

# APPLE II 軟體卡

## CP/M操作手冊

魏易休  
蔡毓琛

# 簡 介

Microsoft BASIC 是為 Z-80 及 8080 微處理機而設計的。它是現今微電腦系統所使用之各種 BASIC 語言中功能最廣的。目前已第五版發行，Microsoft BASIC (或稱為 BASIC-80) 符合 ANSI BSRX3.60-1978 文獻中所規定的 BASIC 規格。它也可與 Applesoft BASIC 相容。

在一裝有 Microsoft SoftCard 系統裏，BASIC-80 的最新版本 (第五版) 首次提供 Apple 使用者運用。它給 Apple 帶來新的能力，增加了許多特性如：PRINT USING，16 有效數字精準度，CALL，CHAIN 及 COMMON，WHILE/WEND，改進的 DISK I/O 等。

SoftCard 套件內包括有兩種 Microsoft BASIC 之版本：  
① MBASIC，它可存在於 16 段落 (sector) 型磁碟或 13 段落型磁碟內，功能包括所有標準 Applesoft extension，從低解析度繪圖能力到聲音及游標之控制皆有。  
② GBASIC，除了具有 MBASIC 所有功能外再加上高解析度繪圖能力，但它只能存放於 16 段落型磁碟上。

本手冊第 4 部份共分為 5 章再加上一些附錄。第 1 章比較了 Microsoft BASIC 和 Applesoft 相異之處，對那些熟悉 Applesoft 之讀者特別重要。第 2 章介紹兩種版本 Microsoft BASIC (本書以後都叫它 BASIC-80) 的初始設定 (initialization) 的命令及其資料表示法。第 3 章介紹 BASIC-80 內各個命令 (command) 及陳述 (statement) 之語法 (syntax) 和文法 (semantics)，這些命令及陳述都按字母順序排列。第 4 章

介紹 BASIC-80 所提供之函數。第 5 章則專門介紹 GBASIC，把 GBASIC 所具有之特性一一說明，附錄則包含有錯誤訊息表，ASCII 碼及數學函數，還有一些使用組合語言及 DISK I/O 時有用的參考資料。

本書並非想作為 BASIC 語言的課本，而只是描述 Microsoft BASIC 各項特性之參考手冊，若你想讀有關 BASIC，我們推薦下列書籍：

*BASIC* by Robert L. Albrecht, LeRoy Finkel, Jerry Brown ( John Wiley & Sons, 1973 )

*BASIC and the Personal Computer* by Thomas A. Dwyer and Margot Critchfield ( Addison-Wesley Publishing Co., 1978 )

*BASIC From the Ground Up* by David E. Simon ( Hayden, 1978 )

# 序

整套SoftCard 配件，包含有①一片叫做SoftCard 的界面板，②磁碟機及其界面板，③SoftCard 之系統磁碟。磁碟裏面存有CP/M 作業系統，MBASIC 及許多公用程式 ( Utilities ) 可供使用。乃是Apple II 愛好者在有了磁碟機後欲擴充週邊裝置 ( 如 printer, console, serial transmission interface, video card 等等 ) 時必備之套件。

原手冊 ( Microsoft SoftCard 手冊 ) 即是爲此而寫，它內容豐富，描述了SoftCard 配件的每一部份，( 內容有①簡介，②軟體及硬體詳述，③CP/M 使用手冊，④MBASIC 使用手冊，⑤公用程式手冊 )，但是其編寫亦有瑕疵之處，譬如在上冊中，有助於了解整個系統的圖示說明不多，所舉例子偏少，觀念亦有時未能交待清楚，同時在描述硬體時竟然連一張電路圖都不附，頗易令人陷入困思之境而不能自拔。

編寫之時有鑒於此，特別參考了美、日，數種不同之有關書籍，精選例題及圖表插入書中有助了解，同時在本手冊第2部份第3章亦插入整片SoftCard 板之電路圖，使讀者能從圖中推啟整個硬體系統間的相互關連，而免於暗中摸索之苦。

本手冊之編寫承蒙儒林出版社楊鏡秋先生鼎力協助支持，好友何狄林、劉玉堂的幫助甚多，謹此致感謝之意。

疏漏錯誤之處難免，敬祈指正。

蔡毓琛

魏易休

謹誌 71 . 9 . 20



# 目 錄

## 第〇部份：系統概述 1

SoftCard 簡介 2

系統需求 5

SoftCard 名詞解釋 5

## 第一部份：組立與操作手冊 7

### 第一章：如何組立Soft Card 9

- APPLE 週邊裝置板：那一種該放到何處 10
- 與CP/M 可直接合用的的界面板 11
- APPLE 磁碟機之裝法 14
- 列印機界面板之裝置 14
- 一般用途I/O之裝置 15
- 外部終端機的使用法 15
- SoftCard 之組立 16

### 第二章：開始使用CP/M系統 19

- 啓動Apple CP/M 系統 20
- 如何複製SoftCard 磁碟 23
- 如何創建CP/M 系統磁碟 26

- APPLE CP/M 與語言板一起使用的方法 28
- I/O 架構 29

### **第三章：APPLE CP/M簡介 31**

- 鍵盤的打入 33
- 輸出控制 34
- CP/M 的暖啓動 ( Warm Boot ) : Ctrl-C 34
- 更換 CP/M 磁碟 35
- CP/M 檔案命名規則 37
- 檔名稱格式 37
- 一些 CP/M 命令 : DIR, ERA, REN, TYPE 38
- CP/M 錯誤訊息 41
- SoftCard 系統磁碟上所包含的程式 43

### **第四章：Microsoft BASIC初步 47**

## **第二部份：軟體與硬體詳述 53**

### **第一章：APPLE II CP/M軟體詳述 55**

- 簡介 54
- I/O 硬體規約 54
- 6502 / Z-80位址轉換 56
- APPLE II CP/M 記憶體使用法 57
- 使用 SoftCard 時組合語言之撰寫 58
- ASCII字元碼 58

- 中斷處理 61

## 第二章：APPLE II CP/M I/O 架構框塊 65

- 簡介 66
- 主控制台游標定址 / 螢光幕控制 66
  - 硬體 / 軟體螢光幕功能表
  - 與終端機無關之螢光幕功能 / 游標定址
- 鍵盤字元之重新定義 71
- 非標準週邊裝置及 I/O 軟體支援 72
  - 把邏輯名字指定給實體 I/O 裝置：IOBYTE
  - 經由 I/O 向量表修補使用者軟體
- 呼叫 6502 副常式 81
- 週邊板的存在與位置之指示 83

## 第三章：硬體的描述 87

- 簡介 88
- 時序架構 88
- SoftCard板之控制 89
- 位址匯流排界面 90
- 資料匯流排界面 91
- 6502 之“更新”(Refresh) 92
- DMA Daisy鏈 93
- 中斷 93
- SoftCard 零件表 94
- SoftCard 電路圖 96



## 第三部份：CP/M參考手冊

### 第一章：CP/M的特色與功能介紹 97

- 簡介 102
- CP/M 2.0 功能概況 104
- CP/M 的功能描述 106
- 通用命令的結構 107
- 檔案參考 107
- 磁碟機的改換 109
- 內建命令的格式 110
  - ERAsE 命令
  - DIRectory 命令
  - REName 命令
  - SANE 命令
  - TYPE 命令
  - USER 命令
- 行的編輯與輸出控制 122
- 暫態程式命令 124
  - STAT
  - ASM
  - LOAD
  - DDT
  - PI-P.
  - ED

SUBMIT

DUMP

- 磁碟作業系統的錯誤訊息 170

## 第二章：CP/M 2.0的介面導引 175

- 簡介 176
- 作業系統的呼叫協定 179
- 檔案複製程式範例 207
- 檔案傾印程式範例 209
- 隨機存取的程式範例 213
- 系統功能摘要 220

## 第三章：CP/M編輯程式 221

- 編輯程式 ( ED ) 簡介 222
- 編輯程式的操作
- 文稿傳送功能 225
- 記憶體緩衝區的構造 232
- 記憶體緩衝區的操作
- 命令字串 236
- 文稿之搜尋及變更 238
- 源檔叢庫 ( Source Libraries ) 241
- 編輯錯誤狀況 242
- 控制字元摘要 243
- 編輯命令摘要 243
- 編輯程式文稿編輯命令 245

## 第四章：CP/M組合語言編輯程式

- 簡介 255
- 程式格式 258
  - 運算元 ( Operand )之形成
  - 標記
  - 數字常數
  - 保留字
  - 算術及邏輯運算子
  - 運算子之優先順序
- 組合語言編輯程式導向指令 265
  - ORG 指令
  - END 指令
  - EQU 指令
  - SET 指令
  - IF 及ENDIF 指令
  - DB 指令
  - DW 指令
- 運算碼 271
  - 跳離、呼叫、還原指令
  - 立即運算元指令
  - 資料移動指令
  - 算術邏輯單位運算
  - 控制指令
- 錯誤消息 278
- 範例 279

## 第五章：CP/M動態除錯工具程式（除錯劑） 285

- 簡介 286
- DOT 命令 290
  - A ( Assemble ) 命令
  - D ( Display ) 命令
  - F ( Fill ) 命令
  - G ( Go ) 命令
  - I ( Input ) 命令
  - L ( List ) 命令
  - M ( Move ) 命令
  - R ( Read ) 命令
  - S ( Set ) 命令
  - T ( Trace ) 命令
  - U ( Untrace ) 命令
  - X ( Examine ) 命令
- 在使用時應注意事項 307
- 範例 308

附錄：CP/M命令集 317



## 名詞對照

argument	引數
bit	比次
byte	拜
character sequence	字元序列
double precision variable	倍準實變數
dummy argument	啞引數
carriage return	回車字元
edit	編輯(修)
expression	表式
extent	延伸
form feed	饋表
format	格式
I/O configuration	Block I/O 架構框塊
interpreter	解譯程式
line	行(此處為橫行)
list	串
literal string	文義字串
mantissa	尾數(即小數點以後之數)
menu	名單
null	空的
overflow	溢位
prompt character	催促字元
random file	隨機檔
recbrd	資料錄
sequential file	循序檔

<b>s e r i a l - b o a r d</b>	串式傳輸界面板
<b>s e r i a l d a t a l i n k</b>	串式傳輸鏈接
<b>s i n g l e p r e c i s i o n v a r i a b l e</b>	單準實變數
<b>s t a t e m e n t</b>	陳述
<b>s t r i n g</b>	字串
<b>S p a c e</b>	空白
<b>t e x t</b>	文字，主文，文稿
<b>t r a p</b>	捕捉
<b>d i r e c t o r y</b>	索引（目錄表）
<b>d i r e c t o r y e n t r y</b>	索引項目
<b>e n t r y p o i n t</b>	進入點，入口
<b>f i l e c o n t r o l b l o c k</b>	檔案控制區（FCB）
<b>b u i l t - i n c o m m a n d</b>	內建命令
<b>t r a n s i e n t c o m m a n d</b>	暫態命令
<b>s o u r c e f i l e</b>	源檔案
<b>d e s t i n a t i o n f i l e</b>	目的檔案

# 第 ○ 部 份

# 系 統 概 述



## Soft Card 簡介

### 電路板

Microsoft SoftCard是一種可以插在APPLE II主機上的電路板。在裝設它之前，必須也閱讀本手冊第一部份組立與操作部份以確保不致因誤裝而導至機器損害。

一旦裝好了SoftCard板，你就可以軟體控制來選擇要使用6502或Z-80模式操作。當使用APPLE主機內處理機（6502模式）時，插在主機內的SoftCard板決不會影響APPLE之運作。

當使用SoftCard內處理機（Z-80模式）時，你就可使用：① Digital Reserch 公司之CP/M作業系統及②Microsoft公司第5.0版MBASIC翻譯執行程式（Interpreter）。這都屬於Soft-Card套件中的一部份。

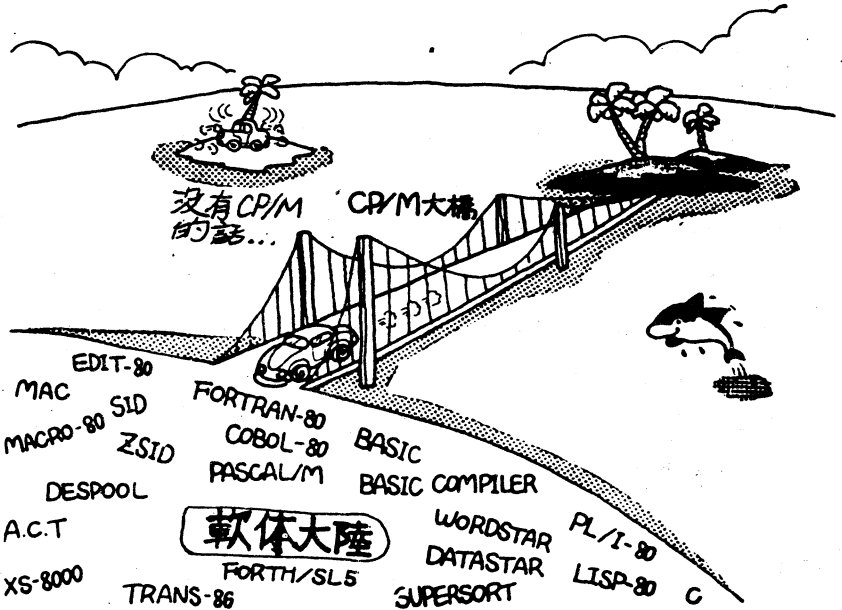
SoftCard板很容易安裝，它不需要額外的軟體及硬體裝置，但是卻大大的增強了APPLE之軟體能力。SoftCard裏邊有一塊Z-80A微處理機，因此它可處理任何為Z-80系統所寫的軟體。

### CP/M作業系統

除了電路板之外，CP/M是允許許多Z-80軟體能夠執行的最重要關鍵裝備。SoftCard套件內包括CP/M第2.2版。

CP/M（Control Program/Microprocessors之縮寫）是為Z-80及8080微處理機而設計的作業系統程式。它的內部含有許多小程序，這些小程序之功能為從磁碟內取出資料或到磁碟上去存

放資料。CP/M 已為大多數（幾乎全部）使用 Z-80 及 8080 處理機的系統所採用。由於它的使用廣泛，使得許多高階程式及應用軟體都是基於 CP/M 系統環境下而寫的。



CP/M可以和許多軟體交通

由於 SoftCard 板之出現，Apple 計算機使用者現在也可使用 CP/M 作業系統了。此乃 Microsoft 公司做了許多修改才做成的。

標準的 CP/M 能與 Apple CP/M 相容，但是把標準的 CP/M 載入 Apple 時有一個困難：Apple 的磁碟資料存放格式 (Format) 和 CP/M 磁碟所使用的不同。如果一個 CP/M 程式是為別種類型的計算機所寫的，若要在 Apple 上使用這些程式，那麼就必須經由標準 CP/M 系統後再載入到 Apple 上。這些過程在本書第 5 部份“軟體公用程式”部份再予說明。

#### 4 Apple II 軟體卡

CP/M 除了可支援各種不同的軟體，還有許多使用便利的優點為 Apple DOS 所不及，這些優點有：①與機器語言程式溝通容易，②較快的磁碟輸出與輸入，③檔案轉移簡單，④使用 Wild-card 型檔案名稱，使得使用者可以一個檔案名稱同時處理數個檔。

#### Microsoft BASIC 語言

Microsoft 公司第 5 版的 ANSI 標準 BASIC 翻譯執行程式 (interpreter) 亦是 SoftCard 套件內之元件之一。它有許多 Applesoft 所沒有的特點，其中包括：PRINT USING, CALL, WHILE/WEND, CHAIN, COMMON 以及許多磁碟輸出與輸入之命令陳述。此外，大部份 APPLESOFT 之繪圖功能在 MBASIC 語言內亦具備以充分地利用 Apple 系統的特殊能力。MBASIC 與 APPLESOFT 的各種不同比較表列可參考本書第 4 部份“MBASIC 參考手冊”。

#### 磁碟

有兩種磁碟，每一種都包含有 CP/M 與 MBASIC 及各種公用程式。但是磁碟內的格式 (format) 不同，一種是每圈軌道 (Track) 只含有 13 段落 (sector)，適用於 Apple 主機內未裝設語言板或不是使用 DOS 3.3 的系統。另一種磁碟則是每圈軌道含有 16 個段落，用於系統有語言板或是使用 DOS 3.3 作業系統時。16 段落型磁碟也包含了 MBASIC 語言之改良型：GBASIC，GBASIC 有高解析度繪圖能力之功能。

## 系統需求

SoftCard能在APPLE II或APPLE II PLUS微電腦上工作，它最少需48K RAM及一架磁碟機。

SoftCard板能與Apple語言板系統相配合，並能使用語言板內16K RAM中的12K(當系統在Z-80模式時，亦即SoftCard內之Z-80A微處理器動作之時)。

CP/M佔了7K RAM，但在執行使用者程式時只需用到其中5K。CP/M及MBASIC佔有的RAM空間超過29K，CP/M及GBASIC合佔的RAM空間則超過37K。

當處於6502模式時，SoftCard不會影響Apple主機之動作。

當使用Z-80模式時，系統就可溝通所有的標準型Apple I/O週邊板及某些獨立的週邊裝置。

## Soft Card 名詞解釋

有一些名詞在本書中常常出現，初次見到它們可能不甚了解，所以下面列出這些名詞，它們的定義及如此命名的原因：

### 44K系統

指的是裝有48K記憶體的APPLE II及APPLE II Plus系統，稱做“44K系統”的原因如下：當在使用SoftCard(Z-80模式)時，你只能用到48K內的44K記憶體，其它4K被用來處理Apple螢光幕及CP/M段落資料的讀寫工作。

### 56K系統

指的是帶有語言板的APPLE II或APPLE

II Plus 系統（總共有64K記憶體）。如48K系統一般，其中4K是用來處理Apple螢光幕及CP/M段落資料的讀寫工作。而又因為語言板上16K RAM中只有12K可被定址（addressable）。所以，總共實際可用到的只有56K。

**13段落 (Sector) 磁碟**

為SoftCard系統所使用的磁碟之一種。此種格式的磁碟適用於Apple DOS 3.2或更早版本以及沒有語言板時。

**16段落 (Sector) 磁碟**

為SoftCard系統所使用的磁碟之另一種，在有APPLE DOS 3.3或有語言板時使用，除了具備13段落型磁碟所有的軟體外，它還包含了BASIC語言之第2版——GBASIC，此語言具有高解析度繪圖功能。

**A: ~ F:**

A: , B: , C: , D: , E: 及 F: 代表磁碟機。以上是標準的CP/M磁碟機命名法則。由於本書使用CP/M系統，所以在本書內到處可以遇到這種符號。至於磁碟機名稱和磁碟機之關係，請參閱本書“組立與操作手冊”那一部份。

**外部終端機**

指的是兩種不同類型的裝置。一外部終端機可以是一24×80螢光幕界面板（video card,如Videx Videoterm），或者是本身就是系統的第2台終端機（如Hazeltine或SOROC）。

**RETURN或〈CR〉** 這些都代表按下APPLE鍵盤上的RETURN或回車字元鍵。

# **第一部份**

## **組立與操作手冊**

### **Installation and Operation**



# 第 一 章

## 如何組立 Soft Card

- APPLE 週邊裝置板：那一種該放到何處
- 與 CP/M 可直接合用的界面板
- APPLE 磁碟機之裝法
- 列印機界面板之裝置
- 一般用途 I/O 之裝置
- 外部終端機的使用法
- SoftCard 之組立



## 10 Apple II 軟體卡

SoftCard 的組立很簡單。但在組立前還是有些事你需注意，因為不適當的組立不但會損毀 SoftCard 而且還會弄壞APPLE系統，所以要特別注意。

```
※※※※※※※※※※※※※※※※
※      在組立 SoftCard 前      ※
※      務必小心詳讀下面的指示  ※
※※※※※※※※※※※※※※※※
```

### APPLE週邊裝置板：那一種該放到何處

在組立 SoftCard之前，必須確定其他的週邊裝置板裝置在APPLE主機內正確的溝槽 ( slot ) 當中，如此才能保證APPLE CP/M 之正常操作。

此上是必須做的，APPLE CP/M 週邊裝置板必需依照不同的目的插在各自特定溝槽中 ( 此點異於APPLESOFT及INTEGER BASIC，但類似APPLE PASCAL )。譬如，若你有一列印機 ( printer ) 界面板，它就必須放置在溝槽 1 上。這樣當你想使用列印機時可免除指定溝槽的麻煩；但在Applesoft 及 Integer BASIC 中則必須指定。

APPLE 語言板使用者請注意：

指定給CP/M 週邊裝置板的溝槽和給APPLE PASCAL 語言板的相同。因此，若你的系統原為APPLE PASCAL 而建的，則不需另外修改結構。

## 與 CP/M 可直接合用的界面板

與 APPLE CP/M 可直接合用 (directly compatible) 的週邊裝置板之各個類型列表如下 (這些板放入適當的溝槽後不需要作任何軟體修改就可工作) :

表 1

類型	板名稱
1	APPLE II 號磁碟控制板
2*	APPLE 通信界面板 加州 Communication Systems 7710A 串式 (serial) 傳輸界面板
3	APPLE 高速串列界面板 APPLE Silentye 列印機界面板 Videx Videoterm 24×80 螢光幕界面板 M & R Enterprises Sup-R-Term 24×80 螢光幕界面板
4	APPLE 並式 (parallel) 列印機界面板

\* CCS 7710A 串式傳輸界面板是第 2 類板中較受歡迎的一種。因為它提供握手式 (handshaking) 及可變的鮑速率 (baud) : 從 110 ~ 19200 鮑。如鮑速率不是 110 或 300 鮑的話, 則 APPLE 通信界面板的硬體需作修改。

還有些界面板子能與 APPLE CP/M 合用但未列在上面。一般

## 12 Apple II 軟體卡

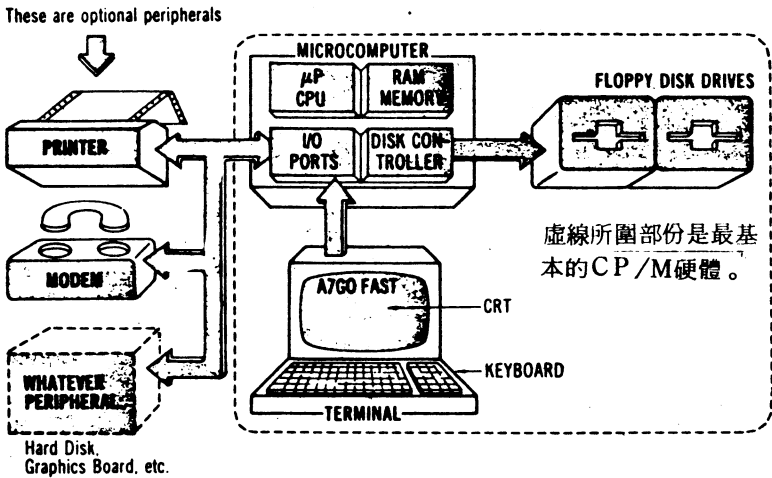
法則如下：任何能與APPLE PASCAL合用而未經修改軟體的界面板可能也與APPLE CP/M直接合用。至於其它的界面板，如果板子製造商所提供的軟體能放入你的CP/M系統（利用CONFIGIO程式）內的話，也可以使用。如想知道有關非標準界面板的更多的資料，可以參閱後面“軟體詳述”及“CONFIGIO公用程式”諸節

下表是每一APPLE溝槽的指定功能以及板的類型（見上表）除非特別註明，不能辨認的板或空的溝槽將不被理會。

溝槽	有效的板類型	功 能
0	I/O 時不使用	此溝槽可裝上語言板或APPLESOFT(或Integer) BASIC ROM板。(後者CP/M不用)。
1	第2, 3, 4類	列印機界面板(CP/M LST: 元件)。
2	輸入: 2, 3, 4類 輸出: 1, 2, 3, 4類	一般用途I/O(CP/M PUN: 及RDR: 元件)。
3	第2, 3, 4類	控制台輸出裝置(CRT或TTY)若沒有板子在此溝槽內, 則一般的APPLE 24×80螢光幕將被當作TTY: 裝置使用。
4	第1類	磁碟機E: 及F: 的磁碟控制板如果此板不存在的話可把SOFTCARD插入此溝槽。

溝槽	有效的板類型	功 能
5	第 1 類	磁碟機 C : 及 D : 的磁碟控制板。
6	第 1 類	磁碟機 A : 及 B : 的磁碟控制板。 (此板一定要有)。
7	任何類型	沒有指定使用目的, Soft Card 可插於此。

**重要事項:** 當你想要安放你的週邊裝置板時, 千萬必須確定 APPLE 主機電源已關掉, 否則 APPLE 主機會招致重大的損害。



一標準的 CP/M 微電腦硬體系統。

## APPLE磁碟機的裝置法

如上表所示，APPLE II 磁碟機控制板可放在第6、5及4溝槽，最少在溝槽6要有一片控制板。磁碟控制板是從第6槽裝起，然後第5，然後第4。

在CP/M中，每一磁碟機都有單一個字母的名字，後面接一個分號(:)。譬如，溝槽6磁碟控制板所連的DRIVE1，在CP/M中為磁碟機A: (見下表)。這就是在本手冊中參考磁碟機的方法。

CP/M名稱	溝槽	DRIVE號碼
第1架磁碟機: A:	6	1
第2架磁碟機: B:	6	2
第3架磁碟機: C:	5	1
第4架磁碟機: D:	5	2
第5架磁碟機: E:	4	1
第6架磁碟機: F:	4	2

DOS 3.3 或 Apple Pascal 使用者請注意：

Apple CP/M 提供DOS 3.3 及 Apple Pascal所使用的高容量16 段落磁碟格式以及標準型APPLE II 13 段落格式兩種磁碟。

## 列印機面板的組立

如果你有列印機 (printer)，它的界面板必須插入溝槽1內。

大多數為 Apple Pascal 所設計的界面板也同樣的適用於 APPLE CP/M。

## 一般用途 I/O 界面板的組立

一般用途 I/O 界面板 (如調復機 (modem), 紙帶讀取器及打孔機等等) 必須要放在第 2 槽。只有在表 1 的那些板子才能被辨識, 其它型板子雖然也可使用但是必須製造商提供所需軟體。對於外來板子界面的詳細方法, 可參看後面“軟體詳述”及“軟體公用程式手冊”內的“CONFIGIO”諸章節。

## 外部終端機界面使用法

表 1 第 2, 3, 4 類板能夠當作外部終端機及 Apple CP/M 之界面。此終端機界面板必須插在第 3 槽內。

SoftCard 提供 Videx Videoterm 及 M & R Sup-R-Term 24×80 字元螢光幕界面板。其它插入型螢光幕界面板 (video-board) 若製造商有提供該板的軟體的話也可使用。

如果某一界面板插入了第 3 槽, I/O 界面板就成了終端機裝置, 不用 APPLE 自己本身的 24×40 字元螢光幕及鍵盤。如果你有外部終端機界面, 建議你把此板從第 3 槽暫時取出, 先使用 Apple 主機的螢光幕及鍵盤。直到 CP/M 系統被建構妥當, 使它能配合此外來終端機共同使用後再換上。參看“軟體公用程式手冊” CONFIGIO 程式部份。

使用外部終端機時, 建議用① CCS 7710A 串式傳輸界面板或② 修改型 Apple 通信界面板來作終端機與 CP/M 系統的界面。

Apple 高速串式傳輸界面板雖勉強可用但最好不要用，因為CP/M沒有辦法檢查此板的狀態（亦即在一 BASIC 程式之外你不能使用“Ctrl-C”）。

### Soft Card的組立

現在開始組立SoftCard，首先：

```

  ※※※※※※※※※※※※※※※※
  ※  確定APPLE  主機電源已關  ※
  ※  ※※※※※※※※※※※※※※※※

```

如果電源未關就開始組立將會造成APPLE 主機及 SoftCard 的嚴重損害。

1. 拿起板子，把有元件那面朝向你，注意有 4 個小開關在 Apple SoftCard 上。須確定所有開關都在 off 的狀態。就是將開關都撥向鍍金的邊緣連接器（connector.）那邊。此是APPLE CP/M 標準的操作位置。
2. 把Apple主機的鍵盤朝向你，清除主機殼上所放置的螢光幕、碟子或舊咖啡杯等擺飾，然後把頂殼拿開：兩隻手抓住頂殼後邊兩個邊緣，輕輕的往上提直到跳離它的連接器，然後朝向主機後邊提起頂殼，移離主機（再次注意！電源是否關掉？）
3. 現在要決定把SoftCard 放在那一溝槽內，除了第○槽外，你可把它插入任何未使用的槽內，但建議你插在第 4 槽，若此槽已插入磁碟控制板，則改插其它未使用的槽。
4. 把SoftCard 連接器垂直地對著你想要插入的槽，然後把板子推入槽內。當推的時候可前後敲敲以確定它是否插得好，不要使板子傾斜，因為這樣會使連接器末端的指頭和接座連接不正（若







## 第 二 章

# 開 始 使 用 CP/M 系 統

- 啓動 Apple CP/M
- 如何複製 (COPY) SoftCard 磁碟
- 如何創建 CP/M 系統磁碟
- Apple CP/M 與語言板一起使用的方法
- I/O 架構

## 20 Apple II 軟體卡

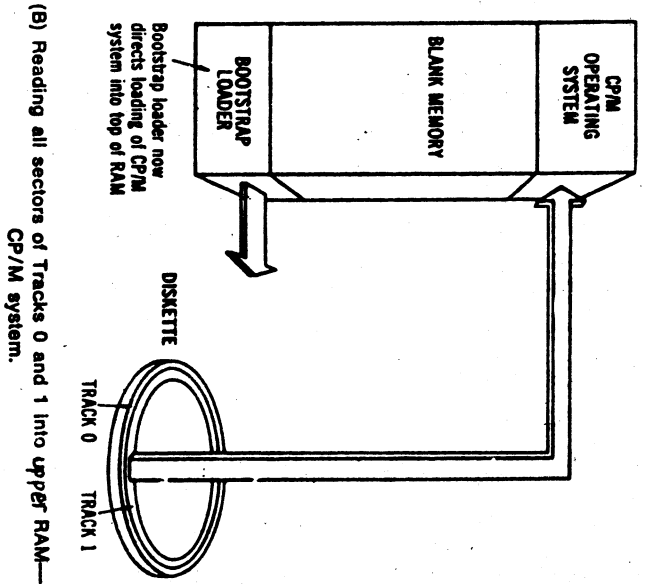
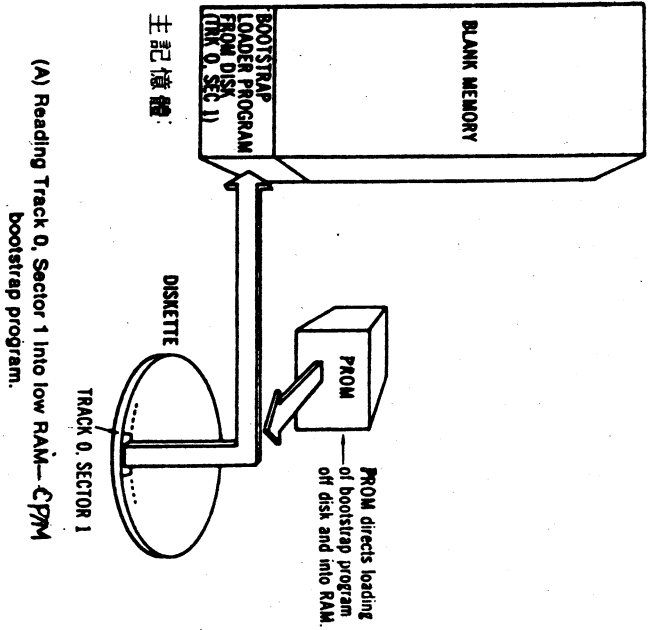
以下將要告訴你怎樣來啓動Apple CP/M。在開啓主機電源前，本章資料務必詳細的閱讀，完全地了解。並且，前章的資料也必須完全明白才可。

### Apple CP/M的啓動

啓動Apple CP/M很簡單，首先必須確定使用正確的磁碟。

SoftCard包裝有兩種磁碟——一種用在16段落格式另一種用在13段落格式。如果你正使用Apple DOS第3.3版或Apple Pascal的語言版，就必須採用16段落(sector)型磁碟，若你是使用Apple DOS第3.3版或更早的版本的話，則需採用13段落型磁碟。—16段落型磁碟在13段落型磁碟機中啓動不起來，反之亦然。

選出適合你系統使用的磁碟，把它插入磁碟機A：中（A：在第6槽的DRIVE 1上）。



- (A) 先由 PROM 內之程式把 CP/M 啟動程式從磁碟載入記憶體
- (B) 再由此 CP/M 啟動程式把 CP/M 程式從磁碟載入記憶體

如何啟動 CP/M 系統

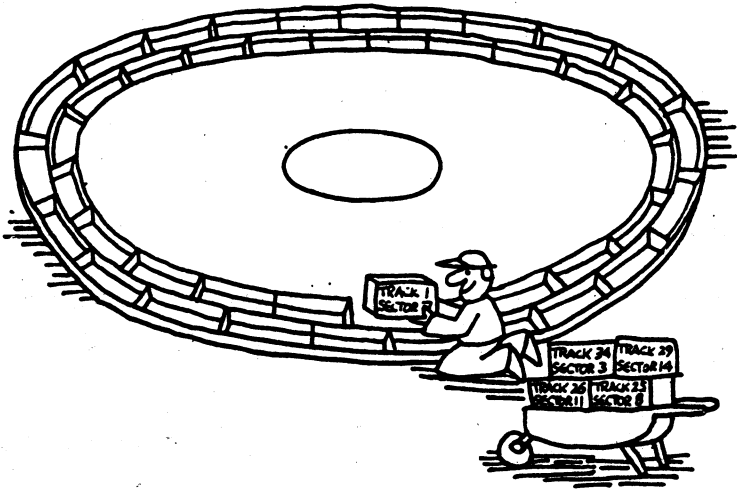


並把原版磁碟放在一安全、乾燥沒有磁場的地方。事實上，製作一片以上的備用磁碟更好。

## 如何複製 ( COPY ) Soft Card 主磁碟

注意：下面的步驟可適用於單一或多磁碟機系統。有關應用 FORMAT 及 COPY 命令 ( 或程式 ) 的方法請參閱“軟體公用程式手冊”。

複製 CP/M 磁碟可分為兩步驟。第 1 步是使用“FORMAT”命令來使一空白磁碟“格式化”這樣以後磁碟才可以做為備用磁碟，因為磁碟需要“格式化”後才可接受資料。第 2 步，使用 COPY 命令來把主磁碟內資料抄入這新“格式化”好的備用磁碟內。



### FORMAT

在使用磁碟之前，先要把它“格式化”

註：CP/M 和 Apple DOS 不同，它並不把磁碟系統軟體程式放在每一片磁碟上。因此 CP/M 的主磁碟與奴磁碟的意義也和 Apple

## 24 Apple II 軟體卡

DOS 不一樣，Soft Card 系統“主”磁碟群是在銷售給你時應該拿它來複製許多備用磁碟後就必須保存及保護的那一些，以後操作時並不拿來使用（使用複製的備用磁碟即可）。除非系統軟體存在特定磁碟上，CP/M 將永遠 BOOT 不進主機記憶體中。因此，使用任一標準 CP/M 磁碟之前必需首先用系統磁碟把 CP/M 載入主記憶體中。

### 備用磁碟的格式化

假定 CP/M 已經載入並開始執行（這時可以看到 A > 催促信號），如果仍有一片 SoftCard 磁碟在磁碟機 A: 內，打入：

FORMAT A: 然後按下 RETURN。不久 Apple 主機會回應：

APPLE II CP/M

xx SECTOR DISK FORMATTER (xx 代表你是使用 13  
(C) COPYRIGHT MICROSOFT 1980 或 16 段落型磁碟)

INSERT DISK TO BE FORMATTED INTO DRIVE A:

此時，把系統磁碟拿出再插入你的空白磁碟。當你準備開始把此空白磁碟格式化時，只要打入 RETURN 即可。注意在敲 RETURN 時放在磁碟機 A: 內的的確是一片空白備用磁碟。

格式化的過程約需 30 秒，整個過程裏磁碟機都會有聲響出現。格式化過程完畢後，磁碟機將會停止，螢光幕會顯示。

FORMAT COMPLETED

INSERT SYSTEM DISK AND PRESS RETURN

當磁碟機上的紅燈熄滅，拿出這片格式化好的磁碟，然後再插入系統磁碟，打入 RETURN 回到 CP/M 控制之中，一兩秒以後 A > 催促信號會出現，它代表你已回到 CP/M 當中了。

### 複製備用磁碟

此時你可開始用 COPY 命令來抄寫 SoftCard 系統磁碟了，打入：

```
COPY A: = A:
```

幾秒後，螢光幕會出現：

```
APPLE II CP/M
xx SECTOR DISK COPY PROGRAM
(C)MICROSOFT 1980
```

```
INSERT MASTER DISK PRESS RETURN
```

因為你想複製的磁碟現已在磁碟機 A 中，只要按下 RETURN 鍵就可開始此過程，磁碟機響了幾秒後，螢光幕會出現：

```
INSERT SLAVE DISK PRESS RETURN
```

拿出系統磁碟後放入格式化好的備用磁碟（到磁碟機 A：）打入 RETURN，幾秒鐘後，螢光幕將顯示：

```
INSERT MASTER DISK
PRESS RETURN
```

拿出備用磁碟再插入主磁碟，打入 RETURN。

以後，螢光幕會再次要求你把主磁碟拿出，奴磁碟插入，這樣的



## 26 Apple II 軟體卡

過程將重覆 3 次。

在最後一次把奴磁碟插入 A: 號磁碟機內時，螢光幕將會顯示：

COPY COMPLETE

DO YOU WISH TO MAKE ANOTHER COPY? (Y/N)

PRESS RETURN

因為現在在磁碟機內的磁碟即為複製完成的 SoftCard 磁碟，你不需再把 SoftCard 主磁碟插入，而可把此片主磁碟保存在一安全、乾燥又沒磁場的地方。

Soft Card 主磁碟最好複製兩份以上當做備用。當你使用其中一片備用磁碟出問題時而又不能馬上斷定是硬體或軟體出毛病，還有第 2 片備用磁碟可供你測試，如此就不必冒險動用原版主磁碟了。

如果你有語言板，又想藉修改 CP/M 來利用額外的語言板記憶體，那麼一定要抄三份以上的副本。建議你只修改副本上的 CP/M 而不要去動原版的主磁碟。

### 複製 CP/M 系統磁碟

一 CP/M 系統磁碟就是當 BOOT 時可載入主記憶體並啓始 CP/M 系統的磁碟。要複製 CP/M 系統磁碟有兩步驟：首先需格式化一片空白磁碟，然後使用 COPY 命令把 CP/M 系統寫入此磁碟內，以下就是複製系統磁碟的過程：

- 1 使用 FORMAT 命令來格式化一空白磁碟，這步驟和以前你準備 COPY SoftCard 主磁碟很接近。
- 2 其次，你必須用 COPY 命令來把 CP/M 系統寫到此格式化好的磁碟上，可使用下面的 “/S” 選擇來完成：

**COPY 命令的使用法：**

1. 把 CP/M 系統磁碟插入磁碟機 A：（此磁碟內含有 COPY 命令的程式）然後 BOOT 此磁碟，當你看到 A > 催促信號時，

打入： COPY A: = A: / S

此“/S”代表你只想抄 CP/M 系統程式而已而非整片磁碟資料，約 1 秒後，螢光幕會顯示：

```
INSERT MASTER DISK PRESS RETURN
```

因為現在磁碟機內的磁碟既有 COPY 程式又有 CP/M，所以只打一 RETURN 即可。此後，磁碟機會吵雜一陣，然後螢光幕會出現：

```
INSERT SLAVE DISK PRESS RETURN
```

此時把你想抄錄 CP/M 系統之格式化好空白磁碟插入後打入 RETURN，不久，磁碟機會停止轉動，螢光幕將顯示：

```
INSERT CP/M SYSTEM DISK INTO DRIVE A:
PRESS RETURN
```

因為現在磁碟機上的磁碟已經是 CP/M 系統磁碟了，只要按下 RETURN 即可返回 CP/M 中。

你的新 CP/M 系統磁碟此時可以 BOOT 也可儲存程式及資料。如果你有一架以上的磁碟機，想同時一次複製一 . 的系統磁碟，可去參考“軟體公用程式手冊”，內 MA 及 COPY 命令的更詳盡介紹。

## APPLE CP/M與語言板一起使用的方法

如果使用語言板的話，就可利用其上 12K 的可定址記憶體。加上這 12K 記憶體後就使得 CP/M 可使用的連續記憶空間達到 56K。但是首先必須修改 CP/M 系統磁碟，使得做 BOOT 磁碟時所載入的是 56K CP/M 而非 44K CP/M，此可由 CPM56 公用程式來完成。

注意：以上述方式修改 CP/M 將不影響 CP/M 的運作，但是，如果系統不具備語言板的話，56K CP/M 無法被 BOOT 進入主記憶體。建議你不要使用原版系統主磁碟來做此種修改，而最好是用備用 CP/M 系統磁碟來做此工作為佳。

要使用 CPM56 公用程式時，首先確定 CP/M 已載入且執行中（你應可看到 A > 催促信號），然後插入備用的 SoftCard 系統磁碟後打入：CPM56 A: 按下 RETURN。此時系統就會自動修改磁碟，當此轉換完成時，螢光幕會顯示：

```
DISK IN DRIVE A: HAS BEEN UPDATED TO 56K
```

現在你就有了一片適用於 56K 記憶體的 CP/M 系統磁碟了，為了重新載入此新版本，可打入 RESET, 6, Ctrl K（如果系統沒有自動啓始 ROM 的話），或把主機關掉再打開也可。不久，催促信號會重新出現，此時螢光幕上會顯示是 56K 的 CP/M 而非 44K CP/M。

## I/O 架構

在組立 CP/M 系統時 I/O 的建構是最後一步，這一步並非所有的系統都需要除非是在以下的情況下：

1. 你想使用額外的終端機。
2. 你希望補些非標準 I/O 所需的軟體到 CP/M 系統。

CONFIGIO 程式可用來處理所有下面所述系統架構問題，更詳細的資料請參閱“軟體公用程式手冊”中 CONFIGIO 使用法。

以下即為此程式所能處理的架構問題：

重新定義鍵盤字元——如果想打入 Apple 鍵盤上（或外部終端機）所沒有的字元，這時就可使用 CP/M 上的 CONFIGIO 公用程式來重新定義鍵盤字元的 ASCII 值。因為 CP/M 程式內有使用到 Apple 鍵盤上所沒有的字元，所以此公用程式可能會對你有所幫助。此方面資料可參看“軟體及硬體詳述手冊”第 2 章及“軟體公用程式”內 CONFIGIO 部份。

載入使用者 I/O 推動器軟體程式——I/O 架構框塊 (block) 提供支援非標準週邊裝置及 I/O 軟體程式。爲了要把非標準週邊裝置（即 11 頁上所未列的裝置）與 Apple 系統相接，你必須把廠商所提供的界面軟體載入 I/O 架構框塊內。但要載入架構框塊的軟體仍然有些限制。這些限制及載入過程可參看“軟體及硬體詳述手冊”第 2 章及“軟體公用程式手冊”內 CONFIGIO 部份。

使外部終端機能與 Apple CP/M 相接用——如果使用外部終端機，必須“設定”Apple CP/M 使它能與你的外部終端機配用。因為 Apple CP/M 提供許多特殊螢光幕及游標控制功能（例如清除螢光幕及定址游標等）給許多 CP/M 程式（例如 Microsoft BASIC

### 30 Apple II 軟體卡

及許多 CP/M word processor 等使用，所以此建構過程是必需的。大多數終端機都具有這些螢幕功能，藉著送出適當的字元序列至終端機，可使它執行適當的功能。因此，必須先使 CP/M 知道這終端機特定螢幕功能命令的字元序列。

CP/M 可接用許多通用的影像終端機如 SOROC IQ 120 / 140, Hazeltine 1500 / 1510 及通用的 24×80 插入型 (plugin) 螢光幕界面板 (video boards) 如 Videx Videoterm 及 M & R Sup-R-Term. 等。

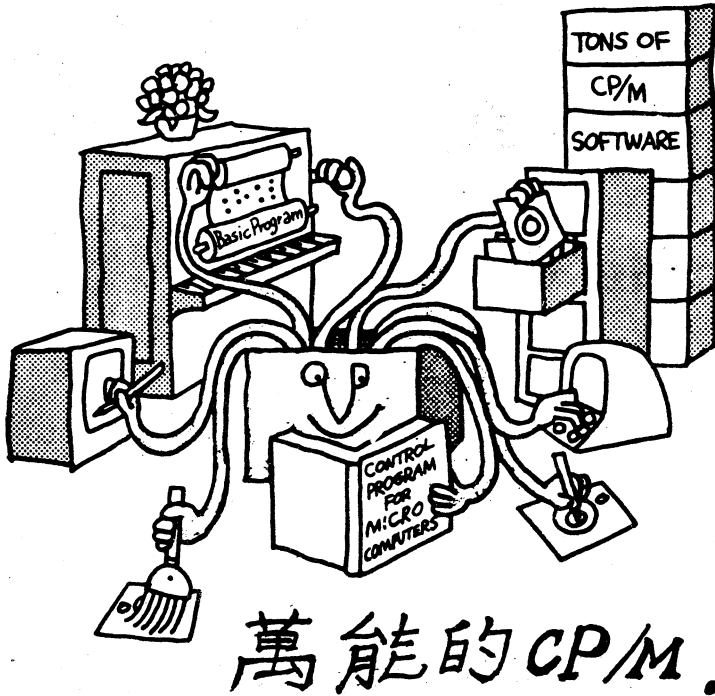
如同前述的軟體板組立中，終端機界面板必需插在 Apple 主機內第 3 槽。詳細資料見 12 頁。

終端機架設的任務可由一個以 Microsoft BASIC 所寫成的程式叫 CONFIGIO 來完成，此程式的使用法及建構 CP/M 的步驟請參閱“軟體公用程式手冊”。

## 第三章

# Apple CP/M 簡介

- 鍵盤的打入
- 輸出控制
- CP/M 的暖啓動：Ctrl-C
- 更換CP/M 磁碟
- CP/M 檔案命名法則
- 檔名稱格式
- 一些CP/M 命令：DIR, ERA, TYPE
- CP/M 錯誤訊息
- 在SoftCard 系統磁碟上的一些程式的功能



## 萬能的CP/M!

本章簡短的介紹APPLE II之CP/M，希望能幫助你開始了解CP/M 是什麼，但想完全了解CP/M 還是得小心研讀CP/M 參考手冊。

CP/M 作業系統的心臟部份並不在於它提供了內藏 (built-in) 的鍵盤命令，而是在於它提供了計算機硬體與軟體間的鏈接，這就是它如此受歡迎的原因——為CP/M 而寫的程式可以很簡單的從一架計算機上換到另一架計算機上去執行。

大多數CP/M 命令事實上都是對應到磁碟上的某一程式（但也有些例外像DIR 命令）且這些程式還具擴展性。爲了要使用這種型態的命令，含有這些命令的磁碟當然需要放在磁碟機內。在使用命令時，需把命令所對應的程式從磁碟載入主記憶體的命令稱爲“暫態命

COPY及FORMAT 命令就是暫態命令。

## 鍵盤的鍵入法

使用CP/M 系統的鍵盤的鍵入法和Integer BASIC 及APPLESOFT 有相當的不同。BACKSPACE 鍵會把游標底下的字元刪除，而FORWARD ARROW (→) 鍵則不具任何功能。本來ESC 鍵所能做的游標移動及編輯功能也不再具有。

但是，CP/M 系統也提供了一些行編輯 (line editing) 功能在做鍵入時很有幫助，還有某些其它重要的控制字元能夠執行很有用的功能〔記住：控制字元 (由Ctrl 來表示) 的鍵入是首先按下CTRL 鍵並在保持按下狀態時再按下另一所要打入的字元〕。

後退 (backspace) 一個字元位置，BACKSPACE 鍵把游標下的字元刪除 (也可用Ctrl-H 來達成)。

- Ctrl-X 後退至一行的開始，刪除此行的所有字元。
- Ctrl-R 把現在指標所指之行重打。
- Ctrl-J 輸入結束，功能和RETURN 鍵類似 (也可用LINE FEED 鍵來達成目的)。
- Ctrl-E 本行到此告一段落，游標跳到下一行開始處，但還沒遇到RETURN 前本行並未結束。
- RUBOUT 刪除並“回印” (echo) 最後字元。也可稱作DEL (DELETE)。(在Apple 鍵盤上打入Ctrl - @ 就可得到RUBOUT，見下面說明)。

在Apple 的鍵盤上有些字元不存在，這些字元都以控制字元來表示：



打入	得到
Ctrl-K	[ (左括號)
Ctrl-@	BUBOUT
Ctrl-B	\ (反 slash)

CP/M 的命令及程式經常需用到這些字元。若要更改 (或去除) 這些安排或想加入新的, 請參閱“軟體公用程式手冊”CONFIGIO 部份。

## 輸出控制

有兩個控制字元可以用來控制輸出到螢光幕或列印機:

- Ctrl-S 輸出字元到終端機的動作暫停, 但當任何字元被敲入時, 程式恢復執行, 字元也恢復輸出。
- Ctrl-P 從螢光幕把所有的字元輸出到列印機及終端機。此種“列印機回印”(printer echo) 一直保持著直到遇見下一個Ctrl-P為止。

## CP/M的暖起動: Ctrl-C

還有另外一個重要的控制字元: Ctrl-C。當打入是在一行第一個字元位置時, Ctrl-C即當作執行CP/M的暖啟動(warm boot), 它使得CP/M從磁碟載入主記憶體內準備工作(此和冷啟動Cold Boot不同, 冷啟動是第一次BOOT CP/M磁碟時稱之)。每當你更換磁碟時, 永遠要記得打入一Ctrl-C(參看下節“更換CP/M磁碟”)。

Ctrl - C      執行 CP/M 之暖啓動

## 更換 CP/M 磁碟

和 DOS 系統不同，CP/M 系統中不能不分青紅皂白的更換磁碟機內之磁碟。當你更換磁碟時，必需讓 CP/M 系統知道。因為有些磁碟索引資料一直都是存於主記憶體內，這些索引可用來配置磁碟的空間位置。當你更換磁碟時，這些索引資料也要隨著更改成新磁碟的索引資料。

要讓 CP/M 知道你更換了磁碟，可在換上新磁碟後，打入 Ctrl - C 做一次暖啓動，如此可使磁碟機的磁碟索引資料更新。你以後需經常使用 Ctrl - C。

如果在更換磁碟之後沒有打入 Ctrl - C 而想對更換後的磁碟做寫入的動作，此時螢光幕會現出：

```
BDOS ERR ON x: DISK R/O (x:代表磁碟機A:~F: ;  
R/O代表唯讀READ ONLY)
```

當你收到以上的錯誤消息時可打入 RETURN。這樣將會再執行一次 CP/M 暖啓動並回到 CP/M 系統中，以上錯誤情況只有在想寫入更換後的磁碟時才會發生，如想讀出更換後磁碟內的資料則不會產生錯誤。

許多 CP/M 程式在程式結束時會執行暖啓動。所以在執行此類程式以後你就不需打入 Ctrl - C，而執行此類程式時可能你會認出做暖啓動時磁碟機所發出之呼嚕聲，這也是判斷一個程式在結束前是否有做暖啓動的方法之一。

更多的資料可在“CP/M 參考手冊”內找到，也可參看本章後面“CP/M 錯誤訊息 (error message)”一節。

## 重置 ( RESET ) 鍵

按下 RESET 鍵會有不同的效果，這與系統主機有無自動啓始 ROM 有關。

如一系統有自動啓始 ROM 的話，①在 CP/M 系統中，按下 RESET 鍵將會使 CP/M 暖啓動，程式控制權也回到 CP/M 中。②在 MBASIC 或 GBASIC 中按下 RESET 將導至“錯誤重置” ( RESET ERROR ) 的結果，使用 ON ERROR GOTO 可將此等錯誤偵出。

如一系統不具自動啓始 ROM 的話，你可藉著打入 Ctrl-Y 後加一 RETURN 來從 RESET 狀態中跳出。此時可能①重新 BOOT CP/M (如果你是在使用 CP/M 系統時誤打入 RESET) 或是②帶著“錯誤重置”的訊息返回 BASIC 程式當中 (如果是使用 MBASIC 或 GBASIC 時誤按 RESET 鍵)。

## CP/M 命令的結構

當看到 A > 催促信號時，你就知道 CP/M 系統在等著你的命令。其中“A”代表磁碟機 A 是“目前標定的磁碟機”。如果命令中未指定磁碟機時，其自然值 (default value) 即為“目前標定的磁碟機”。

CP/M 命令一般來講都非常簡單，有一些常駐在記憶體的命令，其中最有用的是 DIR、ERA、及 REN。DIR 命令用來展示磁碟

檔案目錄，ERA 命令則是用來清除磁碟內之檔案，而REN 命令是用來重新命名一個檔名字。

## CP/M檔案命名規則

在介紹CP/M 命令之前，必須先熟悉CP/M 磁碟檔案命名規則。CP/M 檔案命名規則和APPLE DOS 大不相同。它的檔名稱可以到8個字元長，並可自由選擇3個延伸(extension)字元，這在讓你標明磁碟上相關檔案時相當便利。

### 檔名稱格式

CP/M 檔名稱的格式結構讓你只用單一的指定就參考到一個或多個檔。檔名稱通常是一個名字(可到8個字元長)後接著一句點(.)及3個延伸字元。也可以指定檔案所在磁碟之磁碟機位置，這只要在檔名稱之前接上磁碟機名稱就可辦到。如不指定的話，系統就當做是“目前所標定的磁碟”。以下是一些有效的檔名稱：

A: FNAME.EXT 參考到磁碟機A：內的FNAME.EXT 檔。

TEMP.DLD 參考到目前所標定磁碟機內磁碟的TEMP  
.DLD檔。

B: TEMP.NEW 參考磁碟機B：內的TEMP.NEW。

3個延伸字元經常提供了檔的內部格式之情報。最重要的延伸字元有①COM,代表COMMAND。任何檔名字若帶有COM的話都是暫存命令型檔，使用時只要打入檔名字即可(而不需打入COM)；②BAS,代表BASIC程式及③HEX,ASM和PRN,這些俱為ASM程式(即CP/M 8080 組合程式)所使用及產生。

檔名稱結構中也有可以參考一個檔以上的結構存在，此乃使用 wild card 型檔名稱。其中問號 ( ? ) 可代表 wild card 字元，意即當搜索檔名稱索引來找出相同的檔名稱時它能取代相關位置上的任何字元。而星號 ( \* ) 則可代表檔名稱的任何字串 ( 看下面所舉例子就可明白 )。例如：

B: TEMP.??? 或 B: TEMP.\*

就可參考到磁碟機 B: 上的 TEMP.OLD 及 TEMP.NEW 兩檔，下面還有一些例子說明 wild card 型檔名稱：

- |                   |                              |
|-------------------|------------------------------|
| A: *.COM          | 參考到磁碟機 A: 內所有具有延伸字元 COM 的檔案。 |
| B: *.*            | 參考到磁碟機 B: 內所有的檔。             |
| B: ??????????.??? | 參考到磁碟機 B: 內所有的檔。             |
| DUMP.*            | 參考到目前標定磁碟機內所有以 DUMP 為名字的檔。   |
| C*.*              | 參考到目前標定磁碟機內所有以 C 字母開頭的檔。     |

注意：一“\*”號相當於一串的“?”號。

### 一些 CP/M 命令：DIR, ERA, REN 及 TYPE

以上是 4 種最常用的 CP/M 命令。DIR 用來展示一磁碟內所有檔案名稱的索引；ERA 用來清除磁碟上某一檔；REN 則用來重新命名磁碟檔；而 TYPE 是用來展示在終端機的某一主文檔 ( text file )。以下就是這些命令簡短的介绍。

註：以下的資料僅為簡介而已，更完整及詳盡的資料請參閱“CP/M 參考手冊”。

## DIR 命令

DIR 命令用來展示磁碟上所有檔的名稱，如想展示目前所標定磁碟機內磁碟所有的檔名稱，可打入：DIR 後 RETURN。如要展示別的磁碟機內所含檔的索引，只要加上磁碟機的名字，如：

DIR B: 即可。

如果 DIR 後接著檔名稱的話，只有檔名稱相同的檔會被展示出來，以下就是一些例子：

DIR MBASIC.COM	如果此檔存在於目前所標定的磁碟機中則螢光幕會展示出 MBASIC.COM
DIR A: *.COM	展現在磁碟機 A: 中所有帶有 COM 延伸字元的檔案名稱。
DIR B:	展現在磁碟機 B: 內中所有的檔案名稱。
DIR A: A*.*	展現所有在磁碟機 A: 內開頭是 A 的檔案名稱。

如果要展現的檔找不到的話，CP/M 系統會回答：

NO FILE

## ERA 命令

ERA 命令可用來清除磁碟上的檔，此命令必須附加有要清除檔的名稱。

注意：不要刪除 CP/M 系統磁碟上的檔，假如這樣做了的話趕快再從 Softcard 主磁碟上複製另一份備用 CP/M 系統磁碟。

下面是一些使用 ERA 命令的例子：

- ERA B:TEMP.OLD 清除磁碟機 B：中的名叫 TEMP.  
OLD 檔。
- ERA C:\*.BAK 清除磁碟機 C：中所有含有 BAK 延  
伸字元名稱的檔。
- ERA \*.\* 清除目前所標定磁碟機中所有的檔，  
如果你想清除磁碟上所有的檔，CP  
/M 會問 ALL ( Y/N )？如不想，  
趕緊打入 N。

注意，一 ERA 命令可藉著使用 wildcard 型檔名稱一次清除很多檔。

### REN (重新命名) 命令

REN 命令可用來重新命名一個檔，其格式如下：

REN 新名字 = 舊名字

在 REN 命令中不能使用 wild card 型檔名稱，你可在檔名稱之前放入所指定之磁碟機名字。以下是一些例子：

REN TEMP.NEW = TEMP.OLD 把本名叫做 TEMP.  
OLD 的檔改名為  
TEMP.NEW.

REN B: PEAR.COM=APPLE.COM 把磁碟機B：內本名叫  
APPLE.COM 改稱做  
PEAR.COM。

注意：和 Apple DOS 不同，本系統REN 命令需把新名稱放在  
前面舊名稱放在後面，並以等號“=”相連。

### TYPE命令

TYPE命令是用來展現終端機內的主文檔，必須附有檔名稱始可  
(wild card 型檔名稱則不被允許)。

例如，要在螢光幕上展現DUMP.ASM檔的內容，可打入：

TYPE DUMP.ASM, 然後RETURN. 如果此檔不是主文檔，則  
螢光幕上將報以一些亂七八糟字元。

注意：DUMP.ASM是SoftCard 系統主磁碟上唯一的主文檔。

### CP/M之錯誤訊息

有4種CP/M 錯誤訊息，下面分別列出，並伴隨著造成錯誤的  
可能原因，這些原因依照著可能性的大小而排列。

BDOS ERR ON x: BAD SECTOR

(x: 代表磁碟機名稱A: ~ F:)

造成此錯誤訊息的原因有許多種——並不一定是你的磁碟上有一  
段落損壞了(當然也可能是)，此錯誤訊息與APPLE DOS 內的  
DISK I/O ERROR”頗像，造成它的原因有：



## 42 Apple II 軟體卡

1. 磁碟機內無磁碟。
2. 磁碟機之門沒關好。
3. 磁碟插入不正確。
4. 試圖到一沒有接控制板的磁碟機內去存取資料。
5. 磁碟損壞。

當你接到 BAD SECTOR 錯誤訊號時，CP/M 等著你從鍵盤上打入一個字元，如果打入 Ctrl - C，系統將會執行暖啓動而回到 CP/M 系統上去，打入 R 代表再試一次讀寫動作並繼續執行。其他任何其它字元的輸入都會使此錯誤訊息被忽略並繼續執行程式或動作。

BDOS ERR ON x: R/O

( x: 代表磁碟機名稱 A: ~ F: )

此訊息經常代表兩種意義：

1. 你更換了磁碟機內之磁碟而忘了打入 Ctrl - C。
2. 有一禁寫入封籤蓋住了磁碟上禁寫入之缺口。

當你遇到此訊息，CP/M 也停下等你從鍵盤上輸入一字元，輸入一字元後，系統會執行暖啓動並回到 CP/M 系統中。

BDOS ERR ON x: FILE R/O

( x: 代表磁碟機名稱 A: ~ F: )

此訊息只代表一件事：

1. 試圖對一個已被 STAT 程式標記為唯讀 ( READ NLY ) 狀態的檔案做寫入資料的動作。

遇到此訊息，CP/M 等你打入任一字元後執行暖啓動，並回到

CP/M 系統中。

BDOS ERR ON x: SELECT

(x: 代表磁碟機名稱 A: ~ F:)

此訊息也只代表一件事：

1. 試圖到某一不存在的磁碟機內去存取資料。

遇到此訊息，CP/M 會等你打入任一字元後執行暖啓動並回到 CP/M 系統中。

注意：假如你只有一部磁碟機連接到系統上（即單磁碟機系統），若想到某一不存在的磁碟機內存取資料，那麼所得到的錯誤訊息將是 BAD SECTOR 訊息而非 SELECT 訊息。

## 在Soft Card系統磁碟上所包含的程式

MBASIC，GBASIC 及許多公用程式都存放在 SoftCard 系統磁碟內，以下是這些程式的摘要，並附有何處可找到這些程式完整敘述的參考資料。

**APDOS** 此公用程式可使你從 Apple DOS 磁碟把資料傳至 CP/M 磁碟。APDOS 只能傳主文檔 (text file) 及二進位型檔 (執行此命令需兩架以上磁碟機)。詳細資料參閱“軟體公用程式手冊”

**ASM** ASM 是 CP/M 8080 組合語言編譯程式。ASM 與 DDT 合用可用來撰寫及除錯 8080 組合語言程式。詳見“CP/M 參考手冊”第 4 章

**CONFIGIO CONFIGIO** 公用程式是用來把 CP/M 作業環境

建構以配合你的特殊系統架構。它有4個主要功能—①為外來終端機建構 I / O ; ②重新定義鍵盤字元 ; ③載入使用者 I / O 軟體及④讀出或寫入 I / O 架構框塊 ( configuration Block ) 。要了解更多的 I / O 架構的功能可參看“軟體公用手冊”及“軟體及硬體詳述手冊”第2章。

**COPY** COPY 程式可用來複製 CP / M 磁碟，即把 CP / M 系統程式抄入格式化好的空白磁碟，使其變成 CP / M 系統磁碟。參看“軟體公用程式手冊”

**DDT** DDT 是 CP / M 動態除錯工具。利用它可以做動態交談式測試及 8080 組合語言程式的除錯。見“CP/M 參考手冊”第5章。

**DOWNLOAD** DOWNLOAD及**UPLOAD** 公用程式使得使用者能把 CP / M 檔案從另一架使用 CP / M 的機器以 RS 232 串式鏈接傳送到 APPLE 主機。但**UPLOAD** 在傳送端及接收端的磁碟內現都沒有。使用這程式需有 8080 組合語言程式工作經驗，因此只適合有經驗程式員使用。詳見“軟體公用程式手冊”。

**DUMP** DUMP 將磁碟檔的內容以 16 進位型式展示在螢光幕上。DUMP.ASM是DUMP程式之源程式列印 ( source listing ) ，見“CP/M 界面指引手冊 ( CP / M INTERFACE GUIDE ) ”第2章，在那章裏，此程式列印是被當作一個例子來說明 8080 組合語言在 CP / M 環境下的寫法。

- 詳見“CP/M參考手冊”第1章及第2章。
- ED** ED 是 CP/M 的主文編輯程式 (text editor)。用來新建及編輯主文檔 (text files)。見“CP/M參考手冊”第3章。
- FORMAT** FORMAT 可用來格式化一片空白磁碟，使它開始能接受資料。一片剛格式化磁碟不能做 BOOT 的動作，只能用來存程式及資料。可使用 COPY 程式把一格式化好的磁碟複製成 CP/M 系統磁碟。詳見“軟體公用程式手冊”。
- LOAD** LOAD 可將帶有 .HEX 延伸字元的磁碟檔轉換成機器可執行的帶有 .COM 延伸字元的檔。LOAD 可將組合語言編譯程式之輸出轉換成可執行的機器碼。詳見“CP/M參考手冊”第1章。
- MBASIC** 此即 Microsoft BASIC。這種版本的 BASIC 是一種磁碟上的 BASIC，它能提供在 APPLESOFT 內所沒有的低解析度繪圖功能、聲音及遊戲的控制 (game control)。但此種版本不能提供高解析度繪圖能力。詳見“Microsoft BASIC 參考手冊”
- PIP** PIP 是 CP/M 內最常用到的程式之一。它可將一片磁碟上的檔轉移到另一片磁碟上去。也可用來複製及加長磁碟檔。PIP 也可用來將檔傳送到終端機裝置及列印機。詳見“CP/M參考手冊”第1章。若要複製整片磁碟，或僅將 CP/M 系統磁碟複製到另一片，可參

見“軟體公用程式手冊”內之COPY。

**STAT** STAT可提供磁碟使用狀態一般資料如下：①磁碟容量；②檔案大小；③檔指示器；④裝置安排情况等。可用此程式來更改檔指示器及裝置安排等。詳見“CP/M參考手冊”第1章。

**SUBMIT** SUBMIT 使得系統執行從磁碟檔上輸入的命令與程式而非從鍵盤上輸入的命令，這樣可達自動處理之目的。詳見“CP/M參考手冊”第1章

**XSUB** 當XSUB 與SUBMIT 共用時，在執行程式的任何時間內，允許從磁碟檔輸入任何字元。詳見“CP/M 參考手冊”第1章。

以下三程式只存在於16 段落 ( sector ) 型 SoftCard 系統磁碟

**CPM56** 當44K CP/M 系統上有語言板時，可用CPM56 來把系統變為56K CP/M 系統 ( 44K 再加上語言板上之記憶體 )。注意，CPM56 程式不能用在48 K APPLE 系統上。

詳見“軟體公用程式手冊”

**GBASIC** GBASIC 除了能夠提供高解析度繪圖能力外和MBASIC 完全相同。詳見“BASIC 參考手冊”

**RW13** RW13可使16 段落型CP/M 系統能存取13 段落型CP/M 系統磁碟上面的檔，若RW13與PIP 合用則可將13 段落型磁碟上之檔傳送到16 段落磁碟上去 ( 但需用兩架以上的磁碟機 )。

見“軟體公用程式手冊”

# 第 四 章

## Microsoft BASIC初 步

## 48 Apple II 軟體卡

一旦你複製好 SoftCard 系統磁碟後，你一定急著想探索 Microsoft BASIC。如前所述，SoftCard 包裝內有兩種 BASIC 版本：

**MBASIC** 包括所有的 Microsoft BASIC 第 5.0 版再加上低解析度繪圖能力及其它比 Applesoft 多出來的功能（MBASIC 及 Applesoft 之比較可參考“BASIC 參考手冊”）。MBASIC 在 13 段落型及 16 段落型磁碟上皆可儲存，其檔名稱爲 MBASIC.COM。

**GBASIC** 包含 MBASIC 的所有功能再加上高解析度繪圖能力，GBASIC 只能儲存在 16 段落磁碟上，其檔名稱爲 GBASIC.COM。

要啓動 MBASIC 或 GBASIC，必須先在 CP/M 命令階層裏（即螢光幕上有 A > 催促信號時），如果沒有看到此催促信號，回頭看看前面章節，再重新載入 CP/M 到記憶體一次。

下面是 MBASIC 之啓始指令，但也適用於 GBASIC，只要把所有的 MBASIC 改成 GBASIC 即可。除了 GBASIC 具高解析度繪圖能力外，這兩種 BASIC 的使用法相同。

看到 A > 催促信號時，打入： MBASIC 然後按下 RETURN，螢光幕會顯出：

```
BASIC-80 Version 5.xx
APPLE CP/M Version
Copyright ©1980 by Microsoft
Created: dd-mm-yy
```

xxxx Byte Free

Ok

此時 BASIC 已準備好接受命令了。

以此法啓始後，BASIC 系統會設定一些自然值 ( default value ) : 在執行一 BASIC 程式任何時間內可有 3 個檔爲其所開，在 FDOS 起始點以前的所有記憶體都可爲其所用，每個資料錄 ( record ) 最長可達 128 拜。

如果你不想要用自然值而希望自己設定這些參數 ( 在 “ Microsoft BASIC 參考手冊 ” 內有更詳細的介紹 ) ，你可在打入啓始命令時設定幾個參數，你也可在啓始命令後加入程式名稱，使得啓始命令執行啓始動作完後立即執行此命令。此可加於啓始命令後的格式如下：

MBASIC [ < 檔名稱 > ] [ /F: < 所開檔之數目 > ] [ /M: < 記憶體最高的位址 > ] [ /S: < 最大資料錄長度 > ] 。

( 方括弧 [ ] 表示此項可以選用或不用，角括弧 < > 代表所要接的項目性質 ) 。

先舉幾個例子說明：

A> MBASIC PAYROLL.BAS      代表使用 FDOS 啓始點以下的所有記憶體空間，載入並執行 PAROLL.BAS 程式，可以開 3 檔使用。

A> MBASIC INVENT /F:6      代表使用 FDOS 啓始點以下的所有記憶體空間，載入並執行



INVENT .BAS 程式，可以  
開 6 檔使用。

A> MBASIC /M: 32768

代表使用 32K 記憶體空間及可以  
開 3 檔使用。

A> MBASIC DATAK /F: 2 /M: &H9000

代表使用記憶體開頭 36K 部份，  
載入並執行 DATAK.BAS，  
可以開 2 檔使用。

### MBASIC 命令格式的說明：

<檔名> ( <Filename > ) 的選定使得啓始 MBASIC 之後自動執行此檔。若檔名短於 9 個字元且沒有延伸字元，則自然設其延伸字元為 .BAS。 / F : <檔的數目> 設定執行 BASIC 程式時可以開檔的檔數。開檔的每檔資料塊佔記憶體 166+128 (或 / S : 定之值 ) 拜。 <檔的數目> 可以是 10 進位數，8 進位數 ( 前加 & 0 ) 或 16 進位數 ( 前加 & H )。

/M: <最高的記憶體位置> 的選定 是用來設定 MBASIC 可以使用的最高記憶體位置。在一些情況下，記憶體的數量最好低於 CP/M 之 FDOS，以便保留一些空間給組合語言副常式所使用。而在任何情況下，最高記憶位置也必須低於 FDOS ( 它的位址在第 6 位置及第 7 位置 )。 <最高的記憶體位置> 可以是 10 進位數，8 進位數 ( 前加 & 0 ) 或 16 進位數 ( 前加 & H )。

/S: <最長資料錄> 的選定 用來設定隨機檔所能允許的最大長度。可以指定任何整數，即使大於 128 的整數也可以。

當 BASIC 啓始後，螢光幕會顯出 “Ok” 的催促信號，“Ok” 代表 BASIC 已在命令層次內，此即它已準備好接受命令了，此時就可以輸入直接模式或間接模式型的命令了。

做Microsoft BASIC程式與做Applesoft 程式大致相同，只是前者更明顯的具有威力，詳見“Microsoft 參考手冊”。

到此完成了本手冊的組立及操作部份。此時，你應可把你的SoftCard裝好且把CP/M 及BASIC磁碟複製好，並可把系統載入與執行。本手冊其它部份都有更詳盡的資料能適合你的需要，希望你能參閱它並從其中發現使用SoftCard 的樂趣。

# 第二部份

## 軟體與硬體

### 詳述

# Software and Hardware Details

# 第 一 章

## APPLE II CP/M軟體詳述

- 簡介
- I/O 硬體規約
- 6502/Z-80 位址轉換
- APPLE II CP/M 記憶體使用法
- 使用 SoftCard 時組合語言之撰寫
- ASCII字元碼
- 中斷處理

本章說明 APPLE II CP/M 系統軟體的特性，以及它們與組立在不同溝槽內的硬體間相互的關係。首先要討論 CP/M 系統的硬體 I/O 規約 ( protocol )，然後再討論這些硬體規約的軟體支援：I/O 架構框塊 ( I/O Configuration Block )。有關 CP/M 作業系統之使用方法，請參閱“ CP/M 參考手冊”

## I/O 硬體規約

除了一部份例外，I/O 硬體規約和 APPLE PASCAL 剛出版時是完全一樣。所有標準的 APPLE I/O 週邊裝置都可適用於此系統，並且還有一些其它的，像 California Computer System' 7710A 非同步串式傳輸界面，Videx Videoterm，M&R 企業 Sup-R-Term。但 APPLE CP/M 在 24×40 的螢光幕上不提供水平繞捲 ( scrolling ) 的功能。

Apple 週邊設備界面卡：分別插在何處

不像 Applesoft 與整數 BASIC ( 但與 Apple PASCAL 類似 )，Apple CP/M 需要週邊設備 I/O 卡，這些 I/O 卡視功能而定分別插入不同的槽內。例如，列表機界面卡必須插入第一槽內，這樣才能夠使用列表機。當 CP/M 系統被啟動時，就能夠辨認是否具有某一標準 Apple 週邊設備界面卡。一旦 CP/M 系統被啟動後，即可直接利用硬體方式或者呼叫界面卡上的 6502 軟體來執行 I/O 的動作。

下表列出各槽的功能以及所承認的界面卡（表中只列出各界面板的型式，至於各型式所包括的界面卡則示於表後）。注意：除了表後的附註外，不承認的界面卡或空槽都沒作用。

槽號	可插入的界面卡型式	用途
0	不供 I/O 使用	此槽可插入語言卡 ( Language Card ) 或 Applesoft 或 整數 BASIC ROM 卡 ( 後兩個 CP/M 用不到 ) 。
1	2, 3, 4	列表機界面 ( CP/M LST: 裝置名稱 )
2	輸入: 2, 3, 4 輸出: 1, 2, 3, 4	一般用途 I/O ( CP/M PUN: 與 RDR: 裝置名稱 ) 。
3	2, 3, 4	控制台輸出裝置 ( CRT: 或 TTY ) 。若此槽內未插入界面卡，則以一般 Apple 的 24 × 40 螢幕做為 TTY 裝置。
4	1	磁碟機 E: 與 F: 的控制卡
5	1	磁碟機 C: 與 D: 的控制卡
6	1	磁碟機 A: 與 B: 的控制卡 ( 一定要有的 )
7	任何界面卡	未指定用途。Softcard 可插在此槽內。

附註: Softcord 可插在任意空槽內，除了第 0 槽外。

以下列出目前 Apple CP/M 所承認的 I/O 週邊設備界面卡型式。

型式 界面卡名稱

- 1 Apple Disk II 控制卡
- 2 Apple 訊息交換界面卡  
\*California Computer Systems (ccs) 7710 A  
串列界面卡
- 3 Apple 高速串列界面卡  
Videx Videoterm 24×80 視頻終端機界面卡  
M&R Enterprises Sup-R-Term 24×80 視頻終端機  
界面卡
- 4 Apple 並行列表機界面卡

\*

\*CCS 7710A 串列界面卡是較理想的型式 2 界面卡，因為它具有硬體連繫交換 (handshaking) 的能力，而且鮑率 (baud rate) 可以調整，範圍是 110 ~ 19200 鮑。

## 6502/Z-80位址轉換

因為在 SoftCard 硬體記憶體位址之轉換不同，對 6502 及 Z-80 微處理機要存取相同的資料但卻需指定不同的位址，Z-80 及 6502 記憶體位址對應列於下表 (裏面所有數字都以 16 進位表示)。轉換 6502 BASIC 程式或組合語言軟體供 Soft Card 使用時就相當的有用。

Z-80 ADDRESS	6502 ADDRESS	
0000H-0FFFH	\$1000-\$1FFF	Z-80 location zero
1000H-1FFFH	\$2000-\$2FFF	
2000H-2FFFH	\$3000-\$3FFF	
3000H-3FFFH	\$4000-\$4FFF	
4000H-4FFFH	\$5000-\$5FFF	
5000H-5FFFH	\$6000-\$6FFF	
6000H-6FFFH	\$7000-\$7FFF	
7000H-7FFFH	\$8000-\$8FFF	
8000H-8FFFH	\$9000-\$9FFF	
9000H-9FFFH	\$A000-\$AFFH	
0A000H-0AFFFH	\$B000-\$BFFF	
0B000H-0BFFFH	\$D000-\$DFFF	
0C000H-0CFFFH	\$E000-\$EFFF	
0D000H-0DFFFH	\$F000-\$FFFF	6502 RESET, NMI, BREAK vectors
0E000H-0EFFFH	\$C000-\$CFFF	6502 memory mapped I/O
0F000H-0FFFFH	\$0000-0FFF	6502 zero page, stack, Apple screen

## APPLE II CP/M記憶體之使用

6502 ADDRESS	Z-80 ADDRESS	PURPOSE
\$800-\$FFF	0F800-0FFFFH	Apple disk drivers and disk buffers
\$400-\$7FF	0F400-0F7FF	Apple screen memory
\$200-\$3FF	0F200H-0F3FFH	I/O Configuration Block.
\$000-\$1FF	0F000H-0F1FFH	Reserved 6502 memory area - 6502 stack and zero page.
\$C000-\$CFFF	0E000H-0EFFFH	Apple memory mapped I/O
\$FFFA-\$FFFF	0DFFAH-0DFFFH	6502 RESET, NMI, and BREAK vectors.
\$D400-\$FFF9	0C400H-0DFF9H	56K Language Card CP/M (if Language Card installed)
\$D000-\$D3FF	0C000H-0C3FFH	Top 1K of free RAM space with 56K Language Card CP/M
\$A400-\$BFFF	9400H-0AFFFH	44K CP/M. (Free memory with 56K CP/M)
\$1000-\$A3FF	0000H-093FFH	Free RAM (CP/M uses lowest 256 bytes)



## 使用Soft Card時組合語言的撰寫

Z-80 處理機可執行所有 8080 的指令再加上 Z-80 本身所多出之指令，因此在 SoftCard 上你能執行以 8080 或 Z-80 組合語言所寫成的軟體。只是，Z-80 和 8080 指令的簡名 ( mnemonics ) 不相同。

在標準 CP/M 公用程式內包括有：

ED 以列為單位的編輯程式。

ASM 8080 組合語言編譯程式 ( Assembler )。

DDT 8080 機械語言偵錯程式。

這些程式可用來編寫及修改 8080 程式。

在 SoftCard 上也可叫到 6502 的副常式。Microsoft 組合語言發展系統可個別的用來發展 Z-80 或 6502 的軟體。

## ASC II 字元碼

DEC = ASCII 10 進位碼

HEX = ASCII 16 進位碼

CHAR = ASCII 字元名稱

十進位碼	十六進位碼	字元名稱	按什麼鍵
0	00	NULL	ctrl @
1	01	SOH	ctrl A
2	02	STX	ctrl B
3	03	ETX	ctrl C
4	04	ET	ctrl D
5	05	ENQ	ctrl E
6	06	ACK	ctrl F
7	07	BEL	ctrl G
8	08	BS	ctrl H or ←
9	09	HT	ctrl I
10	0A	LF	ctrl J
11	0B	VT	ctrl K
12	0C	FF	ctrl L
13	0D	CR	ctrl M or RETURN
14	0E	SO	ctrl N
15	0F	SI	ctrl O
16	10	DLE	ctrl P
17	11	DC1	ctrl Q
18	12	DC2	ctrl R
19	13	DC3	ctrl S
20	14	DC4	ctrl T
21	15	NAK	ctrl U or →
22	16	SYN	ctrl V
23	17	ETB	ctrl W
24	18	CAN	ctrl X
25	19	EM	ctrl Y
26	1A	SUB	ctrl Z
27	1B	ESCAPE	ESC
28	1C	FS	ctrl [
29	1D	GS	ctrl shift-M
30	1E	RS	ctrl
31	1F	US	ctrl _
32	20	SPACE	space
33	21	!	!
34	22	"	"
35	23	#	#
36	24	\$	\$
37	25	%	%
38	26	&	&
39	27	'	^
40	28	(	(
41	29	)	)
42	2A	*	*
43	2B	+	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	0

60 Apple II 軟體卡

49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	:	:
59	3B	;	;
60	3C	<	<
61	3D	=	=
62	3E	>	>
63	3F	?	?
64	40	@	@
65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	[	[
92	5C	\	\
93	5D	]	](shift-M)
94	5E	.	.
95	5F	-	-
96	60	'	'
97	61	a	a
98	62	b	b

99	63	c	c
100	64	d	d
101	65	e	e
102	66	f	f
103	67	g	g
104	68	h	h
105	69	i	i
106	6A	j	j
107	6B	k	k
108	6C	l	l
109	6D	m	m
110	6E	n	n
111	6F	o	o
112	70	p	p
113	71	q	q
114	72	r	r
115	73	s	s
116	74	t	t
117	75	u	u
118	76	v	v
119	77	w	w
120	78	x	x
121	79	y	y
122	7A	z	z
123	7B	{	{
124	7C		
125	7D	}	}
126	7E	~	~
127	7F	RUB	

## 中斷處理

因為 SoftCard 利用 Apple 匯流排的 DMA 控制線使 6502 處理機進入停頓狀態，所有的中斷處理必需經由 6502。中斷可以在兩種時段中發生：一是在 Z-80 模式另一則在 6502 模式。

### 6502 模式的中斷處理

與一般方法處理中斷一樣——即遇到 RTI 指令時停止中斷處理跳回原程式。

### Z-80 模式的中斷處理

當一中斷信號發生在 Z-80 模式時，Z-80 及 6502 處理機都會

被中斷，以下就是在此模式下中斷處理的步驟：

1. 把任何可能遭到破壞的暫存器資料放到堆疊上去。
2. 如果一中斷信號發生在執行 6502 副常式呼叫時，保存 6502 副常式呼叫位址之內容。
3. 設定 6502 副常式呼叫位址為 \$FF58，這是 APPLE Monitor ROM 上 6502 RTS 指令之位址。
4. 執行寫入到 SoftCard 位址的動作把控制權歸還給 6502。
5. 在控制權歸還給 Z-80 後，要把 6502 先前副常式呼叫位址再存回。
6. 存回所有在堆疊上 Z-80 暫存器之資料。
7. 用 EI 指令恢復接受中斷信號能力。
8. 用 RET 指令回到原程式上。





## 第 二 章

# APPLE II I/O 架 構 框 塊

- 簡介
- 主控制台游標定址 / 螢光幕控制  
硬體 / 軟體螢光幕功能表  
與終端機無關之螢光幕功能 / 游標定址
- 鍵盤字元之重新定義
- 非標準週邊裝置及 I/O 軟體支援  
把邏輯名字指定給實體 I/O 裝置：IOBYTE  
經由 I/O 向量表修補使用者軟體
- 呼叫 6502 副常式
- 週邊板的存在與位置之指示



## 簡介

I/O 架構框塊 ( Configuration Block ) 包含有 Apple CP/M 與許多 APPLE 系統可使用的硬體及軟體間界面之有關資料。每一 Apple CP/M 之系統磁碟都有自己的 I/O 架構框塊。當 BOOT 系統時，這些框塊也同時被啓始及載入。

I/O 架構框塊有 5 種主要的功能：

1. 控制台游標定址及螢光幕功能界面。
2. 鍵盤字元之重新定義。
3. 非標準週邊裝置及 I/O 軟體之支援。
4. 6502 副常式之呼叫。
5. 週邊板的存在與位置之指示。

這些功能將在以下各節詳述。

註：CONFIGIO 程式可用來檢查及修改 I/O 架構框塊，詳見第 5 部份“軟體公用程式手冊”。

## 主控制台游標定址／螢光幕控制

大多數螢光幕終端機，包括正常的 20 × 40 APPLE 螢光幕，都能提供如下的功能：直接定址游標，清除畫面，對各特定內容以不同的色調強化顯示出來等。Apple CP/M 的應用軟體，如文字處理程式及商業軟體都可輕易的利用這些功能。

控制這些高級螢光幕功能必需要送一定序列的字元到終端機去，對某些功能不同種類的終端機需輸入不同序列的字元。應用到這些功能的大部份應用軟體都可適用於許多常用的終端機。但是若你的終端

機不能和這些軟體相配合，此時就必須自己寫特別的機器語言來解決這個問題。由於 Apple PASCAL 所使用的 Datamedia 螢光幕功能及 24×80 插入式螢光幕界面板在 CP/M 應用程式員眼中看來並不普遍，因此很少有軟體支援。

在多數的情況下，這些問題可以經由翻譯的方法把接收進來的功能翻譯成所希望的終端機功能。這工作可由兩張翻譯表來達成：一是軟體螢光幕功能表另一則為硬體螢光幕功能表。Apple CP/M 用軟體螢光幕功能表來辨識所送入要求螢光幕功能之字元序列，然後把它翻譯對應到硬體螢光幕功能表內之字元序列，這些字元序列其後被用來控制終端機。

例如，假如你想在 Videx Videoterm 24×80 螢光幕界面板上處理（用於 SOROC IQ 120 終端機的）CP/M 文字處理程式，則產生了一個問題：因為 Videoterm 只辨認得出 Datamedia 型終端機的命令字元序列，對於 SOROC 終端機所送過來的螢光幕功能字元序列，Videoterm 則毫無辦法接受。

要解決此問題可利用 CONFIGIO 程式將 SOROC 字元序列編入軟體螢光幕功能表內，而將 Videoterm 終端機所能辨識的 Datamedia 字元序列編入硬體螢光幕功能表內。此後，當文字處理程式送字元序列給終端機時，此字元序列先經轉換變成終端機所能辨識的字元序列，如此終端機才能執行此序列之功能。

## 硬體／軟體螢光幕功能表

APPLE CP/M 提供 9 種螢光幕功能：

1. 清除螢光幕。

## 68 Apple II 軟體卡

2. 清除螢光幕至本頁末。( Clear to End of Page )。
3. 清除螢光幕至本列末。
4. 設定主文正規顯示模式 ( Normal lowlight text mode )。
5. 設定主文反相顯示模式 ( Inverse highlight text mode )。
6. 移游標至螢幕起始位置。
7. 游標定址。
8. 游標回復原位。( Home Cursor )
9. 非破壞性游標前移。

後退 ( backspace ) 字元 ( ASCII 8 ) 用來使游標倒退一位，饋行 ( Line Feed ) 字元 ( ASCII 10 ) 則可使游標往下移一行。

APPLE CP/M所提供的螢光幕功能字元序列有兩種形式：

1. 單一控制字元
2. 單一領入 ( lead-in ) 字元後接著一ASCII字元

並沒有提供超過兩個字元的螢光幕功能序列。

以下是硬體 / 軟體螢光幕功能表的說明，內包含有功能編號，

16 進位位址及每一個功能的描述。此兩表 ( 各 11 拜 ) 的內容格式完全相同。

功能編號	軟體表	硬體表	功能描述
	OF396H	OF3A1H	游標位址坐標偏移量，範圍從 0 — 127。若最高比次 ( bit 為 0，則先送 Y 座標再送 X 座標，若最高比次為 1 則先 X 座標後 Y 座標。
	OF397H	OF3A2H	領入字元。如果沒有領入字元此

拜為 0。

註：以下表中若項目內容為 0 則表示不具備此功能。若最高比次為 1，表示需要有一引進字元，若最高比次為 0 則不需要。

- |   |        |        |                      |
|---|--------|--------|----------------------|
| 1 | OF398H | OF3A3H | 清除螢光幕                |
| 2 | OF399H | OF3A4H | 清除螢光幕上所標定位址之後所有字元。   |
| 3 | OF39AH | OF3A5H | 清除螢光幕現所標定位址以後本行所有字元。 |
| 4 | OF39BH | OF3A6H | 設定主文正規顯示模式。          |
| 5 | OF39CH | OF3A7H | 設定主文反相顯示模式。          |
| 6 | OF39DH | OF3A8H | 將游標移至最左上位置。          |
| 7 | OF39EH | OF3A9H | 游標定址。                |
| 8 | OF39FH | OF3AAH | 游標上移一列。              |
| 9 | OF39FH | OF3AAH | 非破壞性游標前移。            |

標準的 24×40 APPLE 螢光幕均提供這 9 種功能。不過如果軟體螢光幕功能表內項目被設為 0，則此功能便不具備。

用 CONFIGIO 程式可以檢查及修改軟體 / 硬體螢光幕功能表，請參見“APPLE CP/M 軟體公用程式手冊”。

### 與終端機無關之螢光幕功能 / 游標定址

因為硬體及軟體螢光幕功能表的通用特性，使我們在撰寫程式時可以利用到表中螢光幕功能而不必考慮到終端機種類的不同，只要對所使用特殊終端機建立好硬體螢光幕功能表即可達成此目的。以下是

一以 8080 組合語言所寫成的小程式，它可顯示如何使用螢光幕功能而不必去管終端機的種類：

```

:
: Terminal Independent Screen I/O
:

```

```

: This routine will execute the screen function
: specified by E, where E contains the screen function
: number from one to nine. If the function is not implemented,
: the subroutine simply returns. All registers are destroyed.
: (NK 5/80)

```

```

: Equates:

```

```

BDOS EQU 0005H ;CP/M function call address
SXYOFF EQU 0F396H ;Software cursor address XY coord.
; offset
SFLDIN EQU 0F397H ;Software function lead-in character
SSFTAB EQU 0F398H ;Software screen functions
;
SCRFUN: MVI D,0 ;Prepare for index
LXI H,SSFTAB-1 ;Point to Software Screen
; Function table minus one
DAD D ;Index to desired function char.
MOV A,M ;Get the char.
ORA A ;See if a Lead-in is required
RZ ;If the function isn't there, quit
JP CONOUA ;If pos., rio
PUSH PSW ;Save char.
LDA SFLDIN ;Get software lead-in char.
CALL CONOUA ;Output char. in A
POP PSW ;Re-get char.
CONOUA: MOV E,A ;Put char. in its place
CONOUE: MVI C,2 ;Console output function
JMP BDOS ;Call CP/M BDOS at 0005H

```

```

: This routine will position the cursor at the X,Y coords
: in [HL].
:

```

```

GOTOXY: PUSH H ;Save coords while we do seq.
MVI E,7 ;Do a Cursor Address function
CALL SCRFUN
POP H ;Get coordinates back
LDA SXYOFF ;Get software XY coordinate offset
ORA A ;Set CC's on [A]
JP NORVS ;Reverse coords if neg.
MOV E,L ;Reverse H&L
MOV L,H
MOV H,E

```

```

NORVS:  MOV E,A      ;Save offset
        ADD H        ;Add offset
        MOV H,A      ;Save for later
        MOV A,E      ;Re-get offset
        ADD L
        PUSH H       ;Save all this
        CALL CONOUA  ;Output first coord
        POP H        ;Restore coords.
        MOV E,H      ;Output second coordinate
        JMP CONOUE   ;And return.

```

注意；螢光幕功能字元序列是由上面副常式之軟體螢光幕功能表所定義。要使 Apple 螢光幕與這些副常式配合這項工作是必須的。又這兩種表內之項目中若有為 NUL ( 零 ) 的則將使得 Apple 24 × 40 螢光幕喪失此項目所代表之功能。

## 鍵盤字元的重新定義

有些 CP/M 軟體須用到某些特殊鍵，但這些特殊鍵在終端機鍵盤上卻找不到，Apple 之鍵盤就是最好的例子。一些最常用的字元像左括號 ( [ ) 及 RUBOUT 都不能打入，這時就要使用 I/O 架構框塊內的鍵盤字元重新定義表 ( Keyboard Character Redefinition Table ) 了。

此表的功能很簡單：它可重新定義任何鍵盤上敲入的字元為任何所需要的 ASCII 字元。例如，Ctrl-K 可被重新定義為代表左括號 ( [ )，那麼當敲入 Ctrl-K 時，就會產生 ( [ ) 字元。

此表另外一種技巧性使用法如下：譬如要終止某一 BASIC 程式

的執行必須按下Ctrl-C,但是若把Ctrl-C字元重新定義為另一字元,譬如NUL字元,那麼要終止此一BASIC程式將變得不可能。(也因為這樣,把此表弄亂將會帶來很多令人困擾的問題)。

鍵盤的重新定義只有在TTY:及CRT:輸入時才可以做。

### 鍵盤字元重新定義表

鍵盤字元重新定義表可以提供至多6個字元的重新定義,此表位於OF3ACH之位址(Z-80來看的話)。此表上的每一項目有2個拜:第一個拜是要重新定義的鍵盤字元ASCII值,第2個拜是想轉換成的ASCII值。兩個拜之最高階比次必須是零。

若表內項目少於6項,那麼表最後以附加一個拜結束之,此拜之比次為1。

要修改此表可以使用CONFIGIO程式,詳見“軟體公用程式手冊”。

### 非標準週邊裝置及I/O軟體支援

I/O 架構框塊內也提供了非標準週邊裝置及I/O的軟體使用可能性。此乃由改變框塊內的I/O向量表(Vector Table),把本指向CP/M BIOS內副常式的向量值變成指向使用者自己推動程式的值。在I/O 架構框塊內有3個128拜的記憶空間方塊供推動程式使用,每個方塊都有指定的裝置及溝槽,這樣才能避免記憶體的混亂。

位址	指定溝槽	指定的邏輯裝置
OF200H~OF27FH	第1槽	LST：列印機
OF280H~OF300H	第2槽	PUN：與RDR：一般的I/O
OF300H~OF37FH	第3槽	TTY：控制台裝置。

大多數 Apple I/O 面板上都有 6502 ROM 的推動程式，因此這些板和 CP/M 系統之溝通最簡單的方法就是利用 Z-80 程式內指令呼叫在 ROM 上之 6502 副常式。大多數通用的 I/O 要與 CP/M 溝通用此法就已足夠（見下節 6502 副常式之呼叫）。

若在指定的溝槽內沒有插入板子，則配置給它的 128 拜可以被移作和指定的邏輯裝置有關的用途，這些用途包括 Apple 鍵盤的小寫輸入推動器（driver）與卡帶界面等。

I/O 推動器副常式可藉著修改 I/O 向量來適應 CP/M 系統，向量位置及其功能如下表所示：

向量號碼	位址	向量名稱	功能描述
1	OF380H	控制台狀態	若有字元讀入則回應 OFFH 到 A 暫存器，若沒有則回應 00H
2	OF382H	控制台輸入 向量 # 1	從控制台讀入一字元，放入 A 暫存器，並將最高比次清為零
3	OF384H	控制台輸入 向量 # 2	
4	OF386H	控制台輸出 向量 # 1	傳送在暫存器 C 內的 ASCII 字元到控制台去。
5	OF388H	控制台輸出	



## 74 Apple II 軟體卡

		向量 # 2	
6	OF38AH	紙帶機輸入	從紙帶機讀入一字元存放在 A 向量 # 1 暫存器。
7	OF38CH	紙帶機輸入	向量 # 2
8	OF38EH	打孔機輸出	把在 C 暫存器內的字元送到打孔機上。
9	OF390H	打孔機輸出	向量 # 2
10	OF392H	列印輸出	把在 C 暫存器內的字元送到列印機上去。
11	OF394H	列印輸出	向量 # 2

註：在控制台輸出時，B 暫存器內包含了一個數目，此數目對應於 9 種提供的螢光幕功能之一。在正常字元輸出時 B 暫存器內為 0，而執行螢光幕功能 # 7 後在輸出游標位址 X Y 座標時 B 暫存器之資料不為 0。

### 把邏輯名字指定給實體 I/O 裝置：IOBYTE

正如在 CP/M 參考文件內所解釋的，IOBYTE 能用來把邏輯 I/O 裝置對應給實體裝置。而 STAT 程式則可以改變 IOBYTE 之內容，詳見“CP/M 參考手冊”。

IOBYTE 的功能就是產生邏輯裝置與實體裝置之間的對應，而可用 CP/M 程式或 STAT 公用程式來修改，此對應就是把 IOBYTE

分成 4 個欄，如下所示：

IOBYTE 位於 0003H: 

LIST	PUNCH	READER	CONSOLE
------	-------	--------	---------

比次：                    7   6   5   4   3                    2   1                    0

每一個欄內的值可以從 0 ~ 3，指定給每一欄之值的意義如下：

控制台欄 (第 0, 1 比次)

- 0 - 控制台是 TTY: 裝置
- 1 - 控制台是 CRT: 裝置
- 2 - 整批作業模式，以 RDR: ( READER ) 作為控制台輸入，  
LST: 做為控制台的輸出。
- 3 - 使用者定義控制台裝置 ( UC1: )。

READER 欄 (第 2, 3 比次)

- 0 - READER 是 TTY: 裝置
- 1 - READER 是 CRT: 裝置
- 2 - READER 是讀紙帶機 ( PTR: )
- 3 - 使用者定義之 READER 裝置 # 2 ( UR2: )

PUNCH 欄 (第 4, 5 比次)

- 0 - PUNCH 是 TTY 裝置
- 1 - PUNCH 是紙帶打孔機 ( PTP: )
- 2 - 使用者定義之 PUNCH # 1 ( UP1: )
- 3 - 使用者定義之 PUNCH # 2 ( UP2: )

LIST 欄 (第 6, 7 比次)

- 0 - LIST 為 TTY: 裝置

- 1 - LIST 為 CRT: 裝置
- 2 - LIST 為列印機 ( LPT: )
- 3 - 使用者定義之 LIST 裝置 ( UL: 1 )

以下是 Apple CP/M 系統實體裝置的描述：

TTY: 所指的是標準的 Apple 螢光幕及鍵盤，或者是接於第 3 槽界面板之外部終端機，其常式向量為控制台輸入向量 # 1 及控制台輸出向量 # 1，控制台狀態則是經由控制台狀態向量指出。

CRT: 和 TTY 相同。

UC1: 使用者定義之控制台裝置，常式向量為控制台輸入向量 # 2 及控制台輸出向量 # 2。

PTR: 標準的 APPLE 輸入界面，插於第 2 槽內。如果第 2 槽內沒有板子，PTR: 裝置永遠會送回一 end-of-file 字元 1 AH，其向量是 READER 輸入向量 # 1，所送回的字元放在 A 暫存器內。

UR1: 使用者定義之 READER # 1。從此裝置讀入的字元是放入 A 暫存器，其向量是 READER 輸入向量 # 2。

UR2: 使用者定義之 READER # 2，實體上和 UR1: 相同。

PTP: 任何標準的 Apple 輸出裝置，插於第 2 槽，如果第 2 槽內沒有板子，PTP: 裝置不做任何事。輸出到 PTP: 裝置是經由 PUNCH 輸出向量 # 1。

UP1: 使用者定義之 PUNCH # 1，在 C 暫存器內的字元經由 PUNCH 輸出向量 # 2 輸出。

UP2: 使用者定義之 PUNCH # 2，和 UP1 裝置實體相同。

LPT: 任何的標準Apple界面板，插於第1槽內用作輸出。  
在C暫存器內的字元經由List輸出向量#1輸出。

UL1: 使用者定義之List裝置，在C暫存器內的字元經由List輸出向量#2輸出。

IOBYTE 可以用STAT程式來修改，或者也可以用組合語言程式內使用CP/M的GET IOBYTE及SET IOBYTE( #7, #8 ) 功能。詳見“CP/M之特性及設施”(不在本書之中)及“CP/M參考手冊”內的CP/M界面指引那一章。

### 經由I/O向量表增補使用者的軟體

使用者副常式能藉著CONFIGIO程式被補入I/O架構框塊，也可藉著CONFIGIO把此修補的地方永久的儲存在CP/M磁碟之內。要產生碼檔( code file )，先使用ASM來寫出推動器程式，然後LOAD由ASM產生的HEX檔來產生COM檔。

由CONFIGIO所載入的碼檔必具有某特定內部格式，每個碼檔在I/O架構框塊中只容許安放一個碼段( code segment )，但是I/O向量表上的任意向量指標都可以加以修補。

以下是用CONFIGIO把一磁碟碼檔放入I/O架構框塊內在格式：

第一個拜：要修補I/O向量表向量的數目。

下兩個拜：程式碼的目的位址。

下兩個拜：程式碼的長度。

對每一要做的I/O向量修補可重覆以下過程：

下一個拜依補入向量類別——1 或 2 而定。

若補入向量類別為 1

下一個拜：要修補向量的號碼，從 1 ~ 11（見以上向量位置定義）

下兩個拜：前一拜指示之向量修改後的位址（指向使用者的程式）。

若補入向量類別為 2

下一個拜：要修補向量的號碼，從 0 ~ 11（見以上向量位置定義）。

下兩個拜：指定向量的現今內容所要放入的位址（可以是 JMP 的位址欄等）。（藉此位址保存現今內容）。

下兩個拜：要放入指定向量的新內容（位址）。

再接下去：實際的程式碼接著放入。每一與溝槽相關的框塊只能存程式碼 128 拜。依用途適當地使用對應溝槽的框塊。

以下是利用 CONFIGIO 程式補入 I/O 架構框塊的程式例。它放在這裏的目的是供你寫程式時的參考。它在 24×80 螢光幕界面板或標準的 Apple 螢光幕及鍵盤的應用相當方便，所以你可將它輸入試著使用看看。

注意如何使用 OFFSET 調整程式內參考位址。

要在 I/O 架構框塊內將此程式補入，必須：

1. 使用 DDT "S" 命令把程式放到記憶體 0100H 之位址。
2. 使用 CP/M SAVE 命令把它保存到磁碟上去。
3. 使用 CONFIGIO # 3 選擇 ( option ) 將小寫字母推動程式放

入 I/O 架構框塊內。

4. 使用 CONFIGIO #4 選擇 ( option ) 將整個修補後 I/O 架構框塊存入磁碟中。

若你要把此小寫程式補入 I/O 架構框塊時，請注意以下事項：

此推動程式之自然值 ( default ) 為大寫字母 shift lock，前指箭頭 ( → ) 被當作 shift 鍵 ( 註：打入此鍵一次，即可從大寫變小寫或小寫變大寫 )，當你在大寫字母狀態，要想寫小寫字母，只要按一下 shift 鍵 ( 但不必一直按著 )，接著打入字母即是小寫，當要恢復原來模式時，只要再按一次即可。

內在格式設定

```

; APPLE CP/M LOWER CASE INPUT ROUTINE
;
; This routine can be assembled using ASM and
; LOAD to produce a file that can be loaded and
; patched into the I/O Configuration Block with
; CONFIGIO. It is also intended to be used as
; a model for your own programs.
;

0015 = SHFCHR EQU 21 ;Shift key is the forward-arrow
F389 = SLTTYF EQU 0F389H ;Slot types table
E000 = KEYBD EQU 0E000H ;Address of Apple Keyboard
;
0100 ORC 0100H ;This is so LOAD will load at 100h
F300 = ORIGIN EQU 0F300H ;Real origin of Program
OFFSET SET ORIGIN-LWRCASE ;Must be added to 16-bit addresses
;
0100 01 DB 1 ;Make one patch
0101 00F3 DW ORIGIN ;Destination address of Program
0103 3E00 DW PRGENU-LWRCASE ;Length of Program
;
0105 02 DB 2 ;Patch type 2
;
0106 02 DB 2 ;Patch Console Input vector #1
0107 06F3 DW OLDINP+OFFSET ;Place to put current contents of vector
0109 00F3 DW LWRCASE+OFFSET ;New contents of vector
;
; Check to make sure he isn't using an external terminal:
;
0109 3A0BF3 LWRCASE:LDA SLTTYF+2 ;Is there a card in slot 3?
010E FE03 CFI 3 ;Is he using a Com Caro as a terminal?
0110 CA0000 JZ 0000 ;Dummy address
0111 = OLDINP EQU 9-2 ;Place to put normal input routine addr
;
; Get a character from The Apple Keyboard:
;
0113 3A00E0 NLOOP: LDA KEYBD ;See if char available at keyboard
0116 B7 ORA A ;Set condition codes on keybd ioc
0117 F208F3 JP NLOOP+OFFSET ;Loop if char not available
011A 3210E0 STA KEYBD+10H ;Clear keyboard strobe
011B E67F ANI 7FH ;Mask high bit of char
011F 4F MOV C,A ;Save character in [C]

```

# 80 Apple II 軟體卡

```

0120 0615      MVI     B,S#FCHK      ;Shift character into [B]
0122 213DF3    LXI     H,STATE+OFFSET ;Point to shift state
0125 7E        MOV     A,H          ;Get state.
0126 FE01      CPI     1            ;Determine state
0128 79        MOV     A,C          ;Get typed character into [A]
0129 DA3AF3    JC      STATE0+OFFSET ;Carry set - state 0
012C CA2EF3    JZ      STATE1+OFFSET ;State 1
;
; Here if in lower case input mode.
; All alphabetic characters are converted
; to lower case, unless the shift character is
; typed, which enters 'shift next character' mode
;
012F 88        STATE2: CNP     B            ;for shift char.
0130 CA32F3    JZ      SETONE+OFFSET ;It was, set state = 1
0133 FE40      CPI     34         ;It wasn't, so convert all
0135 08        RC              ;Alphabetic chars to lower case.
0136 EE20      XRI     00100000B ;This does the conversion
0138 C9        RET              ;All done
;
; Here if in 'shift next character' mode; entered
; by typing the shift char once in lower case
; input mode. If shift character is typed again,
; upper case shift lock mode will be entered.
;
0139 34        STATE1: INR     H            ;Reset state = 2 = lower case mode
013A 88        CNP     B            ;Hit shift character?
013B C0        RNZ          ;Let upper case character go.
013C 35        BCR     H            ;set state to zero; upper shift lock
013D 35        SETONE: BCR     H
013E C300F3    JNP     LWRCASE+OFFSET ;Get another character
;
; Here if in upper case shift lock mode.
; Shift character must be typed once to enter lower
; case input mode.
;
0141 88        STATE0: CNP     B            ;Did he type shift char?
0142 C0        RNZ          ;Not shift, return upper case char.
0143 3602      MVI     H,2          ;Set state = 2 = lower case input mode
0145 C300F3    JNP     LWRCASE+OFFSET ;and set another character
;
0148 06        STATE: DB      0           ;Shift state. Default = upper lock
;
PRGEND:
0149          END
0100 01 00 F3 3E 00 02 02 06 F3 00 F3 3A 8B F3 FE 03
0110 CA 06 00 3A 00 E0 97 F2 08 F3 32 10 E0 E6 7F 4F
0120 06 15 21 3D F3 7E FE 01 79 DA 36 F3 CA 2E F3 8B
0130 CA 32 F3 FE 40 D8 EE 20 C9 34 8B C0 35 35 C3 00
0140 F3 8B C0 36 02 C3 00 F3 00
-
-
-

```

## 6502 副常式之呼叫

正如本手冊“硬體詳述”那一章所述，Z-80 模式要呼叫 6502 處理機出來工作只要執行一個寫入 \$0EN00H 的動作即可，其中 N 是 Soft Card 溝槽位置編號。相同的，在 6502 模式要呼叫 Z-80 處理機工作也只要執行一個寫入 \$CN00H 的動作即可（參考第 2-4 頁 Z-80，6502 位址轉換表），由於 Soft Card 可以插入的溝槽並 unlimited，所以位址也因所插入之溝槽位置而異。

但是，當系統被啓始（BOOT）時，CP/M 會確定 SoftCard 所在位置並將其存入 I/O 架構框塊內，此後 CP/M 利用它呼叫 6502 的副常式，詳見本手冊“硬體詳述”部份。

呼叫 6502 副常式很簡單，程式員只要設定好 6502 副常式之位址，然後執行上述的寫入 SoftCard 位址動作即可。也可能經由 6502 的 A，X，Y 及 P（狀態）暫存器傳送參數給 6502 副常式或從 6502 副常式內傳出參數。在 6502 副常式呼叫後 6502 堆疊指標（stack pointer）亦可以使用。注意 6502 和 Z-80 的位址並不相同——詳見第 2-4 頁 6502/Z-80 位址轉換表。

Z-80 位址	6502 位址	用 途
0F045H	\$ 45	6502 A 暫存器之傳送區。
0F046H	\$ 46	6502 Y 暫存器之傳送區。
0F047H	\$ 47	6502 X 暫存器之傳送區。
0F048H	\$ 48	6502 P（狀態）暫存器之傳送區。
0F049H	\$ 49	在從副常式內出來時所含的 6502 堆疊指標。



## 82 Apple II 軟體卡

0F3DEH	SoftCard的位址保存在此。低階拜 = 0，高階拜為 0ENH，N 是 SoftCard 所佔的溝槽編號。
0F3D0H	要呼叫的副常式位址以先低階拜後高階拜的格式保存於此。
\$ 3C0	儲存執行 6502 轉換到 Z-80 模式程式的開始位址。6502 的 RESET, NMI, 以及 BREAK 都指向此位址。JMP 到此位址會使 6502 處於 HOLD 狀態並回到 Z-80 模式。

註：6502 副常式不能使用 \$800~\$FFF，因為 Apple 磁碟推動程式軟體及磁碟緩衝區存放於此。

語言板使用者特別注意：

當在 Z-80 模式時，語言板上的 RAM 是既可讀出又可寫入的。當呼叫 6502 副常式時，Apple 主機內的 ROM 就自動的能夠使用，使得 6502 可以使用 ROM 上的監督程式 (MONITOR)，而此時語言板上的 RAM 仍可被寫入，也就是說，若寫入的位址超過 6502 \$D000 時資料將會寫到語言板的 RAM 上。

當副常式使用 APPLE 主機內的 ROM 時，有一副作用會產生：亦即 6502 將無法讀出 Z-80 記憶空間 0C00H~0EFFFH (此即 6502 \$D000 ~\$FFFF)。

APPLE CP/M 並未使用 6502 \$D000 ~DFFF 之間 4K 的記憶空間。

以下是以 8080 組合語言所寫的一個小程式，它可說明如何使用以上的位址來呼叫 6502 副常式：

```

; Subroutine to read the value of
; Paddle zero into register A.
; Demonstrates 6502 subroutine
; calling conventions and parameter
; passing. (NK 5/80)
;
;
; Equates
Z$CPU EQU 0F3DEH ;Location of SoftCard stored here
A$VEC EQU 0F3D0H ;Addr of 6502 sub. to call goes here
A$ACC EQU 0F045H ;6502 A register goes here
A$XREG EQU 0F046H ;6502 Y register pass area
PREAD EQU 0FB1EH ;Apple Monitor paddle read routine
;
PDL: XRA A ;Clear A register
STA A$XREG ;Read paddle zero
LXI H,PREAD ;Get addr of subroutine
SHLD A$VEC ;And store it for 6502 caller
LHLD Z$CPU ;Get SoftCard addr...
MOV M,A ;Go do it! (Must be a write)

Execution resumes here after 6502 does a RTS
LDA A$ACC ;A = paddle value.
RET ;All done - return

```

## 週邊板存在與位置之指示：

### 板型態表 ( Card Type Table )

當 Apple CP/M 啟動 ( BOOT ) 時，它將會去檢查每一溝槽看看有沒有 Apple 的標準界面板在裏面。這是藉著檢查配置給週邊板推動器 ROMs 用的和溝槽有關的記憶空間，看看記憶空間內是否有 ROM 存在，然後拿其中兩個拜與標準 APPLE I/O 週邊板的兩個拜相比較就可確定。

然後將此訊息存入板型態表內，此資料存在板形態表內，此表是 I/O 架構框塊之一部份，它共有 7 個拜，每個拜對應於一個溝槽。

表內項目的值可以從 0~5，每個值的意義如下：

值	說明
0	沒有偵查到有週邊板的 ROM (通常意即沒有板子放在槽內)。
1	偵測到有週邊板的 ROM，但不知其屬於何種形態。
2	APPLE II 磁碟機控制板在溝槽內。
3	APPLE 通信界面板或 CCS 7710A 串式傳輸界面板在溝槽內。
4	Apple 高速串式傳輸界面、Videx Videoterm, M & R Sup-R-Term 或 Apple Silentye 列印機界面板在溝槽內。
5	APPLE 平行列印機界面板在溝槽內。

這些資料對程式員很有用，例如，如果表內第 3 項 (代表溝槽 3——控制台裝置槽) 的板型態的值是 3 或 4；那麼就可假定使用者正使用某種形式的 80 行外部終端機，此時系統就會自動寫入適合 40 行或 80 行終端機的軟體來對應此種架構。

板型態表位於 0F3B9H 位址，每一溝槽所對應於表內項目之位址為 3B8H+S，其中 S 代表溝槽編號。(S 值由 1 到 7)

### 磁碟機數目拜 (Disk Count Byte)

磁碟機數目拜的數目等於現有磁碟機控制板數目的乘 2 (註：每一磁碟機控制板可接兩架磁碟機)，但它不能指示出是否是奇數架磁碟機 (亦即有時是一片控制板只接一架磁碟機的情況)。

磁碟機數目拜位於 0F3B8H。

**不必把電源關掉就可BOOT磁碟的方法**

以下的程式使你不必關掉APPLE主機電源就可從CP/M BOOT磁碟，此程式並非必要，它只是想省掉關掉電源過程而已。

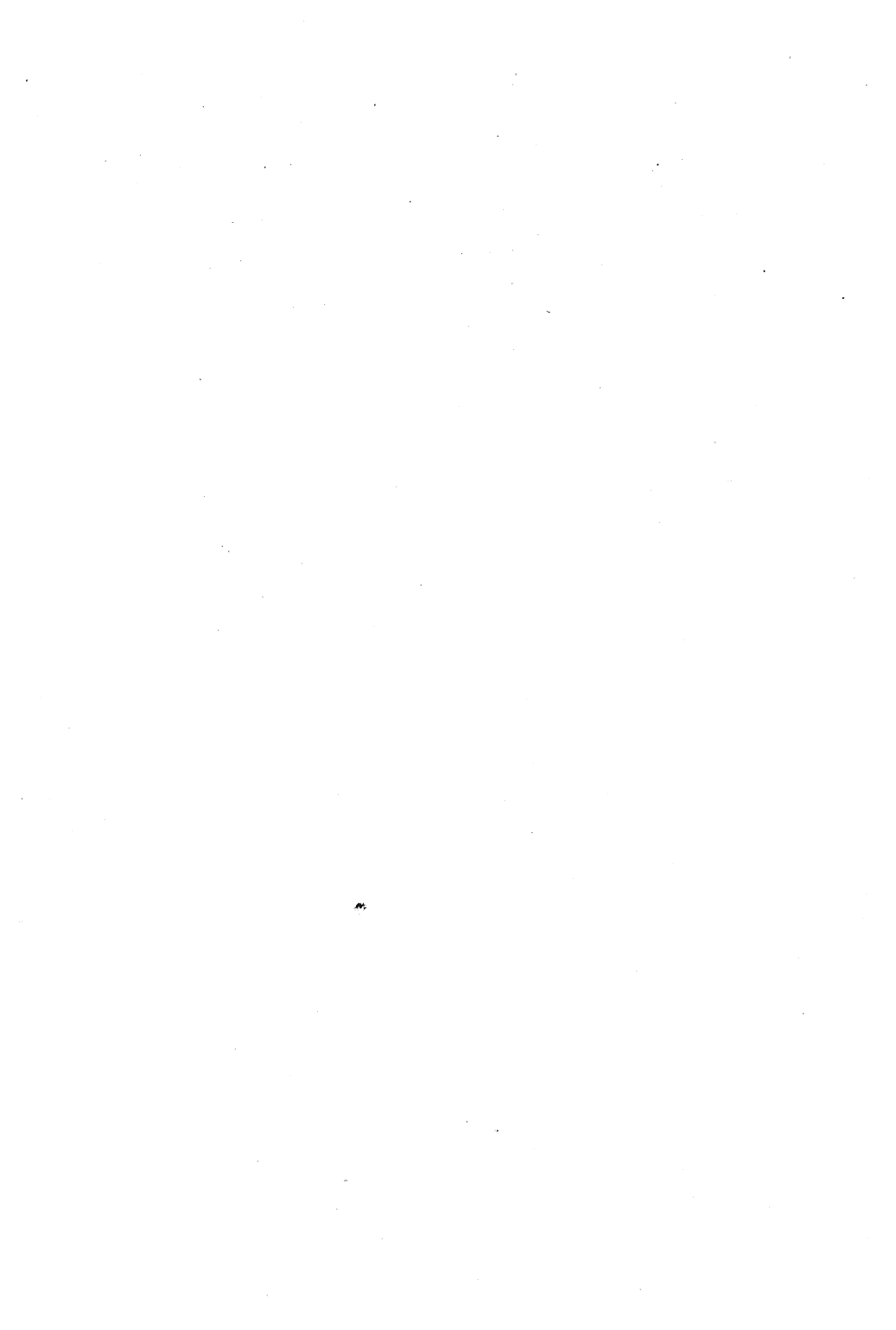
1 使用DDT“S”命令從100H位址輸入以下的資料：

```
0100 0E 01 CD 05 00 21 77 C7 22 00 30 21 00 C6 22 D0
0110 F3 2A DE F3 C3 00 36
```

2 打入Ctrl-C，跳出DDT。

3 打入SAVE 1 BOOT. COM

此時，此程式已存在磁碟裏，要用時，只要打入BOOT後RETURN。等幾秒後再插入你所想啓始（BOOT）的磁碟，再打入任何一個鍵，你的系統就會重新啓動，正如同在APPLESOFT或整數型BASIC內打入PR#6一般。



# 第 三 章

## 硬體的描述

- 簡介
- 時序架構
- SoftCard 的控制
- 位址匯流排界面
- 資料匯流排界面
- 6502 之“更新”(refresh)
- DMA Daisy 鏈
- 中斷
- SoftCard零件表
- SoftCard 電路圖

本章描述 SoftCard 的實體電路及其操作。SoftCard 的使用並不必要用這些資料，這些資料只是用來滿足你的好奇心並在做特別的應用時提供你一些有用的訊息。

## 簡介

SoftCard 為 Apple 計算機族中的一個週邊裝置板，其上含有足夠的硬體溝通 Z-80 微處理機 (SoftCard 板上的) 與 Apple 匯流排，使你能執行 8080 及 Z-80 程式，包括 CP/M 作業系統及所有在 CP/M 環境下所發展出來的程式。

SoftCard 插入 Apple 主板之第 0 槽以外的任一槽中，可用在 APPLE II，APPLE II PLUS，或含語言系統的這兩種機器。若用於含語言系統的機器上，語言板上的記憶體可供 CP/M 或在 CP/M 執行的程式使用。

## 時序架構

SoftCard 上 Z-80 處理機的時序與 Apple 主機的同步且相鎖 (phase locked) 於主機時序上。此由 Apple 主機造出一簡略的時序信號給予 Z-80 而達成的。

在每一個視訊更新 (Video refresh) 週期 (01) 中，~~7 MHz~~ 7 MHz Apple 時鐘分解並造出三個長 135 nsec 的半週時序，第一半週為“高”電位，第二半週為“低”電位，第三半週為“高”電位。第三個半週結束後電位拉低直到下一週 01 出現再重覆之。也就是說 Z-80 時序在整週 02 及小部分 01 時間為“低”電位。此第四半週長約 563 nsec (此時間在每一視訊線末端皆被拉長 69 nsec)。故 Z-

80 有效時序速率為 2.041MHz。

每種“機器週”(machine cycle)中皆具一“記憶存取週”(memory access cycle)( $\beta 2$ )。讀/寫線的“寫入”時序前沿(leading edge)與 SoftCard 時序同步以保證“寫入”線只在 SoftCard 時序為“高”電位時才變為“低”電位。

因為 Z-80 的所有位址在時序為“高”電位時才做位址變遷，故其發生在“ $\beta 1$ ”週(視訊更新作用正執行時)。因此在每“ $\beta 2$ ”週，整週中都具穩定的位址資料。

SoftCard 時序由 U4 及部分的 U1 和 U9 造出，此電路在 7 MHz 時鐘與  $\beta 1$  起始點稍有偏差時仍能工作使其同步。其中 Q1 及附屬零件構成類比緩衝器(buffer)提供在電源電壓零點幾伏內的高速切換特性。

## Soft Card 的控制

藉著寫入命令至適當記憶體而控制 SoftCard，此記憶體原來為 Apple 週邊僅讀記憶體位置。藉寫入指令以確定 6502 未連續做兩次存取動作(避免再切換控制回 6502 CPU)為極重要的控制方式。

APPLE 開機時，重置信號令 SoftCard 成為 OFF 狀態。此重置信號同步於 Apple 時序使寫入動作不被中斷掉。之後，Z-80 CPU 立即進入等待(wait)狀態，直到 SoftCard 開始作用為止。

當一寫入指令寫入到適當記憶區時，SoftCard 動作並亮起板上的紅色 LED。在含有 SoftCard 位址資料的一記憶體週期出現之前，Z-80 CPU 仍維持在等待(wait)模式。記憶體週期出現後，Z-80 CPU 脫離等待(wait)模式，不再在等待週(wait cycle)而開始執程式。

當 SoftCard 執行另一寫入指令，寫入到相同記憶區時，



## 90 Apple II 軟體卡

SoftCard便停止動作。

下表為依各溝槽而定用來控制 SoftCard 的記憶體位址：

SLOT	CONTROL ADDRESSES
1	\$C100-\$C1FF
2	\$C200-\$C2FF
3	\$C300-\$C3FF
4	\$C400-\$C4FF
5	\$C500-\$C5FF
6	\$C600-\$C6FF
7	\$C700-\$C7FF

## 位址匯流排界面

SoftCard 位址匯流排經由一組 ( BANK ) 轉換電路與 Apple I/O 匯流排溝通。此電路由 U7, U8, U11 及半個 U12 構成, 以解決存在於 6502 結構與 CP/M 及 Z-80 微處理機兩者用法之間的位址衝突問題。S1 - 1 開路使此電路作用, 將所有位址加上了 \$1000 的偏移量。結果把 Z-80 中斷位址及 CP/M 起始位址移出 6502 記憶體第零頁, 亦移動了 \$C000 - \$EFFF 的位址使其成為表面上連續的記憶體供 CP/M 使用。下列為實際的位址轉換表：

Z-80 ADDRESS	APPLE ADDRESS
\$0000-\$0FFF	\$1000-\$1FFF
\$1000-\$1FFF	\$2000-\$2FFF
\$2000-\$2FFF	\$3000-\$3FFF
\$3000-\$3FFF	\$4000-\$4FFF
\$4000-\$4FFF	\$5000-\$5FFF
\$5000-\$5FFF	\$6000-\$6FFF
\$6000-\$6FFF	\$7000-\$7FFF
\$7000-\$7FFF	\$8000-\$8FFF
\$8000-\$8FFF	\$9000-\$9FFF
\$9000-\$9FFF	\$A000-\$AFFF
\$A000-\$AFFF	\$B000-\$BFFF
\$B000-\$BFFF	\$D000-\$DFFF
\$C000-\$CFFF	\$E000-\$EFFF
\$D000-\$DFFF	\$F000-\$FFFF
\$E000-\$EFFF	\$C000-\$CFFF
\$F000-\$FFFF	\$0000-\$0FFF

附有語言板時，Z-80 可具有 \$0000-\$DFFF 連續記憶體，而不會取用到 6502 的第零頁記憶體或 Apple 的週邊機位址區。

將 S1 -1 閉路使此電路停止作用，此時 SoftCard 位址匯流排不經轉換而與 Apple I/O 匯流排相接。

所有的位址匯流排皆經三態 (tri-state) 的緩衝器輸出，可供應或吸入 24mA 電流。SoftCard 放棄匯流排控制權後，這些緩衝器關閉掉。開關變動 (turn ON/OFF) 時的特別時序安排避免了 SoftCard 緩衝器在 APPLE 控制匯流排之下推動匯流排。

SoftCard 的時序令所有位址的變動發生在 APPLE 更新視訊顯示器 (及動態記憶體) 之時。位址線在每個“記憶體存取週”一開始就必須處於穩定狀態。Z-80 的“記憶體存取週”不加用等待狀態 (wait states)。

## 資料匯流排界面

由 SoftCard 至 APPLE 的資料線亦經由與位址線同等的高電流緩衝器輸出。此緩衝器在以下兩情況下作用：

1. SoftCard 控制匯流排時。
2. SoftCard 寫入記憶體時。

SoftCard 在讀取記憶體時所讀取的資料被 U15 所緩衝並門住。U15 為三態輸出，僅在 SoftCard 讀入資料時才作用。此門住器將未被 APPLE 門住的資料 (如鍵盤鍵入字元) 保存至 Z-80 取到為止。

因 SoftCard 的時序同步且“相鎖”於 APPLE 的時序，由 Z-80 產出的時序信號可以直接推動控制緩衝器及門住器。

Z-80 得知中斷發生後 (硬體或軟體中斷)，升位電阻令一個對 Z-80 任一中斷模式可預期反應的動作發生。在中斷程序中所讀入之

任一資料皆為 \$FF。

## 6502之更新 (refresh)

6502 為動態微處理機，須利用一定的時序週保持其內部暫存器的資料。APPLE DMA 電路藉關閉 6502 的時序來中斷其運算。欲使 6502 保有原內部資料隨時恢復原運算，時序週 (clock cycle) 應按時返回 6502 做更新 (refresh) 動作。

令 6502 進入非完備狀態 (non-ready state) (令 "RDY" 線為 "低" 電位)，並使其可控制一個記憶體摘取 (fetch) 動作以完成上述更新 (refresh)。6502 不用所摘取的資料。在 "更新" 記憶體週期之後，匯流排的控制權立即回歸 SoftCard。

上述控制功能由 SoftCard 上 Z-80 動態更新控制線 (dynamic refresh control line) 為之。因此 6502 的 "更新" 發生於每一操作碼摘取後，且不影响 SoftCard 及使用者之工作。Z-80 利用 6502 "更新" 的時間對操作碼做解碼的工作，故 Z-80 的機器週 (machine cycle) 不需含等待週 (wait cycle)。6502 取回匯流排控制權時，SoftCard 的位址及資料緩衝器之輸出轉為第三態高阻抗模式 (tristate mode)。

若有更高優先序的 DMA 裝置中斷 SoftCard 動作，6502 的 "更新" 功能便無法再作用。若在此 DMA 之後仍欲保有 6502 內部資料，DMA 週期長度必須限制在數微秒之內 (小於 5 微秒)。

在指令的正式混合之下 6502 的 "更新" 約每 4 - 5 微秒發生一次，比所規定的 40 微秒最大限值良好得多。最長的指令允許，兩次 "更新" 之間距 11.25 微秒。

## DMA Daisy鏈

APPLE具有完整的DMA Daisy 鏈，更高優先序DMA 裝置可令 SoftCard放棄匯流排控制權。S1 - 2 閉路 (ON)時，使DMA 裝置可對 SoftCard 中斷。在S1 - 2 閉路且DMA Daisy 鏈輸入 (27 腳)為低電位時，Z-80先結束當時的機器週，然後Soft-Card 將在I/O 匯流排第22 腳的DMA 控制線電壓升高以放棄對匯流排控制權。此時，另一裝置可能欲以拉低22 腳電壓取得控制權。控制權並不能立即轉移，因SoftCard 緩衝器仍推動着匯流排。

若S1 - 2 開路 (OFF)，在SoftCard 不作用時Daisy鏈仍保持着。當SoftCard 作用，Daisy 鏈的輸出 (24 腳)通知較低優先序的裝置DMA 正動作中。如此低優先序裝置就鎖住不做DMA 動作。同樣地，較高優先序裝置也鎖住不做DMA，因此SoftCard 不放掉匯流排控制權。

## 中斷

Apple 中含有中斷 SoftCard之 Z-80 及 6502 處理機的硬體設施。S1 - 4 閉路時 (ON)，Z-80 對發生於Apple 的中斷做反應。但其中斷處理程式並不中斷之服務，而將控制權交回 6502，由 6502 對中斷做實際處理工作。如此可使 6502 亦得知中斷發生，以便將其中斷狀態 (interrupt status) 清除掉。

不論 Z-80 選用何種中斷模式，在中斷程序中讀入的拜資料皆為 \$FF。此值可用來做向量指標指向存放中斷處理常式的特殊記憶位置

S1 - 3 為不可遮罩中斷 (non-maskable interrupt)，功

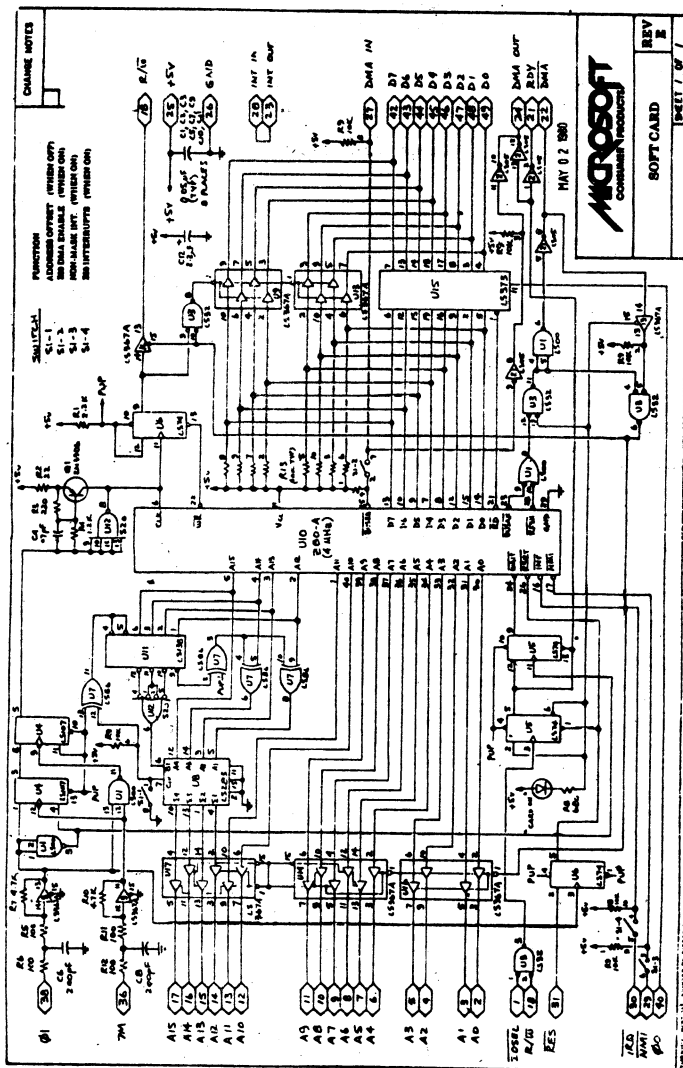
能與S1 - 4同。

### softcard 零件表

Component Identifier	Part No.	Description
U1	74LS00	Quad Nand
U2	74LS05	Hex Inverter
U3	74LS32	Quad Or
U4	74LS107	Dual Flip-Flop
U5	74LS74A	Dual Flip-Flop
U6	74LS74A	Dual Flip-Flop
U7	74LS86	Quad Ex-Or
U8	74LS283	4 Bit Adder
U9	74LS367A	Hex Buffer
U10	Z-80A	Z-80A (4 MHz)
U11	74LS138	Octal Decoder
U12	74S20 (must be "S" part)	Dual Nand
U13	74LS367A	Hex Buffer
U14	74LS367A	Hex Buffer
U15	74LS373	Octal Latch
U16	74LS367A	Hex Buffer
U17	74LS367A	Hex Buffer
Q1	2N3906	PNP Transistor
R1		2.2K $\Omega$ , 5%, 1/4 watt
R2		22 $\Omega$ , 5%, 1/4 watt
R3		220 $\Omega$ , 5%, 1/4 watt
R4		1.2K $\Omega$ , 5%, 1/4 watt
R5		100 $\Omega$ , 5%, 1/4 watt
R6		100 $\Omega$ , 5%, 1/4 watt
R7		4.7K $\Omega$ , 5%, 1/4 watt
R8		680 $\Omega$ , 5%, 1/4 watt
R9		Resistor-Pack, 10K $\Omega$ ,
R10		4.7K $\Omega$ , 5%, 1/4 watt
R11		100 $\Omega$ , 5%, 1/4 watt
R12		100 $\Omega$ , 5%, 1/4 watt
R13		Resistor Pack, 10K $\Omega$
C1		Capacitor, 0.05 $\mu$ F
C2		Capacitor, 0.05 $\mu$ F
C3		Capacitor, 0.05 $\mu$ F
C4		Capacitor, 47 $\rho$ F, 10%, 1000V
C5		Capacitor, 0.05 $\mu$ F

C6	Capacitor, 200 $\mu$ F, 10%, 1000V
C7	Capacitor, 0.05 $\mu$ F
C8	Capacitor, 200 $\mu$ F, 10%, 1000V
C9	Capacitor, 0.05 $\mu$ F
C10	Capacitor, 0.05 $\mu$ F
C11	Capacitor, 0.05 $\mu$ F
C12	Capacitor, Solid Tant., 2.2 $\mu$ F, 20%, 35V
"Card On"	Light Emitting Diode
S1	Dip Switch – Quad
	Printed Circuit Card

softcard 電路圖



**第三部份**

**CP/M**

**參考手冊**





# 第 一 章

## CP/M的特色與功能介紹

- 簡介
- CP/M 2.0功能概況
- CP/M 的功能描述
- 一般命令的結構
- 檔案參考
- 磁碟機的切換
- 內建命令的格式
  - ERAsE 命令
  - DIRectory 命令
  - REName 命令
  - SAVE 命令
  - TYPE 命令
  - USER 命令
- 行的編輯與輸出控制
- 暫態程式命令
  - STAT
  - ASM
  - LOAD
  - DDT

100 Apple II 軟體卡

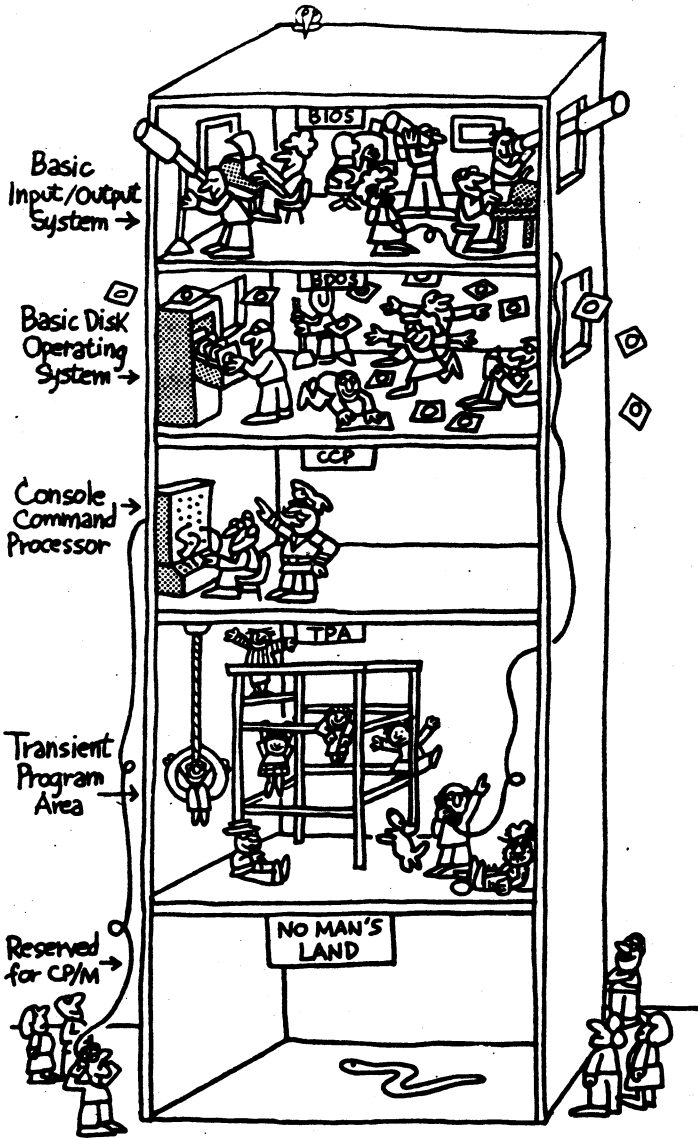
PIP

ED

SUBMIT

DUMP

- BDOS 的錯誤訊息



CP/M MEMORY USAGE

CP/M記憶體的配置

## 簡介

CP/M 是一套為微電腦系統而設計的監督控制程式 (Monitor Control Program)，它使用與 IBM 規格相容的軟性磁碟作為備存記憶體 (Backup Storage)。硬體使用 Intel 8080 微電腦作為系統的主要架構。CP/M 擁有建立程式、儲存程式及編輯程式的功能，也有組合語言和程式偵錯的功能。CP/M 有個重要的特色：就是在任何①以 Intel 8080 或 Zilog Z-80 為中央處理單元 (CPU) ②主記憶體在 16K 拜 (byte) 以上，③並具有 4 部以下與 IBM 相容的磁碟機的系統，CP/M 只要經過簡單的修改，就可在這些機器上運作。雖然標準的 Digital Research 版本只可在單密度的 Intel MDS800 磁碟上作業，但已有幾家硬體製造商證實已經為 CP/M 提供了他們自己的輸入輸出推動程式 (I/O Driver)。

CP/M 監督控制程式有一檔案管理系統，能提供快速的程式存取。這個檔案副系統，有一種具名檔案結構 (Named File Structure)，它能隨機配置檔案空間，並存取循序 (Sequential) 檔及隨機 (random) 檔。利用這種檔案系統，各種不同的程式，都可以用源始 (Source) 或可執行的機器碼形式貯存在磁碟中。

CP/M 同時擁有強而有力的文稿編輯程式 (Context Editor)，與 Intel 相容的組合編譯程式 (Assembler) 以及程式除錯副系統。可選用的軟體，則包括有與 Intel 相容的巨集指令組合編譯程式 (Macro Assembler)，符號除錯程式 (Symbolic debugger)，及各種高階語言。若再把 CP/M 的控制台命令處理程式 (Console Command Processor) 合併起來，所造成的功

與大型電腦應用工具相比，不相上下，甚有過之。

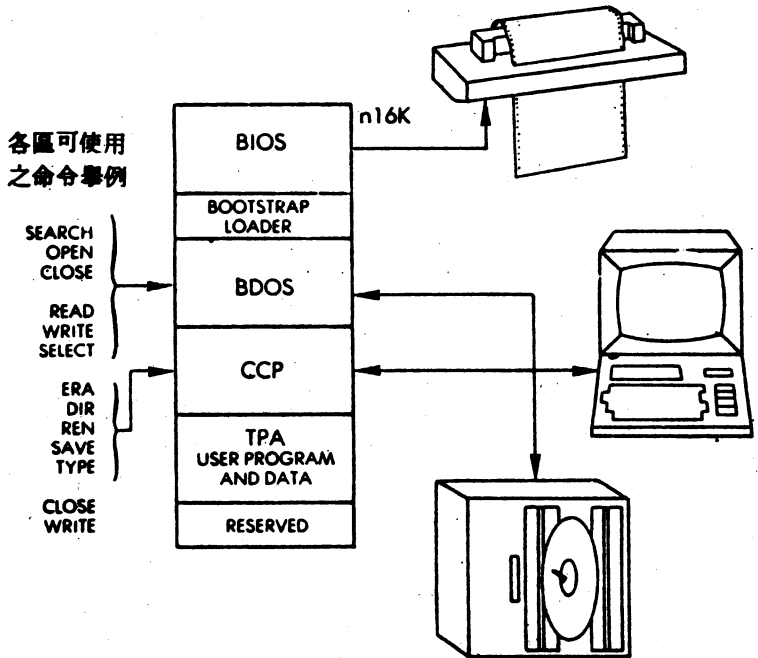
CP/M邏輯上可區分成幾個不同的部分：

BIOS 基本 I / O 系統 ( Basic I / O System )

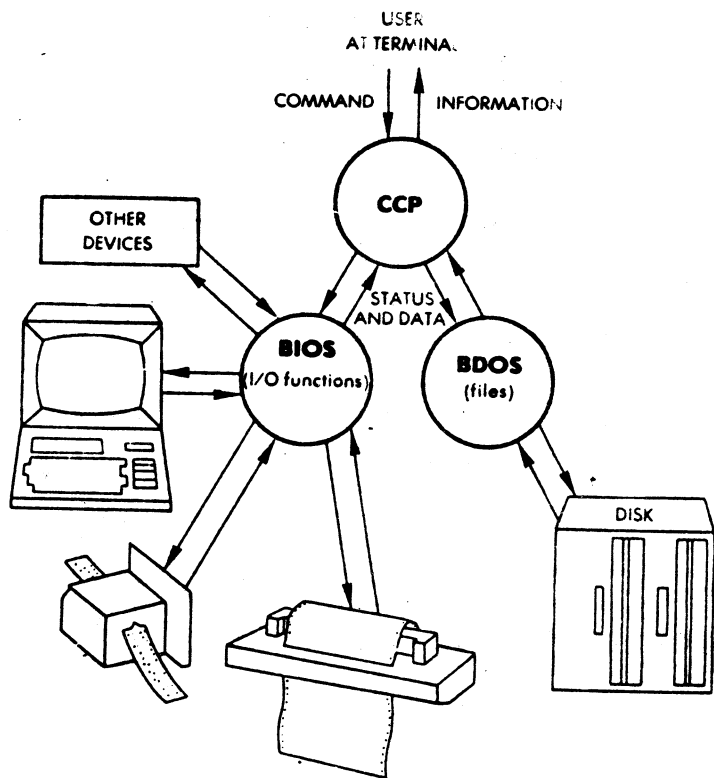
BDOS 基本磁碟作業系統 ( Basic Disk Operating System )

CCP 控制台命令處理程式 ( Console Command Processor )

TPA 暫態程式區 ( Transient Program Area )



CP/M記憶體空間配置



CP / M 控制流向

BIOS 提供存取磁碟機與溝通標準週邊設備所需要的基本動作，其中週邊設備可為：電傳打字機、螢光幕、紙帶機，及使用者 ( user ) 設定的週邊設備。使用者可依自己的特殊硬體設備，增補或刪除 BIOS 的內容。

BDOS 經由一部或多部含有獨立檔案索引的磁碟機，提供磁碟管理。BDOS 的磁碟配置方法提供了完全動態的檔案結構，使存取磁碟所需的讀寫頭移動次數減至最少。標準的 CP / M 系統，每個磁

碟可含有64個不同的檔案，只要不超過一個磁碟的貯存量，每個檔案可含有任何數目的資料錄數。BDOS 有下列幾個動作，每個動作都對應於BDOS 中的一個進入點(Entry Point)，使用者的程式可經由這些進入點，要求BDOS 作指定的動作。

SEARCH	找尋磁碟上特定的檔案
OPEN	開啓檔案
CLOSE	關閉檔案
RENAME	更改檔案名稱
READ	讀取檔案的資料錄
WRITE	把資料錄寫入磁碟的檔案
SELECT	選擇磁碟機

CCP 提供了控制台使用者與其他CP/M系統元件間的符號界面(Symbolic Interface)。CCP 由控制台讀取命令(Command)，並加以處理，這些命令包括列出檔案索引表(directory)、印出檔案內容、及指揮諸如組合編譯程式、編輯程式、偵錯程式等暫態程式的動作。標準的CCP命令將在其他的章節中再行討論。

CP/M最後的部份叫做暫態程式區(TPA)。TPA所佔有的程式，是依據CCP的命令，從磁碟載入(Load)而來的。例如在程式編輯時，TPA就被文稿編輯程式的機器碼與資料區所佔據。同樣地，在CP/M下所建立的程式，也可載入TPA中執行，檢查該程式是否有邏輯上的錯誤。

有件事值得一提，即任何或所有的CP/M的副系統，皆可被正



在執行的程式覆蓋 ( Overlay )。意即，一旦使用者程式載入 TPA, CCP, BDOS 及 BIOS 所佔的區域都可拿來作為程式資料區。只要 BIOS 部份不被蓋掉，使用者仍可由程式取得系統啟動 ( Bootstrap ) 的載入程式 ( Loader )。如此，在程式執行結束後，使用者只需跳至啟動系統的載入程式 ( bootstrap loader ) CP/M 完整的監督程式就會從磁碟中再度載入記憶體。

再提醒讀者，整個 CP/M 作業系統是由幾個不同的模組組成的，包括定義 CP/M 執行時硬體設備的 BIOS 部份在內。因此，只要更換使用者系統所用週邊設備的推動程式，標準的 CP/M 可適用於任何非標準的環境設備上。

## CP/M2.0的功能概況

CP/M 2.0 是個高效能的單控制台作業系統，此系統利用表格化 ( Table Driven ) 的技術，可以欄位架構來適應各種容量不同的磁碟。它去除了所有基本檔案限制，却又與先前發行的第一版本，維持著相當高的可容性。CP/M 2.0 之中，包括一到十六個邏輯磁碟機的欄位規格，每個磁碟機的容量可達到八百萬拜，檔案最大可達到整個磁碟機的最大容量，這容量在未來的版本裏將更可高達三千二百萬拜。檔案索引表大小可藉欄位配置來包含任何合理的入口 ( Entry ) 數目。而每個欄位又可選擇性地附上唯讀 ( Read / Only ) 的及系統的屬性標記。CP/M 2.0 的用戶實體上以使用者號碼來分開，其間亦具有從一個使用者區移到另一使用者區的檔案複製動作的功能。CP/M 2.0 也擁有強有力的相對資料錄 ( Relative Record ) 隨機存取功能，這功能使得使用者能從八百萬拜 ( bytes ) 的檔案中，直接存取 65536 個資料錄中的一個。

CP/M 2.0 所有與磁碟相關的部份，都放置在 BIOS 的磁碟參數區，區內的資料由人工編入 (Hand Coded)，利用由 CP/M 2.0 提供的磁碟定義巨集指令集自動產生。使用者只需指定可用磁碟的最大數目，起始及結束的段號，資料配置的大小，每個邏輯磁碟的最大範圍，索引表大小的訊息，及要保留的磁軌數。巨集指令就會利用這些訊息，產生 CP/M 運作時所需參考的表格。還有和組譯 (Assembly) 與解組譯 (Disassembly) 磁碟段落 (sector) 大小有關的解區 (Deblock) 訊息，也由巨集指令提供；一般磁碟段落的大小是 128 拜的倍數，而系統互換手冊 (System Alternation Manual) 內含的一般用途副程式，就利用了解區訊息來使用更大的磁碟段。利用這些副程式，加上表格化的資料存取演算法 (Algorithm)，使得 CP/M 2.0 真正成爲一個普遍性的資料管理系統。

檔案的擴充，可以達到 512 個邏輯檔案延伸區 (Logical File Extent)，每個邏輯延伸區可標定 16K 拜的資料。但 CP/M 2.0 的結構令一個實體延伸 (physical Extent) 就可標定 128K 拜資料 (一個索引表入口) 如此既可維持與先前版本的相容性，又可完全利用索引空間。

CP/M 2.0 的隨機存取功能，使使用者可立即取得八百萬拜大的檔案中的一個資料錄。使用 CP/M 獨特的資料組織，資料塊 (Data Block) 在真正需要時，才會配置，磁頭移動至資料錄位置的尋找時間極短。循序檔案的存取與先前版本有高度的相容性 (可達八百萬拜)。而隨機存取的相容性，則只達到 512K 拜檔案。由於 CP/M 有較簡單且快速的隨機存取功能，因此應用程式設計者應可盡量更改他們的程式，充分利用 CP/M 2.0 的功能。

幾個 CP/M 2.0 的模組及公用程式 (Utility) 都已改進，以增強檔案系統的能力。STAT 與 PIP 兩者用來處理檔案的屬性及使用

用者區域，而 CCP 提供了一個現行指入 ( Login ) 的功能，以從一個使用者區域轉換成另一個使用者區域。對於索引的顯示，CCP 也設定了較方便的格式。而在加強行編輯功能方面，則採用 CRT 及硬複製裝置。

## CPM的功能描述

使用者經由 CCP 與 CP/M 交談，CCP 由控制台讀取命令，並對它作解釋 ( Interpret )。在一般情況下，CCP 會在幾個連線磁碟中，定址 ( Address ) 出其中一個，這個 CCP 目前定址的磁碟，我們稱其為“所指定使用”磁碟，這個動作就叫做“指定使用” ( Logged In )。為了能清楚地表示出那一個磁碟是現所指定使用的磁碟，CCP 會在螢幕上顯出該磁碟的名稱，其後則跟隨著一個“>”的符號，以表示 CCP 準備接受另一命令。在系統啓動時，CP/M 系統由磁碟 A 中載入，接著 CCP 顯示信息：

```
XX K CP/M VER m.m
```

xx 是 CP/M 管理的記憶體大小，m.m 是 CP/M 的版本號碼。起初所有的 CP/M 系統皆設定在 16 K 的記憶體空間操作，但是可再重新設定，以配合主系統記憶體大小的不同。系統啓動之後，CP/M 自動指定使用磁碟 A，並在螢幕上顯出“A>”，以告知使用者 CP/M 目前指定的磁碟為 A，且正等待命令的鍵入。命令可分成兩種形態：內建命令 ( Build-in Command ) 及暫態程式命令 ( Transient Command )。

註：標準系統定址四個不同的磁碟機，稱為 A，B，C 及 D。

## 通用命令的結構

內建命令是CCP 程式本身的一部分，而暫態程式命令則需經由磁碟載入TPA 後再執行的。內建命令有：

ERA 刪除檔案

DIR 顯示檔案索引表內的檔名

REN 檔案名的更換

SAVE 把記憶體的內容存入某一檔

TYPE 印出儲存在指定磁碟內的檔案內容

USER 在相同的索引下，搬到另一用戶區

幾乎所有的命令，都可參考到一個或同時參考一群檔，檔案名稱的格式將在下節說明。

## 檔案參考 (File References)

檔案參考用來指名連繫 ( attach ) CP/M 系統磁碟上的一個檔或一群檔。這些檔的名稱或為單一檔案名 ( Unambiguous File Name 簡稱 ufn )，或為群體檔案名 ( Ambiguous File Name 簡稱 afn )。單一檔案名指名一個單獨的檔案，而群體檔案名可同時指定數個不同的檔案。

檔案名稱包含兩個部分：主檔案名 ( Primary Name ) 與副檔案名 ( Secondary Name )。雖然副檔案名可省略，但它經常用來

## 110 Apple II 軟體卡

歸屬同類的檔案；例如，副檔案名“ASM”，表示該檔案是個以組合語言編寫的原始程式檔，而主檔案名則用來區別每個特定的原始程式。主、副兩個檔案名以“.”隔開，如下式：

pppppppp.sss

pppppppp 代表主檔案名，其字元數長度不可超過八個，sss 為副檔案名，字元數長不可超過三個。依上所述，檔案名稱

PPPPpppp

也是個正確的表示法，意義與以三個空白為副檔案名相同。用來指定單一檔案名的字元，除了下列特殊符號外，其餘的 ASCII 字元皆可使用。

<> .,: = ? \* [ ]

群體檔案名是拿來作目錄找尋 ( Directory Search ) 或模式匹配 ( Pattern Match ) 用的。除了符號“?”可散佈在主副檔案名外，群體檔案名和單一檔案名的格式是相同的。在 CP/M 各種不同的命令中，符號“?”用以表示在“?”出現的位置可為任意字元代替。因此，群體檔案名

X?Z.C?M

可以是

XYZ.COM            或            X3Z.CAM            等

單一檔的集合。又，群體檔案名

\* . \*            ( 其中 \* 代表任意字串 )

與群體檔案名 ????????.??? 的意義是相等的。而

PPPPPPPP.\* 與 \*\*\*.

則分別是 pppppppp.??? 及 ????????.sss 的縮寫。下例

DIR \*.\*

表示要列出目錄中所有磁碟檔案的 CCP 命令。同樣地，命令

DIR X?Y.C?M

會在磁碟中找尋出檔案名稱與這個群體檔案名相匹配的所有檔案。

下列的檔名都是正確的單一檔案名：

X	XYZ	GAMMA
X.Y	XYZ.COM	GAMMA.1

另外，程式設計員也可同時指名檔案名稱及磁碟機代號 ( Disk Drive Name )。磁碟機是以一個 A 到 Z 間的字母跟隨一個 “ : ” 號來指定。在檔案操作之前，要先指定使用 ( log-in ) 的磁碟機。以下為具有磁碟機代號的正確檔案名稱：

A:X.Y	B:XYZ	C:GAMMA
Z:XYZ.COM	B:X.A?M	C:*.ASM

有件事應該注意，就是在檔案名稱及磁碟機代號中的小寫字母，在 CCP 處理時，會被轉換成大寫字母。

## 磁碟機的改換

在 CCP 等待控制台的輸入命令時，操作員可鍵入磁碟機代號 ( A , B , C 或 D ) 隨著一個 “ : ” 符號來改換現所呼叫的磁碟。

在CP/M系統由磁碟機A載入後，可能會發生以下的催促信號 ( Prompt )和命令：

```

16K CP/M VER 1.4
A>DIR
SAMPLE  ASM
SAMPLE  PRN
A>B:
B>Dir *.ASM
DUMP    ASM
FILES   ASM
B>A:

```

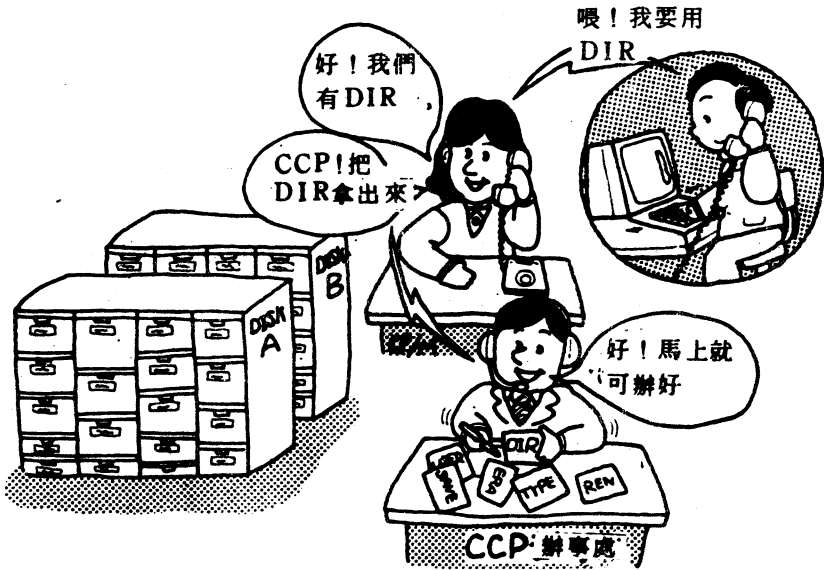
列出磁碟 A 中所有的檔案名稱

改換成磁碟機 B

列出磁碟 B 中所有的“ASM”檔案名稱

再改換回磁碟機 A

### 內建命令的格式



內建命令隨著CP/M系統經常存在主記憶體當中，要使用時不必從磁碟載入。

以上所述的檔案名稱與磁碟機代號格式現可用來充分說明系統命令的結構。在說明之前，我們先作兩個說明，一為簡寫符號的意義：

- ufn 代表單一檔案名
- afn 代表群體檔案名
- cr 代表回車鍵 (Carriage Return)

二為前面提過的，CCP 在處理命令時，總是把小寫字母轉換成大寫字母。因此，在命令與檔案名稱中的所有小寫字母都將視同大寫字母來處理。

### 刪除檔案命令 (ERASE Command)



ERA afn

刪除命令在刪除現所使用磁碟機 (即 CP/M 催促訊息中，在 “>” 之前的磁碟機代號) 中的檔案。磁碟中滿足群體檔案名的所有檔案，都會被刪除掉。下面的例子說明 ERA 命令的使用方法：

ERA X.Y 現所使用磁碟中 X.Y 檔案會被刪除，且原先所佔的空間被空出。



ERA X.\*           所有主檔案名為 X 的檔案，都從現所使用磁碟機中刪除掉。

ERA \*.ASM        所有副檔案名為 ASM 的檔案，都從現所使用磁碟中刪除掉。

ERA \*.\*           刪除現所使用磁碟中的索引目錄。(參考 USERn, 13 頁) CCP 在畫面上，會顯出訊息 ALL ( Y / N ) ?  
若要刪除所有的檔案，鍵入 Y 即可。

ERA B:\*.PRN      刪除磁碟機 B 中所有滿足群體檔案名 ????????.PRN 的檔案。現所使用的磁碟則不作任何改變。

**實例：**

```
A>DIR/
A: MOVCPM   COM : PIP            COM : SUBMIT   COM : XSUB    COM
A: ED        COM : ASM           COM : DDT      COM : LOAD    COM
A: STAT      COM : SYSGEN       COM : DUMP     COM : DUMP    ASM
A: BIOS      ASM : CBIOS        ASM : DEBLOCK  ASM : DISKDEF LIB
A: PTBIOS48 ASM : PTBOOT48   ASM : PTCPM48  COM.
```

原來磁碟 A：所存之檔如上。

A>ERA DISKDEF.LIB/

```
A>DIR/
A: MOVCPM   COM : PIP            COM : SUBMIT   COM : XSUB    COM
A: ED        COM : ASM           COM : DDT      COM : LOAD    COM
A: STAT      COM : SYSGEN       COM : DUMP     COM : DUMP    ASM
A: BIOS      ASM : CBIOS        ASM : DEBLOCK  ASM : PTBIOS48 ASM
A: PTBOOT48 ASM : PTCPM48   COM
```

刪掉其中 DISKDEF.LIB 的命令及其結果如上。

```
A>ERA *.COM/
A>DIR/
A: DUMP      ASM : BIOS      ASM : CBIOS      ASM : DEBLOCK  ASM
A: PTBIOS4B ASM : PTBOOT4B ASM
A>
```

再刪掉所有副檔案名為 COM 之命令及其結果如上。

```
A>ERA *.*
ALL (Y/N)?Y/
A>DIR/
NO FILE
A>
```

刪掉所有檔的命令及其結果如上。

## 索引目錄查尋命令 ( DIRectory Command )

DIR afn

索引目錄查尋命令將所有滿足群體檔案名 afn 的檔案名，在控制台上列出。命令

DIR

是個特別的例子，這個命令把現所使用磁碟中所有的檔案名稱在控制台上列出，它的意義與“DIR\*.\*”相同，下面所列的皆是正確的 DIR 命令。

DIR X.Y

DIR X?Z.C?M

DIR ??Y

跟 CCP 其他的命令一樣，群體檔案名前可加上磁碟機代號，以下的 DIR 命令，在作目錄找尋之前，都先指定所要使用的磁碟機。

DIR B:

DIR B:X.Y

DIR B:\*A?M

假使在所選擇的磁碟上，沒有滿足目錄查尋命令要求的檔案，在控制台上會打出“NOT FOUND”的信息。

### 實例：

```
A>DIR /
A: MOVCPM   COM : PIP           COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM          COM : DDT     COM : LOAD     COM
A: STAT     COM : SYSGEN      COM : DUMP    COM : DUMP     ASM
A: BIOS     ASM : CBIOS       ASM : DEBLOCK ASM : DISKDEF  LIB
A: PTBIOS48 ASM : PTBOOT48   ASM : PTCPM48 COM
A>
```

( 列出 A : 磁碟機所有的檔 )

```
A>DIR B:/
B: BASLIB   REL : BASCOM      COM : M80     COM : L80     COM
B: LIS      COM : CREF80     COM : TESTPRO BAS : FORLIB  REL
B: MBASIC   COM
A>
```

( 列出 B : 磁碟機所有的檔 )

```
A>DIR DUMP.ASM /
A: DUMP     ASM
A>
```

( 查證 DUMP.ASM 檔在不在 A 磁碟機內 )

```
A>DIR B:MBASIC.COM /
B: MBASIC   COM
A>
```

( 查證 MBASIC.COM 檔在不在 B 磁碟機內 )

```
A>DIR ABC.XYZ /
NO FILE
A>
```

( 查證 ABC.XYZ 檔在不在 A 磁碟機內 )

```
A>DIR B:.COM/
B: BASCOM COM : M80 COM : L80 COM : LIB COM
B: CREF80 COM : MBASIC COM
A>
```

( 列出 B : 磁碟機內所有副檔案名為 .COM 的檔 )

```
A>DIR *.ASM/
A: DUMP ASM : BIOS ASM : CBIOS ASM : DEBLOCK ASM
A: PT910S4B ASM : PT800T4B ASM
A>
```

( 列出 A : 磁碟機內所有副檔案名為 .ASM 的檔 )

```
A>DIR DUMP.*/
A: DUMP COM : DUMP ASM
A>
```

( 列出 A : 磁碟機內所有主檔案名為 DUMP 的檔 )

```
A>DIR PT????4B.*/
A: PTBIOS4B ASM : PT800T4B ASM
A>
```

( 列出 A : 磁碟機內所有主檔案名開頭為 PT 結尾為 4B 的檔 )

## 檔案易名命令( RENAME Command )



```
REN ufn2 = ufn1
```

檔案易名命令，可用來更改在磁碟上的檔案名稱。等號右邊的舊檔名，將更換成等號左邊的新檔名，現行磁碟機設定為包含舊檔案的磁碟。若使用者控制台鍵盤有左矢箭頭（←）字鍵，CCP 允許以此字鍵代替等號。REN 命令的例子如下

REN X.Y=Q.R 檔名Q.R 為X.Y 所取代

REN XYZ.COM=XYZ.XXX 檔名XYZ.XXX 為  
XYZ.COM 取代

操作員可在ufn1 或ufn2（或兩者）前加上可省略的磁碟機代號。若ufn1 前有磁碟機代號，則ufn2 被設定存有與ufn1 磁碟機相同的磁碟，同樣地，若ufn2 有磁碟機代號，ufn1 的磁碟機代號也設定為與ufn2 的相同。若ufn1 與ufn2 前皆帶有磁碟機代號，則兩磁碟機的代號必須相同。下列的REN 命令，說明了此種格式的用法。

REN A:X.ASM=Y.ASM X.ASM 取代磁碟機A  
內Y.ASM 檔

REN B:ZAP.BAS=ZOT.BAS 磁碟機B的ZOT.BAS  
檔更名為ZAP.BAS

REN B:A.ASM=B:A.BAK 磁碟機B的A.BAK檔更  
名為A.ASM

若檔名ufn2 已存在磁碟中，錯誤信息“FILE EXISTS”會顯示出來，更換的作業將因而停止，若ufn1 不存在指定的磁碟，則信息“NOT FOUND”會在控制台出現。

實例(1):

```

A>DIR /
A: MOVCPM   COM : PIP           COM : SUBMIT   COM : XSUB     COM
A: ED        COM : ASM          COM : DDT      COM : LOAD     COM
A: STAT      COM : SYSGEN       COM : DUMP     COM : DUMP     ASM
A: BIOS      ASM : CBIOS        ASM : DEBLOCK ASM : DISKDEF LIB
A: PTBIOS48 ASM : PTBOOT48     ASM : PTCPM48 COM
A>REN ABCDEFG.XYZ=BIOS.ASM /

A>DIR /
A: MOVCPM   COM : PIP           COM : SUBMIT   COM : XSUB     COM
A: ED        COM : ASM          COM : DDT      COM : LOAD     COM
A: STAT      COM : SYSGEN       COM : DUMP     COM : DUMP     ASM
A: ABCDEFG  XYZ : CBIOS        ASM : DEBLOCK ASM : DISKDEF LIB
A: PTBIOS48 ASM : PTBOOT48     ASM : PTCPM48 COM
A>REN ED.COM=PIP.COM /
FILE EXISTS
A>
    
```

實例(2):

```

A>DIR B: /
B: BASLIB   REL : BASCOM       COM : M80      COM : L80      COM
B: LIB      COM : CREF80       COM : TESTPRO  BAS : FORLIB   REL
B: MBASIC   COM
A>REN B:12345.999=B:M80.COM /

A>DIR B: /
B: BASLIB   REL : BASCOM       COM : 12345    999 : L80      COM
B: LIB      COM : CREF80       COM : TESTPRO  BAS : FORLIB   REL
B: MBASIC   COM
A>
    
```

儲存命令 ( SAVE Command )

SAVE n ufn

儲存命令是將存放在TPA 的 n 頁記憶體內容儲存到磁碟上，並給予檔案名 ufn。在CP/M 的記憶體分配系統上，TPA 的起始位址是 100H，亦即是主記憶體的第二頁。如此，假使使用者程式所佔的位址，是由 100H 到 2FFH，則 SAVE 命令必須指定二頁的記憶體，以後儲存的機器碼檔可再載入記憶體執行。以下是SAVE 的例子。

SAVE 3 X.COM 複製 100H 到 3FFH 的記憶體內容到

磁碟上，並命名為 X.COM

SAVE 40 Q

複製 100H 到 28FFH 的記憶體內容到磁碟上，並命名為 Q (註：28H=40D)

SAVE 4 X.Y

複製 100H 到 4FFH 的記憶體內容到磁碟上，並命名為 X.Y

SAVE 命令也可在 ufn 部分指出磁碟機代號，如下例。

SAVE 10 B:ZOT.COM

複製 10 頁的記憶體內容 (100H 到 0AFFH) 到磁碟機 B，並命名為 ZOT.COM (註：0AH=10D)

不管 SAVE 動作發生多少次，記憶體中的內容都不會改變。

實例：保存 2 個 pages 給 TESTSAVE.AX 檔

```
A>SAVE 2 TESTSAVE.AX
A>
```

若要檢查檔案有幾個 pages，可用後面所述的 STAT 命令：

```
A>STAT TESTSAVE.AX
```

```
Recs Bytes Ext Acc
  4    1k    1 R/W A:TESTSAVE.AX
Bytes Remaining On A: 103k
```

```
A>
```

把 Recs 那欄數字除以 2，即為頁數 因為 1 Record = 128 拜，1 頁 = 256 拜

## 顯示命令 ( Type Command )

TYPE ufn

顯示命令把現所標定磁碟上的ASCII原始程式檔 ufn，顯示在控制台上。正確的TYPE 命令如下：

TYPE X.Y

TYPE X.PLM

TYPE XXX

TYPE 命令擴張定位符號 ( clt-I 字元 )，設定定位符號位置在每列的第八行 ( Column )。ufn 檔案名可冠以磁碟機代號如下所示：

TYPE B:X.PRN 把磁碟機B的X.PRN檔案顯示  
在控制台上

### 實例(1):

A:TYPE DUMP.ASM/

```

; FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
; COPYRIGHT (C) 1975, 1976, 1977, 1978
; DIGITAL RESEARCH
; BOX 579, PACIFIC GROVE
; CALIFORNIA, 93950
;
;
ORG      100H
BDOS    EQU    0005H    ;DOS ENTRY POINT
CONS    EQU    1        ;READ CONSOLE
TYPEF   EQU    2        ;TYPE FUNCTION
PRINTF  EQU    9        ;BUFFER PRINT ENTRY
BRKF    EQU    11       ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENF   EQU    15       ;FILE OPEN
READF   EQU    20       ;READ FUNCTION
;
FCB     EQU    5CH      ;FILE CONTROL BLOCK ADDRESS
BUFF    EQU    80H      ;INPUT DISK BUFFER ADDRESS
.
.
.

```

( 顯示某一 ASM 型檔 )



實例(2):

A>TYPE B:TESTPRO.BAS/

```

10 '-----
20 '
30 '          ITI  CS  UCS  FOR MOUSE
40 '
50 '-----
60 '
70 PRINT "----- DISCRIMINATED AVOIDANCE-SAME PROGRAM WITH 4 BOXES
80 PRINT
90 '
100 '***** INITIAL SET *****
110 '
120 DEFINT A-Z 'DEFIN ALL VALIABLES TO INTEGER
130 '
140 'DISK FILE NAME DEFINE
150 INPUT "FILE NAME";FILENAME$
160 INPUT "DATE";DATEIN$
170 '
180 'DISK FILE OPEN
190 OPEN "R",#1,FILENAMEIN$
200 '
210 '--- DATA STORE AREA DIM DEFINE ---
220 'DATA BYTE BIT ASSIGNMENT
.
.
.

```

( 顯示在B磁碟機內的某一BASIC型檔 )

實例(3):

A>TYPE DUMP.COM/

```

!9"lMA"PsCB)2!eP"eZ00)(000rWZ0(0)M) 7e0C00rtyleE
000e0e_MAQaI>
NoIf"
0F0C
F70e00)q0)!          MI:~B3MN7J37I_<2!~7I/2I0MIeUE0NM0eIFILE DUMP VERSION 1.00
000(0PUT FILE PRESENT ON DISK0):e0e0M0QAQI~ H~ H~,H~
0700A0:~
070
00C": 0000
^I00"
I

```

顯示某一COM 型檔 ( 在螢光幕上將看到奇奇怪怪的字體 )

使用者命令 ( USER COMMAND )

USER n

n 為介於 0 到 15 的整數值。

在冷啟動時，操作員自動地被指定在使用者區號為零的地方，操作者可在任何時候利用 USER 命令，將使用者區域移往同一索引目錄的另一個邏輯區域。

當操作員改變使用者區域時，原先標定的磁碟機並沒有除役，因為使用者號碼只是用來找尋索引中某些特定項目的一個標記。

現役使用者號碼會一直保持不變，直到下個 USER 命令。或直到下個冷啟動時，使用者號碼才會再被設定回 0。

### 實例(1):

```

>USER 0
A>DIR
A: F80      COM : L80      COM : TEST      SUB : M80
A: LIB      COM : CREF80  COM : SUBMIT    COM : XSUB
A: AA       FOR : ED      COM : FT        FOR : DUMP
A: STAT     COM : DDT      COM : PIP       COM
A>USER 5
A>DIR
A: ABCD     BAK : PIP      COM : STAT      COM : DDT
A: ABCD     ASM : ED      COM
A>ERA *.*
ALL (Y/N)?Y
A>DIR
NO FILE
A>USER 0
A>DIR
A: F80      COM : L80      COM : TEST      SUB : M80
A: LIB      COM : CREF80  COM : SUBMIT    COM : XSUB
A: AA       FOR : ED      COM : FT        FOR : DUMP
A: STAT     COM : DDT      COM : PIP       COM
A>

```

注意，在本例中 ERA\*.\* 命令只刪掉現所指定使用者區號的檔

### 實例(2):

## 124 Apple II 軟體卡

R>STAT USR:

Active User : 0

Active Files: 0 513

A>DIR

A: F00 COM : L00 COM : TEST SUB : M00 COM

A: LIB COM : CREF00 COM : SUBMIT COM : XSUB COM

A: AA FOR : ED COM : FT FOR : DUMP COM

A: STAT COM : DDT COM : PIP COM

A>USER 5

A>DIR

A: PIP COM : ED COM : ABCD BAK : ABCD ASM

A>USER 13

A>DIR

A: PIP COM

A>USER 15

A>DIR

NO FILE

A>USER 0

A>DOT PIP.COM

DOT UERS 2.2

NEXT PC

1E00 0100

-G0

A>USER 15

A>SAVE 29 PIP.COM

A>DIR

A: PIP COM

A>USER 0

A>STAT USR:

Active User : 0

Active Files: 0 51315

A>USER 7

A>DIR

NO FILE

A>USER 0

A>STAT USR:

Active User : 0

Active Files: 0 51315

A>

## 行編輯及輸出控制 ( Line Editing and Output Control )

在鍵入命令行時，CCP 提供了某些行編輯的功能。“control”代表同時按下控制鍵及指定鍵 ( Indicated Key )。CCP 命令的長度，一般可長到 255 個字元，命令要在按下回車鍵 ( Carriage

Return) 之後才會執行。

rubout / delete 刪除並顯示最後輸入的字元。

Control C 當用在一行起頭時，可使CP/M做暖啟動。

實例：  

$$\begin{array}{l} A>^{\wedge}C \\ A> \end{array}$$

Control E 按下之後，Cursor 會跳到下一行，但鍵入的資料仍為連續，命令行要在按回車鍵之後才會送入。

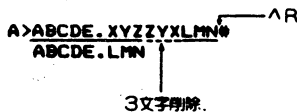
Control H 此控制鍵的作用與 Backspace 相同，即刪除最後鍵入的字元，它與 rubout / delete 的不同點就是它不會把刪除的字元也顯示出來。

Control J 結束現行輸入的命令行，並把 Cursor 移到下一行。作用與饋行一樣 (Line feed)。

Control M 作用與按回車鍵一樣

Control R 在修改的命令行下顯示修正過後的命令行。

實例：



**Control X** 將鍵入的命令整行消掉。

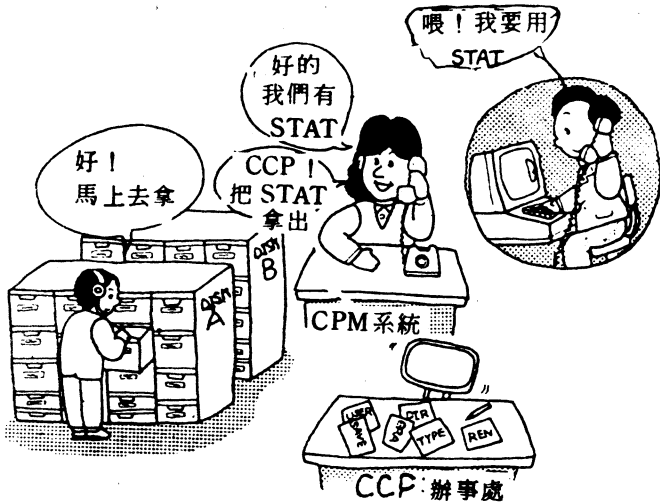
行編輯程式因可追蹤現在游標指到何字元，因此操作員可在 Control R 或 Control X 後，適當地安排對齊輸入資料。

以下之控制功能：Control P 與 Control S 會影響控制台的輸出：

**Control P** 按此鍵後，除了將資料送到控制台外，也會把同樣的資料送到印表機，直到再一次按下 Control P，才會停止把控制台的資料送到印表機。

**Control S** 此鍵會暫停控制台的輸出。程式的執行及輸出，要在下一個字元輸入（另一 Control-S）才會繼續動作。這個特性是用來暫停高速度控制台輸出，以便能清楚的觀察所顯示的段落。

## 暫態程式命令 ( Transient Command )



暫態命令不是經常存在主記憶體內，它是要使用時才到磁碟上去找出來再載入主記憶。

暫存命令由現所使用的磁碟中載入 TPA，並在 TPA 中執行。在 CCP 控制下而能執行的程式命令有下列八個。其餘的功能，可依使用者所需，自己設計。

- |      |  |
|------|--|
| STAT | ①列出現所使用磁碟所剩的記憶空間，②提供某特定檔在磁碟中所佔空間大小的訊息，③列出或改變週邊設備的安排。 |
| ASM  | 載入 CP/M 的組合編譯程式，並組合編譯磁碟上的檔案。                         |
| LOAD | 載入具 Intel 十六進位碼格式的檔案，並產生具                            |

有可載入到TPA 的可執行機器碼格式的檔案（這個被載入的程式，就成為CCP 控制下的一個新命令）。

- DDT        載入CP/M 的除錯程式到TPA ，並開始執行。
- PIP        載入週邊設備交換程式，以備未來磁碟檔案及週邊設備傳送運作之用。
- ED        載入並執行CP/M 的文稿編輯程式。
- SUBMIT    執行一內含許多命令的檔案，作為整批處理之用。
- DUMP      以十六進位格式，傾印檔案的內容。

暫態程式命令的指定方法與系統內建命令相同，其他功能的命令可由使用者簡單地定義出來。暫態程式命令也可在命令之前，冠以磁碟機代號，如此，程式會由指名的磁碟機中載入TPA 中執行，譬如，命令

B:STAT

使CP/M 暫時指定使用磁碟機B：等到載入STAT 的暫態程式以後，再回返到原來指定的磁碟，繼續以後的處理。

基本的暫態程式命令詳細如下。

### 狀態命令 (STAT Command)

STAT 命令提供有關檔案儲存空間的統計訊息及裝置分配狀態，其使用法有下列兩種格式：

**STAT**  
**STAT "command line"**

有Command line 的STAT 命令，用來檢查或改變現行裝置的分配。下列各種不同的命令，有其不同的意義。

STAT < cr > 若使用者鍵入空白的命令行，STAT 程式會計算出所有磁碟機中，磁碟所剩的可用儲存空間，而後印出下列訊息：

或是      x: R/W, SPACE: nnnK  
             x: R/O, SPACE: nnnK

X 表示各磁碟機的代號，R / W 表示磁碟機內磁碟可供讀取或寫入，R / O 表示磁碟機內磁碟只可讀取，nnn 表示該磁碟所剩的可用空間，單位為K。

**實例(1)：**

```
A>STAT J
A: R/W, Space: 106k
B: R/W, Space: 69k

A>
```

**實例(2)：**

```
A>STAT J
A: R/W, SPACE: 2K
B: R/O, SPACE: 120K
A>
```

STAT X: < cr > 假使加上磁碟機代號，在計算可用空間之前，會先指定此磁碟機，而只對該特定磁碟機作統計，所得結果以下面信息印出：

BYTES REMAINING ON B: nnnK

**實例(3)：**

```
A>STAT B:
BYTES REMAINING ON B: 170K
A>
```



## 130 Apple II 軟體卡

`STAT afn < cr >` 命令行可指名一群檔案，來讓 `STAT` 統計各檔案所佔的空間大小。執行之後滿足 `afn` 的檔案，會依檔案名稱字母順序，列出下面的信息：

RECS	BYTES	EX	D:FILENAME.TYP
rrrr	bbbK	ee	d:pppppppp.sss

`rrrr` 表示該檔案的資料錄數，每資料錄長度為 128 個 bytes，`bbb` 表示該檔案所配置的空間 ( $bbb = rrrr * 128 / 1024$ ，以 k bytes 為單位)，`ee` 表延伸區的數目 ( $ee = bbb / 16$ )，`d` 為磁碟機的代號 (A~Z)，`pppppppp` 為主檔案名，`sss` 為副檔案名。在每個檔案列出後，會再列出各檔案可用空間的總計。

### 實例(1):

```
A>STAT JOAN.1RC<cr>
RECS BYTES EX      D:FILENAME.TYP
 5    2K  1      B:JOAN.ARC
A>
```

### 實例(2):

```
A>STAT *.COM<cr>
RECS BYTES EX      D:FILENAME.TYP
 4    2K  1      A:DUMP.COM
48    6K  1      A:(ED.COM)
56    8K  1      A:PIP.COM
24    4K  1      A:STAT.COM
10    2K  1      A:SUBMIT.COM
BYTES REMAINING ON A: 218K
-A>
```

### 實例(3):

```
A>STAT B:EXAMPLE.?X?<cr>
RECS BYTES EX      D:FILENAME.TYP
48    6K  1      B:EXAMPLE.TXT
BYTES REMAINING ON B: 170K
A>
```

`STAT X:afn < cr >` 若冠以磁碟機代號於 `afn` 之前，機器首先找出指定的磁碟機，再執行“`STAT afn < cr >`”命令。

實例：

```
A>STAT B:MBASIC.COM/
      Recs Bytes Ext  d:FILENAME.TYP
      188   24k  2    B:MBASIC.COM
Bytes Remaining On B: 69k

A>
```

STAT d:filename.typ\$S < cr > 磁碟機代號d可有可無，檔案名可為群體檔案名或單一檔案名。這個命令執行之後，產生下列的輸出格式：

```
Size Recs Bytes Ext Acc
  48  48  6K  1  R/O A:ED.COM
  55  55 12K  1  R/O (A:PIP.COM)
65536 128  2K  2  R/W A:X.DAT
```

參數\$S的作用在顯示“Size”的欄位，若使用者認為不需要，此參數可以省略。“Size”欄位列出了檔案的虛擬資料錄數，而“Recs”欄位所示的是每個延伸區（Extent）虛擬資料錄數的總和。對循序檔言，“Size”與“Recs”兩欄位的值是相等的。“Bytes”欄位列出分配給該檔案的真正bytes數目。最小的分配量，在系統建構時決定，對循序檔而言，bytes數對應資料錄數與最後一次配置區（Allocated Block）餘下未用空間的總和；對隨機存取檔言，由於在寫入動作發生時，才可獲得配置的資料區，因此“Bytes”欄位中所含的值，表示真正分配給檔案的bytes數目。而“Size”欄位所含的是邏輯檔結尾的資料錄位置，“Recs”欄位的內容，則是每個延伸區邏輯資料錄數的和，當然延伸區中可能沒有資料錄，“Ext”欄位記載分配給該檔案本地延伸區（Local Extent）的數目，其中每個本地延伸區為16 K，“Acc”欄位表示該檔案的存取模式，是唯讀或可寫可讀，欄位的內容可用後面的STAT命令格式來變動。上面所示，圍繞PIP.COM的括號，表示該檔案具有系統指示集（System Indicator Set），所以無法

由 DIR 命令，看出它的存在。

實例(1):

```
A>STAT BIOS.ASM/
  Recs  Bytes  Ext  Acc
   96   12k    1  R/W  A:BIOS.ASM
Bytes Remaining On A: 106k
A>
```

實例(2):

```
A>STAT *.COM/
  Recs  Bytes  Ext  Acc
   64    8k    1  R/W  A:ASM.COM
   38    5k    1  R/W  A:DDT.COM
    4    1k    1  R/W  A:DUMP.COM
   52    7k    1  R/W  A:ED.COM
   14    2k    1  R/W  A:LOAD.COM
   76   10k    1  R/W  A:MOVCPH.COM
   58    8k    1  R/W  A:PIP.COM
   68    9k    1  R/W  A:PTCPH48.COM
   41    6k    1  R/W  A:STAT.COM
   10    2k    1  R/W  A:SUBMIT.COM
    8    1k    1  R/W  A:SYSGEN.COM
    6    1k    1  R/W  A:XSUB.COM
Bytes Remaining On A: 106k
A>
```

實例(3):

```
A > STAT PIP.COM $$/
```

Size	Recs	Bytes	Ext	Acc
55	55	12K	1	R/OA:PIP.COM

STAT d:filename.typ \$R/O< cr> 這個命令在把 filename.typ 檔或檔案群的存取模式設定為唯讀狀態，此狀態若要更動，則有賴另一個 STAT 命令的設定。唯讀狀態會記錄在含該檔案的索引上，因此在系統冷啓動後，該檔案仍是處於唯讀狀態。若檔案被標記為唯讀，任何要刪除或寫入該檔案的動作，都會引起下列的錯誤信息出現：

```
Bdos Err on D:File R/O
```

實例：

```
A>STAT BROTHER.AND SR/O<cr>
BROTHER.AND SET TO R/O
A>
```

STAT d:filename.typ \$R/W< cr > 這個命令設定檔案成爲永久的可讀可寫狀態。

實例：

```
A>STAT..$R/W<cr>
BROTHER.AND SET TO R/W
SISTER.TOO SET TO R/W.
A>
```

STAT d:filename.typ \$SYS< cr > 把檔案加上系統程式指示標記。

實例(1)：

```
B>STAT XYZ.ASM

Recs Bytes Ext Acc
  14   2k   1 R/O B:XYZ.ASM
Bytes Remaining On B: 166k

B>STAT XYZ.ASM #SYS

XYZ.ASM set to SYS
B>STAT XYZ.ASM

Recs Bytes Ext Acc
  14   2k   1 R/O B:(XYZ.ASM)
Bytes Remaining On B: 166k

B>
```

實例(2)：有 \$SYS 及 \$R/O 標誌的檔不能拿 ED 命令來編輯，如下

```
STAT *.*

Recs Bytes Ext Acc
  1    2k   1 R/O B:ABC.ASM
 64    8k   1 R/W B:ASM.COM
  4    2k   1 R/W B:(DUMP.COM)
 52    8k   1 R/W B:ED.COM
 58    8k   1 R/W B:PIP.COM
 42    6k   1 R/W B:STAT.COM
 12    2k   1 R/O B:(TEST.COM)
 14    2k   1 R/W B:(XYZ.ASM)
Bytes Remaining On B: 166k
```

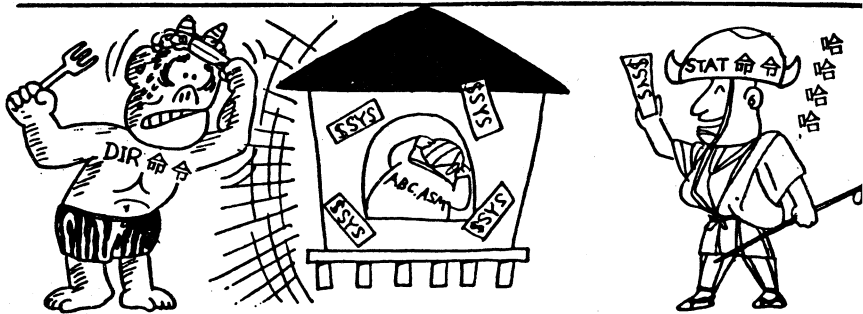
```

B>ED XYZ.ASM
"SYSTEM" FILE NOT ACCESSIBLE ←因為XYZ.ASM檔為$SYS
                               所以不能以ED命令編輯
B>PIP FD.COM=DUMP.COM
NO FILE: =DUMP.COM           因DUMP.COM檔也為$SYS
                               所以也不能複製(COPY)
B>ED ABC.ASM
** FILE IS READ/ONLY ** ←ABC.ASM檔為$R10
                          : *Q    故不能編輯
Q-(Y/N)?Y
B>
    
```

當檔案被指定為系統程式 (\$SYS) 時, DIR 命令就無法調查它的內容了。

裏面

到底是何玩意!



STAT d:filename.typ \$DIR<cr> 把檔案系統程式指示標記取消。

STAT d:DSK:<cr> 列出以“d:”指定的磁碟機的特性, 磁碟機的代號介於A到P之間, 磁碟機的特性以下的格式列出

d:	Drive Characteristics
65536:	128 Byte Record Capacity
8192:	Kilobyte Drive Capacity
128:	32 Byte Directory Entries
0:	Checked Directory Entries
1024:	Records/Extent
128:	Records/Block
58:	Sectors/Track
2:	Reserved Tracks

### 128 BYTE RECORD CAPACITY

表示使用者在磁碟可存放資料錄的最大數目，其中每個資料錄的長度為128個拜 ( byte )。

### KILOBYTE DRIVE CAPACITY

表示使用者在磁碟可存放的最大K拜數目。

### 32 BYTE DIRECTORY ENTRIES

表示使用者在磁碟可存放檔案的最大數目。

### CHECKED DIRECTORY ENTRIES

對於可移動儲存媒介 ( 磁碟 ) 的磁碟機，此值與 “ 32 BYTE DIRECTORY ENTRIES ” 所含的相同，若儲存媒介為固定的磁碟機，則此值為 0 。

RECORDS / EXTENT 每個索引項目所能定址的容量，以每個延伸具多少資料錄表示。( 例如：1024 × 128 拜，即 128 K 容量 )

RECORDS / BLOCK 基本配置容量 ( 例如：每塊 128 個資料錄，乘每資料錄 128 拜 ( 即每塊可存 16 K 拜 ) )

### RESERVED TRACKS

表示不可用來儲存檔案 ( 保留 ) 的磁軌數。

實例：

```
A>STAT B:DSK.<cr>
B: DRIVE CHARACTERISTICS
4096: 128 BYTE RECORD CAPACITY
512: KILOBYTE DRIVE CAPACITY
128: 32 BYTE DIRECTORY ENTRIES
128: CHECKED DIRECTORY ENTRIES
128: RECORDS/EXTENT
16: RECORDS/ BLOCK
58: SECTORS/ TRACK
2: RESERVED TRACKS
A>
```

STAT DSK: < cr > 列出目前所有可使用磁碟機的特性。

實例：

```
A>STAT DSK.<cr>
A: DRIVE CHARACTERISTICS
4096: 128 BYTE RECORD CAPACITY
512: KILOBYTE DRIVE CAPACITY
128: 32 BYTE DIRECTORY ENTRIES
128: CHECKED DIRECTORY ENTRIES
128: RECORDS/EXTENT
16: RECORDS/ BLOCK
58: SECTORS/ TRACK
2: RESERVED TRACKS
B: DRIVE CHARACTERISTICS
4096: 128 BYTE RECORD CAPACITY
512: KILOBYTE DRIVE CAPACITY
128: 32 BYTE DIRECTORY ENTRIES
128: CHECKED DIRECTORY ENTRIES
128: RECORDS/EXTENT
16: RECORDS/ BLOCK
58: SECTORS/ TRACK
2: RESERVED TRACKS
A>
```

STAT USR: < cr > 產生目前定址的磁碟上有檔案的使用者代號表，並以下列的格式印出：

```
A>STAT USR.<cr>
ACTIVE USER: 0
ACTIVE FILES: 0 1 3
A>
```

第一行列出目前定址的使用者代號，這個代號是最後一個 CCP 的命令 — USER 命令所定的，接著是掃描目前索引所得到的使用者代號。在上面的例子中，現役的使用者代號為 0，而在目前磁碟中使用檔案的

使用者代號有三個(代號 0, 1, 3)。操作員可利用 USER1、USER2 或 USER3 來呼叫使用者區域, 然後在 CCP 層次上, 以 DIR 命令審查這些使用者代號的索引。

STAT 命令也可用來處理實體及邏輯裝置的安排問題, 讀者可參考 CP/M 界面導引的 IOBYTE 功能, 一般而言, 在任何特定的瞬間, 系統中有四個邏輯週邊設備存在, 每個邏輯週邊設備分配給數個實體週邊裝置之一。這四個邏輯設備叫做:

CON : 下達命令及顯示訊息的系統控制台裝置

RDR : 接受訊息的讀紙帶裝置

PUN : 送出訊息的紙帶打卡裝置

LST : 列印訊息的列表裝置

任何特定電腦系統連接的實際裝置都由 CP/M 的 BIOS 副程式所控制。如此, 邏輯設備 RDR: 裝置, 可以是高速閱讀機、電傳打字閱讀機 (Teletype Reader) 或卡帶。爲了增加某些裝置命名與分配的彈性, 以下是電腦週邊設備許多實體裝置的定名:

TTY : 電傳打字機 (慢速控制台)

CRT : 陰極射線管裝置 (高速控制台)

BAT : 整批處理 (控制台是目前指定的 RDR:, 輸出爲目前的  
LST: 裝置)

UC1 : 使用者定義的控制台

PTR : 讀紙帶機 (高速閱讀機)



UR1 : 使用者定義的閱讀機 # 1

UR2 : 使用者定義的閱讀機 # 2

PTP : 紙帶打孔機 ( 高速打孔機 )

UP1: 使用者定義的打孔機 # 1

UP2 : 使用者定義的打孔機 # 2

LPT : 行印表機

UL1 : 使用者定義的列表裝置 # 1

有一點必須跟讀者強調，實體裝置的名稱，並不一定真正對應名稱所意味的裝置。意即，如果使用者願意的話，PTP：裝置可用來執行寫入卡帶的功能。真正的對應和控制副程式定義在 CP / M 的 BIOS 部分。在標準的 CP / M 版本中，這些裝置名稱與 MDS 800 發展系統上的相同。

命令：

STAT VAL: <cr>

列出 STAT 命令的各種可用的型態總整理，其輸出如下：

```
A>STAT VAL: }
```

```
Temp R/O Disk: d:=R/O
```

```
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
```

```
Disk Status : DSK: d:DSK:
```

```
User Status :USR:
```

```
Jobyte Assign:
```

```
CON: = TTY: CRT: BAT: UC1:
```

```
RDR: = TTY: PTR: UR1: UR2:
```

```
PUN: = TTY: PTP: UP1: UP2:
```

```
LST: = TTY: CRT: LPT: UL1:
```

```
A>
```

在任何情況下，左邊的邏輯設備可挑選右邊四個實體設備中的任一個。鍵入以下的命令，可獲得邏輯與實體設備的對照表。

```
STAT DEV: <cr>
```

表中的左側為邏輯裝置，右側為現行對應的實體裝置，例如：

```
A>STAT DEV: ↓
CON: is CRT:
RDR: is PTR:
PUN: is PTP:
LST: is LPT:

A>
```

現行的邏輯及實體裝置的安排，可經由如下的 STAT 命令來作改變：

```
STAT ld1 = pd1, ld2 = pd2, ..., ldn = pdn <cr>
```

ld1, ld2, …… , ldn 表邏輯裝置，而 pd1, pd2, …… , pd3 表實體裝置，以下即是用來改變現行邏輯對實體裝置安排的正確 STAT 命令：

```
STAT CON: = CRT: <cr>
STAT PUN: = TTY:, LST: = LPT:, RDR: = TTY: <cr>
```

實例：

```
A>STAT DEV: ↓
CON: is CRT:
RDR: is PTR:
PUN: is PTP:
LST: is LPT:

A>STAT RDR: = TTY:, PUN: = UP1: ↓

A>STAT DEV: ↓
CON: is CRT:
RDR: is TTY:
PUN: is UP1:
LST: is LPT:

A>
```

**組合編譯命令 ( ASM Command )**

ASM ufn

ASM 命令載入並執行 CP / M 8080 的組合編譯程式。ufn 指名一個含組合語言指令的源程式檔案，檔案的副檔案名設定為 ASM ，因此可以省略不寫。以下即是正確的 ASM 命令：

ASM X

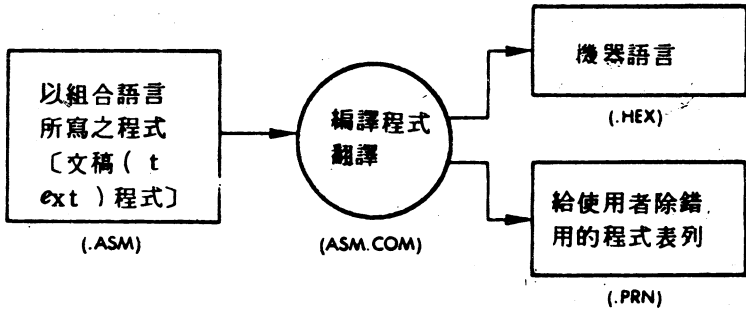
ASM GAMMA

兩次處理 ( Two-Pass ) 的組合編譯程式於鍵入組合編譯命令後會自動執行，在第二次處理時，若有組合編譯錯誤發生，錯誤會在控制台上列出。

在組合編譯後，會產生一 X . PRN 檔，X 為在 ASM 命令中的主檔案名。PRN 檔案中含有一份源程式及指令的機器碼；編譯過程中，若有錯誤發生，也會列在 PRN 檔案中。PRN 檔的內容可用 TYPE 命令在控制台列出，也可用 PIP 命令送到週邊設備印出。值得一提的是，除了最左邊的 16 個字元 ( 程式的位址及機器碼 ) 外，PRN 檔所含的內容與源程式檔 ( ufn ) 一模一樣，所以在使用機器過程中，若不小心把源程式毀了，讀者可利用 ED 系統程式，將

PRN 檔案每行的最左邊 16 個字元拿掉，再利用 REN 命令把 PRN 改成 ASM ，就可獲得與原先一樣的源程式檔。

除了 PRN 檔外，還有一個含有以 Intel 十六進位格式表示的 8080 機器語言檔案 X . HEX 也會在執行中產生。這個檔以後可經 LOAD 命令載入及執行，讀者可參考 LOAD 命令。若要了解 CP / M 組合語言程式的細節，可參考“CP / M 組合語言使用者導引手冊”。



與其他的程式命令一樣，組合編譯命令的 `ufn` 前，也可冠以磁碟機代號。例如，以下的命令

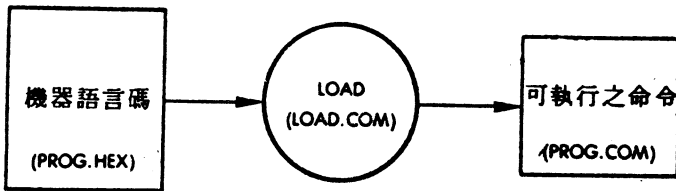
```
ASM B:ALPHA <cr>
```

將使機器從現所使用的磁碟機中，載入組合編譯程式，來編譯存在磁碟機 B 中的源程式檔 ALPHA，執行後產生的 PRN 檔與 HEX 檔，會存放在磁碟機 B 的磁碟上。

### 載入命令 ( Load Command )

```
LOAD ufn < cr >
```

載入命令從磁碟讀入含有十六進位格式的機器碼檔案 `ufn`，然後產生可用來在記憶體上執行的記憶體映像檔 ( Memory Image Code )。檔案名 `ufn` 的副檔案名必須為 HEX，所以通常省略不寫，LOAD 命令執行後會建立一個含有機器可執行碼的 X.COM 檔，其中 X 為 `ufn` 的主檔案名。當使用者在 CCP 等待輸入的 “>” 催促符號後鍵入 X，檔案 X.COM 就會被載入記憶體中執行。



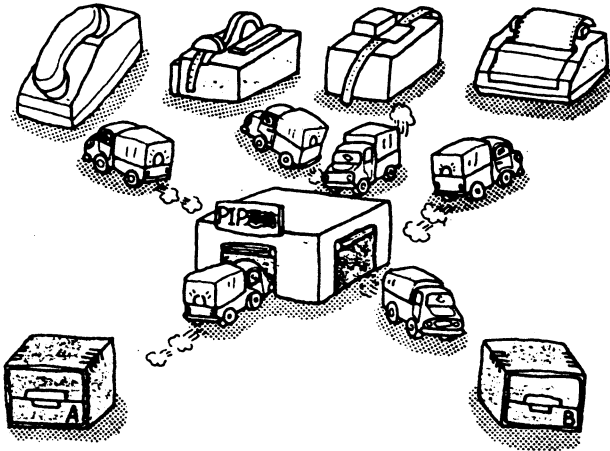
一般說來，CCP 讀取跟隨在“>”符號後的檔名X以後，會在內建系統命令中，找尋一個與X相同的名稱，若找不到，CCP 接著又在系統磁碟的索引，繼續找尋一個叫檔X.COM的檔案名，如果找到，該檔案所存的機器碼，會載入TPA 中，程式便開始執行起來。因此使用者只須LOAD 一次HEX 檔，以後只要鍵入主檔案名，就能不限次數地執行。這樣，使用者即可根據本身的需要，在CCP 控制下，創造出一新的暫態命令。如在檔案名前冠以磁碟機代碼，則作業會在交替的磁碟機上執行，如命令

LOAD B:BETA

會從現所使用的磁碟中把LOAD 程式載入TPA ，接著載入存放在磁碟機B的檔案BETA 。

有一點必須提醒讀者，檔案BETA.HEX 所含的必須是正確的Intel 格式的十六進位機器碼資料錄，程式的起始位址在100H，也就是TPA 的起頭。而且，hex資料錄的位址必須由小到大，在讀入HEX 資料錄時，未填滿的記憶區的空位，LOAD、命令會以零填充。因此，LOAD 只可用來建立在TPA 運作的CP/M 標準COM 檔案，對於要使用TPA 以外記憶區的程式，要用DDT 命令方可載入。

## 週邊設備交換程式命令 ( PIP Command )



PIP主掌各元件間之資料傳送

PIP 是CP/M的週邊設備交換程式 ( Peripheral Interchange Program )，用來完成載入、列表、打孔、複製及組合磁碟檔所需的基本媒介轉換動作。PIP 的用法有下列兩種：

```
PIP <cr>  
PIP " 命令行 " <cr>
```

這兩種形式，都會把PIP 載入TPA 中。在第一種形式下，會有“\*” 催促信號出現要求使用者輸入命令行，讀者可反覆地輸入命令行，直到所鍵入的命令行是< cr > 時為止，每個連續的命令行，會依照即將談到的規則，執行某種媒介轉換。

```
實例(1):  A > PIP /  
           *B: COPY2.BAK = FILE2.TXT /  
           *A: = B: SAMPLE.BAS /  
           *A: = B: PROG.FOR /  
           * /  
           A >
```

第二種形式，在執行字所指定命令行動作後，馬上把控制權交還給 CCP 。

```
實例(2):  A > PIP B: COPY1.BAK = FILE1.TXT /  
           A >
```

命令行的格式如下：

`destination = source #1, source #2, ..., source #n (cr)`

destination 必須是個可接受資料的檔案或週邊裝置，而 source #1, source #2, ..., source #n 代表一串要從右邊複製到左邊的檔案或裝置。

若在命令行的右邊是多重檔案，即 n 的值大於 1，則個別的檔案所含的內容，必須是 ASCII 字元，並且在每個檔案的最後，要有個檔結尾 (End-of-File)。若讀者的控制台上具有“←”的字鍵，命令行中的“=”符號，可以“←”代之，以增加可讀性。另外，命令行的長度，不可超過 255 個字元，對於超過控制台寬度的命令行，可用 `ctl-E` 把 Cursor 移到下一行。

destination 及 Source 元素可用 CP/M 的單一檔名稱，前面亦可冠以磁碟機代號 (此即代表所有磁碟機上的檔都可彼此互相參

考)。若沒有冠以磁碟機代號，則以現所使用的磁碟機為對象。除此之外，destination 的檔案，也可是 Source 檔案中的一個，如此，機器會先將各 Source 檔複製完畢後，再改變該 Source 檔的內容，如果 destination 檔早存在，而命令行也是正確的，則原先的 destination 檔會被蓋掉，但若執行中有錯誤發生，原先的 destination 檔保持不變。以下是一些執行 PIP 的正確命令：

$X=Y < cr >$  將 Y 檔案的內容，複製到 X 檔案中。

$X=Y,Z < cr >$  連接 Y, Z 兩檔案後複製到檔案 X 中。

$X.ASM=Y.ASM,Z.ASM,FIN.ASM < cr >$  連接  
Y.ASM, Z.ASM, FIN.ASM 後複製到  
X.ASM 的檔案中。

$NEW.ZOT=B:OLD.ZAP < cr >$  將磁碟機 B 中的  
OLD.ZAP 複製到現所使用磁碟機的  
NEW.ZOT 檔案上。

$B:A.U=B:B.V,A:C.W,D.X < cr >$  連接磁碟機 B 中的  
B.V, 磁碟機 A 中的 C.W, D.X 三個檔案  
，複製到磁碟機 B 的檔案 A.U。

為了方便起見，PIP 在不同磁碟間的檔案轉移方面，另有些較簡單的命令，其使用型式如下：

PIP x: = afn < cr >

PIP x: = y:afn < cr >

PIP ufn = y: < cr >

PIP x:ufn = y: < cr >



第一種命令型式，將現所使用磁碟上，所有合乎檔案群  $afn$  的檔，複製到磁碟機  $x$ ，且檔案名稱不變。第二種型式和第一種相似，所不同的是  $source$  檔存在磁碟機  $y$  的磁碟上。第三種型式的意義，與命令 “ $PIP\ ufn = y : ufn$ ” 相同，將磁碟機  $y$  上檔名為  $ufn$  的檔案，拷貝到現所使用磁碟機上的檔案  $ufn$ 。第四種型式與第三種相似，唯拷貝的目的地是磁碟機  $x$  的  $ufn$  檔案。

注意，以上所提的簡便型式，在使用時有個首要條件，就是源磁碟 ( Source Disk ) 與目的磁碟 ( Destination Disk ) 必須不同一個。假使指定的檔案名稱，是個群體檔案名，則在拷貝時， $PIP$  會列出每個合乎  $afn$  的單一檔案  $ufn$ 。若目的檔案 ( Destination File ) 所指定的檔案名，已經存在磁碟中，則在複製成功之後，原先的檔案會被蓋掉而代之以新複製進來的檔案。

以下是些磁碟與磁碟間複製作業的正確  $PIP$  命令：

$B : = * . COM < cr >$  將現所使用磁碟機上，副檔案名為  
“  $COM$  ” 的所有檔案複製到磁碟機  $B$   
的磁碟上。

### 實例(1):

```
A>PIP B:=*.COM/
COPYING -
MOVCPM.COM
PIP.COM
SUBMIT.COM
XSUB.COM
ED.COM
ASM.COM
DDT.COM
LOAD.COM
STAT.COM
SYSGEN.COM
DUMP.COM
PTCPM48.COM
A>
```

A:=B:ZAP.\* < cr > 將磁碟機B主檔案名為ZAP的所有檔案，複製到磁碟機A的磁碟上。

實例(2): A>PIP A:=B:MBASIC.COM(V)!  
A>DIR MBASIC.COM!  
A: MBASIC COM  
A>

ZAP.ASM=B: < sr > 同 ZAP.ASM=B:ZAP.ASM

B:ZOT.COM=A: < cr > 同 B:ZOT.COM=A:  
ZOT.COM

B:=GAMMA.BAS < cr > 同 B:GAMMA.BAS=  
GAMMAS.BAS

實例(3-1):

A>PIP B:=STAT.COM!  
A>DIR B:STAT.COM!  
B: STAT COM  
A>

實例(3-2):

A>PIP B:STATPIP.COM=STAT.COM!  
A>DIR B:STATPIP.COM!  
B: STATPIP COM  
A>

B:=A:GAMMA.BAS < cr > 同 B:GAMMA.BAS=A:  
GAMMAS.BAS

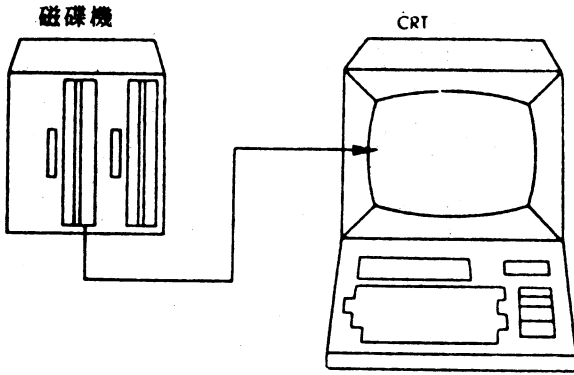
PIP 也可用來參考與 CP/M 系統連繫在一起的實體及邏輯裝置。這些裝置名稱除了有些與 STAT 命令舉出的相同外，尚有其他特殊的裝置名稱。命令 STAT 舉出的邏輯裝置有 CON：(控制台)，RDR：(閱讀機)，PUN：(打孔機)，及 LST：(列表機)，而實體裝置則有：TTY：(控制台、閱讀機、打孔機或列表機)，

CRT: (控制台、列表機), UC1: (控制台), PTR: (閱讀機),  
 UR1: (閱讀機), UR2: (閱讀機), PTP: (打孔機),  
 UP2: (打孔機), LPT: (列表機), UL1: (列表機)。注意,  
 BAT: 實體裝置並未列入其中, 因為這種配置只是用來表明  
 RDR: 和 LST: , 是作為控制台的輸入與輸出。

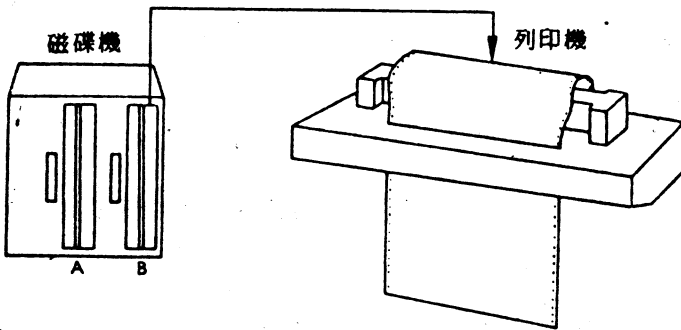
RDR, LST, PUN, 及 CON 都在 CP/M 的 BIOS 部分定義, 因此對於任何特殊的 I/O 系統, 只要經過簡單的修改即可適應。現行實體裝置的對映 (Mapping) 由 IOBYTE 定義, 若要參考此類的功能, 讀者可參考“CP/M 介面導引”手冊 (第3部份第2章)。目標裝置必須能夠接受資料, 而源裝置則必須能製造產生資料。對前者, RDR: 就不適用, 後者, LST: 也不適用。

實例: 以下為一 PIP 命令及其執行過程:

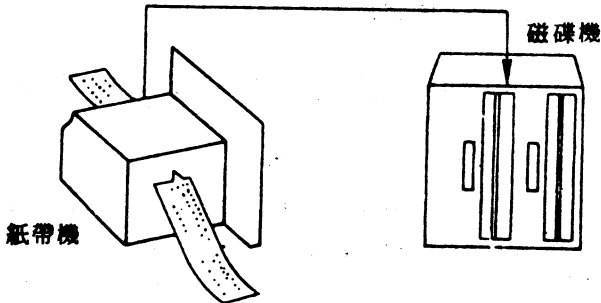
```
A > PIP /
*CON: = SAMPLE.TXT /
*LST: = B:SIMPLE.BAK /
*PROG.BAS = RDR: /
*PUN: = PROG.BAS /
* /
A >
```

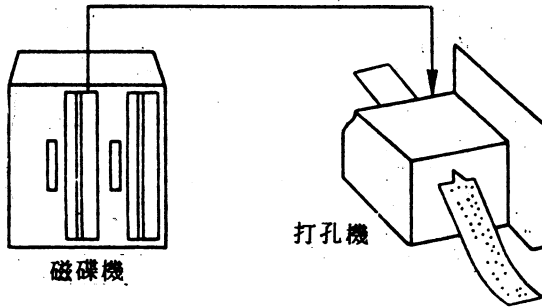


CON: = SAMPLE.TXT



LST: = B: SIMPLE.BAK





**PUN: = PROG.BAS**

注意，本圖中CON:，LST:，RDR:，PUN: 皆可重新定義（見本手冊第2章）為其他種類輸出入裝置元件。而非固定一種類元件而已。

PIP 命令額外的裝置名稱，如下表所示：

NUL：送 40 個空白（00H）到所指定的裝置（可在打孔輸出結束後送出）。

EOF：送出一個 CP/M 的檔結尾記號（ctl-Z）到目的裝置（在拷貝 ASCII 型式的磁碟檔案結束時，PIP 會自動送給目的裝置這個 EOF：）。

INP：從一個使用者建立的特別 PIP 輸入源（Input Source）中取出訊息。

OUT：送出訊息到一個使用者建立的特別 PIP 輸出目的地（Output Destination）。

讀者可在 PIP 中加上特別的輸入及輸出的常規程式。位址 103H, 104H 及 105H 是保留給跳越到讀者的特別輸入常規程式的跳越指令 ( Jump Instruction ) 用的, PIP 呼叫位址 103H, 並輸入一個字元, 然後返回, 讀者可在位址 109H 發現這字元。位址 106H, 107H, 108H 則保留給跳越到讀者的特別輸出常規程式的跳越指令, PIP 把要傳送的字元, 載入到暫存器 C ( Register C ), 然後呼叫位址 106H。除此之外, 在位址 10AH 到 1FFH 的記憶體, 讀者可用 DDT 命令插入自己的特殊推動程式。

PRN : 和 LST: 相同, 但是在每第八個字元的位置, 會設下一個延伸字元, 而所有的資料行會加以編號, 它在啓始時跳頁, 且每六十行會再跳頁。

檔案及裝置名稱可散佈在 PIP 命令中。在任何情況下, 機器會讀取指定的裝置, 直至檔結尾字元 ( ASCII 檔是 ct1-Z, 至於非 ASCII 型式的檔案, 則是 EOF ) 爲止。各裝置或檔案的資料, 由左至右讀入組合, 直到最後一個資料源 ( Data Source ) 讀完爲止。目的裝置或利用源檔的資料寫入的檔案, 當資料寫入結束時, 若資料來自 ASCII 型源檔, 則機器會加寫上一檔結尾字元。有一點要提醒讀者, 假若目的地 ( Destination ) 是個磁碟檔案, 機器會先產生一個暫存檔案 ( 副檔案名爲 \$\$\$ ), 直到複製作業成功後, 再把副檔案名轉換成真正的檔名。對於具有延伸字元 "COM" 的檔案, 機器將視之爲非 ASCII 型態檔。

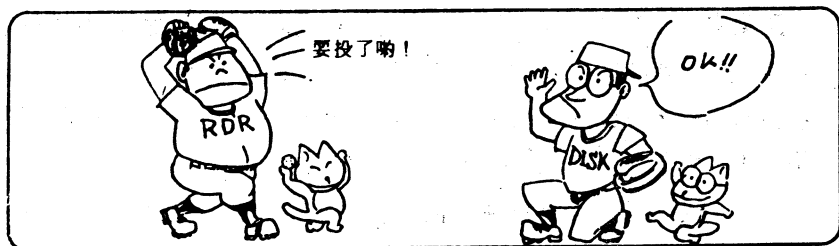
在複製作業時, 使用者可在鍵盤上鍵入任何鍵, 以終止複製動作。此時, PIP 會回應 "ABORTED" 的訊息, 表示作業沒有順利完成。在 PIP 執行時, 若有作業被中止, 或在處理時有錯誤發生, PIP 會剔除用到 SUBMIT 命令的所有的剩餘命令。

另外有一點也是值得讀者注意的，就是假使目標元素是“HEX”型磁碟檔案，而源元素是個週邊裝置（如讀紙帶機），PIP 會執行一種特殊的功能；PIP 程式先檢查源檔（Source File）內的資料錄，是否具有合法的十六進位格式與偵錯欄（Checksum）。如果發現有不正確的輸入資料錄，PIP 會在控制台顯示錯誤訊息，並等待更正的動作。通常在這種錯誤發生時，使用者可開啓閱讀機，重新閱讀一段紙帶（將紙帶拉出約 20 英吋），當紙帶準備重讀時，在控制台鍵入一個回車鍵（Carriage Return），PIP 就會嘗試另一閱讀動作，假使仍然無法正確地閱讀，使用者只要在錯誤訊息出現後，鍵入 Return 鍵，繼續閱讀下去，等磁碟檔案建立之後，再利用 ED 程式來修飾。為了方便起見，假如源檔案是個 RDR：裝置，

PIP 允許使用者在控制台上鍵入檔結尾的記號。在這情況下，PIP 一邊閱讀裝置，一邊監督鍵盤，如果由鍵盤鍵入 `ctl-Z`，閱讀的作業會正常停止。

以下是一些正確的 PIP 命令：

PIP X.PRN=RDR: < cr > 把 RDR：裝置所收的資料傳到 X.PRN 檔，中止 PIP 程式。



PIP < cr > 啓動PIP 程式，以執行一系列的命令（PIP 在螢幕上顯示“\*”符號，以等待使用者輸入命令）。

\*CON:=X.ASM,Y.ASM,Z.ASM< cr > 連接檔案 X.ASM,Y.ASM及Z.ASM，並複製到 CON:裝置。

\*X.HEX=CON: ,Y.HEX,PTR:< cr > 從CON:裝置讀取資料，直至檔結尾出現，接著讀取Y.HEX，最後讀取PTR:，直至ctl-Z 為止，連接以上資料，建立檔案X.HEX。

\*< cr > 結束PIP 程式。

PIP PUN:=NUL: ,X.ASM,EOF: ,NUL:< cr > 送出 40 個空白字元( Null Character ) 到打孔裝置，接著把檔案X.ASM 的資料，複製到打孔裝置，再送出檔結尾記號，最後再送 40 個空白字元。

使用者也可在PIP 的命令行，指定一個或數個PIP 參數，參數間以空白或零隔開，參數群前後以左、右中括號( [ ] ) 圍繞。每個參數皆會影響複製的作業，而中括號內的參數列必須緊跟在所要影響的檔案或裝置後面。一般說來，每個參數後可依需要加入一個十進位整數（參數S與Q除外）。下面即是各種PIP 的參數，及其功能：

B 整區資料傳送模式( Block Mode Transfer ): PIP



從源元素將資料傳送到緩衝區 ( Buffer )，直至收到一個 ASCII x-off 的字元 ( ctl-S )後，再將緩衝區的資料送給指定目標元素，然後 PIP 又回頭接受源元素的資料，這個參數通常是用在將紙帶的資料讀入檔案，至於緩衝區的大小，則視主系統 ( Host System ) 記憶體的大小而定，若緩衝區有溢位 ( Overflow ) 現象，PIP 會發出錯誤訊息。

**Dn** 由源元素傳送到目標元素資料中，除去第n字元以後所有字元，這個參數通常用來除去源元素中一些不必要的字元。

實例：

```
A>PIP
*LST:=DNTEST
ABCDEFGHIJKLM
NOPQRSTUVWXYZ
ABCDEFGHIJKLM
NOPQRSTUVWXYZ
*LST:=DNTEST[D7]
ABCDEF6
NOPQRST
ABCDEF6
NOPQRST
*
```

**E** 將 PIP 的作業過程回印 ( echo ) 在控制台上。

實例：

```
A>PIP B:=CBIOS.ASM[E]
; Skeletal CBIOS for first level of CP/M 2.0 alteration
;
msize equ 20 ;cp/m version memory size in kilobytes
;
; "bias" is address offset from 3400H for memory systems
; than 16K (referred to as "b" throughout the text).
;
```

```

bias    equ    (msize-20)*1024
ccp     equ    3400H+bias      ;base of ccp
bdos    equ    ccp+806h       ;base of bdos
bios    equ    ccp+1600h      ;base of bios
cdisk   equ    0004H         ;current disk number 0=A,...,15=P
iobyte  equ    0003h         ;intel i/o byte
;
;
;
nsects  equ    bios           ;origin of this program
         equ    ($-ccp)/128    ;warm start sector count
;
;
;

```

- F** 除去檔案中的饋表字元 ( form feed ) 與 P 合用插入新饋表字元。
- Gn** 這個參數可使 PIP 從另一個使用者區域複製檔案到現行的使用者區域，十進位數 n 代表原始使用者區域代號。如果操作員在 CCP 下已發出命令 USER4，則 PIP 敘述指令 ( statement ) PIP X.Y=X.Y [G2] 會從使用者代號 2 中讀取檔案 X.Y 到使用者代號為 4 的區域。使用者只能將檔案複製到目前 USER 命令所定的區域，其餘的區域是不行的。
- H** Hex資料傳輸 ( Hex Data Transfer )，這個參數用來檢查所要傳輸的檔案是否合乎 Intel十六進位的檔案格式。hex 資料錄之間的不需要字元在複製過程中被去除。若有錯誤出現，在控制台上會有錯誤訊息出現要求改正。
- I** 作用與H相同，只是對於“ :00 ”的資料錄不作轉換。
- L** 將所有的大寫字母轉換成小寫字母。

實例：

```

A>PIP
*LIST=TEXT.FOR
  WRITE(5,100)
  100 FORMAT(1H ,10HABCDEF GHIJ)
  STOP
  END
*LIST=TEXT.FOR[1]
  write(5,100)
  100 format(1h ,10habcdefg hij)
  stop
  end

```

N 把轉換到目標元素的每一行加上行編號，起始值為 1，增值亦為 1，沒有前導的零 (Leading Zero)，行編號的後頭跟隨一個冒號。若參數為 N2，則前導的零會出現，且在行編號後有 tab 跟著，如設定 T 參數會有擴張的 tab 作用。

實例(1)：

```

A>PIP
*CON1=TEXT.FOR
  WRITE(5,100)
  100 FORMAT(1H ,10HABCDEF GHIJ)
  STOP
  END
*CON1=TEXT.FOR[N]
  1:   WRITE(5,100)
  2:  100 FORMAT(1H ,10HABCDEF GHIJ)
  3:   STOP
  4:   END
*CON1=TEXT.FOR[N2]
000001  WRITE(5,100)
000002  100 FORMAT(1H ,10HABCDEF GHIJ)
000003  STOP
000004  END

```

實例(2)：

```

A>PIP CON1=PTBIOS48.ASM /
; CP/M BASIC INPUT/OUTPUT OPERATING SYSTEM (BIOS)
; TARBELL ELECTRONICS
; 2.2 VERSION OF 2-19-80
; CHANGED SIGN-ON FOR CPM 2.2 2-19-80
;
;----- CUSTOMARISED BY YASU HARU MURASE -----
;-----          8-6-80          -----
;
; THIS MODULE CONTAINS ALL THE INPUT/OUTPUT
; ROUTINES FOR THE CP/M SYSTEM, INCLUDING

```

```
; THE DISK ROUTINES.
;
; THIS SECTION DEFINES THE I/O PORTS AND
; STATUS BITS. BY SETTING THE PROPER VALUES
; FOR THE EQU STATEMENTS, THE I/O MAY BE
; AUTOMATICALLY RECONFIGURED TO FIT MOST
; SITUATIONS. THE TRUE AND FALSE ONES
; CONTROL CONDITIONAL ASSEMBLIES OF DIFFERENT
; SECTIONS OF I/O ROUTINES TO FIT DIFFERENT
; INTERFACE REQUIREMENTS.
```

```
A>PIP CON:=[PTBIOS48.ASM[ND30]]
```

```
1: ; CP/M BASIC INPUT/OUT
2: ; TARBELL ELECTRONICS
3: ; 2.2 VERSION OF 2-19-
4: ; CHANGED SIGN-ON FOR
5: ;
6: ;----- CUSTOMARISED BY
7: ;----- 8-6-8
8: ;
9: ; THIS MODULE CONTAINS
10: ; ROUTINES FOR THE CP/
11: ; THE DISK ROUTINES.
12: ;
13: ; THIS SECTION DEFINES
14: ; STATUS BITS. BY SET
15: ; FOR THE EQU STATEMEN
16: ; AUTOMATICALLY RECONF
17: ; SITUATIONS. THE TRU
18: ; CONTROL CONDITIONAL
19: ; SECTIONS OF I/O ROUT
20: ; INTERFACE REQUIREMEN
```

A>

本例首先用 PIP 命令把檔在 CON:裝置上展示出來，然後再用 PIP 命令來加上行號及刪除第 30 字元後的字元 [ ND30 ] 並把修改後的檔在 CON:裝置上展示出來

O 目標檔案 ( Object file ) ( 非 ASC II ) 的轉換，CP/M 的檔結尾記號被視同一般字元，不發生作用。

Pn 在每 n 行之後插入跳頁信號 ( 列印前會先跳一頁 )，如

果  $n = 1$  或者不指定  $n$  值，每 60 行會有一跳頁。

**Qstz** 複製過程中，遇到字串  $s$  後終止複製工作。（字串  $s$  以 `ctl-z` 結束之）。

**R** 閱讀系統檔案，這個參數的指定，使得有系統屬性的檔案，也可用 PIP 命令複製，否則在檔案複製作業中，系統檔案會被忽略掉。

**Sstz** 在遇到字串  $s$  時，開始作複製工作。參數  $S$  和  $Q$  可用來摘錄檔案中的某一段落。用來開始和終止複製的字串都包括在複製動作中複製之。

注意，若使用含有命令行的 PIP 命令，則跟隨在  $S$  及  $Q$  參數後的字串  $s$ ，會轉換成大寫字母。但若為不含命令行的 PIP 命令，就不會做這種自動轉換的工作。

**Tn** 當從 `source` 到 `Destination` 時，使擴張定位符號 `tab(ctl-I 字元)` 設在每第  $n$  個字元空間。

**U** 在複製作業時，把小寫字母轉換成大寫字母。

**V** 在複製完畢後，重讀複製的資料，以驗證複製作業是否無誤。（注意，目標元素必須是磁碟檔案）

**W** 將資料寫入檔案狀態為  $R/O$  的檔案，而無須經由控制台的查訊。在正常作業中，PIP 不會自動把資料蓋寫一個設定為  $R/O$  狀態的檔案，而控制台會提醒使用者該檔案為  $R/O$  狀態，並等待蓋寫的許可。參數  $W$  使得使用者能省掉這種查訊步驟。

Z 把每個輸入的ASCII字元的檢查比次 ( bit ) 設為零。

以下是含有檔案傳輸參數的正確PIP 命令：

```
PIP X.ASM=B:[v]<cr>
```

將磁碟機B上的檔案X.ASM複製到現所指定的磁碟機上，並證驗複製是否正確。

```
PIP LPT:=X.ASM[nt8u]<cr>
```

將檔案X.ASM複製到LST:的裝置上，每行加上行編號，並把小寫字母轉換成大寫。

```
PIP PUN:=X.HEX[i],Y.ZOT[h]<cr>
```

首先把檔案X.HEX複製到PUN:的裝置，但尾隨“:00”的資料錄不予複製，接著複製含hex資料錄的Y.ZOT，而“:00”的資料錄也會複製過去。

```
PIP X.LIB=Y.ASM[sSUBR1:↑zqJMP L3↑zJ<cr>
```

把檔案Y.ASM中字符串SUBR1: 與字符串JMP L3 之間的段落，含SUBR1: 與JMP L3 本身，複製到檔案X.LIB。

```
PIP PRN:=X.ASM[P50]
```

把檔案X.ASM複製到PRN:裝置，每隔50 行有一跳頁動作。每行有一行號，注意，nt8p60 是PRN 檔案的自然參數列，p50改掉原來的自然值 ( default value )。

請讀者注意，PIP 程式本身首先以SAVE 命令複製到一使用

者區域去（如此隨後的檔案才可被複製），以下的運作將 PIP 程式由一使用者區域搬到另一個使用者區域。

USER 0            選定使用者代號 0

DDT PIP.COM 將 PIP 程式載入記憶體  
(注意 PIP 程式的大小 s)

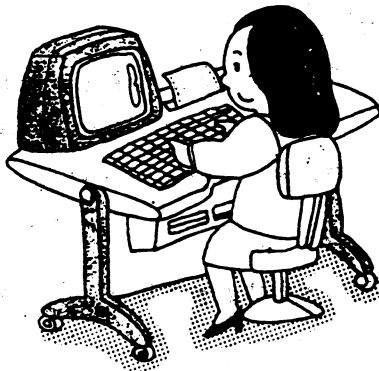
G0                回到 CCP

USER 3            選定使用者代號 3

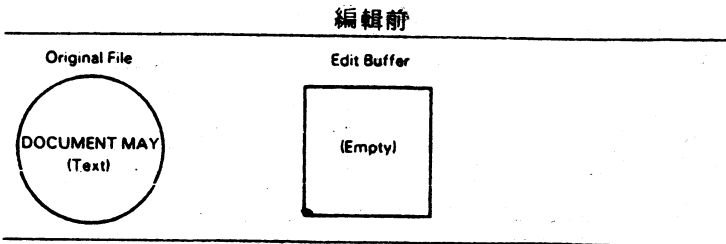
SAVE s PIP.com

其中 s 表 PIP 程式所佔的記憶體頁數，這個數目可在 DDT 配置 PIP.COM 時，參考顯示在“NEXT”下的值。例如，假設下一個可用的位址為 1D00H，那麼 PIP.COM 需要 1C 十六進位的頁數（或  $1 \times 16 + 12 = 28$  頁），因此在後來 SAVE 命令中的 S 為 28。一旦 PIP 程式以這種方式複製後，讀者便可利用標準的 PIP 傳輸，將 PIP 程式複製到屬於同一使用者代號的另一磁碟上。

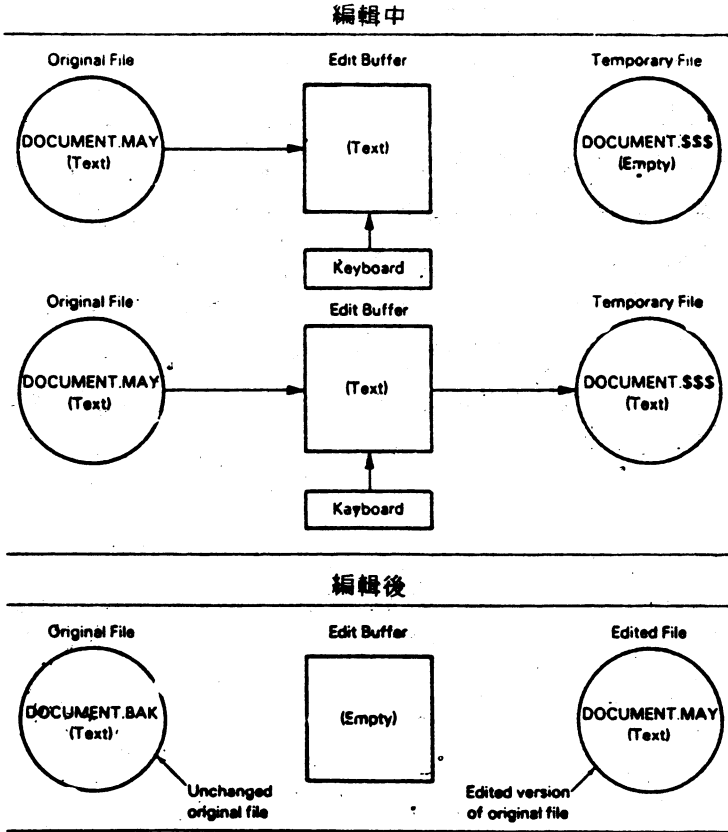
### 編輯程式 (ED)



ED 程式是 CP / M 的系統文稿編輯程式 ( Context Editor ) , 它可用來建立或修改 CP / M 系統的 ASCII 檔案, 詳細的操作過程, 將在本冊第三章 CP / M ED 再作解說。一般說來, ED 可讓操作員建立或操作一串 ASCII 字元組成的原始程式檔案, 每行間以行結尾來隔之。對於行的長度, 並沒有特別的規定, 每行的字元數, 則由相鄰的兩個回車鍵 ( < cr > ) 間, 鍵入的字數決定。ED 程式中有若干命令, 有字元串的找尋、代換及插入, 這些對建立或修改在 CP / M 下的程式或主文檔, 是相當實用的。雖然 CP / M 的記憶工作區空間有限, 在 16K 的 CP / M 系統約為 5000 字元, 但對要編輯的檔案大小却沒有限制, 因為資料經由工作區很容易被編頁 ( Paged ) 。







ED 程式允許原先的檔案與建立的備存檔案分立於兩個不同的磁碟上，這種情形的 ED 命令為

ED ufn d:

其中 ufn 是現所使用磁碟上要修改的檔案，而 d 是另一個磁碟機的代號。ED 程式讀取並處理原先的檔案，然後將新檔案 ufn 寫入磁碟機 d。在處理完畢時，原先的檔案成為備存檔案，如此，若操作員定址

磁碟 A，則命令

ED X.ASM B:

會修改在磁碟機 A 的檔案 X.ASM，並在磁碟機 B 建立新檔案 X. \$\$\$，一但修改工作順利完成，A: X.ASM 易名為 A: X.BAK，而 B: X. \$\$\$ 易名為 B: X.ASM。若在修改開始前，檔案 B: X.ASM 已經存在，控制台上會出現

FILE EXISTS

來防止用戶因疏忽而毀掉檔案。此時，操作員定要先剔除這已存在的檔案，方可重新開始修改的作業。

與其他的程式命令相類似，修改工作可發生在不同於現所指定使用磁碟的磁碟機上，如下例：

ED A: X.ASM 修改磁碟機 A 的檔案 X.ASM，產生的新檔案亦備存在磁碟機 A 上。

ED B: X.ASM A: 修改磁碟機 B 的檔案 X.ASM 後複製到磁碟機 A 的暫時檔案 X. \$\$\$ 上，在修改完成時，磁碟機 B 的 X.ASM 易名為 X.BAK，磁碟機 A 的 X. \$\$\$ 易名為 X.ASM。

在 ED 程式載入記憶體後，若所指名的源檔 ( source file ) 不存在，ED 會建立一個與所指名同名稱的新檔案，且開啓該檔案，以備存取。若源檔存在且須修改，程式設計者可將源檔的資料附加 ( Append 命令 ) 到工作區，這些被附加的資料，可被顯示、修改，而後再從工作區寫回磁碟 ( 參考 W 命令 )。文稿 ( context ) 會自

動編頁並定出程式中的某些特定點（參考 N 命令），使得一個大檔案的特定部份，可以很容易地存取到。

假設操作員鍵入

```
ED X.ASM < cr >
```

ED 程式會建立一個名叫做 X. \$\$\$ 的中間工作檔，來保存在 ED 執行時所修改過的資料。在 ED 程式執行完畢後，原先的檔案 X.ASM 被易名為 X.BAK，而修改過的工作檔易名為 X.ASM，如此，檔案 X.BAK 內含原先未修改的檔，而檔案 X.ASM 則含最近修改過的檔。操作員只要把最後的版本剔除，再把先前的版本易名，就可拿到原先未修改的版本。例如，假設目前的檔案 X.ASM 修改錯誤，下列的 CCP 命令可挽回原先的版本。

```
DIR X.*
```

```
ERA X.ASM
```

```
REN X.ASM=X.BAK
```

注意，操作員可在任何時候中止修改而不會毀掉原先的檔案，在這種情況下，檔案 BAK 是不會建立的，而原先的檔案依然完整。

ED 程式也會斟酌檔案的屬性，若操作員企圖修改一個唯讀狀態的檔案，錯誤訊息。

```
**FILE IS READ/ONLY**
```

會在控制台上出現。這種唯讀檔案只可載入檢查，但絕無法加以修改，遇到這種情形，標準的處理方法是，操作員先終止修改作業，利用 STAT 命令更改檔案的屬性為 R/W 即可。若所要修改的檔案，具

備有系統檔案特性，則訊息

**"SYSTEM" FILE NOT ACCESSIBLE**

會在控制台上顯現，接著修改作業便被中止掉（參看 STAT 命令之實例）。然而，命令 STAT 在必要時亦可用來改變系統的屬性，以便順利執行修改作業。

**SUBMIT 命令 ( SUBMIT Command )**

SUBMIT 命令可將一群結合在一起的 CP/M 命令，一個個拿出來執行。SUBMIT 的格式為：

**SUBMIT ufn parm #1...parm #n(cr).**

其中 ufn 是個檔案型別為 "SUB"，而且是存放在現所指定使用磁碟上的檔案名稱。SUB 檔內可能含有帶有參數的 CP/M 原型命令 ( prototype commands )。實際的參數 parm # 1 ..... parm #n 會取代原型命令中的變數，若沒有錯誤發生，CP/M 會依序處理被取代參數後的命令檔案。

原型命令檔是用 ED 程式建立的，其命令檔中散佈有以下格式的 "\$" 參數：

**\$1 \$2 \$3 ... \$n**

這些參數在檔案被 submit 執行時，會與包含在 SUMMIT 命令中的實際參數相對應，當 SUBMIT 程式執行時，實際參數 parm # 1 ... pram # n 會與原型命令的型式參數 \$1 ..... \$n 配對。

## 實例(1):

```

A>ED TEST.SUB
      : *I
1:   TYPE $1.FOR
2:   REN $2.FOR=$1.FOR
3:   TYPE $2.FOR
4:   :
      : *E

A>SUBMIT TEST AA BB

A>TYPE AA.FOR
      WRITE(5,100)
100  FORMAT(1H , 'ABCDEFGHIJKLMN')
      STOP
      END

A>REN BB.FOR=AA.FOR
A>TYPE BB.FOR
      WRITE(5,100)
100  FORMAT(1H , 'ABCDEFGHIJKLMN')
      STOP
      END

(A>

```

注意，含有底線的命令是使用者鍵入的，其他均系統自動印出。

## 實例(2):

```

A>ED TEST.SUB
      : *I
1:   DIR
2:   TYPE AA.FOR
3:   REN BB.FOR=AA.FOR
4:   ERA AA.FOR
5:   DIR
6:   :
      : *E

A>SUBMIT TEST

A>DIR
A: F80      COM : L80      COM : TEST      BAK : TEST      SUB
A: M80      COM : LIB      COM : CREF80    COM : SUBMIT    COM
A: XSUB     COM : $$$      SUB : AA        FOR : ED        COM
A: FT       FOR : DUMP     COM : STAT      COM : DDT        COM
A: PIP      COM

A>TYPE AA.FOR
      WRITE(5,100)
100  FORMAT(1H , 'ABCDEFGHIJKLMN')
      STOP
      END

```

```

A>REN BB.FOR=AA.FOR
A>ERA AA.FOR
NO FILE
A>DIR
A: F80          COM : L80          COM : TEST          BAK : TEST          SUB
A: M80          COM : LIB          COM : CREF80        COM : SUBMIT         COM
A: XSUB         COM : $$$          SUB : BB            FOR : ED             COM
A: FT           FOR : DUMP         COM : STAT          COM : DDT            COM
A: PIP          COM
A>

```

假若型式參數與實際參數的數目無法相符合，SUBMIT 的程式會中止，並在控制台印出錯誤的訊息。SUBMIT 程式會在現所指定使用的磁碟上建立一個名叫做 \$\$\$ . SUB 的代換命令檔。在系統重新啓動時（SUBMIT 命令的結束輸入），CCP 會讀取代換命令檔，並以之為輸入的來源，而不再經由控制台。若 SUBMIT 程式作業的磁碟，位於磁碟機 A 以外的磁碟機上，則命令會等到磁碟插入磁碟機 A，並重新啓動系統後才執行。此外，使用者可以在命令被讀取而回應（Echo）時鍵入 rubout，以中止命令的執行，此時，檔案 \$\$\$ . SUB 會被刪除，接下來的命令將來自控制台。在 CCP 偵測到任一命令有誤時，命令的處理作業也會中止。當 CP/M 控制下的執行程式發生錯誤，讀者可利用刪除現存的 \$\$\$ . SUB 檔案，以中止命令檔的執行。

因為符號“\$”可存在檔案名稱中，為了避免與型式參數混淆，在 SUB 檔案中，“\$\$”表示符號“\$”。另外，讀者亦可在文字 X 前加上符號“↑”，此可在檔案中產生 ctrl-x 的控制字元。

SUB 檔中的最後一個命令，可用來啓動另一個 SUB 檔，如此一來，便構成了鏈結的整批命令。

假設檔案 ASMBL . SUB 內含以下的原型命令：

```
ASM $1  
DIR $1.*  
ERA *.BAK  
PIP $2:=$1.PRN  
ERA $1.PRN
```

而操作員鍵入以下的命令：

```
SUBMIT ASMBL X PRN <cr>
```

則SUBMIT 程式會讀取檔案ASMBL.SUB，並以X與PRN 分別  
取代檔案中\$1 與\$2 的型式參數，而建立含有以下命令的  
\$\$\$ .SUB 檔案：

```
ASM X  
DIR X.*  
ERA *.BAK  
PIP PRN:=$X.PRN  
ERA X.PRN
```

CCP 會依序處理這些命令。

SUBMIT 程式可以利用在檔案名稱上冠以磁碟機代號，以存取  
位於不同磁碟的SUB 檔。但是只有當磁碟機A的SUB 檔，在  
SUBMIT 時才會生效。因此，若要執行存在磁碟機B中磁碟的SUB  
檔，須先將磁碟機B 的磁碟插入磁碟機A。

### **XSUB命令 (XSUB Command)**

X SUB 擴張了SUBMIT 功能的威力，執行時除了可以送入命  
令行外亦能作字元輸入工作。X SUB 命令位於使用者Submit 檔案  
的第一行，執行時，X SUB 程式會自行配置於CCP 之後。

所有跟隨在XSUB 之後的命令，皆由XSUB 來處理，以致於

所有讀取控制台輸入緩衝區的程式，可直接從 submit 檔中接收他們的輸入。例如，檔案 SAVER.SUB 含有以下的 submit 行：

XSUB	DDT 的 命 令 行 之 輸 入 直 接 自
DDT	
\$1.HEX	Submit 檔 本 身
R	
GO	
SAVE 1 \$2.COM	

結果

SUBMIT SAVER X Y

的 X 和 Y 會分別取代命令行中的 \$1 與 \$2。程式 XSUB 載入記憶體，接著是具有命令行“IX.HEX”“R”及“GO”的 DDT 程式，執行後，將控制權交回給 CCP，最後的命令“SAVE 1 Y.COM”仍然由 CCP 處理。

執行後的 XSUB 程式仍舊留在記憶體中，在每個暖啟動後，控制台會出現“( xsub active )”的訊息，以表示 XSUB 程式仍在。以後的 Submit 命令便不需要 XSUB，除非中途有冷啟動發生。若 XSUB 與 DESPOOL 兩者需同時執行，要注意的是程式 DESPOOL 要先入 SUB 載入記憶體。

實例：有一 Submit 檔如下：



```
A>ED TEST2.SUB
NEW FILE
  : *I
  1: DDT
  2: IABC.HEX
  3: R0
  4: D100
  5: 60
  6: DIR
  7:
  : *E
A>
```

此時執行SUBMIT 命令將有如下之困境：

```
A>SUBMIT TEST2
A>DDT
DDT VERS 2.2
```

當執行到需從 CONSOLE 輸入資料的時候即行停止。

但若把 Submit 檔改成：

```
XSUB
DDT
IABC.HEX
R0
D100
60
DIR
```

多了一XSUB，則執行就順利了。因為DDT 的命令已經可以直接自 Submit 檔本身的下面數行讀取了（上面程式中黑線標出命令行）。

```

A>SUBMIT TEST2

K>XSUB

A>DDT
DDT VERS 2.2
-IGEC.HEX

-R0

NEXT PC
1005.0000
-D100

0100 01 BC 0F C3 E0 13 43 4F 50 59 52 49 47 48 54 20 .....COPYRIGHT
0110 28 43 29 20 31 39 38 30 2C 20 44 49 47 49 54 41 (C) 1980, DIGETA
0120 4C 20 52 45 53 45 41 52 43 48 20 20 20 20 20 L RESEARCH
0130 44 44 54 20 56 45 52 53 20 32 2E 32 24 31 00 02 DDT VERS 2.2#1..
0140 C5 C5 11 30 01 0E 09 CD 05 00 C1 21 07 00 7E 3D ...0.....!..~
0150 90 57 1E 00 D5 21 00 02 78 B1 C0 65 01 0B 7E 12 .M...l..x..e..~
0160 13 23 C3 58 01 D1 C1 E5 62 70 B1 CA 07 01 0B 7B .#.X...bx.....<
0170 E6 07 C2 7A 01 E3 7E 23 E3 6F 7D 17 6F D2 83 01 ...z...#.o).o...
0180 1A 84 12 13 C3 69 01 D1 2E 00 E9 0E 10 CD 05 00 .....i.....
0190 32 5F 1E C9 21 66 1E 70 2B 71 2A 65 1E EB 0E 11 2...!f.p+q+e.....
01A0 CD 05 00 32 5F 1E C9 11 00 00 0E 12 CD 05 00 32 ...2.....2
01B0 5F -1E C9 21 68 1E 70 2B 71 2A 67 1E EB 0E 13 CD ...!h.p+q+g.....
-G0

(xsub active)
A>DIR
A: F80 COM : L80 COM : $$$ SUB : TEST SUB
A: M80 COM : LIB COM : CREF80 COM : SUBMIT COM
A: XSUB COM : ABC ASM : BB FOR : ED COM
A: FT FOR : ASM COM : ABC PRN : ABC HEX
A: DUMP COM : STAT COM : DDT COM : PIP COM
A: TEST2 SUB
A>
    
```

**傾印命令 (DUMP Command)**

DUMP 程式將磁碟檔案 ( ufn ) 以十六進位的格式，傾印在控制台的螢幕上。每次印出十六個 bytes 的檔案內容，在此十六個 bytes 的左側，是以十六進位表示的絕對 bytes 資料位址。讀者可在傾印時，按下 rubout 鍵，以中止傾印作業。( 在 CP/M 界面導引那一章中，有個 DUMP 程式的源程式 ( Source Listing )，它是為 CP/M 設施所寫的程式範例。)

## 172 Apple II 軟體卡

### 實例(1):

```
A>DUMP TEST.ASM
```

```
0000 3B 09 46 49 4C 45 20 44 55 4D 50 20 50 52 4F 47
0010 52 41 4D 2C 20 52 45 41 44 53 20 41 4E 20 49 4E
0020 50 55 54 20 46 49 4C 45 20 41 4E 44 20 50 52 49
0030 4E '54 53 20 49 4E 20 48 45 58 0D 0A 3B 0D 0A 3B
```

```
A>
```

### 實例(2):

```
A>DUMP B:DUMP.COM
```

```
0000 21 00 00 39 22 15 02 31 57 02 CD C1 01 FE FF C2
0010 1B 01 11 F3 01 CD 9C 01 C3 51 01 3E 80 32 13 02
0020 21 00 00 E5 CD A2 01 E1 DA 51 01 47 7D E6 0F C2
```

```
A>
```

### 實例(3):

```
A>DUMP AAAA.ASM
```

```
NO INPUT FILE PRESENT ON DISK
```

```
A>
```

## 磁碟作業系統的錯誤訊息 ( Bdos Error Message )

在檔案處理時，磁碟作業系統會因下面的三種錯誤情況，而遭到截斷。只要其中一種情況被查出，BDOS 就會印出如下的訊息：

```
BDOS ERR ON x: error
```

其中 x 是磁碟機代號，" error " 是三種錯誤訊息之一：

**BAD SECTOR  
SELECT  
R/O**

訊息“BAD SECTOR”表示在讀寫磁碟片，磁碟控制板的電子設備查出了有錯誤的情況，一般說來，這種錯誤情況是起因於磁碟控制板上元件發生故障或是磁碟磨損過度。假若讀者的系統，每月有以上的一次以上的這種錯誤時，讀者應該檢查控制板電子線路及媒介物的情況。另外，若控制板是由不同的廠商所製造，在檔案讀取時，也有可能遇到這種錯誤。在碰到這種錯誤，讀者可鍵入 `ctl-C` 重新啓動 CP/M（這是最保險），或是按入 Return 鍵，則會跳過檔案作業時壞的磁碟段落（Bad Sector）。注意，若磁碟正在做寫入索引的動作，按下 Return 鍵，可能會毀掉磁碟的完整性。所以在這種情況的處理，讀者應先確定是否有該磁碟的備存副片。

訊息“SELECT”發生在讀者企圖定址一個超出 A 到 D 範圍的磁碟機，在這情況下，錯誤訊息的 x 代表讀者所選定的磁碟機。任何從控制台的輸入，皆可使系統再重新啓動。

任何企圖將資料寫入一個已被 STAT 命令指定或 BDOS 設定為唯讀狀態的磁碟，會產生 R/O 的訊息。一般說來，操作員應該使用暖啓動 `ctl-c` 或在更換磁碟後使用冷啓動來重新啓動 CP/M 系統。若所換的磁碟只能讀取不能寫入，則不用暖啓動或冷啓動時該磁碟機會被標記為唯讀狀態。若對它使用暖啓動或冷啓動，則會使該磁碟機轉換為可讀可寫狀態。當 R/O 訊息出現後，CP/M 會等待控制台的輸入，任何輸入皆可引起暖啓動，重新啓動系統。



## 第 二 章

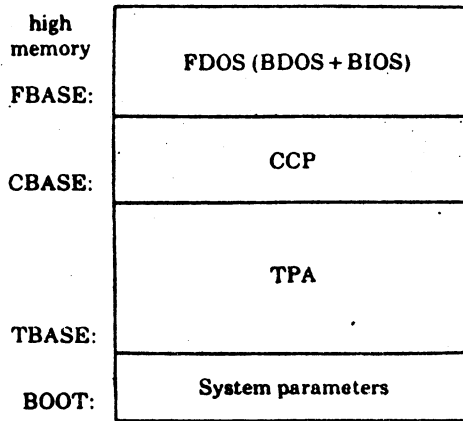
# CP/M 2.0 的介面導引

- 簡介
- 作業系統的呼叫協定
- 檔案對檔案的複製程式範例
- 檔案傾印的公用程式範例
- 隨機存取的程式範例
- 系統功能摘要

## 簡介

本章敘述 CP/M 第二版本的系統組織，其中包括記憶體的结构及系統的進入點 ( Entry Point )，目的在提供編寫使用 CP/M 系統週邊設備及磁碟輸出輸入設備程式時，所需要的訊息。

CP/M 作業系統邏輯上共區分為四大部分：基本輸出輸入系統 ( Basic I/O System ) 簡稱 BIOS、磁碟作業系統 ( Basic Disk Operating System ) 簡稱 BDOS、控制台命令處理程式 ( Console Command Processor ) 簡稱 CCP，和暫存程式區 ( Transient Program Area ) 簡稱 TPA。BIOS 是個和硬體有關 ( Hardware-Dependent ) 的模組 ( Module )，它定義了電腦系統與週邊設備輸出輸入間的低階界面 ( Low Level Interface )。BIOS 及 BDOS 在邏輯上組成具有共同進入點 ( Common Entry Point ) 的 FDOS。CCP 是個使用 FDOS 的程式，它提供人們與編目儲存在備存記憶裝置上的訊息之間的界面。TPA 則是記憶體的一部份，各種非作業系統的命令 ( Command ) 與使用者 ( User ) 程式皆在此執行。記憶體低位址的部份，是保留給系統使用的，詳細的內容在後面的章節中將會提到，以下是 CP/M 系統的記憶體結構：



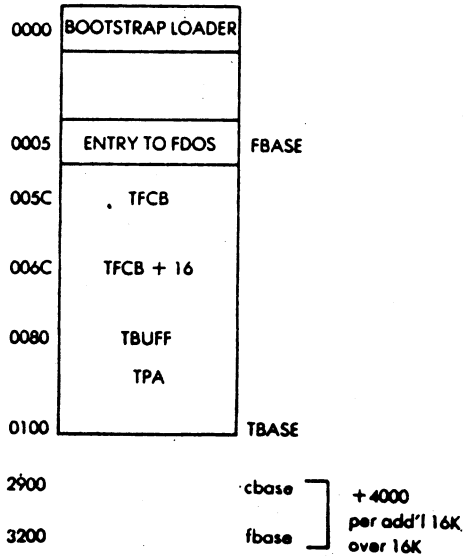
兩種使用 CP/M 系統的 Apple 記憶體結構之基準位址 ( Base Address ) 如下表：

模組	44 K	56 K ( 帶有語言板 )
CCP	9400H	C400H
BDOS	9C00H	CC00H
BIOS	AA00H	DA00H
Top of RAM	AFFFH	DFFFH

所有標準的 CP/M 版本皆設定 **BOOT** 的位址為 **0000H**，這也是隨機存取記憶體 ( RAM ) 的基準位址。在位址 **BOOT** 中所儲存的，是個執行系統暖啟動 ( Warm Start ) 的機器碼，它用來載入及起始把控制權交回給 **CCP** 所需的程式與變數。因此，暫態程式只需跳至位址 **BOOT**，即可把控制權交回給 CP/M。另外，**TBASE = BOOT + 0100H**，亦即位址 **0100H**。而 **FDOS** 的進入點則在位址 **BOOT + 0005H**，即位址 **0005H**。在那裏有個跳至 **FBASE** 的跳越指令。而 **BOOT + 0006H** 是含有 **FBASE** 值的位址欄 ( Address



Field)。FBASE 值可用來決定可以使用的記憶體大小(如果 CCP 被暫態程式所重疊)。



CP / M 記憶體配置略圖

( \* 注意本圖位址 0000 在上端 )

暫態程式經由命令行 ( Command Line ) 的呼叫，載入 TPA 中執行，操作員在每個 “>” 符號後鍵入命令行，與 CCP 連繫。每個命令行可為下列格式之一種：

- 命令 ( command )
- 命令 檔案 1 ( command file 1 )
- 命令 檔案 1 檔案 2 ( command file 1 file 2 )

其中命令可為系統內建命令，諸如 DIR 或 TYPE，也可為暫態命令或程式的名稱。假如命令是個 CP / M 的系統內建命令，它可立即執

行。否則，CCP 會在現所指定使用磁碟中尋找一個叫 command.COM 的檔，如果檔案存在，其必是個要在 TPA 執行的程式記憶映像 ( Program Memory Image )，亦即程式的起始位址必須在記憶體 TBASE，於是 CCP 將這個 COM 檔案，從磁碟中載入記憶體。程式的終止位址可以跨越過 CBASE 位址。

若命令中含有一個或二個檔案，CCP 會在系統參數區中，準備一個或二個檔案控制區 ( File Control Block )，這些可取捨的檔案控制區，在系統透過 FDOS 存取檔案時，可提供某些參考訊息，詳情在下節討論。

暫態程式從 CCP 中取得控制權後即開始執行，執行中可能會用到 FDOS 的 I/O 設備。暫態程式因 CCP 的呼叫而執行，因此在程式執行完畢後，可單純地回返到 CCP，或者可跳至 BOOT 處，把控制權交給 CP/M。前者暫態程式所佔用的記憶體位址，不可超越 CBASE，而後者只要在位址 FBASE-1 以下就沒有問題了。

暫態程式可利用 CP/M 的 I/O 功能，與操作員的控制台和包括磁碟在內的週邊設備裝置作連繫。而這些 I/O 系統的取得，是經由在位址 BOOT+0005H 的 FDOS 進入點，傳送功能代碼與訊息位址給 CP/M。例如，當使用者需要執行一個磁碟讀取的動作時，可在暫態程式送出一個對應磁碟讀取的功能代碼及一個 FCB 的位址給 CP/M 的 FDOS，FDOS 就會執行這個動作，並送回讀取完成的信號，或讀取失敗的錯誤碼。功能代碼與錯誤信號在下節再詳述。

## 作業系統的呼叫協定

本節的目的在提供使用者程式呼叫作業系統時所需的訊息。

CP/M 可供給暫態程式的系統功能，可以分成兩類：①簡單裝置的

輸出輸入，與②磁碟檔的輸出輸入。簡單裝置的 I/O 動作包括：

- 從控制台讀取一個字元 ( Read a Console Character )
- 至控制台寫出一個字元 ( Write a Console Character )
- 從磁帶讀取一個字元 ( Read a Sequential Tape Character )
- 至磁帶寫出一個字元 ( Write a Sequential Tape Character )
- 至列表機印出一個字元 ( Write a List Device Character )
- 取得或設定 I/O 狀態 ( Get or Set I/O Status )
- 寫入控制台緩衝區 ( Print Console Buffer )
- 讀取控制台緩衝區 ( Read Console Buffer )
- 查詢控制台完備狀態 ( Interrogate Console Ready )

FDOS 的磁碟 I/O 動作包括：

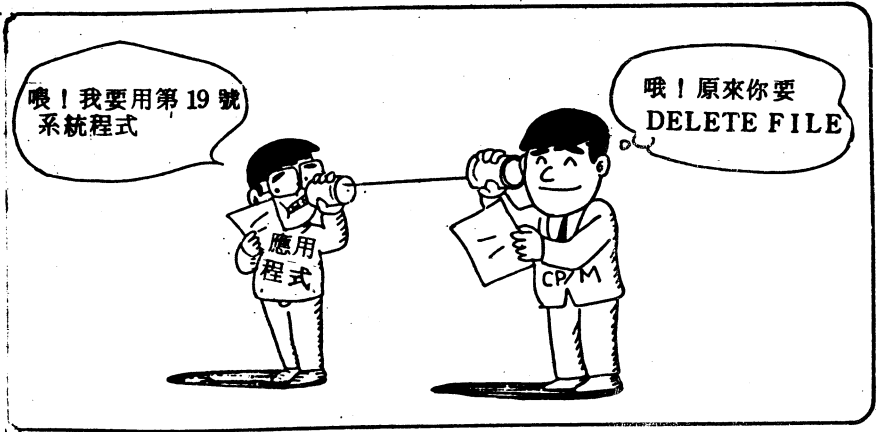
- 磁碟系統重置 ( Disk System Reset )
- 磁碟機選擇 ( Drive Selection )
- 檔案建立 ( File Creation )
- 檔案開啓 ( File Open )
- 檔案關閉 ( File Close )
- 索引目錄找尋 ( Directory Search )
- 檔案易名 ( File Rename )
- 隨機或循序讀取 ( Random or Sequential Read )
- 隨機或循序寫入 ( Random or Sequential Write )
- 檔案刪除 ( File Delete )
- 查詢可用磁碟 ( Interrogate Available Disk )
- 查詢選取磁碟 ( Interrogate Selected Disk )
- 設定 DMA 位址 ( Set DMA Address )

## 設定或重置檔案指示器 ( Set/Reset File Indicator )

如前面所敘述的，FDOS 功能的取得是經由傳送功能代碼及訊息位址，以跳入位址 BOOT+0005H 而完成的。通常，功能代碼存於暫存器 C，訊息位址存於暫存器對 DE，當程式回返 ( Return ) 時，單拜元組的資料存於暫存器 A，雙拜元組則存於暫存器對 HL ( 若 HL 的值為零時表示功能代碼超出範圍)。為了相容緣故，程式回返時，暫存器 A = L 且暫存器 B = H。另外 CP/M 暫存器的傳送協定與 Intel PL/M 系統程式語言的相一致。CP/M 的功能代碼如下：

0	系統重置	19	刪除檔案
1	控制台輸入	20	讀取循序檔
2	控制台輸出	21	寫入循序檔
3	閱讀機輸入	22	建立空白檔 ( Make File )
4	打孔機輸出	23	更改檔名
5	列表機輸出	24	送回現行磁碟指定向量 ( Vector )
6	直接控制台 I / O	25	送回現行磁碟機代號
7	取得 I / O 狀態	26	設定 DMA 位址
8	設定 I / O 狀態	27	取得配置位址
9	列印字串	28	寫入保護磁碟
10	讀取控制台緩衝區	29	取得唯讀向量 ( R / O Vector )
11	取得控制台狀態	30	設定檔案屬性 ( Attributes )
12	送回系統版號	31	取得磁碟參數位址
13	重置磁碟系統	32	設定或取得使用者代號
14	選擇磁碟機	33	讀取隨機檔
15	開啓檔案	34	寫入隨機檔

- 16 關閉檔案
- 17 找尋第一項
- 18 找尋下一項
- 35 計算檔案大小
- 36 設定隨機資料記錄數

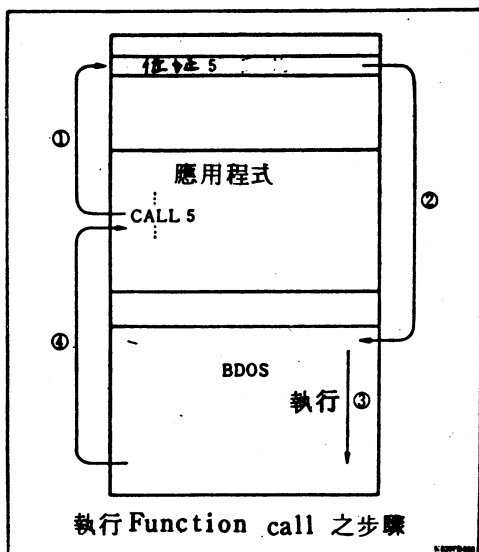


爲了維持與MP/M 有更高的相容性，在應用程式裏應該盡量避免使用功能 28 與功能 32。(MP/M 爲多使用者之 CP/M 作業系統)。

在進入暫態程式時，CCP 會將堆疊 ( Stack ) 指標設定在有八層的堆疊區域，並把 CCP 的回返位址推入堆疊上，如此在堆疊溢位前仍留有七層可使用。暫態程式通常不用這些堆疊，因爲大部分的暫態程式，都經由位址 0000H 回返到 CCP 。而在系統進入時，FDOS 會轉換到區域性的堆疊，因此，上述的堆疊對 CP/M 系統呼叫是足夠大的。例如下面的組合語言程式段，不斷地從控制台讀取字元，直至讀取的字元是個星號 “ \* ” 爲止，屆時再將控制權交回給 CCP ( 假設程式是在具有 BOOT+0000H 的標準 CP/M 系統上執行 ) 。

```

BDOS      EQU      0005H      ;STANDARD CP/M ENTRY
CONIN     EQU      1          ;CONSOLE INPUT FUNCTION
;
;
NEXTC:    ORG      0100H      ;BASE OF TPA
          MVI      C,CONIN    ;READ NEXT CHARACTER
          CALL     BDOS       ;RETURN CHARACTER IN (A)
          CPI      '*'        ;END OF PROCESSING?
          JNZ     NEXTC      ;LOOP IF NOT
          RET      ;RETURN TO CCP
          END
    
```



CP/M 在每個磁碟中擁有具名檔案的結構，這種結構使得任何特定的檔案，可含有零到充滿整個磁碟容量的任何數量資料錄。每個磁碟邏輯上可分為磁碟索引及檔案資料區，而磁碟檔案名稱也分成三個部份：一為磁碟機代號，二為含一到八個字元的檔案名稱，三為含一到三個字元的檔案型別。檔案型別一般用來指出檔案的屬性類別，檔案名稱則用來區別同類檔案的個別檔案。下面所列出的是幾種不同的通用類別：

ASM	組合編譯源程式	PLI	PL / I 源程式
PRN	印表機的列表	REL	可重定位的模組
HEX	十六進位的機器碼	TEX	TEX 格式化之源程式
BAS	BASIC 語言源程式	BAK	ED 源程式備存檔
INT	中間碼	SYM	SID 符號檔
COM	CCP 命令檔案	\$\$\$	暫存檔

源檔是由一串 ASCII 文字所組成的，且在每行後面跟隨著回車鍵 ( 0DH ) 及饋行鍵 ( 0AH ) 的控制碼。因此每一長為 128 bytes 的資料錄，可含有數行的原始文稿。ASCII 檔案以 control Z 或真正的結尾來表示檔案的結束，但對機器碼檔案 ( 如 COM 檔案 ) control Z 並不代表檔案的結束。

在 CP / M 中的檔案可視為一串長 65536 個資料錄，每個含有 128 個拜，每個資料錄以 0 到 65535 編號。如此一個檔案的最大容量為八百萬個拜。資料錄間的關係，在邏輯上可視為相鄰連續的，但若以實體的觀點言，他們却是間斷不連續的。在磁碟檔的內部處理上，所有的檔案，皆被拆解成長 16 K 拜的資料段 ( Segment )，我們稱它為邏輯延伸區 ( Logical Extent )，這樣一來計數器就容易保持 8 個比次 ( bit ) 的值。雖然在此分析延伸區的拆解，這分析對程式設計人員並沒有什麼特殊的影響，因為不論在循序或隨機存取作業中，邏輯延伸區都可自動存取到，不用程式設計人員操心。

在以功能代碼 15 ( 開啓檔案 ) 為開頭的所有檔案操作中，暫存器對 DE 經常用來存放檔案控制區 ( FCB ) 的位址。暫態程式常用由 CP / M 保留而位址在 BOOT + 005 CH 的 FCB，來作單一的檔案操作。所有檔案操作皆以一 128 拜的資料錄為基本單位，由 CP / M 提供 I / O 緩衝區的磁碟位於 BOOT + 0080 H 的地方，這也就是內設

DMA 的位址。除了找尋第一項與找尋下一項外，所有的索引操作都在不影響寫入緩衝區的保留區域下運作，這情形仍與版本 1 的相同。

FCB 的資料區是由 33 個 (循序存取) 或 36 個 (隨機存取) 拜所組成，內設起始位址為 005CH。對隨機存取檔案言，多出的三個拜位於 BOOT+007CH 處。FCB 的格式如下：

dr	f1	f2	/	/	f8	t1	t2	t3	ex	s1	s2	rc	d0	/	/	dn	cr	r0	r1	r2
00	01	02			08	09	10	11	12	13	14	15	16			31	32	33	34	35

其中

- dr            磁碟機代號 ( 0 - 16 )  
 0 ⇒ 使用磁碟機之自然值 ( default value )  
 1 ⇒ 磁碟機 A  
 2 ⇒ 磁碟機 B  
 ⋮  
 16 ⇒ 磁碟機 P
- f1...f8      內含 ASCII 大寫的檔案名稱，高比次 ( High bit ) 設定為 0
- t1, t2, t3   內含 ASCII 大寫的檔案型別，高比次設定為 0，  
 若以 t1', t2', t3' 分別表該拜的最高比次，則  
 t1' = 1 ⇒ 唯讀檔案  
 t2' = 1 ⇒ SYS 檔案，不 DIR 列表
- ex           內含目前的延伸區數，一般被使用者設定為 00，  
 在檔案 I/O 作業時，其值介於 0 至 31 之間



- s1 保留給內部系統用
- s2 保留給內部系統用，在呼叫 OPEN, MAKE 及 SEARCH 程式時設定為 0
- rc 每個延伸內含資料錄數，其值介於 0 - 128
- d0...dn 保留給系統用，其值由 CP/M 填入
- cr 內含目前要讀取或寫入循序檔的資料錄號數
- r0, r1, r2 內含隨機檔的資料錄數。在隨機檔案作業時，r2 值為零表溢位，否則 r0, r1 組成一個十六比次的值，表目前正要讀取或寫入的資料錄號數，其大小介於 0 到 65535 之間（高拜 r1，低拜 r0）。

每個檔案在 CP/M 存取之前，必須要有一個對應的 FCB，這個 FCB 提供了以後檔案操作所需的檔案名稱及配置訊息。當存取檔案時，程式設計員要負責填上 FCB 前十六個拜及 cr 欄位的值，通常拜 1 到 11 設定為 ASCII 碼，以定檔案的名稱與型別，其他的欄位則設為 0。

FCB 儲存在磁碟的索引區，在進行檔案作業之前會被帶入主記憶體（參考 OPEN 與 MAKE 功能）。檔案作業發生時，記憶體的 FCB 內容或許會被更新，因此檔案作業結束後，FCB 須從記憶體再寫回磁碟的檔案目錄（參考 CLOSE 命令）。

CCP 利用審查跟隨在暫態程式命令後的剩餘部份，為程式命令建立一個或二個 FCB，第一個 FCB 建立在位址為 BOOT+005CH 的地方，第二個 FCB 則佔有第一個 FCB 的 d0...dn 部份，因此在檔案作業前，必須先將第二個 FCB 移到另一區域。例如

，操作者鍵入程式命令

```
PROGNAME B:X.ZOT YZAP
```

檔案PROGNAME.COM 載入TPA，位於BOOT+005CH的FCB的磁碟機代號自然值為2，檔案名稱為“X”，而檔案型別為“ZOT”。第二個磁碟機代號為自然值0，存放在BOOT+006CH處，檔案名稱“Y”放在BOOT+006DH，檔案型別“ZAP”則放在BOOT+0075H的地方，其餘的欄位設定為0。再次提醒讀者，將第二個檔案名稱及型別移到其他區域，是程式設計者的責任，因為檔案的開啓動作會蓋掉第二個檔案的名稱及型別。

假使當初的暫態程式命令沒有指定檔案名稱，以BOOT+005DH及BOOT+006DH起始位址的欄位會填以空白。在任何情況下，CCP會將小寫字母轉換成大寫字母以符合CP/M規定。在操作員鍵入暫態程式命令後，暫態程式命令的尾部（即除命令名稱以外的部份），會存放在位址BOOT+0080H的系統內設緩衝區上，其中第一個拜表存放的字數。以上述的例子來說，BOOT+0080H位址起將存放如下的內容：

BOOT+0080H:

```
+00 +01 +02 +03 +04 +05 +06 +07 +08 +09 +10 +11 +12 +13 +14
14 " " "B" "." "X" "." "Z" "O" "T" " " "Y" "." "Z" "A" "P"
```

下面我們要詳細討論各個功能代碼的作用。

### 功能 0：系統重置

入口參數：( entry parameters )

暫存器 C：00H

系統重置的功能，在把控制權交還給CP/M作業系統的CCP。CCP選擇並指定使用磁碟機A，以重新啓動磁碟副系統。這功能與

跳越到 BOOT 位址有同樣的效果。

## 功能 1：控制台輸入

入口參數：

暫存器 C：01H

返回值：

暫存器 A：ASCII 字碼

控制台輸入功能，在把從控制台讀取的字元送給暫存器 A。圖型字元，回車鍵、換行及倒退鍵 (ctl-H) 會在控制台產生回應；定位符號 (tab) (ctl-I) 會擴張到八個字元；對於 ctl-S 和 ctl-P 則會作字串檢查，以確定是起始的還是終止的螢光幕不斷印出 (scroll) 或變回應 (echo)。輸入字元未從控制台鍵入前，FDOS 不會將控制權交還給呼叫程式，如此將抑制程式的執行，等待輸入字元。

## 功能 2：控制台輸出

入口參數：

暫存器 C：02H

暫存器 E：ASCII 字元

這功能將暫存器 E 的 ASCII 字碼，送至控制台。與功能 1 類似，若暫存器 E 所儲的是標籤字元，則會擴張，若為 ctl-S 及 ctl-P 則會作字串檢查。

## 功能 3：閱讀機輸入

入口參數：

暫存器 C : 03H

回返值：

暫存器 A : ASCII 字元

這功能將從邏輯閱讀機讀取到的字元，送到暫存器A。同樣地，在未讀到字元之前，控制權是不會交還給呼叫程式的。

#### 功能 4：打卡機輸出

入口參數：

暫存器 C : 04H

暫存器 E : ASCII 字元

此功能把存放在暫存器E的字元，送到邏輯打孔機。

#### 功能 5：列表機輸出

入口參數：

暫存器 C : 05H

暫存器 E : ASCII 字元

這功能把存放在暫存器E的字元，送到邏輯列表機。

#### 功能 6：直接控制台I/O

入口參數：

暫存器 C : 06H

暫存器 E : 0FFH (輸入) 或字元 (輸出)

回返回值：

暫存器 A : 字元或狀態(無值)

此功能是另一種控制台輸入輸出的方式。一般程式應盡量避免使用，因為這功能會忽略 CP/M 所有標準控制字元的效能（例如，ctl-S 及 ctl-P）。此功能在 CP/M 的以前版本中，是由 BIOS 來完成，但在 CP/M 2.0 以後的版本，則由 BDOS 來控制，此點值得讀者注意。

功能 6 的入口參數暫存器 E 之內容，若為 FFH，表示控制台要求輸入字元，若為其他字元則表示控制台輸出一個字元。若 E 輸入值為 FFH，且沒有字元輸入，回返回值暫存器 A 的值為 00H，否則暫存器 A 含輸入的字元。若暫存器 E 的值不為 FFH，則其值為一即將送到控制台的 ASCII 字元。

## 功能 7：取得 I/O 拜

入口參數：

暫存器 C : 07H

回返回值：

暫存器 A : I/O 拜的值

此功能在將 I/O 拜存放到暫存器 A。

## 功能 8：設定 I/O 拜的值

入口參數：

暫存器 C : 08H

暫存器 E : I/O 拜的值

此功能以暫存器 E 的內容，取代 IOBYTE 的值。

## 功能 9 : 印字串

入口參數：

暫存器 C : 09H

暫存器對 DE : 字串位址

此功能將從存放在暫存器對 DE : 字串位址抓取字串，逐字送到控制台印出，直至碰到“\$”符號為止。定位符號、ctl-S 與 ctl-P 等控制字元，像功能 2 所提一樣會擴張或被檢查。

## 功能 10 : 讀取控制台緩衝區

入口參數：

暫存器 C : 0AH

暫存器對 DE : 緩衝區地址

回返回值：

緩衝區內的控制台字元串

這功能把控制台上經過編輯的輸入行，逐字讀入以暫存器對 DE 為起始位址的緩衝區，直到緩衝區溢位為止。緩衝區的格式如下：

DE: +0 +1 +2 +3 +4 +5 +6 +7 +8 ... +n

mx	nc	c1	c2	c3	c4	c5	c6	c7	...	??
----	----	----	----	----	----	----	----	----	-----	----

“mx”表示緩衝區所能容納的字元數（1到255），“nc”表示真正讀入的字元數（由FDOS在回返時設定），跟隨在“nc”之後的，便是從控制台所讀取的字元。編輯時，可使用的控制字元有：

rub / del	刪除最後鍵入的字元
ctl-C	在一行開端時，表重新啓動CP/M
ctl-E	引起實質的行結束
ctl-H	倒退一個字元位置
ctl-J	終止輸入行（饋行）
ctl-M	終止輸入行（回車）
ctl-R	在新行中重印出本行修正後的內容
ctl-X	倒退到本行的起頭

以上的控制字元會使編輯工作更加便利。

## 功能11：取得控制台的狀態

入口參數：

暫存器 C：0BH

回返回值：

暫存器 A：控制台狀態

此功能在檢查是否已有字元在控制台鍵入。若有則暫存器A值為0FH，否則為00H。

## 功能12：送回系統版號

入口參數：

暫存器 C : 0CH

回返值：

暫存器對 HL : 版本號碼

功能 12 提供與版本無關的程式設計的訊息。暫存器 H = 00 表示使用的版本為 CP/M，H = 01 則為 MP/M 版本。暫存器 L = 00 代表 2.0 以前的版本，L = 20H 為 CP/M 2.0 版本，L = 21H, 22H, …… 2FH 則各代表 2.0 以後的版本。如此，讀者可設計同時能處理循序及隨機存取的應用程式，再利用本功能所得的版號，機動地變化程式的處理能力。

### 功能 13：重置磁碟系統

入口參數：

暫存器 C : 0DH

此功能使所有磁碟皆設定為讀/寫狀態（參考功能 28 及功能 29），並選擇現行磁碟機 A，而 DMA 位址自然值也重置為 BOOT+0080H。這功能使得讀者不需重新啓動系統，就可更換磁碟。

### 功能 14：選擇磁碟機

入口參數：

暫存器 C : 0EH

暫存器 H : 要選的磁碟機代號

此功能指定存放在暫存器 E 的磁碟機代號，為此後檔案作業的磁碟機自然值。暫存器 E = 0 表示磁碟機 A，1 表磁碟機 B，以此類推



，到 15 表磁碟機 P。被選到的磁碟機會設為佔線狀態，直到下個冷啓動、暖啓動或磁碟系統重置為止。若在連線中更換磁碟，會使該磁碟機自動進入唯讀狀態。

### 功能 15：開啓檔案

入口參數：

暫存器 C：0FH

暫存器 DE：FCB 位址

回返回值：

暫存器 A：索引代碼 ( Directory code )

此功能為現正使用系統的使用者代號啓動存在磁碟索引的檔案。FOOS 會在磁碟索引找尋與 DE 所指的 FCB ( S 1 自動設為零 ) 內 1 到 14 拜相符合的 FCB，若有問號 ( 3FH ) 夾在其中，可視其符合任意字元。一般而言其中不含問號，且 “ ex ” 與 “ s2 ” 均被設定為 0。若索引中有相合的檔案，相關的索引訊息會被抄錄到 FCB 的 do 到 dn 拜上。這樣，此後的讀寫動作，就會作用在該檔案上。注意，要存取檔案前，一定要開啓該檔案。檔案開始後會送回索引碼，其值若為 0 到 3 表檔案開啓成功，若為 0FFH 表檔案沒找到。找尋索引時，若 FCB 的檔案名稱有問號，第一個符合的檔案會被選上。在循序檔案作業時，若要由第一個資料錄開始讀，程式上需把 “ cr ” 設定為 0。

### 功能 16：關閉檔案

入口參數：

暫存器 C : 10H

暫存器對 DE : FCB 位址

回返回值 :

暫存器 A : 索引碼

關閉檔案所履行的功能恰與開啓檔案相反，它把開啓檔案或建空檔案所產生新的 FCB 抄回磁碟的索引裏。其餘的規定與開啓檔案相同，暫存器 A 的值 0 到 3 表關閉成功，0 FFH 表檔案找不著。若對檔案只做讀取資料，可不用做檔案關閉的工作，但若有寫入的動作，就必須有關閉的動作，把異動過的 FCB，抄錄到磁碟的目錄上。

### 功能 17：找尋第一項

入口參數：

暫存器 C : 11H

暫存器對 DE : FCB 位址

回返回值：

暫存器 A : 索引碼

此功能在磁碟索引區中，找出一個與 FCB 相合的檔案。暫存器 A 的回返回值，若為 FFH 表檔案未找到，0, 1, 2, 3 則代表檔案存在。若檔案存在，該檔案在索引區的入口資料錄會存入 DMA 位址中，其相對位址為暫存器 A 的值乘 32。這種作業對一般的應用程式雖沒什麼用處，但却可由 DMA 的緩衝區中，取得相關的索引資料。FCB 的“dr”欄位若為“？”，表示自動磁碟選擇功能被取消，而只找尋自然值定的磁碟內部，找尋功能會回輸任一符合的索引入口

論它是已配置或仍未配置，不論它是屬於那一使用者。FCB 的“fl”到“ex”若有“？”在其中，此“？”符合任意字元。若“dr”欄非“？”、“S<sub>2</sub>”拜自動設為零。

### 功能18：找尋下一項（找尋下一個符合的檔）

入口參數：

暫存器 C：12H

回返回值：

暫存器 A：索引碼

這個功能與功能17相似，只不過索引的找尋起點是從上一個相合的入口項目後開始。同樣地，若在索引中找不到相同的檔案，暫存器A的回返回值就為255。

### 功能19：刪除檔案

入口參數：

暫存器 C：13H

暫存器對 DE：FCB位址

回返回值：

暫存器 A：索引碼

此功能在刪除所有與FCB相符合的檔案。檔案名可為夾有問號的群體檔案名，但磁碟機代號却必須明確。暫存器A的回返回值，若為FFH，表示要刪除的檔案不在磁碟上，否則為0到3，表示檔案已經刪除掉。

### 功能20：讀取循序檔

入口參數：

暫存器 C : 14H

暫存器對 DE : FCB 位址

回返回值：

暫存器 A : 索引碼

經過檔案開啓或建空檔的程序後，讀取循序檔功能會從檔案中，讀取一筆長為 128 個拜的資料錄，並將其存於現今 DMA 位址的記憶體中，至於要讀取那一筆資料錄，則由 FCB 的“cr”與“ex”兩欄位的內容決定。每讀取一筆資料錄，“cr”值加一，當“cr”值溢位時，下個邏輯延伸區會自動開啓，且把“cr”的值重設為 0。倘讀取作業成功，暫存器 A 的索引碼為 0，若下個資料錄位置沒有資料（即檔案結束），索引碼為一不等於 0 的值。

## 功能 21：寫入循序檔

入口參數：

暫存器 C : 15 H

暫存器對 DE : FCB 位址

回返回值：

暫存器 A : 索引碼

經過檔案開啓或建空檔的程序後，寫入循序檔功能會把 DMA 位址起的 128 個拜資料，寫入 FCB 所指名的檔案。資料錄號數擺在檔案的“cr”位置上，寫入之後，“cr”值加一，若溢位則下個邏輯延伸區會自動開啓，並把“cr”值重設為 0。寫入作業若發生在已存在的檔案，新產生的資料錄會蓋掉原先的資料。暫存器 A 的索引

碼若為 0 表示寫入成功，若不為 0 則表示磁碟已滿。

## 功能22：建立空白檔

入口參數：

暫存器 C：16H

暫存器對 DE：FCB位址

回返回值：

暫存器 A：目錄碼

建立空白檔與開啓檔案類似，唯一不同點是 FCB 所指名的檔案名稱，不可以在現所指定的磁碟中。運作時，FDOS 會建立這個檔案，並在索引區與記憶體上，設定此檔案為空檔。程式設計者在使用此功能前，必須先確定磁碟中沒有與 FCB 相同的檔案名稱，否則要先刪除掉。程式回返時，若建檔成功，暫存器 A 的值介於 0 到 3 之間，若索引區已無空間可使用，則值為 0FFH。建空白檔功能使得啓動 FCBI，其後就可省掉開檔手續。

## 功能23：更改檔名

入口參數：

暫存器 C：17H

暫存器對 DE：FCB位址

回返回值：

暫存器 A：索引碼

此功能在更改磁碟索引區的檔名，使用 DE 所指的 FCB 改掉前

十六個拜中的檔名成爲第二個 16 拜中的檔名。FCB中位置0的磁碟機代號用來選擇磁碟機，而新檔名的磁碟機代號，視爲0。檔名若更改成功，暫存器A的值界於0到3之間。若原來的檔名在目錄中找不到，暫存器A的值爲0FFH。

### 功能24：送回現行磁碟指定向量

入口參數：

暫存器 C：18H

回返回值：

暫存器對HL：現使用磁碟指定向量

CP/M送回的現使用磁碟指定向量，存放在暫存器對HL，其中暫存器L的最小比次對應第一個磁碟機A，暫存器H的最高比次對應第十六磁碟機P。比次值“0”表示該磁碟機在非佔線狀態，比次值“1”表該磁碟機在佔線狀態。因爲在程式回返時，暫存器A與L的值相同，與早期系統版本具有相容性。

### 功能25：送回現行磁碟機代號

入口參數：

暫存器 C：19H

回返回值：

暫存器 A：現行磁碟機代號

此功能將內設磁碟機的代號，存放在暫存器A，代號的範圍爲0

到 15 之間，分別代表磁碟機 A 到 P。

## 功能26：設定DMA位址

入口參數：

暫存器 C：1AH

暫存器對DE：DMA位址

DMA 是直接記憶存取 ( Direct Memory Access ) 的簡寫，它與磁碟控制板 ( Disk Controller ) 連接，而磁碟控制板則用來存取主記憶體，以傳送資料給磁碟副系統，或由磁碟副系統取得資料，存放於主記憶體。在讀取之後或寫入磁碟之前，資料錄是存放在 DMA 位址起的 128 個拜內。DMA 的位址在冷啟動、暖啟動或磁碟系統重置時，會自動設定在 BOOT+0080H。設定 DMA 位址的功能，就是在改變這個位址，新的位址存於暫存器對 DE，設定之後，要待到下次的冷啟動、暖啟動或磁碟系統重置後，才會再改變回 BOOT+0080H。

## 功能27：取得配置位址

入口參數：

暫存器 C：1BH

回返回值：

暫存器對HL：配置向量位址

每個佔線磁碟機都有個配置向量存在主記憶體中，許多不同的系統程式皆由配置向量所提供的訊息，決定出磁碟所剩餘的記憶

容量。功能 27 的回返值，就是現行選擇磁碟機配置向量所在的位址對於宣告為唯讀狀態的磁碟，配置向量的訊息是無效的。有關配置向量的細節，讀者可參看“CP/M Alternation Guide”。應用程式不常用此功能。

### 功能28：寫入保護磁碟

入口參數：

暫存器 C : 1CH

此功能在將現行選擇的磁碟，暫時設定為寫入保護。在下個冷啓動或暖啓動之前，任何要寫入到該磁碟的動作，都會產生“Bdos Err on d:R/O”的信息。

### 功能29：取得唯讀向量

入口參數：

暫存器 C : 1DH

回返值：

暫存器對HL：唯讀向量值

此功能把顯示各個磁碟機為唯讀狀態的比次向量存於暫存器對HL中，暫存器L的最低位比次對應於磁碟機A，暫存器H的最高比次對應於磁碟機P。至於唯讀比次的設定，一為呼叫功能28的結果，二為CP/M系統在偵測出更換磁碟後軟體自動設定。

### 功能30：設定檔案屬性

入口參數：



## 202 Apple II 軟體卡

暫存器 C : 1EH

暫存器對DE: FCB位址

回返回值：

暫存器 A : 索引碼

此功能允許以程式來處理檔案的屬性，特別是對唯讀狀態及系統檔案的屬性，使用者可依所需來設定它或重置它。功能30 首先找尋一個名稱與FCB 相符合的檔案，依據FCB單一檔名屬性的設定或重置，改變該檔案的索引項目，使其變成新的屬性。  
( R / O與系統屬性由 t1' 及 t2' 設定或重置 )。

### 功能31：取得磁碟參數位址

入口參數：

暫存器 C : 1FH

回返回值：

暫存器對HL：磁碟參數位址

呼叫此功能後，BIOS 的磁碟參數區位址會存放在暫存器對HL 中。這個位址有兩個目的：一為磁碟參數可由此抽取出來顯示，以計算所需空間。二為暫態程式可在磁碟設備變化時，更動磁碟參數來適應系統的變遷。一般應用程式不需此功能。

### 功能32：設定或讀取使用者代號

入口參數：

暫存器 C : 20H

暫存器 E : 0FH或使用者代號

回返回值：

暫存器 A：使用者代號或沒有值

應用程式可利用此功能，改變或查詢現行的使用者代號。若暫存器 E = 0FFH，則現行使用者代號會存放到暫存器 A，其值介於 0 到 31 之間。若不等於 0FFH，則現行使用者代號以 E 值代之，其值也介於 0 到 31 之間。

### 功能33：讀取隨機檔

入口參數：

暫存器 C：21H

暫存器對 DE：FCB 位址

回返回值：

暫存器 A：回返碼

隨機檔的讀取作業與循序檔相似，唯一不同的是，隨機讀取是讀取 FCB 中 r0, r1, r2 所指定號碼的資料錄。其中 r2 僅在計算檔案的大小時才使用，必須設定為 0，若不為 0，則表示溢位，超越檔案結尾。r0 與 r1 視為一雙拜，或稱為“字”（word），所含的內容，就是要讀取的資料錄號數，其值的範圍由 0 到 65535，由此可存取到八百萬拜檔案中的任一資料錄。在隨機處理檔案前，首先要開啓基底延伸區，雖然基底延伸區並不一定已含配置的資料，但這動作可確保檔案在索引裏已適當地登錄，並可由 DIR 命令觀察。接著把欲讀取的資料錄號數存放在隨機記錄號數欄（r0, r1），呼叫 BDOS 即可讀取所要的資料錄。回返回值暫存器 A 若不為零，則代表錯誤代碼，若為零就表示讀取作業成功，此時 DMA 位址中記憶體所存的即是隨機

讀取的資料錄。與循序讀取作業不同的是，資料錄號數欄的值並未加 1，如此，下個隨機讀取動作，仍會讀到相同的資料錄。

每次隨機讀取後，邏輯延伸區與現行資料錄號數已自動設定，因此檔案可從現行位置開始，作循序讀取或循序寫入。注意，在由隨機存取轉換成循序存取時，最後一次處理的資料錄資料，會被重讀，或重寫入。當然，你可在隨機讀取或寫入後先把號數加 1，就可獲得循序 I / O 的效果。

下列是隨機存取失敗後，存在暫存器 A 的錯誤代碼：

- 01 欲讀取尚未寫入的資料
- 02 (在隨機模式中此碼不返回)
- 03 無法關閉現行的延伸區
- 04 找至未寫入的延伸區
- 05 (在讀取模式中此碼不返回)
- 06 超過磁碟實體的結尾

錯誤代碼 01 及 04 在隨機讀取一尚未寫入資料或尚未建立的延伸區時發生的。03 在正常的系統作業中是不會發生的。只要磁碟不是禁寫入，可利用重讀或重開啓第 0 延伸區消除之。06 發生於拜 r2 的值不等於 0。通常回返碼若不等於 0，可視為資料不存在，等於 0 表示資料成功讀取。

### 功能34：寫入隨機檔

入口參數：

暫存器 C : 22H

暫存器對 DE : FCB 位址

回返回值：

暫存器 A：回返碼

除了用現今 DMA 位址的資料寫入磁碟外，寫入磁碟機與讀取隨機檔的準備工作是類似的。若是磁碟延伸區或資料區尚未配置，配置的工作會在寫入作業前完成。同樣的，隨機資料錄號數在寫入之後並未改變，而邏輯延伸區及現所使用資料錄位置會被設定為對應於正被寫入的資料錄。因此在隨機寫入後，也可做循序讀取或循序寫入，注意事項與隨機讀取所提的相同。另一要特別注意的是，隨機存取不像循序存取一樣在讀寫一個延伸區最後一筆資料錄後會自動開啓另一延伸區。錯誤代碼除增加 05 表索引已滿仍無法建立另一新的延伸區外，其餘的皆與隨機讀取的相同。

### 功能 35：計算檔案大小

入口參數：

暫存器 C：23H

暫存器對 DE：FCB 位址

回返回值：

設定隨機資料錄欄位

要計算檔案大小時，將隨機存取格式的 FCB 位址存放在暫存器對 DE 中，FCB 所含的必須是個單一檔案名稱。這個名稱用來與索引作比對。程式回返時，隨機資料錄欄位含虛擬 (virtual) 檔案的大小，事實上也是檔案最後一個資料錄的位置。若 r2 值為 01，表檔案含最大的資料錄數 65536，否則位元組 r0 與 r1 組成一 16 比次的值表示檔案大小。

利用功能35，找出目前的隨機存取資料錄數目 ( r0,r1 )，再將現行資料錄號數 ( cr )，設定為 r0,r1所代表的值，讀者就可將額外資料加入一個檔案的結尾。

若檔案是個循序寫入的循序檔，那麼虛擬值的大小即是檔實質的大小。但對隨機存取檔案言，因可能有空洞散佈其中，所以檔案的實際大小可能小於虛擬值。例如，在一個長為八百萬拜的隨機檔，若只有最後一資料錄被寫上，雖其虛擬值為65536，但實際大小却為1。

### 功能36：設定隨機資料錄號數

入口參數：

暫存器 C : 24H

暫存器對 DE : FCB位址

回返值：

設定隨機資料錄號數欄

此功能使BDOS自動由剛被循序讀寫資料錄至特定點的檔案產生隨機資料錄位置，這在下列兩種作業上是相當有用的。

第一是：在開始時常須先循序讀取循序檔以找出指定的“關鍵”欄。每當遇到此關鍵欄時，即呼叫功能36以計算出對應這關鍵欄的資料錄的隨機資料錄位置。若資料錄的單位長度為128個拜，算出的資料錄位址及鍵可放在記憶體的一張碼表中，待讀完整個檔案後，讀者可利用這張碼表找各鍵所對應的資料錄位址，長度的資料錄，讀取所要的資料。這種作業方式也可用來處理可變以隨機讀取方式者不妨動一下腦筋想想看。

第二是要由循序讀寫轉換成隨機讀寫。當循序存取檔案到某一定點時

讀者可呼叫功能 36 以設定隨機資料錄號數，此後即可轉成爲隨機存取。

## 檔案複製程式範例

下面所示的程式，是個很簡單的檔案作業範例。程式的源檔是由 CP/M ED 系統程式建立的，檔案名稱爲 COPY.ASM，經過 ASM或MAC組合編譯後，產生“HEX”檔，再經由 LOAD 程式產生可直接在 CCP 執行的 COPY.COM 檔。程式由設定堆疊指標開始，然後把擺在 006CH 的第二個檔案名，移到一個長度爲 33 拜，叫做DFCB 的檔案控制區，而由 CCP 在進入 COPY 程式時建立的源檔案控制區 SFCB（位址爲 005CH）也已設定好，接著將某些欄位清除爲零，其中包括現行資料錄欄 007CH。程式於是開啓檔案，並刪除任何現存的目的檔，然後建立新的目的檔，若一切順利的話，程式便在標記COPY 處作迴路，直至每筆資料錄都已從源檔案複製到目的檔爲止，再關閉目的檔，並跳到BOOT 位址，把控制權交還給 CCP。

```

;      sample file-to-file copy program
;
;      at the ccp level, the command
;
;          copy a:x.y b:u.v
;
;      copies the file named x.y from drive
;      a to a file named u.v on drive b.
;
0000 = boot    equ    0000h ; system reboot
0005 = bdos   equ    0005h ; bdos entry point
005c = fcbl   equ    005ch ; first file name
005c = sfcbl  equ    005ch ; source fcb
006c = fcbl   equ    006ch ; second file name
0080 = dbuff  equ    0080h ; default buffer
0100 = tpa    equ    0100h ; beginning of tpa
;
0009 = printf equ    9    ; print buffer func†
000f = openf  equ   15    ; open file func†
0010 = closef equ   16    ; close file func†
0013 = deletf equ   19    ; delete file func†
0014 = readf  equ   20    ; sequential read

```

# 208 Apple II 軟體卡

```

0015 =      writef equ    21      ; sequential write
0016 =      makef  equ    22      ; make file funct
;
0100                org    tpa      ; beginning of tpa
0100 311b02        lxi    sp,stack; local stack
;
;
;      move second file name to dfcb
0103 0e10        mvi    c,16      ; half an fcb
0105 116c00        lxi    d,fc2    ; source of move
0108 21da01        lxi    h,dfcb   ; destination fcb
010b 10          mfcbb; ldx    d      ; source fcb
010c 13          inx    d      ; ready next
010d 77          mov    m,a      ; dest fcb
010e 23          inx    h      ; ready next
010f 0d          dcr    c      ; count 16...0
0110 c20b01        jnz    mfcbb   ; loop 16 times
;
;
;      name has been moved, zero cr
0113 af          xra    a      ; a = 00h
0114 32fa01        sta    dfcbcr  ; current rec = 0
;
;
;      source and destination fcb's ready
;
;
0117 115c00        lxi    d,sfcb   ; source file
011a cd6901        call   open   ; error if 255
011d 118701        lxi    d,nofile; ready message
0120 3c          inc    a      ; 255 becomes 0
0121 cc6101        cz     finis   ; done if no file
;
;
;      source file open, prep destination
0124 11da01        lxi    d,dfcb   ; destination
0127 cd7301        call   delete ; remove if present
;
;
012a 11da01        lxi    d,dfcb   ; destination
012d cd8201        call   make   ; create the file
0130 119601        lxi    d,nodir  ; ready message
;
;
0133 3c          inr    a      ; 255 becomes 0
0134 cc6101        cz     finis   ; done if no dir space
;
;
;      source file open, dest file open
;      copy until end of file on source
;
0137 115c00 copy:  lxi    d,sfcb   ; source
013a cd7801        call   read   ; read next record
013d b7          ora    a      ; end of file?
013e c25101        jnz    eofile   ; skip write if so
;
;
;      not end of file, write the record
0141 11da01        lxi    d,dfcb   ; destination
0144 cd7d01        call   write  ; write record
0147 11a901        lxi    d,space  ; ready message
014a b7          ora    a      ; 00 if write ok
014b c46101        cn-   finis   ; end if so
014e c33701        jmp    copy   ; loop until eof
;
;
; eofile: ; end of file, close destination
0151 11da01        lxi    d,dfcb   ; destination
0154 cd6e01        call   close  ; 255 if error
0157 21bb01        lxi    h,wprot; ready message
015a 3c          inr    a      ; 255 becomes 00
015b cc6101        cz     finis   ; shouldn't happen
;
;
;      copy operation complete, end
015e 11cc01        lxi    d,normal; ready message
;
;
; finis: ; write message given by de, reboot
0161 0e09        mvi    c,printf
0163 cd0500        call   bdos   ; write message
0166 c30000        jmp    boot   ; reboot system
;
;
;      system interface subroutines
;      (all return directly from bdos)
;

```

```

0169 0e0f  open:  mvi    c,openf
016b c30500      jmp    bdos

016e 0e10  close: mvi    c,closef
0170 c30500      jmp    bdos

0173 0e13  delete: mvi   c,deletef
0175 c30500      jmp    bdos

0178 0e14  read:  mvi    c,readf
017a c30500      jmp    bdos

017d 0e15  write: mvi   c,writef
017f c30500      jmp    oDOS

0182 0e16  make:  mvi   c,makef
0184 c30500      jmp    bdos

;
; console messages
0187 6e6f20noFile: db    'no source files'
0196 6e6f20noDir:  db    'no directory spaces'
01a5 6f7574space:  db    'out of data spaces'
01bb 777269surprot: db  'write protected?'
01c 616f70normal: db    'copy completes'
;
; data areas
01da          dfcb:  ds    33      ; destination fcb
01fa          dtcbcr equ   dtcb+32 ; current record
;
01fb          stack:  as    32      ; 16 level stack
021b          end

```

注意，在這個程式中有幾項應做的工作被簡化掉了，第一，程式並未檢查使用者輸入的檔案名稱是否正確，例如群體檔案名在此是不允許的，這可由檢查位址 005CH 起的 32 個拜，是否含有問號來決定。第二，程式並未判斷程式命令中，是否含有檔案名，這可由檢查位址 005DH 及 006DH 有否空白的 ASCII 字元來決定。第三，程式並未檢查源檔案名是否與目的檔名相同。另外，程式可擴大緩衝區的空間，以加快作業的執行速度，這可由位址 0006H 取得 FBASE 的值，再把剩下的記憶體空間整個當做緩衝區。

## 檔案傾印程式範例

下面的傾印程式較上節的檔案複製程式稍許複雜，傾印程式由 CCP 命令行指定的輸入檔案，讀取資料錄，將其以十六進位的格式，在控制台顯示出來。注意，在進入傾印程式時，傾印程式會保存



## 210 Apple II 軟體卡

CCP 的堆疊，重設堆疊到另一地方性的區域，等回返 CCP 之前，再重存 CCP 的堆疊。如此，傾印程式在處理完畢後，可不作暖啓動重置系統。

```

; DUMP program reads input file and displays hex data
;
0100          org      100h
0205 =       bdos     equ      0005h ;dos entry point
0201 =       cons     equ      1      ;read console
0202 =       typef    equ      2      ;type function
0209 =       printf   equ      9      ;buffer print'entry
020b =       brkf     equ      11     ;break key function (true if char
020f =       openf    equ      15     ;file open
0214 =       readf    equ      20     ;read function
;
025c =       fcb      equ      5ch    ;file control block address
0200 =       buff     equ      80h    ;input disk buffer address
;
;       non graphic characters
020d =       cr       equ      0dh    ;carriage return
020a =       lf       equ      0ah    ;line feed
;
;       file control block definitions
025c =       fcbdn    equ      fcb+0  ;disk name
025d =       fcbfn    equ      fcb+1  ;file name
0265 =       fcbft    equ      fcb+9  ;disk file type (3 characters)
0268 =       fcbri    equ      fcb+12 ;file's current reel number
026b =       fcbrc    equ      fcb+15 ;file's record count (0 to 128)
027c =       fcbcr    equ      fcb+32 ;current (next) record number (0
027d =       fcbia    equ      fcb+33 ;fcb length
;
;       set up stack
0100 210000   lxi      h,0
0103 39       dad      sp
;       entry stack pointer in hl from the ccp
;
0104 221502   shld     oldsp
;       set sp to local stack area (restored at finis)
0107 315702   lxi      sp,stktop
;       read and print successive buffers
010a cdc101   call     setup ;set up input file
010d feff     cpi      255 ;255 if file not present
010f c21b01   jnz     openok ;skip if open is ok
;
;       file not there, give error message and return
0112 11f301   lxi      d,opnmsg
0115 cd9c01   call     err
0118 c35101   jmp     finis ;to return
;
openok: ;open operation ok, set buffer index to end
011b 3e00     nvi      a,80h
011d 321302   sta     ibp ;set buffer pointer to 80h
;       hl contains next address to print
0120 210000   lxi      h,0 ;start with 0000
;
;       gloop:
0123 e5       push     h ;save line position
0124 cda201   call    gnb
0127 e1       pop     h ;recall line position
0128 da5101   jc      finis ;carry set by gnb if end file
012b 47       mov     b,a
;       print hex values
;       check for line fold
012c 7d       mov     a,l
012d e60f     ani     0fh ;check low 4 bits
012f c24401   jnz     nonum
;       print line number
0132 cd7201   call    crlf

```

```

;
; check for break key
0135 cd5901 call break
;
; accum lsb = 1 if character ready
0138 0f rrc ;into carry
0139 da5101 jc finis ;don't print any more
;
013c 7c mov a,h
013d cd0f01 call phex
0140 7d ov a,l
0141 cd0f01 call phex
nonum:
0144 23 inx h ;to next line number
0145 3e20 mvi a,' '
0147 cd6501 call pchar
014a 78 mov a,b
014b cd0f01 call phex
014e c32301 jmp glocp
;
; finis:
; end of dump, return to ccp
; (note that a jmp to 0000h reboots)
0151 cd7201 call crlf
0154 2a1502 lhid oldsp
0157 f9 sphl
; stack pointer contains ccp's stack location
0158 c9 ret ;to the ccp
;
;
; subroutines
;
break: ;check break key (actually any key will do)
0159 e5d5c5 push b; push d; push b; environment saved
015c 0e0b mvi c,brkf
015e cd0500 call bdos
0161 c1d1e1 pop b; pop d; pop h; environment restored
0164 c9 ret
;
pchar: ;print a character
;
0165 e5d5c5 push h; push d; push b; saved
0168 0e02 mvi c,typef
016a 5f mov e,a
016b cd0500 call bdos
016e c1d1e1 pop b; pop d; pop h; restored
0171 c9 ret
;
; crlf:
0172 3e0d mvi a,cr
0174 cd6501 call pchar
0177 3e0a mvi a,lf
0179 cd6501 call pchar
017c c9 ret
;
;
; pnib: ;print nibble in reg a
017d c60f ani 0fh ;low 4 bits
017f fe0a cpl 10
0181 d28901 jnc pl0
; less than or equal to 9
0184 c630 adi '0'
0186 c30b01 jmp prn
;
;
; greater or equal to 10
0189 c637 pl0: adi 'a' - 10
018b cd6501 prn: call pchar
018e c9 ret
;
;
; phex: ;print hex char in reg a
018f f5 push psw
0190 0f rrc
0191 0f rrc
0192 0f rrc
0193 0f rrc

```

# 212 Apple II 軟體卡

```

0194 c07d01      call    pnib      ;print nibble
0197 f1         pop     psw
0198 cd7d01      call    pnib
019b c9         ret
;
err:            ;print error message
;               d,e addresses message ending with '$'
019c 0e09        mvi    c,printf  ;print buffer function
019e cd0500      call    bdos
01a1 c9         ret
;
;
gnb:           ;get next byte
01a2 3a1302      lda    ibp
01a5 fe00        cpi    00h
01a7 c2c301      jnz    g0
;               read another buffer
;
;
01aa cdce01      call    diskr
01ad b7         ora    a         ;zero value if read ok
01ae cab301      js    g0         ;for another byte
;               end of data, return with carry set for eof
01b1 37         stc
01b2 c9         ret
;
g0:            ;read the byte at buff+reg a
01b3 5f         mov    e,e      ;ls byte of buffer index
01b4 1600        mvi    d,0      ;double precision index to de
01b6 3c         ldr    a         ;index=index+1
01b7 321302      sta    ibp      ;back to memory
;               pointer is incremented
;               save the current file address
01ba 210000      lxi    h,buff
01bd 19         dad    d
;               absolute character address is in hl
01be 7e         mov    a,m
;
;               byte is in the accumulator
01bf b7         ora    a         ;reset carry bit
01c0 c9         ret
;
setup:         ;set up file
;               open the file for input
01c1 af         xra    a         ;zero to accum
01c2 327c00      sta    fcbcr    ;clear current record
;
01c5 115c00      lxi    d,fcbr
01c8 0e0f        mvi    c,openf
01ca cd0500      call    bdos
;               255 in accum if open error
01cd c9         ret
;
diskr:        ;read disk file record
01ce e5d5c5      push h| push d| push b
01d1 115c00      lxi    d,fcbr
01d4 0e14        mvi    c,readf
01d6 cd0500      call    bdos
01d9 c1d1e1      pop b| pop d| pop h
01dc c9         ret
;
;               fixed message area
01dd 46494c0a$ignon: db    'file du p version 2.05'
01ff 0d0a4e0a$opnmsg: db    'cr,lf,'no input file present on disk$'
;
;               variable area
0213            ibp:    ds    2      ;input buffer pointer
0215            oldsp: ds    2      ;entry sp value from ccp
;
;               stack area
0217            ds    64      ;reserve 32 level stack
stktop:
;
0257            end

```

## 隨機存取程式範例

在此將以較為廣泛，但却較完整的隨機存取程式來作為本章的結束。下面所列的程式，會依照從終端機輸入的命令，執行隨機資料錄的讀取及寫入。假定程式已建立，並經組合編譯成RANDOM.COM。

CCP 命令

RANDOM X.DAT

用來啟動程式的執行，程式首先找尋一個叫X.DAT的檔案，找到就在控制台顯示訊息等待輸入，若找不到就在訊息顯示之前，建立一個空檔案。顯示的訊息格式如下：

next command?

見到這訊息之後，操作員鍵入輸入命令，並以回車鍵（< cr >）作為輸入命令的結束。輸入命令格式為：

nW nR Q

其中n為一介於0到65535的整數值，W，R和Q是簡單的命令字元，其意義分別是隨機寫入，隨機讀寫及終止處理。若輸入W命令字元，RANDOM程式會顯示訊息

type data:

操作員在此至多可輸入 127 個字元，並以回車鍵終結。RANDOM 程式便會把輸入的字串寫入檔案 X.DAT 的第 n 筆資料錄。若命令字元為 R，RANDOM 程式會讀取檔案 X.DAT 的第 n 筆資料錄，並把該錄的內容顯示在控制台。若為 Q 命令字元，RANDOM 程式會關閉檔案 X.DAT，並把控制權交還給 CCP。為了簡潔起見，唯一的錯誤訊息是

error, try again

程式首先開啓或建立輸入檔案，然後在標誌為“ready”的地方作迴圈循環，以解析輸入的命令。FCB 位址的自然值為 005CH，緩衝區位址的自然值為 0080H。另有一叫做“readc”的副程式，它用來處理從終端機鍵入的輸入行。RANDOM 程式展示了隨機存取處理的原理，因此可用來作為程式進一步發展的基礎。

```

;.....
;*
;* sample random access program for cp/m 2.0
;*
;.....
0100          org.      100h      ;base of tpa
;
0000 =       reboot   equ     0000h  ;system reboot
0005 =       bdos     equ     0005h  ;bdos entry point
;
0001 =       coninp   equ     1       ;console input function
0002 =       conout   equ     2       ;console output function
0009 =       pstring  equ     9       ;print string until 'S'
000a =       rstring  equ     10      ;read console buffer
000c =       version  equ     12      ;return version number
000f =       openf    equ     15      ;file open function
0010 =       closef   equ     16      ;close function
0016 =       makef    equ     22      ;make file function
0021 =       readr    equ     33      ;read random
0022 =       writer   equ     34      ;write random
;
005c =       fcb      equ     005ch   ;default file control block
007d =       ranrec   equ     fcb+33  ;random record position
007f =       ranovf   equ     fcb+35  ;high order (overflow) byte
0080 =       buff     equ     0080h   ;buffer address
;
000d =       cr       equ     0dh     ;carriage return
000a =       lf       equ     0ah     ;line-feed
;
;.....
;*
;* load SP, set-up file for random access
;*
;.....

```

```

0100 31bc0      lxi    sp,stack
:
:      version 2.0?
0103 0e8c      mvi    c,version
0105 cd050     call   bdos
0108 fe20      cpi    20h      ;version 2.0 or better?
010a d2160     jnc    versok
:      bad version, message and go back
010d 111b0     lxi    d,badver
0110 cdda0     call   print
0113 c3000     jmp    reboot

versok:
:      correct version for random access
0116 0e8f      mvi    c,openf ;open default fcb
0118 015c0     lxi    d,fcf
011b cd050     call   bdos
011e 3c       inr    a      ;err 255 becomes zero
011f c2370     jnz    ready
:
:      cannot open file, so create it
0122 0e16      mvi    c,makef
0124 115c0     lxi    d,fcf
0127 cd050     call   bdos
012a 3c       inr    a      ;err 255 becomes zero
012b c2370     jnz    ready
:
:      cannot create file, directory full
012e 113a0     lxi    d,nospace
0131 cdda0     call   print
0134 c3000     jmp    reboot ;back to ccp
:
: .....
: *
: * loop back to "ready" after each command
: *
: .....
:
ready:
:      file is ready for processing
:
0137 cde50     call   readcom ;read next command
013a 227d0     shld   ranrec ;store input record#
013d 217f0     lxi    h,ranovf
0140 3600      mvi    m,0    ;clear high byte if set
0142 fe51     cpi    'Q'    ;quit?
0144 c2560     jnz    notq
:
:      quit processing, close file
0147 ve10     mvi    c,closef
0149 115c0     lxi    d,fcf
014c cd050     call   bdos
014f 3c       inr    a      ;err 255 becomes 0
0150 cab90     jz     error ;error message, retry
0153 c3000     jmp    reboot ;back to ccp
:
: .....
: *
: * end of quit command, process write
: *
: .....
notq:
:      not the quit command, random write?
0156 fe57     cpi    'W'
0158 c2890     jnz    notw
:
:      this is a random write, fill buffer until cr
015b 114d0     lxi    d,datmsg
015e cdda0     call   print ;data prompt
0161 0e7f     mvi    c,127 ;up to 127 characters
0163 21800     lxi    h,buff ;destination

```

# 216 Apple II 軟體卡

```

loop:  ;read next character to buff
0166 c5      push  b      ;save counter
0167 e5      push  h      ;next destination
0168 cdc20   call  getchr ;character to a
016a e1      pop   h      ;restore counter
016c c1      pop   b      ;restore next to fill
016d fe8d   cpl   cr     ;end of line?
016f ca780   js    erloop

        not end, store character
0172 77      mov   m,a
0173 23      inx   h      ;next to fill
0174 0d      dcr   c      ;counter goes down
0175 c2660   jnz   rloop  ;end of buffer?

erloop:
        end of read loop, store 00
0178 36C0   mvi   m,0

        write the record to selected record number
017a 0e22   mvi   c,writer
017c 115c0  lxi   d,fcB
017e cd650  call  bdos
0182 b7      ora   a      ;error code zero?
0183 c2b90  jnz   error  ;message if not
0186 c3370  jmp   ready  ;for another record

;.....
;
; end of write command, process read
;
;.....
notw:
        not a write command, read record?
0189 fe52   cpl   'R'
018b c2b90  jnz   error  ;skip if not

        read random record
018e 0e21   mvi   c,readr
0190 115c0  lxi   d,fcB
0192 cd650  call  bdos
0196 b7      ora   a      ;return code ??
0197 c2b90  jnz   error

        read was successful, write to console
019a cdcf0   call  crlf   ;new line
019d 0e80   mvi   c,128 ;max 128 characters
019f 21880  lxi   h,buff ;next to get

wloop:
01a2 7e      mov   a,m    ;next character
01a3 23      inx   h    ;next to get
01a4 e67f   ani   7fh   ;mask parity
01a6 ca370  jz    ready ;for another command if 00
01a9 c5      push  b    ;save counter
01aa c5      push  h    ;save next to get
01ab fe2e   cpl   ' '   ;graphic?
01ad d4c80  cnc   putchr ;skip output if not
01b0 e1      pop   h
01b1 c1      pop   b
01b2 0d      dcr   c    ;count=count-1
01b3 c2a20  jnz   wloop
01b6 c3370  jmp   ready

;.....
;
; end of read command, all errors end-up here
;
;.....
error:
01b9 11590  lxi   d,errmsg
01bc cdda0  call  print
01bf c3370  jmp   ready

```

```

;.....
;
; utility subroutines for console i/o
;.....
getchr:
; read next console character to a
movi c,coninp
01c2 0e01 calli bdos
01c4 cd050 ret
01c7 c9

;
putchr:
; write character from a to console
movi c,conout
01ca 5f mov e,a ;character to send
01cb cd050 calli bdos ;send character
01cc c9 ret

;
crlf:
; send carriage return line feed
01cf 3e0d movi a,cr ;carriage return
01d1 cdc60 calli putchr
01d4 3e0a movi a,lf ;line feed
01d6 cdc00 calli putchr
01d9 c9 ret

;
print:
; print the buffer addressed by de until $
push d
01da d5 calli c,crif
01db cdcf0 calli d ;new line
01de d1 pop d
01df 0e09 movi c,pstring
01e1 cd050 calli bdos ;print the string
01e4 c9 ret

;
readcom:
; read the next command line to the conbuf
01e5 11600 lxi d,prompt
01e8 cdda0 calli print ;command?
01eb 0e0a movi c,rstring
01ed 117a0 lxi d,conbuf
01f0 cd050 calli bdos ;read command line
; command line is present, scan it
01f3 21000 lxi h,0 ;start with 0000
01f6 117c0 lxi d,conlin;command line
01f9 1a readc: ldax d ;next command character
01fa 13 inx d ;to next command position
01fb b7 ora a ;cannot be end of command
01fc c8 rz
; not zero, numeric?
01fd d630 sui '0'
01ff fe0a cpi 10 ;carry if numeric
0201 d2130 jnc endrd
; add-in next digit
0204 29 dad h ;*2
0205 4d mov c,l
0206 44 mov b,h ;bc = value * 2
0207 29 dad h ;*4
0208 29 dad h ;*8
0209 09 dad b ;*2 + *8 = *10
020a 85 add l ;digit
020b 66 mov l,a
020c d1f90 jnc readc ;for another char
020f 24 inr h ;overflow
0210 c3f90 jmp readc ;for another char
;
endrd:
; end of read, restore value in a
0213 c630 adi '0' ;command
0215 fe61 cpi 'a' ;translate case?

```



## 218 Apple II 軟體卡

```

0217 d8          rc
                lower case, mask lower case bits
0218 e65f        ani    10151111b
021a c9          ret
                .....
                * string data area for console messages
                .....
badver:
021b 536f79      db    'sorry, you need cp/m version 25'
nospace:
023a 4e6f29      db    'no directory spaces'
datmsg:
024d 547970      db    'type data: 5'
errmsg:
0259 457272      db    'error, try again.5'
prompt:
026b 4e6570      db    'next command? 5'
                .....
                * fixed and variable data area
                .....
027a 21          conbuf: db    conlen ;length of console buffer
027b             consiz: ds    1      ;resulting size after read
027c             conlin: ds    32     ;length 32 buffer
0021 =           conlen equ    5-consiz
                ;
029c             js      32      ;16 level stack
stack:
02bc             end

```

讀者若有興趣可對 RANDOM 程式作下列幾點改進，如此可提高該程式的運作能力。事實上，若加上某些技術，程式 RANDOM 能發展成一個簡單的資料庫管理系統。例如，我們指定一標準 128 個拜的資料錄，其內含有若干種欄位來組成此錄。我們可以發展一個叫做“GETKEY”的程式，這個程式能讀取循序檔，並且可依操作員的意願，從讀取的資料錄中抽取某特定欄位。例如，命令

GETKEY NAMES.DAT LASTNAME 10 20

使程式 GETKEY 讀取資料庫檔案 NAMES.DAT，從每個資料錄的第 10 個到第 20 個字元，取出 LASTNAME 欄位。程式 GETKEY 在記憶體建一個表，表中含有每一個 LASTNAME 欄及其相對應的隨機資料錄數，並對該表作分類 (Sort)，最後把這表寫到一個叫做 LASTNAME.KEY 的檔案 (這個表在資訊處理上稱為反索引檔)。

把上述程式更名成“Query”，並稍加修飾，使其能把已分類的檔案讀入記憶體，命令行也改為

```
QUERY NAMES.DAT LASTNAME.KEY
```

此時，程式 QUERY 不再讀取一個數值，它所讀取的是個可在資料庫 NAMES.DAT 找到的字串。因為 LASTNAME.KEY 是個已經分類排列好的表，因此我們可利用二分法，很快地找到特定的隨機資料錄號數，再來隨機存取。如此我們便可取得資料的內容，並將其顯示在控制台上。

以此為起步，再加點心思，你就可使用一個大小不為 128 個拜的族群。這可以藉著追踪資料錄號數及拜偏移量 ( bytes offset ) 來完成。當你知道族群的大小時，你即可以隨機存取含有這族群的資料錄，並藉著拜偏移量到資料錄中族群的起點，往下循序讀取直至讀完整個族群為止。

最後，你可以在程式 QUERY 使用布林運算式 ( Boolean Expression )，以增強它的功能。例如你可要求將滿足 LASTNAME 介於 HARDY 與 LAUREL 之間且年齡小於 45 歲的所有資料錄顯示在控制台上。而若你的分類表太大以致於無法存入記憶體，你也可以把它存在磁碟中，再從磁碟中隨機存取此檔。假使你能全盤了解整個作業的過程，則在所有作業完成後的慰藉是，你不再需要本手冊了。

## 系統功能摘要

FUNC	FUNCTION NAME	INPUT PARAMETERS	OUTPUT RESULTS
0	System Reset	none	none
1	Console Input	none	A = char
2	Console Output	E = char	none
3	Reader Input	none	A = char
4	Punch Output	E = char	none
5	List Output	E = char	none
6	Direct Console I/O	see def	see def
7	Get I/O Byte	none	A = IOBYTE
8	Set I/O Byte	E = IOBYTE	none
9	Print String	DE = .Buffer	none
10	Read Console Buffer	DE = .Buffer	see def
11	Get Console Status	none	A = 00/FF
12	Return Version Number	none	HL = Version*
13	Reset Disk System	none	see def
14	Select Disk	E = Disk Number	see def
15	Open File	DE = .FCB	A = Dir Code
16	Close File	DE = .FCB	A = Dir Code
17	Search for First	DE = .FCB	A = Dir Code
18	Search for Next	none	A = Dir Code
19	Delete File	DE = .FCB	A = Dir Code
20	Read Sequential	DE = .FCB	A = Err Code
21	Write Sequential	DE = .FCB	A = Err Code
22	Make File	DE = .FCB	A = Dir Code
23	Rename File	DE = .FCB	A = Dir Code
24	Return Login Vector	none	HL = Login Vect*
25	Return Current Disk	none	A = Cur Disk =
26	Set DMA Address	DE = .DMA	none
27	Get Addr(Alloc)	none	HL = .Alloc
28	Write Protect Disk	none	see def
29	Get R/O Vector	none	HL = R/O Vect*
30	Set File Attributes	DE = .FCB	see def
31	Get Addr (disk parms)	none	HL = .DPB
32	Set/Get User Code	see def	see def
33	Read Random	DE = .FCB	A = Err Code
34	Write Random	DE = .FCB	A = Err Code
35	Compute File Size	DE = .FCB	r0, r1, r2
36	Set Random Record	DE = .FCB	r0, r1, r2

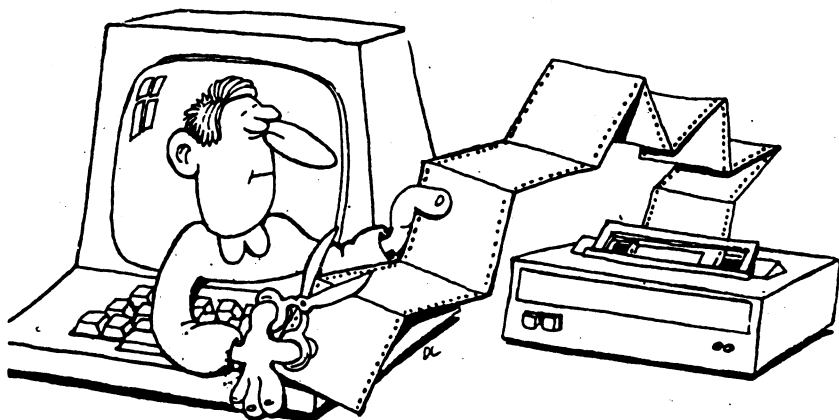
\*Note that A = L, and B = H upon return

## 第三章

# CP/M編輯程式

- 編輯程式 (ED) 簡介
- 編輯程式的操作
- 文稿傳送功能
- 記憶體緩衝區的構造
- 記憶體緩衝區的操作
- 命令字串
- 文稿之搜尋及變更
- 源檔叢庫 (Source Libraries)
- 重覆執行命令
- 編輯錯誤狀況
- 控制字元摘要
- 編輯命令摘要
- 編輯程式文稿編輯命令

## 編輯程式 ( 以下簡稱ED ) 簡介

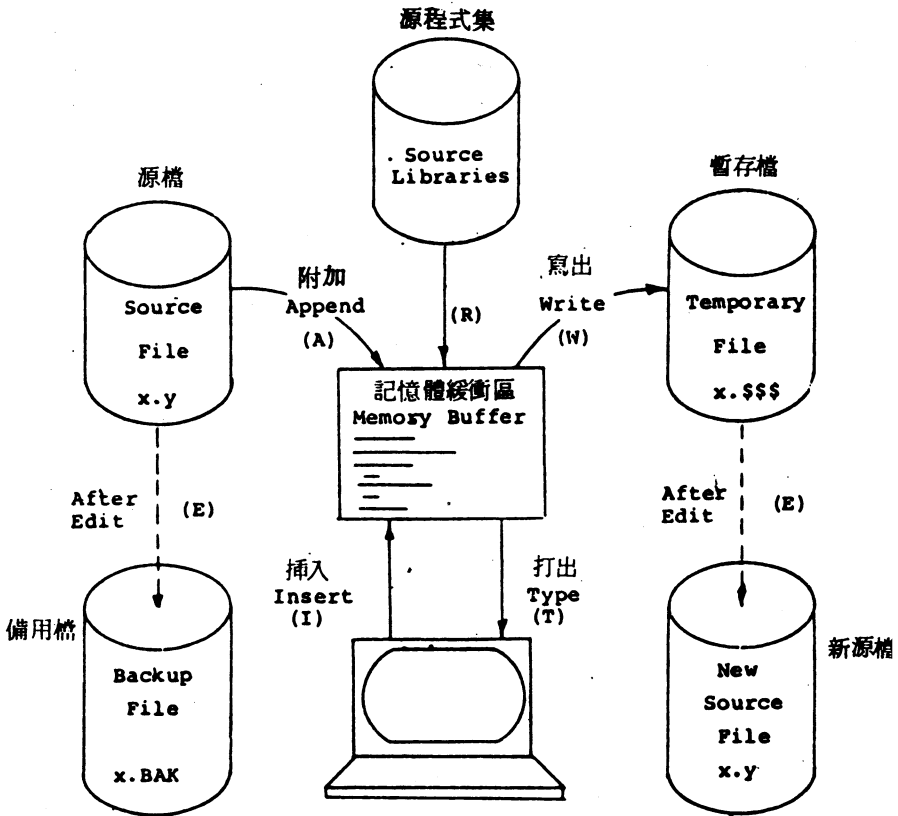


ED 是 CP/M 的編輯程式，用以產生或更改 CP/M 源檔 ( source file )。當我們打入：

$$\text{ED } \left\{ \begin{array}{l} \langle \text{檔案名稱} \rangle \\ \langle \text{檔案名稱} \rangle \cdot \langle \text{檔案類型} \rangle \end{array} \right\}$$

CP/M 就會啟動編輯程式。一般而言，ED 把  $\langle \text{檔案名稱} \rangle$  或是  $\langle \text{檔案名稱} \rangle \cdot \langle \text{檔案類型} \rangle$  所指的源檔 ( source file )，一段一段的讀入中央記憶體，操作員在此改造檔案，更改完成後再寫回磁碟上。假如在編輯之前並無源檔存在，ED 會產生一個空的檔案。ED 的全部動作如圖 1 所示。

圖 1 ED 作業整體架構



如圖 1 所示 ED 程式在檔案 X.Y 上運作，並經由一個記憶體緩衝區傳送所有文稿，這些文稿可以被檢視或更改（在記憶體緩衝區內所能存放的行數隨著所使用之 CP/M 系統不同而有所差異。不過在一個 16KCP/M 系統中，最多只能容納約 6000 字元），編輯過後的文稿，經由操作員的指令而寫入一個暫存檔 (temporary file) 內。當整個編輯動作全部結束時，記憶體緩衝區的內涵以及源檔中原有未

被讀出的文稿都寫入暫存檔案中。此時原來檔案的名稱更改為 X.BAK，因為最近一份先前編輯過的檔案仍存在，所以只要有需要隨時皆可改回來。（參考 CP/M ERASE 及 RENAME 命令），然後這個暫存檔的名字就從 X. \$\$\$ 改為 X.Y，這就是編輯完成的檔案。記憶緩衝區和源檔以及暫存檔之間的邏輯關係如圖 2.1 所示。

圖 2.1 記憶體暫存區組織

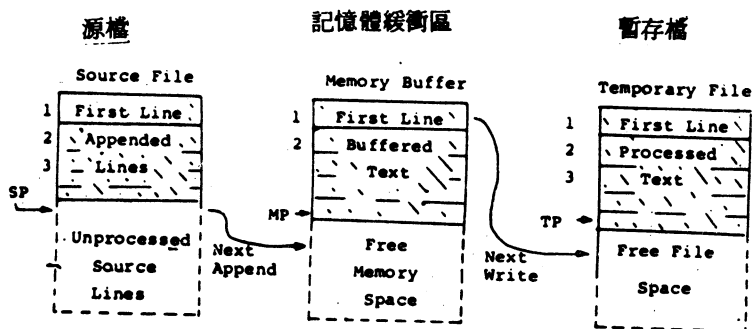
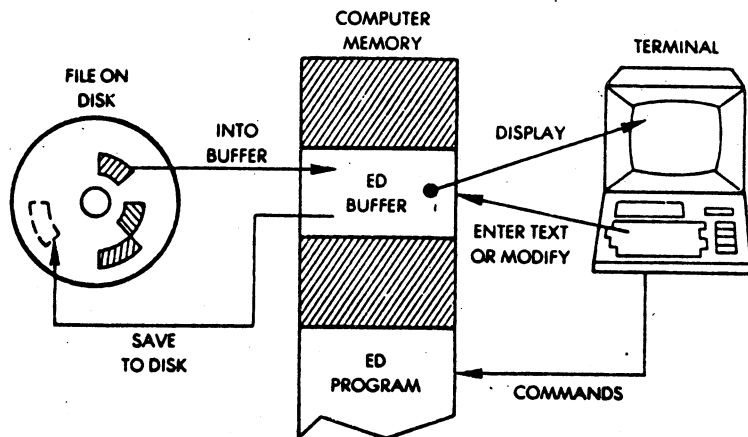


圖 2.2 ED 緩衝區之結構



## 文稿傳送功能

以下的ED命令可用來處理各種不同長度的文稿，它的路徑是從源檔經由記憶暫存區再到暫存檔（最後變成永久檔），其中n代表0到65535的整數。

註：ED程式能接受從控制台(console)上輸入的大小寫ASCII字元。單字母命令不論是以大寫或小寫鍵入都可被接受。

當鍵入U命令，會使ED程式把從主控台打入的小寫字母變成大寫字母再送入記憶體緩衝區中，而回印(echo)在主控台上的字母則和敲入時相同，（小寫字母在螢光幕上仍然是小寫。）而-U（負U）命令則會叫ED程式恢復到“不轉換”狀態，（輸入時是大寫或小寫，忠實地送入記憶體緩衝區中）。ED程式的起始狀態處於正執行-U命令狀態。

nA < cr > 把在源檔中的資料從SP以下n行未被處理過之資料附加到記憶體緩衝區MP之後，再把SP及MP之值各加n。

**實例(1):**假如在磁碟內有一SAMPLE.TEX，它的內容如下：



This is the file SAMPLE.TXT, and this is line 1.  
This is logically line 2 in the file.  
This is line 3, but the buffer may number  
it differently (and this is line 4).  
This is line 5. There are many more lines.  
This is line 6.  
•  
•  
•  
This is line 26.  
This is line 27.  
•  
•  
•  
This is line 43  
•  
•  
•

打入 2A 命令後在緩衝區產生的結果：

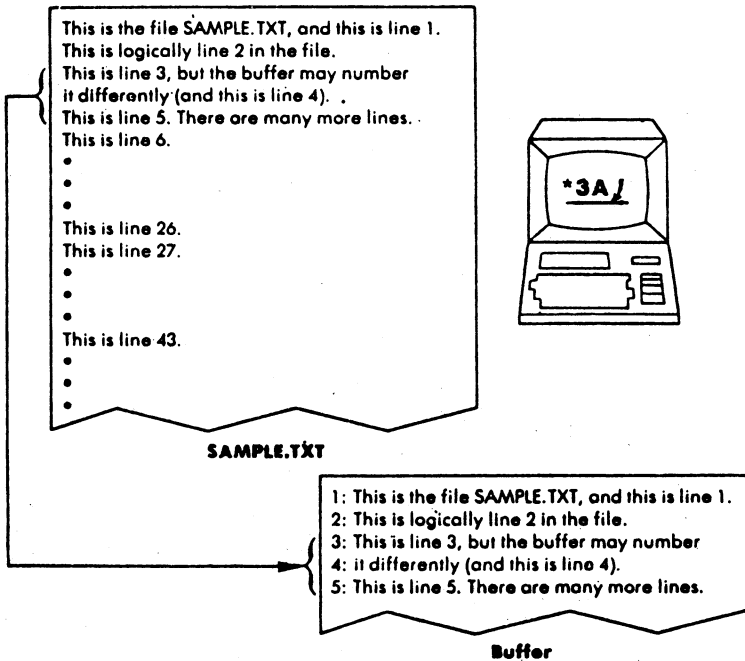


1: This is the file SAMPLE.TXT, and this is line 1.  
2: This is logically line 2 in the file.

(註：命令下之橫線表示是使用者打入的命令)

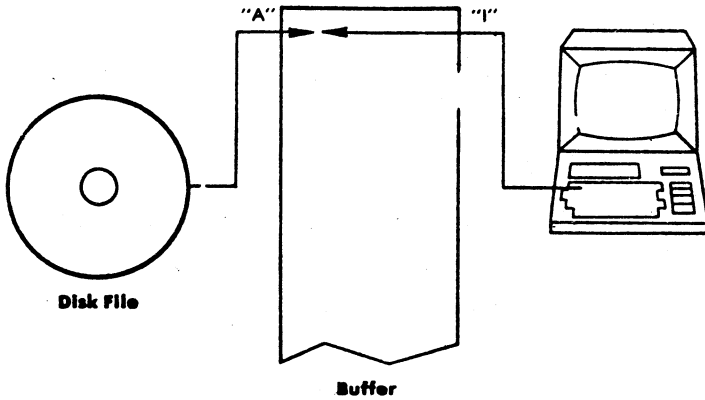
再打入 3A 命令之結果如下：

(註：SP, MP, TP 參考圖 2.1)



`nW < cr >` 把記憶體緩衝區內從頭起  $n$  行的資料寫到暫存檔案中尚未用到的空間，把記憶體緩衝區中從  $n+1$  行起到  $MP$  為止，所餘下的每行資料移到最前頭去。TP 值加  $n$ 。

**實例(2)：**在實例 1 中，如想從螢光幕上插入資料到緩衝區，可用“`I`”命令：



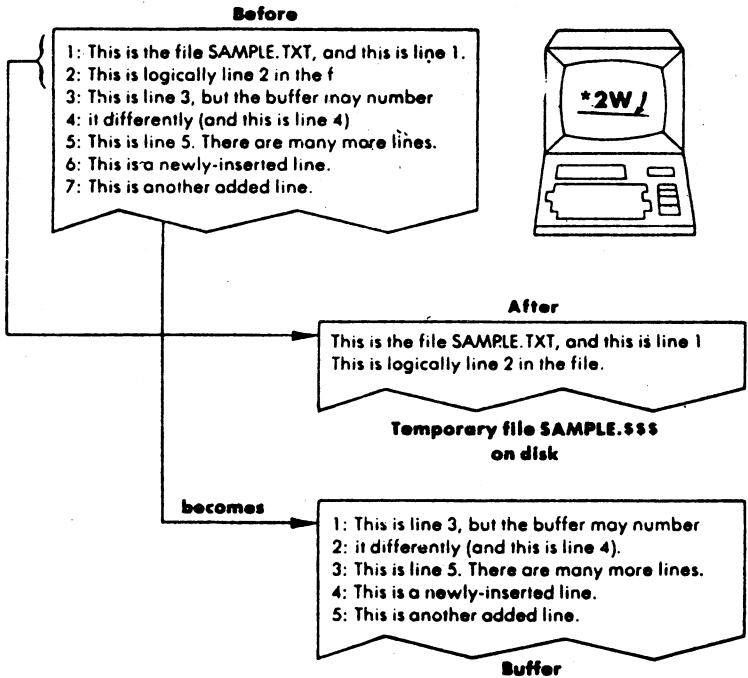
過程如下 (注意, ↑ Z 結束 " I " 命令) :

```
1: This is the file SAMPLE.TXT, and this is line 1.  
2: This is logically line 2 in the file.  
3: This is line 3, but the buffer may number  
4: it differently (and this is line 4).  
5: This is line 5. There are many more lines.  
: ↑  
6: This is a newly-inserted line.  
7: This is another added line.  
8: ↑Z  
:
```

緩衝區

然後再把緩衝區兩行資料存到暫存檔去 (見下圖)。

( 移過去的兩行從緩衝區消失 ) :



E < cr > 結束編輯。把緩衝區中的資料及尚未被處理到的源檔資料全部抄入暫存檔裏，然後依照前面所說的方式，把檔案重新命名。

實例(3)：用“E”命令，可把在緩衝區內已編輯過之資料與在磁碟上此檔尚未附加 (Append) 到緩衝區的剩餘資料合成一新檔，而完成一ED之過程。

This is the file SAMPLE.TXT, and this is line 1.  
 This is logically line 2 in the file.  
 This is line 3, but the buffer may number it differently (and this is line 4).  
 This is line 5. There are many more lines.  
 This is line 6.

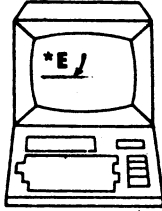
⋮

This is line 26.  
 This is line 27.

⋮

This is line 43.

⋮



**SAMPLE.TXT**  
 renamed to **SAMPLE.BAK**  
 (original file)

output  
 entire  
 buffer

1: This is line 3, but the buffer may number  
 2: it differently (and this is line 4).  
 3: This is line 5. There are many more lines.  
 4: This is a newly-inserted line.  
 5: This is another added line.  
 6: This is line 6.

⋮

**Buffer**

**Temporary output file**  
**SAMPLE.???** renamed to  
**SAMPLE.TXT**  
 (edited file)

This is the file SAMPLE.TXT, and this is line 1.  
 This is logically line 2 in the file.  
 This is line 3, but the buffer may number it differently (and this is line 4).  
 This is line 5. There are many more lines.  
 This is a newly-inserted line.  
 This is another added line.  
 This is line 6.

⋮

This is line 26.  
 This is line 27.

⋮

This is line 43.

⋮

- H < cr > 自動執行E命令，並且移到新檔案的開頭位置。暫存檔變成新的源檔，記憶體緩衝區被清除另外產生了一個新的暫存檔（相當於發出E命令後，接著再叫出ED來編輯X.Y檔案）。
- O < cr > 回到原先的檔案，記憶體緩衝區被清除，暫存檔被刪除，SP指回檔的第一個位置，先前所發出的編輯命令都變成無效。
- Q < cr > 沒有任何檔案被更改，離開編輯程式回到CP/M。

有許多特別的情形要考慮，在上面提到的任何編輯命令中，凡應有正整數n而n被省略者，就被認定為1。所以，命令A是附加一行到記憶體緩衝區，而命令W則是寫一行到暫存檔中。

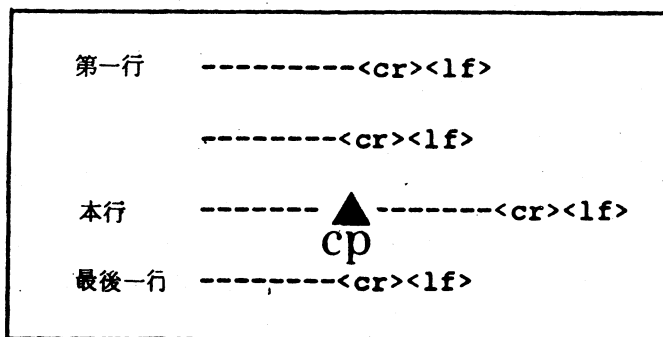
如果以（#）符號代替n，則表示整數65535（n所可能之最大值），因為絕大多數的源檔都能全部存放在記憶體緩衝區中，所以我們經常在編輯檔案之初，就以#A指令把檔的內容全讀入記憶體中。同樣的#W指令把緩衝區內容全寫入暫存檔案。此外，為了便利操作，ED還提供兩種特殊形式的命令：指令0A，把資料放入記憶體緩衝區中，直到填滿一半為止。指令0W，則把資料寫入暫存檔內，直到緩衝區至少一半為空的為止。

在執行各種ED命令時，如果寫入記憶體緩衝區的資料超過它所能容納的數量，ED會發出錯誤信號。這時操作員只能使用不要求記憶體空間的命令（如W）。因記憶體緩衝區已滿而未能讀入的資料將在下一次成功的附加命令中讀入記憶體緩衝區中。

## 記憶體緩衝區構造

我們可以把記憶體緩衝區看為是以A命令從源檔輸入的連續的源檔資料所構成。記憶體緩衝區有一個相關的字元指標(CP)，這一指標隨著操作員所下命令在記憶體緩衝區內移動。圖3表示記憶體緩衝區的邏輯組織，其中虛線代表一行字元資料而以<cr>(carrage return)與<lf>(line feed)兩組字元做為一行字元的結束。CP表示字元指標。字元指標CP位於記憶體緩衝區內，它要不是在第一個字元的前頭或最後一個字元的後面，就是介於兩個字元中間。CL(current line)就是含有CP的那一行。

圖3 記憶體緩衝區的邏輯結構



## 記憶體緩衝區之操作

當ED剛開始運作時，記憶體緩衝區是空的，操作員可以用A命

令，把源檔中的資料附加入記憶體中，或是用 I 命令—— I < cr >，直接從控制台輸入——。

ED 接受輸入的各行資料，每行都是以 < cr > 作結束 (< lf > 是系統自動附上的) 直到操作員打入 control - Z (以 ↑ Z 表示) 為止。這時 CP 的位置在最後輸入的字元之後。

操作員打入資料的順序如下：

```
I<cr>
NOW IS THE<cr>
TIME FOR<cr>
ALL GOOD MEN<cr>
↑z
```

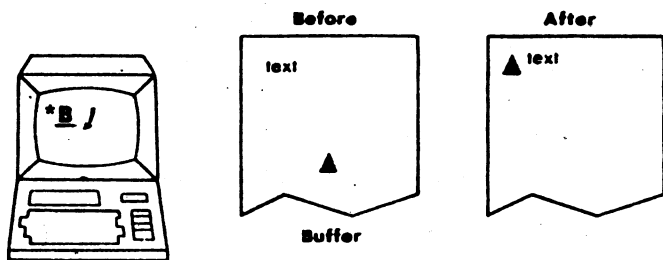
這些資料離開記憶體緩衝器時如下：

```
NOW IS THE<cr><lf>
TIME FOR<cr><lf>
ALL GOOD MEN<cr><lf>␣
```

有好幾種命令可以改變 CP 的位置，或是顯示 CP 附近的文稿。在下面所提到的命令，凡是前面有一個 n 的表示 n 可以賦予任何整數值。如果前面有 ± 符號的，那麼這些命令前面可以給 “+” 或 “-” 或是都不給也可以。另外如以前所說的 # 表示 65535。n 若未設定，系統就認為是 1。而前面可以冠 “±” 號的命令，省略不寫時系統就認定是 “+”。

± B < cr > 若是 “+” CP 移到記憶體緩衝區的最開始之處。反之若為 “-” 號時，CP 移到記憶體緩衝區的最後之處。



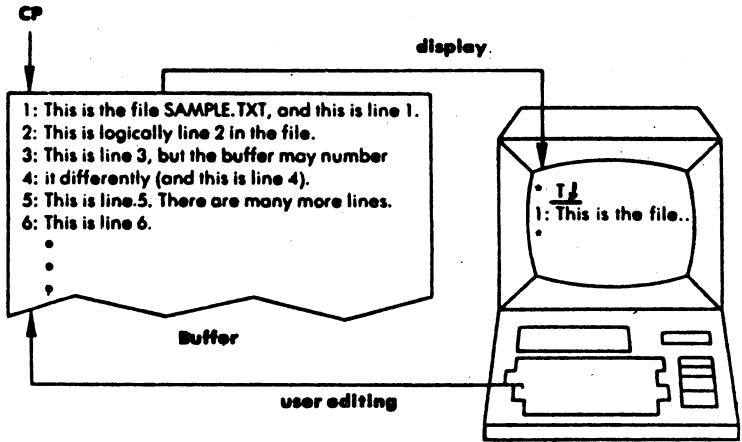


把游標移至最前方

- ± nC < cr >    移動 CP 指標 n 個字元，< cr >、< lf > 也算做兩個獨立的字元。
- ± nD < cr >    “+”時刪去 CP 指標右邊 n 個字元。若為“-”時刪去 CP 指標左邊 n 個字元。
- ± nK < cr >    以 CP 指標為參考點，+nK 則除去 CP 以下的 n 行源資料，-nK 則除去 CP 以上的 n 行源資料。
- ± nL < cr >    若 n = 0，則 CP 指標移到現在那一行的起始點，當 n ≠ 0 時，先將 CP 移到那一行的起始點然後若是 + 將 CP 移下 n 行，若是 - 則把 CP 移上 n 行。如果所給的 n 值太大，則 CP 會停留在記憶體緩衝區的最前端或是最底端。
- ± nT < cr >    n = 0 時，把 CP 所在那一行的資料從頭起到 CP 為止印出來。n = 1 時，則把 CP 所在那一行的資料，從 CP 起到這行結束為止印出來。n > 1 且為 + 時，則把 CP 那一行及以下 n-1 行資料印出來。若是 - n 時則把 CP 以

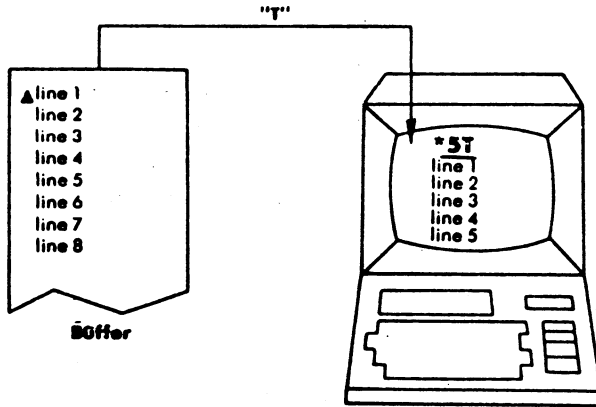
前 n 行資料印出來，在印資料的時候按下“break”鍵可以中止印資料的動作。

實例(1):



使用“T”命令來顯示文稿資料

實例(2):



nT 命令

$\pm n < cr >$  相當於  $\pm nLT$ ，將 CP 移上 ( - ) 或移下 ( + )  
 $n$  行後印出那一行資料。

## 命令字串

操作員可以連續打入任意數目命令 ( 但受限於 CP/M 控制台緩衝區的容量 )，直到打入  $< cr >$  後，這些命令才被執行。因此操作員在未打入  $< cr >$  以前可以用控制台所提供的命令來修改所輸入的命令。

- Rubout 移去前一個字元。
- Control - X 刪去整行。
- Control - C 重新啓動 CP/M 系統。
- Control - E 使游標回到那行的開端，但不將資料傳送到緩衝區去 ( 緩衝區最多容納 128 拜 )。

假設現在記憶體緩衝區中含有前一節圖 4 所示之字元，而 CP 位在緩衝區最後一個字元之後，以下的命令所產生的結果如下：

命令字串	結果	記憶體緩衝區的現時狀況
B2T<cr>	CP 移到緩衝區的起始端，並印出兩行資料	<pre> NOW IS THE&lt;cr&gt;&lt;lf&gt; TIME FOR&lt;cr&gt;&lt;lf&gt; ALL GOOD MEN&lt;cr&gt;&lt;lf&gt; </pre>
	“ NOW IS THE TIME FOR ”	
5C0T<cr>	CP 移動 5 個字元的位置，並把這行從頭起到 CP 為	<pre> NOW I AS THE&lt;cr&gt;&lt;lf&gt; </pre>

命令字串	結果	記憶體緩衝區的時現狀況
	止的資料“NOW I”印出來	
2L-T<cr>	CP 先移到本行的起始端 然後向下移二行，並印出 前一行的資料“TIME FOR”	NOW IS THE<cr><lf> TIME FOR<cr><lf> ⚡ALL GOOD MEN<cr><lf>
-L*K<cr>	CP 向上移一行，並把以 下的 65535 行全部去掉。	NOW IS THE<cr><lf>⚡
I<cr> TIME TO<cr> INSERT<cr> fz	插入二行文稿資料。	NOW IS THE<cr><lf> TIME TO<cr><lf> INSERT<cr><lf>⚡
-2L*T<cr>	CP 向上移二行，把 CP 之後 65535 行資料 “TIME TO INSERT ”印出來	NOW IS THE<cr><lf>⚡ TIME TO<cr><lf> INSERT<cr><lf>
<cr>	CP 向下移一行，並印出 一行資料“INSERT”	NOW IS THE<cr><lf> TIME TO<cr><lf>⚡ INSERT<cr><lf>

### 文稿之搜尋與變更

ED 有一種指令可以定出記憶體緩衝區內一串文字的位置。這一命令的形式如下：

$$nF c_1c_2 \dots c_k \left\{ \begin{matrix} \langle cr \rangle \\ \uparrow Z \end{matrix} \right\}$$

$C_1C_2 \dots C_k$  是想要找尋的字串，這指令以  $\langle cr \rangle$  或  $\uparrow Z$  作結束，若是以  $\uparrow Z$  作結束， $\uparrow Z$  之後可以跟著其他命令。當這命令被執行時，ED 從 CP 現在的位置起向下尋找和  $C_1C_2 \dots C_k$  全部相同的字串，連續做到第  $n$  次，如果成功地找到，CP 就移到  $C_k$  的後面，若是不成功，則 CP 的位置不改變，要尋的字串  $C_1C_2 \dots C_k$  中可以包括  $\uparrow l$  ( control - l )。實際上  $\uparrow l$  代表  $\langle cr \rangle \langle lf \rangle$ 。

下面是 F 命令的一些例子：

命令字串	功 能	記 憶 體 緩 衝 區
B#T<cr>	CP 移到最前端，並印出緩衝區內全部資料。	⚡ NOW IS THE <cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
FST<cr>	找尋字串 S T 並將 CP 放於其後。	NOW IS T⚡HE<cr><lf>
FI↑zOTT	找到下一個“ I ”並把它所在那一行印出來 “ TIME FOR ”	NOW IS THE<cr><lf> TI⚡ME FOR<cr><lf> ALL GOOD MEN<cr><lf>

另一種縮寫的插入命令 ( I ，代表 Insert ) ，常常和 F 命令一起使用做一些簡單的文字上的改變。其形式如下：

$$I c_1c_2 \dots c_n \uparrow z \text{ or } I c_1c_2 \dots c_n \langle cr \rangle$$

$C_1 C_2 \dots C_n$  是要插入的文字，如果插入的字串以 ↑Z 作為結束的話，那麼  $C_1 C_2 \dots C_n$  從 CP 之後插入，並且 CP 位於  $C_n$  之後，若是以 < cr > 作結束其情形和 ↑Z 一樣，唯一不同的是：在  $C_n$  之後，系統自動加上 < cr >, < lf >。

下面是 F 和 I 命令的例子：

(1) 命令字串： BTHIS IS ↑z<cr>

功 能：插入“THIS IS”於最前端。

記憶體緩衝區：  
 THIS IS ▲ NOW THE<cr><lf>  
 TIME FOR <cr><lf>  
 ALL GOOD MEN<cr><lf>

(2) 命令字串： FTIME↑z-4DIPLACE↑z<cr>

功 能：找尋“TIME”並將之刪除掉然後插入“PLACE”

記憶體緩衝區：  
 THIS IS NOW THE<cr><lf>  
 PLACE ▲ FOR<cr><lf>  
 ALL GOOD MEN<cr><lf>

(3) 命令字串： 3FO↑z-3D5DICHANGES↑<cr>

功 能：找第三個出現的“O”（即GOOD的第二個O）去掉前三個及後5個字，然後插入“CHANGES”。

記憶體緩衝區：  
 THIS IS NOW THE<cr><lf>  
 PLACE FOR<cr><lf>  
 ALL CHANGES ▲ <cr><lf>

(4) 命令字串： -8CISOURCE<cr>

功 能：CP 往回移 8 個字元的位置然後插入“SOURCE  
 < cr > < lf >”。

記憶體緩衝區：  
 THIS IS NOW THE<cr><lf>  
 PLACE FOR<cr><lf>  
 ALL SOURCE<cr><lf>  
 ▲ CHANGES<cr><lf>

ED 還提供了另一種命令，它結合了 F 及 I 命令來做字串的替代

$$n S c_1 c_2 \dots c_k \{z d_1 d_2 \dots d_m \left\{ \begin{array}{l} \langle cr \rangle \\ \{z \end{array} \right\}$$

此命令相當於

$$F c_1 c_2 \dots c_k \{z -k D d_1 d_2 \dots d_m \left\{ \begin{array}{l} \langle cr \rangle \\ \{z \end{array} \right\}$$

執行  $n$  次。換句話說，ED 從記憶體緩衝區內 CP 現在的位置起，搜尋到  $C_1 C_2 \dots C_k$  後並以  $d_1 d_2 \dots d_m$  字串代替之，直到  $n$  次執行完畢或是到了緩衝區末端才停止。

爲了使用者方便，ED 還提供一種類似 F 的命令

$$n N c_1 c_2 \dots c_k \left\{ \begin{array}{l} \langle cr \rangle \\ \{z \end{array} \right\}$$

這命令會搜尋整個源檔，找出第  $n$  次出現的  $C_1 C_2 \dots C_k$  字串（注意！F 指令只能在記憶體緩衝區中尋找）。N 指令和 F 指令幾乎都相同，只有當在記憶體緩衝區中找不到這字串時才不同。這時，#W 指令會自動執行，把緩衝區資料全部寫入暫存檔中，然後從源檔中再讀入資料使得緩衝區達到半滿，或是源檔已全部被讀入爲止。如上面所述，搜尋字串的行動繼續進行，直到成功或是全部的源檔案都已傳送到暫存檔案中才停止。

最後再介紹一種並列命令，其形式如下：

$$n J c_1 c_2 \dots c_k \{z d_1 d_2 \dots d_m \{z e_1 e_2 \dots e_q \left\{ \begin{array}{l} \langle cr \rangle \\ \{z \end{array} \right\}$$

這指令將執行  $n$  次以下所述的動作：從現在 CP 位置起，搜尋  $C_1 C_2 \dots C_k$  字串，找到以後插入  $d_1 d_2 \dots d_m$  字串，CP 位於  $d_m$  之後，然後從 CP 起刪掉所有文字直到  $e_1 e_2 \dots e_q$  字串（不刪去這字串，CP 仍然位於  $d_m$  之後。又若  $e_1 e_2 \dots e_q$  字串找不到，就不執行刪除的工作）出現爲止。

例如：現在有一行資料  $\uparrow$ NOW IS THE TIME<cr>

命令字串 JW t2WHATt2fl<cr>

執行結果是

NOW WHAT  $\uparrow$ <cr><lf>

ED 的 F, S, N, J 命令字串中所含的字元數目不能超過 100 個。

## 源檔叢庫 ( Source Libraries )

ED 允許在編輯過程中，以 R 指令把源檔案叢庫包括進來。R 指令的形式為：

R f<sub>1</sub>f<sub>2</sub>...f<sub>n</sub>t2 或

R f<sub>1</sub>f<sub>2</sub>...f<sub>n</sub><cr>

f<sub>1</sub>, f<sub>2</sub>, …… f<sub>n</sub> 為磁碟中的源檔檔名，它們的檔案形式為“LIB”。ED 把這些檔案讀入，從 CP 之後插入於記憶體緩衝區中。例如，當操作員打入 RMACRO < cr > 命令，ED 就把檔案 MACRO.LIB 全部讀入，插入記憶體緩衝區中。

## 重覆命令的執行

巨集命令 M，允許使用者把許多 ED 命令集合起來，重覆執行。

M 命令形式為：

$$n M c_1 c_2 \dots c_k \left\{ \begin{array}{l} \langle cr \rangle \\ t2 \end{array} \right\}$$

C<sub>1</sub>, C<sub>2</sub>, …… C<sub>k</sub> 是一串 ED 命令，但其中不得包含另一個 M 命令。若 n > 1，M 命令被 ED 執行 n 次，若 n = 0 或 1 那麼這串命令被重覆執行直到錯誤發生（如，執行 F 命令到記憶體緩衝區的末端時）。下面例子的巨集指令把記憶體緩衝區內所有的 GAMMA 改為 DELTA 並且把改變過的每一行印出來。



MFGAMMA ↑ z - 5D IDELTA ↑ z 0 TT < cr >

相當於

MSGAMMA ↑ z DELTA ↑ z 0 TT < cr >

## ED錯誤狀況

發生錯誤的時候，ED 會把在錯誤發生前的最後一個字元以及錯誤指示印出來。錯誤指示有：

- ? 不認識的命令。
- > 記憶體緩衝區已滿（這時操作員應用 D，K，N，S 或 W 命令來除去某些資料），或是 F，N，S 命令字串太長。
- # 無法執行完指令所欲做的次數（如在 F 命令時）。
- O 使用 R 命令但無法開啓 LIB 檔案。

CP/M 每次都把 (CRC) Cyclic redundancy check 訊息附寫於輸出的資料錄，以便在隨後的讀入動作偵測是否有錯誤發生。當有錯誤發生時，CP/M 會印出

PERM ERR DISK d

d 是現在正在使用的磁碟，這時操作員只需在控制台上敲入任何字元，系統就不理會這個錯誤（當然，最好去檢查一下記憶體緩衝區中的資料是否讀錯了）。或者使用者也可以重新啓動系統，把備用檔拿出來用。首先我們把 BAK 檔案從螢光幕上印出來，看是否含有正確的訊息：

TYPE x.BAK<cr>

其中 x 是要編輯的檔，然後把原來的檔案清掉：

```
ERA x.y<cr>
```

最後把 BAK 檔案重新命名：

```
REN x.y-x.BAK<cr>
```

我們就又可以開始編輯這個檔案了。

### 控制字元摘要

↑c	系統重新起動
↑e	實際上的 < cr > < lf > ( 在命令中不必加入 )
↑i	邏輯的定位符號 < tab > ( 第 1, 8, 15 ... 行 )
↑l	在搜尋及替代型命令中的邏輯 < cr > < lf >
↑x	刪掉一行
↑z	字串結束
rubout	刪掉字元
break	停止命令的執行 ( 譬如, 停止打出資料 )

### ED命令摘要

命令	作用
nA	從源檔附加 ( append ) n 行到緩衝區內。
±B	CP 移到緩衝區的最前端或最底端
±nC	從 CP 開始向前 ( 或後 ) 移動 n 個字元。
±nD	刪去 n 個字元 ( delete )。
	結束編輯並關閉檔案 ( end ), 此時緩衝區之資料已存

244 Apple II 軟體卡

入磁碟中。

nF 尋找字串 ( find )

H 結束編輯關閉檔案然後再重新進入編輯程式中。

I 插入字元 ( Insert )。

nJ 搜索、取代及刪除 ( Juxtaposition ) ; 有三個字串

附屬於此命令之後。

±nK 去除數行 ( kill )。

±nL CP 往上或下移動 n 行 ( line )。

nM 巨集命令 ( macro )。

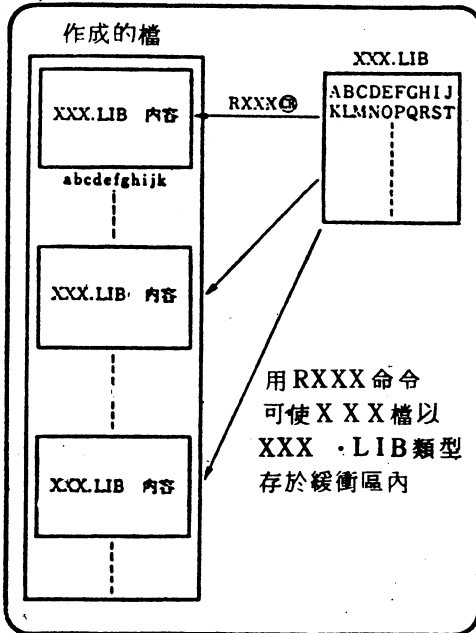
nN 找尋字串並自動掃描整個檔案。

O 回到原來的檔案 ( original file )。

±nP 從 CP 開始的 ±n 頁 ( 1 頁有 24 行 ) 從控制台印出。

Q 不改變源檔，停止執行 ED，回到 CP / M ( quit )。

R 讀源檔叢庫。



nS 替代字串 ( substitute )。

±nT 從 CP 開始印出 ( typeout ) ±n 行。

-U + U 把小寫字母改爲大寫字母，- U 則不做這種轉換。

nW 把緩衝區內之文稿寫入 ( write ) n 行到暫存檔去保存起來。

±nX 把 n 行資料傳送到 X.LIB 暫存檔中。

±nZ 暫停。

±n< cr > 從 CP 開始移動 ±n 行並印出該行資料。( 即 ± nLT )。

## ED文稿編輯命令

ED 文稿編輯程式還有一些命令來增強其編輯能力，包括增加行數，詢問可用的空間以及較詳盡的錯誤報告等。當使用者使用 V 命令，ED 編輯程式會在每行開端把絕對行數顯示出來。

nnnnn:

nnnnn 是絕對行數，它的範圍由 1 到 65535，如果記憶體緩衝區是空的或是已到了記憶體緩衝區的末端，nnnnn 就出現 5 個空白。

使用者若想參考到某一行 ( 絕對行數 ) 只要在任一種命令前打入數字及冒號即可，其格式和上面所述的相同。只要這一行現存在記憶體緩衝區內，ED 就把現在的行數指標移到所給的這個值 ( 絕對值 ) 以下舉例加以說明：當操作員打入 345 : T 表示把行數指標移到第 345 行 ( 絕對值 )，並把此行列印出來。這裏所給的絕對行數，只有在編輯過程產生，並不會記錄到檔案中。

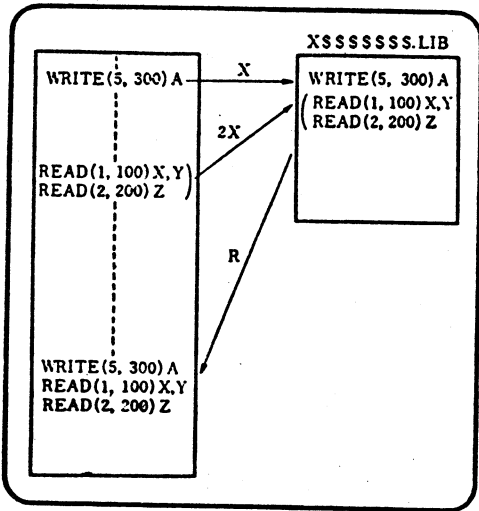
: 400 T 表示把從現在這行起到第 400 行的資料都印出來。同樣的 345 : : 400 T 表示把第 345 行到第 400 行的資料都印出來。另外還有一個 0V 命令，它有特別用途，顯示出記憶體的統計資料如下：

```

free/total
*0U
18488/39647
*
    
```

其中 free 表示在記憶體緩衝區內尚未使用的拜 (byte) 數量 ( 為十進位值 ) total則代表緩衝區內全部擁有的拜數量。

ED 還有一個具有傳送整區資料 ( block move ) 能力的 “ X ” 命令。 nX 是表示目前那一行以後的 n 行資料傳送到 X\$\$\$\$\$\$\$. LIB 的暫存檔中, 此檔只有在編輯過程中才存在。使用者可以用 V 命令把目前的行數指標移到源檔內的任一行去, 再以 nX 命令把所要的部分資料傳送到暫存檔。 nX 命令傳送過去的資料, 就會一個接一個地累積起來。

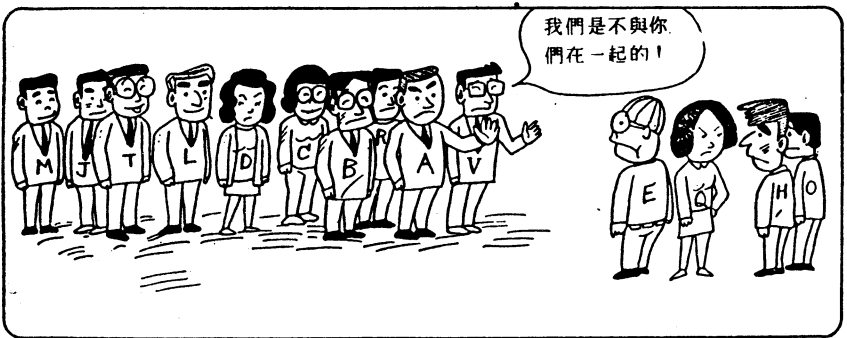


只要使用者輸入 R 命令, 那麼被傳送過的資料就會全部被讀入記憶體緩衝區。有一點要注意的是 R 命令並未把暫存檔內以 X 指令傳送

過去的資料清除掉。換句話說這些資料可再被讀。用 O X 命令則可以把這些資料清除掉。

一般情形下，使用者以 Q 或 E 命令結束編輯時，ED 會把暫存的 .LIB 檔案清除掉，不過若是用 control -C 非正常地結束 ED，則 .LIB 檔案仍然會存在。但是當使用者再次叫出 ED 程式時，系統自動會把這個暫存檔清除掉。

爲了防止一般使用者輸入命令時不小心打錯，ED 程式規定有幾個容易導致嚴重傷害的命令只能單獨使用。E (end), H (head), O (original), Q (quit) 等一次只能打入一個單獨字母。



注意：H，O，Q，E 命令不能與其它命令複合使用

當編輯發生錯誤時，ED 會把錯誤訊息以下列的格式印出：

```
BREAK "x" AT c
```

x 是發生錯誤的字元，c 是發生錯誤的命令。

以下我們將舉一較完整的例子來說明 ED 之用法，此例分 3 部份，第 1 部份爲所要編輯的程式；第 2、3 部份則爲編輯之範例：

## 第 1 部份：源文稿

```

A>ED FIB.PLI /
  1: ##A/      把檔所有資料附加到記憶體緩衝區內。
  1: ##T/      "T" 命令與 "A" 記號合用，可把所有的行打出 ~*~
  1: fibonacci;
  2:   proc options(main);
  3:     dcl i fixed;
  4:     do i = 0 to 100;
  5:       put list(fib(i));
  6:     end;
  7:
  8:     fib;
  9:       proc(n) returns(fixed) recursive;
 10:         dcl n fixed;
 11:         if n = 0 then
 12:           return(1);
 13:         if n = 1 then
 14:           return(1);
 15:         return(fib(n-1) + fib(n-2));
 16:       end fib;
 17:     end fibonacci;
 1: *

```

## 第2部份：文稿之編輯

```

1: s9:|----- 把CP放於第9行之前
9: s1|----- 從CP開始印出該資料
      proc(n) returns(fixed) recursive;
9: s5:|----- CP放在第5行前頭，打出該行資料(用1行的命令來記述)
      put list(fib(1));
5: s3||----- CP前進3行，並置於行前頭
8: s1|----- 從CP開始印出該行資料
      fib;
8: s-4|----- CP後退3行，並置於行前頭
4: s1|----- 印出該行
      do i = 0 to 100;
4: s7:|----- CP前進7行並把該行印出
      if n = 0 then
11: s4|----- CP再前進4行，並印出該行
      return(fib(n-1) + fib(n-2));
15: s-2|----- CP後退2行，並把該行印出
      if n = 1 then
13: s1|----- 印出下一行
      return(1);
14: s1|----- 同上
15: s1|----- return(fib(n-1) + fib(n-2));
15: s1|----- 同上
16: s1|----- end fib;
16: s1|----- 同上
17: end fibonacci;
17: s1|----- 打出第5行，CP在該行前頭
1: s5:|----- ライン5をタイプ。CPはライン5の先頭。
5: put list(fib(1));
5: s^list^Z^ABCDEFG|----- Sと^Zにより "list" を "ABCDEFG" に置き換
5: s0|----- 此命令使CP跑到行之前頭(OLT命令之縮寫)
      put ABCDEF(fib(1));
5: s^fib^712345^Z0:|----- 同様に，以 "1 2 3 4 5" 取代 "fib"，並打出該行
      put ABCDEF(12345(1));
5: s-1|----- 退後一行並打出該行資料
                                                    カコマンドでも可也
4: do i = 0 to 100;
4: s0 to^Z|----- S和↑Z合用，刪除 "0 to" 字串
4: s0|----- 把CP拉回前頭，並印出該行
      do i = 100;
4: s10:|----- 把CP置於第10行並打出該行資料
10: decl n fixed;
10: s1|----- CP所在那行刪除
10: s8:|2t|----- 打出第8~12行，CP置於第8行前頭
      fib;
9: proc(n) returns(fixed) recursive;
10: if n = 0 then
11: return(1);
12: if n = 1 then
8: s10:|----- 把CP置放第10行
10: s3kt|----- 把含CP所在那行在內共3行資料刪除，打出刪除下一行資料
      return(1);
10: s8:|t|----- 從第8行到最後的2行通通打出
      fib;
9: proc(n) returns(fixed) recursive;
10: return(1);
11: return(fib(n-1) + fib(n-2));
12: end fib;
13: end fibonacci;
8: s8:|t|----- CP到編輯程式緩衝區之前端，印出檔所有資料
    
```

(看看刪除後結果如何)

(刪除後ラインの縮寫)



## 第 2 部份(續)：文稿之編輯

```

1: fibonacci;
2:   proc options(main);
3:     dcl i fixed;
4:     do i = 100;
5:       put ABCDEFG(12345(i));
6:     end;
7:
8:     fib;
9:     proc(n) returns(fixed) recursive;
10:      return(1);
11:      return(fib(n-1) + fib(n-2));
12:    end fib;
13:   end fibonacci;

```

(到現在的結果

```

1: $ffib^Zt/ ---- 搜索CP以後之字串 "fib", 把CP置於其後, 然後把CP之後到行終了之資料E
onacc;
1: $3ffib^Zt/ ---- 從CP開始搜索第 3 次出現 "fib" 然後執行與上列相同之動作
(n-2));
11: $b/ ----- CP回到編輯式緩衝區之最前頭
1: $mfib^Zott/ ---- 搜索編輯程式緩衝區內CP之後所有的 "fib" 並把它印出(巨集命令)
1: fibonacci;
8:   fib;
11:   return(fib(n-1) + fib(n-2)); --- (該行第一個fib之故
11:   return(fib(n-1) + fib(n-2)); --- 該行第二個 "fib" 之故
12: end fib;
13: end fibonacci;

```

BREAK "# AT ^Z ---- 因為到 3 buffer 最後, 所以 break 出現

```

13: $b/ ---- CP 回到 editor buffer 之最前頭
1: $mfib^ZFIB^Zott/ ---- CP 之後所有的字串 "fib" 都換成 "FIB" 並印出(巨集命令)
1: FIBonacci;
8:   FIB;
11:   return(FIB(n-1) + FIB(n-2));
11:   return(FIB(n-1) + FIB(n-2));
12: end FIB;
13: end FIBonacci;

```

和巨集命令上之源程式比較看看

BREAK "# AT ^Z ---- 因為再也看不到資料了所以 break 出現

```

13: $11st/ ---- CP 置於第 11 行並打出該行資料
11:   return(FIB(n-1) + FIB(n-2));
11: $i/ ---- 進入 insert 模式
11: INSERT THIS LINE NOW //
12: CP/M Learning System -I- // } 打入此兩行
13: ^Z ---- 打入 ^Z, 終止 insert mode
13: $bott/ ---- 把 CP 放回到編輯程式緩衝區之前頭, 打入全部檔案資料
1: FIBonacci;
2:   proc options(main);
3:     dcl i fixed;
4:     do i = 100;
5:       put ABCDEFG(12345(i));
6:     end;
7:
8:     FIB;
9:     proc(n) returns(fixed) recursive;
10:      return(1);
11:      INSERT THIS LINE NOW !
12:      CP/M Learning System -I-
13:      return(FIB(n-1) + FIB(n-2));
14:    end FIB;
15:   end FIBonacci;
1: $e/ ---- 把編輯好的內容存到磁碟上去, ED 終了

```

最後的文稿

A> ---- 回到 CP/M

第3部份

```

A>ED FIB.PLI/
1: $0v/----- 在編輯程式緩衝區中，空白地區緩衝區的字數/全部緩衝區的字數
27574/27575 並由此來看CP/M，緩衝區大小之增減(此適用於48K CP/M場合)
1: $8a/----- 繼所有的資料都附加到編輯程式緩衝區上去
1: $0v/----- 再來查看CP/M記憶區大小
27258/27575
1: $7c/----- 從CP那一行起打出7行
1: Fibonacci
2: proc options(main);
3:   dcl i fixed;
4:   do i = 100;
5:     put ABCDEF8(12345(1));
6:   end;
7:
1: $7x/----- 從CP那一行起存7行到磁碟上去(CP不變)
1: $7kt/----- 從緩衝區上刪掉此7行，並打出一行資料
1: FIB;
1: $8t/----- 把CP後所有資料全部打出 !
1: FIB;
2: proc(n) returns(fixed) recursive;
3:   return(1);
4: INSERT THIS LINE NOW !
5: CP/M Learning System -I-
6:   return(FIB(n-1) + FIB(n-2));
7:   end FIB;
8: end Fibonacci;
1: $-b/----- 把CP後附加資料之最後一行
1: $7l/----- 以前所存到磁碟的資料插於CP之後
1: $8t/----- 把編輯程式緩衝區內所有資料打出
1: FIB;
2: proc(n) returns(fixed) recursive;
3:   return(1);
4: INSERT THIS LINE NOW !
5: CP/M Learning System -I-
6:   return(FIB(n-1) + FIB(n-2));
7:   end FIB;
8: end Fibonacci;
9: Fibonacci;
10: proc options(main);
11:   dcl i fixed;
12:   do i = 100;
13:     put ABCDEF8(12345(1));
14:   end;
15:
1: $9t/----- 把CP放在第9行前，並打出該行
9: Fibonacci;
9: $rFILEMAKE/ - 在CP之後插入從磁碟送入之"FILEMAKE.LIB" 檔資料
17: $8t/----- 打出編輯程式緩衝區中所有資料
1: FIB;
2: proc(n) returns(fixed) recursive;
3:   return(1);
4: INSERT THIS LINE NOW !
5: CP/M Learning System -I-
6:   return(FIB(n-1) + FIB(n-2));
7:   end FIB;
8: end Fibonacci;
9: CAN'T HELP FALLING IN LOVE
10:
11: Wise men say only fools rush in
12: But I can't help falling love with you
13: Shall I stay would it be a sin
14: If I can't help falling love with you
15:
16: . . . . . song by E.PREGLE
17: Fibonacci;
18: proc options(main);
19:   dcl i fixed;
20:   do i = 100;
21:     put ABCDEF8(12345(1));
22:   end;
23:
1: $8/----- 把修改內容存到磁碟上去，ED終了

```

由此可看到有7行被刪除了

代入後之結果

如此把3個程式編在一起



## 第 四 章

# CP/M組合語言編譯程式

- 簡介
- 程式格式
- 運算元 ( OPERAND ) 之形成
- 標記
  - 數字常數
  - 保留字
  - 字串常數
  - 算術及邏輯運算子
  - 運算子之優先順序
- 組合語言編譯程式指令
  - ORG 指令
  - END 指令
  - EQU 指令
  - SET 指令
  - IF 及 ENDIF 指令
  - DB 指令
  - DW 指令
- 運算碼

## 254 Apple II 軟體卡

跳離，呼叫與還原指令

立即指令

增減指令

資料移動指令

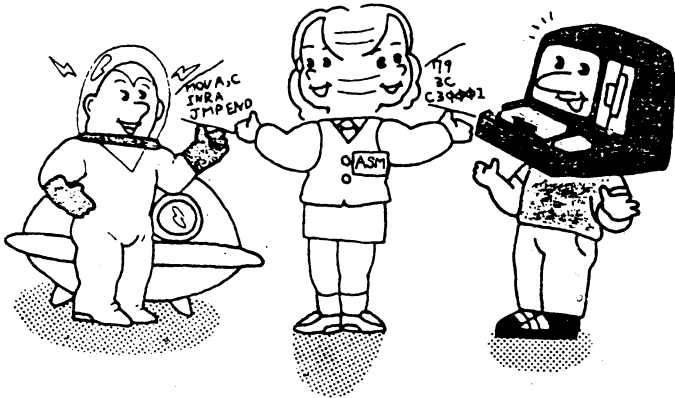
算術邏輯單位運算

控制指令

- 錯誤訊息

- 範例

## 簡介



CP/M 組合語言編譯程式從磁碟讀入組合語言源檔，然後產生 Intel 8080 16 進位機器語言。當使用者打入

ASM filename 或 | ASM filename.parms

CP/M 的組合語言編譯程式就被起動，以上兩種型式都假設磁碟存在一名叫 filename.ASM 的 8080 組合語言之源檔。而它們的差別在於第二種型式可以傳送參數給組合語言編譯程式去控制源檔的存取以及以 16 進位格式印出檔案內容到指定的地方。當 CP/M 組合語言編譯程式載入系統後，即印出下列訊息：

CP/M ASSEMBLER VER n.n

n.n 是現在的版本號碼，操作員打入的指令若沒有參數的話，編譯程式先讀入源檔，然後產生兩個輸出檔 filename.HEX 與 filen-

ame.PRN。“HEX”檔案含有源程式的 Intel 16 進位格式的機器語言。“PRN”檔案則含有源資料、機器語言及錯誤訊息等。如果在翻譯時發生錯誤的話，這些錯誤的訊息不但出現在控制台上，也會記在“PRN”檔案裏。

在第二種形式的指令中，參數部分是由三個字元所組成的，用以指明源檔案的出處以及“HEX”檔和“PRN”檔的目的地。

詳細的格式為：

filename.plp2p3

P1 : A, B, …… , Y	指定源檔案位於那個磁碟上。
P2 : A, B, …… , Y	指定“HEX”檔要存放到那個磁碟上。
Z	不產生“HEX”檔案。
P3 : A, B, …… , Y	指定“PRN”檔存放到那個磁碟上。
X	“PRN”檔印出在控制台上。
Z	不產生“PRN”檔。

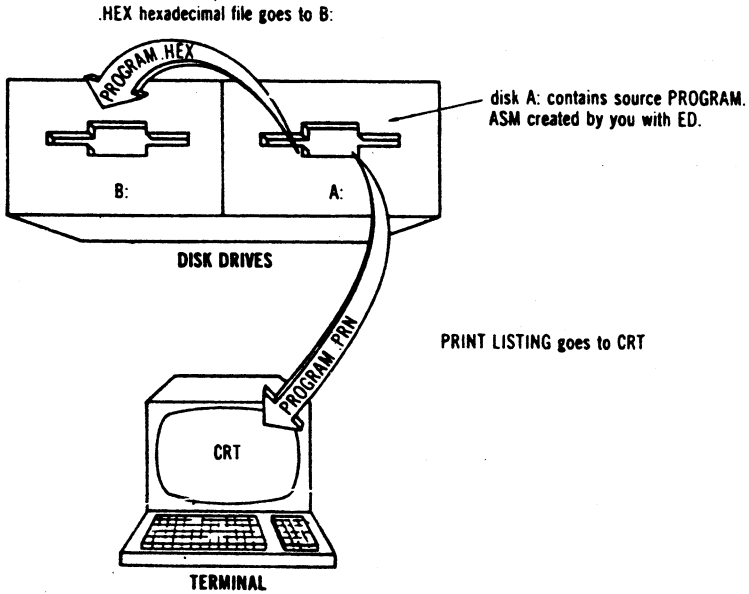
例如：ASM X.AAA 命令

表示源檔（名叫 X.ASM）由 A 磁碟中取得，所產生的“HEX”檔案（X.HEX）及“PRN”檔案（X.PRN）也放到 A 磁碟中。

如果操作員現在在 A 磁碟上執行組合語言編譯程式，那麼命令 ASM X 就相當於 ASM X.AAA。

同樣地：ASM X.ABX

意謂著源檔由 A 磁碟中取得，“HEX”檔案要存入 B 磁碟中，“PRN”檔則要印出在控制台上（如下圖）。



命令

ASM X.BZZ

則是源檔由 B 磁碟中取得，不產生“HEX”及“PRN”檔。(如果欲迅速確定程式語法正確否，此一命令可以加快組合語言編譯程式的執行速度)

這裏所談的源程式格式都和 Intel 8080 組合語言編譯程式(但其巨集指令在 CP/M 編譯程式上尚不能使用)以及 Processor Technology Software Package #1 組合語言編譯程式相配合。換句話說 CP/M 組合語言編譯程式能接受上述二種格式寫成的源程式。CP/M 另有一些擴充功能，使它更容易使用，以下將分別加以描述。



## 程式格式

能被 CP/M 組合語言編譯程式所接受的組合語言程式，是由連續的陳述組成，每個陳述的格式為：

line #	label	operation	operand	comment
行號	標記	運算	運算元	註釋

這些欄可以同時出現或只出現某幾欄。每個組合語言的陳述都以回車信號 < cr > 及饋行信號 < lf > 作為結束，( < lf > 是 ED 程式自動加入) 或是以 “!” 來結束，“!” 被組合語言編譯程式解釋為每行結束 (end-of-line) 的意思。(如此就可以把許多陳述寫在同一行內而用驚歎號 “!” 來分開這些陳述)。

註 < cr > : Carriage Return 之簡寫

< lf > : line feed 之簡寫

line # 是有無皆可的十進位整數值，代表源程式的行數，為了和 Processor Technology Software Package 互相配合，CP/M 組合語言編譯程式容許把它加在任意資料行。一般而言，使用行編輯程式 (line Editor) 建立源程式時，行號應鍵入以構建程式。但 ASM 根本不去管這一欄。

標記欄的格式為：

identifier            或            identifier:    (後者多一冒號)

這一欄有無皆可，只有一些特別的陳述才需要用到。identifier 是由連續的英文字母和數字組成，第一個字元必須為英文字母。程式

員可以隨意地使用它來標記程式步驟，組合語言編譯程式的指令等，但其長度不能超過 16 個字元，所有的字元都是有效字元，只有“\$”符號是用來增進可讀性（即ASM 遇到“\$”會把它當作空格）。而且所有的小寫字母都被當做大寫字母。在 identifier 後的“:”是可有可無的（為了使程式和 Intel 及 Processor Technology 相配合而設的）。

以下的例子都是符合規定的

```
x      xy      long$name
x:     yx1:    longer$name$data:
X1Y2  X1Y2    X234$5678$9012$3456:
```

運算 (operation) 欄中可以是一個組合語言的指令或是虛擬運算指令，或是 8080 機器運算碼。後面將對虛擬運算指令及 8080 運算碼加以描述。

每個陳述的運算元 (operand) 欄是由常數或標記以及算術或邏輯運算所組成。詳細情形也是在後面再加以解說。

註釋 (comment) 欄是從“;”符號一直到 end-of-line 截止，欄中可以寫入任何字元，這些字元僅僅是被組合語言編譯程式讀入並列印出來而已。此外沒有其他作用。

為了和 Processor Technology 相匹配，CP/M 組合語言編譯程式對於在第一行有“\*”的陳述，均認為是註釋陳述。注意，Processor Technology 組合語言編譯程式有對於超過運算元欄以後的所有字元都不予理會的情形。但是 Intel 的 Assembler 則無此情形，因此當我們希望所寫程式能和 Intel 語言互相匹配時，很容易發生混淆的情形。所以程式中在註釋欄前最好都加上“;”，編譯過程才能免於失誤。

組合語言程式最後以一個 END 陳述來結束，在 END 之後的陳述都將被組合語言編譯程式所忽略。

## 運算元之形成

爲了能詳盡地描述運算碼及虛擬運算，我們首先要了解運算元欄的組成。運算元欄包括了運算元（標記，常數及保留字）並配合由邏輯，算術運算子（operator）等合成的副表式（Subexpression）。組合語言編譯程式在編譯過程中對這些表式加以計算，每個表式經編譯後產生了一個16比次的數值，但是產生的數值不能超過所想運用的範圍。例如：表式使用拜移動立即指令時，此表式較高階的8個比次必須爲0，這種關於表式的限制因指令而異。

### 標記

標記是在某一特殊陳述中作辨識用的，通常根據它位在那種形式陳述之前定它一個數值。如果標記所在的陳述是產生機器碼或是保留記憶空間的話（如MOV 指令或DS 虛擬運算），則這個標記標定程式位址。若是標記位於EQU 或SET 指令之前，這個標記是運算元欄算出來的值。除了SET 指令之外，一個標記只能標定一個陳述。當標記出現在運算元欄時，組合語言編譯程式把它的數值代入和其他的運算元及運算子構成某一指令中的運算元欄。

### 數字常數

數字常數是16 比次的資料，可以用好幾種基底來表示，常數的基底是用一個尾隨的字母來表示。

B	二進位常數	( 基底為 2 )
O	八進位常數	( 基底為 8 )
Q	八進位常數	( 基底為 8 )
D	十進位常數	( 基底為 10 )
H	十六進位常數	( 基底為 16 )

因為O經常和 $\phi$ 混淆不清，所以另外也用Q來表示八進位數。如果數字常數沒有附加基底字母，就是十進位數。所以常數是由連續的數字和附隨的基底所組成。數字的範圍和基底有關。二進位常數，數碼為0和1，八進位則由0~7，十進位0~9，十六進位0~9，A(10D)，B(11D)，C(12D)，D(13D)，E(14D)，F(15D)。若是十六進位數，開頭第一個字必須為數字 $\phi$ ~9，以免和標記混淆(開頭增冠一個 $\phi$ 就可以了)，這裏所說的常數的值必須能以16比次來表示，若是超過的話，會被組合編譯程式從右邊切掉。“\$”符號在此也被用來增加可讀性。作為基底指示用的字母如果輸入時是小寫，會被轉變成大寫。

以下所舉的例子都合乎規定

```
1234    1234D    1100B    1111$0000$1111$0000B
1234H    OFFEH    3377O    33$77$22Q
3377o    OfE3h    1234d    0ffffh
```

## 保留字

有許多的保留字元串在一個陳述的運算元欄中，含有其事先規定的意義。如8080的暫存器名稱被使用到時，會轉換成一個數值，其對照表如下：

A	7
B	0
C	1
D	2
E	3
H	4
L	5
M	6
SP	6
PSW	6

(不論是大寫或是小寫字母，得到的值是一樣的)。

機器指令也可能出現在運算元欄中，它們會被轉換成內碼。如果這個指令需要有運算元，則運算元也會成為整個指令的二進位指令碼中的一部分。例如：MOV A, B 指令。MOV 的內碼為 40H (01000000B)，MOV A, B 的值則為 70H, 01 111 000 (參考 Intel 指令手冊)

當“\$”符號出現在運算元欄中(而非插在數字常數或插在 identifier 中時，則此數值代表下一個要產生的指令的位址。

### 字串常數

字串常數是一串 ASCII 字元，前後都以 ( ' ) 框住而成。字串中所有的字元必須在一行內(所以“!”字元也可出現在字串中)，長度不得超過 64 個字元。( ' ) 符號本身也可以出現在字串常數之中，但必須以 ( " ) 來表示(組合語言編譯程式讀入時，會把它當作 ' 看待)。大多數情形下，字串常數的長度限為一個或二個字元( DB

虛擬運算指令例外)，因此字串常數只是 8 比次或 16 比次的數值，至於其數值就是字串內字元的 ASCII 碼，如果有二個字元的話，第一個字元是高位拜，第二個字元為低位拜。在字串常數中，大、小寫是不同的。以下都是正確的例子：

```
'A'      'AB'    'ab'     'c'
'''      'a'''   '""'     '....'
'Walla Walla Wash.'
'She said "Hello" to me.'
'I said "Hello" to her.'
```

### 算術及邏輯運算子

在運算元欄中之運算子有：

$a + b$	a 和 b 之算術和。
$a - b$	a 和 b 之算術差。
$+ b$	單一加 (產生 b)
$- b$	單一減 (相等於 $0 - b$ )
$a * b$	a 乘以 b 之值
$a / b$	a 除以 b 之值
$a \text{MOD} b$	a / b 之餘數
NOT b	邏輯反相 (若 b 有 16 比次長，則其中所有的 1 變成 0，0 變成 1)。
a AND b	a “及” b (1 bit 對 1 bit 運算)
a OR b	a “或” b (1 bit 對 1 bit 運算)
a XOR b	a “互斥或” b (1 bit 對 1 bit 運算)
a SHL b	a 之值 (16 比次長) 向左移動 b 比次 (空下的地方補零)

a SHR b      a 之值 (16 比次長) 向右移動 b 比次 (空下的地方補零)

在上面的例子中，a 和 b 代表簡單的運算元 (標記，數字常數，保留字，字串常數等) 或是由括號框起來的表式。

例如：

```
10+20      10h+37Q      L1/3 (L2+4) SHR 3
('a' and 5fh) + '0'      ('B'+B) OR (PSW+M)
(1+(2+c)) shr (A-(B+1))
```

當組合語言編譯程式在編譯過程時，所有的計算都是無正負號的 16 比次運算，所以 -1 是當作 0-1 來計算，產生的值為  $\phi$  ffffh (即每個比次都為 1)，最終的表示法必須和原來的運算碼相配合，舉例來說，對一個 ADI 指令而言，它的高位 byte 必須都是 0，所以在執行“ADI-1”就會發生錯誤訊息，(因為 -1 為  $\phi$  ffffh，無法以 8 個比次來表示) 而“ADI(-1)AND  $\phi$  FFH”則不會發生錯誤，因邏輯“AND”運算使得高位 byte 全部變為  $\phi$ 。

### 運算子 之優先順序

為了方便程式設計，運算子在應用時有相對的優先順序，程式設計員在寫表式時，可以不必用許多層括號。下面列出不用括號時，各個運算子的優先順序，在上面者優先順序較高，在下面的優先順序較低。同一行者優先順序相同，在運算時依由左而右之順序執行。

```
* / MOD SHL SHR
- +
NOT
AND
OR XOR
```

由於運算子之間有相對之優先順序，組合語言編譯程式在讀入下列左邊的式子時，就將之視為和右邊有括號之式子一樣。

$a * b + c$	$(a * b) + c$
$a + b * c$	$a + (b * c)$
$a \text{ MOD } b * c \text{ SHL } d$	$((a \text{ MOD } b) * c) \text{ SHL } d$
$a \text{ OR } b \text{ AND NOT } c + d \text{ SHL } e$	$a \text{ OR } (b \text{ AND } (\text{NOT } (c + (d \text{ SHL } e))))$

我們可以用括號來迫使運算子不依照原來之優先順序運算，而產生不同的結果，譬如上面所舉的最後一個例子，我們若加上二對括號

於  $(a \text{ OR } b) \text{ AND } (\text{NOT } c) + d \text{ SHL } e$

結果變成

$(a \text{ OR } b) \text{ AND } ((\text{NOT } c) + (d \text{ SHL } e))$

## Assembler Directives

Assembler Directives 用在編譯過程中，設定標記為某一特定值或是執行條件式的編譯，或是定義儲存區及指定程式開始的位址等。每個 Assembler Directive 是以出現在每行的運算欄中的虛擬運算來表示。

虛擬運算有

ORG	程式或是資料的開始。
END	程式的結束，可附加程式開始位址於其後。
EQU	數字“等於”。
SET	數字“設定”。



IF	條件式編譯之開始
ENDIF	條件式編譯之結束
DB	定義資料拜
DW	定義資料字 (word)
DS	定義資料儲存區

## ORG Directive

```
label    ORG    expression
```

“label”是程式標記可有可無。expression是16 bit 長的表式，它可包含任何在ORG 指令之前已定義過之運算元。組合語言編譯程式從 expression所指定的位址開始產生機器碼。“expression”之值是任意的，計算機也不會去檢查是否有程式在那位址。

在CP/M 系統中絕大多數的程式都是以ORG 100 H開始，這個指令使得機器碼從CP/M 暫存程式區的底部開始產生。在ORG 陳述中，若有“Label”時，這標記以後就代表“expression”的值。（此一標記可能會在其他陳述中的運算元欄中出現，那時它將被當作“expression”的值加以運算）。

## END Directive

END 這個陳述在組合語言程式中是可有可無的，但是若要使用它，必須將它置於最後一個陳述。它有兩種形式

```
label    END
label    END    expression
```

label可有可無，若用第一種形式時，編譯過程執行到此一陳述時停

止，程式的起始位置被認定為 0000。若是用第二種形式則 `expression` 會被算出一個值，此一值即為程式的開始位址。（這一數值會被收入“HEX”檔案的最後一個資料錄裏），所以大多數的 CP/M 組合語言程式，結束時都是用

```
END 100H
```

使得程式的開始位址在暫態程式區。

### EQU Directive

EQU 是用來設定某一特定值的同義字。

```
label EQU expression
```

`label` 必須存在，而且不可標定過其他任何陳述。`expression` 會被計算出一個數值，以後 `label` 出現時就被視為這個數值。

例：程式將資料從某一輸入埠之打字機讀入然後送到下一個輸出埠所接的打字機上，EQU 陳述可以用來定義那些輸出入埠。

```
TTYBASE EQU 10H           ;BASE PORT NUMBER FOR TTY
TTYIN EQU TTYBASE        ;TTY DATA IN
TTYOUT EQU TTYBASE+1;TTY DATA OUT
```

在後面程式中做 TTY 的輸出入的陳述可能是這樣的：

```
IN TTYIN ;READ TTY DATA TO REG - A
...
OUT TTYOUT ;WRITE DATA TO TTY FROM REG-A
```

這樣的程式寫法比直接寫出 TTY 的輸出絕對位置，更具有可讀性。而且日後若是硬體結構的改變使 TTY 輸出入埠從 10H 改到 7FH 時，EQU 陳述只要改成下式即可：

```
TTYBASE EQU 7FH ;BASE PORT NUMBER FOR TTY
```

## SET Directive

SET 陳述和 EQU 很類似，只有一點不同，就是同一 label 的 SET 陳述 (label SET expression) 可以出現在同一程式裏其他各處。

expression 同樣地被算出一個值，然後 label 就被視為這個值，直到下一個 SET 陳述出現為止 (下一個 SET 陳述把 label 重定為另一個值) SET 陳述常用來控制條件式的編譯。

## IF及ENDIF Directive

IF 和 ENDIF 陳述用來定義編譯過程中，某一段組合語言是否要包括在內。

```
IF expression
statement #1
statement #2
...
statement #n
ENDIF
```

當組合語言編譯程式遇到 IF 陳述時，首先計算 IF 後面 expression 的值。如果此值不為 0，則陳述 #1 ~ #n 全部都需編譯，若是此值為 0，則陳述 #1 ~ #n 只被列印出來而不加以處理。條件式編譯通常是用在某類程式，此程式要處理很多可能發生的情形，而每一種情形只須用到程式中的某些特定部分而已。

下面用一個程式的片斷為例：

```

TRUE    EQU  0FFFFH    ;DEFINE VALUE OF TRUE
FALSE   EQU  NOT TRUE  ;DEFINE VALUE OF FALSE
;
TTY     EQU  TRUE      ;TRUE IF TTY, FALSE IF CRT
;
TTYBASE EQU  10H       ;BASE OF TTY I/O PORTS
CRTBASE EQU  20H       ;BASE OF CRT I/O PORTS
        IF  TTY        ;ASSEMBLE RELATIVE TO
                    TTYBASE
CONIN   EQU  TTYBASE    ;CONSOLE INPUT
CONOUT  EQU  TTYBASE+1;CONSOLE OUTPUT
        ENDIF
;
        IF  NOT TTY    ;ASSEMBLE RELATIVE TO
                    CRTBASE
CONIN   EQU  CRTBASE    ;CONSOLE INPUT
CONOUT  EQU  CRTBASE+1;CONSOLE OUTPUT
        ENDIF
;
        IN  CONIN      ;READ CONSOLE DATA
;
        OUT CONOUT     ;WRITE CONSOLE DATA

```

在上面例中，程式只編譯經由輸出入埠 10 H 相連 TTY 的程式部分。如果把定義 TTY 的陳述改為：

```
TTY    EQU  FALSE
```

則程式就只處理輸出入埠為 20H 和 CRT 有關的部分。

### DB Directive

程式設計員用 DB 陳述來定義以單一拜 (byte) 為單位的儲存區。

```
label  DB  e#1,e#2,...,e#n
```

e#1 到 e#n 都是 expression，最後皆計算得出 8 比次的數值 (高位 byte 為 0)，或者是 ASCII 字串 (不超過 64 字元)，在一行內 n 之大小並沒限制。這些表式數值或是 ASCII 字串，從程式最後位址起，一個一個 (字串則從第一個字元起，一個字元一個字元地

)依照順序放入記憶體中。若字串長度超過兩個字元，則不能在較複雜的表式中用作運算元，此時需以“ $C_1 \dots C_k$ ”之式表示。ASCII碼存在記憶體內時，其同位位元 ( parity bit ) 均被重置為(0)，這裏的ASCII小寫字母是不會自動翻譯成大寫字母的。label若有給則在程式中可以用此 label 來參考到資料區之位址。

例：

```
data:   DB  0,1,2,3,4,5
        DB  data and 0ffh,5,377Q,1+2+3+4
signon: DB  'please type your name',cr,lf,0
        DB  'AB' SHR 8,'C','DE' AND 7FH
```

## DW Directive

DW 陳述和DB 相類似，差別在DW 以一個字 ( word, 16bits ) 為儲存單位。

```
label  DW  e#1,e#2,...,e#n
```

e #1 到 e #n 這些 expression 被計算成 16 比次的數值，ASCII字串可以是 1 個或 2 個字元 ( 但不能超過二個以上 )。資料儲存方式和 8080 微處理機一樣，低位拜先放再放高位拜。

例：

```
doub:  DW  Offefh,doub+4,signon-$,255+255
        DW  'a',5,'ab','CD',6 shl 8 or 11b
```

## DS Directive

DS 陳述可用以保留一塊記憶區，格式如下：

```
label  DS  expression
```

label可有可無；DS 的作用是使編譯程式先跳過DS 所保留的記憶區後，才開始產生機器碼。

下列陳述相當於DS 陳述的作用

```
label: EQU $ ;LABEL VALUE IS CURRENT CODE LOCATION
        ORG $ + expression ;MOVE PAST RESERVED AREA
```

## 運算碼

組合語言運算碼是組合語言程式的主要部分，構成指令的運算欄部分。ASM 能接受所有 Intel 8080 微電腦的標準 mnemonics (在 Intel 8080 Assembly language Programming Manual 有詳盡之描述)，標記 (label) 是可有可無的，如果有用到它，那麼它的值就是這個指令的位址。在下面一節裏所有的運算子都簡要的列出來，若要詳細了解每個運算子的確實功能，必須參閱 Intel 手冊。

在每一個運算子中

- e3 表示一個 3 比次的數值，範圍從 0 ~ 7，可能是暫存器 A, B, C, D, E, H, L, M, SP 或 PSW 中之任一個。
- e8 代表一個 8 比次的數值，範圍由 0 ~ 255。
- e16 代表一個 16 比次的數值，範圍由 0 ~ 65535。

上面所提到的 e3，e8，e16 都可能由運算子和運算元任意組合而成。某些情況下，運算元的數值應限制在某一容許之特定範圍內 (如 PUSH 指令)，當使用到這些指令時，要注意到這種限制。

下面我們以最普遍的方式列出所有的運算碼，並舉一例予以說明。

## Jump, Call 及 Returns

Jump, Call 及 Returns 指令有許多形式；它可先測試CPU內旗號然後依測試結果做出動作：

JMB	e16	JMP	L1	無條件跳到標記 ( label ) 處
JNE	e16	JMP	L2	不等於 0 時跳到 ( label ) 處
JZ	e16	JMP	100H	等於 0 時跳到 ( label ) 處
JNC	e16	JNC	L1+4	無進位時跳到 ( label ) 處
JC		JC	L3	有進位時跳到 ( label ) 處
JPO	e16	JPO	\$+8	奇同位則跳到標記處
JPE	e16	JPE	L4	偶同位則跳到標記處
JP	e16	JP	GAMMA	結果為正時跳到標記處
JM	e16	JM	a1	結果為負時跳到標記處
CAL1	e16	CALL	S1	無條件的呼叫副程式
CNZ	e16	CALL	S2	不為零時呼叫副程式
CZ	e16	CALL	100H	為零時呼叫副程式
CNC	e16	CNC	S1+4	不進位時呼叫副程式
CC	e16	CC	S3	進位時呼叫副程式
CPO	e16	CPO	\$+8	奇同位時呼叫副程式
CPE	e16	CPE	S4	偶同位時呼叫副程式
CP	e16	CP	GAMMA	結果為正時呼叫副程式
CM	e16	CM	b1 \$ c2	結果為負時呼叫副程式
RST	e3	RST	0	程式控制“重啓動”
RET				從副程式中回到主程式
RNZ				不為零時從副程式中回到主程式

RZ	為零時從副程式中回到主程式
RNC	無進位時從副程式中回到主程式
RC	有進位時從副程式中回到主程式
RPO	奇同位時從副程式中回到主程式
RPE	偶同位時從副程式中回到主程式
RP	結果為正時從副程式中回到主程式
RM	結果為負時從副程式中回到主程式

### 立即運算元指令

許多指令可用來將常數資料載入暫存器或暫存器對內。也有此指令可以在累加器（即暫存器A內）執行算術及邏輯運算：

MVI e3, e8	MVI B, 255	把 e8 值搬入 A, B, C, D, E, H, L 或 M（記憶體）中
ADI e8	ADI 1	把值直接加到 A 暫存器中（不含進位）
ACI e8	ACI OFFH	把值直接加到 A 暫存器中（含進位）
SUI e8	SUI L+3	A 暫存器直接減去該值（不含借位）
SBI e8	SBI LAND11B	A 暫存器直接減去該值（含借位）
ANI e8	ANI \$AND7FH	將值和 A 暫存器直接做邏輯“及”運算
XBI e8	XRI 1111\$0000B	將值和 A 暫存器直接做邏輯“



互斥或”運算

ORI e8	ORI LAND1+1	將值和A暫存器直接做邏輯或運算
CPI e8	CPI “a”	將值和A暫存器直接比較
LXI e3,e16	LXI B,100H	將值直接放入一對暫存器對中 (e3 必須是B, D, H或SP)

增減指令

暫存器或暫存器對都有增減指令：

INR e3	INR E	將單一暫存器加1 ( e3 為A, B, C, D, E, H, L, M)
DCR e3	DCR A	將單一暫存器減1 ( e3 為A, B, C, D, E, H, L, M)
INX e3	INX SP	將暫存器對加1 ( e3 必須是B, D, H或 SP)
DCX e3	DCX B	將暫存器對減1 ( e3 必須是B, D, H或 SP)

資料移動指令

從CPU 搬資料到記憶體及從記憶體搬資料到CPU 之指令如下：

MOV e3,e3	MOV A,B	將資料從右邊暫存器 e <sup>3</sup> 移到左邊暫存器 e <sup>3</sup> ( e <sup>3</sup> 可以是
-----------	---------	--

LDAX e3	LDAX B	A, B, C, D, E, H, L 或 M 但 兩個 e3 不能同時為 M )
		將 e3 所代表之位址中的資料 搬入 A 暫存器中 ( e3 為 B 或 D )
STAX e3	STAX D	將 A 暫存器中之值存入 e3 所 代表之位址處。
LHLD e16	LHLD L1	將位址 e16 及 e16+1 中之資 料搬至 HL 暫存器對。
SHLD e16	SHLD L5+x	將 HL 暫存器中之值搬入位址 e16 及 e16+1 中
LDA e16	LDA Gamma	將位址 e16 中之資料搬入 A 暫 存器
STA e16	STA X3-5	將 A 暫存器中之值搬入位址 e16 處
POP e3	POP PSW	從 stack 最上頭取出資料搬 入成對之暫存器中 ( e3 必須 是 B, D, H 或 PSW )
PUSH e3	PUSH e3	將成對之暫存器中之值搬入 stack 中
IN e8	IN 0	將資料從埠 e8 搬至 A 暫存器 中
OUT e8	OUT 255	將 A 暫存器中之值送到埠 e8 處
XTHL		把 stack 最上面之資料和

			HL 暫存器互換
PCHL			將HL 暫存器之值放入程式計數器 (PC) 中
SPhL			將HL 暫存器之值放入 stack pointer 中
XCHG			將DE 暫存器及HL 暫存器中之值互換

### 算術邏輯單位運算

在累積器上可執行的算術及邏輯運算有：

ADD	e3	ADD	B	將暫存器 e3 ( e3 為 A, B, C, D, E, H 或 I ) 之值加到累積器中 ( 不含進位 )
ADC	e3	ADC	L	將暫存器 e3 ( 含進位, e3 同上 ) 之值加到累積器中
SUB	e3	SUB	H	從暫存器 A 中減去暫存器 e3 之值 ( 不含借位, e3 同上 )
SBB	e3	SBB	2	從暫存器 A 中減去暫存器 e3 之值 ( 含借位, e3 同上 )
ANA	e3	ANA	1+1	暫存器 e3 和暫存器 A 作邏輯 “及” 運算 ( e3 同上 )
XRA	e3	XRA	A	暫存器 e3 和暫存器 A 作邏輯 “互斥或” 運算 ( e3 同上 )
ORA	e3	ORA	B	暫存器 e3 和暫存器 A 作邏輯

				“或”運算 ( e3 同上 )
CMP	e3	CMP	H	暫存器 e3 和暫存器 A 作比較 ( e3 同上 )
DAA				將暫存器 A 調整為 10 進位數
CMA				取暫存器 A 之補數
STC				設定進位旗號為 1
CMC				取進位旗號補數
RLC				將 A 暫存器中每個比次向左轉 一位進位旗號被設為原暫存器 A 中最大的那個比次
RRC				將 A 暫存器中每個比次向右轉 一位進位旗號之值設為原暫存 器 A 中最小的那個比次之值
RAL				進位旗號和暫存器 A 合在一起 向左轉一個比次
RAR				進位旗號和暫存器 A 合在一起 向右轉一個比次
DAD	e3	DAD	B	將成對之暫存器加到 HL 暫存 器中 ( e3 必須是 B, D, H 或 SP )

### 控制指令

以下 4 個指令為控制指令：

HLT                    中止 8080 處理機

DI	關閉中斷功能
EI	起動中斷功能
NOP	無動作

## 錯誤訊息

當組合語言程式中發生錯誤時，會列出一個單一字母的錯誤碼。出現在源程式發生錯誤的那一行最左邊的位置。同時，發生錯誤的那行陳述也會顯示在控制台上。

錯誤碼的意義如下所述：

- D 資料錯誤：無法把資料陳述中的元素放入指定的資料區中。
- E 表式法錯誤：表式的組成錯誤，在編譯過程時無法計算。
- L 標記錯誤：標記不可出現於此（可能是標記重覆）。
- N 未具備：在將來的ASM版本才會具備的特點。
- O 溢位：表式的式子太複雜（太多運算子）無法計算。簡化之。
- P 相位錯誤：在程式中連續出現之相同標記，具有不同值。
- R 暫存器錯誤：指定為暫存器之值和運算碼不配合。
- V 數值錯誤：在表式中之運算元組成不正確。

在終端機的錯誤情況下則有下列數種錯誤指示：

NO SOURCE FILE PRESENT	ASM 命令所指之檔案並不存在磁碟中。
NO DIRECTORY SPACE	磁碟之檔案索引表已滿，應清除無用的檔案。
SOURCE FILE NAME ERROR	ASM 檔案名稱不合規定（例如以“？”來指名特定的檔）。
SOURCE FILE READ ERROR	編譯程式無法正確地讀入源檔，可用 TYPE 指令找出錯誤之處。
OUTPUT FILE WRITE ERROR	輸出檔案無法正確的寫出（很可能是磁碟已滿），清除無用檔案並重試。
CANNOT CLOSE FILE	輸出檔案無法關閉，查看是否磁碟為禁寫入。

### 範例

以下的範例顯示一種交談式的過程，說明在發展組合語言程式時如何使用系統的編輯程式及除錯程式：

# 280 Apple II 軟體卡

ASM SORT Assemble SORT. ASM

CP/M ASSEMBLER - VER 1.0

U13C next free address  
 003H USE FACTOR % of table used 00 to FF (hexadecimal)  
 END OF ASSEMBLY

DIR SORT =

SORT ASM source file  
 SORT BAK backup from last edit  
 SORT PRN prn file (contains tab characters)  
 SORT HEX machine code file  
 A>TYPE SORT PRN

Source line

machine code location	SORT PROGRAM IN CP/M ASSEMBLY LANGUAGE START AT THE BEGINNING OF THE TRANSIENT PROGRAM AREA
0100 ←	ORG 100H
0100 ← generated machine code	
0100 214601 ← SORT	LXI H, SW .ADDRESS SWITCH TOGGLE
0103 3601	MVI M, I .SET TO 1 FOR FIRST ITERATION
0105 214701	LXI H, I .ADDRESS INDEX
0100 3600	MVI H, 0 .I = 0
	COMPARE I WITH ARRAY SIZE
010A 7E COMP.	MOV A, M .A REGISTER = I
0100 FED9	CPI M-1 .CY SET IF I < (M-1)
0100 D21901	JNC CONT .CONTINUE IF I <= (M-2).
	END OF ONE PASS THROUGH DATA
0110 214601	LXI H, SW .CHECK FOR ZERO SWITCHES
0113 7E67C28001	MOV A, M; ORA A; JNZ SORT .END OF SORT IF SW=0
0110 FF-	RST 7 GO TO THE DEBUGGER INSTEAD OF RE-
	CONTINUE THIS PASS
0119 5F16002148CONT	ADDRESSING I, SO LOAD AV(I) INTO REGISTERS
0121 4E752346	MOV E, A; MVI B, 0; LXI H, AV; DAB D; DAB D
	MOV C, M; MOV A, C; INX H; MOV B, H
	LOW ORDER BYTE IN A AND C, HIGH ORDER BYTE IN B
0125 23	MOV H AND L TO ADDRESS AV(I+1)
	INX H
	COMPARE VALUE WITH RECS CONTAINING AV(I)
0126 965778239E	SUB M; MOV B, A; MOV A, B; INX H; SUB H .SUBTRACT
	BORROW SET IF AV(I+1) > AV(I)
0128 BA3F01	JC INCI .SKIP IF IN PROPER ORDER
	CHECK FOR EQUAL VALUES
012E B2CA3F01	ORA B; JZ INCI .SKIP IF AV(I) = AV(I+1)
0132 567A285E	MOV B, M; MOV M, B; DCX H; MOV E, H
0136 7128722873	MOV M, C; DCX H; MOV M, B; DCX H; MOV A, E
	INCREMENT SWITCH COUNT
013B 2646C134	LXI H, SW; INR H

← truncated

# 第四章 CP/M組合語言編輯程式 281

```

INCREMENT 1
013F 21478134C3INC1 LXI M,1 INR M! JMP COMP

DATA DEFINITION SECTION
0146 00 S DB 0 .RESERVE SPACE FOR SWITCH COUNT
0147 1 DS 1 .SPACE FOR INDEX
0148 050064001EHV DW 5,100,10,50,20,7,1000,100,100,102,67
000A * M EQU (B-04)*2 .COMPUTE M INSTEAD OF PPE
015C . equate value END
ATYPE SORT HEX

```

```

10010000214601360121478134007EF69D2190140
100110002146017E67C20001FF5F15002140011563
10012000194E79234623965778239EDA3F01B2CAA7
100130003F0156702B5E712B7220732146013421C
07014000470124C36A01006E
10014000050664001E00320014000700E0032C01B0
1001500064000100BE
000000000

```

} machine code in HEX format

```

A>DBT SORT,HEX start debug run

16K DDT,VER 1.0
NEXT FC
015C 0000 default address (no address on END statement)
-XP

```

```

P=0000 100 change PC to 100
-UFFFF untrace for 65535 steps

```

```

C020M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI H,0146
-T10 trace 10 steps
C020M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI H,0146
C020M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 MVI M,01
C020M0E010 A=01 B=0000 D=0000 H=0147 S=0100 P=0100 LXI H,0147
C020M0E010 A=01 B=0000 D=0000 H=0147 S=0100 P=0100 MVI M,00
C020M0E010 A=01 B=0000 D=0000 H=0147 S=0100 P=0100 MOV A,M
C020M0E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 CPI 09
C120M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 JMC 0119
C120M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 LXI H,0146
C120M1E010 A=00 B=0000 D=0000 H=0146 S=0100 P=0100 MOV A,M
C120M1E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 ORA A
C020M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 JNZ 0100
C020M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI H,0146
C020M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 MVI M,01
C020M0E010 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI H,0147
C020M0E010 A=01 B=0000 D=0000 H=0147 S=0100 P=0100 MVI M,00
C020M0E010 A=01 B=0000 D=0000 H=0147 S=0100 P=0100 MOV A,M=0106
-A10D

```

```

010D JC 119 change to a jump on carry
010D
stopped at 10BH
-XP

```

```

P=0100 100 reset program counter back to beginning of program
-T10 trace execution for 10H steps

```

```

C020M0E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 LXI H,0146
C020M0E010 A=00 B=0000 D=0000 H=0146 S=0100 P=0100 MVI M,01
C020M0E010 A=00 B=0000 D=0000 H=0146 S=0100 P=0100 LXI H,0147
C020M0E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 MVI M,00
C020M0E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 MOV A,M
C020M0E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 CPI 09
C120M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 JC 0119
C120M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 MOV C,M
C120M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 MVI C,00
C120M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 LXI H,0140
C120M1E010 A=00 B=0000 D=0000 H=0148 S=0100 P=0100 DAD D
C020M1E010 A=00 B=0000 D=0000 H=0148 S=0100 P=0100 DAD D

```

} altered instruction



# 282 Apple II 軟體卡

```

C020M1E010 A=00 B=0000 D=0000 H=0140 S=0100 P=0121 MOV C,M
C020M1E010 A=00 B=0005 D=0000 H=0140 S=0100 P=0122 MOV A,C
C020M1E010 A=05 B=0005 D=0000 H=0140 S=0100 P=0123 INX H
C020M1E010 A=05 B=0005 D=0000 H=0140 S=0100 P=0124 MOV B,M=0125
-L100

```

Automatic  
breakpoint

```

0100 LXI M,0146
0103 MVI M,01
0105 LXI M,0147
0108 MVI M,00
010A MOV A,M
010B CPI 09
010D JC 0115
0110 LXI M,0146
0113 MOV A,M
0114 ORA A
0115 JNZ 0100
-L

```

} list some code  
from 100H

```

0110 RST 07
0119 MOV E,A
011A MVI D,00
011C LXI M,0140

```

} list more

^ abort list with rubout

-G 110 start program from current PC (0125H) and run in real time to 11BH

=0127 stopped with an external interrupt 7 from front panel (program was looping indefinitely)  
-T4 look at looping program in trace mode

```

C020M0E010 A=30 B=0064 D=0006 H=0156 S=0100 P=0127 MOV B,A
C020M0E010 A=30 B=0064 D=3006 H=0156 S=0100 P=0128 MOV A,B
C020M0E010 A=00 B=0064 D=3006 H=0156 S=0100 P=0129 INX H
C020M0E010 A=00 B=0064 D=3006 H=0157 S=0100 P=012A SDB M=012B
-D140

```

data is sorted, but program doesn't stop

```

0140 05 00 07 00 14 00 1E 00
0150 32 00 64 00 64 00 2C 01 E0 03 01 00 00 00 00 00 2 0 3
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

-GO return to CP/M

00T SORT HEX reload the memory image

```

16X 00T VER 1.0
NEXT PC
015C 0000
-XP

```

P=0000 100 Set PC to beginning of program

-L100 list bad opcode

```

0100 JNC 0119
0110 LXI M,0146

```

^ abort list with rubout

-A100 assemble new opcode

```

0100 JC 119

```

0110

-L100 list starting section of program

```

0100 LXI M,0146
0103 MVI M,01
0105 LXI M,0147
0108 MVI M,00

```

^ abort list with rubout

^=A103 change switch initialization to 00

0103 MVI A,0

0165

-C return to CP/M with ctrl-c (GO works as well)

SAVE 1 SORT COM save 1 page (256 bytes, from 100H to 1FFMH) on disk in case we have to reload later

A) DDT SORT COM restart DDT with saved memory image

16X DDT VER 1 0

NEXT PC

0200 0100 "COM" file always starts with address 100H

-G run the program from PC=100H

\*0110 programmed stop (RST 7) encountered  
-D140

0140 05 00 07 00 14 00 1E 00  
0150 32 00 64 00 64 00 2C 01 EB 03 01 00 00 00 00 00 00 2 D D  
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

data properly sorted

- GO return to CP/M

ED SORT ASM make changes to original program

ctrl-Z

\*M,0 ZBT find next "0"

MVI A,0 .I = 0

\*- up one line in text

LXI H,1 .ADDRESS INDEX

\*- up another line

MVI A,1 .SET TO 1 FOR FIRST ITERATION

\*RT kill line and type next line

LXI H,1 .ADDRESS INDEX

\*I insert new line

MVI A,0 .ZERO SW

\*T

LXI H,1 .ADDRESS INDEX

\*M,JC ZBT

JMC=T

CONT .CONTINUE IF I <= (N-2)

\*-2D,JC ZBLT

JC CONT .CONTINUE IF I <= (N-2)

\*E

source from disk A

hex to disk A

ASM SORT AAZ ← skip pm file

CP/M ASSEMBLER - VER 1 0

015C next address to assemble

003H USE FACTOR

END OF ASSEMBLY

DDT SORT HEX test program changes

16X DDT VER 1 0

NEXT PC

315C 0000

-G100

\*0110

-D140

0140 05 00 07 00 14 00 1E 00  
0150 32 00 64 00 64 00 2C 01 EB 03 01 00 00 00 00 00 00 2 D D  
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

data sorted

- abort with rubout

-GO return to CP/M - program checks OK



## 第五章

# CP/M動態除錯工具程式

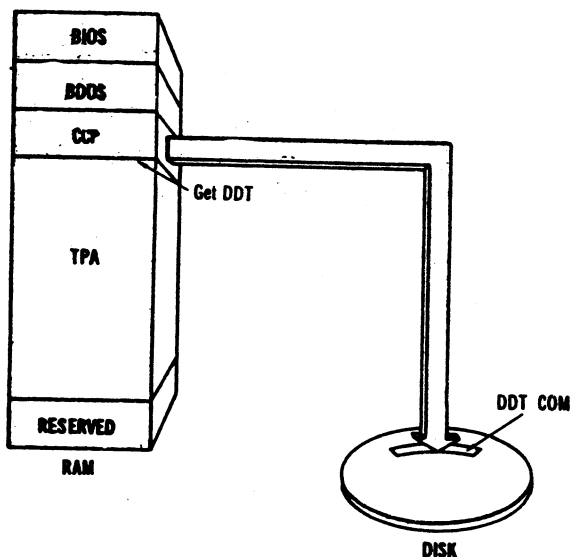
## DDT (除蟲劑)

- 簡介
- DDT 命令
- A ( Assemble ) 命令
- D ( Display ) 命令
- F ( Fill ) 命令
- G ( Go ) 命令
- I ( Input ) 命令
- L ( List ) 命令
- M ( Move ) 命令
- R ( Read ) 命令
- S ( Set ) 命令
- T ( Trace ) 命令
- U ( Untrace ) 命令
- X ( Examine ) 命令
- 使用時應注意事項
- 範例

## 簡介

DDT 程式能讓使用者交談式的測試及除錯。在 CP/M 系統發展程式時，當使用者從控制台上打入

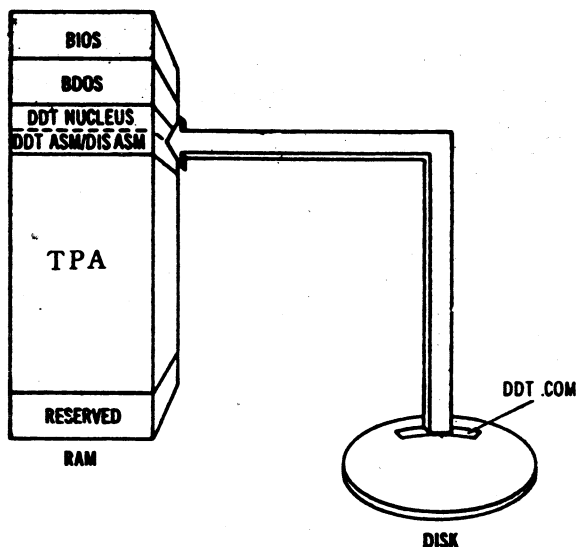
```
DDT  
DDT filename.HEX  
DDT filename.COM
```



(A) CCP 先到磁碟上拿出 DDT 載入其內

時除錯程式就被啓動。file name 是載入等候測試之程式名稱。

DDT 程式被搬到記憶區中 CCP ( Console Comand Processor ) 的位置 ( 參考本書第 3 部份第 1 章，CP/M 系統記憶體配置部份 )，亦即在 CP/M 的 Basic Disk Operation System 下面。BDOS



(B) DDT 蓋在 CCP 上，而要除錯的程式則位於 TPA 內

的起始位址存放在 5 H 位置 JMP 指令中的位址欄，此位址被改到 DDT 起始點以指示出減少了的 TPA 大小。

以上 DDT 命令之第二種及第三種形態之功能，和第一種命令的功能相同，只是多做一項把 HEX 或 COM 檔案自動載入的工作。相當於

```
DDT
I filename.HEX or I filename.COM
R
```

I 和 R 指令在後面會詳細解說。

DDT 程式啟動後，首先印出一串訊息

## 288 Apple II 軟體卡

nnK DDT-s VER m.m

nn 是記憶體的大小。(必須和所使用之 CP/M 系統相符)。

s 是硬體系統，它的代碼如下：

D Digital Research 標準版

M MDS 版

I IMSAI 標準版

O Omron 系統

S Digital Systems 標準版

m.m 是校訂版本號碼

```
A>DDT
32K DDT VER 2.0
```

### 實例 (1)

```
A>DDT TIMER.HEX
32K DDT VER 2.0
```

### 實例 (2)

接著 DDT 立刻顯現 “—” 信號，等待操作員從控制台上輸入命

令。這時操作員可以打入任何命令，最後鍵入 < cr > 系統就執行命令。操作員在輸入命令時，可以使用標準的 CP/M 控制碼來編輯該行輸入。

控制碼如下所列：

rubout	刪去前一鍵入之字元
control -X	刪去整行
control -C	系統重新啓動

每個命令最長可以有 32 個字元 (系統會自動插入 < cr > 作為第 33 個字元)，但第一個字元就決定了命令的類型

- A 輸入組合語言簡號 (mnemonics) 及其運算元
- D 以 16 進位數及 ASCII 格式顯示記憶體之內容
- F 將常數資料放入記憶體中
- G 開始執行程式 (允許有任意的中斷點)
- I 建立標準輸入檔案控制區 (FCB)
- L 用組合語言簡號 (mnemonics) 將記憶體內容印出
- M 將記憶體某一段搬至另一個位置
- R 讀入程式準備做測試
- S 替換記憶體中之數值
- T 追蹤程式 (trace program) 執行
- U 不追蹤程式 (Untraced program) 之監督
- X 檢驗並可選擇性更改 CPU 的狀態。

在命令之後可以附隨著至多 3 個 16 進位數值，以逗點或空白分隔。DDT 所產生的數字輸出，都是 16 進位格式。操作員所打入之



## 290 Apple II 軟體卡

DDT 命令要在 < cr > 被鍵入後，DDT 程式才會去執行。

在除錯的過程中，操作員可以隨時以 Control-C 或 G0 ( 跳至位址 0000H ) 來中止 DDT 程式，並可用 SAVE 命令將當時記憶體的內容保存。

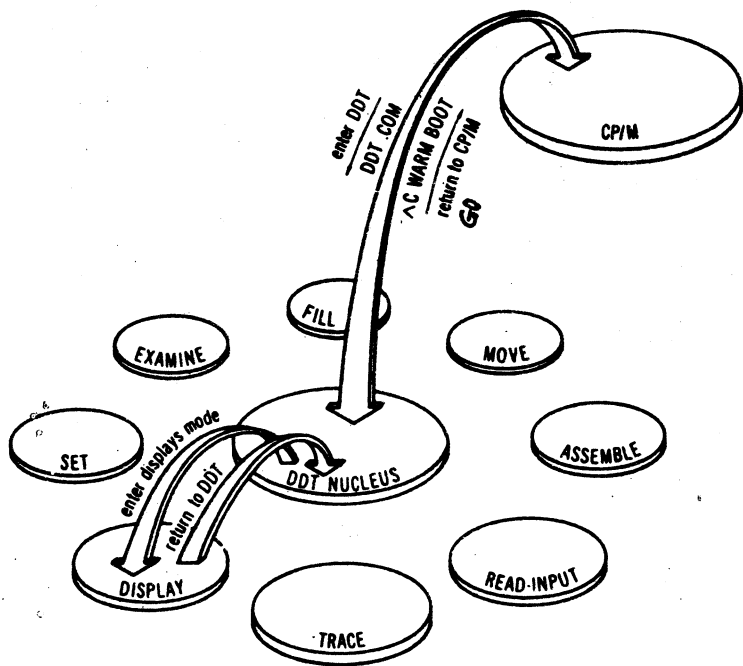
**SAVE n filename.COM**

n 代表所要保存在磁碟中的記憶體頁數 ( 每頁 256 拜，亦即 100 H 拜 )。我們只要將現在所用到的最大載入位址的高位 byte，轉換成 10 進位值，就可得到 n 值，舉例來說，假設暫時程式區的最大位址為 1234H，那麼頁數就是 12H 也就是 18 ( 十進位值 )。

因此操作員在除錯過程中，鍵入 Control-C 接著再鍵入 SAVE 18 X.COM，則記憶體現在之內容就存到磁碟中以 X.COM 為名的檔案裏，接著打入 X 就可即刻執行程式。如果操作員還要做進一步之測試可以用 DDT X.COM 命令，把磁碟所存之原有程式內容重新載入到記憶體位址 100 H 到 18 頁 ( 12 FFH ) 的位置。

注意，在 COM 檔內並不存有機器之狀態，所以必須重頭執行程式，以便能正確地加以測試。

## DDT 命令



CP/M, DDT及其所屬命令間之相互作用關係

DDT 起動以後，操作員必須等到控制台上顯示出“—”符號才能鍵入命令。如果被測試之程式正在執行，還未到達中斷點，操作員可以從前面板上執行RST 7 使控制權回到DDT程式（如果程式正在執行T或U命令，則改為按下rubout 鍵，下面將分別對每個命令作較詳細之描述。在命令字元之後有時跟著一些16進位的數字（以逗點分開）這些數字以小寫字母表示。每個數字最多只有四位數，（超過者，自動從右邊切掉）。

許多DDT命令都工作於“CPU狀態”，CPU狀態保存著正

被除錯的程式的暫存器值。所有的暫存器及旗幟的始初值，除了程式計數器 ( P ) 及堆疊指標 ( S ) ( 一般為 100 H ) 外，均為 0。如果有 HEX 檔被載入的話，程式計數器設定開始的位址為該檔 ( HEX ) 最後一個資料錄所指定的位址。( 詳見 I 與 R 命令 )。

### A ( Assemble ) 命令

當使用者在除錯過程中，常會發現所寫程式中有錯誤，希望能夠立刻更正，而不必先離開 DDT，呼叫 ED 來更改源程式然後以 ASM 重新編譯再回到 DDT 中，A ( Assemble ) 命令提供了這項功能，它能讓使用者在 DDT 中更改程式。A 命令之格式為

As

s 是 16 進位的記憶區起始位址，DDT 程式會在控制台上顯示出這個位址值，然後等待使用者輸入指令〔用標準之 8080 簡號 ( mnemonics ) 以及 16 進位之絕對位址值 ( 代表運算元或暫存器 )。DDT 將組譯 ( assemble ) 此一指令，把機器碼放入記憶體中，然後指出下一個指令的位址，等待使用者再做輸入，使用者可以用 <cr> 結束 A 命令。

完成組合語言輸入後，操作員可用 DDT 解譯程式 ( dis-assembler ) 來觀看記憶體內容 ( 詳見 L 命令 )。

操作員在使用 A 命令時要注意：因為這裡是“即時” ( real time ) 地在作組譯的工作，DDT 每次只組譯一行，因此一些像 EQU 等之虛擬指令，DDT 均不能接受，假如操作員鍵入

-A0100

0100 CALL SETBUFFER

DDT會產生下列反應

```
- A0100
  0100 CALL SETBUFFER
    ?
  0100
```

因為DDT認為SETBUFFER不是一個合法的16進位位址。如果操作員知道原來程式中有一個虛擬指令

```
SETBUFFER EQU 0D800H
```

他必須以0D800H來代替SETBUFFER，而鍵入

```
- A0100
  0100 CALL D800
  0103
```

如此DDT便能接受，並加以組譯。

注意到DDT認為CALL D800是一個3-bytes指令，因此下一個出現之位址為103。

```
-A110/----- 從110 H開始執行組譯程式 ( A 命令 )
0110 JMP 0020/ } 把這組組合語言指令打入後，將使得每
0113 CALL 5000/ } 次只組譯一行命令並產生目的碼
0116 HLT/
0117 J----- 打入<CR>，A命令結束
<0110,11F/ --- 110 H ~ 11 FH 之內容
0110 20 00 CD 00 50 76 01 C3 51 01 3E 80 32 13 02 . . . P v . . q . . 2 . .
```

DDT是由二大部分組成，一是DDT Nucleus，另一部分為DDT Assembler / Disassembler。

在被測試之程式太大時，會把第二部分 ( DDT Assembler / Disassembler ) 部分重疊上。此種情形下，操作員輸入 A 或 L 命令，DDT 會回應一項錯誤訊息。

## D ( Display )

D 命令是用來看記憶體內任意位置的內容 ( 16 進位及 ASCII 格式 ) 。

```
D
Ds
Ds,f
```

第一種形式為：從現在的顯示位址 ( 最初為 100 H )，連續顯示 16 行，每一行的格式為

```
aaaa bb bb bb bb bb bb bb bb bb bb bb bb bb bb ccccccccccccccc
```

aaaa 是字串中第一個 byte 的位址。bb 為 16 進位制的資料拜 ( 印 16 bytes )，cc …… cc 為對應於 16 bytes bb 之 ASCII 值，如果這個值是不可示字元，則 DDT 印出 “ . ” 符號 ( 例如 06H 是 ASCII Ctrl-G 之值無法示出，DDT 會印出 “ . ” ) 第二，三種形式中，s 為起始位址，f 為結束位址，如果只給 S 表示從位址 s 起顯示 16 行 ( 每行 16 拜 )。如果 s，f 均給定，則從位址 s 顯示到 f 位址。在顯示過程中，按下 Rubout 鍵，能中止顯示。

實例(1) :

```
A>DDT
32K DOT VER 2.0
-TIMER.HEX
-R
-DD800
D800 03 5F 23 A0 45 23 03 D8 C9 56 23 ..gy.9.zw..
etc.
```

實例(2) :

A>DDT DUMP.COM /----- 起動DDT，同時把DUMP，COM檔載入記憶體

DDT VERS 2.2 (從100H開始)

NEXT PC  
0300 0100

把記憶體100H~17FH之內容傾印

-D100,17F /----- 出來 (D命令)

```
0100 21 00 00 39 22 15 02 31 57 02 C0 C1 01 FE FF C2 !..9".."1W.....
0110 1B 01 11 F3 01 CD 9C 01 C3 51 01 3E 80 32 13 02 .....Q.>.2..
0120 21 00 00 E5 CD A2 01 E1 DA 51 01 47 7D E6 0F C2 !.....Q.G)....
0130 44 01 CD 72 01 CD 59 01 0F DA 51 01 7C CD 8F 01 D...r..y...Q.l...
0140 7D CD 8F 01 23 3E 20 CD 65 01 78 CD 8F 01 C3 23 }...#> .e.x...@
0150 01 CD 72 01 2A 15 02 F9 C9 E5 D5 C5 0E 0B CD 05 ...r.#.....
0160 00 C1 D1 E1 C9 E5 D5 C5 0E 02 5F CD 05 00 C1 D1 .....
0170 E1 C9 3E 0D CD 65 01 3E 0A CD 65 01 C9 E6 0F FE ...>..e.>..e.....
```

實例(3) : 以下展示一檔名為TESTPRO.BAS之檔以DDT、TYPE及DUMP三種不同命令列印出來的程式，讀者可細心比較其異同。

# 296 Apple II 軟體卡

①

A>DDT B:TESTPRO.BAS!

DDT VERS 2.2  
NEXT PC  
3400 0100

-D100,3400!

```

0100 31 30 20 27 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 10 '-----
0110 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D -----
0120 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D -----
0130 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 0D -----
0140 0A 32 30 20 27 0D 0A 33 30 20 27 20 20 20 20 20 .20 '..30 '
0150 20 20 20 20 49 54 49 20 20 43 53 20 20 55 43 53 ITI CS UCS
0160 20 20 46 4F 52 20 4D 4F 55 53 45 0D 0A 34 30 20 FOR MOUSE..40
0170 27 0D 0A 35 30 20 27 3D 3D 3D 3D 3D 3D 3D 3D 3D '..50 '-----
0180 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D -----
0190 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D -----
01A0 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D -----
01B0 3D 3D 0D 0A 36 30 20 27 0D 0A 37 30 20 50 52 49 ==.60 '..70 PRI
01C0 4E 54 20 22 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 49 NT "----- DISCRI
01D0 4D 49 4E 41 54 45 44 20 41 56 4F 49 44 41 4E 43 MINATED AVOIDANC
01E0 45 2D 53 41 4D 45 20 50 52 4F 47 52 41 4D 20 57 E-SAME PROGRAM W
01F0 49 54 48 20 34 20 42 4F 58 45 53 20 3D 3D 3D 3D ITH 4 BOXES ----
0200 3D 0D 0A 38 30 20 50 52 49 4E 54 0D 0A 39 30 20 =..80 PRINT..90
    
```

②

A>TYPE B:TESTPRO.BAS!

```

10 '-----
20 '
30 ' ITI CS UCS FOR MOUSE
40 '
50 '-----
60 '
70 PRINT "----- DISCRIMINATED JOIDANCE-SAME PROGRAM WITH 4 BOXES ---"
80 PRINT
90 '
100 '***** INITIAL SET *****
110 '
120 DEFINT A-Z 'DEFIN ALL VALIABLES TO INTEGER
130 '
140 'DISK FILE NAME DEFINE
150 INPUT "FILE NAME";FILENAMEIN$
160 INPUT "DATE";DATEIN$
170 '
180 'DISK FILE OPEN
190 OPEN "R",#1,FILENAMEIN$
    
```

③

A>DUMP B:TESTPRO.BAS J

```

0000 31 30 20 27 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D
0010 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D
0020 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D
0030 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 00
0040 0A 32 30 20 27 0D 0A 33 30 20 27 20 20 20 20
0050 20 20 20 20 49 54 49 20 20 43 53 20 20 55 43 53
0060 20 20 46 4F 52 20 4D 4F 55 53 45 0D 0A 34 30 20
0070 27 0D 0A 35 30 20 27 3D 3D 3D 3D 3D 3D 3D 3D
0080 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D
0090 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D
00A0 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D
00B0 3D 3D 0D 0A 36 30 20 27 0D 0A 37 30 20 50 52 49
00C0 4E 54 20 22 3D 3D 3D 3D 3D 20 44 49 53 43 52 49
00D0 4D 49 4E 41 54 45 44 20 41 56 4F 49 44 41 4E 43
00E0 45 2D 53 41 4D 45 20 50 52 4F 47 52 41 4D 20 57
00F0 49 54 48 20 34 20 42 4F 58 45 53 20 3D 3D 3D 3D
0100 3D 0D 0A 38 30 20 50 52 49 4E 54 0D 0A 39 30 20
    
```

## F ( Fill ) 命令

Fs,f,c

s 為起始位址，f 是結束位址，c 是 16 進位常數值。DDT 接受這個命令後，把常數 c 放入位址 s 中，然後檢驗 s 是否大於 f，若是則停止動作，否則重複上述動作。換句話說，F 命令在一塊記憶區存入一個特定值。

實例：

~~~~~  
-F120,14F,E5J ---- 120H ~ 14FH 之間填入 E5H ( F 命令 )

-D100,15FJ ----- 顯示 100H ~ 15FH 之記憶體內容

```

0100 00 01 02 03 04 05 06 07 08 02 CD C1 01 FE FF C2 .....
0110 C3 20 00 CD 00 50 76 01 C3 51 01 3E 80 32 13 02 . . . P v . G . > . 2 .
0120 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 .....
0130 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 .....
0140 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 .....
0150 01 CD 72 01 2A 15 02 F9 C9 E5 D5 C5 0E 08 CD 05 . . r . * . . . . .
    
```

## G ( Go ) 命令

G 命令使程式執行並允許程式有二個中斷點，其形式為：



G  
Gs  
Gs,b  
Gs,b,c  
G,b  
G,b,c

第一種形式，被測試之程式從現在程式計數器的值開始執行下去，沒設中斷點（只當執行 RST7 指令後才回到 DDT）。現在的程式計數器值可以用 X 或 X P 命令來觀察。第二種形式，除了把程式計數器之值定為 s 外，其他均相同。第三種形式和第二種形式相似，只是程式執行到位址 b 時就停止。（位址 b 之指令，並未被執行）。第四種形式，有二個中斷點 b 和 c 只要碰到其中之一，程式就會停止。隨後這二個中斷點都會被清除掉。最後兩種形式，從現在狀態的程式計數器值開始執行，分別設定一個或 2 個中斷點。

當程式從起始位址開始執行後，在未執行到中斷點以前控制權不交回 DDT，除非遇到 RST7 指令。當程式執行至中斷點時，DDT 會印出。

\*d

d 為停止之位址。這時可以用 X 命令來檢查現在的狀態。操作員在鍵入 G 命令時，所給的中斷點值必須和程式計數器當時之值不同。如當程式計數器值已為 1234 H 時，執行

或       G,1234  
          G400,400

立即產生中斷，而什麼指令都未執行。

### I (Input) 命令

操作員用 I 命令把一個檔案名稱插入檔案控制區 (FCB, file control block) 中。(FCB 是 CP/M 產生出來給暫存程式的, 位於 5CH 起, 參閱 CP/M Interface Guide), 這個檔案名稱也被 DDT 用來讀另外的 HEX 及 COM 檔案。

Ifilename

Ifilename.filetype

如果用第二種形式且檔案類別是 HEX 或 COM, 再用 R 命令就可以讀入純二進位機器碼或 16 進位機器碼。

### L (List) 命令

L 命令用以列出程式中某一段中之組合語言片段

L  
Ls  
Ls,f

s 表起始位址, f 為結束位址。如果兩者都未給時, 則從現在的第一行 (由現在的程式計數器值) 起, 印出 12 行組合語言片段。若給定值, 從位址 s 開始印 12 行。若是 s, f 均有; 則從位址 s 起印出至位址 f 為止。在印的過程中, 可以按下 Rubout 鍵使印出過程中止。

實例 -L100,11A/ -----100H ~ 11AH 執行 disassembler (L 命令)

```
0100 LXI H,0000
0103 DAD SP
0104 SHLD 0215
0107 LXI SP,0257
010A CALL 01C1
010D CPI FF
010F JNZ 011B
0112 LXI D,01F3
0115 CALL 019C
0118 JMP 0151
011B
```

### M ( Move ) 命令

M 命令把程式資料區的某一段搬到記憶體的另一個區域。

Ms,f,d

s 為起始位址，f 為結束位址，d 為目標區位址。首先，資料從位址 s 搬到位址 d，然後 s 及 d 均加 1，比較 s 及 f，若 s 大於 f，搬運動作就停止，否則就繼續重複搬運動作。

### R ( Read ) 命令

R 命令通常配合著 I 命令，將磁碟中之 HEX 及 COM 檔讀進暫存程式區中，以供除錯使用。

R  
Rb

b 是偏移位址，當程式或資料載入時，位址會加上 b。若 b 未給定，則被設定為 0000。載入之過程要注意不能蓋掉位址 000 H ~ 0FFH 的任一處（此為記憶體的基頁，存有系統參數）。R 命令之前必須有 I 命令，指明 HEX 及 COM 之檔案名稱。在 I 命令之後，可以用任意次數 R 命令重新讀入程式（如果程式未破壞 5 CH 的位置）任何一個“COM”形式的檔案所含的機器碼為純二進位的（由 LOAD 或 SAVE 命令產生），而其他的檔案都是 Intel 16 進位格式的機器碼。

回想一下前面曾說過

DDT filename filetype

相當於

DDT  
-I filename filetype  
-R

實例：

```
-IDUMP.ASM] ----- 把DUMP·ASM檔登錄在FCB上( I 命令 )
-R] ----- 登錄好的檔載入記憶體 ( 從100H開始 ) [ R 命令 ]
NEXT PC
1180 01C8

-D100,17F] ----- 100H~17FH記憶體內容顯示出查看R命令執行結果
0100 3B 09 46 49 4C 45 20 44 55 4D 50 20 50 52 4F 47 ;.FILE DUMP PROG
0110 52 41 4D 2C 20 52 45 41 44 53 20 41 4E 20 49 4E RAM, READS AN IN
0120 50 55 54 20 46 49 4C 45 20 41 4E 44 20 50 52 49 PUT FILE AND PRI
0130 4E 54 53 20 49 4E 20 48 45 58 0D 0A 3B 0D 0A 3B NTS IN HEX...;
0140 09 43 4F 50 59 52 49 47 48 54 20 28 43 29 20 31 .COPYRIGHT (C) I
0150 39 37 35 2C 20 31 39 37 36 2C 20 31 39 37 37 2C 975, 1976, 1977,
0160 20 31 39 37 38 0D 0A 3B 09 44 49 47 49 54 41 4C 1978...;DIGITAL
0170 20 52 45 53 45 41 52 43 48 0D 0A 3B 09 42 4F 58 RESEARCH...;BOX

-GO] ----- 從0H開始執行程式，可在任何位址設中斷指標
A> ----- 返回CP/M系統
```

當R命令被DDT收到後，若有錯誤發生DDT會回應“？”(例如檔案無法開啓或檢查碼( chechsum )錯誤等)

如果成功，會有另一訊息出現

```
NEXT PC
nynn pppp
```

nynn 是在載入程式後的下一個位址，pppp 則是程式計數器值(對COM檔案而言為100H，對HEX檔案，這個值由HEX檔案的最後一個資料錄中取得)。

**S (Set) 命令**

S 命令可以檢視任一個記憶體位址，並加以更改。

**Ss**

s 是操作員想檢視或更改之記憶體位址 (16 進位值) DDT 會回應位址及該位址之資料無論如何，DDT 繼續。這時操作員若敲下 < cr > 鍵，則資料未被更改；如果操作員想更改資料，就鍵入新的值 (16 進位)，DDT 就會將新值存入該位址中。顯示下一個位址與值，直到操作員鍵入 “.” 或是輸入不合法時，才回到 DDT 命令模式。

例：

```

- SD800
    D800 03 05 ← 操作員鍵入新值
    D801 5F ← 結束
- SD800
    D800 05 ← 值確已更改
    
```

實例：

```

-S100-----從記憶體 100H 開始打入新內容取代舊內容 ( S 命令 )
0100 21 00 }
0101 00 01 }
0102 00 02 }
0103 39 03 }
0104 22 04 } 原資料將被新鍵入資料所取代
0105 15 05 }
0106 02 06 }
0107 31 07 }
0108 57 08 }
0109 02 .----- . 句號代表 S 命令之終結
-D100,10F-----把記憶體 100H ~ 10FH 顯示出來
0100 00 01 02 03 04 05 06 07 08 02 CD C1 01 FE FF C2 .....
    
```

### T (Trace) 命令

當我們在除錯一個程式之時，往往希望能追蹤程式執行的每一步驟，以確定程式是否依照我們所期望的方式執行，或找出程式不能工作的原因，DDT的T命令提供了這項功能。

T  
或  
Tn

n 是任意一個值 ( 16 進位 )，表示執行的步驟數。當 n 之值未給時，被認為是 1，程式只執行一個步驟就停止。T 命令在執行程式的每一步驟之後，會將 CPU 暫存器、計數器、各種旗號指標顯示在控制台上，其格式為：

```

旗號 (flag)      各 暫 存 器
CfEfMfEfIf  A:bb B:dddd D:dddd H:dddd

堆疊指標  程式計數器  現在之指令
S:dddd  P:dddd  Instruction
    
```

和使用 X 命令時，所看到的格式相同。除非遇到中斷點 ( breakpoint )，否則程式會執行完 n 個步驟停下來並顯示

\*hhhh

hhhh 是下一個要被執行的位址。在 D 命令中，顯示位址是 H 及 L 暫存器對之值，而 L 命令的列印位址則同樣是 hhhh。在程式執行 n 個步驟的過程中，操作員可以按下 Rubout 鍵迫使程式中止。

## 實例：

```

A>DDT
32K DDT VER 2.0
-TIMER,HEX
-R
NEXT PC
DB24 0000
-XP
P=0000 DB00
-T10
0020M0E010 A=00 B=0000 D=0000 H=0000 S=0100
P=DB00 JMP DB04
0020M0E020 A=00 B=0000 D=0000 H=0000 S=0100
P=D6J4 PUSH H
0020M0E030 A=00 B=0000 D=0000 H=0000 S=00FE
P=DB05 PUSH D
0020M0E040 A=00 B=0000 D=0000 H=0000 S=00FC
P=DB06 PUSH B
0020M0E050 A=00 B=0000 D=0000 H=0000 S=00FA
P=DB07 PUSH PSW
0020M0E060 A=00 B=0000 D=0000 H=0000 S=00F8
P=DB08 XRA A
0021M0E110 A=00 B=0000 D=0000 H=0000 S=00F8
P=DB09 OUT P0
0021M0E110 A=00 B=0000 D=0000 H=0000 S=00F8
P=DB0A STA DB05
0021M0E110 A=00 B=0000 D=0000 H=0000 S=00F8
P=DB0E MVI B,84
0021M0E110 A=10 B=8400 D=0000 H=0000 S=00F8
P=DB10 MVI A,B0
0021M0E110 A=80 B=8400 D=0000 H=0000 S=00F8
P=DB12 OUT P0
0021M0E110 A=80 B=8400 D=0000 H=0000 S=00F8
P=DB14 DCR B
0020M0E111 A=80 B=8300 D=0000 H=0000 S=00F8
P=DB15 NOP
0020M0E111 A=80 B=8300 D=0000 H=0000 S=00F8
P=DB16 NOP
0020M0E111 A=80 B=8300 D=0000 H=0000 S=00F8
P=DB17 NOP
0020M0E111 A=80 B=8300 D=0000 H=0000 S=00F8
P=DB18 NOP ; DB18

```

使用T命令來追蹤程式，在和CP/M溝通時，程式執行過程不連續，直到控制權由CP/M回到被測程式時才恢復進行。如此，CP/M的即時性存取輸出入裝置（如磁碟）的功能才不致發生即時時序問題。以T命令來執行程式時，其速度比正常之程式執行速度約慢500倍。中斷處理程式也能用T命令來追蹤，但值得注意的是用到斷點（breakpoint）的命令（如G、T和U命令）是利用RST 7指令來完成中斷的，所以被測程式不能使用這個中斷位址。並且追蹤被測程式時，系統仍是可中斷性的，此時如果收到非同步的中斷就可能發生問題。

操作員若想由追蹤程式將控制權取回DDT，必須使用Rubout鍵而不應執行RST 7，以確保追蹤當下指令完畢後才被中斷。

### U (Untrace) 命令

U命令和T命令幾乎完全相同，只是U命令不顯示出所執行的立即程式步驟的CPU狀態。在T命令中曾舉了一例子，若改用U命令其結果如下例所示。

實例：

```
A>DDT
32K DDT VER 2.0
-TIMER.HEX
-R
NEXT PC
D824 0000
-XP
P=0000 D800
-U10
C020M0E010 A=00 B=0000 D=0000 H=0000 S=0100
P=D800 JMP D804
★ D810
-
```

讀者或許會奇怪，既然有T命令為何又要有一個類似功能的U命令，U命令可以讓使用者在DDT控制之下執行被測之程式。在測試一個較長程式時，使用者可以先用U命令快速執行，發覺有問題再用T命令一步步地偵測。聰明的讀者或許會想到，只要先將程式計數器定到想要測試的位址，再執行T命令，不就可以了嗎？然而有許多程式，一開始很多變數都必須先行設定，若是程式跳過這一段從中間直接開始執行，執行所得之結果通常是錯的，所以我們還是需要以U命令快速地執行過直到我們想要開始測試及除錯之位址，再用T命令來追蹤。



**X ( Examine ) 命令**

X 命令用以檢視或改變 CPU 狀態，其形式為：

X  
Xr

r 是 8080 CPU 的暫存器之一。可以是

- C 進位旗號 ( 0/1 )
- Z 零旗號 ( 0/1 )
- M 負旗號 ( 0/1 )
- E 偶同位旗號 ( 0/1 )
- I 中間位數進位 ( 0-FF )
- A 累積器 ( 0-FFFF )
- B BC 暫存器對 ( 0-FFFF )
- D DE 暫存器對 ( 0-FFFF )
- H HL 暫存器對 ( 0-FFFF )
- S 堆疊指標 ( 0-FFFF )
- P 程式計數器 ( 0-FFFF )

用第一種型態時 ( X )，DDT 會回應：

| 旗 | 號 | 暫 | 存 | 器 |   |       |         |         |         |
|---|---|---|---|---|---|-------|---------|---------|---------|
| C | Z | M | E | I | f | A: bb | B: dddd | D: dddd | H: dddd |

|         |         |       |
|---------|---------|-------|
| 堆疊指標    | 程式計數器   | 現在之命令 |
| S: dddd | P: dddd | inst  |

f 為 0 或 1 (旗號值), bb 是一個 byte 值, dddd 為二個 byte 值 (對應於一對暫存器)。指令欄有 CPU 程式計數器所指位址中之指令, 第二種情形 (Xr), 可以檢視及更改某一 CPU 暫存器之值, r 為表中任一個暫存器, DDT 將值回應於控制台上, 如同在 S 命令中一般, 此時若只按 <cr> 鍵, 則此一暫存器之值不會改變, 若是輸入任一合理之新值, 則原值就改變。

```
例：   - XA
        FF00  新輸入之值
        - XA
        00    A 之值由 FF 變成 00
```

**實例：**

```
-XJ ---- CPU內各種狀態 ( X 命令 )
COZOMOEO10 | A=00 B=0000 D=0000 H=0000 | S=0100 P=0100 | LXI H,0000 |
 旗號      | 各暫存器      | SP      PC      | 現在之命令
-TB)
COZOMOEO10 | A=00 B=0000 D=0000 H=0000 | S=0100 P=0100 | LXI H,0000
COZOMOEO10 | A=00 B=0000 D=0000 H=0000 | S=0100 P=0103 | DAD SP
COZOMOEO10 | A=00 B=0000 D=0000 H=0100 | S=0100 P=0104 | SHLD 0215
COZOMOEO10 | A=00 B=0000 D=0000 H=0100 | S=0100 P=0107 | LXI SP,0257
COZOMOEO10 | A=00 B=0000 D=0000 H=0100 | S=0257 P=010A | CALL 01C1
COZOMOEO10 | A=00 B=0000 D=0000 H=0100 | S=0255 P=01C1 | XRA A
COZ1MOE110 | A=00 B=0000 D=0000 H=0100 | S=0255 P=01C2 | STA 007C
COZ1MOE110 | A=00 B=0000 D=0000 H=0100 | S=0255 P=01C5 | LXI D,005C+01CB
```

**使用注意事項**

DDT 容許某些非必要的部份被蓋掉以使得 Transient Program Area 增大以便來處理較大程式。DDT 程式包括兩部分：DDT 核心及 DDT Assembler / Disassembler 模組。

DDT 被起動後, 二者均被載入記憶體中 CCP ( Console Command Processor ) 的位置, 不過若是待測程式較大 TPA ( Transient

Program Area) 放不下, DDT ASM/DIS ASM 模組可以被待測程式所蓋掉。( 如果不需要做 Assembler 或 Disassembler )。

DDT 程式載入記憶體後, 更改記憶體位址 6 H 所存的 BDOS 位址, 換為 DDT 核心的基底位址。如此之後系統認為記憶體邏輯上的終點位址為 DDT 的基底位址, 而不是 BDOS 的基底位址。

ASM/DIS ASM 模組緊接在 DDT 核心之下, 如果要使用到 A, L, T 或 X 命令時, DDT 程式會修正記憶體位址 6 H 中所含的位址, 以便將 ASM/DIS ASM 包含進來, 此後自然記憶體邏輯的終點位址也隨之減低了。如果, 待測程式太大, 載入後超過 ASM/DIS ASM 模組大小, A 及 L 命令就無法執行 ( 控制台會回應出 “ ? ” ) , 而且 T 及 X 命令執行後, 所列出的指令欄位, 將出現 16 進位值, 而不是一解好碼的指令。

16H 為 5 H JMP 指令的位址欄

## 範例

下面這個例子包括程式的編輯、組譯、除錯過程。這個程式讀入一組資料, 當程式結束時, 這組資料中的最大值將存放入 “ LARGE ” 變數中。

```

EB SCAN.ASM
*1:
  |1|  |1|  ORG  |1| 100H  |1| LISTARY OF TRANSIENT AREA
  |1|  |1|  |1| B.LEN  |1| .LENGTH OF VECTOR TO SCAN
  |1|  |1|  |1| C.0    |1| .LARGER.HIT VALUE SO FAR
  |1|  |1|  |1| L.XI   |1| .N.VECT .BASE OF VECTOR
  |1|  |1|  |1| MOV  |1| A.H  |1| .GET VALUE
  |1|  |1|  |1| SMC  |1| C.    |1| .LARGER VALUE IN C?
  |1|  |1|  |1| SMC  |1| NEQND  |1| .JUMP IF LARGER VALUE NOT FOUND
  |1|  |1|  |1| MOV  |1| NEU  |1| .LARGEST VALUE. STORE IT TO C
  |1|  |1|  |1| MOV  |1| C.A
  
```

```

NFOUND INX H ;TO NEXT ELEMENT      Create Source
        DCR B ;MORE TO SCAN?
        JNZ LOOP ;FOR ANOTHER      Program - underlined
;
;
;         END OF SCAN, STORE C      characters typed
        MOV A,C ;GET LARGEST VALUE by programmer
        STA LARGE
        JMP 0 ;REBOOT
;
;
;         TEST DATA
VECT DB 2,0,4,3,5,6,1,5
LEN EQU &VECT ;LENGTH
LARGE DS 1 ;LARGEST VALUE ON EXIT
        END

Z- OOP
ORG 100H ;START OF TRANSIENT AREA
MVI B,LEN ;LENGTH OF VECTOR TO SCAN
MVI C,0 ;LARGEST VALUE SO FAR
LXI H,VECT ;BASE OF VECTOR
LOOP MOV A,H ;GET VALUE
      SUB C ;LARGER VALUE IN C?
      JNC NFOUND ;JUMP IF LARGER VALUE NOT FOUND
      MVI C,A ;NEW LARGEST VALUE, STORE IT TO C
      INX H ;TO NEXT ELEMENT
      DCR B ;MORE TO SCAN?
      JNZ LOOP ;FOR ANOTHER
      END OF SCAN, STORE C
      MOV A,C ;GET LARGEST VALUE
      STA LARGE
      JMP 0 ;REBOOT
;
;         TEST DATA
VECT DB 2,0,4,3,5,6,1,5
LEN EQU &VECT ;LENGTH
LARGE DS 1 ;LARGEST VALUE ON EXIT
        END
    
```

ASM SCAN *Start Assembler*

CP/M ASSEMBLER - VER 1.0

0122  
002H USE FACTOR  
END OF ASSEMBLY

*Assembly Complete - look at program listing*

TYPE SCAN PRN  
*Code Address Source Program*

```

0100 Machine code DRG 100H ;START OF TRANSIENT AREA
01000600 MVI B,LEN ;LENGTH OF VECTOR TO SCAN
0102 0E00 MVI C,0 ;LARGEST VALUE SO FAR
0104 211901 LXI H,VECT ;BASE OF VECTOR
0107 7E LOOP MOV A,H ;GET VALUE
0100 91 SUB C ;LARGER VALUE IN C?
0109 D20D01 JNC NFOUND ;JUMP IF LARGER VALUE NOT FOUND
              NEW LARGEST VALUE, STORE IT TO C
010C 4F MOV C,A
010D 23 NFOUND INX H ;TO NEXT ELEMENT
010E 05 DCR B ;MORE TO SCAN?
010F C20701 JNZ LOOP ;FOR ANOTHER
;
;         END OF SCAN, STPRE C
0112 79 MOV A,C ;GET LARGEST VALUE
0113 322101 STA LARGE
0116 C30000 JMP 0 ;REBOOT
Code/data listing truncated ->
0119 0200040305VECT TEST DATA
        DB 2,0,4,3,5,6,1,5
    
```

# 310 Apple II 軟體卡

```
0000 =          LEN      EQU      $-VECT      ;LENGTH
0121 Value of)  LARGE   DS       !           ;LARGEST VALUE ON EXIT
0122 Equate
A>
```

DDT SCAN HEX

Start Debugger using format machine code

```
16K DDT VER 1.0
NEXT PC
0121 0000
```

```
-X → last load address +1
COZOMOEIO A=00 B=0000 D=0000 E=0000 S=0100 P=0000 OUT 7F PC=0
→ next instruction to execute at
```

```
-P
P=0000 100 → Examine registers before debug run
Change PC to 100
```

```
-I → Look at registers again
PC changed
```

```
COZOMOEIO A=00 B=0000 D=0000 E=0000 S=0100 P=0100 MVI B,08
Li00 → Next instruction to execute at PC=100
```

```
0100 MVI B,08
0102 MVI C,00
0104 LLI M,0119
0107 MOV A,M
0108 SUB C
0109 JNC 010D
010C MOV C,A
010D INX B
010E DCR B
010F JNZ 0107
0112 MOV A,C
-L
```

Disassembled Machine Code at 100H (See Source Listing for composition)

```
0113 STA 0120
0116 JMP 0000
0119 STAX B
011A MOP
011B INR B
011C INX B
011D DCR B
011E MVI B,01
0120 DCR B
0121 LLI D,2200
0124 LLI H,0200
```

A little more machine code (note that Program ends at location 116 with a JMP to 0000)

```
-A116 enter inline assembly mode to change the JMP to 0000 into a RST 7 which will cause the program under test to return to DDT if 116H is ever executed
0116 RST 7
0117 (single carriage return stops assembly mode)
```

```
-L113 List Code at 113H to check that RST 7 was properly inserted
```

```
0113 STA 0121
0116 RST 07 → in place of JMP
0117 MOP
0118 MOP
0119 STAX B
011A MOP
011B INR B
011C INR B
```

```
-I Look at registers
```

```
COZOMOEIO A=00 B=00 E=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08
-I Execute Program for one step. initial CPU state before is executed
COZOMOEIO A=00 B=0000 D=0000 E=0000 S=0100 P=0100 MVI B,08*0102
-T Trace one step again (note 08H in B) automatic breakpoint
COZOMOEIO A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI C,00*0104
```

# 第五章 CP/M動態除錯工具程式DDT (除蟲劑) 311

```

-I Trace again (Register C is cleared)
-I Trace again (Register C is cleared)
COZM0E010 A=00 B=0800 D=0000 H=0119 S=0100 P=0104 LJI H,0119*0107
-T3 Trace three steps
COZM0E010 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M
COZM0E010 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C
COZM0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC 010D*010D
-D119 Display memory standing at 119H Automatic breakpoint at 10EH
1019 02 00 04 03 05 06 01 Program data Lower case x
1020 05 11 00 22 21 00 02 7E E8 77 13 23 78 00 78 B1...! ".v.q.x
1035 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 30 (...).
1040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 Data is displayed
1050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 in ASCII with a "0"
1060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 in the position of
1070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 non-graphic
1080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 characters
1090 00 00 00 00 00 00 00 00 00 00 00 00 00 00
11A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
11B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
11C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00
-I Current CPU state
COZM0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H
-T5 Trace 5 steps from current CPU state
COZM0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H
COZM0E011 A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR B Automatic
COZM0E011A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ 0107 Breakpoint
COZM0E011 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV A,M
COZM0E011 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB C,0109
-U5 Trace without listing intermediate states
COZM0E011 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC 0105*0108
-I CPU State at end of U5
COZM0E011 A=04 B=0600 D=0000 H=011B S=0100 P=0108 SUB C
-C Run program from current PC until completion (in real-time)
breakpoint at 116H caused by executing RST 7 in machine code
-0116
-I CPU state at end of program
COZM0E011 A=00 B=0000 D=0000 H=0121 S=0100 P=0116 RST 07
-XP exasone and change program counter
P=0116 100
-I
COZM0E011 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI B,08
-T10 Trace 10 (hexadecimal) steps subtext for comparison
first data element current largest value
COZM0E011 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI B,08
COZM0E011 A=00 B=0800 D=0000 H=0121 S=0100 P=0102 MVI C,00
COZM0E011 A=00 B=0800 D=0000 H=0121 S=0100 P=0104 LJI H,0119
COZM0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M
COZM0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C

```

# 312 Apple II 軟體卡

```

CCZOMGE011 A=02 B=0000 D=0000 H=0119 S=0100 P=0109 JNC 010D
CCZOMGE011 A=02 B=0000 D=0000 H=0119 S=0100 P=010D INX H
CCZOMGE011 A=02 B=0000 D=0000 H=011A S=0100 P=013E DCR B
CCZOMGE011 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ 0107
CCZOMGE011 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV A,H
CCZOMGE011 A=0C B=0700 D=0000 H=010A S=0100 P=0108 SUB C
CCZ1MGE011 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC 010D
CCZ1MGE111 A=00 B=0700 D=0000 H=011A S=0100 P=010D INX H
CCZ1MGE111 A=00 B=0700 D=0000 H=011B S=0100 P=010E DCR B
CCZOMGE111 A=00 B=0600 D=0000 H=0110 S=0100 P=010F JNZ 0107
CCSSMGE111 A=00 B=0600 D=0000 H=0110 S=0100 P=0107 MOV A,M#0100

```

```

-3109      Insert a "hot patch" into      Program should have moved the
0109 JC 10D      the machine code                value from A into C since A>C
                                to change the                Since this code was not executed
010C      JNC to JC                    it appears that the JNC should
                                have been a JC instruction

```

```

-90      Stop DDT so that a version of
the patched program can be saved

```

```

SAVE I SCAN CGN      Program resides on first page so save 1 page

```

```

A>DDT SCAN CGN      Restart DDT with the saved memory image to continue testing

```

```

16E DDT VER I 0

```

```

NEXT PC
020C 0100

```

```

-L100      List some code .

```

```

0100 MVI B,00
0102 MVI C,00
0104 LXI M,0119      Previous patch is present in XCOM
0107 MOV A,M
0108 SUB C
0109 JC 010D
010C MOV C,A
010D INX H
010E DCR B
010F JNZ 0107
0112 MOV, A,C
-KF      A

```

```

P=010B

```

```

-T10      Trace to see how patched version operates      Data is moved from A to C

```

```

COZOMGE010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08
COZOMGE010 A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI C,00
COZOMGE010 A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI H,0119
COZOMGE010 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M
COZOMGE010 A=02 B=0800 D=0000 H=0119 S=0100 P=0100 SUB C
COZOMGE011 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC 010D
COZOMGE011 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV C,A
COZOMGE011 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX H
COZOMGE011 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR B
COZOMGE011 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ 0107
COZOMGE011 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV A,M
COZOMGE011 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB C
CIZOMIE010 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC 010D
CIZOMIE010 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX H
CIZOMIE010 A=FE B=0702 D=0000 H=011B S=0100 P=010E DCR B
CIZOMIE011 A=FE B=0602 D=0000 H=011B S=0100 P=010F JNZ 0107*0107

```

```

-X      breakpoint after 16 steps

```

```

CIZOMIE011 A=FE B=0602 D=0000 H=011B S=0100 P=0107 MOV A,M

```

```

-5 108      Run from current PC and breakpoint at 108H

```

```

*0108      next data item
-X

```

```

CIZOMIE011 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB C

```

```

-T                               Single step for a few cycles
COZIMOEIII A=04 B=0602 D=0000 H=0110 S=0100 P=0108 SUB C*0109
-T
COZIMOEIII A=02 B=0602 D=0000 E=0110 S=0100 P=0109 JC 010D*010C
-I
COZIMOEIII A=02 B=0602 D=0000 E=0110 S=0100 P=010C MOV C,A
-U                               Run to completion
*0116
-I
COZIMOEIII A=03 B=0003 D=0000 H=0121 S=0100 P=0116 RST 07
-S121                            look at the value of "LARGE"
0121 03                            Wrong Value!
0122 00
0123 22
0124 21
0125 00
0126 02                            End of the S command
0127 7E
-L100
0100 MVI B,08
0102 MVI C,00
0104 LDI H,0119
0107 MOV A,M
0108 SUB C
0109 JC 010D
010C MOV C,A
010D INX H
010E DCR B
010F JNZ 0107
0112 MOV A,C
-L
0113 STA 0121
0116 RST 07
0117 NOP
0118 NOP
0119 STAX B
011A NOP
011E INR B
011C INX B
011D DCR B
011E MVI B,01
0120 DCR B
-IP

P=0116 100                        Reset the PC

-T                               Single step and watch data values
COZIMOEIII A=03 B=0003 D=0000 H=0121 S=0100 P=0100 MVI B,08*0102
-T
COZIMOEIII A=03 B=0803 D=0000 H=0121 S=0100 P=0102 MVI C,00*0104
-I                               court set
                                "largest" set
COZIMOEIII A=03 B=0803 D=0000 H=0121 S=0100 P=0104 LDI H,0119*0107

```

*Review the code*



# 314 Apple II 軟體卡

COZIMOEIII A=03 B=0800 D=0000 H=0109 S=0100 P=0107 MOV A,M\*0100 base address of data set

-T

COZIMOEIII A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C\*0109 first data item brought to A

-T

COZOMOEIII A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC 010D\*010C

-T

COZOMOEIII A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX H\*010E first data item moved to C correctly

-T

COZOMOEIII A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR B\*010F

-T

COZOMOEIII A=0 B=0702 D=0000 H=011A S=0100 P=010F JNZ 0107\*0107

-T

COZOMOEIII A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV A,M\*0100 second data item brought to A

-T

COZOMOEIII A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB C\*0109

-T

CIZOMOEIII A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC 010D\*010D subtract destroys data value which was loaded!!!

-T

CIZOMOEIII A=FE B=0702 D=0000 H=011A S=0100 P=010D INX H\*010E

-L100

0100 MVI B,08  
 0102 MVI C,00  
 0104 LXI H,0119  
 0107 MOV A,M  
 0108 SUB C  
 0109 JC 010D  
 010C MOV C,A  
 010D INX H  
 010E DCR B  
 010F JNZ 0107  
 0112 MOV A,C

This should have been a CMP so that register A would not be destroyed

-A108

0100 CMP C hot patch at 108H changes SUB to CMP

0109

-00 stop DDT for SAVE  
SAVE I SCAN COM

save memory image  
 Restart DDT

A DDT SCAN COM

16K DDT VER I.O  
 NEXT PC  
 0200 0100

-XP

P=0100

-L110  
 0116 RST 07  
 0117 NOP  
 0118 NOP Look at code to see if it was properly loaded  
 0119 STAX B (long typeout aborted with rubout)  
 011A NOP  
 (rubout)

-C 116 Run from 100H to completion

-0116  
~~-IC~~ Look at carry (accidental typo)

C1

-A look at CPU state

CIZIMOEHIII A=06 B=0006 D=0000 H=0121 S=0100 P=0116 RST 07

~~-S121~~ Look at "Large" - it appears to be correct

0121 06

0122 00

0123 22

~~-00~~ stop DDT

ED SCAN ASM Re edit the source program and make both changes

~~-NSUB~~  
~~-OLT~~ SUB C CUZ LARGER VALUE IN C  
~~SSUB+ZCMP+ZOLT~~  
 CMP C LARGER VALUE IN C  
 \* JNC NFOUND JUMP IF LARGER VALUE NOT FOUND  
~~\*SSUB+ZOLT~~  
 JC NFOUND JUMP IF LARGER VALUE NOT FOUND

~~-E~~

ASM SCAN AAZ Reassemble selecting source from disk A  
 hex to disk A  
 CP/M ASSEMBLER - VER 1.0 print to Z (selects no print file)

0122  
 002H USE FACTOR  
 END OF ASSEMBLY

DDT SCAN HEX Return debugger to check changes

16K DDT VER 1.0  
 NEXT PC  
 A121 0000  
~~-L116~~

0116 JMF 0000 check to ensure end is still at 116H  
 0119 STAX B  
 011A NOP  
 0110 INR B  
 - (rubout)

~~-0100 110~~ Go beginning with breakpoint at end

316 Apple II 軟體卡

```
-0116 breakpoint reached
-0121 Look at "LARGE" correct value computed
0121 06 00 22 21 00 02 7E EB 77 13 23 EB 00 70 01
0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- (reboot) abort long typeout
00 stop DDT debug session complete
```

# 附 錄

1. 命令 ( Command ) 摘要 ( 字母大小排列 ) 附 1
2. C P / M 催促信號 ( Prompts ) 附 16
3. 與 C P / M 可相容的語言, 程式及計算機 附 17

## ASM Command Lines

---

**ASM filename <cr>**

Assemble the file filename.ASM; use the currently logged disk for all files.

**ASM filename.opt <cr>**

Assemble the file filename.ASM on drive o: (A:,B:,...,P:). Write HEX file on drive p: (A:,B:,...,P:), or skip if p: is Z:.

Write PRN file on drive t: (A:,B:,...,P:), send to console if p: is X:, or skip if p: is Z:.

## DDT Command Lines

---

**DDT <cr>**

Loads DDT and waits for DDT commands.

**DDT x:filename.typ <cr>**

Loads DDT into memory and also loads filename.typ from drive x into memory for examination modification, or execution.

## DDT COMMAND SUMMARY

---

**A ssss <cr>**

Enter assembly language statements beginning at hexadecimal address ssss.

**D <cr>**

Display the contents of the next 192 bytes of memory.

**D ssss,ffff <cr>**

Display the contents of memory starting at hexadecimal address ssss and finishing at hexadecimal address ffff.

**Fsss,ffff,cc <cr>**

Fill memory with the 8-bit hexadecimal constant cc starting at hexadecimal address ssss and finishing with hexadecimal address ffff.

**G <cr>**

Begin execution at the address contained in the program counter.

**Gsss <cr>**

Begin execution at hexadecimal address ssss.

**Gsss,bbbb <cr>**

Set a breakpoint at hexadecimal address bbbb, then begin execution at hexadecimal address ssss.

**Gsss,bbbb,cccc <cr>**

Set breakpoints at hexadecimal addresses bbbb and cccc, then begin execution at hexadecimal address ssss.

**G,bbbb <cr>**

Set a breakpoint at hexadecimal address bbbb, then begin execution at the address contained in the program counter.

**G,bbbb,cccc <cr>**

Set breakpoints at hexadecimal addresses bbbb and cccc, then begin execution at the address contained in the program counter.

**Ifilename.typ <cr>**

Set up the default file control block using the name filename.typ.

**L <cr>**

List the next eleven lines of assembly language program disassembled from memory.

**Lsss <cr>**

List eleven lines of assembly language program disassembled from memory starting at hexadecimal address ssss.

**Lsss,ffff <cr>**

List the assembly language program disassembled from memory starting at hexadecimal address ssss and finishing at hexadecimal address ffff.

**Msss,ffff,dddd <cr>**

Move the contents of the memory block starting at hexadecimal address ssss and ending at hexadecimal address ffff to the block of memory starting at hexadecimal address dddd.

**R <cr>**

Read a file from disk into memory (use I command first).

**Rnnnn <cr>**

Read a file from disk into memory beginning at the hexadecimal address nnnn higher than normal (use ! command first).

**Ssss <cr>**

Display the contents of memory at hexadecimal address ssss and optionally change the contents.

**Tnnnn <cr>**

Trace the execution of (hexadecimal) nnnn program instructions.

**Unnnn <cr>**

Execute (hexadecimal) nnnn program instructions, then stop and display the CPU registers' contents.

**X <cr>**

Display the CPU registers' contents.

**Xr <cr>**

Display the contents of CPU register or flag r and optionally change it.

## DIR Command Lines

---

**DIR x: <cr>**

Displays directory of all files on drive x:; x: is optional; if omitted, the

currently logged drive is used.

**DIR x:filename.typ <cr >**

Displays directory of all files on drive x: whose names match the ambiguous or unambiguous filename.typ. x: is optional; if omitted, the currently logged drive is used.

## DUMP Command Line

---

**DUMP x:filename.typ <cr >**

Displays the hexadecimal representations of each byte stored in the file filename.typ on drive x: . If filename.typ is ambiguous, displays the first file which matches the ambiguous file name.

## ED Command Line

---

**ED x:filename.typ <cr >**

Invokes the editor, which then searches for filename.typ on drive x: and creates a temporary file x:filename.\$\$\$ to store the edited text. filename.typ is unambiguous x: is optional; if omitted, the currently logged drive is assumed.

## ED COMMAND SUMMARY

---

Note: Non-alphabetic commands follow the Z command.

nA

Append lines. Moves "n" lines from original file to edit buffer.  
 OA moves lines until edit buffer is at least half full

+/-B

Begin/Bottom. Moves CP:  
 +B moves CP to beginning of edit buffer,  
 -B moves CP to end of edit buffer



**+/-nC**

Move by characters. Move CP by "n" character positions:  
+ moves forward  
- moves backward

**+/-nD**

Delete characters. Deletes "n" characters before or after the CP in the edit buffer:  
+ deletes before the CP  
- deletes after the CP

**E**

End. Ends edit, closes files, and returns to CP/M, normal end.

**nFstring ^ Z**

Find string. Find the "n"th occurrence of string, beginning the search after the CP.

**H**

Move to head of edited file. Ends edit, renames files, then edits former temporary file.

**I <cr>**

Enter insert mode. Text from keyboard goes into edit buffer after the CP; exit with Control-Z.

**Istring ^ Z**

Insert string. Inserts string in edit buffer after the CP.

**Istring <cr>**

Insert line. Inserts string and CRLF in the edit buffer after the CP.

**nJfindstring ^ Zinsertstring ^ Zendstring ^ Z**

Juxtaposition. Beginning after the CP, finds findstring, inserts insertstring after it, then deletes all following characters up to but not including endstring; repeats until performed "n" times.

**+/-nK**

Kill lines. Deletes "n" lines:  
 + deletes after the CP  
 - deletes before the CP

**+/-nL**

Move by lines. Moves the CP to the beginning of the line it is in, then moves the CP "n" lines forward or backward:  
 + moves forward  
 - moves backward

**nMcommandstring ^Z**

Macro command. Repeats execution of the ED commands in commandstring "n" times.  
 "n"=0, "n"=1, or "n" absent repeats execution until error occurs.

**nNstring ^Z**

Find string with autoscan. Finds the "n"th occurrence of string, automatically appending from original file and writing to temporary file as necessary.

**O**

Return to original file. Empties edit buffer, empties temporary file, returns to beginning of original file, ignores previous ED commands.

**+/-nP**

Move CP and print pages. Moves the CP forward or backward one page then displays the page following the CP. nP displays "n" pages, pausing after each.

**Q**

Quit edit. Erases temporary file and block move file, if any, and returns to CP/M; original file is not changed.

**R <cr>**

Read block move file. Copies the entire block move file X\$\$\$\$\$\$\$.LIB from disk and inserts it in the edit buffer after the CP.

**Rfilename <cr>**

Read library file. Copies the entire file filename with extension LIB from the disk and inserts it in the edit buffer after the CP.

**n\$findstring ^Zreplacestring ^Z**

Substitute string. Starting at the CP, repeats "n" times: finds findstring and replaces it with replacestring.

**+/-nT**

Type lines. Displays "n" lines:

+ displays the "n" lines after the CP.

- displays the "n" lines before the CP

If the CP is not at the beginning of a line:

OT displays from the beginning of the line to the CP

T displays from the CP to the end of the line

OTT displays the entire line without moving the CP

**+/-U**

Upper case translation. After +U command, alphabetic input to the edit buffer is translated from lower case to upper case; after -U, no translation occurs.

**OV**

Edit buffer free space/size. Displays the decimal number of free (empty) bytes in the edit buffer and the total size of the edit buffer.

**+/-V**

Verify line numbers. After +V, a line number is displayed with each line displayed; ED's prompt is then preceded by the number of the line containing the CP. After -V, line numbers are not displayed, and ED's prompt is \*

**nW**

Write lines. Writes first "n" lines from the edit buffer to the temporary file; deletes these lines from the edit buffer.

nX

Block transfer (Xfer). Copies the "n" lines following the CP from the ed: buffer to the temporary block move file X\$\$\$\$\$\$\$.LIB; adds to previous contents of that file.

nZ

Sleep. Delays execution of the command which follows it. Larger "n" gives longer delay, smaller n gives shorter delay.

n:

Move CP to line number "n." Moves the CP to the beginning of line number "n" (see +/-V)

:m

Continue through line number "m." A command prefix which gives the ending point for the command which follows it. The beginning point is the location of the CP (see +/-V).

+/-r

Move and display one line. Abbreviated form of +/-nLT.

## ERA Command Lines

---

**ERA x:filename.typ <cr>**

Erase the file filename.typ on the disk in drive x. filename and/or typ can be ambiguous. x: is optional; if omitted, the currently logged drive is used.

**ERA x:.\* <cr>**

Erase all files on the disk in drive x.: x: is optional; if omitted, the currently logged drive is used.

## Line Editing Commands

---

### Control-C

Restarts CP/M if it is the first character in command line. Called *warm start*.

### Control-E

Moves to beginning of next line. Used for typing long commands.

### Control-H or Backspace

Deletes one character and erases it from the screen (CP/M version 2.0 and newer).

### Control-J or Line Feed

Same as carriage return (CP/M version 2.0 and newer).

### Control-M

Same as carriage return (<cr>).

### Control-P

Turns on the *list device* (usually your printer). Type it again to turn off the list device.

### Control-R

Repeats current command line (useful with version 1.4); it verifies the line is corrected after you delete several characters (CP/M version 1.4 and newer).

### Control-S

Temporarily stops display of data on the console. Press any key to continue.

**Control-U or Control-X**

Cancels current command line (Control-X in CP/M version 1.4 and newer).

**Rubout (RUB) or Delete (DEL)**

Deletes one character and echoes (repeats) it.

**LOAD Command Line**

---

**LOAD x:filename <cr>**

Reads the file filename.HEX on drive x: and creates the executable program file filename.COM on drive x:.

**MOVCPM Command Line**

---

**MOVCPM <cr>**

Prepare a new copy of CP/M which uses all of memory; give control to the new CP/M, but do not save it on disk.

**MOVCPM nn <cr>**

Prepare a new copy of CP/M which uses "nn" Kbytes of memory; give control to the new CP/M, but do not save it on disk.

**MOVCPM \* \* <cr>**

Prepare a new copy of CP/M, which uses all of memory, to be saved with SYSGEN or SAVE.

**MOVCPM nn \* <cr>**

Prepare a new copy of CP/M, which uses "nn" Kbytes of memory, to be saved with SYSGEN or SAVE.

"nn" is an integer decimal number. "nn" can be 16 through 64 for CP/M 1.3 or 1.4. "nn" can be 20 through 64 for CP/M 2.0 and newer.

## PIP Command Lines

---

**PIP <cr>**

Loads PIP into memory. PIP prompts for commands, executes them, then prompts again.

**PIP pipcommandline <cr>**

Loads PIP into memory. PIP executes the command pipcommandline, then exits to CP/M.

## PIP COMMAND SUMMARY

---

**x.new.typ=y:old.top[p] <cr>**

Copies the file old.top on drive y: to the file new.typ on drive x:, using parameters p.

**x.new.typ=y:old1.top[p],z:old2.tip[q] <cr>**

Creates a file new.typ on drive x: which consists of the contents of file old1.top on drive y: using parameters p followed by the contents of file old2.tip on drive z: using parameters q.

**x:filename.typ=dev:[p] <cr>**

Copies data from device dev: to the file filename.typ on drive x:.

**dev: =x:filename.typ[p] <cr>**

Copies data from filename.typ on drive x: to device dev:.

**dst: =src:[p] <cr>**

Copies data to device dst: from device src:.

## PIP PARAMETER SUMMARY

---

**B**

Specifies *block mode* transfer.

**Dn**

Deletes all characters after the "n"th column.

|        |                                                                 |
|--------|-----------------------------------------------------------------|
| E      | Echoes the copying to the console as it is being performed.     |
| F      | Removes form feed characters during transfer.                   |
| Gn     | Directs PIP to copy a file from user area n.                    |
| H      | Checks for proper Intel Hex File format.                        |
| I      | Ignores any :00 records in Intel Hex File transfers.            |
| L      | Translates upper-case letters to lower-case.                    |
| N      | Adds a line number to each line transferred.                    |
| O      | Object file transfer (ignores end-of-file markers).             |
| Pn     | Issues page feed after every "n"th line.                        |
| Qs ^ Z | Specifies quit of copying after the string "s" is encountered.  |
| R      | Directs PIP to copy from a system file.                         |
| Ss ^ Z | Specifies start of copying after the string "s" is encountered. |
| Tn     | Sets tab stops to every "n"th column.                           |
| U      | Translates lower-case letters to upper-case.                    |
| V      | Verifies copy by comparison after copy finished.                |
| W      | Directs PIP to copy onto a R/O file.                            |
| Z      | Zeros the "parity" bit on ASCII characters.                     |

**PIP DESTINATION DEVICES**

---

|      |      |      |                     |
|------|------|------|---------------------|
| CON: | PUN: | LST: | Logical devices     |
| TTY: | PTP: | LPT: |                     |
| CRT: | UP1: | UL1: | Physical devices    |
| UC1: | UP2: |      |                     |
|      | OUT: | PRN: | Special PIP devices |

**PIP SOURCE DEVICES**

---

|      |      |                  |                     |
|------|------|------------------|---------------------|
| CON: | RDR: | Logical devices  |                     |
| TTY: | PTR: |                  |                     |
| CRT: | UR1: | Physical devices |                     |
| UC1: | UR2: |                  |                     |
| NUL: | EOF: | INP:             | Special PIP devices |



## REN Command Line

---

**REN newname.typ=oldname.typ <cr>**

Finds the file oldname.typ and renames it newname.typ.

## SAVE Command Line

---

**SAVE nnn x:filename.typ <cr>**

Save a portion of the Transient Program Area of memory in the file filename.typ on drive x: where nnn is a decimal number representing the number of pages of memory. x: is the optional drive specifier.

## STAT Command Lines

---

**STAT <cr>**

Displays attributes and amount of free space for all diskette drives accessed since last warm or cold start.

**STAT x: <cr>**

Displays amount of free space on diskette in drive x:

**STAT x:filename.typ <cr>**

Displays size and attributes of file(s) filename.typ on drive x:. filename.typ may be ambiguous. x is optional; if omitted, currently logged drive is assumed.

**STAT x:filename.typ \$atr <cr> (CP/M and newer)**

Assigns the attribute atr to the file(s) filename.typ on drive x:. filename.typ may be ambiguous. x: is optional; if omitted, currently logged drive is assumed.

**STAT DEV: <cr>**

Reports which physical devices are currently assigned to the four logical devices.

**STAT VAL: <cr>**

Reports the possible device assignments and partial STAT command line summary.

**STAT log: =phy: <cr>**

Assigns the physical device phy: to the logical device log: (may be more than one assignment on the line; each should be set off by a comma).

**STAT USR: <cr> (CP/M 2.0 and newer)**

Reports the current user number as well as all user numbers for which there are files on currently logged disks.

**STAT x:DSK: <cr> (CP/M 2.0 and newer)**

Reports the characteristics of disk drive x:.

**STAT x: =R/O <cr> (CP/M 1.4 and newer)**

Assigns a temporary write-protect status to drive x:.

## SUBMIT Command Lines

---

**SUBMIT filename <cr>**

Creates a file \$\$\$SUB which contains the commands listed in filename.SUB; CP/M then executes commands from this file rather than the keyboard.

**SUBMIT filename parameters <cr>**

Creates a file \$\$\$SUB which contains commands from the file filename.SUB; certain parts of the command lines in filename.SUB are replaced by "parameters" during creation of \$\$\$SUB. CP/M then gets commands from this file rather than the keyboard.

## **SYSGEN Command Line**

---

**SYSGEN** <cr>

Loads the SYSGEN program to transfer CP/M from one diskette to another.

## **TYPE Command Line**

---

**TYPE** x:filename.typ <cr>

Displays the contents of file filename.typ from drive x: on the console.

## **USER Command Line**

---

**USER** n <cr>

Sets the User Number to "n", where "n" is an integer decimal number from 0 to 15, inclusive.

## **x: Command Line**

---

**x:** <cr>

Changes the currently logged disk drive to drive x:. x can be A through P.

## CP/M Prompts

- x> CP/M waiting for command; drive x is currently logged drive.
- nx> MP/M waiting for command; drive n is currently logged drive; current user number is n.
- PIP waiting for command.
- ED waiting for command.
- nnn:• ED waiting for command; character pointer is at line number nnn.
- Also used by Microsoft BASIC, EDIT, FORTRAN, COBOL, and Pascal when waiting for a command.
- DDT waiting for command.

In the space below, write in the prompts of other programs you use.

## Sources of CP/M Compatible Languages, Programs, and Computers

---

### Word Processing Programs

|                 |                                   |
|-----------------|-----------------------------------|
| Magic Wand      | Small Business Applications, Inc. |
| Microsoft Edit  | Microsoft, Inc.                   |
| Electric Pencil | Michael Shrayer Software, Inc.    |
| SCOPE           | Vector Graphic, Inc.              |
| WordStar        | MicroPro International            |

### Languages

|                                |                        |
|--------------------------------|------------------------|
| C                              | Whitesmiths, Ltd.      |
| CBASIC                         | Compiler Systems, Inc. |
| CBASIC2                        |                        |
| CRUN (CBASIC run-time module)  |                        |
| XREF (CBASIC debugging module) |                        |
| CIS COBOL                      | MicroFocus, Inc.       |

Microsoft BASIC  
BASCOM (Compiler)  
MBASIC (Interpreter)  
L-80 (Linking Loader)

FORTRAN-80  
(Microsoft FORTRAN)

Pascal Z

FORTH

PL/I-80

Microsoft, Inc.

Microsoft, Inc.

Ithaca InterSystems, Inc.

Forth, Inc.

Digital Research, Inc.

**Data Management Systems**

SELECTOR-IV

Pearl

HDMS

Micro.AP, Inc.

Computer Pathways Unlimited

Micro Data Base Systems

**CP/M Derivative Operating Systems**

CDOS

SDOS

TPM

TSA/OS

Cromemco, Inc.

SD Systems

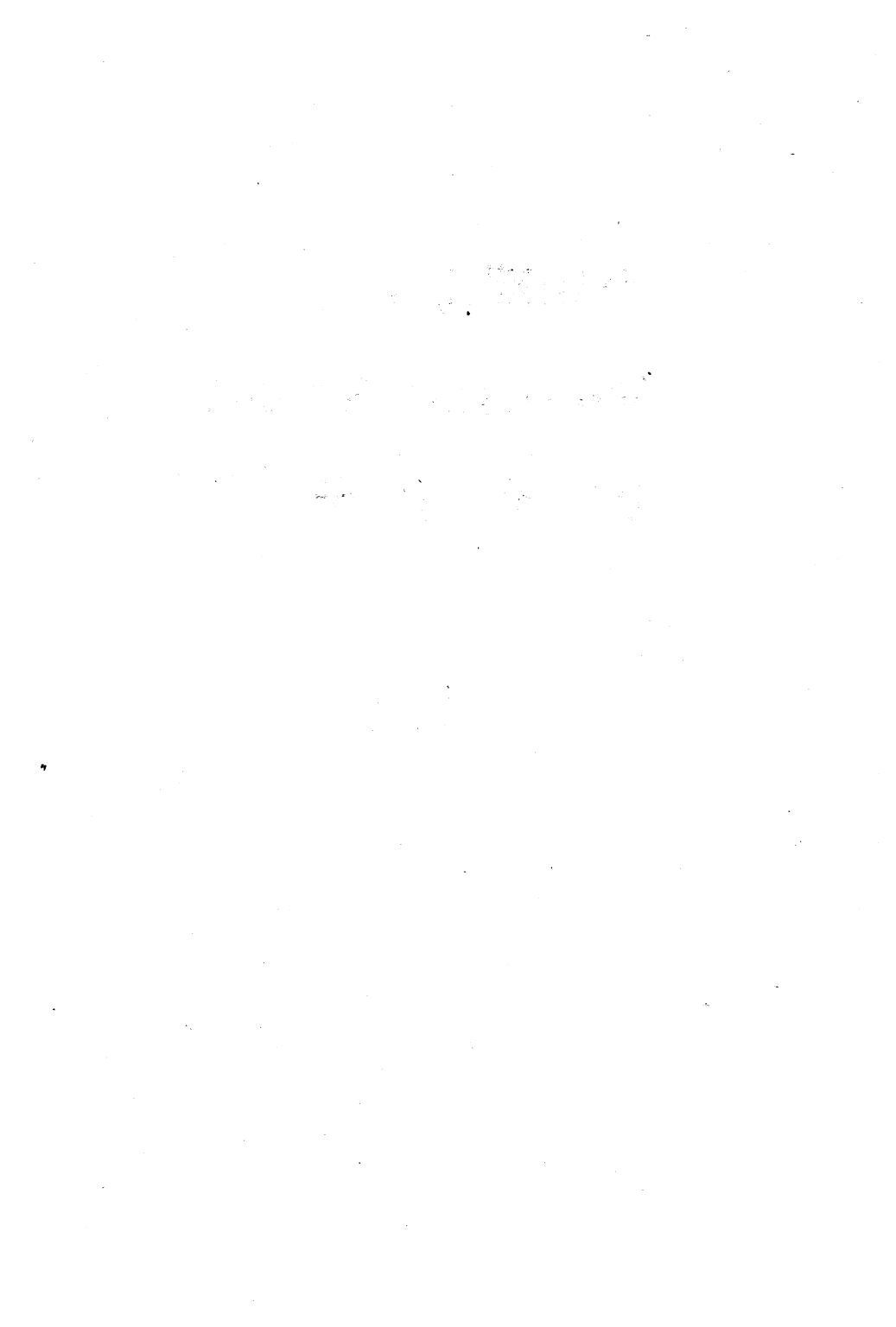
Computer Design Labs

TSA Software

**第四部份：**

**Microsoft BASIC**

**参 考 手 册**



# 目 錄

## 第四部分：Microsoft BASIC 參考手冊

### 簡介

#### 第一章：Microsoft BASIC-80與Applesoft 之比較 1

- Microsoft BASIC所獨具而為Applesoft 所無者 1
- 擴充為Applesoft 功能之處 4
- Microsoft BASIC 與Applesoft 功能相異之處 5
- BASIC-80 特性的差別 5
- Applesoft 內具有而MBASIC 不具有之特性 6

#### 第二章：BASIC-80概要 7

#### 第三章：BASIC-80的命令與陳述 29

#### 第四章：BASIC-80功能函數 97

#### 第五章：高解度析圖形——GBASIC 129

#### 附錄A：BASIC-80 5.0的新增特點 135

#### 附錄B：BASIC-80 磁碟I/O 139

#### 附錄C：組合語言副程式 151

#### 附錄D：由非Applesoft 的BASIC 程式轉換成 159



## BASIC-80程式

附錄E：錯誤碼與錯誤訊息 161

附錄F：數學功能函數 169

附錄G：ASC II字元碼 171

## 第五部份：軟體公用程式手冊 173

簡介 174

格式記號

準備磁碟使其能夠做讀寫的工作：FORMAT 175

複製磁碟：COPY 179

創建CP/M系統磁碟 181

用16段磁碟的CP/M存取13段磁碟的CP/M:RW13 183

建構CP/M成爲56K系統：CPM56 184

將Apple DOS的檔案轉到CP/M上：APDOS 186

建構Apple CP/M作業系統的環境設施：CONFIGIO 190

1. 爲外來終端機建構CP/M
2. 重定義鍵盤字元
3. 載入使用者的I/O軟體推動程式
4. 讀/寫I/O架構塊

從別的計算機把CP/M檔案傳送過來：DOWNLOAD及UPLOAD

# 第一章

## Microsoft BASIC-80

### 與 Applesoft 之比較

Microsoft BASIC-80第5版，包含有許多 Applesoft 所沒有的特點，也有一些性質與 Applesoft 不太相同。在此我們假定會購買 SoftCard 的使用者一定有撰寫 Applesoft BASIC 之經驗，因此一開始我們就列出一表來比較此兩種 BASIC 語言之差異。

如果你注意到這些差異並能使用 BASIC-80 所提供的新特性的話，將可對你的 BASIC 程式威力的增加大有助益。

#### Microsoft BASIC所獨具的特性而為Applesoft所無者

以下特性僅為 Microsoft BASIC 所獨有。此處只做簡要的描述，詳細的資料如語法、目的及每一點特殊之處請參看第 2 及第 3 章。

**CHAIN 與 COMMON** 用來從磁碟呼叫程式並把變數傳給此程式。此項特性使得磁碟可被當作程式記憶體來用。

**CALL** 可用來呼叫 6502 或 Z-80 組合語言副常式或 FORTRAN 副常式。

**PRINT USING** 它使得輸出時規定格式很容易，大大增

## 2 APPLE 軟體卡

加撰寫程式之便利(詳見第3章)。

內藏( built )的磁碟輸出入陳述 因為 Applesoft BASIC 及整數 BASIC 都不是為磁碟而設計, 因此在使用磁碟時都需把 DISK I/O 命令放在 PRINT 的陳述中。Microsoft BASIC 則不需要, 它有自己內藏( built-in) DISK I/O 陳述。

WHILE/WEND 使 BASIC 具有一些“結構化”程式的味道。在一程式迴路( loop )之頭放一個 WHILE 而在其尾放一個 WEND 陳述。只要 WHILE 後接的條件為真, 迴路就會連續不斷的執行。

EDIT 命令 讓你能很容易且有效率地編修各別的程式行, 而不需重新打入整行資料。

AUTO 與 RENUM 就是讓你自動的以自己定義的增量重新把每行編號, 這樣可使編修及除錯時更容易些。AUTO 則是在每一個 Carrige return 後自動產生行編號。

IF...THEN... 比 Applesoft 增加了 ELSE (即多了否定狀態的處理)。

ANSI 相容性 Microsoft 5.0 BASIC 符合了 ANSI BSRX3.60-1978 文獻中對 BASIC 之要求。這就是說在 Apple 上所發展出來的 Microsoft

BASIC 程式也可在其它符合 ANSI BASIC 標準的其它機器上執行。

- 可編譯性** ( Compilability )      Microsoft 已經開發了一套 BASIC 編譯程式 ( compiler )，它能把 BASIC 及 GBASIC 程式編譯成 Z-80 的機器碼 ( machine-code )。此編譯程式已上市但不包括在 SoftCard 套件中 ( 其執行速度比 Applesoft interpreter 快很多 )。
- 威力強大的資料型態**      BASIC 5.0 有三種變數型態——①快速雙拜整變數，②單準變數及③倍準變數——它可達 16 位有效數字而 Applesoft 只能達 9 位有效數字。還有 16 進位及 8 進位之常數也可使用。
- 增加字串函數**      提供有 INSTR，HEX\$，STRING\$ 及用 MID\$ 來直接指定副字串的功能。
- 增加運算子**      新的布林運算子：AND，OR，XOR，IM 及 EQV 都有提供，另外還有整數除法及 MOD 運算子等。
- 使用者定義之函數**      BASIC-80 使用者定義之函數允許使用多引數 ( multiple arguments )。
- 檔案保護**      BASIC 程式可以 2 進位格式保存 ( save ) 下來以達到保護目的。詳見第 3 章 SAVE 陳述。

## 4 APPLE 軟體卡

下面要介紹Microsoft BASIC 4個新的特性，都特別利用了Apple 機器的獨一特性，它們是：

**BUTTON(0)** 此功能可用來決定一按鈕是否被按下。

**BEEP** 此陳述可產生一所指定音高 ( Pitch ) 及持續時間的聲音。

**HSCRN(X,Y)** 此功能用來檢定一點是否已繪在一高解析度螢幕上的特定位置。

**VPOS(0)** 此功能可送回游標之垂直位置。

### Applesoft功能的擴充

MBASIC及GBASIC 都提供有低解析度圖形、聲音、游標控制及其它Applesoft BASIC 之特性。使用16段落 ( sector ) 型磁碟之Microsoft BASIC 版本也提供除了DRAW, XDRAW , SCALE 及 ROT 之外所有Applesoft 之高解析度圖形功能。

在MBASIC 與GBASIC 中與Applesoft 相容 ( compatible ) 之陳述與功能如下所示 ( 附有“\*”記號的陳述表示僅在GBASIC 中才有 )：

GR  
COLOR  
PLOT  
VLIN  
HLIN  
SCRN  
POP  
HGR\*  
HCOLOR\*  
HPLOT\*  
TEXT  
HTAB  
VTAB  
INVERSE  
NORMAL  
PDL(0)

## Microsoft BASIC與Applesoft功能相異之處

某些陳述及命令在MBASIC 及 Applesoft 中使用法稍有不同。在撰寫 BASIC-80 程式時必須注意到這些差異，有差異的陳述或命令列於下（詳細資料請參閱本手冊 2，3 章）：

```
FOR ... NEXT  
INPUT  
ON ERROR GOTO  
RESUME  
TEXT  
GR  
HGR  
IF ... THEN ... ELSE  
CALL
```

## BASIC-80 特性的差別

BASIC-80 的 SoftCard 版與正規的 CP/M Microsoft BASIC 稍有差別，如果你曾使用 CP/M內之 Microsoft BASIC，下面的改變必需注意：

- TRON/TROFF** 此陳述之名稱被改成 TRACE/NOTRACE，而功能不變。
- DELETE** 名稱改成 DEL，功能不變。
- WIDTH** 除了行寬（line width）之外也可選擇螢光幕顯示之高度。同樣的，Apple 螢光幕寬度自然值（default value）為 40 行，外部終端機則為 80 行。
- WAIT** 現被用來監督位址的狀態而非機器輸入埠（

## 6 APPLE II 軟體卡

port ) 之狀態，而其功效 ( effect ) 仍然不變。

CLOAD 未提供。

CSAVE 未提供。

NULL 未提供。

INP 未提供。

OUT 未提供。

註：BASIC-80 的程式若要傳送到 Apple 必需以 ASCII 格式 ( 亦即使用 SAVE 陳述時需用 A 選擇 ) 而不可以二進位格式。

### Applesoft 內具有而 MBASIC 不具有之特性

以下特性為 Applesoft BASIC 所具有，而 MBASIC 內所無的：

|                        |          |
|------------------------|----------|
| FLASH                  | SHLOAD   |
| EBC A, B, C, D 螢光幕編修功能 | XDRAW    |
| STORE                  | DRAW     |
| RECALL                 | SCALE    |
| IN#                    | 錄音帶 LOAD |
| PR#                    | 錄音帶 SAVE |
| HIMEM...LOMEM          | ROT      |

## 第 二 章

# BASIC-80概 要

### 啓始

MBASIC 是Microsoft BASIC 的CP/M 版，它包含了所有的 Applesoft 標準擴充功能（除了高解析度繪圖功能之外）。在 SoftCard 套件中，它能裝在13 段落（sector）型或16 段落型的磁碟上。在此兩種型態磁碟上的檔名稱都叫做MBASIC.COM。又MBASIC 上所有的啓始設定（initialization）命令都可用到GBASIC 上去，只要把原先是MBASIC 之處改成GBASIC 即可（對啓始設定GBASIC 的特別指令詳見第5章）。

要載入及執行Microsoft BASIC-80，只要先以一般方法啓始（bring up）CP/M 作業系統，等A>催促信號出現後打入：

MBASIC

再打入RETURN。幾秒後，一些版權聲明會出現在螢光幕上，此時就可輸入命令了。

這時所開的檔有3個來供執行BASIC 使用（見以下命令行內之 / F，可使用的記憶體為從開始處到CP/M 內FDOS 的起點為止（見以下命令行內之 / M）及設定最大資料錄之大小為128。

如果想在啓始設定後馬上自動執行任何程式或是想指定與上一段不同的系統資源，可打入下面的命令行格式：

MBASIC [ < 檔名稱 > ] [ / F: < 檔數目 > ] [ / M < 記憶



體最高位址>〕〔/ S: <資料錄最大長度>〕

再打入 RETURN

加上<檔名稱>這一項讓你能夠在啓始設定完後自動執行此檔名稱所代表之程式。如果檔名稱內沒有副檔名（即延伸字元）來指定檔的類型時則被當做是 .BAS 型。檔名稱不得長於 9 個字元。這種格式可使得一些 BASIC 程式能以 CP/M 中 SUBMIT 功能來做整批處理。這些程式在執行完其中之一時必須具有一個 SYSTEM 陳述（見第 3 章）才能回到 CP/M，然後使得下一程式繼續執行。

/ F: <檔數目>，此項有無均可，它指定了在一 BASIC 程式執行之時所開啓檔的數目。由此方式配置的每一檔案資料塊需要 166 拜加上 128 拜（此可由 / S: 指定為其它數目）的記憶體空間。如果此項被省略，其自然值（default value）定為 3。檔數目可拿 10 進位，8 進位（前加 & O）及 16 進位（前加 & H）來表示。

/ M: <最高記憶體位置>，此項可有可無，它設定了一 MBASIC 程式所能使用記憶體最高的位址。在某些情況，所使用記憶體空間要在 CP/M 之 FDOS 起始位置以下許多才可，以便保留一些空間給組合語言副程式。而在任何情況下，所使用記憶體最大位置不得超過 FDOS 的起始位址（此位址存放於位址 6 及 7），如果此項省略，那麼所能使用的空間就到 FDOS 之起始位置為止。此項所使用之數字可為 10 進位，8 進位（前加 & O）16 進位（前加 & H）。

/ S: <資料錄最大長度>此項可有可無，它設定了隨機檔的最大資料錄長度，可為任何整數，也可以大於 128。

當 BASIC 啓始設定完畢，螢光幕會顯示：

BASIC-80 Version 5.××

( Appie CP / M Version )

Copyright 1980 (c) by Microsoft

```
Created : dd-mm-yy  
xxxx Bytes free  
OK
```

以下是幾個不同的例子：

```
A > MBASIC PAYROLL.BAS
```

使用到 FDOS 起始位置以下所有記憶空間，可啓開 3 個檔，載入並執行 PAYROLL.BAS

```
A > MBASIC INVENT/F:6
```

記憶空間使用同上，開啓 6 個檔，載入並執行 INVENT.BAS

```
A > MBASIC /M:32768
```

使用記憶體前 32 K，開啓 3 個檔。

```
A > MBASIC DATAK/F:  
2 / M: &H9000
```

使用記憶體前 36 K，開啓 2 個檔，載入並執行 DATAK.BAS

## 操作模式

當 BASIC-80 啓始設定完畢後，它會顯示“Ok”，“Ok”代表 BASIC-80 在命令層次，亦即它已準備好接受命令。此時，BASIC-80 能使用下面兩種模型之一：直接模式 ( direct mode ) 或間接模式 ( indirect mode )。

在直接模式中，BASIC 命令或陳述 ( statements ) 前並不加有行號，只要輸入後馬上執行，算術及邏輯運算結果馬上顯示並儲存

## 10 APPLE 軟體卡

下來備用，但是命令本身在執行後就消失了，此種模式用來除錯或把 BASIC 命令當作計算器 ( calculator ) 來使用 ( 不需完整的程式 ) 時是非常的好用。

間接模式則是用來輸入程式，程式的行號在每一程式陳述之前，並存放在記憶體當中，打入 RUN 命令就可執行此存放在記憶體中之程式。

例：直接模式

```
A = ( 45 + 38 * 50 / 2 ) * COS ( 0.5 )
```

```
OK
```

```
PRINT A
```

```
873.195
```

```
OK
```

例：間接模式

```
110 X=45  
120 Y=38*50/2  
130 Z=(X+Y)*COS(.5)  
140 PRINT Z  
150 END  
OK
```

```
RUN
```

```
873.195
```

```
OK
```

## 磁碟檔

磁碟檔命名法與正規 CP/M 命名法則相同，所有的檔名稱前可以冠以 A: , B: , C: , D: , E: , F: , 等二個字元來指定使用那架磁碟機，若省略則被認為是現正指定使用的那架。磁碟機名稱必須大寫（如，是 A: 而非 a: ），如果檔名稱內沒有類型延伸字元〔在第 3 部份稱為副檔名（secondary file name）〕且其長度小於 9 個字元，在執行 LOAD, SAVE, MERGE 及 RUN 等命令時此種檔名稱會被認為是 .BAS 檔。

## 陳述之格式

在 BASIC 程式內每條陳述有如下之格式（方弧號代表其內之資料可有可無）：

```
nnnnn BASIC statement [:BASIC statement...] <carriage return>
```

程式員可以選擇在一行內放入許多 BASIC 陳述，但每一陳述之間需以冒號（:）隔開。

```
310 X=45:Y=38*50/2
320 Z=(X+Y)*COS(.5)
330 PRINT Z
340 END
OK

RUN

873.195
OK
```

每列陳述的前面是行號，後面是回車字元（carriage return），中間可以最多含有 255 個字元。

在螢幕每行之最終了若按下饋行字元 < line feed > 或 < control J > , 則下一行之資料仍與其上一行屬於同一行, 如此可延長每一行的實際長度。

## 行號

每一 BASIC 程式陳述前有一行號, 它用來指示程式儲存在記憶體的次序, 也用來當做分支 ( branching ) 及編修 ( editing ) 時之參考。行號必須在 0 到 65529 之間。在 EDIT, LIST, AUTO 及 DELETE 命令中可用句點 ( . ) 來表示參考行為現在游標所指的行。

## 字元組 (CHARACTER SET)

BASIC-80 字元組是由字母字元, 數字字元及特殊字元所組成。BASIC-80 之字母字元包括大寫字母及小寫字母, 數字字元從 0 到 9, 可辨識之特殊字元如下:

| 字元 | 名         | 稱 |
|----|-----------|---|
|    | 空白        |   |
| =  | 等號或指定符號   |   |
| +  | 加號        |   |
| -  | 減號        |   |
| *  | 星號或乘號     |   |
| /  | 斜線或除號     |   |
| ↑  | 向上箭號或指數符號 |   |
| (  | 左括號       |   |
| )  | 右括號       |   |

- % 百分號
- # 數目符號或磅符號
- \$ 金錢符號
- ! 驚嘆號
- [ 左中括號
- ] 右中括號
- 逗點
- . 句號或小數點
- ' 引號
- ; 分號
- : 冒號
- & Ampersand
- ? 疑問號
- < 小於
- > 大於
- \ 倒斜線或整數除法記號
- @ At-記號
- 底線
- <rubout> 刪除最後打入的字
- <escape> 從編輯模式副命令中跳開(見2.16節)
- <tab> 把印出位置跳到下一個tab位置。  
Tab每次跳8格
- <carriage 一行輸入的結束  
return>

## 控制字元

下面是 BASIC-80 內所使用的控制字元

- Control @ 相當於 Rubout
- Control-A 在打入的那行進入編輯模式中。
- Control-B 倒斜線
- Control-C 中斷程式的執行並回到 BASIC-80 命令中。
- Control-G 使終端機的鈴響。
- Control-H 往回跳，刪去最後一字，與“←”相同。
- Control-I 相當於 Tab，每次跳 8 格，與“→”相同。
- Control-J 饋行，即跳到下一行。
- Control-K 右中括號
- Control-O 繼續執行程式但停止程式輸出，再打一次 Control-O 即行恢復輸出。
- Control-R 把現今打入之整行資料再重打一行。
- Control-S 使程式執行停滯住。
- Control-Q 在使用 Control-S 後恢復程式執行。
- Control-X 刪除整行陳述。

Control-Y 在有自動啓始ROM(AUTOMATIC ROM)的系統，已按下RESET鍵後，可用此鍵恢復。

→ Tab, 和Control-I 相同。

← 後退鍵，和Control-H 相同。

註：Control@, Control-B, Control-K 及 Control-U 能用CONFIGIO 程式重新定義。

## 常數

有兩種類型的常數：字串常數及數字常數。

一字串常數可以有 255 文數字字元長，這些字元包含在雙引號內。字串常數的例子如下：

```
"HELLO"  
"$25,000.00"  
"Number of Employees"
```

數字常數可以是正數或負數，但不能含有逗點，有 5 種類型的數字常數：

1. 整數常數 值介於 - 32768 及 + 32767，不能有小數點。
2. 定點常數 正或負實數，亦即含有小數點之數。
3. 浮點常數 正或負的定點常數（正號可省）附上一個字母 E 再隨著正或負整數（正號可省）以代表指數



## 16 APPLE 軟體卡

指數範圍從 - 38 到 + 38 之間。

例子：

```
235.988E-7 = .0000235988
2359E6 = 2359000000
```

註：倍準 ( Double precision ) 浮點常數  
使用字母 D 而非 E，見下節。

4. 16 進位常數 16 進位常數帶有 &H 字頭。例如：

```
&H76
&H32F
```

5. 8 進位常數 8 進位常數帶有 &O 或 &字頭，如：

```
&O347
&1234
```

### 數字常數之單準及倍準型式

數字常數可以是單準 ( single precision ) 或倍準 ( Double precision )。倍準可以裝下 16 個有效數字，並可以 16 數字形態印出。

單準常數是：

1. 7 個或更少的數字，或
2. 帶有 E 的指數型式，或
3. 常數尾帶有驚嘆號 ( ! ) 的數

倍準常數是：

1. 8 個以上的數字，或

- 2 帶有D的指數型式，或
- 3 常數尾帶有( # )號的數

例子：

| 單準常數      | 倍準常數         |
|-----------|--------------|
| 46.8      | 345692811    |
| -7.09E-06 | -1.09432D-06 |
| 3489.0    | 3489.0#      |
| 22.5!     | 7654321.1234 |

## 變數

變數是用來代表數值的名字。變數的名字可以由程式員明確的指定，或是經由計算的結果而指定其值。在變數被指定一值之前其值為零。

### 變數的名稱及宣告字元

BASIC-80 的變數名稱長度可以為任意長度；而直到 40 個字元以前都是有效的。變數名稱內可以是字母，數字或小數點，但最前面必須為字母。

變數名稱不可以是保留字 ( reserved word )，如果一變數以 FN 為起始，它就會被假定是在呼叫一使用者定義函數。保留字包括所有 BASIC-80 命令，陳述，函數名稱及運算子 ( operator ) 名稱。

變數可以代表一數值或字串。若是字串變數，則在變數名稱最後一個字元加上金錢符號 ( \$ )。例如：A\$="SALES REPORT"。金錢符號 \$ 是一種變數型態宣告字元，亦即它宣告此變數代表一字串

## 18 APPLE 軟體卡

。若是數值變數，又有整數，單準實變數及倍準實變數，這些變數型態宣告字元如下：

% 整數變數  
! 單準實變數  
# 倍準實變數

而型態之自然值 ( default value ) 定為單準實變數。

BASIC-80 變數名稱的例子如下：

PI# 宣告為倍準實變數  
MINIMUM! 宣告為單準實變數  
LIMIT% 宣告為整數變數  
N\$ 宣告為字串變數  
ABC 代表單準實變數

宣告變數還有另一種方法：可以用 BASIC-80 內之陳述 DEFINE, DEFSTR, DEFSNG 及 DEFDBL 來宣告變數的類型，這些陳述留待第 3 章詳述。

### 陣列 ( ARRAY ) 變數

陣列就是一群數值或一表內元素以相同的變數名稱來指定。在陣列中的每一元素則由相同變數名稱之後的整數 ( 或整數表式 ) 足碼來指定，而陣列變數名稱內之足碼數目隨著陣列的維數 ( dimension ) 而定。例如：V(10) 是在某一個一維陣列中。T(1,4) 是在二維陣列中的一元素，等等。陣列可以有的最大維數為 255，每一維數最多容許元素數量為 32767 個。

## 型態轉換

當需要時，BASIC 能夠把一數值常數從一種型態轉換成另一種以下的規則及例子應熟記於心：

1. 如果某一型態的數值常數要轉變成另一種型態的數值常數，可在指定變數型態時定為另一種型態即可。（但如果把一數值指定給字串變數或把一字串指定給數值變數則會發生“型態不符”的錯誤信號）。

例：

```
10 A% = 23.42
20 PRINT A%
RUN
23
```

2. 在計算表式之值時，在算術或關係運算裏所有的運算元（operand）都會轉換到相同的精確度〔precision〕（與精確度最高的運算元相同）。而算術運算的結果也具有此相同之精確度。

例：

```
10 D#=6#/7          此算術運算以倍準方式運算，其結果
20 PRINT D#         以倍準形式存到D #變數中。
```

RUN

```
.8571428571428571
```

```
10 D=6#/7          此算術運算以倍準方式運算，但其結
20 PRINT D         果以單準變數儲存，於是有些有效數
RUN               字就要被4捨5入了
```

```
.857143
```

3. 邏輯運算子 ( operator ) 先把運算元轉換成整數並把結果以整數型態送回。運算元必須在 -32768 及 32767 之間否則會有溢位 ( overflow ) 之錯誤發生。
4. 當一浮點值 ( floating point value ) 要被轉換成整數時，小數部份將被四捨五入：

例：

```
10 C% = 55.88
20 PRINT C%
RUN
56
```

5. 當一單準數值被指定給一倍準變數時，只有最前面的 7 位有效數字 ( 4 捨 5 入後 ) 可以被正確地轉移，此乃單準數值之有效數字只有 7 位而已。把一單準數值以倍準方式儲存所造成的誤差是原數值之  $6.3 \text{E}-8$  倍以下。

例：

```
10 A = 2.04
20 B# = A
30 PRINT A;B#
RUN
2.04 2.039999961853027
```

## 表式及運算子

一表式 ( expression ) 可以僅是一字串常數，數值常數，變數，或者是常數與變數經過運算而得到的某一單一數值。而運算子 ( operator ) 則可執行數值之數學或邏輯運算。BASIC-80 所提供

的運算子可分成下面四個範疇：

1. 算術運算子
2. 關係運算子
3. 邏輯運算子
4. 函數運算子

### 算術運算子

依運算之優先次序之不同，算術運算子可排列如下：

| 運算子  | 運 算       | 表式例子        |
|------|-----------|-------------|
| ↑    | 指數        | $X^Y$       |
| -    | 負值        | $-X$        |
| *, / | 乘法, 浮點數除法 | $X*Y$ $X/Y$ |
| +    | 加法        | $X+Y$       |

如果表式內有括號，則括號內之運算優先。在括號內運算子優先次序仍如上所示。以下是一些代數表式及其在 BASIC 內之表式對照：

| 代數表式            | BASIC 內之表式 |
|-----------------|------------|
| $X+2Y$          | $X+Y*2$    |
| $X-\frac{Y}{Z}$ | $X-Y/Z$    |
| $\frac{XY}{Z}$  | $X*Y/Z$    |
| $\frac{X+Y}{Z}$ | $(X+Y)/Z$  |
| $(X^2)Y$        | $(X^2)*Y$  |
| $XY^Z$          | $X*(Y^Z)$  |
| $X(-Y)$         | $X*(-Y)$   |

由  $X * (-Y)$  可知，若兩運算子連續出現則需以括號將它們分開。

### 整數除法與整數餘數算術 (Modulus Arithmetic)

整數除法與整數餘數算術運算是 BASIC-80 所新增的兩個運算子。

整數除法是以倒斜線 ( \ ) 來表示 ( 在 Apple 鍵盤上以 Control - B 來表示。它的運算元事先被四捨五入成 32768 到 32767 間的整數，除後其商被切成整數 [ 小數部份被切除 ( Truncated ) ] ) 。

例：

$$\begin{aligned} 10 \setminus 4 &= 2 \\ 25.68 \setminus 6.99 &= 3 \end{aligned}$$

整數除法的運算優先次序在乘法及浮點數除法之後。

整數餘數運算以 MOD 來表示，它表示整數除法運算後之整數餘數。例：

$$\begin{aligned} 10.4 \text{ MOD } 4 &= 2 \text{ (} 10/4=2 \text{ with a remainder } 2 \text{)} \\ 25.68 \text{ MOD } 6.99 &= 5 \text{ (} 26/7=3 \text{ with a remainder } 5 \text{)} \end{aligned}$$

MOD 的運算優先次序在整數除法之後。

### 溢位及除數為 0 時

如果在表式運算時遇到除數為 0 的情形，螢光幕上就會顯現 “Division by zero” 之錯誤訊號，除的結果是機器上無窮大值 ( machine infinity ) 其正負號與分子相同，運算仍將進行 ( 但結果一定不對 )。若除數過小 ( 指數為負且很大 )， “Division by zero” 之錯誤訊號也會出現，此時運算結果是機器正的無窮大值，而運算仍繼續執行 ( 但結果亦不正確 )。

如果溢位情況發生，螢光幕會出現 “Overflow” 之錯誤訊號，結果是帶有正確正負號之機器無窮大值，運算仍將繼續 ( 結果不正確 )。

## 關係運算子

關係 ( relational ) 運算子可用來比較兩個數值，比較的結果以“真” ( -1 ) 或“假” ( 0 ) 來表示。它之結果可以用來決定程式流程之方向。(見第3章 IF 陳述)

| 運算子 | 所測試之關係 | 表式例子     |
|-----|--------|----------|
| =   | 相等     | $X = Y$  |
| <>  | 不相等    | $X <> Y$ |
| <   | 小於     | $X < Y$  |
| >   | 大於     | $X > Y$  |
| <=  | 小於或等於  | $X <= Y$ |
| >=  | 大於或等於  | $X >= Y$ |

(等號也可用來指定一數值給一變數，見第3章 LET 陳述)。

在一表式內若同時有算術運算子及關係運算子，算術運算必須先做，例如：

$$X+Y < (T-1)/Z$$

其意為如果  $X + Y$  的值小於  $(T - 1) / Z$  之值則此表式為真。

同理讀者可試試下面例子：

```
IF SIN(X)<0 GOTO 1000
IF I MOD J <> 0 THEN K=K+1
```

## 邏輯運算子

邏輯運算子可用來做多重關係的測試、比次 ( bit ) 處理及布林 ( Boolean ) 運算。邏輯運算是以比次為單位來運作，其結果以“真



## 24 APPLE 軟體卡

(不為零)及“偽”(零)來表示。在一表式中，邏輯運算之優先順序是在算術運算及關係運算之後。各種邏輯運算的結果如下所示，其先後代表其優先次序之大小：

|     |   |       |  |
|-----|---|-------|--|
| NOT | X | NOT X |  |
|     | 1 | 0     |  |
|     | 0 | 1     |  |

|     |   |   |         |
|-----|---|---|---------|
| AND | X | Y | X AND Y |
|     | 1 | 1 | 1       |
|     | 1 | 0 | 0       |
|     | 0 | 1 | 0       |
|     | 0 | 0 | 0       |

|    |   |   |        |
|----|---|---|--------|
| OR | X | Y | X OR Y |
|    | 1 | 1 | 1      |
|    | 1 | 0 | 1      |
|    | 0 | 1 | 1      |
|    | 0 | 0 | 0      |

|     |   |   |         |
|-----|---|---|---------|
| XOR | X | Y | X XOR Y |
|     | 1 | 1 | 0       |
|     | 1 | 0 | 1       |
|     | 0 | 1 | 1       |
|     | 0 | 0 | 0       |

|     |   |   |         |
|-----|---|---|---------|
| IMP | X | Y | X IMP Y |
|     | 1 | 1 | 1       |
|     | 1 | 0 | 0       |
|     | 0 | 1 | 1       |
|     | 0 | 0 | 1       |

|     |   |   |         |
|-----|---|---|---------|
| EQV | X | Y | X EQV Y |
|     | 1 | 1 | 1       |
|     | 1 | 0 | 0       |
|     | 0 | 1 | 0       |
|     | 0 | 0 | 1       |

正如關係運算一般，邏輯運算也可用來聯接許多關係運算而最後根據結果之真偽來決定程式流程之方向（見第3章 IF 陳述）

例：

```

IF D<200 AND F<4 THEN 80
IF I>10 OR K<0 THEN 50
IF NOT P THEN 100
    
```

邏輯運算子在工作時，首先將其運算元轉換成 16 比次，包括正負號比次，2-Complement 之整數，其數值介於 -32768 與 32767 之間（若超過此範圍就是錯誤）。如果運算元本來就是 2 進位的 0 或 -1，邏輯運算後之結果也是 0 或是 -1。把兩個運算元化成 16 比次 2 進位形式之後每一相對應之比次再一個個分別作邏輯運算。因此，AND 可以把狀態拜（status byte）內單一的比次以“面罩”（mask）方式取出，而 OR 則可把兩個拜併合（merge）起來，見以下例子就可明白：

63 AND 16=16      因為 63=2 進位 111111, 16=2 進位 10000  
 , 所以 63 AND 16=16

15 AND 14=14      15=2 進位 1111, 14=2 進位 1110 所以  
 15 AND 14=14 (2 進位 1110)

-1 AND 8=8      -1=2 進位 1111111111111111, 而 8=2  
 進位 1000, 所以 -1 AND 8=8

4 OR 2=6      4=2 進位 100, 2=2 進位 010 所以 4 OR  
 2=6 (2 進位 110)

10 OR 10=10      10=2 進位 1010, 1010 OR 1010=1010

-1 OR -2=-1      -1=2 進位 1111111111111111 -2=2 進  
 位 1111111111111110 所以 -1 OR -2=-1

$NOTX = -(X+1)$  整數之“2的補數”為原數各比次之互補值再加上1（亦即  $-X = NDTX + 1$ ）。

## 函數運算子

在一表式內的函數是用來呼叫先前已被定義過的運算。BASIC 內存有一些內部的函數像 SQR（開平方）或 SIN（正弦函數）等。有關 BASIC-80 內部函數之資料請參看第4章。BASIC-80 也允許使用者自行定義函數，參看第3章 DEF FN。

## 字串運算

用“+”可以把數個字串連接起來，例如：

```
10 A$="FILE":B$="NAME"
20 PRINT A$ + B$
30 PRINT "NEW" + A$ + B$
RUN
FILENAME
NEW FILENAME
```

字串可以用像如下的數值關係運算子來比較：

= <> < > <= >=

字串比較法是每次從兩個字串中各拿出一個相對應之字元然後比較它們的 ASCII 碼大小。如果所有的 ASCII 碼大小通通相同，那麼這兩字串相等，如果遇到一個 ASCII 碼不同，此值較小的字串的“值”亦較小。又在比較中，若其中有一較短字串在比較完其所有字元後仍然不分勝負，那麼就以短字串的“值”為較小。又字頭與字尾之空白（space）也算是一有效字元。例：

```

"AA" < "AB"
"FILENAME" = "FILENAME"
"X&" > "X#"
"CL " > "CL"
"kg" > "KG"
"SMYTH" < "SMYTHE"
B$ < "9/12/78" where B$ = "8/12/78"

```

由上可知，字串比較可用來測試字串“值”或字母型字串。所有在表式內的字串常數必須以引號（“ ”）括起來。

### 輸入資料時之編修

如果一個資料打入後發現有一不正確之字元，可用 RUBOUT (Control-A) 或 (Control-H) 來刪除它。RUBOUT 或 Control-A 可用倒斜線把要刪除之字元包圍起來，而 Control-H 可以向左移一個字元並把此字元刪除。不要的字元刪除後，就可繼續輸入所要的資料了。

要把打入資料整行刪掉，可打 Control-X。此行刪除後會自動加入一 Carriage return 字元。

如果要修改記憶體內的程式，可將原有程式行號及正確陳述一併打入。此時 BASIC-80 就會自動的把記憶體原有的那條陳述改換成新打入的正確陳述。

系統也提供更複雜的編修能力，見第 3 章 EDIT。

若要把記憶體內整個程式去除，可打入 NEW 命令（見第 3 章）。NEW 經常在打入新程式以前用來清除記憶體空間。

### 錯誤訊息

如果 BASIC-80 偵得一會使程式停止執行之錯誤，此錯誤就會

## 28 APPLE 軟體卡

在螢光幕上印出來。有關BASIC-80的錯誤碼與錯誤訊息之詳情，見附錄E。

## 第 三 章

# BASIC-80的命令與陳述

此章詳述所有 BASIC-80 的命令 ( Command ) 與陳述 ( Statement ) , 每一個命令與陳述依下列四個要點來分析說明 :

- 1 語法 ( Syntax ) : 顯示陳述的正確語法。
- 2 目的 ( Purpose ) : 說明陳述執行後的結果。
- 3 註釋 ( Remark ) : 敘述陳述在使用時應注意的詳細事項。
- 4 範例 ( Example ) : 舉例說明陳述的使用方法。

### 語法符號 ( Syntax Notation )

陳述語法的敘述必須遵循下列幾項規定 :

- 1 陳述中的大寫字母表示必須如所示地鍵入。
- 2 使用者須提供角括號 ( <> ) 內的項目。
- 3 中括號 ( [ ] ) 內的項目, 依程式需要取捨。
- 4 除了角括號與中括號外, 所有的指令中標點 ( 逗點、括號、分號、頓號、等號 ) 不能任意取捨。
- 5 項目後的符號 ( ... ) , 表示可重覆使用該項目直到該行的終點。
- 6 被垂直線符號 ( | ) 所隔開的兩個項目, 只能選擇其中一項。

7. 所有保留字的前後，須有一空格隔開。

## AUTO

語法：AUTO[ <行號> [ , <增量> ] ]。

目的：每次游標回轉移入下行，自動產生該行的行數。

註釋：AUTO 是以<行號>裏的數目為第一行的行號，<增量>裏的數目為第一行以後每一行行號增加的量，如果 AUTO 後沒有<行號>及<增量>即表示第一行行號為 10，增量為 10，如果在<行號>之後只有逗點而不寫<增量>，此<增量>即由前次 AUTO 後所設定之<增量>取代。

假如 AUTO 後鍵入的行號先前已經使用過，這時主機便會在所鍵入的行號後出現一個星號提醒使用者任何輸入的資料將取代原先行號的資料，如果在星號後即時鍵入 RETURN 將產生下一行的行號，同時保留原先那一行的舊資料。

鍵入 Control-C 可終止 AUTO，在鍵入 Control-C 的那一行的所有資料將不會被儲存在記憶體中。當鍵入 Control-C 以後 BASIC 將返回等待輸入的狀態。

範例：AUTO 100, 50 產生的行號為 100, 150, 200, ……

AUTO 產生的行號為 10, 20, 30, 40, ……。

## BEEP

語法：BEEP<音調> , <時距>

目的：利用代號產生某些特定的音調及持續的時間。

註釋：0 代表最高音調；255 代表最低音調。

0 代表最短時距；255 代表最長時距，最長時距（255）可持續約 1 秒鐘。

BEEP 是用來增加音響效果，但並不是被設計用來產生音樂。

範例：10 BEEP PDL(0),PDL(1): GOTO 10

此例用控制桿決定音調及時距

## CALL

語法：CALL <變數名稱> [ ( <引數列 ( argument list ) > ) ]

目的：呼叫 Z-80 組合語言的副程式。

語法 2：CALL% <變數名稱> [ ( <引數> ) ]

目的：呼叫 6502 組合語言的副程式。

註釋：CALL 指令是將程式跳入組合語言副程式的一個方法。（參考第四章 USR 函數）。

<變數名稱> 內含副程式記憶體の起始點位置。但 <變數名稱> 不可為陣列變數名稱，<引數列> 包含了傳至組合語言副程式の引數。

語法 2 中，在 <變數名稱> 前置百分符號 %，可使 CALL 指令呼叫出 6502 組合語言の副程式。



## 32 APPLE 軟體卡

呼叫 6502 副程式最多能帶三個單拜的參數。第一個值放置在 A 暫存器，第二個放置在 X 暫存器，第三個放置在 Y 暫存器。

CALL 指令如用在 Microsoft's FORTRAN, COBOL 及 BASIC 的編譯程式內都產生同樣的結果。

範例：  
110 MYROUT=&HD000  
120 CALL MYROUT  
130 BELL=&HFF3A  
140 CALL % BELL

## CHAIN

語法：CHAIN [MERGE] <檔案名稱> [ , [ <行號表式> ]  
[ , ALL ] [ , DELETE <範圍> ] ]

目的：呼叫出一程式，並將現正在執行之程式的變數輸入此一程式中。

註釋：<檔案名稱> 被呼叫之程式的名稱，例如：

```
CHAIN*PROG1*
```

<行號表式> 由被呼叫出程式的此行號開始執行，如果省略此行號表示式則由程式的第一行開始執行，例：

```
CHAIN*PROG1*,1000
```

<行號表式> 不會被 RENUM 指令所影響，若使用 ALL，將會使現正在執行的程式中所有的變數送入被呼叫的程式裏，假如省略 ALL，現正執行的程式中必須有 COMMON 指令

，列出要送出的變數，例：

```
CHAIN"PROG1",1000,ALL
```

假如使用MERGE，將會使副程式被併入BASIC程式（即為overlay），即MERGE作用了正執行中的程式及被呼叫的程式。被呼叫的程式必須為ASC II 檔才能被MERGE，例

```
CHAIN MERGE"OVRLAY",1000
```

在引入overlay後，經常需要再消除，以便再引入新的overlay，這時就要用到DELETE指令。例：

```
CHAIN MERGE"OVRLAY2",1000,DELETE 1000-5000
```

RETURN指令會影響在〈範圍〉中的行號。

注意：假如MERGE被省略，CHAIN將不會保存連接程式中變數型態及使用者自設的函數，也就是說，在DEFINT，DEFSNG，DEFSTR，DEFFN等指令所共有的變數必須在連接程式中重新定義。

## CLEAR

語法：CLEAR [ , [ <表式 1 > ] [ , <表式 2 > ] ]

目的：設定所有的數值變數為零及所有字串變數為空白，而且選擇性的使用，可設定記憶體終點及堆疊空間（stack space）的大小。

註釋：〈表式 1〉是一個記憶體的位置，可設定 BASIC - 80 中可用的記憶體裏最高的位址。

〈表式 2〉設定 BASIC 的堆疊空間。如果缺此項則為 256 個拜或者為可用記憶體空間的八分之一，電腦會選用此 2 者佔位置較少者。

注意：在上一版本的 BASIC - 80 中，〈表示 1〉是設定字串空間的容量，〈表式 2〉是設定記憶體的終點位址。在沒有 BASIC - 80 的 5.0 版及其後版本中都以動態配置字串空間供 BASIC 使用，會產生“OUT OF STRING SPACE”的錯誤訊息。

範例：  
CLEAR  
CLEAR ,32768  
CLEAR ,,2000  
CLEAR,32768,2000

## CLOSE

語法：CLOSE [ # ] <檔案號碼> [ , [ # ] <檔案號碼...> ]

目的：結束掉一磁碟檔的 I/O。

註釋：〈檔案號碼〉應是已被打開的檔案號碼，沒有引數的 CLOSE 指令將關閉所有已打開的檔案。

特定檔案及檔案號碼之間的相關性在執行 CLOSE 之後解除使用相同的或不同的檔號可重新打開此檔；同樣地，此檔號在此時就可以用來開別的檔案。

循序輸出檔中 ( Sequential output file )，CLOSE 會將輸出緩衝區的資料儲存起來。

END 陳述 ( statement ) 與 NEW 命令 ( command ) 也

都會自動地執行 CLOSE 來關閉所有的磁碟檔。( STOP 不會關閉磁碟檔 )

範例：請看附錄 B。

## COLOR

語法：COLOR = <顏色號碼>

顏色號碼是從 0 到 15。

目的：設定低解析度圖形模式的顏色。

註釋：可被採用的顏色及其代號。

|                       |                  |
|-----------------------|------------------|
| 0 黑色 ( black )        | 8 棕色 ( brown )   |
| 1 紫紅色 ( magenta )     | 9 橙色 ( orange )  |
| 2 深藍色 ( dark blue )   | 10 灰色 ( gray )   |
| 3 紫色 ( purple )       | 11 粉紅色 ( pink )  |
| 4 深綠色 ( dark green )  | 12 綠色 ( green )  |
| 5 灰色 ( gray )         | 13 黃色 ( yellow ) |
| 6 青藍色 ( medium blue ) | 14 水色 ( aqua )   |
| 7 淺藍色 ( light blue )  | 15 白色 ( white )  |

<顏色號碼>也可在 GR 陳述內設定，在 GR 中如未設定 <顏色號碼>，電腦自動設為 0，直到用 COLOR 陳述及色碼設定其他顏色為止。

要找出在螢光幕中某一點的顏色代碼，則需使用 SCRN 函數。

COLOR 陳述只能用在低解析度圖形的模式。

範例：  
10 GR  
20 COLOR=13

## COMMON

語法：COMMON <變數列>

目的：傳送變數給CHAINed 程式。

註釋：COMMON 陳述和CHAIN 陳述在使用時互相關連。一般將COMMON 放在程式的開頭，其實它可放在程式中任何位置，相同的變數不能在一個以上的COMMON 陳述中出現。若變數列中有陣列變數時，則需在其後附加“( )”符號，假如所有的變數均要輸入CHAINed 程式，則可使用CHAIN 加上ALL 陳述如此可省略了COMMON 陳述。

範例： 100 COMMON A,B,C,D),G\$  
110 CHAIN "PROG3",10

## CONT

語法：CONT

目的：在鍵入Control-C，或已經執行了STOP或END 陳述後之程式，欲使程式繼續執行則使用CONT。

註釋：繼續執行中斷點以後的程式

假如是由INPUT 陳述中的prompt 而引起的中斷，將重印此prompt (“?”或是prompt 字串)而繼續執行程式。

CONT 通常與STOP 一起使用，用來檢查程式中錯誤的地方。當執行中止時，可用直接模式陳述加以檢查或改變此程式中已執行過的數值。使用CONT 或直接模式GOTO 陳述可由某特定的行號繼續執行。程式出現錯誤訊息產生的中止可

用 CONT 使其繼續執行。

中止後的程式經過修改，則 CONT 失效。

範例：請見 STOP 陳述中的範例。

## DATA

語法：DATA <常數列>

目的：與 READ 陳述配合使用，用來儲存數值及字串常數。

註釋：DATA 陳述不被執行且可置於程式中任何位置 DATA 陳述可包含一行所能容納的常數（常數間由逗點分開），一程式中可多次的使用 DATA 陳述。

READ 陳述與 DATA 陳述依行號順序相配合，資料讀入電腦先由行號較少之 DATA 開始，此行讀完再讀行號次少之 DATA 資料，不管每一 DATA 後之資料有多少及位置在程式何處，視為連續的項目順序讀入。

<常數列>可包含任何型態的數值常數，但此群不可為數值表式，只有當 DATA 陳述裏的字串常數中含有逗點，冒號或前後有空格時必須以雙引號包圍，其餘情形皆不需使用雙引號。如：DATA "TIME" , 20 , 30

READ 陳述中的變數型態必須與 DATA 陳述中常數的型態一致。

使用 RESTORE 陳述可重讀 DATA 陳述中的資料。

範例：

```

10 DIM A(8)
15 FOR I=1 TO 8
20   READ A(I)
25   PRINT A(I);
30   IF (I=3) OR (I=6) THEN RESTORE 40
35 NEXT I
40 DATA 1,2,3
45 END
OK
RUN
  1  2  3  1  2  3  1  2

```

**DEF FN**

語法：DEF FN <名稱> [ ( <參數列> ) ] = <函數定義>

目的：讓使用者自行定義函數及賦予名稱。

註釋：<名稱>必須為合法之變數名稱，此名稱前面冠以FN，意為函數的名稱，<參數列>由函數被叫出後，將被取代的變數名稱所組成。參數之間，須以逗點分開。

<函數定義>此為函數運算的表式，它的敘述內容不得超過一行，此表式中的變數名稱只能用於定義函數，並不會影響程式中有相同名稱的變數，函數定義中的變數名稱是否出現在參數列中均無所謂。如果出現，則在呼叫此函數時須先設定參數值。如果不出現，使用變數的現今值。

參數串中各變數對應於呼叫函數時所需要的各引數變數或其值都是一對一的呼應。

使用者自定的函數可以是數值或字串。

如果函數名稱指定某資料型式，則表式中的值在返回呼叫程式之前會先變成此型式，如果函數名稱中定義的型式與引數型式不配合，則會出現“Type mismatch”的錯誤訊息。

DEF FN 陳述必須在函數被呼叫之前執行，否則會出現“Undefined user function”的錯誤訊息。

DEF FN 不能使用在直接操作模式。

範例：  
410 DEF FNAB(X,Y)=X+3/Y+2  
420 T=FNAB(I,J)

410定義了FNAB，420呼叫FNAB。

## DEFINT/SNG/DBL/STR

語法：DEF <型態><字母範圍>

此處的<型態>是指 INT, SNG, DBL, 或 STR

目的：宣告變數型態：整數 ( integer ) , 單準實數 ( single precision ) , 倍準實數 ( double precision ) , 字串 ( string ) 。

註釋：DEF 陳述在此用來宣告以某字母開頭的變數名稱成為指定型態的變數，但是，型態宣告字元 ( % , ! , # 等 ) 優先於 DEF 陳述決定變數型態。  
假如沒有宣告陳述，BASIC-80 會將所有沒有宣告的變數視為單準實變數 ( Single precision variables ) 。

範例：10 DEFDBL L-P

表示所有以 L, M, N, O, P 等字母為首的變數，都是倍準實變數 ( double precision variables ) 。

10 DEFSTR A

表示所有以 A 開頭的變數，皆為字串變數。

10 DEFINT I-N, W-Z

表示所有以 I, J, K, L, M, N, W, X, Y, Z 字母開頭的變數，皆為整數變數。

## DEFUSR

語法：DEF USR [ <數字> ] = <整數表式>

目的：定出一組合語言副程式的起始位址。



## 40 APPLE 軟體卡

註釋：〈數字〉可為 0 至 9 任何一數，此數字與被設定位址之USR號碼相應，如果省略〈數字〉，電腦將自行採取DEF USR0。〈整數表式〉的值是USR routine 的起始位址。請看附錄 C，組合語言副程式。

DEF USR 陳述可在程式中出現任意次，用來定副程式的起始位址，所以可以使用很多你需要的副程式。

範例：

```
200 DEF USR0=24000
210 X=USR0(Y+2/2.89)
```

## DEL

語法：DEL[ 〈行號〉 ] [ -〈行號〉 ]

目的：消除程式中不要的行。

註釋：BASIC-80在執行DEL 以後會回至等待命令的狀態。如果不寫〈行號〉，將會出現“Illegal function call”的錯誤訊息。

使用DELETE 與 DEL 同義。

範例：DEL 40 消除行號為 40 的那一行之內容。  
DEL 40 - 100 消除行號 40 ~ 100 的內容。  
DEL - 40 消除自程式開頭到行號為 40 之間的内容。

## DIM

語法：DIM 〈足碼變數列〉

目的：設定陣列變數足碼最大值，並依此值配置儲存空間。

註釋：假如陣列變數不以 DIM 陳述說明其容量空間，則其最大的足碼將被設定為 10。如果出現足碼值超過了最大的儲存空間，會產生“Subscript out of range”錯誤訊息。除非使用 OPTION BASE 陳述特別指定外，足碼的最小值為 0。（參考 OPTION BASE）

DIM 陳述設定陣列元素的起始值為 0。

範例：    10 DIM A(20)                    ( 21 個元素 )  
           20 FOR I=0 TO 20  
           30 READ A(I)  
           40 NEXT I

## EDIT

語法：EDIT<行號>

目的：使被設定的那一行進入準備編輯的狀態。

註釋：若行中有錯，使用 EDIT 編修，只需編修錯誤之處，而不必重新打入整行。當電腦進入 EDIT 編輯狀態時，BASIC-80 會顯出我們要編修的那一行行號，然後跳過一個空格，等待編輯狀態的副命令（Subcommand）。

### (A) 編輯狀態的副命令

編輯狀態的副命令是用來移動游標或在某一行中進行插入、消除、替代、尋找等修改錯誤的工作。大多數的編輯狀態之副命令前面可冠以一整數字，如此可使命令執行許多次，假如不設定整數字，則視為 1。

編輯狀態的副命令被區分如下列幾項功能：

#### 1 移動游標光點（Moving the cursor）

2. 插入新的文字 ( Inserting text )
3. 消除不要的文字 ( Deleting text )
4. 找出需要的文字 ( Finding text )
5. 取代舊的文字 ( Replacing text )
6. 結束和重新進入編輯狀態 ( Ending and restarting Edit Mode ) 。

注意：在以下的描述中 <字元> 表示任一字元，<文字> 表示任意長度的字元字串，[ i ] 表示一可選用的整數，如省略此項則以 1 取代，\$ 則表示 Escape 鍵。

### 1. 移動游標光點

空白鍵 利用鍵盤上的空白鍵按桿可使游標光點向右移動。

← 利用此鍵可使游標光點向左移動，被游標光點經過的字元仍呈現在螢光幕上。

### 2. 插入新的文字

I I <文字> \$ 會在目前游標光點處插入 <文字>。被插入的字元會顯現在終端機上。鍵入 Escape 則終止此插入狀態，鍵入 Return 同樣地可以終止此插入狀態。Control - A 或者是左箭號 ( ← ) 可用來消除游標光點左方的字元，如果新插入的文字超過 255 個字元，終端機會發出警告音響，超過的字元將不會顯現出來。

X Xsubcommand 是用來延伸一行敘述。它將游標光點移至該行末端，以便插入新的文字。若要結束此狀態，鍵入 Escape 或 Return 即可。

### 3. 消除不要的文字

D [ i ] D, 消除游標光點右邊 i 個字元，被消去的字元

會被印於兩倒斜線( \ )之間，而游標光點會停留在最後一個被消去的字元右邊。假如游標光點右方的字元數目少於 i 的數目，則 iD 會消去此行游標光點右方的所有字元。

H H 會消去游標光點右方的所有字元，然後自動地進入插入模式。這在取代一行後端文字時很好用。

#### 4. 找出需要的文字

S [ i ] S < 字元 >，它會尋找出現第 i 個此字元，然後游標光點移至此字元的前面，假如此 < 字元 > 沒被找到，則游標光點會停在該行的最末端，所有被找過的字元都會出現在螢光幕上。

K [ i ] K < 字元 > 除了在找尋期間，會將所有的這個字元消除外，其餘作用與 [ i ] S < 字元 > 類似。被消除的字元由兩倒斜線前後包圍。

#### 取代舊的文字

C C < 字元 > 是以該字元替代下一個字元，如果要替代後 i 個字元，則鍵入 i C 後，再鍵入 i 個新字元。執行後顯示出新的 i 個字元，並返回編輯狀態。

#### 6. 結束且重新進入編輯狀態

< cr > 鍵下 Return 鍵會印出該行其餘的部份，且存下編修過的部份並跳出編輯狀態。

E E subcommand 除了不印出一行其餘的部份外，其他作用與 < cr > 相同。

Q Q subcommand 結束編輯狀態，返回 BASIC-80 等待命令的狀態，且不儲存編輯狀態中已更改的內容。

L L subcommand 印出該行其餘的部份且存下已編修

過的內容，並使游標光點回到該行之開頭。

- A A subcommand 可使你對任一行再編修，它可使始的內容重現，並使游標光點回到該行的開頭。

注意：在編輯狀態下，假如 BASIC-80 收到了不被認可的命令或不合法的字元，它會發出聲響，並且不接受此命令或字元。

(B) 語法錯誤 ( Syntax Errors )

當程式執行時，發現語法錯誤，BASIC-80 會自動在發生錯誤的那一行進入編輯狀態。例：

```
10 K = 2(4)
RUN
?Syntax error in 10.
10
```

在編修完成之後，鍵入 Carriage Return 或是 E 副命令，BASIC-80 會重新插入此行，如此會導致失去所有的變數值。爲了保留變數值，以 Q 副命令跳出編輯狀態，使 BASIC-80 返回待命狀態，如此所有的變數將被保留。

(C) Control-A

鍵入 Control-A 後，將使由目前打的這一行進入編輯狀態。BASIC-80 會回應以 Carriage Return、驚嘆號及一空格。游標光點出現在此行第一個字元的位置，之後再打入編輯狀態副命令。

注意：假如剛鍵入之行要進入編輯狀態，可使用 \*EDIT.\* 命令使得自此行進入編輯狀態。（“.”句點表示以此行作參考位置）

END

語法：END

目的：終止程式的執行，關閉所有的檔案返回等命狀態。

註釋：END 陳述可置於程式的任一適當位置以終止程式的執行，但 END 不會使 BREAK 訊息出現，此點不同於 STOP 陳述，END 陳述的位置大都在程式的最後一行。

範例：520 IF K>1000 THEN END ELSE GOTO 20

## ERASE

語法：ERASE <陣列變數 列>

目的：消除程式中的陣列。

註釋：陣列被 ERASE 消除後，可重新宣告 (redimensioned)，在記憶體中被消除的陣列其空間可用來作其他用途，若陣列重新宣告前無 ERASE 陳述將會產生“Redimensioned array”的錯誤訊息。

範例：

```
450 ERASE A,B
460 DIM B(99)
```

## ERR AND ERL VARIABLES

當進入錯誤處理副程式時，ERR 變數裏含有錯誤碼。

ERL 變數裏含有發生錯誤那一行的行號。ERR 和 ERL 常用在 IF…… THEN 的陳述來決定錯誤捕捉常式的執行流

程。

假如在直接模式狀態下的陳述發生錯誤，ERL 設為

65535 測知是否有直接模式陳述錯誤發生，可使用

|      |                               |    |
|------|-------------------------------|----|
|      | IF 65535 = ERL THEN ...       | 陳述 |
| 否則使用 | IF ERR = error code THEN ...  | 陳述 |
|      | IF ERL = line number THEN ... | 陳述 |

若行號不在相關運算子右邊，此行號不能由 RENUM 重訂。又因為 ERL 和 ERR 都是保留變數，不能出現在 LET 陳述的等號左邊。BASIC - 80 的錯誤號碼列於附錄 E。

## ERROR

語法：ERROR <整數表式>

- 目的：(1) 模擬 BASIC-80 錯誤的產生。  
(2) 讓使用者自行定義錯誤碼。

註釋：<整數表式> 的值，必須大於 0 而小於 255。如果 <整數表式> 的值等於 BASIC-80 裏正在使用的錯誤碼，ERROR 陳述會模擬該碼的錯誤而顯示出此錯誤的訊息。

利用大於任何一 BASIC-80 的錯誤碼之值，來定義你自己的錯誤碼。此自定的錯誤碼可以方便地在錯誤捕捉程式中處理 ( error trap routine )。

假如 ERROR 陳述中指定的錯誤碼予以一個錯誤沒有定義訊息，BASIC-80 將會有 "UNPRINTABLE ERROR" 的訊息出現。ERROR 陳述的執行，如果沒有相應的錯誤捕捉程式處理，就會使錯誤訊息印出並停止執行程式 ( halt )。

範例 1 : LIST  
 10 S = 10  
 20 T = 5  
 30 ERROR S + T  
 40 END  
 Ok  
 RUN  
 String too long in line 30

直接操作模式

Ok  
 ERROR 15  
 String too long  
 Ok

範例 2 :

```
110 ON ERROR GOTO 400
120 INPUT "WHAT IS YOUR BET";B
130 IF B > 5000 THEN ERROR 210

400 IF ERR = 210 THEN PRINT "HOUSE LIMIT IS $5000"
410 IF ERL = 130 THEN RESUME 120
```

**FIELD**

語法：FIELD [ # ] <檔案號碼> , <欄位寬度>AS <字串變數>.....

目的：分配隨機檔案緩衝區內的變數空間。

註釋：FIELD陳述執行過後，才能在 GET 之後從一隨機緩衝區內取出資料，或者在 PUT 之前輸入資料。<檔案號碼>是已被打開的檔案的對應號碼。<欄位寬度>是分配給<字串變數>用的字元數目。例：



```
FIELD 1, 20 AS N$, 10 AS ID$, 40 AS ADD$
```

在隨機檔案緩衝區中分配前 20 個拜給字串變數 N\$，其後 10 個拜給 ID\$，再其後 40 個拜給 ADD\$。FIELD 無法將任何資料放入隨機檔案緩衝區內。(請參閱 LSET/RSET 及 GET) FIELD 陳述中拜的總數，不能超過打開檔案設定的資料錄長度。否則將會出現“Field overflow”的錯誤訊息。對於同一個檔案，FIELD 陳述可被使用多次。

範例：請參閱附錄 B。

注意：在 INPUT 或 LET 的陳述內，不能使用 FIELD 的變數名稱。當一變數名稱被 FIELD 起來後，它就指向隨機檔案緩衝區的正確位置。如果以後遇到 INPUT 或 LET 陳述裏帶有此變數，當執行時，此變數指標被移至字串空間。

## FILES

語法：FILES [ <檔名> ]

目的：印出目前正在使用的磁碟內所儲存的檔案名稱。

註釋：假如省略 <檔名>，則會列出目前正在使用的磁碟機內的所有檔名。<檔名> 是一字串 (string formula)，它可包含 ① 問號 ( ? ) 可表檔名或延展名中的任意一字元及 ② 星號 ( \* ) 若當做檔名或延展字元則可代表任何檔名或任何延展名的組合。

```

範例： FILES
        FILES "*.BAS"
        FILES "B.*"
        FILES "TEST?.BAS"

```

## FOR.....NEXT

```

語法： FOR <變數> = X TO Y [STEP Z]
        :
        :
        NEXT [ <變數> ] [ , <變數> ..... ]

```

在此 X, Y 和 Z 是數值表式。

目的：允許某一系列的陳述，在一給定次數的迴圈內運算。

註釋：<變數>用作計數器，數值 X 為計數的起始值，數值 Y 為計數的終點值，如 FOR 陳述與 NEXT 陳述間的程式執行一次後再迴圈重新執行時，則增加一 STEP 值，直到等於 Y 值，停止迴圈內的執行。如果省略 [ STEP Z ]，則此增量會被電腦視為 1，終點值 Y 如果小於起始值 X，則 Z 值為負，增量為負意即其計數經每一迴圈後漸減其量，直到 X = Y。如果起始值 X 與終點值 Y 不能與 Z 值形成合理的增減量關係，將不執行 FOR ..... NEXT 之間的程式，而直接執行下一個陳述。

每一個 FOR 必須有一個 NEXT 與其對應。

### 巢狀多層迴圈 ( Nested Loops )

一個 FOR ..... NEXT 迴圈可置在另一個 FOR ..... NEXT 迴圈之內，如此巢狀多層迴圈，其每一個迴圈的計數變數名

稱必須不同。

迴圈內容許其它迴圈，先執行內圈再執行外圈，但迴圈間不可互相交叉。假如迴圈終止於同一個位置，則可共用同一 NEXT，但是每一個 FOR 的變數必須在 NEXT 陳述中設定。NEXT 後的變數可以省略，但它必須與最接近它的 FOR 陳述中的變數相配。

如果 NEXT 置於與其成對的 FOR 之前，則會有“NEXT without FOR”的錯誤訊息出現，並終止執行。

```
範例 1 : 10 K=10
          20 FOR I=1 TO K STEP 2
          30 PRINT I;
          40 K=K+10
          50 PRINT K
          60 NEXT
          RUN
          1 20
          3 30
          5 40
          7 50
          9 60
          Ok
```

```
範例 2 : 10 J=0
          20 FOR I=1 TO J
          30 PRINT I
          40 NEXT I
```

此例由於起始值大於終點值且省略 STEP，表示增量為 1，不符邏輯，所以不予執行。

```
範例 3 : 10 I=5
          20 FOR I=1 TO I+5
          30 PRINT I;
          40 NEXT
          RUN
          1 2 3 4 5 6 7 8 9 10
          Ok
```

```

範例 4 : 10 FOR I=1 TO 20
          20 IF I 10 GOTO 100
          30 NEXT
          40 GOTO 110
          100 NEXT
          110 END
    
```

此程式會導致“NEXT without FOR”的錯誤訊息，因為每一個NEXT必有一FOR 與其成對。

## GET

語法 1 : GET [ # ] <檔案號碼> [ , <資料錄號碼> ]

目的 1 : 將隨機磁碟檔中的資料錄讀入隨機緩衝區。

語法 2 : GET <鍵盤字元>

目的 2 : 從鍵盤讀進單一字元。

註釋：語法 1 中的<檔案號碼>是指被打開的檔案號碼，如果省略<資料錄號碼>，上次GET 的下一個資料錄會被讀入緩衝區。可容許最大的資料錄號碼為 32767。在GET 陳述之後，可使用 INPUT #和LINE INPUT #從隨機檔案緩衝區讀出字元。

語法 2 中的<鍵盤字元>不會在螢光幕上顯現。此語法不需要按RETURN 鍵。如果<鍵盤字元>是 Control - @，則會回應 NULL字元。GET “←”或Control - H也會回應 NULL 字元。

範例：語法 1 的例子請參閱附錄 B 。

語法 2 :

```
10 GET A$:PRINT A$;
20 GOTO 10
```

## GOSUB...RETURN

語法：GOSUB <行號>

```
  :
  :
RETURN
```

目的：至副程式中執行，執行完後返回主程式。

註釋：<行號>為副程式第一行的行號。

在程式執行時可以呼叫副程式許多次，副程式也可呼叫另一個副程式，它們使用的限制，由記憶體容量而定。

RETURN 陳述是使 BASIC-80 回到 GOSUB 陳述的下一個陳述繼續執行，副程式中可有數個 RETURN 陳述，它們可使流程在副程式中的不同位置合理地返回主程式中，副程式可以在程式中任何位置，但是最好能與主程式有明顯的區分。為了防止主程式非所願地進入副程式，通常可在副程式前加上 STOP, END 或 GOTO 陳述。

```
範例： 10 GOSUB 40
        20 PRINT "BACK FROM SUBROUTINE"
        30 END
        40 PRINT "SUBROUTINE";
        50 PRINT " IN";
        60 PRINT " PROGRESS"
        70 RETURN
        RUN
        SUBROUTINE IN PROGRESS
        BACK FROM SUBROUTINE
        Ok
```

**GOTO**

語法：GOTO<行號>

目的：無條件地跳出正常流程，進入某一特定行號執行。

註釋：假如程式<行號>為一可執行的陳述，則此陳述及其後之流程將被執行，如果<行號>為一不可執行的陳述，則尋找<行號>以下第一個可執行的陳述，繼續執行。

範例：

```
LIST
10 READ R
20 PRINT "R =";R,
30 A = 3.14*R^2
40 PRINT "AREA =";A
50 GOTO 10
60 DATA 5,7,12
Ok
RUN
R = 5      AREA = 78.5
R = 7      AREA = 153.86
R = 12     AREA = 452.16
?Out of data in 10
Ok
```

**GR**

語法：GR <螢光幕號碼>，<顏色號碼>

<螢光幕號碼>為 0 至 1 的整數，<顏色號碼>為 0 至 15 的整數。

目的：創立低解析度圖形的模式。

註釋：<螢光幕號碼>可設定如下的螢光幕模式：

|       |       |
|-------|-------|
| 螢光幕號碼 | 螢光幕模式 |
|-------|-------|

|   |                       |
|---|-----------------------|
| 0 | 40×40 的繪圖畫面，加上 4 行文字。 |
|---|-----------------------|

螢光幕號碼      螢光幕模式

1.                  40 × 48 的繪圖畫面，沒有文字畫面。

如果〈螢光幕號碼〉被省略，則此號碼會被電腦假設為 0。當創立了低解析度圖形的模式，GR 會清除畫面。〈顏色號碼〉可用來選擇畫面的顏色，如果〈顏色號碼〉被省略，則會被電腦設定為黑色，請參閱 COLOR 陳述中各種顏色的代號。

範例：GR    與 Applesoft GR 陳述相同。

GR 1,15 螢光幕背景為白色，螢光幕面積為 40 × 48 的模式。

注意：此陳述在 MBASIC 及 APPLESOFT 中的用法不盡相同。

## HLIN

語法：HLIN〈X1 座標〉，〈X2 座標〉AT 〈Y 座標〉，此處的 X1 及 X2 為 0 ~ 39 的整數，Y 為 0 ~ 47 的整數。

目的：在低解析度圖形中，畫一水平線從點 ( X1 , Y ) 至點 ( X2 , Y )。

註釋：〈X1 座標〉的值必須小於或等於〈X2 座標〉的值。

畫面的顏色由最近設的 COLOR 陳述決定。

設定 X1 , X2 , Y 的座標如果超過規定的範圍，會出現 ILLEGAL FUNCTION CALL 的錯誤訊息。HLIN 陳述在繪圖畫面中，兩點之連線為點所組成。但在 Y 為 40 ~ 47 的文字畫面或混合畫面中，兩點之連線則為字元所組成。

範例： 10 GR  
20 COLOR=3  
30 HLIN 14,20 AT 39

## HOME

語法：HOME

目的：清除螢光幕上所有內容，且使游標光點回至螢光幕左上角。

註釋：當使用外接終端機，HOME 陳述會傳送清除畫面的字元到終端機。

範例： 10 HOME  
20 VTAB 12  
30 PRINT "A CLEAN SCREEN"

## HTAB

語法：HTAB<螢光幕位置號碼>

目的：使游標光點移至<螢光幕位置號碼>的位置上。

註釋：行的第一個位置之位置號碼為 1，而最後一個位置之位置號碼為 40。

HTAB 後的位址是絕對位址而非相對位址，例如：假設游標光點現在的位置是 10，HTAB 13 執行後將使游標光點移至 13 的位置，而不是 23 的位置。

如果<螢光幕位置號碼>大於 40 而小於 255，電腦會根據此號碼除以 40，以所得餘數定位置。例如：HTAB 60，會將游標光點移至本行位置 20，假如號碼大於 255 則會出現“ ILLEGAL FUNCTION CALL ”。



**IF...THEN...ELSE及IF...GO TO**

語法：IF <表式> THEN <陳述> | <行號> [ ELSE <陳述> | <行號> ]

語法：IF <表式> GOTO <行號> [ ELSE <陳述> | <行號> ]

目的：根據表式的條件，決定程式的流程。

註釋：假如<表式>的結果不為零，則執行THEN或GOTO 後之陳述，THEN 之後可為一行號或一個以上之陳述。但是GOTO之後都為一行號，如果<表式>的結果為零，則不執行THEN或GOTO 的陳述僅執行ELSE的陳述，如果省略了ELSE則執行IF 陳述的下一行程式。(表示為真時其值為-1，表式為偽時其值為零)。

**巢狀多層IF陳述**

IF ... THEN ... ELSE，此陳述可以多層的使用，但不能超過一行字元的長度，即255個字元：例

```
IF X>Y THEN PRINT "GREATER" ELSE IF Y>X
  THEN PRINT "LESS THAN" ELSE PRINT "EQUAL"
```

以上為一合法的陳述。假如陳述中ELSE或THEN 個數不一樣，則ELSE 與最接近的THEN相配。例：

```
IF A=B THEN IF B=C THEN PRINT "A=C"
  ELSE PRINT "A<>C"
```

當A<>B時，並不會印出“A<>C”。

除非先進入間接執行模式中，否則IF...THEN 陳述若含有<行號>，在直接執行模式中，將會產生

“Undefined line ” 的錯誤訊息。

測試計算後的 A 值，是否為 1，可使用

```
IF ABS (A-1.0)<1.0E-6 THEN ...
```

來測知誤差是否小於  $1.0 E - 6$ 。

範例 1：200 IF I THEN GET #1,I

當 I 不為 0 時，以上陳述會 GET 檔案號碼 1 內號碼為 I 的資料錄進入隨機緩衝區。

範例 2：100 IF(I<20)\*(I>10) THEN DB=1979-1:GOTO 300  
110 PRINT "OUT OF RANGE"

此例中，假如  $I > 10$  且  $I < 20$ ，則計算 DB，然後跳至行號為 300 的陳述，否則，執行行號為 110 的陳述。

範例 3：210 IF IOFLAG THEN PRINT A\$ ELSE LPRINT A\$

此例中，假如 IOFLAG 為零，則跳至行列表機，否則，在終端機上顯示。

## INPUT

語法：INPUT[;][ < “催促字串” > ; ] < 變數列 >

INPUT[;][ < “催促字串” > , ] < 變數列 >

目的：為了能在程式執行中，從終端機輸入。

註釋：程式執行時，遇到 INPUT 陳述時，程式會暫停執行，等待資料由終端機輸入，如果 < 催促字串 > 不被省略，此字串將顯現在螢光幕上，然後從螢光幕輸入資料。

在<“催促字串”>後你可選擇分號或逗號，如同Apple-soft 使用分號可在<“催促字串”>後印出問號，使用逗號則不會出現問號。

如果INPUT後，緊跟著分號，則按RETURN 輸入資料後，不會有Carriage return/line feed回應至終端機。輸入的資料必須與<變數列>中的變數有相同的型態，資料項的個數必須與<變數列>中變數個數相同，資料項之間，須以逗點分開。

變數名稱可為數值變數或是字串變數，資料的輸入必須與變數名稱相符。

如果輸入的資料多於或少於變數個數，或輸入與變數名稱型態不符的資料，螢光幕會出現“? Redo from start”，直到符合要求時，才繼續執行。

注意此陳述在MBASIC內和APPLESOFT 用法不同。

```
範例 1 : 10 INPUT X
          20 PRINT X "SQUARED IS" X^2
          30 END
          RUN
          ? 5 (The 5 was typed in by the user
              in response to the question mark.)
          5 SQUARED IS 25
          Ok
```

```
範例 2 : LIST
          10 PI=3.14
          20 INPUT "WHAT IS THE RADIUS";R
          30 A=PI*R^2
          40 PRINT "THE AREA OF THE CIRCLE IS";A
          50 PRINT
          60 GOTO 20
          Ok
          RUN
          WHAT IS THE RADIUS? 7.4 (User types 7.4)
          THE AREA OF THE CIRCLE IS 171.946

          WHAT IS THE RADIUS?
          etc.
```

## INPUT #

語法：INPUT # <檔案號碼> ; <變數列>

目的：從一循序磁碟檔讀入資料項目並安排給程式的變數用之。

註釋：<檔案號碼>，是已打開，準備輸入的檔案的號碼。<變數列>裏的變數被指定與檔案裏的項目互相配合。INPUT #，不會有問號印出。檔案中資料群出現的方式，如同在INPUT 陳述中所規定的格式一樣。輸入的數值或字串前端不含（即使有也被忽略掉）空格，carriage return 或 line feed 字元。遇到第一個非此三者的字元則做為數值或字串的開頭，而以空格，carriage return line feed 或逗點作為數值的結束。如果輸入字串的第一個字元為引號，則前後兩引號間的所有（任意）字元皆被視為此字串項，但此字串項內不可再含引號。如果第一個字元不是引號，則此字串以逗點，carriage return，或 line feed 結束之。（如果讀入的字元超過 255 個，也會自動結束）。正在 INPUT 數值或字串時，若發生檔案結束（end-of-file），表示資料項目已用完。

範例：請參閱附錄 B。

## INVERSE

語法：INVERSE

目的：設定畫面輸出的型態，如此可使螢光幕成為白底黑字的畫面。

註釋：當使用外接的終端機，INVERSE 會將“Hi-lite”的字元序列傳送至有此改變畫面特性的終端機。

## 60 APPLE 軟體卡

INVERSE 並不會影響已經出現在螢光幕上的字元顏色。  
使用 NORMAJ 命令，可使畫面恢復至黑底白字。

範例： 10 PRINT "THESE ARE WHITE CHARACTERS"  
20 INVERSE  
30 PRINT "THESE ARE BLACK CHARACTERS"

## KILL

語法：KILL <檔名>

目的：從磁碟中消除某一檔案。

註釋：KILL 陳述不能用於已經打開的檔案，否則會有“File already open”的錯誤訊息。KILL可消除所有型式的磁碟檔案，如：程式檔、隨機資料檔、循序資料檔。

範例： 200 KILL "DATA1.TXT"  
( 參閱附錄 B )

## LET

語法：[ LET ] <變數> = <表式>

目的：賦予變數以表式的值。

註釋：LET 此字可要可不要，等號即可賦予變數以表式的值。

範例： 110 LET D=12  
120 LET E=12+2  
130 LET F=12+4  
140 LET SUM=D+E+F

或寫成：

```
110 D=12
120 E=12*2
130 F=12*4
140 SUM=D+E+F
```

## LINE INPUT

語法： LINE INPUT [ ; ] [ < “催促字串” > ; ] < 字串變數 >

目的：輸入整行（可最多至 254 字元）至一字串變數，而不需使用分界記號（ delimiters ）。

註釋：輸入變數被接受之前，催促字串已顯示在終端機的螢光幕上，除非問號在催促字串內，否則不會出現問號，催促字串出現以後輸入的資料均屬字串變數。

假如 LINE INPUT 緊跟著分號，鍵入 Carriage return 結束此行的輸入後，在終端機上沒有 Carriage return / line feed 的回應。

鍵入 Control - C 可暫停 LINE INPUT 使 BASIC-80 回到命令階層，如果再鍵入 CONT 則會恢復執行 LINE INPUT。

範例：請參閱 LINE INPUT # 的範例。

## LINE INPUT #

語法： LINE INPUT # < 檔案號碼 > , < 字串變數 >

目的：可從連續磁碟資料檔，不需分界記號下，將一整行的資料

(不得超過 254 個字元) 讀入字串變數中。

註釋：〈檔案號碼〉是已經打開的檔案的號碼，〈字串變數〉是被指定行的變數名稱。LINE INPUT # 可讀入循序檔內的所有字元直到遇到 Carriage return 為止。下一 次的 LINE

INPUT # 再讀入檔內的字元資料直到下一個 Carriage return 出現。

LINE INPUT # 對資料檔的每一行已被分成數欄，或一 BASIC-80 的程式以 ASCII 的模式保存而被其他程式當做資料讀過時，有其特殊的用處。

在 GET 後，INPUT # 及 LINE INPUT # 可從隨機檔案緩衝區讀出字元。

```

範例： 10 OPEN "0",1,"LIST"
        20 LINE INPUT "CUSTOMER INFORMATION? ";C$
        30 PRINT #1, C$
        40 CLOSE 1
        50 OPEN "1",1,"LIST"
        60 LINE INPUT #1, C$
        70 PRINT C$
        80 CLOSE 1
        RUN
CUSTOMER INFORMATION? LINDA JONES   234.4   MEMPHIS
LINDA JONES   234.4   MEMPHIS
Ok

```

## LIST

語法 1：LIST [〈行號〉]

語法 2：LIST [〈行號〉[ - [〈行號〉] ] ]

或是：LIST [〈行號〉[ , [〈行號〉] ] ]

目的：在終端機中，列出所有或部份目前在記憶體裏的程式。

註釋：在執行 LIST 陳述後 BASIC-80 即返回等待命令階層。

語法 1：如果省略〈行號〉，最低行開始列出所有的程式（程式列出時鍵入 Control - C 可終止 LIST）。如果含有〈行號〉，則僅列出此特定的行號陳述。

語法 2：此語法的結構有以下幾項選擇：

1. 如果只設定第一個行號，則會列出此行及所有高於此行號的程式。
2. 如果只設定第二個行號，則自程式開頭至此行的敘述均被列出。
3. 如果以上兩個行號均被設定，則列出此二行號之間所有的程式。

範例：語法 1：

LIST            列出記憶體中所有的程式。

LIST 500    列出行號 500 的敘述。

語法 2：

LIST 150 ←    列出行號 150 至程式終點所有陳述。

LIST - 1000    列出程式行號低於 1000 的所有陳述。

LIST 150 - 1000    列出行號 150 至行號 1000 的所有程式陳述。

## LLIST

語法：LLIST [〈行號〉 [ - [〈行號〉 ] ] ]

目的：將記憶體內的所有或部份程式列印於列印機。



## 64 APPLE 軟體卡

註釋：執行 LLIST 以後，BASIC - 80 返回命令階層。

LLIST 的功能如同 LIST 的語法 2。

LLIST 將列印機視為具 132 拜寬度。

注意：使用 LLIST 陳述，需在 Apple 的溝槽 1 ( Slot 1 ) 插上列印機之控制板 ( printer card )

範例：請參閱 LIST 語法 2 的範例。

## LOAD

語法：LOAD <檔名> [ , R ]

目的：從磁碟載入一檔案至記憶體中。

註釋：<檔名> 檔案存入 ( SAVE ) 磁碟時所使用的檔案名稱。在載入指定的程式之前，LOAD 將關閉所有已開的檔案，消去所有目前記憶體裏所存的程式及變數，如果 “ R ” 與 LOAD 一起使用，則在載入之後立即 RUN 此程式，所有打開的資料檔繼續保持打開的狀態。用 “ R ” 可以連結數個程式或同一程式的數個段落 ( segment )。程式之間可利用磁碟資料檔傳遞訊息。

範例：LOAD "STRTRK",R

## LPRINT and LPRINT USING

語法：LPRINT [ <表式列> ]

LPRINT USING <字串表式> ; <表式列>

目的：在列印機上印出資料。

註釋：除了在列印機印出資料外，其餘用法皆與 PRINT 和

PRINT USING 的用法相同。

LPRINT 用於有 132 字元寬的列印機。

注意：使用 LPRINT 陳述，需在 Apple 的溝槽 1 ( Slot 1 ) 插上列印機界面板 ( Printer card )。

### LSET and RSET

語法：LSET <字串變數> = <字串表式>

RSET <字串變數> = <字串表式>

目的：將記憶體內的資料移至隨機檔案緩衝區。(準備執行 PUT 陳述)。

註釋：假如字串表示所需要的 bytes 少於字串變數欄所擁有的拜數，則 LSET 使表示所需要的拜位置，置於字串變數空間的左側，RSET 則置於右側。如果字串過長，多餘的字元從最右邊開始拋棄。用 LSET 或 RSET 之前，數目值需轉成字串。見 MKI\$, MKS\$, MKD\$ 函數。

範例：150 LSET A\$=MK\$(AMT)  
160 LSET D\$=DESC(\$)

請參閱附錄 B。

注意：LSET 或 RSET 亦可用於未被分欄的字串變數中。

例：110 A\$=SPACE\$(20)  
120 RSET A\$=N\$

將 N\$ 在此 20 個字元之空白欄位內向右緊靠這種功能在設定輸出格式時頗方便。

**MERGE**

語法：MERGE <檔名>

目的：將一特定磁碟檔併入目前在記憶體中的程式。

註釋：<檔名>已被儲存的檔案名稱。此檔案必須以ASCII的組成格式儲存。(否則會出現“Bad file mode”的錯誤訊息)。

如果磁碟檔裏任一行的行號與記憶體裏的程式中的行號相同時，磁碟檔裏的行號陳述將取代記憶體內的行號陳述。

在結束MERGE命令以後BASIC-80將重回命令階層。

範例：MERGE 'NUMBR5'

用法：

```

11          Decision Making
12  IF A$="0" THEN END
13  IF (A$<>"F") AND (A$<>"M") THEN PRINT;A$;" : "
;"Error Input !";GOTO 10
120 PRINT "End of Calculation !"
150 END
SAVE B.BAS,A

```

使用NEW命令，再做程式A.BAS

```

5  ' ***** A.BAS *****
10 INPUT A$
20  WHILE A$="F"
30      SALARY#=1200*4.7/7.9
40      BONUS=150
50      GOTO B0
60  WEND
70      SALARY#=1300*4.9/7.7;BONUS=200
80  PRINT A$;" : "
90  PRINT TAB(5);"SALARY IS : ";SALARY#
100 PRINT TAB(5);"BONUS IS : ";BONUS!
110 STOP

```

使用MERGE命令將B.BAS併入記憶體中的程式中

## MERGE " B. BAS "

### OK

結果記憶體的內容為：

```

5  ***** A.BAS *****
10 INPUT A$
11      Decision Making
12  IF A$="0" THEN END
13  IF (A$<>"F") AND (A$<>"M") THEN PRINT A$;" ";
; "Error Input !":GOTO 10
20      WHILE A$="F"
30          SALARY#=1200*4.7/7.9
40          BONUS=150
50          GOTO 80
60      WEND
70      SALARY#=1300*4.9/7.7:BONUS=200
80  PRINT A$;" ";
90  PRINT TAB(5);"SALARY IS : ";SALARY#
100 PRINT TAB(5);"BONUS IS : ";BONUS!
110 STOP
120 PRINT "End of Calculation !"
150 END

```

## MID\$

語法：MID\$( <字串表式 1> , n [ , m ] ) = <字串表式 2>  
 , 此處 n , m 為整數。

目的：用某一字串取代另一字串的一部份。

註釋：<字串表式 1> 中位置為 n 的字元被 <字串表式 2> 的字元所取代。

可選擇的 m , 為 <字串表式 2> 中取代 <字串表式 1> 的字元數目。省略 m 則 <字串表式 2> 中的所有字元均用於取代原字串。替換的字元長度, 不可多於原字串表示式。

範例：
 

```

10 A$="KANSAS CITY, MO"
20 MID$(A$,14)="KS"
30 PRINT A$
RUN
KANSAS CITY, KS

```

參考第四章：MID\$ 也可以用作函數，回輸字串的一部份。

## NAME

語法：NAME <舊檔案名稱> AS <新檔案名稱>

目的：改變某一磁碟檔的名稱

註釋：使用 NAME 命令以後，磁碟中舊檔案的位置，內容皆不改變，只是舊檔名變為新檔名。

範例： Ok  
NAME 'ACCTS' AS 'LEDGER'  
Ok

## NEW

語法：NEW

目的：清除目前在記憶體內的程式及變數。

註釋：在輸入新程式以前，以 NEW 命令清除記憶體。執行結束後，BASIC-80 再重返命令階層。

## NORMAL

語法：NORMAL

目的：恢復螢光幕畫面至正常的黑底白字。

註釋：NORMAL 通常與 INVERSE 配合使用。NORMAL 不改變已經出現在螢光幕上 INVERSE 模式的字元。對外來終端機作 INVERSE 有“Hi-Lite”的功能，而 NORMAL 則傳送“Lo-lite”。“字元序列。參考”組立及操作手冊”。

範例： 10 INVERSE  
 20 PRINT "THIS IS INVERSE MODE"  
 30 NORMAL  
 40 PRINT "THIS IS NOT"

## ON ERROR GOTO

語法：ON ERROR GOTO <行號>

目的：啓動錯誤捕捉功能並設定處理錯誤副程式的起點。

註釋：錯誤捕捉功能動作後，所有的錯誤（包括直接操作模式產生的）被偵測出都會使程式跳越至處理錯誤的副程式。如果所定義的<行號>實際上不存在，會造成“Undefined line”錯誤訊息。若欲停止捕捉錯誤的功能，只需執行“ON ERROR GOTO 0”，這樣一來，再遇到錯誤時，只會顯示出錯誤訊息，並停止執行程式。其實“ON ERROR GOTO 0”本身就是一錯誤捕捉程式，但它會使BASIC-80停止並打出錯誤訊息。建議您應在所有不需要恢復動作的錯誤捕捉副程式中執行“ON ERROR GOTO 0”陳述。

注意：如果處理錯誤的副程式中，發生錯誤，則會出現錯誤的訊息，並停止執行 處理錯誤的副程式中不做錯誤捕捉。

( error trapping )

範例 . 10 ON ERROR GOTO 1000

注意，此陳述在MBASICB APPLESOFT中用法不同。

## ON...GOSUB及ON...GOTO

## 70 APPLE 軟體卡

ON <表式> GOTO <行號列>

目的：根據表式的結果，決定程式的流向。

註釋：<表式>的值，決定程式流向<行號列>所指的陳述。例如：

<表式>的值為 3，則程式流向<行號列>中排列第三的行號（若<表式>的值，不為整數，則四捨五入取其值）在 ON...GOSUB 的陳述<行號串>中的行號均為副程式起始的行號。

如果<表式>的值是零或大於<行號列>的項數，則 BASIC 繼續執行此陳述的下一行，如果<表式>的值為負或大於 255，則會出現“ Illegal function call ”的錯誤訊息。

範例： 100 ON L-1 GOTO 150,300,320,390

## OPEN

語法：OPEN <模式>， [ # ] <檔案號碼>， <檔名> [ ，  
<資料錄長度> ]

目的：允許磁碟檔的輸入和輸出

註釋：檔案在磁碟輸入及輸出操作運算以前，必須先打開磁碟檔。

OPEN 爲了輸入輸出檔案，提供一緩衝區，且決定該緩衝區的進出模式。

<模式>是一個字串表式，它的第一個字元，有下列幾種：

- O 設定循序輸出狀態
- I 設定循序輸入狀態
- R 設定隨機輸入輸出狀態

<檔案號碼>是整數表式，其值為 1 至 15。檔案打開以後檔案號碼就代表此檔案。

<檔名>為一字串表式其名稱是遵從磁碟檔作業系統規則。

<資料錄長度>是一整數表式，可用來設定隨機檔的資料錄長度，若不寫此項，則表示資料錄長度為 128 個拜。

注意：對於循序輸入，或是隨機存取檔案可以同時使用一個以上的號碼。對於輸出，只能一次打開一個檔案號碼。

範例：10 OPEN "1",2,"INVEN"

請參閱附錄 B。

## OPTION BASE

語法：OPTION BASE n

n 為 1 或 0

目的：宣告陣列足碼的最小值。

註釋：省略 n 則此值被視為 0，如果執行 OPTION BASE 1 陳述，則陣列足碼最小值為 1。

## PLOT

語法：PLOT <X座標>，<Y座標>

此處<X座標>為 0~39 的整數，<Y座標>為 0~47 的整數。

目的：在低解析圖形模式中，畫一定點。

註釋：螢光幕左上角的點為 ( 0, 0 )。

點的顏色由最近被執行的 COLOR 或 GR 的陳述決定。



如果 PLOT 用在文字幕 ( TEXT mode) 或用在混合圖形中 40 ~ 47 的文字窗內，則字元將取代點。

如果 < X 座標 > 或 < Y 座標 > 的範圍超過以上規定則會出現 " ILLEGAL FUNCTION CALL " 的錯誤訊息。

範例：  
GR  
COLOR=9  
PLOT 24,37

## POKE

語法：POKE I, J

此處的 I, J 為整數表式。

目的：將一拜寫進記憶體中的位置。

註釋：I 是記憶體中被 POKE 的位址。J 是被 POKE 的資料，J 的範圍 0 ~ 255，I 的範圍 0 ~ 65536。

與 POKE 相反的是 PEEK，PEEK 是從記憶體中某一位址，讀出一拜。

對於①資料儲存，②載入組合語言副程式，及③把引數及結果傳出或傳到組合語言副程式時，PEEK 及 POKE 陳述都很有用。

注意：在 Applesoft 中，除非先將 POKE 及 PEEK 轉換成 Z-80 的位址，否則無效。參考第 2 部份 Z-80 及 6502 位址轉換表。

範例：10 POKE &H5A00,&HFF

## POP

語法：POP

目的：在 GOSUB 之後，從副程式返回的流程，使其無法回至前一個 GOSUB 陳述的下一行陳述。

註釋：POP 可以用來替代 RETURN 結束某一個 GOSUB，但是它不回至 GOSUB 的下一個陳述，在 POP 之後的 RETURN 會叫程式回至第二個最接近執行的 GOSUB 陳述的下一個陳述。所以，POP 實際上拿掉 RETURN 位址堆疊最上面的一個位址。參看 GOSUB……RETURN。

```
範例： 5 PRINT "HERE WE GO"
        10 GOSUB 100
        20 PRINT "XYZ"
        30 END
        100 GOSUB 200
        110 PRINT "HELLO"
        120 RETURN
        200 POP
        210 RETURN

        RUN
        HERE WE GO
        XYZ
```

## PRINT

語法：PRINT [ <表式列> ]

目的：在終端機輸出資料。

註釋：如省略 <表示列>，將印出一行的空白。如寫出 <表示列>，將在終端機上印出其值。在表式列中的表式可為數值或字串表式。字串表式須用引號括出。

列印位置 ( Print Positions )

在表式列中用來分項的標點，可決定每一項被列印的位置。

BASIC-80 將每行以各 14 個空格來劃分為數個列印區 (Print zone)。逗號可使下一個值，列印至下一個列印區的開頭。分號則使下一個值，緊接著上一個值列印。表式間的數個空格造成的列印方式與分號相同。前後兩項並無空白存在。如果表式群以分號或逗號結束，則下一次 PRINT 時也據此緊跟著列印在此行之後。如果表式列不以分號或逗號結束，則下一次 PRINT 中的值，將在下一行列印。如果列印的資料，超過螢光幕的寬度，則 BASIC-80 會到下一行繼續列印。

列印時遇到數值，其後均自動接一空格。如數值為正，其前自動接一空格。如為負其前會有一負的符號。在單準制中，

10↑(-6) 的輸出型態為 .000001；10↑(-7) 的輸出型態為 1E-7。在倍準制中，會以 16 位數，或是比 16 位數少的型態輸出，如 1D-16 的輸出型態為

0000000000000001；1D-17 的輸出型態為 1D-17。

問號 (?) 可用來代替 PRINT。

```

範例 1 : 10 X=5
          20 PRINT X+5, X-5, X*(-5), X*5,
          30 END
          RUN
          10      0      -25      3125
          Ok
  
```

此例中 PRINT 中的逗號使每一個值均印在每一個列印區之開頭。

```

範例 2 : LIST
10 INPUT X
20 PRINT X "SQUARED IS" X^2 "AND"
30 PRINT X "CUBED IS" X^3
40 PRINT
50 GOTO 10
Ok
RUN
? 9
  9 SQUARED IS 81 AND 9 CUBED IS 729
? 21
  21 SQUARED IS 441 AND 21 CUBED IS 9261
?
    
```

此例中，行號 20 後跟著分號，使行號 20 及 30 所要印的內容，俱印在同一行，行號 40 的陳述表示印出一行空白。

```

範例 3  10 FOR X = 1 TO 5
        20 J=J+5
        30 K=K+10
        40 ?J;K;
        50 NEXT X
Ok
RUN
  5 10 10 20 15 30 20 40 25 50
Ok
    
```

此例中 PRINT 中使用分號，所以各值緊接在一起。行號 40 的問號代替了 PRINT。

## PRINT USING

語法：PRINT USING <字串表式> ; <表示列>

目的：用某一特定的格式列印出字串或數字。

註釋：<表式列>是由被分號分開的字串表式或數字表式所組成，

<字串表式>是由特殊格式字元所構成的字串文字。這些格式的字元，可以決定被印出的字串及數字之欄位及格式。

### 字串欄位 ( String Fields )

當 PRINT USING 被用來列印字串時，可使用三種設定格式的字元裏的一種。

"!" 只列印出給定字串的第一個字元。

"\ n 個空格\" 字串中  $2 + n$  個字元將被印出，若  $n = 0$  時（即兩斜線號中無空格存在），將印出字串中兩個字元。若字串太長，超過  $2 + n$  個空格，則多餘的字元將被忽略而不予印出。

如果欄位容量超過字串長度，列印時，字串會向左靠，剩餘欄位容量則填以空白。

```
例： 10 A$="LOOK":B$="OUT"
      30 PRINT USING "!":A$:B$
      40 PRINT USING "\ \":A$:B$
      50 PRINT USING "\ \":A$:B$;"!"
      RUN
      LO
      LOOKOUT
      LOOK OUT !!
```

"&" 指定一可變長度字串欄，當以 & 指定此欄時，字串就如輸入時一般地輸出：

```
例： 10 A$="LOOK":B$="OUT"
      20 PRINT USING "!":A$:
      30 PRINT USING "&":B$
      RUN
      LOUT
```

數值欄位 ( Numeric Fields )

當使用 PRINT USING 來列印數值，下列有幾個特殊的字元，可用來構成數值欄位。

- # 此符號用來代表每一個數字的位置，如果數字位數少於給定數字的位置，則列印出的數字向右移靠（數字前為空白）小數點可插入此欄位的任何位置，小數點前後的數字均被列印出，如果小數的整數部份為零，它會在小數點前列印出 0，如果欄位過小它會以四捨五入格式印出。

```
例： PRINT USING "###.##";.78
      0.78

      PRINT USING "###.##";987.654
      987.65

      PRINT USING "###.## ";10.2,5.3,66.789,.234
      10.20  5.30  66.79  0.23
```

在最後之例子裏，每個格式化字串的末端插入了 3 個空白。

- + 加號置於字串前面或後面，它會在數值前面或後面印出加號或負號。
- 在格式欄後面的減號，會使負數的負號在後印出。

```
例： PRINT USING "###.## ";-68.95,2.4,55.6,-.9
      -68.95  +2.40  +55.60  -0.90

      PRINT USING "###.##- ";-68.95,22.449,-7.01
      68.95-  22.45  7.01-
```

- \*\* 雙星號可以代表數字位置，當數字位數比給定位置少時，數字之前，會印出星號。

```
例： PRINT USING "***.# ";12.39,-0.9,765.1
      *12.4  *0.9  765.1
```

\$\$ 雙金錢符號能使一金錢符號印在數字前。\$\$ 不能用在指數格式的實數中。在此格式負數無法在前印出負號，除非負號出現在數字後。

```
PRINT USING "$$###.##";456.78
$456.78
```

\*\*\$ 使用此種符號時，\$ 是緊接在數字左邊。若數字位數少於給定位數，則\$左邊多出的空格以星號(\*)填滿。

例：

```
PRINT USING "***$.###";2.34
***$2.34
```

假如小數點左邊有逗點時，該數整數部份會由右向左，三個位數，就印出一個逗點。若逗點在格式字串末端，則該數的小數位數若少於給定位置，會於缺的位置以零補之，然後接著印出逗點；若小數位數多於給定位置，則四捨五入後，再印出逗點。

例：

```
PRINT USING "###.###";1234.5
1,234.50
PRINT USING "###.##";1234.5
1234.50,
```

↑↑↑ 四個箭號置在數字位置之後，表示以指數格式列印，

E + x x 將被印出。

```
例： PRINT USING "##.###↑↑↑";234.56
2.35E+02
PRINT USING "###.###↑↑↑";888888
.8889E+06
PRINT USING "+.###↑↑↑";123
+.12E+03
```

- 此符號用在字串格式會使其後的下一個字元也緊跟著輸出。

例：

```
PRINT USING "_!##.##_!";12.34
!12.34!
```

- % 假如數值位數大於給定的位置，則在數字之前會印出%。

例：

```
PRINT USING "###.##":111.22
%111.22

PRINT USING ".##".999
%1.00
```

設定的數值位數，不能超過 24 位

## PRINT # 與 PRINT # USING

語法：PRINT # <檔案號碼> , [ USING <字串表式> ; ] <表式列>

目的：將資料寫入循序磁碟檔。

註釋：<檔案號碼> 是指為輸出打開的檔案號碼。<字串表式> 格式字元的組成與 PRINT USING 相似。<表式列> 內的表式是指將被寫至檔案的數值或字串。

PRINT # 並不將資料壓縮存入磁碟，它只將原資料映像直接寫入磁碟。

在表式列的數值表式間須以分號分開。

例：PRINT # 1:A;B;C;X;Y;Z

( 如果使用逗號，多餘的空白也會被寫入磁碟 ) 其中的字串表示也須以額外的隔點及分號分開。



例： A\$="CAMERA" and B\$="93604-1"

PRINT #1,A\$,B\$

CAMERA 93604-1 會被寫入磁碟。因此，我們必須將字串分開，以免資料輸入磁碟，不易辨別，所以加入分段的符號於 PRINT #中，例：

PRINT #1,A\$;" ";B\$

寫入磁碟的格式為

CAMERA,93604-1

假如字串中包含有逗點、分號、空格等，則以引號 CHR\$(34) 的使用，將字串輸入磁碟，而不致混亂。例：

A\$="CAMERA, AUTOMATIC" B\$=" 93604-1"

PRINT #1,A\$,B\$

寫入磁碟的格式為

CAMERA, AUTOMATIC 93604-1

陳述 INPUT #1,A\$,B\$

會將 CAMERA 輸入 A\$，AUTOMATC 93604-1 輸入 B\$。為了能適當地分開這些字串，將雙引號代號 CHR\$(34) 寫入磁碟。例：

PRINT #1,CHR\$(34);A\$;CHR\$(34);CHR\$(34);B\$;CHR\$(34)

寫入磁碟的格式為

"CAMERA, AUTOMATIC" " 93604-1"

陳述 INPUT #1,A\$,B\$

將 "CAMERA , AUTOMATIC " 輸入 A\$，" 93604-1" 輸入 B\$。

PRINT #亦可與 USING 同時使用，來設定磁碟檔的格式

。如： PRINT #1,USING"\$\$###.##",J;K;L

PRINT #，PRINT # USING 和 WRITE # 可在 PUT 陳述之前，將字元放入隨機檔案緩衝區。

請參閱附錄 B，PRINT # 及 WRITE # 的範例。

## PUT

語法： PUT [ # ] <檔案號碼> [ , <資料錄號碼> ]

目的：將資料錄從隨機緩衝區寫入隨機磁碟檔。

註釋：<檔案號碼>是指已打開的檔案號碼，如果省略<資料錄號碼>則為最近一個 PUT 下一個資料錄號碼。

範例：請參閱附錄 B。

## RANDOMIZE

語法： RANDOMIZE [ <表式> ]

目的：產生隨機亂數。

註釋：如果省略<表式>，BASIC-80 會暫停執行，印出 Random Number Seed ( - 32768 到 32767 ) 並要求其  
中一值，以執行 RANDOMIZE。

假如不給予隨機變數產生器一新的種子數值，RND 函數  
會一直得到相同的結果。

範例：

```

10 RANDOMIZE
20 FOR I=1 TO 5
30 PRINT RND;
40 NEXT I
:RUN
Random Number Seed (-32768 + 32767)? 3 (user types 3)
.88598 .484668 .586328 .119426 .709225
Ok
RUN
Random Number Seed (-32768 + 32767)? 4 (user types 4 for
new sequence)
.803506 .162462 .929364 .292443 .322921
Ok
RUN
Random Number Seed (-32768 + 32767)? 3 (same sequence as
first RUN)
.88598 .484668 .586328 .119426 .709225
Ok

```

## READ

語法：READ <變數列>

目的：從DATA陳述中，讀入數值並設定給各對應。

註釋：READ 陳述經常與 DATA 陳述相配合使用。READ 中的變數與 DATA 中的資料型態須相符合。否則會產生 "Syntax error"。如果變數個數少於 DATA 內的個數，多餘的資料會被忽略，或由下一個 READ 繼續讀取之。可以使多個 READ，而只用一個 DATA；或是使用一個 READ，而用多個 DATA。若欲由開始端重讀 DATA 資料，可以使用 RESTORE 陳述。

範例 1 :

```
80 FOR I=1 TO 10
90 READ A(I)
100 NEXT I
110 DATA 3.08,5.19,3.12,3.98,4.24
120 DATA 5.08,5.55,4.00,3.16,3.37
```

執行後 A (1) = 3.08 .....

範例 2 :

```
LIST
10 PRINT "CITY", "STATE", ZIP
20 READ C$,S$,Z
30 DATA "DENVER,", COLORADO, 80211
40 PRINT C$,S$,Z
Ok
RUN
CITY          STATE      ZIP
DENVER,      COLORADO  80211
Ok
```

## REM

語法：REM <說明>

目的：允許解釋性的說明，插入程式中

註釋：REM 陳述不被執行，但是會被列印出來。

REM 可由 GOTO 或 GOSUB 陳述跳入，但由其下一行可執行的陳述繼續執行。

也可用單引號為開頭，附加在行的後面來增寫註解

範例：

```
120 REM CALCULATE AVERAGE VELOCITY
130 FOR I=1 TO 20
140 SUM=SUM + V(I)
```

or

```
120 FOR I=1 TO 20 'CALCULATE AVERAGE VELOCITY
130 SUM=SUM+V(I)
140 NEXT I
```

## RENUM

語法：RENUM [ [ <新號碼> ] [ , [ <舊號碼> ] [ , <增量> ] ] ]

目的：重新將程式中的各行編號。

註釋：<新號碼>是指重新編號的第一個行號，若不寫此項，則視<新號碼>為10。<增量>是指新排列行號的增量。<舊號碼>是指程式要重新編號部份的第一個行號，若省略此項，則<舊號碼>視為此程式開始的第一行行號。

RENUM 同時也會修正GOTO, GOSUB, THEN, ON...GOTO, ON...GOSUB 和 ERL等陳述中的參考行號，如果以上陳述所接的行號原並不存在此程式中，則會出現 "Undefined line xxxxx in yyyy" 的錯誤訊息，之後行號 xxxxx 不會更改，但行號 yyyy, 仍會被更改。

注意：RENUM不能更改程式中各陳述的順序，而且改後的數字不可超過65529。否則會出現 "Illegal function call" 的錯誤訊息。

範例：RENUM                    把程式的行號重新更改，從 10 開始增量為 10，直到程式結束。

RENUM 300,,50    更換所有程式的行號，由 300 為起始行號，50 為增量。

RENUM 1000,900,20    將程式中 900 以上的行號改成 1000 開始，以 20 為增量的行號

## RESET

語法：RESET

目的：在更換磁碟之後，重定 CP/M 的索引表配置資料。  
( directory allocation information )

用法：更換磁碟的步驟如下：首先關掉當時已打開的任何資料檔案，然後，移去舊磁碟及插入新磁碟，之後，鍵入 RESET。  
如果步驟有誤可能會失去資料，導致“Disk Read Only”錯誤。

## RESTORE

語法：RESTORE [ <行號> ]

目的：可允許重讀入某特定行之 DATA 陳述。

註釋：執行 RESTORE 陳述之後，如遇下個 READ 陳述，會再從第一個 DATA 陳述中第一項資料開始讀入，如果設定 <行號>，則將從那一行的 DATA 順序讀入。

範例： 10 READ A,B,C  
20 RESTORE  
30 READ D,E,F  
40 DATA 57, 68, 79

## RESUME

語法： RESUME  
RESUME 0  
RESUME NEXT  
RFSUME <line number>

目的：在錯誤修護程序執行後，繼續執行程式。

註釋：如何使用以上四種語法：

RESUME or RESUME 0

在發生錯誤的地方恢復執行。

RESUME NEXT

在發生錯誤的下一行恢復執行。

RESUME <line number>

在<行號>處恢復執行。

RESUME 陳述如不是用在錯誤處理程序中，則會印出  
" RESUME without error " 訊息。

範例： 10 ON ERROR GOTO 900

900 IF (ERR=230)AND(ERL=90) THEN PRINT "TRY  
AGAIN":RESUME 80

注意：此敘述在MBASIC和APPLESOFT 中用法不同。

## RUN

語法 1 : RUN [ <行號> ]

目的：執行記憶體中的程式。

註釋：如果<行號>設定，則由該行號開始執行。否則將從最低行號開始執行。

RUN 執行後 BASIC-80 將返回待命狀態。

範例：RUN

語法 2 : RUN <檔名> [ , R ]

目的：從磁碟中載入某一檔案至記憶體，並執行之。

註釋：<檔名>指已經存起來的檔案名稱。RUN 會先關閉所有打開的檔，而且消去目前在記憶體中的內容，再載入至記憶體。“R”則表示所有已打開的資料檔仍保持打開。

範例：RUN "NEWFIL",R

請參閱附錄 B。

## SAVE

語法：SAVE <檔名> [ , A | , P ]

目的：將程式檔案存入磁碟。

註釋：<檔名>是加引號的字串，假如該檔名已經存在，原檔內容會被新存入者蓋過。

• 選擇 A 表示檔案以 ASCII 格式存入，否則，以二進位碼存



入。雖然 ASCII 格式佔較多的空間，但是有些磁碟存取仍需檔案以 ASCII 格式存入。如 MERGE，LIST 等都是需用 ASCII 格式的檔案。

用 5.0 BASIC 寫的程式，若要轉到 Apple SoftCard 系統也必須以 ASCII 格式存入。

選擇 P 表示保護以二進位碼存入的檔案，已受保護之檔在 RUN 或 LOAD 後，無法被編輯或列印出來。

範例：  
`SAVE"COM2",A  
 SAVE"PROG",P`

請參閱附錄 B。

## STOP

語法： STOP

目的：終止程式執行，返回待命狀態。

註釋：STOP 陳述可用在程式中任何地方以終止執行。當執行程式遇到 STOP 則會出現以下訊息：

Break in line nnnnn

STOP 陳述並未關閉檔案，此與 END 不同。

STOP 陳述執行過後 BASIC-80 又返回待命狀態。

STOP 後可使用 CONT 繼續執行程式。

範例：  
`10 INPUT A,B,C  
 20 K=A*2*5.3:L=B*3/.26  
 30 STOP  
 40 M=C*K+100:PRINT M`

```

RUN
? 1,2,3
BREAK IN 30
Ok
PRINT L
  30.7692
Ok
CONT
  115.9
Ok
    
```

STOP 可以用來檢測程式執行狀況。

## SWAP

語法：SWAP <變數> , <變數>

目的：交換二變數值。

範例：任何型式的變數均可以 SWAP，但此二交換的變數必須具有相同資料型式。

```

範例： LIST
        10 A$=" ONE " : B$=" ALL " : C$="FOR"
        20 PRINT A$ C$ B$
        30 SWAP A$, B$
        40 PRINT A$ C$ B$

        RUN
        Ok
        ONE FOR ALL
        ALL FOR ONE
        Ok
    
```

## SYSTEM

語法：SYSTEM

目的：關閉所有的檔案及返回 CP/M。

註釋：Control - C，只能回至 BASIC，不能回至 CP/M。

## 90 APPLE 軟體卡

範例 :1 SYSTEM  
A>

2 10 INPUT A\$

⋮

80 IF A\$ = " END " THEN SYSTEM

OK

A >

### TEXT

語法：TEXT

目的：從低解析度圖形或高解析度圖形畫面重返Apple 文字畫面模式（24行×40字元）。

註釋：TEXT從低解析度圖形返回時，它會清除畫面；但在高解析度圖形中，返回時不清除畫面。

原來在Text 的模式中，所使用的TEXT與VTAB 24 有同樣的效果。

範例： 10 HGR  
20 COLOR=5  
30 VLIN 24,30 AT 35  
40 TEXT  
50 PRINT "THIS IS A VERTICAL LINE"

### TRACE/NOTRACE

語法： TRACE  
NOTRACE

目的：追蹤程式陳述的執行情形。

註釋：TRACE陳述可幫助除去錯誤（可用於直接操作模式或間接操作模式），它會印出程式中執行過的行號，這些號碼印在一中括號內。用NOTRACE 可消去TRACE的功能。

```

範例： TRACE
        Ok
        LIST
        10 K=10
        20 FOR J=1 TO 2
        30 L=K + 10
        40 PRINT J;K;L
        50 K=K+10
        60 NEXT
        70 END
        Ok
        RUN
        [10][20][30][40] 1 10 20
        [50][60][30][40] 2 20 30
        [50][60][70]
        Ok
        NOTRACE
        Ok
    
```

## VLIN

語法：VLIN <Y1 座標> , <Y2 座標> AT <X 座標> 此處  
Y1 與 Y2 為 0 至 47 的整數，X 為 0 至 39 的整數。

目的：在低解析度圖形模式，從點 ( X, Y1 ) 至點 ( X, Y2 ) 繪出一垂直線。

註釋：<Y1 座標> 必須小於或等於 <Y2 座標>。

如果任何座標超過範圍將會出現 " ILLEGAL FUNCTION CALL " 的錯誤訊息。

垂直線顏色由最近執行過的 COLOR 決定。

在圖形模式範圍內此線由點組成，如在文字模式範圍內則線由字元組成。

```

範例： 10 GR
        20 COLOR=3
        30 VLIN 20,45 AT 12
    
```

## VTAB

語法：VTAB <螢光幕行號>

目的：移動游標光點至<螢光幕行號>所設定的螢光幕位置上。

註釋：螢光幕最上一行為第1行，最底一行為24行。VTAB以絕對位置移動游標光點，例如游標光點現正位於第10行，若執行VTAB 13，則游標光點移至第13行，而不移至第23行。

假如<螢光幕行號>超過24，則將該數除以24後所得餘數，即為所指定的位置。若<螢光幕行號>大於255，則印出“ILLEGAL FUNCTION CALL”錯誤訊息。

範例：10 VTAB 12: PRINT "MIDDLE OF SCREEN"

## WAIT

語法：WAIT <位址> , I [ , J ]

此處I和J是整數表式

目的：檢視位址狀態時，將程式暫停執行。

註釋：WAIT陳述，將<位址>中的每一比次 ( bit ) 與J做XOR 運算，與I做AND 運算，假如運算結果為零，則BASIC-80在該位址由讀資料繼續檢視此資料：若運算結果不為零，則繼續執行下一陳述。不寫J，則認為J值為零。

注意：WAIT會無止境地反覆地檢視位址，而不繼續執行。如此就必須以人工重新啓動機器。( restare )

範例： 100 WAIT &HE000.128  
200 PRINT "KEYPRESS!":GOTO 100

## WHILE...WEND

語法： WHILE <表示式>

⋮

[ <迴圈陳述> ]

⋮

WEND

目的：當條件成立時，繼續執行迴圈內的陳述。

註釋：<表示>不為零時，執行<迴圈陳述>內的陳述群，直到遇到WEND陳述，BASIC再跳回WHILE陳述去檢查<表示式>是否仍成立，如果成立則繼續執行此迴圈，若不成立則執行WEND之後的陳述。

WHILE/WEND可以多層使用，每一WEND與對應的WHILE配合。

範例： 90 'BUBBLE SORT ARRAY A\$  
100 FLIPS=1 'FORCE ONE PASS THRU LOOP  
110 WHILE FLIPS  
115       FLIPS=0  
120       FOR I=1 TO J-1  
130             IF A\$(I)>A\$(I+1) THEN  
                    SWAP A\$(I),A\$(I+1):FLIPS=1  
140       NEXT I  
150 WEND

## WIDTH

語法 1 : WIDTH [ LPRINT ] <行寬>

目的 1 : 為終端機或列表機，設定列印行的寬度。

語法 2 : WIDTH [ <行寬> ] , [ <螢光幕高度> ]

目的 2 : 設定列印行的寬度以及螢光幕高度。

註釋 : <行寬> 為 15 至 255 之間的整數，<螢光幕高度> 為 1 至 24 之間的整數，使用 Apple video 時，省略 <行寬> 此量自然定為 40，省略 <螢光幕高度> 此量自然定為 24。當使用外來終端機，省略 <行寬> 時此量自然定為 80，省略 <螢光幕高度>，此量自然定為 24。

在語法 1 中若省略 LPRINT，則設定終端機的行寬，否則設定列表機行寬。

在語法 2 中，兩個參數，至少須設定一個參數。

若 <行寬> 定為 255，定義無限長的行，此時 POS 及 LPOS 函數回輪的游標或印字頭 ( printhead ) 位址數值在超過 255 時從零再算起。

## WRITE

語法 : WRITE [ <表式列> ]

目的 : 在終端機上輸出資料。

註釋 : 如果省略 <表式列>，將輸出空白的行，否則 <表式列> 中的值會顯現在終端機上，此值可為數值或字串表式，各項間以逗點分開。

每輸出一個項目，會在該項目之後，自動印出逗點，字串的印出在頭尾帶有引號。

輸出數值時，WRITE 所用格式與 PRINT 相同。

```
範例： 10 A=80:B=90:C$=THAT'S ALL
        20 WRITE A,B,C$
        RUN
        80, 90, 'THAT'S ALL'
        Ok
```

## WRITE #

語法：WRITE # <檔案號碼> , <表式列>

目的：將資料寫入循序檔 ( Sequential file )

註釋：<檔案號碼> 為在“0”輸出狀態下被打開的檔案的號碼，表式列中的表式可為字串或數值表式，表式間以逗點分開。WRITE # 與 PRINT # 不同之處為，WRITE # 在將資料寫入磁碟檔時，會以逗點分開各項資料，如為字串則在頭尾加上引號。

使用 PUT 陳述之前，可以用 WRITE #、PRINT # 及 PRINT # USING 將字元放入隨機檔案緩衝區中。

```
範例： Let A$="CAMERA" and B$="93604-1".
        WRITE #1,A$,B$
```

下列形式將被寫入磁碟：

```
'CAMERA','93604-1'
```

使用 INPUT # 陳述從磁碟回輸時：

```
INPUT #1,A$,B$
```

輸入 ' CAMERA ' 至 A\$ ，輸入 ' 93604-1 ' 至 B\$。





## 第 四 章

### BASIC-80 函數

本章介紹 BASIC-80 提供的內部函數。這些函數不需要額外的定義就可由其他程式呼叫利用之。各函數呼叫格式的括號中是函數引數，函數引數的種類以下列簡寫表示：

X 和 Y      代表任一數值表式

I 和 J      代表任一整數表式

X 和 Y      在圖形函數上代表座標位置

X\$ 和 Y\$    代表字串表式

如果在需用整數值時輸入浮點值，BASIC-80 會將分數部分（小數部分）四捨五入並利用結果的整數。

註：用 BASIC-80 翻譯執行程式時（interpreter），函數只會做出整數和單準（single precision）數值。倍準數值（double precision）的函數只有在使用 BASIC 編譯程式（compiler）時才提供。

#### ABS

語法：ABS(X)

## 98 APPLE 軟體卡

動作：回輸X的絕對值。

例子：  
PRINT ABS(7\*(-5))  
35  
Ok

## ASC

語法：ASC(X\$)

動作：回輸字串X\$第一個字元的ASCII碼（參考附錄G）。如果X\$是空字元（null），就回輸“Illegal function call”（不合法的函數呼叫）錯誤訊息。

例子：  
10 X\$ = 'TEST'  
20 PRINT ASC(X\$)  
RUN  
84  
Ok

參考CHR\$ 函數，由ASCII到字元的轉換。

## ATN

語法：ATN(X)

動作：回輸X徑度量（radians）的反正切函數值（arctangent）。結果值的範圍在 $-\pi/2$ 到 $+\pi/2$ 之間。X可以是任一種數值型式，但ATN都以單準數值計算之。

例子：

```
10 INPUT X
20 PRINT ATN(X)
RUN
? 3
1.24905
Ok
```

**BUTTON**

語法： `BUTTON(I)`

動作：回輸遊樂器控制第 I 按鈕的值。

附記：I 的範圍是 0 ~ 3。

按鈕未按下此值為 0，當按鈕按下時此值為 1。

例子： `10 IF BUTTON (0) THEN PRINT "BOOM"`

**CDBL**

語法： `CDBL(X)`

動作：把 X 轉成倍準數值。

例子： `10 A = 454.67`  
`20 PRINT A;CDBL(A)`  
`RUN`  
`454.67 454.6700134277344`  
`Ok`

**CHR \$**

語法： `CHR$(I)`

動作：回輸 ASCII 碼為 I 的字元。(參考附錄 G，ASCII 碼表)。

`CHR$` 常用於把特殊字元送到終端機。例如 `BEL` 字元可由 `CHR$(7)` 送出作為錯誤訊息的前言，或饋表 (form feed) 可由 `CHR$(12)` 送出以清除 CRT 螢幕並使游標返回起始位置。

例子： `PRINT CHR$(66)`  
`B`  
`Ok`

## 100 APPLE 軟體卡

參考ASC 函數，字元至ASCII碼的轉換。

### CINT

語法： CINT(X)

動作：以四捨五入法把X轉換成整數值。如果X範圍不在-32768至32767間，就會發生溢位錯誤(overflow)。

例子： PRINT CINT(45.67)  
          46  
          Ok

參考CDBL及CSNG 函數，把資料型式轉變成倍準數值或轉變成單準數值。

### COS

語法： COS(X)

動作：回輸X徑度量的餘弦函數值(cosine)，以單準數值計算之。

例子： 10 X = 2 \* COS(.4)  
          20 PRINT X  
          RUN  
          1.84212  
          Ok

### CSNG

語法： CSNG(X)

動作：把X轉變成單準數值。

例子：  
 10 A# = 975.3421#  
 20 PRINT A#; CSNG(A#)  
 RUN  
 975.3421 975.342  
 Ok

參考 CINT 及 CDBL 函數，把數值轉變成整數或倍準數值的資料型式。

### CVI, CVS, CVD

語法：CVI (< 2 拜字串 > )  
 CVS (< 4 拜字串 > )  
 CVD (< 8 拜字串 > )

動作：把字串值轉變成數值。從隨機磁碟檔讀入的原數值資料必須由字串轉回數目值。CVI 轉變 2 拜的字串成為整數。CVS 轉變 4 拜的字串成為單準數值。CVD 轉變 8 拜字串成為倍準數值。

例子：

```

    .
    .
    70 FIELD #1,4 AS N$, 12 AS B$, ...
    80 GET #1
    90 Y=CVS(N$)
    .
    .
    
```

參考本章的 MKI\$, MKS\$, MKD\$, 及附錄 B。

### EOF

語法：EOF (< 檔名 > )

## 102 APPLE 軟體卡

動作：如果循序檔 ( sequential file ) 到達結束點，回輸 “ - 1 ” ( 真 )。在輸入 ( INPUT ) 資料時用 EOF 檢查是否已到檔案結束點，以避免輸入超過了結束點的錯誤。 EOF 也可以用在隨機檔 ( random file )。在 GET 超過檔案結束點時，EOF 回輸 - 1。這可配合二進式尋找 ( binary search ) 或其他方法來找出檔案大小。

例子：  
10 OPEN "1",1,"DATA"  
20 C=0  
30 IF EOF(1) THEN 100  
40 INPUT #1,M(C)  
50 C=C+1:GOTO 30  
.  
.  
.

## EXP

語法： EXP(X)

動作：回輸  $e$  的  $X$  次方值 (  $e^x$  )。  $X$  必須小於或等於 87.3365。如果 EXP 值溢位 ( overflow )， “ Overflow ” 錯誤訊息會顯示出，計算結果為帶正確正負號的無窮大數。

例子：  
10 X = 5  
20 PRINT EXP (X-1)  
RUN  
54.5982  
Ok

## FIX

語法： FIX(X)

動作：回輸  $X$  的整數部分 ( 小數部分一律刪除 )。  $FIX(X)$  等效

於  $SGN(X) * INT(ABS(X))$ 。FIX 與 INT 不同的地方是 FIX 不回輸負值 X 的次低整數值 (next lower number)。(如  $FIX(-1.1) = -1$ )

例子：  
 PRINT FIX(58.75)  
 58  
 Ok  
 PRINT FIX(-58.75)  
 -58  
 Ok

## FRE

語法： FRE(0)  
 FRE(X\$)

動作：FRE 中的引數是亞引數 (dummy)。FRE 回輸記憶體中未被 BASIC-80 使用到的拜數量。

FRE ( " " ) 會在回輸自由 ( free ) 拜數量前先做“垃圾搜集” ( garbage collection ) 的工作 ( 清除沒有用的資料 ) 垃圾搜集工作需要用 1 到 1 分半鐘時間。BASIC 本身在自由記憶體全部用完之前不會做垃圾搜集的工作。週期地使用 FRE ( " " ) 可以縮短每次自動做“垃圾搜集”的間隔時間。

例子：  
 PRINT FRE(0)  
 14542  
 Ok

## HEX \$

語法： HEX\$(X)



## 104 APPLE 軟體卡

動作：回輸一字串，此字串代表著 10 進位變數 X 的 16 進位值。  
在 HEX\$(X) 計算前，先把 X 四捨五入成爲整數。

例子：  
10 INPUT X  
20 A\$ = HEX\$(X)  
30 PRINT X "DECIMAL IS " A\$ " HEXADECIMAL"  
RUN  
? 32  
32 DECIMAL IS 20 HEXADECIMAL  
Ok

參考 OCT\$ 函數，做 8 進位數的轉換。

## INKEY \$

語法： INKEY\$

動作：回輸由終端機讀入的字元，如果沒有字元進入就回輸一個空字元 ( NULL )。這個功能不把讀入字元再送回終端機打出 ( no echoed )，除了 Ctrl-C 以外所有輸入 ASCII 字元都可傳送入程式，Ctrl-C 字元會終止程式的執行。

例子：  
1000 'Timed Input Subroutine  
1010 RESPONSE\$=""  
1020 FOR I%=1 TO TIMELIMIT%  
1030 A\$=INKEY\$: IF LEN(A\$)=0 THEN 1060  
1040 IF ASC(A\$)=13 THEN TIMEOUT%=0: RETURN  
1050 RESPONSE\$=RESPONSE\$+A\$  
1060 NEXT I%  
1070 TIMEOUT%=1: RETURN

## INPUT \$

語法： INPUT\$(X[,#]Y)

動作：由終端機或第 Y 檔讀入 X 個字元的字串。由終端機的輸入並不對終端機作回印 ( 回輸顯示於終端機上 )，除了 Ctrl-C

之外所有的控制字元都可以輸入。Ctrl-C 用來中斷 INPUT\$ 函數的執行。

例子 1 :

```

5 LIST THE CONTENTS OF A SEQUENTIAL FILE IN HEXADECIMAL
10 OPEN "I", "DATA"
20 IF EOF(1) THEN 50
30 PRINT HEX$(ASC(INPUT$(1, # 1)));
40 GOTO 20
50 PRINT
60 END

```

例子 2 :

```

100 PRINT "TYPE P TO PROCEED OR S TO STOP"
110 X$=INPUT$(1)
120 IF X$="P" THEN 500
130 IF X$="S" THEN 700 ELSE 100

```

## INSTR

語法：INSTR(I, X\$, Y\$)

動作：在字串 X\$ 中找出第一次出現的 Y\$ 字串，並回輸所處的位置。可以任選起始尋找點的偏移量 I。I 的範圍必須在 0 到 255 間。如果 I > LEN(X\$)，或 X\$ 為空字串（長度零），或找不到 Y\$，則 INSTR 回輸 0。如果 Y\$ 是空字串，INSTR 返回 I 或 1。X\$ 及 Y\$ 可為字串變數（string variable），字串表式（string expressions），或文義字串（string literals）。

```
例子： 10 X$ = "ABCDEB"  
        20 Y$ = "B"  
        30 PRINT INSTR(X$,Y$);INSTR(4,X$,Y$)  
        RUN  
        2 6  
        Ok
```

## INT

語法： INT(X)

動作：回輸小於或等於 X 的最大整數。

```
例子： PRINT INT(99.89)  
        99  
        Ok  
        PRINT INT(-12.11)  
        -13  
        Ok
```

參考FIX及CINT 函數。

## LEFT \$

語法： LEFT\$(X\$,I)

動作：回輸字串 X\$ 的左邊 I 個字元。I 的範圍在 0 到 255 間。

如果 I 大於 LEN(X)，回輸整串 X\$。如果 I = 0，回輸一個空字串（長度零）。

```
例子： 10 A$ = "BASIC-80"  
        20 B$ = LEFT$(A$,5)  
        30 PRINT B$  
        BASIC  
        Ok
```

參考MID\$及RIGHT\$函數。

**LEN**

語法： LEN(X\$)

動作：回輸字串 X\$ 的字元個數。不可印出的字元及“空格”字元 ( blank ) 也計算在內。

例子： 10 X\$ = 'PORTLAND, OREGON'  
 20 PRINT LEN(X\$)  
 16  
 Ok

**LOC**

語法： LOC ( <檔案號碼> )

動作：用在隨機磁碟檔執行 GET 或 PUT 時，LOC 回輸下一個使用的資料錄號碼。用在循序檔時，LOC 回輸此檔自 OPEN 後，已被讀出或寫入磁碟的段落數目 ( 每段 128 拜 )。

例子： 200 IF LOC(1)>50 THEN STOP

**LOF**

語法： LOF ( <檔案號碼> )

動作：回輸已讀入或寫出 ( 現今這個延伸 ( extent ) 中 ) 的資料錄數目。如果檔案不超過一個延伸 ( 128 個資料錄 )，LOF 回輸檔案實際長度 ( 資料錄數目 )。

例子： 110 IF NUM%>LOF(1) THEN PRINT "INVALID ENTRY"

## LOG

語法： LOG(X)

動作：回輸 X 的自然底 ( e ) 對數值。X 必須大於零。

例子： PRINT LOG(45/7)  
1.86075  
Ok

## LPOS

語法： LPOS(X)

動作：回輸在列印機 ( line printer ) 緩衝區中的列印頭現今位置 ( 卻不一定是列印頭的現今實際位置)。X 是啞引數 ( 無作用 )。

例子： 100 IF LPOS(X)>60 THEN LPRINT CHR\$(13)

## MID \$

語法： MID\$(X\$,I[,J])

動作：自第 I 字元開始，回輸 X 字串的 J 個字元。I 和 J 的範圍需在 0 到 255 間。如果 J 省略掉，或 I 字元之後不足 J 個字元，則回輸第 I 字元及其後的所有字元。如果 I > LEN(X\$)，MID\$ 回輸空字串。

例子： LIST  
10 A\$="GOOD "  
20 B\$="MORNING EVENING AFTERNOON"  
30 PRINT A\$;MID\$(B\$,9,7)  
Ok  
RUN  
GOOD EVENING  
Ok

參考LEFT\$及RIGHT\$函數。

### MKI\$ MKS\$ MKD\$

語法：MKI\$ (<整數表式>)  
 MKS\$ (<單準數值表式>)  
 MKD\$ (<雙準數值表式>)

動作：把數值轉變成字串。任何欲以LSET,RSET陳述把數目值放入隨機檔緩衝區的動作前，需先把數目值轉換成字串表式。MKI\$ 把整數值轉換成一個2拜字串，MKS\$ 把單準數值轉換成一個4拜字串。MKD\$ 把雙準數值轉換成8拜字串。

例子：  
 90 AMT=(K+T)  
 100 FIELD #1, 8 AS D\$, 20 AS N\$  
 110 LSET D\$ = MKS\$(AMT)  
 120 LSET N\$ = AS\$  
 130 PUT #1

### OCT\$

語法：OCT\$(X)

動作：回輸一個代表此10進位引數的8進位數值。在計算OCT\$(X)前，X先被四捨五入成整數。

例子：  
 PRINT OCT\$(24)  
 30  
 OK

參考HEX\$ 函數。

## PDL

語法： PDL(I)

動作：回輸遊樂器控制棒“ I ”的現在位置值，範圍為 0 到 255。

記要： I 是 0 到 3 的整數。

兩個遊樂器控制棒不可以用連續的指令讀入，因為讀第一個會被第二個影響。為了能正確地讀入兩者，10 FOR X=1 TO 10 NEXT X 延遲陳敘應插入其間分開它們。

例子：

```
10 PRINT PDL(0): GOTO 10
RUN
0
23
79
100
190
255
C
BREAK IN
OK
```

## PEEK

語法： PEEK(I)

動作：回輸記憶體位置 I 的拜值（拜值的範圍 0 到 255 的 10 進位整數）。I 的範圍在 0 到 65536 間。PEEK 和 POKE 對應，參考 POKE 函數。

註：用在 Applesoft 和 Integer BASIC 的 PEEK 和 POKE 不能用在 BASIC-80，除非先將它們轉成 Z-80 位址。參考本書“軟體及硬體詳述”部分的 6502 對 Z-80 的位址對照表。

例子： A=PEEK(&h5A00)

## POS

語法： POS(X)

動作：回輸現今游標位置。最左位置以 1 表示。X 是亞引數。

例子： IF POS(X)>60 THEN PRINT CHR\$(13)

參考 LPOS 函數。

## RIGHT \$

語法： RIGHT\$(X\$,I)

動作：回輸字串 X\$ 的右邊 I 個字元。如果 I = LEN(X\$)，回輸整串 X\$，如果 I = 0，回輸空字串（長度零）。

例子： 10 A\$="DISK BASIC-80"  
20 PRINT RIGHT\$(A\$,8)  
RUN  
BASIC-80  
OK

參考 MID\$ 及 LEFT\$ 函數。

## RND

語法： RND(X)

動作：回輸一個隨機數目（亂數）（0 到 1 之間的值）。除非重訂亂數產生器（random number generator），每次 RUN 同一 RND 時都會得到相同的亂數序列（參考 RANDOMIZE）。



## 112 APPLE 軟體卡

但如果隨意設一小於零的 X 值，RND 會固定地造出相同的亂數序列。而  $X > 0$  或不用 X，則產出本序列中的下一個數目。X = 0 則重複出現上一個出現的數目。

例子：  
10 FOR I=1 TO 5  
20 PRINT INT(RND\*100);  
30 NEXT  
RUN  
24 30 31 51 5  
Ok

## SCRN

語法：SCRN(X,Y)

X 為 0 到 39 的整數，Y 為 0 到 47 的整數。

動作：回輸 ( X , Y ) 點的顏色號碼。

例子：  
10 GR  
20 COLOR=13  
30 PLOT 10,15  
PRINT SCRN(10,15)  
RUN  
13

## SGN

語法：SGN(X)

動作：如果  $X > 0$ ，回輸 1。如果  $X = 0$ ，回輸 0。如果  $X < 0$ ，回輸 - 1。

例子：ON SGN(X)+2 GOTO 100,200,300  
X < 0 時跳到 100，X = 0 時跳到 200，X = 1 時跳到 300。

**SIN**

語法： SIN(X)

動作：回輸徑度量X的正弦函數值。SIN(X)以單準數值計算。

例子： PRINT SIN(1.5)  
997495  
Ok.

**SPACE \$**

語法： SPACE\$(X)

動作：回輸X個空格。X被四捨五入成爲整數，其範圍需在0到255間。

例子： 10 FOR I = 1 TO 5  
20 X\$ = SPACE\$(I)  
30 PRINT X\$;I  
40 NEXT I  
RUN  
1  
2  
3  
4  
5  
Ok

參考SPC 函數。

**SPC**

語法： SPC(I)

動作：在終端機印出I個空格。SPC 可以用在PRINT和

LPRINT 陳敘上。I 的範圍在 0 到 255 間。

例子：  
PRINT "OVER" SPC(15) "THERE"  
OVER THERE  
Ok

參考 SPACE\$ 函數。

## SQR

語法： SQR(X)

動作：回輸 X 的開平方根值。X 必須大於或等於 0。

例子：  
10 FOR X = 0 TO 25 STEP 5  
20 PRINT X, SQR(X)  
30 NEXT  
RUN  
10 3.16228  
15 3.87298  
20 4.47214  
25 5  
Ok

## STR\$

語法： STR\$(X)

動作：回輸 X 值的字串表示法。

例子：  
5 REM ARITHMETIC FOR KIDS  
10 INPUT "TYPE A NUMBER":N  
20 ON LEN(STR\$(N)) GOSUB 30,100,200,300,400,500

參考 VAL 函數。

**STRING \$**

語法： STRING\$(I,J)  
 STRING\$(I,X\$)

動作：回輸 I 個 ASCII 碼 " J " 的字元或 I 個 X\$ 字串的第一字元。

例子： 10 X\$ = STRING\$(10,45)  
 20 PRINT X\$ "MONTHLY REPORT" X\$  
 RUN  
 \_\_\_\_\_MONTHLY REPORT\_\_\_\_\_

Ok

**TAB**

語法： TAB(I)

動作：在終端機上打出“空白”直到位置 I。如果現今位址已超過 I 位置，TAB 就指到下一行的 I 位置。最左位置定義為“1”，最右位置為螢幕寬度減 1。I 的值必須在 1 到 255 間。TAB 函數可用在 PRINT 及 LPRINT 陳述。

例子： 10 PRINT "NAME" TAB(25) "AMOUNT" : PRINT  
 20 READ A\$,B\$  
 30 PRINT A\$ TAB(25) B\$  
 40 DATA "G. T. JONES", "\$25.00"  
 RUN  
 NAME                    AMOUNT  
 G. T. JONES            \$25.00  
 Ok

**TAN**

語法： TAN(X)

動作：回輸徑度量 X 的正切函數。TAN(X) 以單準數值計算之。

如果TAN 溢位 ( overflow ) , “ Overflow ” 錯誤訊息會顯示出 , 然後以帶適當正負號的無限大值做為結果 , 並繼續執行下一動作。

例子 :  $10 Y = Q * \text{TAN}(X) / 2$

## USR

語法 :  $\text{USR} [ \langle \text{數字} \rangle ] ( X )$

動作 : 帶著引數 X , 呼叫使用者的組合語言副程式。 < 數字 > ( 範圍 0 到 9 ) 對應於 DEF USR 陳述的常式定義數。如果 < 數字 > 省略掉 , 就自動設為 USR 0。參考附錄 C。

例子 :  $40 B = T * \text{SIN}(Y)$   
 $50 C = \text{USR}(B/2)$   
 $60 D = \text{USR}(B/3)$

## VAL

語法 :  $\text{VAL}(X\$)$

動作 : 回輸字串 X\$ 的數值。如果 X\$ 的第一個字元不是 + , - , & , 或一數碼 ( digit ) , 則  $\text{VAL}(X\$) = 0$ 。VAL(X\$) 會先去除引數中的空白、tabs 及饋行 ( linefeed ) 字元 , 再執行。

```

例子： 10 READ NAME$,CITY$,STATE$,ZIP$
        20 IF VAL(ZIP$)<90000 OR VAL(ZIP$)>96699 THEN
        PRINT NAME$ TAB(25) "OUT OF STATE"
        30 IF VAL(ZIP$)>=90801 AND VAL(ZIP$)<=90815 THEN
        PRINT NAME$ TAB(25) "LONG BEACH"

```

參考 STR\$ 函數。

## VARPTR

語法 1：VARPTR (<變數名> )

語法 2：VARPTR ( #<檔案號碼> )

動作：語法 1：回輸“變數名”對應資料的第一拜的位址。在執行 VARPTR 之前必須先設定“變數名”本身的值，否則會出現“Illegal function call”錯誤訊息。“變數名”可以是任何型式（數值，字串，陣列），而回輸的位址值則是範圍 32767 到 - 32768 間的整數。回輸的負位址值加上 65536 才得到實際位址。VARPTR 常用來取得變數或陣列的位址，依此把變數或陣列傳給組合語言副程式。在傳送陣列（array）時，常用 VARPTR(A(0)) 回輸陣列的最低位址。

註：在為陣列呼叫 VARPTR 之前，所有的單變數都應事先被安排設定好，因為任一新設單變數都會改變陣列位址。

語法 2：回輸“<檔案號碼>”檔案的磁碟 I/O 緩衝區的起始位址。

用於隨機檔時，VARPTR ( #<檔案號碼> ) 回輸欄緩衝區 (FIELD buffer) 的位址。

例子：100 X=USR(VARPTR(C))

## VPOS

語法：VPOS(I)

動作：回輸螢幕游標的縱軸位置。最高位置為1。I是啞引數。

例子：  
10 PRINT "NOW YOU SEE IT."  
20 FOR T=0 TO 1000: NEXT T  
30 VTAB VPOS(0)-1  
40 PRINT "NOW YOU DON'T"

**BASIC-80 之命令，陳述及函數的應用範例**

應用例 1 求  $1 + 2 + 3 + \dots + 100$  的結果以 BASIC-80 程式解之。

```

10 S=0; I=1
20 S=S+I
30 IF I=100 THEN 60
40 I=I+1
50 GOTO 20
60 PRINT " 1+2+3+.....+100 = "; S
70 END
RUN
  1+2+3+.....+100 = 5050
OK

```

說明：此程式以連加法解。也可以採用等差級數之和 " $n(1+n)/2$ ， $n = 100$ " 解之。

應用例 2 以 BASIC-80 程式求出  $n = 1$  至 18 的  $N!$  值。

```

10 ANSWER#=1!
20 FOR I=1 TO 18
30 ANSWER#=ANSWER##I
40 PRINT USING "##"; I;
50 PRINT "!=" "; ANSWER#
60 NEXT I
70 END
OK
RUN
  1!= 1
  2!= 2
  3!= 6
  4!= 24
  5!= 120
  6!= 720
  7!= 5040
  8!= 40320
  9!= 362880
 10!= 3628800
 11!= 39916800
 12!= 479001600
 13!= 6227020800
 14!= 87178291200
 15!= 1307674368000
 16!= 20922789888000
 17!= 355687428096000
 18!= 6402373705728000

```



## 120 APPLE 軟體卡

說明：其中利用了倍準變數“ANSWER#”。

應用例 3 將 23, 12, 2, 58, 234, 987, 65, 12, 100, 225 十個數依順序排列之。

```

20 FOR I=1 TO 10:READ A(I):NEXT I
30 DATA 45,32,125,99,54,5,79,98,65,100
50 FOR I=1 TO 9
55 IF A(J)>A(J+1) THEN SWAP A(J),A(J+1)
70 NEXT J
90 NEXT I
100 FOR J=1 TO 10:PRINT A(J);:NEXT J
110 END
OK

```

RUN

5 32 45 54 65 79 98 99 100 125

說明：利用 BASIC-80 的 SWAP 陳述可以很容易地處理排列問題。

應用例 4 以 MBASIC-80 程式列出每個 ASCII 碼對應的 ASCII 字元。

```

10 PRINT TAB(10);"***** Display the ASCII_codes *****"
20 PRINT
30 PRINT TAB(10);"1. ASCII codes from 0 to 31 are control codes"
40 PRINT TAB(10);"2. ASCII codes from 32 to 126 are characters."
50 PRINT
60 FOR I=32 TO 126
70 PRINT TAB(5+B*K);I;CHR*(I);:K=K+1
80 IF K=9 THEN K=0
90 NEXT I
99 END
OK
RUN

```

\*\*\*\*\* Display the ASCII\_codes \*\*\*\*\*

1. ASCII codes from 0 to 31 are control codes
2. ASCII codes from 32 to 126 are characters.

|       |       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 32    | 33 !  | 34 "  | 35 #  | 36 \$ | 37 %  | 38 &  | 39 '  | 40 (  |
| 41 )  | 42 *  | 43 +  | 44 ,  | 45 -  | 46 .  | 47 /  | 48 0  | 49 1  |
| 50 2  | 51 3  | 52 4  | 53 5  | 54 6  | 55 7  | 56 8  | 57 9  | 58 :  |
| 59 ;  | 60 <  | 61 =  | 62 >  | 63 ?  | 64 @  | 65 A  | 66 B  | 67 C  |
| 68 D  | 69 E  | 70 F  | 71 G  | 72 H  | 73 I  | 74 J  | 75 K  | 76 L  |
| 77 M  | 78 N  | 79 O  | 80 P  | 81 Q  | 82 R  | 83 S  | 84 T  | 85 U  |
| 86 V  | 87 W  | 88 X  | 89 Y  | 90 Z  | 91 [  | 92 \  | 93 ]  | 94 ^  |
| 95 _  | 96 `  | 97 a  | 98 b  | 99 c  | 100 d | 101 e | 102 f | 103 g |
| 104 h | 105 i | 106 j | 107 k | 108 l | 109 m | 110 n | 111 o | 112 p |
| 113 q | 114 r | 115 s | 116 t | 117 u | 118 v | 119 w | 120 x | 121 y |
| 122 z | 123 { | 124   | 125 } | 126 ~ |       |       |       |       |

應用例 5 比較-45.67, 45.67, 0, 12.49, -12.49 各數經由 ABS, CINT, FIX, INT, SGN 函數之後所得的結果。

```

10 PRINT " X ABS(X) CINT(X) FIX(X) INT(X) SGN(X) "
20 FOR I=1 TO 5
30 READ X
40 PRINT X;TAB(8);ABS(X);TAB(18);CINT(X);TAB(27);FIX(X);TAB(35);
50 NEXT I INT(X);TAB(43);SGN(X)
60 DATA -45.67,45.67,0,12.49,-12.49
70 END
OK
RUN

```

| X      | ABS(X) | CINT(X) | FIX(X) | INT(X) | SGN(X) |
|--------|--------|---------|--------|--------|--------|
| -45.67 | 45.67  | -46     | -45    | -46    | -1     |
| 45.67  | 45.67  | 46      | 45     | 45     | 1      |
| 0      | 0      | 0       | 0      | 0      | 0      |
| 12.49  | 12.49  | 12      | 12     | 12     | 1      |
| -12.49 | 12.49  | -12     | -12    | -13    | -1     |

應用例 6 以 BASIC-80 程式示意出 BASIC-80 的各字串函數用法。

122 APPLE 軟體卡

```

10 REM *** ILLUSTRATE ALL THE USE OF EACH CHARACTER FUNCTION "
20 X$="ABC-24/26 IS THE MOST POWERFUL MICROCOMPUTER"
30 Y$="C":A$=LEFT$(X$,6):B$=MID$(X$,10,3):C$=RIGHT$(X$,14)
40 D$=A$+B$+C$:E=INSTR(3,X$,Y$):F$=STRING$(10,X$)
50 G$=STRING$(10,65)
60 H=VAL(MID$(X$,4,3))
70 PRINT "X$=ABC-24/26 IS THE MOST POWERFUL MICROCOMPUTER"
80 PRINT:PRINT"CHAR.FUNCTION      MEANING      LENGTH"
85 PRINT STRING$(50,45)
90 PRINT"LEFT$(X$,6)";TAB(20);A$;TAB(45);LEN(A$)
100 PRINT"MID$(X$,10,3)";TAB(20);B$;TAB(45);LEN(B$)
110 PRINT "RIGHT$(X$,14)";TAB(20);C$;TAB(45);LEN(C$)
120 PRINT"A$+B$+C$";TAB(20);D$;TAB(45);LEN(D$)
130 PRINT"INSTR(3,X$,Y$)";TAB(20);E;TAB(45);LEN(STR$(E))
140 PRINT"STRING$(10,X$)";TAB(20);F$;TAB(45);LEN(F$)
150 PRINT"STRING$(10,65)";TAB(20);G$;TAB(45);LEN(G$)
160 PRINT"VAL(MID$(X$,4,3))";TAB(20);H;TAB(45);LEN(STR$(H))
170 END

```

OK

RUN

X\$=ABC-24/26 IS THE MOST POWERFUL MICROCOMPUTER

| CHAR.FUNCTION       | MEANING                 | LENGTH |
|---------------------|-------------------------|--------|
| LEFT\$(X\$,6)       | ABC-24                  | 6      |
| MID\$(X\$,10,3)     | IS                      | 3      |
| RIGHT\$(X\$,14)     | MICROCOMPUTER           | 14     |
| A\$+B\$+C\$         | ABC-24 IS MICROCOMPUTER | 23     |
| INSTR(3,X\$,Y\$)    | 3                       | 2      |
| STRING\$(10,X\$)    | AAAAAAAAAA              | 10     |
| STRING\$(10,65)     | AAAAAAAAAA              | 10     |
| VAL(MID\$(X\$,4,3)) | -24                     | 3      |

應用例 7. 有 10 個學生做一份含 10 道題的選擇題，每題 10 分。用

BASIC-80 程式閱卷計算各生成績。

```

20 DIM A(10),B(10),C(10,10),S(20):S(I)=0
30 FOR I=1 TO 10:READ A(I):NEXT I
40 FOR J=1 TO 10
50 READ B(J)
60 FOR I=1 TO 10
70 READ C(J,I)
80 NEXT I
90 NEXT J
100 FOR J=1 TO 10
110 FOR I=1 TO 10
120 IF A(I)=C(J,I) THEN S(J)=S(J)+1
130 NEXT I
140 NEXT J
150 PRINT "THE STANDARD ANSWER IS"
160 FOR I=1 TO 10:PRINT A(I):NEXT I:PRINT
170 PRINT:PRINT"THE STUDENT'S ANSWER LIST":PRINT
180 PRINT" NO           ANSWER           SCORE"
190 FOR J=1 TO 10
200 PRINT B(J);
210 FOR I=1 TO 10
220 PRINT USING "##";C(J,I);
230 NEXT I
240 PRINT TAB(31);S(J)*10
250 NEXT J
260 DATA 3,4,1,5,2,2,3,4,4,1
270 DATA 101,3,4,1,2,5,3,4,4,1,1
280 DATA 102,3,4,1,5,2,3,4,4,2,1
290 DATA 103,2,4,1,2,5,2,3,4,4,1
300 DATA 104,3,4,1,5,5,2,3,4,4,1
310 DATA 105,3,4,1,5,2,2,3,4,4,1
320 DATA 106,3,4,1,5,2,2,3,4,4,1
330 DATA 107,2,4,3,3,2,2,3,4,4,1
340 DATA 108,3,4,2,5,2,2,3,4,4,1
350 DATA 109,3,4,1,5,2,2,2,3,3,1
360 DATA 110,3,4,1,5,2,2,3,4,4,2
370 END
OK
RUN

```

```

THE STANDARD ANSWER IS
3 4 1 5 2 2 3 4 4 1

```

THE STUDENT'S ANSWER LIST

| NO  | ANSWER              | SCORE |
|-----|---------------------|-------|
| 101 | 3 4 1 2 5 3 4 4 1 1 | 50    |
| 102 | 3 4 1 5 2 3 4 4 2 1 | 70    |
| 103 | 2 4 1 2 5 2 3 4 4 1 | 70    |
| 104 | 3 4 1 5 5 2 3 4 4 1 | 90    |
| 105 | 3 4 1 5 2 2 3 4 4 1 | 100   |
| 106 | 3 4 1 5 2 2 3 4 4 1 | 100   |
| 107 | 2 4 3 3 2 2 3 4 4 1 | 70    |
| 108 | 3 4 2 5 2 2 3 4 4 1 | 90    |
| 109 | 3 4 1 5 2 2 2 3 3 1 | 70    |
| 110 | 3 4 1 5 2 2 3 4 4 2 | 90    |

## 124 APPLE 軟體卡

說明：第 30—90 行：讀入正確答案，各生學號及作答。

第 100—140：計算各生答對題數。

第 150—250 行：印出標準答案及各生答案，分數。

第 260—360 行：輸入的標準答案及各生答案。

應用例 8 寫一 BASIC-80 程式，印出所指定年（公元）的月曆—  
萬年曆程式。

```

10 DIM MON%(12),A$(12);P$="SUN MON TUE WED THU FRI SAT"
20 DATA 31,28,31,30,31,30,31,31,30,31,30,31
30 DATA "JANUARY","FEBRUARY","MARCH","APRIL","MAY","JUNE","JULY","AUGUST"
40 DATA "SEPTEMBER","OCTOBER","NOVEMBER","DECEMBER"
50 FOR I%=1 TO 12
60     READ MON%(I%)
70 NEXT I%
80 FOR I%=1 TO 12
90     READ A$(I%)
100 NEXT I%
110 INPUT "YEAR ( a$ 1911 ) : ",Y%
120 IF Y% MOD 4 =0 THEN MON%(2)=29
130 A%=FIX((Y%-1)*1.25) MOD 7
140 PRINT CHR$(14);TAB(18);Y%;CHR$(15);PRINT
160 PRINT TAB(3);STRING$(70,"-")
170 FOR I%=1 TO 12 STEP 2
180     SW1=0;SW2=0
190     B%=(A%+MON%(I%)) MOD 7
200     C%=A%;D%=B%
210     A%=(B%+MON%(I%+1)) MOD 7
220     A1%=18-FIX(LEN(A$(I%))/2);B1%=58-FIX(LEN(A$(I%+1))/2)
230     PRINT ;PRINT TAB(A1%);A$(I%);TAB(B1%);A$(I%+1)
240     PRINT;PRINT TAB(4);P%;TAB(44);P%
250     M%=1;N%=1
260     WHILE C%<7
270         IF M%>MON%(I%) THEN SW1=1;GOTO 330
280         EX=C%*4+4
290         PRINT TAB(EX);
300         PRINT USING "###";M%;
310         M%=M%+1;C%=C%+1
320     WEND
330     C%=0
340     WHILE D%<7
350         IF N%>MON%(I%+1) THEN SW2=1;GOTO 410
360         EX=D%*4+4
370         PRINT TAB(EX);
380         PRINT USING "###";N%;
390         N%=N%+1;D%=D%+1
400     WEND
410     D%=0
420     IF SW1<>1 OR SW2<>1 THEN 260
430     PRINT;PRINT TAB(3);STRING$(70,"-")
440 NEXT I%
450 INPUT " C(ONTINUE)/E(ND) ? ";E$
460 IF E$="C" THEN 120
470 END
OK
RUN

```

1982

JANUARY

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     | 1   | 2   |
| 3   | 4   | 5   | 6   | 7   | 8   | 9   |
| 10  | 11  | 12  | 13  | 14  | 15  | 16  |
| 17  | 18  | 19  | 20  | 21  | 22  | 23  |
| 24  | 25  | 26  | 27  | 28  | 29  | 30  |
| 31  |     |     |     |     |     |     |

FEBRUARY

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
|     | 1   | 2   | 3   | 4   | 5   | 6   |
| 7   | 8   | 9   | 10  | 11  | 12  | 13  |
| 14  | 15  | 16  | 17  | 18  | 19  | 20  |
| 21  | 22  | 23  | 24  | 25  | 26  | 27  |
| 28  |     |     |     |     |     |     |

MARCH

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
|     | 1   | 2   | 3   | 4   | 5   | 6   |
| 7   | 8   | 9   | 10  | 11  | 12  | 13  |
| 14  | 15  | 16  | 17  | 18  | 19  | 20  |
| 21  | 22  | 23  | 24  | 25  | 26  | 27  |
| 28  | 29  | 30  | 31  |     |     |     |

APRIL

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     | 1   | 2   | 3   |
| 4   | 5   | 6   | 7   | 8   | 9   | 10  |
| 11  | 12  | 13  | 14  | 15  | 16  | 17  |
| 18  | 19  | 20  | 21  | 22  | 23  | 24  |
| 25  | 26  | 27  | 28  | 29  | 30  |     |

MAY

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     | 1   |
| 2   | 3   | 4   | 5   | 6   | 7   | 8   |
| 9   | 10  | 11  | 12  | 13  | 14  | 15  |
| 16  | 17  | 18  | 19  | 20  | 21  | 22  |
| 23  | 24  | 25  | 26  | 27  | 28  | 29  |
| 30  | 31  |     |     |     |     |     |

JUNE

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
|     |     | 1   | 2   | 3   | 4   | 5   |
| 6   | 7   | 8   | 9   | 10  | 11  | 12  |
| 13  | 14  | 15  | 16  | 17  | 18  | 19  |
| 20  | 21  | 22  | 23  | 24  | 25  | 26  |
| 27  | 28  | 29  | 30  |     |     |     |

JULY

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     | 1   | 2   | 3   |
| 4   | 5   | 6   | 7   | 8   | 9   | 10  |
| 11  | 12  | 13  | 14  | 15  | 16  | 17  |
| 18  | 19  | 20  | 21  | 22  | 23  | 24  |
| 25  | 26  | 27  | 28  | 29  | 30  | 31  |

AUGUST

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
| 1   | 2   | 3   | 4   | 5   | 6   | 7   |
| 8   | 9   | 10  | 11  | 12  | 13  | 14  |
| 15  | 16  | 17  | 18  | 19  | 20  | 21  |
| 22  | 23  | 24  | 25  | 26  | 27  | 28  |
| 29  | 30  | 31  |     |     |     |     |

SEPTEMBER

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
|     |     |     | 1   | 2   | 3   | 4   |
| 5   | 6   | 7   | 8   | 9   | 10  | 11  |
| 12  | 13  | 14  | 15  | 16  | 17  | 18  |
| 19  | 20  | 21  | 22  | 23  | 24  | 25  |
| 26  | 27  | 28  | 29  | 30  |     |     |

OCTOBER

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     | 1   | 2   |
| 3   | 4   | 5   | 6   | 7   | 8   | 9   |
| 10  | 11  | 12  | 13  | 14  | 15  | 16  |
| 17  | 18  | 19  | 20  | 21  | 22  | 23  |
| 24  | 25  | 26  | 27  | 28  | 29  | 30  |
| 31  |     |     |     |     |     |     |

NOVEMBER

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
|     | 1   | 2   | 3   | 4   | 5   | 6   |
| 7   | 8   | 9   | 10  | 11  | 12  | 13  |
| 14  | 15  | 16  | 17  | 18  | 19  | 20  |
| 21  | 22  | 23  | 24  | 25  | 26  | 27  |
| 28  | 29  | 30  |     |     |     |     |

DECEMBER

| SUN | MON | TUE | WED | THU | FRI | SAT |
|-----|-----|-----|-----|-----|-----|-----|
|     |     |     | 1   | 2   | 3   | 4   |
| 5   | 6   | 7   | 8   | 9   | 10  | 11  |
| 12  | 13  | 14  | 15  | 16  | 17  | 18  |
| 19  | 20  | 21  | 22  | 23  | 24  | 25  |
| 26  | 27  | 28  | 29  | 30  | 31  |     |

說明：第10—40行：設定變數空間及字串變數PS 給定月的日數及名稱。

第50—70行：將各月的日數設入陣列MON%(1%)中。

第80—100行：將各月名稱設入陣列AS(1%)中。

第110—120行：輸入年份，若是閏年二月為29天。

第130行：求出指定年份的特有偏移量。

第150—160行：印出年份及雙線。

第170,440行：做六次迴圈。

第180行：定下420行之IF 陳述初值。

第190—220行：算出兩個月名稱印出位移量。

第230—240行：印出兩個月並排月名稱，及一星期的每天名稱。

第270—320行：印出左邊月份的一行日期。

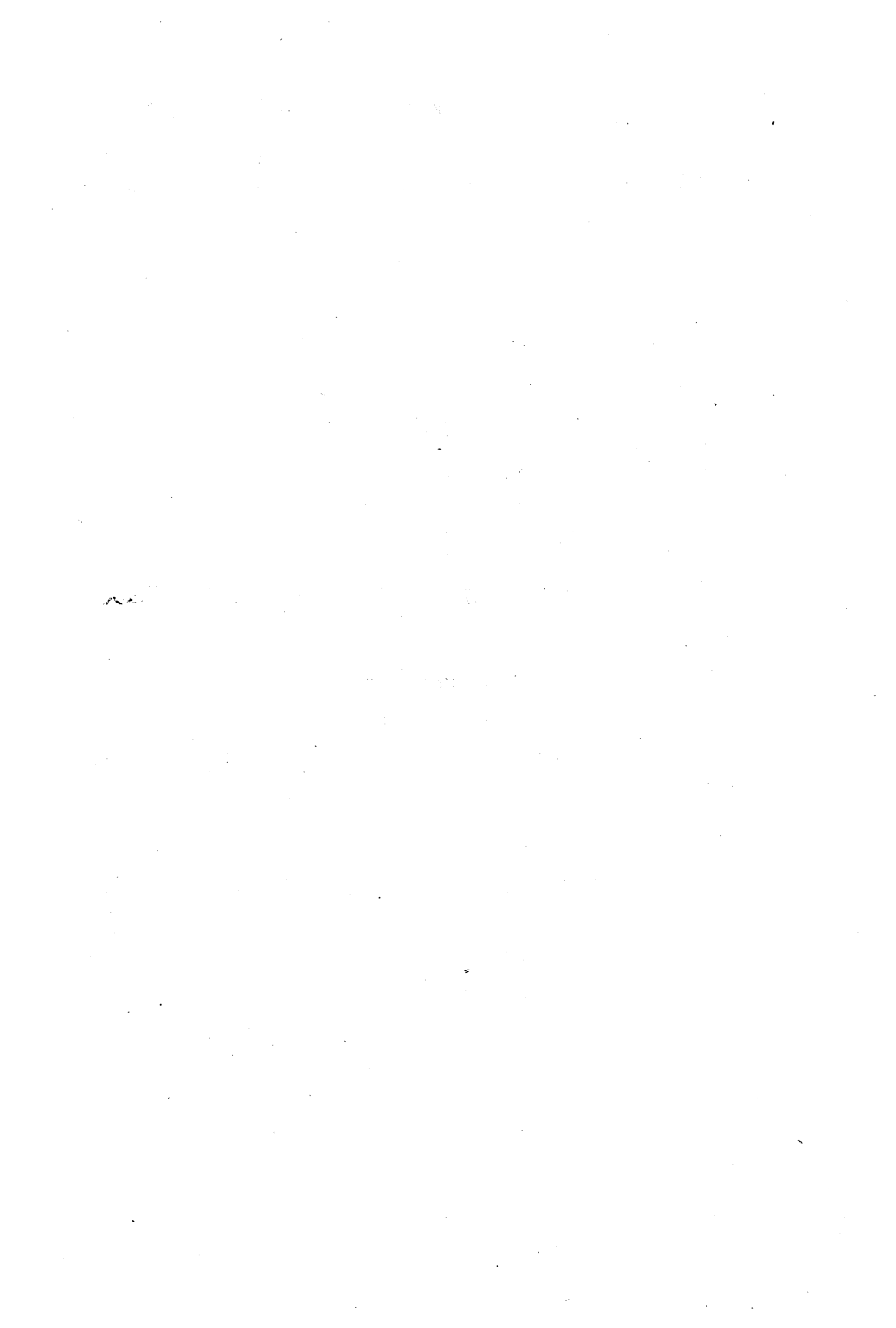
第340—400行：印出右邊月份的一行日期。

第420行：重覆270—400行的動作，直到月份中的日期印完為止。

第430行：印出一線。

第450—470行：指定再印一份或停止執行。





## 第五章

# 高解析度圖形——GBASIC

### 起始設定

GBASIC 是 Microsoft BASIC 的一種 CP/M 版本，除了含所有原 MBASIC 的功能項目外，它增加了高解析度圖形能力。

GBASIC 只在 16 段磁碟中，檔名為 GBASIC.COM。

載入執行 GBASIC 只要先以正常方式把 CP/M 帶入系統（參考“組立及操作手冊”）。當催促信號 A> 出現後，鍵入：

GBASIC

並按 RETURN 鍵。數秒鐘後會顯示出版權所有的通知，此時 GBASIC 已準備好接受你的命令了。

這個起始過程設定了執行 BASIC 程式時可以 OPEN 3 個檔案（參考 / F 選擇），允許高至 CP/M 之 FDOS 起始點的所有記憶體被使用，並設定資料錄（record）最大容量為 128 拜。

以下的命令格式在機器起始之後可以用來設定這些選擇並 / 或自動 RUN 任意程式：

GBASIC <檔名> [ / F: <檔數> ] [ / M: <最高記憶位置> ]

[ / S: <最大資料錄容量> ] 按下 RETURN 鍵

選用 <檔名> 令其在起始動作完畢後自動 RUN 這一個程式。如果不設 <檔名> 或檔名少於 9 個字元長，GBASIC 就自動設入

.BAS 延伸檔名，使得能夠藉 CP/M 的 SUBMIT 讓 BASIC 程式以整批 ( batch ) 模式執行。整批執行的 BASIC 程式中必須含有 SYS - TEM 陳述 ( statement ) ( 參考第 3 章 )，使得執行完畢一程式就返回至 CP/M，再做整批程式串中的下一個程式。

/ F: <檔數> 選擇執行 BASIC 程式時容許的開檔 ( OPEN ) 數。以這個方式配置的每個檔案資料塊 ( file date block ) 耗用 166 拜加 128 拜 ( 或 / S: 定的數目 ) 記憶體。如果省略了 / F: 選擇，檔數自然值定為 3。 <檔數> 可以用 10 進制，8 進制 ( 前接 & O )，或 16 進制 ( 前接 & H ) 數值表示。

/ M: <最高記憶位置> 設定 MBASIC 可以使用到的最高記憶位置。有時，必須設定可用的記憶位置低於 CP/M 的 FDOS 許多，留下空間儲存組合語言副程式。不論如何，最高記憶位置應低於 FDOS 起始點 ( FDOS 起始位址存在位置 6 及 7 上 )。如果 / M 省略不用，所有高至 FDOS 起始點的記憶體都可使用。 <最高記憶位置> 可以用 10 進制，8 進制 ( 前接 & O )，或 16 進制 ( 前接 & H ) 數值表示。

/ S: <最大資料錄容量> 設定隨機檔使用的資料錄最大容量。可以指定任意 ( 可以大於 128 ) 整數。如果省略 / S，最大資料錄容量設為 128。

起始 BASIC-80 時，系統會顯示：

```
BASIC-80 Version 5.xx
(Apple CP/M Version)
Copyright 1980 (c) by Microsoft
Created: dd-Mmm-yy
xxxx Bytes free
Ok
```

以下為數個不同的起始選擇法：

A > GBASIC - PAYROLL . BAS    使用全部的記憶體及 3 個檔案，載入並執行 PAYROLL BAS

A > GBASIC INVENT / F : 6    使用全部的記憶體及 6 個檔案，載入並執行 INVENT . BAS

A > GBASIC / M : 32768    使用前 32K 記憶體及 3 個檔案。

A > GBASIC DATAACK / F : 2    使用前 36K 記憶體，2 個檔案及執行 DATAACK . BAS。  
/ M : & H 9000

除了高解析度圖形的命令外所有關於 GBASIC 的功能都和 MBASIC 一樣，可以參考 1 至 4 章。高解析度圖形是 GBASIC 增加的功能，說明如下：

### 高解析陳述及命令

這些命令的〈色碼〉可為 0 到 12 的整數

### HGR 的陳述與命令

語法：HGR <幕碼><色碼>

其中〈幕碼〉為 0 到 3 的整數，〈色碼〉為 0 到 7 的整數。

目的：起始高解析度圖形模式。

記要：〈幕碼〉指定所採用的顯示模式：

| 螢幕 # | 清除螢幕 | 螢幕模式               |
|------|------|--------------------|
| 0    | 清除   | 280×160 圖形 + 4 行主文 |
| 1    | 清除   | 280×192 圖形，無主文行    |
| 2    | 不清除  | 280×160 圖形 + 4 行主文 |
| 3    | 不清除  | 280×192 圖形，無主文行    |

如果不給定〈幕碼〉，〈幕碼〉自然設為 0

選擇〈色碼〉以指定所採用的顏色。如果不給定〈色碼〉，自然設為 0。採用〈幕碼〉0 及 1 時，〈色碼〉決定的顏色充滿螢幕。參考 HCOLOR 中顏色名稱及對應的色碼。

例子：10 HGR            與 Applesoft 的 HGR 陳述相同  
 10 HGR 1,2        設 280×192 模式，螢幕為紫色  
 10 HGR 3           設 280×192 模式，不清除螢幕

## HCOLOR

語法：HCOLOR=〈色碼〉

其中〈色碼〉為 0 到 12 的整數

目的：設定畫在高解析度圖形模式的顏色

記要：可用的顏色及其色碼：

|     |     |        |
|-----|-----|--------|
| 0 黑 | 4 黑 | 8 黑 1  |
| 1 綠 | 5 橙 | 9 白 1  |
| 2 紫 | 6 藍 | 10 黑 2 |
| 3 白 | 7 白 | 11 白 2 |
|     |     | 12 反向  |

各不同的黑色與白色之區分：0, 3, 4, 7 畫出極細線。黑1, 白1, 黑2, 白2, (8, 9, 10, 及11) 畫出一個較大的點, 即與用綠, 紫, 橙, 藍畫的點和線同寬的粗線。如果欲在同位置畫同寬度的點或線, 黑1 與白1 應與綠或紫混用, 而黑2 與白2 須與橙或藍混用。

如果裝置是黑白螢幕顯示器則只用0, 3, 4, 7 色碼。〈色碼〉可在HGR 陳述中定義。如果HGR 沒有定義, 〈色碼〉被自動設為0, 直到HCOLOR 陳述出現定義另一顏色為止。

HCOLOR 只能用在高解析度圖形。

因為家用TV 的顯色方式的關係, 以HCOLOR=3 (百)或HCOLOR=7 (白) 畫的高解析度圖形只在(x, y) 及(x+1, y) 兩處都點上時才呈現白色點。如果只點一處(x, y), 當x 為偶數時呈現藍色, x 為奇數時呈現綠色。

## H PLOT

語法1 : H PLOT[〈x1〉,〈y1〉][TO 〈x2〉,〈y2〉]...[TO  
 〈xn〉,〈yn〉]

目的： 在高解析度圖形, 以(x1, y1) (x2, y2) 等點座標定義來畫出點或線。

語法2 : H PLOT TO 〈x2〉,〈y2〉

目的： 由已畫的前一點畫一條到(x2, y2)

記要： 語法1 中, H PLOT 〈x1〉 〈y2〉畫出一點。

H PLOT 〈x1〉,〈y1〉 TO 〈x2〉,〈y2〉 TO ...

$\langle x_n \rangle, \langle y_n \rangle$

由 $\langle x_1 \rangle, \langle y_1 \rangle$ 劃線到另一點，並繼續依順序劃到各點。

這個語法只被螢幕本身限制及不得超過 239 個字元的限制。

語法 1 中，點及線的顏色由最近的 HCOLOR 陳述定義。如果不曾定義顏色，就自然設顏色為 0。

語法 2 中，線的顏色亦由最近的 HCOLOR 陳述決定。如果不曾畫過前一點，不能使用語法 2。

H PLOT 只用在高解析度圖形模式。

例子：

```
10 HGR
20 COLOR=2
30 H PLOT 24,125 TO 100,12 TO 270,1
```

## HSCRN

語法：HSCRN(x,y)

動作：如果(x,y)點存在，就回輸“-1”。

記要：HSCRN 和 SCRN 不同，HSCRN 不分辨顏色。X範圍在 0-279，Y範圍在 0-191。

例子：

```
10 HGR: COLOR=3
20 H PLOT 0,100 TO 279,100
30 PRINT HSCRN(46,100), HSCRN(20,20)
RUN
-1      0
```

## 附 錄 A

# BASIC-80 5.0版的新增特點

由Microsoft BASIC 4.51 或更早期的版本上所做的 BASIC 程式在執行時可能會被 5.0 版本的新增特點影響。執行這些程式之前，應先考慮下列幾個項目：

1. 新增的保留字有：CALL, CHAIN, COMMON, WHILE, WEND, WRITE, OPTION BASE, RANDOMIZE。
2. 由浮點值轉換成整數值時，是以四捨五入方式做而不是以刪除小數點後的值來做。這一點不僅影響所安排的陳述（例如  $I \% = 2.5$  結果是  $I \% = 3$ ），也影響了函數及陳述的計算（例如，TAB(4.5) 變成移到第 5 位置，A(1.5) 就是 A(2)， $X = 11.5 \text{ MOD } 4$  得 0）。
3. 在 FOR ..... NEXT 迴圈上，如果迴圈初值乘步距的正負號（sign of step）小於末值乘步距的正負號，此迴圈是不正確的迴圈，會被跳過不做。
4. 分母為零或溢位（overflow）不再造成嚴重錯誤。參考第 2 章。
5. RND 已作修改，不帶引數的 RND 相同於原來帶正值引數



的 RND 。除非 RANDOMIZE 被使用，每次 RUN 時 RND 產生相同順序的隨機數列。參考第 3，4 章。

6. PRINT 出單準數值及倍準數值的規則已做了修改。參考第 3 章的 PRINT。
7. CLEAR 陳述可以動態配置字串空間，第一引數設定可用的記憶體最高位置，第二引數設定堆疊 ( stack ) 空間大小。參考第 3 章的 CLEAR。
8. 帶有太多或太少的項目 ( item )，或錯誤的資料型式 ( 例如以數值代字串 )，或帶回車字元 ( carriage return ) 的 INPUT 陳述會造成 “ ? Redo from start ” 錯誤訊息。直到輸入可接受的資料之後，才可執行。
9. 在 PRINT USING，增用兩個新的欄格式字元 “ & ” 及 “ \_ ”。“ & ” 用於變化長度的字串欄。“ \_ ” 示意出格式字串中的文義字元 ( literal character )。
10. WIDTH 陳述的表式 ( expression ) 若給定 255，BASIC 使用 “無限長” 的行寬度，也就是不插入回車字元 ( carriage return )。WIDTH LPRINT 可用來設列印機的行寬度。參考第 3 章的 WIDTH。
11. “ @ ” 及 “ \_ ” 不再用作編輯字元 ( editing character )。
12. 變數名稱可以長達 40 個字元，在其中可以插入使用保留字。但保留字必須以空格 ( space ) 區分開。為了維持與早期 BASIC 版本的通用性，空格會被自動加入保留字與變數名稱之間。注意：若原行長度已接近 255，由於空格的加入，

- 13 BASIC 程式可藉“受保護的二進制格式”保存起來。參考第 3 章的 SAVE。
- 14 保留字必須用空格加在其前後兩端。

### 用 GBASIC 載入及保存 HIRES 圖像。

以下一個短的 GBASIC 程式示意如何使用隨機磁碟 I/O 的陳述去載入 Hires 圖像並回存至磁碟。用這種方法載入及保存 Hires 圖像與使用 Apple DOS 的 BSAVE 及 BLOAD 陳述時做得一樣快甚至更快。以下程式也會造出一些美麗的 Hires 圖像。

```
10 DEFINT A-Z:DEFSNG A-F,K
20 DIM X(23),Y(23)
30 GOTO 4000

1000 HGR 1,3:MCOLOR=0
1010 HPLLOT 140,96
1020 FOR A=0 TO 3.14159*20 STEP .05
1030 R=SIN(A*2.9)
1040 HPLLOT TO 140+107#R/COS(A),96+95#R#SIN(A)
1050 NEXT
1060 HGR 1,12:FOR T=0 TO 500:NEXT:HGR 1,12
1070 GET A$:GOTO 4000

2000 N=INT(RND*14)+13
2010 PI=6.28318/N:FOR I=0 TO N-1:A=PI*I
2020 X(I)=COS(A)*107+140:Y(I)=SIN(A)*95+96
2030 NEXT
2040 HGR 1,0:MCOLOR=3
2050 FOR I=0 TO N-1:FOR J=1 TO N-1:HPLLOT X(I),Y(I) TO X(J),Y(J):NEXT:NEXT
2060 HGR 1,12:FOR T=0 TO 200:NEXT T:HGR 1,12
2070 GET A$:GOTO 4000

3000 HOME:VTAB 4
3010 ON ERROR GOTO 3020:FILES "*.PIC":PRINT:PRINT:ON ERROR GOTO 0
3020 PRINT "Load or Save (L/S) ?":GET B$
3030 IF B$="S" OR B$="s" THEN D=1:HGR 3:PRINT "Save"
      ELSE D=0:HGR 1+0:PRINT "Load"
3040 PRINT:INPUT "File name? *F$:"F$:IF F$="" THEN 3040
3050 IF INSTR(F$,".")=0 THEN F$=F$+".PIC"

3060 OPEN "R",F$
3070 FIELD#1,128 AS A$:B$=A$
3080 M=16:L=0:P=VARPTR(B$)+1
3090 FOR J=1 TO 64
3100 POKE P,L:POKE P+1,M
3110 IF D=1 THEN LSET A$=B$:PUT 1 ELSE GET 1:LSET B$=A$
3120 L=L XOR 128:IF L=0 THEN M=M+1
3130 NEXT:CLOSE
3140 FOR T=0 TO 1500:NEXT T

4000 TEXT:HOME:VTAB 4:PRINT TAB(5)"### HIRES GRAPHICS DEMO ###"
4010 VTAB 7:PRINT TAB(5)"1. Rose":PRINT:PRINT TAB(5)"2. Polsson":PRINT
4020 PRINT TAB(5)"3. Load/Save Hires Picture":PRINT:PRINT TAB(5)"Which - ?"
4030 A$=INPUT$(1):A$=VAL(A$):IF A$ 3 OR A$ 1 THEN 4030
4040 PRINT A$:PRINT:PRINT TAB(5)"Workina...!"
4050 ON A GOTO 1000,2000,3000
```

註：用 APDOS 自 Apple DOS 轉送至 CP/M 的 Hires 圖像並不能  
用上述程式正確地載入。因為這些檔案的前 4 拜含有目標位址  
及圖像長度兩資料，這 4 拜必須先除掉。這樣做也會省下 1 K 磁  
碟空間。用下列程序可以修改轉送過來的 Apple DOS 圖像。（  
鍵入“\_\_”字元）

```
DDT
DDT VERS 2.2
Ifilename.ext
RFC
M200,2200,100
GO
A>SAVE 32 filename.ext
```

## 附 錄 B

# BASIC-80磁碟I/O

本附錄為BASIC-80的初使用者說明BASIC-80磁碟I/O的操作程序。如果你是使用BASIC-80的新手，或是在操作時遭遇到相關於磁碟的錯誤時，請熟看本附錄的操作程序及程式例子以確保正確地使用各磁碟陳述。

在任何需要檔名的磁碟命令或陳述中，都必須使用符合於作業系統要求的名字。CP/M作業系統會將在SAVE, RUN, MERGE, LOAD命令中的檔名後自動加上“.BAS”

### 程式檔的命令

這裏重提一下處理程式檔可用的命令及陳述。

- SAVE “檔名”〔,A〕 把現今在記憶體中的程式寫入磁碟。若選用A，使程式以一系列的ASCII碼代號寫入磁碟。〔否則BASIC採用壓縮的二進制格式（compressed binary format）寫檔〕
- LOAD “檔名”〔,R〕 由磁碟把程式載入記憶體。若選用R，則令程式載入後立即執行。LOAD在載入

程式之前會關掉 ( close ) 所有檔案，刪除掉記憶體原內容。但是如果選用 R，已開 ( open ) 的資料檔 ( data file ) 仍不關掉。這樣做使得前後的程式可以互相鏈接或一段段地載入並取用相同的資料檔。

RUN “檔名” [ , R ]

載入程式並執行之。在載入程式前會關掉所有檔案，刪除掉記憶體原內容。但如果選用 R，已開的所有資料檔不會關掉。

MERGE “檔名”

自磁碟載入程式但不除掉記憶體原內容在磁碟中的程式的行號 ( line numbers ) 與記憶體中的程式的行號合併共存之。如果出現兩個同號碼行，則保留下磁碟的程式行。MERGE 命令做完後 BASIC 返回命令階層 ( command level )，已合併的程式留在記憶體中。

KILL “檔名”

刪除磁碟檔。“檔名”可為程式檔，循序或隨機存取資料檔。

NAME

更改磁碟檔的名稱。NAME 陳述的語法是 NAME “舊檔名” AS “新檔名”。NAME 可用於程式檔，隨機檔，或循序檔。

## 保護檔案

在 SAVE 命令中選用 P 就可以保護這個被保存的程式免於被列印出或被編輯 ( edited ) 。例如：

```
SAVE "MYPROG".P
```

## 磁碟資料檔 —— 循序及隨機 I/O

有兩種型式的資料檔可被 BASIC-80 程式創建及存取，就是：循序檔及隨機存取檔。

### 循序檔 ( sequential File )

循序檔比隨機檔容易創建，但是可變化性不足，存取資料的速度也慢。寫入並儲存於循序檔的資料是以一項目接著一項目 ( 循序地 ) 送入並依相同順序一項項地被讀出。循序檔所使用的陳述及函數包括：

|       |              |             |        |
|-------|--------------|-------------|--------|
| OPEN  | PRINT#       | INPUT#      | WRITE# |
|       | PRINT# USING | LINE INPUT# |        |
| CLOSE | EOF          | LOC         |        |

創建及存取循序檔資料所需的程式步驟如下：

1. OPEN 檔案，設檔於 " O " 模式。
 

```
OPEN "O",#1,"DATA"
PRINT# 1,A$;B$;C$
```
2. 使用 PRINT# 陳述將資料寫入檔中。  
( 也可以用 WRITE# )

3 應先 CLOSE 檔再 OPEN 它成爲“ I ”

模式，才可取得檔的資料。

```
CLOSE #1
OPEN "I",#1,"DATA"
```

4 使用 INPUT# 陳述將資料由循序檔讀入 INPUT#1,X\$,Y\$,Z\$ 程式中。

以下的 B-1 是個創建循序檔“ DATA ”的程式，檔的資料由終端機輸入。

```
10 OPEN "O",#1,"DATA"
20 INPUT "NAME":N$
25 IF N$="DONE" THEN END
30 INPUT "DEPARTMENT":D$
40 INPUT "DATE HIRED":H$
50 PRINT #1,N$;" ";D$;" ";H$
60 PRINT:GOTO 20
RUN
```

```
NAME? MICKEY MOUSE
DEPARTMENT? AUDIO/VISUAL AIDS
DATE HIRED? 01/12/72
```

```
NAME? SHERLOCK HOLMES
DEPARTMENT? RESEARCH
DATE HIRED? 12/03/65
```

```
NAME? EBENEZER SCROOGE
DEPARTMENT? ACCOUNTING
DATE HIRED? 04/27/78
```

```
NAME? SUPER MANN
DEPARTMENT? MAINTENANCE
DATE HIRED? 08/16/78
```

```
NAME? etc.
```

### 程式 B-1 創建循序資料檔

程式 B-2 用來取得程式 B-1 所建“ DATA ”檔的資料，將 1978 年新顧用人員名單顯示出來。

```

10 OPEN "1",#1,"DATA"
20 INPUT#1,N$,D$,H$
30 IF RIGHT$(H$,2) = "78" THEN PRINT N$
40 GOTO 20
RUN
EBENEZER SCROOGE
SUPER MANN
Input past end in 20
Ok

```

### 程式 B-2 讀取循序檔

B-2 程式循序地讀入檔中每一個項目。全部資料讀入後，在第 20 行造成“Input past end”（輸入超越結束點）錯誤訊息。為避免發生這個訊息，可以插入 15 行利用 EOF 函數檢查檔案的結束點（end-of-file）：

```
15 IF EOF(1) THEN END
```

並把第 40 行改成 GO TO 15。

創建循序檔的程式本身也可以藉 PRINT# USING 陳述把格式化的資料寫入磁碟。例如，

```
PRINT#1,USING"###.#.",":A,B,C,D
```

可以把數值資料寫入磁碟，而不明確地劃界。格式串後端的逗點用來分隔磁碟檔中各項目。

用在循序檔時，LOC 函數回輸在開檔之後，已寫入或讀出檔案的段落（sector）數目。其中每段落含 128 拜資料。

### 資料加入循序檔

如果在磁碟中已有循序檔，不可以單純地用“OPEN 此檔為



“O”模式再寫入資料”的方式在此循序檔尾部加入額外資料。因為OPEN循序檔成“O”模式時，原有內容會立即被破壞掉。以下程序說明如何在已有的“NAMES”循序檔加入額外資料。

1. OPEN“NAMES”檔為“I”模式。
2. OPEN另一檔“COPY”成爲“O”模式。
3. 讀入“NAMES”的資料並將這些資料寫到“COPY”。
4. CLOSE“NAMES”並KILL掉“NAMES”。
5. 在“COPY”中寫入新資料。
6. 重定名“COPY”成爲“NAMES”並CLOSE它。
7. 這時候在磁碟中就有了包含所有新舊資料的檔案“NAMES”

B-3 程式說明了這個方法。B-3 可用來創建或加資料入NAMES檔。B-3 程式也顯示出用LINE INPUT# 由磁碟檔讀入帶有逗點字串的方法。別忘了，LINE INPUT# 由磁碟不斷讀入字元直到遇到回車（carriage return）字元或讀到255個字元才停止（遇到括號或逗點並不停止讀入）。

```

10 ON ERROR GOTO 2000
20 OPEN "I",#1,"NAMES"
30 REM IF FILE EXISTS, WRITE IT TO "COPY"
40 OPEN "O",#2,"COPY"
50 IF EOF(1) THEN 90
60 LINE INPUT #1,A$
70 PRINT #2,A$
80 GOTO 50
90 CLOSE #1
100 KILL "NAMES"
110 REM ADD NEW ENTRIES TO FILE
120 INPUT "NAME";N$
130 IF N$="" THEN 200 'CARRIAGE RETURN EXITS INPUT LOOP
140 LINE INPUT "ADDRESS? ";A$
150 LINE INPUT "BIRTHDAY? ";B$
160 PRINT #2,N$
170 PRINT #2,A$
180 PRINT #2,B$
190 PRINT:GOTO 120
200 CLOSE
205 REM CHANGE FILENAME BACK TO "NAMES"
210 NAME "COPY" AS "NAMES"
2000 IF ERR=53 AND ERL=20 THEN OPEN "O",#2,"COPY":RESUME
    120
2010 ON ERROR GOTO 0

```

### 程式 B-3 加入資料於循序檔

第 2000 行的錯誤捕捉常式捕捉第 20 行發生的 File does not exist ” (檔不存在) 錯誤。如果發生這項錯誤，複製檔案的陳述會被跳過不執行，“COPY” 檔則被創建視為新的輸入檔。

### 隨機檔

創建及存取隨機檔需要較多項的程式步驟，但使用隨機檔有許多好處。好處之一是隨機檔佔用較少磁碟空間，因為 BASIC 以“緊密二進制格式”(packed binary format) 將資料存入磁碟。(循序檔則以一系列 ASCII 字元方式把資料存入磁碟)

資料可以隨機存取是隨機檔最大的好處。可以任意存取磁碟任一

位置；而不需要像循序檔依順序讀入所有的資料。可以隨意存取是因爲資料被存在稱爲“資料錄”(record)的明確儲存單位中，每個資料錄都有固定編號。

使用隨機檔所需的陳述及函數是：

|       |       |           |     |
|-------|-------|-----------|-----|
| OPEN  | FIELD | LSET/RSET | GET |
| PUT   | CLOSE | LOC       |     |
| MKI\$ | CVI   |           |     |
| MKS\$ | CVS   |           |     |
| MKD\$ | CVD   |           |     |

## 創建隨機檔

創建隨機檔應利用下列程式步驟：

1. OPEN 檔案成爲隨機存取檔(“R” OPEN “R”.#1.“FILE”.32 模式)。此例中指定資料錄長爲32拜。如果沒有指定長度，資料錄長自然定爲128拜。
2. 用 FIELD 陳述配置隨機緩衝區 (random buffer) 的變數空間，這些變數其後將被寫入隨機檔。  
FIELD #1 20 AS N\$,  
4 AS A\$, 8 AS P\$
3. 用 LSET 將資料移入緩衝區。數目值放入緩衝區之前，必須轉成字串。可以利用“make”函數：MKI\$ 把整數值轉成字串，MKS\$ 轉換單準實數值，MKD\$ 轉換倍準實數值。  
LSET N\$=X\$  
LSET A\$=MKI\$(AMT)  
LSET P\$=TEL\$
4. 用 PUT 陳述把資料自緩衝區寫入磁碟。  
PUT #1.CODE%

B-4 程式將資料由終端機取入並寫到隨機檔。每執行一個 PUT 陳述，寫入一個資料錄到檔案。在第 30 行輸入的 2 位數碼是資料錄的號碼。

註：勿在 INPUT 或 LET 陳述使用已欄化 ( FIELDed ) 字串變數。這樣做會造成變數的指標指入字串空間 ( string space ) 而不指向隨機檔的緩衝區。

```

10 OPEN "R" #1,"FILE"
20 FIELD #1,20 AS N$, 4 AS A$, 8 AS P$
30 INPUT "2-DIGIT CODE";CODE%
40 INPUT "NAME";X$
50 INPUT "AMOUNT";AMT
60 INPUT "PHONE";TEL$:PRINT
70 LSET N$=X$
80 LSET A$=MK$(AMT)
90 LSET P$=TEL$
100 PUT #1,CODE%
110 GOTO 30
    
```

#### 程式 B-4 創建隨機檔

#### 存取隨機檔

存取隨機檔要用下列程式步驟：

1. OPEN 檔案成爲 " R " 模式。

```

OPEN "R",#1,"FILE",32
FIELD #1 20 AS N$,
4 AS A$, 8 AS P$
    
```

2. 用 FIELD 陳述配置隨機緩衝區的變數空間，這些變數其後將被讀入。

註：程式同時對隨機檔輸入並輸出時，可以只用一個 OPEN 陳述及一個 FIELD 陳述。

3. 用 GET 陳述把所要的資料錄取入隨機緩衝區。

```

GET #1,CODE%
    
```

4. 在緩衝區中的資料可被程式取用。原來的數目值必須先用“Convert”函數轉回；CVI 用於整數，CVS 用於單準實數值，CVD 用於倍準實數值。

```
PRINT N$
PRINT CVS(A$)
```

B-5程式讀取B-4程式所建的隨機檔內容。在終端機上輸入2位數碼，相關的資料就會由檔中讀取並顯示出來。

```
10 OPEN "R",#1,"FILE"
20 FIELD #1, 20 AS N$, 4 AS A$, 8 AS P$
30 INPUT "2-DIGIT CODE";CODE%
40 GET #1, CODE%
50 PRINT N$
60 PRINT USING "$$###.##";CVS(A$)
70 PRINT P$;PRIN
80 GOTO 30
```

#### 程式 B-5 讀取隨機檔

用於隨機檔的LOC函數，回輸“現今資料錄號碼”。現今資料錄號碼是用於GET或PUT陳述的前次資料錄號碼加1。例如：

```
IF LOC(1)>50 THEN END
```

如果在檔案#1的現今資料錄號碼大於50，就終止程式執行。

B-6是個庫存程式，在此用來說明隨機檔的存取動作。這個程式中，以資料錄號碼作為物品的“件號”，在這裏假定物品不超過100件。第900~960行寫入CHR\$(255)作為各資料錄第一字元以起始設定資料檔。這字元在其後(第270行及第500行)被用來決定所對應的那個件號的進入點是否已經存在。

第130~220行顯示出本程式中含有的各種庫存函數。當使用者鍵入所需要的函數號碼時，第230行跳到適當的副程式。

## PROGRAM B-6 - INVENTORY

```

120 OPEN "R", #1, "INVEN.DAT", 39
125 FIELD #1, 1 AS F$, 30 AS D$, 2 AS Q$, 2 AS R$, 4 AS P$
130 PRINT:PRINT "FUNCTIONS":PRINT
135 PRINT 1, "INITIALIZE FILE"
140 PRINT 2, "CREATE A NEW ENTRY"
150 PRINT 3, "DISPLAY INVENTORY FOR ONE PART"
160 PRINT 4, "ADD TO STOCK"
170 PRINT 5, "SUBTRACT FROM STOCK"
180 PRINT 6, "DISPLAY ALL ITEMS BELOW REORDER LEVEL"
220 PRINT:PRINT:INPUT "FUNCTION":FUNCTION

225 IF (FUNCTION<1) OR (FUNCTION>6) THEN PRINT "BAD FUNCTION
      NUMBER":GOTO 130
230 ON FUNCTION GOSUB 900,250,390,480,560,680
240 GOTO 220
250 REM BUILD NEW ENTRY
260 GOSUB 840
270 IF ASC(F$)<>255 THEN INPUT "OVERWRITE":A$:IF A$ <> "Y" THEN
      RETURN
280 LSET F$=CHR$(0)
290 INPUT "DESCRIPTION":DESC$
300 LSET D$=DESC$
310 INPUT "QUANTITY IN STOCK":Q%
320 LSET Q$=MKI$(Q%)
330 INPUT "REORDER LEVEL":R%
340 LSET R$=MKI$(R%)
350 INPUT "UNIT PRICE":P
360 LSET P$=MKS$(P)
370 PUT #1, PART%
380 RETURN
390 REM DISPLAY ENTRY
400 GOSUB 840
410 IF ASC(F$)=255 THEN PRINT "NULL ENTRY":RETURN
420 PRINT USING "PART NUMBER ###":PART%
430 PRINT D$
440 PRINT USING "QUANTITY ON HAND #####":CVI(Q$)
450 PRINT USING "REORDER LEVEL #####":CVI(R$)
460 PRINT USING "UNIT PRICE $$$###":CVS(P$)
470 RETURN
480 REM ADD TO STOCK
490 GOSUB 840
500 IF ASC(F$)=55 THEN PRINT "NULL ENTRY":RETURN
510 PRINT D$:INPUT "QUANTITY TO ADD ":A%
520 Q%=CVI(Q$)+A%
530 LSET Q$=KI$(Q%)
540 PUT #1, PART%
550 RETURN
560 REM REMOVE FROM STOCK
570 GOSUB 840
580 IF ASC(F$)=255 THEN PRINT "NULL ENTRY":RETURN
590 PRINT D$

```

## 150 APPLE 軟體卡

```
600 INPUT "QUANTITY TO SUBTRACT";S%
610 Q%=CVI(Q$)
620 IF (Q%-S%)<0 THEN PRINT "ONLY";Q%;" IN STOCK":GOTO 600
630 Q%=Q%-S%
640 IF Q%=<CVI(R$) THEN PRINT "QUANTITY NOW";Q%:" REORDER
    LEVEL";CVI(R$)
650 LSET Q$=MKI$(Q%)
660 PUT # 1,PART%
670 RETURN
680 DISPLAY ITEMS BELOW REORDER LEV
690 FOR I= 1 TO 100
710 GET # 1,I
720 IF CVI(Q$)<CVI(R$) THEN PRINT D$:" QUANTITY";CVI(Q$) TAB(50)
    "REORDER LEVEL";CVI(R$)
730 NEXT I
740 RETURN
840 INPUT "PART NUMBER";PART%
850 IF(PART%<1) OR (PART%>100) THEN PRINT "BAD PART NUM-
    BER":GOTO 840 ELSE GET # 1,PART%:RETURN
890 END
900 REM INITIALIZE FILE
910 INPUT "ARE YOU SURE";B$:IF B$<>"Y" THEN RETURN
920 LSET F$=CHR$(255)
930 FOR I= 1 TO 100
940 PUT # 1,I
950 NEXT I
960 RETURN
```

### 程式 B - 6 庫存程式

## 順序 I/O 隨機檔

我們也可以執行隨機磁碟檔的順序 I/O 運作。雖然，一般而言，速度較慢，但很類似 Apple DOS 隨機檔 I/O，非常有用。

```
20
30 OPEN "A":1,"RND.TXT"
40 FOR I=1 TO 20
50 WRITE#1,"RECORD ",I," ### IK JK JL ###"
60 PUT 1,I
70 NEXT I
80 CLOSE
90
100
110 OPEN "A":1,"RND.TXT"
120 FOR I=20 TO 1 STEP -1
130 GET 1,I
140 INPUT#1,A$;R;D$
150 PRINT A$;R;D$
160 NEXT I
170 CLOSE
```

## 附 錄 C

### 組合語言副程式

所有的 BASIC-80 版本都提供了與組合語言副程式的溝通功能。USR 函數使得組合語言副程式與 BASIC 內部函數具有相同被呼叫使用的方式。

FRCINT 的位址是 103H，MAKINT 的位址是 105H。

107H 位置含有 CP/M BIOS 進入點的高位拜，用來直接呼叫 BIOS。BASIC 動作之後，位置“0”的 JMP 指令跳到 BASIC 的“Reset error”而非跳到 BIOS 的暖啟動常式。

#### 配置記憶體

在載入組合語言副程式之前必須安排出足夠記憶空間。在啓始時，設入的最高記憶位置應扣除掉組合語言副程式所需佔用記憶空間的數量。因為 BASIC 使用掉自它所佔用位置以上的所有記憶體，所以只有最高位置記憶體可以給予使用者的副程式儲存用。

組合語言副程式被呼叫時，堆疊指標（stack pointer）被設成 8 層（16 拜）堆疊儲存區。如果需要更多堆疊空間，組合語言副程式可以將 BASIC 的堆疊保存起來，另設新的堆疊使用之。如果這樣做，在由副程式返回之前必須先回復 BASIC 的堆疊。

組合語言副程式可由系統監督程式或 BASIC 的 POKE 陳述載入記憶體，或這些副程式（如果使用者手邊有 MACRO-80 或



FORTRAN-80 時)可用 MACRO-80 組合編譯,並由 LINK-80 載入記憶體。

### USR 函數呼叫——

USR 函數語法為

USR [ <數碼> ] (引數)

其中 <數碼> 範圍為 0 到 9, 引數可以是任意數值或字串表式。  
<數碼> 指定出某個 USR 被呼叫, 它對應於 DEF USR 陳述對那一個程式定的號碼。如果沒有指定出 <數碼>, 自然變成呼叫 USR 0  
DEF USR 陳述所給的位址決定副程式起始位址。

USR 函數被呼叫時, 暫存器 A 含有一值, 此值指出所需要的引數資料型式:

A 的值 引數型式

- 2 兩拜的整數 (以 two's complement 表示)
- 3 字串
- 4 單準浮點實數 (floating point number)
- 5 倍準浮點實數

如果引數是數值, (H, L) 暫存器指向浮點累積器 (Floating Point Accumulator (FAC)), 引數應被存入在 FAC 中。

如果引數是個整數:

FAC-3 含引數的低位 8 比次 (bits)

FAC-2 含引數的高位 8 比次。

如果引數是個單準浮點數:

FAC-3 含尾數 (mantissa) 的最低位 8 比次。

FAC-2 含尾數的中間 8 比次。

FAC-1 含尾數的最高位 7 比次，其最高比次為“1”。比次 7 是數值的正負號（0 表示正數，1 表示負數）。

FAC 是指數減掉 128，二進制小數點在尾數的最高比次左邊。

如果引數是倍準浮點數：

FAC-7 到 FAC-4 含額外的 4 個尾數拜，其中 FAC-7 是最低位拜。

如果引數是個字串，〔D，E〕暫存器指向被稱為“字串描述區”的三拜位置。字串描述區的拜 0 含字串長度（0 到 255），拜 1 及拜 2 各是字串空間的字串起始位址的低拜及高拜。

注意：如果引數是在程式中的一個文義字串（string literal），字串描述區會指向程式主文。此時要小心地避免改變或破壞掉你的程式。為防止不可測的結果，可以加上“”至程式的文義字串上。例如：

```
A$ = "BASIC-80"+""
```

這樣做會把文義字串複製入字串空間（string space）防止呼叫副程式時更改程式主文。

通常由 USR 回輸的值具有與引數相同的資料型式（整數，字串，單準實數值，或倍準實數值）。但 MAKINT 函數可以令回輸的值轉成整數型式，它把回輸函數值放在〔H，L〕中，可用以下順序自副程式返回時執行 MAKINT。

```

PUSH    H
LHLD    xxx
XTHL

RET

```

不論函數的引數是什麼型式的值，也可以藉呼叫 FRCINT 常式令引數先轉成整數值（放在〔H，L〕中）。可用以下順序：

```

LXI     H
PUSH    H
LHLD    xxx
PCHL
SUB1:  ....

```

## CALL陳述

使用者的函數（程式）可利用 CALL 陳述呼叫 Z-80 組合語言副程式或 6502 組合語言副程式。參考第三章的 CALL。

## 呼叫Z-80副程式

所用的呼叫順序與Microsoft的FORTRAN, COBOL, 及BASIC編譯程式中的相同。

不帶引數的 CALL 陳述只是個單純的呼叫指令。所對應的副程式最後必須經由“RET”返回。（CALL及RET為8080 運算碼參考8080 相關資料）

具有引數的 CALL 陳述需用較複雜的呼叫順序。在 CALL 的引數串列（argument list）中的每一個引數是由一參數將它們傳送入副程式。這個參數只是引數的低位拜位址。因此不論引數是那一種型式值，參數都佔兩個拜。

傳送參數的方法與參數的數量有關：

1. 如果參數少於或等於 3 個，則傳送入暫存器之中。參數 1 在 HL，參數 2 在 DE，參數 3 在 BC。
2. 如果參數多於 3 個：
  - (1) 參數 1 在 HL。
  - (2) 參數 2 在 DE。
  - (3) 參數 3 至 n 存在一連續的資料塊中 ( data block )，BC 暫存器內容指向此資料塊的低位拜。(即指向第 3 參數的低位拜)

這種方式下，副程式本身必須清楚應有多少個參數該被取用。而呼叫程式也有義務傳送正確個數的參數。參數的型式及數量並不被檢查是否正確。

如果有個副程式需要三個以上的參數，並要把它們傳送到某個局部資料區 ( local data area )，可以利用稱為 \$AT 的引數傳送常式 (是個在 FORTRAN 程式庫 ( FORLIB.REL ) 中的系統副程式)。呼叫時，HL 指向局部資料區，BC 指向第 3 個參數，暫存器 A 內存傳送的引數數量 (即全部數量減 2)。在呼叫 \$AT 之前，此副程式必須先負責把前兩個參數存起來。下例是傳送五個參數的副程式。

```

SUBR: SHLD P1 ;SAVE PARAMETER 1
      XCHG
      SHLD P2 ;SAVE PARAMETER 2
      MVI A,3 ;NO. OF PARAMETERS LEFT
      LXI H,P3 ;POINTER TO LOCAL AREA
      CALL $AT ;TRANSFER THE OTHER 3 PARAMETER

```

[Body of subroutine]

```

      RET ;RETURN TO CALLER
P1: DS 2 ;SPACE FOR PARAMETER 1
P2: DS 2 ;SPACE FOR PARAMETER 2
P3: DS 6 ;SPACE FOR PARAMETERS 3-5

```

引數傳送常式 \$AT 的程式串列如下：

```

00100 ; ARGUMENT TRANSFER
00200 ;[B,C] POINTS TO 3RD PARAM.
00300 ;[H,L] POINTS TO LOCAL STORAGE FOR PARAM 3
00400 ;[A] CONTAINS THE # OF PARAMS TO XFER(TOTAL-2)
00500
00600
00700 ENTRY: $AT ;SAVE [H,L] IN [D,E]
00800 $AT: XCHG
00900 MOV H,B
01000 MOV L,C ;[H,L] = PTR TO PARAMS
01100 AT1: MOV C,M
01200 INX H
01300 MOV B,M
01400 INX H ;[B,C] = PARAM ADR
01500 XCHG ;[H,L] POINTS TO LOCAL STORAGE
01600 MOV M,C
01700 INX H
01800 MOV M,B
01900 INX H ;STORE PARAM IN LOCAL AREA
02000 XCHG ;SINCE GOING BACK TO AT1
02100 DCR A ;TRANSFERRED ALL PARAMS?
02200 JNZ AT1 ;NO, COPY MORE
02300 RET ;YES, RETURN

```

別忘了參數只是個指向真正引數的指標

**註**

現在我們已經完全了解呼叫程式中的引數無論數字，型式，與長度都必須與被呼叫副程式中的參數相符。不論是 BASIC 副程式或者是以組合語言寫的副程式都必須如此。

**呼叫6502副程式**

對 6502 副程式的 CALL 的語法與對 Z-80 副程式 CALL 不同，必須在變數名稱前多加個“%”符號。呼叫 6502 副程式時，使用的參數可達 3 個。所有參數都是單拜參數。傳送方法如下：

- 1 參數 1 在 6502 的 A 暫存器
- 2 參數 2 在 6502 的 X 暫存器
- 3 參數 3 在 6502 的 Y 暫存器

例如：

```
CALL % ROUTINE (10,20,30)
```

其中 10 存入 A，20 存入 X，30 存入 Y。呼叫時參數可有可無視需要給定之。

**中斷**

組合語言副程式也可以寫來做中斷處理工作。所有的中斷處理常式都必須做保存堆疊，A-L 暫存器及 PSW 的工作。因為中斷動作會自動制止 (disable) 後來的中斷發生，常式在返回前需重開放 (re-enable) 中斷的發生。CP/M BASIC 中，所有的中斷向量指標皆可供利用。參考 SoftCard 第二部分的“軟體及硬體詳



## 附 錄 D

# 由非 Applesoft 的 BASIC 程式轉換成 BASIC-80 程式

非 BASIC-80 的程式在 BASIC-80 上執行之前必須做少許更改。在此提出一些轉換 BASIC 程式所要做的事。

### 字串維數 ( Dimension )

刪除掉任何宣告字串長度的陳述。具有 J 個 I 字元長的元素的字串陣列之維數陳述 DIM A\$( I , J ) 應轉換成 BASIC-80 的陳述 DIM A\$( J )。

有些 BASIC 使用逗點或 “ & ” 符號作為字串的連結 ( concatenation ) 。它們都應被改成 BASIC-80 字串連結運算子 “ + ” 符號。

BASIC-80 中，MID\$, RIGHT\$, 及 LEFT\$ 函數用來取出字串的一部份。在別的基本中，用來取第 I 字元的 A\$( I ) 或取第 I 至第 J 字元的 A\$( I , J ) 應改成：

其他 BASIC

X\$=A\$(I)  
X\$=A\$(I,J)

BASIC-80

X\$=MID\$(A\$,I,1)  
X\$=MID\$(A\$,I,I+1)



如果字串的字元是由左邊算起，若欲以 X\$ 字串代換 A\$ 中部分字元時：

其他 BASIC

A\$(I)=X\$  
A\$(I,J)=X\$

BASIC-80

MID\$(A\$,1,1)=X\$  
MID\$(A\$,I,J-I+1)=X\$

### 多重指定 ( Assignments )

有些 BASIC 容許下列型式的陳述：

10 LET B=C=0

使 B 和 C 都等於零。但 BASIC-80 會把左邊等號視為邏輯運算子 ( operator )，變成當 C = 0 時設定 B 為 -1。所以應將上式改為兩個指定陳述。

10 C=0:B=0

### 多重陳述 ( Statements )

有些 BASIC 使用 “ / ” 符號在一行上分開兩個陳述。在 BASIC-80 則使用冒號 “ : ” 分開行上各陳述。

### MAT 功能函數

在許多 BASIC 中的函數 MAT 必須改寫成 FOR... NEXT 迴圈。

# 附 錄 E

## 錯誤碼與錯誤訊息

### 號碼

### 訊息

- 1 “NEXT without FOR” 沒有 FOR 但遭遇 NEXT。  
NEXT 陳述的變數沒有對應的執行事件。FOR 陳述的變數不匹配。
- 2 “Syntax error” 語法錯誤。  
遭遇到含有不正確字串順序的敘述行（例如不配合的括號，拼錯的命令或陳述，不正確的標點，等等）。
- 3 “Return without GOSUB” 沒有 GOSUB 但遭遇返回陳述。  
前面沒有 GOSUB，但遭遇 RETURN 陳述。不配合的 GOSUB 陳述。
- 4 “Out of data” 沒有資料。  
當程式中已經沒有可讀的資料時，却執行了 READ 陳述。
- 5 “Illegal function call” 不合法的呼叫函數。  
超出限定範圍值的參數傳送給數學或字串函數。以下結果也造成 FC 錯誤。

1. 負的或太大得不合理的足碼 ( subscript )
2. LOG 帶負或“零”引數
3. SQR 帶有負引數
4. 帶有非整數指數 ( non-integer exponent ) 的負尾數 ( negative mantissa )
5. 呼叫一個未給定起始位址的USR 函數。
6. 帶有不適當引數的MID\$, LEFT\$, RIGHT\$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING\$, SPACE\$, INSTR, 或 ON ..... GOTO

6 “Overflow” 溢位

計算結果太大以致無法用BASIC-80 數目格式表示出來。如果不足位 ( underflow ) 一發生，結果為 0，不造成錯誤，程式可繼續執行。

7 “Out of memory” 記憶體用完

程式太大，有太多FOR 迴圈或GOSUB，太多變數，或太複雜的表式 ( expression )。

8 “Undefined line” 未定義之行

GOTO, GOSUB, IF ..... THEN ..... ELSE, 或DELETE 中的對應指定之行並不存在。

9 “Subscript out of range” 足碼超出範圍

陣列元素 ( array element ) 具有不正確足碼，或其足碼超

出陣列定義的維數( dimensions )。

- 10 “Redimensioned array” 重複設定陣列維數  
有兩個 DIM 陳述定義同一陣列。或陣列已被自然定為 10 個維數之後，出現一個定義此陣列的 DIM 陳述。
- 11 “Division by zero” “零”為分母  
在表式( expression )中出現以零為分母的情況，或冪方運算中發生零的負次方冪。發生在除法結果是帶分子正負號的無限大值，冪運算結果是正號無限大值，此後繼續執行程式。
- 12 “Illegal direct” 不合法的直接模式( direct mode)  
(第 2 章的操作模式)不可用於直接模式的陳述被當作直接模式命令輸入。
- 13 “Type mismatch” 不配合的資料型式  
以數值存於字串變數名，或反之。給予需要數值引數的函數一個字串引數，或反之。
- 14 “Out of string space” 字串空間已用完  
字串變數使 BASIC 超出了所剩下的可用記憶體。BASIC 會動態地( dynamic )配置字串空間，直到耗盡記憶體為止。
- 15 “String too long” 字串太長  
試圖建立長過 255 個字元的字串。
- 16 “String formula too complex” 字串格式太複雜  
字串表式( expression )太長或太複雜。此表式必須被拆成較短的表式。

164 APPLE 軟體卡

- 17 “Can't continue”無法繼續執行  
在下列狀況下試圖繼續執行程式。
- 1 已遭遇錯誤停止。
  - 2 執行斷點 ( break ) 時程式遭修改。
  - 3 程式不存在。
- 18 “Undefined user function”未定義的使用者函數  
USR 函數在被給定函數定義 ( DEF 陳敘 ) 之前被呼叫使用。
- 19 “NO RESUME”沒有 RESUME 陳述  
進入錯誤捕捉 ( trap ) 常式，但常式不具 RESUME 陳述。
- 20 “RESUME without error”沒有錯誤但遭遇 RESUME  
在進入錯誤捕捉常式之前遭遇 RESUME 陳述。
- 21 “Unprintable error”不可印出的錯誤  
沒有可用的錯誤訊息之錯誤狀況。發生在未定義錯誤碼的  
ERROR。
- 22 “Missing operand”缺乏運算元  
表示 ( expression ) 中運算子 ( operator ) 之後沒有運算  
元 ( operand ) 。
- 23 “Line buffer overflow”行緩衝區溢位  
試圖在一行上輸入太多字元。
- 26 “FOR without NEXT”FOR 之後沒有 NEXT

C-101 / D-10

- FOR 之後沒有與其配合的 NEXT。
- 29 “ WHILE without WEND ” WHILE 之後沒有 WEND  
WHILE 之後沒有與其配合的 WEND。
- 30 “ WEND without WHILE ” 沒有 WHILE 但遭遇 WEND  
WEND 之前沒有與其配合的 WHILE。
- 31 “ Reset error ” 重置錯誤  
Apple 鍵盤上的 RESET 鍵被按下。
- 32 “ Graphic statement not implemented ” 圖形陳述未  
被製作於內  
MBASIC 中未製作圖形陳述。此陳述只可用於 GBASIC。

### 磁碟錯誤

- 50 “ Field overflow ” 欄溢位  
FIELD 陳述企圖對隨機檔配置多於此檔已被指定資料錄長度的  
的拜數。
- 51 “ Internal error ” 內部錯誤  
在磁碟 BASIC-80 本身發生了內部不良運算。如果出現這個  
訊息，請通知 Microsoft。
- 52 “ Bad file number ” 不正確的檔案號碼  
陳述或命令中帶有未 OPEN 的檔案或檔案數超出起始時所定  
的檔案數範圍。
- 53 “ File not found ” 找不到檔案

LOAD, KILL, 或 OPEN 陳述中的指定檔案在現今磁碟上找不到。

54 “Bad file mode” 不正確的檔案模式

試圖使用 PUT, GET, 或 LOF 處理循序檔；LOAD 隨機檔；或以不是 I, O, R 的檔案模式 OPEN 檔案。

55 “File already open” 檔案已 OPEN

對已 OPEN 的檔案再做循序輸出模式的 OPEN；或 KILL 一個已 OPEN 的檔案。

57 “DISK I/O error” 磁碟 I/O 錯誤

操作磁碟 I/O 時，發生的 I/O 錯誤。屬於嚴重錯誤 (fatal error)，此時作業系統無法自錯誤本身回復正常。

58 “File already exists” 檔案已存在

在 NAME 陳述中指定的檔名相同於已存在磁碟中的一檔名。

61 “Disk full” 磁碟用滿

所有磁碟空間都已使用了。

62 “Input past end” 輸入超越結束點

在檔案資料已全被 INPUT 之後，仍執行 INPUT 陳述；或 INPUT 一個空檔案。避免發生這種錯誤，可利用 EOF 函數偵測檔案結束點。

63 “Bad record number” 不正確的資料錄號碼

在 PUT 或 GET 陳述中的資料錄號碼太大 (超出最大限度 32767) 或等於 0。

- 64 “Bad file name” 不正確的檔名  
在 LOAD, SAVE, KILL, 或 OPEN 中使用了不合法格式的檔名（例如，太多字元的檔名）。
- 66 “Direct statement in file” 檔中使用直接模式陳述  
當 LOADING 一個 ASCII 格式檔時，遭遇到直接陳述，LOAD 動作停止。
- 67 “Too many files” 檔案過多  
當所有 255 個索引表進入點（directory entry）用滿時，試圖用 SAVE 或 OPEN 建新檔。
- 68 “Disk read only” 磁碟僅可讀  
磁碟具有寫入保護（禁寫入），或未先用 RESET 就換掉磁碟。這個錯誤訊息通常會連續顯示兩次，不必因此過慮。
- 69 “Drive select error” 磁碟機選擇錯誤  
選擇了未存在的磁碟機。
- 70 “File read only” 檔案僅可讀  
試圖對已設為 “Read Only”（用 STAT 程式設）的檔案做寫入動作。





# 附 錄 F

## 數學函數

### 導出函數

不存在 BASIC -80 的其他函數可用下列方法計算得出

| 函數                         | 等效解法                                                         |
|----------------------------|--------------------------------------------------------------|
| SECANT                     | $SEC(X) = 1/COS(X)$                                          |
| COSECANT                   | $CSC(X) = 1/SIN(X)$                                          |
| COTANGENT                  | $COT(X) = 1/TAN(X)$                                          |
| INVERSE SINE               | $ARCSIN(X) = ATN(X/SQR(-X*X + 1))$                           |
| INVERSE COSINE             | $ARCCOS(X) = -ATN(X/SQR(-X*X + 1)) + 1.5708$                 |
| INVERSE SECANT             | $ARCSEC(X) = ATN(X/SQR(X*X - 1)) + SGN(SGN(X) - 1) * 1.5708$ |
| INVERSE COSECANT           | $ARCCSC(X) = ATN(X/SQR(X*X - 1)) + (SGN(X) - 1) * 1.5708$    |
| INVERSE COTANGENT          | $ARCCOT(X) = ATN(X) + 1.5708$                                |
| HYPERBOLIC SINE            | $SINH(X) = (EXP(X) - EXP(-X))/2$                             |
| HYPERBOLIC COSINE          | $COSH(X) = (EXP(X) + EXP(-X))/2$                             |
| HYPERBOLIC TANGENT         | $TANH(X) = (EXP(X) - EXP(-X)) / (EXP(X) + EXP(-X))$          |
| HYPERBOLIC SECANT          | $SECH(X) = 2 / (EXP(X) + EXP(-X))$                           |
| HYPERBOLIC COSECANT        | $CSCH(X) = 2 / (EXP(X) - EXP(-X))$                           |
| HYPERBOLIC COTANGENT       | $COTH(X) = (EXP(X) + EXP(-X)) / (EXP(X) - EXP(-X))$          |
| INVERSE HYPERBOLIC SINE    | $ARCSINH(X) = LOG(X + SQR(X*X + 1))$                         |
| INVERSE HYPERBOLIC COSINE  | $ARCCOSH(X) = LOG(X + SQR(X*X - 1))$                         |
| INVERSE HYPERBOLIC TANGENT | $ARCTANH(X) = LOG((1 + X)/(1 - X))/2$                        |
| INVERSE HYPERBOLIC SECANT  | $ARCSECH(X) = LOG((SQR(X*X + 1) + 1)/X)$                     |

INVERSE HYPERBOLIC  
COSECANT

$$\text{ARCCSCH}(X) = \text{LOG}((\text{SGN}(X) \cdot \text{SQR}(X^2 + 1) + 1) / X)$$

INVERSE HYPERBOLIC  
COTANGENT

$$\text{ARCCOTH}(X) = \text{LOG}((X + 1) / (X - 1)) / 2$$

# 附 錄 G

## ASCII 字 元 碼

| ASCII<br>碼 | 字元     | ASCII<br>碼 | 字元 | ASCII<br>碼 | 字元 |
|------------|--------|------------|----|------------|----|
| 000        | NUL    | 036        | \$ | 072        | H  |
| 001        | SOH    | 037        | %  | 073        | I  |
| 002        | STX    | 038        | &  | 074        | J  |
| 003        | ETX    | 039        | '  | 075        | K  |
| 004        | EOT    | 040        | (  | 076        | L  |
| 005        | ENQ    | 041        | )  | 077        | M  |
| 006        | ACK    | 042        | *  | 078        | N  |
| 007        | BEL    | 043        | +  | 079        | O  |
| 008        | BS     | 044        | ,  | 080        | P  |
| 009        | HT     | 045        | .  | 081        | Q  |
| 010        | LF     | 046        | :  | 082        | R  |
| 011        | VT     | 047        | /  | 083        | S  |
| 012        | FF     | 048        | 0  | 084        | T  |
| 013        | CR     | 049        | 1  | 085        | U  |
| 014        | SO     | 050        | 2  | 086        | V  |
| 015        | SI     | 051        | 3  | 087        | W  |
| 016        | DLE    | 052        | 4  | 088        | X  |
| 017        | DC1    | 053        | 5  | 089        | Y  |
| 018        | DC2    | 054        | 6  | 090        | Z  |
| 019        | DC3    | 055        | 7  | 091        | [  |
| 020        | DC4    | 056        | 8  | 092        | \  |
| 021        | NAK    | 057        | 9  | 093        | ]  |
| 022        | SYN    | 058        | :  | 094        | {  |
| 023        | ETB    | 059        | ;  | 095        | <  |
| 024        | CAN    | 060        | <  | 096        | '  |
| 025        | EM     | 061        | =  | 097        | a  |
| 026        | SUB    | 062        | >  | 098        | b  |
| 027        | ESCAPE | 063        | ?  | 099        | c  |
| 028        | FS     | 064        | @  | 100        | d  |
| 029        | GS     | 065        | A  | 101        | e  |
| 030        | RS     | 066        | B  | 102        | f  |

172 APPLE 軟體卡

|     |       |     |   |     |     |
|-----|-------|-----|---|-----|-----|
| 031 | US    | 067 | C | 103 | g   |
| 032 | SPACE | 068 | D | 104 | h   |
| 033 | !     | 069 | E | 105 | i   |
| 034 | "     | 070 | F | 106 | j   |
| 035 | #     | 071 | G | 107 | k   |
| 108 | l     | 115 | s | 122 | z   |
| 109 | m     | 116 | t | 123 |     |
| 110 | n     | 117 | u | 124 |     |
| 111 | o     | 118 | v | 125 |     |
| 112 | p     | 119 | w | 126 | ~   |
| 113 | q     | 120 | x | 127 | DEL |
| 114 | r     | 121 | y |     |     |

ASCII碼以十進制表示。

LF = Line Feed (饋行), FF = Form Feed (饋表), CR = Carriage Return (回車), DEL = Rubout (擦掉)。

# 第五部份：

## 軟體公用程式手冊

- 簡介

  - 格式記號

- 準備磁碟使其能夠做讀寫的工作：FORMAT

- 複製磁碟：COPY

  - 創建CP/M系統磁碟

- 用16段磁碟的CP/M存取13段磁碟的CP/M：RW13

- 建構CP/M成爲56 K系統：CPM56

- 將Apple DOS的檔案轉到CP/M上：APDOS

- 建構Apple CP/M作業系統的環境設施：CONFIGIO

- 1 爲外來終端機建構CP/M
- 2 重定義鍵盤字元
- 3 載入使用者的I/O軟體推動程式
- 4 讀/寫I/O架構

從別的計算機把CP/M檔案傳送過來：DOWNLOAD與  
UPLOAD

## 簡介

在 SoftCard 套件中含有許多公用程式，它們協助使用者在 Apple II 計算機上用 CP/M 完成一些特定的工作。

**FORMAT** 用來格式化空磁碟，使其能用在 SoftCard 系統。

**COPY** 用來複製磁碟。

**CONFIGIO** 為各種不同的硬體及軟體系統組合設定 I/O 架構。

**RW13** 由 16 段磁碟的 CP/M 存取 13 段磁碟的 CP/M。

**CPM56** 為 56 K 語言板系統設定 CP/M 架構。

**APDOS** 將 Apple DOS 的主文檔 (text file) 及二進制檔 (binary file) 傳到 CP/M 上。

**UPLOAD/DOWNLOAD** 將檔案由標準的 CP/M 機器傳到 Apple CP/M 系統上。

使用這些程式的方法在下面說明之。

## 格式記號

此後所提的陳述或命令格式都遵守下列規則：

- 1  $d_1, d_2, d_3$  等等表示你所指定的磁碟機。合用的磁碟機名稱可以是 A:, B:, C:, D:, E:, 和 F:。
- 2  $n$  是 0 到 9 的一個整數，計算機依據此正使用的軟體而將此

數顯示出來。

- 3 在中括號 ( [    ] ) 內的項目是個選用項目。

## 準備磁碟使其能夠做讀寫的工作：FORMAT

命令格式：

FORMAT 或 FORMAT:d

目的：

FORMAT 程式讓你準備空白磁碟使其能夠做讀寫工作。在 Apple SoftCard 系統上使用空白磁碟之前，必須先將此磁碟格式化。

使用規則：

欲格式化空白磁碟，首先把具有 CP / M 及 FORMAT 公用程式的磁碟插入磁碟機。然後叫出 CP / M ( 參考“組立及操作手冊” ) 。當看到 CP / M 的催促信號 A > 出現時，就可以開始使用了。

FORMAT 程式有兩種啓始 ( initialized ) 方法，根據使用者打算格式化一個或多個磁碟而決定採用那一種方法。

採用第一種方法：只格式化一個磁碟時。

- 1 首先鍵入：

**FORMAT d:**



並按下 RETURN 鍵。若系統具有兩個或以上的磁碟機，在按下 RETURN 鍵之前必須確定好已經將空白磁碟插入所指定的磁碟機上。如果你只有一個磁碟機且指定為 A：，此時仍放着有 FORMAT 程式的磁碟。

- 2 螢幕上會顯示出程式的版權聲明：

```
APPLE II CP/M
nn SECTOR DISK FORMATTER
(C) 1980 MICROSOFT
```

INSERT DISK TO BE FORMATTED IN DRIVE d:

照這個指示做，之後，按 RETURN 鍵開始執行格式化磁碟的工作。

- 3 如果是在多磁碟機系統上，格式化工作完成後，計算機會自動返回 CP / M 去。如果僅有一部磁碟機，將顯示如下：

```
FORMAT COMPLETED
INSERT CP/M SYSTEM DISK IN DRIVE A: AND PRESS
RETURN
```

照指示插入含有 CP / M 的磁碟，然後按 RETURN。你現在已經有了一個新的已格式化磁碟，它已經能接受 CP / M 或用來儲存程式或資料了。

注意：

這個新的格式化磁碟內部並未含有 CP / M 作業系統，所以沒辦法 BOOT 它。必須使用 COPY 公用程式（參考 COPY 用法）。將 CP / M 系統鍵入這新的已格式化磁碟，才能 BOOT 它。

採用第二種方法：打算格式化數個磁碟時。

1. 鍵入：

**FORMAT**

按下 RETURN，計算機會顯示：

**APPLE II CP/M  
nn-SECTOR DISK FORMATTER  
(C) MICROSOFT 1980**

**FORMAT DISK IN WHICH DISK DRIVE?**

2. 指定所在的磁碟機。鍵入：

**d:**

按下 RETURN 鍵。如果系統具有兩個或以上的磁碟機，在按下 RETURN 鍵之前必須先確定已經將空白磁碟插入所指定的磁碟機上。如果只有一個磁碟機，指定為 A：，此時應放着含有 FORMAT 程式的磁碟。如果在按 RETURN 之前沒有先指定磁碟機，計算機會返回 CP / M 中。

3. 在多磁碟系統上，此時計算機會格式化所指定磁碟機上的磁碟。如果是在單磁碟機系統，計算機顯示下列訊息：

**INSERT DISK TO BE FORMATTED IN DRIVE A:  
PRESS RETURN TO BEGIN**

照著指示做，插入欲格式化的磁碟，按下 RETURN 鍵，開始格式化工作。

4. 格式化過程完成後，會顯示出：

```
FORMAT COMPLETE
FORMAT DISK IN WHICH DRIVE?
```

你可以依照這種方法，無限制地繼續格式化其他空白磁碟，只要將空白磁碟正確地放入所指定的磁碟機。全部做完之後，以 RETURN 回答上項詢問，計算機就返回 CP / M 中。若是單磁碟機系統，在按 RETURN 之前必須先把含有 CP / M 的磁碟重新插入磁碟機。

注意：

新格式化磁碟的內部並未含有 CP / M 作業系統，所以無法 BOOT 它。要把 CP / M 系統建入這個已格式化磁碟，必須使用 COPY 公用程式（參考 COPY 用法）。

註：假如你試圖格式化一個原來已經含有資料的磁碟，計算機會顯示下列訊息：

```
DISK IN DRIVE d: WILL BE ERASED.
CONTINUE(Y/N)?
```

如果回答 Y，計算機會把磁碟完全洗掉，重新格式化它。如果回答 N，計算機會再詢問如下：

```
FORMAT DISK IN WHICH DRIVE?
```

以便讓你插入另一片磁碟或指定別的磁碟機。如果這時候按下 RETURN 鍵，FORMAT 程式便終止執行，計算機返回 CP / M 中。

錯誤訊息：

如果 FORMAT 沒有執行成功計算機會顯示以下三個錯誤訊息之

一。

**DISK WRITE PROTECTED** 你所要格式化的磁碟上具有寫入保護（禁寫入）封籤。將這個寫入保護的封籤拿掉，再重複 FORMAT 過程。

**DISK I/O ERROR** 因某種原因而使得計算機無法格式化磁碟。檢查磁碟機中是否確有磁碟，或磁碟機的門是否未關好。

**COMMAND ERROR** 命令不被接受。重鍵入命令，確定命令格式正確無誤。

錯誤信號之後，計算機會再顯示出“FORMAT DISK IN WHICH DRIVE?”

### 複製磁碟：COPY

命令格式：

COPY d<sub>1</sub> = d<sub>2</sub>;

選用“/S”只複製 CP/M（軌道 0-2）。

目的：

這個複製磁碟公用程式可以複製 Apple 的 CP/M 磁碟。也可以為新格式化的磁碟建入 CP/M 系統。

**使用規則：**

將磁碟內容複製到一空白的已格式化磁碟之前，必須先把含有 COPY 公用程式和 CP/M（兩者都存在 SoftCard 磁碟上）的磁碟插入磁碟機，並叫出 CP/M（參考“組立及操作手冊”）。當 CP/M 的催促信號 A> 出現後，就如下操作。

## 1. 鍵入：

COPY d<sub>1</sub>:=d<sub>2</sub>:

d<sub>1</sub>: 是複製的目標磁碟機，d<sub>2</sub>: 是複製的來源磁碟機。例如 A:=B: 表示將 B: 的內容複製到 A:。如果只一個磁碟機就鍵入 A:=A:。

如果只打入 COPY，計算機會顯示出催促信號（\*）等待你鍵入命令行（d<sub>1</sub>:=d<sub>2</sub>:）之後再繼續執行。

鍵入命令之後，計算機顯示下列訊息：

APPLE II CP/M  
xx-SECTOR DISK DUPLICATION PROGRAM  
(C) 1980 MICROSOFT

（如果只有一個磁碟機，以下直接看第三步驟。

## 2 在多磁碟機系統上，計算機顯示下列訊息：

INSERT MASTER DISK IN d<sub>1</sub>:  
INSERT SLAVE DISK IN d<sub>2</sub>:  
PRESS RETURN TO BEGIN

照此指示，把來源磁碟（source, master disk）插入磁碟機 d<sub>1</sub>:，把目標磁碟（destination, slave disk）插入磁碟機 d<sub>2</sub>:，然後按下 RETURN 鍵開始執行複製工作。（繼續第四步驟）

3. 在單磁碟機系統上，計算機會顯示以下訊息：

```
INSERT MASTER DISK  
PRESS RETURN TO CONTINUE
```

把 SoftCard 磁碟拿掉，換上要複製的來源磁碟。按下 RETURN 鍵。一會兒之後，計算機會顯示：

```
INSERT SLAVE DISK   PRESS RETURN
```

把來源磁碟拿掉，換上空白已格式化的磁碟，然後按 RETURN 鍵。一會兒之後上述訊息重複出現。重複第三步驟的動作直到 COPY COMPLETE 訊息出現為止。（繼續第四步驟）

4. 一件複製工作完成之後，計算機將顯示以下訊息：

```
COPY COMPLETE  
DO YOU WISH TO MAKE ANOTHER COPY? (Y/N)  
PRESS RETURN
```

按下 Y 做下一件複製。在按 RETURN 鍵之前先將另一已格式化磁碟插入磁碟機 d<sub>1</sub>。

按下 N 則返回 CP / M 的命令中。

### 創建 CP/M 系統磁碟：

在“COPY”中選用“/S”可以只將 CP/M 作業系統由一個磁碟複製到另一磁碟。兩磁碟的其他檔案不受影響。必須把 CP/M 複製入欲在 SoftCard 系統上使用的磁碟（這個磁碟被複製之前須先用 FORMAT 格式化）。

```
COPY d1:=d2:/S
```

$d_2$  : 是目標磁碟機 ,  $d_1$  : 是來源磁碟機。“ / S ” 告訴計算機只要複製 CP / M , 其他過程和複製磁碟的一樣。

**注意 :**

除非用了“ / S ” , 複製磁碟時 , 所有在目標磁碟內的檔案都會被洗掉。

目標磁碟在複製前必須是已格式化的。

**錯誤訊息 :**

如果 COPY 沒有執行成功 , 計算機會顯示下列三個錯誤訊息之一。

**DISK WRITE PROTECTED** 所要複製入的目標磁碟上有寫入保護封籤。將這個寫入保護封籤拿掉 , 再重複 COPY 過程。

**DISK I / O ERROR** 因某種原因 , 計算機無法對磁碟做存取的工作。檢查磁碟機中是否確有磁碟 , 磁碟機的門是否關好。

**COMMAND ERROR** 命令不被接受。重新鍵入命令 , 並確定命令格式正確無誤。

## 用16段磁碟的CP/M 存取13段磁碟的CP/M：RW13

### 命令格式：

RW13 d<sub>i</sub>;

與

RW13 X

(將磁碟機轉換回 16 段落式)

### 目的：

16 段磁碟的系統可以利用 RW13 來讀寫 13 段磁碟。RW13 執行時，13 段磁碟的檔案可被 16 段磁碟的 CP/M 存取。當與 PIP 一起使用時，可以將 13 段磁碟的檔案傳送到 16 段的磁碟上。

RW13 X 命令用來將磁碟機轉換回 16 段落式。RW13 程式只存在 16 段的 SoftCard 磁碟上，使用它時需用兩個或以上的磁碟機，其中磁碟機 A：，不能轉成 13 段落式操作。

### 使用規則：

插入含有 CP/M 和 RW13 程式的磁碟（兩者都在 SoftCard 磁碟上）到磁碟機上，並叫出 CP/M（參考“組立及操作手冊”）。當出現 CP/M 催促信號 A> 時，就如下操作。

#### 1. 鍵入：

RW13 d<sub>i</sub>;

d<sub>i</sub> 是 B：至 F：的磁碟機。可以指定除了 A：之外的任一磁碟機，然後按下 RETURN 鍵。



2 計算機會顯示以下訊息：

APPLE II CP/M  
13-SECTOR DISK CONVERSION  
(C) 1980 MICROSOFT

DRIVE D<sub>1</sub>: CONVERTED TO 13 SECTOR OPERATION.

3. 任何 13 段磁碟插入此“已轉換”的磁碟機後就能被任一個 CP/M 程式讀取或寫入。這時候，你可利用 PIP（參考“組立及操作手冊”）把在“已轉換”磁碟機上的 13 段磁碟檔案傳送到在“未轉換”磁碟機上的 16 段磁碟。或者，使用其他操作 13 段磁碟系統的 CP/M 軟體。

註：不可用 COPY 程式複製檔案。

4. 使用完畢後，應把磁碟機轉換回 16 段式，鍵入：

RW13 X

按下 RETURN 鍵。磁碟機就轉回 16 段操作方式。

註：RW13 佔用 4 K 拜記憶體，所以使用它的時候，其他程式可利用的記憶體少掉 4 K 拜。

**建構 CP/M 成爲 56K 系統：CPM56**

**命令格式：**

CPM56 d:

**目的：**

更新 CP/M 以便使用 56K 語言卡系統。假如你有個 56K 的

系統，在使用 CP/M 之前應先做這項轉換，才能用到額外的 8K 記憶體。在 48K 系統上，不需要使用這個公用程式。

### 使用規則：

將含有 CP/M 及 CPM56 程式的磁碟（兩者都在 SoftCard 磁碟上）插入磁碟機，並 BOOT CP/M（參考“組立及操作手冊”。當 CP/M 催促信號 A > 出現時，就如下操作。

#### 1. 鍵入：

CPM56 d:

把含有 CP/M 複製品的磁碟插入指定磁碟機（參考 COPY 公用程式的複製 CP/M 方法）。按下 RETURN 鍵。

2. 按下 RETURN 鍵之後，計算機會自動將在所指定磁碟機的 CP/M 複製品建構成 56K 記憶體系統。轉換完成後，計算機會顯示下列訊息：

DISK IN DRIVE d: HAS BEEN UPDATED TO 56K.

此時，你已擁有了一個使用於 56K 記憶體系統的 CP/M 磁碟。

註：如果你已經利用 CONFIGIO 公用程式定義特定的字元，這些字元在 CPM56 更新 CP/M 後仍保存著未改變掉。

### 錯誤訊息：

如果 CPM56 程式沒有執行成功，下列三個訊息之一會顯示出來

- DISK I/O ERROR** 由於某種原因使得磁碟機無法存取磁碟。檢查磁碟機內是否有磁碟，磁碟機的門是否關好。
- DISK WRITE PROTECTED** 磁碟上有“寫入保護”封籤。拿掉寫入保護封籤並重複 CPM56 程序。
- COMMAND ERROR** 命令不被接受。重新鍵入命令，並確定命令格式正確無誤。

## 將Apple DOS的檔案轉到CP/M上：APDOS

### 命令格式：

APDOS d<sub>1</sub>:filename.typ=d<sub>2</sub>:filename

或

APDOS d<sub>2</sub>:

### 目的：

將 Apple DOS 的主文檔及二進制檔轉送到 CP/M 上。

APDOS 本身無法直接讀取 BASIC 檔，也不能寫入 Applesoft 磁檔。如果你想把 DOS3.2 的檔案轉到 16 段 CP/M 磁碟上，必須先利用 RW13 把磁碟機轉成 13 段式操作，再使用 APDOS。

### 使用規則：

將含有 CP/M 及 APDOS 程式的磁碟插入磁碟機（兩者都存在 SoftCard 磁碟上），並叫出 CP/M（參考“組立及操作手冊”）。

CP/M 的催促信號 A > 出現後，如下操作。

## 1 鍵入

**APDOS**

並按 RETURN 鍵，計算機顯示出：

**APPLE II CP/M  
APPLE DOS CP/M FILE TRANSFER  
(C) 1980 MICROSOFT**

並打出一催促的冒號“:”。如果你鍵入 CAT d: 在磁碟機 d: 的 Apple DOS 磁碟檔案目錄會顯示出來。

## 2 鍵入

**[d<sub>1</sub>:] Fname.typ = [d<sub>2</sub>:] Filename**

把 Apple DOS 檔“Filename”(在磁碟機 d<sub>2</sub>:) 轉成 CP/M 檔“Fname, type”(在磁碟機 d<sub>1</sub>)。如果沒有指定磁碟機，則自然值 (default) 定 d<sub>1</sub>: 為 A:，d<sub>2</sub>: 為 B:。

## 3 之後繼續做由 Apple DOS 複製檔案到 CP/M 磁碟的工作，鍵入：

**Fname.typ = Filename**

此時，d<sub>1</sub>: 及 d<sub>2</sub>: 已定成剛才的值。如果要改它們可照原來的格式鍵入 APDOS 命令。

APDOS 轉換 Apple DOS 主文檔時，將它們的每個字元 ( character ) 高位比次 ( bits ) 清除掉。而轉換 Apple DOS 二進位檔時，保留檔案最前四拜 ( 位址及長度資料 )，實際資料轉換自第五拜開始。參考 Apple DOS 3.2 或 3.3 手冊所述的主文檔及二進位檔格式。

使用下列程序將 Apple DOS 的 Applesoft 或 Integer BASIC 程式轉換至 CP/M 檔。這個程序首先將 Applesoft 或 Integer BASIC 程式轉成主文檔，再用 APDOS 轉成 CP/M 檔。

1. BOOT Apple DOS 3.2 或 3.3 磁檔，其中含有你所要轉送的程式。BOOT 之後將程式載入 ( LOAD ) 記憶體。

2. 鍵入下列程式，作為程式的第一行：

```
0PRINT "ctrl-D OPEN APPLEPROG":PRINT "ctrl-D WRITE
APPLEPROG":POKE 33,33:LIST:PRINT "ctrl-D CLOSE":END
```

( ctrl-D 看不見，但應鍵入 )

3. RUN 這個程式。程式做完後，你就得到一個存在 Apple DOS 磁碟的主文檔，稱為 "APPLEPROG"，這個主文檔就是原程式的主文複製品。
4. BOOT CP/M 磁碟。
5. 鍵入 APDOS。
6. 將 Apple DOS 磁碟插入磁碟機 B：( 在單磁碟機系統則為 A：)。
7. 如果是在多磁碟機系統，鍵入 APPLE.BAS=APPLEPROG

並按下 RETURN 鍵。如果是在單磁碟機系統，鍵入 APPLE.BAS=APPLEPROG，並按下 RETURN 鍵。如果是在單磁碟機系統，鍵入 APPLE.BAS=A:APPLEPROG，並按下 RETURN 鍵。

- 8 鍵入 ctrl-C 跳出 APDOS。
- 9 鍵入 MBASIC 或 GBASIC 進入 BASIC 中。
- 10 鍵入 LOAD "APPLE"，並按 RETURN 鍵。
- 11 刪除第 0 行（第二步驟時所鍵入的）。
- 12 你現在已經把 Applesoft 或 Integer BASIC 程式的複製品轉成 Apple CP/M，它們開始時可能無法 RUN。你可能需要對程式作些編修，將 POKE, PEEK, CALL，及其他磁碟檔的陳述（statement）改成等效的 Microsoft BASIC 陳述。多數的 POKE, PEEK, CALL 都不能用在 Microsoft BASIC。應該將它們取代掉。參考第四部分之 Microsoft BASIC 關於轉換程式成爲 Microsoft BASIC 的詳細說明。

### 錯誤訊息：

假如 APDOS 轉換工作不成功，下列錯誤訊息之一會顯示出來。

DISK I/O ERROR 由於某種原因使得磁碟機無法存取磁碟。檢查磁碟機中確已插入磁碟，並關好磁碟機的門。

|                         |                                    |
|-------------------------|------------------------------------|
| DISK WRITE<br>PROTECTED | 磁碟上有“寫入保護”封籤。拿掉寫入保護封籤並重複 APDOS 程序。 |
| COMMAND ERROR           | 命令無法被接受。重新鍵入命令，並確定命令格式正確無誤。        |

## 建構 ( Configure ) Apple CP/M 作業系統的環境設施：

### CONFIGIO

#### 目的：

CONFIGIO 公用程式用來建構 Apple CP/M 系統之設施以配合使用者的特定系統架構 ( configuration ) 。

#### 使用規則：

將含有 MBASIC ( 或 GBASIC ) 及 CONFIGIO 的 CP/M 系統磁碟插入磁碟機。叫出 CP/M ( 參考“組立及操作手冊” ) 。當出現 CP/M 催促信號 A > ; 鍵入：

MBASIC CONFIGIO

並按下 RETURN 鍵。

假如你是使用標準的 Apple ( 即，沒有外來終端機 ) ，計算機會詢問

CAN YOUR APPLE DISPLAY LOWER CASE (Y/N)?

假如你的 Apple 具有適當的硬體可以把小寫主文 ( lower case text ) 直接顯示在 Apple 螢幕上，就回答 Y 。否則回答 N 。回答 N

，使得小寫字元先被轉成大寫才打在 Apple 螢幕上（這個功能可以藉下述的“第四項選擇”永久地設定之）。

之後，計算機會顯示出下列名單：

```
++ I/O CONFIGURATION PROGRAM ++  
1. CONFIGURE CP/M FOR EXTERNAL TERMINAL  
2. REDEFINE KEYBOARD CHARACTERS  
3. LOAD USER I/O DRIVER SOFTWARE  
4. READ/WRITE I/O CONFIGURATION BLOCK  
Q. QUIT PROGRAM  
SELECT -
```

選擇 1，2，3，4 或 Q 執行下述的功能。

1. 為外來的終端機建構 CP/M 可讓你訂定字元序列，此字元序列是你的特定軟體及硬體在執行特定的螢幕功能時所必須使用的。一旦適當地設定好這些序列，系統就會在你的終端機和軟體間自動轉換這些字元序列。
2. 重定義鍵盤字元 讓你可以重定義鍵盤上任意特定鍵的 ASCII 值。利用這個項目，你可以使一鍵（例如“3”）產出不同的字元（例如“!”的碼）。也可以利用“Ctrl-V”產出“`<`”字元，這個項目特別有用是在它能產出原來



Apple 鍵盤所沒有的字元。

3. 載入使用者的 I/O 軟體推動程式 可載入你的 I/O 軟體推動程式並將它併入磁碟的“ I/O 架構框塊” ( I/O Configuration Block ) 以便可以使用非標準的 Apple 週邊裝置。
4. 讀 / 寫 I / O 架構框塊, 可讓你讀或寫磁碟的 I / O 架構塊。CONFIGIO 的 1 至 3 項中所做的更改可藉著把 I/O 架構框塊寫入磁碟而永久設定之。

Q. 中止程式 跳出 CONFIGIO, 返回 BASIC。

關於這些功能, 在軟體詳述 ( 上册第二部分 ) 第二章中有更詳細說明。以下是四個功能的用法:

## 1. 為外來終端機建構 CP/M

### 簡介

大多數的視訊終端機 ( 包括 Apple 的 24 × 40 螢幕 ) 都會提供一群特殊的螢幕功能, 如清除螢幕 ( clear screen ), 明顯化主文 ( Highlight Text ), 及定址游標 ( Address Cursor )。做法是將一串特殊字元送至終端機令終端機執行特殊功能。而多數的應用軟體 ( 例如以螢幕為基礎的文字處理程式 ) 在這方面却只能配合小部分 ( 懂得這些軟體送出的字串序列 ) 的終端機。

Apple 的 CP/M 提供了轉換表可用來處理 ( 你的硬體及軟體的 ) 螢幕功能的適當字元序列 ( character sequence ) 要求。以下就是設定 Apple CP/M 來配合你的特定系統架構的方法。

註：參考“軟體及硬體詳述”的第二章，其中具有較多關於終端機架構的資料。

從主要名單上選出數目“1”之後，計算機會顯示出另一張名單

+ TERMINAL SCREEN FUNCTION DEFINITION +

| FUNCTION       | SOFTWARE | HARDWARE |
|----------------|----------|----------|
| CLEAR SCREEN   | ESC *    | FF       |
| CLR TO EOS     | ESC Y    | VT       |
| CLR TO EOL     | ESC T    | GS       |
| LO-LITE TEXT   | ESC )    | SO       |
| HI-LITE TEXT   | ESC (    | SI       |
| HOME CURSOR    | RS       | EM       |
| ADDRESS CURSOR | ESC -    | RS       |
| XY COORD OFFST | 32       | 32       |
| XY XMIT ORDER  | YX       | XY       |
| CURSOR UP      | VT       | US       |
| CURSOR FORWARD | FF       | FS       |

1. SOROC IQ 120/IQ 140
  2. HAZELTINE 1500/1510
  3. DATAMEDIA
  4. OTHER
- Q. QUIT

SELECT -

這是硬體和軟體的螢幕功能表 (Hardware and Software Screen Function Tables)。

註：在為外來終端機建構 CP / M 時，應把第三溝槽上的界面板拿掉並使用標準的 Apple 視訊顯示器。在建構的過程完畢後，再把界面板重插入。

硬體和軟體螢幕功能表的內容都是以標準 ASCII 字元名稱顯示出。表中的 NUL 進入點表示此功能不能使用。

在這裏也可以為某一普通的終端機建表，只要如以下方法鍵入適當數目選擇它即可。

1. **SOROC IQ 120 / IQ 140** 鍵入“1”，為 SOROC IQ 120 / IQ 140 視訊終端機建構軟體或硬體螢幕功能表。鍵入“1”後，計算機會詢問你要重建構那一個表（硬體或軟體的？）。  
因為螢幕功能表已初始設定成使用 SOROC IQ 120 / IQ 140 視訊終端機的模式，除非為了配合軟體欲重定義軟體螢幕功能表，你並不需要變更這兩表就可以使用這種終端機。  
註：SOROC終端機開機時，其自然值定為“顯著”（Hi-lite）主文模式。BOOT 系統時，CP/M 會送出“Lo-lite”字元序列。
  
2. **Hazeltine 1500/1510** 鍵入2，為Hazeltine 1500 / 1510 視訊終端機建構硬體螢幕功能表。  
選擇“2”，應只設立硬體螢幕功能表。因為CP/M以非標準方式管理Hazeltine 游標定址功能（未使用X-Y座標偏移量），所以此處不鼓勵在軟體表使用Hazeltine 螢幕功能序列（sequences）。最好為Hazeltine 設立硬體表，而為其他通用的終端機（如SOROC IQ 120 / 140）設立軟體表。
  
3. **Datamedia** 鍵入3，為Datamedia 型的終端機建構硬體螢幕功能表。此型是個配合24 × 80視訊終端機板（如Vidioterm或Sup-R-Term）的架構。  
選擇“3”，也只能設立硬體螢幕功能表，因為CP/M軟體通常不提供Datamedia 終端機序列（sequences）。你應該為24 × 80 視訊板設硬體表，而為其他通用終端機（

如 SOROC IQ 120 / 140' ) 設立軟體表。Hi-lite 主文及 Lo-lite 主文 ( INVERSE 及 NORMAL ) ( 反向及正常情形 ) 功能並非每個 Datamedia 型終端機都具備，因此這些功能表的進入點任意定值，使這些進入點為非零值。

4. 其他型終端機 如果為其他未提到的終端機設立軟體或硬體表，就鍵入“4”，鍵入“4”後，計算機會顯示出另一組名單，你可改變其中部分或全部的螢幕功能字元序列。

++ SCREEN FUNCTION DEFINITION ++

- 1 - LEAD-IN CHARACTER
- 2 - CLEAR SCREEN
- 3 - CLR TO EOS
- 4 - CLR TO EOL
- 5 - LO-LITE TEXT
- 6 - HI-LITE TEXT
- 7 - HOME CURSOR
- 8 - ADDRESS CURSOR
- 9 - CURSOR UP
- 10 - CURSOR FORWARD
- Q - QUIT

SELECT -

這時候就可以將上列“終端機螢幕功能定義”表的值改掉。

註：你可以在終端機手冊上找到適用的螢幕功能命令字元。有特殊的程式用來協助找出這些字碼（如：文字處理程式），必要時可參考這特殊程式的說明手冊。

選擇 1 到 10 的數目去定義下列函數的字元序列 ( character sequences ) :

| 數目 | 標題           | 說明                                                 |
|----|--------------|----------------------------------------------------|
| 1  | Lead-in char | 定義出“領入字元”——在螢幕功能命令字元之前的字元（常為ESC）。特殊的螢幕功能不一定需要領入字元。 |

## 196 APPLE 軟體卡

- |    |                  |                                                      |
|----|------------------|------------------------------------------------------|
| 2  | Clear screen     | 清除螢幕並將游標設在螢幕的左上角。                                    |
| 3  | Clear to EOS     | 把自游標至螢幕終點的部份清除掉。                                     |
| 4  | Clear to EOL     | 把自游標至本行終點的部分清除掉。                                     |
| 5  | Lo-lite text     | 設定以正常的視訊模式顯示主文。                                      |
| 6  | Hi-lite text     | 設定反向或倍強度（依所使用的終端機而定）視訊模式顯示主文。                        |
| 7  | Home cursor      | 把游標設到螢幕左上角，但不清除螢幕。                                   |
| 8  | Address cursor   | 用後面跟著的兩個字元令終端機指到某特定游標位址。                             |
|    | XY Coord. Offset | 如第“8”項的部分定義。當這個“XY 座標偏移量”送到終端機，它們就被加到X和Y的座標上（通常為32）。 |
|    | XY Xmit Order    | 如第“8”項的部分定義。建立座標送出的順序 - XY 或 YX（通常是YX）。              |
| 9  | Cursor Up        | 把游標向上移一行。                                            |
| 10 | Cursor Forward   | 使游標在行上向前移，不刪除所經過的字元'。                                |

要為這些功能設定適當的字元序列時，只需把對應的數目鍵入，並按下 RETURN 鍵。

選擇數目“1”，設定一個螢幕功能的“領入字元”(Lead-in character)。

計算機會顯示：

LEAD-IN CHAR:

輸入所要的領入字元。輸入的字元必須符合下列中任一種格式：

2或3個字元的ASCII名稱。

CTRL -ch ch 是任意鍵盤字元。

ch ch 是任意鍵盤字元。

LC -ch LC- 表示其後的字元是小寫。如果你的鍵盤沒有小寫字，可以用其它取代。

ASCII 16 進位碼(前面要加&H) 如果字元無法鍵入，可用等效ASCII 16 進位碼代表。(參考“軟體詳述”中的ASCII碼圖表)

在鍵入所要的領入字元之後，計算機會詢問：

SOFTWARE OR HARDWARE (S/H)?

根據領入字元用在軟體螢幕功能表或硬體表而定，按下S或H。

定義其他的螢幕功能，只要鍵入那個功能所對應的數目，計算機就會

打出催促信號要你鍵入那個特定功能的命令字元。例如鍵入“2”，計算機會打出：

CLEAR SCREEN

鍵入使用這個功能的字元（用前述的五種格式之一），然後按下 RETURN 鍵。不要用已經使用的領入字元。如果終端機無此功能，就輸入 NUL ( ASCII 00 )。

在輸入字元後，計算機會詢問：

REQUIRE LEAD-IN (Y/N)?

如果要讓此功能的命令字元前帶“領入字元”，就鍵入 Y，否則鍵入 N。之後計算機再問：

SOFTWARE OR HARDWARE (S/H)?

鍵入 S 更改軟體螢幕功能表。鍵入 H 更改硬體表。

然後，計算機返回去顯示“SCREEN FUNCTION DEFINITION”（螢幕功能定義）名單，等待你再選另一數目或鍵入 Q 中止執行。藉這方法，你可以任意對表做多項更改。（其中定址游標“8”的更改過程稍有不同，請看以下說明）在這名單上鍵入 Q，會重顯示“螢幕功能表”。

如果選擇“8”，定址游標。開始程序與上述相同，直到出現

Require Lead-in (Y/N)?

回答 Y 或 N 之後，計算機會打出：

XY COORD OFFST

輸入一個表示空格數的數目，這數目在 X / Y 座標送出前加在 X / Y 上。最後還會被詢及：

XY XMIT ORDER

如果 Y 先傳送，輸入 YX，如果 X 先傳送則輸入 XY。然後計算機才提出以下詢問。

SOFTWARE OR HARDWARE (S/H)?

回答之後，繼續做其他的功能。

### 關於 CP/M 終端機架構的注意事項

限制：螢幕功能表只能以兩種字元序列使用：單一控制字元，或任何字元前面加一領入字元。較長的字元序列可以被製做出以配合特殊用途 I / O 推動程式。參考“軟體和硬體詳述”（第二部分）。

為了讓螢幕功能表的變更永久化，必須選用“ I / O 架構程式”（ I / O Configuration Program ）名單中的第四項目。若不利用這項目把 I / O 架構框塊（ BLOCK ）寫入 CP/M 磁碟，在重 BOOT 系統後，表中的變更會自然還原。

只要表中每個進入點都不是“零”值，不論在表中插入什麼值，它們都可以配合正常的 Apple 24 × 40 螢幕正常工作。

軟體螢幕功能表的內容必須符合實際的軟體在執行螢幕功能時送出的字元序列，而硬體螢幕功能表也必須符合實際硬體裝置的字元序列要求。

只要全部九個功能的硬體和軟體螢幕功能表進入點都沒有被設為“零”值，Microsoft BASIC 可以與任何終端機配合使用。

以設定軟體螢幕功能表來模擬 SOROC IQ 120 / 140 型終端



機是個好辦法，CP/M 軟體大部分採用這種架構。

## 2. 重定義鍵盤字元

鍵盤字元重定義可以使得原在鍵盤上沒有的字元變得可以使用。

如果選用數目“2”，計算機就會顯示：

```

+ + KEYBOARD CHARACTER DEFINITION + +
Ctrl-K ->  [
Ctrl-@ ->  RUB
Ctrl-U ->  Ctrl-I
Ctrl-B ->  \
ADD/DELETE/QUIT (A/D/Q)-
    
```

表中有三個字元被重定義：Ctrl-K, Ctrl-@, 和 Ctrl-B。這些字元已經被重定義成常用的三個字元（Apple 鍵盤中沒有的）

“[”，“RUBOUT”，及“\”。

你可以選用 A, D, 或 Q 去再多定義一些字元，刪除一些字元或返回顯示出主要名單。

如果鍵入“A”，計算機會顯示：

CHAR:

輸入要被重定義的原字元。字元的輸入格式必須符合下列幾種格式之

ch            ch 代表任意字元

2 或 3 個字元的 ASCII 名稱 (如: NUL)

Ctrl-ch ch 代表任意字元

LC-ch 當小寫字元無法直接鍵入時，用 LC- 輸入小寫字元。

ASCII 的 16 進位碼 (前面要加&H)

如果字元無法直接鍵入，可以用等效的 ASCII 16 進位碼代表。(參考“軟體及硬體詳述”的 ASCII 碼圖表)

舉例說，如果你要防止不經意地鍵入 Ctrl-C 而中斷掉 BASIC 程式，可以把 Ctrl-C 重定義為 NUL (ASCII 00)。在出現催促信號 CHAR: 後，首先鍵入：

CTRL-C

如果輸入字元被接受，計算機會打出箭號催促你輸入新定義的字元。

CTRL-C -> NUL

上式已經輸入 NUL。

如果輸入字元不被接受，計算機就取消它，再等待正確的字元輸入。

當按下 RETURN 鍵後，螢幕會顯示出重定義鍵盤字元的列表，新的重定義已顯示於其中。此後，每當按下 Ctrl-C，實際上會送入 NUL 字元。(你現在就可以試一下，按 Ctrl-C 看看會不會跳出 CONFIGIO 程式)

如果在選擇數目“2”之後鍵入“D”，可以刪除重定義的鍵盤字元。例如要取消剛才對 Ctrl-C 的重定義，鍵入 D。計算機會打出催促信號 CHAR:，要求你輸入所欲取消的鍵盤字元重定義。輸入 Ctrl-C 並按下 RETURN 後，重定義鍵盤字元列表再度顯示出，其

中已經沒有 Ctrl-C → NUL 了。

如果鍵入 Q，就會返回到顯示主名單的狀態

### 關於鍵盤字元重定義的注意事項

如果一個鍵盤字元重定義在你的鍵盤用不上，最好把它刪除掉。例如你的鍵盤已有 RUBOUT 鍵，就應把 Ctrl-@ 那個重定義刪除掉。

把 Ctrl-C 重定義成 NUL 是個有用的方法，它避免了無意中用 Ctrl-C 把 BASIC 程式中斷掉，但在 CP/M 命令模式中可能會出點問題。因為 CP/M 常用 Ctrl-C 重新起始 ( re-initialize ) 系統。

許多終端機有它們自己的一些重定義。例如，Videx Videoterm 的 Ctrl-A 用來觸發大寫和小寫字元的輸入模式。在 Apple BASIC 中 Ctrl-A 也用來觸發以進入 EDIT 模式，所以應該把其他字元重定義成 Ctrl-A ( 例如，Ctrl-W )。

### 3. 載入使用者的 I/O 軟體推動程式

有些專門處理非標準化硬體的 I/O 軟體應被載入記憶體並合併入 I/O 架構框塊 ( I/O Configuration Block )。這可以選用數目 " 3 " 來做。但這個從磁碟載入的 I/O 軟體程式資料應具備特殊的內部格式才行。( 參考 " 軟體與硬體詳述 " 部分 )

如果鍵入 " 3 "，計算機會顯示：

```
++ LOAD USER I/O DRIVER SOFTWARE ++
```

```
OBJECT FILE NAME?
```

鍵入含程式的資料檔名並按下 RETURN 鍵，計算機會顯示以下訊息：

LOADING...

表示正做載入及合併的工作。做完之後計算機會返回顯示主名單。

#### 4. 讀／寫 I/O 架構框塊

這個功能可以讓你令 I / O 架構框塊自磁碟讀入記憶體，或自記憶體寫入磁碟，可以讓你檢查及修改任一 CP/M 磁碟的 I / O 架構框塊並將它們儲存回任意個數的磁碟中。把 I / O 架構框塊寫入 CP/M 系統磁碟機可使得由 CONFIGIO 程式所做的任何變更成為永久性變更。

鍵入 4，計算機會顯示：

+ READ/WRITE I/O CONFIGURATION BLOCK +

READ OR WRITE (R/W)?

如果鍵入 W，計算機會顯示：

DESTINATION DRIVE (A:-F:)?

插入 CP / M 系統磁碟並選擇正確的磁碟機。磁碟中的 I / O 架構框塊就會被在記憶體的新 I / O 架構框塊取代。使用 W，令“1”至“3”中所做的任何變更都成永久性變更。寫入工作完成後，返回顯示主名單 ( main menu )。

如果鍵入 R，計算機會詢問：

SOURCE DRIVE (A:-F:)?

插入CP/M系統磁碟並選擇正確的磁碟機。I/O架構框塊就會由磁碟被讀入記憶體。讀入工作完畢後，返回顯示主名單。

注意不要讀入一個具有不同記憶體大小（不同於正在工作的本系統）架構的CP/M磁碟之I/O架構框塊。（即，不要用56K CP/M的Apple系統去讀入48K的I/O架構框塊）。必須確定你所要讀/寫I/O架構框塊的磁碟與磁碟機A：上的磁碟有相同的CP/M架構（CP/M configuration）。

### 從別的計算機把CP/M檔案傳送過來：Download 與Upload

注意：使用這些程式之前必須已熟悉8080組合語言的程式設計。這些程式只供資深有經驗的程式師使用。

#### 目的：

DOWNLOAD及UPLOAD公用程式讓使用者可以經由RS-232串式傳輸鏈接（link）把CP/M檔案由另一CP/M機器傳送到Apple上來。UPLOAD公用程式必須鍵入非Apple的CP/M系統〔即來源機器上（source machine）〕，所以必須藉CP/M的DDT公用程式建構UPLOAD使其配合來源機器的特定I/O架構。在Apple使用DOWNLOAD時，溝槽2上必須備有Apple通信界面或CCS7710A串式傳輸界面板。DOWNLOAD存在任一SoftCard磁碟上。

使用Download及Upload公用程式，必須：

- 1 具有對 CP / M 之 DDT 程式及 8080 組合語言程式設計的操作知識。
- 2 具有一個以 CP / M 為基礎的計算機系統 ( Apple 以外的) , 此系統應具有由控制台 I / O 使用之外的額外 RS -232 串式傳輸 I / O 埠。
- 3 在 Apple 的溝槽 2 上必須具有 Apple 通信界面板或 California Computer System 7710A 串式傳輸界面板。

**使用規則：**

**步驟 1**

在來源機器 ( source machine ) 上使用 DDT 把 UPLOAD 機器語言程式由 0163H 位置開始輸入。

```

0163 3A 80 00 B7 11 D7 01 CA CC 01 CD 03 01
0170 0E 0F 11 5C 00 CD 05 00 3C 11 E5 01 CA CC 01 3E
0180 52 CD 43 01 CD 23 01 FE 53 C2 7F 01 3E 47 CD 43
0190 01 11 06 02 CD D2 01 0E 14 11 5C 00 CD 05 00 B7
01A0 C2 C9 01 21 80 00 0E 00 16 80 7E CD 43 01 A9 4F
01B0 23 15 C2 AA 01 79 CD 43 01 CD 23 01 FE 42 CA A3
01C0 01 FE 47 CA 97 01 C3 B9 01 11 F4 01 CD D2 01 C3
01D0 00 00 0E 09 C3 05 00 43 6F 6D 6D 61 6E 64 20 45
01E0 72 72 6F 72 24 46 69 6C 65 20 6E 6F 74 20 66 6F
01F0 75 6E 64 24 0D 0A 55 50 4C 4F 41 44 20 43 6F 6D
0200 70 6C 65 74 65 24 55 70 6C 6F 61 64 69 6E 67 2E
0210 2E 2E 24 FE
    
```

在 0100H 輸入以下 3 拜：

C3 63 01 (指令 JMP 0163H)

輸入之後再三地檢查，確定輸入資料正確無誤。用 DDT 的“L”命令列印程式，拿它與本書後頁的 UPLOAD 列印表相比較。在修補它之前，先跳出 DDT，鍵入 SAVE 2 UPLOAD.COM 把程式保存起來。（參照“CP/M 參考手冊”可得到 DDT 的詳細說明）

## 步驟 2

這一步驟要修補 UPLOAD 使它認知來源機器的串式傳輸 I/O 埠。使用 DDT 寫入下述三個副程式來完成它。每個程式都應從指定位址開始，寫入已配置 32 拜長的空間中。

1. 0103H 啓始設定串式傳輸埠 這個常式需啓始設定來源機器的串式傳輸埠。（鮑率（Baud rate），資料格式（data format），等等）。資料格式應設為 8 個資料比次，一個停止比次（stop bit），無同位（parity）比次，這樣做才會與 Apple 通信板或 CCS7710A 板互相配合。
2. 0123H 回輸串式傳輸埠的狀態 假如串式傳輸埠仍未有完備字元，這副程式必須回輸 A = 00。如果有一拜已完備可用了，程式應讀入此拜將它放入 A 暫存器中。
3. 0143H 寫至串式傳輸埠 輸出一拜至串式傳輸埠。事先必須把所有的暫存器（包括 A 暫存器）內容儲存起來。

這些常式被寫入並合併入 UPLOAD 程式後，應再用 SAVE 命令把它保存起來。

**步驟 3**

用電纜將 Apple 和源機器連接起來。其中之一埠接成送出裝置 (DTE——Data Terminal Equipment)，另一埠接成爲接收裝置 (DCE——Data Communication Equipment)。有時候發送 (Xmit) 和接收 (Recv) 線 (RS-232 接頭的第 2, 3 腳) 應反接，或者一些特定的握手式控制 (hand shaking) 線應短路接在一起。如果你用 CCS7710A 串式傳輸界面板，在 Apple 端應把第 4, 6, 20 腳都接在一起。確定兩個串式傳輸埠定著相同的資料格式。

**步驟 4**

當 UPLOAD 已修補好且電纜也接妥之後，就可以開始使用了。在 Apple 上，鍵入

```
DOWNLOAD fname.typ
```

其中 fname.typ 是要傳過來存入的程式指定名稱。

在來源機器上，鍵入

```
UPLOAD fname.typ2
```

其中 fname.typ<sub>2</sub> 是所欲傳送的檔名。一旦建立好通信管道，Apple 就顯示出：

```
DOWNLOADING
```

而來源機器上會顯示出：

```
UPLOADING...
```



每當 128 拜的資料錄傳送完畢後，Apple 螢幕就打出一個句點（“.”）。假如傳送 128 拜資料錄時發生錯誤，就打出“B”，並試著再次傳送此資料錄。

傳送完畢之後，來源機器會適時顯示出：

UPLOAD COMPLETE

並返回 CP/M。

當這個訊息出現在來源機器時，在 Apple 上鍵入 Ctrl-C，磁碟會轉一會兒，然後 Apple 顯示出：

DOWNLOAD COMPLETE

並返回 CP/M。

傳送每一檔案都要重複這個過程。如果希望操作一次就傳送許多檔案，可以利用 CP/M 的 SUBMIT 程式。

SOURCE LISTING: UPLOAD

```

                UPLOAD
                WRITTEN 5/80 BY NEIL KONZEN
                (C) 1980 MICROSOFT

0000 =      BOOT EQU 0000H      .BOOT SYSTEM
0005 =      BDOS EQU 0005H      .BDOS ENTRY POINT
000C =      FCB EQU 000CH      .DEFAULT FCB
0000 =      BUFFER EQU 0000H    .DEFAULT BUFFER ADDR

0100      ORG 0100H            .START AT TPA

0100 C36301  UPLOAD JMP  ENTRY    .JUMP AROUND ALL THESE

INIT                                .INITIALIZE SERIAL PORT

; THIS SUBROUTINE SHOULD DO ANY INITIALIZATION
; OF THE SERIAL PORT THAT MAY BE REQUIRED IF
; NONE IS REQUIRED, A 'RET' WILL DO

0103      DS 32                .SPACE FOR ROUTINE
                INPSTS          .INPUT STATUS/READ

; THE INPUT STATUS/READ ROUTINE RETURNS ZERO IN (A) IF NO
; BYTE IS AVAILABLE IF A BYTE IS AVAILABLE.
; THE BYTE SHOULD BE READ AND RETURNED IN THE
; (A) REGISTER

0123      DS 32                .SPACE FOR ROUTINE
                OUTPUT          .SEND A BYTE TO APPLE

; THE OUTPUT ROUTINE SHOULD TRANSMIT THE BYTE IN THE
; (A) REGISTER TO THE APPLE VIA THE SERIAL
; PORT ALL REGS INCLUDING (A) SHOULD BE SAVED

0143      DS 32                .SPACE FOR ROUTINE

0163 3A0000  ENTRY LDR  BUFFER    .MAKE SURE HE TYPED SOME SORT OF FILE NAME
0166 B7      ORA  A              .A NON-ZERO NO. OF CHARS IN CMD LINE
0167 1D7001  LXI  D, CMDMSG      .DEFAULT TO COMMAND ERROR MESSAGE
016A CACC01  JZ  EXIT              .QUIT
016D C08301  CALL INIT              .INITIALIZE SERIAL PORT
0170 0E0F   MVI  C, 15          .OPEN FILE COMMAND
0172 115C00  LXI  D, FCB                .POINT TO FCB
0175 C0E500  CALL BDOS                  .OPEN IT UP

0178 3C      INR  A              .FF RECORDS 7:RO
0179 11E501  LXI  D, FNMSG             .DEFAULT TO FILE NOT FOUND MSG
017C CACC01  JZ  EXIT              .NO FILE

; SEND A BUNCH OF 'R'S UNTIL DOWNLOAD ANSWERS

017F 3E52   ROL  P              .SEND 'R' FOR 'READY'
0181 C04301  CALL OUTPUT              .SEND VIA SERIAL LINE
0184 C02301  CALL INPSTS              .DID HE RESPOND?
0187 FE53   CPI  'S'          .S FOR 'SET'
0189 C27F01  JNZ  ROLP                .THEN TRY AGAIN

018C 3E47   MVI  A, 0          .
018E C04301  CALL OUTPUT              .SEND TO DOWNLOAD

0191 110602  LXI  D, WRMSG             .TELL HIM WE'RE DONG IT
0194 C0D201  CALL PRMSG

0197 0E14   READ MVI  C, 20          .READ SEQUENTIAL FUNCTION
0199 115C00  LXI  D, FCB                .
019C C0E500  CALL BDOS                  .DO READ 120 BYTES
019F 07      ORA  A              .ERROR?
01A0 C2C901  JNZ  EOF                  .END OF FILE
01A3 210000  TRVGH LXI  H, BUFFER        .POINT TO THE 120 BYTES
01A6 0E00   MVI  C, 0          .CHECKSUM = 0
01A8 1600   MVI  D, 00H             .BYTE COUNT = 120
    
```

```

01A8 7E      LOOP1  MOV    A,R      .GET CHR
01A9 CD4301  CALL   OUTPUT    .SEND IT (MUST SAVE (A))
01AE A9      MOV    C          .CALCULATE C-R-SUM
01AF 4F      MOV    C,R      .AND UPDATE IT
01B0 23      INX    M         .PT TO NEXT BYTE
01B1 15      DCR    D         .DEC BYTE COUNT
01B2 C0A001  JNZ    LOOP1    .KEEP GOING UNTILL ALL 120 SENT
01B5 79      MOV    A,C      .NOW SEND CHECKSUM BYTE
01B6 CD4301  CALL   OUTPUT    .SEND IT

                WAIT FOR VERIFICATION 'G'-GOOD, 'B'-BAD

01B9 CD2301  VFYLP  CALL   INPSTS    .GET A CHR
01BC FE42    CPI    'B'       .A BAD READ?
01BE CA0301  JZ     TRVGN     .START AGAIN
01C1 FE47    CPI    'G'       .A GOOD READ?
01C2 CA9701  JZ     READ     .GO GET NEXT RECORD THEN
01C6 C3B901  JMP    VFYLP     .CHR MUST NOT HAVE BEEN READ

01C9 11F401  EOF    LXI    D,DDMSG .OUTPUT MESSAGE
01CC CDD201  EXIT   CALL   PRMSG .ALL FINISHED
01CF C3B000  JMP

01D2 0E09    PRMSG  MVI    C,9      .PRINT MESSAGE FUNCTION
01D4 C3B500  JMP

01D7 426F6D6D61C7DMSG DB  'COMMAND ERROR'
01E5 46696C6538F7FMSG DB  'FILE NOT FOUND'
01F4 8089655B4CDD0MSG DB  '13.18. 'UPLOAD' COMPLETE'
0206 55786C6F61M0MSG DB  'UPLOADING 0'

0213      END
    
```

SOURCE LISTING: DOWNLOAD

```

;
;          DOWNLOAD
;
; THIS PROGRAM WORKS WITH AN APPLE COMMUNICATIONS
; INTERFACE OR A CCS 7710A SERIAL INTERFACE IN SLOT
; TWO.
;
; WRITTEN 5/80 BY NEIL KONZEN
; (C) 1980 MICROSOFT
;
0000 =      BOOT   EQU    0000H      ;BOOT SYSTEM
0005 =      BROS  EQU    0005H      ;BROS ENTRY POINT
005C =      FCB   EQU    005CH      ;DEFAULT FCB
0080 =      BUFFER EQU    0080H      ;DEFAULT BUFFER ADDR
;
E0AE =      CONSTS EQU    0E0AEH     ;COM OR CCS CARD STATUS LOC
E0AF =      CONDAT EQU    0E0AFH     ;COM CARD DATA - SLOT 2
E000 =      APPABD EQU    0E000H     ;APPLE KEYBOARD
;
;
0100          ORG    0100H          ;START AT PTA
;
;
0100 3A8000  DNMLD: LDA    BUFFER      ;MAKE SURE THERE'S A FILE NAME
0103 B7          ORA    A            ;ANY CHARS IN CMD LINE?
0104 11C001     LXI    D,CMDMSG     ;POINT TO CMD ERROR MSG
0107 CABB01     JZ     EXIT         ;QUIT
010A 0E13      MVI    C,19         ;DELETE FILE
010C 115C00     LXI    D,FCB       ;SAVE PTR TO FCB
010F B5          PUSH           ;
0110 CD0500     CALL   BROS        ;
0113 D1          POP     D          ;RESET PTR TO FCB
0114 0E16      MVI    C,22         ;MAKE FILE
0116 CD0500     CALL   BROS        ;
;
;
0119 3C          INR    A            ;CHECK FOR ERROR
011A 11CE01     LXI    D,BDSMSG     ;GET READY TO PRINT 'NO DIR. SPACE'
011B CABB01     JZ     EXIT         ;
;
;          WAIT TILL UPLOAD SENDS AN 'R'
;
0120 CD4001     RDYLP: CALL   RDCOM     ;GET A CHAR FROM COM CARD
0123 FE52      CPI    'R'          ;'R' FOR 'READY'?
0125 C22001     JNZ   RDYLP        ;
;
0128 1E53      MVI    E,'S'        ;GET 'S' FOR 'SET'
012A CD9301     CALL   WRCOM       ;
;
0129 CD4001     GETCEE: CALL   RDCOM     ;WAIT FOR 'C' FOR 'CO'
0130 FE47      CPI    'C'          ;
0132 C22001     JNZ   GETCEE       ;
;
0135 21F501     LXI    H,WRNMSG     ;POINT TO 'DOWNLDNG' MSG
0138 7E          MOV    A,H         ;GET CHR
0139 B7          ORA    A            ;SET CC'S
;
013A CA4701     JZ     TRYACH       ;GO DO DOWNLOAD
013D E5          PUSH           ;
013E 5F          MOV    E,A         ;CHAR TO LEJ FOR CONOUT
013F CD8001     CALL   CONOUT      ;
0142 E1          POP     H          ;
0143 23          INX    H           ;
0144 C33001     JMP    PRLP        ;

```

# 212 APPLE 軟體卡

```

0147 218000 TRYAGN: LXI H,BUFFER ;POINT TO 128 BYTE BUFFER
0148 0E00 MVI C,0 ;CLEAR CHASUM
014C 0E81 MVI C,B1R ;READ 128 BYTES + 1 CHASUM

;
014E CDA001 LOOPR: CALL RDCOM ;READ A BYTE
0151 77 MOV M,A ;STORE IT
0152 A9 XRA C ;CALC CHASUM
0153 4F MOV C,A ;UPDATE IT
0154 23 INX H ;NEXT BYTE
0155 15 DCR B ;DECREMENT BYTE COUNT
0156 C24E01 JNZ LOOP1 ;NOT DONE - CONTINUE
0159 B7 ORA A ;WAS CHASUM ZERO?
015A CA6A01 JZ GOODRD ;THINGS ARE OK

;
015D 1E42 BADRD: MVI E,'B' ;'B' FOR BAD'
015F CDBR01 CALL CONDUIT ;SEND 'B' TO UFDLOAD
0162 1E42 MVI E,'B'
0164 CD9301 CALL WRCOM
0167 C34701 JMP TRYAGN

;
016A 1E2E GOODRD: MVI E,'.' ;PRINT A PERIOD
016C CDBR01 CALL CONDUIT
016F 113C00 LXI D,FCB ;POINT TO FCB
0172 0E15 MVI C,21 ;WRITE SEU.
0174 CD0500 CALL BDOS
0177 1E47 MVI E,'G' ;SEND UFDLOAD A 'G' FOR 'GOOD'
0179 CD9301 CALL WRCOM
017C C34701 JMP TRYAGN

;
;
0177 3210E0 DONE: STA APPABD+10H ;CLR NBD STROBE
0182 113C00 LXI D,FCB
0185 0E10 MVI C,16 ;CLOSE THE FILE
0187 CD0500 CALL BDOS
018A 11E101 LXI D,DOWNMSG ;ALL DONE MSG
018D CDB401 EXIT: CALL PRMSG ;PRINT THE MESSAGE
0190 C30000 JMP

;
0193 3AAEE0 WRCOM: LDA CONSTS ;COM CARD STATUS
0196 E602 ANI 2 ;CHECK BIT 2
0198 CA9301 JZ WRCOM
019B 7B MOV A,E ;GET CHAR TO SEND
019C 32AFE0 STA CONDAT ;STORE HERE
019F C9 RET

;
01A0 3AAEE0 RDCOM: LDA CONSTS ;COM CARD STATUS
01A3 1F RAR ;STS BIT TO CARRY
01A4 DAB201 JC READIT
01A7 3A60E0 LDA APPABD ;SEE IF CTRL-C TYPED
01AA FEB3 CPI 081H ;?
01AC CA7F01 JZ DONE

;
01AF C3A001 JMP RDCOM ;NO. WAIT FOR CHAR

;
01B2 3AAFE0 READIT: LDA CONDAT ;GET INCOMING CHARACTER
01B5 C9 RET

;
01B4 0E09 PRMSG: MVI C,9 ;PRINT MESSAGE
01B8 C30500 JMP BDOS

;
01B9 0E02 CONDUIT: MVI C,2 ;CONSOLE OUTPUT
01BD C30500 JMP BDOS

01C0 436F6D6E1CNDMSG: DB 'Command Errors'
01CE 4E6F206469NDMSG: DB 'No directory spaces'
01E1 0D0A46F77DDNMSG: DB '13,10: Download completes'
01F5 446F776E6CWRNMSG: DB 'Downloading' 0

0201 END

```