公告本

303549

申請日期	83.07.18
案號	83106551
類別	Int.Cl6 H03M 7/30

（以上各欄由本局填註）

A4
C4

303549

發明 ~~新型~~ 專利說明書

一、發明 ~~新型~~ 名稱	中文	用以執行按序資料壓縮演算法之方法及裝置
	英文	"METHOD AND APPARATUS FOR EXECUTING A SEQUENTIAL DATA COMPRESSION ALGORITHM"
二、發明 ~~創作~~ 人	姓名	1.馬丁・歐瑞利諾・哈斯納 2.修・朵・卡尼 3.伍・史威吉夏恩 4.田村・徹也
	國籍	1. 美國　2. 以色列　3. 德國　4. 日本
	住、居所	1.美國加州波洛艾托市波托拉街1610號 2.以色列柯拉尼市柯拉尼街60號 3.德國多默凱奇霍德市霍萊艾奇46號 4.日本〒252大和市つきみの1-7-1
三、申請人	姓名（名稱）	美商萬國商業機器公司
	國籍	美國
	住、居所（事務所）	美國紐約州阿蒙市
	代表人姓名	費羅普

303549

（由本局填寫）

承辦人代碼：	A6 B6
大　　類：	
ＩＰＣ分類：	

本案已向：

國（地區）	申請專利，申請日期：	案號：	，□有　□無主張優先權
美	1993.7.8	08/089211	

有關微生物已寄存於：　　，寄存日期：　　，寄存號碼：

（請先閱讀背面之注意事項再填寫本頁各欄）

裝　訂　線

五、發明說明（1）

發明之領域：

本發明係關於用以在資料儲存系統中壓縮及解壓縮資料之裝置及方法，特別是關於牽涉一按序資料壓縮演算法區塊平行執行之裝置及方法，特別適用於低檔資料儲存系統。

發明之背景：

已知資料壓縮及解壓縮之效率基本上與緩衝器尺寸及使用的編碼執行過程有關。利用軟體執行壓縮／解壓縮演算法速度慢且因而不適於高速或即時之應用。利用硬體執行該演算法需要一定量的硬體－依所採實施技術之平行程度而變。若需求太多的硬體，可能很難將該資料壓縮演算法整合在一控制器中。

Lempel及Ziv於1977年5月在IEEE資訊理論會刊頁數337-343所發表之"一通用型循序資料壓縮演算法"論文中描述了一種供有效率地壓縮資料之演算法。

此Lempel-Ziv 1（LZ1）演算法乃一循序演算法－將諸串具不同長度的二進制資料壓縮在一固定長度之經壓縮二進制格式中。其執行係用一歷史緩衝器，該緩衝器包含正確序列中一檔案之最近諸位元組或字組。有規律地，藉由重複執行一基本常式，只要輸入位元組之序列與歷史緩衝器中之一序列相匹配，新的諸位元組即被讀取，因而產生一循序的資料流。因為各輸入位元組均循序地與歷史緩衝器中之各位元組相比較，故需要極大量的計算時間，使此技術不適於即時之應用。

修正 84 12 30 日
補充

五、發明說明（　）

1991年12月31日於美國申請且已共同轉讓之序號07/807,007案（備審案件目錄AT991-030）描述了LZ1演算法之一典型執行過程且接著引用了一些專利案（本發明未視為材料），該等專利案涉及增進LZ1演算法之執行速度或獲致之壓縮量等技術。

本發明所引用之此應用（已共同轉讓）描述了以硬體實施LZ1演算法之一完全平行架構。在以一內容可定址記憶體（CAM）充當歷史緩衝器之情況下，各輸入位元組均同時地與歷史緩衝器中之所有位元組比較。此完全平行硬體方法適宜地提供了LZ1演算法之最快速執行。然而，其各不同緩衝器位置（亦即CAM位址）均需一各自之比較器，且只有當歷史緩衝器滿了時才能獲致最大的效率（速度／硬體之性能）；亦即只有在資料儲存媒體各磁區（sector）或輸入資料欄位之一初始載入期間後方能獲致最大效率。因此，若該扇形區之尺寸約等於歷史緩衝器，則該完全平行執行過程將需要許多冗餘的操作。

因為一裝置控制器晶片之尺寸基本上同於實施該純平行壓縮所需晶片之尺寸，故一平行壓縮晶片無法有效率地用於一裝置控制器中從事壓縮。此完全平行方法之主要利用係給主機資料壓縮，其中該壓縮晶片位於主機控制器之內。

有需要提出一種資料壓縮／解壓縮之裝置及方法一藉由使用一模組化架構實施LZ1演算法：

1.將歷史緩衝器劃分為複數個區塊，在一區塊中平行地

五、發明說明（　）

比較其內所有位元組，且循序掃瞄諸區塊；

2.使一設計者可選擇自LZ1演算法之慢速循序執行至上述之最佳平行執行過程等任何速度（藉由選擇所要的平行程度）以針對特定應用之需要來限制硬體成本；

3.其特別適合用於那些在裝置控制器內進行資料壓縮之應用－所需執行速度約比在主機控制器內進行壓縮所需之速度小一個大小等級；及

4.當一輸入資料磁區及歷史緩衝器包含約相同數目之位元組時，其特別有利。

發明之總結：

一種用以執行按序資料壓縮演算法之裝置及方法，特別適用於需資料壓縮之裝置控制器中（與主機區別）。一歷史緩衝器壓縮一陣列共i個相同的水平片式單元。各片式單元儲存j個符號以分別定義出j個區塊，其中各片式單元內各符號間之距離正好為i個符號。在一串共i個的輸入符號中之眾符號乃平行地與諸片式單元先前所儲存的符號相比較，以識別匹配的符號序列。一控制單元控制該按序演算法之執行以調節諸比較器平行地掃瞄諸符號，而在各區塊中則循序掃瞄，且使匹配的符號序列及不匹配的符號序列儲存在陣列中。基於演算法執行速度對硬體成本間之折衷，參數i及j乃選擇以限制所需比較器之數目而達到所要程度的演算法執行效率。一優先順序編碼器由諸片式單元的輸出信號計算各j，i位址（匹配序列於其中被識別），但只輸出其中一位址（像最小者）。輸入之眾符號

五、發明說明（ 4 ）

乃串列地寫入緩衝器內連接之符號位置中直到所有的緩衝器位置均填滿，然後緩衝器內最舊之符號串即被輸入符號取代。

圖式之簡單說明：

圖1 乃本發明資料處理系統之一實施方塊圖。

圖2 乃本發明資料壓縮／解壓縮裝置之一實施示意圖，包括一陣列相同的水平片式單元及一優先順序編碼器。

圖3 乃圖2 所示各水平片式單元架構之詳細圖式。

圖4 乃圖2 所示優先順序編碼器之一簡化電路邏輯圖。

較佳實施例之說明：

如圖1 所示，實施本發明之一資料處理系統包含一主電腦10－與包括壓縮／解壓縮裝置14之裝置控制器12互相進行資料之接收及傳送。裝置14包含一壓縮引擎15及一解壓縮引擎16－可分別用以壓縮及解壓縮資料。引擎15提供經壓縮之資料輸出至一輸入／輸出（I/O）裝置18（像包含複數個磁碟之一磁碟驅動器）。裝置18提供經壓縮之資料輸入至裝置控制器12之解壓縮引擎16。

如圖2 所示，資料壓縮／解壓縮裝置14包含一控制單元20，一歷史緩衝器22，及一優先順序編碼器24。輸入／輸出（I/O）匯流排11與控制單元20間互有資料來去傳送。資料之形式為"符號"，乃申請專利範圍所用之一上位術語，用以表位元組、半字組、字組、或任何其他預選數目之位元。然而，為有助於瞭解，此中假定了位元組之形式。

如所示者，歷史緩衝器22包含一陣列共128 個相同的水

五、發明說明（　）

平片式單元HS0 至HS127 ，其中各片式單元儲存4 位元組，因而將緩衝器劃分成4 個128-位元組儲存區塊。此模組架構產生了能儲存512 （128 x 4 ）個連續位元組之一歷史緩衝器，其中各HS單元內之各位元組正好以128 位元組（亦即，一區塊之尺寸）之距離彼此分開，各HS單元儲存4 位元組，該4 位元組各來自各區塊。

緩衝器22中各位元組b 均具有一獨特的位址－由其區塊索引B 及HS索引所識別。如此，對位元組0-127 ，區塊索引B 為0 ；對位元組128-255 ，區塊索引B 為1 ；對位元組256-383 ，區塊索引B 為2 ；及對位元組384-511 ，區塊索引B 為3 。HS索引為位元組位址對區塊尺寸做mod 運算所得者。

如此，位元組356 之位址將為[2, 100]。4 位元組b ，各來自相同HS位址（0...127）之各區塊，包含一字組W （其位址在此情況下將為（1,L; 2,L; 3,L; 4,L），其中L為一特定的HS位址）。在所示緩衝器為4x128-位元組之情況下，若位元組之位址以二進制格式表示，則區塊索引可用二最高位元代表，而HS單元索引可用七較低位元代表。

現在參考圖3 ，各HS單元包含一記憶暫存器或單元ME，一位址選擇器單元SE，一比較器單元CO，一多工器MU及閂鎖器S1、S2、S3。控制器20傳送一輸入符號串之相同8-位元位元組給各HS單元，且傳送4 個信號c1-c4 給各HS單元以控制該HS單元之多工器MU。控制器20亦平行地提供給各HS單元：r1及r2等信號，分別用以重置閂鎖器S1及S2；區

五、發明說明（　）

塊選擇信號ad1及ad2；一時脈信號ck，用以同步所有HS單元之操作；及一寫入致能信號w。

各HS單元具有分別代表閂鎖S1、S2、S3目前狀態之輸出s10、s20、s30。這些輸出以一循環方式構成下一個接續HS單元之輸入s1i、s2i、s3i（看圖2），HS127之輸出s10，s30係HS0之輸入s1i，s3i。多工器MU利用控制信號c1-c4，且產生一輸出m。

在操作中，假定緩衝器22之內容（亦即1L，2L，3L，4L，S1，S2，S3）未定義。此狀態發生於每次一新的磁區被壓縮時。一開始，控制單元20平行地送出一信號r1至所有HS單元內之及閘42以重置所有HS單元內之閂鎖器S1。一位元組只有在以下之條件下才能寫入一給定HS單元其記憶單元之一位置內：

1.來自控制單元20之寫入致能信號w必須為"1"。

2.此HS單元之閂鎖器S1必須為"0"（亦即在重置狀態）。

3.輸入此HS單元之s1i必須為"1"。

這些條件係以圖3所示之及閘50加以滿足。當一區塊滿了，所有的HS單元（除了最後一個，在此為HS127）均被設定（亦即為"1"）。其S1="0"之HS單元乃新輸入位元組取代緩衝器22中一舊位元組之位置。然而，因為所有HS單元之S1閂鎖初值為"0"，控制器20乃經由導線44及或閘46（圖2）提供一"1"信號。此"1"信號持續直到所有的HS單元（除了最後一個，在此為HS127）均為"1"。此程

五、發明說明（　）

序重複（亦即，重置所有HS單元內之S1閂鎖器及在導線44上持續"1"信號之初值化）直到所有區塊均滿了；於其時導線44上之信號即由控制單元20切換至"0"。

各HS單元之閂鎖器S1亦提供該單元之輸出s10。接連地，所有HS單元之S1閂鎖器乃用做一128-位元移位暫存器。此移位暫存器藉寫入致能信號w進行更新（該寫入致能信號係由控制單元20平行地傳送至所有HS單元）。

寫入致能信號w亦使來自導線48之一輸入位元組寫入HS單元之記憶單元ME。此位置由諸閂鎖器S1及諸輸入s1i在1L，2L，3L，4L等位置中決定一個－如由選擇器單元SE所決定者。諸閂鎖器S1的優點之一乃：諸記憶單元不需在一新磁區的開始時被重置，如此降低了功率消耗。此特點在一裝置控制器執行過程具有實際的重要性－在功率損耗因電池電源限制而相當嚴格時。

為以一新區塊啟始一串比較操作，控制單元20平行地送一信號r2至所有HS單元以經由及閘40重置閂鎖器S2。S2之輸出控制各HS單元內之選擇器單元SE。而且，控制單元20送二信號ad1及ad2至所有HS單元之SE單元。這些2-位元信號ad1及ad2代表鄰近區塊位址（例如B1及B2）。依各HS單元內閂鎖器S2之狀態而定，ad1及ad2二者之一變為該HS單元之決定區塊位址ad0。一開始因所有S2閂鎖器均重置，對所有HS單元而言ad0等於ad1。一HS單元之S2閂鎖器輸出變為輸出s20且接著變為下一個HS單元之輸入s2i。

五、發明說明（ 8 ）

如此所有的閂鎖器S2形成了一個其輸入（來自HSO 之s2i ）總是"1" 之128-位元移位暫存器。當輸入位元組串與緩衝器22內容之匹配繼續著，此移位暫存器乃由底部開始填"1" ，如此致使愈來愈多的HS單元選擇ad2 取代ad1 。此在匹配過程期間構成區塊邊界之一虛擬移位。例如，假定匹配已至緩衝器22之位元組255 （亦即至位元組位址[2,127] ），則下一個輸入位元組必須與緩衝器之位元組256 相比較（其具有不同的區塊位址，及[3,0] ）。

輸入之位元組亦經由導線48之分支傳送至各HS單元之一對應比較器單元CO。在所有128 個HS單元內之比較器單元CO同時以共128 位元組與輸入位元組相比較（該128 位元組乃由選擇器單元SE在4 個區塊中所選一特定區塊內所含者）。若由諸記憶單元ME之一所輸出之位元組與導線48上之輸入位元組相匹配，則比較器CO將提供一"1" 輸出信號m 至多工器單元MU。

多工器單元MU由控制器單元20調節為4 個模式或狀態其中之一－由c1-c4 等信號決定。由這些信號所執行的功能如下：

信號c1表示該輸入位元組將為一新匹配串之第一位元組且目前正動作（亦即其位址由ad0 給定）之區塊係滿的。在此情況下，閂鎖器S1之狀態可予忽略。

信號c2表示該輸入位元組亦將為一新匹配串之第一位元組但目前正動作之區塊尚未滿（亦即，此區塊中某些記憶位置之內容未定義，因而一匹配只能在閂鎖

五、發明說明（ 9 ）

器S1為"1" 時才會發生）。

信號c3表示該輸入位元組不是一匹配串之第一位元，且閂鎖器S1之狀態不能忽略。此情形之發生只在：當整個緩衝器已滿且一區塊之移位區域與另一區塊之區域（匹配串諸位元組將寫入之區塊）重疊時。

信號c4表示該輸入位元組不是一匹配串之第一位元，但閂鎖器S1之狀態可予忽略。

若一輸入位元組不是一匹配串之第一位元組，則一連續的匹配需要其先行的位元組亦必須是匹配者（由該先行HS單元之輸入信號s3i 所指之條件）方能成立。

如此，當多工器MU以信號c1或c2加以調節，信號s3i 即可忽略，此因s3i 指出一串列之一初始位元組之匹配。比較器單元C0之m信號選擇性地與信號c1在52處做邏輯"及"，與信號c2及s10 在54處做邏輯 "及"，與信號c3、s10、及s3i 在56處做邏輯 "及"，與信號c4及s3i 在58處做邏輯 "及"。52、54、56及58等及閘之輸出在60處做邏輯"或"，產生輸出信號m。若輸出信號m 為"1"，表示緩衝器22內之一匹配串已擴展以包括該最近的輸入位元組。所有HS單元之m 輸出在閘62處（圖2）做邏輯"或"以發信告知控制單元20：目前輸入序列已在目前正動作之區塊中找到一有效的匹配。一HS單元之此輸出信號m 亦輸入至閂鎖器S3。因此，若一HS單元其閂鎖器S3之輸出s30 為"1"，則表示由此單元及位址ad0 所決定的緩衝器位置為先前所存一序列（與最近輸入序列相匹配者）到目前為止之最後

五、發明說明（10）

位元組。各HS單元之s30輸出均連接至優先順序編碼器24。

必須注意到緩衝器22之更新係與串比較操作無關且可同時完成者。

優先順序編碼器24利用來自128個HS單元之s30輸出以計算歷史緩衝器22內一序列（與待壓縮輸入資料之最近序列相匹配者）之終止位址並對其實施編碼。在一平行執行過程，如此中所描述者，歷史緩衝器22中可能有一個以上之序列與輸入序列相匹配。在此情況下，根據本發明之一特點，編碼器24對一匹配序列將只提供一單一的終止位址。

此單一終止位址選擇之方式將藉由參考圖4所示之簡化的優先順序編碼器24'做說明。此編碼器對512個位元組實施編碼，但為了簡化圖式及說明乃假定只有8個HS單元，各儲存64個位元組，各位元組來自64個區塊中之各對應區塊，以編碼成只有3個位元（而不是7個位元－轉移來自128個圖3所示種類的HS單元之位址所必需者）。

如圖4所示，簡化的優先順序編碼器24'接收來自所有（現為8）HS單元之8個s30信號當做輸入。接著在HS諸單元之8個索引中決定出其s30為"1"之最低索引。此係由圖4所示之邏輯達成，該邏輯忽略所有的s30信號（除了具有最小HS位址者）。如此，決定了一匹配序列之一唯一終止位址。此位址送至控制單元20。

因為解壓縮演算法需要一匹配序列之啟始位址，故必須

五、發明說明（　）

由優先順序編碼器24所提供之終止位址計算出此啟始位址。因已知該匹配序列啟始於目前正動作之區塊中（所有的S2閂鎖器一開始均被重置），故只需計算該位址之最後7個位元。此可藉由單純地減去該匹配序列之長度而完成；亦即，自該終止位址減去此序列中之位元數且將此結果對HS單元之數目（在圖3所示實施例中為128）做mod運算而得到。匹配序列之長度乃經壓縮資料之一部份，因此在控制單元20中其係可利用的（可利用一遞增計數器輕易獲致）。

供經由申請人之LZ1修正型壓縮演算法壓縮過之資料實施解壓縮之演算法本質上乃屬循序者，因此與壓縮引擎15所用之平行程度無關。然而，為了有效率的硬體利用，壓縮引擎15之歷史緩衝器22乃加以利用。經壓縮之輸入資料包含一起始緩衝器位址，一串長度數，及在經壓縮串列中為最後符號之一字元。控制單元20於初始位址取出緩衝器內容且以該取出之符號在同於壓縮引擎15之方式下對緩衝器22進行更新。此步驟以該經壓縮串列之長度重複進行。在此處理期間，緩衝器位址係以循環之方式固定遞增。最後，最後之符號亦送至主機10，且控制單元20再度準備好接收下一個經壓縮串列之資料。

隨附於此之附錄A乃用C語言所寫之一程式，用以實施根據本發明所修正之LZ1壓縮演算法；附錄B亦為用C語言所寫之一程式，用以對經由申請人之修正型LZ1壓縮演算法壓縮過之資料實施解壓縮。二程式均有註解以提供實

五、發明說明（12）

施本發明系列步驟之額外解釋。

儘管本發明已參照其較佳實施例特別地做了說明與教示，習於此藝者將瞭解在未偏離本發明精神及範疇下，許多形式上及細節上之改變均為可能。因此，本發明應只受限於申請專利範圍所請求者。

五、發明說明 (13)

附錄A-壓縮

```
/*****************************************************************
********** 此程式模擬一分割之資料壓縮                    **********
********** 演算法LZ1,其中各串均^^^^^^                    **********
********** 繼之以一單一之位元組^^^^^^                    **********
*****************************************************************/

#include <stdio.h>

#define B1   128   /* 記憶區塊數 */
#define B2   4     /* 一區塊內之位元組數 */
#define L_K 6
#define L    64    /* L=2**L_K, 一串之最大長度 */
#define C_K 9
#define C    512   /* C=2**C_K, 最大區塊C=B1*B2 */
#define M_NR 10000
#define MAX(u,v) ((u>v) ? u : v)
#define MIN(u,v) ((u<v) ? u : v)

int s1[B1], s2[B1], s3[B1];          /* 閂鎖器 */
int match[B1];                       /* 中間結果 */
int a1, a2;                          /* 緩衝器中新寫入位置 */
int p1, p2;                          /* 掃瞄中實際之緩衝器之位置 */
int o1, o2;                          /* 掃瞄之第一緩衝器位置 */
int z1, z2;                          /* 串之起始位址 */
int buffer[B1][B2];                  /* 緩衝器 */
int byte;                            /* 輸入位元組 */
int oldbyte[2];                      /* 最先二輸入位元組之備份 */
int recbyte;                         /* 最近位元組之備份 */
int l, l_a;                          /* 串長度值 */
int r;                               /* 掃瞄計數器 */
int any_mt;                          /* 至少有一匹配 */
int first, new, end;                 /* 邏輯變數 */
FILE *output;                        /* 輸出檔案 */

long c_input, c_output, c_iter, c_bits, c_cycles;
      /* 供輸入, 輸出, 重複, 輸出位元及週期之計數器 */
long c[L+1];                         /* 匹配之計數器陣列 */

/* 此功能計算一串之起始 */

void new_p()
{
/* 若有一至少為2 之串匹配, 指標之位置即被計算 */


   if (new && l>=2) {
/* 記憶區塊之數目, 假設 1 < B1. */
/* 一匹配被找到之位置(第一位置) */
      for (z1=0; !s1[z1]; z1++);
/* 位置之計算乃朝向原指標位置 */
      if (z1<l-1) {
         z1 = z1+B1-l+1;
      }
      else {
```

五、發明說明（ 14 ）

```
                z1 = z1-1+1;
            *
            z2 = r;
        *
    *

    /* 此功能將一指標前進至一緩衝器位置. */

    void inc_p(int *p1, int *p2)
    *
        *p1 = (*p1+1) % B1;
        if (!(*p1))
            (*p2)++;
    *

    /* 此功能寫入一緩衝器位置且使對應之指標前進. */

    void write_b()
    *
        int i;

    /* 目前之位元組剛讀進來，必須將其存在緩衝器中. */
        recbyte          = byte;
        buffer[a1][a2] = byte;

    /* 供緩衝器諸位置之移位暫存器必須前進 */
        if (s3[B1-2])
            for (i=0; i<B1; i++)
                s3[i] = 0;
        else *
            for (i=B1-1; i>0; i--)
                s3[i] = s3[i-1];
            s3[0] = 1;
        *

    /* 目前緩衝器指標之位置前進了. */
        inc_p(&a1, &a2);
    *

    int block_c()
    *
        int i, j;
        int t, k;

    /* 重置所有的計數器. */
        c_input = c_output = c_iter = c_bits = c_cycles = 0;

    /* 讀進第一位元組. */
        if ((byte = getchar()) == EOF)
            return 1;
        c_input++;
```

五、發明說明 (15)

```
/* 重置緩衝器之指標及重置諸閂鎖器s1. */
    a1 = a2 = 0;
    end = 0;
    for (i=0; i < B1; i++)
        s1[i] = s3[i] = 0;

/* 啟始一新串: 第一及新等變數被設定，長度1為0,
   o1及o2反映目前緩衝器位置, 以目前緩衝器位置
   (j = a2)為啟始且掃瞄次數r 被重置. */

    while (1) *
        first = new = 1;
        l     = 0;
        o1    = a1;
        o2    = a2;
        j     = a2;
        r     = a2;

/* 以一新掃瞄為啟始: 所有的s2閂鎖器均重置(以所有
   記憶區塊中之相同位置為啟始). */
        while (1) *
            for (i=0; i < B1; i++)
                s2[i] = 0;

/* 以一新匹配為啟始 */
            while (1) *

/* 在所有記憶區塊內之獲選內容與目前位元組相比較.
   此需要一個週期. */
                c_cycles++;
                c_iter++;

/* 處理中之位元組與諸記憶區塊內之獲選內容相比較. */
                any_mt = 0;
                for (i=0; i < B1; i++) *

/* 考慮緩衝器之循環結構. */
                    t = (byte == buffer[i][j+s2[i]]);

/* 一預期串其第一位元組之匹配不同於其餘位元組之匹配. */

                    if (first && new)
                        match[i] = (t && s3[i]);
                    else if (first)
                        match[i] = t;
                    else
                        match[i] = (i) ? (t && s1[i-1]) : (t && s1[B1-1]);
/* 有任何匹配? */
                    any_mt = (any_mt || match[i]);
                *

/* 將諸位元組存於暫時緩衝器中. */
```

五、發明說明（16）

```
                if (new) *

    /* 一預期串之第一或第二位元組被保存. */
                    if (l<=1)
                        oldbyte[l] = byte;

    /* 該位元組寫入緩衝器中且存在最近的位元組緩衝器中. */
                    write_b();
                *
    /* 更新緩衝器之下一個位置 */
                else
                    inc_p(&p1, &p2);

    /* 已有一匹配 */
                if (any_mt) *

    /* 匹配結果存於移位暫存器s1中 */
                    for (i=0; i < B1; i++)
                        s1[i] = match[i];

    /* 我們不再處於串之啟始處. */
                    first = 0;

    /* s2循環移位陣列之更新. */
                    if (s2[B1-1]) *
                        j++;
                        for (i=0; i < B1; i++)
                            s2[i] = 0;
                    *
                    else *
                        for (i=B1-1; i > 0; i--)
                            s2[i] = s2[i-1];
                        s2[0] = 1;
                    *

    /* 目前串長度與先前整個串長度相匹配且變為新的整個串長度. */
                    if (!new && (l==l_a))
                        new = 1;

    /* 由輸入獲致下一個位元組. */
                    if (new) *
                        l++;
                        if (c_input == C || l == L) *
                            end = 1;
                            break;
                        *
                        c_input++;
                        if ((byte = getchar()) == EOF)
                            break;
                    *

    /* 下一個位元組已讀進. */
                    else *
```

五、發明說明 (17)

```
                        l_a++;
/* 由舊的位元組緩衝器獲致下一個位元組. */
                    if (l_a == 1)
                        byte = oldbyte[1];
/* 由最近的位元組緩衝器獲致下一個位元組. */
                    else if (l_a == 1)
                        byte = recbyte;
/* 由緩衝器獲致下一個位元組，此花費一個週期. */
                    else *
                        byte = buffer[p1][p2];
                        c_cycles++;
                    *
                *
            *
/* 不匹配，因此必須啟始一新的重覆. */
            else
                break;
        *                   /* 結束內 while */

/* 已到達檔案之終點，串具有其最大長度，區塊尺寸係最大，
   或掃瞄已完成等四者其中之一. */
        if (byte == EOF || l == L || r == 0 || c_input == C) *
/* 若尚未完成則串之啟始指標即被決定. */
            new_p();
            break;
        *

/* 我們可繼續下一掃瞄. */
        else *
/* 決定字串之啟始指標. */
            new_p();

/* 為下一回合重置該二進制變數"新" */
            new   = 0;

/* 下一比較為一串之第一位元組. 此位元組取自該舊的
   位元組緩衝器. */
            first = 1;
            byte  = oldbyte[0];

/* 重置該暫時串之長度. */
            l_a = 0;
/* 增加目前掃瞄的數目. */
            r--;

/* 採緩衝器中先前串之第一位元組位置. */
            p1 = o1;
            p2 = o2;

/* 掃瞄以記憶區塊(循環的)中之下一個位置為啟始. */
            j = r;
        *
    *                       /* 終止中間 while 迴圈 */

/* 不匹配，9 位元輸出. */
```

五、發明說明（18）

```
            if (l==0) {
                if (byte != EOF)
                    fprintf(output, "0 %d\n", recbyte);
                c_output++;
                c_bits += 9;
            }
        /* 只有1 位元組匹配，18位元輸出. */
            else if (l==1) {
                fprintf(output, "0 %d ", oldbyte[0]);
                if (byte != EOF && !end)
                    fprintf(output, "0 %d\n", recbyte);
                c_output += 2;
                c_bits += 18;
            }
        /* 至少有2 位元組匹配. */
            else {
                fprintf(output, "1 %d", z1);
                fprintf(output, " %d", z2);
                fprintf(output, " %d\n", l);
                if (byte != EOF && !end)
                    fprintf(output, "0 %d\n", recbyte);
                end = 0;
                c_output += 3;
            }
        /* 依緩衝器中之資料量輸出. */
                for (i=0, k=1; k<=B2*B1; i++, k*=2)
                    if (o1+B1*o2 < k) {
                        c_bits += 10+i+L_K;
                        break;
                    }
            }

            (c[l])++;

            if (byte == EOF || c_input == C)
                break;

            else {
                c_input++;
                if ((byte = getchar()) == EOF) {
                    c_output++;
                    (c[0])++;
                    c_bits += 9;
                    break;
                }
            }
        }                           /* 終止外 while 迴圈. */

        return (byte == EOF);
    }


    int main()
```

五、發明說明（19）

```
        float d_input, d_output, d_iter, d_bits, d_cycles;
        long mi_output, mi_iter, mi_bits, mi_cycles;
        long ma_output, ma_iter, ma_bits, ma_cycles;
        float mi_rat1, mi_rat2, ma_rat1, ma_rat2, h_rat;
        int i;
        int j;

    /* 重置所有的計數器. */
        d_input = d_output = d_iter = d_bits = d_cycles = 0;
        mi_output = mi_iter = mi_bits = mi_cycles = M_NR;
        ma_output = ma_iter = ma_bits = ma_cycles = 0;
        mi_rat1 = mi_rat2 = 2;
        ma_rat1 = ma_rat2 = 0;

        output = fopen("comp.dat", "w");

        for (j=0; j<L; j++)
            c[j] = 0;
        i = 0;

        while (1) *
            if (block_c())
                break;
            i++;
            d_input += (float) c_input;
            d_output += (float) c_output;
            d_iter += (float) c_iter;
            d_bits += (float) c_bits;
            d_cycles += (float) c_cycles;
            mi_output = MIN(mi_output, c_output);
            ma_output = MAX(ma_output, c_output);
            mi_iter = MIN(mi_iter, c_iter);
            ma_iter = MAX(ma_iter, c_iter);
            mi_bits = MIN(mi_bits, c_bits);
            ma_bits = MAX(ma_bits, c_bits);
            mi_cycles = MIN(mi_cycles, c_cycles);
            ma_cycles = MAX(ma_cycles, c_cycles);
            h_rat = (float) (c_output*9) / (float) (c_input*8);
            mi_rat1 = MIN(mi_rat1, h_rat);
            ma_rat1 = MAX(ma_rat1, h_rat);
            h_rat = (float) (c_bits) / (float) (c_input*8);
            mi_rat2 = MIN(mi_rat2, h_rat);
            ma_rat2 = MAX(ma_rat2, h_rat);
        *

        if (i<=0)
            printf("The file has no content");
        else *
            printf("Number of blocks: %d\n", i);
            printf("Inputbytes: %.0f   ", d_input);
            printf("Inputbits: %.0f\n", d_input*8);
            printf("Output-9-bits: Summe %.0f   ", d_output);
            printf("Average %.2f   ", d_output/i);
            printf("Maximal %d   ", ma_output);
```

五、發明說明（20）

```
        printf("Minimal %d\n", mi_output);
        printf("Outputbits: Summe %.0f    ", d_output*9);
        printf("Average %.2f    ", d_output*9/i);
        printf("Maximal %d    ", ma_output*9);
        printf("Minimal %d\n", mi_output*9);
        h_rat = (d_output*9) / (d_input*8);
        printf("Comp_ratio1: Average %.4f    ", h_rat);
        printf("Maximal %.4f    ", ma_rat1);
        printf("Minimal %.4f\n", mi_rat1);
        h_rat = (d_bits) / (d_input*8);
        printf("Comp_ratio2: Average %.4f    ", h_rat);
        printf("Maximal %.4f    ", ma_rat2);
        printf("Minimal %.4f\n", mi_rat2);
        printf("Iterations: Summe %.0f    ", d_iter);
        printf("Average %.2f    ", d_iter/i);
        printf("Maximal %d    ", ma_iter);
        printf("Minimal %d\n", mi_iter);
        printf("Cycles: Summe %.0f    ", d_cycles);
        printf("Average %.2f    ", d_cycles/i);
        printf("Maximal %d    ", ma_cycles);
        printf("Minimal %d\n", mi_cycles);
        for (j=0; j<=L; j++)
            if (c[j])
                printf("Matches for length %d: %d\n", j, c[j]);
*
fclose(output);
return;
```

五、發明說明（21）

附錄B-解壓縮

```
/*****************************************************************************
**********此程式模擬一分割之資料壓縮                                    **********
**********演算法LZ1,其中各串均                                        **********
**********繼之以一單一之位元組                                        **********/

#include <stdio.h>

#define B1   128  /* 記憶區塊數 */
#define B2   4    /* 一區塊內之位元組數 */
#define L_K 6
#define L    64   /* L=2**L_K, 一串之最大長度 */
#define C_K 9
#define C    512  /* C=2**C_K, 最大區塊C=B1*B2 */
#define M_NR 10000
#define MAX(u,v) ((u>v) ? u : v)
#define MIN(u,v) ((u<v) ? u : v)


int a1, a2;                              /* 緩衝器中新寫入位置 */
int buffer[B1][B2];                      /* 緩衝器 */
FILE *input, *output;                    /* 供延伸之輸入及輸出 */

/* 此功能將一指標前進一緩衝器位置. */

void inc_p(int *p1, int *p2)
{
    *p1 = (*p1+1) % B1;
    if (!(*p1))
        (*p2)++;
}

/* 此功能寫入一緩衝器位置且使對應之指標前進. */


void write_b(int byte)
{
/* 目前位元組剛讀進, 必須存於緩衝器中. */

    buffer[a1][a2] = byte;

/* 目前緩衝器指標之位置前進了. */
    if ((a1 == B1-1) && (a2 == B2-1))
        a1 = a2 = 0;
    else
        inc_p(&a1, &a2);
}


int main()
{
    int i;
    int byte;
    int datum1, datum2, datum3;
```

五、發明說明（22）

```
        int flag;

        input = fopen("comp.dat", "r");
        output = fopen("outp.dat", "w");
        a1 = a2 = 0;
        while (fscanf(input, "%d", &flag) != EOF) *
            fscanf(input, "%d", &datum1);
            if (!flag) *
                fputc((unsigned char) datum1, output);
                write_b(datum1);
            *
            else *
                fscanf(input, "%d", &datum2);
                fscanf(input, "%d", &datum3);
                for (i=0; i < datum3; i++) *
                    byte = buffer[datum1][datum2];
                    fputc((unsigned char) byte, output);
                    write_b(byte);
                    inc_p(&datum1, &datum2);
                *
            *
        *
        fclose(input);
        fclose(output);
        return;
    *
```

四、中文發明摘要（發明之名稱：　用以執行按序資料壓縮演算法之方法及裝置　）

一種用以執行按序資料壓縮演算法之裝置及方法，特別適用於需資料壓縮之裝置控制器中（與主機區別）。一歷史緩衝器壓縮一 i 個相同的水平片式單元的陣列。各片式單元儲存 j 個符號以分別定義出 j 個區塊，其中各片式單元內各符號間之距離正好為 i 個符號。比較器將一串共 i 個的輸入符號中之眾符號平行地與先前儲存於諸片式單元內之符號加以比較，以識別匹配的符號序列。一控制單元控制該按序演算法之執行以調節諸比較器平行地掃瞄諸符號，而在各區塊中則循序掃瞄，且使匹配的符號序列及不匹配的符號序列儲存在陣列中。基於演算法執行速度對硬體成本間之折衷，參數 i 及 j 乃選擇以限制所需比較器之數目而達到所要程度的演算法執行效率。一優先順序編碼

英文發明摘要（發明之名稱：　"METHOD AND APPARATUS FOR EXECUTING A SEQUENTIAL DATA COMPRESSION ALGORITHM"　）

An apparatus and method for executing a sequential data compression algorithm that is especially suitable for use where data compression is required in a device (as distinguished from host) controller. A history buffer compresses an array of i identical horizontal slice units. Each slice unit stores j symbols to define j separate blocks in which the symbols in each slice unit are separated by exactly i symbols. Symbols in a string of i incoming symbols are compared by i comparators in parallel with symbols previously stored in the slice units to identify matching sequences of symbols. A control unit controls execution of the sequential algorithm to condition the comparators to scan symbols in parallel but in each of the

（請先閱讀背面之注意事項再填寫本頁各欄）

裝　訂　線

A5
B5

四、中文發明摘要（發明之名稱：　　）

器由諸片式單元的輸出信號計算各j，i位址，匹配序列以各位址識別，但只輸出其中一位址（像最小者）。

英文發明摘要（發明之名稱：　　）

blocks sequentially and cause matching sequences and nonmatching sequences of symbols to be stored in the array. The parameters i and j are selected to limit the number of comparators required to achieve a desired degree of efficiency in executing the algorithm based upon a trade-off of algorithm execution speed versus hardware cost. A priority encoder calculates from signals output by the slice units each j,i address in which a matching sequence is identified, but it outputs the address of only one (such as the smallest) of these addresses.

（請先閱讀背面之注意事項再填寫本頁各欄）

裝　訂　線

六、申請專利範圍

1. 一種用以執行一按序資料壓縮演算法之裝置，包含：

一i個相同水平片式單元的陣列，各單元儲存j個符號以分別定義出j個分離區塊，其中各片式單元內各符號間之距離正好為i個符號；

i個比較器，將一串共i個輸入符號中之眾符號平行地與先前儲存於該等片式單元內之眾符號加以比較，以識別匹配的符號序列；及

包括控制器裝置之裝置，用以(i)執行該按序演算法以調節諸比較器平行地掃瞄諸符號，而在各該區塊中則循序掃瞄，及(ii)使匹配的符號序列及不匹配的符號序列儲存在該陣列中。

2. 根據申請專利範圍第1項之裝置，其中：

基於演算法執行速度對硬體成本間之折衷，參數i及j乃選擇以限制所需之該比較器之數目，以便達到所要程度的演算法執行效率。

3. 根據申請專利範圍第1項之裝置，其中各片式單元均提供一輸出信號，若一匹配的序列被識別則該輸出信號即為一狀態，且包括：

一優先順序編碼器，用以由該等輸出信號計算出匹配序列被識別之各j，i位址，但只輸出該等位址中之一位址。

4. 根據申請專利範圍第3項之裝置，其中各輸出信號在該一狀態時係辨識與輸入符號串中最近序列相匹配之一先前儲存序列其到目前為止之最後位元組。

六、申請專利範圍

5. 根據申請專利範圍第1項之裝置，包括：

用以禁止演算法施加至任何不含符號之區塊的裝置。

6. 根據申請專利範圍第1項之裝置，包括：

在各片式單元中均有之單一閂鎖器，各閂鎖串列連結而構成一移位暫存器；及

用以在每次諸符號要存入該陣列之不同位置前重置該等閂鎖器因而重置該移位暫存器之裝置。

7. 一種資料壓縮裝置，包含：

一歷史緩衝器，劃分為j個區塊，各區塊能儲存i個連續之符號，致使在對應符號位置其相鄰區塊內之符號彼此以i個符號相隔；

i個比較器；

一控制器，用以平行地將一串共i個輸入符號之各符號傳送至該等i個比較器；及

i個選擇器裝置，用以使諸輸入符號串列地寫入緩衝器諸接連符號位置中，直到所有的緩衝器位置均被填滿，然後使緩衝器中最舊的符號串以輸入符號取代。

8. 一種執行一按序資料壓縮演算法之方法，包含以下諸步驟：

提供一共i個相同的水平片式單元的陣列，各單元儲存j個符號以分別定義出j個分離區塊，其中各片式單元內各符號間之距離正好為i個符號；

利用i個比較器，將一串共i個輸入符號中之眾符號平行地與先前儲存於該等片式單元內之眾符號相比較，

六、申請專利範圍

以識別匹配的符號序列；及

藉由（i）平行地掃瞄諸符號但在各該區塊中則循序掃瞄，及（ii）儲存匹配的符號序列及不匹配的符號序列於一輸入／輸出裝置中來執行按序演算法。

9. 根據申請專利範圍第8項之方法，包括以下之步驟：

基於演算法執行速度對硬體成本間之折衷，選擇參數i及j以限制所需該比較器之數目，以便達到所要程度的演算法執行效率。

10. 根據申請專利範圍第9項之方法，包括以下諸步驟：

計算用以辨識匹配序列之各j，i位址；及

只輸出該等位址中之最小位址。

11. 根據申請專利範圍第10項之方法，包括以下之步驟：

辨識與輸入符號串中最近序列相匹配之一先前儲存序列其到目前為止之最後位元組。

12. 根據申請專利範圍第8項之方法，包括以下之步驟：

禁止演算法施加至任何未含有效符號之區塊。

13. 一種資料壓縮之方法，包含以下諸步驟：

提供一歷史緩衝器，該緩衝器劃分為j個區塊，各區塊能儲存i個連續之符號，致使在對應符號位置其相鄰區塊內之符號彼此以i個符號相隔；

提供i個比較器；

平行地將一串共j個輸入符號之各符號傳送至該等i個比較器；

將諸輸入符號串列地寫入緩衝器諸接連符號位置中，

（請先閱讀背面之注意事項再填寫本頁）

訂

線

六、申請專利範圍

直到所有的緩衝器位置均被填滿；及

接著以輸入符號取代緩衝器中最舊的符號串。

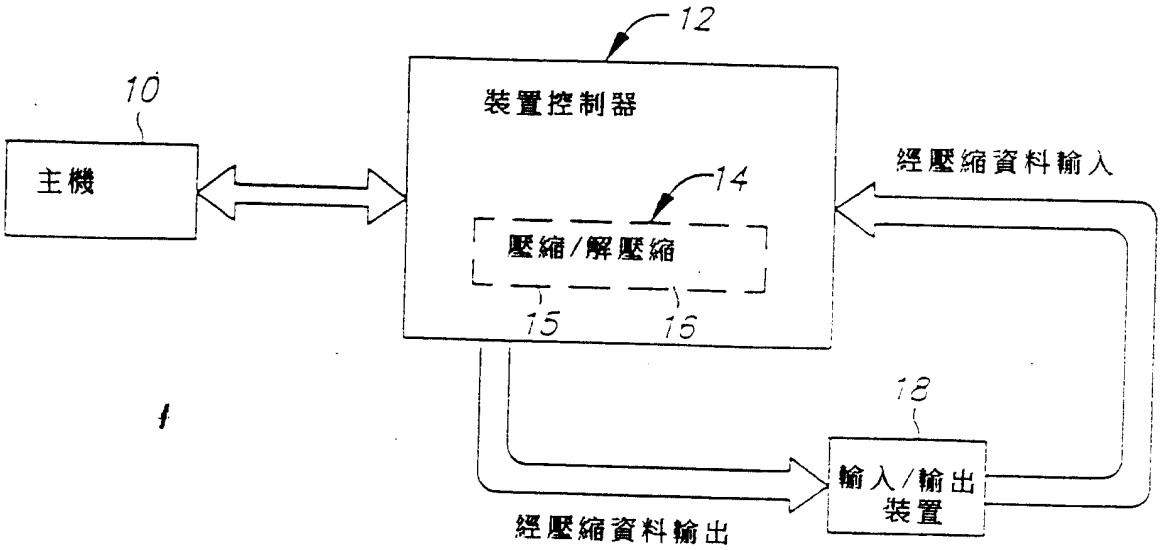

圖 1

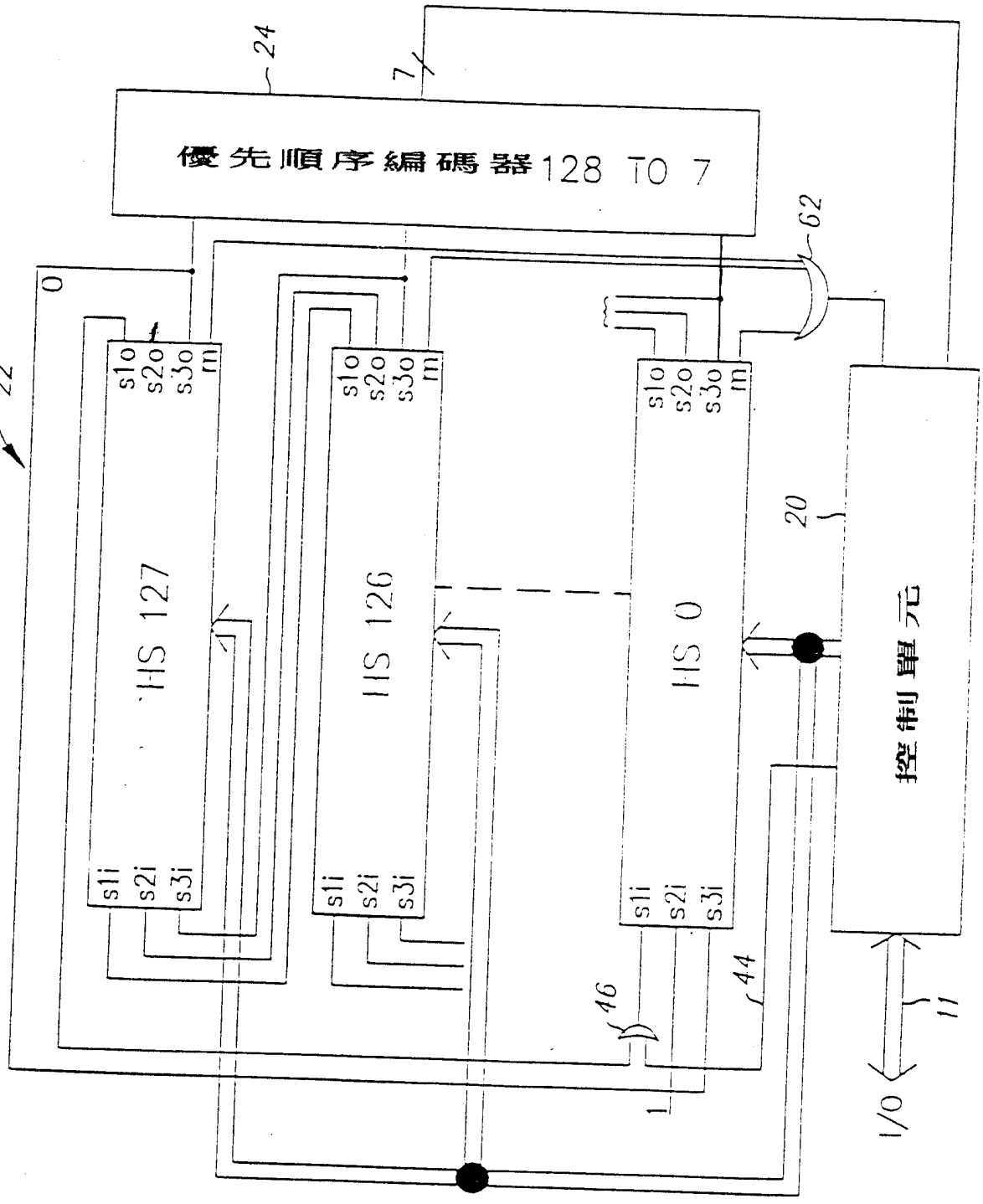
24
7
優先順序編碼器128 TO 7
62
0
22
s1o
s2o
s3o
m
IIS 127
IIS 126
IIS 0
s1i
s2i
s3i
20
控制單元
46
44
1
11
I/O

圖 2

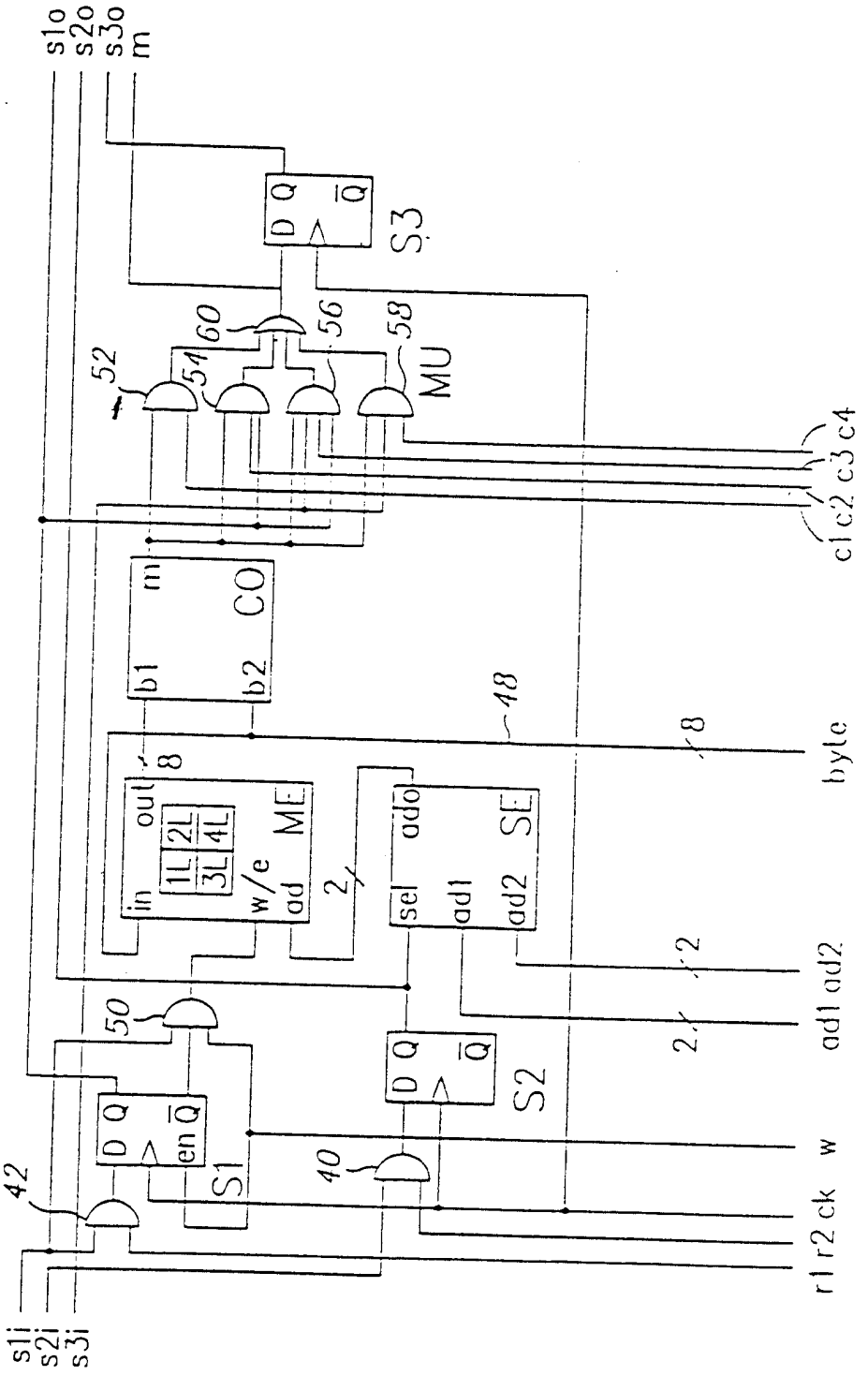
s1o
s2o
s3o
m
S3
MU
52
54
60
56
58
c1 c2 c3 c4
b1
b2
CO
m
18
8
byte
out
in
1L
2L
3L
4L
w/e
ad
ME
2
sel
ado
ad1
ad2
SE
2
2
ad1 ad2
50
S2
S1
42
40
s1i
s2i
s3i
r1 r2 ck w

圖 3

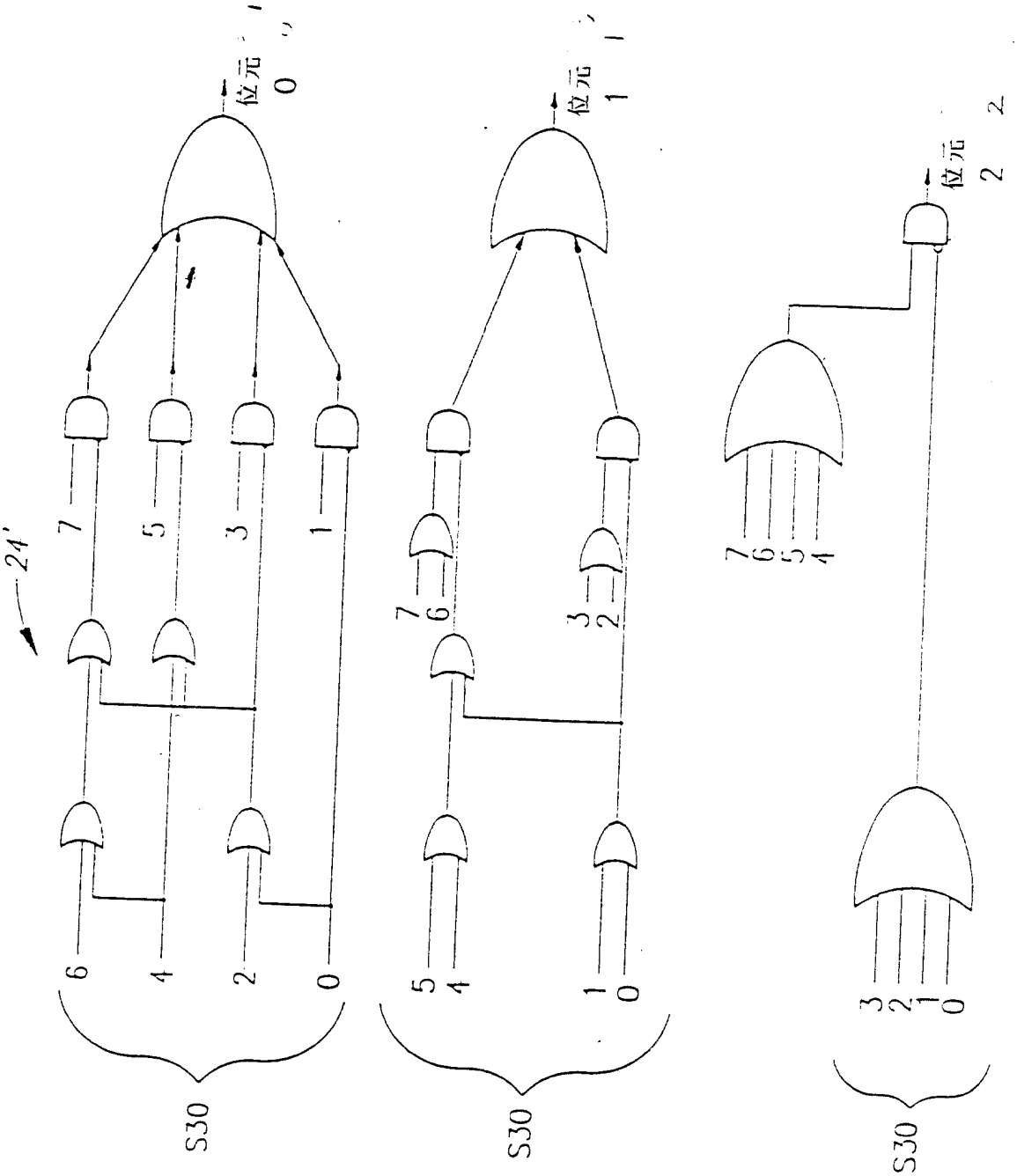

圖 4