a[ 6,22]	a[ 6,23]

迭代 (6) , 基地址=20。

0000:	a[ 6,24]	a[ 6,25]	a[ 6,26]	a[ 6,27]	a[ 7,16]	a[ 7,17]	a[ 7,18]	a[ 7,19]	a[ 7,20]	a[ 7,21]	a[ 7,22]	a[ 7,23]
0012:	a[ 7,24]	a[ 7,25]	a[ 7,26]	a[ 7,27]	a[ 0,16]	a[ 0,17]	a[ 0,18]	a[ 0,19]	A[ 0,20]	a[ 0,21]	a[ 0,22]	a[ 0,23]
0024:	a[ 0,24]	a[ 0,25]	a[ 0,26]	a[ 0,27]	a[ 1,16]	a[ 1,17]	a[ 1,18]	a[ 1,19]	a[ 1,20]	a[ 1,21]	a[ 1,22]	a[ 1,23]
0036:	a[ 1,24]	a[ 1,25]	a[ 1,26]	a[ 1,27]	a[ 2,16]	a[ 2,17]	a[ 2,18]	a[ 2,19]	a[ 2,20]	a[ 2,21]	a[ 2,22]	a[ 2,23]
0048:	a[ 2,24]	a[ 2,25]	a[ 2,26]	a[ 2,27]	a[ 3,16]	a[ 3,17]	a[ 3,18]	a[ 3,19]	a[ 3,20]	a[ 3,21]	a[ 3,22]	a[ 3,23]
0060:	a[ 3,24]	a[ 3,25]	a[ 3,26]	a[ 3,27]	a[ 4,16]	a[ 4,17]	a[ 4,18]	a[ 4,19]	a[ 4,20]	a[ 4,21]	a[ 4,22]	a[ 4,23]
0072:	a[ 4,24]	a[ 4,25]	a[ 4,26]	a[ 4,27]	a[ 5,16]	a[ 5,17]	a[ 5,18]	a[ 5,19]	a[ 5,20]	a[ 5,21]	a[ 5,22]	a[ 5,23]
0084:	a[ 5,24]	a[ 5,25]	a[ 5,26]	a[ 5,27]	a[ 6,16]	a[ 6,17]	a[ 6,18]	a[ 6,19]	a[ 6,20]	a[ 6,21]	a[ 6,22]	a[ 6,23]

迭代 (7) , 基地址=24。

0000:	a[ 6,24]	a[ 6,25]	a[ 6,26]	a[ 6,27]	a[ 6,28]	a[ 6,29]	a[ 6,30]	a[ 6,31]	a[ 7,20]	a[ 7,21]	a[ 7,22]	a[ 7,23]
0012:	a[ 7,24]	a[ 7,25]	a[ 7,26]	a[ 7,27]	a[ 7,28]	a[ 7,29]	a[ 7,30]	a[ 7,31]	a[ 0,20]	a[ 0,21]	a[ 0,22]	a[ 0,23]
0024:	A[ 0,24]	a[ 0,25]	a[ 0,26]	a[ 0,27]	a[ 0,28]	a[ 0,29]	a[ 0,30]	a[ 0,31]	a[ 1,20]	a[ 1,21]	a[ 1,22]	a[ 1,23]
0036:	a[ 1,24]	a[ 1,25]	a[ 1,26]	a[ 1,27]	a[ 1,28]	a[ 1,29]	a[ 1,30]	a[ 1,31]	a[ 2,20]	a[ 2,21]	a[ 2,22]	a[ 2,23]
0048:	a[ 2,24]	a[ 2,25]	a[ 2,26]	a[ 2,27]	a[ 2,28]	a[ 2,29]	a[ 2,30]	a[ 2,31]	a[ 3,20]	a[ 3,21]	a[ 3,22]	a[ 3,23]
0060:	a[ 3,24]	a[ 3,25]	a[ 3,26]	a[ 3,27]	a[ 3,28]	a[ 3,29]	a[ 3,30]	a[ 3,31]	a[ 4,20]	a[ 4,21]	a[ 4,22]	a[ 4,23]
0072:	a[ 4,24]	a[ 4,25]	a[ 4,26]	a[ 4,27]	a[ 4,28]	a[ 4,29]	a[ 4,30]	a[ 4,31]	a[ 5,20]	a[ 5,21]	a[ 5,22]	a[ 5,23]
0084:	a[ 5,24]	a[ 5,25]	a[ 5,26]	a[ 5,27]	a[ 5,28]	a[ 5,29]	a[ 5,30]	a[ 5,31]	a[ 6,20]	a[ 6,21]	a[ 6,22]	a[ 6,23]

迭代 (8) , 基地址=28。

0000:	a[ 6,24]	a[ 6,25]	a[ 6,26]	a[ 6,27]	a[ 6,28]	a[ 6,29]	a[ 6,30]	a[ 6,31]	a[ 6,32]	a[ 6,33]	a[ 6,34]	a[ 6,35]
0012:	a[ 7,24]	a[ 7,25]	a[ 7,26]	a[ 7,27]	a[ 7,28]	a[ 7,29]	a[ 7,30]	a[ 7,31]	a[ 7,32]	a[ 7,33]	a[ 7,34]	a[ 7,35]
0024:	a[ 0,24]	a[ 0,25]	a[ 0,26]	a[ 0,27]	A[ 0,28]	a[ 0,29]	a[ 0,30]	a[ 0,31]	a[ 0,32]	a[ 0,33]	a[ 0,34]	a[ 0,35]
0036:	a[ 1,24]	a[ 1,25]	a[ 1,26]	a[ 1,27]	a[ 1,28]	a[ 1,29]	a[ 1,30]	a[ 1,31]	a[ 1,32]	a[ 1,33]	a[ 1,34]	a[ 1,35]
0048:	a[ 2,24]	a[ 2,25]	a[ 2,26]	a[ 2,27]	a[ 2,28]	a[ 2,29]	a[ 2,30]	a[ 2,31]	a[ 2,32]	a[ 2,33]	a[ 2,34]	a[ 2,35]
0060:	a[ 3,24]	a[ 3,25]	a[ 3,26]	a[ 3,27]	a[ 3,28]	a[ 3,29]	a[ 3,30]	a[ 3,31]	a[ 3,32]	a[ 3,33]	a[ 3,34]	a[ 3,35]
0072:	a[ 4,24]	a[ 4,25]	a[ 4,26]	a[ 4,27]	a[ 4,28]	a[ 4,29]	a[ 4,30]	a[ 4,31]	a[ 4,32]	a[ 4,33]	a[ 4,34]	a[ 4,35]
0084:	a[ 5,24]	a[ 5,25]	a[ 5,26]	a[ 5,27]	a[ 5,28]	a[ 5,29]	a[ 5,30]	a[ 5,31]	a[ 5,32]	a[ 5,33]	a[ 5,34]	a[ 5,35]

迭代 (9) , 基地址=32。

0000:	a[ 5,36]	a[ 5,37]	a[ 5,38]	a[ 5,39]	a[ 6,28]	a[ 6,29]	a[ 6,30]	a[ 6,31]	a[ 6,32]	a[ 6,33]	a[ 6,34]	a[ 6,35]
0012:	a[ 6,36]	a[ 6,37]	a[ 6,38]	a[ 6,39]	a[ 7,28]	a[ 7,29]	a[ 7,30]	a[ 7,31]	a[ 7,32]	a[ 7,33]	a[ 7,34]	a[ 7,35]
0024:	a[ 7,36]	a[ 7,37]	a[ 7,38]	a[ 7,39]	a[ 0,28]	a[ 0,29]	a[ 0,30]	a[ 0,31]	A[ 0,32]	a[ 0,33]	a[ 0,34]	a[ 0,35]
0036:	a[ 0,36]	a[ 0,37]	a[ 0,38]	a[ 0,39]	a[ 1,28]	a[ 1,29]	a[ 1,30]	a[ 1,31]	a[ 1,32]	a[ 1,33]	a[ 1,34]	a[ 1,35]
0048:	a[ 1,36]	a[ 1,37]	a[ 1,38]	a[ 1,39]	a[ 2,28]	a[ 2,29]	a[ 2,30]	a[ 2,31]	a[ 2,32]	a[ 2,33]	a[ 2,34]	a[ 2,35]
0060:	a[ 2,36]	a[ 2,37]	a[ 2,38]	a[ 2,39]	a[ 3,28]	a[ 3,29]	a[ 3,30]	a[ 3,31]	a[ 3,32]	a[ 3,33]	a[ 3,34]	a[ 3,35]
0072:	a[ 3,36]	a[ 3,37]	a[ 3,38]	a[ 3,39]	a[ 4,28]	a[ 4,29]	a[ 4,30]	a[ 4,31]	a[ 4,32]	a[ 4,33]	a[ 4,34]	a[ 4,35]
0084:	a[ 4,36]	a[ 4,37]	a[ 4,38]	a[ 4,39]	a[ 5,28]	a[ 5,29]	a[ 5,30]	a[ 5,31]	a[ 5,32]	a[ 5,33]	a[ 5,34]	a[ 5,35]

如上所示，该过程以及相关过程有这样一种效果：以预选的次数在条内逻辑移位行。具体地，该移位是根据工作存储器 166 内的块的数量而执行的。在上述例子中，当存储第一三个条集后，在被存储在工作存储器中时，第二条集向下移位一行。当下面三个随后的条集存储在工作存储器 166 中时，然后它们往下移位两行。通过每三个集一行的方式（在所讨论的例子中）迭代此过程，直至在图像的行中的所有单元被处理。

本领域的技术人员应该明白根据单元行对说明性实施例进行讨论仅作范例。也可使用单元的其他配置。例如，相似的原理可应用于单元列。相应地，多种实施例限于单元行。

除了上述的优势，说明性的实施例并不要求 CPU105 保留指针的跟踪。相反，可使用简单的模命令和计数器来递增指针直到满足一定的条件（例如，读取指针指向与由基指针所指向的地址位置相同的地址位置）。相应地，优化了 CPU 的使用。此外，当装载其他条时处理一个图像数据单元（即流水线作业）也改善了系统的性能。

本发明的多种实施例可以至少部分实现在任何传统的计算机程序语言内。例如，一些实施例可在过程程序语言（如“C”语言）或面向目标程序语言（如“C++”语言）内实现。本发明的其他实施例可以实现为编程的硬件元件（如专用集成电路、FPGA 以及数字信号处理器）或其他相关的元件。

在替代的实施例中，所公开的装置和方法可实现为用于与计算机系统结合使用的计算机程序产品。这种实现可包括一系列的计算机指令，这些指令可固定在有形介质上，如计算机可读介质（如磁盘，CD-ROM，ROM 或硬盘），或通过调制解调器或其他接口设备（如通过介质连接至网络的通信适配器）可向计算机系统传送。该介质既

---

可以是有形介质（如光缆或模拟通信线路）也可以是使用无线技术实现的介质（如微波、红外线或其他传输技术）。

该一系列的计算机指令包含本文前面所描述的所有或部分关于系统的功能。本领域的技术人员会理解，这种计算机指令可被写入到许多程序语言中，用于与许多计算机构造或操作系统结合使用。而且，这种指令可存储在任何存储设备内，如半导体、磁性的、光学的或其他的存储设备，并可使用任何通信技术进行传输，如光缆、红外线、微波或其他传输技术。

期望这种计算机程序产品可作为带附属打印或电子文档（如紧缩套装软件）的可移除介质销售，预装载进计算机系统（如在系统 ROM 上或硬盘上），或通过网络从服务器或电子公告板（如，因特网或万维网）进行销售。当然，可执行本发明的一些实施例可实现为软件和硬件的组合（如计算机程序产品）。本发明的其他实施例可实现为完全的硬件或完全的软件（如计算机程序产品）。

尽管已公开了本发明各种示例实施例，但对本领域的技术人员来说，作各种修改以实现本发明的优势而不超出本发明的范围，这是显而易见的。这些和其他明显的修改将包括在所附的权利要求内。

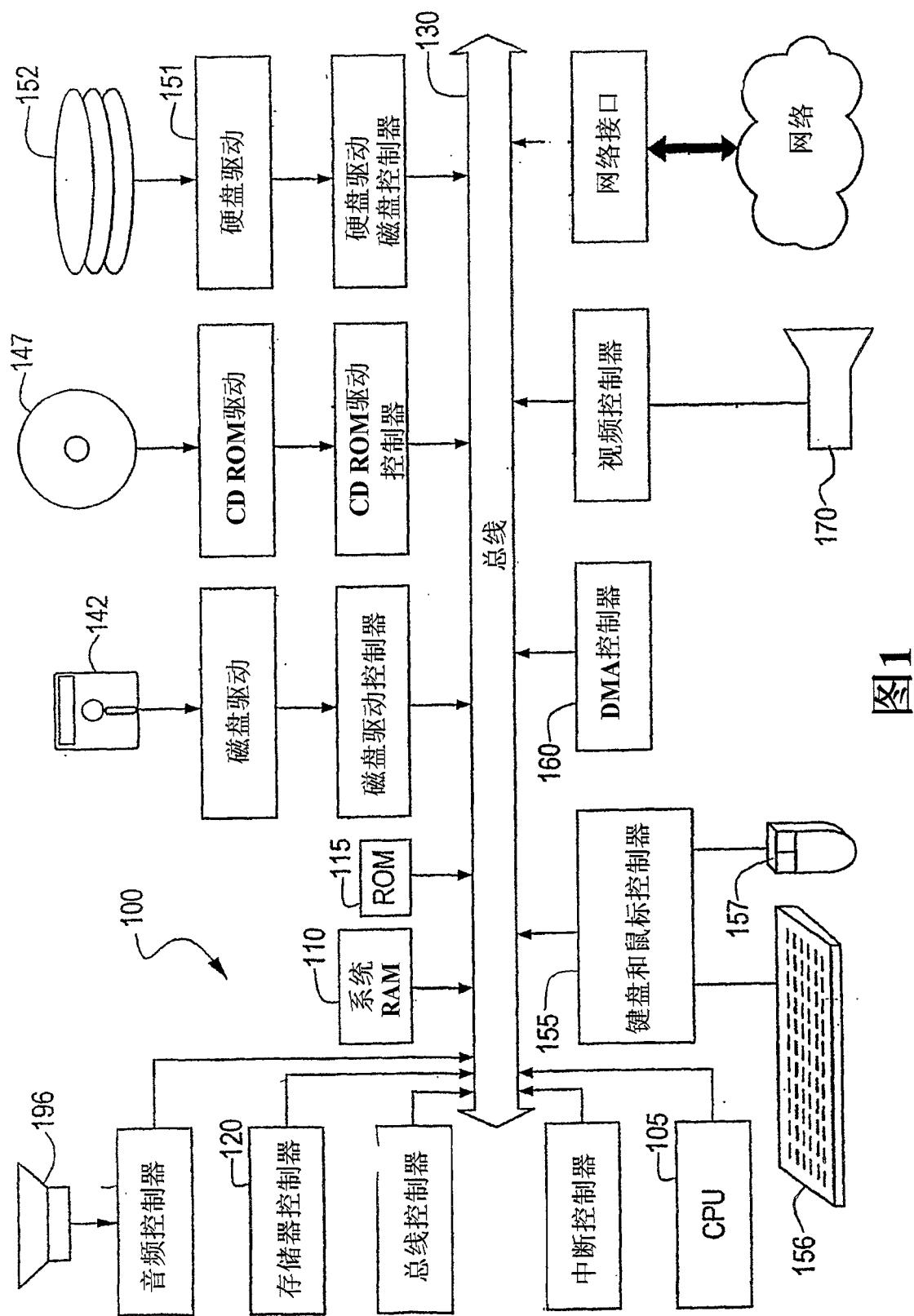


图1

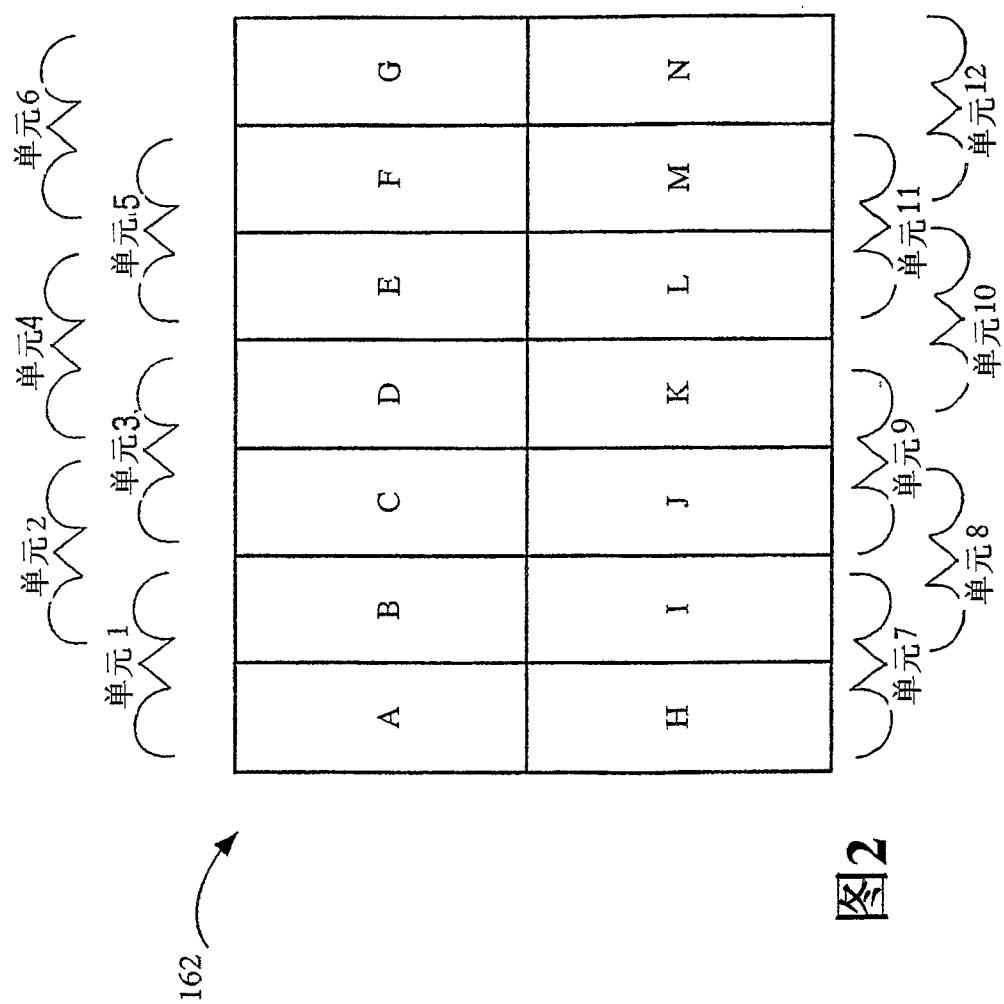


图2

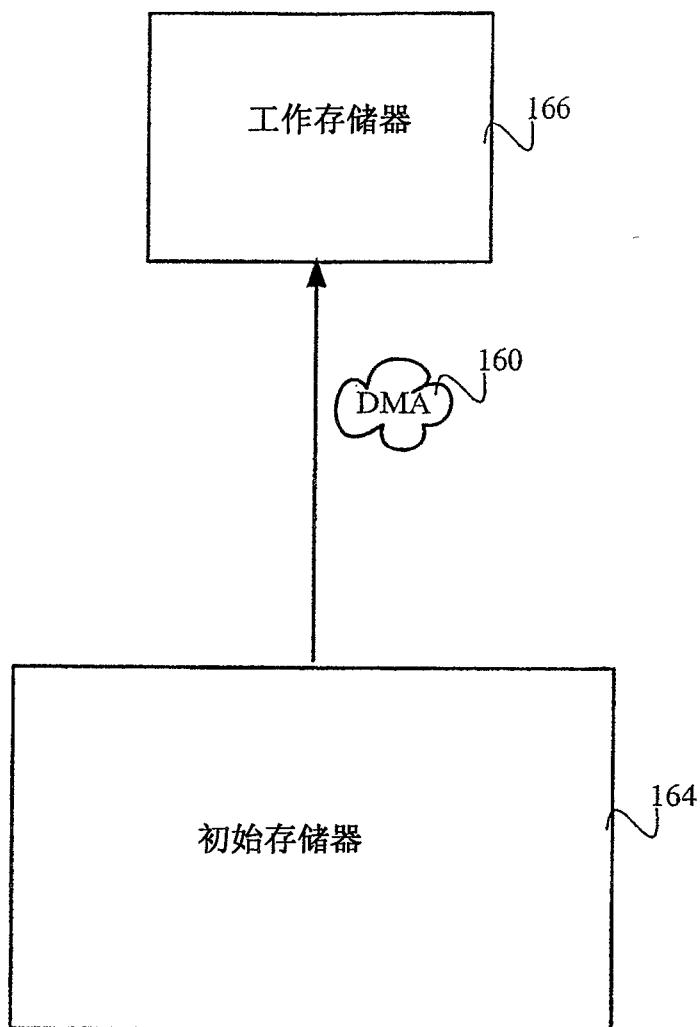


图3

图4A

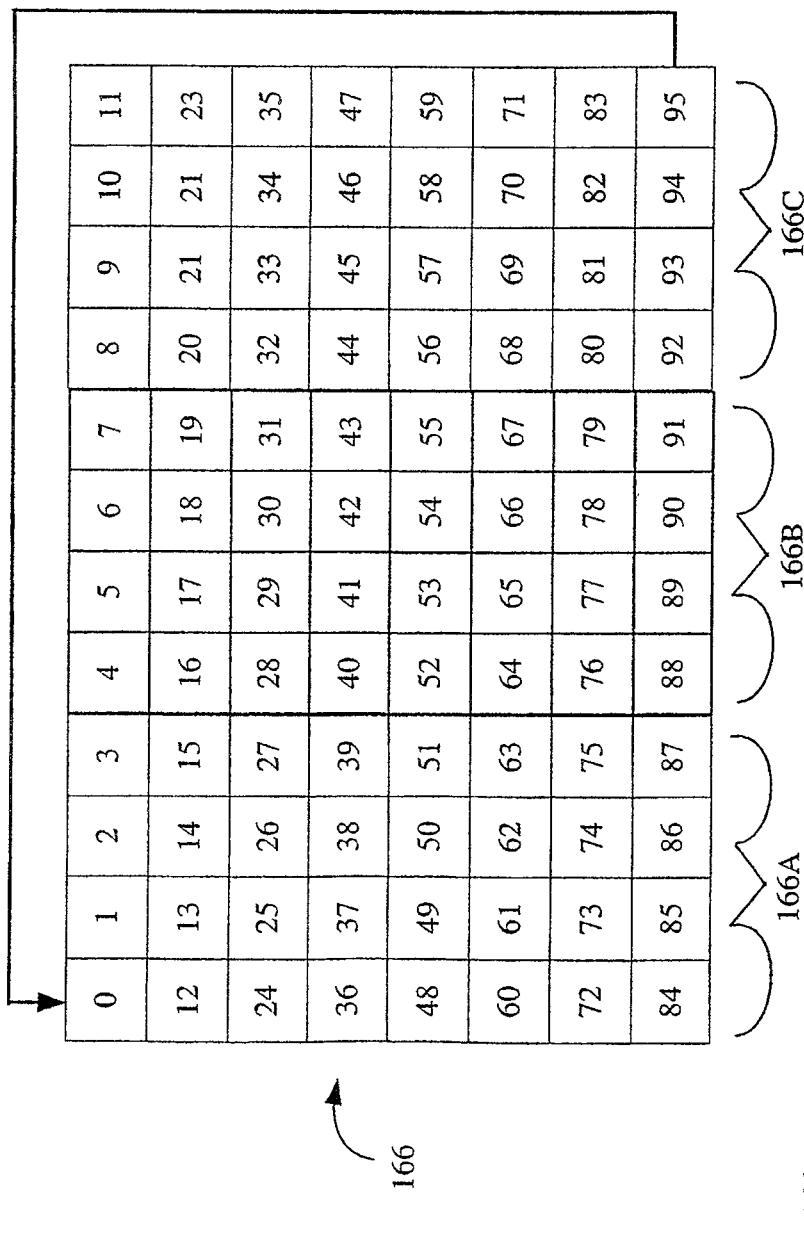
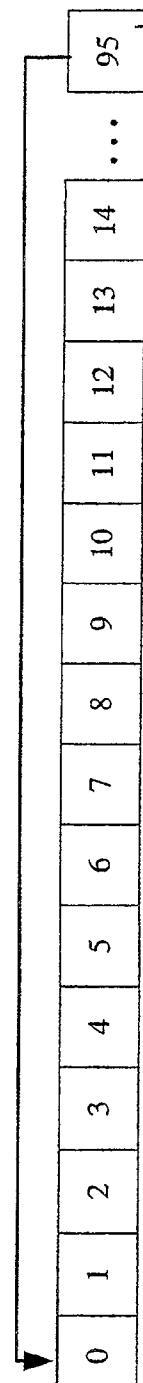


图4B



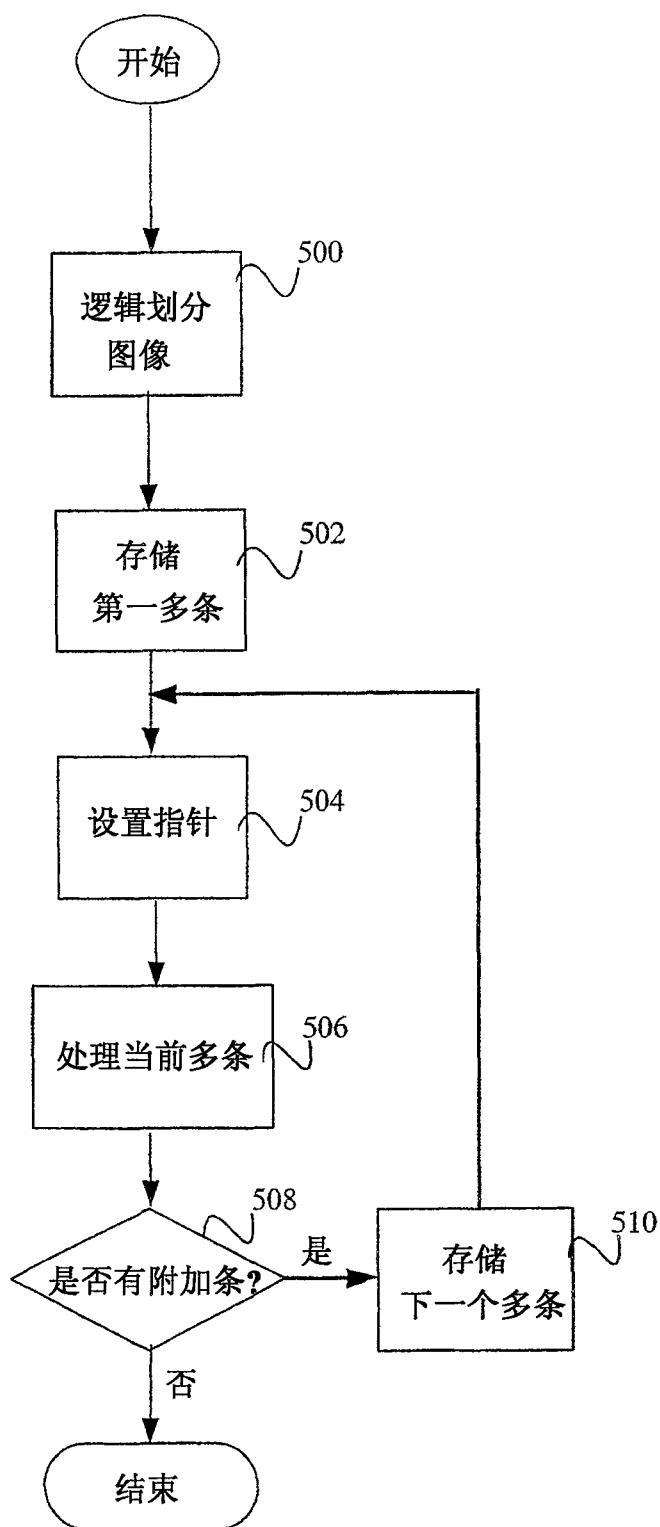


图5

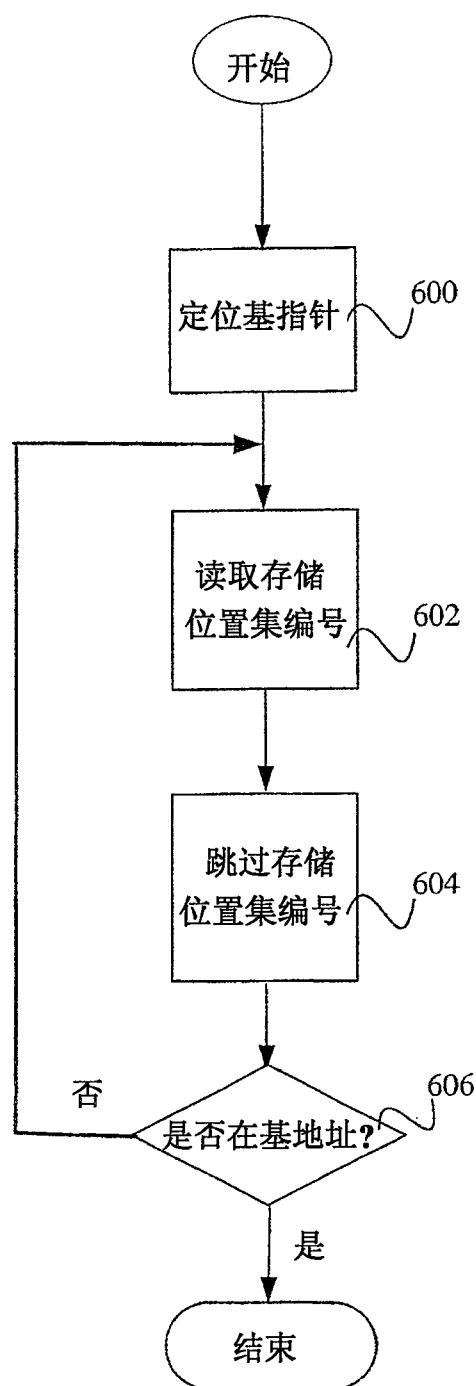


图6