台港及海外中文报刊资料专辑

电子电讯工业

第 3 辑

1986

书目文献出版社

电 子 电 讯 工 业（3）

——台港及海外中文报刊资料专辑（1986）

北京图书馆文献信息服务中心剪辑

书目文献出版社出版

（北京市文津街七号）

北京百善印刷厂印刷

新华书店北京发行所发行　　各地新华书店经售

787×1092毫米　1/16开本　5　印张　128　千字

1987年3月北京第1版　1987年3月北京第1次印刷

印数 1—2,000 册

统一书号：15201·12　定价：1.30元

〔内部发行〕

出版说明

由于我国“四化”建设和祖国统一事业的发展，广大科学研究人员，文化、教育工作者以及党、政有关领导机关，需要更多地了解台湾省、港澳地区的现状和学术研究动态。为此，本中心编辑《台港及海外中文报刊资料专辑》，委托书目文献出版社出版。

本专辑所收的资料，系按专题选编，照原报刊版面影印。对原报刊文章的内容和词句，一般不作改动（如有改动，当予注明），仅于每期编有目次，俾读者开卷即可明了本期所收的文章，以资查阅；必要时附“编后记”，对有关问题作必要的说明。

选材以是否具有学术研究和资料情报价值为标准。对于某些出于反动政治宣传目的，蓄意捏造、歪曲或进行人身攻击性的文章，以及渲染淫秽行为的文艺作品，概不收录。但由于社会制度和意识形态不同，有些作者所持的立场、观点、见解不免与我们迥异，甚至对立，或者出现某些带有诬蔑性的词句等等，对此，我们不急予置评，相信读者会予注意，能够鉴别。至于一些文中所言一九四九年以后之“我国”、“中华民国”、“中央”之类的文字，一望可知是指台湾省、国民党中央而言，不再一一注明，敬希读者阅读时注意。

为了统一装订规格，本专辑一律采取竖排版形式装订，对横排版亦按此形式处理，即封面倒装。

本专辑的编印，旨在为研究工作提供参考，限于内部发行。请各订阅单位和个人妥善管理，慎勿丢失。

北京图书馆文献信息服务中心

程序设计语

最刺激最 陈 明 1

漫谈电脑 李淑珍 8

以Bench 12

认识BAS 赵文钦 13

编译与解 20

COBOL语 李淑珍 21

高效率、 张永宗 30

区块式结构 赵文钦 37

人工智能与专家系统

专家系统的技术 43

机器人与自动化 范 毅 51

电脑软体与人工智能 孙春在译 54

人工智慧在我国 马西屏 65

开放系统面临的挑战和希望 谢坤融译 67

通往了解思考之路 曾慧娟译 74

日本的机器人 沈天赐 80

最刺激最快速的組合語言

如果您精通這種奇難無比的語言，它將帶給您在PC上編碼最簡潔，執行最快速，且最令人興奮的程式設計經驗。

／陳明

已經半夜兩點了，新寫的組合語言程式的第一部份即將完工，程式印出來足足有四張報表紙，全是一些如 MOV，JMP，SHR，ADC，MUL，DIV，PUSH 與 POP 之類奇奇怪怪的指令，其中沒有任何一個單獨的指令，可以提供您些微有關它到底在做什麼的線索；那感覺彷彿要用牙籤挑起一座吊橋一般。那，這程式到底在做什麼呢？也許這其中的一大堆程式碼所做的就相當於一個 BASIC 述句：

```
INPUT "Enter 3 numbers! ",A,B,C
```

問題並不就那麼簡單，萬一所輸入的數字中有一個帶著兩個小數點，可能程式就此當掉，且畫面上剎那間佈滿"＆"號也說不定；看起來很神奇，但您還得把毛病找出來。明天再做嗎？不，程式快完工了，也許改正這個毛病程式就 OK 了，再試一下吧！

簡潔而快速

可以確定的是：讓一個組合語言程式執行無誤雖是個人電腦上最令人興奮的經驗之一，但也是個人電腦上所能辦到的最難事情之一。

組合語言程式可直接翻譯成機器碼，它是 CPU 可自記憶體讀取並立即執行的碼。我們通常寫的高階語言程式，也需由編譯器將它翻譯成機器碼才能交給 CPU 執行，但這是一種很間接、概括性的翻譯方式，所譯出來的機器語言程式既臃腫又遲緩。除非那天編譯器程式可以比人還聰明，否則組合語言仍將永遠是個人電腦上編碼最簡潔，執行最快速的語言。尤其在行家手中，組合語言可以做出如 LOTUS 1-2-3 一般功效強大的程式，如 Xy-Write 一般速度奇快的軟體，以及簡潔得令人難以相信的 Turbo Pascal 編譯器與編校器。

學習不易難以通用

那麼人們爲何不直接用組合語言來寫所有程式呢？主要原因不單單在於它難（學習所需的時間長，設計程式所需的時間也長），更在於它缺乏通用性（portability）。每部電腦都有它自己的組合語言且各不相同，因爲組合語言對各廠機型結構的依賴性甚高。就發展商用軟體程式者的觀點而言，通用性成爲一個大問題：原來在某一機型上寫好了的程式，如果要移到另一型 CPU 不同的機器上去時，幾乎整個程式都要完全重寫。

在 IBM PC 上所使用的組合語言，以其所用之 CPU 爲名，叫做 8086/8088 組合語言。這是一種極端困難的語言，學習難，精通更難，維修程式還要更難。因爲原始碼中的每一行要翻譯成 1 到 6 位元組的機器碼，所以組合語言原始程式檔案，通常要比所產生的機器語言程式檔案大很多很多倍。組合語言也沒有固定的結構，以 8086／8088 組合語言爲例，光 JUMP 指令就有 32 種不同的變化；而且與高階語言不同的是，組合語言程式中絕不可能不用到一大堆跳躍（JUMP）指令，否則根本就寫不出任何程式。

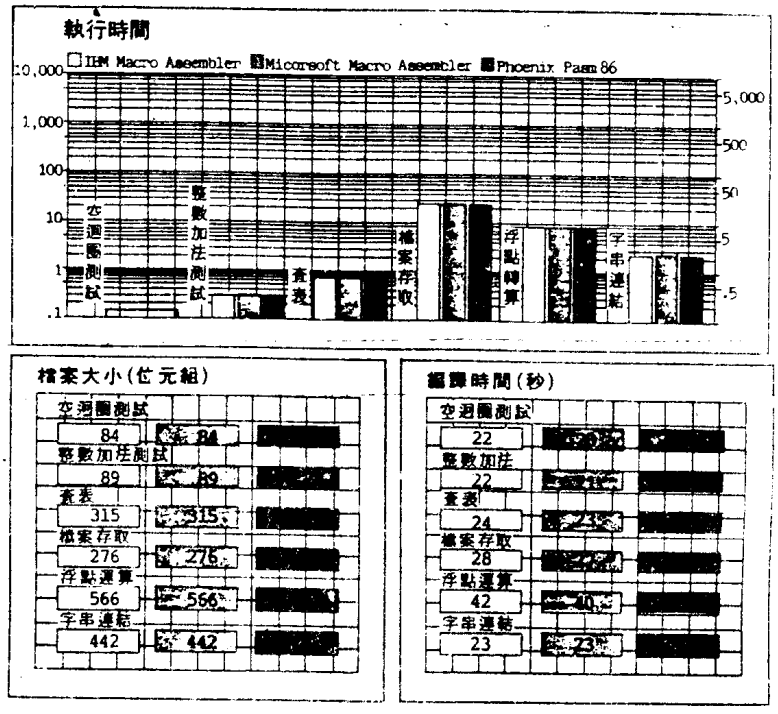
8086/8088 組合語言中沒有 PRINT 指令，沒有 INPUT 指令，而且只能進行整數間的運算。當需要進行浮點運算時，你必須自己寫一個浮點運算常式。需要自鍵盤讀入東西，需要在螢幕上印出東西，或需要存取檔案時，也都必須自己寫個常式，而且這時不僅要會使用組合語言助憶碼（mnemonics），還必須對 PC-DOS 的函數呼叫（function call）及 BIOS 瞭若指掌，運用自如。

即使只是一件簡單的事情，例

如把以十六進位儲存的一個數字，轉換成一般人可讀得懂的十進位ASCII代碼，（這在BASIC中只要一道PRINT指令），也都得要一個很大的副程式，並把四捨五入，去除開頭的0，及在正確位置加入逗號或小數點等等細微末節都一一考慮進去。組合語言讓您可以完全掌握機器，但每一細節都要自行費心。

負責將組合語言原始程式轉換成機器碼的程式叫做組合器（Assembler）。組合器不會告訴你PUSH與POP是否成雙成對，也不會告訴你程式中那裡是否忘了個RET指令。不過你很快就會知道程式有毛病（雖然不一定知道錯在那裡）。——通常在組合語言程式還有"蟲，bug"存在的階段最常見的現象就是：系統當掉。

有時候一些組合語言程式設計老手，會從螢幕上移開迷矇的雙眼自言自語道："這樣不值得，也許我應該用C或Pascal來寫這程式！"。在痛苦的組合語言程式經驗中，這當然是個很大的誘惑，於是很多人就此改用如Pascal或C之類的高階語言——雖然組合語言仍是那麼地神秘而神奇。你可以設計出一個不到20位元組且具實際用途的.COM檔案的程式（例如簡單的WordStar-to-ASCII轉換程式）；然而有些程式，如被DOS或BIOS固定放在記憶體中的一些公用程式，就絕對必須用組合語言才寫得出來。



IBM Macro Assembler, Microsoft Macro Assembler，與Phoenix Pasm 86 Macro Assembler等三種巨集組合器的benchmark測試結果。

- 所有測試都是在一部配有兩個軟式磁碟機，但沒有8087並處理器（coprocessor）的IBM PC上執行。
- 所有檔案大小都是以位元組為單位，所有時間都是以秒做單位。
- 因為所譯出來的機器語言碼完全相同，檔案大小與執行時間對每一組合器也都完全相同。
- 不論對那一個組合器，檔案存取如果在硬式磁碟機上做的話都會變快很多很多。

分析三種組合器

我們就IBM PC的各種8086/8088組合器中挑選了三種專業化且功能齊備的巨集組合器（Macro Assembler），分別來自IBM，Microsoft以及新近崛起的Phoenix等三家公司。巨集組合器可將一些常用的指令集合（如讀入一筆資料，印出一個數值等），配以可替換的參數存在檔案中，叫做巨集指令（macro instruction）要用時不必再重覆抄寫這些常用的指令集合，只要引用巨集指令之名

```
CSEG            Segment Public 'CODE'
                Assume  CS:CSEG, DS:CSEG, ES:CSEG, SS:CSEG

                Extrn   StartTime:Near, PrintTime:Near

                Org     8100h

Entry:          Call    StartTime               ; External Subroutine

                Mov     CX,10000                ; 10,000 repetitions
Test1Loop:      Loop    Test1Loop               ; Same line loop

                Call    PrintTime               ; External Subroutine
                Int     20h                     ; Exit Program

CSEG            EndS

                End     Entry
```

IBM Macro Assembler: 空迴圈測試

```
CSEG            Segment Public 'CODE'
                Assume  CS:CSEG, DS:CSEG, ES:CSEG, SS:CSEG

                Extrn   StartTime:Near, PrintTime:Near

                Org     8100h

Entry:          Call    StartTime               ; External Subroutine

                Sub     AX,AX                   ; AX starts at zero
Test2Loop:      Cmp     AX,32767                ; Count until 32767
                Je      Test2End
                Inc     AX                      ; Otherwise increment
                Jmp     Test2Loop               ; And do it again

Test2End:       Call    PrintTime               ; External Subroutine
                Int     20h                     ; Exit

CSEG            EndS

                End     Entry
```

IBM Macro Assembler: 整數加法測試

稱即可。

至於六個 benchmark 測試程式，我們儘量讓一般熟悉高階語言的讀者也能接受。查表測試中則避免使用 REP, MOVSW 等快速搜尋指令。

因爲8087浮點運算共處理器（coprocessor）的指令可直用在 .ASM 檔案中，使用 8087 可大幅簡化並加快浮點數值之運算，且所選用的三個組合器也都能接受8088與8087之合作，所以我們的浮點運算測試程式也使用了8087。這整個程式的 .COM 檔案長度只有136位元組，其中一半以上是用以計時。這1000次的浮點乘法與除法運算只花了8087不到1.5秒的時間，驚人嗎？不，只因爲這是用組合語言的緣故！

IBM Macro Assembler

IBM Macro Assembler 的2.0版顯然是組合器中的上上之選。它包括一片磁片及兩本手册。雖然磁片上 MASM.EXE 程式版權爲 IBM 與 Microsoft 共享（事實上也與 Microsoft 新版的組合器非常相似），IBM 在手册中仍有詳盡的說明。

從 1.0 版到 2.0 版最大的改變在於增加了有關 8087/80287 共處理器，及 80286 實位址（real-address）非保護模式（nonprotected mode）的指令。1.0 版裡引人非議的一些毛病也都已清除：SHR與SHL假工作碼（pseudo-op）如今已正常工作，型式檢驗也已大幅改進。令人驚訝的是 IBM 一改常態，在手册中坦白承認 1.0 版確有實際情況與說明文件不符之處。

另一個很確實的改進之處在於：2.0版的手册及其磁片上之範例檔案明明白白地告訴使用者，製造一個可執行的組合語言程式需要些什麼？這看起來只是一些瑣事，但在 1.0 版時的確也難倒過不少初次接觸組合器的使用者。2.0 版手册中甚至還詳細討論 .COM 檔案與 EXE 檔案之結構及各自的優缺點。

在 IBM PC 問世以前，組合語言程式設計人員常會感嘆早期的組合器多缺乏公用程式庫，使用不便。公用程式庫是指將一些常用的副程式以 OBJ 格式存起來，可在連結時放入程式中。2.0 版的IBM Macro Assembler 裡，包含了一個程式庫管理器程式 LIB（版權也是與 Microsoft共享）；可惜的是 IBM 在手册中並未詳細說明 LIB的使用方法，只用 10 頁的篇幅介紹其指令及語法，對於例如使用具有一致性的段落名稱（segment name），爲日後版本留餘地…等建構優良程式庫的原則卻隻字未提。

手册中足足有 38 頁的篇幅是在討論 IBM 自行發展的預處理器程式 SALUT（Structured Assembly Language Utilities，結構化組合語言公用程式）。使用 SALUT 前得先寫好含有控制程式流程用之特殊結構化指令的程式，賦予 .SAL 檔名，然後執行 SALUT 即可將程式轉換成一般的 .ASM 檔案，再依照一般的步驟以 MASM 組合成機器語言程式。

在組合語言程式中加入一些結構化特性的確是一個很有趣、也很值得一試的做法；不過第一次使用 SALUT 時聽到印表機重置的聲音（這表示 SALUT 是用 BASIC 寫的）也許會嚇你一跳。拿 BASIC 程式來當組合器中的結構化預處理公用程式，確實讓人很難接受。

其實對一個組合器而言，一個好的程式庫系統遠比一個如 SALUT 之類的好預處理系統來得重要；很可惜在手册上 IBM 沒有對前者做應有的詳細討論。

這套裝系統中還包括一個連結程式，一個 cross-reference 公用程式，及一個迷你版的組合器 ASM。ASM 不能處理巨集指令及 8087 指令，在遇到錯誤（error）時也只列出其號碼；它只有在記憶體非常不夠時才派得上用場。

總而論之，IBM Macro Assembler 是一個很好，很紮實，功能齊備的組合器，所附的手册對初學者及程式設計老手都適用；值得每一位 PC 玩家把它收做自己軟體庫的一部份。

Microsoft Macro Assembler

3.0 版的 Microsoft Macro Assembler 與 IBM Macro Assembler 2.0 版幾乎完全相同，只是多添幾個功能而已。它一樣地也在非保護模式中包含了 8087，80287，80286 的指令，並加上了保護模式下的 80286 指令（雖然這在大多數程式中不太用得到）。比起 IBM，它沒有如 SALUT 之類的程式（這沒什麼關係），但有 LIB，MAKE，SYMDEB 等程式；其中 LIB 與 IBM 一樣是個程式庫，MAKE 是個程式維護器（program maintainer），SYMDEB 是個符號偵錯器程式（symbolic debugger）。這裡頭的 SYMDEB 可與 Microsoft 其他語言產品共用，單單它就值得整套組合器系統的價錢了。

這整套系統裡最可惜的就是那

```
CSEG            Segment Public 'CODE'
                Assume  CS:CSEG, DS:CSEG, ES:CSEG, SS:CSEG

                Extrn   StartTime:Near, PrintTime:Near

                Org     8100h

Entry:          Jmp     Test4           ; Skip over data

ShortString     db      (ShortStringEnd - ShortString - 1)      ; Length
                db      'This is a string'                      ; String
ShortStringEnd  Label   Byte

LongString      db      (LongStringEnd - LongString - 1)
                db      'This is a string with lots of words in it.'
LongStringEnd   Label   Byte

CatString       db      0,255 dup (0)

Test4:          Call    StartTime       ; External Subroutine

                Cld                     ; String Moves forward
                Mov     CX,10000        ; Number of Repetitions

Test4Loop:      Push    CX

                Mov     SI,Offset ShortString
                Lodsb                   ; Get length
                Mov     DL,AL           ; Save length of 1st string
                Sub     AH,AH           ; Zero out type byte
                Mov     CX,AX           ; CX = length
                Mov     DI,[ + Offset CatString ; Destination
                Rep     Movsb           ; Move it in

                Mov     SI,Offset LongString
                Lodsb                   ; Length of second string
                Mov     CX,AX           ; Set to CX also
                Add     AL,DL           ; AL = Length of both strings
                Jnc     StringLengthOK  ; If under 255, no problem
                Mov     AL,255          ; Truncated length of total string
                Mov     CL,AL           ; Now CX = 255
                Sub     CL,DL           ; Now CX = truncated length of 2nd
StringLengthOK: Rep     Movsb           ; Move in 2nd string after first
                Mov     [CatString],AL  ; Put in the total length
                Pop     CX              ; Get back repetition counter
                Loop    Test4Loop       ; Do it CX times
                Call    PrintTime       ; External Subroutine
                Int     20h             ; Exit
CSEG            EndS
                End     Entry
```

字串連接測試

本的/簡不良、使用不易的手冊，可惜之至。如果您買了這套軟體，建議您去找一些有關8086/8088 組合語言指令的說明資料做參考，因為Microsoft的手冊中對每道指令的說明文字都只有一行，例如

"JCXZ label — Jump on CX zero"

或"Loop label — Loop"

與IBM一樣的是，Microsoft在手冊中只給LIB 14頁的篇幅，雖說是比IBM 要詳細，但還是很簡陋，與實際所需相距甚遠。至於程式維護器則是 Microsoft所獨有，它在您建立合適的說明檔案時，可用以檢查 .ASM, .OBJ, .MAP, .EXE 等各式檔案的修改日期，並進行更正日期所需的一切組合與連結工作。

SYMDEB

Microsoft 這套巨集組合器真正最可貴的在於其符號偵錯器 SYMDEB，它看起來，用起來都彷彿是 DOS DEBUG公用程式的改良版，但實際上有一些重大的改變。一般 DEBUG的程式表列中沒有敘述性標記（label），也不能有地址名稱，事實上對絕大多數人而言是非常難讀的；例如看到 CALL

```
CSEG            Segment Public 'CODE'
                Assume  CS:CSEG, DS:CSEG, ES:CSEG, SS:CSEG

                Extrn   StartTime:Near, PrintTime:Near

                Org     8100h

Entry:          Jmp     Test5                   ;Skip over data

Array1          dd       1.0 ,  2.0 ,  3.0 ,  4.0 ,  5.0
                dd       6.0 ,  7.0 ,  8.0 ,  9.0 , 10.0
                dd      11.0 , 12.0 , 13.0 , 14.0 , 15.0
                dd      16.0 , 17.0 , 18.0 , 19.0 , 20.0
                dd      21.0 , 22.0 , 23.0 , 24.0 , 25.0

Array2          dd      25 dup (?)

Test5:          Call    StartTime       ; External Subroutine

                Mov     CX,1000                 ; Number of trials

Test5Loop:      Push    CX

                Sub     BX,BX                   ; Initial Index
                Mov     CX,25                   ; Number to move

Test5Move:      Mov     AX,Word Ptr Array1[BX]  ; Get from first
                Mov     DX,Word Ptr Array1[BX + 2]
                Mov     Word Ptr Array2[BX],AX  ; Put into second
                Mov     Word Ptr Array2[BX + 2],DX
                Add     BX,4                    ; Push up BX
                Loop    Test5Move

                Pop     CX
                Loop    Test5Loop       ; Do it 1000 times

                Call    PrintTime       ; External Subroutine
                Int     20h             ; Exit

CSEG            Ends

                End     Entry
```

IBM Macro Asembler: 查表測試

OE87時只覺很眼熟，但不會知道接下來要執行的就是FATAL ERROR副常式。

若要完全發揮 SYMDEB的功效，得先連結其選擇項目 map，再執行 MAPSYM 公用程式，把程式所在的 .MAP 檔案轉換成 SYMDEB 識得懂的符號檔案，接著並自動在螢幕上列出各程序、群（group）、及標記之名稱與位址。當然，如果只是單一模組程式的話，.MAP對照檔案是不會有太多此類訊息出現的，但我們可以把原始程式檔裡的所有變數與常式都故意弄成公用（public），以便 SYMDEB可列得出並用得了它們。

SYMDEB還有其他許多美妙的特色，例如它可將資數變數以短格式，長格式或 10-byte 格式傾印（dump）出來。如果電腦的序列埠（serial port）接的有傾印終端設備，還可將 SYMDEB 的輸入與輸出設定到此終端設備上去，以免干擾原程式中有關鍵盤及畫面的工作。

SYMDEB 還可與 Microsoft 的 C, FORTRAN, Pascal 編譯器等其他語言產品共同使用，以便在原始程式階段進行偵錯工作。它可據出任何鏈結目的模組或程式庫中的公用位址（public address），據此我們可以經由各被編譯程式之內部工作進行追踪。

SYMDEB 實在是個奇妙的小程式，只放在 Microsoft Macro Assembler 套裝軟體中一起賣，有的人買這套軟體要的其實只是

SYMDEB 程式而已。

如果您買了 IBM 與 Microsoft 兩套巨集組合器，大可挑 IBM 的手册與 Microsoft 的程式出來用；除了 Microsoft 的程式比 IBM 晚四個月問世以外，在使用上兩個 MASM.EXE 程式幾乎完全相同，無法辨別。

Pasm 86

Pasm 86 算是專業化巨集組合器市場上的新進同伴，它乃是原本專寫 IBM 相容 ROM BIOS 的 Phoenix Computer Products 公司的產品。Pasm 86 聲稱是不但與 IBM 相容也與 Microsoft 一相容，且 bug較少，速度快一倍；事實上也差不多就是這樣。

正如廣告所宣傳的，對大多數程式而言，Pasm 86的速度大約是 IBM 或 Microsoft 組合器的兩倍。如果平常 MASM 處理程式時您得盯著螢幕等上很久的話，可能特別想看看到底這所謂兩倍是有多快？（也可能想把程式分多重目的模組並引用程式庫）。不過一般標準測試程式都很短，而組合之快慢主要取決於磁碟檔案的存取動作，因此感覺不出這所謂兩倍的差異；但如果程式有五頁或更長時，就可以感覺到明顯的巨大差異。

相容性如何

至於與 IBM 或 Microsoft 的相容性倒是應該加個問號！首先，指令行就明顯地有差異：其 .ASM

檔案存取測試

```
;       TEST6.ASM - Assembly Language Test Program 6 -- File Input/Output
;       ------------------------------------------------------------------
;       Charles Petzold, August 3, 1985
;
CSEG            Segment Public 'CODE'
                Assume  CS:CSEG, DS:CSEG, ES:CSEG, SS:CSEG
                Extrn   StartTime:Near, PrintTime:Near
                Org     005Ch
FileBuffer      Label   Byte                    ; Put buffer in PSP
                Org     0100h
Entry:          Jmp     Test6                   ; Skip over data
FileAscIIz      db      'TEST.DAT',0                    ; Name of File
FileHandle      dw      ?                       ; File Handle
RecordSize      dw      132                     ; Record Size
ModifiedFile    db      'Modified',0            ; New File contents
ErrorMessage    db      '-- File I/O Error -- $'        ; All-purpuse message
Test6:          Call    StartTime               ; External Subroutine
                Cld                             ; String Moves Forward
                Mov     DX,Offset FileAscIIz    ; File name
                Mov     AL,2                    ; Read / Write Access
                Mov     AH,3Ch                  ; CREATF file Call
                Int     21h                     ; Do it
                Jc      Test6Error              ; Error Exit
                Mov     [FileHandle],AX         ; Otherwise save handle
                Sub     AX,AX                   ; Record Count
WriteLoop:      Push    AX                      ; Save current record
                Call    LSEEK                   ; Points file to record offset
                Jc      Test6Error              ; Error exit
                Call    WRITE                   ; Writes a 132 byte record
                Jc      Test6Error              ; Error exit
                Pop     AX                      ; Get back current record
                Inc     AX                      ; Up it by 1
                Cmp     AX,100                  ; See if reached 100 yet
                Jb      WriteLoop               ; If not, keep going
                Sub     AX,AX                   ; Phase 2 Record Count
ReadWriteLoop:  Push    AX                      ; Save record count
                Call    LSEEK                   ; Move pointer to record
                Jc      Test6Error
                Call    READ                    ; Read 132 byte record
                Jc      Test6Error
                Mov     DI,Offset ModifiedFile  ; Beginning of string to write
                Mov     SI,DI                   ; Set SI to it also
                Mov     CX,[RecordSize]         ; Max characters in record
                Mov     AL,0                    ; Search for terminating zero
                Repne   Scasb                   ; Find it
                Sub     CX,[RecordSize]         ; Convert to string length
                Not     CX
                Mov     DI,Offset FileBuffer    ; Destination is buffer
                Rep     Movsb                   ; Move it in
                Pop     AX                      ; Get record number again
                Push    AX
                Call    LSEEK                   ; Set pointer to record
                Jc      Test6Error
                Call    WRITE                   ; Write the buffer
                Jc      Test6Error
                Pop     AX                      ; Record number
                Inc     AX                      ; Increment it

                Cmp     AX,100                  ; Continue if less than 100
                Jb      ReadWriteLoop
                Mov     BX,[FileHandle]         ; Handle of file
                Mov     AH,3Eh                  ; CLOSE File Call
                Int     21h
                Jmp     Test6Exit
TestError:      Mov     DX,Offset ErrorMessage  ; Error exit
                Mov     AH,9
                Int     21h
Test6Exit:      Call    PrintTime               ; External Subroutine
                Int     20h                     ; exit
```

```
;       Subroutines for File I/O
;       ------------------------
LSEEK:          Mul     [RecordSize]    ; DX:AX = offset in bytes
                Mov     CX,DX
                Mov     DX,AX           ; CX:DX = offset in bytes
                Mov     BX,[FileHandle]
                Mov     AX,4200h        ; Move from beginning of file
                Int     21h
                Ret
READ:           Mov     AH,3Fh          ; Read Call
                Jmp     Short READWRITE
WRITE:          Mov     AH,40h          ; Write Call
READWRITE:      Mov     DX,Offset FileBuffer    ; Buffer
                Mov     CX,[RecordSize]         ; Number of bytes
                Mov     BX,[FileHandle]         ; Handle
                Int     21h                     ; Call DOS
                Jc      ReadWriteExit           ; Carry set if error
                Cmp     AX,CX                   ; Also set if AX < CX
ReadWriteExit:  Ret
CSEG            EndS
                End     Entry
```

檔名之後不能放個逗號，且所有參數旗號（flag）前面必須採 UNIX 方式放個短橫（dash），不能用一般DOS的斜線（slash）。

浮點運算測試也有問題：浮點乘法及除法測試程式都假設數值是以 BASIC的雙倍精確度（double-precesion）格式儲存，而非8087格式。IBM與Microsoft 兩組合器皆可產生這兩種格式，以BASIC爲預設（default）格式，不成問題，可是 Phoenix 的組合器卻只有8087格式，不能切換至BASIC格式，因此浮點測試程式移到 Pasm 86 上時得做某種程度的修改，使相容性再打個折扣。

對大多數程式而言，若以Pasm 86寫好後以IBM的 LINK 連結，則所產生的檔案將與原本即出自IBM或Microsoft 之組合器的完全相同。其中間階段之 .OBJ 檔案雖然有些不同，但只要 LINK 能正確處理就沒什麼關係。

有些在IBM與Microsoft 組合器之下毫無問題的程式，到Pasm 86 下會有些麻煩，問題出在組合語言程式會用到的組合語言假指令（assembly language directive）而非指令碼（instruction code），例如在多模組程式裡用到EXTRN指令，還有某些指令用到"&"與"%"巨集假指令等情況。有時候我們也有法子弄些小改變，讓程式只能在 Pasm 86下動作，不能在IBM與Microsoft組合器下使用。這並不表示它們之間誰對誰錯，因爲在手册出錯或講不清楚時，程式設計人員原就得用嘗試錯誤（trial and error）的土方法找出各該組合器或編譯器下什麼能做，什麼不能做？有的人可能不贊成這種說法，但畢竟 Pasm 86 不是IBM出品的，免不了會有這種情形。

有時一個程式在 Pasm 86下組合時不出錯，到以2.2版 LINK（取自DOS 3.1）連結卻會出現"DUP record too complex"的錯誤訊息；這可能只是 Phoenix 故弄玄虛。

Phoenix的手册資料比 Microsoft 豐富，但不如IBM 完整，印刷也不及 IBM。此手册對8086/8088/80286的每道指令與組合語言做指令都用一頁的篇幅來描述，可是對 8087 指令卻未適當說明。

如果您需要一個速度更快，功能更齊備的巨集組合器，Phoenix 這套產品很合適；雖然它不能與IBM及 Microsoft的組合器完全相容，相信您終必能習慣的。

結論

這三個組合器所做的都是一樣的事情。除 Phoenix 速度最快及 Microsoft 所附的一些很好的公用程式外，IBM Macro Assembler 2.0 版應該算是其中最好的一個—只因它包含了有經驗的組合語言程式設計者所最想要的：良好的參考手册。

本文所敘各組合器之需求對照表

Name	RAM	disk drive	DOS Version
IBM Macro Assembler	128K	one	DOS 1.1 or higher
Microsoft Macro Assembler	128K	one	DOS 2.x
Phoenix Pasm86 V1.01	100K		DOS 2.x

（原载：微电脑时代〔台〕1986 年 5 期 114 — 120 页）

漫談電腦語言

雖然你不是專業的程式設計人員，但可能已花了不少時間與心力在PC上設計自己的程式；不過，您是否選用了最合適的程式語言？衆多語言中，又當如何比較與判斷？

／李淑珍

根據統計，個人電腦使用者中至少有60％以上自行設計所需要的程式，但卻只有4％是專業的程式設計人員。

就現有數千種套裝軟體程式而言，從簡單的打字訓練到完備的文書處理，從簡單的財務分析到整體的會計系統，幾乎可說是無所不備；但爲什麼仍有這麼多的使用者寧願親自與神秘艱深的電腦語言奮戰？爲什麼他們願意去學習新語言和複雜的程式設計技巧——這種技巧事實上應該只是專業程式人員的事情！爲什麼他們不直接去買個符合自己所需的現有套裝軟體呢？

這當然牽涉到許多因素，其中一個主要的理由是：包括個人電腦的ＯＳ在內，沒有任何一個現成的套裝軟體可以始終完全符合每一個個別使用者的所有要求，使用者很可能總是需要再多一項功能。但是一般套裝軟體却不可能同時滿足衆多使用者的各種需求，以致許多人乾脆挽起袖子，自行設計自己所需要的程式。

另一個原因是爲了要在所使用的各種套裝軟體與硬體設備間建立傳輸資料的橋樑，因爲它們相互之間有時並不能有效地溝通資料。例如有的人可能想要在“LOTUS 1-2-3”或“Microsoft Word”中以列表機印出用“WordStar”所建的檔案，或者也可能有人需要在“dBASE Ⅱ”（或dBASE Ⅲ）中用到以“Visicalc”建的資料等。

充滿挑戰性的遊戲

不過最迫切的原因大概是爲了享受自行設計程式的挑戰性及成就感。那種刻骨銘心的程式設計經驗，比一些激烈運動過之而無不及。

人的一生中可以遭遇許許多多具有強烈挑戰性的事物，諸如探險，賽車等都是；但只有程式設計可以贏了又輸，輸了又贏，卻仍不造成任何心靈或生理上的創傷。許多人在設計程式時不眠不休地把一個程式做到正確無誤，隔天晚上仍又向電腦挑戰，只爲了使程式再增加一項功能或稍做改進。到目前爲止，程式設計所造成的唯一眞實的損害只有在離婚法庭上見得到：許多妻子因受不了丈夫竟日沈迷電腦、冷落家人，憤而訴請離婚；不過我們很難想像有那個法官在判決離婚時，會以“冷酷而不專情的程式設計”做理由。

不管您是要解決任何問題，個人電腦程式設計永遠安全、有趣，並讓您覺得值得。

話說回來，設計程式眞的很容易嗎？當您第一次嘗試時也許會覺得艱難無比，甚至因而心灰意冷，但久而久之，就會發現這其實遠比設計一個微波烤爐或解一道數學難題來得簡單而安全。至於容易到什麼程度，就看您設計的是那種程式？用的是那種程式語言？——這其間並不很有關聯性，完全因人而異。

神奇的運算性

另一個有趣的現象是：電腦語言的壽命往往比它們所使用的電腦要長，因爲硬體發展的速度比軟體快了許多。如果每一種用軟體都必須隨著新硬體設備的引進速度而一再重寫，那將是人類工作負荷及財力負擔上極大的災難。因此對一個決策者而言，最重要的應是創造一個穩定的電腦使用環境。

話說從頭

電腦世界中充滿了各種電腦語言，正如同歷史學與地理學中充滿了各種人類語言。每一種電腦語言都有其由來、發展歷史、特殊目的，及其擁護者——技術上，或純粹基於情感因素的擁護。

遠在電腦語言誕生之前，電腦

即已誕生。第一部可程式化的電腦在設計程式時，必須撥換開關，或在麵包板上接線以連接電腦各部份，接成各種指令。到記憶體（computer memory）問世後，程式終於可以預先存在電腦內部；但寫程式的唯一方法是以二進位方式（用0與1來輸入），其後再演進爲以十六進位或八進位方式輸入。以今日的眼光視之，這雖是非常困難而不方便的方法，但在當時卻勝過接線與撥換開關甚多。

程式設計人員後來終於爲自己寫了一種程式，可自動將一些簡短的電腦指令代語（computer instruction mnemonics）翻譯成二進位碼。這種程式叫做"組合程式"（Assembler），因爲每一代語都是各自被直接組合（assemble）成其二進位碼。之後更進步到可處理符號變數（symbolic variable），以取代直接指明記憶體位址的方法，這使得程式設計更簡化許多。這在當時已是非常了不起的進步。

仍然不夠簡單

許多思想較新的程式設計師還是覺得不滿足。幾年後高階語言（high-level computer language）出現了，這使得程式設計人員可直接專注於問題本身，無需分神斤斤計較於如何教導電腦一步一步做下去。這種用高階語言寫的程式，執行起來當然比用組合語言（assembly language）要慢，但因設計程式所需花費的時間大幅減少，使生產力大幅提高，結果還是非常划算的。

第一種這種高階語言就是大家所熟知，專門用以處理科學與數學方面問題的"FORTRAN"，這名稱來自於"FORmula TRANslation"，即爲"公式翻譯"之意。第二個即是可以處理大量會計及資料處理問題的"COBOL"，其名來自"COmmon Business Oriented Language"，即"通用商用語言"之意。

IBM當年所發展的FORTRAN與COBOL，到目前爲止壽命都已超過30年，但仍能在今日進步神速的電腦市場中歷久不衰，也都能用在一般個人電腦上。但它們也並非完全沒經歷過嚴重的打擊，其中之一即在於其本身發展程式的方法。今日大多數的電腦語言，包括FORTRAN與COBOL在內，都是用所謂編譯器（compiler）的方法：將程式設計者所輸入的高階語言原始程式，轉換成由一串串0與1所組成的機器語言（machine language）碼，經過一個叫做連結編輯（link editing）的中間步驟，再實際去執行。如果程式有錯，必須修正原始程式，再重覆編譯（compile），連結（link），執行（run）等整個過程。

改用解譯

這種用編譯器的方法在處理程式上效率雖是很高，但若將今日最寶貴的電腦資源加入，即程式設計人員所投入的時間時，卻顯得很沒效率。尤其在程式很長，編譯所需時間可觀，或新進程式人員尚做慣學，必須一再修改程式，一再編譯時，編譯器更不是一種效率很高的工具。

爲了改善這個困擾，出現了兩種"解譯語言"（interpreted language），分別是IBM的Kenneth Iverson在1962年發表的APL（A Programming Language），和Dartmouth學院John Kemeny，Thomas Kurtz兩教授在1964年發展的BASIC（Beginner's All-purpose Symbolic Instruction Code）。

與編譯器（Compiler）不同的是，解譯（Interpreter）在檢視每一道原始程式中的指令後，立刻將之轉換成二進位的機器碼，並立即執行，然後才再檢視下一道指令。這樣當然使程式執行起來比整個編譯過的要慢許多；但這也讓程式設計者可以輸入程式後立刻執行，看結果對不對，再修改程式，再執行一遍……，而不必每回修改後都得停下來等編譯和連結完畢。於是解譯語言成爲便捷且吸引人的學習及發展程式的工具；而BASIC與APL也都與FORTRAN，COBOL一樣歷久不衰，也都能在今日的個人電腦上使用。

結構化

再更進一步的趨勢，則是要求

程式的設計必須依照一些約略的概念與規則，也就成了後來熱門的所謂“結構化程式設計”（Structured Programming）。這使程式設計趨向於藝術化。這種程式看起來簡潔、優雅，容易看懂，一道道指令明確地描述出問題與其解決方法；並使程式的設計、發展、維護等工作，比起早期結構化程式等，顯得簡單無比。

這種結構化的概念與規則也許仍有些模糊，無法一一條列，但依舊廣受歡迎（尤其是一般商用程式設計者），公認爲能使程式設計人員更有效地運用其時間。

雖說任何一種語言都可以有它結構化的程式寫法，近十幾年來陸陸續續又出現了幾種專爲便於程式之結構化而設計的程式語言，如歐洲的ALGOL（國內較少人使用），IBM的PL/I，以及更新的如Knuth的PASCAL，貝爾實驗室的C語言等。這些語言幾乎是強迫使用者運用各種程式結構化的工具，如函數、副程式。或其他塊狀程式結構，以及如錄（record），集（set）等使記憶體位址名稱更具有眞正意義的資料結構。

選擇自己所用的語言

時至今日能在個人電腦上使用的程式語言雖然很多，但一個使用者會時常運用的總只是固定的少數幾種。以下挑出比較常用，較適合在個人電腦上發展的幾種語言，做進一步的探討。

BASIC

一般的個人電腦使用者在設計程式時所使用的程式語言，還是以BASIC爲主，尤其在APPLE Ⅱ上，這情形非常明顯。這些使用者不見得就只熟悉BASIC一種語言，而是因爲BASIC的普遍性。以IBM PC爲例，Microsoft發展的BASIC解譯器可放在ROM裡，也可放在磁片上，並且在絕大多數的PC相容電腦作業系統下也都能照常使用；使用起也很方便，不需要什麼繁雜的步驟。

BASIC之所以如此普遍，也因爲它易於學習，又能有效掌握個人電腦的各種資源，尤其是畫面處理等功能。如果使用其他語言，或在傳統的大型電腦，迷你電腦上，我們都得另外添購副程式庫（library routines）或自行設計組合語言程式，才能得到如全螢幕畫面處理或繪圖功能；可是以個人電腦配合BASIC語言，卻是輕而易舉。

BASIC是一種解譯器，執行程式當然不如用編譯器的語言快，但這種交談式（interactive），解譯式的程式環境，正是其創始者Kemeny教授與Kurtz教授的本意。在將BASIC程式寫好並除錯（debug）後，我們可以買個“BASIC編譯器程式”（BASIC Compiler），把程式編譯成機器語言，執行速度就不比用其他語言寫的程式差了。或者，當然我們也可以嘗試著用其他語言設計程式。

8086/8088組合語言

比起任何高階語言，組合語言不論學習或使用都是既艱難又古怪。可是個人電腦使用者中會用到其組合語言的卻也不少。這是因爲除了BASIC以外，只有組合語言可以靈活且有效地運用個人電腦各項資源，其他高階語言都望塵莫及。如果有什麼事情是用組合語言程式辦不到的，那就是這套電腦根本沒法做到的事情。

設計組合語言程式實在是既困難又費時，但程式執行起來速度之快足以令人咋舌，成就感也特別高，因此總會讓使用者覺得值回票價，樂此不疲。這好比千辛萬苦攀上萬尺高峯，那感覺絕非征服三、五百公尺平地小丘所可比擬。

PASCAL

PASCAL語言爲1971年瑞士計算機科學教授Niklaus Wirth所發明，最初設計之構想也跟BASIC一樣著眼於：如何使設計程式變成一件更簡單易學的事情。它與當時其他語言最大的不同在於它強調程式結構化的觀念，並希望以此使電腦語言更容易學習，程式更容易設計。它的語法設計更是強迫使用者必須設計結構良好的程式。

Wirth以“PASCAL”來命名這個語言，是爲了紀念十七世紀法國大數學家Blaise Pascal，而不像BASIC，FORTRAN，COBOL，PL/I等，是將一長串有意義的

詞彙壓縮成的簡稱。PASCAL原本並不常用在個人電腦上，直到幾年前一種專門用於個人電腦的PASCAL編譯器："Turbo Pascal"問世後，普遍性直線上升。

PASCAL問世後風行一時，一直被當做很好的教學用語言（最早我們一直都是以FORTRAN做爲教學用語言）。但它的一些限制所造成的問題，卻大大限制了它在商業方面的應用。就像大多數強調塊狀結構的語言一樣，PASCAL格式嚴謹，它的形式定義（formal definition）不能超出由基本指令所構成的核心（kenel），這種限制，在學術應用上不構成大問題，但在需要較多功能變化的商業應用上卻成大礙。

於是出現了各種擴充版本的編譯器程式，各自添加或擴充了一些功能與指令，但又變成互不相容的大雜燴。這相容性問題使得PASCAL仍然不適於商業用途，因爲若想把一個程式移到另一部擁有不同的PASCAL編譯器的機器上時，勢必要改寫程式，又難又費時，有時甚至辦不到。

後來Borland將他的編譯器逐步應用到PC以外的各型微電腦系統上，因而形成一種標準，日後勢將運用到更多的機型上。

C語言

談到"C"這個名字，有的人可能會以爲是取"Compact"之意，因爲這確實是很經濟、很有效且結構良好的一種語言。事實上，C語言是由B語言蛻變而來，所以命名爲C；而B語言則是貝爾實驗室（Bell Lab.）內部自己發展系統程式時所用的一種語言。

當初Brian W. Kernighan與Dennis M. Ritchie兩人發展C語言，是爲了用做寫DEC PDP-11迷你電腦之UNIX作業系統的系統程式語言。它是一種塊狀結構（block-structured）的語言，將PL/I，ALGOL，PASCAL等其他語言最好的特性都包容進去。後來C與UNIX又都有了爲各種不同機器設計的不同版本，但AT & T與Bell Lab.創立基礎設定標準的努力仍是功不可沒。當然，在IBM PC上也都能使用C與UNIX。

C很快的就流行起來，爲各種大小型電腦上專業應用程式與系統程式發展者所喜用。以C之簡潔、有效，及可攜帶性（portability，在不同機型，不同系統間程式之通用性），C的普遍性勢必與日俱增。

COBOL

根據統計，現今運作中的電腦程式竟有80％以上是用COBOL寫成的（不過在個人電腦上可能只有不到10％），這實在是一個驚人的數字。身爲現存最古老的一種商用程式語言，COBOL之能歷數十年而不衰，不是因爲它功能強或使用簡便，只是因爲已用COBOL寫成的程式數量龐大，勢力雄厚。若將這些程式改用其他語言重寫既不划算，就需要大量的COBOL程式師來維修現有程式系統，而且這些人員在擴展原系統或發展新程式時當然也都會偏愛COBOL，因此COBOL不太可能失寵。

COBOL在當年確實是會計與資料處理方面最好的程式語言。不過，隨著電腦語言設計觀念與結構的演變，許多電腦科學家們期待著能有新的商用程式語言能取代COBOL的地位，也確實有不少人這樣嘗試過，但都失敗了。在商用程式的世界中，COBOL王國著實難以撼動。因此，若您想在個人電腦上寫些會計程式，不妨也嘗試用COBOL看看。

（原载：微电脑时代〔台〕1986 年 5 期 81 — 85 页）

以Benchmark測試法比較前述各種程式語言

Benchmark 是一種測定效能的標準。因為每一種語言各有其長處、短處，沒有任何單一、通用的測試方法可用以判別孰優孰劣（即或是同一語言的不同版本間亦然）；另一方面電腦運算的速度往往以微秒（10^{-6}），微毫秒（10^{-9}秒）為單位，難以比較其快慢，所以必須重複千百次才能分出快慢。因此我們選用了六種簡單的任務，在IBM PC 上用各種語言各種版本重複執行千百萬次，來比較功能強弱與速度快慢。比較結果請見下文。

在您選擇語言時，設計Benchmark 測試程式的難易，應該比設計好後的執行速度還重要；因為這是要花你的時間，而不是要花電腦的時間，你必須花時間去學習語言並設計程式。我們選擇了六種測試方法，分別用前述五種語言設計程式並加以執行，在此先介紹這六種測試方法，程式表列請參見後文。

空迴圈

任何事情如果只用電腦程式去做一次，那還不如不做，因為大多數程式花在迴圈中一連串指令上（迴繞，iteration）的時間就佔了一大半，迴圈本體（body）的執行所花的時間反而不多。這"空迴圈，empty loop" benchmark 測試，就是什麼都不做，只是迴繞10000次，用以測試各語言處理迴繞過程的快慢。

整數加法

除了迴圈之外，一連串整數相加（integer addition）是程式中最常用的功能。單純的兩個整數相加常見於程式迴圈的計數，連串整數相加，則常用以控制層層迴圈間的進入與跳出。總之，這是任何電腦語言都不可輕視的重要功能，後文所設計的第二種 benchmark 程式即用以比較各種語言的整數加法技巧。

浮點運算

浮點運算（floating-point arithmetic）對IBM PC上的電腦語言而言，是一件艱難的工作，因為這不像整數加法那麼簡單，PC的8088 CPU並不直接具有這種功能，任何一個浮點加減乘除運算，都必須經過一長串的步驟才能完成。比起其他五個benchmark 程式，在這個測試程式中，語言設計者的技巧好壞顯得特別重要，直接影響測試結果的優劣。如果您打算做數學、工程、或其他科學、財務方面的應用，這個測試的結果在您選擇語言時自是特別重要。

字串連接

字元處理雖然在文字與文書方面的應用特別重要，但它在一般通用型電腦（general-purpose computer）的記憶體管理（memory management）上也非常有用。雖然其他字串運算也能用以測定語言的字串處理能力，但還是以字串串接（character string concatenation）最常用。這測試的結果非常有趣，也許會讓您對各種語言如何把兩串句子串接起來覺得好奇。

查表

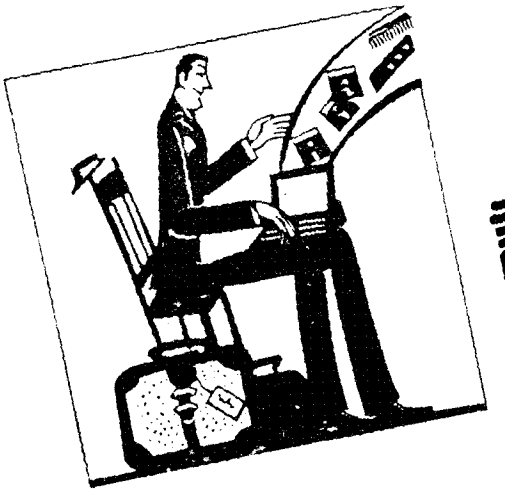
把一連串數字或資料存在一個表內，要用時再去查表（table look-up），這也是電腦程式中非常常用的一種功能。各種語言處理查表工作所用的方法及組合各不相同，差異甚大，所測出的benchmark速度也快慢懸殊。因為各種語言比較在意數值的型態，不論數值的使用或不同型態數值的轉換規則都比較多，有些語言在處理數值時則比較寬鬆但快速。

檔案存取

磁碟上資料的讀寫，對任何個人電腦上的應用程式都非常重要，但各種語言執行檔案存取（file access）的快慢似乎並不那麼重要，因為這一定得用到DOS的功能呼叫（function call）來處理煩人的磁碟機控制事宜。不過使用132 byte 的錄（record）來測試，對DOS與語言却有些不公平，因為這並非最佳或標準的大小，會迫使DOS或語言多做一些額外的工作。事實上，以某種語言寫一個程式來處理檔案的快慢，可能要比該語言真正執行該程式的快慢還重要。

（原載：微电脑时代〔台〕1986 年 5 期 84 — 85 页）

趙文欽



認識BASIC

任何人都能盡情使用這種流行最廣的電腦語言，
天下沒有它不能做的事，也沒有人學不會。

傳統的電腦程式觀念中，BASIC 似乎只是一套不入流的玩意兒，只懂得 BASIC 一種程式語言的人，也不怎麼夠承認具有程式設計師（programmable）的資格。即或在今日的許多電腦玩家眼中，如果你口裏不掛著C或PASCAL等時髦名詞，就不算懂得電腦語言，BASIC 只是一個卑微的小丫頭。

令人不屑一顧

事實上這是一種非常不公平的觀念。當初個人電腦引進國內時，唯一能用的語言就是BASIC；那種變數名稱只認前兩字元，所有數值都是實數的 APPLESOFT，因此大多數人都是由 BASIC 入門，把 BASIC 用得出神入化的也大有人在。當然，你也可以拿著 APPLE II 從組合語言或機器語言學起，也確實有人這麼做，這種人後來不是發瘋，就是發明 VisiCalc 之類的軟體，卻都不能在幾天之內寫就一個有實際用途的完整程式。

的確，就傳統程式設計的觀點來看，BASIC的確算不得一種好語言；但直到現今，BASIC 仍是唯一直接在家裡的桌子上就能操作，並完全發揮電腦神奇功能的一種高階程式語言。不過這一方面也歸功於：每部個人電腦都有其專爲一般使用者設計，並將自身功能發揮到極致的BASIC。

當然，這些特製的 BASIC 確實也造成通用性（portability）的問題。各機種特長各異，爲發揮特長所增添的指令也各不相同，使得在甲機型上用的程式移到乙機型上時可能就動彈不得了；也許這程式用到某些甲機型自己增添的指令，乙機型不認得，也許這程式用到某些甲機型特有的硬體功能，乙機型根本辦不到。不過話又說回來，也只有這類特殊化了的改裝型BASIC，才能發揮各特定機種的所有特色，通用性良好，適用於所有機種的標準語言，就絕對無法顧及各機型各自的硬體特長。如果以通用性爲設計語言的第一考慮，何不乾脆整個工業界只生產一種機型，共同使用一種結構漂亮的標準語言？但這就不會是BASIC了。

簡言之，BASIC 最大的秘訣在於，它是能發揮其各該機種所有低階功能的一種高階語言。

四種BASIC版本

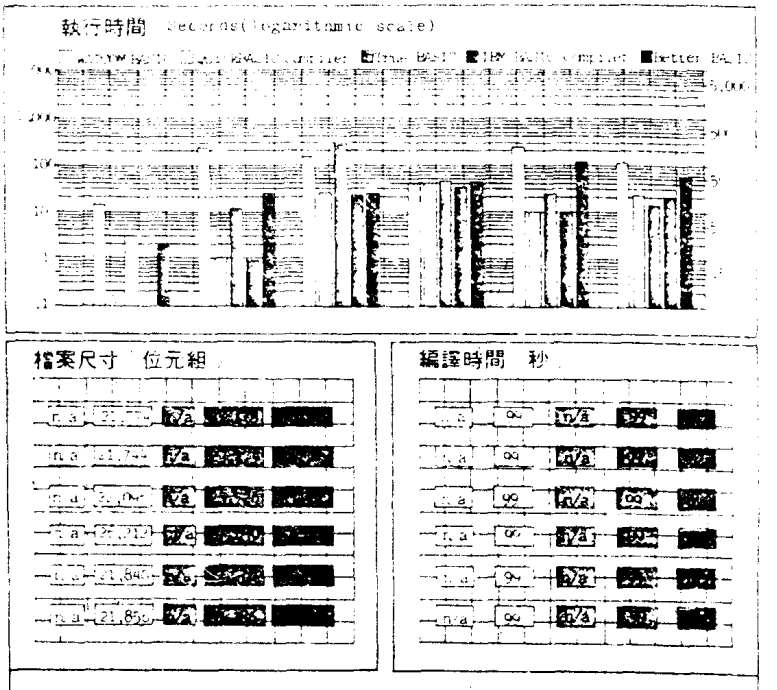
有些新版本的改良式 BASIC，不但保留了 BASIC 的交談式（interactive）特色，且速度比原來的解譯器（Interpreter）版本還快。現今通用的主要有四種不同的 BASIC：傳統解譯器版本，解

譯器版本，與兩種混合型版本。其中一種叫“ BetterBASIC ”的混合型版本，就是所謂的“交談式編譯器(Interactive Compiler)”。

交談式編譯器用起來就如同是一種解譯式語言，你可以直接在內藏編校器（ internal editor ）下輸入原始程式後，立刻打“RUN”指令執行程式，並幾乎是立刻就得到反應，速度比解譯器快過許多。這是因爲它將原始程式編譯成機器碼存入記憶體，而不是一次只解譯一行，也不是以傳統編譯器方式經由 DOS 呼叫編譯器來編譯。當然，它與傳統編譯器一樣，可以將編譯成的機器碼以 .EXE與 .COM 的檔案儲存起來。

另一種混合型BASIC叫“ True BASIC”，用起來也與傳統解譯器完全相同，但可“編譯”原始程式爲機器碼，而且這原始程式可以移到任何提供 True BASIC 的機型上使用。

許多新的解譯器或編譯器都已稱得上是完全結構化的語言，可以完全不用到 GOTO , GOSUB就寫得出很複雜的程式。這些漂亮的BASIC都有多行函數（multiline function），變數可分通用性（global）或區域性（local），也有如 DO～UNTIL , CASE～ENDCASE等漂亮的結構；甚至可由 SoftCraft , Computer Control Systems等軟體公司提供外加之二元樹結構（ b-tree ），IS-AM等功能，或由 Software Bot-



圖說：

WATCOM BASIC , Microsoft QuickBASIC Compiler , True BASIC , IBM BASIC Compiler , 與 BetterBASIC等五種 BASIC 的 benchmark 測試結果：

- 所有測試都是在一部配有兩個軟式磁碟機的 IBM PC上執行。
- 所有檔案大小都是以位元組爲單位，所有時間都是以秒做單位。
- True BASIC 與 WATCOM BASIC 都是解譯器，所以不產生目的碼（ object code），也無所謂編譯時間（ compile time ）。
- BetterBASIC在其解譯器內進行翻譯，所以翻譯時間應是“瞬間”。
- Microsoft QuickBASIC 與 IBM BASIC對每一測試程式的編譯時間都是 2 秒，圖中所示的 99 秒乃包含標準連結時間（ link time ）97 秒在內。
- Better BASIC , Microsoft QuickBASIC , 與 IBM BASIC ，除了一般編譯之外，也能編譯成需要與另外的執行模組（ run-time module ）配合才能執行的執行目的碼（ run-time object code ）；執行目的碼及執行模組的檔案大小皆未列入本圖表中。