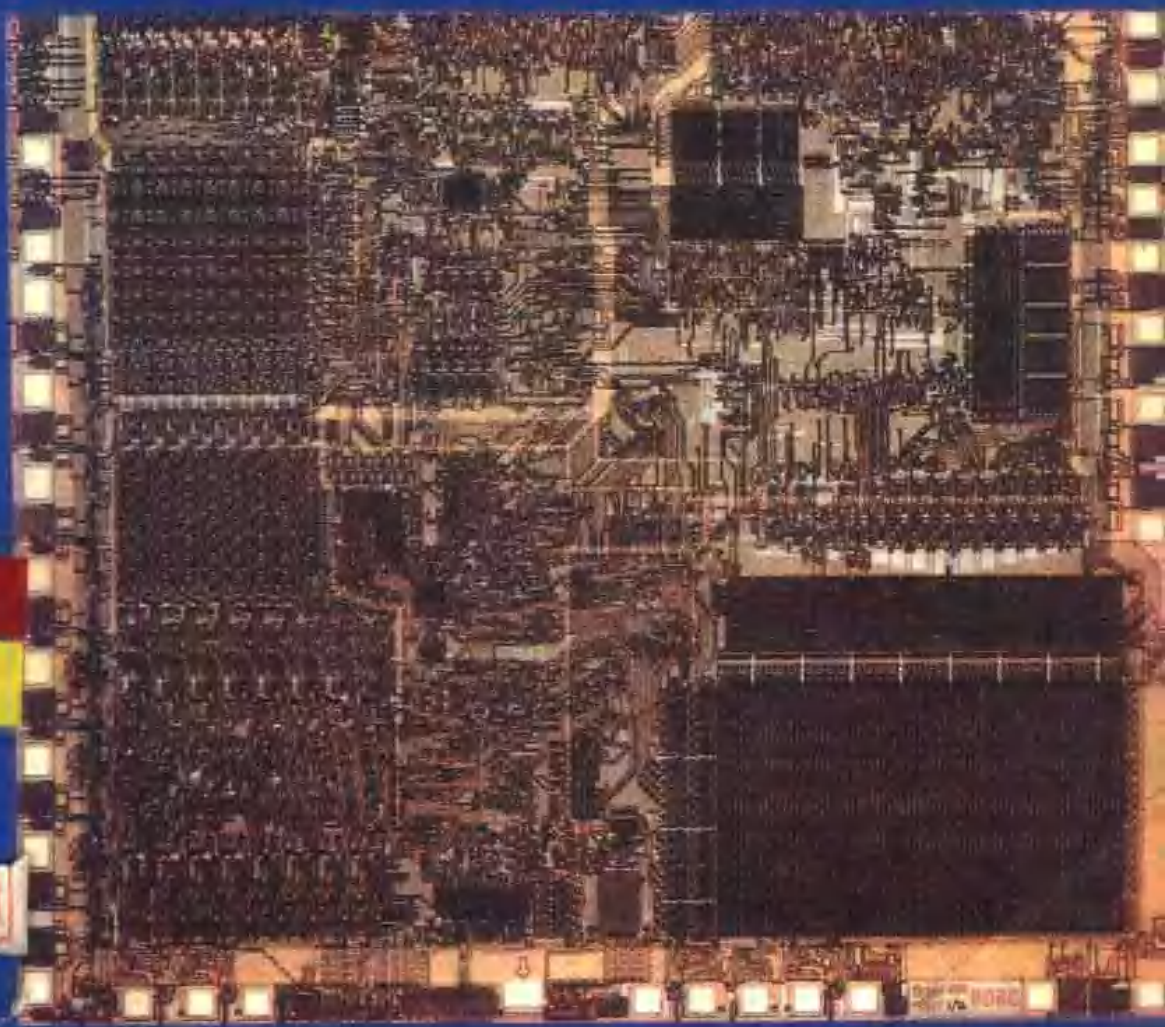
IBM PC

組合語言程式設計

魏易休・蔡毓琛　譯



IBM PC

組合語言程式設計

魏易休・蔡毓琛　譯

儒林圖書公司 印行

IBM PC 組合語言程式設計

原著發行日期：1983 年
原著書名：IBM PC ASSEMBLY LANGUAGE
原 著 者：LEO J. SCANLON
譯　　者：魏易休・蔡毓琛
發 行 人：楊　鏡　秋
出 版 者：儒　林　圖　書　有　限　公　司
地　　址：台北市重慶南路一段 111 號
電　　話：3812302　3110883　3140111
郵政劃撥：0106792-1 號
吉豐印刷廠有限公司承印
板橋市三民路二段正隆巷 46 弄 7 號
行政院新聞局局版台業字第 1492 號

1984 年 11 月 5 日
定價新台幣 [redacted] 元正
$200

前　言

這本書能夠使你掌握 IBM 個人用電腦的全部功能，而組合語言的速度和效率正是這個功能的關鍵--若使用如 BASIC 之類的高階語言便會失去這種功能的效用。

如果你正進行着無需快速度動作的程式，那麼使用 BASIC 並沒有什麼不對。但你若是正在寫一個需同時進行許多運算與執行的程式，譬如在一個遊戲的程式中，欲移動螢幕中好幾個物體，你便會發現 BASIC 無法完成這項工作。這些物體可能會神秘地消失或以斷斷續續的方式在移動。很明顯地，在這樣的工作要求下，BASIC--定無法勝任。

BASIC 和其他大部份的高階語言都不適合與即時系統之週邊設備互相通訊。也許你的記憶體空間本來就不多，而使用這些語言寫的程式會佔據了許多記憶體空間。因爲這種種的問題，此時便可以考慮使用組合語言來寫你的程式了。

即使現在你認爲組合語言是最佳的選擇，你仍可能懷疑是否你有足夠的背景去學習組合語言程式設計？如果你曾經使用過任一種語言來設計程式，那麼你便能辦得到。譬如你已了解 BASIC 或其他高階語言便是很好的例子。但若你曾經以組合語言來設計過程式就更好了。爲了前述只曾用過高階語言的使用者著想，這本書共有兩個出發點。

如果你從未以組合語言設計過程式便應從第 0 章開始，它能夠告訴你關於二進制和十六進制數字系統的重要觀念。若你已了解這些術語的意義並且明白如何使用它們則可以直接由第一章開始。

這本書的內容

這本書共有八章。在第一章中，我們介紹 8088 微處理機—即個人用電腦的核心部份—並且討論它在一個系統中扮演的角色。

第二章一般性地討論組合程式（assembler）的概念，再描述兩種基本的組合程式，即 IBM Small Assembler 及 MACRO Assembler。（雖然這本書只使用兩種組合程式作實例，但其一般原理適用於你所擁有之其他任何 IBM 組合程式）。第二章同時也告訴你如何將一個程式輸入電腦及如何去執行它。並且以一個簡單的程式做爲例子。

第三章敍述 8088 微處理機的指令集，即用來與你的個人用電腦溝通的指令，這本書係以同類功能的分組來介紹 8088 之指令，而非依據字母排列順序。也就是說，我們將加法與減法分爲一組，乘法與除法分爲一組等等。透過這種方式你不僅能夠了解這些指令的用途，同時也能認識這些指令如何互相配合應用。

在第四章你會學到如何組合數個指令來執行較高級的數學運算，這些在微處理機的指令中並不直接提供出來。第五章包含了對串列及表的運算。

IBM個人用電腦有一內建作業系統（built-in operating system）稱爲 BIOS（即 BASIC I/O System 之簡稱），用來管理系統中的設備。所有能使微處理機與鍵盤、螢幕顯示器及其他週邊設備溝通的必需條件都由它執行。因爲這樣做，BIOS 就好比個人用電腦的“行政管理主管”。BIOS 包含許多有用的特性能夠使你在應用時，節省許多執行程式的時間，第六章說明你應如何去使用它們。

在第七和第八章你會學到如何撰寫程式使能在螢幕顯示器上描繪出簡單的圖形及透過電腦的內建喇叭而產生聲音。

這本書爲著讀者的便利提供了四個附錄。附錄 A 中之表格能

幫助你將十進制數與十六進制數互相轉換。附錄B概述ASCII字元。附錄C及附錄D依據字母排列順序介紹8088微處理機的指令，並且說明8088執行每一個指令所需要的時間，每一個指令在記憶體中所佔位元組（ byte ）個數及它對於旗號的影響。

習　題

大部份章節後面均附有一些問題和有關程式的習題。這些習題部份用以測驗你對該章內容的了解程度，部份用以擴展你對更進一步和相關領域知識的認識。

使用這本書你所需要的設備

欲使用此書，你需要一部附有顯示器，大容量儲存裝置（即磁碟片或卡帶）之IBM 個人用電腦。同時也需要含有8088組合程式之軟體套裝程式。

這本書描述兩種IBM推出之組合程式即 Small Assembler 和MACRO Assembler ，但其一般原則都適用於其他與IBM相容之組合程式。在此書中，個人用電腦之所有組合程式都是在軟式磁碟機上執行的。一部磁碟機雖然就夠用，但兩部磁碟機却能使工作簡使許多。當然，由於使用了磁碟片你也需要磁碟作業系統（DOS ）軟體套裝程式。

補充參考書籍

這本書主要用來補充IBM 個人用電腦使用手册之不足，所以你可能不再需要其他參考書籍。然而每一個認眞的個人用電腦使用者都應擁有一本IBM技術參考手册（ IBM Technical Reference Manual），因它是無價的技術資訊的來源。在其他

有關資料中，這本技術參考手册尚包含了個人用電腦內建作業系統（即BASIC I/O System ）完全且充份的註釋及程式列表。

其次，如果你正在爲個人用電腦設計附加硬體，或希望能獲得該系統中關於晶片之完整細節，你可能需要一些下面參考資料:

- *iAPX 86,88 User's Manual* and Numeric Supplement
- *iAPX 88 Book*
- *iAPX 88/10 Data Sheet*

目　　錄

0

計算機數字系統簡介

除非你從其他的星球來訪問地球，否則你大概一生都使用十進制數字來計算事物。十進制系統是一種以 10 爲底的數字系統，包含 10 個阿拉伯數字--0 到 9 。

人類相當習慣於使用十進位數字系統（可能因爲我們天生便擁有十隻手指及十隻脚趾），但對電腦却不適用。實際上，電腦被設計爲以 2 爲底的數字系統，即系統中僅包含兩個數字—0 和 1 。因此爲了要和電腦互相交通，你必須熟悉於二進制數字系統。除了二進制數字系統之外，組合語言程式設計師們也使用其他數字系統——主要爲以 16 爲底（16進制數字系統）——來表示二進制數字的組合，所以你必須要同時熟悉這些數字系統。

本章對於在電腦數字系統毫無概念的讀者們是重要的一章。也就是爲什麼我們稱此章爲第 0 章的原因。如果你已經了解二進制和十六進制數字系統，那麼就跳過這章，直接進入第一章。

0.1 二進制數字系統

在一台電腦中，所有的程式指令和資料都被儲存在電腦的記憶體中。記憶體是由大群的電子零件組合而成。這些零件就像燈

泡的開關，他們只有兩種可能的狀態：“開”和“關”。經由這兩種狀態，記憶體零件的組合才能表示各種大小不同的數字。至於如何表示，請繼續讀下去吧！

記憶體零件的開和關兩種狀態對應到二進制數字系統。因爲只有兩個數字，1代表開及0代表關，因此二進制數字系統是個以2爲基底的數字系統。（這恰好與擁有0到9十個數字的十進制數字系統成一對比，因爲它是以10爲底的數字系統）。

這個像關關一般的記憶體零件稱爲“位元”（bit），即二進制數字之簡稱。一般來說，當一個位元之值爲1時，代表“開”，而其值爲0時，便代表“關”。因此二進制數字系統便顯得範圍太狹窄，但如果考慮十進制數字系統也不過只有0至9十個數字，就會覺得二進制也沒什麼不好。如同你組合十進制數字以構成大於9的數字一樣，你也可組合二進位數字來構成大於1的數字。

如你所知，爲了表示出大於9的十進位數字便需要其他的數字，即一個十位數數字。同樣地，要表示出大於99的十進位數字便需要一個百位數數字，其他位數依此類推。而你所增加的每一個十進制數字比起緊靠其右邊的位數有着10倍於該位數的構重（weight）。

例如，你可以將十進制數字324表示爲：（3×100）+（2×10）+（4×1）成（3×10^2）+（2×10^1）+（4×10^0）。所以由數學上的觀點來看，每一個十進制位數都是位於其右側位置之位數的10倍。

同樣的原理亦適用於二進制數字系統。在此系統中，每一個二進制位數都是位於其右側位元位置之位數的2倍。位於最右邊的位元，其權重爲2^0（即十進位制數字1），緊接於其左的位元則權重爲2^1（即十進制數字2），以此類推。例如，二進制數字101，其轉換爲十進制後之值爲5，因爲$101_2=(1\times2^2)+(0\times2^1)+(1\times2^0)=(1\times4)+(0\times2)+(1\times1)=5_{10}$。

現在你了解二進制數字是如何被構成的嗎？爲了求得任一指定位元位置的值，你必須將右一位元位置之構重乘以 2 倍。如此一來，前爲八個二進位制之構重便分別爲 1 ，2 ，4 ，8 ，16 ，32 ，64 ，128 。如圖 0-1 所示。

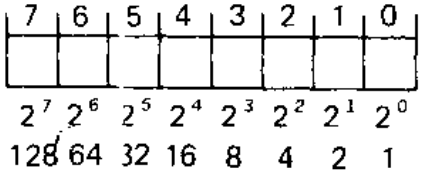


圖 0-1

欲將一個十進制的數值轉換成二進制，必須做一連串簡單的減法。而每一次的減法運算便可求出一個二進位制數字（或位元）的值。

一開始，先將十進制的數減去儘可能最大的二進制權重並且在該位元位置填入一個“ 1 ”，然後將前面運算的結果再減去可能的次大二進制構重並且在下一位元位置填入“ 1 ”。如此繼續運算至減法運算結果爲 0 爲止。減法過程中，若十進制的數不夠被減去某個位元的權重時，便將“ 0 ”填入該位元位置內。

例如，將十進制的數 50 轉換成二進制的數：

```
  50
- 32      位元位置 5 = “ 1 ”
  18
- 16      位元位置 4 = “ 1 ”
   2
-  2      位元位置 1 = “ 1 ”
   0
```

最後再將“ 0 ”填入其他位元位置內（即位元位置 3，2，0 ），即得到最終的結果— 110010 。

爲了證明等於十進位數 50 的二進位數確實是 110010 ，只

需將該二進制的數中各位數爲“ 1 ”的十進位權重相加即可得到。（譯註：讀者也可用連續除 2 的方法，將十進位數轉換爲二進位數）

```
  32    位元 5 的權重
  16    位元 4 的權重
+  2    位元 1 的權重
  50
```

0.1.1 八個位元組成一個位元組

Apple Ⅱ和Apple Ⅲ，Commodore PET/CBM，Radio Shack TRS-80 及其他許多普遍的微電腦都是以八位元的微處理機設計的。八位元微處理機之所以被如此命名是因爲它們一次操作僅能處理八位元的資訊。因此爲了能處理超過八位元的資訊便需二或多次的處理。

在電腦的術語中，一個以八位元爲單位的資訊被稱爲位元組（ byte ），一個位元組因爲包含八個位元，因此能夠表示出十進制數 0（即二進制數 00000000 ）到 255（即二進制數 11111111 ）。

因爲位元組是電腦處理的基本單位，所以微電腦系統也就是以其記憶體中所包括的位元組個數而進行設計。另外，微電腦製造者們通常製造記憶體都以 1024 個位元組之容量爲一個區段（ block ）。這個特殊的數值反應出此區段具有 2^{10} 個位元組。

1024 這個值有一個標準縮寫，即用字母 K 表示。如在電腦廣告中提到它擁有 48K容量之記憶體就是告訴你這項產品具有之儲存能力爲“ 48×1024 ”位元組。

0.1.2 二進制數之加法運算

欲將二進制數相加就如同十進制數相加的道理一樣：即將此

行的進位數傳送至左一行數字中。例如，如果你將十進位數 7 和 9 相加，你必須把進位數“ 1 ”送至十位數，才能計算出正確的結果（ 16 ）。同樣地，如果你將二進位數 1 和 1 相加，便必須把進位數“ 1 ”送至 2' 位數所在，而計算出正確答案（ 10 ）。

如果你欲相加多位元的數字，那麼這種加法運算便會變得較爲複雜，因爲尚包含其他位數的進位問題。如下列式子

```
  1011
+   11
------
  1110
```

這個運算牽涉到兩個進位數。在最右邊一行中（ 1 + 1 ）的運算產生之結果爲 0 ，同時產生一個進位數至第二行。由於這個進位數，使第二行中的加法（ 1 + 1 + 1 ），產生之結果爲 1 ，並且進位至第三行。

一般的規則可用下表來表示：

輸入			結果	
運算元 # 1	運算元 # 2	進位數	和	進位數
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

0.1.3 有正負號的數字

直到目前爲止，我們一直都在討論無正負號數的二進制表示法。對一個無正負號數來說，根據位元所在的位置，每一個位元都有其一定的權值（如我們前面所討論）。其中最低有效的位元

之權重爲 2^0，其他有效的位元之權重皆爲其右一位元權重的兩倍。。因此，如果一位元組中所有位元均爲 0 則此位元組之值即爲 0；如果所有位元皆爲 1，則此位元組之值即爲 255。

然而在許多的運算當中，你一定會遭遇到正或負的數值，也就是有正負號的數字（signed number）。當位元組中包含了一個有正負號的數字時，便只有後面七個較低效之位元用來表示這個資料，而最大有效位元（即位元位置 7）便用來表示數字的正負。如果這個數字爲正或 0 則第七位元位置即爲 0，如果爲負則此位元爲 1。圖 0-2 顯示出無正負號數字和有正負號數字的排列情形。

一個包含無正負號之位元組可以表示出之數值範圍爲 0（二進制數 00000000）至 +127（二進制數 01111111）。而包含負數之位元組可以表示出之數值範圍爲 -1（二進位數 11111111）至 -128（二進位數 10000000）。

2的補數表示法

爲什麼 -1 在二進位制中被表示成 11111111 而非 10000000 呢？這個答案就是“負數”是以二的補數形式表示。電腦科學家們發明了 2 的補數形式來除去“0”以兩種不同方式（即全部位元均爲 0（“正 0”）和記號位元位置爲 1 其餘位元均爲 0（“負 0”））被表示的問題。

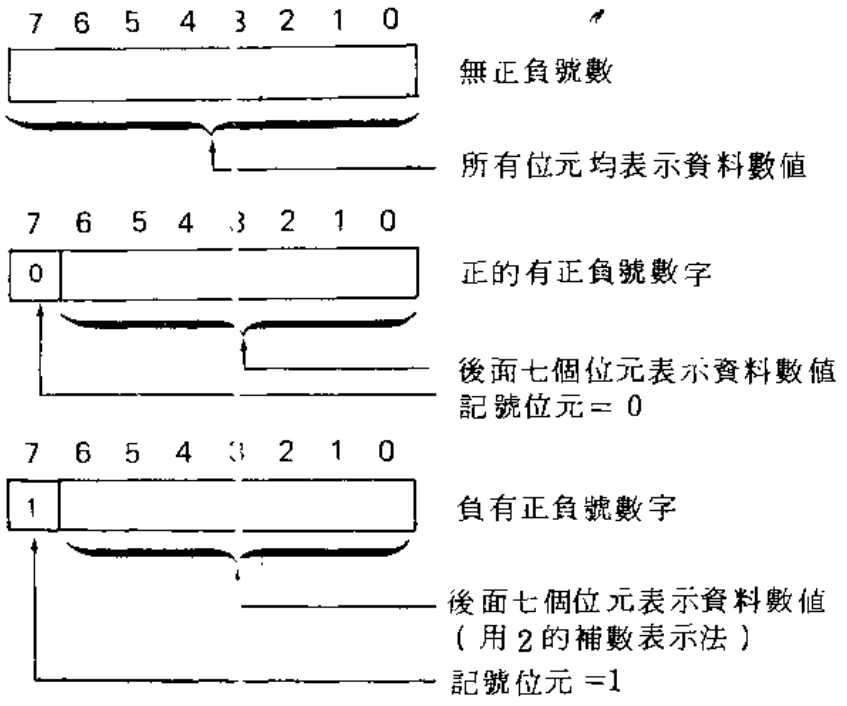


圖 0-2 有正負號數字和無正負號數字表示法

欲求得一個負數的二進制表示（也就是求其2的補數形式）必須取該數之正的二進制表示，然後轉換每一位元——將1轉爲0，0轉爲1——最後再加上1於上述結果中。下面的例子表示出求得−32的二進位制表示的步驟：

```
  00100000    -32
  11011111    反轉換每一位元
+        1    加1
  11100000    2的補數
```

當然也可以用同樣的過程（轉換全部位元並加1）算出一個負數的絕對值的表示。

幸運地是大部份組合語言程式都允許以數字的十進制形態輸入（有正負號或無正負號），並且做好適當的轉換。然而有時必