本書簡介　　書號：44460

開　數：　16 開　頁數：392 頁
適合讀者：組合語言初學者及程式設計師
所需硬體：IBM PC AT/XT 或相容機型
所需軟體：MACRO ASSEMBLER或 TURBO ASSEMBLER
書附磁片：無

本書對讀者之貢獻：

學會組合語言，但就是無法真正的撰寫程式。

本書不論你是初學者或是程式設計師，只要你希望徹底的了解如何寫一個簡潔有效率的程式，或如何對硬體更有效的指揮控制。

本書依功能，I/O 的處理、數值轉換、多位數運算、繪圖、聲音、字串處理以及檔案的管理，分類並例舉詳加說明。

讀者可利用本公司完整的程式，一面執行一面測試，以達最有效率之學習環境。

本書特點

1. 按功能分類，模擬多種需要依輸入/ 輸出/ 暫存器/ 參考區段/ 副程式/ 程式及解說詳細剖解。

2. 本書與組合語言詳解(IBM PC)或組合語言基本應用一起研讀對初學者有相乘之學習效果。

內容概要：

相關書籍：

MACRO ASSEMBLER 5.1 使用手冊
" 組合語言 " 在硬體上之活用
" 組合語言 " 在週邊介面之活用
TURBO ASSEMBLER 參考手冊
組合語言詳解 (IBM PC)

<< 目　　錄 >>

第一章　簡介

第二章　輸入 /輸出程式

第三章　二進位的轉換

第四章　BCD (Binaray Coded Decimal) 的轉換

第五章 浮點表示法轉換

第六章 多位數的算術計算

第七章 繪圖

第八章 聲音

第九章 字串

第十章 檔案管理

第一章　簡介

組合語言對於 IBM PC 上的程式設計而言，是一種最快並且最有威力的語言。然而，即使是一個有經驗的程式設計師，也常常要花很多時間來寫一些特定用途的組合語言程式。尤其是對於一些僅接觸過像BASIC或是 PASCAL 等高階語言的人來說，讓他們來用組合語言寫作程式甚至是不可能的。

這本書的目的就是要讓所有的程式設計者 -- 不論他們是否會組合語言 -- 都能夠在他們各別的程式中來使用快速而有效率的組合語言副程式。這本書內所附的組合語言程式都可以被組合語言以及 BASIC，PASSCAL 來當做副程式叫用。

這本書並不是讓你從頭讀到尾的，除非你對組合語言特別的著迷，否則它只不過像是一本食譜，當你遇到特殊的程式設計問題時才須要用到它。對組合語言用者而言，也可以把本書的組合語言副程式當做一個模組 (Module) 來發展你們自己的程式。

這些副程式的用途分布在許多不同的層面：有關 I/O的處理，數值的轉換、多位數 (Multidigit) 運算、繪圖、聲音、字串(string)處理、以及檔案的管理等等。在這些用途時，程式中的一些直接控制是很重要的，因此有了這些組合語言的副程式，你可以用機器語言直接和 CPU 溝通而增加速度，以及直接控制硬體，凡此種種都是在高階語言時很難作到的。

本書第一章解釋本書的用途，以及一些當你將這些副程式與高階語言連接時，你所需要用到的功能及一些基本的觀點，例如暫存器的使用情形也會討論到，再則本書為什麼不採用 MACRO的原因都將在本章中討論。其餘各章都各自屬於特定的使用範圍，這樣的安排可以讓你很快的找到你所需要的副程式。

1.1 本書是為誰設計的

不論你是初學者或是專家，只要你正在為IBM PC寫一些組合語言或是高階語言的程式，而你希望能有更快的速度或是更強的硬體控制的話，這本書將會是你的好夥伴。

通常替 Intel 8086/8088寫過組合語言程式的人都知道，IBM PC 是採用 8088 微處理機，然而它的指令卻是完全和 8086 一樣的 (請參考螢圃電腦出版之書本所附圖書目錄中，有關組合語言的其他書藉)。本書後面各章的程式都是可以通用的，尤其是一些有關數值 (Numeric)以及字串(String) 的程式對你們來說也許是相當重要。

如果想要有效的使用本書，你並不一定要是一個 8086/8088組合語言的專家，對於初學者以及高階語言程式設計師來說，也不須要去了解這些副程式是怎麼工作，你更不須要真的去寫它們，然而如果你有志成為一個組合語言的老手的話，你可以讀一讀本書，相信是會對你有所幫助。

對初學者以及高階語言使用者，你們可以將這些程式和你自己的程式結合起來，至於結合的方法，我們在本章後面會提到。

1.2 本書的特色

IBM PC有特別的繪圖及聲音的設備，在每一個這類用途上，本書都有單獨的一章來專門使用它們。對高階語言來說，使用這些設備（繪圖、聲音）是不可能的，然而組合語言卻可以。就以畫圖為例，數以十萬計的點必須一次就被快速的處理，然後才能產生實用的畫面，因此在第七章有關繪圖的程式都有，我們儘量以最快的速度做為要求。這些繪圖或是聲音的程式儘管是為 IBM PC 所寫的，但是經過少許修改後，仍然可以替別機器工作。

本書中許多程式必須叫用 IBM PC 上 DOS的命令，我們採用的DOS是 2.0版，然而許多程式仍然可以和別的版本，甚至 CP/M 86 相容，即使是 I/O部份亦然。

在第十章有關檔案管理的部份，這些副程式可以提供一些通常程式設計者必須自己動手做的東西，像是磁碟檔案的管理等等，都可因此而簡化你的程式，此外我們的程式還會讓你知道如何的去使用它，例如從通訊設備上存檔到磁碟中等等。

DOS 2.0 版提供了一個過濾器 (filters)，它是一種類似 UNIX 的特色，讓組合語言程式設計者在使用鍵盤及螢幕的情況下來測試文書處理的程式，即使出錯，仍然不須修改就可直接去使用磁碟中的檔案。在第十章中有許多像這樣過濾器的程式，它們可以直接計算或清除檔案，並且很容易的就可以依你個人的需要加以修改。

在本書中的組合語言程式都已經過仔細的測試，並且在相容的環境下各種因素也都考慮進去了。

1.3　使用本書時所需的東西

首先你要有一部 IBM PC ，或是相容的機器，至少要有640K記憶體，以及 MS DOS 或 PC DOS 3.0 以上。此外還要有 Macro Assembler (MASM)。這些組合語言編譯器都可以允許 31 個字元以下的變數名稱。

你同時也會用到一些螢幕編輯程式，例如 PE,如果你使用 Macro Assembler 來編寫程式的話，你就必須用到兩部磁碟機。

在第七章中那些有關繪圖的程式，必須有 color/graphic Adapter (彩色圖形界面卡)，至於第八章有關聲音的部份，則大部份的 PC 或相容機器都會提供喇叭。

1.4　本書結構

從第二章起，每一章的開頭都會有一段介紹，這是解釋此章中程式的應用範圍、目的，以及一些特定的需要。然後就是有關的程式。

每一個程式在開頭時都會有詳細的說明，它們分為以下幾項：功能、輸入 /輸出說明、暫存器使用的情形、區段 (segment)的使用情形，所需要呼叫的副程式，以及特別注意的事項，對於大部份的使用者或機構而言，這些說明是非常重要的。

在說明之下，就是程式的原始碼：尚未經過 MASM 的 .ASM 的檔案 (source code) 每一個程式的第一行一定是 PROC 的指令。所謂的PROC也就是Procedure.通常都是指一段具有特定功能的程式。它們可以由外面的程式叫用，或是像 Macro一樣直接的插進所需的程式中，對每一個副程式而言，最後一行通常是 ENDP (END Procedure) ，這樣可以通知編譯器這個副程式的範圍，有了清楚的開頭與結尾的指令後，可以使副程式的結構簡單明瞭，而且增加程式設計的效率以及降低維護的費用。

在副程式的每一行後面，我們大概都會有註解來說明它的目的，至於同性質的區域，我們只在第一行加上說明，至於空行的存在只是為了方便閱讀而已，同時也方便你來修改它。

副程式的前後順序是根據他們互相之間的關係決定的，在它被別的程式叫用之前它必須要出現，這樣也可使你在使用時確定你所要找的範圍。

至於各章間的安排順序也是同樣的道理，先從第二章基本的 I/O，然後第三章 Numerical Conversion (數值轉換)，第四章 BCD(Binary Code Decimal: 二進位法表示的十進位數) 以及第五章浮點運算，前幾章先建立了基本工具之後，再來是第六章 Multidigit Arithmetic (多位數運算)，它可以使程式中的四則運算達到任何你所要求的精確度，第七、八章分別是繪圖以及聲音處理，第九章則是字串的處理，以及第十章有關檔案管理的一些功能。

1.5 如何將組合語言連結到其他高階語言

要發揮本書中的副程式功能，你必須將它們和別的程式結合為完整的程式。不論對於組合語言或其他高階語言的程式而言，這些副程式都可以一起工作，我們先從如何與組合語言程式開始討論，然後再討論如何和 BASIC等高階語言結合。在開始前我們先看看有關記憶體區段 (memory segmentation)的問題。

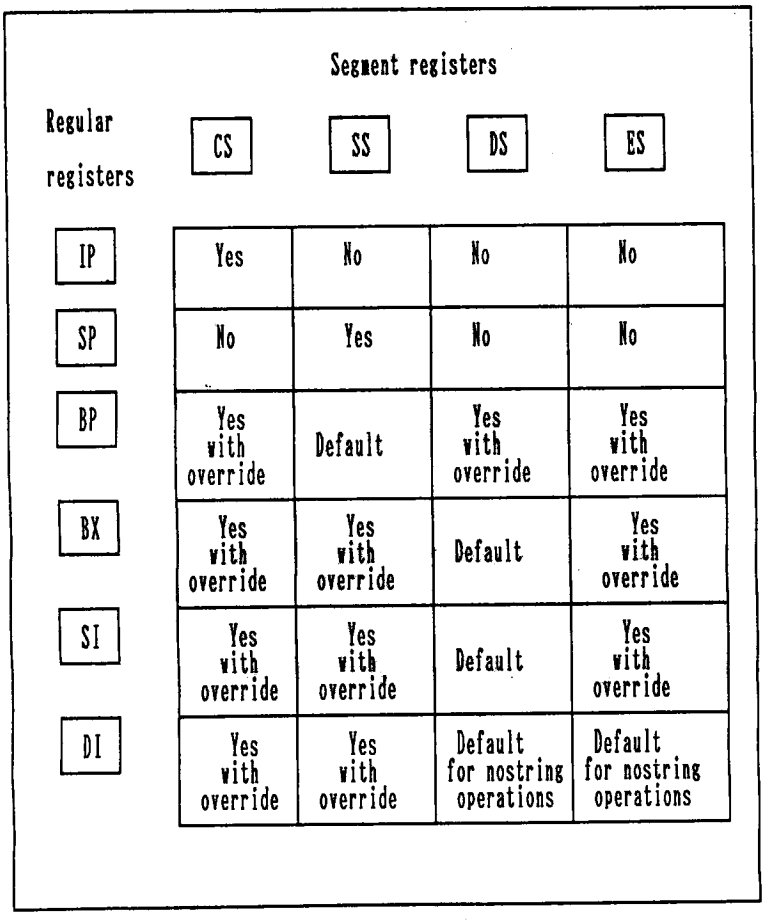
Segment Usage (區段的使用)

對使用 8088 來設計組合語言程式的使用者而言，你必須先對區段 (Segment) 有所了解， 8088 處理機在使用記憶體時以 64K的區段為單位，在總共 1M bytes 的 8088 記憶區中，它有 4個 Segment registers(區段暫存器) 來記載每一個特定區段的開始位址。因此任何時間這四個區段暫存器都是在被使用的情況下。它們分別叫做CS (Code Segment:程式區段)，SS (Stack Segment: 堆疊區段)，DS (Data Segment)以及 ES (Extra Segment: 輔助區段)，他們的功能正如他們的名字一樣，分別是用來存放程式、堆疊 (Stack) 、資料、以及其他更多的資料等等。

當 IP (Instruction pointer: 指令指標暫存器) 開始進出記憶體時，它會找到 CPU的指令，即指向程式區段 (Code Segment) 中的某一個位址，至於其它像 BX, SI,通常是指向資料區段(data Segment) 中的某一處位址，還有 SP、BP，則是指到堆疊區段 (Stack Segment) 中的某一位址，而 輔助區段 (Extra Segment) 對於 DI 而言則是一個預設值 (default)。

如果將許多組合語言程式一起 連結 (link) 起來時，不同的區段會被放在一起而且以許多方式相連結，在某種特定情況下來說，不同程式的程式區段會分別到的放在記憶體中，然後其中一個程式可以用 FARCALLS 來呼叫別的程式，通常在做這類程式區段間的跳越時只須要很少的時間，而且大部份都會由編譯器幫你完成。

對於堆疊區段而言，即使是在一個包含很多副程式的大程式中，也只須在一個副程式中宣告即可。它必須被宣告成堆疊區 (STACK)，然後作業系統就自動會為你的程式產生一個堆疊堆在執行程式之前，作業系統自動將堆疊區段暫存器，以及堆疊指標分別指向堆疊區段的開始及結束的地方，當程式執行時，Stack 會逐漸向下增加。



Regular registers	CS	SS	DS	ES
IP	Yes	No	No	No
SP	No	Yes	No	No
BP	Yes with override	Default	Yes with override	Yes with override
BX	Yes with override	Yes with override	Default	Yes with override
SI	Yes with override	Yes with override	Default	Yes with override
DI	Yes with override	Yes with override	Default for nostring operations	Default for nostring operations

Segment registers

圖 1-1 Segment register 使用情形

資料區段 (Data Segment) 是用不同的方式來處理的，事實上對大多數的程式而言，它的許多副程式是共同使用一個資料區段，這樣可以很容易的安排 -- 只要在各副程式中使用的資料區段都宣告成 PUBLIC 並且是同一個名字即可。如果程式中只用一個資料區段，可以簡化程式並減少執行程式時交換跳越所花費的時間。將程式的本身與資料分開，

是一個相當重要的觀念，這也是現代程式語言的一種類似模組的（module）結構化設計，此外，因為每一個區段只有 64K，因此將其它東西移開（例如資料、堆疊（ stack ）...等等），可以留更大的空間以供程式本身使用。

在這種安排下，即使是程式在編譯時，其中任一項資料位址的offset（ 位移 ）也是無法計算的，除非是所有的副程式都連接後才能知道，因此，組合語言指令中的 OFFSET ，如果遇到許多副程式只用一個資料區段的情況時，無法傳回這個區段中資料的正確位移大小。但是對 CPU而言，它仍提供了一個 LEA (Load Effective Address：載入有效位址 ）指令，在程式執行時，它會將特定變數的位移裝進一個特定的暫存器中，也就是說，當你需要去計算一些變數的位址時，CPU 的指令會代替OFFSET的指令。

輔助區段 （ Extra Segment ）有許多功能。例如它可以當做是另一個 Data Segment,也可以當做作業系統所需資料的特定存放區，或是當做像是影像記憶區之類的特別區段，你都可以在我們提供的副程式中看到它的用法。

兩種與其它程式連結的方式

這本書的副程式可以和其他組合語言連結，你可以用 EDITOR （ 編輯程式）來做，將它們直接寫進你自己的程式，或是把他們放在別的檔案中，然後等到執行時再叫用它們並且連結起來，你當然也可以兩者都用。