4

旗號與決策

幾乎在前面的所有例子內，我們都假設計算機與外部輸入／輸出設備間幾乎不需有任何同步。因此，輸出口永遠可立即接受計算機所送來的資料，且當計算機執行至程式的 PEEK 命令時，輸入口隨時都有正確的資料可讀取。事實並不盡然。經常，我們都必須處理輸入／輸出設備比計算機速度慢的情況。

4－1　輸入/輸出設備的同步

由於並非所有的輸入／輸出設備隨時都可立即應付計算機的作業（輸入或輸出），因此，計算機與輸入／輸出設備間必須有某種彼此取得同步（協調）的方式。這種同步通常涉及使用稱爲旗號（ flag ）的信號。這些信號用於顯示每一設備的狀態——忙碌或空閒，準備好或未準備好等等。換言之，這些信號即代表了設備的狀態（此乃爲何旗號經常被稱爲狀態旗號的理由）。

爲了舉例說明方便起見，我們假設我們欲將一個設備界面（連接之意）至Apple計算機上。這個設備不定期地供應八位元的資料值給計算機。通常，在準備將資料送給計算機時，這個設備同時亦會產生一個旗幟信號。圖4－1所示即爲這個設備。注意，其以一標準的三態輸入口將資訊送給計算機。READY（已準備好）旗號產生了一個很有趣的問題。那就是，計算機如何監聽或查看READY旗號的狀況，以得知又有新的資料值欲輸入呢？

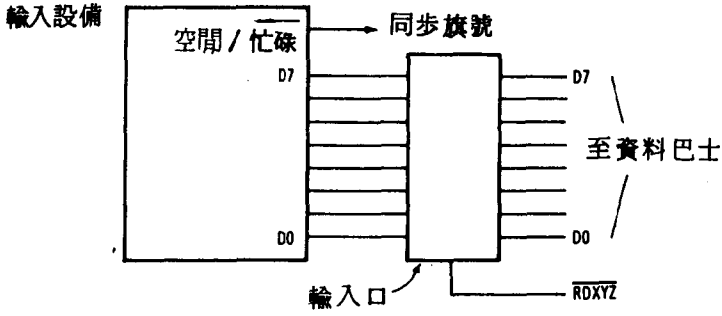

圖4－1　附有同步旗號輸出的簡單輸入設備

前面我們曾經說過，輸入口並無限制說不能傳遞眞正的數值。事實上，計算機根本不知道它所正傳遞的八位元值，譬如01100100_2，是代表數目100，或是一項代表五個設備關三個設備開的狀態訊息。因此，我們可設置另一個輸入口，將輸入設備的狀態訊息傳給計算機。這個輸入口僅用一個位元就可以了，其它七個位元可以不用，或用以顯示其它外部設備的狀態。如此，我們即可以軟體步驟來查驗外部設

備的狀態了。

在以計算機程式檢查旗號狀態時，我們可將程式設計成，教計算機一直等到旗號變成某種狀態時，再進行某種必要的作業；或者，程式亦可設計成週期性地檢查旗號狀態，讓計算機能同時做其它的計算。

不論 BASIC 語言或組合語言，兩者均有能檢查個別旗號狀態或一八位元字組之個別位元的邏輯運算指令。使用這些指令，我們可輕易地測知某一旗號究竟是呈邏輯 1 或邏輯 0，以讓計算機根據旗號的狀態作決策，採取適當的進一步行動。

4－2　邏輯運算與旗號

就測知旗號狀態而言，最有用的運算或許就是 AND 運算了。您應還記得，兩個位元 A 與 B 可以“ AND ”在一起。如圖 4－2 所示，這兩個位元 AND 運算的結果，唯有當

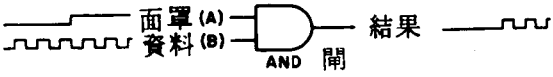

圖 4－2　邏輯 AND 運算：以 DATA(資料) 與 MASK(面罩) 產生 RESULT(結果)。

兩個位元同時爲邏輯 1 時，輸出結果才爲邏輯 1，否則，其它情況輸出結果均爲邏輯 0。另一種看法是，將“ A ”輸入

視爲“罩蓋”（mask）位元，而將“B”輸入視爲“資料”輸入。當我們欲遮蓋掉資料位元時，我們只要令面罩位元爲0，輸出結果就一定爲0。只要面罩位元爲1，資料位元就原封不動呈現於輸出端。如此一來，我們即可隨意地將某些位元遮蓋掉，而讓某些位元順利“通過”。舉個例子而言，假設資料字組00111010中之第五位元（D_5）代表我們所欲檢查之某一設備的狀態，則我們就可使用00100000的面罩，讓這兩者AND，以單獨分出我們所要的位元。這時，經AND運算後，倘若結果爲零，我們即可知道我們所要測知的狀態位元值亦爲邏輯0；否則，其即爲邏輯1。這個測試結果就可作爲程式進一步作決策的根據。

特別記得，在設計BASIC程式時，面罩一定要轉換成十進等效。就剛的例子而言，面罩就是十進數32。

4－3 測知旗號的軟體

界面一旦像圖4－4般地做好，讓各種旗號能被測試時，軟體即可根據這些旗號的狀態作決策。

旗號值	00111010	00011010	11110000	00011111
面　罩	00100000	00100000	00100000	00100000
結　果	00100000	00000000	00100000	00000000

圖4－3　兩個八位元字組，位元對位元作AND運算

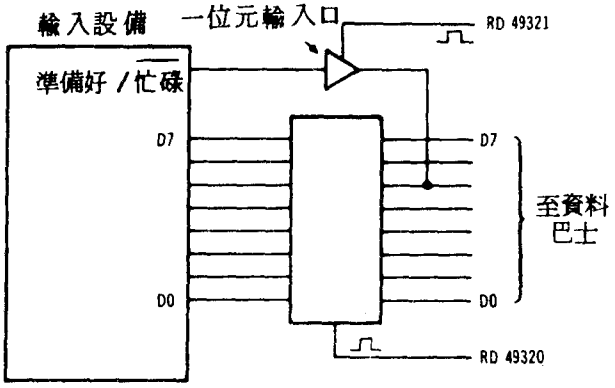

圖 4 － 4　旗號可以軟體測知的完整界面

有些 BASIC 方言具有能作位元對位元 AND 運算（如圖 4 － 3 所示）的邏輯運算指令。在這些情況下，BASIC 程式即可以很簡單的式子，對兩個值均介於 0 至 255 之間的資料字組作邏輯 AND 運算。記得，雖然程式上寫的是十進數，但眞正被“ AND ”的是其二進等效。例題 4 － 1 與 4 － 2 所示即爲以 AND 運算測知自 49321（位址）輸入口之 D_5 位元所輸入的旗號狀態情形。

例題 4 － 1　以邏輯 0 旗號作控制

```
4010  A = PEEK(49321)
4020  IF (A AND 32) = 0 THEN 200
4030  . . . 若旗號爲邏輯 1 ，則繼續執行這兒的指令
```

例題 4 － 2　以邏輯 1 旗號作控制

```
4010  A = PEEK(49321)
4020  IF (A AND 32) > 0 THEN 200
4030  . 若旗號爲邏輯 0 ，則繼續執行這一列上的指令
```

不論那一種情況，每當適當的條件滿足時，程式即跳去做某種此時應該做的工作，譬如，自輸入口輸入資料等等。

可惜，Apple計算機的邏輯運算並不這樣用。在Apple，AND 運算只能對兩個不同的眞假狀況作 AND ，因此，其就很難遮掉一八位元字組之某七個位元，而求知某一位元的狀態。除非您願意花費很多時間，否則，您會發現以組合語言副程式來做這個運算相當方便迅速。由於組合語言常式用來極爲簡便，因此，我們想順便在這裡提一下。事實上，後面我們將提供您許多簡單且易於使用的常式。

4－4　組合語言邏輯運算

6502 微處理器的指令集中含有一個 AND 運算與一個 OR 運算。這兩個指令均對兩個八位元的資料運算，並產生一八位元的結果。因此，我們必須寫一個做這個運算的簡短常式。

由於Apple在第 03H 個記憶頁上有一些“空閒”可讀寫記憶位置，因此，我們選擇將這些常式置於這一頁上。如此，常式即可與您所使用之計算機的記憶器大小無關。這個常式的全部列表即如表 4－1 所示。注意，表中特別列出了十六進與十進位址與資料／指令值。您即使不懂組合語言程式設計，同樣亦可使用這個常式。不過，爲了讓您知道這個程式如何發生作用，我們特別作了一些說明。

常式使用三個可讀寫記憶位置暫時儲存 MASK

表 4－1　組合語言邏輯副程式

位址位元組		資料位元組		
十六進	十進	十六進	十進	
0300	768	—	—	面罩位元組置於此。
0301	769	—	—	資料位元組置於此。
0302	770	—	—	答案置於此。
0303	771	48	72	PHA A暫存器內含存起
0304	772	AD	173	LDA 自面罩位置取入A
0305	773	00	0	
0306	774	03	3	
0307	775	2D	45	AND Reg A 與 DATA *
0308	776	01	1	
0309	777	03	3	
030A	778	8D	141	STA 結果存入
030B	779	02	2	答案位置。
030C	780	03	3	
030D	781	68	104	PLA 取回A 之內含
030E	782	60	96	RTS 回至 BASIC

* OR運算時代換成 ODH 或十進數 13。

BYTE、與 ANSWER 等三個資料位元組。MASK 位置儲存面罩位元組，BYTE 位置儲存欲運算的位元組。邏輯運算後，所得結果則儲存於 ANSWER 的位置上。

使用這個常式前，您必須將面罩位元組存入位址 768 的記憶位置，且將欲運算的資料位元組存入位址 769 的記憶位置。這可以 POKE 命令達成。這些資料存好後，您只要叫用（call）這個組合語言副程式，所需運算即會自動完成。至於，如何叫用這個副程式呢？

從 BASIC 叫用一個組合語言副程式並不難。在 Apple 計算機內，您只需將一個三位元組的跳越（jump）指令存入位址 10，11，與 12（表示成十六進制則爲 OA，OB，與 OC）三個連續記憶位置就可以了。由於副程式儲存於位址 771（＝ 0303H）起之記憶位置，因此，這三個位置所

必須儲存的資訊分別爲：位址 10 存 76 ，位址 11 存 3 ，位址 12 亦存 3 。這三個位置的資訊一旦存好了，您即可以 USR 函數叫用這個組合語言副程式。當然，您必須先將面罩與欲運算之資料位元組存入剛剛提過的位置，然後才能使用 USR 函數。這個情形即如例題 4 － 3 所示。

例題 4 － 3　叫用邏輯運算副程式

```
1590  POKE 768,32: POKE 769,129
1594  Q = USR(5)
```

就例題 4 － 3 之例子而言， 32 爲面罩位元組，而 129 則爲欲與 32 作 AND 的資料位元組。Q 乃一使用 USR 函數所必須的“虛擬”（ dummy ）變數，且 5 亦爲一毫不影響副程式的“虛擬”值。只要在其它地方不用，則 Q 可代換成任何一個變數，同樣地，5 亦可換成任何其它數值，譬如說 0 。

叫用過組合語言副程式後，邏輯 AND 運算的結果就儲存於位址 770 的記憶位置內。這時，您即可以 PEEK 命令獲得這項結果。例題 4 － 4 所示即爲這個副程式的完整用法。當然，我們假設組合語言副程式已事先儲存在記憶器內，譬如使用監督程式。程式的第一列述句將三位元組的跳越指令存入位址 10 ， 11 ，與 12 三個連續記憶位置內。然後，第二列述句再將面罩位元組與資料位元組分別存入適當位置。您可看出，欲運算的資料位元組以 PEEK 命令由一輸入口獲得。

例題 4 － 4　邏輯運算副程式的完整用法

```
2030  POKE 10,76: POKE 11,3: POKE 12,3
2040  POKE 768,32: POKE 769,PEEK(49321)
2050  Q = USR(7)
2060  IF PEEK(770) > 0 THEN 3460
2070  . . . 若旗號爲 0 ，則繼續執行此一列指令
```

只要將 AND 運算的運算碼（ op-code ）2DH 改成 OR 運算的運算碼 ODH ，剛剛的副程式即可做邏輯 OR 運算。同樣地，這亦可在副程式使用前以一 POKE 命令達成。由此可見，表 4 － 1 所示的副程式既可做 AND 運算，亦可做 OR 運算。

讀者您應自己會使用 Apple 計算機的監督程式（ monitor），將我們所要的副程式取入可讀寫記憶器內（第 03H 頁）。監督程式的用法請參閱參考手冊。事實上，您亦可以 12 個 POKE 命令將這個副程式存入記憶器內，不過，這種方法極易造成錯誤。因此，勸讀者還是少用。

很可惜，由於 Apple 之 BASIC 無法直接使用這些邏輯運算，因此，您得必須假手於組合語言副程式。不過，所幸這個組合語言副程式倒還十分容易使用。其用法就是以上我們所舉例說明的。

4 － 5　複雜旗號

至此，您或許會問，倘若圖 4 － 4 之輸入設備上的旗號用以顯示輸入設備有一八位元資料欲送出，那輸入設備又怎

能知計算機是否已讀取其所欲送出的資料呢？這個問題的答案是這樣的。計算機在讀取資料後，可送一個信號給輸入設備，告訴它，資料已被讀取。而我們一般的作法都以這個信號“清除”掉旗號。這個旗號清除動作可另以一個信號達成，亦可以輸入口之控制信號一併達成。圖 4 － 5 所示即爲這種狀況，而圖 4 － 6 所示則爲一簡單的時序圖。

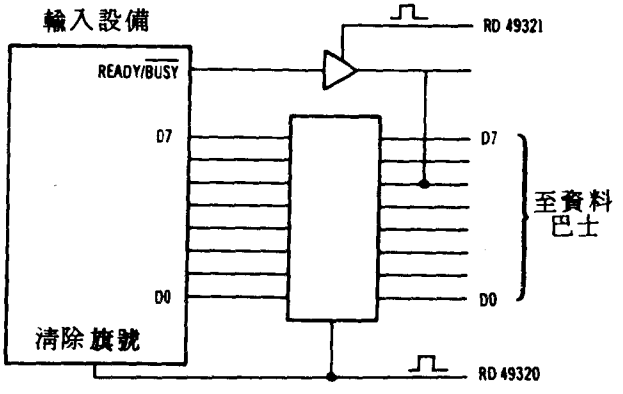

圖 4 － 5　以計算機產生之脈衝清除旗號的完整旗號電路

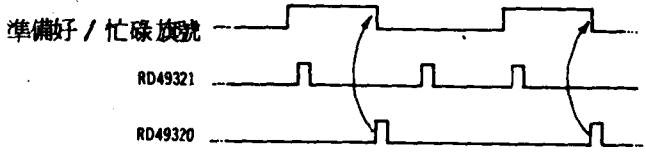

圖 4 － 6　旗號之時序圖

當旗號爲邏輯 1 時，即表示設備有資料要送給計算機。

RD 49321脈衝代表旗號狀態資訊傳遞至計算機。當計算機測試旗號且發現其值爲邏輯1時，其即會執行程式步驟，將資料自設備讀入計算機。這時候，RD 49320脈衝的功用就是在正確的時間使三態緩衝器致能（或動作）。同時，這個脈衝亦負責清除設備內部的旗號。

第二個RD 49321脈衝（見圖4－6）再度讀取旗號狀態，不過，由於這時旗號值爲邏輯0（表設備無資料欲送出），因此，計算機未採取任何進一步行動。不過，當第三個RD 49321脈衝又出現時，由於旗號爲邏輯1，因此，資料再度由設備傳至計算機，而且旗號被清除。例題4－5所示即爲一可用以控制這個界面的簡單程式。假設邏輯AND副程式以及三個位元組的指示器（pointer）均已存入。

例題4－5　簡易的旗號測試程式

```
1050  POKE 768,32: POKE 769,PEEK(49321)
1060  Q = USR(0)
1070  IF PEEK(770) = 0 THEN 50
1080  D = PEEK(49320)
1090  . . . 資料輸入後繼續執行此一指令
```

以這種方式使用旗號的常見設備包括鍵盤、軟性磁碟、類比至數位轉換器、以及其它不定期地產生資料位元組的設備。

4－6　旗號電路

有時，輸入／輸出設備內部並無簡易旗號控制之必要旗號電路，否則，其所產生之邏輯準位，穩定時間就太短，無法爲計算機所正確測知。在這些狀況下，“旗號”可能是一個很短暫的脈衝。事實上，若僅以三態輸入口輸入，有些旗號脈衝根本就短得無法教計算機測知。

在這種狀況下，我們就必須設計一種能“抓住”旗號脈衝，使其稍後能爲計算機所測知的電路。否則，即令計算機幾個微秒就能測試一個旗號位元，其還是經常會“錯過”僅有數微秒長的短暫脈衝。

記憶旗號脈衝一般均使用正反器或鎖住器電路。典型的正反器元件如 SN7474 D型正反器，以及 SN7476 J-K 正反器。這些元件在絕大部份的數位電子學書上都有介紹。

圖 4－7 所示即爲一典型的正反器旗號電路。在這個電路內，輸入設備產生之 READY（資料準備好）脈衝加至正反器之時序輸入，控制將D輸入之邏輯準位傳遞至Q輸出。計算機則經由一輸入口（該輸入口不同於傳遞資料位元組的輸入口）測知Q輸出之準位。誠如前面所說過的，計算機很輕易即可預知旗號位元的狀態。此時，一旦必要的動作都做完之後，資料即由輸入設備輸入至計算機，然後，旗號正反器被清除爲 0。旗號的清除是以在正反器的清除輸入（CR）端加上一邏輯零的脈衝 $\overline{\text{CLEAR}}$ 達成。雖然控制八位元輸入

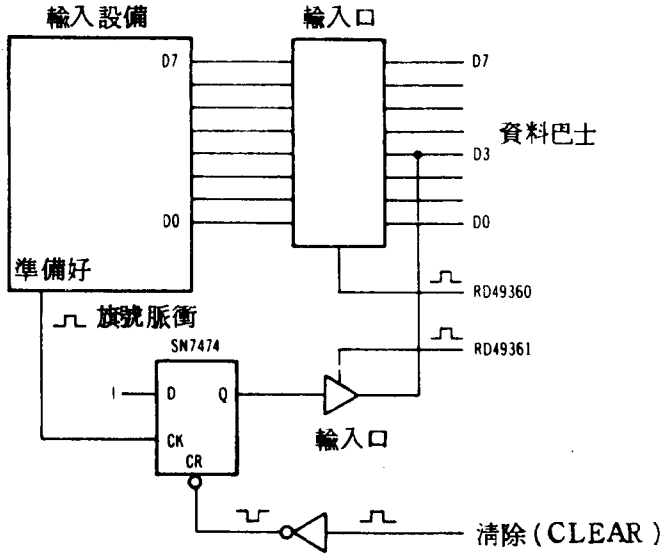

圖 4－7　用以測知旗號脈衝的正反器電路

口的 RD 49360 脈衝亦可用以清除正反器，但我們則將之個別畫成一個信號，以便能畫出如圖 4－8 所示的時序關係圖。

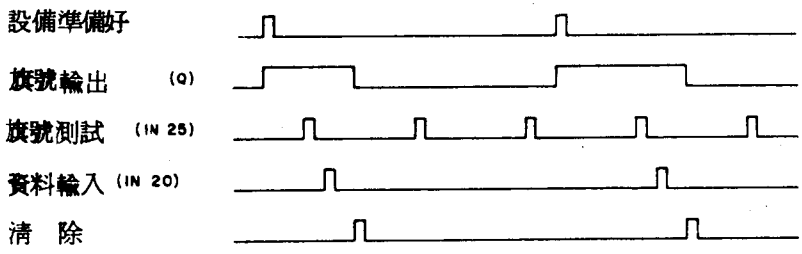

圖 4－8　旗號正反器的時序圖

在這個時序圖內，READY（備好）脈衝設定了正反器

，因此，Q 輸出爲邏輯 1 。計算機自 49361 的輸入口輸入這個狀態旗號訊息，並加以測試。由於旗號狀態爲邏輯 1 ，故計算機執行必要的程式步驟，將輸入設備上的資料輸入計算機，並且清除旗號。雖然 CLEAR 信號可以 POKE 命令以及適當的電路產生，但使用現成的 RD 49360 脈衝可能更容易一點。

就圖 4 － 8 之例子而言，旗號在邏輯 0 狀態時被測試兩次。由於兩次測試均顯示輸入設備無進一步新的資料，故在這一段期間內，無任何資料傳遞或旗號清除作業發生。

本書末了的許多實驗均牽涉到旗號的使用。

4 － 7　多個旗號

許多系統都具有數個必須不斷檢查的旗號。有時，由於某些設備較其它設備重要，必須先受注意，因此，必須建立一套優先順序（ priority ）。由於各旗號位元先後被測試的順序就代表了那些設備比其它設備先受注意，因此，優先順序在程式內很容易建立。圖 4 － 6 所示的程式步驟即爲一循序測試數個旗號位元的例子。這個程式依序測試 D7 至 D5 等幾個位元，因而，在這些位元所對應的設備之間，建立起一套優先順序。

例題 4 — 6　建立優先順序的旗號測試程式

```
300  POKE 769,PEEK(54098):POKE 768,128: Q=USR(0)
305  IF PEEK(770) > 0 THEN 1050
310  POKE 768,64:Q=USR(0)
315  IF PEEK(770) = 0 THEN 20
320  POKE 768,32:Q=USR(0)
325  IF PEEK(770) = 0 THEN 1010
330  . . . 其它位元同理類推
```

在這個例子裡面，D7 位元所代表的旗號爲邏輯 1 時，表示設備準備好。相反地，其它兩個旗號則爲邏輯 0 時表示設備已準備好。程式可再加入其它旗號的測試步驟，而且，旗號位元被測試的順序隨時亦可改變。留意到，有關 AND 運算的資料並未改變，而且，其只需在指令系列一開始時輸入一次即可。

4 — 8　插斷（ Interrupts ）

有時候，輸入 / 輸出設備在一準備好時即需立即受到服務。其根本無法等待計算機慢慢地檢查旗號，然後再根據旗號狀態作決策的幾毫秒鐘。在這種情況下，幾乎所有的計算機都至少具有一個可應付此種立即“需求”的插斷（ interrupt ）輸入。Apple計算機所使用之 6502 處理器晶片上具有兩支插斷輸入接脚：一支爲插斷請求輸入（ IRQ ），另一支爲非罩蓋（ nonmaskable ）插斷輸入（ NMI ）。其中， IRQ 輸入準位觸發（邏輯 0 狀態），而 NMI 輸入則爲邊緣觸發（ edge sensitive ）——受邏輯 1 至邏輯 0

邊緣觸發。基本上，這兩個輸入在Apple計算機內都不用。不過，其在內部界面接點上可以找得到，而且可於再加上週邊設備與界面時使用。

倘若某一設備需要極快速的服務——快到必須使用插斷，則我們就必須非討論不可。由於這超出本書的範圍，因此，若讀者有興趣，希望您進一步參考“6502 程式設計與界面實驗”以及“6502 軟體設計”（儒林圖書公司出版，陳金追譯）兩本書。這兩本書對插斷都有詳細的討論，並附有控制插斷的例題與組合語言程式。

Apple插斷 IRQ 與 NMI 均使用特定的記憶位置，儲存每一插斷之服務（處理）常式的起始位址，在進行插斷處理前，6502 處理器將先“獲取”這些位址。這些位置分別是：位址 FFFAH與 FFFBH（NMI），以及位址 FFFEH與FFFFH（IRQ）之記憶位置。這些位置由於實際都在含BASIC解釋程式（interpreter）與監督程式的唯讀記憶器內，因此，這四個位置的位址是固定的，您永無法加以改變。不過，這些固定位址僅指至其它可讀寫位置罷了，您實際上還是可由這些可讀寫位置上改變插斷服務副程式（interrupt service subroutine）的指示器。有關這些“向量”位置的用法，詳情請看“Apple II 參考手册”。

4－9 結 語

結束本章之前，最後我們還需說幾句話。本章，我們曾

介紹了一個對兩個位元組做邏輯 AND 或 OR 運算的簡單組合語言副程式，以及組合語言副程式叫用指令 USR 的用法。實際上，Apple 計算機之指令集本身即含有一旗號檢查命令：WAIT。這個指令可用以檢查每一個別旗號或一組旗號，同時，其亦可測知邏輯 1 與邏輯 0 旗號。但是，其用法亦有限制。倘若測不到適當的旗號內含，則程式將離開不了旗號檢查作業。此時，爲了重新獲得控制，您必須令計算機重置（ reset ）。同時，您亦無法在旗號（某一或某幾個）值爲 1 時，令控制跳至某一程式部份，而在旗號值爲 0 時令控制跳往另一個方向。若使用 WAIT，那您只有繼續 WAIT（等候）到條件滿足爲止。由於這樣非常沒彈性，因此，我們一直避免使用 WAIT 命令。

若您擴大視野，進一步學習組合語言程式設計，您會發現，叫用組合語言副程式的 USR 命令極有價值。不過，若您僅想叫用一個諸如邏輯 AND 運算的副程式，那您可使用 CALL　指令，並在這個指令之後附上副程式之十進起始位址。譬如，CALL 771 即可叫用前面的邏輯 AND 副程式。當然，在叫用之前，您仍然必須將面罩與資料位元組 POKE　入適當位置。

本章的目的乃在向您介紹一點 Apple 計算機的威力，以及其如何處理不同的工作。相信您有感覺，易途並不總是最有趣或最富教育性的。