

[19] 中华人民共和国国家知识产权局

[51] Int. Cl.

G06F 3/06 (2006.01)

G06F 12/02 (2006.01)



[12] 发明专利申请公布说明书

[21] 申请号 200680009222.3

[43] 公开日 2008 年 4 月 16 日

[11] 公开号 CN 101164037A

[22] 申请日 2006.2.8

[21] 申请号 200680009222.3

[30] 优先权

[32] 2005.2.16 [33] US [31] 11/060,249

[86] 国际申请 PCT/US2006/004658 2006.2.8

[87] 国际公布 WO2006/088727 英 2006.8.24

[85] 进入国家阶段日期 2007.9.21

[71] 申请人 桑迪士克股份有限公司

地址 美国加利福尼亚州

[72] 发明人 艾伦·韦尔什·辛克莱

彼得·约翰·史密斯

[74] 专利代理机构 北京律盟知识产权代理有限责任公司
代理人 刘国伟

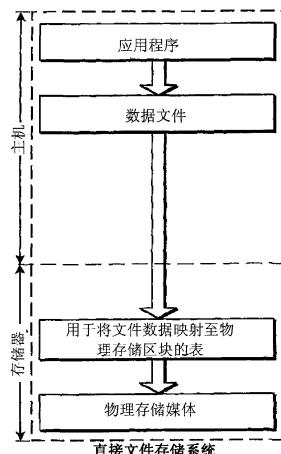
权利要求书 2 页 说明书 46 页 附图 32 页

[54] 发明名称

闪速存储器中的直接数据文件存储

[57] 摘要

使用每一文件的唯一标识以及数据在文件内的偏移量，但不使用任何中间逻辑地址或存储器的虚拟地址空间，将主机系统数据文件直接写入至大的擦除区块闪速存储器系统中。由存储器系统的控制器而非由主机在存储器系统内维护所述文件存储于存储器中的位置的目录信息。主机与存储器系统之间基于文件的接口使存储器系统控制器能够以增高的效率利用存储器内的数据存储区块。



1、一种在主机系统与具有存储单元的可再编程非易失性大容量存储系统之间传送数据的方法，所述存储单元被组织成可一同擦除并存储多个数据单位的存储单元区块，在向单独区块中写入新数据之前擦除单独区块，其中：

所述主机系统通过唯一的文件标识符以及数据在单独文件内的偏移量来标识其所产生的所述单独数据文件，并将这些文件标识符及偏移量发送至所述大容量存储系统，

所述大容量存储系统每次将从所述主机系统接收的文件数据的至少一个单位写入到至少一个先前被擦除的存储单元区块中，直至所述接收的主机系统文件数据已被写入为止，其中一单独数据单位包含多于 512 个字节，及

所述大容量存储系统直接将从所述主机接收的文件标识符及偏移量转换成向其中写入所述经标识文件的数据的存储单元区块的物理地址。

2、如权利要求 1 所述的方法，其中所述单独数据单位包含至少 1024 个字节。

3、如权利要求 2 所述的方法，其中将至少 8 个数据单位写入至所述大容量存储系统的单独区块内。

4、如权利要求 1 所述的方法，其中所述大容量存储系统的区块中的存储单元被布置成多个行，且写入数据包括在所述至少一个区块内每行写入至少两个数据单位。

5、如权利要求 4 所述的方法，其中写入数据另外包括将所述数据单位按顺序从所述至少一个区块的一端至其另一端每次写入至所述至少一个区块内所述存储单元的一行。

6、如权利要求 1 所述的方法，其中所述大容量存储系统通过维持大容量存储系统区块的对应地址的目录以及所述唯一文件标识符和偏移量而将从所述主机接收的所述文件标识符及偏移量转换成存储单元区块的物理地址。

7、如权利要求 1 所述的方法，其中写入所述所接收数据包括每次将所述所接收主机文件数据的一个单位写入到至少两个先前被擦除的存储单元区块。

8、如权利要求 7 所述的方法，其另外包括将各个区块链接在一起作为将向其中写入所述所接收主机文件数据的元块。

9、如权利要求 1 所述的方法，其中按照 NAND 架构组织所述存储单元。

10、如权利要求 1 所述的方法，其中将所述大容量存储系统包含于具有外部接点的壳体内，所述外部接点适于与产生所述主机数据文件的主机相连接。

11、如权利要求 1 所述的方法，其中所述大容量存储系统包含足以存储至少 256 兆字节数据的若干个存储单元。

12、一种闪速存储器系统，其具有至少 256 兆字节的数据存储容量的存储器及所述存储器的控制器，所述存储器被组织成其存储元件分组成可一同擦除的元件区块，

所述控制器根据一种包括如下步骤的方法操作：

从所述存储器系统的外部接收拟存储在所述存储器中的由主机产生的数据文件，各单独所接收数据文件的数据是由唯一文件标识符及数据在所述文件内的偏移量来标识，

标识被擦除的存储元件区块，

每次将所述经接收数据文件的一个或多个数据单位存储在所述经标识区块中，所述数据单位分别包含多于 512 个字节的所接收数据，及

在所述存储器内维持所述区块以及所述区块内所述各单独数据文件的数据通过其唯一文件标识符及数据在所述文件内的偏移量而存储于其中的存储元件的地址转换记录。

13、如权利要求 12 所述的方法，其中所述数据单位各自包含至少 1024 个字节的所接收主机文件数据。

14、如权利要求 12 所述的方法，其中存储所述所接收数据文件包括每次将所述所接收主机文件数据的一个单位写入到至少两个被擦除存储单元区块内。

15、如权利要求 14 所述的方法，其另外包括将各个区块在逻辑上链接在一起作为将向其中存储所述所接收主机文件数据的元块。

16、如权利要求 12 所述的方法，其中所述控制器操作方法是在容纳于壳体内的所述闪速存储器系统中进行，所述壳体具有适于与产生所述主机文件的主机相连接的外部接点。

17、一种大容量存储存储器系统，其包括：

可再编程非易失性存储单元，其形成于至少一个半导体衬底上，所述存储单元被布置成多个可一同擦除并一同连接成多个串联存储单元串的存储单元区块，所述多个串联存储单元串共享跨越其延伸以界定存储单元行的导电字线，各单独行中的所述存储单元存储至少 1024 个字节的数据，且所述区块分别包含至少八个存储单元行，

控制器，其包括微处理器，所述微处理器适于接收各自由唯一文件标识符及数据在所述文件内的偏移量标识的数据文件，并致使所接收数据文件被存储在一个或多个存储单元区块中，及

由所述控制器维持的地址转换记录，其标识其中使用唯一文件标识及数据在所述文件内的偏移量存储所述所接收数据的所述区块及存储单元行。

18、如权利要求 17 所述的存储器系统，其中所述控制器将至少一些所述存储单元区块连接在一起以形成多个元块，所述元快各自包含两个或两个以上区块，且其中所述控制器使接收的数据文件的数据并行写入至少一个元块的所述区块的行中。

19、如权利要求 17 所述的存储器系统，其中所述存储器系统包含在手持壳体内，所述手持壳体具有适于与产生所述数据文件的主机可移除地连接的外部接点。

闪速存储器中的直接数据文件存储

技术领域

本申请案涉及例如半导体闪速存储器等可再编程非易失性存储器系统的操作，且更具体而言，涉及对主机装置与存储器之间接口的管理。本文所提及的所有专利、专利申请案、论文及其它公开案的全部内容均以引用方式并入本文中。

背景技术

在早期的一代商业闪速存储器系统中，将一存储单元矩形阵列划分为大量单元群组，其中每一单元群组均存储一标准磁盘驱动器扇区的数据量，即 512 个字节。通常在每一群组中还包含额外量的数据（例如 16 个字节），以存储一错误修正码(ECC)及其他可能与用户数据及与存储该用户数据的存储单元群组有关的开销数据。每一此类群组中的存储单元均为可一同擦除的最小组数量的存储单元。换句话说，擦除单位实际上是存储有一个数据扇区及所包含的任何开销数据的存储单元数量。此种类型存储器系统的实例阐述于美国专利第 5,602,987 号及第 6,426,893 号中。闪速存储器的一特性是在以数据对存储单元再编程之前需要擦除存储单元。

闪速存储器系统最常以存储卡或闪速驱动器的形式来提供，其以可抽换方式连接各种各样的主机（例如个人计算机、照相机或类似装置），但也可嵌于此类主机系统中。在传统系统中，当将数据写入至存储器时，主机通常在存储器系统中一连续的虚拟地址空间内为扇区、群集或其他数据单位指配唯一的逻辑地址。如磁盘操作系统(DOS)一样，主机将数据写入存储器系统的逻辑地址空间内的地址中并自所述地址中读取数据。该存储器系统内的控制器将自主机接收的逻辑地址转换为存储器阵列内实际存储有数据的物理地址，并随后记录所述地址转换。存储器系统的数据存储容量至少与在为该存储器系统界定的整个逻辑地址空间内可寻址的数据量一样大。

在随后的几代闪速存储器系统中，擦除单位的大小增大至由足以存储多个数据扇区的存储单元形成的区块。尽管存储器系统所连接到的主机系统可按小的最小单位（例如扇区）对数据进行编程及读取，然而在闪速存储器的单个擦除单位中要存储大量的扇区。随着主机对逻辑数据扇区的更新或替换，区块内的某些数据扇区常常会变得过时。由于必须将整个区块擦除才能覆写存储于所述区块中的任何数据，因而新的或更新后的数据通常存储于另一已得到擦除并具有用于所述数据的剩余容量的区块中。该过程使原始区块带有过时数据，该等过时数据会占用存储器内的宝贵空间。但是，如果其中剩余任何有效数据，则该区块便不能被擦除。

因此，为更好地利用存储器的存储容量，常常通过将有效的局部区块数据量复制至一被擦除区块中来合并或收集所述有效的局部区块数据量，以便可随后擦除从其复制这些数据的区块并重新利用其整个存储容量。为以数据扇区的逻辑地址的次序来划分区块内的数据扇区，也需要复制数据，因为这会提高读取数据及将所读取数据传送至主机的速度。如果此种数据复制进行得过于频繁，则存储器系统的操作性能可能会变差。当存储器的存储容量几乎不大于主机可通过所述系统的逻辑地址空间所定址的数据量时（这是一种典型的情形），此尤其会影响存储器系统的操作。在此种情形中，可能需要进行数据合并或收集才能执行主机编程命令。此时，编程时间会变长。

在连续的几代存储器系统中，为增大可在一既定半导体区域中存储的数据位数，区块的大小一直在增大。存储 256 个数据扇区或更多数据扇区的区块正在变得日益常见。另外，常常将不同阵列或子阵列的两个、四个或更多个区块在逻辑上链接成元块，以增大数据编程及读取的平行度。与这样大容量的操作单元相伴而来的是在有效地对其进行操作方面所面临的挑战。

发明内容

目前已开发出许多技术以在不同程度上克服在有效地操作这样大的擦除区块闪速存储器系统方面所遇到的某些问题。本发明另一方面通过改变存储器与主机系统之间的数据传送接口而采用一种更基本的方法。其并非如当前所进行的一样使用虚拟地址空间内的逻辑地址在存储器与主机系统之间传送数据，而是通过由主机所指配的文件名以及文件内的偏移地址来标识数据文件。存储器系统由此得知每一扇区或其他数据单位所属的主机文件。本文所论述的文件单位是一有序（例如通过具有顺序性的偏移地址）且由在主计算系统中运行的应用程序所创建并唯一标识的数据集合。

当前的商业存储器系统并不采用这种方法，因为主机现在通过共用的一组逻辑地址集合向存储器系统标识所有文件内的数据，而不标识文件。通过以文件对象识别主机数据而非使用逻辑地址，存储器系统控制器可以一种会降低如此频繁地进行数据合并及收集的需要的方式来存储数据。数据复制操作的频率以及所复制的数据量由此会显著减小，从而增强存储器系统的数据编程及读取性能。进一步，所述存储器系统控制器维护其中存储主机文件的存储区块的目录及索引表信息。因而无需使主机维护当前为管理逻辑地址接口所需的文件分配表（FAT）。

本发明的其他方面、优点、特征及细节包含于下文对本发明实例性实例的说明中，该说明应结合附图一起阅读。

附图说明

图 1 示意性地图解说明如当前所实施的主机及所连接的非易失性存储器系统；

图 2 是用作图 1 所示非易失性存储器的一实例性闪速存储器系统的方块图；

-
- 图 3 是可用于图 2 所示系统中的存储单元阵列的代表性电路图；
图 4 图解说明图 2 所示系统的一实例性实体存储器组织形式；
图 5 显示图 4 所示实体存储器的一部分的展开图；
图 6 显示图 4 及 5 所示实体存储器的一部分的又一展开图；
图 7 图解说明主机与可再编程存储器系统之间的常见现有技术逻辑地址接口；
图 8 以不同于图 7 的方式图解说明主机与可再编程存储器系统之间的常见现有技术逻辑地址接口；
图 9 图解说明根据本发明主机与可再编程存储器系统之间的直接文件存储接口；
图 10 以不同于图 9 的方式图解说明根据本发明主机与可再编程存储器系统之间的直接文件存储接口；
图 11 显示一实例性存储器系统的功能层次；
图 12A-12E 给出一组实例性的直接文件接口命令；
图 13A-13D 显示将数据文件直接写入存储器内的四个不同实例；
图 14A-14E 显示一将单个数据文件直接写入存储器内的序列；
图 15 显示对图 14E 中所示数据文件进行垃圾收集的结果；
图 16 给出常见区块的一实例；
图 17 图解说明将一共用数据群组编程至数个开放共用区块之一内；
图 18 显示将数据元页编程至非易失性存储器内不同文件中的数个实例；
图 19 图解说明一文件索引表 (FIT) 的结构以及来自图 14A、14C、14E 及 15 所示实例的表项；
图 20 概念性地图解说明一实例性文件检索数据结构；
图 21 显示图 20 所示文件目录的页的结构；
图 22 显示图 20 所示文件索引表的页的结构；
图 23 显示作为对图 21 所示结构的替代，图 20 所示文件目录的页的结构；
图 24 图解说明图 21 及 23 所示文件目录的操作；
图 25 图解说明图 22 所示文件索引表的操作；
图 26 是一流程图，其显示本文所述存储器系统的总体操作顺序；
图 27 是图 26 所示“读取文件数据”区块的流程图；
图 28 是图 26 所示“对文件数据编程”区块的流程图；
图 29 图解说明包含于图 28 所示流程图中的两个操作的相对定时；
图 30 是图 26 所示“删除文件”区块的流程图；
图 31 是图 26 所示“垃圾收集”区块的流程图；
图 32 是图 31 所示“共用区块垃圾收集”区块的流程图；及
图 33 是在本文实例性存储器系统的所述操作期间存储单元区块的状态图。

具体实施方式

闪速存储器系统大体说明

参照图 1-8 来说明当前的闪速存储器系统及与主机装置的典型操作。本发明的各个方面即可构建于此一系统中。图 1 所示主机系统 1 将数据存储于闪速存储器 2 中并从闪速存储器 2 检索数据。尽管闪速存储器可嵌于主机内，然而图中将存储器 2 图解说明为更流行的卡形式，此种卡形式通过机械及电连接器的配合部件 3 及 4 可拆卸地连接至主机。当前存在诸多不同的市售闪速存储卡，其实例为小型闪存卡(CF)、多媒体卡(MMC)、安全数字(SD)卡、迷你 SD 卡、存储棒、智能媒体卡及 TransFlash 卡。尽管这些卡中的每一个均根据其标准化规范而具有唯一的机械及/或电接口，然而每一卡中所包含的闪速存储器均极为相似。这些卡均可自 SanDisk 公司(本申请案的受让人)购得。SanDisk 还以其 Cruzer 商标提供一组闪速驱动器，其为小封装形式的手持存储器系统，其具有一用于通过插入主机的 USB 插孔内而与主机相连接的通用串行总线(USB)。这些存储卡及闪速驱动器中的每一个均包含与该主机介接并控制其中的闪速存储器作业的控制器。

使用此类存储卡及闪速驱动器的主机系统数量众多且多种多样。其包括个人计算机(PC)、膝上型计算机及其他便携式计算机、蜂巢式电话、个人数字助理(PDA)、数字照相机、数字摄像机及便携式声频播放器。通常，主机包含一用于一个或多个类型的存储卡或闪速驱动器的内建插孔，但有些需要使用适配器来承插存储卡。

就存储器 2 而言，图 1 所示主机系统 1 可视为具有两个主要部件，该两个主要部件是由电路与软件组合构成。其为应用程序部分 5 以及与存储器 2 介接的驱动器部分 6。例如，在个人计算机中，应用程序部分 5 可包含一运行字处理、图形、控制或其他流行应用程序软件的处理器。在照相机、蜂窝式电话或者其他主要专用于执行单组功能的主机系统中，应用程序部分 5 包括用于操作照相机进行拍照及存储照片、操作蜂窝式电话拨打及接收电话等等的软件。

图 1 所示存储器系统 2 包括闪速存储器 7 及电路 8，二者均介接所述卡所连接到的主机，以来回传递数据并控制存储器 7。在数据编程及读取期间，控制器 8 通常在由主机 1 所用数据的逻辑地址与存储器 7 的物理地址之间进行转换。

参见图 2，对可用作图 1 所示非易失性存储器 2 的典型闪速存储器系统的电路进行说明。所述系统控制器通常构建于单个集成电路芯片 11 上，集成电路芯片 11 通过系统总线 13 与一个或多个集成电路存储器芯片并联，在图 2 中是显示单个此种存储器芯片 15。所示特定总线 13 包含单独的一组用于载送数据的导体 17、一组用于存储器地址的导体 19 以及一组用于控制及状态信号的导体 21。另一选择为，可使单组导体在这三种功能之间分时共享。此外，可采用系统总线的其他配置，例如在 2004 年 8 月 9 日提出申请且名称为“环形总线结构及其在闪速存储器系统中的应用(Ring Bus Structure and Its Use in Flash Memory Systems)”的第 10/915,039 号美国专利申请案中所述。

典型的控制器芯片 11 具有其自身的内部总线 23，内部总线 23 通过接口电路 25

介接系统总线 13。通常连接至所述总线的主要功能是处理器 27（例如微处理器或微控制器）、包含用于将系统初始化（“引导”）的只读存储器（ROM）29、主要用于缓冲在存储器与主机直接传送的数据的只读存储器（RAM）31、以及电路 33，电路 33 为在存储器与主机之间通过控制器的数据计算及检查纠错码（ECC）。控制器总线 23 通过电路 35 介接一主机系统，在包含于一存储卡中的图 2 所示系统情形中，这是通过所述卡中作为连接器 4 一部分的外部触点 37 来实现。时钟 39 与控制器 11 的每一其他组件相连并供所述每一其他组件使用。

存储器芯片 15 以及与系统总线 13 相连的任何其他存储器芯片通常包含一组织成多个子阵列或平面形式的存储单元阵列，为简明起见，在图中显示两个此种平面 41 及 43，但也可改为使用更多个（例如四个或八个）此种平面。另一选择为，芯片 15 的存储器单元阵列可不划分成平面。然而，当如此划分时，每一平面均具有其自身的可相互独立操作的列控制电路 45 及 47。电路 45 及 47 从系统总线 13 的地址部分 19 接收其各自存储单元阵列的地址，并将其解码，以对特定的一个或多个各自位线 49 及 51 进行寻址。响应于在地址总线 19 上接收到的地址而通过行控制电路对字线 53 进行寻址。源极电压控制电路 57 及 59 也与各自平面相连，p-阱电压控制电路 61 及 63 也是如此。如果存储器芯片 15 具有单个存储单元阵列，且如果在系统中存在两个或更多个此种芯片，则每一芯片的阵列均可类似于上述多平面芯片内的平面或子阵列进行操作。

数据通过与系统总线 13 的数据部分 17 相连的各自数据输入/输出电路 65 及 67 传入及传送出平面 41 及 43。电路 65 及 67 使得通过各自的列控制电路 45 及 47 既能够将数据编程入存储单元内，又能够从其各自的存储单元中读取数据。

尽管控制器 11 控制存储器芯片 15 的操作，以对数据编程、读取数据、擦除及照管各种内务，然而每一存储器芯片还包含某些用于从控制器 11 执行命令的控制电路来执行这些功能。接口电路 73 连接至系统总线 13 的控制及状态部分 21。来自控制器的命令提供至状态机 75，状态机 75 随后提供对其他电路的特定控制，以便执行这些命令。控制线 77-81 将状态机 75 与这些其他电路相连，如在图 2 中所示。来自状态机 75 的状态信息通过线 83 传送至接口 73，以便通过总线部分 21 传输至控制器 11。

存储单元阵列 41-43 的 NAND 架构在当前是较佳的，当然还可改为使用例如 NOR 等其他架构。作为存储器系统的一部分的 NAND 闪速存储器及其操作的实例可通过参考美国专利第 5,570,315 号、第 5,774,397 号、第 6,046,935 号、第 6,373,746 号、第 6,456,528 号、第 6,522,580 号、第 6,771,536 号及第 6,781,877 号及美国专利申请公开案第 2003/0147278 号而获得。

图 3 的电路图图解说明一实例性 NAND 阵列，其是图 2 所示存储器系统的存储单元阵列 41 的一部分。提供大量的全局位线，然而为使解说简明起见，在图 2 中仅显示四个此种线 91-94。若干个串联连接的存储单元串 97-104 连接于这些位线之一与一参考电位之间。使用存储单元串 99 作为代表性实例，多个电荷存储存储单元 107-110

与所述串两端处的选择晶体管 111 及 112 串联连接。当使串的选择晶体管导电时，所述串便连接于其位线与参考电位之间。随后每次对该串内的一个存储单元进行编程或读取。

图 3 所示字线 115-118 分别贯穿若干存储单元串中每一者内一个存储单元的电荷存储元件，且栅极 119 及 120 控制所述串每一端处的选择晶体管的状态。使共享共用字线及控制栅极线 115-120 的存储单元串形成一被一同擦除的存储单元区块 123。该单元区块包含可同时被物理擦除的最小数量的单元。每次对沿其中一条字线 115-118 的一行存储单元进行编程。通常，以一规定次序对 NAND 阵列中的各行进行编程，在本实例中是从最接近于连接至地或另一共用电位的串的端部的沿字线 118 的行开始。接下来编程沿字线 117 的存储单元行，依此类推而贯穿整个区块 123。最后，对沿字线 115 的行进行编程。

第二个区块 125 与此类似，其存储单元串与第一个区块 123 中的串连接至相同的全局位线，但具有不同的一组字线及控制栅极线。字线及控制栅极线由行控制电路 55 驱动至其恰当的操作电压。如果在系统中存在不止一个平面或子阵列，例如图 2 中的平面 1 及 2，则一个存储器架构使用在其间延伸的共用字线。另一选择为，可存在不止两个共享共用字线的平面或子阵列。在其他存储器架构中，分别驱动各单独平面或子阵列的字线。

如在上文所提及的几个 NAND 专利及公开申请案中所述，可对存储器系统进行操作，以在每一电荷存储元件或区域中存储不止两个可检测的电荷电平，从而在每一者中存储不止一个数据位。存储单元的电荷存储元件最常为导电性浮动栅极，但也可为非导电性介电电荷陷获材料，如在第 2003/0109093 号美国专利申请公开案中所述。

图 4 概念性地图解说明在下文进一步说明中用作一实例的闪速存储单元阵列 7(图 1)的组织形式。四个存储单元平面或子阵列 131-134 可位于单个集成存储单元芯片上、两个芯片（每一芯片上两个平面）上或四个单独芯片上。该特定排列对下文说明而言并不重要。当然，在一系统中还可存在其他数量的平面，例如 1 个、2 个、8 个、16 个或更多个。所述平面分别划分成在图 4 中由矩形所显示的存储单元区块，例如分别位于平面 131-134 内的区块 137、138、139 及 140。每一平面内可存在数十个或数百个区块。如上文所述，存储单元区块是擦除单位，即可一同进行物理擦除的最小数量的存储单元。然而，为增强平行性，各区块是以更大的元块单元形式操作。将来自每一平面的一个区块在逻辑上链接在一起而形成一元块。图中显示四个区块 137-140 形成一个元块 141。一元块内的所有单元通常被一同擦除。用于形成元块的各区块不必限定于在其各自平面内位于相同的相对位置上，如在由区块 145-148 所构成的第二元块 143 中所显示。尽管通常较佳可使元块遍布在所有平面上，然而为获得高的系统性能，可使存储器系统在操作时能够动态地形成由不同平面内一个、两个或三个区块中的任何或所有区块组成的元块。此使元块的大小能够更密切地匹配在一次编程作业内可供存储的数据量。

出于操作目的，如在图 5 中所图解说明，又将各个区块划分成存储单元页。例如，区块 131–134 中每一者的存储单元均被划分为八个页 P0–P7。或者，每一区块内可存在 16 个、32 个或更多个存储单元页。页是区块内的数据编程及读取单位，其含有可同时编程的最小数据量。在图 3 所示的 NAND 架构中，页是由区块内沿一字线的存储单元形成。然而，为增强存储器系统操作的平行性，可将两个或更多个区块内的此类页在逻辑上链接成元页。图 1D 中图解说明一个元页 151，其是由四个区块 131–134 中每一个中的一个物理页形成。元页 151（例如）包含这四个区块中每一个中的页面 P2，但一个元页中的各页未必必须在每一区块内具有同一相对位置。尽管较佳在所有四个平面中平行地对最大数据量实施编程及读取，然而为获得高的系统性能，还可操作存储器系统以由不同平面内各单独区块中的一个、两个或三个页中的任何或所有页形成元页。此使编程及读取作业能够自适应性地与可便于平行处理的数据量相匹配，并减少其中元页的一部分仍保持未以数据编程的情形。

如在图 5 中所示，一由多个平面的物理页形成的元页包含沿那些多个平面的字线行的存储单元。并非同时将一个字线行中的所有单元编程，其更通常是以两个或更多个交错的群组实施交替编程，其中每一群组均存储一数据页（位于单个区块中）或一数据元页（跨越多个区块）。通过每次对交替的存储单元实施编程，将不需要为每一条位线提供一由包含数据寄存器及感测放大器的周边电路形成的单元，而是在相邻位线之间分时共享该单元。此会节约周边电路所需的衬底空间大小并能够沿各个行以增大的密度包装存储单元。相反，为使一既定存储器系统可提供的平行性最大化，较佳同时编程沿一行的每一单元。

参照图 3，通过沿 NAND 串的至少一端提供两行选择晶体管（未显示）而非图中所示的单个行，会最方便地实现对沿一行的每隔一个存储单元的同时数据编程。随后，响应于一个控制信号，一行中的各选择晶体管在区块中每隔一个串地连接至其各自的位线，且响应于另一控制信号，另一行的各选择晶体管在每隔一个串中间连接至其各自的位线。因此，在每一行存储单元中写入两个数据页。

每一逻辑页中的数据量通常是一个或多个数据扇区的整数倍，按照惯例，每一扇区包含 512 个字节的数据。图 6 显示一页或元页的数据中两个扇区 153 及 155 的逻辑数据页。每一扇区通常包含一由 512 个字节的所存储用户或系统数据形成的部分 157 及另一数量的开销数据字节 159，该另一数量的开销数据字节 159 与所述部分 157 中的数据相关或者与存储所述数据的物理页或区块相关。开销数据的字节数量通常是 16 个字节，从而使每一扇区 153 及 155 总共为 528 个字节。开销部分 159 可包含一在编程期间根据数据部分 157 计算出的 ECC、其逻辑地址、所述区块已被擦除及再编程的次数的经验计数值、一个或多个控制旗标、工作电压电平、及/或类似内容、加上一根据此开销数据 159 计算出的 ECC。另一选择为，可将开销数据 159 或其一部分存储于其他区块的不同页中。

随着存储器平行性的增大，元块的数据存储容量会增大且因此数据页及元页的大

小也会增大。数据页可因而包含多于两个数据扇区。由于一数据页中具有两个扇区，且每一元页具有两个数据页，因而在一元页中存在四个扇区。每一元页因此存储 2048 个字节的数据。这是一较高的平行度，且可能会随着各行中存储单元数量的增大而更进一步增大。出于此种原因，为增大页及元页中的数据量，闪速存储器的宽度正在扩大。

可购得的上文所识别的小型可再编程非易失性存储卡及闪速驱动器的数据存储容量为 512 兆字节(MB)、1 吉字节(GB)、2 GB 及 4 GB，且可变得更高。图 7 图解说明主机与此一大容量存储器系统之间最常见的接口。该主机处理由该主机所执行的应用软件或固件程序所产生或使用的数据文件。字处理数据文件即为一实例，且计算机辅助设计(CAD)软件的制图文件为另一实例，其主要见于一般计算机主机(例如 PC、膝上型计算机及类似装置)内。pdf 格式的文档也是此类文件。静态数字摄像机为存储于存储卡上的每一照片产生一数据文件。蜂窝式电话利用来自一内部存储卡上的文件(例如电话目录)的数据。PDA 存储并使用几个不同文件，例如地址文件、日历文件及类似文件。在任一此类应用中，存储卡还可含有用于操作主机的软件。

在图 7 中图解说明主机与存储器系统之间的常用逻辑接口。一连续逻辑地址空间 161 大到足以以为所有可存储于存储器系统内的数据提供地址。主机地址空间通常被划分为数据群集的递增量。每一群集均可在一既定主机系统内设计成含有一定数量的数据扇区，通常为 4 至 64 个扇区不等。一标准扇区含有 512 个字节的数据。

图 7 的实例中显示已创建了三个文件 1、2 及 3。一在主机系统上运行的应用程序以一有序数据集形式来创建每一文件，并通过唯一名称或其他参引方式来识别该文件。由主机为文件 1 指配尚未分配给其他文件的足够大的可用逻辑地址空间。图中显示已为文件 1 指配一连续的可用逻辑地址范围。通常还将某些地址范围分配给特定用途，例如为主机操作软件分配一特定范围，从而即使在主机正为数据指配逻辑地址时所述地址尚未得到利用，也不会使用所述地址来存储所述数据。

当此后主机创建一文件 2 时，主机同样地指配逻辑地址空间 161 内两个不同的连续地址范围，如图 7 中所示。无需为一文件指配连续逻辑地址，而是还可分配位于已分配给其他文件的地址范围中间的地址片断。此实例随后显示，将主机地址空间中先前未分配给文件 1 及 2 及其他数据的其他部分分配给由主机创建的再一文件 3。

该主机通过保持一文件分配表(FAT)来记录存储器逻辑地址空间，其中该主机指配给各种主机文件的逻辑地址均保持于该文件分配表中。FAT 表通常存储于非易失性存储器中及一主机存储器中，并当存储新文件、删除其他文件、修改文件及进行此类作业时由主机频繁地加以更新。例如，当删除一主机文件时，该主机随后通过更新 FAT 表而将先前分配给所删除文件的逻辑地址解除分配，以显示其现在可供用于其他数据文件。

主机并不关心存储器系统控制器选择用于存储所述文件的物理位置。典型的主机仅知晓其逻辑地址空间及其已分配给其各文件的逻辑地址。另一方面，通过一典型主

机/卡接口，存储器系统仅知晓逻辑地址空间中已被写入数据的部分，但不知晓分配给特定主机文件的逻辑地址，或甚至主机文件数量。存储器系统控制器 106 将主机所提供的用于存储或检索数据的逻辑地址转换成存储有主机数据的闪速存储单元阵列内的唯一物理地址。一区块 163 表示一由存储器系统控制器保持的关于这些逻辑-物理地址转换的工作表。

存储器系统控制器编程成以一种使系统性能保持处于高水平的方式将数据文件存储于一存储器阵列 165 的区块及元块内。在此图解说明中使用四个平面或子阵列。较佳在由每一平面中的一区块所形成的整个元块中以系统所允许的最大平行度来编程及读取数据。通常分配至少一个元块 167 作为一预留区块来存储存储器控制器所使用的操作固件及数据。可分配另一元块 169 或多个元块来存储主机操作软件、主机 FAT 表及此类数据。大多数物理存储空间仍用于存储数据文件。然而，存储器系统控制器并不知晓主机是如何在其各个文件对象之间分配所接收的数据。通常，存储器控制器从与主机的交互作用中仅获知：主机写入至特定逻辑地址的数据是存储于由控制器的逻辑-物理地址表 163 所保持的对应物理地址中。

在一典型存储器系统中，除所需区块以外还提供几个额外区块的存储容量以存储地址空间 161 内的数据量。可提供这些额外区块中的一个或多个作为冗余区块来取代可能在存储器使用寿命期间变得有缺陷的其他区块。包含于各个元块内的区块的逻辑群组通常可出于多种原因而改变，包括用一冗余区块替代一最初指配给该元块的缺陷区块。通常在一已擦除区块池内保持有一个或多个额外区块，例如元块 171。当主机将数据写入存储器系统时，控制器将主机所指配的逻辑地址转换为已擦除区块池内一元块中的物理地址。随后将未用于存储逻辑地址空间 161 内的数据的其他元块擦除并指定为已擦除池区块，以供在下一数据写入操作期间使用。

在原始所存储数据变得过时时，存储于特定主机逻辑地址处的数据会频繁地受到新数据的覆写。存储器系统控制器响应于此而将新数据写入已擦除区块中并随后针对那些逻辑地址来改动逻辑-物理地址表，以识别那些逻辑地址处的数据所存储至的新物理区块。包含位于那些逻辑地址处的原始数据的区块然后被擦除并可供用于存储新数据。若在写入开始之前在来自已擦除区块池的预先擦除区块中不存在足够的存储容量，则常常必须在可完成一当前数据写入操作之前进行此种擦除。此可不利地影响系统的数据编程速度。存储器控制器通常仅当主机将新数据写入至其相同逻辑地址时才知晓主机已使一既定逻辑地址处的数据过时。因此，存储器的许多区块可能正在存储此种无效数据达一定时间。

为有效地利用集成电路存储器芯片的面积，区块及元块的尺寸正在增大。此使大部分单独数据写入所存储的数据量小于一元块的存储容量，且在许多情形中甚至小于一区块的存储容量。由于存储器系统控制器通常将新数据指引至已擦除池元块，因此此可导致元块的某些部分未得到填充。若新数据是存储于另一元块中的某些数据的更新值，则来自该另一元块中具有与新数据元页邻接的逻辑地址的其余有效数据元页页

也较佳地以逻辑地址次序复制至新元块内。旧元块可保持其他有效数据元页。此会随着时间而使一单独元块的某些元页的数据变得过时且无效，并被写入至一不同元块的具有相同逻辑地址的新数据所取代。

为保持足以在整个逻辑地址空间 161 内存储数据的物理存储器空间，周期性地压缩或合并（垃圾收集）此种数据。还希望尽可能地以与数据扇区的逻辑地址相同的次序将数据扇区保持于元块内，因为此会使读取邻接逻辑地址中的数据更为有效。因此，通常以该额外目标来实施数据压缩及垃圾收集。当接收到局部区块数据更新值时对存储器实施管理的某些方面以及元块的使用阐述于第 6,763,424 号美国专利中。

数据压缩通常涉及到从元块中读取所有有效数据元页并将其写入至新区块中，在该过程中忽略具有无效数据的元页。具有有效数据的元页也较佳地以与其中所存储数据的逻辑地址次序相一致的物理地址次序进行排列。占据新元块的元页数量将小于占据旧元块的元页数量，这是因为包含无效数据的元页不被复制至新元块中。旧区块随后被擦除并可供用于存储新数据。通过合并所获得的额外元页的容量随后可用于存储其他数据。

在垃圾收集期间，从两个或更多个元块收集具有邻接或接近邻接的逻辑地址的有效数据元页，并将其重写至另一元块内，通常是被擦除区块池中的一个元块内。当从所述原始的两个或更多个元块复制出所有有效数据元页时，其便可被擦除以供将来使用。

数据合并及垃圾收集会耗用时间并可影响存储器系统的性能，特别是当需要进行数据合并或垃圾收集才能执行来自主机的命令时。这些操作通常由存储器系统控制器进行调度，以尽可能地在后台进行，但执行这些操作的需要可能导致控制器须向主机发出忙状态信号，直到此种操作完成为止。可将主机命令的执行延迟的一实例是：在被擦除区块池中没有足够的预先被擦除元块来存储主机想要写入至存储器中的所有数据，且需要进行垃圾收集来清空一个或多个随后可被擦除的有效数据元块。因此，为使此种破坏最小化，一直在关注对存储器控制的管理。在如下各美国专利申请案中阐述了许多这种技术：2003 年 12 月 30 日提出申请且名称为“对具有大擦除区块的非易失性存储器系统的管理（Management of Non-Volatile Memory Systems Having Large Erase Blocks）”的第 10/749,831 号申请案；2003 年 12 月 30 日提出申请且名称为“具有区块管理系统的非易失性存储器及方法（Non-Volatile Memory and Method with Block Management System）”的第 10/750,155 号申请案；2004 年 8 月 13 日提出申请且名称为“存储器平面对齐的非易失性存储器（Non-Volatile Memory and Method with Memory Planes Alignment）”的第 10/917,888 号申请案；2004 年 8 月 13 日提出申请的第 10/917,867 号申请案；2004 年 8 月 13 日提出申请且名称为“进行分阶段编程故障处理的非易失性存储器及方法（Non-Volatile Memory and Method with Phased Program Failure Handling）”的第 10/917,889 号申请案；以及 2004 年 8 月 13 日提出申请且名称为“带控制数据管理的非易失性存储器（Non-Volatile Memory and Method with Control

Data Management)”的第 10/917,725 号申请案。

在有效控制具有极大擦除区块的存储器阵列的操作方面所面临的一个调整是使在一既定写入操作期间所存储的数据扇区数量与存储器区块的容量及边界一致并对接。一种方法是以不到最大数量的元块来配置一用于存储来自主机的新数据的元块，所述不到最大数量的元块是为存储不到能填充一完整元块的量的数据量所需的。自适应性元块的使用阐述于 2003 年 12 月 30 日提出申请且名称为“自适应性元块(Adaptive Metablocks)”的第 10/749,189 号美国专利申请案中。数据区块之间边界与元块之间物理边界的配合阐述于 2004 年 5 月 7 日提出申请的第 10/841,118 号专利申请案以及 2004 年 12 月 16 日提出申请且名称为“数据运行编程 (Data Run Programming)”的第 11/016,271 号专利申请案中。

为更有效地操作存储器系统，存储器控制器也可使用来自 FAT 表的数据，所述 FAT 表由主机存储于非易失性存储器中。一个此种用途是获知主机何时通过解除数据的逻辑地址而将数据标识为过时。得知此时刻便使存储器控制器能够在其通常通过主机向那些逻辑地址写入新数据而获知此时刻之前对擦除包含此种无效数据的区块进行调度。此阐述于 2004 年 7 月 21 日提出申请且名称为“用于在非易失性存储器系统上保持数据的方法及装置 (Method and Apparatus for Maintaining Data on Non-Volatile Memory Systems)”的第 10/897,049 号美国专利申请案中。其他技术包括监测主机向存储器写入新数据的图案，以推断一既定写入操作是否为单个文件，或者如果是多个文件，则这些文件之间的边界位于何处。2004 年 12 月 23 日提出申请且名称为“进行 FAT 分析来实现最佳化依序群集管理 (FAT Analysis for Optimized Sequential Cluster Management)”的第 11/022,369 号美国专利申请案即阐述了此种类型技术的应用。

为有效地操作存储器系统，希望使控制器尽可能多地得知由主机指配给其各个文件的数据的逻辑地址。然后，由控制器将数据文件存储于单个元块或元块群组内，而非在不知道文件边界时分散于较大量元块中。结果使数据合并及垃圾收集操作的数量减少。如此一来，存储器系统的性能便得到提高。但如上文所述，当主机/存储器接口包含逻辑地址空间 161 (图 7) 时，存储器控制器很难得知关于主机数据文件结构的很多知识。

参见图 8，其以不同方式图解说明在图 7 中所已显示的典型逻辑地址主机/存储器接口。由主机为主机所产生的数据文件分配逻辑地址。存储器系统随后查看这些逻辑地址并将其映射入实际存储有数据的存储单元块的物理地址。

对实例性基于文件的接口实施例的说明

主机与用于存储大量数据的存储器系统之间的改良接口使得无需使用逻辑地址空间。主机通过唯一的文件 ID (或另一唯一的参考量) 及数据单元 (例如字节) 在文件内的偏移地址对每一文件进行寻址。该文件地址是直接提供给存储器系统控制器，存储器系统控制器然后保持其自己的关于实际上在何处存储每一主机文件的数据的表。该新的接口可使用与上文参照图 2-6 所述的相同的存储器系统来构建。与上文所

述的主要区别是存储器系统与主机系统进行通信的方式。

该基于文件的接口图解说明于图 9 中，应将其与图 7 所示逻辑地址接口相比较。文件 1、2 及 3 中每一者的标识及数据在图 9 所示文件内的偏移量直接传递至存储器控制器。然后，由一存储器控制器功能 173 将该逻辑地址信息转换成存储器 165 的元块及元页的物理地址。

在图 10 中还图解说明基于文件的接口，应将其与图 8 所示逻辑地址接口相比较。在图 10 中不存在图 8 所示逻辑地址空间及主机所保持的 FAT 表。而是，通过文件号及数据在文件内的偏移量来向存储器系统识别主机所产生的数据文件。然后，存储器系统将文件直接映射至存储单元阵列的物理区块。

参见图 11，其图解说明本文所述的一实例性大容量存储系统的各功能层。其为“直接文件存储后端系统”，是本说明的主要说明对象。此处于存储器系统的操作内，且通过“直接文件接口”以及“基于文件的前端系统”、经由基于文件的接口信道与主机系统进行通信。每一主机文件均例如通过文件名来唯一地识别。文件内的数据是通过在该文件所独有的线性地址空间内的偏移地址来识别。无需为存储器系统界定逻辑地址空间。

命令

来自主机系统的一组直接文件接口命令支持存储器系统的操作。实例性的一组此种命令在图 12A-12E 中给出。这些只是简要概述，以供在本说明其余部分中作参考。图 12A 列示用于根据所界定的协议使数据在主机与存储器系统之间传送的主机命令。指定文件 (<fileID>) 在该文件内处于特定偏移量 (<offset>) 处的数据或者被写入至存储器系统，或者从存储器系统读出。在传输写入(Write)、插入(Insert)或更新(Update)命令之后，从主机向存储器系统传输数据，且存储器系统通过将所述数据写入其自己的存储器阵列中而作出响应。主机传输读取(Read) 命令会使存储器系统通过将指定文件的数据发送至主机而作出响应。如果存储器系统保持一指针来标识可存储所述文件的额外数据的下一存储位置，则无需随写入命令发送数据偏移量。然而，如果写入命令包含在早已写入的文件内的偏移位置，则存储装置可将其解译为一用于更新在该偏移位置处开始的文件数据的命令，从而无需一单独的更新命令。对于读取命令，如果要读取整个文件，则主机无需规定时间偏移量。这些图 12A 所示数据命令之一的执行是响应于主机系统传输任何其他命令而结束。

另一数据命令是移除(Remove)命令。不同于图 12A 中的其他数据命令，在移除命令后不进行数据传输。其作用是使存储器系统将所规定偏移量 1 与偏移量 2 之间的数据标记为过时。然后，在对存在过时数据的文件或区块的下一次数据压缩或垃圾收集期间，移除这些数据。

图 12B 列示用于管理存储器系统内的文件的主机命令。当主机要将新文件的数据写入于存储器系统中时，其首先发出打开(Open)命令，且存储器系统通过打开新文件来作出响应。通常将规定可同时保持打开的文件的数量。当主机关闭文件时，关闭

(Close) 命令将告知存储器系统其用于保持所打开文件的资源可改向。存储器系统通常将立即对此一文件进行调度来进行垃圾收集。对于所说明的直接文件接口，垃圾收集是以逻辑方式得到管理并主要对文件执行，而非以物理方式对各单独存储单元区块执行。关闭_之后 (Close_after) 命令向存储器系统发出文件即将被关闭的通知。文件删除 (file Delete) 命令使存储器系统根据所规定优先权规则立即对包含所要擦除文件中的数据的存储单元区块进行调度。擦除 (Erase) 命令规定将所规定文件的数据从存储器中立即擦除。

删除命令与擦除命令之间的主要区别是将优先权赋予擦除指定文件的数据。主机可使用擦除命令在最早可行的时刻移除存储器中的安全数据或敏感数据，而删除命令则使该数据以较低的优先权被擦除。在将存储器系统断电时使用擦除命令会在从主机取出存储装置之前移除敏感数据，且由此防止在下一次使用所述存储装置期间将该数据散布给其他用户或者主机系统。这两个命令均较佳在后台执行，即不会放慢主数据命令的执行 (图 12A)。总之，自主机接收到另一命令将通常使存储器控制器结束任何后台操作。

涉及到存储器系统内的目录的主机命令列示于图 12C 中。每一目录命令均包括对该命令所属的目录的标识 (<directoryID>)。尽管存储器系统控制器保持所述目录，然而关于所述目录以及所述目录的名称的命令是由主机系统提供。存储器控制器依照存储于存储器系统中的固件、使用主机所提供的目录名称来执行这些命令。

<fileID>参数可以是文件的完整路径名、或者文件的某个简短标识符—在本文中称作文件句柄 (file_handle)。文件路径名与某些命令相结合地提供至图 11 所示的直接文件接口。这使得当文件首次打开时能够在文件目录中创建完全显式的条目，并使得当打开现有文件时能够存取文件目录中的正确现有条目。文件路径名语法可符合 DOS 文件系统所使用的标准。路径名描述目录的层次及位于目录最低层内的文件。路径段可由 “\” 来划界。带有前缀 “\” 的路径是以根目录为基准。而不带有前缀 “\” 的路径则以当前目录为基准。一 “..” 段则表示当前目录的母目录。

另一选择为，打开的文件可由一文件_句柄参数来标识，该文件_句柄参数是在首次创建所述文件时由存储装置指配。所述存储装置可此后每当主机打开所述文件时将简短的文件名称传送至主机。主机可此后将所述文件_句柄与打开的文件的写入、插入、更新、读取、关闭及关闭_之后命令一起使用。主机对文件的存取通常将比使用完整路径名时快，因为不需要在文件目录的层次中进行导航。当使用打开命令首次打开文件时，通常使用完整路径名，因为存储器系统可能尚未为该文件指配文件_句柄。但是如果已经具有，则可使用文件_句柄。对于图 12A 及 12B 中利用文件 ID 的其余删除及擦除命令，则较佳使用完整的文件路径名作为一种安全措施来防止主机提供不正确的文件_句柄。主机会更难无意间产生一与现有的但非指定的文件相匹配的不正确路径名。

通过其所属目录的<directoryID>标识，图 12C 中的各目录命令类似地由图 11 中的直接文件接口接收到。完整路径名是使用目录命令接收到的优选目录 ID。

文件_句柄是一简短标识符，其是由大容量存储装置响应于打开命令而在直接文件接口处返送回主机。将所述文件_句柄定义为指向所述文件目录条目中所存在 FIT 的指针会较为方便。该指针界定该文件在区块内的逻辑 FIT 区块编号以及逻辑文件编号。使用此作为文件_句柄便能够在无需首先搜索文件目录中的文件的情况下存取文件 FIT 条目。例如，如果存储装置可具有多达 64 个 FIT 区块，且每一 FIT 区块可对多达 64 个文件加索引，则一帶文件_句柄 1107 的文件将指向其在 FIT 中的数据群组条目的指针设定为 FIT 区块 11 中的逻辑文件 7。该文件_句柄是在响应于打开命令而创建文件的目录及 FIT 条目时由存储器系统控制器产生，并响应于关闭命令而变无效。

图 12D 给出用于管理主机与存储器系统之间接口状态的主机命令。空闲 (Idle) 命令告知存储器系统其可执行先前已得到调度的内部操作，例如数据擦除及垃圾收集。响应于接收到待机 (Standby) 命令，存储器系统将停止执行后台操作，例如垃圾收集及数据擦除。关机 (Shut-down) 命令则向存储器控制器预先发出即将掉电的告警，此允许完成未决的存储器操作，包括将数据从易失性控制器缓冲器写入至非易失性闪速存储器。

图 12E 中所示的大小 (Size) 命令将通常在一写入命令之前由主机发出。存储器系统响应于此而向主机报告可供进一步写入文件数据的可用容量。此可根据可用的未编程物理容量减去为管理对所规定文件数据容量的存储所需的物理容量来计算。

当主机发出一状态 (Status) 命令 (图 12E) 时，存储装置将以其当前状态作出响应。该响应可呈一个或多个具有不同的位字段的二进制字形式，从而向主机提供关于存储装置的不同的具体信息项。例如，一个两位的字段可报告所述装置是否正忙，且如果是，则根据存储装置所正忙于进行的事项来提供不止一个忙状态。一个忙状态可指示存储装置正在忙于执行一用于传送数据的主机写入或读取命令—此为后台操作。第二忙状态指示可用于在存储器系统正在执行后台内务操作 (例如数据压缩或垃圾收集) 时告知主机。主机可决定是否等待至该第二忙状态结束之后再向存储装置发送另一命令。如果在内务操作完成之前发送另一命令，则存储装置将结束内务操作并执行所述命令。

主机可将第二装置忙状态与空闲命令相结合来允许在存储装置内进行内务操作。在主机发送可能使得需要使装置进行内务操作的命令、或一系列命令之后，主机可发送空闲命令。如下文所述，可对存储装置进行编程，使其通过启动内务操作来响应于空闲命令并同时如上文所述起动第二忙状态。例如，根据下文所述的演算法，删除命令会使得需要执行垃圾收集。在已发出一系列删除命令之后来自主机的空闲命令会随后使装置能够有时间来执行垃圾收集，而垃圾收集可能是为使存储装置能够响应于下一主机写入命令所必需的。否则，可能需要在接收到下一写入命令之后、但在可执行下一命令之前执行垃圾收集，从而使该指令的执行明显放慢。

写入数据

当将新的数据文件编程入存储器中时，从被擦除区块中第一物理位置开始并按次

序循序经过区块中的各位置来将数据写入被擦除存储单元区块中。数据是按从主机接收的次序进行编程，而无论所述数据在文件内的偏移量次序如何。编程继续进行，直到文件中的所有数据均已写入存储器内为止。如果文件中的数据量超过单个擦除区块的容量，则当第一区块变满时，编程会在第二被擦除区块中继续进行。第二存储器区块是以与第一存储器区块相同的方式，从第一位置起按顺序进行编程，直到文件中的所有数据均得到存储或者第二区块变满为止。可使用文件中的剩余数据对第三或其他区块进行编程。用于存储单个文件的数据的多个区块或元块不需要在物理上或在逻辑上邻接。为便于解释，除非另外指明外，否则本文中所使用的术语“区块”打算指代擦除区块单位或者多个区块“元块”，此视在具体系统中是否使用元块而定。

参见图 13A，图解说明将数据文件写入至存储器系统。在本实例中，数据文件 181 大于存储器系统中一个区块或元块 183 的存储容量（在图中显示为在两条垂直实线之间延伸）。因此，数据文件 181 的一部分 184 也写入至第二区块 185 中。图中显示这些存储单元区块在物理上邻接，但其并非必需如此。在自主机串流接收到文件 181 的数据时，写入所述数据，直到所述文件的所有数据均已写入至存储器中为止。在图 13A 的实例中，数据 181 是在图 12A 的写入命令之后从主机接收到的初始文件数据。

一种使存储器系统管理及记录所存储数据的较佳方式是使用可变大小数据群组。也就是说，将文件数据存储为多个数据群组，这多个数据群组可按规定次序链接在一起而形成完整的文件。然而，较佳地，数据群组在文件内的次序是由存储器系统控制器使用文件索引表（FIT）来保持。当正在写入来自主机的数据流时，每当在文件数据的逻辑偏移量位置中或在存储数据的物理空间中存在间断点时，均开始一个新的数据群组。此种物理间断点的一实例是当文件数据填满一个区块并开始写入至另一个区块时。此图解说明于图 13A 中，其中第一数据群组填充第一区块 183，所述文件的其余部分 184 则作为第二数据群组存储于第二区块 185 中。第二数据群组可由（F0, D0）表示，其中 F0 是数据文件的开头的逻辑偏移量，而 D0 是存储器内所述文件开头处的物理位置。第二数据群组可由（F1, D1）表示，其中 F1 是存储于第二区块 185 开头处的数据的逻辑文件偏移量，而 D1 是其中存储有该数据的物理位置。

通过主机-存储器接口传送的数据量可用数据字节数、数据扇区数或用某种其他粒度来表达。主机最常是以字节粒度来界定其文件的数据，但随后当通过当前逻辑地址接口与大容量存储器系统进行通信时，将字节分成分别有 512 个字节的扇区、或者分成分别有多个扇区的群集。为简化存储器系统的操作，通常如此进行。尽管本文所述的基于文件的主机-存储器接口可使用某种其他数据单位，然而一般较佳使用原始的主机文件字节粒度。也就是说，较佳以最小的合理数据单位字节而非以扇区、群集或类似单位来表达时间偏移量、长度及类似量。此能够更有效地使用本文所述技术来利用闪速存储器存储容量。

在常用的现有逻辑地址接口中，主机还规定所写入数据的长度。此也可使用本文所述的基于文件的接口来进行，但由于其并非是为执行写入命令（图 12A）所必需的，

因而较佳使主机不提供所写入数据的长度。

由此,以图 13A 所示方式写入存储器中的新文件在 FIT 中表示为所述数据群组的索引条目序列 (F0, D0), (F1, D1) (按该次序)。也就是说,每当主机系统想要存取一特定文件时,主机便将其文件 ID 或其他标识发送至存储器系统,存储器系统随后存取其 FIT 来识别构成该文件的数据群组。各单独数据群组的长度<length>也可包含于其各自的条目中,以便于存储器系统的操作。当使用时,存储器控制器计算并存储数据群组的长度。

只要主机使图 13A 中的文件保持打开状态,便较佳还保持一物理写入指针 P,以界定用于写入从主机接收到的该文件任何其他数据的位置。文件的任何新数据均写入文件在物理存储器中的末尾处,而无论新数据在文件内的逻辑位置如何。所述存储器系统允许多个文件同时保持打开,例如 4 个或 5 个此种文件,并为其中每一文件保持一写入指针 P。不同文件的写入指针指向不同存储器区块中的位置。如果在已达到存储器系统的打开文件数量的限值时,主机系统想要打开新文件,则首先关闭其中一个已打开的文件并随后打开新文件。在文件已关闭之后,便不再需要保持该文件的写入指针 P。

图 13B 图解说明主机也是使用写入命令(图 12A)将数据附加至先前所写入但仍打开的图 13A 所示文件的末尾。图中显示数据 187 由主机系统附加至文件的末尾,数据 187 还写入第二区块 185 中该文件的数据末尾处。所附加的数据变成数据群组 (F1, D1) 的一部分,因此,数据群组 (F1, D1) 现在包含更多数据,因为在现有数据群组 184 与所附加数据 189 之间既不存在逻辑地址间断也不存在物理地址间断。因而,整个文件仍在 FIT 中表示为一索引条目序列 (F0, D0), (F1, D1)。指针 P 的地址也改变至所存储的附加数据的末尾。

在图 13C 中显示将数据区块 191 插入先前所写入的图 13A 所示文件中的一实例。尽管主机正在将数据 191 插入文件中,然而存储器系统将所插入数据附加在先前所写入文件数据末尾处的一位置 193 上。当将数据插入打开的文件中时,不必将文件的数据以其逻辑次序进行重写,尽管在主机关闭文件后可在后台进行此种操作。由于所插入数据完全存储于第二存储器区块 185 内,因而其形成单个新的群组 (F1, D3)。但进行此种插入会导致图 13A 所示的先前数据群组 (F0, D0) 被划分成两个群组,一个群组 (F0, D0) 在所插入项之前,一个群组 (F2, D1) 在所插入项之后。这是因为每当所述数据存在逻辑间断(例如在插入项的开头 F1 及插入项的末尾 F2 处出现)时,均需要形成新的数据群组。群组 (F3, D2) 是物理位置 D2 作为第二区块 185 的开头的结果。群组 (F1, D3) 与 (F3, D2) 保持分开,尽管其存储于同一存储器区块中,这是因为其所存储的数据的偏移量存在间断。因而,带有所述插入项的原始文件在存储器系统 FIT 中由数据群组索引条目 (F0, D0)、(F1, D3)、(F2, D1)、(F3, D2) (按该次序) 表示。在图 13A、13B 及 13C 的实例中应注意,可在无需使存储器中的任何数据过时的情况下写入新的或现有的文件的新数据。也就是说,执行写入及插入

命令（图 12A）并不会使任何其他数据变得无效或过时。

图 13D 图解说明另一实例，其中使用更新命令（图 12A）对原先以图 13A 所示方式写入的数据的某一部分进行更新。图中显示更新数据文件的一部分 195。并非通过更新而在存储器系统中重写整个文件，而是将所述文件的一更新部分 197 附加至先前所写入的数据。先前所写入的数据的一部分 199 现在过时。尽管通常希望合并所更新文件，以便腾出由过时数据所占据的空间，然而在主机使所述文件保持打开时通常不这样进行，而是可在将文件关闭之后在后台进行。在更新之后，所述文件在存储器系统 FIT 中由数据群组索引条目 (F0, D0)、(F1, D3)、(F2, D1)、(F3, D2) 以该次序表示。图 13A 中的单个数据群组 (F0, D0) 又划分成图 13D 中的多个片断，一个位于更新部分之前、更新部分、且一个位于更新部分之后。

为进一步例示可变长度数据群组的使用，在图 14A-14E 中按次序显示涉及到同一文件的一系列数个写入操作。如在图 14A 中所示，首先使用写入命令（图 12A）将原始文件数据 W1 写入至存储器系统的两个区块中。此时，所述文件由两个数据群组界定：第一群组在一物理存储器区块的起始处开始，且第二群组是在一物理存储器区块边界之后需要。图 14A 所示文件因而由数据群组的如下索引条目序列描述：(F0, D0)，(F1, D1)。

在图 14B 中，使用更新命令（图 12A）更新在图 14A 中所写入的文件数据。更新的文件数据 U1 紧接在先前群组 (F1, D1) 之后立即写入，其中所更新数据的先前版本变得过时。图 14A 中的先前群组 (F0, D0) 缩短至图 14B 所示的修改后群组 (F0, D0)，且先前群组 (F1, D1) 缩短至群组 (F4, D2)。更新后的数据写入于两个群组 (F2, D3) 及 (F3, D4) 中，因为其交叠一存储器区块边界。某些数据存储于一第三存储器区块中。所述文件现在由数据群组的如下索引条目序列描述：(F0, D0)，(F2, D3)，(F3, D4)，(F4, D2)。

在图 14C 中通过使用插入命令（图 12A）插入新的文件数据 II 来进一步修改图 14B 所示文件。新数据 II 紧接在图 14B 的先前群组 (F4, D2) 之后写入存储器作为图 14C 中的新群组 (F5, D6) 及 (F6, D7)，因为所插入的数据交叠一存储器区块边界。使用一第四存储器区块。由于插入新数据 II，图 14B 的先前群组 (F0, D0) 分裂成图 14C 中的缩短的群组 (F0, D0) 及 (F7, D5)。所述文件现在由数据群组的如下索引条目序列描述：(F0, D0)，(F5, D6)，(F6, D7)，(F7, D5)，(F8, D3)，(F9, D4)，(F10, D2)。

图 14D 显示对图 14C 所示数据文件的进一步修改，其使用写入命令（图 12A）将新数据 W2 附加至文件末尾处。新数据紧接在图 14C 中的先前群组 (F10, D2) 之后写入作为图 14D 中的新群组 (F11, D8)。所述文件现在由数据群组的如下索引条目序列描述：(F0, D0)，(F5, D6)，(F6, D7)，(F7, D5)，(F8, D3)，(F9, D4)，(F10, D2)，(F11, D8)。

在图 14E 中显示对所打开文件的第二次更新，其中通过由主机发出一更新命令而

将所更新文件数据 U2 写入至图 14D 中的文件。所更新文件数据 U2 紧接在图 14D 中的先前群组 (F11, D8) 之后写入，其中该数据的先前版本变得过时。图 14D 中的先前群组 (F9, D4) 缩短至图 14E 所示的修改后群组 (F9, D4)，先前群组 (F10, D2) 变得完全过时，且先前群组 (F11, D8) 缩短至形成一新的群组 (F14, D9)。更新后的数据写入于图 14E 中的新群组 (F12, D10) 及 (F13, D11) 中，交叠区块边界。现在需要使用一第五区块来存储所述文件。所述文件现在由数据群组的如下索引条目序列描述：(F0, D0), (F5, D6), (F6, D7), (F7, D5), (F8, D3), (F9, D4), (F12, D10), (F13, D11), (F14, D9)。

在文件创建或根据前面的说明加以修改之后，每一文件的数据的偏移量较佳以正确的逻辑次序保持连续。因此，例如，作为执行插入命令的一部分，由主机提供的所插入数据的偏移量是从紧接所述插入部分之前的偏移量连续，且已处于所述文件中插入部分之后的数据递增所插入数据的量。更新命令最常使一现有文件的给定地址范围内的数据被类似量的更新数据替换，因而通常不需要替换所述文件中其他数据的偏移量。另一选择为，可使用写入命令取代单独的更新命令来更新文件的数据，因为存储器系统可将从主机接收到具有早已存在于所存储文件数据中的偏移量范围的数据解译为一要更新该偏移量范围中的数据的指令。

应注意，上文所述以及图 13 和 14 所示的所有数据分配及加索引功能均由存储器系统的控制器执行。与图 12A 所示写入命令、插入命令、或更新命令中的一者一起，主机仅传送所述文件内正发送至存储器系统的文件 ID 及数据偏移量。存储器系统进行其余操作。

以刚刚所述的方式将数据文件从主机直接写入至闪速存储器的一个优点在于，可使如此存储的数据的粒度或分辨率保持与主机的相同。如果主机应用程序以例如 1 字节的粒度写入文件数据，则该数据也可以 1 字节的粒度写入至闪速存储器内。然后，以字节数量来度量数据在单个数据群组内的量及位置。也就是说，可在主机应用程序文件内单独寻址的数据的相同偏移量单位在该文件存储于闪速存储器中时也可在该文件内单独寻址。随后，可在索引表中将区块内同一文件的数据群组之间的任何边界规定至最接近的字节或其他主机偏移量单位。类似地，区块内不同文件的各数据群组之间的边界以主机偏移量单位加以界定。

尽管不用于写入数据，然而也相关地考虑使用删除命令来删除文件的一部分，因为此种操作是插入命令的反向操作。删除命令可以如下格式应用于一文件偏移地址范围：删除<文件 ID><偏移量><长度>。所述文件中从<偏移量>开始且从该地址开始继续至删除<长度>的部分内的数据被删除。然后，在删除之后，主机使所述文件的其余数据的偏移地址递增，以在整个文件中保持邻接的偏移地址。这是插入命令的相反操作—在插入命令中，主机将数据加至文件的中间并然后使插入部分之后的其余文件数据的偏移量递增，以使修改后的文件的偏移量连续。

垃圾收集

从图 14B 及 14E 中应注意到，更新命令使为存储文件所需的物理空间大于文件中的数据量。这是因为已被更新部分所取代的数据仍保持存储于存储器中。因此，非常希望通过去除过时的、无效的数据而将文件数据合并（垃圾收集）至变小的物理存储空间内。因此，会由更大的存储空间可供用于其他数据。

亦可注意到，除图 14B 及 14E 所示的文件数据更新之外，图 14C 所示的数据插入也会使文件数据不按次序存储。也就是说，更新及插入部分是在进行更新及插入时附加至存储于存储器中的文件的末尾处，同时其差不多始终在逻辑上位于所述文件内的某处。图 14B、14C 及 14E 中的实例即为此种情形。因此，可能希望对存储于存储器中的文件的数据进行重新排序，以与在文件内的偏移量次序相一致。此随之会提高读取所存储数据的速度，因为按顺序读取各个页及区块将使所述文件的数据以其偏移量次序给出。此还会提供所述文件的最大可能的碎片消除。但对文件数据进行重新排序以使读取效率更高对于存储器系统的性能而言并不如文件数据合并重要，因为文件数据合并会潜在地释放一个或多个存储器区块来用于存储其他数据。因此，对文件中数据的重新排序在所增加的操作开销使其益处不值得时，将通常不会自己进行，但可作为许多垃圾收集操作的一部分来进行而几乎不会或根本不会增加操作开销。

图 14E 中的文件包含存储于存储器中的过时数据群组（灰色部分），因为已进行了这两处数据更新 U1 及 U2。因此，用于存储所述文件的存储容量的大小明显大于文件的大小，此在图 14E 中一目了然。因此，垃圾收集是恰当的。图 15 提供对图 14E 所示数据文件进行垃圾收集的结果的图解说明。该文件在垃圾收集之前占据接近五个区块的存储容量（图 14E），而同一文件在垃圾收集之后装在略大于三个存储单元区块内（图 15）。作为垃圾收集操作的一部分，从最初将数据写入至其他被擦除区块的区块中复制数据，并然后将所述原始区块擦除。若对整个文件进行数据收集，则可将其数据以与在文件内的数据逻辑偏移量次序相同的物理次序复制至新区块中。例如，更新部分 U1 及 U2 以及插入部分 II 在垃圾收集（图 15）之后以与其在主机文件中所出现的相同次序进行存储。

垃圾收集也通常会使得在所合并的文件内形成新的且不同的数据群组。在图 15 所示情形中，由新数据群组的如下新的索引条目序列来描述所述文件：(F0, D0), (F1, D1), (F2, D2), (F3, D3)。此数据群组数量比在图 14E 所示文件的状态中所存在的数据群组数量少得多。此时，其中已复制有所述文件的数据的每一存储单元区块存在一个数据群组。作为垃圾收集操作的一部分，更新文件索引表 (FIT) 来反映形成所述文件的新数据群组。

当一文件是垃圾收集的备选项时，对该文件的 FIT 数据群组条目进行检查，以判定所述文件是否满足垃圾收集的设定标准。当包含所述文件的数据的任何存储单元区块还包含过时数据时，将进行垃圾收集。否则，不需要进行垃圾收集。图 14E 中的文件包含过时数据从上面所给出的数据群组索引条目序列中一目了然。例如，从这两个连续数据群组 (F7, D5) 及 (F8, D3) 可以看出，在存储文件数据的前两个区块中存

在过时数据。物理地址位置 D5 与 D3 的差远大于逻辑偏移量 F7 与 F8 的差。根据文件索引数据群组条目还显而易见，所述数据未按逻辑偏移量次序存储。另一选择为，文件的各个 FIT 条目可保持该文件的被指过时的过时数据群组的记录。然后，控制器仅扫描每一文件的 FIT 条目，以识别任何过时的数据群组以及存储过时数据群组的物理区块。

通常不应对打开的文件执行垃圾收集，因为主机可能对所述文件继续进行更新或插入，从而进一步产生过时数据及/或不以逻辑偏移量次序存储数据。由此可造成许多此种垃圾收集操作。但在其中被擦除池区块的数量降至一设定水平以下时，可能需要对打开文件进行垃圾收集，以提供足够的被擦除区块来用于存储新数据或其他操作。

主机关闭文件通常会使得考虑对该文件进行垃圾收集。较佳不在文件关闭之后立即实施垃圾收集，而是在垃圾收集将不会干扰当前的存储器操作时由存储器控制器对一关闭的文件进行调度，以在后台进行垃圾收集。可保持一垃圾收集队列，其中在一文件关闭之后将所述文件加至所述队列中，且随后当不再存在任何其他所要实施的更高优先权存储器系统操作时，由存储器控制器选择已在所述队列中所处时间最长的文件，且进行垃圾收集（如果需要）。此种选择及垃圾收集操作可例如响应于从主机接收到一空闲命令（图 12D）而进行。

由于复制数据是垃圾收集中最耗时的部分，尤其是对于大的文件，因而可通过在短的猝发中在任一次仅复制文件数据的一部分来将该任务划分成多个组成部分。随后，可将此种局部文件复制与其他存储器操作进行交错，并甚至在主机正将数据传送至存储器系统或自存储器系统传送数据的同时进行。也可响应于被擦除区块的数量降至低于某个指定数量而增大各单独复制数据猝发的长度。

目标是完全在后台执行垃圾收集，而不干扰或减慢与主机系统传送数据的主操作。但并非总可以如此，尤其是如果被擦除区块池中可供用于编程新数据的被擦除区块的数量变得少于某个预设最小值时。此时，使垃圾收集优先，并可优先在后台对所述队列中的任何文件进行垃圾收集，以便合并数据，从而提供可供用于从主机系统接收新数据的额外被擦除区块。如果在所述队列中没有文件，则可能需要对打开的文件进行垃圾收集。当垃圾收集变为优先项时，主机将通常从存储器系统接收到忙状态信号，并将因此延缓对新数据的任何编程，直到再次存在足够数量的被擦除区块为止。相反，如果存在足够数量或更多的被擦除池区块，则可降低进行垃圾收集操作的频率，并推迟可能会影响存储器系统的性能的垃圾收集。

应注意，垃圾收集是对主机数据文件实施。响应于文件状态而对所述文件启动垃圾收集。当启动时，作为该过程的一部分，检查所述文件的所有数据群组在 FIT 中的索引条目。当对文件进行垃圾收集时，以在所述文件的数据群组索引条目中所规定的次序，将其数据每次一个数据群组地从其现有区块复制至一新打开的复制区块。此不同于现有闪速存储器垃圾收集—其是完全基于各单独存储单元区块的状态。

然而，一般规则是，将仅对存储所述文件的数据且也包含过时数据的各单独区块

进行垃圾收集。因此，并非对存储一文件的数据的所有区块均进行垃圾收集。如果一文件仅存储于例如两个区块中，且第一区块包含过时数据，而第二区块不包含过时数据，则将对第一区块进行垃圾收集，而不管第二区块的数据。将有效数据从第一区块复制至一被擦除复制区块中，且所述复制区块将随后具有某些被擦除的页或者其他容量剩下，所述某些被擦除的页或其他容量大约为过时数据的量。下文将说明使用不到一个区块的被擦除存储容量。在图 14E 的实例中，在进行垃圾收集之前，存储空间在包含所述文件数据的五个区块中的四个区块中均具有过时数据（灰色区域）。

作为对仅对包含过时数据的那些区块进行垃圾收集的一般规则的修改，一旦判定出将对一给定文件进行垃圾收集，便在垃圾收集操作中包含处于一局部填充区块中的所述文件的任何数据。因此，将图 14E 的第五区块中的 U2 数据包含于垃圾收集操作中，尽管在该区块中并不存在过时数据。通过将所有五个区块中的数据复制至四个被擦除区块中而合并所述数据，因为在不包含第五区块的数据时，所得到的四个复制区块中的两个将仅被局部地填充数据。在图 15 的情形中，仅一个复制区块保持被局部填充。通过具有更少的局部利用的区块来改善存储器性能。

由主机系统发出的图 12B 所示的某些文件命令可启动垃圾收集。上文已对接收到文件关闭命令进行了说明。所关闭的文件被置于所述队列中以进行垃圾收集。删除及擦除命令也可促成垃圾收集。删除文件可使包含过时文件数据的区块被置于垃圾收集队列中。对被删除文件进行垃圾收集的效果是不存在被删除文件的无效数据，因而不会进行向其他区块的数据复制。所有仅包含被删除文件的数据的区块仅作为垃圾收集过程的一部分被擦除。擦除命令具有类似的效果，只是对仅包含被擦除文件的过时数据的区块的垃圾收集可立即进行，或者根据优先权进行—例如通过将所述区块置于垃圾收集队列的顶部。

所述直接文件存储系统的一重要优点在于，当主机删除文件时，存储器会立即得知，因为用于进行此种操作的命令是直接发至存储器系统。随后，一旦可以进行擦除，存储器控制器便可立即擦除用于存储被删除文件的数据的区块，而不会不利地影响存储器的其他操作。被擦除区块随后便可供用于存储新数据。

在所述实施方案中，删除或擦除文件的命令均使该文件的数据直接响应于此而被擦除。相反，在典型的基于逻辑的接口中，此种命令则并不直接到达存储器控制器。而是，主机仅释放存储器逻辑地址空间中曾被所删除或被擦除文件占据的某些段。仅当主机此后向那些相同逻辑地址中的一个或多个写入数据时，存储器系统才得知所重新利用的逻辑地址段中的数据已经过时。存储器仍不知道所述文件所占据的其他逻辑地址段的数据已被删除或擦除。存储器仅在主机向那些其他逻辑地址写入数据时才知道先前写入至那些逻辑段的数据已经过时。

常用区块

使用单个文件的数据来填充连续的区块的结果是，当所述文件关闭并进行垃圾收集时，所述文件数据中的某些可能仅占据一存储单元区块的一部分。进一步，小文件

的数据可能甚至填充不到一整个区块。如果在一区块中仅可存储一个文件的数据，则此将导致各单独区块的大量空间得不到利用。各单独区块内的各页存储容量将保持被擦除并可供用于存储数据，但将得不到利用。

因此，某些存储单元区块较佳存储来自两个或更多个文件中每一文件的较少数据量（整个的小文件及/或在已填充其他区块之后剩下的残留文件数据）。残留文件数据是关闭的文件中未占据完整区块的数据，并可包含一个或多个数据群组。一文件映射表可记录单个存储单元区块中多于一个文件的数据群组，其记录方式仿佛所述区块中的所有数据群组均是一个文件的数据群组一样。各单独数据群组的映射表包含该数据群组作为一部分的文件的文件 ID。使用共用区块的主要目的是使可能因如上文所述将文件数据写入至存储器中而造成的未得到利用的物理存储器存储容量最小化。最常见地，因对文件进行垃圾收集而产生的一个或多个数据群组的残留数据将作为垃圾收集的一部分而被写入至一公用区块中。一实例是图 15 中进行垃圾收集的文件，其中最末数据群组（F3, D3）仅占据最末区块中的一小部分。如果其保持此种方式，则最末区块中的大部分将得不到利用。图 15 的数据群组（F3, D3）可写入至包含来自另外一个或多个文件的一个或多个数据群组的共用区块中，或者可将该最末存储器区块指定为共用区块。在后一情形中，来自一个或多个其他文件的数据群组将随后被写入至该同一区块中。

另一选择为，当已知残留数据量是一将适合装于其中一个指定共用区块内被擦除空间的量时，可在垃圾收集期间将残留文件数据直接从主机写入至共用区块中。关闭_之后命令（图 12B）可用于识别残留文件数据，并允许其写入一具有足以存储作为所述命令的一部分而规定的数据量的开放共用区块，而非将残留文件数据写入至将得不到完全填充的擦除池区块。此可消除在下一垃圾收集操作期间复制残留数据的需要。

当将新数据文件编程至存储器内时，作为对前面所述且在图 13A 中所示的数据写入方法的替代，可将数据写入至包含一个或多个其他文件的残留数据的开放共用区块中，从所述区块中第一可用物理位置开始并按次序依序经过所述区块的各位置。如果主机对一文件发送打开命令，随后立即在发送所述文件的数据之前发送关闭_之后命令，则可判定所述文件的全部数据可适合装于一开放共用区块内且所述文件将在到达所述开放共用区块的末尾之前被关闭。即使不知道新文件的数据的长度，也可将新文件的数据写入至开放共用区块中。

因此，较佳在存储器系统中保持若干个共用区块，以用于存储两个或更多个不同文件的共用数据群组。共用数据群组的大小可高达一区块的存储容量，粒度为一个字节。仅其中一个共用区块较佳包含一给定文件的数据，但可包含所述文件的多于一个数据群组。此外，一公用数据群组较佳存储于仅一个共用区块中，且不划分成存储于多个共用区块中。此会避免在所述数据群组变得过时时在多个共用区块中进行垃圾收集。当一公用区块中的数据群组变得过时时，对该共用区块进行垃圾收集。将此一公用区块中任何剩余的无效数据群组写入至另一公用区块中的可用被擦除空间中、或者

写入至一被擦除池区块中，并随后擦除所述共用区块。如果进行垃圾收集的共用区块包含来自不同文件的两个或更多个有效数据群组，则所述两个或更多个有效数据群组不需要保持在一起，而是可复制至不同的区块。

图 16 中显示一共用区块的实例，其已使用来自三个不同数据文件中每一数据文件的一共用数据群组进行编程。由于在一般的 NAND 阵列中限制写入数据从位于区块一端的页前进至位于另一端的页，因而使各数据群组相互邻接地进行存储。所述区块一端处的白色空间指示所述区块中尚未写入有数据的页。难以将可用数据群组理想地装入整个共用区块中。

被存储器系统指定为可在其中写入多于一个文件的残留数据的开放共用区块的数量将通常仅为几个，但如果被进行垃圾收集的大量文件具有不适合装入任一现有开放共用区块中可用空间的残留文件数据，则可变为许多个。将残留文件数据写入至其中一个能最佳地利用总存储容量的开放共用区块中。当在任一开放共用区块中不存在足够的被擦除空间时，关闭其中一个现有共用区块，并代之以打开另一共用区块。另一选择为，不需要关闭现有的开放共用区块，并可允许增加开放共用区块的数量。可将新的共用区块指定为其中一个被擦除池区块，但较佳是一已包含某些残留文件数据的未得到完全写入的区块，例如图 15 中仅包含残留文件数据群组 (F3, D3) 的最末区块。然而，为对共用区块进行垃圾收集，在必要时，较佳将一被擦除池区块指定为新的共用区块。

图 17 图解说明在如下情形中写入残留文件数据的实例性过程：其中存在五个已包含来自另一文件的残留数据的一个或多个数据群组的残留或共用单元（每一者具有一个不同的区块或元块）。从图 15 所示进行垃圾收集的文件产生的最末数据群组 (F3, D3) 即为此种残留文件数据的实例，尽管其可包含来自单个文件的多于一个数据群组。存在所示的三种可能性。第一种可能性 (A) 将残留数据写入至残留单元 2，因为其具有最多可用的被擦除空间。第二种可能性 (B) 为残留文件数据选择残留单元 5，因为该残留单元 5 是这五个残留单元中最适合的。残留文件数据接近填满单元 5，且由此在单元 2 中留下更大的可供用于在将来接收更大量数据的空间。在单元 1、3 或 4 的任一者中不存在足以接纳所述残留数据的空间，因此这些单元被立即排除。

第三种可能性 (C) 将残留文件数据写入至擦除池中的区块或元块单元中。一旦所述残留文件数据已写入至被完全擦除的单元中，则其变为一残留单元，并可随后由存储器系统控制器作为一共用区块打开。在所示实例中，通常将不使用可能性 (C)，因为在残留单元 2 或 5 中的一者中存在用于残留文件数据的空间。

另一选择为，可允许一文件的残留数据分裂成两个或更多个部分，以存储于不同的共用区块中。例如，现有的开放共用区块可能均不具有足以存储一特定文件的残留数据的可用空间，且没有被擦除区块可供打开作为新的共用区块来用于所述残留文件数据。在此种情形中，可在两个或更多个开放共用区块之间划分所述残留文件数据。

可在一起共用区块填满数据之前将其关闭。如上文所述，当需要打开另一共用区块

以存储一文件的给定量的残留数据时，一具有最少可用被擦除空间量的开放共用区块一般将关闭。这部分地是因不将文件的残留数据分开存储在不同共用区块中的优选操作所造成。当存储器系统通过响应于主机的大小（Size）命令（图 12E）而报告可供用于新数据的存储器存储空间时，不包含所关闭共用区块中这些较小量的未利用容量，因为其不能立即可用。

由于残留文件数据因其所属的文件被删除或擦除而变得过时，因而还合并共用区块内的数据。例如，每当无论出于何种原因而指定共用区块内的残留文件数据过时时，均将该区块加至上述过时文件数据区块队列或者另一队列中。如果数据因存在要删除其文件的主机删除命令而变得过时，则将所述共用区块置于队列末尾处。但是，如果其是因擦除命令而变得过时，则所述共用区块放到所述队列的顶部，或者以其他方式被赋予垃圾收集优先权。

可并非存在单个队列，而是在存储器系统的操作期间由控制器维持五个不同的垃圾收集队列，其中前两个队列中的项被赋予优先权：

- (1) 过时区块优先权队列，过时区块即那些作为对文件的擦除命令（图 12B）的结果而仅包含过时数据的区块；
- (2) 共用区块优先权队列，其包含因对文件的擦除命令而变过时的数据但还包含某些有效数据；
- (3) 过时区块（仅包含过时数据的区块）队列，所述过时区块是因执行更新或删除命令（图 12A 及 12B）而形成或者因在垃圾收集期间其所有有效数据均复制至另一区块而形成；
- (4) 共用区块队列，所述共用区块包含某些过时数据，但其还包含有效数据，响应于删除命令或者在垃圾收集期间发现在共用区块中存在过时数据；及
- (5) 已接收到对其发出的关闭命令或关闭_之后命令的文件队列。

可按上面所列次序赋予这五个队列优先权，队列（1）中的所有项均在队列（2）中的项之前进行垃圾收集，依此类推。之所以将所述区块列于优先权队列（1）及（2）中，是因为其各自的所有或某些数据均是通过擦除命令而非通过删除命令而变过时。各个区块及文件系以在存储器系统的操作期间所标识的次序添加至每一队列中，每一队列中最早的区块及文件均首先得到垃圾收集（先进先出，或 FIFO）。

队列（4）与（5）中所列项的优先权大致相同，因此可颠倒。另一选择为，可存在一更复杂的优先权系统，其中可能甚至在更高优先权队列中的一者或者变空之前，根据设定标准从队列（4）及（5）中选择文件及共用区块。存在两个共用区块垃圾收集管理目标。一个是在共用区块内的数据群组变得过时时，使可能因对共用区块的垃圾收集而引起的对存储器系统正常操作的破坏最小化。另一目的是当在对共用区块进行垃圾收集期间对所述区块的数据群组进行重新定位时使索引更新最小化。

在对共用区块进行垃圾收集期间，所述区块中剩余有效数据的所有共用群组每次一个地复制至具有空间的共用区块中的一者。若在打开的共用区块中不存在用于一所

复制共用群组的空间，则将所述共用群组从擦除池写入至一区块中，并可随后指定该区块作为一公用区块。然后，作为该过程的一部分，通常关闭其中一个打开的公用区块。如同所有垃圾收集操作一样，更新文件索引表（FIT），以反映所复制数据群组的新存储位置。

垃圾收集队列应记录于非易失性存储器中。应对包含队列信息的页或元页执行读取-修改-写入操作，以添加或移除一条目。包含垃圾收集队列信息的页可处于一专用区块或元块中，或者可与其他类型的页共享一区块或元块（例如一交换区块）。

在编程期间对元页的缓冲及使用

上文说明了存储器系统的操作，而未具体考虑存储单元区块是否作为元块链接于一起。主机与存储器系统之间基于文件的接口、及上述相关存储器系统操作是在使用元块的存储器系统中以及在不使用元块的存储器系统中工作。趋势必定是增大每次所写入及读取的数据量（平行度），因为这会直接改善存储器性能。各单独存储单元区块以数据位数量表示的有效宽度能通过使用由两个或更多个此种区块所形成的元块而得到增大。

参见图 3，例如，沿单个区块内每一字线的所有存储单元均可作为一页一同编程及读取，所述页在每一行中潜在地存储数个—一个、两个、四个或更多个由 512 个用户数据字节形成的扇区。且当将两个或更多个区块在逻辑上链接成一元块时，这两个或更多个区块中每一者的一行中的所有存储单元均可一同编程及读取。来自两个或更多个一同编程及读取的区块的这两个或更多个页形成一元页。由于具有每次编程许多数据的能力，对元页内的数据进行某种分级可有助于完全利用此种可用的平行性。

在图 18 中图解说明三个主机文件中每一者的一个数据元页，为简明起见，图中显示每一元页仅包含两个页。元页中每一页的数据均编程至元块中的一不同区块中。

对于主机文件 1，在图 18A 中显示所述元页填充有具有邻接偏移量（逻辑地址）的数据作为单个数据群组 0 的一部分。这些数据平行地编程至接收文件 1 的数据的元块内按次序的下一元页中。在图 18B 中，显示该实例的主机文件 2 在如下方面不同：其第一页的一部分包含具有连续偏移量的数据群组 1 的一部分，且所述元页的其余部分包含具有连续偏移量的另一数据群组 2 的一部分。尽管在文件 2 内这两个数据群组交汇之处，数据偏移量可能存在间断，然而这些数据一同编程至单个元页中。如前所述，直接文件存储接口是从主机接收到主机文件的数据时写入主机文件的数据，而无论所述数据在文件内的偏移量次序如何。

某些闪速存储器并不允许不止一次地将数据编程至被擦除页中。在此种情形中，同时编程图 18B 中的数据群组 1 及 2 二者。但是如果存储器允许进行局部页编程，则可将文件 2 的这两个数据群组在不同时刻编程至非易失性存储器的单个元页中。在此种情形中，页 0 将在不同时刻得到编程，首先以数据群组 1 进行编程，且然后将数据群组 2 编程入页 0 的其余部分以及整个页 1 中。然而，通常偏好对这两个数据群组平行编程，以便提供系统性能。

在图 18C 中，显示已将单个数据群组 3 的末尾写入文件 3 的元页中。存在某些如下情况：当将如此少量的数据编程入非易失性存储器的元页内时，该元页的其余部分仍保持被擦除。一种情况是当主机对其中数据群组 3 是残留文件的文件发出关闭命令（图 12B）时。如果主机试图打开超过所允许数量的文件数量且选择关闭文件 3，则存储器控制器也可关闭数据文件 3，以便允许改为打开另一更有效的文件。如果在缓冲存储器中没有足以用于已打开的所有文件元页缓冲器的容量，则存储器控制器也可关闭文件 3。在这些情况中的任一种中，均希望将文件缓冲器的局部数据内容立即写入至非易失性存储器中（“冲洗”文件缓冲器）。主机也可发送一关机命令（图 12D），此意味着存储器可能要断电且其易失性缓冲存储器内的所有数据如果未立即编程至非易失性存储器中则将丢失。

如果此后想要将文件 3 的另一数据群组 4 的开头写入至图 18C 的存储器元页，且存储器不允许进行局部页编程，则将数据群组 4 的第一页写入至文件 3 元页的第二页中，如在图 18D 中所示。此可导致非易失性存储器元页内的某些存储容量得不到使用，如图 18D 中数据群组 3 与 4 之间的空白部分所示。可通过在文件 3 关闭之后执行垃圾收集操作来恢复该未利用的容量，或者可允许其一直存在，因为此种情形可能不会频繁出现。

重要的是，应注意，本发明的直接文件存储技术可容忍非易失性存储器的区块内具有此种未填充的间隙。而如图 7 及 8 所示的其中存储器系统通过逻辑地址空间与主机进行接口的当前系统可能无法轻易地容忍这种间隙。与存储器区块或元块具有相同大小的逻辑数据群组映射至这些区块或元块内。假如在现有存储器系统的区块或元块内所存储的数据中存在一间隙，则映射至所述间隙内的逻辑数据地址将实际上不可供主机使用。在此种现有接口中，主机假定其具有整个逻辑地址空间可供其利用，但很难（即使有可能）将新数据写入标定该间隙的逻辑地址处。

系统控制器中缓冲存储器的一部分（例如图 2 中的存储器 31）通常用作编程数据缓冲器。对于主机所写入的每一现用文件，在该存储器中均应存在缓冲容量。每一此种“文件缓冲器”均应具有等于闪速存储器中至少一个元页的容量。现用文件是主机最近已向其写入数据的打开的文件。

存储器系统在任一时刻所处理的现用主机文件的数量可等于当前打开的存储器系统文件的数量，或者可为更少的数量。如果所允许的现用文件的最大数量小于所允许的打开文件的最大数量，则必须采取措施使文件状态在现用与打开之间改变，反之亦然。为能够这样，将闪速存储器中的一临时存储区块指定为一交换区块，并将长度不到一个元页的数据从文件缓冲器写入至交换缓冲器，反之亦然。有效交换文件缓冲器的索引保持在交换区块中。文件缓冲数据作为整数个页复制至交换区块，随后是提供每一所复制文件缓冲器的长度及位置的索引的单个页、以及与其相关的文件。周期性地压缩所述交换区块，并在其便满时将其写入至被擦除区块。

当例如因对打开的文件 A 进行主机写入操作而必须使打开的文件 A 现用时，识

别最早写入的现用文件 B，并将其文件缓冲数据从控制器缓冲存储器复制至交换区块中接下来的可用页中。然后，在交换区块中识别文件 A 的文件缓冲数据，并将其复制至控制器缓冲存储器中先前分配给文件 B 的可用空间。

较佳根据图 18 所示情形中的一种，将一文件缓冲器中的数据写入至闪速存储器中该文件的打开的写入区块中的下一可用元页。在图 18A 中，当在文件 1 缓冲器中存在形成文件 1 内单个数据群组 0 的一部分的足够数据时，将其在单次操作中编程至一完整元页。在图 18B 中，当在文件 2 缓冲器中存在形成文件 2 内数据群组 1 的末尾及数据群组 2 的开头的足够数据时，将其在单次操作中编程至一完整元页。如果闪速存储装置支持对单个页进行多次编程操作（局部页编程），则当在文件 2 缓冲器中存在形成文件 2 内数据群组 1 的末尾的足够数据时，可将其编程至页 0 的一部分。当在文件 2 缓冲器中具有形成文件 2 内数据群组 2 的开头的足够数据时，可在单独的编程操作中将其随后编程至同一元页的其余部分。

如在图 18C 中所示，当在文件 3 缓冲器中存在形成文件 3 内数据群组 3 的末尾的数据且必须执行缓冲器冲洗操作时，将其编程至页 0 的一部分。如果闪速存储装置不支持对单个页进行多次编程操作（局部页编程），则当在文件 3 缓冲器中存在形成文件 3 内数据群组 4 的开头的足够数据时，随后在单独的编程操作中将其编程至同一元页中的页 1。

在垃圾收集期间也使用元页文件缓冲器。可将文件的有效数据群组从非易失性存储器中读出并写入至该文件的控制器缓冲器元页中。如果同时正对所述文件进行重新排序，则将数据群组以其主机数据偏移量次序写入至缓冲器中。当然，每一数据群组中的数据均具有邻接的逻辑偏移量。一旦元页缓冲器变满，其数据便被平行地编程至非易失性存储器的新区块中。

文件加索引

存储于存储器系统中的每一文件均如上文参照图 13-16 所具体说明由其自身的索引条目序列来界定。当在其中附加、插入或更新文件数据时，以及当对文件进行垃圾收集时，这些条目会随时间发生变化。图 19 图解说明在几个不同时刻 0、2、4、及 5 中的每一者处，一个文件的文件索引表（FIT）中的索引条目序列。这些是在上文中分别参照图 14A、14C、14E 及 15 所述的序列。FIT 中的数据较佳由存储器控制器在不存在来自主机系统的协助的情况下写入并保持为当前数据。主机系统在将数据写入至存储器系统时提供路径名、文件名及数据在文件内的偏移量，但主机不参与界定数据群组或者将数据群组存储于存储单元阵列中的何处。在图 19 的条目中，图 14 及 15 的存储单元区块以 1 开始从左侧开始编号。因此，对于图 14C 中所示状态的文件，在图 19 中标注其第三数据群组（F6，D7）存储于区块 004（从左侧起第四个区块）中从该区块起始地址开始的 D7 个字节中。较佳还随所述表的每一条目一起包含每一数据群组的长度。

图 19 中针对一个文件所示的序列索引条目由存储器控制器在对文件的改动使文

件的数据群组得到修改时，或者以其他不太频繁的间隔进行重写。控制器可将这些改动存储于其存储器中，并随后将其中的许多同时写入至闪速存储器。在任一时刻一文件仅存在一组有效的索引；在图 19 中显示用于在不同时刻界定所述文件的四组此种索引。存储器系统控制器随后根据需要使用所述文件的当前一组索引将额外数据编程至所述文件，从所述文件读取数据，对所述文件的数据进行垃圾收集，及可能进行其他操作。因此，如果各单独文件索引条目以其文件偏移量 (Fx) 次序进行存储，则 FIT 会更易于使用，但是如果不是，则控制器可当然地以该逻辑次序读取所述条目。存储器控制器读取特定文件的索引条目的最常见原因是在执行主机命令的过程中。

图 20 显示一种维持及使用文件索引表 (FIT) 作为一文件映射表一部分的较佳技术。一文件加索引结构链用于使用一规定路径名识别具有规定偏移地址的数据在文件内的物理位置。文件映射表包含一目录 201，目录 201 具有实质与在传统逻辑地址接口中所用的标准磁盘操作系统 (DOS) 根目录及子目录结构相同的逻辑结构。文件目录 201 可存储于存储器系统内的一个或多个专用闪速区块中。但文件目录 201 较佳由存储系统而非由主机来管理。本文中图 12C 所示的各主机目录命令是由存储器系统控制器执行，但以控制器所决定的时刻及方式执行。较佳不允许主机直接写入至文件目录。

在文件目录 201 中保持若干大小均匀的条目，每一条目均标识一目录或文件。使用一组邻接的条目来规定一特定命令内的各要素。一规定一目录的条目中的指针字段标识用于规定该目录内各要素的其他邻接条目的开头。一规定一文件的条目中的指针字段界定一相关联文件索引表 (FIT) 203 内的区块数量及文件数量。

FIT 203 包含数据群组的大小均匀的条目，每一条目均标识在数据群组的存储器系统内所述数据群组的开头的偏移量及物理位置。如果以字节粒度保持逻辑地址，则每一 FIT 条目的偏移量均包含从指定数据群组开头的文件起始处算起的规定数量的字节。类似地，可使用字节粒度来规定数据群组起始处的物理位置。上文参照图 19 说明了一实例性文件在不同时刻的 FIT 条目的内容。为每一数据群组维持一单独的条目。每一文件皆由文件内各数据群组的一组邻接的索引条目来界定。每一文件的这些条目的数量将通常各异。

主机为文件提供的路径名用于穿过文件目录的层次结构，以获得在 FIT 内的区块数量及文件数量。路径名通常包括一个、两个或更多个目录及子目录。这些目录及子目录用于存取文件目录 201 内包含一试图要存取的文件的目录。然后，搜索该目录内的文件名，以找到由主机所提供的文件名的一条目 205。当找到时，条目 205 提供一指向该文件在 FIT 203 中的一邻接条目群组的指针。这些条目是参照图 19 所述的条目性质。然后，将这些条目的偏移量与主机所提供的偏移地址相比较，以识别正确的条目 207，并由此识别正确的数据群组。如果不存在相同的偏移量匹配项（由于包含于条目中的偏移量仅仅是数据群组的起始地址，因而可能常常如此），则选择用于标识一在其中包含该偏移量的数据群组的条目。在该实例中，FIT 203 的所选条目 207 包含含

有主机试图存取的数据的存储位置的物理区块及字节。

图 20 中文件的目录条目 205 的指针是在所述文件首次创建时或者在所述文件的数据群组条目在 FIT 203 中的位置发生变化时由存储器系统控制器指配。该指针是前面所述文件_句柄的一实例。为避免每次存取文件时主机必须遍历文件目录 201 的层次结构才能找到文件的 FIT 目录，存储器系统可将文件_句柄发送至主机，以使其可此后直接存取打开的文件的 FIT 条目。

图 21 图解说明将各单独条目存储于文件目录 201 中的存储器系统内的各页。某些条目提供指向文件目录 201 内的目录的指针，而其他条目提供指向 FIT 203 内的数据的指针。一实例性条目 209—其为存储于单个存储器页中的诸多条目中的一个—图解说明每一条目均具有四个字段。第一字段包含由主机指配的目录或文件的名称。第二字段包含主机所定义的目录或文件的属性。在为一目录的条目的情形中，第三字段中的指针指向文件目录中的另一条目。在为文件的条目的情形中，第三字段包含指向 FIT 203 的文件条目（例如条目 207）的指针。在目录的条目中，第四字段（标识为“数据群组”）是空的，但在为文件的条目时，该字段规定包含文件数据的数据群组的数量且因而规定所述文件在 FIT 203 中的条目数量。

应注意，图 21 中所示的文件目录页包含两个区段。在最近写入的页 211 中，同时存在目录条目（一个示于页 213 中的 209 处）及页指针。在其他目录页（例如页 213）中，存在目录条目，但页的一区域包含过时页指针。当前页指针维持于最近写入的页（在本实例中为页 211）的同一部分中。目录区块中的每一逻辑页存在一个页指针，其将存储器控制器指引至对应于其正存取的逻辑页的物理页。通过将页指针维持于最近写入的目录页中，页指针便与目录条目同时得到更新，而无需随后更新另一页。在 Gorobets 等人于 2004 年 8 月 13 日提出申请的第 10/917,725 号美国专利申请案中针对逻辑地址类型的文件系统对此种技术进行了更详细说明。

在特定实施方案中，目录区块包含固定数量的逻辑页，该数量是存储单元区块中总页数的一指定比例（例如 50%）。目录区块较佳是专用于存储目录页的元块，但也可为单个擦除区块。其可与用于存储数据群组的元块具有相同的平行性，或者其可具有降低的平行性。当文件目录区块变满时，在擦除原始区块之前，通过将每一逻辑页的有效版本复制至新的区块来压缩所述文件目录区块。在已压缩一目录区块之后，立即以目录条目写入新复制区块中一规定比例（例如 50%）的页。其余页则处于被擦除状态，以允许写入经过更新的或新的条目。

当创建、修改或删除目录或文件的条目时，将所述目录的包含该目录或文件的该组条目重写于目录区块的下一可用被擦除页中，从而使目录的条目保持邻接。这是通过对先前包含该目录的条目的页进行读取/修改/写入操作来完成的。所述先前的页随后变得过时。还对该逻辑页的页指针条目进行更新，以标识其新物理页。此可在与在写入目录条目时所用的相同页编程操作中完成。

如果创建一条目将使目录的该组条目溢出一页，则可转而将其作为另一页的第一

条目来写入。如果需要，也可重写现有的页，以将目录的该组条目移动至所述页的末尾。可在页指针中重新指配目录区块内的逻辑页编号，以使目录的条目保持邻接。

图 20 的文件索引表 (FIT) 203 存储构成所述装置中所有文件的数据群组的索引。所述 FIT 存储于存储器系统的一个或多个专用 FIT 区块中。每一 FIT 区块可存储多达一最大数量的文件的索引。FIT 是通过来自文件目录的逻辑指针进行存取，所述逻辑指针规定由该表加索引的其中一个文件。所述指针使用间接寻址（通过存取作为最近写入的 FIT 页 217 的一部分而提供的文件指针），以使其不会在更新索引并将索引重写入 FIT 区块内时发生改变。

文件目录 201 中的文件条目（例如条目 205）的 FIT 指针具有两个主要字段。第一字段是 FIT 区块编号，其标识构成 FIT 的其中一个逻辑 FIT 区块。对分配给 FIT 区块编号的实际物理区块地址进行单独管理。FIT 指针的第二字段是 FIT 文件编号，其标识所标识 FIT 区块内的逻辑文件编号。所述逻辑文件编号由存储于 FIT 区块的最近写入的页内的特定文件指针转换成物理文件条目位置。

FIT 不具有预定大小，且 FIT 区块的数量是文件数量以及数据所组织成的数据群组的数量的函数。在装置运行期间，将创建新的 FIT 区块并消除 FIT 区块。

参见图 22，各单独 FIT 页具有若干个数据群组条目，每一数据群组一个条目 219，如前面参照图 19 所述。在该特定实例中，每一条目 219 包含四个字段。文件偏移量字段规定由所述条目所标识的数据群组的开头在所述文件内的偏移地址。区块地址字段规定区块或元块在包含所述数据群组的存储器系统内的物理地址。字节地址字段规定所述页在区块或元块内的地址、及所述页内所述数据群组开始处的字节。第四字段是数据群组的长度。数据群组长度可能并非在所有情形中均需要，因为可根据相邻条目的数据来计算数据群组的长度。然而，通过保持有所述长度，不必每当需要数据群组的长度时均进行此种计算。如果各数据群组条目未以其逻辑地址次序写入于 FIT 中，则此会成为特别有价值的信息—此时所述计算变得更加困难。

主机文件较佳由一组邻接的数据群组条目界定，所述一组邻接的数据群组条目一同界定用于形成所述文件的有序数据群组。图 22 的页 215 中所示的阴影邻接群组条目界定一个文件，且页 217 中的那些群组条目界定另一文件。对于 FIT 中的每一单独文件，在最后所写入的 FIT 页 217 中均存在一个文件指针 221，以用于保持于该区块中的各单独文件。文件指针 221 界定一具有特定 FIT 文件编号的文件的文件条目在 FIT 区块内的开头。其包含两个字段。一个字段是其中驻留有数据群组索引条目的页的物理编号，而另一字段是第一群组条目在该页内的编号。对于 FIT 区块中的每一可能的文件编号，均存在一文件指针，即文件指针的数量等于 FIT 区块中的最大文件数量。文件指针条目中的一预留代码（未显示）指示未在 FIT 区块中得到使用的特定文件编号。尽管文件指针可存在于其他 FIT 页中，然而其仅在 FIT 区块中最近写入的页中有效。

FIT 区块较佳是一专用于存储 FIT 页的元块。其可具有与用于存储数据群组的元

块相同的平行性，或者其可具有降低的平行性。也可改为使用单个擦除区块。一旦 FIT 区块已得到压缩，所述区块中便应仅有一规定比例（例如 50%）的页包含条目。其余页应处于被擦除状态，以允许写入经过更新的或新的条目。当维持有大量的文件时，可能需要不止一个 FIT 区块。

通过将一个或多个完整文件的数据群组条目写入于 FIT 区块的下一可用未编程页来更新 FIT。文件的文件指针条目也得到更新，以标识文件条目的新位置。文件数据群组条目与文件指针二者是在同一页编程操作中写入。文件条目的先前位置随后在 FIT 区块中变得过时。另一选择为，可通过将文件条目写入于一不同的 FIT 区块或新的 FIT 区块中来更新文件条目。在此种情形中，这两个区块中的文件指针均应更新，且应修改文件目录中的文件逻辑指针。

当一 FIT 区块变满时，将有效群组条目以压缩形式复制至新的被擦除区块，并擦除先前的 FIT 区块。该操作不改变逻辑 FIT 区块编号及逻辑文件编号。应对数据群组条目的数量加以限制，以便在压缩后的 FIT 区块中仅有一规定比例（例如百分之五十）的页得到编程。如果需要，应将文件条目移至其他 FIT 区块，且应对其在文件目录中的逻辑指针进行修改。

希望将一单独文件的所有 FIT 条目保存于一个页或元页中。此使得相对易于读取文件的所有条目。尽管可通过在文件的每一 FIT 条目中包含下一 FIT 条目的物理地址来将各 FIT 条目链接于一起，然而此将有可能需要读取不止一个 FIT 页或元页。还希望使各 FIT 条目在所述页内保持邻接。此可导致当 FIT 条目数量增多时频繁地写入新的页，当打开一关闭的文件并对其添加数据群组时尤其如此。当将新的 FIT 条目写入新的页中时，现有的条目是从另一页进行复制并随同新条目一起写入新的页中。恢复现有条目的先前页中过时数据所占据的空间以在压缩 FIT 区块时使用。

如上文所述，各单独数据群组包含于单个区块或元块内，可在所述区块内的任一字节边界上开始，并可具有等于任意整数个字节的长度。可在将每一数据群组写入存储器中时由控制器对其附加一报头。此一报头可包含两个字段。第一字段包含用于标识数据群组开头的代码。该代码对于所有数据群组可均相同，且被选取为位图案，所述位图案不常存在于作为所述群组一部分加以存储的数据中，但并非所述代码必定永远不出现于此种数据中。可通过如下方式找到数据群组的开头：由存储器控制器扫描为该代码所存储的数据，然后当其找到所述代码的位图案时，确认其确实是数据群组的开头而非群组内的数据。报头的第二字段使得能够进行该确认。该第二字段是一指针，其指向包含所述数据群组的文件在 FIT 203（图 20 及 22）中的文件条目。第二字段界定 FIT 区块编号及 FIT 文件条目。控制器随后读取 FIT 文件条目，以查看其是否指向至从中读出所述代码的数据群组。如果是，则确认所述代码是对其处于数据群组报头内的指示。如果不是，则得知所述代码的位图案已从其他数据读出，因而被作为数据群组报头忽略。

通过包含此一报头作为存储于各单独数据群组中的数据的一部分，存储器中任一

数据群组所属的文件均可通过如下方式加以确定：读取其报头并存取所述报头所指向的 FIT 条目。FIT 条目 219（图 22）可包含文件 ID。此种能力在本文中例如用于在对共用区块进行垃圾收集期间标识共用区块中的数据群组。如果因为共用区块内的一个或多个其他数据群组过时而开始垃圾收集，则希望能够标识每一其余有效数据群组所属的文件。

可并非使报头作为每一数据群组的一部分，而是可每一文件提供一个报头。当已知文件中数据群组的 FIT 条目的位置时，较佳能够标识文件的完整路径名，且文件报头使此成为可能。可提供此一报头作为 FIT 中文件的该组条目中的第一个，或者文件目录中目录的一组条目中的第一个。

该文件报头包含一用于将其标识为 FIT 文件报头的代码、及一反向指针，所述反向指针指向指向它的目录条目。FIT 文件报头也可包含其他信息，例如文件长度及数据群组的数量。另一选择为，报头条目可占据一个或多个正常条目的空间。

类似地，目录报头条目包含一用于将其标识为目录报头的代码、以及一反向指针，所述反向指针指向指向它的目录条目。根目录是在其报头中以显式方式标识。目录报头也可包含其他信息，例如目录中的条目数量。

作为对使用此种数据群组或文件报头的替代，可在每一共用区块的末尾写入一索引，以标识共用区块中每一数据群组的相关文件。当关闭共用区块时，可写入此一索引。所述索引较佳包含所述共用区块内每一数据群组在该区块中开始的物理字节地址、及一指向该数据群组的 FIT 条目的指针。然后，可参照在每一数据群组的索引条目中所提供的 FIT 条目来确定每一共用区块数据群组作为其一部分的文件。

如果希望使存储于存储器中的数据群组具有某种文件标识冗余度，则即使采用这些共用区块索引，仍可保留上述数据群组报头。

由于存储装置管理文件目录及文件索引，因此其可控制报告给主机的装置配置参数。而当如在现有商业系统中一样由主机管理装置的文件系统时，这通常是不可能的。存储装置可改变所报告的存储容量，例如总装置的容量以及可用的未写入容量二者。

在直接文件接口处对文件所使用的路径名可标识存储装置自身的 ID 以及存储装置内的分区。允许存储装置修改分区的大小及数量可以是一优点所在。如果一分区变满，则可将来自一个或多个其他分区的未使用容量重新指配给满的分区。类似地，如果创建新的分区，则可指配来自其他分区的未使用容量。该装置可如上文所述修改各分区的所报告容量。

图 20 及 21 中所示的目录结构使用目录条目中的指针来识别该目录内各要素的一组其他邻接目录的开头。该指针指向目录区块内的逻辑页，且当对应于该逻辑页的物理页改变时仍保持不变。然而，目录内的要素数量会频繁地改变，且具体目录的该组条目将频繁地需要从一个逻辑页移动至另一个逻辑页，或者需要在其当前逻辑页内移动。此可导致频繁地需要更新目录区块内的指针参考项。在图 23 中显示一种用于替代图 21 所示技术的目录结构条目加索引技术，其中其对应要素是以相同的参考编号加以

标识但增加一撇号(')。

图 23 所示结构与图 22 所示 FIT 的结构相同，但具有与目录相关的不同术语。具体条目并非如在图 21 中一样仅指向一页。此提供一种比上文参照图 21 所述方法更有效地在目录区块内对规定目录的一组条目进行间接寻址的方法。对目录(图 23)及 FIT(图 22)区块使用同一加索引方案将较为恰当，因为二者具有非常相似的特性及要求。其均存储与目录或文件相关的成组的连续条目。对一组条目的更新可由改变现有条目的内容、或者改变条目数量组成。

参照图 22 所述的用于更新 FIT 条目的方法会在对物理区块进行压缩之后使所述区块中包含所述 FIT 条目的一定比例(例如 50%)的页处于被擦除状态。此然后便允许将更新后的 FIT 条目写入至被压缩区块的其余部分内。所有 FIT 区块均包含此种容量开销，甚至当没有由区块加索引的文件实际打开时也如此。因此，FIT 占用比所需存储容量更大的存储容量。

一种替代的 FIT 更新方法是使更新后的各组 FIT 条目写入于单独的 FIT 更新区块中，而非写入于其原先所处的区块中的可用被擦除容量中。图 24 及 25 分别概述利用更新区块的文件目录与 FIT 加索引技术。这些技术是相同的，只是目录与 FIT 的术语存在差别。

下文说明涉及到更新如图 25 中所示 FIT 区块中的群组条目，但同等地适用于更新文件目录中的群组条目(图 24)。

从 FIT 区块中读取群组条目如上文所述。FIT 指针(FIT Pointer)的 FIT 区块编号(FIT Block No)字段界定逻辑 FIT 区块。FIT 区块列表(FIT Block List)包含于闪速存储器中的数据结构中，并提供用于将 FIT 区块编号转换成其所处于的区块的物理地址。每当在压缩或合并操作中移动 FIT 区块时，均更新 FIT 区块列表中的区块地址。

该区块中最近写入的页的文件指针(File Pointers)区域允许将 FIT 指针中的 FIT 文件编号(FIT File No)值转换成文件指针，所述文件指针指明所规定文件的该组文件群组条目(File Group Entries)的起点。随后可从 FIT 区块读取所述文件群组条目。

当更新群组条目的内容、或者文件的该组群组条目中的群组条目数量时，会将完整的该组条目重写于一被擦除页中。(这是基于如下假定：页是闪速存储器中的最小编程单位，在各次擦除操作之间禁止对同一页进行多次写入操作。) 该组可占据多个页(如果需要)。

在当前所述的技术中，该页是 FIT 区块中下一可用的页。在经过修订的方案中，该页是单独 FIT 更新区块中下一可用的页。FIT 更新区块具有与 FIT 区块相同的页结构，如在图 23 中所示。其存在是由在 FIT 更新区块列表(FIT Update Block List)中存在目标 FIT 区块编号来标识，所述 FIT 更新区块列表还编号更新区块的物理区块地址及对应于原始 FIT 文件编号(FIT File No)的更新文件编号(Update File No)。每一所更新的 FIT 区块均可存在单独的 FIT 更新区块，或者较佳地，一 FIT 更新区块可与多个 FIT 区块相关。单个 FIT 区块也可与多个 FIT 更新区块相关。

当 FIT 更新区块变满时，其有效数据可以压缩形式写入至被擦除区块，所述被擦除区块变为新的 FIT 更新区块。如果更新仅与几个文件相关，则可存在少至单页的有效数据。可将多个 FIT 更新区块一同压缩至单个区块。如果所述更新区块与仍打开的可能继续进行更新的一个或多个文件相关，则区块压缩优于区块合并。

当在 FIT 更新区块中更新文件的群组条目时，原始 FIT 区块中该文件的条目变得过时。在某一阶段中，原始 FIT 区块必须经历垃圾收集操作，以对其进行整理。此可通过将 FIT 区块与 FIT 更新区块中的有效数据合并至被擦除区块中来完成。

如果条目数量在更新过程中已增加，且有效数据无法合并至单个被擦除区块中，则原先指配给该 FIT 区块的文件可重新指配给两个或更多个 FIT 区块，且可对两个或更多个区块执行合并。

来自 FIT 更新区块的条目可与来自 FIT 区块的条目合并，且因此从 FIT 更新区块中消除，而其他文件的条目则可保留在 FIT 更新区块中。

作为另一选择，可将目录区块与 FIT 区块结构并入单个索引区块（Index Block）结构中，所述单个索引区块结构可包含目录条目与文件群组条目二者。当存在单独的目录区块与 FIT 区块结构时，或者当存在一组合式索引区块结构时，一索引更新区块（Index Update Block）可为目录条目及文件群组条目充当更新区块。

应认识到，上文参照图 20-25 所述的文件目录及 FIT 是在 DOS 系统上进行建模。另一选择为，其可在 Linux、Unix、NT 文件系统（NTFS）或某种其他已知的操作系统上进行建模。

具体存储器系统操作实例

图 26-32 所示的操作流程图提供如上文参照图 2-6 所述、但使用参照图 9 及图 10-22 所述的直接文件接口技术的具体组合加以构造的存储器系统的操作的实例。包含于图 26-32 的流程图中的功能主要由执行其所存储固件的控制器 11（图 2）来实施。

首先参见图 26，其显示一总体系统操作。在第一步骤 251 中，将存储器系统初始化，此包括处理器 27（图 2）执行 ROM 29 中的引导代码来将非易失性存储器中的固件装入 RAM 31 中。在初始化之后，在该固件的控制下，存储器系统随后查找来自主机系统的命令，如步骤 253 所示。如果主机命令待决，则从在图 12 中所列出的那些命令中识别所述命令。如果为读取命令（图 12A），则在步骤 255 中识别出该命令并由一在 257 处所指示的过程来执行，所述过程将在下文中参照图 27 进行更全面说明。如果不是读取命令，则在图 26 的步骤 259 中识别出图 12A 所示写入、插入或更新编程命令中的任一者，并由一作为图 28 的显示主题的过程 261 来执行。如果是删除或擦除命令（图 12B），则在步骤 262 中识别出该命令并在步骤 263 中执行，如在图 30 中所更详细说明。在图 26 的 264 处识别出来自主机的空闲命令（图 12D），且该命令引起在图 31 的流程图中所示的垃圾收集操作 265。图 26-32 所示的该实例是针对如前面所述以元页及元块进行操作的存储器系统所述，但也可由组织成页及/或区块形式的存储器系统执行。

如果主机命令并非读取、写入、插入、更新、删除、擦除或空闲命令，则在该具体实例中，由图 26 的步骤 267 来执行此种另一命令。在这些其他命令中，有那些使得将垃圾收集操作添加至一队列的命令，例如在图 12B 中所列且在上文所述的关闭命令及关闭_后命令。在由步骤 257、261、263、265 或 267 中的任一步骤执行所接收命令之后，下一步骤 268 询问优先权垃圾收集队列是否是空的。如果是，则所述处理返回至步骤 253 来执行待决的主机命令（如果存在）。如果不是，则所述处理返回至步骤 265，以继续进行垃圾收集，而非允许执行另一主机命令。对于参照图 26-32 的流程图所述的具体实例，存在上文所述的五个不同的垃圾收集队列：两个优先权队列，其用于过时元块（仅具有过时数据的元块）及具有过时数据的共用元块，其中所述过时数据是因擦除命令而产生；两个其他队列，其用于过时元块及具有过时数据的共用元块，其中所述过时数据是因执行除擦除命令之外的命令而产生；以及一个使得对文件进行垃圾收集的队列。对于在这三个非优先权队列中所列的垃圾收集，赋予另一待决的主机命令优先权来执行所列的垃圾收集操作。然而，对于优先权队列中的垃圾收集，则赋予垃圾收集比执行新主机命令高的优先权。也就是说，可使主机等待至完成任何垃圾收集操作后才可执行新的主机命令。这是因为主机此前已将优先权赋予使用擦除命令来擦除优先权队列中的元块中的数据。优先权垃圾收集还使得以相对很短的处理时间来产生额外的被擦除元块。但是如果无优先权，则当主机空闲时在后台执行垃圾收集，或者当需要维持被擦除区块池时与其他操作相交织。如果步骤 268 确定出不存在待决的优先权垃圾收集操作，则下一步骤随后是步骤 253，在该步骤中执行新的主机命令（如果有）。

返回图 26 中的步骤 253，如果没有待决的主机命令，则执行垃圾收集 265，包括非优先权垃圾收集—如果步骤 269 确定出主机不活动或空闲已达一预定时间长度、或者最近接收到的主机命令是空闲命令，或者步骤 264 确定出待决的主机命令是空闲命令。在这些情况下，随后最可能完全在后台执行垃圾收集。

图 27 所示流程图的主题是在图 26 的步骤 257 中执行读取命令。第一步骤 271 是读取文件索引表 (FIT)，如上文参照图 19 及 22 所述。主机读取命令包括文件 ID 及文件内要开始读取处的偏移量（参见图 12A）。要读取的文件的所有 FIT 条目或者其某些部分较佳从非易失性存储器读入控制器存储器 31（图 2）中，以避免每次需要非易失性存储器中的某些数据时均需要从非易失性存储器中读取 FIT。在步骤 273 中，将数据群组编号计数器初始化为构成起始偏移量所在的所请求文件的数据群组的编号。这是通过将主机所规定的起始偏移量与主机所指定文件的 FIT 条目中各数据群组的偏移量进行比较来完成。接下来在步骤 275 中，将数据群组长度计数器初始化成从主机所提供偏移量至数据群组末尾的初始数据群组内的数据量。每次读取一个数据群组，且步骤 273 及 275 设置两个用于管理对第一数据群组的读取的计数器。所要读取的数据在非易失性存储器内的起始物理字节位置是根据初始数据群组的 FIT 条目加以确定。数据在群组内的逻辑偏移量与物理字节地址是线性相关的，因此如果读取不是在数据

群组的起始处开始，则根据主机所提供的在文件内的起始偏移量来计算开头字节地址。另一选择为，为简化数据群组长度计数器的初始化，可将每一数据群组的长度添加至 FIT 中的其记录 219 中（图 22）。

在对各流程图的本说明的其余部分中，假定存储器系统是以元块进行操作，且数据是以元页为单位进行读取及编程，如前面所述。在步骤 277 中从存储于非易失性存储器中的包含起始字节地址的初始数据群组中读取数据元页。所读取数据通常随后写入至控制器缓冲存储器（例如图 2 中的 RAM 31）中，以便传送至主机。然后，在步骤 279 中将数据群组长度计数器递减所述一个元页。然后在步骤 281 中读取该计数器，以判定其是否已达到 0。如果没有，则还有要读取的所述初始数据群组的数据。但是在返回至步骤 277 以按次序读取下一数据元页之前，步骤 283 检查主机是否已发出另一命令。如果是，则终止读取操作且该过程返回至图 26 的步骤 253，以识别所接收的命令并随后执行所述命令。如果不是，则通过在图 27 的步骤 277 及 279 中读取其下一页来继续读取所述初始数据群组。此会继续进行到通过步骤 281 确定出数据群组长度计数器已到达 0 为止。

在发生此种情况时，在步骤 285 中再次读取文件的 FIT 条目，以在步骤 287 中确定当前文件中是否存在要读取的其他数据群组。如果是，则在步骤 289 中更新数据群组编号计数器，以标识下一数据群组，且在步骤 291 中将数据群组长度计数器初始化成新群组中的数据长度。然后，如果通过步骤 283 确定出不存在待决的其他主机命令，则在步骤 277 中读取新数据群组的第一元页。然后，对新数据群组的每一元页重复步骤 277 及 279，直到其所有数据均被读取为止，此时，步骤 285 及 287 判定是否仍存在另一数据群组，依此类推。当通过步骤 287 判断出所述文件中处于主机所提供偏移量之后的所有数据群组均已得到读取时，所述处理返回至图 26 的步骤 253 来执行另一主机命令。

在其中使用一文件作为循环缓冲器的特殊情形中，可在图 27 的步骤 287 之后重复对文件的读取，而非返回至图 26 的步骤 253。在读取期间在步骤 283 中主机响应于将数据写入至当前文件的主机命令而正对同一文件的数据进行编程的情况下，即能出现此种情形。

在图 28 的流程图中给出图 26 的数据编程操作 261 的一实例。当从主机接收到图 12A 所示的其中一个数据编程命令时，该命令包含数据所要写入的文件的文件 ID。第一步骤 295 判定该指定文件是否当前正打开以进行编程。如果是，则下一步骤 297 判定在控制器缓冲器（例如图 2 中的 RAM 31）中是否存在该文件的数据。数据是由主机系统传送至控制器缓冲存储器中，并然后由存储器系统控制器传送至闪速存储器中。

但是，如果通过步骤 295 确定出主机所指定的文件未打开，则下一步骤 299 询问存储器系统当前所打开的进行编程的文件数量是否等于或大于存储器系统允许同时打开的最大数量 N1。数量 N1 预设于存储器系统中，并可为 5 个、8 个或某一其他数量的文件。如果所打开文件的数量小于 N1，则下一步骤 301 通过提供为将数据编程至新

文件所需的系统资源而打开新文件，且所述处理进行至步骤 297。然而，如果在步骤 299 中确定出打开的文件数量等于或大于 N1，则需要首先关闭一当前打开的文件，如步骤 303 所指示，才能在步骤 301 中打开新文件。在步骤 303 中选择关闭一打开的文件的依据可各不相同，但最常见的将是主机最近写入数据最少的所打开的文件。据此假定主机在近期不可能向该文件写入数据。但是如果在近期要向该文件写入数据，则在另一打开的文件关闭之后重新打开该文件（如果此时需要如此）。

当在步骤 297 中确定出当前文件的至少一数据元页处于控制器缓冲器中时，则下一步骤 305 判定存储器阵列内的元块是否已打开以进行编程。如果是，则随后在步骤 307 中将数据从控制器缓冲存储器编程至打开的元块中。如果不是，则在步骤 308 中通过提供为将数据编程至该元块中所需的系统资源而首先打开该元块。在该实例中，每次以一个元页为单位将数据写入至打开的存储器元块中。一旦写入该数据单位，便在下一步骤 309 中判定所打开的写入元块是否装满数据。如果没有，则所述过程通常经过步骤 311 及 313 而回到步骤 297，以重复所述过程以将下一数据元页编程于当前打开的文件中。

然而，如果通过步骤 309 确定出所述写入元块已装满，则在步骤 315 中关闭该元块，且在步骤 317 中更新 FIT，包括关闭当前的数据群组，因为已到达存储器元块边界。然后，在步骤 318 中更新较低优先权过时元块垃圾收集队列的元块条目。在步骤 317 中进行 FIT 更新期间，判定对所述写入元块的填充是否已创建了另一包含当前文件的所有过时数据的元块或者是一包含当前文件的过时数据的共用元块。如果是，则在步骤 318 中将该元块加至一适当的较低优先权元块队列，以进行垃圾收集。然后，所述处理返回至步骤 311 并经过步骤 313 回到步骤 297。此次经过步骤 305 及 307，步骤 308 将打开新的写入元块，因为前一元块刚刚通过步骤 315 关闭。

在已通过一包含步骤 307 的路径写入每一数据元页之后，步骤 311 均询问当前存在于被擦除元块池中的元块数量是否超过所已确定出的有效操作存储器系统所需要的最小数量 N2。如果是，则步骤 313 询问是否已接收到另一主机命令。如果没有其他待决的主机命令，则重复步骤 297，以将下一数据元页编程至存储器内。但是，如果已接收到主机命令，则在步骤 319 中更新 FIT，以关闭已写入的数据群组。当在步骤 320（类似于上文所述的步骤 318）中更新处于较低优先权过时元块垃圾收集队列中的元块条目之后，所述过程返回至图 26 的步骤 253。

但是，如果在图 28 的步骤 311 中确定出被擦除元块不足以存储数据（等于或少于预设数量 N2），则执行一子例程以在后台中对文件或共用元块进行垃圾收集，以便增大被擦除元块的数量。此种垃圾收集较佳并非在将每一数据元页写入存储器中之后均执行，而是仅在每当已写入 N3 个元页之后执行。一写入元块元页编程计数器保持对已接连编程而在其间未进行任何垃圾收集的主机数据元页的数量的计数值。每当执行垃圾收集时，该计数器均复位至 0，并此后每当对一数据元页进行编程时进行递增。在步骤 321 中，判定该计数值是否超过预定数量 N3。如果未超过，则在步骤 323 中将

计数器递增 1，以记录一元页在步骤 307 中写入至存储器中。但是如果编程计数器的计数值超过数量 N3，则通过步骤 325、327 及 329 进行垃圾收集，随后在步骤 331 中将编程计数器复位至 0。

在该过程中的此时进行垃圾收集的目的是为被擦除元块持形成额外的被擦除元块。但较佳每次通过执行步骤 325、327 及 329 仅执行垃圾收集操作的一部分。所述复制操作划分成若干在时间上散布的较小操作，以使存储器不能供主机使用的间隔的持续时间最小化，从而使对数据编程性能的影响最小化。且出于同一原因，仅每 N3 个编程循环执行一次局部垃圾收集操作。尽管数量 N2 及 N3 可预设于系统中，然而另一选择为，其也可在存储器系统的运行期间加以确定，以适应于可能遇到的特定情况。

由于对文件或共用元块进行完整垃圾收集操作以提供一个或多个额外被擦除元块所需的大部分时间是由将有效数据复制至一个或多个复制元块中所耗用，因而在图 28 中与数据编程相交织的主要时此种复制操作。步骤 325 选择前台模式进行垃圾收集 329，这参照图 31 的流程图进行说明。然后，将数据群组中某一预设数量的元页从进行垃圾收集的元块中接连复制至先前被擦除的复制元块中。步骤 327 将以此种交织方式复制的元页的计数器复位，以便在步骤 329 中，在所述操作从垃圾收集步骤返回至数据编程循环之前复制预设数量的元页。然后，在每一垃圾收集操作 329 之后，通过步骤 331 将步骤 321 所引用的写入元块元页编程计数器复位。此使得在前台中进行另一垃圾收集操作之前，将 N3 个数据元页写入至存储器中，且从而使因垃圾收集所引起的任何数据编程延迟分散。

该垃圾收集-编程算法的交织性质由图 29 中的时间线显示。在各次垃圾收集操作之间，来自主机的 N3 个数据元页连续写入至存储器中。每一垃圾收集操作被限定为复制 N4 个数据元页，此后进行另外 N3 个数据元页的编程。在处于垃圾收集阶段中的同时，可使主机等待存储器系统完成垃圾收集后再将额外的数据元页传送至存储器系统控制器缓冲器。

图 26 中删除文件例程 263 的执行是由图 30 的流程图进行显示。其主要目的是识别具有所删除文件的过时数据的过时区块及共用区块，并随后将这些区块放置于适当的垃圾收集队列中。当删除或擦除命令与所要删除的文件的名称或其他标识一同由存储器系统接收到时，步骤 335 将数据群组编号计数器初始化至 0。在步骤 337 中，读取文件目录及文件索引表 (FIT)，以获得存储于存储器中的构成所指定文件的数据群组的身份。然后，通过递增指向逻辑序列中每一数据群组的数据群组编号计数器而每次检查所述文件的每一数据群组。

对数据群组的第一询问 339 是其是否位于共用元块中。如果是，则步骤 341 将该元块添加至共用元块垃圾收集队列中。如果所述文件是通过擦除命令删除，则将所述元块置于优先权共用元块队列中，而如果是通过删除命令删除，则将所述元块置于另一共用元块队列中。对具有所要删除的文件的数据群组的任何共用元块进行调度以进行垃圾收集。如果通过步骤 339 确定出数据群组不处于共用元块中，则下一询问 343

判定所述数据群组是否因处于过时元块垃圾收集队列中而处于一早已被调度进行垃圾收集的元块中。如果所述元块早已被调度进行垃圾收集，则不应再将其添加至同一队列中。但是，如果尚未得到调度，则通过步骤 345 将其添加至其中一个过时元块垃圾收集队列中。如果文件是通过擦除命令被删除，则将元块置于优先权过时元块队列中，而如果是通过删除命令被删除，则将其置于另一过时元块队列中。

在已执行步骤 341 或 345 之后，或者在步骤 343 中的询问得到肯定结果时，对于一个数据群组的过程即告结束。在下一步骤 347 中递增数据群组编号计数器。然后，进行关于在文件中是否存在另一数据群组的询问 349。如果有，则所述处理返回至步骤 339，并对下一数据群组进行重复。如果没有，则得知所有包含所要删除文件的数据的元块均已输入过时及共用元块垃圾收集队列中。然后，在步骤 351 中更新 FIT，以使所要删除的文件的数据群组记录变得过时。然后，当压缩 FIT 时，通常消除对过时数据群组的 FIT 的记录。在最终步骤 353 中，更新文件目录 201（图 20），以从中移除所删除的文件。

图 31 的流程图图解说明图 26 所示垃圾收集操作 265 及图 28 所示垃圾收集操作 329 的具体实例。当主机发送空闲命令（图 26 的步骤 264）或者当主机已空闲一定时间（图 26 的步骤 265）时，该算法被输入后台，或者在编程操作（图 28 的步骤 329）过程中当在被擦除元块池中保留有不到 N_2 个被擦除元块时，被输入前台。第一询问 355 是是否存在尚未完成的仍在进行中的垃圾收集操作。从图 31 的该说明中将会看出，所述处理在某些情况下退出垃圾收集，例如当主机发出另一命令或者当数据复制与其他操作相交织时。因此，如果存在未完成的待决垃圾收集操作，则接着进行步骤 357，因为待决的垃圾收集被赋予优先权。但是如果通过步骤 355 确定出不存在待决的垃圾收集操作，则随后在下一步骤 356 中查看各垃圾收集队列，以看其中是否存在至少一个条目。如果存在，则在下一步骤 358 中根据上文所述的优先权来选择其中一个条目（如果存在不止一个）。过时元块块队列中按次序的下一元块的垃圾收集通常将被赋予高于共用元块队列中的元块或高于要进行垃圾收集的文件队列中的文件的优先权。这是因为由于不需要复制任何数据，因而可通过过时的元块进行垃圾收集来更快地增大被擦除元块池的大小。

下一组步骤 360、362 及 366 针对所选的队列条目确定垃圾收集目标，因为所述过程对文件、共用元块及过时元块而言是不同的。询问 360 询问其是否是正对文件进行垃圾收集。如果是，则在步骤 370 及 372 中按次序读取所述文件的 FIT 条目，以设定某些计数器及一计数值。在步骤 370 中，将第一计数器设定为文件中的数据群组数量，并将第二计数器设定为文件的第一数据群组中数据元页的数量（长度）。如果 FIT 数据群组条目的长度不属于其条目的一部分，则可根据 FIT 数据群组条目来计算该长度。另一选择为，可包含每一数据群组的长度作为其在 FIT 中的条目的一部分，如在图 20 及 22 中所示，以便无需每次需要数据群组长度时均对其进行计算。在步骤 732 中，将第三计数器设定为存储于包含过时数据或其他文件的数据的存储区块中的当前

文件数据的元页数量。这是当前文件中需要移动的数据。该第三计数值较佳忽略所述文件内的所有过时的及无效的数据群组。最后，对假如将文件的有效数据压缩至并填充完整数个元块时所将得到残留数据元页数量进行计数。换句话说，残留元页计数值是文件中不到一元块的数据的量，假如在垃圾收集中包括所有含所述文件的数据的元块，则所述数据将须占据一公用的或局部得到填充的元块。在设定这三个计数器并进行残留元页计数及存储所述计数之后，在下一步骤 359 中对所述文件的数据进行垃圾收集。

返回至步骤 360，如果步骤 358 所选的条目不是一文件，则下一询问 362 判定其是否是将被进行垃圾收集的共用元块。如果是，则执行图 32 所示的共用元块垃圾收集 364。如果不是，则询问 366 判定所选条目是否对应于一过时元块，例如通过图 30 所示步骤 345 添加至过时元块队列中的过时元块。如果是，则在步骤 368 中擦除所选元块。在共用元块垃圾收集 364、元块擦除 368 之后，或者如果询问 366 的结果是否定的响应，则所述处理返回至图 26 所示步骤 253。如果垃圾收集操作仍不完整，则其将在下次进入垃圾收集算法时继续进行。

返回至步骤 355，如果存在待决的垃圾收集，则将对其进行进一步处理。对于从垃圾收集队列中所选的新的项，图 31 所示步骤 357 判定所述待决的操作是否对应于一文件，如步骤 360 一样。如果不对应于一文件，则执行上文针对新的项所述的步骤 362、364、366 及 368 的处理，并随后通过返回至图 26 所示步骤 253 来结束垃圾收集。

如果图 31 所示步骤 357 确定出继续进行的垃圾收集是对文件进行，则接下来进行询问 359。接下来的处理对于继续进行垃圾收集的文件（通过步骤 357）或对于选自队列中的其计数器及残留元页计数值通过步骤 370 及 372 加以设定的新文件（通过步骤 360）而言实质上相同。当对文件的垃圾收集正在进行并通过步骤 357 继续进行时，当首先开始对文件进行垃圾收集时，对数据群组数量及文件元页数量计数器以及残留元页计数值进行设定。数据群组数量及元页计数器可能已递减至不同于通过对当前文件进行早先垃圾收集所最初计算出的值。当中止对文件的垃圾收集时，存储所有四个值，并在对同一文件进行继续处理期间存取这四个值。

共用步骤 359 询问在当前数据群组内是否存在一个或多个尚待复制的元页。这是参照数据群组长度计数器加以确定。通过各后续步骤每次检查及复制有效数据群组的一个元页。如果通过步骤 359 确定出数据仍保留于待决数据群组中，则下一询问 361 是判定一复制元块是否打开。如果是，则通过步骤 363 将所正进行垃圾收集的文件的当前数据群组的一数据元页从存储其的元块中读出，并在步骤 365 中写入至复制元块。然后在步骤 367 中将数据群组长度计数器递减一个元页，并在步骤 369 中将文件元页计数器递减 1。

如果通过步骤 361 确定出一复制元块未打开，则在下一步骤 371 中判定正进行垃圾收集的文件是否是打开的文件。如果是，则在下一步骤 373 中打开一复制元块，且所述过程随后进行至前面所述的步骤 363。但是如果不是一打开的文件，则在下一步骤 375 中询问文件元页计数器的递减后的计数值是否等于残留元页计数值。如果不等

于，则在步骤 373 中打开一复制元块。但是如果其相等，则此意味着在当前文件中不再剩下要复制的数据元块。因此，在步骤 377 中，识别一打开的共用元块以将残留数据写入其中。步骤 363 及之后的处理是将当前文件的当前数据群组的剩余元页复制至一共用元块中。当通过步骤 371 确定出正被进行垃圾收集的是一打开的文件时，不采用该路径，因为可将其他原本将填充另一元块的数据群组写入至当前文件中。在打开的文件关闭之后，将其置于垃圾收集队列中，此时，通过步骤 377 将任何残留数据复制至一共用元块。

图 31 所示询问 379 判定在写入该额外数据元页之后所述复制元块现在是否已满。如果是，则在步骤 381 中关闭复制元块，并在步骤 383 中更新 FIT，以反映该事实。如果复制元块未满或者在步骤 383 之后，询问 385 判定被进行垃圾收集的元块中的所有数据是否现在均已变无效。如果是，则在步骤 387 中擦除元块。如果不是，或者在元块擦除之后，询问 389 询问在编程操作期间是否通过图 28 所示的步骤 325 设定前台模式。如果不是，则图 31 所示步骤 391 判定一主机命令是否待决。如果是，且如果在步骤 392 中确定正在进行的垃圾收集不具有优先权，则通过在步骤 393 中更新 FIT 并随后返回至图 26 所示步骤 253 来中止垃圾收集。然而，如果在步骤 391 中确定一主机命令不在待决，或者如果存在待决的主机命令且在步骤 392 中确定正在进行的垃圾收集不具有优先权，则所述处理返回至询问 359，以将下一数据元页从当前文件元块复制至所示复制元块中。

如果图 31 所示询问 389 确定出已设定前台模式，则在下一步骤 395 中递增通过图 28 所示步骤 327 复位的复制元块元页计数器。如果一步骤 397 确定已接连复制了预定数量 N4 个元页，则在下一步骤 399 中将前台模式复位并通过步骤 393 更新 FIT。但是如果在达到数量 N4 之前还有元页要复制，则所述过程继续进行步骤 359，除非询问 391 确定存在待决的主机命令。如果存在待决的主机命令，则中断复制操作，且所述过程返回至图 26 所示的步骤 253。

返回至询问 359，如果数据群组长度计数值为 0，则此意味着已完成一个数据群组从文件元块向复制元块的复制。在此种情形中，在步骤 401 中更新 FIT 以反映该状态。接下来，在步骤 403 中参照数据群组编号计数器来判定正被进行垃圾收集的当前文件是否包含另一数据群组。如果不包含，则对文件的垃圾收集已完成。然后，所述过程返回至步骤 356，以按次序对队列中的下一文件或元块进行垃圾收集。其将不返回至步骤 355，因为对当前文件的垃圾收集的完成意味着可不再存在正在进行且可重新开始的文件垃圾收集。

如果从步骤 403 得知当前文件还存在至少一个要复制的数据群组，则在步骤 405 中判定包含该下一数据群组的元块是否也包含其他过时的数据。如前面所述，对文件的包含过时数据的元块进行垃圾收集，但较佳不对文件的不包含过时数据的元块进行垃圾收集，除非所述元块是共用元块或包含文件的残留数据的不完整元块。因此，如果在步骤 405 中判断出在下一数据群组的元块中存在过时数据，则在步骤 407 中将数

据群组编号计数器更新成按次序的下一数据群组的编号，且在步骤 409 中以新数据群组中的数据量将数据群组长度计数器初始化。所述处理然后进行至步骤 361，以按前面所述的方式将第一元页的数据从文件的新数据群组复制至一复制元块中的下一被擦除元页。

但是即使在步骤 405 中确定出在其中存在下一数据群组的元块中没有过时数据，如果（1）下一数据群组是共用元块或者（2）处于不完整元块中且所述文件是关闭的文件，则仍可进行对该元块中数据的垃圾收集。首先作出关于当前数据群组是否处于共用元块中的询问 411。如果是，则在下一步骤 413 中将共用元块添加至共用元块垃圾收集队列中以便此后进行垃圾收集，但所述过程继续进行步骤 407 及 409，以将数据群组编号计数器及数据群组长度计数器更新成要由步骤 361 等等进行复制的下一元块的值。然而，如果当前数据群组不处于共用元块中，则在步骤 415 中询问当前数据群组是否处于不完整的元块中。换句话说，步骤 415 判定当前数据群组是否存在仍具有至少最小量的被擦除容量的元块中（例如图 15 所示的数据群组 F3, D3）。如果不是，则不复制当前数据群组，而是所述处理返回至步骤 403 以处理所述文件的下一数据群组（如果存在一个下一数据群组）。但是，如果当前数据群组处于不完整的元块中，则下一询问 417 询问所述不完整元块是否包含打开的文件的数据。如果是，则通过返回至询问 403 来跳过对当前数据群组的复制，以继续所述文件的任何其他数据群组。但是如果其中存在下一数据群组的元块并不包含打开的文件的数据，则对该下一数据群组执行步骤 407、409、361 等等。

返回至图 31 的步骤 356，甚至在判断出在垃圾收集队列中不存在文件或元块时，也可进行垃圾收集。如果不存在，而是结束所述过程，则在步骤 421 中检查在被擦除元块池中是否存在多于 N2 个被擦除元块。如果是，则垃圾收集过程结束并返回至图 26 所示的步骤 253。但是如果在系统中不存在多于 N2 个被擦除元块，则获取机会从步骤 421 开始执行垃圾收集。如步骤 423 所示，此种垃圾收集可对打开的文件（如果存在一个打开的文件）进行。由于队列中无任何项，因而当所述池中被擦除元块的数量为 N2 或以下时，有可能不存在任何其他更多被擦除元块来源。如果存在多于一个打开的文件，则在步骤 425 中选择其中一个。然后如前面所述通过步骤 370、372 及 359 对打开的文件进行所述处理。

当图 31 中的询问 362 确定出正对共用元块中的数据群组执行当前的垃圾收集操作时，垃圾收集 364 略微不同于上文针对其他元块所述。图 32 所示流程图概述对共用元块的垃圾收集。在步骤 431 中询问垃圾收集是否正在进行中且因此重新开始。如果是，则数据群组长度计数器保留有所述垃圾收集中止时的最后值。如果不是，则首先在步骤 435 中参照作为处理对象的第一数据群组的 FIT 来将数据群组编号计数器初始化，并随后通过步骤 433 确定该数据群组的长度。

当通过步骤 433 确定出数据群组长度计数器大于 0 时，则对当前数据群组的复制开始。在步骤 437 中，从共用元块中读取当前数据群组的元页，并在步骤 439 中将所

述元页编程至打开的共用元块中。在步骤 441 中，然后将数据群组长度计数器递减一个元页。如果如步骤 443 所示在前台进行垃圾收集，则在步骤 445 中使所述复制元块元页计数器递增 1。该计数器记录在当前序列中所已复制的数据元页的数量。如果在步骤 447 中确定出该计数值超过预定数量 N4，则所述过程随后在步骤 449 中退出前台模式。随后在步骤 451 中更新 FIT，且所述处理返回至图 26 所示的步骤 253。

如果通过图 32 中的步骤 447 确定出所述复制元块元页计数器的值为 N4 或以下，则在下一步骤中询问是否有一主机命令待决。如果是，则在步骤 451 中更新 FIT，且所述处理返回至图 26 的步骤 253，以便可执行待决的主机命令。如果不是，则所述处理返回至图 32 的步骤 433，以询问在当前数据群组中是否存在另一数据元页，且如果存在，则将其从过时共用元块复制至打开的共用元块，等等。类似地，如果在步骤 443 中确定出未在前台执行垃圾收集，则接下来执行步骤 453。除非在步骤 453 中确定出有一主机命令待决，否则所述处理随后返回至步骤 433，以潜在地复制当前数据群组中的下一数据元页，而无论所述复制元块元页计数器是否超过 N4，因为已绕过了步骤 447。在此种情形中，垃圾收集是在后台执行。

返回至图 32 中的步骤 433，如果数据群组长度计数值不大于 0，则得知当前数据群组的所有元页均已得到复制。然后在步骤 455 中更新 FIT，且在步骤 457 中判定在被进行垃圾收集的共用元块中是否存在另一数据群组。如果不存在，则在步骤 459 中擦除过时的共用元块，且所述处理返回至图 26 所示的步骤 253。但是，如果在共用元块中存在另一数据群组，则在步骤 461 中更新数据群组编号计数器。然后在步骤 463 中从共用元块索引读取 FIT 指针，且在步骤 465 中读取由该指针所界定的 FIT 条目。如果在步骤 467 中确定出 FIT 条目与当前数据群组相一致，则一询问 468 判定当前数据群组是否是已识别出的残留数据的一部分。如果是，则在步骤 469 中使数据群组长度计数器初始化至从 FIT 读取的数据群组的长度。

但是如果在步骤 468 中确定出当前数据群组不处于现有残留数据中，则然后在步骤 470 中识别由其作为一部分的新残留数据。在步骤 471 中，然后识别具有足以存储新残留数据的所有数据的空间的打开的共用区块。此会防止当残留数据包含两个或更多个数据群组时在两个或更多个不同元块之间划分文件的残留数据。否则，假如独立地复制这两个或更多个数据群组，则可能会发生此种情况。在此种情形中，可能将第一数据群组复制至具有足以存储该数据群组的被擦除空间但不具有也足以存储第二数据群组的空间的共用区块中。然后，第二数据群组会复制至一不同的共用区块。这将是人们所不期望的，因为不应将文件的残留数据在两个不同元块之间进行划分。假如这样对其进行划分，则对文件或对共用区块的垃圾收集将耗用更多的时间。

然后，在步骤 469 中将当前数据群组的数据群组长度计数器初始化。在步骤 437 及 439 中将当前数据群组的第一数据元页从共用元块复制至所识别的打开的共用元块中。然而，如果在步骤 467 中确定出 FIT 条目不与当前数据群组相一致，则所述处理返回至步骤 457，以判定在共用元块中是否存在另一数据群组。

图 32 中流程图所示的垃圾收集继续进行，直至当前共用区块的所有有效数据群组均已复制至一先前被擦除的元块（其随后变为新的共用元块）或者复制至一个或多个打开的共用元块为止。可将不同文件的数据群组复制至不同的打开的共用元块。所述共用元块先前置于垃圾收集队列中，因为其包含过时的数据群组。对共用元块中每一有效数据群组的所有数据元页进行完整传送会使得对于每一此种元页均经过图 32 所示各步骤一遍。如果垃圾收集是在前台进行，则此种复制可每 N4 个元页中断一次，或者当在前台或后台运行时接收到新主机命令时中断。

作为永久性预设 N3 及 N4 数值的替代方式，可由存储器系统控制器响应于主机的数据编程图案来更改这些数值，以保持均匀的数据编程速度。

存储区块在操作期间的各种状态

图 33 的图式显示在上述类型直接文件存储存储器系统内，系统中存储区块或元块的各种单独状态、这些状态之间的变迁。

在左边的栏中，显示区块 501 处于被擦除状态，处于被擦除区块池中。

在下一栏中，区块 503、505 及 507 分别包含某些有效数据，但还具有可在其中写入主机数据的被擦除容量。写入区块 503 被局部写入有单个文件的有效数据，且所述文件的其他数据在由主机提供时应写入至该区块。复制区块 505 被局部写入有单个文件的有效数据，且所述文件的其他数据当在对所述文件进行垃圾收集期间得到复制时应写入至该区块。打开的共用区块 507 被局部写入有两个或更多个文件的有效数据，且在垃圾收集期间可将任何文件的残留数据群组写入至该区块。

下一栏的区块 509 及 511 则充满文件数据。文件区块 509 充满单个文件的有效数据。共用区块 511 充满两个或更多个文件的有效数据。

靠近图 33 的右手栏包括区块 513、515、517、519 及 521，其分别包含某些过时数据。过时文件区块 513 充满单个文件的有效数据与过时数据的任意组合。过时写入区块 515 被局部写入有单个文件的有效数据与过时数据的任意组合，且所述文件的其他数据在由主机提供时应写入至该区块。过时复制区块 517 被局部写入有单个文件的有效数据与过时数据的任意组合。过时打开共用区块 519 被局部写入有两个或更多个文件的有效数据与过时数据的任意组合。过时共用区块 521 充满两个或更多个文件的有效数据与过时数据的任意组合。

图 33 右手栏中的过时区块 523 仅包含过时数据。

各个区块在图 33 所示各区块状态之间的变迁也由以小写字母进行标记的线加以显示。这些变迁如下：

- a - 被擦除区块 501 至写入区块 503：将单个主机文件的数据写入至被擦除区块。
- b - 写入区块 503 至写入区块 503：将来自主机的单个文件的数据写入至该文件的写入区块中。
- c - 写入区块 503 至文件区块 509：写入来自主机的单个文件的数据以填充该文件的一写入区块。

d - 文件区块 509 至过时文件区块 513: 文件区块中数据的一部分因主机在该文件的写入区块中写入所述数据的更新版本而变得过时。

e - 过时文件区块 513 至过时区块 523: 过时文件区块中的所有有效数据因在垃圾收集期间所述数据被复制至另一区块或因主机删除所述文件而变得过时。

f - 写入区块 503 至过时写入区块 515: 写入区块中数据的一部分因主机将所述数据的更新版本写入于同一写入区块中、或者因在垃圾收集期间将所述数据复制至另一区块中而变得过时。

g - 过时写入区块 515 至过时写入区块 513: 将来自主机的单个文件的数据写入至该文件的过时写入区块。

h - 过时写入区块 515 至过时文件区块 513: 写入来自主机的单个文件的数据，以填充该文件的过时写入区块。

i - 过时写入区块 515 至过时区块 523: 过时写入区块中的所有有效数据因在垃圾收集期间将所述数据复制至另一区块或因主机删除所述文件而变得过时。

j - 被擦除区块 501 至复制区块 505: 在垃圾收集期间将单个文件的数据从另一区块复制至被擦除区块。

k - 写入区块至复制区块 505: 在垃圾收集期间将单个文件的数据从另一区块复制至该文件的一写入区块。

l - 复制区块 505 至复制区块 505: 在垃圾收集期间将单个文件的数据从另一区块复制至该文件的一复制区块。

m - 复制区块 505 至文件区块 509: 在垃圾收集期间从另一区块复制单个文件的数据，以填充该文件的一复制区块。

n - 复制区块 505 至写入区块 503: 当在垃圾收集期间重新打开来自主机的单个文件时，将所述单个文件的数据写入该文件的复制区块中。

o - 复制区块 505 至过时复制区块 517: 复制区块中的一部分或所有数据因主机在所述文件的写入区块中写入所述数据的更新版本或者因主机删除所述文件而变得过时。

p - 过时复制区块 517 至过时区块 523: 过时复制区块中的所有有效数据因在垃圾收集期间将所述数据复制至另一区块或者因主机删除所述文件而变得过时。

q - 写入区块 503 至打开的共用区块 507: 在垃圾收集期间将文件的残留数据写入至一不同的关闭的文件的写入区块中。

r - 在垃圾收集期间将文件的残留数据写入至一包含不同文件的残留数据的复制区块中。

s - 在垃圾收集期间将文件的残留数据从一不同的区块复制至一打开的共用区块。

t - 打开的共用区块 507 至过时的打开的共用区块 519: 打开的共用区块中一个文件的一部分或所有数据因主机将所述数据的更新版本写入所述文件的写入区块中、因

在垃圾收集期间将所述数据复制至另一区块、或者因主机删除所述文件而变得过时。

u - 过时的打开的共用区块 519 至过时区块 523: 过时的打开的共用区块中的所有有效数据因在垃圾收集期间将所述数据复制至另一区块或者因主机删除所述文件而变得过时。

v - 打开的共用区块 507 至共用区块 511: 在垃圾收集期间从另一区块复制文件的残留数据群组，以填充该文件的打开的共用区块。

w - 共用区块 511 至过时共用区块 521: 共用区块中一个文件的一部分或所有数据因主机将所述数据的更新版本写入至所述文件的写入区块中、因在垃圾收集期间将所述数据写入至另一区块中、或者因主机删除所述文件而变得过时。

x - 过时共用区块 521 至过时区块 523: 过时共用区块中的所有有效数据因在垃圾收集期间将数据复制至另一区块中或者因主机删除所述文件而变得过时。

y - 写入区块 503 至过时区块 523: 写入区块中单个文件的所有有效数据因主机删除所述文件而变得过时。

z - 复制区块 505 至过时区块 523: 复制区块中的所有有效数据因在垃圾收集期间将数据复制至另一区块或者因主机删除所述文件而变得过时。

aa - 文件区块 509 至过时区块 523: 文件区块中的所有数据因主机删除所述文件而变得过时。

ab - 过时区块 523 至被擦除区块 501: 在垃圾收集期间存储过时区块。

D. 结论

虽然已根据本发明的各实例性实施例对本发明的各个方面进行了阐述，但应了解，本发明有权在随附权利要求书的整个范围内受到保护。

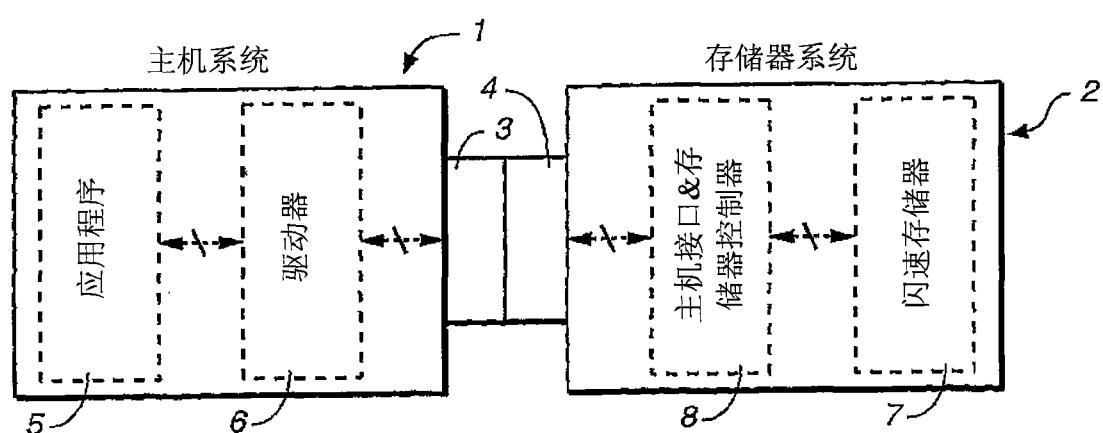


图 1

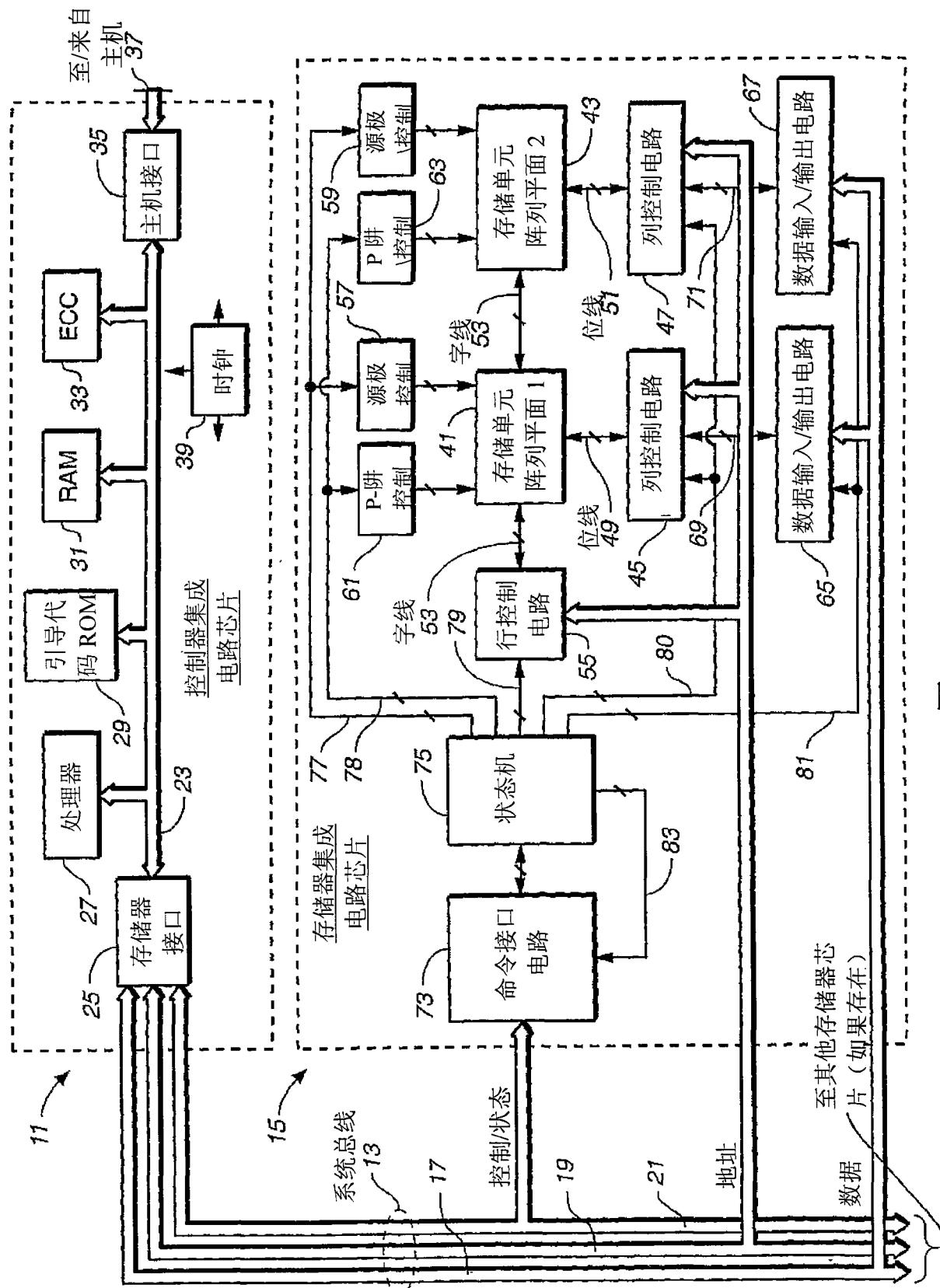


图 2

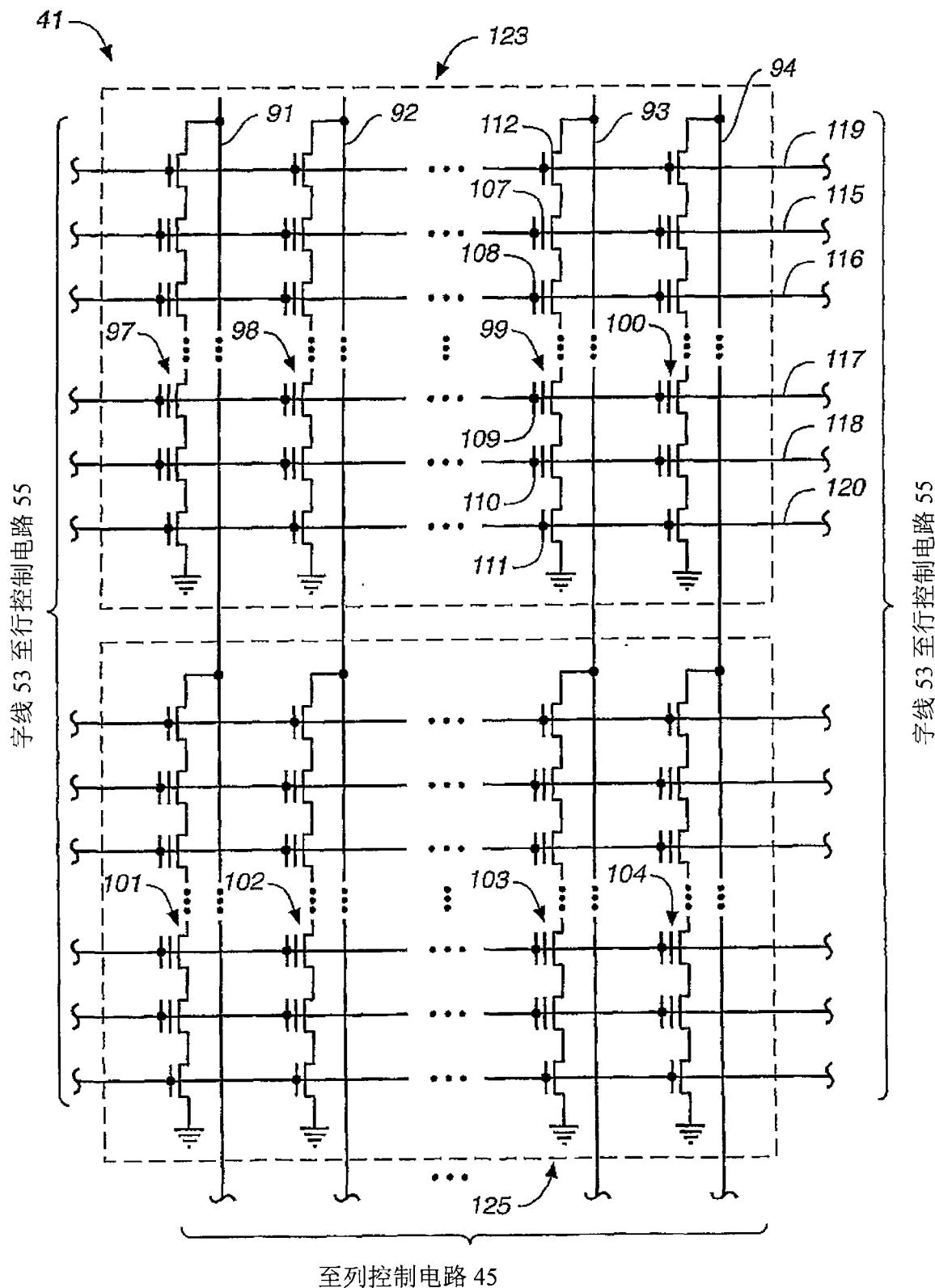


图 3

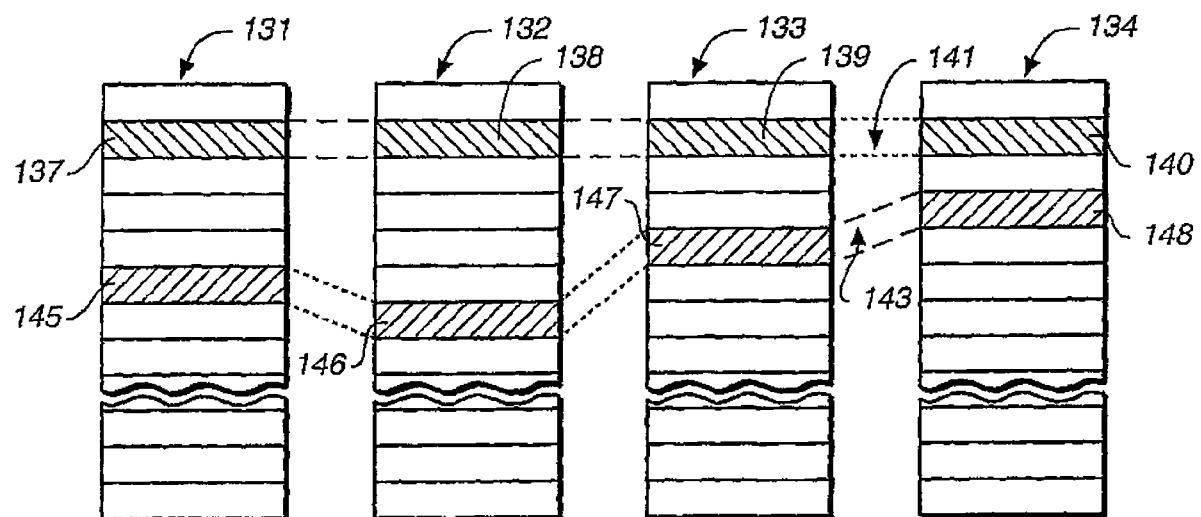


图 4

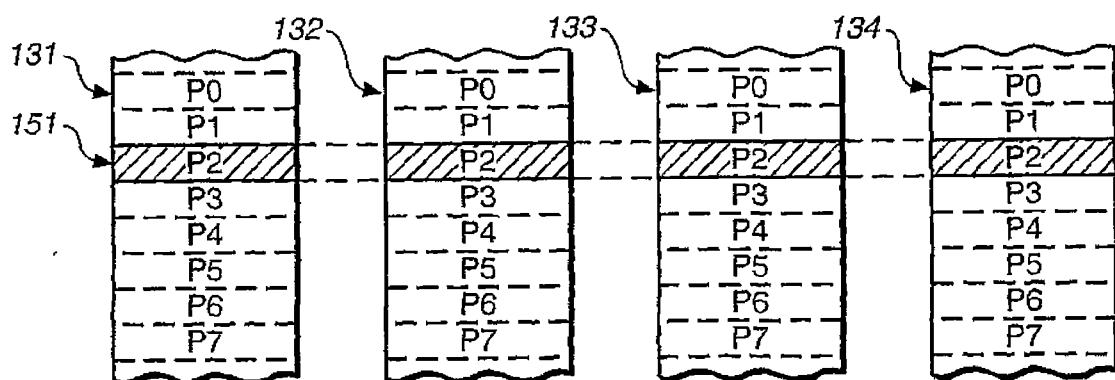


图 5

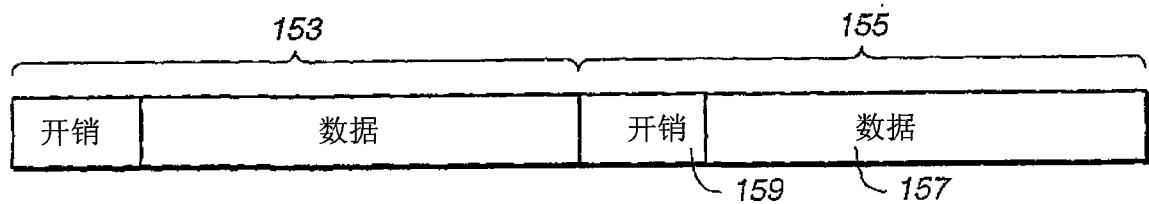


图 6

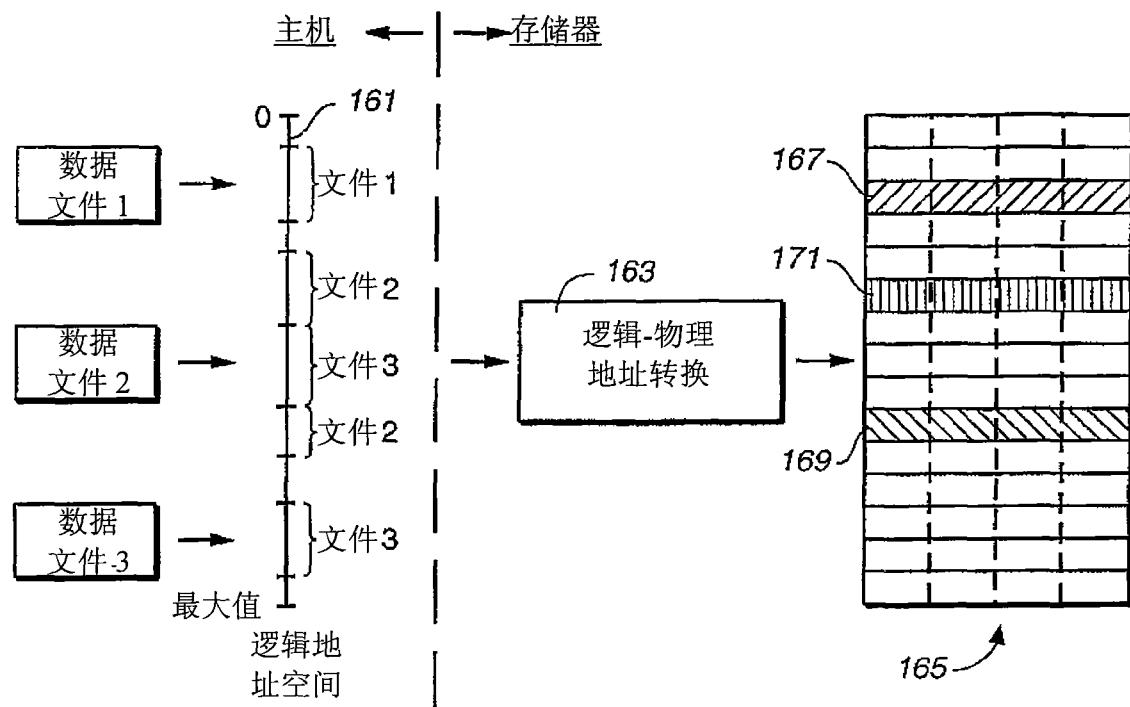


图 7

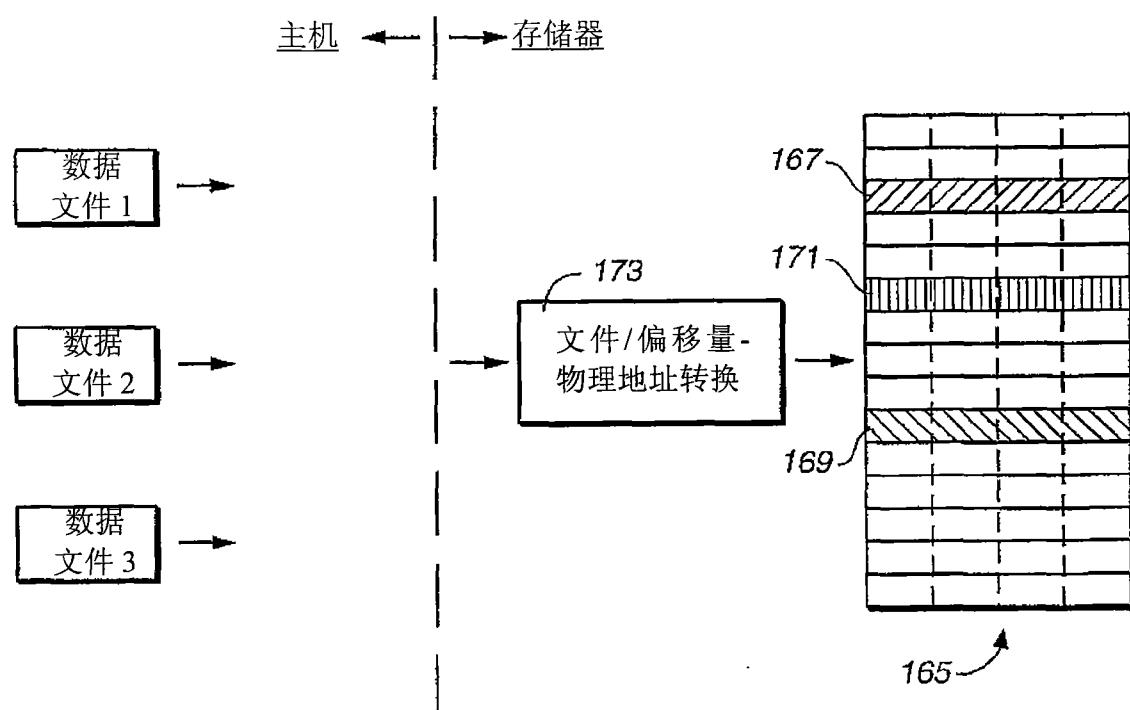


图 9

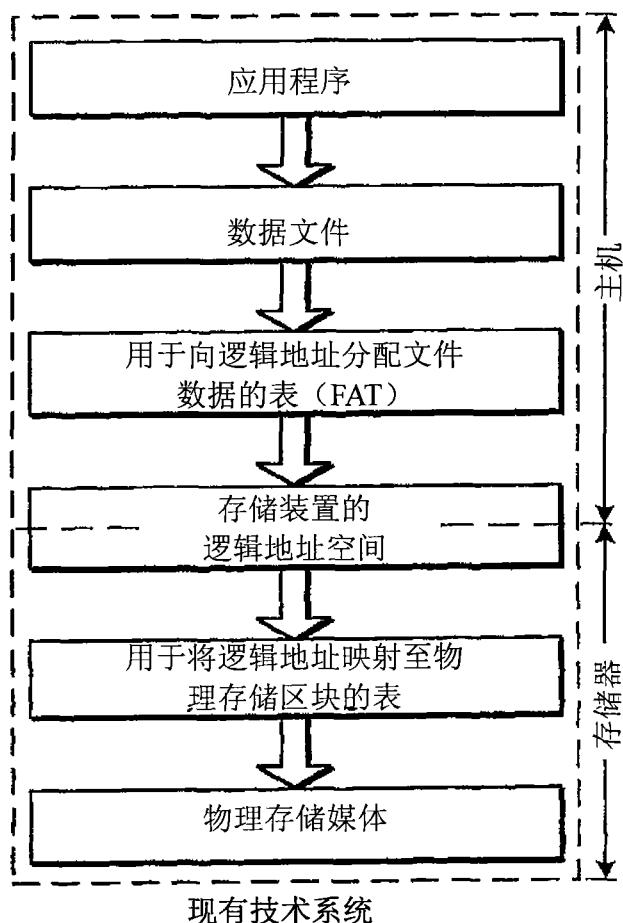


图 8

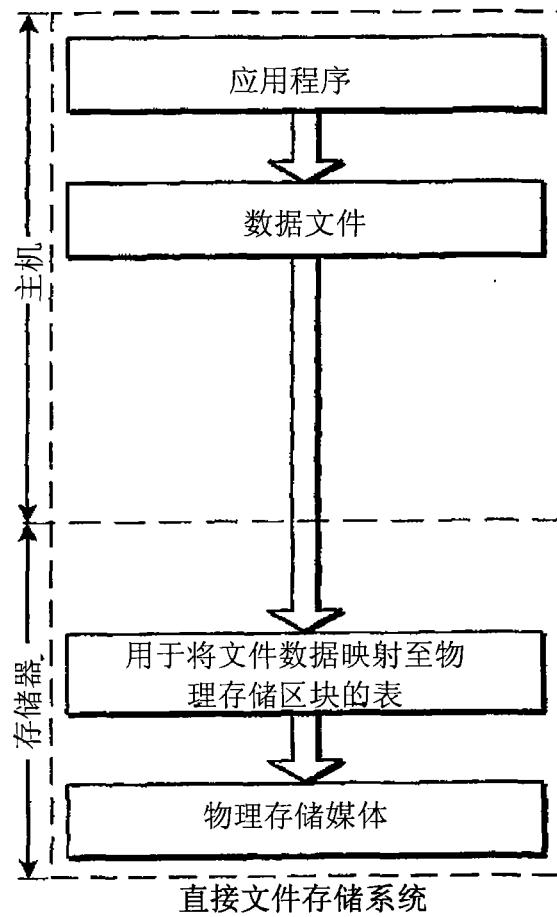


图 10

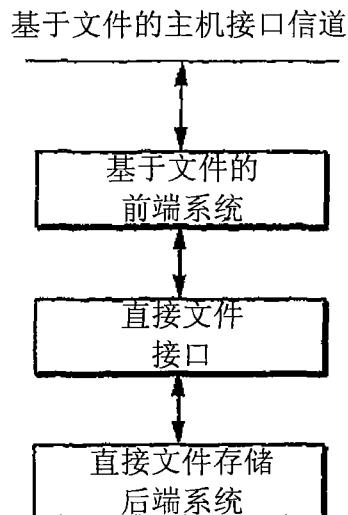


图 11

数据命令

命令	参数	说明
写入	<文件 ID>	将数据附加至所规定文件的末尾。
插入	<文件 ID><偏移量>	将数据从距开头<偏移量>处插入所规定文件内。
更新	<文件 ID><偏移量>	从距开头<偏移量>处覆盖所规定文件内的数据。
读取	<文件 ID><偏移量>	从距开头<偏移量>处读取所规定文件的数据。
移除	<文件 ID><偏移量 1> <偏移量 2>	移除所规定文件内距开头<偏移量 1>与<偏移量 2>之间的数据。

图 12A

文件命令

命令	参数	说明
打开	<文件 ID>	请求提供资源以用于写入至所规定文件。
关闭	<文件 ID>	可从所规定文件立即移除资源，且可对所述文件进行垃圾收集调度。
关闭_后	<文件 ID><长度>	可在从主机进一步传送文件数据<长度>之后从所规定文件移除资源，此时可对所述文件进行垃圾收集调度。
删除	<文件 ID>	指示应删除所规定文件的表条目及数据。 可擦除文件数据。
擦除	<文件 ID>	指示应删除所规定文件的表条目及数据并应立即擦除该文件数据。

图 12B

目录命令

命令	参数	说明
列表	<目录 ID>	来自主机的使装置报告命令及文件信息的请求。
当前	<目录 ID>	由主机作出的对在其中打开新文件、创建新目录或删除目录的当前目录的定义。
创建	<目录 ID>	来自主机的要创建新目录的请求
删除	<目录 ID>	来自主机的要删除目录的请求。也删除目录内的子目录及文件。
擦除	<目录 ID>	来自主机的要删除目录的请求。也删除目录内的子目录及文件。应立即擦除所规定文件的数据。

图 12C

状态命令

命令	参数	说明
空闲		主机接口正进入空闲状态，在该状态中，所述装置可执行内部操作。可通过由主机传输另一命令来结束空闲状态，无论所述装置是否忙于内部操作。 在接收到此种另一命令时，应在规定时间内终止正在装置内进行的任何内部操作。
待机		主机接口正进入待机状态，在该状态中，主机可不执行内部操作。可通过由主机传输另一命令来结束待机状态。
关机		主机接口将关机且当所述装置接下来处于不忙状态时将移除所述卡的电源。

图 12D

存储器系统命令

命令	参数	说明
大小		来自主机的使所述装置报告可用于新文件数据的容量的请求。
状态		来自主机的使所述装置报告其当前状态的请求。

图 12E

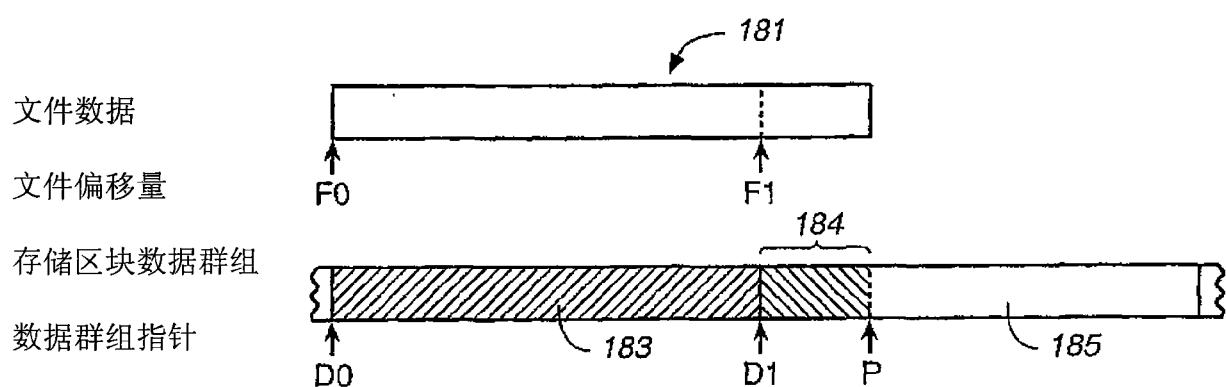


图 13A 写入

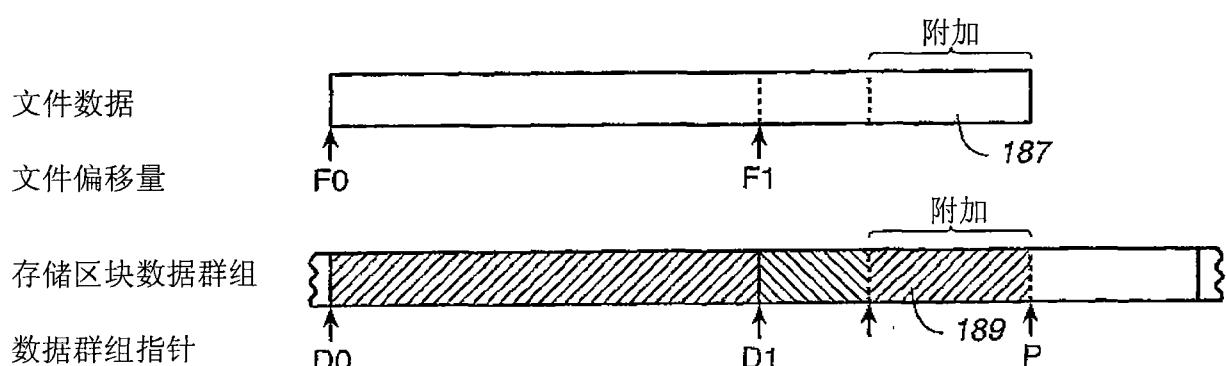


图 13B 写入

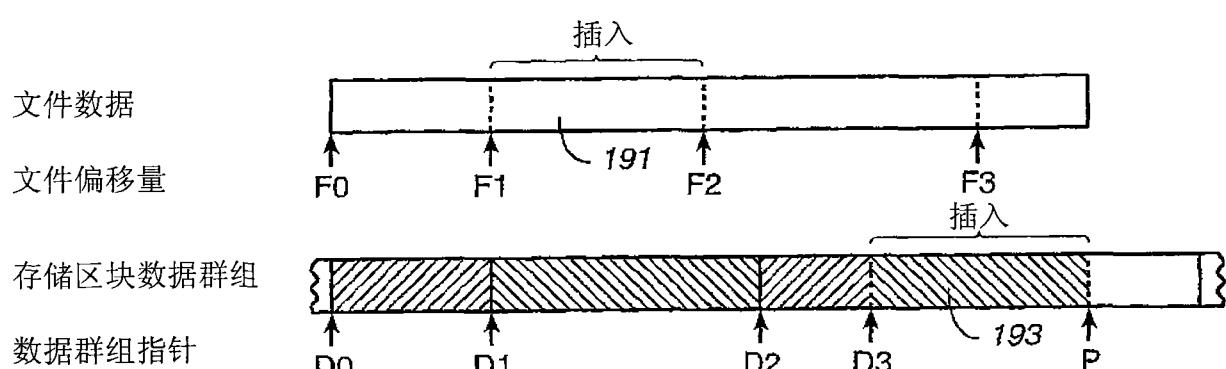


图 13C 插入

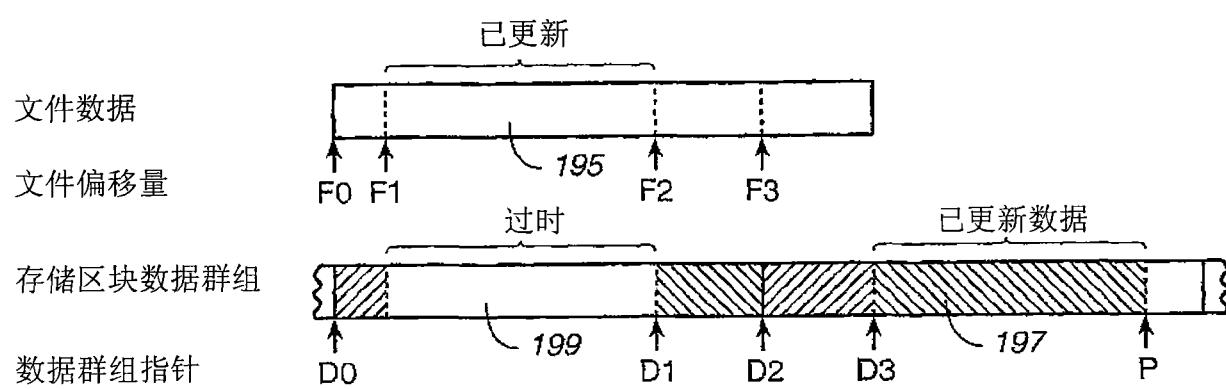
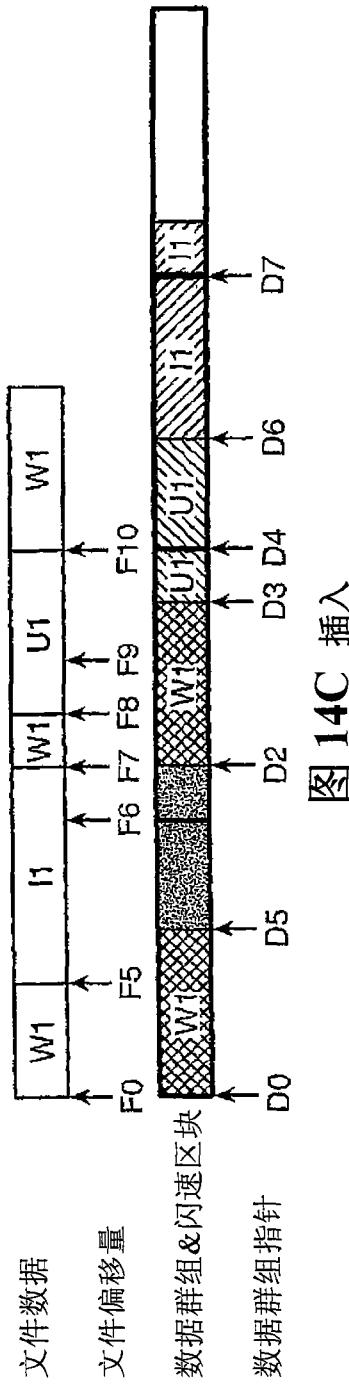
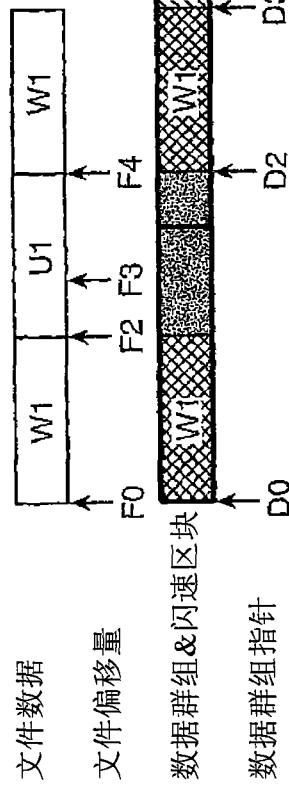
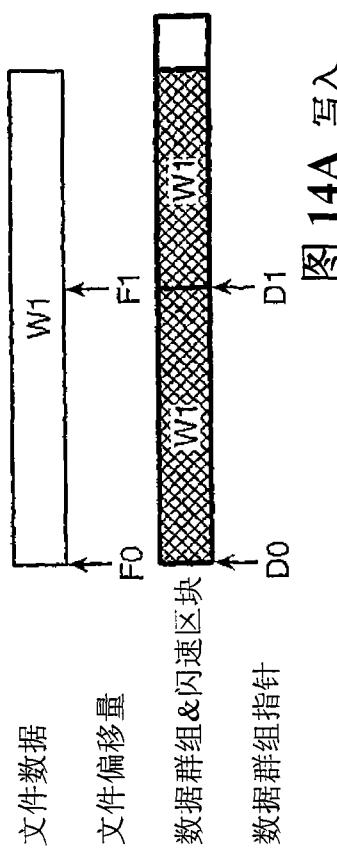


图 13D 更新



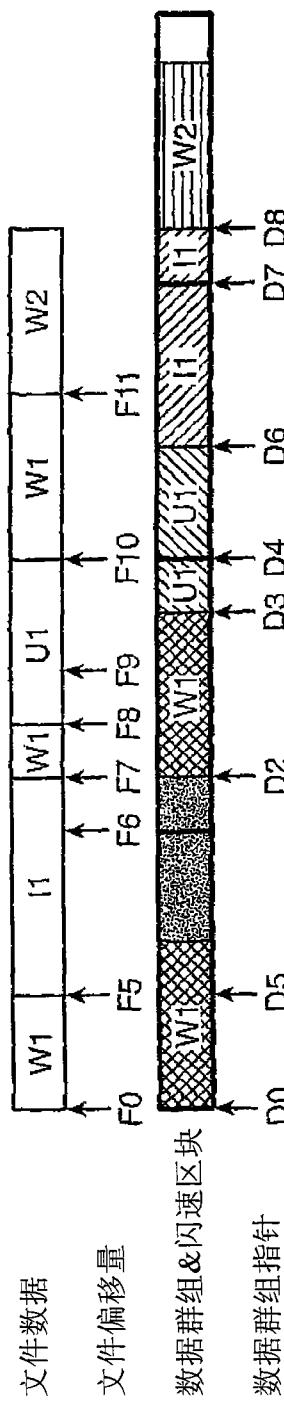


图 14D 更新

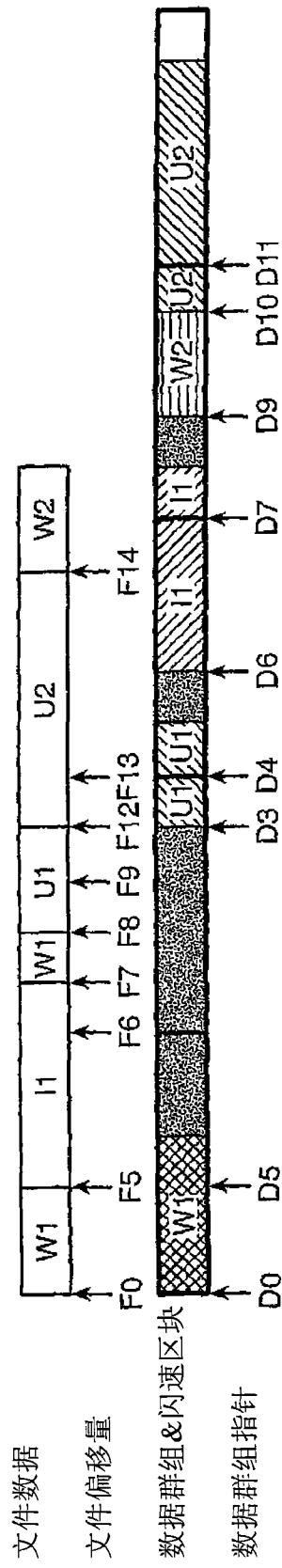


图 14E

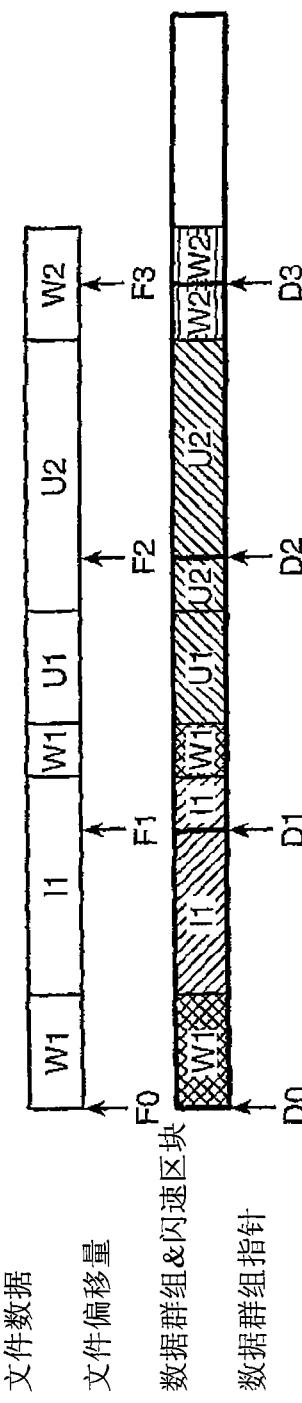


图 15

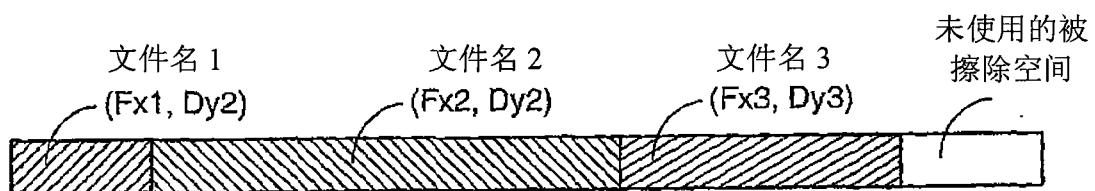


图 16

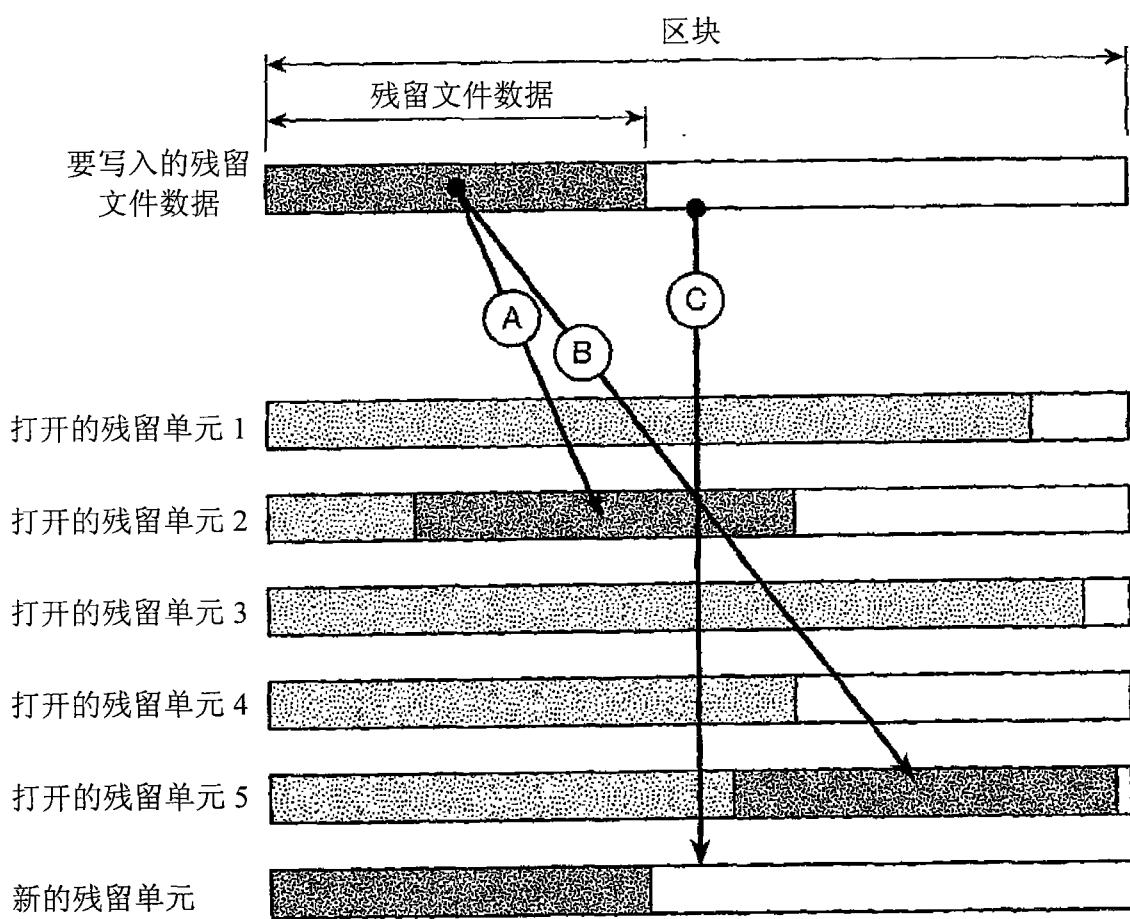


图 17

图 18

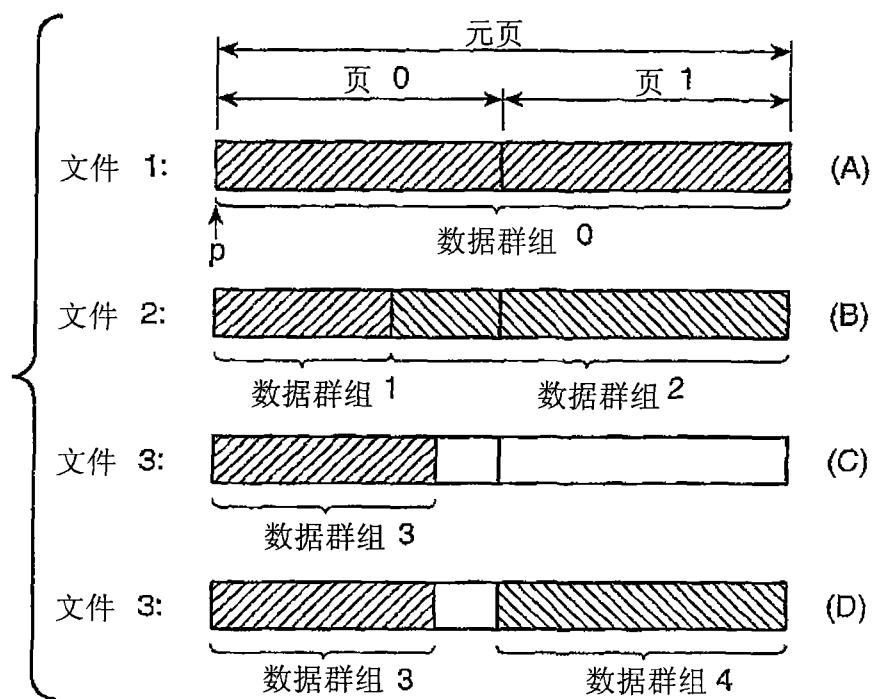


图 19

文件索引表 (FIT)

	偏移量	区块	字节	长度
时刻 0 (图 14A)	F0	001	D0	—
	F1	002	D1	—
时刻 2 (图 14C)	F0	001	D0	—
	F5	003	D6	—
	F6	004	D7	—
	F7	001	D5	—
	F8	002	D3	—
	F9	003	D4	—
时刻 4 (图 14E)	F10	002	D2	—
	F0	001	D0	—
	F5	003	D6	—
	F6	004	D7	—
	F7	001	D5	—
	F8	002	D3	—
	F9	003	D4	—
	F12	004	D10	—
时刻 5 (图 15)	F13	005	D11	—
	F14	004	D9	—
	F0	001	D0	—
	F1	002	D1	—
	F2	003	D2	—
	F3	004	D3	—

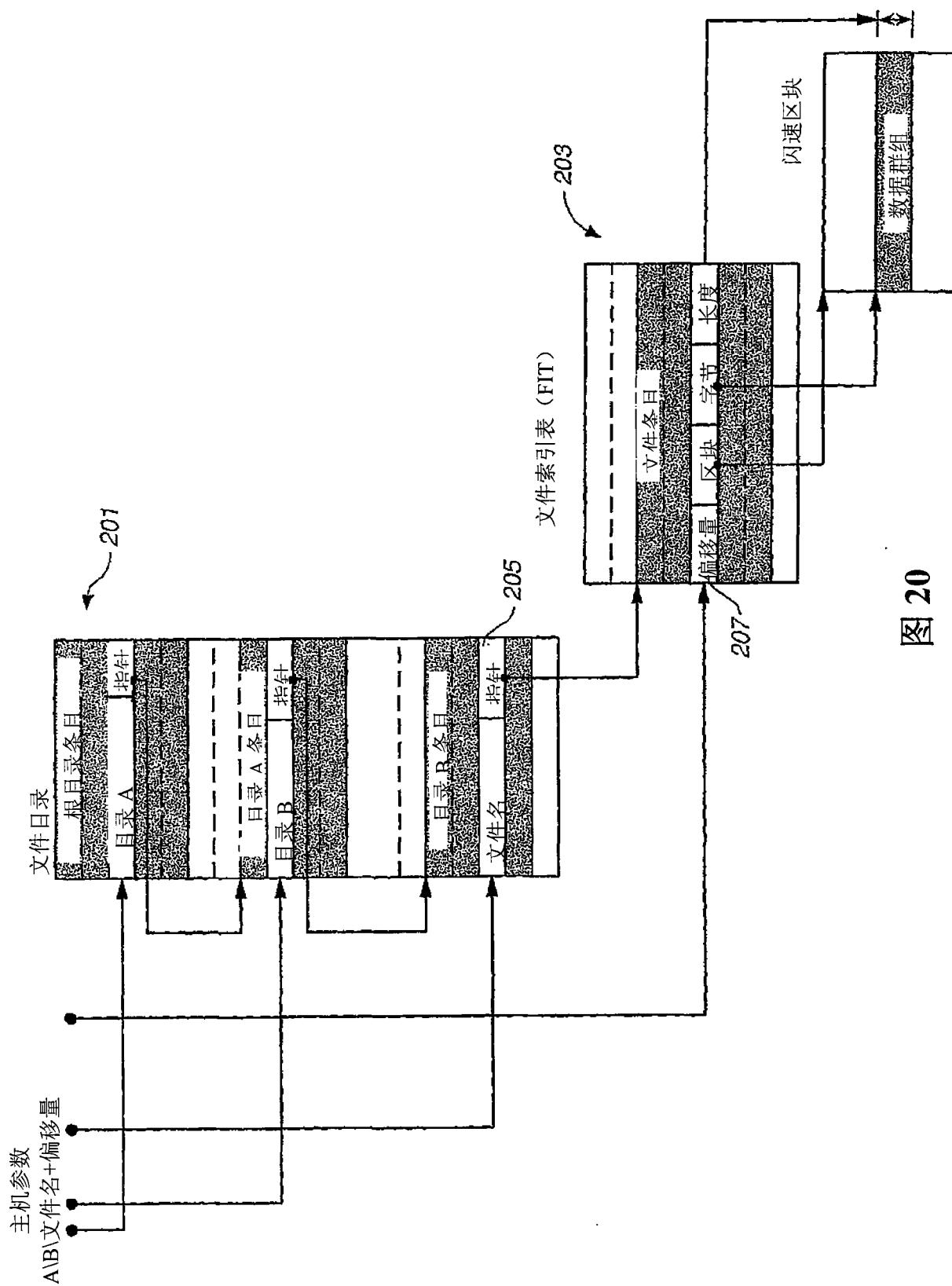


图 20

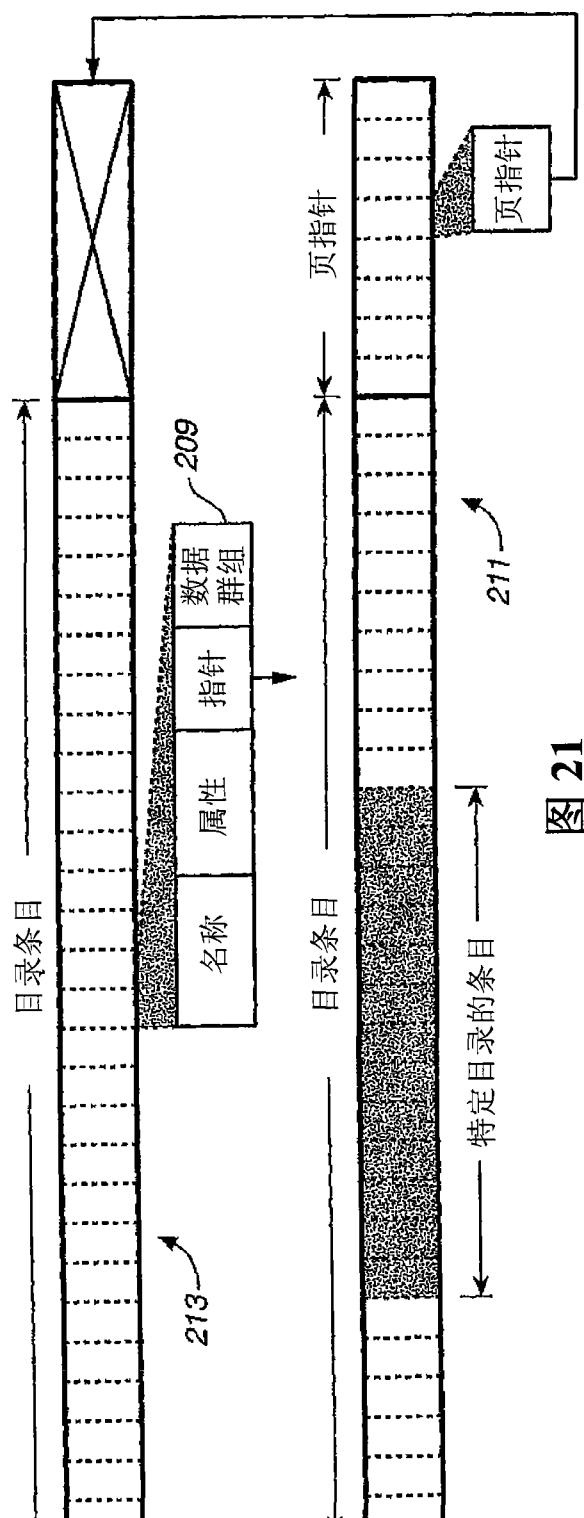


图 21

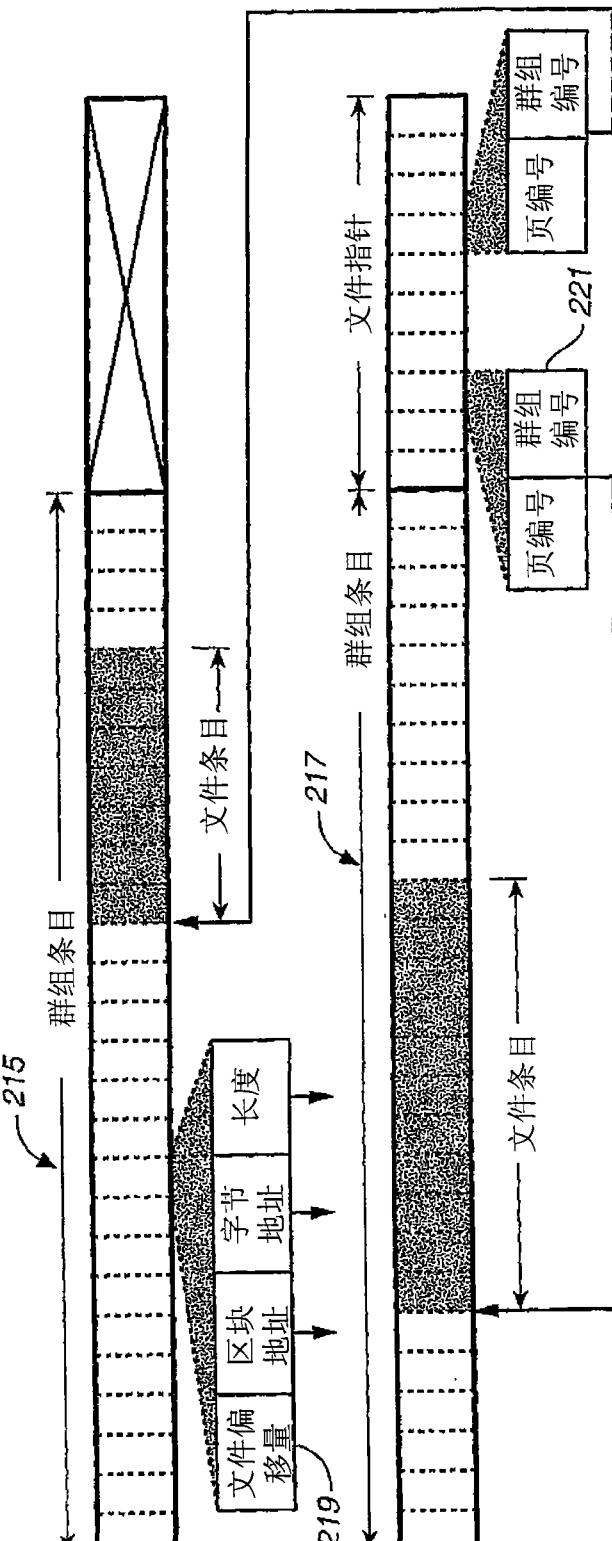


图 22

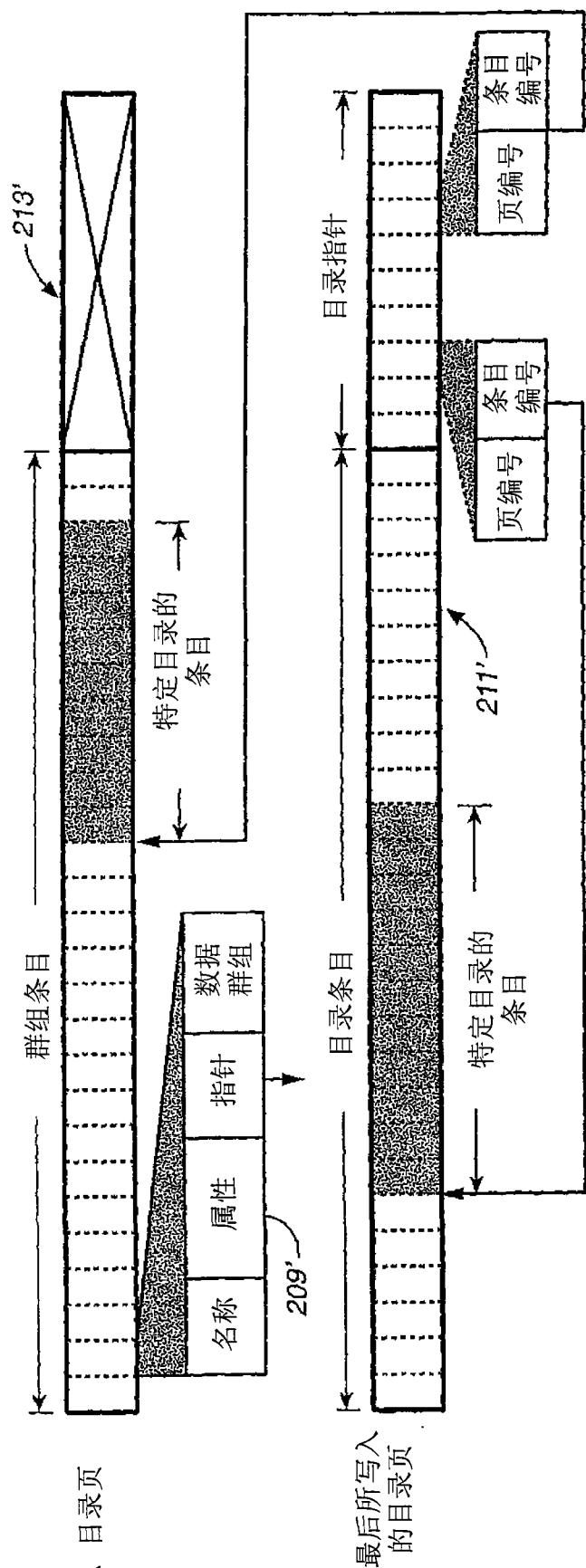


图 23

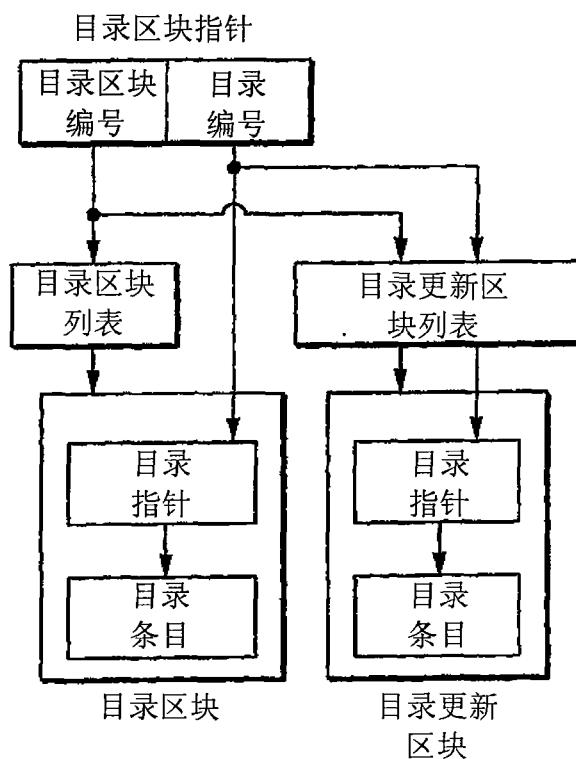


图 24

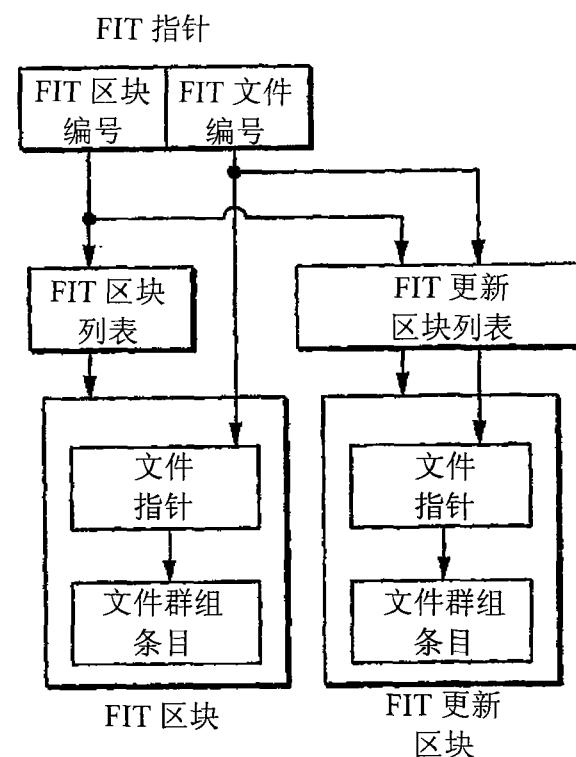


图 25

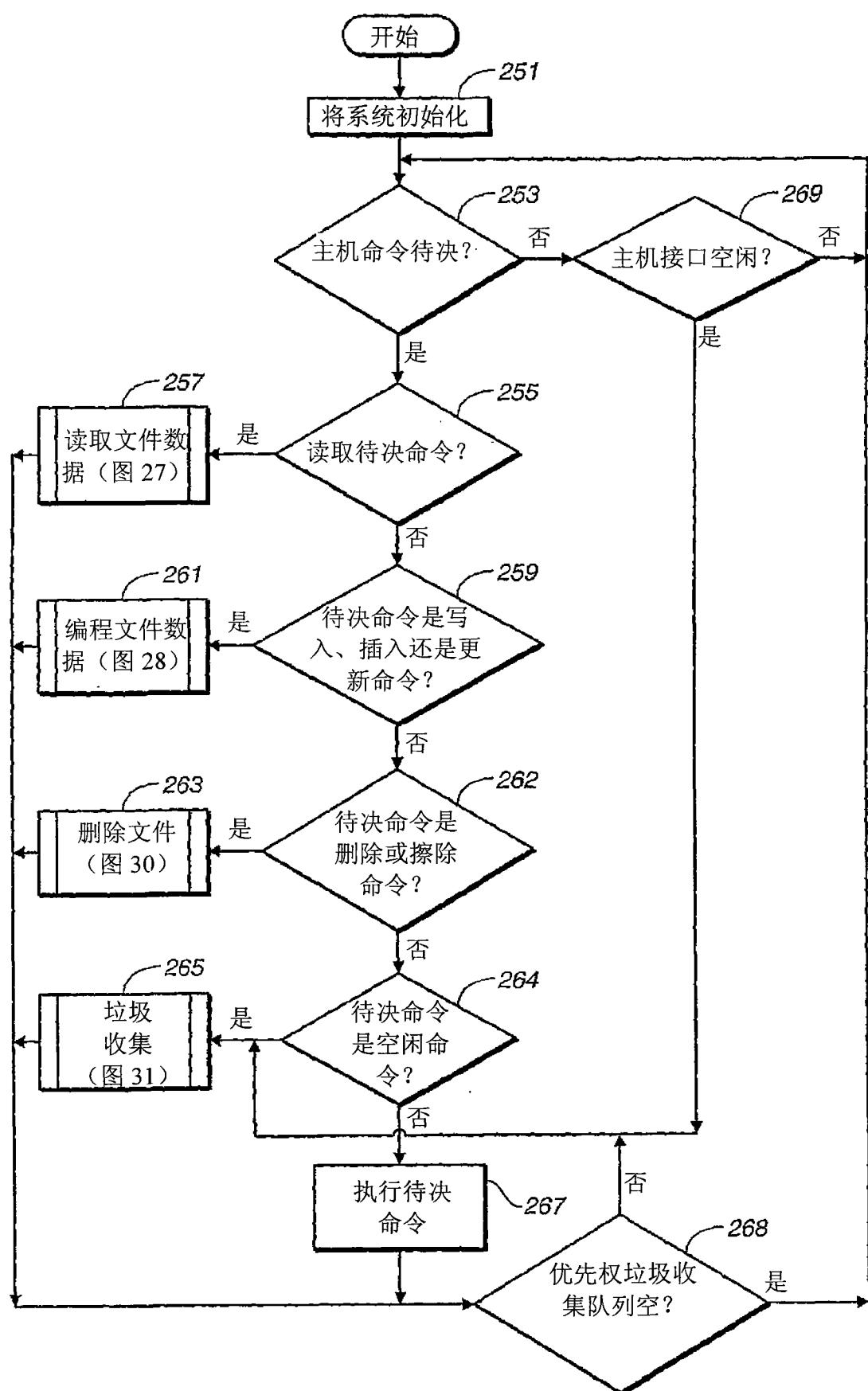


图 26

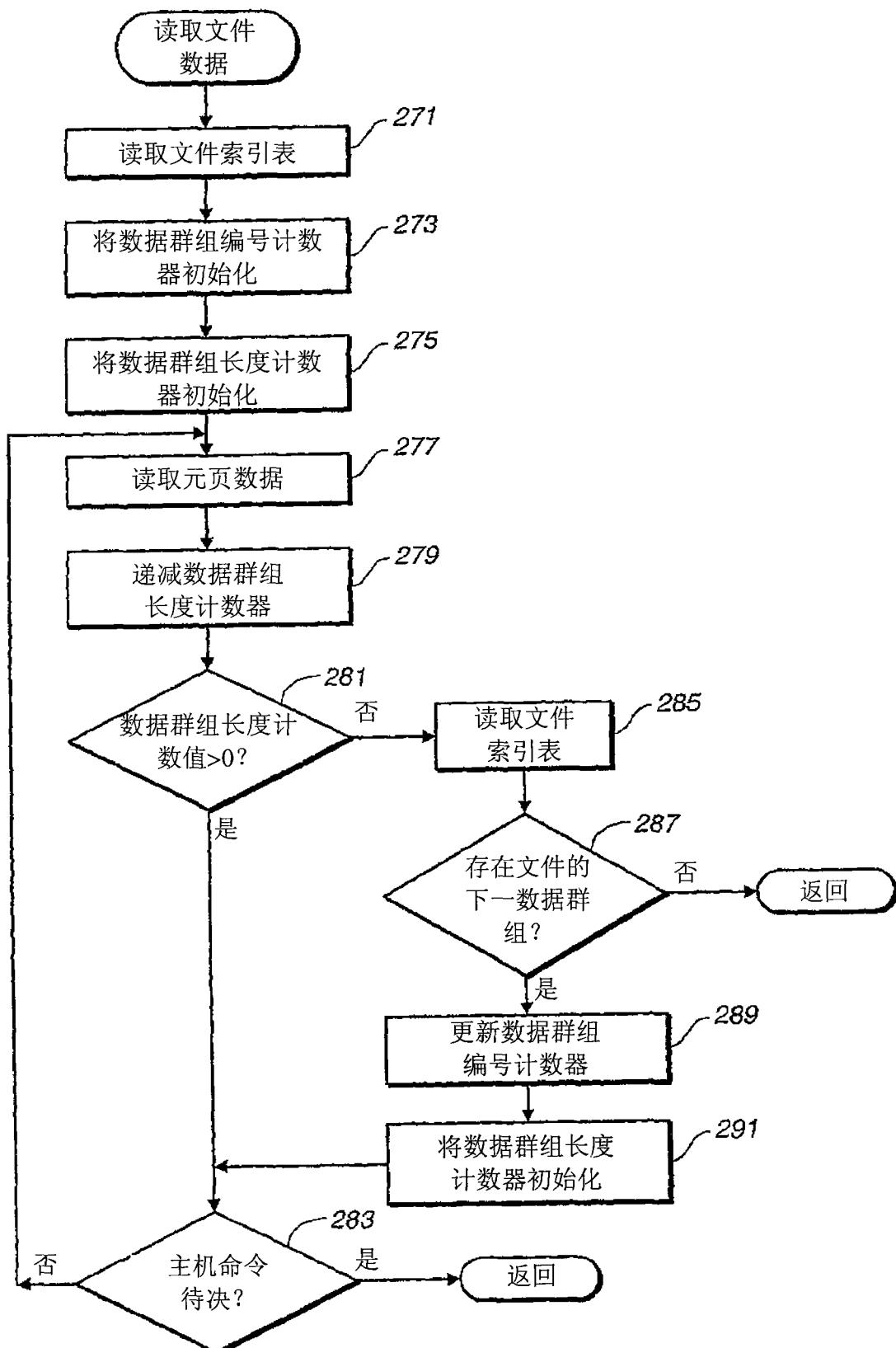
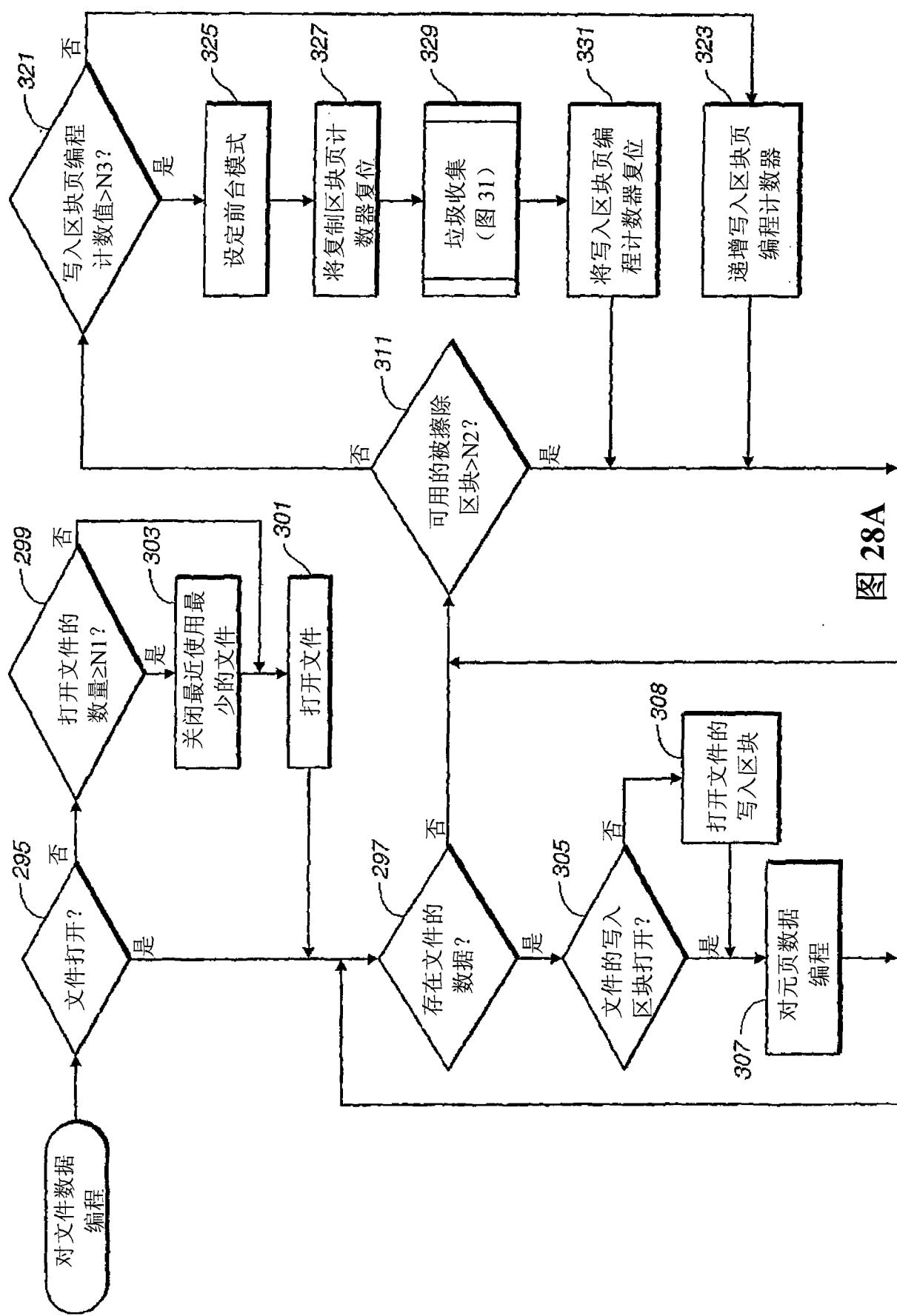
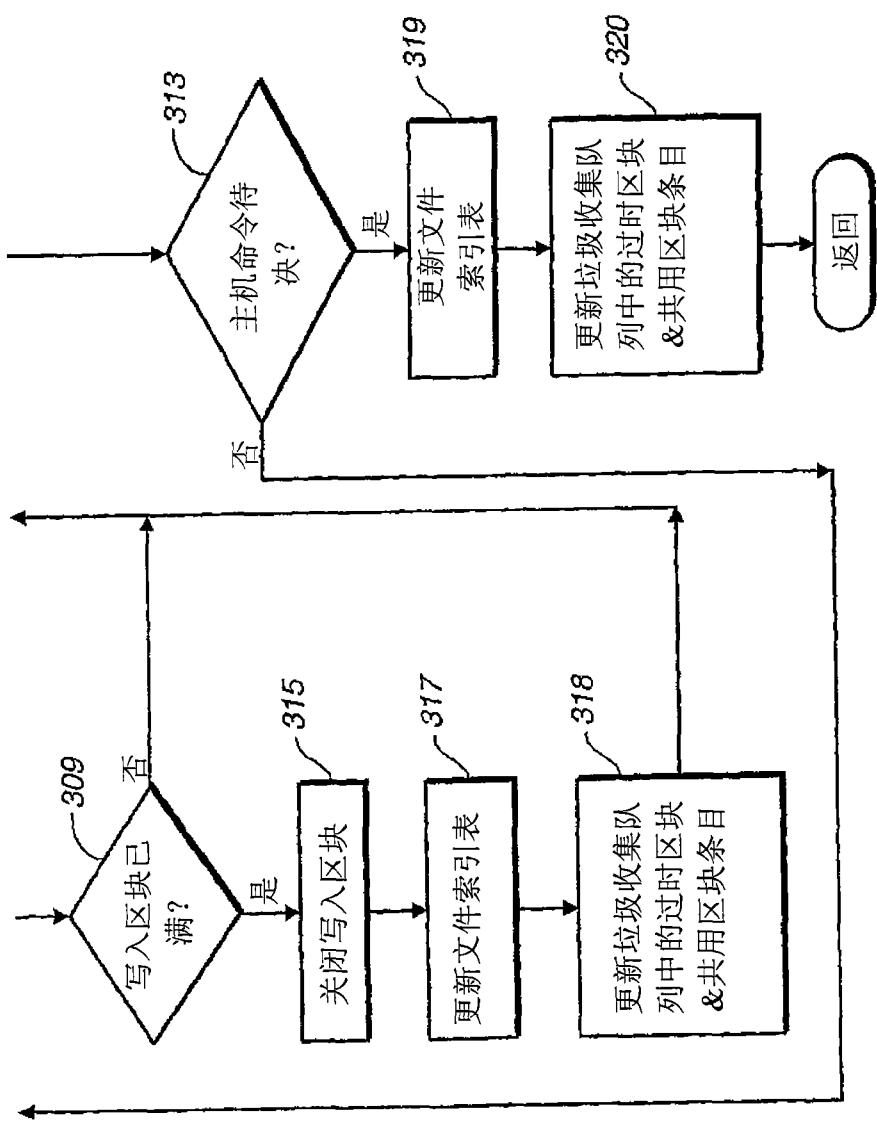


图 27





28B
四

28

28A

图 28B

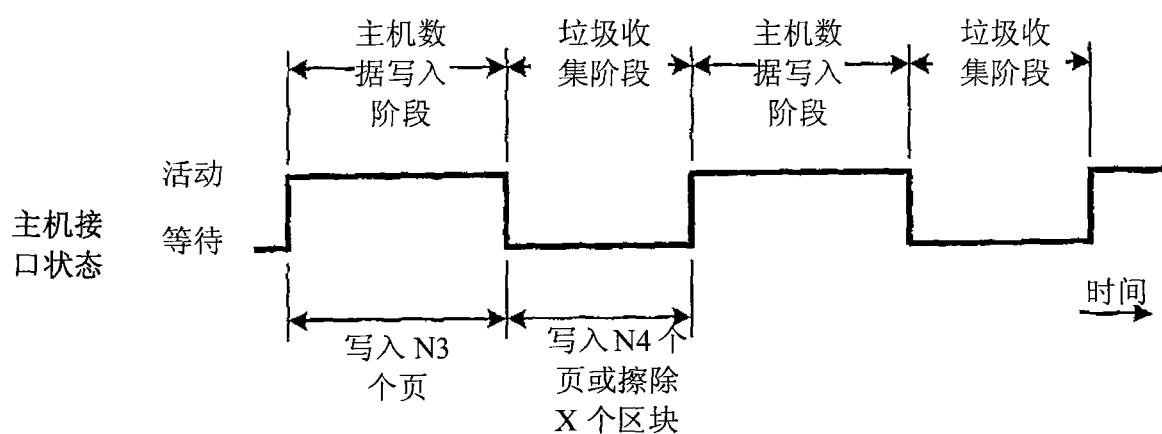


图 29

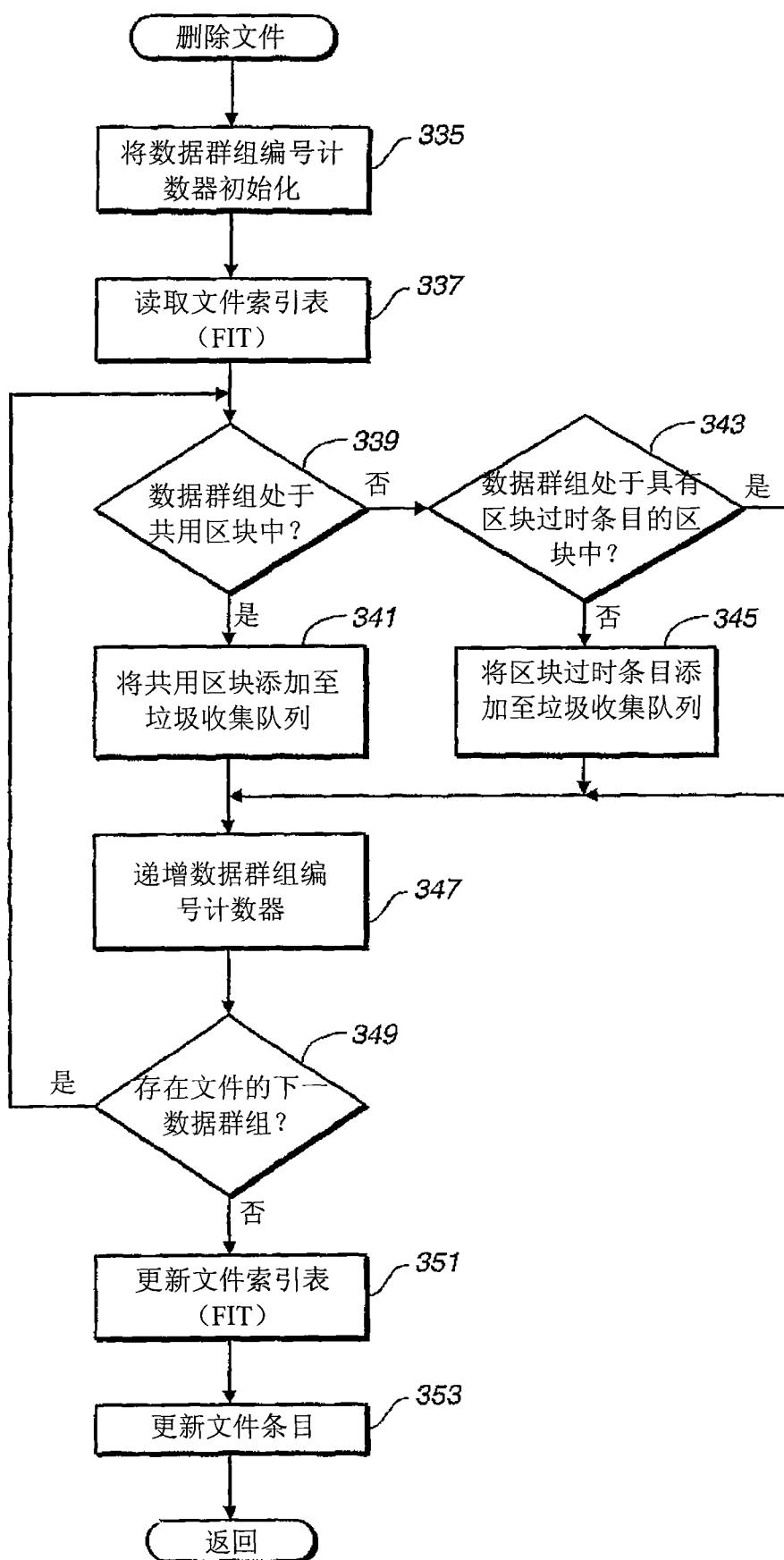
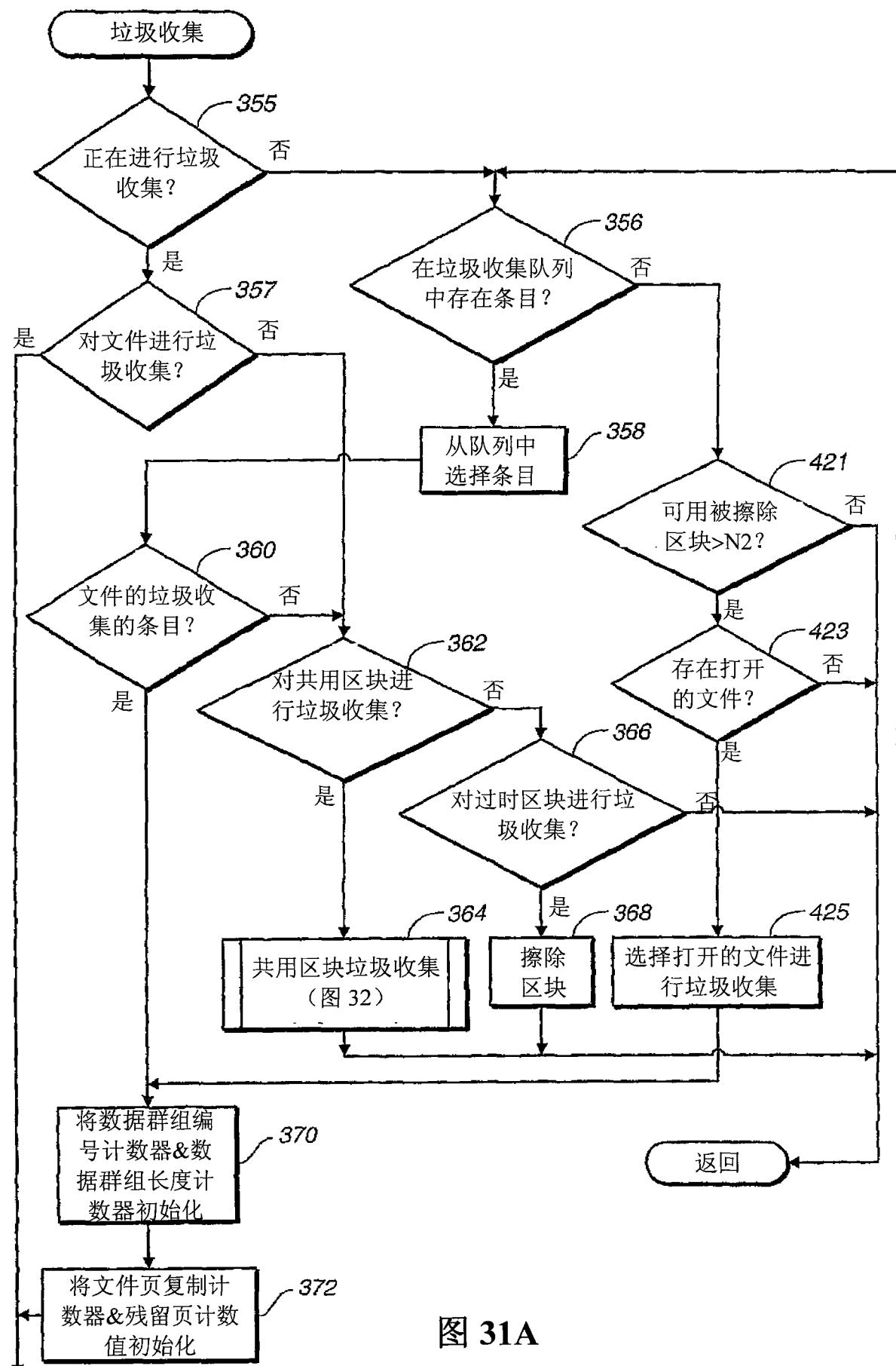


图 30



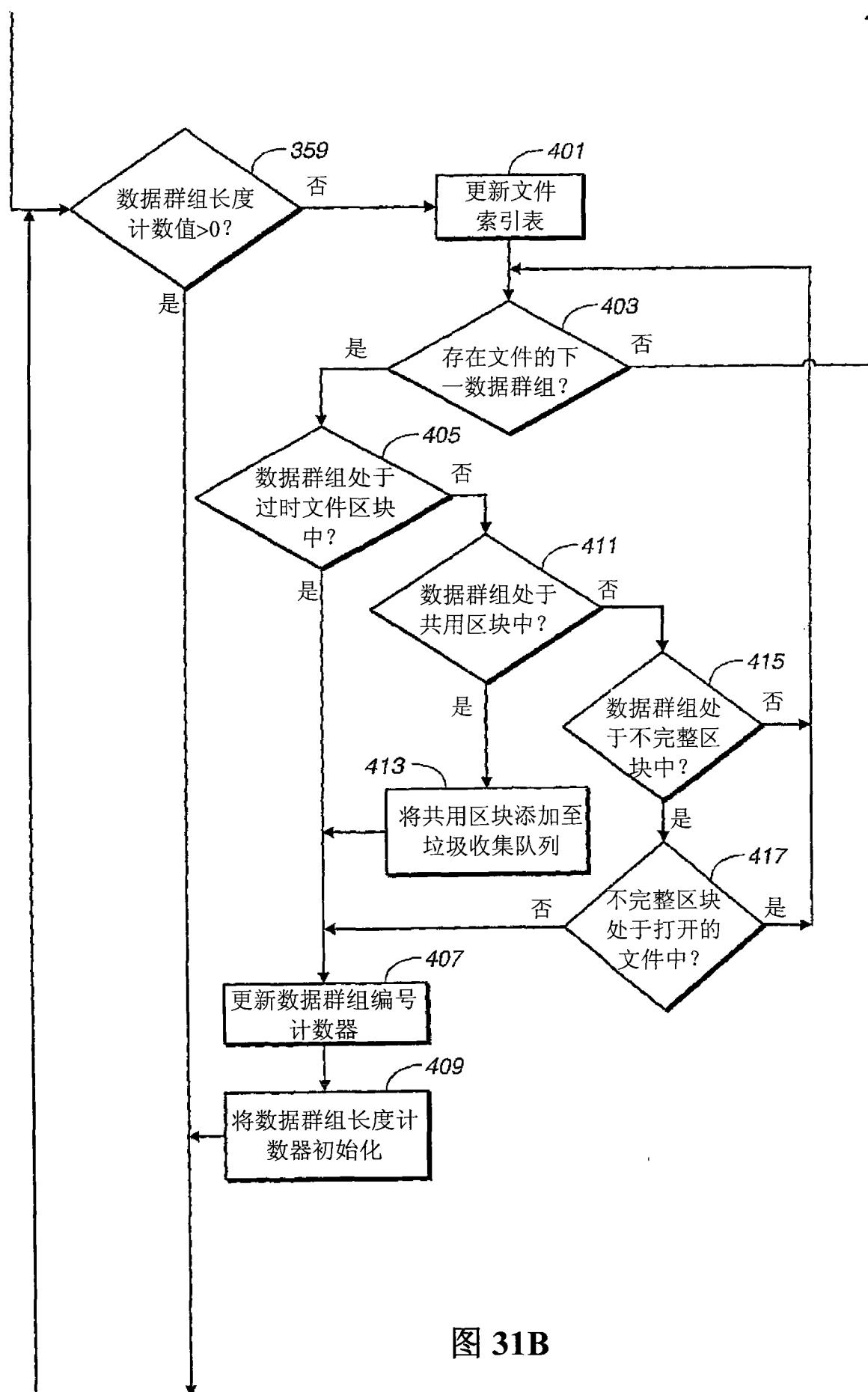


图 31B

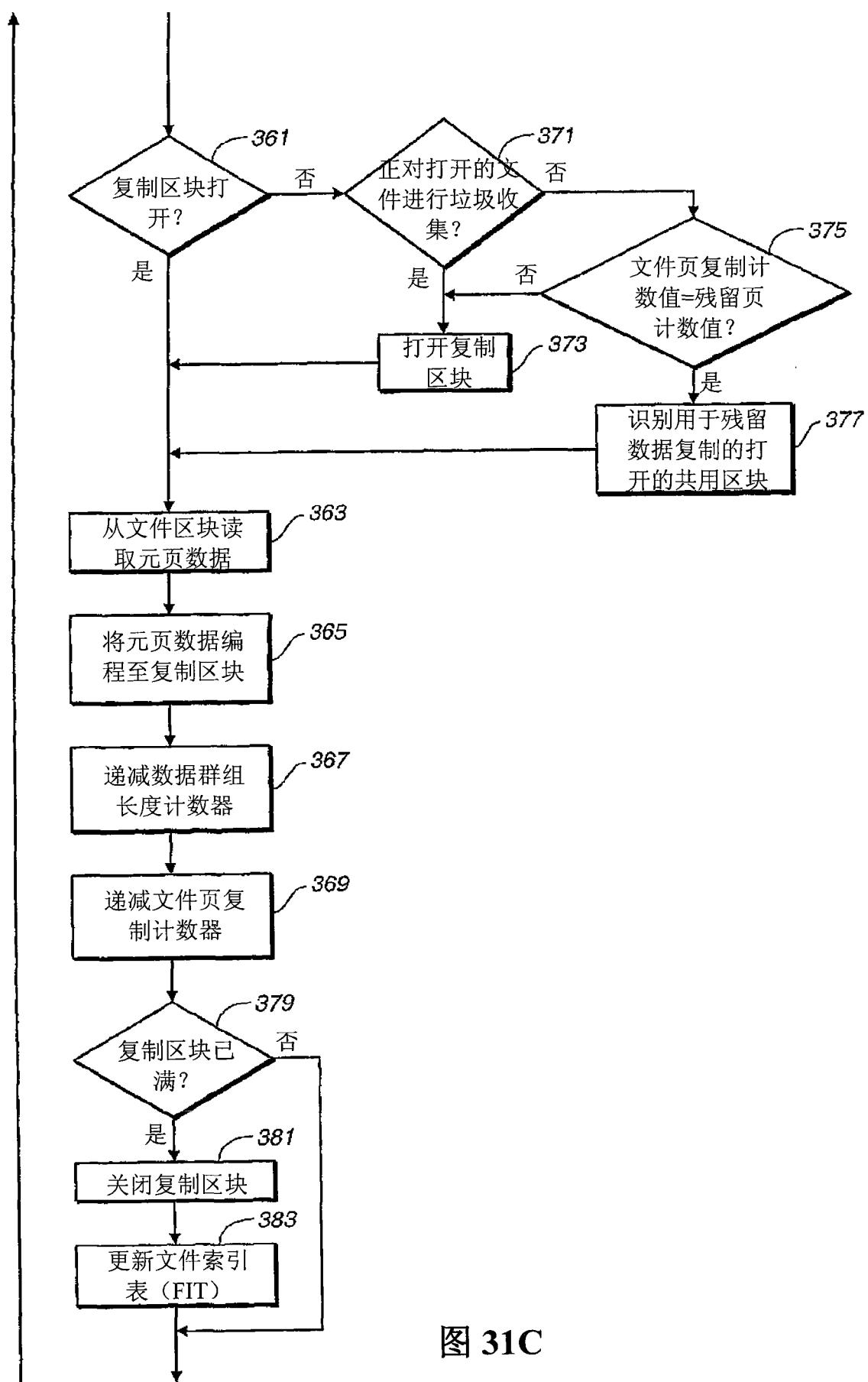


图 31C

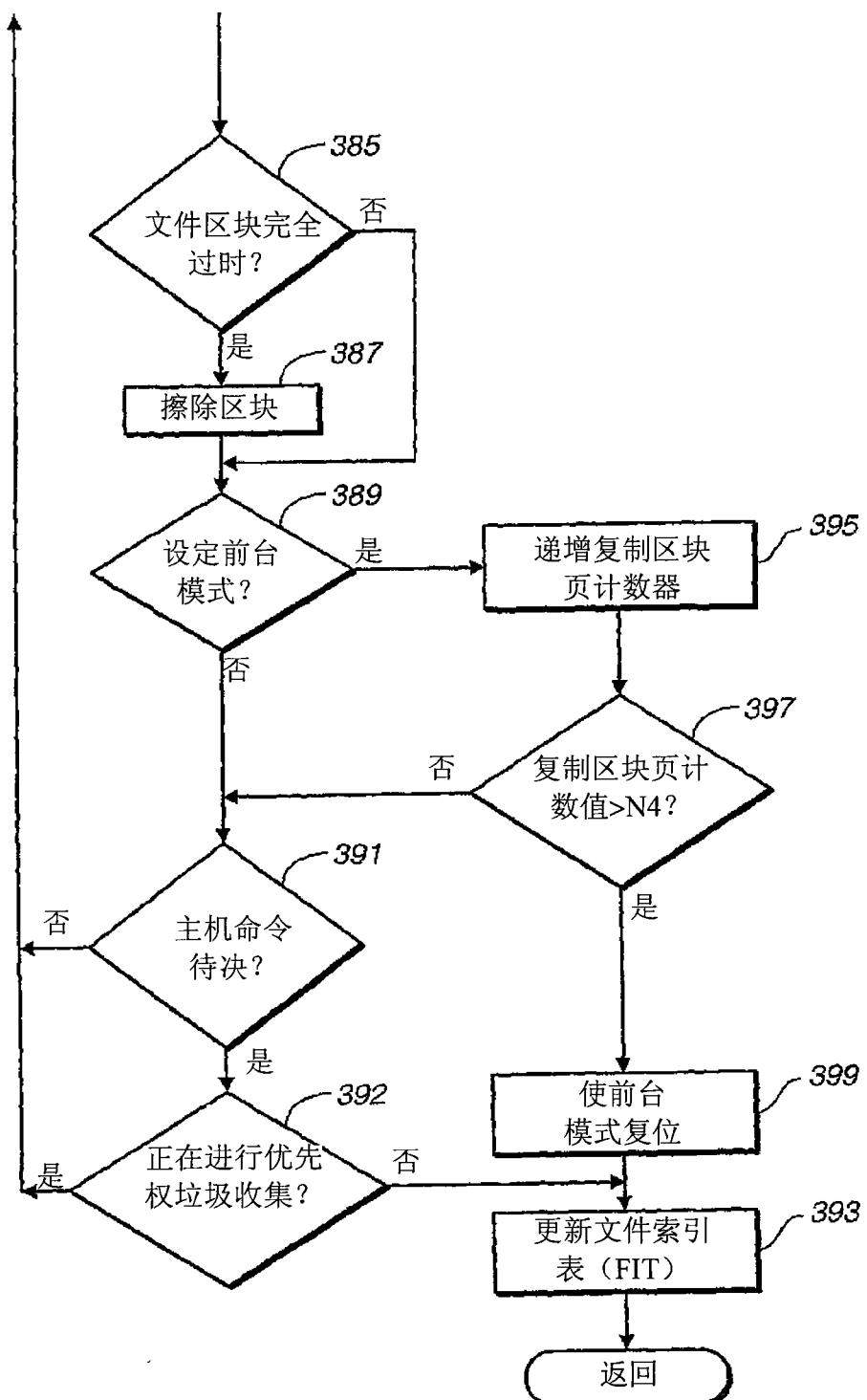


图 31A

图 31B

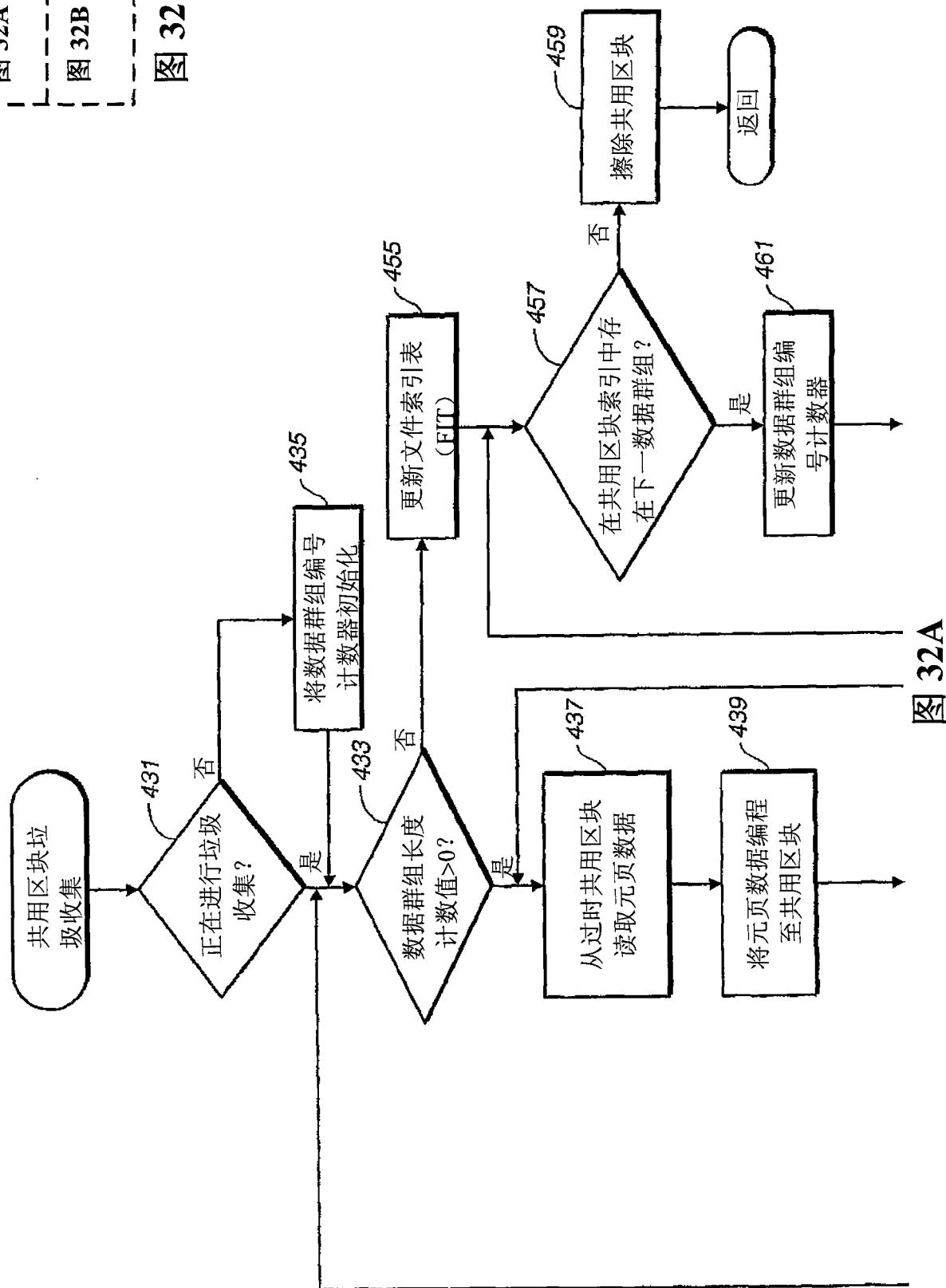
图 31C

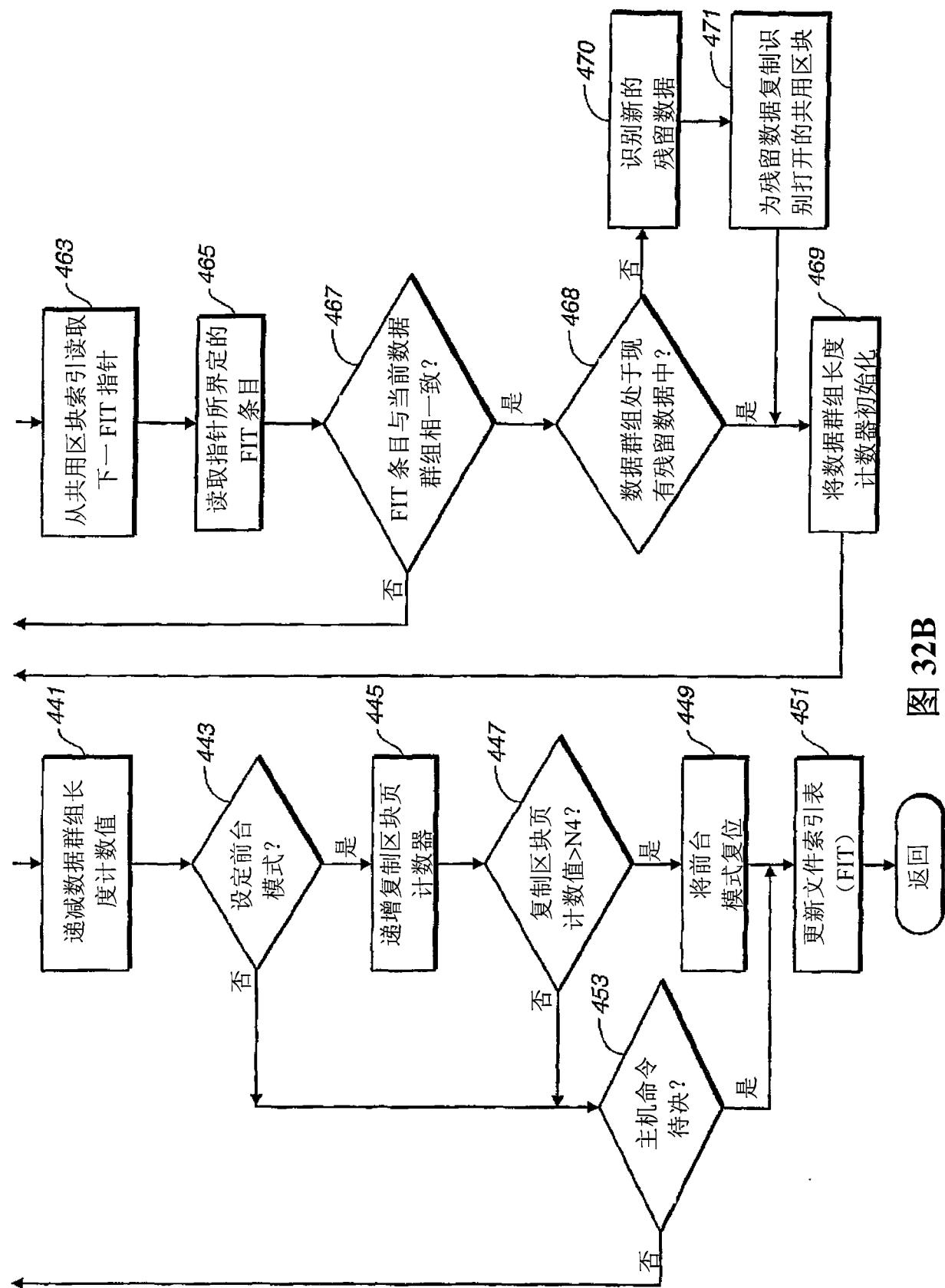
图 31D

图 31

图 31D

图 32A
图 32B
图 32





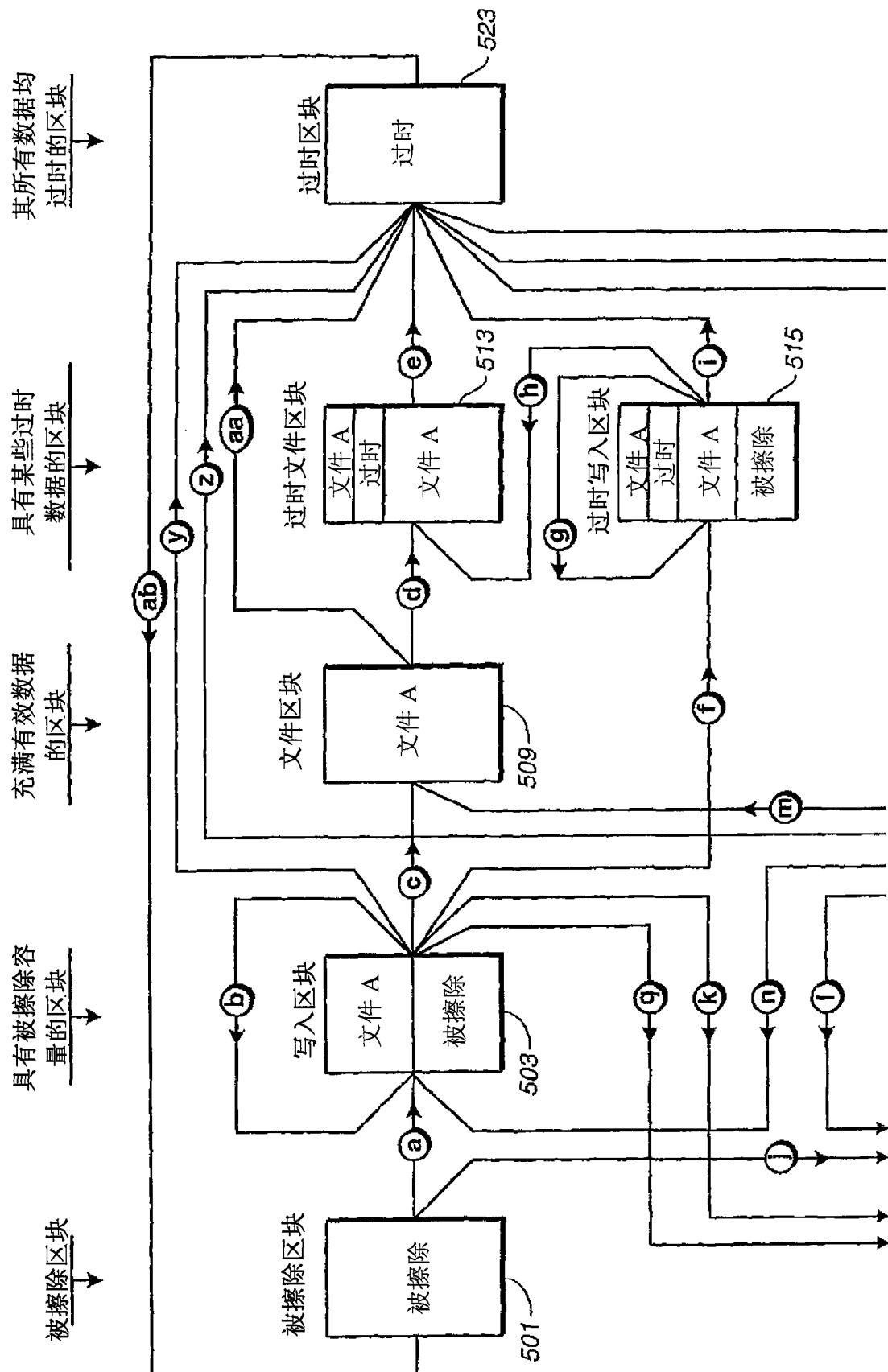


图 33A

