IBM PC BIOS
程式剖析

IBM PC的推出，幾乎搶去了Apple的市場，其它的十六位元個人用電腦也相形失色。招架不住IBM PC的來勢洶洶，各家電腦廠商無不推出所謂相容性電腦來和IBM PC相抗衡，因為一句IBM PC相容性電腦，即可代替所有的廣告，並可享有IBM公司為IBM PC所做的促銷行動的成功。但相容性電腦遭遇到專利權和著作權的法律問題。IBM公司在臺灣所註册的三項硬體專利權，並非IBM公司用來對相容性電腦採取不利行動的法律依據。真正讓相容性電腦招架不住的是基本輸入輸出系統（BASIC INPUT/OUTPUT SYSTEM, BIOS）的著作權。

BIOS是一段8K位元組（byte）的程式，而且燒在僅讀記憶體內（ROM），其功用在於輸入和輸出的處理。它包含了數段小程式，這些小段程式就是系統和週邊裝置的界面程式。沒有了這些界面程式，使用者就不能由鍵盤輸入命令給系統，系統也無法將數據顯示在螢幕上。更無法將資料儲存在磁片或錄音帶上。當然也無法將數據印在報表紙上和傳輸到大電腦上或其它的週邊裝置或個人電腦。由以上的說明，可想而知BIOS是相當重要的。

BIOS內有很多不需要的指令，這些指令也是IBM公司用來對抗相容性電腦的暗樁。本書將BIOS程式一個指令一個指令的解析開來，並配合流程圖和例子來說明。但筆者並不指出這些暗樁，因為筆者也不鼓勵仿冒。筆者倒是認為當讀者熟讀完本書後，也就可以自己來寫BIOS程式了。IBM公司在抓仿冒品時，並不以功能相同為依據，而是以書寫程式的方式為依據。換句話說，相容性電腦的BIOS程式的功能要和IBM PC相容，但寫法不能和IBM PC的BIOS程式相同。

儒林圖書公司 印行

IBM PC BIOS
程式剖析

林書華 著

IBM PC BIOS
程式剖析

林書華 著

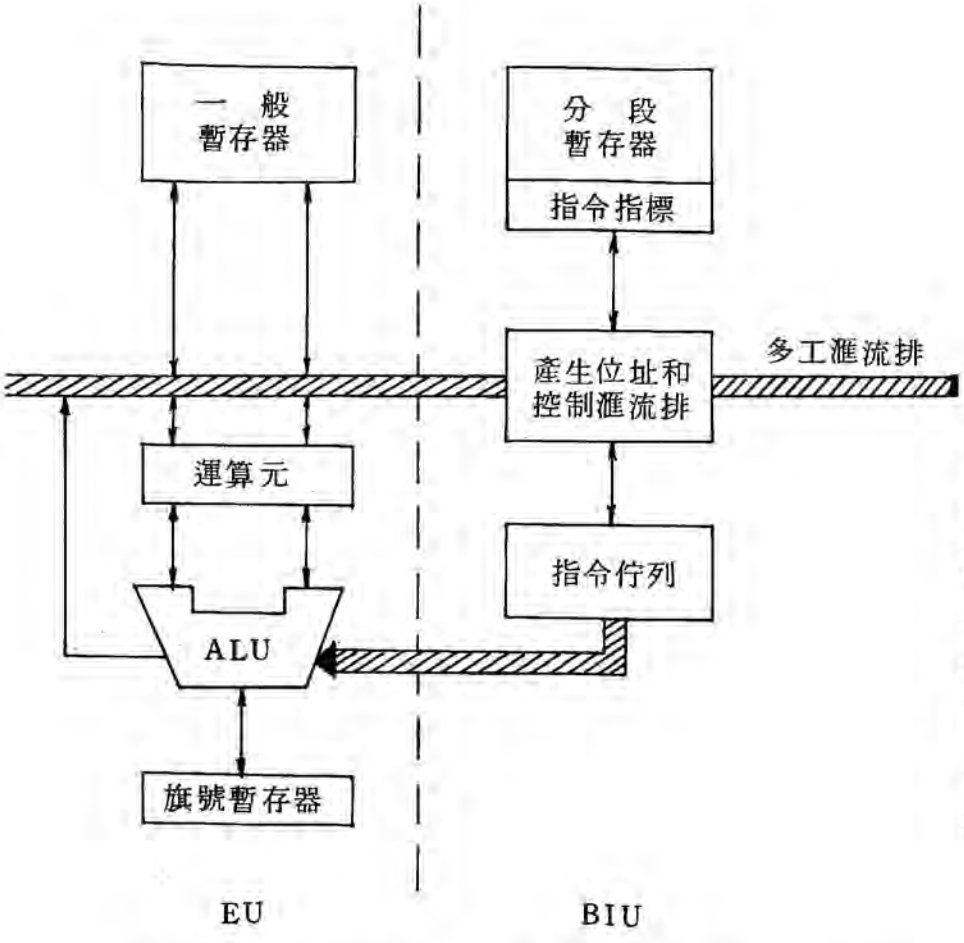
儒林圖書公司 印行

目　錄

第一章

8088組合語言介紹

雖然8088的資料滙流排（data bus）只有8條，但其內的資料滙流排有16條，我們還是叫它16位元（bit） 微處理機。因爲內在的資料滙流排有16條，所以8088的暫存器都是16位元。雖然如此，8088依然可以處理8位元的資料。我們知道8088的位址滙流排（address bus）有20條，所以它的記憶體最大容量可達一百萬個位元組（1M byte）。8088的特點是它的內部結構分爲二個單元。其一爲執行單元（Execution Unit, EU），它和外界是隔離的，它只執行指令，指令由滙流排界面單元（Bus Interface Unit, BIU）供給。另一單元爲滙流排界面單元（Bus Interface Unit），它負責外界的溝通工作，放出位址線執行取指令碼或資料的存取。它有一個可暫時存放4個指令碼的指令佇列（instruction queue），8088的滙流排界面單元（BIU）隨時將指令佇列填滿，以便供給執行單元（EU）執行，所以8088的滙流排是很“忙”的。



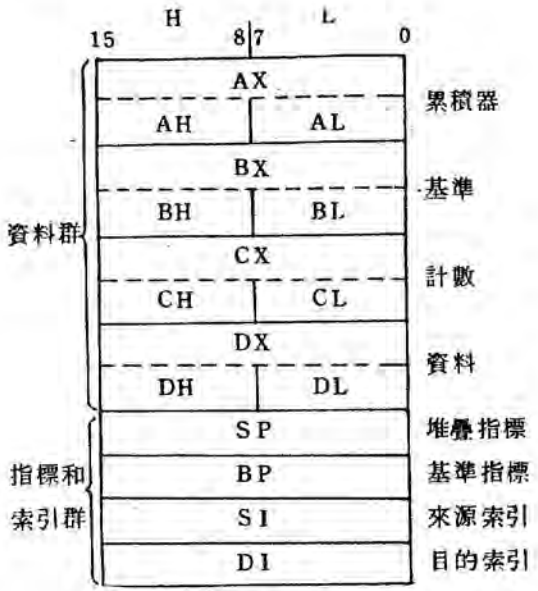
8088的暫存器可以分爲：分段暫存器(Segment Registers)、一般暫存器(General Registers)、指令指標(Instructions Pointer)和旗號(Flags)暫存器。分別敍述如下：

分段暫存器

分段暫存器有4個，它們都是16位元。

8088將一百萬位元組記憶體(1M byte memory)分割成16個64K位元組的段落。8088 可以同時進入其中4個段落。每個段落的基準位址(base address)存放在分段暫存器內。程式分段(code segment,CS)暫存器指著程式部分，指令碼由CS取來。堆疊分段(stack segment,SS)暫存器指著堆疊部分，有關堆疊的指令由SS處理。資料分段(data segment,DS)暫存器指著資料部分，它通常存放著程式的變數和資料。額外分段(extra segment,ES)暫存器指著額外部分，它通常用做資料的儲存。

一般暫存器



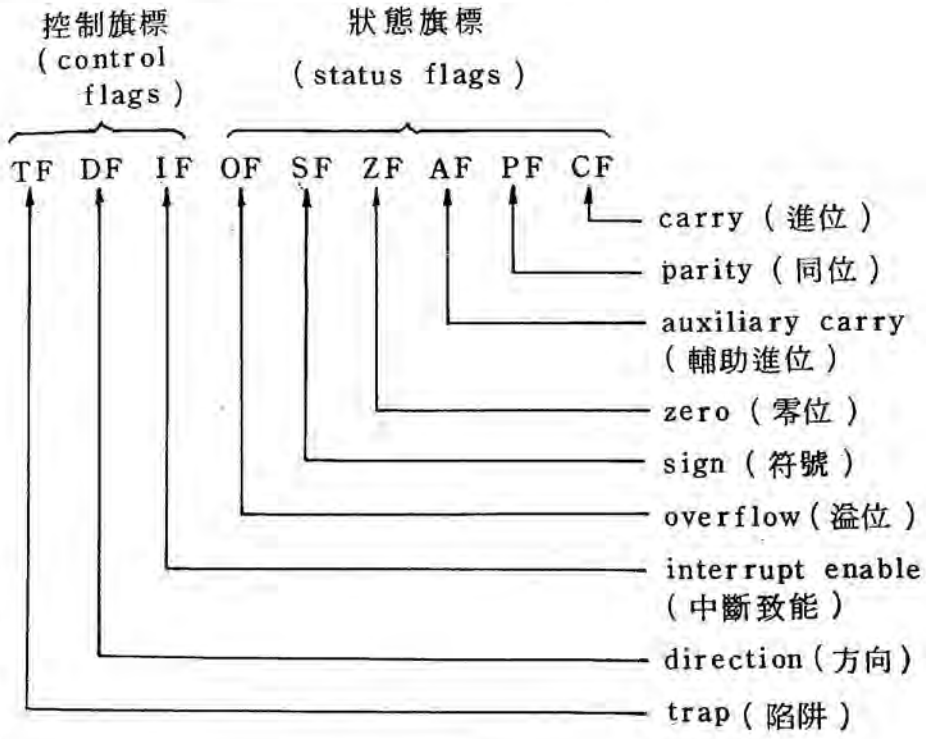
在資料群（data group）（AX，BX，CX和DX）中，每個16位元暫存器可以分爲二個8位元暫存器（AL，AH，BL，BH，CL，CH，DL和DH）。所以8088可以處理8位元的資料。指標和索引群（pointer and index group）（SP，BP，SI和DI）不能分成二個8位元暫存器，它們都是16位元暫存器。

分段暫存器和一般暫存器在產生實際位址（physical address generation）、定址模式（addressing mode）和組合語言（assembly language）中將有詳細的說明。

指令指標

它是一個16位元暫存器，其中放著相對於CS的間距（offset），指著下一個要取的指令碼。組合語言可以改變指令指標的值，例如JMP；也可以將指令指標存放到堆疊上，例如CALL；也可以從堆疊上取回指令指標，例如RET。

旗號暫存器

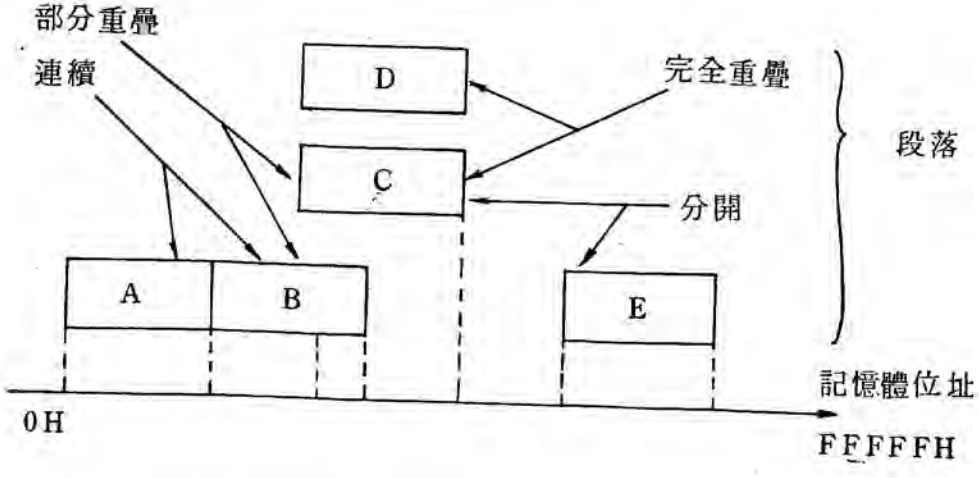


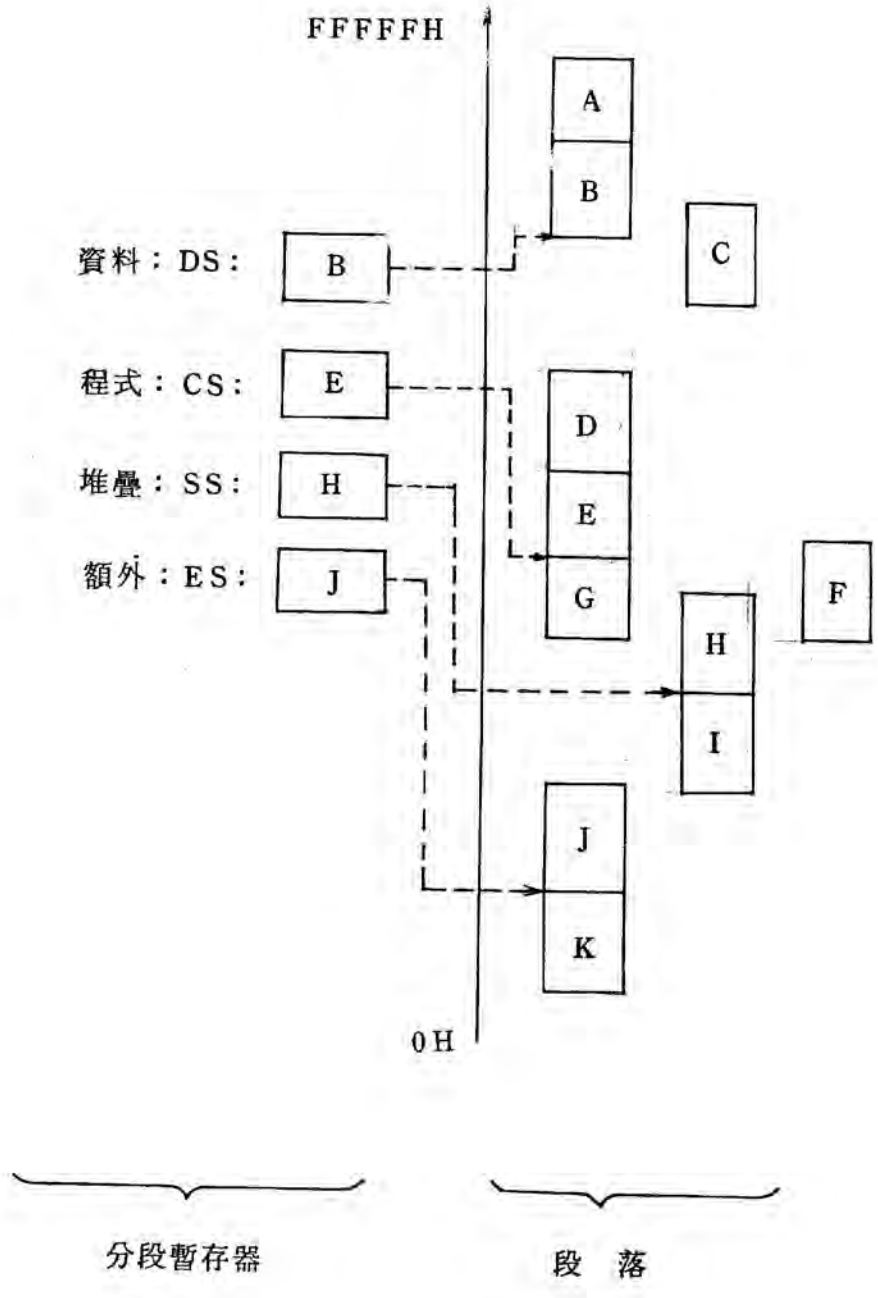
1. 如果AF爲1，就表示從低半位元組（low nibble）（位元0到位元3）有1個進位到高半位元組（high nibble）（位元4到位元7），或者是從高半位元組借位到低半位元組。AF旗號通常用在十進位制的算術中。
2. 如果CF爲1，就表示有一進位或一借位進入結果的最高位元（位元7或位元15），CF旗號通常用在加和減的算術中；旋轉指令也用到CF旗號。
3. 如果OF爲1，就表示在算術中產生溢位（overflow）現

象，這時會產生溢位中斷信號。

4. 如果SF爲1，結果的最高位元（位元7或位元15）爲1。SF旗號代表著結果的正負號（0表示正，1表示負）。
5. 如果ZF爲1，結果爲零。
6. 如果DF爲1，在字串指令（string instruction）中，索引指標（DI和SI）將會自動減少。如果DF爲0，在字串指令中，索引指標將會自動增加。
7. 如果IF爲1，就允許8088可以處理外來的可以遮罩的中斷信號。
8. 如果TF爲1，則8088就會進入單一步驟模式，方便於除錯工作。

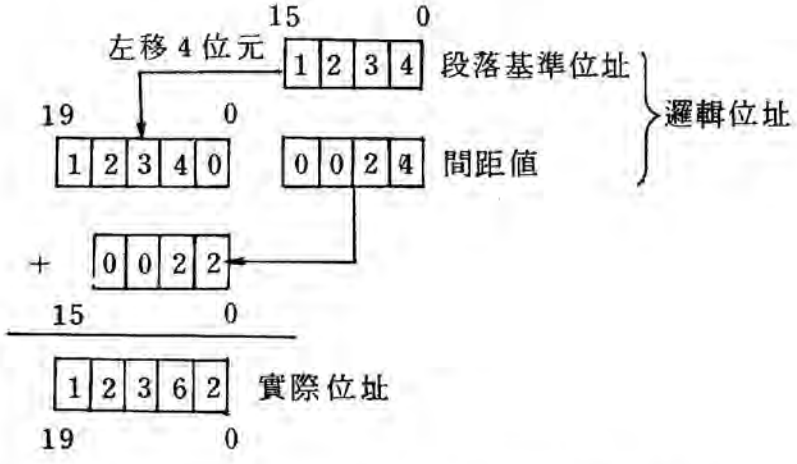
8088將一百萬位元組（1M byte）的記憶體分割成數個段落（segment），每一個段落有64K位元組容量。段落的基準位址保存在分段暫存器中（CS，ES，SS，DS），所以8088可以同時進入4個段落。段落可以相互重疊、連續或者分開，這個決定於分段暫存器內的起始位址。



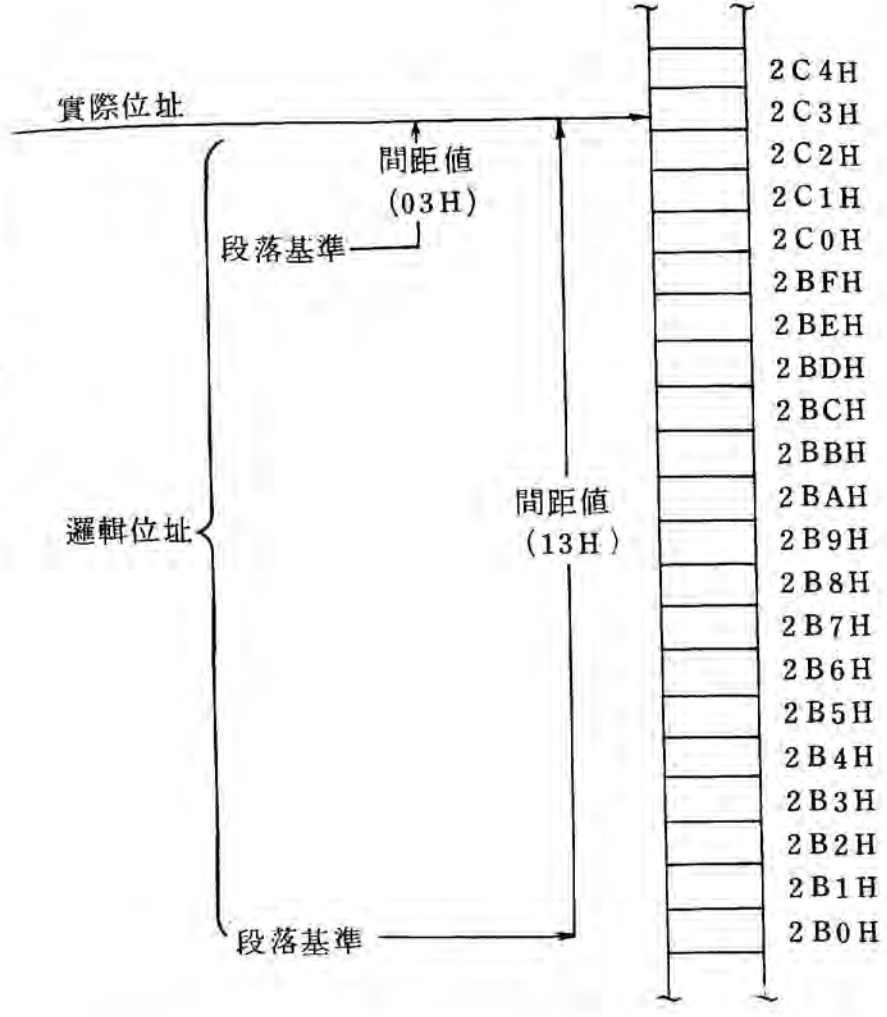


實際位址產生 (Physical Address Generation)

這裡所謂的實際位址，是爲了區分邏輯位址（logical address）之故。實際位址從零到FFFFFH，也就是8088 所能讀寫的記憶體的位址。在設計硬體時，我們注意的是實際位址，在寫組合語言時，我們就得注意邏輯位址了。邏輯位址由分段暫存器內的基準位址和間距所組成。分段暫存器內的基準位址（base address）和間距都是 16 位元。請看下面的例子，實際位址和邏輯位址的關係。



不同的邏輯位址可能指向相同的實際位址，請看下例。



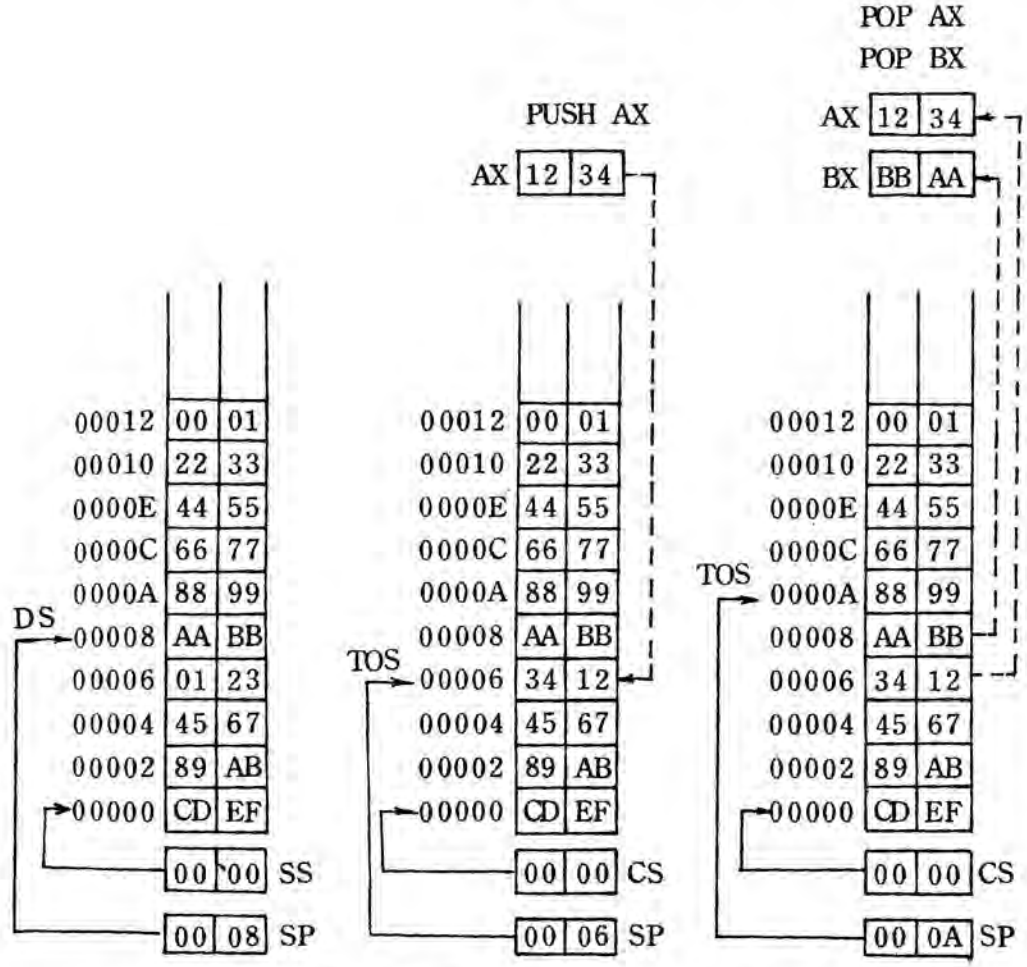
既然邏輯位址是由分段暫存器內的基準位址和一個間距值組成，那麼我們可以指令的功能來列出一個表格。這個表格很明白地告訴我們邏輯位址的組合和指令功能的關係。

指令功能	CPU 預設的基準點分段暫存器	可代替的基準點分段暫存器	間距
取指令(instruction fetch)	CS	無	IP
堆疊(stack)	SS	無	SP
變數(variable)	DS	CS,ES,SS	EA*
字串源地(string source)	DS	CS,ES,SS	SI
字串目的地(string destination)	ES	無	DI
BP 被使用做基準暫存器	SS	CS,ES,DS	EA*

* EA(Effective Address):有效位址。在定址模式中有詳細的說明。

上表中所謂的CPU預設的基準點分段暫存器(default segment base register),意思爲若程式設計者不告訴CPU 8088 需要使用其他的基準點分段,那麼 CPU 8088 就以其預設的分段暫存器內的位址爲基準位址。上表中,可代替的基準點分段暫存器裡列出三個分段暫存器,表示程式設計者可以使用三個的其中一個。上表中,間距值也就是IP,SP,SI,DI 各個暫存器內的值。EA 有效位址表示一個經過定址模式演算之後得到的位址間距。

下面的例子相當重要,它使我們了解堆疊的動作。如果堆疊分段暫存器(stack segment register)放著 0000H,堆疊指標放著0008H,那麼堆疊指標(TOS,TOP OF STACK)指著00008H的位址,請注意每一個PUSH或POP後堆疊指標和堆疊指標的變化(SS 不會改變)。注意:PUSH和POP 都是以二個位元組爲單位。



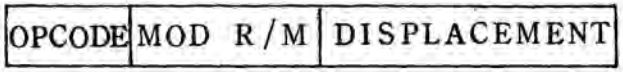
由上面的例子,我們可以看出做 PUSH 之前,先將堆疊指標減2,然後將暫存器 AX 內的低位元組放入低位址(00006H)內,高位元組放入高位址(00007H)內。POP 指令先將低位址內的資料放入暫存器的低位元組內,高位址內的資料放入暫存器的高位元組內,然後將堆疊指標加2。所以堆疊指標(TOS)是指著已被佔用的位址。PUSH 和 POP 指令也可以用在分段暫存器和記憶體位址上。這裡只是爲了解釋堆疊的動作,並不表示 PUSH 和 POP 指令只能用在一般暫存器上。BP 暫存器也可以當做堆疊的間距。

定址模式（Addressing Mode）

8088 的定址模式只有 7 種，但其中的變化萬千。用 8088 的組合語言寫程式時，這 7 種定址模式的提供相當方便。若要熟練的寫 8088 的組合語言程式，必須先熟悉它的定址模式。它們並不難，只要稍微用心即可了解。在說明 7 種定址模式之前，我們必須要了解有效位址（Effective Address，EA）的意義。

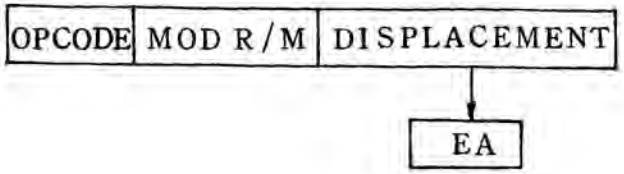
有效位址（EA）就是在決定記憶體位址相對於分段暫存器的間距（offset）。有效位址是由 8088 的執行單元（EU）根據機器碼（machine code）演算出來。在說明定址模式後，有效位址的計算會有詳細的說明。

我們都有一個圖配合說明每一個定址模式，這個圖是機器碼的表示。

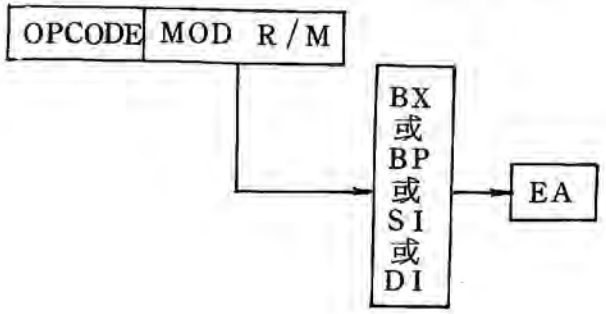


其中 opcode 爲機器碼第一個位元組，其值隨每一個指令而不同。MOD R/M 是機器碼第二個位元組，這個位元組相當重要，它提供了有效位址（EA）的計算根據。位移（DISPLACEMENT）的彈性很大，它可以不存在，也可以爲 8 位元或 16 位元。完全決定於MOD部份。

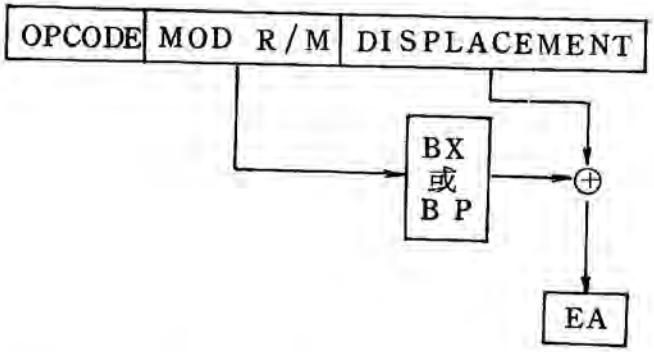
一、直接定址 (Direct Addressing)：這是最簡單的定址方式。有效位址就是位移（DISPLACEMENT），而沒有其它暫存器介入。直接定址通常用在簡單的變數或資料的存取。



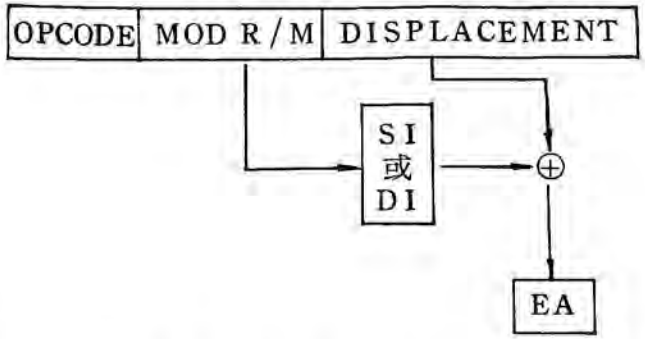
二、暫存器間接定址 (Register Indirect Addressing)：有效位址根據基準暫存器（Base Register）（BX或BP）或索引暫存器（Index Register）（DI或SI）演算而來。



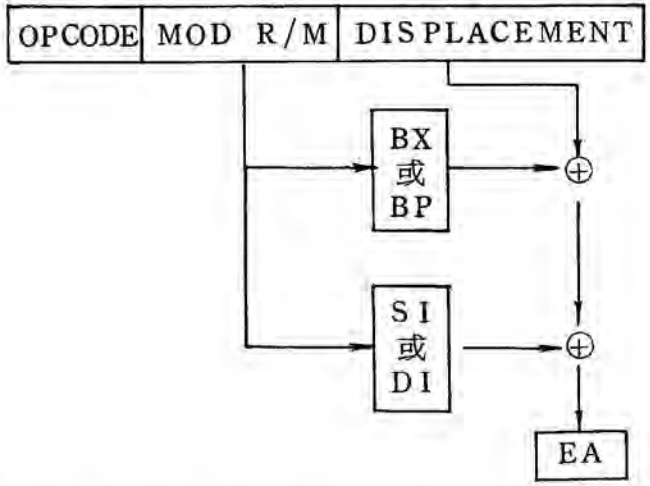
三、基準定址 (Based Addressing)：有效位址是 BX 或 BP 暫存器內的值加上位移（DISPLACEMENT）。



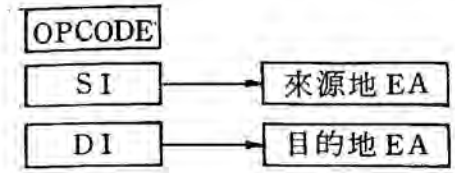
四、索引定址 (Indexed Addressing)：有效位址是 DI 或 SI 暫存器內的值加上位移（DISPLACEMENT）。



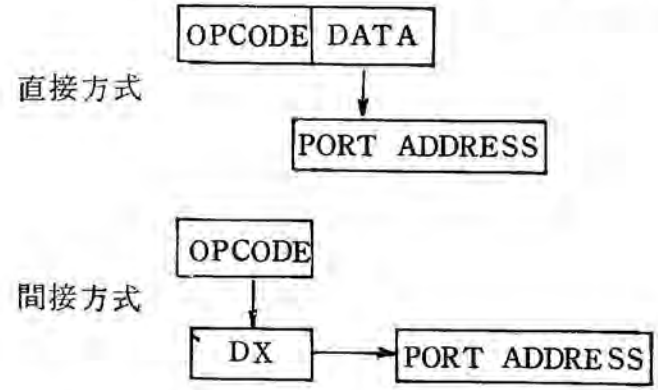
五、基準索引定址 (Based Indexed Addressing)：有效位址是DI或SI暫存器內的值加上BX或BP暫存器內的值再加上位移(DISPLACEMENT)。



六、字串定址 (String Addressing)：SI暫存器指著來源地(source)的第一個位元組(byte)或字組(words)，DI暫存器指著目的地(destination)的第一個位元組或字組。執行字組指令後(例如MOVS)，8088會自動調整DI和SI暫存器內的值(由DF旗號的值決定是增加還是減少)。

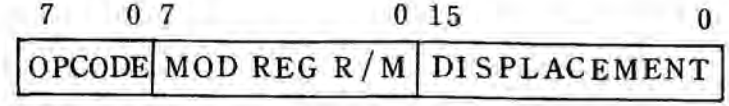


七、輸入/輸出埠定址 (I/O port Addressing)：這個定址方式分爲二種，其一爲直接方式，就是將I/O埠的位址直接放在opcode後面。另一種爲間接方式，就是將I/O埠的位址放在DX暫存器內，它的機器碼只有一個，也就是只有opcode。直接方式只能有256個I/O埠，間接方式可以到達65535個I/O埠。



有效位址(EA)的計算

需要計算有效位址的指令一定和記憶體位址有關。這些指令至少有2個機器碼(machine code)，8088就根據這些機器碼來計算有效位址。機器碼的表示如下所示：



其中opcode爲8位元，前面6個位元隨著每個指令的不同而不同

。opcode 的位元1爲D。D值代表著資料的方向，D等於零，表示資料由暫存器移至記憶體中。D等於1，表示資料從記憶體移至暫存器內。opcode 的位元0爲W，W值代表著位元組（byte）或字組（words）。W等於0表示資料爲位元組，W等於1表示資料爲字組。這裡所說的暫存器由機器碼的第二個位元組內的 REG 部分決定。我們來看2個例子。如果我們想把某一記憶體內的資料（如爲 byte ）移至某一暫存器內，則我們的機器碼的第一個位元組應該爲10001010。前面6個位元（100010）爲MOV 指令的機器碼。D值爲1，表示資料是移入暫存器內，而且此資料爲位元組（W＝0）。如果我們想把某一暫存器的資料（如爲words）移入記憶體內，則我們的機器碼的第一個位元組應該爲 10001001。前面六個位元（100010）爲MOV 指令的機器碼。D值爲0表示資料是移入記憶體內，而且此資料爲字組（words）。必須注意的是當W值爲0時，暫存器必爲8位元的暫存器（AL，AH，BL，BH，DL，DH，CL，CH）。當W值爲1時，暫存器必爲 16 位元的暫存器（AX，BX，CX，DX，SP，BP，SI，DI）。必須要記得的是，在做字組（words）資料轉移時，暫存器低位元組（位元0到位元7）的資料是相對於記憶體的較低位址，暫存器高位元組（位元8到位元15）的資料是相對於記憶體的較高位址。如果我們說將 AX 暫存器內的資料（AABB）移到位址 00000H 的地方，我們的意思爲將BB放到位址00000H的地方，將AA放到00001H的地方。

在機器碼的第二個位元組部分有三個單位，分別是MOD、REG和R/M。它們的分配如下所示。

BIT 7 6	5 4 3	2 1 0
MOD	REG	R/M

它們的意義爲：

MOD：決定位移部份的值。請看下表：

MOD	位移的值
00	位移＝0
01	位移爲8位元，符號延伸至16位元
10	位移爲16位元
11	沒有位移，有效位址爲R/M部份

其中① MOD＝00時，位移＝0，除了R/M＝110時例外。MOD＝00並且R/M＝110時，位移爲16位元。

② 符號延伸至16位元（sign-extended to 16 bit）的意思爲位元7的值重覆到位元8到位元15內。例如10001111，符號延伸至16位元後變成1111111110001111。00001111 符號延伸至16位元後變成0000000000001111。

③ MOD＝11，表示有效位址就是R/M 所代表的暫存器內的值。這個較爲特別，待會再詳細說明。

REG：代表著暫存器，請看下表。

16位元[W=1]		8位元[W=0]	
000	AX	000	AL
001	CX	001	CL
010	DX	010	DL
011	BX	011	BL
100	SP	100	AH
101	BP	101	CH
110	SI	110	DH
111	DI	111	BH

R/M：指出計算有效位址的暫存器，請看下表。

R/M	有效位址(EA)
000	(BX)+(SI)+DISPLACEMENT
001	(BX)+(DI)+DISPLACEMENT
010	(BP)+(SI)+DISPLACEMENT
011	(BP)+(DI)+DISPLACEMENT
100	(SI)+DISPLACEMENT
101	(DI)+DISPLACEMENT
110	(BP)+DISPLACEMENT
111	(BX)+DISPLACEMENT

其中① DISPLACEMENT的值由MOD決定。

② 將暫存器括弧起來表示該暫存器內的值。

我們來看例子：

① 機器碼爲8B901234

第1個位元組爲8B(10001011)，由前六個位元，我們知道此爲MOV指令，D=1表示資料移入暫存器內。W=1表示此資料爲字組(words)。

第2個位元組爲90(10010000)，所以MOD=10，則DISPLACEMENT爲16位元。REG=010，所以我們知道資料是移入DX暫存器內。R/M=000，表示出計算有效位址的式子爲(BX)+(SI)+DISPLACEMENT。

第3和第4個位元組表示出DISPLACEMENT的值，第3個位元組爲DISPLACEMENT的低位元組，第4個位元組爲高位元組，所以DISPLACEMENT的值爲3412。根據以上所述，我們可以知道有效位址爲BX和SI暫存器內的值相加，再加上DISPLACEMENT值(3412)。到目前爲止，我們尚未找到眞正的實際位址，因爲我們還沒將DS暫存器內的值左移4個位元，再加上有效位址。算出這個實際位址後，將其內的資料放入DL暫存器中，再將位址加1，然後取出其內的資料放入DH暫存器中。這時整個搬移動作已完成了。

② 機器碼8837

第1個位元組爲88(10001000)，由前面六個位元，我們知道此爲MOV指令，D=0表示資料移入記憶體內，W=0表示此資料爲位元組。

第2個位元組爲37(00110111)，所以MOD=00，則DISPLACEMENT爲0。REG=110，所以我們知道資料是由DH暫存器移入記憶體內。R/M=111，表示出計

算有效位址的式子爲（BX）+ DISPLACEMENT。根據以上所述，我們可以知道有效位址就是 BX 暫存器內的值（DISPLACEMENT=0）。此時再將 DS 暫存器內的值左移 4 位元，加上 BX 暫存器內的值，就算出了實際位址。將 DH 暫存器內的資料移入實際位址內，就完成了整個動作。

現在來解釋MOD=11 的意義，機器碼的第二個位元組的前二個位元（MOD）爲 11 的話，就表示不用計算有效位址，資料的搬移是從暫存器到暫存器。例如：

① MOV DS,AX[8ED8]

② SUB CX,CX[2BC9]

③ XOR AH,AH[32E4]

R/M 部分的 3 個位元就表示某個暫存器，前面的例子中① 000 表示 AX，② 001 表示 CX，③ 100 表示 AH 。資料的動作和每個指令有關。在介紹每個指令時，若有暫存器到暫存器的動作的話，會詳加說明。

我們說過如果我們不告訴 CPU 8088 指定的分段暫存器，8088 就會使用預設的分段暫存器。請看下面的指令。

```
ADD AX,[BP]          即相等於  ADD AX,SS:[BP]
MOV [BX+2],AX        即相等於  MOV DS:[BX+2],AX
XOR [BX+SI],CX       即相等於  XOR DS:[BX+SI],AX
AND [BP+SI],CX       即相等於  AND SS:[BP+XI],CX
MOV BX,[DI].FLD      即相等於  MOV BX,DS:[DI].FLD
AND [SI],CX          即相等於  AND DS:[SI],CX
SUB [SP],DX          即相等於  SUB SS:[SP],DX
```


如果我們指定其他的分段暫存器，則 8088 將會使用我們指定的分段暫存器，請看下面的指令。

```
ADD AX,DS:[BP]          不會執行成  ADD AX,SS:[BP]
MOV CS:[BX+2],AX        不會執行成  MOV DS:[BX+2],AX
XOR SS:[BX+SI],CX       不會執行成  XOR DS:[BX+SI],CX
AND DS:[BP+SI],CX       不會執行成  XOR SS:[BP+SI],CX
MOV BX,CS:[DI].FLD      不會執行成  MOV BX,DS:[DI].FLD
AND ES:[SI],CX          不會執行成  AND DS:[SI],CX
```

如果你是使用巨集組合語言（MACRO ASSEMBLY LANGUAGE）的話，指定分段暫存器時，只需在要計算有效位址的暫存器（BX，BP，SI 和DI）前寫入指定的分段暫存器的代號，然後加上一個冒號就可以了（如上所示）。巨集組合語言會將它轉換成 8088 能夠執行的機器碼。如此在產生實際位址時，會使用指定的分段暫存器。如果你是使用機器碼，那就必須在指令的機器碼前面，加上指定的分段暫存器的機器碼即可。下表爲分段暫存器和機器碼的關係。

```
ES : 26H
CS : 2EH
SS : 36H
DS : 3EH
```

我們來看個例子。

在巨集組合語言中我們的指令爲：

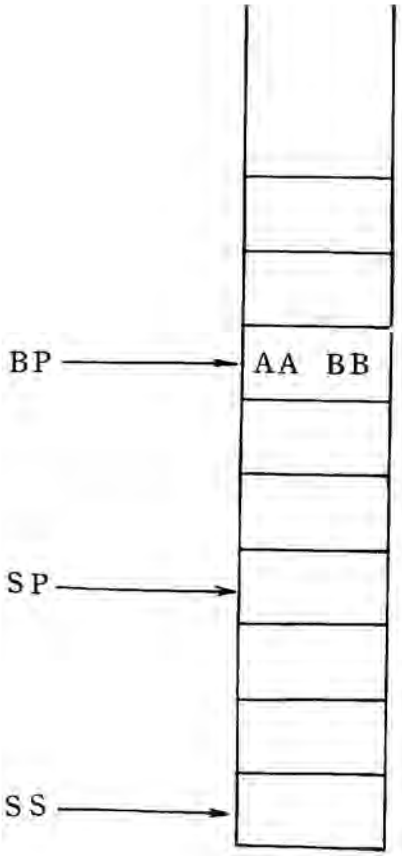
```
ADD AL,CS:[BX]
```

轉換成機器碼時爲 2E0207 。

如果在這裡我們沒有指定 CS，那麼不必寫入 2E，直接寫入 0207 就可以了（ 0207 爲MOV AL,[BX]的機器碼）。

第 10 頁的表格中，我們指出了指令的功能和邏輯位址的關係，其中有一條爲“ BP被使用爲基準暫存器”，CPU預設的分段暫存器爲 SS，我們現在來舉個例子解釋它的妙處。

我們知道堆疊的動作是由 SS 和 SP 暫存器來完成，而且資料都是字組（words），如果我們想在堆疊上讀取（或寫入）資料（字組或位元組），但不希望破壞堆疊指標暫存器內的值，那就得借重BP暫存器了。



由上圖，如果我們的指令爲MOV AL,[BP]，那麼AL暫存器內的值爲AA。如果我們的指令爲MOV AX,[BP]，那麼AX暫存器內的值爲BBAA。這二個指令都不影響SP內的值。

在介紹指令前，希望讀者熟悉有效位址的計算。並且請記牢BX,BP,DI和SI在計算有效位址CPU 8088 預設的分段暫存器。

暫 存 器	CPU 8088 預設的分段暫存器
BX	DS
BP	SS
DI或SI	DS
BX和DI	DS
BX和SI	DS
BP和DI	SS
BP和SI	SS

現在我們介紹 8088 的組合語言，在這裡只介紹ROM-BIOS常用的指令及其形式，詳細的參考資料，請查閱 INTEL的 86/87/88組合語言參考手册。

ADC (Add With Carry)

如果CF旗號等於1，則

(DEST)←(LSRC)+(RSRC)+1

如果CF旗號等於0，則

(DEST)←(LSRC)+(RSRC)

① 機器碼

000100dw	mod reg r/m

如果d=1，則LSRC=REG，RSRC=EA，DEST=REG。

如果d=0，則LSRC=EA，RSRC=REG，DEST=EA。

(a) 暫存器到暫存器，例如：

ADC AX,SI [機器碼13C6，mod=11，r/m表示著SI]

(b) 記憶體到暫存器，例如：

ADC AX,BETA[SI] [機器碼1304，如果位移不爲0，則機器碼的第二個位元組就不是04，如果有位移的話，位移緊接在機器碼的後面，位移的低位元組在前，高位元組在後。]

(c) 暫存器到記憶體，例如：

ADC BETA[DI],BX [機器碼111D，機器碼的第2個位元組如(b)所述。]

② 機器碼

0001010w	data	data if w=1

如果w=0則LSRC=AL，RSRC=data，DEST=AL。

如果w=1則LSRC=AX，RSRC=data，DEST=AX。

(a) 直接給定數值到累積器(Accumulator)，例如：

ADC AL,3H [機器碼1403]

ADC AX,333H [機器碼153303]

③ 機器碼

100000sw	mod 011 r/m	data	data if sw=01

LSRC=EA，RSRC=data，DEST=EA。

(a) 直接給定數值到記憶體，例如：

ADC BETA[SI],4 [機器碼801C04，機器碼的第2個位元組如①之(b)所述，如果有位移的話，位移在機器碼的第2個位元組後面，data的前面。]

ADC MEM-LOC[DI],7369H [機器碼811D6973]

註：如果機器碼的第1個位元組的S等於1，則表示data在相加之前先要符號延伸至16位元(data只有一個位元組，sw=11)。

(b) 直接給定數值到暫存器，例如：

ADC DH,65 [機器碼80DE65，mod=11，r/m=110。在這個情況下，有效位址爲r/m所代表的暫存器。]

被影響的旗號 AF,CF,OF,PF,SF,ZF

相信讀者一定不了解ADC指令。因爲其中有些英文簡寫尚未解釋，也有些必須該注意的地方尚未提出。以後介紹指令時，都將如ADC指令般形式。下面的解釋，讀者必須要熟識。

LSRC(LEFT SOURCE)：指令的左邊暫存器或記憶體，如ADC AX,SI指令中的AX暫存器，ADC BETA[DI],BX指令中的BETA[DI]。

RSRC(RIGHT SOURCE)：指令的右邊暫存器或記憶體，如ADC AX,SI指令中的SI暫存器，ADC AX,BETA[SI]指令中的BETA[SI]。

DEST(DESTINATION)：指令經過運算後，結果存放的地方

，可能是暫存器或記憶體。

LSRC＝REG：表示LSRC爲機器碼的第2個位元組內的REG部分所指的暫存器（REG和w值有關）。

RSRC＝EA：表示RSRC爲EA（有效位址），實際位址必須配合CPU 8088預設的分段暫存器或程式設計者指定的分段暫存器。

DEST＝REG：表示存放結果的地方爲機器碼的第2個位元組內的REG部分所指的暫存器。

（LSRC）：表示LSRC內的值。

（RSRC）：表示RSRC內的值。

（DEST）：表示存入DEST內。

＋號：表示相加。

←號：表示“存入”。

所以，ADC指令爲將LSRC內的值加上RSRC內的值，再加上1（如果CF＝1），然後將結果存入DEST內。LSRC、RSRC和DEST會隨著d值指令形式的不同（機器碼的不同）而不同。

我們知道有效位址的計算和機器碼的第2個位元組內的MOD和R/M有關，在機器碼中，我們通常不將位移寫出，讀者必須根據MOD值來加入位移。位移要緊跟在機器碼的第2個位元組後面，低位元組在前，高位元組在後（如果位移爲16位元的話）。

ADC指令有三種不同的機器碼形式，這是因爲處理的對象不同，例如①中有3種處理對象，這3種形式都使用相同的機器碼。

在②部分機器碼的第3個位元組爲“data if w=1”，意思爲，如果w值爲1，表示資料爲16位元，資料的低位元組在前（即第2個位元組），高位元組在後（即第3個位元組）。

在③部分的第1個位元組中，s取代了d。s表示正負符號（sign），如果s等於1，表示資料在處理之前，先要經過符號延伸至16位元。當s等於1時，表示資料要符號延伸至16位元，所示w值爲1，但在機器碼中，資料只有一個位元組。如果有暫存器參與的話（③的(b)），暫存器爲16位元暫存器（s＝1）。

在解釋指令的最後有一列爲“被影響的旗號”，意思爲指令執行後，會被影響的旗號。

請讀者再把ADC指令複習一次。在以後指令的介紹中，舉出的例子不再附上機器碼，請讀者自行練習。

ADD (addition)

LSRC內的值和RSRC內的值相加，然後存入DEST內。

（DEST）←（LSRC）＋（RSRC）

① 機器碼

000000dw	mod reg r/m

如果d＝1，則LSRC＝REG，RSRC＝EA，DEST＝REG。

如果d＝0，則LSRC＝EA，RSRC＝REG，DEST＝EA。

(a) 暫存器到暫存器，例如：ADD AX,BX

(b) 記憶體到暫存器，例如：ADD AX,BETA[SI]

(c) 暫存器到記憶體，例如：ADD BETA[DI],AX

② 機器碼

0000010w	data	data it w=1

如果w＝0，則LSRC＝AL，RSRC＝data，DEST＝AL。

如果w＝1，則LSRC＝AX，RSRC＝data，DEST＝AX。

(a) 直接給定數值到累積器（Accumulator），例如：ADD AL,3。

③ 機器碼

100000sw	mod 000 r/m	data	data if sw=01

LSRC=EA，RSRC=data，DEST=EA

(a) 直接給定數值到記憶體，例如：ADD MEM_WORD，48。

(b) 直接給定數值到暫存器，例如：ADD DX，1776

被影響的旗號：AF，CF，OF，PF，SF，ZF。

AND (AND: logical conjunction)

將LSRC內的值和RSRC內的值做“交集”（位元對位元），然後將結果存放於DEST內。此時CF和OF被設定成零。

(DEST)←(LSRC)&(RSRC)　[&表示“交集”]

(CF)←0

(OF)←0

① 機器碼

001000dw	mod reg r/m

如果d等於1，則LSRC=REG，RSRC=EA，DEST=REG

如果d等於0，則LSRC=EA，RSRC=REG，DEST=EA

(a) 暫存器到暫存器，例如：AND AX，BX

(b) 記憶體到暫存器，例如：AND AX，ALPHA[DI]

(c) 暫存器到記憶體，例如：AND ALPHA[DI]，AX

② 機器碼

0010010w	data	data if w=1

如果w等於0，則LSRC=AL，RSRC=data，DEST=AL

如果w等於1，則LSRC=AX，RSRC=data，DEST=AX

(a) 直接給定數值到累積器，例如：AND AL，7AH

③ 機器碼

1000000w	mod 100 r/m	data	data if w=1

LSRC=EA，RSRC=data，DEST=EA

(a) 直接給定數值到暫存器，例如：AND CH，3EH

(b) 直接給定數值到記憶體，例如：AND MEM_BYTE，46H。

被影響的旗號：CF，OF，PF，SF，ZF。

不確定的旗號：AF。

CALL (call a procedure)

在介紹CALL指令前，先來解釋2個名詞。

異段間（inter-segment）：程式分段暫存器（CS segment register）2個段落中所存的值不同。例如，CS內的值爲0000，如CS內的值爲F000，即爲2個不同的段落。在做CALL、JUMP等指令時，必須留心要呼叫的副程式或要跳入的程式是否爲另一個段落。若爲另一個段落，則必須執行異段間的CALL或JUMP等指令。

同段間（intra-segment 或 intra-group）：若呼叫的副程式或要跳入的程式在同一個段落中，則必須執行同段間的CALL或JUMP等指令。（group 爲若干個段落的集合，由程式設計者設定。）

如果爲異段間CALL，則CS內的值要被存入堆疊中，然後將IP存入堆疊內。最後再將DEST內的值放入IP中。同段間CALL不將CS值存入堆疊中。

(1) 若是「異段間」情況，則

(SP)←(SP)−2

((SP)+1:(SP))←(CS)

(CS)←SEG

(2) (SP)←(SP)−2

(3) ((SP)+1:(SP))←(IP)

(4) (IP)←DEST

如果爲異段間 CALL，則先將CS存入堆疊內，然後將 IP 也存入堆疊內。機器碼中的 SEG 將會放入 CS 中，DEST 內的值會成爲 IP。如果爲異段間 CALL（不論直接或間接），則從(1)開始。如果爲同段間（不論直接或間接）從(2)開始，即 CS 內的值不變。((SP)+1:(SP))←(CS)的意思爲 CS 內的低位元組存入 SP 所指的位址，CS 內的高位元組存入 SP+1 所指的位址。

① 機器碼

11101000	disp-low	disp-high

DEST=(IP)+disp

(a) 直接同段間，例如：CALL NEAR_LABEL。CS 內的值不變，原來的 IP 被存入堆疊中，然後原來的 IP 值加上位移值以取代 IP 值。disp-low 表示位移的低位元組。disp-high 表示位移的高位元組。

② 機器碼

10011010	offset-low	offset-high	seg-low	seg-high

DEST=offset，SEG=seg

(a) 直接異段間，例如：CALL FAR_LABEL

CS 值由 seg 部分取代，IP 值由間距部分取代。

③ 機器碼

11111111	mod 011 r/m

DEST=(EA)，SEG=(EA+2)

(a) 間接異段間，例如：CALL DWORD PTR[BX]

由mod和r/m部分算出有效位址，將EA和EA+1內的值取出放入 IP 中。將EA+2和EA+3內的值取代放入 CS 中。

④ 機器碼

11111111	mod 010 r/m

DEST=(EA)

(a) 間接同段間，例如：CALL word PTR[BX]

CS 內的值不變。由mod和r/m部分算出EA，然後將EA和EA+1內的值放入 IP 內。

被影響的旗號：沒有。

CBW (convert byte to word)

將AL暫存器符號延伸至16位元，也就是將AL的位元7，重覆抄入AH暫存器的每一個位元。若AL暫存器小於80H（位元7=0），經過CBW後，AH暫存器內的值爲00H。若AL暫存器內的值大於或等於80H（位元7=1），經過CBW後，AH暫存器內的值爲FFH。

機器碼

10011000

被影響的旗號：沒有。

CLC (clear carry flag)

將CF旗號設定爲0。

(CF)←0

機器碼

11111000

被影響的旗號：CF。

CLD (clear direction flag)

將DF旗號設定爲0。

(DF)←0

機器碼

11111100

被影響的旗號：DF

CLI (clear interrupt flag)

將IF旗號設定爲0。

(IF)←0

機器碼 | 11111010 |

被影響的旗號：1F

CMC (complement carry flag)

如果CF=0，則CMC指令使得CF=1，如果CF=1，則CMC指令使得CF=0。

機器碼 | 11110101 |

被影響的旗號：CF。

CMP (compare)

CMP指令將LSRC和RSRC內的值相減，只爲了影響旗號。LSRC和RSRC內的值都不會改變。常用來測試比較結果的旗號爲CF和ZF。如果LSRC大於RSRC，則CF=0，ZF=0。如果LSRC等於RSRC，則CF=0，ZF=1。如果LSRC小於RSRC，則CF=1，ZF=0。

① 機器碼 | 001110dw | mod reg r/m |

如果d=1，則LSRC=REG，RSRC=EA

如果d=0，則LSRC=EA，RSRC=REG

(a) 暫存器和暫存器，例如：CMP AX,DX

(b) 記憶體和暫存器，例如：CMP MEM_word,SI

(c) 暫存器和記憶體，例如：CMP DI,MEM_WORD

② 機器碼 | 0011110w | data | data if w=1 |

如果w=0，則LSRC=AL，RSRC=data

如果w=1，則LSRC=AX，RSRC=data

(a) 直接給定數值和累積器，例如：CMP AL,6

③ 機器碼 | 100000 sw | mod 111 r/m | data | data if sw=01 |

LSRC=EA，RSRC=data

(a) 直接數值和暫存器，例如：CMP BH,7

(b) 直接數值和記憶體，例如：CMP [BX][DI],6ACEH

被影響的旗號：AF,CF,OF,PF,SF,ZF。

CMPS/CMPSB/CMPSW (compare byte string, compare word string)

CMPS指令爲字串比較指令。RSRC使用DI暫存器爲索引，LSRC使用SI暫存器做爲索引，LSRC和RSRC各自找出其內的值然後做比較，LSRC和RSRC內的值不會被改變，但CMPS會影響旗號的值。執行完CMPS後，DI和SI暫存器會自動增加或減少一個單位。如果DF旗號爲0，則DI和SI暫存器會增加一個單位。如果DF旗號爲1，則DI和SI暫存器會減少一個單位。這裡所說的單位和w值有關，如果w=0，此一單位爲1；如果w=1，此一單位爲2。

(LSRC)-(RSRC)

如果DF=0，則

(SI)←(SI)+DELTA

(DI)←(DI)+DELTA

如果DF=1，則

(SI)←(SI)-DELTA

(DI)←(DI)-DELTA

機器碼　1010011w

如果w＝0，則LSRC＝(SI)，RSRC＝(DI)；

DELTA＝1(BYTE)。

如果w＝1，則LSRC＝(SI)＋1：(SI)，RSRC＝(DI)＋1：(DI)，DELTA＝2(word)。

被影響的旗號：AF，CF，OF，PF，SF，ZF。

例子：MOV　SI，STRING1

　　MOV　DI，STRING2

　　CMPS　STRING1，STRING2

DEC (decrement destination by one)

(DEST)←(DEST)－1

① 機器碼　01001reg

DEST＝REG

(a) 暫存器(此暫存器爲16位元暫存器)，例如：

　　DEC　AX。

② 機器碼　1111111w | mod 001 r/m

DEST＝EA

(a) 暫存器，例如：DEC　BL。

(b) 記憶體，例如：DEC　MEM_BYTE[DI]。

被影響的旗號：AF，OF，PF，SF，ZF。

HLT (halt)

使得8088進入停止(HALT)狀態，離開停止狀態必須有外界的中斷信號或者重置(RESET)動作。

機器碼　11110100

被影響的旗號：沒有。

IN (input byte and input word)

IN指令從I/O埠讀取一個資料(位元組或字組)。

(DEST)←(SRC)

① 機器碼　1110010w | port

如果w＝0，則SRC＝port，DEST＝AL。

如果w＝1，則SRC＝port＋1：port，DEST＝AX。

例如：IN　AX，word_port

　　IN　AL，BYTE_port

② 機器碼　1110110w

如果w＝0，則SRC＝(DX)，DEST＝AL。

如果w＝1，則SRC＝(DX)＋1：(DX)，DEST＝AX。

例如：IN　AX，DX

　　IN　AL，DX

被影響的旗號：沒有。

IN指令有2個不同的機器碼：①爲固定埠(fixed port)，②爲變動埠(variable port)。在固定埠中，直接將埠的號碼寫入，所以固定埠可以有0到255個埠。在變動埠中，將埠的號碼放入DX暫存器中，所以變動埠可以有0到64K個埠。

INC (increment destination by one)

(DEST)←(DEST)＋1

① 機器碼　01000reg

DEST＝REG

(a) 暫存器，例如：INC　AX。此暫存器爲16位元暫存器。

② 機器碼 | 1111111w | mod 000 r/m |

DEST = EA

(a) 暫存器，例如：INC CX，INC BL。

(b) 記憶體，例如：INC MEM_WORD[BX]。

被影響的旗號：AF，OF，PF，SF，ZF。

INT (interrupt)

當8088執行到INT指令時，它會根據TYPE的號碼，到中斷指標表（interrupt pointer table）去尋找事先設定處理中斷的副程式的基準位址。TYPE號碼可以由0到255個，每個TYPE佔據4個位元組（CS分段暫存器佔用2個位元組，IP暫存器佔用2個位元組），所以中斷指標表佔有1024個位元組。中斷指標表從位址00000開始到003FF。

當8088執行到INT n指令時，先將n乘以4，再加上2（即n × 4 + 2），然後將n × 4 + 2內的資料放入CS暫存器的低位元組，n × 4 + 3的資料放入CS暫存器的高位元組。填完CS暫存器，將n×4內的資料放入IP暫存器的低位元組，n × 4 + 1內的資料放入IP暫存器的高位元組。此時CS和IP暫存器內就是處理中斷的副程式的基準位址。在跳入此副程式前，原來的旗號（flags）暫存器、CS分段暫存器、IP暫存器都會陸續地被保存在堆疊中。

(SP) ← (SP) − 2

((SP) + 1 : (SP)) ← FLAGS

(IF) ← 0

(TF) ← 0

(SP) ← (SP) − 2

((SP) + 1 : (SP)) ← (CS)

(CS) ← (TYPE * 4 + 2)

(SP) ← (SP) − 2

((SP) + 1 : (SP)) ← (IP)

(IP) ← (TYPE * 4)

機器碼 | 1100110v | TYPE if v=1 |

如果v = 0，則TYPE = 3。

如果v = 1，則TYPE = TYPE。

例如：INT 3（一個位元組指令，v = 0）

INT 2（二個位元組指令，v = 1）

被影響的旗號：IF，TF。

IRET (interrupt return)

處理中斷的副程式通常最後一個指令爲IRET。表示回到呼叫此一副程式的程式去。IRET指令和INT指令相配合。

(IP) ← ((SP) + 1 : (SP))

(SP) ← (SP) + 2

(CS) ← ((SP) + 1 : (SP))

(SP) ← (SP) + 2

FLAGS ← ((SP) + 1 : (SP))

(SP) ← (SP) + 2

機器碼 | 11001111 |

被影響的旗號：全部。

JA/JNBE (jump if not below nor equal, or jump if above)

如果CF和ZF都爲0，那麼一個符號延伸至16位元的位移（

displacement)就要加上原來IP暫存器內的值，而成爲新的IP值。ZF等於0表示不相等，CF等於0表示不小於。（請參考CMP指令）

如果（CF）= 0而且（ZF）= 0，則

（IP）←（IP）+位移（符號延伸至16位元）

機器碼 | 01110111 | disp |

例如：JA TRGET-LABLE

JNBE TRGET-LABLE

被影響的旗號：沒有。

JNB/JAE (jump if not below, or jump if above or equal)

如果CF等於0，那麼一個符號延伸至16位元的位移（displacement)就要加上原來 IP 暫存器內的值，而成爲新的IP值。CF等於0，表示不小於。

如果CF = 0，則（IP）←（IP）+位移（符號延伸至16位元）。

機器碼 | 01110011 | disp |

例如：JNB TRGET-LABLE

JAE TRGET-LABLE

被影響的旗號：沒有。

JB/JC/JNAE (jump if below, or jump if not above nor equal, or jump if carry)

如果CF = 1，那麼一個符號延伸至16位元的位移（displacement）就要加上原來IP暫存器內的值，而成爲新的IP值；CF等於1表示小於。

如果CF = 1，則

（IP）←（IP）+位移（符號延伸至16位元）

機器碼 | 01110010 | disp |

例如：JB TRGET-LABEL

JNAE TRGET-LABEL

JC TRGET-LABEL

被影響的旗號：沒有。

JBE/JNA (jump if below or equal, or jump if not above)

如果CF = 1或是ZF = 1，那麼一個符號延伸至16位元的位移（displacement）就要加上原來IP暫存器內的值，而成爲新的IP值。CF等於1表示小於，ZF等於1表示等於。

如果CF = 1或ZF = 1，則

（IP）←（IP）+位移（符號延伸至16位元）

機器碼 | 01110110 | disp |

例如：JBE TRGET-LABEL

JNA TRGET-LABEL

被影響的旗號：沒有。

JCXZ (jump if CX is zero)

如果CX暫存器內的值爲0，那麼一個符號延伸至16位元的位移（displacement）就要加上原來IP暫存器內的值，而成爲新的IP值。

如果（CX）= 0，則

（IP）←（IP）+位移（符號延伸至16位元）

機器碼 | 11100011 | disp |

例如：JCXZ TRGET_LABEL

被影響的旗號：沒有。

JE/JZ (jump if equal, jump if zero)

如果ZF=1，那麼一個符號延伸至16位元的位移（displacement）就要加上原來IP暫存器內的值，而成爲新的IP值

如果ZF=1，則

(IP)←(IP)+位移（符號延伸至16位元）

機器碼 | 01110100 | disp |

例如：CMP CX,DX
JE LAB2
⋮
⋮
LAB2

被影響的旗號：沒有。

JMP (jump)

JMP指令爲無條件JUMP指令。其它的JUMP指令（JC，JNC,JA,JE……等等）是有條件的JUMP 指令。當條件成立時才執行JUMP指令，而且位移不得超過+127和-128。因爲位移爲位元組，在和IP值相加前要先做正負符號延伸（sign-extended）。當條件不成立時，就執行下一個指令。

JMP指令爲無條件指令。8088執行到JMP 指令時，就得跳到所指的副程式去。如果爲異段間 jump，CS 要被取代。如果爲同段間 jump，CS不須被取代，只有IP被取代即可。

如果異段間，則(CS)←SEG，(IP)←DEST。

如果同段間，則(IP)←DEST。

① 機器碼 | 11101001 | disp-low | disp-high |

DEST=(IP)+disp

(a) 同段間或直接同段間

例如：JMP NEAR_LABEL

② 機器碼 | 11101011 | disp |

DEST=(IP)+位移（符號延伸至16位元）

(a) 直接同段間短跳躍（intra-segment direct short），跳躍的範圍爲-128到+127個位元組。

例如：JMP SHORT NEAR_LABEL

③ 機器碼 | 11101010 | offset-low | offset-high | seg-low | seg-high |

DEST=offset，SEG=seg

(a) 直接異段間

例如：JMP FAR PTR LABEL_NAME

④ 機器碼 | 11111111 | mod 101 r/m |

DEST=(EA)，SEG=(EA+2)

(a) 間接異段間

例如：JMP ALPHA[BP][DI]

⑤ 機器碼 | 11111111 | mod 100 r/m |

DEST=(EA)

(a) 同段間或間接同段間

例如：JMP AX，JMP TABLE[BX]。

被影響的旗號：沒有。

JNC (jump if not carry)

如果CF=0，那麼一個符號延伸至16位元的位移（displacement）就要加上原來IP暫存器內的值，而成爲新的IP值。

如果CF = 0 ，則

(IP)←(IP)+位移（符號延伸至16位元）

機器碼 | 01110011 | disp |

例如：JNC TRGET_LABEL

JNC LABEL_NAME

被影響的旗號：沒有。

JNZ/JNE

如果ZF = 0 ，那麼一個符號延伸至16位元的位移(displacement)就要加上原來IP暫存器內的值，而成爲新的IP值。

如果ZF = 0 ，則

(IP)←(IP)+位移（符號延伸至16位元）

機器碼 | 01110101 | disp |

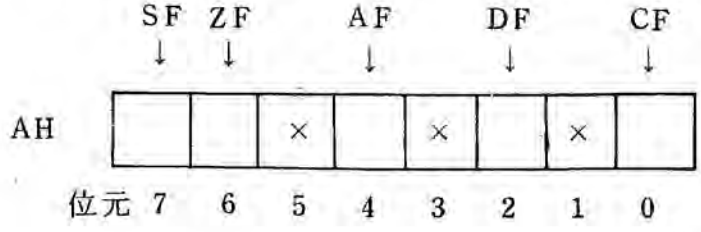
例如：JNZ TRGET_LABEL

JNE TRGET_LABEL

被影響的旗號：沒有。

LAHF (load AH from flags)

LAHF指令將部分旗號放入AH暫存器內。相對位址如下所示。



×號表示不理會

機器碼 | 10011111 |

被影響的旗號：沒有。

LDS (load data segment register)

LDS 指令從記憶體中取出資料放入指定的暫存器和DS分段暫存器內，共取出4個位元組，EA和EA+1內的資料放入指定的暫存器內。EA+2 、EA+3內的資料放入 DS 分段暫存器內。

(REG)←(EA)

(DS)←(EA+2)

機器碼 | 11000101 | mod reg r/m | （mod不能爲11）

例如：LDS BX,ADDR_TABLE[SI]

LDS SI,NEWSEG[BX]

被影響的旗號：沒有。

LODS (load byte or word string)

LODS指令將SRC內的值（位元組或字組）放入AL或AX暫存器內。SI暫存器指出SRC所在處。執行完LODS後，SI暫存器內的值會自動變化一個單位。如果DF=0 ，則SI會自動增加一個單位。如果DF=1 ，則SI會自動減少一個單位。如果w = 0 ，則此單位爲1 ；如果w = 1 ，則此單位爲2 。

(DEST)←(SRC)

如果(DF) = 0 ，則(SI)←(SI) + DELTA

如果(DF) = 1 ，則(SI)←(SI) − DELTA

機器碼 | 1010110w |

如果w = 0 ，則SRC = (SI) ，DEST = AL ，DELTA=1 。

如果w = 1 ，則SRC = (SI)+1 : (SI) ，DEST = AX ，DELTA = 2 。

例如：① CLD
MOV SI,OFFSET BYTE_STRING
LODS BYTE_STRING ;SI←SI+1

② STD
MOV SI,OFFSET WORD_STRING
LODS WORD_STRING ;SI←SI−2

被影響的旗號：沒有。

LOOP (loop, or iterate instruction sequence until count complete)

LOOP 指令先將 CX 暫存器內的值減 1，如果 CX 內的值不爲 0，則一個符號延伸至 16 位元的位移（displacement）就要加上原來 IP 暫存器內的值，而成爲新的 IP 值。如果 CX 內的值爲 0，則繼續執行下一個指令。

(CX)←(CX)−1
如果(CX)≠0，則
(IP)←(IP)+位移（符號延伸至16位元）

機器碼

11100010	disp

例如：①
```
        MOV CX,LENGTH ARRAY
        MOV AX,0
        MOV SI,AX
NEXT:   ADD AX,ARRAY[SI]
        ADD SI,TYPE ARRAY
        LOOP NEXT
        MOV CKS,AX
```

②
```
        MOV AX,0
        MOV BX,1
        MOV CX,N
        MOV DI,AX
FIB:    MOV SI,AX
        ADD AX,BX
        MOV BX,SI
        MOV FIBONACCI[DI],AX
        ADD DI,TYPE_FIBONACCI
        LOOP FIB
```

被影響的旗號：沒有。

注意：①LOOP 指令爲先將 CX 暫存器內的值減 1。
②跳躍的範圍爲 −128 到 +127 個位元組。

LOOPE/LOOPZ (loop on equal, or loop on zero)

LOOPE 和 LOOPZ 將 CX 暫存器內的值減 1，如果 CX 內的值不爲 0，而且 ZF 旗號等於 1，那麼一個符號延伸至 16 位元的位移（displacement）就要加上原來 IP 內的值，而成爲新的 IP 值。如果 CX 內的值爲 0，或 ZF = 0，則繼續執行下一個指令。

(CX)←(CX)−1
如果ZF = 1而且(CX)≠0，則
(IP)←(IP)+位移（符號延伸至16位元）

機器碼

11100001	disp

例如：
```
        MOV CX,LENGTH ARRAY
        MOV SI,-1
NEXT:   INC SI
        CMP ARRAY[SI],0
        LOOPE NEXT
```

```
        JNE  OKENTRY
        :
        :
        :
OKENTRY:
```

被影響的旗號：沒有。

注意：LOOPE/LOOPZ指令先將CX暫存器內的值減1。必須ZF=1而且CX內的值不爲0才做跳躍的動作，跳躍的範圍爲－128到＋127個位元組。

LOOPNE/LOOPNE (loop on not zero, or loop on not equal)

LOOPNE和LOOPNZ將CX暫存器內的值減1，如果CX內的值不爲0，而且ZF旗號等於0，那麼一個符號延伸至16位元的位移（displacement）就要加上原來IP內的值，而成爲新的IP值。如果CX內的值爲0，或ZF＝1，則繼續執行下一個指令。

(CX)←(CX)－1

如果ZF＝0而且(CX)≠0，則

(IP)←(IP)＋位移（符號延伸至16位元）

機器碼

11100000	disp

例如：

```
        MOV  AX,0
        MOV  SI,-1
        MOV  CX,N
NONZER: INC  SI
        MOV  AL,ARRAY1[SI]
        MUL  ARRAY2[SI]
```



```
        MOV  PRODUCT[SI],AX
        LOOPNZ  NONZER
```

被影響的旗號：沒有。

注意：LOOPNE/LOOPNZ指令先將CX暫存器內的值減1。必須ZF=0而且CX內的值不爲0才做跳躍的動作，跳躍的範圍爲－128到＋127個位元組。

MOV (move)

MOV指令將SRC內的值移入DEST內。SRC內的值不會改變，而且MOV指令不會影響旗號的值。

① 機器碼

1010001w	address-low	address-high

如果w＝0，則SRC＝AL，DEST＝address。

如果w＝1，則SRC＝AX，DEST＝address＋1：address。

(a) 從累積器（accumulator）到記憶體

例如：MOV ALPHA_MEM,AX
MOV GAMMA_BYTE,AL

② 機器碼

1010000w	addr-low	addr-high

如果w＝0，則SRC＝address，DEST＝AL。

如果w＝1，則SRC＝address＋1：address，DEST＝AX。

(a) 從記憶體到累積器

例如：MOV AX,BETA_MEM
MOV AL,GAMMA_BYTE

③ 機器碼

10001110	mod 0 reg r/m

SRC＝EA，DEST＝REG，reg不得爲01（CS）。

(a) 從記憶體到分段暫存器，CS分段暫存器除外。

例如：MOV ES,SS:NEW_WORD[DI]

(b) 從暫存器到分段暫存器，CS分段暫存器除外。

例如：MOV ES,DX
MOV DS,AX
MOV SS,BX

④ 機器碼

10001100	mod 0 reg r/m

SRC=REG，DEST=EA，(DEST)←(SRC)。

(a) 從分段暫存器到記憶體

例如：MOV GAMMA,CS
MOV DX,DS
MOV BX,ES

(b) 從分段暫存器到暫存器

例如：MOV DX,DS
MOV BX,ES

⑤ 機器碼

100010dw	mod reg r/m	addr-low	addr-high

如果 d=1，則SRC=EA，DEST=REG。

如果 d=0，則SRC=REG，DEST=EA。

(a) 從暫存器到暫存器

例如：MOV AX,BX
MOV CL,DH
MOV CX,DI

(b) 從記憶體到暫存器

例如：MOV AX,MEM_VALUE
MOV DX,ARRAY[SI]
MOV DI,MEM[BX][DI]

(c) 從暫存器到記憶體

例如：MOV ARRAY[DI],DX
MOV MEM_VALUE,AX
MOV [BX][SI],DI

注意：①如果爲暫存器到暫存器，如MOV CX,DX；機器碼的第3和第4個位元組不存在。

②如果計算有效位址時，爲暫存器間接定址而且沒有位移的話，如MOV [BX][SI],DX，MOV AX,[BP][DI]，機器碼的第3個和第4個位元組不存在。

⑥ 機器碼

1011wreg	data	data if w=1

SRC=data，DEST=REG。

(a) 直接給定數值到暫存器

例如：MOV AX,77
MOV DI,618

⑦ 機器碼

1100011w	mod 000 r/m	data	data if w=1

SRC=data，DEST=EA。

(a) 直接給定數值到記憶體

例如：MOV MEM_WORD,1999

(b) 直接給定數值到暫存器

例如：MOV BX,84

被影響的旗號：沒有。

MOVS (move byte string or move word string)

MOVS指令將DS分段暫存器內的資料（以SI暫存器內的值爲位移）移入ES分段暫存器內的某一個位址（以DI暫存器內的值爲位移）。執行完MOVS指令後，DI和SI暫存器會

自動變化一個單位。如果DF = 0，則DI和SI暫存器會自動增加一個單位。如果DF = 1，則DI和SI暫存器會自動減少一個單位。如果w = 0，則此一單位爲1，如果w = 1，則此一單位爲2。

(DEST)←(SRC)

如果DF = 0，則

(SI)←(SI) + DELTA

(DI)←(DI) + DELTA

如果DF = 1，則

(SI)←(SI) − DELTA

(DI)←(DI) − DELTA

機器碼 | 1010010w |

如果w = 0，則

SRC = (SI)，DEST = (DI)，DELTA = 1

如果w = 1，則

SRC = (SI) + 1 : (SI)，DEST = (DI) + 1 : (DI)，DELTA = 2

例如：

```
MOV  SI,OFFSET  SOURCE
MOV  DI,OFFSET  DEST
MOV  CX,LENGTH  SOURCE
REP  MOVS  DEST,SOURCE
```

被影響的旗號：沒有。

MUL (multiply accumulator by register-or-memory; unsigned)

MUL指令將AL或AX暫存器內的值乘以RSRC內的值。如果結果的前半部(EXT)爲0的話，CF和OF旗號等於0，如果結果的前半部(EXT)不爲0的話，CF和OF旗號等於1。

(DEST)←(LSRC) * (RSRC)，where * is unsigned

如果(EXT) = 0，則(CF)←0，(OF)←0

如果(EXT)≠0，則(CF)←1，(OF)←1

機器碼 | 1111011w | mod 100 r/m |

如果w = 0，則

LSRC = AL，RSRC = EA，DEST = AX，

EXT = AH

如果w = 1，則

LSRC = AX，RSRC = EA，DEST = DX : AX，

EXT = DX

例如：(a)

```
MOV  AL,LSRC_BYTE
MUL  RSRC_BYTE   ;結果在AX暫存器內
```

(b)

```
MOV  AX,LSRC_WORD
MUL  RSRC_WORD   ;結果的高字組在DX內
                 ,結果的低字組在AX內
```

被影響的旗號：CF,OF。

不確定的旗號：AF,PF,SF,ZF。

NEG (Negate, or form 2's complement)

NEG指令將所指定的暫存器或記憶體內的值做2補數(2's complement)。2補數的做法爲，先將暫存器或記憶體內的值做1補數(1變成0，0變成1，例如10100101的1補數爲01011010)，然後再加上1。

(EA)←SRC − (EA)

(EA)←(EA) + 1

如果(EA) = 0，則(CF)←0。

如果（EA）≠0，則（CF）←1。

機器碼 | 1111011w | mod 011 r/m |

如果w＝0，則SRC＝FFH

如果w＝1，則SRC＝FFFFH

例如：①如果AL內的值爲13H（00010011），則NEG AL使得AL暫存器內的值變爲－13H或EDH（11101101）。

②如果MEM-BYTE內的值爲AFH（10101111），則NEG MEM-BYTE使得MEM-BYTE內的值爲－AFH或51H（01010001）。

（MEM-BYTE只是爲了舉例說明所取的名字）

③如果SI暫存器內的值爲2FC3H，則NEG SI使得SI暫存器內的值爲D03DH。

被影響的旗號：AF，CF，OF，PF，SF，ZF。

注意：SRC－（EA），其中SRC＝FF或SRC＝FFFF，就是將EA內的值做1補數。

NOP (no operation)

當8088執行到NOP指令時，將不會做任何動作。NOP指令使得8088停止工作3個時序週期，然後再執行下一個指令。

機器碼 | 10010000 |

被影響的旗號：沒有。

NOT (not, or form 1's complement)

NOT指令將所指定的暫存器或記憶體內的值做1補數。

（EA）←SRC－（EA）

機器碼 | 1111011w | mod 010 r/m |

如果w＝0，則SRC＝FFH。

如果w＝1，則SRC＝FFFFH。

例如：①如果AH暫存器內的值爲13H（00010011），則NOT AH使得AH暫存器內的值爲ECH（11101100）。

②如果MEM-BYTE內的值爲AFH（10101111），則NOT MEM-BYTE使得MEM-BYTE內的值爲50H（01010000）。

③如果DX暫存器內的值爲2FC3，則NOT DX使得DX暫存器內的值爲D03CH。

被影響的旗號：沒有。

OR (or, inclusive)

OR指令將LSRC和RSRC內的值做“或集”（位元對位元），而且CF和OF旗號都被設定爲0。

（DEST）←（LSRC）|（RSRC）

（CF）←0

（OF）←0

① 機器碼 | 000010dw | mod reg r/m |

如果d＝1，則

LSRC＝REG，RSRC＝EA，DEST＝REG

如果d＝0，則

LSRC＝EA，RSRC＝REG，DEST＝EA

(a) 暫存器到暫存器

例如：OR AH，BL

(b) 記憶體到暫存器

例如：OR AX,MEM_WORD

(c) 暫存器到記憶體

例如：OR MEM_BYTE,DH

② 機器碼

0000011w	data	data if w=1

如果w = 0，則

LSRC=AL，RSRC=data，DEST=AL

如果w = 1，則

LSRC=AX，RSRC=data，DEST=AX

(a) 直接給定數值到累積器

例如：OR AL,11110110B

OR AL,F6H

OR AX,23F6H

③ 機器碼

1000000w	mod 001 r/m	data	data if w=1

LSRC = EA，RSRC = data，DEST = EA

(a) 直接給定數值到暫存器

例如：OR AH,F6H

OR CL,37

OR DI,23F5H

(b) 直接給定數值到記憶體

例如：OR MEM_BYTE,3DH

OR GAMMA[BX][DI],FACEH

被影響的旗號：CF,OF,PF,SF,ZF。

不確定的旗號：AF。

OUT (output byte and output word)

OUT指令將AL或AX暫存器內的值移入指定的I/O埠中。

(DEST)←(SRC)

① 機器碼

1110011w	port

如果w = 0，則

SRC = AL，DEST = port

如果w = 1，則

SRC=AX，DEST=port + 1：port

(0 < port < 255)

例如：OUT BYTE_PORT_VAL,AL

OUT WORD_PORT_VAL,AX

② 機器碼

1110111w

如果w = 0，則SRC = AL，DEST = (DX)。

如果w = 1，則SRC = AX，DEST = (DX) + 1：DX。

例如：OUT DX,AL

OUT DX,AX

被影響的旗號：沒有。

POP (pop word off stack into destination)

在本章前面已經討論過堆疊的動作。堆疊指標(SP)永遠指著已被佔用的位置。當執行 POP 指令時，先將堆疊指標指著的二個位元組取出放在指定的位置，然後再將堆疊指標內的值加2。

(DEST)←((SP) + 1：(SP))

(SP)←(SP) + 2

① 機器碼

01011 reg

(此reg爲16位元暫存器)

DEST = REG

例如：POP CX
POP DX

② 機器碼 | 000 reg 111 |

reg 不得爲 01（CS），這裡所指的 reg 爲分段暫存器。

DEST = REG

例如：POP SS
POP ES
POP DS

注意：沒有 POP CS 指令。

③ 機器碼 | 10001111 | mod 000 r/m |

DEST = EA

例如：POP ALPHA[BX]

被影響的旗號：沒有。

POPF (pop flag off stack)

POPF 指令將堆疊指標（SP）所指的二個位元組，放入旗號暫存器內。它們相對的位置如下所示：

OF ← 位元 11
DF ← 位元 10
IF ← 位元 9
TF ← 位元 8
SF ← 位元 7
ZF ← 位元 6
AF ← 位元 4
PF ← 位元 2
CF ← 位元 0

FLAGS ← ((SP) + 1 : (SP))
(SP) ← (SP) + 2

機器碼 | 10011101 |

被影響的旗號：全部。

注意：POPF 指令可以強迫某個旗號爲某值，尤其是不能以直接指令影響其值的旗號，例如 TF，沒有 SET TF、CLEAR TF 這樣的指令。如果現在要設定 TF 旗號爲 1，您可以這樣寫：

```
MOV AX,0100H
PUSH AX
POPF
```

PUSH (push word onto stack)

在本章前面已經討論過堆疊的動作。堆疊指標永遠指著已被佔用的位置。當執行 PUSH 指令時，先將堆疊指標（SP）減 2，然後將指定的暫存器或位置內的值移到堆疊指標所指的位置。

(SP) ← (SP) − 2
((SP) + 1 : (SP)) ← SRC

① 機器碼 | 01010 reg | （此 reg 爲 16 位元暫存器）

SRC = REG

例如：PUSH AX
PUSH SI

② 機器碼 | 000 reg 110 |

這裡所指的 REG 爲分段暫存器。

SRC = REG

例如：PUSH SS

```
PUSH ES
PUSH DS
PUSH CS
```

③ 機器碼 | 11111111 | mod 110 r/m |

SRC = EA

例如：PUSH BETA[BX]

被影響的旗號：沒有。

PUSHF (push flag onto stack)

PUSHF 指令將旗號暫存器保存在堆疊中。它們相對的位置如下所示：

```
OF  →  位元 11
DF  →  位元 10
IF  →  位元 9
TF  →  位元 8
SF  →  位元 7
ZF  →  位元 6
AF  →  位元 4
PF  →  位元 2
CF  →  位元 0
```

(SP)←(SP)－2

((SP)＋1：(SP))←FLAGS

機器碼 | 10011100 |

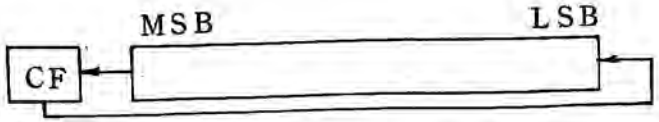
被影響的旗號：沒有。

注意：PUSHF 指令只是將旗號暫存器內的值拷貝（COPY）到堆疊中，並不會影響旗號暫存器內的值。PUSHF 指令主要的目的在保護旗號暫存器。請讀者比較 PUSHF 指令和 POPF 指令。

RCL (rotate left through carry)

RCL 指令將指定的暫存器或記憶體內的值向左旋轉數次。若旋轉的次數爲1，則直接將1寫在RSRC的位置，如RCL AH，1。如果旋轉的次數不爲1，那麼旋轉次數的值應該保存在CL暫存器內，如RCL AH，CL 。如果旋轉的次數爲1，而且旋轉後EA內的最高位元不等於CF旗號，則OF旗號等於1。如果旋轉的次數不爲1，OF旗號不確定。旋轉的過程是這樣的：先將旋轉次數（count）保存起來，如果次數不爲0，才做旋轉的動作；然後將 CF 旗號保存起來。將 EA 內的每一個位元向左移一位，最高的位元（位元7或位元15）移入CF旗號內，原先保存的CF旗號移入位元0中，最後將次數減1。

```
(temp)←COUNT
do while (temp)≠0
(tempcf)←(CF)
(CF)←high-order bit of (EA)
(EA)←(EA)*2+(tempcf)   註：(EA)*2即向左移一位。
(temp)←(temp)－1
if COUNT=1，then
    if high-order bit of (EA)≠(CF) then
        (OF)←1
    else (OF)←0
else (OF) undefined
```



機器碼	110100vw	mod 010 r/m

如果 v = 0 ，則 COUNT = 1。

如果 v = 1 ，則 COUNT = (CL)。

例如：① RCL AH,1
RCL BL,1

② MOV CL,3
RCL DH,CL

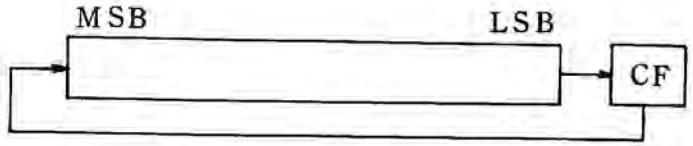
③ MOV CL,6
RCL MEM_WORD,CL

被影響的旗號：CF,OF。

RCR (rotate right through carry)

RCR 指令大致和 RCL 相似，只不過 RCR 指令是向右旋轉，而 RCL 指令向左旋轉。(請參考RCL指令)

```
(temp) ← COUNT
do while (temp) ≠ 0
(tempcf) ← (CF)
(CF) ← low-order bit of (EA)
(EA) ← (EA)/2      註 (EA)/2 即向右移一位。
high-order bit of (EA) ← (tempcf)
(temp) ← (temp) - 1
if COUNT = 1 , then
    if high-order bit of (EA) ≠ next-to-high-order
        bit of (EA) then (OF) ← 1
    else (OF) ← 0
else (OF) undefined
```

註：如果次數(count) 等於1，而且旋轉後最高位元和次高位元不相等(如位元15和位元14，位元7和位元6)，則(OF)=1。如果次數等於1，而且最高位元和次高位元相等的話，則(OF)=0。如果次數不等於1，則OF旗號不確定。

機器碼	110100vw	mod 011 r/m

如果 v = 0 ，則 COUNT = 1。

如果 v = 1 ，則 COUNT = (CL)。

例如：① RCR AH,1
RCR BL,1

② MOV CL,3
RCR DH,CL

③ MOV CL,6
RCR MEM_WORD,CL

被影響的旗號：CF,OF。

注意：請讀者了解RCR和RCL旋轉的動作，並注意COUNT = 1時，OF旗號的值。

REP/REPZ/REPE/REPNE/REPNZ (repeat string operation)

REP指令通常和字串指令共同使用。REP指令使得字串指令

（如MOVS、SCAS、CMPS等）重覆執行，直到CX暫存器內的值爲0。每執行一次REP指令後，CX暫存器內的值會自動減1。如果爲CMPS和SCAS指令時，當ZF旗號內的值不等於REP的機器碼的位元0 (Z)的值時，將不會繼續重覆執行CMPS和SCAS指令。

```
do while (CX) ≠ 0
    service pending interrupt (if any)
        ;如果有中斷信號的話會執行中斷處理。
    execute primitive string operation in succeeding
        byte    ;執行要重覆執行的字串指令。
    (CX) ← (CX) - 1
    if primitive operation is CMPS or SCAS and
        (ZF) ≠ Z then exit from while loop
        ;如果爲CMPS或SCAS指令時，而且(ZF)≠
         Z（機器碼的位元0），將會停止重覆動作。
```

機器碼 | 1111001 z |

例如：REP MOVS DEST,SOURCE
REPE CMPS DEST,SOURCE
REPNE SCAS DEST

被影響的旗號：和字串指令有關，請看CMPS、MOVS、SCAS等指令。

RET (return from procedure)

RET指令通常放在副程式的最後。RET指令先從堆疊中取出二個位元組放到IP暫存器內，如果爲異段間，那麼還得從堆疊中取出二個位元組放入CS暫存器內，當作返回的位址。如果我們加上一個偶數的直接數值，那麼堆疊指標將會加上這一個直接數值。


```
(IP) ← ((SP) + 1 : (SP))
(SP) ← (SP) + 2
if inter-segment then
(CS) ← ((SP) + 1 : (SP))
(SP) ← (SP) + 2
if add immediate to stack pointer then (SP) + data
```

① 機器碼 | 11000011 |

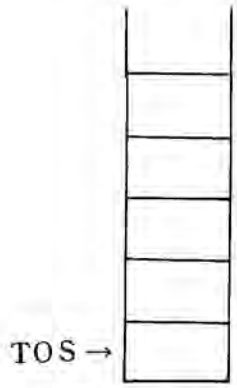
(a) 同段間

例如：RET

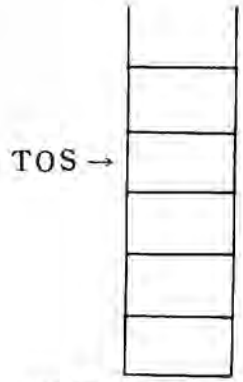
② 機器碼 | 11000010 | data-low | data-high |

(a) 同段間並加上一數字到SP暫存器

例如：RET 4



執行RET 4之前的堆疊



執行RET 4之後的堆疊

③ 機器碼 | 11001011 |

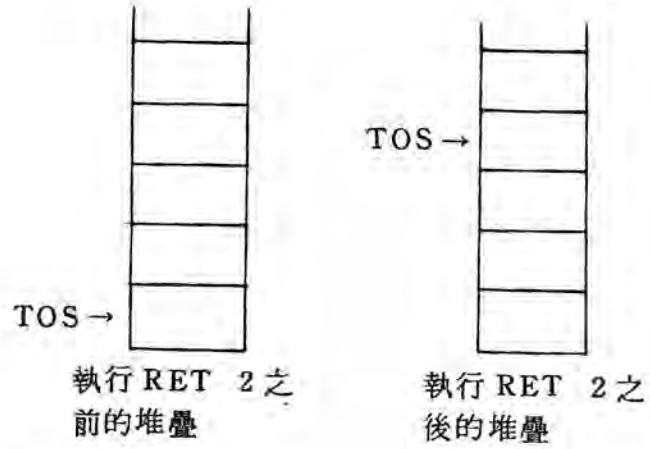
(a) 異段間

例如：RET

④ 機器碼 | 11001010 | data-low | data-high |

(a) 異段間並加上一數字到SP暫存器

例如：RET 2



執行RET 2之前的堆疊　　執行RET 2之後的堆疊

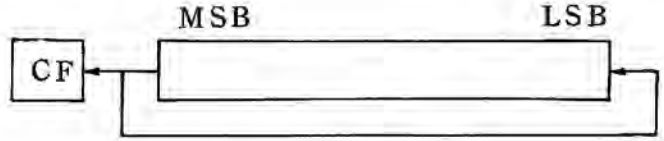
被影響的旗號：沒有。

ROL (rotate left)

ROL 指令將指定的暫存器或記憶體內的值向左旋轉數次。若旋轉的次數爲1的話，則直接將1寫在RSRC的位置，如ROL AH,1。如果旋轉的次數不爲1，那麼旋轉的次數應該保存在CL暫存器內，如ROL AH,CL 。如果旋轉的次數爲1，而且旋轉後EA內的最高位元不等於新的CF旗號的話，OF 旗號等於1。如果旋轉的次數爲1，而且旋轉後的 EA 內的最高位元等於新的 CF 旗號的話，OF 旗號等於0，如果旋轉的次數不爲1的話，OF 旗號不確定。旋轉的過程是這樣的：先將 COUNT（旋轉次數）保存起來，如果 COUNT不爲0，才做旋轉的動作。然後將 EA 內的最高位元移入 CF 旗號內。再將EA內的每一個位元向左移一位。將新的CF旗號移入EA的位元0中。原來的CF旗號內的值已被破壞掉。旋轉後，

將COUNT減1。

```
(temp)←COUNT
do while (temp)≠0
(CF)←high-order-bit of (EA)
(EA)←(EA)*2+(CF)      註：(EA)*2即向左移
                          一位。
(temp)←(temp)-1
if COUNT=1 , then
    if high-order-bit of (EA)≠(CF) then
       (OF)←1
    else (OF)←0
else (OF) undefined
```



機器碼

110100 vw	mod 000 r/m

如果 v = 0，則COUNT = 1。

如果 v = 1，則COUNT = (CL)。

例如：① ROL AH,1

② MOV CL,3
ROL DH,CL

③ MOV CL,6
ROL MEM-WORD,CL

被影響的旗號：CF,OF。

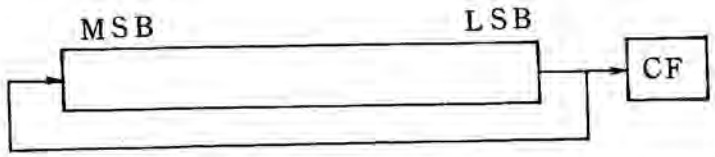
注意：請讀者比較RCL和ROL指令的不同。

ROR (rotate hight)

ROR 指令大致和 ROL 相似，只不過 ROR 指令是向右旋轉，

而ROL指令是向左旋轉。（請參考ROL指令）

```
(temp) ← COUNT
do while (temp) ≠ 0
(CF) ← low-order-bit of (EA)
(EA) = (EA) / 2     註：(EA) / 2即向右移一位。
    high-order-bit of (EA) ← (CF)
    (temp) ← (temp) - 1
if COUNT = 1 then
    if high-order-bit of (EA) ≠ next-to-high-order
        -bit of (EA) then (OF) = 1
    else (OF) = 0
else (OF) undefined
```



註：如果次數（count）等於1，而且旋轉後最高位元和次高位元不相等，則(OF)=1，如果次數等於1，而且旋轉後最高位元和次高位元相等的話，(OF)=0。如果COUNT不等於1，則OF不確定。

機器碼

110100vw	mod 011 r/m

如果v=0，則COUNT=1。

如果v=1，則COUNT=(CL)。

例如：①
```
ROR  AH,1
```
②
```
MOV  CL,3
ROR  DH,CL
```
③
```
MOV  CL,6
ROR  MEM_WORD,CL
```

被影響的旗號：CF,OF。

注意：請讀者自行比較RCR和ROR指令的不同。

SAHF

SAHF指令相對於LAHF指令。SAHF指令將AH暫存器內的值保存在FLAGS中，它們相對的位置如下所示：

(SF) ← 位元7
(ZF) ← 位元6
(AF) ← 位元4
(PF) ← 位元2
(CF) ← 位元0

機器碼

10011110

被影響的旗號：AF,CF,PF,SF,ZF。

SHL和SAL (shift logical left and shift arithmetic left)

SHL和SAL指令將指定的暫存器或記憶體內的值向左移數次。若左移的次數爲1，則直接將1寫在RSRC的位置，如SHL AH,1。如果左移的次數不爲1，那麼左移的次數應該保存在CL暫存器內，如SHL AH,CL。如果左移的次數爲1，而且左移後，EA的最高位元不等於新的CF旗號的值，則OF旗號等於1。如果左移的次數爲1，而且左移後，EA的最高位元等於新的CF旗號的值，則OF旗號等於0。如果左移的次數不爲1，則OF旗號不確定。左移的過程是這樣的：先將左移次數保存起來，如果次數不爲0，才做左移的動作。然後將EA的最高位元移入CF旗號內。再將EA內的每一個位元左移一位，EA的位元0補入0，最後將次數減1。

```
(temp)←COUNT
do while (temp)≠0
(CF)← high-order-bit of (EA)
(EA)←(EA)*2        註：(EA)*2即爲左移一位。
(temp)←(temp)-1
if COUNT=1 then
    if high-order-bit of (EA)≠(CF) then
        (OF)←1
    else (OF)←0
else (OF) undefined
```

機器碼

110100vw	mod 100 r/m

如果 v = 0，則COUNT = 1。

如果 v = 1，則COUNT = (CL)。

例如：① SHL AH,1

② SHL MEM_BYTE,1

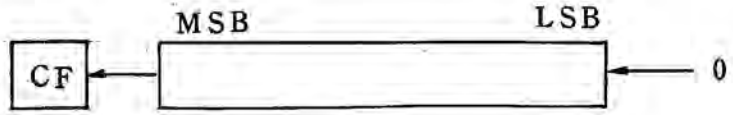
③ MOV CL,3
SHL DH,CL

④ MOV CL,6
SHL MEM_WORD,CL

被影響的旗號：CF,OF,PF,SF,ZF。

不確定的旗號：AF。

注意：SHL和SAL指令的左移動作是如下的：



SHR (shift logical right)

SHR 和 SHL 指令大致相似，只不過 SHL 指令是向左移，而SHR指令是向右移。（請參考 SHL 指令）

```
(temp)←COUNT
do while (temp)≠0
(CF)← low-order-bit of (EA)
(EA)←(EA)/2        註：(EA)/2即向右移一位。
(temp)←(temp)-1
if COUNT=1 , then
    if high-order-bit of (EA)≠next-to-high-order
       bit of (EA) then (OF)←1
    else (OF)←0
else (OF) undefined
```

註：如果次數爲1，而且右移後最高位元和次高位元不相等，則(OF)=1。如果次數爲1，而且最高位元和次高位元相等，則(OF)≠0。如果次數不等於1，則 OF 旗號不確定。

機器碼

110100vw	mod 101 r/m

如果 v = 0，則COUNT = 1。

如果 v = 1，則COUNT = 0。

例如：① SHR AH,1
SHR BL,1

② SHR MEM_BYTE,1

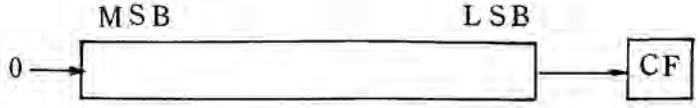
③ MOV CL,3
SHR DH,CL

④ MOV CL,6
SHR MEM_WORD,CL

被影響的旗號：CF，OF，PF，SF，ZF。

不確定的旗號：AF。

注意：SHR的右移動作如下：



SAR (shift arithmetic right)

SAR 指令大致和 SHR 指令相似，不同的地方爲 SHR 指令向右移時，最高位元一定補上 0；但 SAR 指令向右移時，最高位元不一定補上 0，它有可能補上 1。補上 1 或 0 完全在於要右移的值的正負符號值（也就是最高位元的值）。如果爲負數（即正負符號值爲 1，也就是最高位元爲 1），右移時最高位元補上 1。如果爲正數（即正負符號值爲 0，也就是最高位元爲 0），右移時最高位元補上 0。

```
(temp) ← COUNT
do while (temp) ≠ 0
(CF) ← low-order-bit of (EA)
(EA) ← (EA) / 2          "/"表示帶有正負符號的除法。
(temp) ← (temp) - 1
if COUNT = 1 then
    if high-order-bit of (EA) ≠ next-to-high-order
        -bit of (EA) then (OF) ← 1
    else (OF) ← 0
else (OF) ← 0
```

註：如果次數（count）爲 1，而且右移後最高位元和次高位元不相等，則 (OF) ← 1。如果次數爲 1，而且右移後最高位元和次高位元相等，則 (OF) ← 0。如果次數不等於 1，則 (OF) 旗號一定等於 0。

機器碼

110100vw	mod 111 r/m

如果 v = 0，則 COUNT = 1。

如果 v = 1，則 COUNT = (CL)。

例如：① SAR AH,1
SAR BL,1

② SAR MEM_BYTE,1

③ MOV CL,3
SAR DH,CL

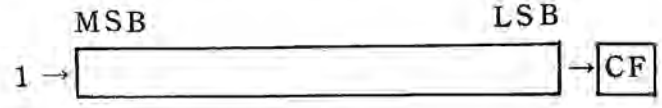
④ MOL CL,6
SAR MEM_WORD,CL

被影響的旗號：CF，OF，PF，SF，ZF。

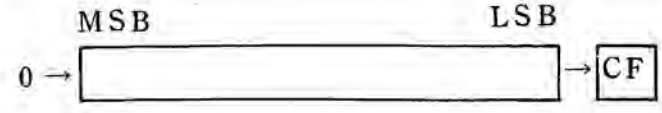
不確定的旗號：AF。

注意：SAR 右移的動作如下：

(a) 如果 high-order-bit = 1



(b) 如果 high-order-bit = 0



SBB (subtract with borrow)

SBB 指令將 LSRC 內的值減去 RSRC 內的值，將結果存放在 DEST 內。如果 CF 旗號內的值爲 1，那麼還得再減 1。

如果(CF)=1，則

(DEST)←(LSRC)-(RSRC)-1

如果(CF)=0，則

(DEST)←(LSRC)-(RSRC)

① 機器碼

000110dw	mod reg r/m

如果d=1，則

LSRC=REG，RSRC=EA，DEST=REG

如果d=0，則

LSRC=EA，RSRC=REG，DEST=EA

(a) 暫存器和暫存器

例如：SBB AX,BX

SBB CH,DL

(b) 暫存器和記憶體

例如：SBB DX,MEM_WORD

SBB DI,ALPHA[SI]

SBB MEM_WORD,AX

SBB GAMMA[BX][DI],SI

② 機器碼

0001110w	data	data if w=1

如果w=0，則

LSRC=AL，RSRC=data，DEST=AL

如果w=1，則

LSRC=AX，RSRC=data，DEST=AX

(a) 累積器減去直接數值

例如：SBB AL,4

SBB AX,6606

③ 機器碼

100000sw	mod 011 r/m	data	data if sw=01

LSRC=EA，RSRC=data，DEST=EA

(a) 從暫存器中減去直接數值

例如：SBB BX,2001

(b) 從記憶體中減去直接數值

例如：SBB MEM_BYTE,12

SBB GAMMA[DI][BX],1984

被影響的旗號：AF,CF,OF,PF,SF,ZF。

SCAS (scan byte string or scan word string)

SCAS指令和CMP指令一樣，只做相減的動作，但不影響LSRC和RSRC內的值，並且不產生結果，只影響旗號的值。SCAS指令將AL或AX暫存器內的值和額外分段暫存器內的值（以DI暫存器爲位移）做一比較。比較之後，DI暫存器會自動變化一個單位。如果DF=0，則DI暫存器會增加一個單位。如果DF=1，則DI暫存器會減少一個單位。如果w=0，則此一單位爲1。如果w=1，則此一單位爲2。

機器碼

1010111w

如果w=0，則LSRC=AL，RSRC=(DI)，DELTA=1。

如果w=1，則LSRC=AX，RSRC=(DI)+1:(DI)，

DELTA=2。

例如：①

```
CLD
MOV DI,OFFSET DEST_BYTE_STRING
MOV AL,'M'
SCAS DEST_BYTE_STRING
```

②

```
STD
MOV DI,OFFSET WORD_STRING
MOV AX,'MD'
SCAS WORD_STRING
```

被影響的旗號：AF，CF，OF，PF，SF，ZF。

STC (set carry flag)

STC指令使得進位旗號（carry flag, CF）爲1。

(CF)←1

機器碼 | 11111001 |

被影響的旗號：CF。

STD (set direction flag)

STD指令設定方向旗號（direction flag, DF）爲1。

(DF)←1

機器碼 | 11111101 |

被影響的旗號：DF。

STI (set interrupt flag)

STI指令設定中斷旗號（interrupt flag，IF）爲1，如此8088就能處理外界可以遮罩的中斷信號，但必須執行完下一個指令後，才會跳至處理中斷信號的程式去。

(IF)←1

機器碼 | 11111011 |

被影響的旗號：IF。

STOS (store byte string or store word string)

STOS指令將AL或AX暫存器內的值存放到額外分段暫存器內的某一個位置（以DI暫存器內的值爲位移）。執行完STOS指令後，DI暫存器會自動變化一個單位。如果DF=0，則DI暫存器會自動增加一個單位，如果DF＝1，則DI暫存器會自動減少一個單位。如果w＝0，則此一單位爲1，如果w＝1，則此一單位爲2。

(DEST)←(SRC)

如果(DF)＝0，則(DI)←(DI)＋DELTA

如果(DF)＝1，則(DI)←(DI)－DELTA

機器碼 | 1010101w |

如果w＝0，則

SRC＝AL，DEST＝(DI)，DELTA＝1

如果w＝1，則

SRC=AX，DEST=(DI)+1：(DI)，DELTA=2

例如：MOV DI,OFFSET BYTE_DEST_STRING
STOS BYTE_DEST_STRING

被影響的旗號：沒有。

SUB (subtract)

SUB指令將LSRC內的值減去RSRC內的值，然後將結果儲存在DEST內。進位旗號（CF）的值將不會影響SUB指令。SUB指令和SBB指令中，進位旗號（CF）等於0的情況是相等的。

(DEST)←(LSRC)－(RSRC)

① 機器碼 | 001010dw | mod reg r/m |

如果d＝1，則

LSRC=REG，RSRC=EA，DEST=REG

如果d＝0，則

LSRC=EA，RSRC=REG，DEST=EA

(a) 暫存器減去暫存器

例如：SUB AX,BX
SUB CH,DL

(b) 暫存器減去記憶體

例如：SUB DX,MEM_WORD
SUB BL,MEM_BYTE

(c) 記憶體減去暫存器

例如：SUB MEM_WORD,AX
SUB GAMMA[BX][DI],SI

② 機器碼

0010110w	data	data if w=1

如果w = 0，則

LSRC=AL，RSRC=data，DEST=AL

如果w = 1，則

LSRC=AX，RSRC=data，DEST=AX

(a) 累積器減去直接數值

例如：SUB AL,4
SUB AX,9999

③ 機器碼

100000sw	mod 101 r/m	data	data if sw=01

LSRC=EA，RSRC=data，DEST=EA

(a) 暫存器減去直接數值

例如：SUB BX,2001

(b) 記憶體減去直接數值

例如：SUB MEM_BYTE,12
SUB GAMMA[DI][BX],1984

被影響的旗號：AF,CF,OF,PF,SF,ZF。

TEST (test, or logical compare)

TEST指令將LSRC內的值和RSRC 內的值做“交集”(位元對位元)，並且設定(CF)旗號和(OF)旗號爲0。LSRC內的值和RSRC內的值不會改變，TEST指令只影響旗號的值。

(LSRC)&(RSRC)　　“&”表示交集
(CF)←0
(OF)←0

① 機器碼

1000010w	mod reg r/m

LSRC=REG，RSRC=EA

(a) 暫存器和暫存器

例如：TEST AX,DX
TEST SI,BP

(b) 暫存器和記憶體

例如：TEST MEM_WORD,SI
TEST AX,GAMMA[BP][SI]

② 機器碼

1010100w	data	data if w=1

如果w = 0，則LSRC = AL，RSRC = data。

如果w = 1，則LSRC = AX，RSRC = data。

(a) 累積器和直接給定數值

例如：TEST AL,6
TEST AX,999

③ 機器碼

1111011w	mod 000 r/m	data	data if w=1

LSRC=EA，RSRC=data

(a) 暫存器和直接給定數值

例如：TEST BH,7

```
TEST  SI,798
```

(b) 記憶體和直接給定數值

例如：TEST [BP][DI],6ACEH

被影響的旗號：CF,OF,PF,SF,ZF。

不確定的旗號：AF。

XCHG (exchange)

XCHG指令將DEST內的值和SRC內的值互換。

(TEMP)←(DEST)
(DEST)←(SRC)
(SRC)←(TEMP)

① 機器碼 | 10010reg |

SRC=REG，DEST=AX

(a) 累積器和暫存器

例如：
```
XCHG  AX,BX
XCHG  SI,AX
XCHG  CX,AX
```

② 機器碼 | 1000011w | mod reg r/m |

SRC=EA，DEST=REG

(a) 暫存器和記憶體

例如：
```
XCHG  BETA_WORD,CX
XCHG  BX,DELTA_WORD
XCHG  DH,ALPHA_BYTE
```

(b) 暫存器和暫存器

例如：
```
XCHG  BL,AL
```

被影響的旗號：沒有。

XLAT (translate)

XLAT指令以BX暫存器和AL暫存器內的值爲有效位址，找出一個位元組放到AL暫存器內。讀者可以將BX暫存器內的值看成起始位址（starting address），AL暫存器內的值看成間距值（offset），所以XLAT指令通常是用在找尋記憶體中的某一個位元組。

(AL)←((BX)+(AL))

機器碼 | 11010111 |

例如：
```
MOV  BX,OFFSET  TABLE_NAME
XLAT  TABLE_ENTRY
```

被影響的旗號：沒有。

XOR (exclusive or)

XOR指令將LSRC內的值和RSRC內的值做“互斥”。

(DEST)←(LSRC)⊕(RSRC)
(CF)←0
(OF)←0

① 機器碼 | 001100dw | mod reg r/m |

如果d=1，則

LSRC=REG，RSRC=EA，DEST=REG

如果d=0，則

LSRC=EA，RSRC=REG，DEST=EA

(a) 暫存器和暫存器

例如：
```
XOR  AH,BL
XOR  SI,DX
```

(b) 暫存器和記憶體

例如：XOR AX,MEM_WORD
XOR GAMMA[DI],BX

② 機器碼

0011010w	data	data if w=1

如果w＝0，則

LSRC＝AL，RSRC＝data，DEST＝AL

如果w＝1，則

LSRC＝AX，RSRC＝data，DEST＝AX

(a) 累積器和直接數值

例如：XOR AL,11110110B
XOR AX,23F6H

③ 機器碼

1000000w	mod 110 r/m	data	data if w=1

LSRC＝EA，RSRC＝data，DEST＝EA

(a) 暫存器和直接數值

例如：XOR AH,F6H
XOR DI,23F5H

(b) 記憶體和直接數值

例如：XOR MEM_BYTE,3DH
XOR GAMMA[BX][DI],FACEH

被影響的旗號：CF,OF,PF,SF,ZF。

不確定的旗號：AF。

在技術手册裡所列出的BIOS程式是用巨集組合語言的表示法，在這裡我們介紹幾個常用的表示法。

① 我們可以給某個位址一個名字，當使用到這個位址時，只需寫出它的名字，不必寫出實際記憶體的位址。

例如：KB_DATA EQU 60H
KB_CTL EQU 61H

② 我們可以使用DB、DW來預先指定資料的儲存位址。

例如：DATA SEGMENT AT 40H
RS232_BASE DW 4DUP(?)
PRINTER_BASE DW 4DUP(?)
EQUIP_FLAG DW ?
MFG_TST DB ?
MEMORY_SIZE DW ?
IO_RAM_SIZE DW ?

註：DB 表示預先設定1個位元組。
DW 表示預先設定2個位元組（1個字組）。
DW 4DUP(?) 表示預先設定8個位元組（4個字組），並且表示其內的值不確定。
? 表示其內的值不確定。

③ EQU也可以表示某個名字的值。

例如：KB_FLAG DB ?
INS_STATE EQU 80H
CAPS_STATE EQU 40H

註：首先我們設定一個位元組爲資料的儲存位置，並且其內的值不確定。如果其內的值爲80H，就表示KB_FLAG 爲INS_STATE。如果其內的值爲40H，就表示KB_FLAG爲CAPS_STATE。

④ 我們可以在指令的前面寫上一個名字，以方便JMP、

LOOP 等指令的書寫。

例如：
```
        XOR  AX,DI
        JNZ  ERRO1
        JZ   C10
ERRO1:HLT
  C10:MOV  AL,0
      ⋮
      ⋮
```

⑤ 我們可以設定一段完整的程序（procedure）的各個段落的起始位址。

例如：ASSUME CS:CODE,SS:CODE,ES:ABS0
DS:DATA

⑥ 我們也可以設定程序的形式（遠或近），近（near）和遠（far）的區分在於是否爲同段間或異段間的 CALL 指令。我們知道 CALL 指令有異段間和同段間之分，而 RET 指令也有異段間和同段間之分，所以我們在設定程序的形式時必須相當小心。如果某個程序被呼叫的可能都是由同一個段落內的程式而來，我們就可以大膽的設定爲近。如果我們不敢保證只有同一個段落內的程式才會呼叫這個程序，最好將其設定遠。不過在使用 FAR CALL時，必須小心處理，以免不能回到預定的位址。

例如：STGTST PROC NEAR

本章只是大略介紹 8088 的結構、指令和巨集組合語言。筆者希望讀者自行參閱 INTEL 公司的 IAPX 86/88 和 8086/8087/8088 巨集組合語言參考手册，或者是巨集組合程式（MACRO ASSEMBLER）。

第二章

開機後自行測試

在 IBM 技術手册（Technic Manual）中，附錄 A ROM BIOS LISTINGS的位址是間距值（offset），它的段落（segment）值爲 F0000，這是因爲 8088經過重置（RESET）後，程式分段暫存器（CS register）內放 FFFF，IP內放 0000，此時，8088 放出位址 FFFF0，取得一個指令碼（instruction code），這個指令碼爲 EA，所以我們知道 8088 取得的第一個指令爲異段間直接跳躍（inter-segment direct jump）。8088 必須再取 4 個位元組做爲程式分段暫存器和指令指標暫存器（IP register）內的值。在段落是FFFF，間距值是 0000的組合語言爲

JMP RESET（機器碼 EA5BE000F0）

由 JMP 指令，我們知道程式分段暫存器內的值爲 F000，指令指標暫存器內的值爲 E05B，所以開機後自行測試（power-on self-test, POST）是由 RESET 程式開始（位址爲 FE05B）。

POST中分爲若干小的測試程式（Test Routine），在本章的討論裡，我們也以這些小的測試程式爲段落，一段一段地爲讀者介紹。爲了說明的方便，本書中將會把程式重抄一次，並繪成類似流程圖的圖表，請讀者根據圖表旁的數字，配合說明文字，一個步驟一個步驟地討論下去。在每一個圖表的開始會標明間距值，其它的

地方就不標明間距值，也不附上機器碼（machine code）。讀者必須參閱附錄A，才能找到詳細的間距值和機器碼。

TEST. 01 8088測試(8088 PROCESSOR TEST

這個小測試程式主要是測試8088的旗號暫存器和條件跳躍（conditional jumps）是否工作正常。我們先來看它是如何測試旗號和條件跳躍。

E05B

①
```
CLI
MOV     AH, D5H
SAHF
```

②
```
JNC     ERRO1
JNZ     ERRO1
JNP     ERRO1
JNS     ERRO1
```

③
```
LAHF
MOV     CL, 5
SHR     AH, CL
JNC     ERRO1
```

④
```
MOV     AL, 40H
SHL     AL, 1
JNO     ERRO1
```


⑤
```
XOR     AH, AH
SAHF
JC      ERRO1
JZ      ERRO1
JS      ERRO1
JP      ERRO1
```

⑥
```
LAHF
MOV     CL, 5
SHR     AH, CL
JC      ERRO1
```

⑦
```
SHL     AH, 1
JO      ERRO1
```

①首先將IF旗號設定爲0，表示在這個測試中，不允許有外界中斷信號（可遮罩）。在AH暫存器內放入D5，然後使用SAHF指令，強迫SF，ZF，AF，PF，CF爲1，這樣做的原因爲②中將測試條件跳躍和旗號是否能設定爲1（因爲重置後，旗號均爲0）。

②如果CF，ZF，PF，SF旗號，其中任一個爲0的話，就會跳至ERRO1（在OFFSET E0AF處，指令爲HLT）停止系統。如果CF，ZF，PF，SF旗號都爲1，但却跳至ERRO1，則表示條件跳躍不正確。

③LAHF指令將旗號的值放入AH暫存器內（請參閱LAHF和SAHF指令），然後將AH暫存器右移5個位元。如果CF不爲1，則會跳至ERRO1，停止系統。（LAHF後，AH暫存器的位元4

爲1，右移5個位元後，CF內的值，將爲AH暫存器的位元4內的值。）

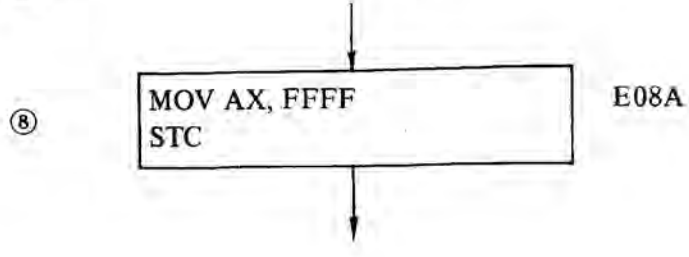
④在AL暫存器內放入40H，然後將AL暫存器左移1個位元，如果OF不爲1，則會跳至ERRO1。我們知道，當左移次數爲1時，如果新的CF旗號不等於新的最高位元值，則OF旗號將爲1。40H經過左移後變成80H，新的CF值（其值爲0）不等於新的最高位元值（其值爲1），所以OF旗號爲1。如果OF旗號不爲1，則表示有錯誤。

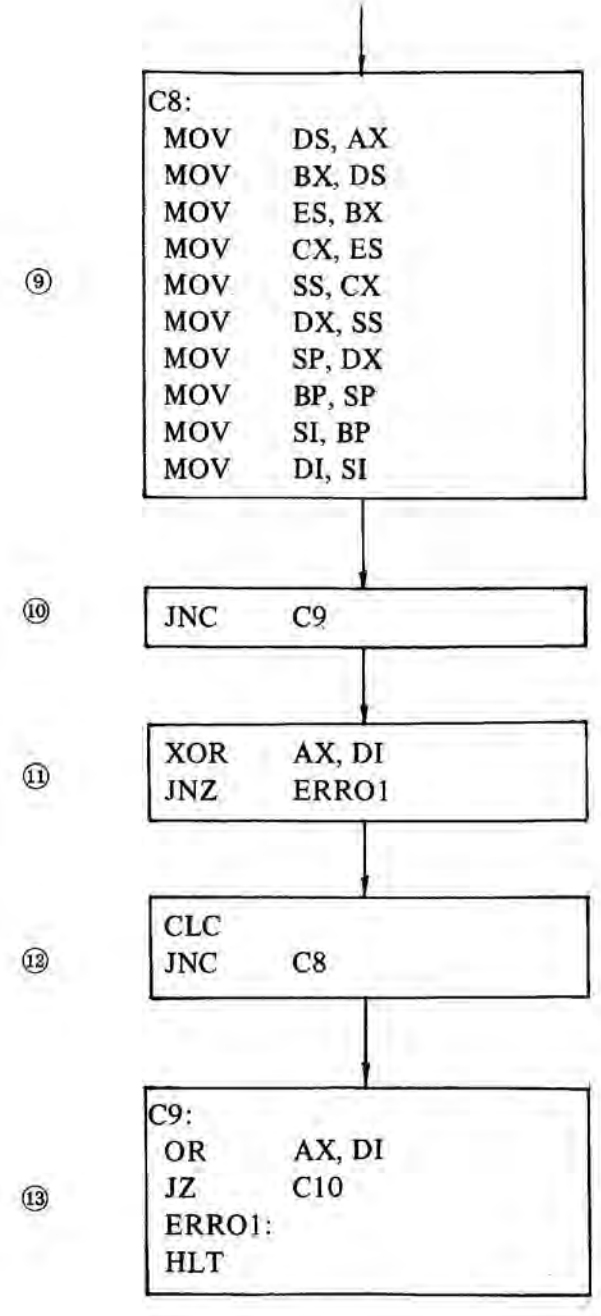
⑤XOR AH,AH指令將AH暫存器清除爲0，SAHF指令強迫CF,ZF,SF,PF旗號爲0。如果CF,SF,ZF,PF旗號，其中任一個爲1，則會跳至ERRO1，停止系統。如果條件跳躍不正確的話，也可能跳至ERRO1。

⑥LAHF指令將旗號的值放入AH暫存器內（此時AH暫存器內的值應該爲00H，然後將AH暫存器右移5個位元。如果CF爲1，則會跳至ERRO1（CF爲1，表示有錯誤）。

⑦執行到這裡，AH暫存器內的值爲00H。AH暫存器右移一個位元後，OF旗號應該爲0（因爲新的CF值等於新的最高位元值）。如果OF旗號爲1，表示有錯誤，會跳至ERRO1，停止系統。

測完旗號後，將繼續測暫存器，先使用FFFF，再使用0000來測試暫存器。





⑧在AX暫存器內放入FFFF，做爲測試資料。將CF旗號設定爲1，是因爲我們要使用⑨二次。第一次用FFFF測試時，CF旗號爲1，第二次用0000測試時，CF旗號爲0，以便做爲跳躍的依據。

⑨將AX暫存器內的值寫入每一個暫存器內。

⑩這裡利用CF旗號的值來決定測試的步驟。如果CF旗號爲1，表示暫存器內的值爲FFFF，要用⑪中的測試方法。如果CF旗號爲0，表示暫存器內的值爲0000，要用⑬中的測試方法。

⑪將AX暫存器內的值（原始值）和DI暫存器內的值（搬移後的最後值）做互斥（XOR）。如果AX和DI暫存器內的值都是FFFF，ZF旗號應爲1，如果不爲1，表示FFFF經過搬移後有錯誤，會跳至ERRO1停止系統。

⑫CLC指令將CF旗號清除爲0，然後跳至⑨中，將0000寫入每個暫存器內。（因爲⑪中XOR AX,DI後，AX暫存器內的值爲0000。）

⑬在用FFFF測試完每個暫存器後，步驟⑫會將CF旗號清除爲0。將0000寫入每個暫存器後，經由步驟⑩的跳躍，進入此步驟做測試的工作。OR AX,DI後，ZF旗號應爲1（因爲AX和DI暫存器內的值應都爲0000）。如果不爲1，就表示有錯誤，則不會跳入C10（C10爲下一個測試程式的進入點），而直接執行HLT指令，停止系統。

我們可以看出，TEST.01測試旗號和暫存器時，使用全部是1和全部是0的資料，這是因爲邏輯的值，不是1就是0，一般的測試都是用這種方式。

TEST.02 BIOS ROM核對和測試 I (ROM CHECKSUM TEST I)

這個測試程式主要是爲了測試8K的BIOS ROM，8K位元組的值加起來，其值要爲0，否則會停止系統。在做測試之前，需做些硬體上的設定。

```
E0B0  C10:
①       MOV     AL, 0
        OUT     A0H, AL
        OUT     83H, AL
②       MOV     AL, 99H
        OUT     CMD_PORT, AL
③       MOV     AL, FCH
        OUT     PORT_B, AL
④       SUB     AL, AL
        MOV     DX, 3D8H
        OUT     DX, AL
⑤       INC     AL
        MOV     DX, 3B8H
        OUT     DX, AL
⑥       MOV     AX, CODE
        MOV     SS, AX
        MOV     BX, E000H
        MOV     SP, OFFSET C1
        JMP     ROS_CHECKSUM
```

①在I/O埠位址A0H處寫入00，表示此時不理會不可遮罩中斷信號，這是因爲在8088的NMI輸入脚處，多加了一個正反器和AND閘。其位址在A0H處，並且由資料滙流排的第8個位元（位元7）來控制。在I/O埠位址A0H處寫入00，則8088的NMI輸入脚一直呈現低電位。在I/O埠位址A0H處寫入80，則8088的NMI輸入脚有可能呈高電位（請參閱技術手册內主機板的綫路圖）。在I/O埠位址83H處寫入00，是爲了定DMA PAGE REGISTER（74LS670）的初值。請查看主機板（system board）綫路圖。需要74LS670做爲DMA PAGE REGISTER的原因爲DMA controller 8237所能控制的位址綫只有16條。

②在I/O埠CMD_Port（位址爲63H）處寫入99H，表示設定8255的A和C埠爲輸入，B埠爲輸出。請參考8255的資料表（data sheet）。

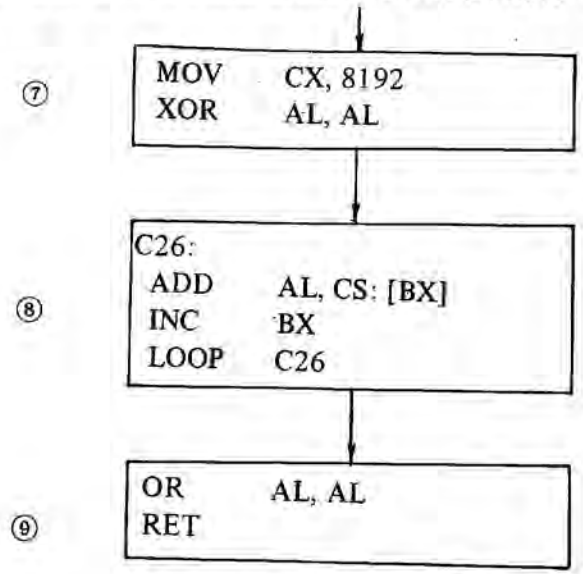
③在I/O埠PORT_B（位址爲61H）處寫入FCH，表示將8255的B埠所控制的功能做一設定（請查看技術手册中8255 I/O位元表）。

④在I/O埠位址3D8H處寫入00，表示關掉彩色訊號，請參考技術手册中彩色圖表界面卡部分。

⑤在I/O埠位址3B8H處寫入01，表示設定高解析度畫面，但此時仍未開啓綠色螢幕訊號，請參閱綠色螢幕界面卡部分。

⑥前面5個步驟是在做硬體上的設定，這個步驟是爲ROS CHECKSUM做準備。在SS內放入F000（選擇碼的值爲F000），是因爲做完ROS CHECKSUM之後要返回TEST.02內。在BX暫存器內放入E000H，是因爲ROM BIOS的起始位址爲FE000。SP暫存器內放入OFFSET C1（其值爲E016），是爲了設定儲存返回位址的位址。在討論ROS CHECKSUM副程式的RET指令時，會將此步驟的SS和SP暫存器內的值做一配合討論。請注意這裡的JMP ROS_CHECKSUM爲同段間直接跳躍（intra segment direct jump）指令，所以不會影響到堆疊（stack）。

我們繼續看ROS_CHECKSUM副程式的動作：



⑦在CX暫存器內放入8192，表示要相加的資料有8K位元組（此8192爲十進位表示法，1K爲1024，8K爲8192），然後將AL暫存器設定爲0。

⑧這個步驟將ROM BIOS所在的8K位元組相加起來（ROM BIOS的位址由FE000到FFFFF）。請注意LOOP指令和CX暫存器內的值的關係。

⑨將AL暫存器內的值做OR，主要的目的爲影響ZF旗號。這裡的RET指令爲同段間（intra-segment）指令，所以它必須在堆疊中取出2個位元組做爲返回位址。在步驟⑥中，我們設定SS暫存器爲F000，SP暫存器爲E016，所以RET指令會在堆疊分段暫存器（stack segment register）爲F000，間距值爲E016內

取出二個位址當做返回位址。在位址FE016處，其內的值爲E0D8。所以做完ROS CHECKSUM後會回到FE0D8處。（同段間的RET指令不會影響CS暫存器內的值）

在位址FE0D8處爲JNE ERRO1 指令。如果在步驟⑨中的OR AL,AL使得ZF旗號不爲1（也就是ROM BIOS 8K位元組加起來其值不爲0），則會跳至ERRO1，停止系統。如果在步驟⑨中的OR AL,AL指令，使得ZF旗號爲1（也就是ROM BIOS 8K位元組加起來其值爲0），則進入下個測試程式。

TEST.03 8237直接記憶體存取控制器及初始化測試(8237 DMA INITIALIZATION CHANNEL REGISTER TEST)

在討論這個測試前，需要將DMA Controller 8237做一說明。

8237有4個獨立的DMA通道（channel），其動作由內部的暫存器控制，它們分別是：

名稱	大小	數目
基準位址暫存器 (BASE address registers)	16位元	4
基準字組數暫存器 (BASE word count registers)	16位元	4
現在位址暫存器 (Current address registers)	16位元	4
現在字組數暫存器 (Current word count registers)	16位元	4
暫時位址暫存器 (Temporary address registers)	16位元	1
暫時字組數暫存器 (Temporary word count registers)	16位元	1
狀態暫存器 (Status register)	8位元	1
命令暫存器 (Command register)	8位元	1
暫時暫存器 (Temporary register)	8位元	1
模式暫存器 (Mode register)	6位元	4
遮罩暫存器 (Mask register)	4位元	1
要求暫存器 (Request register)	4位元	1

①現在位址暫存器（Current address register）：每一個通道都有一個16位元現在位址暫存器，在做DMA時，這個暫存器內放著位址在做完每一個傳送後，這個暫存器會自動增加或減少。8088要讀取或寫入這個暫存器時，必須是8位元的值。當做完整個DMA

傳送後，這個暫存器還可以恢復其原來的值。（自動設定）

②現在字組數暫存器（Current word count register）：每一個通道都有一個16位元現在字組數暫存器，這個暫存器放著要傳送的資料的數目。眞正傳送的數目將會比寫入的數目多1。例如寫入100，則傳送的數目爲101。8088要讀取或寫入這個暫存器時，必須是8位元的值。在做完每一個傳送後，這個暫存器會自動減少。當做完整個DMA傳送後，這個暫存器還可以恢復其原來的值。（自動設定）

③基準位址和字組數暫存器（Base address and Base word count registers）：每一個通道都有一對這個暫存器。這一對暫存器放著現在位址暫存器和現在字組數暫存器的初值。在自動設定（autoinitialize）時，這個原始值將會寫入相對的暫存器（register）。8088在寫入資料於這一對暫存器時，必須是8位元的值。8088無法讀出這一對暫存器內的值。

④命令暫存器（Command register）：這個8位元的暫存器控制著8237的動作。請看圖1命令暫存器各位元的功能。

⑤模式暫存器（Mode register）：每一個通道都有一個模式暫存器。請看圖2模式暫存器各位元的功能。

⑥要求暫存器（Request register）：這個暫存器的功能如硬體上的DMA Request（DREQ）。這個暫存器提供了軟體上的DMA要求（Request），並且這個DMA要求是不可遮罩的（non-maskable）。軟體上的DMA要求必須爲區段模式（block mode）。區段模式的意思爲：8237會一直做傳送工作，直到TC（terminal count）或外界的EOP（End of process）出現爲止。請看圖3要求暫存器各位元的意義。

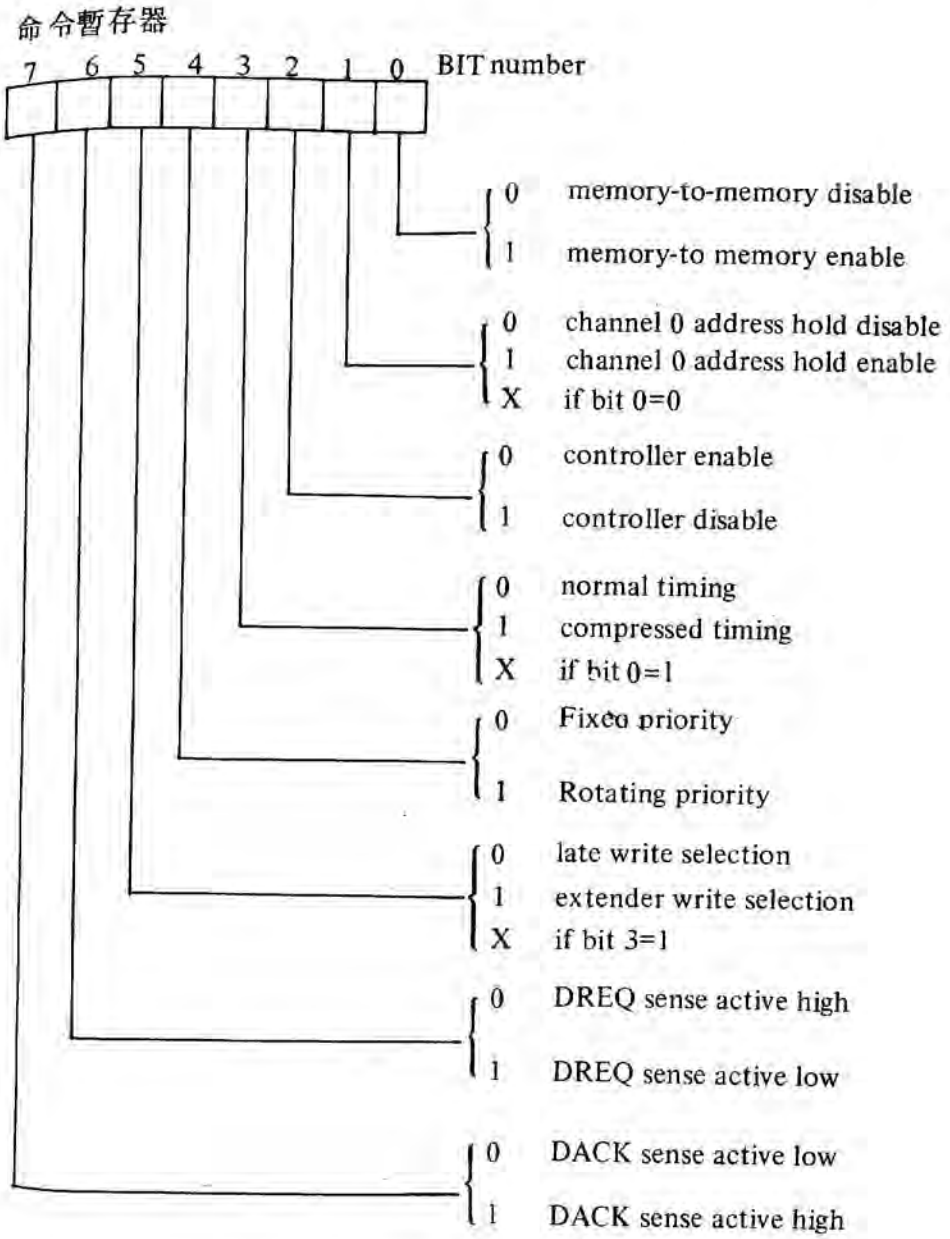


圖1：命令暫存器各位元的功能

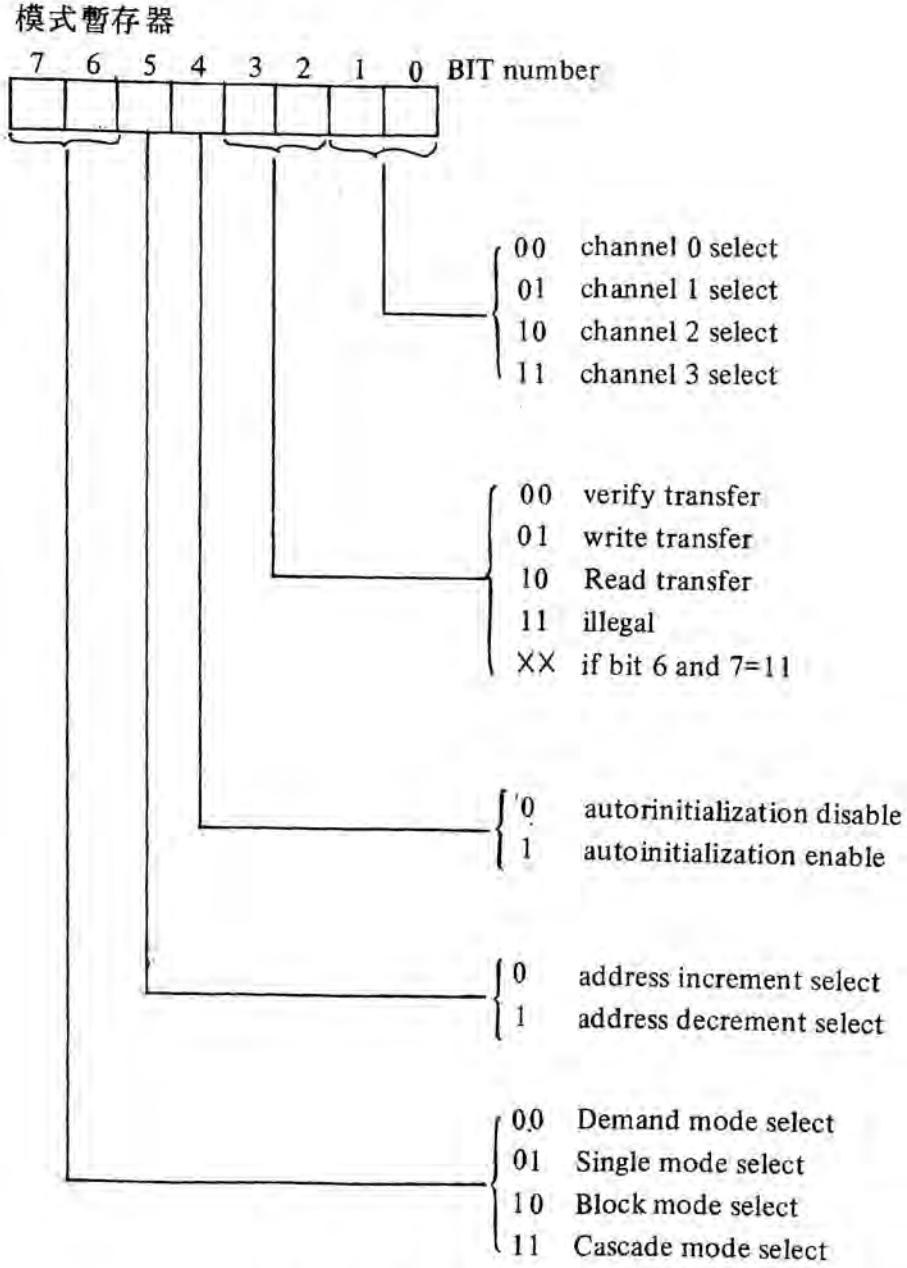


圖 2 ：模式暫存器各位元的功能

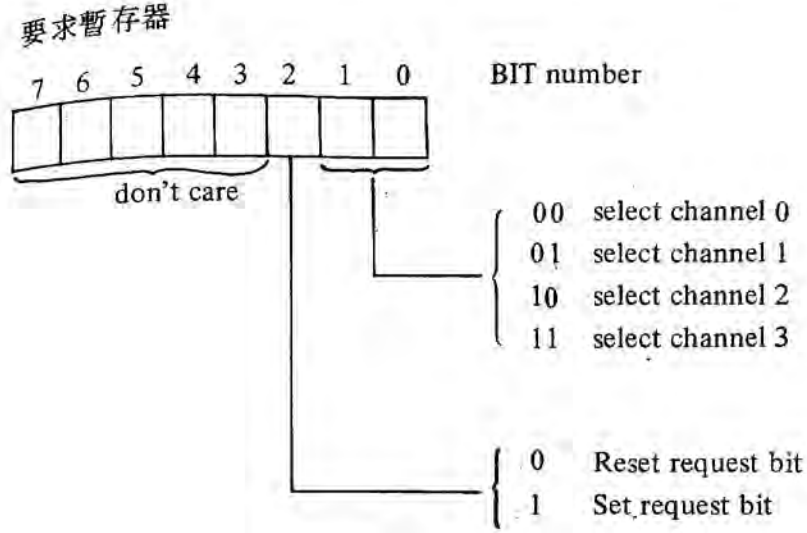
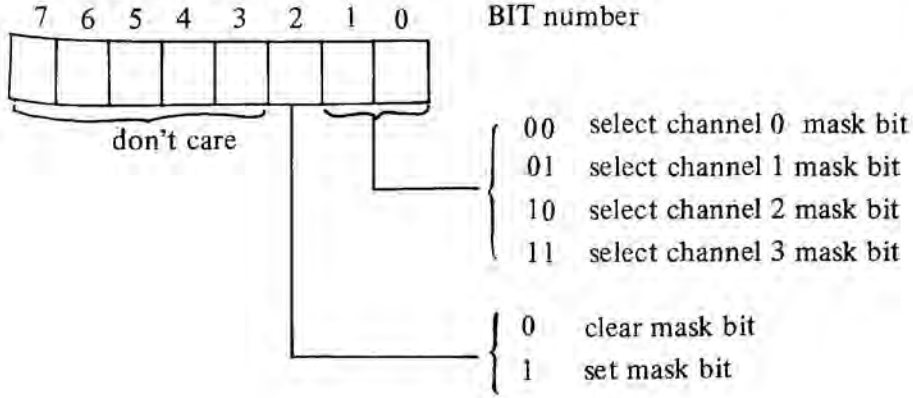
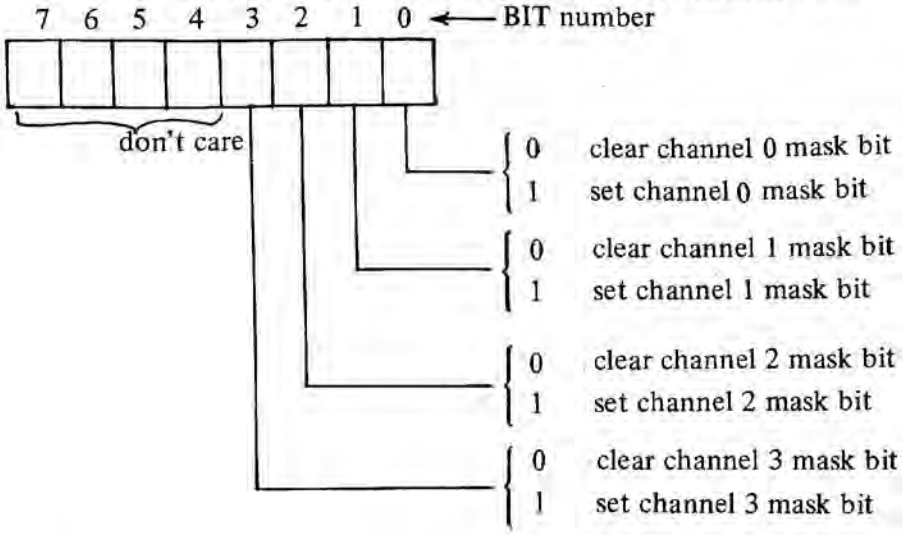


圖 3 ：要求暫存器各位元的意義

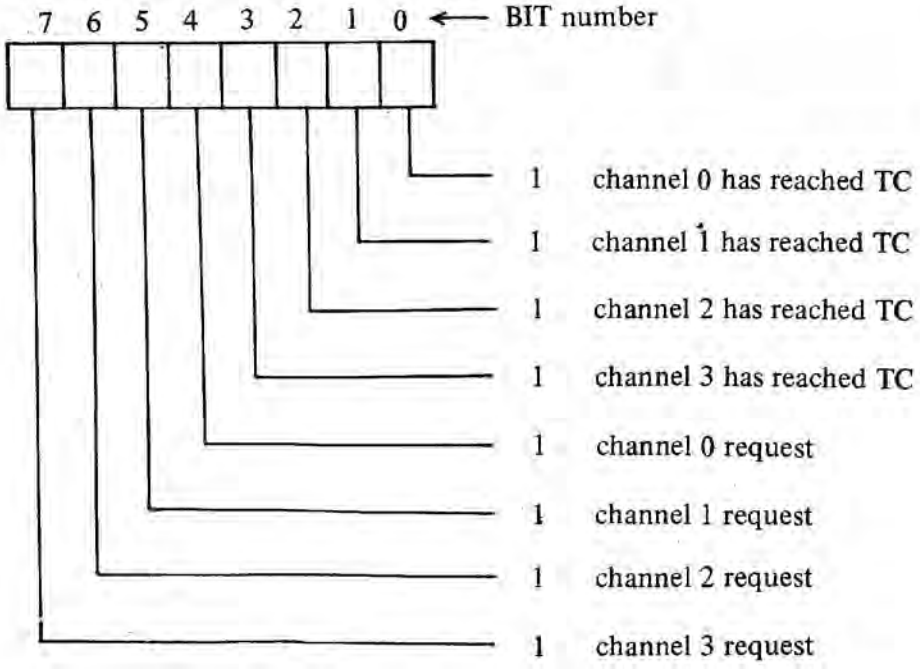
⑦遮罩暫存器（Mask register）：每一個通道都有一個遮罩位元（mask bit），當遮罩位元設定爲 1 時，8237 將不理會相對通道的 DMA 要求訊號（DREQ）。當通道產生 $\overline{EOP}$ 時，其相對的遮罩位元將爲 1（此時這個通道必不在自動設定模式中）。4 個遮罩位元也可用軟體來各別控制。請看下圖控制遮罩位元的軟體寫法（選擇位址爲 1010）。



也可以用一個指令同時控制 4 個遮罩位元（選擇位址爲 1111）。



⑧狀態暫存器（Status register）：這個暫存器記錄著現在 8237 的狀態，它可以被 8088 讀取。請看下圖狀態暫存器各位元的意義。



⑨暫時暫存器（Temporary register）：在記憶體到記憶體傳送（memory-to-memory transfer）時，暫時暫存器做爲資料暫時存放的地方。

⑩軟體命令（Software commands）：下面三個軟體指令也控制 8237 的動作。

(a)清除先後正反器（clear First/Last Flip-Flop）：在讀取或寫入位址暫存器或字組數暫存器之前，必須先執行這一指令，如此 8237 才能以正反器（Flip-Flop）爲根據做正確的資料處理（高位元組或低位元組），這是因爲這些暫存器一次只能做 8 位元的資料處理。

(b)重置（Master clear）：這個軟體指令就如硬體上的重置（RESET）訊號，此時命令、狀態、要求、暫時和先後正反器暫存器被設定爲 0，而遮罩暫存器被設定爲 1。

(c)清除遮罩暫存器（Clear Mask Register）：這個軟體指令將每個通道的遮罩位元設定爲 0，使得它們能夠接收 DMA 要求。

下表列出暫存器和位址選擇的關係

A3	A2	A1	$\overline{A0}$	$\overline{IOR}$	$\overline{IOW}$	operation
1	0	0	0	0	1	Read status register
1	0	0	0	1	0	Write command register
1	0	0	1	1	0	Write Request register
1	0	1	0	1	0	Write single Mask Register bit
1	0	1	1	1	0	Write Mode Register
1	1	0	0	1	0	Clear Byte pointer Flip/Flop
1	1	0	1	0	1	Read Temporary register
1	1	0	1	1	0	Master clear
1	1	1	0	1	0	Clear Mask Register
1	1	1	1	1	0	Write all Mask Register Bits

表一：暫存器和位址選擇的關係

下表列出字組數和位址暫存器位址選擇碼：

Channel	Register	Operation	$\overline{CS}$ $\overline{IOR}$ $\overline{IOW}$ A3 A2 A1 A0	Flip-Flop	data bus DB0-DB7
0	Base and current address	Write	0 1 0 0 0 0 0	0	A0-A7
			0 1 0 0 0 0 0	1	A8-A15
	Current address	Read	0 0 1 0 0 0 0	0	A0-A7
			0 0 1 0 0 0 0	1	A8-A15
	Base and current word count	Write	0 1 0 0 0 0 1	0	W0-W7
			0 1 0 0 0 0 1	1	W8-W15
	Current word count	Read	0 0 1 0 0 0 1	0	W0-W7
			0 0 1 0 0 0 1	1	W8-W15
1	Base and current address	Write	0 1 0 0 0 1 0	0	A0-A7
			0 1 0 0 0 1 0	1	A8-A15
	Current address	Read	0 0 1 0 0 1 0	0	A0-A7
			0 0 1 0 0 1 0	1	A8-A15
	Base and current word count	Write	0 1 0 0 0 1 1	0	W0-W7
			0 1 0 0 0 1 1	1	W8-W15
	Current word count	Read	0 0 1 0 0 1 1	0	W0-W7
			0 0 1 0 0 1 1	1	W8-W15
2	Base and current address	Write	0 1 0 0 1 0 0	0	A0-A7
			0 1 0 0 1 0 0	1	A8-A15
	Current Address	Read	0 0 1 0 1 0 0	0	A0-A7
			0 0 1 0 1 0 0	1	A8-A15
	Base and current word count	Write	0 1 0 0 1 0 1	0	W0-W7
			0 1 0 0 1 0 1	1	W8-W15
	Current word count	Read	0 0 1 0 1 0 1	0	W0-W7
			0 0 1 0 1 0 1	1	W8-W15
3	Base and current address	Write	0 1 0 0 1 1 0	0	A0-A7
			0 1 0 0 1 1 0	1	A8-A15
	Current Address	Read	0 0 1 0 1 1 0	0	A0-A7
			0 0 1 0 1 1 0	1	A8-A15
	Base and current word count	Write	0 1 0 0 1 1 1	0	W0-W7
			0 1 0 0 1 1 1	1	W8-W15
	Current word count	Read	0 0 1 0 1 1 1	0	W0-W7
			0 0 1 0 1 1 1	1	W8-W15

表二： Word Count and Address Register Command Codes

這裡所介紹的只是8237各個暫存器和軟體指令。其它如腳位、DC和AC特性，請讀者自行參考8237資料表。和DMA動作息息相關的IC爲74LS670，它也是一個暫存器（4字組×4位元），4個輸出腳接至4個高位址。這是因爲8237只能控制16條位址綫，所以需要74LS670的輔助。我們稱這個74LS670爲DMA PAGE REGISTER。

我們知道IBM PC的dynamic RAM REFRESH動作由8253 Timer的通道1配合8237 DMA控制器的通道1控制著，所以這裡不能不對8253 Timer做一說明。

8253 Timer有3個獨立的16位元計數器（counter），每個計數器都是向下算。請看下表8088讀取或寫入各個計數器的選擇碼。

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A1	A0	operation		
0	1	0	0	0	Load	counter	No. 0
0	1	0	0	1	Load	counter	No. 1
0	1	0	1	0	Load	counter	No. 2
0	1	0	1	1	Write	mode	word
0	0	1	0	0	Read	counter	No. 0
0	0	1	0	1	Read	counter	No. 1
0	0	1	1	0	Read	counter	No. 2

最值得注意的是A1，A0爲11時的寫入模式字組（write mode word），因爲它實質地控制著各個計數器的動作。寫入模式字組的資料格式如下：

D_7	D_6	D_5	D_4	D_2	D_2	D_1	D_0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

它們的意義分述於下：

SC–Select Counter

SC1	SC0	operation
0	0	sclect counter 0
0	1	select counter 1
1	0	select counter 2
1	1	illegal

RL–Read/Load

RL1	RL0	operation
0	0	Counter latching operation
1	0	Read/Load most significant byte only
0	1	Read/Load least significant byte only
1	1	Read/Load least significant byte first, then most significant byte

M–MODE

M2	M1	M0	operation
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD

BCD	operation
0	Binary Counter 16–BITS
1	Binary coded Decimal (BCD) counter

每個計數器可能爲6個模式（mode）中的其中一種。我們繼續討論各個模式的動作情形。

mode 0：interrupt on teminal count。輸出爲低電位，直到計數器計數到0才升至高電位。

mode 1：programmable one-shot。當GATE輸入脚由低電位變成高電位時，輸出脚呈低電位，直到計數器算到0才變成高電位。

mode 2：Rate generator（Divide by N counter ）。輸出脚保持高電位，直到計數器算到0才呈低電位，這個低電位持續一個CLOCK的時間，然後輸出脚又呈高電位。輸出脚恢復至高電位的同時，計數器會自動計數，當算至0時，輸出脚再次呈低電位；如此地重複下去。

mode 3：square wave Rate generator。mode 3和mode 2相似，不同的是高電位和低電位各占一半的時間。

mode 4：software Triggered strobe 。輸出脚保持高電位，直到計數器算到0才變成低電位，此低電位持續一個CLOCK的時間，然後又恢復高電位。

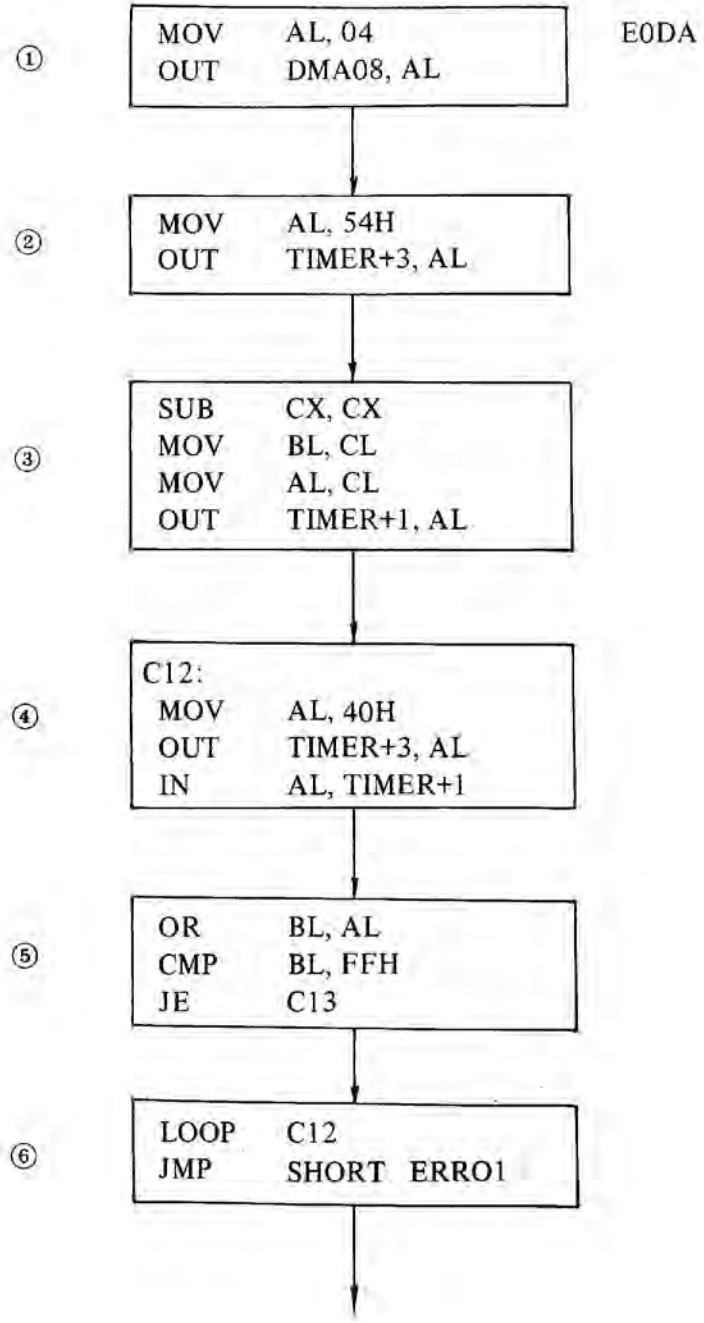
mode 5：Hardware Triggered strobe。當GATE 輸入脚由低電位變成高電位時，計數器開始算，此時輸出呈高電位直到計數器算至0才呈低電位，此低電位持續一個CLOCK的時間，然後又恢復高電位。

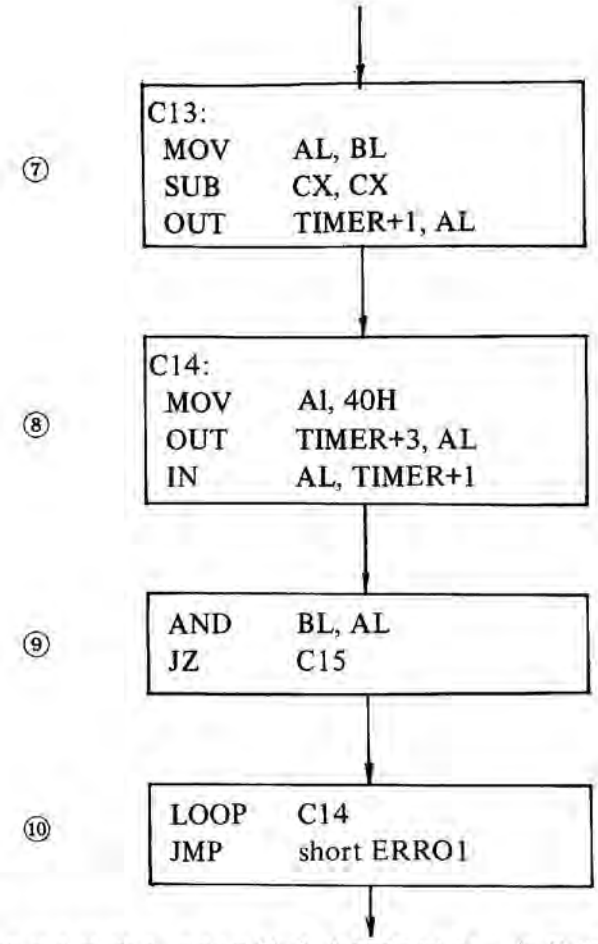
若要讀計數器內的計數，我們需用 latch operation。整個步驟是這樣的：

①寫入模式字組（A1A0=11），D7和D6（SC1和SC0）選擇計數器，D5和D4（RL1和RL0）爲00，如此即可 latch 住計數器內的值。（計數器會繼續算，不受影響）。

②使用Read指令，將此值讀出。

好了，我們已有相當的準備來討論TEST.03程式了。





①將04H寫入8237的命令暫存器內，如此即disable 8237。

②從此步驟到步驟⑩爲測試8253的counter 1是否工作正常。首先將54H寫入8253的模式字組暫存器（mode word register），如此即設定計數器1爲mode 2（Rate generator），而且只設定寫入低位元組（READ/LOAD Least Byte）。

③將CX，BL，AL暫存器清除爲0，並且在計數器1內寫入0（此即計數器1的計數）。將CX暫存器設定爲0，是爲了配合步驟⑥中的LOOP指令（請注意LOOP指令先將CX暫存器內的值減1，然後再決定是否要跳躍。0減1的結果爲FFFF）。將BL暫

存器清除爲 0 是爲了在步驟⑤中測試計數器 1 。將 AL 暫存器設定爲 0 是爲了設定計數器 1 的計數爲 0 (計數爲 0 造成FULL count, 即計算 2^8 或 2^{16} 次後才會產生低電位 (mode 2))。

④在 8253 的模式字組暫存器內寫入 40 H , latch 住計數器 1 瞬間的計數值，然後將此值取出。

⑤此步驟爲測試 8253 的計數器 1內各位元是否能成爲 1 ， OR BL ,AL 是爲了記錄成爲 1 的位元。例如：如果由計數器 1 latch 出來的值爲 01000001 ，則表示計數器內的位元 6 和位元 0 能成爲 1 。OR BL ,AL 後， BL 暫存器內的值爲 01000001 ，如此即將位元 6 和位元 0 記錄下來。CMP BL ,FFH 是爲了測試是否每一個位元都能成爲 1 。如果相等 (即表示計數器 1 內各位元都能成爲 1) ，就跳至步驟⑦做其他的測試，若不相等，則進入步驟⑥。

⑥ LOOP指令使得 IP 暫存器回到步驟④，繼續 latch 計數器 1 內的瞬間值，步驟⑤記錄能成爲 1 的位元，直到計數器 1 內各位元都能成爲 1 爲止才跳至步驟⑦。如果步驟④和⑤重複了 FFFFH 次 (CX暫存器初值爲 0) ，仍不能使得計數器 1 內各位元成爲 1 ，那我們就認爲 8253不良，會跳至 ERRO1 停止系統。

⑦進入此步驟的 BL暫存器，其值爲 FFH。寫 FFH值到8253 的計數器 1 內做爲計數。將 CX 暫存器清除爲 0 是爲了配合步驟⑩的 LOOP指令。

⑧ Latch計數器 1 內瞬間的值，然後讀出。

⑨ AND BL ,AL 是爲了記錄已成爲 0 的位元。如果 BL 暫存器內的值爲 0 (即表示計數器 1 內各位元能成爲 0) ，就會跳至 C15 執行其他程式，否則進入步驟⑩。

⑩ LOOP指令重複步驟⑧和⑨。如果重複了 FFFFH 次後仍不能使得計數器 1 內各位元成爲 0 的話，則我們認爲 8253 不良，會跳至 ERRO1 停止系統。

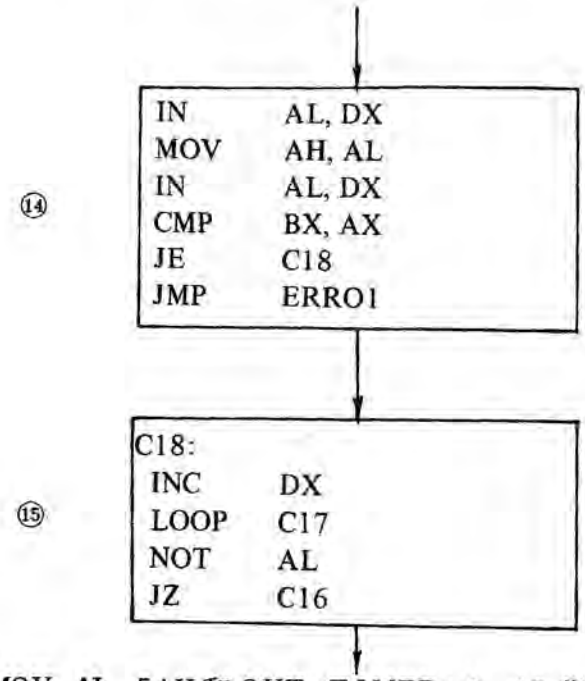
由上面的討論，我們可以看出，因爲此小段程式是爲了測試 8253 的計數器 1 是否計數正常，並非做 dynamic RAM的REFRESH動作，所以首先要關掉 8237。測試 8253 的計數器 1 的方法爲先在計數器內寫入 00H (此時 8253 的計數器 1 會向下計數) ，然後隨機讀出瞬間值，只要各位元能成爲 1 ，即認爲可能是好的。接著在計數器內寫入 FFH，然後隨機讀出瞬間值，只要各位元能成爲 0 ，即認爲 8253 的計數器 1 是好的。

我們繼續看 TEST . 03 測試程式。

⑪
```
C15:
 MOV    AL, 54H
 OUT    TIMER+3, AL
 MOV    AL, 18
 OUT    TIMER+1, AL
 OUT    DMA+0DH, AL
```

⑫
```
 MOV    AL, FFH
C16:
 MOV    BL, AL
 MOV    BH, AL
 MOV    CX, 8
 MOV    DX, DMA
```

⑬
```
C17:
 OUT    DX, AL
 OUT    DX, AL
 MOV    AX, 0101H
```



⑪MOV AL,54H和OUT TIMER+3,AL指令選擇8253的計數器1爲mode 2，並且只設定寫入低位元組而已。MOV AL,18和OUT TIMER+1,AL 指令，設定計數器1的計數爲18。此時計數器1的輸出脚每18個CLOCK就有一個CLOCK的低電位。這裡所言的CLOCK爲8253的CLOCK。OUT DMA+0DH,AL造成8237的重置（重置的指令與資料滙流排（data bus）無關，只要是位址滙流排（address bus）正確而且是I/O write即可），此時8237的命令（Command）、狀態（Status）、要求（Request）、暫時（Temporary）和先後正反器（internal First/Last Flip-Flop）暫存器會被設定爲0，而遮罩暫存器（Mask Register）會被設定爲1。最值得注意的是遮罩暫存器，因爲此時8253的計數器1的輸出脚每18個CLOCK就有一個CLOCK的低電位，這個低電位經過電路上的處理（一個D型正反器）會成爲DMA要求訊號，但因8237的遮罩暫存器內每個遮罩位元爲1，所以8237不理會這個要求訊號。這樣做的原因是，當尚未確定8237是良好的之前，是不能開始dynamic RAM的REFRESH動作。（步驟⑫到⑮將會測試8237）

⑫將AL,BL,BH暫存器內填入FFH，在AL暫存器內填入FFH是爲了測試8237內的基準和現在位址以及基準和現在字組數暫存器。BL和BH暫存器內填入FFH是做爲測試的依據。在CX暫存器內寫入8，是因爲要測試8個暫存器（基準位址暫存器（Base address register）和現在位址暫存器（Current address register）是同時寫入，但基準位址暫存器不能被讀取。基準字組數暫存器（Base word count register）和現在字組數暫存器(Current word count register)是同時寫入，但基準字組數暫存器不能被讀出）。在DX暫存器內放入DMA（00H）是因爲8個暫存器的起始位址在00H。

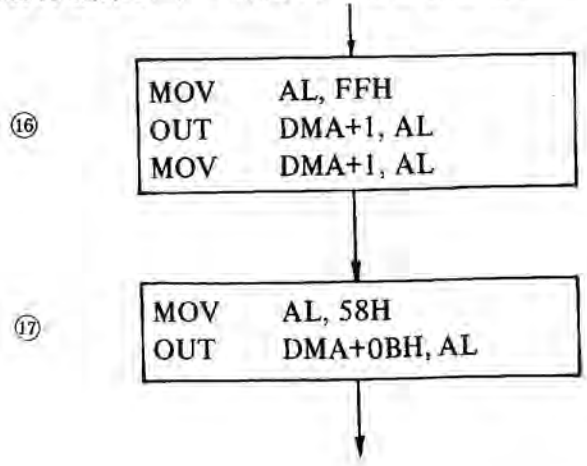
⑬這個步驟和步驟⑭、⑮是爲了測試8237內的基準和現在位址暫存器（Base and current address register）和基準和現在字組數暫存器（Base and current word count register）。先在第1個暫存器內寫入FFFFH，然後讀出，並且測試是否和寫入的值相同，若相同，則繼續測試下一個暫存器，若不相同，則跳至ERRO1停止系統。若8個暫存器用FFFFH測試正確後，再用0000H測試這8個暫存器是否正確。這步驟先將要測試的暫存器填入AL暫存器內的值（FFH或00H），值得注意的是，8237一次只接受8.位元的資料。然後將AX暫存器內的值換成不完全是0和不完全是1的資料，這樣做的原因是，從暫存器內讀出來的值要放到AX暫存器內（寫入暫存器內的值也是由AL暫存器供給）。

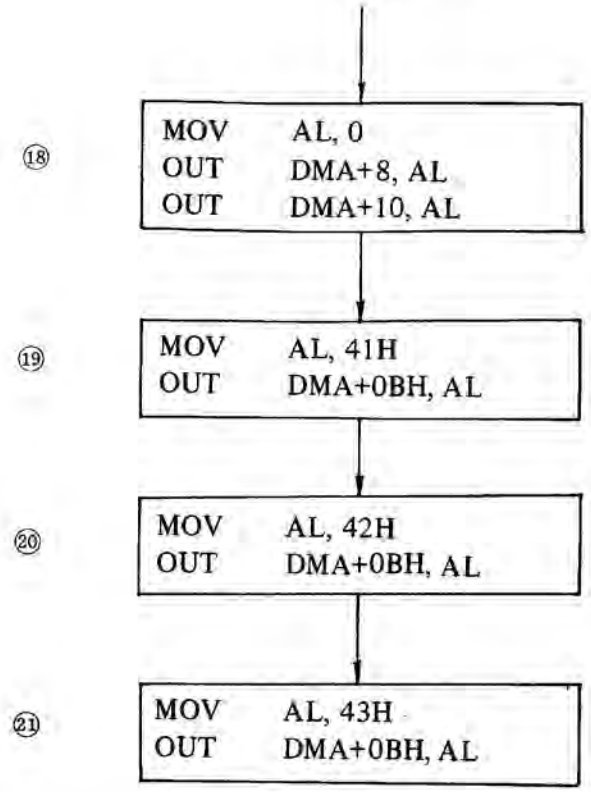
⑭將寫入的值讀出，然後和BX暫存器內的值比較，如果相等，則進入⑮測試下個暫存器，如果不相等，則跳至ERRO1停止系統。

⑮ INC DX 是爲了測試下一個暫存器。LOOP 指令和 CX 暫存器內的值相互配合。如果 CX 暫存器內的值不爲 0，表示尚未測完 8 個暫存器。如果 CX 暫存器內的值爲 0，表示已測完了 8 個暫存器，不再做跳躍動作，而執行下個指令。NOT AL指令和 JZ C16相配合。我們先以 FFFFH來測試暫存器，所以AL 暫存器內的值爲 FFH，NOT AL後，AL 暫存器內的值成爲 00H，然後跳至步驟⑫將 BL，BH暫存器設定爲 00H，以便做爲測試的依據。此時會重複步驟⑬和⑭。當用 00H測試完 8 個暫存器後，NOT AL 將 AL 暫存器內的值變成 FFH，此時就不再跳至步驟⑫，而會執行下面的指令。

由上面的討論我們可以看出，這段程式主要是爲了測試 8237 內的現在位址暫存器和現在字組數暫存器，並非執行 REFRESH 動作，所以步驟⑪的重置指令使得 8237 不理會 DMA 的要求訊號。測試 8237 的暫存器的方法爲先用 FFH 寫入，然後讀出，檢查是否相等，如果不相等表示此暫存器不良。接著用 00H寫入，然後讀出，檢查是否相等，相等才表示良好。

既然 8253的計數器 1 和 8237 的暫存器是良好的（如果任一個不良就會停止系統），那麼就可以開始REFRESH的動作了。





⑯將 FFFFH寫入 8237的通道 0的基準和現在字組數暫存器中。（基準和現在位址暫存器其內的值爲 0000H）

⑰將 58H寫入 8237的模式暫存器（Mode Register）中，表示我們選擇通道 0爲讀取資料、自動設定、位址遞增和單一模式。

⑱將 00H寫入 8237的命令暫存器中，表示我們選擇正常時序（normal timing），固定優先權（Fixed priority），DREQ高態動作（DREQ sense active high），DACK低態動作(DACK sense active low），並且開啓 8237 DMA 控制器，到目前爲止，8237 尚不能接受DERQ 訊號，因爲每個通道的遮罩位元都是 1。OUT DMA+10，AL 指令爲寫入單一遮罩位元，使得 8237 的通道 0的遮罩位元爲 0，換句話說 8237 的通道 0 可以接受DREQ訊號。此時

，REFRESH的動作已經設定好了，剩下來的是硬體上的工作了。讀者也許會問，8237的基準和現在位址暫存器與基準和現在字組數暫存器都是64K容量，如果機器上的記憶體容量超過64K的話，該怎麼辦？不必擔心，因爲機器上的REFRESH動作是每一個BANK同時動作（一個BANK是64K位元組），所以64K的字組數是足夠的（請參考線路圖上的REFRESH部分）。

⑲設定8237的通道1爲查驗傳輸（verify transfer），關閉自動設定位址遞增，單一模式。

⑳設定8237的通道2爲查驗傳輸，關閉自動設定，位址遞增，單一模式。

㉑設定8237的通道3爲查驗傳輸、關閉自動設定、位址遞增、單一模式。必須注意的是，8237的通道1，2，3它們的遮罩位元還是1。筆者建議讀者將8237 DMA控制器的資料表詳細的研讀一番，不僅能對此段程式有更踏實的瞭解，也能對8237 DMA控制器和其程式設計有所幫助。

TEST. 04 基礎16K記憶體測試(BASE 16K READ/WRITE STORAGE TEST)

這個測試程式主要是用FF，AA，55，01，00來測試最低位址的16K記憶體。

①
```
MOV     AX, DATA
MOV     DS, AX
MOV     BX, RESET_FLAG
```

②
```
SUB     AX, AX
MOV     ES, AX
MOV     DS, AX
SUB     DI, DI
```

③
```
IN      AL, PORT_A
AND     AL, 0CH
```

④
```
ADD     AL, 4
MOV     CL, 12
SHL     AX, CL
MOV     CX, AX
MOV     AH, AL
```

⑤
```
CLD
C19:
STOSB
LOOP    C19
```

①此步驟爲將RESET_FLAG保存在BX暫存器內，因爲我們要將主機板上的記憶體填入00H（最低位址的16K位元組）。RESET_FLAG是判別溫起動（warm start）和冷起動（cold start）的依據，它的位址在segment = 40H，OFFSET=0072H的地方。

②將AX，ES，DS，DI暫存器清除爲0000H。ES和DI暫存器

ES和DI暫存器是爲了配合步驟⑤。

③ IN AL,PORT_A指令讀DIP開關（switch）1的值（ON爲0，OFF爲1）。AND AL,0CH取出DIP開關1的位元2和位元3的值（位元2和位元3決定基準記憶體（BASE MEMORY）的大小）。請看下表位元2和位元3所對應的基準記憶體的大小值。

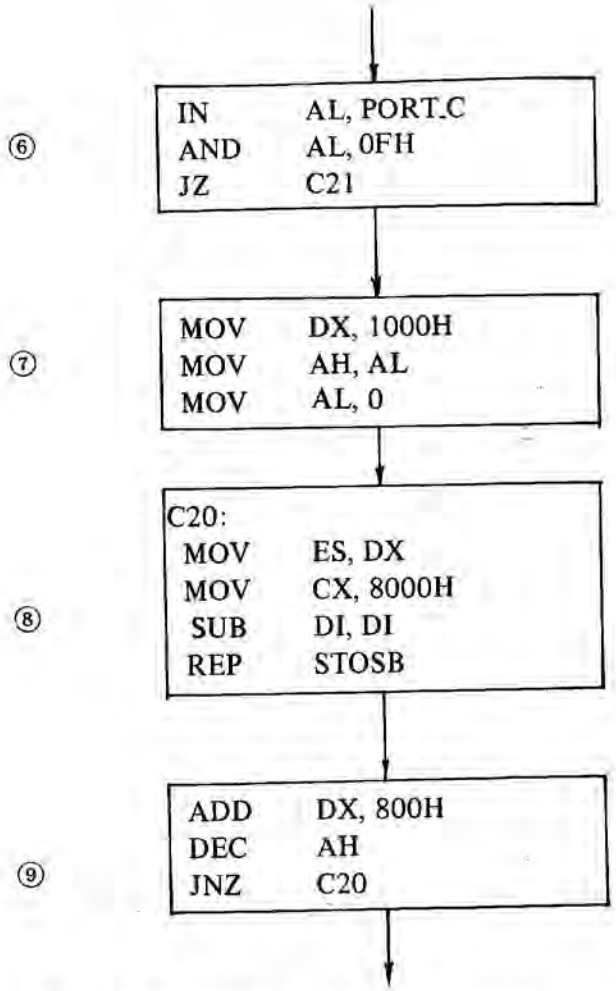
位元3	位元2	記憶體大小
0	0	16K
0	1	32K
1	0	48K
1	1	64K

④前面三個指令爲將位元3和位元2轉換成實際的大小。例如：位元3和位元2都爲0，加上4H後，變成04H，左移12次後變成4000H，即爲16K。若位元3和位元2都爲1。加上4H後爲10H，左移12次後變成10000H，即爲64K，然後保存在CX暫存器內，這樣做的原因爲配合步驟⑤的LOOP指令。

⑤ CLD指令首先設定DF旗號爲0，表示STOSB指令爲向上增加。STOSB和LOOP C19二個指令將位元3和位元2所表示的記憶體大小，全部填入00H。請參閱STOSB和LOOP指令，在步驟④中的左移指令將AL暫存器清除爲00H。

我們已將基準記憶體填上00H，現在要測試是否有其他的記憶體，如果有的話，也要填入00H。我們知道IBM PC有二種型式，一種是主機板上的記憶體由16K到64K（memory IC爲16K×1，共有4個BANK）；另一種是主機板上的記憶體由64K到256K（記憶體IC爲64K×1，共有4個BANK）。我們所謂的基準記憶體就是基本的64K記憶體（主機板爲64K到256K），或者是16K、32K、48K、64K的任一種（主機板爲16K到64K）。基準記憶體由DIP開關1的位元3和位元2來決定。更大的記憶體容量由DIP開關2的位元0到位元3決定。要注意的是，主機板上的4個BANK都插上記憶體IC後，才能插記憶體擴充卡。下面的這段小程式就是將基準記憶體以外的記憶體填入00H，我們先看DIP開關2的位元0到位元3和記憶體大小的關係。

位元3	位元2	位元1	位元0	記憶體大小
0	0	0	0	64K或64K以下
0	0	0	1	96K
0	0	1	0	128K
0	0	1	1	160K
0	1	0	0	192K
0	1	0	1	224K
0	1	1	0	256K
0	1	1	1	288K
1	0	0	0	320K
1	0	0	1	352K
1	0	1	0	384K
1	0	1	1	416K
1	1	0	0	448K
1	1	0	1	480K
1	1	1	0	512K
1	1	1	1	544K



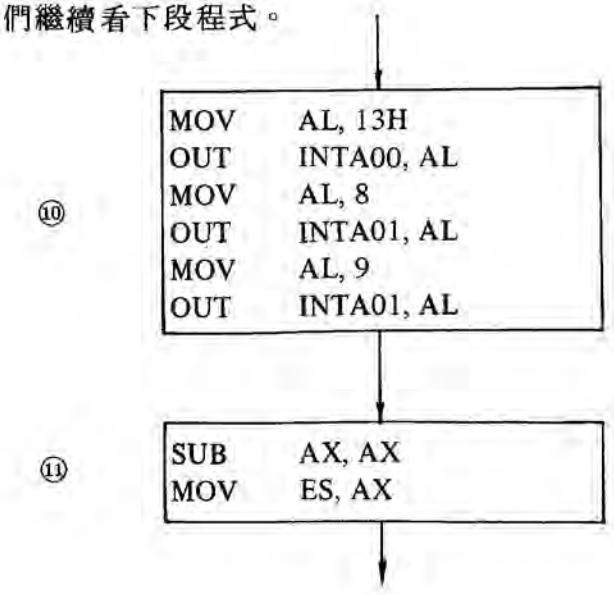
⑥讀DIP開關2的位元0到位元3的值，如果爲0，就表示除了基準記憶體外，沒有其他的記憶體，此時要跳至下一段程式。

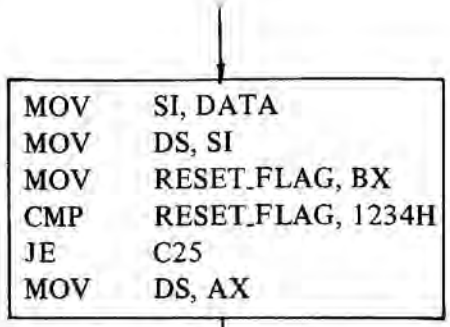
⑦在DX暫存器內放入1000H，以便在步驟⑧中轉入ES暫存器。這是因爲如果是64K到256K的主機板，第1個BANK(最低的64K)已被認爲是基準記憶體。如果是16K到64K的主機板，必須要插滿4個BANK，才能插記憶體擴充板，當4個BANK都插滿時，這64K已被認爲是基準記憶體，所以要填入除了基準記憶體以外的記憶體，要從位址64K開始(10000H爲64K)。然後將由DIP開關2讀到的值保存在AH暫存器，以便在步驟⑨中以AH暫存器做爲跳躍的依據。再將AL暫存器清除爲00H，以便填入所有記憶體。

⑧先將ES暫存器設定爲1000H。CX暫存器內放入8000H，表示一次填32K。將DI暫存器清除爲0000H（別忘了此時DF旗號爲0）。對ES,CX,DI暫存器的設定是爲了配合下面的二個指令(REP和STOSB)。請讀者參閱REP和STOSB指令，就可以明白此步驟的意義。

⑨將DX暫存器內的值加上800H，也就是將實際位址暫存器調高32K。我們說過AH暫存器內的值(即由DIP開關2讀到的值)即表示32K段落的數目，也就是說，將AH暫存器內的值乘以32K，所得到的值就是除了基準記憶體以外的記憶體容量。所以這裡以AH暫存器內的值來決定是否要回到步驟⑧繼續填入下個32K段落。

我們繼續看下段程式。





⑫

```
MOV     SI, DATA
MOV     DS, SI
MOV     RESET_FLAG, BX
CMP     RESET_FLAG, 1234H
JE      C25
MOV     DS, AX
```

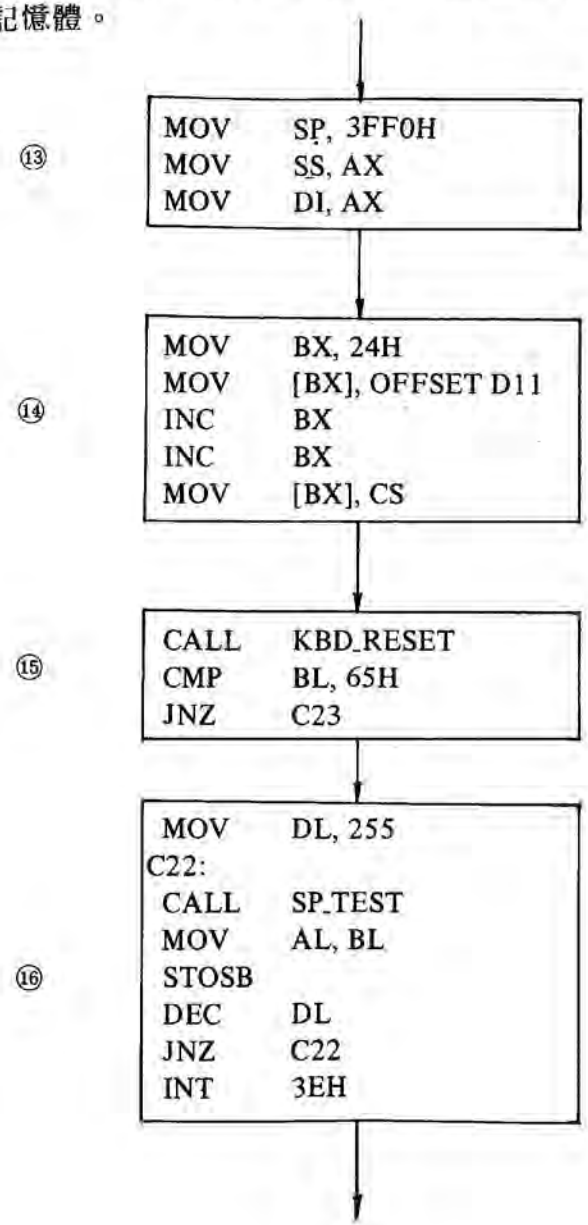
⑩這個步驟將8259中斷控制器（interrupt controller）設定爲單一模式，邊緣觸發模式（edge triggered mode），緩衝8088模式（buffered 8088 mode）。請讀者自行參考8259的資料表。並且設定IRQ的向量（vector）值，請看下表。

向量值

	D7	D6	D5	D4	D3	D2	D1	D0
IRQ7	0	0	0	0	1	1	1	1
IRQ6	0	0	0	0	1	1	1	0
IRQ5	0	0	0	0	1	1	0	1
IRQ4	0	0	0	0	1	1	0	0
IRQ3	0	0	0	0	1	0	1	1
IRQ2	0	0	0	0	1	0	1	0
IRQ1	0	0	0	0	1	0	0	1
IRQ0	0	0	0	0	1	0	0	0

⑪先將ES暫存器設定爲0000H，以便在下個測試記憶體的程式中使用。

⑫將原本保存在BX暫存器內的RESET_FLAG放回原處，然後檢查其值是否爲1234H，若是，則認爲是溫起動（warm start），不執行記憶體測試，直接跳至C25。若RESET_FLAG的值不爲1234H，則認爲是冷起動（cold start），要執行記憶體測試。先將DS暫存器設定爲0000H，以便指向最低位址的16K位元組記憶體。



⑬

```
MOV     SP, 3FF0H
MOV     SS, AX
MOV     DI, AX
```

⑭

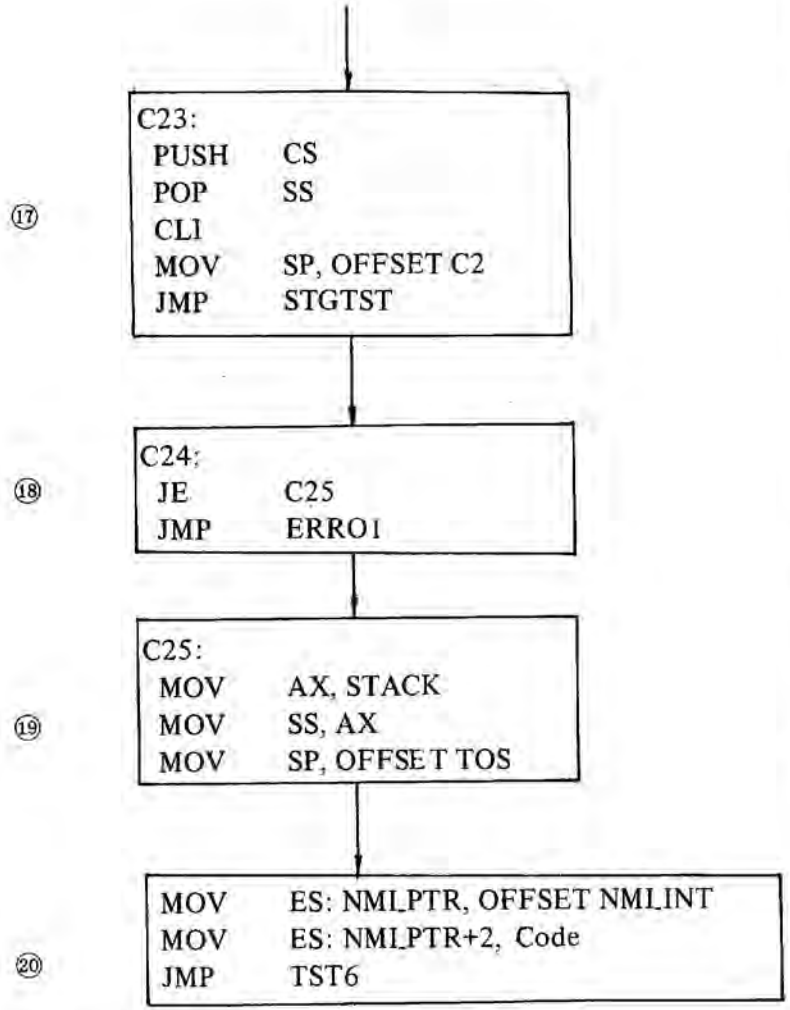
```
MOV     BX, 24H
MOV     [BX], OFFSET D11
INC     BX
INC     BX
MOV     [BX], CS
```

⑮

```
CALL    KBD_RESET
CMP     BL, 65H
JNZ     C23
```

⑯

```
MOV     DL, 255
C22:
CALL    SP_TEST
MOV     AL, BL
STOSB
DEC     DL
JNZ     C22
INT     3EH
```



⑬前面二個指令爲暫時設定堆疊的位址（SS暫存器爲0000H，SP暫存器爲3FF0H），並且將DI暫存器清除爲0000H（進入此步驟的AX暫存器，其內的值爲0000H）。

⑭將OFFSET D11(E2B6)存放到實際位址00024H處（DS暫存器爲0000H，BX暫存器爲24H），並且將CS暫存器內的值（F000）放到實際位址00026H處。這樣做的原因是爲了等一下要讀鍵盤的值。

⑮叫用KBD_RESET副程式，從鍵盤處得到一個鍵，然後檢查此鍵的值是否爲65H，如果是的話，表示要測試鍵盤是否工作正常（步驟⑯），如果此鍵不是65H的話，則直接進入步驟⑰測試記憶體。

⑯此步驟是從鍵盤處得到255個鍵，並且將此255個鍵經由STOSB指令儲存在位址00000H到000FEH處（ES暫存器內爲0000H，DI暫存器內爲0000H，DF旗號爲0）。SP_TEST副程式包含在KBD_RESET副程式內，待會再一併詳細討論。此步驟中，最值得注意的是INT 3EH這個指令。3EH是TYPE（請參考INT指令），實際位址在000F8H到000FBH，剛好是在鍵盤所提供的255個鍵內。由此可知鍵盤放出的255個鍵提供了一段程式，此段程式的進入位址在000F8H到000FBH(000FA和000FB內放著CS值，000F8和000F9內放著IP值)。讀者若對此段程式有興趣的話，請自行將鍵盤內的8048CPU內的程式列示（LIST）出來。

⑰此步驟爲跳入STGTST副程式測試記憶體前的準備工作。設定SS暫存器爲F000H和SP暫存器爲OFFSET C2，是爲了在STGTST副程式執行完後，能回到C24的地方（和在TEST.02中設定返回位址的方式是一樣的）。清除IF爲0，是爲了避免在測試記憶體時受到中斷干擾。

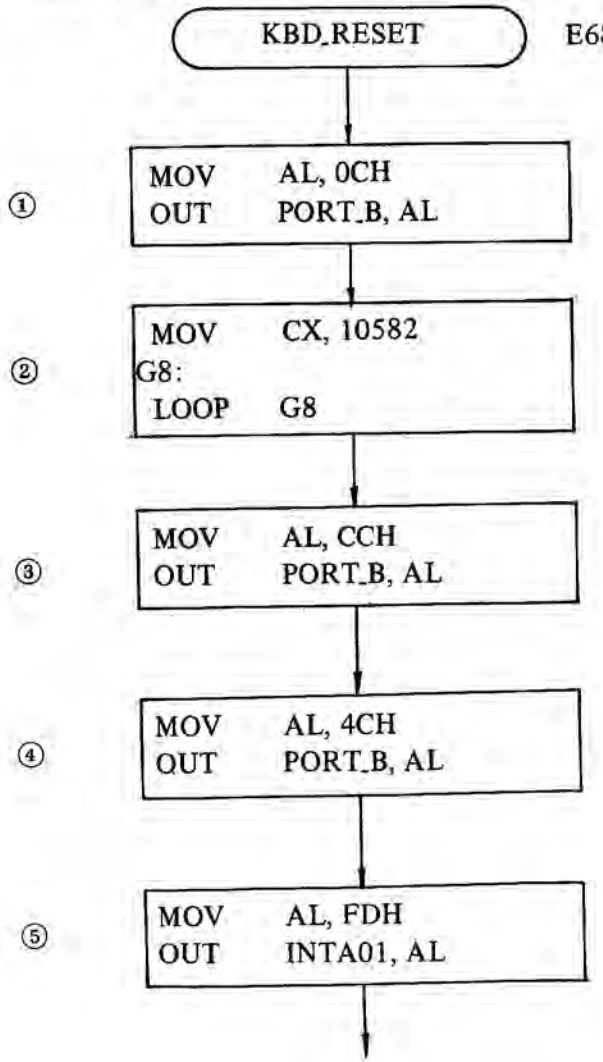
⑱STGTST副程式會返回這步驟，測試記憶體是否正確，如果不正確（ZF等於0），則跳入ERRO1停止系統。如果正確（ZF等於1），則進入步驟⑲。STGTST副程式待會也有詳細的說明。

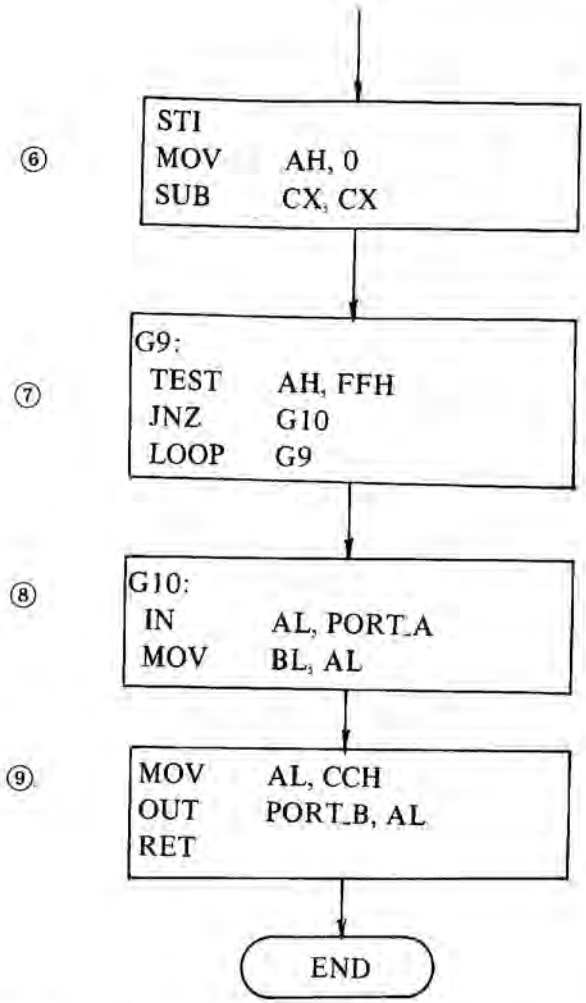
⑲設定堆疊位址（SS內爲0030H，SP內爲0100H）。

⑳前面二個指令設定處理NMI中斷信號的程式的起始位址。

處理NMI的程式的起始位址在FE2C3H處（其中CS暫存器內的值爲F000H，存放在實際位址0000A和0000B內，間距值為E2C3H，存放在實際位址00008和00009內）。接下來的JMP指令跳入TEST.06測試程式。

在討論完KBD_RESET副程式和STGTST 副程式後，讀者就能更進一步的了解，我們先看KBD_RESET副程式。





①將鍵盤的CLOCK強迫爲低電位，請參閱綫路圖。

②這個步驟只是等待20ms，也就是說強迫鍵盤的CLOCK成爲低電位20ms。

③這個步驟恢復鍵盤的CLOCK正常工作。這三個步驟給鍵盤一個重置（RESET）訊號。

④此步驟爲SP_TEST的進入點。這裡使得鍵盤能產生中斷要求（IRQ1）。

⑤將 8259 中 IRQ1 的遮罩位元設定爲 0 ，表示 8259 能夠接受 IRQ1 的中斷要求。

⑥ STI 指令使得 8088 可以接受中斷訊號。將 AH 暫存器設定爲 0 ，是爲了在步驟⑦中要用 AH 暫存器來檢查是否有硬體中斷要求。當 AH 暫存器內的值不爲 0 ，就表示有中斷訊號產生，而且已做完處理中斷的副程式（處理中斷的副程式會將 AH 暫存器設定爲 1 ），CX 暫存器是爲了配合步驟⑦的 LOOP 指令。

⑦這個步驟測試是否有硬體中斷要求，不論有沒有，都要進入步驟⑧。

⑧讀由鍵盤上得到的鍵碼，並且保存在 BL 暫存器內。

⑨清除 IRQ1 ，請看綫路圖。

我們繼續看處理鍵盤中斷信號的副程式。（這個副程式只是做測試用，眞正處理鍵盤中斷信號的副程式不是這個。）

```
①  MOV     AH, 1                    E2B6
    PUSH    AX
        |
        v
②  MOV     AL, FFH
    OUT     INTA 01, AL
    MOV     AL, EOI
    OUT     INTA00, AL
        |
        v
③  POP     AX
    IRET
```


①在 AH 暫存器內填入 1 ，表示已有中斷信號產生。並且將其保存在堆疊中。

②前面二個指令將 8259 內所有的遮罩位元都設定爲 1 ，表示此時 8259 不再接受任何中斷訊號。後面二個指令告訴 8259 中斷信號已經處理完了。

③取回 AX 暫存器的值，並且回到被中斷的位址。

由上面的討論，我們知道鍵盤和系統的溝通方式是經由硬體上的中斷信號（ IRQ1 ），所以由鍵盤送出的鍵碼爲 8 個位元，第一個位元一定爲 1 ，這個爲 1 的位元就是用來產生硬體中斷信號。請讀者自行參閱綫路圖上鍵盤部分，再配合 KBD_RESET 副程式，必能對其綫路和處理有更深一層的了解。

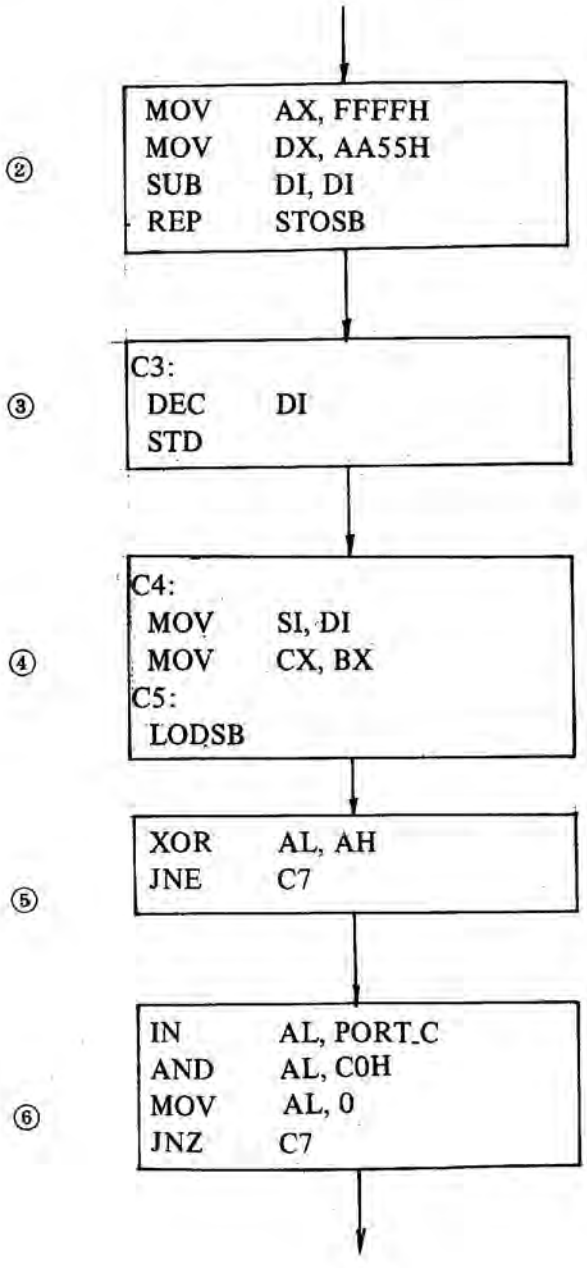
在步驟⑭中，我們將暫時處理中斷信號的副程式的起始位址，放到實際位址 00024H到 00027H的原因，爲 IRQ1的向量值爲 9 。當 8088 接到 IRQ1 的向量值後，先乘以 4 再加上 2 ，然後以此爲位址，取出 CS 值。接著將向量乘以 4 ，然後以此爲位址，取出IP值。

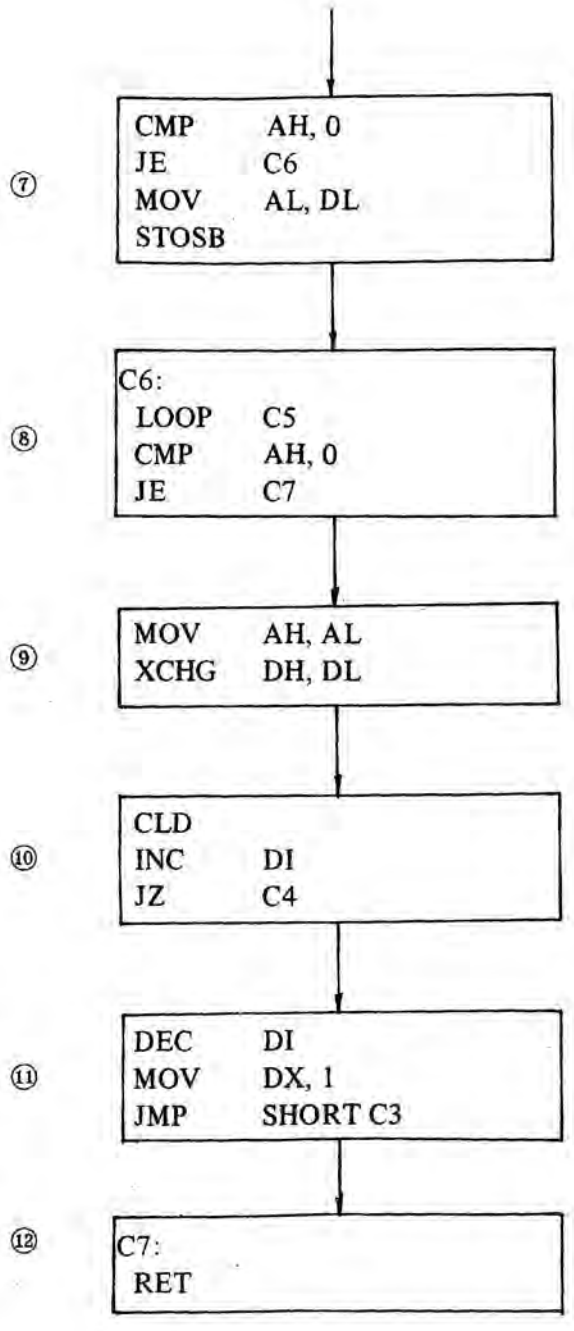
$$(CS) \leftarrow (Vector * 4 + 2)$$
$$(IP) \leftarrow (Vector * 4)$$

這就是爲什麼處理中斷信號的副程式的起始位址，要放在實際位址 00024H到 00027H的原因。

我們繼續討論測試記憶體的副程式 STGTST 。

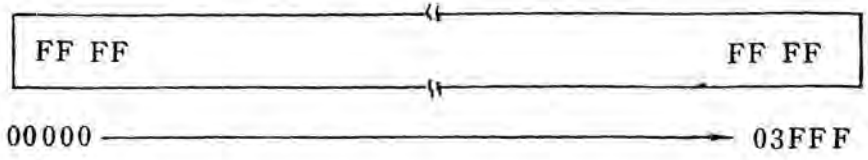
```
①  MOV     CX, 4000H                E01A
    CLD
    MOV     BX, CX
        |
        v
```


②
MOV AX, FFFFH
MOV DX, AA55H
SUB DI, DI
REP STOSB
③
C3:
DEC DI
STD
④
C4:
MOV SI, DI
MOV CX, BX
C5:
LODSB
⑤
XOR AL, AH
JNE C7
⑥
IN AL, PORT_C
AND AL, C0H
MOV AL, 0
JNZ C7

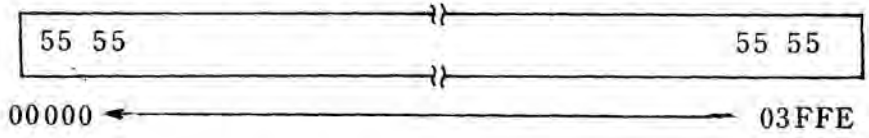

⑦
CMP AH, 0
JE C6
MOV AL, DL
STOSB
⑧
C6:
LOOP C5
CMP AH, 0
JE C7
⑨
MOV AH, AL
XCHG DH, DL
⑩
CLD
INC DI
JZ C4
⑪
DEC DI
MOV DX, 1
JMP SHORT C3
⑫
C7:
RET

STGTST副程式測試最低位址的16K位元組記憶體，它的測試方式是這樣的：

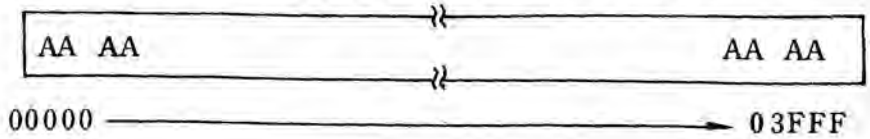
(a) 先將16K位元組都填入FFH，填入的順序爲由低位址向高位址。



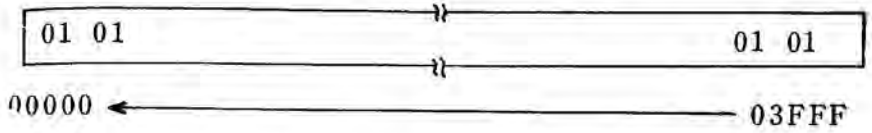
(b) 然後讀出位址03FFF內的值，測試其是否爲FFH，若不是，則回到TEST.04中，停止系統。若位址03FFF內的值爲FFH，則將位址03FFF內填入55H。然後讀出位址03FFE內的值，測試其值是否爲FF，若不是則回到TEST.04中，停止系統。若位址03FFE內的值爲FFH，則將位址03FFE內填入55H。依序向低位址推進，直到全部16K位元組的內容均爲55H爲止。



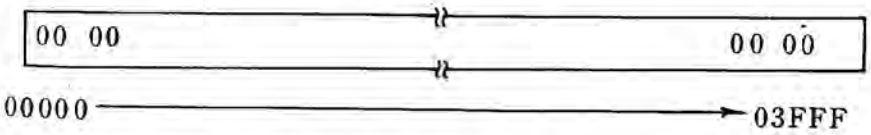
(c) 然後讀出位址00000內的值，測試其是否爲55H，若不是，則回到TEST.04中，停止系統。若位址00000內的值爲55H，則將位址00000內填入AAH。然後讀出位址00001內的值，測試其是否爲55H，若不是則回到TEST.04中，停止系統。若位址00001內的值爲55H，則將位址00001內填入AAH。依序向高位址推進，直到全部16K位元組的內容均爲AAH爲止。



(d) 然後讀出位址03FFF內的值，測試其是否爲AAH，若不是則回到TEST.04中，停止系統。若位址03FFF內的值爲AAH，則將位址03FFF內填入01H。然後讀出位址03FFE內的值，測試其是否爲AAH，若不是則回到TEST.04中，停止系統。若位址03FFE內的值爲AAH，則將位址03FFE內填入01H。依序向低位址推進，直到全部16K位元組的內容均爲01H爲止。



(e) 然後讀出位址00000內的值，測試其是否爲01H，若不是則回到TEST.04中，停止系統。若位址00000內的值爲01H，則將位址00000內填入00H。然後讀出位址00001內的值，測試其是否爲01H，若不是則回到TEST.04中，停止系統。若位址00001內的值爲01H，則將位址00001內填入00H 依序向高位址推進，直到全部16K位元組的內容均爲00H爲止。



(f) 然後讀出位址03FFF內的值，測試其是否爲00H，若不是則回到TEST.04中，停止系統。若位址03FFF內的值爲00H，則繼續測試位址03FFE內的值是否爲00H。依序向低位址推進，直到全部16K位元組都測試完。注意，此時不再填入任何值。

好了，讀者已經知道STGTST副程式測試最低位址16K位元組記憶體的步驟了。我們繼續討論STGTST副程式，這段程式組合語言用得很好，值得注意。

①在CX暫存器內放入4000H，以便配合步驟②的REP指令。4000H表示16K位元組，設定DF旗號爲0，表示在步驟②中，每執行完一次REP指令，DI暫存器內的值要加1。CX暫存器內的值保存在BX暫存器內，以便步驟④中再取出。

②將AH暫存器內填入FFH是爲了做測試用（在上述(b)中讀出FFH時）。AL暫存器和DX暫存器內的值是用來做寫入/讀出的數據（01和00在步驟⑪中設定）。DI暫存器用做STOSB指令的指標（ES暫存器爲0000H）。REP STOSB指令將最低位址的16K位元組都填入FFH。

③進入此步驟的DI暫存器內的值爲4000H，這裡將其減1，是爲了指向位址03FFF。設定DF旗號爲1，表示此時要向低位址前進。

④設定SI暫存器內的值和DI暫存器內的值相同，這是希望此步驟中的LODSB指令和步驟⑦中的STOSB指令指向同一個位址（請自行參考LODSB和STOSB指令）。MOV CX,BX指令恢復CX暫存器的初值。LODSB指令讀出SI暫存器所指的位址內的值。

⑤利用AH暫存器內的值，測試由LODSB指令讀出的值。這二個值應該要相同，否則會經由步驟⑫的RET指令，回到TEST.04中，停止系統。

⑥從8255的PORT-C的位元6和位元7來測試是否有同位查驗錯誤（parity check error）產生（位元7爲主機板上的同位查驗位元（parity check bit），位元6爲記憶體擴充卡上的同位查驗位元，若任一個位元爲1，就表示有同位查驗錯誤產生）。如果有同位查驗錯誤產生（ZF等於0），則會經由步驟⑫的RET指令，回到TEST.04中，停止系統。注意，MOV指令不影響任何旗號。

⑦首先測試AH暫存器內的值是否爲00H，若是，則不再填入任何值（直接進入步驟⑧，不執行本步驟的STOSB指令，請參考上述討論中的(f)），若AH暫存器內的值爲00H，則表示還要填入測試值。DL暫存器內的值爲要填入的值，它經由AL暫存器和STOSB指令，填入DI暫存器所指的位址。（DI暫存器所指的位址也就是SI暫存器所指的位址。LODSB指令利用SI暫存器讀出某個位址內的值，經測試正確後，STOSB指令利用DI暫存器寫入測試值於同一位址內。例如上述討論中的(b)，讀出位址03-FFF內的FFH，然後寫入55H於位址03FFF內，這55H的值由DL暫存器供給。）

⑧如果尚未填滿（或測試）16K位元組，則回到步驟④繼續填寫（或測試）。如果全部16K位元組都已讀出/寫入，則再一次檢查AH暫存器內的值是否爲00H，若是的話，就表示上述討論中的(f)已經處理完了，此時回到TEST.04中繼續執行程式。若AH暫存器內的值不爲00H，則表示尚未處理到上述討論中的(f)部分，也就表示整個測試尚未完成。這裡的CMP AH,0指令有別於步驟⑦的CMP AH,0指令，步驟⑦的CMP AH,0指令決定是否要寫入測試值。本步驟的CMP AH,0指令決定測試是否完成。

⑨將AL暫存器內的值移到AH暫存器內，以便做爲步驟⑤中測試所讀出的值是否和所寫入的值相同（AL暫存器內的值爲填入整個16K位元組的測試值，請看步驟⑦的MOV AL,DL指令）。將DH暫存器內的值和DL暫存器的值互換。DL暫存器內的值爲已經填滿整個16K位元組的測試值。DH暫存器內的值爲下個要填滿整個16K位元組的測試值。

⑩執行到此步驟，表示整個16K位元組內的值都是55H（01H），此時設定DF旗號爲0，表示要由低位址向高位址推進填入

AAH(01H)於整個16K位元組記憶體內。將DI 暫存器內的值加1，如果其值爲0，則要回到步驟④。這個INC DI 指令眞正的用意是，決定是否整個16K位元組記憶體已經都是AAH。在上述討論中的(c)，如果整個16K位元組都已是AAH的話，此時DI暫存器內的值爲4000H，則在此步驟中，會進入步驟⑪而不回到步驟④。在上述討論中的(b)。如果整個16K位元組記憶體內的值均爲55 H的話，此時DI暫存器內的值爲FFFFH，則在此步驟中會回到步驟④。讀者應該可以看出此步驟在測試是否已經使用過FF，55，AA三個測試值。

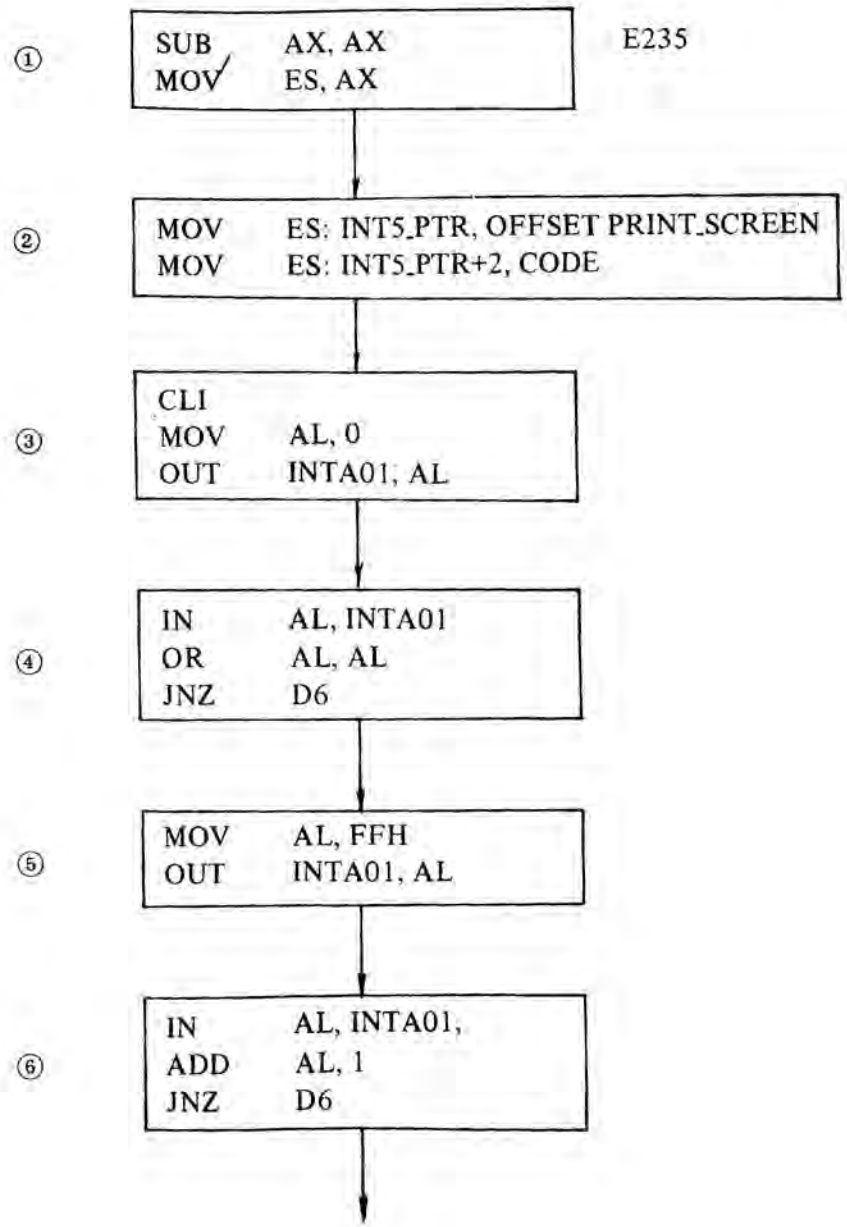
⑪進入此步驟時，就表示已經使用過FF，55，AA三個測試值。這裡將DI 暫存器內的值減1，是因爲步驟⑩將其加1。MOV DX，1指令預先準備測試值，DH暫存器內爲00H，是最後使用的測試值；DL暫存器內的值爲01H，是先用的測試值。然後回到步驟③執行測試。但別忘了此時 AH暫存器內的值爲AAH。

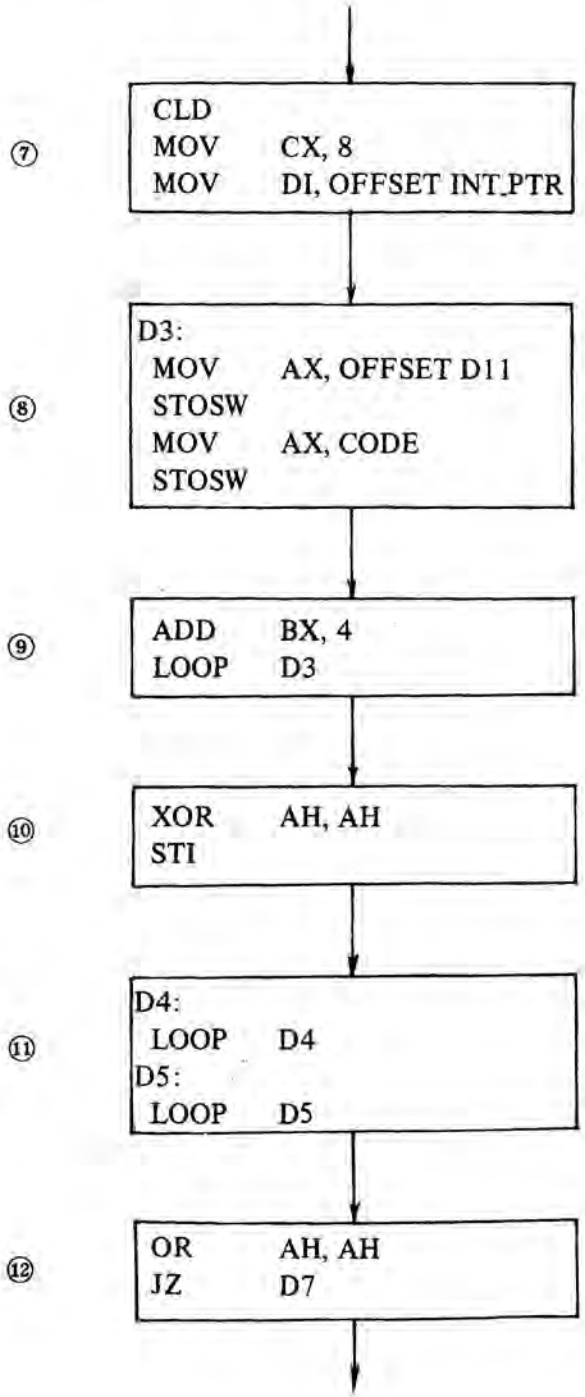
⑫這裡RET指令是同段間（intra-segment），所以它會到堆疊所指的位址，去拿出二個位元組當做返回位址。TEST.04中的⑰已經在堆疊中放入E018，所以RET指令會將位址FE018內的值取出（也就是TEST.04中的⑱）當做返回位址。

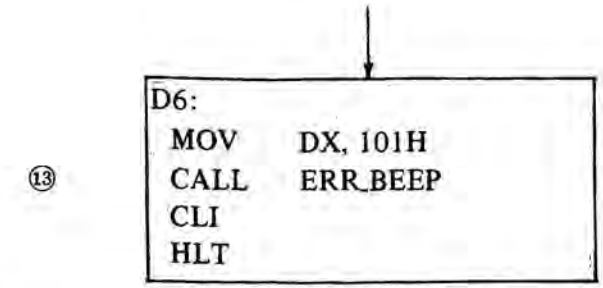
好了，我們已經討論完了TEST.04 程式，筆者希望讀者重新TRACE一次TEST.04 程式，雖然它很短，但內容却相當精彩。

TEST.06 8259中斷控制器測試(8259 Interrupt Controller Test)

現在我們來測試8259中斷控制器是否工作正常。







這段程式測試 8259 中斷控制器的方式是這樣的：先用 00H 寫入 8259 內的中斷遮罩暫存器（interrupt mask register），然後讀出並且測試其是否還是 00H。若正確，則再用 FFH 寫入 8259 的中斷遮罩暫存器，然後讀出並且測試其是否還是 FFH。若其中任何一項不正確，就會使喇叭聲響，並且停止系統，此時仍讓 8259 的中斷遮罩暫存器內的值爲 FFH，即表示 8259 不接受任何中斷信號。我們預先設定好暫時處理中斷信號的程式（即 TEST．04 中的處理中斷信號的程式），若這時的確執行了處理中斷信號的程式，即表示 8259 是不良的，也要使喇叭聲響，並停止系統。我們現在就來看 TEST．06 是如何寫的。

①將 ES 暫存器設定爲 0000H，以便在②中填入處理 print_screen 的程式的位址。

②當我們在鍵盤上按 shift-PrtSc 鍵時，印表機就會印出螢幕畫面上的文字。這是因爲我們預先設定好一個程式來處理這個動作。這個程式的起始位址在 FFF54，我們將這位址放在位址 00014 到 00016 處，也就是在中斷向量（interrupt vector）中的 TYPE 5。換句話說，當我們按 shift-PrtSc 鍵時，就會產生 INT 5H 指令（經由鍵盤 I/O 程式），即可跳至位址 FFF54 去執行程式。這個步驟將位址 FFF54 存放在中斷向量中的 TYPE 5。

③首先設定 IF 旗號爲 0，表示此時 8088 不接受任何中斷信號

，然後在 8259 的中斷遮罩暫存器內填入 00H 。

④讀出 8259 的中斷遮罩暫存器內的值，並且測試其值是否爲 00H，若不是，即表示 8259 不良，要跳入步驟⑬，先讓喇叭響一長聲一短聲，然後停止系統。如果由 8259 的中斷遮罩暫存器內讀出來的值爲 00H，則進入步驟⑤。

⑤在 8259 的中斷遮罩暫存器內填入 FFH。

⑥讀出 8259 的中斷遮罩暫存器內的值，並測試其是否爲 FFH，若不是，則進入步驟⑬，若是，進入下個步驟。

⑦我們知道 8259 所產生的中斷向量爲 8 到 F，也就是說，位址 00020 到 0003F爲 8259的中斷向量所佔用的位址。從此步驟到步驟⑨爲填入處理 8259 的中斷（interrupt）的副程式的起始位址。每一個向量佔用 4 個位元組位址，較低位址的二個位元組爲間距值（即 IP 值），較高位址的二個位元組爲分段值（即 CS 值）。在這裡，因爲做爲測試用，所以 8 個向量的間距值均爲 E2B6，CS 值均爲 F000，也就是說，處理 8259的 8 個中斷的副程式的起始位址均是 FE2B6。這個副程式已在 TEST.04 中討論過了。在此步驟中，首先設定 DF旗號爲 0，在 CX 暫存器內放入 8，表示要填入 8 個向量。在 DI暫存器內放入 INT_PTR（0020）是因爲 8259的向量由 8 開始。（別忘了 8088 得到向量值後要先乘以 4 ）

⑧這步驟填入 OFFSET 值和 CS值，請參閱 STOSW 指令。

⑨這裡將 BX 暫存器內的值加 4，是因爲每一個向量佔用 4 個位元組。如果尚未填完 8 個向量，會回到步驟⑧繼續填。如果已填入 8 個向量，進入下個步驟。

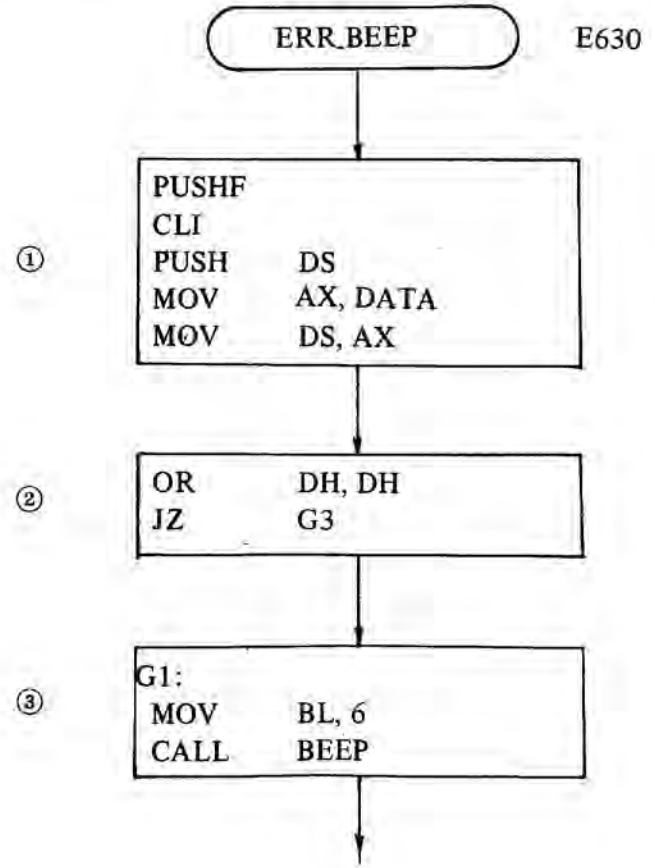
⑩將 AH 暫存器清除爲 00H是爲了測試是否有中斷產生。STI 指令使得 8088可以接受中斷信號。

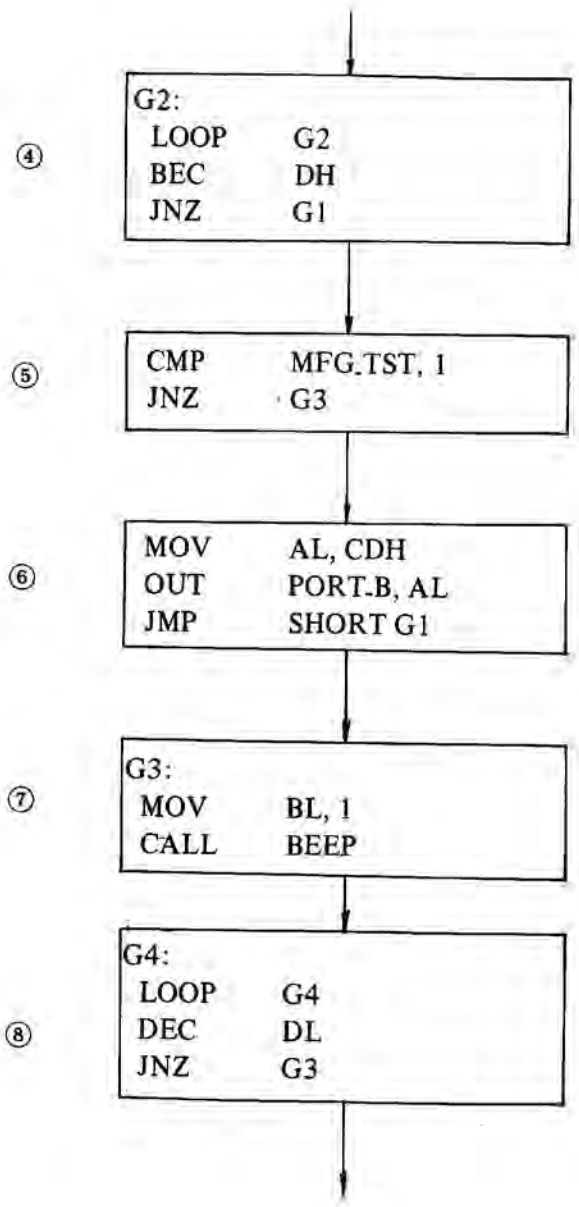
⑪這個步驟使得 8088 LOOP 1 秒鐘，以便等待中斷產生。

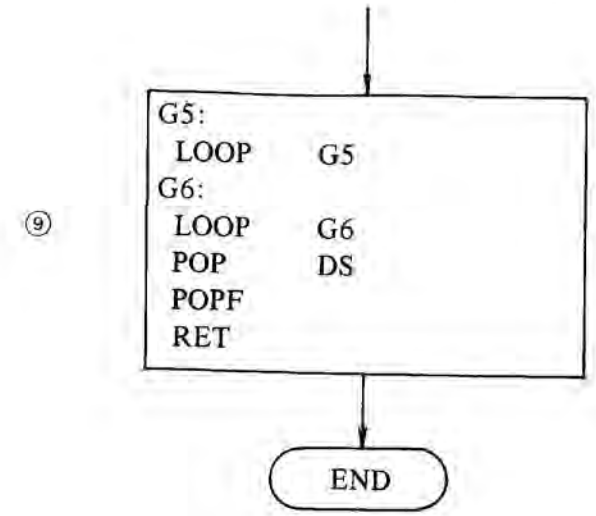
⑫我們知道暫時處理中斷的副程式首先會將 AH 暫存器設定爲 1。如果此時 AH暫存器爲 00H，即表示 8088 沒有處理過中斷要求。我們就認爲 8259 是良好的，而跳入下個測試程式。如果此時 AH 暫存器爲 1，即表示 8088處理過中斷要求。但是我們將 8259 的中斷遮罩暫存器填入 FFH，即表示我們希望 8259不接受 IRQ，並且不要求中斷處理，可是 8259 的確要求中斷處理，所以我們認爲 8259 是不良的。

⑬前面二個指令使得喇叭發出一長聲和一短聲。然後清除IF旗號，並且停止系統。

我們繼續討論如何使喇叭發出聲音。







這個程式使得喇叭發出聲音。長聲（3秒鐘）和短聲（1秒鐘）的次數保存在DH暫存器和DL暫存器內。

①將旗號暫存器和DS暫存器內的值保存在堆疊中，並且將IF旗號設定爲0，表示8088不處理中斷信號。將DS暫存器指向DATA部分（segment值爲0040）。

②檢查DH暫存器內的值是否爲00H，如果是的話（ZF旗號等於1），則表示不發長聲，直接進入步驟⑦發短聲。

③將BL暫存器內填入6，然後呼叫BEEP副程式，使得喇叭發出長聲。在BL暫存器內填入6的原因將會在BEEP副程式中解釋。

④這個LOOP指令只是做延遲用，以便區分聲音的次數。將DH暫存器內的值減1，以便決定是否還要發出長聲。

⑤檢查MFG_TST內的值是否爲1，若不爲1（ZF旗號爲0），則跳過步驟⑥，進入步驟⑦發出短聲。若MFG_TST內的值爲1，則表示在做生產（manufacturing）測試時發生錯誤，而使得喇叭發出連續的長聲。有關生產測試在下個測試程式中，將有詳細

說明。

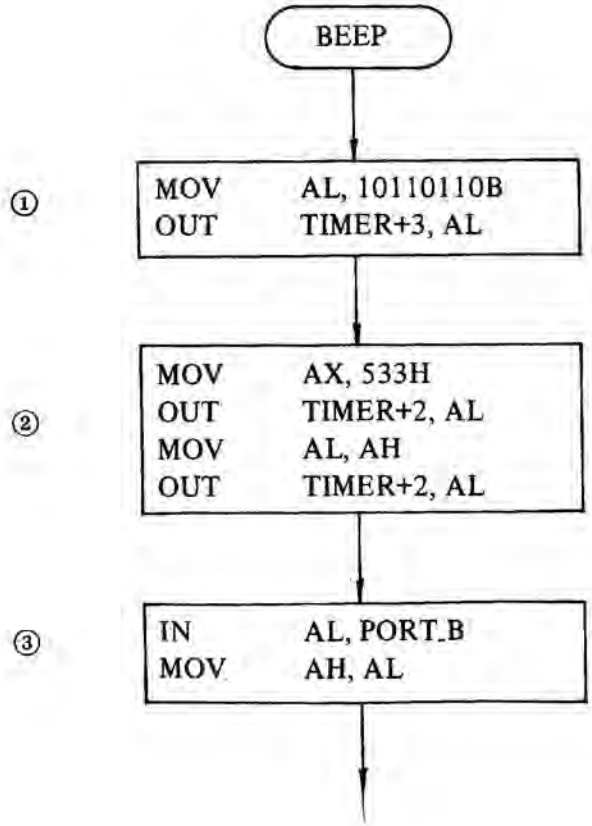
⑥熄掉鍵盤上的 LED ，然後回到步驟③發出長聲。熄掉鍵盤上的 LED 由8255的 PORT_B的位元 6 來控制。

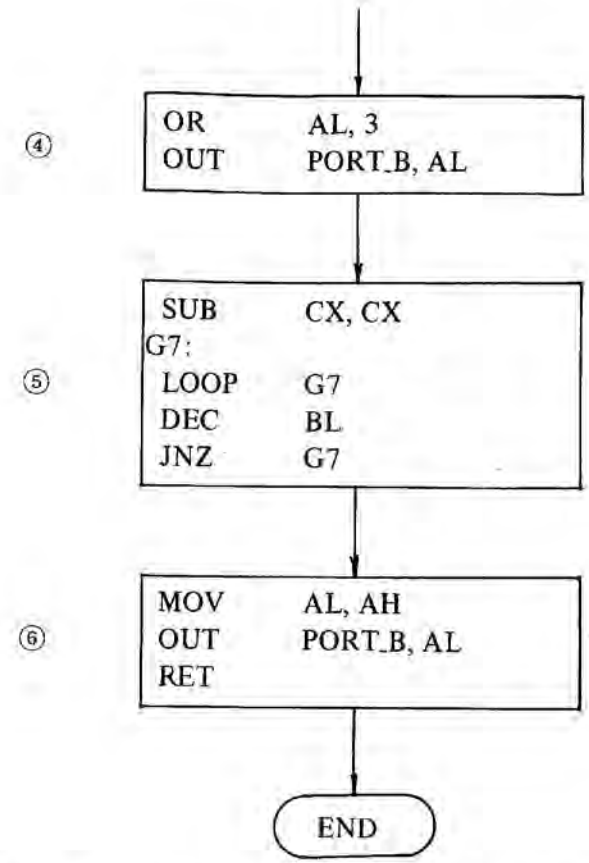
⑦在 BL暫存器內放入 1 ，然後呼叫 BEEP 副程式使喇叭發出短聲。

⑧ LOOP 指令做爲聲音間的延遲，然後檢查是否還要發出短聲。

⑨在返回呼叫 ERR_BEEP副程式的程式前，先做延遲。然後恢復 DS 暫存器和旗號暫存器的初值。

我們可以看出 ERR_BEEP副程式主要在做長聲和短聲的次數判斷和聲音間的延遲。眞正使得喇叭發出聲音的是 BEEP副程式。





①我們已經介紹過了 8253 Timer控制器，若讀者尚不熟，請自行翻至前面複習一下。這個步驟選擇 8253的計數器2，並且先設定其爲讀入最低有效位元組 (Read /Load least significant byte) ，接著爲最高有效位元組和模式 3 。

②此步驟將計數器 2 塡入 533 H 。我們知道模式 3 爲 square wave rate generator，高電位和低電位各佔計數的一半。這樣做使得喇叭發 1 KHz 的聲音。

③讀 8255 的 port_B 值，然後保存在 AH 暫存器內。之所以能讀 8255 的 port_B 值是因爲 8255 將輸出值 latch 住。

④將 8255 port_B 值的位元0和位元 1 補上 1 ，然後再將它寫

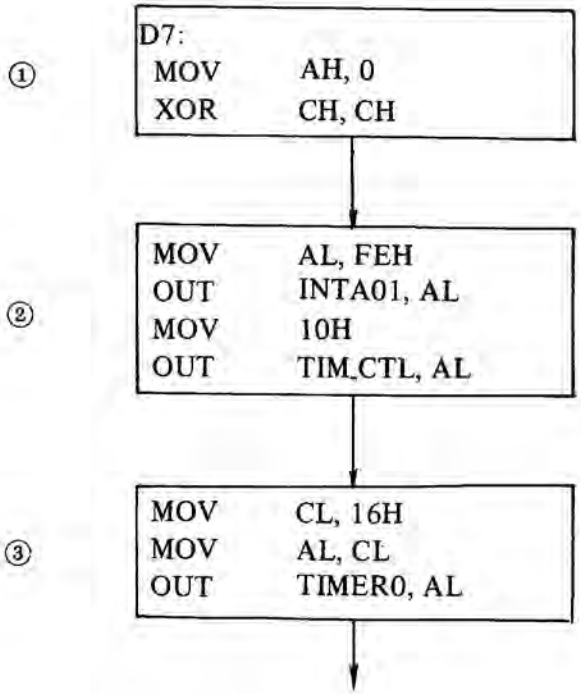
到 8255 的 port_B。這樣做的原因爲開啓喇叭，而不改變其他位元的狀態。

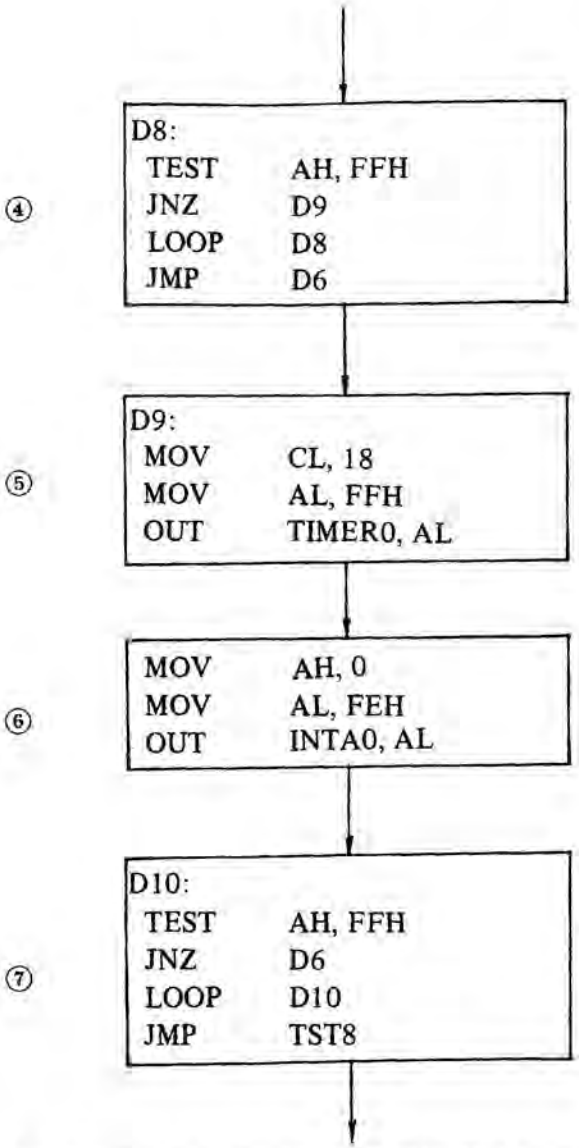
⑤這裡的 LOOP指令是做爲時間的延遲。步驟④已使得喇叭發出 1KHz 的聲音，這步驟決定喇叭聲響的時間。BL 暫存器內的值，對時間的長短有著重要的影響。

⑥將 8255 port_B的初值放到AL暫存器內，然後寫到 8255 的 port_B。如此使得 port_B 各位元的狀態，在喇叭聲響前後是一樣。

TEST. 07 8253時序控制器測試(8253 TIMER CHECKOUT)

這個測試程式測試 8253 的計數器是否算得太快或太慢。這裡使用 8253的計數器 0 和 8259的 IRQ0。





①首先清除 AH暫存器，因爲我們這裡使用 TEST.04中所討論的暫時處理中斷的副程式，然後也清除 CH暫存器。

②將FEH寫入 8259的中斷遮罩暫存器，表示 8259 可以接受 IRQ0的中斷要求。將 10H寫入 8253的模式字組暫存器，表示我們選擇 8253的計數器 0，並且設定其爲僅寫入最低有效位元組（

Read/Load least significant byte only)和模式 0。我們知道模式 0 爲 interrupt on terminal count，也就是說，計數器 0 的輸出一直爲低電位，直到計數器計數到 0 才升至高電位。

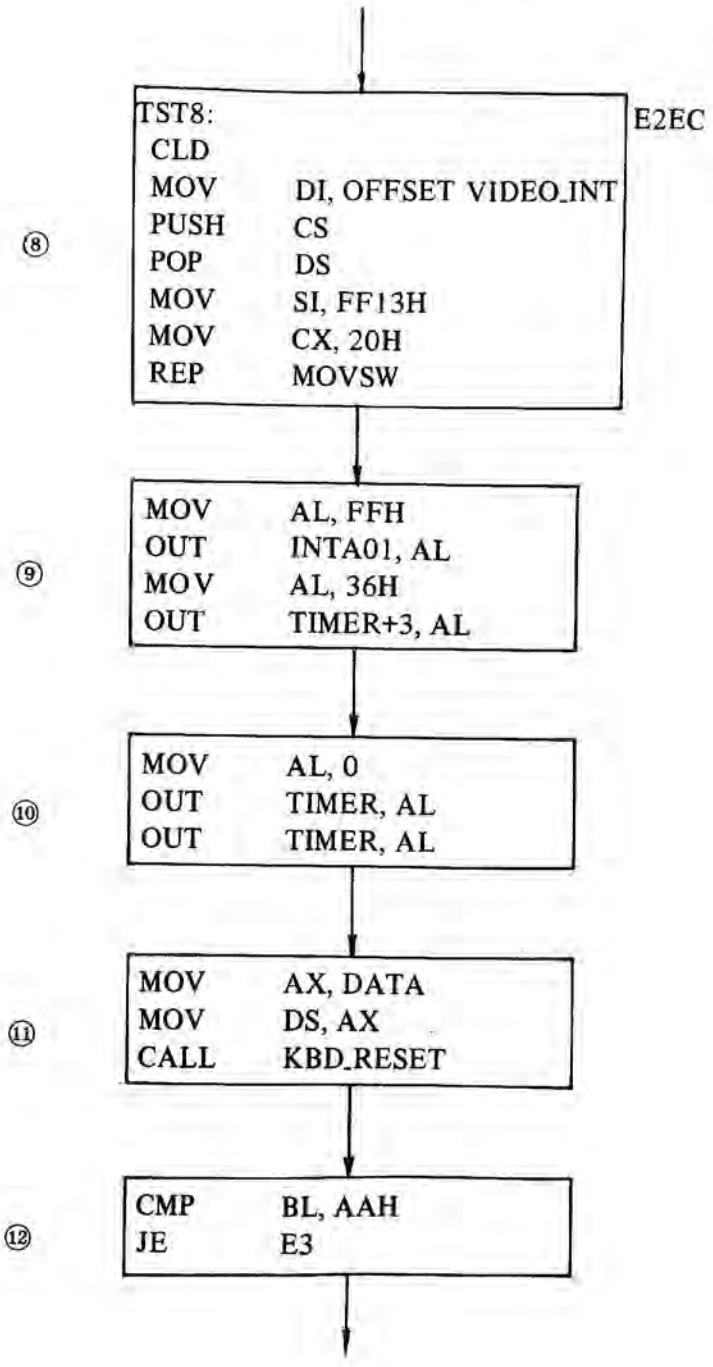
③在 CL 暫存器內放入 16H，以便配合步驟④的 LOOP 指令，並在 8253的計數器 0 內放入 16H。也就是說經過了 22 個 8253的 CLOCK 後（ 8253的 CLOCK爲 1.19MHz），計數器 0 的輸出脚成爲高電位，此時提供了 8259的 IRQ0 中斷要求信號。

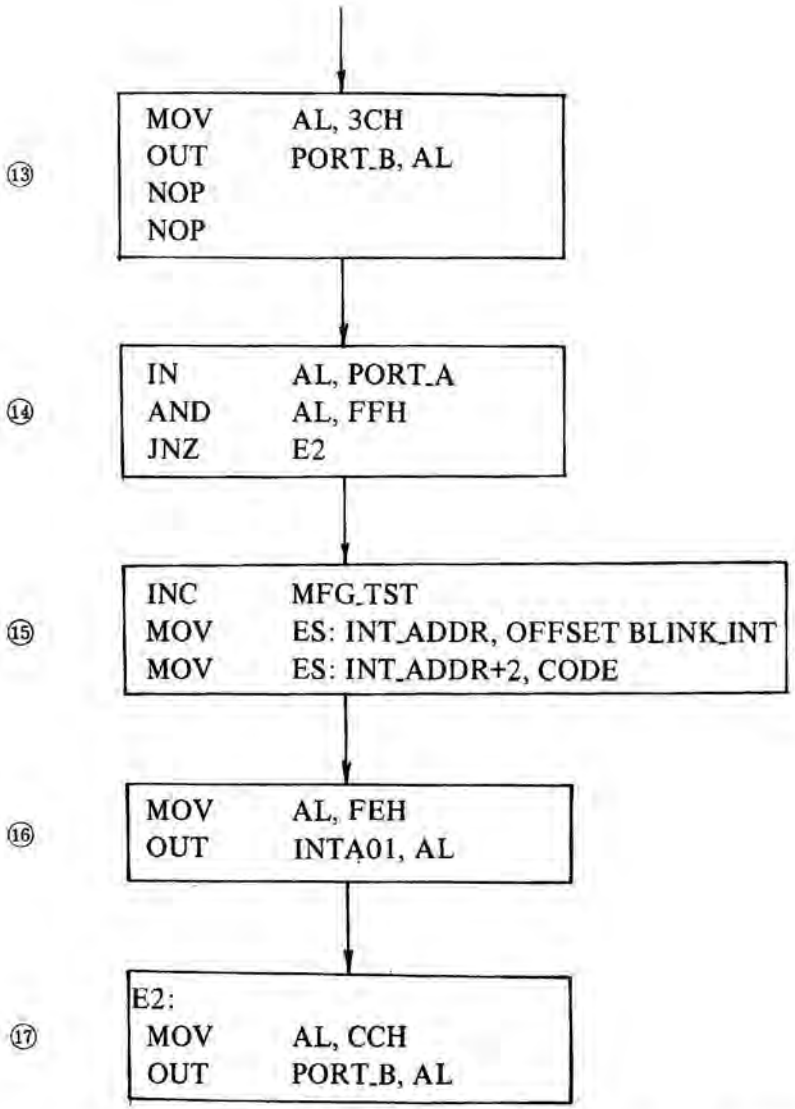
④這個TEST指令測試是否曾經處理中斷信號。若處理過中斷信號，則 AH 暫存器內的值爲 1 ，會跳入步驟⑨，表示計數器計數並不慢。若 LOOP過 22 次後，仍沒有中斷要求產生，則表示計數器計數太慢，會跳入D6（也就是TEST.06 8259中斷控制器測試中的步驟⑬），使喇叭發出一長一短的聲音。這個LOOP 22 次的時間絕對長於 8253的 22 個 CLOCK時間。

⑤在 CL 暫存器內放入 18 ，以便配合步驟⑦的 LOOP指令。在 8253的計數器 0 內放入 FFH，以便做爲計數用。

⑥暫時處理中斷信號的副程式會將 AH 暫存器設定爲 1 ，而且將 8259的中斷遮罩暫存器設定爲 FFH ，所以在這裡要重新設定。

⑦同樣地，這裡的 TEST 指令在測試是否曾經處理過中斷信號。因爲 256 個 8253的 CLOCK時間一定長於 LOOP 18 次的時間，所以，如果在 LOOP 的時間範圍內有中斷信號產生的話，即表示 8253 的計數器計數太快了，也要跳入 D6 ，使得喇叭發出一長一短的聲音，並停止系統。如果 LOOP 18 次後，仍沒有中斷信號產生，即表示 8253 的計數器計數不會太快。既然 8253 的計數不快也不慢，我們就認爲 8253 是良好的，會跳入 TST8 去執行程式。其實 TST8 是包含在 TEST.07 內，只因爲步驟⑦後有二個副程式，所以這裡使用 JMP 指令。





⑧這個步驟將處理 INT指令的副程式的起始位址填入中斷向量中，這些資料放在位址 FFF13H開始的 32個字組中。在 CX 暫存器內填入 20H表示要建立 16 個向量，每個向量使用二個字組。執行完這個步驟後，中斷向量將有如下的內容：（從 10H開始的 16 個向量）。

physical location	interrupt number	offset / code	interrupt function
00040H ~ 00043H	10H	F065 / F000	Video_IO
00044H ~ 00047H	11H	F84D / F000	equipment
00048H ~ 0004BH	12H	F841 / F000	memory_size
0004CH ~ 0004FH	13H	EC59 / F000	Diskette_IO
00050H ~ 00053H	14H	E739 / F000	RS-232C_IO
00054H ~ 00057H	15H	F859 / F000	cassette_IO
00058H ~ 0005BH	16H	E82E / F000	Keyboard_IO
0005CH ~ 0005FH	17H	EFD2 / F000	printer_IO
00060H ~ 00063H	18H	0000 / F600	ROM BASIC ENTRY POINT
00064H ~ 00067H	19H	E6F2 / F000	BOOT_STRAP
00068H ~ 0006BH	1AH	FE6E / F000	Time of day

0006CH ~ 0006FH	1BH	FF53 F000	Keyboard Break address
00070H ~ 00073H	1CH	FF53 F000	Timer Break address
00074H ~ 00077H	1DH	F0A4 F000	video parameters
00078H ~ 0007BH	1EH	EFC7 F000	Disk parameters
0007CH ~ 0007FH	1FH	0000 0000	pointer to video ext

如果我們執行到 INT 10H指令，就表示我們要跳至位址FF065H去執行程式。請讀者自行參閱 INT指令。

⑨將FFH寫入8259的中斷遮罩暫存器去，表示8259 此時不接受任何中斷要求。將 36H寫入 8253 的模式字組暫存器中，表示我們選擇計數器 0 ，並且先設定其爲寫入最低有效位元組（ Read/Load least significant byte ），接著爲最高有效位元組(most significant byte)和模式 3 。

⑩在 8253 的計數器 0 內填入 0000H。這裡使用 8253 的計數器 0 是爲了閃爍鍵盤上的 LED。

⑪呼叫 KBD_RESET 副程式從鍵盤上得到一個鍵碼。

⑫檢查此鍵碼是否爲AAH，若是AAH的話，表示不用閃爍鍵盤上的LED，直接跳入步驟⑰；若沒有得到 AAH 這個鍵碼，則進入下個步驟做進一步的測試。

⑬這個步驟將鍵盤的 CLOCK 保持成低電位，並將 LS322 的 clear腳保持高電位，此時等候鍵盤輸入資料。

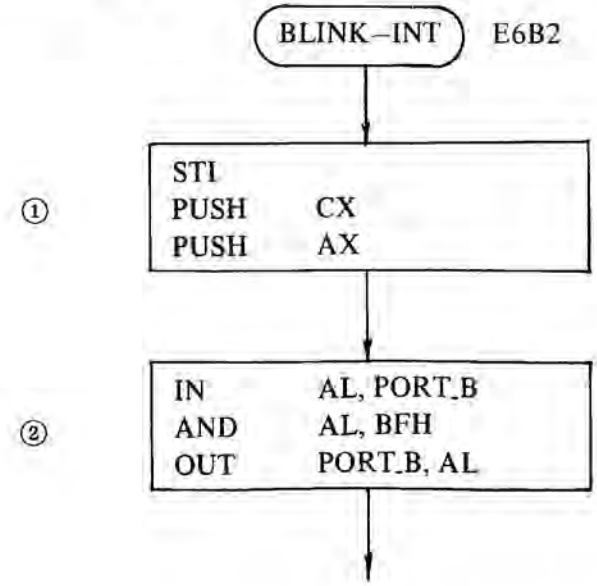
⑭讀鍵盤值。如果其值不爲 0（ ZF 等於 0 ），則表示不用閃爍鍵盤上的 LED。如果其值爲 0 ，表示鍵盤沒有輸入資料。此時進入下個步驟，做閃爍鍵盤上的 LED的準備工作。

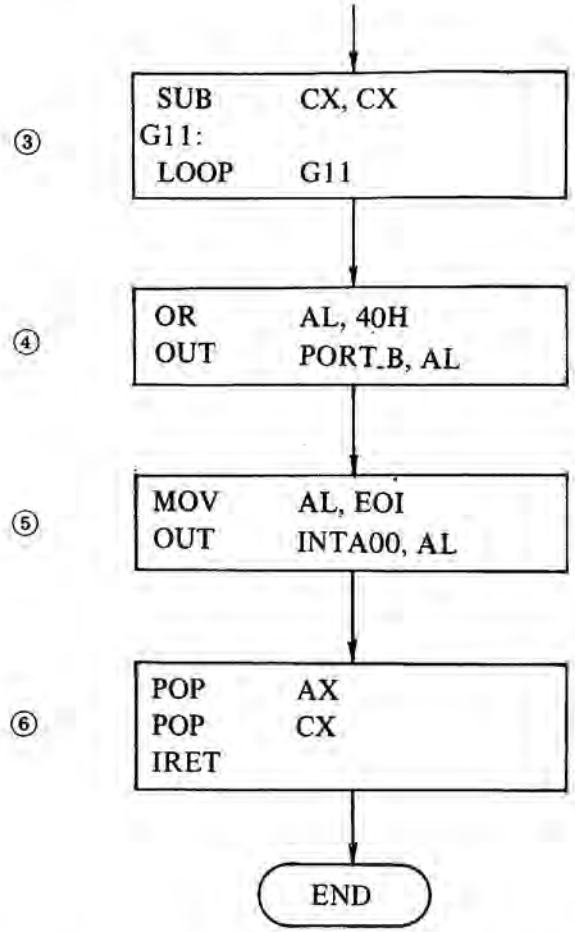
⑮首先將MFG_TST 值加 1 ，我們知道 8253 計數器 0的輸出腳接至 8259 的 IRQ0腳，也就是說，當 8253 的計數器 0 輸出變成高電位時，會產生向量值爲8 的中斷要求，所以接下來的二個 MOV指令在位址 00020 H到 00023 H處，預先放好處理閃爍鍵盤 LED的副程式的起始位址。

⑯將 FEH寫入 8259的中斷遮罩暫存器中，表示 8259可以接受 IRQ0中斷要求訊號。現在只要 8253 的計數器 0 輸出高電位，就會閃爍鍵盤的 LED。

⑰這個步驟使得鍵盤的 CLOCK 正常，但 LS322 的 clear腳仍是低電位。

我們現在來看如何使鍵盤上的 LED閃爍。





①首先設定IF旗號爲1，然後將CX暫存器和AX暫存器內的值保存在堆疊中，以免被破壞。

②讀8255 PORT_B的值，將其位元6設定爲0後，寫回8255的PORT_B。此時鍵盤上的LED會發亮。

③這個LOOP指令，讓LED繼續亮著。

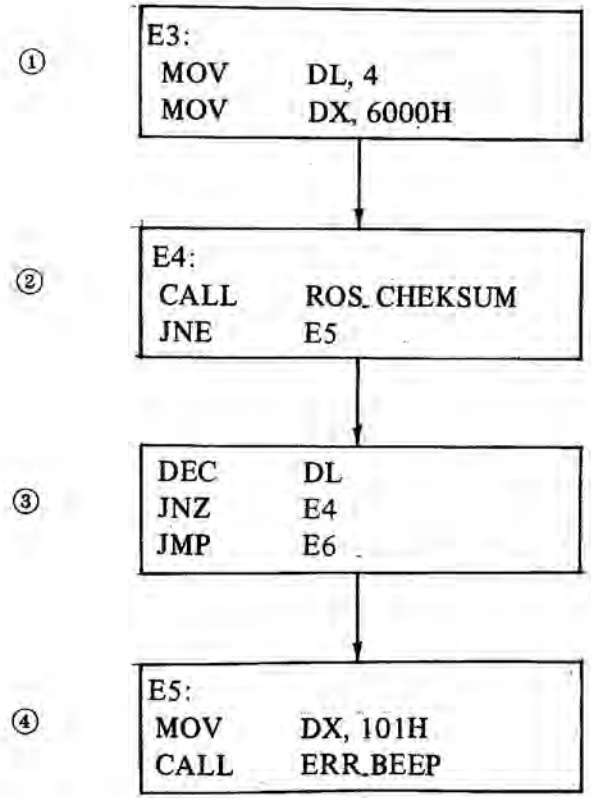
④將PORT_B的位元6設定爲1，然後寫到8255的PORT_B，此時鍵盤上的LED會熄滅。

⑤將20H寫入8259內，使得8259的通道0的遮罩位元變成1。

⑥恢復AX暫存器和CX暫存器的初值，然後回到被中斷的地方。

TEST. 05 BIOS ROM核對和測試Ⅱ (ROM CHECKSUM TEST Ⅱ)

這個測試程式，測試4個存放BASIC程式的僅讀記憶體(ROM BASIC MODULE)，每個8K的2764 ROM其內的資料相加得到的值應該是零。



①在DL暫存器內放入4，表示有4個ROM MODULES。在BX暫存器內放入6000H，是因爲第1個ROM MODULES的起始位址爲F6000H。

②呼叫ROS_CHECKSUM副程式，將8K位元組的資料相加。然後測試其值是否爲0，若不爲0，則表示錯誤，進入步驟④使喇叭發出一長聲一短聲。（ROS_CHECKSUM副程式已在前面討論過）

③DEC DL 指令測試，是否已測試完4個ROM MODULES，若已測試完4個ROM MODULES（ZF等於1），則跳入下個測試程式。若尚未測試完4個ROM MODULES（ZF等於0），則回到步驟②繼續測試。

④這個步驟使喇叭發出一長聲一短聲。

TEST. 08 6845和界面卡的測試(INITIALIZE AND START CRT CONTROLLER (6845) TEST VIDEO READ/WRITE STORAGE)

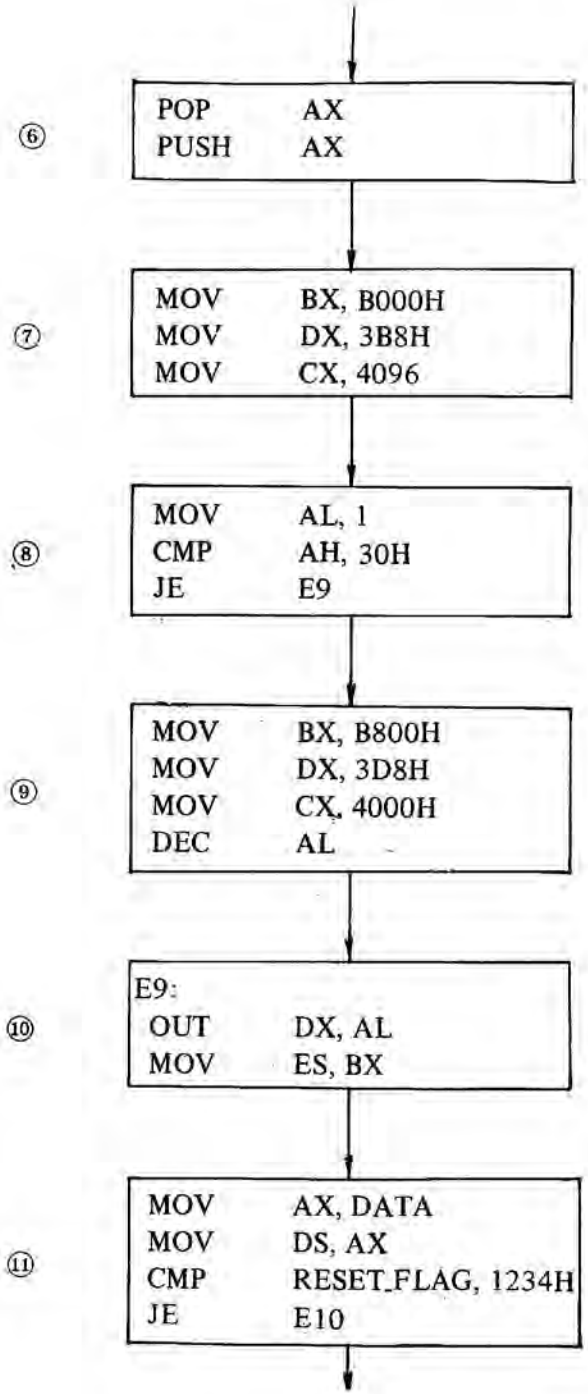
從技術手册中，我們知道DIP開關1的位元5和位元6決定螢幕的模式。

bit6	bit5	display mode
OFF	OFF	monochrone
ON	OFF	40×25
OFF	ON	80×25
ON	ON	Undefined

ON表示 logic 0，OFF表示 logic 1。

這個測試程式主要是爲了測試螢幕界面卡上的記憶體，所以首先必須決定出記憶體的起始位址（單色 monochrone）的記憶體起始位址爲B0000，彩色/繪圖（color/graphics）的記憶體起始位址爲B8000）和螢幕界面卡的I/O埠值。

```
                                        E352
①   IN      AL, PORT_A
    MOV     AH, 0
    MOV     EQUIP_FLAG, AX
        |
        v
②   AND     AL, 30H
    JNZ     E7
    JMP     E19
        |
        v
③   E7:
    XCHG    AH, AL
    CMP     AH, 30H
    JE      E8
        |
        v
④   INC     AL
    CMP     AH, 20H
    JNE     E8
    MOV     AL, 3
        |
        v
⑤   E8:
    PUSH    AX
    SUB     AH, AH
    INT     10H
        |
        v
```




```
CALL    STGTST_CNT
JE      E10
MOV     DX, 102H
CALL    ERR_BEEP
```

①首先讀DIP開關1的值，並且將其保存在EQUIP_FLAG中。

②檢查DIP開關1的位元5和位元6值，若位元5和位元6都是ON，表示爲設定錯誤，則不進行測試直接跳至TEST.11中。若DIP開關1的位元5和位元6不完全是OFF，則進入下步驟。

③在步驟①中，我們在AH暫存器內放入0，這裡的XCHG指令將AH暫存器和AL暫存器內的值互換，此時AL暫存器內的值爲0，AH暫存器內放著顯像模式（display mode）值。這是因爲在步驟⑤要呼叫display副程式，而display副程式以AL暫存器內的值來決定模式值。這個步驟和下個步驟，就是以AH暫存器內顯像模式值，來決定AL暫存器內的值。在這個步驟中，若DIP開關1的位元5和位元6都是OFF，則表示爲單色（monochrone），此時AL暫存器內的值爲0，進入步驟⑤。

④在這個步驟中，若測得DIP開關1的位元5和位元6爲ON和OFF，則表示爲80×25模式，AL暫存器內的值爲3。若測得DIP開關1的位元5和位元6爲OFF和ON，則表示爲40×25模式，AL暫存器內的值爲2。

⑤先將AX暫存器保存起來。將AH暫存器清除爲0，是因爲在display副程式（INT 10）中，AH暫存器內的值爲0表示爲設定模式，AL暫存器內的值表示顯像模式。此時螢幕畫面的左上角有游標（cursor）在閃爍。

⑥取出AX暫存器的初值（AH暫存器內爲DIP開關1的位元5和位元6值，AL暫存器內爲表示顯像模式的值），然後再將它保存在堆疊中。

⑦預先設定好單色的記憶體起始位址（B0000H），I/O port（3B8H）和記憶體容量（4096D）。

⑧先在AL暫存器內放入1，然後測試是否爲單色，若是（ZF等於1），則進入步驟⑩，做記憶體測試前的準備工作。若不是單色，則一定是彩色繪圖界面卡（color/graphics card），進入下個步驟。

⑨設定彩色繪圖界面卡的顯像記憶體的起始位址（B8000H），I/O port（3D8H）和顯像記憶體容量（4000H），並將AL暫存器設定爲0。

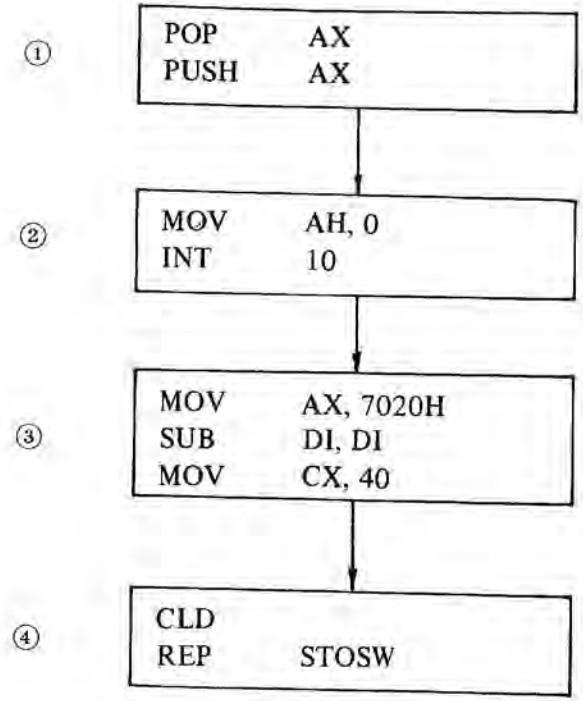
⑩這個OUT指令主要爲關掉視頻信號（disable video signal），請讀者自行參考技術手册中單色和彩色繪圖界面卡的介紹。下面這個MOV指令是爲了在步驟⑫中呼叫STGTST_CNT副程式測試顯像記憶體中，指出記憶體的起始位址。

⑪在DS暫存器內放入DATA（0040H）是因爲RESET_FLAG在segment值爲0040H的記憶體範圍內。這裡將1234H和RESET_FLAG比較是爲了測試是否爲溫起動（warm start）。若爲溫起動（由鍵盤上發出軟體RESET信號），則RESET_FLAG內的值爲1234H（在keyboard程式中。我們將會討論這個問題），表示不必測試顯像記憶體了，直接跳入下個測試程式，因爲冷起動已經測試過了。

⑫呼叫STGTST_CNT副程式，測試顯像記憶體（STGTST_CNT副程式已在前面討論過了）。若完成測試（ZF旗號爲1），則進入下個步驟，若測試結果不良，則使得喇叭發出一長二短的聲音。

TEST. 09 在螢幕上設定顯示資料以測試顯示列（Setup video data on screen for video line test）

這個測試程式主要爲開啓視頻信號和寫一列反白橫綫於畫面上。



①取出AX暫存器的初值，並且將其保存在堆疊中。

②這個步驟開啓視頻信號

③在AX暫存器內放入7020H是爲了寫入反白字樣。70H爲屬性值，表示反白。20H爲字元值，表示空白。設定DI暫存器爲0，表示所填入的反白字樣從顯像記憶體的最低位址開始。在CX暫存器內放入40，是爲了配合步驟，填入40個反白字樣。

④這個步驟使得螢幕出現一列40行的反白字樣於左上方。

這個簡短程式是爲了測試螢幕界面卡的輸出是否正常，到目前爲止，我們寫了一列40行的反白字樣於螢幕上，但是我們並不知道它是否正確。下個測試程式就是爲了測試這一列40行的反白字樣是否正常。

TEST. 10 螢幕界面板測試(CRT INTERFACE LINES TEST)

這段程式主要測試40行的反白字樣是否顯像正常。

①
```
POP     AX          E3C0
PUSH    AX
```

②
```
CMP     AH, 30H
MOV     DX, 3BAH
JE      E11
MOV     DX, 3DAH
```

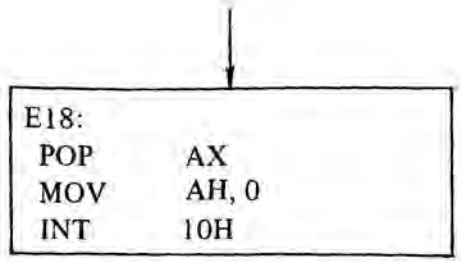
③
```
E11:
 MOV    AH, 8
E12:
 SUB    CX, CX
E13:
 IN     AL, DX
```

④
```
 AND    AL, AH
 JNZ    E14
 LOOP   E13
 JMP    SHORT E17
```

⑤
```
E14:
 SUB    CX, CX
E15:
 IN     AL, DX
 AND    AL, AH
 JZ     E16
 LOOP   E15
 JMP    SHORT E17
```

⑥
```
E16:
 MOV    CL, 3
 SHR    AH, CL
 JNZ    E12
 JMP    SHORT E18
```

⑦
```
E17:
 MOV    DX, 102H
 CALL   ERR_BEEP
```



⑧

```
E18:
 POP     AX
 MOV     AH, 0
 INT     10H
```

①讀取AX暫存器的初值，並且將它保存在堆疊中。

②這個步驟決定I/O埠的值。如果爲單色(monochrone)(AH暫存器內的值爲30H)，則I/O埠爲3BAH，若爲彩色/繪圖(color/graphics)則I/O埠爲3DAH。當用IN指令讀這個I/O埠的值，位元3表示視頻信號(單色)或垂直同步信號(Vsync signal)。在這裡，我們利用這個信號來判斷顯像是否正常。注意：MOV指令不影響任何旗號。

③在步驟④和步驟⑤中，我們要測試I/O埠值的位元3，而且我們都是以AH暫存器來測試，所以首先設定AH暫存器的位元爲1。將CX暫存器設定爲0，是爲了配合LOOP指令，然後讀I/O埠的值。

④AND指令測試I/O埠位元3的值。若其值爲1(ZF旗號爲0)，則進入步驟⑤，測試其是否能變成0。若其值爲0(ZF旗號爲1)，則返回至步驟③繼續讀I/O埠值，直到其變成1爲止，若LOOP指令執行364K次仍不能得到I/O埠的位元3值爲1，則我們認爲顯像部分是不良的，進入步驟⑦，使喇叭發出一長二短的聲音。

⑤這個步驟大致和步驟④相同，只不過這裡是測試I/O埠的位元3值是否能變成0。

⑥進入這個步驟，就已經表示單色的視頻信號或彩色/繪圖的垂直同步信號是正常的。這裡將AH暫存器右移3個位元，使得AH暫存器的位元0爲1，這是因爲我們要利用I/O埠的位元0值，來測試單色的水平同步信號或彩色/繪圖的顯像開啓信號，是否正常。JNZ E12決定是否二種信號都測試過了；若二種信號都已測試，則ZF旗號爲1，所以要進入步驟⑧。

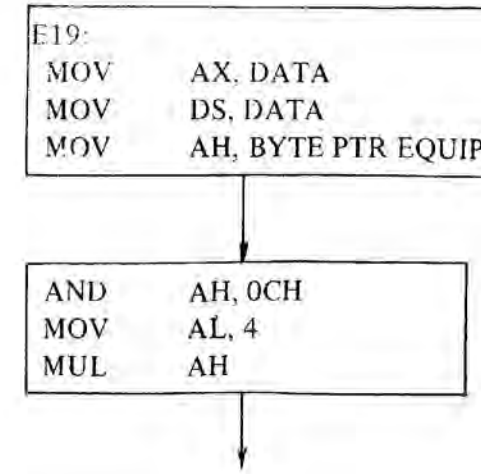
⑦這個步驟使得喇叭發出一長二短的聲音。

⑧取出AX暫存器的初值，然後呼叫顯像副程式，清除畫面。(此時40行的反白字樣會從螢幕畫面上消失)

若讀者對單色或彩色/繪圖綫路了解的話，就更能深入了解TEST.10的動作。INT 10H 這個副程式相當大，另闢一章說明。

TEST. 11 其他記憶體測試(ADDITIONAL READ/WRITE STORAGE TEST)

這個測試程式是爲了測試除了最低位址的16K位元組記憶體外，所有的記憶體是否正常。首先必須決定到底有多少記憶體，然後再呼叫STGTST副程式來測試記憶體。

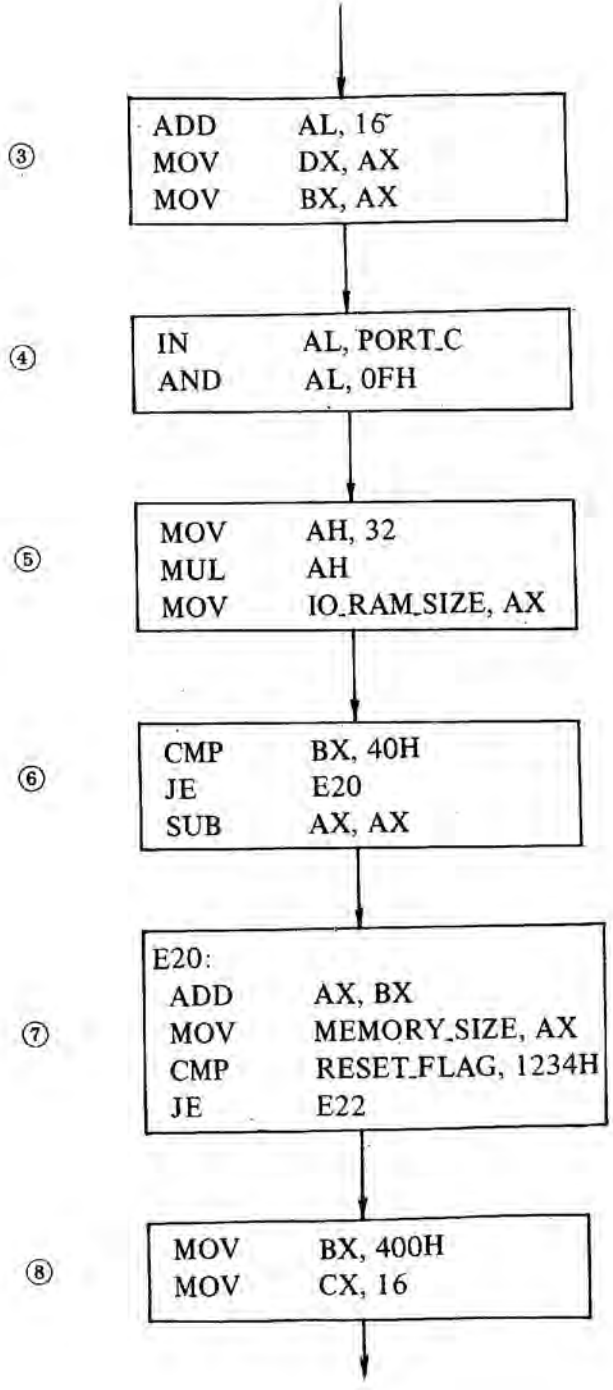


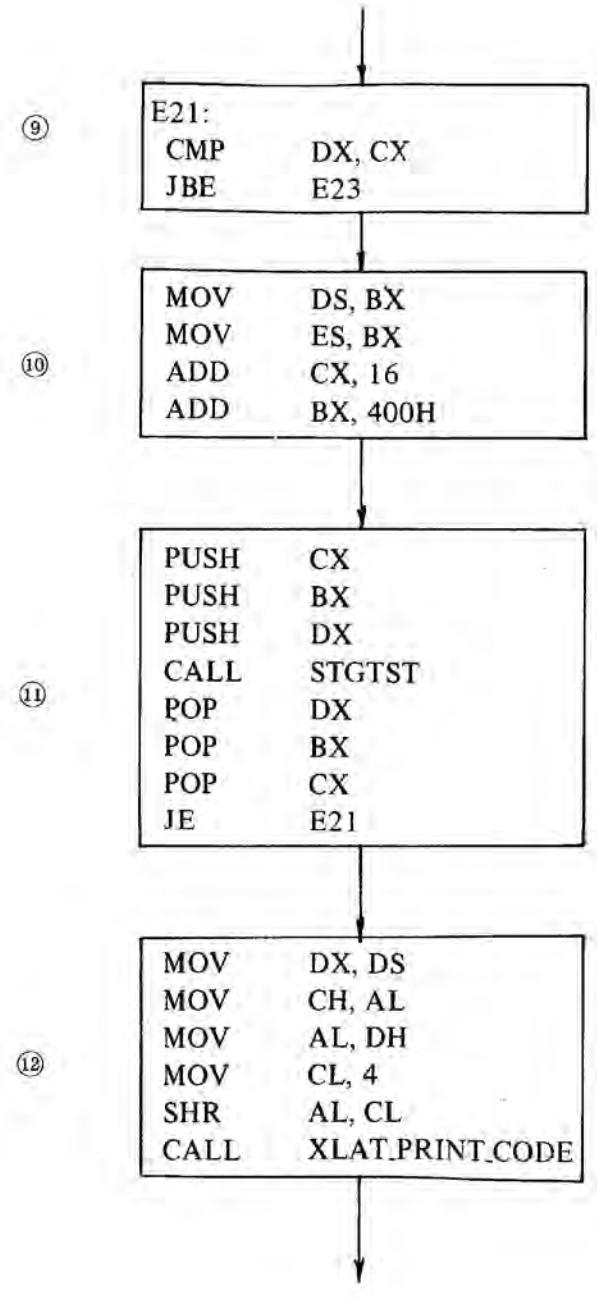
①

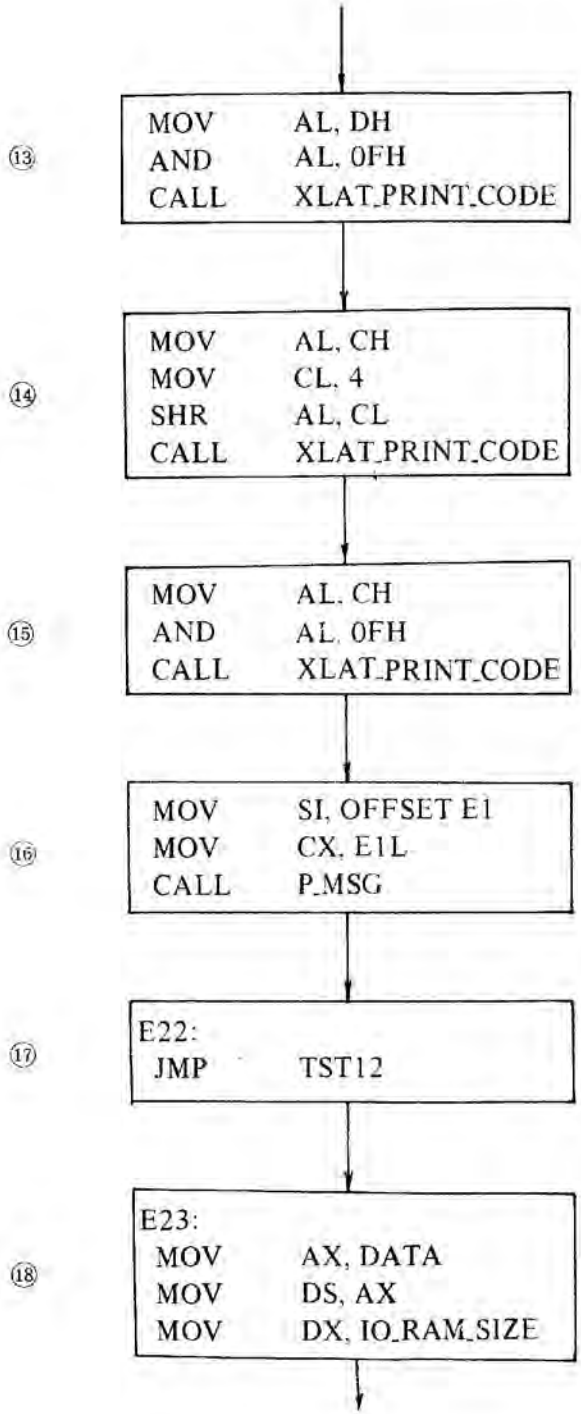
```
E19:                                    E3F8
 MOV     AX, DATA
 MOV     DS, DATA
 MOV     AH, BYTE PTR EQUIP_FLAG
```

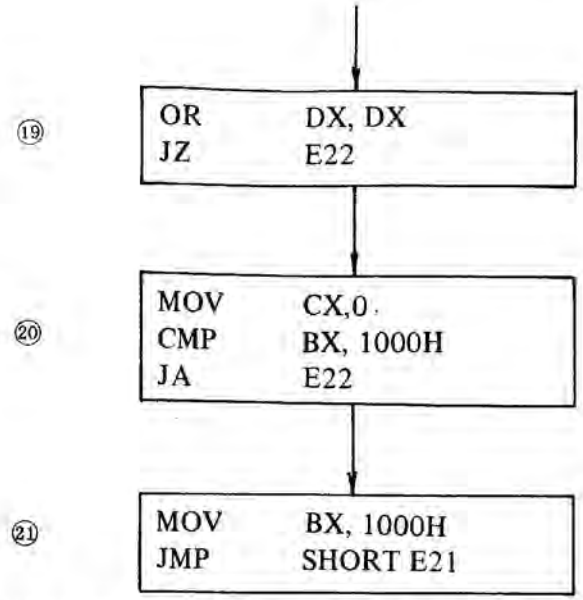
②

```
 AND     AH, 0CH
 MOV     AL, 4
 MUL     AH
```


③
ADD AL, 16
MOV DX, AX
MOV BX, AX
④
IN AL, PORT_C
AND AL, 0FH
⑤
MOV AH, 32
MUL AH
MOV IO_RAM_SIZE, AX
⑥
CMP BX, 40H
JE E20
SUB AX, AX
⑦
E20:
ADD AX, BX
MOV MEMORY_SIZE, AX
CMP RESET_FLAG, 1234H
JE E22
⑧
MOV BX, 400H
MOV CX, 16


⑨
E21:
CMP DX, CX
JBE E23
⑩
MOV DS, BX
MOV ES, BX
ADD CX, 16
ADD BX, 400H
⑪
PUSH CX
PUSH BX
PUSH DX
CALL STGTST
POP DX
POP BX
POP CX
JE E21
⑫
MOV DX, DS
MOV CH, AL
MOV AL, DH
MOV CL, 4
SHR AL, CL
CALL XLAT_PRINT_CODE





①將DIP開關1的值放到AH暫存器內，設定DS暫存器爲DATA(0040)，是因爲DIP開關1的值(EQUIP_FLAG)在段落爲DATA的記憶體範圍內。

②取出DIP開關的位元3和位元4值，並將其值乘以4。這是因爲DIP開關1的位元3和位元4值決定主機板上RAM(planar RAM)的大小。

bit4	bit3	planar RAM size
ON	ON	16K
ON	OFF	32K
OFF	ON	48K
OFF	OFF	64K

註：OFF表示邏輯1，ON表示邏輯0。

DIP開關1的位元3和位元4值，已被存放在AH暫存器內的位元

2和位元3，而且AH暫存器其他的位元都是0。乘以4，也就是將AH暫存器左移2個位元，執行完MUL後，結果放在AX暫存器內。所以執行完此步驟後，AX暫存器內的值反映出planar RAM的大小值。

AX暫存器	planar RAM size
0000H	16K
0010H	32K
0020H	48K
0030H	64K

③將AL暫存器加上16(16即爲10H)是因爲已經測試過最低位址的16K位元組記憶體。此時AX暫存器和planar RAM大小的關係是這樣的：

AX暫存器	planar RAM size
0010H	16K
0020H	32K
0030H	48K
0040H	64K

將此AX暫存器內的值轉移到DX暫存器內，是爲了在步驟⑨中做比較。將此AX暫存器內的值轉移到BX暫存器內，是爲了在步驟⑦中加上I/O通道的RAM大小。

④讀DIP開關2的值，並取出決定I/O RAM大小的位元。此4個位元保存在AL暫存器的最低4個位元。從技術手册中，我們知道DIP開關2和I/O通道RAM大小的關係。在這裡我們來看AL暫存器內的值和I/O通道RAM大小的關係。(註：DIP開關2的OFF表示邏輯1，ON表示邏輯0)

AL暫存器	I/O channel RAM size
00H	0K
01H	32K
02H	64K
03H	96K
04H	128K
05H	160K
06H	192K
07H	224K
08H	256K
09H	288K
0AH	320K
0BH	352K
0CH	384K
0DH	416K
0EH	448K
0FH	480K

由此表我們可以看出AL暫存器內的值每增加1，I/O通道RAM就增加32K。

⑤前面2個指令將AL暫存器內的值乘上32（結果在AX暫存器內），是因爲AL暫存器內的值每增加1，I/O通道RAM就增加32K，並且把代表I/O通道RAM大小的AX暫存器內的值保存在IO_RAM_SIZE內。執行完此步驟後，AX暫存器內的值和I/O通道RAM大小的關係爲：

AX暫存器	I/O channel RAM size
0000H	0K
0020H	32K
0040H	64K
0060H	96K
0080H	128K
00A0H	160K
00C0H	192K
00E0H	224K
0100H	256K
0120H	288K
0140H	320K
0160H	352K
0180H	384K
01A0H	416K
01C0H	448K
01E0H	480K

⑥這裡將BX暫存器內的值（也就是在步驟③中決定出的planar RAM大小的值）和40H比較。若BX暫存器內的值不等於40H（也就是說planar RAM小於64K），此時會執行SUB AX,AX指令，亦即表示此段程式只測試planar RAM，因爲planar RAM要64K，才允許有I/O通道RAM。若BX暫存器內的值等於40H，也就是此時可能有I/O通道RAM，直接進入下個步驟。

⑦進入此步驟的BX暫存器表示著planar RAM的大小，AX暫存器表示著I/O通道RAM的大小，將此二個RAM大小加在一起，就表示全部的RAM大小，然後將全部RAM大小（AX暫存器）保存在MEMORY_SIZE中。再將RESET_FLAG和1234H比較，若RESET_FLAG內的值等於1234H表示溫起動（warm start），也就是說冷起動時已經測試過RAM，直接進入步驟⑰。若RESET_FLAG不等於1234H，表示此時爲冷起動，要進入下個步驟開始測試記憶體。

⑧在BX暫存器內放入400H，是因爲要由第二個16K BANK開始測試，在CX暫存器內放入16（10H）是爲了在下個步驟中和DX暫存器比較。

⑨若DX暫存器內的值小於或等於CX暫存器內的值，就表示已經測試過RAM了，直接進入步驟⑱。注意，進入此步驟有二種可能，一由步驟⑧來測試planar RAM，一由步驟㉑來測試I/O RAM。

⑩將BX暫存器內的值轉移到DS和ES暫存器內，以表示要測試的RAM的起始位址。將CX暫存器內的值加上16，BX暫存器內的值加上400H，是爲了下個16K BANK做準備。（BANK與BANK之間，分段值相距400H）

⑪呼叫STGTST副程式測試16K位元組記憶體。因爲STGTST副程式會將CX,BX和DX暫存器破壞掉，所以要保存起來。如果16K位元組記憶體測試OK（ZF旗號爲1），則返回至步驟⑨，測試下一個16K BANK記憶體。若測試後，結果是不正確的，則AL暫

存器內記載著是那一個位元不正確。例如，由 STGTST 副程式返回時，ZF 旗號不等於 1，AL 暫存器內的值爲 00000001B，則表示所測試的那個 16K BANK 的位元 1 不良。DS 暫存器內的值，指出是那個16K BANK不良。

⑫從此步驟到步驟⑯是印出記憶體不良的錯誤碼，例如 64K到80K，這個 16K BANK 的位元 1 不良，由 STGTST 副程式返回後，AL暫存器內的值爲 00000001B,DS暫存器內的值爲 1000H。此步驟到步驟⑯將在螢幕畫面上印出 1001 201 的字樣。步驟⑫到步驟⑮印出前面 4 位數（ 1001 ），步驟⑯印出 201 字樣。

此步驟先將DS 暫存器內的值保存在DX暫存器內，然後將AL暫存器內的值保存在CH暫存器內。因爲 XLAT_PRINT_CODE副程式一次只能印出一位數，所以每個數字要分開處理。DH 暫存器內記載著測試不良的那個 16K BANK 的位址的高位元組（ higher byte ），我們要將這高位元組印出（ 也就是例子中最前面的二位數，10），將其右移 4 個位元是爲了配合XLAT_PRINT_CODE 副程式，因爲它一次只能印一位數。這個步驟印出第一位數。

⑬這個步驟印出第 2 個數字。

⑭此步驟和下個步驟所印出的數字，表示出是那一個位元不良。此步驟印出 AL 暫存器內的前 4 個位元（ 位元 7 到位元 4 ）。我們知道 16 進位制，每個數字由 4 個位元組成，所以這裡要將AL暫存器內的值分成二次印出。若印出的值爲 A，則表示位元 7 和位元 5 都壞了。

⑮此步驟印出 AL 暫存器內的後 4 個位元（ 位元 3到位元 0 ）。若印出的字爲 4 ，則表示位元 2 是壞的。請讀者先翻至後面 XLAT_PRINT_CODE 的討論。

⑯此步驟印出 201 錯誤碼，因爲 201錯誤碼放在位址FE2E8 處，所以在 SI 暫存器內放入 OFFSET E1（E2E8）以便配合

P_MSG 副程式。在 CX 暫存器內放入 EIL（4 ）表示要印出 4 個字。（ 在 201錯誤碼前有一空白 ）

⑰當測試完全部的 RAM ，或者是當有錯誤發生，印出錯誤碼後，經由此步驟，跳入下個測試程式。

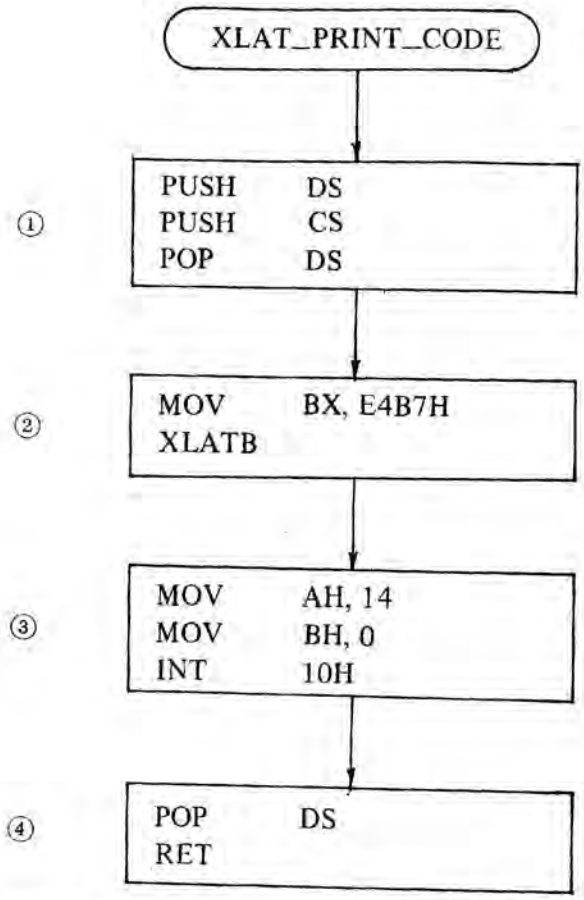
⑱取出 IO_RAM_SIZE 值放到 DX 暫存器內。此時要測試 I/O 通道 RAM了。在DS暫存器內放入 DATA（ 0040 ）是因爲IO_RAM_SIZE 在分段值爲 0040的記憶體範圍內。

⑲檢查DX 暫存器內的值是否爲 0 ，若爲 0 （ ZF旗號爲 1 ）則表示 I/O 通道 RAM 爲 0（ 即沒有 I/O 通道 RAM），此時進入下個測試程式。若 DX 暫存器內的值不爲 0 ，即表示有 I/O 通道 RAM，進入下個步驟。

⑳在 CX暫存器內放入 0 ，是因爲要從 I/O 通道RAM 的最低的BANK 開始測試。將 BX暫存器內的值和 1000H比較是因爲步驟⑨起的測試記憶體的 3 個步驟，並不能知道現在是測試planar RAM，或者是 I/O通道RAM ，當它測試完它應該測試的記憶體後（ 步驟⑨的 CMP 指令），它都會進入步驟⑱，所以我們必須在此步驟決定出是否測試過 I/O通道RAM。當BX 暫存器內的值大於 1000H就表示已經測試過 I/O通道 RAM了，因爲 I/O通道RAM 的分段值一定大於 1000H，此時也要經由步驟⑰進入下個測試程式。若BX 暫存器內的值小於或等於 1000H，就表示尚未測試過I/O通道RAM，因爲 planar RAM的分段值一定小於或等於1000 H，此時進入下個步驟開始測試 I/O 通道RAM。

㉑在 BX暫存器內放入 1000H，是因爲 I/O 通道 RAM的分段值從1000H 開始。然後跳入步驟⑨開始測試記憶體。

我們繼續討論XLAT_PRINT_CODE副程式。這個副程式利用 AL 暫存器內的值做爲指標，印出 16 進制的數字（ 0 到F ）。



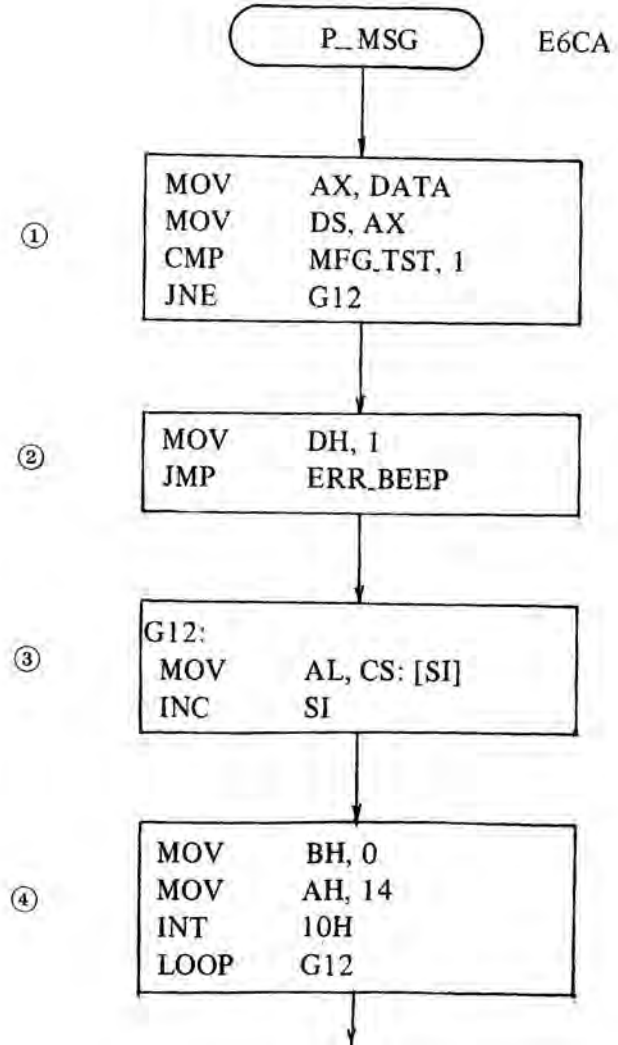
①先保存DS暫存器內的值，然後將DS暫存器內填入F000H，是因爲步驟②中，XLAT指令以DS暫存器內的值爲分段值。

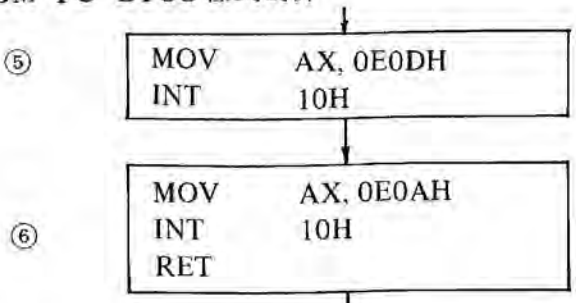
②我們複習一下XLATB指令。XLATB指令將BX暫存器內的值和AL暫存器內的值相加，然後以此和當作一個位址，再將此位址內的資料放入AL暫存器內。在位址E4B7H起的16個位址內，我們依順序放了0到F的ASCII碼。此步驟依照AL暫存器內的值，來取其相對的值，若AL暫存器內的值爲1，則取到1的ASCII碼，若AL暫存器內的值爲A，則取到A的ASCII碼。

③呼叫display副程式（INT 10H）將數字印出。在AH暫存器內放入14（0EH），BH暫存器內放入0，是爲了配合display副程式。

④取回DS暫存器的初值，然後返回至呼叫此副程式的程式去。

我們接著來討論P_MSG副程式。P_MSG副程式印出字樣於螢幕畫面上。





①這個步驟測試是否爲生產模式（manufacturing mode）（MFG_TST內的值爲1），若爲生產模式（ZF旗號等於1），則進入步驟②，使喇叭發出一長聲（短聲次數不定，依進入此步驟時，DL暫存器內的值而定），若不爲生產模式，則直接進入步驟③。在DS暫存器內放入DATA（0040），是因爲MFG_TST在分段值爲0040的記憶體範圍內。

②此步驟使喇叭發出聲音。

③將所要印出的字的ASCII碼放到AL暫存器內。此ASCII碼放在分段值爲CS，間距值爲SI暫存器內的值的位址內。將SI暫存器加1是爲了取出下一個ASCII碼做準備。所以在進入此副程式前，應先將SI暫存器設定好。

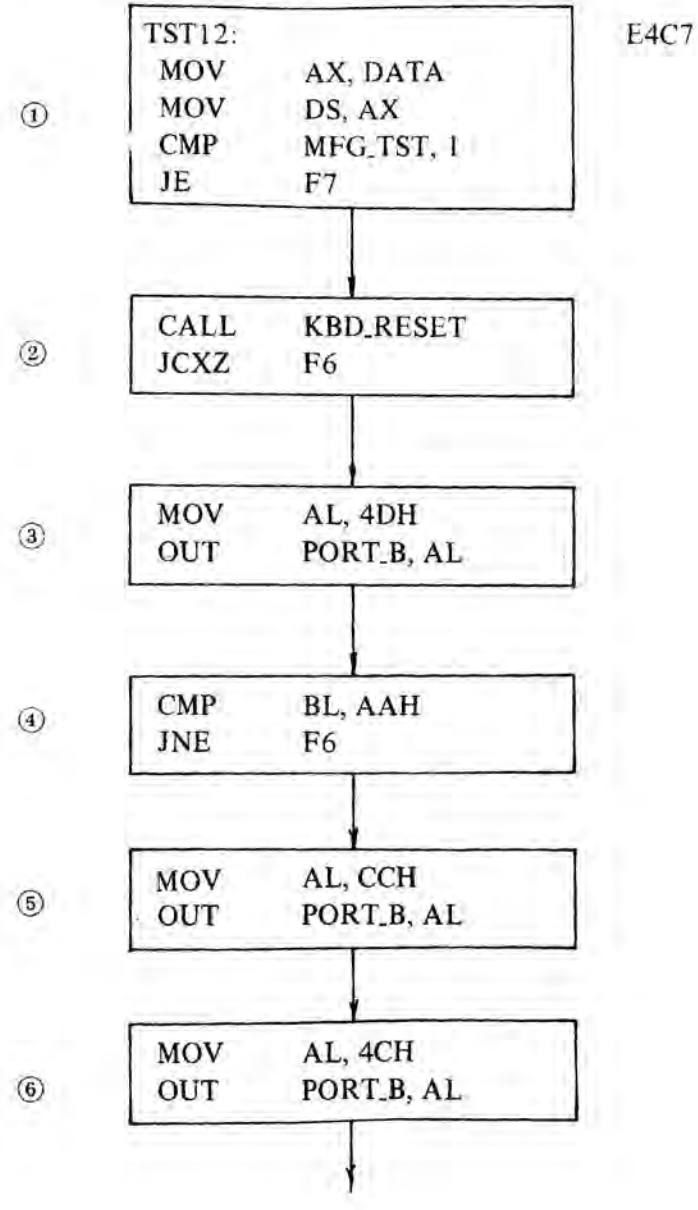
④呼叫display副程式將此字印出。設定BH和AH暫存器是爲了配合display副程式（INT 10H）。LOOP指令會將所要印的字全部印出，我們知道LOOP指令執行的次數和CX暫存器內的值有關。在進入此副程式前，應先將要印的字的數目放在CX暫存器內。

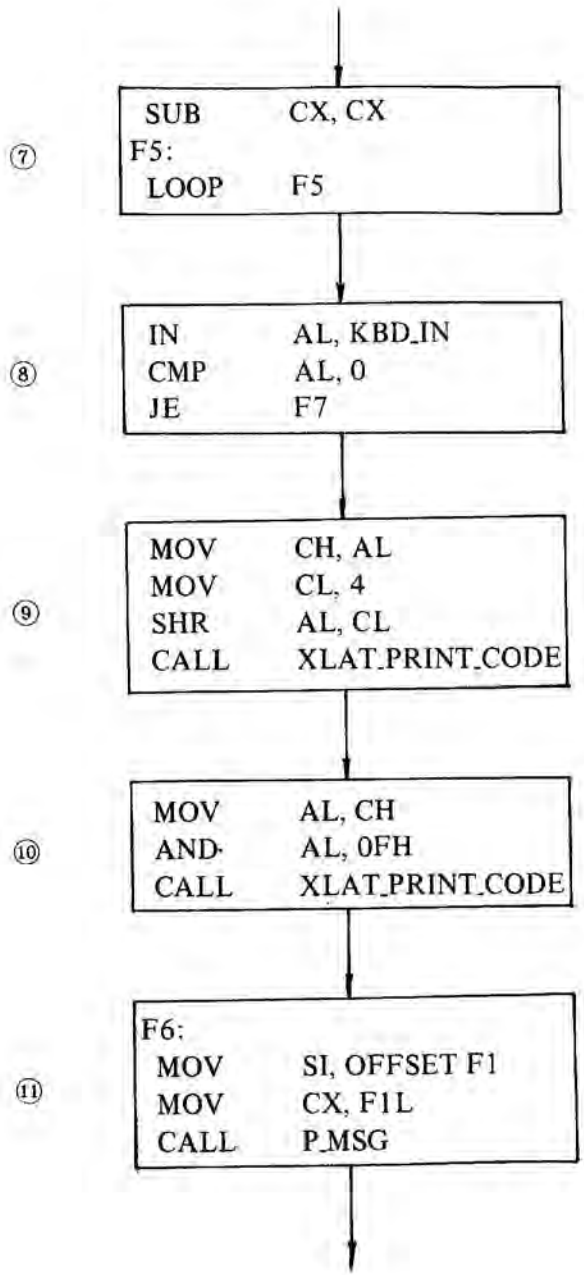
⑤這個步驟將游標移至下一列的最左端。

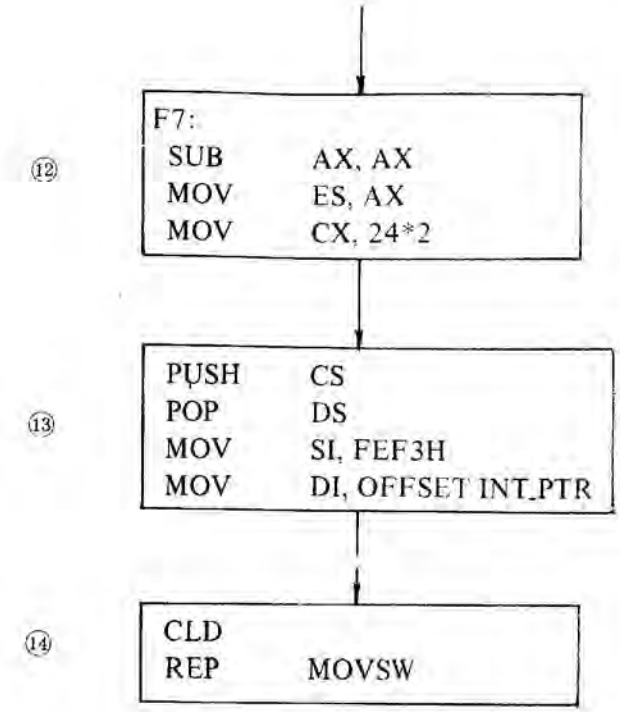
⑥這個步驟將螢幕向上捲動（如果游標已被移至螢幕畫面的下一列的話），然後返回至呼叫此副程式的程式去。

這個副程式呼叫了display副程式三次。可見INT 10H對此副程式相當重要。INT 10H相當大，留至另一章討論。

TEST. 12 鍵盤測試(Keyboard TEST)







①將MFG_TST內的值和1比較，若MFG_TST 內的值爲1，表示已經測試過鍵盤，直接跳入步驟⑫建立中斷表。若MFG_TST內的值不爲1，即表示尚未測試過鍵盤，進入下個步驟。

②呼叫KBD_RESET副程式，從鍵盤上得到一個鍵碼。（我們已經討論過KBD_RESET副程式了，不熟的讀者請自行複習）從KBD_RESET副程式回到這裡，若CX暫存器內的值爲零，就表示我們並沒有得到一個鍵碼，此時跳入步驟⑪印出301錯誤碼於螢幕畫面上。若CX暫存器內的值不爲零，即表示鍵盤有送出鍵碼，而且此鍵碼保存在BL暫存器內。

③將4DH寫到8255的PORT_B，是因爲KBD_RESET將LS-322的 $\overline{\text{CLR}}$ 腳保持低電位，這個OUT PORT_B，AL指令將其變成高電位，同時也可以產生鍵盤中斷信號（IRQ1）。

④將由鍵盤處得到的鍵碼和AAH比較。若其值爲AAH，則進入下個步驟測試是否有被卡住的鍵。若其值不爲AAH，則表示鍵盤不良，進入步驟⑪印出 301 錯誤碼。

⑤這個步驟設定儲存鍵碼的暫存器（LS 322）的輸出爲低電位（8個位元都爲低電位）。

⑥這個步驟啓動鍵盤，使得鍵盤可以送出鍵碼。

⑦這個迴圈等待鍵盤的中斷信號產生，也就是說，等待鍵盤送出鍵碼。如果有任何鍵被卡住，這時鍵盤內的CPU 8048 一定會掃描到，而且會送出這個鍵的鍵碼（SCAN CODE）。如果沒有鍵被卡住，此時LS 322的 8 個輸出位元都還會是 0 。

⑧讀LS322 的 8 個輸出位元，若其值爲 0 ，就表示沒有鍵被卡住，跳至步驟⑫建立中斷表。若LS322 的 8 個輸出位元不爲 0 ，則表示有鍵被卡住，進入下個步驟，陸續印出鍵碼和 301 錯誤碼。

⑨此步驟將鍵碼的第1 個數字，經由 XLAT_PRINT_CODE 副程式，印在螢幕畫面上。

⑩此步驟將鍵碼的第 2 個數字，經由 XLAT_PRINT_CODE 副程式，印在螢幕畫面上。

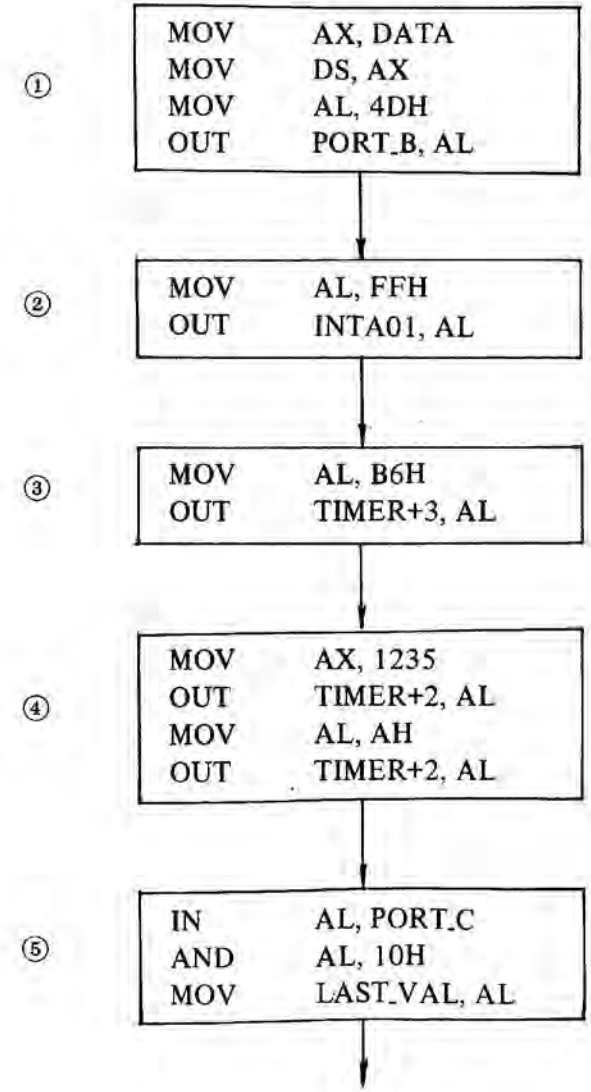
⑪此步驟印出錯誤碼 301 於螢幕畫面上。301 各字的ASCII 碼保存在位址FE4A7 起的 4 個位址內（301前有一空白），所以將E4A7放入SI暫存器內，以便P_MSG 副程式取出資料。在CX 暫存器內放入F1L（0004）表示要印出 4 個字。

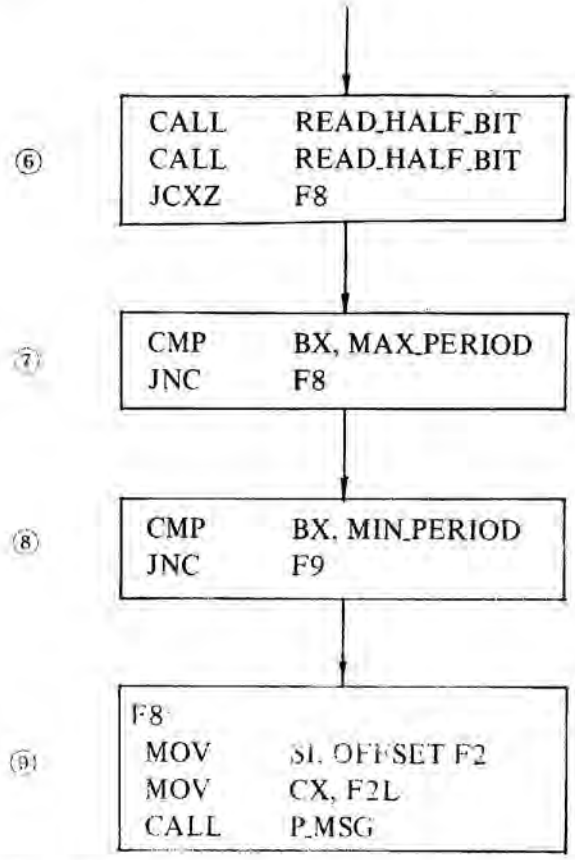
⑫以下的3個步驟建立中斷表。將ES暫存器設定爲0 是因爲中斷表的段落值爲 0000H。在CX 暫存器內放入 24*2（30H），表示要建立 12 個向量值。（一個向量佔用 2 個字組）

⑬因爲中斷表的值放在位址FF EF 3H 起的24個字組，所以在DS 暫存器內放入 F000H，SI 暫存器內放入FEF3H，以便下個步驟中MOVSW 指令取用資料。在DI 暫存器內放入OFFSET INT_PTR（0020），是因爲所要建立的中斷表由位址 00020H 開始，也就是從 INT 8H 開始建立起（ES 暫存器內的值爲 0000H）。

⑭此步驟開始填入中斷表的值。（請參考MOVSW指令）

TEST. 13 錄音機測試(CASSETTE DATA WRAP TEST)





①首先設定DS暫存器爲DATA(0040H)。然後將4DH寫到8255的PORT_B，如此使得8253的計數器2輸出爲錄音機所用。我們從技術手册主機板上的綫路圖知道，8253的計數器2輸出也做爲喇叭聲響之用。但如果8255的PORT_B的位元1爲0的話，喇叭就不會響了，則此時8253計數器2的輸出爲錄音機所用。

②將FFH寫到8259的中斷遮罩暫存器(interrupt mask register)中，表示在這個測試中，不接受任何中斷信號。

③此步驟選擇8253的計數器2，並且先設定其爲模式3和寫入最低有效位元組，接著爲最高有效位元組。

④此步驟將1235(04D3H)寫到8253的計數器2內。

⑤cassette data由8255的PORT_C的位元4進入。此步驟讀取cassette data並且保存在LAST_VAL內。LAST_VAL在分段值爲DATA(0040)的記憶體範圍內，這就是爲什麼在步驟①要先設定DS暫存器爲DATA的理由。

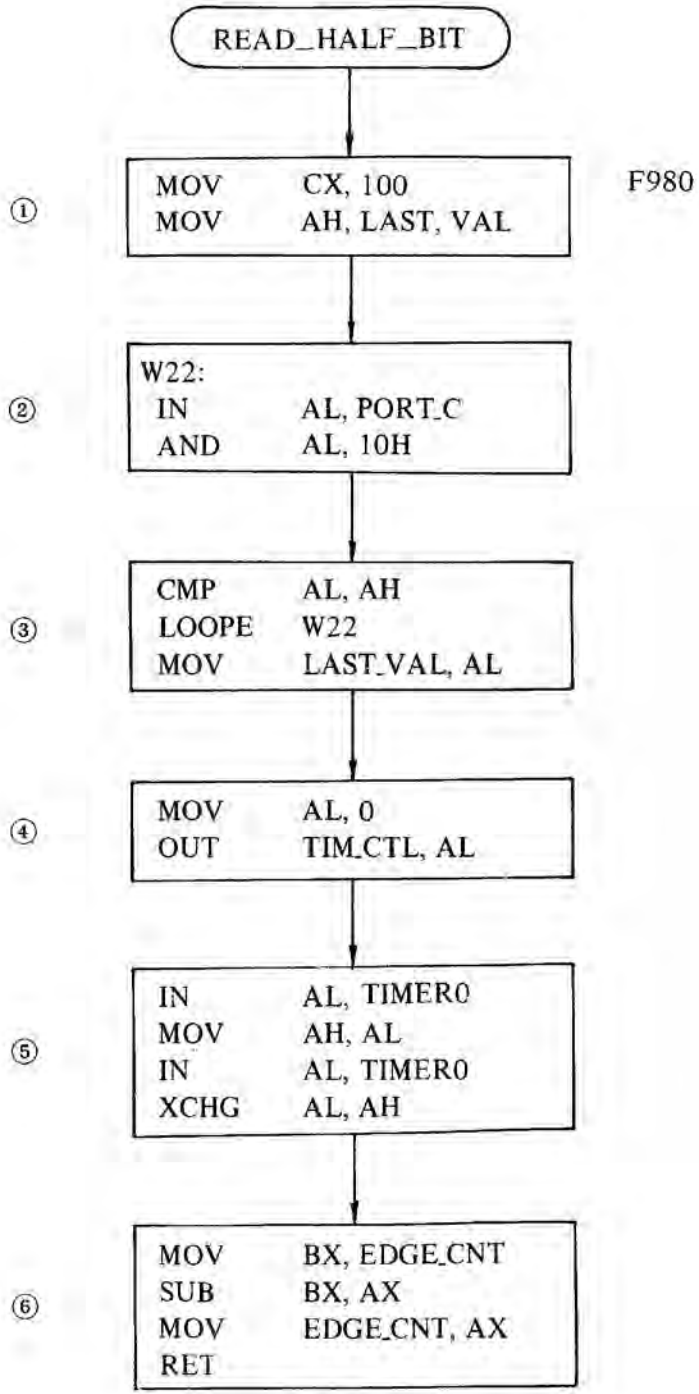
⑥呼叫READ_HALF_BIT副程式二次，讀cassette data一個位元。返回時，若CX暫存器內的值爲0，就表示沒有讀到cassette data，跳入步驟⑨印出131錯誤碼。

⑦進入此步驟表示有讀到cassette data，我們知道cassette data是以時間的長短來代表1或0。我們在步驟④中設定8253的計數器2爲04D3H，所以讀cassette data一個位元的時間不應相距此值太多(最長時間爲0540H，最短時間爲0410H，這裡的時間是指8253的計數)。由READ_HALF_BIT副程式回來的BX暫存器，其內的值表示讀一個位元的計數。這步驟將BX暫存器內的值和最長時間比較，若時間太長，則進入步驟⑨印出131錯誤碼。(若BX暫存器內的值大於MAX_PERIOD，則CF旗號爲0，因爲沒有借位)

⑧將BX暫存器內的值和最短時間比較，若時間太短，則進入步驟⑨印出131錯誤碼。若BX暫存器內的值介於最長時間和最短時間之間，則進入下個測試程式。(若BX暫存器內的值小於MIN_PERIOD，則CF旗號爲1，因爲有借位)

⑨此步驟印出131錯誤碼於螢幕畫面上。錯誤碼131的ASCII碼放在從位址FE4AB起的3個記憶體內，所以在SI暫存器內放入E4ABH，以便P_MSG副程式取131的ASCII碼。因爲要印出3個字，所以在CX暫存器內放入F2L(0003)。

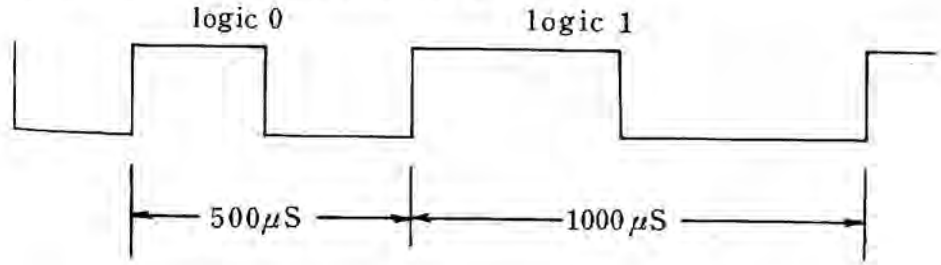
我們繼續討論READ_HALF_BIT副程式。



①在 CX暫存器內放入 100（ 0064H ），以便配合步驟③的 LOOPE指令。將LAST_VAL 值保存在AH 暫存器內。

②讀 cassette data。

③在步驟②中所讀的 cassette data 在AL 暫存器內。LAST_VAL 爲前一個cassette data 值，在AH 暫存器內。這個步驟將這二個值做一比較，若其值相等，就表示 cassette data 沒有變化（即仍是同一電位），則回到步驟②繼續等待 cassette data變換電位。若 CX 暫存器內的值成爲 0（表示 LOOP了100次），cassette data 仍沒有變換電位，則表示 cassette是不良的。若AH暫存器和AL暫存器內的值不相等，則表示 cassette data 變換了電位，此時 CX 暫存器內的值必不爲 0。不論 cassette data 是否變換電位，都要將最後讀到的 cassette data 放到 LAST_VAL 中。由此步驟，我們可以知道 cassette data 是以電位變化的時間來分別 1 或 0。（每個位元有 2 次電位變化）



④此步驟鎖住 8253計數器 2 的瞬間計數，以便下個步驟讀出此值。

⑤讀 8253 計數器 2 的計數，到AX暫存器內。

⑥EDGE_CNT 內的值爲上一次讀半位元（ half bit ）的計數，將BX暫存器內的值減去 AX暫存器內的值，就得到這次讀半位元的計數（結果放在 BX 暫存器內）。最後將讀到電位變換的計數瞬間值放到EDGE_CNT內，然後返回到呼叫此副程式的程式去。

TEST. 14 磁碟機測試(Diskette Attachment TEST)

E55C

①
```
F9:
 MOV     AL, FCH
 OUT     INTA01, AL
```

②
```
 MOV     AL, BYTE PTR EQUIP_FLAG
 TEST    AL, 01H
 JNZ     F10
 JMP     F22
```

③
```
F10:
 MOV     AL, BCH
 OUT     INTA01, AL
```

④
```
 MOV     AH, 0
 INT     13H
 TEST    AH, FFH
 JNZ     F13
```

⑤
```
 MOV     DX, 3F2H
 MOV     AL, 1CH
 OUT     DX, AL
```

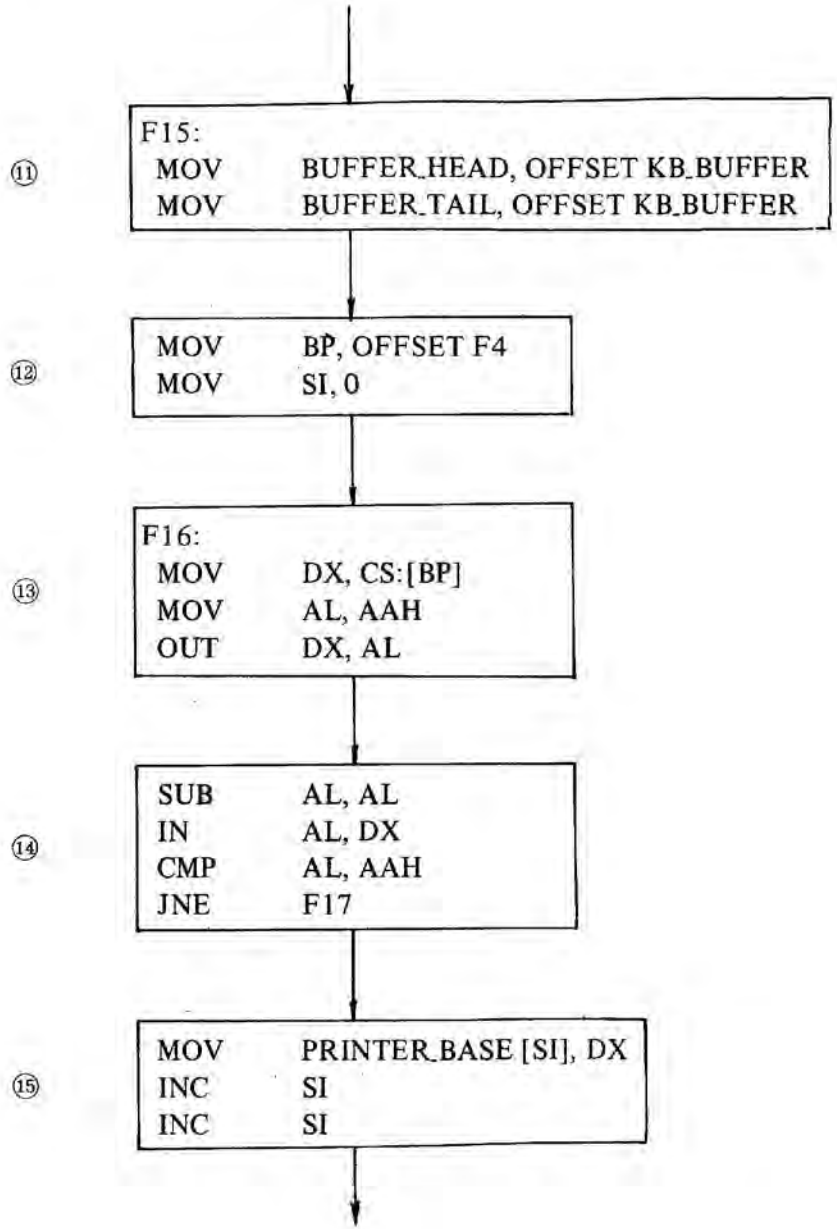
⑥
```
 SUB     CX, CX
F11:
 LOOP    F11
F12:
 LOOP    F12
```

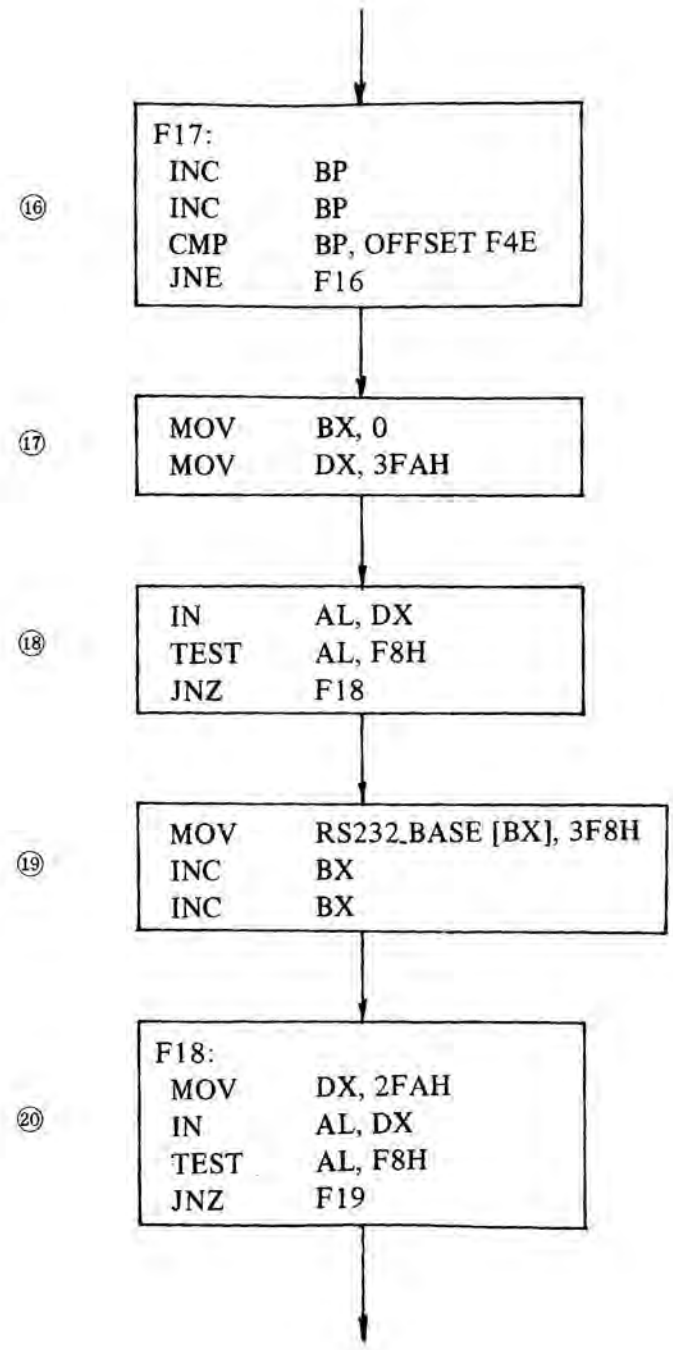
⑦
```
 XOR     DX, DX
 MOV     CH, 1
 MOV     SEEK_STATUS, DL
 CALL    SEEK
 JC      F13
```

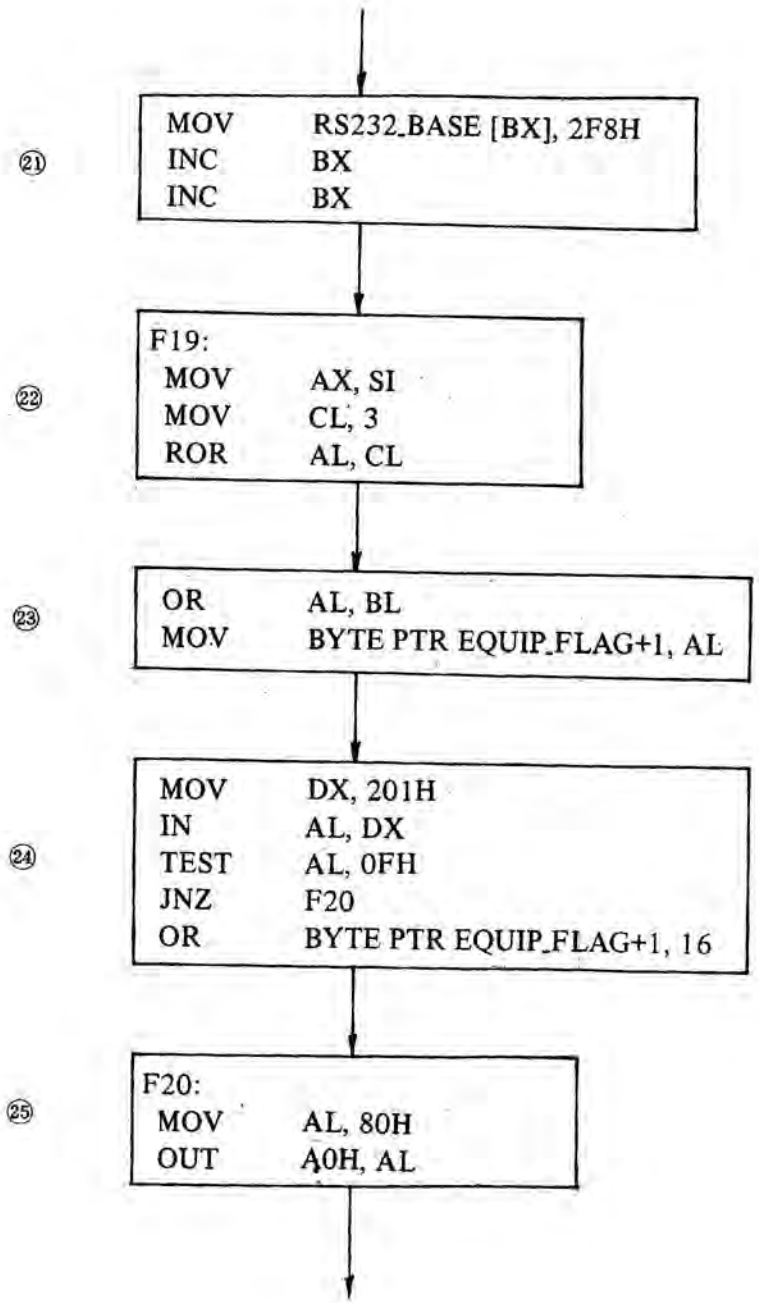
⑧
```
 MOV     CH, 34
 CALL    SEEK
 JNC     F14
```

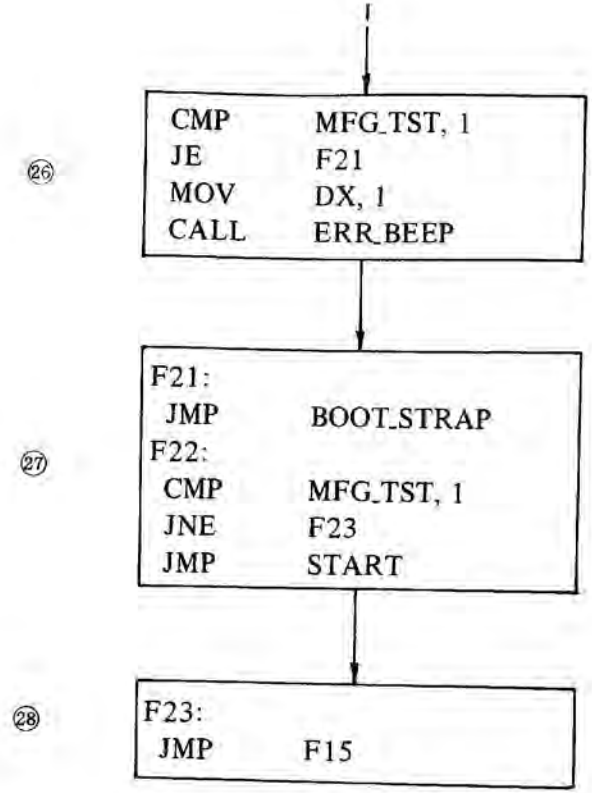
⑨
```
F13:
 MOV     SI, OFFSET F3
 MOV     CX, F3L
 CALL    P_MSG
```

⑩
```
F14:
 MOV     AL, 0CH
 MOV     DX, 3F2H
 OUT     DX, AL
```


⑪
F15:
MOV BUFFER_HEAD, OFFSET KB_BUFFER
MOV BUFFER_TAIL, OFFSET KB_BUFFER
⑫
MOV BP, OFFSET F4
MOV SI, 0
⑬
F16:
MOV DX, CS:[BP]
MOV AL, AAH
OUT DX, AL
⑭
SUB AL, AL
IN AL, DX
CMP AL, AAH
JNE F17
⑮
MOV PRINTER_BASE [SI], DX
INC SI
INC SI


⑯
F17:
INC BP
INC BP
CMP BP, OFFSET F4E
JNE F16
⑰
MOV BX, 0
MOV DX, 3FAH
⑱
IN AL, DX
TEST AL, F8H
JNZ F18
⑲
MOV RS232_BASE [BX], 3F8H
INC BX
INC BX
⑳
F18:
MOV DX, 2FAH
IN AL, DX
TEST AL, F8H
JNZ F19





①將FCH寫入8259的中斷遮罩暫存器內。這使得8259可以接受IRQ0（Timer）和IRQ1（鍵盤）的中斷要求。

②這個步驟測試DIP開關1的位元1（即資料滙流排（data bus）的位元0）是否爲0。若其爲0（ZF旗號爲1），則表示沒有磁碟機，不做磁碟機的測試，直接進入步驟㉗。若其爲1（ZF旗號爲0），則表示有磁碟機，進入下個步驟進行磁碟機的測試。

③將BCH寫入8259的中斷遮罩暫存器內。這使得8259還可以接受IRQ6（磁碟）的中斷要求。

④在AH暫存器內放入0，然後呼叫磁碟機副程式（INT 13H），這個動作重置NEC 765磁碟機界面卡。TEST 指令測試是否NEC 765是好的，若其爲不良（ZF旗號爲0），則進入步驟⑨印

出601錯誤碼。若其爲良好，則進入下個步驟。（磁碟機副程式相當大；另一章討論）

⑤將1CH寫到磁碟機界面卡上的I/O埠3F2H處，使得磁碟機A被開啓，並且啓動INT和DMA要求信號。讀者可以由技術手册中磁碟機界面卡的介紹，得知這步驟的用意，熟悉磁碟機界面卡的綫路圖的讀者，更可以知道I/O埠3F2H只是一個74LS273而已。

⑥這個步驟等待一秒鐘，爲了使馬達起動並旋轉平穩。

⑦SEEK副程式爲磁碟機副程式（INT 13H）的其中一部分，進入SEEK副程式時，DL暫存器內的值選擇磁碟機，DH暫存器內選擇磁頭，CH暫存器內的值選擇磁軌（track），所以此步驟將磁碟機A的磁頭移至磁軌1的地方。SEEK副程式主要是移動磁頭至特定的磁軌。如果返回時，CF旗號爲1，則表示磁頭無法移至磁軌1，此時要進入步驟⑨印出601錯誤碼。如果返回時，CF旗號爲0，則表示磁頭已移至磁軌1，進入下個步驟。

⑧此步驟將磁頭移至磁軌34處，如果返回時，CF旗號爲0，表示磁頭已移至磁軌34處，進入步驟⑩。如果返回時，CF旗號爲1，則表示無法將磁頭移至磁軌34處，進入步驟⑨。

⑨此步驟印出601錯誤碼，601這三個字的ASCII碼存放在F3（E4AE）起的3個記憶體內。F3L爲3，表示印出3個字。

⑩這步驟將0CH寫到磁碟機界面卡上的I/O埠3F2H處，這樣關閉磁碟機A，並且停止磁碟機A的馬達。

⑪我們知道鍵盤的緩衝區（buffer）可容納16個位元組，而此緩衝區從實際位址0041EH開始（分段值0040H，間距值001EH）。Buffer_HEAD（0041AH）指出緩衝區的起始，Buffer_TAIL指出緩衝區的結束。由Buffer_HEAD和Buffer_TAIL內的值就可以知道鍵盤緩衝區內，現正有多少個鍵碼尚未被取用。由鍵盤上鍵入的鍵碼先暫時放到此緩衝區內，等待程式取用。此步驟將Buffer_HEAD和Buffer_TAIL都填入OFFSET KBD_BUFFER（001EH），這表示著鍵盤緩衝區內沒有任何鍵碼能被取用。

⑫OFFSET F4（E4B1H）起的三個字組值，存放著印表機（printer）的I/O埠值（IBM PC可以接受三個印表機），將OFFSET F4放到BP暫存器內，是爲了在下個步驟中，取出印表機的I/O埠的值。在SI暫存器內放入0，是爲了在步驟⑮中，儲存印表機的I/O埠值時做指標用。

⑬取出LPT1的I/O埠值放到DX暫存器內，然後將AAH寫到LPT1的I/O埠去，其實這個I/O埠只是一個Latch IC而已。

⑭首先清除AL暫存器內的值，然後再讀出I/O埠的值到AL暫存器內，然後再測試此值是否爲AAH，若不是，則ZF旗號爲0，跳入步驟⑯。若讀出的值仍是AAH，則認爲LPT1存在，進入步驟⑮保存此I/O值。

⑮將此I/O埠值存放在printer_BASE內。printer_BASE在段落值爲0040H，間距值爲0008H起的4個字組記憶體內。在步驟⑫中，首先設定SI暫存器爲0，此時將I/O埠的值存放到printer_BASE的第一個字組處，然後將SI暫存器內的值加2，以便下個I/O埠值存放到下一個字組處。

⑯將BP暫存器內的值加2，以便取到下一個I/O埠的值。然後將BP暫存器內的值和OFFSET F4E比較（OFFSET F4E是一個OFFSET值，比OFFSET F4多6個位元組，也就是三個字組，這三個字組存放著印表機的三個I/O埠值。），若BP值已等於OFFSET F4E，則表示三個印表機都已測試完了，進入下個步驟。若BP值比OFFSET F4小，則回到步驟⑬，繼續測試下個印表機。必須注意的是，若系統上只有LPT2，而沒有LPT1，則此時LPT2的I/O埠值是放到printer_BASE的第1個字組的位置，而不是第2個字組位址，這是因爲沒有經過步驟⑮，將SI

暫存器內的值加2的緣故。

⑰將BX暫存器清除爲0，此BX暫存器的功用，有如測試印表機時SI暫存器的功用，然後將3FAH放到DX暫存器內。3FAH是COM1的I/O埠值（8250的中斷辨認暫存器）。

⑱在重置後，8250的中斷辨認暫存器（interrupt identification register）內的值將是01H。此步驟讀出8250的中斷辨認暫存器內的值，然後和F8H做比較（TEST指令的格式爲（TEST來源，目的），這道命令並不改變任何值，只影響旗號），若結果不爲0，此時ZF旗號爲0，進入步驟⑳測試COM2。若結果爲0，則認爲COM1存在，進入下個步驟。

⑲將3F8H存放到RS232_BASE的第1個字組中，然後將BX暫存器內的值加2。

⑳同樣的，此步驟測試COM2的8250的中斷辨認暫存器（2FAH是COM2的I/O埠值）。若COM2不存在（ZF旗號爲0），則進入步驟㉒。若COM2存在，進入下個步驟。

㉑將2F8H存放到RS232_BASE去。然後將BX暫存器內的值加2。同樣地，若只有COM2存在，而COM1不存在，則COM2的I/O埠值（2F8H）是放到RS232_BASE的第1個字組中，而非第2個字組，這是因爲沒有經過步驟⑲的緣故。

㉒進入此步驟的SI暫存器內的值爲印表機個數的2倍(即2*N，N=No. of printer)，所以SI暫存器內的可能有4種（0，2，4，6）；先將這個值保存到AX暫存器內（其實AH暫存器內的值爲0，AL暫存器才眞正保存此值），然後右旋轉AL暫存器3次，所以此時AL暫存器的位元7，位元6和位元5的值指出印表機的個數。

AL暫存器			No. of printer
位元7	位元6	位元5	
0	0	0	0
0	1	0	1
1	0	0	2
1	1	0	3

㉓進入此步驟的BL暫存器，其內的值爲RS232的個數的2倍（即2*N，N=No. of RS232），所以BL暫存器的值可能有3種（0，2，4）。OR指令將此值放到AL暫存器內的位元2，位元1和位元0內。

AL暫存器			No. of RS232
位元2	位元1	位元0	
0	0	0	0
0	1	0	1
1	0	0	2

所以此時AL暫存器的位元7，位元6和位元5指出印表機的個數，位元2，位元1和位元0指出RS232的個數。然後將此AL暫存器內的值保存在EQUIP_FLAG的高位元組處。（EQUIP_FLAG是一個字組。位元組PTR指出這時只使用一個位元組，EQUIP_FLAG+1指出爲高位元組）

㉔此步驟測試是否有遊戲控制卡，201H是遊戲控制卡的I/O埠值。若沒有遊戲控制卡（ZF旗號爲0），進入下個步驟。若有遊戲控制卡（ZF旗號爲1），則將EQUIP_FLAG的高位元組的位元4設定成1，然後進入下個步驟。

㉕此步驟起動NMI正反器（NMI Flip-Flop）。我們知道8088的NMI信號，由一個LS08控制著。而LS08的其中一個輸入脚是由一個正反器的輸出脚而來。當此正反器的輸出脚爲低電位時，LS08的輸出也爲低電位，換句話說8088 NMI信號一直爲低電位，此時就不可能有NMI中斷要求產生。當正反器的輸出脚爲高電位時，才可能有NMI中斷要求產生。此正反器在I/O埠A0H處，其輸出由資料滙流排的位元7控制著。此步驟的位元7爲1，所以此時允許有NMI信號產生。

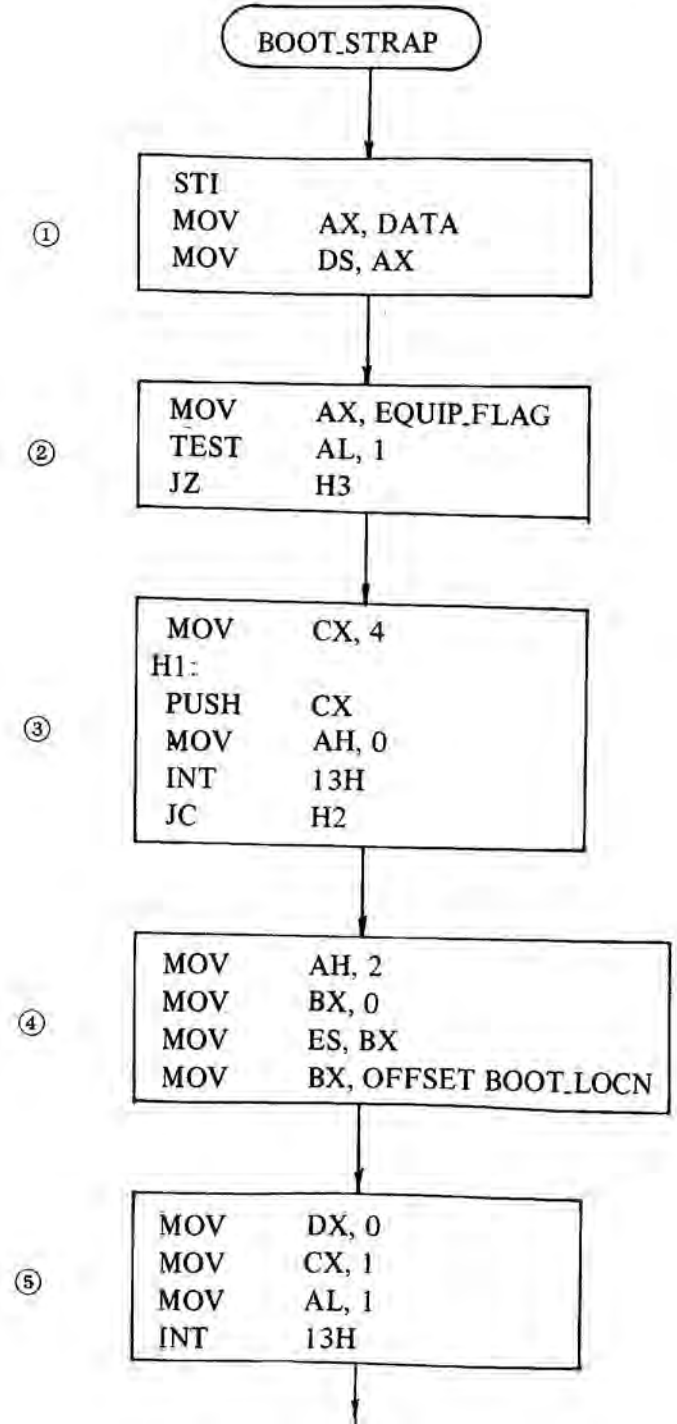
㉖測試MFG_TST內的值是否爲1，若爲1，則進入下個步驟，若不爲1，則使喇叭發出一短聲；這就是開機後，磁碟機啓動前，喇叭會發出一短聲的原因。

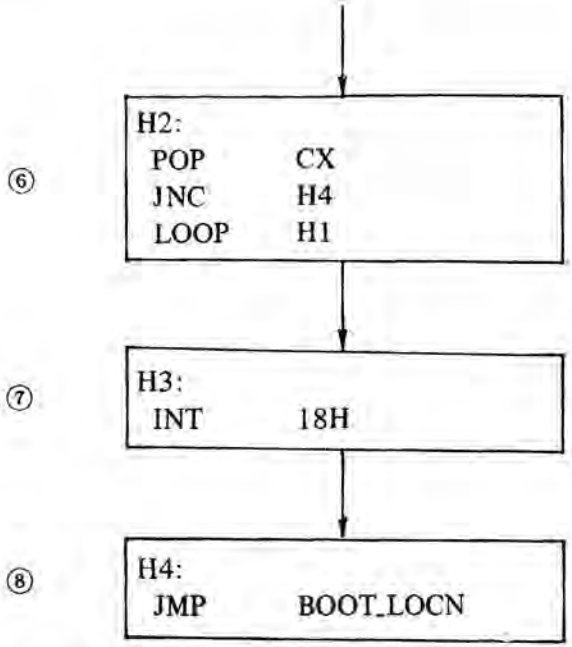
㉗喇叭發出一短聲後，進入此步驟，直接跳入BOOT_STRAP副程式內（INT 19H）。進入此步驟的F22是由步驟②而來，這時並沒有磁碟機存在。測試MFG_TST內的值是否爲1，若爲1，則進入START副程式中（START副程式也就是RESET副程式），若不爲1，進入下個步驟。

㉘進入步驟⑪，測試printer和RS232，此時還是會跳入BOOT_STRAP副程式中，只是喇叭不會響而已。請讀者自己TRACE一次。

INT 19 啓動載入程式(BOOT STRAP LOADER)

BIOS執行到這裡時，大部分都已測試完成，此時要將控制權交給DOS了。若磁碟機內沒有磁片，或磁碟機的門沒有關上，就會跳至BASIC中。





①首先設定IF旗號爲1，表示8088此時可以接受中斷信號。在DS暫存器內放入DATA（0040H），是因爲EQUIP_FLAG在分段值爲0040H的記憶體範圍內。

②測試DIP開關1的位元1是否爲0，若其爲0（ZF旗號爲1），則進入步驟⑦，跳入BASIC中。若其爲1，則表示有磁碟機，進入下個步驟。

③在CX暫存器內放入4，是表示有4次重置磁碟機的機會，然後將CX暫存器內的值保存在堆疊中。在進入磁碟機副程式時，若AH暫存器內的值爲0，就表示爲重置磁碟機。若返回時，CF旗號爲1，就表示有錯誤產生，進入步驟⑥。若重置成功（CF旗號爲0），則進入下個步驟。

④在AH暫存器內放入2，表示要讀磁片上的資料。在讀磁片上的資料時，要先設定好所讀的資料要放在何處（分段值由ES暫存器內的值決定，間距值由BX暫存器內的值決定）。此步驟設定分段值爲0，OFFSET值爲7C00H。

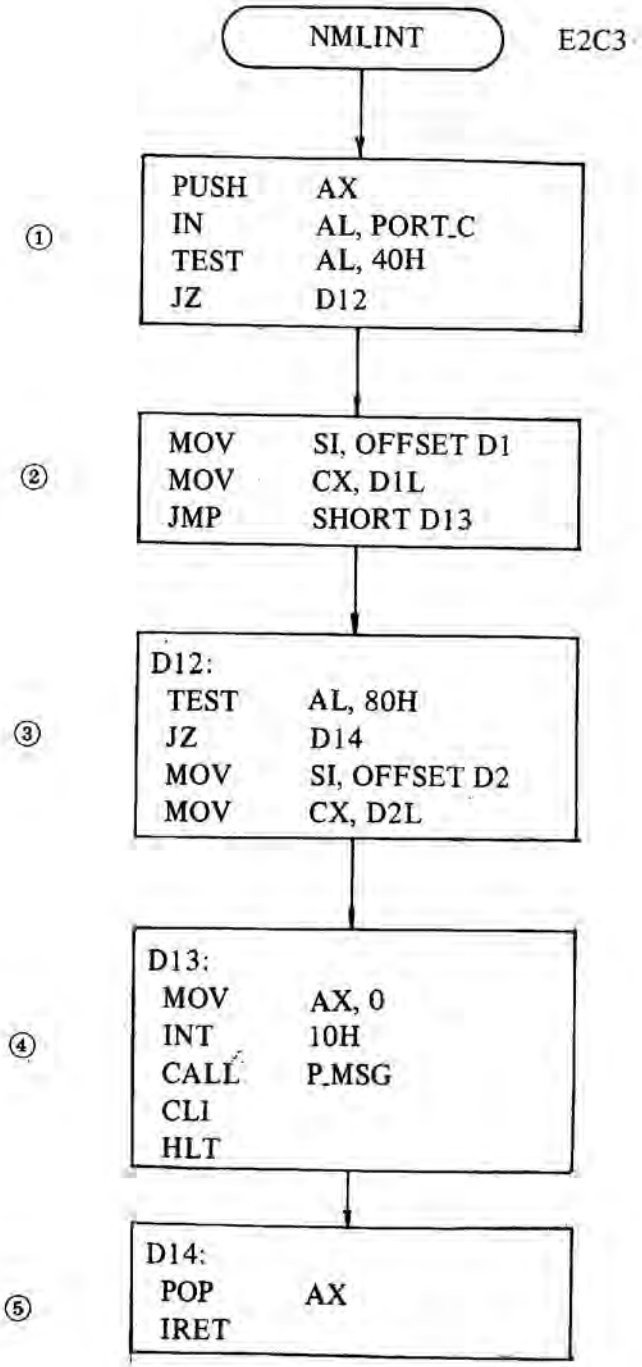
⑤在讀磁片上的資料時，一定要設定好磁碟機（DRIVE），磁頭（HEAD），磁軌（TRACK），磁區（SECTOR）和共要讀若干磁區。此步驟設定磁碟機0，磁頭0（DH和DL暫存器均爲0），磁軌0，磁區1（CH暫存器爲0，CL暫存器爲1），共讀1個磁區（AL暫存器爲1）。然後呼叫INT 13H，讀磁片上的512位元組的資料到實際位址07C00H處。

⑥取出CX暫存器的值後，首先測試讀磁片上的資料時是否有錯誤產生，若有錯誤產生（CF旗號爲1），則回到步驟③（若CX暫存器內的值已減至0，則進入下個步驟，跳入BASIC中），若沒有錯誤產生（CF旗號爲0），則進入步驟⑧，跳入所讀的512位元組中。

⑦早在前面的程式中，已預先設定好INT 18H內爲BASIC進入點的位址（segment值爲F600H，OFFSET值爲0），這裡呼叫INT 18H，也就是進入BASIC中。

⑧我們所讀到的512位元組是放到實際位址07C00H起的512個位元組記憶體內。這裡的JMP指令直接跳入此段程式內去執行命令。

ROM BIOS的測試部分已測試完畢，控制權也已經交給DOS或BASIC。在這裡我們還要討論處理NMI中斷信號的副程式。處理NMI中斷信號的副程式的起始位址在實際位址00008H到0000BH內，也就是在中斷向量表的pointer 2內。



①首先將 AX 暫存器內的值保存起來，然後測試 8255 PORT_C的位元 6 是否爲 0 。(8255 PORT_C的位元 6 爲 I/O CHECK，若其爲 1 ，即表示 I /O通道上有滙流排錯誤產生)若其爲 0 ，即表示並非 I /O 通道的滙流排錯誤，進入步驟③測試是否爲主機板的記憶體有滙流排錯誤產生。若 8255PORT_C的位元 6 爲 1，進入下個步驟。

②進入此步驟即表示I/O通道上有滙流排錯誤產生，這裡在螢幕畫面上印出“ parity check 2 ”字樣(共14 個字)。“parity check 2 ”這 14個字的ASC II碼放在位址FE219H (OFFSET D1) 起的 14 個記憶體內，在CX暫存器內放入 D1L (000EH)是因爲要印出 14 個字。

③這步驟測試是否爲主機板上的記憶體產生滙流排錯誤。若 8255 PORT_C的位元 7 爲 1 (即此時 AL 暫存器內的位元 7)，則在螢幕畫面上印出“ parity check 1 ”的字樣。“ parity check 1 ”這 14 個字的 ASC II 碼放在位址FE227H(OFF-SET D2) 起的 14 個記憶體內，在 CX 暫存器內放入 D2 L (000EH) 表示要印出 14 個字；若 8255 PORT_C的位元 7 爲 0 ，跳入步驟⑤。

④步驟②和步驟③只是做要印字的準備工作，此步驟才眞正印字於畫面上。進入 INT 10H時，若 AH 暫存器內的值爲 0，即表示清除畫面。然後呼叫 P_MSG副程式將所要印的字印在畫面上。印完後，清除 IF 旗號爲 0 ，並停止系統。

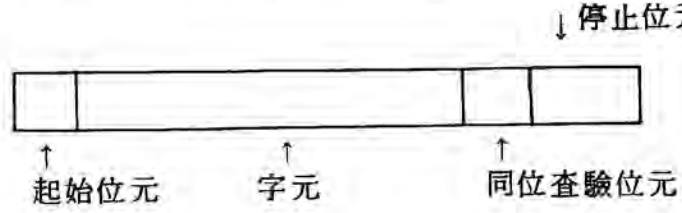
⑤進入此步驟是因爲沒有 I / O 通道滙流排錯誤和主機板上的滙流排錯誤產生。取回 AX 暫存器的初值，然後回到被中斷的程式去。

第三章

RS-232C

RS-232C是一種常用的通訊方式。終端機和主機的連接，PC和PC的連接都是利用RS-232C。終端機和PC要接上調變解調器（Modem）時，也要用RS-232C，可見RS-232C在電腦的通訊上相當重要。

RS-232C是一種非同步（asynchronous）的傳輸方式。字元（characters）在傳輸時，前有一個起始位元（start bit），後有一個同位查驗位元（parity check bit），再緊接著停止位元（stop bit）。它的格式是這樣的：



其中字元可以爲5，6，7或8個位元；

同位查驗位元可以爲偶數、奇數或無；

停止位元可以爲1，$1\frac{1}{2}$或2個位元。

IBM PC不但在硬體上提供了二組RS-232C（ COM1和COM2 ），而且在ROM BIOS內提供RS-232C的副程式(INT 14H)，以便使用者使用RS-232C。本章就是介紹這個副程式(INT 14H)，因爲IBM PC的RS-232C是由INS8250來控制，所以在討論這個副程式前，必須先介紹INS8250 。讀者了解INS8250後，就能夠對RS-232C的傳輸動作有所知曉。

8250是一個可程式化（ programmable ）的非同步通訊元件(asynchronous communication element,ACE)。ACE在傳送時，將平行進入的資料轉換成串列的資料輸出，在接收時，將串列進入的資料轉換成平行的資料。串列的格式(format)如上所述。

整個的通訊動作由使用者寫入資料到8250的暫存器內，由8250的暫存器來控制。8250各個暫存器和位址的關係如下表：

DLAB	A2	A1	A0	暫存器
0	0	0	0	Receiver buffer (read) transmitter holding register (write)
0	0	0	1	interrupte enable
X	0	1	0	interrupt identification (read)
X	0	1	1	Line control
X	1	0	0	Modem control
X	1	0	1	Line status
X	1	1	0	Modem status
X	1	1	1	None
1	0	0	0	Divisor Latch (least significant byte)
1	0	0	1	Divisor Latch (most significant bye)

註：除數鎖定存取位元是線控制暫存器（ Line Control Register)的最高有效位元(the most significant bit)。

我們現在就來介紹各個暫存器的功用。

① TX和RX暫存器 ：暫時儲存資料。在傳送時，將要傳送的資料寫入這個暫存器內。在接收時，串列進入的資料暫時放在這個暫存器內，以便被讀取。

② 線控制暫存器：

(a) 位元0和位元1決定傳送和接收的字元的長度。

位元1	位元0	字元長度
0	0	5位元
0	1	6位元
1	0	7位元
1	1	8位元

(b) 位元2決定停止位元的數目。若位元2爲0，則停止位元的數目爲1，若位元2爲1，則停止位元的數目和字元的長度有關。

字元長度	停止位元的個數
5位元	$1\frac{1}{2}$
6，7，8位元	2

(c) 位元3是同位位元（ parity bit ）。當位元3爲1，在傳送時，會產生一個同位位元。在接收時，會檢查同位位元。

(d) 位元4是選擇偶數同位位元，此位元有效的條件是位元3爲1。當位元3爲1時，位元4的功用如下表：

位元4	同位位元的查驗方式
0	奇數
1	偶數

(e) 位元5是stick同位位元，當位元3爲1，位元5爲1時，則同位位元查驗的形式剛好如位元4所指定的相反。

(f) 位元6是設定中止控制位元，當位元6爲1，則串列輸出腳（SOUT）被強迫爲0，直到位元6變成0爲止。在電腦通訊系統中，這個特點使得CPU 能夠選擇另一個終端機做爲通訊的對象。

(g) 位元7是除數鎖定存取位元（ Divisor Latch Access Bit，DLAB）要進入傳輸率產生器（ Baud rate generator ）的除數鎖定暫存器（ Divisor Latches ）時，DLAB必須爲1，進入TX和RX暫存器及中斷啓動暫存器（ interrupt enable register ）時，DLAB必須爲0。

③ 可程式化傳輸率產生器（ Programmable Baud Rate Generator ）：所謂傳輸率（ Baud Rate ）是每秒傳送的位元數，單位爲b／s。8250可以有二種不同的振盪頻率–1.8432MHz和3.072MHz。不同的頻率有不同的除數。在IBM PC 上是使用1.8432MHz的振盪頻率，所以在此，我們列出1.8432MHz幾個常見的傳輸率和除數的關係。

傳輸率	110	150	300	600	1200	2400	4800	9600
除　數	1047	768	384	192	96	48	24	12

將傳輸率相對的除數寫入除數鎖定暫存器中即可得到想要的傳輸率（除數是十六進位制）。

④ 線狀態暫存器（ Line Status Register ）：這個8位元暫存器提供了通訊的狀態。

(a) 位元0是資料備妥指示器（ Data Ready（DR）indicator ），當位元0爲1，就表示串列輸入的資料已經進入接收緩衝區（ Receive Buffer ）中。當CPU讀取接收緩衝區內的資料後，位元0變成0，CPU 也可以直接寫入0於位元0中。

(b) 位元1是超位錯誤指示器（ Overrun Error（OE）indicator ）。當位元1爲1時，表示CPU 尚未讀取接收緩衝區內的資料前，下一個字元資料已經進入接收緩衝區中，如此即破壞了前一個資料。當CPU讀取線狀態暫存器內的資料時，位元1會變成0。

(c) 位元2是同位位元錯誤指示器（Parity Error（PE)indicator ），當位元2爲1時，表示同位查驗不正確。當CPU讀取線狀態暫存器內的資料時，位元2會變成0。

(d) 位元3是架構錯誤指示器（ Framing Error（FE）indicator ），當位元3爲1時，表示所測試到的停止位元不正確。

(e) 位元4是中止中斷指示器（ Break Interrupt （BI） indicator ）。當位元4爲1時，就表示串列輸入的資料都是0，並且其時間超過整個字組的時間（起始位元＋資

料位元＋同位位元＋停止位元）。

(f) 位元5是TX暫存器空乏指示器（Transmitter Holding Register Empty（THRE）Indicator），當位元5爲1時，表示TX器是空的，此時8250正等待著CPU重新寫入一個資料以便傳輸，同時8250會發出一個中斷要求信號給CPU（如果中斷起動暫存器內的TX暫存器空乏，起動中斷位元是1的話）。當CPU將資料寫入TX暫存器後，位元5會變成0。

(g) 位元6是傳送移位暫存器空乏指示器（Transmitter Shift Register Empty（TSRE）indicator）。當位元6爲1時，就表示傳送移位暫存器內沒有資料要傳送出去。當TX暫存器傳送資料到傳送移位暫存器內時，位元6會變成0。位元6不能被寫入，只能被讀取。

(h) 位元7永遠爲0。

⑤ 中斷辨認暫存器（Interrupt Identification Register，IIR）：IIR內的值用來指示那一層的中斷要求尚未被CPU處理（pending），8250有4個中斷層，如下所示。

priority 1	Receiver Line Status
priority 2	Receiver Data Ready
priority 3	Transmitter Holding Register Empty
priority 4	Modem Status

(a) 位元0：當位元0爲0時，就表示有中斷要求尚未被處理。當位元0爲1時，就表示沒有中斷要求是在處理中。

(b) 位元1和位元2：這二個位元用來指出是那一個中斷層被處理。

位元2	位元1	中斷層
1	1	priority 1
1	0	priority 2
0	1	priority 3
0	0	priority 4

(c) 位元3到位元7永遠是0。

⑥ 中斷啓動暫存器：這個暫存器的前4個位元（位元0到位元3），相對於4個層的中斷要求，當某個位元爲1時，其相對的中斷才會被啓動。

(a) 位元0：當位元0是1時，接收資料已到的中斷要求才會有效。

(b) 位元1：當位元1爲1時，TX暫存器空乏的中斷要求才會有效。

(c) 位元2：當位元2爲1時，接收線狀態的中斷要求才會有效。

(d) 位元3：當位元3爲1時，調變解調器狀態（Modem status）的中斷要求才會有效。

⑦ 調變解調器控制暫存器：這個暫存器控制和調變解調器相接的界面。

(a) 位元0：位元0控制著資料端備妥（Data Terminal Ready）這個輸出腳。當位元0爲1，$\overline{\text{DTR}}$ 輸出腳爲0

，當位元0爲0，$\overline{\text{DTR}}$輸出脚爲1。

(b) 位元1：位元1控制著傳輸要求（Request to send（$\overline{\text{RTS}}$）這個輸出脚。位元1的值和$\overline{\text{RTS}}$輸出脚的電位相反。

(c) 位元2：位元2控制著OUTPUT1（$\overline{\text{OUT1}}$）這個輸出脚。$\overline{\text{OUT1}}$的輸出電位和位元2的值相反。

(d) 位元3：位元3控制著OUTPUT2（$\overline{\text{OUT2}}$）這個輸出脚。$\overline{\text{OUT2}}$的輸出電位和位元3的值相反。

(e) 位元4：當位元4爲1時，8250進入診斷模式(diagnostic mode）。在診斷模式中，傳送部份的輸出連接到接收部份的輸入，所以傳送的資料會馬上被接收到。CPU可以利用這個模式測試8250是否良好。

(f) 位元5到位元7永遠爲0。

⑧ 調變解調器狀態暫存器（Modem Status Register）：這個8位元暫存器提供了和調變解調器連接的控制線狀態。

(a) 位元0：位元0是Delta Clear to Send(DCTS) indicator。當$\overline{\text{CTS}}$這個輸入脚的狀態改變時，位元0會變成1。

(b) 位元1：位元1是Delta Data Set Ready（DDSR）indicator。當$\overline{\text{DSR}}$這個輸入脚的狀態改變時，位元1會變成1。

(c) 位元2：位元2是Trailing Edge of Ring Indicator（TERI）。當$\overline{\text{RI}}$這個輸入脚由高電位變成低電位時，位元2會變成1。

(d) 位元3：位元3是Delta Received Line Signal Detector(DRLSD)。當$\overline{\text{RLSD}}$這個輸入脚改變狀態時，位元3會變成1。

注意：位元0到位元3變成1時，會產生調變解調器狀態中斷要求（priority 4）。

(e) 位元4：位元4內的值和Clear to send（$\overline{\text{CTS}}$）這個輸入脚的狀態相反。

(f) 位元5：位元5內的值和Data set Ready（$\overline{\text{DSR}}$）這個輸入脚的狀態相反。

(g) 位元6：位元6的值和Ring indicator（$\overline{\text{RI}}$）這個輸入脚的狀態相反。

(h) 位元7：位元7的值和Received line signal Detect（$\overline{\text{RLSD}}$）這個輸入脚的狀態相反。

我們在這裡介紹的8250，只著重其暫存器。它的接脚的功用、電氣特性，請讀者自行查閱8250的資料表。

筆者建議讀者翻開技術手册中的RS-232C的線路圖。讀者可以發現其線路相當簡單，只要了解8250的各個接脚的功用即可了解此線路圖。因爲除了8250外，只有解碼振盪、驅動（Drive）和接收（Receive）四個部份。

在了解了8250各個暫存器後，我們就可以來討論INT 14H這個程式了。

INT 14H程式以進入此程式時，AH暫存器內的值，大約可以區分爲4個部份。

① AH暫存器內的值爲0時，目的是將通訊埠初始化（initialize the communications port），其中AL暫存器內存放著初始化的變數。

(a) AL暫存器的位元1和位元0內的值表示著要通訊的字組長度。

位元1	位元0	字組長度
1	0	7位元
1	1	8位元

(b) AL暫存器內的位元2表示著停止位元（ stop bit ）的數目。

位元2	停止位元
0	1
1	2

(c) AL暫存器內的位元4和位元3表示著 同位位元查驗的形式。

位元4	位元3	同位位元查驗
×	0	無
0	1	奇數
1	1	偶數

(d) AL暫存器內的位元7，位元6和位元5表示著傳輸率。

位元7	位元6	位元5	傳輸率
0	0	0	110
0	0	1	150
0	1	0	300
0	1	1	600
1	0	0	1200
1	0	1	2400
1	1	0	4800
1	1	1	9600

② AH暫存器內的值爲1時，表示要傳送一個資料出去。AL暫存器內放著要傳送的資料。

③ AH暫存器內的值爲2時，表示要接收一個資料，接收到的資料將會放到AL暫存器內。

④ AH暫存器內的值爲3時，表示要了解8250的狀態，執行完此部份後，AH暫存器內將保有線控制的狀態（ Line Control Status ）。

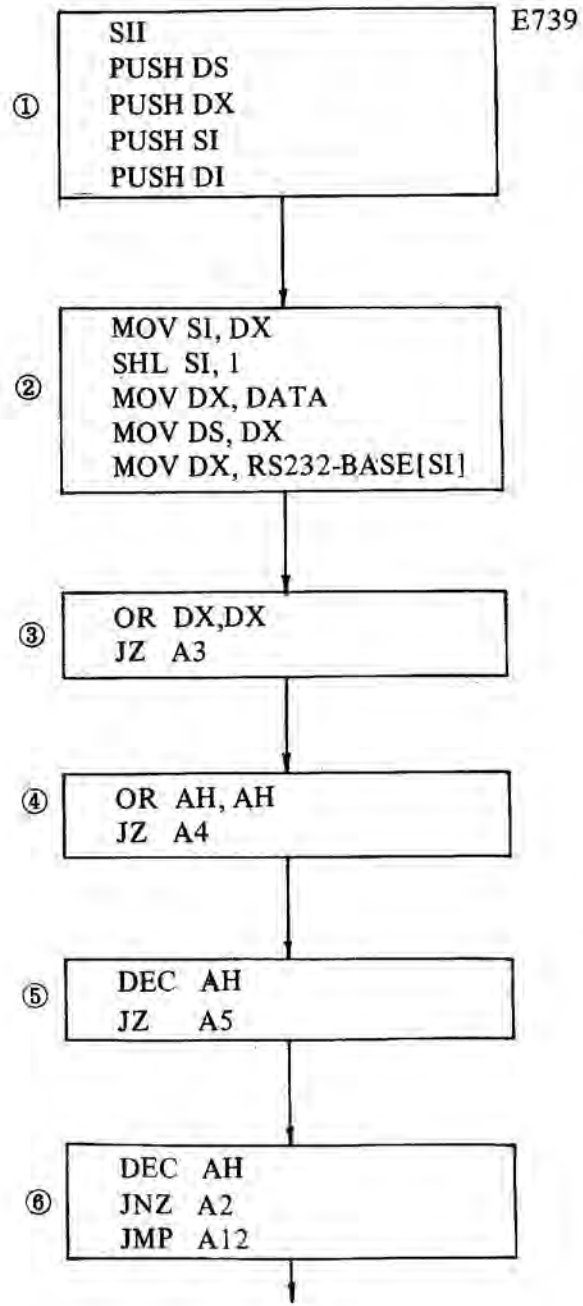
AH暫存器	功　能
bit 7	Time out
bit 6	Transmitter shift register empty
bit 5	Transmitter Holding register Empty
bit 4	Break detect
bit 3	Framing Error
bit 2	Parity Error
bit 1	Overrun Error
bit 0	Data Ready

AL暫存器將保有調變解調器的狀態（ Modem status ）：

AL暫存器	功　能
bit 7	Received line signal detect
bit 6	Ring Indicator
bit 5	Data set ready
bit 4	clear to send
bit 3	Delta Receive line signal Detect
bit 2	Trailing Edge Ring Detector
bit 1	Delta Data set Ready
bit 0	Delta clear to send

值得注意的是，進入INT 14H時，DX暫存器內的值指出是那一個RS-232C port（ 0或1 ）。

我們現在就來看 INT 14H是怎樣寫的：



① 首先設定 IF 旗號爲 1 ，表示CPU此時可以接受中斷信號，然後依序將DS，DX，SI和DI 暫存器內的值保存起來。

② 進入 INT 14H時，DX暫存器內的值指出是那一個RS-232C界面卡。將其乘以2（左移一個位元），這是因爲每一個RS - 232界面卡的I／O 埠值是 2 個位元組。因爲RS232_BASE是在分段值爲DATA的記憶體範圍內，所以在DS 暫存器內放入DATA。然後以SI暫存器（即進入此程式時，DX 暫存器內的值左移一個位元後的值）爲指標，取出RS - 232C 的 I／O 埠值於 DX 暫存器內。

③ 檢查DX暫存器內的值是否爲 0 ，若爲0 （ ZF 旗號爲1 ），則表示並沒有RS - 232C 界面卡，進入步驟⑧，返回呼叫此程式的程式去。若有RS - 232C界面卡，則進入下個步驟。

④ 進入此程式的AH暫存器內的值，指出要做的功能。此步驟檢

查AH暫存器內的值是否爲0，若爲0（ZF旗號爲1），則表示要將通訊埠初始化（initialize communication mode），跳入A4中。若AH暫存器內的值不爲0，則進入下個步驟。

⑤ 先將AH暫存器內的值減1，若結果爲0（ZF旗號爲1），則表示要傳送一個資料出去，跳入A5中。若AH暫存器內的值不爲1，進入下個步驟。

⑥ 再將AH暫存器內的值減1，若結果爲0（ZF旗號爲1），則表示要接受一個資料進來，跳入A12中。若AH暫存器內的值不爲2，進入下個步驟。

⑦ 再將AH暫存器內的值減1，若結果爲0（ZF旗號爲1），則表示要了解8250的狀態，跳入A18中。若AH暫存器內的值也不爲3，則進入下個步驟。

⑧ 進入此步驟表示要返回呼叫INT 14H的程式去。正常的工作做完後，也要經過此步驟返回。DX暫存器內的值爲0（沒有RS-232C界面卡時）或AH暫存器內的值不在0到3範圍內，也要經由此步驟返回，而不做任何事。

我們現在來各別討論各種功能，首先介紹將通訊埠初始化部份（即AH暫存器內的值爲0）。

```
A4:
①      MOV   AH, AL
       ADD   DX, 3
       MOV   AL, 80H
       OUT   DX, AL

②      MOV   DL, AH
       ROL   DL, 1
       ROL   DL, 1
       ROL   DL, 1
```



```
③      ROL   DL, 1
       AND   DX, 0EH
       MOV   DI, OFFSET A1
       ADD   DI, DX

④      MOV   DX, RS232_BASE[SI]
       INC   DX
       MOV   A , CS: [DI]+1
       OUT   DX, AL

⑤      DEC   DX
       MOV   AL, CS:[DI]
       OUT   DX, AL

⑥      ADD   DX, 3
       MOV   AL, AH
       AND   AL, 1FH
       OUT   DX, AL

⑦      SUB   DX,2
       MOV   AL, 0
       OUT   DX, AL
       JMP   SHORT A18
```

① 進入此部份的AL暫存器內的值指出傳輸率、同位位元形式、停止位元數目和字組長度。首先將AL暫存器內的值保存在AH

暫存器內，然後將所得到的RS-232C的I／O埠值加3（此I／O埠值是8250所有I／O埠值最小的一個），以便指向8250的線控制暫存器（line control register）。然後將80H寫入8250的線控制暫存器內，如此使得DLAB爲1，這是因爲要進入除數鎖定暫存器的緣故。

② 將要初始化8250的參數（parameters）轉移到DL暫存器內，然後將DL暫存器左旋轉三次，使得決定傳輸率的三個位元（位元7，位元6和位元5）轉移到位元2，位元1和位元0。

③ 再一次將DL暫存器左旋轉一次，這是因爲除數的值都是二個位元組。此時決定傳輸率的三個位元在位元3，位元2和位元1中。AND DX，0EH指令取出這三個位元。OFFSET A1（E729）起的8個字組中存放著8個除數值。將OFFSET A1加上DX暫存器內的值（此值的位元3，位元2和位元1保有設定傳輸率的碼，其它的位元爲0），然後將結果存放在DI暫存器內，以便用來取出除數。

baud rate code	OFFSET A1 + DX	divisor table
000	E729+0000	1047
001	E729+0002	768
010	E729+0004	384
011	E729+0006	192
100	E729+0008	96
101	E729+000A	48
110	E729+000C	24
111	E729+000E	12

註：DX暫存器爲AL暫存器內的位元7，位元6和位元5，經過左旋轉4次後，再隔離（isolate）此三個位元的值。OFFSET A1＋DX的值放在DI暫存器內，除數表內的值爲十進位。

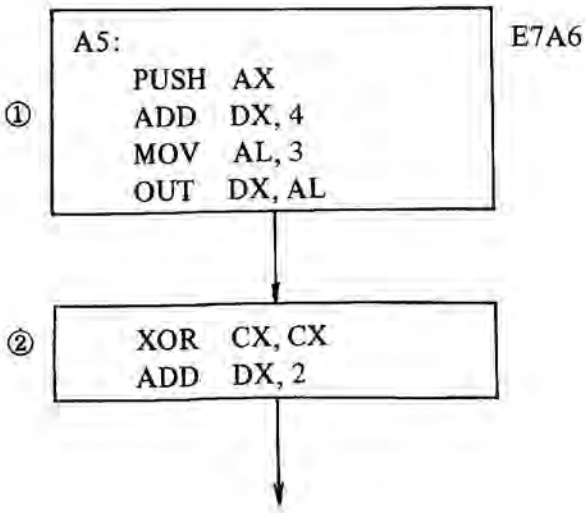
④ 取出RS-232C的I／O埠值，將其加1，以便指向除數鎖定暫存器的高位元組。利用DI暫存器內的值做指標，取出除數表中除數的高位元組，然後將此值寫到8250的除數鎖定暫存器的高位元組（此時DLAB爲1）。

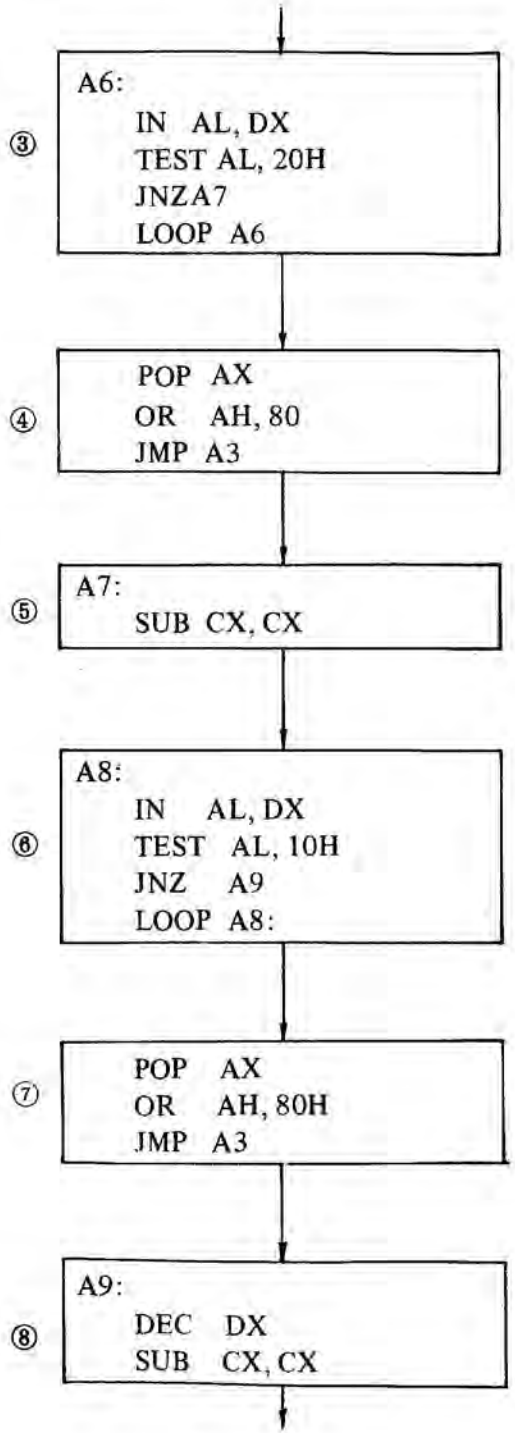
⑤ 將I／O埠值指向除數鎖定暫存器的低位元組，然後利用DI暫存器內的值做指標取出除數表中的低位元組，並寫到8250的除數鎖定暫存器的低位元組。

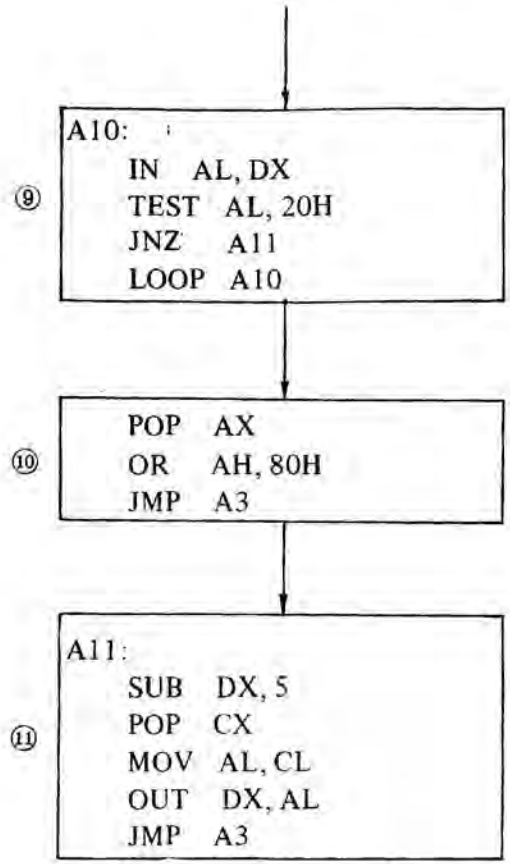
⑥ 將I／O埠值指向線控制暫存器，並且取回initialize的參數。然後剔除傳輸率碼，並將其它的參數寫入線控制暫存器內。如此便設定好了同位位元形式、停止位元個數和字組長度，並將DLAB設定爲0。

⑦ 將I／O埠值指向中斷啓動暫存器（interrupt enable register），並將00H寫入其中，如此使得8250不會接受任何中斷要求。然後跳入A18中，讀取狀態暫存器後，返回呼叫INT 14H的程式去（A18將在第4部份中討論）。

我們繼續看第2部份，傳送一個資料出去。







① 首先將AX暫存器內的值保存在堆疊中，其中AL 暫存器內放著要傳送出去的資料。然後將I／O埠值指向調變解調器控制暫存器，並將3寫入其中，如此使得8250發出Data Terminal Ready（DTR）和Request to Send（RTS）信號。

② 清除CX暫存器內的值，以便配合步驟③的LOOP指令。然後將I／O埠值指向調變解調器狀態暫存器。

③ 首先讀調變解調器狀態暫存器內的值，測試其位元5值是否爲1。若爲1（ZF旗號爲0），則表示Data Set Rready（DSR）信號已經進來了，跳入步驟⑤。若不爲1（ZF旗號爲

1），則表示DSR信號尚未進來，還要繼續讀調變解調器狀態暫存器，直到DSR信號進來爲止，若已經LOOP過64K次後，DSR信號仍沒有進來，則進入下個步驟。

④ 取回AX暫存器的值，然後將AH暫存器的值OR80（注意此80是十進位制，即等於十六進位制的50H），表示time out，再經由A3返回到呼叫INT 14H的程式去。

⑤ 再一次清除CX暫存器的值，以便配合步驟⑦的LOOP指令。

⑥ 讀調變解調器狀態暫存器內的值，測試其位元4值是否爲1。若其爲1（ZF旗號爲0），則表示clear to send（CTS）信號已經進來了，跳入步驟⑧。若不爲1（ZF旗號爲1），則表示CTS信號尚未進來，還要繼續讀調變解調器狀態暫存器，直到CTS信號進來爲止，若已經LOOP過64K次後，CTS信號仍沒有進來，則進入下個步驟。

⑦ 取回AX暫存器的值，然後將AH暫存器的值OR80H，表示time out，再經由A3返回到呼叫INT 14H的程式去。

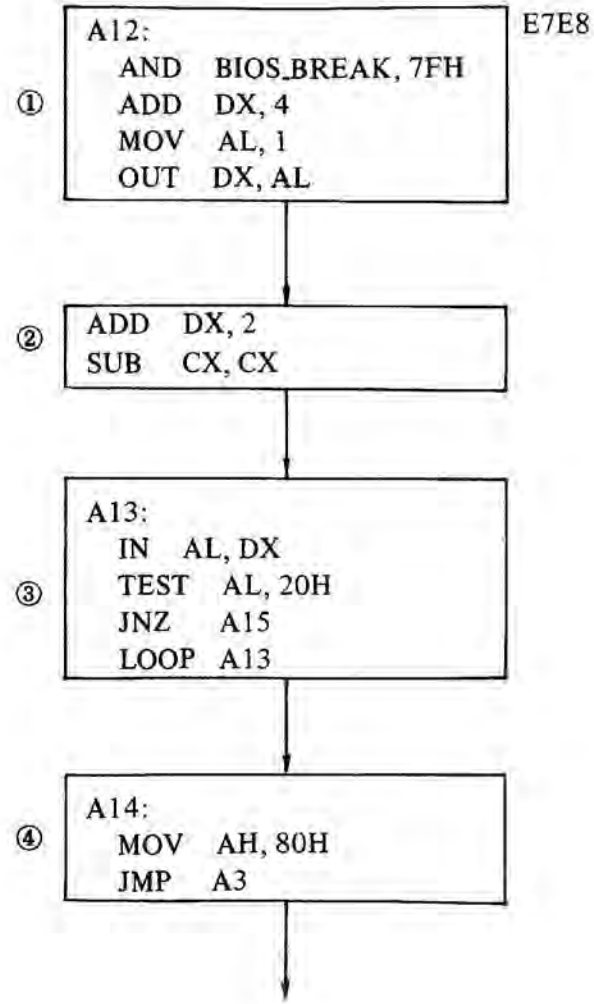
⑧ 將I／O埠值指向線狀態暫存器，並將CX暫存器內的值清除爲0，以便配合步驟⑨的LOOP指令。

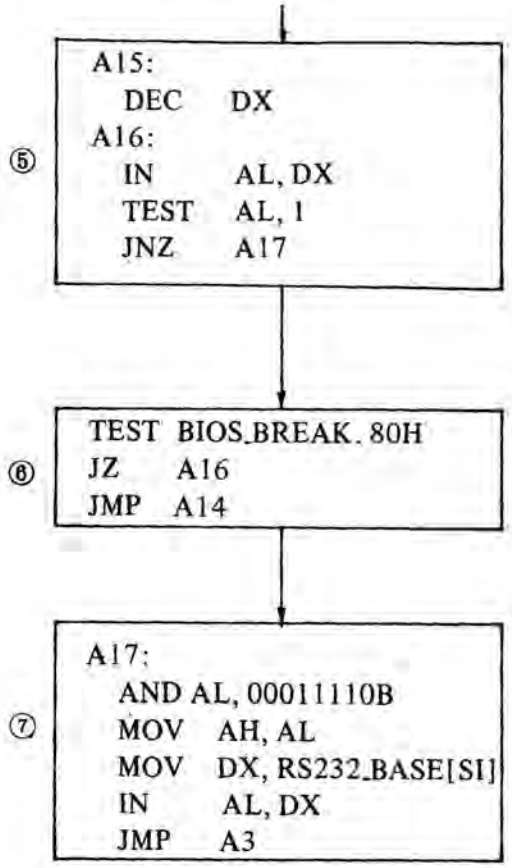
⑨ 讀線狀態暫存器內的值，並測試其位元5值是否爲1，若爲1（ZF旗號爲0），則表示TX暫存器空乏，可以傳送資料了，進入步驟⑪。若不爲1（ZF旗號爲1），則繼續讀線狀態暫存器內的值，直到TX暫存器空乏爲止，若LOOP過64K次後，仍不能得到TX暫存器空乏的指示，則進入下個步驟。

⑩ 取回AX暫存器的值，然後將AH暫存器的值OR 80H，表示time out，再經由A3返回到呼叫INT 14H的程式去。

⑪ 將I／O埠值指向TX暫存器，然後取回要傳送的資料於AL暫存器，並將其寫入TX暫存器內，再經由A3返回到呼叫INT 14H的程式去（將資料寫入TX暫存器後，8250會將這個資料串列輸出去。DLAB在通訊埠初始化時，寫入同位位元形式、停止位元數目和字組長度到線控制暫存器時，已將其設定爲0了）。

我們再來看第3部份，接收一個資料進來。





① 先將BIOS_BREAK的位元7設定爲0，以便在步驟⑥中做測試。將I／O埠值指向調變解調器控制暫存器，然後將1寫入其中，此時8250會發出Data Terminal Ready (DTR)信號。

② 將I／O埠值指向調變解調器狀態暫存器，然後將CX暫存器清除爲0，以便配合下個步驟的LOOP指令。

③ 讀調變解調器狀態暫存器內的值，並測試其位元5的值是否爲1，若爲1（ZF旗號爲0），則表示Data set ready(DSR)已經進來了，進入步驟⑤。若不爲1（ZF旗號爲1），則繼續讀調變解調器狀態暫存器內的值，直到DSR信號進來爲止。若已經LOOP過64K次後，DSR信號仍沒有進來，則進入下個步驟。

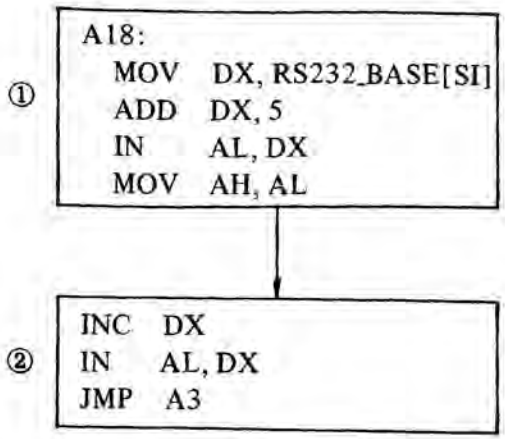
④ 在AH暫存器內放入80H，表示time out，然後返回到呼叫INT 14H的程式去。

⑤ 將I／O埠值指向線狀態暫存器，然後讀其內的值，並測試其位元0值是否爲1，若爲1（ZF旗號爲0），則表示資料已經進入接收緩衝區了，可以進入步驟⑦。讀此資料，若不爲1（ZF旗號爲1），則表示資料尚未進入接收緩衝區內，進入下個步驟。

⑥ 測試BIOS_BREAK的位元7值是否爲1，若爲1（ZF旗號爲0）則表示使用者在鍵盤上按下Control-BREAK鍵，停止接受資料，此時要跳入步驟④，返回呼叫INT 14H的程式去。若BIOS_BREAK的位元7值不爲1（ZF旗號爲1），則返回步驟⑤繼續等資料進入接收緩衝區。若資料一直沒有進來，則程式一直停留在這裡，除非資料進來或按下 Control - BREAK鍵，才有可能離開這個永無休止的迴圈。

⑦ 進入此步驟的AL暫存器，其內的值爲線狀態暫存器內的值。這裡的AND指令是爲了保留其中的錯誤狀態（超位錯誤、同位錯誤、架構錯誤、中止中斷），並且保存在AH暫存器內。然後將I／O埠值指向接收緩衝區，並且讀其內的資料到AL暫存器內，再經由A3返回到呼叫INT 14H的程式去。

我們最後來看第4部份，讀取線狀態暫存器和調變解調器狀態暫存器。



① 將I／O埠值指向線狀態暫存器，並讀取其中的值，然後將其保存在AH暫存器內。

② 將I／O埠值指向調變解調器狀態暫存器，並讀取其內的值，然後經由A3返回到呼叫INT 14H的程式去。返回時，AL暫存器內放著調變解調器狀態暫存器內的值，AH 暫存器放著線狀態暫存器內的值。

好了，我們已經討論完了INT 14H。相信讀者也對RS-232C的傳輸方式有所了解了。

第四章

磁碟機

UPD 765介紹

INT 13H提供了控制軟式磁碟機（Floppy Diskette Drive）的界面程式，所有磁碟機的動作都可以由INT 13H 來控制。磁碟機界面卡上的UPD 765 控制著整個動作，只要了解UPD 765，就可對磁碟機各種動作有所了解。在討論 INT 13H 前，先來看UPD765。

UPD765 可以執行 15 種動作，分別是：

Read Data	Write data
Read ID	Format a Track
Read Deleted data	Write deleted Data
Read a track	Seek
Scan equal	Recalibrate (Restore to Track 0)
Scan high or equal	Sense interrupt status
Scan low or equal	Sense Drive status
Specify	

在介紹完UPD765時，筆者會列出一表格，以便顯示如何下指令給UPD765來做 15 種其中任一種的動作。各個動作的詳細情形在INT 13H 中將會被討論。

UPD765是靠著6個暫存器和CPU 溝通，主狀態暫存器（Main Status Register，MSR）指示出UPD765的狀態，其它的

4個狀態暫存器顯示出執行每個指令後的結果。資料暫存器（Data Register，DR）保存著磁碟機的狀態。

每一個動作都有三種階段（phase），分別是命令階段（Command phase）、執行階段（Execution phase）和結果階段（Result phase）。

命令階段——當CPU要執行一種動作前，要給UPD765一些資料。

執行階段——UPD765執行一個動作。

結果階段——執行動作後，CPU可以讀出UPD765的狀態。

我們現在來看和CPU溝通的6個暫存器。值得注意的是ST0，ST1，ST2，ST3這4個狀態暫存器只有在結果階段中才能被讀出。

A0	$\overline{RD}$	$\overline{WR}$	Function
0	0	0	Illegal
0	0	1	Read Main Satus Register
0	1	0	Illegal
1	0	0	Illegal
1	0	1	Read from Data Register
1	1	0	Write into Data Register

資料暫存器

當CPU要執行一個動作時，將資料位元組寫入資料暫存器（Data Register），或執行完動作後，讀出資料暫存器內的資料位元組，以便了解執行後的狀態。資料暫存器是一個堆疊（STACK），所以在讀取狀態或寫入資料時，一定要按照順序。

主狀態暫存器 (Main Status Register, MSR)

位元	7	6	5	4	3	2	1	0
	RQM	DIO	EXM	CB	D3B	D2B	D1B	D0B

① RQM（Request for Master）

0——資料暫存器尚未備妥（not ready）。

1——資料暫存器備妥（ready）。

② DIO（Data Input/Output）

0——資料（data）由CPU傳向UPD765。

1——資料（data）由UPD765傳向CPU。

③ EXM（Execution Mode，non-DMA mode only）

0——執行狀態結束，階段狀態開始。

1——執行中。

④ CB（controller Busy）

0——UPD765可以接受命令。

1——UPD765不可以接受命令。

⑤ D3B（Floppy disk drive 3 Busy）

0——3號磁碟機不在工作中，可以接受命令。

1——3號磁碟機在工作中，不可以接受命令。

⑥ D2B（Floppy disk drive 2 Busy）

0——2號磁碟機不在工作中，可以接受命令。

1——2號磁碟機在工作中，不可以接受命令。

⑦ D1B（Floppy disk drive 1 Busy）

0——1號磁碟機不在工作中，可以接受命令。

1——1號磁碟機在工作中，不可以接受命令。

⑧ D0B（Floppy disk drive 0 Busy）

0——1號磁碟機不在工作中，可以接受命令。

1——1號磁碟機在工作中，不可以接受命令。

狀態暫存器0(Status Register 0, ST0)

位元	7	6	5	4	3	2	1	0
	IC		SE	EC	NR	HD	US	
							US1	US0

① IC(Interrupt code)

位元7　位元6

0	0	正常結束。命令已完成，且經執行。
0	1	命令異常結束。命令剛開始執行，但並未完成。
1	0	錯誤命令。已發出命令，但不能執行。
1	1	執行中時，來自FDD的備妥信號改變狀態，而使命令不正常終止。

② SE(Seek End)

0——尋找命令(Seek)沒有完成。

1——尋找命令(Seek)完成。

③ EC(Equipment Check)

0——沒有錯誤。

1——在接到來自FDD的障礙信號，或者重新回到第零軌，經過77個步進脈衝而仍未找到磁軌0信號之後，此旗標被設定。

④ NR(Not Ready)

0——磁碟機是在備妥狀態。

1——磁碟機尚未備妥(ready)。

⑤ HD(Head Address)

0——磁頭0被選擇。

1——磁頭1被選擇。

⑥ US(Unit select)

US1	US0	
0	0	磁碟機0被選擇。
0	1	磁碟機1被選擇。
1	0	磁碟機2被選擇。
1	1	磁碟機3被選擇。

狀態暫存器1(Status Register 1, ST1)

位元	7	6	5	4	3	2	1	0
	EN	0	DE	OR	0	NI	NW	MA

註：位元6和位元3永遠爲0。

① EN(End of TRACK)

0——沒有錯誤。

1——FDC 試圖去存取磁柱最後一磁段以外的磁段。

② DE(Data Error)

0——沒有錯誤。

1——不論ID區或資料區，只要FDC查到一個CRC錯誤時，此旗標即被設定。

③ OR(Over Run)

0——沒有錯誤。

1——在資料傳送時，CPU來不及將資料取出。

④ NI(No ID)

0——沒有錯誤。

1——在執行讀取資料，寫入刪除資料或掃描指令時，如

果 FDC 不能在 ID 暫存器中找到所指定的磁段，此旗標即設定。在執行讀取 ID 命令時，如果FDC 讀入的 ID 區有錯誤，則此旗標設定。在執行讀取一磁柱命令時，如果不能找到開始的磁段，此旗標亦會被設定。

⑤ NW (Not Writeable)

0 ——沒有錯誤。

1 ——在執行寫入資料，寫入刪除資料或定一磁柱的格式命令時，如果 FDC 從 FDD 偵測到一個防止寫入信號，則此旗標會被設定。

⑥ MA (Missing Address Mark)

0 ——沒有錯誤。

1 ——如果 FDC 不能偵測到 ID 位址記號，則此旗標會被設定，同時，狀態暫存器 2 的 MD 也會被設定。

狀態暫存器 2 (Status Register2, ST2)

位 元	7	6	5	4	3	2	1	0
	0	CM	DD	WT	SH	SN	BT	MD

① CM (Control Mark)

0 ——沒有錯誤。

1 ——在讀取資料或掃描命令時，如果 FDC 碰到一個含有刪除位址記號的磁段，則此旗標被設定。

② DD (Data Error in Data Field)

0 ——沒有錯誤。

1 —— FDC 在資料上偵測到一個 CRC 錯誤時，此旗標即設定。

③ WT (Wrong Track)

0 ——沒有錯誤。

1 ——此位元和 ND 位元有關係，當磁片上 C 的內容和存在 ID 暫存器中的不同時，此旗標即設定。

④ SH (SCAN EQUAL HIT)

0 ——在掃描命令中，沒有相等的情形發生。

1 ——在掃描命令中，有相等的情況發生。

⑤ SN (Scan not Satified)

0 ——沒有錯誤。

1 ——執行掃描命令時，如果 FDC 不能在磁柱上找到一磁段，其內的資料符合所要求的情況，則此旗標被設定。

⑥ BT (Bad Track)

0 ——沒有錯誤。

1 ——此位元和 ND 位元有關係，當磁片上 C 的內容和存在 ID 暫存器內的不同，而且 C 的內容又是 FF 時，此旗標被設定。

⑦ MD (Missing Address Mark in Data Field)

0 ——沒有錯誤。

1 ——從磁片讀取資料時，如果 FDC 不能找到一個資料位址記號或刪除的資料位址記號，則此旗標會被設定。

狀態暫存器 3 (Status Register3, ST3)

位 元	7	6	5	4	3	2	1	0
	FT	WP	RY	T0	TS	HD	US1	US0

① FT（Fault）
此位元是來自 FDD 的障礙信號的狀態。

② WP（Write Protect）
此位元來自 FDD 的防止寫入信號的狀態。

③ RY（Ready）
此位元來自 FDD 的備妥信號的狀態。

④ T0（Track 0）
此位元是來自 FDD 的 0 磁軌信號的狀態。

⑤ TS（Two side）
此位元是來自 FDD 的雙面信號的狀態。

⑥ HD（Head Address）
此位元來自 FDD 的面選擇信號的狀態。

⑦ US1（Unit Select 1）
此位元來自 FDD 的單元選擇1的信號狀態。

⑧ US0（Unit Select 0）
此位元來自 FDD 的單元選擇 0 的信號的狀態。

在介紹完UPD765 的暫存器後，以下介紹UPD765 和FDD（磁碟機）溝通的幾支腳（FDD Status interface）：

① RDY（Ready）：當此輸出腳爲高電位時，表示磁碟機已經準備好了，UPD765可以使用磁碟機。

② IDX（Index）：當此輸入腳爲高電位時，表示磁片上的指示孔（index hole）已被測試到。（指示孔是爲了同步UPD765的時序）。

③ $\overline{RW}$/SEEK（Read Write/SEEK）：當此腳的輸出爲低電位時，表示要進行讀取寫入的動作。當此腳的輸出爲高電位時，表示要進行SEEK動作。

$\overline{RW}$/SEEK	Mode	Active FDD interface Signals
Low	Read/Write	WP, FLT, LCT, FR
High	SEEK	TS, TRK0, DIR, STP

④ WP/TS（Write Protect/ Two side）：在讀取/寫入（Read/write）模式中，當此腳的輸入爲高電位時，表示磁片上的防止寫入（Write protect）缺角已被貼住。在Seek 模式中，當此腳的輸入爲高電位時，表示磁片是雙面的（double-sided）。

⑤ FLT/TRK0（Fault/TRACK ZERO）：在讀取/寫入模式中，當此腳的輸入爲高電位時，表示磁碟機失敗(FDD Fault)。在seek 模式中，當此腳的輸入爲高電位時，表示磁頭已到了磁軌 0 的位址。

⑥ LCT/DIR（Low Current/Direction）：在讀取/寫入模式中，當此腳的輸出爲低電位時，表示磁頭是在內圈磁軌（inner Tracks），當此腳的輸出爲高電位時，表示磁頭是在外圈磁軌（outer tracks）。在seek模式中，LCT/DIR這支輸出腳控制著磁頭的移動方向，當此腳的輸出爲高電位時，磁頭向外移動，當此腳的輸出爲低電位時，磁頭向內移動。

⑦ FR/STP（Fault Reset/ Step）：在讀取/寫入模式中，當此腳的輸出爲高電位時，會重置（Reset）磁碟機的錯誤指示在seek 模式中，此腳提供了移動磁頭的脈衝（pulse）。

⑧ HDL（Head Load）：當此腳的輸出爲高電位時，表示將磁頭和磁片接觸；當此腳的輸出爲低電位時，表示將磁頭抬起，不和磁片接觸。

⑨ HD（Head Select）：當此腳的輸出爲高電位時，表示選擇磁頭 1（位於第 1 面），當此腳的輸出爲低電位時，表

示選擇磁頭0（位於第0面）。

⑩ US0,US1(Unit select)：這二個輸出脚用來選擇磁碟機。

US0	US1	Drive select
0	0	0
0	1	1
1	0	2
1	1	3

⑪ MFM(MFM mode)：當此脚的輸出爲高電位時，表示爲MFM模式，當此脚的輸出爲低電位時，表示爲FM模式。

我們說過UPD765可以處理15種動作，以下就是對這15種動作逐一做介紹。

Ⓐ讀取資料：

在執行讀取資料動作前，CPU必須依順序寫入9個位元組給UPD765(命令狀態)，在執行完讀取資料動作後，CPU可以由UPD765讀出7個位元組，以便了解執行的結果(結果狀態)。

當UPD765接到讀取資料的9個位元組後，首先將磁頭接觸到磁片上，並且等待磁頭定位時間(head settling time)(由Specify動作決定)，然後開始讀ID位址標記(ID Address Mark)和ID欄(ID fields)。如果從ID欄讀出來的磁區號碼(R)和我們所指定的磁區號碼(在9個位元組內)相同的話，就開始讀資料區的資料。

命令階段：

R/W	位元組	7	6	5	4	3	2	1	0
W	1	MT	MF	SK	0	0	1	1	0
	2	×	×	×	×	×	HD	US1	US0
	3	TRACK Number (T)							
	4	Head Number (H)							
	5	Sector Number (R)							
	6	Number of Data Bytes (N)							
	7	End of Track (EOT)							
	8	Gap Length (GPL)							
	9	Data Length (DTL)							

其中：① MT(Multi-Track)：Multi-Track使得UPD765可以讀磁片的二面資料。

② MF(MFM/FM)：此位元用來顯示是何種模式。

③ SK(Skip Deleted Data Address Mark)：當SK=0，而且UPD765讀到Deleted Data Address Mark，則UPD765停止Read Data動作。如果SK=1，而且UPD765讀到Deleted Data Address Mark，則UPD765會跳過這個磁區而去讀下個磁區的資料。

結果階段：

R/W	BYTE	
R	1	Status Register 0 (ST0)
R	2	Status Register 1 (ST1)
R	3	Status Register 2 (ST2)
R	4	Track Number (T)
R	5	Head Number (H)
R	6	Sector Number (R)
R	7	Number of Data Bytes (N)

在執行完讀取資料動作後，UPD765會將磁頭抬起，離開磁片。若有錯誤產生，則UPD765會停止讀取資料動作，並將錯誤顯示在狀態暫存器內。

Ⓑ寫入資料：

在執行寫入資料動作前，CPU 必須依順序寫入 9 個位元組給UPD765（命令階段），在執行完寫入資料動作後，CPU 可以由UPD765讀出 7 個位元組，以便了解執行的結果（結果階段）。

當UPD765接到寫入資料的 9 個位元組後，首先將磁頭接觸到磁片上。當我們讀出的 T，H，R，N 4 個位元組和我們指定的相同時，UPD 765就會將資料寫上去。

命令階段：

R/W	BYTE	7	6	5	4	3	2	1	0
W	1	MT	MF	0	0	0	1	0	1
	2	×	×	×	×	×	HD	US1	US0
	3	TRACK Number (T)							
	4	Head Number (H)							
	5	Sector Number (R)							
	6	Number of Data Bytes (N)							
	7	End of Track (EOT)							
	8	Gap Length (GPL)							
	9	Data Length (DTL)							

結果階段：

R/W	BYTE	7	6	5	4	3	2	1	0
	1	Status Register 0 (ST0)							
	2	Status Register 1 (ST1)							
	3	Status Register 2 (ST2)							
	4	Track Number (T)							
	5	Head Number (H)							
	6	Sector Number (R)							
	7	Number of Data Bytes (N)							

在執行完寫入資料動作後，UPD765會將磁頭抬起，離開磁片。若在執行時有錯誤產生，則UPD765 會停止寫入資料動作，並將錯誤顯示在狀態暫存器內。

Ⓒ寫入刪除資料：

這個動作和寫入資料動作相同，只不過在資料區中寫下Deleted Data Address Mark。而寫入資料動作中是寫入Normal Data Address Mark，其它的動作都相同。

命令階段：

R/W	位元組	7	6	5	4	3	2	1	0
W	1	MT	MF	0	0	1	0	0	1
	2	×	×	×	×	×	HD	US1	US0
	3	Track Number (T)							
	4	Head Number (H)							
	5	Sector Number (R)							
	6	Number of Data Bytes (N)							
	7	End of Track (EOT)							
	8	Gap Length (GPL)							
	9	Data Length (DTL)							

結果階段：

R/W	BYTE	7	6	5	4	3	2	1	0
R	1	Status Register 0 (ST0)							
	2	Status Register 1 (ST1)							
	3	Status Register 2 (ST2)							
	4	Track Number (T)							
	5	Head Number (H)							
	6	Sector Number (R)							
	7	Number of Data Bytes (N)							

Ⓓ讀取刪除資料：

這個動作和讀取資料動作相同，只有下面二點不同。

① 如果SK＝0，當UPD765讀到資料位址標記，它會將這個磁區內的資料讀出。

② 如果SK＝1，當UPD765讀到資料位址標記，它不讀這個磁區的資料，而去讀下個磁區的資料。

命令階段：

R/W	BYTE	7	6	5	4	3	2	1	0
W	1	MT	MF	SK	0	0	1	1	0
	2	×	×	×	×	×	HD	US1	US0
	3	Track Number (T)							
	4	Head Number (H)							
	5	Sector Number (R)							
	6	Number of Data Bytes (N)							
	7	End of Track (EOT)							
	8	Gap Length (GPL)							
	9	Data Length (DTL)							

結果階段：

R/W	BYTE	7	6	5	4	3	2	1	0
R	1	Status Register 0 (ST0)							
	2	Status Register 1 (ST1)							
	3	Status Register 2 (ST2)							
	4	Track Number (T)							
	5	Head Number (H)							
	6	Sector Number (R)							
	7	Number of Data Bytes (N)							

Ⓔ讀取磁軌：

這個動作和讀取資料的動作相似，只不過這個動作要讀整個磁軌的資料，並非只有一個磁區。當UPD765測試到指示孔，它就開始讀這個磁軌的全部資料。讀磁軌不允許Multi-track和skip動作。

當UPD765從磁片上讀到的磁區數目和EOT相同，則表示已讀完了整個資料。如果UPD765連續測試到二次指示孔而沒有讀到ID位址標記，則UPD765會停止動作，並將ST1的MA設定爲1，ST0 的位元7和位元6設定0和1。

命令階段：

R/W	BYTE	7	6	5	4	3	2	1	0
W	1	0	MF	SK	0	0	0	1	0
	2	×	×	×	×	×	HD	US1	US0
	3	Track Number (T)							
	4	Head Number (H)							
	5	Sector Number (R)							
	6	Number of Data Bytes (N)							
	7	End of Track (EOT)							
	8	Gap Length (GPL)							
	9	Data Length (DTL)							

結果階段：

R/W	BYTE	7	6	5	4	3	2	1	0
R	1	Status Register 0 (ST0)							
	2	Status Register 1 (ST1)							
	3	Status Register 2 (ST2)							
	4	Track Number (T)							
	5	Head Number (H)							
	6	Sector Number (R)							
	7	Number of Data Bytes (N)							

Ⓕ讀取ID

這個動作是要知道磁頭的位置。UPD765 控制磁頭，使它能讀到最近的 ID。如果在第二次測試到指示孔之前，無法找到正確的 ID 位址標記，則ST1 的MA會被設定爲 1。若沒有讀到 ID，則 ST1 的NI 會被設定爲 1，ST0 的位元7 和位元 6會被設定成 0和 1，然後停止動作。在READ ID動作時，UPD765不會將資料傳給CPU。動作結束後，CPU可以讀出結果。

命令階段：

R/W	BYTE	7	6	5	4	3	2	1	0
W	1	0	MF	0	0	1	0	1	0
	2	×	×	×	×	×	HD	US1	US0

結果階段：

R/W	BYTE	7	6	5	4	3	2	1	0
R	1	Status Register 0 (ST0)							
	2	Status Register 1 (ST1)							
	3	Status Register 2 (ST2)							
	4	Track Number (T)							
	5	Head Number (H)							
	6	Sector Number (R)							
	7	Number of Data Bytes (N)							

Ⓖ磁軌格式化(Format a Track)

這個磁軌格式化的動作，也就是說將間隙（Gaps），位址標記，ID 欄和資料區寫到一個磁軌上。UPD765 會要求 CPU 給每一個磁區 4 個值，即磁軌號碼 (T)，磁頭號碼 (H)，磁區號碼 (R) 和每個磁區的位元組數 (N)，如此即可辨認出各個磁區。

命令階段：

R/W	BYTE	7	6	5	4	3	2	1	0
W	1	0	MF	0	0	1	1	0	1
	2	×	×	×	×	×	HD	US1	US0
	3	Number of Bytes (N)							
	4	Sector per Track (ST)							
	5	Gap Length (GPL)							
	6	Data Pattern (D)							

其中Data pattern (D) 是暫時寫入資料區的資料。

結果階段：

R/W	BYTE	7	6	5	4	3	2	1	0
R	1	Status Register 0 (ST0)							
	2	Status Register 1 (ST1)							
	3	Status Register 2 (ST2)							
	4	Track Number (T)							
	5	Head Number (H)							
	6	Sector Number (R)							
	7	Number of Data Bytes (N)							

Ⓗ Scan Command

這個動作是將從磁片上讀出的資料和 CPU 所提供的資料互相比較。有三種 SCAN 動作，分別是SCAN EQUAL , SCAN LOW

OR EQUAL, SCAN HIGH OR EQUAL。

① SCAN EQUAL

命令階段：

R/W	BYTE	7	6	5	4	3	2	1	0
W	1	MT	MF	SK	1	0	0	0	1
	2	×	×	×	×	×	HD	US1	US0
	3	Track Number (T)							
	4	Head Number (H)							
	5	Sector Number (R)							
	6	Number of Data Bytes (N)							
	7	End of Track (EOT)							
	8	Gap Length (GPL)							
	9	Sector Test Process (STP)							

其中Sector Test Process (STP)：當STP = 1 時，連續的磁區(continuous sector)內的資料將會和CPU送來的資料比較。當STP = 2 時，間隔的磁區(alternate sector) 內的資料會和CPU送來的資料比較。

結果階段：

R/W	BYTE	7	6	5	4	3	2	1	0
R	1	Status Register 0 (ST0)							
	2	Status Register 1 (ST1)							
	3	Status Register 2 (ST2)							
	4	Track Number (T)							
	5	Head Number (H)							
	6	Sector Number (R)							
	7	Number of Data Bytes (N)							

② SCAN LOW OR EQUAL

命令階段：

R/W	BYTE	7	6	5	4	3	2	1	0
W	1	MT	MF	SK	1	1	0	0	1
	2	×	×	×	×	×	HD	US1	US0
	3	Track Number (T)							
	4	Head Number (H)							
	5	Sector Number (R)							
	6	Number of Data Bytes (N)							
	7	End of Track (EOT)							
	8	Gap Length (GPL)							
	9	Sector Test Process (STP)							

結果階段：

R/W	BYTE	7	6	5	4	3	2	1	0
R	1	Status Register 0 (ST0)							
	2	Status Register 1 (ST1)							
	3	Status Register 2 (ST2)							
	4	Track Number (T)							
	5	Head Number (H)							
	6	Sector Number (R)							
	7	Number of Data Bytes (N)							

③ SCAN HIGH OR EQUAL

命令階段：

R/W	BYTE	7	6	5	4	3	2	1	0
W	1	MT	MF	SK	1	1	1	0	1
	2	×	×	×	×	×	HD	US1	US0
	3	Track Number (T)							
	4	Head Number (H)							
	5	Sector Number (R)							
	6	Number of Data Bytes (N)							
	7	End of Track (EOT)							
	8	Gap Length (GPL)							
	9	Sector Test Process (STP)							

結果階段：

R/W	BYTE	7	6	5	4	3	2	1	0
R	1	Status Register 0 (ST0)							
	2	Status Register 1 (ST1)							
	3	Status Register 2 (ST2)							
	4	Track Number (T)							
	5	Head Number (H)							
	6	Sector Number (R)							
	7	Number of Data Bytes (N)							

⑴ SEEK

這個動作將磁頭移至指定的磁軌(New Track Number，NTN)。UPD765 給每個磁碟機二個獨立的 Present Track Number (PTN)。在執行 SEEK 動作前 UPD765 會將 PTN 和 NTN比較，如果：

① PTN＜NTN ：使 LCT/DIR 輸出高電位，並發出步進脈衝，使磁頭向內移動。

② PTN＞NTN ：使 LCT/DIR 輸出低電位，並發出步進脈衝，使磁頭向外移動。

命令階段：

R/W	BYTE	7	6	5	4	3	2	1	0
W	1	0	0	0	0	1	1	1	1
	2	×	×	×	×	×	0	US1	US0
	3	New Track Number (NTN)							

Seek 動作沒有結果階段。

⑵ 歸零(Recalibrate)

這個動作將磁頭拉至磁軌 0 的位置。如果UPD765的 TR0 這個輸入脚爲低電位，則表示磁頭並不在磁軌 0 的位置，此時UPD765使LCT/DIR的輸出爲低電位，並發出步進脈衝，直到 TR0 輸入爲高電位爲止。(步進脈衝不得超過 256 個，若發出 256 個步進脈衝後，仍得不到 TR0 的輸入爲高電位，則設定 ST0 的SE和EC爲1，ST1的位元7位元6爲0和1。)

命令階段：

R/W	BYTE	7	6	5	4	3	2	1	0
W	1	0	0	0	0	0	1	1	1
	2	×	×	×	×	×	0	US1	US0

歸零沒有結果階段。

⑶ 偵測中斷狀態 (Sense interrupt status)

UPD765 在下列4 個情況時，會發出 INT 要求訊號。

1. 執行下列動作，進入結果階段時，
   a. Read Data Command
   b. Read a Track Command
   c. Read ID Command
   d. Read Deleted Data Command
   e. Write data Command
   f. Format a Track Command
   g. Write deleted Data Command
   h. SCAN Command
2. 當RDY(READY) 輸入脚變換電位時，
3. Seek 和歸零命令結束時，
4. 在Non-DMA 模式中執行狀態時，

由情況 1 和4 所產生的 INT 要求訊號很容易被 CPU 辨認，但由情況 2 和3 所產生的 INT 要求訊號不容易被 CPU 辨認，所以需要偵測中斷狀態動作。

命令階段：

R/W	BYTE	7	6	5	4	3	2	1	0
W	1	0	0	0	0	1	0	0	0

結果階段：

R/W	BYTE	7	6	5	4	3	2	1	0
R	1	Status Register 0 (ST0)							
	2	Present Track Number (PTN)							

當CPU讀出ST0 就可以辨認出是那一個提出 INT 要求。請讀者自行複習ST0 。

①SPECIFY

這個動作是爲了設定三個時間 (Step rate time , head load time , head unload time) 和DMA 模式。

① Step rate time (SRT): 這個時間是UPD765發出步進脈衝的間距時間。

② head load time (HLT) : 這個時間是UPD765 的HDL輸出爲高電位到 R / W 動作開始的間距時間。

③ head unload time (HUT): 這個時間是 R / W 動作結束時到磁頭抬起的等待時間。

命令階段：

R /W	BYTE	7	6	5	4	3	2	1	0
W	1	0	0	0	0	0	0	1	1
	2	SRT				HUT			
	3	HLT							ND

其中ND： ①ND = 1 , 表示Non-DMA 模式。

②ND = 0 , 表示DMA 模式。

SPECIFY 沒有結果階段。

ⓜSense Drive Status

這個動作是爲了得到磁碟機的狀態。

命令階段：

R /W	BYTE	7	6	5	4	3	2	1	0
W	1	0	0	0	0	0	1	0	0
	2	×	×	×	×	×	HD	US1	US0

結果階段：

R/W	BYTE	7	6	5	4	3	2	1	0
R	1	Status Register 3 (ST3)							

好了，我們已經大略地了解UPD765。現在，我們來看 INT 13H 磁碟機界面程式是如何寫的。

INT 13H 進入點

INT 13H提供磁碟機的界面程式，當討論完此程式後，讀者會明白磁碟機的動作情形，並且能對UPD765有進一步的了解。若讀者想要更詳細的了解磁碟機系統和UPD765，筆者建議讀者參考 Intel 公司的microsystem components handbook 第二册的8272部份及其應用。（註：Intel 公司的 8272 即NEC 公司的UPD765）

INT 13H 大約可以分爲 6 個部分：

(AH) = 0　　Reset Diskette system

(AH) = 1　　Read the status of system into (AL)

(AH) = 2　　Read the desired sectors into memory

(AH) = 3　　Write the desired sectors from memory

(AH) = 4　　Verify the desired sectors

(AH) = 5　　Format the desired track

當AH暫存器內的值爲 2 到 5 時，下列暫存器各有其功用：

(DL)：Drive number (0–3)

(DH)：Head number (0–1)

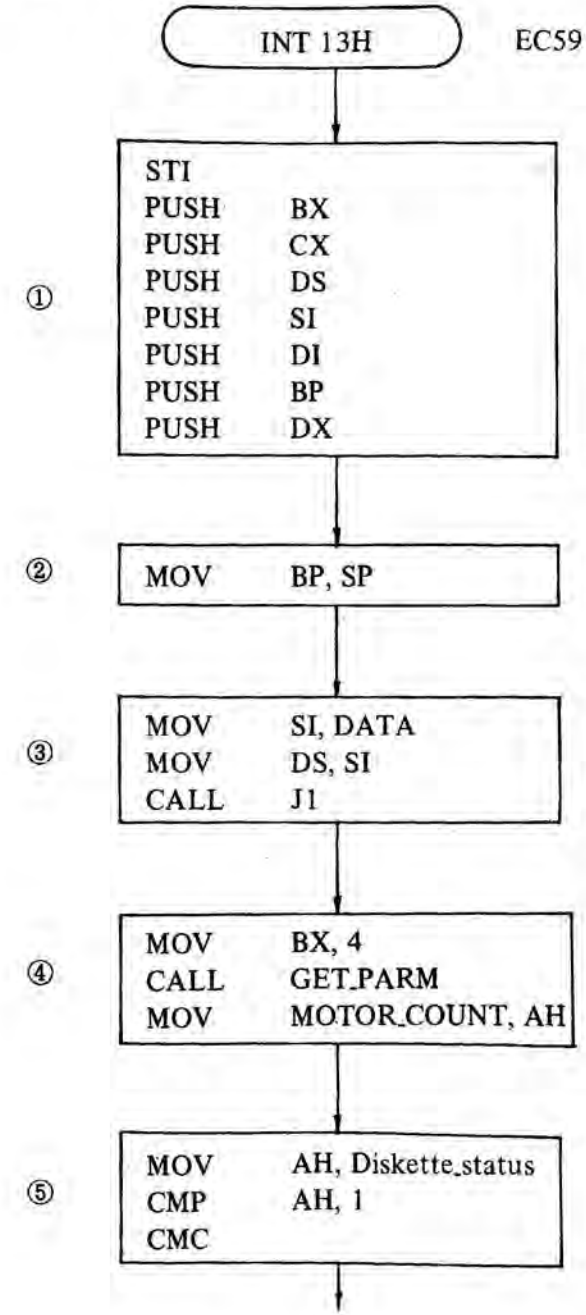
(CH)：Track number (0–39)

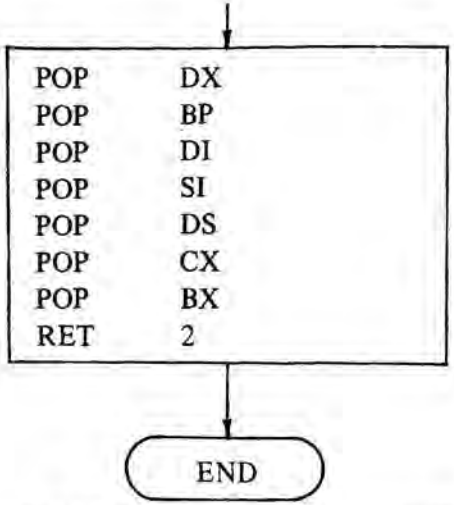
(CL)：Sector number (1–8)

(AL)：Number of sectors (MAX=8)

(ES:BX)　Address of Buffer

我們現在就進入 INT 13H。





① 首先設定 IF 旗號爲 1 ，表示在 INT 13H時，仍可以接受中斷要求。然後依順序將BX , CX , DS , SI , DI , BP , DX 等暫存器保存在堆疊中。

② 將 SP 暫存器內的值保存在 BP 暫存器內。在上一個步驟中，最後存入堆疊的是DX 暫存器（ 在Read/Write/Verify / format 動作時，DL 暫存器內的值指出磁碟機，DH 暫存器內的值指出磁頭號碼），所以此時 BP 暫存器內的值所指的位址內的值爲磁碟機和磁頭號碼。

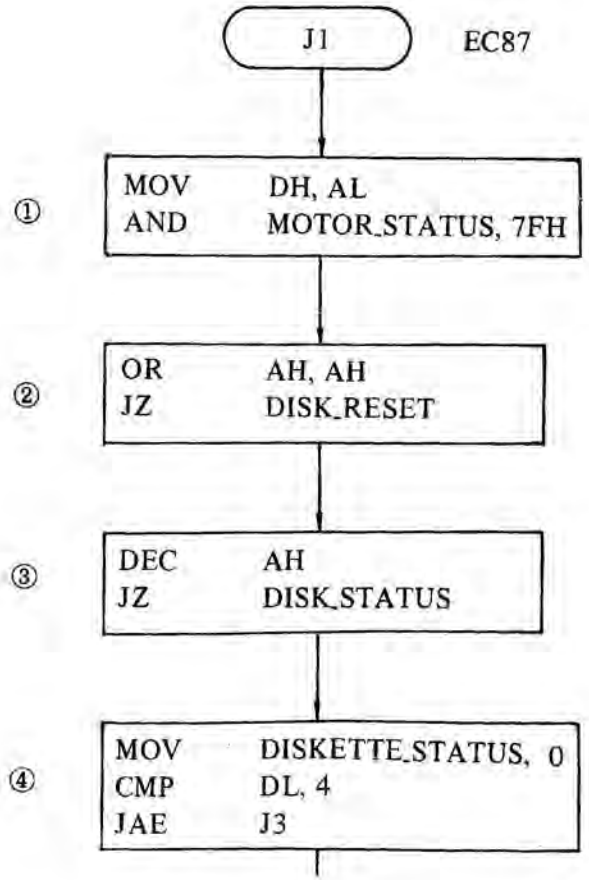
③ 將DS 暫存器設定爲DATA（0040），然後呼叫 J1 副程式，以便根據 AH 暫存器內的值去執行動作。

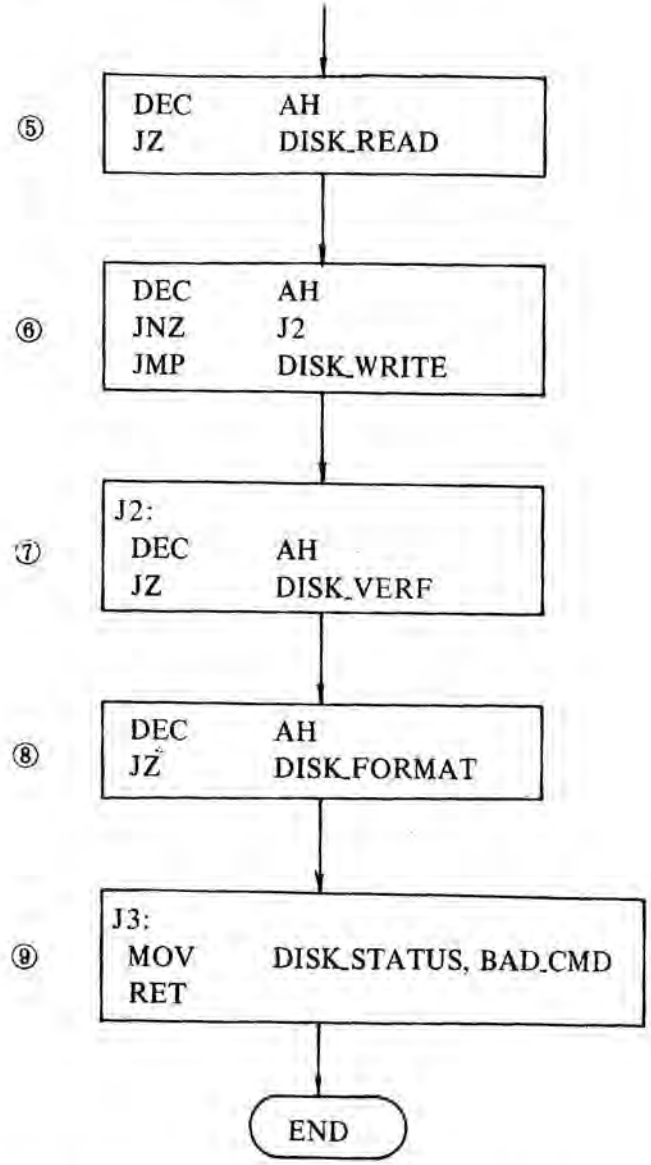
④ 在 BX 暫存器內放入 4 ，呼叫GET_PARM副程式，以便取出 MOTOR_WAIT TIME到AH 暫存器內，然後將其保存到 MOTOR_COUNT 內。

⑤ 執行完動作，Diskette_status內放著錯誤碼，除非diskette_status內的值爲 00，否則就表示有錯誤產生。先將 diskette_status內的值保存在 AH 暫存器內，然後和 1 比較，若 diskette_status 內的值大於等於 1 ，就表示有錯誤，CF旗號爲 0 ，若 diskette_status 內的值爲 0 ，就表示正確，CF 旗號爲 1 。經過 CMC 指令後，有錯誤時，則 CF 旗號爲 1 ，若爲正確，CF旗號爲 0 。

⑥ CPU 執行 INT 13H 時，會先將旗號暫存器保存在堆疊中，但 INT 13H 是以 CF 旗號來表示是否有錯誤產生，所以在陸續取回暫存器的初值後，使用RET2指令跳過原先儲存的旗號暫存器，而回到呼叫 INT 13H的程式去。

從上面的討論，我們可以看出，J1 副程式才是 INT 13H的眞正進入點。我們繼續討論 J1 副程式。





① 將AL暫存器內的值（磁區數目）保存在DH暫存器內。讀者不必擔心會破壞掉磁頭號碼因爲在上一段程式中，已設定好BP暫存器指向磁頭號碼和磁碟機號碼，請讀者記住此步驟，以下的討論將會常常用到。然後將MOTOR_STATUS的位元7設定爲0，以便表示爲讀取動作。

② 檢查AH暫存器內的值是否爲0，若爲0（ZF旗號爲1）則表示要重置磁碟，跳入DISK_RESET副程式，若AH暫存器內的值不爲0，進入下個步驟。

③ 檢查AH暫存器內的值是否爲1，若爲1（ZF旗號爲1），則表示要讀diskette_status，跳入DISK_STATUS副程式，若AH暫存器內的值不爲1，進入下個步驟。

④ 將Diskette_status設定爲0，然後將DL暫存器內的值和4比較，若DL暫存器內大於等於4，即表示使用者欲使用第5個磁碟機，跳入步驟⑨，設定錯誤的指令。若DL暫存器內的值小於4，進入下個步驟。

⑤ 檢查AH暫存器內的值是否爲2，若爲2（ZF旗號爲1），則表示要讀取磁區內的資料，跳入DISK_READ程式，若AH暫存器內的值不爲2，進入下個步驟。

⑥ 檢查AH暫存器內的值是否爲3，若爲3（ZF旗號爲1），則表示要將資料寫入磁區內，跳入DISK_WRITE副程式，若AH暫存器內的值不爲3，進入下個步驟。

⑦ 檢查AH暫存器內的值是否爲4，若爲4（ZF旗號爲1）則表示要VERIFY，跳入DISK_VERF程式，若AH暫存器內的值不爲4，進入下個步驟。

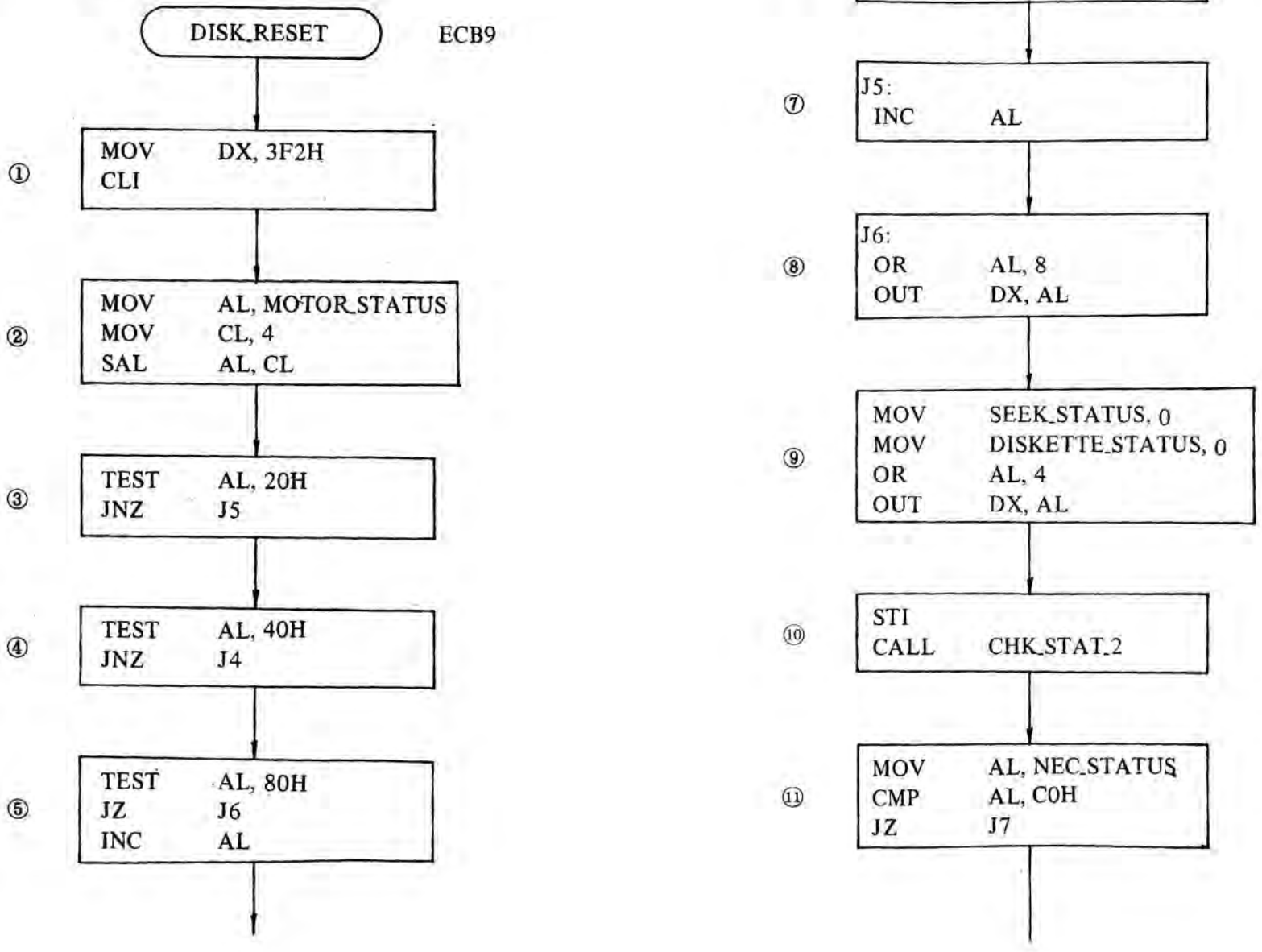
⑧ 檢查AH暫存器內的值是否爲5，若爲5（ZF旗號爲1），則表示要格式化一磁軌（FORMAT a track），跳入DISK_FORMAT副程式，若AH暫存器內的值不爲5，則表示使用者下錯命令（Command），進入步驟⑨。

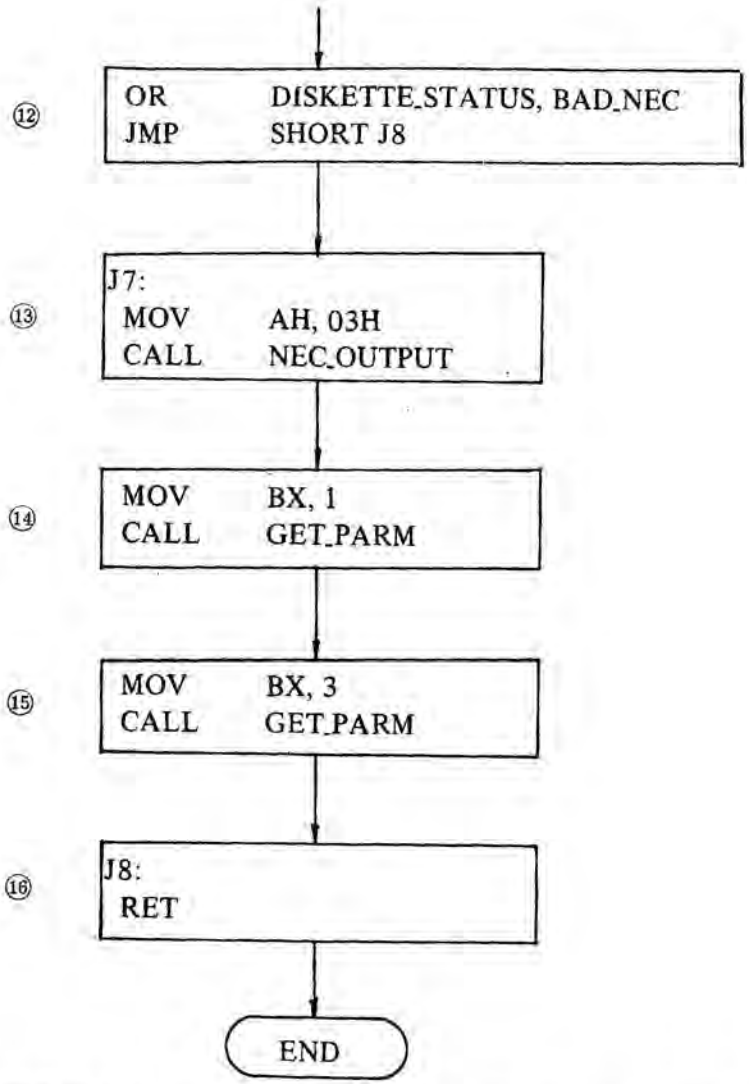
⑨ 在DISK_STATUS內放入BAD_CMD（01），然後返回。

討論到這裏，讀者會發現尚未眞正討論到INT 13H的動作，以下

我們就以AH暫存器內的值做各別討論。

(a) (AH)=0, DISK_RESET





① 在DX暫存器內放入3F2H，以便指向磁碟機控制卡的數位輸出暫存器（Digital Output Register），然後設定IF旗號爲0。

② MOTOR_STATUS的位元3到位元0，是用來表示是那一個磁碟機正在使用。位元7用來表示READ/WRITE動作。這裏將MOTOR_STATUS轉到AL暫存器內是爲了找出是那一個磁碟機正在使用，然後將AL暫存器左移4個位元，這是爲了配合磁碟機控制卡的數位輸出暫存器。

③ 檢查是否爲磁碟機B，若是磁碟機B（ZF旗號爲0），則進入步驟⑦，若不爲磁碟機B，進入下個步驟。

④ 檢查是否爲磁碟機C，若是磁碟機C（ZF旗號爲0），則進入步驟⑥，若不爲磁碟機C，進入下個步驟。

⑤ 檢查是否爲磁碟機D，若是磁碟機D（ZF旗號爲0），則將AL暫存器加1，進入下個步驟，若不是磁碟機D，則必是磁碟機A（ZF旗號爲1），進入步驟⑧。

⑥ 進入此步驟是因爲磁碟機C的馬達在轉動。我們知道磁碟機控制板上的數位輸出暫存器的位元1和位元0是用來選擇磁碟機的。

bit 1	bit 0	drive
0	0	A
0	1	B
1	0	C
1	1	D

因爲磁碟機C的馬達正在轉動，所以在輸出值到數位輸出暫存器前，必須將AL暫存器的位元1和位元0設定爲1和0，所以在步驟⑧前先將AL暫存器內的值加2，以便選擇到磁碟機C。

⑦ 進入此步驟是因爲磁碟機B的馬達正在轉動，將AL暫存器內的值加1，理由和步驟⑥一樣。

⑧ 將AL暫存器的位元3設定爲1，然後輸出此值到數位輸出暫存器。（注意位元2爲0，表示會給UPD765一個重置（RESET）信號）。

由以上的討論，我們列一個表，此表爲步驟⑧輸出到數位輸出暫存器的值。

drive A	00011000
drive B	00101001
drive C	01001010
drive D	10001011

註：位元2爲0，表示給UPD765一個重置信號。
位元3爲1，表示起動INT & DMA要求。

筆者希望讀者RETRACE步驟①到⑧。

⑨ 將SEEK_STATUS,DISKETTE_STATUS 都設定爲0，然後將 AL 暫存器的位元2設定爲1，並寫到數位輸出暫存器。（其它位元沒有改變）。如此才完整地給UPD765一個重置信號。

⑩ 設定 IF 旗號爲1，然後呼叫CHK_STAT_2 副程式，以便瞭解重置後的狀態。CHK_STAT_2 副程式待會討論。

⑪ NEC_STATUS 是7個位元組的記憶體位址，是爲了儲存UPD765的結果狀態。這裏將NEC_STATUS（ST0）放到AL 暫存器內，並測試其位元7和位元6是否都爲1，若其都爲1，則表示UPD765是好的，進入步驟⑬，若 ST0 的位元7和位元6不同時爲1，即認爲UPD765是不良的，進入下個步驟。

⑫ 此步驟將DISKETTE_STATUS的位元5設定爲1，然後經由步驟⑯返回。BAD_NEC的值爲20H。

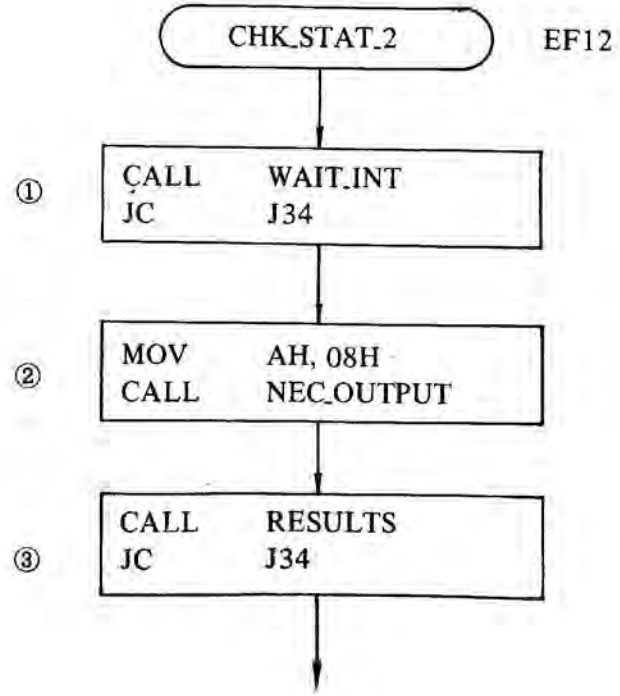
⑬ 經由NEC_OUTPUT 副程式將 03H 寫到UPD765的資料暫存器（03爲SPECIFY的第一個命令位元組（Command Byte）。

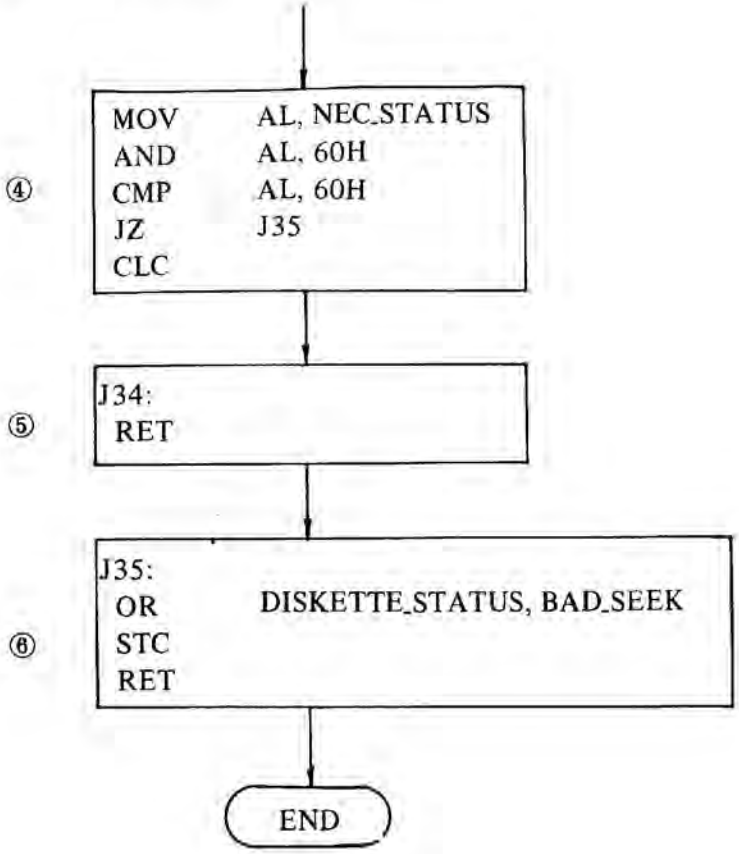
⑭ 經由GET_PARM副程式取出SPECIFY 的第2個命令位元組，並寫到UPD765的資料暫存器。因爲 BX 暫存器內爲奇數，所以取出的值會經過NEC_OUTPUT 副程式。
NEC_OUTPUT和GET_PARM副程式待會討論。

⑮ 經由GET_PARM副程式取出SPECIFY 的第3個命令位元組，並寫到UPD765的資料暫存器，如此即完成SPECIFY動作。

⑯ 返回到INT 13H的步驟④去。

我們繼續討論CHK_STAT_2副程式。





① 呼叫WAIT_INT 副程式，等待UPD765發出 INT 信號，UPD765執行動作後，都會發出 INT 信號。若返回時，CF 旗號爲 1，即表示UPD765沒有發出 INT 信號，則跳至步驟⑤返回。

② 將08H經由NEC_OUTPUT 副程式寫到UPD765的資料暫存器，表示要偵測中斷狀態。（08H是偵測中斷狀態的命令位元組）。

③ 呼叫RESULTS副程式讀結果階段，如果返回時，CF 旗號爲 1，即表示有錯誤產生，跳至步驟⑤返回。

④ 將RESULTS副程式所讀到的第一個結果狀態位元組（ST0）放到AL 暫存器內，並測試其位元 6、位元 5 是否爲 1，若位元 6 和位元 5 均爲 1（ZF 旗號爲 1），則進入步驟⑥，設定錯誤碼和旗號。若位元 6 和位元 5 不均爲 1，則清除CF 旗號，然後進入下個步驟。

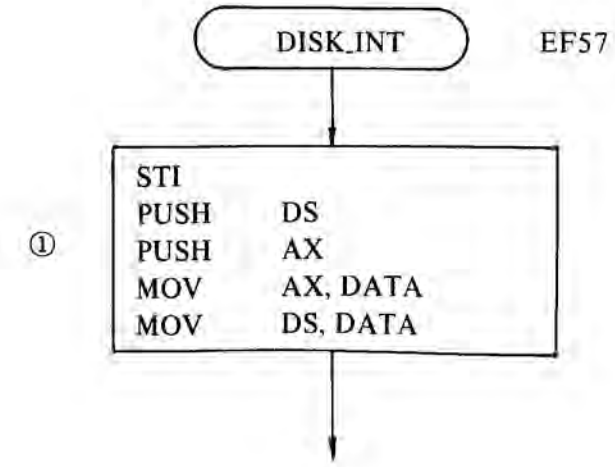
⑤ 返回到呼叫此程式的程式去。

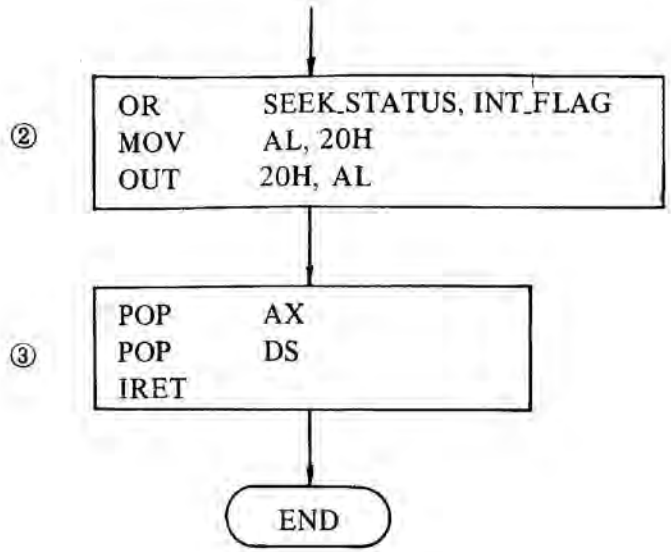
⑥ 將DISKETTE_STATUS的位元 6 設定爲 1（BAD_SEEK 爲 40H），並設定 CF 旗號爲 1，然後返回到呼叫此程式的程式去。

在短短的CHK_STAT_2 程式中用到三個副程式，分別是WAIT_INT，NEC_OUTPUT，RESULTS。接下來，我們依序討論這三個副程式。

WAIT_INT

在討論WAIT_INT 副程式前，我們應該先看處理UPD765中斷信號的副程式（DISK_INT）。



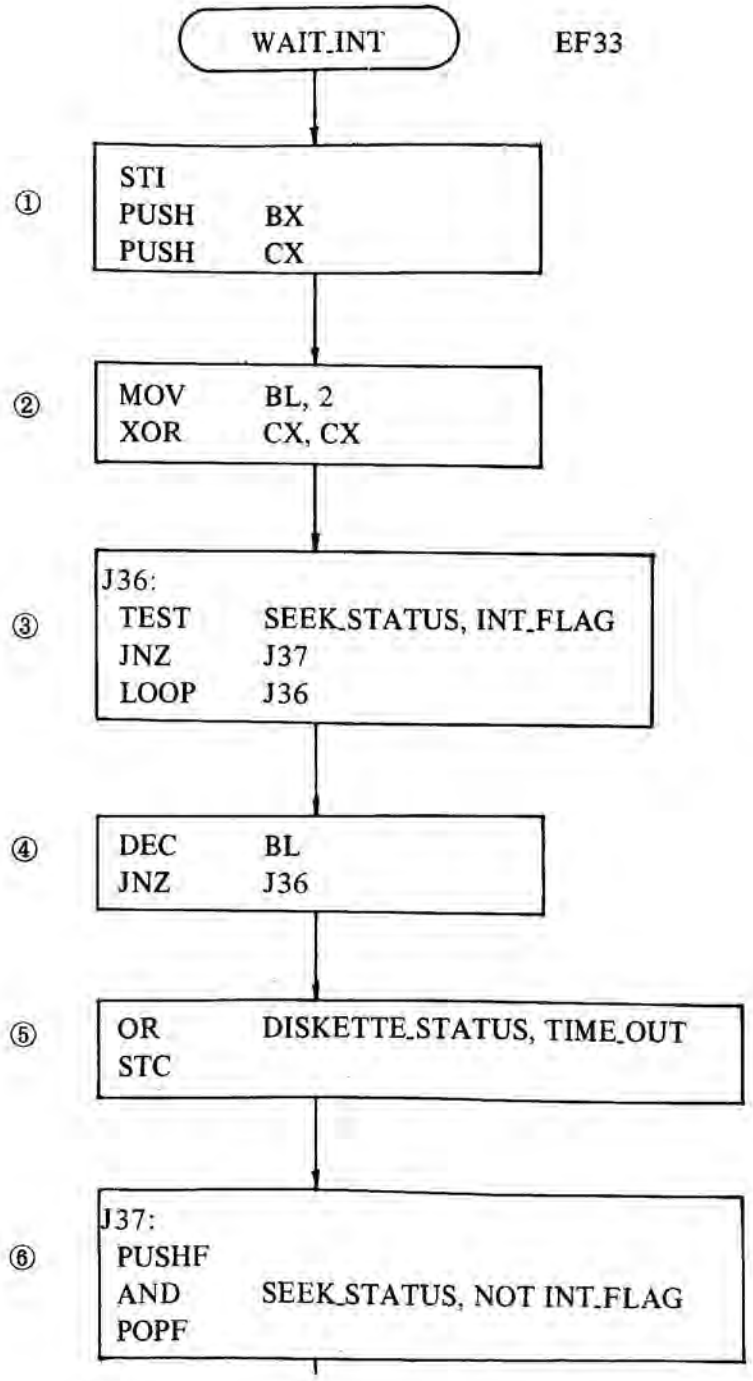


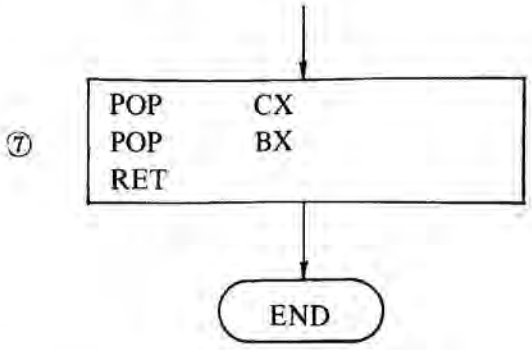
① 首先設定 IF 旗號爲 1，表示此時 CPU 仍可以接受中斷信號，然後依序將DS 、AX 暫存器保存在堆疊中，並且設定DS暫存器爲 0040 (DATA)。

② 將SEEK_STATUS的位元7 設定爲 1，INT_FLAG爲 80H；然後將 20H 寫到 8259 去，表示已處理完中斷信號了。

③ 依序取回AX, DS 暫存器，然後返回。

由上面的討論，我們知道DISK_INT 副程式只是將SEEK_STATUS 的位元7 設定爲 1 而已，知道這個觀念後，就比較容易討論WAIT_INT副程式了。





① 設定 IF 旗號爲 1，並依序將BX,CX 暫存器保存在堆疊中。

② 設定BL 暫存器爲 2，並將 CX 暫存器清除爲 0，以便做爲等待 INT產生的LOOP 時間。

③ 測試SEEK_STATUS的位元7 是否爲 1，若爲 1（ ZF 旗號爲 0 ），則表示 INT 已產生，跳入步驟⑥。若SEEK_STATUS的位元 7 不爲 1，則繼續測試，直到 CX 暫存器內的值變成零，才進入下個步驟。

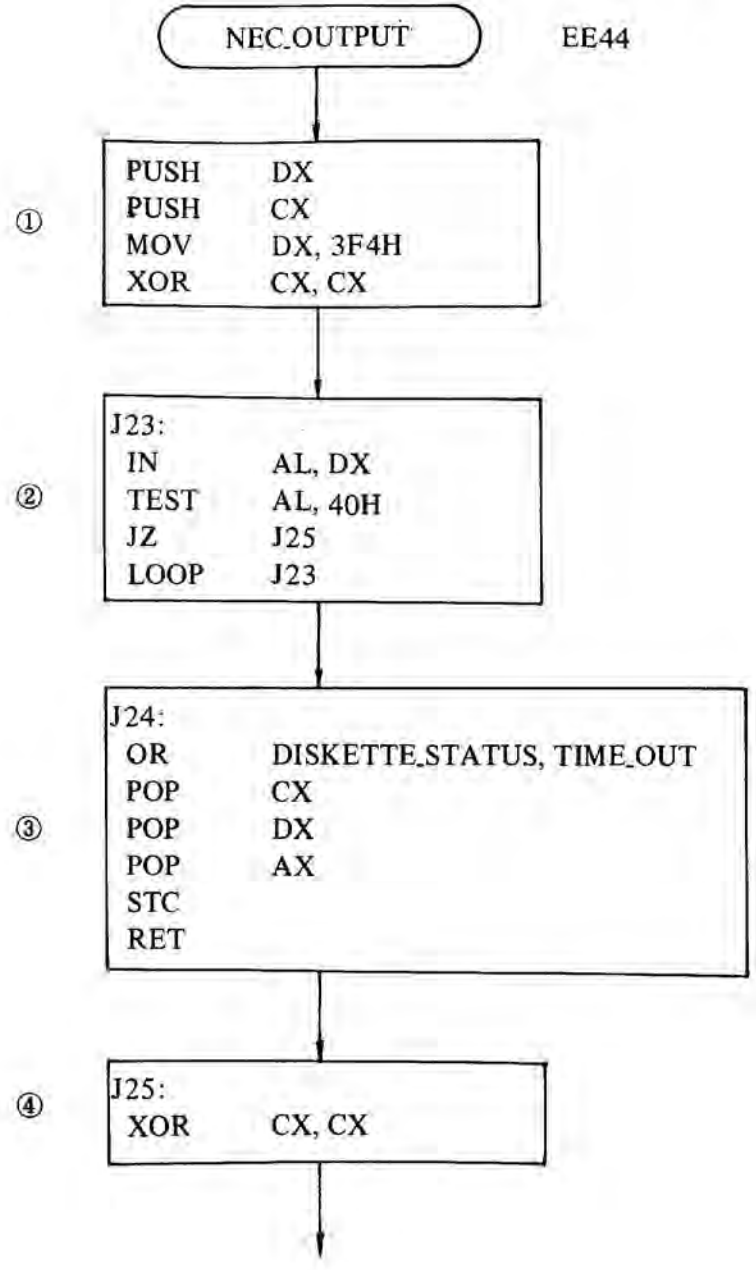
④ 將 BL 暫存器內的值減 1，若結果不爲 0（ ZF 旗號爲 0 ），則返回到步驟③，繼續測試 SEEK_STATUS的位元7，若結果爲 0，則進入下個步驟。

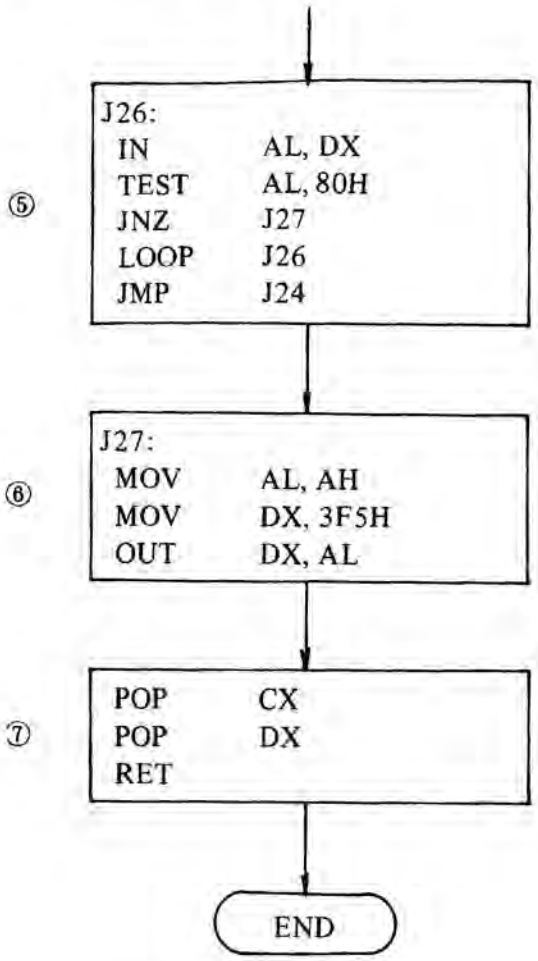
⑤ 進入此步驟是因爲沒有 INT 信號產生，將DISKETTE_STATUS的位元7 設定爲 1，表示 time out，並將CF 旗號設定爲 1。

⑥ 先將旗號暫存器保存在堆疊中，以免下個指令破壞了CF 旗號，然後將 SEEK_STATUS的位元 7 清除爲 0（ NOT INT_FLAG爲7FH ），再將原先的旗號暫存器取回。

⑦ 依序取回 CX 和BX 暫存器，然後返回到呼叫此程式的程式去。

NEC_OUTPUT





① 依序將DX和CX暫存器保存在堆疊中，然後在DX暫存器內放入3F4H，以便指向UPD765的狀態暫存器，並將CX暫存器設定爲0，以便配合LOOP指令。

② 讀出UPD765狀態暫存器內的值，並測試其值的位元6是否爲1，若不爲1（ZF旗號爲1），則表示資料滙流排是由CPU指向UPD765（即UPD765的資料暫存器此時要接收資料），跳入步驟④。若狀態暫存器內的值其位元6爲1，則必須要等

待其爲0爲止，若CX暫存器內的值已變成0，而狀態暫存器內的值其位元6仍爲1，則進入下個步驟，表示time_out。

③ 將Diskette_Status的位元7設定爲1（Time_out爲80H），然後依序取出CX，DX，AX暫存器內的值，並設定CF旗號爲1才返回。讀者一定會奇怪，在步驟①中我們只保存二個值，但這裏却取回三個值，其實這個步驟主要是配合READ/WRITE動作（在READ/WRITE動作時，再詳細討論）。在這裏讀者必須要特別小心了，如果A程式呼叫B程式，而B程式又呼叫NEC_OUTPUT副程式，當time out產生時，進入此步驟，此時是返回到A程式，而非B程式。這是因爲B程式的返回位址被POP AX指令取出，而RET指令眞正取到的返回位址是A的。

④ 清除CX暫存器，以便配合LOOP指令。

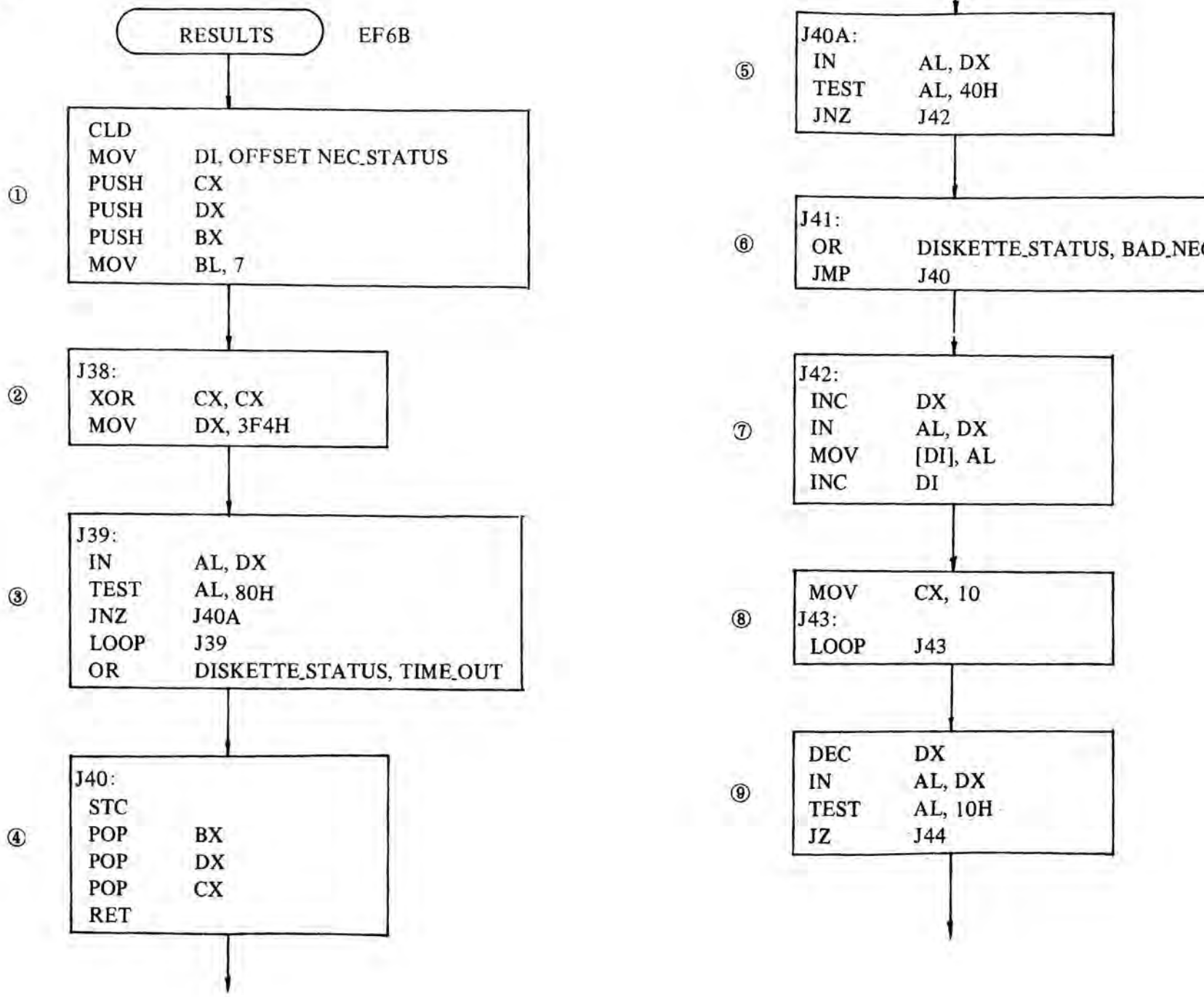
⑤ 讀UPD765的狀態暫存器內的值，並測試其值的位元7是否爲1，若爲1（ZF旗號爲0），則表示UPD765的資料暫存器已準備好了，可以接受資料，跳入步驟⑥。若UPD765的狀態暫存器內的值的位元7爲0，則表示資料暫存器尚未準備好，還得繼續等待。若CX暫存器內的值已LOOP到0了，則表示time out，跳入步驟③中。

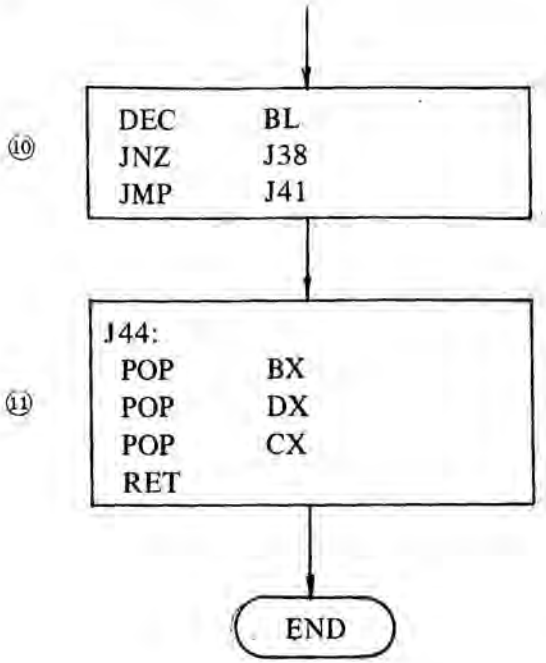
⑥ 進入此程式時，AH暫存器內的值爲要寫到UPD765的資料暫存器內的資料，首先移到AL暫存器內以便輸出，然後在DX暫存器內放入3F5H，以便指向資料暫存器，接下來的OUT指令，將資料寫入資料暫存器內。

⑦ 依序取出CX，DX暫存器，然後返回到呼叫NEC_OUTPUT副程式的程式去。

接著我們要討論如何讀出結果階段的資料。

RESULTS


RESULTS
EF6B
①
CLD
MOV DI, OFFSET NEC_STATUS
PUSH CX
PUSH DX
PUSH BX
MOV BL, 7
②
J38:
XOR CX, CX
MOV DX, 3F4H
③
J39:
IN AL, DX
TEST AL, 80H
JNZ J40A
LOOP J39
OR DISKETTE_STATUS, TIME_OUT
④
J40:
STC
POP BX
POP DX
POP CX
RET
⑤
J40A:
IN AL, DX
TEST AL, 40H
JNZ J42
⑥
J41:
OR DISKETTE_STATUS, BAD_NEC
JMP J40
⑦
J42:
INC DX
IN AL, DX
MOV [DI], AL
INC DI
⑧
MOV CX, 10
J43:
LOOP J43
⑨
DEC DX
IN AL, DX
TEST AL, 10H
JZ J44



① 首先清除DF 旗號，然後將OFFSET NEC_STATUS 放到DI 暫存器內（ OFFSET NEC_STATUS 是一個記憶體位址，段落爲 0040，OFFSET爲 0042 ），以便做爲指標，再依序將CX，DX，BX暫存器保存起來。在BL 暫存器內放入 7，是因爲結果階段最多只有 7 個位元組。

② 將 CX 暫存器清除爲 0，是爲了配合下個步驟的LOOP指令。在DX 暫存器內放入 3F4H是爲了指向UPD765的狀態暫存器

③ 讀UPD765的狀態暫存器內的值，並測試其位元 7 是否爲 1，若爲 1（ ZF 旗號爲 0 ），則表示UPD765的資料暫存器已準備好了（ ready ），進入步驟⑤。若UPD765的狀態暫存器內的值的位元 7 不爲 1（ ZF 旗號爲 1 ），則表示資料暫存器尚未準備好，還得繼續等待。如果CX 暫存器已LOOP到 0 了，則表示 time out，OR 指令將 diskette_status 的位元 7 設定爲 1（ time_out 爲 80H ），然後進入下個步驟。

④ 設定 CF 旗號爲 1，然後陸續取回BX，DX，CX 暫存器的初值，再返回到呼叫RESULTS 副程式的程式去。

⑤ 這個步驟測試UPD765的狀態暫存器的位元 6 是否爲 1，若爲 1（ ZF 旗號爲 0 ），則表示資料滙流排是由UPD765指向CPU，此時CPU 可以讀取資料暫存器內的資料（即結果狀態）。若UPD765的狀態暫存器的位元 6 爲 0（ ZF 旗號爲 1 ），則表示UPD765是不良的（結果狀態中，資料滙流排是指向CPU 的），進入下個步驟。

⑥ 將 diskette_status 的位元 5 設定爲 1（ BAD_NEC爲 20H），然後跳至步驟④返回。

⑦ 將 DX 暫存器指向UPD765的資料暫存器，然後讀出其內的值，並以 DI 暫存器內的值爲指標，將讀出的值存放到結果（ Results ）（ 00442H ）的第一個位址；然後將 DI 暫存器內的值加 1，以便指向下一個結果內的位址。

⑧ 從UPD765的資料表中，我們知道讀資料暫存器內的值，其間必須要有延遲的時間。這個步驟就是在做延遲。

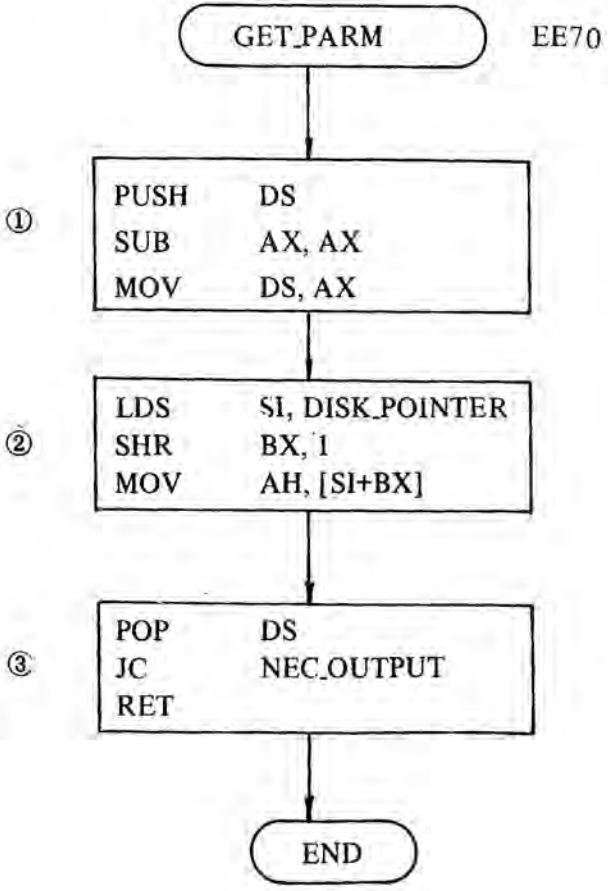
⑨ 將DX 暫存器指向狀態暫存器，並讀出其內的值。測試其位元 4 是否爲 1，若爲 0，則表示已經讀完了所有結果狀態的位元組（結果狀態的位元組被讀完後，狀態暫存器的位元 4（CB）會變成 0，表示不忙），進入步驟⑪。若狀態暫存器的位元 4 爲 1（ ZF 旗號爲 0 ），則表示結果階段內的位元組尚未被讀完，進入下個步驟。

⑩ 將 BL 暫存器內的值減 1，若不爲 0（ ZF 旗號爲 0 ），則表示從資料暫存器內取出的值尚不足 7 個（結果階段最多只有 7 個位元組），回到步驟②繼續讀結果階段。若 BL 暫存器已減

至0，則表示已讀出7個位元組，而UPD765仍是忙碌的，我們就認爲UPD765是不良的，跳入步驟⑥，設定 BAD_NEC 後返回。

⑪ 陸續取出BX，DX，CX暫存器內的值後返回到呼叫Results副程式的程式去。

在Disk_Reset 程式中，我們還呼叫了GET_PARM 副程式，我們現在就來討論它。



① 先將DS 暫存器內的值保存起來，再將 DS 暫存器設定爲0000，以便在下個步驟中，取出DISK_POINTER 內的值。

② 這個程式主要是爲了取出disk_BASE內的資料。disk_BASE是11個記憶體位址，而其起始位址放在實際位址00078H起的四個位址內。（也就是INT 1EH）。LDS指令將INT 1EH的分段值放到DS暫存器內，間距值放到SI 暫存器內，如此DS：SI就指著disk_BASE的起始位址。進入此程式的BX 暫存器內的值指出要取出那一個值，這裏將其右移一位（位元0移入CF 旗號）是爲了在下一個步驟中，決定是否要經由NEC_OUTPUT 輸出。接下來的MOV 指令取出所要的資料於AH 暫存器內。

③ 恢復DS 暫存器的初值，然後測試CF旗號，以便決定是否要由NEC_OUTPUT副程式返回。

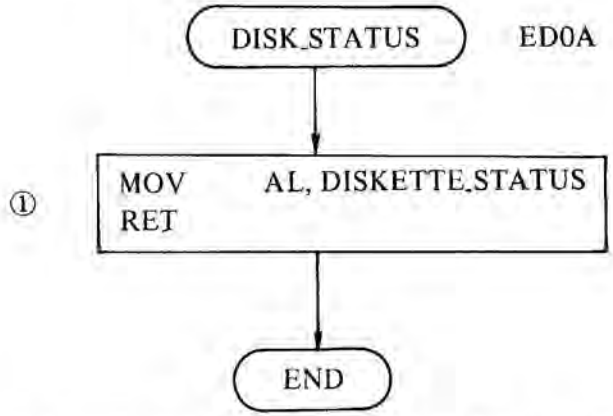
由上面的討論讀者可以看出，如果要經由NEC_OUTPUT 副程式，則BX暫存器必須是奇數。若讀者想取第6個位元組，且經由NEC_OUTPUT，則BX暫存器內的值爲000BH（disk_BASE由0起算）。若不經由NEC_OUTPUT，則BX暫存器內的值爲000AH。

disk_BASE的實際位址在FEFC7H。我們列出其內的資料於下：

segment : F000	offset : EFC7	CF	1st specify byte
	EFC8	2	2nd specify byte
	EFC9	25	motor wait
	EFCA	2	512 Bytes/Sector
	EFCB	8	EOT
	EFCC	2A	gap length

EFCD	FF	DTL
EFCE	50	gap length for format
EFCF	F6	fill byte for format
CFD0	25	head settle time
CFD1	4	motor start time

(b) (AH)=1, Read status



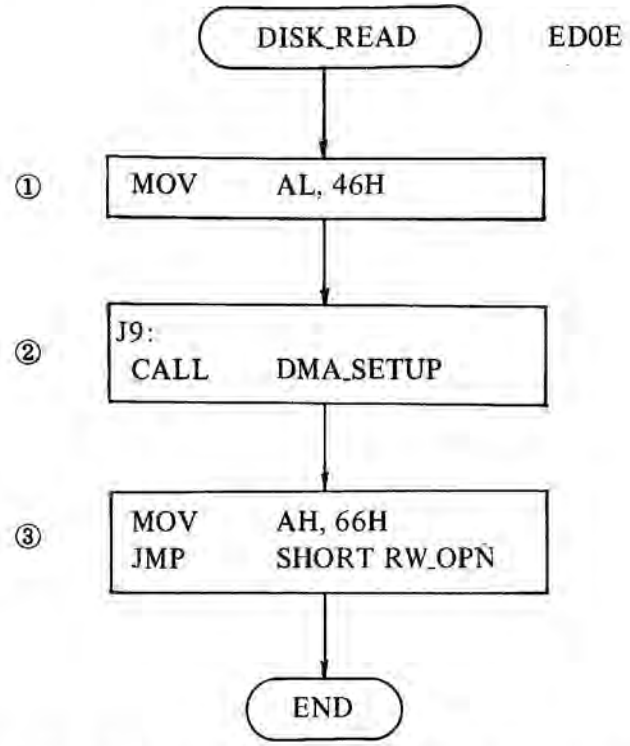
① 將DISKETTE_STATUS內的值放到AL暫存器內後返回。

這個程式只有一個步驟而已，我們來看DISKETTE_STATUS各位元的意思。

bit 7	time out
bit 6	Bad seek
bit 5	Bad NEC
bit 4	Bad CRC
bit 3	Bad DMA
bit 2	Record not find
bit 1	Bad address mark
bit 0	Bad command

註：位元3和位元0同時爲1，則表示超出DMA邊界(boundary)；
位元1和位元0同時爲1，則表示防止寫入。

(c) (AH)=2, Read Sector



① 在AL暫存器內放入46H，是爲了配合DMA_SETUP副程式，設定8237之用。

② 呼叫DMA_SETUP副程式，設定DMA的起始位址。

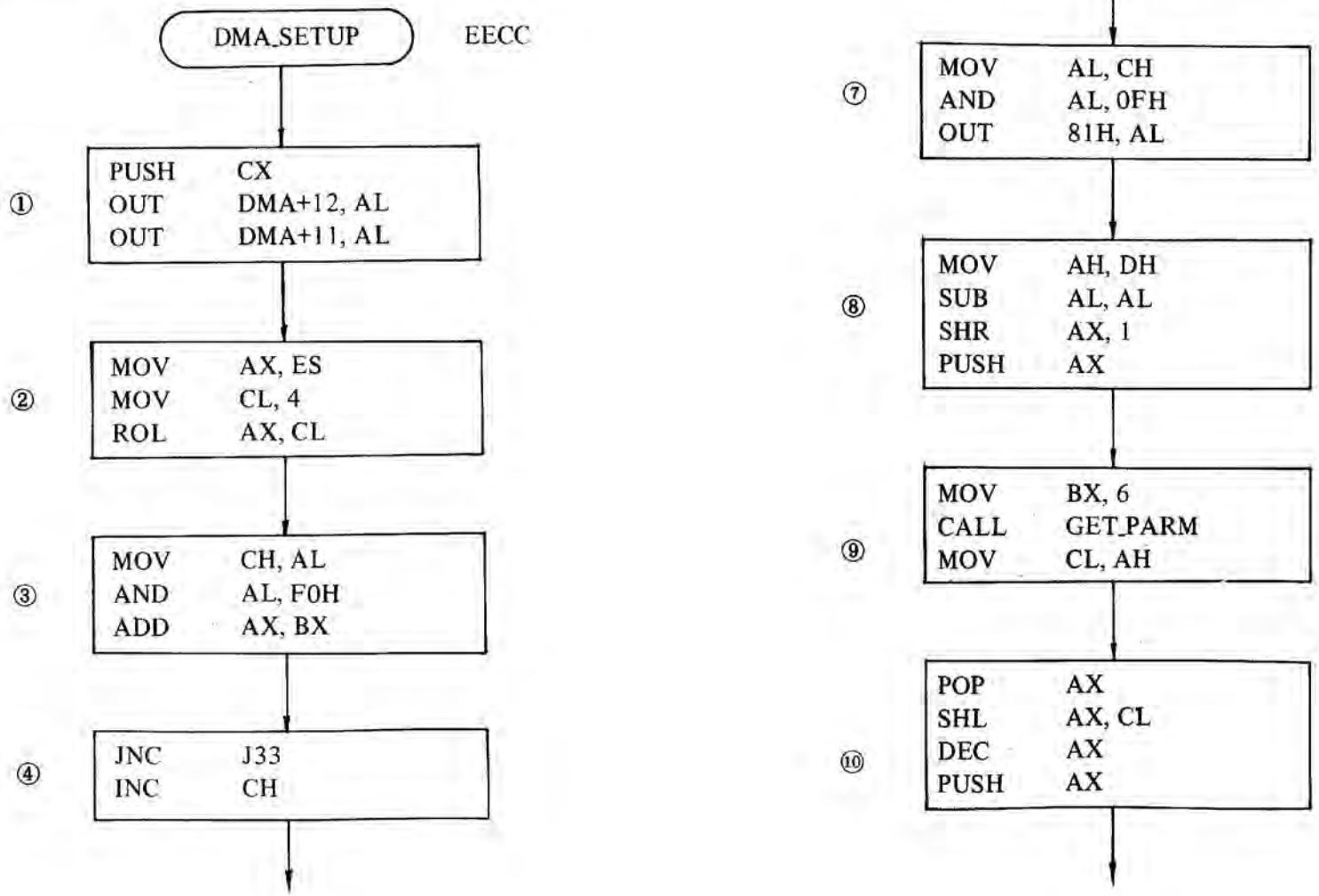
③ 在AH暫存器內放入66H後，跳入RW_OPN 副程式，以便執行讀取磁區的動作。66H是UPD765 讀取磁區的第一個命令位元組 (command byte)。

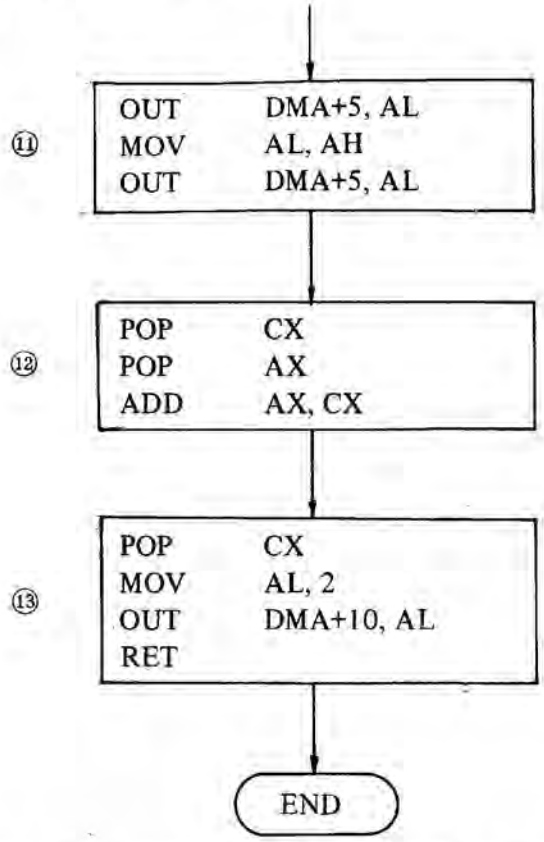
其實整個的Read/Write/format/verify 動作都是這麼短的

，主要的工作是在DMA_SETUP副程式和RW_OPN 副程式完成的。而 read/write/format/verify 的區分，在於進入DMA_SETUP 副程式的AL 暫存器內的值和進入RW_OPN 副程式的AH 暫存器內的值。

DMA_SETUP

我們先來看DMA_SETUP 副程式。





① 先將CX暫存器內的值保存起來，以免被破壞。接下來的OUT指令只是清除byte pointer Flip/Flop，與AL暫存器內的值無關。第二個OUT指令將AL暫存器內的值寫入8237的模式暫存器（mode register）。請讀者自行參考8237的模式暫存器，下面列出read/write/format/verify動作，寫入模式暫存器的值：

(a) read : (AL) = 46H
(b) verify : (AL) = 42H
(c) format : (AL) = 4AH
(d) write : (AL) = 4AH

② 我們知道實際位址是段落內的值左移4個位元再加上間距值。8237處理DMA是以實際位址來找記憶體位址。所以步驟②到④是將邏輯位址ES：BX轉換成實際位址，並寫入DMA PAGE暫存器（LS670）和8237的位址暫存器（Address Register），此步驟將ES內的值轉到AX暫存器，並左旋轉4個位元。如果ES暫存器內的值爲ABCD，執行完此步驟後，AX暫存器內的值爲BCDA。

③ 先將AL暫存器內的值（上例中爲DA）保存在CH暫存器內。剔除掉AL暫存器的低半位元組（lower nibble）後，將AX暫存器內的值加上BX暫存器內的值。請看下列：若ES暫存器內的值爲ABCDH，BX暫存器內的值爲0101H。

(a) 左旋轉ES暫存器內的值4個位元，得到BCDAH（存於AH暫存器）。
(b) 剔除掉AL暫存器的低半位元組，得到BCD0H。
(c) 加上BX暫存器內的值得到BDD1H。

如此即得到實際位址的最後4位數，最高的位數在CH暫存器內（實際位址爲ABDD1H）。

④ 若在上個步驟中，ADD AX,BX的結果有進位，則CF旗號爲1，必須將實際位址的最高位數加1，若沒有進位，則不必加1。

⑤ 將AX暫存器（實際位址的最低4位數）的值保存在堆疊中，然後將低有效位元組（Lower significant byte）寫到通道2的基準和現在位址暫存器（Base and current address register）去。

⑥ 此步驟將高有效位元組（higher significant.byte）寫到通道2的基準和現在位址暫存器去。

⑦ 將CH暫存器內的值（實際位址的最高位數）轉到AL暫存器

內，並剔除掉高半位元組（higher nibble）（因爲DMA PAGE暫存器只接受低半位元組）後，寫進DMA PAGE暫存器（LS670）內，如此即設定好了DMA的起始位址。

⑧ 進入此程式的DH暫存器內的值指出要讀多少個磁區（sector）（磁區數目的值原本在AL暫存器內，在J1副程式中已轉到DH暫存器了）。這裏將DH暫存器內的值轉到AH暫存器內，並將AL暫存器清除爲0。讀者可以把這個看成將磁區數目乘以256，再將此值右移一個位元，讀者可以看成將磁區數目乘以128，然後將此值保存在堆疊中。

⑨ 在BX暫存器內放入6，經由GET_PARM副程式取出2（512位元組／磁區），並將此值轉到CL暫存器去。

⑩ 取出步驟⑧保存的值，然後將此值左移2個位元（因爲步驟⑨取到的值爲2），讀者可以將此值看成磁區數目乘以512（因爲一個磁區有512位元組），也就是要讀取／寫入的位元組。將此值減1，是因爲8237的字組數暫存器（word count register）是以0起算，最後將此值保存起來以免被破壞。

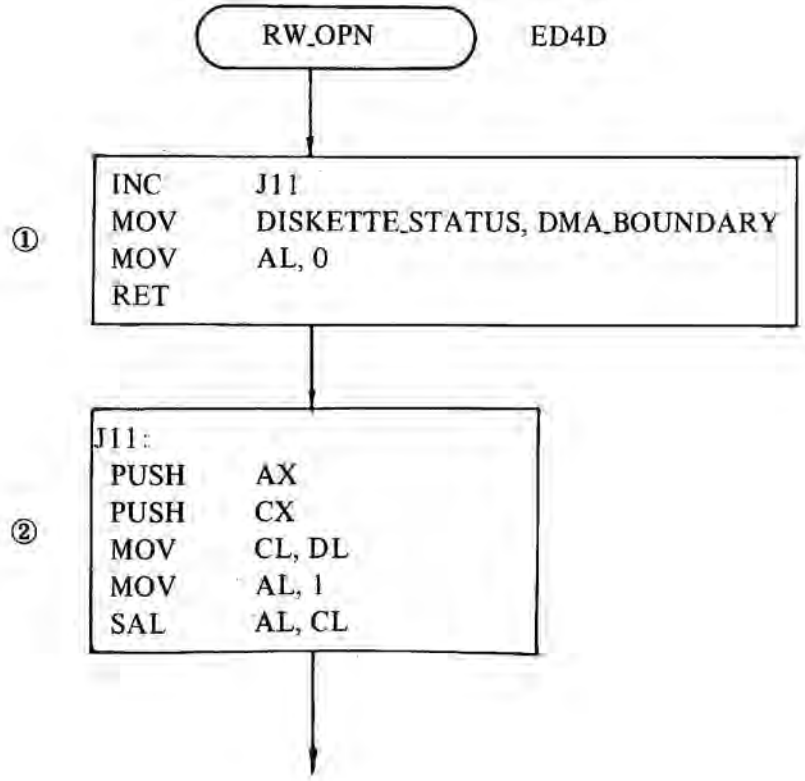
⑪ 此步驟將總共要讀取／寫入的位元組數寫到8237的通道2的基準和現在字組數暫存器（Base and current word count register）去，如此即設定好了8237要傳輸的位元組數。

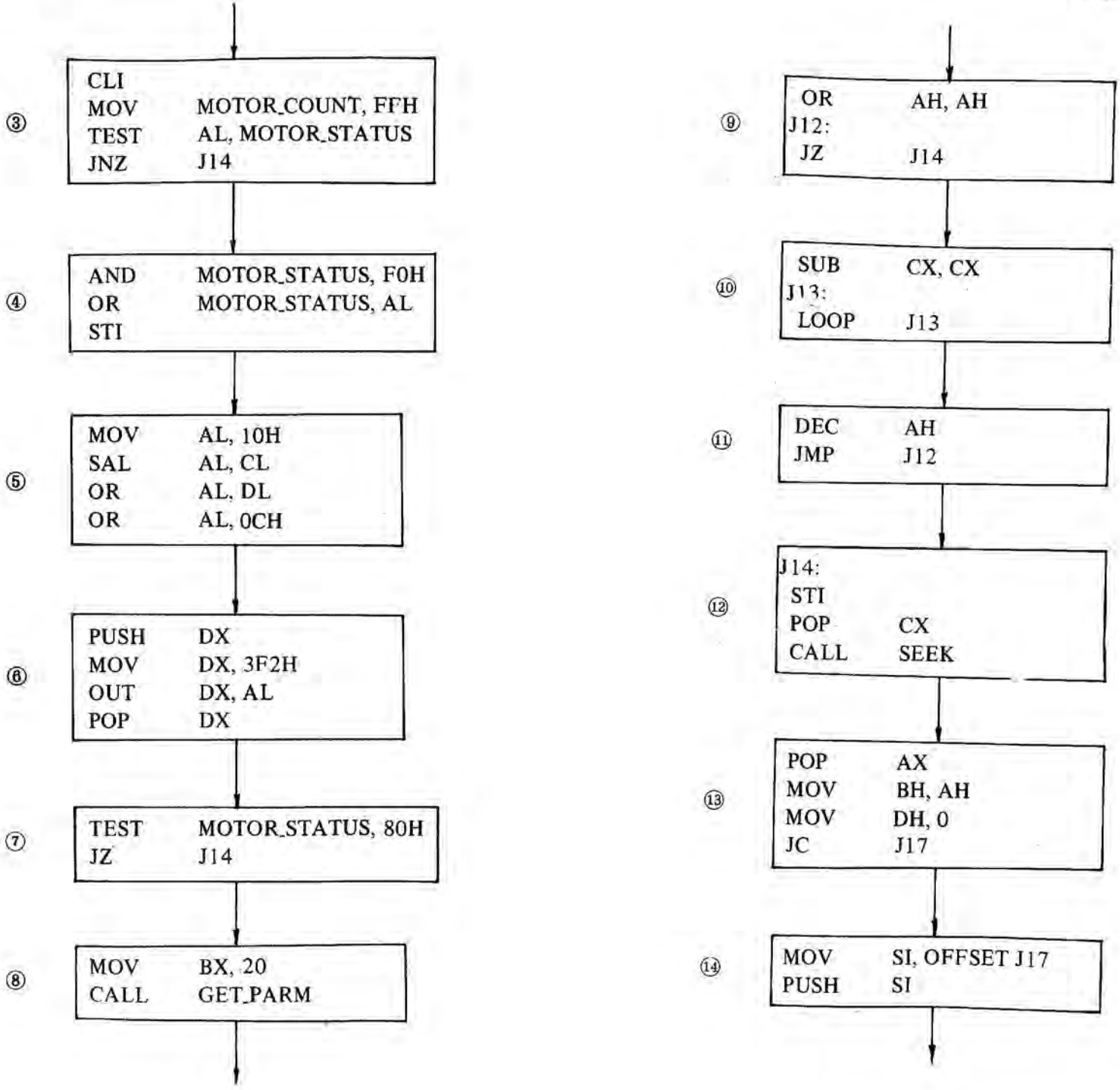
⑫ 第1個pop指令將步驟⑩儲存的位元組數取出放到CX暫存器去，第2個pop指令將步驟⑤儲存的實際位址的最低4位數取出放到AX暫存器內，然後將這二個值相加，相加的目的是要測試是否有進位，若有進位，CF旗號爲1，這表示8237的位址暫存器內的值加上字組數暫存器內的值超過64K，而8237無法處理超過64K的部分（因爲位址暫存器只有16個位元），這種情況我們稱之爲超出DMA邊界。在這裏並不處理超出DMA邊界的情況，只設定CF旗號，交由RW_OPN副程式處理。

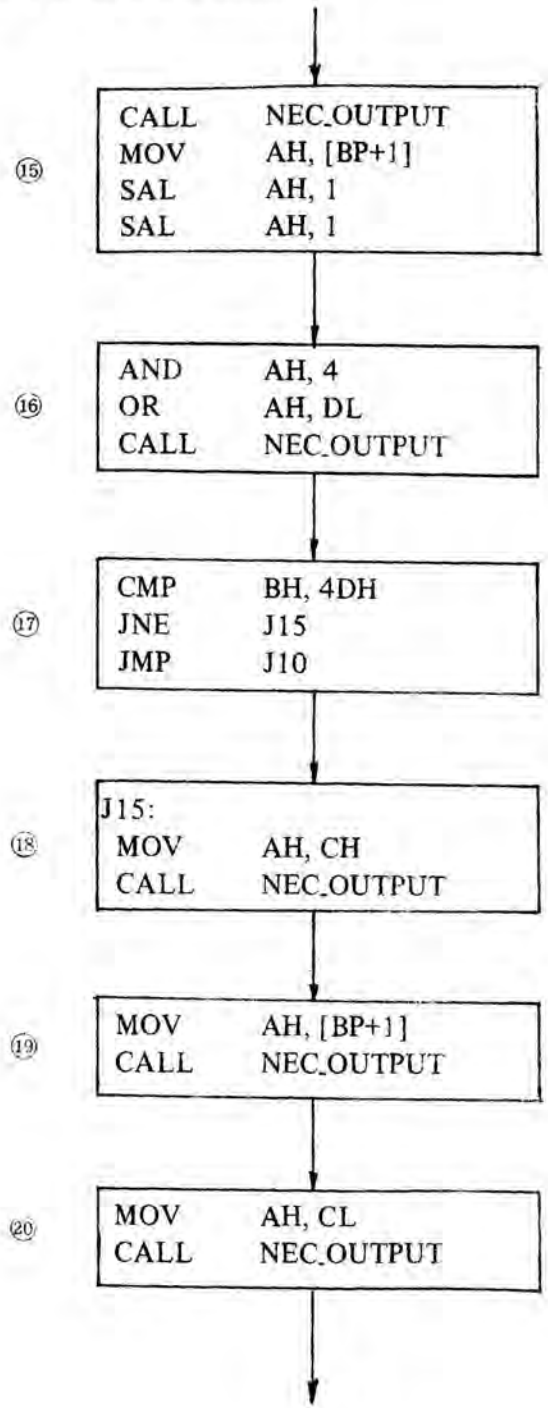
⑬ 首先取出在步驟①保存的CX暫存器的值，然後將2寫到遮罩暫存器（Mask Register）（此OUT指令爲寫入單一遮罩暫存器位元），如此即將8237的通道2的遮罩位元清除爲0，也就是說8237可以接受DRQ2的要求信號。

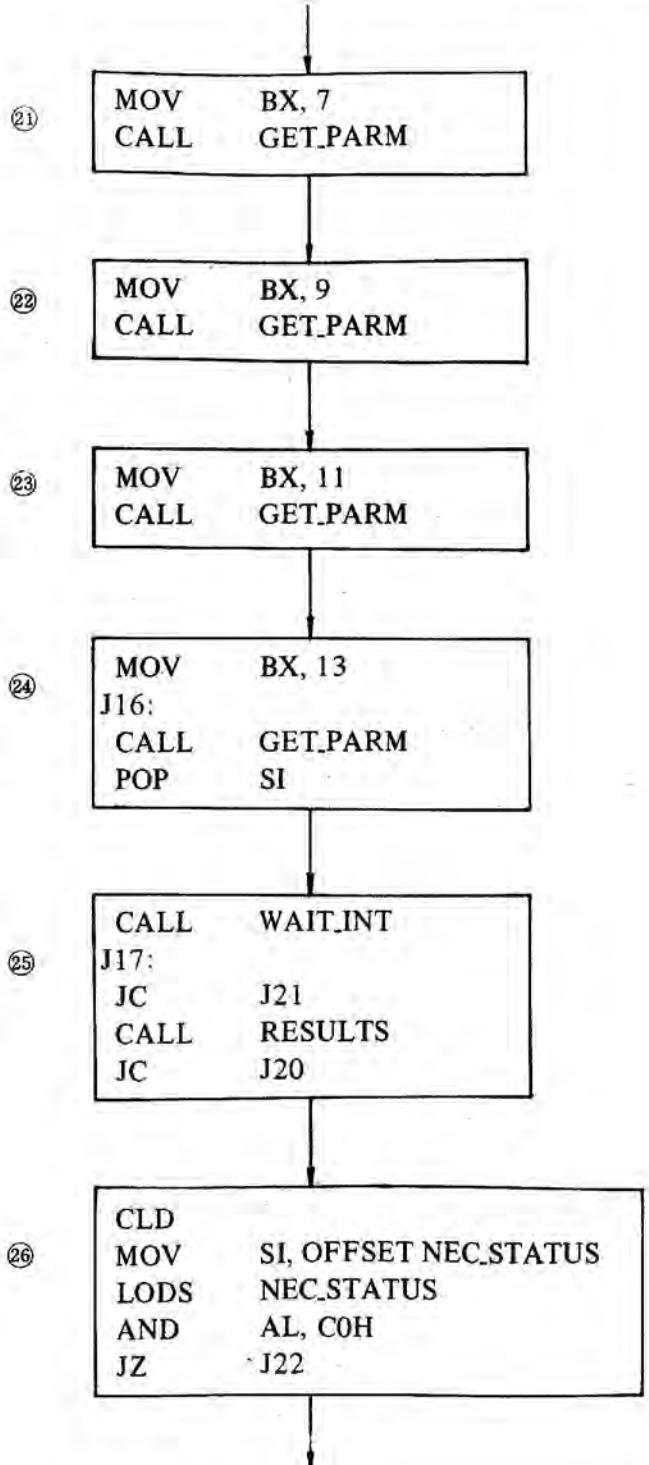
DMA_SETUP副程式就執行到這裏，現在只等RW_OPN副程式將命令位元組寫到UPD765的資料暫存器後，就開始做DMA的動作。

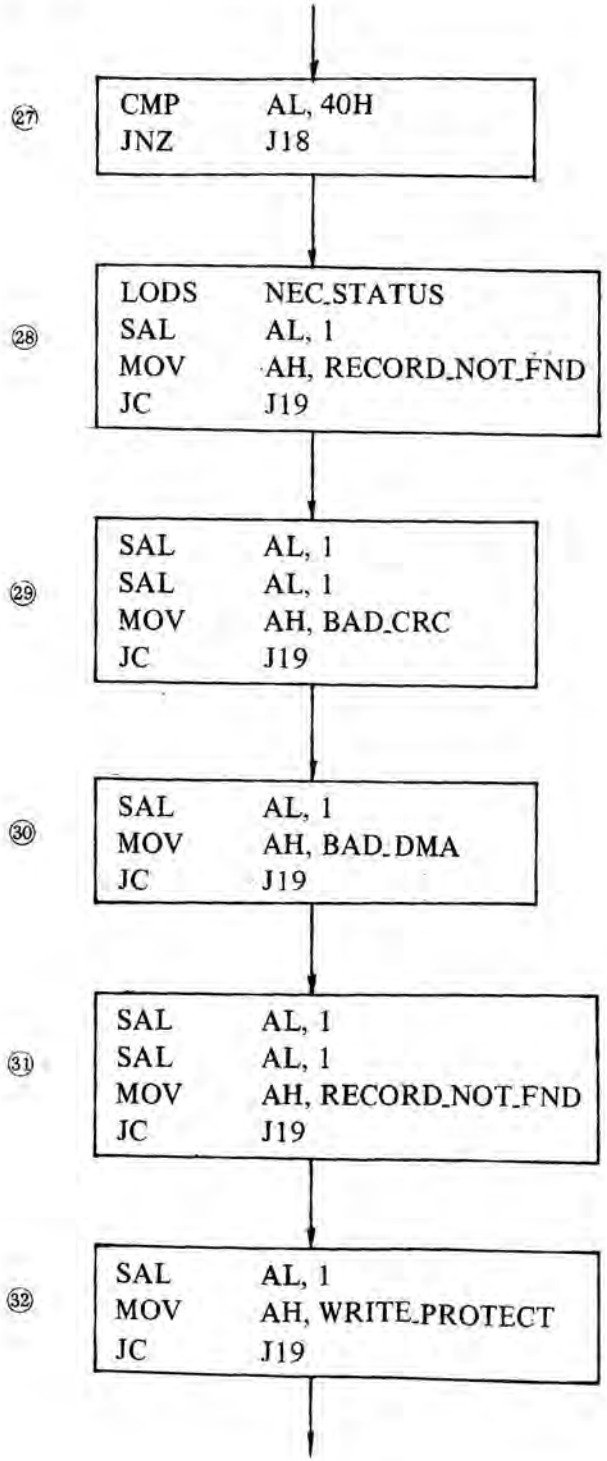
RW_OPN

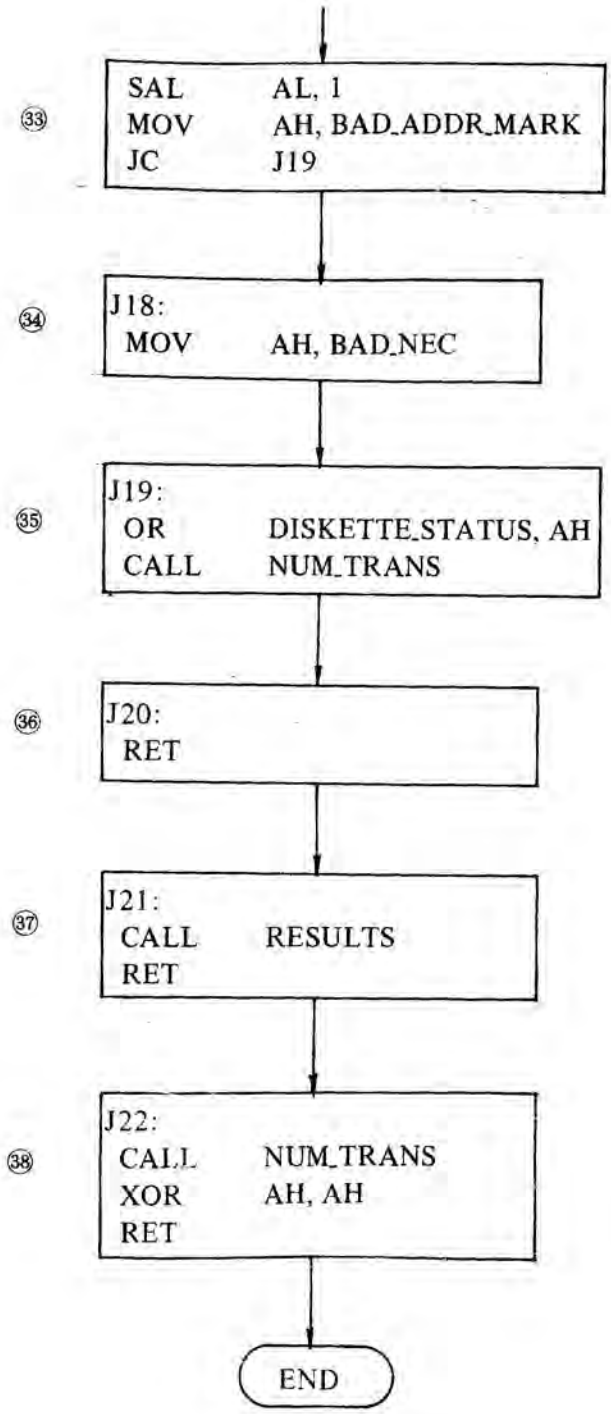



③
CLI
MOV MOTOR_COUNT, FFH
TEST AL, MOTOR_STATUS
JNZ J14
④
AND MOTOR_STATUS, F0H
OR MOTOR_STATUS, AL
STI
⑤
MOV AL, 10H
SAL AL, CL
OR AL, DL
OR AL, 0CH
⑥
PUSH DX
MOV DX, 3F2H
OUT DX, AL
POP DX
⑦
TEST MOTOR_STATUS, 80H
JZ J14
⑧
MOV BX, 20
CALL GET_PARM
⑨
OR AH, AH
J12:
JZ J14
⑩
SUB CX, CX
J13:
LOOP J13
⑪
DEC AH
JMP J12
⑫
J14:
STI
POP CX
CALL SEEK
⑬
POP AX
MOV BH, AH
MOV DH, 0
JC J17
⑭
MOV SI, OFFSET J17
PUSH SI


⑮
CALL NEC_OUTPUT
MOV AH, [BP+1]
SAL AH, 1
SAL AH, 1
⑯
AND AH, 4
OR AH, DL
CALL NEC_OUTPUT
⑰
CMP BH, 4DH
JNE J15
JMP J10
⑱
J15:
MOV AH, CH
CALL NEC_OUTPUT
⑲
MOV AH, [BP+1]
CALL NEC_OUTPUT
⑳
MOV AH, CL
CALL NEC_OUTPUT


㉑
MOV BX, 7
CALL GET_PARM
㉒
MOV BX, 9
CALL GET_PARM
㉓
MOV BX, 11
CALL GET_PARM
㉔
MOV BX, 13
J16:
CALL GET_PARM
POP SI
㉕
CALL WAIT_INT
J17:
JC J21
CALL RESULTS
JC J20
㉖
CLD
MOV SI, OFFSET NEC_STATUS
LODS NEC_STATUS
AND AL, C0H
JZ J22


㉗
CMP AL, 40H
JNZ J18
㉘
LODS NEC_STATUS
SAL AL, 1
MOV AH, RECORD_NOT_FND
JC J19
㉙
SAL AL, 1
SAL AL, 1
MOV AH, BAD_CRC
JC J19
㉚
SAL AL, 1
MOV AH, BAD_DMA
JC J19
㉛
SAL AL, 1
SAL AL, 1
MOV AH, RECORD_NOT_FND
JC J19
㉜
SAL AL, 1
MOV AH, WRITE_PROTECT
JC J19


㉝
SAL AL, 1
MOV AH, BAD_ADDR_MARK
JC J19
㉞
J18:
MOV AH, BAD_NEC
㉟
J19:
OR DISKETTE_STATUS, AH
CALL NUM_TRANS
㊱
J20:
RET
㊲
J21:
CALL RESULTS
RET
㊳
J22:
CALL NUM_TRANS
XOR AH, AH
RET
END

① 首先測試CF 旗號，若CF 旗號爲1，則表示超出DMA 邊界，將Diskette_status 的位元3和位元0設定成1，並清除AL 暫存器內的值後返回。若CF 旗號爲0，進入下個步驟。

② 先將AX和CX 暫存器內的值保存起來，然後將DL 暫存器內的值（指出磁碟機）轉到CL 暫存器內，在AL 暫存器內放入1，再將AL 左移CL 次，這樣做的用意是在測試我們要使用的磁碟機的馬達是否正在旋轉。

我們看下表所選擇的磁碟機和執行完此步驟後AL 暫存器內的值。

所選擇的磁碟機	DL 暫存器內的值	AL 暫存器內的值
drive A	00	00000001
drive B	01	00000010
drive C	02	00000100
drive D	03	00001000

③ 首先清除IF 旗號，使得在執行步驟③和④時不被打斷。在MOTOR_COUNT中放入FFH，以便返回時做爲停止馬達的等待時間。我們已經知道MOTOR_STATUS的位元3到位元0是用來顯示磁碟機的馬達是否正在旋轉。接下來的TEST指令就是用來測試我們所選擇的磁碟機的馬達是否正在旋轉，若測試後，ZF 旗號爲0，就表示我們所選擇的磁碟機的馬達正在旋轉，進入步驟⑫做read/write/verify/format 動作。若測試後，ZF 旗號爲1，即表示我們所選擇的磁碟機的馬達正處於停止狀態，進入下個步驟使它旋轉。

④ 首先清除MOTOR_STATUS的位元3到位元0，然後再將相對於磁碟機的位元設定成1。

drive A ——→bit 0
drive B ——→bit 1
drive C ——→bit 2
drive D ——→bit 3

最後設定IF旗號爲1。

⑤ 此步驟是在設定磁碟機界面卡的數位輸出暫存器的輸出值。我們一個指令一個指令地來討論。

(a) MOV AL,10H
SAL AL,CL （CL 暫存器內的值爲0到3，指出磁碟機）。

執行完這二個指令後，AL 暫存器內的值將相對於數位輸出暫存器的磁碟機。

AL 暫存器內的值	相對的磁碟機
10H	drive A
20H	drive B
40H	drive C
80H	drive D

(b) OR AL,DL （DL 暫存器內的值爲0到3，指出磁碟機）

執行這個指令後，AL 暫存器內的值和所選擇的磁碟機關係如下：

AL 暫存器內的值	所選擇的磁碟機
10 H	drive A
21 H	drive B
42 H	drive C
83 H	drive D

(c) OR AL,0CH 指令將 AL 暫存器內的值的位元3和位元2均設定爲1（不論所選擇的Drive），如此做是爲了不須重置UPD765 和開啓INT & DMA 要求。

⑥ 先將 DX 暫存器內的值保存起來，然後將 I/O 埠指向數位輸出暫存器，並將步驟⑤得到的值寫到數位輸出暫存器。如此，我們選擇的磁碟機馬達會轉動，LED 燈會亮，最後恢復 DX 暫存器的初值。

⑦ 測試MOTOR_STATUS的位元7是否爲1，若爲1（ZF旗號爲0）則表示爲寫入動作，進入下個等待馬達旋轉平穩。若結果不爲1（ZF旗號爲1），則表示爲讀取動作，不必等待馬達旋轉平穩，直接進入步驟⑫。

⑧ 在 BX 暫存器內放入 20（十進位），然後呼叫GET_PARM副程式，取出馬達起動時間到AH 暫存器內。

⑨ 測試AH 暫存器內的值是否爲0，若爲0（ZF 旗號爲1），則表示所要等待的時間已過，進入步驟⑫。若結果不爲0（ZF 旗號爲0），則進入下個步驟，執行等待時間的迴圈指令。

⑩ 此步驟等待LOOP 64K 次的時間。

⑪ 將AH 暫存器內的值減1，然後跳至步驟⑨去測試是否等待時間已過。

⑫ 首先設定 IF 旗號爲1，然後從堆疊中取回磁軌號碼和磁頭號碼於CX 暫存器內，再呼叫SEEK副程式，將磁頭移至我們所選擇的磁軌去。SEEK副程式待會討論。

⑬ 從堆疊中取回要寫到UPD765的第一個命令位元組，並且將其保存在 BH 暫存器內。在DH 暫存器內放入0，以便在SEEK動作發生錯誤時，用來表示沒有磁區被讀出。如果SEEK動作有錯誤時，CF 旗號爲1，此時要中止一切動作，跳入步驟㉕。若SEEK動作OK,CF 旗號爲0，進入下個步驟。

⑭ 在NEC_OUTPUT 副程式的步驟③中，我們已說明了若有錯誤產生的話，返回位址是取自第二個保存在堆疊中的位址。在這裏我們先將 J17 的間距值保存於堆疊，以便在下面的步驟中呼叫NEC_OUTPUT 副程式產生錯誤時，返回到 J17（步驟㉕）去處理錯誤（CALL指令會將返回位址保存在堆疊中，若有錯誤產生，會取到OFFSET J17，而非正常的返回位址，若沒有錯誤產生，還是會取到CALL指令的返回位址。）

⑮ 首先呼叫NEC_OUTPUT 副程式，將要寫到UPD765的第一個命令位元組（在 AH暫存器內）寫到UPD765的資料暫存器。在INT 13H副程式的步驟②，我們已經說明了BP 暫存器所指的位址內的值保存著磁頭號碼和磁碟機號碼。這裏的MOV AH,[BP+1]就是取出磁頭號碼，並將其左移2個位元，這是爲了配合第二個命令位元組的格式。

⑯ 這裏的 AND 指令是單單取出關於磁頭號碼的位元，然後再OR 上磁碟機號碼（在DL 暫存器內），如此 AH暫存器內的值就符合了第2個命令位元組的格式（請讀者參考 read/write/format 的第2個命令位元組）。然後呼叫NEC_OUTPUP 副程式將其寫到UPD765的資料暫存器。

⑰ 在步驟⑬中，我們將第1個命令位元組保存在 BH 暫存器內，這裏將其和 4DH 比較，若相等（ZF 旗號爲1），則爲format 指令，跳入 J10（在 format 動作內，待會再討論）去處理 format 動作，然後再跳至步驟㉔。若不是 format 動作，進入下個步驟。

⑱ 將磁軌號碼（在 CH 暫存器內）寫到UPD765的資料暫存器。

⑲ 將磁頭號碼（爲 BP 暫存器內的值所指的位址內的高位元組）寫到UPD765的資料暫存器。

⑳ 將磁區號碼（在CL 暫存器內）寫到UPD765 的資料暫存器。

㉑ 在BX 暫存器內放入7，取出資料位元組的數目（N），並寫

到UPD765的資料暫存器（因爲BX 暫存器內的值爲奇數，所以會經過NEC_OUTPUT副程式）。

㉒ 在BX 暫存器內放入9，取出 end of track（EOT），並寫到UPD765的資料暫存器。

㉓ 在BX 暫存器內放入11，取出Gap length（GPL），並寫到UPD765的資料暫存器。

㉔ 在BX 暫存器內放入13，取出Data Length（DTL），並寫到UPD765的資料暫存器（讀取/寫入動作的命令位元組都是9個，所以是一連串的寫資料到UPD765的資料暫存器的動作）。以下的步驟都不再使用NEC_OUTPUT 副程式，所以取出間距（OFFSET）J17 的值。

㉕ 以上的步驟已將命令位元組寫到UPD765了，這時候呼叫WAIT_INT副程式，以便等待UPD765做完動作後發出INT信號。從WAIT_INT 副程式回來時，若CF 旗號爲1，表示有錯誤（UPD765沒有發出 INT 信號），跳至步驟㊲。若從WAIT_INT 副程式回來時，CF 旗號爲0，則表示UPD765已發出 INT 信號，此時UPD765已進入結果狀態，呼叫RESULTS 副程式，讀結果階段的位元組。若從RESULTS副程式回來時，CF 旗號爲1，則表示有錯誤產生，跳入步驟㊱；若從RESULTS副程式回來時，沒有錯誤產生（CF 旗號爲0），進入下個步驟。

㉖ 設定DF 旗號爲0，以便配合LODS指令，在SI 暫存器內放入OFFSET NEC_STATUS，以便取出結果位元組（result bytes）。LODS 指令取出結果階段的第一個位元組，並測試其位元7和位元6是否均爲0，若均爲0（ZF 旗號爲1），則表示一切正確，進入步驟㊳（請讀者自行參考read/write/format的結果階段和ST0）。若ZF 旗號爲0（結果階段的第1個位元組的位元7和位元6不均爲0），則表示有錯誤產生，進入下個步驟找出錯誤原因。

㉗ 測試AL 暫存器內的值（即狀態暫存器0）的位元7和位元6是否爲0和1，若是（ZF 旗號爲1），則表示爲不正常終止（abnormal termination），進入下個步驟。若結果爲否（ZF 旗號爲0），則認爲UPD765是不良的，進入步驟㉞。

㉘ 在上一次執行LODS指令時，已將SI 暫存器內的值加1（因爲DF 旗號爲0），這裏的LODS指令取出結果階段的第2個位元組（即狀態暫存器1）到AL 暫存器內，並將其位元7左移至CF 旗號中，在AH 暫存器內放入RECORD_NOT_FND（04H），以便CF 旗號爲1時，進入J19（步驟㉟），OR入DISKETTE_STATUS內。從此步驟到步驟㉝，若CF旗號爲1，則表示錯誤的種類是預先存入 AH 暫存器內的那種，以便在步驟㉟中，OR入DISKETTE_STATUS 內。

㉙ 此步驟測試是否爲BAD_CRC。

㉚ 此步驟測試是否爲BAD_DMA。

㉛ 此步驟測試是否爲RECORD_NOT_FND。

㉜ 此步驟測試是否爲WRITE_PROTECT。

㉝ 此步驟測試是否爲BAD_ADDR_MARK。（請讀者自行參考狀態暫存器1和DISKETTE_STATUS，DISKETTE_STATUS 在DISK_STATUS副程式中，有詳盡的說明）。

㉞ 若不是上述各種的錯誤，我們就認爲UPD765 是不良的。則在AH 暫存器內放入BAD_NEC（20H），以便在下個步驟中，，OR入DISKETTE_STATUS內。

㉟ 將AH 暫存器內的值OR 入DISKETTE_STATUS 內，然後呼叫NUM_TRANS副程式，決定讀寫了若干磁區。

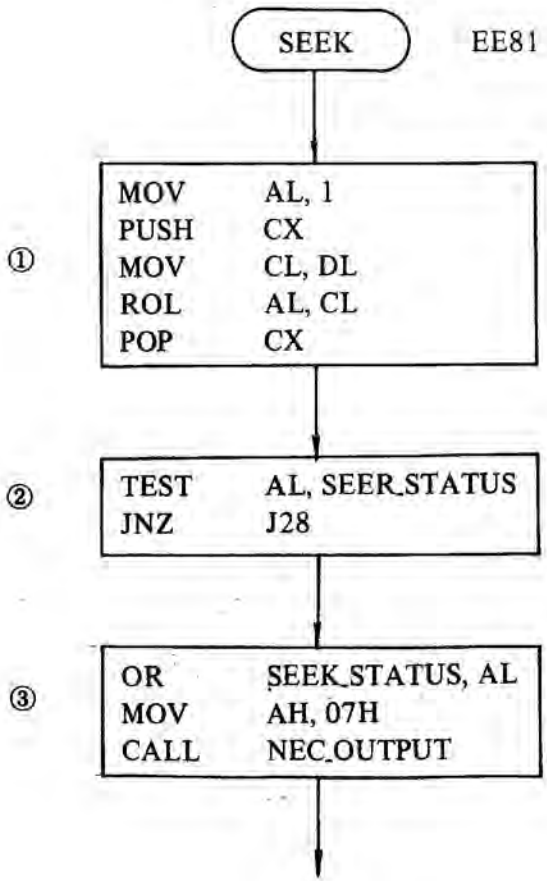
㊱ 返回到 INT 13H副程式的步驟④。

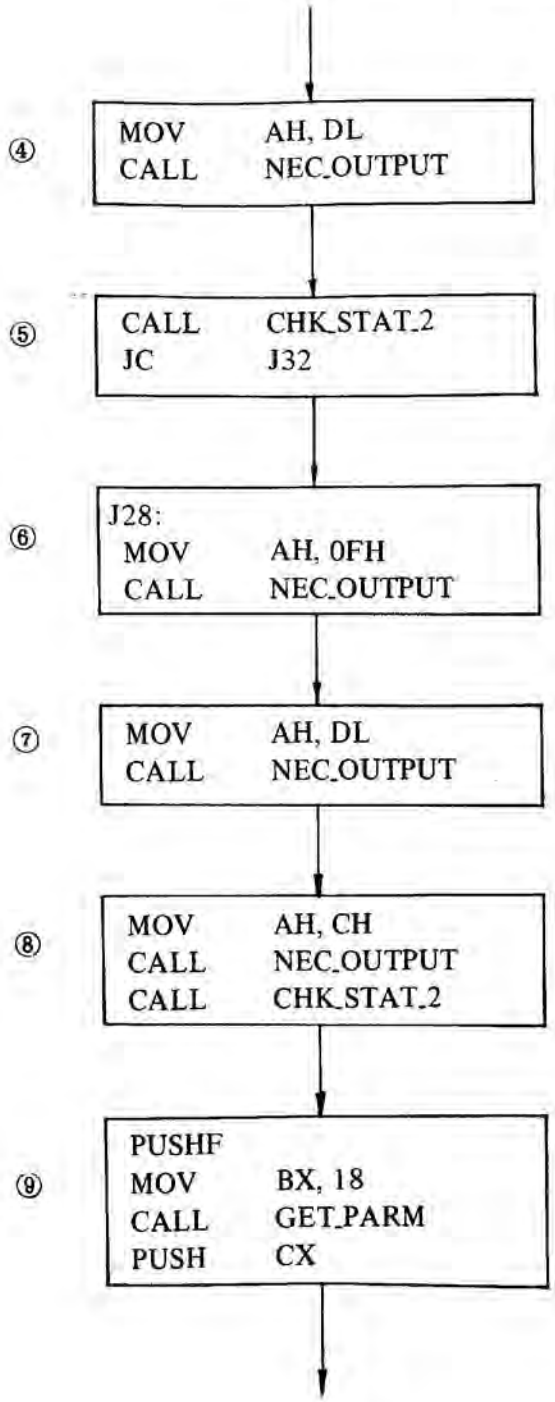
㊲ 進入此步驟是因爲有錯誤產生，呼叫RESULTS副程式的主要目的是爲了使UPD765離開結果狀態，然後返回。

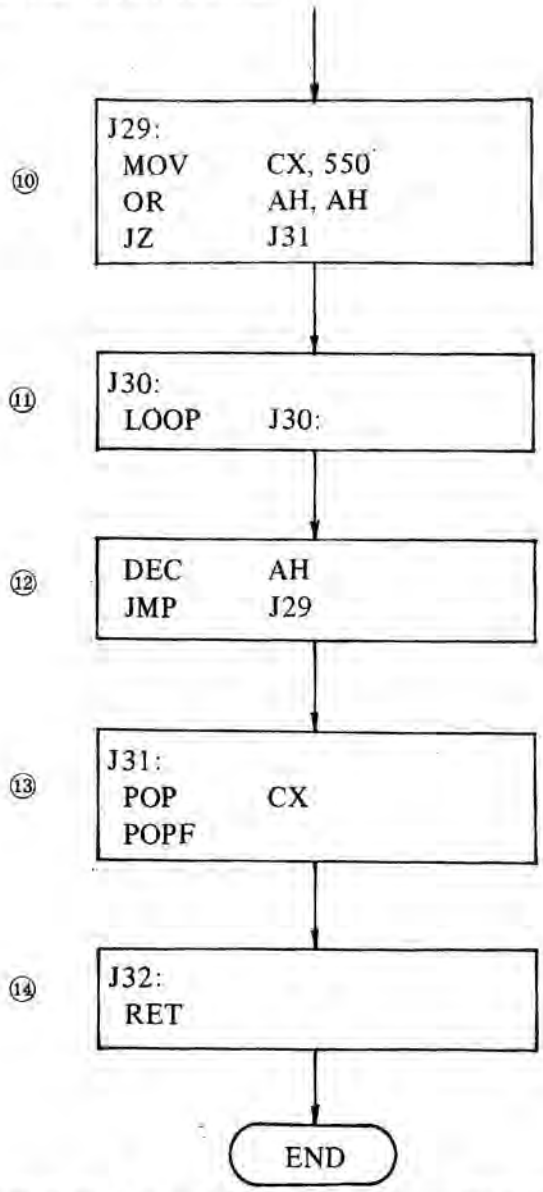
㊳ 進入此步驟是因爲 read/write/format 一切正確，呼叫NUM_TRANS副程式決定讀取/寫入了若干磁區，然後清除AH暫存器內的值，再返回。

RW_OPN副程式呼叫二個我們尚未討論的副程式，SEEK和NUM_TRANS。我們先來看SEEK副程式。

SEEK







① 此步驟的目的是依據所選擇的磁碟機來設定AL 暫存器內的值，執行完此步驟，AL 暫存器內的值和所選擇的磁碟機的關係如下：（所選擇的磁碟機在DL 暫存器內）。

DL 暫存器內的值	所選擇的磁碟機	AL 暫存器內的值
00	drive A	00000001
01	drive B	00000010
02	drive C	00000100
03	drive D	00001000

此結果和RW_OPN 副程式的步驟② 相同。保存CX 暫存器是因爲要避免左旋轉時，破壞CH 暫存器內的初值（保存著磁軌號碼）。

② SEEK_STATUS的位元3 到位元0 的值指出相對磁碟機是否需要將磁頭拉至磁軌0 位置（歸零），若其值爲1，則表示不必歸零。

SEEK_STATUS	相對的磁碟機
bit 3	drive D
bit 2	drive C
bit 1	drive B
bit 0	drive A

這裏將SEEK_STATUS和步驟① 所得到的結果做TEST，以便決定是否要將所選擇的磁碟機的磁頭拉至磁軌0 的位址。若測試結果ZF 旗號爲0，則表示不必歸零，進入步驟⑥；若測試結果ZF 旗號爲1，則表示要歸零，進入下個步驟。

③ 首先將AL 暫存器內的值OR 入SEEK_STATUS內，以便指出所選擇的磁碟機已在此時歸零了，下次使用此磁碟機時，不必再做歸零。在AH 暫存器內放入07H（歸零的第1 個命令位元組），然後呼叫NEC_OUTPUT 副程式將其寫到UPD765 的資料暫存器。

④ 將DL 暫存器內的值（磁碟機號碼）轉到AH 暫存器內（歸零的第2個命令位元組），然後呼叫NEC_OUTPUT 副程式將其寫到UPD765的資料暫存器。

⑤ 在介紹UPD765時，我們說到歸零和SEEK都需要偵測中斷狀態，所以這裏呼叫CHK_STAT_2 副程式，以便了解歸零動作是否正確，若返回時，CF 旗號爲1，則表示歸零動作不正確，進入步驟⑭返回到呼叫SEEK副程式的程式去。若返回時，CF 旗號爲0，則表示歸零動作正確，磁頭已在磁軌0的位置，進入下個步驟。

⑥ 在AH 暫存器內放入0FH（SEEK的第1個命令位元組），然後呼叫NEC_OUTPUT 副程式將其寫到UPD765的資料暫存器。

⑦ 將DL 暫存器內的值（磁碟機號碼）轉到 AH暫存器內（SEEK的第2個命令位元組），然後呼叫NEC_OUTPUT 副程式將其寫到UPD765的資料暫存器。

⑧ 將CH 暫存器內的值（磁軌號碼）轉到 AH 暫存器內（ SEEK的第3個命令位元組），然後呼叫NEC_OUTPUT 副程式，將其寫到UPD765的資料暫存器。再呼叫CHK_STAT_2 副程式，偵測中斷狀態。若SEEK正確，則CF 旗號爲0，若SEEK不正確，則CF 旗號爲1。在這裏並不測試SEEK是否正確，交由呼叫SEEK副程式的程式去測試。

⑨ 保存旗號暫存器以免CF 旗號被破壞，然後在BX 暫存器內放入18，呼叫GET_PARM 副程式取出磁頭定位時間到AH暫存器。再將CX暫存器內的值保存在堆疊中，以免LOOP指令將其破壞。

⑩ 在CX 暫存器內放入550（十進位），以便配合下個步驟的LOOP指令。測試AH 暫存器內的值是否爲0，若爲0（ZF旗號爲1），則表示磁頭已穩定，進入步驟⑬。若不爲0（ZF旗號爲0），進入下個步驟。

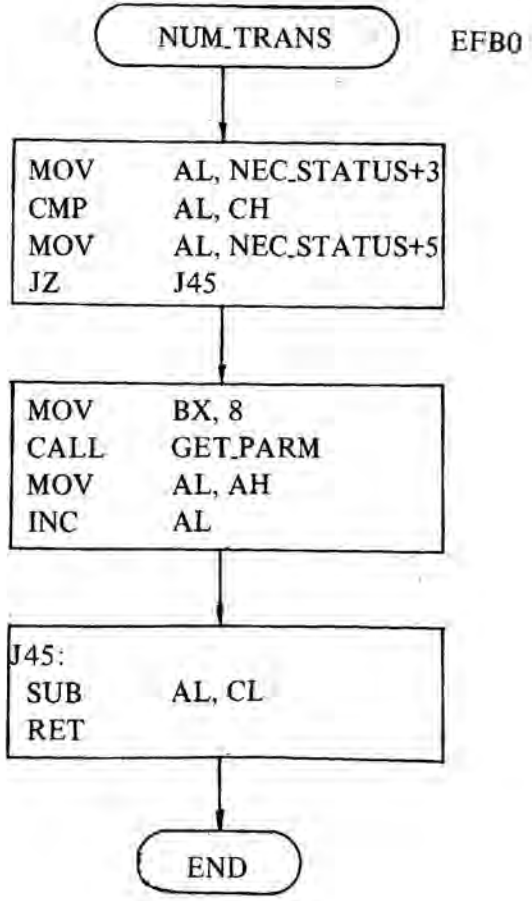
⑪ 這個步驟LOOP 550次，時間爲1 ms。

⑫ 將AH 暫存器內的值減1，然後跳回步驟⑩測試是否還要繼續LOOP。

⑬ 進入此步驟時，磁頭已平穩了，取回CX 暫存器和旗號暫存器的初值後，進入下個步驟。

⑭ 返回到呼叫SEEK副程式的程式去。

NUM_TRANS



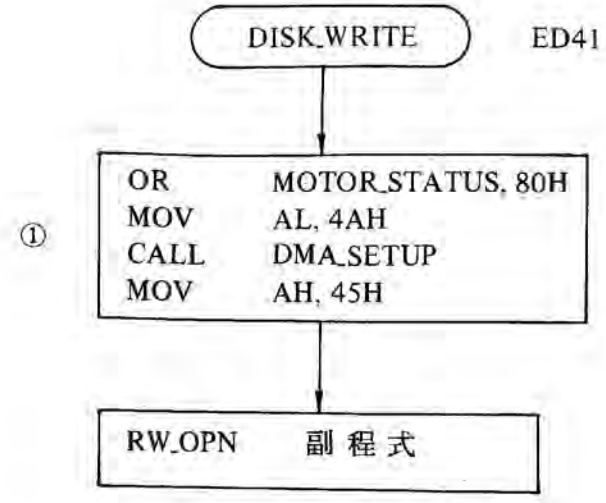
如果讀取／寫入動作以後，結果狀態內的磁軌號碼不相等於命令階段（ command phase ）的磁軌號碼，就表示已讀取／寫入這個磁軌上的最後一個磁區了。

① 從NEC_STATUS 取出結果狀態中的磁軌號碼，然後和命令狀態中的磁軌號碼（在CH 暫存器內）比較，如果相等的話，ZF旗號爲1，若不相等的話，ZF 旗號爲0。然後再從NEC_STATUS 中取出結果狀態的磁區號碼於AL 暫存器內。最後再測 ZF旗號是否爲1，若爲 1，則表示還沒有讀取／寫入到最後一個磁區，跳入步驟③，若ZF 旗號爲0，則表示已讀取／寫入到最後一個磁區了，進入下個步驟。

② 在BX暫存器內放入8，呼叫GET_PARM副程式，取出 end of track（EOT）到AH暫存器內，然後轉到AL暫存器內，並將其值加1。

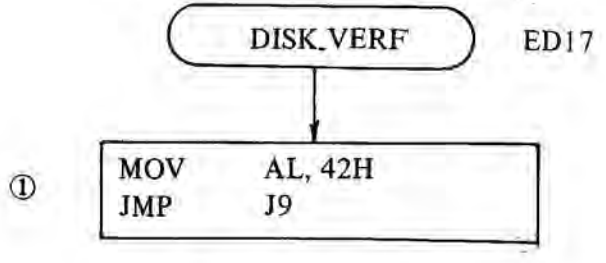
③ 進入此步驟有二種可能：一是磁軌號碼相等時，此種情形，結果狀態中的磁區號碼已指向下一個磁區，所以不必再加1。一是磁軌號碼不相等時，此種情形已經讀取／寫入到最後一個磁區，所以要將 EOT 值加1。將 AL 暫存器內的值減去命令狀態的磁區號碼（在CL 暫存器內）便得到總共讀取／寫入了若干磁區於 AL 暫存器內；最後返回到呼叫此程式的程式去。

(d) (AH)=3, Write sector



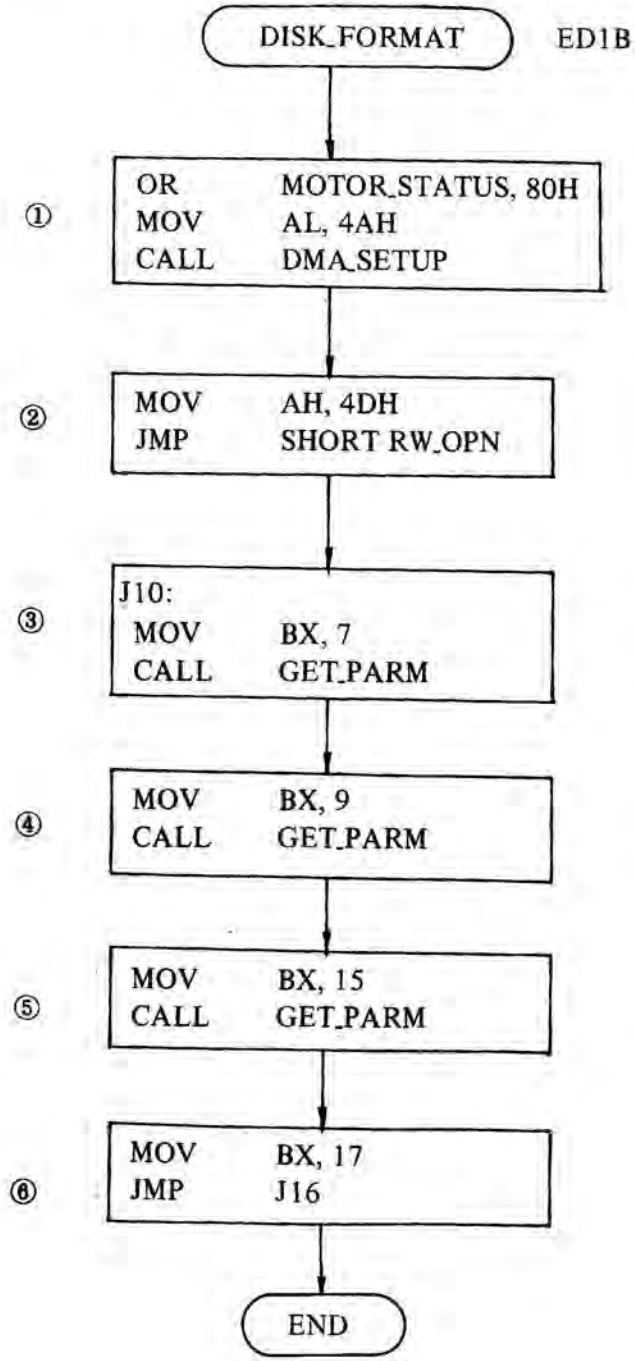
① 首先將MOTOR_STATUS的位元7設定爲1，表示爲寫入動作。然後在AL 暫存器內放入 4AH（此值要寫入 8237 的模式暫存器），呼叫DMA_SETUP副程式，設定模式和起始位址。最後在AH 暫存器內放入 45H（此值是寫入磁區的第一個命令位元組），直接進入RW_OPN 副程式去做寫入磁區的動作

(e) (AH)=4, verify the desired sector



① 在AL 暫存器內放入42H（此值要寫入8237 的模式暫存器），然後跳入讀取磁區副程式的步驟②，去做 verify的動作。

(f) (AH)=5, Format a track



① 設定MOTOR_STATUS的位元7爲1，以便指出爲寫入動作。在AL暫存器內放入4AH（此值要寫到8237的模式暫存器），然後呼叫DMA_SETUP副程式設定模式和起始位址。

② 在AH暫存器內放入4DH（此值爲將一磁軌格式化的第一個命令位元組），然後跳入RW_OPN副程式去做格式化磁軌動作的預先準備。

③ 進入此步驟是因爲RW_OPN副程式的步驟⑰查覺出是格式化一磁軌的動作後，會跳入這裏，以便取出將磁軌格式化的另4個命令位元組。在BX暫存器內放入7，以便取出資料位元組的數目（N），並經由GET_PARM副程式將其寫到UPD765的資料暫存器。

④ 這個步驟取出EOT值，並寫到UPD765的資料暫存器。

⑤ 這個步驟取出Gap Length，並寫到UPD765的資料暫存器。

⑥ 在BX暫存器內放入17，以便於跳入RW_OPN副程式的步驟㉔後取出data pattern，並寫到UPD765的資料暫存器。然後將一磁軌格式化的動作就交給了RW_OPN副程式來執行。

好了，我們已將INT 13H副程式整個討論完畢，相信讀者對磁碟機系統有了粗淺的概念，若讀者想要更詳細地了解磁碟機系統的硬體和軟體，筆者介紹讀者看INTEL公司的microsystem components handbook第二册的8272部分及其應用。

第五章

印表機

我們知道 IBM PC上可以裝上三個印表機(printer)，它們的Base I/O埠值分別是3BCH、378H、278H。若這些印表機存在的話，在開機後自行測試(power on self test)中，已將它們的 I/O 埠值存在printer_base 內了。這三個印表機除了 I/O 埠值不一樣以外，其它部分均相同。我們來看印表機界面卡和印表機之間的控制脚。

① $\overline{\text{STROBE}}$ ：當此脚爲低電位時，印表機才會接受資料。

② $\overline{\text{ACKNLG}}$(acknowledge)：當此脚爲低電位時，就表示印表機已接受了資料。

③ BUSY：若此脚爲高電位，就表示印表機此時正忙，不能接受資料。

④ PE：當此脚爲高電位時，表示印表機上沒有紙了。

⑤ SLCT(select)：當此脚爲高電位時，就表示印表機已被選上。

⑥ Auto Feed XT：當此脚爲低電位時，表示印表機每印完一列以後跳一列，以便下次印到第 3 列。

⑦ $\overline{\text{INIT}}$(initialize)：當此脚爲低電位時，即表示給印表機一個重置(reset) 信號，此低電位至少要 50μs 。

⑧ ERROR ：當此脚爲低電位時，表示印表機部分有錯誤產生。

⑨ $\overline{\text{SLCT IN}}$：當此脚爲低電位時，印表機才可能接受資料。

更詳細的資料，請讀者自行看技術手册印表機部分。

印表機的界面程式（ INT 17H ）以 AH 暫存器內的值可以分成三部分。

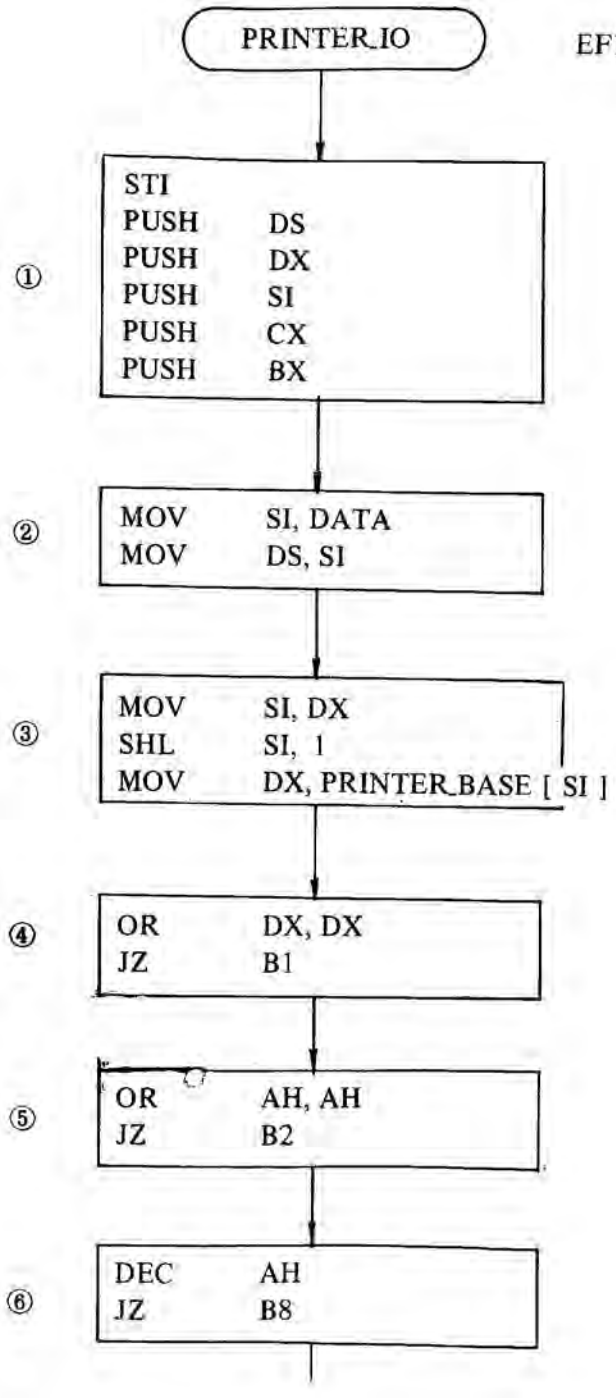
(AH) = 0　印出 AL 暫存器內的值。

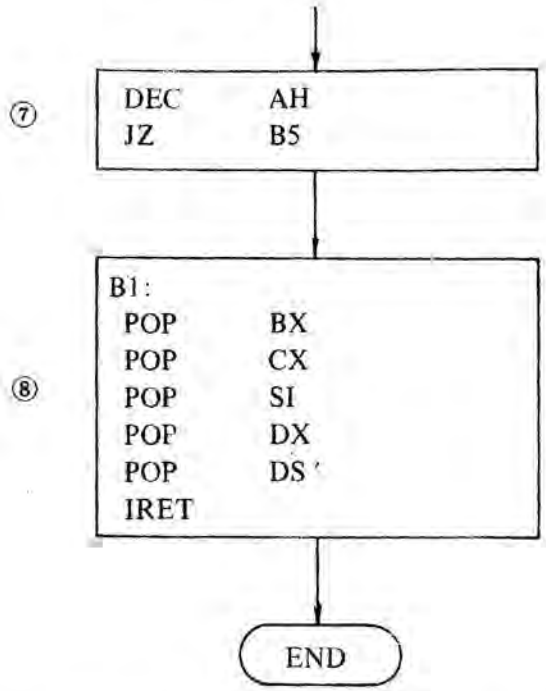
(AH) = 1　起始印表機（ initialize the printer ）。

(AH) = 2　讀取印表機狀態（ read the printer status）到AH暫存器內，讀出後，AH暫存器內各位元的意義如下：

bit 7	BUSY
bit 6	ACKNOWLEDGE
bit 5	out of paper
bit 4	selected
bit 3	I/O error
bit 2 ~ bit 0	unused

進入 INT 17H 的DX 暫存器內的值，指出是那個印表機要被使用，其值的範圍由 0 到 2 。離開 INT 17H 後，只有 AH 暫存器內的值會改變，其它的暫存器不變。





① 首先設定 IF 旗號爲 1，然後陸續將 DS, DX, SI, CX, BX 暫存器內的值保存到堆疊去。

② 將 DS 暫存器內放入 DATA，以便在下個步驟取出印表機的 Base I/O address。

③ 將 DX 暫存器內的值轉到 SI 暫存器內，以便取印表機 Base I/O address時做指標用。將 SI 暫存器左移一個位元是因爲 I/O 位址是二個位元組；接下來的 MOV 指令，將 I/O 位址取出並放到 DX 暫存器內。

④ 此步驟檢查 DX 暫存器內的值是否爲 0，若爲 0（ZF 旗號爲 1），則表示我們所選擇的印表機根本不存在，跳入步驟⑧。若結果不爲 0，進入下個步驟。

⑤ 此步驟測試 AH 暫存器內的值是否爲 0，若爲 0，則表示要印出 AL 暫存器內的值，跳入 B2。若 AH 暫存器內的值不爲 0，進入下個步驟。

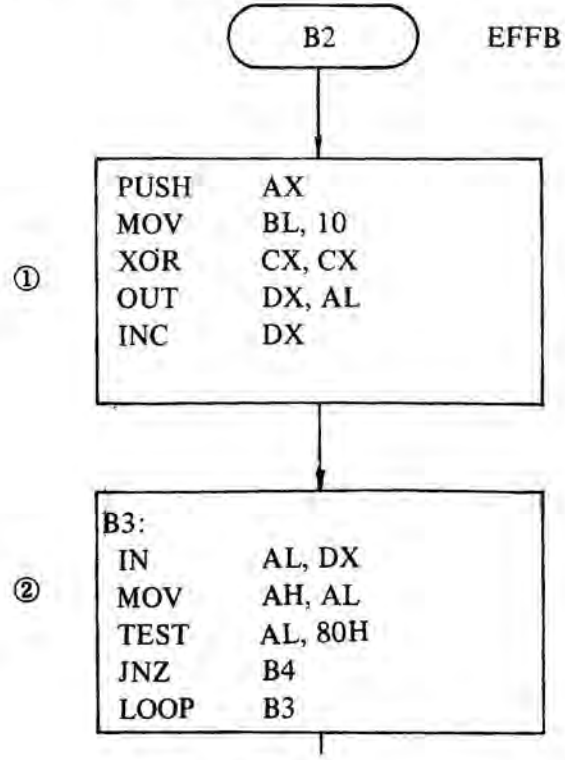
⑥ 此步驟測試 AH 暫存器內的值是否爲 1，若爲 1（ZF 旗號爲 1）則表示要起始印表機，跳入 B8。若結果不爲 1，則進入下個步驟。

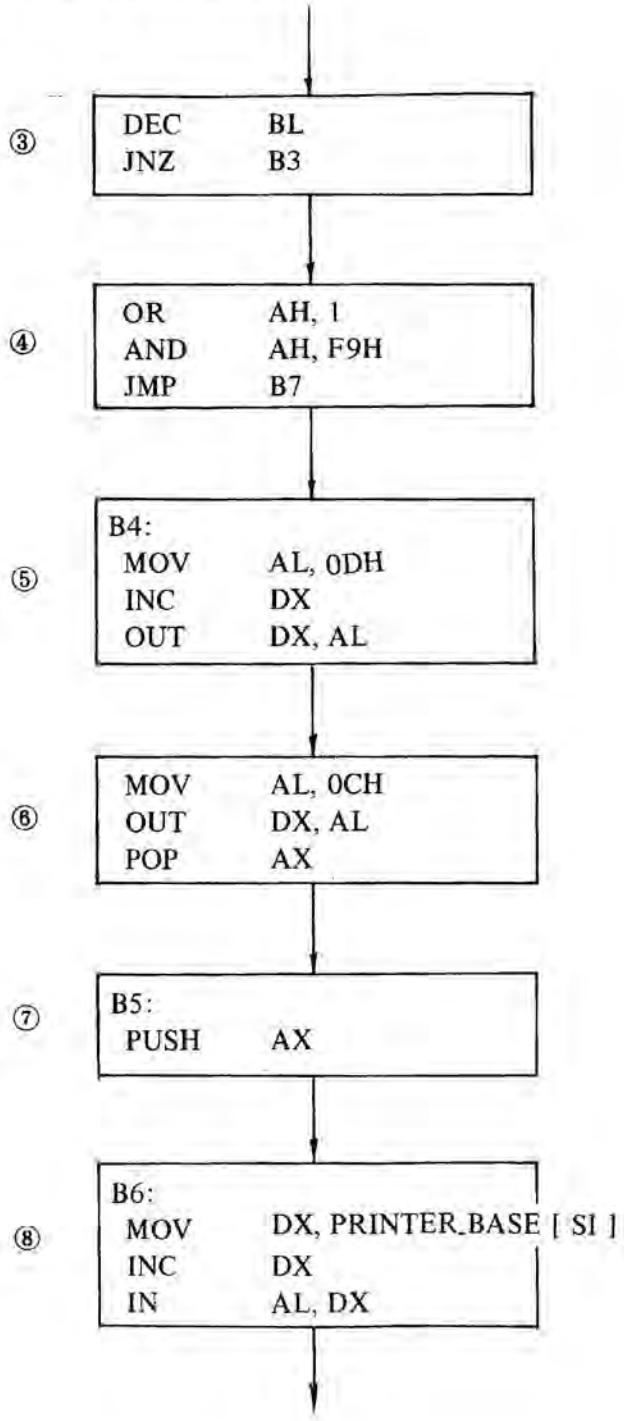
⑦ 此步驟測試 AH 暫存器內的值是否爲 2，若爲 2（ZF 旗號爲 1），則表示要讀取印表機狀態，跳入 B5。若結果不爲 2，則表示 AH 暫存器內的值超過範圍了，進入下個步驟。

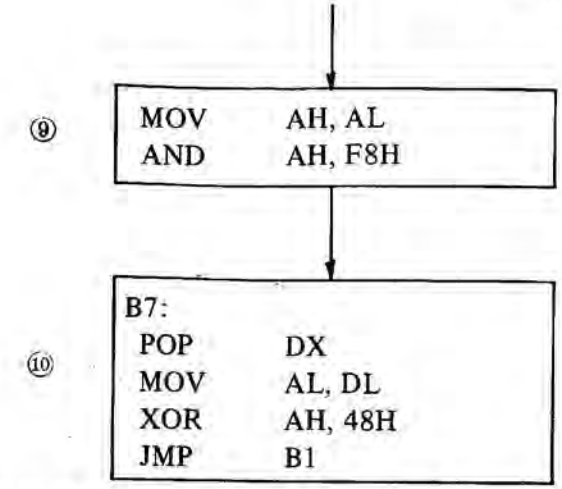
⑧ 陸續取回 BX, CX, SI, DX, DS 等暫存器的初值，然後返回到呼叫 INT 17H的程式去。

接下來，我們以 AH 暫存器內的值來分段討論。

(a) (AH) = 0 印出AL暫存器內的值







① 先將AX 暫存器內的值保存起來。在BL 暫存器內放入10 並清除CX 暫存器內的值，是爲了要配合步驟②和③做等待印表機不忙的LOOP時間。然後將要印的字寫到資料埠（ data port ）去（資料埠是一個 latch ），再將DX暫存器內的值加1，以便指向狀態埠（ status port ）。

② 讀取狀態埠的值，並先保存在 AH 暫存器內。然後測試其是否在忙碌的狀態。若已不在忙碌的狀態（ ZF 旗號爲0 ），則跳入步驟⑤；若仍在忙碌的狀態（ ZF 旗號爲1 ），則得繼續等待（由印表機送出的信號會先經由 LS240 反相後，才進入資料滙流排）。

③ 將BL 暫存器內的值減1，若結果不爲0（ ZF 旗號爲0 ），則仍要等待印表機不在忙碌的狀態，回到步驟②。若結果爲0（ ZF 旗號爲1 ），則表示time out（已LOOP過10 個64K次了），進入下個步驟。

④ 進入此步驟是因爲 time out，此時AH 暫存器內的值仍是從狀態埠讀出的值，將AH 暫存器的位元0 設定爲1，並剔除掉位元2和位元1後跳入步驟⑩。

⑤ 此步驟將 0DH 寫到控制埠（ control port ），如此即給

STROBE 脚一個低電位（輸出到印表機前會有一個反相）

⑥ 將 0CH 寫到控制埠，如此即給 STROBE 脚一個高電位。此時印表機就會接受在資料埠上的資料，然後取回 AX 暫存器的初值。

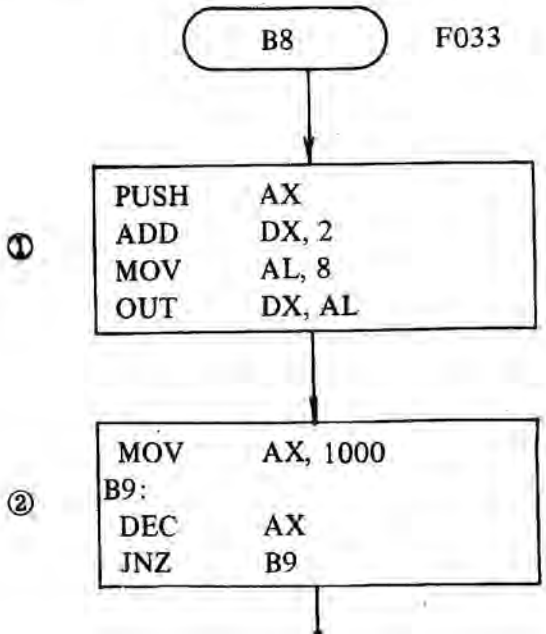
⑦ 將 AX 暫存器內的值保存起來，進入此步驟也可能是要讀取印表機狀態。

⑧ 此步驟讀狀態埠的值。

⑨ 將讀到的值保存在 AH 暫存器內，然後剔除掉位元 2 到位元 0。

⑩ POP DX 指令將步驟⑦中保存的 AX 暫存器內的值取出放到 DX 暫存器內，這樣做是爲了不破壞步驟⑨中得到的 AH 暫存器內的值。MOV AL,DL 指令其實只是取回 AL 暫存器的初值，最後將 AH 暫存器內的值和 48H 互斥後跳至 printer_IO 的步驟⑧返回。

(b) (AH)＝1 起動印表機



① 首先保存 AX 暫存器的初值，然後將 8 寫到控制埠去，如此即給 INIT 脚一個低電位。

② 此步驟只是做等待 50 μs 時間。

③ 將 0CH 寫到控制埠，如此即恢復 INIT 脚爲高電位。此時已給印表機一個重置信號了，跳入 B2 的步驟⑧讀狀態值。

(c) (AH)＝2 讀取印表機狀態

此動作由 B2 的步驟⑦開始執行。

INT 17H 程式相當簡短，若能配合技術手册的印表機部分和線路圖來研究此段程式，會有更大的收穫。

第六章

螢　幕

6845和界面卡介紹

我們知道IBM PC可以接上單色（monochrone）或彩色/繪圖（color/graphics）界面卡來控制螢幕（monitor）顯像。而彩色/繪圖界面卡（color/graphics adapter）又有七種模式，可想而知螢幕顯像的界面程式INT 10H是多麼的龐大與複雜（INT 10H佔了2K 位元組容量），不論單色（單綠色）或彩色/繪圖界面卡都是由MC6845來控制。在討論INT 10H 程式前，我們先介紹6845和界面卡上的I/O埠。

MC6845 控制著顯像時序，整個的時序控制由18個暫存器來做。要將資料寫入18 個暫存器之前，必須將暫存器的位址先寫入位址暫存器（Address Register,AR）；我們現在就陸續介紹這19個暫存器。

① 位址暫存器（Address Register,AR）：此暫存器內的值指出是那一個暫存器將要被寫入（讀取）。

② 水平總計暫存器（Horizantal Total Register(R0)）：此暫存器內的值指出一條水平掃描綫共有多少個字元（character），此值包括RETRACE。

③ 水平顯示暫存器（Horizontal Displayed Register (R1)）:此暫存器內的值指出在螢幕上一列共有若干字元。

④ 水平同步位置暫存器（Horizontal Sync Position Register(R2)）：此暫存器內的值指出水平同步信號的位置。

⑤ 水平同步寬暫存器（Sync Width Register(R3)）：此暫存器內的值指出水平和垂直同步信號的寬度。位元3到位元0爲水平同步，位元7到位元4爲垂直同步。

⑥ 垂直總計暫存器（Vertical Total Register(R4)）：此暫存器內的值指出總共有若干列文字，此值包括RETRACE。

⑦ 垂直總計調整暫存器（Vertical Total Adjust Register(R5)）：此暫存器內的值指出調整掃描綫的值。

⑧ 垂直顯示暫存器（Vertical Displayed Register (R6)）：此暫存器內的值指出螢幕上有若干列文字要被顯像出來。

⑨ 垂直同步位置暫存器（Vertical Sync Position Register (R7)）：此暫存器內的值指出垂直同步信號的位置。

⑩ 間條工作方式暫存器（Interlace and Skew Register (R8)）：這個暫存器內的值指出掃描綫的模式和CUDISP信號與DISPTMG信號的延遲。

⑪ 最大掃描綫位址暫存器（Maximum Raster Address Register(R9)）：暫存器內的值指出每個字元有若干條掃描綫。

⑫ 游標起點暫存器（Cursor Start Raster Regist (R10)）：此暫存器內的值指出游標的起始掃描綫和游標顯示模式。

⑬ 游標終點暫存器（Cursor End Raster Register(R11)）：此暫存器內的值指出游標的結束掃描綫。

⑭ 起始位址暫存器（Start address Register(R12, R13)）：此二個暫存器內的值指出MC6845掃描記憶體的起始位址。

⑮ 游標暫存器（Cursor Register (R14,R15)）：此二個暫存器內的值指出游標的位址。

⑯ 光筆暫存器（Light Pen Register (R16,R17)）：此二個暫存器內的值指出光筆的位置。

筆者希望讀者能將MC6845的資料表先看過一次。

接下來，我們介紹單色和彩色/繪圖界面卡上的I/O埠和其功用。

① 單色界面卡

(a) 6845 位址暫存器 (3B4)

(b) 6845 資料暫存器 (3B5)

(c) CRT 控制埠1 (3B8)

位元5 閃爍致能（enable blink）

位元3 視頻致能（Video enable）

位元0 高解析度模式（high resolution mode）

(d) CRT 狀態埠 (3BA)

位元3 黑白影像（black/white video）

位元0 水平同步信號（horizontal sync signal）

② 彩色/繪圖界面卡

(a) 模式控制暫存器（Mode Control Register）(3D8)

位元5	位元4	位元3	位元2	位元1	位元0	顯像模式
1	0	1	1	0	0	40×25 黑白
1	0	1	0	0	0	40×25 彩色
1	0	1	1	0	1	80×25 黑白
1	0	1	0	0	1	80×25 彩色
×	0	1	1	1	0	320×200 黑白
×	0	1	0	1	0	320×200 彩色
×	1	1	1	1	0	640×200 黑白

位元5	啓動閃爍
位元4	640 × 200 黑白
位元3	起動視頻信號
位元2	選擇黑白
位元1	選擇320 × 200圖形
位元0	80 × 25文數字選擇

(b)顏色選擇暫存器 (color select register)(3D9)

位元0,1,2,3	這4個位元在40×25文數字模式中選擇邊界顏色 (border color) 。在320×200彩色 / 繪圖模式中選擇背景顏色 (background color)
位元4	在文數字模式中，此位元選擇背景顏色。
位元5	在320×200模式中，此位元用來選擇顯像的顏色。

(c)狀態暫存器 (status Register)(3DA)

位元0	顯示起動
位元1	光筆觸發器 (trigger) 設定
位元2	光筆開關
位元3	垂直同步信號

(d)清除光筆鎖定 (clear light-pen latch) (3DB)

(e)設定光筆鎖定 (preset light-pen latch)(3DC)

(f) 6845位址暫存器 (6845 address register) (3D4)

(g) 6845資料暫存器 (6845 data register) (3D5)

這裡只是將單色和彩色 / 繪圖界面卡上的幾個 I / O 埠做一簡單的介紹。詳細的資料，讀者應該由技術手册和綫路圖上獲知。

我們接著看 INT 10H程式中，各個功能和其暫存器的意義。

① (AH)=0 Set Vedio mode (設定顯像模式)

(AL) = 0	40 × 25	BW
(AL) = 1	40 × 25	color
(AL) = 2	80 × 25	BW
(AL) = 3	80 × 25	color
(AL) = 4	320 × 200	color
(AL) = 5	320 × 200	BW
(AL) = 6	640 × 200	BW
(AL) = 7	80 × 25	BW card (monochrone)

② (AH)=1 Set Cursor Type (設定游標形式)

(CH) : Bit 4–0 start line for cursor
(CL) : Bit 4–0 end line for cursor

③ (AH) = 2 Set cursor position (設定游標位置)
(DH, DL) : Row, column (0,0) is upper left
(BH) : Page number (must be 0 for graphics)

④ (AH) = 3 Read cursor position (讀出游標位置)
(BH): Page number
on exit (DH, DL) Row, column of current cursor
(CH, CL) Cursor mode currently set

⑤ (AH) = 4 Read light-pen position (讀光筆位置)
on exit
(AH) = 0 Light-pen switch not down/not trigger
(AH) = 1 Valid light-pen value in register
(DH, DL): Row, column of character
(CH): Raster line
(BX): Pixel column

⑥ (AH) = 5 Select active display page (選擇顯像"頁")
(AL): New page value

⑦ (AH) = 6 Scroll active page up (向上捲動)
(AL): Number of lines, (AL) = 0 Means blank entire window
(CH, CL): Row, column of upper left corner of scroll
(DH, DL): Row, column of lower right corner of scroll
(BH): Attribute to be used on blank line

⑧ (AH) = 7 Scroll active page down (向下捲動)
(AL): Number of lines
(CH, CL): Row, column of upper left corner of scroll
(DH, DL): Row, column of lower right corner of scroll
(BH): Attribute to be used on blank line

⑨ (AH) = 8 Read attribute/character at current cursor position (讀取游標位置內的字元和屬性)
(BH): Display page
on exit
(AL): Character read
(AH): Attribute of character read

⑩ (AH) = 9 Write attribute/character at current cursor position (寫入字元和屬性值到游標位置)

(BH): Display page
(CX): Count of characters to write
(AL): Character to write
(BL): Attribute of character

⑪ (AH) = 10 Write character only at cursor position (寫入字元到游標位置)
(BH): Display page
(CX): Count of character to write
(AL): Character to write

⑫ (AH) = 11 Set color palette (設定顏色)
(BH): Palette color ID being set
(BL): Color value to be used with that color ID

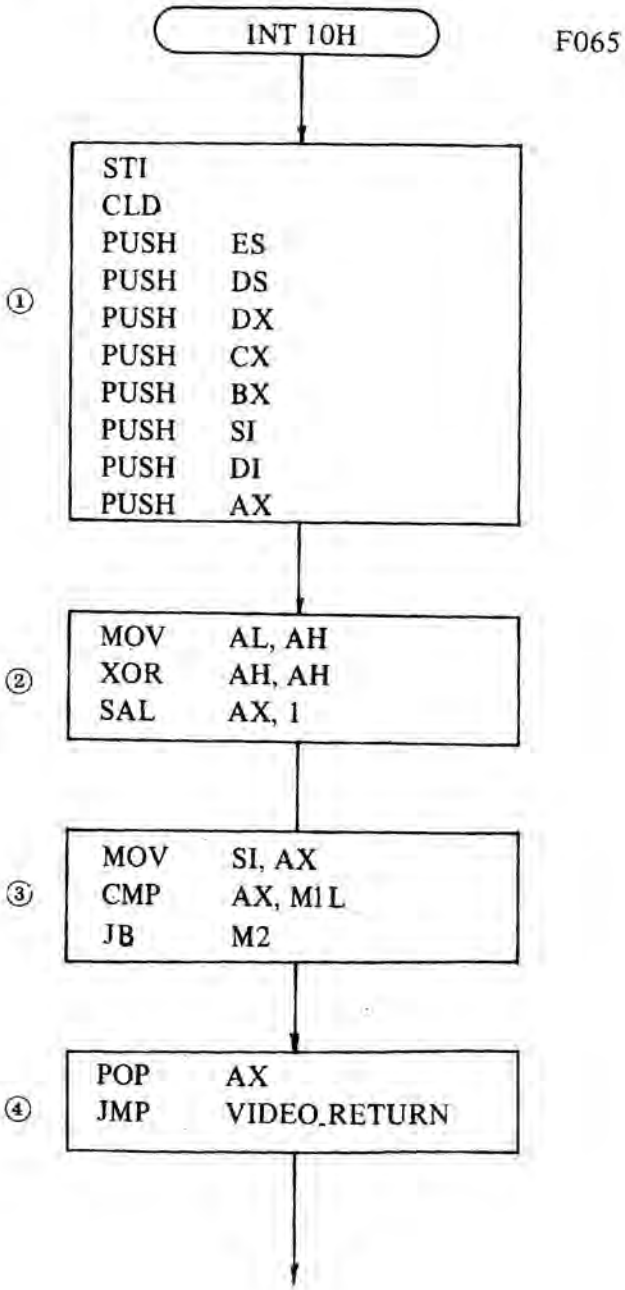
⑬ (AH) = 12 Write dot (句點)
(DX): Row number
(CX): Column number
(AL): Color value

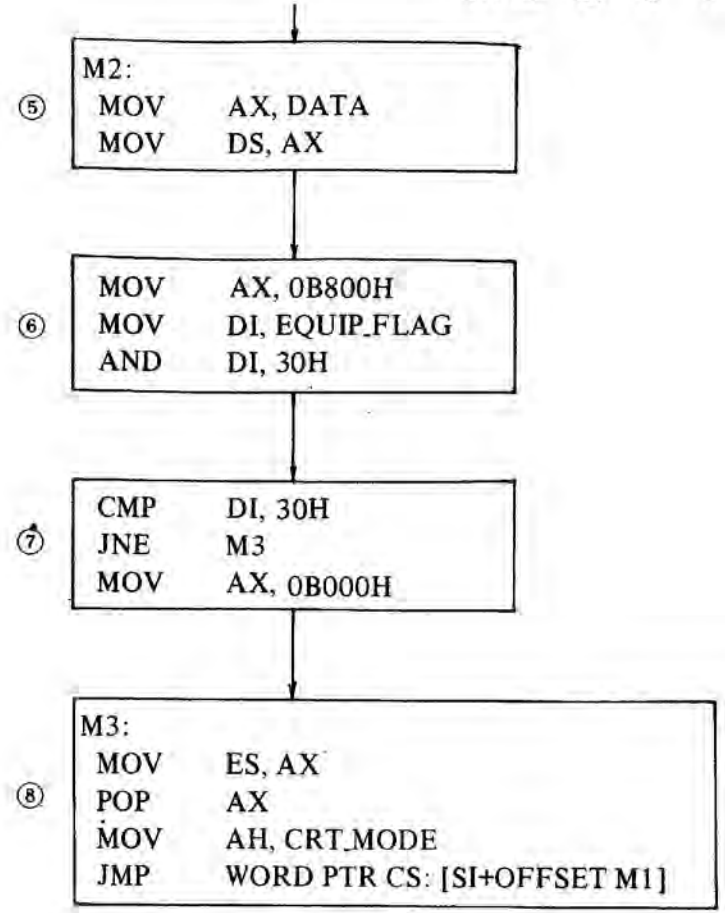
⑭ (AH) = 13 Read dot (讀點)
(DX): Row number
(CX): Column number
(AL): Returns the dot read

⑮ (AH) = 14 Write teletype (電傳輸出)
(AL): Character to write
(BL): Foreground color in graphics mode
(BH): Display page in alphanumeric mode

⑯ (AH) = 15 Current video state (目前顯現狀態)
(AL): Mode currently set
(AH): Number of character columns on screen
(BH): Current active display page

我們現在就進入 INT 10H 程式





① 首先分別將 IF 旗號和 DF 旗號設定爲 1 和 0，然後陸續將 ES，DS，DX，CX，BX，SI，DI，AX 暫存器保存在堆疊中。

② 將 AH 暫存器內的值（指出要執行何種功能）轉到 AL 暫存器內，並將 AH 暫存器清除爲 0。最後將 AX 暫存器內的值左移一位，這是因爲要利用此值找出各個功能的位址，而此位址是一個字組（二個位元組）。

③ 將 AX 暫存器內的值轉入 SI 暫存器內。將 AX 暫存器內的值和 M1L（20H）比較，若小於 M1L 則表示進入 INT 10H

時的AH暫存器的值是在範圍內，進入步驟⑤。若AX暫存器內的值大於或等於M1L，則表示進入INT10H時，AH暫存器內的值超過了範圍，進入下個步驟。

④ 取回AX暫存器的初值，然後跳到VIDEO_RETURN返回到呼叫INT 10H的程式去。

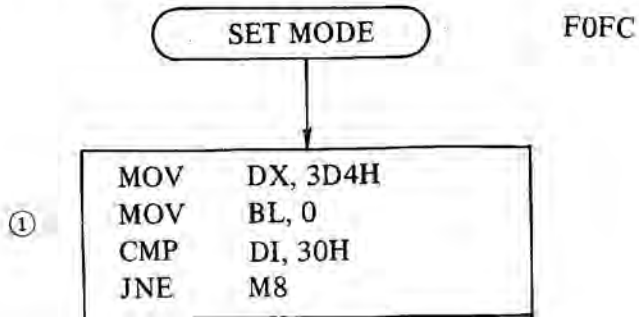
⑤ 此步驟設定DS暫存器爲DATA(0040H)。

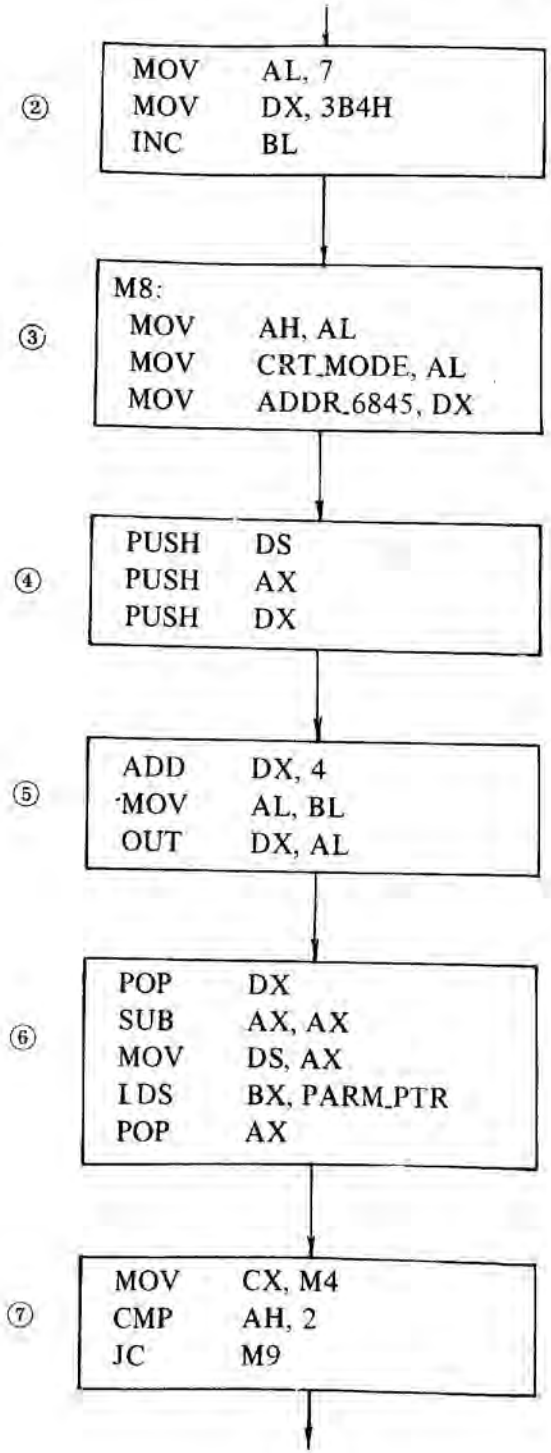
⑥ 首先在AX暫存器內放入B800H(先假設爲彩色/繪圖)，然後取出EQUIP_FLAG的位元5和位元4。

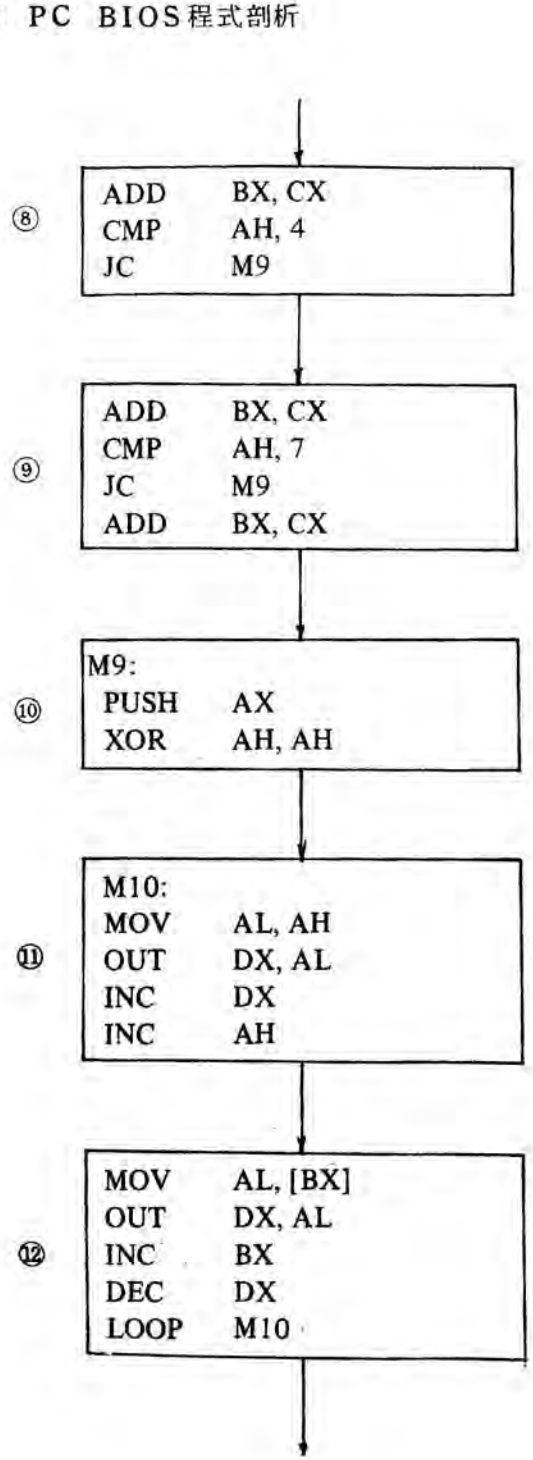
⑦ 比較EQUIP_FLAG的位元5和位元4是否同時爲1，若不同時爲1(ZF旗號爲0)，則表示爲彩色/繪圖卡，直接進入步驟⑧，若結果爲ZF旗號爲1，則表示爲單色卡，在AX暫存器內放入B000H，然後進入步驟⑧。

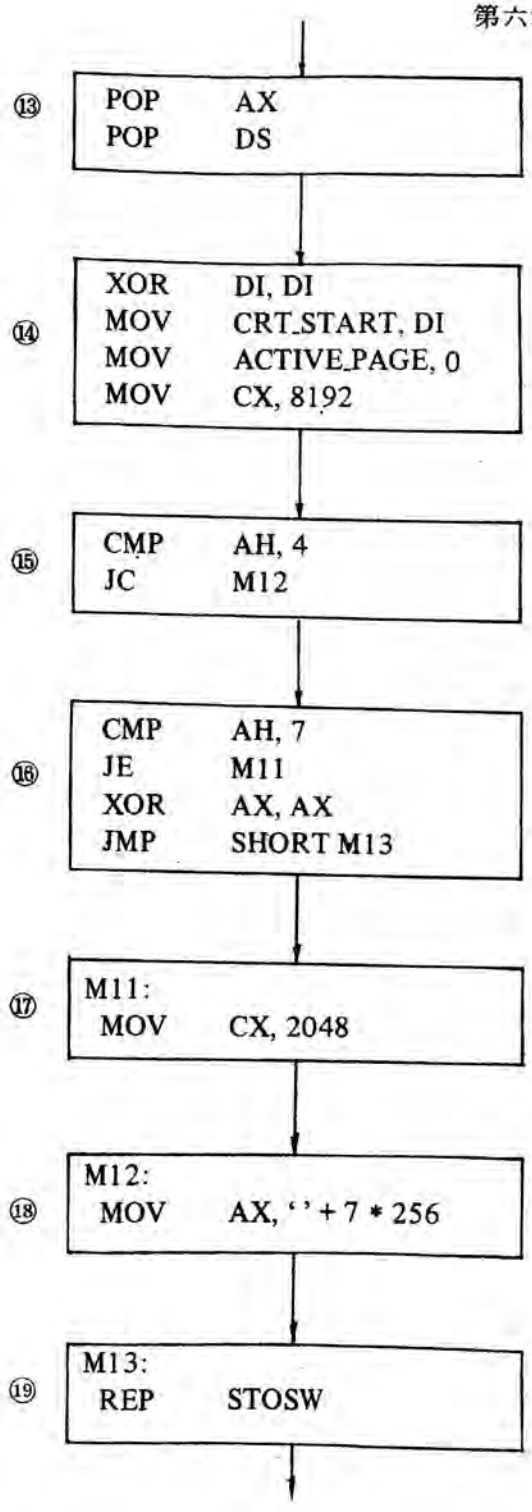
⑧ 將AX暫存器內的值轉到ES暫存器內(若爲單色，ES爲B000H，若爲彩色/繪圖，ES爲B800H)。然後取回AX暫存器的初值，並將記憶體位址CRT_MODE(在DATA段落內)內的值放到AH暫存器內。最後跳到進入INT 10H程式時，AH暫存器所指的功能的副程式去。從OFFSET M1起的16個字組存放著處理各個功能的副程式的起始位址。我們現在就來討論INT 10H程式中的各個功能。

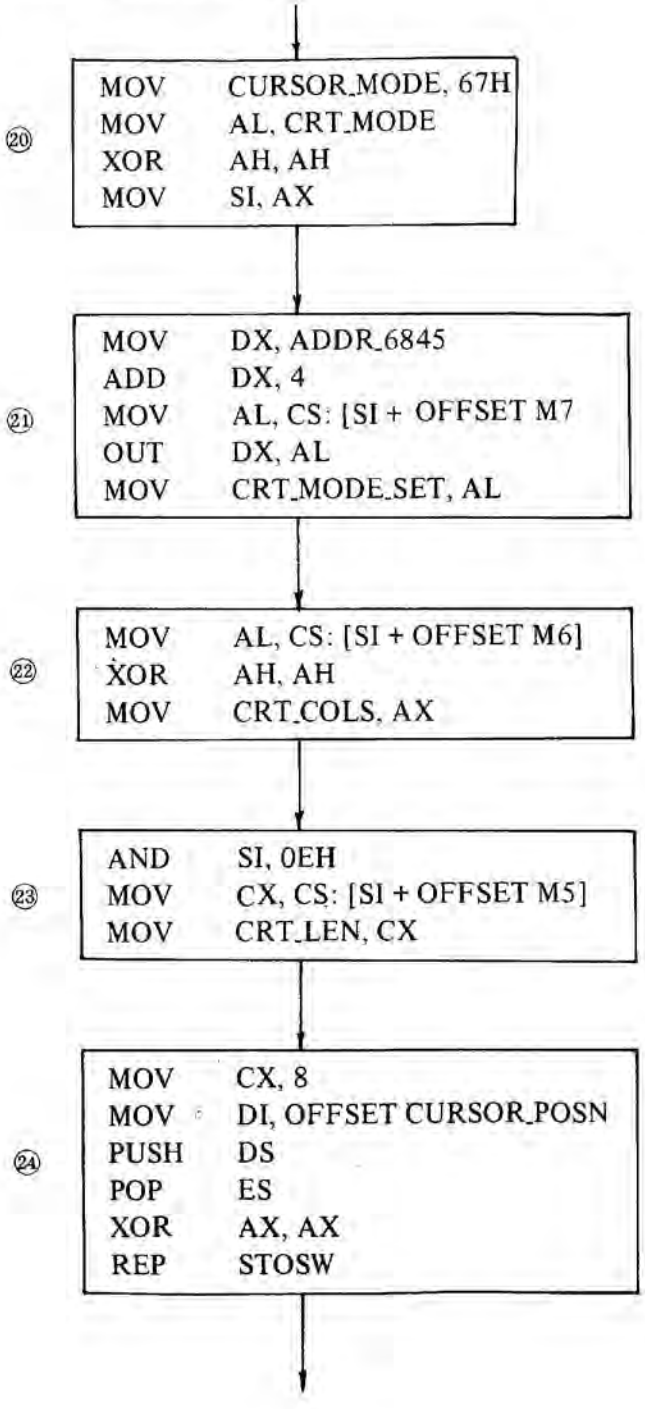
(a) (AH) = 0　設定模式

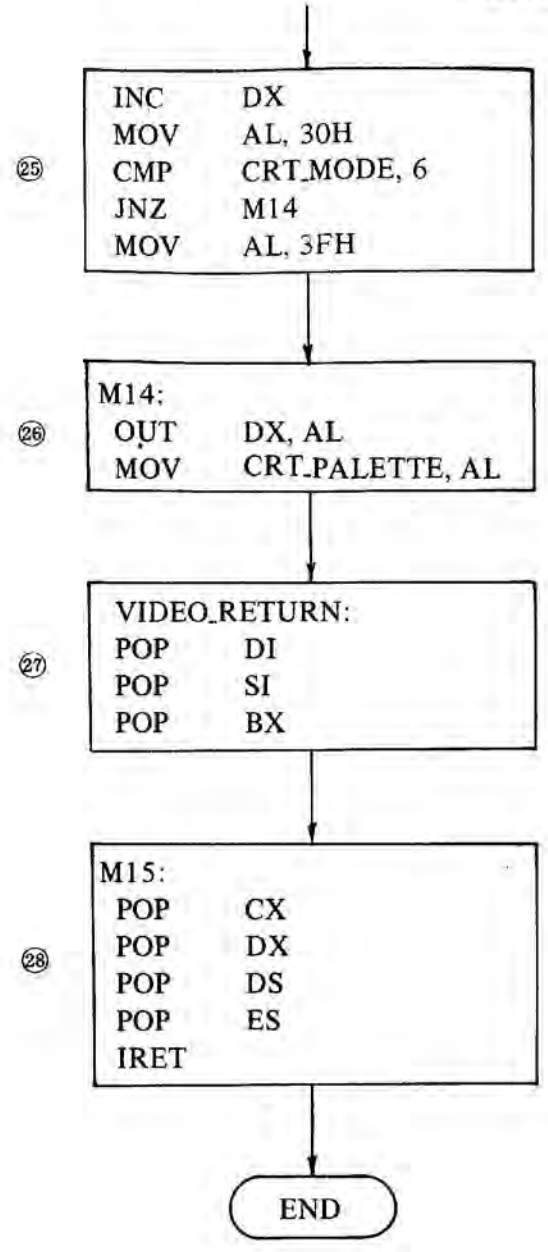





⑧
ADD BX, CX
CMP AH, 4
JC M9
⑨
ADD BX, CX
CMP AH, 7
JC M9
ADD BX, CX
⑩
M9:
PUSH AX
XOR AH, AH
⑪
M10:
MOV AL, AH
OUT DX, AL
INC DX
INC AH
⑫
MOV AL, [BX]
OUT DX, AL
INC BX
DEC DX
LOOP M10


⑬
POP AX
POP DS
⑭
XOR DI, DI
MOV CRT_START, DI
MOV ACTIVE_PAGE, 0
MOV CX, 8192
⑮
CMP AH, 4
JC M12
⑯
CMP AH, 7
JE M11
XOR AX, AX
JMP SHORT M13
⑰
M11:
MOV CX, 2048
⑱
M12:
MOV AX, ' ' + 7 * 256
⑲
M13:
REP STOSW


⑳
MOV CURSOR_MODE, 67H
MOV AL, CRT_MODE
XOR AH, AH
MOV SI, AX
㉑
MOV DX, ADDR_6845
ADD DX, 4
MOV AL, CS: [SI + OFFSET M7
OUT DX, AL
MOV CRT_MODE_SET, AL
㉒
MOV AL, CS: [SI + OFFSET M6]
XOR AH, AH
MOV CRT_COLS, AX
㉓
AND SI, 0EH
MOV CX, CS: [SI + OFFSET M5]
MOV CRT_LEN, CX
㉔
MOV CX, 8
MOV DI, OFFSET CURSOR_POSN
PUSH DS
POP ES
XOR AX, AX
REP STOSW


㉕
INC DX
MOV AL, 30H
CMP CRT_MODE, 6
JNZ M14
MOV AL, 3FH
㉖
M14:
OUT DX, AL
MOV CRT_PALETTE, AL
㉗
VIDEO_RETURN:
POP DI
POP SI
POP BX
㉘
M15:
POP CX
POP DX
POP DS
POP ES
IRET
END

① 首先假設是彩色／繪圖，在DX內放入3D4H，在BL內放入0，然後再來測試是何種界面卡（進入此副程式時，DI暫存器內的值是取自EQUIP-FLAG）。若DI內的值等於30H（ZF 旗號爲1），則表示爲單色，進入下個步驟，設定單色的值。若爲彩色／繪圖（ZF 旗號爲0），跳入步驟③中。

② 進入此步驟是因爲單色在AL暫存器內放入7，表示單色在DX暫存器內放入3B4H，將BL內的值加1使其成爲1，然後進入下個步驟。

③ 將AL暫存器內的值轉到AH暫存器內和CRT_MODE中，並將DX暫存器內的值存在ADDR_6845內，如此CRT_MODE和ADDR_6845就因不同的界面卡和模式（MODE）而有其相對應的值。

④ 陸續將DS，AX，DX暫存器保存在堆疊中，這是因以下的步驟會陸續用到這些暫存器，但我們不希望它們的初值被破壞掉。

⑤ 將DX暫存器內的值加4，以便指向控制埠（單色爲3B8H，彩色／繪圖爲3D4H），然後將BL暫存器內的值轉入AL暫存器內（單色爲01，彩色／繪圖爲00），並將此值寫到控制埠去，如此即關閉視頻信號。

⑥ 首先取回DX暫存器的初值，接著將DS暫存器設定爲0000H，這是爲了要取出PARM_PTR內的值（PARM_PTR在分段值爲0000H的記憶體的範圍內）。在POST中，我們已經設定PARM_PTR內的值爲F000：F0A4，所以LDS指令設定DS暫存器爲F000H，BX暫存器爲F0A4H。最後取回AX暫存器的初值。從記憶體F000：F0A4起的64個位元組儲存了要寫入6845的

暫存器的值；這個表格共有4種值，每種值佔了16個位元組。第一種值是設定40×25模式，第2種值是設定80×25，第3種值是設定繪圖，第4種值是設定單色。執行完此步驟後，BX暫存器內的值指向第一種值。

⑦ 在CX暫存器內放入M4（10H）是因爲每一種值佔了16個位元組。在步驟③我們已將模式值（在AL暫存器內）轉入AH暫存器內了。這裡將其和2比較是爲了決定是否爲40×25模式，若是（CF 旗號爲1），直接跳入步驟⑩，此時BX暫存器內的值指向第1種值。若不是40×25模式，則進入下個步驟。

⑧ 首先將BX暫存器內的值加上CX 暫存器內的值，以便指向第2種值。然後將模式值和4比較，若CF旗號爲1，則表示爲80×25模式，直接進入步驟⑩。若不是80×25模式，進入下個步驟。

⑨ 再將BX暫存器內的值加上CX暫存器內的值，以便指向第3種值。將模式值和7比較，若是繪圖模式CF旗號爲1，則跳入步驟⑩，若不是繪圖模式，則必爲單色，再將BX暫存器內的值加上CX暫存器內的值，以便指向第4種值。

⑩ 再將AX暫存器內的值保存起來，並將AH 暫存器內的值清除爲0；這是爲了在下二個步驟中，將資料寫入6845的資料暫存器時的起始值。

⑪ 這個步驟和下個步驟是將資料寫入6845的資料暫存器。在將資料寫入資料暫存器前，要先將資料暫存器的位址寫入位址暫存器內。進入此步驟的DX暫存器指向著6845的位址暫存器，將DX加1，則指向6845的資料暫存器，同時，BX 指著要寫入資料暫存器的資料的起始位址，

CX暫存器指出要寫入多少個資料。在此步驟中，AH暫存器指出資料暫存器的位址，DX指向6845的位址暫存器。

⑫ 在此步驟中，BX暫存器指著要寫入資料暫存器的資料的位址。INC BX是爲了指向下一個資料，DEC DX是爲了指向6845的位址暫存器。LOOP指令配合CX內的值，共填入16個資料到16個資料暫存器內。

⑬ 取回AX暫存器和DS暫存器的初值。

⑭ 將DI暫存器清除爲0，並放到CRT_START內，這表示6845會從記憶體（顯像緩衝區）的最起點開始掃描。在ACTIVE_PAGE放入0，表示現在是使用顯像緩衝區的頁0（單色和繪圖只有一個頁（page），40×25有8個頁，80×25有4個頁），在CX暫存器內放入8192（8K）是先假設爲彩色/繪圖卡，以便在步驟⑲中，清除畫面。

⑮ 從此步驟到步驟⑲是爲了清除畫面，但因界面卡的不同而有不同大小的顯像緩衝區，或者是模式不同而有不同的清除方法。在這裡，將模式和4比較，若CF旗號爲1，則表示爲40×25或80×25模式，進入步驟⑱，若CF旗號爲0，進入下個步驟。

⑯ 將模式和7比較，若相等（ZF旗號爲1），則表示爲單色，進入下個步驟設定CX暫存器的值（單色的顯像緩衝區爲4K），若不相等，則必爲繪圖模式，先清除AX暫存器內的值，再跳入步驟⑲。

⑰ 進入此步驟是因爲單色界面卡，設定CX暫存器內的值爲2048(2K)。

⑱ 在AX暫存器內放入0720H(註：‘ ’表示空白，其值爲

20H，7×256表示將7H左移8個位元，故得到0720H）。將0720填入顯像緩衝區時，20H位於低位元組爲字元（character），07H位於高位元組爲屬性。

⑲ 這個步驟將空白（或黑色）填到顯像緩衝區去。在單色中，CX值爲2048，AX暫存器爲0720H，這是因爲單色的顯像緩衝區爲4K。在80×25或40×25模式中，CX值爲8192，AX爲0720H，在繪圖中，CX值爲8192，AX爲0000H(黑色)。值得注意的是CX的值，因爲一次放一個字組（二個位元組），而REP指令一次只將CX值減1，所以CX值均爲顯像緩衝區的一半。

⑳ 在CURSOR_MODE中放入67H,以便指出游標的顯示模式。爲了在下面三個步驟分別取出CRT_MODE_SET，CRT_COLS和CRT_LEN值，所以這裡根據模式值來設定SI暫存器，以便做爲指標，在表格中取出相對的值。

㉑ 前面二個步驟設定DX值爲3B8H或3D8H，然後根據SI的值，到M7表格中取出模式值，並寫到模式暫存器。M7表格中存放著要寫到模式暫存器的值，依不同模式而不同。最後也將此值保存在CRT_MODE_SET中。

㉒ M6表格中存放著各個模式的顯像行數。此步驟取出行（column）值，並保存在CRT_COLS中。

㉓ M5表格中放著各個模式的頁的大小值（40×25爲2048，80×25爲4096，繪圖爲16384），因爲每個值都是字組（二個位元組），所以要先將SI暫存器的位元0清除爲0，再取出其值放到CRT_LEN中。

㉔ 在分段值爲0040H，間距值爲0050起的8個位元組記憶體是爲了儲放游標的位址（因爲最多有8個頁，所以需要8個位元組）。此步驟是將此8個位元組都填入00H，

以表示游標是在螢幕的左上角。因爲STOSW是使用ES和DI暫存器，所以必須設定好ES和DI的值。

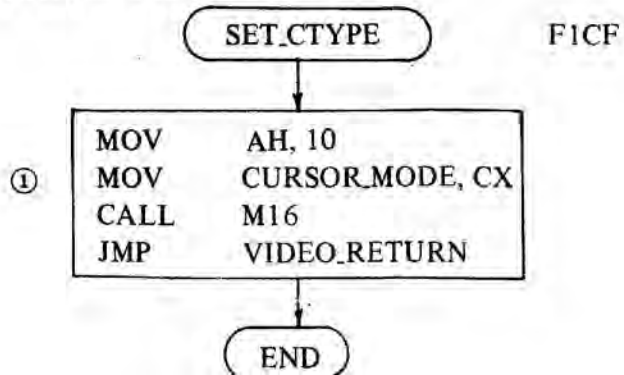
㉕ 此步驟和下個步驟是爲了設定顏色選擇暫存器（color-select register）（此二個步驟對單色沒有意義，因爲3B9H這個I/O 埠不具任何功能）。這步驟首先判斷是否爲高解析度模式，即640×200(CRT_MODE值爲6），若是（ZF旗號爲1），則這個顏色選擇暫存器對它來說也沒有意義，因爲640×200模式中不能選擇顏色，只能決定某點亮或不亮，所以在AL暫存器內放入3FH，以便將顏色選擇暫存器都填入1（顏色選擇暫存器只有6個位元），若不爲640×200模式（ZF旗號爲0），則在AL暫存器內放入30H，以便在下個步驟中寫到顏色選擇暫存器。

㉖ 將AL暫存器內的值寫到顏色選擇暫存器，並保存到CRT_PALETTE去。請讀者自行參考顏色選擇暫存器。

㉗ 這個步驟是VIDEO_RETURN的進入點，陸續取出DI，SI和BX暫存器的初值。

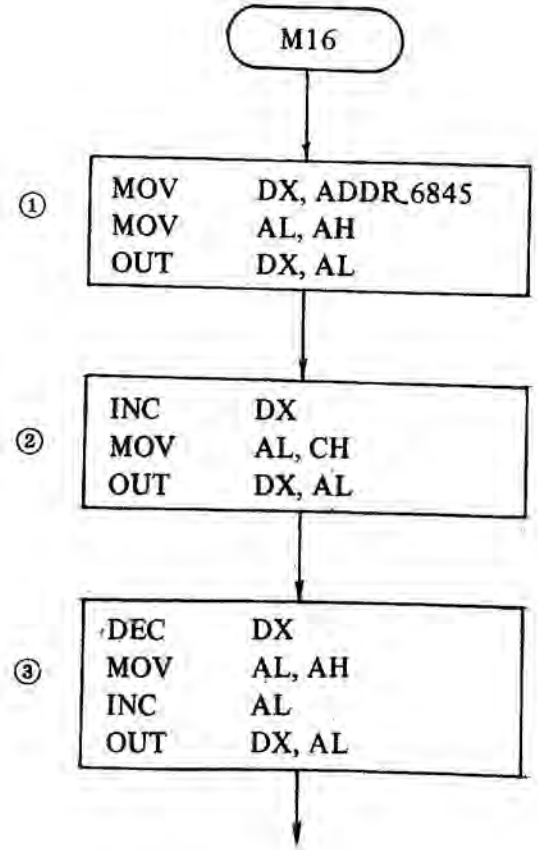
㉘ 有的處理各個功能的副程式，要由此步驟返回。陸續取回CX，DX，DS，ES暫存器的初值後返回到呼叫INT 10H的程式去。

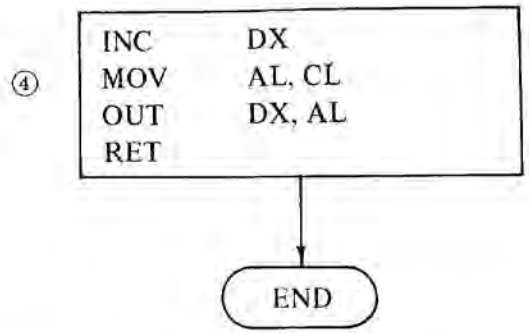
(b)（AH）＝1設定游標形式



① 這個功能只需一個步驟4個指令。在AH暫存器內放入10，這是因爲6845內第10個資料暫存器決定游標（cursor）的起始掃描綫，第11個資料暫存器決定游標的結束掃描綫（Raster）。進入此程式的CX暫存器內的值指出游標的起始和結束的掃描綫（CH爲起始，CL爲結束），先將此值保存在CURSOR_MODE中。在呼叫M16將CH和CL內的值寫到6845的資料暫存器後，跳入VIDEO_RETURN返回呼叫INT 10H的程式去。

我們來看M16是如何將CX內的值寫到6845的資料暫存器去。





① 我們已經知道要將資料寫到資料暫存器之前，必需先將此資料暫存器的位址寫到位址暫存器內。這裡先將ADDR_6845內的值放到DX內，此值爲3B4H（單色）或3D4H（彩色／繪圖），這二個值都指向6845的位址暫存器，將資料暫存器的位址（在AH暫存器內）放到AL內，並寫入6845的位址暫存器。

② INC DX將I/O埠值指向6845的資料暫存器。將要寫入該資料暫存器的資料（在CH暫存器內）放到AL暫存器內，並寫到資料暫存器內。

③ DEC DX將I/O埠值指向6845的位址暫存器，接下來的三個指令，將進入此程式時AH暫存器內的值加1，並寫到6845的位址暫存器。

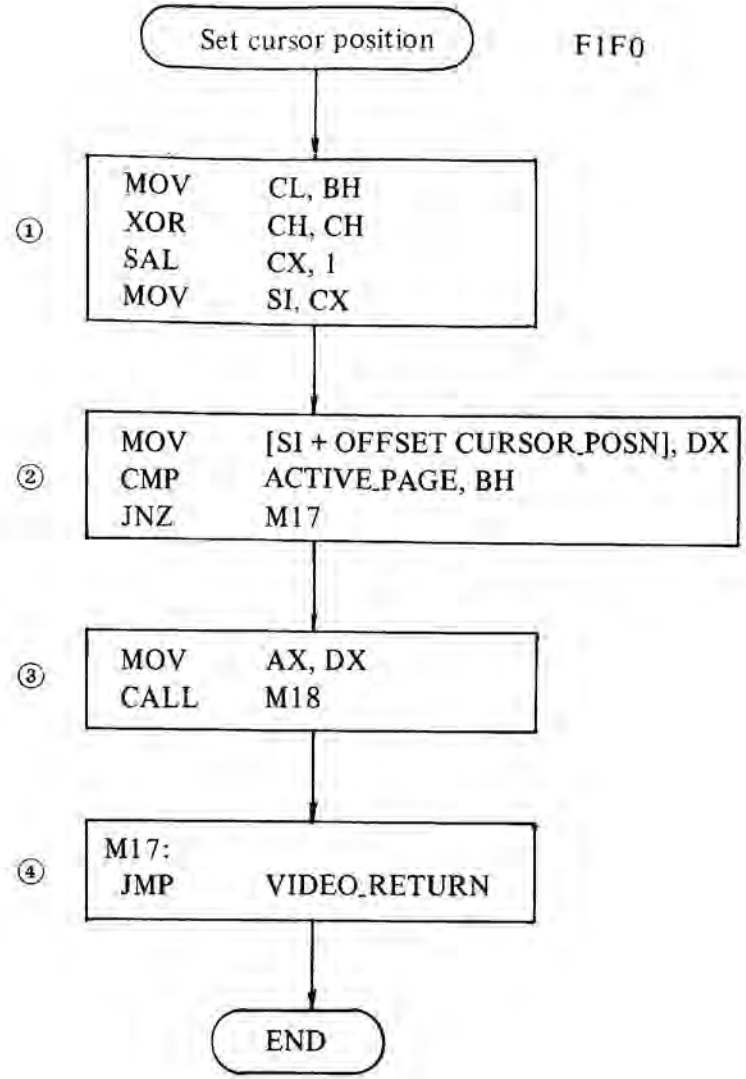
④ INC DX將I/O埠值指向6845的資料暫存器，將要寫入該資料暫存器的資料（在CL暫存器內）放到AL暫存器內，並寫到資料暫存器內。

(c)（AH）=2 設定游標位置

Input：

(DX) = Row, Column of New cursor

(BH) = display page of cursor



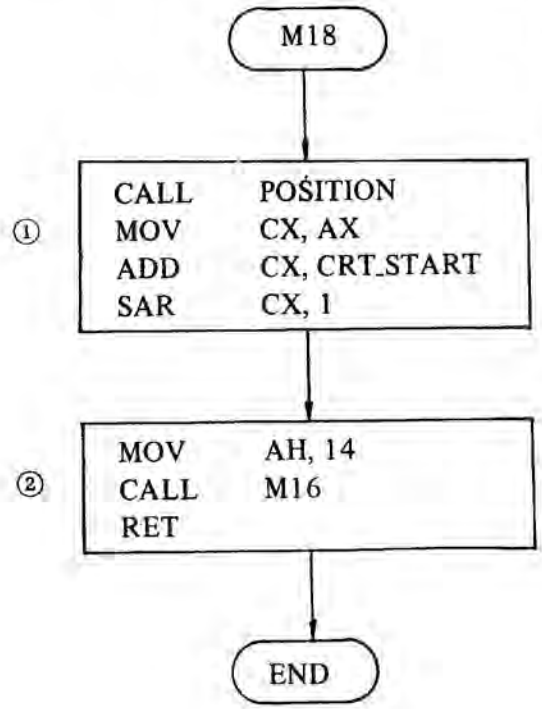
① 將顯像頁（display page）（在BH暫存器內）放到CL暫存器內，並清除CH暫存器，再將CX暫存器左移一個位元，最後將CX暫存器內的值放到SI內。這樣做的原因是要以SI暫存器內的值做爲指標，將新的游標位置（在DX暫存器內）放入游標位置表格。

② 將新的游標位址放到游標位置表格內相對於顯像頁的位址。然後將BH暫存器內的值和ACTIVE_PAGE值比較，若不相等（ZF旗號爲0），則跳入步驟④。若相等（ZF旗號爲1），則進入下個步驟。

③ 將新的游標位置放到AX暫存器內，然後呼叫M18，將游標移至指定的位置。

④ 跳至VIDEO_RETURN返回到呼叫INT 10H的程式去。

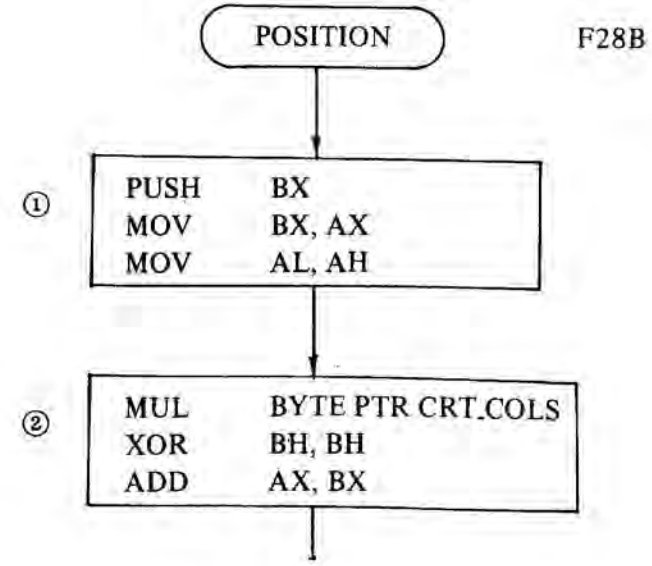
我們繼續看M18如何將游標移至指定的位置。

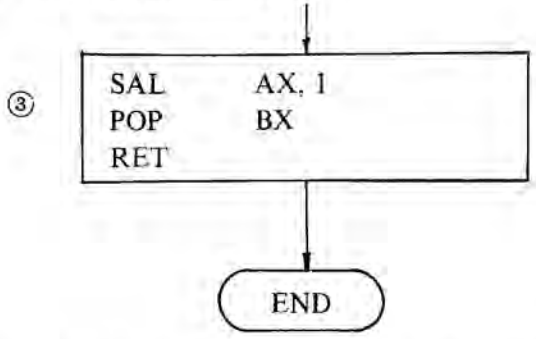


① 在進入M18時，AX指出游標所在的位置（列，行），所以這裡要呼叫POSITION副程式將（列，行）轉換成實際的記憶體間距值（此間距值相對於CRT_START）。從POSITION副程式返回後的AX暫存器保存著記憶體間距值（包含字元和屬性），將AX暫存器內的值轉到CX暫存器內，並將CX暫存器內的值加上CRT_START，最後再將CX暫存器內的值右移一位，因爲游標位置是以字元（Character）爲單位。此最後的CX暫存器內的值將會在下個步驟寫到6845內的游標位址暫存器（cursor address register）內。

② 在AH暫存器內放入14，這是因爲6845的游標位址暫存器的位置爲14，然後呼叫M16，將新的游標位置寫到6845的游標位址暫存器。

我們來看POSITION副程式如何將（列，行）轉換成記憶體間距值。





① 將 BX 暫存器內的值保存起來，以免被破壞。將 AX 暫存器內的值放到 BX 暫存器內。最後將列（ROW）值（在 AH 暫存器內）轉到 AL 暫存器內。

② 將列值乘以每列的字元數，結果在 AX 暫存器內。清除 BH 暫存器，只留下行值於 BL 暫存器內。最後將 AX 暫存器內的值加上 BX 暫存器內的值，此時 AX 暫存器內的值就包含了字元的間距值。

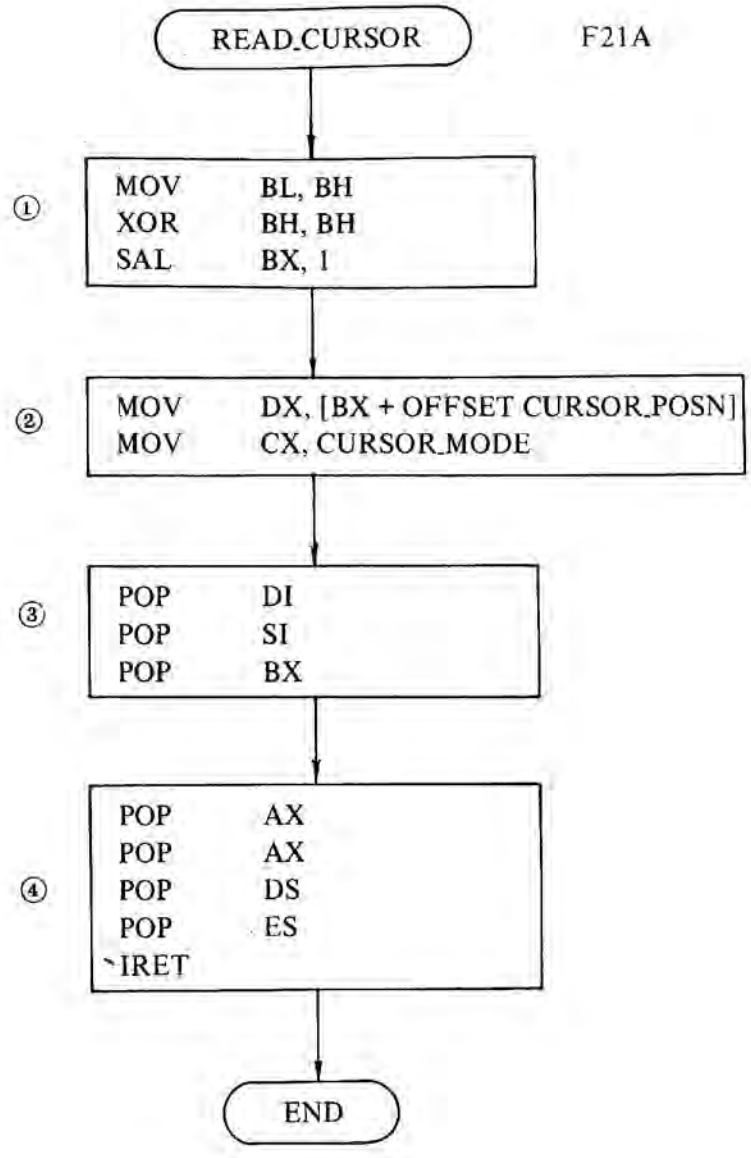
③ 將 AX 暫存器內的值左移一個位元（即乘以 2 之意），此時 AX 內即是相對於 CRT_START 的字元 / 屬性的間距值。取出 BX 暫存器的初值後，回到呼叫 POSITION 的程式去。

(d) (AH) = 3 讀出游標位置

Input : Page of cursor

Output : (DX) = Row, Column of the current cursor position

(CX) = current cursor mode



① 將顯像頁的值（在BH暫存器內）放到BL暫存器內，並將BH暫存器清除爲0，再將BX暫存器左移一個位元，這是因爲在下個步驟中，要以BX暫存器做指標取出游標位置值，而每個游標位置都佔了二個位元組。

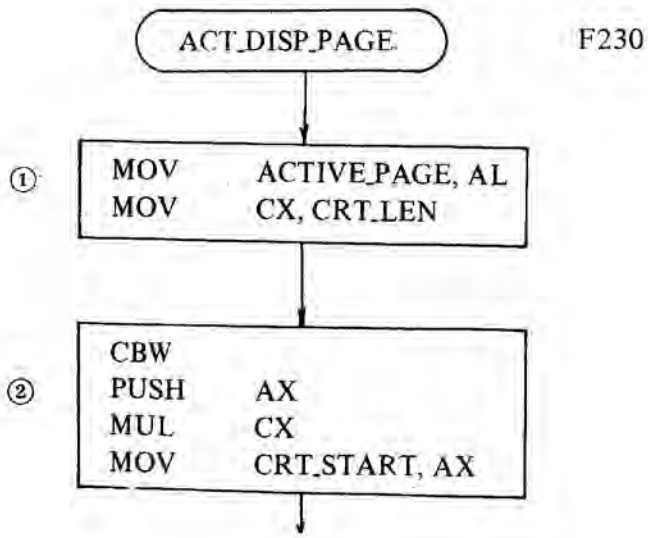
② 先以BX暫存器做指標從游標位置表格內取出相對於顯像頁的游標位置值到DX暫存器內。再將游標模式（cursor mode）的值放到CX暫存器內。

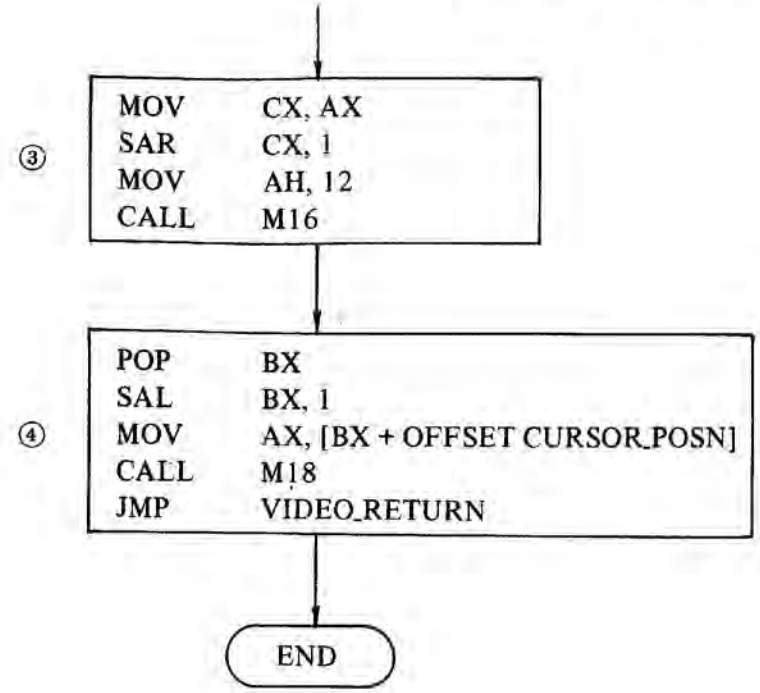
③ 我們不從VIDEO_RETURN返回到呼叫INT 10H的程式去，因爲VIDEO_RETURN會取回DX和CX的初值，但此時我們並不希望如此做，所以我們從這裡返回到呼叫INT 10H的程式去，和VIDEO_RETURN 同樣地先陸續取回DI，SI和BX的初值。

④ 將原先保存在堆疊中的DX和CX的初值取出放到AX暫存器內，這樣做的目的是爲了不破壞步驟②中得到的DX和CX暫存器的值。在陸續取出DS和ES暫存器的初值後，IRET指令會返回到呼叫INT10H的程式去。

(e) (AH) = 5 選擇顯像頁數

Input : (AL) = has the new active display page





① 先將顯像頁的值（在AL暫存器內）保存在ACTIVE_PAGE內。然後將CRT_LEN 值（也就是每一個頁佔用的記憶體數目，此值包括字元和屬性值）放到CX暫存器內。

② 將顯像頁值用CBW指令使其由位元組轉換成字組（請讀者自行參考CBW指令），並且保存在堆疊中，因爲步驟④還要用到此值。將顯像頁值乘以CRT_LEN（在CX暫存器內），並且將結果放到CRT_START內。如此CRT_START就指向該顯像頁的起始記憶體位址。

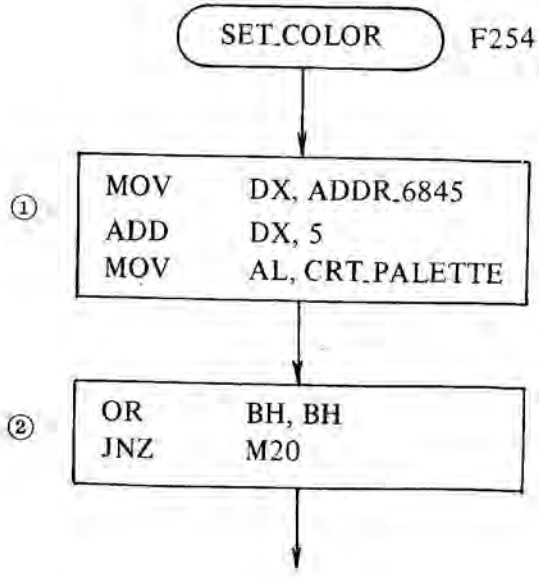
③ 將CRT_START值（在AX暫存器內）放到CX暫存器內。將CX暫存器內的值右移一個位元（即除2之意）是因爲6845掃描記憶體是以字元爲單位。在AH暫存器內

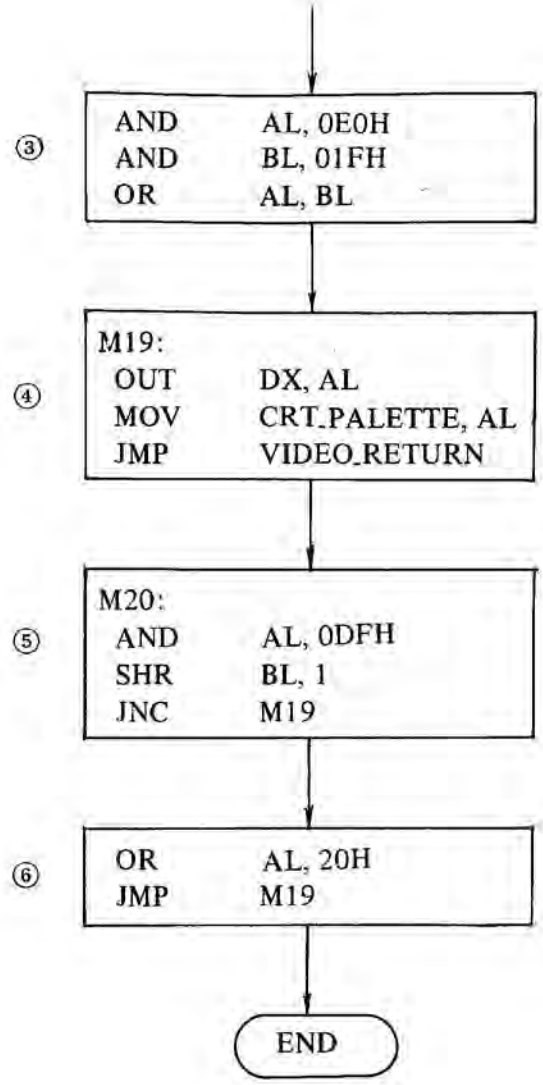
放入12，呼叫M16將CRT_START值寫到6845的起始位址暫存器（start address register）。

④ 取出在步驟②保存的顯像頁的值到BX暫存器內。先將其左移一個位元，因爲要以BX暫存器內的值做指標，從游標位置表格內取出該顯像頁的游標位置值，而游標位置表格內的每一個游標位置值都是二個位元組。接下來的MOV值取出該顯像頁的游標位置，然後呼叫M18將游標移至指定的位置。最後經由VIDEO-RETURN返回到呼叫INT 10H的程式去。

(f) (AH)=11 設定顏色組

Input: (BH) : has color ID
If (BH) = 0 , the background color value is set
from the low bits of BL (0-31)
If (BH) = 1 , the palette selection is made based
on the low bit of BL:
0 = Green, Red, Yellow
1 = Blue, Cyan, Magenta





① 從ADDR-6845內取出I/O埠值，再加上5，以便指向顏色選擇暫存器（3D9H），然後從CRT-PALETTE內取出原來的顏色值放到AL暫存器內。

② 此步驟檢查BH暫存器的值是否爲0，若爲0（ZF旗號爲1），則表示要改變背景的顏色，進入下個步驟。若結果不爲0（ZF旗號爲0），則表示要選擇顏色組（palette）跳入步驟⑤。

③ 進入此步驟是要改變背景的顏色。AND AL，0E0H指令將原來的背景清除掉，但不影響原來的顏色組值（位元5）。AND BL，01FH指令取出BL暫存器內決定背景顏色的位元（位元4到位元0）。OR AL，BL指令將上二個指令的結果組合起來（結果在AL暫存器內）。

④ 將顏色選擇（在AL暫存器內）寫到顏色選擇暫存器（3D9H）。然後再將這個顏色保存在CRT_PALETTE內。最後經由VIDEO_RETURN返回到呼叫INT 10H的程式去。

⑤ 進入此步驟是因爲要選擇顏色組，AND AL，0DFH指令將原來的顏色組值（位元5）清除爲0，而不改變原來的背景值。SHR BL，1是爲了檢查要選擇那一個顏色組，若結果是CF旗號爲1，則表示要選擇藍、青、紫紅這組顏色，進入下個步驟。若結果是CF旗號等於0，則表示要選擇綠、紅、黃這組顏色，跳入步驟④將顏色選擇值寫到顏色選擇暫存器（此時顏色選擇在AL暫存器內，其位元5爲0）。

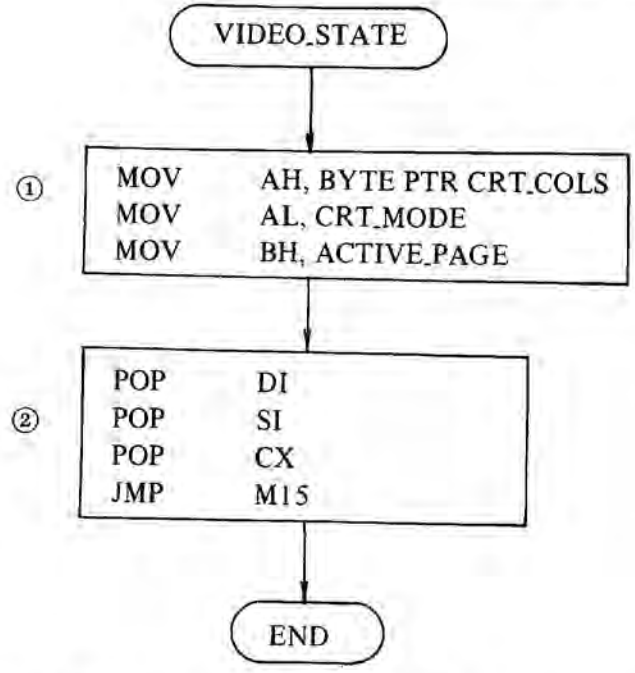
⑥ 將顏色選擇值的位元5設定爲1後，跳入步驟④，將此值寫到顏色選擇暫存器。

（g） (AH)=15 current VIDEO STATE

Output: (AL) = mode currently set

(AH) = Number of character columns on screen

(BH) = current active display page



① 這個步驟分別從CRT_COLS，CRT_MODE和ACTIVE_PAGE中取出值到AH，AL和BH暫存器內。

② 爲了不破壞在步驟①得到的ACTIVE_PAGE值，所以首先取回DI和SI暫存器的初值，再將原來保存在堆疊中的BX值取出放到CX暫存器內，這樣做是爲了不破壞BH暫存器內的值。讀者不必擔心會破壞CX暫存器的初值，因爲M15會取出CX暫存器的初值到CX暫存器內。M15是VIDEO_RETURN的一部份。

(h) (AH) = 6 scroll active page up

Input : (AH) = current CRT mode

(AL) = number of rows to scroll

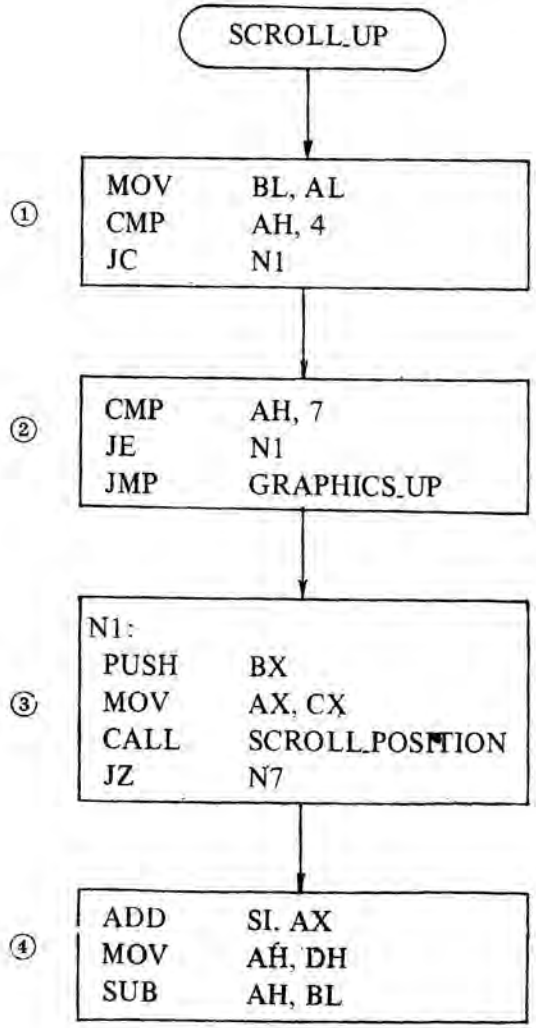
(CX) = Row/column of upper left corner

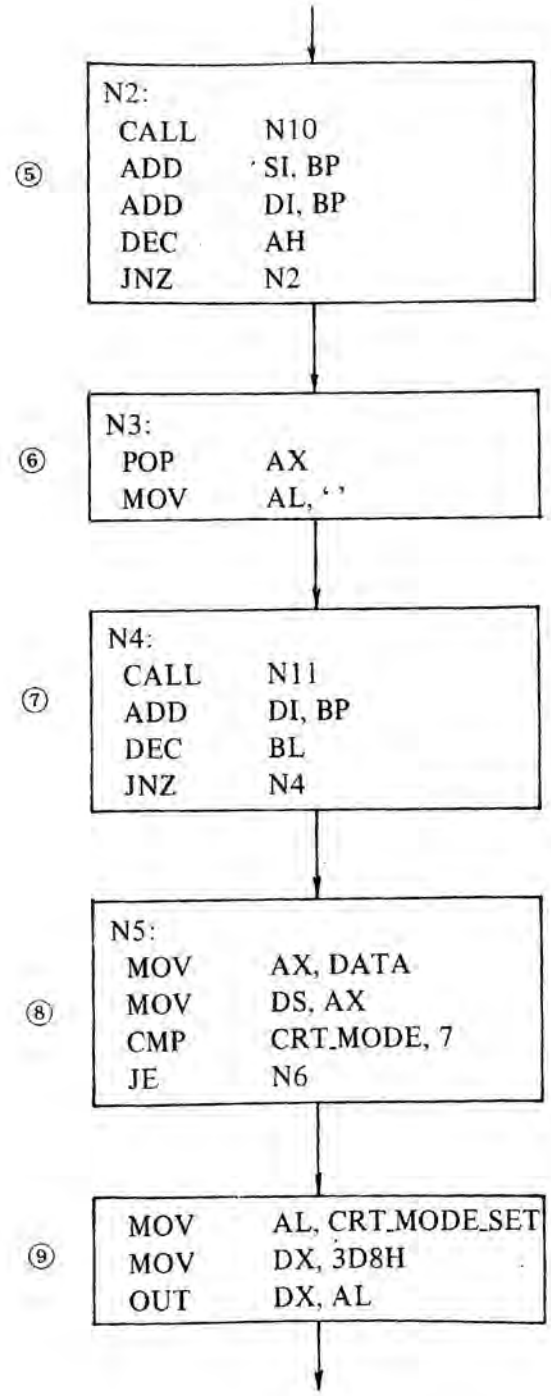
(DX) = Row/column of lower right o corner

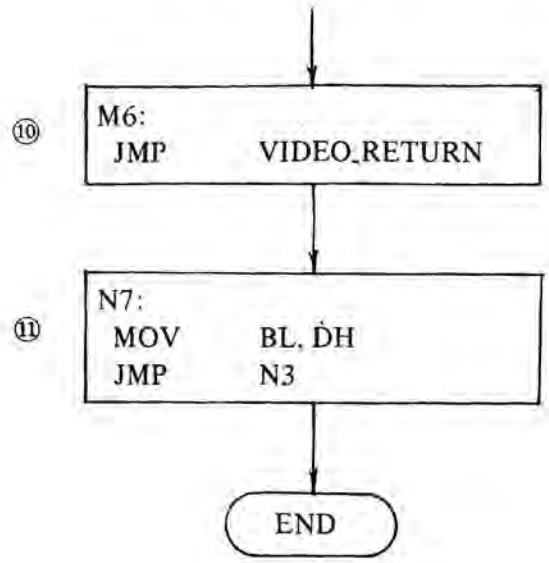
(BH) = attribute to be used on blanked line

(DS) = data segment

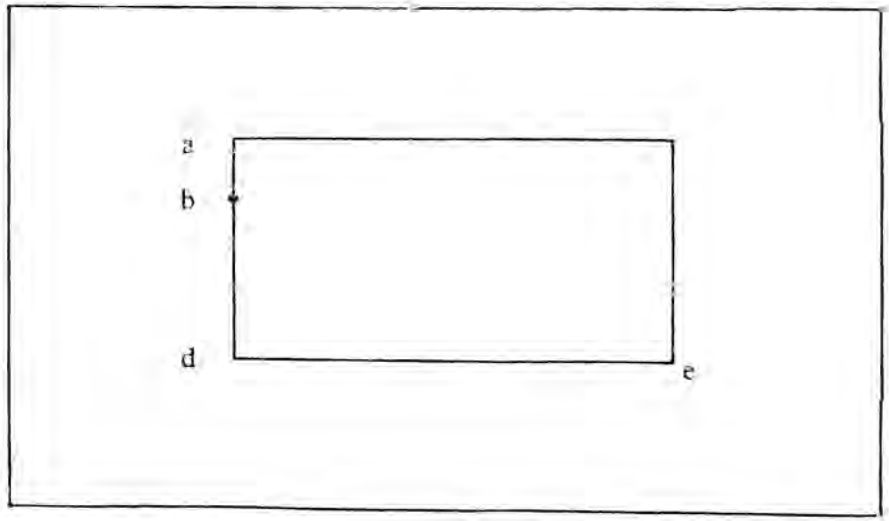
(ES) = Regen buffer segment







在討論這個程式之前，我們先來看上捲（ scroll up)的概念，請讀者務必了解這裡的介紹，因爲討論程式時，將引用這裡的例子。請先看下圖。



註：這個圖適用於文字的顯像模式。

外圍的大方格表示著整個作用頁(active page），列數均爲25，行數在CRT_COLS內（此值只有字元數）。中間的小方格是我們要上捲的視窗（ window），視窗內的列數爲d－a＋1，行數爲e－d＋1（註：a，b，c，d，e的座標均爲（列，行））。。如果現在要上捲n列，則先要找出a點的實際記憶體位址，其值爲CRT_START加上a點的列數值乘以2（因爲尚有屬性值），再乘以CRT_COLS，再加上a點的行數值乘以2。其次，將a點的實際記憶體值加上n乘以CRT_COLS再乘以2，如此便指向b點。此時要搬移記憶體內的值了。先將b點內的值搬到a點（註：一次搬二個位元組，因爲還有屬性值），然後將a點和b點的記憶體位址都加2，以便指向下一行，接著也將b點加2內的值搬到a點加2的位址，如此繼續下去，直到視窗內的行數都搬完了。接下來要搬移下一列的值，先將a點和b點的記憶體位址都加上2乘以CRT_COLS值，以便指向下一列，然後將這一列的值搬上去。如此繼續下去，直到視窗的下方留下n列爲止（也就是要搬動d－a＋1－n列）；此時已完成了上捲n列的動作。最後的動作是將屬性值填到所空出來的n列。

有了上面的概念後，我們來看上捲這個程式如何來實行上面的概念。

① 先將要上捲n列的值（在AL暫存器內）保存在BL暫存器內，然後將顯像模式值（在AH暫存器內）和4比較，這是因爲繪圖的情形要分開處理。若爲40×25或80×25模式（ CF旗號爲1），則進入步驟③，若CF旗號爲0，則可能是繪圖或單色，進入下個步驟再做一次檢查。

② 將顯像模式值和7比較，若相等則爲單色，進入步驟③，若不相等（ ZF旗號爲0）則爲繪圖，跳入Graphics-up去做繪圖的上捲動作。graphics-up將會在繪圖的部

分中討論。

③ 先將 BX暫存器內的值保存起來，其中 BH暫存器內爲屬性值，BL 暫存器內爲上捲 n 列的值，然後將視窗的左上角的座標（即 a 點，在 CX 暫存器內）放到 AX 暫存器內，以便呼叫 Scroll_Position 副程式算出 a 點的實際值。從 Scroll_Position 副程式回來時，（ES：DI）和（DS：SI）都指著 a 點，AX 暫存器內的值指出 a 點和 b 點的間距（OFFSET）值。Scroll_ Position 副程式的最後一個指令是 CMP BL，0，也就是將上捲 n 列的值和 0 比較。所以在這裡要測試 ZF 旗號，若 ZF 旗號爲 1（BL 等於 0），則表示要清除整個視窗，而填上屬性值，跳入步驟⑪。若 ZF 旗號爲 0，則進入下個步驟。

④ 將 SI 暫存器內的值加上 AX 暫存器內的值，如此（DS：SI）即指向 b 點。從 Scroll_Position 副程式回來的 DH 值爲視窗內的列數值，DL 值爲視窗內的行數值（DH＝d－a＋1，DL＝e－d＋1），這裡將 DH 值轉到 AH 暫存器內，再減去 BL 值（即上捲的列數值），如此即得到了要搬動的列數值（d－a＋1－n），結果在 AH 暫存器內。

⑤ 呼叫 N10 搬一列，返回後 SI 和 DI 都加上 BP 內的值（BP 內的值爲 CRT_COLS 的二倍，在 Scroll_Position 中運算出）以便指向下一列。將 AH 暫存器內的值減 1，若結果不爲 0（ZF 旗號爲 0），則表示尚未搬動 d－a＋1－n 列，返回到此步驟的前頭繼續搬。若結果爲 0（ZF 旗號爲 1），則表示已搬了 d－a＋1－n 列，進入下個步驟將視窗下方空出來的 n 列，填上屬性值。

⑥ 從堆疊中取出在步驟③保存的 BX 值到 AX 暫存器內。此時 AH 暫存器內爲屬性值。然後在 AL 暫存器內放入空白的 ASCII 碼。

⑦ 呼叫 N11 將屬性值填滿一列。返回後將 DI 加上 BP 內的值，以便指向下一列，將 BL 內的值（即視窗下方空出來的 n 列值）減 1，若結果爲 0（ZF 旗號爲 1），則表示 n 列都已填上屬性值了，進入下個步驟。若結果不爲 0（ZF 旗號爲 0），則需要繼續填屬性值，回到此步驟的最前頭。

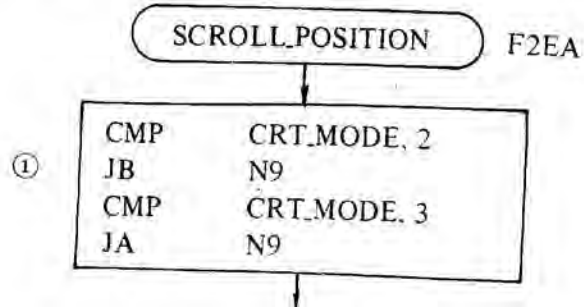
⑧ 在 DS 內放入 DATA，以便取出 CRT_MODE 值，將 CRT_ MODE 值和 7 比較，若相等（ZF 旗號爲 1），進入步驟⑩，若不相等，則需要進入下個步驟，設定 I/O 埠 3D8H 的值。

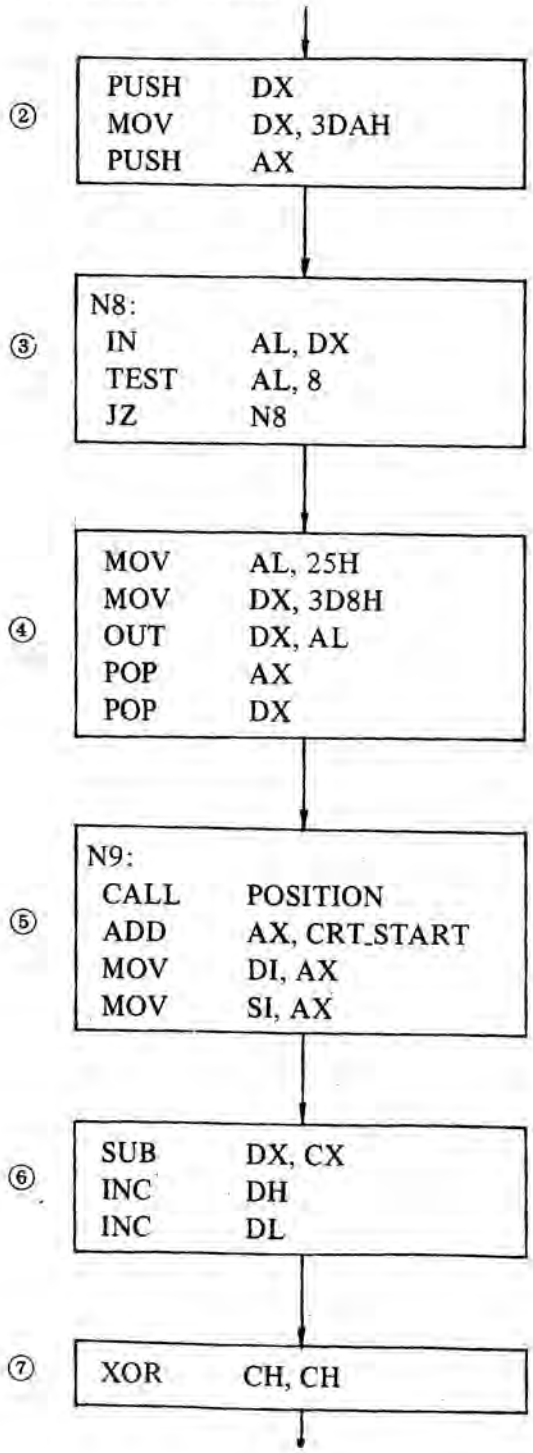
⑨ CRT_ MODE _SET 的值即 I/O 埠 3D8H 的原來值，在 Scroll_Position 副程式中可能會將此 I/O 埠的值破壞（80×25 顯像模式），所以這裡要保證 I/O 埠的正確。將 CRT_MODE_SET 的值寫到 3D8H 去。

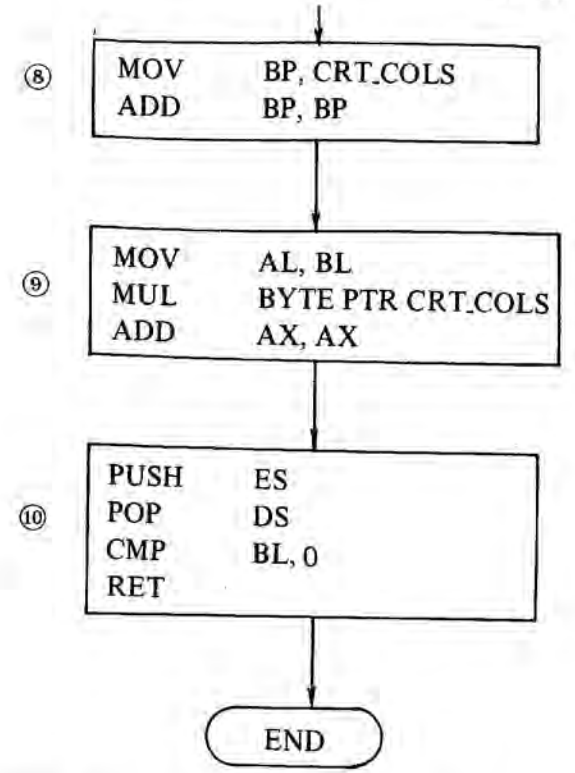
⑩ 經由 VIDEO_RETURN 返回到呼叫 INT 10H 的程式去。

⑪ 進入此步驟是因爲上捲 n 列的值爲 0（表示要整個視窗都填入屬性值。將視窗內的列數（在 DH 內）放到 BL 內，跳到步驟⑥去做填屬性值的動作。此時（ES：DI）還是指著 a 點。

我們現在來看 Scroll_Position 副程式







① 這個步驟在測試是否爲 80×25 顯像模式，若爲 80×25，則在搬動螢幕畫面以前，先要關閉視頻信號，進入下個步驟，若不爲 80×25，則直接進入步驟⑤。

② 保存DX和AX的值，在DX內放入 3DAH ，以便讀取垂直同步信號。

③ 此步驟等待垂直同步信號發生，垂直同步信號發生後，才進入下個步驟。

④ 在同步信號發生時，將 25 H 寫到 3D 8H ，如此即可關閉視頻信號，這樣做是爲了不影響畫面的顯示。然後取回AX和DX的初值。

⑤ 進入此程式的 AX 值爲視窗的左上角的座標（上捲時；若爲下捲則爲右下角）。呼叫 POSITION副程式算出它的

記憶體間距值，返回後再加上CRT_START值，如此AX內的值即相對於顯像緩衝區的起始位址的間距值。同時將AX內的值保存在DI和SI內。

⑥ 此步驟算出視窗內的列數和行數，DX爲右下角，CX爲左上角的座標，視窗的列數在DH內，行數在DL內。

⑦ 本來CH內的值爲視窗左上角的列座標，因爲在N10和N11中都使用了REP指令，而且LOOP的次數只需要CL內的值即可，所以在這裡先將CH清除爲0。

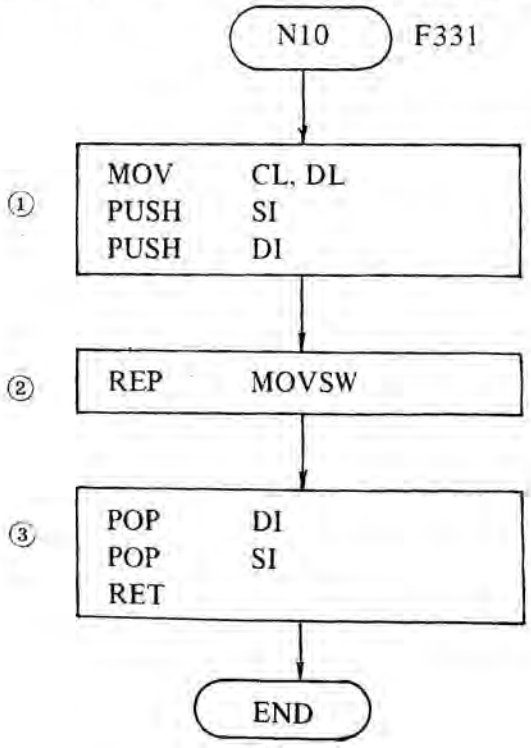
⑧ 這裡將CRT_COLS內的值乘以2，放到BP內。

⑨ 此步驟算出a點與B點的間距（OFFSET）值，乘以2是因爲屬性值，上捲n列的值在BL內。

⑩ 此步驟將DS和ES都指向顯像緩衝區，如此（ES:DI）和（DS:SI）都指著a點，而a點和b點的間距（OFFSET）值在AX內，BP內的值爲CRT_COLS的值的2倍。

。最後測試上捲n列的值（在BL內）是否爲0，然後返回。

我們接著看搬動記憶體內的值的工作者：N10。

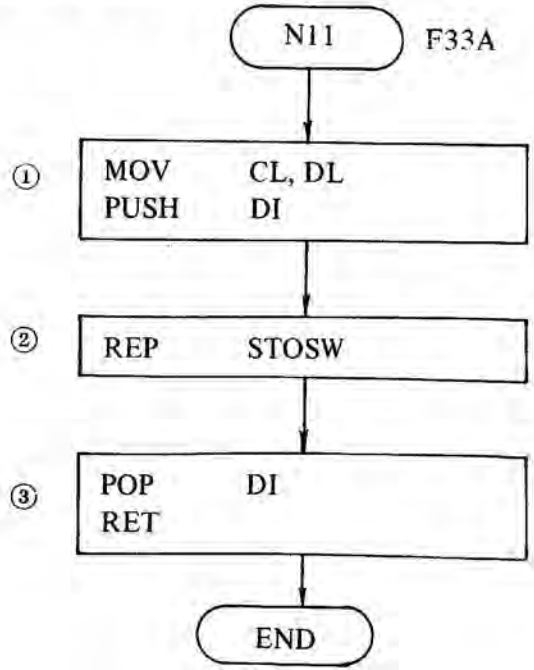


① 將視窗內的行數（在DL暫存器內）放到CL內，以便配合REP指令。然後將SI和DI暫存器內的值保存起來。

② 這個步驟做搬動記憶體內的值的工作。每做一次，CX內的值會減1，直到CX內的值變成0爲止，才停止搬移的工作，當CX內的值爲0時，也就表示已搬完了整個的行數。每次搬二個位元組，其一爲字元，另一爲屬性，搬移的動作是這樣的：將（DS:SI）所指的記憶體位址內取出二個位元組搬到（ES:DI）所指的記憶體位址內，然後將DI和SI都加2，以便指向下一行，若CX值不爲0，則重覆此動作。

③ 取回DI和SI的初值後返回，返回的程式應該將DI和SI都加上BP內的值（CRT_COLS的二倍），以便指向下一列，再呼叫此程式搬移下一列，直到搬完該搬的列數爲止。

我們接著看收尾的工作者N11如何填上屬性值。



① 將視窗內的行數（在DL內）放到CL內，以便配合REP指令。然後將DI內的值保存起來，因爲STOSW只使用到DI，而不用SI，所以只需保存DI即可。

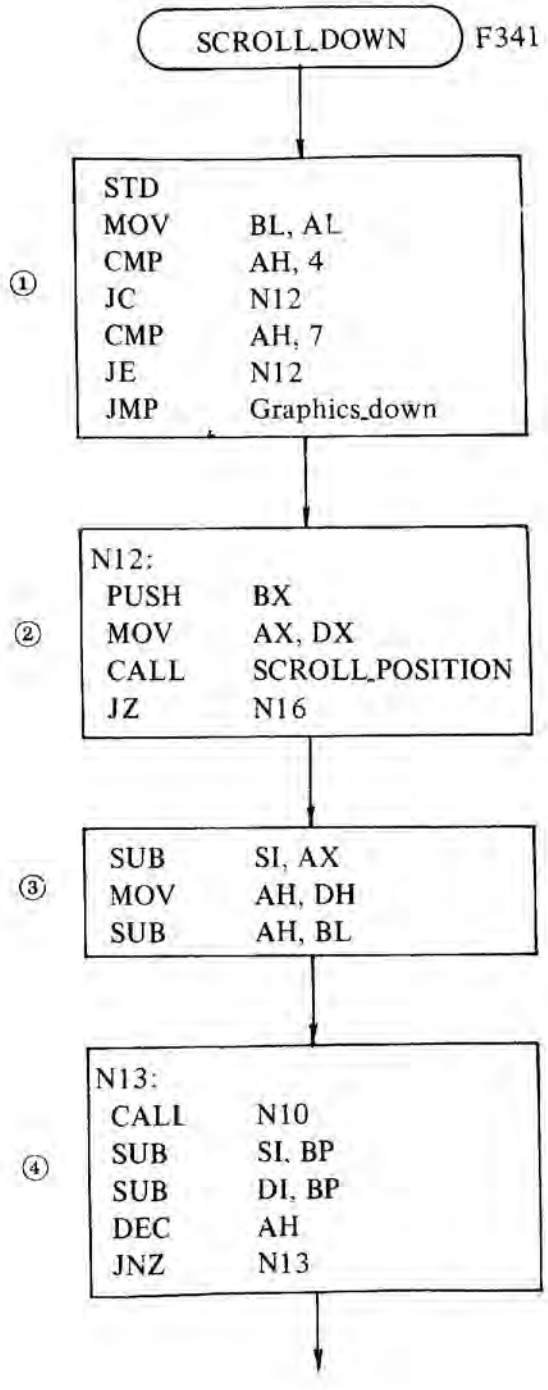
② STOSW指令將AX內的值搬到（ES：DI）所指的記憶體位址內，因爲有REP指令，所以每執行一次STOSW，CX值會減1，DI值會加2。CX值減1是爲了檢查是否已搬完了視窗內的行數。DI值加2是爲了指向下一行。STOSW指令將AH內的值（屬性）放到高位址，AL內的值（字元此時爲空白）放到低位址。

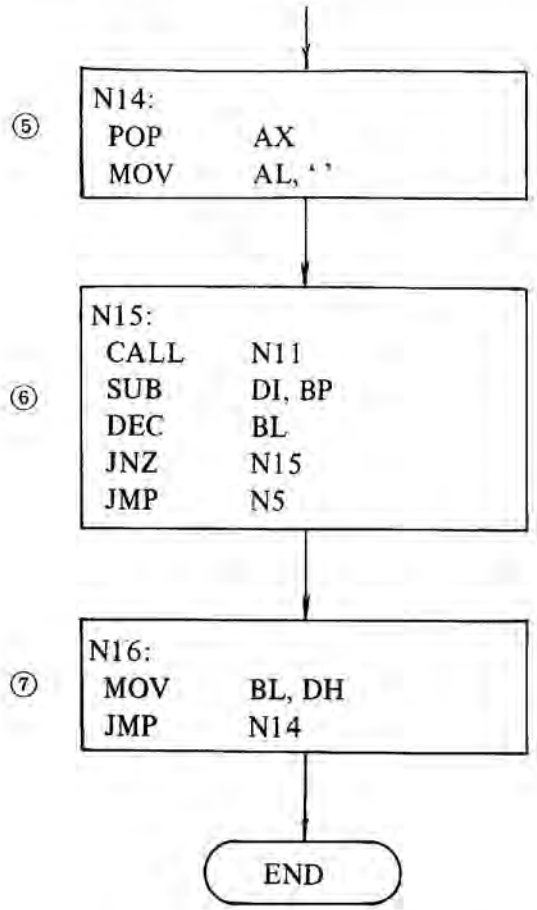
③ 取回DI的值，然後返回，返回的程式應該將DI內的值加上BP內的值（CRT_COLS的二倍），以便指向下一列，然後再呼叫此程式，直到填完了該填的列數（即捲動的列數）爲止。

好了，我們已將整個上捲的動作介紹完了，筆者請讀者自己再TRACE一次，以便有更深入的瞭解。

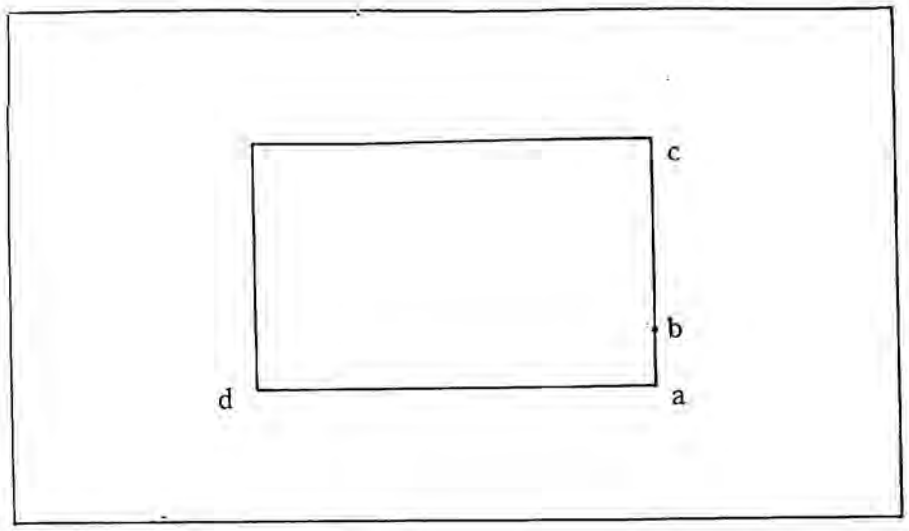
(i) (AH) = 7 Scroll active page down

Input : (AH) = current CRT MODE
(AL) = number of lines to scroll
(CX) = upper left corner of region
(DX) = lower right corner of region
(BH) = fill attribute
(DS) = DATA segment
(ES) = Regen segment





下捲（Scroll down）和上捲（Scroll up）大致相同，最大的差異在方向，也就是DF旗號的值。上捲的動作爲由左至右，由上而下，但下捲的動作爲由右至左，由下而上。下捲也使用捲動位置N10和N11副程式。讀者只要注意方向，要了解下捲就簡單多了。先看下例的說明。



註：此圖只適用文字顯像模式

外圍的方格爲整個螢幕畫面，列數均爲25，行數在CRT_COLS內。中間的小方格是要下捲的視窗，視窗內的列數爲a－c＋1，行數爲a－d＋1。若現在要下捲n列，則先找出a點的實際記憶體位址，然後將a點的實際位址減去n乘以CRT_COLS再乘以2的值（因爲有屬性值），如此即得到b點的實際位址。接下來要搬移記憶體內的值了，先將b點內的值搬到a點，然後將指標都減2，以便指向左一行，將b－1行內的值搬到a－1行內，如此繼續下去，直到將視窗內的行數搬完爲止。接下來將指標分別指到b列和a列的上一列，同樣地，將這一列內各行的值都搬移，如此繼續下去，直到視窗的上方空出n列爲止（也就是要搬動a－c＋1－n列）。最後將屬性值填到視窗上方的n列。

我們現在來討論下捲程式。

① 首先設定DF旗號爲1，如此在N10和N11內的MOVSW和STOSW指令執行後，會將SI和DI內的值減2。請讀者翻至本書的前面，再複習一下STOSW和MOVSW 指令，然後將要捲動的列數（在AL內）放到BL內。接下來的5個指令測試是否爲繪圖模式，若是，則要跳至graphics_down去執行動作，graphics_down在繪圖中討論。

② 將BX內的值保存起來，然後將右下角的座標（即a點，在DX內）放到AX暫存器內，呼叫scroll_position。從scroll_position回來時，（ES:DI）和（DS:SI）都指著a點，AX內爲a點和b點的間距（OFFSET）值，BP內爲CRT_COLS值的2倍（因爲有屬性值），DH內爲視窗內的列數，DL內爲視窗內的行數。捲動位置（scroll position）最後測試BL內的值（捲動的列數值），若等於0（ZF旗號爲1），則表示要將屬性值填滿整個視窗，進入步驟⑦。

③ SUB SI,AX 指令使得（DS:SI）指著b點，接下來的二個指令算出要搬動的列數值（a－c＋1－n）在AH暫存器內。

④ 呼叫N10搬動一列，注意此時DF旗號爲1。在前面所討論的N10和N11都是往上捲，DF等於0爲依據做說明，這時讀者若要看討論的內容時，要注意方向。返回後，將SI和DI都減去BP內的值，以便指向上一列，然後測試是否已搬動（a－c＋1－n）列，若尚未完成則繼續，若已完成，進入下個步驟。

⑤ 取出在步驟②中保存的BX值，此時AH內爲屬性值，然後在AL內填入空白的ASCII碼。

⑥ 呼叫N11填滿一列的屬性值，返回後將DI內的值減去BP內的值，以便指向上一列，然後測試是否已填滿了n列，若尚未完成則繼續，若已完成（ZF旗號爲1），則跳入上捲的步驟⑧，保證I/O埠3D8H的值正確。

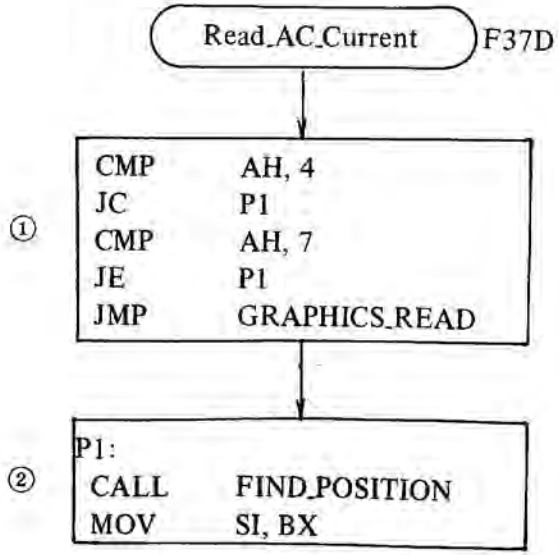
⑦ 進入此步驟是因爲要將整個視窗都填入屬性值。將DH值（視窗內的列數值）放到BL內，然後跳入步驟⑤去做填屬性值的動作。此時，（ES:DI）仍指著a點（右下角）。

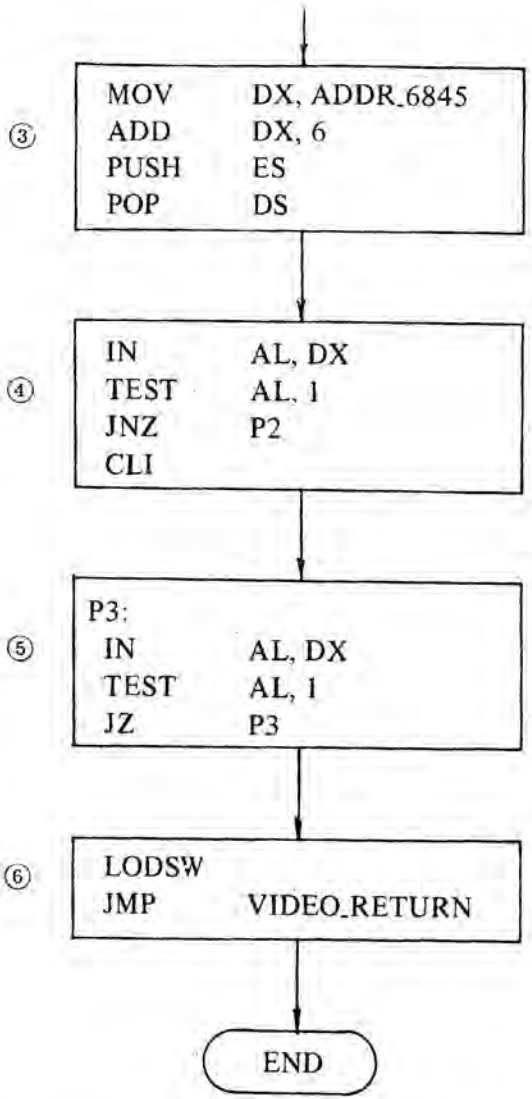
（j） (AH) = 8 Read Attribute/Character at current cursor position

This routine reads the attribute and character at the current cursor position and returns them to the caller

Input : (AH) = current CRT mode
(BH) = display page
(DS) = data segment
(ES) = Regen segment

Output : (AL) = character read
(AH) = attribute read





① 這個步驟檢查CRT mode，若爲繪圖，則跳到graphics_read去。graphics_read在繪圖中討論。

② 首先呼叫FIND_POSITION，找出游標位置的實際記憶體位址，結果在BX暫存器內。返回後將此值轉到SI暫存器內。

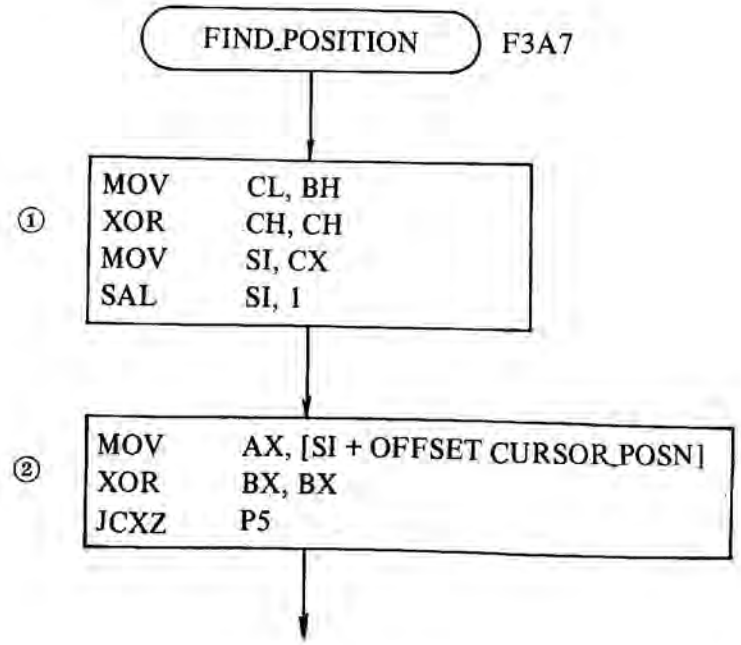
③ 首先將I/O埠值指向狀態暫存器，這是因爲要在水平Retrace時讀出游標位置內的值，然後將DS暫存器也指向Regen buffer。

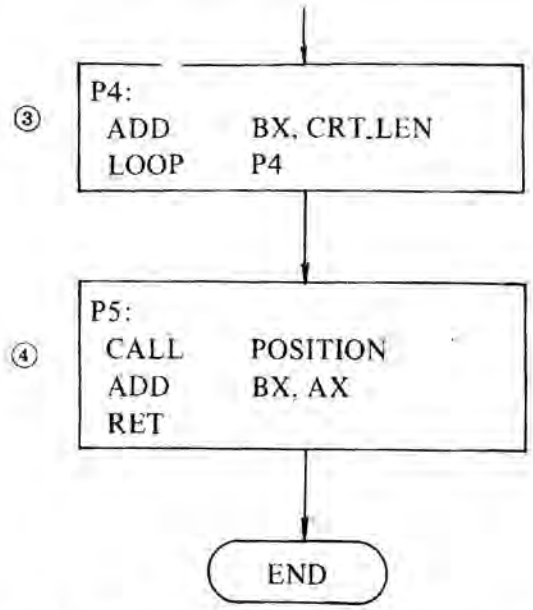
④ 此步驟測試水平同步信號是否爲低電位，若不是，則繼續等待。若已是低電位了，則將IF旗號設定爲0，以便在讀資料時不被中斷。

⑤ 此步驟測試水平同步信號是否爲高電位，若仍是低電位，則繼續等待。若水平同步信號變成高電位（Retrace時間），進入下個步驟。

⑥ 進入此步驟的(DS:SI)指著游標位址的記憶體，LODSW指令將其讀出放到AX暫存器內。其中字元（低位元組）在AL內，屬性（高位元組）在AH暫存器內，最後經由VIDEO_RETURN返回到呼叫INT 10H的程式去。

我們接下來看FIND_POSITION副程式





① 將顯像頁（在BH內）放到CL內，並清除CH暫存器，然後將CX內的值轉到SI內。這裡的CX值也就是顯像頁數目，將在步驟③中用到此值。將 SI 暫存器內的值左移一位（即乘以2之意），因爲在下個步驟中將使用 SI 暫存器做指標取出游標位置，而游標位置表格內，每一個位置使用二個位元組。

② 從游標位置表格內取出相關於該顯像頁的游標位置到 AX 暫存器內。清除 BX 暫存器，然後測試 CX 暫存器內的值是否爲 0，若爲 0，則直接進入步驟④；若不爲 0，進入下個步驟，找出該顯像頁的起始記憶體間距（OFFSET）值（此間距（OFFSET）值相對於Regen buffer)。

③ 此步驟找出該顯像頁的起始記憶體間距（OFFSET）值，此時 CX 暫存器內的值爲顯像頁的數目。

④ 呼叫 POSITION 副程式，算出游標位置相對於該顯像頁的起始位址的間距（OFFSET）值，結果在AX暫存器內。

返回後，將 AX 暫存器內的值加上 BX 暫存器內的值，結果在 BX 暫存器內，如此 BX 暫存器內的值爲游標位置相對於 Regen buffer 的記憶體間距（OFFSET）值。最後返回到呼叫 FIND_POSITION 副程式的程式去。

(k) (AH) = 9 write attribute/character at current cursor position

This routine writes the attribute and character at the current cursor position

Input : (AH) = current CRT mode

(BH) = display page

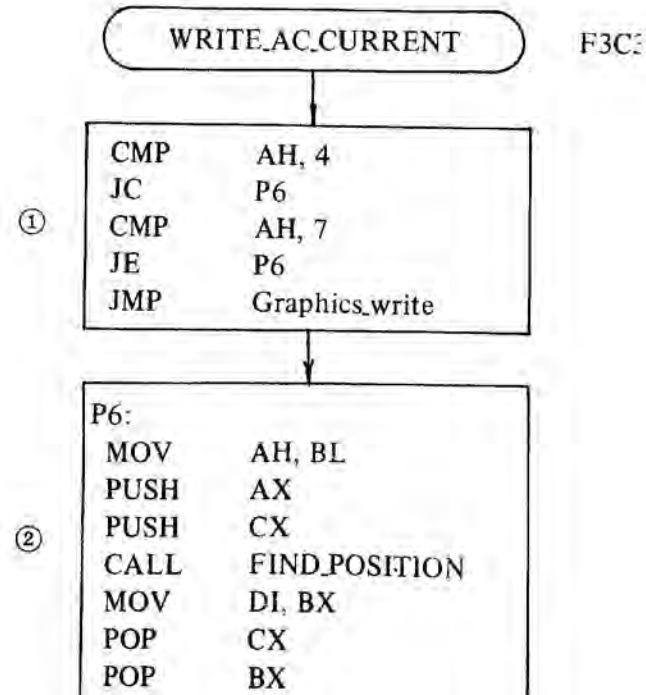
(CX) = count of characters to write

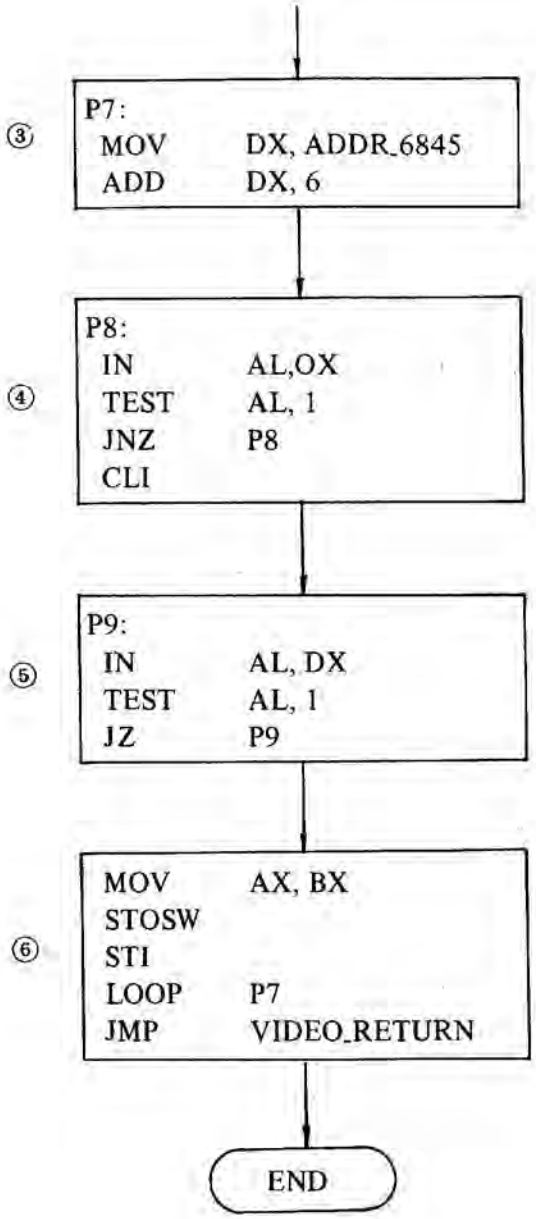
(AL) = character to write

(BL) = attribute to write

(DS) = data segment

(ES) = Regen segment





① 此步驟檢查CRT模式，若爲繪圖，則跳入graphics_write,graphics_write在繪圖中討論。

② 先將屬性值放到AH暫存器內，如此AX暫存器內就保有了屬性（在AH內）和字元（在AL內）值。因爲FIND-POSITION副程式會破壞AX和CX值，所以先保存在堆疊中。呼叫FIND_POSITION副程式算出游標位置相對於Regen Segment的記憶體間距（OFFSET）值。返返回後，將此值轉到DI暫存器內。然後取出CX暫存器的初值，並取出AX暫存器的初值到BX暫存器內，如此BX暫存器保存了字元/屬性值。

③ 此步驟將I/O埠值指向狀態暫存器。

④ 此步驟測試水平同步信號是否爲低電位，直到爲低電位時才設定IF旗號爲0，進入下個步驟。

⑤ 此步驟等待水平同步信號成爲高電位（Retrace時間）。

⑥ 將字元/屬性值（在BX暫存器內）轉到AX暫存器內，然後STOSW將字元/屬性值寫到游標的位置（由ES:DI指著）。字元放在ES:DI所指的位址的低位元組，屬性放在高位元組。在設定IF旗號爲1後，依據CX暫存器內的值，做重覆的動作（步驟③）。執行STOSW指令後，會自動將DI暫存器內的值加2，以便指向下一個位置，最後經由VIDEO_RETURN返回到呼叫INT 10H的程式去。

(1) (AH)=10 write character only at current cursor position

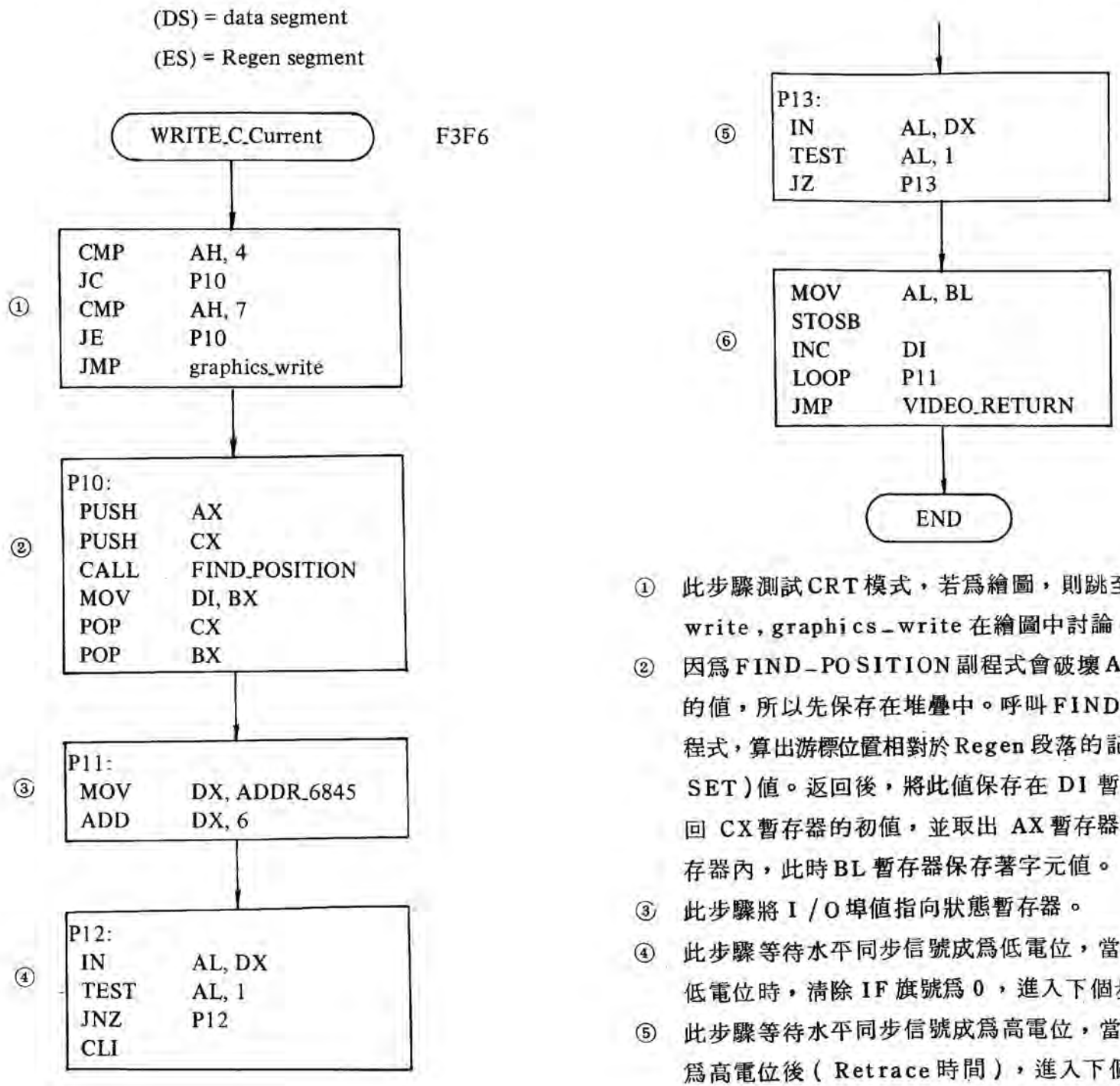
This routine writes the character at the current cursor position, attribute unchanged.

Input : (AH) = current CRT mode

(BH) = display page

(CX) = count of characters to write

(AL) = character to write



① 此步驟測試CRT模式，若爲繪圖，則跳至graphics_write，graphics_write在繪圖中討論。

② 因爲FIND_POSITION副程式會破壞AX和CX暫存器的值，所以先保存在堆疊中。呼叫FIND_POSITION副程式，算出游標位置相對於Regen段落的記憶體間距（OFFSET）值。返回後，將此值保存在DI暫存器內，然後取回CX暫存器的初值，並取出AX暫存器的初值到BX暫存器內，此時BL暫存器保存著字元值。

③ 此步驟將I/O埠值指向狀態暫存器。

④ 此步驟等待水平同步信號成爲低電位，當水平同步信號爲低電位時，清除IF旗號爲0，進入下個步驟。

⑤ 此步驟等待水平同步信號成爲高電位，當水平同步信號成爲高電位後（Retrace時間），進入下個步驟。

⑥ 將字元值（在BL暫存器內）放到AL暫存器內。然後STOSB指令將字元放到游標的位置（由ES:DI指著），因爲執行STOSB完後，DI暫存器內的值會自動加1，爲了要指向下一個位置，所以還要將DI暫存器內的值加1，這是因爲螢幕上每個位置都佔了二個位元組，一爲字元，一爲屬性。這裡只改變字元，而沒有改變屬性。然後根據CX暫存器內的值，做重覆的動作（由步驟③起）。最後經由VIDEO_RETURN返回到呼叫INT 10H的程式去。

接下來我們將討論六個繪圖模式中的功能，graphics_write graphics_read，graphics scroll_up，graphics scroll_down，write dot read dot。在討論這些功能之前，我們必須知道螢幕畫面和顯像緩衝區的關係。我們知道繪圖有二個模式，一爲中解析度（320×200），另一爲高解析度（640×200）。我們分開討論。在中解析度中，螢幕上的一點由二個位元構成，一個位元組可構成四個點，其結構如下：

位元	7 6	5 4	3 2	1 0
	C1 C0	C1 C0	C1 C0	C1 C0
	第1點	第2點	第3點	第4點

一個位元組顯像出4個點，在螢幕上第一點在左方，第四點在右方，四個點是連續的。

中解析度一列有320個點，所以由80個位元組組成一列，這80個位元組在記憶體上是連續的，但奇數列和偶數列的記憶體不是連續的。下圖中，我們列出每列的起始記憶體位址。它們的分段值均爲B800H，我們只列出它們的間距值。

螢幕畫面上的列數	起始位址
0	0H
1	2000H
2	50H
3	2050H
4	A0H
5	20A0H
⋮	⋮
⋮	⋮
依此類推	⋮

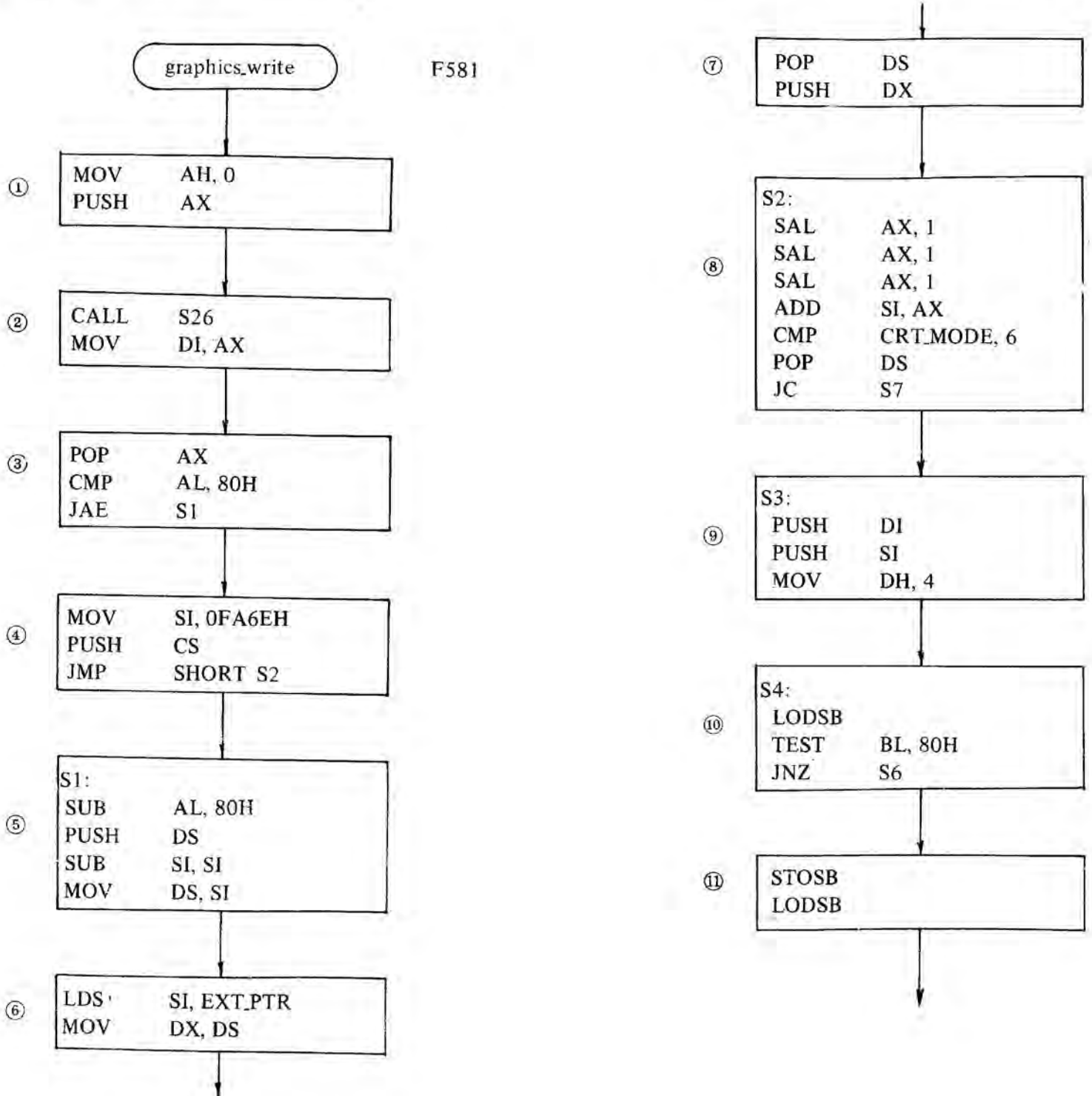
註：50H爲80D。

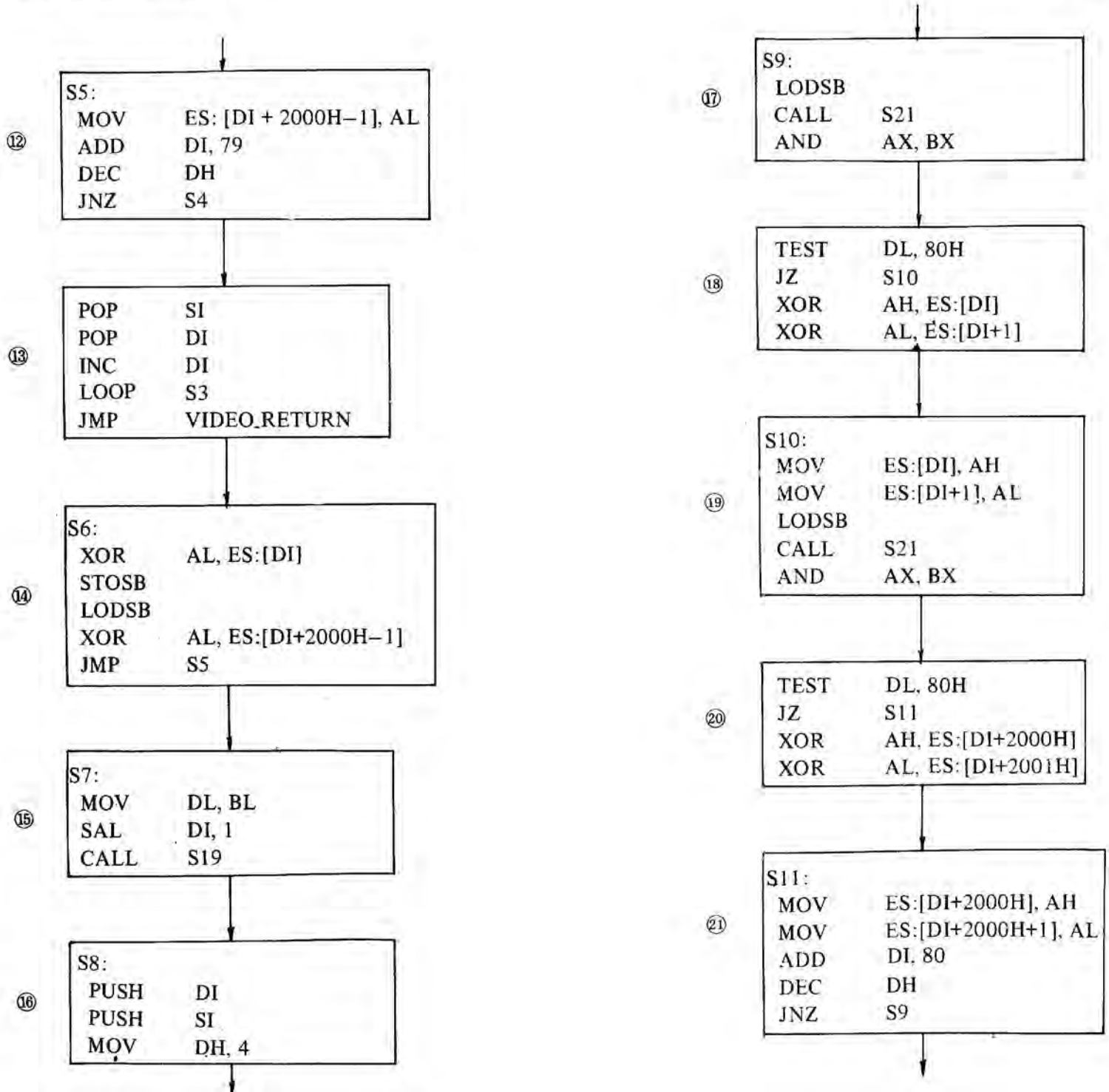
在高解析度中，螢幕上的一點由一個位元組成。高解析度和中解析度除了這點不相同以外，其餘均相同。當然，中解析度可以選擇顏色，而高解析度只能決定點的亮不亮而已。和中解析度一樣，高解析度的位元7代表第一點，位元0代表第8點。在螢幕上第一點在左方，第8點在右方，8個點是連續的。有了上面的概念後，我們就可以討論繪圖的常式（routine）了。

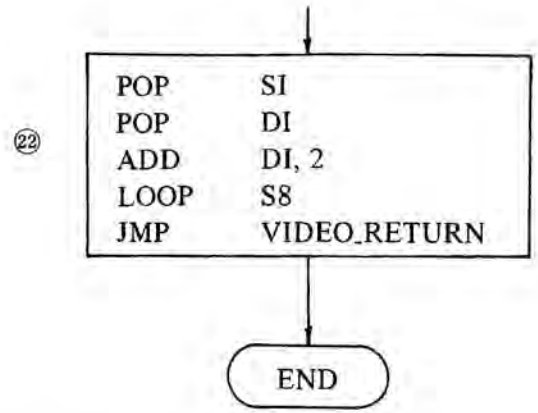
(m) graphics write

This routine writes the ASCII character to the current position on the screen

Input : (AL) = character to write
(BL) = color attribute to be used for foreground color
(CX) = number of characters to write
(DS) = data segment
(ES) = Regen segment


graphics_write
F581
①
MOV AH, 0
PUSH AX
②
CALL S26
MOV DI, AX
③
POP AX
CMP AL, 80H
JAE S1
④
MOV SI, 0FA6EH
PUSH CS
JMP SHORT S2
⑤
S1:
SUB AL, 80H
PUSH DS
SUB SI, SI
MOV DS, SI
⑥
LDS SI, EXT_PTR
MOV DX, DS
⑦
POP DS
PUSH DX
⑧
S2:
SAL AX, 1
SAL AX, 1
SAL AX, 1
ADD SI, AX
CMP CRT_MODE, 6
POP DS
JC S7
⑨
S3:
PUSH DI
PUSH SI
MOV DH, 4
⑩
S4:
LODSB
TEST BL, 80H
JNZ S6
⑪
STOSB
LODSB


⑫
S5:
MOV ES: [DI + 2000H−1], AL
ADD DI, 79
DEC DH
JNZ S4
⑬
POP SI
POP DI
INC DI
LOOP S3
JMP VIDEO_RETURN
⑭
S6:
XOR AL, ES:[DI]
STOSB
LODSB
XOR AL, ES:[DI+2000H−1]
JMP S5
⑮
S7:
MOV DL, BL
SAL DI, 1
CALL S19
⑯
S8:
PUSH DI
PUSH SI
MOV DH, 4
⑰
S9:
LODSB
CALL S21
AND AX, BX
⑱
TEST DL, 80H
JZ S10
XOR AH, ES:[DI]
XOR AL, ES:[DI+1]
⑲
S10:
MOV ES:[DI], AH
MOV ES:[DI+1], AL
LODSB
CALL S21
AND AX, BX
⑳
TEST DL, 80H
JZ S11
XOR AH, ES:[DI+2000H]
XOR AL, ES:[DI+2001H]
㉑
S11:
MOV ES:[DI+2000H], AH
MOV ES:[DI+2000H+1], AL
ADD DI, 80
DEC DH
JNZ S9

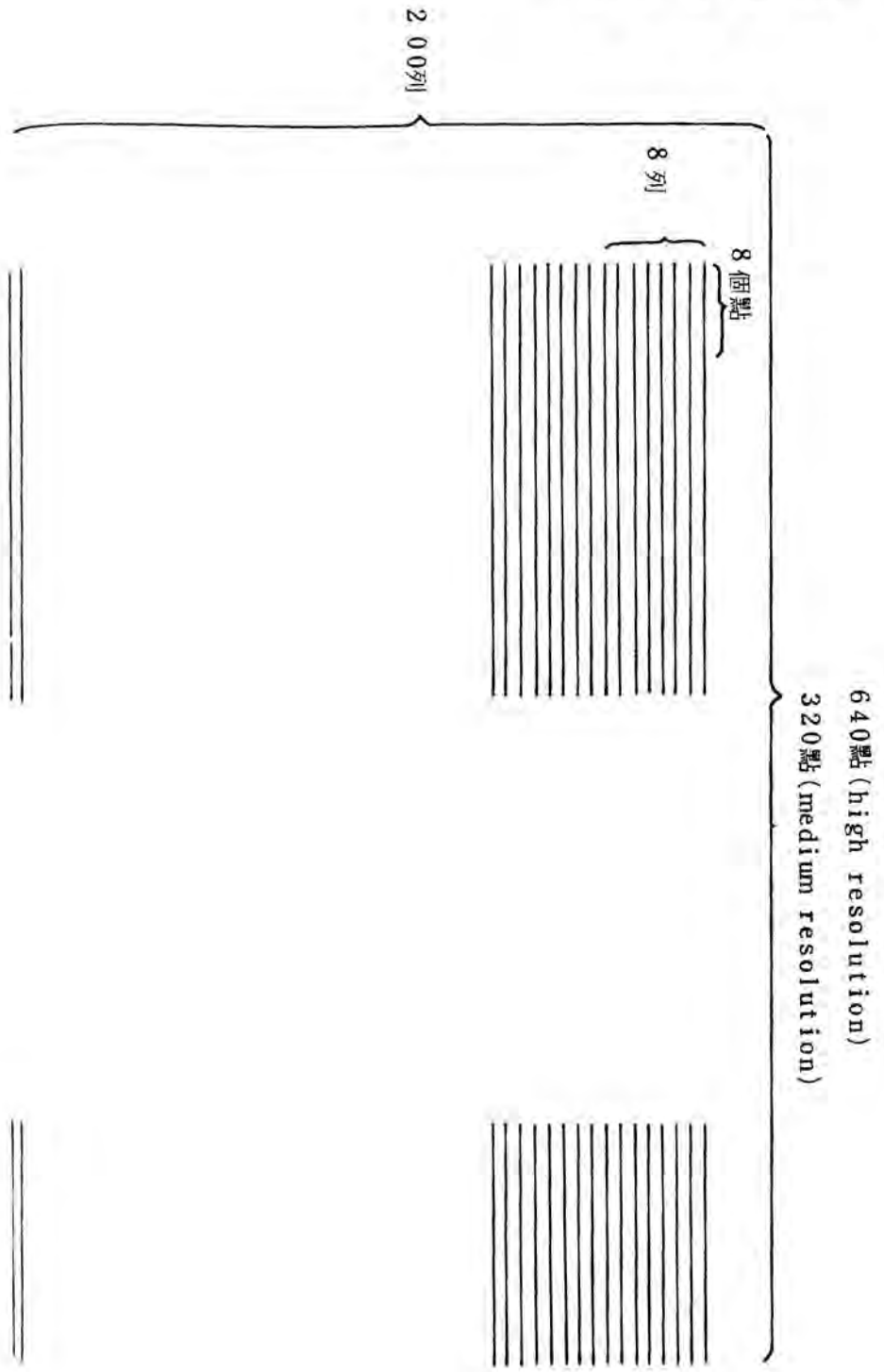


㉒

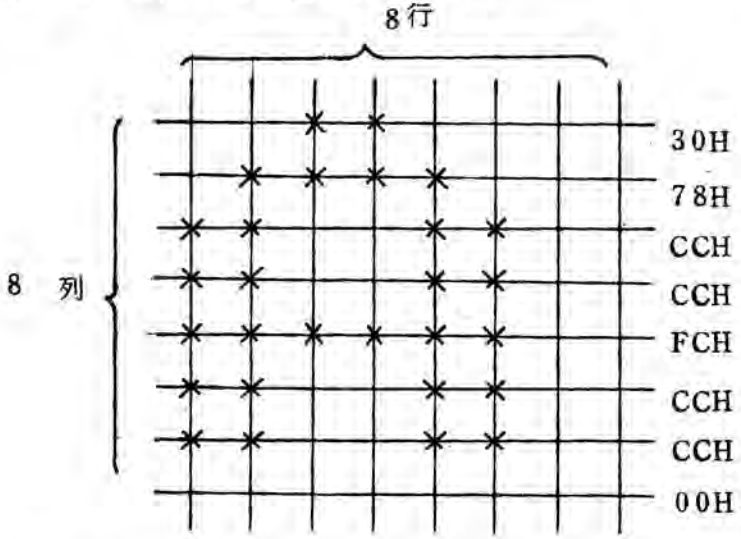
我們知道，在文字模式中，硬體綫路上有個字元產生器（character generator）。當我們要顯示字元時，只需將字元的ASCII碼放到指定的記憶體內即可。CRT控制器自然會把這個字元的字形顯像在螢幕畫面上。在繪圖中，硬體綫路上並不提供字元產生器，所以要在繪圖模式中顯像一個字元的字形是頗費周章的。

首先要了解的是螢幕畫面的結構，請看下圖。若以文字的眼光來看繪圖，我們可以將8個點（即8行）乘以8列的小方格當作一個字元儲存格（character cell），所以我們可以把中解析度看成40×25的顯像模式，把高解析度看成80×25的顯像模式。所不同的是，高解析度需要8個位元組來填滿一個字元儲存格，中解析度則更多，需要16個位元組在游標位置表格內所存放的列、行值就是以字元儲存格爲單位。如果說游標在第1列、第1行，那麼我們的意思是游標指著第8列到第15列、第8行到第15行這個範圍（在中解析度有200列×320行，高解析度有200列×640行，列與行均由0起算）（如下頁所示）。

在程式中，我們要將字形顯示在一個字元內，我們不可能將程式切成256個小段，以便分別來處理不同的ASCII碼（ASCII碼有256個），所以我們要建立字形的表格。這個表格是依照



ASCII 的順序和字形建立起來的，每一個ASCII碼佔用 8個位元組記憶體，這是因爲一個字元儲存格有 8 列。我們拿A這個字元爲例，A的ASCII 碼爲 41 H，則描出A的字形的資料在字形表格內，距起始位址 208H的位置處，這些資料連續佔用 8 個位元組（所以 B 的字形資料在間距 210 H 起的 8 個位元組。這些資料分別是 30H，78H，CCH，CCH，FCH，CCH，CCH，00H，只要程式依序的將這些值填到字元儲存格內，即可顯出A的字形。



IBM PC BIOS 侷限於大小的關係，字形表格只有一半，另一半需要使用者自行建立，而將其起始位址放到中斷表內向量值爲 1FH 的位址。

在討論程式前，先大略說明 graphics write 的動作。

(a)高解析度

① 從游標表格中找出游標位置，然後算出游標所指的字元儲存格的起始位址。字元儲存格的起始列一定爲偶數列，每一個字元儲存格內有 4 列偶數列，在演算時可以只算出偶數列的間距值，這是因爲相鄰奇數列和偶數列位址相差 2000 H。

② 根據ASCII 碼決定字形表格是用BIOS 內的還是使用者自己建立的。

③ 根據ASCII 碼，從字形表格取出字形資料，填到字元儲存格。

(b)中解析度

① 和高解析度的步驟①一樣，找出游標的起始位址。

② 和高解析度的步驟②一樣，決定字形表格。

③ 從字形表格內取出掃描資料，將其 8 位元延伸成 16 位元，這是因爲 2 個位元決定一點，8 位元延伸成 16 位元是這樣的：

8 位元	01010101
16 位元	0011001100110011

④ 將顏色寫到前景（foreground）（1 表示前景，0 爲背景），如果顏色值爲 01（即選擇顏色組的第 1 個顏色），則由步驟③得到的 16 位元變成 0001000100010001 。

⑤ 將步驟④得到的 16 位元寫到特定的列數，雖爲 16 位元，但只構成 8 個點，前景即可顯示出所選的顏色。

⑥ 重複步驟③，直到填滿了字元儲存格內的 8 列。

好了，有以上的認識，討論程式就輕鬆多了。

① 先將 AH 暫存器清除爲 0，然後將 AX 暫存器內的值保存起來，其中 AL 內保存著字元。

② 呼叫 S26副程式算出游標所指的字元儲存格的起始位址，S26 算出的位址在 AX 暫存器內。返回後，將此值轉到 DI 暫存器內，此時 ES：DI 指著字元儲存格的起始記憶體位址。

③ 此步驟決定用那一個字形表格。首先取出字元值和 80 H 比較，若大於等於 80H，即表示要用使用者自行建立的字形

表格，跳入步驟⑤。若字元值小於80H，即表示要用BIOS內的字形表格，進入下個步驟。

④ BIOS內的字形表格從位址F000:FA6E 起，所以這裡將FA6EH放到SI內。然後將CS值（F000H）推到堆疊中，以便在步驟⑧中轉到DS去。準備好後，跳進步驟⑧。

⑤ 進入此步驟是因爲要選擇使用者自己建立的字形表格，先將字元值減去80H，以便利用此值，在步驟⑧中找出此字元值的字形資料相對於字形表格的間距值。先保存DS值，因爲在取EXT_PTR（中斷向量（ interrupt vector ）爲1FH）時，DS要指向另一個資料表格。然後設定DS值爲0，以便指向中斷表。

⑥ LDS SI,EXT_PTR將中斷向量1FH內的分段值放到DS內，並將間距值放到SI內，如此DS:SI即指向字形表格的起始位址。然後將DS值保存在DX 內，這是因爲在步驟⑧中，DS值必須爲0040H才能取出CRT_MODE值。

⑦ 取出DS的初值（ 0040H ），然後將DX內的值推到堆疊中，以便在下個步驟中轉到DS去。

⑧ 先將AX內的值左移3次（即乘以8），這是因爲每個字元的字形資料均爲8個位元組，然後將此值加到SI內。在取出字形表格的分段值後（ POP DS ），DS:SI指向此字元的字形資料。比較CRT_MODE值是否爲6，若爲6，則表示爲高解析度，進入下個步驟。若CRT_MODE小於6（ CF旗號爲1 ），則表示爲中解析度，跳入步驟⑮去處理。

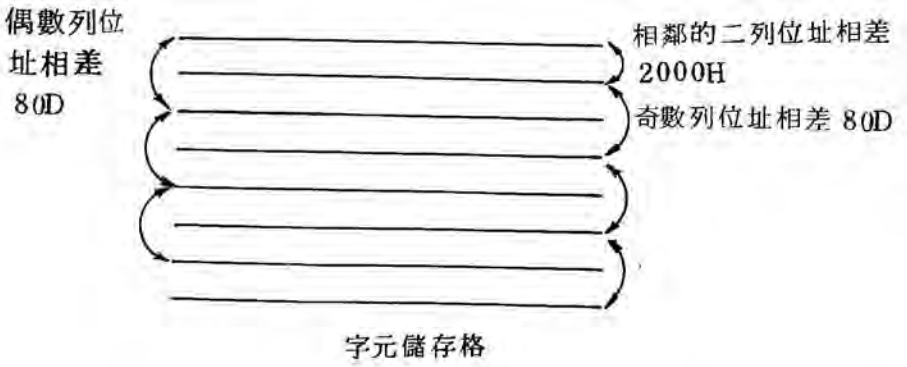
⑨ 進入此步驟是因爲高解析度。DS:SI指著字形表格，

ES:DI指著字元儲存格的起始位址，先將SI和DI的值保存起來。然後在DH內放入4，這是因爲每次都填二個字形資料，8列需要填4次。

⑩ LODSB取出字形資料到AL內，然後測試顏色屬性（在BL內）的位元7值是否爲1，若爲1（ ZF旗號爲0），，則表示在填入字形資料前需要和原來的值互斥（ XOR），跳入步驟⑭。

⑪ STOSB將AL內的值放到ES:DI所指的位址，然後再取出下一個字形資料到AL內。（ STOSB和LODSB 執行完後會分別將DI和SI內的值加1 ）

⑫ 在步驟⑪中是填字元儲存格內的偶數列的字形資料，此步驟要填在其下一列（奇數列）的字形資料。所以要將DI加上2000H後又減1，是因爲步驟⑪中的STOSB 指令已將DI的值加1，並且相鄰的二列其位址相差2000H。填完奇數到的字形資料後，將DI值加上79（ DI值已在步驟⑪中加1了），以便指向下一個偶數列，請看下圖。



將DH內的值減1，然後測試是否已填完了8列，若尚未完成工作（ ZF 旗號爲0 ），則回到步驟⑩。若已完成工

作（ZF旗號爲1），進入下個步驟。

⑬ 取回SI和DI的初值，然後將DI內的值加1，以便指向下一個字元儲存格。進入此程式的CX值指出要填若干字元儲存格。這裡的LOOP指令就是配合CX值，最後經由VIDEO_RETURN返回到呼叫INT 10H的程式去。

⑭ 進入此步驟是因爲顏色屬性的位元7值爲1。將字形資料（在AL內）和原來的字形資料（由ES:DI指著）互斥，然後將結果放回。LODSB取出下一個字形資料到AL內，同樣地做一次互斥後，跳到步驟⑫去填回原來的地方。

⑮ 進入此步驟是因爲中解析度。先將顏色屬性（在BL內）保存到DL內。在步驟②中呼叫S26副程式來算出游標所指的字元儲存格的起始位址。S26副程式是這樣算的：CRT_COLS內的值乘以列數，再乘以4，然後再加上行數。這樣的算法在中解析度中，並不能找到游標所指的字元儲存格的起始位址，因爲S26少乘以2，CRT_COLS內的值爲40，但實際上一列有80個位元組，而行數也要乘2，因爲螢幕上的8點需要2個位元組來組成。詳細情形請看下面討論的S26副程式。所以這裡要將DI乘以2（左移一個位元即乘以2），如此ES:DI才會指向字元儲存格的第一個起始位址，然後呼叫S19副程式，將顏色屬性值延展開來，如果顏色屬性值爲01（即要選擇顏色組的第一個顏色），則S19副程式會將BX暫存器內的值延展成0101010101010101B，以便顏色可以涵蓋到每個點（二個位元）。

⑯ 將DI和SI的值保存起來，並在DH內放入4，理由和高解析度相同。

⑰ LODSB指令取出字形資料到AL內，然後呼叫S21副程式將8位元延展成16位元。如果字形資料是A5H，那麼S21副程式會將其延展成1100110000110011B。

AL（8位元） 10100101

↓

經過S21副程式

AX（16位元） 1100110000110011

做這樣的轉換是因爲在中解析度中，二個位元構成一個點，爲了不改變字形需要這樣的轉換。執行到這裡，AX內的值指出字形，BX內的值爲顏色。爲了將顏色寫到前景，我們需要AND AX,BX。以上個例子而言，結果爲0100010000010001B。在顯示時，原本字形中的前景（邏輯1）會顯出我們所選擇的顏色。

⑱ 測試顏色屬性（在DL內）的位元7值是否爲1，若不爲1（ZF旗號爲1），則進入下個步驟。若爲1（ZF旗號爲0）。則表示字形資料要先和原來的字形資料互斥後，才填到字元儲存格。我們說過字形資料的高位元（high bit）是在螢幕畫面的左方，所以這裡是將步驟⑰得到的字形資料的高位元組（在AH內）和ES:DI所指的低位元組互斥，並將字形資料的低位元組和ES:DI所指的高位元組互斥。

⑲ 將字形資料的高位元組放到ES:DI所指的低位元組並將字形資料的低位元組放到ES:DI所指的高位元組。然後使用LODSB指令取出下一個字形資料，並且呼叫S21副程式將其延展成16位元，並將顏色寫上其前景。

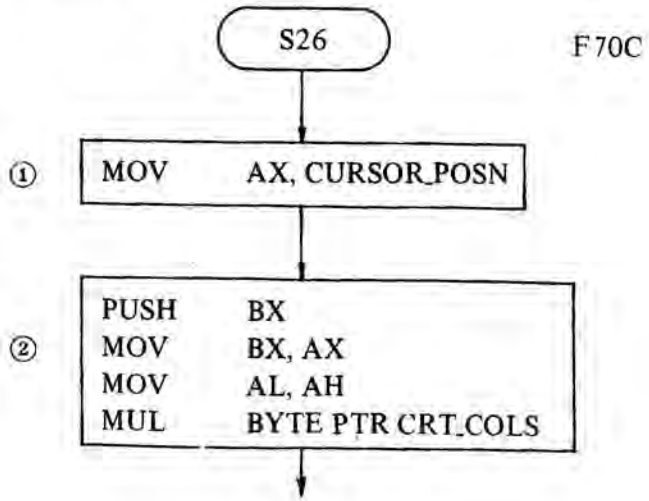
⑳ 步驟⑲是填偶數列的字形資料，本步驟和下個步驟是填奇數列的字形資料。和步驟⑱相同，此步驟先測試是否需要

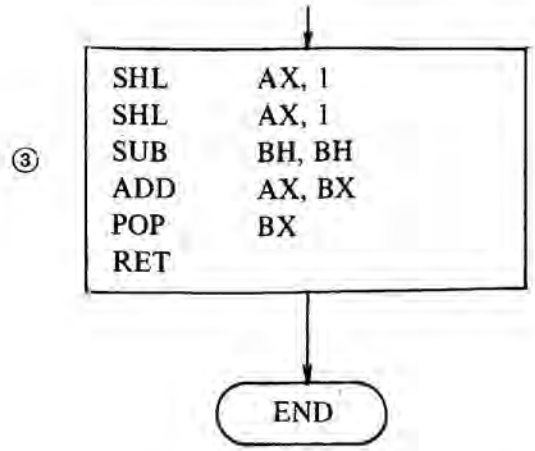
互斥，因爲偶數列和奇數列位址相差2000H，所以這裡需要加上2000H。

㉑ 將字形資料的高位元組放到奇數列位址的低位元組，將字形資料的低位元組放到奇數列位址的高位元組。放完奇數列的字形資料後，將DI內的值加上80，以便指向下一個偶數列，然後將DH內的值減1，測試是否已填完8列，若尚未完成工作（ZF旗號爲0），跳回步驟⑰繼續工作，若已完成工作（ZF旗號爲1），進入下個步驟。

㉒ 取回SI和DI的初值後，將DI內的值加2，以便指向下一個字元儲存格。進入此程式的CX內的值指出要連續填若干字元儲存格，這裡的LOOP指令就是配合CX值。最後經由VIDEO_RETURN返回到呼叫INT 10H的副程式去。

若讀者對graphics_write副程式仍有疑問，請別心急，因爲我們尚未討論算出位址的S26副程式，將顏色值延展開的S19副程式和將字形資料由8位元延展成16位元的S21副程式。以下我們先討論S26副程式。





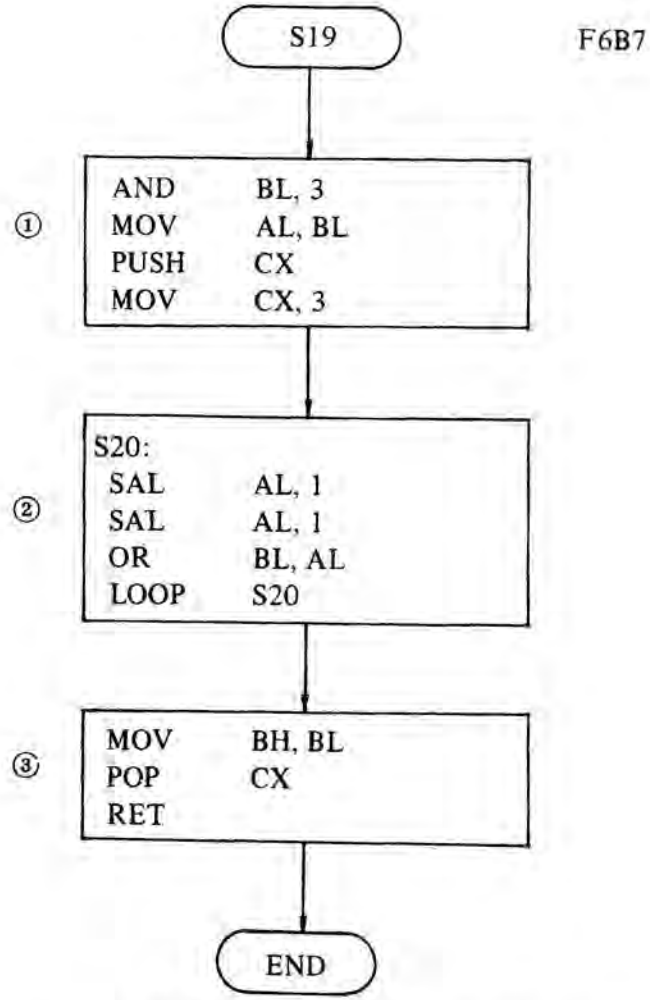
① 從游標位置表格內取出游標所在的位址，這個值是以字元儲存格爲單位。

② 此步驟是GRAPH_POSN的進入點，有的程式不是算游標的位址，而是算指定的位址，只要將指定的位址放到AX暫存器內即可。先將BX暫存器保存起來，並將AX內的值轉到BX內。MOV AL,AH指令是將列數放到AL內，接下來的MUL指令將列數乘以每列有若干字元（CRT_COLS），結果在AX內。

③ 將步驟②得到的結果乘以4（左移2個位元即乘以4），這是因爲每個字元儲存格內有4條偶數列，不乘以8是因爲奇數列的位址是由偶數列的位址加上2000H得來。在步驟②已將AX內的值轉到BX內了。這裡先將BH暫存器內的值清除爲0，然後將BX暫存器內的值（行數）加到AX暫存器內，所以AX暫存器內即放著游標所指的字元儲存格的起始位址（間距值，相對於Regen buffer）。值得注意的是，此值對高解析度是正確的，但對中解析度是不正確的，因爲中解析度的CRT_COLS爲40，但實際上

一列有 80 個位元組，而且中解析度掃描出 8 個點需要 2 個位元組，所以在中解析度時，尚要將這裡得到的 AX 值乘以 2 。最後取出 BX 暫存器的初值後返回。

接下來看將顏色延展開的 S19 副程式。



我們先建立一些概念，顏色屬性值有 4 種可能值：0 ，1 ，2 ，3 ，這 4 個值最多佔用二個位元。因爲顏色必須涵蓋到每一個點，所以要將顏色值拷貝到每一位元對（ BIT PAIR ）上。如果顏色值爲 01（ 在 BL 內 ），經過此程式的運算後，BX 內將成爲：0101010101010101 。

① AND BL，3 取出顏色值（ 在BL 內的位元 1 和位元 0 ）。然後先轉到 AL 內。保存 CX 值是因爲下個步驟要使用 LOOP 指令。在CX 內放入 3 ，表示要 LOOP 3 次。

② 二個 SAL AL，1 指令將顏色值轉到上一個位元對。OR BL，AL 指令將其保存到 BL 內；LOOP 3 次是因爲一個位元組有 4 個位元對。請看下面的例子：

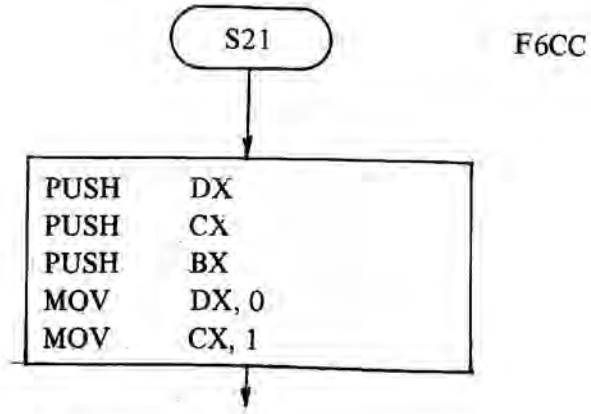
(a) 如果步驟①得到的 AL 值爲 000000AB 。

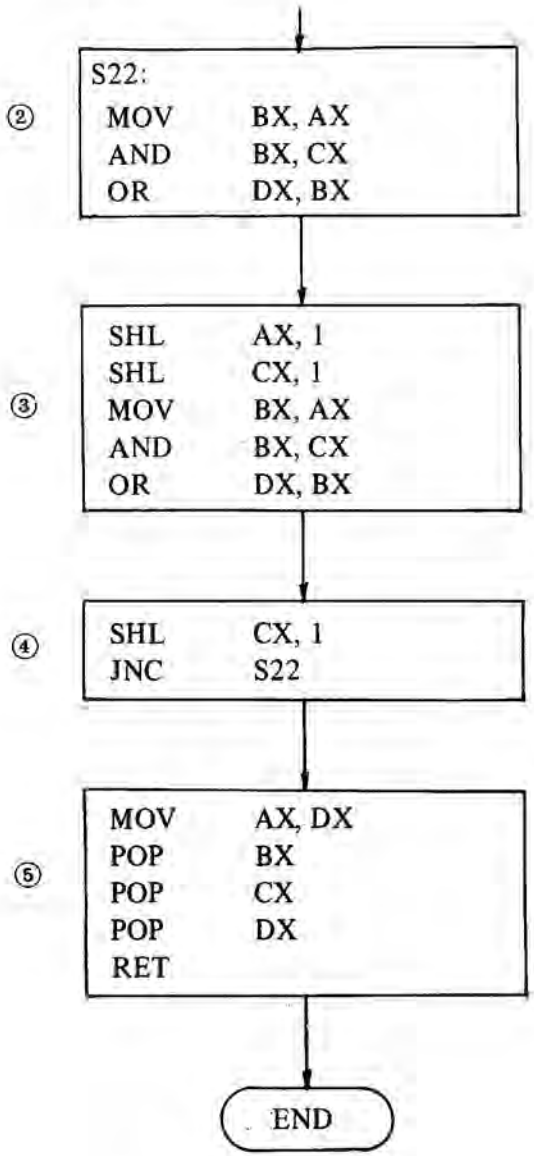
(b) 二個 SAL AL，1 指令將其移上二個位元，即 AL 內的值爲 0000AB00 。然後 OR 指令將其轉到 BL內，則 BL 內的值爲 0000ABAB。

(c) 如此繼續將位元 7 到位元 4 都填上顏色值，從 LOOP 出來的 BL 內的值爲 ABABABAB。

③ 將 BL 內的值拷貝一份到 BH 內，則 BX 內的值爲：ABABABABABABABAB ，在取回 CX 暫存器的初值後返回。

我們現在討論比較複雜的 S 21 副程式。





這個程式將8位元的字形資料轉換成16 位元的字形資料，這是因爲在中解析度中，二個位元才構成1點。如果字形資料爲ABCDEFGH，那麼此程式將其轉換成AABBCCDDEEFFGGHH，轉換的步驟是這樣的：

(a) 取出LSB，0000000H，放到結果暫存器內。

(b) 左移字形資料一個位元，BCDEFGH0。

(c) 取出H，並放到結果暫存器內，則結果暫存器爲0000000000000HH。

(d) 重複8次，直到完成整個工作。

我們現在就來看程式是如何寫的：

① 依序保存DX，CX和BX暫存器的初值，然後將結果暫存器清除爲0（DX），並在遮罩暫存器（CX）內放入1，以便取出位元。

② 先將AX暫存器內的值（AH爲00H，AL爲字形資料）轉到BX內，以免被破壞。AND BX，CX 指令取出某一個位元（例如：LSB 位元0），然後保存到結果暫存器內。

③ 將 AX內的值左移一個位元（例如位元0左移到位元1）；將CX內的值也左移一個位元（例如位元0左移到位元1），將CX內的值也左移一個位元，以便取到同樣的位元。同樣的，將AX內的值轉到BX內，以免被破壞。AND BX，CX指令取到相同的位元，但位置已在左邊一個位元。ORDX，BX指令將其保存到DX內（如果BIT0爲A，則此時DX內的值爲000000000000AA）。

④ 如果步驟②和③重複做8次，則表示已完成了工作，這裡的SHL CX，1指令會使CF旗號爲1，進入下個步驟。如果尚未完成工作，SHL CX，1使得CF旗號爲0，並在下次取左一個位元（例如：上次取位元0，這裡返回到

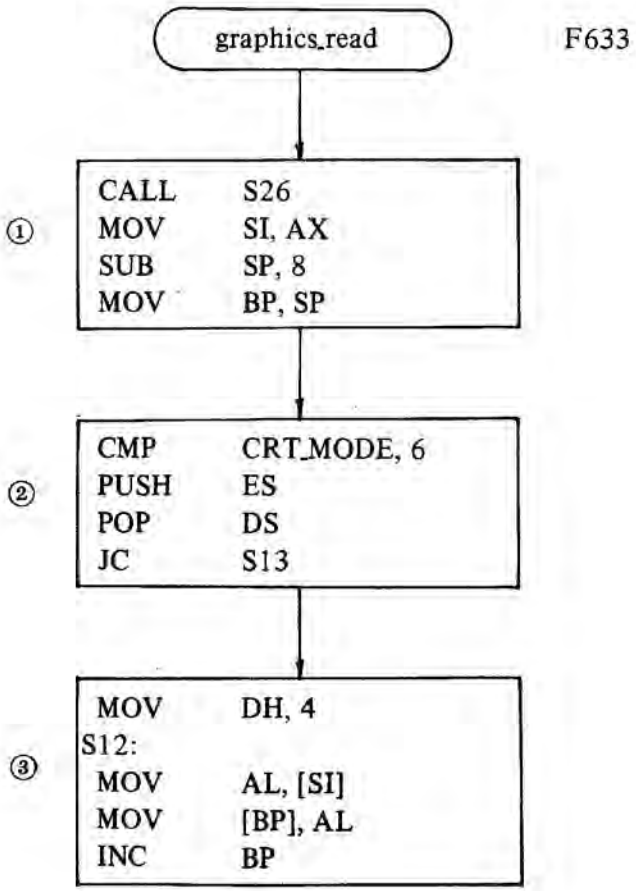
步驟②是取位元1）。這裡不能將AX值左移一個位元，否則會取到相同的位元。

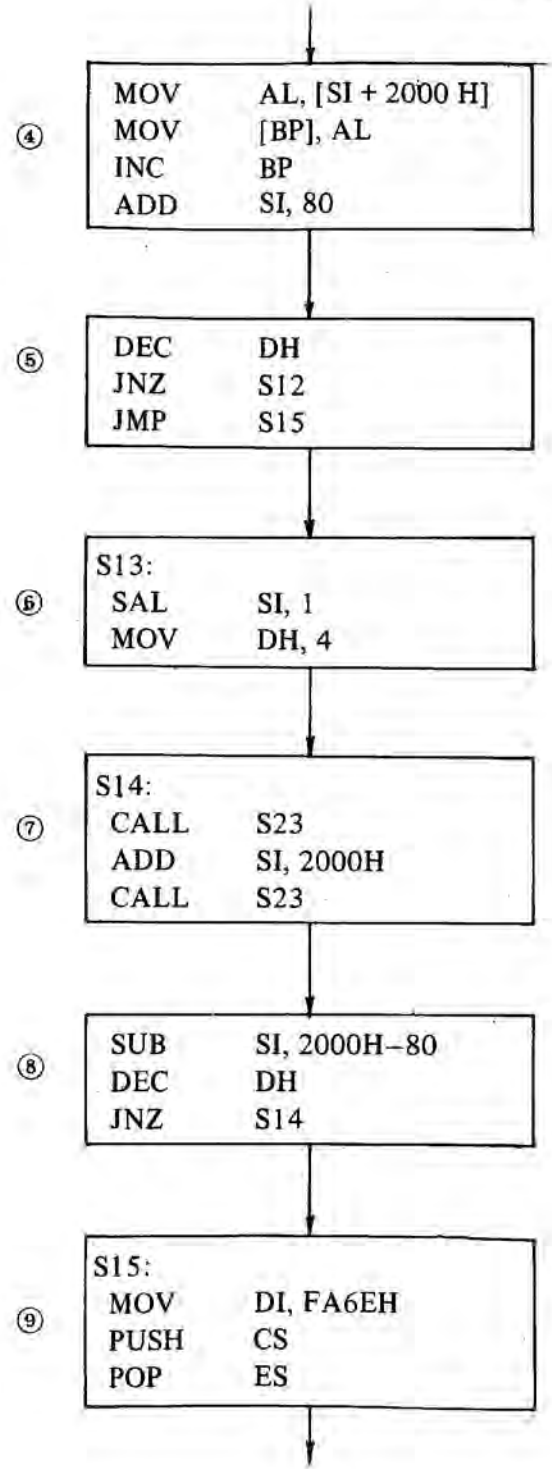
⑤ 將結果暫存器轉到AX暫存器，並陸續取出BX，CX和DX暫存器的初值後返回。

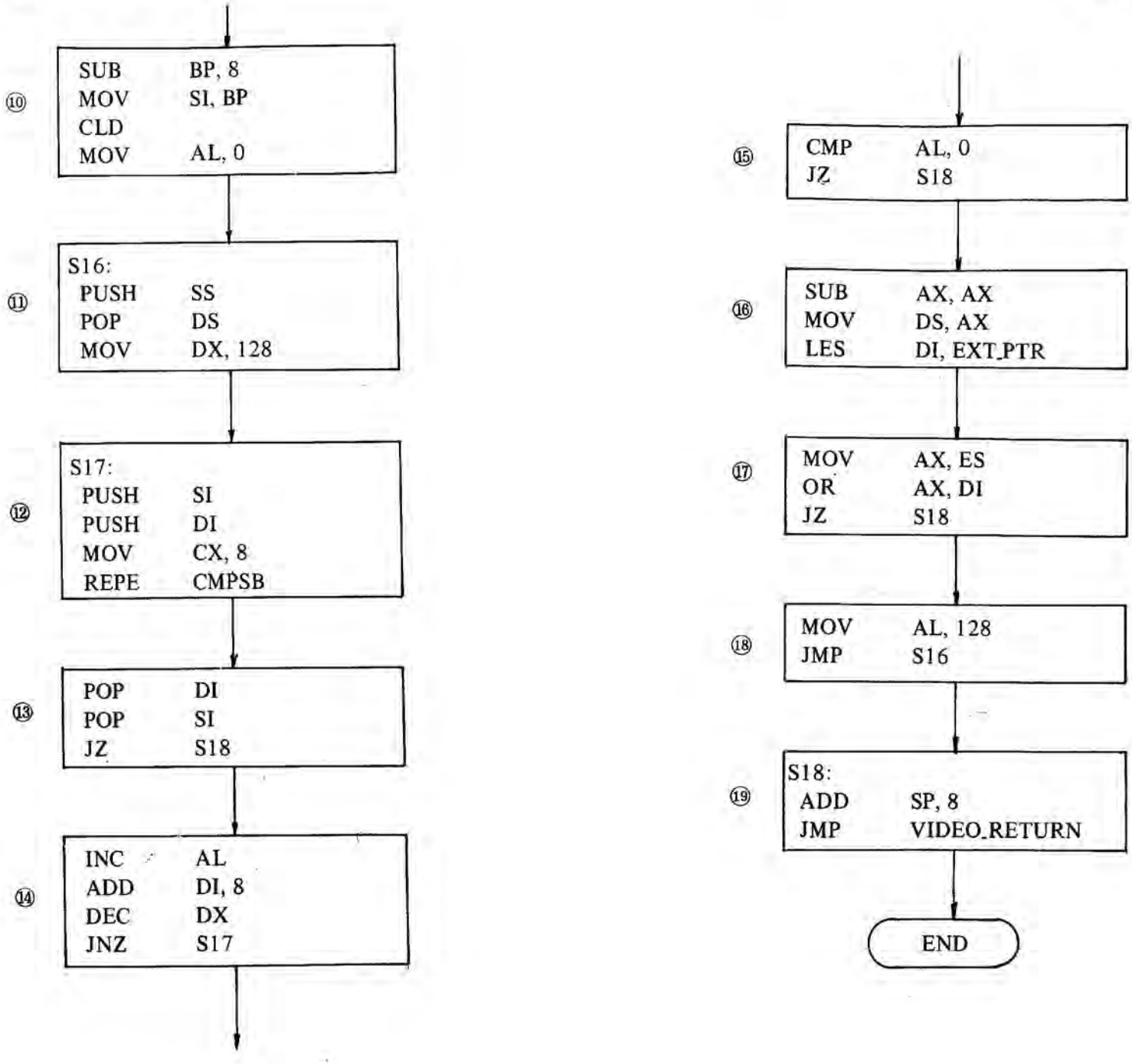
筆者希望讀者在了解S26，S19和S21 副程式之後，再一次TRACE graphics_write 程式，務必要明白graphics_write程式，因爲graphics_read 程式與其息息相關。

（n） graphics_read

Output : (AL) = character read






⑩
SUB BP, 8
MOV SI, BP
CLD
MOV AL, 0
⑪
S16:
PUSH SS
POP DS
MOV DX, 128
⑫
S17:
PUSH SI
PUSH DI
MOV CX, 8
REPE CMPSB
⑬
POP DI
POP SI
JZ S18
⑭
INC AL
ADD DI, 8
DEC DX
JNZ S17
⑮
CMP AL, 0
JZ S18
⑯
SUB AX, AX
MOV DS, AX
LES DI, EXT_PTR
⑰
MOV AX, ES
OR AX, DI
JZ S18
⑱
MOV AL, 128
JMP S16
⑲
S18:
ADD SP, 8
JMP VIDEO_RETURN
END

我們先建立一些概念。我們已經知道在繪圖中顯示一個字元，需要將字形表格內的字形資料填到字元儲存格內。現在我們要知道字元儲存格內的ASCII碼，則必須將字元儲存格內的字形資料取出和字形表格內的字形資料比較，若8個位元組都相等，才算找到這個字元的ASCII碼。graphics_read的做法是這樣的，先將字元儲存格內的8個字形資料取出放到堆疊中，然後將8個位元組和BIOS的字形表格內的字形資料比較，若沒找到才和使用者自己建立的字形表格內的字形資料比較。尋找的方式是從字形表格的最起頭比較，每次8個位元組，每次比較都會先在AL內放入此8個字形資料所相對的ASCII碼，若比較相同的話，此字元儲存格內的字元的ASCII碼即在AL內。

① 首先呼叫S26副程式算出游標所指的字元儲存格的起始位址，返回後將結果轉到SI內。將SP暫存器減去8，以便保留8個位元組來儲存字形資料，並將SP內的值轉到BP內（除非特別指定，否則BP也是使用SS段落）。

② 比較CRT_MODE內的值是否爲6，若爲6（CF旗號爲0），則表示爲高解析度，進入下個步驟。若小於6（CF旗號爲1），則表示爲中解析度，跳入步驟⑥。PUSH ES和POP DS指令使得DS也指著Regen buffer。

③ 在DH內放入4，是因爲每個字元儲存格有4條偶數列和4條奇數列。首先取出字元儲存格（偶數列的字形資料，並保存在BP所指的堆疊位址內。然後將BP內的值加1，以便指向下一個堆疊位址。

④ MOV AL,〔SI+2000H〕指令取出字元儲存格內的奇數列的字形資料，並保存到堆疊中。同樣的將BP內的值加1，以便指向下一個堆疊位址。將SI內的值加上80，是爲了要指向下一條偶數列。

⑤ 將DH內的值減1，測試是否已做完工作，若尚未完成工作（ZF旗號爲0），返回到步驟③。若已完成了工作（ZF旗號爲1），跳入步驟⑨。

⑥ 進入此步驟是因爲中解析度。先將SI內的值乘以2，這是因爲S26副程式少乘以2，然後也在DH內放入4，理由和步驟③相同。

⑦ 呼叫S23副程式，將16位元的字形資料轉成8位元的字形資料，並保存在堆疊中，S23副程式會將BP內的值加1，以便指向下一個堆疊位址，然後將SI內的值加上2000H，以便指向奇數列。同樣的呼叫S23副程式取出字形資料到堆疊中。

⑧ SUB SI,2000H−80使得DS:SI指向下一條偶數列。然後將DH內的值減1，測試是否已做完工作，若尚未完成工作（ZF旗號爲0），則跳回步驟⑦繼續工作。若已完成工作（ZF旗號爲1），進入下個步驟。

⑨ 此步驟將ES:DI指向BIOS內的字形表格。

⑩ 將BP內的值減8，以便指向堆疊內的字形資料的起始位址，並將此值轉到SI暫存器內。清除DF旗號，以便配合步驟⑫中的CMPSB指令。將00H放到AL內，是因爲要要從字形表格的起始位址比較起，而第1組字形資料（8個位元組）所相對的ASCII碼爲00H。

⑪ 將SS內的值轉到DS內，使得DS:SI指著堆疊內的字形資料。然後在DX內放入128，這是因爲每個字形表格內有128組字形資料（每組8個位元組）。

⑫ 首先保存SI和DI暫存器的初值。在CX內放入8是因爲

每組字形資料有8個位元組。REPE CMPSB 指令做比較的動作，只要碰到不相同的位元組，即停止繼續比較下去。請讀者自行參考REPE和CMPSB指令。

⑬ 取回DI和SI的初值，然後測試ZF旗號，若ZF旗號爲0，則表示步驟⑫中比較8個位元組都相等，既然找到結果（ASCII碼在AL內），跳進步驟⑲。若ZF旗號爲0，則表示步驟⑫中的比較有不相等的情形產生，進入下個步驟做下一組字形資料的比較的準備工作。

⑭ 既然要比較下一組字形資料，所以也要將AL內的值加1，以便預先設定這一組字形資料的ASCII碼。將DI內的值加8，以便指向下一組字形資料。將DX內的值減1，以便檢查是否已比較完128組字形資料。若尚未完成工作（ZF旗號爲0），跳回步驟⑫繼續做比較的動作。若已比較完128組字形資料，仍尚未找到相同的字形資料，則進入下個步驟。

⑮ 此步驟的目的是測試是否已使用過使用者自己建立的字形表格。若只使用過BIOS的字形表格，進入此步驟的AL內的值爲128，當然不等於0，進入下個步驟。若連使用者自己建立的字形表格都比較過了，進入此步驟的AL內的值爲0（因爲每個字形表格有128組字形資料，二個字形表格共有256組，256這個值對8位元暫存器而言爲00H，但CF旗號爲1）。既然二個字形表格內的字形資料都不和堆疊內的字形資料相同，那就得返回到呼叫INT 10 H的程式，跳入步驟⑲。

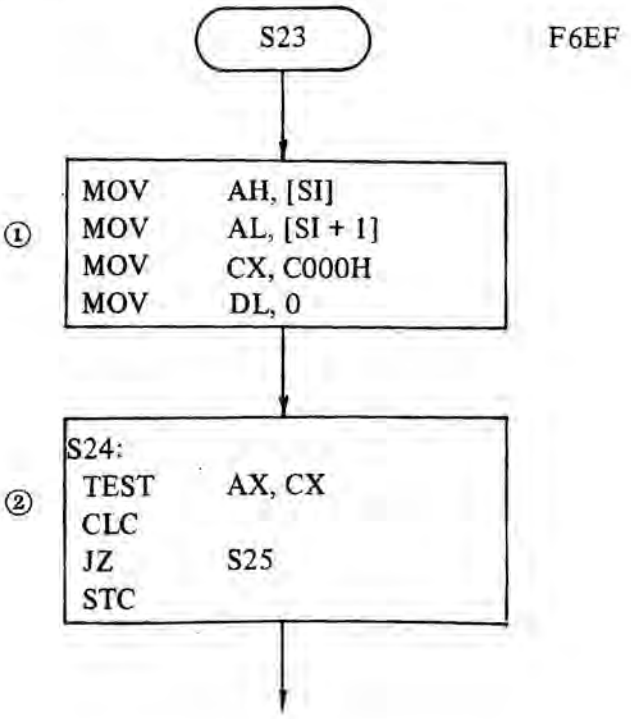
⑯ 此步驟取出使用者自己建立的字形表格的起始位址到ES：DI內。因爲EXT_PTR在分段值爲0000H的記憶體範圍內，所以要設定DS內的值爲0000H。

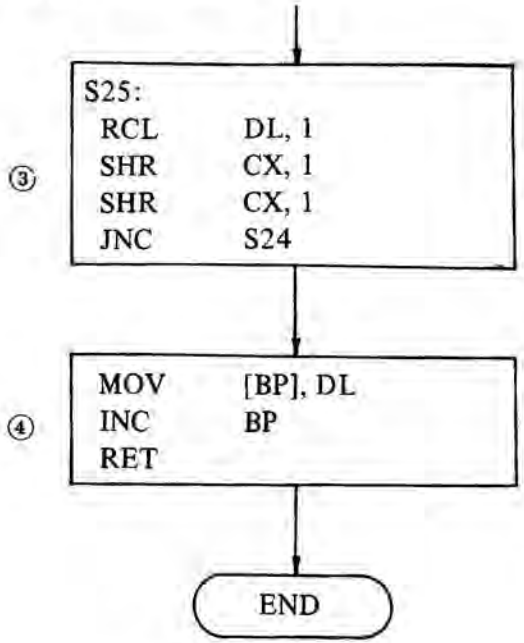
⑰ 此步驟測試是否有使用者自己建立的字形表格存在。若步驟⑯中得到的ES和DI值都爲0，則表示使用若並沒有建立字形表格（ZF旗號爲1），跳入步驟⑲。否則進入下個步驟。

⑱ 在AL內放入128，是因爲使用者自己建立的字形表格，其ASCII碼由128開始。然後跳入步驟⑪去做比較的動作。

⑲ 將SP內的值加8，指回原來的位址後，經由VIDEO_RETURN返回到呼叫INT 10 H的程式去。

在了解graphics_write後，再來看graphics_read是相當輕鬆的，只有在中解析度中，S23副程式是比較傷腦筋的，我們現在來討論S23副程式。





我們先建立一些概念。其實S23副程式是S21副程式的反動作。S21副程式將8位元的字形資料轉換成16位元，而S23副程式是將 16 位元的字形資料轉換成8位元。如果 16 位元的字形資料是AABBCCDDEEFFGGHH，那S23副程式將其轉換成ABCDEFGH。轉換的步驟是這樣的：

(a) 取出16位元字形資料的最高2個位元AA 0000000000000。

(b) 測試其爲1或0，若爲1，設定CF旗號爲1，若爲0設定CF旗號爲0。

(c) 利用ROL指令，將CF旗號的值左旋入特定的8位元暫存器（DL）。如上例，DL內的值爲0000000A。

(d) 回到步驟(a)取出位於AA右邊的位元對（bit pair），並重複步驟(b)和(c)，則DL內的值爲000000AB。重複做步驟(a)，(b)和(c)8次即可得到8位元的字形資料。

我們現在來看S23是怎麼寫的。

① 先前的二個MOV指令取出DS：SI所指的字形資料到AX暫存器內。在CX內放入C000H是爲了取出位元對，並在結果暫存器（DL）內放入00H。

② TEST AX,CX指令測試CX所取出的位元對的值（注意：TEST指令不改變AX和CX內的值），若CX所取出的位元對值爲0，則設定CF旗號爲0，若位元對值爲1（ZF旗號爲0），則設定CF旗號爲1。

③ RCL DL,1指令將CF旗號內的值左旋入DL內。然後將CX內的值右移二個位元，以便取出下一對位元對。若步驟②和③已重複做過8次，則這裡的CF旗號會爲1，表示已取出8位元的字形資料於DL內，進入下個步驟。若這裡的SHR CX,1指令仍不能使CF旗號爲1，則表示尚未完成工作，跳回步驟②，繼續工作。

④ 將8位元的字形資料放到BP所指的堆疊位址後，將BP內的值加1，以便指向下一個堆疊位址。最後經由RET指令返回到呼叫S23副程式的程式去。

討論完graphics_write和graphics_read後，相信讀者對繪圖中螢幕畫面和Regen buffer內的記憶體的關係已經清楚了，那麼我們在討論繪圖上捲（scroll up）和下捲（scroll down）時，將會比較輕鬆。我們先討論graphics scroll up副程式。

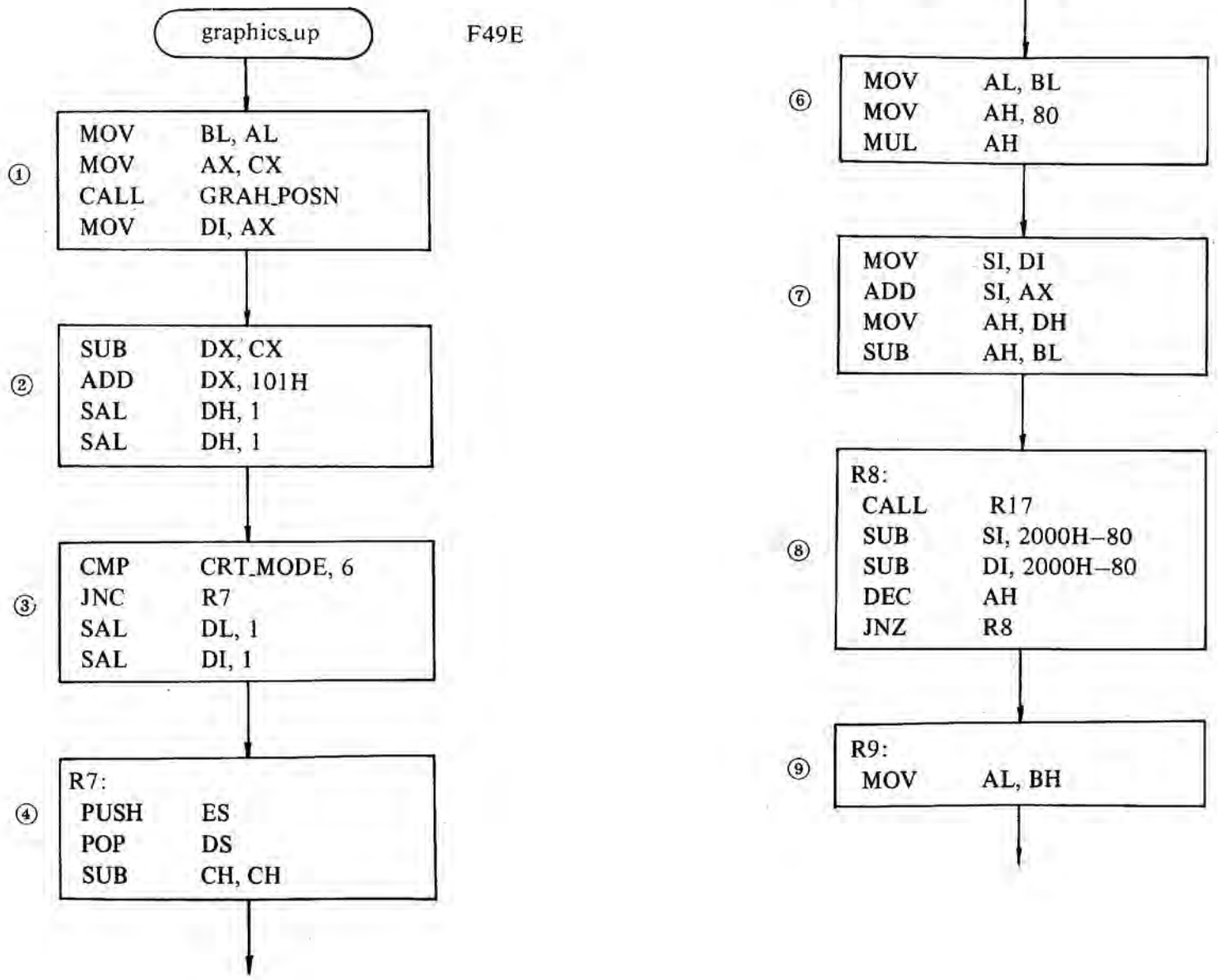
(o) graphics_up

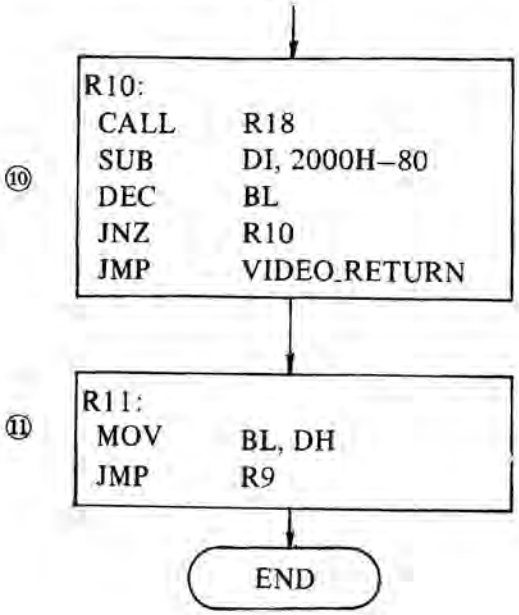
```
This routine scrolls up the information on the CRT
Input :  (CH, CL) = upper left corner of region to scroll
         (DH, DL) = lower right corner of region to scroll
         (BH) = fill value for blanked lines
```

(AL) = number of lines to scroll (AL=0 means
blank the entire field)

(DS) = data segment

(ES) = Regen segment





其實 graphics_up 和文字（Alphabetic）中的上捲動作是相同的，只不過是繪圖中，每個字元儲存格是由 8 個位元組（高解析度）或 16 個位元組（中解析度）所組成，所以搬動的次數多出很多，也較麻煩。讀者只要記住繪圖中每個字元儲存格內有 8 列—4 列偶數列和 4 列奇數列，而相鄰的偶數列和奇數列位址相差 2000H（我們姑且稱它們爲綫對（line pair）），每次的搬移動作都是以綫對爲單位。若讀者仍對文字中的上捲和 graphics_write 副程式不了解的話，筆者建議您再複習一次，因爲 graphics_up 將用到文字的上捲觀念和 graphics_write 中螢幕畫面和 Regen buffer 內的記憶體的關係。我們在討論 graphics_up 副程式時將不再舉例。

① 先將要向上捲動的列數（在AL內，單位爲字元儲存格）保存在BL內，然後將左上角的座標（在CX內，單位爲字元儲存格轉到AX內，並呼叫GRAPH_POSN副程式算出此字元儲存格的起始位址。返回後將結果轉到DI內（GRAPH_POSN副程式爲S26的步驟②）。

② SUB DX,CX和ADD DX,101H 指令算出要捲動的視窗（window）內共有若干列和行，單位爲字元儲存格。列數在DH內，行數在DL內。然後將DH內的值乘以4，這是因爲每個字元儲存格內有4條線對。

③ 此步驟檢查CRT_MODE是否爲6，若爲6（CF旗號爲0），表示爲高解析度，以上的演算都是正確的，直接跳入步驟④。若CRT_MODE不爲6（CF旗號爲1），則表示爲中解析度，則尚需將行數（在DL內）和由步驟①算出的位址乘以2，才進入步驟④。

④ 前二個指令將DS也指向Regen buffer，然後將CH清除爲0，以便配合R17和R18中的REP指令。

⑤ 在步驟①中已將要往上捲動的列數保存在BL內（單位爲字元儲存格），同樣的也要乘以4，若結果爲0（ZF旗號爲1），表示要使整個視窗空白，跳入步驟⑪。若結果不爲0，進入下個步驟。

⑥ 這個步驟將步驟⑤得到的結果乘以80，這是因爲每列有80個位元組，以便在下個步驟中找出搬動時的來源（source）。

⑦ 首先將DI內的值轉到SI內。如此ES:DI和DS:SI指著相同的位址（即視窗的左上角座標的起始位址）。然後將步驟⑥得到的值加到SI內，如此DS:SI指著搬動時的來源位址，ES:DI指著搬動時的目的位址。此二個位址相差（要上捲的字元儲存格的列數）乘以4再乘以80，接著的二個指令算出總共要搬動的線對的數目於AH內。

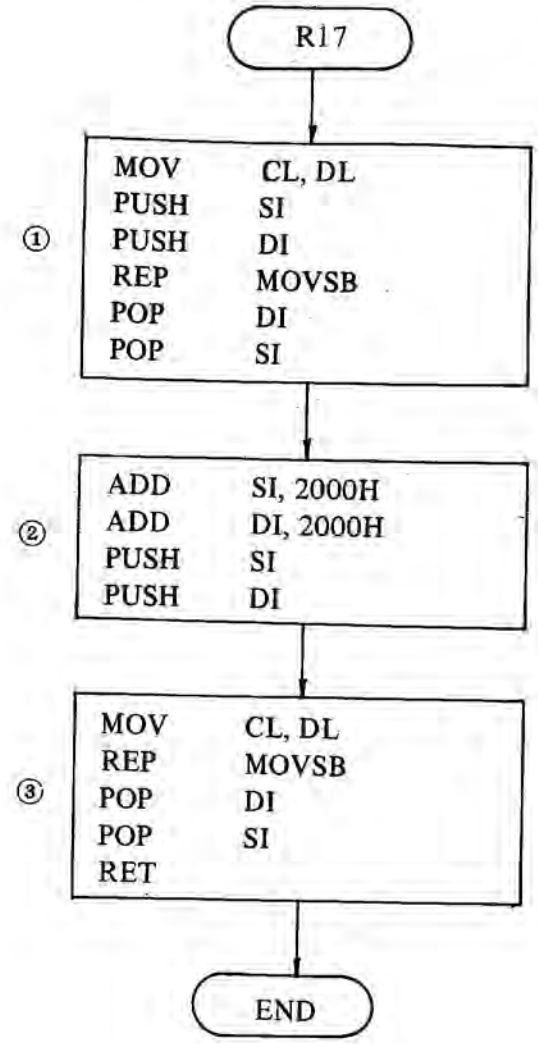
⑧ 首先呼叫R17副程式搬動一條線對，返回後，同時將DI和SI內的值減去（2000H－80），以便同時指向下一條線對。然後將AH內的值減1，測試是否已做完整個搬動工作，若尚未做完（ZF旗號爲0），則仍需繼續搬動；若已做完整個搬動工作，則進入下個步驟。此時視窗下方將會空出（要上捲的字元儲存格列數）乘以4的線對數（在BL內）。

⑨ 將顏色屬性值（在BH內）放到AL內。

⑩ 首先呼叫R18副程式將顏色屬性值填到線對去。返回後將DI內的值減去（2000H－80）以便指向下一條線對。然後將BL內的值減去1，測試是否已做完工作，若尚未完成（ZF旗號爲0），則仍需繼續填顏色屬性值。若已完成工作，則經由VIDEO_RETURN返回到呼叫INT 10H的程式去。

⑪ 進入此步驟是因爲要空白整個視窗，所以將視窗內全部的線對數（在DH內）放到BL內，然後跳到步驟⑨去做填顏色屬性值的工作。

在剛才討論的graphics_up程式中，我們可以看出graphics_up內都是一些計算和測試，眞正的搬移動作是由R17副程式完成，填顏色屬性值由R18副程式來做。我們現在就來看這二個核心程式。先討論R17。



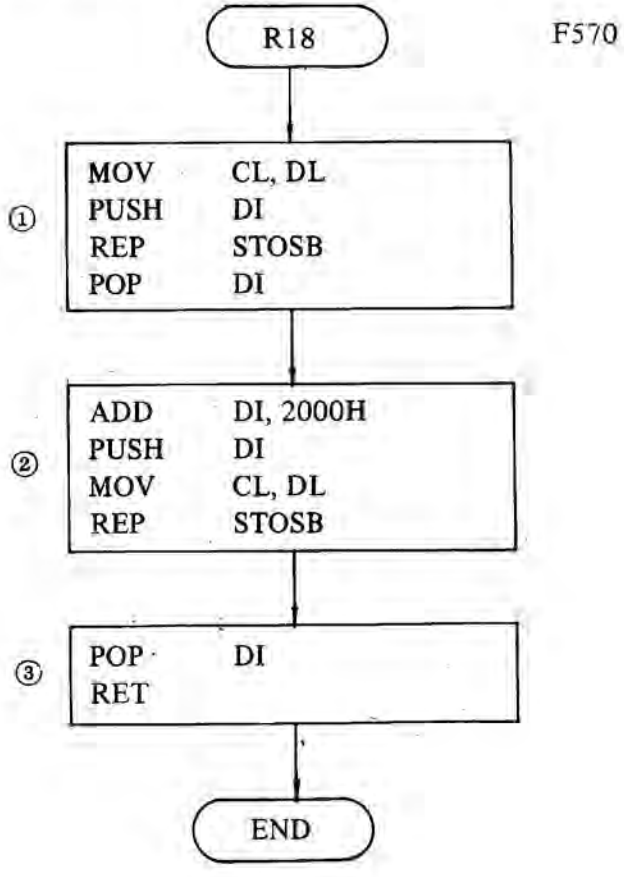
① 將視窗內的行數（在DL內）放到CL內，以便配合REP指令，然後將SI和DI內的值都保存起來。REP MOVSB做搬動的工作，將DS：SI所指的列內全部的資料搬到ES:DI所指的列（這裡所說的全部是指視窗內）。這條列是偶數列，步驟③是搬奇數列。最後取回DI和SI

的初值。

② 將SI和DI內的值都加上2000H，以便指向下一條奇數列，並將此SI和DI的值保存起來。

③ 將視窗內的行數（在DL內）放到CL內，以便配合REP指令。REP MOVSB將DS:SI所指的奇數列內全部的資料搬到ES:DI所指的奇數列，如此即完成將線對向上搬的動作，在取回DI和SI的值後返回。

我們繼續討論R18副程式。



① 將視窗內的行數（在DL內）放到CL內，以便配合REP指令。先將DI內的值保存起來。REP STOSB指令將AL內的值放到ES:DI所指的偶數列內所有的位址去，AL內的值爲顏色屬性值，然後取出DI的值。

② 將DI內的值加上2000H，以便指向奇數列。同樣的將DI內的值保存起來，然後將視窗內的行數（在DL內）放到CL內，以便配合REP指令。REP STOSB指令將顏色屬性值（在AL內）填到ES:DI所指的奇數列所有的位址，如此即完成了將顏色屬性值填到線對的工作。

③ 取回DI的值後返回。

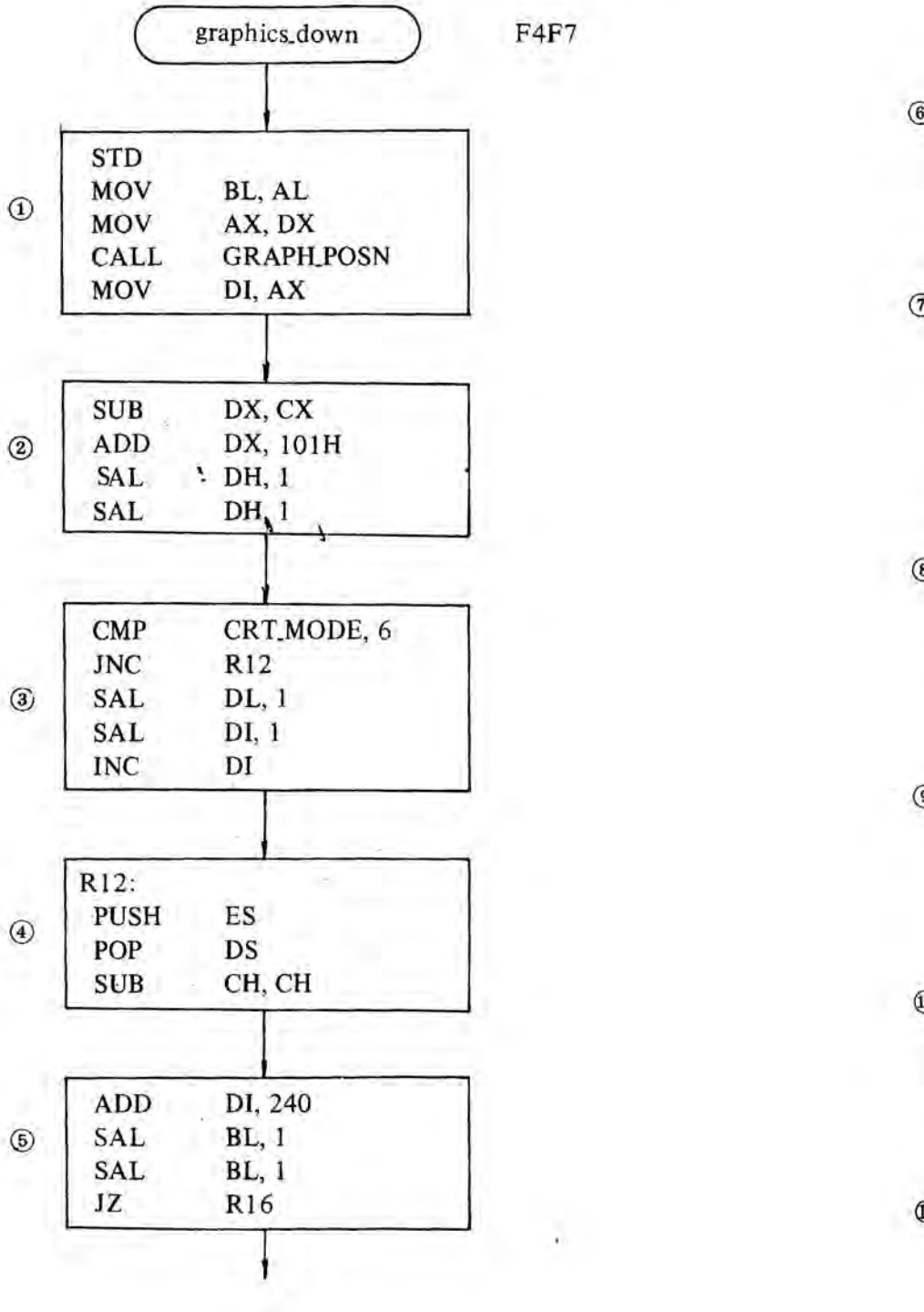
在了解文字的上捲和graphics_write之後，graphics_up是不難的；若讀者仍有疑問。只要稍爲對照scroll_up和graphics_write就能找到解答。

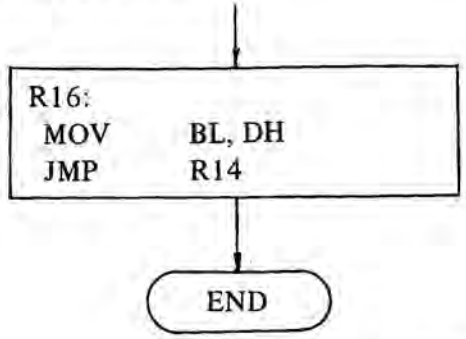
graphics-down是graphics-up的反動作，對讀者來說是容易的。

(p) graphics_down

This routine scroll down the information on the CRT

Input : (CH, CL) = upper left corner of Region to scroll
(DH, DL) = lower right corner of region to scroll
(BH) = color attribute
(AL) = number of lines to scroll
(DS) = Data segment
(ES) = Regen buffer


graphics_down
F4F7
①
STD
MOV BL, AL
MOV AX, DX
CALL GRAPH_POSN
MOV DI, AX
②
SUB DX, CX
ADD DX, 101H
SAL DH, 1
SAL DH, 1
③
CMP CRT_MODE, 6
JNC R12
SAL DL, 1
SAL DI, 1
INC DI
④
R12:
PUSH ES
POP DS
SUB CH, CH
⑤
ADD DI, 240
SAL BL, 1
SAL BL, 1
JZ R16
⑥
MOV AL, BL
MOV AH, 80
MUL AH
⑦
MOV SI, DI
SUB SI, AX
MOV AH, DH
SUB AH, BL
⑧
R13:
CALL R17
SUB SI, 2000H+80
SUB DI, 2000H+80
DEC AH
JNZ R13
⑨
R14:
MOV AL, BH
⑩
R15.
CALL R18
SUB DI, 2000H+80
DEC BL
JNZ R15
⑪
CLD
JMP VIDEO_RETURN



① graphics_down是graphics_up的反動作。首先設定DF旗號爲1，所以在R17和R18中的MOVSB和STOSB指令是將SI和DI內的值減1，然後將要下捲的列數（單位爲字元儲存格在AL內）轉到BL內。將視窗內的右下角的座標（在DX內，單位爲字元儲存格）轉到AX內，並呼叫GRAPH_POSN副程式算出此字元儲存格的起始位址，返回後，將結果轉到DI內（GRAPH_POSN副程式爲S26副程式的步驟②）。

② 此步驟算出視窗內的列數和行數（單位爲字元儲存格），列數在DH內，行數在DL內。然後將列數乘以4，這是因爲每個字元儲存格內有4條線對。

③ 此步驟先檢查CRT_MODE是否爲6，若爲6（CF旗號爲0）則表示爲高解析度，以上的演算都是正確的，直接跳入下個步驟。若CRT_MODE不爲6（CF旗號爲1），則表示爲中解析度，還得將行數（在DL內）和由GRAPH_POSN副程式算出的位址乘以2。因爲在中解析度中，二個位元構成一個點，所以還要將DI內的值加1，以便指向右邊4個點的字形資料的位址。

④ 此步驟將DS也指向Regen buffer，同時將CH內的值清除爲0，以便配合R17和R18中的REP指令。

⑤ 討論到這裡，ES:DI指向視窗右下角的字元儲存格的起始位址（中解析度指向右邊4個點的那個位元組）。請看下圖

因爲現在要下捲，而DF旗號爲1，所以還要將DI內的值加上240，以便指向此字元儲存格的列7的那個位元組。然後將要下捲的列數（單位爲字元儲存格，在BL內）乘以4，這是因爲每個字元儲存格內有4條線對，而R17和R18每次動作的對象都是線對。這裡還要測試BL內的值是否爲0，若爲0（ZF旗號爲1），則表示要空白整個視窗，跳入步驟⑫。若BL內的值不等於0，則進入下個步驟。

⑥ 此步驟將步驟⑤得到的結果乘以80，以便在搬動時能指向來源地。

⑦ MOV SI,DI使得ES:DI和DS:SI指著相同的位元組，然後SUB SI,AX指令使得DS:SI指著搬動時的來源地，這裡的最後二個指令算出總共要搬動若干線對，結果在AH內。

⑧ 首先呼叫R18副程式搬動一條線對（注意現在的方向爲後退）。返回後同時將DI和SI內的值減去2000H+80，以便指向上一條線對。然後將AH內的值減1，測試是否已完成工作，若尚未完成，則仍需工作。若已完成工作（

ZF旗號爲1），進入下個步驟去填顏色屬性值。此時視窗上方將會空出（要下捲的字元儲存格的列數）乘4的線對數（在BL內，在步驟⑤已算出）。

⑨ 將顏色屬性值（在BH內）轉到AL去，以便在R18中填到線對去。

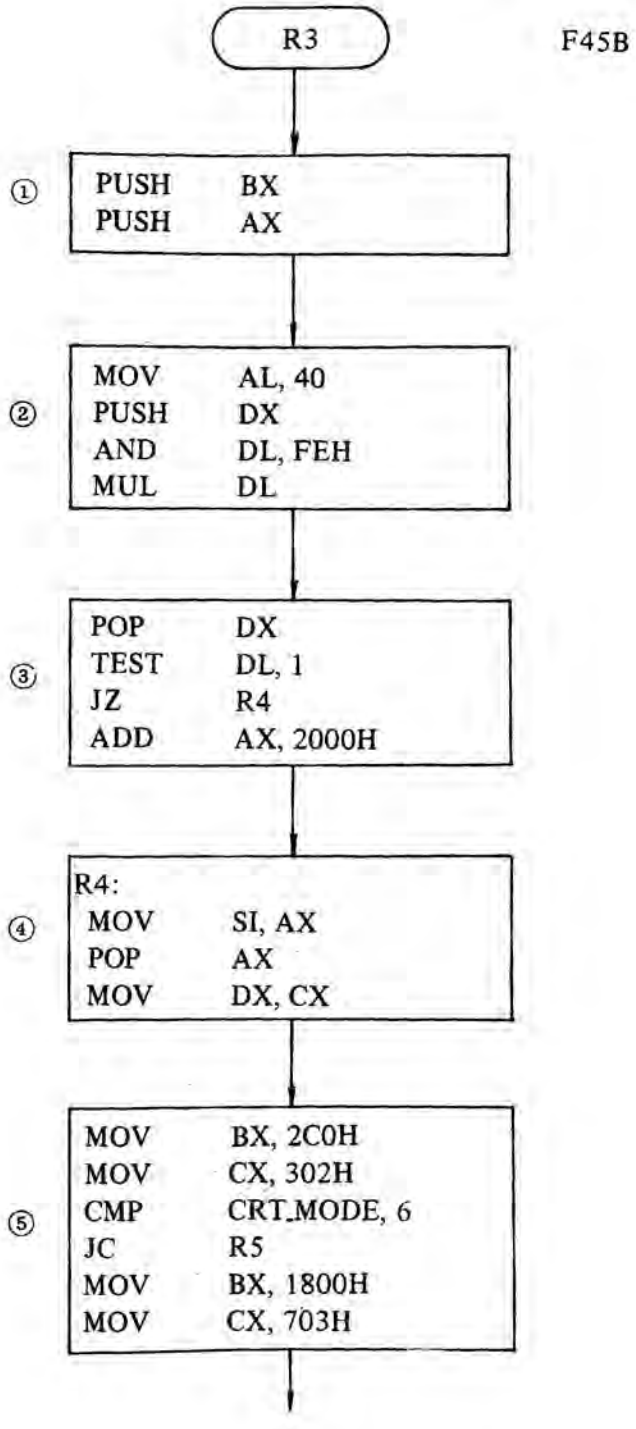
⑩ 呼叫R18副程式填一條線對的顏色屬性值。返回後，將BL內的值減1，測試是否已完成工作，若尚未完成（ZF旗號爲0），則仍需繼續工作。若已完成了工作（ZF旗號爲1），進入下個步驟。

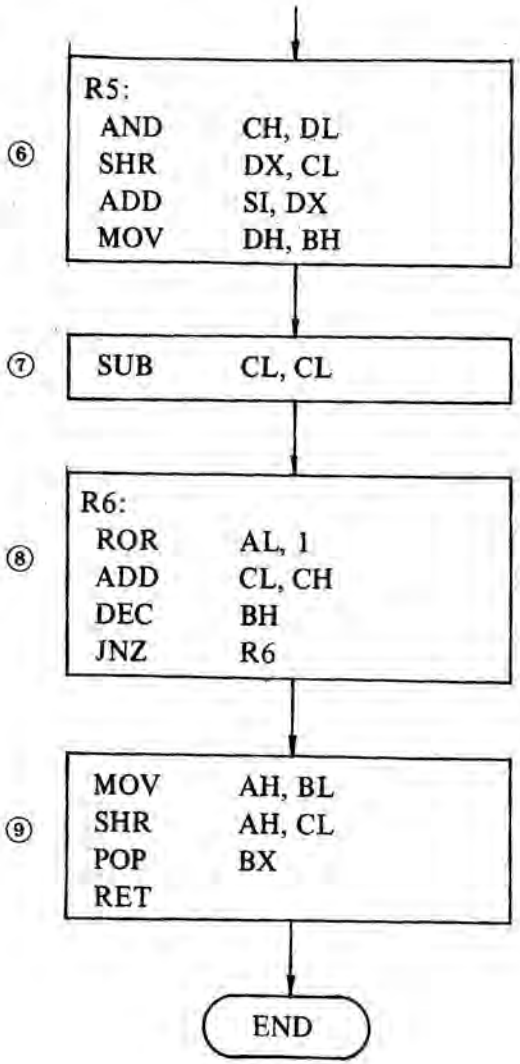
⑪ 清除DF旗號後，經由VIDEO_RETURN返回到呼叫INT 10H的程式去。

⑫ 進入此步驟是因爲要空白整個視窗。將視窗內的線對數（在DH內）放到BL內，然後跳到步驟⑨去做填顏色屬性的工作。

我們現在已討論了4種繪圖功能，它們都是以字元儲存格爲單位。事實上，我們知道繪圖顯像模式是以點爲基礎。所以我們說高解析度爲640×200，中解析度爲320×200。我們接著討論二個繪圖功能，它們的對象就是點。它們是write_dot和Read_dot，這二個程式都需要R3副程式的大力幫忙，所以我們先看R3副程式。

當我們要寫入或讀取某一點時，我們將它的列數（0—199）放到DX內，將行數（0—639）放到CX內，然後再呼叫write_dot或read_dot副程式。R3也用到這二個暫存器內的值來找出指定的點。





① 首先將BX和AX內的值保存起來。

② 進入此程式時，DX內的值指出列數，此步驟要根據DX內的值算出此列的起始位址。先在AL內放入40，以便做爲被乘數。然後將DX保存起來，將DL的位元0設定爲0是因爲偶數列的位址加上2000H後即等於奇數列的值。MUL DL指令算出此列的起始位址（此位址值爲相對於Regen buffer的間距（OFFSET）值。如果DX內爲奇數列，這時算出的是它上一列偶數列的位址），結果在AX內。我們來看二個例子：

例一：若DX內的值爲7，則

(a) AND DL,FEH得到00000110B=6

(b) MUL DL得到240於AX內。

其實240是第6列的起始位址，下個步驟會將240加上2000H，即得第7列的起始位址。

例二：若DX內的值爲8，則

(a) AND DL,FEH得到00001000B=8

(b) MUL DL得到320

320就是第8列的起始位址。

③ 取回DX的初值，並測試其所指的是奇數列或偶數列，若爲偶數列（ZF旗號爲1），則不需加上2000H，直接進入下個步驟。若爲奇數列（ZF旗號爲0），則步驟②算出的起始位址要加上2000H。

④ 將列數的起始位址轉到SI內，然後取回AX的初值，進入此程式時，AL暫存器存放著要寫到指定點的值（此值只在write_dot副程式中用到）。然後將行數值（在CX內）轉到DX去。

⑤ 此步驟根據CRT_MODE值而設定不同的變數。若爲中解析度，則變數爲：

BH=02H，BL=C0H，CH=03H，CL=02H

若爲高解析度，則變數爲：

BH=01H，BL=80H，CH=07H，CL=03H

其中BH指出構成1個點的位元數，BL是用來遮罩點的

值，CL 是用來算出行數的位址，CH是用來算我們指定的點是在那個位元組的第幾個。這 4 個變數在下面的步驟中都有重要的表現，當然也有詳盡的說明。

⑥ 將我們指定的點的列數放到DX 內，行數放到CX 內，我們已經算出這列的起始位址（結果在SI 內）；而且也將行數值轉到DX了，AND CH,DL 指令算出我們指定的點是在那個位元組的第幾個點（結果在CH內，AND 指令不改變DL的值）。我們來看例子：

例一：在中解析度中，若我們指定第 82 行（ 52H ），此時CH值爲 03H

```
        DL=01010010
        CH=00000011
-------------------------
AND     CH=00000010=2
```

如此即找出我們所指定的點在那個位元組的第 2 點。這是因爲中解析度中一個位元組構成 4 點。第 82 行也就指出了前面已有 20 個位元組，第 20 個位元組包含著我們指定的點（第 20 個位元組包含第 80 行到 83行），所以我們可以知道第 82行是在這一列的第 20 個位元組的第2個點（位元組和點都是由 0 起算）

例二：在高解析度中，若我們指定第 333 行（ 14DH）此時CH值爲 07 H

```
        DL=01001101
        CH=00000111
-------------------------
AND     CH=00000101 = 5
```

如此即找出我們所指定的點在那個位元組的第 5 點。這是

因爲高解析度中，一個位元組構成 8 點，第 333 行也就指出了前面已有 41 個位元組，第 41 個位元組包含著我們指定的點（第 41 個位元組包含著第 329 行到第 337行），所以我們可以知道第 333 行在這一列的第 41 個位元組的第 5 個點。AND CH,DL 的結果將在步驟⑧中有重要的表現。我們接下來看下面二個指令 SHR DX,CL，ADD SI,DX,這二個指令算出我們指定的點是在這列的第幾個位元組。同樣的，我們舉例子代替說明。

例一：在中解析度中，若我們指定第 82 行（ 52 H ）此時CL值爲 02 H

```
DX= 0000000001010100B
```

右移 2 個位元後得到 DX= 0000000000010100 B=14H = 20 所以我們指定的點在那列的第 20 個位元組。將此值加到這列的起始位址去，則我們得到包含指定點的位元組的位址（此位址爲相對於Regen buffer 的間距值）於 SI 內。右移 2 個位元即是除以 4 ，這是因爲每個位元組構成 4 個點。

例二：在高解析度中，若我們指定第 333 行 (14DH) 此時CL值爲 03H。

```
DX = 0000001010011101 B
```

右移三個位元得到DX= 0000000000101001B=29H =41，所以我們指定的點在那列的第 41 個位元組。將此值加到這列的起始位址去，我們就得到包含指定點的位元組的位址於 SI 內。右移三個位元即是除以 8 ，這是因爲每個位元組構成 8 個點。

這裡算出的 SI 值對 write_dot 和 Read_dot 副程式都相當重要。這個步驟的最後一個指令是MOV DH,BH。DH值在 read_dot 中將被用到，討論 read_dot 時再說明 DH值的意義。

⑦ 在步驟⑧中將利用CL暫存器來存放結果，所以在這裡先清除爲0。

⑧ 進入此步驟的CH值指出我們指定的點是在那個位元組的第幾個點，而BH內的值決定執行ROR AL，1和ADD CL，CH的次數。我們以例子來解釋這個步驟。

例一：在中解析度中，我們指定第82行（52H），此時CH內的值爲02H，BH內的值爲02H，AL內爲要寫到該點的顏色值，我們假設其爲000000AB。

第一次執行後　AL=B000000A
　　CL=00000010

第二次執行後　AL=AB000000
　　CL=00000100

因爲BH值爲02H，所以執行二次。AL值只在write_dot中用到，先將顏色值移到位元7和位元6，以便在write_dot中使用。在write_dot副程式中，再說明此例的意義。這裡算出的CL值在下個步驟中將很有用。

例二：在高解析度中，我們指定第333行（14DH），此時CH值爲05H，BH內的值爲01H，AL內爲要寫到該點的顏色值，我們假設其爲0000000A。

執行後　AL=A0000000
　　CL=00000101

因爲BH值爲01H，所以只執行一次。下個步驟中，將以這二個例子來做說明。

⑨ 先將CL內的值（高解析度中爲80H，中解析度爲C0H）轉到AH去。然後SHR AH，CL 指令將遮罩移到我們指定的點的位置，我們看下面例子。

例一：在中解析度中，我們指定第82行，在步驟⑧我們得到CL值爲04H，此時AH內爲C0H。

右移前：　AH=11000000
右移後：　AH=00001100

讀者可以發現右移後的AH值，其爲1的二個位元。剛好落在我們指定的點上（第2點），所以我們只要取出包含我們指定的點的那個位元組，然後在AND AH即可取出我們要的點，而不影響其它的點。

例二：在高解析度中，我們指定第333行，在步驟⑧中，我們得到CL值爲05H，此時AH內爲80H

右移前：　AH=10000000
右移後：　AH=00000100

讀者可以發現右移後的AH值，其爲1的位元剛好落在我們指定的點上（第5點）。

好了，我們已經討論完R3副程式，經過R3的運算，若干暫存器內的值對write_dot和Read_dot副程式是相當重要的：

(a) SI：保存著包含我們指定的點的位元組的位址，此位址值爲相對於Regen buffer的間距（OFFSET）值。

(b) AH：保存著我們指定的點的遮罩值。

(c) CL：保存著向右移的次數。

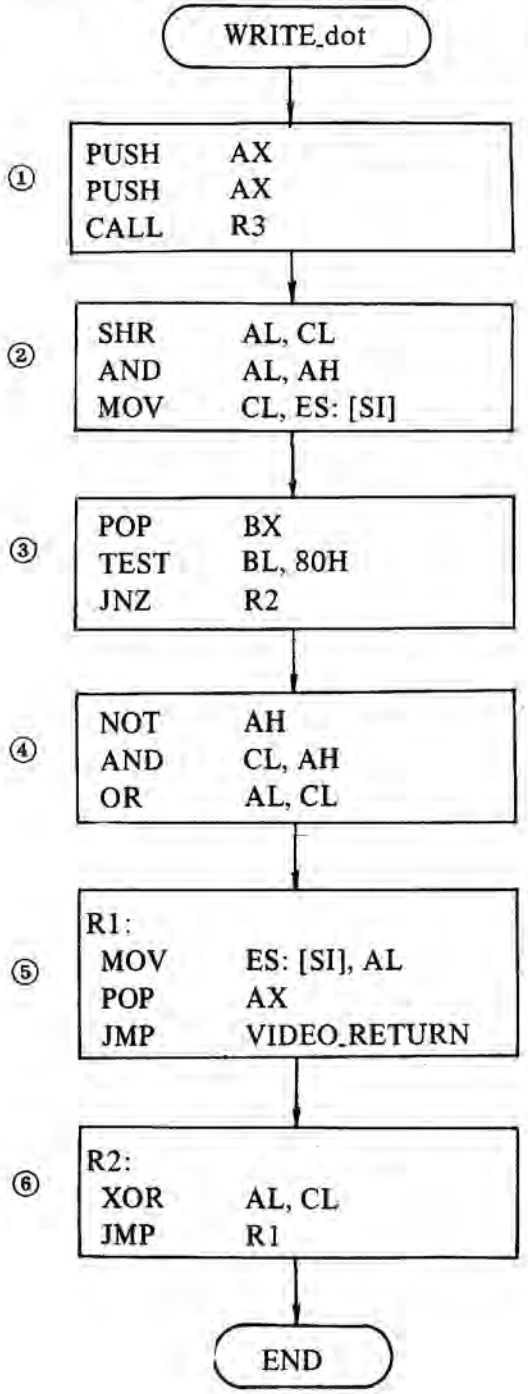
(d) DH：保存著構成點的位元數。

了解R3之後，我們先討論write_dot副程式。

(q) Write_dot

This routine write a dot at the indicated location

Input : (DX) = Row (0-199)
(CX) = column (0-639 或 0-319)
(AL) = color value to write
(DS) = data segment
(ES) = Regen buffer



① 首先將AX內的值保存在堆疊內二次，然後呼叫R 3副程式以得到一些重要的資料。

② R3副程式已將顏色值移至高位元了（如R3副程式中的例子，在中解析度中，AL內爲AB000000；在高解析度中，AL內爲A0000000 ）。從R3返回的CL值保存著向右移的次數，所以SHR AL，CL指令將顏色值移到我們指定的點上，然後AND AL，AH指令取出我們指定的點的顏色值。我們以下面例子說明。

例一：在中解析度中，從R3副程式返回的AL值爲AB000000，CL值爲04H，AH值爲00001100。

那麼：

SHR AL，CL 使得AL內的值爲0000AB00

AND AL，AH 使得AL內的值爲0000AB00

例二：在高解析度中，從R3副程式返回的AL值爲A0000000，CL值爲05H，AH值爲00000100。

那麼：

SHR AL，CL 使得AL內的值爲00000A00

AND AL，AH 使得AL內的值爲00000A00

然後下個指令，MOV CL，ES：〔SI〕，取出包含我們指定的點的那個位元組內的值放到CL內。

③ POP BX 指令取回原始的顏色值於BL內，並測試其位元7值是否爲1，若爲1（ZF旗號爲0），則表示要先將顏色值和原來的顏色值互斥後再寫到指定的點上，跳入步驟⑥，否則進入下個步驟。

④ 此步驟在做write_dot的動作。NOT AH和AND CL，AH指令清除我們要寫的點，OR AL，CL 將顏色值寫到指定的點上。我們看例子：

例一：在中解析度中，CL 內的值爲包含指定點的那個位元組內的值，我們假設爲LMNOPQRS

NOT AH 使得AH內的值爲11110011

AND CL,AH 使得CL內的值爲LMNO00RS

OR AL,CL 使得AL內的值爲LMNOABRS

如此即將顏色值寫到我們指定的點上，而不影響其它的位元。

例二：在高解析度中，CL內的值爲包含指定點的那個位元組內的值，我們假設其爲LMNOPQRS

NOT AH指令使得AH內的值爲11111011

AND CL,AH指令使得CL內的值爲LMNOP0RS

OR AL,CL指令使得AL內的值爲LMNOPARS

如此即將位元值寫到我們指定的點上，而不影響其它的位元。

⑤ 將步驟④中得到的結果放回去，然後取回AX暫存器的初值後，經由VIDEO_RETURN返回到呼叫INT 10H的程式。

⑥ 進入此步驟是因爲要先將顏色值和原來的顏色值互斥。

例一：在中解析度中，CL內的值爲包含指定點的那個位元組內的值，我們假設其爲LMNOPQRS，並且假設AB和PQ互斥得到TU。則

AL=0000AB00

CL=LMNOPQRS

XOR AL=LMNOTURS

如此即互斥了顏色值，而不影響其它的位元。

例二：在高解析度中，CL內的值爲包含指定點的那個位元組內的值，我們假設其爲LMNOPQRS，並且假設A和Q互斥得到T。則

AL=00000A00

CL=LMNOPQRS

XOR AL=LMNOPQRS

如此即互斥了顏色值，而不影響其它的位元。然後跳入步驟⑤將結果放回原來的地方。

(r) read_dot

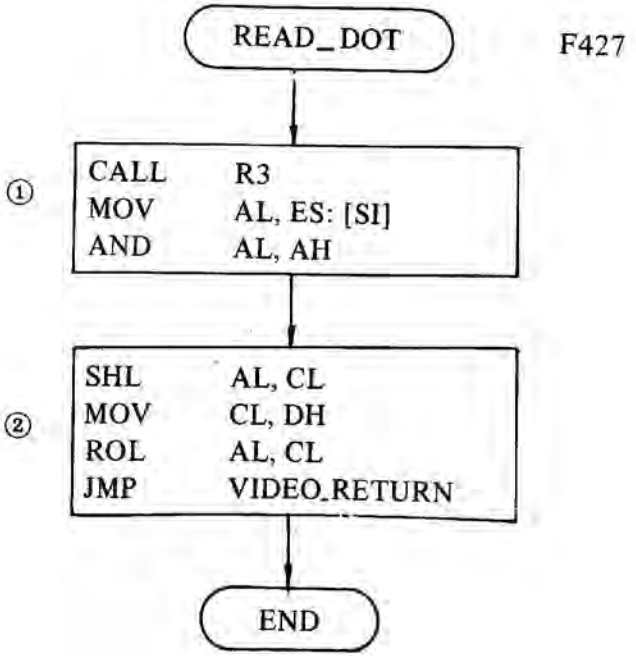
This routine read a dot at the indicated location

Input : (DX) = Row (0-199)

(CX) = column (0-639, 0-319)

(DS) = data segment

(ES) = Regen buffer



① 此步驟首先呼叫 R3 副程式，得到一些重要的資料，然後取出包含我們指定點的那個位元組的值於 AL 中。AND AL，AH 隔離出我們指定的點（ AH 內保存著我們指定的點的遮罩值）。

例一：在中解析度中，AH 內的值爲 00001100，AL內爲包含我們指定點的那個位元組內的值，我們假設其爲LMNOPQRS，則

```
        AH=00001100
        AL=LMNOPQRS
-------------------------
AND     AL=0000P000
```

如此即隔離我們要的點。

例二：在高解析度中，AH 內的值爲 00000100，AL內爲包含我們指定點的那個位元組內的值，我們假設其爲LMNOPQRS，則

```
        AH=00000100
        AL=LMNOPQRS
-------------------------
AND     AL=00000Q00
```

如此即隔離出我們要的點。

② 此步驟將我們要的點轉到低位元去。做法是這樣的，首先將我們要的點移到高位元（ SHL AL，CL），然後利用左旋轉指定將其旋轉到低位元去(MOV CL，DH和ROL AL，CL 指令）。我們看例子：

例一：在中解析度中，CL值爲 04H，DH值爲 02H，
AL內爲步驟①得到的結果，0000PQ00 ，
SHL AL，CL 使得AL 內爲PQ 000000
MOV CL，DH使得CL內爲 02H
ROL AL，CL 使得AL 內爲 000000PQ
如此即將我們要的點轉到低位元去了。

例二：在高解析度中，CL 值爲 05H，DH值爲 01H，
AL 內爲步驟①得到的結果爲 00000Q00。
SHL AL，CL 使得AL 內爲Q0000000
MOV CL，DH使得CL 內爲 01 H
ROL AL，CL 使得AL 內爲 0000000Q
如此即將我們要的點轉到低位元去了。

最後經由VIDEO_RETURN返回到呼叫 INT 10H 的程式去。

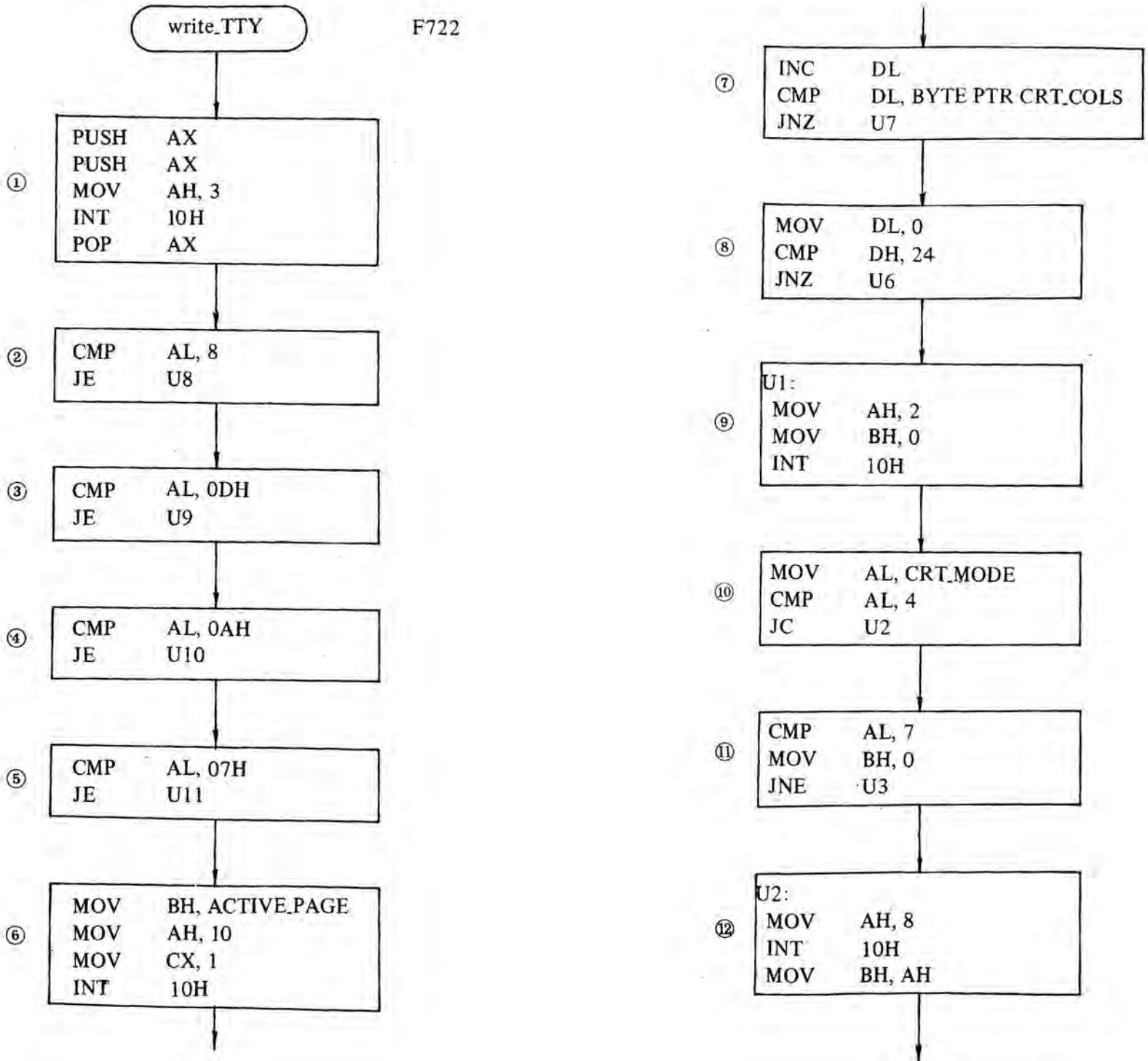
write_dot，read_dot 和R3副程式寫得短而精彩，讀者務必要了解，對您自己寫組合語言程式時一定有幫助。

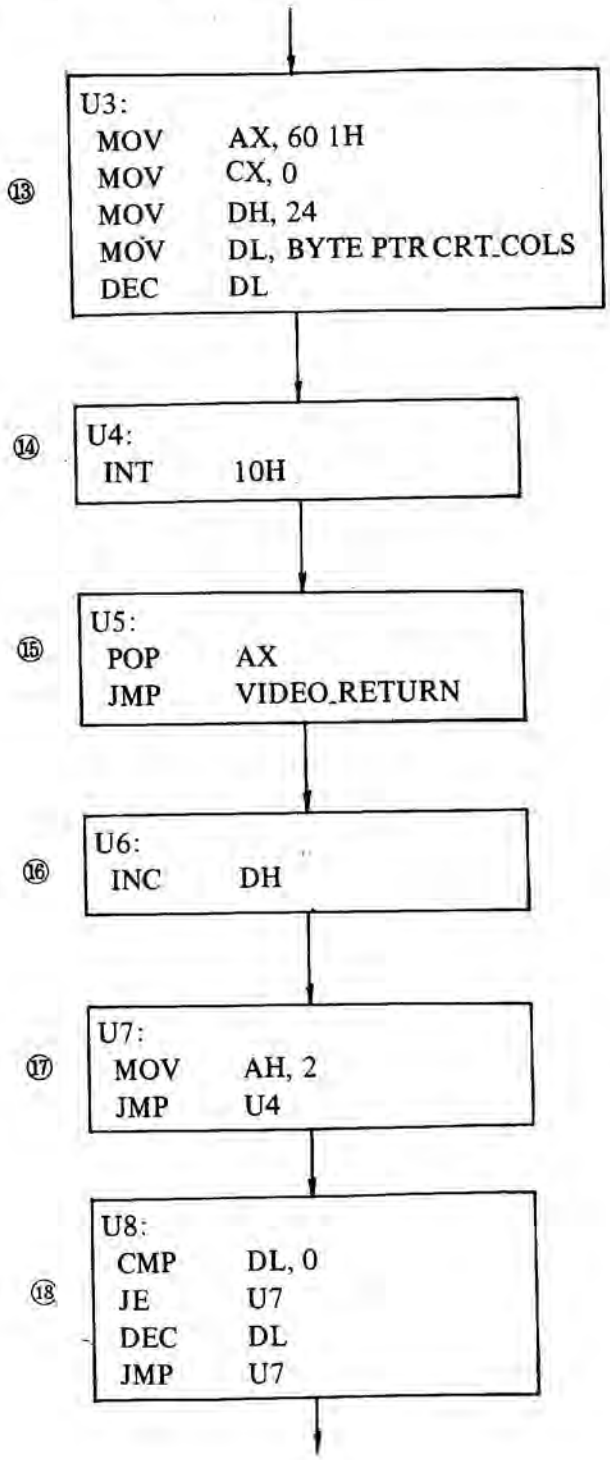
(s) (AH)=14 write_Teletype

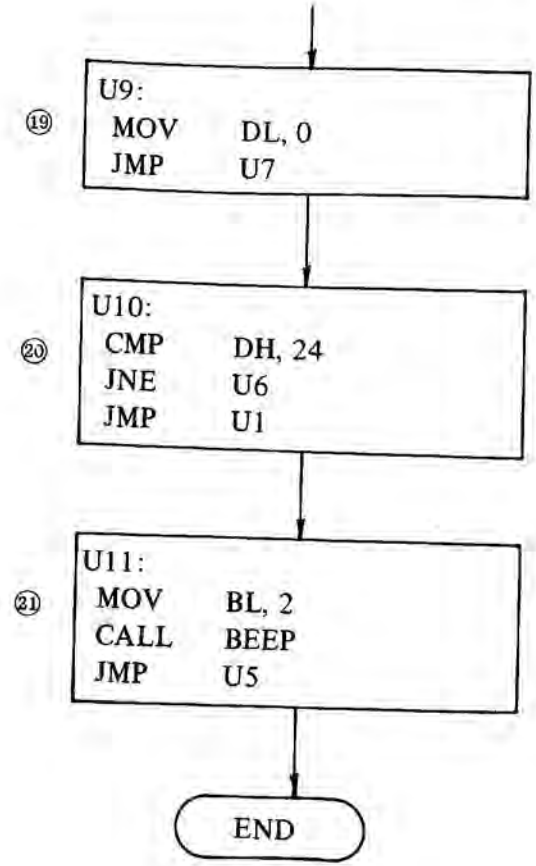
This routine writes character to the current cursor position, and the cursor is moved to the next position, line feed, carriage return and scroll up are supported,

Input : (AH) = current CRT mode
(AL) = character to be written
(BL) = Foreground color for character write if currently in a graphics mode

此程式可以說是INT 10H程式的綜合應用，其中用到 INT 10H 內若干副程式（Subroutine）。


write_TTY
F722
①
PUSH AX
PUSH AX
MOV AH, 3
INT 10H
POP AX
②
CMP AL, 8
JE U8
③
CMP AL, 0DH
JE U9
④
CMP AL, 0AH
JE U10
⑤
CMP AL, 07H
JE U11
⑥
MOV BH, ACTIVE_PAGE
MOV AH, 10
MOV CX, 1
INT 10H
⑦
INC DL
CMP DL, BYTE PTR CRT_COLS
JNZ U7
⑧
MOV DL, 0
CMP DH, 24
JNZ U6
U1:
⑨
MOV AH, 2
MOV BH, 0
INT 10H
⑩
MOV AL, CRT_MODE
CMP AL, 4
JC U2
⑪
CMP AL, 7
MOV BH, 0
JNE U3
U2:
⑫
MOV AH, 8
INT 10H
MOV BH, AH





① 首先保存AX內的值二次，然後在AH內放入3，呼叫INT 10H，以便讀出游標的位址於DX內（DH爲列，DL爲行），然後取出AX的初值，其中AL內爲字元值。

② 檢查此字元是否爲退位鍵碼（Back - space）（ASCII碼爲08H），若是（ZF旗號爲1），進入步驟⑱。否則進入下個步驟。

③ 檢查此字元值是否爲歸位鍵碼（carriage-return）（ASCII碼爲0DH），若是（ZF旗號爲1），進入步驟⑲，否則進入下個步驟。

④ 檢查此字元值是否爲換列碼（line feed）（ASCII碼爲0AH），若是（ZF旗號爲1），進入步驟⑳，否則進入下個步驟。

⑤ 檢查此字元值是否爲BELL碼（ASCII碼爲07H），若是（ZF旗號爲1），進入步驟㉑，否則進入下個步驟。

⑥ 既然字元不是控制用的字元，這個步驟將字元值寫到游標的位址去。

⑦ 將行數加1（在DL內），然後和CRT_COLS比較，若不相等（ZF旗號爲0），則表示游標尚未到達螢幕的最右邊，跳入步驟⑰。若行數等於CRT_COLS內的值（ZF旗號爲1）則表示游標已超過螢幕的最右端，此時必須將游標移到下一列的最左端，進入下個步驟。

⑧ 將行數（在DL內）設定爲0，然後將列數和24比較，若列數不等於24，即表示尚不需上捲，跳入步驟⑯，若列數等於24（ZF旗號爲1），表示要上捲，進入下個步驟。

⑨ 此步驟設定游標的位置，即列數爲24，行數爲0的位置。

⑩ 此步驟測試CRT_MODE，若CRT_MODE小於4（CF旗號爲1），則表示爲文字模式（Alphabetic mode），跳入步驟⑫，若CRT_MODE大於4，則進入下個步驟。

⑪ 此步驟測試是否爲單色（monochrone），並預先在BH內放入00H，以便做爲上捲時的屬性值。若不爲單色，則跳入步驟⑬。若爲單色（ZF旗號爲1.），進入下個步驟。

⑫ 進入此步驟是因爲彩色/繪圖卡是文字模式或單色。此步驟讀出第24列，第0行位置內的屬性值到BH內，以便在上捲一列後，將此屬性值填到最底端一列。

⑬ 此步驟爲上捲一列做準備，視窗的左上角座標爲（0，0），在CX內。右下角的座標爲（24，CRT_COLS-1），在DX內。所以要將CRT_COLS減1是因爲行數是由0起算。值得注意的是繪圖模式中，顏色屬性值爲00H（在BH內，由步驟⑪設定），在文字和單色中，屬性值是第24列，第0行位置內的屬性值（在BH內，由步驟⑫得到）。

⑭ 呼叫INT 10H上捲一列。

⑮ 取回AX的初值後，經由VIDEO_RETURN返回到呼叫INT 10H的程式去。

⑯ 進入此步驟是因爲游標已到螢幕的最右端，但尚不需要上捲（步驟⑧）。將列數（在DH內）加1，進入下個步驟。

⑰ 進入此步驟可能由步驟⑯而來，或是因爲游標尚未到達螢幕的最右端（步驟⑦）。在AH內放入2，以便在呼叫INT 10H表示要設定游標位置，然後跳入步驟⑭去設定游標位置，並返回。

⑱ 進入此步驟是因爲要寫到螢幕畫面去的字元值爲退位鍵碼(步驟②)，首先測試行數是否爲0，若是（ZF旗號爲1），則游標再也不能向左移了，跳入步驟⑰去設定游標位置。若游標不在螢幕畫面的最左端，則將行數減1後，跳入步驟⑰去設定游標位置。

⑲ 進入此步驟是因爲測試到字元值爲歸位鍵碼(步驟③)，將行數設定0後，跳入步驟⑰去設定游標位置。值得注意的是歸位鍵碼只將游標移到同列的最左端，而不是移至下一列的的最左端。

⑳ 進入此步驟是因爲測試到字元爲換列碼（步驟④）。測試列數是否爲24，若不是（ZF旗號爲0），則尚不需上捲，跳入步驟⑯，將列數加1後，設定游標的位置。注意：換列只將列數加1，而不改變行數的位置。若列數等於24(ZF旗號爲1），則表示需要上捲了，跳入步驟⑨去做上捲的動作。值得注意的是，此時上捲所用的屬性值不再是第24列，第0行位置內的屬性值了，因爲此時行數不再是0。屬性值將取自（24,DL）這個位置內的屬性值（繪圖的屬性值仍爲00H)。

㉑ 進入此步驟是因爲字元爲BELL（步驟⑤），在BL內放入2，然後呼叫BEEP副程式，使喇叭發出聲音後，經由步驟⑮返回。

(t) (AH)＝4 read light pen position

Output : (AH) = 0 if no light pen information is available

(AH) = 1 if light pen is available

(DH, DL) = Row, column of current light pen position

(CH) = raster position

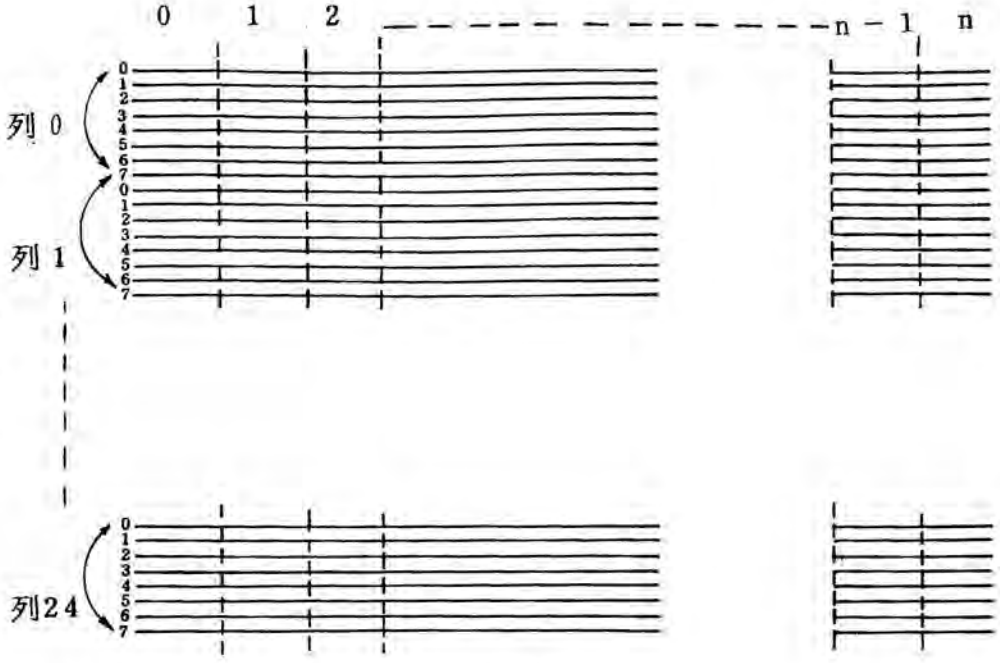
(BX) = best guess at pixel horizontal position

IBM PC 的單色界面卡沒有光筆功能，只有彩色／繪圖界面卡才有光筆功能。

在討論程式前，我們必須先建立一些概念。

文字模式（40×25，80×25）

我們先看下圖，螢幕畫面結構和6845的Refresh Memory Address(MA)，Raster Address(RA)的關係。



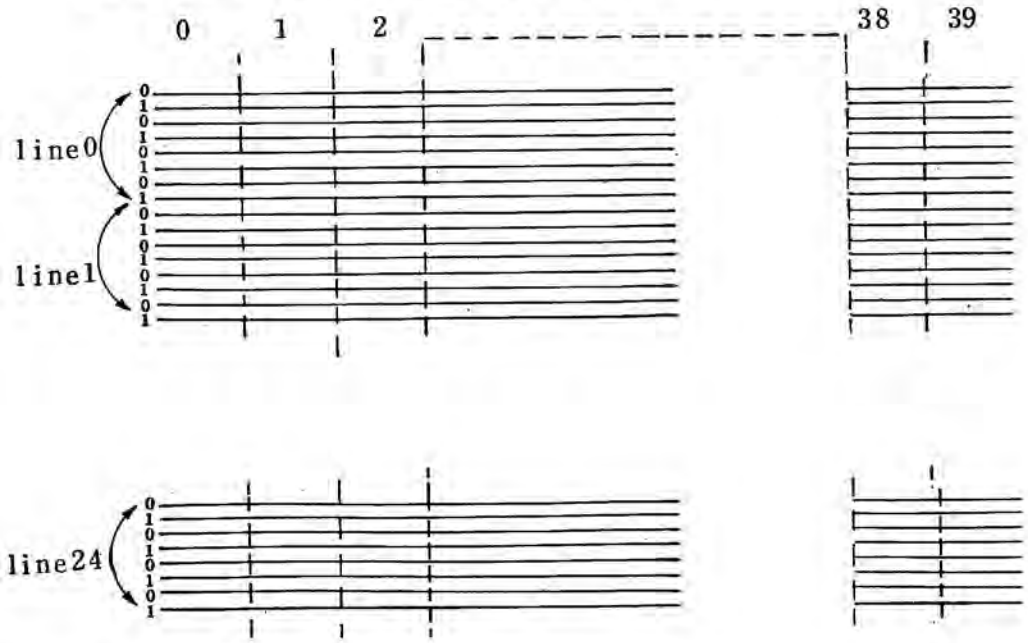
其中：80×25時 n＝79

40×25時 n＝39

每個字元均有8條掃描線（Raster line）（0—7）。6845掃描的動作是這樣的：先使得RA和MA都爲0，此時掃描到line 0的字元0的RA0。然後將MA的值加1，不改變RA的值，此時掃描到line 0的字元1的RA0。然後不改變RA的值，而陸續增加MA的值，直到MA的值等於n爲止，也就是掃描到line 0的字元n的RA0爲止。此時，6845將RA值加1，並設定MA的值爲0，這樣就掃描到line 0的字元0的RA1，同樣的不改變RA值，只增加MA的值，直到MA的值等於n爲止。此時6845又將RA值加1，並設定MA的值爲0，這樣就掃描到line 0的字元0的RA2。如此繼續下去，直到掃描到line0 的字元n的RA7爲止。此時6845不將MA設定爲0了，它將MA設定成n＋1，並設定RA值爲0，此時6845掃描到line 1的字元0的

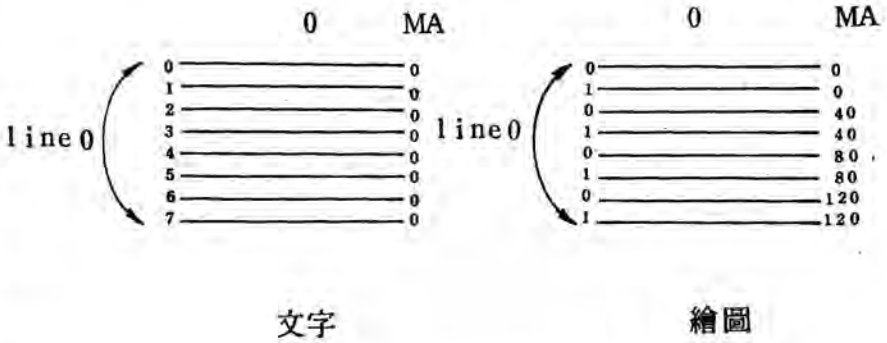
RA 0，然後重複至 line 0 的步驟，直到掃描到 line 1 的字元 2n＋1 的RA7，然後 6845 將RA設定爲 0，並將MA 設定 2n+2，以便掃描到 line 3。依此重覆動作直到 6845 掃描到 line 24 的最後一個字元的RA7 後。6845 同時設定MA 和RA 爲 0，以便從 line 0 再開始掃描起。從上面的敍述，我們可以說 6845重覆 8 次將MA 值由字元 0 陸續加到字元 n 的動作，然後才指向下一列。所以 line 0 的MA 值由 0 開始，line 1 的MA 值由 n＋1 開始，依此類推，line 24 的MA 值由 24n+24 開始到 25n+24 爲止。

文字模式的掃描是比較單純的，因爲 6845 的 R1 暫存器被設定成 40（40×25）或 80（80×25）。但是繪圖中，6845 的 R1 暫存器被設定成 40，而不管是高解析度或中解析度。實際上繪圖中，每列均有 80 個位元組。換句話說，每個 6845 的MA 值要涵蓋 2 個位元組，中解析度中每個 MA 值涵蓋 8 個點，高解析度中每個MA 值涵蓋 16 個點，請看下圖它們的關係。



其中最後一個字元爲字元 39。

6845 掃描的動作和文字一樣，所不同的是 6845 只重覆二次，將MA 值由字元 0 陸續加到字元 n 的動作，而且每個 MA 值涵蓋二個字元組。必須注意的是每個字元有 8 條掃描綫，但這 8 條掃描綫的MA 值並不相同，請看下圖。



文字　　繪圖

從上圖我們可以看出文字和繪圖的字元的差異，請讀者記下下列的原則：

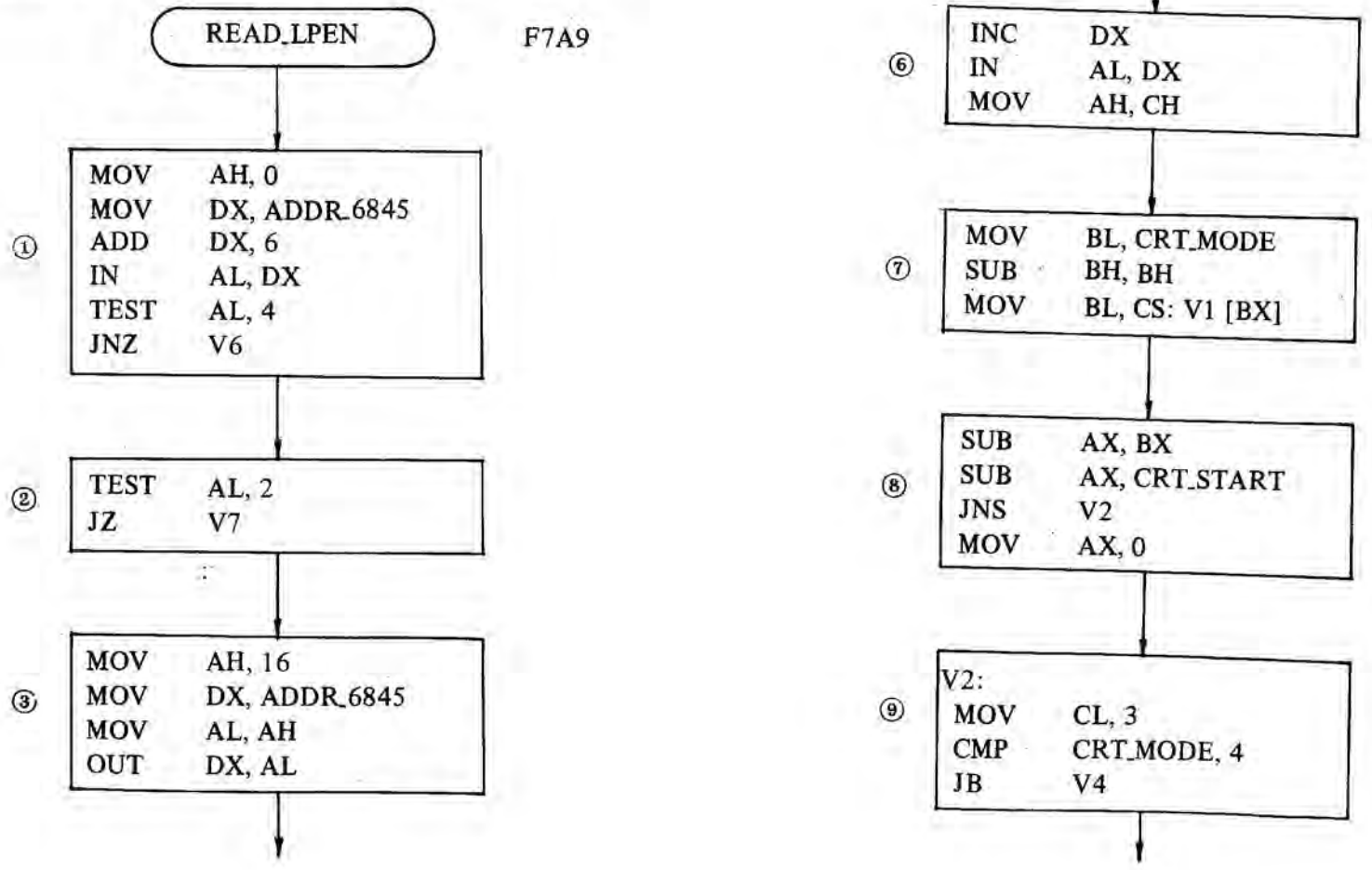
① 在文字中，每個字元有 8 條掃描綫，6845 要掃描 8 條掃描綫後，才改變 MA 的值，否則只重覆字元 0 加到字元 n 的動作。

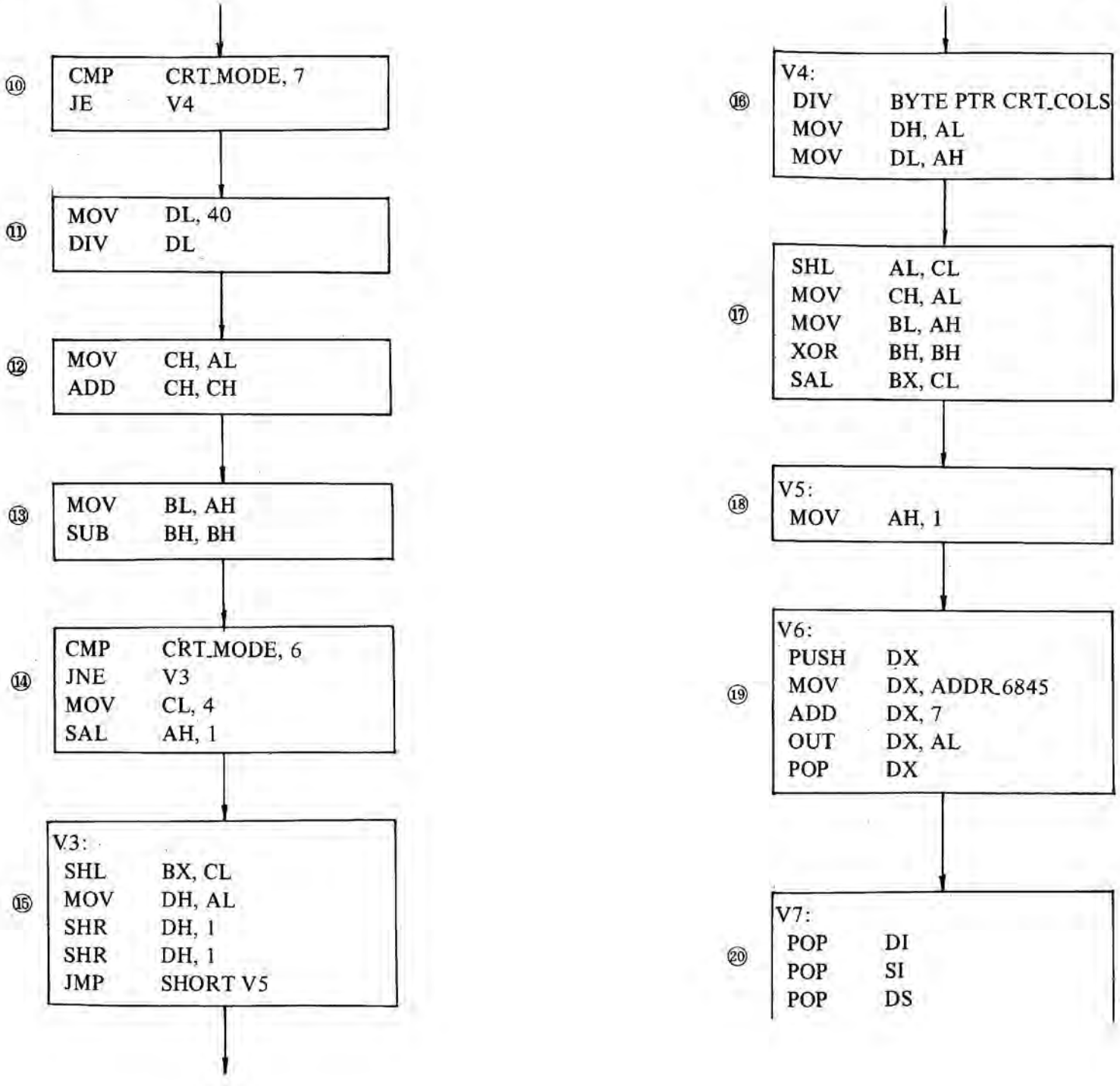
② 在繪圖中，每個字元有 8 條掃描綫，6845 掃描 2 條掃描綫後，改變 MA 的值，否則只重覆字元 0 加到字元 39 的動作。在中解析度中，每個MA 值涵蓋 8 個點（2 個位元值）。在高解析度中，每個MA 值涵蓋 16 個點（2 個位元組）。

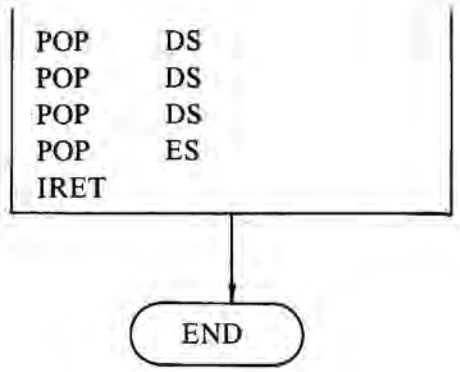
只要記著這二個原則，討論 Read_LPEN就方便多了。在討論程式前，還得說明一點，那就是當光筆（light pen）接到彩色/繪圖卡之後，它必須“告訴”彩色/繪圖卡系統上已有光筆了，這

個“告訴”的訊號是經由I/O埠值3DAH的位元2。若位元2爲0，才表示有光筆。當光筆指著螢幕上的某一點，此時光筆等著CRT（陰極射綫管）掃描這一點。若CRT掃描到這一點，則光筆會給6845一個訊號，此訊號使得6845將現在的MA值保存到R16和R17中，以便程式讀出它。同時，光筆也會使I/O埠值3DAH的位元1成爲高電位，以便顯示已有光筆的訊號了。6845保存在R16和R17內的MA值實際上要減去若干值，此差異是因爲綫路上的延遲（delay）。

有了這些概念後，我們可以來討論程式。




⑩
CMP CRT_MODE, 7
JE V4
⑪
MOV DL, 40
DIV DL
⑫
MOV CH, AL
ADD CH, CH
⑬
MOV BL, AH
SUB BH, BH
⑭
CMP CRT_MODE, 6
JNE V3
MOV CL, 4
SAL AH, 1
⑮
V3:
SHL BX, CL
MOV DH, AL
SHR DH, 1
SHR DH, 1
JMP SHORT V5
⑯
V4:
DIV BYTE PTR CRT_COLS
MOV DH, AL
MOV DL, AH
⑰
SHL AL, CL
MOV CH, AL
MOV BL, AH
XOR BH, BH
SAL BX, CL
⑱
V5:
MOV AH, 1
⑲
V6:
PUSH DX
MOV DX, ADDR_6845
ADD DX, 7
OUT DX, AL
POP DX
⑳
V7:
POP DI
POP SI
POP DS



① 此步驟測試是否有光筆存在，若沒有（ZF旗號爲0），跳入步驟⑲；若有光筆（I/O埠3DAH的位元2值爲0，此時ZF旗號爲1），則進入下個步驟。

② 此步驟測試I/O埠3DAH的位元1是否爲1，若不爲1（ZF旗號爲1），則表示沒有光筆訊號產生，跳入步驟⑳，若有光筆訊號產生（ZF旗號爲0），進入下個步驟

③ 此步驟將16寫到6845的位址暫存器（address register），表示我們要讀R16的值。

④ 讀出R16的值，並轉到CH內。

⑤ 此步驟將17寫到6845的位址暫存器，表示我們要讀R17的值。

⑥ 此步驟讀出R17的值於AL，並把R16的值由CH內轉到AH內。如此AX內即保存著MA值。

⑦ 因爲綫路上的延遲，所以從V1表格內取出差額。此差額因模式不同而不同。（結果在BX內）。

⑧ 首先減去差額，然後再減去CRT_START內的值（我們所舉的例子是以頁0做依據。若CRT_START內的值不爲0，則字元0的RA0的MA值將是CRT_START內的值，請看INT 10H中AH值爲5的選擇顯像頁數的部份），若結果爲負，則我們認爲MA值爲0（結果在AX內），然後才進入下個步驟，否則以相減後的結果（在AX內）進入下個步驟。

⑨ 預先在CL內放入3，這是給中解析度模式使用的，在步驟⑮中將用到。然後測試CRT_MODE，若CRT_MODE內的值小於4，則表示爲文字，跳入步驟⑯。若不爲文字，則進入下個步驟。

⑩ 爲了保證爲繪圖，則再將CRT_MODE值和7比較，若爲7，則表示爲單色（其實單色卡上並沒有光筆功能），跳入步驟⑯當做文字處理。若爲繪圖（ZF旗號爲0），進入下個步驟。

⑪ 因爲繪圖中，每條掃描綫有40個MA值，所以這裡將MA值除以40。列數在AL內，行數在AH內。

⑫ 這個步驟將列數乘以2，並保存在CH內。我們來看例子：若MA值爲83，則除以40後，AL內的值爲2，AH內的值爲3，但事實上，MA值爲83是在第4和第5列，所以這裡需將AL內的值乘以2。請參考我們的說明。

⑬ 將行數放到BL內，並清除BH內的值。以便用來猜光筆的大概位置（以點做單位）。

⑭ 此步驟分辨出高解析度或中解析度，若爲中解析度（ZF旗號爲0），直接進入下個步驟而不需修改。若爲高解析度，則設定CL爲4，並將行數乘以2，這樣做的原因是高解析度中。每個MA值涵蓋16個點。

⑮ 將行數乘以8（中解析度）或乘以16（高解析度），以便

猜出光筆的位置。以上例來說，MA值爲83時，行數爲3，乘以8後得到24，或乘以16後得到48；也就是說我們猜光筆在這一列的第24個點或第48個點。同時將列數除以4，以便找出光筆是在那一列（單位爲字元儲存格），除以4是因爲一個字元儲存格內有4條線對（結果在DH內）。然後跳入步驟⑱表示做完工作了。

⑯ 進入此步驟是因爲文字，同樣的將MA值除以CRT_COLS，得到的列數轉到DH去，行數轉到DL去（單位爲字元）。

⑰ 進入此步驟的CL值爲3（在步驟⑨中已設定）。將列數乘以8，以便猜光筆是在那條掃描線，乘以8是因爲每個字元有8條掃描線，然後將結果轉到CH去。這裡也將行數乘以8，以便猜出光筆是在那一列的第幾個點，乘以8是因爲每個MA值涵蓋8個點（結果在BX內）。

⑱ 此步驟設定AH爲1，以便表示已做完工作。

⑲ 此步驟清除光筆訊號，以免下次讀到同樣的結果。

⑳ 當此程式讀到光筆訊號時，將結果放到BX，DX，CH內。

(a) DH：包含著光筆所在的列數（單位爲字元）

(b) DL：包含著光筆所在的行數（單位爲字元）

(c) CH：光筆所在的掃描線數（此值是用猜的）

(d) BX：光筆所在點的位置（此值是用猜的）

既然已有資料在這三個暫存器內，所以就不能取回它們的初值，最後返回到呼叫INT 10H的程式去。

好了，我們已將INT 10H全部討論完畢，相信讀者對MONITOR已經有了相當的了解。在PC上，顯像是相當重要的，讀者最好是能熟悉顯像的部分，以便在PC上有更佳的表現。

第七章

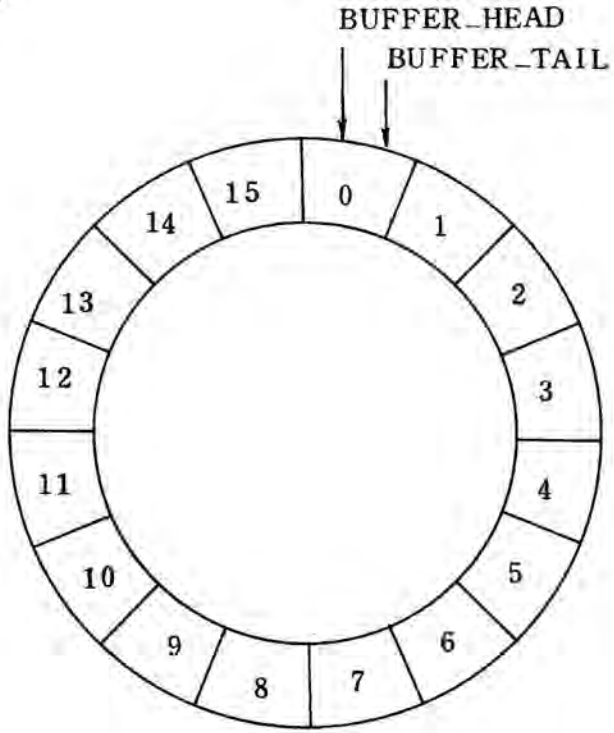
鍵　盤

鍵盤（Keyboard）和螢幕（Monitor）都是和使用者溝通的工具，藉著這二樣工具，人們可以和電腦“交談”。IBM PC上的螢幕已在第六章詳盡討論過了。在這一章裡，我們討論IBM PC的鍵盤。

IBM PC的鍵盤是使用8048當作中央處理機，8048的工作是探測是否有鍵被按下和送出此鍵的SCAN CODE（也就是表示鍵碼所在位置的號碼），8048送出SCAN CODE後，促使主機板上發出一個中斷信號。此中斷信號通知8088有鍵被按下了，此時8088跳入KB_INT副程式，將SCAN CODE轉換成此鍵的ASCII碼，並且同時將SCAN CODE和ASCII碼保存在鍵盤緩衝區（Keyboard Buffer）內。8088可以先不管此鍵爲何（除非是RESET，control-C，shift-PrtSc等），直到某程式需要按鍵時，8088才到鍵盤緩衝區去拿鍵碼。KB_INT副程式只是一些比較的指令，筆者不打算討論，若讀者有興趣，請自行研究。筆者認爲鍵盤緩衝區結構的了解和KEYBOARD_IO副程式（INT16H）的認識，對使用者來說較有意義，因爲一般使用者不會呼叫KB_INT（INT 9H，其實INT 9H是由8259所發出的）。

我們先來看鍵盤緩衝區的結構，KB_BUFFER爲分段(segment)值爲0040H，間距（OFFSET）值爲001EH起的16個字組記

憶體。由於有BUFFER_HEAD和BUFFER_TAIL 二個指標的幫忙，KB_BUFFER就變成一個圓環，請看下圖，POST中最初設定BUFFER_HEAD和BUFFER_TAIL，都指向KB_BUFFER的起始位址。



我們來看資料存取時，BUFFER_HEAD和BUFFER_TAIL的變化。

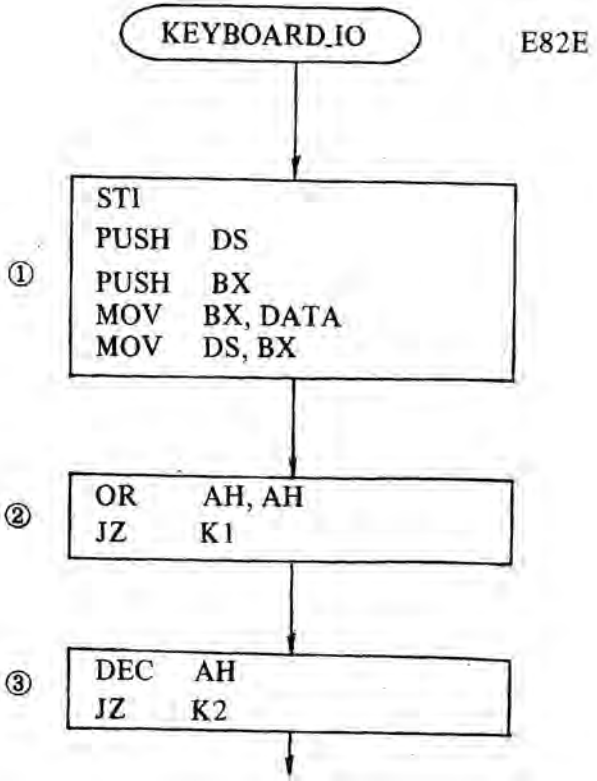
① 儲存資料是由KB_INT來做，我們以最初的狀態討論起，此時BUFFER_HEAD和BUFFER_TAIL都指著KB_BUFFER的起始位址（字組0）。當KB_INT將SCAN CODE轉換成ASCII碼後，它先將 BUFFER_TAIL 加2，然後測試BUFFER_TAIL是否已超出了KB_BUFFER的範圍。若尚未，則將SCAN CODE和ASCII碼放到前一個字組去（例如：BUFFER_TAIL由0增加成1，此時BUFFER_TAIL尚未超過KB_BUFFER，則將SCAN CODE和ASCII碼放到字組0，ASCII碼在低位元組，SCAN CODE在高位元組）。若BUFFER_TAIL加2後超過了KB_BUFFER的範圍（由字組15增加到字組16，但KB_BUFFER並沒有字組16），則必須將BUFFER_TAIL指向緩衝區的起始位址（字組0），但此時若BUFFER_HEAD 還指著字組0（表示沒有資料被取走），則此時不能將SCAN CODE和ASCII碼存放到字組15的位置，並且將BUFFER_TAIL再指回字組15的位置，這樣做的原因是，避免下次儲存資料時蓋掉尚未被取走的資料。此時KB_INT 副程式會使喇叭發出聲響，並且忽略掉所按的第16個鍵，所以KB_BUFFER 一次最多只能儲存15個鍵。若BUFFER_TAIL超過了KB_BUFFER的範圍，但BUFFER_HEAD 不指著字組0（表示已有資料被取走了），則此時仍可以將ASCII碼和SCAN CODE 存到字組15的位置，並且將BUFFER_TAIL 指向字組0，也就是說在儲存資料時，BUFFER_TAIL不能追上BUFFER_HEAD，並且至少要保存一個字組的距離。

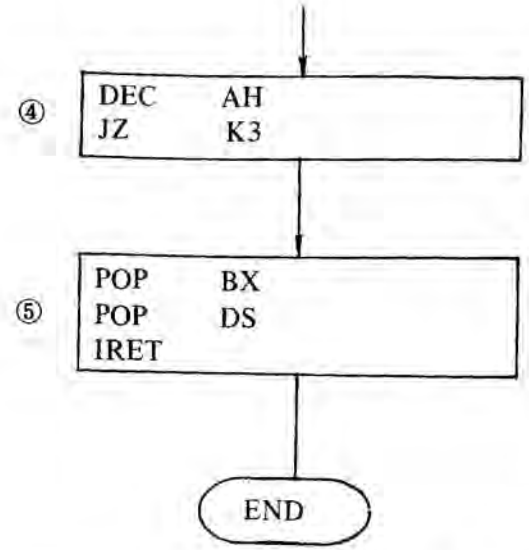
取出資料的動作是由KEYBOARD_IO來做。KEYBOARD_IO在取出資料之前，先要測試BUFFER_HEAD是否和BUFFER_TAIL相等，若二者相同，即表示緩衝區內，並沒有資料可取，必須等到有鍵被按下（BUFFER_TAIL 向前進一個字組以上），KEYBOARD_IO才能取出BUFFER_HEAD 所指的字組內的資料。BUFFER_HEAD可以追上BUFFER_TAIL，但不能超越它。同樣的，若BUFFER_HEAD 超過了緩衝區的範圍，KEYBOARD_IO要將BUFFER_HEAD 指向字

組 0 。

了解鍵盤緩衝區後，討論KEYBOARD_IO 副程式就輕鬆多了。KEYBOARD_IO有三種功能，我們將會討論。

```
KEYBOARD_IO
  This routine provides keyboard support
  Input: (AH) = 0 Read the next ASCII character struck
                  from the keyboard return the result in
                  (AL), scan code in (AH)
         (AH) = 1 set the Z flag to indicate if an ASCII
                  character is available to be read
                  (ZF) = 1  no code available
                  (ZF) = 0  code is available
         (AH) = 2 return the current shift status in AL
```





① 首先設定 IF旗號爲 1 ，並陸續保存 DS 和 BX 內的值後，將 DATA (0040) 轉到 DS 去。

② 此步驟檢查 AH 內的值是否爲 0 ，若爲 0 (ZF 旗號爲 1)，則表示要讀字元，跳入 K1 。若 AH 內的值不爲 0 ，進入下個步驟。

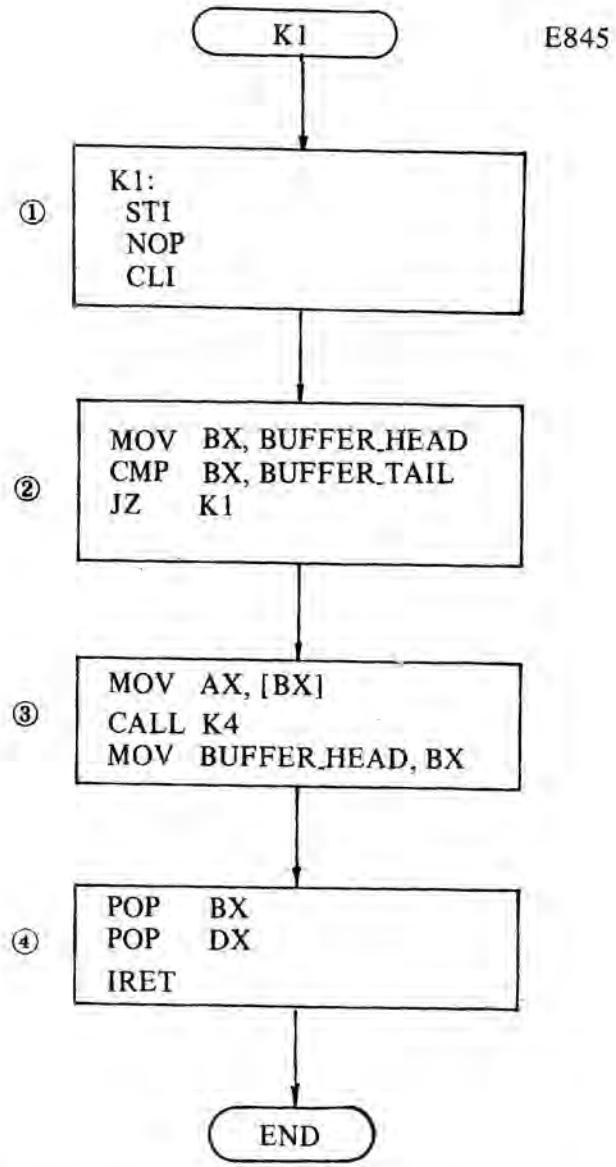
③ 此步驟檢查 AH 內的值是否爲 1 ，若爲 1 (ZF 旗號爲 1)，則表示要測試是否有資料可讀，跳入 K2 。若 AH 內的值不爲 1 ，進入下個步驟。

④ 此步驟檢查 AH 內的值是否爲 2 ，若爲 2 (ZF 旗號爲 1)，則表示要知道 shift 狀態，跳入 K3 。若 AH 內的值不爲 2 ，則表示 AH 的值大於 2 ，是不正確的，進入下個步驟。

⑤ 陸續取回 BX 和 DS 的初值後，返回到呼叫 INT 16H 的程式去。

讀者可以發現KEYBOARD_IO只是做檢查的工作。眞正的動

作是由K1，K2和K3來做，我們先來看K1。

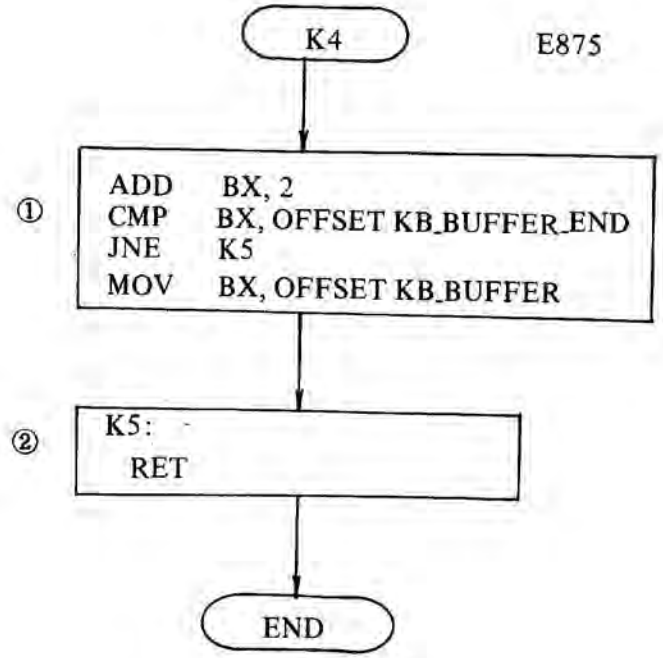


① 此步驟執行了一個NOP指令，以便等待其它的中斷信號產生，然後清除IF旗號，因爲以下的步驟不希望被打斷。

② 此步驟比較BUFFER_HEAD是否和BUFFER_TAIL相等，若相等（ZF旗號爲1），則表示沒有資料可取，回到步驟①，等待鍵盤發出中斷信號。若 BUFFER_HEAD 不和BUFFER_TAIL相等，就表示緩衝區內有資料可取，進入下個步驟。

③ 首先將BUFFER_HEAD所指的字組內的資料取出到AX內。然後呼叫K4副程式將BUFFER_HEAD值加2，以便指向下一個字組。返回後，將指向下一個字組的值轉到BUFFER_HEAD內，此時BUFFER_HEAD就指向下一個字組了。

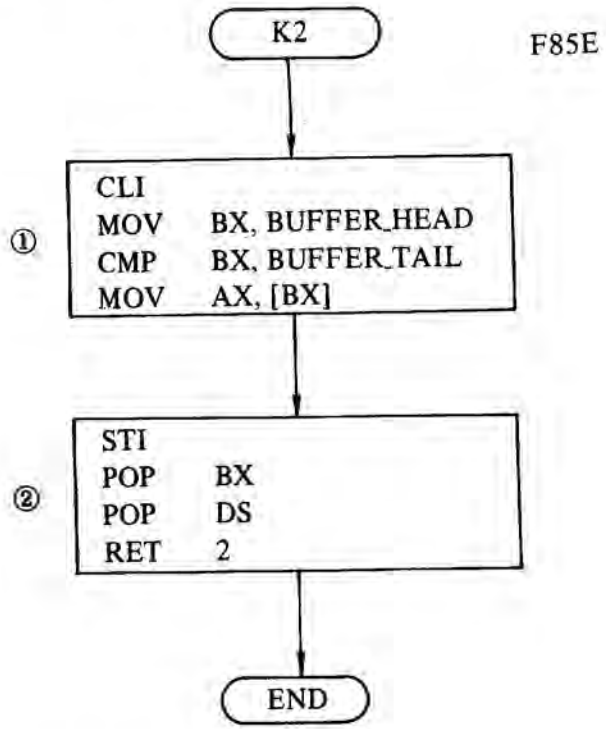
④ 陸續取回BX和DS暫存器的初值後，返回到呼叫INT 16H的程式去。



首先將BX內的值加2，以便指向下一個字組。然後和 KB_

BUFFER_END比較，若不相等（ZF旗號爲0），則表示尚未超過緩衝區的範圍，則直接進入步驟②，若BUFFER_HEAD加2後和KB_BUFFER_END相等，則表示已超過緩衝區的範圍。此時，將BUFFER_HEAD設定成緩衝區的起始位址，以便指向字組0。

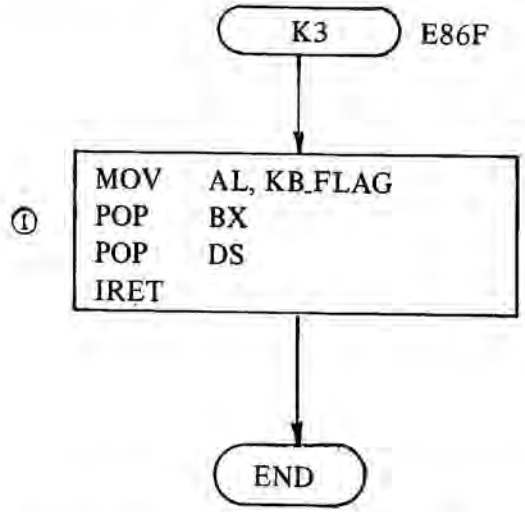
② 返回到呼叫K4的程式去。



① 此步驟比較BUFFER_HEAD是否和BUFFER_TAIL相等，若相等，則ZF旗號爲1，反之，ZF旗號爲0。不論比較的結果爲何，都將BUFFER_HEAD所指的字組內的值取出到AX內。但是並不增加BUFFER_HEAD。

② 首先設定IF旗號爲1，然後陸續取出BX和DS的值後，執行RET 2指令，返回到呼叫INT 16H的程式去。要使用RET 2的原因是IRET會取出原先保存在堆疊中的旗號暫存器，而破壞步驟①的結果，所以要執行RET 2指令，只取回返回位址，而跳過旗號暫存器。請讀者自行查閱RET和IRET指令。

KEYBOARD_IO的最後一項功能是讀KB_FLAG內的值，以便知道那些特別鍵被按下了。



① 從KB_FLAG內取出資料放到AL內後，即返回到呼叫INT 16H的程式去。

KB_FLAG內的值和特殊鍵被按下的關係如下頁圖。

KB_FLAG	KEY
80H	Insert
40H	Caps Lock
20H	Num Lock
10H	Scroll Lock
08H	Alternate shift
04H	Control shift
02H	Lf Left shift
01H	Right shift

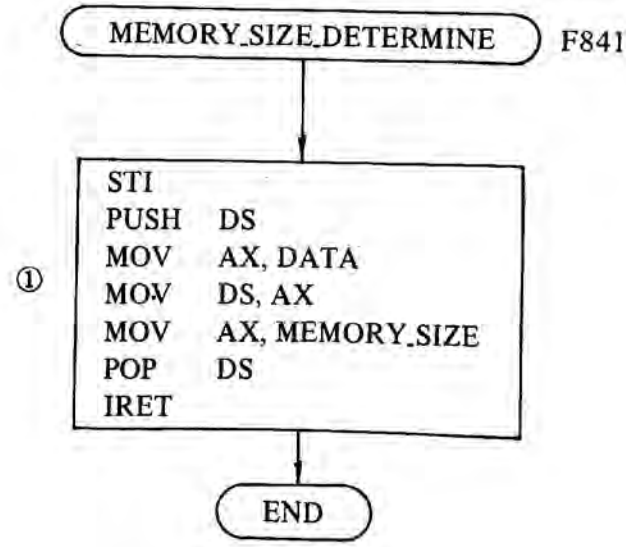
註：KB_FLAG的值由KB_INT設定。

第八章

其他副程式

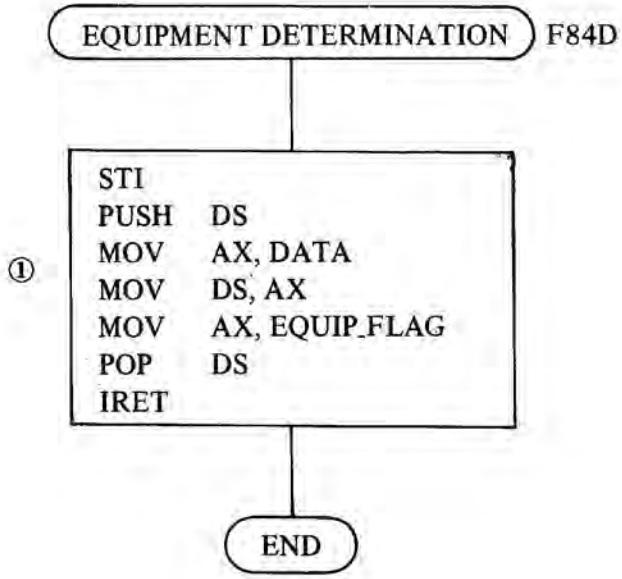
在IBM PC上的BIOS內尚有幾個小程式，我們尚未討論，它們是MEMORY_SIZE_DETERMINE(INT 12H)、EQUIPMENT_DETERMINATION（INT 11H）、TIMER_INT、TIME_of_DAY（INT 1AH）和PRINT_SCREEN（INT 5H）。這些程式雖然短，但却很重要。在本章中，將陸續討論這5個程式。我們首先看MEMORY_SIZE_DETERMINE。

記憶體大小決定(MEMORY SIZE DETERMINE)



① 這個程式只有一個步驟，那就是將MEMORY_SIZE 內的值取出，放到AX內後返回到呼叫INT 12H的程式去。MEMORY_SIZE由POST來設定（TEST.11）。
MEMORY_SIZE內的值即表示出系統上有若干K記憶體。例如：MEMORY_SIZE內的值爲0040H，即表示有64K，0100H表示有256K，依此類推。請讀者自行複習 POST中的TEST.11。

裝備決定(EQUIPMENT DETERMINATION)

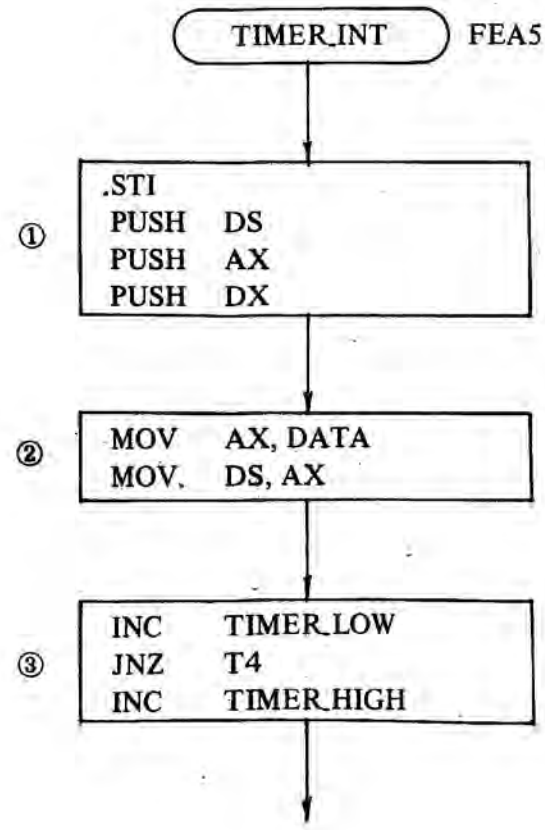


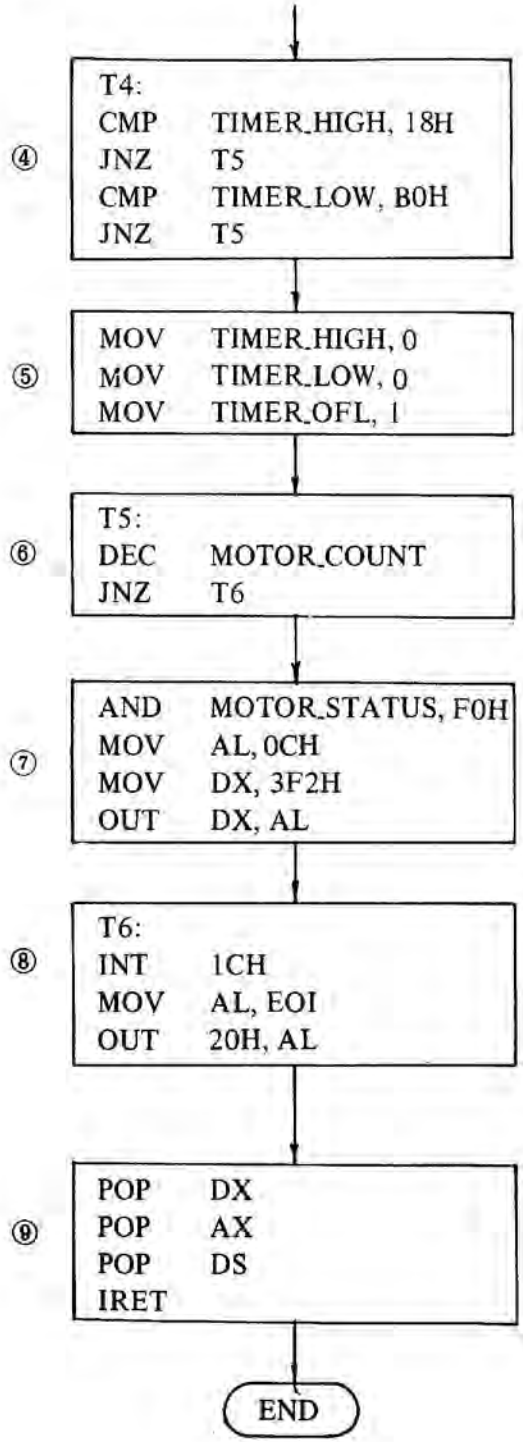
① 這個程式也只有一個步驟，那就是將EQUIP_FLAG 內的值放到AX內後，返回到呼叫INT 11H的程式。EQUIP_FLAG也是由POST來設定（TEST.08和TEST.14）。我們來看EQUIP_FLAG內各位元的意義。

BIT 15, 14 = number of printers attached
BIT 12 = game control card
BIT 11, 10, 9 = numbers of RS232 card attached
BIT 7, 6 = number of diskette drive
BIT 5, 4 = video mode
BIT 3, 2 = planar ram size
BIT 0 = IPL from diskette

請讀者自行複習POST中的TEST.08和TEST.14。

時序(TIMER INT)





TIMER_INT副程式是由8253的通道0發出信號給8259（大約每秒18.2次），然後由8259發出中斷信號給8088.8088接到中斷信號後，即跳到此程式。

此程式有二個主要的工作：

(a) 計時：既然是大約每秒處理此程式18.2次，那麼64K次（65536）大約是3600秒，也就是一小時，當然還有一些尾數。TIMER_HIGH記載小時數，TIMER_LOW記載秒數。若超過24小時，則必須將TIMER_HIGH和TIMER_LOW都設定成0，並將TIMER_OFL設定成1，以便表示超過24小時。

(b) 停掉磁碟機的馬達。在INT 13H中並沒有將磁碟機的馬達關掉的步驟。當一個程式用完磁碟機後，系統並不立即停掉磁碟機的馬達，以免另外的程式接著用磁碟機時，要重新起動馬達；但也不能始終讓馬達旋轉。在IBM PC上，磁碟機等待2秒鐘（馬達等待爲25H），若沒有其它程式繼續使用的話，即停掉磁碟機的馬達。

我們現在來看此程式是如何寫的：

① 首先設定IF旗號爲1，並陸續保存DS，AX和DX暫存器的初值，以免被破壞。

② 此步驟將DS設定成DATA（0040H）。

③ 將TIMER_LOW內的值加1（TIMER_LOW內的值要除以18.2才是眞正的秒數），並測試是否爲0，若爲0（ZF旗號爲1），則表示已經3600秒了（一小時），要將TIMER_HIGH內的值加1；若TIMER_LOW內的值不爲0（ZF旗號爲0），則直接進入下個步驟。

④ 此步驟首先測試TIMER_HIGH內的值是否爲18H(24)，若

不爲18H（ZF旗號爲0），則跳入步驟⑥。若TIMER_HIGH內的值爲18H，則繼續測TIMER_LOW內的值是否爲B0H（即尾數）。若TIMER_LOW尚未到達B0H，則同樣的進入步驟⑥，若TIMER_LOW已到達B0H（即表示開機已24小時），則進入下個步驟。

⑤ 將TIMER_HIGH和TIMER_LOW都設定成0，並設定TIMER_OFL爲1，以便表示已超過24小時。

⑥ 此步驟將MOTOR_COUNT內的值減一，若不爲0，則跳入步驟⑧。若MOTOR_COUNT內的值已成爲0（即表示自從上一個程式使用磁碟機後，2秒鐘內沒有其它的程式來使用磁碟機），則需要停掉磁碟機的馬達，進入下個步驟。

⑦ 首先將MOTOR_STATUS的低半位元組（位元3到位元0）清除爲0，以便表示沒有磁碟機的馬達是旋轉的，然後將0CH寫到3F2H去，停掉馬達。

⑧ 此步驟呼叫INT 1CH副程式，此副程式由使用者自行設定，若使用者希望每次TIMER_INT產生時，都去處理某個程式，則可以將此程式的起始位址放到實際位址70H～73H。然後將20H（EOI）寫到8259去（I／O埠值爲20H），以便表示做完TIMER_INT了。

⑨ 陸續取回DX，AX和DS的初值後，返回到被打斷的程式去。

Time of day (INT 1AH)

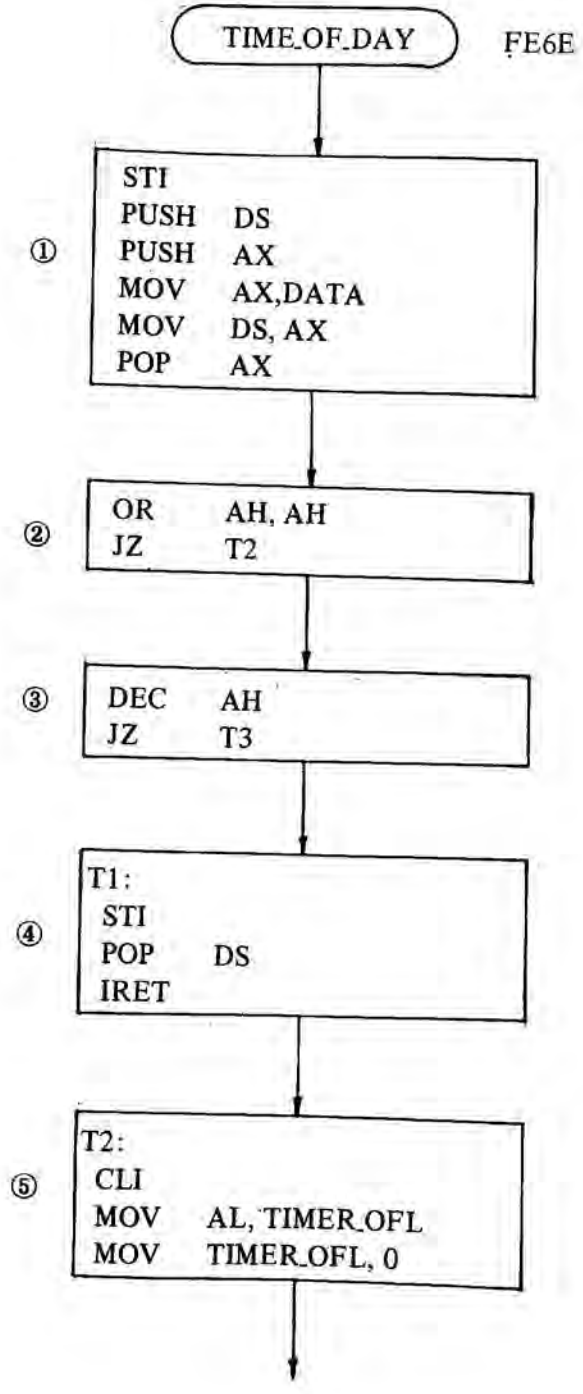
This routine allows the clock to be set/read

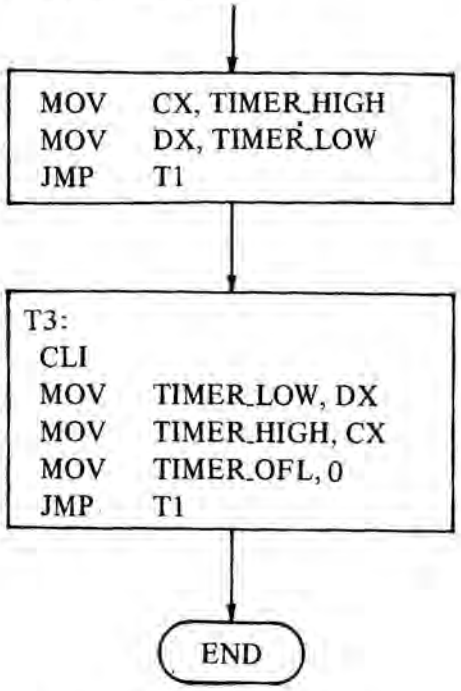
Input : (AH) = 0 Read the current clock setting

(AH) = 1 set the current clock

CX = high portion of count

DX = low portion of count





① 首先設定IF旗號爲1，並保存DS的值，然後將DS 設定成DATA（0040H）。

② 此步驟檢查AH內的值是否爲0，若爲0（ZF旗號爲1），則表示要Read clock，跳入步驟⑤。若AH內的值不爲0，進入下個步驟。

③ 此步驟檢查AH內的值是否爲1，若爲1（ZF旗號爲1），則表示要Set clock，跳入步驟⑦。若AH內的值大於1，則表示AH內的值超過範圍，進入下個步驟。

④ 再一次設定IF旗號爲1，並取回DS的初值後，返回到呼叫INT 1AH的程式。

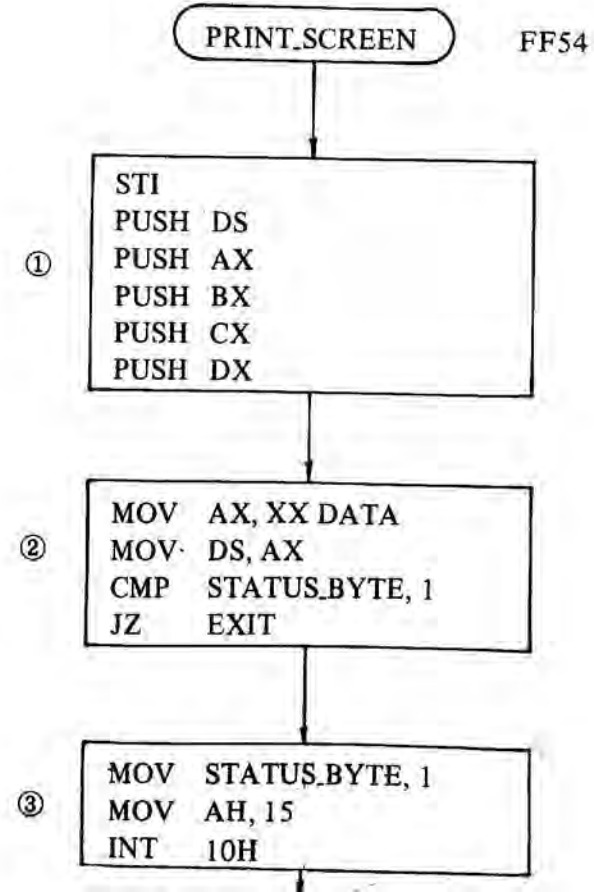
⑤ 進入此步驟是因爲要Read clock。設定IF旗號爲0，以免被中斷。然後將TIMER_OFL內的值放到AL內，並接著清除TIMER_OFL。

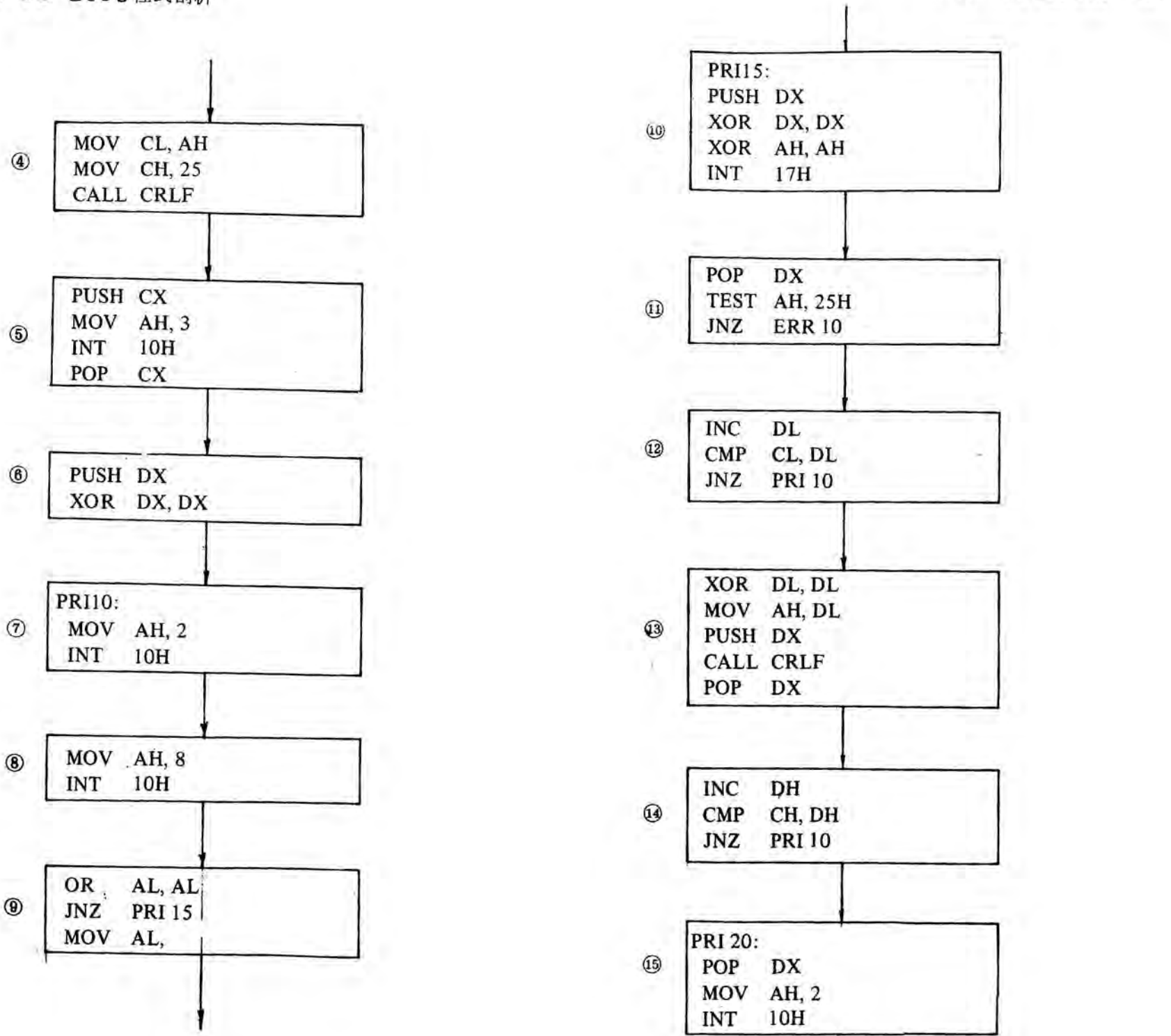
⑥ 將TIMER_LOW內的值放到CX內，並將TIMER_HIGH內的值放到DX內後，經由T1返回到呼叫INT 1AH的程式去。

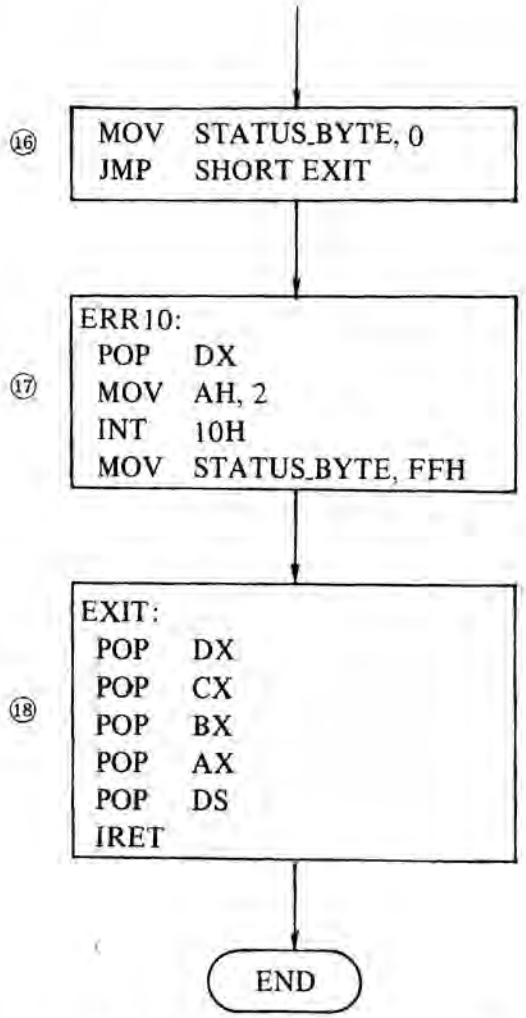
⑦ 進入此步驟是因爲要Set clock，設定IF旗號爲0，以免被中斷。然後將DX內的值放到TIMER_LOW，將CX內的值放到TIMER_HIGH，並清除TIMER_OFL後，經由T1返回到呼叫INT 1AH的程式去。

列印螢幕(PRINT SCREEN (INT 5H))

當從鍵盤上按下 shift-PrtSc 鍵後，就會跳進此程式將螢幕畫面上的字樣送到印表機，將其印出。




④
MOV CL, AH
MOV CH, 25
CALL CRLF
⑤
PUSH CX
MOV AH, 3
INT 10H
POP CX
⑥
PUSH DX
XOR DX, DX
⑦
PRI10:
MOV AH, 2
INT 10H
⑧
MOV AH, 8
INT 10H
⑨
OR AL, AL
JNZ PRI 15
MOV AL,
⑩
PRI15:
PUSH DX
XOR DX, DX
XOR AH, AH
INT 17H
⑪
POP DX
TEST AH, 25H
JNZ ERR 10
⑫
INC DL
CMP CL, DL
JNZ PRI 10
⑬
XOR DL, DL
MOV AH, DL
PUSH DX
CALL CRLF
POP DX
⑭
INC DH
CMP CH, DH
JNZ PRI 10
⑮
PRI 20:
POP DX
MOV AH, 2
INT 10H



① 首先將 IF 旗號設定爲 1 ，並陸續保存 DS，AX，BX，CX 和 DX 的值於堆疊中。

② 將DS指向XXDATA（ 0050H ），並測試 STATUS_BYTE 內的值是否爲 1 ，若爲 1（ ZF 旗號爲 1 ），則表示 Print Screen 的工作正在進行，必須忽略掉這次的要求，跳進步驟⑱。若 STATUS_BYTE 內的值不爲 1 ，則進入下個步驟。

③ 將 1 填入 STATUS_BYTE 內，表示現在正在做 Print Screen 的工作。然後在 AH 內放入 15 ，呼叫 INT 10H，讀出 VIDEO-STATE ，返回後，AL 內爲模式值，AH 內爲行數，BH 內爲 PAGE 數。

④ 將行數（在AH內）放到 CL 內，並在 CH 內放入 25 ，表示有 25 列。然後呼叫 CRLF 副程式將印表機的印字頭移至下一列的最左端。CRLF 副程式待會討論。

⑤ 此步驟讀出 DX 內現在游標的位置。

⑥ 將現在游標的位置值保存在堆疊中，以便做完 Print Screen 工作後，將游標放回原來的位置。然後將 DX 內的值清除爲 0 ，以便從螢幕畫面的左上角開始印起。

⑦ 此步驟將游標移至所指定的位置去。

⑧ 此步驟讀出游標位置內的字元值於 AL 內。

⑨ 首先檢查 AL 內的值是否爲 0 ，若爲 0（ ZF 旗號爲 1 ），則須將空白的 ASCII 碼（ 20H ）放到 AL 內；若 AL 內的值不爲 0 ，則直接進入下個步驟。

⑩ 首先將指定游標位置的值（在 DX 內先保存在堆疊中），然後 XOR DX，DX 指令指出是使用印表機 0 ，XOR AH，AH 指令指出要列印字元，呼叫 INT 17H 將字元值（在 AL 內）印出。

⑪ 取回游標位置值。然後測試 AH 內的值是否爲 25H ，若 AH 內的值爲 25H，則表示有錯誤產生，跳入步驟⑰，否則進入下個步驟。

⑫ INC DL 指令將游標位置向右移一格，然後 CMP CL ，DL 測試此時游標是否已超過螢幕畫面的右端，若游標已超過畫面的最右端（ ZF 旗號爲 1 ），則進入下個步驟準備印下一列的字元。若游標位置尚未超過畫面的最右端，則回到步驟⑦，繼續印此列的字元值。

⑬ XOR DL,DL指令將行數設定成0（螢幕畫面的最左端），並設定AH內的值爲0後，呼叫CRLF副程式將印表機的印字頭移至下一列的最左端。

⑭ 將列數加1（INC DH），以便要印下一列的字元，CMP CH，DH指令測試是否已將整個畫面印完，若尚未（ZF旗號爲0），則回到步驟⑦，繼續印字元。若已印完整個畫面（ZF旗號爲1），則進入下個步驟。

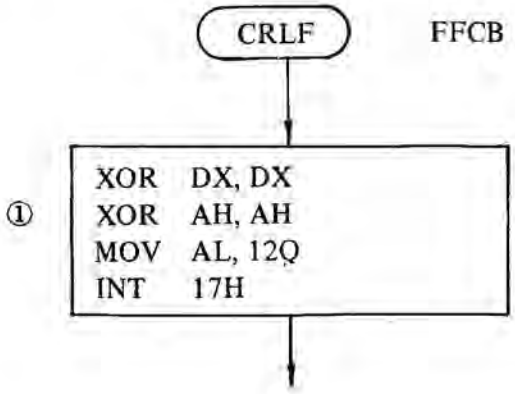
⑮ 取回游標位置的初值後，在AH內放入2，呼叫INT 10H將游標放回原來的位置。

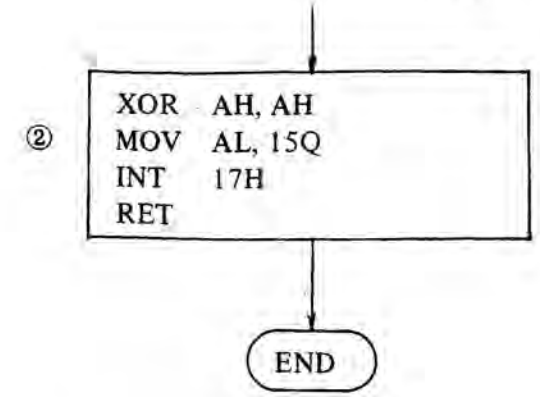
⑯ 在狀態位元組內放入0，以便表示做完Print Screen的工作，然後跳入步驟⑱。

⑰ 進入此步驟是因爲印字時有錯誤產生。同樣地，取回游標位置的初值後，在AH內放入2，呼叫INT 10H將游標放回原來的位置。然後在STATUS_BYTE內放入FFH，以便表示有錯誤產生。

⑱ 陸續取回DX,CX,BX,AX和DS的初值後，返回到被中斷的程式去。

我們繼續看CRLF副程式。





① 此步驟送一個換列（line feed）信號給印表機。XOR DX，DX表示爲印表機0。XOR AH,AH表示印字元，在AL內放入12Q（Q爲8進位的表示法，12Q即爲0AH）表示爲換列，呼叫INT 17H將印字頭移至下一列（並不一定在最左端）。

② 此步驟送一個歸位（carriage return）信號給印表機。15Q即爲0DH，爲歸位的ASCII碼，呼叫INT 17H副程式將印字頭移至最左端。然後返回到呼叫此程式的程式。

好了，我們已將IBM PC的BIOS程式內大部份的程式討論完了（除了KB_INT和cassette_IO之外），相信這本書對讀者是有幫助的。IBM公司當初寫BIOS時，放入許多不需要的指令，以便做爲捉仿冒者的證據。例如，在Print Screen中步驟⑱中的MOV AH,DL指令。若讀者想要重寫BIOS程式，就必須相當小心這些指令了。

附錄

ROM BIOS LISTINGS

```
LOC OBJ                 LINE    SOURCE

                          1     $TITLE(BIOS FOR THE IBM PERSONAL COMPUTER XT)
                          2
                          3     ;--------------------------------------------------------------------
                          4     ;       THE  BIOS ROUTINES ARE MEANT TO BE ACCESSED THROUGH         :
                          5     ;       SOFTWARE INTERRUPTS ONLY.  ANY ADDRESSES PRESENT IN         :
                          6     ;       THE LISTINGS  ARE INCLUDED  ONLY FOR  COMPLETENESS,         :
                          7     ;       NOT FOR  REFERENCE.   APPLICATIONS WHICH  REFERENCE         :
                          8     ;       ABSOLUTE   ADDRESSES   WITHIN   THE   CODE  SEGMENT         :
                          9     ;       VIOLATE THE STRUCTURE AND DESIGN OF BIOS.                   :
                         10     ;--------------------------------------------------------------------
                         11
                         12     ;--------------------------------------
                         13     ;               EQUATES                 :
                         14     ;--------------------------------------
0060                     15     PORT_A          EQU     60H             ; 8255 PORT A ADDR
0061                     16     PORT_B          EQU     61H             ; 8255 PORT B ADDR
0062                     17     PORT_C          EQU     62H             ; 8255 PORT C ADDR
0063                     18     CMD_PORT        EQU     63H
0020                     19     INTA00          EQU     20H             ; 8259 PORT
0021                     20     INTA01          EQU     21H             ; 8259 PORT
0020                     21     EOI             EQU     20H
0040                     22     TIMER           EQU     40H
0043                     23     TIM_CTL         EQU     43H             ; 8253 TIMER CONTROL PORT ADDR
0040                     24     TIMER0          EQU     40H             ; 8253 TIMER/CNTER 0 PORT ADDR
0001                     25     TMINT           EQU     01              ; TIMER 0 INTR RECVD MASK
0008                     26     DMA08           EQU     08              ; DMA STATUS REG PORT ADDR
0000                     27     DMA             EQU     00              ; DMA CH.0 ADDR. REG PORT ADDR
0540                     28     MAX_PERIOD      EQU     540H
0410                     29     MIN_PERIOD      EQU     410H
0060                     30     KBD_IN          EQU     60H             ; KEYBOARD DATA IN ADDR PORT
0002                     31     KBDINT          EQU     02              ; KEYBOARD INTR MASK
0060                     32     KB_DATA         EQU     60H             ; KEYBOARD SCAN CODE PORT
0061                     33     KB_CTL          EQU     61H             ; CONTROL BITS FOR KEYBOARD SENSE DATA
                         34
                         35     ;--------------------------------------
                         36     ;       8088 INTERRUPT LOCATIONS        :
                         37     ;--------------------------------------
                         38
                         39     ABS0    SEGMENT AT 0
                         40     STG_LOC0        LABEL   BYTE
                         41             ORG     2*4
                         42     NMI_PTR         LABEL   WORD
                         43             ORG     5*4
                         44     INT5_PTR        LABEL   WORD
                         45             ORG     8*4
                         46     INT_ADDR        LABEL   WORD
                         47     INT_PTR         LABEL   DWORD
                         48             ORG     10H*4
                         49     VIDEO_INT       LABEL   WORD
                         50             ORG     10H*4
                         51     PARM_PTR        LABEL   DWORD           ; POINTER TO VIDEO PARMS
                         52             ORG     18H*4
                         53     BASIC_PTR       LABEL   WORD            ; ENTRY POINT FOR CASSETTE BASIC
                         54             ORG     01EH*4                  ; INTERRUPT 1EH
                         55     DISK_POINTER    LABEL   DWORD
                         56             ORG     01FH*4                  ; LOCATION OF POINTER
                         57     EXT_PTR LABEL   DWORD                   ; POINTER TO EXTENSION
                         58             ORG     400H
                         59     DATA_AREA       LABEL   BYTE            ; ABSOLUTE LOCATION OF DATA SEGMENT
                         60     DATA_WORD       LABEL   WORD
                         61             ORG     0500H
                         62     MFG_TEST_RTN    LABEL   FAR
                         63             ORG     7C00H
                         64     BOOT_LOCN       LABEL   FAR
                         65     ABS0    ENDS
                         66
                         67     ;--------------------------------------
                         68     ; STACK -- USED DURING INITIALIZATION ONLY      :
                         69     ;--------------------------------------
                         70
                         71     STACK   SEGMENT AT 30H
0000 (128                72             DW      128 DUP(?)
     ????
     )
0100                     73     TOS     LABEL   WORD
                         74     STACK   ENDS
                         75
                         76     ;--------------------------------------
                         77     ;       ROM BIOS DATA AREAS
```

```
LOC OBJ                 LINE    SOURCE
                         78     ;--------------------------------------
                         79
                         80     DATA    SEGMENT AT 40H
0000 (4                  81     RS232_BASE      DW      4 DUP(?)        ; ADDRESSES OF RS232 ADAPTERS
     ????
     )
0008 (4                  82     PRINTER_BASE    DW      4 DUP(?)        ; ADDRESSES OF PRINTERS
     ????
     )
0010 ????                83     EQUIP_FLAG      DW      ?               ; INSTALLED HARDWARE
0012 ??                  84     MFG_TST         DB      ?               ; INITIALIZATION FLAG
0013 ????                85     MEMORY_SIZE     DW      ?               ; MEMORY SIZE IN K BYTES
0015 ??                  86     MFG_ERR_FLAG    DB      ?               ; SCRATCHPAD FOR MANUFACTURING
                         87                     DB      ?               ; ERROR CODES
                         88
                         89     ;--------------------------------------
                         90     ;       KEYBOARD DATA AREAS             :
                         91     ;--------------------------------------
                         92
0017 ??                  93     KB_FLAG         DB      ?
                         94
                         95     ;----- SHIFT FLAG EQUATES WITHIN KB_FLAG
                         96
0080                     97     INS_STATE       EQU     80H             ; INSERT STATE IS ACTIVE
0040                     98     CAPS_STATE      EQU     40H             ; CAPS LOCK STATE HAS BEEN TOGGLED
0020                     99     NUM_STATE       EQU     20H             ; NUM LOCK STATE HAS BEEN TOGGLED
0010                    100     SCROLL_STATE    EQU     10H             ; SCROLL LOCK STATE HAS BEEN TOGGLED
0008                    101     ALT_SHIFT       EQU     08H             ; ALTERNATE SHIFT KEY DEPRESSED
0004                    102     CTL_SHIFT       EQU     04H             ; CONTROL SHIFT KEY DEPRESSED
0002                    103     LEFT_SHIFT      EQU     02H             ; LEFT SHIFT KEY DEPRESSED
0001                    104     RIGHT_SHIFT     EQU     01H             ; RIGHT SHIFT KEY DEPRESSED
                        105
0018 ??                 106     KB_FLAG_1       DB      ?               ; SECOND BYTE OF KEYBOARD STATUS
                        107
0080                    108     INS_SHIFT       EQU     80H             ; INSERT KEY IS DEPRESSED
0040                    109     CAPS_SHIFT      EQU     40H             ; CAPS LOCK KEY IS DEPRESSED
0020                    110     NUM_SHIFT       EQU     20H             ; NUM LOCK KEY IS DEPRESSED
0010                    111     SCROLL_SHIFT    EQU     10H             ; SCROLL LOCK KEY IS DEPRESSED
0008                    112     HOLD_STATE      EQU     08H             ; SUSPEND KEY HAS BEEN TOGGLED
                        113
0019 ??                 114     ALT_INPUT       DB      ?               ; STORAGE FOR ALTERNATE KEYPAD ENTRY
001A ????               115     BUFFER_HEAD     DW      ?               ; POINTER TO HEAD OF KEYBOARD BUFFER
001C ????               116     BUFFER_TAIL     DW      ?               ; POINTER TO TAIL OF KEYBOARD BUFFER
001E (16                117     KB_BUFFER       DW      16 DUP(?)       ; ROOM FOR 15 ENTRIES
     ????
     )
003E                    118     KB_BUFFER_END   LABEL   WORD
                        119
                        120     ;------ HEAD = TAIL INDICATES THAT THE BUFFER IS EMPTY
                        121
0045                    122     NUM_KEY         EQU     69              ; SCAN CODE FOR NUMBER LOCK
0046                    123     SCROLL_KEY      EQU     70              ; SCROLL LOCK KEY
0038                    124     ALT_KEY         EQU     56              ; ALTERNATE SHIFT KEY SCAN CODE
001D                    125     CTL_KEY         EQU     29              ; SCAN CODE FOR CONTROL KEY
003A                    126     CAPS_KEY        EQU     58              ; SCAN CODE FOR SHIFT LOCK
002A                    127     LEFT_KEY        EQU     42              ; SCAN CODE FOR LEFT SHIFT
0036                    128     RIGHT_KEY       EQU     54              ; SCAN CODE FOR RIGHT SHIFT
0052                    129     INS_KEY         EQU     82              ; SCAN CODE FOR INSERT KEY
0053                    130     DEL_KEY         EQU     83              ; SCAN CODE FOR DELETE KEY
                        131
                        132     ;--------------------------------------
                        133     ;       DISKETTE DATA AREAS             :
                        134     ;--------------------------------------
003E ??                 135     SEEK_STATUS     DB      ?               ; DRIVE RECALIBRATION STATUS
                        136                                             ; BIT 3-0 = DRIVE 3-0 NEEDS RECAL
                        137                                             ; BEFORE NEXT SEEK IF BIT IS = 0
                        138
0080                    139     INT_FLAG        EQU     080H            ; INTERRUPT OCCURRENCE FLAG
003F ??                 140     MOTOR_STATUS    DB      ?               ; MOTOR STATUS
                        141                                             ; BIT 3-0 = DRIVE 3-0 IS CURRENTLY
                        142                                             ;   RUNNING
                        143                                             ; BIT 7 = CURRENT OPERATION IS A WRITE,
                        144                                             ;   REQUIRES DELAY
                        145
0040 ??                 146     MOTOR_COUNT     DB      ?               ; TIME OUT COUNTER FOR DRIVE TURN OFF
0025                    147     MOTOR_WAIT      EQU     37              ; 2 SECS OF COUNTS FOR MOTOR TURN OFF
                        148
```

```
LOC OBJ                 LINE    SOURCE

075C 07                 1535            POP     ES
075D 59                 1536            POP     CX
075E 5B                 1537            POP     BX
075F C3                 1538            RET
                        1539    WAIT_INT        ENDP
                        1540
0760                    1541    MD_INT  PROC    NEAR
0760 50                 1542            PUSH    AX
0761 B020               1543            MOV     AL,EOI          ; END OF INTERRUPT
0763 E620               1544            OUT     INT_CTL_PORT,AL
0765 B007               1545            MOV     AL,07H          ; SET DMA MODE TO DISABLE
0767 E60A               1546            OUT     DMA+10,AL
0769 E421               1547            IN      AL,021H
076B 0C20               1548            OR      AL,020H
076D E621               1549            OUT     021H,AL
076F 58                 1550            POP     AX
0770 CF                 1551            IRET
                        1552    MD_INT  ENDP
                        1553
                        1554    ;--------------------------------------
                        1555    ; PORTS                               :
                        1556    ;       GENERATE PROPER PORT VALUE    :
                        1557    ;       BASED ON THE PORT OFFSET      :
                        1558    ;--------------------------------------
                        1559
0771                    1560    PORT_0  PROC    NEAR
0771 BA2003             1561            MOV     DX,HF_PORT      ; BASE VALUE
0774 50                 1562            PUSH    AX
0775 2AE4               1563            SUB     AH,AH
0777 A07700             1564            MOV     AL,PORT_OFF     ; ADD IN THE OFFSET
077A 0300               1565            ADD     DX,AX
077C 58                 1566            POP     AX
077D C3                 1567            RET
                        1568    PORT_0  ENDP
                        1569
077E                    1570    PORT_1  PROC    NEAR
077E E8F0FF             1571            CALL    PORT_0
0781 42                 1572            INC     DX              ; INCREMENT TO PORT ONE
0782 C3                 1573            RET
                        1574    PORT_1  ENDP
                        1575
0783                    1576    PORT_2  PROC    NEAR
0783 E8F8FF             1577            CALL    PORT_1
0786 42                 1578            INC     DX              ; INCREMENT TO PORT TWO
0787 C3                 1579            RET
                        1580    PORT_2  ENDP
                        1581
0788                    1582    PORT_3  PROC    NEAR
0788 E8F8FF             1583            CALL    PORT_2
078B 42                 1584            INC     DX              ; INCREMENT TO PORT THREE
078C C3                 1585            RET
                        1586    PORT_3  ENDP
                        1587
                        1588    ;--------------------------------------------
                        1589    ; SW2_OFFS                                  :
                        1590    ;       DETERMINE PARAMETER TABLE OFFSET    :
                        1591    ;       USING CONTROLLER PORT TWO AND       :
                        1592    ;       DRIVE NUMBER SPECIFIER (0-1)        :
                        1593    ;--------------------------------------------
                        1594
078D                    1595    SW2_OFFS        PROC    NEAR
078D E8F3FF             1596            CALL    PORT_2
0790 EC                 1597            IN      AL,DX           ; READ PORT 2
0791 50                 1598            PUSH    AX
0792 E8E9FF             1599            CALL    PORT_1
0795 EC                 1600            IN      AL,DX
0796 240E               1601            AND     AL,2            ; CHECK FOR ERROR
0798 58                 1602            POP     AX
0799 7516               1603            JNZ     SW2_OFFS_ERR
079B 8A2E4300           1604            MOV     AH,CMD_BLOCK+1
079F 80E420             1605            AND     AH,00100000B    ; DRIVE 0 OR 1
07A2 7504               1606            JNZ     SW2_AND
07A4 D0E8               1607            SHR     AL,1            ; ADJUST
07A6 D0E8               1608            SHR     AL,1
07A8                    1609    SW2_AND:
07A8 2403               1610            AND     AL,0110
07AA B104               1611            MOV     CL,4
```

```
LOC OBJ                 LINE    SOURCE

07AC D2E8               1612            SHL     AL,CL           ; ADJUST
07AE 2AE4               1613            SUB     AH,AH
07B0 C3                 1614            RET
07B1                    1615    SW2_OFFS_ERR:
07B1 F9                 1616            STC
07B2 C3                 1617            RET
                        1618    SW2_OFFS        ENDP
                        1619
07B3 30362F31362F3632   1620            DB      '06/16/82'      ; RELEASE MARKER
                        1621
07BB                    1622    END_ADDRESS     LABEL   BYTE
----                    1623    CODE    ENDS
                        1624            END
```