BASIC Commander
及
InnoBASIC Workshop

使用手冊

版本 1.0

INNOVATI 與M形圖案為利基應用科技股份有限公司註冊商標

基於對產品的持續改善，本公司得不經通知隨時變更本資料或本資料中所提及之產品。未經本公司書面同意或授權，不可重製、散布本產品局部或全部內容。

免責聲明

使用者在使用本產品所做的任何應用，使用者須自行承擔一切風險。公司對於因使用本產品所生之直接、間接或附帶損害，包括且不限設備損失、人身安全健康損失、利潤信譽損失，不負任何責任。本公司產品不可使用於救生或相關儀器設備。未滿 13 歲兒童須有成人陪同方可使用本產品進行相關實驗。

勘誤

希望使用者會覺得這是一本生動而且實用的使用者手冊。我們花費很多心力於讓這本手冊更加完整而正確的傳達我們希望使用者了解的訊息，然而難免仍有疏漏之處。為了提供使用者手冊提供最新最詳實的資訊，我們會持續改善增補手冊內容。如在本手冊中發現錯誤之處，歡迎利用電子郵件與我們連絡。如有改善任何相關資訊更新皆會揭露於網站上，請經常瀏覽我們的網站，以便獲知最新資訊。

電子郵件：technical@innovati.com.tw
網站：http://www.innovati.com.tw

目錄

系統概述 1

簡介

本系統包含了幾個部份：innoBASIC Workshop 是一個安裝在電腦上的軟體平台，提供使用者以 innoBASIC 程式語言編輯、編譯及下載程式；終端視窗(Terminal Window) 是程式執行時的人機介面及除錯平台；完成的程式碼透過 USB 介面，下載到 BASIC Commander 單板電腦 (Single Board Computer)，這也是系統的核心部份。

BASIC Commander 提供三種資源：第一個是通用的 I/O；第二個是 cmdBUS (TM)，透過它可同時並聯 32 個利基科技的週邊模組。第三個是 DEBUG 介面，用來讓 innoBASIC Workshop 終端視窗 對 BASIC Commander 上傳或下傳資料，DEBUG 介面不僅作為除錯使用，它同時也是一個方便的人機介面。

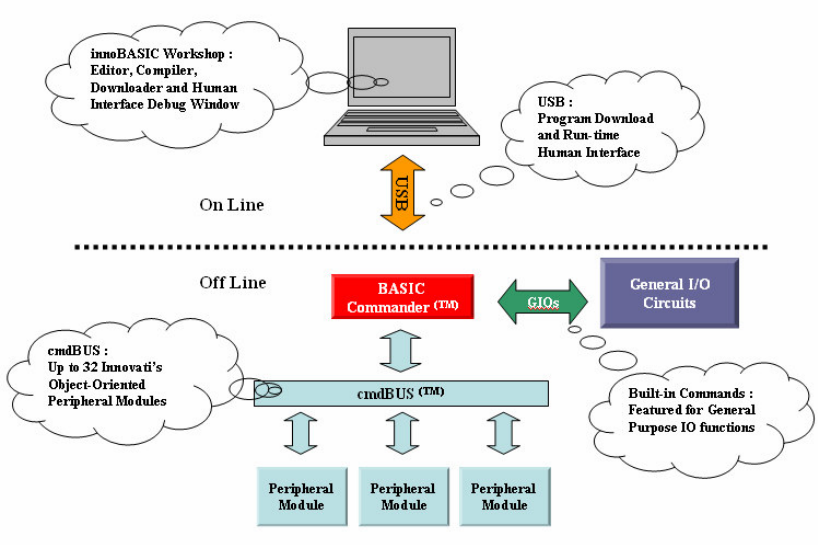


圖 1-1　BASIC Commander 與 and innoBASIC Workshop 系統架構圖

InnoBASIC 程式語言

InnoBASIC 程式語言是利基科技為了提供使用者一個簡單、快速、好用的程式開發工具所開發出來的程式語言。它以大家熟知的 BASIC 程式語言為基礎，加上自己的一些不一樣的特色，可以很容易地用來控制硬體的周邊模組。

BASIC Commander 單板電腦

BASIC Commander 為本系統的核心，它是一個小型的單板電腦(Single-Board

Computer)。在程式編寫過程的除錯階段，或者程式碼的下載時，BASIC Commander 透過 USB 線與 PC 連接，方便它與 PC 間的資料傳遞，但當程式下載完畢，它也可以脫離 PC 而獨立作業。

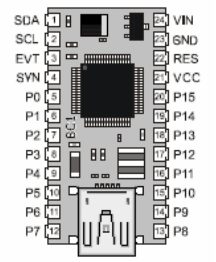


圖 1-2a　24引腳 BASIC Commander

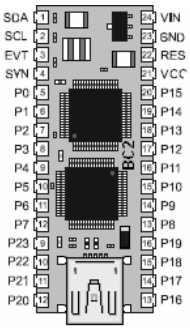


圖 1-2b　32引腳　BASIC Commander

因 BASIC Commander 之零件相當精密，手持時須當心避免靜電破壞。而當要將 BASIC Commander 插入腳座時，請注意方向的正確，以免發生嚴重的破壞。另外，所有的 I/O 腳的輸入電壓請勿超過 6.0 伏特。

周邊模組

模組的使用是利基科技 BASIC Commander 系統的特色之一。利基科技開發的模組包含有 I/O 擴充模組、液晶顯示模組、馬達趨動模組、衛星定位模組等等。每一個模組都有他自己的識別名字，這個識別名字不僅代表他的產品名，同時也提供給編寫程式時呼叫使用。例如，液晶顯示模組的識別名字為“LCD2x16A”。

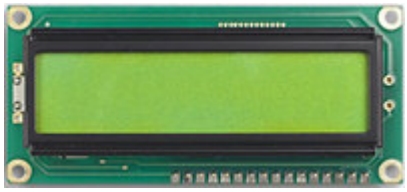


圖 1-3a　LCD2×16A 正面

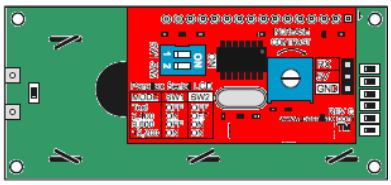


圖 1-3b　LCD2×16A 反面

每一個週邊模組均需要搭配一個兩端有杜邦接頭六線式的排線，以便連接到教育板上的 cmdBUS。使用杜邦排線時請注意正極與負極的方向，避免因方向錯誤導致系統的損壞。

圖 1-4　六線 cmdBUS™ 線

教育板

教育板(Education Board)上有 BASIC Commander 的腳座可供插上 BASIC Commander 單板電腦；有一塊麵包板可讓使用者插上必須的零件，也可以將電源線及 BASIC Commander 的 I/O 接過來使用。

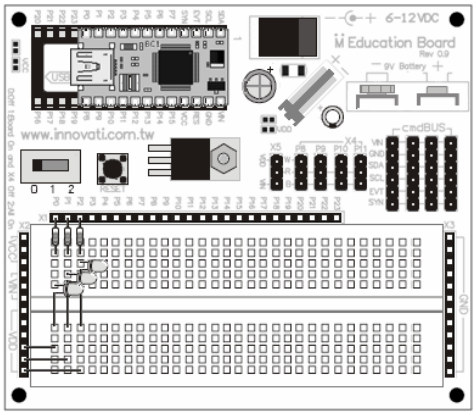


圖 1-5　教育板(Education Board)

教育板基本上並不包含 BASIC Commander，您可以選擇將 24 腳的 BC1 或 32 腳的 BC2 插上腳座使用。請務必依據教育板上的所印的圖案，以正確的位置及方向將 BASIC Commander 插上，若位置或方向錯誤，將導致系統產生無法挽回的損壞。

安裝與開始使用 2

簡介

本章介紹如何安裝與使用本系統。本系統適用於 IBM 或與其相容的個人電腦，Windows 98 或以上的版本/2000/ME/XP/Vista 等作業系統均可使用。建議以光碟片來安裝 innoBASIC Workshop 軟體。電腦須有 USB 接頭，用來下載程式碼及除錯。

安裝 InnoBASIC Workshop 軟體

插入光碟片，依照螢幕指示執行自動安裝，也可以自行到利基科技官方網站(www.innovati.com.tw)下載最新版本的 innoBASIC workshop 軟體。安裝成功後，執行 innoBASIC workshop 將可看到以下的視窗，這就是編寫程式的軟體平台。

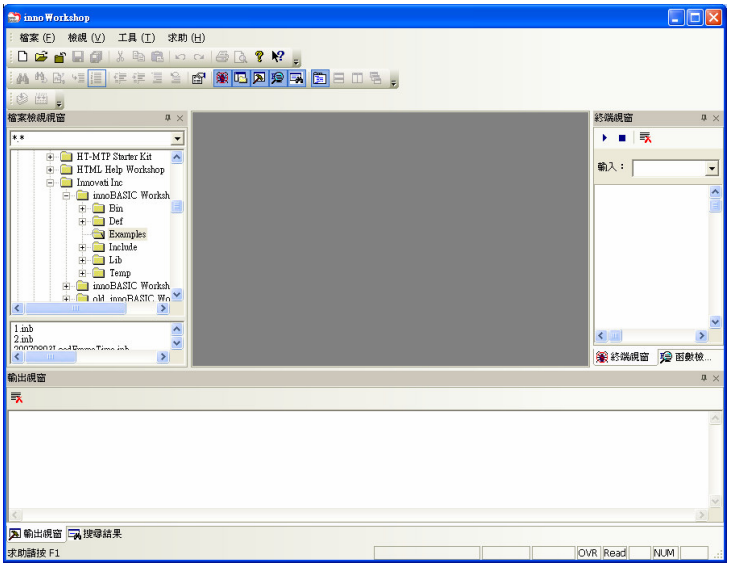


圖 2-1　innoBASIC Workshop 整合開發環境

安裝硬體

將 USB 傳輸線接在電腦 USB 埠與 BASIC Commander 之間。 USB 將提供 BASIC Commander 所需的電源；不過 USB 的電源可能不足以提供使用者的應用電路。如果您的應用電路總電流超過 500mA，那就必須外加電源供應器。

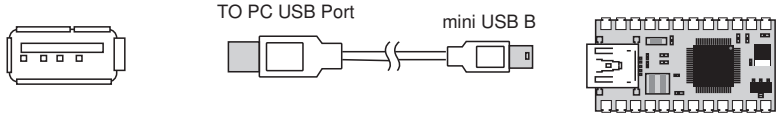


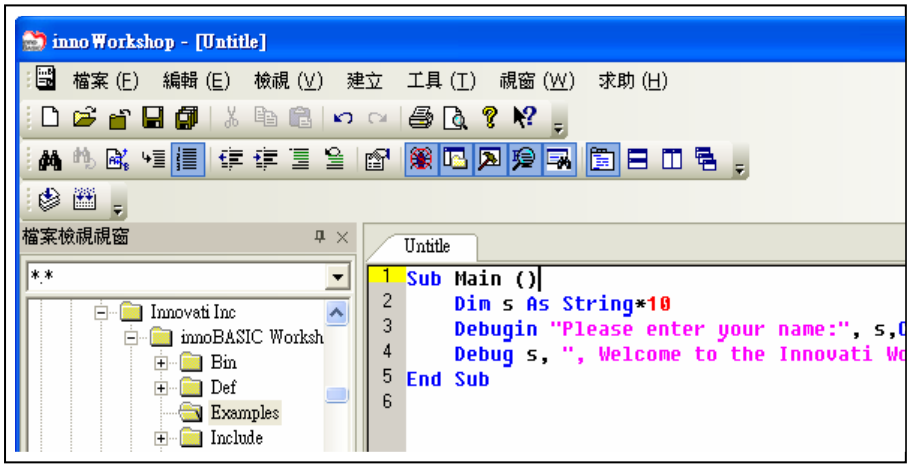
圖 2-2　硬體連接圖

編寫第一個程式

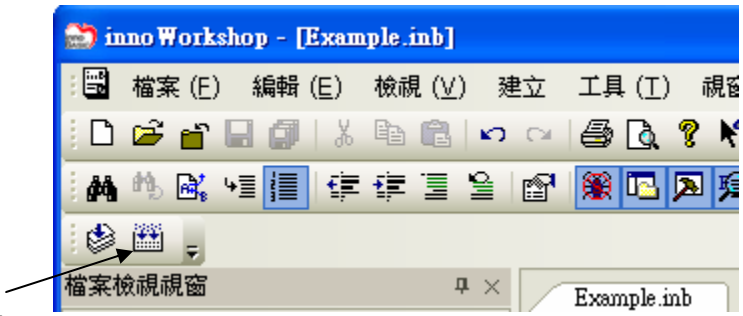
在軟體與硬體成功安裝後,您可以依以下步驟來編寫您的第一個程式了,步驟如下：

1. 從檔案/開新檔案選單或點擊“開新檔案”圖示開啟一個檔案。
2. 在空白的程式編寫區鍵入以下的程式：

```
Sub Main ()
    Dim s As String * 10
    Debugin "Please enter your name:", s,CR
    Debug s, ", Welcome to the Innovati World! "
End Sub
```



3. 點選檔案/儲存檔案選單或點擊標準的”儲存檔案” Icon 圖示以儲存檔案。您可從檔案檢視視窗（File View Window）選擇要放程式的位置。以下的例子會將程式放在 innoBASIC Workshop 檔案夾下面的“Example”檔案夾中。注意程式的副檔名必須為“.inb”，代表是 innoBASIC 的程式。
4. 程式編寫完成後，點選建立選單下的編譯或直接點擊編譯 Icon 圖示來編譯程式。執行程式編譯後，在輸出視窗將顯示結果正確或錯誤的訊息，如果有錯誤訊息，那表示程式必須做除錯或修改。
5. 編譯成功後即可將此程式的”機器碼”透過 USB 線下載到 BASIC Commander，請點選建立選單下的建立或直接點擊建立 Icon 圖示，系統將再一次編譯，且在沒有錯誤時自動執行下載程式機器碼，BASIC Commander 上的綠燈閃爍表示正在下載程式碼。



編譯及下載圖示

圖 2-4　編譯及下載程式碼

6. 下載完成後程式即自動執行。這時可在終端視窗上看到系統要求輸入名字，這時請輸入您的名字並按下 ENTER 鍵，此時系統將回應一個歡迎的訊息。這代表您的 BASIC Commander 硬體已經可以透過 USB 與您在終端視窗上溝通了。

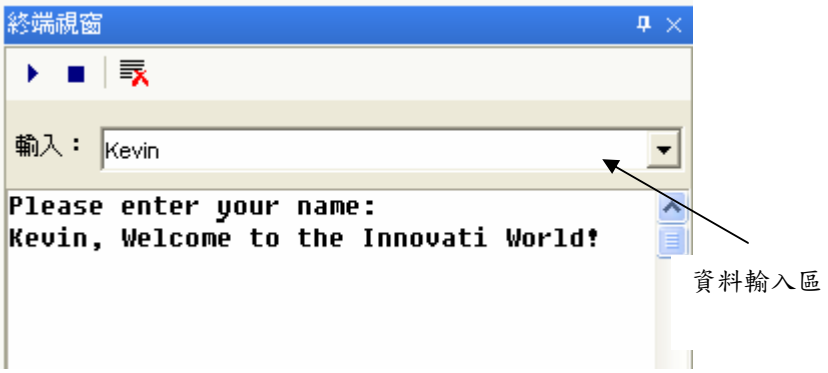


圖 2-5　終端視窗上的歡迎訊息

3 InnoBASIC Workshop 整合開發環境

簡介

InnoBASIC Workshop 是本系統開發程式的工作平台，提供程式的編寫、除錯、編譯、下載等應有的功能，執行 Workshop 之後，除了熟悉的視窗畫面之外，值得注意的是程式編輯視窗及終端視窗，前者是用來編寫程式的視窗，而後者則是用來與硬體 BASIC Commander 溝通的視窗。

視窗各部解說

螢幕視窗被細分成幾個功能視窗，分別簡單說明如下：

檔案檢視視窗

這個視窗顯示可用的檔案夾及其下面的檔案。點擊二次可打開所需的檔案，這時檔案的文字內容會顯示在其右方的程式編輯視窗中。使用者可

以同時打開多個檔案，顯示時會有檔名的標籤掛在檔案內容文字的上方。

程式編輯視窗

這是編寫程式的區域，使用者可以同時打開多個檔案，但在這個視窗上同時只能顯示一個檔案的文字內容，如果要顯示其他檔案的內容，可以點擊上方的檔案標籤。

輸出視窗

當程式被編譯或下載時，這個區域會顯示相關訊息，包括成功，失敗或錯誤等等。

函數檢視視窗

這個視窗可列出主程式中所有的函數或副程式的清單，點擊清單中的名稱可以讓游標直接跳到該函數或副程式中，方便在大程式中找尋函數或副程式的位置。

終端視窗

任何 DEBUG 或 DEBUGIN 命令執行的結果都會顯示在這個區域，使用者透過這個視窗來與硬體 BASIC Commander 做溝通。在終端視窗上方有三個功能 Icon 圖示：Start， Stop 及 Clear。點擊 Start 圖示將對 BASIC Commander 執行重置(Reset)動作，讓程式從第一個命令重新執行；Stop 功能則是讓 BASIC Commander 和這個視窗的溝通(DEBUG 或 DEBUGIN 命令)停止；Clear 的功能只是用來清除終端視窗中的內容，對 BASIC Commander 或程式本身的運作沒有影響。

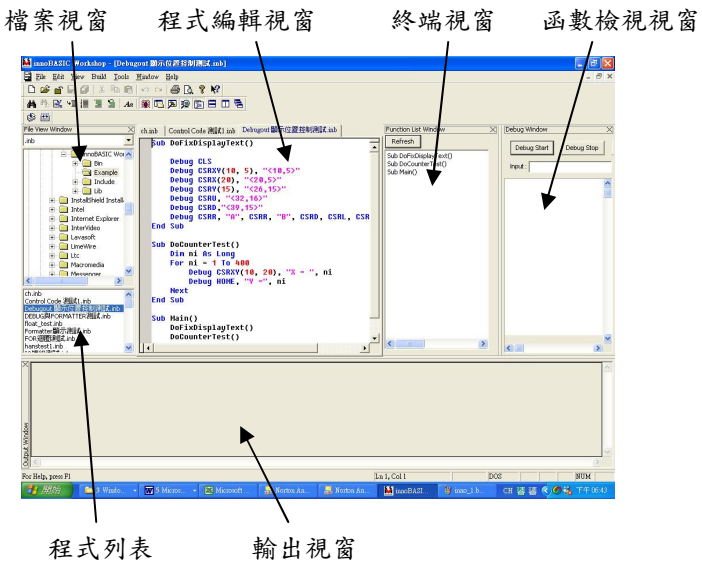


圖 3-1　innoBASIC Workshop 視窗

這些視窗有一些圖示可用來切換顯示與否，提供使用者作較快速的切換。

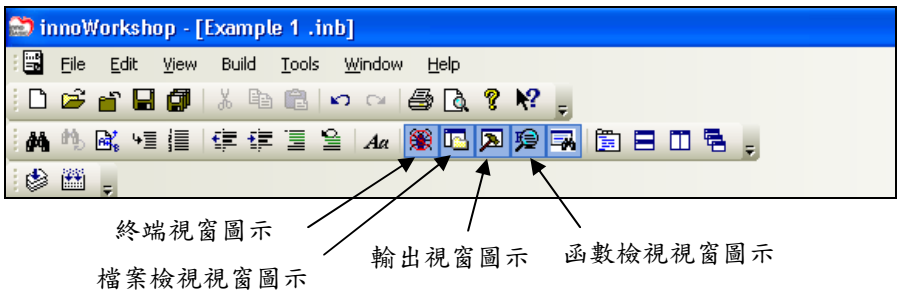


圖 3-2　Workshop Window 圖示

程式開發步驟

如同第二章”編寫第一個程式”中所述，程式開發步驟可整理如下之流程圖：

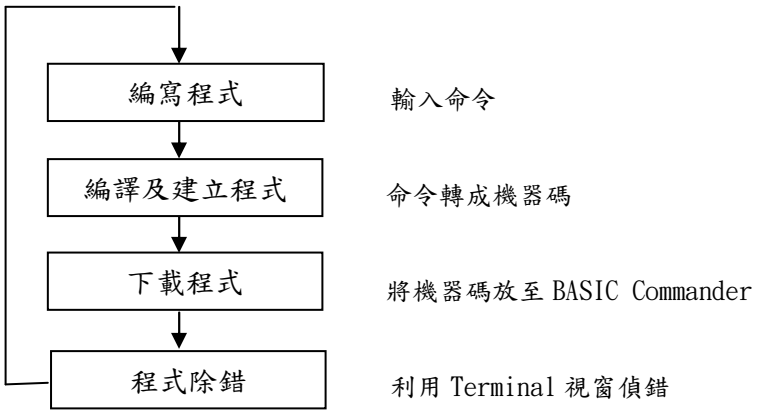


圖 3-3　程式開發步驟

編寫程式

就如同一般簡單文書編輯軟體一樣，您可以在程式編輯視窗中逐行輸入程式命令。

編譯及建立程式

程式在程式編輯視窗編寫完成後，必須先轉成機器碼才能下載到 BASIC Commander。轉換機器碼的過程稱為編譯(Compiling)；系統在轉換機器碼之前會先檢查程式的內容，如果有發現錯誤，例如輸入了錯誤的命令，則系統會產生一個錯誤的訊息並顯示在輸出視窗上，這時只要在錯誤訊息上點擊二次，游標將跳到錯誤的程式位置上以方便檢視。至於建立(Build)功能，除了重複編譯的功能外，還會將編譯過的程式碼做一

些處理，再透過 USB 線傳到 BASIC Commander 硬體上。

除錯
如果編譯過程有錯誤訊息顯示在輸出視窗時，就必須進行除錯的動作。您可以參考輸出視窗上提示的錯誤訊息，並利用終端視窗的功能逐一找出錯誤的原因，進而將錯誤排除。

選單及命令

選單的功能與一般視窗式的軟體類似，分述如下：

檔案選單
包括開啟舊檔、關閉檔案、儲存檔案、列印等等常用的功能，還有最近開啟過的檔案列表，方便快速的選擇常用的檔案。

編輯選單
除了一般視窗軟體用來編輯文字的剪下、複製、貼上、尋找、取代、重複和復原等功能外，還額外加入了以下幾項功能：

註解選取
可同時選擇一行或多行的程式，在每行程式的開頭加上單引號 ’ ，被加上單引號 ’ 的程式將被視為說明文字，系統編譯時不會將它們翻成程式碼。

註解解除
可同時選擇一行或多行的程式，將每行程式開頭的單引號 ’ 移除。

跳至行號

可直接指定一個行號，讓游標直接跳到指定的位置。整個程式的行號是由系統自動產生的，使用者可選擇檢視選單中的行號選項或編輯工具列中的 Icon 圖示，決定是否讓行號顯示在每行程式的前面。

檢視選單

選擇 innoBASIC Workshop 螢幕上哪些控制項要隱藏或顯示。例如之前提過的行號選項，可決定是否要在每行程式的前面顯示行號。

建立選單

在這裡可以將程式進行編譯並且從電腦下載到 BASIC Commander。

編譯

在初步編寫程式及除錯的過程中並不需要常常做下載的動作，這時可利用建立選單下的編譯功能，只做〝編譯〞的動作。〝編譯〞這個動作是將程式中的 innoBASIC 命令翻譯成機器碼，如果有像是拼字或語法的錯誤，就會顯示在輸出視窗中。在這錯誤訊息上點擊二次，游標將跳至程式中發生錯誤的地方。如過您只是想檢查程式是否有錯誤，利用這個功能可以省下下載的時間。

建立

建立這個功能結合了〝編譯〞和〝下載〞的功能。系統除了編譯程式之外，會緊接著做下載的動作。

工具選單

列印字型設定

選擇列印文字字體的大小及型態。

字型設定

乃針對程式編輯視窗(Program Editing Window)中程式的文字大小及型態作設定。

偏好設定

這裡有一些可根據個人喜好來設定 innoBASIC Workshop 的選項。編輯器之邊際選擇可選擇程式行之前是否要留邊際空間，顯示行號之設定可選擇每行程式之前是否要顯示行號，單行背光則選擇游標行是否要變顏色，也可選擇是否將搜尋結果顯示於螢幕中央。字型偏好可以選擇文字字型。顏色選項顧名思義就是選擇 Workshop 視窗上的一些相關的顏色，範例中可預覽你所選擇的顏色。終端視窗選項可以改變最大顯示列數以及每列最大顯示字數。原始設定選項可讓你回到系統安裝時的初始設定。

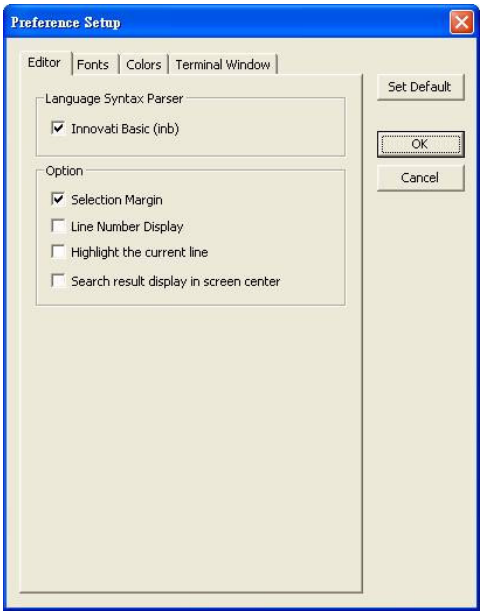


圖 3-4　偏好設定

視窗選單

這裡提供了許多方便的功能：可以利用重疊(Cascade)功能來重疊顯示多個檔案，或利用並列(Tile)功能來並列多個檔案以方便同時參考一個以上的檔案；可以利用新視窗功能再開一個指定的檔案，或利用切割(Split)功能將檔案作畫面切割以方便對同一個檔案的內容做比較參考；也可以利用標籤 MDI (Tabbed MDI)功能在檔案上方加上分頁標籤以節省視窗空間。

求助選單

使用 InnoBASIC Workshop 時如果對一些功能或選項有疑問，可透過求助主題來尋求幫助。另外，可由關於功能來查看 InnoBASIC Workshop 的版本。

4 硬體說明

簡介

本系統硬體中以 BASIC Commander 最為重要，它就是一般所謂的單板電腦(Single Board Computer)，簡稱 SBC，它是整個系統的核心，也是你的專案的大腦。另一個重要的硬體是教育板(Education Board)，它可以讓 BASIC Commander 插在上面工作，也可以在上面插上一些電子元件配合你的程式來完成你的專案。另外還有一些周邊硬體模組，這些模組各有各的功能，但都可以和 BASIC Commander 結合成所需的專案。

BASIC Commander 單板電腦

BASIC Commander 是一個完整的單板電腦(Single Board Computer，簡稱 SBC)，它是控制整個專案的核心。它上面的主要元件包括有一顆工業等級微控制器、一些時脈電路、USB 介面等等，它的外觀如附圖所示。

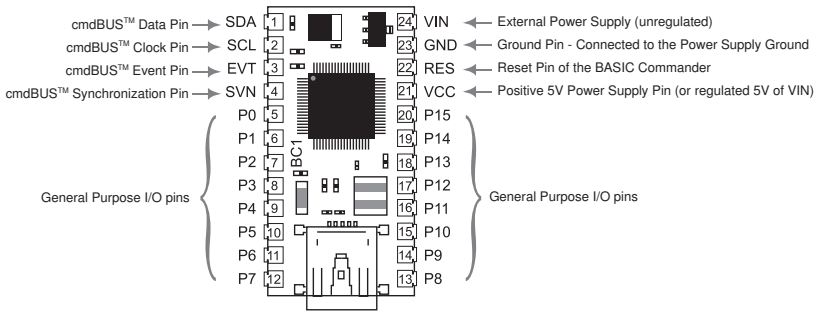


圖 4-1a　24 引腳 BASIC Commander

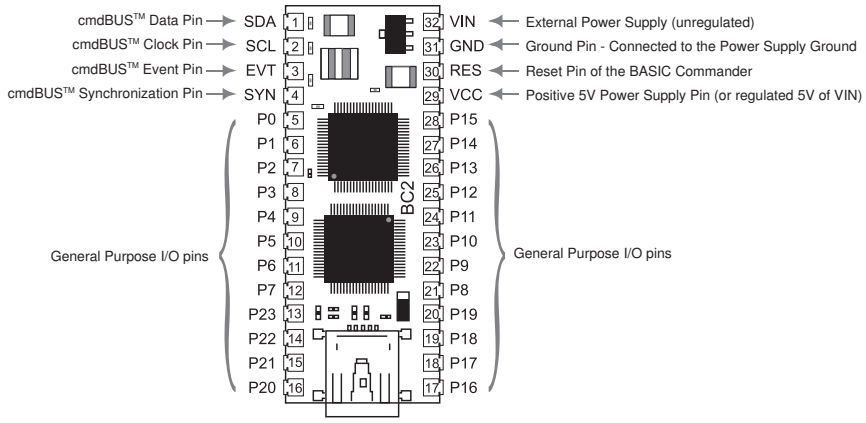


圖 4-1b　32引腳BASIC Commander

當 BASIC Commander 以 USB 線接上電腦後，BASIC Commander 將可直接使用電腦的電源。

BASIC Commander 引腳說明

BASIC Commander 有 24-pin 及 32-pin 二種模組形式。接腳與接腳之間

的寬度為標準的 0.1 英吋，可相容應用於標準的 PCB 插槽和麵包板。每支接腳的功能如附表。

接腳名稱	接腳功能
SDA	cmdBUS(TM) 資料訊號
SCL	cmdBUS(TM) 時脈訊號
EVT	cmdBUS(TM) 事件訊號
SYN	cmdBUS(TM) 同步訊號
P7~P20	一般 I/O 訊號
P16~P19	一般 I/O 訊號
P8~P15	一般 I/O 訊號
VCC	正 5V 電源（或從 VIN 穩壓過的 5V 電源）
RES	BASIC Commander 重置
GND	接地（與電源地線連接）
VIN	外部電源（未穩壓）

教育板

大部分的專案都需要外加一些零件。教育板上有一塊小麵包板可供使用者在上面插上所需的零件，例如：開關、發光二極體、電阻、電容等等；教育板上也提供 BASIC Commander I/O 腳的插槽，方便直接使用 BASIC Commander 內建的 I/O 功能。

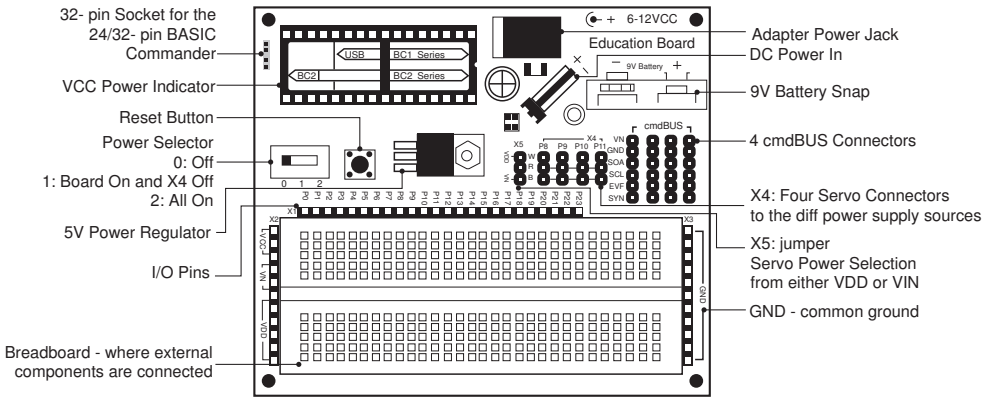


圖 4-2　教育板（Education Board）

教育板上有一個 32-pin 的腳座，不論是 24-pin 或 32-pin 的 BASIC Commander 均可直接插在上面使用，特別注意插上 BASIC Commander 時須依照板子上的方向提示圖案小心插上，避免因方向錯誤造成 BASIC Commander 損壞。

請務必小心注意插上 BASIC Commander 時的方向，方向錯誤將導致 BASIC Commander 無法挽回的損壞。建議在關閉電源的情況之下插上 BASIC Commander，開啟電源前再次確認方向是否正確。

電源供給

教育板的電力來源有二種選擇：第一，來自 USB 埠者稱為 VCC，其插孔位於麵包板的旁邊，當 USB 線接上 BASIC Commander 時， BASIC Commander 腳座旁邊的 LED 將被點亮。但因 USB 埠最多只能提供 500mA 的電力，因此，對於較大功率的應用就必須使用外加電源。

第二，就是外加電源 VIN，使用者可選擇直接以 6-12V DC 電源插上板子上的電源插銷，或者插上一個 6-12V DC 電源供應器，或者接上一個

9V 的鈕扣電池。教育板會將 VIN 穩壓成 5 伏特的 VDD 使用，其可提供的最大電流為 1 安培。當使用 VIN 電源時，教育板中央位置的 LED 將被點亮。

在標記為 X4 的位置有四組伺服馬達的接頭，標記分別為 P8、P9、P10、P11，可讓使用者接上四個伺服馬達。伺服馬達的電源可在 X5 的位置利用跳接插梢來選擇 VDD 或 VIN。使用伺服馬達時為了節省 9V 電池的功率消耗，可將滑動開關的位置從 2 切到 1，這樣就可關掉對伺服馬達的電源供應。

教育板上的滑動開關是用來切換電源用的，切到 0 時，VDD 關掉;切到 1 時 VDD 打開，但 X4 上的 VIN 則關掉;切到 2 時，VDD 及 X4 上的 VIN 則均打開。只要 VDD 被打開，VDD 的指示燈就會被點亮。請參考下表：

開關位置	VDD	VIN or VDD (X4 接頭)	VIN (母插座)	VCC (USB)
0	關	關	關	關
1	開	關	開	開
2	開	開	開	開

RESET 按鍵用來重置 BASIC Commander，程式重新開始執行。

教育板上有四組 cmdBUS 的接頭，可分別接上四條 6 線的 cmdBUS 排線，用來連接利基科技的周邊模組。下表是教育板上各接頭及開關的簡單說明：

名　　稱	功　　能
32-pi 腳座	24-/32-pin BASIC Commander 腳座
電源插頭	內正外負 6~12 V 直流電源插頭

電源插硝	6~12 V 直流電源輸入插硝
電池扣	9V 電池扣
滑動開關	電源開/關及伺服馬達電源開/關
RESET 按鍵	重置按鍵
X4	4 組伺服馬達接頭
X5	伺服馬達電源選擇插硝
cmdBUS	4 組周邊模組接頭

教育板上共有 2 個發光二極體（ LED ）。一個在 BASIC Commander 腳座旁邊，點亮時表示 USB 接頭已經連上 USB 埠。另一個在教育板的中央附近，點亮時表示已經接上外部電源。

周邊模組

利基科技提供了一系列的周邊模組，使用者不必再自己用各種零件去組裝模組，省去了自己花在電子工程上的時間。下圖是一個鍵盤模組的例子

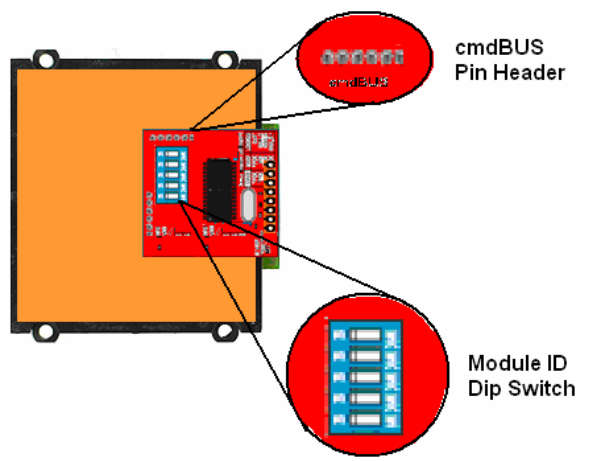


圖 4-3　鍵盤周邊模組

使用周邊模組首先須先設定模組電路板上的指撥開關，作為這模組的識別碼(Module ID)。識別碼的範圍可從 0 到 31，請注意同一個 BASIC Commander 控制的 cmdBUS 不能有二個模組設定相同的識別碼。每一個周邊模組接上一條 6 線的 cmdBUS 排線，再接上教育板的模組接頭。

圖 4-4　6線式 cmdBUS™ 排線

VIN 是未穩壓的外部電源，電壓範圍為 6~12 伏特直流電，這個電源也被穩壓成 5 伏特後供給周邊模組使用。特別注意插上 cmdBUS 時的方向，若插錯方向將導致元件損壞。當程式中宣告模組的識別碼與硬體指撥開關的設定碼相同時，這模組就可開始使用。當然，每個模組都有它專屬的命令和功能，請參閱各模組的使用手冊。

靜電預防措施

為了方便運輸及儲存，BASIC Commander 已經使用防靜電的包裝。但若以手拿 BASIC Commander 時，請小心提防靜電破壞其中的 IC，這些靜電可能因為乾燥的空氣或某些特殊材料的環境造成。

InnoBASIC 程式語言 5

簡介

innoBASIC 是專為利基科技的 BASIC Commander 系統開發的高階程式語言，它雖然是一套簡單易學的程式語言，但它強大的功能也適合有經驗的使用者使用。以下的例子將引導您開始進入 innoBASIC 的有趣世界。

```
Sub Main()
Debug “Hello World!”
End Sub
```

敘述

程式中的敘述(Statements)讓編譯器將命令編譯成可執行的對應程式碼。敘述可以包含常數、變數、運算子以及函數，用來定義常數、宣告變數、執行算術及邏輯運算、以及執行程式控制轉換及宣告副程式。

通常一個敘述使用一行，但如果有許多比較短的敘述要表示時，為了方便閱讀，可以將它們組成一行，但敘述之間要用冒號“:”分開。如果

比較長的敘述會超過一行時，可用一個底線“_”加上一個空格來做連接。以下是一個參考的例子。

```
Sub Main()
Debug _
“Hello World!”
End Sub
```

請注意註釋及字串值不能做行的連接。

註釋

以撇號開頭的文字用來代表註釋(Comments)，例如：

```
Dim a As Short          'Here is the Comment!
```

除非是字串值，否則所有撇號右邊的文字都被視為註釋，不會被編譯器所編譯。請注意註釋不能做行的連接。

識別字

識別字(Identifiers)是一個名字，必須以英文字開頭，後面跟著英文字母、數字、或底線，最大長度為 31 個，沒有大小寫的區分。請注意關鍵字不能當識別字使用。

關鍵字

關鍵字(Keywords)是一些有特殊意義的保留字，不可以被拿來當識別字使用。請參閱附錄 B，該處列出所有 innoBASIC 語言的關鍵字。

標籤

標籤(Labels)必須出現在行的最前面，而且必須是合法的識別字，後面以冒號結束，它用來標示一段可供 GOTO 敘述跳躍的程式開頭。

常數、變數及資料型態

常數是一個固定值，它不會因為程式的執行而改變，宣告時須以關鍵字 CONST 宣告，其型式如下：

CONST Constname = expression

以下是宣告常數的例子：

```
Const DaysofMay as Byte = 31
Const JAN as String*7 = “January”
```

常數值需符合所給的型態。常數隱含有可被整個程式擷取的意義，所以

它必須以全域宣告。

相對於常數而言，變數所代表的值則可能在程式執行中被改變。所有的變數均須以關鍵字 DIM 來宣告，例如：

DIM Variablename As Type [= value]

任何變數在被程式使用前，系統必須先被告知它的存在，以及它的型態、大小等。變數須先設定初始值，系統預設的初始值是 0 或空的字串。慣例上，變數的原始資料型態為 Boolean、Byte、Short、Word、Integer、DWord、Long 及 Float 等，且均會實際佔用記憶體的位置。 innoBASIC 另外一種特殊的變數型態是使用在 EEPROM 記憶體的持久性變數。如果 DIM 被宣告在程序裡，那麼這變數就屬於區域性變數，只能給所在位置的程序使用;如果是宣告在程序外面，那麼就屬於全域變數，就可以全部的程序使用。請注意固定型態的變數須在程式一開始的地方宣告，而且必須宣告在所有的副程式外面。

另外需注意如果變數宣告成全域變數,那麼它的初始值不可以在宣告時設定，而必須是在程式中設定。但如果是區域變數則不在此限。

變數型態	大小	內容
BOOLEAN	1 Byte 或 1 Bit	通常作為旗標狀態或布林結果的變數。當宣告為區域變數時，使用 1 Byte；當宣告為全域變數時，使用 1 Bit。
BTYE	1 Byte	無符號變數，值域為 0~255。
SHORT	1 Byte	有符號變數，值域為 -128~+127
WORD	2 Bytes	無符號變數，值域為 0~65535
INTEGER	2 Bytes	有符號變數，值域為 -32768~+32767
DWORD	4 Bytes	無符號變數，值域為 0~4294967295
LONG	4 Bytes	有符號變數，值域為 -2147483648~2147483647
FLOAT	4 Bytes	浮點數變數，值域為

		-3.4E+38~+3.4E+38。
STRING	N Bytes	字串變數，將 ASCII 字元用雙引號””括起來表示，字元長度由使用者指定，但必須限制在變數記憶體大小的範圍內。如果字串裡有雙引號字元，則該字元要寫二次雙引號。
PERSISTENTBYTE PERSISTENTSHORT	1 Byte	儲存在非揮發性記憶體 EEPROM 裡的變數。由於操作的限制，二種資料型態實際上是相同的。
PERSISTENTWORD PERSISTENTINTEGER	2 Bytes	儲存在非揮發性記憶體 EEPROM 裡的變數。由於操作的限制，二種資料型態實際上是相同的。
PERSISTENTDWORD PERSISTENTLONG	4 Bytes	儲存在非揮發性記憶體 EEPROM 裡的變數。由於操作的限制，二種資料型態實際上是相同的。
PERSISTENTFLOAT	4 Bytes	儲存在非揮發性記憶體 EEPROM 裡的浮點數變數，值域為 -1.7E+38~+1.7E+38。

型態轉換

資料型態的轉變是指轉換變數儲存的資料，而不是變數本身。轉變型態通則是從較窄的運算域轉到較寬的運算域，例如從 Short 轉成 Integer 或 Long 型態。如果反過來操作，那麼多出來的高位元資料將會遺失，須小心使用。

文字值

用來表示變數數值的文字稱為文字值(Literal)。包含布林值(Boolean)、整數值(Integral number)、浮點數(Floating point)、字串(String)及字元(Character)。

布林值 (Boolean Literals)

布林型態的文字值為 True 及 False，分別對應到「真」和「偽」，其數值為 1 和 0。

整數值 (Integral Literals)

整數文字值可以為十進位、十六進位、八進位或二進位。十進位文字值是一串不需要字首的十進位數字(0-9);十六進位文字值是一串以&H 為字首的十六進位數字(0-9， A-F);八進位文字值是一串以&O 為字首的八進位數字(0-7) ;二進位文字值是一串以&B 為字首的二進位數字(0 or 1)。十進位文字值直接表現出整數文字值的十進位值，但八進位和十六進位文字值卻表現出整數文字值的二進位值。

浮點數值 (Floating-Point Literals)

浮點數值包含一個整數、一個可有可無的十進位點(ASCII 碼的句點)及小數、以及一個可有可無的以 10 為底的指數。小數點前後的位數總數限制為 5 位。

字串值 (String Literals)

字串值是將 ASCII 字元用雙引號” ”括起來表示，如果字串裡有雙引號字元，則該字元要寫二次雙引號。考慮到字元長度必須限制在變數記憶體大小的範圍內，所以字元長度必須在字串變數宣告時設定。

字元值 (Character Literals)

字元值沒有特殊的資料格式，它由一個單一的 ASCII 字元碼表示。字元值和單一字元的字串值的分別，是在字元值的字尾加一個字母 c。例如：

```
myChar = "H"c       ' stands for a single character H
myString = "H"      ' stands for a string with one character H
```

陣列 (Arrays)

陣列裡含有一些元素，其型態是除了布林、字串、或其他陣列的變數型態。擷取這些元素都必須以索引的方式。如果一個陣列有二組索引，則稱為二維陣列，以上則以此類推。陣列宣告時可以在名稱後面順便宣告陣列的大小。陣列的索引是從 0 開始計算。例如：

```
Dim myarray(5) As Short = {1,2,3,4,5,6}
```

以下例子為利用陣列寫出九九乘法表。

```
Sub Main()

    Dim m(8,8) As BYTE
    Dim I As BYTE
    Dim J As BYTE

    For I=0 to 8
    For J=0 to 8
        m(I,J)=(I+1)*(J+1)
        Debug m(I,J)," "
    Next J
        Debug CR
    Next I

End Sub
```

終端視窗 (Terminal Window) 將會顯示出一個九九乘法表。請注意一個陣列最大限制為 100 個元素。

運算子

運算子(Operator)分為下列二類：單運算子運算子 (Unary operators) 是以一個前置的符號操作一個運算子，例如正、負符號。雙運算子運算子 (Binary operators)則是以一個置中符號操作二個運算子，例如加、減號。當運算式中含有多個運算子時，運算子的優先序如下表，當然，你可以使用括弧來改變優先順序。

運算子優先順序由高而低排列如下：

類別	運算子
單運算子的正和負號	+, -
乘和除	*, /, Mod
加和減	+, -
移位	<<, >>
關係	=, <>, <, >, <=, >=
位元運算 AND、OR、XOR 及補數	AND、OR、XOR 及 ~
邏輯 NOT、AND 及 OR	NOT、AND 及 OR

以下介紹各種不同的運算子。

算數運算子

共有七種算數運算子，
+ 加
- 減
* 乘
/ 除（浮點數）
\ 除（整數）

MOD 餘數（整數除法餘數）
<< 左移（等同二進位乘法）
>> 右移（等同二進位除法）

關係運算子

關係運算子比較二個數值並回傳一個「真」(1) 或「偽」(0) 的結果。

> 大於

>= 大於等於

< 小於

<= 小於等於

= = 等於

!= 不等於

位元運算子

共有四個位元運算子， AND、OR、XOR 及 ~ (1的補數)。

邏輯運算子

邏輯運算子支援AND、OR和 NOT等邏輯運算，用以產生「真」或「偽」。

&& 為 AND 邏輯

|| 為 OR 邏輯

! 為 NOT 邏輯

指定運算子

共有六種指定運算子(Assignment Operators)。最簡單的運用是將等號右邊的變數值取代等號左邊的變數值。其他五種是混合的指定運算子。以A += B為例，將等同於寫成A = A + B。

=

+ =

- =

* =

/ =

\ =

程式控制流程

程式敘述以其在程式中的順序依序被執行，但可使用條件敘述、非條件敘述等方式改變其執行順序。條件敘述有四種：If...Then...Else 敘述、Select...Case 敘述、 Do…Loop and For…Next 敘述。非條件敘述則是指 Goto 及呼叫敘述。

IF...THEN...ELSE 敘述

如果 If 敘述為「真」，則 If 敘述之下的程式段將被執行;如果 If 敘述為「偽」，則 Else 敘述之下的程式段將被執行。另外可選擇使用 Elesif 敘述，只要 Elesif 為「真」，則 Elesif 敘述之下的程式段將被執行。任何被選擇執行的程式段執行完畢後，程式就結束 If...Then...Else 敘述的執行。

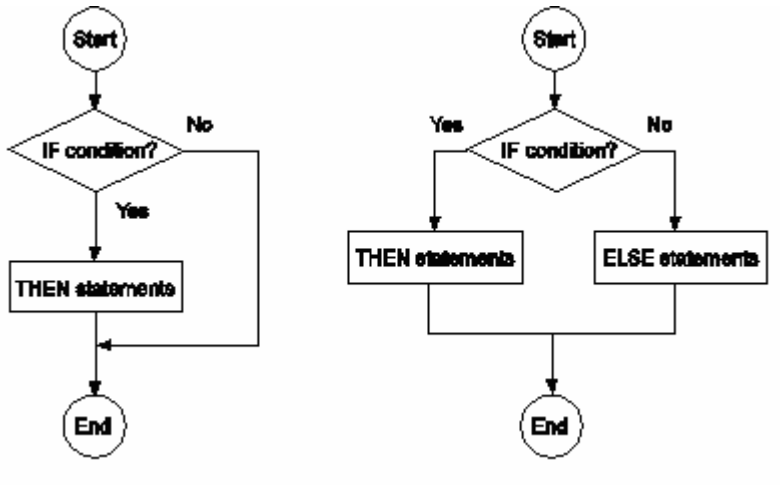


圖 5-1 If…Then…Else 敘述

如果 If 及 Else 的敘述都是單一敘述，則可合併成一行敘述，例如：

```
Sub Main()

Dim a,b,Max As Integer

Debugin "Enter the First Number:", a
Debugin "Enter the Second Number:", b

If a < b Then
    Max = b
Else
    Max = a
End If

Debug "Max is:", Max

End Sub
```

SELECT...CASE 敘述

SELECT…CASE 可說是 IF…THEN…ELSE 的進階架構，它可以在許多可能中擇一執行。執行此敘述時，Select 的值將與 Case 的值逐一比對，當某一個 Case 的值符合時，該 Case 之下的程式段將被執行。如果沒有任何一個 Case 的值符合，則 Case Else 之下的程式段將被執行。任何被選擇執行的程式段執行完畢後，程式就結束 SELECT…CASE 敘述的執行。

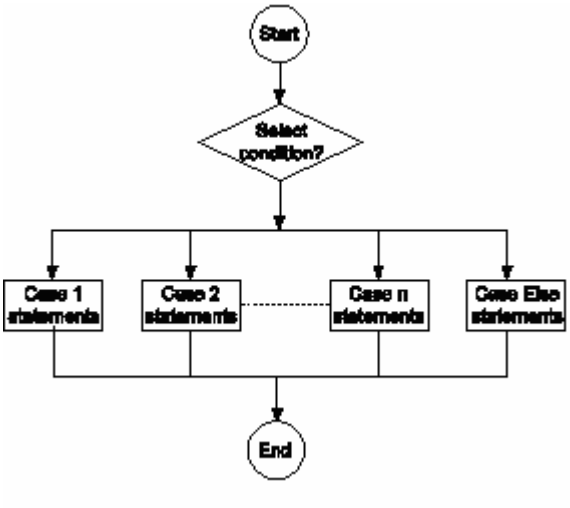


圖 5-2　Select…Case 敘述

以下為 SELECT…CASE 敘述的例子：

```
Sub Main()
    Dim x As Byte
    Do
    Debugin "Enter a 1 to 7 to find the nth day of a week.", x, CR

    Select Case x
     Case 1
          Debug "It's Sunday.", CR
     Case 2
          Debug "It's Monday.", CR
     Case 3
          Debug "It's Tuesday.", CR
     Case 4
```

```
          Debug "It's Wednesday.", CR
     Case 5
          Debug "It's Thursday.", CR
     Case 6
          Debug "It's Friday.", CR
     Case 7
          Debug "It's Saturday.", CR
     Case Else
          Debug "Wrong Number!", CR
     End Select
     Loop
End Sub
```

DO...LOOP 敘述

DO…LOOP 敘述可重複執行某一段程式段。

Do [{While | Until} condition]
[statements]
[Exit Do]
[statements]
Loop [{While | Until} condition]

基本的 DO … LOOP 命令會讓其中的程式段永遠不斷地執行。你可以在迴圈的開始或結束的地方加上 WHILE 敘述(但不能二處都加),如此當布林條件為「真」時,其下的程式段將被執行。如果 WHILE 被放在迴圈結束的地方,則程式段將至少被執行一次,然後才做布林條件的判斷。

UNTIL 的用法類似 WHILE 的用法，只是 UNTIL 是當布林條件為「真」時結束而不是繼續執行程式段。EXIT DO 命令可放在程式段之中，用來立即離開 LOOP 迴圈。

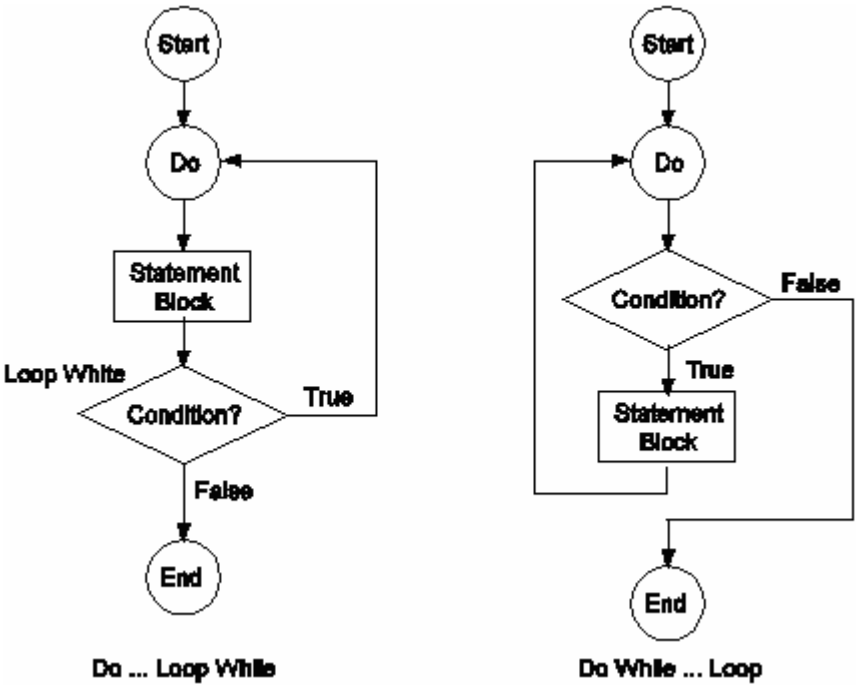


圖 5-3a　Do…Loop 敘述

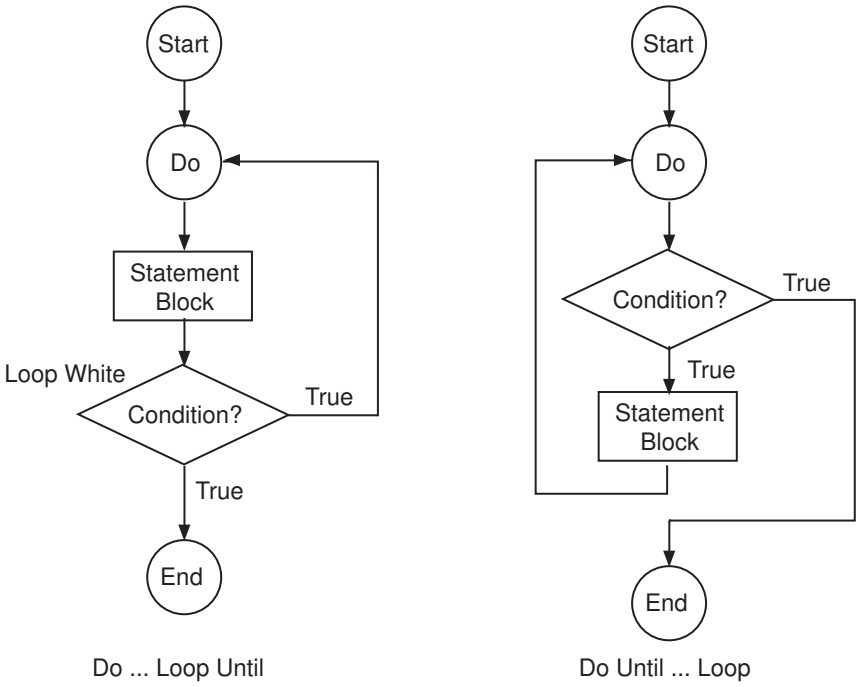


圖 5-3b　Do…Loop 敘述

以下為 DO … LOOP 敘述的例子：

```
Sub Main()
    Dim x As Short
    x = 1
    Do While x<5
        Debug "*"
```

```
        x+=1                'display 4 asterisks
    Loop
    Debug CR

    x = 1
    Do
        Debug "*"
        x+=1                'display 4 asterisks
    Loop While x<5

    Debug CR

    x = 5
    Do Until x=0
        Debug "*"
        x-=1                'display 5 asterisks
    Loop

    Debug CR

    x = 5
    Do
        Debug "*"
        x-=1                'display 5 asterisks
    Loop Until x=0
End Sub
```

FOR...NEXT 敘述

如果一個程式要做一些已知次數的循環，則可使用 For...Next 敘述，你可以利用一個迴圈變數的增加或減少來做循環控制。 For...Next 敘述需有一個迴圈變數、一個起始值、 一個結束值和一個間距值。程式在 FOR 之後將以起始值代入迴圈變數開始執行，以後每執行一次程式段，變數值就以 STEP 後面的間距值增加或減少，直到變數值等於結束值才離開程式迴圈去執行 NEXT 之後的敘述。如果是要控制變數增加的

迴圈，則 STEP 值必須是正數，而且結束值不可以小於起始值;相反的，如果是要控制變數減少的迴圈，則 STEP 值必須是負數，而且結束值不可以大於起始值，如果計數方向錯誤，則程式將不會執行。 以下為 For…Loop 的流程圖。

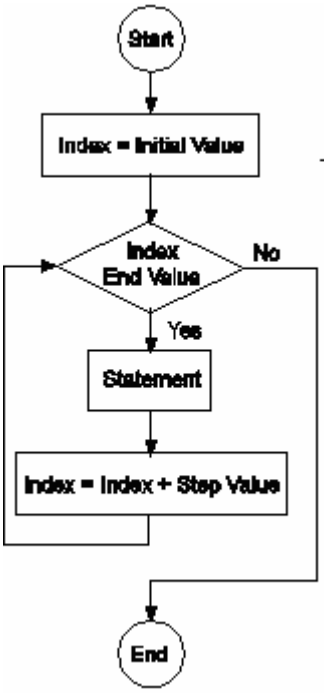


圖 5-4 For…Next 敘述

請注意不同型態變數的值域範圍。例如 SHORT 變數的值域為-128 到+127， BYTE 變數的值域為 0 到+255。控制迴圈的變數須以 As 標出，且須為原始的數字型態(Byte，Short，Word，Integer，DWord，Long 等)。例如;

```
Sub Main()
    Dim Index As Byte
    Dim Sum As Byte

    Sum=0
    For Index = 1 To 10 Step 1
    Sum+=Index
    Next
    Debug "1+2+...+10=",sum
End Sub
```

For...Next 迴圈控制變數不可以做為其內部嵌入的 For...Next 迴圈控制變數。Next 敘述永遠與其內圍的上尉配對的 For 敘述配對。除非為了方便程式閱讀，通常位於 Next 之後的迴圈控制變數名稱可以省略。但是如果標示出 Next 之後的迴圈控制變數名稱，但卻沒有正確配對，則會於編譯時產生錯誤訊息。你可以使用關鍵字 Exit 跳出 For...Next 迴圈，但不可以從迴圈外直接跳入迴圈。

GOTO 敘述

GOTO 命令將強迫程式跳到指定的程式段，而該程式段必須先給一個標籤名字。

呼叫敘述

呼叫敘述是由關鍵字 CALL 來執行 SUB 趁序或 FUNCTION 函數。程式跳到 SUB 或 FUNCTION 執行後再回到呼叫敘述的下一個敘述繼續執行。EVENT 的功能類似 SUB 或 FUNCTION 程式的呼叫，只是它是在某一事件發生時

自動執行的。細節請參考以下敘述。

SUB 與 FUNCTION

一個程式至少須有一個以”main”為命名的 Sub 程序。通常以“Sub Main()“敘述來表示一個程式的開始，一直到 Main()的最後結束。除了 Main()程序之外，你可以加上其它的 Sub 程序，函數(Function)等，讓程式更具結構化，更有效率。

SUB 和 FUNCTION 副程式都可以有引數(Arguments)，但只有 FUNCTION 副程式會有回傳值。通常，一段重複使用的程式會使用 SUB 副程式來減少程式空間及增加可讀性，FUNCTION 副程式則用來執行一段運算，並將運算結果的值回傳主程式。

Sub 程序

Sub 程序不回傳值。每次 SUB 被呼叫後即開始執行，直到 End Sub 出現為止。Sub Main()本身即是一個 Sub 敘述，也是一個程式的開始點。

```
Sub Main()
    Display()
End Sub

Sub Display()
    Debug "Sub Procedure Display() has executed."
End Sub
```

Functions 函數

FUNCTION 會有回傳值，通常用來求值、計算或轉換資料。宣告 FUNCTION 的方式與 Sub 類似。宣告時須有關鍵字”Function”。以下為如何使用 FUNCTION 的例子：

```
Function Max(I As Integer, J As Integer) As Integer
    If I>J Then Return I Else Return J
End Function

Sub Main()
    Dim X, Y, Z As Integer
    Do
    Debugin "Enter the First Number:", X
    Debug X, CR
    Debugin "Enter the Second Number:", Y
    Debug Y, CR
    Z = Max(X,Y)
    Debug "The Maximum value is ",Z , CR
    Loop
End Sub
```

參數

參數是代入 SUB 或 FUNCTION 程序的引數。在宣告 SUB 或 FUNCTION 程序時，參數被置於程序名字後面的括弧中。注意參數必須指定型態，且字串及陣列不可以為參數型態。

有二種參數傳遞方式，包括傳址呼叫 ByRef 和傳值呼叫 ByVal。如果沒有明確表示，系統預設為 ByVal。宣告傳址呼叫時須在參數前加上 ByRef。傳址呼叫的參數並不會佔用新的記憶體位置，而是延用先前呼叫它的引數的變數位置，改變參數內容將會同時改變引數的變數內容。以下的例子為傳址呼叫的應用：

```
Sub Swap(ByRef a As Integer, ByRef b As Integer)
    Dim t As Integer = a
    a = b
    b = t
End Sub

Sub Main()
    Dim x As Integer = 1
    Dim y As Integer = 2
    Debug "before: x = ", x, ", y = ", y, CR
    Swap(x, y)
    Debug "after: x = ", x, ", y = ", y, CR
End Sub
```

程式的輸出為：

之前　：x = 1, y = 2

之後　：x = 2, y = 1

周邊模組相關程式功能

周邊模組是本系統一個非常特殊的功能，使用上針對其軟體有以下幾項必須說明：

周邊模組的宣告

每一種周邊模組都有其產品名稱，且這些名稱都已由利基科技統一命名。例如，2x16LCD模組，其產品名稱為LCD2X16A。 使用者可宣告一個

自己命名的模組名稱，其為LCD2X16A的形式，後面跟著一個”位址識別碼”（ID Address），識別碼的範圍可以從0到31。請注意這個敘述必須以全域（global declaration）的方式宣告，即寫在SUB、FUNCTION 或EVENT的外面。

周邊模組命令的執行

由於不同的模組有不同的特殊功能命令，使用時請各別參考其相關文件。操作命令時要在模組名稱和命令之間加上一點 ”.” ，如下例：

```
Peripheral myLCD As LCD2X16A @ 0
Sub Main()
    myLCD.Display("Hi there!")
End Sub
```

事件（EVENT）程序

事件程序使用於有不可預期的事件發生，或者用來代替比較沒有效率的輪詢(Polling)。使用 EVENT 功能可讓使用者不必時常去輪詢周邊模組的狀況。操作時先寫好 EVENT 的程序，然後再開啟 EVENT 即可。EVENT 程序是一段由 EVENT 和 END EVENT 敘述包起來的程式。你的主程式可以去處理一些比較重要的事情，當有事件發生時，程式會自動跳到該事件的程序去執行。

周邊模組的事件（EVENT）宣告

一個 EVENT 宣告須包含一個模組名稱和一個有效的模組事件名稱，這二部分中間由一個點作結合。第一個部份的模組名稱是用以分辨一個以上的週邊模組;第二部份的 EVENT 名稱由個別模組提供，使用者必須先參考其使用文件。請注意 EVENT 程序在程式中有最高優先順序，當 EVENT 程序被執行時，其它的工作均暫停，直到離開 EVENT 為止。因此一般建

議不要在 EVENT 程序中停留過久。另外，EVENT 程序沒有參數傳遞或回傳值。

```
Event MyKeypad.Keypressed()
    Dim KeyID as Byte
    MyKeypad.GetKey(KeyID)                    ' To get the Key ID
    Debug "Key ", KeyID, "is Pressed!", CR    ' To display
End Event
```

使用周邊模組範例

以下這個例子使用一個 2 行 16 字的液晶顯示模組及一個 16 鍵的按鍵模組，組合成一個簡單的模組，範例中僅使用很少的命令即可完成將案件的數字顯示在液晶螢幕上。

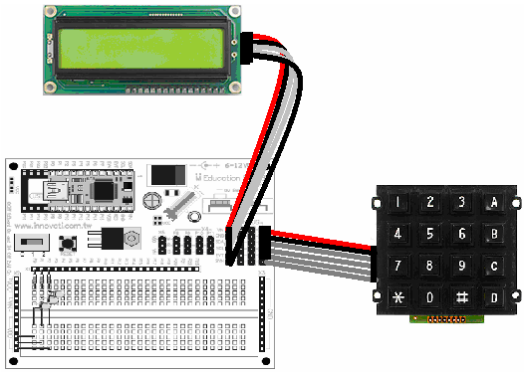


圖 5-5　周邊模組使用範例

```
Peripheral myLCD As LCD2X16A @ 0
```

```
Peripheral myKeypad As KeypadA @ 1

Sub Main()
    Dim KeyVaue As Byte
    Do
    myKeypad.getkey(KeyValue)
    myLCD.Display(KeyValue)
    Loop
End Sub
```

當然您可以依照自己的創意及想像力，加上任何模組及零件，完成另一個不同功能的應用！

命令集 6

簡介

本章列出 innoBASIC 程式語言的命令規格，提供給使用者參考使用，各命令乃按照字母順序逐一介紹。

命令形式

基本上，本系統的命令可分成二類，第一類為一般傳統的 BASIC 命令，另一類則為本系統特有的系統命令。在傳統 BASIC 命令的部份，將硬體的週邊模組與傳統 BASIC 命令做了結合，提供了獨具特色的命令。在系統命令的部份，也以獨具特色的命令而與傳統 BASIC 命令有所不同。

文件慣例

本章針對命令的敘述中，命令必要的文字以大寫粗體字母及斜粗體形式標示。大寫粗體的文字是必須依照原字寫出，但斜粗體的部份則是要以使用者的值來取代。非必要的文字則以大括號{}括起來，這些內容可依需要決定加入或省略，使用時大括號{}不必標出來。但中括號[ ]及小括號( )在使用時則必須標出來。另外，符號 “|” 表示其二邊的元素為互斥元素，

即二者之中只有一個成立。 當然，編寫程式時並不需要將文字寫成大寫字母或粗體或斜體等，那只是本使用手冊方便說明的表示方式而已。

分類

InnoBASIC 語言的命令可以分類成三種。第一種為基本命令，它們是構成程式的結構，包括宣告、流程控制、決定判斷等。第二種為輸出入命令，利用 BASIC Commander 硬體的輸出入口所設計的各種函數命令，包括基本的 I/O 操作、計數器、脈衝偵測、通訊及其它的特色命令。第三種為數學算術及運算命令。下表即為這三種分類的命令總表。

基本命令	
CALL	GOTO
DIM	IF …THEN … ELSE
DO… LOOP	PERIPHERAL
ENUM … END ENUM	RETURN
EVENT … END EVENT	SELECT… CASE
FOR… NEXT	SUB … END SUB
FUNCTION…END FUNCTION	

輸出入命令	
BUTTON	OUTPUT

CHECKMODULE	PAUSE
COUNT	PULSEIN
DEBUG	PULESOUT
DEBUGIN	PWM
GETDIRPORT	RANDOM
HIGH	RCTIME
IN	READPORT
INPUT	RESETMODULE
LCDCMD	REVERSE
LCDIN	SETDIRPORT
LCDOUT	TOGGLE
LOW	WRITEPORT

數學及轉換命令	
ABS	FLOOR
ACOS	INTEGER2FLOAT
ASIN	LCASE
ATAN	LEFT
ATAN2	LEN
BYTE2FLOAT	LOG
CEIL	LOG10
COS	LONG2FLOAT

DWORD2FLOAT	MID
EXP	RIGHT
EXP10	SGN
FLOAT2BYTE	SHORT2FLOAT
FLOAT2DWORD	SIN
FLOAT2INTEGER	SQRT
FLOAT2LONG	STRING2FLOAT
FLOAT2REALSTRING	STRREVERSE
FLOAT2SHORT	UCASE
FLOAT2STRING	WORD2FLOAT
FLOAT2WORD	

命令總覽

以下按照字母順序逐一解釋每一條命令。這裡是所有命令說明與範例參考的主要區域。

ABS

語法

Result = ABS(*Argument*)

操作

回傳浮點數的絕對值。

- ***Argument*** - ABS 函數的浮點運算元。
- ***Result*** - 接受 ABS 函數回傳值的浮點變數。

說明

ABS 命令回傳浮點數的絕對值。ABS 的結果非負值。

範例

```
Sub Main()
    Dim Result As Float

    Result = ABS(2.0)
    Debug "ABS(2.0)=", Result, CR
    Result = ABS(-2.0)
    Debug "ABS(-2.0)=", Result, CR
End Sub
```

ACOS

語法

Result = ACOS(*Argument*)

操作

回傳浮點數的反餘弦值。

- ***Argument*** - ACOS 函數的浮點運算元。值域介於-1 與 1 之間。
- ***Result*** - 接受 ACOS 函數回傳值的浮點變數。回傳值值域介於 π 與 0 弧度之間。

說明

ACOS 命令回傳浮點數運算元的反餘弦值(arccosine)。運算元的值域介於-1 與 1 之間。回傳值的值域介於 π 與 0 弧度之間。如果轉換為度數，則 360 度等於 2π 弧度。

範例

```
Sub Main()
    Dim Result As Float

    Result = ACOS(0.5)              ' the Result is 1.0472
    Debug "ACOS(0.5) = ", Result," in radians.", CR
End Sub
```

ASIN

語法

Result = ASIN(*Argument*)

操作

回傳浮點數的反正弦值。

- ***Argument*** - ASIN 函數的浮點運算元。值域介於-1 與 1 之間。
- ***Result*** - 接受 ASIN 函數回傳值的浮點變數。回傳值值域介於 - π/2 與 +π/2 弧度之間。

說明

ACOS 命令回傳浮點數運算元的反餘弦值(arccosine)。運算元的值域介於-1 與 1 之間。回傳值的值域介於 π 與 0 弧度之間。如果轉換為度數，則 360 度等於 2π 弧度。

範例

```
Sub Main()
    Dim Result As Float

    Result = ASIN(0.5)              ' the Result is 0.52360
    Debug "ASIN(0.5) = ", Result," in radians.", CR
End Sub
```

ATAN

語法

Result = ATAN(*Argument*)

操作

回傳浮點數的反正切值。

- ***Argument*** - ATAN 函數的浮點運算元。值域介於負無限大與正無限大之間。
- ***Result*** - 接受 ATAN 函數回傳值的浮點變數。回傳值值域介於 - π/2 與 +π/2 弧度之間。

說明

ATAN 命令回傳浮點數運算元的反正切值(arctangent)。運算元的值域介於負無限大與正無限大之間。回傳值的值域介於 - π/2 與 +π/2 弧度之間。如果轉換為度數，則 360 度等於 2π 弧度。

範例

```
Sub Main()
    Dim Result As Float

    Result = ATAN(0.5)              ' the Result is 0.46365
    Debug "ATAN(0.5) = ", Result," in radians.", CR
End Sub
```

ATAN2

語法

Result = ATAN2(*ArgumentY*, *ArgumentX*)

操作

回傳 X,Y 軸浮點數的反正切值。

- ***ArgumentY*** - ATAN 函數的 Y 軸浮點運算元。值域介於負無限大與正無限大之間。
- ***ArgumentX*** - ATAN 函數的 X 軸浮點運算元。值域介於負無限大與正無限大之間。
- ***Result*** - 接受 ATAN 函數回傳值的浮點變數。回傳值值域介於 - π/2 與 +π/2 弧度之間。

說明

ATAN2 命令回傳 X 與 Y 軸浮點數運算元的反正切值(arctangent)。運算元的值域介於負無限大與正無限大之間。回傳值的值域介於 - π/2 與 +π/2 弧度之間。如果轉換為度數，則 360 度等於 2π 弧度。

範例

```
Sub Main()
    Dim Result As Float

    Result = ATAN2(1,2)             ' the Result is 0.46365
    Debug "ATAN2(1,2) = ", Result," in radians.", CR
End Sub
```

BUTTON

語法

BUTTON *Pin*, *Onstate*, *Delay*, *Rate*, *LoopCounter*, *TargetState*, *Address*

操作

對於外接按鍵操作行為，提供程式分支和延遲的操作功能。

- ***Pin***- 常數或變數(0 ~23)，用來定義外接按鈕所連接到的引腳號碼。 例如一個 24-pin 的 BASIC Commander，它的值的範圍從 0~15。
- ***Onstate*** - 常數或變數(0 或 1)，表示當按下按鈕時的邏輯準位。如果連接的按鍵，輸入常態為 high，按鍵使之為 low，這裡的數值就設為 0。若輸入常態為 low，按鍵使之為 high，則數值應設為 1。
- ***Delay*** - 常數或變數(0 ~255)，指按鍵啟動自動重複功能的延遲時間。這個延遲用程式的週期數來計算。如果值為 0，將不會有延遲或自動重複功能。如果值為 255，則只有去彈跳(Debounce)功能，但不產生自動重複功能。這個參數可用來消除按鈕所造成的影響。
- ***Rate*** - 常數或變數(0 ~255)，指任兩個自動重複之間的時間，數值代表所需的程式周期數。
- ***LoopCounter*** - 位元組變數，當 BUTTON 命令重覆執行時的迴圈計數器。它的值在第一次執行 BUTTON 命令前要清除為 0,而且之後的使用者不可更改內容值。
- ***TargetState*** - 常數或變數(0 或 1)，指定引腳的狀態，依此產生程式分支。若數值為 0，不壓按鍵就會產生分支。若數值為 1，壓下按鍵時就發生分支。
- ***Address*** - 當按鍵符合 TargetState 陳述時,程式分支到 Address 標籤處。

說明

絕大多數的設計上，都有外接按鍵的部份，提供控制按鍵組合，以及按下外接按鍵時的反應動作。按鍵是機械裝置，所以第一次按壓時，在穩定之前，內部連結會在幾毫秒間來回彈跳。在消除彈跳的這段時間裡，low 和 high 的訊號都會偵測到，造

成誤判。為了避免這種情況發生，通常會加入約 20ms 的延遲，再開始判讀按鍵值。在 BUTTON 命令中，可使用延遲參數完成目的，當偵測到第一個按鍵訊號後，會等候一段大於彈跳的時間，再偵測其他按鍵訊號。時間長度由 delay 參數指定，每當執行 BUTTON 命令時，就由指定的數值倒數。倒數至零，若按鍵仍是有效狀態，就產生程式分支。此時，計數器又開始倒數，到達零時，再執行另一個程式分支。以此種方式重現按鍵功能，類似一般電腦鍵盤的運作方式。下列圖示，由 high 切換到 low 時，機械的彈跳作用。

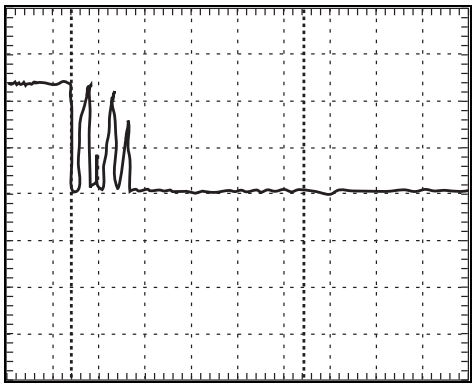


圖 6-1 按鍵機械彈跳

可以連接輸入按鍵，當按下按鍵時輸入引腳可以變為 low 或是 high 的狀態。下圖顯示這兩種連結的方式。

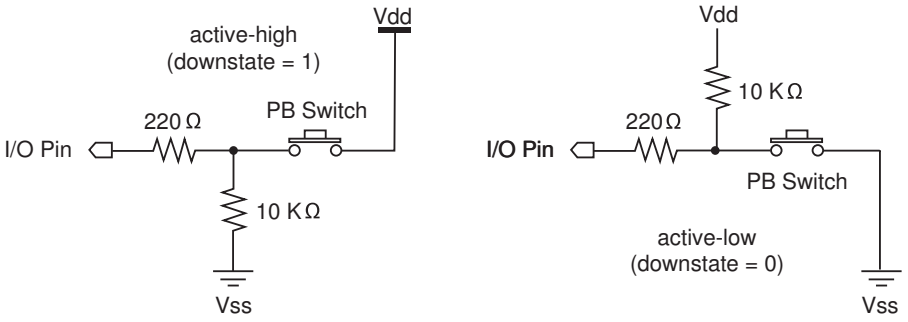


圖 6-2　Active-low 和 Active-high 按鍵的電路

範例

將低壓動作 (active-low)按鍵電路，接在 BASIC Commander 的引腳 P0。第一次按鍵時，測試程式會在終端視窗顯示出星號“*”，在自動重複開始前延遲 2 秒。自動重複功能會以大約 200ms(20 x 10 ms 暫停)的速率，送出連續的按鍵訊號。

```
Sub Main()
    Dim PIN As Byte
    Dim LoopCounter As Byte = 0    ‘ cleared before BUTTON is used

Start:
    Pause 10
    BUTTON 0, 0, 200, 20, LoopCounter, 1, Display
    Goto Start
Display:
    Debug "*"
    Goto Start
End Sub
```

改變 Delay 值，看看在不同模式下有何影響。“0”表示無延遲，自動重複功能會立即開始；“1”到“254”代表自動重複開始前的延遲時間；“255”則表示沒有重複功能，按下按鍵，只有一個反應。

BYTE2FLOAT

語法

Result = BYTE2FLOAT(*Argument*)

操作

將位元組轉換為浮點數值形態格式。

- ***Argument*** - BYTE2FLOAT 函數的位元組運算元。
- ***Result*** -浮點變數接受 BYTE2FLOAT 函數結果。

說明

BYTE2FLOAT 命令將位元組數值轉換為浮點數值的格式，浮點數結果是介於 0.0 到 255.0 的整數。

範例

```
Sub Main()
    Dim MyByte As Byte
    Dim MyFloat As Float
    MyByte = 0
    MyFloat = BYTE2FLOAT(MyByte)
    Debug "MyFloat = ", MyFloat, CR
    MyByte = 255
    MyFloat = BYTE2FLOAT(MyByte)
    Debug "MyFloat = ", MyFloat, CR
End Sub
```

CALL

語法
{CALL} *Name*({*Arglist*})

操作
呼叫一個程序，包含一個額外的參數列。

- ***Name*** – 程序名稱，包含一串字母，數字和底線。開頭字元必須為字母。
- ***Arglist*** – 程序中所需要的參數列。參數是以傳值(byval)或傳址(byref)方式傳遞。如果沒有參數，括弧可以刪除。

說明
關鍵字 CALL 是額外的。命令被執行時，程式會分支到 *Name* 所指定的程序。當程序遇到 END SUB 或 RETURN 命令時，程式會回到呼叫它的敘述。

範例

```
Sub SayHello()
    Debug "Hello!", CR
End Sub

Sub Main()
    Call SayHello()
End Sub
```

CEIL

語法

Result = CEIL(*Argument*)

操作

傳回最接近但不小於浮點參數的整數值。

- ***Argument*** – CEIL 函數的浮點運算元。
- ***Result*** – 浮點變數用來接收 CEIL 函數結果。

說明

CEIL 命令傳回最接近但不小於浮點參數的整數值(浮點值)。通常是用來將浮點數四捨五入為整數。配對函數為 FLOOR 函數，用來傳回最接近但不大於浮點參數的整數值(浮點值)。

範例

```
Sub Main()
    Dim Result As Float

    Result = CEIL(2.3)              ' the result is 3.0
    Debug "CEIL of 2.3 is ", result,CR
    Result = CEIL(-2.3)             ' the result is -2.0
    Debug "CEIL of -2.3 is ", result,CR
End Sub
```

CHECKMODULE

語法

Statust = Checkmodule()

操作

用 cmdBUS(TM)狀態來檢查週邊模組。

- ***Statust*** – 接收模組狀態的變數。如果週邊模組通訊正常，回傳 0。如果週邊模組在指定時間內沒有回應，回傳 1。如果 cmdBUS(TM)執行這些協定失敗，回傳 2。

說明

當週邊模組在應用程式中使用時，為了要得到更可靠的系統操作，使用者可以用這個命令去監控週邊模組以及 cmdBUS(TM)的故障狀態，可能是無法預期的環境電波或電磁干擾。

如果週邊模組通訊正常，回傳 0。如果週邊模組在指定時間內沒有反應，回傳 1。通常都可以事先知道週邊模組無法正常操作。如果 cmdBUS(TM)執行這些協定失敗，回傳 2。這在 cmdBUS(TM)上是十分嚴重的系統故障，並且會暫停週邊模組的訊號系統。原因可能由任一個連接到 cmdBUS(TM)的週邊模組甚至是 BASIC Commander 所引起。請注意，狀態檢查命令會傳回最新的週邊模組狀態，它應該被放在在週邊命令之後。一旦偵測到系統故障，就採取必要的步驟來控制故障，沒有內建的回復方法。

範例

```
Peripheral myLCD As LCD2X16A @ 0


Sub Main()
    Dim Status As Byte
```

```
    myLCD.Display("Hi")
    Status = checkmodule()

    If Status = 1 Then
        Debug "Module Timeout!",CR
    Elseif Status = 2 Then
        Debug "cmdBUS Error!",CR
    Else
        Debug "Command executed successfully!"
    End If
End Sub
```

COS

語法

Result = COS(*Argument*)

操作

執行數學 cosine 函數。

- ***Argument*** - cosine 函數的浮點運算元，範圍為 0 到 2π。
- ***Result*** - 接收 cosine 函數值結果的浮點變數。

說明

COS 函數回傳浮點參數的 cosine 值，範圍為 0 到 2π。參數的單位為弧度。如果轉換為度數，則 360 度等於 2π 弧度。

範例

```
Sub Main()
    Dim myArg As Float
    Dim result As Float

    myArg = pi/4
    result = cos(myArg)            ' the result is 0.707107
    Debug "cos(pi/4) = ",result,CR
End Sub
```

COUNT

語法

COUNT *Pin, Duration, Variable*

操作

計算在一段時間中出現在指定引腳的 0-1-0(或 1-0-1)轉換的周期數，將值放入指定的變數。

- ***Pin*** – 常數或變數(0 ~23)，用來定義轉換計數的引腳號碼。對一個 24-pin 的 BASIC Commander 而言，它的引腳值範圍為 0~15。
- ***Duration*** – 常數或變數(1 ~65535)，用來定義轉換被計數的期間。*Duration* 的單位為一毫秒(1 ms)。
- ***Variable*** – WORD 變數(1~65535)，用來存放計數值。

說明

這個命令計算出在指定引腳上出現 0-1-0(或 1-0-1)轉換的周期數，可以計算外部事件改變的數量。執行 COUNT 指定的引腳在執行時會自動被設為輸入。*Duration* 定義時間單位為 1 毫秒(ms)。每一次由高到低或由低到高的輸入訊號最小幅寬必須大於 10us。換句話說，最大的輸入訊號頻率必須是不大於 50k 赫茲同等功率的方形波，否則有些轉換無法被計算到。如果高低功率不相同，較短的功率要大於 10ms。換句話說，最大輸入頻率被較短功率限定而會遠小於 50k 赫茲。如果計數大於 65535，將會產生溢位 0 然後繼續計數。我們要預防這種情況。

範例

程式教我們用 COUNT 命令去完成一個有趣的小遊戲，測試你按鍵的速度能有多快。按鍵的應用線路圖如下圖。

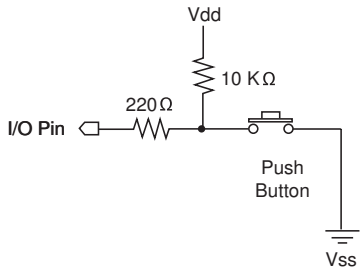


圖 6-3　按鍵的電路

```
Sub Main()
    Dim  PushBtn As Byte=0      ' Push button on P0
    Dim  Cycles As Word=0       ' Counted cycles
    Dim m As Byte=0

    Do
        Debug CLS,_
        "How fast can you press within 5 seconds?", CR
        Pause 1000
        Debug "Ready!",CR
        Pause 1000
        Debug "Set!",CR
        Pause 2000
        Debug "Go!", CR

        COUNT PushBtn, 5000, Cycles

        Debug CR, "Your score is ", Cycles, CR, CR
        Pause 2000
        Debug "Press button to try again!"
        Do
```

```
            m=in(Pushbtn)
        Loop Until m=0
    Loop
End Sub
```

DEBUG

語法

DEBUG *Item* {,*Item*}

操作

DEBUG 命令讓 BASIC Commander 跟使用者藉由顯示訊息、控制碼或數值在 innoBASIC Workshop 的終端機視窗中作溝通。

- *Item* – 用來顯示在終端機視窗上的訊息、控制碼或變數。如果超過一個可以用冒號分開。

說明

有時意思需要 BASIC Commander 跟使用者去做溝通對話，方法是藉由在 innoBASIC Workshop 的終端機視窗上對程式下 DEBUG 命令。程式除錯是在應用程式發展中的基礎部分，因為很少程式可以在第一次就做對的。因此在程式中加入 DEBUG 命令可以讓你了解程式執行到的位置以及它的變數內容。例如：

```
DEBUG "This is the wrong path, and the variable i is", i
```

除錯命令可以放置在例外的地方，當分支執行時會顯示相關的變數值 i 幫助診斷錯誤。

控制碼

除了除錯外，DEBUG 也是很有用的互動人機介面。為了提供更符合使用者使用的控制台顯示，我們另外提供了控制碼。這些特殊的控制碼摘要在下表：

函數	敘述
CLS	清除螢幕
CR	歸位(換行)

TAB	放置一個 tab
CSRL	游標左移
CSRR	游標右移
CSRU	游標往上
CSRD	游標往下
BKSP	游標強制返回
CLREOL	清除從游標到行尾
CLREOS	清除從游標到螢幕結尾
CSRXY（x,y）	將游標移到行 x 列 y
CSRX（x）	將游標移到行 x
CSRY（y）	將游標移到列 y
BELL(n)	產生一個 Windows 內建的音效

資料格式化程式

數字資料可以各種形式顯示在終端機視窗中。如果沒有特別定義，數值將會以小數形式顯示。以上面的例子看來，這並不會造成任何問題而值可以正確的顯示。不過如果要顯示成二進位或十六進位格式，則要用符號”%”加上想要的格式。下表包含了所有的用法：

格式化程式符號	敘述
?	如果使用“?”格式化程式，額外的字串“*symbol* =” 會加在顯示值的前面，並於顯示值後換行。*symbol* 代表一個使用者定義的變數名稱。
%DEC{n{L|R}}	資料以帶正負值小數格式顯示，額外的 n 代表行寬。如果 n 值小於實際的數字，寬度會自動延展到符合實際數字寬度。額外的值 L 跟 R 代表左右對齊。如果刪除 L 或 R，預設為靠左對齊。在顯示時前導的數字 0 會被刪除。
%BIN{n{L|R}}	資料以不帶正負值的二進位顯示，額外的 n 代表行寬。如果 n 值小於實際的數字，8 位元變數是 8，16 位元變數是 16，寬度會自動延展到符合實際數字寬度。額外的值 L 跟

	R 代表左右對齊。如果刪除 L 或 R，預設為靠左對齊。在 8 或 16 位元的數字資料格式中前導的數字 0 會顯示。
%HEX{n{L|R}}	資料以不帶正負值的十六進位顯示，額外的 n 代表行寬。如果 n 值小於實際的數字，8 位元變數是 2，16 位元變數是 4，寬度會自動延展到符合實際數字寬度。額外的值 L 跟 R 代表左右對齊。如果刪除 L 或 R，預設為靠左對齊。在 2 或 4 位數的數字資料格式中前導的數字 0 會顯示。
%CHR	資料以 ASCII 字元格式顯示
%FLOAT{n.m{L|R}}	以科學格式顯示浮點資料，額外的 n 代表行寬額外的 m 代表 1 到 5 位的有效位數。如果 n 值小於實際的數字，8 位元變數是 2，16 位元變數是 4，寬度會自動延展到符合實際數字寬度。額外的值 L 跟 R 代表左右對齊。如果刪除 L 或 R，預設為靠左對齊。
%REAL{n.m{L|R}}	跟%FLOAT{n.m{L|R}}上面所提的一樣，除了浮點數是實數格式。
%REP{n}	重複常數或變數 n 次。如果沒有設定額外的 n 值，預設值為 1。

上面的例子的 DEBUG 命令可以修改為將變數顯示成如下的二進位格式：

```
Debug "The value for i is ", %BIN i
```

注意：DEBUG 命令需要一些時間藉由 USB 介面在 BASIC Commander 和 PC 間作溝通。有時間考量的應用程式，要預防在程式關鍵路徑中使用 DEBUG 命令。在開發階段使用 DEBUG 命令除錯，時脈可能會跟 DEBUG 命令移除做正式操作時有所不同。

範例

```
Sub main()
```

```
    Dim X As Byte = 100

    Debug ? X              ‘ shows "X=100" with carriage return
    Debug "Column 10, aligned to the left ", %DEC10L X,CR
    Debug "Column 10, aligned to the right ", %DEC10R X,CR
End Sub
```

DEBUGIN

語法

DEBUGIN *Item* { , *Item*}

操作

這個命令允許使用者在運轉時間內經由終端機視窗送資料到 BASIC Commander。

- ***Item*** – 從除錯控制台接收的資料變數。為了操作上的方便，Debugin 命令也支援顯示在終端機視窗的訊息及控制碼，使用者可以了解等待輸入的是哪種資訊。否則使用者要靠另一個除錯命令來提供這些資料。超過一個時可以用冒號分開。

說明

有時需要 BASIC Commander 跟使用者去做溝通對話，可藉由在 innoBASIC Workshop 的終端視窗上對程式下 DEBUGIN 命令。這個方法可以用來做程式除錯，或當作從使用者端獲得資料的人機介面。例如：

```
DEBUGIN "Please enter your lucky number.", num, CR
```

其它說明事項與 DEBUG 命令相同，使用時請參考 DEBUG 命令。

注意：DEBUGIN 命令需要一些時間藉由 USB 介面在 BASIC Commander 和 PC 間作溝通。有時間考量的應用程式，要預防在程式關鍵路徑中使用 DEBUGIN 命令。在開發階段使用 DEBUGIN 命令除錯，時脈可能會跟 DEBUGIN 命令移除做正式操作時有所不同。如果 DEBUGIN 命令沒有被移除獨立操作，當 USB 介面沒有和 PC 連接時，程式會進入迴圈無止盡的等待資料輸入。

範例

```
Sub Main()
```

```
Dim yourname As String * 20
    Dim  Key As Byte

    Debugin "Please enter your name.", yourname, CR
    Debug "Hi ",yourname,"!"
    Do
        Debugin CR,"Enter in DEC: ", Key, CR        ‘ say, 100
        Debug "The number in DEC is: ",Key, CR

        Debugin CR,"Enter in BIN: ", %BIN Key, CR   ‘say, 01010101
        Debug "The number in BIN is: ",%BIN Key, CR

        Debugin CR,"Enter HEX: ", %HEX Key, CR      ‘say, FA
        Debug "The number in Hex is: ",%HEX Key, CR

        Debugin CR,"Enter a letter: ", %CHR Key, CR
        Debug "The letter is ", %CHR Key,", ASCII Code is ", Key, CR
    Loop
End Sub
```

DIM

語法

DIM *Variable* AS *Type* {* *Size*}

操作

宣告區域或全域變數。

- ***Variable*** – 用來儲存值的變數。
- ***Type*** – 合乎規定的變數形態名稱，包含布林值、位元組、整數、字元組、長整數、浮點數、持久位元組、持久整數、持久字元組、持久長整數和持久浮點數。
- ***Size*** – 定義字串形態變數大小的常數，字串的大小受限於可獲得的 RAM 資料記憶體。

說明

變數要先做宣告。傳統的變數資料型態有布林值、位元組、短整數、字元組、整數、雙字元組、長整數、浮點數和字串形態，用於 RAM 資料記憶體，其它 innoBASIC 特殊的變數形態有持久形態變數，用於持久性 EEPROM 資料記憶體。如果 DIM 在程序中，變數宣告為區域變數，它們只有在程序內部才能看的到。相反的，如果 DIM 是用在所有程序外，變數就是全域變數，也就是說它們在程式的任何地方都可以看到。持久形態變數就是全域，所以持久形態變數必須在所有程序外作宣告。

不像基本變數，字串文字可以儲存超過一個字元非數值的值。因為 RAM 資源的限制，宣告的字串大小要適當。

範例

```
    Dim G As Byte     ' global variable, no initializer


Sub Main()
    Dim X,Y As Byte
```

```
    Dim Z As Short =-1     ' local variable, optional initializer
    Debug ? G
    Debug ? X
    Debug ? Y
    Debug ? Z
End Sub
```

DO…LOOP

語法

DO {*Modifier Condition*}
{*statements*}
LOOP {*Modifier Condition*}

操作

一個命令重複執行的程式迴圈，不需依賴使用者定義的條件。

- ***Modifier*** – WHILE 或 UNTIL 的額外的修飾辭，放置在 DO 或 LOOP 的後面，不是兩者都用。WHILE 修飾辭被使用時，如果 *condition* 一直為真，則迴圈會繼續。UNTIL 修飾辭被使用時，則迴圈會繼續直到 *condition* 為真。
- ***Condition*** – 布林值。
- ***Statements*** – 額外的有效敘述，包含額外的 EXIT DO 或 CONTINUE 命令。

說明

基本的DO … LOOP命令會構成一個無窮迴圈，被DO和LOOP包圍的命令會重複執行。你可以在迴圈開始或結束的地方加上WHILE修飾辭，布林值會被測試。只要條件為真，迴圈主體就會執行。如果WHILE放在迴圈結束的地方，迴圈至少會執行一次。UNTIL修飾辭跟WHILE修飾辭相似，除了迴圈在布林值為真時是結束而不是繼續。敘述可以包含額外的EXIT DO或CONTINUE 命令。EXIT DO命令可以放置在迴圈主體中，在迴圈測試前就跳離現在的迴圈。CONTINUE命令 將程式執行移轉到迴圈程式段的尾端，然後在做下一個重複。更進一步詳細的解釋請參照第五章。

範例

更進一步的例子請參照第五章。

DWORD2FLOAT

語法

Result = DWORD2FLOAT(*Argument*)

操作

轉換 DWord 值成浮點數值。

- ***Argument*** - LONG2FLOAT 函數的 DWord 運算元。
- ***Result*** - 承接 DWORD2FLOAT 函數回傳值的浮點變數值。

說明

FLOAT2DWORD 命令將 DWord 型態數值轉換為浮點型態數值。浮點形態數值的結果範圍是 0 到 4294967295 的整數值。由於使用的是單精確度浮點數值，變數可能無法很精確的表示出來。回傳的是最接近的浮點整數值。當你在應用程式中使用這個命令時需要很小心

你可以利用二進位形式來檢視 DWord 值是否被精確的表示。去除二進位前端及後端的 0，如果剩餘的位元數大於 24，則該數值無法精確表示。

範例

由於使用的是單精確度浮點數，數值 4294967295 無法精確的顯示。回傳的值會是最接近的 4294967296。

```
Sub Main()
    Dim MyDword As Dword
    Dim MyFloat As Float

    MyDword = 4294967295
    MyFloat = DWORD2FLOAT(MyDword)
```

```
    Debug "MyDword = ", MyDword,CR, "MyFloat = ",MyFloat,CR
End Sub
```

ENUM … END ENUM

語法

ENUM *Identifier*
EnumeratorList
END ENUM

操作

宣告一個列舉。

- *Identifier* - 列舉名稱。
- *EnumeratorList* - 所有列舉的成員，包含額外的賦予值。

說明

ENUM 命令用來宣告常數列舉(Enumeration)。列舉值必須被宣告在所有程序外，也就是全域宣告，而且只能公共存取。列舉成員可以使用“=“賦予其列舉值。如果列舉的第一個成員沒有賦予列舉值，則列舉值為 0。其餘列舉成員如果沒有賦予值，則其列舉值會是前一個列舉值加 1。下面的例子顯示如何使用 ENUM 命令。使用時在列舉名稱跟列舉成員中間使用”.”運算元來存取其列舉值。下面例子顯示如何使用 ENUM 命令。

範例

```
Enum Color
     Red
     Yellow = 3
     Blue = 1
     Green
End Enum

Sub Main()
```

```
    Debug "Enumerator Red = ", Color.Red, CR
    Debug "Enumerator Yellow = ", Color.Yellow, CR
    Debug "Enumerator Blue = ", Color.Blue, CR
    Debug "Enumerator Green = ", Color.Green, CR
End Sub
```

EVENT… END EVENT

語法

EVENT *ModuleName.EventName*()
{*statements*}
END EVENT

操作

宣告一個事件程序。

- *ModuleName* - 使用者宣告的事件模組名稱。
- *EventName* - 事件名稱。

說明

EVENT 命令宣告一個程序，當一個特定的事件發生時會被週邊模組所呼叫。即使沒有參數，括號也不能省略。當宣告事件程序時，事件名稱是由兩個部分組成。第一個部分是模組名稱而第二個部分是事件名稱。兩個名稱由一個點”. ”連接。下面例子顯示事件宣告以及如何啟動事件函數。

範例

```
Peripheral MyKeypad As KeypadA @ 0

Event MyKeypad.KeyPressed()
    Dim KeyID As Byte
    MyKeypad.GetKeyID(KeyID)
    Debug "Key ", KeyID, " is Pressed!", CR
End Event

Sub Main()
    MyKeypad.EnableKeypadEvent()    ' 啟動 EVENT
```

```
    Debug "Press Key Pad.", CR
    Do:Loop
End Sub
```

當程式在 EVENT 程序中時，其它事件會被暫停直到 EVENT 程序執行完畢。強烈建議在 EVENT 程序中盡可能使用少量的敘述，讓主程式來處理其餘的敘述。

EXP

語法

Result = EXP(*Argument*)

操作

回傳浮點參數的指數值。

- ***Argument*** – EXP 函數的浮點運算元。
- ***Result*** – 接收 EXP 函數結果的浮點變數。

說明

EXP 函數回傳一個浮點參數的指數值。結果恆為正值。在數學中，它被表示成 $y = e^x$，e 是 Euler 數 2.71828….。它的反函數為對數 LOG 函數。

範例

```
Sub Main()
    Dim Result As Float

    Result = EXP(2.0)
    Debug "EXP of 2.0 is ", result, CR
    Result = EXP(-2.0)
    Debug "EXP of -2.0 is ", result, CR
End Sub
```

EXP10

語法

Result = EXP10(*Argument*)

操作

回傳一個浮點參數基底為 10 的指數值。

- ***Argument*** - EXP10 函數的浮點運算元。
- ***Result*** - 接收 EXP10 函數結果的浮點變數。

說明

EXP10 函數回傳一個浮點參數基底為 10 的指數值。結果恆為正值。在數學中，它被表示成 $y = 10^{x}$ ^{x}。它的反函數為基底 10 的對數 LOG10 函數。

範例

```
Sub Main()
    Dim Result As Float

    Result = EXP10(2.0)             ' 100
    Debug "EXP10 of 2.0 is ", result, CR
    Result = EXP10(-2.0)        ' 0.01
    Debug "EXP10 of -2.0 is ", result, CR
End Sub
```

FLOAT2BYTE

語法

Result = FLOAT2BYTE(*Argument*)

操作

將浮點數轉換為位元組。

- ***Argument*** – FLOAT2BYTE 函數的浮點運算元。
- ***Result*** – 用來接收 FLOAT2BYTE 函數結果的位元組變數。

說明

FLOAT2BYTE 命令將浮點數值轉換為位元組。小數的部分會被四捨五入。如果值超過位元組值的範圍，則可能會產生數值 0 或 255。

範例

```
Sub Main()
    Dim MyFloat As Float
    Dim MyByte As Byte

    MyFloat = 2.4
    MyByte = FLOAT2BYTE(MyFloat)        ' the result is 2
    Debug "FLOAT2BYTE of 2.4 : ", MyByte, CR

    MyFloat = 2.5
    MyByte = FLOAT2BYTE(MyFloat)        ' the result is 3
    Debug "FLOAT2BYTE of 2.5 : ", MyByte, CR

    MyFloat = -1.0
```

```
    MyByte = FLOAT2BYTE(MyFloat)        ' the result is 0
    Debug "FLOAT2BYTE of -1.0 : ", MyByte, CR

    MyFloat = 256.0
    MyByte = FLOAT2BYTE(MyFloat)        ' the result is 255
    Debug "FLOAT2BYTE of 256.0 : ", MyByte, CR
End Sub
```

FLOAT2DWORD

語法

Result = FLOAT2DWORD(*Argument*)

操作

將浮點數轉換為 DWord。

- ***Argument*** - FLOAT2DWORD 函數的浮點運算元。
- ***Result*** -用來接收 FLOAT2DWORD 函數結果的位元組變數。

說明

FLOAT2DWORD 命令將浮點數轉換為 DWord。小數的部分會被四捨五入。如果值超過 DWord 值的範圍，則會使用數值 0 或 4294967295。

範例

```
Sub Main()
    Dim MyFloat As Float
    Dim MyDword As Dword

    MyFloat = 2.4
    MyDword = FLOAT2DWORD(MyFloat)      ' the result is 2
    Debug "FLOAT2DWORD of 2.4 : ", MyDword, CR

    MyFloat = 2.5
    MyDword = FLOAT2DWORD(MyFloat)      ' the result is 3
    Debug "FLOAT2DWORD of 2.5 : ", MyDword, CR

    MyFloat = -1.0
```

```
    MyDword = FLOAT2DWORD(MyFloat)      ' the result is 0
    Debug "FLOAT2DWORD of -1.0 : ", MyDword, CR

    MyFloat = 4.3E9
    MyDword = FLOAT2DWORD(MyFloat) ' the result is 4294967295
    Debug "FLOAT2DWORD of 4.3E9 : ", MyDword, CR
End Sub
```

FLOAT2INTEGER

語法

Result = FLOAT2INTEGER(*Argument*)

操作

將浮點數轉換為整數。

- ***Argument*** – FLOAT2INTEGER 函數的浮點運算元。
- ***Result*** – 用來接收 FLOAT2INTEGER 函數結果的整數變數。

說明

FLOAT2INTEGER 命令將浮點數轉換為整數。小數的部分會被四捨五入。如果值超過整數值的範圍，則會使用數值+32767 或-32768。

範例

```
Sub Main()
    Dim MyFloat As Float
    Dim MyInteger As Integer

    MyInteger = 22
    Debug "MyInteger : ", MyInteger, CR

    MyFloat = 2.4
    MyInteger = FLOAT2INTEGER(MyFloat)  'the result is 2
    Debug "FLOAT2INTEGER of 2.4 : ", MyInteger, CR

    MyFloat = 2.5
    MyInteger = FLOAT2INTEGER(MyFloat)  ' the result is 3
```

```
    Debug "FLOAT2INTEGER of 2.5 : ", MyInteger, CR

    MyFloat = -1.0
    MyInteger = FLOAT2INTEGER(MyFloat)  ' the result is -1
    Debug "FLOAT2INTEGER of -1.0 : ", MyInteger, CR

    MyFloat = 32768.0
    MyInteger = FLOAT2INTEGER(MyFloat)  ' the result is 32767
    Debug "FLOAT2INTEGER of 32768.0 : ", MyInteger, CR

    MyFloat = -32769.0
    MyInteger = FLOAT2INTEGER(MyFloat)  ' the result is -32768
    Debug "FLOAT2INTEGER of -32769.0 : ", MyInteger, CR
End Sub
```

FLOAT2LONG

語法

Result = FLOAT2LONG(*Argument*)

操作

將浮點數轉換為 LONG。

- ***Argument*** - FLOAT2LONG 函數的浮點運算元。
- ***Result*** -用來接收 FLOAT2LONG 函數結果的 LONG 變數。

說明

FLOAT2LONG 命令將浮點數轉換為 LONG。小數的部分會被四捨五入。如果值超過 long 整數值的範圍，則會使用數值+2147483647 或-2147483648。

範例

```
Sub Main()
    Dim MyFloat As Float
    Dim MyLong As Long

    MyFloat = 2.4
    MyLong = FLOAT2LONG(MyFloat)        ' the result is 2
    Debug "FLOAT2LONG of 2.4 : ", MyLong, CR

    MyFloat = 2.5
    MyLong = FLOAT2LONG(MyFloat)        ' the result is 3
    Debug "FLOAT2LONG of 2.5 : ", MyLong, CR

    MyFloat = -1.0
```

```
    MyLong = FLOAT2LONG(MyFloat)        ' the result is -1
    Debug "FLOAT2LONG of -1.0 : ", MyLong, CR

    MyFloat = 4.3E9
    MyLong = FLOAT2LONG(MyFloat)        ' the result is 2147483647
    Debug "FLOAT2LONG of 4.3E9 : ", MyLong, CR

    MyFloat = -4.3E9
    MyLong = FLOAT2LONG(MyFloat)        ' the result is -2147483648
    Debug "FLOAT2LONG of -4.3E9 : ", MyLong, CR
End Sub
```

FLOAT2REALSTRING

語法

StringVar = FLOAT2REALSTRING(*Argument*)

操作

將浮點數轉換為實數格式 ASCII 字元字串。

- ***Argument*** – FLOAT2REALSTRING 函數的浮點運算元。
- ***StringVar*** – 用來接收轉換的結果的字串變數。

說明

FLOAT2REALSTRING 將浮點數轉換為實數格式 ASCII 字元字串。小數點後有五位有效尾數的基值。例如+0.31416。請注意，除了小數點後第一位，多出的 0 都會被刪除。例如 2 會被轉換成+2.0。

範例

```
Sub Main()
    Dim MyFloat As Float
    Dim MyString As String * 11

    MyFloat = 0.31416
    FLOAT2REALSTRING(MyFloat, MyString)
    Debug "FLOAT2REALSTRING of 0.31416 : ", MyString, CR
End Sub
```

FLOAT2SHORT

語法

Result = FLOAT2SHORT(*Argument*)

操作

將浮點數轉換為 SHORT 變數。

- ***Argument*** - FLOAT2SHORT 函數的浮點運算元。
- ***Result*** - 接收 FLOAT2SHORT 函數結果的 SHORT 變數。

說明

FLOAT2SHORT 命令將一個浮點數轉換為 SHORT。小數的部分會被四捨五入。如果值超過 SHORT 值的範圍，則會使用數值+127 或-128。

範例

```
Sub Main()
    Dim MyFloat As Float
    Dim MyShort As Short

    MyFloat = 2.4
    MyShort = FLOAT2SHORT(MyFloat)      ' the result is 2
    Debug "FLOAT2SHORT of 2.4 : ", MyShort, CR

    MyFloat = 2.5
    MyShort = FLOAT2SHORT(MyFloat)      ' the result is 3
    Debug "FLOAT2SHORT of 2.5 : ", MyShort, CR

    MyFloat = -1.0
    MyShort = FLOAT2SHORT(MyFloat)      ' the result is -1
```

```
    Debug "FLOAT2SHORT of -1.0 : ", MyShort, CR

    MyFloat = 128.0
    MyShort = FLOAT2SHORT(MyFloat)      ' the result is 127
    Debug "FLOAT2SHORT of 128 : ", MyShort, CR

    MyFloat = -129.0
    MyShort = FLOAT2SHORT(MyFloat)      ' the result is -128
    Debug "FLOAT2SHORT of -129 : ", MyShort, CR
End Sub
```

FLOAT2STRING

語法

StringVar = FLOAT2STRING(*Argument*)

操作

將浮點數轉換為浮點格式的 ASCII 字元字串。

- ***Argument*** - FLOAT2STRING 函數的浮點運算元。
- ***Result*** -用來接收轉換結果的字串變數。

說明

FLOAT2STRING 命令將浮點數轉換為浮點格式的 ASCII 字元字串，它是由一個正負符號字元，一個小數點後有五位尾數的基值，一個指數符號字元 E，指數符號字元和兩位數的指數。例如：+3.1416E-01。

範例

```
Sub Main()
    Dim MyFloat As Float
    Dim MyString As String * 13

    MyFloat = 0.31416
    FLOAT2STRING(MyFloat, MyString)
    Debug "FLOAT2STRING of 0.31416 : ", MyString, CR
End Sub
```

FLOAT2WORD

語法

Result = FLOAT2WORD(*Argument*)

操作

將浮點數轉換為 Word。

- *Argument* – FLOAT2WORD 函數的浮點運算元。
- *Result* –接收 FLOAT2WORD 函數結果的 Word 變數。

說明

FLOAT2WORD 命令將浮點數轉換為 Word。小數的部分會被四捨五入。如果值超過 Word 值的範圍，則會使用數值 0 或 65535。

範例

```
Sub Main()
    Dim MyFloat As Float
    Dim MyWord As Word

    MyFloat = 2.4
    MyWord = FLOAT2WORD(MyFloat)        ' the result is 2
    Debug "FLOAT2WORD of 2.4 : ", MyWord, CR

    MyFloat = 2.5
    MyWord = FLOAT2WORD(MyFloat)        ' the result is 3
    Debug "FLOAT2WORD of 2.5 : ", MyWord, CR
```

```
    MyFloat = -1.0
    MyWord = FLOAT2WORD(MyFloat)        ' the result is 0
    Debug "FLOAT2WORD of -1.0 : ", MyWord, CR

    MyFloat = 65536.0
    MyWord = FLOAT2WORD(MyFloat)        ' the result is 65535
    Debug "FLOAT2WORD of 65536 : ", MyWord, CR
End Sub
```

FLOOR

語法
Result = FLOOR(*Argument*)

操作
回傳最接近但不大於浮點參數的整數值。

- ***Argument*** - FLOOR 函數的浮點運算元。
- ***Result*** - 用來接收 FLOOR 函數結果的浮點變數。

說明
FLOOR 命令傳回最接近但不大於浮點參數的整數值(浮點值)。通常是用來將浮點數四捨五入為整數。配對的函數為 CEIL 函數，用來傳回最接近但不小於浮點參數值的整數值(浮點值)。

範例

```
Sub Main()
    Dim Result As Float
    Result = FLOOR(2.3)                          ' the result is 2.0
    Debug "FLOOR of 2.3 is ", result, CR
    Result = FLOOR(-2.3)                     ' the result is -3.0
    Debug "FLOOR of -2.3 is ", result, CR
End Sub
```

FOR…NEXT

語法

FOR *Index* = *StartValue* TO *EndValue* {STEP *StepValue*}
{*statements*}
NEXT {*Index*}

操作

在 FOR 跟 NEXT 命令間建立一個可重複的迴圈。

- ***Index*** – 一個變數，用來儲存一個以數字表示的值，用來控制迴圈執行的次數。為了程式的清楚起見，建議在 NEXT 後加一個額外的索引。
- ***StartValue*** – 一個定義計數器起始值的常數或變數。
- ***EndValue*** – 一個定義計數器結束值的常數或變數。如果計數器值超出這個值，程式會在下一次執行到 NEXT 命令時跳出 FOR/NEXT 迴圈然後執行 NEXT 的下一個命令。
- ***StepValue*** – 常數或變數，用來定義每次迴圈執行時所要增加的索引值。如果沒有使用 STEP 命令，則預設值 1 會被使用。
- ***Statements*** – 額外的合法敘述，包含 EXIT FOR 或 CONTINUE 命令。

說明

FOR 跟 NEXT 命令會建立一個可重複的迴圈，有許多敘述存在迴圈主體中。每一次程式遇到 NEXT 命令就會回到 FOR 命令。第一次遇到 FOR 命令時，變數 *Index* 包含 StartValue 和下一次重複就會被載入，*Index* 以 *StepValue* 作遞增。如果新的 *Index* 值沒有超過 *EndValue*，迴圈主體就會被再一次執行，否則程式會繼續 NEXT 命令後的敘述。其他的迴圈可以巢狀式存在一個 FOR… NEXT 迴圈中，巢狀的最大限制是程式空間。敘述可以包含額外的 EXIT FOR 或 CONTINUE 命令。EXIT FOR 命令可以放置於迴圈主體中，它可以在迴圈限制測試執行前就立即離開目前的迴圈。CONTINUE 命令將執行轉換到迴圈區塊結束處然後開始下一個重複。

範例

更詳細的例子請參照第五章。

FUNCTION… END FUNCTION

語法

FUNCTION *FunctionName*({*Arglist*}) AS *ReturnType*
{*statements*}
END FUNCTION

操作

宣告有額外參數串列的函數。

- ***FunctionName*** – 函數名稱，可以包含字母、數字和底線，但開頭必須為一個字母。
- ***Arglist*** – 函數所需的參數列。每個參數前都會加上一個傳值(byval)或傳址(byref)修飾辭以顯示參數傳遞方法。即使沒有參數括號也不能省略。
- ***ReturnType*** – 回傳的資料型態，可以是位元組、整數、字元組、長整數或浮點數。

說明

FUNCTION 命令宣告由函數呼叫執行的使用者定義函數。與 Sub 程式不同的是有回傳值給呼叫者。

範例

```
Function Sum(X As Short,Y As Short) As Integer
    Return X+Y
End Function

Sub main()
    Dim X, Y As Short
    Dim Z As Integer
```

```
    X=1
    Y=2
    Z=Sum(X,Y)
    Debug "X=", X, CR, "Y=", Y, CR, "Z=X+Y=", Z, CR
End Sub
```

GETDIRPORT

語法

Result = GETDIRPORT *Port*

如果連接埠號碼是一個常數，你可以使用下列任一種格式型態。
Result = GETDIRPORT0
Result = GETDIRPORT1
Result = GETDIRPORT2.

操作

取得指定的連接埠的 I/O 方向設定。

- *Port* – 常數或變數值(0-2)，用來指定連接埠號碼。連接埠 0 包含引腳 P0-P7，連接埠 1 包含引腳 P8-P15，連接埠 2 包含引腳 P16-P23。對一個 24 引腳的 BASIC Commander 來說，連接埠的值是 0 或 1。
- *Result* – 接收連接埠方向設定的位元組變數。

說明

用 GETPORT 命令來讀取應用程式中指定引腳現在的 I/O 方向設定， I/O 方向可以在程式執行中作輸入輸出模式切換。0 表示輸出，1 表示輸入模式。索引值較小的引腳為資料位元組中順序較低的位元。例如，P0 的設定會出現在資料讀取的位元 0。I/O 方向在程式開始後預設為輸入。

範例

下面的例子改變了 P0 的 I/O 方向，它是經由 7 個電阻將 7 個 LED 連接到 VCC，而 LED 會依此產生相對應的 on/off。

```
Sub Main()
    Dim  Key  As Byte
```

```
    Dim  PortStatus As Byte

Start:                    ' P0 ~ P7 are default input mode.
    WRITEPORT0 &H00       ' Write low to output buffers
    Do
        Debugin "Input any key to turn on LED 0, 2, 4, 6: ",%CHR Key, CR
        SETDIRPORT 0,&HAA     ' Switch P0, P2, P4, P6 to OUTPUT mode
        PortStatus=GETDIRPORT0
        Debug "Port0 status is: ", %BIN PortStatus, CR,CR

        Debugin "Input any key to turn on LED 1, 3, 5, 7: ",%CHR Key, CR
        SETDIRPORT 0,&H55     ' Switch P1, P3, P5, P7 to OUTPUT mode
        PortStatus=GETDIRPORT0
        Debug "Port0 status is: ", %BIN PortStatus,CR,CR
    Loop
End Sub
```

GOTO

語法
GOTO *LabelName*

操作
GOTO 命令會將程式移轉到使用者指定的標籤位置.

- ***LabelName*** – 標籤定義程式分支要去的地方。

說明
這個位置是由跟隨在 GOTO 命令後的標籤名稱所給定。GOTO 命令後的敘述並不會被執行。標籤所定義的就是 GOTO 命令的下一個命令。 因此 GOTO 命令被使用者用來做為控制程式的方式。一般並不建議使用者在他們的程式中用 GOTO 命令，因為常常會造成程式閱讀上的困難。而對一個巢狀敘述架構來說，要分支到最外接的迴圈，GOTO 敘述的使用就顯得十分方便。

範例

下面範例顯示出如何使用 GOTO 命令。你可以發現有很多的結構式敘述。

```
Sub Main()

Start:
    Debug ”Tick!”,CR
    Pause 1000
    Goto Start

End Sub
```

HIGH

語法
HIGH *Pin*

操作
將指定的引腳設為邏輯高準位。

- ***Pin*** – 常數或變數(0 ~23)，用來定義高準位要應用到的引腳。對一個 24-pin 的 BASIC Commander 而言，它的引腳值範圍為 0~15。

說明
這個命令會將指定的引腳設為接近 5 伏特的高準位。一般來說，在執行對應的輸入輸出操作前引腳要預先改變成輸入或輸出模式。然而在 HIGH 命令運作時只有一個引腳會參與其中，所以命令會自動將這引腳改變為輸出模式。使用者無須在命令執行前用手動方式去將引腳改變為輸出模式。

下圖所顯示的是一個用 HIGH 跟 LOW 命令去開啟小 LED 的簡單電路。當圖中的引腳執行 HIGH 命令時 LED 將會關閉，執行 LOW 命令時會被開啟。每一個 I/O 引腳都有約 20 微安的汲入電流能力及約 10 微安的輸出電流能力，也就是說引腳是 low 時，裝置會消耗掉更多的電流。

圖 6-4　LED 的電路

引腳有電流驅動的極限，像驅動較大的負載如燈泡或繼電器，會需要比 LED 更多的電流，所以需要將電晶體開關連接到輸出。下圖就是這種操作的簡單電路圖。

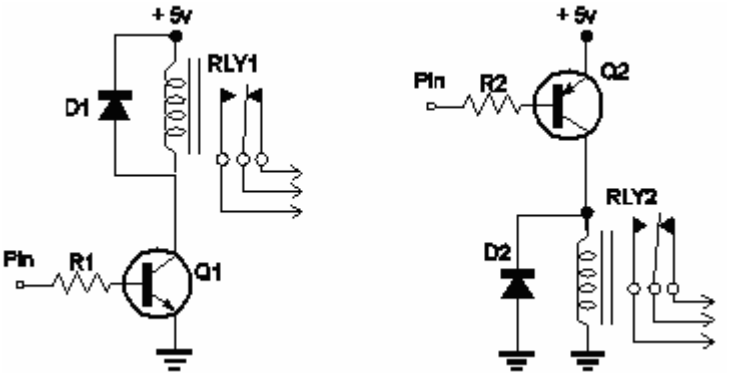


圖 6-5　繼電器的電路

重要的是要查看製造廠的資料頁來決定繼電器線圈電流的要求，確保電晶體可以控制這樣的電流。

範例

下列程式可以讓上圖顯示的 LED 電路中連接到引腳 0 的 LED 以 1 赫茲(Hz)的頻率閃爍。

```
Sub Main()
    Do
        HIGH 0        'LED on
        Pause 500     '0.5 second delay
        LOW 0         'LED off
        Pause 500     '0.5 second delay
        TOGGLE 0 'LED on
```

```
        Pause 500    '0.5 second delay
        TOGGLE 0 'LED off
        Pause 500    '0.5 second delay
    Loop
End Sub
```

IF …THEN … ELSE

語法

IF …Then … Else 命令有兩種形式，單行型式與區塊型式的 IF …Then … Else 命令。

單行型式：

IF ***Condition(s)*** **Then** ***Statement(s)*** **{Else** ***Statement(s)*****}**

區塊型式：

If ***Condition(s)*** **Then**
Statement(s)
{Elseif ***Condition(s)*** **Then**
Statement(s) **…}**
{Else
Statement(s)**}**
End If

操作

IF …Then … Else 命令依表示式的真偽值來執行。

- ***Condition(s)*** – 布林值條件表示式。
- ***Statement(s)*** – 有效的 innoBASIC 敘述。

說明

If...Then...Else 命令是基本的條件式命令。每一個 If...Then...Else 命令中的表示式都要被轉換成布林值。如果在 If 命令中的表示式為「真」，那包含在 Then 程式段中的敘述就會被執行。如果表示式為「偽」，那每一個 ElseIf 值都會被比較。如果其中一個 ElseIf 表示式的值為「真」，那它所對應到的程式段就會被執行。如果沒有值符合「真」而有另一個 Else 程式段，那 Else 程式段就會被執行。一旦有程式段被執行，程式就會執行到 End If 命令。單行型式的 If...Then...Else 命令適

用於在 Then 跟 Else 後面只有一個敘述的程式。如果加上一個以上的敘述，只要在兩個敘述間加上”：”即可。

範例

更詳細的例子請參照第五章。

IN

語法
Result = IN *Pin*

操作
從引腳讀取外部的邏輯輸入值。

- ***Pin*** – 常數或變數(0 ~23)，用來定義要讀取高階的引腳。對一個 24-pin 的 BASIC Commander 而言，它的引腳值範圍為 0~15。
- ***Result*** – 用來接收 *Pin* 所得到的外部邏輯值的位元組變數。

說明
這個命令會從引腳讀取外部的邏輯輸入值。通常，引腳要預先變成輸入模式。而 IN 命令運作時只有一個引腳會參與其中，系統會自動將這引腳改變為輸入模式。使用者無須在命令執行前用手動方式去將引腳改變為輸入模式。當電源開啟時，所以的 I/O 都變成輸入模式。

範例
以下例子告訴我們如何利用 IN 命令去讀取外部邏輯值。

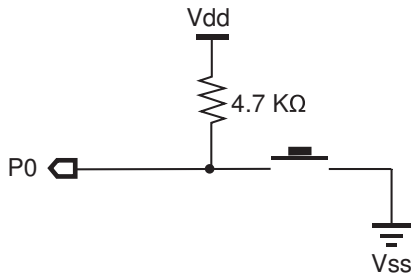


圖 6-6　輸入電路

```
Sub Main()
```

```
    Do
Wait:
    If IN(0) = 1 Then
        Pause 10
        Goto Wait
    Else
        Debug "Button is pressed.", CR
Release:
        If IN(0) = 0 Then Goto Release
        Debug "Button is released.", CR
        End If
    Loop
End Sub
```

INPUT

語法
INPUT *Pin*

操作
將指定的 *Pin* 設定為輸入模式。

- ***Pin*** – 常數或變數(0 ~23)，用來指定要被設為輸入模式的引腳。對一個 24-pin 的 BASIC Commander 而言，它的引腳值範圍為 0~15。

說明
如果你想讀取一個外界的數位訊號，用來讀取訊號的引腳要先設為輸入模式。 為了簡化引腳相關的輸入輸出命令，模式改變是由系統自動執行。使用者只需簡單的使用輸入輸出命令而無需擔心引腳的方向。然而，我們依然提供了 INPUT 命令。如果你使用 INPUT 命令來將引腳由輸出模式改為輸入模式，引腳上的 high/low 狀態會消失而引腳會被置於高電阻狀態。因此建議在引腳加上提拉電阻以避免不確定的邏輯狀態存在而造成邏輯上或電子上的危害。

範例
下面的例子改變了 P0 的 I/O 方向，它經由電阻連接到 LED，而 LED 會依此產生相對應的開/關。

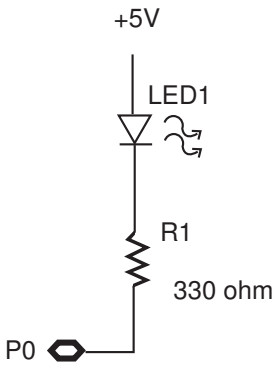


圖 6-7　輸出電路

```
Sub Main()
    Dim key As Byte

Start:
    WRITEPORT0 &H00            ' Write low to output buffers

    Do
        Debugin "Input any key to turn on LED.", %CHR key, CR
        OUTPUT 0          ' Switch P0 to OUTPUT mode, turn on LED0

        Debugin "Input any key to turn off LED.", %CHR key, CR
        INPUT 0           ' Switch P0 to INPUT mode, turn off LED0
    Loop
End Sub
```

INTEGER2FLOAT

語法

Result = INTEGER2FLOAT(*Argument*)

操作

整數值轉換成浮點格式。

- *Argument* – INTEGER2FLOAT 函數的整數運算元。
- *Result* – 用來接收 INTEGER2FLOAT 函數回傳值的浮點形態變數。

說明

INTEGER2FLOAT 命令將整數值轉換成浮點格式。浮點數的值是從-32768.0 到 +32767.0 的整數值。

範例

```
Sub Main()
    Dim MyInteger As Integer
    Dim MyFloat As Float

    MyInteger = -32768
    MyFloat = INTEGER2FLOAT(MyInteger)
    Debug "INTEGER2FLOAT of -32768 : ", MyFloat, CR

    MyInteger = 32767
    MyFloat = INTEGER2FLOAT(MyInteger)
    Debug "INTEGER2FLOAT of 32767 : ", MyFloat, CR
End Sub
```

LCASE

語法
LCASE(*TargetString*)

操作
將指定的字串內容轉換成小寫字元。

- ***TargetString*** - 要轉換的字串。

說明
LCASE 命令將指定的字串內容轉換成小寫字元。

範例

```
Sub Main()
    Dim MyString As String * 12

    MyString = "Hello World!"
    Debug "Original string: ", MyString, CR

    LCASE(MyString)
    Debug "All in lower case: ", MyString, CR

    UCASE(MyString)
    Debug "All in upper case: ", MyString, CR
End Sub
```

LCDCMD

語法

LCDCMD *Pin,Command*

操作

傳送命令給 LCD 模組。

- ***Pin*** – 常數或變數(1、9 或 17)，用來指定 LCD 模組所連接到的 7 個連續引腳的第 1 隻腳。舉例來說：1 代表引腳 1~8，依此類推。對一個 24 引腳的 BASIC Commander 而言，它的引腳值範圍為 1 或 9。
- ***Command*** – 常數或變數(0~255)，用來指定要下達的 LCD 命令。

說明

有三個 LCD 命令(LCDCMD、LCDIN 及 LCDOUT)允許 BASIC Commander 直接對使用 Hitachi 44780 或相容的 LCD 顯示器模組的控制器下達命令。有 1x16，2x16，4x20 字元型的 LCD 模組可使用。為了減少引腳數的需求，我們使用了 4 位元 LCD 介面。因此，總共只需要 7 個 I/O 引腳。為了使用者的 I/O 使用彈性，你可以指定第一個引腳值 1、9 或 17，也就是使用 P1~P7 或 P9~P15 或 P17~P23。下圖所顯示的是當 P1~P7 被使用時的線路圖。

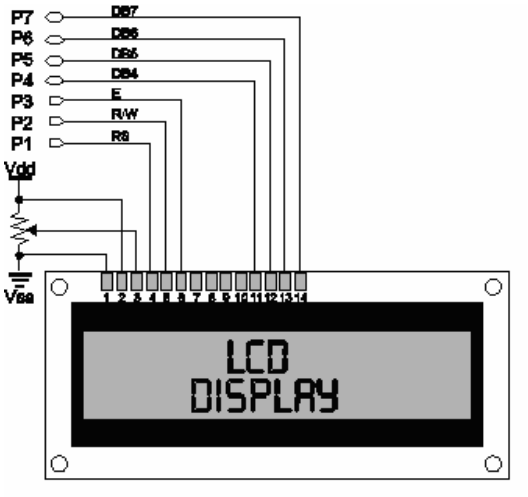


圖 6-8　LCD 模組接線圖（使用 P1~P7）

當 LCD 電源第一次開啟時，預設值為 8 位元的介面，我們在傳送命令前須要將它設定為 4 位元的匯流排。這個過程就是 LCD 的初始化，也是在你的程式開始時所要作的第一件事。請參閱範例 LCD 初始化程式碼。

Hitachi 44780 LCD 控制元件對顯示器初始化提供了一些特別的命令、移動游標、 改變預設的排列等等。請參照以下的表格。

命令碼（二進制）								說明
7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	無作用
0	0	0	0	0	0	0	1	清除顯示並將游標移至 HOME 位置
0	0	0	0	0	0	1	0	將游標移至 HOME 位置
0	0	0	0	0	1	M	S	遊標方向(M:0＝向左，１＝向右)與 捲動顯示(S:0＝不捲動，１＝捲動)
0	0	0	0	1	D	U	B	顯示(D:0＝關閉，1＝開啟) 顯示底線游標(U:0＝不顯示，１＝顯示)

								顯示閃爍方塊(B:0＝不顯示，１＝顯示)
0	0	0	1	C	M	0	0	移動顯示內容或游標(C: 0＝遊標，1＝顯示內容) 向左或向右移動(M: 0＝向左，1＝向右)
0	0	1	B	L	F	0	0	資料匯流排寬度(B: 0＝4 位元，１＝8 位元) 顯示器行數(L: 0＝１行，１＝2 行) 字型(F:0＝5x8、１＝5x10)
0	1	A	A	A	A	A	A	將資料指標移至字元產生器（CGRAM）由A所指定的位置
1	A	A	A	A	A	A	A	將資料指標移至資料顯示器（DDRAM）由A所指定的位置

下面圖中顯示的是一般的 DDRAM 對應表，你須要知道才有辦法對它來做更詳細的操作。

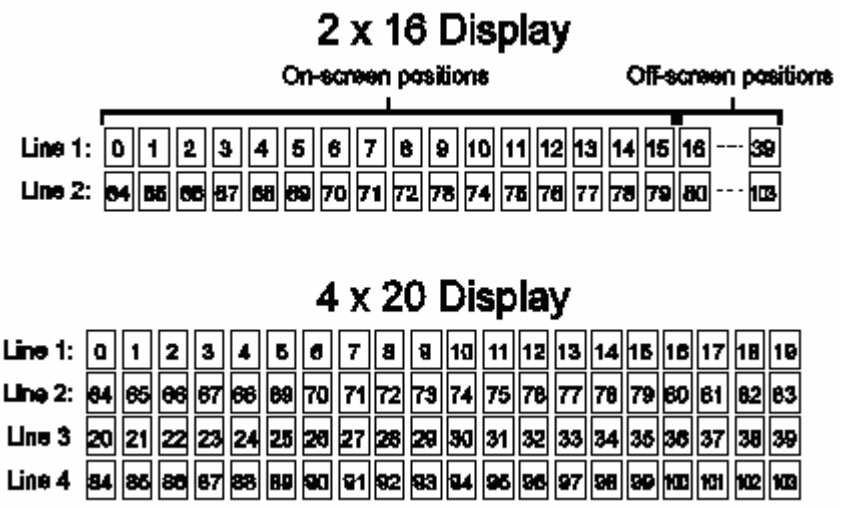


圖 6-9　DDRAM 地址分配圖

範例

```
Sub Main()
```

```
    Dim I, DataIn As Byte
    Dim MyString(8) As Byte = "Hi There!"

    PAUSE 1000                ' allow LCD to self-initialize first
    LCDCMD 1, 32              ' set data bus to 4-bit mode
    LCDCMD 1, 40              ' set to 2-line mode with 5x8 font
    LCDCMD 1, 15              ' display on with blinking cursor
    LCDCMD 1, 6               ' auto-increment cursor
    LCDCMD 1, 1               ' clear display

    For I=0 To 8
        LCDOUT 1, &H80+I, [MyString(I)]     ' display “Hi There!”
    Next

    Debug "Read from LCD : "

    For I=0 To 8
        LCDIN 1,&H80+I, [DataIn]       ' &H80, DDRAM address of line 1
    Debug %CHR DataIn
    Next

End Sub
```

LCDIN

語法
LCDIN *Pin*,*Command*, [*DataIn*]

操作
從 LCD 模組接收資料。

- ***Pin*** – 常數或變數(1，9 或 17)，用來指定 LCD 模組所連接到的 7 個連續引腳的第 1 隻腳。舉例來說：1 代表引腳 1~8，依此類推。對一個 24 引腳的 BASIC Commander 而言，它的引腳值範圍為 1 或 9。
- ***Command*** – 常數或變數(0~255)，用來指定要下達的 LCD 命令。
- ***DataIn*** – 變數串列，用來接收 LCDIN 命令讀入的資料。

說明
有三個 LCD 命令(LCDCMD、LCDIN 及 LCDOUT)允許 BASIC Commander 直接對使用 Hitachi 44780 或相容的 LCD 顯示器模組的控制器下達命令。LCDIN 命令是用來送出一個命令而後從 LCD 的字元產生隨機存取記憶體或顯示器中接收至少一個資料位元組。資料隨機存取記憶體相關的介面及線路詳細資料請參照 LCDCMD 命令。請注意所有的 I/O 引腳一開始都會被設為輸出模式。當 LCDIN 命令要結束的時候，較高的 4 個 I/O 引腳(4~7 或 12~15 或 20~23)會被設為輸入模式。

當LCD電源第一次開啟時，預設值為8位元的介面，我們在傳送命令前須要將它設定為4位元的匯流排。這個過程就是LCD的初始化，也是在你的程式開始時所要作的第一件事。更詳細的LCD初始化資訊請參照LCDCMD命令。

LCDIN命令是用來接收ASCII 字元值，十進位，十六進位以及二進位的轉換以及從LCD的字元產生記憶體或顯示器的資料記憶體中所得到的字串資料。

範例
命令使用方法請參閱 LCDCMD 命令的範例。

LCDOUT

語法
LCDOUT *Pin*,*Command*, [*DataOut*]

操作
傳送資料到 LCD 模組。

- ***Pin*** – 常數或變數(1、9 或 17)，用來指定 LCD 模組所連接到的 7 個連續引腳的第 1 隻腳。舉例來說：1 代表引腳 1~8，依此類推。對一個 24 引腳的 BASIC Commander 而言，它的引腳值範圍為 1 或 9。
- ***Command*** – 常數或變數(0~255)，用來指定要下達的 LCD 命令。
- ***DataOut*** – 常數或變數資料串列，為要顯示的資料。

說明
有三個 LCD 命令(LCDCMD、LCDIN 及 LCDOUT)允許 BASIC Commander 直接對使用 Hitachi 44780 或相容的 LCD 顯示器模組的控制器下達命令。LCDIN 命令是用來送出一個命令而後從 LCD 的字元產生隨機存取記憶體或顯示器中接收至少一個資料位元組。資料隨機存取記憶體相關的介面及線路詳細資料請參照 LCDCMD 命令。請注意所有的 I/O 引腳一開始都會被設為輸出模式。當 LCDIN 命令要結束的時候，較高的 4 個 I/O 引腳(4~7 或 12~15 或 20~23)會被設為輸入模式。

當LCD電源第一次開啟時，預設值為 8位元的介面，我們在傳送命令前須要將它設定為4位元的匯流排。這個過程就是LCD的初始化，也是在你的程式開始時所要作的第一件事。更詳細的LCD初始化資訊請參照LCDCMD命令。

LCDOUT命令是用來傳送一個命令後面跟隨著至少一個資料位元組到LCD。輸出的資料會寫入LCD的字元產生記憶體或顯示器的資料記憶體。

範例
命令使用方法請參閱 LCDCMD 命令的範例。

LEFT

語法

LEFT(*TargetString, length*)

操作

保留 *TargetSring* 中從最左邊數起的指定長度的字元，其他字元則截取掉。

- *TargetString* - LEFT 函式的字串運算子。
- Length - 常數或變數，用來指定字串要保留的長度。

說明

LEFT 命令保留了字串中最左邊被指定的長度的字元而將剩餘的截取掉。如果所定的長度比原本目標字串的長度還長時，超過的長度會被忽略而字串保持不變。

範例

```
Sub Main()
    Dim MyString As String * 12

    MyString = "Hello World!"
    LEFT(MyString, 5)
    Debug "Leftmost 5 letters : ", MyString, CR

    MyString = "Hello World!"
    RIGHT(MyString, 5)
    Debug "Rightmost 5 letters : ", MyString, CR

    MyString = "Hello World!"
    MID(MyString, 3, 5)
```

```
    Debug "Middle 5 letters from the third letter : ", MyString, CR

End Sub
```

LEN

語法
Length = LEN(*StringVar*)

操作
回傳字串長度。

- ***StringVar*** - LEN 函式的字串運算子。
- ***Length*** - 變數用來承接所指定字串的長度。

說明
LEN 命令回傳一個字串的長度。對於空字串，意指不含 ASCII 字元，它的長度為 0。最大長度值為字串宣告的大小。

範例

```
Sub Main()
    Dim MyString As String * 12
    Dim Length As Byte

    MyString = ""
    Length = LEN(MyString)
    Debug "Length of a null string is ", Length, CR
    MyString = "Hello!"
    Length = LEN(MyString)
    Debug "Length of MyString is ", Length, CR
End Sub
```

LOG

語法
Result = LOG(*Argument*)

操作
回傳浮點參數的對數值。

- ***Argument*** - LOG 函式的浮點運算子。
- ***Result*** - 浮點變數用來接收 LOG 函數回傳值。

說明
LOG 函數回傳一個浮點參數的對數值。在數學上表示為 y = ln(x)。相反的函式為 EXP 指數函式。

範例

```
Sub Main()
    Dim Result As Float

    Result = LOG(2.7183)
    Debug "LOG of 2.7183 is ", Result, CR
    Result = LOG(7.3891)
    Debug "LOG of 7.3891 is ", Result,CR
End Sub
```

LOG10

語法

Result = LOG10 (*Argument*)

操作

回傳浮點參數以 10 為基底的對數值。

- ***Argument*** - LOG10 函數的浮點運算子。
- ***Result*** - 浮點變數用來承接 LOG10 函式回傳值。

說明

LOG10 函式回傳浮點參數以 10 為基底的對數值。***Argument*** 值恆為正。在數學上，表示為 y = log(x)。相反的函式為以 10 為基底的 EXP10 指數函式。

範例

```
Sub Main()
    Dim Result As Float

    Result = LOG10(10.0)
    Debug "LOG10 of 10.0 is ", Result, CR
    Result = LOG10(100.0)
    Debug "LOG10 of 100.0 is ", Result, CR
End Sub
```

LONG2FLOAT

語法

Result = LONG2FLOAT(*Argument*)

操作

將 Long 型態數值轉換為浮點型態數值。

- ***Argument*** - LONG2FLOAT 函式的 Long 運算元。
- ***Result*** - 浮點變數用來承接 LONG2FLOAT 函數回傳值。

說明

LONG2FLOAT 命令將 Long 型態數值轉換為浮點型態數值。浮點型態數值為從 +2147483647 到 -2147483648。由於使用的是單精確度浮點數，所以 LONG 變數可能無法很精確的表示出來。回傳的是最接近的浮點整數值。當你在應用程式中使用這個命令時需要很小心。你可以利用二進位形式來檢視 LONG 值是否被精確的表示。去除二進位前端及後端的 0，如果剩餘的位元數大於 24，則該數值無法精確表示。

範例

由於使用的是單精確度浮點數，數值 2147483647 無法精確的顯示。回傳的值會是最接近的 2147483648。

```
Sub Main()
    Dim MyLong As Long
    Dim MyFloat As Float

    MyLong = -2147483648
    MyFloat = LONG2FLOAT(MyLong)        ' the result is -2147483000
    Debug "LONG2FLOAT of -2147483648 : ", MyFloat, CR
```

```
    MyLong = 2147483647
    MyFloat = LONG2FLOAT(MyLong)        ' the result is 2147484000
    Debug "LONG2FLOAT of 2147483647 : ", MyFloat, CR
End Sub
```

LOW

語法
LOW *Pin*

操作
將指定引腳設為邏輯低準位。

- *Pin* - 常數或變數(0~23)，用來指定邏輯低準位所要實施的引腳。對一個 24 引腳的 BASIC Commander 而言，它的引腳值範圍為 0~15。

說明
這個命令會將指定的引腳設為接近 0 伏特的低準位。一般來說，在執行對應的輸入輸出操作前引腳要預先改變成輸入或輸出模式。然而在 LOW 命令運作時只有一個引腳會參與其中，所以命令會自動將這引腳改變為輸出模式。使用者無須事先執行 OUTPUT 命令改變引腳成為輸出模式。其他詳細功能介紹，請參閱 HIGH 命令。

範例
命令使用方法請參閱 HIGH 命令的範例。

MID

語法

MID(*TargetString, Start, Length*)

操作

保留指定字串 ***TargetSring*** 從中間 ***Start*** 位置開始長度為 ***Length*** 的所有字元，剩餘的則截取掉。

- ***TargetString*** – MID 函數的字串運算元。
- ***Start*** – 常數或變數，用來定義字串的起始位置。
- ***Length*** – 常數或變數，用來定義要被複製的字串的長度。

說明

MID 命令保留了指定字串從中間 ***Start*** 位置開始長度為 ***Length*** 的所有字元。如果 ***Start*** 比目標字串的字元數目還要大，則字串為空字串。如果指定的長度比目標字串從 ***Start*** 位置算起的剩餘長度還要長，超出的部份將會被忽略。

範例

命令使用方法請參閱 LEFT 命令的範例。

OUTPUT

語法
OUTPUT *Pin*

操作
將指定的 *Pin* 設為輸出模式。

- ***Pin*** – 常數或變數(0~23)，用來指定要被設定成輸出模式的引腳。對一個 24 引腳的 BASIC Commander 而言，它的引腳值範圍為 0~15。

說明
如果你想傳送一個數位訊號到外界，用來輸出訊號的引腳要首先設為輸出模式。 為了簡化引腳相關的輸入輸出命令，模式改變是由系統自動執行。使用者只需簡單的使用輸入輸出命令而無需過於擔心引腳的方向。然而，我們依然提供了 OUTPUT 命令。OUTPUT 命令會將引腳由輸入模式改為輸出模式，之前最後輸出的 High/Low 準位會在轉態時被輸出，使用時請注意是否有電位上的衝突。

範例
命令使用方法請參閱 INPUT 命令的範例。

PAUSE

語法
PAUSE *Duration*

操作
命令強制程式等待指定的時間。

- *Duration* – 常數或變數，用來定義暫停動作的持續時間。持續時間單位為 1 毫秒。

說明
這個命令會在前一個段落及後一個段落間放入延遲。PAUSE 命令允許放在程式的任何地方，讓使用者對程式執行速度有些許的控制力。持續時間單位為 1 毫秒。

範例
下列程式示範如何使用不同的暫停時間讓 LED 閃爍。

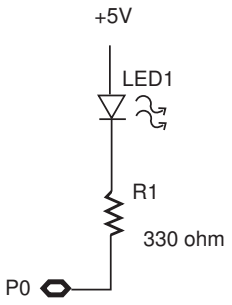


圖 6-10　LED 線路

```
Sub Main()
    Dim Period As Word
```

```
START:
    Debugin "Set pause (0~65535 ms) between LED ON/OFF.", Period, CR

    Do
        HIGH 0
        Pause(Period)
        LOW 0
        Pause(Period)
    Loop
End Sub
```

PERIPHERAL

語法
PERIPHERAL *Name* AS *TypeName* @ *ID*

操作
宣告週邊模組。

- ***Name*** – 使用者所宣告的名字，類似於變數名稱。
- ***TypeNname*** – 週邊模組形態的一種，概念上類似資料形態。
- ***ID*** – 一個常數(0~31)，用來指示週邊模組上的指撥開關所設定的週邊 ID。

說明
這個指令是用來宣告週邊設備模組，給予一個獨特的模組名稱和一個 *TypeName* 去定義模組的種類。這個 ID 是一個唯一的位址，所有連接到 cmdBUS(TM)的模組都可以被獨立定址。所有利基科技新發展出來的模組都會被指定一個新的型態名稱，*TypeNname* 必須是這些型態名稱之一。週邊設備模組要在所有程序之外作宣告。換句話說，週邊模組要像全域變數一樣運作，在整個程式中都可以被辨認。

跟其他指令不同，週邊設備指令是跟週邊設備硬體聯繫在一起的。伴隨著模組硬體的文件會補充模組型態的名稱以及相關的函數。請注意，只要被指定不同的 ID，必要時 cmdBUS(TM)可以同時連接一個以上的相同模組。

範例
下列程式示範如何去宣告及將命令下達到週邊設備模組。

```
eripheral myLCD As LCD2x16a @ 0     ' a 2x16 LCD module with ID 0


Sub Main()
    myLCD.Display(“Hi There!”)
End Sub
```

PULSEIN

語法

PULSEIN *Pin*, *State*, *Variable*

操作

這個指令是用來測量在指定的引腳上出現的脈衝波的脈衝頻寬。

- ***Pin*** – 常數或變數值(0-23)，用來指定要測量脈衝頻寬的引腳。對一個 24 引腳 BASIC Commander，這個引腳值的範圍為 0~15。
- ***State*** – 常數或變數值(0~1)，用來定義正波或負波。如果寫 0，則表示測量負波，如果寫 1，則表示測量正波。
- ***Variable*** – 一個 WORD 型態的變數用來存放所測量到的脈衝頻寬。測量的單位為 5　s。如果測量到的脈衝波單位數超過最大值 65535 或是沒有脈衝波出現，回傳值為 0，表示是一個無效的測量。

說明

這個測量指令是用來測量特定引腳上所表示的脈衝的頻寬。這個指令在脈衝頻寬被外接硬體使用來表示外部測量時很有用。可能的例子像是一些外接 IC 用來測量溫度，壓力等等，這些輸出是以脈衝頻寬表示而非伏特電壓。這 IC 可以直接對應到用來測量這些 IC 的 BASIC Commander 和 PULSEIN 指令。待測的脈波可以是高脈波或低脈波。請務必注意，當執行指令接觸到第一個脈衝邊緣時它會開始測量脈衝頻寬，*State* 變數設 1 時將會是一個高脈衝的上升邊緣，*State* 變數設為 0 時將會是一個低脈衝的下降邊緣。測量的單位為 5　s。如果測量到的脈衝波單位數超過最大值 65535 或是沒有脈衝波出現， 回傳值為 0。如果脈衝沒出現，數值 0 會被載入 ***Variable***，程式則會繼續執行下一個指令。這可以防止當沒有脈衝出現時程式不會停在這個指令。當 PULSEIN 指令完成測量時，不論成功否，程式都會繼續執行下一個指令。

範例

下面程式示範以 PULSEIN 指令去量測外部的低準位電壓寬度。

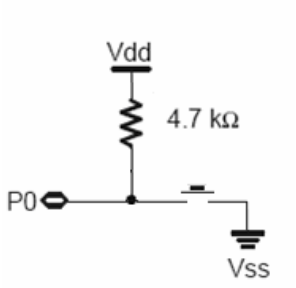


圖 6-11　PULSEIN 量測線路

```
Sub Main()
     Dim Result As Word

    Do
      Debug "Press the key.", CR
wait:
      Pulsein 0,0,Result          ' measure low pulse on P0
      If Result=0 Then Goto wait   ' not pressed or longer than 327 ms

      Result\=200                      ' convert 5us unit to 1ms unit
      Debug "The key pressed for ", Result," ms.",CR
    Loop

End Sub
```

PULSEOUT

語法

PULESOUT *Pin, Duration*

操作

測量在引腳上出現的脈衝波的脈衝頻寬。

- ***Pin*** - 常數或變數值(0-23)，用來指定脈衝頻寬產生的引腳。對一個 24 引腳 BASIC Commander，這個引腳值的範圍為 0~15。
- ***Duration*** - 常數或變數值(0~65535)用來定義脈衝頻寬的長度。這個單位為 5us。

說明

PULSEOUT 指令在引腳上產生一個使用者指定頻寬的脈衝。這個型態的脈衝頻寬，無論高或低，取決於脈衝產生引腳上的初始值。如果 PULSEOUT 指令執行時引腳在低的狀態，則會產生高脈衝。如果引腳初始是高的狀態，則會產生低脈衝。換言之，引腳在脈衝產生時間會被反向。

範例

以下程式是以 PULSEOUT 指令去驅動伺服馬達的例子。通常伺服馬達的控制是在一個 20ms 週期內，給予 0.5ms~2.5ms 的高準位訊號。這個例子設定了 1ms 與 2ms 二個位置，伺服馬達將以 4 秒為週期來回轉動。請將電源接至馬達電源線並將引腳 P0 接至馬達訊號線。

```
Sub main()

    Dim b As Byte

    low 0             ' initialize pin 0 to low to have high pulse

    Do
```

```
    For b=0 To 100         ' 2 seconds servo rotation
             PULSEOUT 0,200    ' move to position of 1ms pulse width
             Pause 19        ' constitute a 20 ms cycle
       Next

       For b=0 To 100     ' 2 seconds servo rotation
             PULSEOUT 0,400  ' move to position of 2ms pulse width
             Pause 18        ' constitute a 20 ms cycle
       Next
    Loop

End Sub
```

PWM

語法

PWM *Pin, Duty, Cycles*

操作

以 PWM 訊號輸出模擬類比訊號。

- ***Pin*** – 常數或變數值(0-23)，用來指定產生 PWM 的引腳。初始時這個引腳會被設成輸出模式，當指令完成時會設為輸入模式。對一個 24 引腳 BASIC commander，這個引腳值的範圍為 0~15。
- ***Duty*** – 常數或變數值 (0~255)，用來定義輸出波形的工作週期(佔空比)。
- ***Cycles*** – 定義 PWM 輸出產生所需的週期數(大約每個循環 1.15 ms)， 實際上也就是定義 PWM 輸出運作的時間。*Cycles* 為 0 將不會產生 PWM 輸出。它的值可以是常數，變數或是一個表示式而且必須在範圍 0-255 之間。

說明

PWM 指令允許 BASIC Commander 在它的數位引腳上產生一個類似的伏特電壓輸出。當你將輸出引腳設為 high 時，引腳的伏特電壓會接近 5 伏特，如果將輸出引腳設為 low，引腳的伏特電壓會接近 0 伏特。 如果你將引腳值在 high low 間迅速地變換，那你將會得到一個在這之間的伏特電壓。你所得到的實際伏特電壓值是取決於由 high 到 low 的時間比例，稱之為功率週期。舉例來說，如果功率為 150，(150/255) *5V = 2.94V。PWM 指令輸出一串列的脈衝，平均伏特電壓值為 2.94V。功率週期是 1.15ms，如下圖。

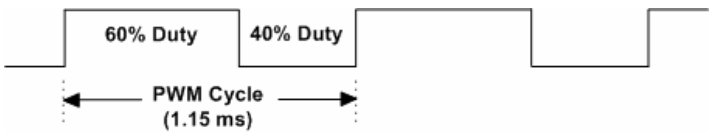


圖 6-12　工作週期為60%的PWM波形

下列低通路電阻/電容電路系統在指令完成之後會濾出脈衝和保持類似物電壓。類似物電壓的保留取決於外接電路從那裡取走多少電流，包括電容電流流失。為了保留電壓，週期性的執行 PWM 指令去幫電阻/電容電路充電是必要的。

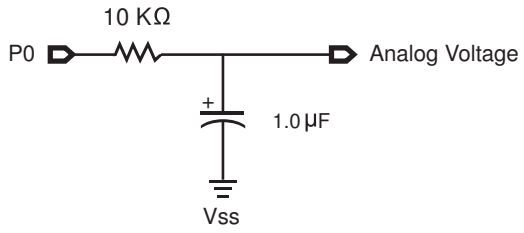


圖 6-13　PWM低通輸出電路

一開始要將電容充到希望的電壓需要時間。你可以使用近似值的規則：充電時間 = 5*R*C，去估算充電最少所需的週期數。如下圖描述：

充電時間 = 5 * 10 x 10^{3} * 1 x 10^{-6} = 50 x 10^{-3} 秒，或 50 ms。

對於BASIC Commander，每個週期約1.15 ms，因此電容充電至少需要44個週期。 假設使用引腳 0，則命令會像：

```
PWM 0, 150, 44          ' charge to 2.94 V, on Pin 0
```

範例

```
Sub Main()
    Dim f As Float
    Dim duty As Byte
    Dim b As Byte
```

```
start:
    Debugin "Enter desired voltage(0~5V) : ",f,CR
    If f<0 Or f>5 Then
        Debug "Invalid value.",CR
        Goto start
    Else
        f=f*255/5
        duty=float2byte(f)
        Debug "duty=",duty,CR
        For b=1 To 100        ' Hold the voltage for 5 seconds
        PWM(0,duty,44)
        Next b
    End If

    Goto start
End Sub
```

RANDOM

語法
RANDOM *Variable*

操作
產生隨機數。

- *Variable* - 產生的隨機數會置於之前所定義的變數。

說明
這個指令會產生虛擬隨機數然後放置於定義的變數中。虛擬隨機數並非是完全的隨機而是以一個序列的形式在一段時間後本身會重覆。然而這個方式所產生的隨機數在大多數應用目的上都是合適的。要產生一個真正的隨機數一些外部控制的事件必須在數目產生中被考慮到。隨機數序列也需倚賴所選擇的起始數去開始產生。這些可以藉由為變數設定初始值加以控制。這個值將會被視為隨機數產生的基礎。這就是我們所知的隨機數產生種子值。

範例

下列程式顯示如何去組合一個外接按鈕和 RANDOM 指令去產生隨機數。

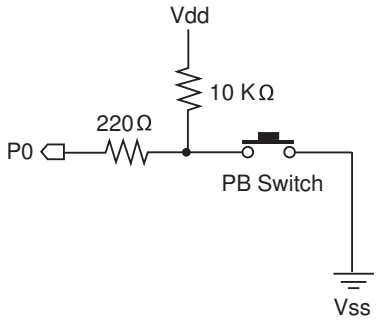


圖 6-14　配合Random命令產生亂數的線路

```
Sub Main()
    Dim Key As Byte
    Dim Seed As Dword =10
    Dim X,Y As Byte
    Dim Temp As Dword
    Dim NumArray(5) As Dword
    Do
        Debug "Press push button to get your lucky numbers.", CR

Wait:
        RANDOM(Seed)
        If IN(0)<>0 Then Goto Wait
        Debug "The lucky numbers are "

        For Y=0 To 5
GenRandom:
            RANDOM(Seed)
            Temp = Seed Mod 47
            Temp += 1

            For X = 0 To 5
                If Temp = NumArray(X) And X <> Y Then
                    Goto GenRandom
                End If
            Next

            NumArray(Y) = Temp
```

```
            Debug NumArray(Y), " "
        Next
        Debug CR
    Loop
End Sub
```

RCTIME

語法

RCTIME *Pin*, *State*, *Variable*

操作

測量電阻/電容(RC)電路充電放電的時間。

- ***Pin*** - 常數或變數值(0-23)，用來指定要使用的 I/O 引腳。這個引腳會被設成輸入模式。對一個 24 引腳 BASIC commander，這個引腳值的範圍為 0~15。
- ***State*** - 常數或變數值(0 或 1)，用來定義想要測量到的狀態。一但 *Pin* 不存在 *State* 中，命令結束並將結果儲存在 *Variable* 中。
- ***Variable*** - 一個變數，通常是 word，時間測量被儲存的地方。變數的時間單位為 5us。

說明

這將允許你去測量電阻或電容外接的R或C型態感應器的值。像是電熱器，電位計或電容溼度感應器。RCTIME可以用來測量電阻/電容電路充電放電的時間。

當 RCTIME 執行時，會將 *Pin* 設為輸入，然後開始一個計數。當指定的引腳不再是 *State* (0 或 1)，計數會立刻停止。當命令執行時，如果引腳不是在狀態時， RCTIME 會回傳 1 到 *Variable*。如果引腳維持在 *State* 超過 65535 時間週期，回傳值將為 0。

下面兩個圖顯示RCTIME命令所需的電路。在RCTIME停止計時前你可以任選一個大約0.7VDD伏特電壓幅度的圖。

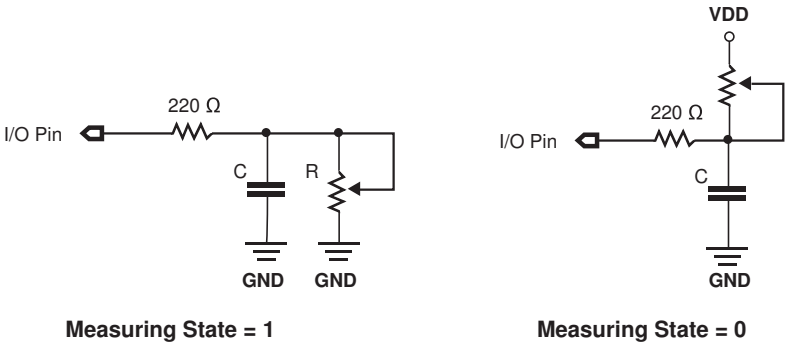


圖 6-15　RCTIME命令搜需配合的線路

一旦計時器停止，計數器值會存入變數中。我們要如何取得 R 或 C 的值呢？
RC 時間常數，或短促的 tau（τ），R 和 C 的乘法運算，表示著給定的 RC 電路要充電或放電初始電壓的 63%所需要的時間。

一般的 RC 充電放電方程式：

time = $-\tau$ * (ln (V_{final} / $V_{initial}$))

比如我們用一個10 k歐姆的電阻和 0.1 μF電容。 因為狀態 1或 0， V_{final} / $V_{initial}$都固定為 0.4，我們可以估計時間：

time=$9.163*10^{-4}$

計時器的單位為5us，大約計數183次。

因此，我們可以用近似規則來計算問題中的R或C值。

RCTIME counts = 183 x R (in kohm) x C (in uF)

在 RCTIME 執行前，電容必須被充電至 5V 成狀態 1 或放電至 0V 成狀態 0。你可以用近似的規則：

Charge time = 5 * τ

去預估充電至少所需的時間。例如：假設電容為 0.1uF 電阻為 200 歐姆，如同圖中所描述的，充電時間至少為：

充電時間 = 5 * 220 * 0.1 x 10^{-6} = 0.11 ms.

範例

下列程式示範如何使用 RCTIME 指令去顯示經由引腳 0 所連接到上面的 RC 電路的電熱器的電阻值。

```
Sub Main()
    Dim result As Word
    Dim resultf As Float
    Do
        High 15
        Pause 1
        RCTIME 0,1,result
        resultf=Word2float(result)
        resultf=resultf/183*10
        Debug "The potentiometer value (Kohm) : ", resultf, CR
    Loop
End Sub
```

READPORT

語法

Result = READPORT ***Port***

如果連接埠號碼是一個常數， 你也可以使用下列的形式。

Result = READPORT0
Result = READPORT1
Result = READPORT2

操作

讀取指定的 I/O 埠。

- ***Port*** – 常數或變數值(0 ~2)，用來定義連接埠的號碼。連接埠 0 包含引腳 P0~P7，連接埠 1 包含引腳 P8~P15，以及連接埠 2 包含引腳 P16~P23。對一個 24 引腳的 BASIC Commander 而言，連接埠的值為 0 或 1。
- ***Result*** – 一個位元組變數，用來接收連接埠上的外部邏輯值。

說明

READPORT 指令是用來讀取外界的數位訊號。不同於 IN 指令，使用本指令時，引腳的 I/O 模式不會自動被改變為輸入模式。對於 READPORT 指令，總共有 8 個引腳參與其中，每一個引腳都可以設為輸入或輸出模式。所以你必須要明確的設定每一個位元。索引值比較小的引腳是存比較低位元順序的資料。

請注意，當 I/O 引腳被設定為輸入模式時，建議每一個引腳都連接到一個 10K 歐姆的提拉電阻。否則由於這些引腳的高阻抗浮接狀態，I/O 引腳可能會被隨機的讀成 1 或 0 而無法反應出引腳上真正的數位訊號。

對於設為輸出模式的引腳部分，從這些引腳無法讀取真正的 I/O 引腳狀態，僅能讀出之前寫入輸出引腳的資料。

範例

所有 8 個連接埠 0 的 I/O 引腳都連接了 10 K 歐姆的提拉電阻並且 8 個按鈕連接到地線。如果任何引腳被按壓，你可以用這個程式來偵測。

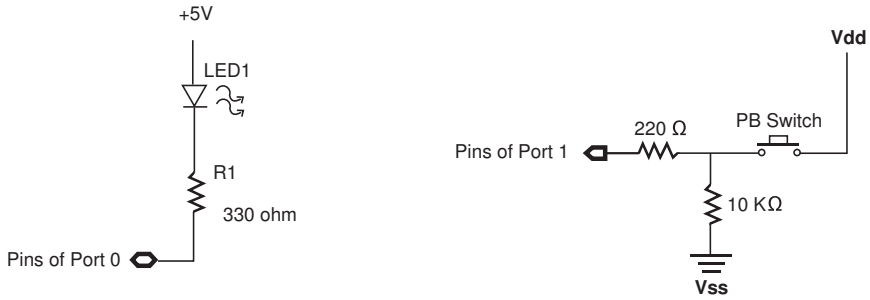


圖 6-16　Port 1 作為輸入引腳， Port 0作為輸出引腳

```
Sub Main()
    Dim IOStatus as Byte

    SETDIRPORT1(&B11111111)    ‘ set Port 1 as input
    SETDIRPORT0(&B00000000)    ‘ set Port 0 as output
    Do
        IOStatus=READPORT1()
        WRITEPORT0(IOStatus)
    Loop
End Sub
```

RETURN

語法

RETURN {*ReturnValue*}

操作

從子程式，函數或事件中返回。如果從函數中返回，則必須提供一個 *ReturunValue*。

- ***ReturnValue*** - 額外的回傳值，只會出現在函數程式中。

說明

ReturnValue 只有在函數中使用來將函數執行的結果回傳。當 END SUB，END FUNCTION 或 END EVENT 出現時，子程式，函數或事件程序就會結束。但是如果 RETURN 指令在 END SUB，END FUNCTION 或 END EVENT 之前出現，則會立即結束子程式，函數或事件程序。如在任何子程式，函數或事件發生前就執行到 RETURN 命令，將會跳轉至主程式末尾將主程式結束。

範例

```
Function Max(Z1 As Integer, Z2 As Integer) As Integer
    If Z1>Z2 Then
        Return Z1
    Else
        Return Z2
    End If
End Function

Sub Main()
    Dim X, Y, Z As Integer
    Debugin "Enter X = ",X
```

```
    Debug X,CR
    Debugin "Enter Y = ",Y
    Debug Y,CR
    Z = Max(X,Y)
    Debug "Max(X,Y)=", Z, CR
End Sub
```

REVERSE

語法
REVERSE *Pin*

操作
將指定的引腳方向做反轉，不論在輸入或輸出模式。

- ***Pin*** – 常數或變數值(0 ~23)，用來指定要被設為輸入和輸出相反模式的引腳。對一個 24 引腳的 BASIC Commander 而言，引腳的值為 0~15。

說明
除了 INPUT 和 OUTPUT 指令外，也提供了 REVERSE 指令，可以將現在的輸入或輸出模式作反向。如果要讀或寫數位訊號到外界，引腳必須在之前就改變成對應的模式。由於要簡化引腳相關的輸入輸出指令，模式改變大部分是由系統自動執行的，使用者只須使用輸入輸出指令而不需要煩惱引腳的方向。使用應用電路時必須注意，要避免因錯誤的路徑造成無法預期的輸入輸出方向所造成的危害。

範例

下列範例改變了引腳 P0 經由電阻連接到 LED 電容的 I/O 方向。LED 會依此做相對的開關動作。

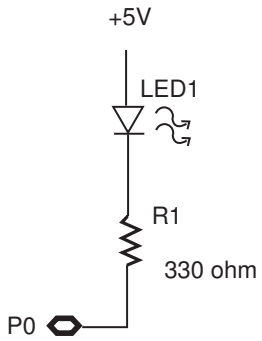


圖 6-17 LED 線路

```
Sub Main()

    Dim key As Byte

Start:
    WRITEPORT0 & H00  ' Write low to output buffers

    Do
    Debugin "Input any key to toggle LED On/Off.", %CHR key, CR
    REVERSE 0         ' Switch P0 to OUTPUT mode, turn On LED0
    Loop

End Sub
```

RIGHT

語法
RIGHT(*TargetString, length*)

操作
保留指定字串 ***TargetSring*** 中，最右邊總長為 ***length*** 的字母，多餘的則截取掉。

- ***TargetString*** – RIGHT 函數的字串運算元。
- ***Length*** – 用來定義要保留的字串長度的變數。

說明
RIGHT 指令保留了字串中，最右邊總長為 *length* 的字母，其餘的則截取掉。
如果定義的長度比目標字串還要長，多出的長度將會被忽略而字串維持不變。

範例
指令使用方法請參閱 LEFT 指令的範例。

SELECT… CASE

語法

SELECT {CASE} *Expression*
{ CASE *Const*
Statement(s)... }
{ CASE ELSE
Statement(s) }
END SELECT

操作

SELECT 指令根據表示式的值來執行指令。

- *Expression* – 變數，常數，或表示式。
- *Const* – 用來跟 *Expression* 做比較的常數，如果相同則 CASE 中的指令會被執行。
- *Statement(s)* – 任何有效的 innoBASIC 陳述。

說明

SELECT…CASE 可說是 IF…THEN…ELSE 的進階架構，它可以在許多可能中擇一執行。執行此敘述時，Select 的值將與 Case 的值逐一比對，當某一個 Case 的值符合時，該 Case 之下的程式段將被執行。如果沒有任何一個 Case 的值符合，則 Case Else 之下的程式段將被執行。任何被選擇執行的程式段執行完畢後，程式就結束到 End Select 指令。EXIT SELECT 指令可以放置於迴圈主體中，它可以在迴圈限制測試執行前就立即離開目前的迴圈。請注意，習慣上額外的 CASE 可以伴隨著額外的 SELECT。

範例

更進一步的詳細範例請參照第五章。

SETDIRPORT

語法

SETDIRPORT *Port, Dir*

如果連接埠號碼是一個常數，你可以用下列任一種格式型態。

SETDIRPORT0 Dir
SETDIRPORT1 Dir
SETDIRPORT2 Dir

操作

設定指定的連接埠的 I/O 方向設定。

- *Port* – 常數或變數值(0-2)，用來指定連接埠號碼。連接埠 0 包含引腳 P0-P7，連接埠 1 包含引腳 P8-P15，連接埠 2 包含引腳 P16-P23。以一個 24 引腳的 BASIC Commander 來說，連接埠的值是 0 或 1。
- *Dir* – 用來指定 I/O 方向的位元組。資料位元組中的每一個位元都定義著一個引腳的方向，0 是輸出，1 是輸入。
- *Data* – 用來定義資料要寫到特定連接埠的常數或變數值。

說明

如果你想用 READPORT 指令從外界讀取一個數位訊號，相對應的引腳要首先設為輸入模式。用 SETDIRPORT 指令來設定指定連接埠的 I/O 方向設定。每一個連接埠引腳都可以獨立設定。資料 0 表示輸出，資料 1 表示輸入模式。索引值較小的引腳將會成為資料位元組中順序較低的位元。例如，P0 的設定會出現在資料讀取的位元 0。I/O 方向在程式開始後預設為輸入。

當 I/O 引腳被設定為輸入模式時，建議每一個引腳都外接一個 10K 歐姆的提拉電阻。否則由於這些引腳的高阻抗浮接狀態，I/O 引腳可能會被隨機的讀成 1 或 0 而無法反應出引腳上真正的數位訊號。

範例

指令使用方法請參閱 GETDIRPORT 指令的範例。

SGN

語法

Result = SGN(*Argument*)

操作

回傳浮點數的正負符號值。

- ***Argument*** – SGN 函數的浮點運算元。
- ***Result*** – 接受 SGN 函數回傳值的 SHORT 變數。

說明

SGN 指令回傳符點數的正負符號值。浮點數為正數，則回傳值為 1。浮點數為負數，則回傳值為 -1。浮點數為 0，則回傳值為 0。

範例

```
Sub Main()
    Dim Result As Short

    Result = SGN(-0.1)
    Debug "SGN(-0.1) = ", Result, CR
    Result = SGN(0)
    Debug "SGN(0) = ", Result, CR
    Result = SGN(0.1)
    Debug "SGN(0.1) = ", Result, CR
End Sub
```

SHORT2FLOAT

語法
Result = SHORT2FLOAT(*Argument*)

操作
將 SHORT 值轉換成浮點格式。

- ***Argument*** - SHORT2FLOAT 函數的 SHORT 運算元。
- ***Result*** - 用來接收 BYTE2FLOAT 函數結果的浮點變數。

說明
SHORT2FLOAT 指令將一個 SHORT 值轉換成浮點格式。浮點數的值會是一個範圍在 -128.0 to +127.0 的整數。

範例

```
Sub Main()
    Dim MyShort As Short
    Dim MyFloat As Float

    MyShort = -128
    MyFloat = SHORT2FLOAT(MyShort) ' MyFloat has the value -128.0
    Debug "SHORT2FLOAT of -128 : ", MyFloat, CR
    MyShort = 127
    MyFloat = SHORT2FLOAT(MyShort)      ' MyFloat has the value 127.0
    Debug "SHORT2FLOAT of 127 : ", MyFloat, CR
End Sub
```

SIN

語法
Result = SIN(*Argument*)

操作
執行 sine 數學函數。

- ***Argument*** - sine 函數的浮點運算元範圍為 0 to 2π。
- ***Result*** - 用來接收 sine 函數值的浮點變數。

說明
SIN 函數回傳一個範圍為 0 到 2π 的浮點參數的 sine 值。這個參數的單位是弧度。如果轉換為度數，記得 360 度等同於弧度 2π。

範例

```
Sub Main()
    Dim MyFloat As Float
    Dim Result As Float

    MyFloat = pi/4
    Result = SIN(MyFloat)          ' the Result is 0.707107
    Debug "SIN(pi/4)=", Result, CR
End Sub
```

SQRT

語法

Result = SQRT(*Argument*)

操作

回傳浮點參數的平方根值。

- ***Argument*** - SQRT 函數的浮點運算元。
- ***Result*** - 浮點變數用來接收 SQRT 函數的回傳值。

說明

SQRT 指令回傳浮點參數的平方根值。SQRT 的結果是一個非負的值。

範例

```
Sub Main()
    Dim MyFloat, Result As Float

    MyFloat=100
    Result = SQRT(MyFloat)          ' the result is 10
    Debug "SQRT of 100 : ", Result, CR

    MyFloat=-100
    Result = SQRT(MyFloat)          ' the result is NaN
    Debug "SQRT of -100 : ", Result, CR
End Sub
```

STRING2FLOAT

語法
FloatVar = STRING2FLOAT(*StringVar*)

操作
將 ASCII 字元轉換為浮點數值。

- ***StringVar*** – STRING2FLOAT 函數的 ASCII 字元字串運算元。
- ***FloatVar*** –用來接收轉換的結果的浮點變數。

說明
STRING2FLOAT 指令將 ASCII 字元轉換為浮點數值。ASCII 字元字串數值表示法的值可以是浮點格式或實數格式。浮點格式是由一個正負符號字元，一個小數點後有五位尾數的基值，一個指數符號字元 E，指數符號字元和兩位數的指數所組成。例如：+3.1416E-01。

範例

```
Sub Main()
    Dim MyString As String * 12
    Dim MyFloat As Float

    MyString = "9500"
    MyFloat =STRING2FLOAT(MyString)     ' incorrect format
    Debug "STRING2FLOAT of ", MyString, " : ", MyFloat, CR

    MyString = "+3.1416E-01"
    MyFloat =STRING2FLOAT(MyString)     ' the result is +3.1416E-01
    Debug "STRING2FLOAT of ", MyString, " : ", MyFloat, CR
```

```
    MyString = "0.031416"
    MyFloat =STRING2FLOAT(MyString)     ' incorrect format
    Debug "STRING2FLOAT of ", MyString, " : ", MyFloat, CR
End Sub
```

STRREVERSE

語法
STRREVERSE(*TargetString*)

操作
回傳一個字串，將所有被指定字串的字元都顛倒過來。

- ***TargetString*** – 要被轉換的字串。

說明
STRREVERSE 指令將給定字串中的字元作反轉。

範例

```
Sub Main()
    Dim TargetString As String * 12

    TargetString = "Hello World!"
    Debug "TargetString is ", TargetString, CR
    STRREVERSE(TargetString)   ' MyString contains "!dlrow olleH"
    Debug "TargetString has been reversed to ", TargetString, CR
End Sub
```

SUB… END SUB

語法

SUB *SubName*(*AargList*) … END SUB

操作

宣告一個有額外參數串列的程序。

- ***SubName*** - 函數的名字，包含了字母序列，數字根底線。開頭必須為一個字母。
- ***ArgList*** - 函數中所需的參數串列。每一個參數前都會加上一個傳值(byval)或傳址(byref)修飾辭用以顯示參數傳遞方法。即使沒有參數括號也不能省略。

說明

SUB 指令宣告了一個函數，可以由它的 SubName 呼叫去執行指定的程式。

範例

```
Sub SayHi()
     Debug "Hi!",CR
End Sub

Sub Main()
     SayHi()
End Sub
```

TOGGLE

語法
TOGGLE *Pin*

操作
轉換引腳的狀態。

- ***Pin*** – 常數或變數值(0 ~23)，用來定義高或低階所要作用到的引腳。 對一個 24 引腳的 BASIC Commander 而言，它的引腳值範圍為 0~15。

說明
這個指令會切換所指定的引腳為 5 伏特高電位或 0 伏特低電位。通常，引腳必須在執行對應的輸入輸出運作前就改變為輸入或輸出模式。然而，因為只有一個引腳參與 TOGGLE 運作，引腳會由系統自動變成輸出模式，不需要預先執行 OUTPUT 指令。其他相關資訊請參考 HIGH 指令。

範例
指令使用方法請參閱 HIGH 指令的範例。

UCASE

語法

UCASE(*TargetString*)

操作

將指定字串改變成大寫字元。

- *TargetString* - 要被轉換的字串。

說明

UCASE 指令將指定字串改變成大寫字元。

範例

指令使用方法請參閱 LCASE 指令的範例。

WORD2FLOAT

語法

Result = WORD2FLOAT(*Argument*)

操作

將字轉換為浮點形式。

- ***Argument*** - WORD2FLOAT 函數的 WORD 運算元。
- ***Result*** - 儲存 WORD2FLOAT 函數回傳值的符點變數。

說明

WORD2FLOAT 指令將 Word 值轉換為符點格式。符點結果將為一個範圍為 0.0 到 6553.0 的整數值。

範例

```
Sub Main()
    Dim MyWord As Word
    Dim MyFloat As Float

    MyWord = 0
    MyFloat = WORD2FLOAT(MyWord)
    Debug "MyWord = ", MyWord, " MyFloat = ", MyFloat, CR

    MyWord = 65535
    MyFloat = WORD2FLOAT(MyWord)
    Debug "MyWord = ", MyWord, " MyFloat = ", MyFloat, CR
End Sub
```

WRITEPORT

語法

WRITEPORT *Port, Data*

如果連接埠號碼是一個常數，可以用下列的形式。

WRITEPORT0 *Data*
WRITEPORT1 *Data*
WRITEPORT2 *Data*

操作

將資料寫入指定的 I/O 埠。

- *Port* – 常數或變數值(0-2)，用來指定連接埠號碼。連接埠 0 包含引腳 P0-P7，連接埠 1 包含引腳 P8-P15，連接埠 2 包含引腳 P16-P23。以一個 24 引腳的 BASIC Commander 來說，連接埠的值是 0 或 1。
- *Data* – 用來定義資料要寫到特定連接埠的常數或變數值。

說明

WRITEPORT 指令是用來將數位資料寫到輸出入埠。跟 OUT 指令不一樣， 引腳的 I/O 模式不會自動變成輸出模式。WRITEPORT 指令包含了 8 個引腳，每一個引腳都可以被設成輸出或輸入模式。你須要詳細的對每一個位元做設定。資料 0 會將相對應的引腳設定成 low 狀態，1 設為 high 狀態。索引值較小的引腳會成為資料位元組中順序較低的位元。

那些被設定為輸入模式的引腳，資料寫入並不會影響到 I/O 引腳。如果稍後這些引腳被重新設定為輸出模式，之前所儲存在緩衝區內的輸出資料就會作用。

範例

指令使用方法請參閱 READPORT 指令的範例。

附錄

附錄 A — ASCII 對應表

以下表格列出 ASCII 前 128 個字符的對應表。最前面的 32 個碼為控制碼，並無標準的螢幕顯示符號，一般僅以文字來代表這些控制碼。例如一般在銀幕上將遊標移至下一行的第一個位置時，就需要使用到 LF 與 CR 二個控制碼。

Dec	Hex	Char	Name / Function
0	00	NUL	Null
1	01	SOH	Start Of Heading
2	02	STX	Start Of Text
3	03	ETX	End Of Text
4	04	EOT	End Of Transmit
5	05	ENQ	Enquiry
6	06	ACK	Acknowledge
7	07	BEL	Bell
8	08	BS	Backspace
9	09	HT	Horizontal Tab
10	0A	LF	Line Feed
11	0B	VT	Vertical Tab
12	0C	FF	Form Feed
13	0D	CR	Carriage Return
14	0E	SO	Shift Out
15	0F	SI	Shift In
16	10	DLE	Data Line Escape
17	11	DC1	Device Control 1
18	12	DC2	Device Control 2
19	13	DC3	Device Control 3
20	14	DC4	Device Control 4
21	15	NAK	Non Acknowledge
22	16	SYN	Synchronous Idle
23	17	ETB	End Transmit Block
24	18	CAN	Cancel
25	19	EM	End Of Medium
26	1A	SUB	Substitute
27	1B	ESC	Escape
28	1C	FS	File Separator
29	1D	GS	Group Separator
30	1E	RS	Record Separator
31	1F	US	Unit Separator

Dec	Hex	Char
32	20	space
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	delete

附錄 B — InnoBASIC 關鍵字

下表依照字母順序列出所有 innoBASIC 語言的關鍵字。

ABS	ACOS	AND	AS
ASIN	ATAN	ATAN2	BELL
BKSP	BOOLEAN	BUTTON	BYREF
BYTE	BYTE2FLOAT	BYVAL	CALL
CASE	CEIL	CHECKMODULE	CLREOL
CLREOS	CLS	CONST	CONTINUE
COS	COUNT	CSRD	CSRL
CSRR	CSRU	CR	DEBUGIN
DEFAULT	DIM	DEBUG	DIRPIN0 ~ DIRPIN31
DIRPORT0 ~ DIRPORT2	DO	DWORD	ELSE
ELSEIF	END	ENUM	EVENT
EXIT	EXP	EXP10	FALSE
FLOAT2DWORD	FLOAT2INTEGER	FLOAT2LONG	FLOAT2REALSTRING
FLOAT2SHORT	FLOAT2STRING	FLOOR	FOR
FREQOUT	FUNCTION	GETDIRPORT	GOTO
HIGH	HOME	IF	IN
INPUT	INTEGER	INTEGER2FLOAT	LCDCMD
LCDIN	LCDOUT	LOG	LOG10
LONG	LONG2FLOAT	LOOP	LOW
MOD	NEXT	NOT	OR
OUTPUT	PAUSE	RANDOM	PERIPHERAL
PERSISTENTBYTE	PERSISTENTDWORD	PERSISTENTFLOAT	PERSISTENTINTEGER
PERSISTENTLONG	PERSISTENTSHORT	PERSISTENTWORD	PIN0 ~ PIN31
PORT0 ~ PORT2	POT	PULSEIN	PULSEOUT

PWM	RANDOM	RCTIME	READPORT
RESETMODULE	RETURN	REVERSE	SELECT
SETDIRPORT	SGN	SHORT	SIN
SQRT	STEP	STRING	STRING2FLOAT
SUB	TAB	THEN	TO
TOGGLE	TRUE	UNTIL	WHILE
WORD	WORD2FLOAT	WRITEPORT	XOR