

發明專利說明書

(本說明書格式、順序及粗體字，請勿任意更動，※記號部分請勿填寫)

※ 申請案號：93141426

※ 申請日期：93.12.30.

※IPC 分類：G06F 12/02

一、發明名稱：(中文/英文)

G11C 7/00 (2006.01)

具非循序更新區塊管理之非揮發性記憶體及方法

NON-VOLATILE MEMORY AND METHOD WITH
NON-SEQUENTIAL UPDATE BLOCK MANAGEMENT

二、申請人：(共1人)

姓名或名稱：(中文/英文)

1. 美商桑迪士克股份有限公司
SANDISK CORPORATION

代表人：(中文/英文)

1. 查爾斯 樊 歐登
VAN ORDEN, CHARLES
2. 梅根 康普特
COMFORT, MEGAN

住居所或營業所地址：(中文/英文)

美國加州桑尼貝市卡斯本可特140號
140 CASPIAN COURT, SUNNYVALE, CA 94089, U. S. A.

國 籍：(中文/英文)

美國 U.S.A.

三、發明人：(共 3 人)

姓 名：(中文/英文)

1. 艾倫 維希 辛克萊爾
SINCLAIR, ALAN WELSH
2. 瑟蓋 安那托里維屈 葛洛貝斯
GOROBETS, SERGEY ANATOLIEVICH
3. 艾倫 大衛 班尼特
BENNETT, ALAN DAVID

國 籍：(中文/英文)

- 1.3.均英國 UNITED KINGDOM
- 2.俄羅斯 RUSSIA

四、聲明事項：

☐ 主張專利法第二十二條第二項 ☐ 第一款或 ☐ 第二款規定之事實，其事實發生日期為： 年 月 日。

☒ 申請前已向下列國家（地區）申請專利：

【格式請依：受理國家（地區）、申請日、申請案號 順序註記】

☒ 有主張專利法第二十七條第一項國際優先權：

1. 美國；2003年12月30日；10/750,155

2. 美國；2004年08月13日；10/917,867

☐ 無主張專利法第二十七條第一項國際優先權：

1.

2.

☐ 主張專利法第二十九條第一項國內優先權：

【格式請依：申請日、申請案號 順序註記】

☐ 主張專利法第三十條生物材料：

☐ 須寄存生物材料者：

國內生物材料 【格式請依：寄存機構、日期、號碼 順序註記】

國外生物材料 【格式請依：寄存國家、機構、日期、號碼 順序註記】

☐ 不須寄存生物材料者：

所屬技術領域中具有通常知識者易於獲得時，不須寄存。

九、發明說明：

【發明所屬之技術領域】

本發明一般有關非揮發性半導體記憶體，尤其有關具有記憶體區塊管理系統的非揮發性半導體記憶體。

【先前技術】

能夠進行電荷之非揮發性儲存的固態記憶體，尤其是屬於封裝成小型記憶卡之EEPROM及快閃EEPROM的形式，近來已成為各種行動及手持裝置選擇的儲存裝置，特別是資訊家電及消費性電子產品。不像也是固態記憶體的RAM(隨機存取記憶體)，快閃記憶體係為非揮發性，即使在關閉電源後仍能保留其儲存的資料。還有，不像ROM(唯讀記憶體)，快閃記憶體和磁碟儲存裝置一樣都是可再寫。儘管成本較高，但在大量儲存應用中，快閃記憶體的使用漸增。基於旋轉磁性媒體的習用大量儲存裝置，如硬碟機及軟碟，並不適合行動及手持環境。這是因為磁碟機傾向於體積龐大，容易機械故障且具有高等待時間及高功率需求。這些不想要的屬性使得基於磁碟的儲存裝置無法在大多數的行動及可攜式應用中使用。另一方面，既為內嵌式且為抽取式記憶卡之形式的快閃記憶體在理想上極為適用於行動及手持環境，因其尺寸小、功率消耗低、高速及高可靠性等特性。

快閃EEPROM和EEPROM(電子可抹除及可程式唯讀記憶體)的相似處在於，其為能夠被抹除且能將新的資料寫入或「程式化」至其記憶體單元中的非揮發性記憶體。二者均

利用場效電晶體結構中，位在半導體基板中通道區之上且在源極及汲極區之間的浮動(未連接)導電閘極。接著在浮動閘極之上提供控制閘極。浮動閘極所保留的電荷量可控制電晶體的定限電壓特性。也就是說，就浮動閘極上給定位準的電荷而言，其中有必須在「開啟」電晶體之前施加於控制閘極的對應電壓(定限)，以允許在其源極及汲極區之間的導電。尤其，如快閃EEPROM的快閃記憶體允許同時抹除記憶體單元的整個區塊。

浮動閘極可以保持某個電荷範圍，因此，可經程式化為定限電壓窗內的任何定限電壓位準。定限電壓窗的尺寸受限於裝置的最小及最大定限位準，而裝置的最小及最大定限位準對應於可經程式化至浮動閘極上的電荷範圍。定限窗一般根據記憶體裝置的特性、操作條件及記錄而定。原則上，在此窗內，各有所不同的、可解析的定限電壓位準範圍均可用於指定單元之明確的記憶體狀態。

通常會藉由兩種機制之一，將作為記憶體單元的電晶體程式化為「已程式化」狀態。在「熱電子注入」中，施加於汲極的高電壓可加速橫跨在基板通道區上的電子。同時，施加於控制閘極的高電壓可吸引熱電子通過薄閘極介電質到浮動閘極上。在「穿隧注入」中，會相對於該基板施加一高電壓給該控制閘極。依此方式，可將基板的電子吸引到中間的浮動閘極。雖然習慣使用用語「程式化」描述藉由注入電子至記憶體單元的初始抹除電荷儲存單元來寫入記憶體以改變記憶體狀態，現已和較為常見的用語，

如「寫入」或「記錄」交換使用。

可以利用下面數種機制來抹除該記憶體裝置。對EEPROM而言，藉由相對於該控制閘極施加一高電壓給該基板，致使可於該浮動閘極中誘發出電子，使其穿隧一薄氧化物進入該基板通道區(也就是，Fowler-Nordheim穿隧效應)，便可電抹除一記憶體單元。一般來說，EEPROM可以逐個位元組的方式來抹除。對快閃EEPROM而言，可每次電抹除所有的記憶體或是每次電抹除一個以上最小可抹除區塊，其中一最小可抹除區塊可能係由一個以上的區段所組成且每個區段均可儲存512個位元組以上的資料。

該記憶體裝置通常包括可被安裝於一記憶卡上的一個以上記憶體晶片。各記憶體晶片包含為周邊電路(如解碼器和抹除、寫入及讀取等電路)所支援之記憶體單元陣列。較為精密的記憶體裝置還附有執行智慧型和較高階記憶體操作及介面連接的控制器。

現今已有許多市售成功的非揮發性固態記憶體裝置。這些記憶體裝置可能係快閃EEPROM，或是可採用其它類型的非揮發性記憶體單元。快閃記憶體與系統及其製造方法的範例揭示於美國專利案第5,070,032號、第5,095,344號、第5,315,541號、第5,343,063號、第5,661,053號、第5,313,421號、以及第6,222,762號。明確地說，具NAND線串結構的快閃記憶體裝置係描述於美國專利案第5,570,315號、第5,903,495號、第6,046,935號之中。另外，亦可利用具有一用來儲存電荷之介面層的記憶體單元來製造非揮發性記憶

體裝置。其係利用一介電層來取代前面所述的導電浮動閘極元件。此等利用介電儲存元件的記憶體裝置已描述於 Eitan 等人於 2000 年 11 月在 IEEE Electron Device Letters，第 21 冊，第 11 號，第 543-545 頁中所發表的「NROM: A Novel Localized Trapping, 2-Bit Nonvolatile Memory Cell」一文中。有一 ONO 介電層會延伸跨越源極與汲極擴散區間的通道。其中一個資料位元的電荷會於靠近該汲極的介電層中被局部化，而另一個資料位元的電荷會於靠近該源極的介電層中被局部化。舉例來說，美國專利案第 5,768,192 號及第 6,011,725 號便揭示一種於兩層二氧化矽層間夾放一陷捕介電質的非揮發性記憶體單元。藉由分開讀取該介電質內空間分離的電荷儲存區域的二進制狀態，便可實現多重狀態的資料儲存。

為了提高讀取及程式化效能，會平行讀取或程式化陣列中的多個電荷儲存元件或記憶體電晶體。因此，會一起讀取或程式化記憶體元件的「頁面」。在現有的記憶體架構中，一列通常含有若干交錯的頁面或一列可構成一個頁面。一個頁面的所有記憶體元件會被一起讀取或程式化。

在快閃記憶體系統中，抹除操作可能需要多達長於讀取及程式化操作的數量級。因此，希望能夠具有大尺寸的抹除區塊。依此方式，即可降低總數很大之記憶體單元上的抹除時間。

依照快閃記憶體的性質，資料必須寫入已抹除的記憶體位置。如果要更新主機之特定邏輯位址的資料，一個方式

是將更新資料再寫入相同的實體記憶體位置中。也就是說，邏輯對實體位址映射不會變更。然而，這便表示必須先抹除含有該實體位置的整個抹除區塊，然後再以更新資料寫入。此更新方法很沒有效率，因其需要抹除及再寫入整個抹除區塊，尤其在要更新的資料只佔用抹除區塊的一小部分時。此外，還會造成比較頻繁之記憶體區塊的抹除再循環，而這對此類型記憶體裝置的有限耐久性而言並不理想。

另一個管理快閃記憶體系統的問題是必須處理系統控制及目錄資料。各種記憶體操作的過程期間會產生且存取該資料。因此，其有效處理及迅速存取將直接影響效能。由於快閃記憶體是用來儲存且是非揮發性，因此希望能夠在快閃記憶體中維持此類型的資料。然而，因為在控制器及快閃記憶體之間有中間檔案管理系統，所以無法直接存取資料。還有，系統控制及目錄資料傾向於成為作用中及成為片段，而這對於具有大尺寸區塊抹除之系統中的儲存毫無助益。照慣例，會在控制器RAM中設定此類型的資料，藉此允許由控制器直接存取。在開啟記憶體裝置的電源後，初始化程序會掃描快閃記憶體，以編譯要放在控制器RAM中的必要系統控制及目錄資訊。此程序既耗時又需要控制器RAM容量，對於持續增加的快閃記憶體容量更是如此。

US 6,567,307揭露一種在大抹除區塊中處理區段更新的方法，其包括在多個當作可用記憶區的抹除區塊中記錄更

新資料，及最後在各種區塊中彙總有效區段，然後按邏輯循序順序重新排列有效區段後再將其寫入。依此方式，在每個最微小的更新時，不必抹除及再寫入區塊。

WO 03/027828及WO 00/49488均揭露一種處理大抹除區塊中更新的記憶體系統，其中包括以區域為單位分割邏輯區段位址。小區域的邏輯位址範圍係保留用於和另一個用於使用者資料之區域分開的作用中系統控制資料。依此方式，在其自己區域中的系統控制資料操控便不會和另一區域中關聯的使用者資料互相作用。更新屬於邏輯區段層級，而且寫入指標會指向要寫入之區塊中的對應實體區段。映射資訊會被緩衝在RAM中，且最後被儲存在主記憶體的區段配置表中。邏輯區段的最新版本會淘汰現有區塊中的所有先前版本，這些區塊因此變成部分淘汰。執行廢棄項目收集可保持部分淘汰區塊為可接受的數量。

先前技術的系統傾向於使更新資料分布在許多區塊上或更新資料會使許多現有的區塊受到部分淘汰。結果通常是大量為部分淘汰之區塊所需的廢棄項目收集，這很沒有效率且會造成記憶體提早老化。還有，和非循序更新相比，更缺少有系統及有效的方式來處理循序更新。

因此，普遍需要高容量及高效能的非揮發性記憶體。尤其，更需要有能夠在大區塊中執行記憶體操作且沒有上述問題的高容量非揮發性記憶體。

【發明內容】

一種非揮發性記憶體系統係按照實體記憶體位置的實體

群組來組織。各實體群組(中繼區塊)為可抹除的單元且能用來儲存一個邏輯群組的資料。記憶體管理系統允許藉由配置記錄邏輯群組之更新資料專用的中繼區塊來更新一個邏輯群組的資料。更新中繼區塊按照所接收的順序記錄更新資料，且對記錄按照原始儲存的正確邏輯順序(循序)與否(混亂)並沒有限制。最後會關閉更新中繼區塊以進行其他記錄。會發生若干程序之一，但最終會以按照正確順序完全填滿之取代原始中繼區塊的中繼區塊結束。在混亂的例子中，會以有助於經常更新的方式將目錄資料維持在非揮發性記憶體中。系統支援多個同時更新的邏輯群組。

本發明的一個特色允許以逐個邏輯群組的方式來更新資料。因此，在更新邏輯群組時，會限制邏輯單元的分布(還有更新淘汰之記憶體單元的散佈)範圍。這在邏輯群組通常含在實體區塊內時更是如此。

在邏輯群組的更新期間，通常必須指派一或兩個緩衝已更新之邏輯單元的區塊。因此，只需要在相對較少數量的區塊上執行廢棄項目收集。藉由彙總或壓縮即可執行混亂區塊的廢棄項目收集。

相較於循序區塊來說，更新處理的經濟效能於更新區塊的一般處理中會愈加明顯，因而便無須為混亂(非循序)更新配置任何額外的區塊。所有的更新區塊均被配置為循序的更新區塊，而且任何的更新區塊均可被變更成混亂的更新區塊。更確切地說，可任意地將一更新區塊從循序式變更成混亂式。

有效使用系統資源允許同時進行更新多個邏輯群組。這可進一步增加效率及減少過度耗用。

分散在多個記憶體平面上的記憶體對齊

根據本發明的另一方面，對於一被組織成複數個可抹除區塊且由多個記憶體平面所構成(因而可平行地讀取複數個邏輯單元或是將複數個邏輯單元平行地程式化至該等多個平面之中)的記憶體陣列，當要更新儲存於特定記憶體平面中第一區塊的原始邏輯單元時，會供應所需以將已更新的邏輯單元保持在和原始相同的平面中。這可藉由以下方式來完成：將已更新的邏輯單元記錄到仍在相同平面中之第二區塊的下一個可用位置。較佳將邏輯單元儲存在平面中和其中其他版本相同的位移位置，使得給定邏輯單元的所有版本係由相同組的感測電路予以服務。

據此，在一項較佳具體實施例中，以邏輯單元的目前版本來填補介於上一個程式化記憶體單元與下一個可用平面對齊記憶體單元之間的任何中間間隙。將邏輯上位於該最後被程式化之邏輯單元後面的該等邏輯單元的目前版本以及邏輯上位於被儲存在該下一個可用的平面排列記憶體單元中之邏輯單元前面的該等邏輯單元的目前版本填入間隙中，便可完成該填補作業。

依此方式，可將邏輯單元的所有版本維持在具有和原始相同位移的相同平面中，致使在廢棄項目收集操作中，不必從不同平面擷取邏輯單元的最新版本，以免降低效能。在一項較佳具體實施例中，可利用該等最新的版本來更新

或填補該平面上的每個記憶體單元。因此，便可從每個平面中平行地讀出一邏輯單元，其將會具有邏輯順序而無需進一步重新排列。

此方案藉由允許平面上重新排列邏輯群組之邏輯單元的最新版本，且不必收集不同記憶體平面的最新版本，而縮短彙總混亂區塊的時間。這很有好處，其中主機介面的效能規格可定義由記憶體系統完成區段寫入操作的最大等待時間。

階段性程式錯誤處置

根據本發明的另一方面，在具有區塊管理系統的記憶體中，在時間緊急的記憶體操作期間，區塊中的程式失敗可藉由繼續中斷區塊(breakout block)中的程式化操作來處置。稍後，在較不緊急的時間，可將中斷前記錄在失敗區塊中的資料傳送到其他可能也是中斷區塊的區塊。接著即可丟棄失敗的區塊。依此方式，在遇到缺陷區塊時，不會因必須立刻傳送缺陷區塊中儲存的資料而損失資料及超過指定的時間限制，即可加以處理。此錯誤處置對於廢棄項目收集操作尤其重要，因此在緊急時間期間不需要對一嶄新的區塊重複進行整個作業。其後，在適宜的時間，藉由重新配置到其他區塊，即可挽救缺陷區塊的資料。

程式失敗處置在彙總操作期間尤其重要。正常的彙總操作可將常駐在原始區塊及更新區塊中之邏輯群組的所有邏輯單元的目前版本彙總至彙總區塊。在彙總操作期間，如果在彙總區塊中發生程式失敗，則會提供另一個當作中斷

彙總區塊的區塊，以接收其餘邏輯單元的彙總。依此方式，不必複製邏輯單元一次以上，而仍可在正常彙總操作指定的期間內完成例外處理的操作。在適宜的時間，將群組所有未處理完成之邏輯單元彙總至中斷區塊中，即可完成彙總操作。適宜的時間將是在目前主機寫入操作以外的一些其他有時間執行彙總之期間的期間。一個此種適宜的時間是在另一個其中有更新但無關聯之彙總操作之主機寫入的期間。

實質上，可將程式失敗處置的彙總視為以多階段來實施。在第一階段中，在發生程式失敗後，會將邏輯單元彙總至一個以上區塊中，以避免彙總各邏輯單元一次以上。在適宜的時間會完成最後階段，其中會將邏輯群組彙總至一個區塊中，較佳藉由按循序順序將所有邏輯單元收集至中斷彙總區塊中。

非循序更新區塊索引

根據本發明的另一方面，在具有支援具非循序邏輯單元之更新區塊之區塊管理系統的非揮發性記憶體中，非循序更新區塊中邏輯單元的索引被緩衝儲存在RAM中，並定期將其儲存至非揮發性記憶體中。在一項具體實施例中，將索引儲存在專用於儲存索引的區塊中。在另一項具體實施例中，將索引儲存在更新區塊本身中。在又另一項具體實施例中，將索引儲存在各邏輯單元的標頭中。在另一方面中，在上一個索引更新之後但在下一個索引更新之前寫入的邏輯單元會將其索引資訊儲存在各邏輯單元的標頭中。

依此方式，在電源中斷後，不必在初始化期間執行掃描，即可決定最近寫入之邏輯單元的位置。在又另一方面中，將區塊管理成部分循序及部分非循序指向一個以上邏輯子群組。

控制資料完整性與管理

根據本發明的另一方面，如部分或全部控制資料的關鍵資料如果被維持在複製項中，則保證額外等級的可靠性。複製的執行方式對於採用兩次編碼過程(two-pass)程式化技術以連續程式化相同組記憶體單元之多位元的多重狀態記憶體系統而言，第二次編碼過程中的任何程式化錯誤都無法毀損第一次編碼過程建立的資料。複製還有助於偵測寫入中止、偵測誤測(即，兩個複本有良好的ECC但資料不同)，且可增加額外等級的可靠性。若干資料複製的技術均已考慮。

在一項具體實施例中，在稍早程式化編碼過程中程式化給定資料的兩個複本後，後續程式化編碼過程可避免程式化用於儲存該等兩個複本中至少一個的記憶體單元。依此方式，在後續程式化編碼過程在完成之前中止及毀損稍早編碼過程的資料時，該等兩個複本中至少一個也不會受到影響。

在另一項具體實施例中，某一給定資料的兩個複本會被儲存於兩個不同的區塊中，而且該等兩個複本中至多僅有其中一個的記憶體單元會於後面的程式化編碼過程中被程式化。

於另一具體實施例中，於一程式化編碼過程中儲存某一給定資料的兩個複本之後，便不再對用於儲存該等兩個複本的記憶體單元組實施任何進一步的程式化。於該記憶體單元組的最終程式化編碼過程中來程式化該等兩個複本便可達成此目的。

在又另一項具體實施例中，可於二進制程式化模式中將某一給定資料的該等兩個複本程式化至一多重狀態的記憶體之中，致使不會對該等已程式化的記憶體單元進行任何進一步的程式化。

在又另一項具體實施例中，對於採用兩次編碼過程程式化技術以連續程式化相同組記憶體單元之多位元的多重狀態記憶體系統而言，會採用容錯碼以編碼多個記憶體狀態，使稍早程式化編碼過程所建立的資料不會受到後續程式化編碼過程中錯誤的影響。

根據本發明的另一方面，在具有區塊管理系統的非揮發性記憶體中，可實施記憶體區塊的「控制廢棄項目收集」或先佔式重新配置，以避免發生大量的更新區塊均恰巧同時需要進行重新配置的情形。例如，在更新用於控制區塊管理系統操作的控制資料時會發生此情況。控制資料類型的層級可和不同程度的更新次數共存，導致其關聯的更新區塊需要不同速率的廢棄項目收集或重新配置。會有一個以上控制資料類型之廢棄項目收集操作同時發生的特定次數。在極端的情況中，所有控制資料類型之更新區塊的重新配置階段會進行整頓，導致所有的更新區塊都需要同時

重新配置。

參考本發明以下結合附圖之較佳具體實施例的說明，即可瞭解本發明的其他特色及優點。

【實施方式】

圖1以示意圖顯示適於實施本發明之記憶體系統的主要硬體組件。記憶體系統20通常透過主機介面以主機10操作。記憶體系統通常形式為記憶卡或內嵌式記憶體系統。記憶體系統20包括由控制器100控制操作的記憶體200。記憶體200包含分布於一或多個積體電路晶片上的一或多個陣列的非揮發性記憶體單元。控制器100包括：介面110、處理器120、選用性副處理器121、ROM 122(唯讀記憶體)、RAM 130(隨機存取記憶體)、及選用性可程式非揮發性記憶體124。介面110有一個連接控制器和主機的組件及另一個連接至記憶體200的組件。儲存於非揮發性ROM 122及/或選用性非揮發性記憶體124中的韌體可提供處理器120程式碼以實施控制器100的功能。處理器120或選用性副處理器121可處理錯誤校正碼。在一項替代性具體實施例中，控制器100可藉由狀態機器(未顯示)來實施。在又另一項具體實施例中，控制器100可在主機內實施。

邏輯與實體區塊結構

圖2為根據本發明一較佳具體實施例的記憶體，其係被組織成數個實體區段群組(或中繼區塊)並且由該控制器的記憶體管理器來管理。該記憶體200會被組織成數個中繼區塊，其中每個中繼區塊係一群組之可一起抹除的實體區段

S_0 、...、 S_{N-1} 。

主機10可在檔案系統或作業系統下執行應用程式時存取記憶體200。一般來說，該主機會以邏輯區段為單位來定址資料，其中，例如，各區段可含有512位元組的資料。還有，主機通常亦會以邏輯叢集為單位來讀取或寫入該記憶體系統，各邏輯叢集由一或多個邏輯區段所組成。在部分主機系統中，可存在選用性主機端記憶體管理器，以執行主機的較低階記憶體管理。在大部分的例子中，在讀取或寫入操作期間，主機10實質上會對記憶體系統20發出指令，以讀取或寫入含有一串具連續位址之資料邏輯區段的程式段。

一記憶體端記憶體管理器被實施在該記憶體系統20的控制器100之中，用以管理於該快閃記憶體200的複數個中繼區塊中儲存與擷取主機邏輯區段的資料。於該較佳的具體實施例中，該記憶體管理器含有數個軟體模組，用於管理該等中繼區塊的抹除作業、讀取作業、以及寫入作業。該記憶體管理器還會於該快閃記憶體200及該控制器RAM 130之中維護和其作業相關的系統控制與目錄資料。

圖3A(i)-3A(iii)為根據本發明一較佳具體實施例介於一邏輯群組與一中繼區塊間之映射的概略示意圖。該實體記憶體的該中繼區塊具有N個實體區段，用於儲存一邏輯群組的N個邏輯區段資料。圖3A(i)所示的係來自邏輯群組 LG_i 的資料，其中該等邏輯區段呈現連續的邏輯順序0、1、...、N-1。圖3A(ii)所示的係正以相同的邏輯順序被儲存於該中

繼區塊中的相同資料。當依此方式儲存時，該中繼區塊便係所謂的「循序式」。一般來說，該中繼區塊可能會具有以不同順序儲存的資料，於該情況中，該中繼區塊則係所謂的「非循序式」或「混亂式」。

在邏輯群組的最低位址及其映射之中繼區塊的最低位址之間會有位移。此時，邏輯區段位址會於該中繼區塊內以環狀的方式從該邏輯群組的底部反繞回至頂端。例如，在圖3A(iii)中，該中繼區塊會在其始於邏輯區段k之資料的第一位置中進行儲存。在到達最後邏輯區段N-1時，中繼區塊會繞回至區段0，最後在其最後實體區段中儲存和邏輯區段k-1關聯的資料。在較佳具體實施例中，會使用頁面標記來識別任何位移，例如，識別在中繼區塊之第一實體區段中所儲存之資料的起始邏輯區段位址。當兩個區塊僅相差一個頁面標記時，則會認為該等兩個區塊係以相同的順序來儲存其邏輯區段。

圖3B為介於複數個邏輯群組與複數個中繼區塊間之映射的概略示意圖。每個邏輯群組均映射至一唯一的中繼區塊，除了其中的資料正在被更新的少數邏輯群組以外。一邏輯群組在被更新之後，其可能會映射至一不同的中繼區塊。可將映射資訊維持在一組邏輯對實體目錄中，稍後將會詳細說明。

亦考慮其它類型的邏輯群組至中繼區塊映射關係。舉例來說，由Alan Sinclair於和本發明同一天提出之共同待審及共同擁有的美國專利申請案，標題為「Adaptive Metablocks」

之中便揭示具有可變大小的中繼區塊。本文以引用的方式併入該共同待審申請案全部的揭示內容。

本發明的一個特色在於系統以單一邏輯分割操作，及記憶體系統的整個邏輯位址範圍中的邏輯區段群組均以相同的方式來處理。例如，可將含有系統資料的區段及含有使用者資料的區段分布在邏輯位址空間中的任何地方。

不像先前技術的系統，並無系統區段(即，有關檔案配置表、目錄或子目錄的區段)的特別分割或分區，以局部化在可能含有高次數及小尺寸之更新資料的邏輯位址空間區段中。而是，更新區段之邏輯群組的本方案會有效處理為系統區段典型且為檔案資料典型的存取模式。

圖4顯示中繼區塊和實體記憶體中結構的對齊。快閃記憶體包含可當作一個單元一起抹除之記憶體單元的區塊。此種抹除區塊為快閃記憶體之最小的抹除單元或記憶體的最小可抹除單元(MEU)。最小抹除單元是記憶體的硬體設計參數，不過，在一些支援多個MEU抹除的記憶體系統中，也可以設定包含一個以上MEU的「超級MEU」。對於快閃EEPROM，一個MEU可包含一個區段，但較佳包含多個區段。在所示的範例中，其具有M個區段。在較佳具體實施例中，各區段可以儲存512位元組的資料且具有使用者資料部分及用於儲存系統或附加項資料的標頭部分。如果中繼區塊係以P個MEU所構成且各MEU含有M個區段，則各中繼區塊將具有 $N = P * M$ 個區段。

在系統層級，中繼區塊代表記憶體位置的群組，如，可

一起抹除的區段。快閃記憶體實體位址空間會被處理成一組中繼區塊，其中中繼區塊是最小的抹除單元。於本份說明書內，「中繼區塊」與「區塊」等詞語係同義詞，用來定義媒體管理於系統層級的最小抹除單位，而「最小抹除單位」或MEU一詞則係用來表示快閃記憶體的最小抹除單位。

連結數個最小抹除單位(MEU)以構成中繼區塊

為了最大化程式化速度及抹除速度，會儘可能利用平行方式，其係藉由配置多個要平行程式化的頁面資訊(位在多個MEU中)，及配置多個要平行抹除的MEU。

在快閃記憶體中，一個頁面是可在單一操作中一起程式化之記憶體單元的群組。一個頁面可包含一或多個區段。還有，可將記憶體陣列分割成一個以上平面，其中一次只能程式化或抹除一個平面內的一個MEU。最後，可在一或多個記憶體晶片中分布各平面。

在快閃記憶體中，MEU可包含一或多個頁面。可將快閃記憶體晶片內的數個MEU按平面來組織。由於可同時程式化或抹除各平面的一個MEU，因此有利於從各平面選擇一個MEU以形成多個MEU中繼區塊(請見下文圖5B)。

圖5A顯示從連結不同平面之最小抹除單元所構成的中繼區塊。各中繼區塊(例如MB0、MB1、...)均係以記憶體系統之不同平面的數個MEU所構成，其中不同平面可分布在一或多個晶片上。圖2所示的中繼區塊連結管理器170可管理各中繼區塊之MEU的連結。如果MEU之一沒有失敗，在初

始格式化程序期間設定各中繼區塊，並在整個系統壽命保留其組成的MEU。

圖5B顯示從各平面選擇一最小抹除單元(MEU)以連結成中繼區塊的一項具體實施例。

圖5C顯示其中從各平面選擇一個以上MEU以連結成中繼區塊的另一項具體實施例。在另一項具體實施例中，可從各平面選擇一個以上MEU以形成一超級MEU。例如，舉例來說，一超級MEU可能係由兩個MEU所構成的。此時，會採取一次以上編碼過程以進行讀取或寫入操作。

由Carlos Gonzales等人於和本發明同一天提出之共同待審及共同擁有的美國專利申請案，標題為「Adaptive Deterministic Grouping of Blocks into Multi-Block Structures」之中亦揭示將複數個MEU連結及再連結成中繼區塊。本文以引用的方式併入該共同待審申請案全部的揭示內容。

中繼區塊管理

圖6為如控制器及快閃記憶體中實施之中繼區塊管理系統的示意方塊圖。中繼區塊管理系統包含實施於控制器100中的各種功能模組，並在依階層式分布於快閃記憶體200及控制器RAM 130的表格及清單中維持各種控制資料(包括目錄資料)。實施於控制器100中的功能模組包括：介面模組110、邏輯對實體位址轉譯模組140、更新區塊管理器模組150、抹除區塊管理器模組160及中繼區塊連結管理器170。

介面 110 允許中繼區塊管理系統介接主機系統。邏輯對實體位址轉譯模組 140 將主機的邏輯位址映射至實體記憶體位置。更新區塊管理器模組 150 管理記憶體中給定之資料邏輯群組的資料更新操作。已抹除區塊管理器 160 管理中繼區塊的抹除操作及其用於儲存新資訊的配置。中繼區塊連結管理器 170 管理區段最小可抹除區塊之子群組的連結以構成給定的中繼區塊。這些模組將在其個別段落中詳細說明。

於作業期間，該中繼區塊管理系統會產生且配合控制資料(例如位址、控制與狀態資訊)來運作。由於許多控制資料傾向於是經常變更的小型資料，因此無法在具有大型區塊結構的快閃記憶體中予以隨時有效儲存及維持。為了在非揮發性快閃記憶體中儲存比較靜態的控制資料，同時在控制器 RAM 中尋找數量較少之比較有變化的控制資料，以進行更有效的更新及存取，會採用階層式與分散式的方案。在發生電源關機或故障時，此方案允許掃描非揮發性記憶體中一小組的控制資料，以在揮發性控制器 RAM 中快速重建控制資料。這之所以可行是因為本發明可限制和給定邏輯群組之資料之可能活動關聯的區塊數量。依此方式，即可限制掃描。此外，會將部分需要持久性的控制資料儲存在按區段更新的非揮發性中繼區塊中，其中各更新將會記錄取代先前區段的新區段。控制資料會採用區段索引方案以在中繼區塊中記錄按區段的更新。

非揮發性快閃記憶體 200 儲存大量相對較靜態的控制資料。這包括：群組位址表 (GAT) 210、混亂區塊索引

(CBI)220、已抹除的區塊清單(EBL)230及MAP 240。GAT 210可記錄區段之邏輯群組及其對應中繼區塊之間的映射。除非更新，否則該等映射關係不會改變。CBI 220可記錄更新期間邏輯上非循序區段的映射。EBL 230可記錄已經抹除之中繼區塊的集區。MAP 240是顯示快閃記憶體中所有中繼區塊之抹除狀態的位元對映。

揮發性控制器RAM 130儲存一小部分經常變更及存取的控制資料。這包括配置區塊清單(ABL)134及清除區塊清單(CBL)136。ABL 134可記錄中繼區塊用於記錄更新資料的配置，而CBL 136可記錄已解除配置及已抹除的中繼區塊。在較佳具體實施例中，RAM 130可當作儲存在快閃記憶體200之控制資料的快取記憶體。

更新區塊管理器

更新區塊管理器150(如圖2所示)處理邏輯群組的更新。根據本發明的一方面，會配置進行更新之區段的各邏輯群組一用於記錄更新資料的專用更新中繼區塊。在較佳具體實施例中，會將邏輯群組之一或多個區段的任何程式段記錄在更新區塊中。可管理更新區塊以接收按循序順序或非循序(又稱為「混亂」)順序的更新資料。混亂更新區塊允許邏輯群組內按任何順序更新區段資料，並可任意重複個別區段。尤其，不必重新配置任何資料區段，循序更新區塊可變成混亂更新區塊。混亂資料更新不需要任何預定的區塊配置；任何邏輯位址的非循序寫入可被自動納入。因此，不像先前技術的系統，和先前技術系統不同的係，並不必

特別處置該邏輯群組之各個更新程式段究竟係邏輯循序或非循序。一般更新區塊只用來按照主機請求的順序記錄各種程式段。例如，即使主機系統資料或系統控制資料傾向於依混亂方式加以更新，仍不必以和主機使用者資料的不同方式來處理邏輯位址空間對應於主機系統資料的區域。

較佳將區段之完整邏輯群組的資料按邏輯上循序順序儲存在單一中繼區塊中。依此方式，可預定義已儲存之邏輯區段的索引。當中繼區塊按照預定順序儲存給定邏輯群組的所有區段時，其可說是「完整」。至於更新區塊，當其最後按邏輯上循序順序填滿更新資料時，則更新區塊將成為隨時可取代原始中繼區塊之已更新的完整中繼區塊。另一方面，如果更新區塊按邏輯上和完整區塊的不同順序填滿更新資料，更新區塊為非循序或混亂更新區塊，則必須進一步處理順序紊亂的程式段，以便最後能按和完整區塊相同的順序來儲存邏輯群組的更新資料。在較佳的例子中，其在單一中繼區塊中係按照邏輯上循序的順序。進一步的處理涉及將更新區塊中已更新的區段和原始區塊中未變更的區段彙總至又另一個更新中繼區塊。然後彙總的更新區塊將按照邏輯上循序的順序並能夠用來取代原始區塊。在一些預定條件下，彙總程序之前會有一或多個壓縮程序。壓縮程序只是將混亂更新區塊的區段重新記錄成取代的混亂更新區塊，同時除去任何已由相同邏輯區段的後續更新淘汰的複製邏輯區段。

更新方案允許同時執行多個多達預定極大值的更新執行

緒。各執行緒為使用其專用更新中繼區塊進行更新的邏輯群組。

循序資料更新

在先更新屬於邏輯群組的資料時，會配置中繼區塊及將其專用為邏輯群組之更新資料的更新區塊。當從主機接收寫入邏輯群組之一或多個區段之程式段的指令時(現有的中繼區塊已經儲存所有完整區段)，會配置更新區塊。對於第一主機寫入操作，會將第一程式段的資料記錄在更新區塊上。由於各主機寫入是具有連續邏輯位址之一或多個區段的一個程式段，因此會遵循第一更新在特性上永遠循序。在後續的主機寫入中，會按照從主機接收的順序將相同邏輯群組內的更新程式段記錄在更新區塊中。一個區塊繼續接受管理為循序更新區塊，而關聯邏輯群組內由主機更新的區段維持邏輯上循序。在此邏輯群組中更新的所有區段會被寫入此循序更新區塊，直到區塊關閉或轉換為混亂更新區塊。

圖7A顯示由於兩個分開的主機寫入操作而按循序順序寫入循序更新區塊之邏輯群組中之區段的範例，而邏輯群組之原始區塊中對應的區段變成淘汰。在主機寫入操作#1中，會更新邏輯區段LS5-LS8的資料。更新成為LS5'-LS8'的資料會被記錄在新配置的專用更新區塊中。

為了方便，會將邏輯群組中要更新的第一區段記錄在始於第一實體區段位置的專用更新區塊中。一般而言，要更新的第一邏輯區段不一定是群組的邏輯第一區段，因此，

在邏輯群組的起點及更新區塊的起點之間會有位移。此位移稱為「頁面標記」，如先前結合圖3A所述。後續區段將按照邏輯上循序的順序加以更新。在寫入邏輯群組的最後區段時，群組位址會繞回及寫入序列會從群組的第一區段繼續。

在主機寫入操作#2中，會更新邏輯區段LS9-LS12中資料的程式段。更新成為LS9'-LS12'的資料會被記錄在直接在最後寫入結束處之後之位置中的專用更新區塊。圖中顯示兩個主機寫入如下：按邏輯上循序順序記錄在更新區塊中的更新資料，即LS5'-LS12'。更新區塊可視為循序更新區塊，因其已按邏輯上循序的順序填入。記錄在更新區塊中的更新資料可淘汰原始區塊中對應的資料。

混亂資料更新

當關聯的邏輯群組內由主機更新的任何區段為邏輯上非循序時，可為現有的循序更新區塊啟始混亂更新區塊管理。混亂更新區塊是資料更新區塊的形式，其中關聯邏輯群組內的邏輯區段可按任何順序進行更新並可重複任意次。其建立可藉由以下方式：在主機寫入的區段是邏輯上非循序時，從循序更新區塊轉換成受更新之邏輯群組內先前寫入的區段。所有其後在此邏輯群組中更新的區段會被寫入混亂更新區塊中的下一個可用區段位置，無論其在群組內的邏輯區段位址為何。

圖7B顯示由於五個分開的主機寫入操作而按混亂順序寫入混亂更新區塊之邏輯群組之區段的範例，而邏輯群組之

原始區塊中被取代的區段及混亂更新區塊中被複製的區段變成淘汰。在主機寫入操作#1中，會更新儲存於原始中繼區塊之給定邏輯群組的邏輯區段LS10-LS11。已更新的邏輯區段LS10'-LS11'會被儲存到新配置的更新區塊中。此時，更新區塊為循序的更新區塊。在主機寫入操作#2中，會將邏輯區段LS5-LS6更新成為LS5'-LS6'及將其記錄在緊接上一個寫入之後之位置的更新區塊中。這可將循序的更新區塊轉換為混亂的更新區塊。在主機寫入操作#3，再次更新邏輯區段LS10及將其記錄在更新區塊的下一個位置成為LS10"。此時，更新區塊中的LS10"可取代先前記錄中的LS10'，而LS10'又可取代原始區塊中的LS10。在主機寫入操作#4中，再次更新邏輯區段LS10的資料及將其記錄在更新區塊的下一個位置中成為LS10"'。因此，LS10"'現在是邏輯區段LS10的最後且唯一有效的資料。在主機寫入操作#5中，會更新邏輯區段LS30的資料及將其記錄在更新區塊中成為LS30'。因此，此範例顯示可以按照任何順序及任意重複，將一邏輯群組內的複數個邏輯區段寫入至一混亂更新區塊中。

強制循序更新

圖8顯示由於兩個在邏輯位址有中斷之分開的主機寫入操作而在邏輯群組中按循序順序寫入循序更新區塊之區段的範例。在主機寫入#1中，將邏輯區段LS5-LS8的更新資料記錄在專用更新區塊中成為LS5'-LS8'。在主機寫入#2中，將邏輯區段LS14-LS16的更新資料記錄在上一個寫入之後

的更新區塊中成為LS14'-LS16'。然而，在LS8及LS14之間有位址跳躍，及主機寫入#2通常會使更新區塊成為非循序。由於位址跳躍不是很多，一個選項是在執行主機寫入#2之前將原始區塊之中間區段的資料複製到更新區塊，以先執行填補操作(#2A)。依此方式，即可保存更新區塊的循序特性。

圖9顯示根據本發明的一般具體實施例，更新區塊管理器更新一個邏輯群組之資料的程序流程圖。更新程序包含以下步驟：

步驟260：該記憶體被組織成複數個區塊，每個區塊均被分割成可一起抹除的複數個記憶體單元，每個記憶體單元係用於儲存一邏輯單元的資料。

步驟262：該資料被組織成複數個邏輯群組，每個邏輯群組均被分割成複數個邏輯單元。

步驟264：在標準的例子中，根據第一指定順序，較佳為邏輯上循序的順序，將邏輯群組的所有邏輯單元儲存在原始區塊的記憶體單元中。依此方式，即可得知存取區塊中個別邏輯單元的索引。

步驟270：對於給定邏輯群組(如， LG_x)的資料，請求更新 LG_x 內的邏輯單元。(邏輯單元更新係為舉例說明。一般而言，更新將是由 LG_x 內一或多個連續邏輯單元所組成的程式段。)

步驟272：請求的更新邏輯單元將會儲存在專用於記錄 LG_x 之更新的第二區塊中。記錄順序係根據第二順序，通常

是更新請求的順序。本發明的一個特色允許設定初始對按照邏輯上循序或混亂順序記錄的資料為一般的更新區塊。因此根據第二順序，第二區塊可以是循序更新區塊或混亂更新區塊。

步驟274：當程序迴圈返回到步驟270時，第二區塊繼續記錄所請求的邏輯單元。在關閉的預定條件成形時將會關閉第二區塊，以接收進一步更新。此時，程序繼續進行至步驟276。

步驟276：判斷該已關閉之第二區塊是否以和原始區塊相同的順序來記錄其更新邏輯單元。當該等兩個區塊記錄邏輯單元僅相差一頁面標記時，該等兩個區塊便會被視為具有相同的順序，如結合圖3A所述。如果這兩個區塊具有相同的順序，程序繼續進行至步驟280，否則，必須在步驟290執行廢棄項目收集。

步驟280：由於第二區塊具有和第一區塊相同的順序，因此其可用來取代原始的第一區塊。然後，更新程序結束於步驟299。

步驟290：從第二區塊(更新區塊)及第一區塊(原始區塊)收集給定邏輯群組之各邏輯單元的最新版本。然後按照和第一區塊相同的順序將給定邏輯群組之已彙總的邏輯單元寫入第三區塊。

步驟292：由於第三區塊(彙總的區塊)具有和第一區塊相同的順序，因此其可用來取代原始的第一區塊。然後，更新程序結束於步驟299。

步驟299：當結束程序建立完整的更新區塊時，該區塊會變成給定邏輯群組的新標準區塊。將會終止此邏輯群組的更新執行緒。

圖10顯示根據本發明的一項較佳具體實施例，更新區塊管理器更新一個邏輯群組之資料的程序流程圖。更新程序包含以下步驟：

步驟310：對於給定邏輯群組(如， LG_x)的資料，會請求更新 LG_x 內的邏輯區段。(區段更新係為舉例說明。一般而言，更新將是由 LG_x 內一或多個連續邏輯區段所組成的程式段。)

步驟312：如果 LG_x 專用的更新區塊尚未存在，繼續進行至步驟410以啟始邏輯群組之新的更新執行緒。這可藉由以下方式來完成：配置記錄邏輯群組之更新資料專用的更新區塊。如果已經有開啟的更新區塊，則繼續進行至步驟314，開始將更新區段記錄至更新區塊上。

步驟314：如果目前更新區塊已經混亂(即，非循序)，則直接繼續進行至步驟510，以將請求的更新區段記錄至混亂更新區塊上。如果目前更新區塊為循序，則繼續進行至步驟316，以處理循序更新區塊。

步驟316：本發明的一項特點係允許於初始時將一更新區塊設置成通用於以邏輯循序或混亂順序的方式來記錄資料。不過，因為該邏輯群組最終會將其資料以邏輯循序的順序儲存於一中繼區塊之中，所以，因此希望儘可能保持該更新區塊的循序狀態。接著，當關閉一更新區塊以進行

進一步更新時，將會需要較少的處理，因為並不需要進行廢棄項目收集。

因此，判斷所請求的更新是否遵循更新區塊的目前循序順序。如果更新循序遵循，則繼續進行至步驟510以執行循序更新，及更新區塊將維持循序。另一方面，如果更新未循序遵循(混亂更新)，則其會在未採取任何動作時將循序更新區塊轉換成混亂更新區塊。

在一項具體實施例中，不會進行任何動作來挽救此情況，然後程序直接進行至步驟370，其中允許更新將更新區塊變成混亂更新區塊。

選用性強制循序程序

在另一項具體實施例中，會選用性執行強制循序程序步驟320，以儘可能因顧及懸置的混亂更新而保存循序更新區塊。其中有兩個情況，這兩個情況都需要複製原始區塊的遺失區段，以維持更新區塊上記錄之邏輯區段的循序順序。第一個情況是其中更新可建立短的位址跳躍。第二個情況是提早結束更新區塊以將其保持循序。強制循序程序步驟320包含以下子步驟：

步驟330：如果更新建立的邏輯位址跳躍未大於預定的數量 C_B ，則程序繼續進行至步驟350的強制循序更新程序，否則程序繼續進行至步驟340，以考慮其是否適合進行強制循序結束。

步驟340：如果未填充的實體區段數量超過預定的設計參數 C_C (其代表值為更新區塊尺寸的一半)，則更新區塊為相

對未被使用，因此不會提早關閉。程序繼續進行至步驟370，及更新區塊會成為混亂。另一方面，如果實質上已填充更新區塊，則將其視為已經充分利用，因此進入步驟360以進行強制循序結束。

步驟350：只要位址跳躍未超過預定的數量 C_B ，強制循序更新允許目前循序更新區塊維持循序。實質上，會複製更新區塊之關聯原始區塊的區段，以填充位址跳躍跨越的間隙。因此，在繼續進行至步驟510之前，會以中間位址的資料填補循序更新區塊，以循序記錄目前的更新。

步驟360：如果目前循序更新區塊在實質上已經被填充，而非由懸置的混亂更新轉換為混亂更新區塊，則強制循序結束允許目前循序更新區塊成為關閉狀態。混亂或非循序更新的定義是具有以下項目的更新：不為上述位址跳躍例外所涵蓋的正向位址轉變、反向位址轉變、或位址重複。為了防止循序更新區塊為混亂更新所轉換，會藉由複製更新區塊之關聯之原始部分淘汰區塊的區段來填充更新區塊的未寫入區段位置。然後完全淘汰及抹除原始區塊。現在，目前的更新區塊具有完整組的邏輯區段，然後結束成為取代原始中繼區塊的完整中繼區塊。然後，程序繼續進行至步驟430以在其位置配置更新區塊，以接受在步驟310先請求之懸置區段更新的記錄。

轉換成混亂更新區塊

步驟370：當懸置的更新未按循序順序且可有可無時，如果無法滿足強制循序條件，則在程序繼續進行至步驟510

時，藉由允許在更新區塊上記錄具有非循序位址的懸置更新區段，以允許循序更新區塊轉換為混亂更新區塊。如果最大數量的混亂更新區塊存在，則在允許轉換進行之前，必須關閉最久未存取的混亂更新區塊；因而不會超過最大數量的混亂區塊。最久未存取之混亂更新區塊的識別和步驟420所述的一般例子相同，但僅限於混亂更新區塊。此時關閉混亂更新區塊可藉由如步驟550所述的彙總來達成。

配置受系統限制的新更新區塊

步驟410：將抹除中繼區塊配置為更新區塊的程序始於決定是否超過預定的系統限制。由於資源有限，記憶體管理系統通常允許可同時存在之更新區塊的預定最大數量 C_A 。此限制是循序更新區塊及混亂更新區塊的總數，也是設計參數。在一項較佳具體實施例中，此限制為例如最大8個更新區塊。還有，由於系統資源的較高需求，在可同時開啟之混亂更新區塊的最大數量上還有對應的預定限制(如，4)。

因此，在已經配置 C_A 個更新區塊時，則只有在關閉現有的配置請求之一後，才能滿足下一個配置請求。程序繼續進行至步驟420。當開啟的更新區塊數量小於 C_A 時，程序直接進行至步驟430。

步驟420：在超過更新區塊的最大數量 C_A 時，關閉最久未存取的更新區塊及執行廢棄項目收集。最久未存取的更新區塊會被識別為和最久未存取之邏輯區塊關聯的更新區塊。為了決定最久未存取的區塊，存取包括邏輯區段的寫入及選擇性讀取。會按照存取的順序維持開啟更新區塊的

清單；初始化時，不會假設任何存取順序。更新區塊的關閉在更新區塊為循序時，將按照結合步驟360及步驟530所述的相同程序，在更新區塊為混亂時，將按照結合步驟540所述的相同程序。此關閉可挪出空間以在步驟430配置新的更新區塊。

步驟430：配置一新的中繼區塊作為該給定邏輯群組LG_x專屬的更新區塊便可滿足該配置要求。接著，程序繼續進行至步驟510。

在更新區塊上記錄更新資料

步驟510：將請求的更新區段記錄在更新區塊的下一個可用實體位置上。然後，程序繼續進行至步驟520，決定更新區塊是否準備結束就緒。

更新區塊結束

步驟520：如果更新區塊還有接受附加更新的空間，則繼續進行至步驟522。否則繼續進行至步驟570以結束更新區塊。在目前請求的寫入嘗試寫入多於區塊所有空間的邏輯區段時，有兩項填滿更新區塊的可能實施例。在第一實施例中，會將寫入請求分成兩個部分，其中第一部分可寫入直到區塊的最後實體區段。然後關閉區塊及將寫入的第二部分處理為下一個請求的寫入。在另一個實施例中，會在區塊填補其餘區段時保留請求的寫入，然後再將其關閉。請求的寫入會被處理為下一個請求的寫入。

步驟522：如果更新區塊為循序，則繼續進行至步驟530以進行循序關閉。如果更新區塊為混亂，則繼續進行至步

驟540以進行混亂關閉。

循序更新區塊結束

步驟530：由於更新區塊為循序且已完全填充，因此其中儲存的邏輯群組為完整。中繼區塊為完整且能取代原始的中繼區塊。此時，會完全淘汰原始區塊及將其抹除。然後程序繼續進行至步驟570，其中給定邏輯群組的更新執行緒結束。

混亂更新區塊結束

步驟540：由於更新區塊為非循序填充，可能含有一些邏輯區段的多個更新，因此會執行廢棄項目收集以挽救其中的有效資料。混亂更新區塊可為已經壓縮或彙總。在步驟542將會決定執行的程序。

步驟542：要執行壓縮或彙總將根據更新區塊的退化而定。如果邏輯區段已更新多次，其邏輯位址將高度退化。記錄在更新區塊上的將有相同邏輯區段的多個版本，而只有最後記錄的版本為該邏輯區段的有效版本。在含有多個版本之邏輯區段的更新區塊中，有所區分之邏輯區段的數量會比邏輯群組的少很多。

在較佳具體實施例中，當更新區塊中有所區分之邏輯區段的數量超過預定的設計參數 C_D (其代表值為邏輯群組尺寸的一半)時，結束程序會在步驟550執行彙總，否則程序會繼續進行至步驟560的壓縮。

步驟550：如果要彙總混亂更新區塊，則將以含有彙總資料之新的標準中繼區塊取代原始區塊及更新區塊。彙總

後，更新執行緒將結束於步驟570。

步驟560：如果要壓縮混亂更新區塊，則將以載有壓縮資料之新的更新區塊取代之。壓縮後，已壓縮更新區塊的處理將結束於步驟570。或者，可將壓縮延遲直到再次寫入更新區塊，因此排除壓縮之後為沒有中間更新之彙總的可能性。然後，當步驟502中出現下一個要求於 LG_X 中進行更新時，在給定邏輯區塊的進一步更新中使用新的更新區塊。

步驟570：當結束程序建立完整的更新區塊時，該區塊會變成給定邏輯群組的新標準區塊。此邏輯群組的更新執行緒將會終止。在結束程序建立取代現有更新區塊之新的更新區塊時，會使用新的更新區塊記錄為給定邏輯群組請求的下一個更新。在更新區塊未結束時，處理會在步驟310中出現下一個要求於 LG_X 中進行更新時時繼續。

從上述程序可知，在關閉混亂更新區塊時，會進一步處理其上記錄的更新資料。尤其，其有效資料的廢棄項目收集係藉由以下程序：壓縮至另一個混亂區塊的程序，或和其關聯之原始區塊彙總以形成新的標準循序區塊的程序。

圖11A為詳細顯示關閉圖10所示之混亂更新區塊之彙總程序的流程圖。混亂更新區塊彙總是執行於結束更新區塊時(如當更新區塊已填滿其寫入的最後實體區段位置時)的兩個可能程序之一。區塊中寫入之有所區分的邏輯區段數量超過預定的設計參數 C_D 時，將選擇彙總。圖10所示的彙總程序步驟550包含以下子步驟：

步驟551：在混亂更新區塊被關閉時，會配置可取而代之

的新中繼區塊。

步驟552：在混亂更新區塊及關聯之原始區塊中，收集各邏輯區段的最新版本並忽略所有的淘汰區段。

步驟554：將所收集的有效區段按邏輯上循序順序記錄在新的中繼區塊上，以形成完整區塊，即，按循序順序記錄之邏輯群組之所有邏輯區段的區塊。

步驟556：以新的完整區塊取代原始區塊。

步驟558：抹除已結束的更新區塊及原始區塊。

圖11B為詳細顯示關閉圖10所示之混亂更新區塊之壓縮程序的流程圖。區塊中寫入之有所區分的邏輯區段數量低於預定的設計參數 C_D 時，將選擇壓縮。圖10所示的壓縮程序步驟560包含以下子步驟：

步驟561：在混亂更新區塊被壓縮時，會配置取而代之的新中繼區塊。

步驟562：在要壓縮之現有的混亂更新區塊中，收集各邏輯區段的最新版本。

步驟564：將所收集的區段記錄在新的更新區塊上，以形成具有壓縮區段之新的更新區塊。

步驟566：以具有壓縮區段之新的更新區塊取代現有的更新區塊。

步驟568：抹除已結束的更新區塊。

邏輯與中繼區塊狀態

圖12A顯示邏輯群組的所有可能狀態，及其間在各種操作下的可能轉變。

圖 12B 為列出邏輯群組之可能狀態的表格。邏輯群組狀態的定義如下：

1. 完整：邏輯群組中的所有邏輯區段已依照邏輯循序順序)被寫入單一中繼區塊之中，其可能利用頁面標記捲繞方式。

2. 未寫入：邏輯群組中未曾寫入任何邏輯區段。邏輯群組在群組位址表中會標示為未寫入且沒有任何配置的中繼區塊。返回預定的資料模式，以回應此群組內每個區段的主機讀取。

3. 循序更新：邏輯群組內的一些區段已經按邏輯上循序順序被寫入中繼區塊中，可能使用頁面標記，因此其可取代群組中任何先前完整狀態之對應的邏輯區段。

4. 混亂更新：邏輯群組內的一些區段已經按邏輯上非循序順序被寫入中繼區塊中，可能使用頁面標記，因此其可取代群組中任何先前完整狀態之對應的邏輯區段。群組內的區段可被寫入一次以上，其中最新版本將取代所有先前的版本。

圖 13A 顯示中繼區塊的所有可能狀態，及其間在各種操作下的可能轉變。

圖 13B 顯示中繼區塊的所有可能狀態，及其間在各種操作下的可能轉變：

1. 已抹除：中繼區塊中的所有區段已被抹除。

2. 循序更新：中繼區塊已經被部份寫入，其區段呈現邏輯循序順序，可能使用頁面標記。所有區段均屬於相同的

邏輯群組。

3. 混亂更新：該中繼區塊已經被部份或完全寫入，其區段呈現邏輯非循序順序。任何區段均可被寫入一次以上。所有區段均屬於相同的邏輯群組。

4. 完整：中繼區塊已經以邏輯循序的順序被完全寫入，可能使用頁面標記。

5. 原始：中繼區塊先前為完整但至少一區段已因主機資料更新而淘汰。

圖 14(A)-14(J)為顯示邏輯群組狀態上及實體中繼區塊上各種操作效果的狀態圖。

圖 14(A)顯示對應於第一寫入操作之邏輯群組及中繼區塊轉變的狀態圖。主機按邏輯上循序順序將先前未寫入之邏輯群組的一或多個區段寫入至新配置的已抹除中繼區塊。邏輯群組及中繼區塊進入循序更新狀態。

圖 14(B)顯示對應於第一完整操作之邏輯群組及中繼區塊轉變的狀態圖。先前未寫入的循序更新邏輯群組因主機循序寫入所有區段而變成完整。如果記憶卡以預定的資料模式填充其餘未寫入的區段來填滿群組，則也會發生轉變。中繼區塊變成完整。

圖 14(C)顯示對應於第一混亂操作之邏輯群組及中繼區塊轉變的狀態圖。先前未寫入的循序更新邏輯群組在主機非循序寫入至少一區段時變成混亂。

圖 14(D)顯示對應於第一壓縮操作之邏輯群組及中繼區塊轉變的狀態圖。從舊區塊將先前未寫入之混亂更新邏輯

群組內的所有有效區段複製到新的混亂中繼區塊，然後再將其抹除。

圖 14(E)顯示對應於第一彙總操作之邏輯群組及中繼區塊轉變的狀態圖。從舊的混亂區塊移動先前未寫入之混亂更新邏輯群組內的所有有效區段，以按邏輯上循序順序填充新配置的已抹除區塊。主機未寫入的區段會以預定的資料模式加以填充。然後抹除舊的混亂區塊。

圖 14(F)顯示對應於循序寫入操作之邏輯群組及中繼區塊轉變的狀態圖。主機按邏輯上循序順序將完整邏輯群組的一或多個區段寫入新配置的已抹除中繼區塊。邏輯群組及中繼區塊成為循序更新狀態。先前完整的中繼區塊變成原始中繼區塊。

圖 14(G)顯示對應於循序填充操作之邏輯群組及中繼區塊轉變的狀態圖。循序更新邏輯群組在主機循序寫入所有其區段時變成完整。這也可發生於以原始區塊的有效區段填充循序更新邏輯群組以使其完整的廢棄項目收集期間，在此之後會抹除原始區塊。

圖 14(H)顯示對應於非循序寫入操作之邏輯群組及中繼區塊轉變的狀態圖。循序更新邏輯群組在主機非循序寫入至少一區段時變成混亂。非循序區段寫入可造成更新區塊或對應原始區塊中的有效區段變成淘汰。

圖 14(I)顯示對應於壓縮操作之邏輯群組及中繼區塊轉變的狀態圖。從舊區塊將混亂更新邏輯群組內的所有有效區段複製到新的混亂中繼區塊，然後再將其抹除。原始區塊

不會受到影響。

圖 14(J)顯示對應於彙總操作之邏輯群組及中繼區塊轉變的狀態圖。從舊的混亂區塊複製混亂更新邏輯群組內的所有有效區段，以按邏輯上循序順序填充新配置的已抹除區塊。然後抹除舊的混亂區塊及原始區塊。

更新區塊追蹤及管理

圖 15顯示用於追蹤已開啟及已關閉之更新區塊與已抹除區塊以進行配置的配置區塊清單(ABL)的結構的較佳具體實施例。該配置區塊清單(ABL)610會被保存於控制器RAM 130之中，以允許管理已抹除區塊之配置、配置的更新區塊、關聯的區塊及控制結構，以啟用正確的邏輯對實體位址轉譯。在較佳具體實施例中，ABL包括：已抹除區塊的清單、開啟的更新區塊清單614、及關閉的更新區塊清單616。

開啟的更新區塊清單614是ABL中具有開啟更新區塊之屬性的區塊項目組。開啟的更新區塊清單具有目前開啟之各資料更新區塊的一個項目。各項目保有以下資訊。LG是目前更新中繼區塊專用的邏輯群組位址。循序/混亂是代表以循序或混亂更新資料填充更新區塊的狀態。MB是更新區塊的中繼區塊位址。頁面標記是在更新區塊之第一實體位置記錄的起始邏輯區段。寫入的區段號碼代表目前在更新區塊上寫入的區段號碼。MB₀是關聯之原始區塊的中繼區塊位址。Page Tag₀是關聯之原始區塊的頁面標記。

關閉的更新區塊清單616是配置區塊清單(ABL)的子集。

其為ABL中具有關閉更新區塊之屬性的區塊項目組。關閉的更新區塊清單具有一個已關閉之各資料更新區塊的項目，但其項目在邏輯對主實體目錄中並未更新。各項目保有以下資訊。LG是目前更新區塊專用的邏輯群組位址。MB是更新區塊的中繼區塊位址。頁面標記是在更新區塊之第一實體位置記錄的起始邏輯區段。MB₀是關聯之原始區塊的中繼區塊位址。

混亂區塊索引

循序更新區塊具有按邏輯上循序順序儲存的資料，因此很容易尋找區塊中的任何邏輯區段。混亂更新區塊具有其未按順序儲存的邏輯區段並可儲存一個邏輯區段的多個更新世代。必須維持附加資訊，以記錄各有效邏輯區段被配置在混亂更新區塊中的位置。

在較佳具體實施例中，混亂區塊索引資料結構允許追蹤及快速存取混亂區塊中的所有有效區段。混亂區塊索引獨立管理小型區域的邏輯位址空間，並可有效處理系統資料及使用者資料的熱區。索引資料結構實質上允許在快閃記憶體中維持具有不常更新需求的索引資訊，以免效能明顯受到影響。另一方面，會將混亂區塊中最近寫入區段的清單保留在控制器RAM的混亂區段清單中。還有，會將快閃記憶體之索引資訊的快取記憶體保留在控制器RAM中，以減少位址轉譯之快閃區段存取的數量。各混亂區塊的索引係儲存於快閃記憶體的混亂區塊索引(CBI)區段中。

圖16A顯示混亂區塊索引(CBI)區段的資料欄位。混亂區

塊索引區段(CBI區段)含有邏輯群組中各區段映射至混亂更新區塊的索引，可定義邏輯群組各區段在混亂更新區塊或其關聯之原始區塊內的位置。CBI區段包括：記錄混亂區塊內有效區段的混亂區塊索引欄位、記錄混亂區塊位址參數的混亂區塊資訊欄位、及記錄儲存CBI區段之中繼區塊(CBI區塊)內有效CBI區段的區段索引欄位。

圖 16B顯示記錄於專用中繼區塊中之混亂區塊索引(CBI)區段的範例。專用的中繼區塊可稱為CBI區塊620。在更新CBI區段時，會將其寫入CBI區塊620中下一個可用的實體區段位置。因此，CBI區段的多個複本可存在CBI區塊中，其中只有最後寫入的複本為有效。例如，已經利用有效版本的最新版本將邏輯群組LG₁的CBI區段更新三次。區塊中最後寫入之CBI區段的一組索引可識別CBI區塊中各有效區段的位置。在此範例中，區塊中最後寫入的CBI區段是LG₁₃₆的CBI區段，及其索引組是取代所有先前索引組的有效索引組。當CBI區塊最後變成以CBI區段予以完全填充時，會將所有有效區段再寫入新的區塊位置，以在控制寫入操作期間壓縮區塊。然後抹除整個區塊。

CBI區段內的混亂區塊索引欄位含有邏輯群組內各邏輯區段的索引項目或映射至混亂更新區塊的子群組。各索引項目代表對應邏輯區段之有效資料所在之混亂更新區塊內的位移。保留的索引值代表混亂更新區塊中沒有邏輯區段的有效資料，及代表關聯之原始區塊中的對應區段為有效。一些混亂區塊索引欄位項目的快取記憶體會被保留在

控制器 RAM 中。

CBI區段內的混亂區塊資訊欄位含有關於存在於系統中各混亂更新區塊的一個項目，以記錄區塊的位址參數資訊。此欄位中的資訊只在CBI區塊的最後寫入區段中有效。此資訊也會出現在RAM的資料結構中。

各混亂更新區塊的項目包括三個位址參數。第一個參數是和混亂更新區塊關聯之邏輯群組(或邏輯群組數)的邏輯位址。第二個參數是混亂更新區塊的中繼區塊位址。第三個參數是寫入混亂更新區塊中最後區段的實體位址位移。位移資訊設定初始化期間混亂更新區塊的掃描起點，以在RAM中重建資料結構。

區段索引欄位含有關於CBI區塊中各有效CBI區段的項目。其可定義CBI區塊內有關各許可之混亂更新區塊之最近寫入的CBI區段所在的位移。索引中位移的保留值代表許可的混亂更新區塊並不存在。

圖 16C 為顯示存取正在進行混亂更新之給定邏輯群組之邏輯區段之資料的流程圖。在更新程序期間，會將更新資料記錄在混亂更新區塊中，而未變更的資料則留在和邏輯群組關聯的原始中繼區塊中。在混亂更新下存取邏輯群組之邏輯區段的程序如下：

步驟 650：開始尋找給定邏輯群組的給定邏輯區段。

步驟 652：在CBI區塊中尋找最後寫入的CBI區段。

步驟 654：藉由查詢最後寫入之CBI區段的混亂區塊資訊欄位，尋找和給定邏輯群組關聯的混亂更新區塊或原始區

塊。此步驟可在步驟662前的任何時間執行。

步驟658：如果最後寫入的CBI區段係指向給定的邏輯群組，則可尋找CBI區段。繼續進行至步驟662。否則，繼續進行至步驟660。

步驟660：藉由查詢最後寫入之CBI區段的區段索引欄位，尋找給定邏輯群組的CBI區段。

步驟662：藉由查詢所尋找到之CBI區段的混亂區塊索引欄位，尋找混亂區塊或原始區塊中的給定邏輯區段。

圖16D顯示根據其中已將邏輯群組分割成子群組的替代性具體實施例，存取正在進行混亂更新之給定邏輯群組之邏輯區段之資料的流程圖。CBI區段的有限容量只能記錄預定最大數量的邏輯區段。當邏輯群組具有多於單一CBI區段所能處理的邏輯區段時，會將邏輯群組分割成多個具有指派給各子群組之CBI區段的子群組。在一個範例中，各CBI區段具有足夠追蹤由256區段所組成及多達8個混亂更新區塊之邏輯群組的容量。如果邏輯群組具有超過256區段的尺寸，則存在用於邏輯群組內各256區段子群組之分開的CBI區段。CBI區段可存在用於邏輯群組內多達8個子群組，以支援尺寸多達2048區段的邏輯群組。

在較佳具體實施例中，會採用間接索引方案以促進索引的管理。區段索引的各項目具有直接及間接欄位。

直接區段索引可定義CBI區塊內有關特定混亂更新區塊之所有可能CBI區段所在的位移。此欄位中的資訊只在有關該特定混亂更新區塊之最後寫入的CBI區段中有效。索引中

位移的保留值代表CBI區段並不存在，因為對應之有關混亂更新區塊的邏輯子群組或是不存在，或是由於已配置更新區塊而未更新。

間接區段索引可定義有關各許可的混亂更新區塊之最近寫入之CBI區段所在之CBI區塊內的位移。索引中位移的保留值代表許可的混亂更新區塊並不存在。

圖16D顯示在混亂更新下存取邏輯群組之邏輯區段的程序，其步驟如下：

步驟670：將各邏輯群組分割成多個子群組及指派CBI區段給各子群組。

步驟680：開始尋找給定邏輯群組之給定子群組的給定邏輯區段。

步驟682：在CBI區塊中尋找最後寫入的CBI區段。

步驟684：藉由查詢最後寫入之CBI區段的混亂區塊資訊欄位，尋找和給定子群組關聯的混亂更新區塊或原始區塊。此步驟可在步驟696前的任何時間執行。

步驟686：如果最後寫入的CBI區段係指向給定的邏輯群組，則繼續進行至步驟691。否則，繼續進行至步驟690。

步驟690：藉由查詢最後寫入之CBI區段的間接區段索引欄位，尋找給定邏輯群組之多個CBI區段中最後寫入的CBI區段。

步驟691：已經尋找到和給定邏輯群組之子群組其中之一關聯的至少一CBI區段。繼續。

步驟692：如果所尋找到的CBI區段指向給定子群組，則

可尋找給定子群組的CBI區段。繼續進行至步驟696。否則，繼續進行至步驟694。

步驟694：藉由查詢目前尋找之CBI區段的直接區段索引欄位，尋找給定子群組的CBI區段。

步驟696：藉由查詢給定子群組之CBI區段的混亂區塊索引欄位，尋找混亂區塊或原始區塊中的給定邏輯區段。

圖16E顯示在其中將各邏輯群組分割成多個子群組的具體實施例中，混亂區塊索引(CBI)區段及其功能的範例。邏輯群組700原來將其完整的資料儲存在原始中繼區塊702中。接著，邏輯群組配合配置專用的混亂更新區塊704進行更新。在本範例中，將邏輯群組700分割成子群組，這些子群組A、B、C、D各具有256個區段。

為了尋找子群組B中的第i個區段，會先尋找CBI區塊620中最後寫入的CBI區段。最後寫入之CBI區段的混亂區塊資訊欄位可提供尋找給定邏輯群組之混亂更新區塊704的地址。同時，其還可提供寫入混亂區塊中之最後區段的位置。此資訊在掃描及重建索引時很有用。

如果最後寫入之CBI區段結果是給定邏輯群組的四個CBI區段之一，則會進一步決定其是否正是含有第i個邏輯區段之給定子群組B的CBI區段。如果是，則CBI區段的混亂區塊索引會指向儲存第i個邏輯區段之資料的中繼區塊位置。區段位置會在混亂更新區塊704中或在原始區塊702中。

如果最後寫入之CBI區段結果是給定邏輯群組之四個

CBI區段之一但卻非屬於子群組B，則會查詢其直接區段索引，以尋找子群組B的CBI區段。在尋找此確切的CBI區段後，會查詢混亂區塊索引，以在混亂更新區塊704及原始區塊702中尋找第i個邏輯區段。

如果最後寫入之CBI區段結果不是給定邏輯群組之四個CBI區段的任一個，則會查詢間接區段索引以尋找四個區段中的一個。在圖16E所示的範例中，會尋找子群組C的CBI區段。然後，子群組C的此CBI區段查詢其直接區段索引，以尋找子群組B之確切的CBI區段。此範例顯示在查詢混亂區塊索引時，將發現第i個邏輯區段為未變更及會在原始區塊中尋找到其有效資料。

在給定邏輯群組的子群組C中尋找第j個邏輯區段時也會做出同樣的考慮。此範例顯示最後寫入之CBI區段結果不是給定邏輯群組之四個CBI區段的任何一個。其間接區段索引指向給定群組之四個CBI區段之一。所指向之四個中的最後寫入結果也正是子群組C的CBI區段。在查詢其混亂區塊索引時，將發現第j個邏輯區段被尋找在混亂更新區塊704中的指定位置。

控制器RAM中存在系統中之各混亂更新區塊的混亂區段清單。每份清單均含有一份自快閃記憶體中最後被更新的相關CBI區段開始至目前區段為止被寫入該混亂更新區塊之中的區段的記錄。特定混亂更新區塊之邏輯區段位址(可保留在混亂區段清單中)的數量是8至16之代表值的設計參數。清單的最佳尺寸可決定為其對混亂資料寫入操作之過

度耗用的作用及初始化期間之區段掃描時間之間的權衡。

在系統初始化期間，為了識別自其關聯之CBI區段之一的先前更新後所寫入的有效區段，必須掃描各混亂更新區塊。在控制器RAM中，會構成各混亂更新區塊的混亂區段清單。只需要在最後寫入的CBI區段中，從各區塊之混亂區塊資訊欄位中定義的最後區段位址開始掃描各區塊即可。

在配置混亂更新區塊時，會寫入CBI區段以對應於所有的更新邏輯子群組。混亂更新區塊的邏輯及實體位址會被寫入區段中可用的混亂區塊資訊欄位，其中空值項目在混亂區塊索引欄位中。會在控制器RAM中開啟混亂區段清單。

在關閉混亂更新區塊時，會以從區段中混亂區塊資訊欄位移除的區塊邏輯及實體位址寫入CBI區段。RAM中對應的混亂區段清單變成未使用。

可修改控制器RAM對應的混亂區段清單，以包括寫入混亂更新區塊之區段的記錄。當控制器RAM中的混亂區段清單沒有寫入混亂更新區塊之其他區段記錄的任何可用空間時，會為有關清單中區段的邏輯子群組寫入已更新的CBI區段，然後清除清單。

當CBI區塊620變滿時，會將有效的CBI區段複製到已配置的已抹除區塊中，然後抹除先前的CBI區塊。

位址表

圖2所示的邏輯對實體位址轉譯模組140負責關聯快閃記憶體中主機的邏輯位址和對應的實體位址。邏輯群組及實體群組(中繼區塊)間的映射係儲存於非揮發性快閃記憶體

200及揮發性卻較為敏捷之RAM 130(見圖1)中分布的一組表格及清單中。位址表係維持在含有記憶體系統中每個邏輯群組之中繼區塊位址的快閃記憶體中。此外，最近寫入區段的邏輯對實體位址記錄會暫時保留在RAM中。在系統啟動後進行初始化時，可從快閃記憶體中的區塊清單及資料區段標頭中重新構成這些揮發性記錄。因此，快閃記憶體中的位址表只需要偶而更新，以降低控制資料之過度耗用寫入操作的百分比。

邏輯群組之位址記錄的階層包括：在RAM中之開啟的更新區塊清單、關閉的更新區塊清單及維持在快閃記憶體中的群組位址表(GAT)。

開啟的更新區塊清單是控制器RAM中目前開啟用於寫入已更新之主機區段資料之資料更新區塊的清單。區塊的項目在區塊關閉時會被移至關閉的更新區塊清單。關閉的更新區塊清單是控制器RAM中已經關閉之資料更新區塊的清單。清單中項目的子集在控制寫入操作期間會被移至群組位址表中的區段。

群組位址表(GAT)是記憶體系統中主機資料所有邏輯群組之中繼區塊位址的清單。GAT含有根據邏輯位址循序排序之各邏輯群組的一個項目。GAT中的第n個項目含有具位址n之邏輯群組的中繼區塊位址。在較佳具體實施例中，GAT是快閃記憶體中的表格，其中包含一組具定義記憶體系統中每個邏輯群組之中繼區塊位址之項目的區段(稱為GAT區段)。在快閃記憶體中，會將GAT區段尋找在一或多

個專用的控制區塊(稱為GAT區塊)中。

圖 17A 顯示群組位址表(GAT)區段的資料欄位。GAT區段可如具有足夠含有一組128個連續邏輯群組之GAT項目的容量。各GAT區段包括兩個成分，即：一組用於範圍內各邏輯群組之中繼區塊位址的GAT項目，及GAT區段索引。第一成分含有用於尋找和邏輯位址關聯之中繼區塊的資訊。第二成分含有用於尋找GAT區塊內所有有效GAT區段的資訊。各GAT項目有三個欄位，即：中繼區塊號碼、如先前結合圖 3A(iii)所述的頁面標記、及代表中繼區塊是否已經重新連結的旗標。GAT區段索引列出GAT區塊中有效GAT區段的部分。此索引會在每個GAT區段中但會被GAT區塊中下一個寫入之GAT區段的版本所取代。因此只有最後寫入之GAT區段中的版本為有效。

圖 17B 顯示記錄在一或多個GAT區塊中之群組位址表(GAT)區段的範例。GAT區塊是記錄GAT區段專用的中繼區塊。在更新GAT區段時，會將其寫入GAT區塊720中下一個可用的實體區段位置。因此，GAT區段的多個複本可存在GAT區塊中，其中只有最後寫入的複本為有效。例如，GAT區段255(含有邏輯群組LG₃₉₆₈ - LG₄₀₉₈的指標)至少已經使用有效版本的最新版本更新兩次。區塊中最後寫入之GAT區段的一組索引可識別GAT區塊中各有效區段的位置。在此範例中，區塊中最後寫入的GAT區段是GAT區段236，及其索引組是取代所有先前索引組的有效索引組。當GAT區塊最後變成以GAT區段完全填充時，會將所有有效區段再

寫入新的區塊位置，以在控制寫入操作期間壓縮區塊。然後抹除整個區塊。

如上述，GAT區塊在邏輯位址空間的區域中含有邏輯上連續組之群組的項目。GAT區塊內的GAT區段各含有128個連續邏輯群組的邏輯對實體映射資訊。在GAT區塊所跨越的位址範圍內，儲存所有邏輯群組項目所需的GAT區段數僅佔用區塊內總區段位置的一小部分。因此，在區塊中下一個可用區段位置將GAT區段寫入，即可將其更新。GAT區塊中所有有效GAT區段及其位置的索引係維持在最近寫入之GAT區段中的索引欄位中。GAT區塊中有效GAT區段所佔用之總區段的一小部分係為系統設計參數，其通常為25%。然而，每個GAT區塊中最多有64個有效GAT區段。在大邏輯容量的系統中，可能必須在一個以上的GAT區塊中儲存GAT區段。此時，各GAT區塊係和固定範圍的邏輯群組關聯。

可將GAT更新執行作為控制寫入操作的部分，此操作會在ABL用盡配置的區塊時受到觸發(見圖18)。其執行和ABL填充及CBL清空操作同時進行。在GAT更新操作期間，一個GAT區段具有以關閉之更新區塊清單中對應項目所更新的項目。GAT項目被更新時，會從關閉的更新區塊清單(CUBL)移除任何對應的項目。例如，會根據關閉的更新區塊清單中的第一項目選擇要更新的GAT區段。可將更新區段寫入GAT區塊中的下一個可用區段位置。

當沒有任何區段位置可供一已更新的GAT區段使用時，

於控制寫入操作期間便會發生GAT再寫入操作。將會配置新的GAT區塊及從完整的GAT區塊按循序順序複製GAT索引所定義的有效GAT區段。然後抹除整個GAT區塊。

GAT快取記憶體是控制器RAM 130中，GAT區段之128個項目之子分割之項目的複本。GAT快取記憶體項目的數量是一項系統設計參數，代表值為32。每次從GAT區段讀取一個項目時，即可建立相關區段子分割的GAT快取記憶體。將會維持多個GAT快取記憶體。其數量是代表值為4的設計參數。GAT快取記憶體會根據最久未使用以不同區段子分割的項目來覆寫。

抹除的中繼區塊管理

圖2所示的抹除區塊管理器160可使用一組維持目錄及系統控制資訊的清單來管理抹除區塊。這些清單係分布於控制器RAM 130及快閃記憶體200中。當必須配置已抹除的中繼區塊以儲存使用者資料或儲存系統控制資料結構時，會選擇保留在控制器RAM中的配置區塊清單(ABL)中的下一個可用中繼區塊號碼(見圖15)。同樣地，在撤出中繼區塊後而將其抹除時，會將其號碼新增至同樣保留在控制器RAM中的清除區塊清單(CBL)。相對較靜態的目錄及系統控制資料係儲存於快閃記憶體中。這些包括列出快閃記憶體中所有中繼區塊之抹除狀態的已抹除區塊清單及位元對映(MAP)。已抹除的區塊清單及MAP係儲存於個別區段中，且會記錄在稱為「MAP區塊」的專用中繼區塊中。這些分布於控制器RAM及快閃記憶體中的清單可提供已抹除區塊

記錄的層級以有效管理已抹除中繼區塊的使用。

圖18為顯示使用及再循環已抹除區塊之控制及目錄資訊之分布及流程的示意方塊圖。控制及目錄資料係維持在被保留在常駐於快閃記憶體200之控制器RAM 130或在MAP區塊750中的清單。

在較佳具體實施例中，控制器RAM 130會保留配置區塊清單(ABL)610及清除區塊清單(CBL)740。如先前結合圖15所述，配置區塊清單(ABL)可記錄最近已配置哪個中繼區塊以儲存使用者資料或儲存系統控制資料結構。在必須配置新的已抹除中繼區塊時，會在配置區塊清單(ABL)中選擇下一個可用的中繼區塊號碼。同樣地，會使用清除區塊清單(CBL)記錄已經解除配置及抹除的更新中繼區塊。在控制器RAM 130(見圖1)中會保留ABL及CBL以在追蹤相對作用中更新區塊時進行快速存取及簡易操控。

配置區塊清單(ABL)可記錄即將成為更新區塊之已抹除中繼區塊的集區及已抹除中繼區塊的配置。因此，各個這些中繼區塊可由指定其是否為ABL懸置配置中的已抹除區塊、開啟的更新區塊、或關閉的更新區塊之屬性來說明。圖18顯示ABL含有：已抹除ABL清單612、開啟的更新區塊清單614、及關閉的更新區塊清單616。此外，和開啟的更新區塊清單614關聯的是關聯的原始區塊清單615。同樣地，和關閉的更新區塊清單關聯的是關聯的已抹除原始區塊清單617。如先前圖15所示，這些關聯的清單分別是開啟的更新區塊清單614及關閉的更新區塊清單616的子集。已

抹除的ABL區塊清單612、開啟的更新區塊清單614、及關閉的更新區塊清單616均為配置區塊清單(ABL)610的子集，各清單中的項目分別具有對應的屬性。

MAP區塊750是儲存快閃記憶體200中之抹除管理記錄專用的中繼區塊。MAP區塊儲存MAP區塊區段的時間序列，其中各MAP區段不是抹除區塊管理(EBM)區段760，就是MAP區段780。當已抹除區塊在配置用盡且在撤出中繼區塊時再循環時，關聯的控制及目錄資料較佳含在可在MAP區塊更新的邏輯區段中，其中會將更新資料的各例項記錄在新的區塊區段中。EBM區段760及MAP區段780的多個複本可存在MAP區塊750中，其中只有最新版本為有效。有效MAP區段之位置的索引係含在EMB區塊的欄位中。有效的EMB區段總是在控制寫入操作期間最後被寫入MAP區塊中。當MAP區塊750已滿時，會在控制寫入操作期間將所有有效區段再寫入新的區塊位置而將其壓縮。然後抹除整個區塊。

各EBM區段760含有已抹除的區塊清單(EBL)770，此清單是已抹除區塊總體之子集位址的清單。已抹除的區塊清單(EBL)770可當作含有已抹除之中繼區塊號碼的緩衝器，從此緩衝器中會定期取用中繼區塊號碼以重新填充ABL，並定期將中繼區塊號碼新增至此緩衝器中以重新清空CBL。EBL 770可當作用於以下項目的緩衝器：可用的區塊緩衝器(ABB)772、已抹除的區塊緩衝器(EBB)774及已清除的區塊緩衝器(CBB)776。

可用的區塊緩衝器(ABB)772含有緊接先前ABL填充操作之後之ABL 610之項目的複本。其實際上是正在ABL填充操作之後之ABL的備份複本。

已抹除的區塊緩衝器(EBB)774含有先前從MAP區段780或CBB清單776傳送之已抹除的區塊位址(說明如下)，且該等位址可用在ABL填充操作期間傳送至ABL 610。

已清除的區塊緩衝器(CBB)776含有在CBL清空操作期間已從CBL 740傳送及其後會被傳送至MAP區段780及EBB清單774之已抹除區塊的位址。

各個MAP區段780含有稱為「MAP」的位元對映結構。MAP會使用快閃記憶體中各中繼區塊的一位元，以用來表示各區塊的抹除狀態。對應於EBM區段中ABL、CBL或已抹除的區塊清單所列之區塊位址的位元在MAP中不會被設為抹除狀態。

區塊配置演算法永遠不會使用在MAP、已抹除的區塊清單、ABL或CBL內，任何未含有有效資料結構及未被指定為已抹除區塊的區塊，因此無法存取此類區塊用於儲存主機或控制資料結構。這可提供從可存取的快閃記憶體位址空間排除具有缺陷位置之區塊的簡單機制。

圖18所示之階層可有效地管理已抹除區塊記錄，並且對被儲存於該控制器之RAM中的該等區塊位址清單提供完整的安全性。可以不頻繁的方式在該些區塊位址清單及一個以上的MAP區段780之間交換已抹除的區塊項目。可於電源關閉之後的系統初始化期間，透過被儲存於快閃記憶體中

複數個區段中該等已抹除區塊清單及位址變換表中的資訊，以及有限地掃描快閃記憶體中少量被參照的資料區塊，來重建該些清單。

用於更新已抹除中繼區塊記錄之階層所採用的該等演算法可以下面的順序來配置使用已抹除區塊：將來自該MAP區塊750的區塊叢於位址順序中交錯來自該CBL 740的區塊位址叢，其反映的係區塊被該主機更新的順序。對大部份的中繼區塊大小與系統記憶體容量而言，單一MAP區段可針對該系統中的所有中繼區塊提供一位元對映。於此情況中，已抹除的區塊必定會以和被記錄於此MAP區段中相同的位址順序來配置使用。

抹除區塊管理操作

如上述，ABL 610是具有以下位址項目的清單：可經配置使用的已抹除中繼區塊，及最近已配置為資料更新區塊之中繼區塊。ABL中區塊位址的實際數量介於為系統設計變數之最大及最小限制之間。在製造期間，格式化之ABL項目的數量是記憶卡類型及容量的函數。此外，由於可用之已抹除區塊的數量會因壽命期間的區塊故障而縮減，也會縮減ABL中項目的數量接近系統壽命終點。例如，在填充操作後，ABL中的項目可指定可用於以下用途的區塊。每個區塊具有一個項目之部分寫入資料更新區塊的項目不超過系統對同時開啟之最大更新區塊的限制。用於配置為資料更新區塊之抹除區塊的一至二十個項目之間。配置為控制區塊之已抹除區塊的四個項目。

ABL填充操作

由於ABL 610會因為配置而變成耗盡，因此需要進行重新填充。填充ABL的操作發生於控制寫入操作期間。此係觸發於以下情況時：必須配置區塊，但ABL含有不足用於配置為資料更新區塊或一些其他控制資料更新區塊的已抹除區塊項目。在控制寫入期間，ABL填充操作係和GAT更新操作同時進行。

在ABL填充操作期間會發生以下動作。

1. 保留具有目前資料更新區塊之屬性的ABL項目。
2. 保留已關閉資料更新區塊之屬性的ABL項目，除非該區塊的某個項目正於該同時進行的GAT更新作業中被寫入，於此情況中則會從該ABL中移除該項目。
3. 保留用於未配置之抹除區塊的ABL項目。
4. 壓縮ABL以移除因移除項目所產生的間隙，以維持項目的順序。
5. 藉由附加來自該EBB清單中下次可用的項目，以完全填充該ABL。
6. 利用該ABL中該等目前的項目來覆寫該ABB清單。

CBL清空操作

CBL是控制器RAM中已抹除區塊位址的清單，對已抹除區塊項目數量的限制和ABL相同。清空CBL的操作發生於控制寫入操作期間。因此，其和ABL填充/GAT更新操作或CBI區塊寫入操作同時進行。在CBL清空操作中，會從CBL 740移除項目並將其寫入CBB清單776。

MAP交換操作

當EBB清單774已為清空時，在MAP區段780之抹除區塊資訊及EBM區段760間的MAP交換操作可定期發生於控制寫入操作期間。如果系統中的所有已抹除中繼區塊均記錄在EBM區段760中，將無任何MAP區段780存在及不會執行任何MAP交換。在MAP交換操作期間，用於將已抹除區塊饋送給EBB 774的MAP區段被視為來源MAP區段782。相反地，用於從CBB 776接收已抹除區塊的MAP區段被視為目的地MAP區段784。如果只有一個MAP區段，則可當作來源及目的地MAP區段，其定義如下。

在MAP交換期間會執行以下動作。

1. 以遞增指標的方式為基礎，選擇一來源MAP區段。
2. 以不在該來源MAP區段中之第一CBB項目中的區塊位址為基礎來選擇一目的MAP區段。
3. 如該CBB中相關項目所定義的方式來更新該目的MAP區段，並且從該CBB中移除該等項目。
4. 將該已更新的目的MAP區段寫入至該MAP區塊之中，除非沒有分離的來源MAP區段存在。
5. 如該CBB中相關項目所定義的方式來更新該來源MAP區段，並且從該CBB中移除該等項目。
6. 將該CBB中剩餘的項目附加至該EBB之中。
7. 利用該來源MAP區段所定義之已抹除區段位址盡可能地填充該EBB。
8. 將該已更新的來源MAP區段寫入至該MAP區塊中。

9. 將一已更新的EBM區段寫入至該MAP區塊中。

清單管理

圖18顯示各種清單間控制及目錄資訊的分布與流程。為了方便，在清單元件間移動項目或變更項目屬性的操作，在圖18中識別為[A]至[O]，說明如下。

[A] 在將抹除區塊配置為主機資料的更新區塊時，會將其在ABL中的項目屬性從已抹除的ABL區塊變更為開啟的更新區塊。

[B] 在將已抹除的區塊配置為控制區塊時，會移除其在ABL中的項目。

[C] 在建立一具有開啟更新區塊屬性的ABL項目時，會將關聯的原始區塊欄位新增至項目，以記錄被更新之邏輯群組的原始中繼區塊位址。從GAT可獲得此資訊。

[D] 關閉更新區塊時，其在ABL中的項目屬性會從開啟的更新區塊變更為關閉的更新區塊。

[E] 關閉更新區塊時，會抹除其關聯的原始區塊，及會將其在ABL中項目之關聯原始區塊欄位的屬性變更為已抹除的原始區塊。

[F] 在ABL填充操作期間，任何其位址在相同控制寫入操作期間於GAT中更新的已關閉更新區塊會從ABL中移除其項目。

[G] 在ABL填充操作期間，在從ABL移除已關閉更新區塊的項目時，會將其關聯之已抹除原始區塊的項目移至CBL。

[H] 在抹除控制區塊時，會將其所用項目新增至CBL。

[I] 在ABL填充操作期間，會從EBB清單將已抹除區塊項目移至ABL，且被賦以已抹除之ABL區塊的屬性。

[J] 在ABL填充操作期間修改所有相關的ABL項目後，ABL中的區塊位址將取代ABB清單的區塊位址。

[K] 和控制寫入期間的ABL填充操作同時進行，將CBL中已抹除區塊的項目移至CBB清單。

[L] 在MAP交換操作期間，從CBB清單將所有相關項目移至MAP目的地區段。

[M] 在MAP交換操作期間，從CBB清單將所有相關項目移至MAP來源區段。

[N] 在MAP交換操作期間的[L]與[M]之後，從CBB清單將所有其餘項目移至EBB清單。

[O] 在MAP交換操作期間的[N]之後，如果可能，從MAP來源區段移動除了在[M]中移動之項目以外的項目，以填充EBB清單。

邏輯對實體位址轉譯

為了在快閃記憶體中尋找邏輯區段的實體位置，圖2所示的邏輯對實體位址轉譯模組140可執行邏輯對實體位址轉譯。除了最近已更新的邏輯群組外，可以使用常駐在控制器RAM 130中快閃記憶體200或GAT快取記憶體的群組位址表(GAT)執行大多數的轉譯。最近已更新之邏輯群組的位址轉譯會需要查詢主要常駐在控制器RAM 130中之更新區塊的位址清單。因此，邏輯區段位址之邏輯對實體位址轉譯

的程序端視和區段所在之邏輯群組關聯之區塊的類型而定。區塊的類型如下：完整區塊、循序資料更新區塊、混亂資料更新區塊、關閉的資料更新區塊。

圖 19 為顯示邏輯對實體位址轉譯程序的流程圖。實質上，先使用邏輯區段位址查詢各種更新目錄(例如，開啟的更新區塊清單及關閉的更新區塊清單)，即可尋找對應的中繼區塊及實體區段。如果關聯的中繼區塊並不屬於更新程序的部分，則由 GAT 提供目錄資訊。邏輯對實體位址轉譯包括以下步驟：

步驟 800：給定一邏輯區段位址。

步驟 810：查詢控制器 RAM 中開啟之更新區塊清單 614 的給定邏輯位址(見圖 15 與 18)。如果查詢失敗，繼續進行至步驟 820，否則繼續進行至步驟 830。

步驟 820：在關閉的更新區塊清單 616 中查詢給定的邏輯位址。如果查詢失敗，則給定的邏輯位址並不屬於任何更新程序的部分；繼續進行至步驟 870，以進行 GAT 位址轉譯。否則繼續進行至步驟 860，以進行關閉的更新區塊位址轉譯。

步驟 830：如果含有給定邏輯位址的更新區塊為循序，則繼續進行至步驟 840，以進行循序更新區塊位址轉譯。否則繼續進行至步驟 850，以進行混亂更新區塊位址轉譯。

步驟 840：使用循序更新區塊位址轉譯來取得中繼區塊位址。繼續進行至步驟 880。

步驟 850：使用混亂更新區塊位址轉譯來取得中繼區塊位

址。繼續進行至步驟880。

步驟860：使用關閉的更新區塊位址轉譯來取得中繼區塊位址。繼續進行至步驟880。

步驟870：使用群組位址表(GAT)轉譯來取得中繼區塊位址。繼續進行至步驟880。

步驟880：將中繼區塊位址轉換為實體位址。轉譯方法端視中繼區塊是否已經重新連結而定。

步驟890：已取得實體區段位址。

下文將更詳細說明該等各種位址轉譯處理：

循序更新區塊位址轉譯(步驟840)

從開啟之更新區塊清單614 (圖15及18)的資訊即可直接完成和循序更新區塊關聯之邏輯群組中目標邏輯區段位址的位址轉譯，說明如下。

- 1.從清單的「頁面標記」及「寫入的區段號碼」欄位可決定目標邏輯區段是否已經配置在更新區塊或其關聯的原始區塊中。

- 2.從清單中可讀取適合目標邏輯區段的中繼區塊位址。

- 3.從合適的「頁面標記」欄位可決定中繼區塊內的區段位址。

混亂更新區塊位址轉譯(步驟850)

和混亂更新區塊關聯之邏輯群組中目標邏輯區段位址的位址轉譯序列如下。

- 1.如果從RAM中的混亂區段清單決定區段是最近寫入的區段，則直接從其在此清單中的位置即可完成位址轉譯。

2. 在 CBI 區塊中最近寫入的區段在其混亂區塊資料欄位內含有和目標邏輯區段位址相關之混亂更新區塊的實體位址。其在間接區段索引欄位內也含有有關此混亂更新區塊最後寫入之 CBI 區段之 CBI 區塊內的位移(見圖 16A-16E)。

3. 該些欄位中的資訊均被快取儲存於 RAM 之中，而不需要於後續的位址轉譯期間來讀取該區段。

4. 讀取步驟 3 由間接區段索引欄位所識別的 CBI 區段。

5. 將最近被存取之混亂更新子群的直接區段索引欄位快取儲存於 RAM 之中，而不需要實施步驟 4 處的讀取以重複存取相同的混亂更新區塊。

6. 在步驟 4 或步驟 5 讀取的直接區段索引欄位接著可識別有關含有目標邏輯區段位址之邏輯子群組的 CBI 區段。

7. 從步驟 6 中識別的 CBI 區段讀取目標邏輯區段位址的混亂區塊索引項目。

8. 該最近被讀取之混亂區塊索引欄位可被快取儲存於控制器 RAM 之中，而不需要實施步驟 4 與步驟 7 處的讀取以重複存取相同的邏輯子群。

9. 混亂區塊索引項目可定義目標邏輯區段在混亂更新區塊或關聯之原始區塊中的位置。如果目標邏輯區段的有效複本係在原始區塊中，則可使用原始中繼區塊及頁面標記資訊將其尋找。

關閉的更新區塊位址轉譯(步驟 860)

從關閉之更新區塊清單的資訊即可直接完成和關閉之更新區塊關聯之邏輯群組中目標邏輯區段位址的位址轉譯

(參見圖18)，說明如下。

1. 從清單中可讀取指派給目標邏輯群組的中繼區塊位址。

2. 從清單中的「頁面標記」欄位可決定中繼區塊內的區段位址。

GAT位址轉譯(步驟870)

如果邏輯群組不會受到開啟或關閉之區塊更新清單的參考，則其在GAT中的項目為有效。由GAT所參考之邏輯群組中目標邏輯區段位址的位址轉譯序列如下。

1. 評估RAM中可用GAT快取記憶體的范围，以決定目標邏輯群組的項目是否含在GAT快取記憶體中。

2. 如果在步驟1發現目標邏輯群組，則GAT快取記憶體含有完整的群組位址資訊，包括中繼區塊位址及頁面標記，因此允許轉譯目標邏輯區段位址。

3. 如果目標位址不在GAT快取記憶體中，則必須讀取目標GAT區塊的GAT索引，以識別有關目標邏輯群組位址之GAT區段的位置。

4. 最後存取之GAT區塊的GAT索引會保留在控制器RAM中，且不用從快閃記憶體讀取區段即可存取。

5. 將一份由每個GAT區塊之中繼區塊位址及被寫入每個GAT區塊之中的區段數量所組成的清單保存在控制器RAM之中。假使步驟4處無法取得必要的GAT索引，則可立刻從快閃記憶體之中讀取。

6. 從步驟4或步驟6處所獲得的GAT索引所定義的GAT區

塊中的區段位置中讀取有關目標邏輯群組位址的GAT區段。以含有目標項目之區段的子分割來更新GAT快取記憶體。

7.從目標GAT項目內的中繼區塊位址及「頁面標記」欄位取得目標區段位址。

中繼區塊對實體位址轉譯(步驟880)

如果和中繼區塊位址關聯的旗標代表中繼區塊已經被重新連結，則會從BLM區塊讀取相關的LT區段，以決定目標區段位址的抹除區塊位址。否則，會從中繼區塊位址決定抹除區塊位址。

控制資料管理

圖20顯示在記憶體管理的操作過程中，在控制資料結構上執行的操作階層。資料更新管理操作可對常駐在RAM中的各種清單發生作用。控制寫入操作可對快閃記憶體中各種控制資料區段及專用區塊發生作用，並還能和RAM中的清單交換資料。

資料更新管理操作會於RAM中針對ABL、CBL、以及該混亂區段清單來實施。當一已抹除區塊被配置為一更新區塊或控制區塊時，或是關閉一更新區塊時，便會更新該ABL。當抹除一控制區塊時，或是將一已關閉的更新區塊的某個項目寫入該GAT之中時，便會更新該CBL。當一區段被寫入一混亂更新區塊之中時，便會更新該更新混亂區段清單。

控制寫入操作會使得來自RAM中之控制資料結構的資訊

被寫入快閃記憶體中的控制資料結構之中，必要時會隨之更新快閃記憶體與RAM之中其它支援的控制資料結構。當該ABL不含欲被配置為更新區塊的已抹除區塊的任何其它項目時，或是再寫入該CBI區塊時，便會觸發控制寫入操作。

在較佳具體實施例中，會在每個控制寫入操作期間執行ABL填充操作、CBL清空操作、及EBM區段更新操作。當含有EBM區段的MAP區塊已滿時，會將有效的EBM及MAP區段複製至已配置的已抹除區塊，然後抹除先前的MAP區塊。

在每個控制寫入操作期間，寫入一個GAT區段，也會隨著修改關閉的更新區塊清單。當GAT區塊已滿時，將執行GAT再寫入操作。

如上述，經過幾次的混亂區段寫入作業之後，便會寫入一CBI區段。當CBI區塊變滿時，會將有效的CBI區段複製到已配置的抹除區塊中，然後抹除先前的CBI區塊。

如上述，MAP交換操作係執行於EBM區段的EBB清單中沒有其他已抹除的區塊項目時。

每次再寫入MAP區塊時，會在專用的MAPA區塊中寫入用於記錄MAP區塊之目前位址的MAP位址(MAPA)區段。當MAPA區塊已滿時，會將有效的MAPA區段複製至已配置的已抹除區塊，然後抹除先前的MAPA區塊。

每次再寫入MAPA區塊時，會將啟動區段寫入目前的啟動區塊中。當啟動區塊已滿時，會將目前版本之啟動區塊的有效啟動區段複製至備份版本，然後該版本再變成目前的

版本。先前的目前版本會被抹除並變成備份版本，並會將有效的啟動區段寫回其中。

分散在多個記憶體平面上之記憶體的對齊

如先前結合圖4及圖5A-5C所述，為了增加效能，會平行操作多個記憶體平面。基本上，各平面有其自己的感測放大器組作為讀取及程式電路的部分，以平行服務跨越平面之記憶體單元的對應頁面。在結合多個平面時，可平行操作多個頁面，使得效能更為提高。

根據本發明的另一方面，對於一被組織成複數個可抹除區塊且由多個記憶體平面所構成(因而可平行地讀取複數個邏輯單元或是將複數個邏輯單元平行地程式化至該等多個平面之中)的記憶體陣列，當要更新儲存於特定記憶體平面中第一區塊的原始邏輯單元時，會供應所需以將已更新的邏輯單元保持在和原始相同的平面中。這可藉由以下方式來完成：將已更新的邏輯單元記錄到仍在相同平面中之第二區塊的下一個可用位置。較佳將邏輯單元儲存在平面中和其中其他版本相同的位移位置，使得給定邏輯單元的所有版本係由相同組的感測電路予以服務。

據此，在一項較佳具體實施例中，以邏輯單元的目前版本來填補介於上一個程式化記憶體單元與下一個可用平面對齊記憶體單元之間的任何中間間隙。將邏輯上位於該最後被程式化之邏輯單元後面的該等邏輯單元的目前版本以及邏輯上位於被儲存在該下一個可用的平面排列記憶體單元中之邏輯單元前面的該等邏輯單元的目前版本填入間隙

中，便可完成該填補作業。

依此方式，可將邏輯單元的所有版本維持在具有和原始相同位移的相同平面中，致使在廢棄項目收集操作中，不必從不同平面擷取邏輯單元的最新版本，以免降低效能。在一項較佳具體實施例中，可利用該等最新的版本來更新或填補該平面上的每個記憶體單元。因此，便可從每個平面中平行地讀出一邏輯單元，其將會具有邏輯順序而無需進一步重新排列。

此方案藉由允許平面上重新排列邏輯群組之邏輯單元的最新版本，且不必收集不同記憶體平面的最新版本，而縮短彙總混亂區塊的時間。這很有好處，其中主機介面的效能規格可定義由記憶體系統完成區段寫入操作的最大等待時間。

圖21顯示以多個記憶體平面所構成的記憶體陣列。記憶體平面可以來自相同的記憶體晶片或多個記憶體晶片。各平面910具有其自己的讀取及程式電路912以平行服務記憶體單元的頁面914。在不失一般性之下，在所示的範例中，記憶體陣列具有四個平行操作的平面。

一般而言，邏輯單元是主機系統存取的最小單元。通常一個邏輯單元是尺寸512位元組的區段。頁面是平面中平行讀取或程式化的最大單元。通常一個邏輯頁面含有一或多個邏輯單元。因此，在結合多個平面時，可將平行讀取或程式化的最大總數單元視為記憶體單元的中繼頁面，其中中繼頁面係由多個平面中之各平面的頁面所構成。例如，

如 MP_0 之中繼頁面具有四個頁面，即來自各平面 P_0 、 P_1 、 P_2 、及 P_3 的頁面，其中平行儲存邏輯頁面 LP_0 、 LP_1 、 LP_2 、 LP_3 。因此，和僅在一個平面中的操作相比，記憶體讀取及寫入效能增加四倍。

記憶體陣列會進一步組織成複數個中繼區塊，如 MB_0 、...、 MB_j ，其中各中繼區塊內的所有記憶體單元可成為一個單元一起抹除。如 MB_0 的中繼區塊係以多個記憶體位置所構成，以儲存資料的邏輯頁面 914，如 LP_0 - LP_{N-1} 。中繼區塊中的邏輯頁面係根據其填充於中繼區塊的順序，按預定的序列分布於四個平面 P_0 、 P_1 、 P_2 及 P_3 中。例如，在按邏輯上循序順序填充邏輯頁面時，會以第一平面中第一頁面、第二平面中第二頁面等等的循環順序造訪平面。在到達最後的平面後，填充會以循環的方式返回，以從下一個中繼頁面的第一平面重新開始。依此方式，即可在所有平面均為平行操作時平行存取連續的邏輯頁面。

一般而言，如果有 W 個平面平行操作中及中繼區塊係按邏輯上循序順序進行填充，則中繼區塊中第 k 個邏輯頁面將常駐在平面 x 中，其中 $x = k \text{ MOD } W$ 。例如，有四個平面， $W = 4$ ，在按邏輯循序順序填充區塊時，第 5 個邏輯頁面 LP_5 將常駐在由 $5 \text{ MOD } 4$ 給定的平面中，即平面 1，如圖 21 所示。

各記憶體平面中的記憶體操作係由一組讀取/寫入電路 912 來執行。進出各讀取/寫入電路的資料係透過在控制器 920 之控制下的資料匯流排 930 進行傳送。控制器 920 中的緩衝器 922 可經由資料匯流排 930 協助緩衝資料的傳送。尤其

在第一平面的操作需要存取第二平面的資料時，將需要兩個步驟的程序。控制器會先讀出第二平面的資料，然後經由資料匯流排及緩衝器傳送至第一平面。事實上，在大多數的記憶體架構中，在兩個不同的位元線之間傳送資料也需要透過資料匯流排920交換資料。

至少，這涉及在一平面中從一組讀取/寫入電路傳送出去，然後進入另一平面中的另一組讀取/寫入電路。在其中平面係來自不同晶片的例子中，將需要在晶片之間傳送。本發明可提供記憶體區塊管理的結構及方案，以避免一個平面從另一個平面存取資料，以便將效能最大化。

如圖21所示，一中繼頁面係由多個邏輯頁(各位於其中一個平面之中)所構成。每個邏輯頁可能係由一個以上的邏輯單元所組成。當資料欲以逐個邏輯單元的方式被記錄於一跨越該等平面的區塊中時，每個邏輯單元便將會落在該等四個記憶體平面之一中。

在更新邏輯單元時會發生平面對齊的問題。在目前的範例中，為了便於解說，將邏輯單元視為512位元組的邏輯區段，一個邏輯頁面也是一個邏輯單元寬。由於快閃記憶體不允許未先抹除整個區塊而再寫入區塊的一部分，因此不會將邏輯頁面的更新寫入現有的位置之上，而是將其記錄在區塊未使用的位置中。然後會將邏輯單元的先前版本視為淘汰。在一些更新後，區塊可含有一些由於已經更新因此變成淘汰的邏輯單元。然後此區塊可以說是「不乾淨」，而廢棄項目收集操作會忽略不乾淨的邏輯單元而收集各個

邏輯單元的最新版本並按邏輯上循序順序將其重新記錄在一或多個新的區塊中。然後抹除及再循環不乾淨的區塊。

當該已更新邏輯單元被記錄於某一區塊中下個未被使用的位置之中時，其通常不會被記錄於和先前版本相同的記憶體平面之中。當要進行廢棄項目收集操作時，如彙總或壓縮，一邏輯單元的最新版本便會被記錄於和原來相同的平面之中，以維持原來的順序。然而，如果必須從另一個平面擷取最新版本，效能將會降低。

因此，根據本發明的另一方面，給定平面之第一區塊的原始邏輯單元。這可藉由以下方式來完成：將已更新的邏輯單元記錄到仍在相同平面中之第二區塊的下一個可用位置。在一項較佳具體實施例中，會以和原始區塊中原始邏輯單元的相同相對位置之邏輯單元的目前版本，填補(即，藉由複製來填充)任何在上一個程式化記憶體單元和下一個可用平面對齊記憶體單元之間的中間間隙。

圖22A顯示根據本發明的一般實施例，具有平面對齊之更新之方法的流程圖。

步驟950：於一被組織成複數個區塊的非揮發性記憶體之中，每個區塊均被分割成可一起抹除的複數個記憶體單元，每個記憶體單元係用於儲存一邏輯單元的資料。

步驟952：以多個記憶體平面構成記憶體，各平面具有一組用於平行服務記憶體頁面的感測電路，該記憶體頁面含有一或多個記憶體單元。

步驟954：依照第一順序將邏輯單元的第一版本儲存於一

第一區塊的複數個記憶體單元之中，每個第一版本邏輯單元均被儲存於該等記憶體平面之一中。

步驟956：依照不同於第一順序的第二順序將邏輯單元的後續版本儲存於一第二區塊之中，每個後續版本均被儲存於和該第一版本相同的記憶體平面中下一個可用的記憶體單元之中，以便可利用該組相同的感測電路從該相同的平面中來存取一邏輯單元的所有的版本。

圖22B顯示在圖22A所示之流程圖中儲存更新之步驟的較佳具體實施例。

步驟956'包括步驟957、步驟958及步驟959。

步驟957：將各區塊分割成中繼頁面，各中繼頁面係以各平面的一頁面所構成。此步驟可在儲存步驟的任一項之前執行。

步驟958：根據和第一順序不同的第二順序將邏輯單元的後續版本儲存至第二區塊，各後續版本係儲存於具有中繼頁面中和第一版本相同位移的下一個可用記憶體單元中。

步驟959：和儲存邏輯單元的後續版本同時進行，根據第一順序複製邏輯單元的目前版本，以逐個中繼頁面的方式來填補在該下一個可用記憶體單元之前的任何未使用記憶體單元。

圖23A顯示不顧平面對齊按循序順序寫入至循序更新區塊之邏輯單元的範例。該範例顯示出每個邏輯頁的大小均係一個邏輯區段，例如LS0、LS1...。於該四平面的範例中，每個區塊(例如MB₀)均可視為被分割成複數個中繼頁面

MP_0 、 MP_1 、...，其中每個中繼頁面(例如 MP_0)均含有四個區段(例如LS0、LS1、LS2、以及LS3)，每個區段分別來自平面P0、P1、P2、以及P3。所以，該區塊會以循環順序逐個區段地被填入平面P0、P1、P2、以及P3中的邏輯單元之中。

在主機寫入操作#1中，正在更新邏輯區段LS5-LS8中的資料。會將更新成為LS5'-LS8'的資料記錄在始於第一可用位置之新配置的更新區塊中。

在主機寫入操作#2中，正在更新邏輯區段LS9-LS12中資料的程式段。會將更新成為LS9'-LS12'的資料記錄在緊接在最後寫入結束處之後之位置中的更新區塊中。圖中顯示兩次主機寫入的方式係以邏輯循序方式將該更新資料記錄於該更新區塊之中，即LS5'-LS12'。更新區塊可視為循序更新區塊，因其已按邏輯上循序的順序被填補。記錄在更新區塊中的更新資料可淘汰原始區塊中對應的資料。

然而，更新邏輯區段係根據下一個可用位置但卻不顧平面對齊而記錄在更新區塊中。例如，區段LS5原來是記錄在平面P1中，但已更新的LS5'現在則記錄在P0中。同樣地，其他更新區段全部無法對齊。

圖23B顯示不顧平面對齊按非循序順序寫入混亂更新區塊之邏輯單元的範例。

在主機寫入操作#1中，會更新儲存於原始中繼區塊之給定邏輯群組的邏輯區段LS10-LS11。已更新的邏輯區段LS10'-LS11'會被儲存到新配置的更新區塊中。此時，更新

區塊為循序的更新區塊。在主機寫入操作#2中，會將邏輯區段LS5-LS6更新成為LS5'-LS6'及將其記錄在緊接上一個寫入之後之位置的更新區塊中。這可將循序的更新區塊轉換為混亂的更新區塊。在主機寫入操作#3，再次更新邏輯區段LS10'及將其記錄在更新區塊的下一個位置中成為LS10"。此時，更新區塊中的LS10"可取代先前記錄中的LS10'，而LS10'又可取代原始區塊中的LS10。在主機寫入操作#4中，再次更新邏輯區段LS10的資料及將其記錄在更新區塊的下一個位置中成為LS10"。因此，LS10"現在是邏輯區段LS10的最後且唯一有效的版本。LS10的所有先前版本現在均已淘汰。在主機寫入操作#5中，會更新邏輯區段LS30的資料及將其記錄在更新區塊中成為LS30'。在此範例中，可按任何順序及以任何重複將邏輯群組內的邏輯單元寫入至混亂更新區塊。

同樣地，更新邏輯區段係根據下一個可用位置但卻不顧平面對齊而記錄在更新區塊中。例如，區段LS10原來是記錄在平面P2(即，MP₂、第三平面)中，但更新LS10'現在卻記錄在P0(即，MP₀'，第一平面)中。同樣地，在主機寫入#3中，會再次將邏輯區段LS10'更新成為LS10"並被放在結果也在平面P0(MP₁'的第一平面)的下一個可用位置中。因此，一般而言，從圖中可見，將更新區段記錄至區塊的下一個可用位置會使更新區段被儲存在和其先前版本不同的平面中。

具有以填補填充之中間間隙之平面對齊的循序更新區塊

圖 24A 顯示根據本發明的一項較佳具體實施例，具有平面對齊及填補之圖 23A 的循序更新範例。

在主機寫入操作 #1 中，會將更新成為 LS5'-LS8' 的資料記錄在始於第一可用平面對齊位置之新配置的更新區塊。在此例中，LS5 原來在 P1 中，P1 是中繼頁面的第二平面。因此，會在更新區塊之第一可用中繼頁面 MP₀' 的對應平面中程式化 LS5'-LS7'。同時，會以原始區塊中繼頁面中在 LS5 前之邏輯區段 LS4 的目前版本填補 MP₀' 中未使用之第一平面的間隙。然後將原始 LS4 處理成淘汰的資料。然後將剩餘的 LS8' 記錄在下一個中繼頁面 MP₁' 的第一個平面中並已平面對齊。

在主機寫入操作 #2 中，會將更新成為 LS9'-LS12' 的資料記錄在下一個可用平面對齊位置的更新區塊中。因此，會將 LS9' 記錄在下一個可用的平面對齊記憶體單元中，即 MP₁' 的第二平面。此時，不會造成任何間隙，也不需要任何填補。更新區塊可視為循序更新區塊，因其已按邏輯上循序的順序填入。此外，其將因各更新邏輯單元和其原始的一樣都在相同平面中而已平面對齊。

具中間間隙之平面對齊的混亂更新區塊

圖 24B 顯示根據本發明的一項較佳具體實施例，具有平面對齊及不具有任何填補之圖 23B 的混亂更新範例。

在主機寫入操作 #1 中，將已更新的邏輯區段 LS10'-LS11' 儲存在新配置的更新區塊中。並不會將其儲存在下一個可用的記憶體單元中，而是將其儲存在下一個可用的平面對

齊記憶體單元中。由於LS10'及LS11'原來係分別儲存在平面P2及P3(原始區塊之MP₂的第三及第四平面)，下一個可用的平面對齊記憶體單元將在更新區塊之MP₀'的第三及第四平面中。此時，更新區塊為非循序，其中將按「未填充」、「未填充」、LS10'及LS11'的順序填充中繼頁面MP₀的頁面。

在主機寫入操作#2中，會將邏輯區段LS5-LS6更新成為LS5'-LS6'及將其記錄在下一個可用之平面對齊的更新區塊中。因此，會將原始區塊中在MP₁之第二(P1)及第三(P2)平面或記憶體單元的LS5'及LS6'程式化至更新區塊中下一個可用中繼頁面MP₁'之對應的平面中。這會在MP₁'中留下在前面之未使用的第一平面。

在主機寫入操作#3，再次更新邏輯區段LS10'及將其記錄在更新區塊的下一個平面對齊位置中成為LS10"。因此，會將其寫入下一個可用的第三平面，即在MP₂'中。這會在MP₁'的最後平面及MP₂'的前兩個平面中留下位在前面的間隙。這將會淘汰在MP₀'中的LS10'。

在主機寫入操作#4中，會再次更新邏輯區段LS10"中的資料並將其記錄在更新區塊中中繼頁面MP₂'的下一個可用第三平面中成為LS10"'。因此，LS10"'現在是邏輯區段LS10的最後且唯一有效的版本。這會留下由MP₂'之最後平面及MP₃'之前兩個平面所組成的間隙。

在主機寫入操作#5中，會更新邏輯區段LS30的資料及將其記錄在更新區塊中成為LS30'。由於原始的LS30常駐於中繼頁面的P2或第三平面中，因此會將其寫入更新區塊中下

一個可用的第三平面。此時，其將是 MP_4' 的第三平面。將因 MP_3' 的最後平面與 MP_4' 的前兩個平面而產生間隙。因此，此範例顯示可以平面對齊的方式，按照任何順序及任意重複，將一邏輯群組內的複數個邏輯區段寫入至一混亂更新區塊中。在後續的廢棄項目收集操作中，將便利地由相同組的感測電路來服務給定邏輯區段的所有版本，尤其是最新版本。

具有以填補填充之中間間隙之平面對齊的混亂更新區塊

圖24C顯示根據本發明的另一項較佳具體實施例，具有平面對齊及填補之圖23B的混亂更新範例。

此操作和圖24B所示的相同，但中間間隙會先以填補加以填充。在主機寫入操作#1中，會先以常駐在原始區塊之LS8及LS9的目前版本，來填補由中繼頁面 MP_0' 之第一及第二未使用的平面所產生的間隙。這會使原始區塊中的LS8及LS9淘汰。此時，更新區塊是循序的更新區塊，其中中繼頁面 MP_0' 的填充順序為LS8、LS9、LS10'及LS11'。

在主機寫入操作#2中，將因 MP_1' 中在前面之未使用的第一平面而產生間隙，其將先以LS4進行填補。這將使原始區塊中的LS4淘汰。和之前一樣，第二寫入可將循序更新區塊轉換為混亂更新區塊。

在主機寫入操作#3中，將因 MP_1' 中未使用的最後平面及 MP_2' 的前兩個平面而產生間隙。會先以在上一個程式化之LS6'之後的LS7來填補 MP_1' 的最後平面，然後以在LS10之前的邏輯單元(即LS8及LS9)來填補 MP_2' 的前兩個平面。這會

淘汰MP₀'中的LS10'及原始區塊中的LS7-LS9。

在主機寫入操作#4中，將產生由MP₂'的最後平面及MP₃'的前兩個平面所組成的間隙。MP₂'的最後平面可由中繼頁面MP₂'中在最後寫入之LS10''之後之邏輯單元目前版本的LS11'進行填補。MP₃'的前兩個平面分別可藉由LS8及LS9進行填補，和中繼頁面MP₃'中在LS10'''之前的邏輯單元一樣。

在主機寫入操作#5中，也會跟著分別以LS11'、LS28及LS29來填補從MP₃'之最後平面至MP₄'前兩個平面之間的間隙。因此，此範例顯示可以平面對齊的方式，按照任何順序及任意重複，將一邏輯群組內的複數個邏輯區段寫入至一混亂更新區塊中。

在較佳具體實施例中，一個中繼頁面含有來自個別平面的循環頁面。由於中繼頁面可以平行進行讀取或程式化，因此以中繼頁面的粒度實施主機更新會很方便。在有任何填補時，其可和按中繼頁面的更新邏輯單元一起記錄。

在圖24A及圖24C之範例顯示的具體實施例中，在各主機寫入期間，會在要程式化更新之平面對齊記憶體單元之前未使用的記憶體單元上執行填補。在下一個主機寫入之前，會延後在上一個程式化之記憶體單元後之任何未使用記憶體單元的動作。一般而言，會在各中繼頁面的邊界內填補任何在前面之未使用的記憶體單元。換言之，如果在前面的間隙跨越於兩個中繼頁面之上，則按各中繼頁面合適的邏輯上循序順序在各中繼頁面上執行填補，但不顧橫

跨邊界的連續性。在彙總區塊時，最後寫入的中繼頁面，如為部分寫入，可藉由填補進行完整填充。

在另一項具體實施例中，任何部分填充的中繼頁面可在移至下一個中繼頁面之前進行完全填補。

記憶體單元粒度

根據個別記憶體架構所支援的彈性，讀取或程式化的單元可以有各種變化。個別平面的獨立特性允許獨立讀取及程式化中繼頁面中個別平面的各頁面。上述範例具有成為各平面中頁面的程式化最大單元。在中繼頁面內，可以有小於所有頁面的局部中繼頁面程式化。例如，可以程式化中繼頁面的前三個頁面，然後再程式化第四個頁面。

還有，在平面層級，一個實體頁面可含有一或多個記憶體單元。如果各記憶體單元可以儲存一個區段的資料，則一個實體頁面可儲存一或多個區段。一些記憶體架構可支援局部頁面程式化，其中藉由抑制頁面內選定記憶體單元的程式化，可在多個程式化編碼過程上，在不同的時間個別程式化選定的邏輯單元。

在記憶體平面內用於邏輯群組之混亂更新的邏輯單元對齊

在區塊記憶體管理系統中，按邏輯上循序順序將邏輯單元的邏輯群組儲存於原始區塊中。在更新邏輯群組時，會將邏輯單元的後續版本儲存於更新區塊中。如果將邏輯單元混亂地(即，非循序)儲存於更新區塊中，最後會執行廢棄項目收集以收集原始區塊及更新區塊中邏輯單元的最新版本，以循序將其整合為新的原始區塊。如果將給定邏輯單

元的更新版本全部儲存在和其原始區塊中原始版本對齊的更新區塊中，致使相同組的感測電路可以存取所有版本，則廢棄項目收集操作會更有效。

根據本發明的另一方面，在上述區塊記憶體管理系統中，在將記憶體組織成一系列的記憶體頁面時(其中記憶體單元的各頁面係由一組感測電路進行平行服務)，如果給定邏輯單元的所有版本在所儲存的頁面中全部具有相同的位移位置，則所有版本均已對齊。

圖 25 顯示其中各頁面含有兩個用於儲存兩個邏輯單元(如兩個邏輯區段)之記憶體單元的範例記憶體組織。在原始區塊中，由於邏輯區段係按邏輯上循序順序加以儲存，會將邏輯區段 LS0 及 LS1 儲存於頁面 P0 中，將邏輯區段 LS2 及 LS3 儲存於頁面 P1 中，及將邏輯區段 LS4 及 LS5 儲存於頁面 P3 中等。可以看出在此兩個區段的頁面中，左邊算起第一個區段的頁位移為「0」，而第二個區段的頁位移為「1」。

在更新循序儲存於原始區塊中之邏輯區段的邏輯群組時，會將已更新的邏輯區段記錄在更新區塊中。例如，邏輯區段 LS2 常駐在原始區塊中具有位移「0」的頁面 P0 中。如果在第一寫入中，如果將 LS2 更新為 LS2'，則會將其儲存於具有相同頁面位移「0」之更新區塊的第一可用位置。這會是在頁面 P0' 的第一記憶體單元中。如果在第二寫入中，將 LS5 更新為 LS5'，則會將其儲存於具有相同頁面位移「1」之更新區塊的第一可用位置。這會是在具有頁面 P1' 之位移「1」的第二記憶體單元中。然而，在儲存 LS5' 之前，會在

其中複製至少在各頁面中將會維持邏輯循序順序之邏輯區段的最新版本，以先填補P0'中具有位移「1」的未使用記憶體單元及P1'中的位移「0」。此時，會將LS3複製至P0'中位移「1」位置及將LS4複製至P1'中位移「0」位置。如果在第三寫入中，再次將LS2'更新為LS2"，則會將其儲存在P2'的位移「0」中。如果在第四寫入中，分別將LS22及LS23更新為LS22'及LS23'，則會分別將其儲存在P3'的位移「0」及「1」中。然而，在那之前，會以LS3填補在P2'中具有位移「1」的未使用記憶體單元。

上述更新序列假設可以在頁面內程式化個別區段。對於一些其中不支援局部頁面程式化的記憶體架構，必須一起程式化頁面內的區段。此時，在第一寫入中，會將LS2'及LS3一起程式化至P0'中。在第二寫入中，會將LS4及LS5'一起程式化至P1'中。在第三寫入中，會將LS2"及LS3一起程式化至P2'中等等。

中繼頁面內的平面對齊

或者，程式化的單元可具有中繼頁面的粒度。如果寫入混亂更新區塊的粒度變成中繼頁面，則結合圖16A及16B所述之CBI區塊中的項目將和中繼頁面有關，而非和區段有關。增加的粒度會減少必須為混亂更新區塊所記錄之項目的數量，並允許直接消除索引及每個中繼區塊使用單一CBI區段。

圖26A和圖21的記憶體結構相同，只是各頁面含有兩個區段而非一個。因此，從圖中可見中繼頁面MP₀現在各有其能

夠儲存兩個邏輯單元之資料的頁面。如果各邏輯單元是一個區段，則將邏輯區段循序儲存於平面P0中LS0及LS1及平面P1中LS2及LS3等的 MP_0 之中。

圖26B顯示圖26A所示之具有以線性圖方式布局之記憶體單元的中繼區塊。和圖21的單一區段頁面相比，邏輯區段係以循環的方式儲存於各頁面中具有兩個區段的四個頁面中。

一般而言，如果有W個平行操作的平面及每個頁面有K個記憶體單元，且按邏輯上循序順序填充中繼區塊，則中繼區塊中第k個邏輯頁面將常駐於平面x中，其中 $x = k' \text{ MOD } W$ ，其中 $k' = \text{INT}(k/K)$ 。例如，有四個平面， $W = 4$ ，每個頁面2個區段， $K = 2$ ，則對於 $k = 5$ ，即指第五個邏輯區段LS5，其將常駐於由 $2 \text{ MOD } 4$ 所給定的平面中，即平面2，如圖24A所示。一般而言，相同原理適用於實施上述的平面對齊。

上述範例係用於多重平面架構中平面和頁面的對齊。在具有多個區段之頁面的例子中，也維持頁面內的區段對齊會很有利。依此方式，使用相同組的感測電路有利於相同邏輯區段的不同版本。可有效執行如區段之重新配置及「讀取-修改-寫入」的操作。在對齊頁面內的區段順序時，可以採用和對齊頁面與平面的相同技術。還有根據具體實施例而定，可以填補也可以不填補任何中間間隙。

不用填補的邏輯單元平面對齊

圖27顯示的替代性方案如下：不用填補要從一個位置複

製到另一個的邏輯單元，即可在更新區塊中進行平面對齊。可將和更新區塊交叉之四個平面的部分視為四個收集接收自主機之平面對齊的已更新邏輯單元的緩衝器。不用在合適緩衝器的下一個可用記憶體單元中填補，即可程式化各接收自主機的邏輯單元。依照接收自該主機的邏輯單元位址的序列，可能會有不同數量的邏輯單元被程式化於每個平面之中。

混亂更新區塊 MB'_1 可含有邏輯中繼頁面之所有邏輯單元的已更新版本，如用於 MP'_0 的。其還可含有小於中繼頁面的所有邏輯單元，如用於 MP'_1 的。在 MP'_1 的例子中，可從對應的原始區塊 MB_0 取得遺失的邏輯單元 LS_4 。

此替代性方案在記憶體架構可支援平行讀取各平面的任意邏輯頁面時尤其有效。依此方式，可在單一平行讀取操作中讀取一個中繼頁面的所有邏輯頁面，即使個別邏輯頁面並非來自相同列。

階段性程式錯誤處置

當區塊中有程式失敗時，則通常會將所有要儲存至區塊的資料移至另一個區塊並將失敗的區塊標示為不良。根據其中遇到失敗之操作的時序規格，可能沒有足夠的時間可另外將儲存的資料移至另一個區塊。最壞的情況是在正常廢棄項目收集操作期間的程式失敗，其中需要另一個相同廢棄項目收集操作以將所有資料重新配置至另一個區塊。於此情況中，可能會違反一給定主機/記憶體裝置所規定的寫入等待時間限制，其該限制通常係被設計成容納一次(而

非兩次)廢棄項目收集操作。

圖28顯示其中缺陷區塊在彙總操作期間發生程式失敗時會在另一個區塊上重複彙總操作的方案。在此範例中，區塊1是按邏輯上循序順序儲存邏輯群組之完整邏輯單元的原始區塊。為了便於解說，原始區塊含有區段A、B、C、及D，各儲存一個子群組的邏輯單元。當主機更新群組的特定邏輯單元時，會將邏輯單元的較新版本記錄在更新區塊中，即區塊2。如先前結合更新區塊所述，根據主機而定，此更新可按循序或非循序(混亂)順序記錄邏輯單元。最後，會因更新區塊已滿或一些其他原因而關閉更新區塊以接收進一步更新。當更新區塊(區塊2)關閉時，會將常駐於更新區塊或原始區塊(區塊1)上之邏輯單元的目前版本彙總在新的區塊(區塊3)上，以形成邏輯群組之新的原始區塊。此範例顯示更新區塊在區段B及D中含有邏輯單元的較新版本。為了方便，圖中將區段B及D顯示在區塊2中未必是其記錄的位置，而是對齊其在區塊1中原始位置的位置。

在彙總操作中，會按循序順序將原來常駐於區塊1之邏輯群組之所有邏輯單元的目前版本記錄於彙總區塊(區塊3)中。因此，會先從區塊1將區段A的邏輯單元複製至區塊3，接著再從區塊2將區段B複製至區塊3。在此範例中，在從區塊1將區段C的邏輯單元複製至區塊3時，區塊3的缺陷將導致程式失敗。

一種處置程式失敗的方式是在全新的區塊(區塊4)上重新啟動彙總程序。因此，會將區段A、B、C、D複製在區塊

4上，然後丟棄缺陷區塊3。然而，這將表示執行兩個串聯的彙總操作，結果造成複製多達兩個充滿邏輯單元的區塊。

記憶體裝置具有完成特定操作的特定時間容限。例如，在主機寫入記憶體裝置時，會預計寫入操作在指定的時間內完成，已知為「寫入等待時間」。當記憶體裝置，如記憶卡，正忙於寫入主機的資料時，會發信「BUSY(忙碌)」狀態給主機。如果「BUSY」狀態持續超過寫入等待時間的長度，主機會使寫入操作逾時，然後對寫入操作登錄例外或錯誤。

圖29以示意圖顯示具有允許足夠時間完成寫入(更新)操作及彙總操作之時序或寫入等待時間的主機寫入操作。主機寫入操作具有寫入等待時間 T_w ，其可提供足夠完成寫入主機資料至更新區塊之更新操作972的時間(圖29(A))。如先前在區塊管理系統所述，對更新區塊的主機寫入可觸發彙總操作。因此，時序亦允許在更新操作972之外的彙總操作974(圖29(B))。然而，必須重新啟動彙總操作以回應失敗的彙總操作將會花費太多時間並超過指定的寫入等待時間。

根據本發明的另一方面，在具有區塊管理系統的記憶體中，在時間緊急的記憶體操作期間，區塊中的程式失敗可藉由繼續中斷區塊(breakout block)中的程式化操作來處置。稍後，在較不緊急的時間，可將中斷前記錄在失敗區塊中的資料傳送到其他可能也是中斷區塊的區塊。接著即可丟棄失敗的區塊。依此方式，在遇到缺陷區塊時，不會因必須立刻傳送缺陷區塊中儲存的資料而損失資料及超過

指定的時間限制，即可加以處理。此錯誤處置對於廢棄項目收集操作尤其重要，因此在緊急時間期間不需要對一嶄新的區塊重複進行整個作業。其後，在適宜的時間，藉由重新配置到其他區塊，即可挽救缺陷區塊的資料。

圖 30 顯示根據本發明一般方案之程式失敗處置的流程圖。

步驟 1002：將非揮發性記憶體組織成區塊，將各區塊分割成可一起抹除的記憶體單元，各記憶體單元可儲存一邏輯單元的資料。

程式失敗處置(第一階段)

步驟 1012：在第一區塊中儲存一連串邏輯單元的資料。

步驟 1014：為了回應儲存一些邏輯單元後在第一區塊的儲存失敗，在作為第一區塊之中斷區塊的第二區塊中儲存後續的邏輯單元。

程式失敗處置(最後階段)

步驟 1020：為了回應預定的事件，將儲存在第一區塊的邏輯單元傳送至第三區塊，其中第三區塊和第二區塊可以相同或不同。

步驟 1022：丟棄第一區塊。

圖 31A 顯示程式失敗處置的一項具體實施例，其中第三(最後的重新配置)區塊和第二(中斷)區塊不同。在階段 I 期間，會在第一區塊上記錄一連串的邏輯單元。如果邏輯單元是來自主機寫入，則可將第一區塊視為更新區塊。如果邏輯單元是來自壓縮操作的彙總，則可將第一區塊視為重

新配置區塊。如果在某個點在區塊1中遇到程式失敗，則可提供當作中斷區塊的第二區塊。在區塊1及後續邏輯單元中記錄失敗的邏輯單元會被記錄在中斷區塊上。依此方式，不需要額外的時間來取代失敗的區塊1及常駐其上的資料。

在中間階段II中，可在區塊1及區塊2之間取得序列中所有的已記錄邏輯單元。

在最後階段III中，會將邏輯單元重新配置至可當作重新配置區塊的區塊3，以取代失敗的區塊1及常駐其上的資料。因此，可挽救已失敗區塊中的資料，然後再丟棄失敗的區塊。會排定最後階段的時間，使其不會和任何同時之記憶體操作的時序衝突。

在此具體實施例中，重新配置區塊3和中斷區塊2有所區分。這在中間階段期間已經以附加的邏輯單元記錄中斷區塊時會很方便。因此，中斷區塊已經變成更新區塊，可能不適用於將缺陷區塊1的邏輯單元重新配置至其中。

圖31B顯示程式失敗處置的另一項具體實施例，其中第三(最後的重新配置)區塊和第二(中斷)區塊相同。階段I及II和圖31A所示的第一具體實施例相同。然而，在階段III中，會將缺陷區塊1的邏輯單元重新配置至中斷區塊2。這在未以先前寫入操作之原始序列以外的附加邏輯單元記錄中斷區塊2時會很方便。依此方式，儲存所論邏輯單元所需的區塊最小。

在彙總期間之程式失敗處置的具體實施例

程式失敗處置在彙總操作期間尤其重要。正常的彙總操

作可將常駐在原始區塊及更新區塊中之邏輯群組的所有邏輯單元的目前版本彙總至彙總區塊。在彙總操作期間，如果在彙總區塊中發生程式失敗，則會提供另一個當作中斷彙總區塊的區塊，以接收其餘邏輯單元的彙總。依此方式，不必複製邏輯單元一次以上，而仍可在正常彙總操作指定的期間內完成例外處理的操作。在適宜的時間，將群組所有未處理完成之邏輯單元彙總至中斷區塊中，即可完成彙總操作。適宜的時間將是在目前主機寫入操作以外的一些其他有時間執行彙總之期間的期間。一個此種適宜的時間是在另一個其中有更新但無關聯之彙總操作之主機寫入的期間。

實質上，可將程式失敗處置的彙總視為以多階段來實施。在第一階段中，在發生程式失敗後，會將邏輯單元彙總至一個以上區塊中以避免彙總各邏輯單元一次以上。在適宜的時間會完成最後階段，其中會將邏輯群組彙總至一個區塊中，較佳藉由按循序順序將所有邏輯單元收集至中斷彙總區塊中。

圖 32A 顯示造成彙總操作之初始更新操作的流程圖。

步驟 1102：將非揮發性記憶體組織成區塊，將各區塊分割成可一起抹除的記憶體單元，各記憶體單元可儲存一邏輯單元的資料。

步驟 1104：將資料組織成複數個邏輯群組，各邏輯群組為可儲存在區塊中之邏輯單元的群組。

步驟 1112：接收封裝在邏輯單元中的主機資料。

步驟1114：根據第一順序在第一區塊中儲存邏輯群組之邏輯單元的第一版本，以建立邏輯群組的原始區塊。

步驟1116：根據第二順序在第二區塊中儲存包括邏輯群組之邏輯單元的後續版本，以建立邏輯群組的更新區塊。

步驟1119：在上述部分的預定事件，執行廢棄項目收集以在各種區塊中收集邏輯單元的目前版本，及將其重新記錄至新的區塊。

圖32B顯示根據本發明一項較佳具體實施例之多階段彙總操作的流程圖。

彙總失敗處置(階段I)

錯誤處置的彙總，階段I操作1120包含步驟1122及步驟1124。

步驟1122：以和第一順序相同的順序，在第三區塊中儲存該邏輯群組之邏輯單元的目前版本，以建立邏輯群組的彙總區塊。

步驟1124：為了回應彙總區塊的儲存失敗，以和第一順序相同的順序，在第四區塊中儲存該第三區塊所沒有之邏輯群組的邏輯單元，以提供中斷彙總區塊。

由於已將區塊1及區塊2中的資料傳送至區塊3及區塊4，因此可抹除區塊1及區塊2以釋放空間。在較佳具體實施例中，可立即釋放區塊2至EBL(已抹除的區塊清單，見圖18)再予以使用。區塊1只能在以下的條件下釋放：如果其為關閉的更新區塊及其中有對應的GAT項目指向的另一個區塊。

實質上，區塊3會變成邏輯群組的原始區塊，而區塊4變成區塊3的取代循序更新區塊。

在完成階段I彙總後，記憶體裝置藉由釋放BUSY信號來發信給主機。

中間操作(階段II)

階段II，即中間操作1130，可在階段III彙總操作1140之前發生。如步驟1132、1134、1136中任一者所提出般，可能會有數種可能的情況。

步驟1132：或是在邏輯群組的寫入操作中，寫入作為更新區塊的第四區塊(中斷彙總區塊)。

如果主機寫入所論邏輯群組，則區塊4(其為中斷彙總區塊且其至此已是取代循序更新區塊)將用作正常更新區塊。根據主機寫入而定，其可維持循序或變成混亂狀態。作為更新區塊，其將在某個點觸發關閉另一個混亂區塊，如先前的較佳具體實施例所述。

如果主機寫入另一個邏輯群組，則直接進行至階段III操作。

步驟1134：或是在讀取操作中，讀取其中第三區塊為邏輯群組原始區塊及第四區塊為更新區塊的記憶體。

此時，會從為邏輯群組之原始區塊的區塊3讀取區段A及B的邏輯單元，及從為群組之更新區塊的區塊4讀取區段C及D的邏輯單元。由於從區塊3只能讀取區段A及B，將無法存取其中程式化失敗的頁面，且無法存取其後未寫入的部分。雖然尚未更新快閃記憶體中的GAT目錄及其仍指向為

原始區塊的區塊1，但不會從中讀取任何資料，且此區塊本身已於稍早抹除。

另一種可能性是主機讀取邏輯群組中邏輯單元。此時，會從為邏輯群組之原始區塊的區塊3讀取區段A及B的邏輯單元，及從為群組之循序區塊的區塊4讀取區段C及D的邏輯單元。

步驟1136：或在電源開啟初始化中，藉由掃描其中內容以重新識別第一至第四區塊中的任一項。

中間階段的另一個可能性是關閉記憶體裝置的電源，然後重新啟動。如上述，在電源開啟初始化期間，會掃描配置區塊清單中的區塊(要使用的抹除集區區塊，見圖15及圖18)以識別邏輯群組中已成為特殊狀態原始區塊(區塊3)及關聯之循序更新區塊(區塊4)的缺陷彙總區塊。中斷區塊(區塊4)之第一邏輯單元中的旗標將指示關聯的區塊為已遭遇程式錯誤的原始區塊(區塊3)。藉由查閱區塊目錄(GAT)，即可尋找區塊3。

在一項具體實施例中，會將旗標程式化至中斷彙總區塊(區塊4)的第一邏輯單元。這可協助指示邏輯群組的特殊狀態：即，其已經彙總成兩個區塊，即，區塊3及區塊4。

使用旗標以識別具缺陷區塊之邏輯群組的一個替代方法是，利用不像原始區塊應為已滿的特性(除非錯誤發生在最後頁面，及最後頁面沒有ECC錯誤)偵測在掃描期間為缺陷的區塊。還有，根據實施例而定，其中會有有關儲存在快閃記憶體中控制資料結構之失敗群組/區塊的資訊記錄，而

不只是在寫入中斷彙總區塊(區塊4)之第一區段之標頭區中的旗標。

彙總完成(階段III)

步驟1142：為了回應預定的事件，對於自階段I後未進一步記錄第四區塊時的第一種情況，以和該第一順序相同的順序，在其中儲存該邏輯群組之所有未處理完成之邏輯單元的目前版本；對於自階段I後已進一步記錄第四區塊時的第二種情況，將第三及第四區塊彙總為第五區塊。

步驟1144：之後，對於第一種情況，操作記憶體時，以彙總的第四區塊作為邏輯群組的原始區塊；對於第二種情況，操作記憶體時，以第五區塊作為邏輯群組的原始區塊。

只要有任何不會違反任何指定時間限制的機會，即可執行階段III中的最後彙總。一個較佳情況是，在其中有另一個未附帶彙總操作之邏輯群組之更新操作時，「掛附(piggy-back)」在下一個主機寫入時槽上。如果另一個邏輯群組的主機寫入觸發本身的廢棄項目收集，則將使階段III彙總延後。

圖33顯示多階段彙總操作之第一及最後階段的範例時序。主機寫入等待時間是具有持續期間 T_w 之各主機寫入時槽的寬度。主機寫入1是簡單的更新，及邏輯群組 LG_1 中第一組邏輯單元的目前版本會被記錄在關聯的更新區塊上。

在主機寫入2，會在邏輯群組 LG_1 上發生更新，致使更新區塊被關閉(如，已滿)。會提供新的更新區塊以記錄其餘的更新。提供新的更新區塊會觸發廢棄項目收集，而導致關

於LG₄的彙總操作，以便再循環要再使用的區塊。LG₄群組的目前邏輯單元會按循序順序記錄在彙總區塊上。彙總操作可繼續進行直到在彙總區塊中遭遇缺陷為止。然後叫用階段I彙總，其中彙總操作在中斷彙總區塊上繼續。同時，LG₄(階段III)的最後彙總會等待下一個機會。

在主機寫入3，也會發生邏輯群組LG₂之邏輯單元的寫入以觸發LG₂的彙總。這表示已經完全利用時槽。

在主機寫入4，操作只是將LG₂的一些邏輯單元記錄至其更新區塊。時槽中剩餘的時間可提供執行LG₄之最後彙總的機會。

未將中斷彙總區塊轉換為更新區塊的具體實施例

圖34A及圖34B分別顯示圖28及圖31之範例適用之多階段彙總之階段I及階段III操作的第一案例。

圖34A顯示其中中斷彙總區塊並非用作更新區塊而是用作其彙總操作已經中斷之彙總區塊的例子。尤其，圖34A是指圖33所示的主機寫入#2，其中主機寫入屬於邏輯群組LG₁之邏輯單元的更新，及在此期間，此操作也會觸發和另一個邏輯群組LG₄關聯之區塊的彙總。

原始區塊(區塊1)及更新區塊(區塊2)的形成方式和圖28的範例相同。同樣地，在彙總操作期間，已知彙總區塊(區塊3)在彙總區段C的邏輯單元時有缺陷。然而，不像圖28所示的重新彙總方案，本多階段方案可在新提供之可當作中斷彙總區塊的區塊(區塊4)上繼續彙總操作。因此，在階段I彙總操作中，已在彙總區塊(區塊3)中彙總區段A及B中

的邏輯單元。當彙總區塊中發生程式失敗時，會循序將區段C及D中其餘的邏輯單元複製至中斷彙總區塊(區塊4)。

如果主機原來在第一邏輯群組中寫入更新會觸發和第二邏輯群組關聯之區塊的彙總操作，則會將第一邏輯群組的更新記錄至第一邏輯群組的更新區塊(通常為新的更新區塊)。此時，中斷彙總區塊(區塊4)不會用來記錄彙總操作之外的任何更新資料且會維持必須完成的中斷彙總區塊。

由於區塊1及區塊2中的資料現在完全含在另一個區塊(區塊3及區塊4)中，因此可將其抹除以便再循環。位址表(GAT)會被更新以指向區塊3，作為邏輯群組的原始區塊。更新區塊的目錄資訊(在ACL中，見圖15及圖18)也會被更新，以指向已成為邏輯群組(如，LG₄)之循序更新區塊的區塊4。

結果，彙總的邏輯群組並未侷限在一個區塊中，而是分布在缺陷彙總區塊(區塊3)及中斷彙總區塊(區塊4)之上。此方案的重要特色是群組中的邏輯單元只會在此階段期間彙總一次，但卻將彙總散布在一個以上區塊之上。依此方式，可在正常指定的時間內完成彙總操作。

圖34B顯示始於圖34A之多階段彙總的第三及最後階段。如結合圖33所述，會在第一階段後的適宜時間(例如在後續不會觸發隨附彙總操作的主機寫入期間)執行階段III彙總。尤其，圖34B是指其中發生如圖33所示之主機寫入#4的時槽。在該期間中，主機寫入可更新屬於邏輯群組LG₂的邏輯單元而不會觸發另一個額外的彙總操作。因此，有

利於將時槽中剩餘的時間用於階段III操作，以完成邏輯群組LG₄的彙總。

此操作可將還不在中斷區塊中之LG₄之所有未處理完成之邏輯單元彙總至中斷區塊。在此範例中，這表示會從區塊3按邏輯上循序順序將區段A及B複製至中斷區塊(區塊4)。由於區塊中邏輯單元的繞回方案及使用頁面標記(見圖3A)，即使範例顯示在區塊4中，區段A及B會被記錄在區段C及D之後，但仍會將已記錄的序列考慮為等同於A、B、C、D的循序順序。根據實施例而定，較佳從區塊3取得要複製之未處理完成之邏輯單元的目前版本，因其已經是彙總的形式，不過也可從尚未被抹除的區塊1及區塊2中收集。

在中斷區塊(區塊4)上完成最後彙總後，會將其指定為邏輯群組的原始區塊並跟著更新合適的目錄(如，GAT，見圖17A)。同樣地，會將失敗的實體區塊(區塊3)標示為不良並將其排除。其他區塊，區塊1及區塊2，將被抹除及再循環。同時，會將LG₂的更新記錄在和LG₂關聯的更新區塊中。

將中斷彙總區塊變成更新區塊的具體實施例

圖35A及圖35B分別顯示圖28及圖33之範例適用之多階段彙總之階段I及階段III操作的第二案例。

圖35A顯示其中維持中斷彙總區塊為接收主機寫入之更新區塊而非彙總區塊的例子。這適用於如更新邏輯群組LG₄的主機寫入，及在此程序中，也會觸發相同邏輯群組的彙總。

和圖34A的案例一樣，將區塊1與區塊2彙總於區塊3中會

繼續進行，直到在處理區段C時遇到程式失敗。然後會在中斷彙總區塊(區塊4)上繼續彙總。在中斷區塊(區塊4)中彙總未處理完成之邏輯單元(如，在區段C及D中)後，並不在階段III中等待完成其中的邏輯群組彙總，而是維持中斷區塊為更新區塊。此案例尤其適於其中主機寫入可更新邏輯群組及觸發相同邏輯群組之彙總的情況。在此範例中，這可將邏輯群組LG₄之主機更新的記錄記錄在中斷彙總區塊(區塊4)中，而非記錄至新的更新區塊。此更新區塊(先前為中斷彙總區塊(區塊4))可根據其中記錄的主機資料而為循序或變成混亂。在所示的範例中，區塊4已變成混亂，因區段C中邏輯單元的後續較新版本使得區塊4中的先前版本被淘汰。

在中間階段期間，會將區塊3視為LG₄的原始區塊，及區塊4會是關聯的更新區塊。

圖35B顯示始於第二例中圖35A之多階段彙總的第三及最後階段。如結合圖33所述，會在第一階段後的適宜時間執行階段III彙總，如在後續不會觸發隨附彙總操作的主機寫入期間。在該期間中，主機寫入可更新屬於邏輯群組的邏輯單元而不會觸發另一個額外的彙總操作。因此，有利於將時槽中剩餘的時間用於階段III操作，以完成邏輯群組LG₄的彙總。

然後從區塊3及區塊4將邏輯群組LG₄的廢棄項目收集至新的彙總區塊(區塊5)。然後將區塊3標示為不良，將區塊4再循環，及新的彙總區塊(區塊5)將變成邏輯群組LG₄之新

的原始區塊。其他區塊，區塊1及區塊2，也會被抹除及再循環。

階段性程式失敗處置的其他具體實施例

圖31A、31B、34A、34B、35A及35B中所述的範例適用於較佳的區塊管理系統，其中各實體區塊(中繼區塊)僅儲存屬於相同邏輯群組的邏輯單元。本發明同樣適用於其他其中並無邏輯群組對實體區塊對齊的區塊管理系統，如WO 03/027828及WO 00/49488中所揭露的區塊管理系統。在這些其他系統中實施階段性程式失敗處置方法的一些範例如圖36A、36B及36C所示。

圖36A顯示套用於主機寫入觸發關閉更新區塊且更新區塊為循序時之情況的階段性程式錯誤處置方法。此例中的關閉係藉由以下方式來完成：將原始區塊2之其餘的有效資料(B及C)複製至循序更新區塊3。在資料部分C程式化起點之程式失敗的例子中，會將部分C程式化至保留的區塊4。然後可將新的主機資料寫入新的更新區塊5(未顯示)。此方法的階段II及III和混亂區塊關閉的情況相同。

圖36B顯示在更新區塊之更新的例子中套用於(局部區塊系統)的階段性程式錯誤處置方法。在此例中，會將邏輯群組儲存在原始區塊1及其他的更新區塊中。彙總操作包括將原始區塊1及其他更新區塊2的資料複製至更新區塊(根據一些規則選定，圖中的區塊3)之一。和已經說明的主要情況不同處在於區塊3已經部分被寫入。

圖36C顯示處理廢棄項目收集操作的階段性程式錯誤，或

不支援映射至中繼區塊之邏輯群組之記憶體區塊管理系統中的清除。此種記憶體區塊管理(循環儲存)系統係說明於WO 03/027828 A1中。循環儲存系統明顯的特色是不對單一邏輯群組配置區塊。其中支援中繼區塊中控制資料的多個邏輯群組。廢棄項目收集涉及從部分淘汰的區塊，將可能沒有任何關係(隨機邏輯區塊位址)的有效資料區段取至其中可能已經有些資料的重新配置區塊。如果重新配置區塊在操作期間變滿，則將開啟另一個區塊。

非循序更新區塊索引

在上文關於混亂區塊索引及結合圖16A-16E的段落中，CBI區段可用來儲存可記錄隨機儲存在混亂或非循序更新區塊中之邏輯區段位置的索引。

根據本發明的另一方面，在具有支援具非循序邏輯單元之更新區塊之區塊管理系統的非揮發性記憶體中，緩衝儲存在RAM中之非循序更新區塊之邏輯單元的索引被定期儲存在非揮發性記憶體中。在一項具體實施例中，將索引儲存在專用於儲存索引的區塊中。在另一項具體實施例中，會將索引儲存在更新區塊中。在又另一項具體實施例中，將索引儲存在各邏輯單元的標頭中。在另一方面中，在上一個索引更新之後但在下一個索引更新之前寫入的邏輯單元會將其索引資訊儲存在各邏輯單元的標頭中。依此方式，在電源中斷後，不必在初始化期間執行掃描，即可決定最近寫入之邏輯單元的位置。在又另一方面中，將區塊管理成部分循序及部分非循序指向一個以上邏輯子群組。

在預定觸發事件後儲存在CBI區塊中CBI區段的索引指標

根據結合圖16A-16E所述的方案，會將混亂區塊中最近寫入區段的清單保留在控制器RAM中。只在和給定混亂區塊關聯之邏輯群組之預定數量的寫入後，含有最新索引資訊的CBI區段才被寫入快閃記憶體(CBI區塊620)。依此方式，可縮減CBI區塊更新的數量。

在邏輯群組中CBI區段的下一個更新之前，會將邏輯群組中最近寫入區段的清單保留在控制器RAM中。此清單在記憶體裝置遭遇電源關閉時將會遺失，但可在電源啟動後的初始化中藉由掃描更新區塊加以重建。

圖37顯示在每N個區段寫入相同的邏輯群組後將CBI區段寫入關聯之混亂索引區段區塊的排程範例。此範例顯示兩個同時進行更新的邏輯群組LG₃及LG₁₁。初始時，會將LG₃的邏輯區段按循序順序儲存在原始區塊中。群組中邏輯區段的更新會按照主機指定的順序被記錄在關聯的更新區塊上。此範例顯示混亂更新序列。同時，也會以和其更新區塊相同的方式更新邏輯群組LG₁₁。在每個邏輯區段寫入後，會將其在更新區塊中的位置保留在控制器RAM中。在每個預定的觸發事件後，會以混亂索引區段的形式將更新區塊中邏輯區段的目前索引寫入非揮發性混亂索引區段區塊。例如，預定的觸發事件可發生於每N個寫入之後，其中N為3。

雖然提供的範例有關作為區段的資料邏輯單元，但熟習本技術者應明白，邏輯單元也可以是一些其他集合體，如含有一個區段或一組區段的頁面。還有，循序區塊中的第

一頁面不一定是邏輯頁面0，因為繞回的頁面標記可能是第一頁面。

在預定觸發事件後儲存在混亂更新區塊中CBI區段的索引指標

在另一項具體實施例中，會在其中每N個寫入後，將索引指標儲存在混亂更新區塊本身中的專用CBI區段中。此方案和上述其中也將索引儲存在CBI區段中的具體實施例相同。其中的差異在於上述具體實施例中，CBI區段被記錄在CBI區段區塊中，而非更新區塊本身中。

此方法係基於將所有混亂區塊索引資訊保持在混亂更新區塊本身中。圖38A、38B及38C分別顯示同樣按照三個不同階段儲存CBI區段之更新區塊的狀態。

圖38A顯示直到在預定數量的寫入後在其中記錄CBI區段時的更新區塊。在此範例中，在主機已經循序寫入邏輯區段0-3後，接著將會發出再次寫入邏輯區段1之另一版本的指令，因而破壞資料寫入的循序序列。然後實施CBI區段中載送的混亂區塊索引，將更新區塊轉換為混亂更新區塊。如上述，CBI是含有混亂區塊之所有邏輯區段之索引的索引。例如，第0個項目代表第0個邏輯區段之更新區塊的位移，同樣地，第n個項目代表第n個邏輯區段的位移。可將CBI區段寫入更新區塊中的下一個可用位置。為了避免頻繁的快閃存取，會在每N個資料區段寫入後寫入CBI區段。在此範例中，N為4。如果此時損失電源，則最後寫入的區段成為CBI區段，並將此區塊視為混亂更新區塊。

圖 38B 顯示圖 38A 之進一步在索引區段後在其中記錄邏輯區段 1、2 及 4 的更新區塊。邏輯區段 1 及 2 的較新版本可取代先前在更新區塊中記錄的舊版本。在此時之電源週期的例子中，必須先找到最後寫入的區段，然後必須掃描多達 N 個區段，以尋找最後寫入的索引區段及最近寫入的資料區段。

圖 38C 顯示圖 38B 之具有另一寫入以觸發索引區段下一個記錄之邏輯區段的更新區塊。在另 N 個 ($N=4$) 區段寫入後的相同更新區塊可記錄 CBI 區段的另一個目前版本。

此方案的優點是不需要分開的 CBI 區塊。同時還不必擔心實體快閃區段的附加項資料區是否大到足以容納混亂更新區塊中有效區段之索引所需項目的數量。然後，混亂更新區塊含有所有的資訊，且位址轉譯不需要外部的資料。這可讓演算法比較簡單，其中可縮減有關 CBI 區塊壓縮的控制更新數；也可讓串接控制更新比較簡短。(請見上文有關 CBI 區塊管理的部分)。

關於儲存在混亂更新區塊中資料區段標頭之最近寫入區段的資訊

根據本發明的另一方面，會在每 N 個寫入後，將記錄在區塊中之邏輯單元的索引儲存在非揮發性記憶體中，及將有關中間寫入之邏輯單元的目前資訊儲存在各邏輯單元寫入的附加項部分。依此方式，在電源重新啟動後，不必掃描區塊，即可從區塊中最後寫入邏輯單元的附加項部分快速取得有關自上一個索引更新後寫入之邏輯單元的資訊。

圖 39A 顯示儲存於混亂更新區塊中各資料區段標頭之中

間寫入的中間索引。

圖39B顯示在寫入的各區段標頭中儲存中間寫入之中間索引的範例。在此範例中，在寫入四個區段 LS_0 - LS_3 後，會寫入CBI索引作為區塊中的下一個區段。其後，會將邏輯區段 LS'_1 、 LS'_2 及 LS_4 寫入區塊。每次，標頭都會儲存自上一個CBI索引後寫入之邏輯單元的中間索引。因此， LS'_2 中的標頭將具有提供上一個CBI索引及 LS'_1 之位移(即，位置)的索引。同樣地， LS_4 中的標頭將具有提供上一個CBI索引及 LS'_1 和 LS'_2 之位移(即，位置)的索引。

最後寫入的資料區段永遠含有有關多達N個最後寫入之頁面的資訊(即，一直到最後寫入之CBI區段)。只要電源重新啟動時，上一個CBI索引可提供邏輯單元在CBI索引區段之前寫入的索引資訊，及後續寫入之邏輯單元的索引資訊可在最後寫入之資料區段的標頭中找到。這具有優點如下：在初始化時不必為其後寫入的區段掃描區塊以決定其位置。

在資料區段的標頭中儲存中間索引資訊的方案同樣適用於無論CBI索引區段係儲存在更新區塊本身或在分開的CBI區段區塊中，如前文所述。

儲存在混亂更新區塊中資料區段標頭的索引指標

在另一項具體實施例中，會將整個CBI索引儲存在混亂更新區塊中各資料區段的附加項部分。

圖40顯示在混亂更新區塊之各資料區段標頭中儲存的混亂索引欄位中的資訊。

區段標頭的資訊容量有限，因此可將由任何單一區段所提

供的索引範圍設計為層級索引方案的一部分。例如，記憶體特定平面內的區段可提供索引給僅在該平面內的區段。還有，可將邏輯位址的範圍分割成一些子範圍，以允許採用間接索引方案。例如，如果可將有64個邏輯位址的區段儲存在一個平面中，則各區段可以有3個欄位用於區段位移值，各欄位能夠儲存4個位移值。第一欄位可定義邏輯位移範圍0-15、15-31、32-47、及48-63內最後寫入區段的實體位移。第二欄位可定義在其相關範圍內各4個區段之4個子範圍的實體位移值。第三欄位可定義在其相關子範圍內4個區段的實體位移值。因此，藉由讀取多達3個區段的間接位移值，即可決定混亂更新區塊內邏輯區段的實體位移。

此方案的優點是也不需要分開的CBI區塊或CBI區段。然而，這卻只適用於實體快閃區段的附加項資料區大到足以容納混亂更新區塊中有效區段索引所需的項目數量時。

混亂更新區塊之邏輯群組內受限的邏輯範圍

在邏輯群組內，可縮減可非循序寫入之區段的邏輯範圍。此技術的主要優點如下：由於只需要讀取一個多重區段頁面(在多重晶片的例子中，可以平行讀取頁面)，即可取得目的地頁面的所有資料(假設來源和目的地已對齊，若未對齊，則需要另一個讀取)，使得循序寫入資料的複製能夠更快速地完成，因此範圍之外的區段在原始區塊中保持循序寫入，及廢棄項目收集操作能在更短的時間內完成。還有，使用晶片上的複本特色，可將循序資料從來源複製至目的地，而不用在控制器之間來回傳送資料。如果來源資

料已經分散，如混亂區塊中所發生的，則需要每個區段讀取多達一個頁面，才能收集所有要寫入目的地的區段。

在一項具體實施例中，實際上並不將邏輯範圍限制在某個數量的區段，而是經由限制CBI的數量來完成(只限制大群組/中繼區塊的混亂範圍很合理，因其需要多個混亂區塊索引才能涵蓋整個邏輯群組的範圍)。例如，如果中繼區塊/群組有2048個區段，則其將需要多達8個CBI區段，各涵蓋一個子群組256個區段的連續邏輯範圍。如果將CBI的數量限制為4，則混亂區塊可用來寫入多達4個子群組的區段(其中任何一個)。因此，允許邏輯群組有多達4個部分或完全混亂子群組，且最少有4個子群組將維持完全循序。如果一個混亂區塊有4個和其關聯的有效CBI區段，及主機寫入在這些CBI區段範圍之外(混亂子群組)的區段，則應彙總及關閉混亂邏輯群組。但是這卻極不可能發生，因為在真實的應用中，主機在2048個區段的範圍(邏輯群組)內不需要多於4個256個區段的混亂範圍(子群組)。結果，在正常的情況中，廢棄項目收集也不會受到影響，但限制規則的防範卻形成廢棄項目收集太長(會觸發主機的逾時)的極端例子。

部分循序混亂更新區塊的索引

當循序更新區塊在將區塊轉換為混亂管理模式之前被部分寫入時，邏輯群組之循序更新區段的全部或部分可繼續被處理成已經循序更新，及可將混亂更新管理僅套用於邏輯群組之位址範圍的子集。

控制資料完整性與管理

儲存在記憶體裝置中的資料可能會因為電源中斷或特定記憶體位置變成有缺陷而成為已毀損。如果碰到記憶體區塊缺陷，則將資料重新配置至不同的區塊及將缺陷區塊丟棄。如果錯誤不會擴大，則可藉由和資料一起儲存的錯誤校正碼(ECC)在執行中進行校正。然而，還是會有ECC無法校正已毀損資料的時候。例如，當錯誤位元的數量超過ECC的容量時。這對於如和記憶體區塊管理系統關聯之控制資料的關鍵資料是無法接受的。

控制資料的範例為目錄資訊及和記憶體區塊管理系統關聯的區塊配置資訊，如結合圖20所述。如上述，可將控制資料維持在高速RAM及較慢的非揮發性記憶體區塊中。任何經常變更的控制資料會被維持在具有定期控制寫入的RAM中，以更新儲存於非揮發性中繼區塊中的同等資訊。依此方式，不必經常存取，即可將控制資料儲存在非揮發性但較慢的快閃記憶體中。如圖20所示GAT、CBI、MAP、及MAPA之控制資料結構的層級會被維持在快閃記憶體中。因此，控制寫入操作造成RAM中控制資料結構的資訊可更新快閃記憶體中同等的控制資料結構。

關鍵資料備份

根據本發明的另一方面，如部分或全部控制資料的關鍵資料如果被維持在複製項中，則保證額外等級的可靠性。複製的執行方式對於採用兩次編碼過程(two-pass)程式化技術以連續程式化相同組記憶體單元之多位元的多重狀態記憶體系統而言，第二次編碼過程中的任何程式化錯誤都

無法毀損第一次編碼過程建立的資料。複製還有助於偵測寫入中止、偵測誤測(即，兩個複本有良好的ECC但資料不同)，且可增加額外等級的可靠性。若干資料複製的技術均已考慮。

在一項具體實施例中，在稍早程式化編碼過程中程式化給定資料的兩個複本後，後續程式化編碼過程可避免程式化用於儲存該等兩個複本中至少一個的記憶體單元。依此方式，在後續程式化編碼過程在完成之前中止及毀損稍早編碼過程的資料時，該等兩個複本中至少一個也不會受到影響。

在另一項具體實施例中，某一給定資料的兩個複本會被儲存於兩個不同的區塊中，而且該等兩個複本中至多僅有其中一個的記憶體單元會於後面的程式化編碼過程中被程式化。

於另一具體實施例中，於一程式化編碼過程中儲存某一給定資料的兩個複本之後，便不再對用於儲存該等兩個複本的記憶體單元組實施任何進一步的程式化。於該記憶體單元組的最終程式化編碼過程中來程式化該等兩個複本便可達成此目的。

在又另一項具體實施例中，可於二進制程式化模式中將某一給定資料的該等兩個複本程式化至一多重狀態的記憶體之中，致使不會對該等已程式化的記憶體單元進行任何進一步的程式化。

在又另一項具體實施例中，對於採用兩次編碼過程程式

化技術以連續程式化相同組記憶體單元之多位元的多重狀態記憶體系統而言，會採用容錯碼以編碼多個記憶體狀態，使稍早程式化編碼過程所建立的資料不會受到後續程式化編碼過程中錯誤的影響。

在各記憶體單元可儲存一位元以上資料的多重狀態記憶體中會引發資料複製的複雜性。例如，一個4狀態記憶體可以兩個位元來表示。一個現有的技術是使用2次編碼過程程式化來程式化此種記憶體。第一位元(下方頁面位元)可由第一次編碼過程進行程式化。其後，可在第二次編碼過程中程式化相同單元以代表所要的第二位元(上方頁面位元)。為了不要變更第二次編碼過程中第一位元的值，會使第一位元的記憶體狀態表示取決於第二位元的值。因此，在第二位元的程式化期間，如果因電源中斷或其他原因而發生錯誤及造成不正確的記憶體狀態，則也會毀損第一位元的值。

圖41A顯示當各記憶體單元儲存兩個位元的資料時，4狀態記憶體陣列的定限電壓分布。此四個分布代表四個記憶體狀態「U」、「X」、「Y」及「Z」的總體。在程式化記憶體單元之前，會先將其抹除至其「U」或「未寫入」狀態。在記憶體單元逐漸被程式化時，會累進達到記憶體狀態「X」、「Y」及「Z」。

圖41B顯示現有使用格雷碼(Gray code)的2次編碼過程程式化方案。此四個狀態可以兩個位元表示，即，下方頁面位元及上方頁面位元，如(上方頁面位元,下方頁面位元)。對於要平行程式化之單元的頁面，實際上有兩個邏輯頁

面：邏輯下方頁面及邏輯上方頁面。第一程式化編碼過程只會程式化邏輯下方頁面。藉由合適的編碼，不用重設邏輯下方頁面，單元相同頁面上的後續第二程式化編碼過程會程式化邏輯上方頁面。一般使用的程式碼是格雷碼，其中只有一個位元會在轉變至相鄰的狀態時變更。因此，此程式碼具有優點如下：對於錯誤校正的要求較少，因只涉及一個位元。

一般使用格雷碼的方案是假設「1」代表「未程式化」條件。因此，已抹除的記憶體狀態「U」可表示為：(上方頁面位元，下方頁面位元) = (1, 1)。在程式化邏輯下方頁面的第一次編碼過程中，任何儲存資料「0」的單元將因此具有其從(x, 1)至(x, 0)的邏輯狀態轉變，其中「x」代表上方位元的「任意(don't care)」值。然而，由於上方位元尚未被程式化，因此為了一致，可將「x」標示為「1」。(1, 0)邏輯狀態可藉由程式化單元為記憶體狀態「X」來表示。也就是說，在第二次程式編碼過程之前，下方位元值「0」可以表示為記憶體狀態「X」。

執行第二次編碼過程程式化可儲存邏輯上方頁面的位元。只有這些需要上方頁面位元值「0」的單元才會被程式化。在第一次編碼過程後，頁面中的單元在邏輯狀態(1, 1)或(1, 0)。為了保存第二次編碼過程中下方頁面的值，必須區分下方位元值「0」或「1」。對於從(1, 0)至(0, 0)的轉變，會將所論記憶體單元程式化為記憶體狀態「Y」。對於從(1, 1)至(0, 1)的轉變，會將所論記憶體單元程式化為記憶體狀

態「Z」。依此方式，在讀取期間，藉由決定在單元中程式化的記憶體狀態，即可解碼下方頁面位元及上方頁面位元。

然而，格雷碼的2次編碼過程程式化方案在第二次編碼過程程式化錯誤時會成為問題。例如，在下方位元為「1」時將上方頁面位元程式化為「0」，將造成(1, 1)轉變成(0, 1)。這需要將記憶體單元累進從「U」程式化通過「X」及「Y」而至「Z」。如果在完成程式化前發生電源中斷，則記憶體單元將結束於轉變記憶體狀態之一，如「X」。在讀取記憶體單元時，會將「X」解碼為邏輯狀態(1, 0)。這對上方及下方位元造成不正確的結果，因其應為(0, 1)。同樣地，如果程式化在達到「Y」時受到中斷，其將對應於(0, 0)。雖然上方位元現在是正確的，但下方位元還是錯的。

因此，可以看出上方頁面程式化的問題可毀損已經在下方頁面的資料。尤其當第二次編碼過程程式化涉及在中間記憶體狀態上通過時，程式中止會使程式化結束於該記憶體狀態，造成解碼不正確的下方頁面位元。

圖42顯示藉由儲存複製的各區段以防衛關鍵資料的方式。例如，可將區段A、B、C、及D儲存在複製複本中。如一個區段複本中有資料毀損，則可以讀取另一個來取代。

圖43顯示其中通常將複製區段儲存在多重狀態記憶體的非健全性。如上述，在範例的4狀態記憶體中，多重狀態頁面實際上包括分別在兩次編碼過程中進行程式化的邏輯下方頁面及邏輯上方頁面。在所示的範例中，頁面為四個區段寬。因此，區段A及其複製會同時被程式化在邏輯下方頁

面中，同樣地，對於區段B及其複製也是如此。然後在邏輯上方頁面中後續之程式化的第二次編碼過程中，會同時程式化區段C, C，及對於區段D, D也是如此。如果在程式化區段C, C的中間發生程式中止，則會毀損下方頁面中的區段A, A。除非，在上方頁面程式化之前先讀取及緩衝下方頁面區段，否則如果毀損將無法復原。因此，同時儲存兩個關鍵資料的複本，如區段A, A，無法防止其為其上方頁面中後續區段C, C之有問題的儲存所毀損。

圖44A顯示將關鍵資料錯開之複製複本儲存至多重狀態記憶體的一項具體實施例。基本上，會以和圖43的相同方式儲存下方頁面，即，區段A, A及區段B, B。然而，在上方頁面程式化中，區段C及D會和其複製交錯成C, D, C, D。如果支援局部頁面程式化，則可同時程式化兩個區段C的複本，及對於兩個區段D的複本也是如此。如果兩個區段C的程式遭到中止，則只會在區段A的一個複本及區段B的一個複本上毀損下方頁面。另一個複本將維持不受影響。因此，如果儲存在第一次編碼過程中的關鍵資料有兩個複本，則其不將不會受到後續第二次編碼過程同時程式化的影響。

圖44B顯示只將關鍵資料之複製複本儲存至多重狀態記憶體之邏輯上方頁面的另一項具體實施例。此時，未使用下方頁面的資料。關鍵資料及其複製，如區段A, A及區段B, B只會被儲存至邏輯上方頁面。依此方式，如果有程式中止，則可將關鍵資料再寫入另一個邏輯上方頁面，而下方頁面資料的任何毀損將無關緊要。此辦法基本上使用各

多重狀態頁面一半的儲存容量。

圖44C顯示又另一項以多重狀態記憶體的二進制模式儲存關鍵資料之複製複本的具體實施例。此時，會依二進制模式程式化各記憶體單元，其中僅將其定限範圍分成兩個區域。因此，其中只有一次編碼過程程式化，且在發生程式中止時可在不同位置中重新啟動程式化。此辦法也使用各多重狀態頁面一半的儲存容量。依二進制模式操作多重狀態記憶體係說明於美國專利第6,456,528 B1號，其整個揭露內容在此以提及的方式併入本文中。

圖45顯示同時將關鍵資料之複製複本儲存至兩個不同中繼區塊的又另一項具體實施例。如果區塊之一變成不可用，則可從另一個區塊讀取資料。例如，關鍵資料係含在區段A、B、C、D及E、F、G、H及I、J、K、L內。各區段會被儲存在複製中。這兩個複本將被同時寫入兩個不同的區塊，區塊0及區塊1。如果將一個複本寫入邏輯下方頁面，則會將另一個複本寫入邏輯上方頁面。依此方式，永遠會有程式化至邏輯上方頁面的複本。如果發生程式中止，則可將其重新程式化至另一個邏輯上方頁面。同時，如果下方頁面已經毀損，則在其他區塊中永遠會有另一個上方頁面複本。

圖46B顯示使用容錯碼同時儲存關鍵資料之複製複本的又另一項具體實施例。圖46A和圖41A同在顯示4狀態記憶體陣列的定限電壓分布並顯示作為圖46B的參考。容錯碼實質上可避免在任何中間狀態中轉變的任何上方頁面程式

化。因此，在下方頁面程式化的第一次編碼過程中，邏輯狀態(1, 1)轉變為(1, 0)，如表示為程式化已抹除的記憶體狀態「U」為「Y」。在上方頁面位元為「0」的第二次編碼過程程式化中，如果下方頁面位元為「1」，則邏輯狀態(1, 1)轉變為(0, 1)，如表示為程式化已抹除的記憶體狀態「U」為「X」。如果下方頁面位元為「0」，則邏輯狀態(1, 0)轉變為(0, 0)，如表示為程式化記憶體狀態「Y」為「Z」。由於上方頁面程式化僅涉及程式化為下一個相鄰的記憶體狀態，因此程式中止無法變更下方頁面位元。

串列寫入

關鍵資料的複製複本較佳如上述同時寫入。另一個避免同時毀損兩個複本的方式是循序寫入複本。此方法較慢，但複本本身代表其在控制器檢查兩個複本時程式化是否成功。

圖 47 為顯示兩個資料複本之可能狀態及資料有效性的表格。

如果第一及第二複本沒有 ECC 錯誤，則可將資料的程式化視為完全成功。有效資料可從任一個複本取得。

如果第一複本沒有 ECC 錯誤，但第二複本有 ECC 錯誤，便表示程式化在第二複本程式化的中間受到中斷。第一複本含有有效資料。即使錯誤為可校正，第二複本資料已不可靠。

如果第一複本沒有 ECC 錯誤且第二複本已經清空(抹除)，便表示程式化在第一複本程式化結束後但在第二複本

開始前受到中斷。第一複本含有有效資料。

如果第一複本有ECC錯誤及第二複本已經清空(抹除)，便表示程式化在第一複本程式化的中間受到中斷。即使錯誤為可校正，第一複本仍含有無效資料。

為了讀取維持在複製中的資料，以下技術為較佳，因其利用複製複本的存在。讀取及比較兩個複本。此例中，圖47所示兩個複本的狀態可用來確保沒有任何錯誤誤測。

在另一項具體實施例中，控制器只讀取一個複本，為了顧及速度及簡單性，複本讀取較佳在兩個複本之間輪替。例如，在控制器讀取控制資料時，其可讀取如複本1，下一個控制讀取(任何控制讀取)則應來自複本2，然後再是複本1等。依此方式，即可讀取及定期檢查兩個複本的完整性(ECC檢查)。其可減少以下風險：無法在因變質的資料保留所造成的時間錯誤中進行偵測。例如，如果通常只讀取複本1，則複本2會逐漸變質至其中錯誤無法為ECC挽救的程度，因而無法再使用第二複本。

先估式資料重新配置

如結合圖20所述，區塊管理系統在其操作期間可在快閃記憶體中維持一組控制資料。此組控制資料會被儲存在和主機資料相同的中繼區塊中。因此，控制資料本身會受到區塊管理，因而會受到更新的影響，及因此也受到廢棄項目收集操作的影響。

其中還說明控制資料的階層，其中較低層級中的控制資料更新比較高層級中的頻繁。例如，假設每個控制區塊有N

個要寫入的控制區段，則通常會發生以下控制更新及控制區塊重新配置的序列。再次參考圖 20，每 N 個 CBI 更新可填滿 CBI 區塊及觸發 CBI 重新配置(再寫入)及 MAP 更新。如果混亂區塊遭受關閉，則其也會觸發 GAT 更新。每個 GAT 更新可觸發 MAP 更新。每 N 個 GAT 更新可填滿區塊及觸發 GAT 區塊重新配置。此外，當 MAP 區塊變滿時，也會觸發 MAP 區塊重新配置及 MAPA 區塊(如果存在的話，否則 BOOT 區塊會直接指向 MAP)更新。此外，當 MAPA 區塊變滿時，也會觸發 MAPA 區塊重新配置、BOOT 區塊更新及 MAP 更新。此外，在 BOOT 區塊變滿時，將會觸發另一個 BOOT 區塊之作用中 BOOT 區塊重新配置。

由於階層的形成為：頂部的 BOOT 控制資料，接著是 MAPA、MAP、然後 GAT，因此，在每 N^3 個 GAT 更新中，將有「串接控制更新」，其中所有的 GAT、MAP、MAPA 及 BOOT 區塊都會被重新配置。此例中，在因主機寫入導致的混亂或循序更新區塊關閉造成 GAT 更新時，也會有廢棄項目收集操作(即，重新配置或再寫入)。在混亂更新區塊廢棄項目收集的例子中，會更新 CBI，而這也會觸發 CBI 區塊重新配置。因此，在此極端的情況中，必須同時收集大量中繼區塊的廢棄項目。

從圖中可見，階層的各控制資料區塊在取得填充及接受重新配置上有其自己的週期性。如果各控制資料區塊進行正常，則將有發生以下情形的時候：大量區塊的階段進行整頓，因而觸發大量同時涉及所有這些區塊的重新配置廢

棄項目收集。許多控制區塊的重新配置將會花費很長的時間，因此應加以避免，因為部分主機不容許因大量控制操作所造成的長時間延遲。

根據本發明的另一方面，在具有區塊管理系統的非揮發性記憶體中，可實施記憶體區塊的「控制廢棄項目收集」或先佔式重新配置，以避免發生大量的更新區塊均恰巧同時需要進行重新配置的情形。例如，在更新用於控制區塊管理系統操作的控制資料時會發生此情況。控制資料類型的層級可和不同程度的更新次數共存，導致其關聯的更新區塊需要不同速率的廢棄項目收集或重新配置。會有一個以上控制資料類型之廢棄項目收集操作同時發生的特定次數。在極端的情況中，所有控制資料類型之更新區塊的重新配置階段會進行整頓，導致所有的更新區塊都需要同時重新配置。

本發明可避免這種不想要的情況，其中目前的記憶體操作無論何時均可容納自發性的廢棄項目收集操作，更新區塊的先佔式重新配置可預先在完全填充區塊前發生。尤其，會將優先權提供給具有最慢速率之最高階層資料類型的區塊。依此方式，在重新配置最慢速率區塊後，將不再需要另一個相對較長時間的廢棄項目收集。還有，階層中較高的較慢速率區塊沒有太多可觸發之重新配置的串接。可將本發明方法視為：為了避免所論各種區塊之階段對齊，而將某種抖動引入事物的整體混合。因此，只要一有機會，即可以先佔式的方式重新配置有些微不受完全填充

之邊限的緩慢填充區塊。

在具有階層中較低的控制資料因串接效應而變更快於階層中較高的控制資料之控制資料階層的系統中，會將優先權提供給階層中較高之控制資料的區塊。一個執行自發性先佔式重新配置之機會的範例是在以下情況時：當主機寫入本身無法觸發重新配置，因此可利用其等待時間中的任何剩餘時間來進行先佔式重新配置操作。一般而言，務必重新配置之區塊前的邊限是在區塊全滿前之預定數量的未寫入記憶體單元。所考慮的是足以加速在完全填充之區塊前但又不會過早之重新配置的邊限，以免資源浪費。在較佳具體實施例中，預定數量的未寫入記憶體單元係在一到六個記憶體單元之間。

圖48顯示先佔式重新配置儲存控制資料之記憶體區塊的流程圖。

步驟1202：將非揮發性記憶體組織成區塊，各區塊已分割成可一起抹除的記憶體單元。

步驟1204：維持不同類型的資料。

步驟1206：對不同類型的資料指派階層。

步驟1208：儲存複數個區塊中該等不同類型資料的更新，使得各區塊實質上儲存相同類型的資料。

步驟1210：為了回應具有少於預定數量之清空記憶體單元及具有該等複數個區塊中最高階層資料類型的區塊，將該區塊之資料的目前更新重新配置至另一個區塊。若未受到中斷，到步驟1208。

實施圖 20 所示控制資料之先佔式重新配置的範例演算法如下：

如果((沒有任何因使用者資料所造成的廢棄項目收集)或(MAP留有6個或更少的未寫入區段)或(GAT留有3個或更少的未寫入區段))

則

如果(BOOT留有1個未寫入區段)

則重新配置BOOT(即，重新配置至區塊)

否則

如果(MAPA留有1個未寫入區段)

則重新配置MAPA及更新MAP

否則

如果(MAP留有1個未寫入區段)

則重新配置MAP

否則

如果(上一個更新或最大的GAT留有1個未寫入區段)

則重新配置GAT

否則

如果(CBI留有1個未寫入區段)

則重新配置CBI

否則

否則

離開

因此，先佔式重新配置通常完成於其中未發生任何使用

者資料廢棄項目收集時。在最糟的情況中，當每個主機寫入觸發使用者資料廢棄項目收集，但還有足夠的時間進行一個區塊的自發性重新配置時，一次可執行一個控制區塊的先佔式重新配置。

由於使用者資料廢棄項目收集操作及控制更新可能和實體錯誤同時發生，因此，最好具有較大的安全性邊限，其係藉由事先如在區塊還有2個或更多未寫入之記憶體單元(如，區段)時，先行進行先佔式重新配置或控制之廢棄項目收集。

雖然已經針對特定具體實施例說明本發明的各種方面，但應明白，本發明有權受到隨附申請專利範圍之完整範疇的保護。

【圖式簡單說明】

圖1以示意圖顯示適於實施本發明之記憶體系統的主要硬體組件。

圖2顯示根據本發明的一項較佳具體實施例，經組織成區段(或中繼區塊)之實體群組並為控制器之記憶體管理器所管理的記憶體。

圖3A(i)-3A(iii)根據本發明的一項較佳具體實施例，以示意圖顯示邏輯群組及中繼區塊間的映射。

圖3B以示意圖顯示邏輯群組及中繼區塊間的映射。

圖4顯示中繼區塊和實體記憶體中結構的對齊。

圖5A顯示從連結不同平面之最小抹除單元所構成的中繼區塊。

圖 5B 顯示其中從各平面選擇最小抹除單元 (MEU) 以連結至中繼區塊的一項具體實施例。

圖 5C 顯示其中從各平面選擇一個以上 MEU 以連結至中繼區塊的另一項具體實施例。

圖 6 為如控制器及快閃記憶體中實施之中繼區塊管理系統的示意方塊圖。

圖 7A 顯示邏輯群組中按循序順序寫入循序更新區塊之區段的範例。

圖 7B 顯示邏輯群組中按混亂順序寫入混亂更新區塊之區段的範例。

圖 8 顯示由於兩個在邏輯位址有中斷之分開的主機寫入操作而在邏輯群組中按循序順序寫入循序更新區塊之區段的範例。

圖 9 根據本發明的一般具體實施例，為顯示更新區塊管理器更新一個邏輯群組之資料的程序流程圖。

圖 10 顯示根據本發明的一項較佳具體實施例，更新區塊管理器更新一個邏輯群組之資料的程序流程圖。

圖 11A 為詳細顯示關閉圖 10 所示之混亂更新區塊之彙總程序的流程圖。

圖 11B 為詳細顯示關閉圖 10 所示之混亂更新區塊之壓縮程序的流程圖。

圖 12A 顯示邏輯群組的所有可能狀態，及其間在各種操作下的可能轉變。

圖 12B 為列出邏輯群組之可能狀態的表格。

圖 13A 顯示中繼區塊的所有可能狀態，及其間在各種操作下的可能轉變。中繼區塊是對應於邏輯群組的實體群組。

圖 13B 為列出中繼區塊之可能狀態的表格。

圖 14(A)-14(J) 為顯示邏輯群組狀態上及實體中繼區塊上各種操作效果的狀態圖。

圖 15 顯示用於記錄開啟及關閉之更新區塊及配置之已抹除區塊之配置區塊清單 (ABL) 結構的較佳具體實施例。

圖 16A 顯示混亂區塊索引 (CBI) 區段的資料欄位。

圖 16B 顯示記錄於專用中繼區塊中之混亂區塊索引 (CBI) 區段的範例。

圖 16C 為顯示存取進行混亂更新之給定邏輯群組之邏輯區段之資料的流程圖。

圖 16D 根據其中已將邏輯群組分割成子群組的替代性具體實施例，為顯示存取進行混亂更新之給定邏輯群組之邏輯區段之資料的流程圖。

圖 16E 顯示在其中將各邏輯群組分割成多個子群組的具體實施例中，混亂區塊索引 (CBI) 區段及其功能的範例。

圖 17A 顯示群組位址表 (GAT) 區段的資料欄位。

圖 17B 顯示記錄在 GAT 區塊中之群組位址表 (GAT) 區段的範例。

圖 18 為顯示使用及再循環已抹除區塊之控制及目錄資訊之分布及流程的示意方塊圖。

圖 19 為顯示實體位址轉譯邏輯程序的流程圖。

圖 20 顯示在記憶體管理的操作過程中，在控制資料結構

上執行的操作層級。

圖 21 顯示以多個記憶體平面所構成的記憶體陣列。

圖 22A 顯示根據本發明的一般實施例，具有平面對齊之更新之方法的流程圖。

圖 22B 顯示在圖 22A 所示之流程圖中儲存更新之步驟的較佳具體實施例。

圖 23A 顯示不顧平面對齊按循序順序寫入循序更新區塊之邏輯單元的範例。

圖 23B 顯示不顧平面對齊按非循序順序寫入混亂更新區塊之邏輯單元的範例。

圖 24A 顯示根據本發明的一項較佳具體實施例，具有平面對齊及填補之圖 23A 的循序更新範例。

圖 24B 顯示根據本發明的一項較佳具體實施例，具有平面對齊及不具有任何填補之圖 23B 的混亂更新範例。

圖 24C 顯示根據本發明的另一項較佳具體實施例，具有平面對齊及填補之圖 23B 的混亂更新範例。

圖 25 顯示其中各頁面含有兩個用於儲存兩個邏輯單元(如兩個邏輯區段)之記憶體單元的範例記憶體組織。

圖 26A 和圖 21 的記憶體結構相同，只是各頁面含有兩個區段而非一個。

圖 26B 顯示圖 26A 所示之具有以線性圖方式布局之記憶體單元的中繼區塊。

圖 27 顯示的替代性方案如下：不用填補要從一個位置複製到另一個的邏輯單元，即可在更新區塊中進行平面對齊。

圖28顯示其中缺陷區塊在彙總操作期間發生程式失敗時會在另一個區塊上重複彙總操作的方案。

圖29以示意圖顯示具有允許足夠時間完成寫入(更新)操作及彙總操作之時序或寫入等待時間的主機寫入操作。

圖30根據本發明的一般方案，顯示程式失敗處置的流程圖。

圖31A顯示程式失敗處置的一項具體實施例，其中第三(最後的重新配置)區塊和第二(中斷)區塊不同。

圖31B顯示程式失敗處置的另一項具體實施例，其中第三(最後的重新配置)區塊和第二(中斷)區塊相同。

圖32A顯示造成彙總操作之初始更新操作的流程圖。

圖32B顯示根據本發明的一項較佳具體實施例，多階段彙總操作的流程圖。

圖33顯示多階段彙總操作之第一及最後階段的範例時序。

圖34A顯示其中中斷彙總區塊並非用作更新區塊而是用作其彙總操作已經中斷之彙總區塊的例子。

圖34B顯示始於圖34A之多階段彙總的第三及最後階段。

圖35A顯示其中維持中斷彙總區塊為接收主機寫入之更新區塊而非彙總區塊的例子。

圖35B顯示始於第二例中圖35A之多階段彙總的第三及最後階段。

圖36A顯示套用於主機寫入觸發關閉更新區塊且更新區塊為循序時之情況的階段性程式錯誤處置方法。

圖 36B 顯示在更新區塊之更新的例子中套用於(局部區塊系統)的階段性程式錯誤處置方法。

圖 36C 顯示處理廢棄項目收集操作的階段性程式錯誤，或不支援映射至中繼區塊之邏輯群組之記憶體區塊管理系統中的清除。

圖 37 顯示在每 N 個區段寫入相同的邏輯群組後將 CBI 區段寫入關聯之混亂索引區段區塊的排程範例。

圖 38A 顯示直到在預定數量的寫入後在其中記錄 CBI 區段時的更新區塊。

圖 38B 顯示圖 38A 之進一步在索引區段後在其中記錄資料頁面 1、2 及 4 的更新區塊。

圖 38C 顯示圖 38B 之具有另一寫入以觸發索引區段下一個記錄之邏輯區段的更新區塊。

圖 39A 顯示儲存於混亂更新區塊中各資料區段標頭之中間寫入的中間索引。

圖 39B 顯示在寫入的各區段標頭中儲存中間寫入之中間索引的範例。

圖 40 顯示在混亂更新區塊之各資料區段標頭中儲存的混亂索引欄位中的資訊。

圖 41A 顯示當各記憶體單元儲存兩個位元的資料時，4 狀態記憶體陣列的定限電壓分布。

圖 41B 顯示現有使用格雷碼(Gray code)的 2 次編碼過程程式化方案。

圖 42 顯示藉由儲存複製的各區段以防衛關鍵資料的方

式。例如，可將區段A、B、C、及D儲存在複製複本中。如
在一個區段複本中有資料毀損，則可以讀取另一個來取代。

圖43顯示其中通常將複製區段儲存在多重狀態記憶體的非健全性。

圖44A顯示將關鍵資料錯開之複製複本儲存至多重狀態
記憶體的一項具體實施例。

圖44B顯示只將關鍵資料之複製複本儲存至多重狀態記
憶體之邏輯上方頁面的另一項具體實施例。

圖44C顯示又另一項以多重狀態記憶體的二進制模式儲
存關鍵資料之複製複本的具體實施例。

圖45顯示同時將關鍵資料之複製複本儲存至兩個不同中
繼區塊的又另一項具體實施例。

圖46A和圖41A同在顯示4狀態記憶體陣列的定限電壓分
布並顯示作為圖46B的參考。

圖46B顯示使用容錯碼同時儲存關鍵資料之複製複本的
又另一項具體實施例。

圖47為顯示兩個資料複本之可能狀態及資料有效性的表
格。

圖48顯示先佔式重新配置儲存控制資料之記憶體區塊的
流程圖。

【主要元件符號說明】

- | | |
|---|--------|
| 1 | 缺陷區塊 |
| 2 | 中斷區塊 |
| 3 | 重新配置區塊 |

10	主機
20	記憶體系統
100	控制器
110	介面
120	處理器
121	選用性副處理器
122	唯讀記憶體 (ROM)
124	選用性可程式非揮發性記憶體
130	隨機存取記憶體 (RAM)
132	快取記憶體
134、610	配置區塊清單 (ABL)
136、740	清除區塊清單 (CBL)
140	邏輯對實體位址轉譯模組
150	更新區塊管理器模組
152	循序更新
154	混亂更新
160	抹除區塊管理器模組
162	結束區塊管理器
170	中繼區塊連結管理器
180	控制資料互換
200	記憶體
210	群組位址表 (GAT)
220	混亂區塊索引 (CBI)
230、770	已抹除的區塊清單 (EBL)

240	MAP
612	已抹除的 ABL 區塊清單
614	開啟的更新區塊清單
615	關聯的原始區塊清單
616	關閉的更新區塊清單
617	已抹除的原始區塊清單
620	CBI 區塊
700	邏輯群組
702	原始中繼區塊
704	混亂更新區塊
750	MAP 區塊
760	抹除區塊管理 (EBM) 區段
772	可用的區塊緩衝器 (ABB)
774	已抹除的區塊緩衝器 (EBB)
776	已清除的區塊緩衝器 (CBB)
780	MAP 區段
782	來源 MAP 區段
784	目的地 MAP 區段
910	平面
912	讀取及程式電路
914	頁面
920	控制器
922	緩衝器
930	資料匯流排

五、中文發明摘要：

在具有支援具非循序邏輯單元之更新區塊之區塊管理系統的非揮發性記憶體中，非循序更新區塊中邏輯單元的索引被緩衝儲存在RAM中，並定期將其儲存至非揮發性記憶體中。在一項具體實施例中，將索引儲存在專用於儲存索引的區塊中。在另一項具體實施例中，將索引儲存在更新區塊本身中。在又另一項具體實施例中，將索引儲存在各邏輯單元的標頭中。在另一方面中，在上一個索引更新之後但在下一個索引更新之前寫入的邏輯單元會將其索引資訊儲存在各邏輯單元的標頭中。依此方式，在電源中斷後，不必在初始化期間執行掃描，即可決定最近寫入之邏輯單元的位置。在又另一方面中，將區塊管理成部分循序及部分非循序指向一個以上邏輯子群組。

六、英文發明摘要：

十、申請專利範圍：

1. 一種在一組織成區塊的非揮發性記憶體中用於儲存及更新資料的方法，其中各區塊已分割成可一起抹除的記憶體單元，各記憶體單元用於儲存一邏輯單元的資料，該方法包含：

將資料組織成複數個邏輯群組，各邏輯群組係為一群組邏輯的單元；

接收封裝在邏輯單元中的主機資料；

根據一第一順序，在一第一區塊中儲存一邏輯群組之邏輯單元的一第一版本；及

根據和該第一順序不同的一第二順序，在一包括該等邏輯單元之後續版本的第二區塊中儲存邏輯單元的最新版本；及

為了回應預定的觸發事件，在一第三區塊中儲存自一最新觸發事件後儲存於該第二區塊中之邏輯單元的一目錄。

2. 如請求項1之方法，進一步包含：

提供用於各邏輯單元的一標頭部分；及

對於自一最新之預定觸發事件後儲存於該第二區塊中的中間邏輯單元，在各該等中間邏輯單元的該標頭部分中儲存該等中間邏輯單元的一目錄。

3. 如請求項1之方法，其中該非揮發性記憶體具有浮動閘極記憶體單元。

4. 如請求項1之方法，其中該非揮發性記憶體係為快閃

EEPROM。

5. 如請求項1之方法，其中該非揮發性記憶體係為NROM。
6. 如請求項1之方法，其中該非揮發性記憶體係在一記憶卡中。
7. 如請求項1至6中任一項之方法，其中該非揮發性記憶體具有各儲存一位元之資料的記憶體單元。
8. 如請求項1至6中任一項之方法，其中該非揮發性記憶體具有各儲存一位元以上之資料的記憶體單元。
9. 一種在一組織成區塊的非揮發性記憶體中用於儲存及更新資料的方法，其中各區塊已分割成可一起抹除的記憶體單元，各記憶體單元用於儲存一邏輯單元的資料，該方法包含：

將資料組織成複數個邏輯群組，各邏輯群組係為一群組邏輯的單元；

接收封裝在邏輯單元中的主機資料；

根據一第一順序，在一第一區塊中儲存邏輯單元的第一版本；及

根據和該第一順序不同的一第二順序，在一包括該等邏輯單元之後續版本的第二區塊中儲存邏輯單元的最新版本；及

為了回應一預定的觸發事件，在該第二區塊的一可用位置中儲存自一最新觸發事件後儲存於該第二區塊之邏輯單元的一目錄。

10. 如請求項9之方法，進一步包含：

提供用於各邏輯單元的一標頭部分；及

對於自一最新之預定觸發事件後儲存於該第二區塊中的中間邏輯單元，在各該等中間邏輯單元的該標頭部分中儲存該等中間邏輯單元的一目錄。

11. 如請求項9之方法，其中該非揮發性記憶體具有浮動閘極記憶體單元。
12. 如請求項9之方法，其中該非揮發性記憶體係為快閃EEPROM。
13. 如請求項9之方法，其中該非揮發性記憶體係為NROM。
14. 如請求項9之方法，其中該非揮發性記憶體係在一記憶卡中。
15. 如請求項9至14中任一項之方法，其中該非揮發性記憶體具有各儲存一位元之資料的記憶體單元。
16. 如請求項9至14中任一項之方法，其中該非揮發性記憶體具有各儲存一位元以上之資料的記憶體單元。
17. 一種在一組織成區塊的非揮發性記憶體中用於儲存及更新資料的方法，其中各區塊已分割成可一起抹除的記憶體單元，各記憶體單元用於儲存一邏輯單元的資料，該方法包含：

在一第一區塊中儲存邏輯單元；

為了回應一預定的觸發事件，在該非揮發性記憶體的一部分中儲存至此在該第一區塊中之邏輯單元的一目錄；

提供用於各邏輯單元的一標頭部分；及

對於自一最新之預定觸發事件後儲存於該區塊中的中

間邏輯單元，在各該等中間邏輯單元的該標頭部分中儲存該等中間邏輯單元的一目錄。

18. 如請求項17之方法，其中該非揮發性記憶體具有浮動閘極記憶體單元。

19. 如請求項17之方法，其中該非揮發性記憶體係為快閃EEPROM。

20. 如請求項17之方法，其中該非揮發性記憶體係為NROM。

21. 如請求項17之方法，其中該非揮發性記憶體係在一記憶卡中。

22. 如請求項17至21中任一項之方法，其中該非揮發性記憶體具有各儲存一位元之資料的記憶體單元。

23. 如請求項17至21中任一項之方法，其中該非揮發性記憶體具有各儲存一位元以上之資料的記憶體單元。

24. 一種在一組織成區塊的非揮發性記憶體中用於儲存及更新資料的方法，其中各區塊已分割成可一起抹除的記憶體單元，各記憶體單元用於儲存一邏輯單元的資料，該方法包含：

將資料組織成複數個邏輯群組，各邏輯群組係為一群組邏輯的單元；

接收封裝在邏輯單元中的主機資料；

根據一第一順序，在一第一區塊中儲存邏輯單元的第一版本；及

根據和該第一順序不同的一第二順序，在一包括該等邏輯單元之後續版本的第二區塊中儲存邏輯單元的最新

版本；

提供用於各邏輯單元的一標頭部分；及

在儲存於該第二區塊中之各邏輯單元的該標頭部分中儲存目錄資料，該目錄資料可用於儲存在該第二區塊中的邏輯單元。

25. 如請求項24之方法，其中該非揮發性記憶體具有浮動閘極記憶體單元。

26. 如請求項24之方法，其中該非揮發性記憶體係為快閃EEPROM。

27. 如請求項24之方法，其中該非揮發性記憶體係為NROM。

28. 如請求項24之方法，其中該非揮發性記憶體係在一記憶卡中。

29. 如請求項24至28中任一項之方法，其中該非揮發性記憶體具有各儲存一位元之資料的記憶體單元。

30. 如請求項24至28中任一項之方法，其中該非揮發性記憶體具有各儲存一位元以上之資料的記憶體單元。

31. 一種在一組織成區塊的非揮發性記憶體中用於儲存及更新資料的方法，其中各區塊已分割成可一起抹除的記憶體單元，各記憶體單元用於儲存一邏輯單元的資料，該方法包含：

將資料組織成複數個邏輯群組，各邏輯群組係為一群組邏輯的單元；

接收封裝在邏輯單元中的主機資料；

根據一第一順序，在一第一區塊中儲存一邏輯群組之

邏輯單元的第一版本；及

根據一第二順序，在一包括該等邏輯單元之後續版本的第二區塊中儲存邏輯單元的最新版本，其中已儲存的該等邏輯單元具有和該第一順序相同順序的一第一子群組及和該第一順序不同順序的一第二子群組；及

在該非揮發性記憶體的部分中維持該第二子群組之邏輯單元的一目錄。

32. 如請求項31之方法，其中該非揮發性記憶體具有浮動閘極記憶體單元。

33. 如請求項31之方法，其中該非揮發性記憶體係為快閃EEPROM。

34. 如請求項31之方法，其中該非揮發性記憶體係為NROM。

35. 如請求項31之方法，其中該非揮發性記憶體係在一記憶卡中。

36. 如請求項31至35中任一項之方法，其中該非揮發性記憶體具有各儲存一位元之資料的記憶體單元。

37. 如請求項31至35中任一項之方法，其中該非揮發性記憶體具有各儲存一位元以上之資料的記憶體單元。

十一、圖式：

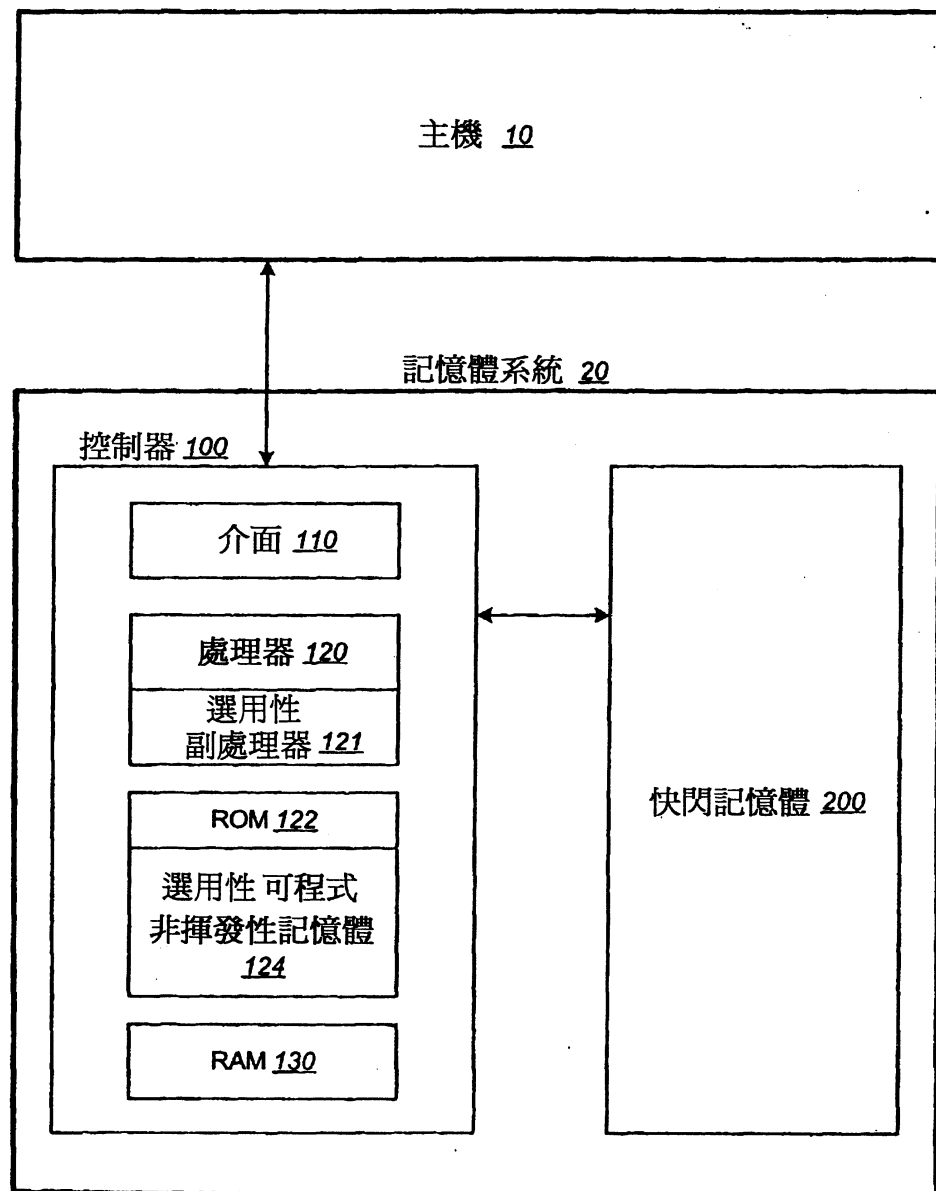


圖 1

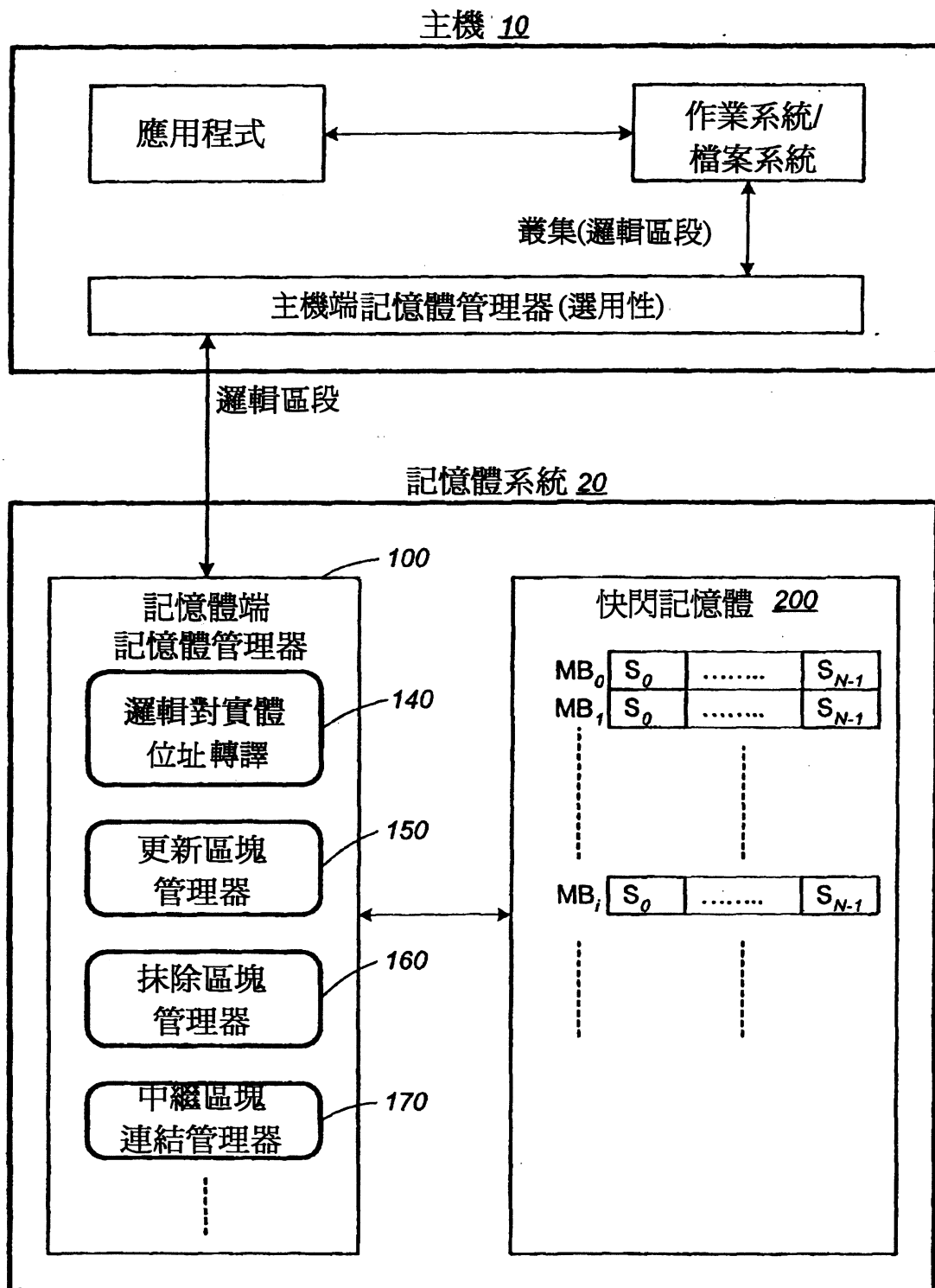


圖 2

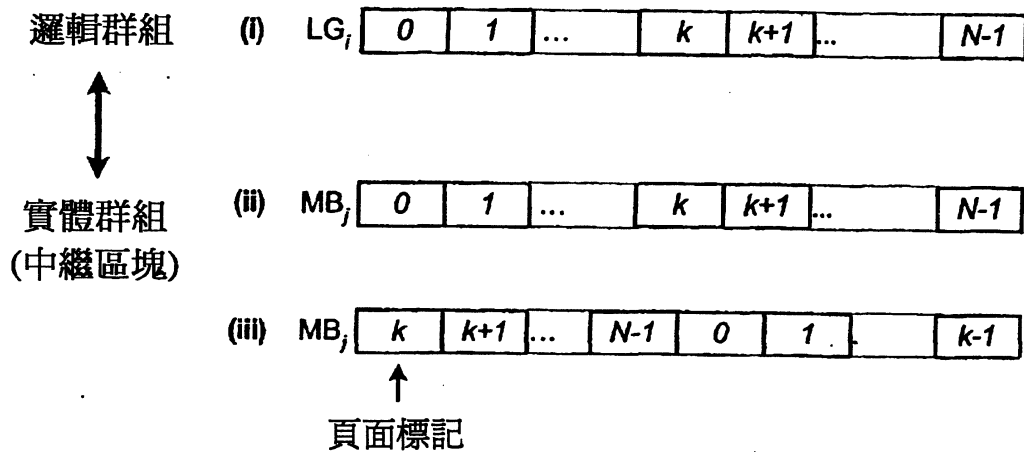


圖 3A

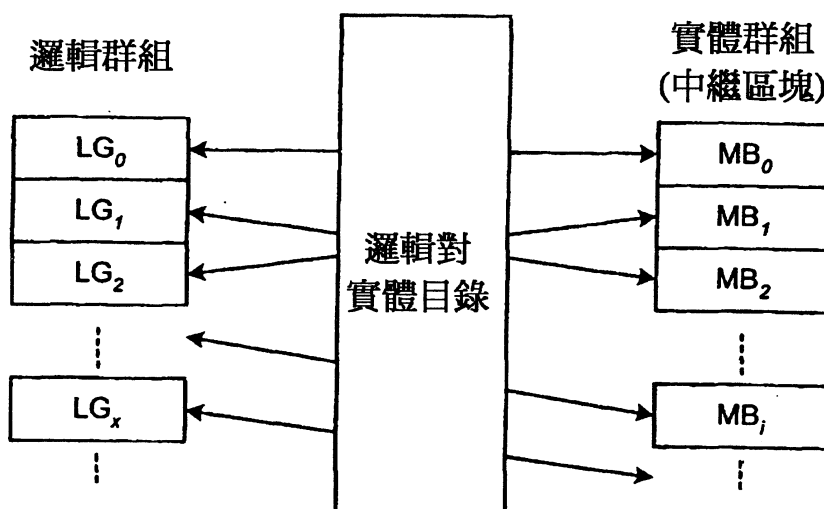


圖 3B

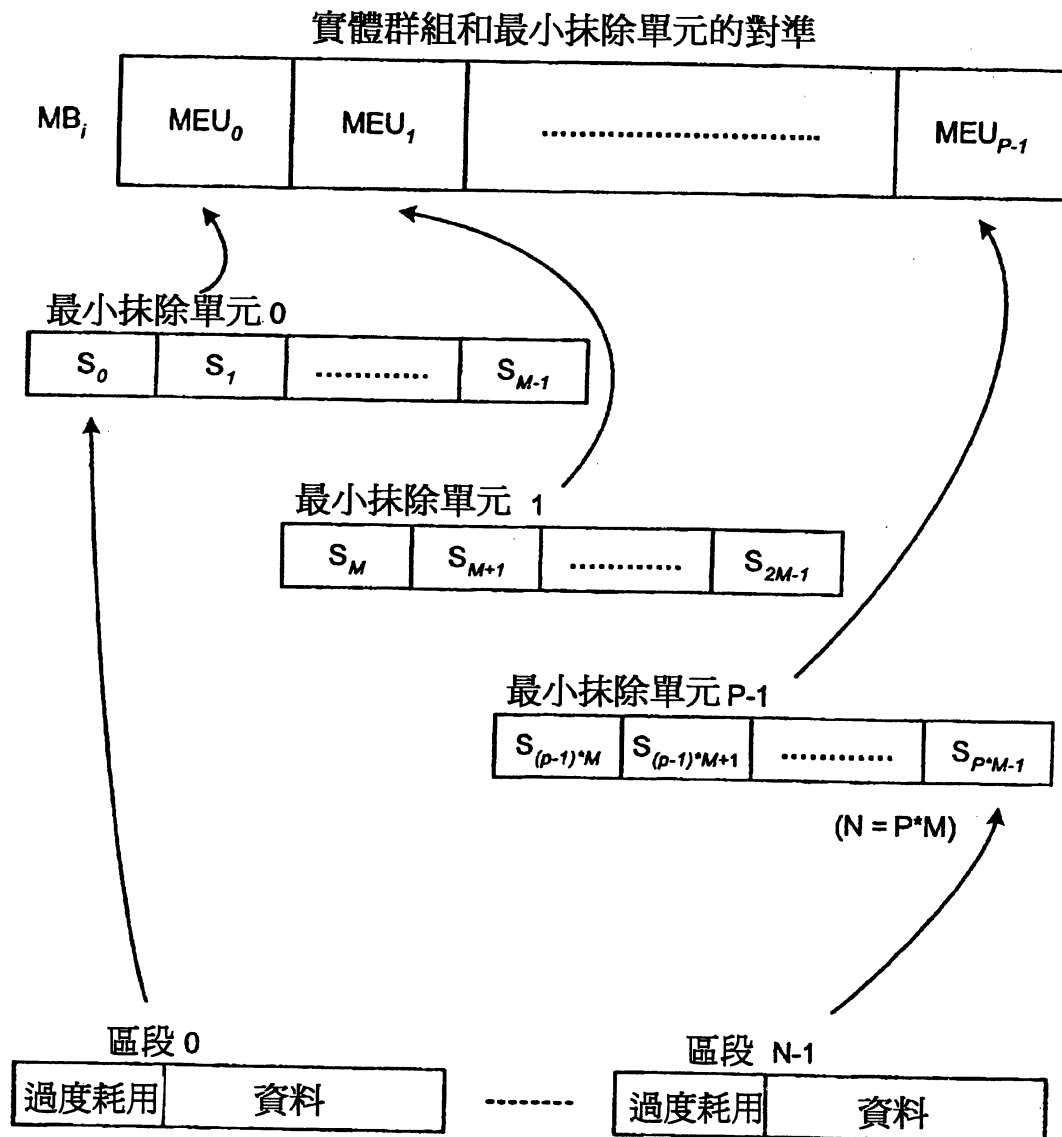


圖 4

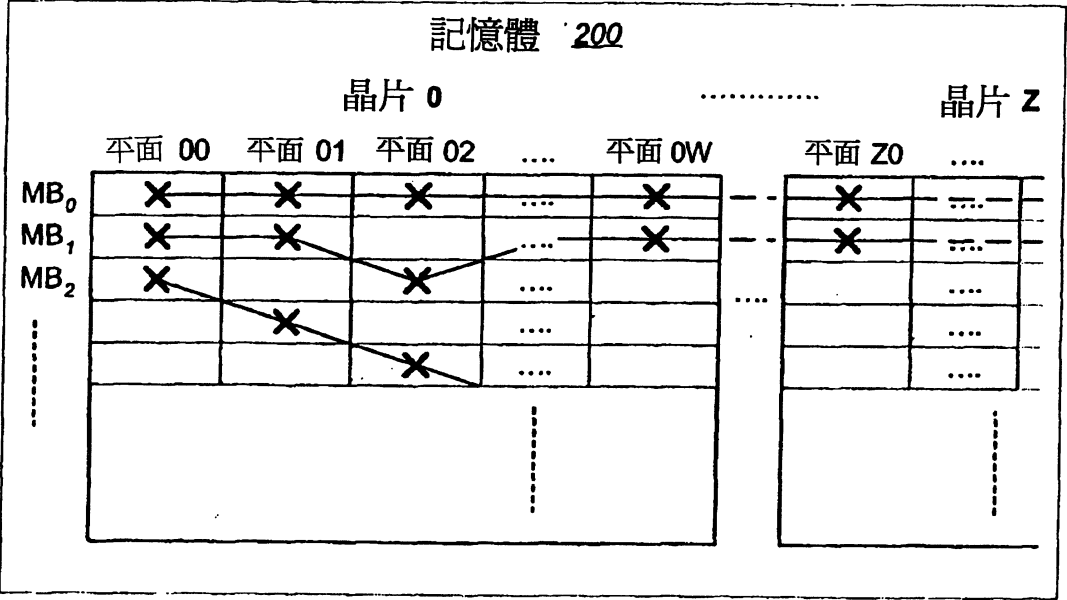


圖 5A



圖 5B

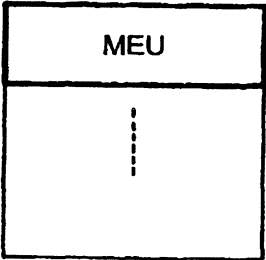


圖 5C

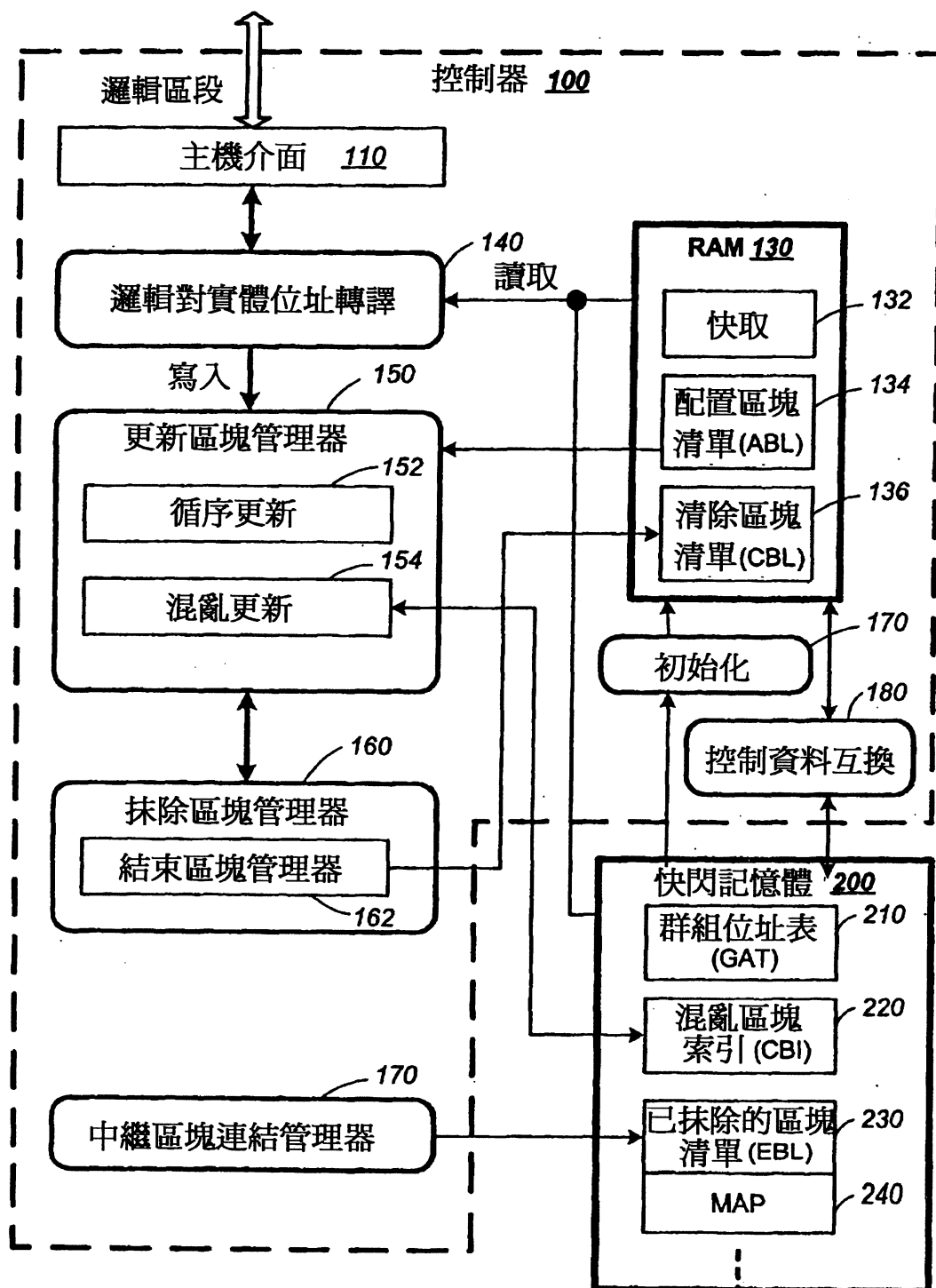
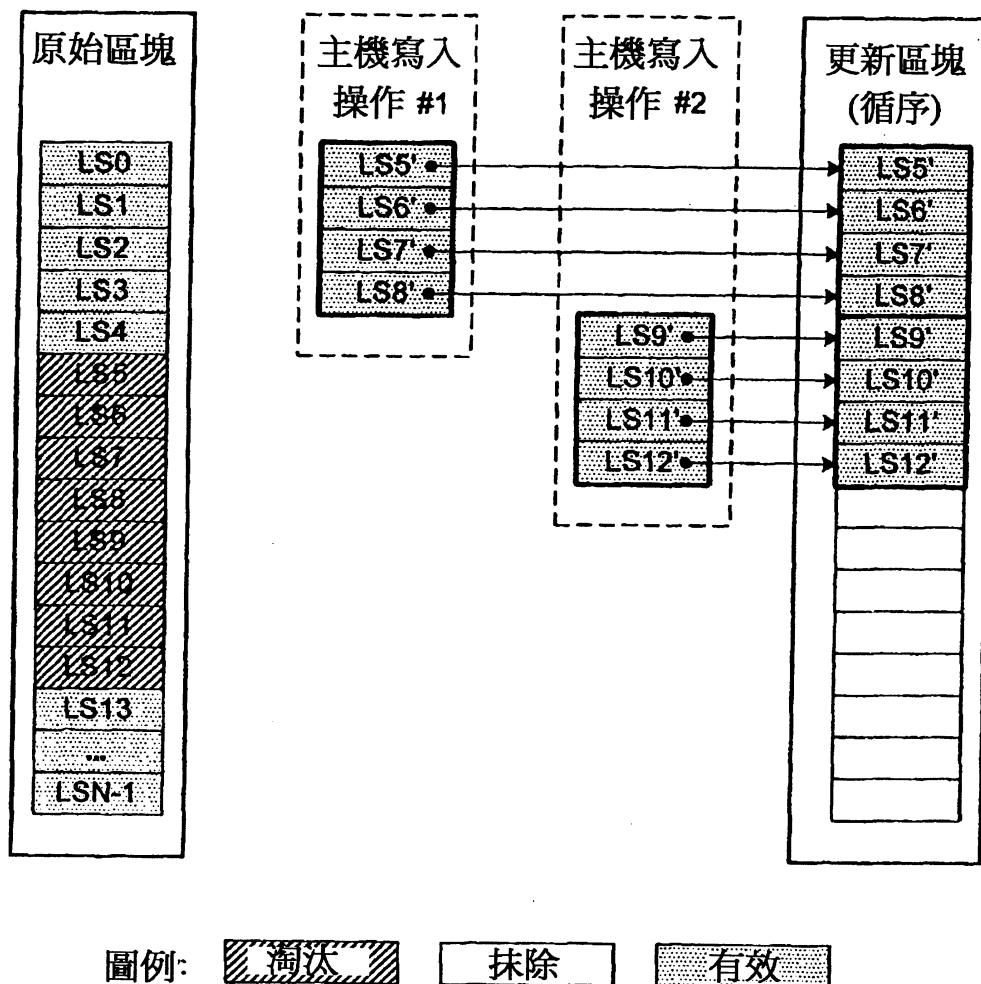
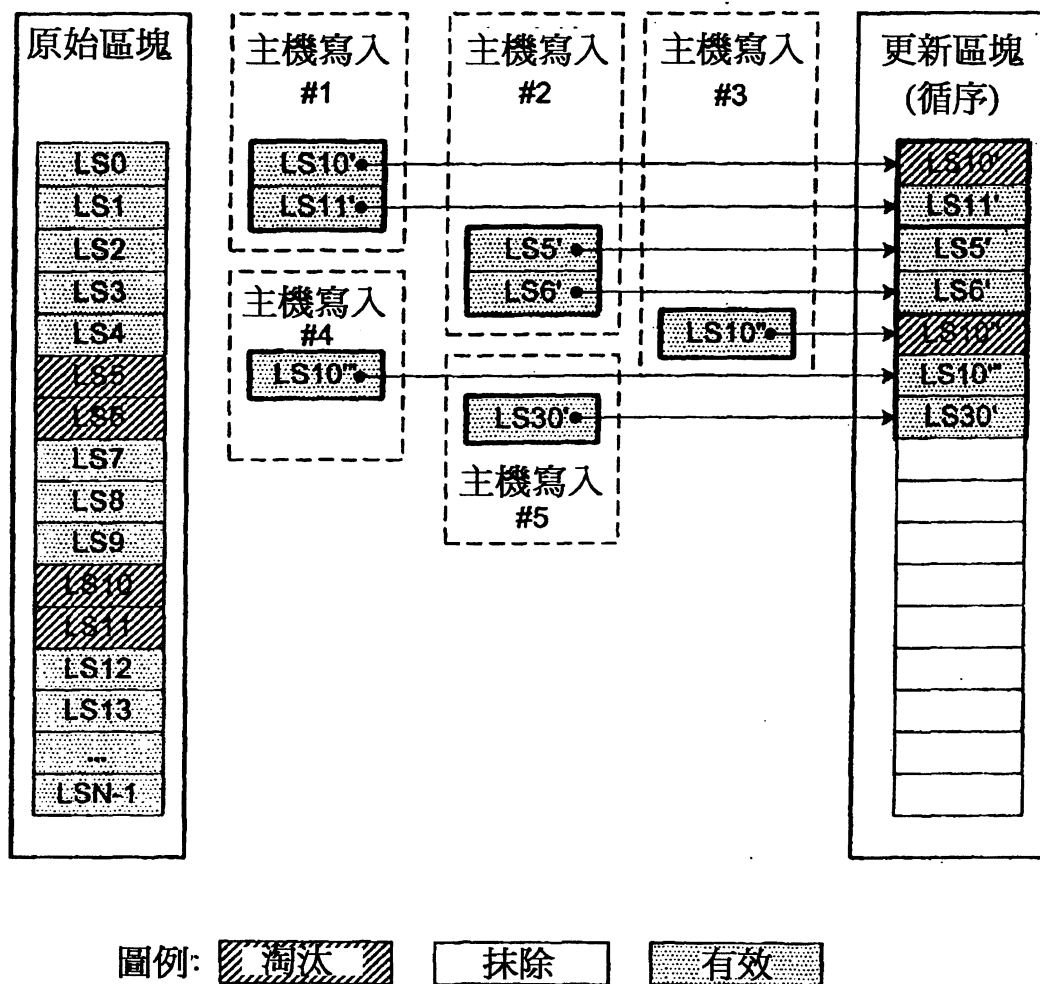


圖 6



循序更新範例

圖 7A



混亂更新(非循序)範例

圖 7B

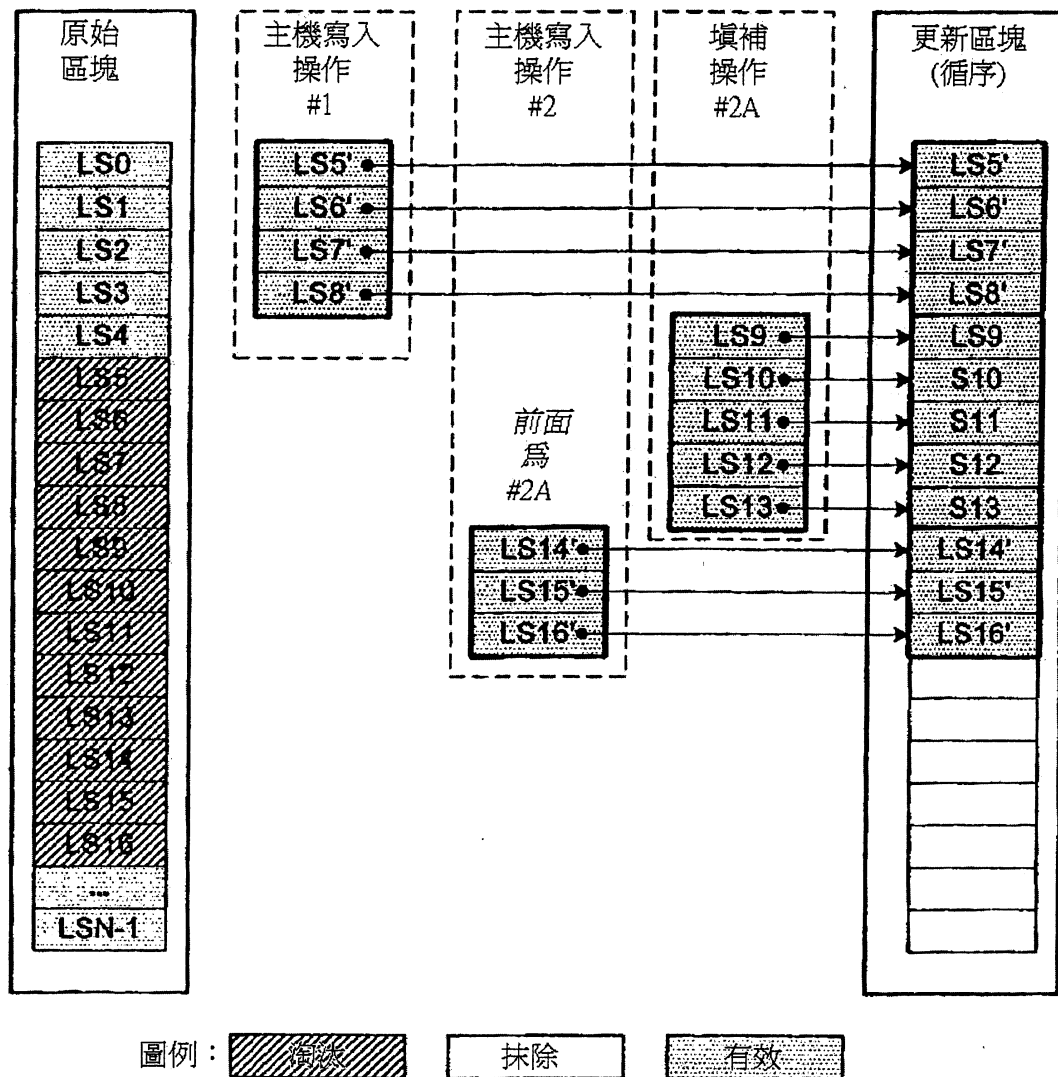


圖 8

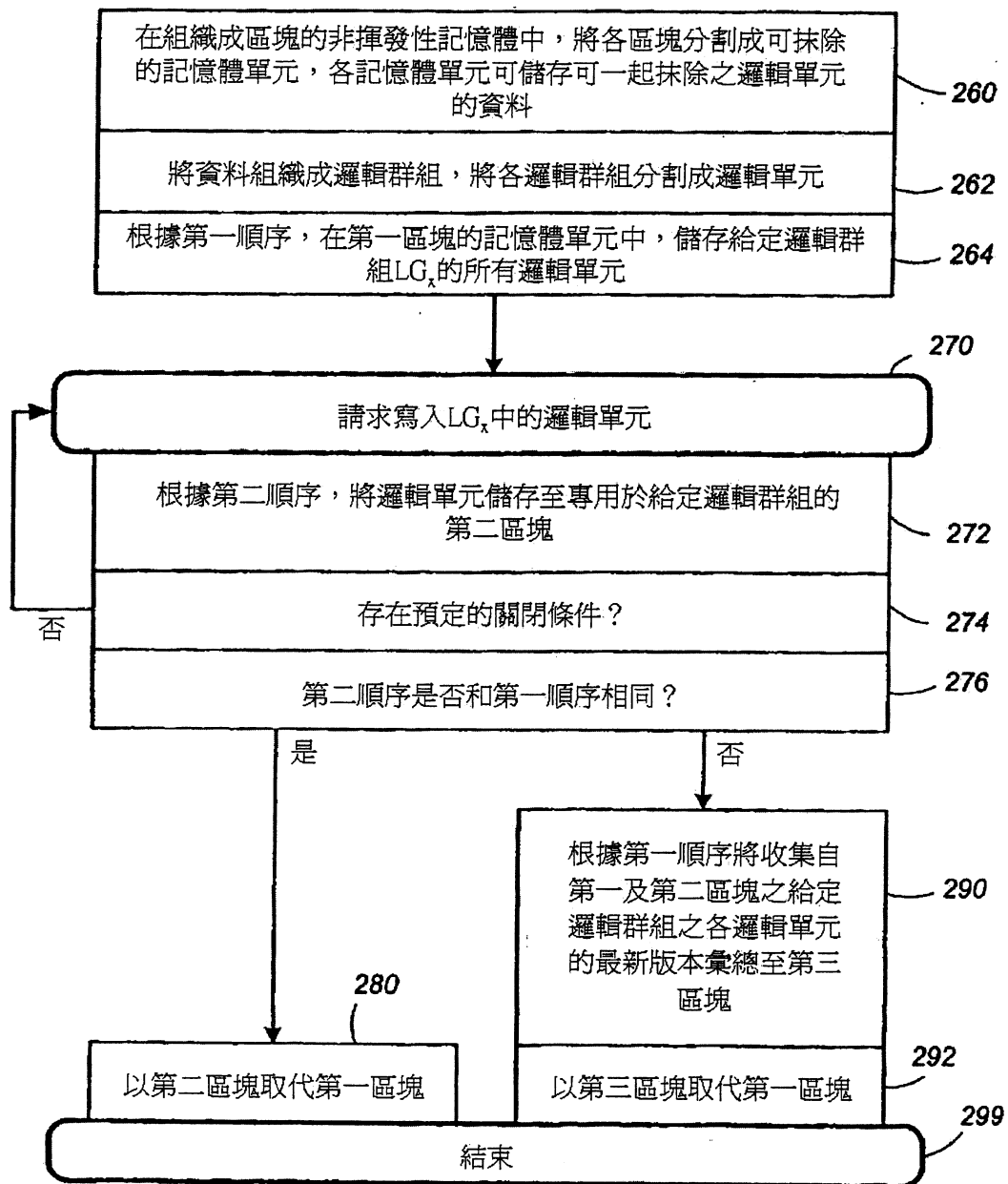


圖 9

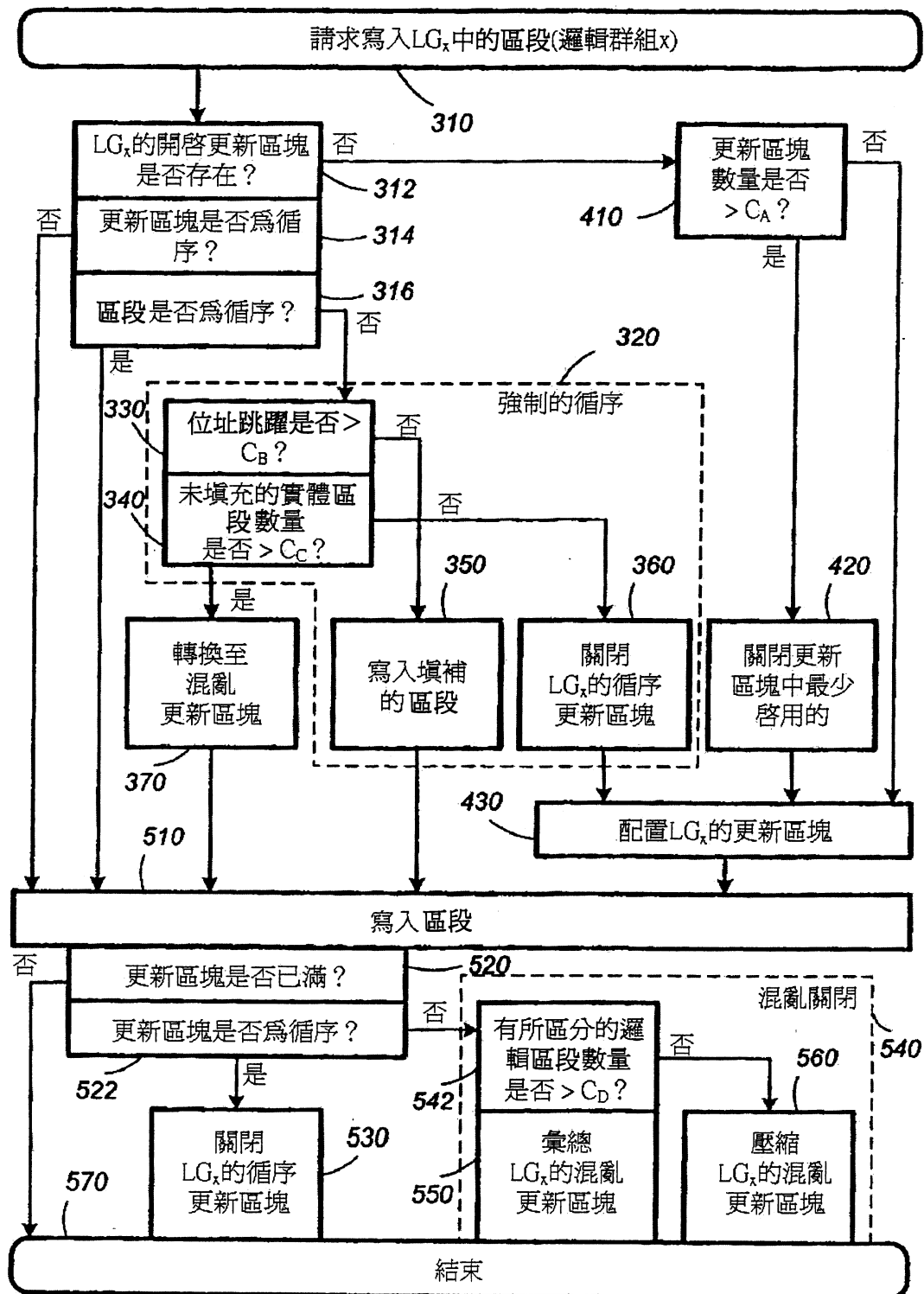


圖 10

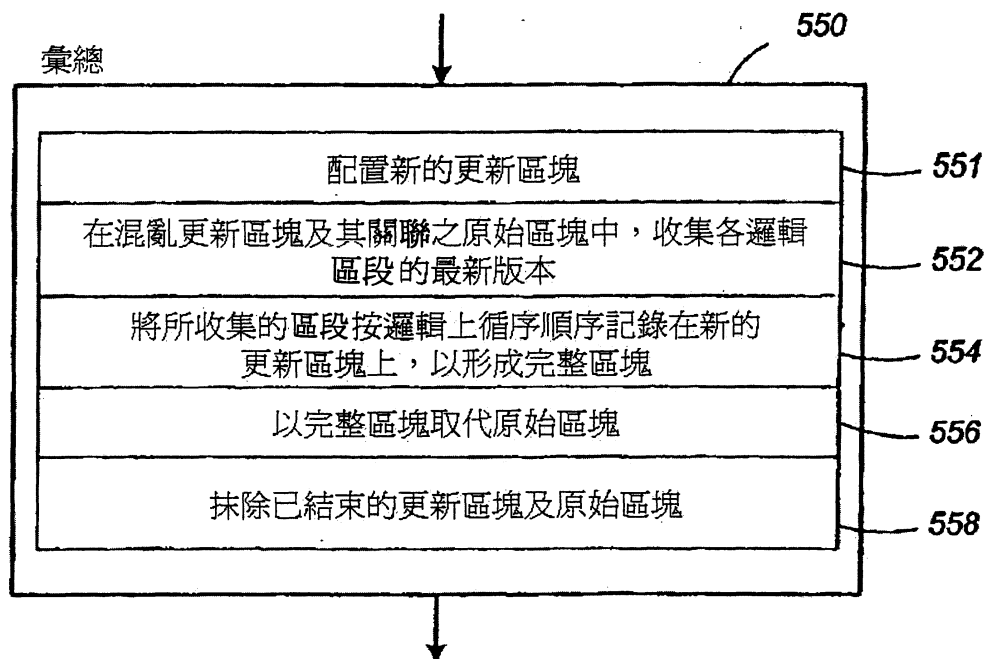


圖 11A

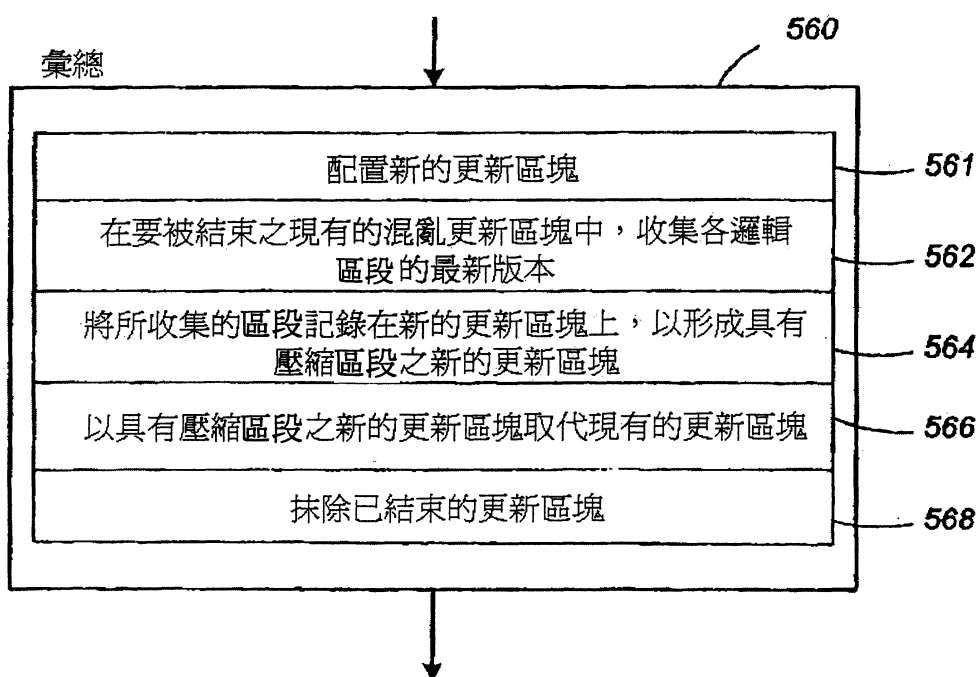
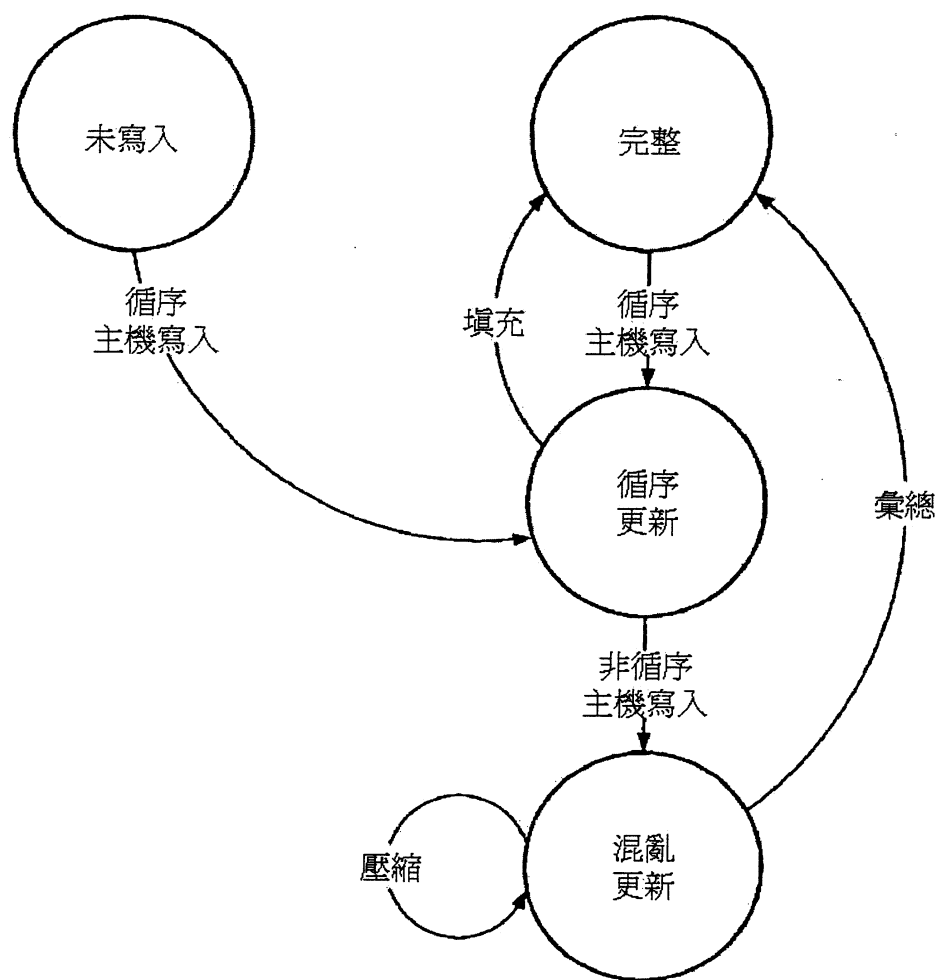


圖 11B

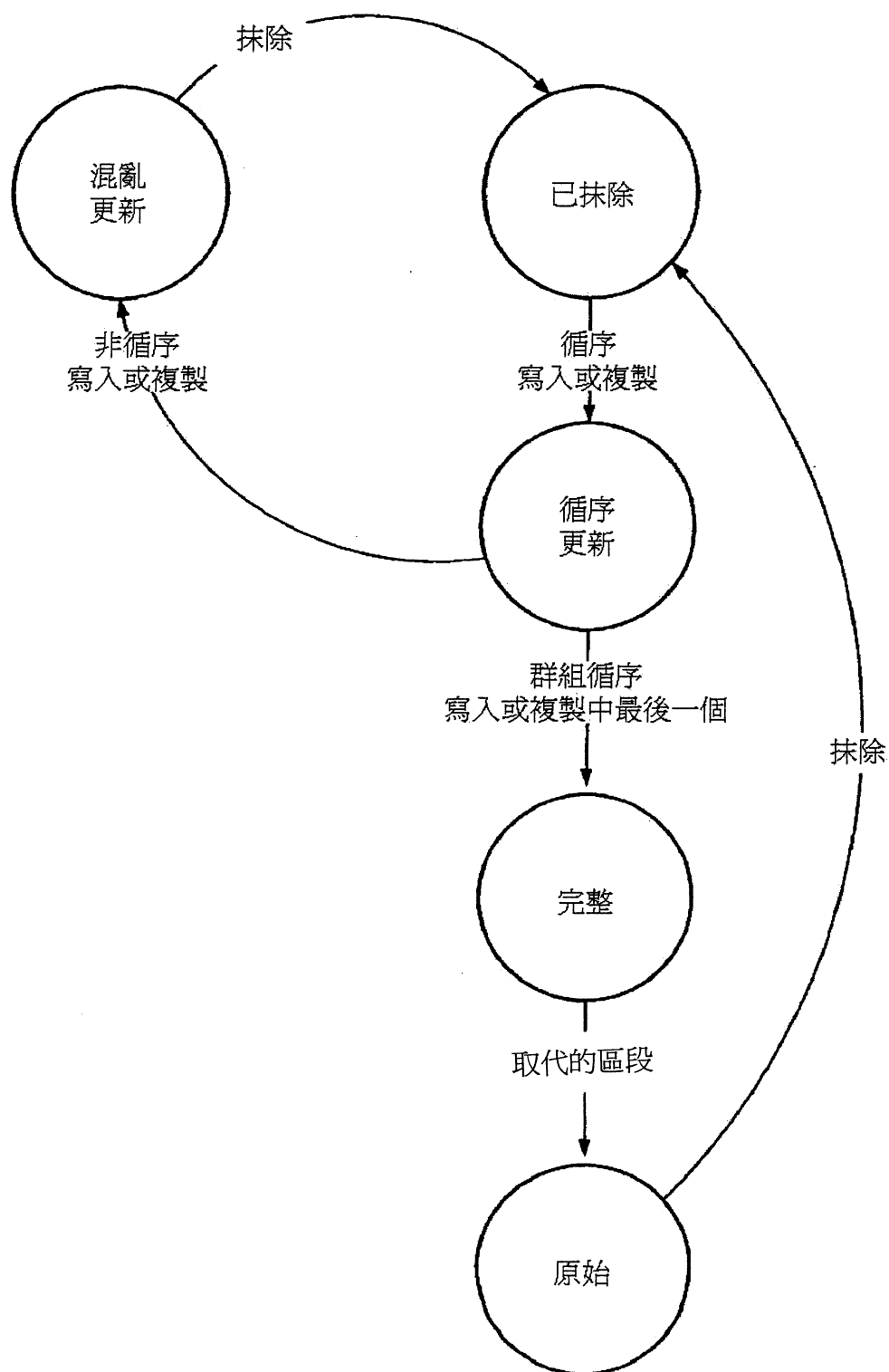


邏輯群組狀態

圖 12A

邏輯 群組狀態	描述
完整	單一中繼區塊中按邏輯上循序順序的完整邏輯群組
未寫入	邏輯群組尚未有任何部分被寫入
循序 更新	將邏輯群組的更新程式段寫入更新區塊中而未變更其現有的循序特性
混亂更新	將邏輯群組的更新程式段按邏輯上非循序順序寫入更新區塊中

圖 12B



實體中繼區塊狀態

圖 13A

中繼區塊 狀態	描述
已抹除	中繼區塊中的所有區段均被抹除
循序 更新	中繼區塊中按邏輯上循序順序部分寫入區段，可能使用 頁面標記。所有區段均屬於相同的邏輯群組
非循序 (混亂)更新	中繼區塊中按邏輯上非循序順序部分或完全寫入區段。 任何區段均可被寫入多於一次。所有區段均屬於相同的 邏輯群組
完整	按邏輯上循序順序完全寫入中繼區塊， 可能使用頁面標記
原始	中繼區塊先前為完整但至少一區段已為主機資料 更新所淘汰

圖 13B

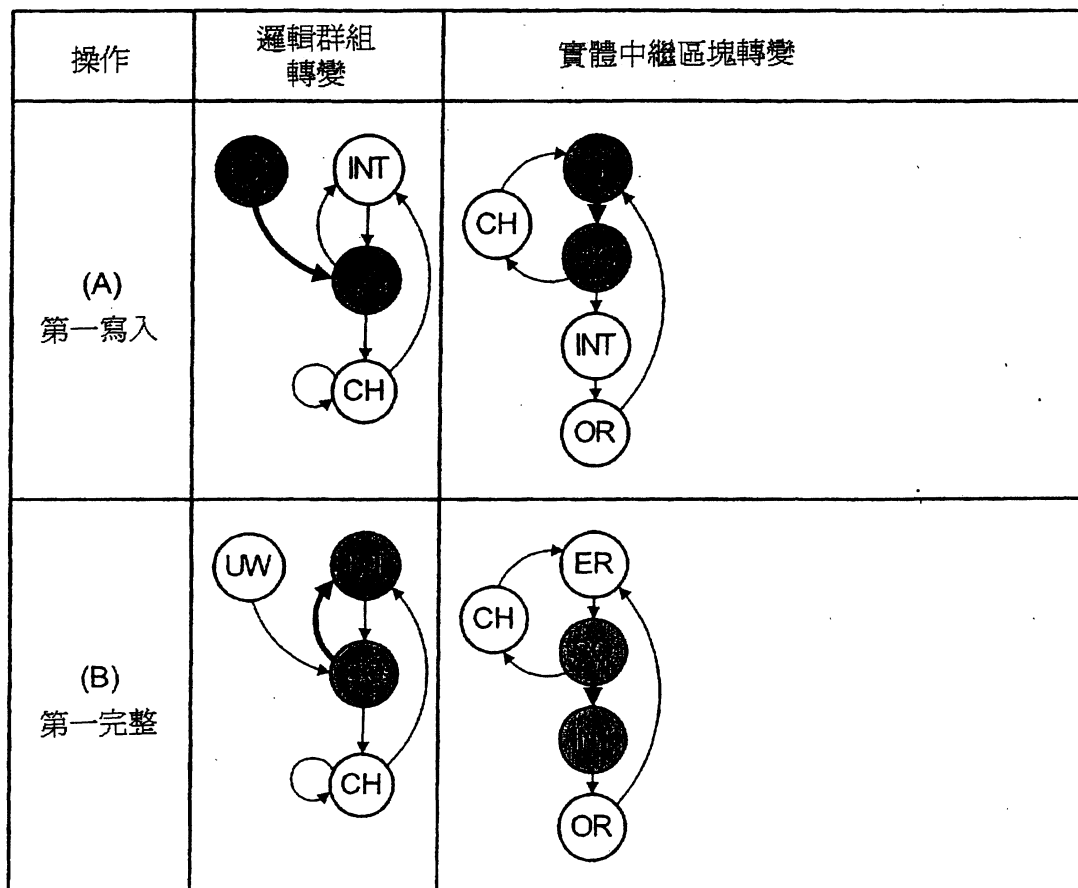


圖 14(A)-14(B)

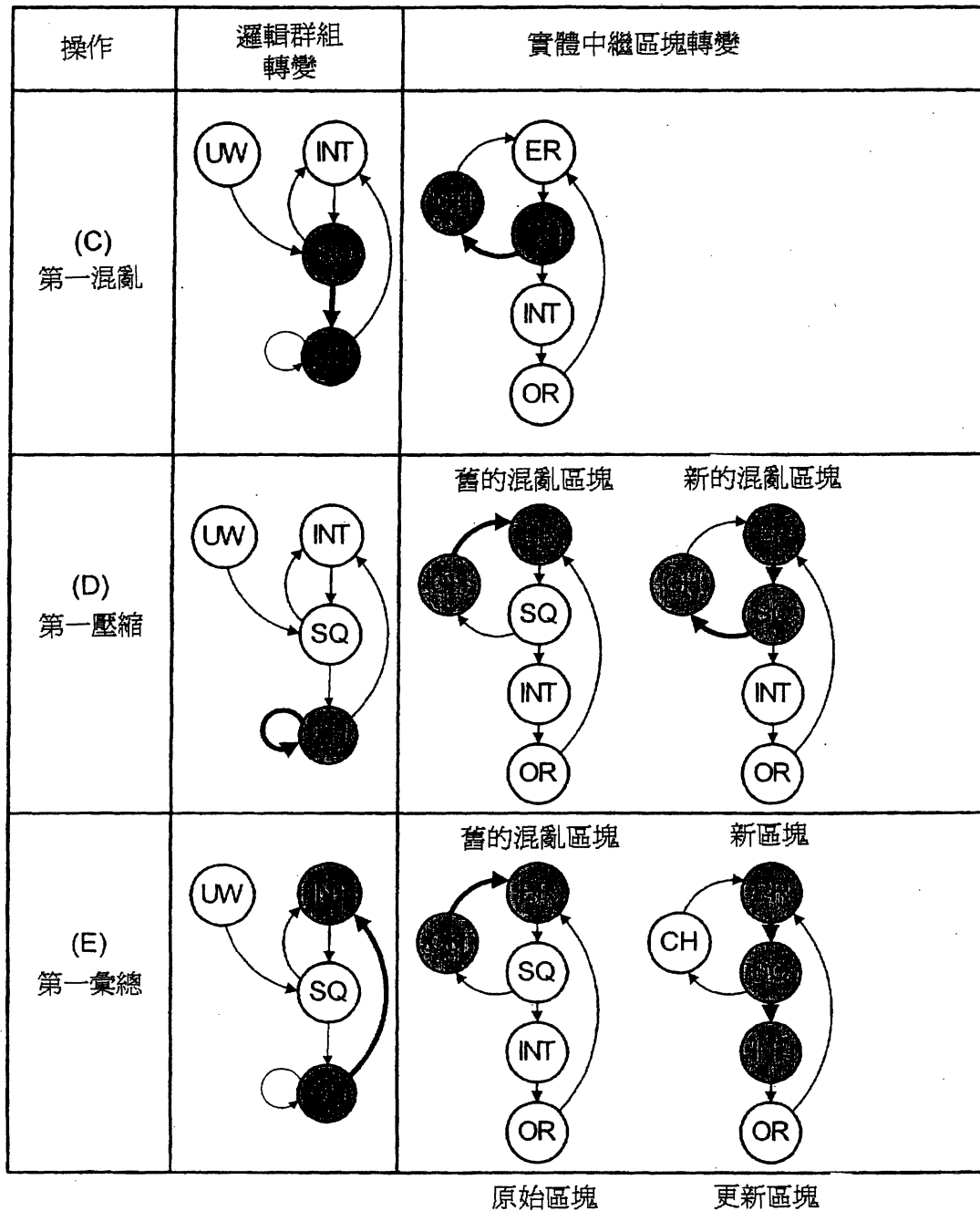


圖 14(C)-14(E)

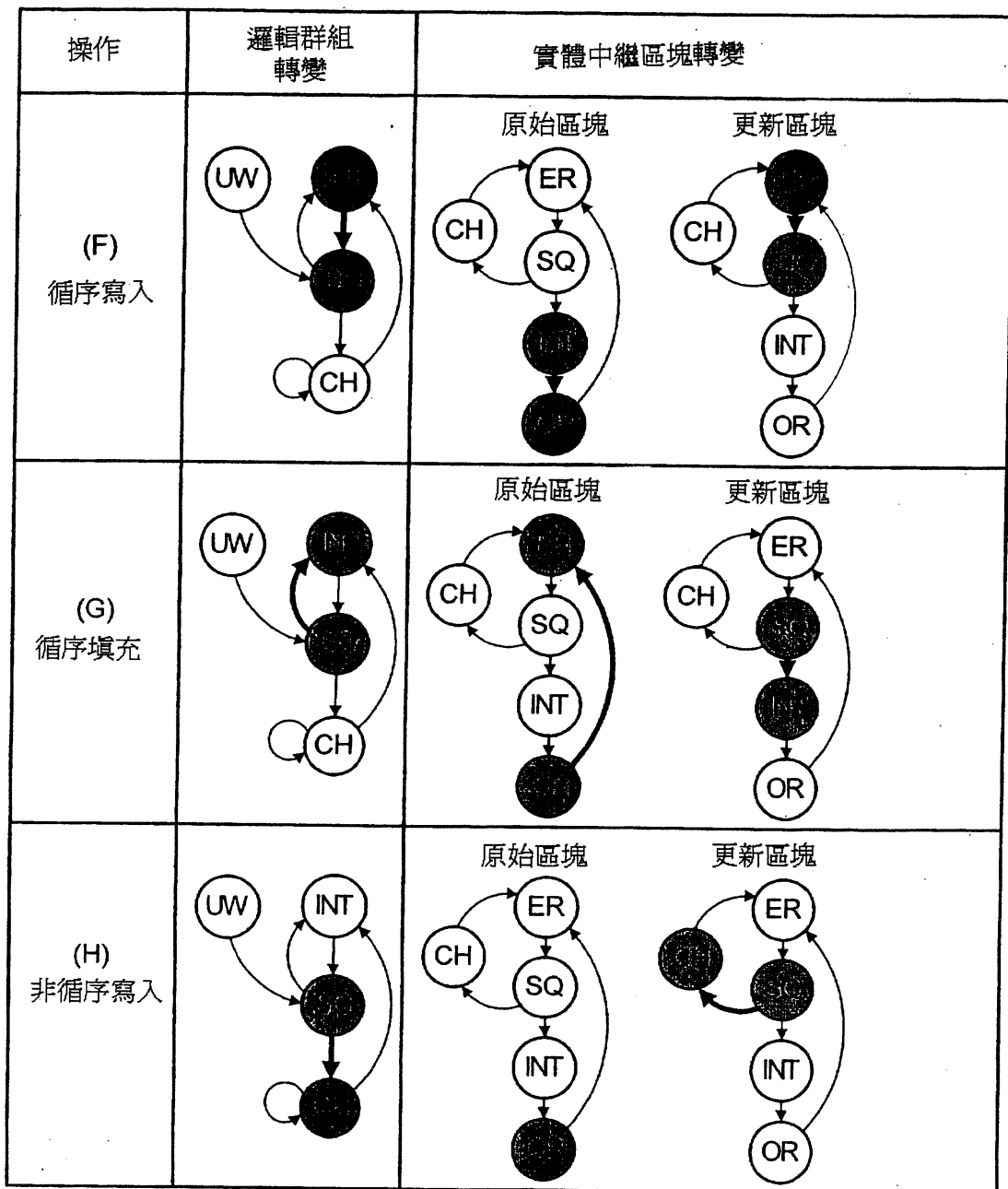


圖 14(F)-14(H)

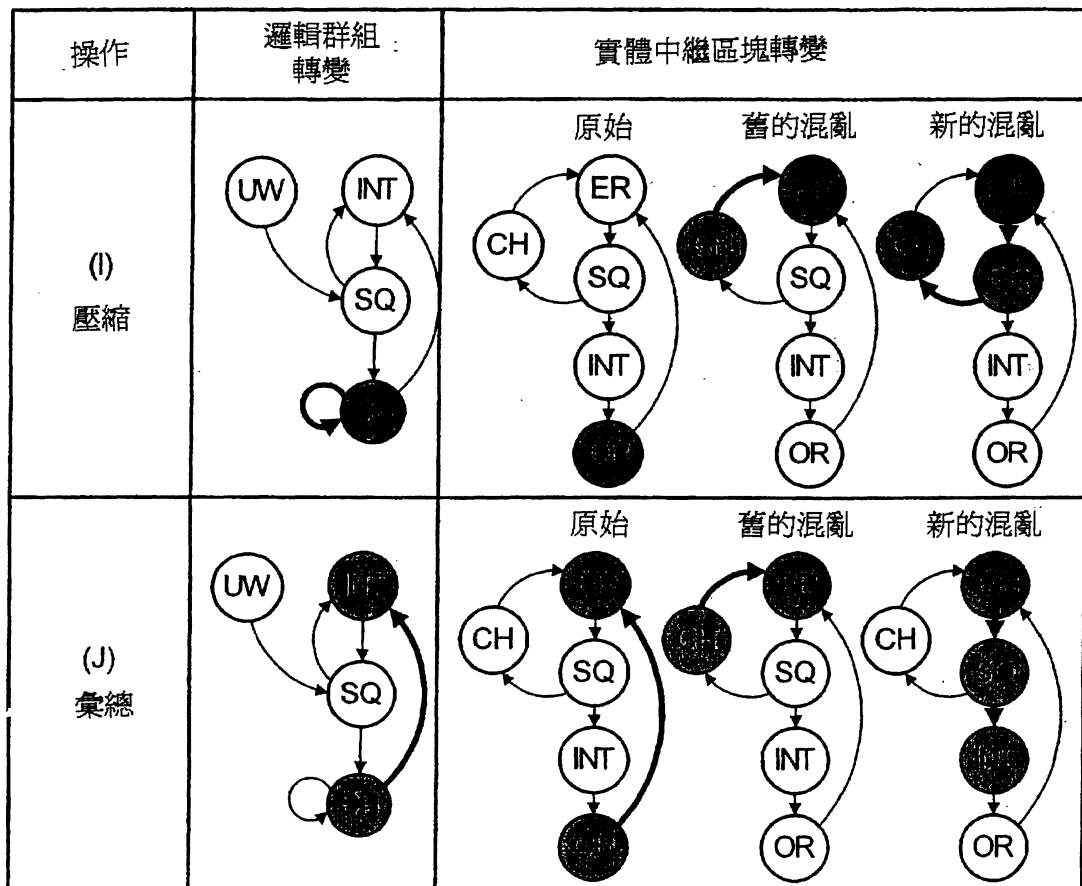


圖 14(I)-14(J)

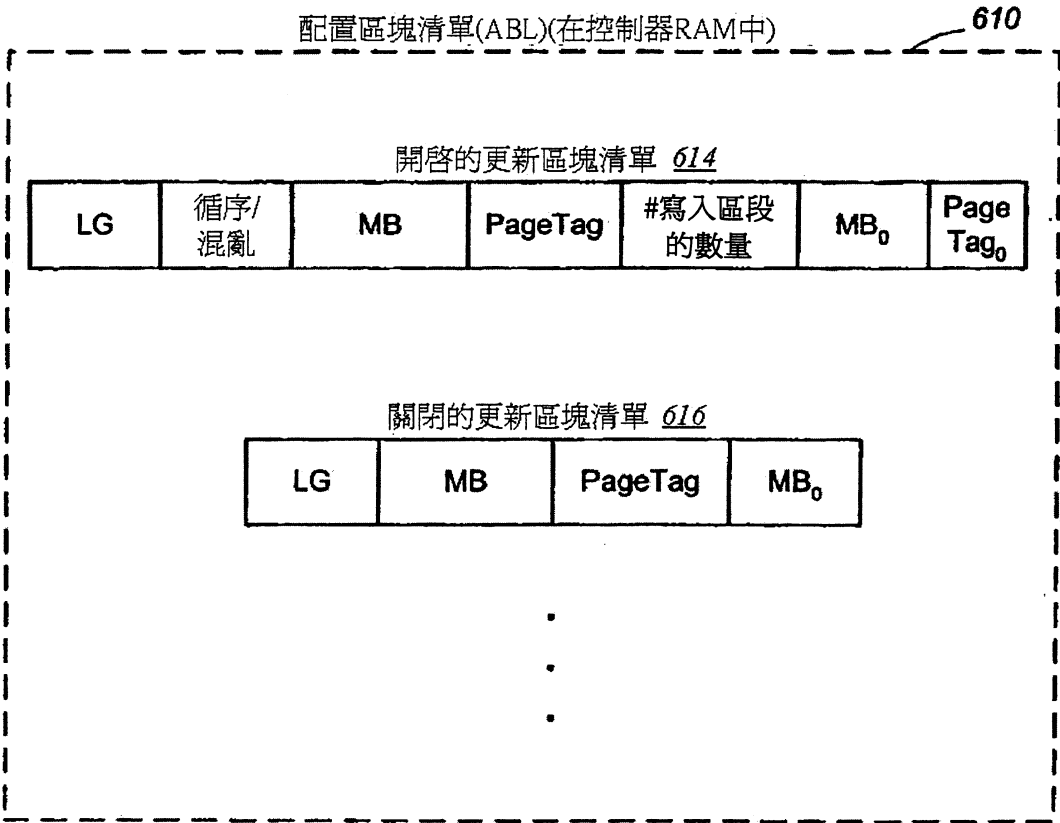


圖 15

混亂區塊索引(CBI)區段		
混亂區塊索引	混亂區塊資訊	CBI區段索引

圖 16A

CBI區塊 620	
LG ₀ 的CBI區段	
LG ₁ 的CBI區段	
LG ₂ 的CBI區段	
⋮	
LG _i 的CBI區段	
⋮	
LG _i 的CBI區段(最新版本)	
LG ₂₀₂ 的CBI區段	
LG ₁₃₅ 的CBI區段	
LG ₁₃₆ 的CBI區段(最後寫入)	
未寫入	

圖 16B

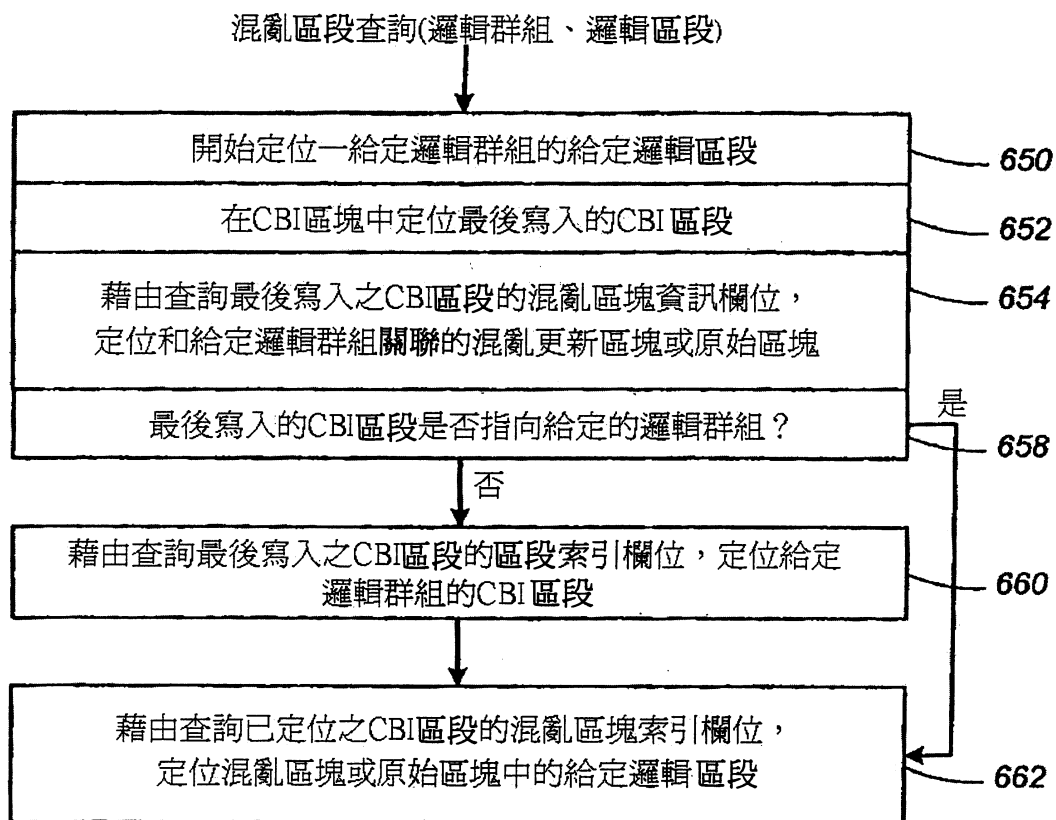


圖 16C

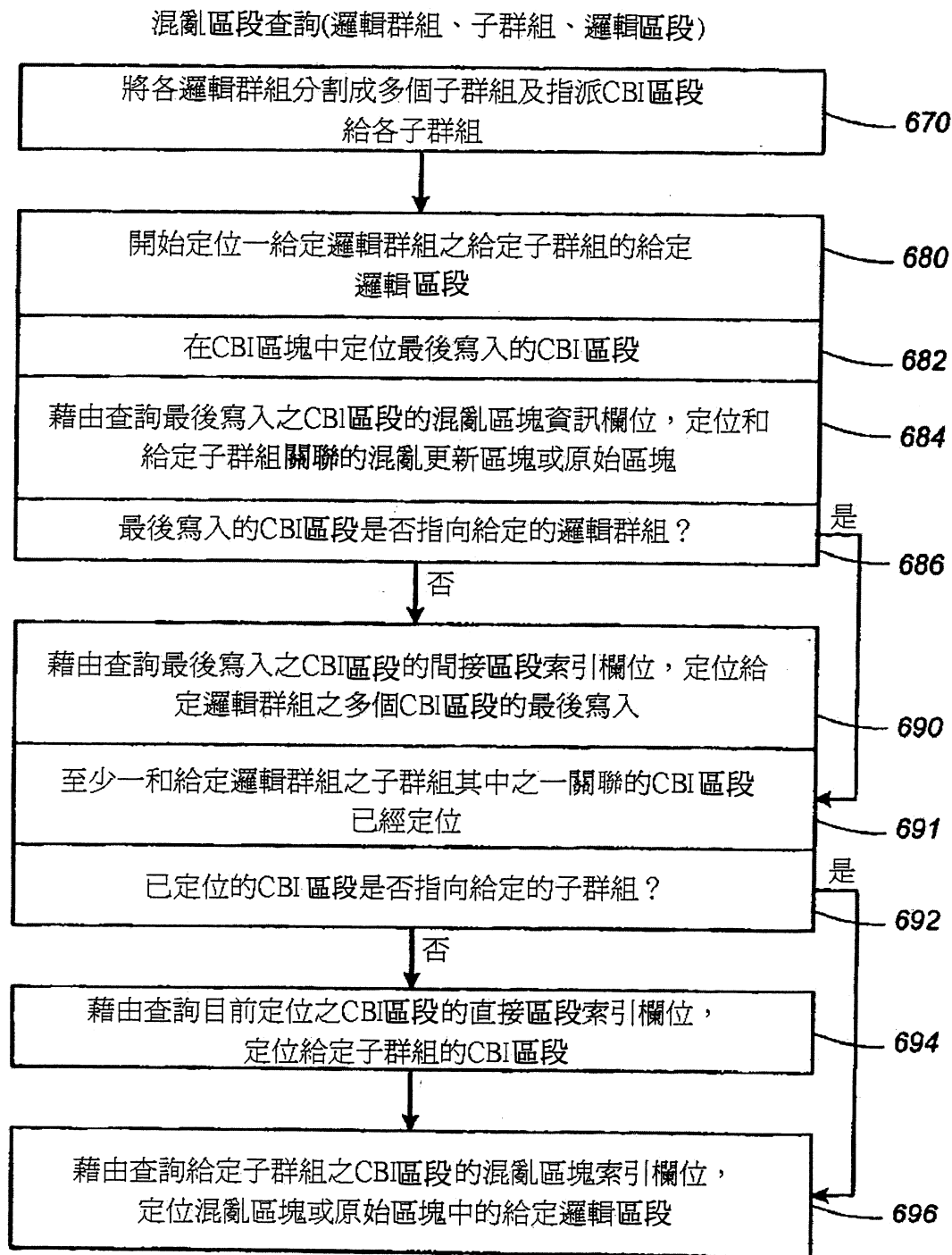


圖 16D

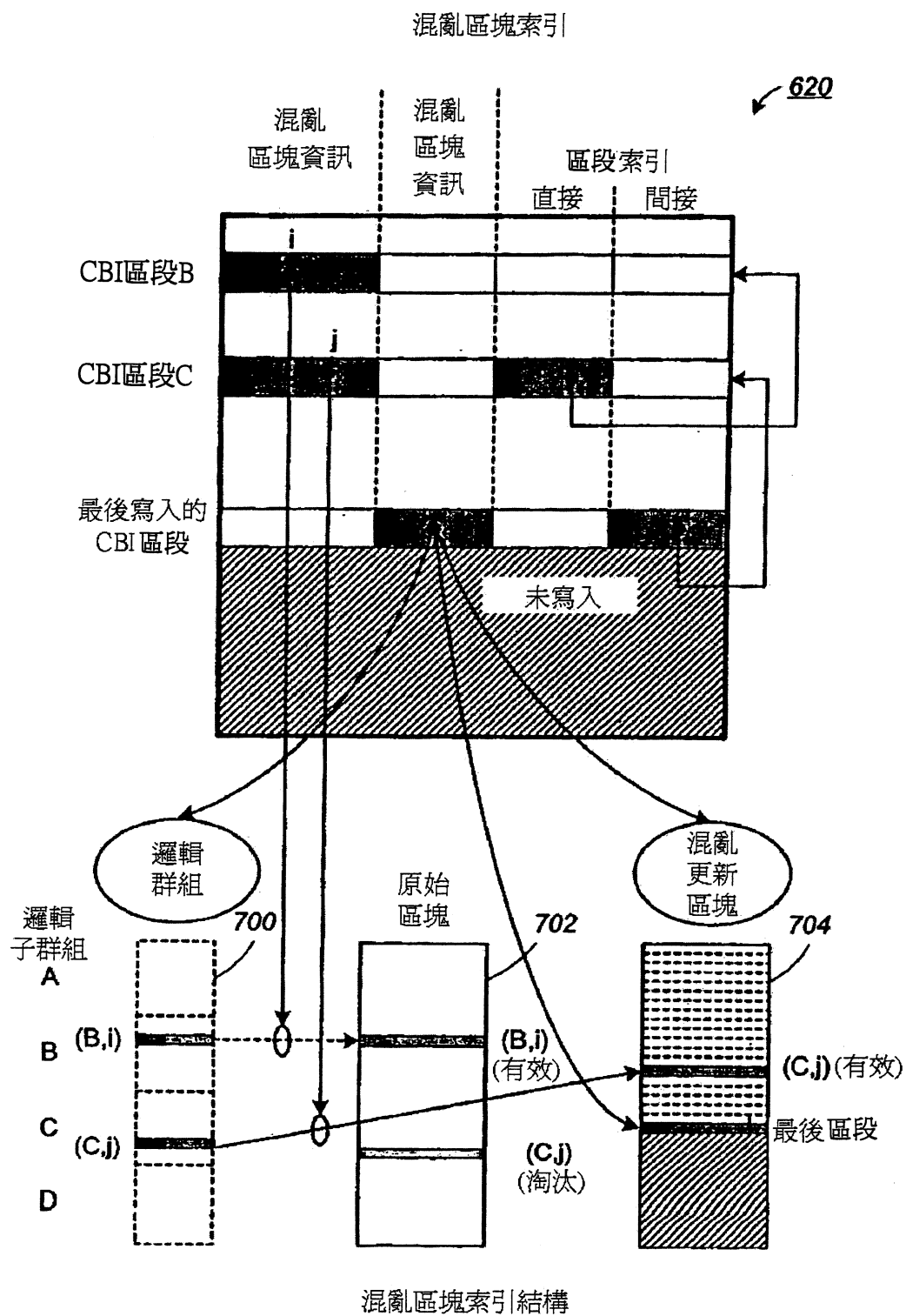


圖 16E

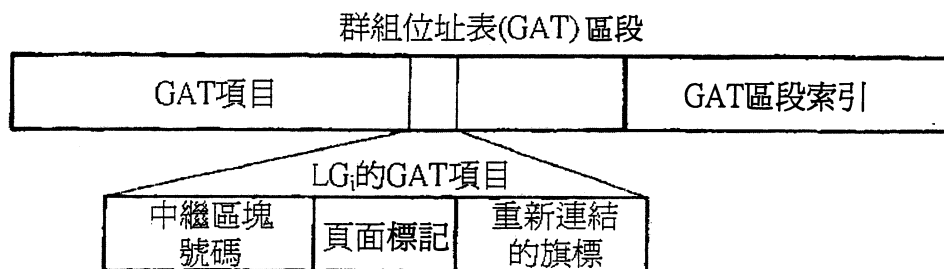


圖 17A

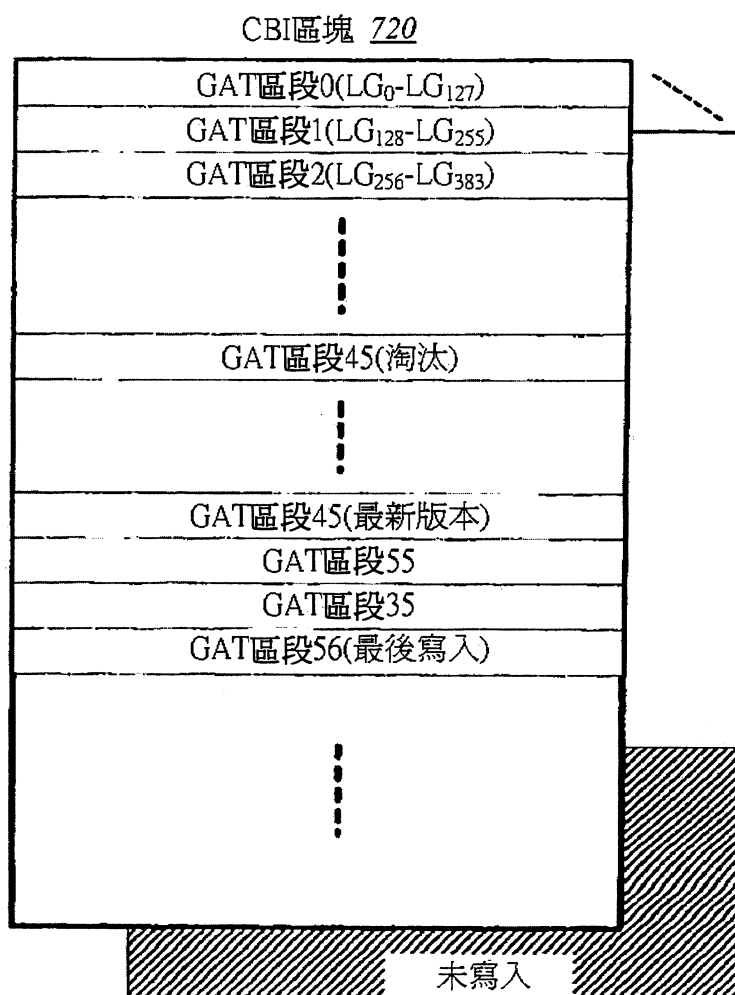


圖 17B

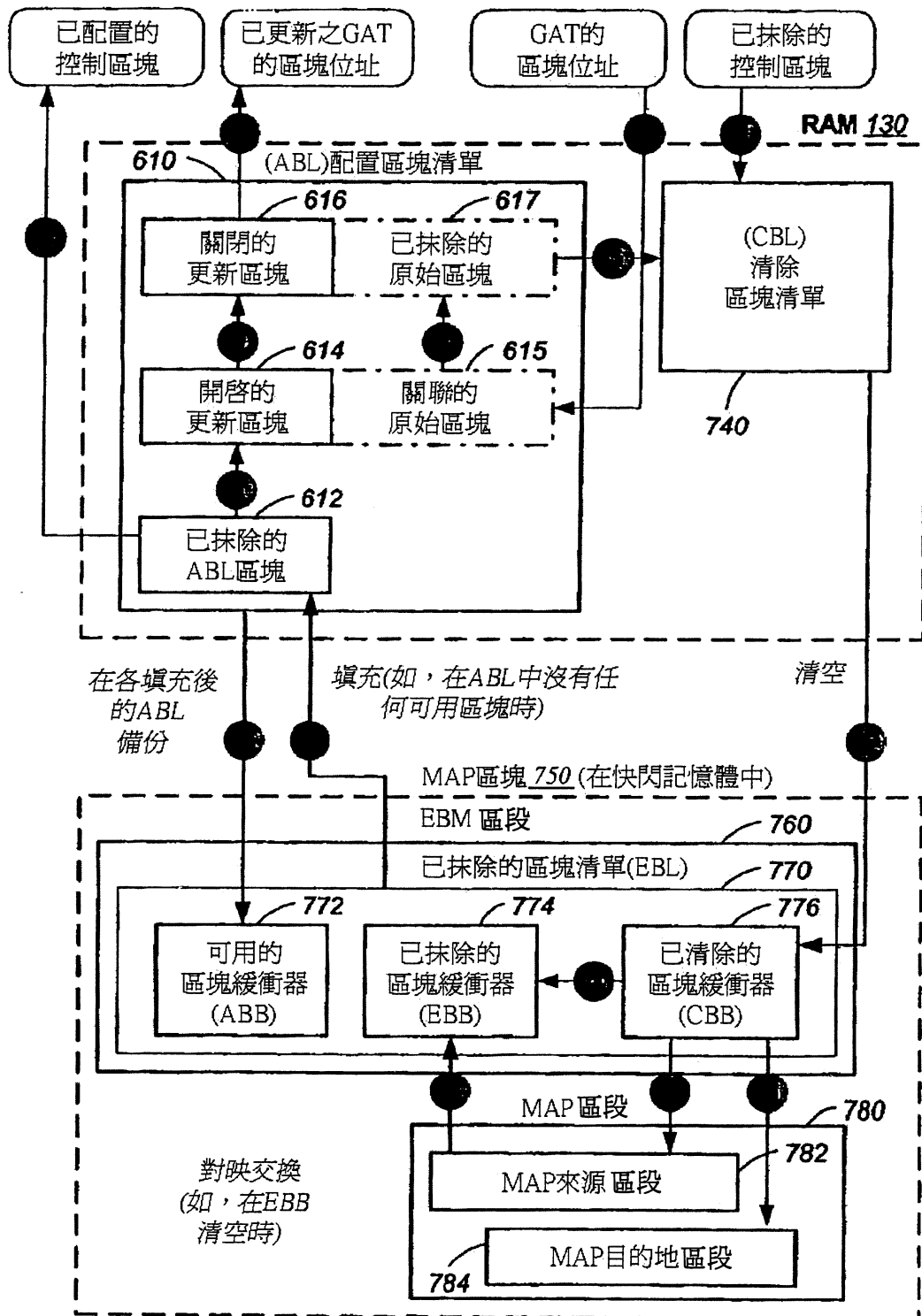
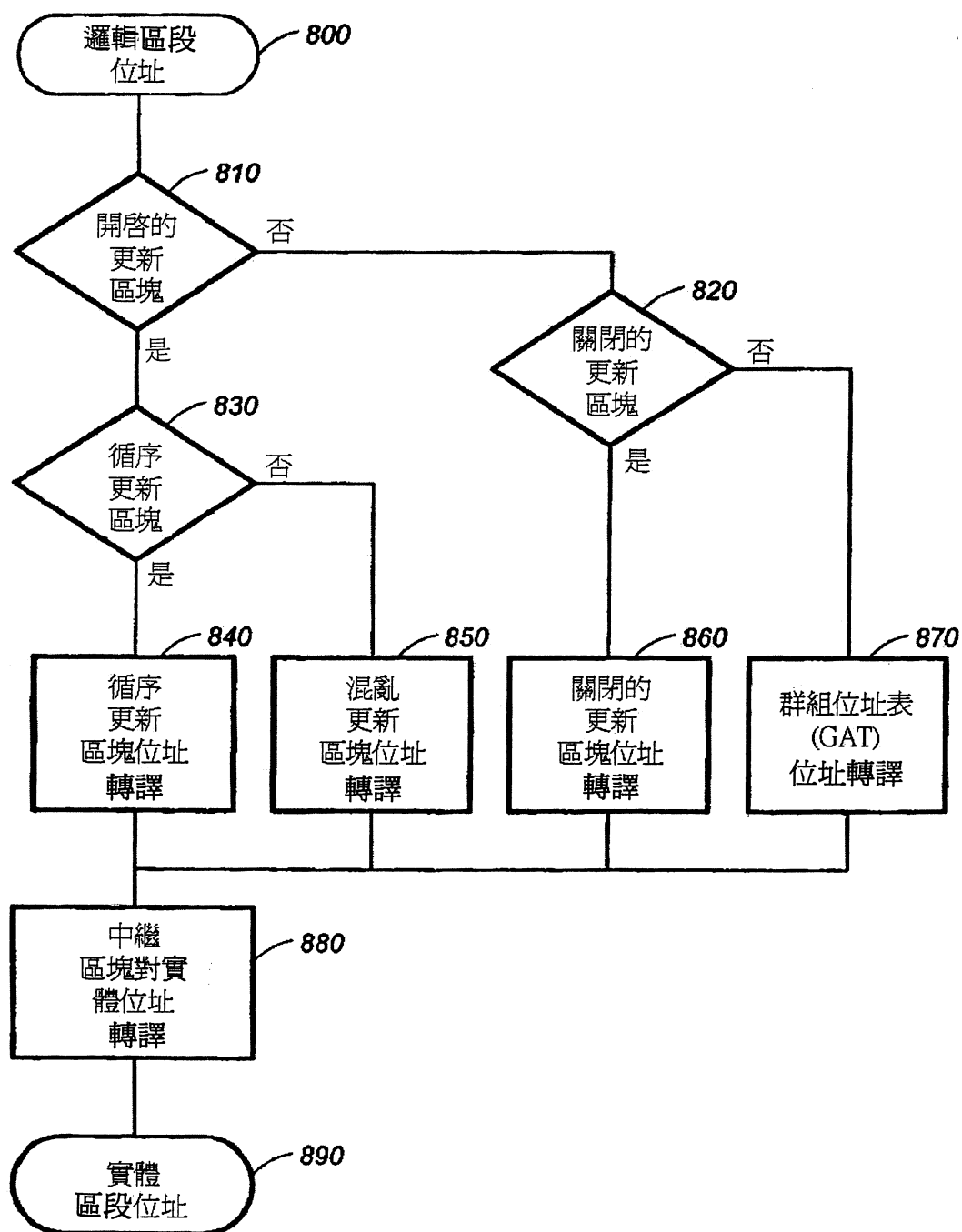
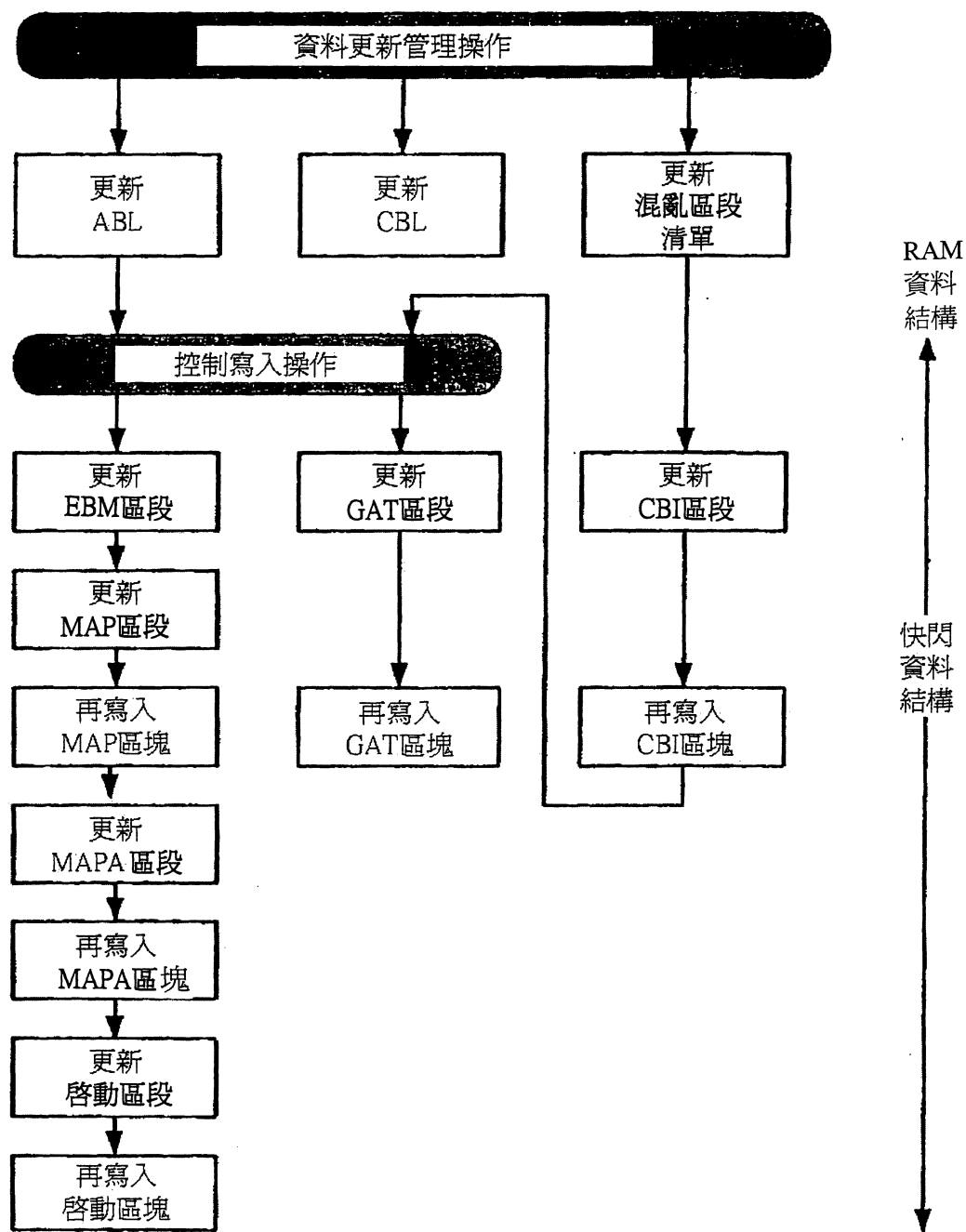


圖 18



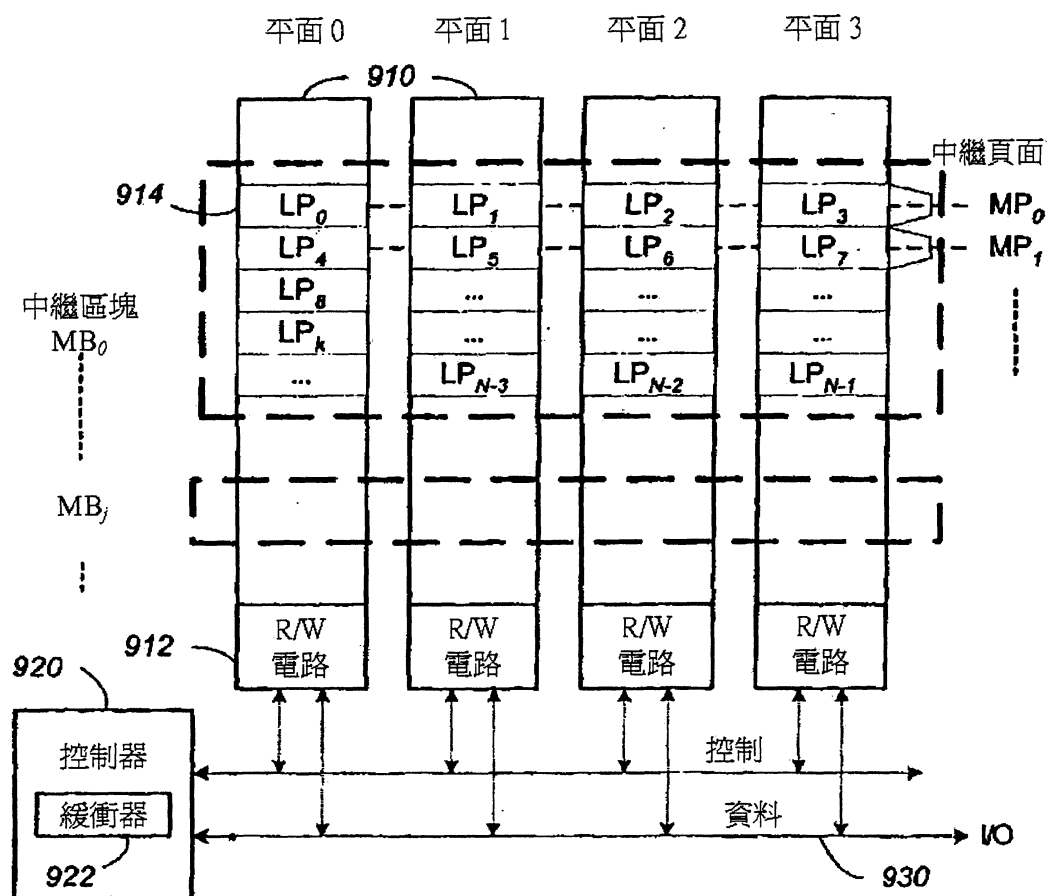
位址轉譯

圖 19



控制資料結構的操作

圖 20



邏輯頁面LP_k常駐於平面x
 其中 $x = k \text{ MOD}(k, W)$ ，
 W(如=4)是平面的總數

圖 21

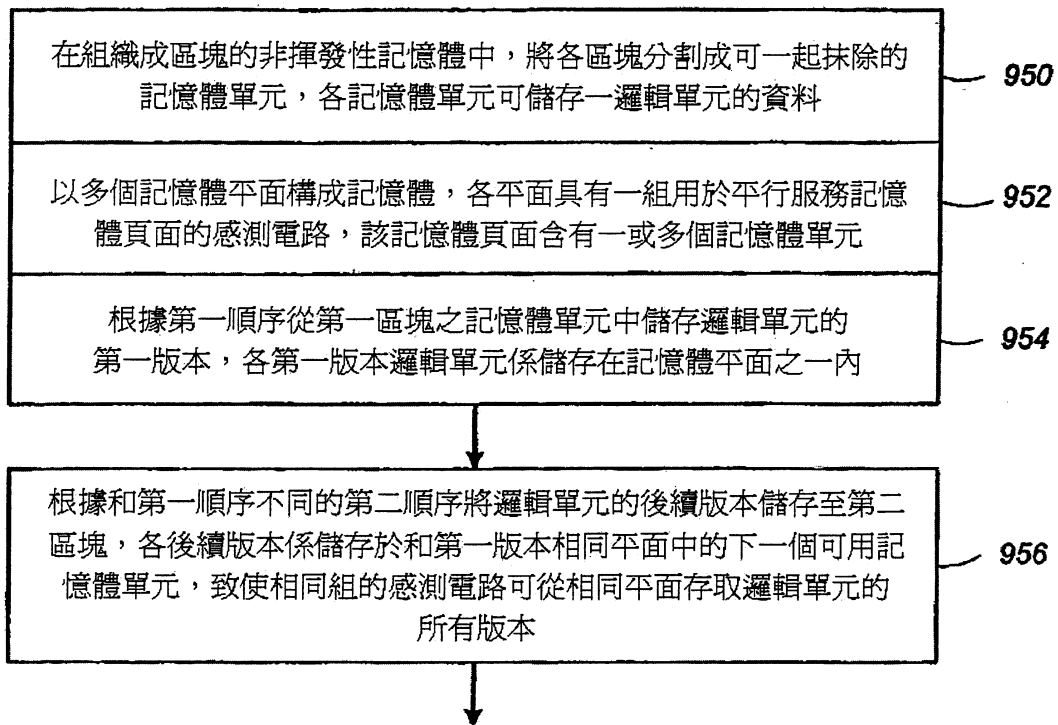


圖 22A

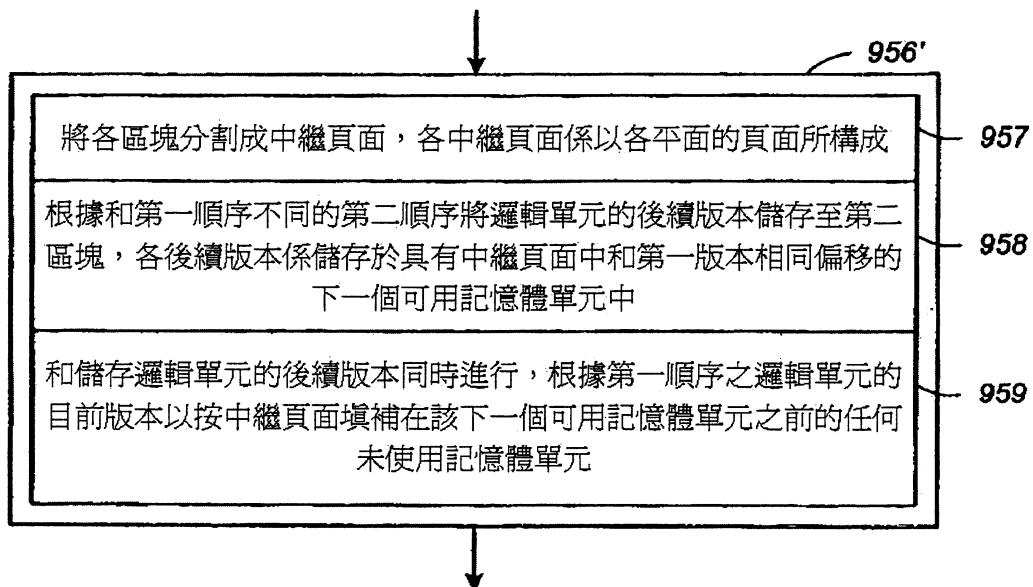


圖 22B

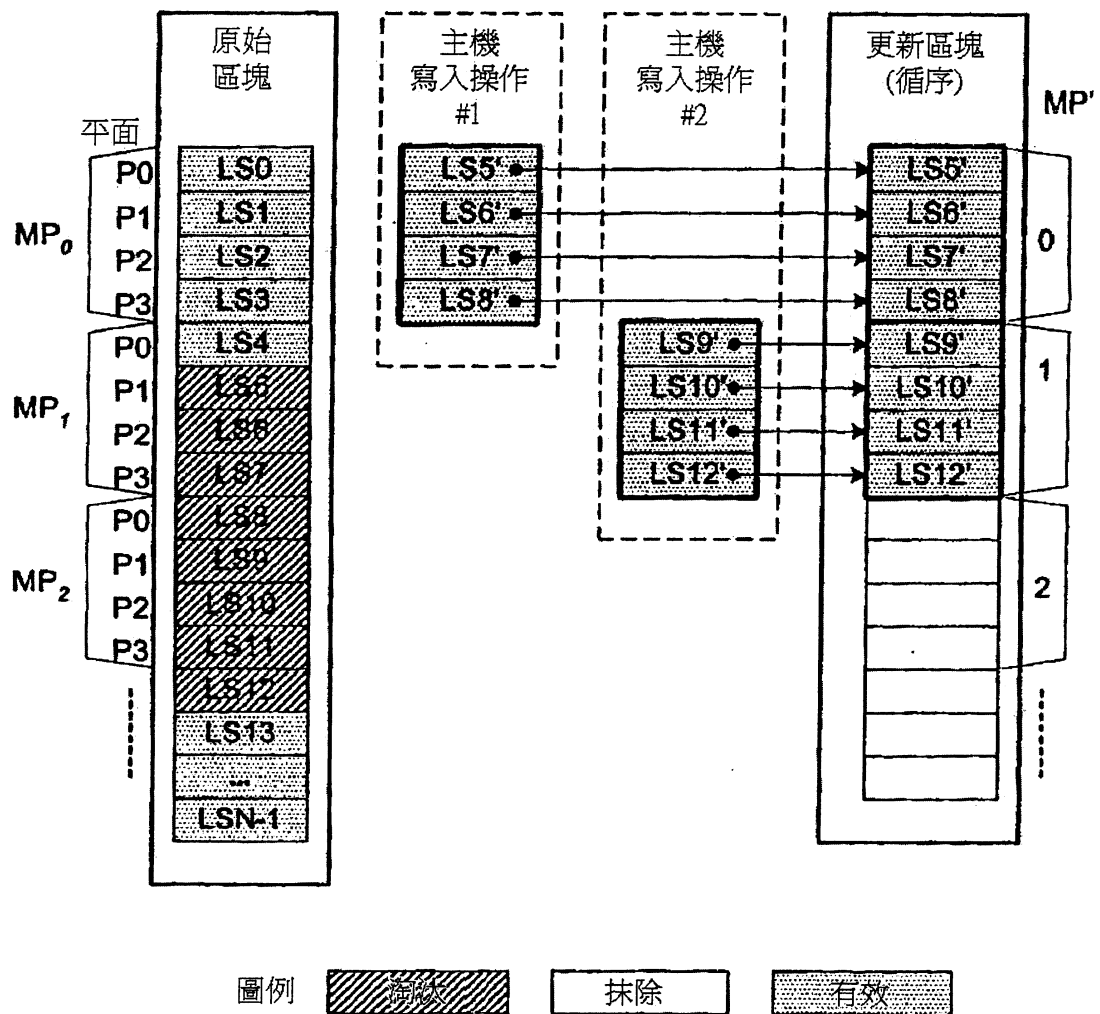


圖 23A

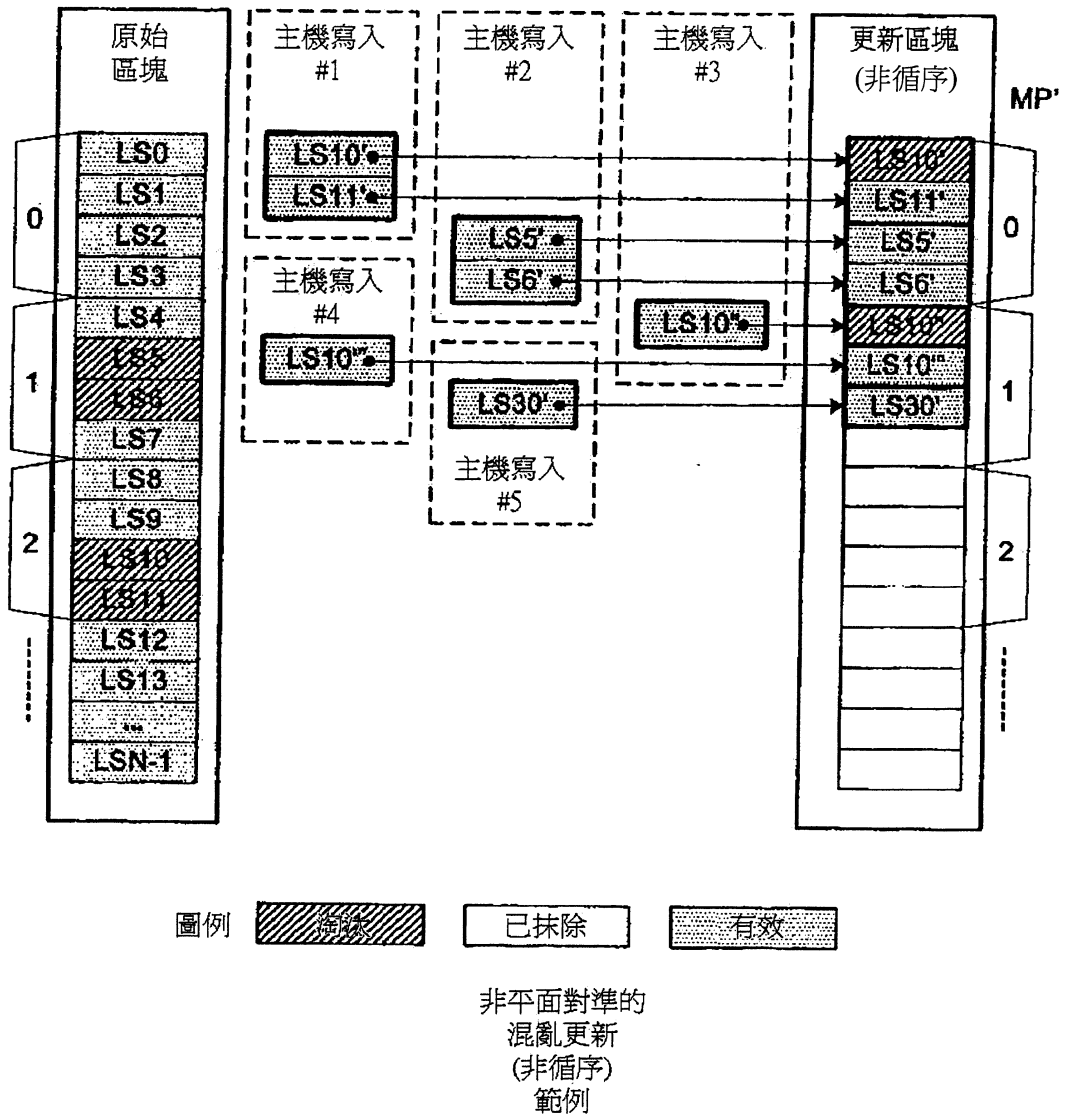


圖 23B

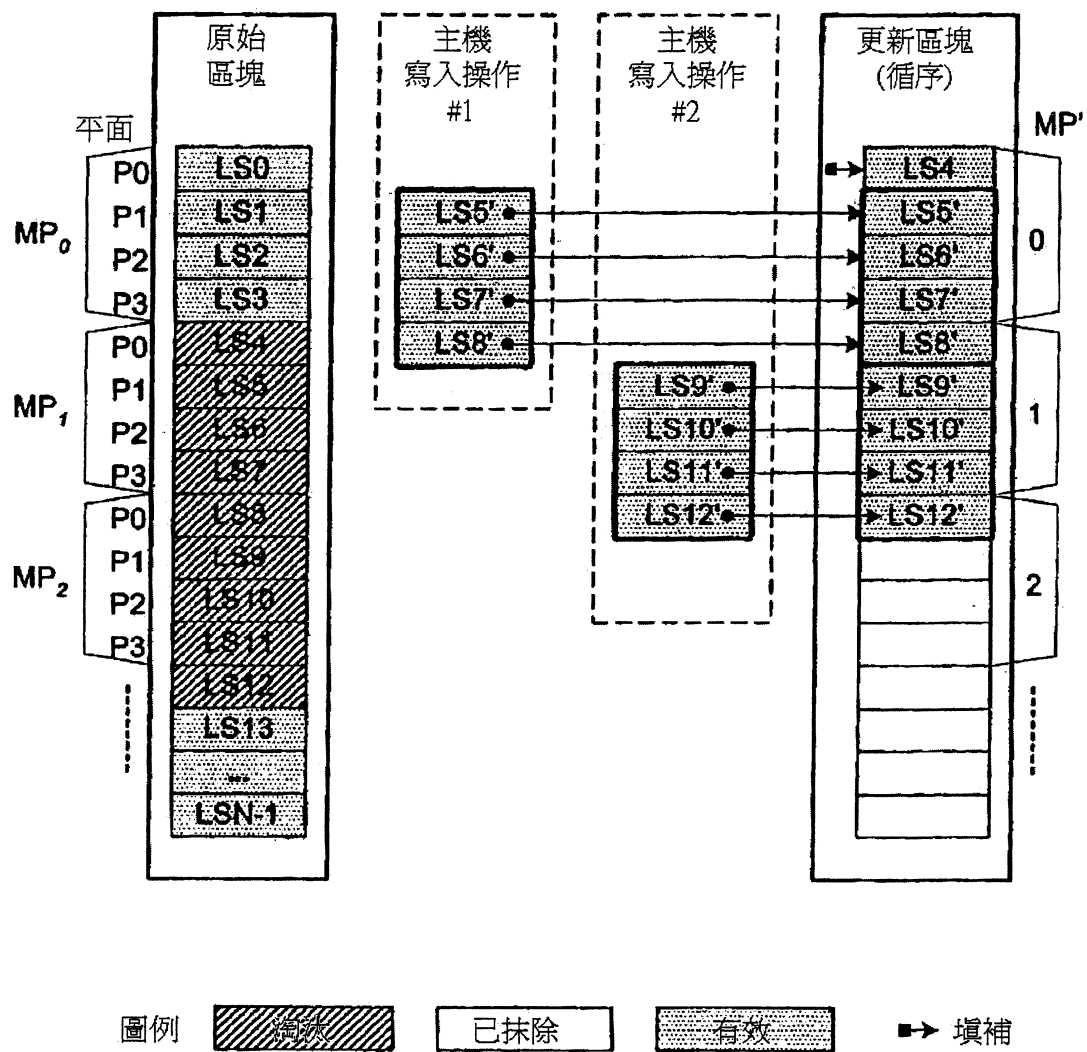
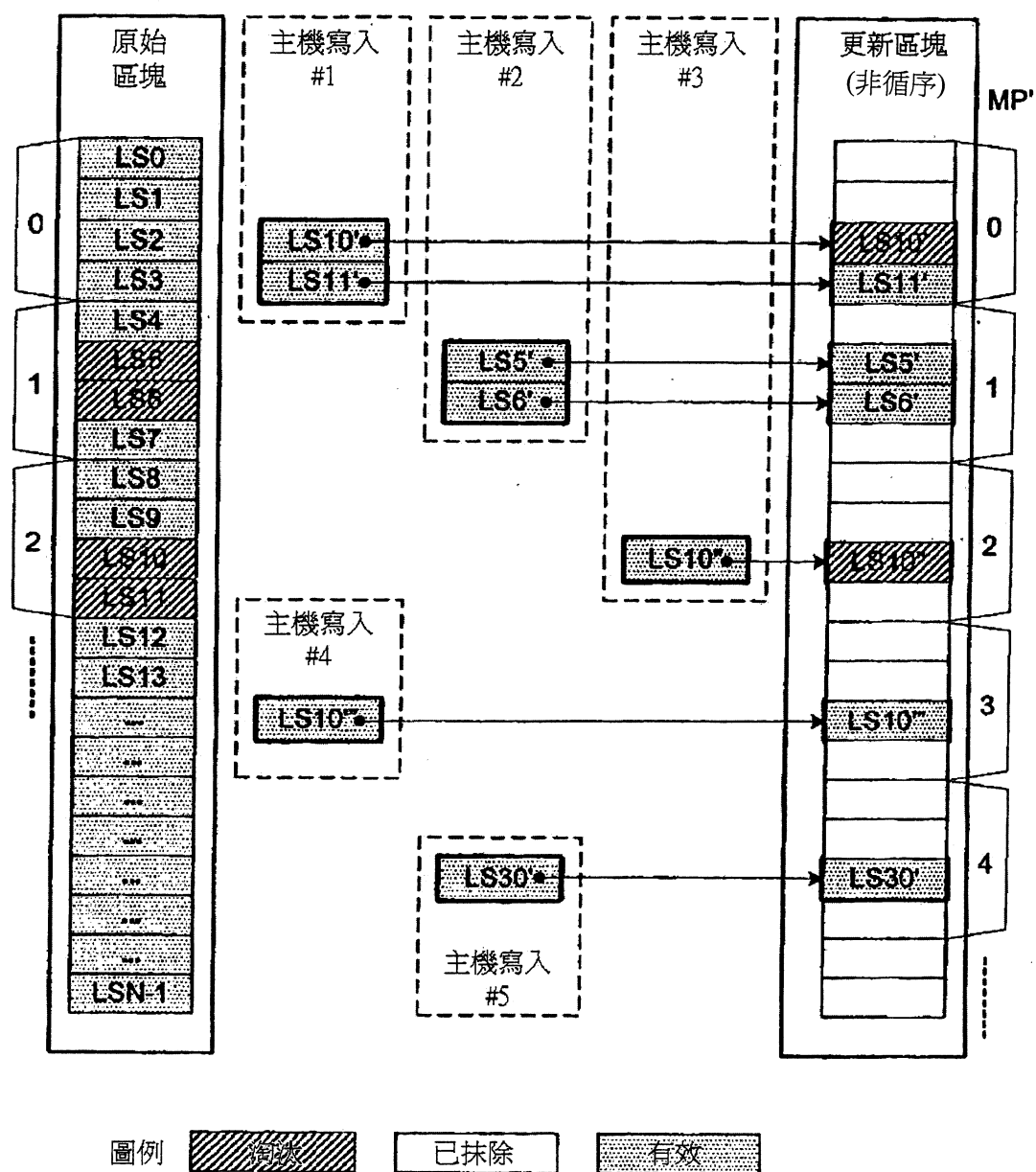
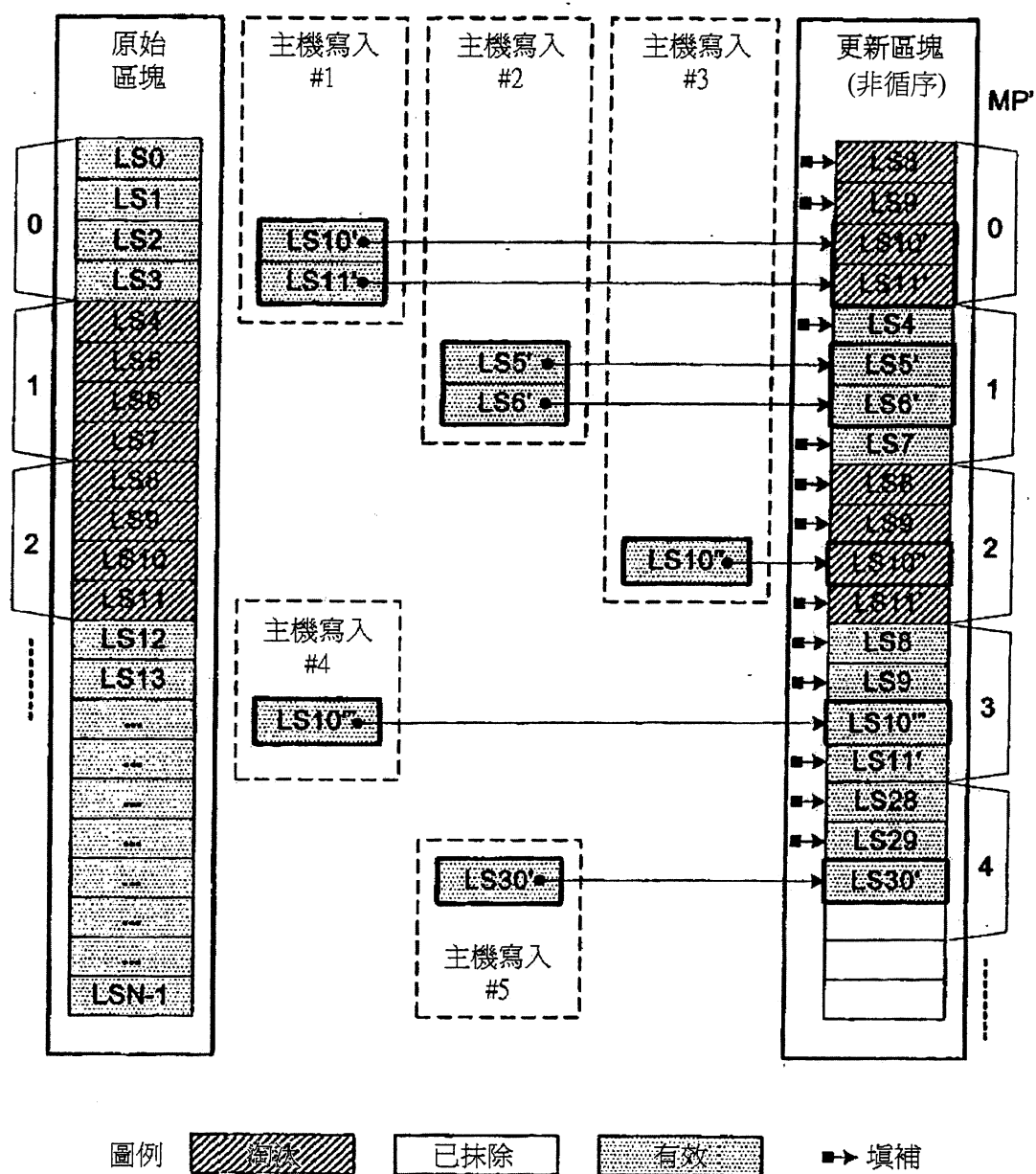


圖 24A



平面對準的
混亂更新
(非循序)
非填補範例

圖 24B



平面對準的
混亂更新
(非循序)
填補範例

圖 24C

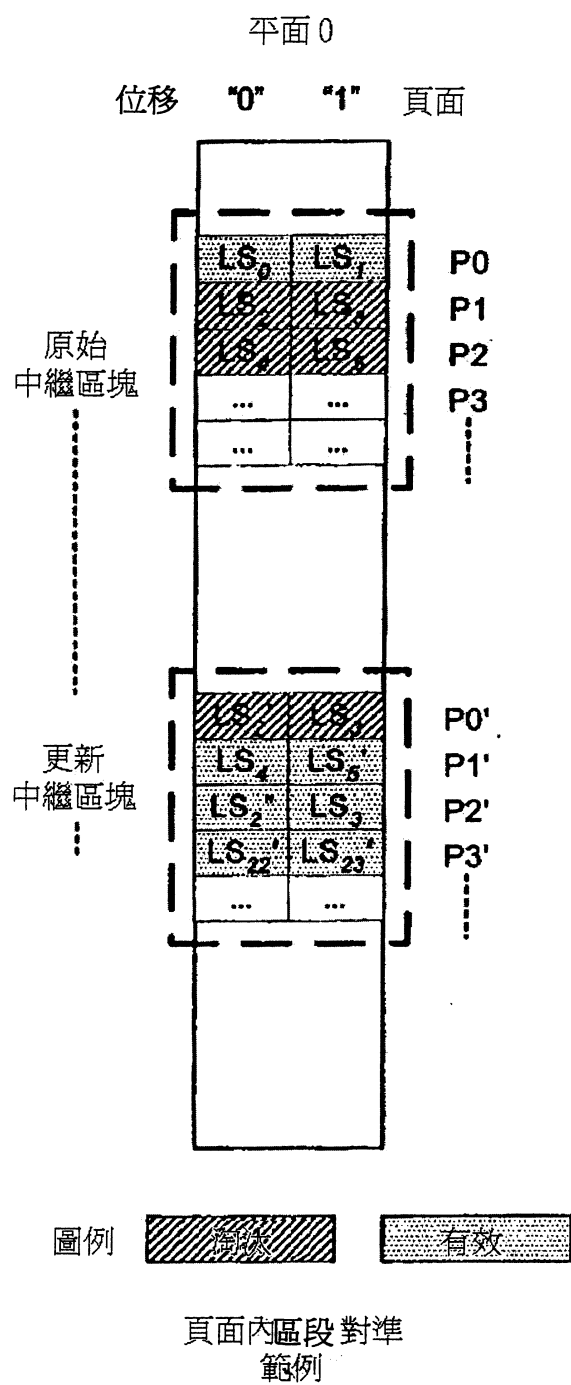
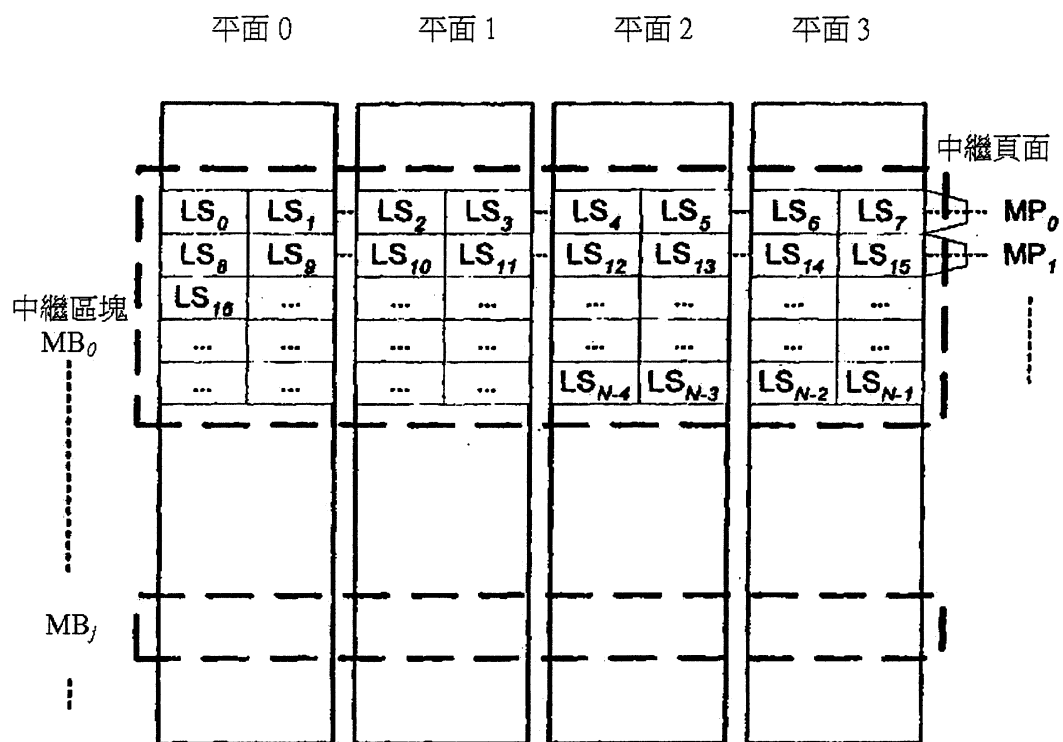


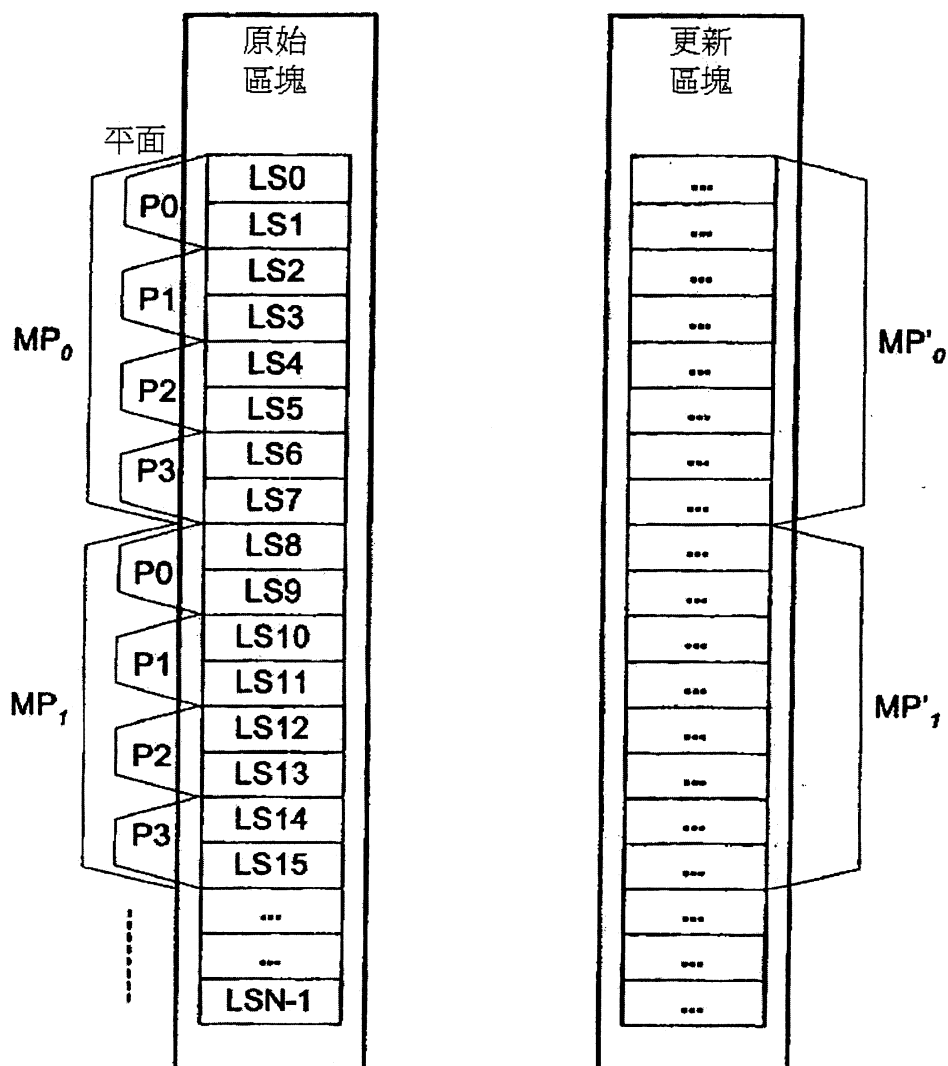
圖 25



LS_k 常駐於平面 x 內
 其中 $x=k \text{ MOD } (W)$ ，
 $k'=\text{INT}(k/K)$ ，
 K (如=2)是平面之頁面中的區段總數，
 W (如=4)是平面總數。

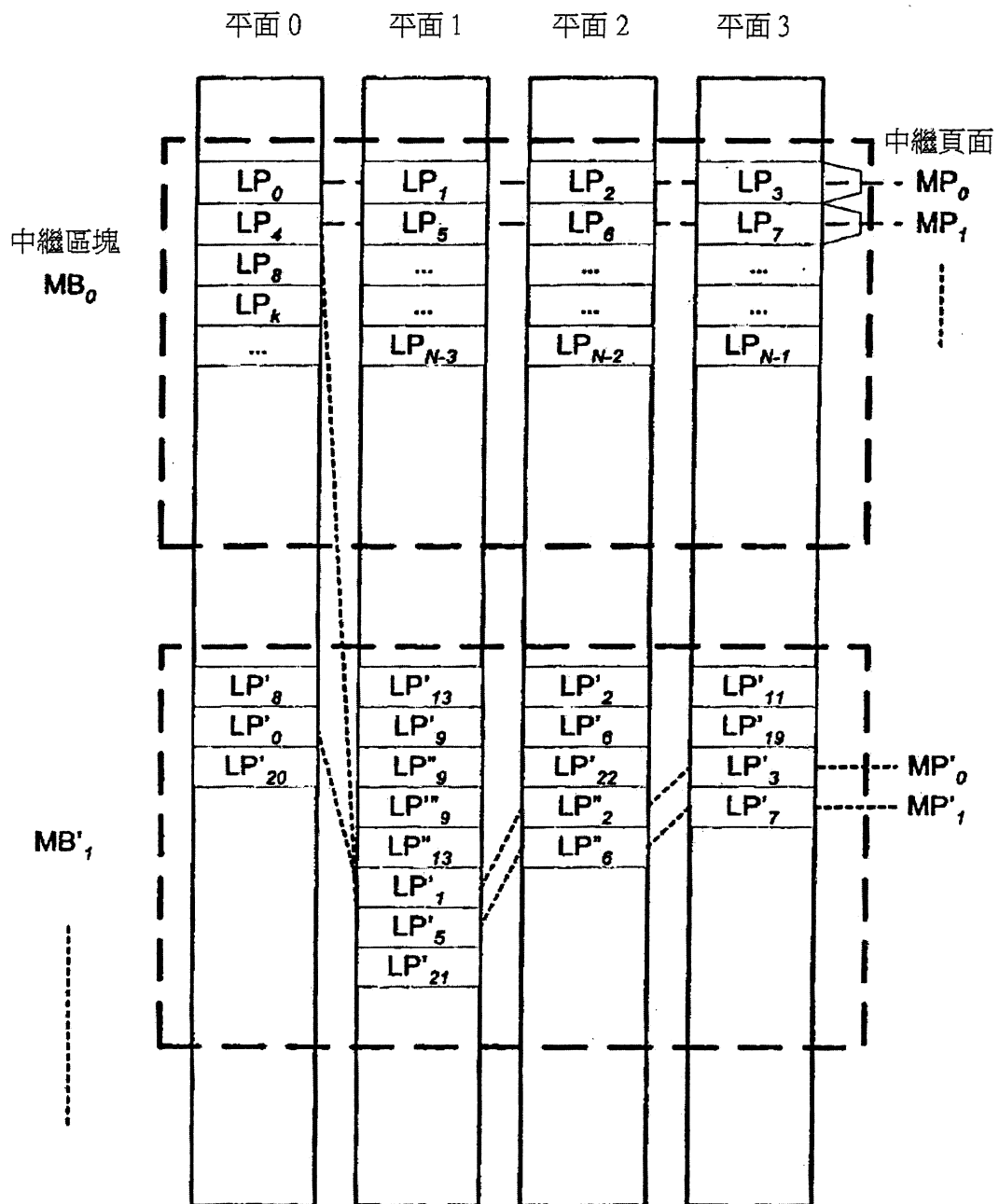
多個區段頁面
 平面對準

圖 26A



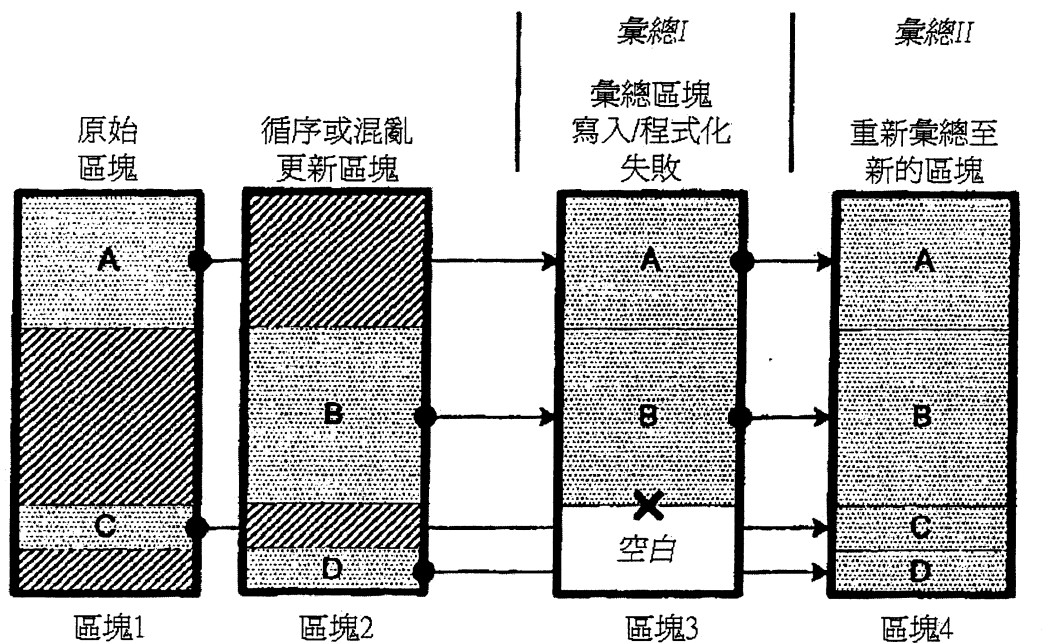
多個區段頁面
平面對準

圖 26B



不用填補的平面對準

圖 27



圖例 有效 已抹除 刪除

利用重新彙總的彙總失敗處置

圖 28

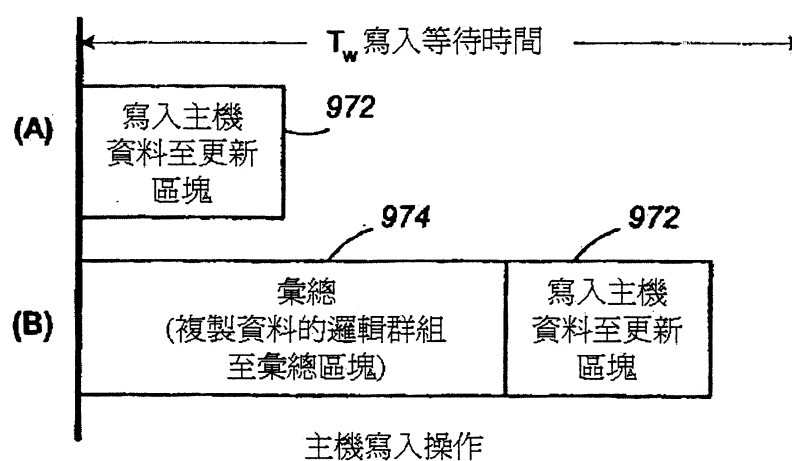


圖 29

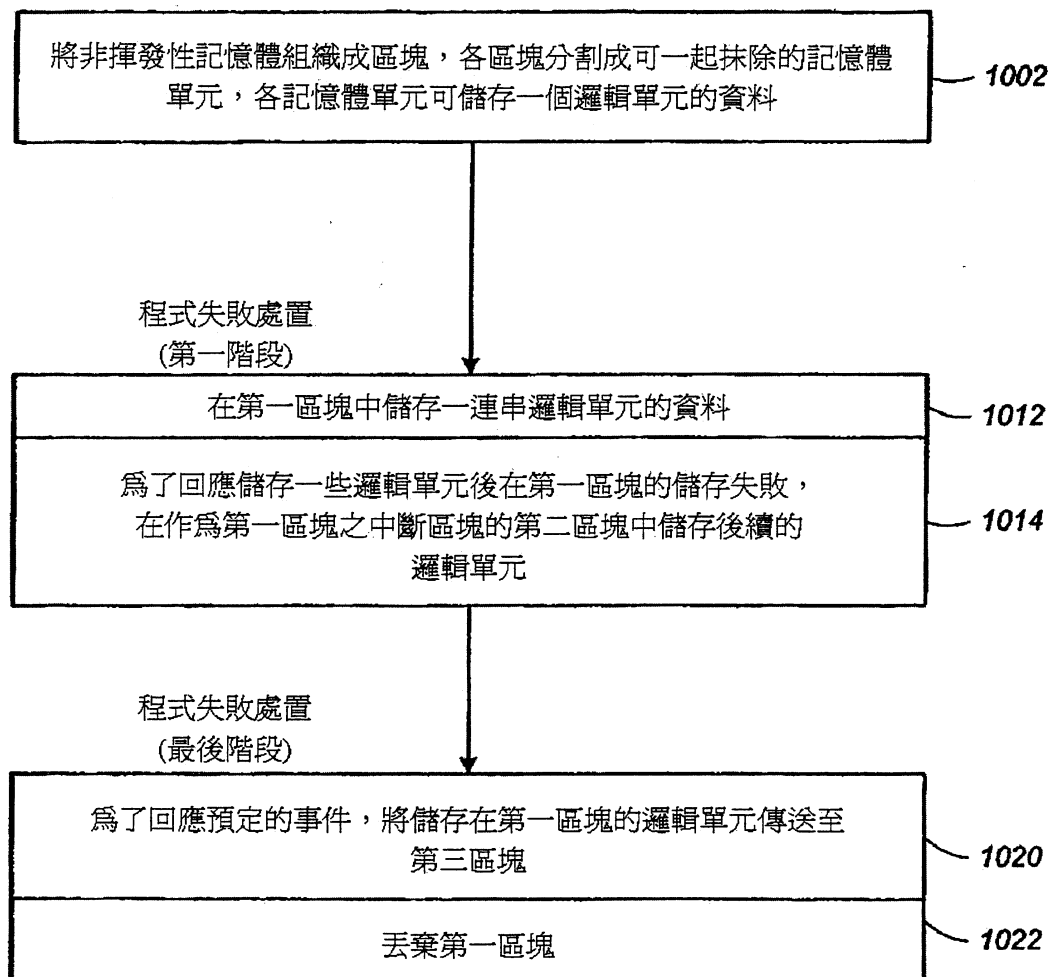
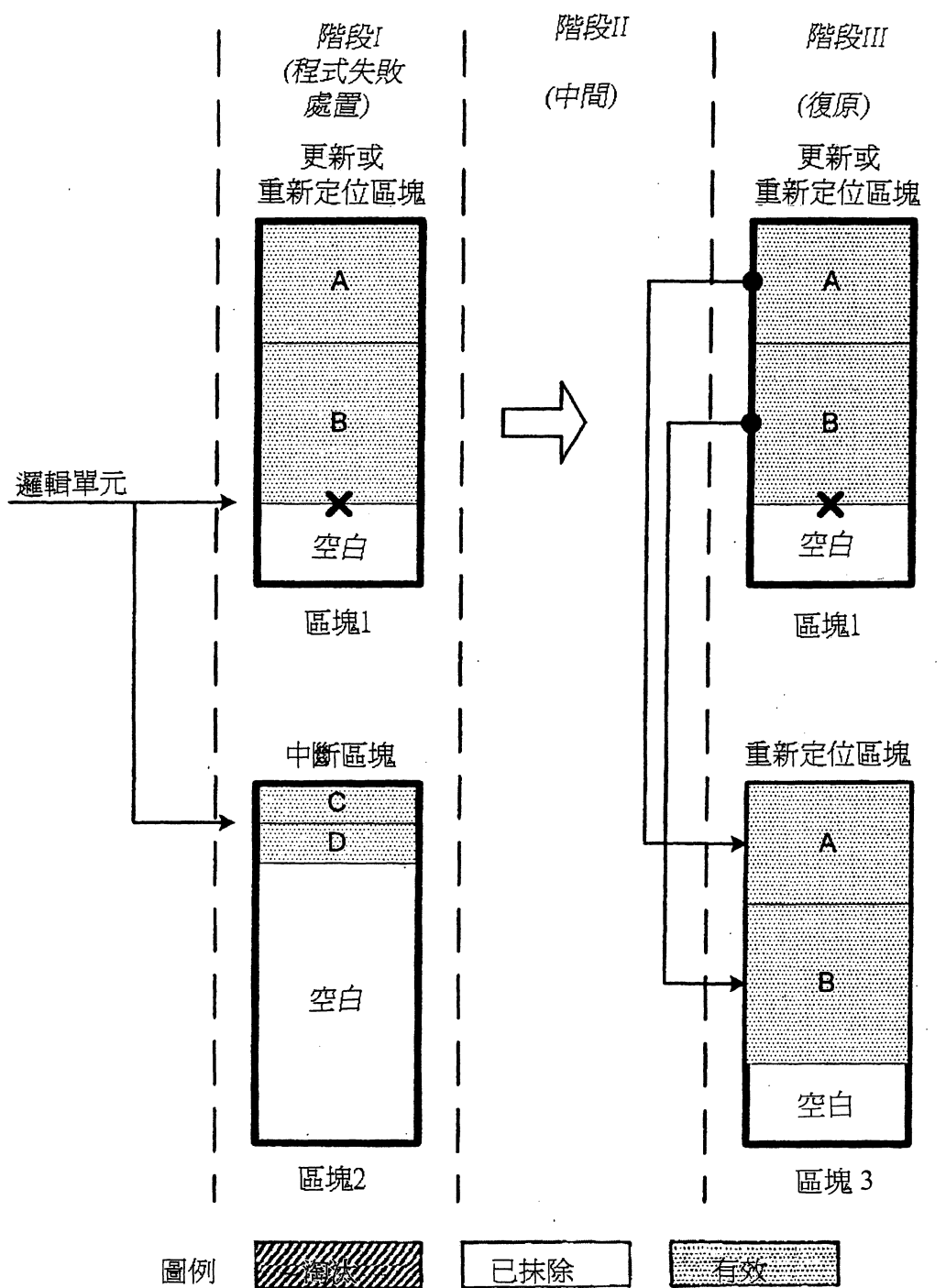
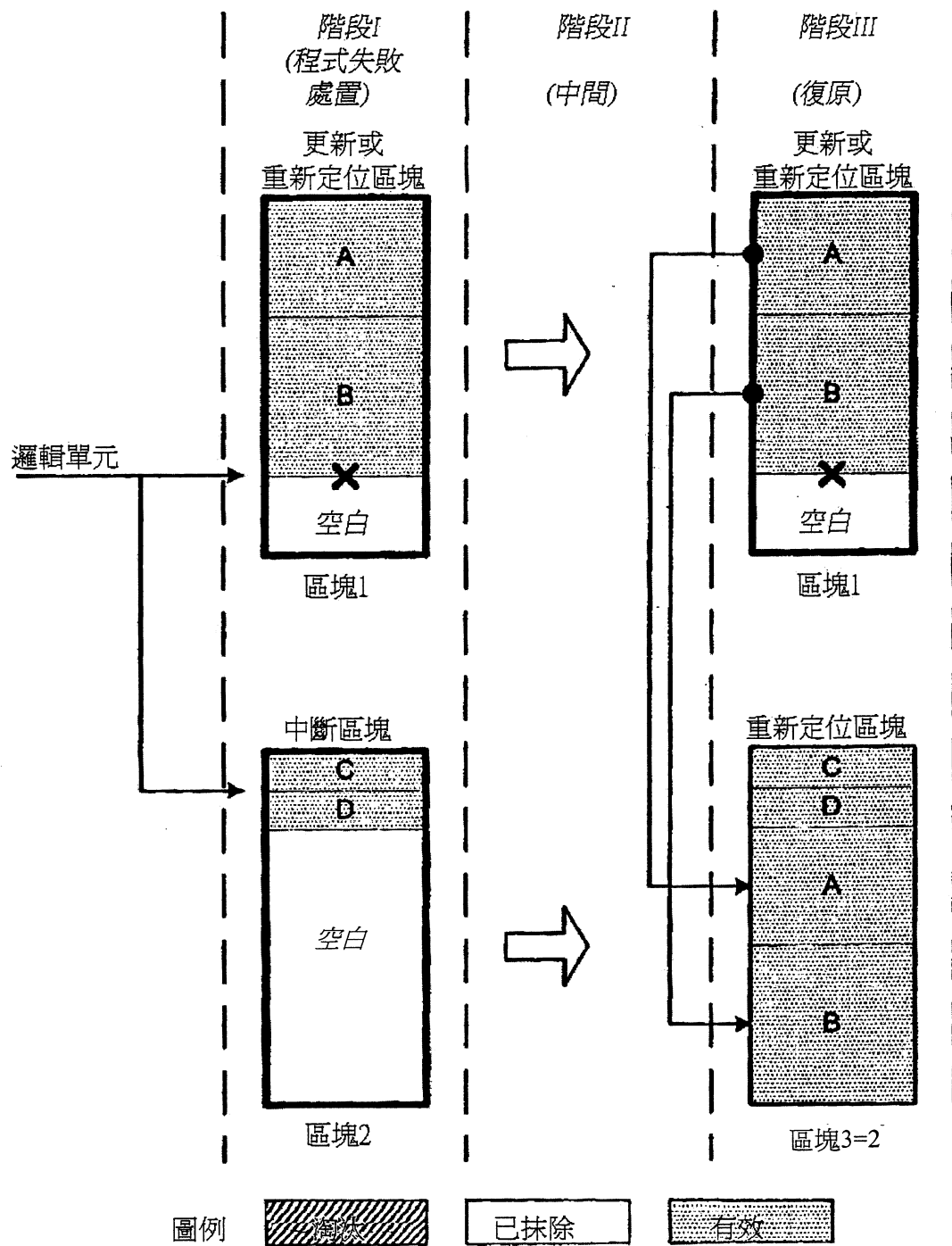


圖 30



程式失敗處置
(案例 1：區塊 3=新)

圖 31A



程式失敗處置
(案例2：區塊3=區塊2)

圖 31B

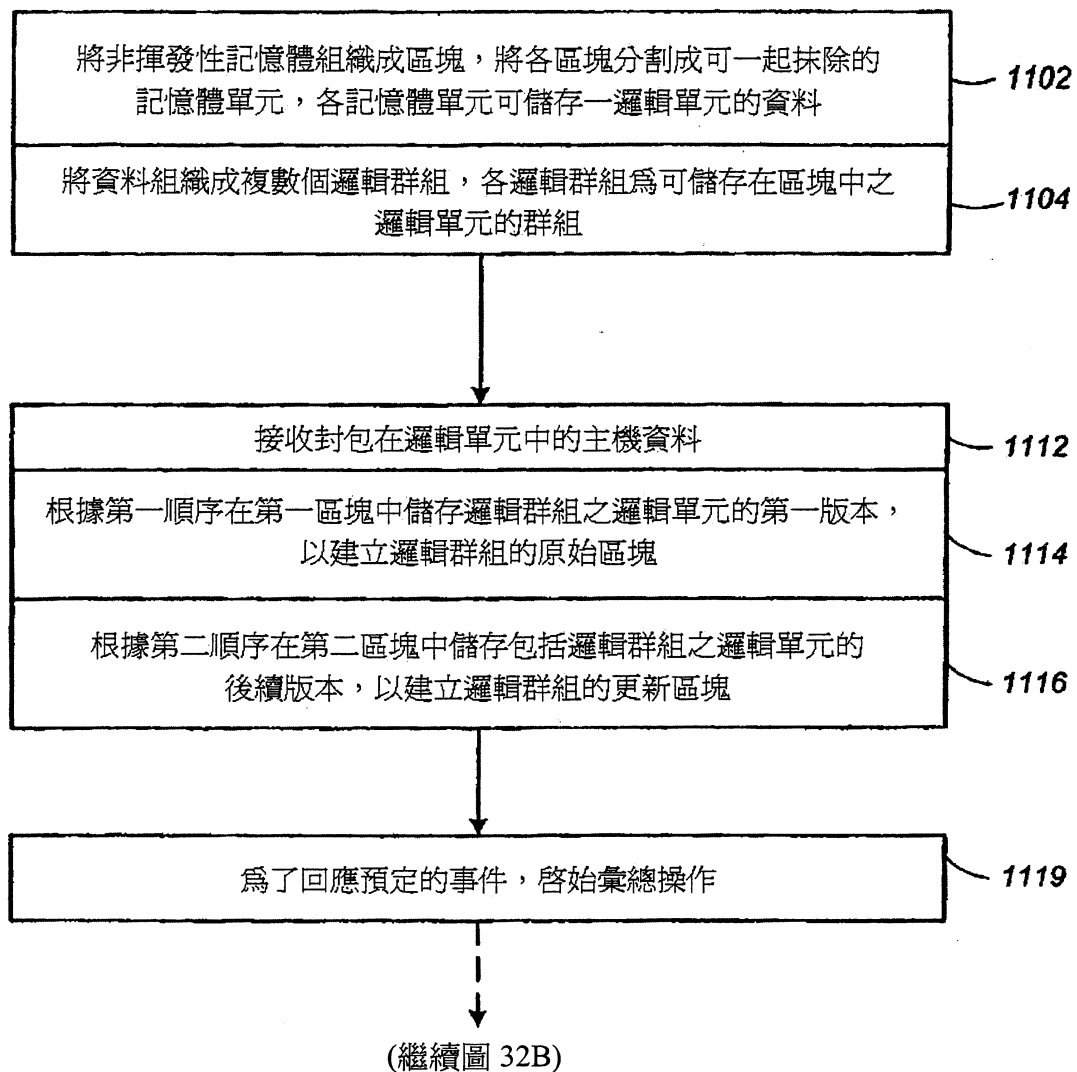


圖 32A

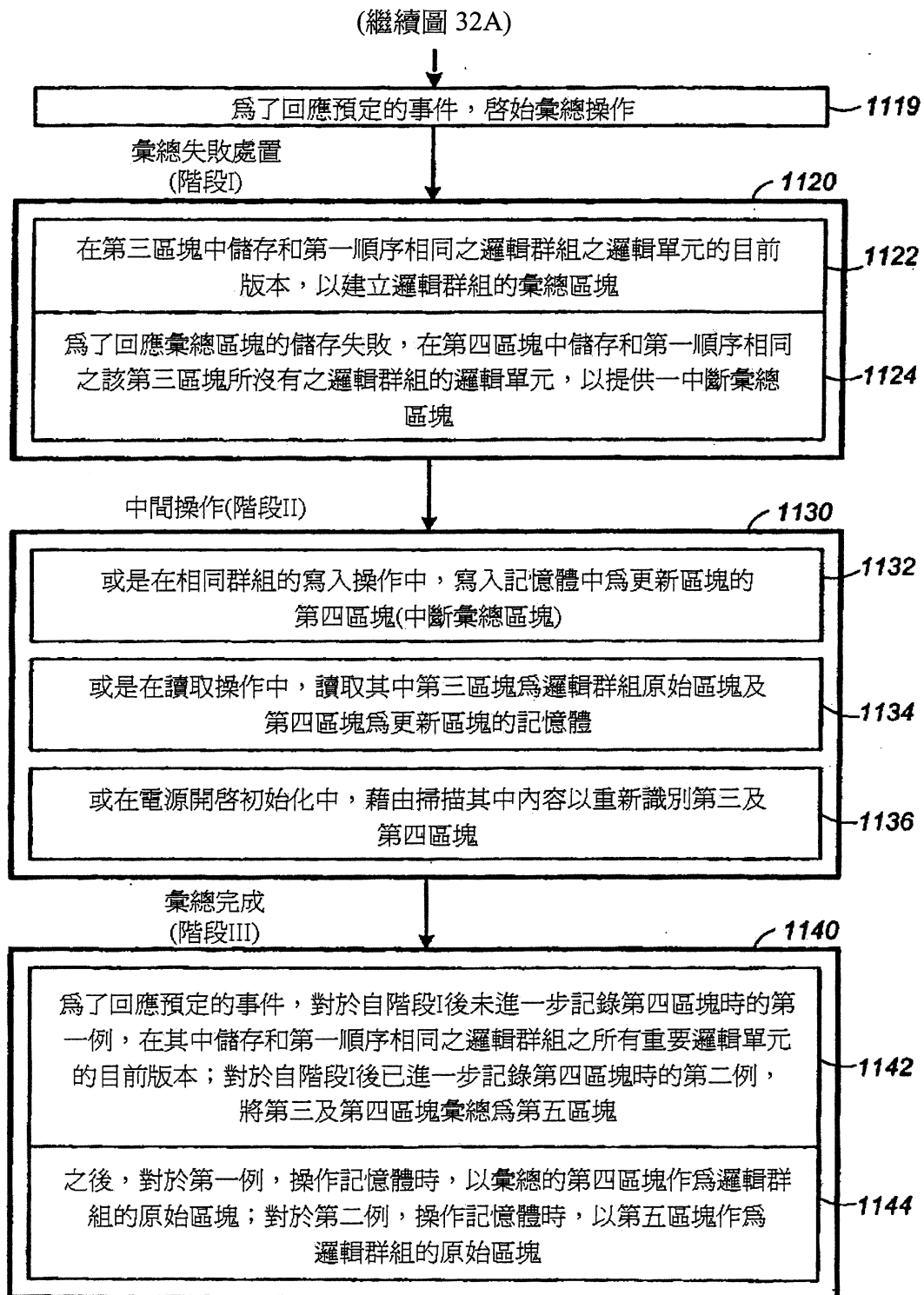


圖 32B

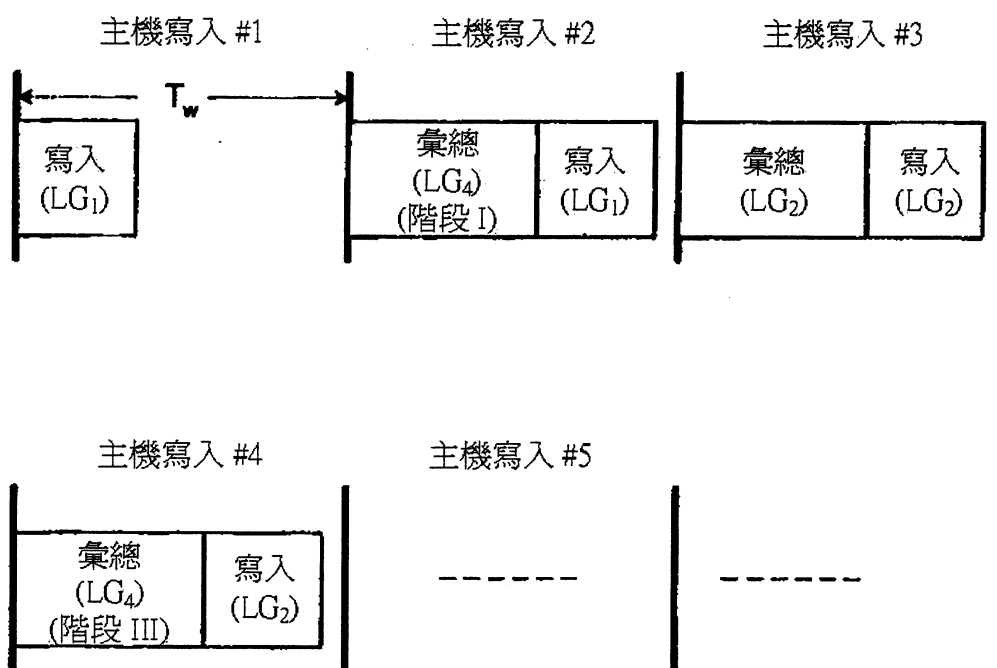
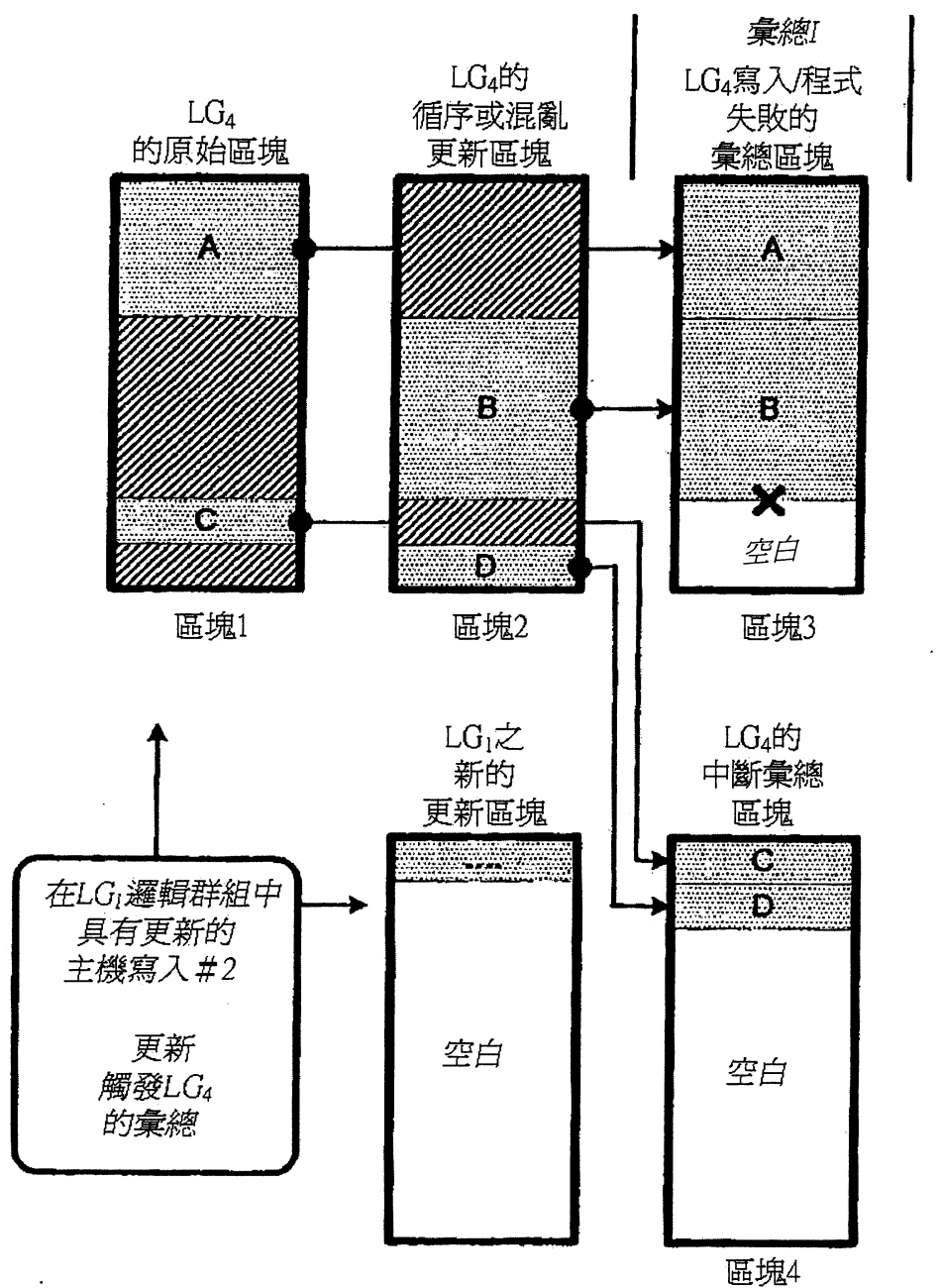


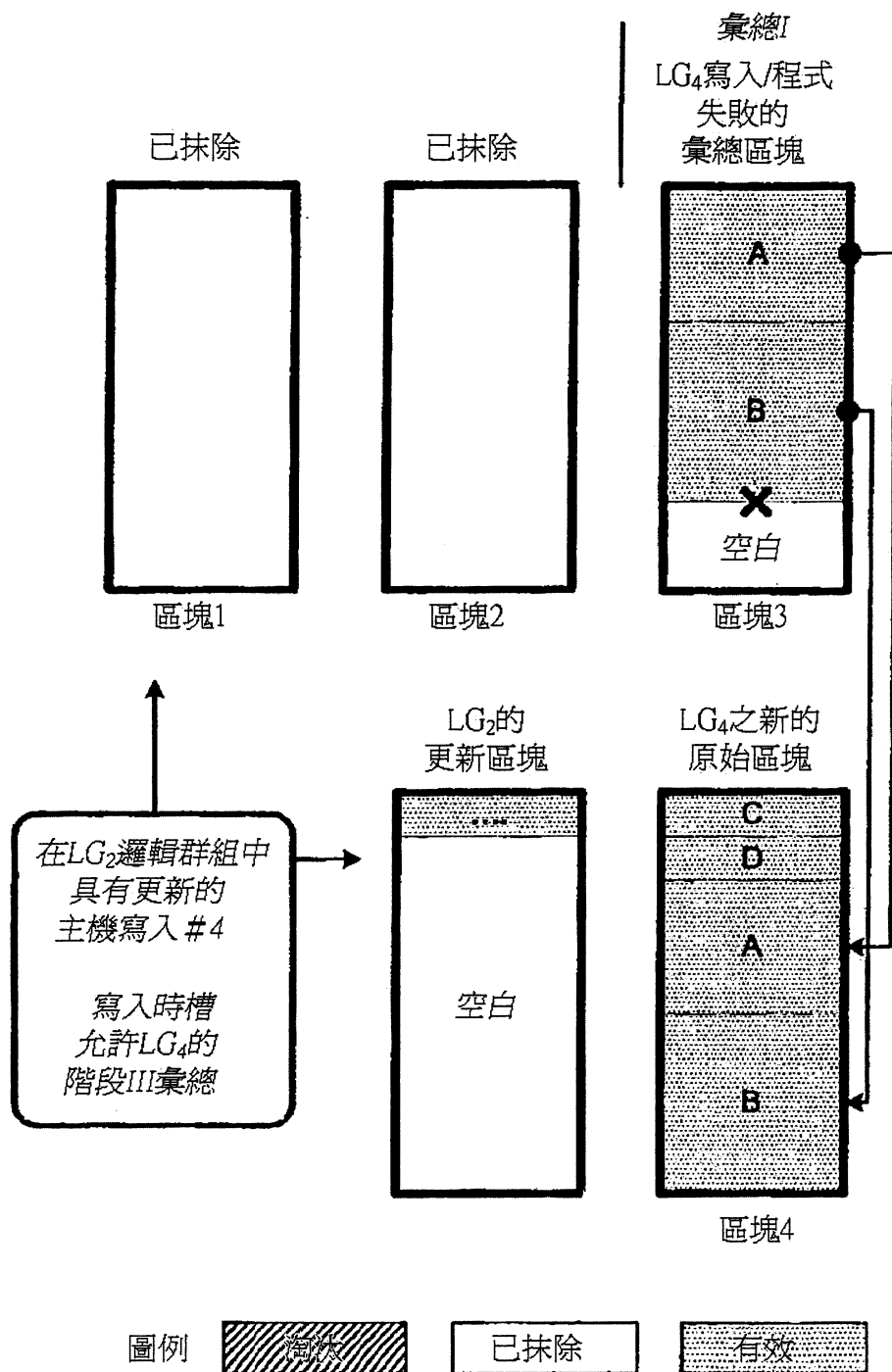
圖 33



圖例 淘汰 已抹除 有效

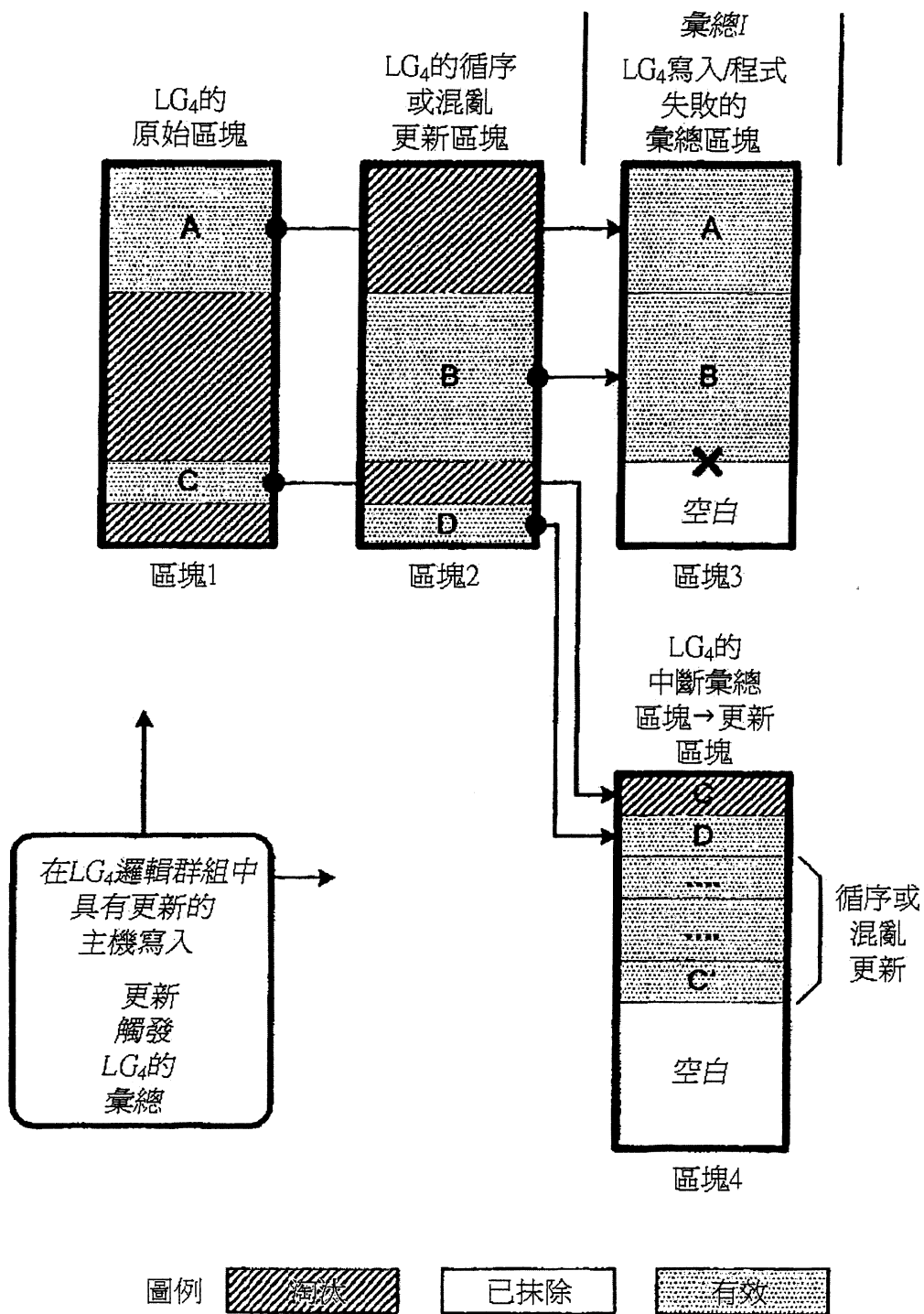
彙總失敗處置(階段I)
(案例1：中斷區塊上沒有額外更新)

圖 34A



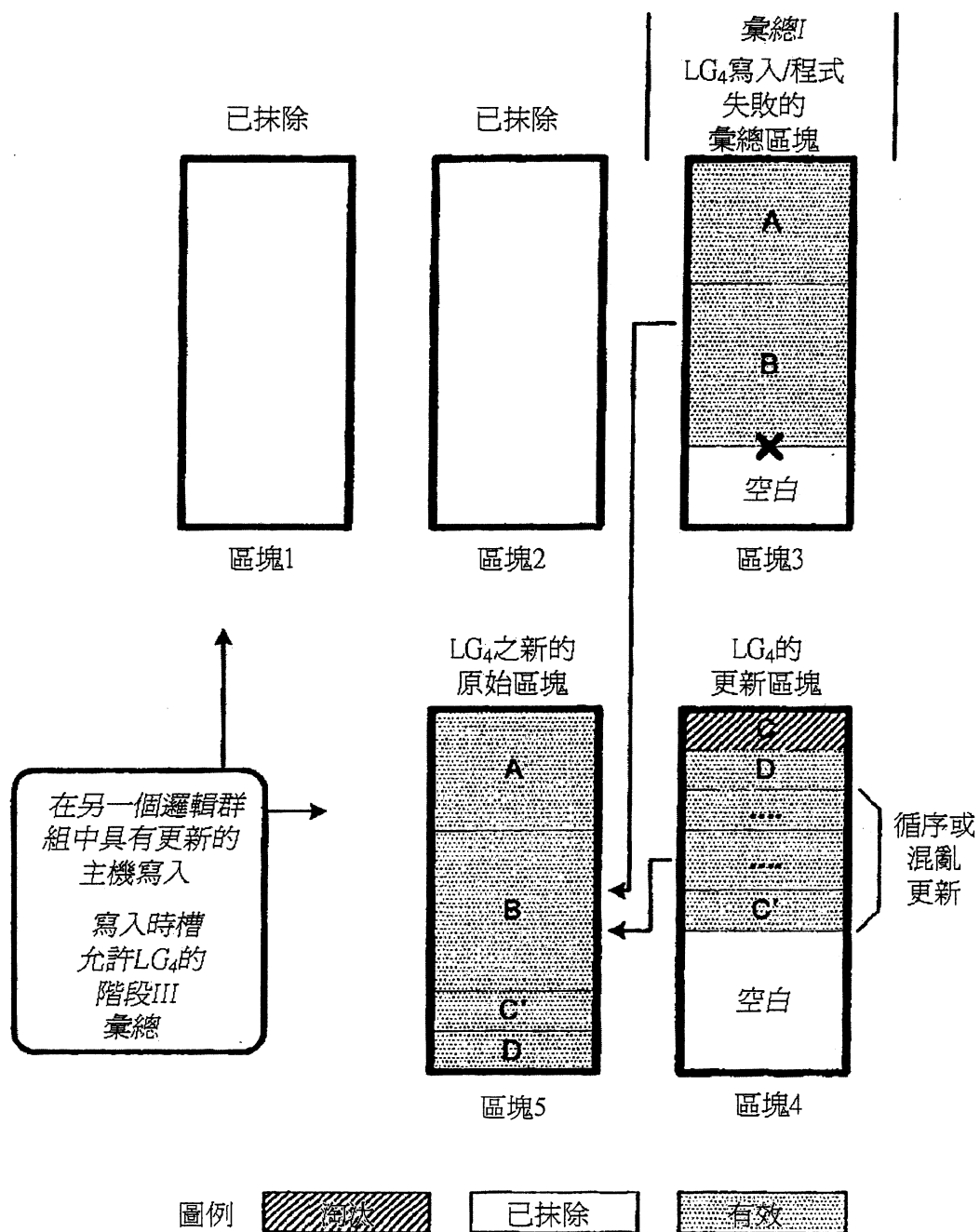
彙總失敗處置(階段III) (案例1：中斷區塊上沒有額外更新)

圖 34B



彙總失敗處置(階段I)
(案例2：中斷區塊上的額外更新)

圖 35A



彙總失敗處置(階段III) (案例2：中斷區塊上的額外更新)

圖 35B

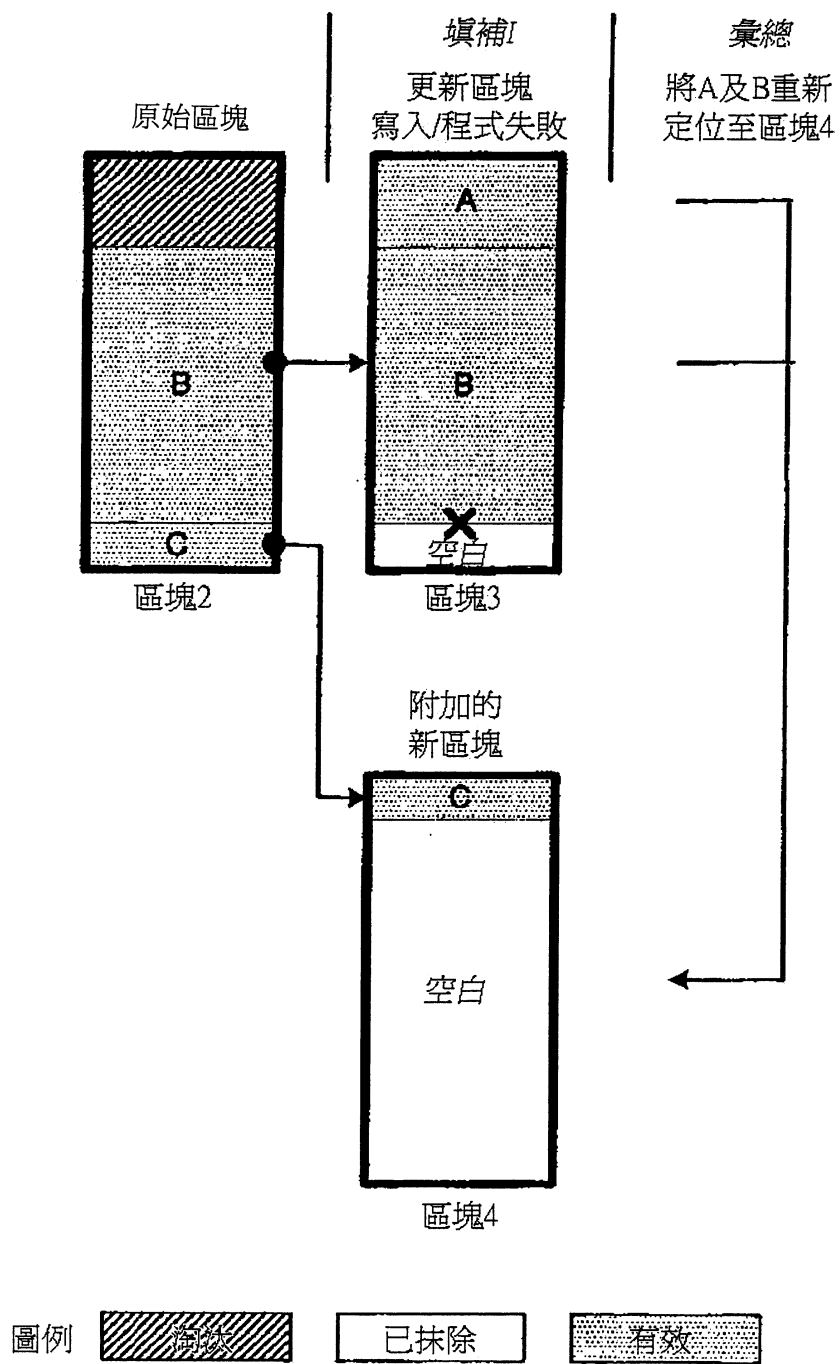
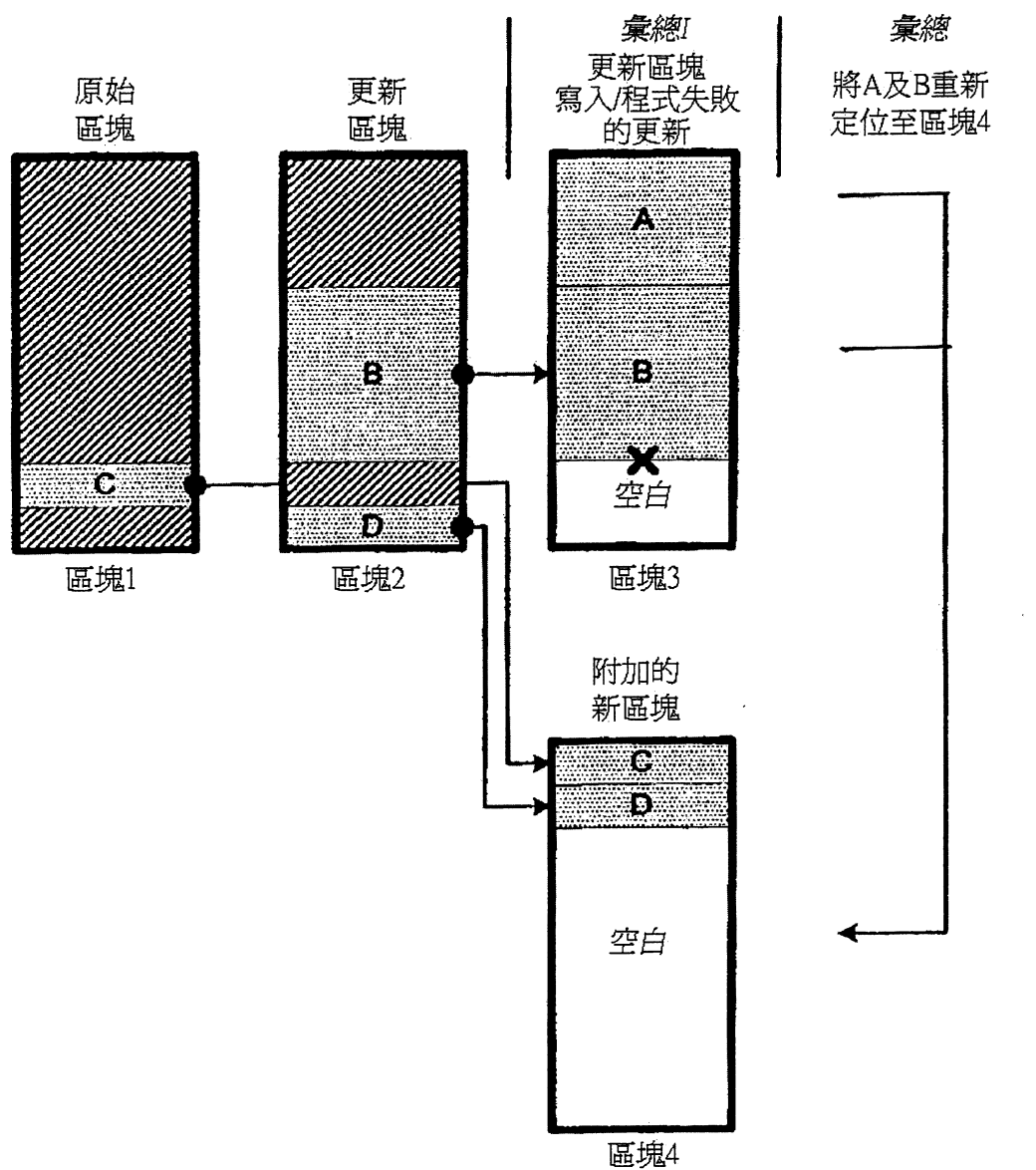


圖 36A



圖例

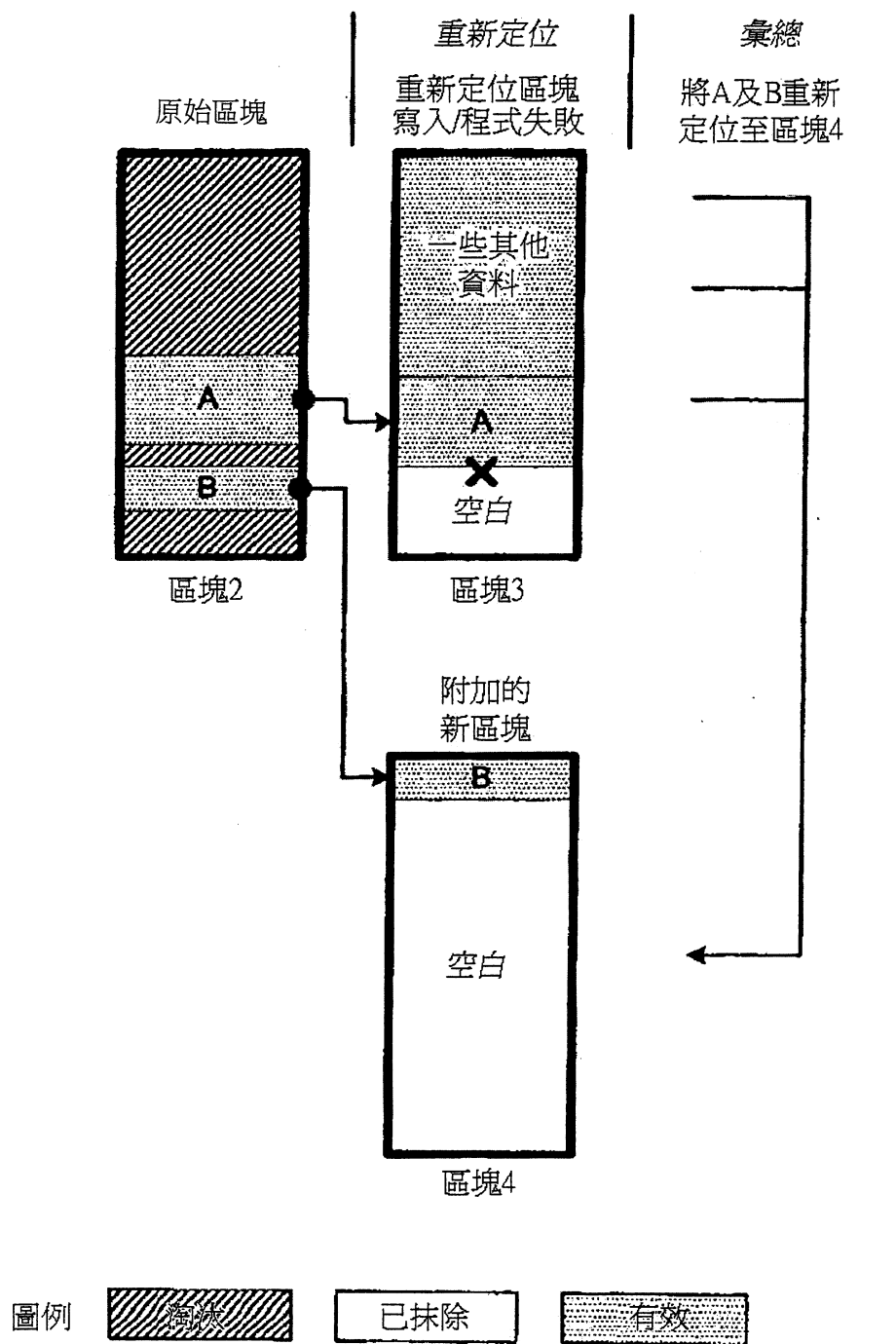
淘汰

已抹除

有效

更新區塊失敗處置之更新的彙總(階段I及III)

圖 36B



重新定位失敗處置
(沒有邏輯分組)
(階段I及III)

圖 36C

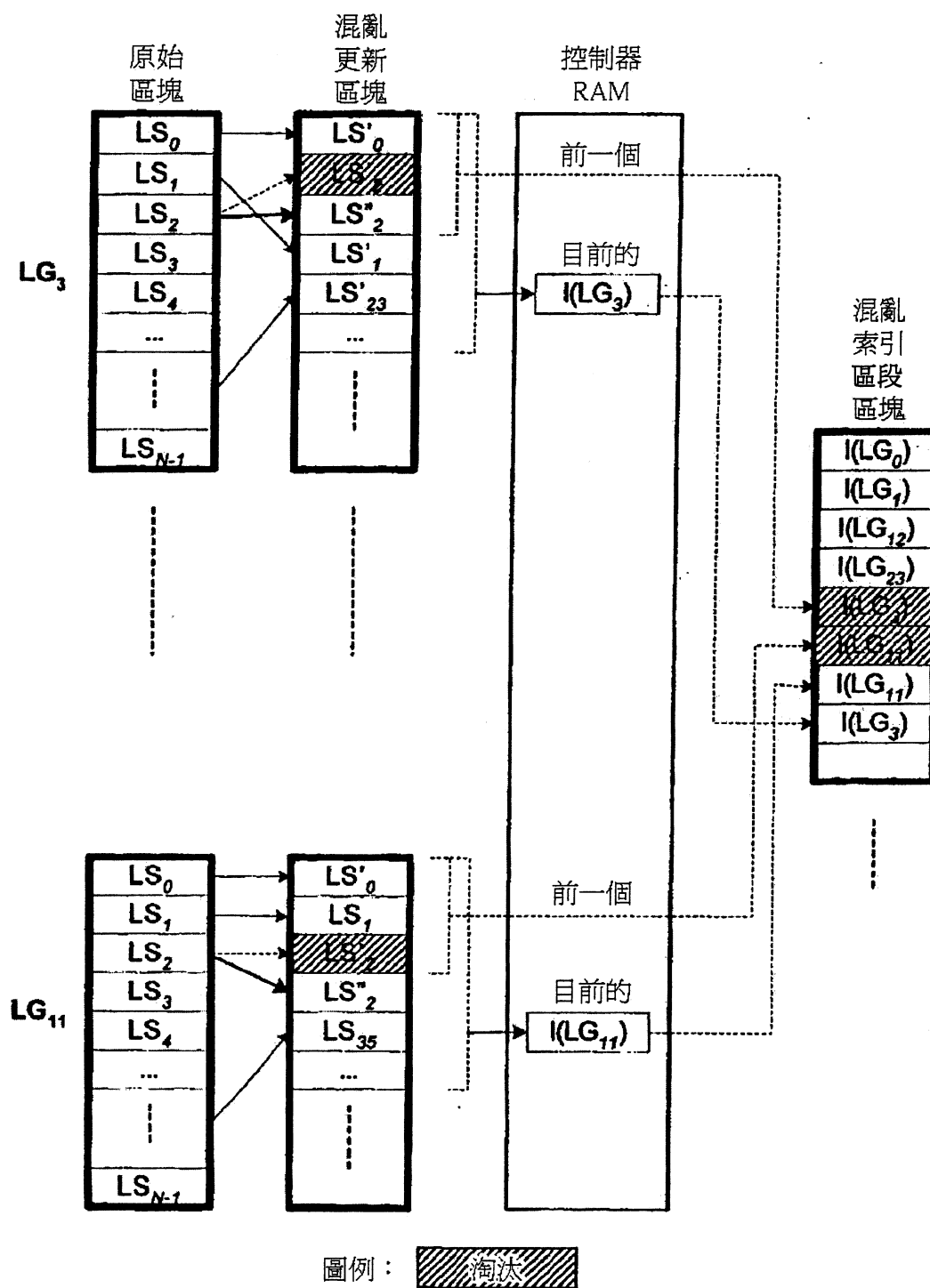
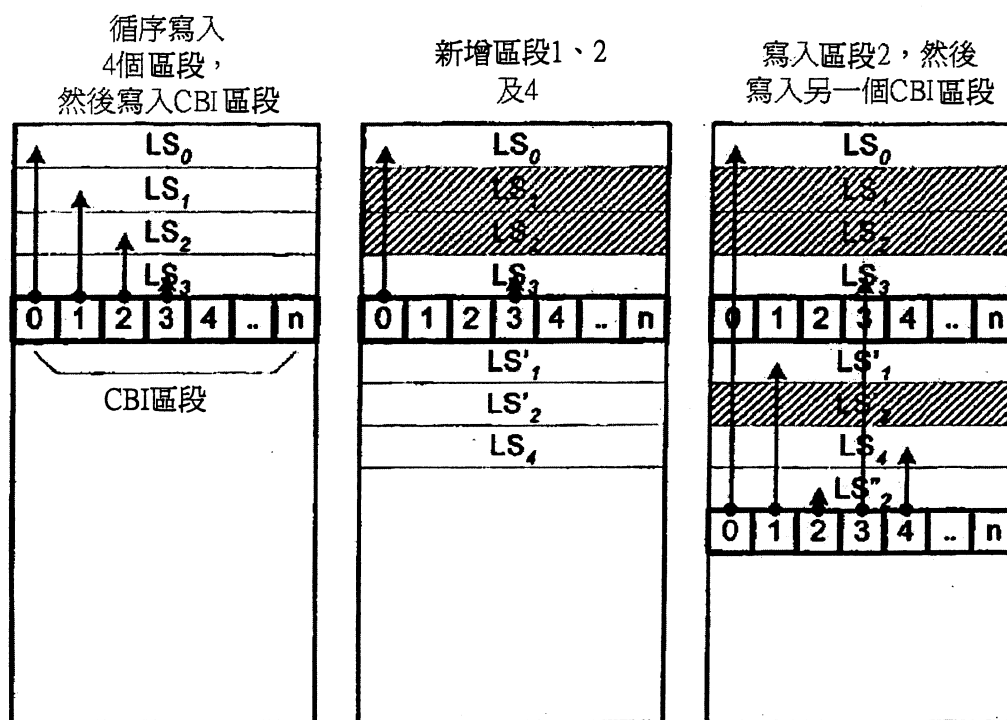


圖 37



圖例：  淘汰

圖 38A

圖 38B

圖 38C



圖 40

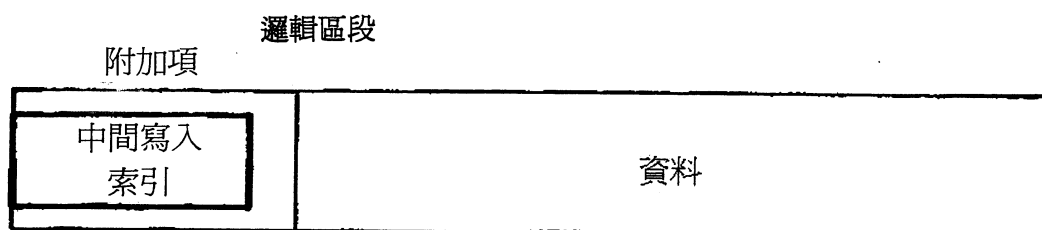
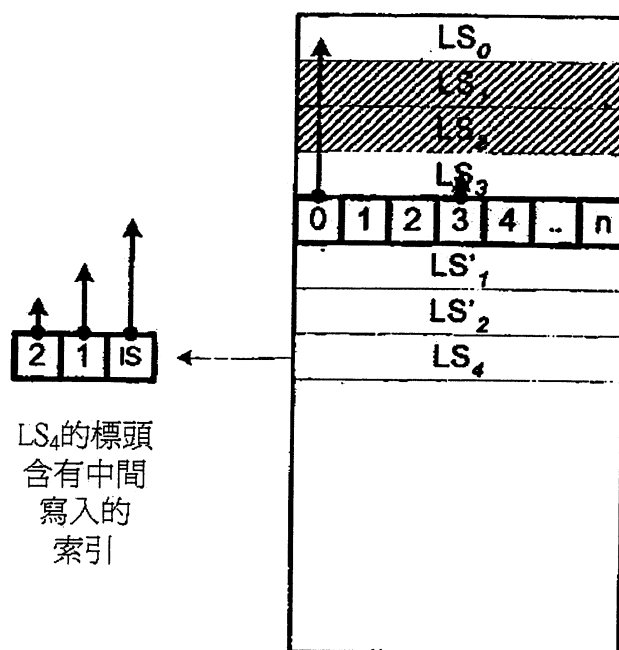


圖 39A



圖例： 淘汰

圖 39B

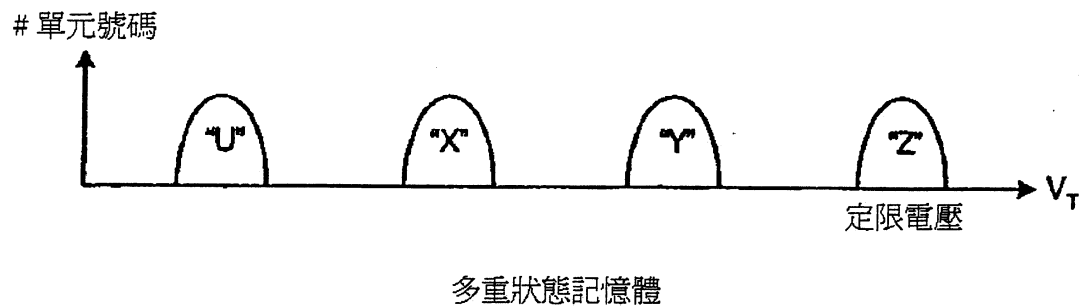
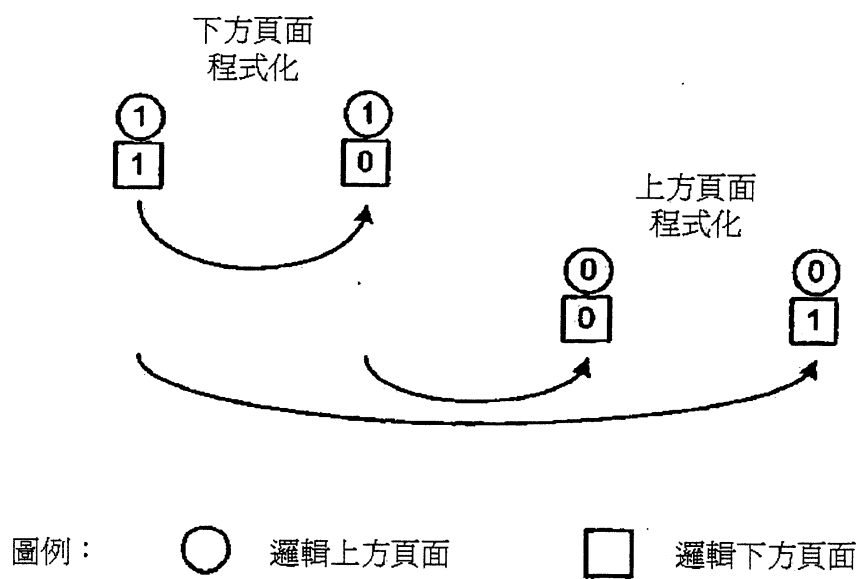
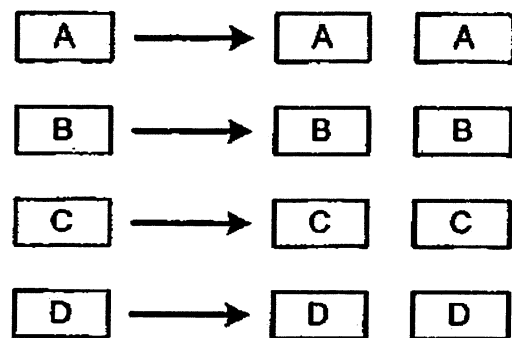


圖 41A



具格雷碼的二次編碼
過程程式化

圖 41B



要複製的區段

圖 42

多重狀態頁面

		D	D
A	A	B	B

上方頁面

下方頁面

第二程式

第一程式

複製頁面中區段的非健全配置

圖 43

多重狀態頁面

~~C~~	D	~~C~~	D	上方頁面	第二程式
A	A	B	B	下方頁面	第一程式

複製頁面中區段的非健全配置

圖 44A

多重狀態頁面

A	A	B	B	上方頁面	第二程式
x	x	x	x	下方頁面	第一程式

C	C	D	D	上方頁面	第二程式
x	x	x	x	下方頁面	第一程式

複製頁面中區段的非健全配置

圖 44B

多重狀態頁面

A	A	B	B	頁面	二進制 程式
---	---	---	---	----	-----------

C	C	D	D	頁面	二進制 程式
---	---	---	---	----	-----------

二進制程式模式的複製區段

圖 44C

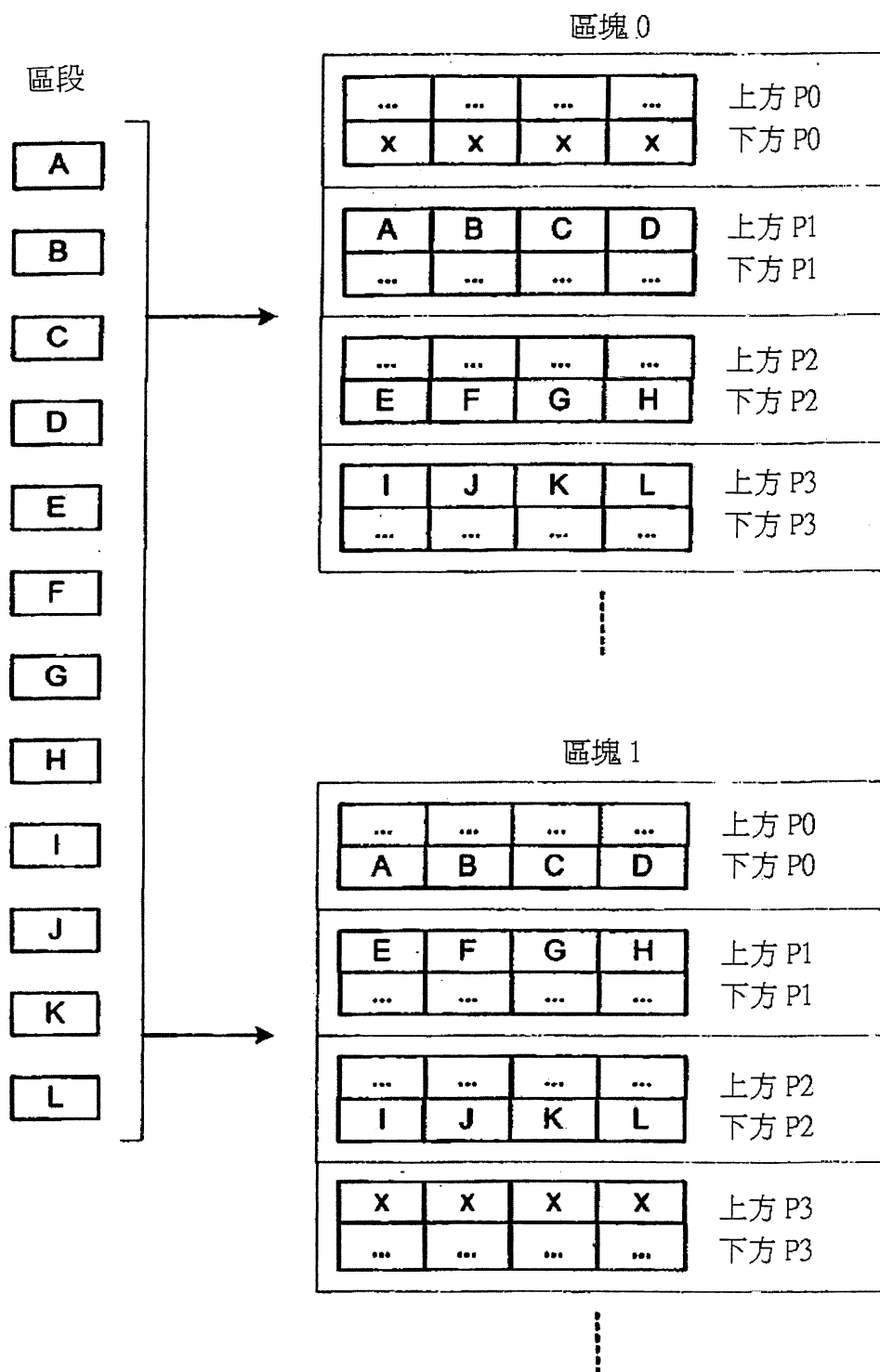
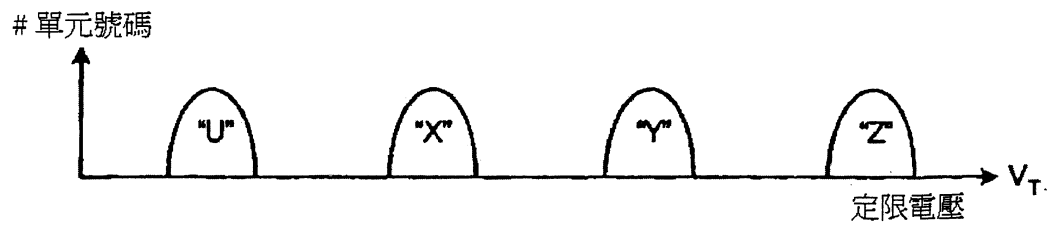
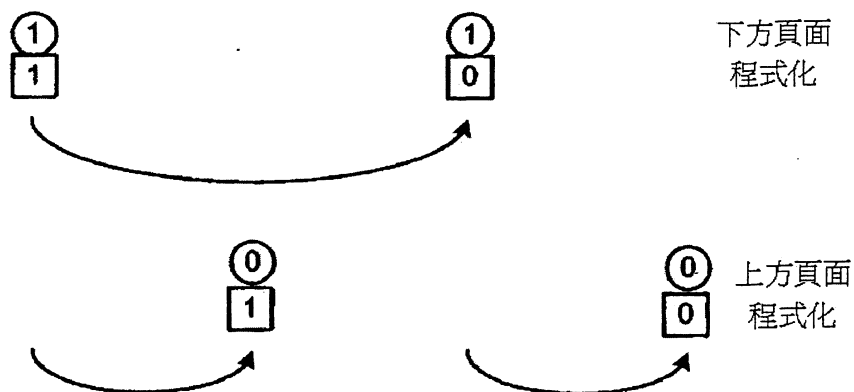


圖 45



多重狀態記憶體

圖 46A



圖例： ○ 邏輯上方頁面 □ 邏輯下方頁面

具容錯碼的二次編碼
過程程式化

圖 46B

狀態	備份1	備份2	有效備份
1	ECC良好	ECC良好	備份1及2
2	ECC良好	ECC不良	備份1
3	ECC良好	空白	備份1
4	ECC不良	空白	無

圖 47

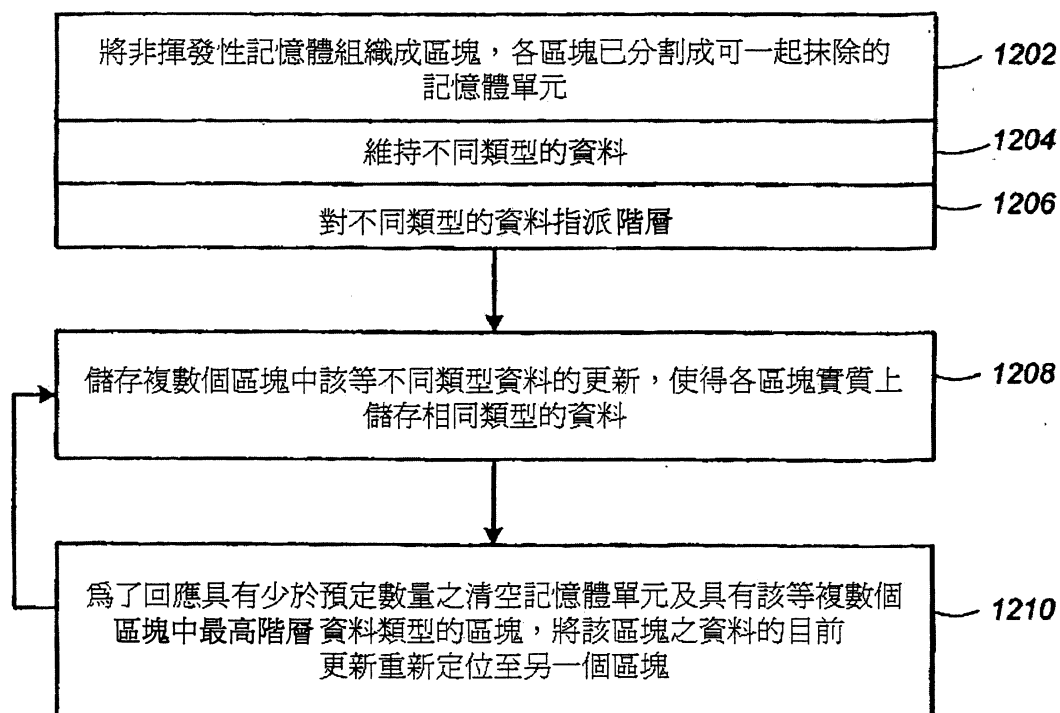


圖 48

七、指定代表圖：

(一)本案指定代表圖為：第 (37) 圖。

(二)本代表圖之元件符號簡單說明：

(無元件符號說明)

八、本案若有化學式時，請揭示最能顯示發明特徵的化學式：

(無)